[CSED211] Introduction to Computer Software Systems

Lecture 10: Memory Hierarchy

Prof. Jisung Park

2023.11.06

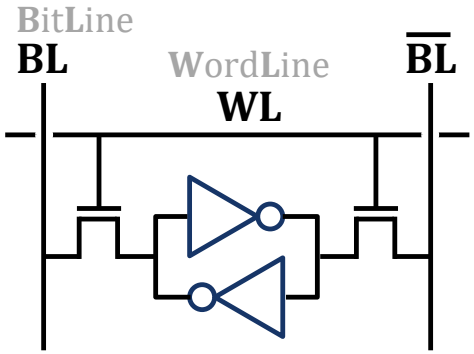
Lecture Agenda

- Storage Technologies and Trends
- Locality of Reference
- Caching in the Memory Hierarchy

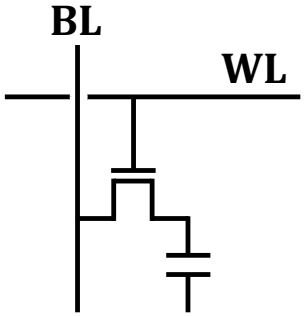
Random-Access Memory (RAM)

- RAM is traditionally packaged as a chip
  - Basic storage unit is normally a cell (e.g., one bit per cell)
  - Multiple RAM chips form a memory module
- 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 a bit with one capacitor and one transistor
  - Value must be refreshed every 10-100 ms (e.g., 64 ms)
  - More sensitive to disturbances (EMI, radiation,...) than SRAM
  - Slower and cheaper than SRAM

SRAM vs DRAM: Summary



SRAM Cell



DRAM Cell

	TR per Bit	Access Time	Needs Refresh?	Needs EDC?	Cost	Applications
SRAM	4 or 6	1×	No	Maybe	100×	On-chip Cache
DRAM	1	10×	Yes	Yes	1×	Main Memory/ Frame Buffer

Enhanced DRAMs

- A 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), DDR4 (16 bits)
    - By 2010, standard for most server and desktop systems
    - Intel Core i7 supports DDR3 and DDR4 SDRAM

Nonvolatile Memory (NVM)

- DRAM and SRAM are volatile
  - Lose information if powered off
- Nonvolatile memory retains stored data even if power 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 (block-level) erase capability
  - 3D XPoint (Intel Optane) & emerging NVMs

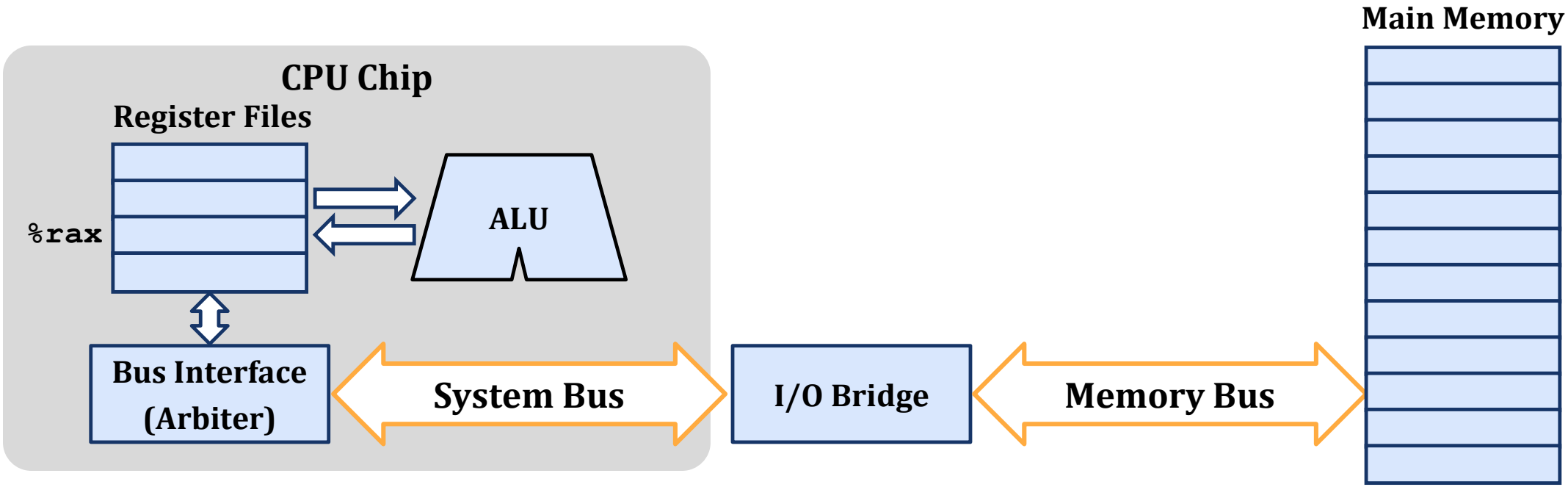
The Only Survivor after SRAM/DRAM

Nonvolatile Memory (Cont.)

- NVM usages
  - Firmware programs stored in a ROM
    - e.g., BIOS, controllers for disks, network cards, graphics accelerators, security subsystems, etc.
  - Solid-state drives (SSDs)
    - Replacing traditional hard-disk drives (HDDs) in thumb drives, smart phones, tablets, laptops, data centers, etc.

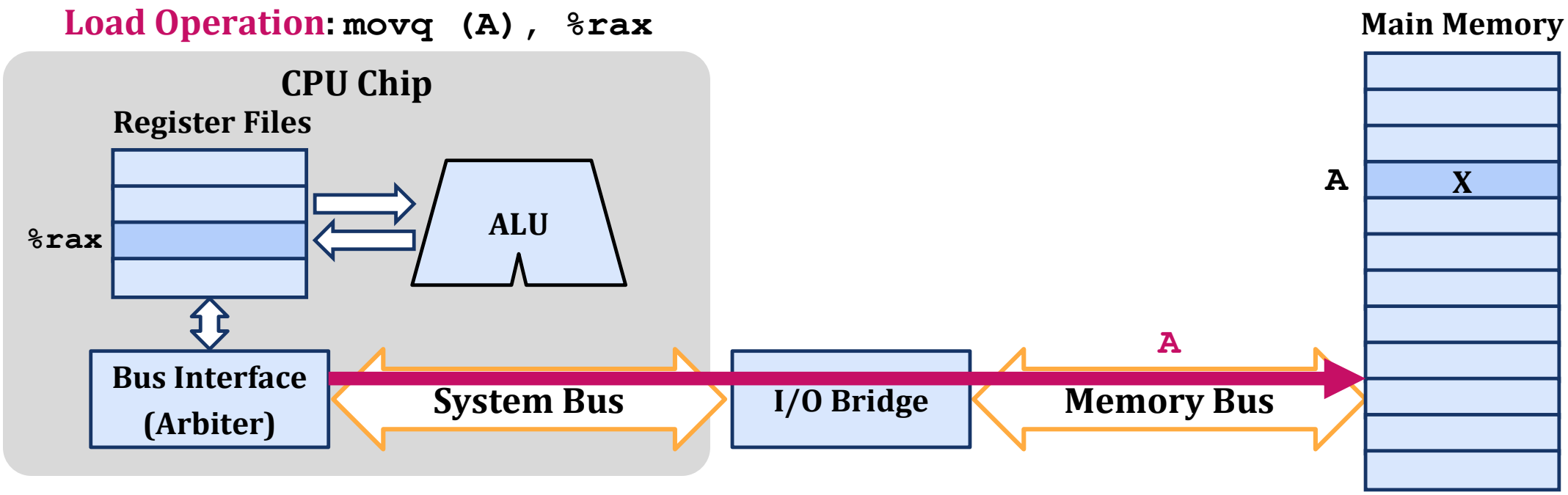
Bus Structure Connecting CPU and Memory

- A bus is a collection of parallel wires that carry addresses, data, and control signals (e.g., commands)
  - Typically shared by multiple devices



