The Memory Hierarchy

Chapter 6

Outline

- Storage technologies and trends
- Locality of reference
- Caching in the memory hierarchy

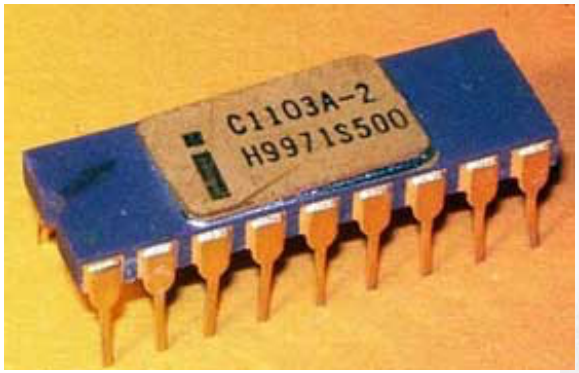
Random-Access Memory (RAM)

- **Key features**
  - RAM is traditionally packaged as a chip.
  - Basic storage unit is normally a cell (one bit per cell).
  - Multiple RAM chips form a memory.
- **Static RAM (SRAM)**
  - Each cell stores a bit with a four or six-transistor circuit.
  - Retains value indefinitely, as long as it is kept powered.
  - Relatively insensitive to electrical noise (EMI), radiation, etc.
  - Faster and more expensive than DRAM.
- **Dynamic RAM (DRAM)**
  - Each cell stores bit with a capacitor. One transistor is used for access
  - Value must be refreshed every 10-100 ms.
  - More sensitive to disturbances (EMI, radiation,…) than SRAM.
  - Slower and cheaper than SRAM.

SRAM vs DRAM Summary

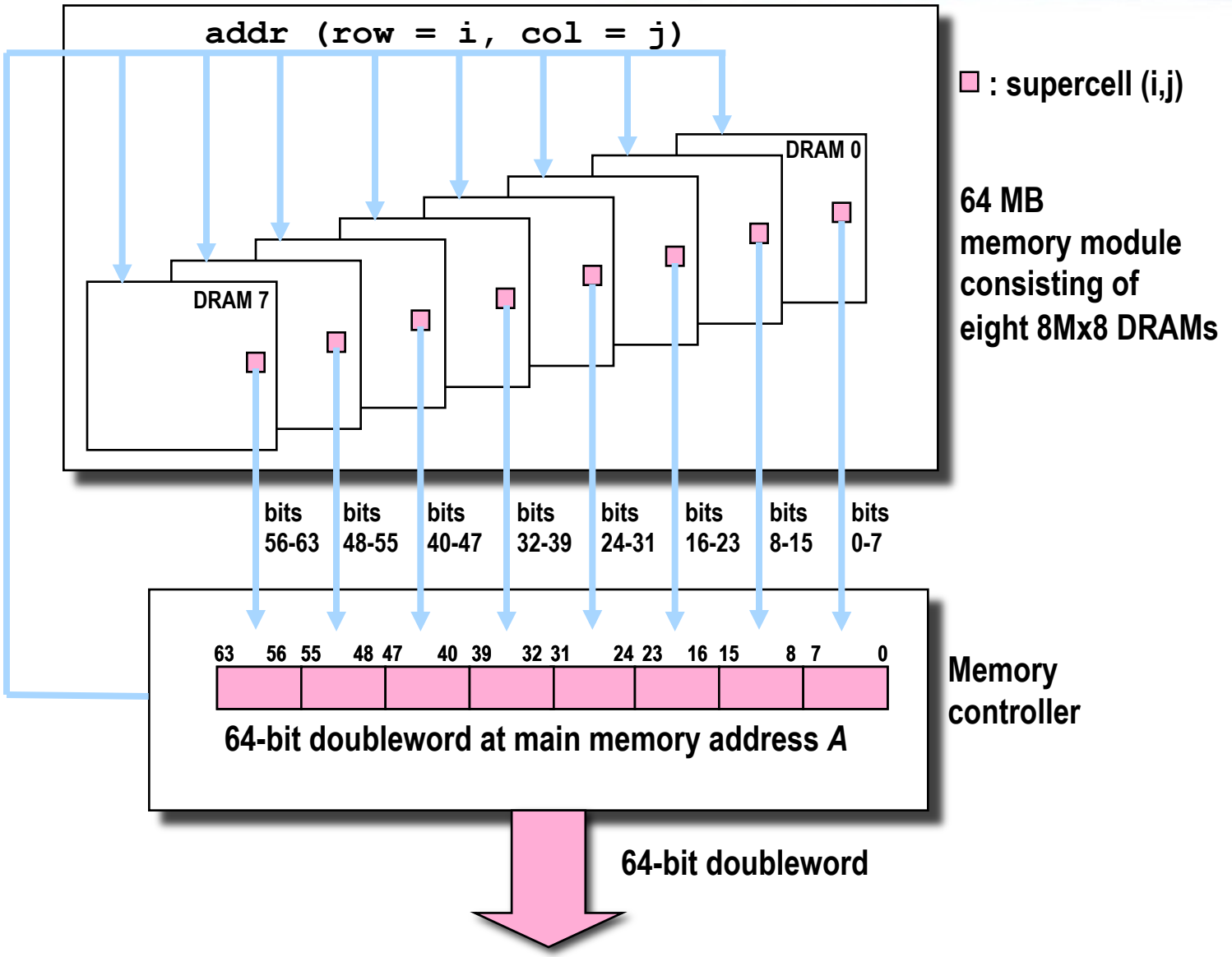
	Transistors per bit	Access time	Needs refresh?	Needs EDC?	Cost	Applications
SRAM	4 or 6	1X	No	Maybe	100x	Cache memories
DRAM	1	10X	Yes	Yes	1X	Main memories, frame buffers

Enhanced DRAMs



- **Basic DRAM cell has not changed since its invention in 1966.**
  - Commercialized by Intel in 1970.
- **DRAM cores with better interface logic and faster I/O :**
  - Synchronous DRAM (SDRAM)
    - Uses a conventional clock signal instead of asynchronous control
    - Allows reuse of the row addresses (e.g., RAS, CAS, CAS, CAS)
  - Double data-rate synchronous DRAM (DDR SDRAM)
    - Double edge clocking sends two bits per cycle per pin
    - Different types distinguished by size of small prefetch buffer:
      - DDR (2 bits), DDR2 (4 bits), DDR4 (8 bits)
    - By 2010, standard for most server and desktop systems
    - Intel Core i7 supports only DDR3 SDRAM

Memory Modules



: supercell (i,j)

64 MB
memory module
consisting of
eight 8Mx8 DRAMs

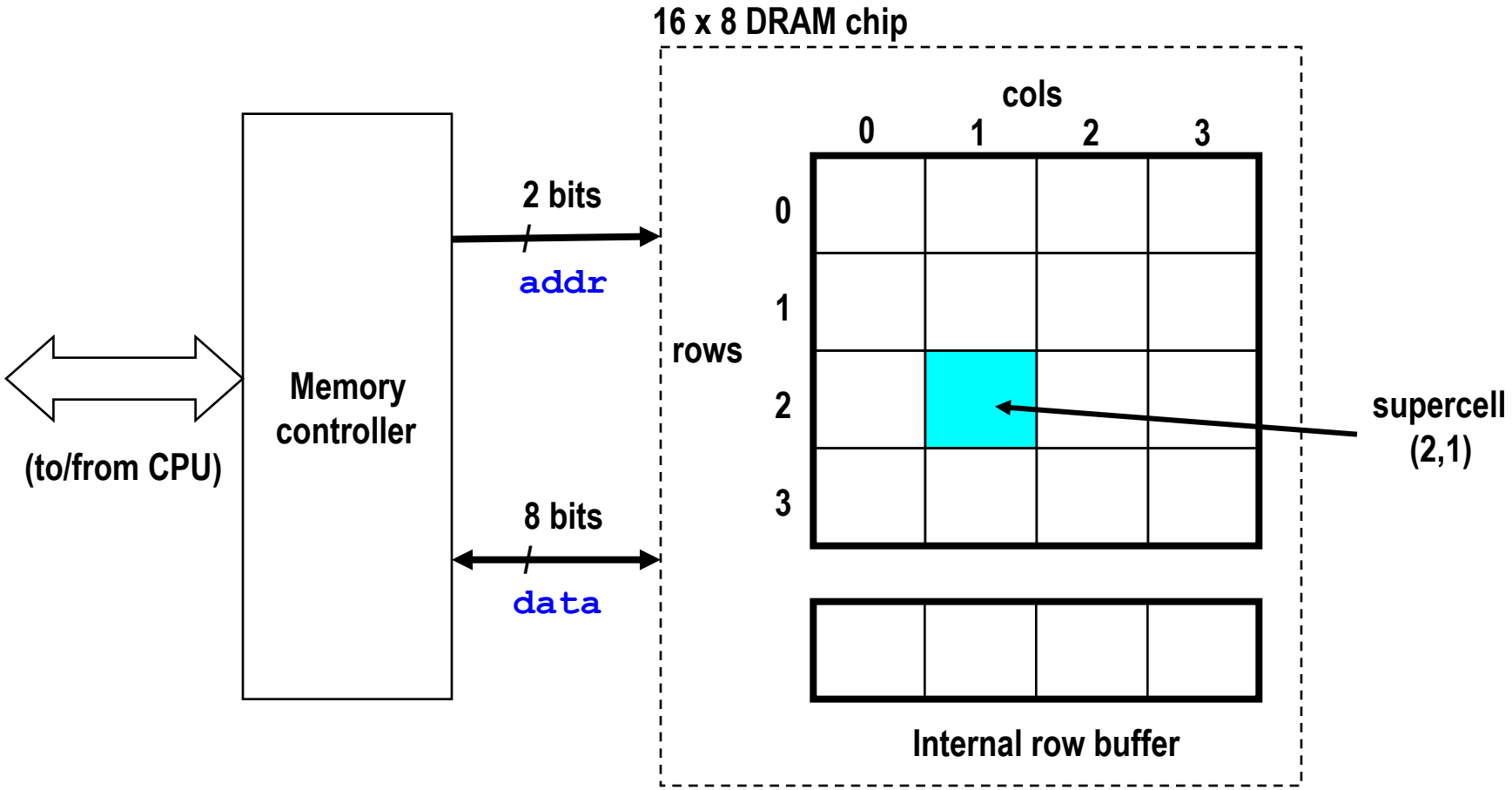
Memory
controller

Conventional DRAM Organization

- **d × w DRAM: (e.g. 16 × 8 = 128 bits)**

  Each supercell has w bits (e.g., w=8)

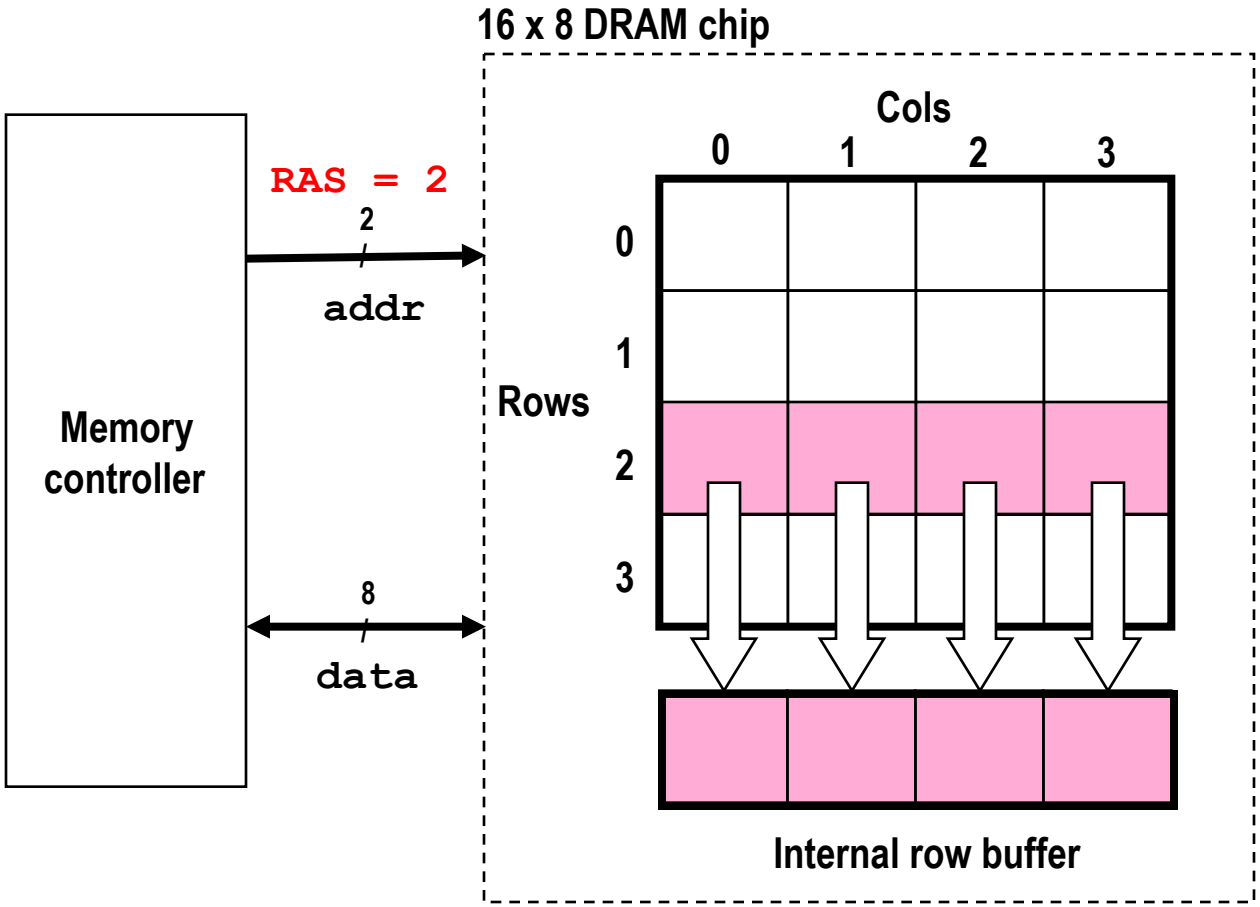
  Number of supercells is rows × cols = d (e.g., 4 × 4 = 16)



Reading DRAM Supercell (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.

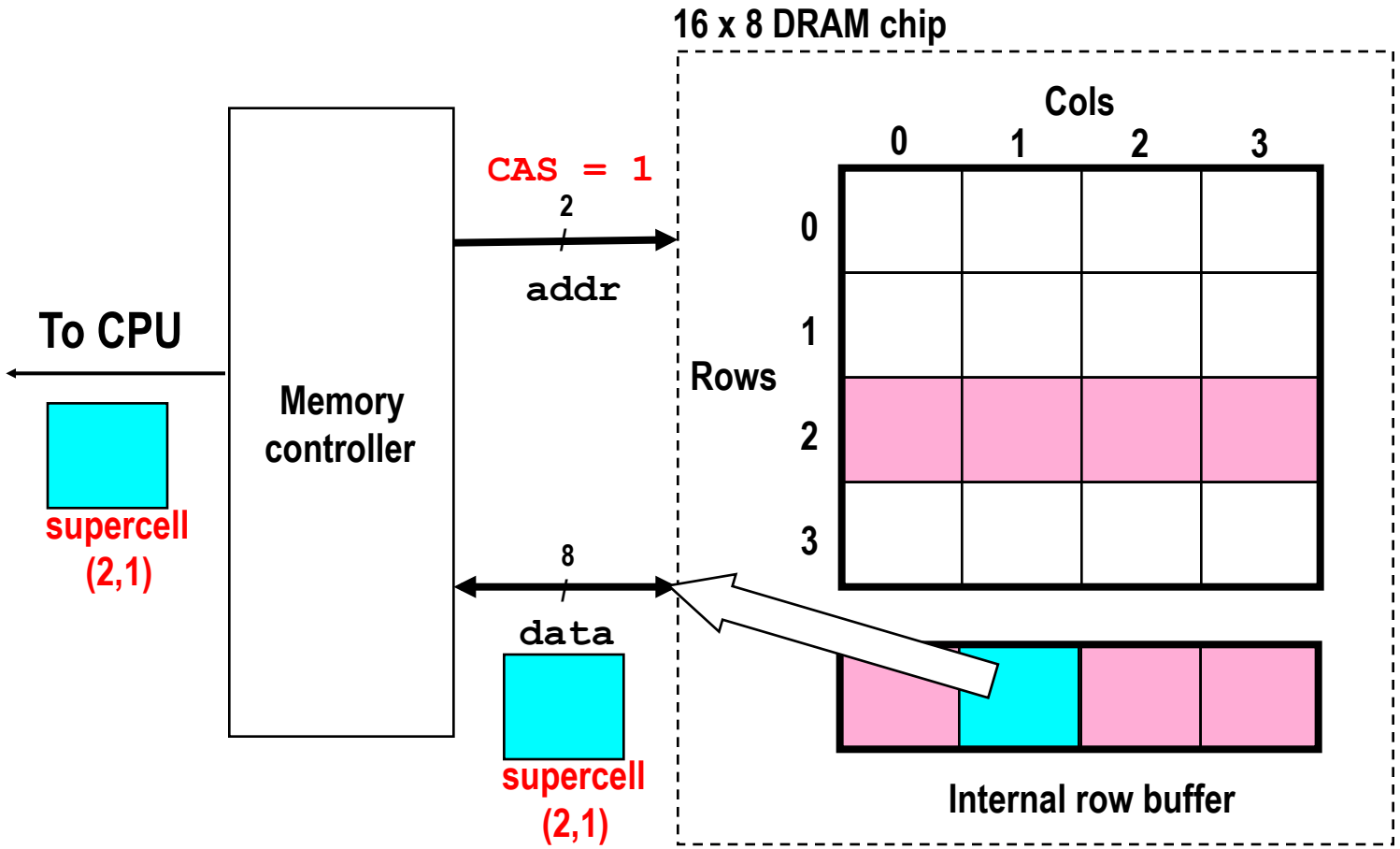
Step 1(b): Row 2 copied from DRAM array to row buffer.



Reading DRAM Supercell (2,1)

Step 2(a): Column access strobe (CAS) selects column 1.

Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.



