Memory

Random-Access Memory (RAM)

Key features

- RAM is packaged as a chip.
- Basic storage unit is a cell (one bit per cell).
- Multiple RAM chips form a memory.

Static RAM (SRAM)

- Each cell stores bit with a six-transistor circuit.
- Retains value indefinitely, as long as it is kept powered.
- Relatively insensitive to disturbances such as electrical noise.
- Faster and more expensive than DRAM.

Dynamic RAM (DRAM)

- Each cell stores bit with a capacitor and transistor.
- Value must be refreshed every 10-100 ms.
- Sensitive to disturbances.
- Slower and cheaper than SRAM.

SRAM vs DRAM Summary

	Tran. per bit	Access time	Persist?	Sensitive?	Cost	Applications
SRAM	6	1X	Yes	No	100x	cache memories
DRAM	1	10X	No	Yes	1X	Main memories, frame buffers

Random access memory

Can address a randomly chosen location in a fixed amoutn of time.

SRAM

Fast, expensive, and stable.

Stable means that the memory remains intact as long as the power is on.

DRAM Slow (10 times slower than DRAMS), cheap, and not stable

They must be refreshed every few milliseconds.

Memory controller

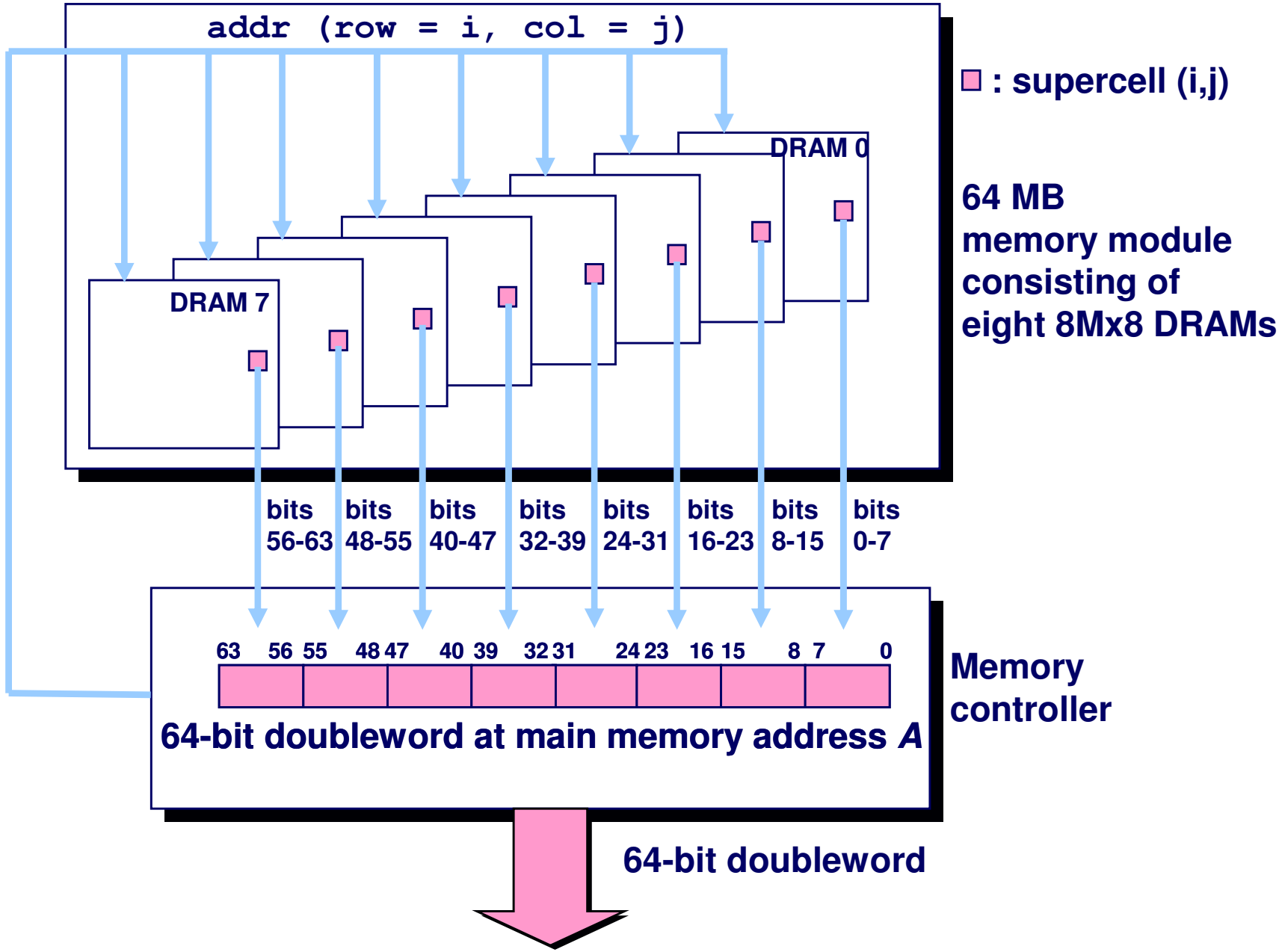
The memory controller is a piece of hardware that communicates with the CPU via the memory bus.

Memory Cycle

The memory cycle is equivalent to the CPU cycle.

It requires more than one memory cycle to read or write memory.

Memory Modules



Enhanced DRAMs

DRAM Cores with better interface logic and faster I/O :

- Synchronous DRAM (SDRAM)

Uses a conventional clock signal instead of asynchronous control

- Double data-rate synchronous DRAM (DDR SDRAM)

Double edge clocking sends two bits per cycle per pin

- RamBus™ DRAM (RDRAM)

Uses faster signaling over fewer wires (source directed clocking) with a Transaction oriented interface protocol

Obsolete Technologies :

- Fast page mode DRAM (FPM DRAM)

Allowed re-use of row-addresses

- Extended data out DRAM (EDO DRAM)

Enhanced FPM DRAM with more closely spaced CAS signals.

- Video RAM (VRAM)

Dual ported FPM DRAM with a second, concurrent, serial interface

- Extra functionality DRAMS (CDRAM, GDRAM)

Nonvolatile Memories

Nonvolatile memories retain value even if powered off

- Read-only memory (ROM): programmed during production
- Magnetic RAM (MRAM): stores bit magnetically (in development)
- Ferro-electric RAM (FERAM): uses a ferro-electric dielectric
- Programmable ROM (PROM): can be programmed once
- Eraseable PROM (EPROM): can be bulk erased (UV, X-Ray)
- Electrically eraseable PROM (EEPROM): electronic erase capability
- Flash memory: EEPROMs with partial (sector) erase capability

