

# IF2130 – Organisasi dan Arsitektur Komputer

sumber: Greg Kesden, CMU 15-213, 2012

Achmad Imam Kistijantoro ([imam@informatika.org](mailto:imam@informatika.org))

Storage

Rahmat Mulyawan

Infall Syafalni

# Today

---

- ▶ **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

---

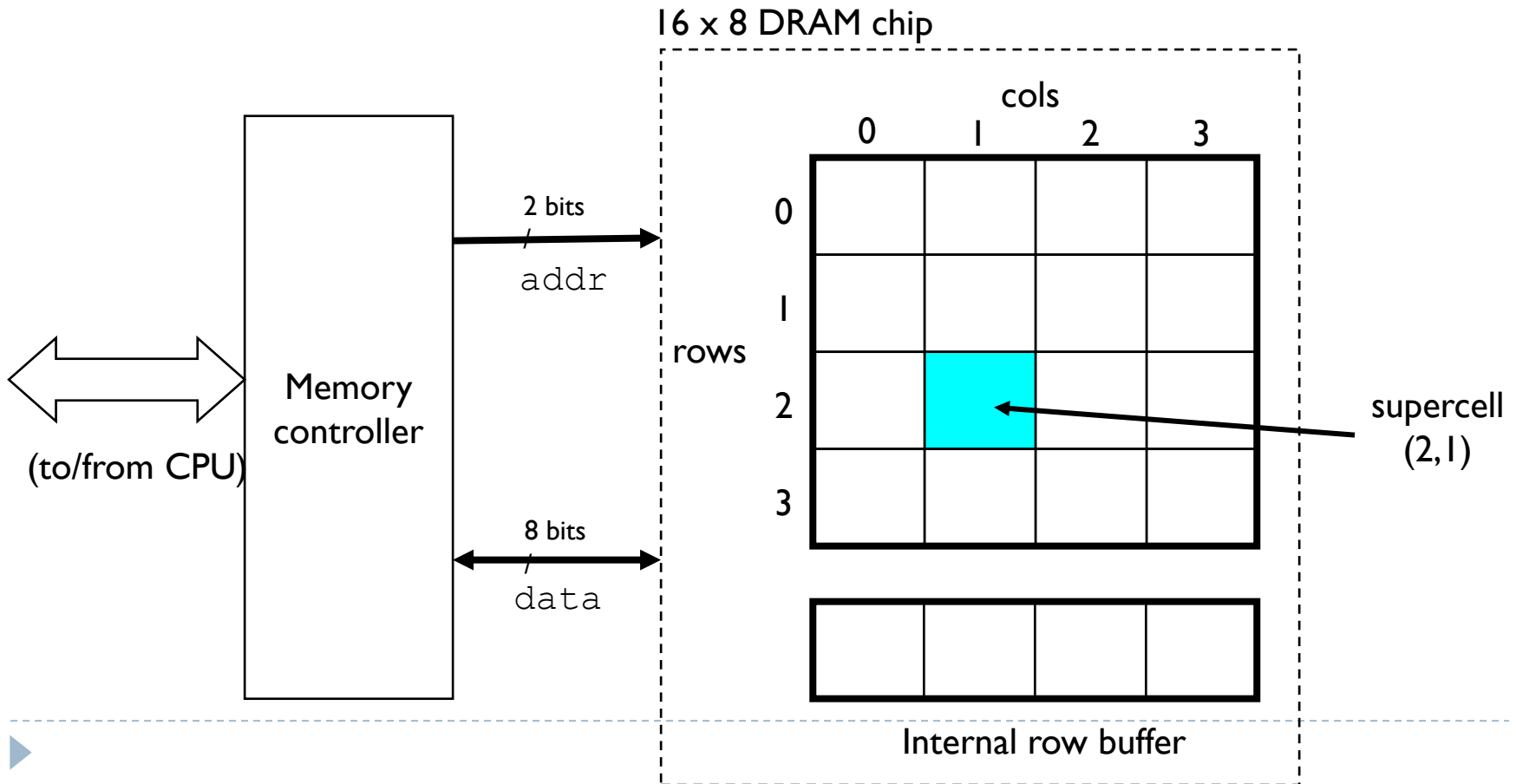
	Trans. per bit	Access time	Needs refresh?	Needs EDC?	Cost	Applications
SRAM	4 or 6	1X	No	Maybe	100x	Cache memories
DRAM	1	10X	Yes	Yes	1X	Main memories, frame buffers



# Conventional DRAM Organization

- ▶  $d \times w$  DRAM:

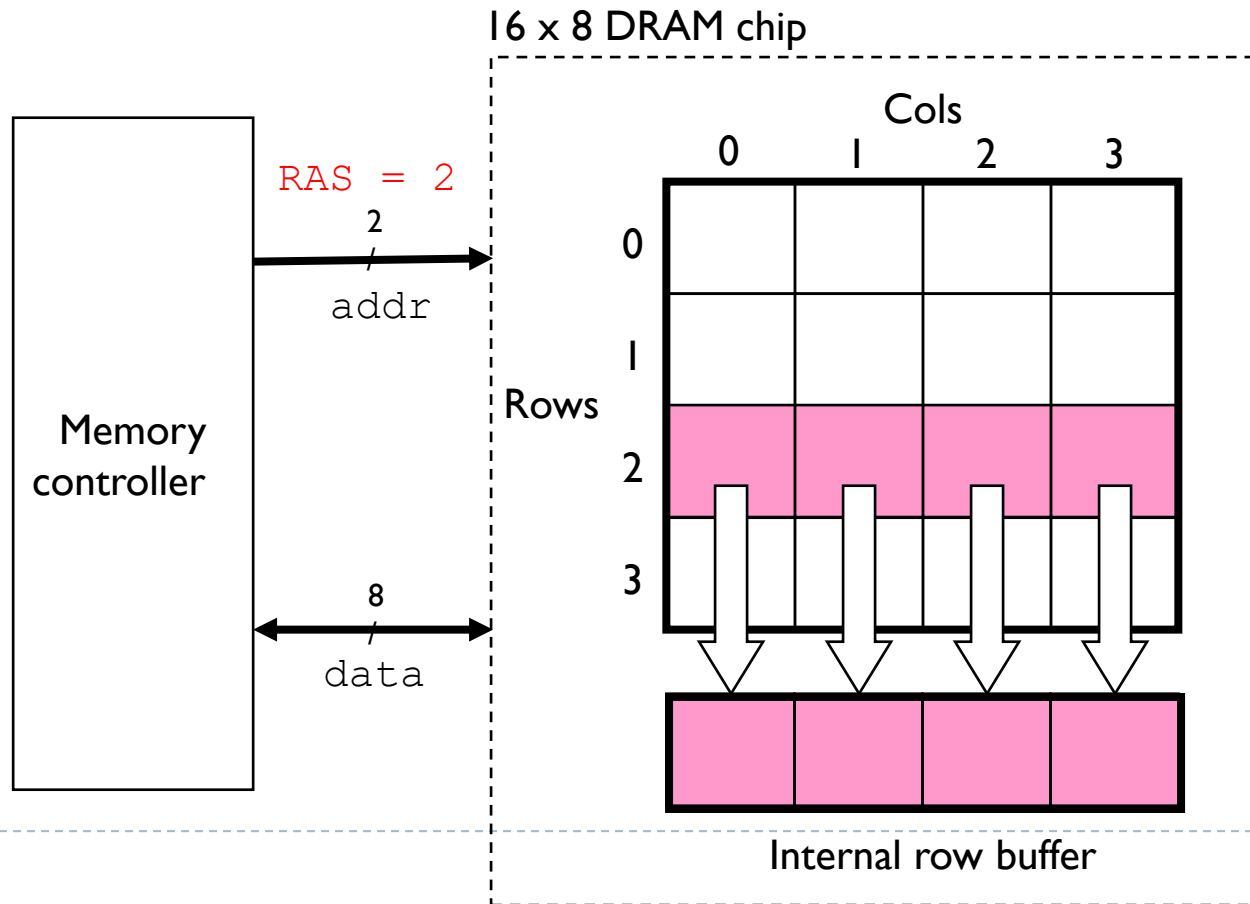
- ▶  $dw$  total bits organized as  $d$  **supercells** of size  $w$  bits



# Reading DRAM Supercell (2,1)

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

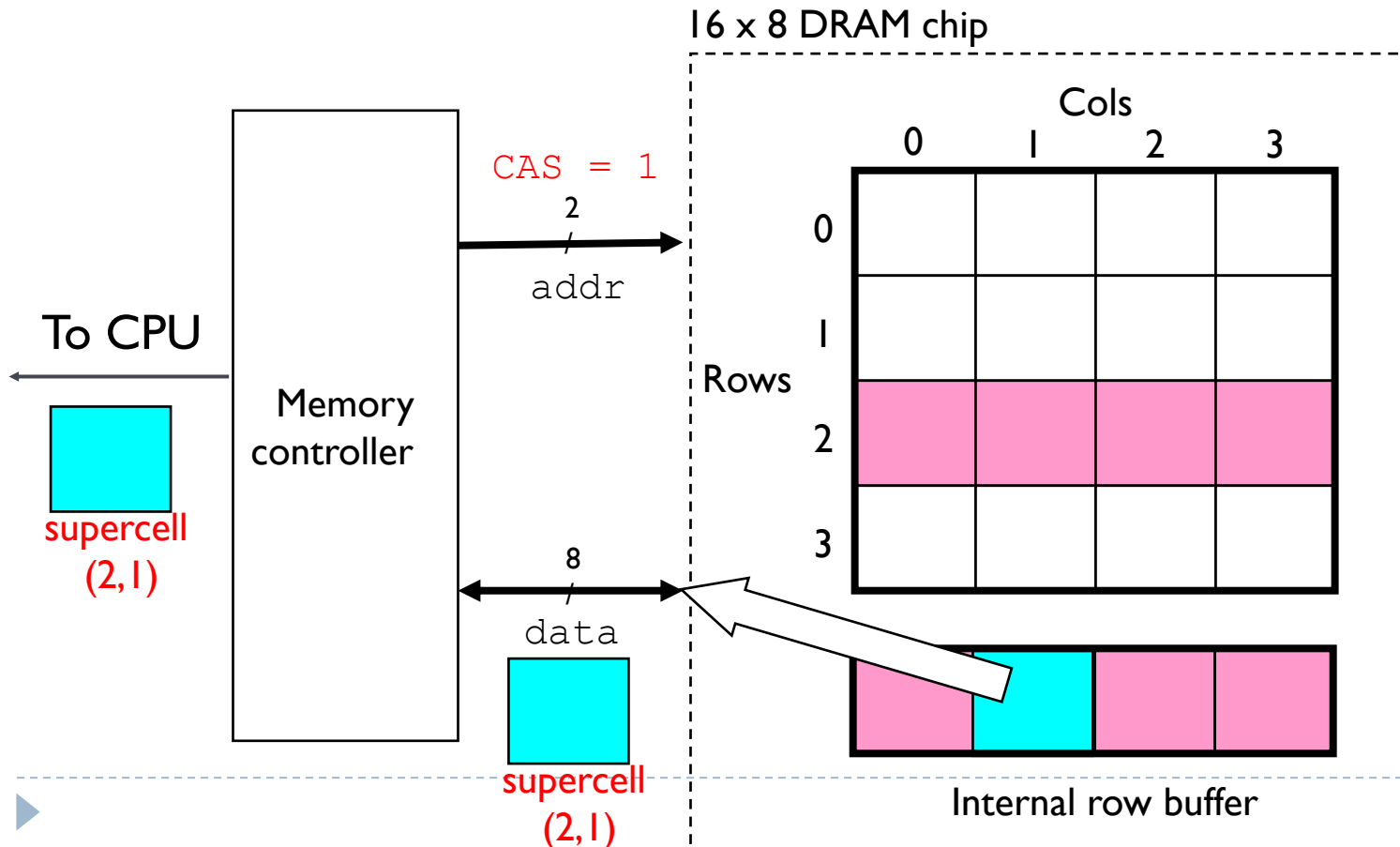
Step 1(b): Row 2 copied from DRAM array to row buffer.



