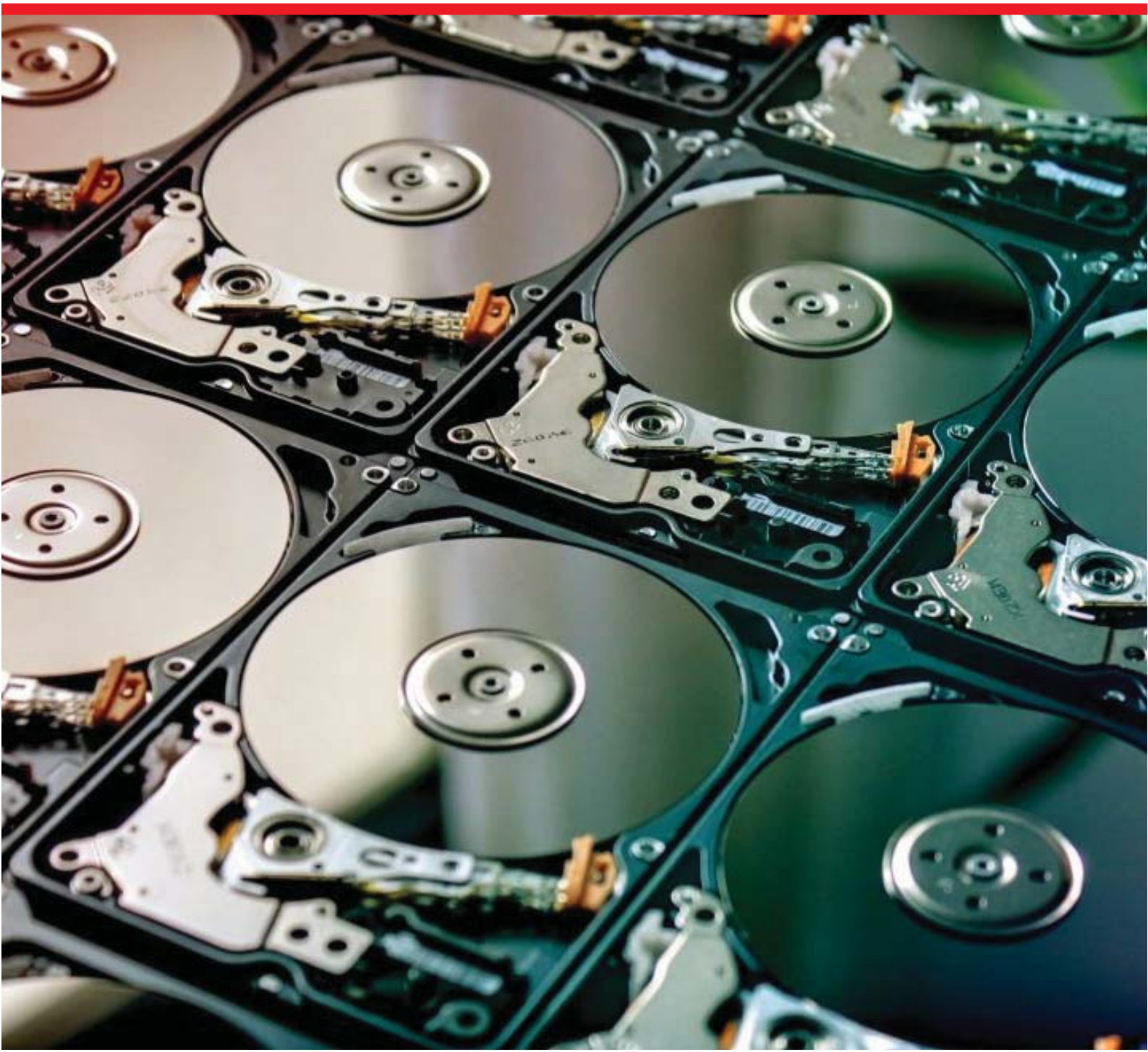
Computer Data Storage and Data Storage Device

Jimmie Frazier



COMPUTER DATA STORAGE AND DATA STORAGE DEVICE

COMPUTER DATA STORAGE AND DATA STORAGE DEVICE

Jimmie Frazier

BIBLIOTEX
Digital Library

Computer Data Storage and Data Storage Device
by Jimmie Frazier

www.bibliotex.com

Ebook ISBN: 9781984664310

Published by:
Bibliotex
Canada
Website: www.bibliotex.com

Contents

1

Computer Data Storage

Computer data storage, often called storage or memory, refers to computer components and recording media that retain digital data used for computing for some interval of time. Computer data storage provides one of the core functions of the modern computer, that of information retention. It is one of the fundamental components of all modern computers, and coupled with a central processing unit (CPU, a processor), implements the basic computer model used since the 1940s.

In contemporary usage, *memory* usually refers to a form of semiconductor storage known as random-access memory, typically DRAM (Dynamic-RAM) but *memory* can refer to other forms of fast but temporary storage. Similarly, *storage* today more commonly refers to storage devices and their media not directly accessible by the CPU (secondary or tertiary storage) — typically hard disk drives, optical disc

drives, and other devices slower than RAM but more permanent. Historically, *memory* has been called *main memory*, *real storage* or *internal memory* while storage devices have been referred to as *secondary storage*, *external memory* or *auxiliary/peripheral storage*.

The contemporary distinctions are helpful, because they are also fundamental to the architecture of computers in general. The distinctions also reflect an important and significant technical difference between memory and mass storage devices, which has been blurred by the historical usage of the term *storage*. Nevertheless, this article uses the traditional nomenclature. Many different forms of storage, based on various natural phenomena, have been invented. So far, no practical universal storage medium exists, and all forms of storage have some drawbacks. Therefore a computer system usually contains several kinds of storage, each with an individual purpose. A digital computer represents data using the binary numeral system. Text, numbers, pictures, audio, and nearly any other form of information can be converted into a string of bits, or binary digits, each of which has a value of 1 or 0. The most common unit of storage is the byte, equal to 8 bits. A piece of information can be handled by any computer whose storage space is large enough to accommodate *the binary representation of the piece of information*, or simply data. For example, using eight million bits, or about one megabyte, a typical computer could store a short novel. Traditionally the most important part of every computer is the central processing unit (CPU, or simply a processor), because it actually operates on data, performs any calculations, and

controls all the other components. Without a significant amount of memory, a computer would merely be able to perform fixed operations and immediately output the result. It would have to be reconfigured to change its behaviour. This is acceptable for devices such as desk calculators or simple digital signal processors. Von Neumann machines differ in that they have a memory in which they store their operating instructions and data. Such computers are more versatile in that they do not need to have their hardware reconfigured for each new programme, but can simply be reprogrammed with new in-memory instructions; they also tend to be simpler to design, in that a relatively simple processor may keep state between successive computations to build up complex procedural results. Most modern computers are von Neumann machines.

In practice, almost all computers use a variety of memory types, organized in a storage hierarchy around the CPU, as a trade-off between performance and cost. Generally, the lower a storage is in the hierarchy, the lesser its bandwidth and the greater its access latency is from the CPU. This traditional division of storage to primary, secondary, tertiary and off-line storage is also guided by cost per bit.

HIERARCHY OF STORAGE

PRIMARY STORAGE

Primary storage (or main memory or internal memory), often referred to simply as memory, is the only one directly accessible to the CPU. The CPU continuously reads instructions stored there and executes them as required.

Any data actively operated on is also stored there in uniform manner. Historically, early computers used delay lines, Williams tubes, or rotating magnetic drums as primary storage.

By 1954, those unreliable methods were mostly replaced by magnetic core memory. Core memory remained dominant until the 1970s, when advances in integrated circuit technology allowed semiconductor memory to become economically competitive. This led to modern random-access memory (RAM). It is small-sized, light, but quite expensive at the same time. (The particular types of RAM used for primary storage are also volatile, i.e. they lose the information when not powered). There are two more sub-layers of the primary storage, besides main large-capacity RAM:

- Processor registers are located inside the processor. Each register typically holds a word of data (often 32 or 64 bits). CPU instructions instruct the arithmetic and logic unit to perform various calculations or other operations on this data (or with the help of it). Registers are the fastest of all forms of computer data storage.
- Processor cache is an intermediate stage between ultra-fast registers and much slower main memory. It's introduced solely to increase performance of the computer. Most actively used information in the main memory is just duplicated in the cache memory, which is faster, but of much lesser capacity. On the other hand it is much slower, but much larger than processor registers. Multi-level hierarchical cache setup is also commonly used—*primary cache* being

smallest, fastest and located inside the processor; *secondary cache* being somewhat larger and slower.

Main memory is directly or indirectly connected to the central processing unit via a *memory bus*. It is actually two buses: an address bus and a data bus. The CPU firstly sends a number through an address bus, a number called memory address, that indicates the desired location of data. Then it reads or writes the data itself using the data bus. Additionally, a memory management unit (MMU) is a small device between CPU and RAM recalculating the actual memory address, for example to provide an abstraction of virtual memory or other tasks. As the RAM types used for primary storage are volatile (cleared at start up), a computer containing only such storage would not have a source to read instructions from, in order to start the computer. Hence, non-volatile primary storage containing a small startup programme (BIOS) is used to bootstrap the computer, that is, to read a larger programme from non-volatile *secondary* storage to RAM and start to execute it. A non-volatile technology used for this purpose is called ROM, for read-only memory (the terminology may be somewhat confusing as most ROM types are also capable of *random access*).

Many types of "ROM" are not literally *read only*, as updates are possible; however it is slow and memory must be erased in large portions before it can be re-written. Some embedded systems run programmes directly from ROM (or similar), because such programmes are rarely changed. Standard computers do not store non-rudimentary programmes in ROM, rather use large capacities of secondary

storage, which is non-volatile as well, and not as costly. Recently, *primary storage* and *secondary storage* in some uses refer to what was historically called, respectively, *secondary storage* and *tertiary storage*.

SECONDARY STORAGE

Secondary storage (also known as external memory or auxiliary storage), differs from primary storage in that it is not directly accessible by the CPU. The computer usually uses its input/output channels to access secondary storage and transfers the desired data using intermediate area in primary storage. Secondary storage does not lose the data when the device is powered down—it is non-volatile. Per unit, it is typically also two orders of magnitude less expensive than primary storage. Consequently, modern computer systems typically have two orders of magnitude more secondary storage than primary storage and data is kept for a longer time there.

In modern computers, hard disk drives are usually used as secondary storage. The time taken to access a given byte of information stored on a hard disk is typically a few thousandths of a second, or milliseconds. By contrast, the time taken to access a given byte of information stored in random access memory is measured in billionths of a second, or nanoseconds. This illustrates the significant access-time difference which distinguishes solid-state memory from rotating magnetic storage devices: hard disks are typically about a million times slower than memory. Rotating optical storage devices, such as CD and DVD drives, have even longer access times. With disk drives, once the disk read/

write head reaches the proper placement and the data of interest rotates under it, subsequent data on the track are very fast to access. As a result, in order to hide the initial seek time and rotational latency, data is transferred to and from disks in large contiguous blocks.

When data reside on disk, block access to hide latency offers a ray of hope in designing efficient external memory algorithms. Sequential or block access on disks is orders of magnitude faster than random access, and many sophisticated paradigms have been developed to design efficient algorithms based upon sequential and block access. Another way to reduce the I/O bottleneck is to use multiple disks in parallel in order to increase the bandwidth between primary and secondary memory. Some other examples of secondary storage technologies are: flash memory (e.g. USB flash drives or keys), floppy disks, magnetic tape, paper tape, punched cards, standalone RAM disks, and Iomega Zip drives. The secondary storage is often formatted according to a file system format, which provides the abstraction necessary to organize data into files and directories, providing also additional information (called metadata) describing the owner of a certain file, the access time, the access permissions, and other information. Most computer operating systems use the concept of virtual memory, allowing utilization of more primary storage capacity than is physically available in the system. As the primary memory fills up, the system moves the least-used chunks (*pages*) to secondary storage devices (to a swap file or page file), retrieving them later when they are needed. As more of these retrievals from slower secondary storage are necessary, the more the overall system performance is degraded.

TERTIARY STORAGE

Tertiary storage or tertiary memory, provides a third level of storage. Typically it involves a robotic mechanism which will *mount* (insert) and *dismount* removable mass storage media into a storage device according to the system's demands; this data is often copied to secondary storage before use. It is primarily used for archival of rarely accessed information since it is much slower than secondary storage (e.g. 5–60 seconds vs. 1-10 milliseconds). This is primarily useful for extraordinarily large data stores, accessed without human operators. Typical examples include tape libraries and optical jukeboxes. When a computer needs to read information from the tertiary storage, it will first consult a catalog database to determine which tape or disc contains the information. Next, the computer will instruct a robotic arm to fetch the medium and place it in a drive. When the computer has finished reading the information, the robotic arm will return the medium to its place in the library.

OFF-LINE STORAGE

Off-line storage is a computer data storage on a medium or a device that is not under the control of a processing unit. The medium is recorded, usually in a secondary or tertiary storage device, and then physically removed or disconnected. It must be inserted or connected by a human operator before a computer can access it again. Unlike tertiary storage, it cannot be accessed without human interaction. Off-line storage is used to transfer information, since the detached medium can be easily physically transported. Additionally, in case a disaster, for example a fire, destroys the original

data, a medium in a remote location will probably be unaffected, enabling disaster recovery. Off-line storage increases general information security, since it is physically inaccessible from a computer, and data confidentiality or integrity cannot be affected by computer-based attack techniques. Also, if the information stored for archival purposes is accessed seldom or never, off-line storage is less expensive than tertiary storage. In modern personal computers, most secondary and tertiary storage media are also used for off-line storage. Optical discs and flash memory devices are most popular, and to much lesser extent removable hard disk drives. In enterprise uses, magnetic tape is predominant. Older examples are floppy disks, Zip disks, or punched cards.

CHARACTERISTICS OF STORAGE

Storage technologies at all levels of the storage hierarchy can be differentiated by evaluating certain core characteristics as well as measuring characteristics specific to a particular implementation. These core characteristics are volatility, mutability, accessibility, and addressibility. For any particular implementation of any storage technology, the characteristics worth measuring are capacity and performance.

VOLATILITY

NON-VOLATILE MEMORY

Will retain the stored information even if it is not constantly supplied with electric power. It is suitable for long-term storage of information.

VOLATILE MEMORY

Requires constant power to maintain the stored information. The fastest memory technologies of today are volatile ones (not a universal rule). Since primary storage is required to be very fast, it predominantly uses volatile memory.

DIFFERENTIATION

DYNAMIC RANDOM ACCESS MEMORY

A form of volatile memory which also requires the stored information to be periodically re-read and re-written, or refreshed, otherwise it would vanish.

STATIC MEMORY

A form of volatile memory similar to DRAM with the exception that it never needs to be refreshed as long as power is applied. (It loses its content if power is removed).

MUTABILITY

READ/WRITE STORAGE OR MUTABLE STORAGE

Allows information to be overwritten at any time. A computer without some amount of read/write storage for primary storage purposes would be useless for many tasks. Modern computers typically use read/write storage also for secondary storage.

READ ONLY STORAGE

Retains the information stored at the time of manufacture, and write once storage (Write Once Read Many) allows the information to be written only once at some point after manufacture. These are called immutable storage. Immutable storage is used for tertiary and off-line storage. Examples include CD-ROM and CD-R.

SLOW WRITE, FAST READ STORAGE

Read/write storage which allows information to be overwritten multiple times, but with the write operation being much slower than the read operation. Examples include CD-RW and flash memory.

ACCESSIBILITY

RANDOM ACCESS

Any location in storage can be accessed at any moment in approximately the same amount of time. Such characteristic is well suited for primary and secondary storage.

SEQUENTIAL ACCESS

The accessing of pieces of information will be in a serial order, one after the other; therefore the time to access a particular piece of information depends upon which piece of information was last accessed. Such characteristic is typical of off-line storage.

ADDRESSABILITY

LOCATION-ADDRESSABLE

Each individually accessible unit of information in storage is selected with its numerical memory address. In modern computers, location-addressable storage usually limits to primary storage, accessed internally by computer programmes, since location-addressability is very efficient, but burdensome for humans.

FILE ADDRESSABLE

Information is divided into *files* of variable length, and a particular file is selected with human-readable directory and file names. The underlying device is still location-addressable, but the operating system of a computer provides the file system abstraction to make the operation more understandable. In modern computers, secondary, tertiary and off-line storage use file systems.

CONTENT-ADDRESSABLE

Each individually accessible unit of information is selected based on the basis of (part of) the contents stored there.

Content-addressable storage can be implemented using software (computer programme) or hardware (computer device), with hardware being faster but more expensive option. Hardware content addressable memory is often used in a computer's CPU cache.

CAPACITY

RAW CAPACITY

The total amount of stored information that a storage device or medium can hold. It is expressed as a quantity of bits or bytes (e.g. 10.4 megabytes).

MEMORY STORAGE DENSITY

The compactness of stored information. It is the storage capacity of a medium divided with a unit of length, area or volume (e.g. 1.2 megabytes per square inch).

PERFORMANCE

LATENCY

The time it takes to access a particular location in storage. The relevant unit of measurement is typically nanosecond for primary storage, millisecond for secondary storage, and second for tertiary storage. It may make sense to separate read latency and write latency, and in case of sequential access storage, minimum, maximum and average latency.

THROUGHPUT

The rate at which information can be read from or written to the storage. In computer data storage, throughput is usually expressed in terms of megabytes per second or MB/s, though bit rate may also be used.

As with latency, read rate and write rate may need to be differentiated. Also accessing media sequentially, as opposed to randomly, typically yields maximum throughput.

ENERGY USE

- Storage devices that reduce fan usage, automatically shut-down during inactivity, and low power hard drives can reduce energy consumption 90 percent.
- 2.5 inch hard disk drives often consume less power than larger ones. Low capacity solid-state drives have no moving parts and consume less power than hard disks. Also, memory may use more power than hard disks.

FUNDAMENTAL STORAGE TECHNOLOGIES

As of 2008, the most commonly used data storage technologies are semiconductor, magnetic, and optical, while paper still sees some limited usage. Some other fundamental storage technologies have also been used in the past or are proposed for development.

SEMICONDUCTOR

Semiconductor memory uses semiconductor-based integrated circuits to store information. A semiconductor memory chip may contain millions of tiny transistors or capacitors. Both *volatile* and *non-volatile* forms of semiconductor memory exist. In modern computers, primary storage almost exclusively consists of dynamic volatile semiconductor memory or dynamic random access memory. Since the turn of the century, a type of non-volatile semiconductor memory known as flash memory has steadily gained share as off-line storage for home computers. Non-volatile semiconductor memory is also used for secondary storage in various advanced electronic devices and specialized computers.

MAGNETIC

Magnetic storage uses different patterns of magnetization on a magnetically coated surface to store information. Magnetic storage is *non-volatile*. The information is accessed using one or more read/write heads which may contain one or more recording transducers. A read/write head only covers a part of the surface so that the head or medium or both must be moved relative to another in order to access data. In modern computers, magnetic storage will take these forms:

- Magnetic disk
  - o Floppy disk, used for off-line storage
  - o Hard disk drive, used for secondary storage
- Magnetic tape data storage, used for tertiary and off-line storage

In early computers, magnetic storage was also used for primary storage in a form of magnetic drum, or core memory, core rope memory, thin-film memory, twistor memory or bubble memory. Also unlike today, magnetic tape was often used for secondary storage.

OPTICAL

Optical storage, the typical optical disc, stores information in deformities on the surface of a circular disc and reads this information by illuminating the surface with a laser diode and observing the reflection. Optical disc storage is *non-volatile*. The deformities may be permanent (read only media), formed once (write once media) or reversible (recordable or read/write media). The following forms are currently in common use:

- CD, CD-ROM, DVD, BD-ROM: Read only storage, used for mass distribution of digital information (music, video, computer programmes)
- CD-R, DVD-R, DVD+R, BD-R: Write once storage, used for tertiary and off-line storage
- CD-RW, DVD-RW, DVD+RW, DVD-RAM, BD-RE: Slow write, fast read storage, used for tertiary and off-line storage
- Ultra Density Optical or UDO is similar in capacity to BD-R or BD-RE and is slow write, fast read storage used for tertiary and off-line storage.

Magneto-optical disc storage is optical disc storage where the magnetic state on a ferromagnetic surface stores information. The information is read optically and written by combining magnetic and optical methods. Magneto-optical disc storage is *non-volatile*, *sequential access*, slow write, fast read storage used for tertiary and off-line storage. 3D optical data storage has also been proposed.

PAPER

Paper data storage, typically in the form of paper tape or punched cards, has long been used to store information for automatic processing, particularly before general-purpose computers existed. Information was recorded by punching holes into the paper or cardboard medium and was read mechanically (or later optically) to determine whether a particular location on the medium was solid or contained a hole. A few technologies allow people to make marks on paper that are easily read by machine—these are widely used for tabulating votes and grading standardized tests.

Barcodes made it possible for any object that was to be sold or transported to have some computer readable information securely attached to it.

UNCOMMON

VACUUM TUBE MEMORY

A Williams tube used a cathode ray tube, and a Selectron tube used a large vacuum tube to store information. These primary storage devices were short-lived in the market, since Williams tube was unreliable and Selectron tube was expensive.

ELECTRO-ACOUSTIC MEMORY

Delay line memory used sound waves in a substance such as mercury to store information. Delay line memory was dynamic volatile, cycle sequential read/write storage, and was used for primary storage.

OPTICAL TAPE

is a medium for optical storage generally consisting of a long and narrow strip of plastic onto which patterns can be written and from which the patterns can be read back. It shares some technologies with cinema film stock and optical discs, but is compatible with neither. The motivation behind developing this technology was the possibility of far greater storage capacities than either magnetic tape or optical discs.

PHASE-CHANGE MEMORY

uses different mechanical phases of Phase Change Material to store information in an X-Y addressable matrix,

and reads the information by observing the varying electrical resistance of the material. Phase-change memory would be non-volatile, random access read/write storage, and might be used for primary, secondary and off-line storage. Most rewritable and many write once optical disks already use phase change material to store information.

HOLOGRAPHIC DATA STORAGE

Stores information optically inside crystals or photopolymers. Holographic storage can utilize the whole volume of the storage medium, unlike optical disc storage which is limited to a small number of surface layers. Holographic storage would be non-volatile, sequential access, and either write once or read/write storage. It might be used for secondary and off-line storage. See Holographic Versatile Disc (HVD).

MOLECULAR MEMORY

stores information in polymer that can store electric charge. Molecular memory might be especially suited for primary storage. The theoretical storage capacity of molecular memory is 10 terabits per square inch.

RELATED TECHNOLOGIES

NETWORK CONNECTIVITY

A secondary or tertiary storage may connect to a computer utilizing computer networks. This concept does not pertain to the primary storage, which is shared between multiple processors in a much lesser degree.

- Direct-attached storage (DAS) is a traditional mass storage, that does not use any network. This is still a most popular approach. This term was coined lately, together with NAS and SAN.
- Network-attached storage (NAS) is mass storage attached to a computer which another computer can access at file level over a local area network, a private wide area network, or in the case of online file storage, over the Internet. NAS is commonly associated with the NFS and CIFS/SMB protocols.
- Storage area network (SAN) is a specialized network, that provides other computers with storage capacity. The crucial difference between NAS and SAN is the former presents and manages file systems to client computers, whilst the latter provides access at block-addressing (raw) level, leaving it to attaching systems to manage data or file systems within the provided capacity. SAN is commonly associated with Fibre Channel networks.

ROBOTIC STORAGE

Large quantities of individual magnetic tapes, and optical or magneto-optical discs may be stored in robotic tertiary storage devices. In tape storage field they are known as tape libraries, and in optical storage field optical jukeboxes, or optical disk libraries per analogy. Smallest forms of either technology containing just one drive device are referred to as autoloaders or autochangers.

Robotic-access storage devices may have a number of slots, each holding individual media, and usually one or

more picking robots that traverse the slots and load media to built-in drives. The arrangement of the slots and picking devices affects performance. Important characteristics of such storage are possible expansion options: adding slots, modules, drives, robots. Tape libraries may have from 10 to more than 100,000 slots, and provide terabytes or petabytes of near-line information. Optical jukeboxes are somewhat smaller solutions, up to 1,000 slots.

Robotic storage is used for backups, and for high-capacity archives in imaging, medical, and video industries. Hierarchical storage management is a most known archiving strategy of automatically *migrating* long-unused files from fast hard disk storage to libraries or jukeboxes. If the files are needed, they are *retrieved* back to disk.

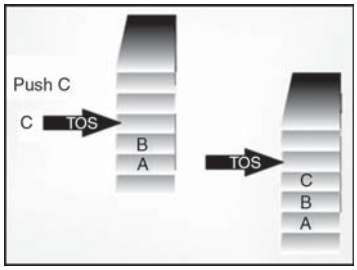
2

Data Structure

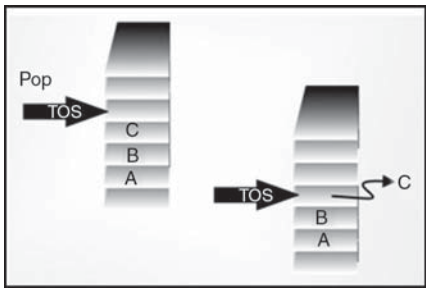
The art of Programmeming beyond this simple beginning depends on inventing, or should reinventing, sophisticated data structures. The problem is where to begin and the general consensus is that the stack is where it's at.

A stack is exactly what it sounds like. Make a stack of cards, say, on a table and you have everything you need to know about a stack. What are the basic stack operations – you can put a new card on the top of the stack and you can take a card off the top of the stack.

As long are you aren't cheating and dealing from the bottom (that would make it a data structure called a deque) putting something on the top and taking something off the top are the only two stack operations allowed. Usually these two operations are called "push" and "pull" or "push" and "pop" but what you call them doesn't really matter as long as you understand what is going on.



Pushing C onto a stack stores it where the TOS pointer indicates



Popping the stack retrieves the top data item and moves the TOS down one

If we have a stack and we do Push A, Push B and Push C what do you think you get if we next do a Pop? If your answer is C you understand how a stack works. Stacks are interesting because they can be used to alter the order of things. You stack A, B and C and you get back C, B and then A.

It reverses the order without you having to do anything and this is very useful. This accounts for the other name for a stack – a Last In First Out stack or LIFO stack. In fact this order changing ability is such a powerful property that you can build computers that have no other memory than a stack and no other storage operations than push and pop.

You can even create Programmeming languages that have nothing but a stack as their single data structure and all of their commands refer to the stack. Even though you can

have stack-oriented languages there are few popular or common languages that have stacks as standard – you have to create your own using whatever they provide. If you know how fine, if not then you have to use something less appropriate.

So, how is it done? All you need is an array and a variable to act as a pointer to the top of stack or TOS as it is usually known. The array simply has to be big enough to hold the maximum number of items. The pointer is simply set initially to the start of the array. You might set up the stack as:

```
Dim Stack(10)
Pointer=1
```

To push something on to the stack the operation is:

```
Stack(Pointer)=something
Pointer=Pointer+1
```

To pop something off the stack the operation is:

```
Pointer=Pointer-1
Something=Stack(Pointer)
```

If you want to get picky about stacks you can argue the point of whether the pointer should point to the item on the top of the stack or to the next free space on the stack. Experts prefer to have the pointer to the item on the top of stack rather than the next free space.

