Computer Memory, Applications and Management

Nikola Zlatanov
Applied Materials
44 PUBLICATIONS **8** CITATIONS
SEE PROFILE

Computer Memory, Applications and Management

Nikola Zlatanov*

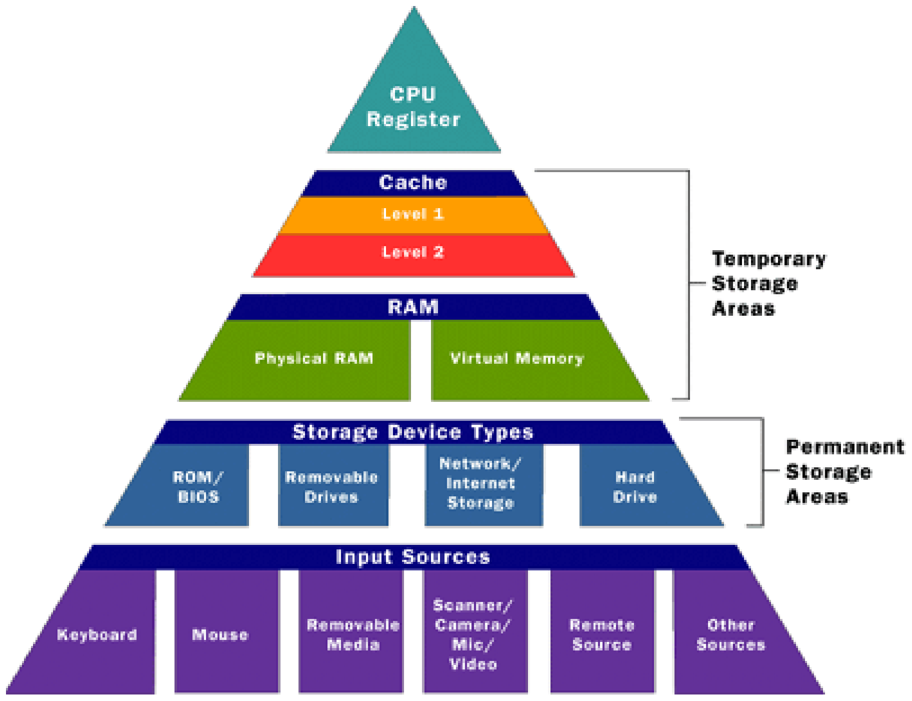
In computing, memory refers to the computer hardware devices used to store information for immediate use in a computer; it is synonymous with the term "primary storage". Computer memory operates at a high speed, for example random-access memory (RAM), as a distinction from storage that provides slow-to-access program and data storage but offers higher capacities. If needed, contents of the computer memory can be transferred to secondary storage, through a memory management technique called "virtual memory". An archaic synonym for memory is store.

The term "memory", meaning "primary storage" or "main memory", is often associated with addressable semiconductor memory, i.e. integrated circuits consisting of silicon-based transistors, used for example as primary storage but also other purposes in computers and other digital electronic devices. There are two main types of semiconductor memory, volatile and non-volatile. Examples of non-volatile memory are flash memory (used as secondary memory) and ROM, PROM, EPROM and EEPROM memory (used for storing firmware such as BIOS). Examples of volatile memory are primary storage, which is typically dynamic random-access memory (DRAM), and fast CPU cache memory, which is typically static random-access memory (SRAM) that is fast but energy-consuming, offering lower memory areal density than DRAM.

Most semiconductor memory is organized into memory cells or bistable flip-flops, each storing one bit (0 or 1). Flash memory organization includes both one bit per memory cell and multiple bits per cell (called MLC, Multiple Level Cell). The memory cells are grouped into words of fixed word length, for example 1, 2, 4, 8, 16, 32, 64 or 128 bit. Each word can be accessed by a binary address of *N* bit, making it possible to store 2 raised by *N* words in the memory. This implies that processor registers normally are not considered as memory, since they only store one word and do not include an addressing mechanism. Typical secondary storage devices are hard disk drives and solid-state drives.

It is amazing how many different types of electronic memory you encounter in daily life. Many of them have become an integral part of our vocabulary: RAM, ROM, Cache, Dynamic RAM, Static RAM, Flash memory, Memory Sticks, Virtual memory, Video memory, BIOS.