Uses for Nonvolatile Memories

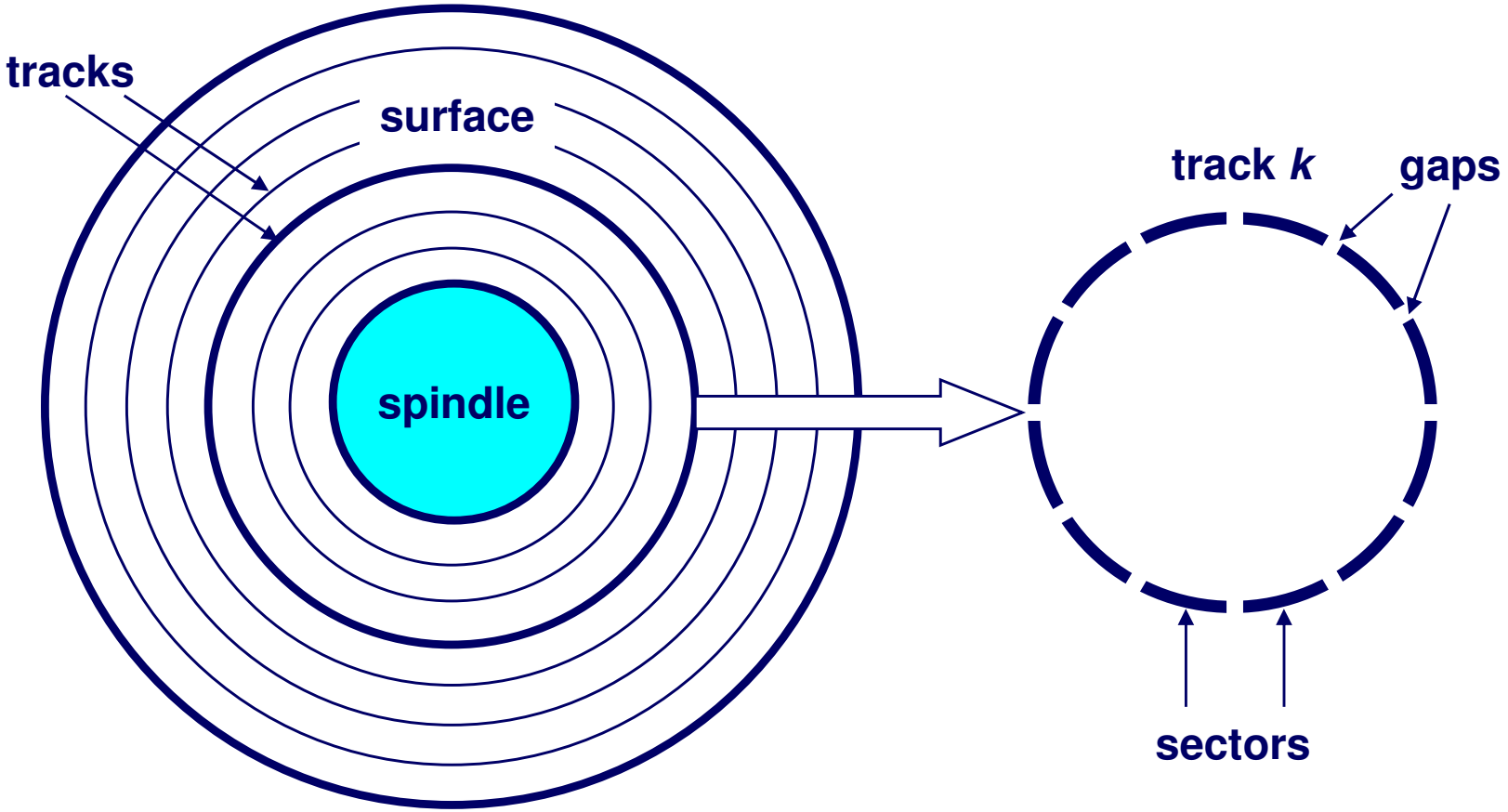
- Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,…)
- Solid state disks (flash cards, memory sticks, etc.)
- Smart cards, embedded systems, appliances
- Disk caches

Disk Geometry

Disks consist of platters, each with two surfaces.

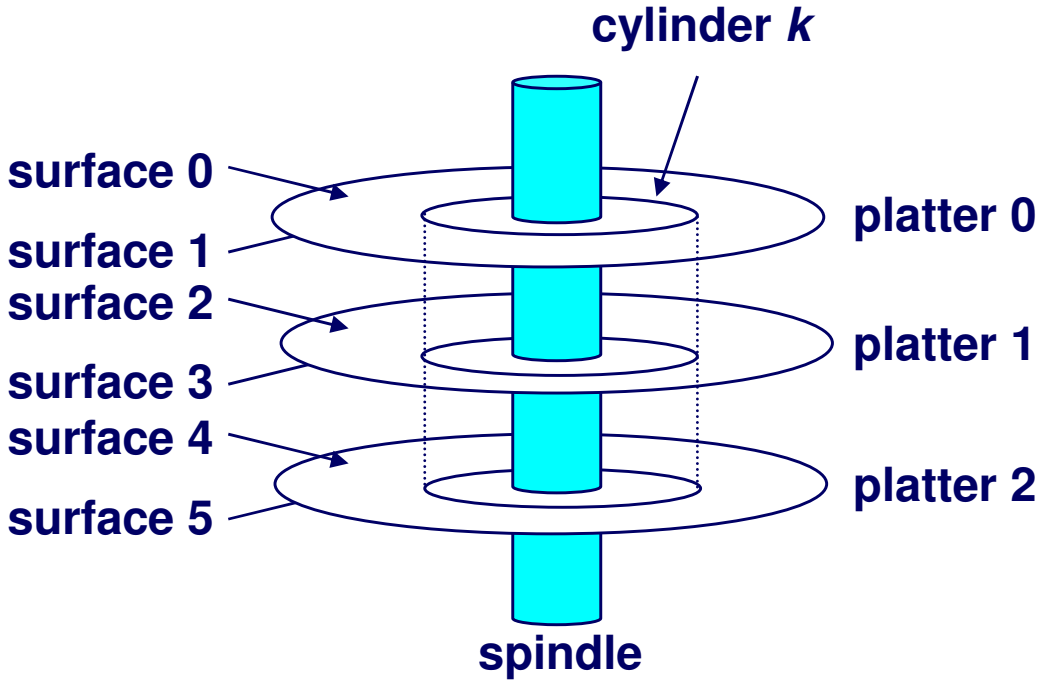
Each surface consists of concentric rings called tracks.

Each track consists of sectors separated by gaps.



Disk Geometry (Muliple-Platter View)

Aligned tracks form a cylinder.



Disk Capacity

Capacity: maximum number of bits that can be stored.

- Vendors express capacity in units of gigabytes (GB), where 1 GB = 10^9.

Capacity is determined by these technology factors:

- Recording density (bits/in): number of bits that can be squeezed into a 1 inch segment of a track.
- Track density (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment.
- Areal density (bits/in2): product of recording and track density.