About the only thing that can go wrong with a stack is that it runs out of space i.e. the stack pointer goes beyond the end of the array or that it under runs, i.e. the user of the stack attempts to pop something off the stack when there isn't anything to pop.

OBJECTS

Of course the modern view isn't that the stack is a data structure – it's an object. That is, something with methods, i.e. pop and push, and an existence that goes beyond mere data. If

"CStack" is a class based implementation of a stack then in an object-oriented language we can create as many stacks as we like:

```
Dim MyStack1 As CStack
Dim MyStack2 As CStack
```

These would be used something like:

```
MyStack1.push "A"
```

or

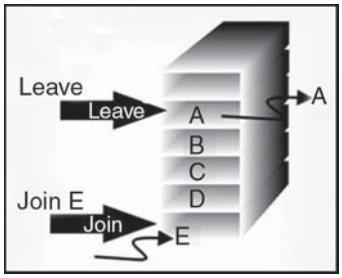
```
Data=MyStack2.pop
```

where it is assumed that the methods are push and pop.

Notice that the object-based implementation has the advantage that you don't need to ask how the stack works, what's inside the object if you like. This means that the Programmemer can implement the internal working any way that they want and they can even change things without the outside world being any the wiser. Objects are good…

QUEUE & THE DEQUE

Once you have seen a stack you can invent all of the stack-like objects that have ever been thought of, and some. For example, a queue or a First In First Out (FIFO) stack is simply an array with two pointers – one to the start and one to end of the queue. Anything added to the queue goes to the end anything taken from the queue comes from the front.



In object-oriented terms the queue has two methods, Join and Leave. A queue doesn't change the order that things are

stored in and it's mostly useful to slow things down so that they can be processed when the Programme is ready. (Hardware people often call queues "buffers".)

A queue has two pointers – one to the start of the queue and one to the end A deque is a queue that you can join and leave from the front or the back and it has four methods, JoinFront, JoinEnd, LeaveFront and LeaveEnd.

You are probably getting the idea by now. This sort of data structure simply has methods that adds new things and methods that retrieve new things according to when they were added.

TREES

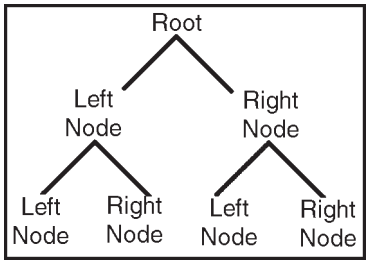
The tree is probably the most important of the really advanced data structures and in many ways it's the most complicated and sophisticated. To be abstract for the moment, a tree consists of a set of nodes – the places where the branches do their branching – and the branches that connect the nodes.

The idea, or rather the picture, that you should have in your head of a branching tree-like structure is simple enough, actually implementing it in computer terms is not quite as straightforward. To start with the nodes are the things that we work with.

A node has space to store the data in the tree, a person's name for example, and it also stores "pointers" to the nodes that are connected to its child nodes. The big problem is how many pointers?

In general a tree can have nodes with as many child nodes as you care to imagine and this is difficult to implement. As a result

we like trees that have a fixed number of child nodes – 2 say gives us the binary tree. In this case, i.e. a binary tree, we can refer to the "right child node" and the "left child node" or just the "right" and "left" nodes. A tree also has a first or "root node" that everything else grows from.



In a binary tree each node has two child nodes – one left and one right.

Now we can invent an object that behaves like a node in a binary tree, i.e. an object with two properties - left and right child nodes, and a single data property, which stores the value at the node.

We can now build a tree by creating a root node:

```
Dim root As New Node
```

and then add extra nodes as required:

```
Root.LeftNode=New Node
Root.RightNode=New Node
```

How do we get at these nodes so that we can grow the tree another level? Easy we just continue the same notation:

```
Root.LeftNode.LeftNode=New Node
 Root.LeftNode.RightNode=New Node
```

and so on.

Of course we are also assuming that nodes have a data property which can be used to store whatever you want at each node. For example,

```
Root.LeftNode.data= "john"
```

would store the indicated name at the first left child node from the root. You should be able to see how to build up a

data structure that corresponds to a family tree in which each “node” gives rise to just two offspring.

If you want more, or even a variable number of, offspring at each node then it works in the same sort of way but you need a collection of pointers to the child nodes and this is awkward to write down. There is also the small problem of working with trees. Clearly you can keep on writing out Left Node.Right Node.Right Node. Left Node type names to specify a node of your choice – it’s too clumsy. What actually happens in practice is that a pointer to a node is used as the “current” node and this moves its way down the tree by being set to the current nodes left or right child:

```
CurrentNode=CurrentNode.LeftNode
```

And so on. You can generalise this to more than two child nodes and even work out ways of writing Programmes that do standard tasks such as visiting every node in a tree. This is again another area where things can seem complicated because you can ask that every node is visited in a particular order.

For example going down each branch as far as possible before starting again to go down the next branch is called"depth first", while visiting all nodes at the same depth before going deeper, is called"breadth first". The jargon also gets more impressive - we talk of"traversing the tree" and eventually you will encounter the fact that trees and"recursion" go together. Recursion is a whole topic in its own right and the source of the only good computer science joke I know - dictionary definition of recursion"Recursion - see recursion".

What are trees used for? Well, everything from keeping track of where files are stored on a disk drive to analysing

natural language and applications in artificial intelligence. It's worth point out that XML is a data language that can only describe tree structures so the idea must be powerful. You can't Programme for long without meeting trees.

METHODOLOGY AUGMENT OF DATA STRUCTURES

There are some Programmeming situations that can be perfectly solved with standard data structures such as a linked lists, hash tables, or binary search trees. Many others require a dash of creativity. Only in rare situations will you need to create an entirely new type of data structure, though. More often, it will suffice to augment (to modify) an existing data structure by storing additional information in it. You can then Programme new operations for the data structure to support the desired application. Augmenting a data structure is not always straightforward, however, since the added information must be updated and maintained by the ordinary operations on the data structure. Data structure that supports general order-statistic operations on a dynamic set. It's called dynamic order statistics. Any order statistic could be retrieved in O(n) time from an unordered set. How red-black trees can be modified so that any order statistic can be determined in O(lg(n)) time. It presents two algorithms OS-Select(i), which returns i-th smallest item in a dynamic set, and OS-Rank(x), which returns rank (position) of element x in sorted order.

General methodology of how to augment a data structure.

Augmenting a data structure can be broken into four steps:

- Choosing an underlying data structure.
- Determining additional information to be maintained in the underlying data structure.
- Verifying that the additional information can be maintained for the basic modifying operations (insert, delete, rotate, etc.) on the underlying data structure.
- Developing new operations.

The second part of the lecture applies this methodology to construct a data structure called interval trees. This data structure maintains a dynamic set of elements, with each element x containing an interval. Interval is simply pair of numbers (low, high). For example, a time interval from 3 o'clock to 7 o'clock is a pair (3, 7).

Lecture gives an algorithm called Interval-Search(x), which given a query interval x, quickly finds an interval in the set that overlaps it. Time complexity of this algorithm is O(lg(n)).

ABOUT LUCTURE

The lecture is motivated by two things:

- The implementation of ADTs that extend standard ADTs by one or more additional operations;
- Application in which data "live in" various data structures simultaneously.

Augmentation is a process by which one adds fields to the nodes as necessary.

Augmenting a data structure can be broken into four steps:

- Choosing an underlying data structure
- Determining additional information to be maintained in the underlying data structure

- Verifying that the additional information can be efficiently maintained for the basic modifying operations on the underlying data structure
- Developing new operations

DYNAMIC ORDER-STATISTIC TREES

ADT: set S; Dictionary operations (*Insert, Delete, Search*) + two additional operations:

- *Rank(S,x)*: Returns rank of node x (smallest rank is 1)
- *Select(S, i)*: Select ith smallest element from set S

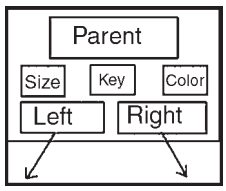
IMPLEMENTATION

An *order-statistic tree* T is simply a red-black tree with additional information stored at each node. Besides the usual red-black tree fields *key*[x], *left*[x], *right*[x],*p*[x] and *Colour*[x] in a node *x*, we have another field *size*[*x*]. This field contains the number of internal nodes in the subtree rooted at *x* (including x itself), that is, the size of the subtree.

$$size[x] = size[left[x]] + size[right[x]] + 1$$

The node of the *order-statistic tree* will now look as follows:

figure



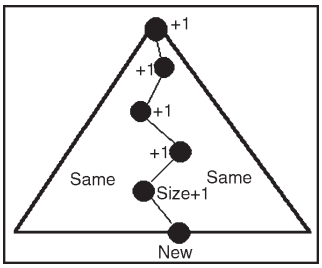
There are couple of things though that we have to worry about:

1. Can *size* be updated in $O(\log_2 n)$ time per operation?
2. *Rank*(x) and *select*(S, i) to be done in $O(\log_2 n)$ time.

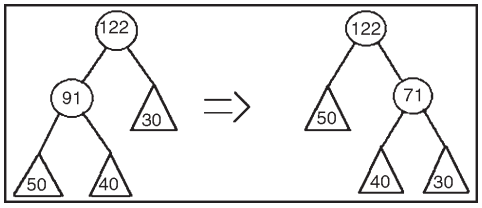
OPERATIONS ON ORDER-STATISTIC TREES

INSERT

As we know, insertion into a red-black tree consists of two phases. Phase 1 goes down the path from the root inserting the new node as a child of the existing node. To maintain the subtree sizes we simply increment *size*[*x*] for each node *x* on the path traversed from the root down toward the leaves. The new node added gets size 1. Figure illustrates Phase 1.



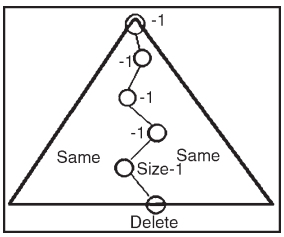
In the Phase 2, the only structural changes to the underlying red-black tree are caused by *rotations*, of which there are at most two. Rotation is a local operation and it invalidates only the two *size* fields in the nodes incident on the link around which the rotation is performed. Figure shows *Left* and *Right*rotations.



DELETE

Phase 1 splices out the node we wish to delete. To update subtree sizes we simply traverse a path from node we wish

to delete up to the root, decrementing*size* field for each node on the path. Figure illustrates this process.



The rotations in the Phase 2 are handled in the same manner as for insertion.

Running time: To maintain tree sizes in the Phase 1 of insertion or deletion we have to increment or decrement *size*[*x*] for each node *x* on the path from the root down toward the leaves. Since there are $O(\log_2 n)$ nodes on the traversed path, the additional cost of maintaining the *size* fields is $O(\log_2 n)$. Moreover, rotation is a local operation and hence only O(1) additional time is spent updating *size* fields in the Phase 2. Thus, both insertion and deletion take $O(\log_2 n)$ time.

SELECT

The procedure *Select*(*x*, *i*) returns a pointer to the node containing the ith smallest key in the subtree rooted at *x*.

```
Select(x, i)
r (rank of root) = size[left[x]] + 1
case r = i: return x
case r > i: return Select(left[x], i)
case r < i: return Select(right[x], i)
```

Because each recursive call goes down one level in the order-statistic tree, the total time for *Select* is at worst proportional to the height of the tree. Since the tree is the red-black tree, its height is $O(\log_2 n)$. Thus, the running time of *Select* is $O(\log_2 n)$.

RANK

The procedure *Rank*(*T*, *x*) returns the position of *x* in the linear order determined by an inorder tree walk of *T*. Let *x* is a pointer to the node in the tree.

```
Rank(T, x)
r = size[left[x]] + 1
y = x
while y <> root do
if right[parent[y]] = y
then r += size[left[parent[y]]] + 1
y = parent[y]
return r
```

Since each iteration of the while loop takes O(1) time and *y* goes up one level in the tree with each iteration, the running time of *Rank* is at worst proportional to the height of the tree: $O(\log_2 n)$. Figure illustrates the *Rank* procedure.

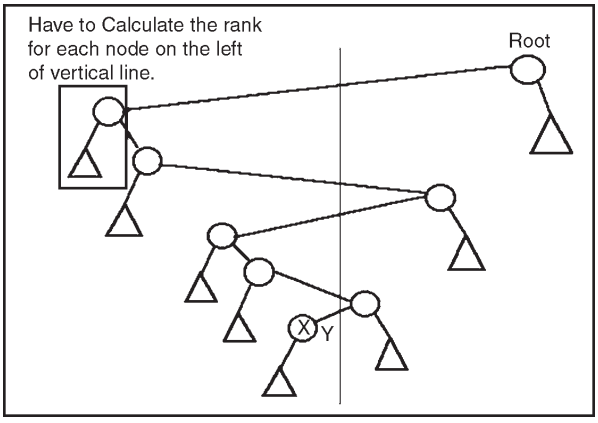


Fig. Link to our Java Applet

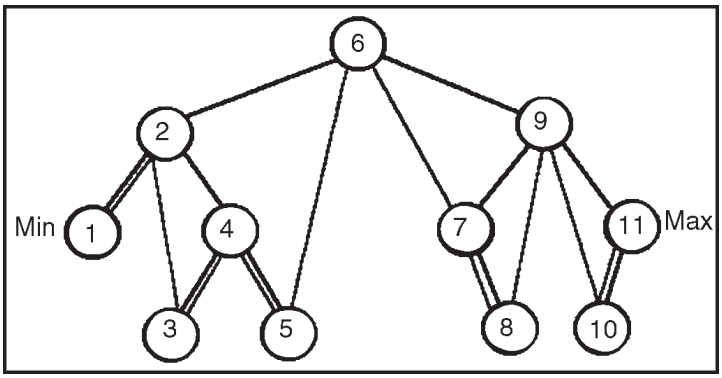
BINARY SEARCH TREES FOR BROWSING ADT

Dictionary + Browsing operations:

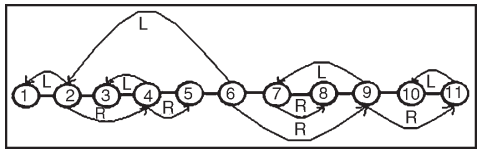
- MIN
- MAX
- Predecessor
- Successor

IMPLEMENTATION

The underlying data structure is a red-black tree where besides usual red-black tree fields each node is augmented with *Min*[*x*], *Max*[*x*], *Predecessor*[*x*] and*Successor*[*x*]. Figure shows the augmented Red-Black tree on the input {1 2 3 4 5 6 7 8 9 10 11}.

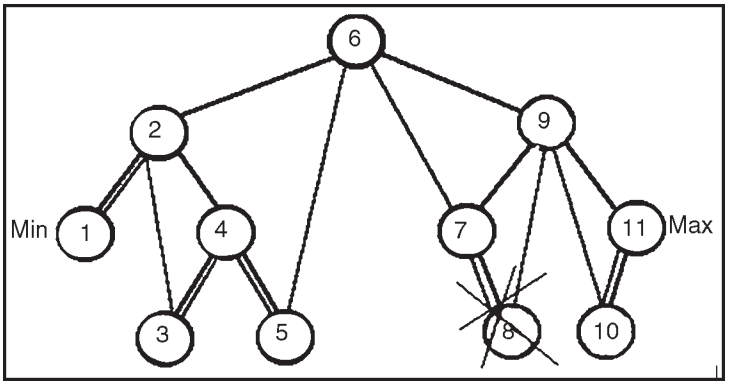


The data structure in Figure below can be looked at as a binary search tree or a sorted linked list as shown in Figure. In fact, according to Prof. Devroye, it is a smooth "marriage" of both data structures.



When we insert or delete a node we always have to update *predecessor* and *successor* pointers. We have deleted a new node with a *key* value 8. Accordingly we have to update *successor* and *predecessor* pointers.

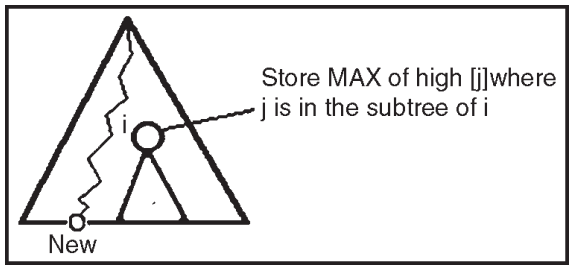
The time it takes to update pointers is equal to $O(\log_2 n)$, since we either follow a path up the tree or down the tree. Rotation operations take as usual O(1) time, so the running time for insertion or deletion is $O(\log_2 n)$. In figure the updated pointer.



INTERVAL TREES

Interval tree is a binary search tree for intervals which are efficient for the dictionary operations and overlap. *Overlap*(*x*, *i*) for interval *i* and a tree rooted at *x*, returns a pointer to an interval in the collection that overlaps the given interval *i* or returns NIL otherwise. The underlying data structure is a red-black tree in which each node *x* contains an interval *int*[*x*] and the key of *x* is a low endpoint of interval, *low*[*int*[*x*]].*high*[*x*] is the high endpoint of interval.

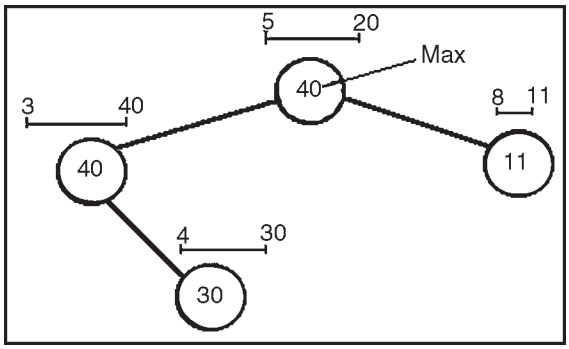
In addition, each node *x* contains a value *max*[*x*] which is the maximum value of any interval endpoint stored in the subtree rooted at *x*.



OPERATIONS

- Determine overlap with some interval in the tree: return Yes or No

- Dictionary operations (Insert, Delete, Search)



Given interval *i* and pointer to the root *x* Overlap(x, i) returns NIL if no overlap occured or pointer to the node if there is an overlap.

```
Overlap(x, i)
if x = NIL then return NIL
while x <> NIL and i doesn't overlap int[x] do
if left[x] <> NIL and low[i] Ü max[left[x]]
then x = left[x]
else x = right[x]
```

RUNNING TIME

The search for the inteval that overlaps *i* starts with *x* at the root of the tree and proceeds downward. It terminates when either overlapping interval is found or *x* becomes NIL. Since each iteration of the basic loop takes O(1) time, and since the height or the red-black tree is $O(\log_2 n)$ *Overlap*takes $O(\log_2 n)$ time.

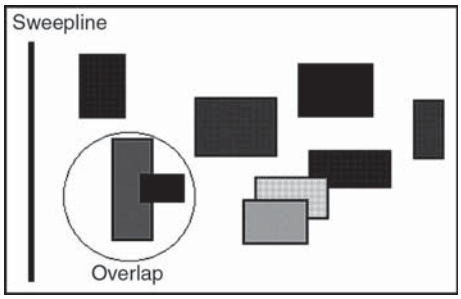
Insertion of a node *x* into a tree consists of *two* phases. During the first phase *x* is inserted as a child of an existing node. The value of *max*[*x*] can be computed in O(1) time since it depends only on information in the other fields of *x* itself and *x*'s children, but *x*'s children are both NIL.

Once *max*[*x*] is computed, the change propagates up the tree. Thus, total time for the first phase is $O(\log_2 n)$. During

second phase the only structural changes are caused by rotations. Since only *two* nodes change in rotation, the total time for updating the *max* fields is $O(\log_2 n)$ per rotation. Since the number of rotations during insertion is at most *two*, the total time for insertion is $O(\log_2 n)$.

In the first phase of deletion, changes occur if the deleted node is replaced by its successor, and then again when either the deleted node or its successor is spliced out. Propagating the updates to *max* caused by these changes costs at most $O(\log_2 n)$ since the changes modify the tree locally. Fixing up the red-black tree during the second phase requires at most *three* rotations, and each rotation requires at most $O(\log_2 n)$ time to propagate the updates to *max*. Thus, like insertion, the total time for deletion is $O(\log_2 n)$.

Problem: Layout of VSLI chips



In this section we present a problem which is motivated by automated chip design. The problem is how can we place many printed circuits on one chip without overlapping? To make the problem more amenable to analysis, we assume that all n rectangles to be aligned with horizontal axis. We also assume that we are given a configuration of the rectangles, and our task is merely to check whether there is any overlap between rectangles.

In the naive solution we try to remove overlapped rectangles and place them randomly again. It takes time of $\Theta(n^2)$. The

more efficient algorithm, which takes $O(n\log_2 n)$ time, uses a technique known as *sweeping*.

In sweeping an imaginary vertical sweep line passes through a given set of geometric objects (rectangles), usually from left to right. Sweeping provides a method for ordering geometric objects, usualy by placing them into dynamic data structure. We sort the rectangles' endpoints by increasing x-coordinate and proceed from left to right. We insert y-direction intervals into an interval tree when its left endpoint is encountered and we delete it from interval tree when its right endpoint is encountered. Whether two segments first become consecutive in the total order, we check if they overlap.

3

Mass Storage

In computing, mass storage refers to the storage of large amounts of data in a persisting and machine-readable fashion. Devices and/or systems that have been described as mass storage include tape libraries, RAID systems, hard disk drives, magnetic tape drives, optical disc drives, magneto-optical disc drives, drum memory (historic), floppy disk drives (historic), punched tape (historic) and holographic memory (experimental). Mass storage includes devices with removable and non-removable media. It does not include random access memory (RAM), which is volatile in that it loses its contents after power loss.

The notion of "large" amounts of data is of course highly dependent on the time frame and the market segment, as mass storage device capacity has increased by many orders of magnitude since the beginnings of computer technology in the late 1940s and continues to grow; however, in any

time frame, common mass storage devices have tended to be much larger and at the same time much slower than common realizations of the contemporaneous primary storage technology. The term *mass storage* was used in the PC marketplace for devices far smaller than devices that were not considered mass storage in the mainframe marketplace.

Mass storage devices are characterized by:

- Sustainable transfer speed
- Seek time
- Cost
- Capacity

Today, magnetic disks are the predominant storage media in personal computers. Optical discs, however, are almost exclusively used in the large-scale distribution of retail software, music and movies because of the cost and manufacturing efficiency of the molding process used to produce DVD and compact discs and the nearly-universal presence of reader drives in personal computers and consumer appliances. Flash memory (in particular, NAND flash) has an established and growing niche as a replacement for magnetic hard disks in high performance enterprise computing installations because it has no moving parts (making it more robust) and has a much lower latency; as removable storage such as USB sticks, because in lower capacity ranges it can be made smaller and cheaper than hard disks; and on portable devices such as notebook computers and cell phones because of its lower size and weight, better tolerance of physical stress caused by e.g.

shaking or dropping, and low power consumption. The design of computer architectures and operating systems are often dictated by the mass storage and bus technology of their time. Desktop operating systems such as Windows are now so closely tied to the performance characteristics of magnetic disks that it is difficult to deploy them on other media like flash memory without running into space constraints, suffering serious performance problems or breaking applications.

USAGE

Mass storage devices used in desktop and most server computers typically have their data organized in a file system. The choice of file system is often important in maximizing the performance of the device: general purpose file systems (such as NTFS and HFS, for example) tend to do poorly on slow-seeking optical storage such as compact discs. Some relational databases can also be deployed on mass storage devices without an intermediate file system or storage manager. Oracle and MySQL, for example, can store table data directly on raw block devices. On removable media, archive formats (such as tar archives on magnetic tape, which pack file data end-to-end) are sometimes used instead of file systems because they are more portable and simpler to stream.

On embedded computers, it is common to memory map the contents of a mass storage device (usually ROM or flash memory) so that its contents can be traversed as in-memory data structures or executed directly by programmes.

STABLE STORAGE

Stable storage is a classification of computer data storage technology that guarantees atomicity for any given write operation and allows software to be written that is robust against some hardware and power failures. To be considered atomic, upon reading back a just written-to portion of the disk, the storage subsystem must return either the write data or the data that was on that portion of the disk before the write operation. Most computer disk drives are not considered stable storage because they do not guarantee atomic write: an error could be returned upon subsequent read of the disk where it was just written to in lieu of either the new or prior data.

MULTIPLE TECHNIQUES

Multiple techniques have been developed to achieve the atomic property from weakly-atomic devices such as disks. Writing data to a disk in two places in a specific way is one technique and can be done by application software. Most often though, stable storage functionality is achieved by mirroring data on separate disks via RAID technology (level 1 or greater). The RAID controller implements the disk writing algorithms that enable separate disks to act as stable storage. The RAID technique is robust against some single disk failure in an array of disks whereas the software technique of writing to separate areas of the same disk only protects against some kinds of internal disk media failures such as bad sectors in single disk arrangements.

DRAM PACKAGING

For economic reasons, the large (main) memories found in personal computers, workstations, and non-handheld

game-consoles (such as PlayStation and Xbox) normally consists of dynamic RAM (DRAM). Other parts of the computer, such as cache memories and data buffers in hard disks, normally use static RAM (SRAM).

GENERAL DRAM PACKAGING FORMATS

Dynamic random access memory is produced as integrated circuits (ICs) bonded and mounted into plastic packages with metal pins for connection to control signals and buses. Today, these DRAM packages are in turn often assembled into plug-in modules for easier handling. Some standard module types are:

- DRAM chip (Integrated Circuit or IC)
  - o Dual in-line Package (DIP)
- DRAM (memory) modules
  - o Single In-line Pin Package (SIPP)
  - o Single In-line Memory Module (SIMM)
  - o Dual In-line Memory Module (DIMM)
  - o Rambus In-line Memory Module (RIMM), technically DIMMs but called RIMMs due to their proprietary slot.
  - o Small outline DIMM (SO-DIMM), about half the size of regular DIMMs, are mostly used in notebooks, small footprint PCs (such as Mini-ITX motherboards), upgradable office printers and networking hardware like routers. Comes in versions with:
    - 72-pin (32-bit)
    - 144-pin (64-bit) used for PC100/PC133 SDRAM
    - 200-pin (72-bit) used for DDR and DDR2
    - 240-pin (72-bit) used for DDR3

  - Small outline RIMM (SO-RIMM). Smaller version of the RIMM, used in laptops. Technically SO-DIMMs but called SO-RIMMs due to their proprietary slot.
- Stacked vs. non-stacked RAM modules
  - Stacked RAM modules contain two or more RAM chips stacked on top of each other. This allows large modules (like 512 MB or 1 GB SO-DIMM) to be manufactured using cheaper low density wafers. Stacked chip modules draw more power, and with the advent of commodity BGA memories are no longer physically possible to construct.

COMMON DRAM MODULES

Common DRAM packages as illustrated to the right, from top to bottom:

1. DIP 16-pin (DRAM chip, usually pre-FPRAM)
2. SIPP (usually FPRAM)
3. SIMM 30-pin (usually FPRAM)
4. SIMM 72-pin (often EDO RAM but FPM is not uncommon)
5. DIMM 168-pin (SDRAM)
6. DIMM 184-pin (DDR SDRAM)
7. RIMM 184-pin (RDRAM)
8. DIMM 240-pin (DDR2 SDRAM/DDR3 SDRAM)

VARIATIONS

While the fundamental DRAM cell and array has maintained the same basic structure (and performance) for many years, there have been many different interfaces for

speaking with DRAM chips. When one speaks about "DRAM types", one is generally referring to the interface that is used.