Computer Memory Basics



Although memory is technically any form of electronic storage, it is used most often to identify fast, temporary forms of storage. If your computer's **CPU** had to constantly access the **hard drive** to retrieve every piece of data it needs, it would operate very slowly. When the information is kept in memory, the CPU can access it much more quickly. Most forms of memory are intended to store data temporarily.

As you can see in the diagram above, the CPU accesses memory according to a distinct hierarchy. Whether it comes from permanent storage (the hard drive) or input (the **keyboard**), most data goes in **random access memory** (**RAM**) first. The CPU then stores pieces of data it will need to access, often in a **cache**, and maintains certain special instructions in the **register**. We'll talk about cache and registers later.

All of the components in your computer, such as the CPU, the hard drive and the **operating system**, work together as a team, and memory is one of the most essential parts of this team. From the moment you turn your computer on until the time you shut it down, your CPU is constantly using memory. Let's take a look at a typical scenario:

- You turn the computer on.
- The computer loads data from read-only memory (**ROM**) and performs a power-on self-test (POST) to make sure all the major components are functioning properly. As part of this test, the memory controller checks all of the memory addresses with a quick read/write operation to ensure that there are no errors in the memory chips. Read/write means that data is written to a **bit** and then read from that bit.
- The computer loads the basic input/output system (**BIOS**) from ROM. The BIOS provides the most basic information about storage devices, boot sequence, security, Plug and Play (auto device recognition) capability and a few other items.
- The computer loads the operating system (OS) from the hard drive into the system's RAM. Generally, the critical parts of the **operating system** are maintained in RAM as long as the computer is on. This allows the CPU to have immediate access to the operating system, which enhances the performance and functionality of the overall system.
- When you open an application, it is loaded into **RAM**. To conserve RAM usage, many applications load only the essential parts of the program initially and then load other pieces as needed.
- After an application is loaded, any files that are opened for use in that application are loaded into RAM.
- When you save a file and close the application, the file is written to the specified storage device, and then it and the application are purged from RAM.

In the list above, every time something is loaded or opened, it is placed into RAM. This simply means that it has been put in the computer's temporary storage area so that the CPU can access that information more easily. The CPU requests the data it needs from RAM, processes it and writes new data back to RAM in a continuous cycle. In most computers, this shuffling of data between the CPU and RAM happens millions of times every second. When an application is closed, it and any accompanying files are usually purged (deleted) from RAM to make room for new data. If the changed files are not saved to a permanent storage device before being purged, they are lost.

One common question about desktop computers that comes up all the time is, "Why does a computer need so many memory systems?"

Types of Computer Memory

A typical computer has:

Level 1 and level 2 caches

Normal system RAM

Virtual memory

A hard disk

Fast, powerful CPUs need quick and easy access to large amounts of data in order to maximize their performance. If the CPU cannot get to the data it needs, it literally stops and waits for it. Modern CPUs running at speeds of about **1 GigaHertz** can consume massive amounts of data -- potentially billions of **bytes** per second. The problem that computer designers face is that memory that can keep up with a 1-gigahertz CPU is extremely **expensive** -- much more expensive than anyone can afford in large quantities.

Computer designers have solved the cost problem by "**tiering**" memory -- using expensive memory in small quantities and then backing it up with larger quantities of less expensive memory.

The cheapest form of read/write memory in wide use today is the hard disk. Hard disks provide large quantities of inexpensive, permanent storage. You can buy hard disk space for pennies per megabyte, but it can take a good bit of time (approaching a second) to read a megabyte off a hard disk. Because storage space on a hard disk is so cheap and plentiful, it forms the final stage of a CPUs memory hierarchy, called virtual memory.

The next level of the hierarchy is **RAM**. We discuss RAM in detail in **How RAM Works**, but several points about RAM are important here.

The **bit size** of a CPU tells you how many bytes of information it can access from RAM at the same time. For example, a 16-bit CPU can process 2 bytes at a time (1 byte = 8 bits, so 16 bits = 2 bytes), and a 64-bit CPU can process 8 bytes at a time.

Megahertz (MHz) is a measure of a CPU's processing speed, or clock cycle, in millions per second. So, a 32-bit 800-MHz Pentium III can potentially process 4 bytes simultaneously, 800 million times per second (possibly more based on pipelining)! The goal of the memory system is to meet those requirements.