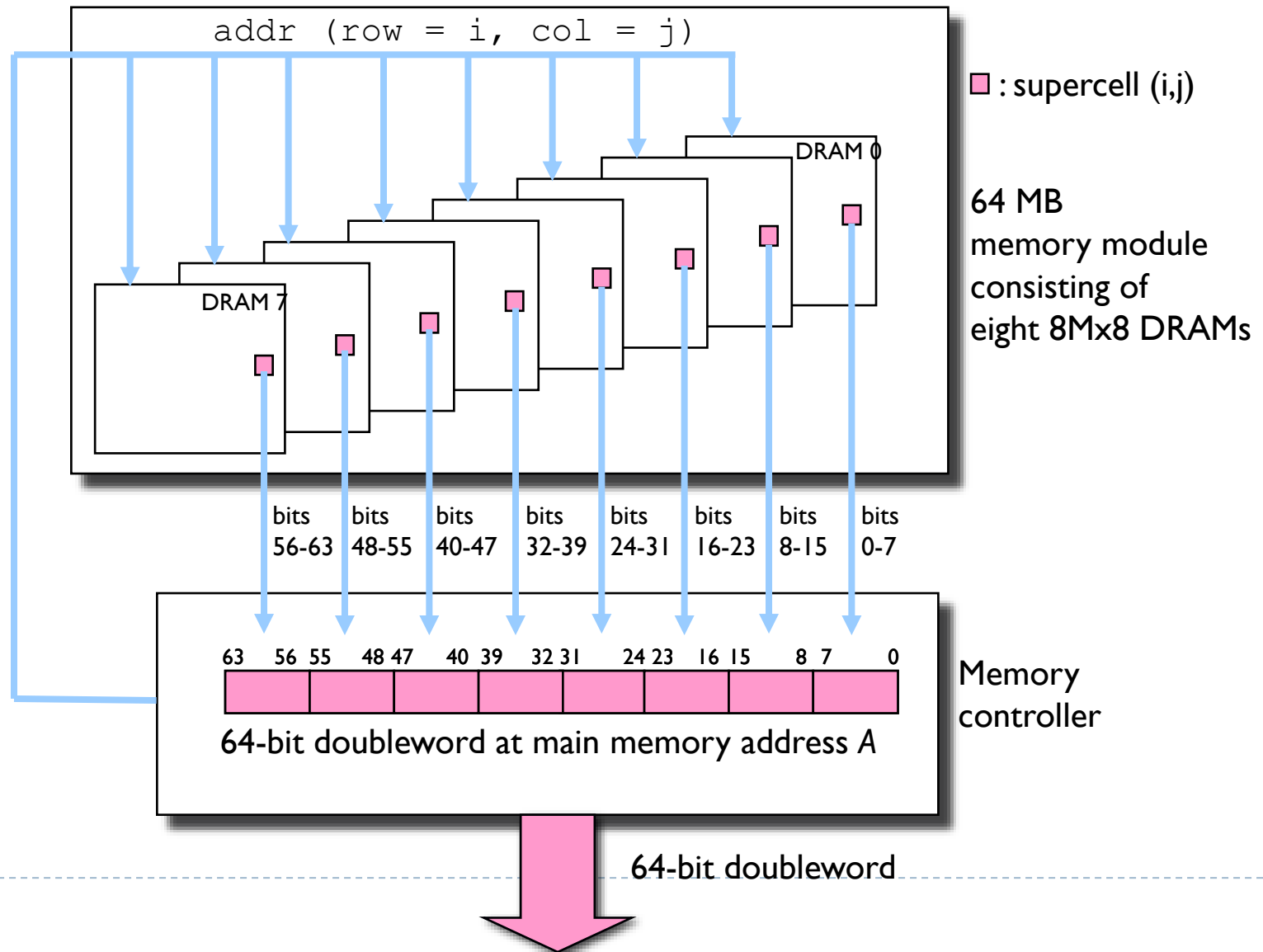
# Reading DRAM Supercell (2,1)

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.



# Memory Modules





# 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), **DDR3** (8 bits)
    - ▶ By 2010, standard for most server and desktop systems
    - ▶ Intel Core i7 supports only DDR3 SDRAM



# Nonvolatile Memories

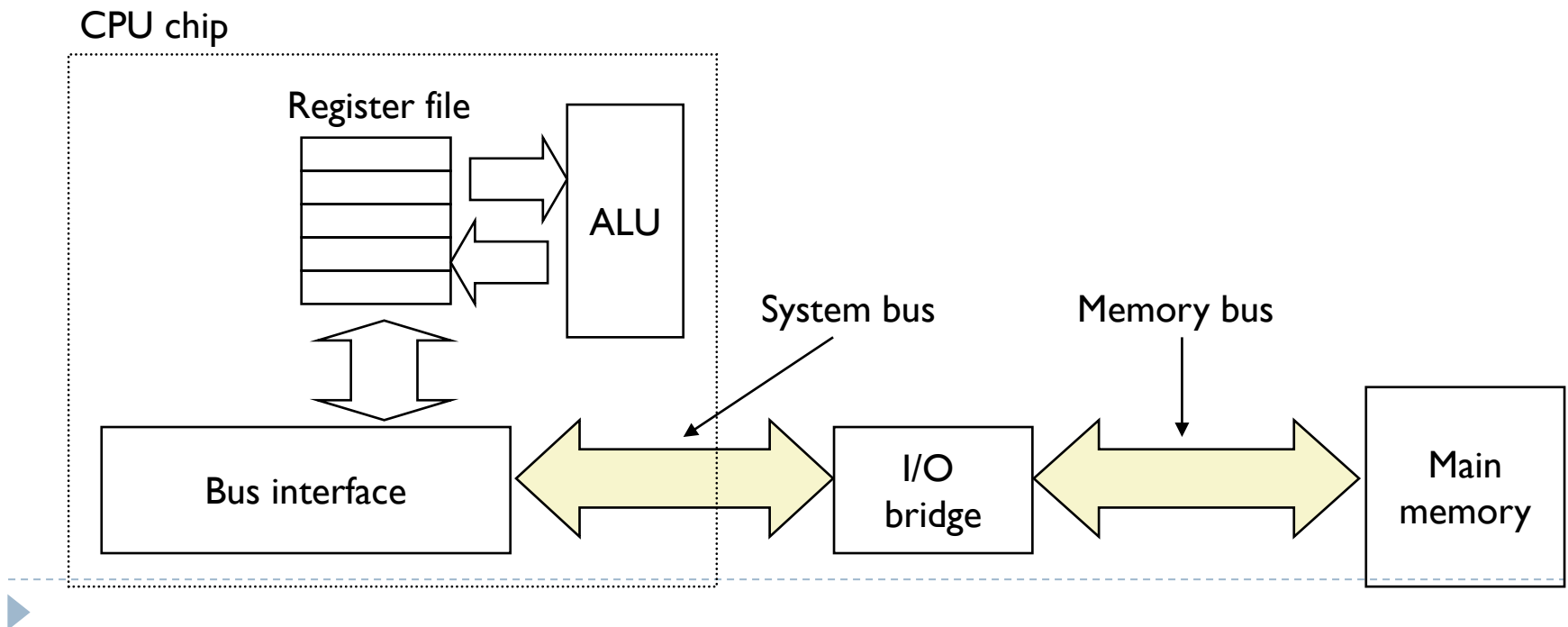
---

- ▶ 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
  - ▶ Erasable PROM (**EPROM**): can be bulk erased (UV, X-Ray)
  - ▶ Electrically erasable 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



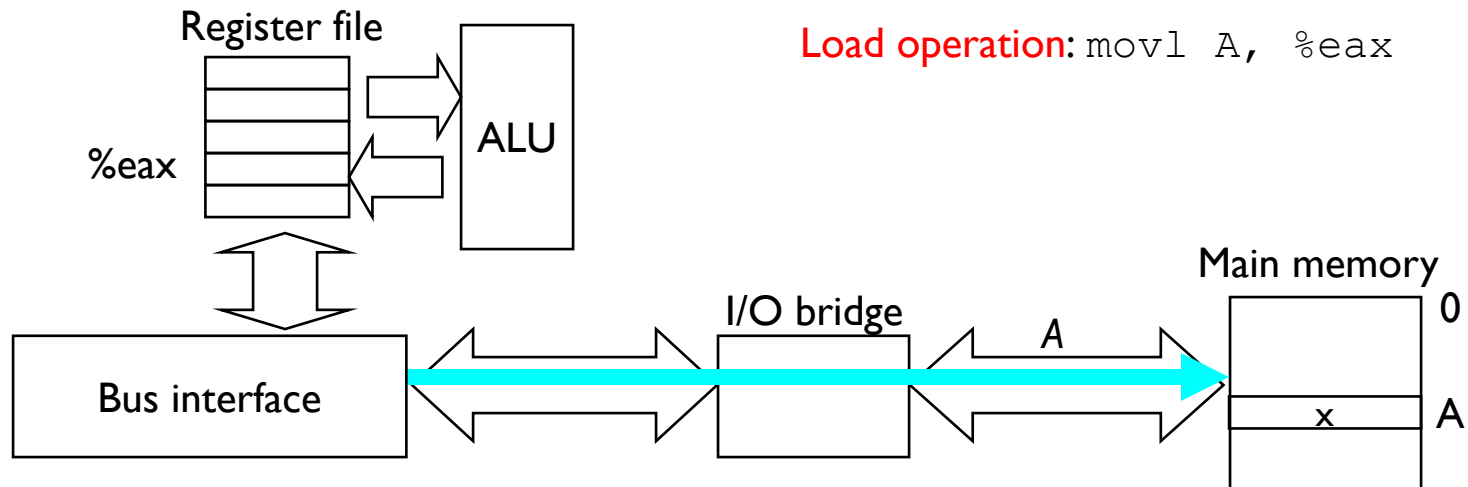
# Traditional Bus Structure Connecting CPU and Memory

- ▶ A **bus** is a collection of parallel wires that carry address, data, and control signals.
- ▶ Buses are typically shared by multiple devices.



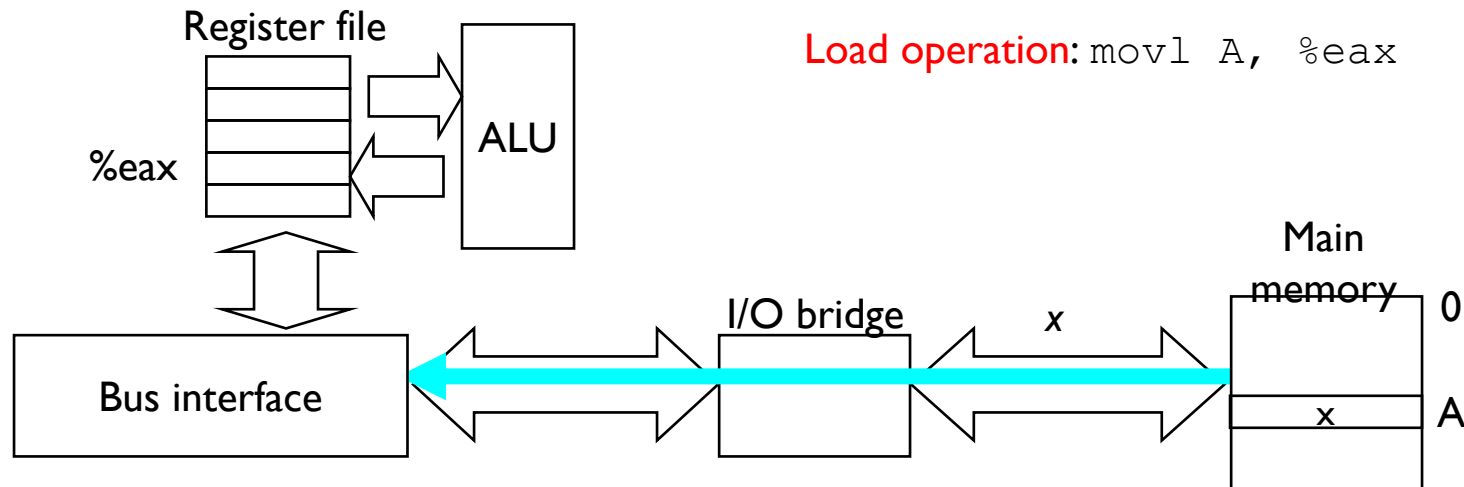
# Memory Read Transaction (1)

- ▶ CPU places address A on the memory bus.



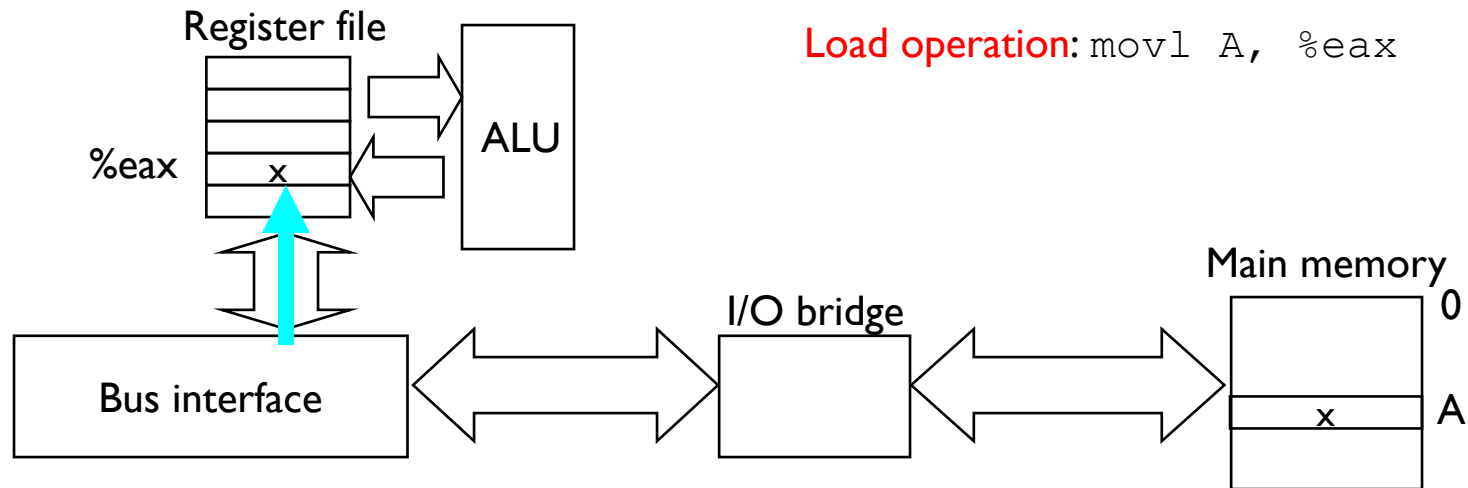
## Memory Read Transaction (2)

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



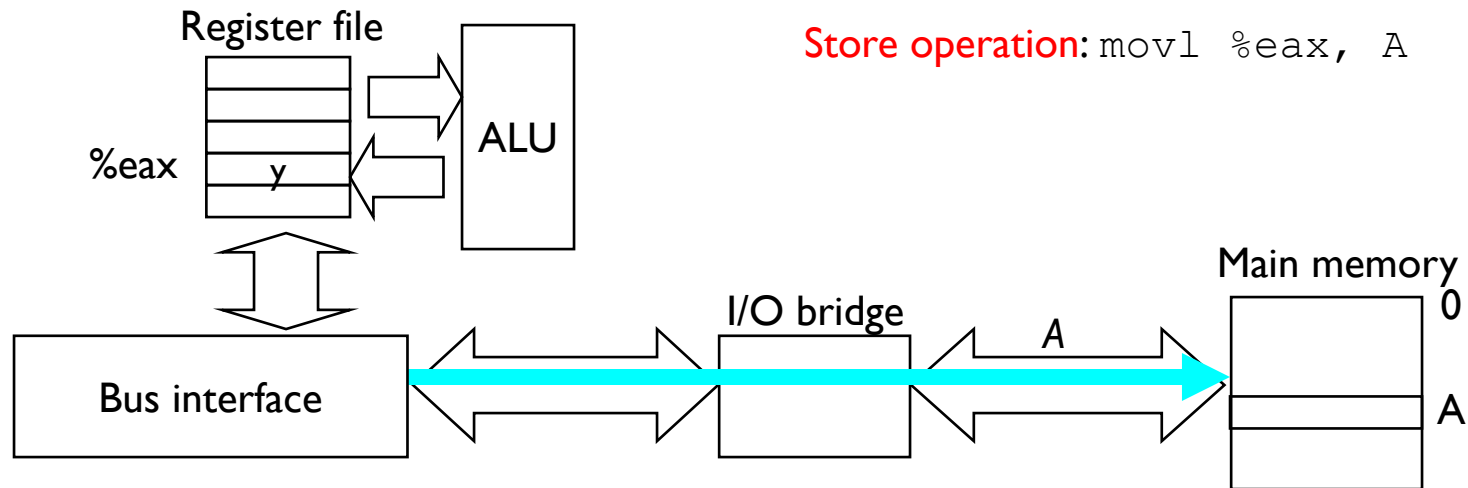
# Memory Read Transaction (3)

- ▶ CPU read word  $x$  from the bus and copies it into register `%eax`.