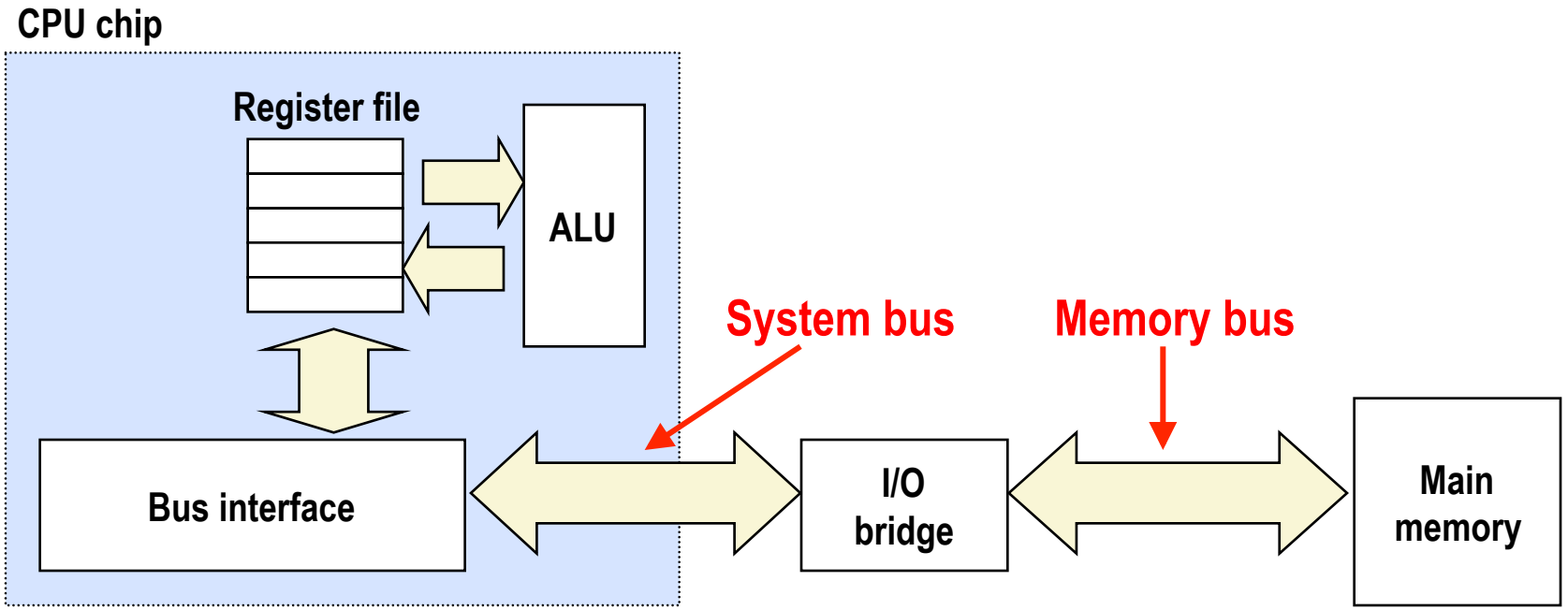
Nonvolatile Memories

- **DRAM and SRAM are volatile memories**
  - Lose information if powered off.
- **Nonvolatile memories retain value even if powered off**
  - Read-only memory (ROM): programmed during production
  - 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
    - Wears out after about 100,000 erasings.
- **Uses for Nonvolatile Memories**
  - Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,…)
  - Solid state disks (replace rotating disks in thumb drives, smart phones, mp3 players, tablets, laptops,…)
  - Disk caches

Traditional Bus Structure

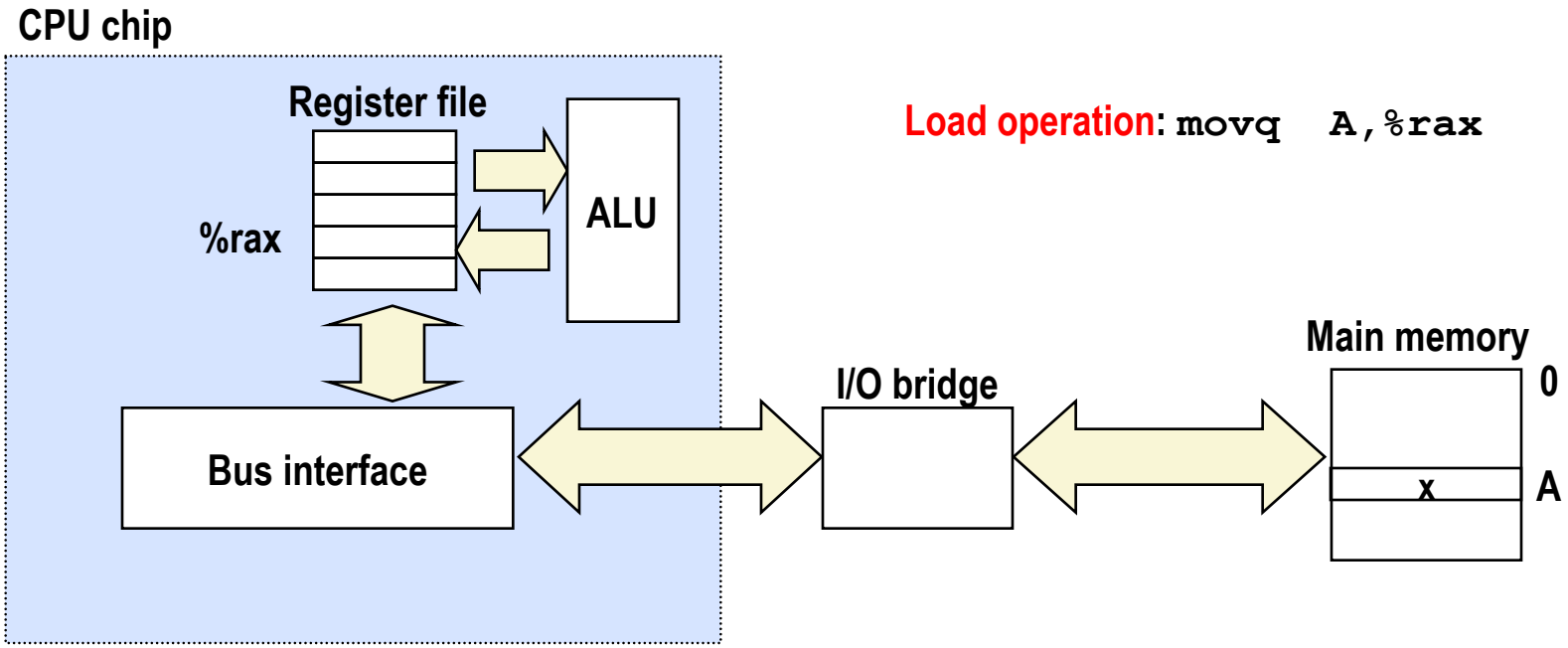
A bus is a collection of parallel wires that carry address, data, and control signals.

Buses are typically shared by multiple devices.



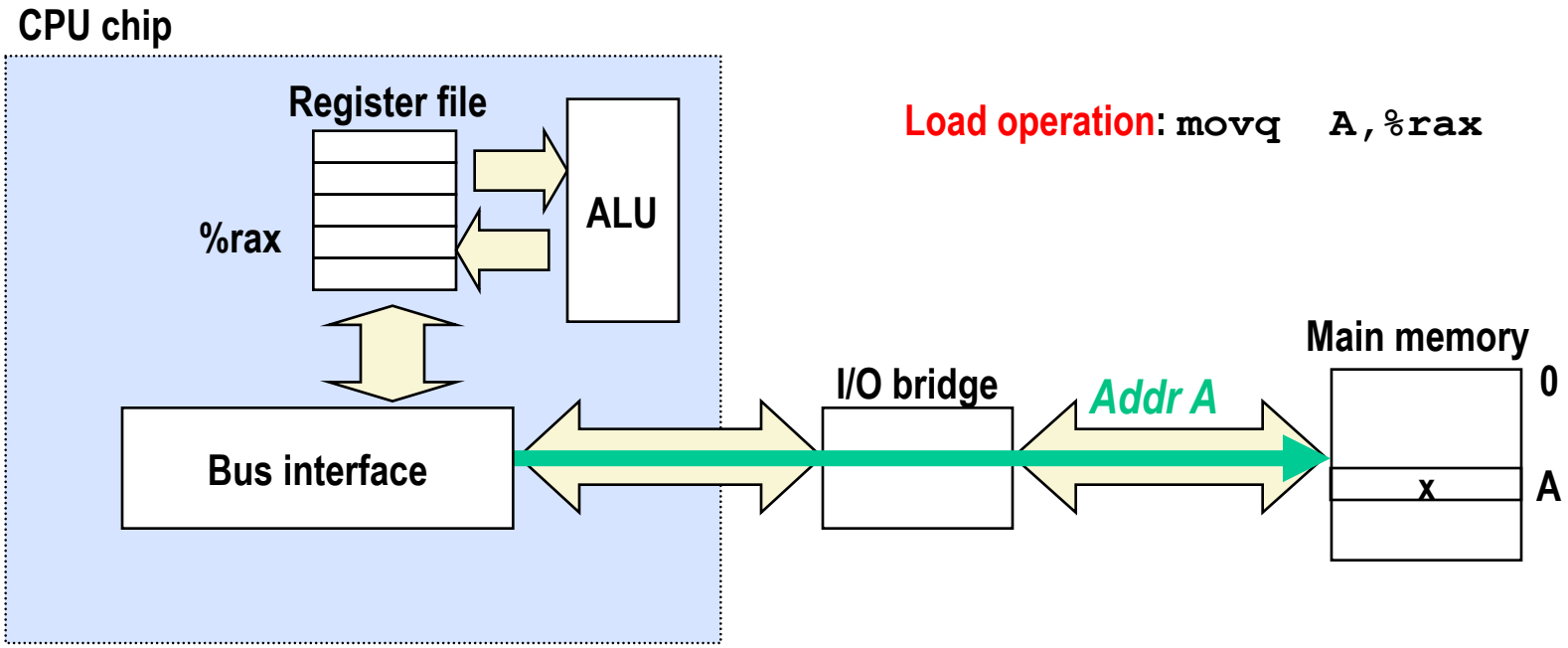
Memory Read Transaction

The CPU places address A on the memory bus.



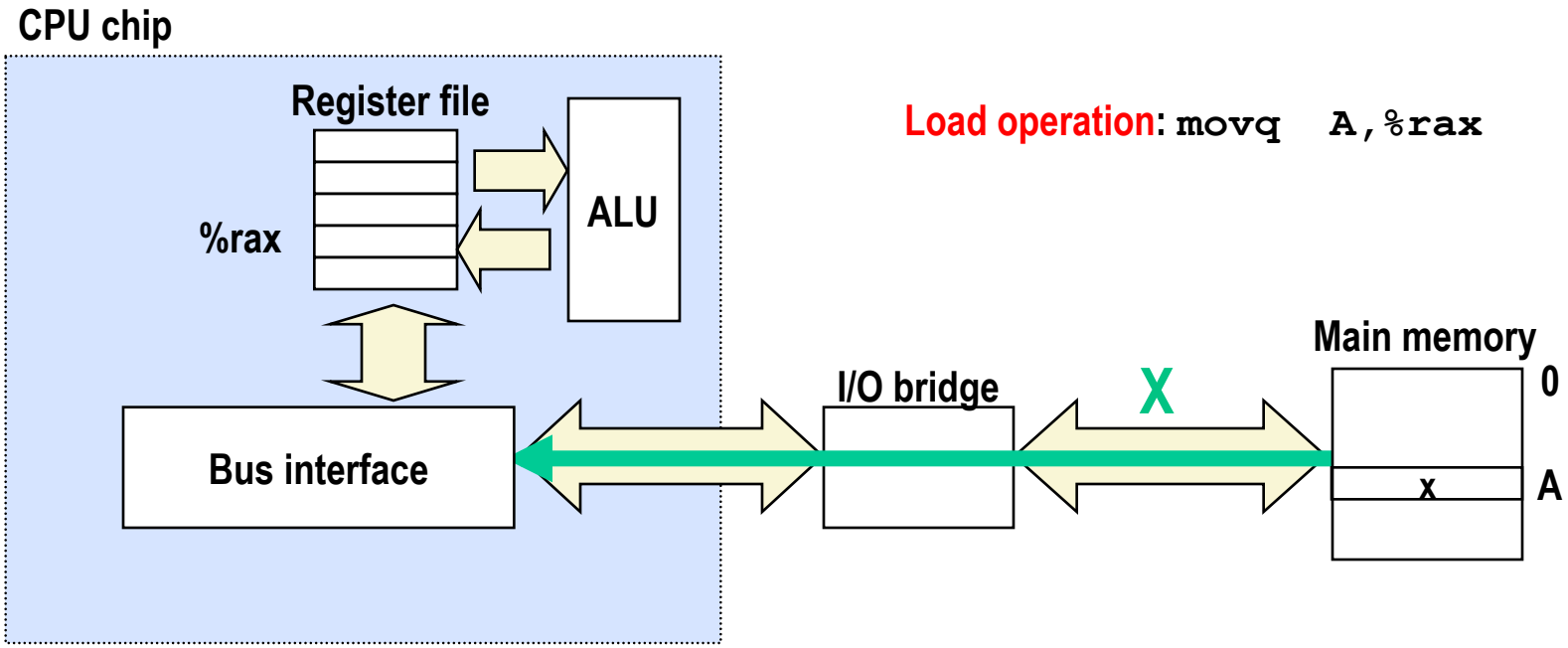
Memory Read Transaction

The CPU places address A on the memory bus.



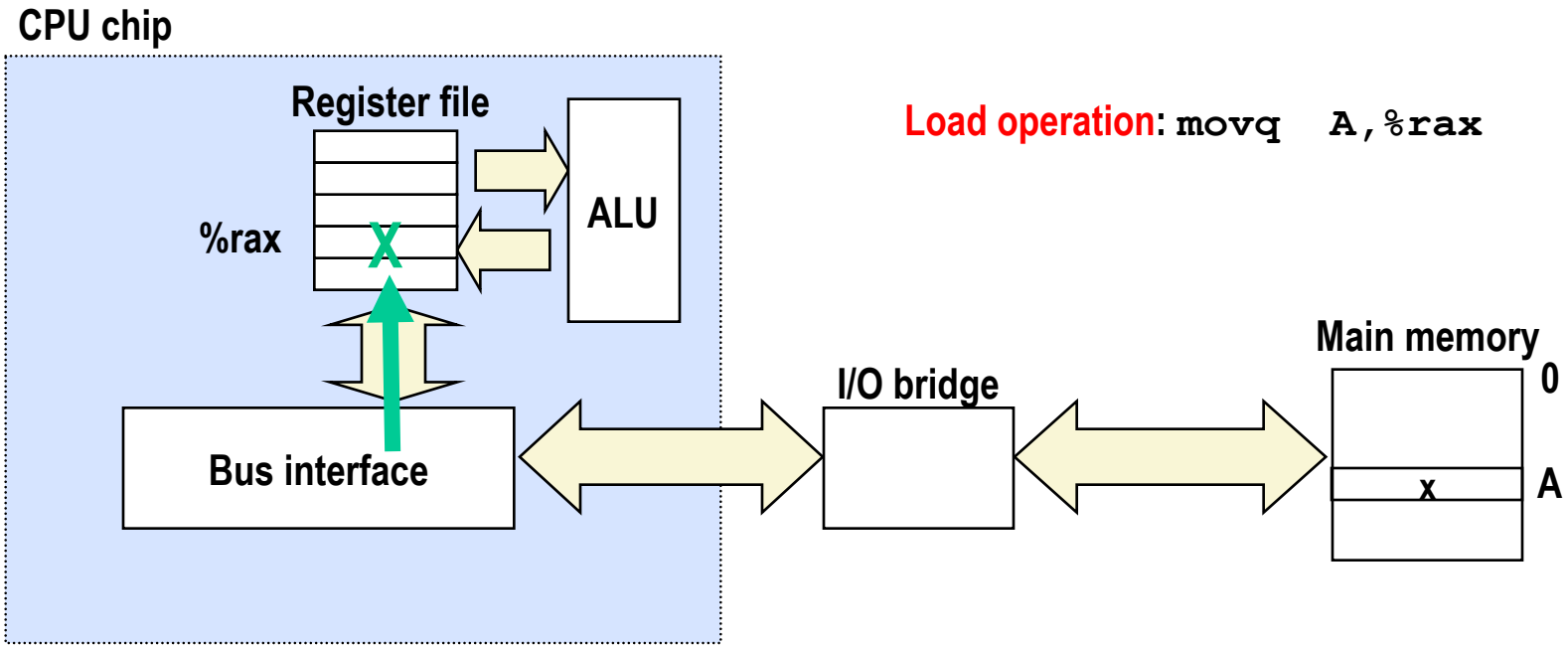
Memory Read Transaction

Main memory reads A from the memory bus, retrieves word x, and places it on the bus.