A computer's system RAM alone is not fast enough to match the speed of the CPU. That is why you need a **cache** (discussed later). However, the faster RAM is, the better. Most chips today operate with a cycle rate of 50 to 70 nanoseconds. The read/write speed is typically a function of the type of RAM used, such as DRAM, SDRAM, RAMBUS. We will talk about these various types of memory later.

First, let's talk about system RAM.

System RAM

System RAM speed is controlled by **bus width** and **bus speed**. Bus width refers to the number of bits that can be sent to the CPU simultaneously, and bus speed refers to the number of times a group of bits can be sent each second. A **bus cycle** occurs every time data travels from memory to the CPU. For example, a 100-MHz 32-bit bus is theoretically capable of sending 4 bytes (32 bits divided by 8 = 4 bytes) of data to the CPU 100 million times per second, while a 66-MHz 16-bit bus can send 2 bytes of data 66 million times per second. If you do the math, you'll find that simply changing the bus width from 16 bits to 32 bits and the speed from 66 MHz to 100 MHz in our example allows for three times as much data (400 million bytes versus 132 million bytes) to pass through to the CPU every second.

In reality, RAM doesn't usually operate at optimum speed. **Latency** changes the equation radically. Latency refers to the number of clock cycles needed to read a bit of information. For example, RAM rated at 100 MHz is capable of sending a bit in 0.00000001 seconds, but may take 0.00000005 seconds to start the read process for the first bit. To compensate for latency, CPUs uses a special technique called **burst mode**.

Burst mode depends on the expectation that data requested by the CPU will be stored in **sequential memory cells**. The memory controller anticipates that whatever the CPU is working on will continue to come from this same series of memory addresses, so it reads several consecutive bits of data together. This means that only the first bit is subject to the full effect of latency; reading successive bits takes significantly less time. The **rated burst mode** of memory is normally expressed as four numbers separated by dashes. The first number tells you the number of clock cycles needed to begin a read operation; the second, third and fourth numbers tell you how many cycles are needed to read each consecutive bit in the row, also known as the **wordline**. For example: 5-1-1-1 tells you that it takes five cycles to read the first bit and one cycle for each bit after that. Obviously, the lower these numbers are, the better the performance of the memory.

Burst mode is often used in conjunction with **pipelining**, another means of minimizing the effects of latency. Pipelining organizes data retrieval into a sort of assembly-line process. The memory controller simultaneously reads one or more words from memory, sends the current word or words to the CPU and writes one or more words to memory cells. Used together, burst mode and pipelining can dramatically reduce the lag caused by latency.

So why wouldn't you buy the fastest, widest memory you can get? The speed and width of the memory's bus should match the system's bus. You can use memory designed to work at 100 MHz in a 66-MHz system, but it will run at the 66-MHz speed of the bus so there is no advantage, and 32-bit memory won't fit on a 16-bit bus.

Even with a wide and fast bus, it still takes longer for data to get from the memory card to the CPU than it takes for the CPU to actually process the data. That's where caches come in.

Cache and Registers

Caches are designed to alleviate this bottleneck by making the data used most often by the CPU instantly available. This is accomplished by building a small amount of memory, known as **primary** or **level 1** cache, right into the CPU. Level 1 cache is very small, normally ranging between 2 kilobytes (KB) and 64 KB.

The **secondary** or **level 2** cache typically resides on a memory card located near the CPU. The level 2 cache has a direct connection to the CPU. A dedicated integrated circuit on the **motherboard**, the **L2 controller**, regulates the use of the level 2 cache by the CPU. Depending on the CPU, the size of the level 2 cache ranges from 256 KB to 2 megabytes (MB). In most systems, data needed by the CPU is accessed from the cache approximately 95 percent of the time, greatly reducing the overhead needed when the CPU has to wait for data from the main memory.

Some inexpensive systems dispense with the level 2 cache altogether. Many high performance CPUs now have the level 2 cache actually built into the CPU chip itself. Therefore, the size of the level 2 cache and whether it is **onboard** (on the CPU) is a major determining factor in the performance of a CPU. For more details on caching, see **How Caching Works**.