Modern disks partition tracks into disjoint subsets called recording zones

- Each track in a zone has the same number of sectors, determined by the circumference of innermost track.
- Each zone has a different number of sectors/track

Computing Disk Capacity

Capacity = (# bytes/sector) x (avg. # sectors/track) x (# tracks/surface) x (# surfaces/platter) x (# platters/disk)

Example:

- 512 bytes/sector
- 300 sectors/track (on average)
- 20,000 tracks/surface
- 2 surfaces/platter
- 5 platters/disk

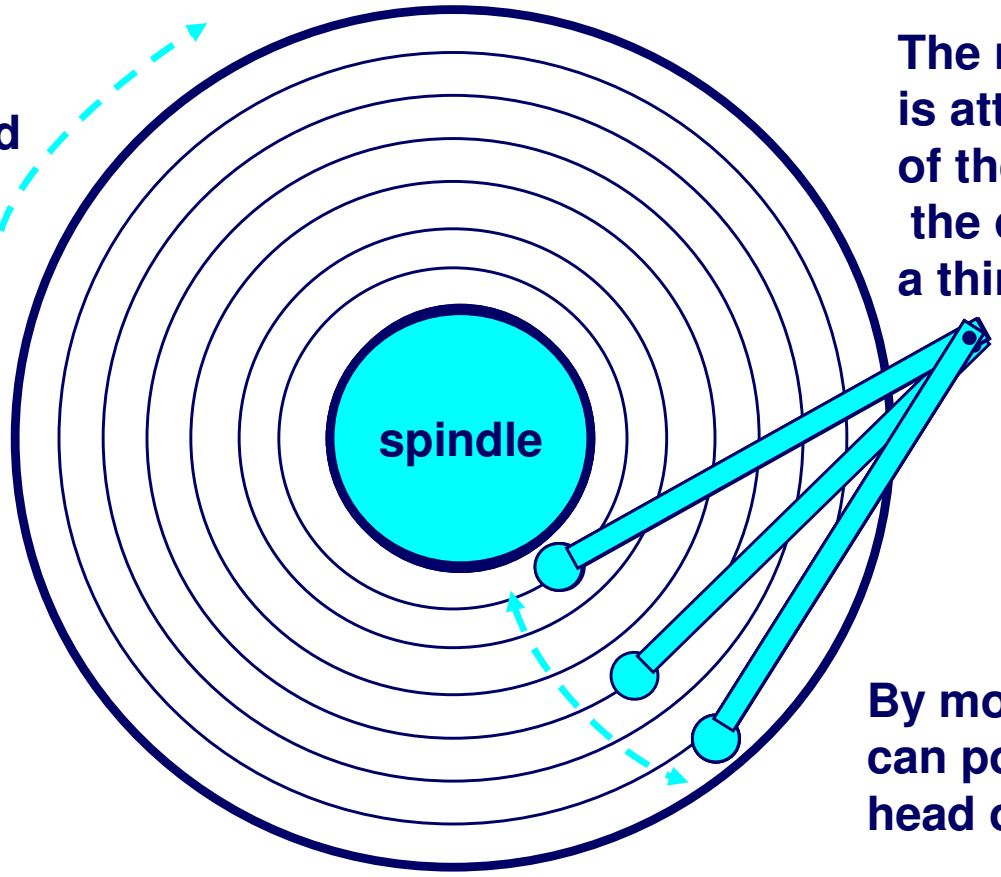
Capacity = 512 x 300 x 20000 x 2 x 5

= 30,720,000,000

= 30.72 GB

Disk Operation (Single-Platter View)

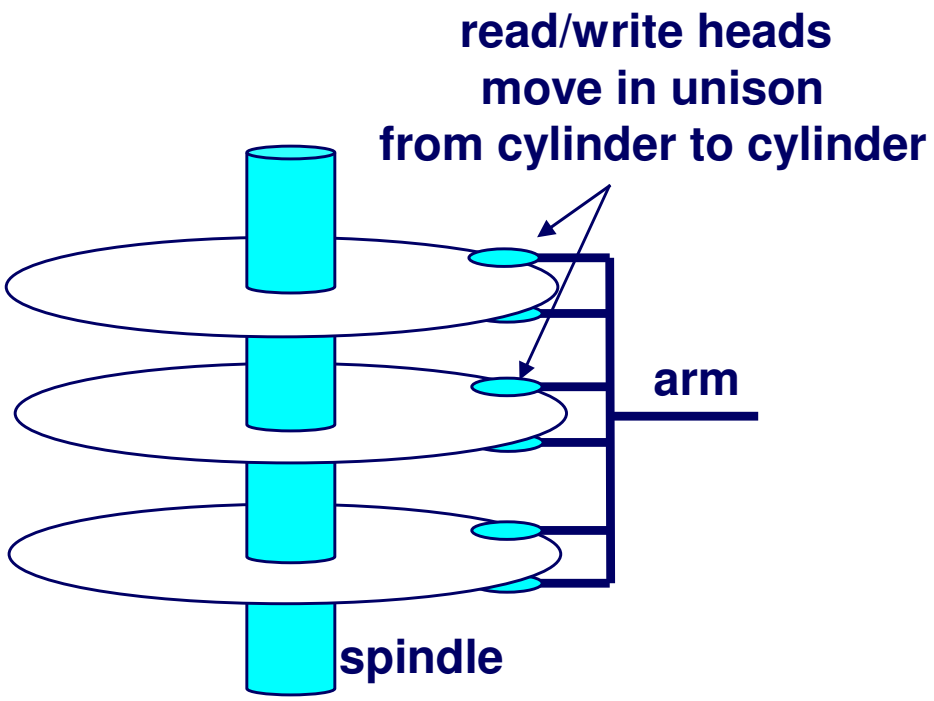
The disk surface spins at a fixed rotational rate



The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

Disk Operation (Multi-Platter View)



Disk Access Time

Average time to access some target sector approximated by :

- Taccess = Tavg seek + Tavg rotation + Tavg transfer

Seek time (Tavg seek)

- Time to position heads over cylinder containing target sector.
- Typical Tavg seek = 9 ms

Rotational latency (Tavg rotation)

- Time waiting for first bit of target sector to pass under r/w head.
- Tavg rotation = 1/2 x 1/RPMs x 60 sec/1 min

Transfer time (Tavg transfer)

- Time to read the bits in the target sector.
- Tavg transfer = 1/RPM x 1/(avg # sectors/track) x 60 secs/1 min.

Disk Access Time Example

Given:

- Rotational rate = 7,200 RPM
- Average seek time = 9 ms.
- Avg # sectors/track = 400.

Derived:

- Tavg rotation = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms.
- Tavg transfer = 60/7200 RPM x 1/400 secs/track x 1000 ms/sec = 0.02 ms
- Taccess = 9 ms + 4 ms + 0.02 ms