ASYNCHRONOUS DRAM

This is the basic form, from which all others derive. An asynchronous DRAM chip has power connections, some number of address inputs (typically 12), and a few (typically one or four) bidirectional data lines. There are four active low control signals:

- /RAS, the Row Address Strobe. The address inputs are captured on the falling edge of /RAS, and select a row to open. The row is held open as long as /RAS is low.
- /CAS, the Column Address Strobe. The address inputs are captured on the falling edge of /CAS, and select a column from the currently open row to read or write.
- /WE, Write Enable. This signal determines whether a given falling edge of /CAS is a read (if high) or write (if low). If low, the data inputs are also captured on the falling edge of /CAS.
- /OE, Output Enable. This is an additional signal that controls output to the data I/O pins. The data pins are driven by the DRAM chip if /RAS and /CAS are low, /WE is high, and /OE is low. In many applications, /OE can be permanently connected low (output always enabled), but it can be useful when connecting multiple memory chips in parallel.

This interface provides direct control of internal timing. When /RAS is driven low, a /CAS cycle must not be attempted until the sense amplifiers have sensed the memory state, and /RAS must not be returned high until the storage cells have been refreshed. When /RAS is driven high, it must be held high long enough for precharging to complete. Although the RAM is asynchronous, the signals are typically generated by a clocked memory controller, which limits their timing to multiples of the controller's clock cycle.

VIDEO DRAM (VRAM)

VRAM is a dual-ported variant of DRAM that was once commonly used to store the frame-buffer in some graphics adaptors.

WINDOW DRAM (WRAM)

WRAM is a variant VRAM that was once used in graphics adaptors such as the Matrox Millenium and ATI 3D Rage Pro. WRAM was designed to perform better and cost less than VRAM. WRAM offered up to 25% greater bandwidth than VRAM and accelerated commonly used graphical operations such as text drawing and block fills.

FAST PAGE MODE (FPM) DRAM OR FPRAM

Fast page mode DRAM is also called FPM DRAM, Page mode DRAM, Fast page mode memory, or Page mode memory. In page mode, a row of the DRAM can be kept "open" by holding /RAS low while performing multiple reads or writes with separate pulses of /CAS so that successive reads or writes within the row do not suffer the delay of precharge

and accessing the row. This increases the performance of the system when reading or writing bursts of data. Static column is a variant of page mode in which the column address does not need to be strobed in, but rather, the address inputs may be changed with /CAS held low, and the data output will be updated accordingly a few nanoseconds later. Nibble mode is another variant in which four sequential locations within the row can be accessed with four consecutive pulses of /CAS. The difference from normal page mode is that the address inputs are not used for the second through fourth /CAS edges; they are generated internally starting with the address supplied for the first /CAS edge.

CAS BEFORE RAS REFRESH

Classic asynchronous DRAM is refreshed by opening each row in turn. This can be done by supplying a row address and pulsing /RAS low; it is not necessary to perform any /CAS cycles. An external counter is needed to iterate over the row addresses in turn. For convenience, the counter was quickly incorporated into RAM chips themselves. If the /CAS line is driven low before /RAS (normally an illegal operation), then the DRAM ignores the address inputs and uses an internal counter to select the row to open. This is known as /CAS-before-/RAS (CBR) refresh. This became the standard form of refresh for asynchronous DRAM, and is the only form generally used with SDRAM.

HIDDEN REFRESH

Given support of CAS-before-RAS refresh, it is possible to deassert /RAS while holding /CAS low to maintain data

output. If /RAS is then asserted again, this performs a CBR refresh cycle while the DRAM outputs remain valid. Because data output is not interrupted, this is known as "hidden refresh".

EXTENDED DATA OUT (EDO) DRAM

EDO DRAM, sometimes referred to as Hyper Page Mode enabled DRAM, is similar to Fast Page Mode DRAM with the additional feature that a new access cycle can be started while keeping the data output of the previous cycle active. This allows a certain amount of overlap in operation (pipelining), allowing somewhat improved performance. It was 5% faster than Fast Page Mode DRAM, which it began to replace in 1995, when Intel introduced the 430FX chipset that supported EDO DRAM. To be precise, EDO DRAM begins data output on the falling edge of /CAS, but does not stop the output when /CAS rises again. It holds the output valid (thus extending the data output time) until either /RAS is deasserted, or a new /CAS falling edge selects a different column address. Single-cycle EDO has the ability to carry out a complete memory transaction in one clock cycle. Otherwise, each sequential RAM access within the same page takes two clock cycles instead of three, once the page has been selected. EDO's performance and capabilities allowed it to somewhat replace the then-slow L2 caches of PCs. It created an opportunity to reduce the immense performance loss associated with a lack of L2 cache, while making systems cheaper to build. This was also good for notebooks due to difficulties with their limited form factor, and battery life limitations. An EDO system

with L2 cache was tangibly faster than the older FPM/L2 combination. Single-cycle EDO DRAM became very popular on video cards towards the end of the 1990s. It was very low cost, yet nearly as efficient for performance as the far more costly VRAM.

Much equipment taking 72-pin SIMMs could use either FPM or EDO. Problems were possible, particularly when mixing FPM and EDO. Early Hewlett-Packard printers had FPM RAM built in; some, but not all, models worked if additional EDO SIMMs were added.

BURST EDO (BEDO) DRAM

An evolution of EDO DRAM, Burst EDO DRAM, could process four memory addresses in one burst, for a maximum of 5 1 1 1, saving an additional three clocks over optimally designed EDO memory. It was done by adding an address counter on the chip to keep track of the next address. BEDO also added a pipelined stage allowing page-access cycle to be divided into two components. During a memory-read operation, the first component accessed the data from the memory array to the output stage (second latch). The second component drove the data bus from this latch at the appropriate logic level. Since the data is already in the output buffer, quicker access time is achieved (up to 50% for large blocks of data) than with traditional EDO.

Although BEDO DRAM showed additional optimization over EDO, by the time it was available the market had made a significant investment towards synchronous DRAM, or SDRAM. Even though BEDO RAM was superior to SDRAM in some ways, the latter technology quickly displaced BEDO.

MULTIBANK DRAM (MDRAM)

Multibank RAM applies the interleaving technique for main memory to second level cache memory to provide a cheaper and faster alternative to SRAM. The chip splits its memory capacity into small blocks of 256 kB and allows operations to two different banks in a single clock cycle.

This memory was primarily used in graphic cards with Tseng Labs ET6x00 chipsets, and was made by MoSys. Boards based upon this chipset often used the unusual RAM size configuration of 2.25 MB, owing to MDRAM's ability to be implemented in various sizes more easily. This size of 2.25 MB allowed 24-bit color at a resolution of 1024×768, a very popular display setting in the card's time.

SYNCHRONOUS GRAPHICS RAM (SGRAM)

SGRAM is a specialized form of SDRAM for graphics adaptors. It adds functions such as bit masking (writing to a specified bit plane without affecting the others) and block write (filling a block of memory with a single colour). Unlike VRAM and WRAM, SGRAM is single-ported. However, it can open two memory pages at once, which simulates the dual-port nature of other video RAM technologies.

SINGLE DATA RATE (SDR)

Single data rate SDRAM (sometimes known as SDR) is a synchronous form of DRAM.

DOUBLE DATA RATE (DDR)

Double data rate SDRAM (DDR) was a later development of SDRAM, used in PC memory beginning in 2000.

Subsequent versions are numbered sequentially (DDR2, DDR3, etc.).

PSEUDOSTATIC RAM (PSRAM)

PSRAM or PSDRAM is dynamic RAM with built-in refresh and address-control circuitry to make it behave similarly to static RAM (SRAM). It combines the high density of DRAM with the ease of use of true SRAM. PSRAM (made by Numonyx) is used in the Apple iPhone and other embedded systems. Some DRAM components have a "self-refresh mode". While this involves much of the same logic that is needed for pseudo-static operation, this mode is often equivalent to a standby mode. It is provided primarily to allow a system to suspend operation of its DRAM controller to save power without losing data stored in DRAM, not to allow operation without a separate DRAM controller as is the case with PSRAM. An embedded variant of pseudostatic RAM is sold by MoSys under the name 1T-SRAM. It is technically DRAM, but behaves much like SRAM. It is used in Nintendo Gamecube and Wii consoles.

1T DRAM

Unlike all of the other variants described here, 1T DRAM is actually a different way of constructing the basic DRAM bit cell. 1T DRAM is a "capacitorless" bit cell design that stores data in the parasitic body capacitor that is an inherent part of Silicon on Insulator transistors. Considered a nuisance in logic design, this floating body effect can be used for data storage. Although refresh is still required, reads are non-destructive; the stored charge causes a

detectable shift in the threshold voltage of the transistor. There are several types of 1T DRAM memories: The commercialized Z-RAM from Innovative Silicon, the TTRAM from Renesas and the A-RAM from the UGR /CNRS consortium. Note that classic one-transistor/one-capacitor (1T/1C) DRAM cell is also sometimes referred to as "1T DRAM".

RLDRAM

Reduced Latency DRAM is a high performance double data rate (DDR) SDRAM that combines fast, random access with high bandwidth. RLDRAM is mainly designed for networking and caching applications.

SECURITY

Although dynamic memory is only *guaranteed* to retain its contents when supplied with power and refreshed every 64 ms, the memory cell capacitors will often retain their values for significantly longer, particularly at low temperatures. Under some conditions, most of the data in DRAM can be recovered even if the DRAM has not been refreshed for several minutes. This property can be used to recover "secure" data kept in memory by quickly rebooting the computer and dumping the contents of the RAM or by cooling the chips and transferring them to a different computer. Such an attack was demonstrated to circumvent popular disk encryption systems, like the open source TrueCrypt, Microsoft's BitLocker Drive Encryption, as well as Apple's FileVault.

4

Static Random-Access Memory

Static random-access memory (SRAM) is a type of semiconductor memory where the word *static* indicates that, unlike *dynamic* RAM (DRAM), it does not need to be periodically refreshed, as SRAM uses bistable latching circuitry to store each bit. SRAM exhibits data remanence, but is still *volatile* in the conventional sense that data is eventually lost when the memory is not powered.

DESIGN

Each bit in an SRAM is stored on four transistors that form two cross-coupled inverters. This storage cell has two stable states which are used to denote 0 and 1. Two additional *access* transistors serve to control the access to a storage cell during read and write operations. A typical SRAM uses six MOSFETs to store each memory bit. In addition to such 6T SRAM, other kinds of SRAM chips use 8T, 10T, or more

transistors per bit. This is sometimes used to implement more than one (read and/or write) port, which may be useful in certain types of video memory and register files implemented with multi ported SRAM circuitry. Generally, the fewer transistors needed per cell, the smaller each cell can be. Since the cost of processing a silicon wafer is relatively fixed, using smaller cells and so packing more bits on one wafer reduces the cost per bit of memory.

Memory cells that use fewer than 6 transistors are possible — but such 3T or 1T cells are DRAM, not SRAM (even the so-called 1T-SRAM). Access to the cell is enabled by the word line (WL in figure) which controls the two *access* transistors M_5 and M_6 which, in turn, control whether the cell should be connected to the bit lines: BL and BL. They are used to transfer data for both read and write operations. Although it is not strictly necessary to have two bit lines, both the signal and its inverse are typically provided in order to improve noise margins.

During read accesses, the bit lines are actively driven high and low by the inverters in the SRAM cell. This improves SRAM bandwidth compared to DRAMs—in a DRAM, the bit line is connected to storage capacitors and charge sharing causes the bitline to swing upwards or downwards. The symmetric structure of SRAMs also allows for differential signaling, which makes small voltage swings more easily detectable. Another difference with DRAM that contributes to making SRAM faster is that commercial chips accept all address bits at a time. By comparison, commodity DRAMs have the address multiplexed in two halves, i.e. higher bits followed by lower bits, over the same package pins in order

to keep their size and cost down. The size of an SRAM with m address lines and n data lines is 2^m words, or $2^m \times n$ bits. At present appeared the advanced scheme with disconnected by signal record by feedback, which does not require the transistor of the load and is accordingly saved from high consumption of the energy when writing.

SRAM OPERATION

An SRAM cell has three different states it can be in: *standby* where the circuit is idle, *reading* when the data has been requested and *writing* when updating the contents. The SRAM to operate in read mode and write mode should have "readeability" and "write stability" respectively. The three different states work as follows:

STANDBY

If the word line is not asserted, the *access* transistors M_5 and M_6 disconnect the cell from the bit lines. The two cross coupled inverters formed by M_1 – M_4 will continue to reinforce each other as long as they are connected to the supply.

READING

Assume that the content of the memory is a 1, stored at Q. The read cycle is started by precharging both the bit lines to a logical 1, then asserting the word line WL, enabling both the *access* transistors. The second step occurs when the values stored in Q and Q are transferred to the bit lines by leaving BL at its precharged value and discharging BL through M_1 and M_5 to a logical 0. On the BL side, the

transistors M_4 and M_6 pull the bit line toward V_{DD}, a logical 1. If the content of the memory was a 0, the opposite would happen and BL would be pulled toward 1 and BL toward 0. Then these BL and BL-bar will have a small difference of delta between them and then these lines reach a sense amplifier, which will sense which line has higher voltage and thus will tell whether there was 1 stored or 0. The higher the sensitivity of sense amplifier, the faster the speed of read operation is.

WRITING

The start of a write cycle begins by applying the value to be written to the bit lines. If we wish to write a 0, we would apply a 0 to the bit lines, i.e. setting BL to 1 and BL to 0. This is similar to applying a reset pulse to a SR-latch, which causes the flip flop to change state. A 1 is written by inverting the values of the bit lines. WL is then asserted and the value that is to be stored is latched in. Note that the reason this works is that the bit line input-drivers are designed to be much stronger than the relatively weak transistors in the cell itself, so that they can easily override the previous state of the cross-coupled inverters. Careful sizing of the transistors in an SRAM cell is needed to ensure proper operation.

BUS BEHAVIOUR

A RAM memory with an access time of 70 ns will output valid data within 70 ns from the time that the address lines are valid. But the data will remain for a hold time as well (5-10 ns). Rise and fall times also influence valid timeslots

with approximately ~5 ns. By reading the lower part of an address range bits in sequence (page cycle) one can read with significantly shorter access time (30 ns).

APPLICATIONS AND USES

CHARACTERISTICS

SRAM is more expensive, but faster and significantly less power hungry (especially idle) than DRAM. It is therefore used where either bandwidth or low power, or both, are principal considerations. SRAM is also easier to control (interface to) and generally more truly *random access* than modern types of DRAM. Due to a more complex internal structure, SRAM is less dense than DRAM and is therefore not used for high-capacity, low-cost applications such as the main memory in personal computers.

CLOCK RATE AND POWER

The power consumption of SRAM varies widely depending on how frequently it is accessed; it can be as power-hungry as dynamic RAM, when used at high frequencies, and some ICs can consume many watts at full bandwidth. On the other hand, static RAM used at a somewhat slower pace, such as in applications with moderately clocked microprocessors, draw very little power and can have a nearly negligible power consumption when sitting idle — in the region of a few micro-watts. Static RAM exists primarily as:

- general purpose products
  - o with *asynchronous* interface, such as the 28 pin 32Kx8 chips (usually named XXC256), and similar products up to 16 Mbit per chip

  - with *synchronous* interface, usually used for caches and other applications requiring burst transfers, up to 18 Mbit (256Kx72) per chip
- integrated on chip
  - as RAM or cache memory in micro-controllers (usually from around 32 bytes up to 128 kilobytes)
  - as the primary caches in powerful microprocessors, such as the x86 family, and many others (from 8 kB, up to several megabytes)
  - to store the registers and parts of the state-machines used in some microprocessors—see register file
  - on application specific ICs, or ASICs (usually in the order of kilobytes)
  - in FPGAs and CPLDs

EMBEDDED USE

Many categories of industrial and scientific subsystems, automotive electronics, and similar, contain static RAM. Some amount (kilobytes or less) is also embedded in practically all modern appliances, toys, etc. that implement an electronic user interface. Several megabytes may be used in complex products such as digital cameras, cell phones, synthesizers, etc.

SRAM in its dual-ported form is sometimes used for realtime digital signal processing circuits.

IN COMPUTERS

SRAM is also used in personal computers, workstations, routers and peripheral equipment: internal CPU caches and external burst mode SRAM caches, hard disk buffers, router

buffers, etc. LCD screens and printers also normally employ static RAM to hold the image displayed (or to be printed). Small SRAM buffers are also found in CDROM and CDRW drives; usually 256 kB or more are used to buffer track data, which is transferred in blocks instead of as single values. The same applies to cable modems and similar equipment connected to computers.

HOBBYISTS

Hobbyists often prefer SRAM due to the ease of interfacing. It is much easier to work with than DRAM as there are no refresh cycles and the address and data buses are directly accessible rather than multiplexed. In addition to buses and power connections, SRAM usually require only three controls: Chip Enable (CE), Write Enable (WE) and Output Enable (OE). In synchronous SRAM, Clock (CLK) is also included.

TYPES OF SRAM

NON-VOLATILE SRAM

Non-volatile SRAMs have standard SRAM functionality, but they save the data when the power supply is lost, ensuring preservation of critical information. nvSRAMs are used in a wide range of situations—networking, aerospace, and medical, among many others—where the preservation of data is critical and where batteries are impractical.

ASYNCHRONOUS SRAM

Asynchronous SRAM are available from 4 Kb to 32 Mb. The fast access time of SRAM makes asynchronous SRAM

appropriate as main memory for small cache-less embedded processors used in everything from industrial electronics and measurement systems to hard disks and networking equipment, among many other applications. They are used in various applications like switches and routers, IP-Phones, IC-Testers, DSLAM Cards, to Automotive Electronics.

BY TRANSISTOR TYPE

- Bipolar junction transistor (used in TTL and ECL) — very fast but consumes a lot of power
- MOSFET (used in CMOS) — low power and very common today

BY FUNCTION

- Asynchronous — independent of clock frequency; data in and data out are controlled by address transition
- Synchronous — all timings are initiated by the clock edge(s). Address, data in and other control signals are associated with the clock signals

BY FEATURE

- ZBT (ZBT stands for zero bus turnaround) — the turnaround is the number of clock cycles it takes to change access to the SRAM from write to read and vice versa. The turnaround for ZBT SRAMs or the latency between read and write cycle is zero.
- syncBurst (syncBurst SRAM or synchronous-burst SRAM) — features synchronous burst write access to the SRAM to increase write operation to the SRAM.

- DDR SRAM — Synchronous, single read/write port, double data rate IO
- Quad Data Rate SRAM — Synchronous, separate read & write ports, quadruple data rate IO

5

Input Output Devices

A computer is only useful when it is able to communicate with the external environment. When you work with the computer you feed your data and instructions through some devices to the computer. These devices are called Input devices. Similarly computer after processing, gives output through other devices called output devices. For a particular application one form of device is more desirable compared to others. We will discuss various types of I/O devices that are used for different types of applications. They are also known as peripheral devices because they surround the CPU and make a communication between computer and the outer world.

INPUT DEVICES

Input devices are necessary to convert our information or data in to a form which can be understood by the computer. A good input device should provide timely, accurate and

useful data to the main memory of the computer for processing followings are the most useful input devices.

Component or peripheral (such as a barcode reader, graphic tablet, keyboard, magnetic-stripe reader, modem, mouse, scanner, stylus) that feeds data or instruction into a computer for display, processing, storage, or outputting or transmission. Input devices convert the user's actions and analog data (sound, graphics, pictures) into digital electronic signals that can be 'handled' or 'read' by a computer. Digital data (such as from barcode readers, modems, scanners, *etc.*) does not require any conversion and is input direct into a computer. It is through input devices that a user exercises control over a computer, its operations, and outputs.

An input device is any device that provides input to a computer. There are dozens of possible input devices, but the two most common ones are a keyboard and mouse. Every key you press on the keyboard and every movement or click you make with the mouse sends a specific input signal to the computer. These commands allow you to open programs, type messages, drag objects, and perform many other functions on your computer. Since the job of a computer is primarily to process input, computers are pretty useless without input devices. Just imagine how much fun you would have using your computer without a keyboard or mouse. Not very much. Therefore, input devices are a vital part of every computer system. While most computers come with a keyboard and mouse, other input devices may also be used to send information to the computer. Some examples include joysticks, MIDI keyboards, microphones, scanners, digital cameras, web cams, card readers, UPC scanners, and scientific measuring

equipment. All these devices send information to the computer and therefore are categorized as input devices.

To be of any use at all a computer has to be able to take input, yet this basic premise can easily escape the modern computer user. With the quality and range of input devices now available, it seems hard to believe that computer input had once to be literally hardwired. New circuits had to be constructed to solve individual problems by arranging cables and jack sockets on vast circuit boards. Clearly at that time word processing was simply beyond imagining, but years of developments have made this seem absurd, and word processing has become a staple of the computer's work. Word processing, of course, relies on perhaps the most basic computer input device: the keyboard. Plainly modelled on the typewriter, in Western countries most modern computer keyboards are based on the QWERTY layout, or closely-related variants such as the French AZERTY layout. There are additional keys not normally found on typewriters such as function keys, a numeric keypad and so on, and even in countries where different alphabets or writing systems are in use, the physical layout of the keys is often quite similar.

The original 1981 IBM PC's keyboard was severely criticized by typists for its non-standard placement of the return and left shift keys. In 1984, IBM corrected this on its AT keyboard, but shortened the backspace key, making it harder to reach. Then, in 1987, it introduced the enhanced keyboard, which relocated all the function keys, the Ctrl keys, and the Esc key to the positions we commonly see them today. In recent years, so-called "Internet keyboards" have also become popular, including extra buttons for specific applications or

functions (typically a browser or email client). Laptops might also have vendor specific keys included in the keyboard.

TYPES OF INPUT DEVICES

KEYBOARD

The keyboard is probably the easiest kind of input device to understand. When the user presses a key, a code is sent to the cpu, which translates the code into some sort of storage format (usually ASCII.) There is generally no way for the cpu to send information back to the keyboard, so it is an input only device. Usually, you buy a keyboard as part of a computer system, but there may be a reason you want a specialized model. Certain keyboards are designed to be more ergonomically safe to prevent Carpal-Tunnel Syndrome.

Fig. Keyboard.

This is the standard input device attached to all computers. The layout of keyboard is just like the traditional typewriter of the type QWERTY. It also contains some extra command keys and function keys. It contains a total of 101 to 104 keys. A typical keyboard used in a computer is shown in Fig below. You have to press correct combination of keys to input data. The computer can recognize the electrical signals corresponding to the correct key combination and processing is done accordingly.

MOUSE

Mouse is an input device shown in Fig below that is used with your personal computer. It rolls on a small ball and has two or three buttons on the top. When you roll the mouse across a flat surface the screen censors the mouse in the direction of mouse movement. The cursor moves very fast with mouse giving you more freedom to work in any direction. It is easier and faster to move through a mouse.

Fig. Mouse.

Many modern computers rely heavily on the mouse. This is a small object, usually with a roller ball on the bottom that can be dragged along the desktop. Mice usually have one or more buttons on them that can be activated with the hand controlling the mouse. As the mouse is moved on the desk, a pointer moves on the screen. This mouse pointer is analogous to the user's hand. Many find this a more natural way of controlling the computer than typing on a keyboard. There are a number of "mouse substitutes" available, but they all basically work in the same way. Laptop computers often have a small ball embedded in the keyboard that can be rolled so the mouse can be activated even when the computer is being held on the user's lap.

SCANNER

Scanners are often though about in the office, but are rarely thought about as an input device for a computer. When you

have a scanner, you can scan newspaper clippings, articles, and pictures. This gives you the ability to save everything onto your computer. There are multiple types of scanners available; some even come on the printer you already have.

The keyboard can input only text through keys provided in it. If we want to input a picture the keyboard cannot do that. Scanner is an optical device that can input any graphical matter and display it back. The common optical scanner devices are Magnetic Ink Character Recognition (MICR), Optical Mark Reader (OMR) and Optical Character Reader (OCR).

MAGNETIC INK CHARACTER RECOGNITION (MICR)

This is widely used by banks to process large volumes of cheques and drafts. Cheques are put inside the MICR. As they enter the reading unit the cheques pass through the magnetic field which causes the read head to recognise the character of the cheques.

OPTICAL MARK READER (OMR)

This technique is used when students have appeared in objective type tests and they had to mark their answer by darkening a square or circular space by pencil. These answer sheets are directly fed to a computer for grading where OMR is used.

OPTICAL CHARACTER RECOGNITION (OCR)

This technique unites the direct reading of any printed character. Suppose you have a set of hand written characters on a piece of paper. You put it inside the scanner of the

computer. This pattern is compared with a site of patterns stored inside the computer. Whichever pattern is matched is called a character read. Patterns that cannot be identified are rejected. OCRs are expensive though better the MICR.

CD-ROM

The same kinds of technology that enable us to store music digitally on compact disks allows us to store other kinds of information on the same medium. Compact Disks essentially store numbers. They are inexpensive to create, and can hold large amounts of information. (600 Mb) CD-ROMs are frequently used to sell software which has become too large to fit on floppy disks. Unfortunately, a CD - ROM cannot be written to with typical home technology. The ROM part of CD-ROM refers to this characteristic. A CD - ROM is an input device because it can send information to the CPU, but the CPU cannot send information to it.

CD Drives are measured in comparison to the speed of music CDs. An audio CD always runs at a constant speed. A 2X CD drive is twice the speed of an audio CD player. 6X and 8x drives are available at this writing. Even these drives are not as fast as typical hard drives. Erasable CD drives are becoming popular, because they hold as much information as a regular CD, but the user can store things to them as well as reading from them. Such devices will probably be much more prevalent as their price comes down and reliability improves.

AUDIO/VIDEO INPUT

Web cams and digital cameras can also be considered input devices. They provide visual data to the computer in the form

of images and video. Some web cams can even be used as pointing devices by tracking the location of a person's hands or face. Microphones and digital musical instruments, such as midi keyboards, are audio input devices that provide the computer with audio data. Even an electric guitar, when hooked up to a computer, can be an input device.

MICROPHONE

A microphone is a great input device to add to your computer, simply because it gives you greater capabilities when calling someone online, or trying to use a video conference. By purchasing a microphone, anyone you wish will be able to hear you. You may also record things to your computer with the microphone, giving you multiple functions with one input device.

WEBCAM

Webcams help to round out the video conferencing experience. Webcams rarely actually come with desktop computers (but may be built into some laptops). Purchasing a webcam can be a great investment, however, for anyone who is looking to have a video conference, or to simply see someone as you talk with them. Webcams are available at low prices, as well as high prices, depending on the quality you expect.