A particular type of **RAM**, **static random access memory** (SRAM), is used primarily for cache. SRAM uses multiple transistors, typically four to six, for each memory cell. It has an external **gate** array known as a **bistable multivibrator** that switches, or **flip-flops**, between two states. This means that it does not have to be continually refreshed like DRAM. Each cell will maintain its data as long as it has power. Without the need for constant refreshing, SRAM can operate extremely quickly. But the complexity of each cell make it prohibitively expensive for use as standard RAM.

The SRAM in the cache can be **asynchronous** or **synchronous**. Synchronous SRAM is designed to exactly match the speed of the CPU, while asynchronous is not. That little bit of timing makes a difference in performance. Matching the CPU's clock speed is a good thing, so always look for synchronized SRAM. (For more information on the various types of RAM, see **How RAM Works**.)

The final step in memory is the **registers**. These are memory cells built right into the CPU that contain specific data needed by the CPU, particularly the **arithmetic and logic unit** (ALU). An integral part of the CPU itself, they are controlled directly by the compiler that sends information for the CPU to process.

Volatility

Memory can be split into two main categories: volatile and nonvolatile. Volatile memory loses any data as soon as the system is turned off; it requires constant power to remain viable. Most types of RAM fall into this category.

Nonvolatile memory does not lose its data when the system or device is turned off. A number of types of memory fall into this category. The most familiar is ROM, but Flash memory storage devices such as CompactFlash or SmartMedia cards are also forms of nonvolatile memory.

Types of Memory

Volatile Memory

Volatile memory is computer memory that requires power to maintain the stored information. Most modern **semiconductor** volatile memory is either static RAM (**SRAM**) or dynamic RAM (**DRAM**). SRAM retains its contents as long as the power is connected and is easy for interfacing, but uses six transistors per bit. Dynamic RAM is more complicated for interfacing and control, needing regular refresh cycles to prevent losing its contents, but uses only one transistor and one capacitor per bit, allowing it to reach much higher densities and much cheaper per-bit costs.

SRAM is not worthwhile for desktop system memory, where DRAM dominates, but is used for their cache memories. SRAM is commonplace in small embedded systems, which might only need tens of kilobytes or less. Forthcoming volatile memory technologies that aim at replacing or competing with SRAM and DRAM include **Z-RAM** and **A-RAM**.

Non-Volatile Memory

Non-volatile memory is computer memory that can retain the stored information even when not powered. Examples of non-volatile memory include read-only memory (see **ROM**), **flash memory**, **3D XPoint**, most types of magnetic computer storage devices (e.g. **hard disk drives**, **floppy disks** and **magnetic tape**), **optical discs**, and early computer storage methods such as **paper tape** and **punched cards**.

Forthcoming non-volatile memory technologies include **FeRAM**, **CBRAM**, **PRAM**, **SONOS**, **RRAM**, **racetrack memory**, **NRAM** and **millipede memory**.

Memory Management

Proper management of memory is vital for a computer system to operate properly. Modern **operating systems** have complex systems to properly manage memory. Failure to do so can lead to bugs, slow performance, and at worst case, takeover by viruses and malicious software.

Nearly everything a computer programmer does requires him or her to consider how to manage memory. Even storing a number in memory requires the programmer to specify how the memory should store it.

Memory management bugs

Improper management of memory is a common cause of bugs, including the following types:

In an **arithmetic overflow**, a calculation results in a number larger than the allocated memory permits. For example, a signed 8-bit integer allows the numbers −128 to +127. If its value is 127 and it is instructed to add one, the computer cannot store the number 128 in that space. Such a case will result in undesired operation, such as changing the number's value to −128 instead of +128.

A **memory leak** occurs when a program requests memory from the operating system and never returns the memory when it's done with it. A program with this bug will gradually require more and more memory until the program fails as it runs out.

A **segmentation fault** results when a program tries to access memory that it does not have permission to access. Generally a program doing so will be terminated by the operating system.

A **buffer overflow** means that a program writes data to the end of its allocated space and then continues to write data to memory that has been allocated for other purposes. This may result in erratic program behavior, including memory access errors, incorrect results, a crash, or a breach of system security. They are thus the basis of many software vulnerabilities and can be maliciously exploited.

Early computer systems

In early computer systems, programs typically specified the location to write memory and what data to put there. This location was a physical location on the actual memory hardware. The slow processing of such computers did not allow for the complex memory management systems used today. Also, as most such systems were single-task, sophisticated systems were not required as much.