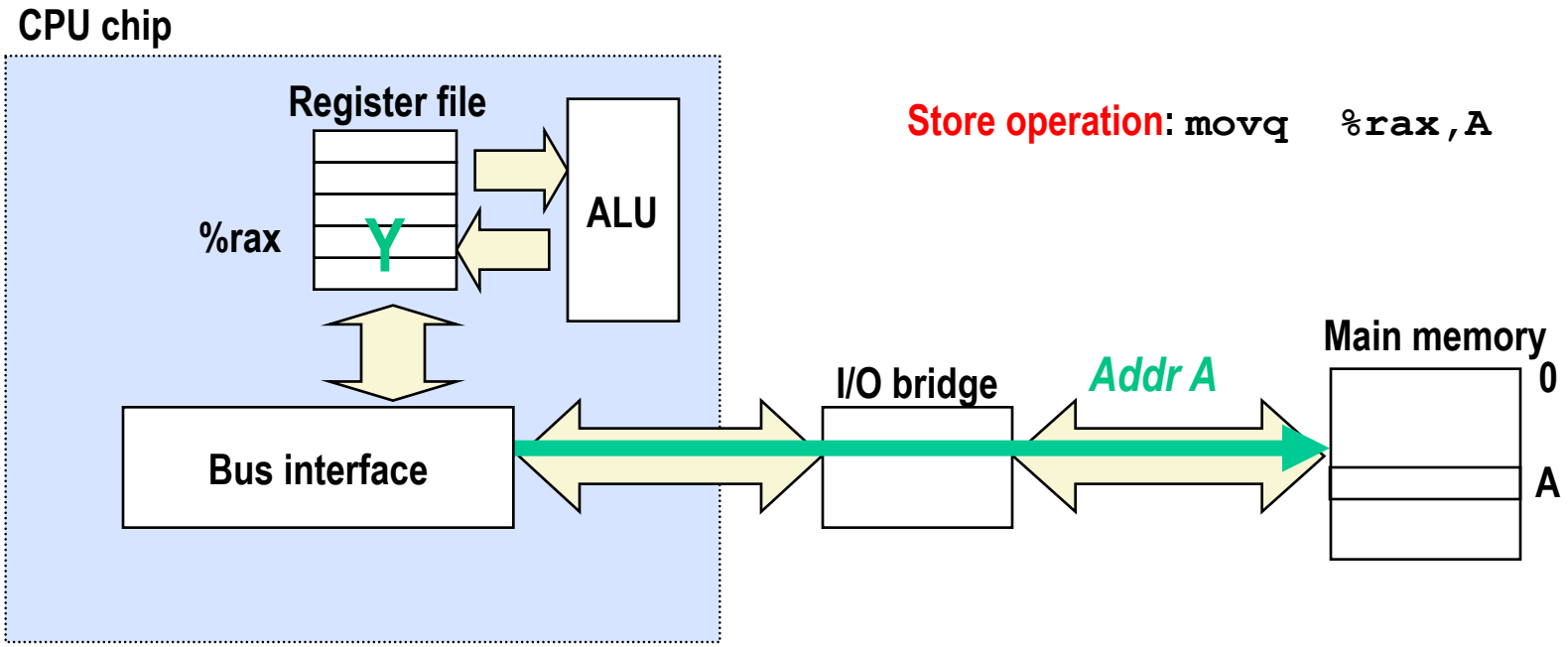
Memory Read Transaction

The CPU reads word x from the bus and copies it into register %rax.



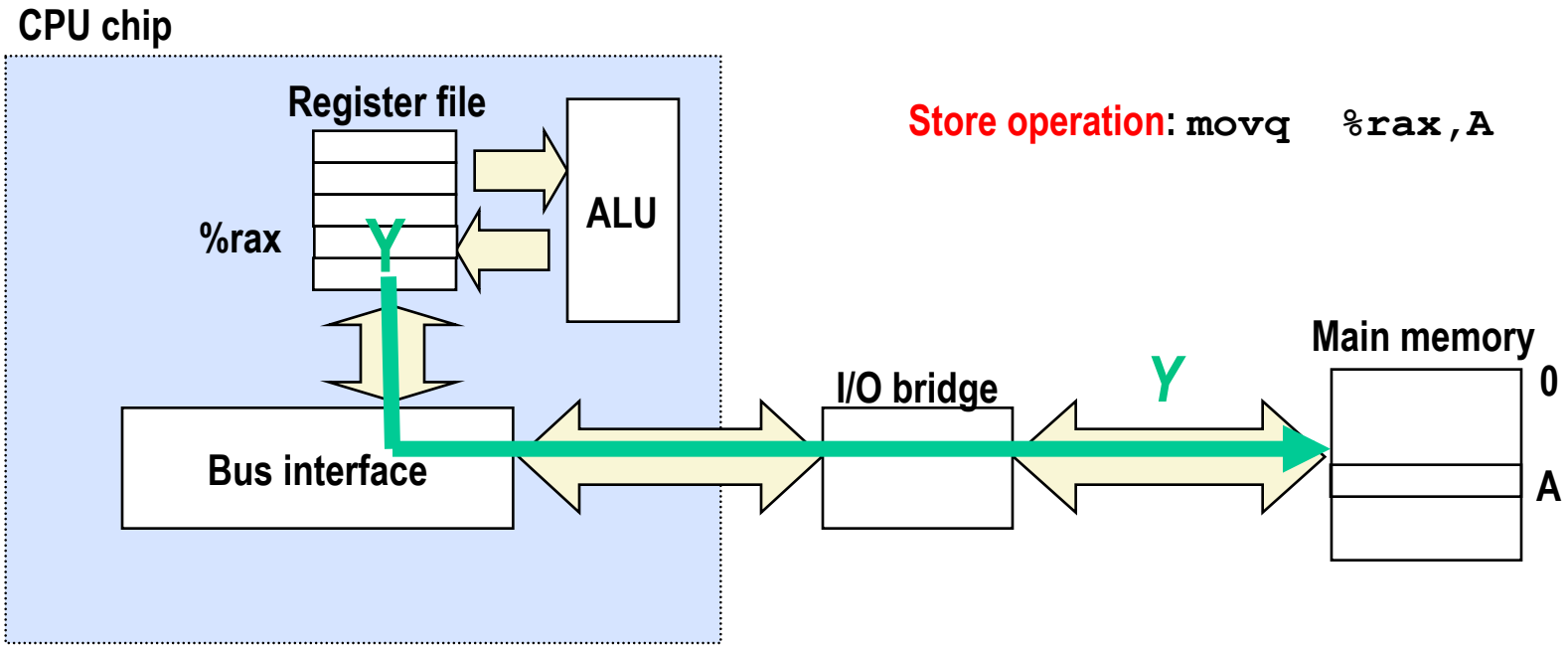
Memory Write Transaction

The CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.



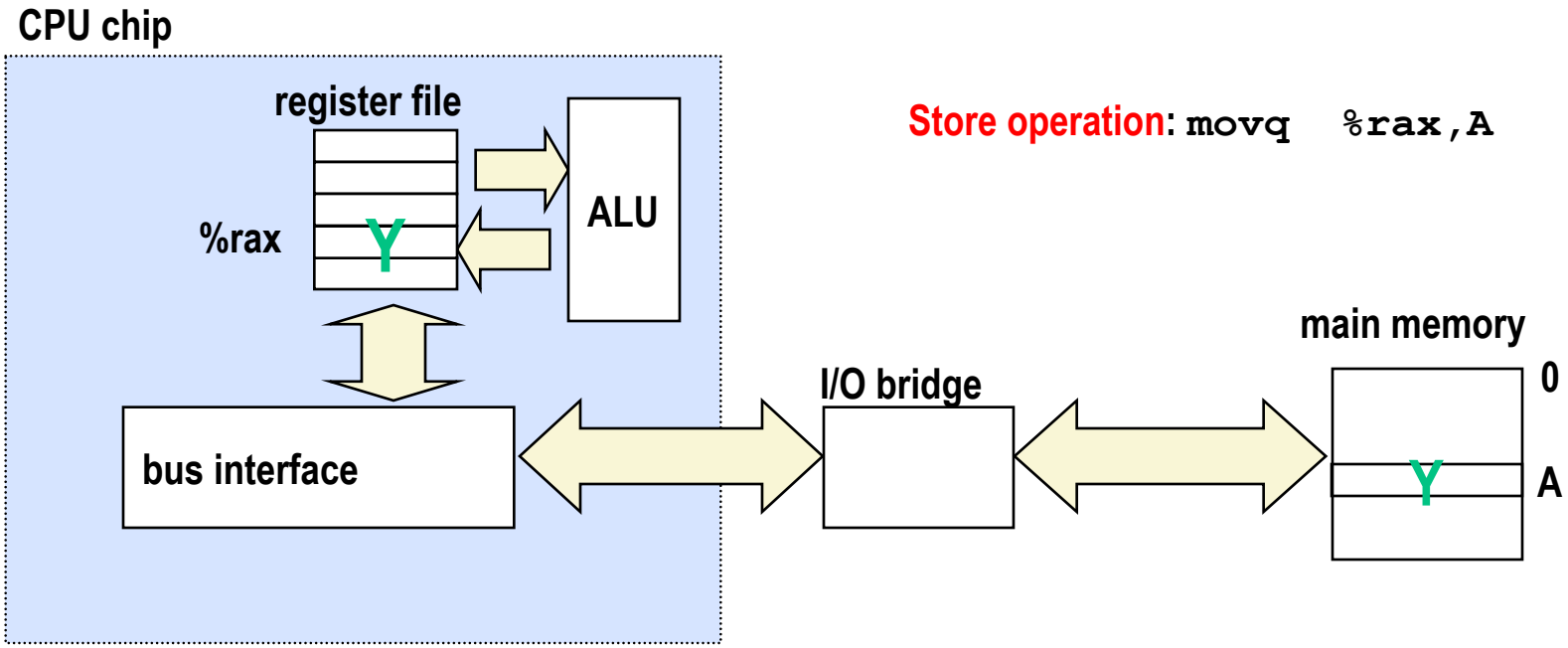
Memory Write Transaction

The CPU places data word Y on the bus.



Memory Write Transaction

Main memory reads data word Y from the bus and stores it at address A.



What's Inside A Disk Drive?

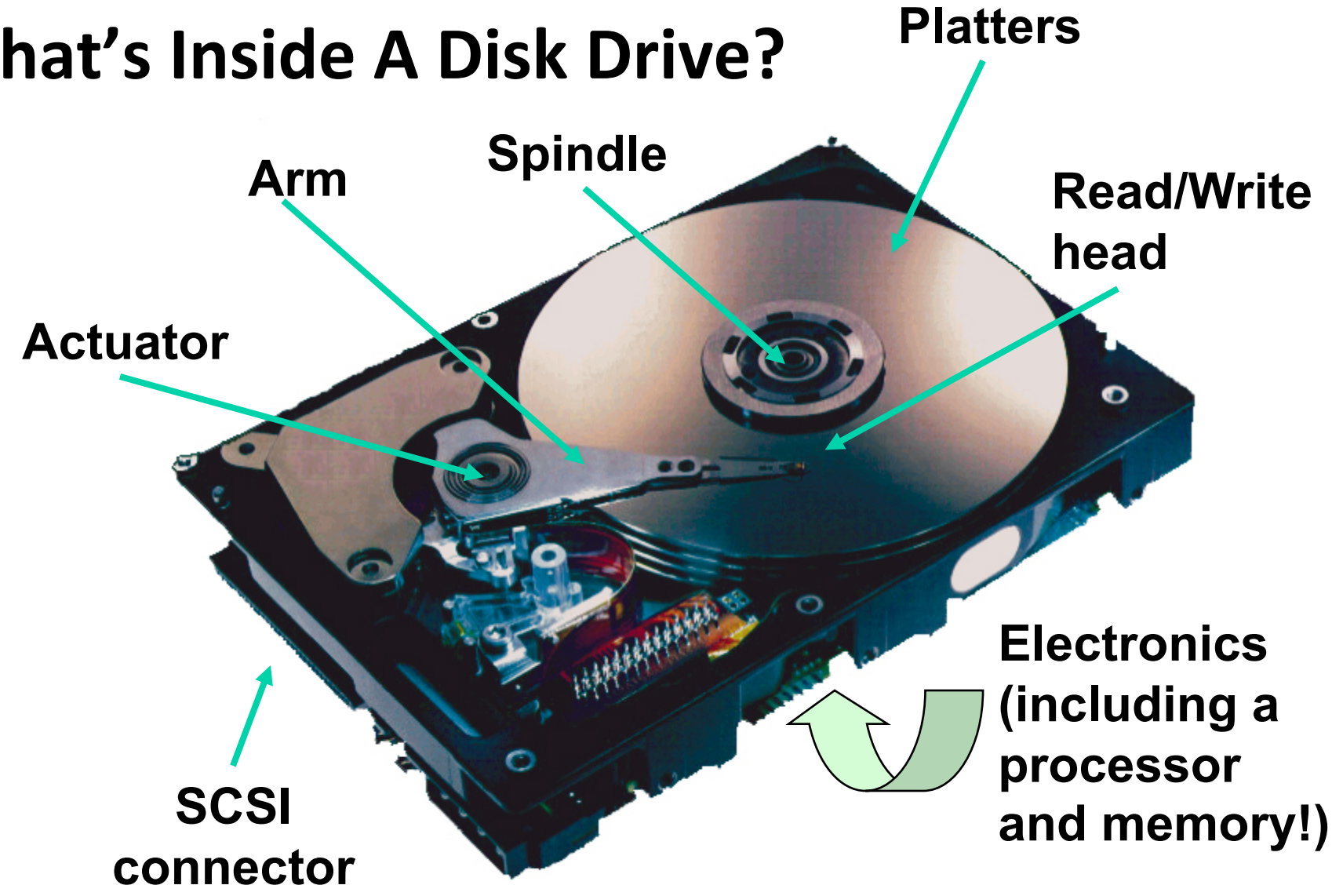
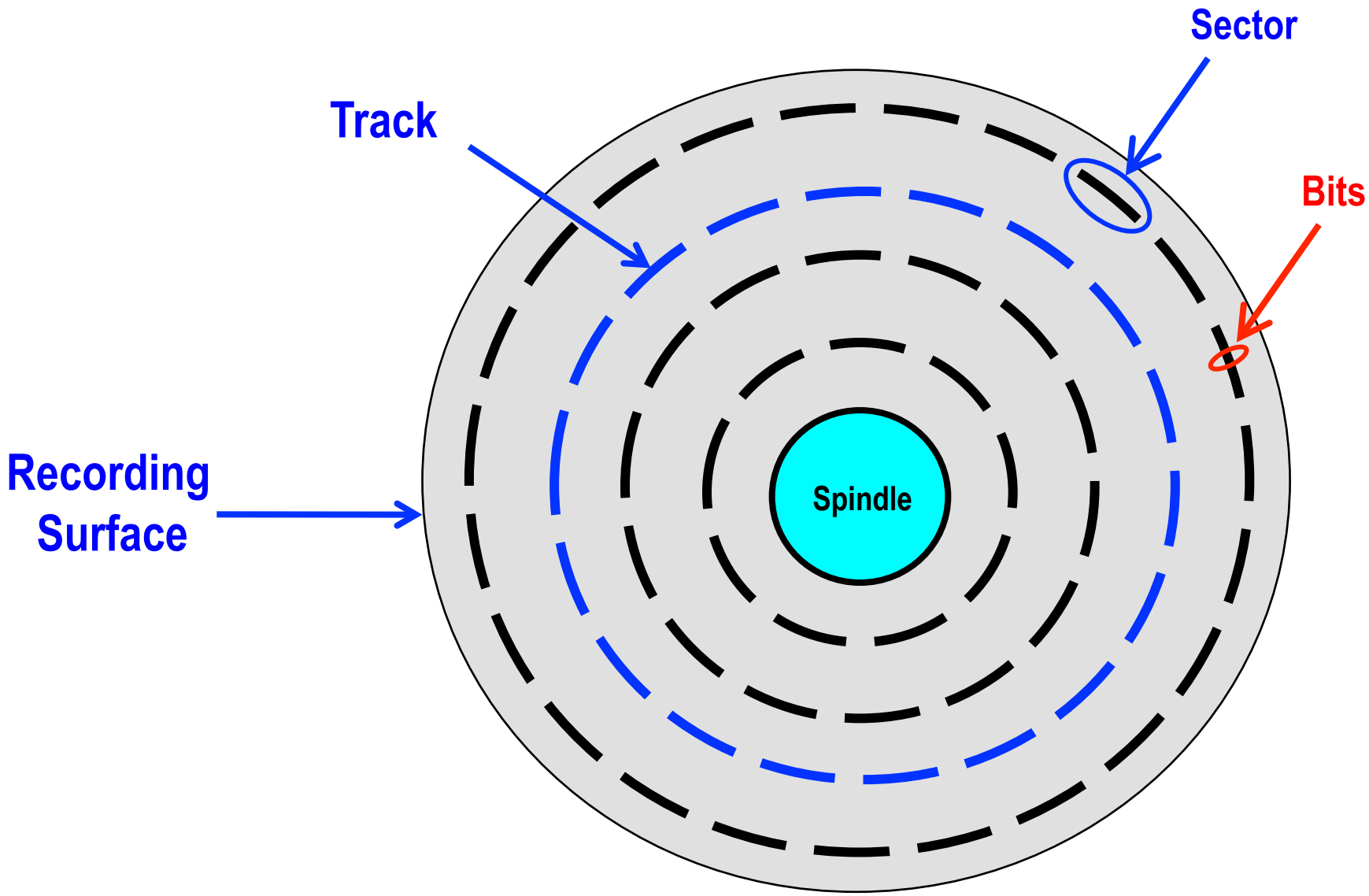


Image courtesy of Seagate Technology

Disk Geometry



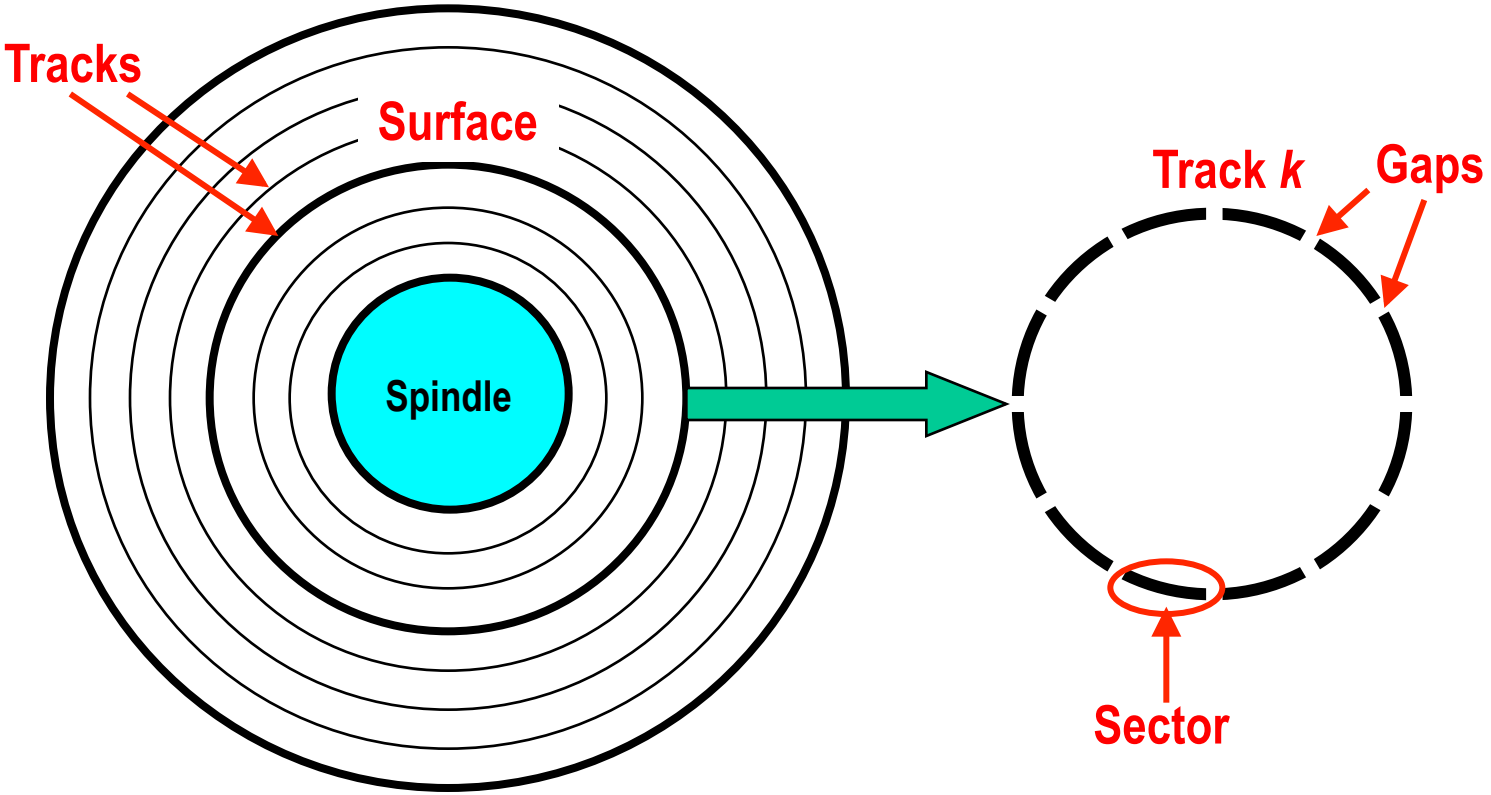
Disk Geometry

A sector is a sequence of bytes (usually 512 bytes).