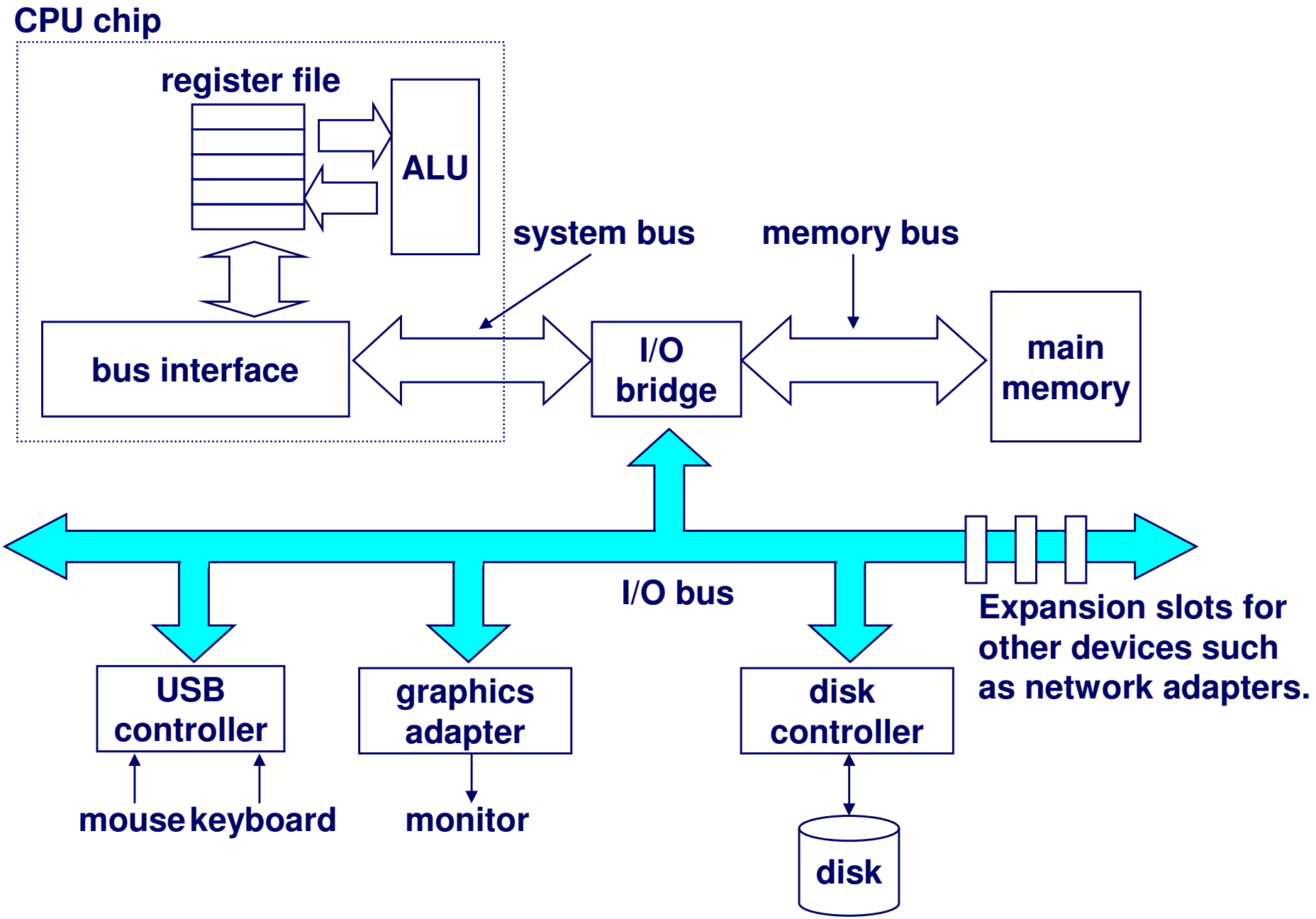
Important points:

- Access time dominated by seek time and rotational latency.
- First bit in a sector is the most expensive, the rest are free.
- SRAM access time is about 4 ns/doubleword, DRAM about 60 ns
  - Disk is about 40,000 times slower than SRAM,
  - 2,500 times slower then DRAM.

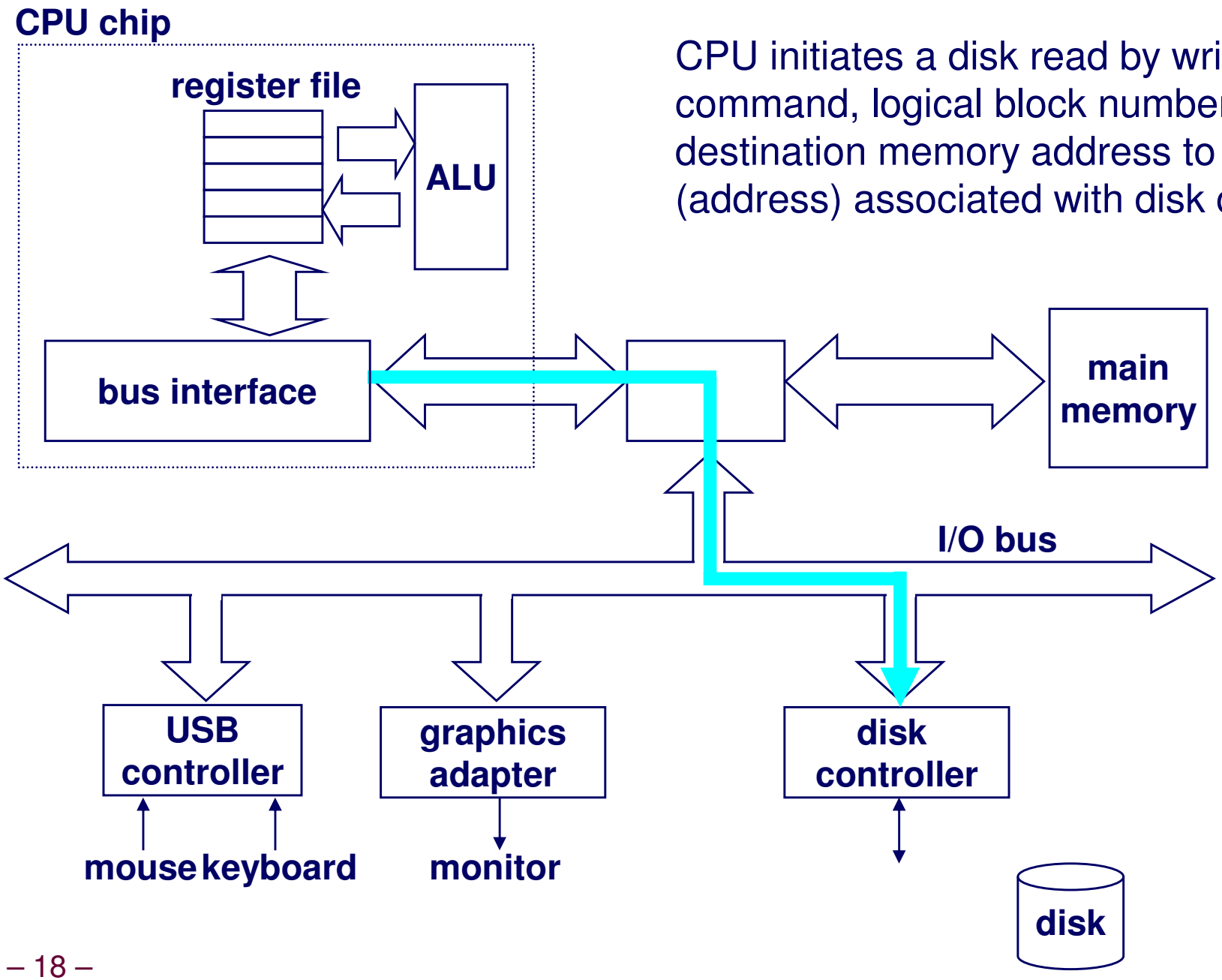
Logical Disk Blocks

- Modern disks present a simpler abstract view of the complex sector geometry:
  - The set of available sectors is modeled as a sequence of b-sized logical blocks (0, 1, 2, ...)
- Mapping between logical blocks and actual (physical) sectors
  - Maintained by hardware/firmware device called disk controller.
  - Converts requests for logical blocks into (surface,track,sector) triples.
- Allows controller to set aside spare cylinders for each zone.
  - Accounts for the difference in "formatted capacity" and "maximum capacity".