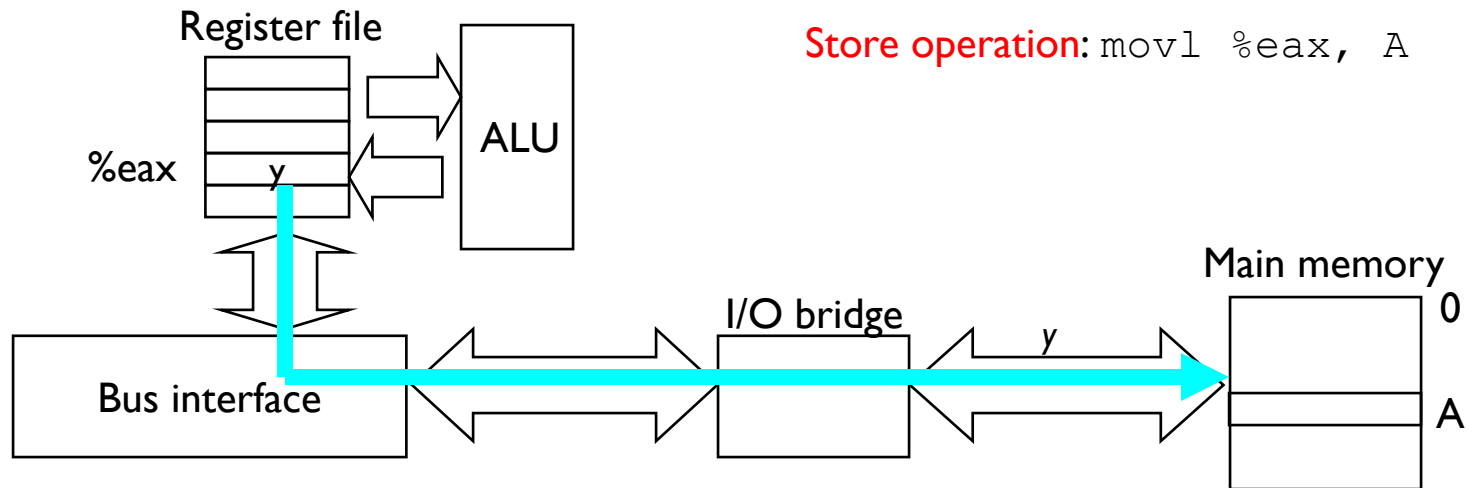
# Memory Write Transaction (1)

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



# Memory Write Transaction (2)

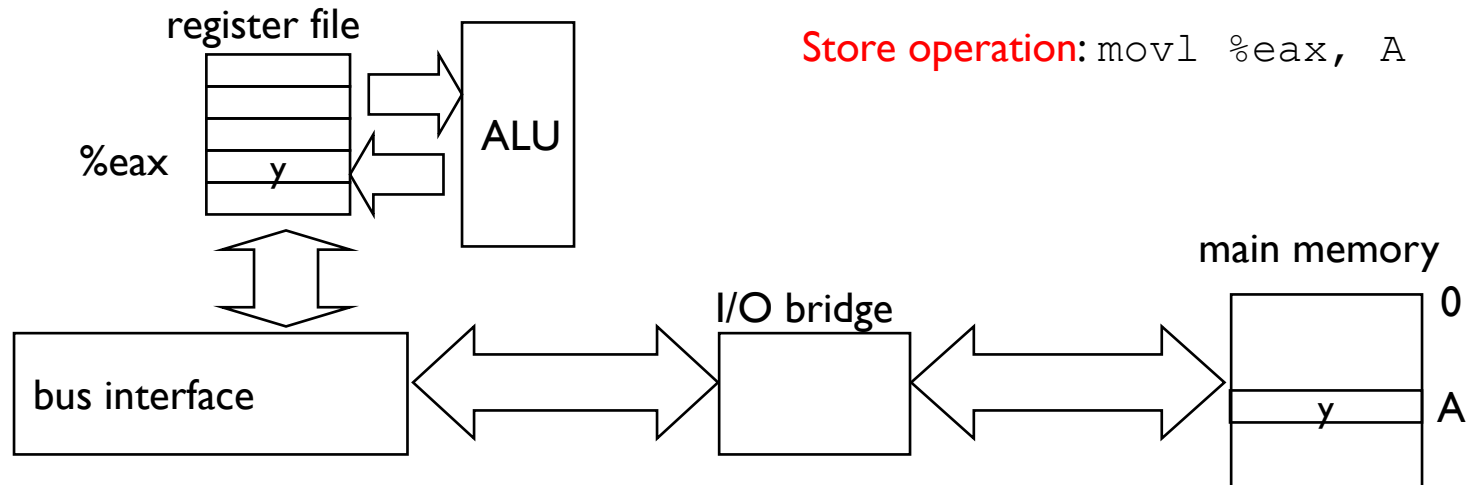
- ▶ CPU places data word  $y$  on the bus.





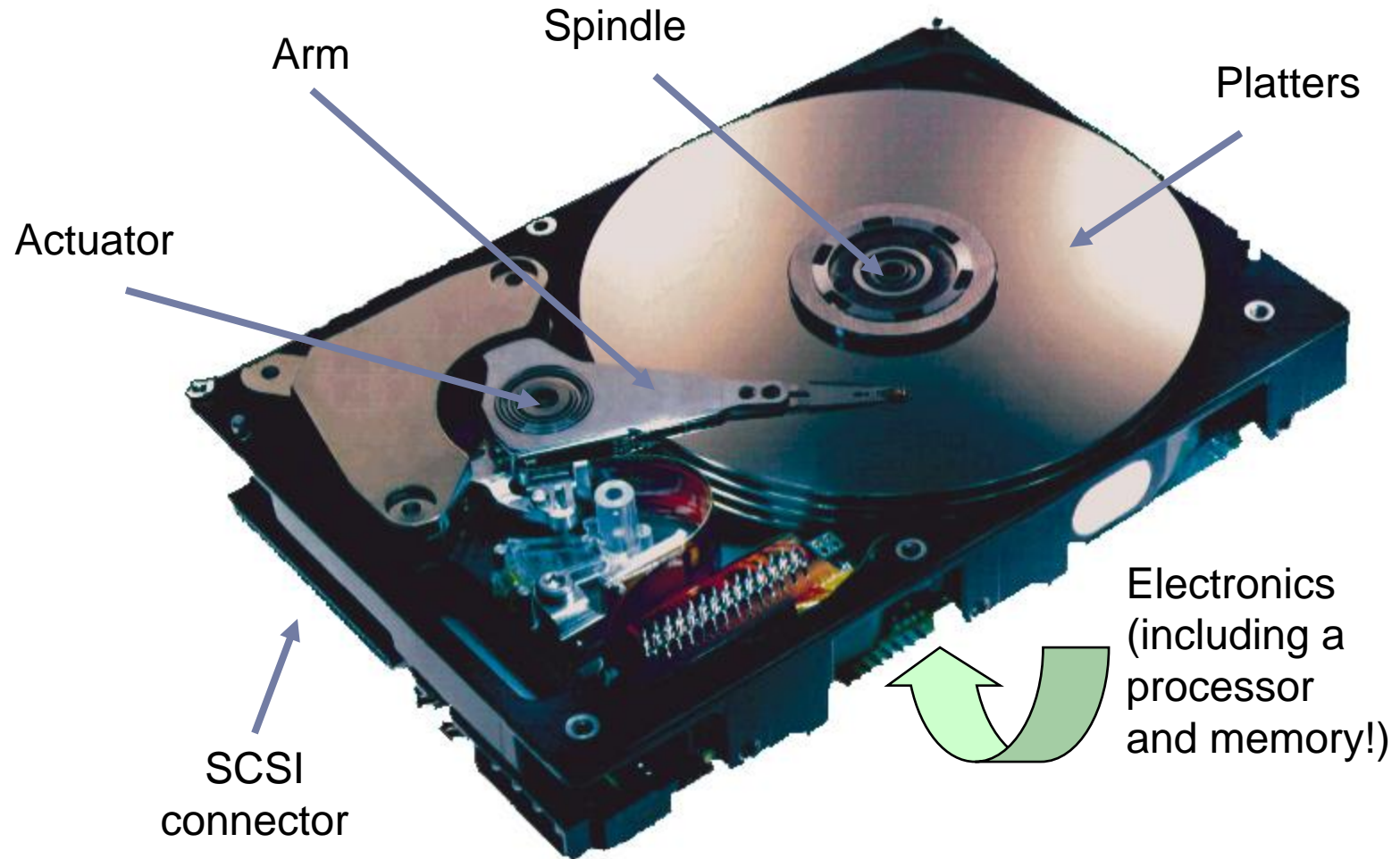
# Memory Write Transaction (3)

- ▶ Main memory reads data word  $y$  from the bus and stores it at address  $A$ .