OUTPUT DEVICES

Any device that outputs information from a computer is called, not surprisingly, an output device. Since most information from a computer is output in either a visual or auditory format, the most common output devices are the

monitor and speakers. These two devices provide instant feedback to the user's input, such as displaying characters as they are typed or playing a song selected from a playlist. While monitors and speakers are the most common output devices, there are many others. Some examples include headphones, printers, projectors, lighting control systems, audio recording devices, and robotic machines. A computer without an output device connected to it is pretty useless, since the output is what we interact with. Anyone who has ever had a monitor or printer stop working knows just how true this is.

VISUAL DISPLAY UNIT

The most popular input/output device is the Visual Display Unit (VDU). It is also called the monitor. A Keyboard is used to input data and Monitor is used to display the input data and to receive massages from the computer.

A monitor has its own box which is separated from the main computer system and is connected to the computer by cable. In some systems it is compact with the system unit. It can be *color* or *monochrome*.

Fig. Desktop monitor.

Fig. Liquid crystal display.

TERMINALS

It is a very popular interactive input-output unit. It can be divided into two types: hard copy terminals and *soft copy* terminals. A *hard copy* terminal provides a printout on paper whereas soft copy terminals provide visual copy on monitor. A terminal when connected to a CPU sends instructions directly to the computer. Terminals are also classified as dumb terminals or intelligent terminals depending upon the work situation.

PRINTER

Fig. Laser Printer.

It is an important output device which can be used to get a printed copy of the processed text or result on paper. There

are different types of printers that are designed for different types of applications. Depending on their speed and approach of printing, printers are classified as *impact* and *non-impact* printers. Impact printers use the familiar typewriter approach of hammering a typeface against the paper and inked ribbon.

Dot-matrix printers are of this type. Non-impact printers do not hit or impact a ribbon to print. They use electro-static chemicals and ink-jet technologies. *Laser printers* and *Ink-jet printers* are of this type. This type of printers can produce color printing and elaborate graphics.

INPUT AND OUTPUT DEVICES

INTRODUCTION

Is a system, consisting of many components. Some of those components, like Windows XP, and all your other programs, are software. The stuff you can actually see and touch, and would likely break if you threw it out a fifth-story window, is hardware. Not everybody has exactly the same hardware. But those of you who have a desktop system, like the example shown in Figure, probably have most of the components shown in that same figure. Those of you with notebook computers probably have most of the same components. Only in your case the components are all integrated into a single book-sized portable unit. The computer is really of less use until it is able to communicate with the outside world. The third most important element of a computer after memory and the CPU is the input and output device. The various input/output devices are required for users to communicate with the computer for bringing in information and pushing information out of a system.

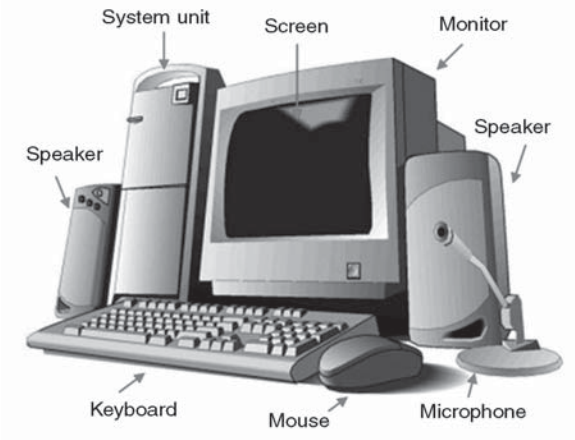


Fig. Input and Output devices.

COMMON DEVICES

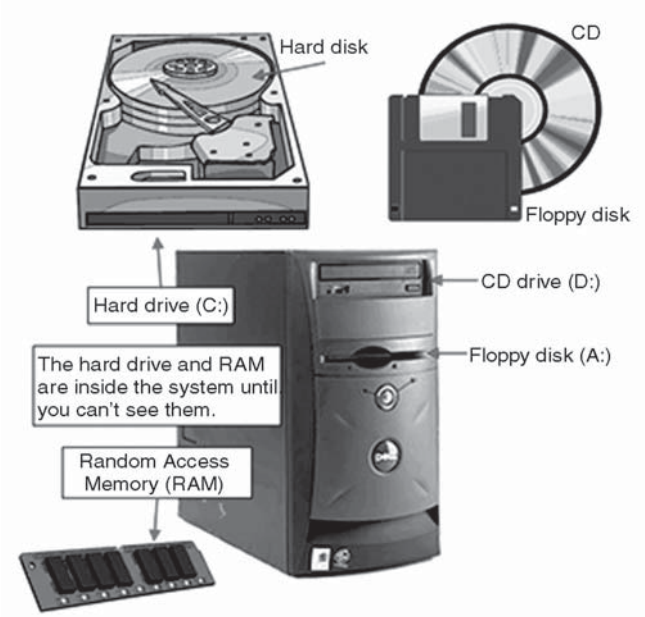
INPUT DEVICES

1. Keyboard
2. Mouse
3. Joystick
4. Scanner
5. Light Pen
6. Touch Screen.

OUTPUT DEVICES

1. Monitor
2 LCD
3. Printer
4. Plotter.

The input and output devices substantially differ in many characteristics. One of the factor among input and output devices is their data processing rates, also referred to as the average number of characters that can be processed by a device per second.



- Keyboard: It is a text based input device that allows the user to input alphabets, number and other characters. It consists of a set of keys mounted on a board.
- Mouse: The mouse is a small device used to point to a specific pixel on the screen and select single or multiple pixels in order to perform one or more actions. It can be used to select menu command, size windows, start programs *etc.*
- Joystick: It is a vertical stick which moves the graphic cursor in a direction in which the stick is moved. It has a button on the top used to select options pointed by the cursor. It is mainly used for computer games robotic applications *etc.*
- Scanner: It is an input device used for direct data entry from the source document to the computer system. It mainly converts a document image into digital form to store it in the memory devices. It mainly reduces error from direct data entry options.

- Bar code Scanners: Bar code is a set of lines of different thickness representing a number. A beam is shined on the lines that make up the code and it detects the amount of light reflected back.
- Light pen: It is a pen shaped device used to select objects on a display screen. Very similar to a mouse but uses a light pen to move the pointer and select objects on screen.
- Touch Screen: It allows the user to select application icons by touching the screen.
- Digital camera: It stores pictures in large quantities. The pictures are stored in memory card which can be later transferred to a computer.
- Speech input device: The microphone is a speech input device which can be operated using a talking software. We need to add a sound card for attaching this device.

When we deal with an Input/Output module in a computer system there are many things that we need to keep in mind. We do not generally connect an external device into a bus structure of a computer. A wide variety of devices require different logical interfaces as it is not a logical thing to expect that the CPU realizes every kind of a device with different data rates and really different data representations.

DESCRIPTION ABOUT DEVICES

OTHER INPUT-ONLY DEVICES

There are other devices available for input, but they tend to be specialty devices. Some examples are:

- Touch - sensitive screens for mall kiosks.
- Adaptive keyboards for individuals with physical disabilities
- Infrared mice for a mouse without cords
- Special presentation devices that allow a speaker to send input to the computer while delivering a presentation away from the keyboard.

OUTPUT DEVICES

Any device that outputs information from a computer is called, not surprisingly, an output device. Since most information from a computer is output in either a visual or auditory format, the most common output devices are the monitor and speakers. These two devices provide instant feedback to the user's input, such as displaying characters as they are typed or playing a song selected from a playlist. While monitors and speakers are the most common output devices, there are many others. Some examples include headphones, printers, projectors, lighting control systems, audio recording devices, and robotic machines. A computer without an output device connected to it is pretty useless, since the output is what we interact with. Anyone who has ever had a monitor or printer stop working knows just how true this is.

PRINTER

Printers take information from the CPU and transfer it to paper. There are a number of different printer technologies available.

Dot Matrix:

Fig. Dot Matrix.

Dot Matrix printers are inexpensive and reliable, but they are loud and slow. They do not have nearly the print quality of some of the other types of printers.

Ink Jet:

Fig. Ink Jet

Ink Jet printers squirt small streams of ink onto the paper. They tend to be slightly more expensive than dot matrix printers, but the quiet operation and improved print quality make them very popular with buyers of home computers. Some ink jet printers can make color prints. These can be very entertaining, but the ink becomes expensive.

Laser Printers: Laser Printers use a combination of laser and copying technology to make very clear copies. Laser printers tend to make clearer copies than the other types of

printers, and operate much more quickly, but they can be quite expensive to purchase and maintain.

Fig. Laser Printers.

Many people try to save money on a printer purchase and are disappointed. If the main reason you will use your computer is to type letters, remember that the people you write to will not see your monitor. What they will see is only what comes out of your printer. If printed documents are an important part of what you will use a computer for, consider a higher quality printer. Ink Jets are probably most appropriate for home use, and laser printers are more popular in an office setting. A dot matrix machine is fine for test printing or use on a kid's computer, but you will be disappointed with the results if you try to use it for business correspondence. Before buying an ink jet, find out how much the ink cartridges cost, and how long they tend to last. With a laser printer, be sure to check on the cost of toner cartridges.

MONITORS

The monitor is the part of the system that you look at most of the time. Monitors resemble televisions. Most computer monitors use the same technology as televisions,

but with much higher resolution. Often the monitor will come packaged with a computer system, but you may wish to upgrade.

SIZE

The size of a monitor can make a big impact. You might get a headache squinting at a screen that is too small. The size of a monitor is measured in diagonal inches. A 15 inch monitor is 15 inches diagonally from corner to corner of the screen.

Dot Pitch and DPI: Monitors are also measured by their precision. There are two main measures, Dot Pitch, and DPI. Dot pitch is a measure of the size of each tiny dot the monitor can display. The smaller each dot is, the nicer the picture will look, but the more expensive the monitor will be. When discussing Dot Pitch, SMALLER IS BETTER!! DPI stands for Dots Per Inch. As you can guess, the smaller the dot pitch rating for a monitor is, the more of those tiny dots you could squeeze into a square inch. If you are considering DPI, LARGER IS BETTER. Computer sales people are not above taking advantage of this confusion.

Video Controller Card: Dealing with graphics takes a lot of work. Modern computers usually have a separate computer built in just to help with controlling the monitor. This little computer has its own cpu and memory! The power and speed of this little computer, as well as its memory capacity, have a huge effect on how graphics are drawn to your screen. This little computer is referred to as a graphics controller card. The most common cards now are called SVGA. Of course it gets way more complicated than this, but all you have to

know is that there are fancier cards that do more and cost more, but you may not need the fanciest one out there for your first computer.

PROJECTORS

A hardware device that enables an image, such as a computer screen, to be projected onto a flat surface. These devices are commonly used in meetings and presentations as they allow for a large image to be shown so everyone in a room can see. A general diagram of projector as an example image of what a projector may look like is shown below. As can be seen in this image the projector is a small device often a little bigger than a toaster and typically weighs a few pounds.

Fig. Projectors.

SPEAKERS

A hardware device connected to a computer's sound card that outputs sounds generated by the card. Below is a graphic image example of the general speakers with subwoofer; speakers like the ones shown below are an example of what most computer speakers resemble.

Fig. Speakers.

PLOTTERS

A plotter can be used to produce very large drawings on paper sizes up to A0 (16 times as big as A4). A plotter draws onto the paper using very fine pens. There are two types of plotter. They differ in the way that the pen can be moved about on the piece of paper to draw lines:

Flatbed Plotter: The paper is fixed and the pen moves left and right and up and down across the paper to draw lines.

Drum Plotter: The pen moves up and down on the paper and the paper is moved left and right by rotating a drum on which the paper is placed.

Fig. Flatbed Plotter.

Fig. Drum Plotter.

Plotters can automatically change their pens and so can produce colour output. The lines drawn by a plotter are continuous and very accurate. Plotters are very slow but produce high quality output. They are usually used for Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) applications such as printing out plans for houses or car parts. The quality of the output produced by ink jet printers is now very good and large format (big) ink-jet printers are steadily replacing plotters for most tasks.

COMMON INPUT OUTPUT DEVICES

RAM

RAM is perhaps the most important of the input/output devices. When we talk about computer memory, we are mainly talking about RAM. In this class, when we think about the banks of light switches that can be manipulated, we are thinking of RAM memory. The term Random Access is pretty unfortunate. There is nothing random about how memory is accessed. The program running will determine what is in

memory. (of course, the program itself is in memory too!) RAM can be read from by the CPU. This means that the CPU can 'look' at any address in RAM and get the contents of that address. It can also be written to by the CPU, meaning that the CPU can change the value of memory cells on the fly.

One VERY important aspect of RAM memory is that it requires power. RAM can only hold values while power is going through it. If the power is interrupted, the RAM will lose ALL the values in it. This is why it is so important to save your work frequently when working on a computer. RAM memory is volatile. When the room you are working on is hit by a tsunami and the power goes out of your computer, you lose everything that was in RAM. This could be bad. The answer is to make copies of RAM, and place these copies on other kinds of media. That's what disk drives are for.

The amount of RAM in your computer is obviously a pretty important factor. The more memory you have, the more 'room' there is in your computer for information and programs. Modern programs have gotten HUGE, and the kinds of information you can work with have gotten much larger. Some early home computers had 4 or 16 K of RAM. The original IBM PC had 640 K of RAM. In its day, this was thought of as an extravagant amount of memory that would never be fully used. Modern computers with less than 16 Megabytes (16,000 K) are considered a bit lame. (pun completely intended.) Older computers can be quite happy with much less RAM, but they will not be able to run the newer programs.

If you are buying or upgrading a computer, you should seriously consider as much RAM as you can afford. There is probably no more cost-effective upgrade than RAM. If you

find that you need more memory, It is relatively cheap and easy to do a memory upgrade. Note that not all types of RAM are interchangeable.

DISK DRIVES

Disk drives are special devices that allow us to make copies of parts of RAM and store them magnetically. If RAM memory is electronic, think of disk drives as a special kind of magnetic memory. When you save something to a disk, the electronic impulses in RAM are copied and stored to the disk as a series of magnetic impulses. All a disk drive does is translating electronic impulses and magnetic impulses back and forth.

Disk drives are handy because magnetic impulses are more permanent than electronic ones. The disk drive does not require electricity to keep values in memory, so if you store something to a disk, the information will be there when the tidal wave knocks the power out to your computer. (Assuming, of course, that the disk stayed dry and clean) Disks are sometimes thought of as secondary storage for this reason.

There are a number of different kinds of disks. We will discuss a couple of main types:

Floppy Drives: A floppy drive is a machine that is designed to read floppy disks. Floppy disks are the removable devices that you stick in slots in the front of the machine. I know, they look square, not disk-shaped, and they don't look floppy at all, but they are indeed floppy disks. Floppy disks come in hard plastic cases to make them a little more sturdy and easier to handle, but inside the plastic case, there is an actual disk. It is made of a plastic-like substance called mylar which really is floppy. Floppy disks come in two main sizes; 5 1/4 inch, and 3 1/2 inch. The 5 1/4 disks are becoming obsolete,

but you still see them from time to time. The 3 1/2 inch disks, although physically smaller, can usually hold more information!

There are different flavors of floppy disks. In a modern computer, the only kind of floppy you need to purchase is High Density 1.44 MB. As you can guess, they hold 1.44 MegaBytes of information, which is a pretty good amount. (You could hold several hundred pages of text or about a dozen full color pictures on a 1.44 MB disk) This type of disk may carry some other markings as well, such as DSHD or HD. The HD is the important part. That tells you it can handle 1.44 MB. If you have an older computer, you may find that it needs a different type or size of disks. Check your documentation to be sure.

Hard Drives: A hard drive is a special disk that is usually mounted permanently inside your computer's cabinet. You rarely see the hard drive, and almost never take it out. Hard drives are made of different material than floppies, and they are physically hard (although if you touched the actual hard surface, you would destroy it!) They spin much more quickly than floppies, and require much more precision. They are sealed inside a special case, and that is sealed inside the computer case. A hard drive has a much larger capacity than a floppy, and is much faster at saving and retrieving information.

Modern computers frequently have hard drives with 500 MB or more of capacity. As this capacity grows, people are beginning to measure it in terms of gigabytes. Software programs are always becoming larger and taking more room on hard drives. It never takes long to completely fill up the

capacity of a drive. If you can afford a large drive when you buy the computer, you won't be sorry, but you can usually add another drive or upgrade later.

Fancier kinds of Drives: There are a number of other types of drives you may encounter when buying or upgrading a computer. You may encounter such things as Bernoulli tape drives, WORM (write once, read many) drives, optical floppies, and removable hard drives. All these things are really cool, but you may want to stick with the basics until you are a little more comfortable with the technology. Any computer system ought to have at least one 3.5 inch floppy drive, as large a hard drive as possible, and a CD-ROM drive.

Drive Controller Cards: Most computers come with a small card installed that helps control all your disk drives. It is called the Drive Controller Card. There are two major kinds, IDE and SCSI. The only time you will ever care about this is when you buy a hard drive. Just know that the terms IDE and SCSI are terms that describe the drive controller card.

NETWORK CARD

In a home computer, you will generally not have a network card, but these devices are very common for computers in office settings. A network card is a special card which allows your computer to talk to other computers that are physically attached via cables. Depending on how the network is set up, you can send messages from computer to computer, run programs that are stored on different computers, and share devices like hard drives and printers. In many offices, the network also gives you access to the Internet. Network connections are generally faster than modem connections.

MODEM

Modems were once thought of as somewhat extravagant, but with the advent of the Internet, they are becoming a necessity for home computers. They allow you to connect your computer to the Internet or other computer systems through a telephone connection. The term Modem stands for Modulator/Demodulator. It is a device that converts back and forth from the digital signals that computers understand to analog signals (sounds) that can be transferred over telephone lines. Modems can be internal or external. The external ones have their own little case and power supply, and are generally a little more expensive than the internal ones, which are little cards that fit inside the computer. Modems are rated by their transmission speed, which is measured in BAUD (Bits of Audio Data/Second). If you want to do any Internet connections, you need a baud rate of 1440 BAUD (also sometimes called 14400 BPS or 14.4 KBPS) You can also purchase faster modems, but they are of course more expensive. In the near future, something will happen in this arena. It is likely that we will be switching to a completely new kind of communication technology for home computers, but nobody knows exactly what that will be just yet. Some modems also include faxing and voice mail capabilities. These features can be very convenient for people with home offices.

SOUND CARD

Sound cards are another peripheral device that was once thought to be extravagant, but is now pretty much standard equipment on any new machine. A sound card is a device

which enables the computer to handle sounds (duh!). It can handle sounds in two major ways. It can create sound effects entirely through programming, or it can record sounds through a standard microphone. Sound cards are attached to speakers which recreate the sound.

Obviously, sound cards are a big boost to computer game players, but they are frequently being used for more serious pursuits as well. People can attach voice annotations to documents they have written, musicians can test and modify a composition with a computer, and people with visual disabilities can have screens of text read to them. The basic sound card is referred to as an 8-bit sound card, because it processes 8 bits of information at a time. You can now find 16-bit, and even 32-bit sound cards if you are willing to pay for them. As always, it depends on what you want your computer to do for you.

THE INPUT/OUTPUT INTERFACE

Input and output in IF/Prolog take place using device drivers which are managed on the basis of a driver model. A device driver is responsible for operations for a particular stream type, *e.g.,* files, character strings or pipes. Prolog can manage different streams with the same set of generic built-in predicates.

Prolog provides the following built-in devices:

- File is the device for operating system files
- Null is the “bit bucket” device
- Pipe is the device for operating system pipes
- Standard is the device for the standard streams of the Prolog system

- String is the device for input/output using character strings
- Socket is the device for interprocess communication.

The I/O-related COM interfaces which are defined by header files in the oskit/io directory. Most of these interfaces are fairly generic and can be used in a wide variety of situations. Some of these interfaces, such as the bufio interface, are extensions to other more primitive interfaces, and allow objects to export the same functionality in different forms, permitting clients to select the service that most directly meets their needs thereby reducing interface crossing overhead and increasing overall performance.

OSKIT_ABSIO: ABSOLUTE I/O INTERFACE

The `oskit_absio` interface supports reading from and writing to objects at specified absolute offsets, with no concept of a seek pointer. The `oskit_absio` interface is identical to the oskit_blkio COM interface, except that the block size is always one, since absolute IO is byte-oriented. In fact, an object that supports byte-granularity reads and writes can easily export both oskit_blkio and `oskit_absio` using exactly the same function pointer table, simply by implementing an `oskit_blkio` interface that always returns one from getblocksize, and then returning a pointer to that interface on queries for either oskit_blkio or oskit_absio.

The `oskit_absio` COM interface inherits from I Unknown, and has the following additional methods:

`read`:

Read from this object, starting at the specified offset.

`write`:

Write to this object, starting at the specified offset.

`getsize`:

Get the current size of this object.

`setsize`:

Set the current size of this object.

READ FROM THIS OBJECT, STARTING AT SPECIFIED OFFSET

SYNOPSIS

```
#include <oskit/io/absio.h>
```

OSKIT_COMDECL oskit_absio_read `(oskit_absio_t *f, void *buf, oskit_off_t offset, oskit_size_t amount, [out] oskit_size_t *out_actual);`

DESCRIPTION

This method reads no more than amount bytes into buf from this object, starting at `offset`. `out_actual` is set to the actual number of bytes read.

PARAMETERS

`f`:

The object from which to read.

`buf`:

The buffer into which the data is to be copied.

`offset`:

The offset in this object at which to start reading.

`amount`:

The maximum number of bytes to read.

`out_actual`:

The actual number of bytes read.

RETURNS

Returns 0 on success, or an error code specified in `<oskit/error.h>`, on error.

WRITE TO THIS OBJECT, STARTING AT SPECIFIED OFFSET

SYNOPSIS

```
#include <oskit/io/absio.h>
```

OSKIT_COMDECL oskit_absio_write(oskit_absio_t *f, const void *buf, oskit_off_t offset, oskit_size_t amount, [out] oskit_size_t *out_actual);

DESCRIPTION

This method writes no more than amount bytes from buf into this object, starting at `offset`. `out_actual` is set to the actual number of bytes written.

PARAMETERS

f:

The object to which to write.

buf:

The buffer from which the data is to be copied.

***offset*:**

The offset in this object at which to start writing.

***amount*:**

The maximum number of bytes to write.

***out_actual*:**

The actual number of bytes written.

RETURNS

Returns 0 on success, or an error code specified in `<oskit/error.h>`, on error.

GET THE SIZE OF THIS OBJECT

SYNOPSIS

```
#include <oskit/io/absio.h>
```

OSKIT_COMDECL oskit_absio_getsize`(oskit_absio_t *f, [out] oskit_off_t *out_size);`

DESCRIPTION

This method returns the current size of this object in bytes.

PARAMETERS

f:

The object whose size is desired.

out_size:

The current size in bytes of this object.

RETURNS

Returns 0 on success, or an error code specified in <oskit/error.h>, on error.

SET THE SIZE OF THIS OBJECT

SYNOPSIS

```
#include <oskit/io/absio.h>
```

OSKIT_COMDECL oskit_absio_setsize `(oskit_absio_t *f, oskit_off_t new_size);`

DESCRIPTION

This method sets the size of this object to new_size bytes. If new_size is larger than the former size of this object, then the contents of the object between its former end and its new end are undefined.

Note that some absolute I/O objects may be fixed-size, such as objects representing preallocated memory buffers; in such cases, this method will always return `OSKIT_E_NOTIMPL`.

PARAMETERS

f:

The object whose size is to be changed.

***new_size*:**

The new size in bytes for this object.

RETURNS

Returns 0 on success, or an error code specified in `<oskit/error.h>`, on error.

OSKIT_ASYNCIO: ASYNCHRONOUS I/O INTERFACE

The `oskit_asyncio` interface provides interfaces in support of basic asynchronous I/O, based on registered callback objects. This can be used, for example, to implement Unix SIGIO or select or POSIX.1b aio.

This interface supports a notion of three kinds of interesting events: readability, writeability, and "other" exceptional conditions. These are defined via the flags: OSKIT_ASYNCIO_READABLE, OSKIT_ASYNCIO_WRITEABLE, and OSKIT_ASYNCIO_EXCEPTION which are passed and returned in a mask in the various methods.

The `oskit_asyncio` COM interface inherits from IUnknown, and has the following additional methods:

`poll`:

Poll for currently pending asynchronous I/O conditions.

`add_listener`:

Add a callback object for async I/O events.

`remove_listener`:

Remove a previously registered callback object.

`readable`:

Returns the number of bytes that can be read.

POLL FOR PENDING ASYNCHRONOUS I/O CONDITIONS ON THIS OBJECT

SYNOPSIS

```
#include <oskit/io/asyncio.h>
```

OSKIT_COMDECL oskit_asyncio_poll`(oskit_asyncio_t *`*`io`*`);`

DESCRIPTION

Poll for currently pending asynchronous I/O conditions, returning a mask indicating which conditions are currently present.

PARAMETERS

`io`:

The async I/O object.

RETURNS

If successful, returns a mask of the OSKIT_ASYNCIO flags above. Otherwise, returns an error code specified in `<oskit/error.h>`.

ASSOCIATE A CALLBACK WITH THIS OBJECT

SYNOPSIS

```
#include <oskit/io/asyncio.h>
```

OSKIT_COMDECL oskit_asyncio_add_listener (`oskit_asyncio_t *io, oskit_listener_t *l, oskit_s32_t mask);`

DESCRIPTION

Add a callback listener object to handle asynchronous I/O events. When an event of interest occurs on this I/O object (*i.e.*, when one of the one to three I/O conditions becomes true), *all* registered listeners will be called.

The mask parameter is an OR'ed combination of the `OSKIT_ASYNCIO` flags above. It specifies which events the listener is interested in. Note that spurious notifications are possible, the listener must use `oskit_asyncio_poll` to determine the actual state of affairs. Also, if successful, this method returns a mask describing which of the OSKIT_ASYNCIO conditions are already true, which the caller must check in order to avoid missing events that occur just before the listener is registered.

PARAMETRS

io:

The async I/O object.

l:

The `oskit_listener` object to call.

mask:

A mask of flags indicating which events are of interest.

RETURNS

If successful, returns a mask of the OSKIT_ASYNCIO currently pending. Otherwise, returns an error code specified in `<oskit/error.h>`.

DISASSOCIATE A CALLBACK FROM THIS OBJECT

SYNOPSIS

```
#include <oskit/io/asyncio.h>
```

OSKIT_COMDECLoskit_asyncio_remove_listener`(oskit_asyncio_t *io, oskit_listener_t *l);`

DESCRIPTION

Remove a previously registered listener callback object. Returns an error if the specified callback has not been registered.

PARAMETERS

io:

The async I/O object.

l:

The oskit_listener object to call.

RETURNS

Returns 0 on success, or an error code specified in `<oskit/error.h>`, on error.

RETURN NUMBER OF BYTES AVAILABLE FOR READING FROM OBJECT

SYNOPSIS

```
#include <oskit/io/asyncio.h>
```

OSKIT_COMDECL oskit_asyncio_readable(oskit_asyncio_t *io);

DESCRIPTION

Returns the number of bytes that can be read from the I/O object.

PARAMETRS

io:

The async I/O object.

RETURNS

The number of bytes available for reading.

OSKIT_BLKIO: BLOCK I/O INTERFACE

The oskit_blkio interface supports reading and writing of raw data in units of fixed-sized blocks which are some power of two.