Each track consists of sectors separated by gaps.

Each surface consists of many tracks.

Disks consist of platters, each with two surfaces.



Disk Capacity

Capacity: maximum number of bits that can be stored.

Vendors express capacity in units of gigabytes (GB)

Note: 1 GB = 10^9 Bytes, not 2^{30} Bytes

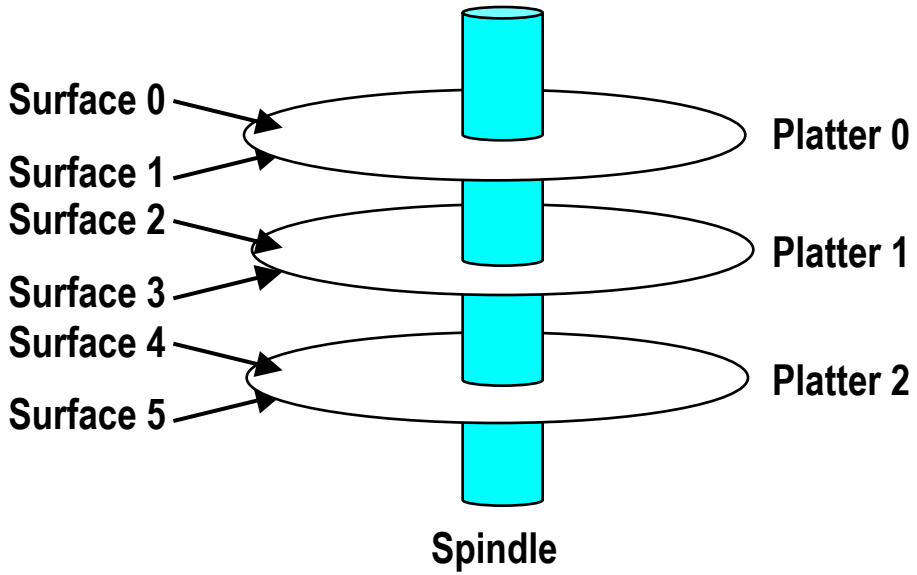
Capacity is determined by these technology factors:

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.

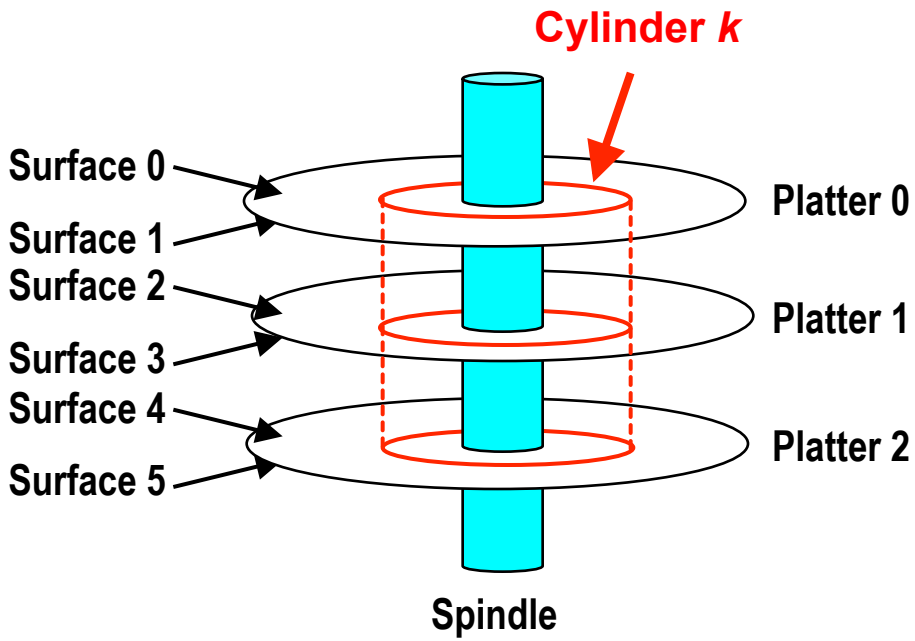
Areal density (**bits/in^2**): product of recording and track density.

Disk Geometry (Muliple-Platter View)



Disk Geometry (Muliple-Platter View)

Aligned tracks form a cylinder.

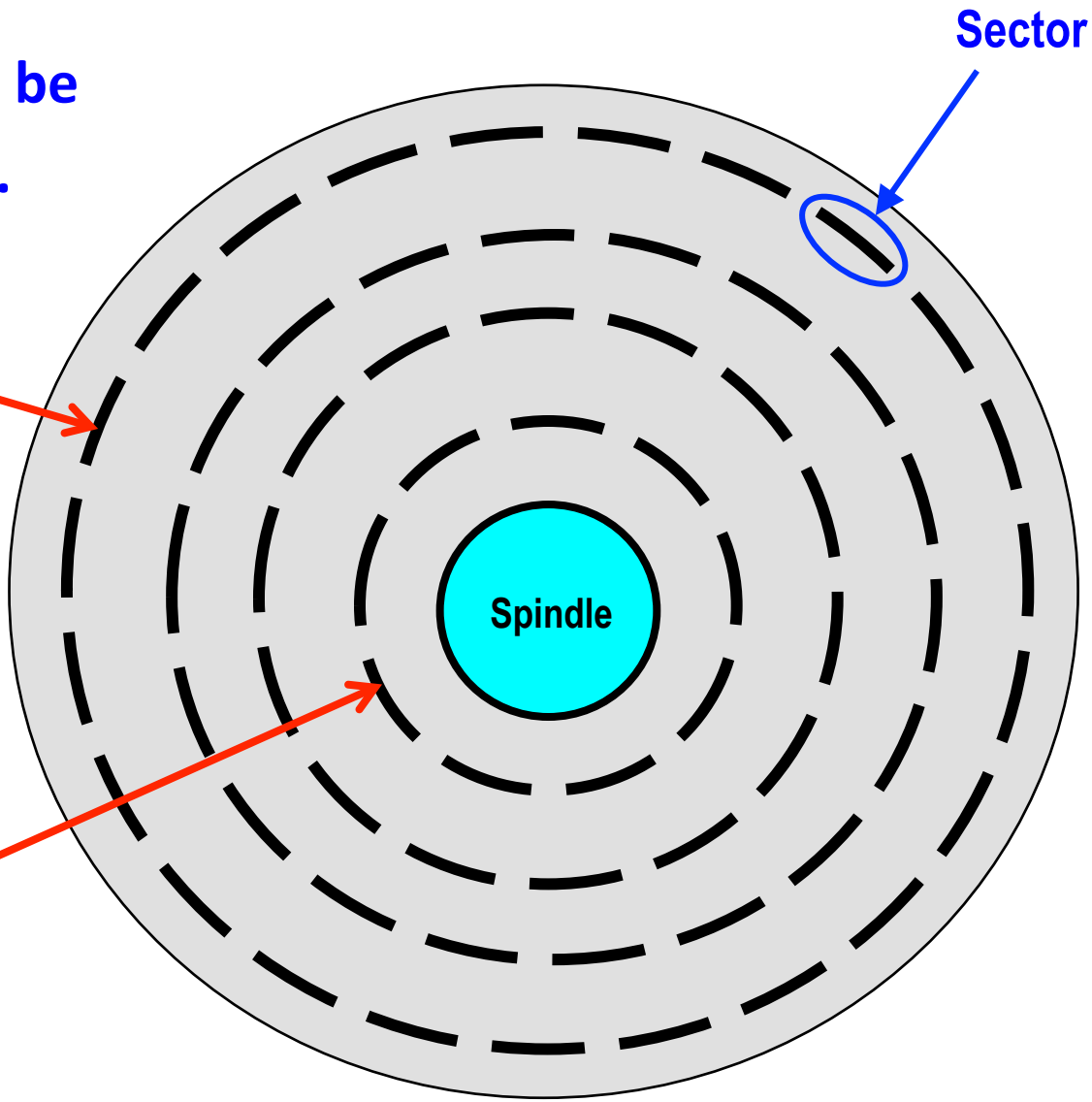


Recording Zones

Want each sector to be about the same size.

Outer tracks have lots of sectors

Inner tracks have fewer sectors



Recording Zones

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.

- Outer zones have more sectors/track
- Inner zones have fewer sectors/track

Use **average** number of sectors/track when computing capacity.

Computing Disk Capacity

Capacity = (# bytes/sector) × (average # sectors/track) × (# tracks/surface) × (# surfaces/platter) × (# platters/disk)

Example:

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

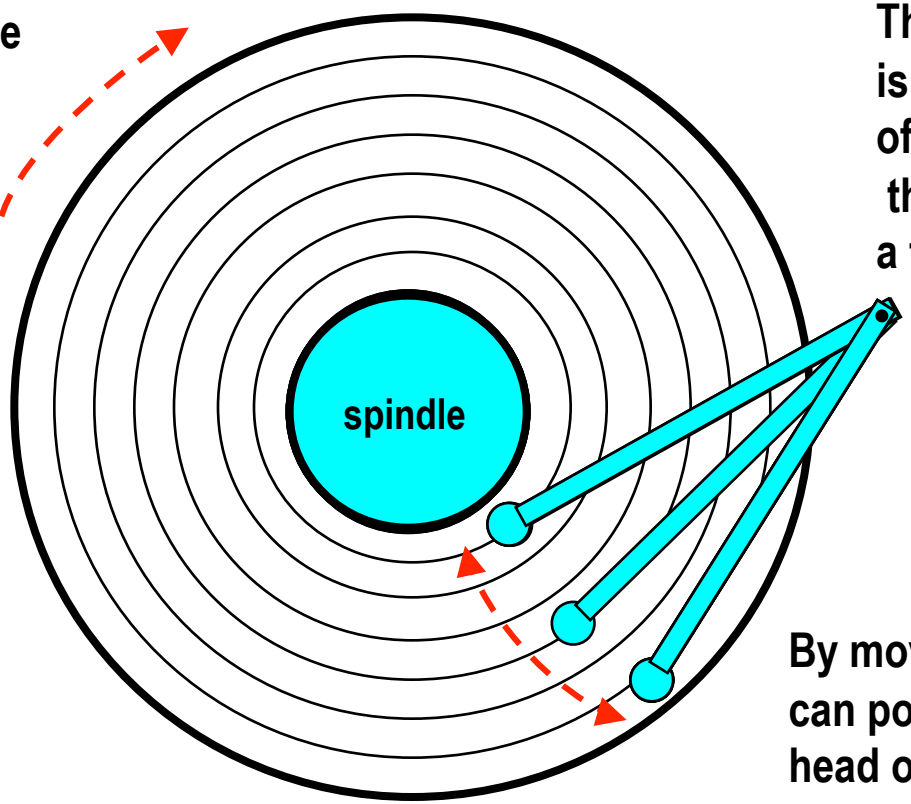
Capacity = 512 × 300 × 20000 × 2 × 5

= 30,720,000,000

= 30.72 GB

Disk Operation (Single-Platter View)

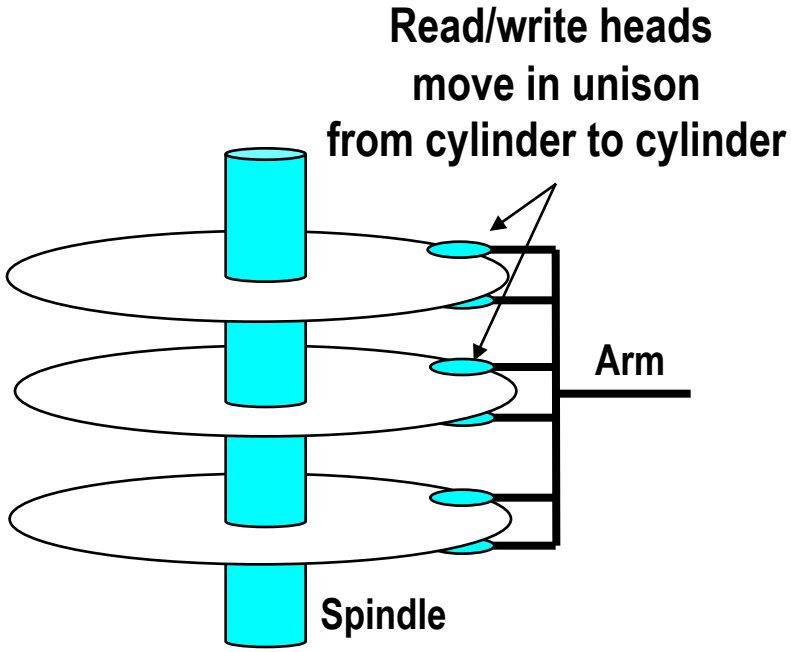
The disk surface spins at a fixed rotational rate



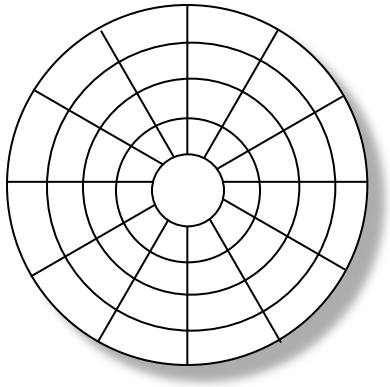
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

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

Disk Operation (Multi-Platter View)



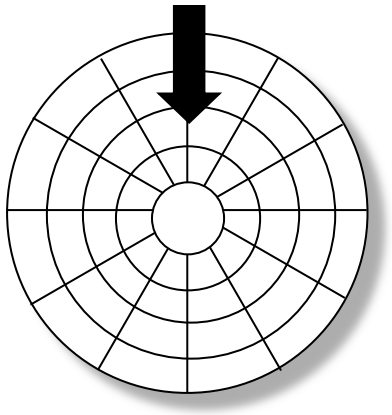
Disk Structure - top view of single platter