# What's Inside A Disk Drive?

---

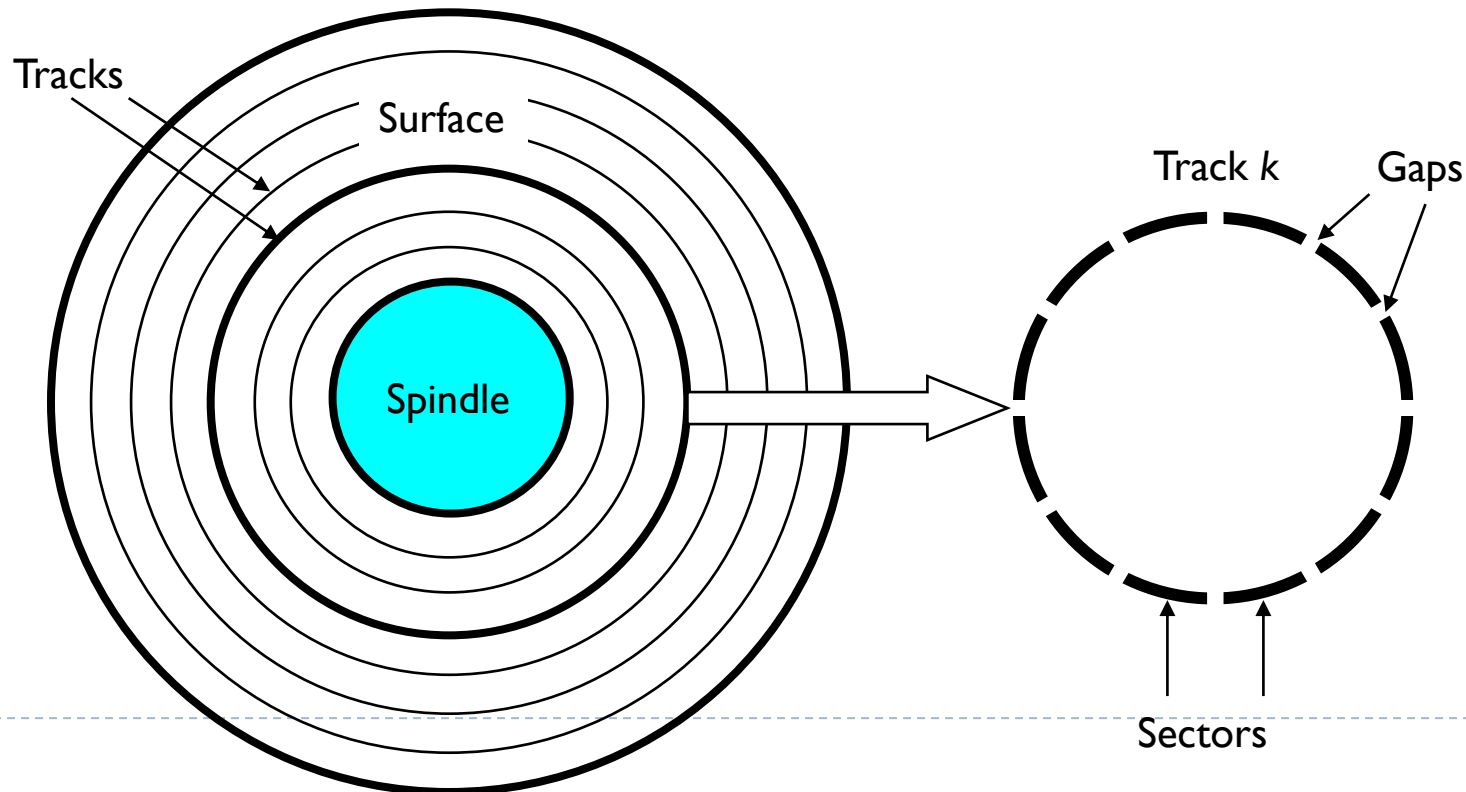


---

*Image courtesy of Seagate Technology*

# Disk Geometry

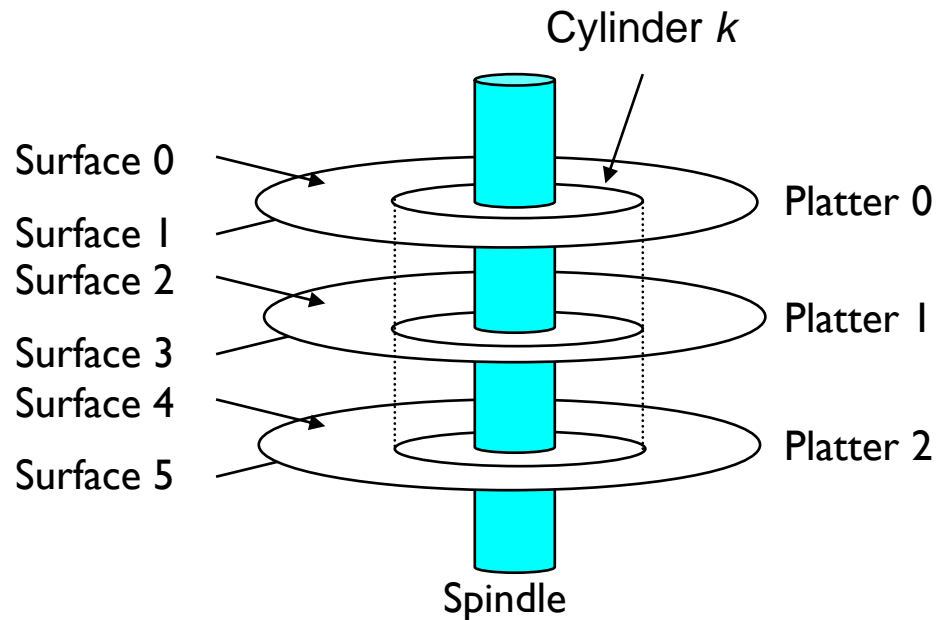
- ▶ Disks consist of **platters**, each with two **surfaces**.
- ▶ Each surface consists of concentric rings called **tracks**.
- ▶ Each track consists of **sectors** separated by **gaps**.



# Disk Geometry (Multiple-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 =  $10^9$  Bytes (Lawsuit pending! Claims deceptive advertising).
- ▶ 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<sup>2</sup>): 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

---

$$\text{Capacity} = (\# \text{ bytes/sector}) \times (\text{avg.} \# \text{ sectors/track}) \times$$
$$(\# \text{ tracks/surface}) \times (\# \text{ surfaces/platter}) \times$$
$$(\# \text{ platters/disk})$$

Example:

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

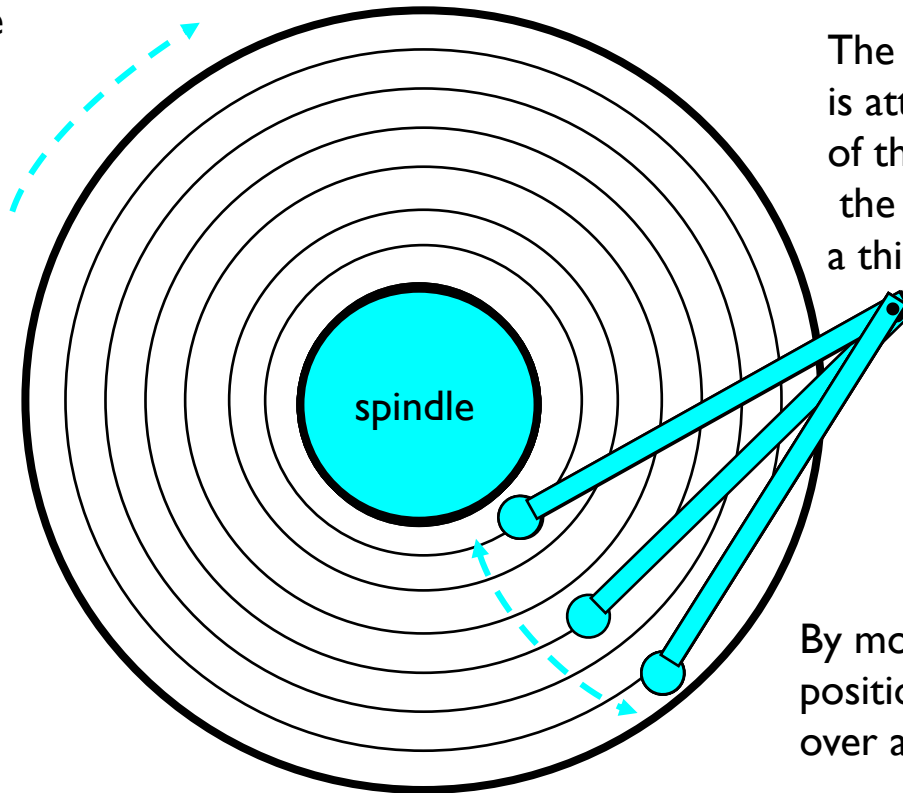
$$\begin{aligned}\text{Capacity} &= 512 \times 300 \times 20000 \times 2 \times 5 \\ &= 30,720,000,000 \\ &= 30.72 \text{ GB}\end{aligned}$$



# Disk Operation (Single-Platter View)

---

The disk surface spins at a fixed rotational rate



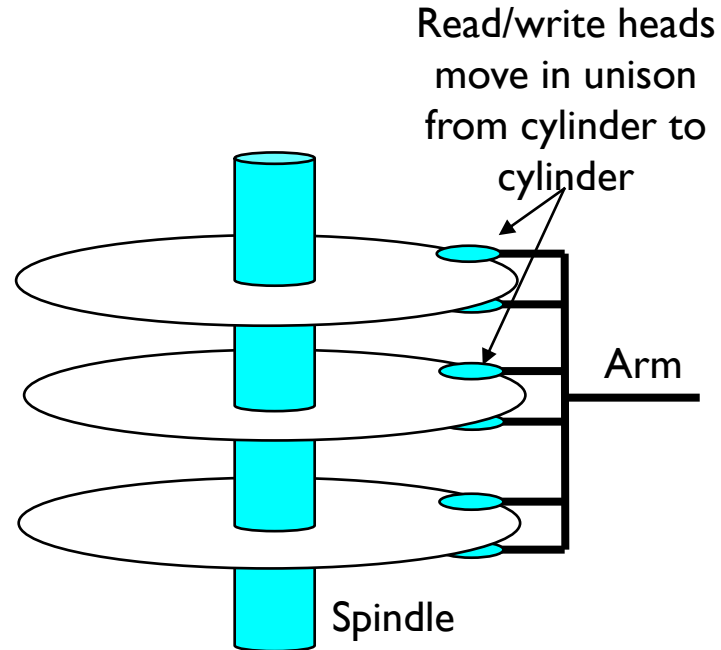
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