This interface is identical to the oskit_absio interface except for the addition of a getblocksize method; in fact, an object that supports byte-granularity reads and writes can easily export both oskit_blkio and oskit_absio using exactly the same function pointer table, simply by implementing an oskit_blkio interface that always returns one from getblocksize, and then returning a pointer to that interface on queries for either oskit_blkio or oskit_absio.

The **oskit_blkio** interface inherits from IUnknown, and has the following additional methods:

getblocksize:

Return the minimum block size of this block I/O object.

read:

Read from this object, starting at the specified offset.

write:

Write to this object, starting at the specified offset.

getsize:

Get the current size of this object.

setsize:

Set the current size of this object.

RETURN THE MINIMUM BLOCK SIZE OF THIS BLOCK I/O OBJECT

SYNOPSIS

```
#include <oskit/io/blkio.h>
OSKIT_COMDECL_U oskit_blkio_getblocksize(oskit_blkio_t *f);
```

DESCRIPTION

This method simply returns the block size of the object, which must be a power of two. Calls by the client to read from or write to the object must only use offsets and sizes that are evenly divisible by this block size.

PARAMETERS

f:

The block I/O object.

RETURNS

Returns the block size of the object.

OSKIT_BUFIO: BUFFER-BASED I/O INTERFACE

The **oskit_bufio** interface extends the **oskit_absio** interface, providing additional alternative methods of accessing the object's data. In particular, for objects whose data is stored in an in-memory buffer of some kind, this interface allows

clients to obtain direct access to the buffer itself so that they can read and write data using loads and stores, rather than having to copy data into and out of the buffer using the read and write methods. In addition, this interface provides similar methods to allow clients to "wire" the buffer's contents to physical memory, enabling DMA-based hardware devices to access the buffer directly. However, note that only the read/write methods, inherited from oskit_absio, are mandatory; the others may consistently fail with OSKIT_E_NOTIMPL if they cannot be implemented efficiently in a particular situation. In that case, the caller must use the basic read and write methods instead to copy the data. In other words, oskit_bufio object implementations are not *required* to implement direct buffer access, either software- or DMA-based; the purpose of this interface is merely to allow them to provide this optional functionality easily and consistently.

In general, the map and wire methods should only be implemented if they can be done more efficiently than simply copying the data. Further, even if a buffer I/O implementation does implement map and/or wire it may allow only one mapping or wiring to be in effect at once, failing if the client attempts to map or wire the buffer a second time before the first mapping is undone. Similarly, on some buffer I/O implementations, these operations may only work on certain areas of the buffer or only when the request has certain size or alignment properties: for example, a buffer object that stores data in discontiguous segments, such as BSD's mbuf system, may only allow a buffer to be mapped if the requested region happens to fall entirely within one segment. Thus, the client of a bufio object should call the map or wire methods

whenever it can take advantage of direct buffer access, but must always be prepared to fall back to the basic copying methods. A particular buffer object may be semantically read-only or write-only; it is assumed that parties passing bufio objects around will agree upon this as part of their protocols. For a read-only buffer, the write method may or may not fail, and a mapping established using the map method may or may not actually be a read-only memory mapping; it is the client's responsibility not to attempt to write to the buffer. Similarly, for a write-only buffer, the read method may or may not fail; it is the client's responsibility not to attempt to read from the buffer.

The `oskit_bufio` interface extends the `oskit_absio` interface with the following additional methods:

`map:`
: Map some or all of this buffer into locally accessible memory.

`unmap:`
: Release a previously mapped region of this buffer.

`wire:`
: Wire a region of this buffer into contiguous physical memory.

`unwire:`
: Unwire a previously wired region of this buffer.

`copy:`
: Create a copy of the specified portion of this buffer.

MAP SOME OR ALL OF THIS BUFFER INTO LOCALLY ACCESSIBLE MEMORY

SYNOPSIS

```
#include <oskit/io/bufio.h>
```

OSKIT_COMDECL map (oskit_bufio_t **io*, [out] void ***addr*, oskit_off_t *offset*, oskit_size_t *amount*);

DESRIPTION

This method attempts to map some or all of this buffer into memory directly accessible to the client, so that the client can access it using loads and stores. The operation may or may not succeed, depending on the parameters and the implementation of the object; if it fails, the client must be prepared to fall back to the basic read and write methods. If the mapping operation succeeds, the pointer returned is not guaranteed to have any particular alignment.

If a call to the map method requests only a subset of the buffer to be mapped, the object may actually map more than the requested amount; however, since no information is passed back indicating how much of the buffer was actually mapped, the client must only attempt to access the region it requested.

Note that this method does not necessarily twiddle with virtual memory, as its name may seem to imply; in fact in most cases in which it is implemented at all, it just returns a pointer to a buffer if the data is already in locally-accessible memory.

PARAMETRS

io:

The object whose contents are to be mapped.

addr:

On success, the method returns in this parameter the address at which the client can directly access the requested buffer region.

offset:

The offset into the buffer of the region to be mapped.

size:

The size of the region to be mapped.

RETURNS

Returns 0 on success, or an error code specified in <oskit/error.h>, on error.

RELEASE A PREVIOUSLY MAPPED REGION OF THIS BUFFER

SYNOPSIS

```
#include <oskit/io/bufio.h>
```

OSKIT_COMDECL unmap(oskit_bufio_t **io*, void **addr*, oskit_off_t *offset*, oskit_size_t *amount*);

DESCRIPTION

After a successful call to the map method, the client should call this method after it is finished accessing the buffer directly, so that the buffer object can clean up and free any resources that might be associated with the mapping.

The *addr* parameter passed to this method must be exactly the value returned by the map request, and the *offset* and *amount* parameters must be exactly the same as the values previously passed in the corresponding map call. In other words, clients must only attempt to unmap whole regions; they must not attempt to unmap only part of a region, or to unmap two previously mapped regions in one call, even if the two regions appear to be contiguous in memory.

PARAMETRS

io:

The object whose contents are to be mapped.

addr:

The address of the mapped region, as returned from the corresponding map call.

offset:

The offset into the buffer of the mapped region, as passed to the corresponding map call.

size:

The size of the mapped region, as passed to the corresponding map call.

RETURNS

Returns 0 on success, or an error code specified in `<oskit/error.h>`, on error.

WIRE A REGION OF THIS BUFFER INTO CONTIGUOUS PHYSICAL MEMORY

SYNOPSIS

```
#include <oskit/io/bufio.h>
```

OSKIT_COMDECL wire(oskit_bufio_t **io*, [out] oskit_addr_t **phys_addr*, oskit_off_t *offset*, oskit_size_t *amount*);

DESCRIPTION

This method attempts to wire down some or all of this buffer into memory directly accessible by DMA hardware. The operation may or may not succeed, depending on the parameters and the implementation of the object; if it fails, the client must be prepared to fall back to the basic read and write methods. If the wiring operation succeeds, the physical address of the buffer is guaranteed not to change or otherwise become invalid until the region is unwired or the bufio object is released. The wired buffer is not guaranteed to have any particular alignment or location properties: for example, on a PC, if the device that is going to be accessing the buffer requires memory below 16MB, then it must be

prepared to use appropriate bounce buffers if the wired buffer turns out to be above 16MB. If a call to the wire method requests only a subset of the buffer to be mapped, the object may actually wire more than the requested amount; however, since no information is passed back indicating how much of the buffer was actually wired, the client must only attempt to use the region it requested.

PARAMETRS

io:

The object whose contents are to be wired.

addr:

On success, the method returns in this parameter the physical address at which DMA hardware can directly access the requested buffer region.

offset:

The offset into the buffer of the region to be wired.

size:

The size of the region to be wired.

RETURNS

Returns 0 on success, or an error code specified in <oskit/error.h>, on error.

UNWIRE A PREVIOUSLY WIRED REGION OF THIS BUFFER

SYNOPSIS

```
#include <oskit/io/bufio.h>
```

OSKIT_COMDECL unwire (oskit_bufio_t **io*, void **addr*, oskit_off_t *offset*, oskit_size_t *amount*);

DESCRIPTION

After a successful call to the wire method, the client should call this method after the hardware is finished accessing the buffer directly, so that the buffer object can clean up and free any resources that might be associated with the wiring.

The *addr* parameter passed to this method must be exactly the value returned by the wire request, and the *offset* and *amount* parameters must be exactly the same as the values previously passed in the corresponding wire call. In other words, clients must only attempt to unwire whole regions; they must not attempt to unwire only part of a region, or to unwire two previously wired regions in one call, even if the two regions appear to be contiguous in physical memory.

PARAMETRS

io:

The object whose contents are to be wired.

addr:

The address of the wired region, as returned from the corresponding map call.

offset:

The offset into the buffer of the wired region, as passed to the corresponding wire call.

size:

The size of the wired region, as passed to the corresponding wire call.

RETURNS

Returns 0 on success, or an error code specified in <oskit/error.h>, on error.

CREATE A COPY OF THE SPECIFIED PORTION OF THIS BUFFER

SYNOPSIS

```
#include <oskit/io/bufio.h>
```

OSKIT_COMDECL copy(oskit_bufio_t **io*, oskit_off_t *offset*, oskit_size_t *amount*, [out] oskit_bufio_t ***out_io*);

DESCRIPTION

This method attempts to create a logical copy of a portion of this buffer object (possibly the whole buffer), returning a new oskit_bufio object representing the copy. As with the map and wire methods, this method should only be implemented by an object if it can be done more efficiently than a simple "brute-force" copy using read. For example, in virtual memory environments, the object may be able to use copy-on-write optimizations.

Similarly, if the buffer's contents are stored in special memory not efficiently accessible to the processor, such as memory on a video or coprocessor board, this method could use on-board hardware to perform a much faster copy.

PARAMETERS

io:

The object whose contents are to be copied.

offset:

The offset into the buffer of the region to be copied.

size:

The size of the region to be copied.

out_io:

On success, this parameter holds the bufio object representing the newly created copy of the buffer's contents.

RETURNS

Returns 0 on success, or an error code specified in `<oskit/error.h>`, on error.

OSKIT_NETIO: NETWORK PACKET I/O INTERFACE

This interface builds on the above interfaces to provide a clean and simple but powerful interface for passing network packets between device drivers and protocol stacks, and possibly between layers of a protocol stack as well.

The oskit_netio interface uses a symmetric sender-driven model for asynchronous communication. Each party involved (*e.g.,* the network device driver and the protocol stack) must implement a netio object and pass a reference to its own netio object to the other party. For example, in the `oskit_netdev` interface, which represents a network device of some kind, this exchange of netio objects occurs when the protocol stack or other client opens the device.

The `oskit_netio` interface defines only a single additional method beyond the basic methods inherited from `oskit_iunknown`; this method, appropriately named push, is used to "push" a network packet to the "other" party. For example, when a network device driver receives a packet from the hardware, the driver calls the push method on the netio object provided by the protocol stack; conversely, when the protocol stack needs to send a packet, it calls the netio object returned by the device driver at the time the device was opened. Thus, a netio object essentially represents a "packet consumer."

The following section describes the specifics of the push method.

PUSH A PACKET THROUGH TO THE PACKET CONSUMER

SYNOPSIS

```
#include <oskit/io/netio.h>
```

OSKIT_COMDECL push(oskit_netio_t *_io_, oskit_bufio *_buf_, oskit_size_t _size_);

DESCRIPTION

This method feeds a network packet to the packet consumer represented by the netio object; what the consumer does with the packet depends entirely on who the consumer is and how it is configured. The packet is contained in a bufio object which must be at least the size of the packet, but may be larger; the *size* parameter on the push call indicates the actual size of the packet. If the consumer needs to hold on to the provided bufio object after returning from the call, it must call addref on the bufio object to obtain its own reference; then it must release this reference at some later time when it is done with the buffer. Otherwise, if the consumer doesn't obtain its own reference, the caller may recycle the buffer as soon as the call returns. The passed buffer object is logically read-only; the consumer must not attempt to write to it. The size parameter to this call is the actual size of the packet; the size of the buffer, as returned by the getsize method, may be larger than the size of the packet.

PARAMETERS

io:

The oskit_netio interface representing the packet consumer.

buf:

The oskit_bufio interface to the buffer object containing the packet.

size:

The actual size of the packet; must be less than or equal to the size of the buffer object.

RETURNS

Returns 0 on success, or an error code specified in `<oskit/error.h>`, on error.

OSKIT_POSIXIO: POSIX I/O INTERFACE

The oskit_posixio interface defines the minimal POSIX I/O interface that any POSIX I/O object (file, device, pipe, socket, etc) can be expected to support. Only per-object methods are provided by this interface. Additional I/O operations are supported through separate interfaces, such as the oskit_stream interface and oskit_absio COM interface.

The `oskit_posixio` COM interface inherits from oskit_iunknown, and has the following additional methods:

stat:

Get this object's attributes.

setstat:

Set this object's attributes.

pathconf

Get this object's value for a configuration option variable.

GET ATTRIBUTES OF THIS OBJECT

SYNOPSIS

```
#include <oskit/io/posixio.h>
```

OSKIT_COMDECL oskit_posixio_stat(oskit_posixio_t *f, [out] oskit_stat_t *out_stats);

DESCRIPTION

This method returns the attributes of this object. Depending on the type of object, only some of the attributes may be meaningful. `out_stats` is a pointer to a `oskit_stat_t` structure defined as follows:

```
  struct oskit_stat {
     oskit_dev_t        dev;      /* device on which inode resides
*/
     oskit_ino_t        ino;      /* inode's number
*/
     oskit_mode_t       mode;     /* file mode
*/
     oskit_nlink_t      nlink;    /* number of hard links to file
*/
     oskit_uid_t        uid;      /* user id of owner
*/
     oskit_gid_t        gid;      /* group id of owner
*/
     oskit_dev_t        rdev;     /* device number, for device
files */
     oskit_timespec_t   atime;    /* time of last access
*/
     oskit_timespec_t   mtime;    /* time of last data
modification            */
     oskit_timespec_t   ctime;    /* time of last attribute change
*/
     oskit_off_t        size;     /* size in bytes
*/
     oskit_u64_t        blocks;   /* blocks allocated for file
*/
     oskit_u32_t        blksize;  /* optimal block size in bytes
*/
  };
```

PARAMETERS

f:

The object whose attributes are desired.

out_stats:

The attributes of the specified object.

RETURNS

Returns 0 on success, or an error code specified in `<oskit/error.h>`, on error.

SET THE ATTRIBUTES OF THIS OBJECT

SYNOPSIS

```
#include <oskit/io/posixio.h>
```

OSKIT_COMDECL oskit_posixio_setstat(oskit_posixio_t *f, oskit_u32_t *mask*, const oskit_stat_t **stat*);

DESCRIPTION

This method sets the attributes specified in mask to the values specified in stat. mask may be any combination of the following:

OSKIT_STAT_MODE:

Set the file mode, except for the file type bits, as in the Unix chmod system call.

OSKIT_STAT_UID:

Set the file user id, as in the Unix chown system call.

OSKIT_STAT_GID:

Set the file group id, as in the Unix chown system call.

OSKIT_STAT_SIZE:

Set the file size, as in the Unix truncate system call.

OSKIT_STAT_ATIME:

Set the file's last access timestamp to a particular value, as in the Unix utimes system call with a non-NULL parameter.

OSKIT_STAT_MTIME:

Set the file's last data modification timestamp to a particular value, as in the Unix utimes system call with a non-NULL parameter.

OSKIT_STAT_UTIMES_NULL:

Set the file's last access timestamp and data modification timestamp to the current time, as in the Unix utimes system call with a NULL parameter.

Typically, this method is not supported for symbolic links.

PARAMETERS

f:

The object whose attributes are to be changed.

***mask*:**

The attributes to be changed.

***stat*:**

The new attribute values.

RETURNS

Returns 0 on success, or an error code specified in **<oskit/error.h>**, on error.

GET VALUE OF A CONFIGURATION OPTION VARIABLE

SYNOPSIS

#include <oskit/io/posixio.h>

OSKIT_COMDECL oskit_posixio_pathconf(oskit_posixio_t *f, oskit_s32_t *option*, [out] oskit_s32_t **out_val*);

DESCRIPTION

This method returns the value of the specified configuration option variable for this object. The value of option may be one of the following:

OSKIT_PC_LINK_MAX:

Get the maximum file link count.

OSKIT_PC_MAX_CANON:

Get the maximum size of the terminal input line.

OSKIT_PC_MAX_INPUT:

Get the maximum input queue size.

OSKIT_PC_NAME_MAX:

Get the maximum number of bytes in a filename.

OSKIT_PC_PATH_MAX:

Get the maximum number of bytes in a pathname.

OSKIT_PC_PIPE_BUF:

Get the maximum atomic write size to a pipe.

OSKIT_PC_CHOWN_RESTRICTED:

Determine whether use of chown is restricted.

OSKIT_PC_NO_TRUNC:

Determine whether too-long pathnames produce errors.

OSKIT_PC_VDISABLE:

Get value to disable special terminal characters.

OSKIT_PC_ASYNC_IO:

Determine whether asynchronous IO is supported.

OSKIT_PC_PRIO_IO:

Determine whether prioritized IO is supported.

OSKIT_PC_SYNC_IO:

Determine whether synchronized IO is supported.

PARAMETERS

f:

The object from which to obtain a configuration option value

***option*:**

The configuration option variable

***out_val*:**

The value of the configuration option value.

RETURNS

Returns 0 on success, or an error code specified in **<oskit/error.h>**, on error.

OSKIT_TTYSTREAM: INTERFACE TO UNIX TTY-LIKE STREAMS

This interface extends the standard COM IStream interface with POSIX/Unix TTY functionality, such as methods to control serial port settings, enable, disable, and control line editing, flush the input and output queues, *etc.* This interface is currently exported by character-oriented device drivers incorporated into the OSKit from legacy systems such as BSD and Linux, in which full Unix TTY functionality can be provided easily. In the future, these drivers are expected to export more minimal, lower-level interfaces instead of or in addition to this interface; however, in the short term, this interface allows clients to obtain full Unix terminal functionality quickly and easily. The `oskit_ttystream` interface inherits from oskit_stream, and has the following additional methods:

`getattr:`
: Get the stream's current TTY attributes.

`setattr:`
: Set the stream's TTY attributes.

`sendbreak:`
: Send a break signal over the line.

`drain:`
: Wait until all buffered output has been transmitted.

`flush:`
: Discared buffered input and/or output data.

`flow:`
: Suspend or resume data transmission or reception.

In addition, this header file defines a structure called oskit_termios, corresponding to the standard POSIX termios structure, and a set of related definitions used to specify terminal-related settings. See the POSIX and Unix standards for details on the exact contents and meaning of this structure.

GET THE STREAM'S CURRENT TTY ATTRIBUTES

SYNOPSIS

```
#include <oskit/io/ttystream.h>
```

OSKIT_COMDECL getattr(oskit_ttystream_t **tty*, [out] struct oskit_termios **attr*);

DESCRIPTION

This method retrieves the current line settings of this stream and returns them in the specified `oskit_termios` structure. This method corresponds to the POSIX tcgetattr function; see the POSIX standard for details.

PARAMETRS

tty:

The TTY stream object to query.

attr:

The structure to be filled with the current line settings.

RETURNS

Returns 0 on success, or an error code specified in <oskit/error.h>, on error.

SET THE STREAM'S TTY ATTRIBUTES

SYNOPSIS

```
#include <oskit/io/ttystream.h>
```

OSKIT_COMDECL setattr(oskit_ttystream_t **tty*, const struct oskit_termios **attr*);

DESCRIPTION

This method sets the line settings of this stream based on the specified `oskit_termios` structure. This method corresponds to the POSIX tcsetattr function; see the POSIX standard for details.

PARAMETERS

tty:

The TTY stream object to modify.

attr:

The structure containing the new line settings.

RETURNS

Returns 0 on success, or an error code specified in <oskit/error.h>, on error.

SEND A BREAK SIGNAL

SYNOPSIS

```
#include <oskit/io/ttystream.h>
```

OSKIT_COMDECL sendbreak(oskit_ttystream_t **tty*, oskit_u32_t *duration*);

DESCRIPTION

On streams controlling asynchronous serial communication, this method sends a break signal (a continuous stream of zero-valued bits) for a specific duration. This method corresponds to the POSIX tcsendbreak function; see the POSIX standard for details.

PARAMETERS

tty:

The TTY stream on which to send the break.

duration:

The duration of the break signal to send. If this parameter is zero, then the duration will be between 0.25 and 0.5 seconds.

RETURNS

Returns 0 on success, or an error code specified in <oskit/error.h>, on error.

WAIT UNTIL ALL BUFFERED OUTPUT HAS BEEN TRANSMITTED

SYNOPSIS

```
#include <oskit/io/ttystream.h>
```

OSKIT_COMDECL drain(oskit_ttystream_t **tty*);

DESCRIPTION

This method waits until any buffered output data that has been written to the stream is successfully transmitted. This method corresponds to the POSIX tcdrain function; see the POSIX standard for details.

PARAMETERS

tty:

The TTY stream object to drain.

RETURNS

Returns 0 on success, or an error code specified in <oskit/error.h>, on error.

DISCARD BUFFERED INPUT AND/OR OUTPUT DATA

SYNOPSIS

```
#include <oskit/io/ttystream.h>
```

OSKIT_COMDECL flush(oskit_ttystream_t **tty*, int *queue_selector*);

DESCRIPTION

This method discards any buffered output data that has not yet been transmitted, and/or any buffered input data that has not yet been read, depending on the *queue_selector* parameter. This method corresponds to the POSIX tcflush function; see the POSIX standard for details.

PARAMETERS

tty:

The TTY stream object to flush.

queue_selector:

Must be one of the following:

OSKIT_TCIFLUSH:

Flush the input buffer.

OSKIT_TCOFLUSH:

Flush the output buffer.

OSKIT_TCIOFLUSH:

Flush the input and output buffers.

RETURNS

Returns 0 on success, or an error code specified in <oskit/error.h>, on error.

SUSPEND OR RESUME DATA TRANSMISSION OR RECEPTION

SYNOPSIS

```
#include <oskit/io/ttystream.h>
```

OSKIT_COMDECL flow(oskit_ttystream_t **tty*, int *action*);

DESCRIPTION

This method controls the transmission or reception of data on this TTY stream. This method corresponds to the POSIX tcflow function; see the POSIX standard for details.

PARAMETERS

tty:

The TTY stream object to control.

action:

Must be one of the following:

OSKIT_TCOOFF:

Suspend output.

OSKIT_TCOON:

Restart output.

OSKIT_TCIOFF:

Transmit a STOP character.

OSKIT_TCION:

Transmit a START character.

6

Database Storage Structures

Database tables/indexes are typically stored on hard disk in one of many forms, ordered/unordered Flat files, ISAM, Heaps, Hash buckets or B+ Trees. These have various advantages and disadvantages discussed in this topic. The most commonly used are B+trees and ISAM.

UNORDERED

Unordered storage typically stores the records in the order they are inserted. While having good insertion efficiency (), it may seem that it would have inefficient retrieval times (), but this is usually never the case as most databases use indexes on the primary keys, resulting in or for keys that are the same as database row offsets within the database file storage system, efficient retrieval times.

ORDERED

Ordered storage typically stores the records in order and may have to rearrange or increase the file size in the case

a record is inserted, this is very inefficient. However is better for retrieval as the records are pre-sorted, leading to a complexity of.

STRUCTURED FILES

HEAPS

- simplest and most basic method
  - o insert efficient, records added at end of file – 'chronological' order
  - o retrieval inefficient as searching has to be linear
  - o deletion – deleted records marked requires periodic reorganization if file is very volatile
- advantages
  - o good for bulk loading data
  - o good for relatively small relations as indexing overheads are avoided
  - o good when retrievals involve large proportion of records
- disadvantages
  - o not efficient for selective retrieval using key values, especially if large
  - o sorting may be time-consuming
- not suitable for 'volatile' tables

HASH BUCKETS

- Hash functions calculate the address of the page in which the record is to be stored based on one or more fields in the record

  - Hashing functions chosen to ensure that addresses are spread evenly across the address space
  - 'occupancy' is generally 40% – 60% of total file size
  - unique address not guaranteed so collision detection and collision resolution mechanisms are required
- open addressing
- chained/unchained overflow
- pros and cons
  - efficient for exact matches on key field
  - not suitable for range retrieval, which requires sequential storage
  - calculates where the record is stored based on fields in the record
  - hash functions ensure even spread of data
  - collisions are possible, so collision detection and restoration is required

B+ TREES

These are the most used in practice.

- the time taken to access any tuple is the same because same number of nodes searched
- index is a full index so data file does not have to be ordered
- Pros and cons
  - versatile data structure – sequential as well as random access
  - access is fast
  - supports exact, range, part key and pattern matches efficiently

- o 'volatile' files are handled efficiently because index is dynamic – expands and contracts as table grows and shrinks
- o less well suited to relatively stable files – in this case, ISAM is more efficient

INDEX (DATABASE)

Database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of slower writes and increased storage space. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records. The disk space required to store the index is typically less than that required by the table (since indices usually contain only the key-fields according to which the table is to be arranged, and exclude all the other details in the table), yielding the possibility to store indices in memory for a table whose data is too large to store in memory. In a relational database, an index is a copy of one part of a table. Some databases extend the power of indexing by allowing indices to be created on functions or expressions. For example, an index could be created on upper(last_name), which would only store the upper case versions of the last_name field in the index. Another option sometimes supported is the use of "filtered" indices, where index entries are created only for those records that satisfy some conditional expression. A further aspect of flexibility is to permit indexing on user-defined functions, as well as expressions formed from an assortment of built-in functions. Indices may be defined as

unique or non-unique. A unique index acts as a constraint on the table by preventing duplicate entries in the index and thus the backing table.

INDEX ARCHITECTURE

Index architectures can be classified as clustered or nonclustered.

NON-CLUSTERED

The data is present in random order, but the logical ordering is specified by the index. The data rows may be randomly spread throughout the table. The non-clustered index tree contains the index keys in sorted order, with the leaf level of the index containing the pointer to the page and the row number in the data page. In non-clustered index:

- The physical order of the rows is not the same as the index order.
- Typically created on column used in JOIN, WHERE, and ORDER BY clauses.
- Good for tables whose values may be modified frequently.

Microsoft SQL Server creates non-clustered indices by default when CREATE INDEX command is given. There can be more than one non-clustered index on a database table. There can be as many as 249 nonclustered indexes per table. It also creates a clustered index on a primary key by default.

CLUSTERED

Clustering alters the data block into a certain distinct order to match the index, resulting in the row data being stored in order. Therefore, only one clustered index can be

created on a given database table. Clustered indices can greatly increase overall speed of retrieval, but usually only where the data is accessed sequentially in the same or reverse order of the clustered index, or when a range of items is selected. Since the physical records are in this sort order on disk, the next row item in the sequence is immediately before or after the last one, and so fewer data block reads are required. The primary feature of a clustered index is therefore the ordering of the physical data rows in accordance with the index blocks that point to them. Some databases separate the data and index blocks into separate files, others put two completely different data blocks within the same physical file(s). Create an object where the physical order of rows is same as the index order of the rows and the bottom(leaf) level of clustered index contains the actual data rows. They are known as "index organized tables" under Oracle database.