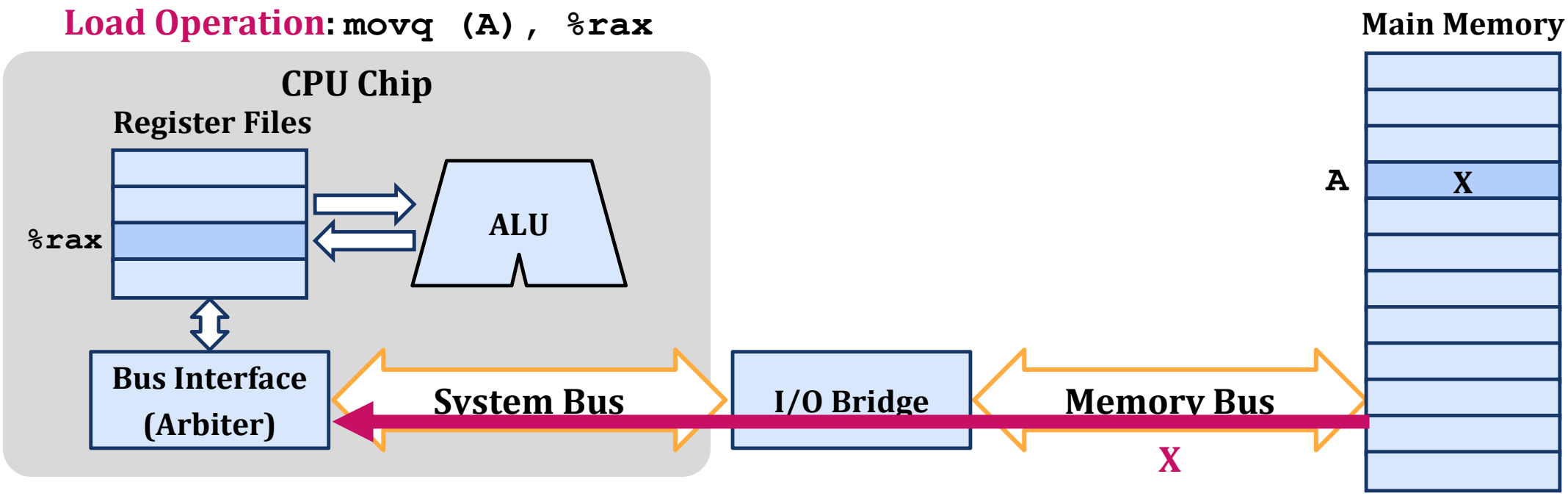
Memory Transaction: Read

- Step 1: CPU places address `A` on the memory bus



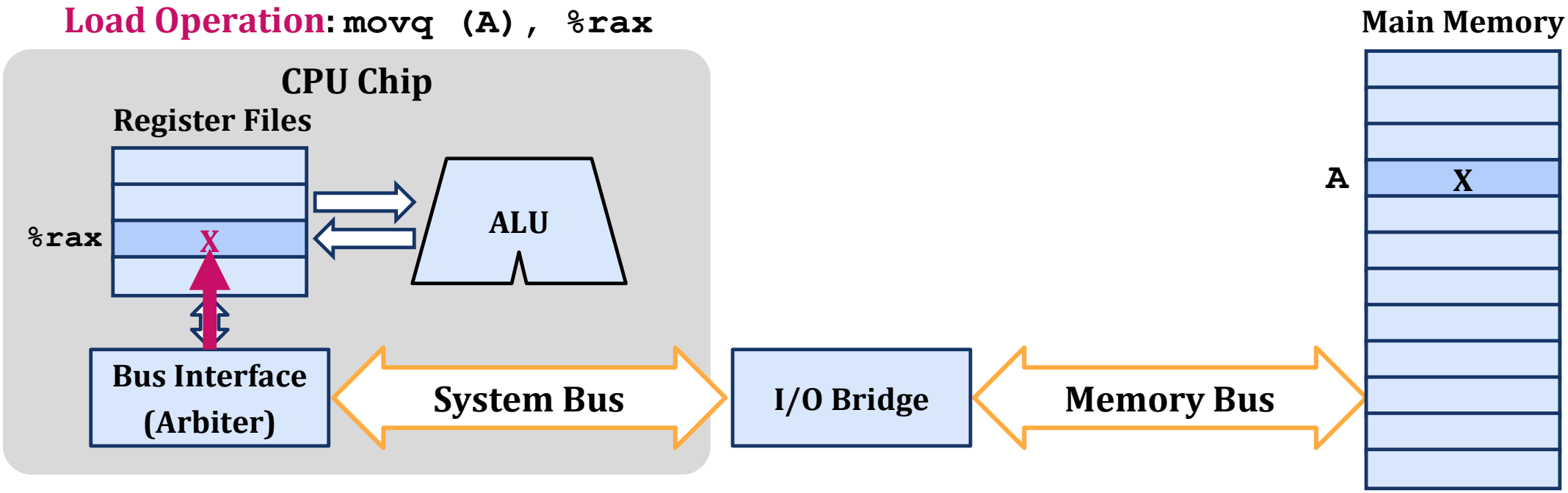
Memory Transaction: Read

- Step 1: CPU places address `A` on the memory bus
- Step 2: Memory reads `A` from the bus, retrieves word X, and places it to the bus



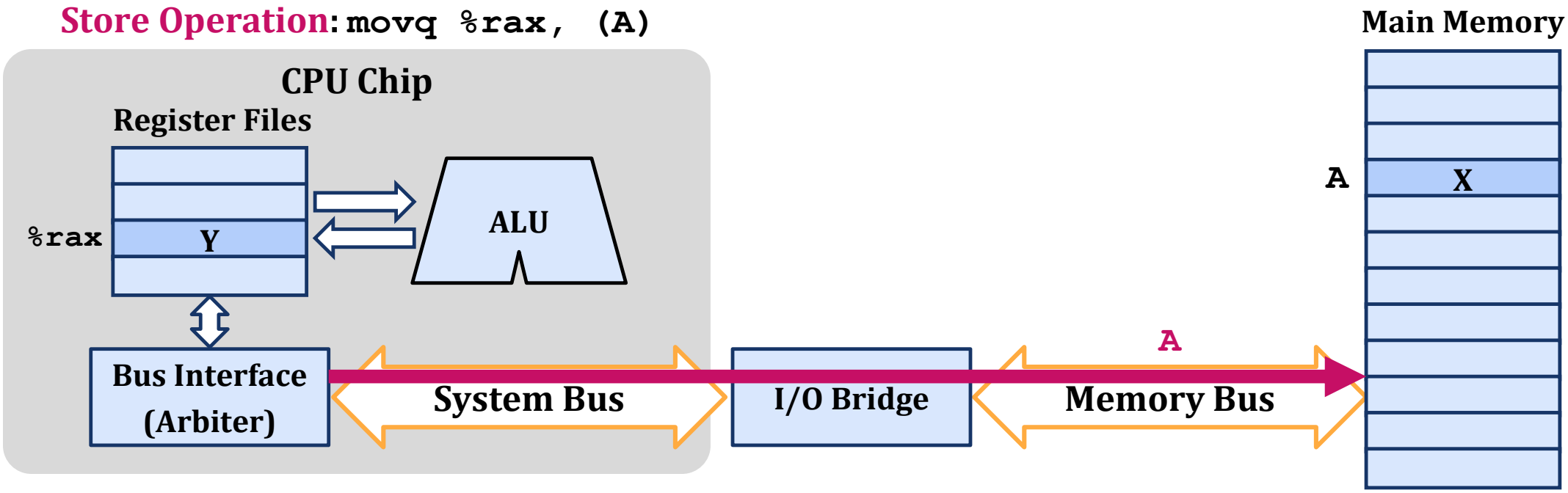
Memory Transaction: Read

- Step 1: CPU places address `A` on the memory bus
- Step 2: Memory reads `A` from the bus, retrieves word X, and places it to the bus
- Step 3: CPU read word X from the bus and copies it into register `%rax`