Surface organized into tracks

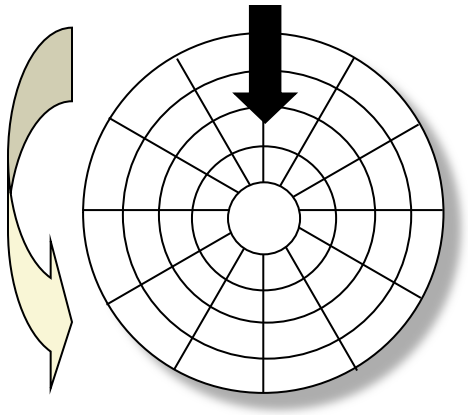
Tracks divided into sectors

Disk Access



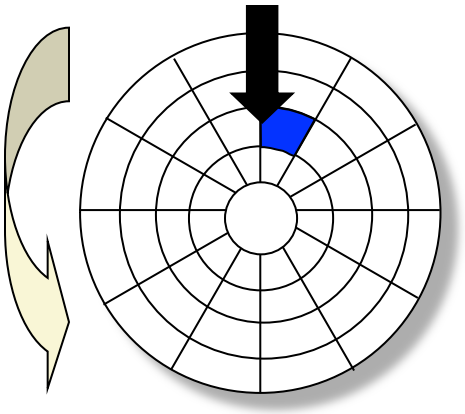
Head in position above a track

Disk Access



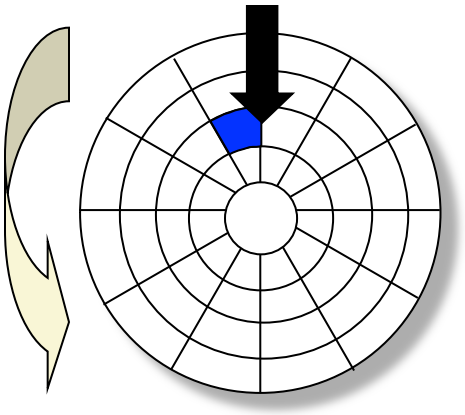
Rotation is counter-clockwise

Disk Access – Read



About to read blue sector

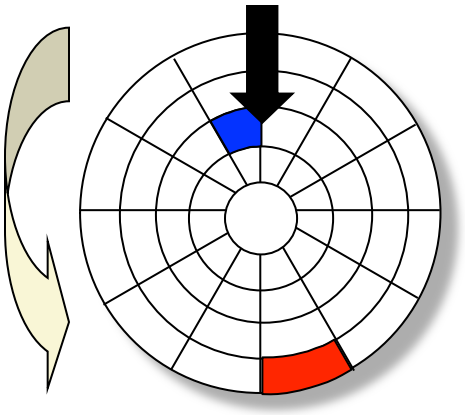
Disk Access – Read



After BLUE read

After reading blue sector

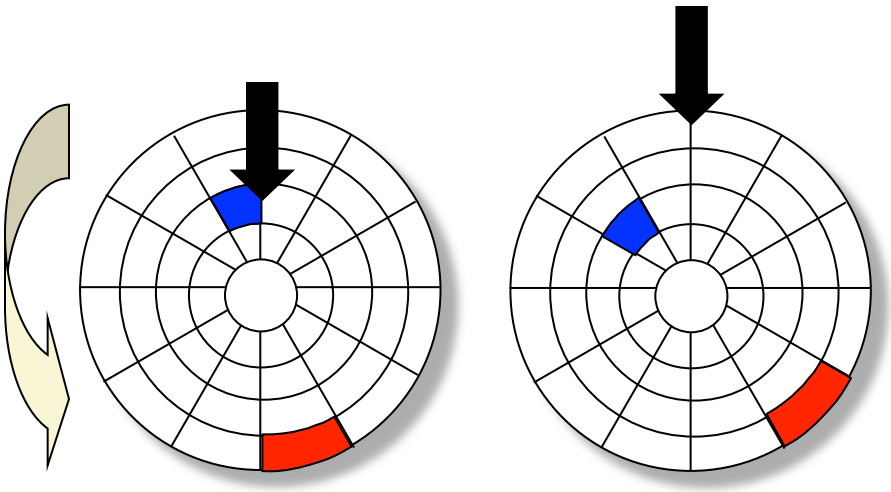
Disk Access – Read



After BLUE read

Red request scheduled next

Disk Access – Seek

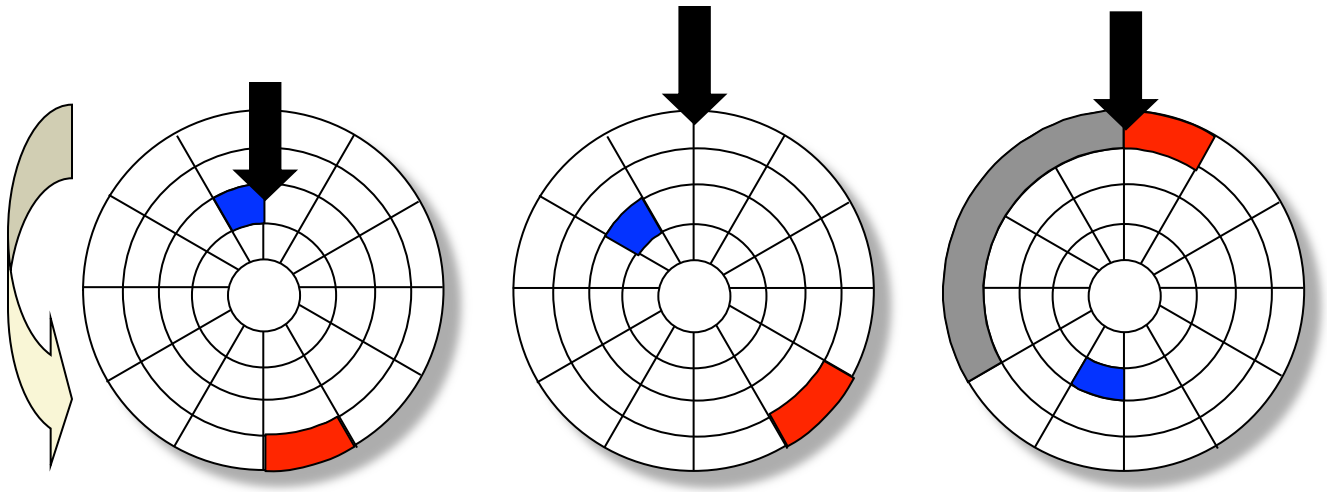


After BLUE read

Seek for RED

Seek to red's track

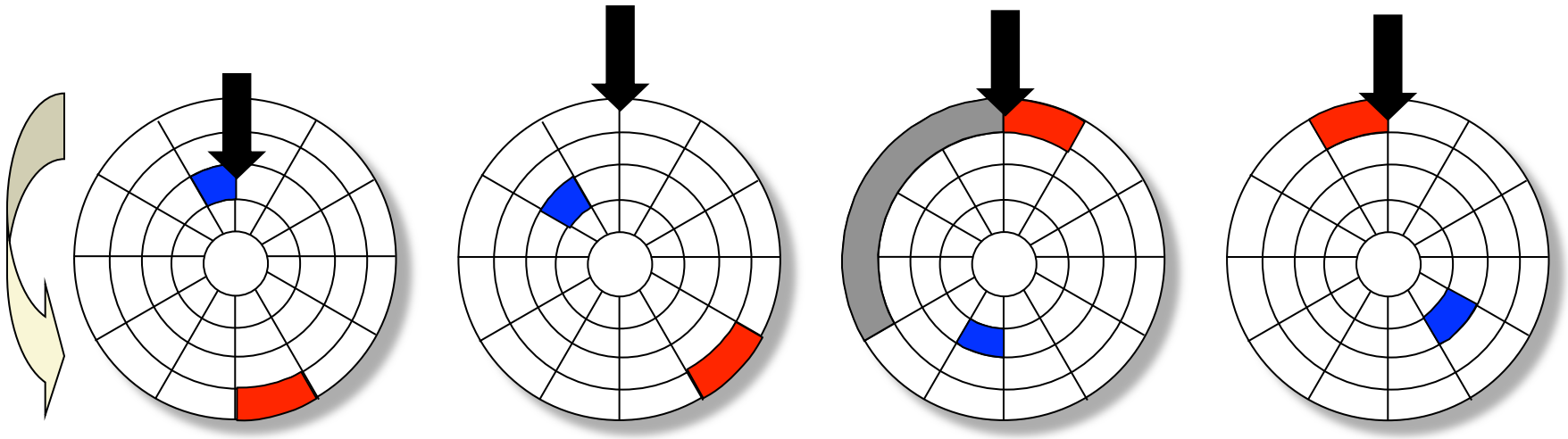
Disk Access – Rotational Latency



After BLUE read Seek for RED Rotational latency

Wait for red sector to rotate around

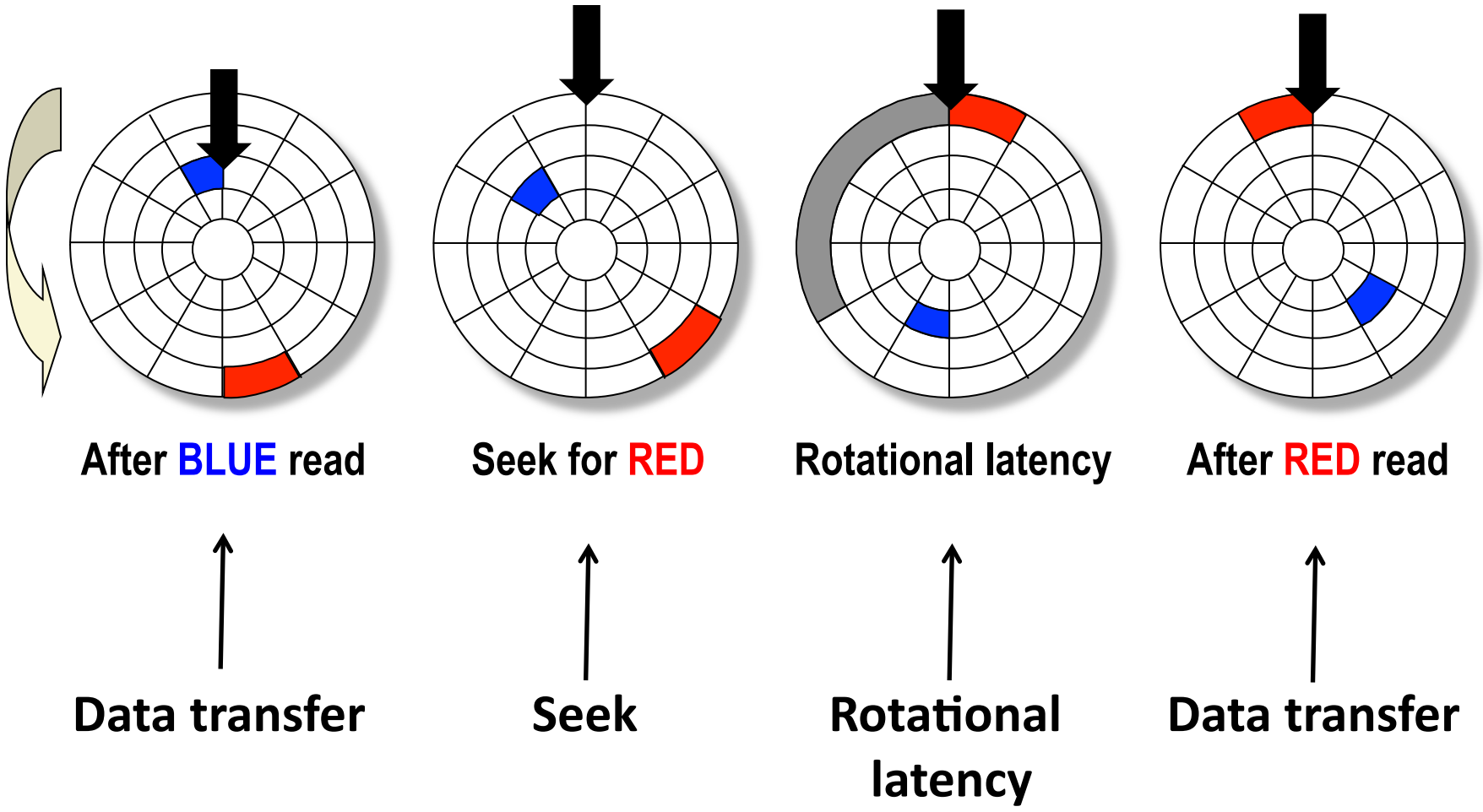
Disk Access – Read



After BLUE read | Seek for RED | Rotational latency | After RED read

Complete read of red

Disk Access – Service Time Components



Disk Access Time

Average time to access some target sector:

$$T_{access} = T_{avg\ seek} + T_{avg\ rotation} + T_{avg\ transfer}$$

Seek time ($T_{avg\ seek}$)

Time to position heads over cylinder containing target sector.

Typical: 3—9 ms

Rotational latency ($T_{avg\ rotation}$)

Time waiting for first bit of target sector to pass under r/w head.

$T_{avg\ rotation}$ = 1/2 × 1/RPM × 60,000 ms/min

Typical: 1/2 × 1/7200 × 60,000 ms/min = 4 ms

Transfer time ($T_{avg\ transfer}$)

Time to read the bits in the target sector.

$T_{avg\ transfer}$ = 1/RPM × 1/(avg # sectors/track) × 60,000 ms/min

Typical: 1/7200 × 1/400 sectors × 60,000ms/min = 0.02 ms

Disk Access Time Example

Given:

Rotational rate = 7,200 RPM

Average seek time = 9 ms.

Avg # sectors/track = 400.

Derived:

$T_{avg\ rotation}$ = 1/2 × 1/7200 RPM × 60,000 ms/min= 4 ms.

$T_{avg\ transfer}$ = 1/7200 RPM × 1/400 secs/track × 60,000 ms/min= 0.02 ms

T_{access} = 9 ms + 4 ms + 0.02 ms = 13.02 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, ...)

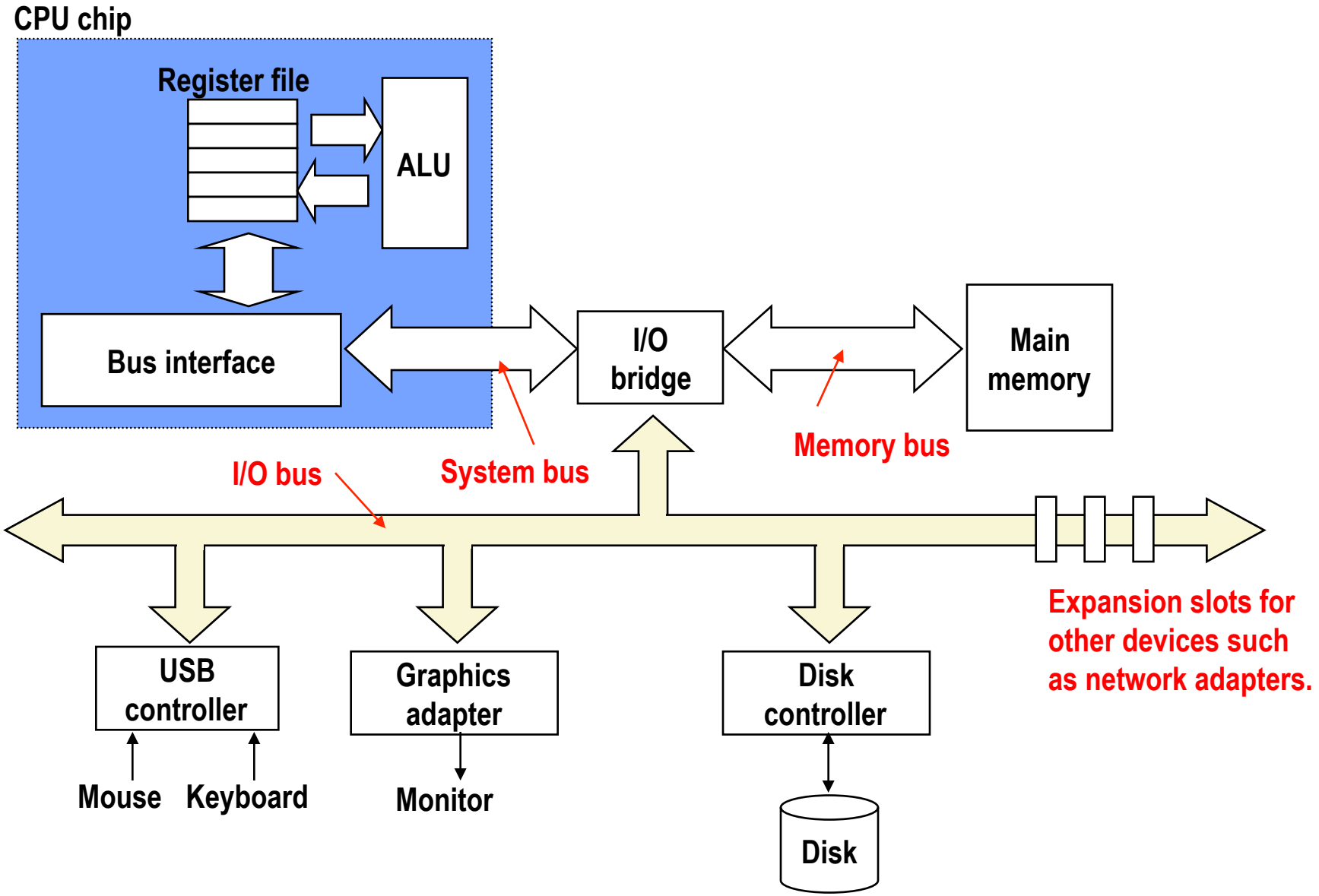
Mapping between logical blocks and actual (physical) sectors

- Maintained by hardware/firmware device called disk controller.
- Converts requests for logical blocks into (surface,track,sector) triples.

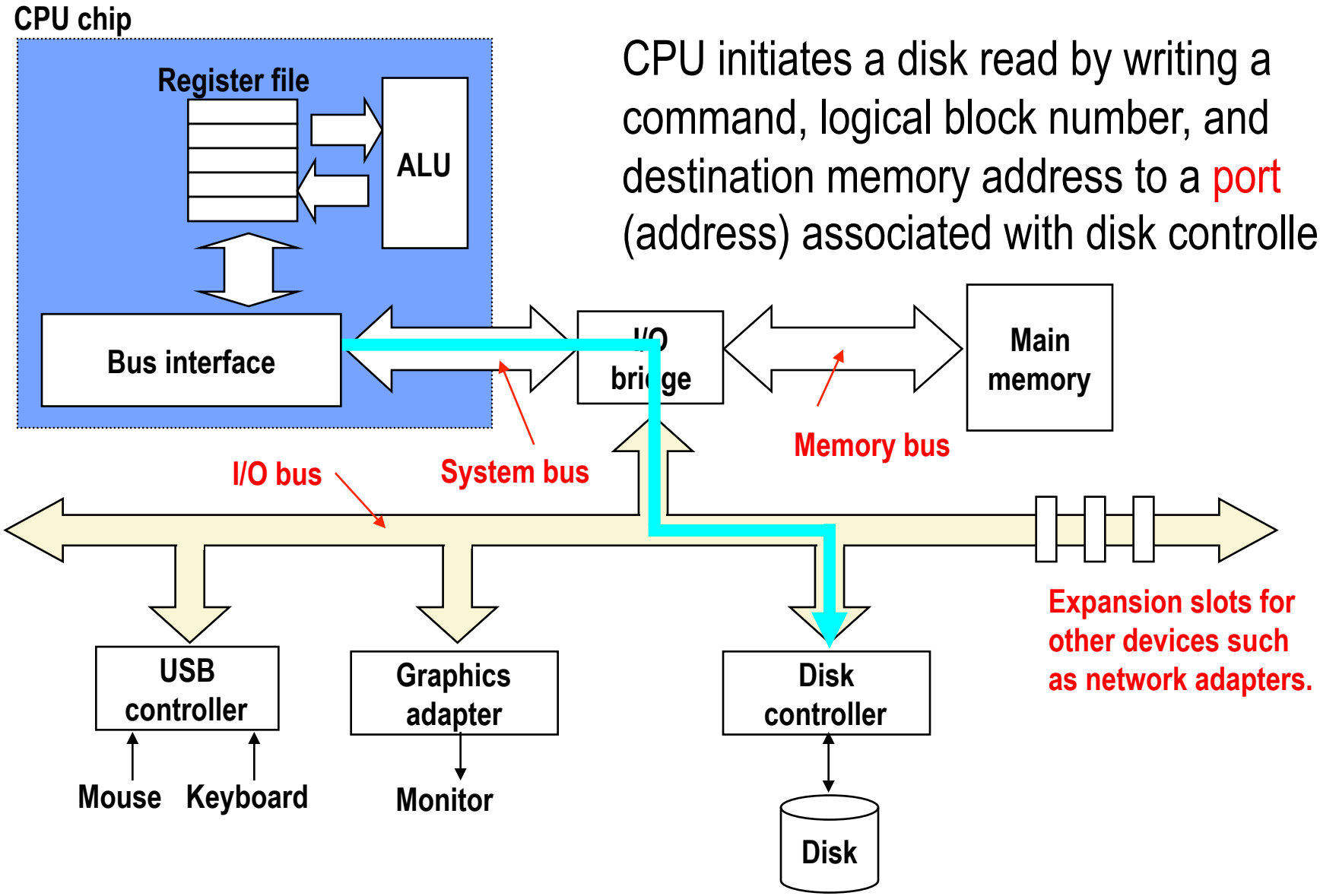
Allows controller to set aside spare cylinders for each zone.