COLUMN ORDER

The order in which columns are listed in the index definition is important. It is possible to retrieve a set of row identifiers using only the first indexed column. However, it is not possible or efficient (on most databases) to retrieve the set of row identifiers using only the second or greater indexed column. For example, imagine a phone book that is organized by city first, then by last name, and then by first name. If you are given the city, you can easily extract the list of all phone numbers for that city. However, in this phone book it would be very tedious to find all the phone numbers for a given last name. You would have to look

within each city's section for the entries with that last name. Some databases can do this, others just won't use the index.

APPLICATIONS AND LIMITATIONS

Indices are useful for many applications but come with some limitations. Consider the following SQL statement: SELECT first_name FROM people WHERE last_name = 'Smith';. To process this statement without an index the database software must look at the last_name column on every row in the table (this is known as a full table scan). With an index the database simply follows the B-tree data structure until the Smith entry has been found; this is much less computationally expensive than a full table scan. Consider this SQL statement: SELECT email_address FROM customers WHERE email_address LIKE '%@yahoo.com';. This query would yield an email address for every customer whose email address ends with "@yahoo.com", but even if the email_address column has been indexed the database still must perform a full table scan. This is because the index is built with the assumption that words go from left to right. With a wildcard at the beginning of the search-term, the database software is unable to use the underlying b-tree data structure (in other words, the WHERE-clause is *not sargable*). This problem can be solved through the addition of another index created on reverse(email_address) and a SQL query like this: SELECT email_address FROM customers WHERE reverse(email_address) LIKE reverse('%@yahoo.com');. This puts the wild-card at the right-most part of the query (now moc.oohay@%) which the index on reverse(email_address) can satisfy.

TYPES

BITMAP INDEX

A bitmap index is a special kind of index that stores the bulk of its data as bit arrays (bitmaps) and answers most queries by performing bitwise logical operations on these bitmaps. The most commonly used index, such as B+trees, are most efficient if the values it indexes do not repeat or repeat a smaller number of times. In contrast, the bitmap index is designed for cases where the values of a variable repeat very frequently. For example, the gender field in a customer database usually contains two distinct values: male or female. For such variables, the bitmap index can have a significant performance advantage over the commonly used trees.

DENSE INDEX

A dense index in databases is a file with pairs of keys and pointers for every record in the data file. Every key in this file is associated with a particular pointer to *a record* in the sorted data file. In clustered indices with duplicate keys, the dense index points *to the first record* with that key.

SPARSE INDEX

A sparse index in databases is a file with pairs of keys and pointers for every block in the data file. Every key in this file is associated with a particular pointer *to the block* in the sorted data file. In clustered indices with duplicate keys, the sparse index points *to the lowest search key* in each block. primary key is a sparse index.

REVERSE INDEX

A reverse key index reverses the key value before entering it in the index. E.g., the value 24538 becomes 83542 in the index. Reversing the key value is particularly useful for indexing data such as sequence numbers, where new key values monotonically increase.

INDEX IMPLEMENTATIONS

Indices can be implemented using a variety of data structures. Popular indices include balanced trees, B+ trees, Fractal Tree™ indexes and hashes. In Microsoft SQL Server, the leaf node of the clustered index corresponds to the actual data, not simply a pointer to data that resides elsewhere, as is the case with a non-clustered index. Each relation can have a single clustered index and many unclustered indices.

INDEX CONCURRENCY CONTROL

An index is typically being accessed concurrently by several transactions and processes, and thus needs concurrency control. While in principle indexes can utilize the common database concurrency control methods, specialized concurrency control methods for indexes exist, which are applied in conjunction with the common methods for a substantial performance gain.

COVERING INDEX

In most cases, an index is used to quickly locate the data record(s) from which the required data is read. In other words, the index is only used to locate data records in the

table and not to return data. A covering index is a special case where the index itself contains the required data field(s) and can return the data. Consider the following table (other fields omitted):

ID	Name	Other Fields
12	Plug	...
13	Lamp	...
14	Fuse	...

To find the Name for ID 13, an index on (ID) will be useful, but the record must still be read to get the Name. However, an index on (ID, Name) contains the required data field and eliminates the need to look up the record. A covering index can dramatically speed up data retrieval but may itself be large due to the additional keys, which slow down data insertion & update. To reduce such index size, some systems allow non-key fields to be included in the index. Non-key fields are not themselves part of the index ordering but only included at the leaf level, allowing for a covering index with less overall index size.

STANDARDIZATION

There is no standard about creating indexes because the ISO SQL Standard does not cover physical aspects, and indexes are one of the pysical part of databse conception among others like storage (tablespace or filegroups). However RDBMS vendors all give a CREATE INDEX syntax with some specific options which depends on functionalities they provide to customers.

DATABASE SECURITY

Database security concerns the use of a broad range of information security controls to protect databases (potentially including the data, the database applications or stored functions, the database systems, the database servers and the associated network links) against compromises of their confidentiality, integrity and availability. It involves various types or categories of controls, such as technical, procedural/ administrative and physical. *Database security* is a specialist topic within the broader realms of computer security, information security and risk management. Security risks to database systems include, for example:

- Unauthorized or unintended activity or misuse by authorized database users, database administrators, or network/systems managers, or by unauthorized users or hackers (e.g. inappropriate access to sensitive data, metadata or functions within databases, or inappropriate changes to the database programmes, structures or security configurations);
- Malware infections causing incidents such as unauthorized access, leakage or disclosure of personal or proprietary data, deletion of or damage to the data or programmes, interruption or denial of authorized access to the database, attacks on other systems and the unanticipated failure of database services;
- Overloads, performance constraints and capacity issues resulting in the inability of authorized users to use databases as intended;
- Physical damage to database servers caused by computer room fires or floods, overheating, lightning,

accidental liquid spills, static discharge, electronic breakdowns/equipment failures and obsolescence;

- Design flaws and programming bugs in databases and the associated programmes and systems, creating various security vulnerabilities (e.g. unauthorized privilege escalation), data loss/corruption, performance degradation etc.;
- Data corruption and/or loss caused by the entry of invalid data or commands, mistakes in database or system administration processes, sabotage/criminal damage etc.

Many layers and types of information security control are appropriate to databases, including:

- Access control
- Auditing
- Authentication
- Encryption
- Integrity controls
- Backups
- Application security

Traditionally databases have been largely secured against hackers through network security measures such as firewalls, and network-based intrusion detection systems. While network security controls remain valuable in this regard, securing the database systems themselves, and the programmes/functions and data within them, has arguably become more critical as networks are increasingly opened to wider access, in particular access from the Internet. Furthermore, system, programme, function and data access

controls, along with the associated user identification, authentication and rights management functions, have always been important to limit and in some cases log the activities of authorized users and administrators. In other words, these are complementary approaches to database security, working from both the outside-in and the inside-out as it were. Many organizations develop their own "baseline" security standards and designs detailing basic security control measures for their database systems.

These may reflect general information security requirements or obligations imposed by corporate information security policies and applicable laws and regulations (e.g. concerning privacy, financial management and reporting systems), along with generally-accepted good database security practices (such as appropriate hardening of the underlying systems) and perhaps security recommendations from the relevant database system and software vendors.

The security designs for specific database systems typically specify further security administration and management functions (such as administration and reporting of user access rights, log management and analysis, database replication/synchronization and backups) along with various business-driven information security controls within the database programmes and functions (e.g. data entry validation and audit trails). Furthermore, various security-related activities (manual controls) are normally incorporated into the procedures, guidelines etc. relating to the design, development, configuration, use, management and maintenance of databases.

VULNERABILITY ASSESSMENTS AND COMPLIANCE

One technique for evaluating database security involves performing vulnerability assessments or penetration tests against the database. Testers attempt to find security vulnerabilities that could be used to defeat or bypass security controls, break into the database, compromise the system etc. Database administrators or information security administrators may for example use automated vulnerability scans to search out misconfiguration of controls within the layers mentioned above along with known vulnerabilities within the database software. The results of such scans are used to harden the database (improve the security controls) and close off the specific vulnerabilities identified, but unfortunately other vulnerabilities typically remain unrecognized and unaddressed. A programme of continual monitoring for compliance with database security standards is another important task for mission critical database environments. Two crucial aspects of database security compliance include patch management and the review and management of permissions (especially public) granted to objects within the database.

Database objects may include table or other objects listed in the Table link. The permissions granted for SQL language commands on objects are considered in this process. One should note that compliance monitoring is similar to vulnerability assessment with the key difference that the results of vulnerability assessments generally drive the security standards that lead to the continuous monitoring programme. Essentially, vulnerability assessment is a

preliminary procedure to determine risk where a compliance programme is the process of on-going risk assessment. The compliance programme should take into consideration any dependencies at the application software level as changes at the database level may have effects on the application software or the application server. In direct relation to this topic is that of application security.

ABSTRACTION

Application level authentication and authorization mechanisms should be considered as an effective means of providing abstraction from the database layer. The primary benefit of abstraction is that of a single sign-on capability across multiple databases and database platforms. A Single sign-on system should store the database user's credentials (login id and password), and authenticate to the database on behalf of the user.

DATABASE ACTIVITY MONITORING (DAM)

Another security layer of a more sophisticated nature includes real-time database activity monitoring, either by analyzing protocol traffic (SQL) over the network, or by observing local database activity on each server using software agents, or both. Use of agents or native logging is required to capture activities executed on the database server, which typically include the activities of the database administrator. Agents allow this information to be captured in a fashion that can not be disabled by the database administrator, who has the ability to disable or modify native audit logs.

Analysis can be performed to identify known exploits or policy breaches, or baselines can be captured over time to build a normal pattern used for detection of anomalous activity that could be indicative of intrusion. These systems can provide a comprehensive Database audit trail in addition to the intrusion detection mechanisms, and some systems can also provide protection by terminating user sessions and/or quarantining users demonstrating suspicious behaviour.

Some systems are designed to support separation of duties (SOD), which is a typical requirement of auditors. SOD requires that the database administrators who are typically monitored as part of the DAM, not be able to disable or alter the DAM functionality. This requires the DAM audit trail to be securely stored in a separate system not administered by the database administration group.

NATIVE AUDIT

In addition to using external tools for monitoring or auditing, native database audit capabilities are also available for many database platforms. The native audit trails are extracted on a regular basis and transferred to a designated security system where the database administrators do not have access.

This ensures a certain level of segregation of duties that may provide evidence the native audit trails were not modified by authenticated administrators. Turning on native impacts the performance of the server. Generally, the native audit trails of databases do not provide sufficient controls to enforce separation of duties; therefore, the network and/

or kernel module level host based monitoring capabilities provides a higher degree of confidence for forsenics and preservation of evidence.

PROCESS AND PROCEDURES

A *database security* programme should include the regular review of permissions granted to individually owned accounts and accounts used by automated processes. The accounts used by automated processes should have appropriate controls around password storage such as sufficient encryption and access controls to reduce the risk of compromise.

For individual accounts, a two-factor authentication system should be considered in a database environment where the risk is commensurate with the expenditure for such an authentication system. In conjunction with a sound *database security* programme, an appropriate disaster recovery programme should exist to ensure that service is not interrupted during a security incident or any other incident that results in an outage of the primary database environment. An example is that of replication for the primary databases to sites located in different geographical regions. After an incident occurs, the usage of database forensics should be employed to determine the scope of the breach, and to identify appropriate changes to systems and/or processes to prevent similar incidents in the future.

DATABASE TRANSACTION

A database transaction comprises a unit of work performed within a database management system (or similar

system) against a database, and treated in a coherent and reliable way independent of other transactions. Transactions in a database environment have two main purposes:

1. To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure, when execution stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status.
2. To provide isolation between programmes accessing a database concurrently. Without isolation the program's outcomes are possibly erroneous.

A database transaction, by definition, must be atomic, consistent, isolated and durable. Database practitioners often refer to these properties of database transactions using the acronym ACID. Transactions provide an "all-or-nothing" proposition, stating that each work-unit performed in a database must either complete in its entirety or have no effect whatsoever. Further, the system must isolate each transaction from other transactions, results must conform to existing constraints in the database, and transactions that complete successfully must get written to durable storage.

PURPOSE

Databases and other data stores which treat the integrity of data as paramount often include the ability to handle transactions to maintain the integrity of data. A single transaction consists of one or more independent units of work, each reading and/or writing information to a database or other data store. When this happens it is often important

to ensure that all such processing leaves the database or data store in a consistent state. Examples from double-entry accounting systems often illustrate the concept of transactions. In double-entry accounting every debit requires the recording of an associated credit. If one writes a check for €100 to buy groceries, a transactional double-entry accounting system must record the following two entries to cover the single transaction:

1. Debit €100 to Groceries Expense Account
2. Credit €100 to Checking Account

A transactional system would make both entries — or both entries would fail. By treating the recording of multiple entries as an atomic transactional unit of work the system maintains the integrity of the data recorded. In other words, nobody ends up with a situation in which a debit is recorded but no associated credit is recorded, or vice versa.

TRANSACTIONAL DATABASES

A '*transactional database* is a DBMS where write transactions on the database are able to be rolled back if they are not completed properly (e.g. due to power or connectivity loss). Most modern relational database management systems fall into the category of databases that support transactions. In a database system a transaction might consist of one or more data-manipulation statements and queries, each reading and/or writing information in the database. Users of database systems consider consistency and integrity of data as highly important. A simple transaction is usually issued to the database system in a language like SQL wrapped in a transaction, using a pattern similar to the following:

1. Begin the transaction
2. Execute several data manipulations and queries
3. If no errors occur then commit the transaction and end it
4. If errors occur then rollback the transaction and end it

If no errors occurred during the execution of the transaction then the system commits the transaction. A transaction commit operation applies all data manipulations within the scope of the transaction and persists the results to the database. If an error occurs during the transaction, or if the user specifies a rollback operation, the data manipulations within the transaction are not persisted to the database. In no case can a partial transaction be committed to the database since that would leave the database in an inconsistent state. Internally, multi-user databases store and process transactions, often by using a transaction ID or XID.

IN SQL

SQL is inherently transactional, and a transaction is automatically started when another ends. Some databases extend SQL and implement a START TRANSACTION statement, but while seemingly signifying the start of the transaction it merely deactivates autocommit. The result of any work done after this point will remain invisible to other database-users until the system processes a COMMIT statement. A ROLLBACK statement can also occur, which will undo any work performed since the last transaction. Both COMMIT and ROLLBACK will end the transaction,

and start a new. If autocommit was disabled using START TRANSACTION, autocommit will often also be reenabled. Some database systems allow the synonyms BEGIN, BEGIN WORK and BEGIN TRANSACTION, and may have other options available.

DISTRIBUTED TRANSACTIONS

Database systems implement distributed transactions as transactions against multiple applications or hosts. A distributed transaction enforces the ACID properties over multiple systems or data stores, and might include systems such as databases, file systems, messaging systems, and other applications. In a distributed transaction a coordinating service ensures that all parts of the transaction are applied to all relevant systems. As with database and other transactions, if any part of the transaction fails, the entire transaction is rolled back across all affected systems.

TRANSACTIONAL FILESYSTEMS

The Namesys Reiser4 filesystem for Linux supports transactions, and as of Microsoft Windows Vista, the Microsoft NTFS filesystem supports distributed transactions across networks.

CONCURRENCY CONTROL

In information technology and computer science, especially in the fields of computer programming, operating systems, multiprocessors, and databases, concurrency control ensures that correct results for concurrent operations are generated, while getting those results as quickly as

possible. Computer systems, both software and hardware, consist of modules, or components. Each component is designed to operate correctly, i.e., to obey to or meet certain consistency rules. When components that operate concurrently interact by messaging or by sharing accessed data (in memory or storage), a certain component's consistency may be violated by another component. The general area of concurrency control provides rules, methods, design methodologies, and theories to maintain the consistency of components operating concurrently while interacting, and thus the consistency and correctness of the whole system. Introducing concurrency control into a system means applying operation constraints which typically result in some performance reduction. Operation consistency and correctness should be achieved with as good as possible efficiency, without reducing performance below reasonable.

CONCURRENCY CONTROL IN DATABASES

COMMENTS:

1. This section is applicable to all transactional systems, i.e., to all systems that use *database transactions* (*atomic transactions*; e.g., transactional objects in Systems management and in networks of smartphones which typically implement private, dedicated database systems), not only general-purpose database management systems (DBMSs).

2. DBMSs need to deal also with concurrency control issues not typical just to database transactions but rather to operating systems in general. These issues (e.g., see *Concurrency control in operating systems* below) are out of

the scope of this section. Concurrency control in Database management systems (DBMS; e.g., Bernstein et al. 1987, Weikum and Vossen 2001), other transactional objects, and related distributed applications (e.g., Grid computing and Cloud computing) ensures that *database transactions* are performed concurrently without violating the data integrity of the respective databases. Thus concurrency control is an essential element for correctness in any system where two database transactions or more, executed with time overlap, can access the same data, e.g., virtually in any general-purpose database system. Consequently a vast body of related research has been accumulated since database systems have emerged in the early 1970s. A well established concurrency control theory for database systems is outlined in the references mentioned above: serializability theory, which allows to effectively design and analyze concurrency control methods and mechanisms. An alternative theory for concurrency control of atomic transactions over abstract data types is presented in (Lynch et al. 1993), and not utilized below. This theory is more refined, with a wider scope, but has been less utilized in the Database literature than the classical theory above. Each theory has its pros and cons, emphasis and insight. To some extent they are complementary, and their merging may be useful.

To ensure correctness, a DBMS usually guarantees that only *serializable* transaction schedules are generated, unless *serializability* is intentionally relaxed to increase performance, but only in cases where application correctness is not harmed. For maintaining correctness in cases of failed (aborted) transactions (which can always happen for

many reasons) schedules also need to have the *recoverability* (from abort) property. A DBMS also guarantees that no effect of *committed* transactions is lost, and no effect of *aborted* (rolled back) transactions remains in the related database. Overall transaction characterization is usually summarized by the ACID rules below. As databases have become distributed, or needed to cooperate in distributed environments (e.g., Federated databases in the early 1990, and Cloud computing currently), the effective distribution of concurrency control mechanisms has received special attention.

DATABASE TRANSACTION AND THE ACID RULES

The concept of a *database transaction* (or *atomic transaction*) has evolved in order to enable both a well understood database system behaviour in a faulty environment where crashes can happen any time, and *recovery* from a crash to a well understood database state. A database transaction is a unit of work, typically encapsulating a number of operations over a database (e.g., reading a database object, writing, acquiring lock, etc.), an abstraction supported in database and also other systems. Each transaction has well defined boundaries in terms of which program/code executions are included in that transaction (determined by the transaction's programmer via special transaction commands). Every database transaction obeys the following rules (by support in the database system; i.e., a database system is designed to guarantee them for the transactions it runs):

- Atomicity - Either the effects of all or none of its operations remain ("all or nothing" semantics) when a transaction is completed (*committed* or *aborted* respectively). In other words, to the outside world a committed transaction appears (by its effects) to be indivisible, atomic, and an aborted transaction does not leave effects at all, as if never existed.
- Consistency - Every transaction must leave the database in a consistent (correct) state, i.e., maintain the predetermined integrity rules of the database (constraints upon and among the database's objects). A transaction must transform a database from one consistent state to another consistent state (it is the responsibility of the transaction's programmer to make sure that the transaction itself is correct, i.e., performs correctly what it intends to perform while maintaining the integrity rules). Thus since a database can be normally changed only by transactions, all the database's states are consistent. An aborted transaction does not change the state.
- Isolation - Transactions cannot interfere with each other. Moreover, usually the effects of an incomplete transaction are not visible to another transaction. Providing isolation is the main goal of concurrency control.
- Durability - Effects of successful (committed) transactions must persist through crashes (typically by recording the transaction's effects and its commit event in a non-volatile memory).

WHY IS CONCURRENCY CONTROL NEEDED?

If transactions are executed *serially*, i.e., sequentially with no overlap in time, no transaction concurrency exists. However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur. Here are some typical examples:

1. The lost update problem: A second transaction writes a second value of a data-item (datum) on top of a first value written by a first concurrent transaction, and the first value is lost to other transactions running concurrently which need, by their precedence, to read the first value. The transactions that have read the wrong value end with incorrect results.
2. The dirty read problem: Transactions read a value written by a transaction that has been later aborted. This value disappears from the database upon abort, and should not have been read by any transaction ("dirty read"). The reading transactions end with incorrect results.
3. The incorrect summary problem: While one transaction takes a summary over the values of all the instances of a repeated data-item, a second transaction updates some instances of that data-item. The resulting summary does not reflect a correct result for any (usually needed for correctness) precedence order between the two transactions (if one is executed before the other), but rather some random result, depending on the timing of the updates, and whether certain update results have been included in the summary or not.

CONCURRENCY CONTROL MECHANISMS

CATEGORIES

The main categories of concurrency control mechanisms are:

- Optimistic - Delay the checking of whether a transaction meets the isolation and other integrity rules (e.g., serializability and recoverability) until its end, without blocking any of its (read, write) operations ("...and be optimistic about the rules being met..."), and then abort a transaction to prevent the violation, if the desired rules are to be violated upon its commit. An aborted transaction is immediately restarted and re-executed, which incurs an obvious overhead (versus executing it to the end only once). If not too many transactions are aborted, then being optimistic is usually a good strategy.
- Pessimistic - Block an operation of a transaction, if it may cause violation of the rules, until the possibility of violation disappears. Blocking operations is typically involved with performance reduction.
- Semi-optimistic - Block operations in some situations, if they may cause violation of some rules, and do not block in other situations while delaying rules checking (if needed) to transaction's end, as done with optimistic.

Different categories provide different performance, i.e., different average transaction completion rates (*throughput*), depending on transaction types mix, computing level of parallelism, and other factors. If selection and knowledge

about trade-offs are available, then category and method should be chosen to provide the highest performance. The mutual blocking between two transactions (where each one blocks the other) or more results in a deadlock, where the transactions involved are stalled and cannot reach completion. Most non-optimistic mechanisms (with blocking) are prone to deadlocks which are resolved by an intentional abort of a stalled transaction (which releases the other transactions in that deadlock), and its immediate restart and re-execution. The likelihood of a deadlock is typically low.

METHODS

Many methods for concurrency control exist. Most of them can be implemented within either main category above. The major methods, which have each many variants, and in some cases may overlap or be combined, are:

1. Locking (e.g., Two-phase locking - 2PL) - Controlling access to data by locks assigned to the data. Access of a transaction to a data item (database object) locked by another transaction may be blocked (depending on lock type and access operation type) until lock release.
2. Serialization graph checking (also called Serializability, or Conflict, or Precedence graph checking) - Checking for cycles in the schedule's graph and breaking them by aborts.
3. Timestamp ordering (TO) - Assigning timestamps to transactions, and controlling or checking access to data by timestamp order.

4. Commitment ordering (or Commit ordering; CO) - Controlling or checking transactions' chronological order of commit events to be compatible with their respective precedence order.

Other major concurrency control types that are utilized in conjunction with the methods above include:

- Multiversion concurrency control (MVCC) - Increasing concurrency and performance by generating a new version of a database object each time the object is written, and allowing transactions' read operations of several last relevant versions (of each object) depending on scheduling method.
- Index concurrency control - Synchronizing access operations to indexes, rather than to user data. Specialized methods provide substantial performance gains.

The most common mechanism type in database systems since their early days in the 1970s has been *Strong strict Two-phase locking* (SS2PL; also called *Rigorous scheduling* or *Rigorous 2PL*) which is a special case (variant) of both Two-phase locking (2PL) and Commitment ordering (CO). It is pessimistic. In spite of its long name (for historical reasons) the idea of the SS2PL mechanism is simple: "Release all locks applied by a transaction only after the transaction has ended." SS2PL (or Rigorousness) is also the name of the set of all schedules that can be generated by this mechanism, i.e., these are SS2PL (or Rigorous) schedules, have the SS2PL (or Rigorousness) property.

MAJOR GOALS OF CONCURRENCY CONTROL MECHANISMS

Concurrency control mechanisms firstly need to operate correctly, i.e., to maintain each transaction's integrity rules while transactions are running concurrently, and thus the integrity of the entire transactional system. Correctness needs to be achieved with as good performance as possible. In addition, increasingly a need exists to operate effectively while transactions are distributed over processes, computers, and computer networks. Other subjects that may affect concurrency control are recovery and replication.

CORRECTNESS

SERIALIZABILITY

For correctness, a common major goal of most concurrency control mechanisms is generating schedules with the *Serializability* property. Without serializability undesirable phenomena may occur, e.g., money may disappear from accounts, or be generated from nowhere. Serializability of a schedule means equivalence (in the resulting database values) to some *serial* schedule with the same transactions (i.e., in which transactions are sequential with no overlap in time, and thus completely isolated from each other: No concurrent access by any two transactions to the same data is possible). Serializability is considered the highest level of isolation among database transactions, and the major correctness criterion for concurrent transactions. In some cases compromised, relaxed forms of serializability are allowed for better performance (e.g., the

popular *Snapshot isolation* mechanism) or to meet availability requirements in highly distributed systems, but only if application's correctness is not violated by the relaxation (e.g., no relaxation is allowed for money transactions, since by relaxation money can disappear, or appear from nowhere). Almost all implemented concurrency control mechanisms achieve serializability by providing *Conflict serializablity*, a broad special case of serializability (i.e., it covers, enables most serializable schedules, and does not impose significant additional delay-causing constraints) which can be implemented efficiently.

RECOVERABILITY

Comment: While in the general area of systems the term "recoverability" may refer to the ability of a system to recover from failure, within concurrency control of database systems this term has received a specific meaning.

Concurrency control typically also ensures the *Recoverability* property of schedules for maintaining correctness in cases of aborted transactions (which can always happen for many reasons). Recoverability (from abort) means that no committed transaction in a schedule has read data written by an aborted transaction. Such data disappear from the database (upon the abort) and are parts of an incorrect database state. Reading such data violates the consistency rule of ACID. Unlike Serializability, Recoverability cannot be compromised, relaxed at any case, since any relaxation results in quick database integrity violation upon aborts. The major methods listed above provide serializability mechanisms. None of them in its

general form automatically provides recoverability, and special considerations and mechanism enhancements are needed to support recoverability. A commonly utilized special case of recoverability is *Strictness*, which allows efficient database recovery from failure (but excludes optimistic implementations; e.g, Strict CO (SCO) cannot have an optimistic implementation, but has semi-optimistic ones).

Comment: Note that the *Recoverability* property is needed even if no database failure occurs and no database *recovery* from failure is needed. It is rather needed to correctly automatically handle transaction aborts, which may be unrelated to database failure and recovery from it.

DISTRIBUTION

With the fast technological development of computing the difference between local and distributed computing over low latency networks is blurring. Thus the quite effective utilization of local techniques in such distributed environments is common, e.g., in computer clusters. However for a large-scale distribution local concurrency control techniques typically do not scale well.

DISTRIBUTED SERIALIZABILITY AND COMMITMENT ORDERING

As database systems have become distributed, or started to cooperate in distributed environments (e.g., Federated databases in the early 1990s, and nowadays Grid computing, Cloud computing, and networks with smartphones), some transactions have become distributed. A distributed transaction means that the transaction spans processes,

and may span computers and geographical sites. This generates a need in effective distributed concurrency control mechanisms. Achieving the Serializability property of a distributed system's schedule effectively poses special challenges typically not met by most of the regular serializability mechanisms, originally designed to operate locally. This is especially due to a need in costly distribution of concurrency control information amid communication and computer latency. The only known general effective technique for distribution is Commitment ordering, which was disclosed publicly in 1991 (after being patented). Commitment ordering (Commit ordering, CO; Raz 1992) means that transactions' chronological order of commit events is kept compatible with their respective precedence order.