I/O Bus

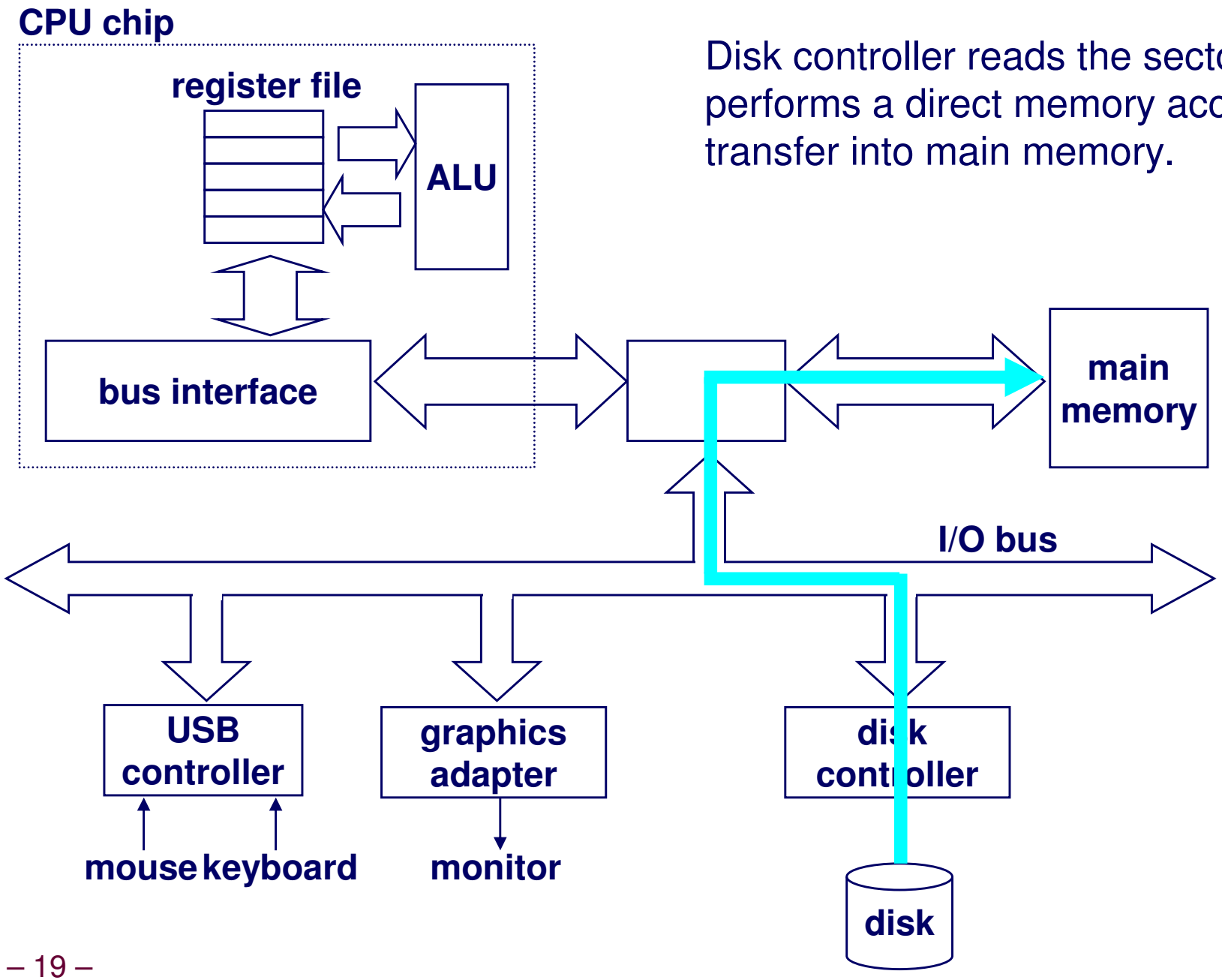

CPU chip
register file
ALU
system bus
memory bus
bus interface
I/O bridge
main memory
I/O bus
Expansion slots for other devices such as network adapters.
USB controller
graphics adapter
disk controller
mouse
keyboard
monitor
disk

Reading a Disk Sector (1)



CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

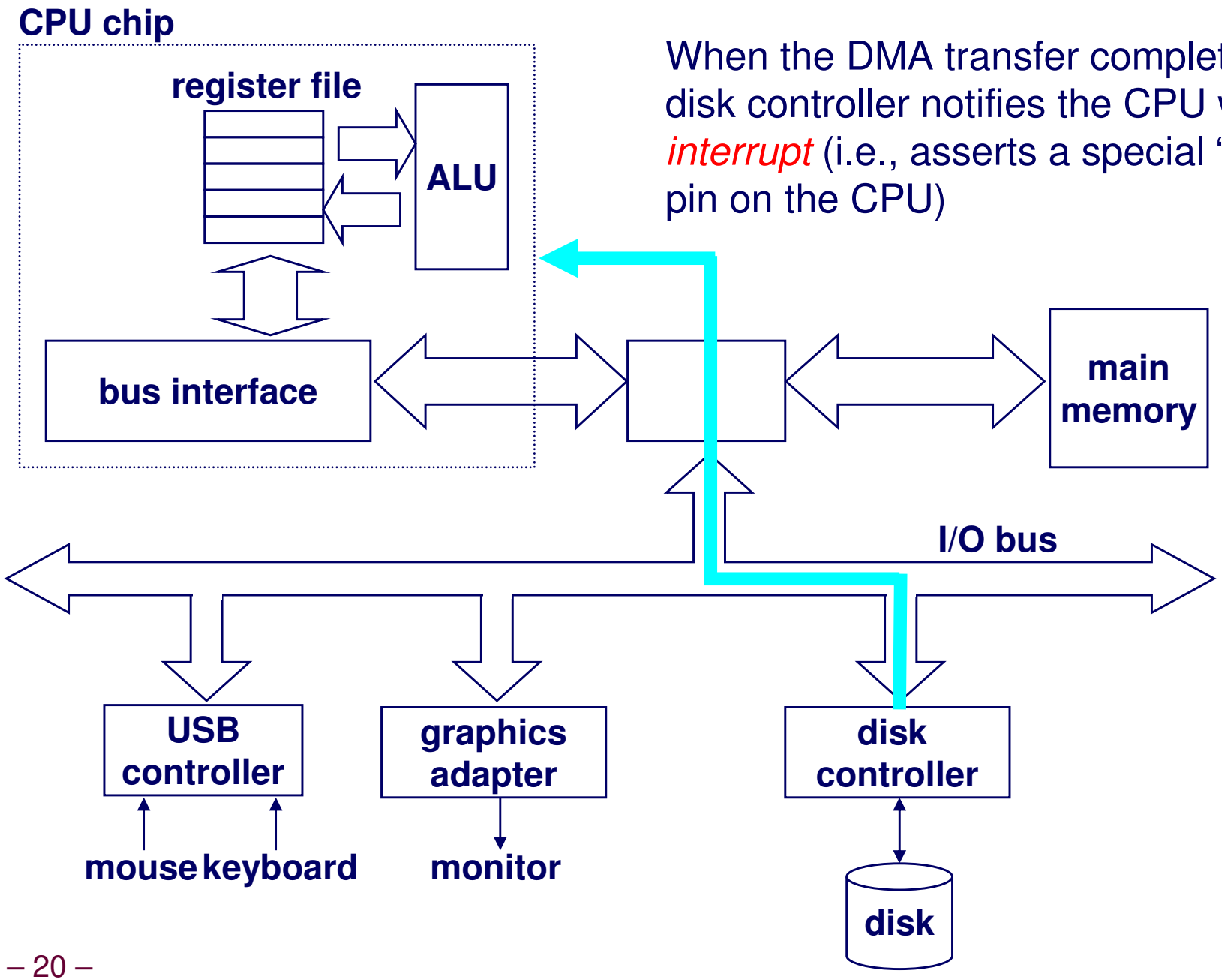
Reading a Disk Sector (2)



Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

Reading a Disk Sector (3)

When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special "interrupt" pin on the CPU)



Storage Trends

SRAM

metric	1980	1985	1990	1995	2000	2005	*2005:1980*
$/MB	19,200	2,900	320	256	100	75	*256*
access (ns)	300	150	35	15	12	10	*30*

DRAM

metric	1980	1985	1990	1995	2000	2005	*2005:1980*
$/MB	8,000	880	100	30	1	0.20	*40,000*
access (ns)	375	200	100	70	60	50	*8*
typical size(MB)	0.064	0.256	4	16	64	1,000	*15,000*

Disk

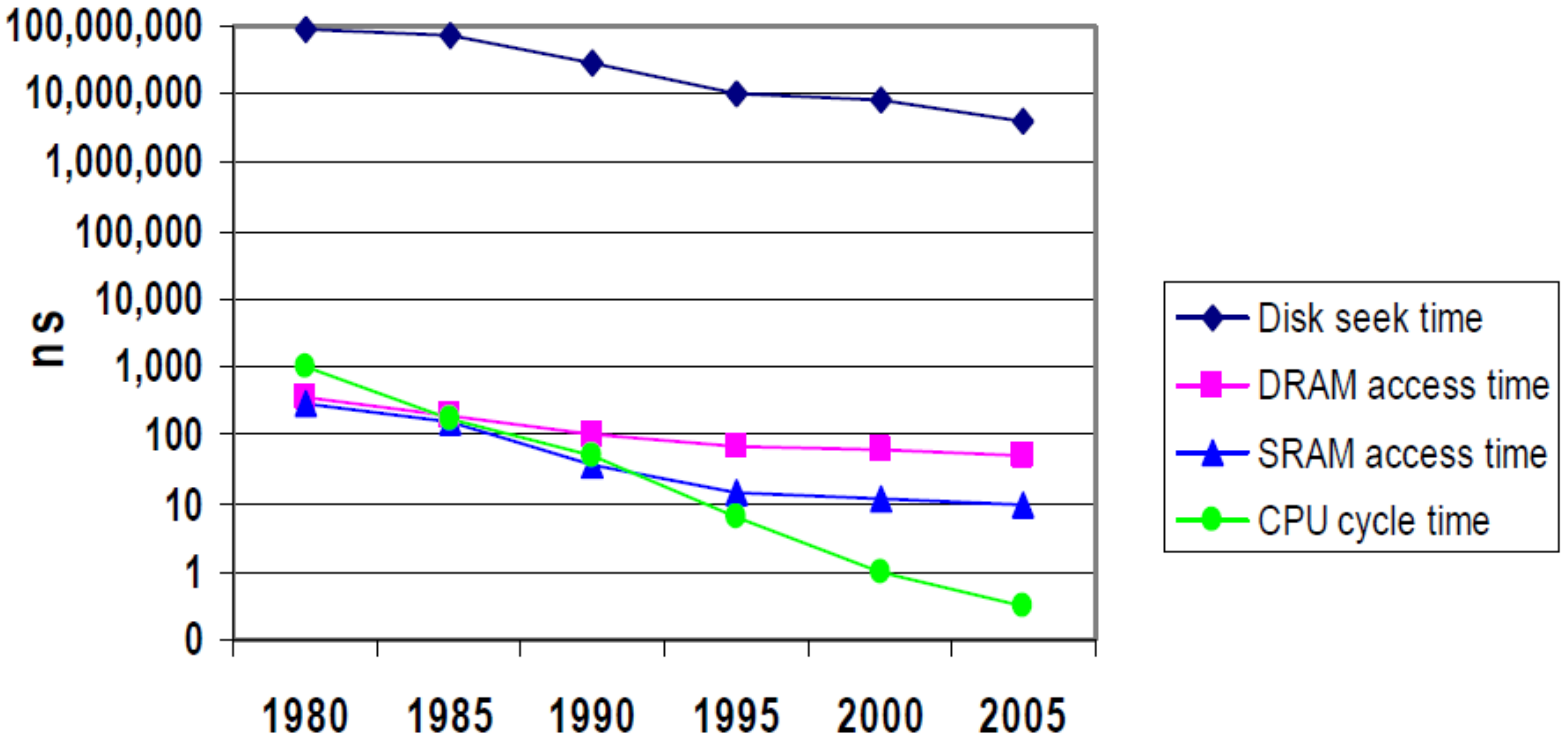
metric	1980	1985	1990	1995	2000	2005	*2005:1980*
$/MB	500	100	8	0.30	0.05	0.001	*10,000*
access (ms)	87	75	28	10	8	*4*	*22*
typical size(MB)	1	10	160	1,000	9,000	400,000	*400,000*

CPU Clock Rates

	1980	1985	1990	1995	2000	2005	*2005:1980*
processor	8080	286	386	Pentium	P-III	P-4	
clock rate(MHz)	1	6	20	150	750	3,000	3,000
cycle time(ns)	1,000	166	50	6	1.3	0.3	3,333

The CPU-Memory Gap

The increasing gap between DRAM, disk, and CPU speeds.



INTERLEAVED MEMORY

A cure for the cycle time is to pipeline memory references.

Memory is divided into banks as illustrated below.

Bank 0	Bank 1	Bank 2	Bank 3
00000000	00000001	00000010	000000011
00000100	00000101	00000110	000000111
00001000	00001001	00001010	000010111
00001100	00001101	00001110	000011111
00010000	00010001	00010010	000100111
...	...	...	...

Note that consecutive addresses move across the banks.

If we make reference to addresses 0, 1, 2, 3, 4, 5, ..., we will honor them from banks 0, 1, 2, 3, 0, 1, ...

Locality

Principle of Locality:

- **Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.**
- **Temporal locality: Recently referenced items are likely to be referenced in the near future.**
- **Spatial locality: Items with nearby addresses tend to be referenced close together in time.**

Locality Example:

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- **Data**
  - Reference array elements in succession (stride-1 reference pattern): **Spatial locality**
  - Reference `sum` each iteration: **Temporal locality**
- **Instructions**
  - Reference instructions in sequence: **Spatial locality**
  - Cycle through loop repeatedly: **Temporal locality**

Locality Example

Question: Can you permute the loops so that the function scans the 3-d array `a[]` with a stride-1 reference pattern (and thus has good spatial locality)?

```
int sumarray3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[k][i][j];
    return sum
}
```

Memory Hierarchies

Some fundamental and enduring properties of hardware and software:

- **Fast storage technologies cost more per byte and have less capacity.**
- **The gap between CPU and main memory speed is widening.**
- **Well-written programs tend to exhibit good locality.**

These fundamental properties complement each other beautifully.