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



# Disk Operation (Multi-Platter View)

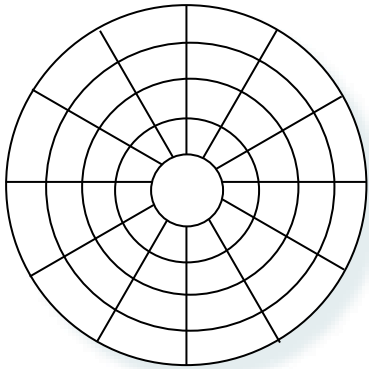
---





# Disk Structure - top view of single platter

---



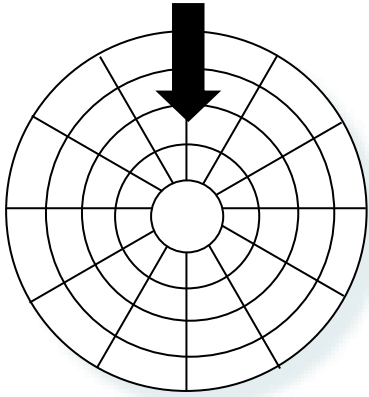
Surface organized into tracks

Tracks divided into sectors



# Disk Access

---

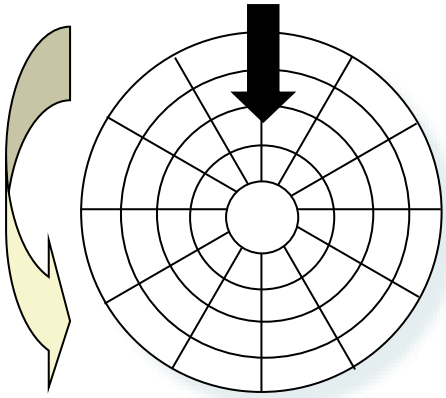


Head in position above a track



# Disk Access

---

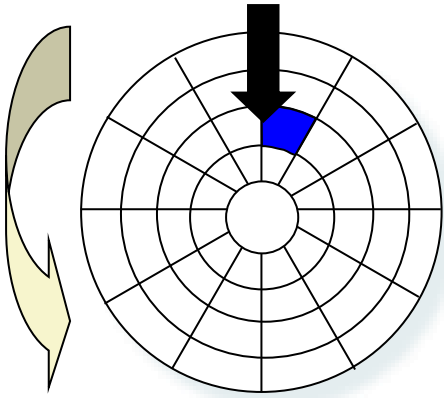


Rotation is counter-clockwise



# Disk Access – Read

---

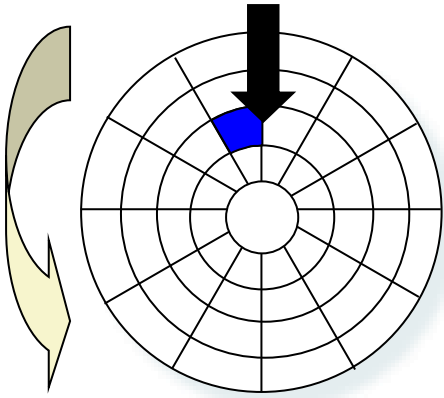


About to read blue sector



# Disk Access – Read

---



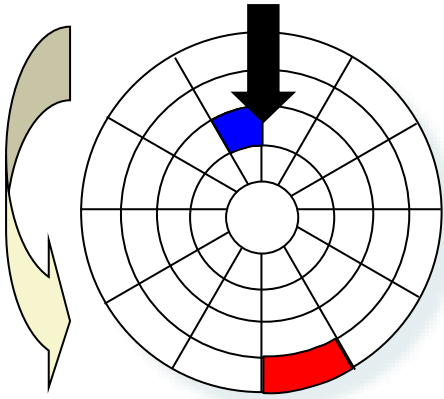
After **BLUE** read

After reading blue sector



# Disk Access – Read

---



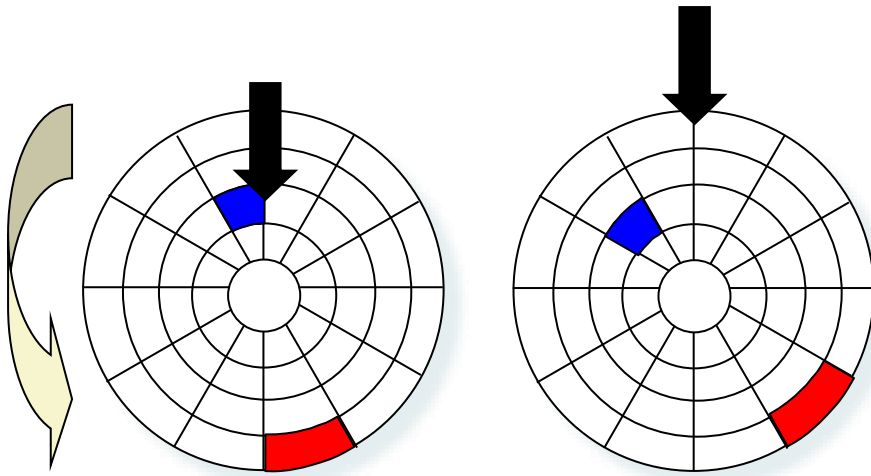
After **BLUE** read

Red request scheduled next



# Disk Access – Seek

---



After **BLUE** read

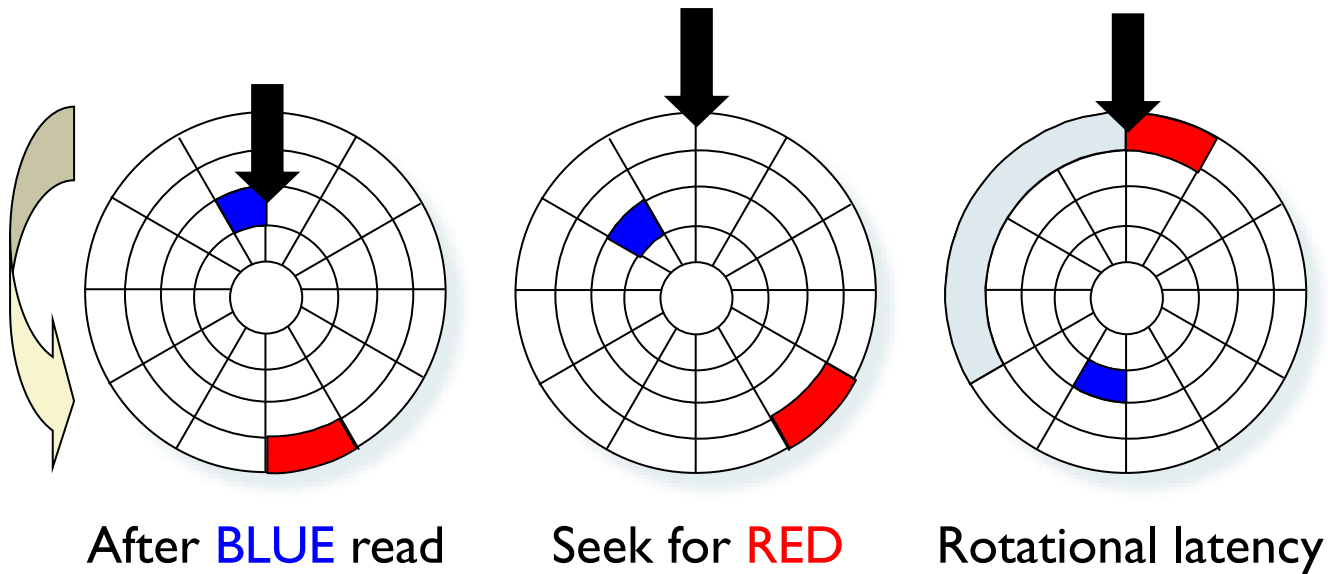
Seek for **RED**

Seek to red's track



# Disk Access – Rotational Latency

---

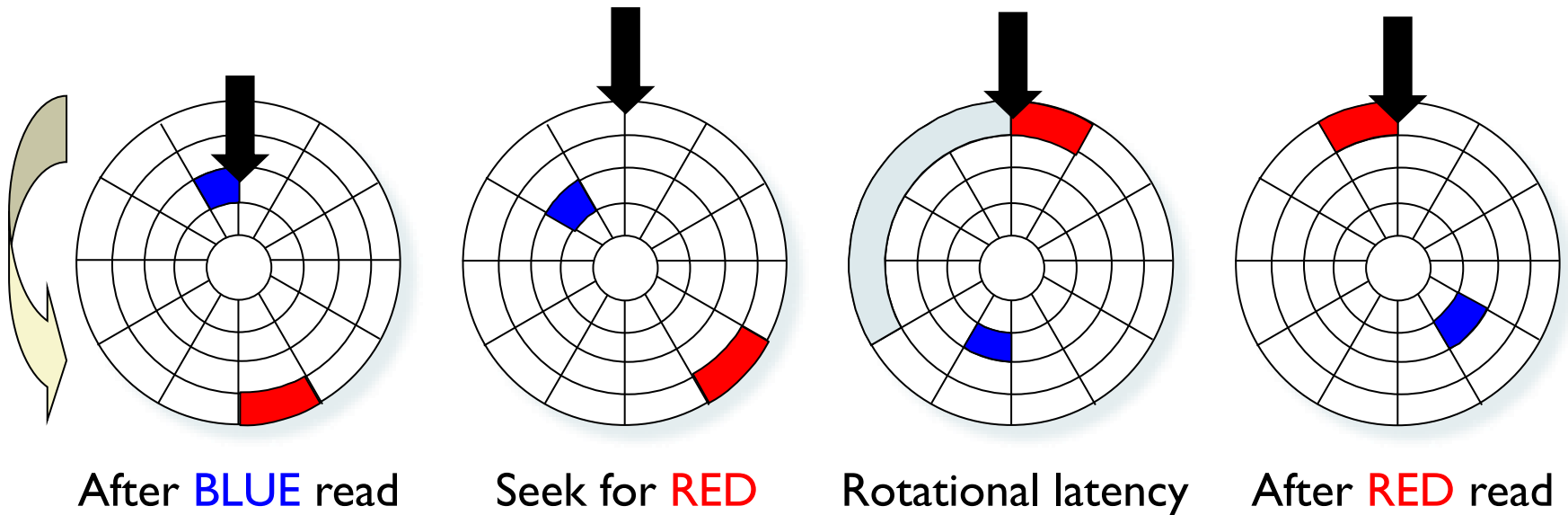


Wait for red sector to rotate around



# Disk Access – Read

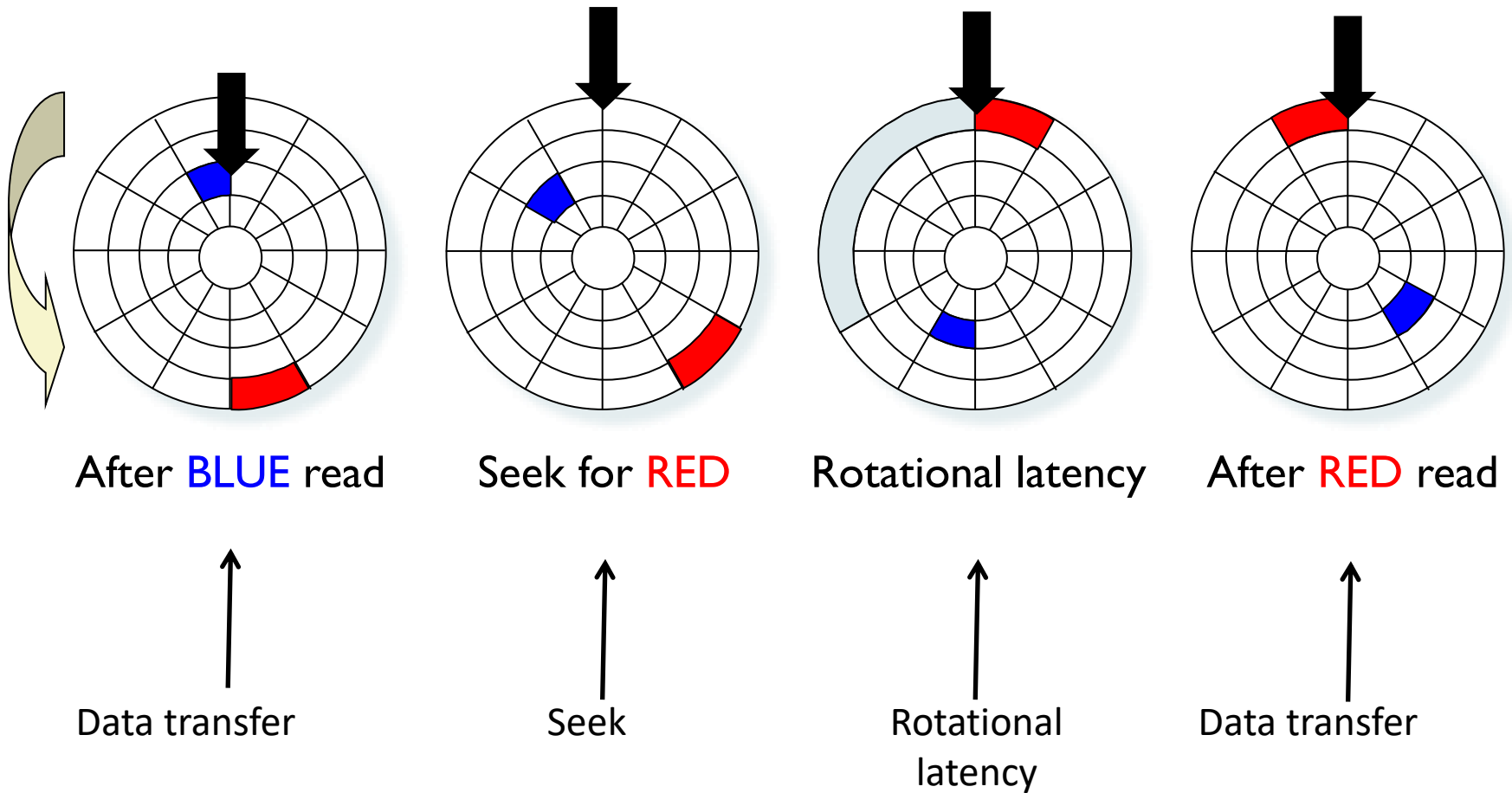
---



Complete read of red



# Disk Access – Service Time Components



# Disk Access Time

---

- ▶ Average time to access some target sector approximated by :
  - ▶  $T_{\text{access}} = T_{\text{avg seek}} + T_{\text{avg rotation}} + T_{\text{avg transfer}}$
- ▶ **Seek time** ( $T_{\text{avg seek}}$ )
  - ▶ Time to position heads over cylinder containing target sector.
  - ▶ Typical  $T_{\text{avg seek}}$  is 3—9 ms
- ▶ **Rotational latency** ( $T_{\text{avg rotation}}$ )
  - ▶ Time waiting for first bit of target sector to pass under r/w head.
  - ▶  $T_{\text{avg rotation}} = 1/2 \times 1/\text{RPMs} \times 60 \text{ sec}/1 \text{ min}$
  - ▶ Typical  $T_{\text{avg rotation}} = 7200 \text{ RPMs}$
- ▶ **Transfer time** ( $T_{\text{avg transfer}}$ )
  - ▶ Time to read the bits in the target sector.
  - ▶  $T_{\text{avg transfer}} = 1/\text{RPM} \times 1/(\text{avg \# sectors/track}) \times 60 \text{ secs}/1 \text{ min}.$



# Disk Access Time Example

---

## ▶ Given:

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

## ▶ Derived:

- ▶  $T_{\text{avg rotation}} = 1/2 \times (60 \text{ secs}/7200 \text{ RPM}) \times 1000 \text{ ms/sec} = 4 \text{ ms}.$
- ▶  $T_{\text{avg transfer}} = 60/7200 \text{ RPM} \times 1/400 \text{ secs/track} \times 1000 \text{ ms/sec} = 0.02 \text{ ms}$
- ▶  $T_{\text{access}} = 9 \text{ ms} + 4 \text{ ms} + 0.02 \text{ ms}$