CO does not require the distribution of concurrency control information and provides a general effective solution (reliable, high-performance, and scalable) for both distributed and global serializability, also in a heterogeneous environment with database systems (or other transactional objects) with different (any) concurrency control mechanisms. CO is indifferent to which mechanism is utilized, since it does not interfere with any transaction operation scheduling (which most mechanisms control), and only determines the order of commit events. Thus, CO enables the efficient distribution of all other mechanisms, and also the distribution of a mix of different (any) local mechanisms, for achieving distributed and global serializability. The existence of such a solution has been considered "unlikely" until 1991, and by many experts also later, due to

misunderstanding of the CO solution. An important side-benefit of CO is automatic distributed deadlock resolution. Contrary to CO, virtually all other techniques (when not combined with CO) are prone to distributed deadlocks (also called global deadlocks) which need special handling. CO is also the name of the resulting schedule property: A schedule has the CO property if the chronological order of its transactions' commit events is compatible with the respective transactions' precedence (partial) order.

SS2PL mentioned above is a variant (special case) of CO and thus also effective to achieve distributed and global serializability. It also provides automatic distributed deadlock resolution (a fact overlooked in the research literature even after CO's publication), as well as Strictness and thus Recoverability. Possessing these desired properties together with known efficient locking based implementations explains SS2PL's popularity. SS2PL has been utilized to efficiently achieve Distributed and Global serializability since the 1980, and has become the de-facto standard for it. However, SS2PL is blocking and constraining (pessimistic), and with the proliferation of distribution and utilization of systems different from traditional database systems (e.g., as in Cloud computing), less constraining types of CO (e.g., Optimistic CO) may be needed for better performance.

COMMENTS

1. The *Distributed conflict serializability* property in its general form is difficult to achieve efficiently, but it is achieved efficiently via its special case *Distributed CO*: Each local component (e.g., a local DBMS) needs both to provide

some form of CO, and enforce a special *voting strategy* for the *Two-phase commit protocol* (2PC: utilized to commit distributed transactions). Differently from the general Distributed CO, *Distributed SS2PL* exists automatically when all local components are SS2PL based (in each component CO exists, implied, and the voting strategy is now met automatically). This fact has been known and utilized since the 1980s (i.e., that SS2PL exists globally, without knowing about CO) for efficient Distributed SS2PL, which implies Distributed serializability and strictness (e.g., see Raz 1992, page 293; it is also implied in Bernstein et al. 1987, page 78). Less constrained Distributed serializability and strictness can be efficiently achieved by Distributed Strict CO (SCO), or by a mix of SS2PL based and SCO based local components.

2. About the references and Commitment ordering: (Bernstein et al. 1987) was published before the discovery of CO in 1990. CO is called *Dynamic atomicity* in (Lynch et al. 1993, page 201; see The History of Commitment Ordering). CO is described in (Weikum and Vossen 2001, pages 102, 700), but the description is partial and misses CO's essence. (Raz 1992) was the first refereed and accepted for publication article about CO. Other CO articles followed.

DISTRIBUTED RECOVERABILITY

Unlike Serializability, *Distributed recoverability* and *Distributed strictness* can be achieved efficiently in a straightforward way, similarly to the way Distributed CO is achieved: In each database system they have to be applied locally, and employ a voting strategy for the Two-phase commit protocol (2PC; Raz 1992, page 307).

OTHER SUBJECTS OF ATTENTION

The design of concurrency control mechanisms is often influenced by the following subjects:

RECOVERY

All systems are prone to failures, and handling *recovery* from failure is a must. The properties of the generated schedules, which are dictated by the concurrency control mechanism, may have an impact on the effectiveness and efficiency of recovery. For example, the Strictness property is often desirable for an efficient recovery.

REPLICATION

For high availability database objects are often *replicated*. Updates of replicas of a same database object need to be kept synchronized. This may affect the way concurrency control is done.

REPLICATION (COMPUTER SCIENCE)

Replication is the process of sharing information so as to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility. It could be *data replication* if the same data is stored on multiple storage devices, or *computation replication* if the same computing task is executed many times. A computational task is typically *replicated in space*, i.e. executed on separate devices, or it could be *replicated in time*, if it is executed repeatedly on a single device. The access to a replicated entity is typically uniform with access to a single, non-replicated entity. The

replication itself should be transparent to an external user. Also, in a failure scenario, a failover of replicas is hidden as much as possible. It is common to talk about active and passive replication in systems that replicate data or services. *Active replication* is performed by processing the same request at every replica.

In *passive replication*, each single request is processed on a single replica and then its state is transferred to the other replicas. If at any time one master replica is designated to process all the requests, then we are talking about the *primary-backup* scheme (*master-slave* scheme) predominant in high-availability clusters. On the other side, if any replica processes a request and then distributes a new state, then this is a *multi-primary* scheme (called *multi-master* in the database field). In the multi-primary scheme, some form of distributed concurrency control must be used, such as distributed lock manager. Load balancing is different from task replication, since it distributes a load of different (not the same) computations across machines, and allows a single computation to be dropped in case of failure. Load balancing, however, sometimes uses data replication (esp. multi-master) internally, to distribute its data among machines. Backup is different from replication, since it saves a copy of data unchanged for a long period of time. Replicas on the other hand are frequently updated and quickly lose any historical state.

REPLICATION IN DISTRIBUTED SYSTEMS

Replication is one of the oldest and most important topics in the overall area of distributed systems. Whether

one replicates data or computation, the objective is to have some group of processes that handle incoming events. If we replicate data, these processes are passive and operate only to maintain the stored data, reply to read requests, and apply updates. When we replicate computation, the usual goal is to provide fault-tolerance. For example, a replicated service might be used to control a telephone switch, with the objective of ensuring that even if the primary controller fails, the backup can take over its functions. But the underlying needs are the same in both cases: by ensuring that the replicas see the same events in equivalent orders, they stay in consistent states and hence any replica can respond to queries.

REPLICATION MODELS IN DISTRIBUTED SYSTEMS

A number of widely cited models exist for data replication, each having its own properties and performance:

1. *Transactional replication*. This is the model for replicating transactional data, for example a database or some other form of transactional storage structure. The one-copy serializability model is employed in this case, which defines legal outcomes of a transaction on replicated data in accordance with the overall ACID properties that transactional systems seek to guarantee.
2. *State machine replication.* This model assumes that replicated process is a deterministic finite state machine and that atomic broadcast of every event is possible. It is based on a distributed computing

problem called *distributed consensus* and has a great deal in common with the transactional replication model. This is sometimes mistakenly used as synonym of *active replication*. State machine replication is usually implemented by a replicated log consisting of multiple subsequent rounds of the Paxos algorithm. This was popularized by Google's Chubby system, and is the core behind the open-source Keyspace data store.

3. *Virtual synchrony*. This computational model is used when a group of processes cooperate to replicate in-memory data or to coordinate actions. The model defines a new distributed entity called a *process group*. A process can join a group, which is much like opening a file: the process is added to the group, but is also provided with a checkpoint containing the current state of the data replicated by group members. Processes can then send events (*multicasts*) to the group and will see incoming events in the identical order, even if events are sent concurrently. Membership changes are handled as a special kind of platform-generated event that delivers a new *membership view* to the processes in the group.

Levels of performance vary widely depending on the model selected. Transactional replication is slowest, at least when one-copy serializability guarantees are desired (better performance can be obtained when a database uses log-based replication, but at the cost of possible inconsistencies if a failure causes part of the log to be lost). Virtual synchrony is the fastest of the three models, but the handling of

failures is less rigorous than in the transactional model. State machine replication lies somewhere in between; the model is faster than transactions, but much slower than virtual synchrony. The virtual synchrony model is popular in part because it allows the developer to use either active or passive replication. In contrast, state machine replication and transactional replication are highly constraining and are often embedded into products at layers where end-users would not be able to access them.

DATABASE REPLICATION

Database replication can be used on many database management systems, usually with a master/slave relationship between the original and the copies. The master logs the updates, which then ripple through to the slaves. The slave outputs a message stating that it has received the update successfully, thus allowing the sending (and potentially re-sending until successfully applied) of subsequent updates. Multi-master replication, where updates can be submitted to any database node, and then ripple through to other servers, is often desired, but introduces substantially increased costs and complexity which may make it impractical in some situations. The most common challenge that exists in multi-master replication is transactional conflict prevention or resolution. Most synchronous or eager replication solutions do conflict prevention, while asynchronous solutions have to do conflict resolution.

For instance, if a record is changed on two nodes simultaneously, an eager replication system would detect

the conflict before confirming the commit and abort one of the transactions. A lazy replication system would allow both transactions to commit and run a conflict resolution during resynchronization. The resolution of such a conflict may be based on a timestamp of the transaction, on the hierarchy of the origin nodes or on much more complex logic, which decides consistently on all nodes. Database replication becomes difficult when it scales up. Usually, the scale up goes with two dimensions, horizontal and vertical: horizontal scale up has more data replicas, vertical scale up has data replicas located further away in distance. Problems raised by horizontal scale up can be alleviated by a multi-layer multi-view access protocol. Vertical scale up is running into less trouble since internet reliability and performance are improving.

DISK STORAGE REPLICATION

Active (real-time) storage replication is usually implemented by distributing updates of a block device to several physical hard disks. This way, any file system supported by the operating system can be replicated without modification, as the file system code works on a level above the block device driver layer. It is implemented either in hardware (in a disk array controller) or in software (in a device driver). The most basic method is disk mirroring, typical for locally-connected disks. Notably, the storage industry narrows the definitions, so *mirroring* is a local (short-distance) operation. A *replication* is extendable across a computer network, so the disks can be located in physically distant locations. The purpose is to avoid damage done by,

and improve availability in case of local failures or disasters. Typically the above *master-slave* theoretical replication model is applied. The main characteristic of such solutions is handling write operations:

- Synchronous replication - guarantees “zero data loss” by the means of atomic write operation, i.e. write either completes on both sides or not at all. Write is not considered complete until acknowledgement by both local and remote storage. Most applications wait for a write transaction to complete before proceeding with further work, hence overall performance decreases considerably. Inherently, performance drops proportionally to distance, as latency is caused by speed of light. For 10 km distance, the fastest possible roundtrip takes 67 ìs, whereas nowadays a whole local cached write completes in about 10-20 ìs.
  - An often-overlooked aspect of synchronous replication is the fact that failure of *remote* replica, or even just the *interconnection*, stops by definition any and all writes (freezing the local storage system). This is the behaviour that guarantees zero data loss. However, many commercial systems at such potentially dangerous point do not freeze, but just proceed with local writes, losing the desired zero recovery point objective.
- Asynchronous replication - write is considered complete as soon as local storage acknowledges it. Remote storage is updated, but probably with a small lag. Performance is greatly increased, but in case of

losing a local storage, the remote storage is not guaranteed to have the current copy of data and most recent data may be lost.

- Semi-synchronous replication - this usually means that a write is considered complete as soon as local storage acknowledges it and a remote server acknowledges that it has received the write either into memory or to a dedicated log file. The actual remote write is not performed immediately but is performed asynchronously, resulting in better performance than synchronous replication but with increased risk of the remote write failing.
  - o Point-in-time replication - introduces periodic snapshots that are replicated instead of primary storage. If the replicated snapshots are pointer-based, then during replication only the changed data is moved not the entire volume. Using this method, replication can occur over smaller, less expensive bandwidth links such as iSCSI or T1 instead of fiber optic lines.

Most important implementations:

- DRBD module for Linux.
- NetApp SnapMirror
- EMC SRDF
- IBM PPRC and Global Mirror (known together as IBM Copy Services)
- Hitachi TrueCopy
- Hewlett-Packard Continuous Access (HP CA)
- Symantec Veritas Volume Replicator (VVR)

- DataCore SANsymphony & SANmelody
- FalconStor Replication & Mirroring (sub-block heterogeneous point-in-time, async, sync)
- Compellent Remote Instant Replay
- EMC RecoverPoint

FILE BASED REPLICATION

File base replication is replicating files at a logical level rather than replicating at the storage block level. There are many different ways of performing this and unlike storage level replication, they are almost exclusively software solutions.

REAL TIME CAPTURE WITH A FILTER DRIVER

With the use of a Filter Driver (generally a kernel driver), filesystem activity is captured real time by having a process that intercepts calls to the filesystem. This utilises the same type of technology that real time active virus checkers employ. At this level, logical file operations are captured like file open, write, delete etc. The Kernel Driver will then transmit these command to another process, generally over a network to a different machine, which will mimic the operations of the source machine. Like Disk Storage replication, it is possible to perform this both Synchronous and Asynchronously. In Synchronous mode write operations on the source machines arent acknowledged until the destination machine has confirmed it has successfully replicated. Synchronous mode is less common with file replication products although a few solutions exists. File level replication solution yield a few benefits. Firstly because

data is captured at a logical level it can make an informed decision on whether to replicate based on the location of the file and the type of file. Hence unlike Disk Storage Replication where a whole volume needs to be replicated, file replication products have the ability to exclude cache, temporary files or parts of a filesystem that hold no business value. This can substencially reduce the amount of data sent from the source machine as well as decrease the storage burden on the destination machine. A further benefit to decreasing bandwidth is the data transmitted can be more granular than Disk Storage Replication. If an application writes 100byte, only the 100bytes is transmitted converse to a complete disk block which is generally 4k. On the negative side, because this is a software only solution, a hardware solution employing Disk Storage Replication will provide less impact and provide greater throughput to file level replication with a filter driver.

Notable implementations:

- Cofio Software AIMstor Replication
- Double-Take Software Availability

FILESYSTEM JOURNAL

In many ways working like a database journal, many filesystems have the ability to journal their activity. This can be used to playback events near real-time. That is periodically send the latest file activity journal to the destination. The main downside is that because it is not trully real-time, there is greater exposure to data loss should the primary source fail. Notable implementations:

- Microsoft DPM

  This type of replication is performed by capturing file system activity or with interaction with the filesystem.

BATCH REPLICATION

This is the process of comparing the source and destination filesystems and ensuring that the destination matches the source. The key benefit is that such solutions are generally free or inexpensive. The downside is that the process of synchronizing them is quite system intensive and consequently this process is generally run infrequently.

Notable implementations:

- rsync

DISTRIBUTED SHARED MEMORY REPLICATION

Another example of using replication appears in distributed shared memory systems, where it may happen that many nodes of the system share the same page of the memory - which usually means, that each node has a separate copy (replica) of this page.

PRIMARY-BACKUP AND MULTI-PRIMARY REPLICATION

Many classical approaches to replication are based on a primary/backup model where one device or process has unilateral control over one or more other processes or devices. For example, the primary might perform some computation, streaming a log of updates to a backup (standby) process, which can then take over if the primary fails. This approach is the most common one for replicating databases, despite

the risk that if a portion of the log is lost during a failure, the backup might not be in a state identical to the one the primary was in, and transactions could then be lost. A weakness of primary/backup schemes is that in settings where both processes could have been active, only one is actually performing operations. We're gaining fault-tolerance but spending twice as much money to get this property. For this reason, starting in the period around 1985, the distributed systems research community began to explore alternative methods of replicating data. An outgrowth of this work was the emergence of schemes in which a group of replicas could cooperate, with each process backup up the others, and each handling some share of the workload.

Jim Gray, a towering figure within the database community, analyzed multi-primary replication schemes under the transactional model and ultimately published a widely cited paper skeptical of the approach ("The Dangers of Replication and a Solution"). In a nutshell, he argued that unless data splits in some natural way so that the database can be treated as n disjoint sub-databases, concurrency control conflicts will result in seriously degraded performance and the group of replicas will probably slow down as a function of n. Indeed, he suggests that the most common approaches are likely to result in degradation that scales as $O(n^3)$. His solution, which is to partition the data, is only viable in situations where data actually has a natural partitioning key.

The situation is not always so bleak. For example, in the 1985-1987 period, the virtual synchrony model was proposed and emerged as a widely adopted standard (it was used in

the Isis Toolkit, Horus, Transis, Ensemble, Totem, Spread, C-Ensemble, Phoenix and Quicksilver systems, and is the basis for the CORBA fault-tolerant computing standard; the model is also used in IBM Websphere to replicate business logic and in Microsoft's Windows Server 2008 enterprise clustering technology). Virtual synchrony permits a multi-primary approach in which a group of processes cooperate to parallelize some aspects of request processing. The scheme can only be used for some forms of in-memory data, but when feasible, provides linear speedups in the size of the group. A number of modern products support similar schemes. For example, the Spread Toolkit supports this same virtual synchrony model and can be used to implement a multi-primary replication scheme; it would also be possible to use C-Ensemble or Quicksilver in this manner. WANdisco permits active replication where every node on a network is an exact copy or replica and hence every node on the network is active at one time; this scheme is optimized for use in a wide area network.

7

Data Storage Device

A data storage device is a device for recording (storing) information (data). Recording can be done using virtually any form of energy, spanning from manual muscle power in handwriting, to acoustic vibrations in phonographic recording, to electromagnetic energy modulating magnetic tape and optical discs.

A storage device may hold information, process information, or both. A device that only holds information is a recording medium. Devices that process information (data storage equipment) may either access a separate portable (removable) recording medium or a permanent component to store and retrieve information.

Electronic data storage is storage which requires electrical power to store and retrieve that data. Most storage devices that do not require vision and a brain to read data fall into this category. Electromagnetic data may be stored in either

an analog or digital format on a variety of media. This type of data is considered to be electronically encoded data, whether or not it is electronically stored in a semiconductor device, for it is certain that a semiconductor device was used to record it on its medium.

Most electronically processed data storage media (including some forms of computer data storage) are considered permanent (non-volatile) storage, that is, the data will remain stored when power is removed from the device. In contrast, most *electronically stored* information within most types of semiconductor (computer chips) microcircuits are volatile memory, for it vanishes if power is removed.

With the exception of barcodes and OCR data, electronic data storage is easier to revise and may be more cost effective than alternative methods due to smaller physical space requirements and the ease of replacing (rewriting) data on the same medium. However, the durability of methods such as printed data is still superior to that of most electronic storage media. The durability limitations may be overcome with the ease of duplicating (backing-up) electronic data.

TERMINOLOGY

Devices that are not used exclusively for recording (e.g. hands, mouths, musical instruments) and devices that are intermediate in the storing/retrieving process (e.g. eyes, ears, cameras, scanners, microphones, speakers, monitors, video projectors) are not usually considered storage devices. Devices that are exclusively for recording (e.g. printers), exclusively for reading (e.g. barcode readers), or devices that

process only one form of information (e.g. phonographs) may or may not be considered storage devices. In computing these are known as input/output devices. All information is data. However, not all data is information. Many data storage devices are also media players. Any device that can store and playback multimedia may also be considered a media player such as in the case with the HDD media player. Designated hard drives are used to play saved or streaming media on home entertainment systems.

TRENDS

International Data Corporation estimated that the total amount of digital data was 281 billion gigabytes in 2007, and had for the first time exceeded the amount of storage.

DATA STORAGE EQUIPMENT

Any input/output equipment may be considered data storage equipment if it writes to and reads from a data storage medium. Data storage equipment uses either:

- portable methods (easily replaced),
- semi-portable methods requiring mechanical disassembly tools and/or opening a chassis, or
- inseparable methods meaning loss of memory if disconnected from the unit.

The following are examples of those methods:

PORTABLE METHODS

- Hand crafting
- Flat surface

  - Printmaking
  - Photographic
- Fabrication
  - Automated assembly
  - Textile
  - Molding
  - Solid freeform fabrication
- Cylindrical accessing
- Memory card reader/drive
- Tape drive
  - Mono reel or reel-to-reel
  - Compact Cassette player/recorder
- Disk accessing
  - Disk drive
  - Disk enclosure
- Cartridge accessing/connecting (tape/disk/circuitry)
- Peripheral networking
- Flash memory devices

SEMI-PORTABLE METHODS

- Hard disk drive
- Circuitry with non-volatile RAM

INSEPARABLE METHODS

- Circuitry with volatile RAM
- Neurons

RECORDING MEDIUM

A recording medium is a physical material that holds data expressed in any of the existing recording formats.

With electronic media, the data and the recording medium is sometimes referred to as "software" despite the more common use of the word to describe computer software. With (traditional art) static media, art materials such as crayons may be considered both equipment and medium as the wax, charcoal or chalk material from the equipment becomes part of the surface of the medium.

Some recording media may be temporary either by design or by nature. Volatile organic compounds may be used to preserve the environment or to purposely make data expire over time. Data such as smoke signals or skywriting are temporary by nature. Depending on the volatility, a gas (e.g. atmosphere, smoke) or a liquid surface such as a lake would be considered a temporary recording medium if at all.

ANCIENT AND TIMELESS EXAMPLES

- Optical
  - o Any object visible to the eye, used to mark a location such as a, stone, flag or skull.
  - o Any crafting material used to form shapes such as clay, wood, metal, glass, wax or quipu.
  - o Any hard surface that could hold carvings.
  - o Any branding surface that would scar under intense heat (chiefly for livestock or humans).
  - o Any marking substance such as paint, ink or chalk.
  - o Any surface that would hold a marking substance such as, papyrus, paper, skin.
- Chemical
  - o RNA

- o DNA
- o Pheromone

MODERN EXAMPLES BY ENERGY USED

- Chemical
  - o Dipstick
- Thermodynamic
  - o Thermometer
- Photochemical
  - o Photographic film
- Mechanical
  - o Pins and holes
    - - Punched card
    - - Paper tape
    - - Music roll
    - - Musical box cylinder or disk
  - o Grooves
    - - Phonograph cylinder
    - - Gramophone record
    - - Dictabelt (groove on plastic belt)
    - - Capacitance Electronic Disc
- Magnetic storage
  - o Wire recording (stainless steel wire)
  - o Magnetic tape
  - o Drum memory (magnetic drum)
  - o Floppy disk
- Optical storage
  - o Optical jukebox

  - Photographic paper
  - X-ray
  - Microform
  - Hologram
  - Projected transparency
  - Optical disc
  - Magneto-optical drive
  - Holographic data storage
  - 3D optical data storage
- Electrical
  - Semiconductor used in volatile RAM microchips
  - Floating-gate transistor used in non-volatile memory cards

MODERN EXAMPLES BY SHAPE

A typical way to classify data storage media is to consider its shape and type of movement (or non-movement) relative to the read/write device(s) of the storage apparatus as listed:

- Paper card storage
  - Punched card (mechanical)
- Cams and tracers (pipe organ combination-action memory memorizing stop selections)
- Tape storage (long, thin, flexible, linearly moving bands)
  - Paper tape (mechanical)
  - Magnetic tape (a tape passing one or more *read/write/erase heads*)
- Disk storage (flat, round, rotating object)

  - Gramophone record (used for distributing some 1980s home computer programmes) (mechanical)
  - Floppy disk, ZIP disk (removable) (magnetic)
  - Holographic
  - Optical disc such as CD, DVD, Blu-ray Disc
  - Minidisc
  - Hard disk drive (magnetic)
- Magnetic bubble memory
- Flash memory/memory card (solid state semiconductor memory)
  - xD-Picture Card
  - MultiMediaCard
  - USB flash drive (also known as a "thumb drive" or "keydrive")
  - SmartMedia
  - CompactFlash I and II
  - Secure Digital
  - Sony Memory Stick (Std/Duo/PRO/MagicGate versions)
  - Solid-state drive

Bekenstein (2003) foresees that miniaturization might lead to the invention of devices that store bits on a single atom.

WEIGHT AND VOLUME

Especially for carrying around data, the weight and volume per MB are relevant. They are quite large for written and printed paper compared with modern electronic media. On the other hand, written and printer paper do not require

(the weight and volume of) reading equipment, and handwritten edits only require simple writing equipment, such as a pen.

With mobile data connections the data need not be carried around to have them available.

8

USB Flash Drive

A USB flash drive consists of a flash memory data storage device integrated with a USB (Universal Serial Bus) interface. USB flash drives are typically removable and rewritable, and physically much smaller than a floppy disk. Most weigh less than 30 g (1 oz). Storage capacities in 2010 can be as large as 256 GB with steady improvements in size and price per capacity expected. Some allow 1 million write or erase cycles and offer a 10-year shelf storage time.

USB flash drives are often used for the same purposes for which floppy disks or CD-ROMs were used. They are smaller, faster, have thousands of times more capacity, and are more durable and reliable because of their lack of moving parts. Until approximately 2005, most desktop and laptop computers were supplied with floppy disk drives, but floppy disk drives have been abandoned in favour of USB ports. USB Flash drives use the USB mass storage standard,

supported natively by modern operating systems such as Linux, Mac OS X, Windows, and other Unix-like systems.

USB drives with USB 2.0 support can store more data and transfer faster than a much larger optical disc drives like CD-RW or DVD-RW drives and can be read by many other systems such as the Xbox 360, PlayStation 3, DVD players and in some upcoming mobile smartphones. Nothing moves mechanically in a flash drive; the term *drive* persists because computers read and write flash-drive data using the same system commands as for a mechanical disk drive, with the storage appearing to the computer operating system and user interface as just another drive. Flash drives are very robust mechanically. A flash drive consists of a small printed circuit board carrying the circuit elements and a USB connector, insulated electrically and protected inside a plastic, metal, or rubberized case which can be carried in a pocket or on a key chain, for example. The USB connector may be protected by a removable cap or by retracting into the body of the drive, although it is not likely to be damaged if unprotected. Most flash drives use a standard type-A USB connection allowing plugging into a port on a personal computer, but drives for other interfaces also exist. USB flash drives draw power from the computer via external USB connection. Some devices combine the functionality of a digital audio player with USB flash storage; they require a battery only when used to play music.

TECHNOLOGY

Flash memory combines a number of older technologies, with lower cost, lower power consumption and small size

made possible by advances in microprocessor technology. The memory storage was based on earlier EPROM and EEPROM technologies. These had very limited capacity, were very slow for both reading and writing, required complex high-voltage drive circuitry, and could only be re-written after erasing the entire contents of the chip. Hardware designers later developed EEPROMs with the erasure region broken up into smaller "fields" that could be erased individually without affecting the others. Altering the contents of a particular memory location involved copying the entire field into an off-chip buffer memory, erasing the field, modifying the data as required in the buffer, and re-writing it into the same field. This required considerable computer support, and PC-based EEPROM flash memory systems often carried their own dedicated microprocessor system. Flash drives are more or less a miniaturized version of this.

The development of high-speed serial data interfaces such as USB made semiconductor memory systems with serially accessed storage viable, and the simultaneous development of small, high-speed, low-power microprocessor systems allowed this to be incorporated into extremely compact systems. Serial access requires far fewer electrical connections for the memory chips than does parallel access, which has simplified the manufacture of multi-gigabyte drives.

Computers access modern flash memory systems very much like hard disk drives, where the controller system has full control over where information is actually stored. The actual EEPROM writing and erasure processes are, however,

still very similar to the earlier systems described above. Many low-cost MP3 players simply add extra software and a battery to a standard flash memory control microprocessor so it can also serve as a music playback decoder. Most of these players can also be used as a conventional flash drive, for storing files of any type.