They suggest an approach for organizing memory and storage systems known as a memory hierarchy.

MEMORY

Memory is organized in a hierarchy

Registers

These are pieces of memory inside the CPU — very fast.

Typically there are fewer than 100 registers.

Cache

This a buffer of memory between the CPU and main memory.

It is slower than registers but faster than main memory.

It now usually comes in two levels, one on the CPU chip and one on the motherboard

Main memory

This is what we usually mean by memory.

Slower than cache.

Used to come in KB's, then in MB's, and now frequently in GB's

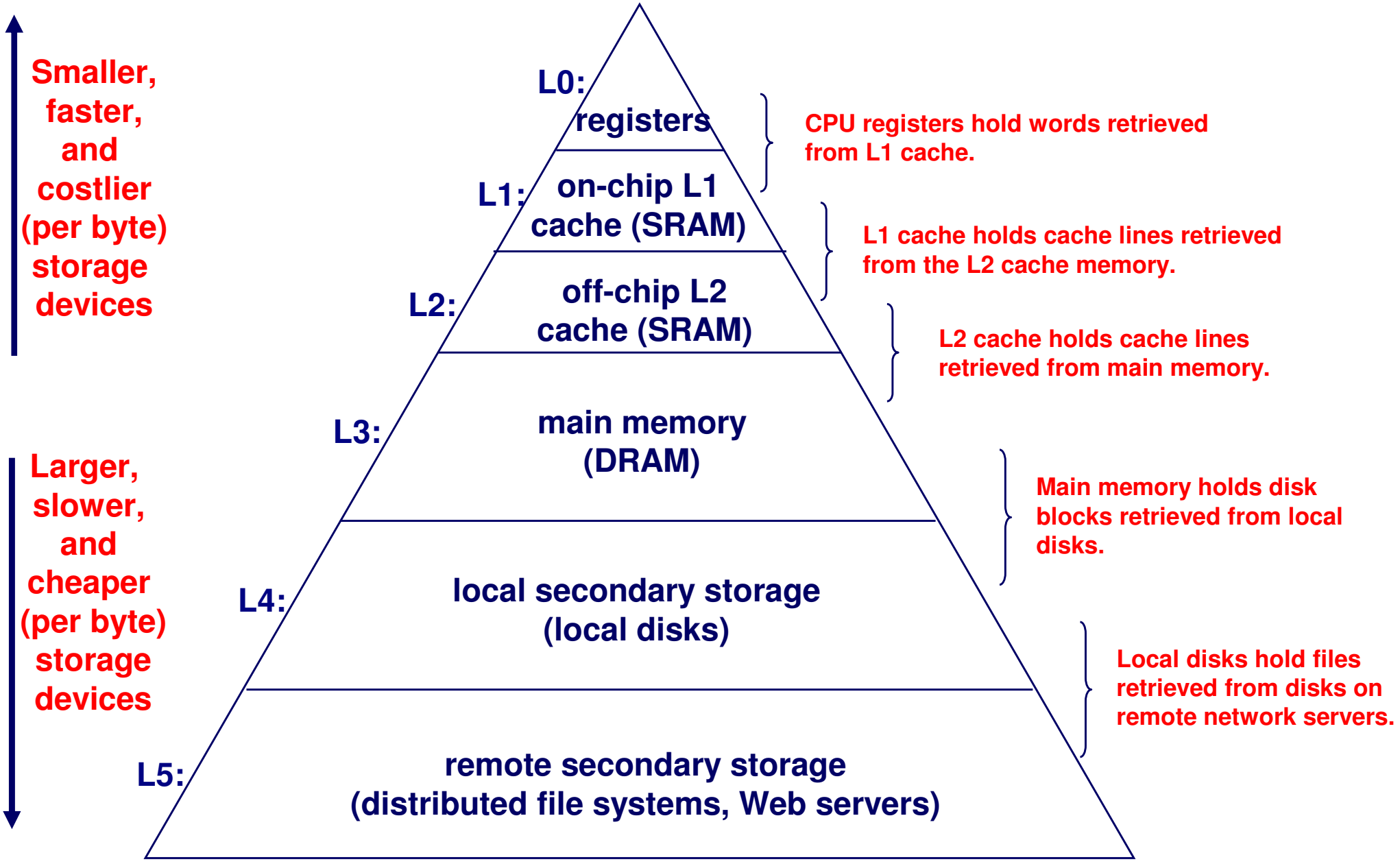
Virtual memory

Resides on disk.

Much bigger and much slower than main memory.

Main memory is effectively a cache for virtual memory.

An Example Memory Hierarchy



Caches

Cache: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

Fundamental idea of a memory hierarchy:

- For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.

Why do memory hierarchies work?

- Programs tend to access the data at level k more often than they access the data at level k+1.
- Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.
- Net effect: A large pool of memory that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.

THE MEMORY PIPELINE

Bank 0	Bank 1	Bank 2	Bank 3
00000000	00000001	00000010	000000011
00000100	00000101	00000110	000000111
00001000	00001001	00001010	000010111
00001100	00001101	00001110	000011111
00010000	00010001	00010010	000100111
...	...	...	...

Assume that:

1. We are reading consecutive locations in memory.
2. A memory transaction requires four memory cycles to complete.

Here is how the reading proceeds in time.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1 5
Bank 0	r	-	-	d	r	-	-	d	r	-	-	d	r	-	-
Bank 1	-	r	-	-	d	r	-	-	d	r	-	-	d	r	-
Bank 2	-	-	r	-	-	d	r	-	-	d	r	-	-	d	r
Bank 3	-	-	-	r	-	-	d	r	-	-	d	r	-	-	d

STRIDES

Bank 0	Bank 1	Bank 2	Bank 3
00000000	00000001	00000010	000000011
00000100	00000101	00000110	000000111
00001000	00001001	00001010	000010111
00001100	00001101	00001110	000011111
00010000	00010001	00010010	000100111
...	...	...	...

Suppose one references memory addresses in the order

$$a \quad a+s \quad a+2s \quad a+3s \quad \ldots$$

Then s is called the stride of the set of references.

Suppose in our four bank example, we reference with a stride of 2 starting with $a = 0$. Then we will reference banks 0, 2, 0, 2, 0, 2, ...

The result is that we get a read every two cycles — half the rate with a stride of 4.

If however we take a stride of 7, we get the banks in order 0, 3, 2, 1, 0, 3, ..., and we get a read every cycle.

If the banks and the stride are relatively prime all works well.

THE IDEA OF A CACHE

Cache is a small fast memory that buffers references to main memory.

The cache is divided into cache blocks or lines.

Memory is also divided into memory blocks of the same size.

When the CPU makes a memory reference to a particular memory block, it first looks to see if it is in cache.

> If it is in cache (a cache hit), then it honors the reference from cache.
>
> Otherwise (a cache miss)it brings the memory block into cache from main memory and honors the reference from cache.