This approach has its pitfalls. If the location specified is incorrect, this will cause the computer to write the data to some other part of the program. The results of an error like this are unpredictable. In some cases, the incorrect data might overwrite memory used by the operating system. Computer crackers can take advantage of this to create viruses and malware.

Virtual memory

Virtual memory is a system where all physical memory is controlled by the operating system. When a program needs memory, it requests it from the operating system. The operating system then decides what physical location to place the memory in.

This offers several advantages. Computer programmers no longer need to worry about where the memory is physically stored or whether the user's computer will have enough memory. It also allows multiple types of memory to be used. For example, some memory can be stored in physical RAM chips while other memory is stored on a hard drive. This drastically increases the amount of memory available to programs. The operating system will place actively used memory in physical RAM, which is much faster than hard disks. When the amount of RAM is not sufficient to run all the current programs, it can result in a situation where the computer spends more time moving memory from RAM to disk and back than it does accomplishing tasks; this is known as **thrashing**.

Virtual memory systems usually include protected memory, but this is not always the case.

Memory protection

Protected memory is a system where each program is given an area of memory to use and is not permitted to go outside that range. Use of protected memory greatly enhances both the reliability and security of a computer system.

Without protected memory, it is possible that a bug in one program will alter the memory used by another program. This will cause that other program to run off of corrupted memory with unpredictable results. If the operating system's memory is corrupted, the entire computer system may crash and need to be rebooted. At times programs intentionally alter the memory used by other programs. This is done by viruses and malware to take over computers.

Protected memory assigns programs their own areas of memory. If the operating system detects that a program has tried to alter memory that does not belong to it, the program is terminated. This way, only the offending program crashes, and other programs are not affected by the error.

Protected memory systems almost always include virtual memory as well.

Memory Hierarchy

The term memory hierarchy is used in **computer architecture** when discussing performance issues in computer architectural design, algorithm predictions, and the lower level **programming** constructs such as involving **locality of reference**. A "memory hierarchy" in **computer storage** distinguishes each level in the "hierarchy" by response time. Since response time, complexity, and capacity are related, the levels may also be distinguished by the controlling technology.

The many trade-offs in designing for high performance will include the structure of the memory hierarchy, i.e. the size and technology of each component. So the various components can be viewed as forming a hierarchy of memories (m1,m2,...,mn) in which each member mi is in a sense subordinate to the next highest member mi+1 of the hierarchy. To limit waiting by higher levels, a lower level will respond by filling a buffer and then signaling to activate the transfer.

There are four major storage levels.