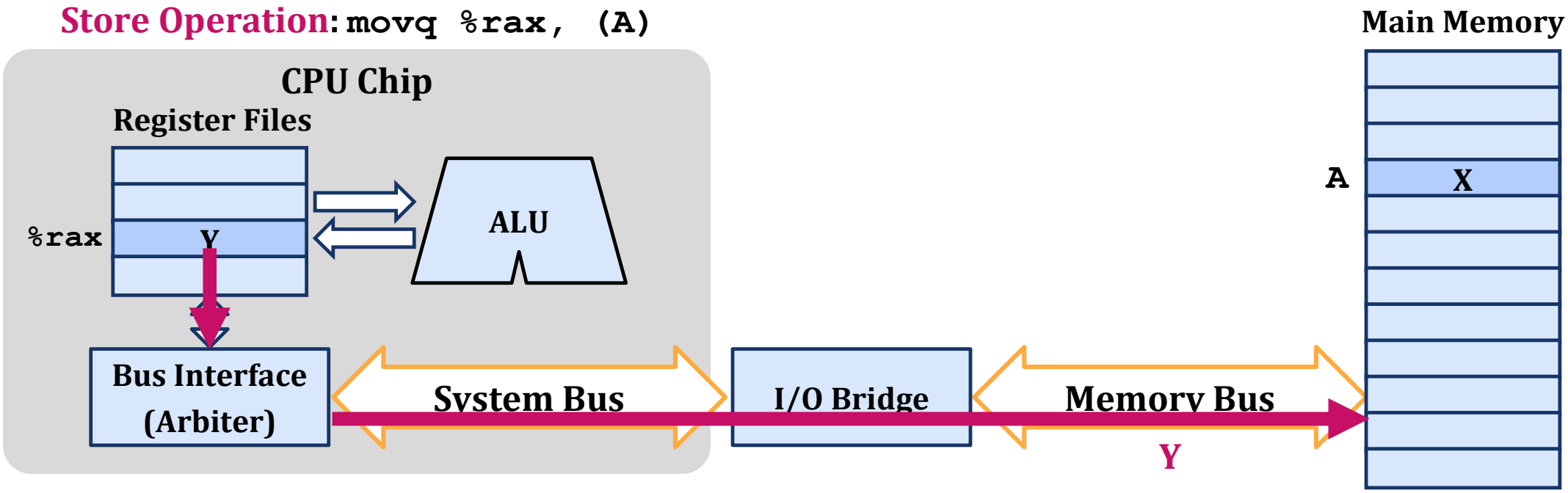
Memory Transaction: Write

- Step 1: CPU places address **A** on the bus, and memory waits for the data



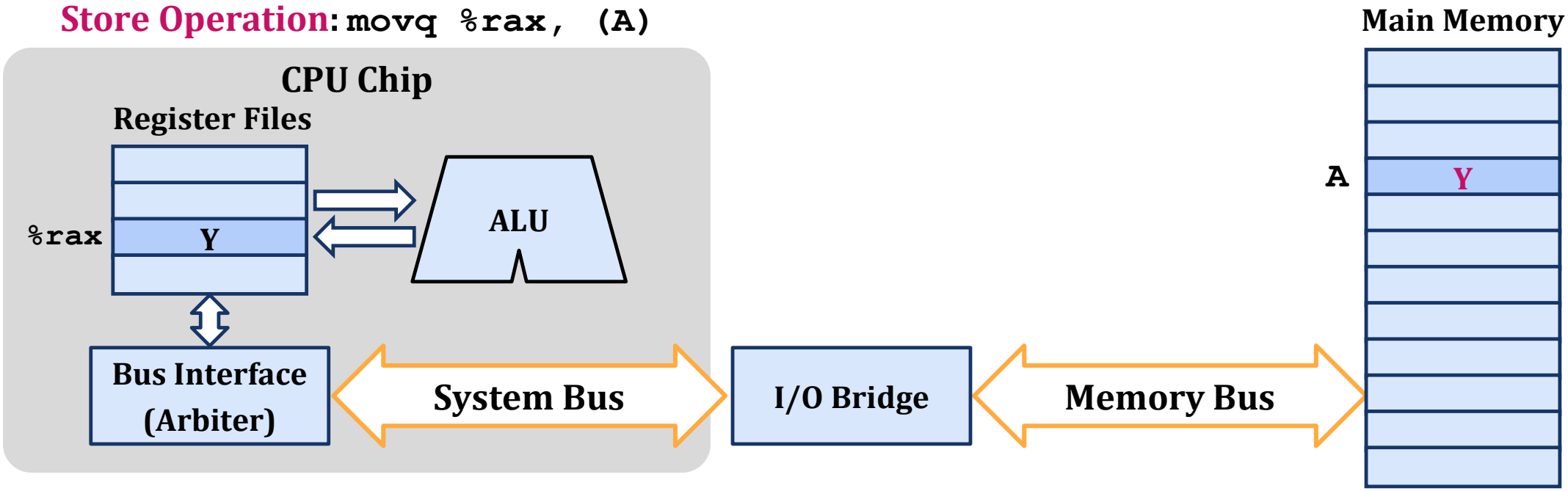
Memory Transaction: Write

- Step 1: CPU places address `A` on the bus, and memory waits for the data
- Step 2: CPU places data word Y on the bus