HISTORY

FIRST COMMERCIAL PRODUCT

Trek Technology and IBM began selling the first USB flash drives commercially in 2000. The Singaporean Trek Technology sold a model under the brand name "ThumbDrive", and IBM marketed the first such drives in North America with its product named the "DiskOnKey" which was developed and manufactured by the Israeli company M-Systems. IBM's USB flash drive became available on December 15, 2000, and had a storage capacity of 8 MB, more than five times the capacity of the then-common floppy disks.

In 2000 Lexar introduced a Compact Flash (CF) card with a USB connection, and a companion card read/writer and USB cable that eliminated the need for a USB hub. Both Trek Technology and Netac Technology have tried to protect their patent claims. Trek won a Singaporean suit, but a court in the United Kingdom revoked one of Trek's UK patents. While Netac Technology has brought lawsuits against PNY Technologies, Lenovo, aigo, Sony, and Taiwan's Acer and Tai Guen Enterprise Co, most companies that

manufacture USB flash drives do so without regard for Trek and Netac's patents.

SECOND GENERATION

Modern flash drives have USB 2.0 connectivity. However, they do not currently use the full 480 Mbit/s (60MB/s) which the USB 2.0 Hi-Speed specification supports because of technical limitations inherent in NAND flash. The fastest drives currently available use a dual channel controller, although they still fall considerably short of the transfer rate possible from a current generation hard disk, or the maximum high speed USB throughput. File transfer speeds vary considerably and should be checked before purchase. Speeds may be given in Mbyte per second, Mbit per second or optical drive multipliers such as "180X" (180 times 150 KiB per second). Typical fast drives claim to read at up to 30 megabytes/s (MB/s) and write at about half that speed. This is about 20 times faster than older "USB full speed" devices which are limited to a maximum speed of 12 Mbit/s (1.5 MB/s).

DESIGN AND IMPLEMENTATION

One end of the device is fitted with a single male type-A USB connector. Inside the plastic casing is a small printed circuit board. Mounted on this board is some power circuitry and a small number of surface-mounted integrated circuits (ICs). Typically, one of these ICs provides an interface to the USB port, another drives the onboard memory, and the other is the flash memory. Drives typically use the USB mass storage device class to communicate with the host.

ESSENTIAL COMPONENTS

There are typically four parts to a flash drive:

- Male type-A USB connector – provides a physical interface to the host computer.
- USB mass storage controller – implements the USB host controller. The controller contains a small microcontroller with a small amount of on-chip ROM and RAM.
- NAND flash memory chip – stores data. NAND flash is typically also used in digital cameras.
- Crystal oscillator – produces the device's main 12 MHz clock signal and controls the device's data output through a phase-locked loop.

ADDITIONAL COMPONENTS

The typical device may also include:

- Jumpers and test pins – for testing during the flash drive's manufacturing or loading code into the microprocessor.
- LEDs – indicate data transfers or data reads and writes.
- Write-protect switches – Enable or disable writing of data into memory.
- Unpopulated space – provides space to include a second memory chip. Having this second space allows the manufacturer to use a single printed circuit board for more than one storage size device.
- USB connector cover or cap – reduces the risk of damage, prevents the ingress of fluff or other

contaminants, and improves overall device appearance. Some flash drives use retractable USB connectors instead. Others have a swivel arrangement so that the connector can be protected without removing anything.

- Transport aid – the cap or the body often contains a hole suitable for connection to a key chain or lanyard. Connecting the cap, rather than the body, can allow the drive itself to be lost.
- Some drives offer expandable storage via an internal memory card slot, much like a memory card reader.

SIZE AND STYLE OF PACKAGING

Some manufacturers differentiate their products by using elaborate housings, which are often bulky and make the drive difficult to connect to the USB port. Because the USB port connectors on a computer housing are often closely spaced, plugging a flash drive into a USB port may block an adjacent port. Such devices may only carry the USB logo if sold with a separate extension cable.

USB flash drives have been integrated into other commonly carried items such as watches, pens, and even the Swiss Army Knife; others have been fitted with novelty cases such as toy cars or LEGO bricks. The small size, robustness and cheapness of USB flash drives make them an increasingly popular peripheral for case modding. Heavy or bulky flash drive packaging can make for unreliable operation when plugged directly into a USB port; this can be relieved by a USB extension cable. Such cables are USB-compatible but do not conform to the USB standard.

FILE SYSTEM

Most flash drives ship preformatted with the FAT or FAT 32 file system. The ubiquity of this file system allows the drive to be accessed on virtually any host device with USB support.

Also, standard FAT maintenance utilities (e.g. ScanDisk) can be used to repair or retrieve corrupted data. However, because a flash drive appears as a USB-connected hard drive to the host system, the drive can be reformatted to any file system supported by the host operating system.

Defragmenting: Flash drives can be defragmented, but this brings little advantage as there is no mechanical head that moves from fragment to fragment. Flash drives often have a large internal sector size, so defragmenting means accessing fewer sectors. Defragmenting shortens the life of the drive by making many unnecessary writes.

Even Distribution: Some file systems are designed to distribute usage over an entire memory device without concentrating usage on any part (e.g. for a directory); this even distribution prolongs the life of simple flash memory devices. Some USB flash drives have this functionality built into the software controller to prolong device life, while others do not, therefore the end user should check the specifications of his device prior to changing the file system for this reason.

Hard Drive: Sectors are 512 bytes long, for compatibility with hard drives, and the first sector can contain a Master Boot Record and a partition table. Therefore USB flash units can be partitioned as hard drives.

LONGEVITY

Barring physical destruction of the drive, the memory or USB connector of a flash drive will eventually fail. SLC based memory is good for around 100,000 writes; more commonly used MLC for around 10,000. The USB connector can withstand approximately 1,500 connect/disconnect cycles.

FAKE PRODUCTS

Fake USB flash drives are sometimes sold, claiming to have higher capacities than they actually have. These are typically low capacity USB drives which are modified so that they emulate larger capacity drives (e.g. a 2 GB drive being marketed as an 8 GB drive). When plugged into a computer, they report themselves as being the larger capacity they were sold as, but when data is written to them, either the write fails, the drive freezes up, or it overwrites existing data. Software tools exist to check and detect fake USB drives. In some cases it is possible to repair these devices to remove the false capacity information and use them normally.

USES

PERSONAL DATA TRANSPORT

The most common use of flash drives is to transport and store personal files such as documents, pictures and videos. Individuals also store medical alert information on MedicTag flash drives for use in emergencies and for disaster preparation.

SECURE STORAGE OF DATA, APPLICATION AND SOFTWARE FILES

With wide deployment(s) of flash drives being used in various environments (secured or otherwise), the issue of data and information security remains of the utmost importance. The use of biometrics and encryption is becoming the norm with the need for increased security for data; OTFE systems are particularly useful in this regard, as they can transparently encrypt large amounts of data. In some cases a Secure USB Drive may use a hardware-based encryption mechanism that uses a hardware module instead of software for strongly encrypting data. IEEE 1667 is an attempt to create a generic authentication platform for USB drives and enjoys the support of Microsoft with support in Windows 7 and in Windows Vista Service Pack 2 with a hotfix.

SYSTEM ADMINISTRATION

Flash drives are particularly popular among system and network administrators, who load them with configuration information and software used for system maintenance, troubleshooting, and recovery. They are also used as a means to transfer recovery and antivirus software to infected PCs, allowing a portion of the host machine's data to be archived. As the drives have increased in storage space, they have also replaced the need to carry a number of CD ROMs and installers which were needed when reinstalling or updating a system.

APPLICATION CARRIERS

Flash drives are used to carry applications that run on the host computer without requiring installation. While any

standalone application can in principle be used this way, many programmes store data, configuration information, etc. on the hard drive and registry of the host computer The U3 company works with drive makers (parent company SanDisk as well as others) to deliver custom versions of applications designed for Microsoft Windows from a special flash drive; U3-compatible devices are designed to autoload a menu when plugged into a computer running Windows. Applications must be modified for the U3 platform not to leave any data on the host machine. U3 also provides a software framework for independent software vendors interested in their platform. Ceedo is an alternative product with the key difference that it does not require Windows applications to be modified in order for them to be carried and run on the drive. Similarly, other application virtualization solutions and portable application creators, such as VMware ThinApp (for Windows) or RUNZ (for Linux) can be used to run software from a flash drive without installation. In October 2010, Apple Inc. released their newest iteration of the MacBook Air, which had the system's restore files contained on a USB card drive rather than the traditional install CDs due to the Air not coming with an optical drive. A wide range of portable applications which are all free of charge, and able to run off a computer running Windows without storing anything on the host computer's drives or registry, can be found in the list of portable software.

COMPUTER FORENSICS AND LAW ENFORCEMENT

A recent development for the use of a USB Flash Drive as an application carrier is to carry the Computer Online

Forensic Evidence Extractor (COFEE) application developed by Microsoft. COFEE is a set of applications designed to search for and extract digital evidence on computers confiscated from suspects. Forensic software should not alter the information stored on the computer being examined in any way; other forensic suites run from CD-ROM or DVD-ROM, but cannot store data on the media they are run from (although they can write to other attached devices such as external drives or memory sticks).

BOOTING OPERATING SYSTEMS

Most current PC firmware permits booting from a USB drive, allowing the launch of an operating system from a bootable flash drive. Such a configuration is known as a Live USB. Original flash memory designs had very limited estimated lifetimes. The failure mechanism for flash memory cells is analogous to a metal fatigue mode; the device fails by refusing to write new data to specific cells that have been subject to many read-write cycles over the device's lifetime. Originally, this potential failure mode limited the use of "live USB" system to special purpose applications or temporary tasks, such as:

- Loading a minimal, hardened kernel for embedded applications (e.g. network router, firewall).
- Bootstrapping an operating system install or disk cloning operation, often across a network.
- Maintenance tasks, such as virus scanning or low-level data repair, without the primary host operating system loaded.

As of 2011, newer flash memory designs have much higher estimated lifetimes. Several manufacturers are now offering warranties of 5 years, or more. That should make the device more attractive for more applications. By reducing the probability of the device's premature failure, flash memory devices can now be considered for use where a magnetic disk would normally have been required. Flash drives have also experienced an exponential growth in their storage capacity over time (following the Moore's Law growth curve). As of 2011, single packaged devices with capacities of 64GB are readily available, and devices with 8GB capacity are very economical. Storage capacities in this range have traditionally been considered to offer adequate space, because they allow enough space for both the operating system software and some free space for the user's data.

WINDOWS VISTA AND WINDOWS 7 READYBOOST

In Windows Vista and Windows 7, the ReadyBoost feature allows use of flash drives (up to 4 GB in the case of Windows Vista) to augment operating system memory

AUDIO PLAYERS

Many companies make small solid-state digital audio players, essentially producing flash drives with sound output and a simple user interface. Examples include the Creative MuVo, Philips GoGear and the first generation iPod shuffle. Some of these players are true USB flash drives as well as music players; others do not support general-purpose data storage. Many of the smallest players

are powered by a permanently fitted rechargeable battery, charged from the USB interface.

MUSIC STORAGE AND MARKETING

Digital audio files can be transported from one computer to another like any other file, and played on a compatible media player (with caveats for DRM-locked files). In addition, many home Hi-Fi and car stereo head units are now equipped with a USB port. This allows a USB flash drive containing media files in a variety of formats to be played directly on devices which support the format. Artists have sold or given away USB flash drives, with the first instance believed to be in 2004 when the German band WIZO released the "Stick EP", only as a USB drive. In addition to five high-bitrate MP3s, it also included a video, pictures, lyrics, and guitar tablature. Subsequently artists including Kanye West, Nine Inch Nails, Kylie Minogue and Ayumi Hamasaki have released music and promotional material on USB flash drives. In 2009 a USB drive holding fourteen remastered Beatles albums in both FLAC and MP3 was released.

IN ARCADES

In the arcade game *In the Groove* and more commonly *In The Groove 2*, flash drives are used to transfer high scores, screenshots, dance edits, and combos throughout sessions. As of software revision 21 (R21), players can also store custom songs and play them on any machine on which this feature is enabled. While use of flash drives is common, the drive must be Linux compatible. In the arcade games *Pump it Up NX2* and *Pump it Up NXA*, a special

produced flash drive is used as a "save file" for unlocked songs, as well as progressing in the WorldMax and Brain Shower sections of the game. In the arcade game *Dance Dance Revolution X*, an exclusive USB flash drive was made by Konami for the purpose of the link feature from its Sony PlayStation 2 counterpart. However, any USB flash drives can be used in this arcade game.

BRAND AND PRODUCT PROMOTION

The availability of inexpensive flash drives has enabled them to be used for promotional and marketing purposes, particularly within technical and computer-industry circles (e.g. technology trade shows). They may be given away for free, sold at less than wholesale price, or included as a bonus with another purchased product. Usually, such drives will be custom-stamped with a company's logo, as a form of advertising to increase mind share and brand awareness. The drive may be a blank drive, or preloaded with graphics, documentation, web links, Flash animation or other multimedia, and free or demonstration software. Some preloaded drives are read-only while others are configured with both read-only and user-writable segments, such dual-partition drives are more expensive. Flash drives can be set up to automatically launch stored presentations, websites, articles, and any other software immediately on insertion of the drive using the Microsoft Windows AutoRun feature. Autorunning software this way does not work on all computers, and is normally disabled by security-conscious users.

BACKUP

Some value-added resellers are now using a flash drive as part of small-business turnkey solutions (e.g. point-of-sale systems). The drive is used as a backup medium: at the close of business each night, the drive is inserted, and a database backup is saved to the drive. Alternatively, the drive can be left inserted through the business day, and data regularly updated. In either case, the drive is removed at night and taken offsite.

- This is simple for the end-user, and more likely to be done;
- The drive is small and convenient, and more likely to be carried off-site for safety;
- The drives are less fragile mechanically and magnetically than tapes;
- The capacity is often large enough for several backup images of critical data;
- And flash drives are cheaper than many other backup systems.

It is also easy to lose these small devices, and easy for people without a right to data to take illicit backups.

MERITS AND AND DEMERITS

MERITS

Data stored on flash drives is impervious to scratches and dust, and flash drives are mechanically very robust making them suitable for transporting data from place to place and keeping it readily at hand. Most personal

computers support USB as of 2010. Flash drives also store data densely compared to many removable media. In mid-2009, 256 GB drives became available, with the ability to hold many times more data than a DVD or even a Blu-ray disc. Compared to hard drives, flash drives use little power, have no fragile moving parts, and for most capacities are small and light.

Flash drives implement the USB mass storage device class so that most modern operating systems can read and write to them without installing device drivers. The flash drives present a simple block-structured logical unit to the host operating system, hiding the individual complex implementation details of the various underlying flash memory devices. The operating system can use any file system or block addressing scheme. Some computers can boot up from flash drives. Specially manufactured flash drives are available that have a tough rubber or metal casing designed to be waterproof and virtually "unbreakable". These flash drives retain their memory even after being submerged in water, even through a machine wash. Leaving such a flash drive out to dry completely before allowing current to run through it has been known to result in a working drive with no future problems. Channel Five's *Gadget Show* cooked one of these flash drives with propane, froze it with dry ice, submerged it in various acidic liquids, ran over it with a jeep and fired it against a wall with a mortar. A company specializing in recovering lost data from computer drives managed to recover all the data on the drive. All data on the other removable storage devices tested, using optical or magnetic technologies, were destroyed.

DEMERITS

Like all flash memory devices, flash drives can sustain only a limited number of write and erase cycles before the drive fails. This should be a consideration when using a flash drive to run application software or an operating system. To address this, as well as space limitations, some developers have produced special versions of operating systems (such as Linux in Live USB) or commonplace applications (such as Mozilla Firefox) designed to run from flash drives. These are typically optimized for size and configured to place temporary or intermediate files in the computer's main RAM rather than store them temporarily on the flash drive. Most USB flash drives do not include a write-protect mechanism, although some have a switch on the housing of the drive itself to keep the host computer from writing or modifying data on the drive. Write-protection makes a device suitable for repairing virus-contaminated host computers without risk of infecting the USB flash drive itself.

A drawback to the small size is that they are easily misplaced, left behind, or otherwise lost. This is a particular problem if the data they contain are sensitive. As a consequence, some manufacturers have added encryption hardware to their drives—although software encryption systems which can be used in conjunction with any mass storage medium achieve the same thing,. Most drives can be attached to keychains, necklaces and lanyards. The USB plug is usually fitted with a removable and easily lost protective cap, or is retractable.

USB flash drives are more expensive per unit of storage than large hard drives, but are less expensive in capacities of a few tens of gigabytes as of 2011. Maximum available capacity is increasing with time, but is less than larger hard drives. This balance is changing, but the rate of change is slowing.

COMPARISON WITH OTHER PORTABLE STORAGE

OPTICAL MEDIA

The various writable and rewritable forms of CD and DVD are portable storage media supported by the vast majority of computers as of 2008. CD-R, DVD-R, and DVD+R can be written to only once, RW varieties up to about 1,000 erase/write cycles, while modern NAND-based flash drives often last for 500,000 or more erase/write cycles. DVD-RAM discs are the most suitable optical discs for data storage involving much rewriting.

Optical storage devices are among the cheapest methods of mass data storage after the hard drive. They are slower than their flash-based counterparts. Standard 12 cm optical discs are larger than flash drives and more subject to damage. Smaller optical media do exist, such as business card CD-Rs which have the same dimensions as a credit card, and the slightly less convenient but higher capacity 8 cm recordable CD/DVDs. The small discs are more expensive than the standard size, and do not work in all drives. Universal Disk Format (UDF) version 1.50 and above

has facilities to support rewritable discs like sparing tables and virtual allocation tables, spreading usage over the entire surface of a disc and maximising life, but many older operating systems do not support this format. Packet-writing utilities such as DirectCD and InCD are available but produce discs that are not universally readable (although based on the UDF standard). The Mount Rainier standard addresses this shortcoming in CD-RW media by running the older file systems on top of it and performing defect management for those standards, but it requires support from both the CD/DVD burner and the operating system. Many drives made today do not support Mount Rainier, and many older operating systems such as Windows XP and below, and Linux kernels older than 2.6.2, do not support it (later versions do). Essentially CDs/DVDs are a good way to record a great deal of information cheaply and have the advantage of being readable by most standalone players, but they are poor at making ongoing small changes to a large collection of information. Flash drives' ability to do this is their major advantage over optical media.

TAPE

The applications of current data tape cartridges hardly overlap those of flash drives: on tape, cost per gigabyte is very low for large volumes, but the individual drives and media are expensive. Media has a very high capacity and very fast transfer speeds, but store data sequentially and is very slow for random seek of data. While disk-based backup is now the primary medium of choice for most companies, tape backup is still popular for taking data off-

site for worst-case scenarios and for very large volumes (more than a few hundreds of TB). See LTO tapes.

FLOPPY DISK

Floppy disk drives are rarely fitted to modern computers and are obsolete for normal purposes, although internal and external drives can be fitted if required. Floppy disks may be the method of choice for transferring data to and from very old computers without USB or booting from floppy disks, and so they are sometimes used to change the firmware on, for example, BIOS chips. Devices with removable storage like older Yamaha music keyboards are also dependent on floppy disks, which require computers to process them. Newer devices are built with USB flash drive support.

FLASH MEMORY CARDS

Flash memory cards, e.g. Secure Digital cards, are available in various formats and capacities, and are used by many consumer devices. However, while virtually all PCs have USB ports, allowing the use of USB flash drives, memory card readers are not commonly supplied as standard equipment (particularly with desktop computers). Although inexpensive card readers are available that read many common formats, this results in two pieces of portable equipment (card plus reader) rather than one.

Some manufacturers, aiming at a "best of both worlds" solution, have produced card readers that approach the size and form of USB flash drives (e.g. Kingston MobileLite, SanDisk MobileMate.) These readers are limited to a specific subset of memory card formats (such as SD, microSD, or

Memory Stick), and often completely enclose the card, offering durability and portability approaching, if not quite equal to, that of a flash drive. Although the combined cost of a mini-reader and a memory card is usually slightly higher than a USB flash drive of comparable capacity, the reader + card solution offers additional flexibility of use, and virtually "unlimited" capacity. An additional advantage of memory cards is that many consumer devices (e.g. digital cameras, portable music players) cannot make use of USB flash drives (even if the device has a USB port) whereas the memory cards used by the devices can be read by PCs with a card reader.

EXTERNAL HARD DISK

Particularly with the advent of USB, external hard disks have become widely available and inexpensive. External hard disk drives currently cost less per gigabyte than flash drives and are available in larger capacities. Some hard drives support alternative and faster interfaces than USB 2.0 (e.g. IEEE 1394 and eSATA). For writes and consecutive sector reads (for example, from an unfragmented file) most hard drives can provide a much higher sustained data rate than current NAND flash memory. Unlike solid-state memory, hard drives are susceptible to damage by shock, e.g., a short fall, vibration, have limitations on use at high altitude, and although they are shielded by their casings, they are vulnerable when exposed to strong magnetic fields.

In terms of overall mass, hard drives are usually larger and heavier than flash drives; however, hard disks sometimes weigh less per unit of storage. Hard disks also suffer from file fragmentation which can reduce access speed.

OBSOLETE DEVICES

Audio tape cassettes and high-capacity floppy disks (e.g. Imation SuperDisk), and other forms of drives with removable magnetic media such as the Iomega Zip and Jaz drives are now largely obsolete and rarely used. There are products in today's market which will emulate these legacy drives for both tape & disk (SCSI1/SCSI2, SASI, Magneto optic, Ricoh ZIP, Jaz, IBM3590/ Fujitsu 3490E and Bernoulli for example) in state of the art Compact Flash storage devices - CF2SCSI.

ENCRYPTION AND SECURITY

As highly portable media, USB flash drives are easily lost or stolen. All USB flash drives can have their contents encrypted using third party disk encryption software, which can often be run directly from the USB drive without installation (for example, FreeOTFE) although some, such as TrueCrypt, require the user to have administrative rights on every computer it's run on. Archiving software can achieve a similar result by creating encrypted ZIP or RAR files. Some USB flash drive manufacturers have produced USB flash drives which use hardware based encryption as part of the design, removing the need for third-party encryption software; though a number of these have been shown to have security problems, and are typically more expensive than software based systems which are available for free.

A minority of flash drives support biometric fingerprinting to confirm the user's identity. As of mid-2005, this was an expensive alternative to standard password protection offered on many new USB flash storage devices. Most fingerprint scanning drives rely upon the host operating system to

validate the fingerprint via a software driver, often restricting the drive to Microsoft Windows computers. However, there are USB drives with fingerprint scanners which use controllers that allow access to protected data without any authentication. Some manufacturers deploy physical authentication tokens in the form of a flash drive. These are used to control access to a sensitive system by containing encryption keys or, more commonly, communicating with security software on the target machine. The system is designed so the target machine will not operate except when the flash drive device is plugged into it. Some of these "PC lock" devices also function as normal flash drives when plugged into other machines.

SECURITY THREATS

Flash drives may present a significant security challenge for some organizations. Their small size and ease of use allows unsupervised visitors or employees to store and smuggle out confidential data with little chance of detection. Both corporate and public computers are vulnerable to attackers connecting a flash drive to a free USB port and using malicious software such as keyboard loggers or packet sniffers. For computers set up to be bootable from a USB drive it is possible to use a flash drive containing a bootable portable operating system to access the files of a computer even if the computer is password protected. The password can then be changed; or it may be possible to crack the password with a password cracking programme, and gain full control over the computer. Encrypting files provides considerable protection against this type of attack. USB

flash drives may also be used deliberately or unwittingly to transfer malware and autorun worms onto a network. Some organizations forbid the use of flash drives, and some computers are configured to disable the mounting of USB mass storage devices by users other than administrators; others use third-party software to control USB usage. The use of software allows the administrator to not only provide a USB lock but also control the use of CD-RW, SD cards and other memory devices. This enables companies with policies forbidding the use of USB flash drives in the workplace to enforce these policies. In a lower-tech security solution, some organizations disconnect USB ports inside the computer or fill the USB sockets with epoxy.

NAMING

By August 2008, "USB flash drive" had emerged as a common term for these devices, and most major manufacturers use similar wording on their packaging, although potentially confusing alternatives (such as Memory Stick or *USB memory key or 'Pen drive'*) still occur. The myriad different brand names and terminology used, in the past and currently, make USB flash drives more difficult for manufacturers to market and for consumers to research. Some commonly-used names actually represent trademarks of particular companies, such as Cruzer, DataTraveler, TravelDrive, ThumbDrive, and Disgo.

CURRENT AND FUTURE DEVELOPMENTS

Semiconductor corporations have worked to reduce the cost of the components in a flash drive by integrating

various flash drive functions in a single chip, thereby reducing the part-count and overall package-cost. Flash drive capacities on the market increase continually. As of 2010, few manufacturers continue to produce models of 1 GB and smaller; and many have started to phase out 2 GB capacity flash memory. High speed has become a standard for modern flash drives and capacities of up to 256 GB have come on the market, as of 2009.

Lexar is attempting to introduce a USB FlashCard, which would be a compact USB flash drive intended to replace various kinds of flash memory cards. Pretec introduced a similar card, which also plugs into every USB port, but is just one quarter the thickness of the Lexar model. Until 2008, SanDisk manufactured a product called SD Plus, which was a SecureDigital card with a USB connector. SanDisk has also introduced a new technology to allow controlled storage and usage of copyrighted materials on flash drives, primarily for use by students. This technology is termed FlashCP.

FLASH DDRIVES FOR NON-USB INTERFACES

The majority of flash drives use USB, but some flash drives use other interfaces, such as IEEE1394 (FireWire), one of their theoretical advantages when compared to USB drives being the minimal latency and CPU utilisation that the IEEE1394 protocol provides, but in practice because of the prevalence of the USB interfaces all IEEE1394-based flash drives that have appeared used old slow flash memory chips and no manufacturer sells IEEE1394 flash drives with modern fast flash memory as of 2009, and the currently

available models go up only to 4 GB, 8 GB or 16 GB, depending on the manufacturer. FireWire flash drives that needs to be connected to FireWire 400 port cannot be connected to a FireWire 800 port without an adaptor and vice-versa.

In late 2008, flash drives that utilize the eSATA interface became available. One advantage that an eSATA flash drive claims over a USB flash drive is increased data throughput, thereby resulting in faster data read and write speeds. However, using eSATA for flash drives also has some disadvantages. The eSATA connector was designed primarily for use with external hard disk drives that often include their own separate power supply. Therefore, unlike USB, an eSATA connector does not provide any usable electrical power other than what is required for signaling and data transfer purposes. This means that an eSATA flash drive still requires an available USB port or some other external source of power to operate it.

Additionally, as of September 2009, eSATA is still a fairly uncommon interface on most home computers, therefore very few systems can currently make use of the increased performance offered via the eSATA interface on such-equipped flash drives. Finally, with the exception of eSATA-equipped laptop computers, most home computers that include one or more eSATA connectors usually locate the ports on the back of the computer case, thus making accessibility difficult in certain situations and complicating insertion and removal of the flash drive.