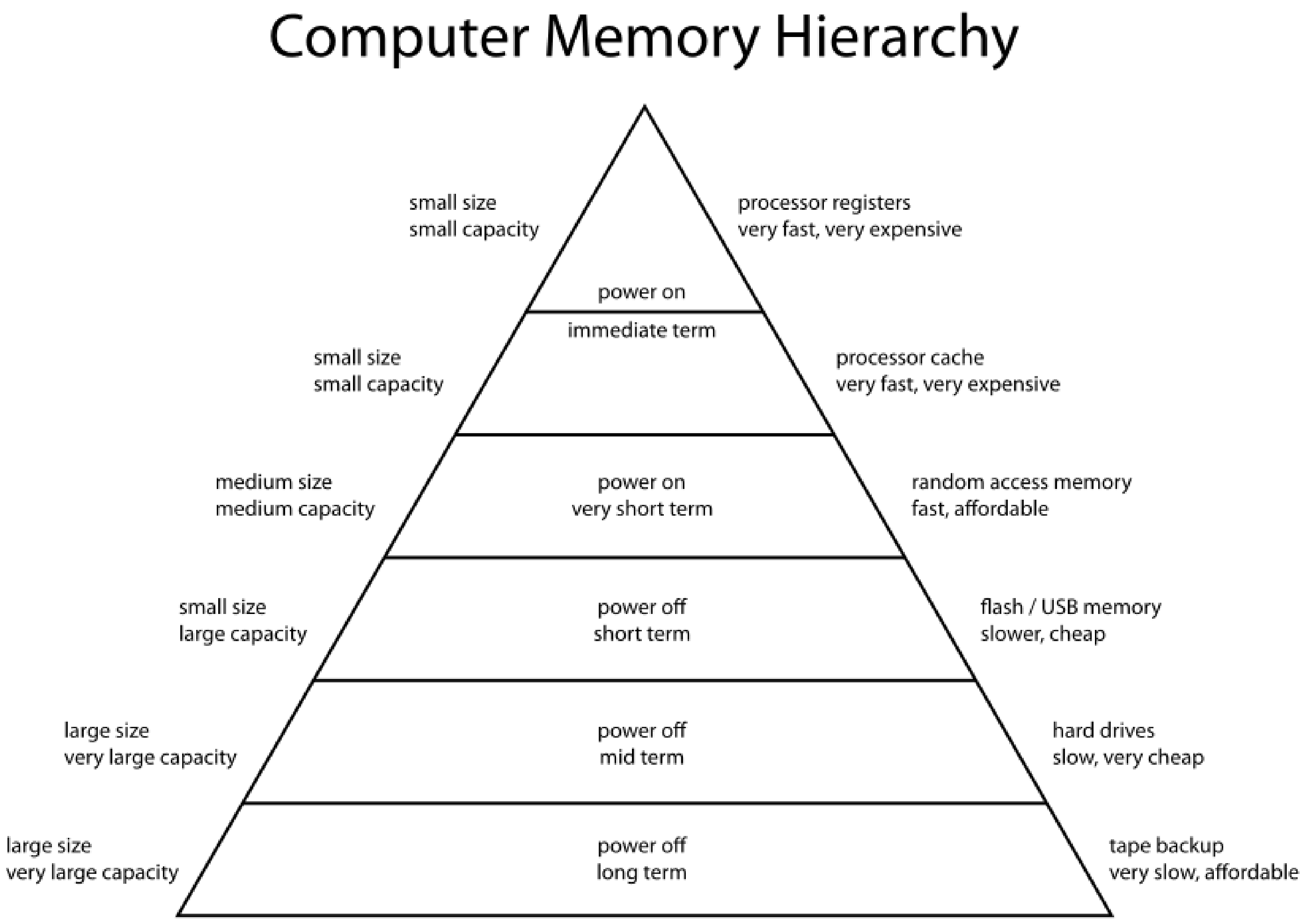
Internal – **Processor registers** and **cache**.

Main – the system **RAM** and controller cards.

On-line mass storage – Secondary storage.

Off-line bulk storage – Tertiary and Off-line storage.

This is a general memory hierarchy structuring. Many other structures are useful. For example, a paging algorithm may be considered as a level for **virtual memory** when designing a **computer architecture**, and one can include a level of **nearline storage** between online and offline storage.



- Adding complexity slows down the memory hierarchy.
- CMOx memory technology stretches the Flash space in the memory hierarchy
- One of the main ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.
- Latency and bandwidth are two metrics associated with caches and memory. Neither of them is uniform, but is specific to a particular component of the memory hierarchy.
- Predicting where in the memory hierarchy the data resides is difficult.
- The location in the memory hierarchy dictates the time required for the prefetch to occur.

The number of levels in the memory hierarchy and the performance at each level has increased over time. For example, the memory hierarchy of an Intel Haswell Mobile processor circa 2013 is:

- **Processor registers** – the fastest possible access (usually 1CPU cycle). Few thousand bytes in size
- **Cache**
    - Level 0 (L0) **Micro operations** cache – 6 **KiB** in size
    - Level 1 (L1) **Instruction** cache – 128 KiB in size
    - Level 1 (L1) Data cache – 128 KiB in size. Best access speed is around 700 **GiB**/second
    - Level 2 (L2) Instruction and data (shared) – 1 **MiB** in size. Best access speed is around 200 GiB/second
    - Level 3 (L3) Shared cache – 6 MiB in size. Best access speed is around 100 GB/second
    - Level 4 (L4) Shared cache – 128 MiB in size. Best access speed is around 40 GB/second
- **Main memory** (Primary storage) – Gigabytes in size. Best access speed is around 10 GB/second. In the case of a NUMA machine, access times may not be uniform
- **Disk storage** (Secondary storage) – Terabytes in size. As of 2013, best access speed is from a solid state drive is about 600 MB/second
- **Nearline storage** (Tertiary storage) – Up to exabytes in size. As of 2013, best access speed is about 160 MB/second
- **Offline storage**

The lower levels of the hierarchy – from disks downwards – are also known as tiered storage. The formal distinction between online, nearline, and offline storage is:

- Online storage is immediately available for I/O.
- Nearline storage is not immediately available, but can be made online quickly without human intervention.
- Offline storage is not immediately available, and requires some human intervention to bring online.