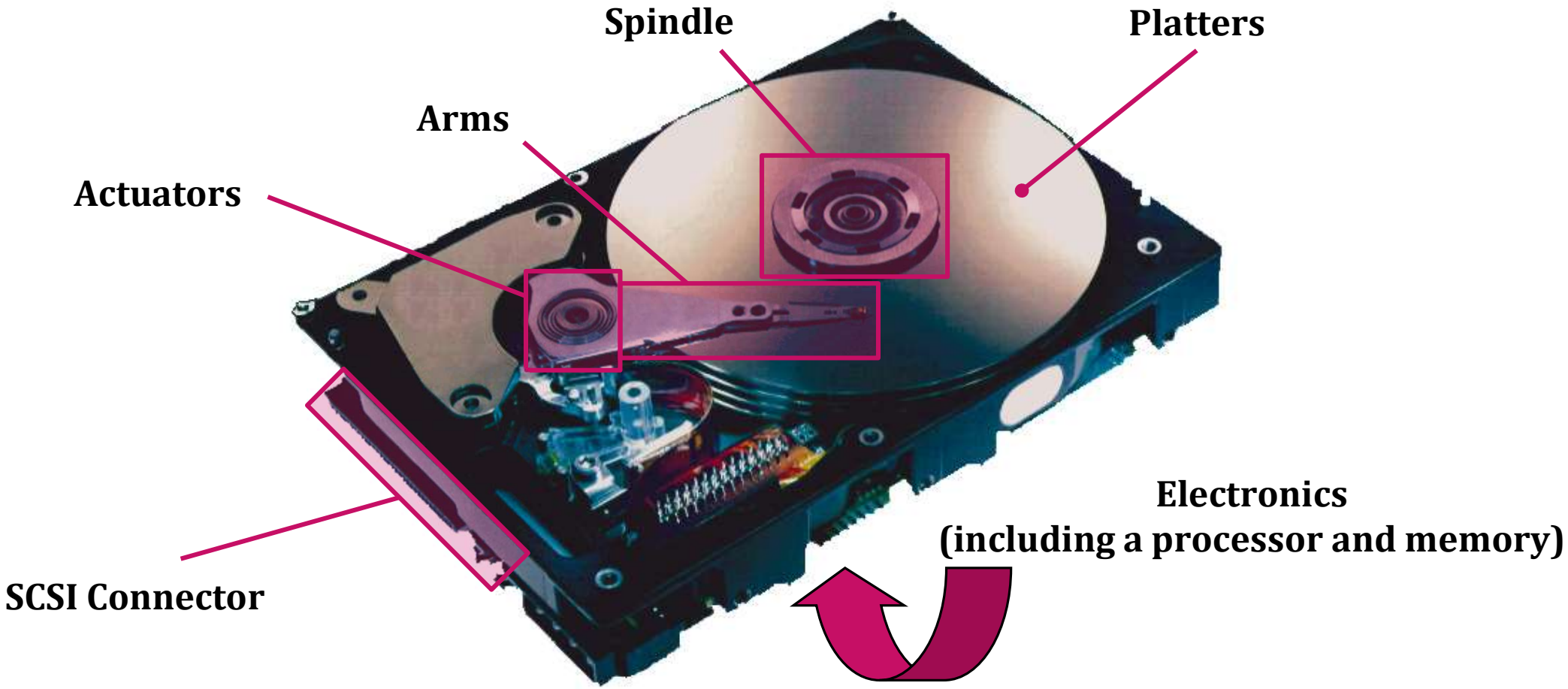


Memory Transaction: Write

- Step 1: CPU places address `A` on the bus, and memory waits for the data
- Step 2: CPU places data word Y on the bus
- Step 3: Memory reads data word Y from the bus and stores it at address `A`

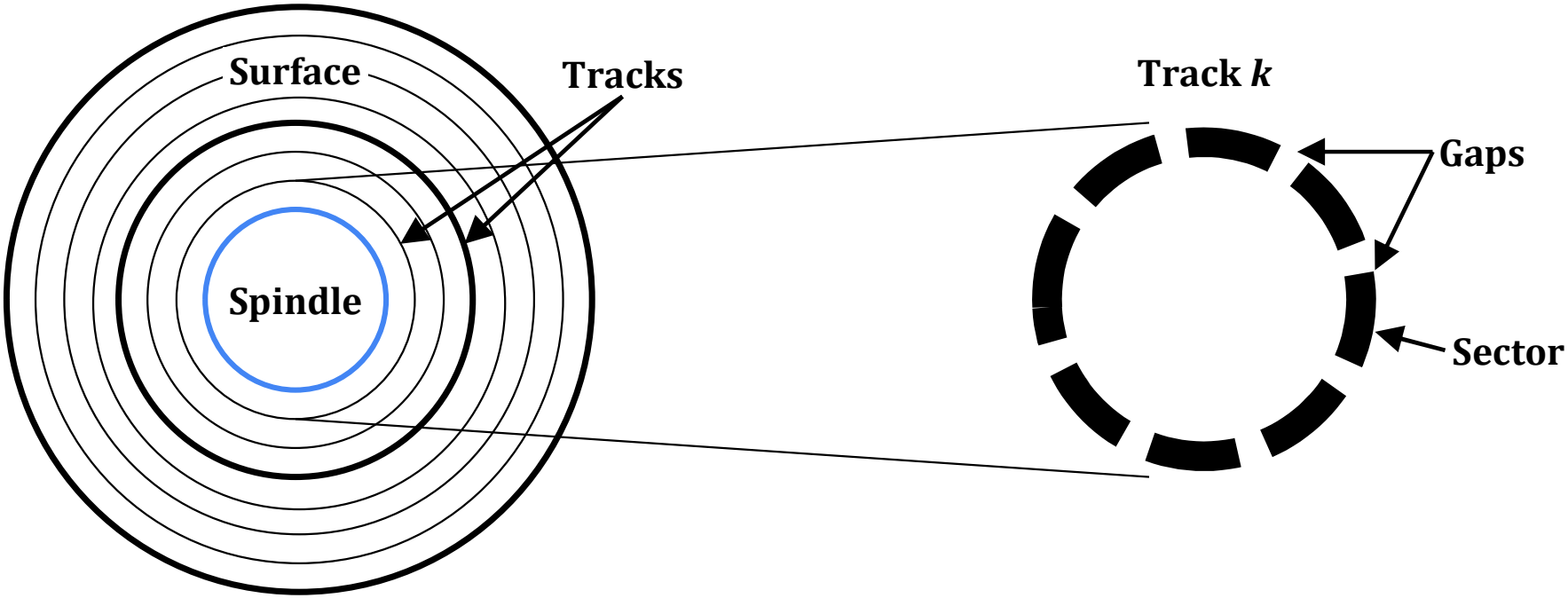


Anatomy of a Hard-Disk Drive (HDD)



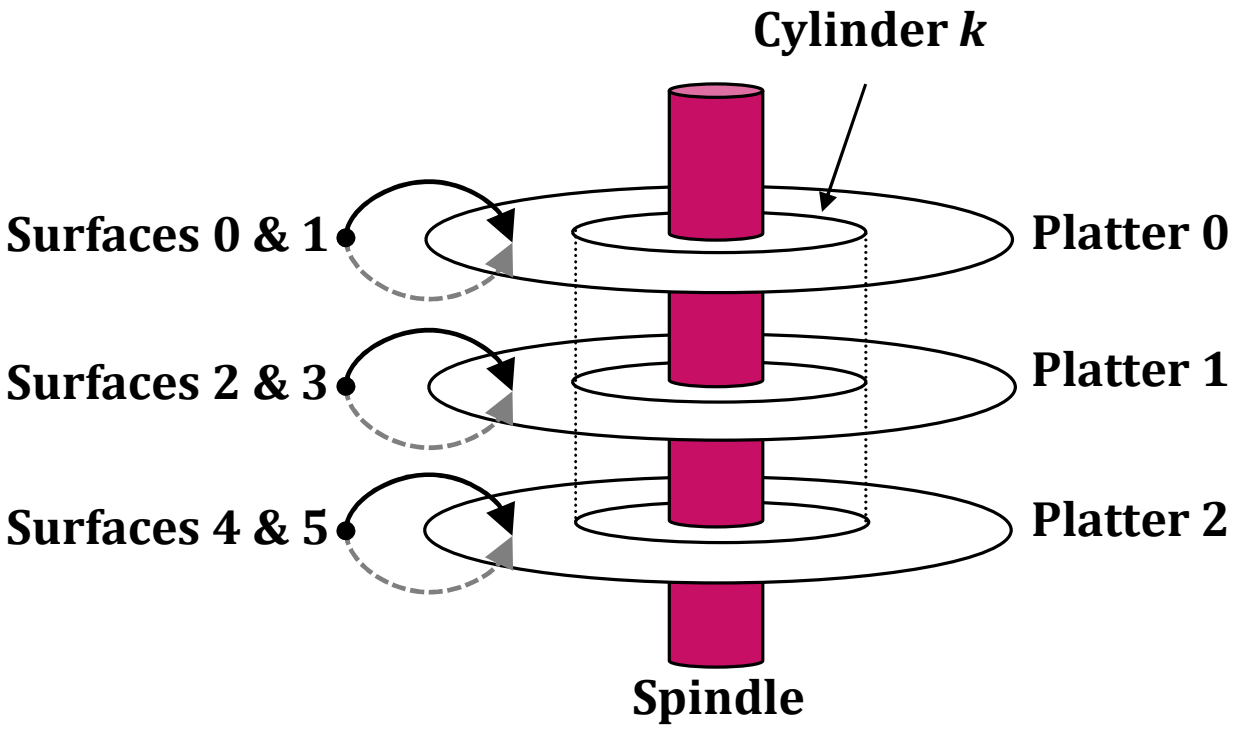
Disk Geometry

- 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)