- Accounts for the difference in “formatted capacity” and “maximum capacity”.

I/O Bus

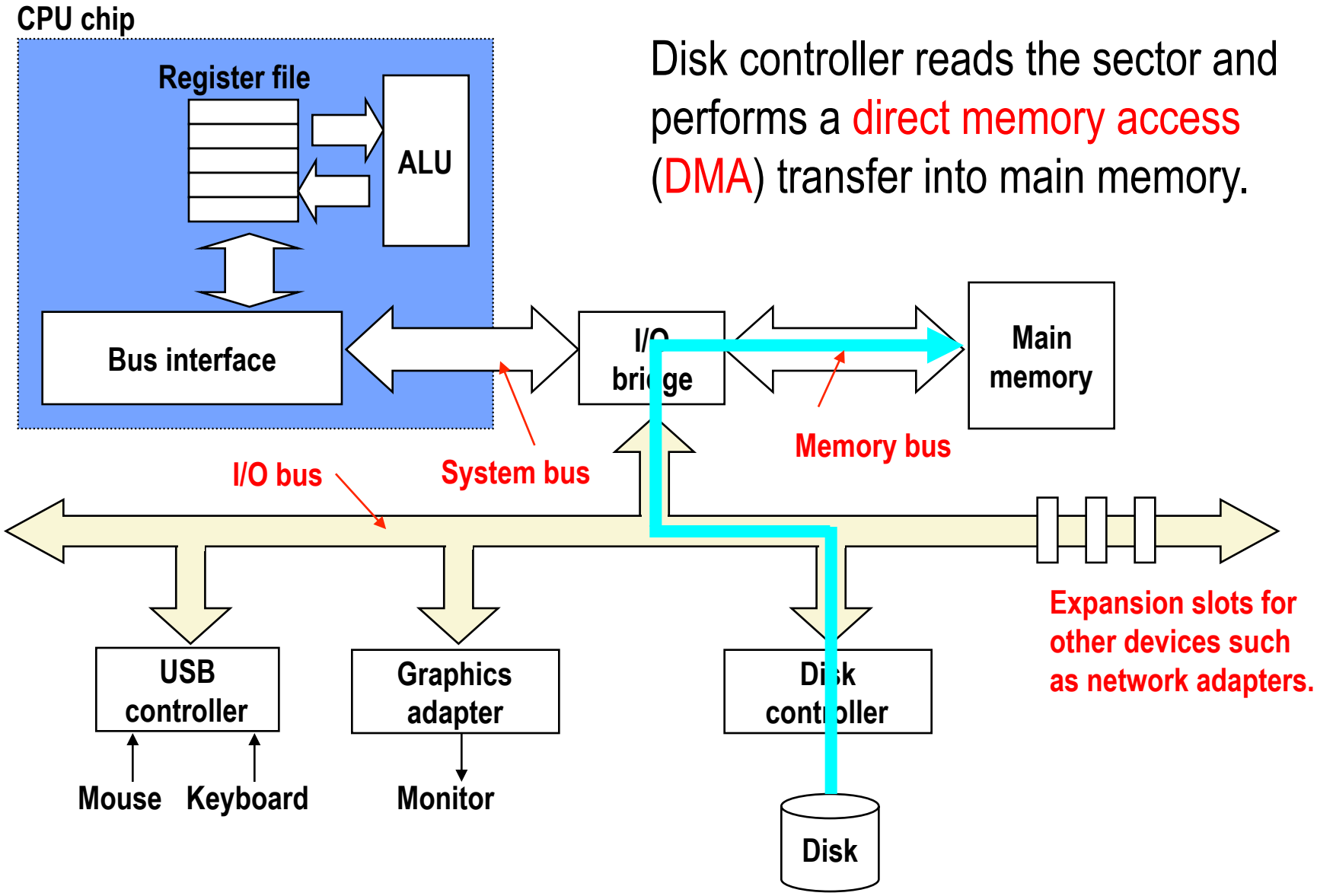

CPU chip
Register file
ALU
Bus interface
I/O bridge
Main memory
Memory bus
I/O bus
System bus
Expansion slots for other devices such as network adapters.
USB controller
Graphics adapter
Disk controller
Mouse
Keyboard
Monitor
Disk

Reading a Disk Sector (1)



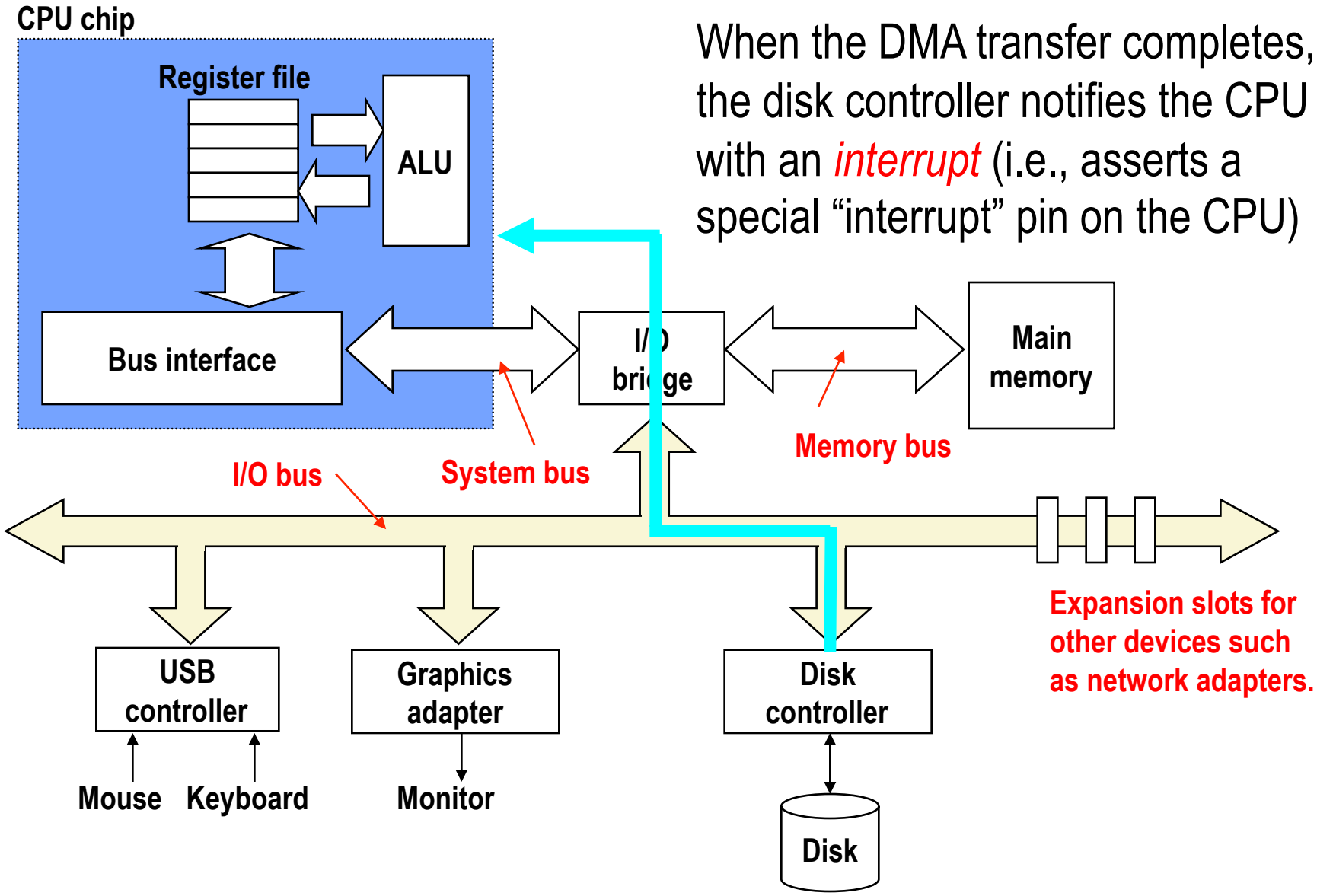
CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

Reading a Disk Sector (2)



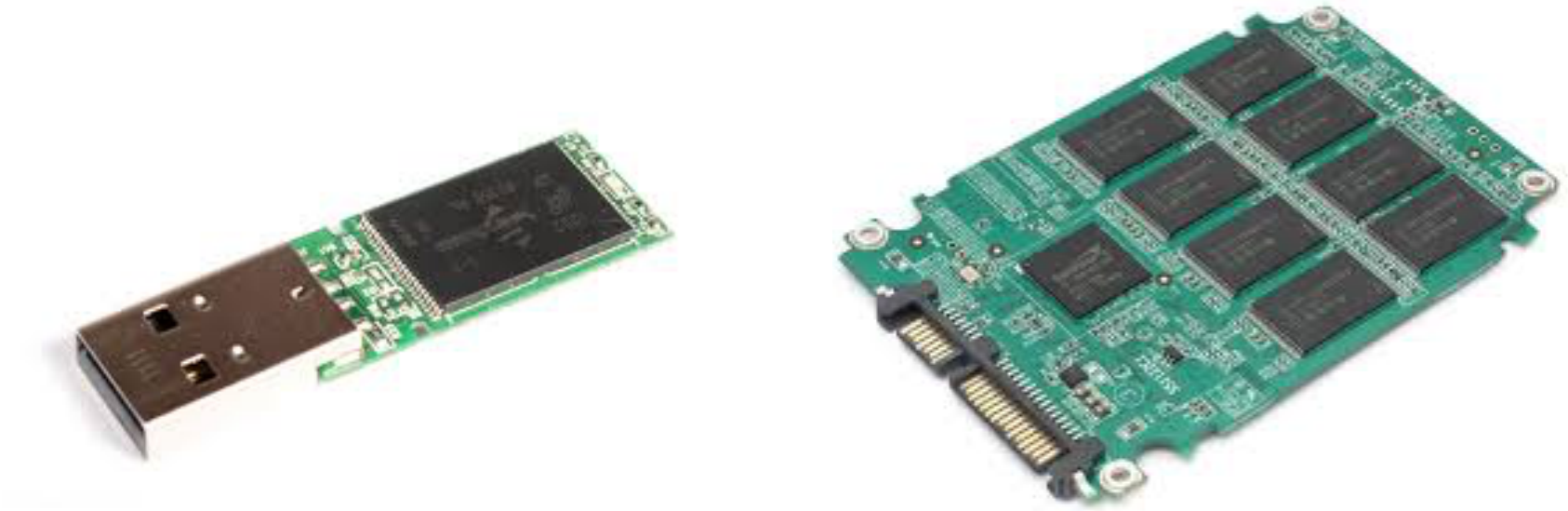
Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

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)

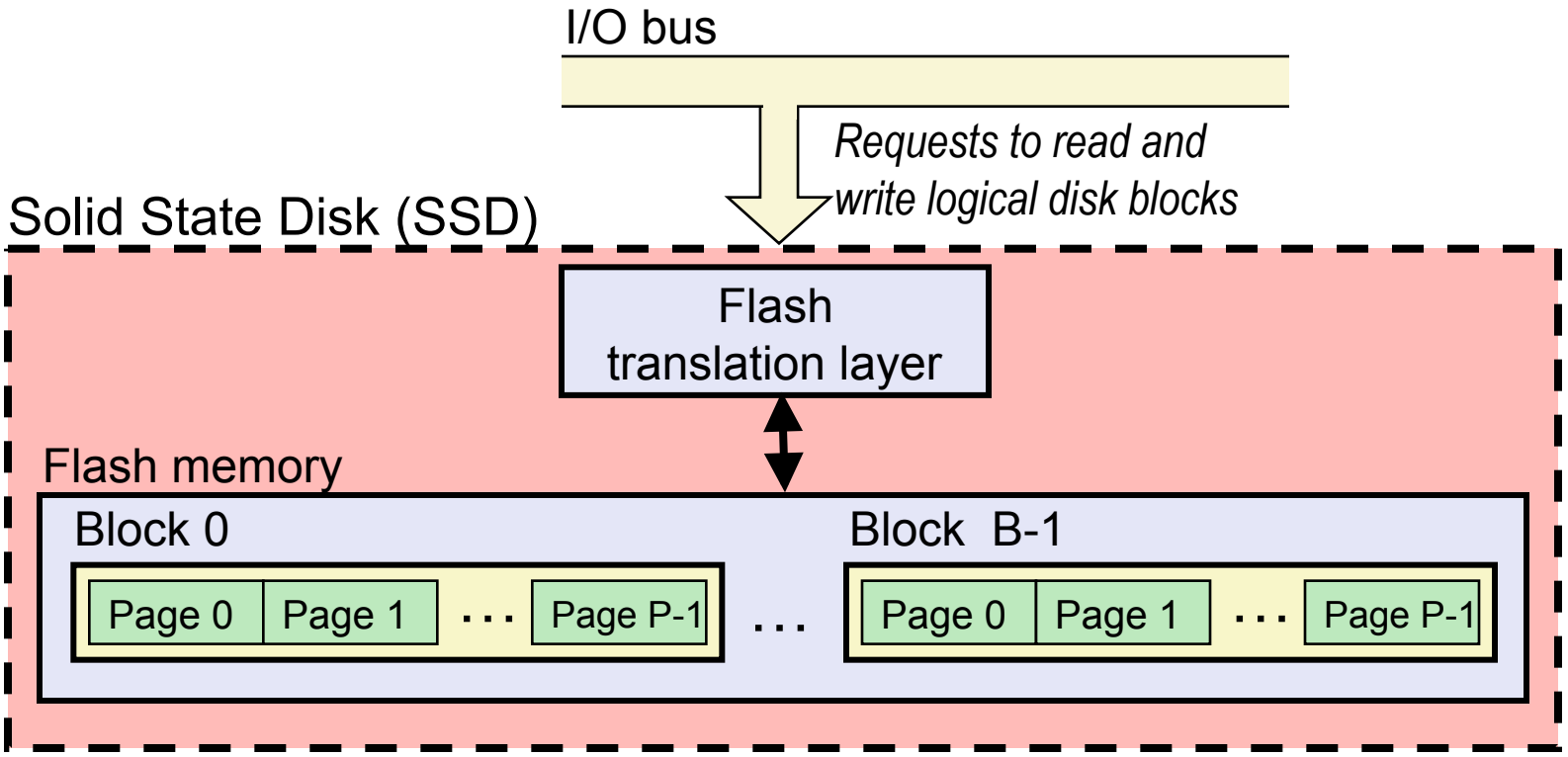
Flash Memory: Solid State Disk (SSD)



Non-volatile: Retains data without power

- **Fast (like DRAM)**
- **More rugged than disks.**
- **More expensive than disks.**

Solid State Disks (SSDs)



Pages: 512KB to 4KB, Blocks: 32 to 128 pages

- **Data is read/written in units of pages.**
- **Page can only be written after its block has been erased**
- **A block wears out after 100,000 repeated writes.**

SSD Performance Characteristics

Sequential read thruput	550 MB/s	Reading
Random read thruput	365 MB/s	
Avg sequential read time	50 us	

Sequential write thruput	470 MB/s	Writing
Random write thruput	303 MB/s	
Avg sequential write time	60 us	

Sequential access is faster than random access.

Random writes are slow.

- Erasing a block takes a long time (~1 ms)
- Modifying a block page requires all other pages to be copied to new block

Source: Intel SSD 730 product specification.