For example, always-on spinning disks are online, while spinning disks that spin-down, such as massive array of idle disk (MAID), are nearline. Removable media such as tape cartridges that can be automatically loaded, as in a tape library, are nearline, while cartridges that must be manually loaded are offline.

Most modern **CPUs** are so fast that for most program workloads, the **bottleneck** is the **locality of reference** of memory accesses and the efficiency of the **caching** and memory transfer between different levels of the hierarchy. As a result, the CPU spends much of its time idling, waiting for memory I/O to complete. This is sometimes called the space cost, as a larger memory object is more likely to overflow a small/fast level and require use of a larger/slower level. The resulting load on memory use is known as pressure (respectively register pressure, cache pressure, and (main) memory pressure). Terms for data being missing from a higher level and needing to be fetched from a lower level are, respectively: register spilling (due to register pressure: register to cache), cache miss (cache to main memory), and (hard) **page fault** (main memory to disk).

Modern **programming languages** mainly assume two levels of memory, main memory and disk storage, though in **assembly language** and **inline assemblers** in languages such as **C**, registers can be directly accessed. Taking optimal advantage of the memory hierarchy requires the cooperation of programmers, hardware, and compilers (as well as underlying support from the operating system):

- **Programmers** are responsible for moving data between disk and memory through file I/O.
- **Hardware** is responsible for moving data between memory and caches.
- **Optimizing compilers** are responsible for generating code that, when executed, will cause the hardware to use caches and registers efficiently.

Many programmers assume one level of memory. This works fine until the application hits a performance wall. Then the memory hierarchy will be assessed during **code refactoring**.

The New Generation of Memory

Typically, memory chips store information in *transistors*, tiny three-pronged devices that can house electrons. Some memory company, like Intel and Micron have done away the transistor, storing information in what is really just a lattice of wires. Each piece of information sits where two wires cross.

The Third Dimension

The “3D” bit indicates that those cross-point wire lattices can be stacked on top of each other. Basically, these chips store data by changing the electrical resistance at those many cross points. This is done with a tiny contraption that Fazio calls a “selector.” This is similar to **a diode**, which can switch from low resistance to high.

That’s quite different from memory technologies like DRAM and Flash, which use transistors. A transistor is essentially a switch made from three basic components: a source, gate, and drain. When a certain voltage is applied to the gate, current flows from the source to the drain, and the transistor is “on.” Apply another voltage, the current stops, and the transistor is “off.” 3D XPoint doesn’t move current like this. It changes the property of the material. It doesn’t try to store an electron.

This simple architecture is what allows the chips to store so much more data than DRAM (per area). They still aren’t as fast as DRAM, but unlike DRAM, they’re “non-volatile,” meaning they can retain data even when a machine is powered down. Flash is non-volatile too, but 3D XPoint is significantly faster than flash. That extra speed comes from the unique combination of materials that chip manufacturers use to build the lattice and the selector.