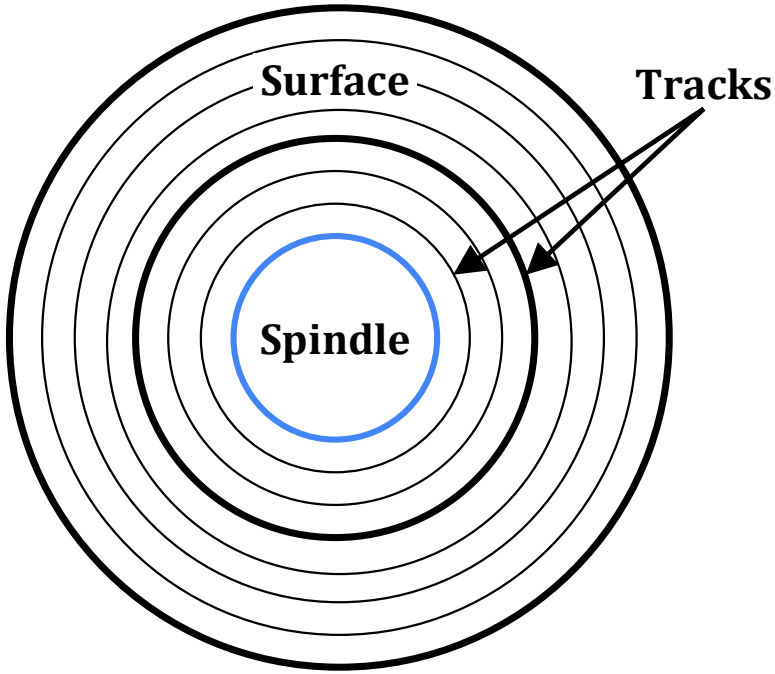
- Aligned tracks form a cylinder



Disk Capacity

- Capacity: maximum number of bits that can be stored
  - Vendors express capacity in units of gigabytes (GB) or terabytes (TB), where 1 GB = 10^9 bytes and 1 TB = 10^{12} bytes

- Capacity is determined by three technology factors:
  - Recording density (bits/in): number of bits that can be squeezed into a l-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



Computing Disk Capacity

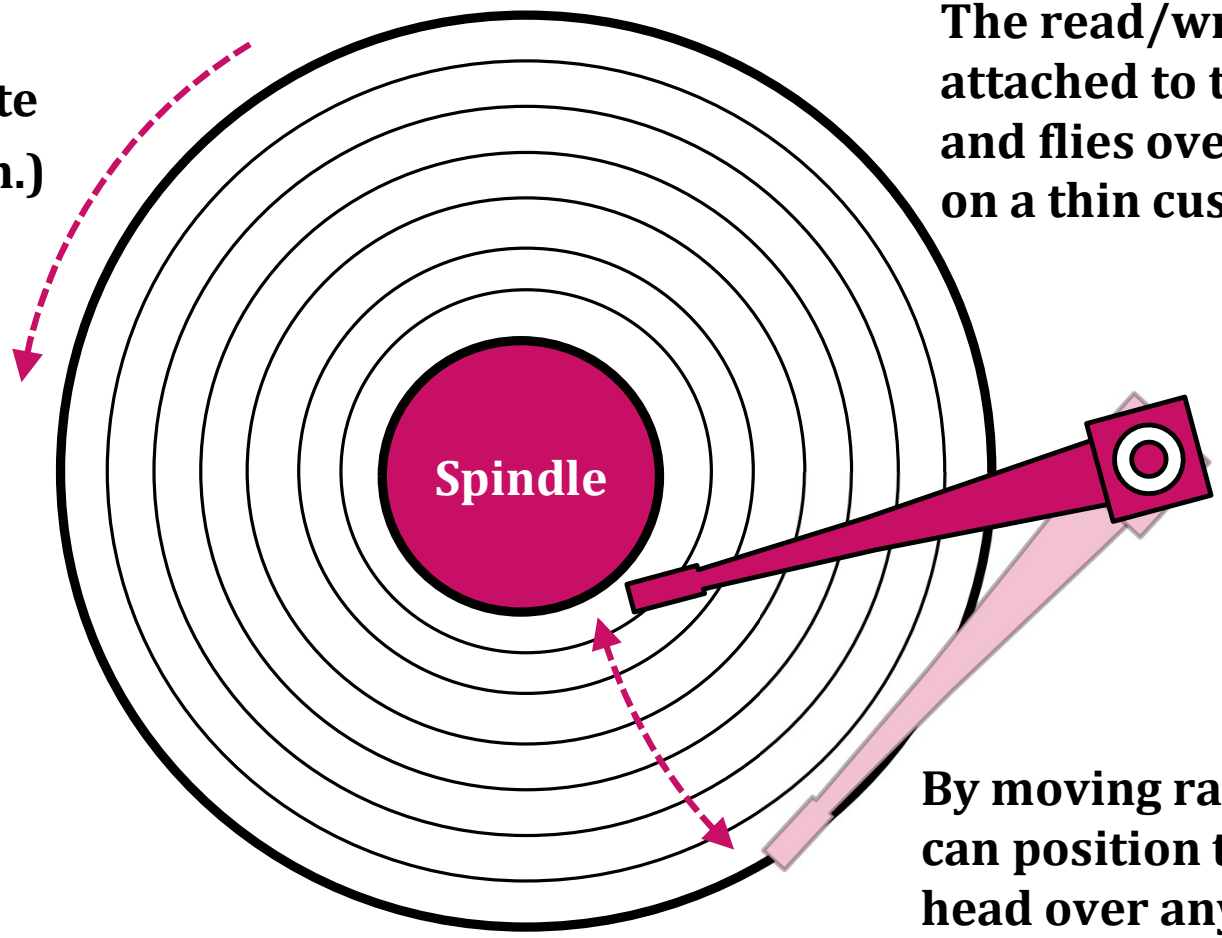
- Capacity = (# bytes/sector) × (avg. # sectors/track) × (# tracks/surface)
  (# surfaces/platter) × (# platters/disk)

- Example disk configuration
  - 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 Bytes
= 30.72 GB

Disk Operation (Single-Platter View)