SSD Tradeoffs vs Rotating Disks

Advantages to SSD

No moving parts → faster, less power, more rugged

Disadvantages

Have the potential to wear out

Mitigated by "wear leveling logic" in flash translation layer

- Intel SSD 730 guarantees 128 petabyte (128 x 10^{15} bytes) of writes before they wear out

In 2015, about 6 times more expensive per byte *

** Best Buy: $13/32GB flash, $60/1TB External Disk*

Applications

Phones, laptops, portable electronics

Beginning to appear in desktops and servers

Storage Trends

SRAM

Metric	1985	1990	1995	2000	2005	2010	2015	2015:1985
$/MB	2,900	320	256	100	75	60	320	116
access (ns)	150	35	15	3	2	1.5	200	115

DRAM

Metric	1985	1990	1995	2000	2005	2010	2015	2015:1985
$/MB	880	100	30	1	0.1	0.06	0.02	44,000
access (ns)	200	100	70	60	50	40	20	10
typical size (MB)	0.256	4	16	64	2,000	8,000	16.000	62,500

Disk

Metric	1985	1990	1995	2000	2005	2010	2015	2015:1985
$/GB	100,000	8,000	300	10	5	0.3	0.03	3,333,333
access (ms)	75	28	10	8	5	3	3	25
typical size (GB)	0.01	0.16	1	20	160	1,500	3,000	300,000

CPU Clock Rates

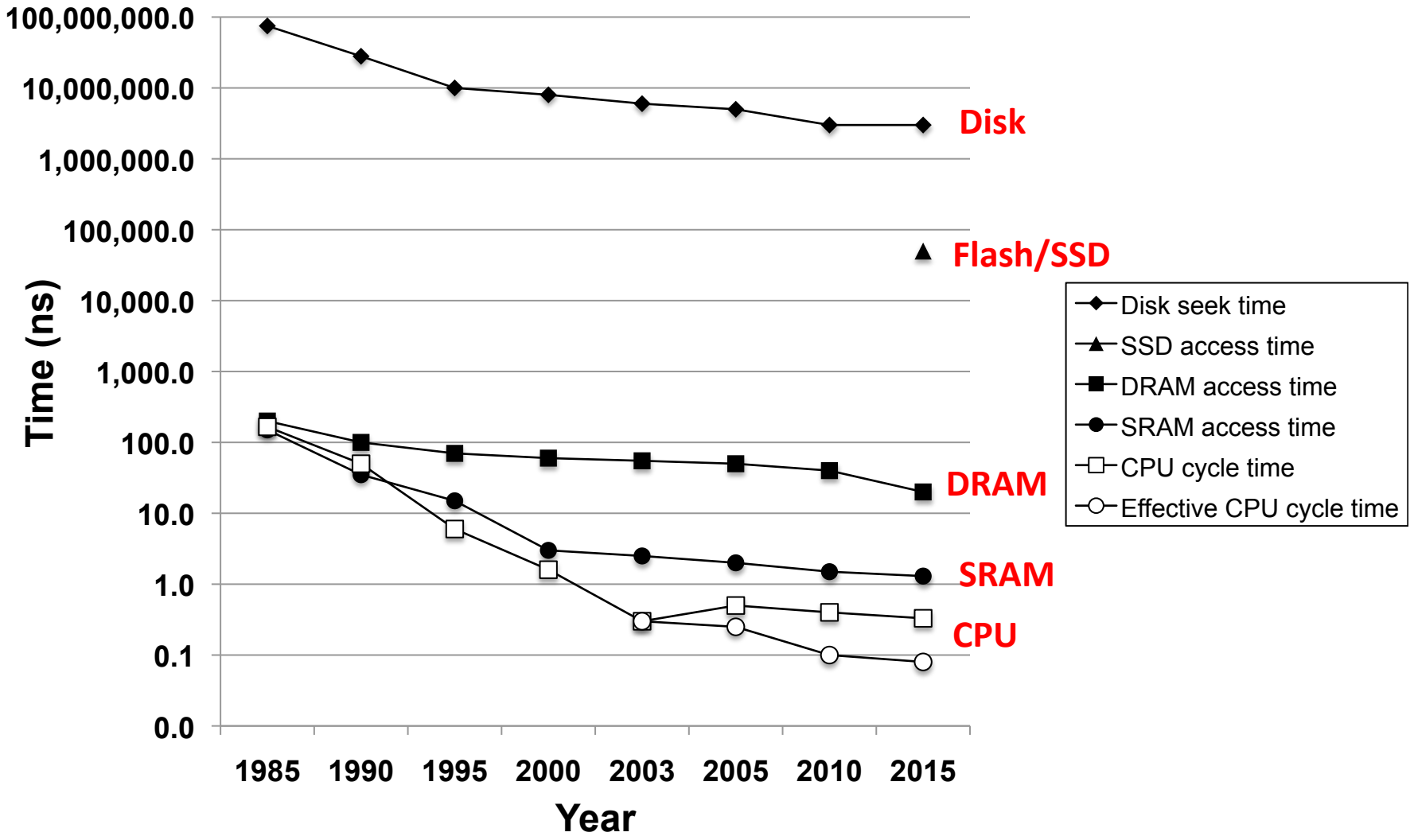
Inflection point in computer history when designers hit the "Power Wall"

	1985	1990	1995	2003	2005	2010	2015	*2015:1985*
CPU	80286	80386	Pentium	P-4	Core 2	Core i7(n)	Core i7(h)	
Clock rate (MHz)	6	20	150	3,300	2,000	2,500	3,000	500
Cycle time (ns)	166	50	6	0.30	0.50	0.4	0.33	500
Cores	1	1	1	1	2	4	4	4
Effective cycle time (ns)	166	50	6	0.30	0.25	0.10	0.08	2,075

(n) Nehalem processor
(h) Haswell processor

The CPU-Memory Gap

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

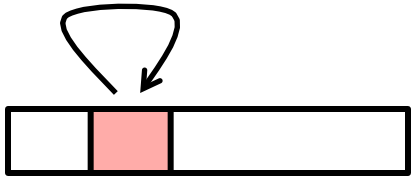


Locality to the Rescue!

The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as locality

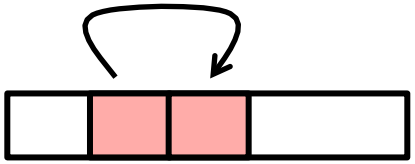
Locality

Principle of Locality: **Programs tend to use data and instructions with addresses near or equal to those they have used recently**



Temporal locality:

Recently referenced items are likely
to be referenced again 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 references

- Reference array elements in succession (stride-1 reference pattern). — **Spatial locality**
- Reference variable **sum** each iteration. — **Temporal locality**

Instruction references

- Reference instructions in sequence. — **Spatial locality**
- Cycle through loop repeatedly. — **Temporal locality**

Qualitative Estimates of Locality

Claim: Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.

Question: Does this function have good locality with respect to array `a`?

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

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

Qualitative Estimates of Locality

Claim: Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.

Question: Does this function have good locality with respect to array `a`?

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

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

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 sum_array_3d(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

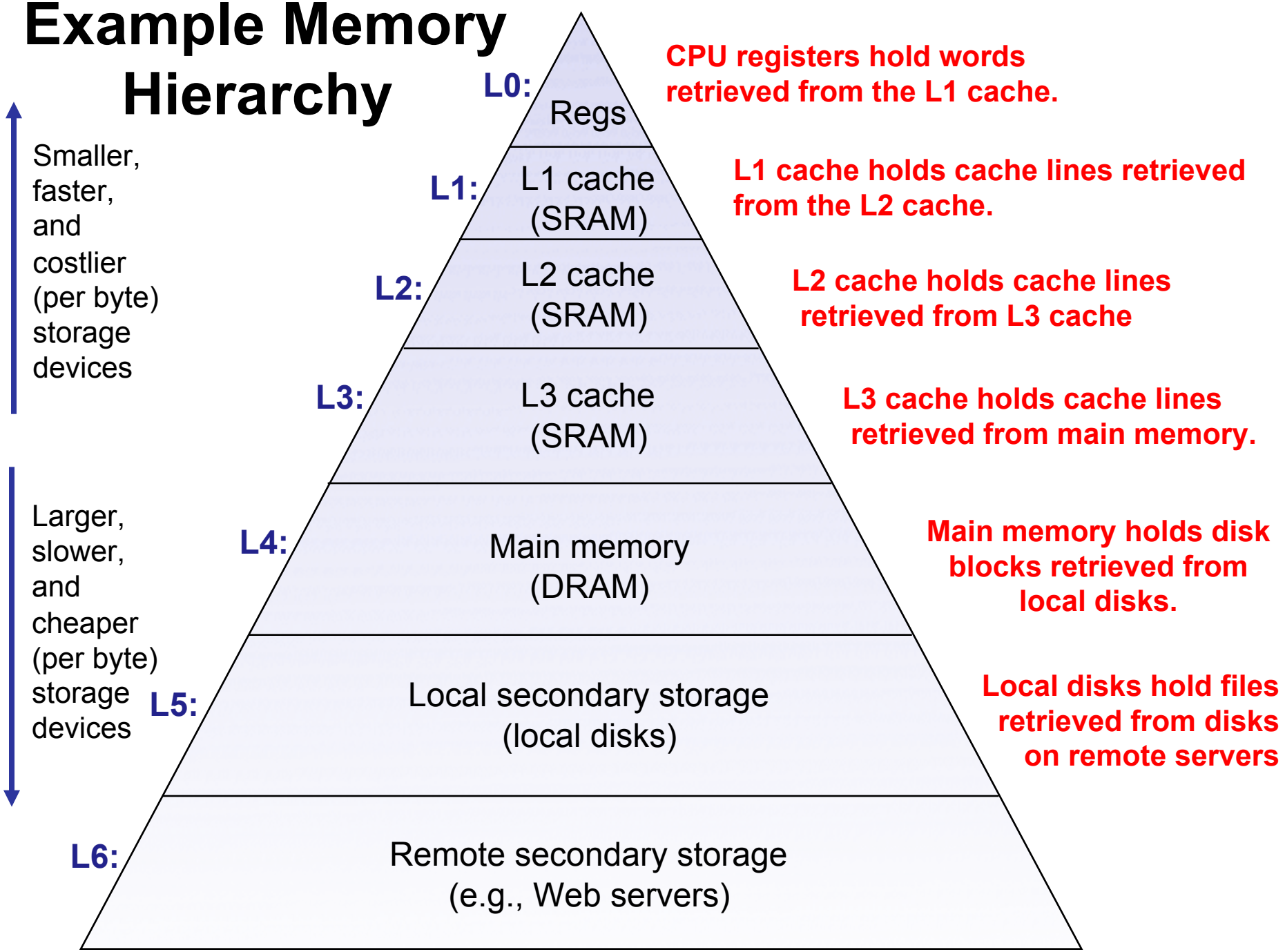
Some fundamental and enduring properties of hardware and software:

- Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
- 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.

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:

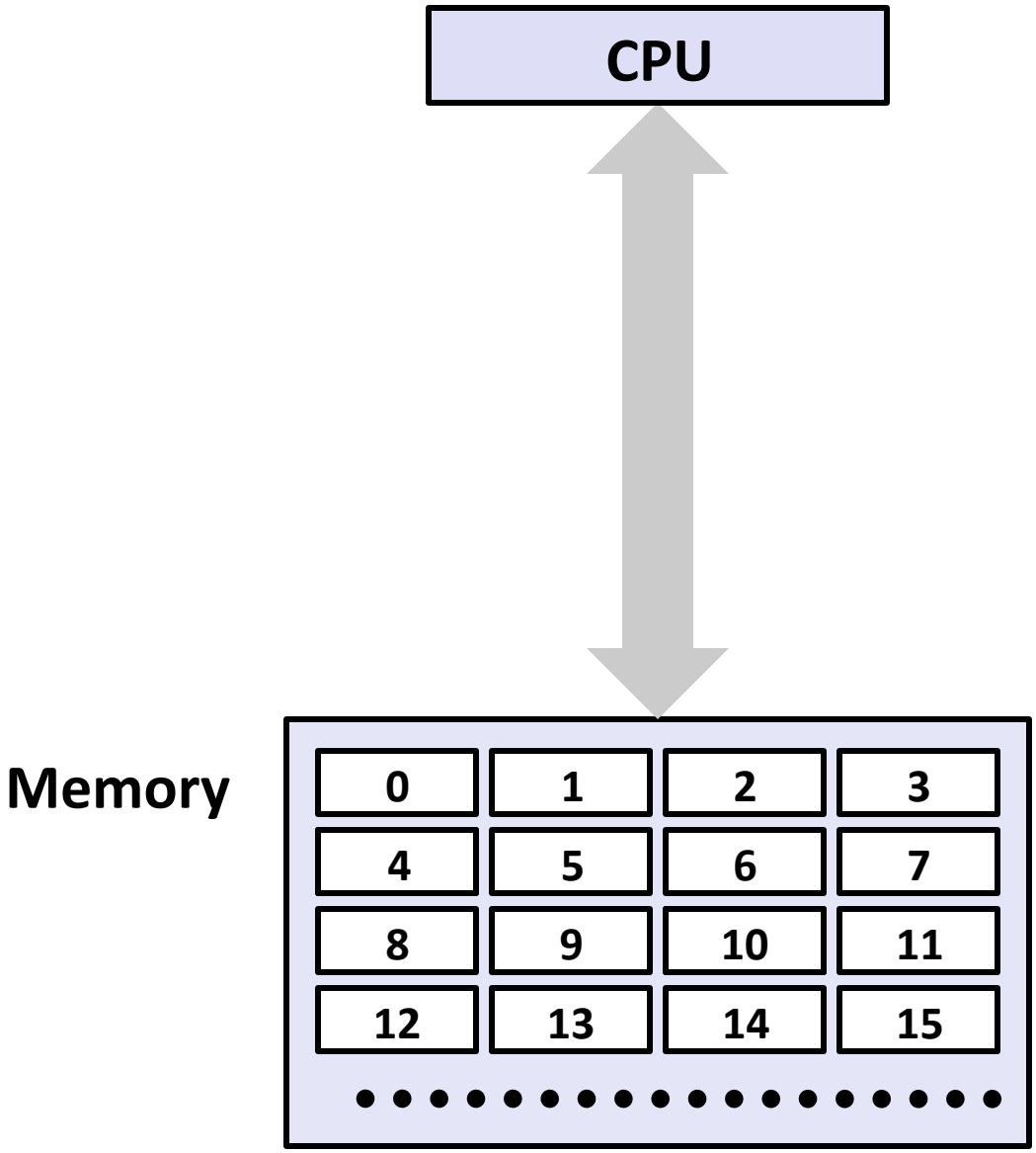
- For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.

Why do memory hierarchies work?

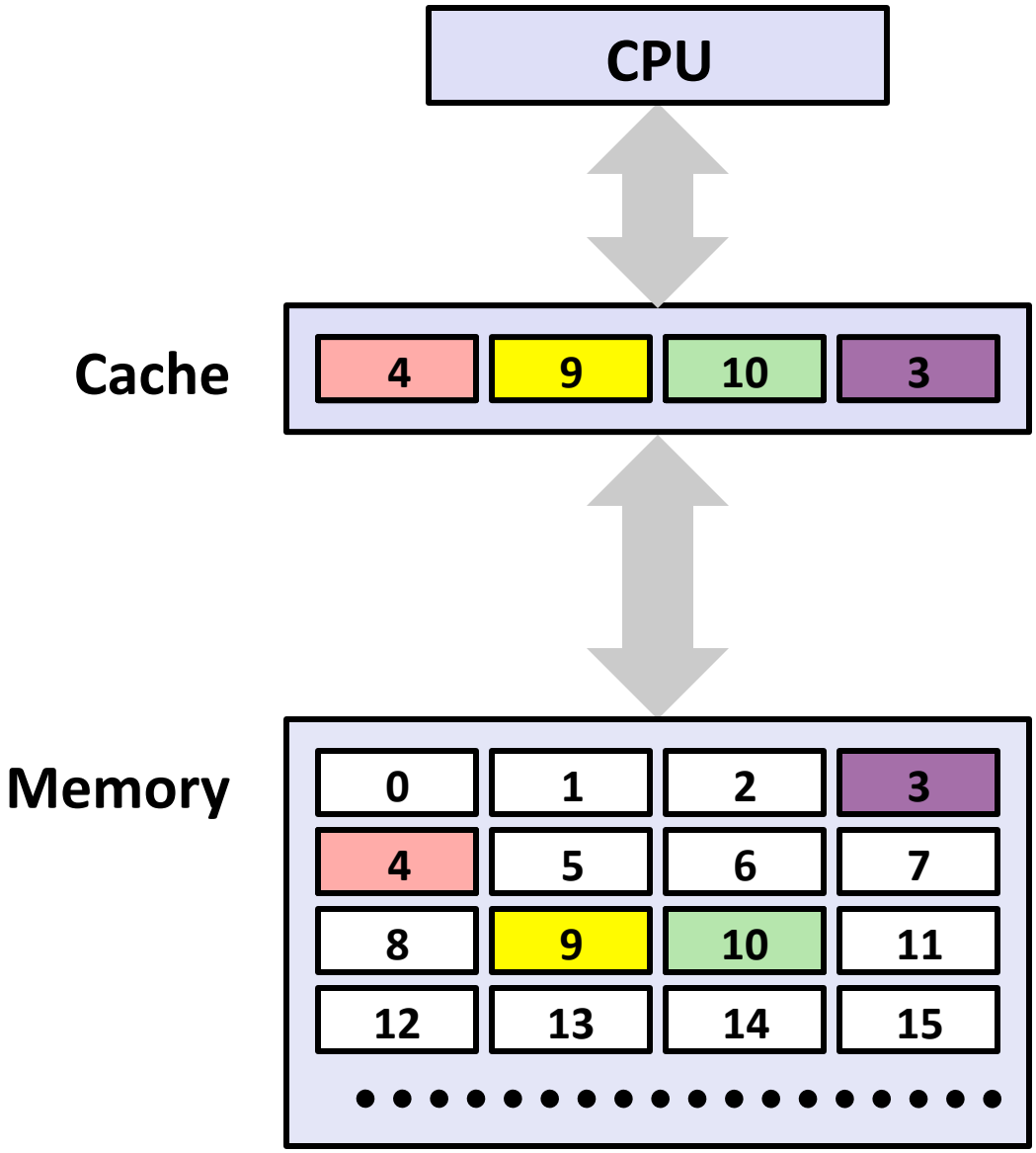
- Because of locality, programs tend to access the data at level k more often than they access the data at level k+1.
- Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.