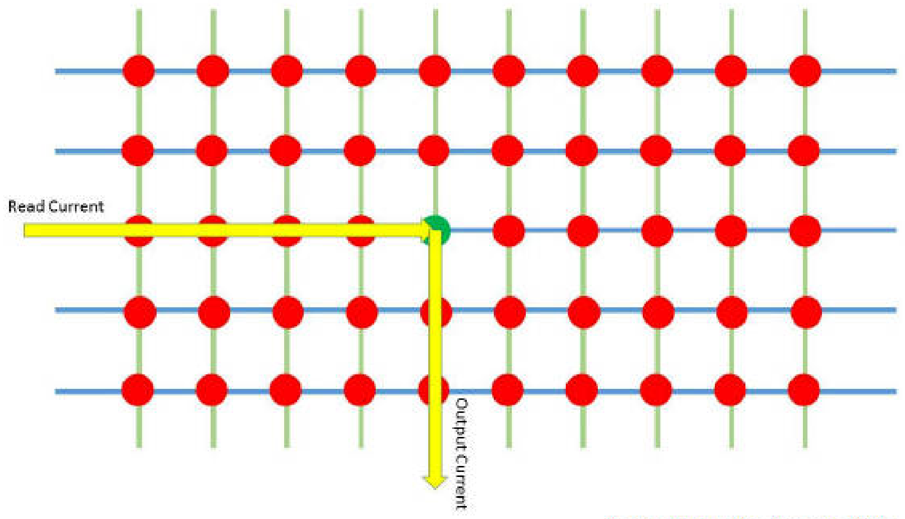
A Real Need For Speed

3D XPoint will be great for gaming machines. Today, game designers are restricted in what they can do by the capabilities of the system and that may well be the case. But the real need is inside the massive computer data centers that power the most popular web services Google, Facebook, Amazon.

In the data center, these companies must spread information across thousands of machines. In an effort to speed the storage and retrieval of this data, **they’re replacing traditional hard drives with faster flash memory**, and in some cases, they’re **running databases that keep data in DRAM**, sidestepping both hard drives and flash. As described 3D XPoint can provide still more performance inside these warehoused-sized computing centers—something the Facebooks and the Googles are always hungry for.

At the same time the new tech can help more traditional businesses. The typical enterprise doesn’t store data across thousands of machines the way Facebook or Google does. But smaller outfits are running “in-memory databases”—i.e. in-DRAM databases—on individual machines or small numbers of machines. 3D XPoint can potentially help here as well. It can provide more memory per machine.

Figure 3. Reading a bit in a crosspoint array



Source: Objective Analysis, 2015

References:

[1] A.M. Turing and R.A. Brooker (1952). Programmer's Handbook for Manchester Electronic Computer Mark II. University of Manchester.

[2] Stanek, William R. (2009). Windows Server 2008 Inside Out. O'Reilly Media, Inc. p. 1520. ISBN 9780735638068. Retrieved 2012-08-20. Windows Server Enterprise supports clustering with up to eight-node clusters and very large memory (VLM) configurations of up to 32 GB on 32-bit systems and 2 TB on 64-bit systems.

[3] Miller, Stephen W. (1977), Memory and Storage Technology, Montvale.: AFIPS Press

[4] Memory and Storage Technology, Alexandria, Virginia.: Time Life Books, 1988

[5] Toy, Wing; Zee, Benjamin (1986). Computer Hardware/Software Architecture. Prentice Hall. p. 30. *ISBN 0-13-163502-6*.

[6] "Memory Hierarchy". Unity Semiconductor Corporation*. Retrieved 16 September 2009*.

[7] Pádraig Brady. *"Multi-Core". Retrieved 16 September 2009*.

[8] van der Pas, Ruud (2002). Santa Clara, California: Sun Microsystems: 26. 817-0742-10 *http://www.sun.com/*.

[9] Crothers, Brooke. "Dissecting Intel's top graphics in Apple's 15-inch MacBook Pro - CNET". News.cnet.com*. Retrieved 2014-07-31*.

[10] "Intel's Haswell Architecture Analyzed: Building a New PC and a New Intel". AnandTech. Retrieved 2014-07-31.

[11] b c d e "Si Software Zone". Sisoftware.co.uk. Retrieved 2014-07-31.

[12] "Charts, benchmarks SSD Charts 2013, AS-SSD Sequential Read". Tomshardware.com. Retrieved 2014-07-31.

[13] "Ultrium - LTO Technology - Ultrium Generations LTO". Lto.org. Retrieved 2014-07-31.

[14] Pearson, Tony (2010). "Correct use of the term Nearline." IBM Developer works, Inside System Storage*. Retrieved 2015-08-16*.

[15] Clarke, Peter (28 July 2015)*,* "Intel, Micron Launch "Bulk-Switching" ReRAM"*, www.eetimes.com*

[16] Neale, Ron (14 Aug 2015), "Imagining What's Inside 3D XPoint"*, www.eetimes.com*

* Mr. Nikola Zlatanov spent over 20 years working in the Capital Semiconductor Equipment Industry. His work at Gasonics, Novellus, Lam and KLA-Tencor involved progressing electrical engineering and management roles in disruptive technologies. Nikola received his Undergraduate degree in Electrical Engineering and Computer Systems from Technical University, Sofia, Bulgaria and completed a Graduate Program in Engineering Management at Santa Clara University. He is currently consulting for Fortune 500 companies as well as Startup ventures in Silicon Valley, California.