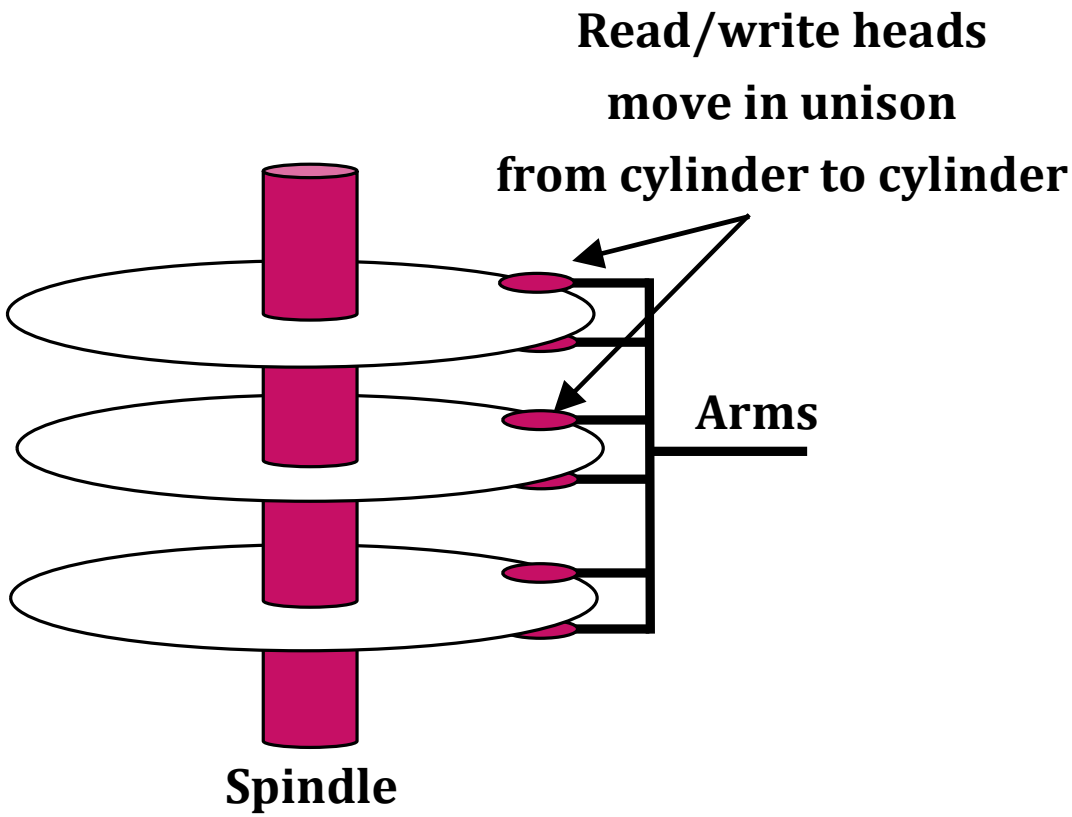
The disk surface spins at a fixed rotational rate (RPM: rotation per min.)



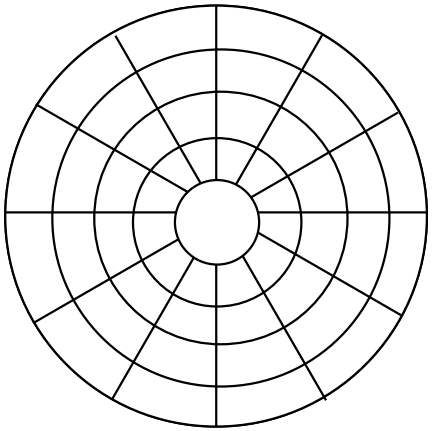
The read/write head is attached to the end of the arm and flies over the disk surface on a thin cushion of air

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

Disk Operation (Multi-Platter View)