Big Idea: **The memory hierarchy creates a large pool of storage 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.**

General Cache Concepts



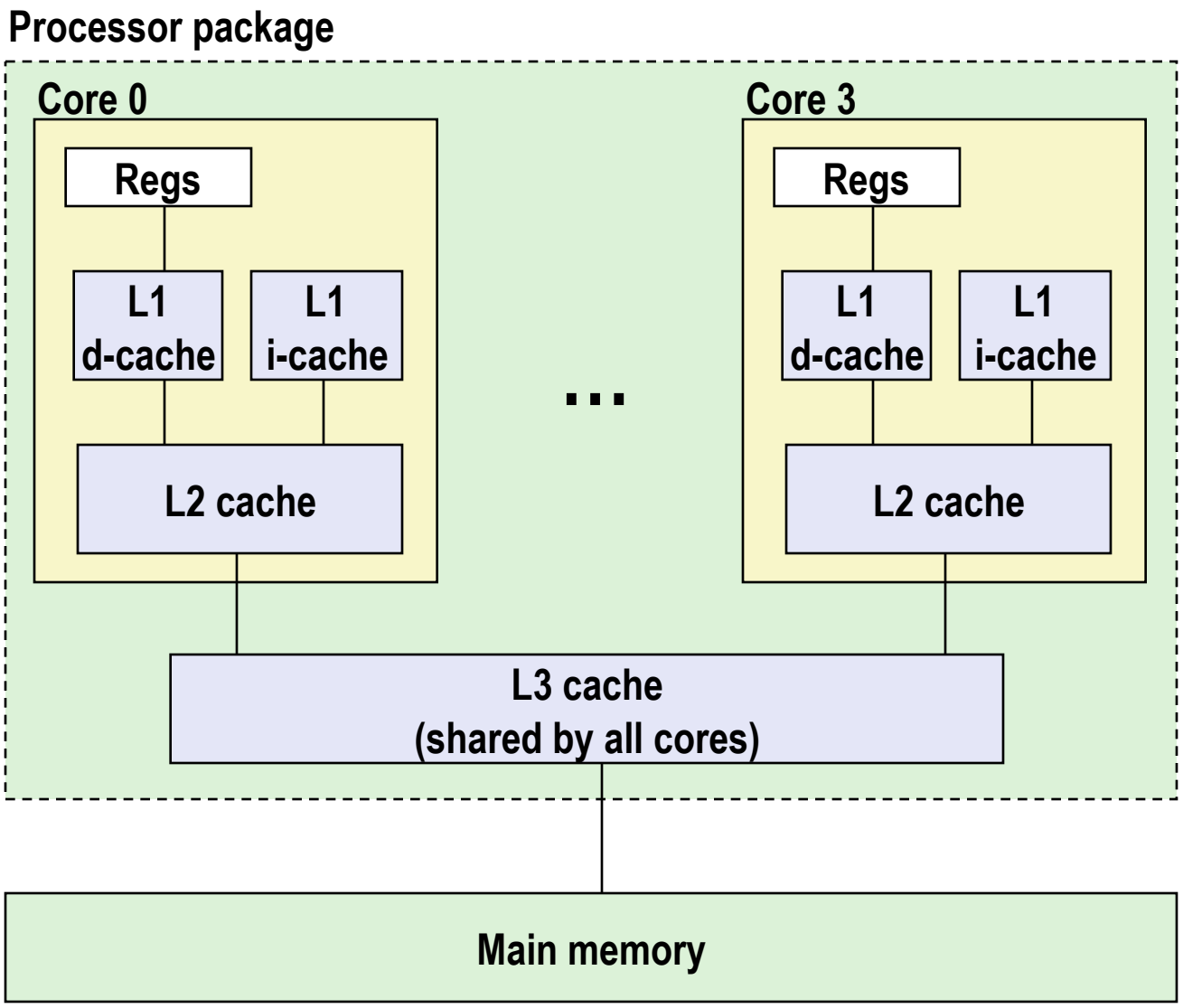
General Cache Concepts



Smaller, faster, more expensive memory caches a subset of the blocks

Larger, slower, cheaper memory viewed as partitioned into “blocks”

Intel Core i7 Cache Hierarchy



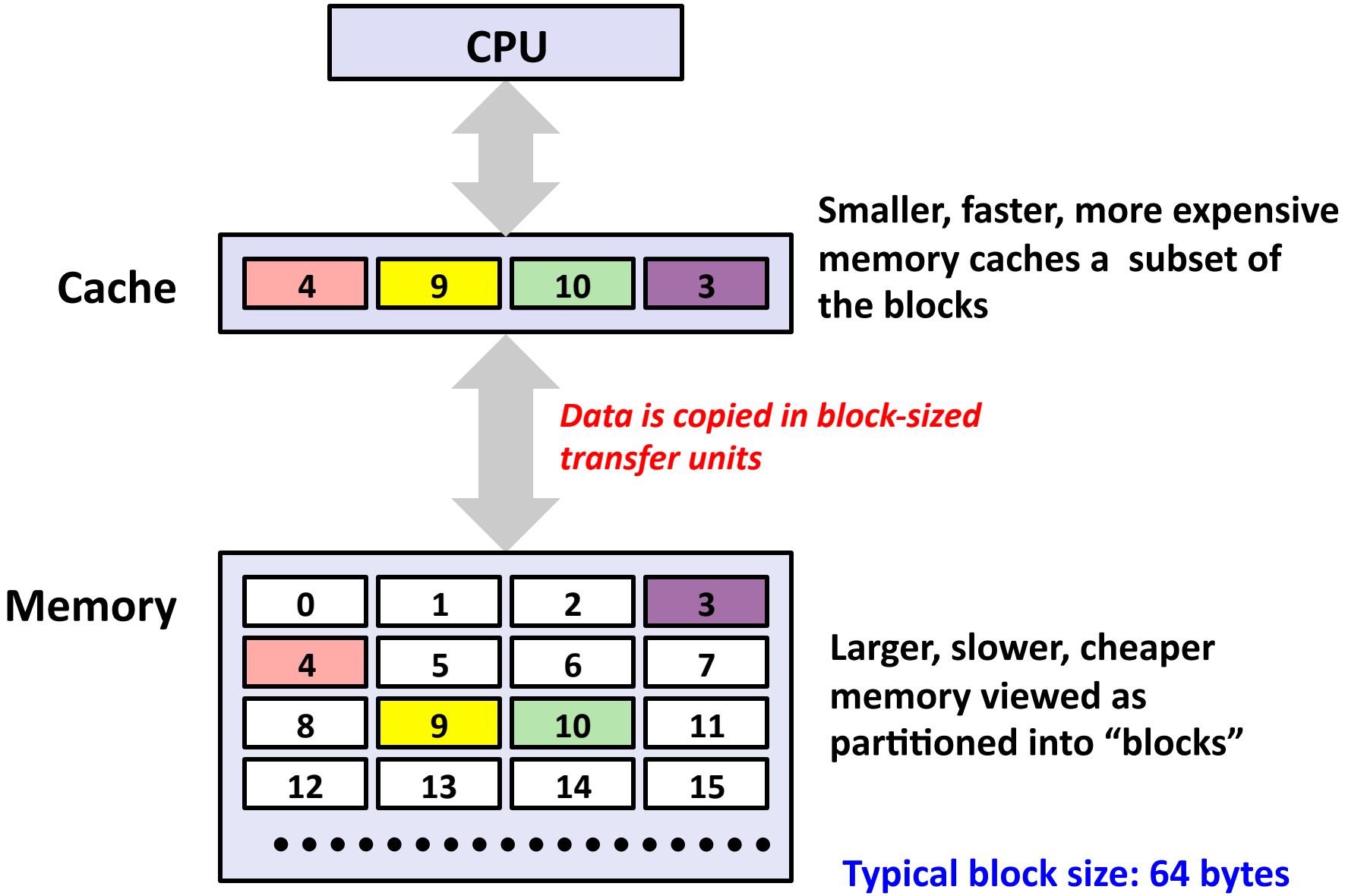
L1 i-cache and d-cache:
32 KB
Access: 4 cycles

L2 cache:
256 KB
Access: 10 cycles

L3 cache:
8 MB
Access: 40-75 cycles

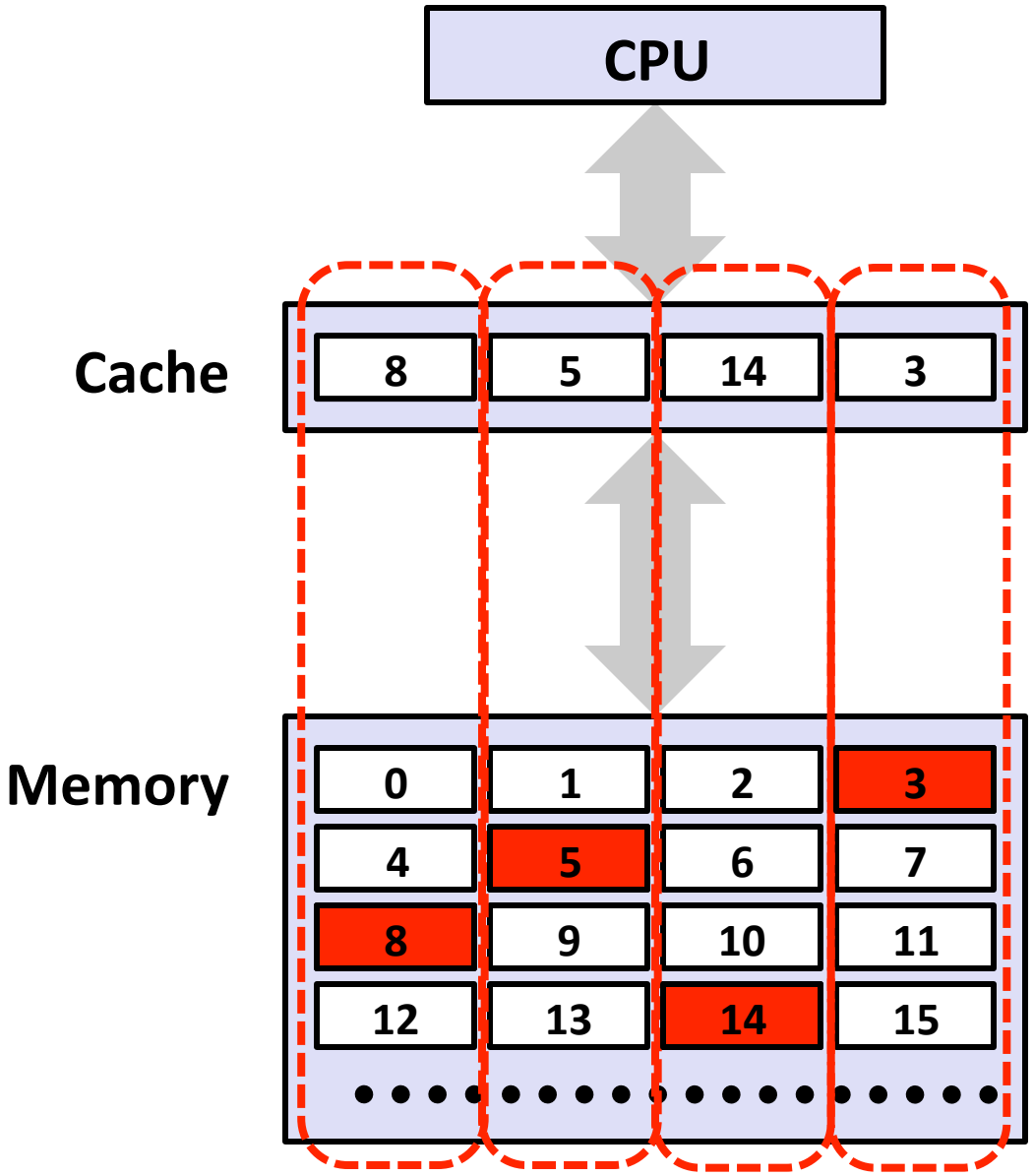
Block size: 64 bytes for all caches.

General Cache Concepts



General Cache Concepts: Placement

animation

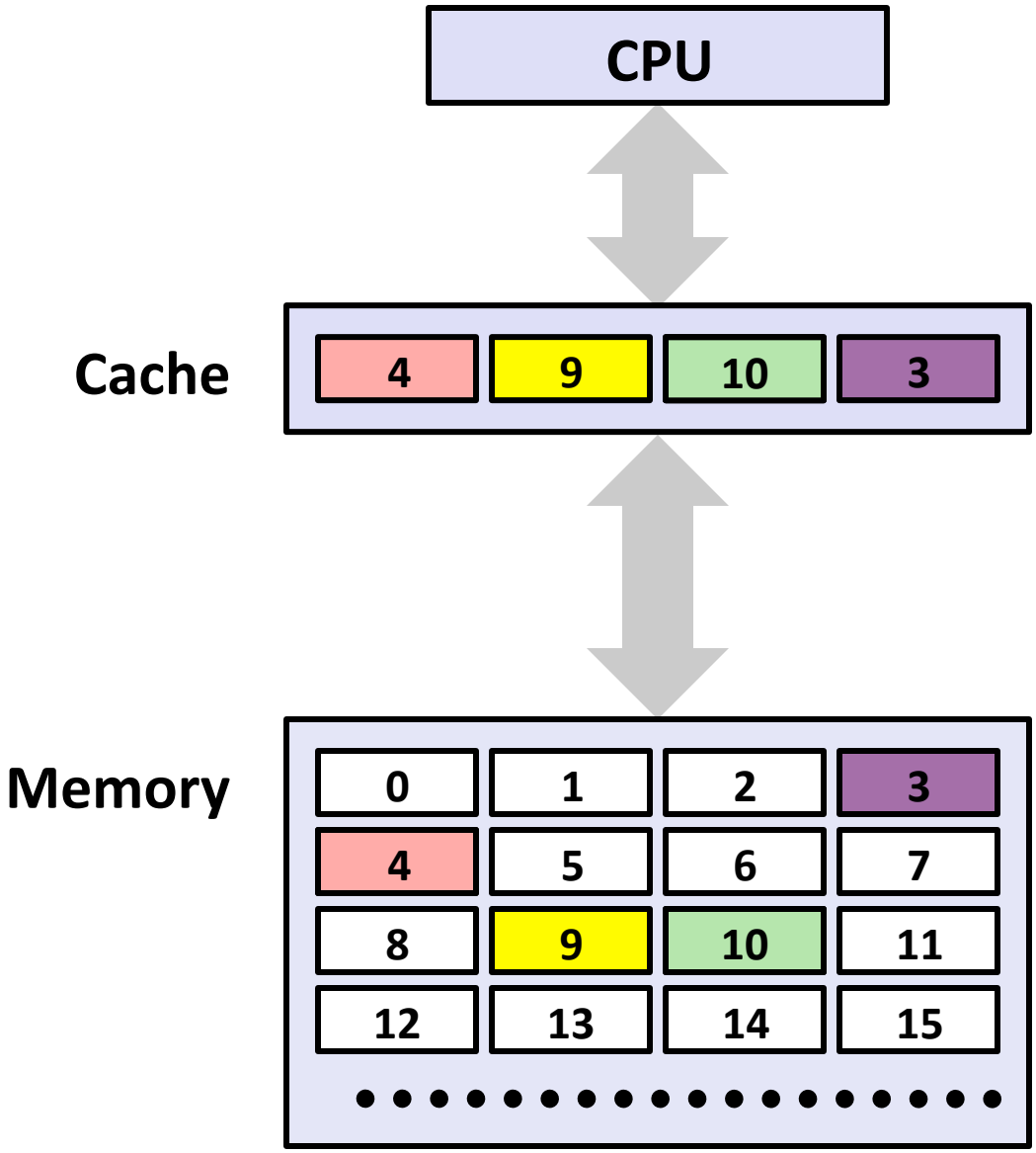


The cache has a limited size.

There are constraints on where data can be placed.

We cannot put any block any place!

General Cache Concepts: Hit

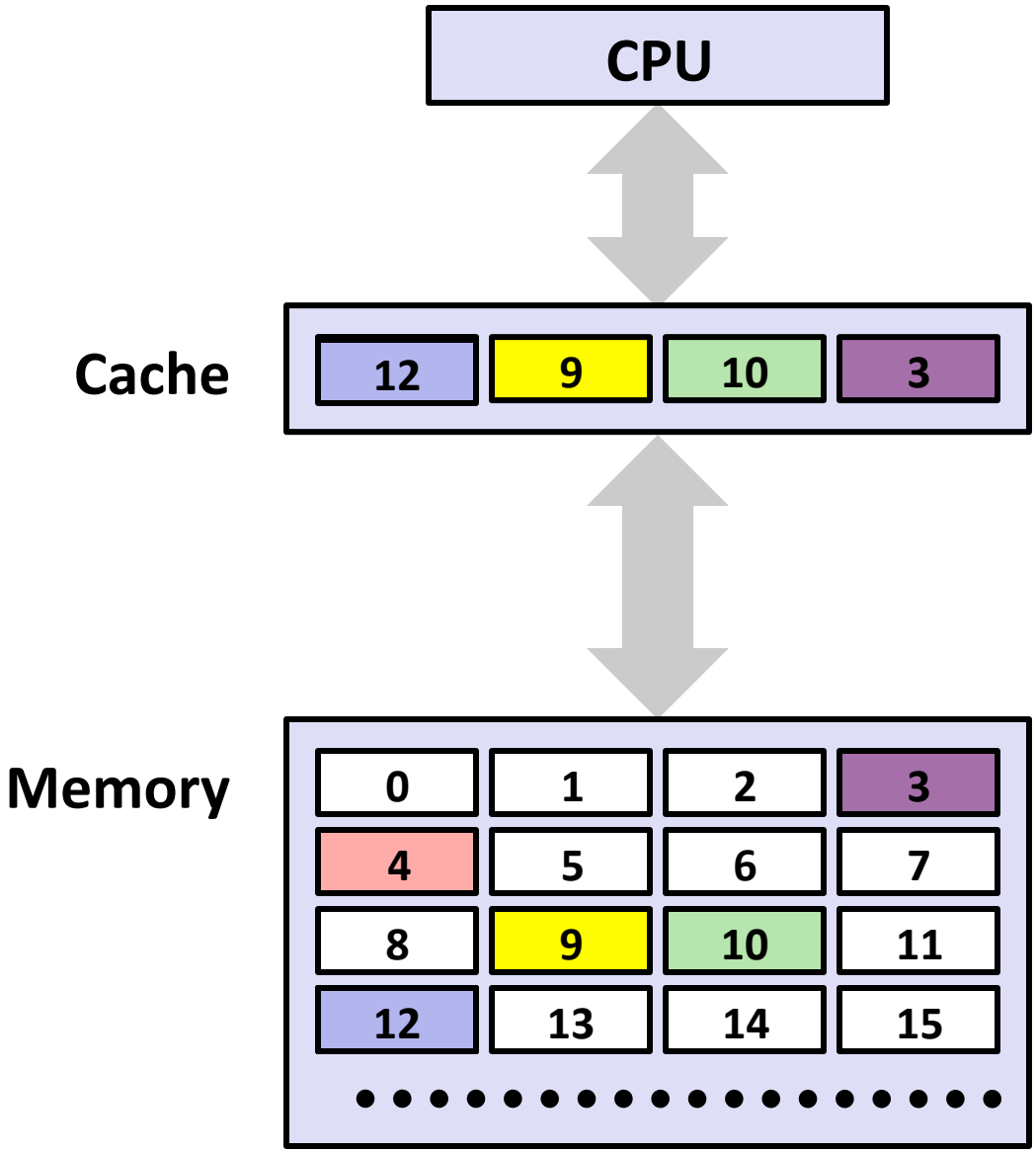


Data in block b is needed

Block b is in cache:

Hit!

General Cache Concepts: Miss



Data in block b is needed

Block b is NOT in cache!

Block is fetched from memory

Block is stored in cache.

Request is satisfied.

Block now available for future accesses.

Questions

A block of data (b) is loaded into cache…

Where do you put it?

Placement policy:

Where can you place b in the cache?

Not every block can go into every place in cache.

Why? Efficiency.

Where will you place b?

Some other block will get “evicted”.

“Replacement policy”

General Caching Concepts: Types of Cache Misses

Cold (compulsory) miss

Cold misses occur because the cache is empty.

Conflict miss

Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.

- E.g. Block i at level k+1 must be placed in block (i mod 4) at level k.

Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block.

- E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.

Capacity miss

Occurs when the set of active cache blocks (working set) is larger than the cache.

Examples of Caching in the Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 bytes words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware
L1 cache	64-bytes block	On-Chip L1	1	Hardware
L2 cache	64-bytes block	On/Off-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Disk firmware
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

Summary

- The speed gap between CPU, memory and mass storage continues to widen.
- Well-written programs exhibit a property called locality.
- Memory hierarchies based on caching close the gap by exploiting locality.