## ▶ 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 than DRAM.



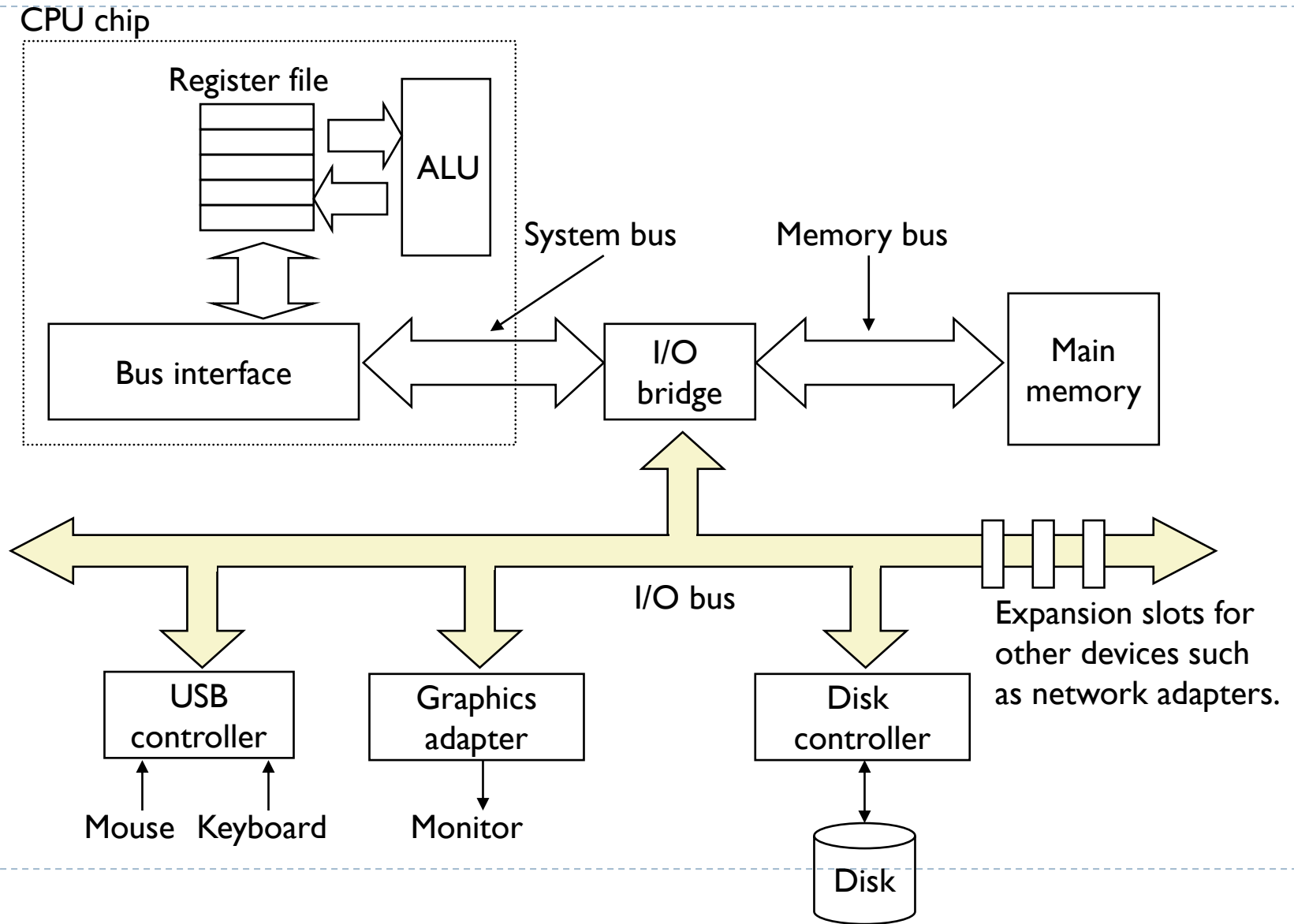
# Logical Disk Blocks

---

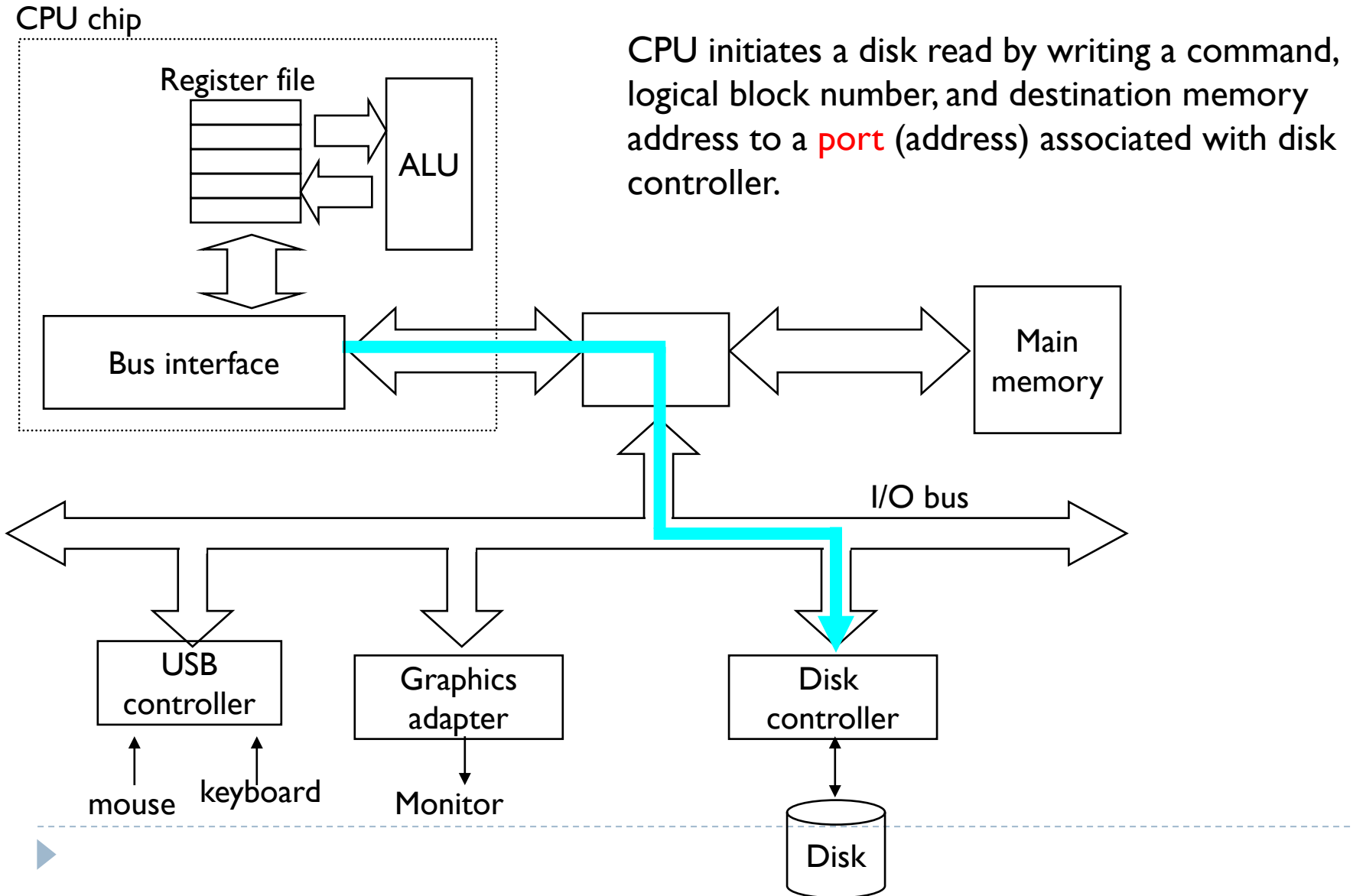
- ▶ Modern disks present a simpler abstract view of the complex sector geometry:
  - ▶ The set of available sectors is modeled as a sequence of b-sized **logical blocks** (0, 1, 2, ...)
- ▶ Mapping between logical blocks and actual (physical) sectors
  - ▶ Maintained by hardware/firmware device called 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

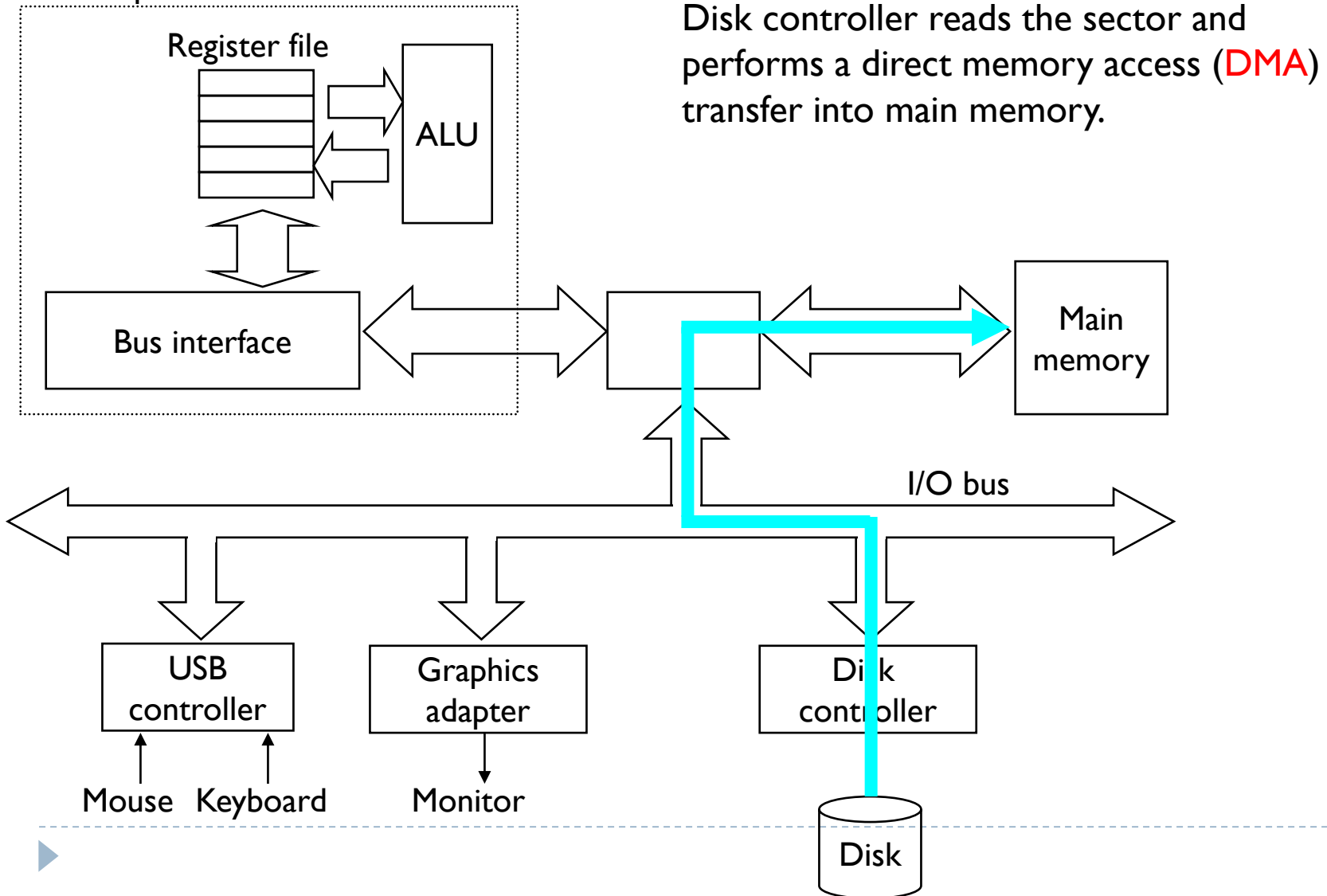


# Reading a Disk Sector (1)



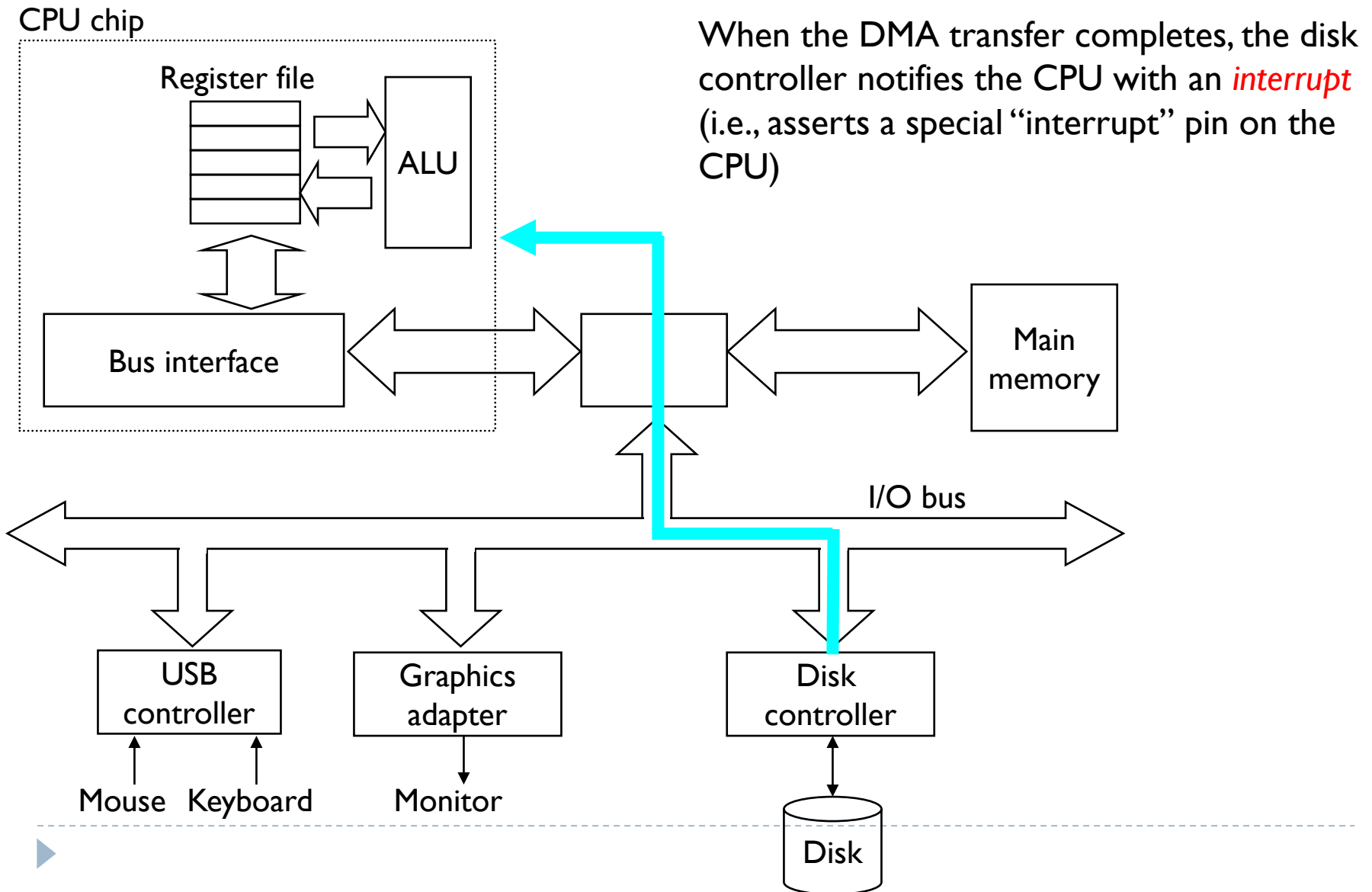
# Reading a Disk Sector (2)