Disk Structure: Top View of a Single Platter

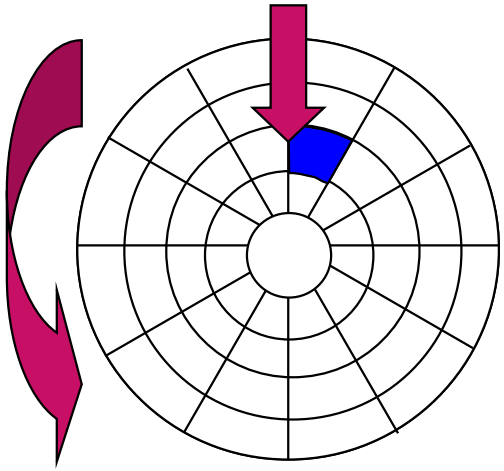


Surface Organized into Tracks

Tracks Divided into Sectors

Disk Access

Head in position above a track

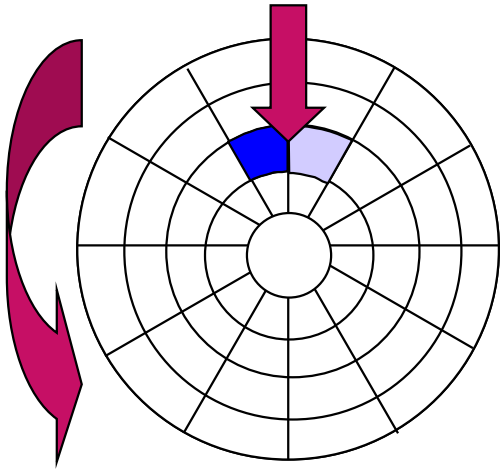


About to read blue sector

Rotation is counter-clockwise

Disk Access

Head in position above a track

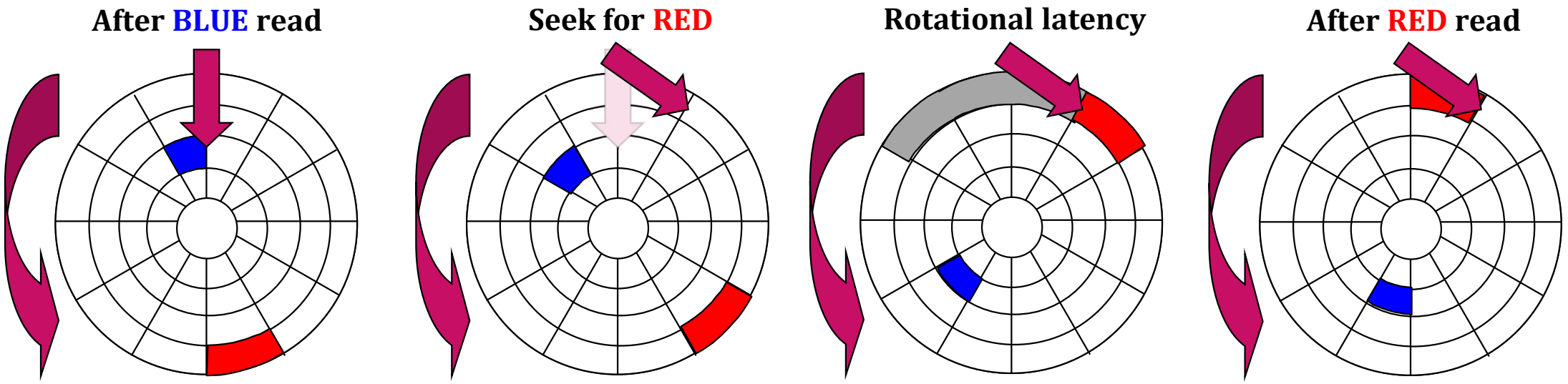


~~*About to read blue sector*~~

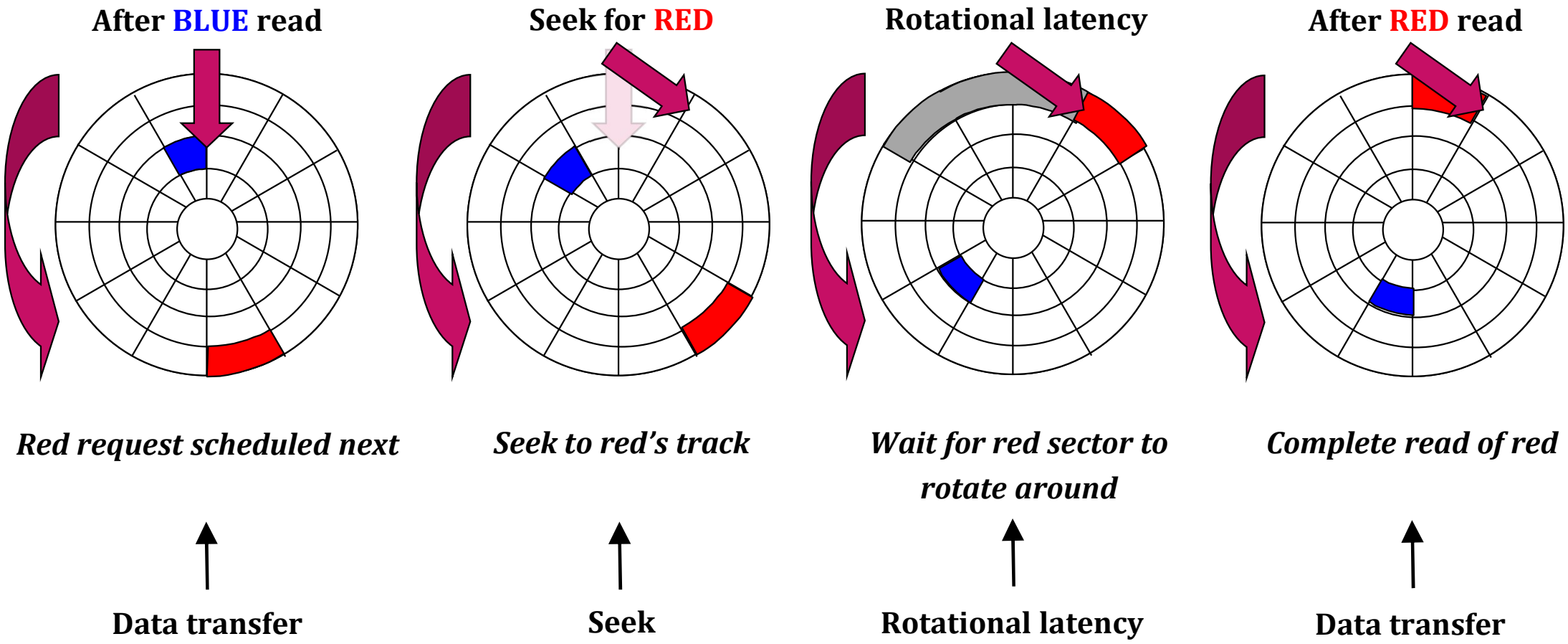
After reading blue sector

Rotation is counter-clockwise

Disk Access: Seek



Disk Access: Seek



Disk Access Time

- Average time to access some target sector approximated by :
  - $T_{ACCESS} = T_{SEEK} + T_{ROTATION} + T_{TRANSFER}$
- Seek time (T_{SEEK})
  - Time to position heads over cylinder containing target sector
  - Typical T_{SEEK} is 3~9 ms
- Rotational latency ($T_{ROTATION}$)
  - Time waiting for first bit of target sector to pass under read/write head
  - $T_{ROTATION} = 1/2 \times 1/RPM \times 60$ sec/1 min
  - Typical RPM = 7,200
- Transfer time ($T_{TRANSFER}$)
  - Time to read the bits in the target sector.
  - $T_{TRANSFER} = 1/RPM \times 1/(avg \# sectors/track) \times 60$ sec/1 min.

Disk Access Time Example

- Given:
  - Rotational rate = 7,200 RPM
  - Average seek time T_{SEEK} = 9 ms
  - Avg # sectors/track = 400
- Derived:
  - $T_{ROTATION}$ = 1/2 × (60 sec/7,200 RPM) × 1,000 ms/sec = 4 ms
  - $T_{TRANSFER}$ = 60/7,200 RPM × 1/400 sec/track × 1,000 ms/sec = 0.02 ms
  - T_{ACCESS} = 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
  - Access time of SRAM: about 4 ns/doubleword, DRAM: about 60 ns
    - Disk is about 40,000 times slower than SRAM,
    - 2,500 times slower than DRAM

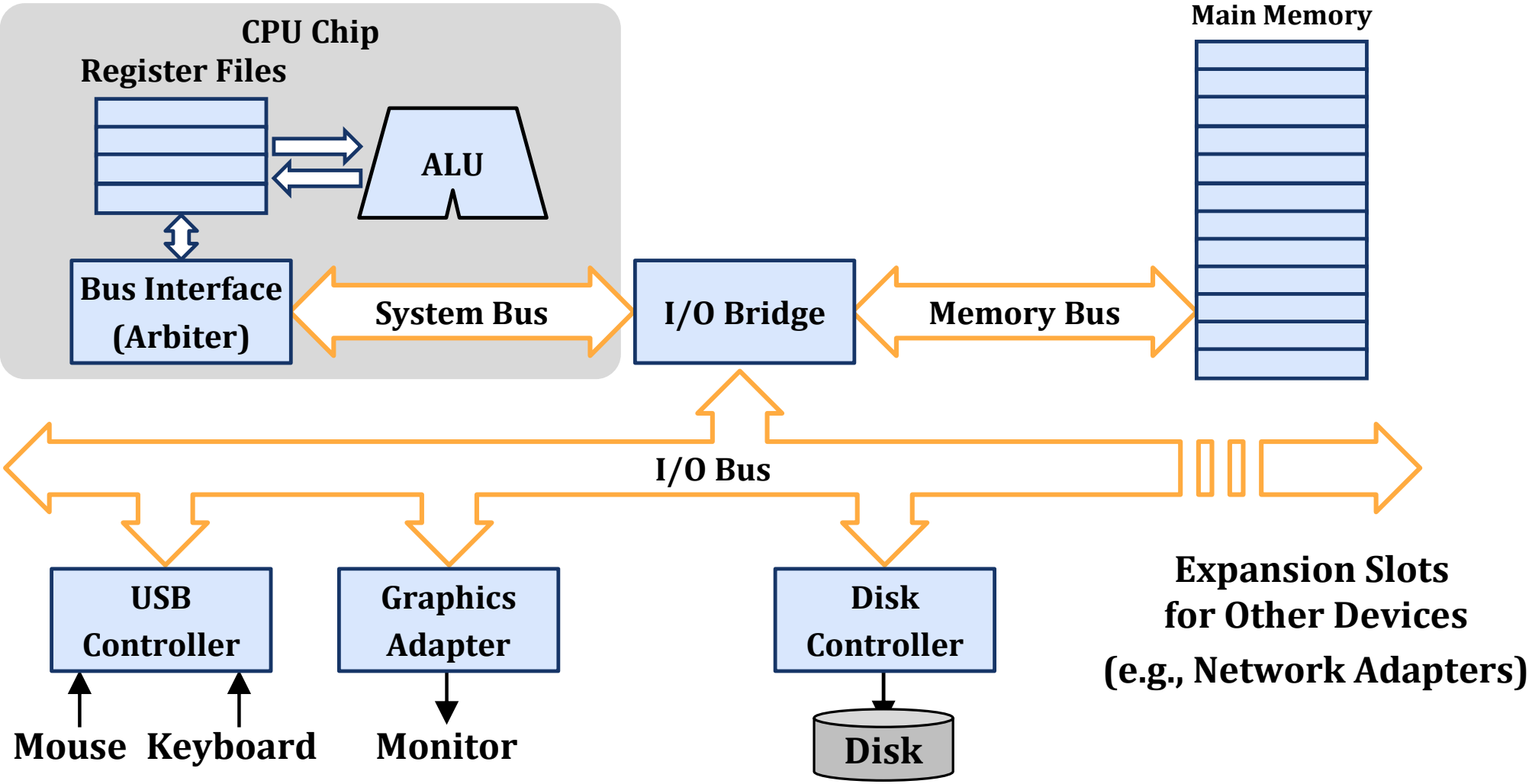
Logical Disk Blocks

- Modern disks present a simple abstract view of the complex sector geometry
  - Available sectors are modeled as a sequence of *B*-sized logical blocks (0, 1, 2, ...)