The cache improves performance because once a block is in cache several references are made to it.

> Consider, for example, memory reference with stride one.
>
> This property is called locality of reference.

The word 'cache' has many uses.

SEVEN QUESTIONS

1. Where in the cache are memory blocks constrained to lie?
2. How does one determine if a memory block is in the cache?
3. When a new block is brought in, what block does it displace?
4. What happens on a read hit?
5. What happens on a write hit?
6. What happens on a read miss?
7. What happens on a write miss?

CACHE ORGANIZATION

1. Where in the cache are memory blocks constrained to lie?
2. How does one determine if a memory block is in the cache?

———— ◇ ————

We will consider the following situation.

Main memory has a 32 bit address space.

There are $1024 = 2^{10}$ cache blocks.

The cache block size is 32 bytes.

In consequence an address in cache has 15 bits ($2^{10} \times 2^5 = 2^{15}$). A block address has 10 bits. Consequently a cache address can be partitioned in the form,

$$\underbrace{\text{bbbbbbbbbb}}_{10}\underbrace{\text{ooooo}}_{5}$$

The bits ooooo are called the offset. They are the addresses of bytes within a block.

FULLY ASSOCIATIVE CACHE

To determine if a memory block is in cache, we must eliminate invalid cache blocks — ones that do not contain memory blocks.

A completely cold cache will occur when a process is activated.

Each block of a cache has a valid bit. If it is 1, the block is valid, if not it is invalid.

In a fully associative cache, any memory block can be mapped to any cache block.

A memory address has the form

$$\underbrace{\texttt{ttttttttttttttttttttttttttt}}_{27}\underbrace{\texttt{ooooo}}_{5}$$

The 27-bit tag $\texttt{ttttttttttttttttttttttttttt}$ uniquely determines the memory block.

Each block has a tag field, which is set to the block it contains (if any).

To find out if a block is in cache, one searches the tag fields.

For a large cache, the search is prohibitively expensive

1024 tags in our example.

SET-ASSOCIATIVE CACHE

In a set associative cache, the cache blocks are divided into sets of 2^k blocks. (We will take $k = 2$) for our example.

Each memory block is assigned to a set.

The memory block can be placed anywhere in its set.

Such a cache is called a 2^k-way set-associative cache.

A 4-WAY SET ASSOCIATIVE CACHE

To implement a 4-way set-associative cache, we partition the memory address space as follows.

$$\underbrace{\texttt{ttttttttttttttttttt}}_{19}\underbrace{\texttt{ssssssss}}_{8}\underbrace{\texttt{ooooo}}_{5}$$

The 8-bits ssssssss are the number needed to address the $256 = 1024/4$ sets of four blocks.

The nineteen bits ttttttttttttttttttt are a tag that uniquely determines the memory block given the set address.

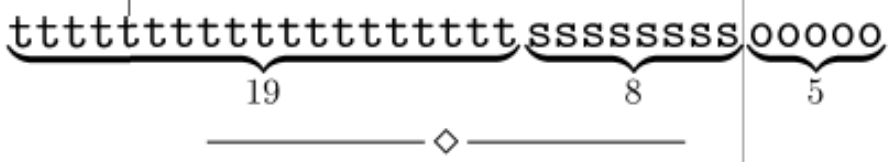
Since there are four sets withing each block, we can give them an address bb.

It is determined by searching the tag fields in the set.

The final address for the memory reference is

ssssssssbboooo

SUMMARY



1. Split out the tag, set, and offset fields of the address.
2. Compare the tag for the memory address with each of the tags in the four cache blocks beginning with the addresses

   ssssssss0000000
   ssssssss0100000
   ssssssss1000000
   ssssssss1100000

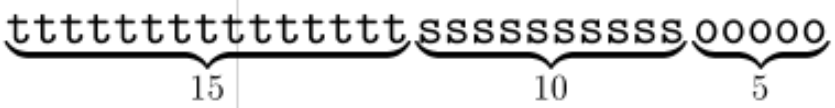
3. If there is no match, declare a cache miss and quit.
4. If there is a match with block bb, the cache address is

   ssssssssbboooo.

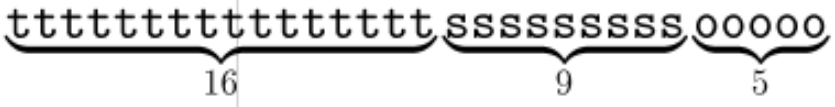
FOLLOWUP

The associativity depends on the size of the set field in the memory address.

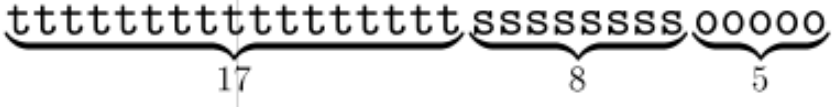
1-way set-associative (direct mapped)



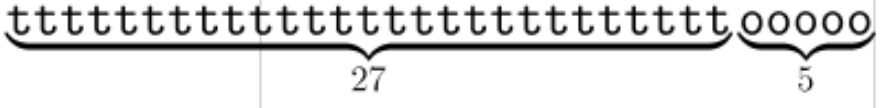
2-way set-associative



4-way set-associative



1024-way set-associative (fully associative)



Typically caches are 1-, 2-, or 4-way set associative.

REPLACEMENT (THIRD QUESTION)

When a new memory block arrives and its set is full, one of the blocks of the set must be displaced.

For direct mapped caches, this is not a problem. There is only one choice.

For higher associativity there are different replacement strategies.

The least recently used (LRU) strategy chooses the block that has been in the set longest without a hit.

An adaptive replacement cache (ARC) is more elaborate. It takes into account memory blocks that have been recently replaced.

Random replacement works well and it cheap to implement.

But it may result in unreproducible behavior.

RESPONSE TO READ AND WRITE HITS

Read hits are no problem. Since they make not change in memory or cache the read is honored from cache.

Write hits are more difficult, since any writes to cache must eventually appear in memory.

A write back cache writes only the cache but not main memory.

When a write occurs, a dirty bit is set to indicate that cache and memory disagree.

When the block is replaced, its contents are written back to memory.

This is efficient if there is good locality of reference, but otherwise there will be many expensive write backs.

A write through cache writes both cache and memory.

There are no write backs.

Because writes to main memory are slow, if the cache block is written to frequently it will slow down.

A buffer can help—somewhat.

RESPONSE TO READ AND WRITE MISSES

On a read miss, swap in the missing block.

The action on a write miss depends on the type of cache.

> On a write-back cache swap in the missing block. If locality of reference holds, it will soon be referenced again.
>
> On a write-through cache, it is not clear what swapping will accomplish. Doing nothing is reasonable.

ADDITIONAL CACHES

The cache connected directly to the CPU (the L1 cache) is small and fast. This has two consequences.

- Because it is small, miss rates are large.
- Because it is fast, refreshing it from memory will slow it down.

Thus a second cache (the L2 cache) that is larger and slower is frequently placed between the L1 cache and main memory.

A common practice is to have two caches: a data cache for data and and an instruction cache for instructions.

- In a cache hierarchy, the L2 cache often holds both instructions and data, in which case it is called a unified cache.