CPU chip

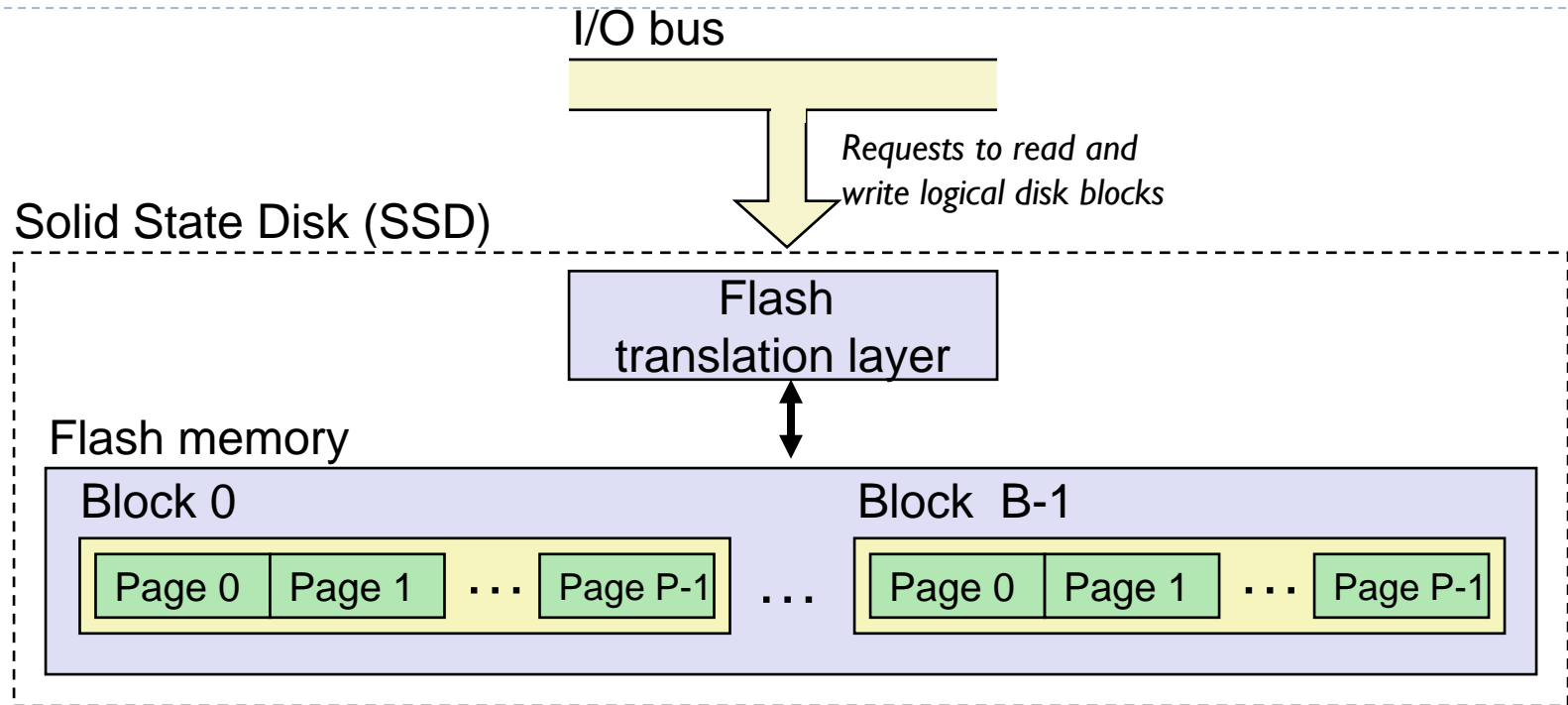




# Reading a Disk Sector (3)



# Solid State Disks (SSDs)



- ▶ Pages: 512 to 4KB, Blocks: 32 to 128 pages
- ▶ Data read/written in units of pages.
- ▶ Page can be written only after its block has been erased
- ▶ A block wears out after 100,000 repeated writes.

# SSD Performance Characteristics

---

Sequential read tput	550 MB/s	Sequential write tput	470 MB/s
Random read tput	365 MB/s	Random write tput	303 MB/s
Avg seq read time	30 us	Avg seq write time	60 us

- ▶ Sequential access faster than random access
  - ▶ Common theme in the memory hierarchy
- ▶ Random writes are somewhat slower
  - ▶ Erasing a block takes a long time (~1 ms)
  - ▶ Modifying a block page requires all other pages to be copied to new block
  - ▶ In earlier SSDs, the read/write gap was much larger.

# SSD Tradeoffs vs Rotating Disks

---

## ▶ Advantages

- ▶ No moving parts → faster, less power, more rugged

## ▶ Disadvantages

- ▶ Have the potential to wear out
  - ▶ Mitigated by “wear leveling logic” in flash translation layer
  - ▶ E.g. Intel SSD 730 guarantees 128 petabyte ( $128 \times 10^{15}$  bytes) of writes before they wear out
- ▶ In 2015, about 30 times more expensive per byte

## ▶ Applications

- ▶ MP3 players, smart phones, laptops
  - ▶ Beginning to appear in desktops and servers
- 



# Storage Trends

## SRAM

Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
\$/MB	19,200	2,900	320	256	100	75	60	320
access (ns)	300	150	35	15	3	2	1.5	200

## DRAM

Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
\$/MB	8,000	880	100	30	1	0.1	0.06	130,000
access (ns)	375	200	100	70	60	50	40	9
typical size (MB)	0.064	0.256	4	16	64	2,000	8,000	125,000

## Disk

Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
\$/MB	500	100	8	0.30	0.01	0.005	0.0003	1,600,000
access (ms)	87	75	28	10	8	4	3	29
typical size (MB)	1	10	160	1,000	20,000	160,000	1,500,000	1,500,000

# CPU Clock Rates

Inflection point in computer history  
when designers hit the “Power Wall”

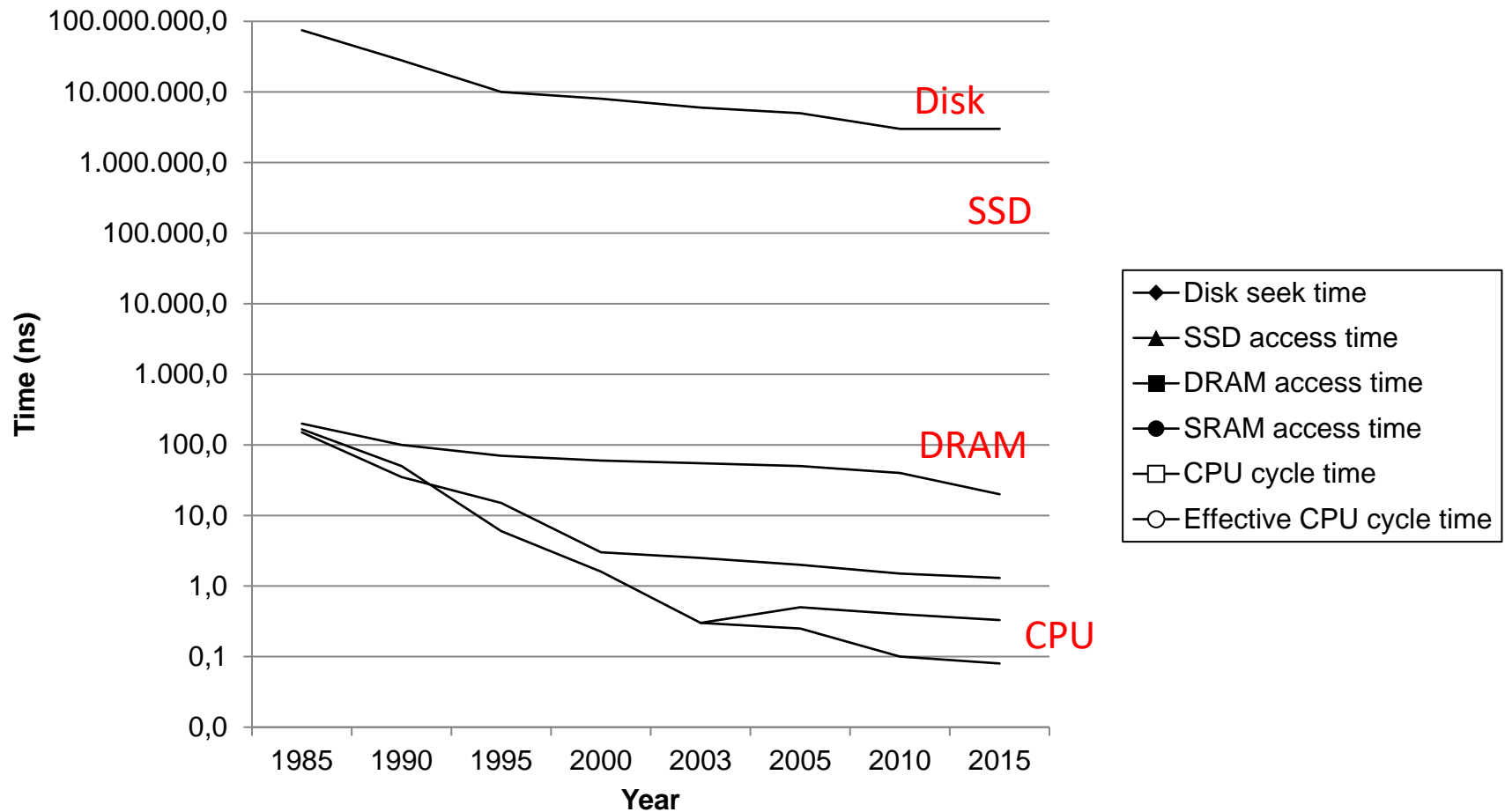


	1980	1990	1995	2000	2003	2005	2010	2010:1980
CPU	8080	386	Pentium	P-III	P-4	Core 2	Core i7	---
Clock rate (MHz)	1	20	150	600	3300	2000	2500	2500
Cycle time (ns)	1000	50	6	1.6	0.3	0.50	0.4	2500
Cores	1	1	1	1	1	2	4	4
Effective cycle time (ns)	1000	50	6	1.6	0.3	0.25	0.1	10,000



# 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**





# Today

---

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



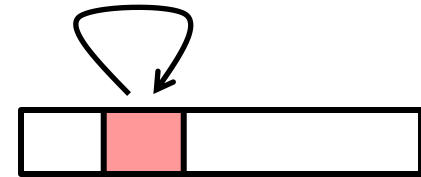
# Locality

---

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

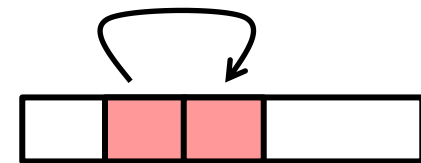
- ▶ **Temporal locality:**

- ▶ 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).
- ▶ Reference variable `sum` each iteration.

Spatial locality

Temporal locality

## ▶ Instruction references

- ▶ Reference instructions in sequence.
- ▶ Cycle through loop repeatedly.

Spatial locality

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;
}
```

# Locality Example

---

- ▶ **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 < N; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < M; 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**.



# Today

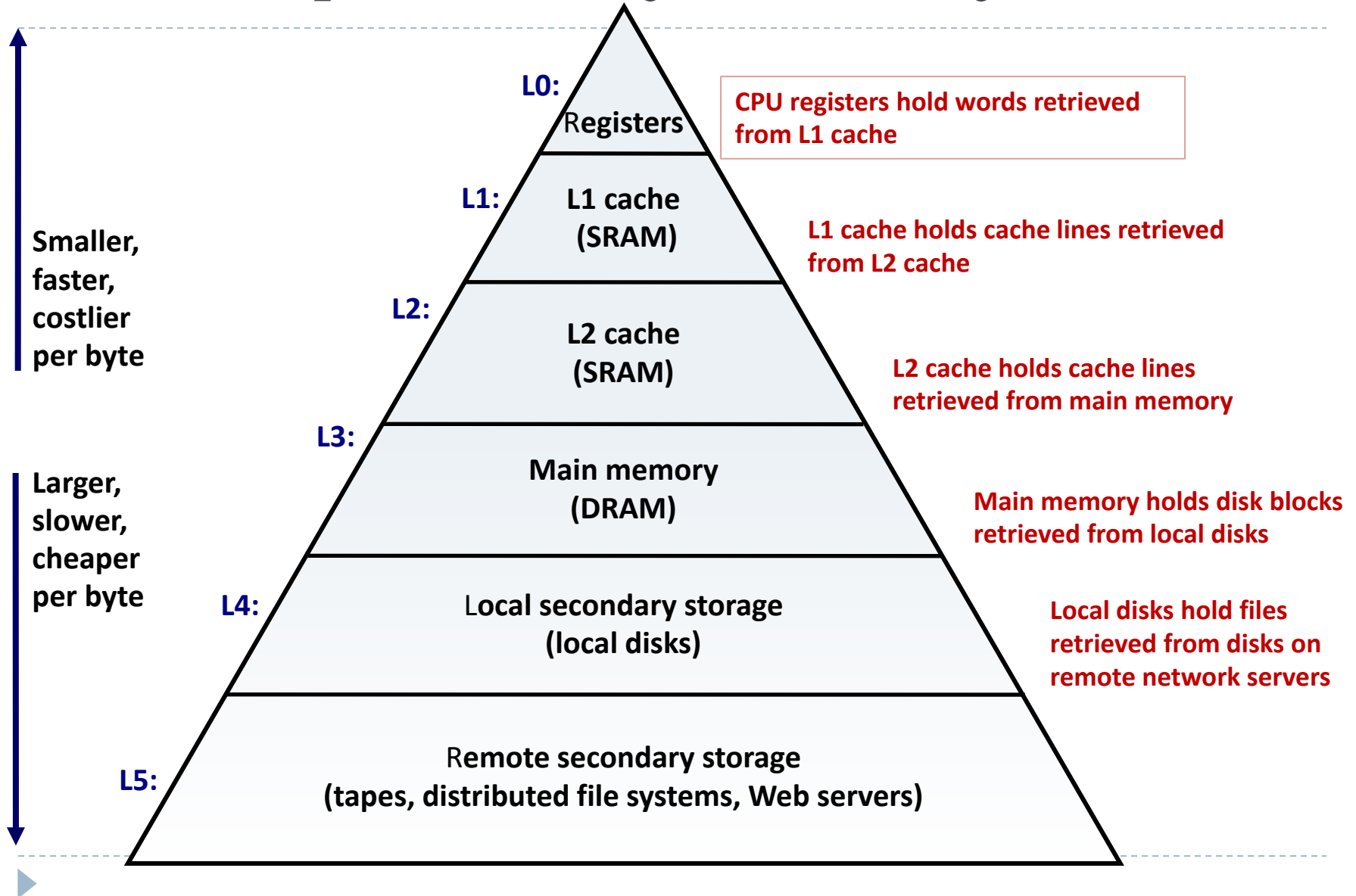
---

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





# An Example Memory Hierarchy



# Caches

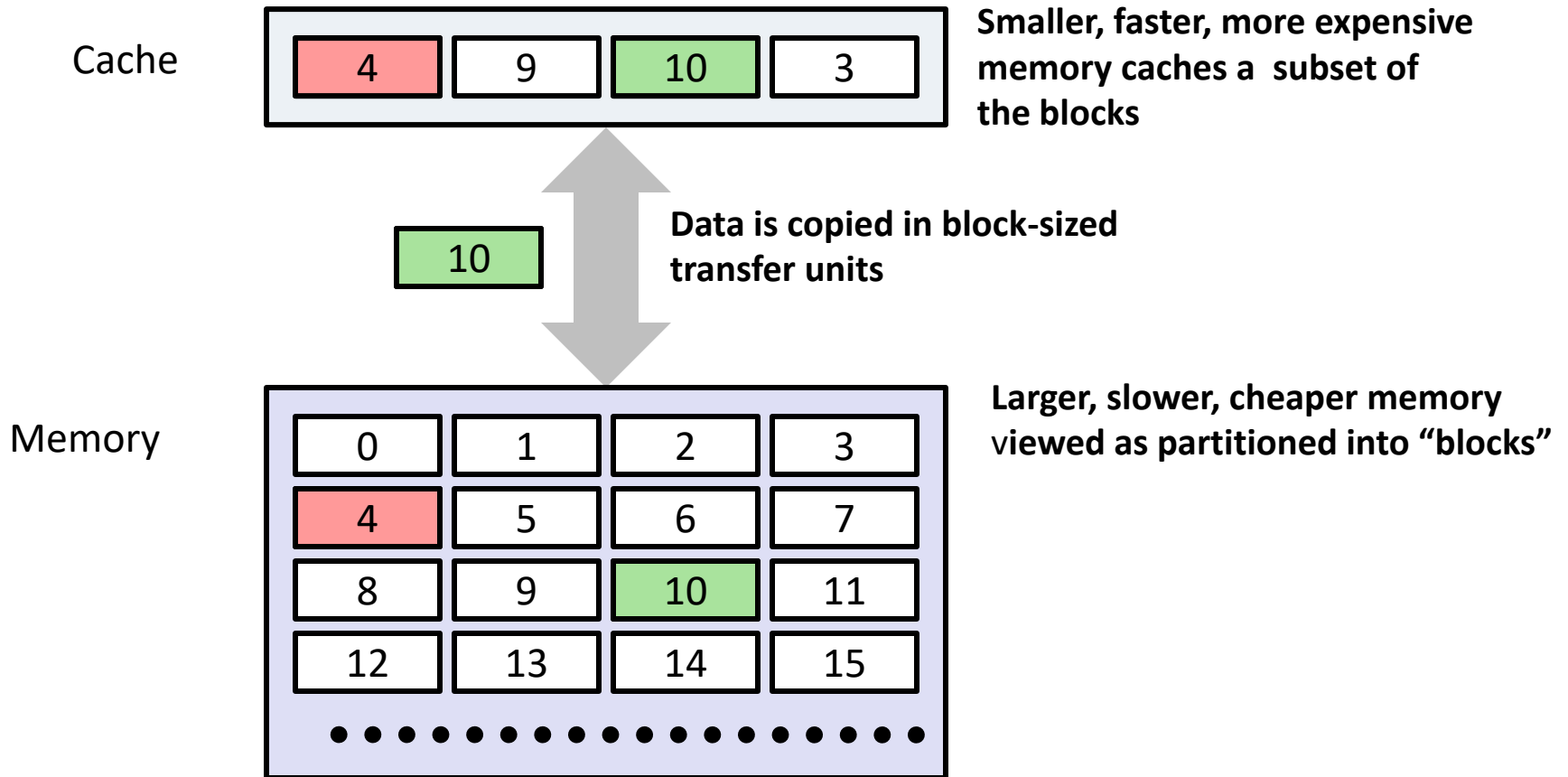
---

- ▶ **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?
    - ▶ 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.
- 

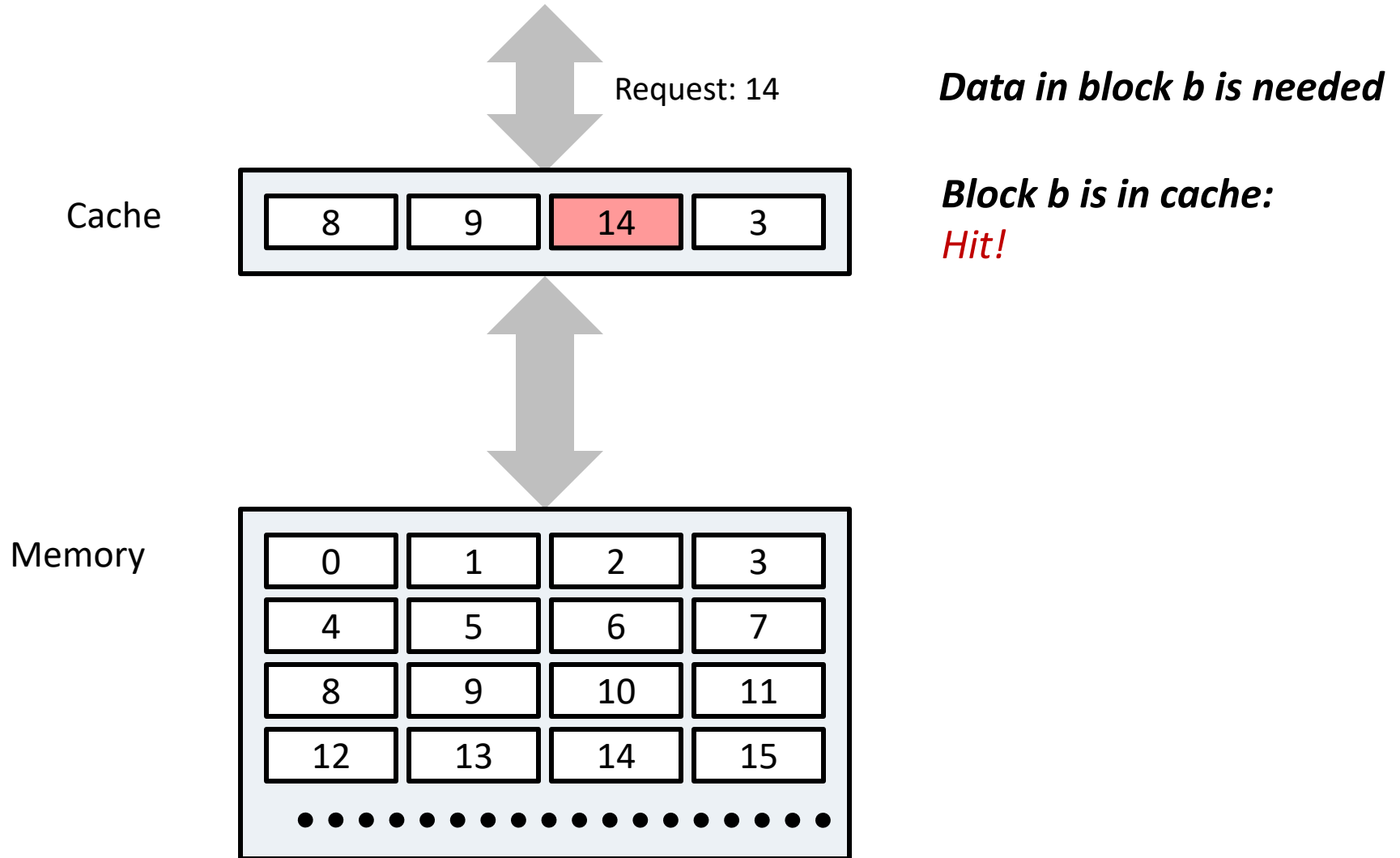


# General Cache Concepts

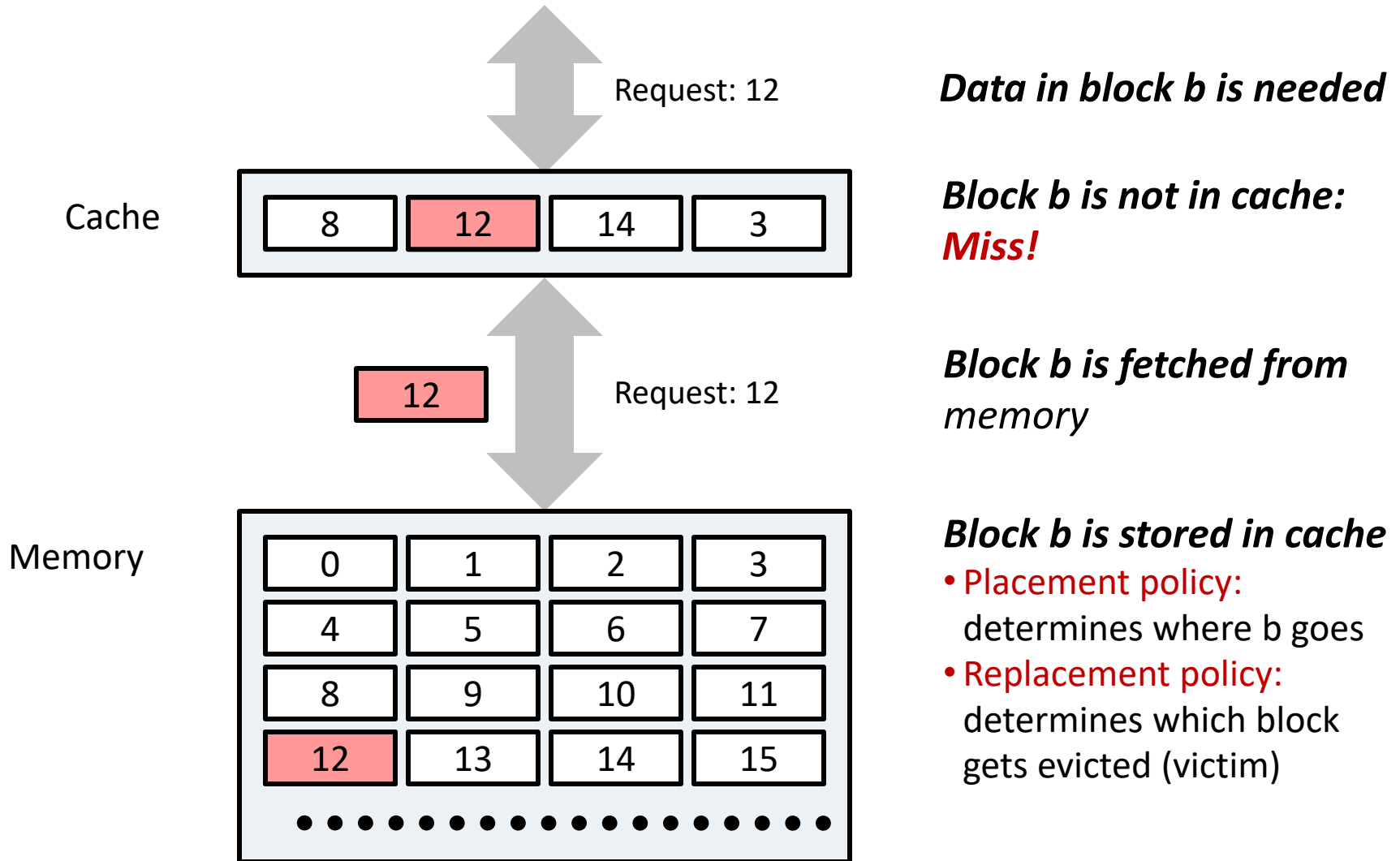
---



# General Cache Concepts: Hit



# General Cache Concepts: Miss



# 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  $(i \bmod 4)$  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

# Latency number every programmers should know

---

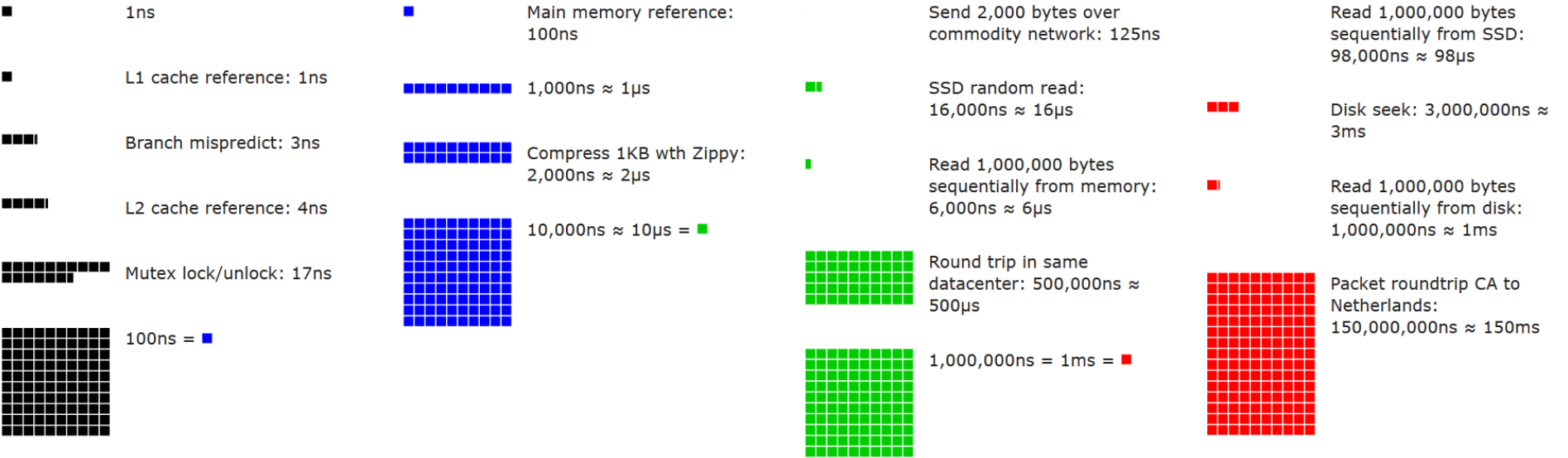
execute typical instruction	$1/1,000,000,000 \text{ sec} = 1 \text{ nanosec}$
fetch from L1 cache memory	0.5 nanosec
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec





Latency Numbers Every Programmer Should Know

2017



# 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.