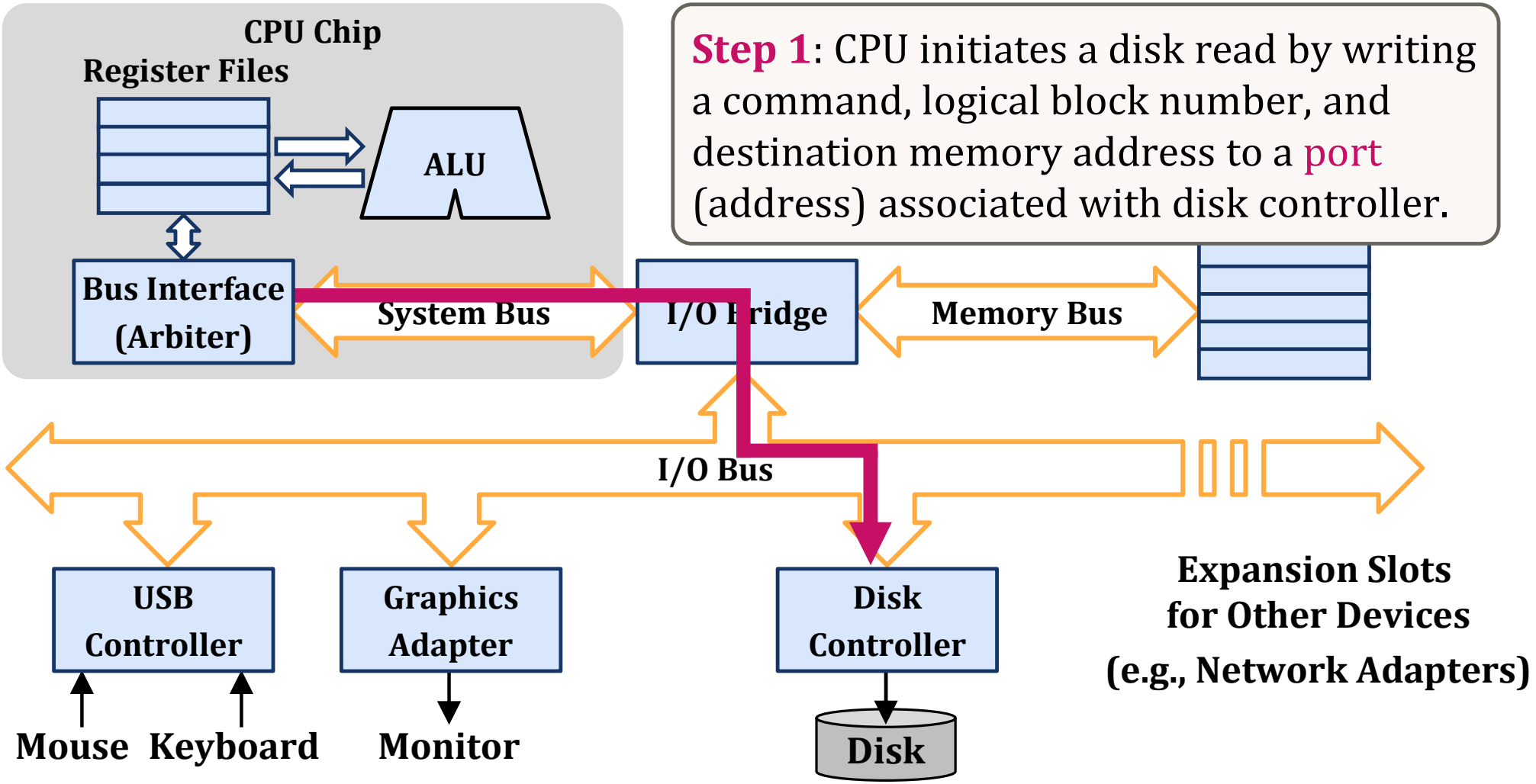
- Mapping between logical blocks and actual (physical) sectors
  - Maintained by hardware/firmware device called disk controller
  - Converts requests for logical blocks into ⟨*surface*, *track*, *sector*⟩ triples

- Allows controller to set aside spare cylinders for each zone
  - Accounts for the difference in formatted capacity and maximum capacity

I/O Bus

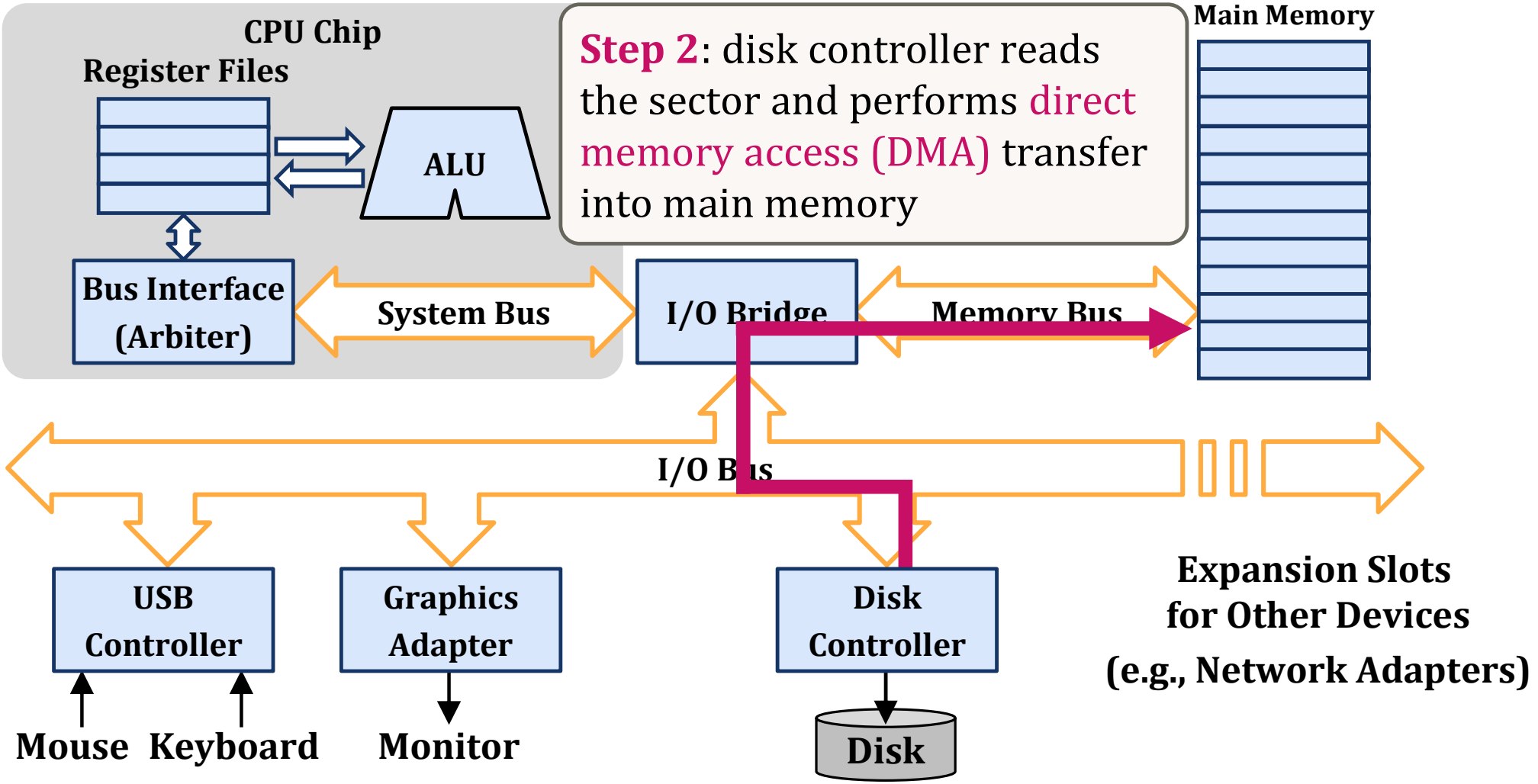

CPU Chip
Register Files
ALU
Bus Interface
(Arbiter)
System Bus
I/O Bridge
Memory Bus
Main Memory
I/O Bus
USB
Controller
Graphics
Adapter
Disk
Controller
Expansion Slots
for Other Devices
(e.g., Network Adapters)
Mouse
Keyboard
Monitor
Disk

Reading a Disk Sector

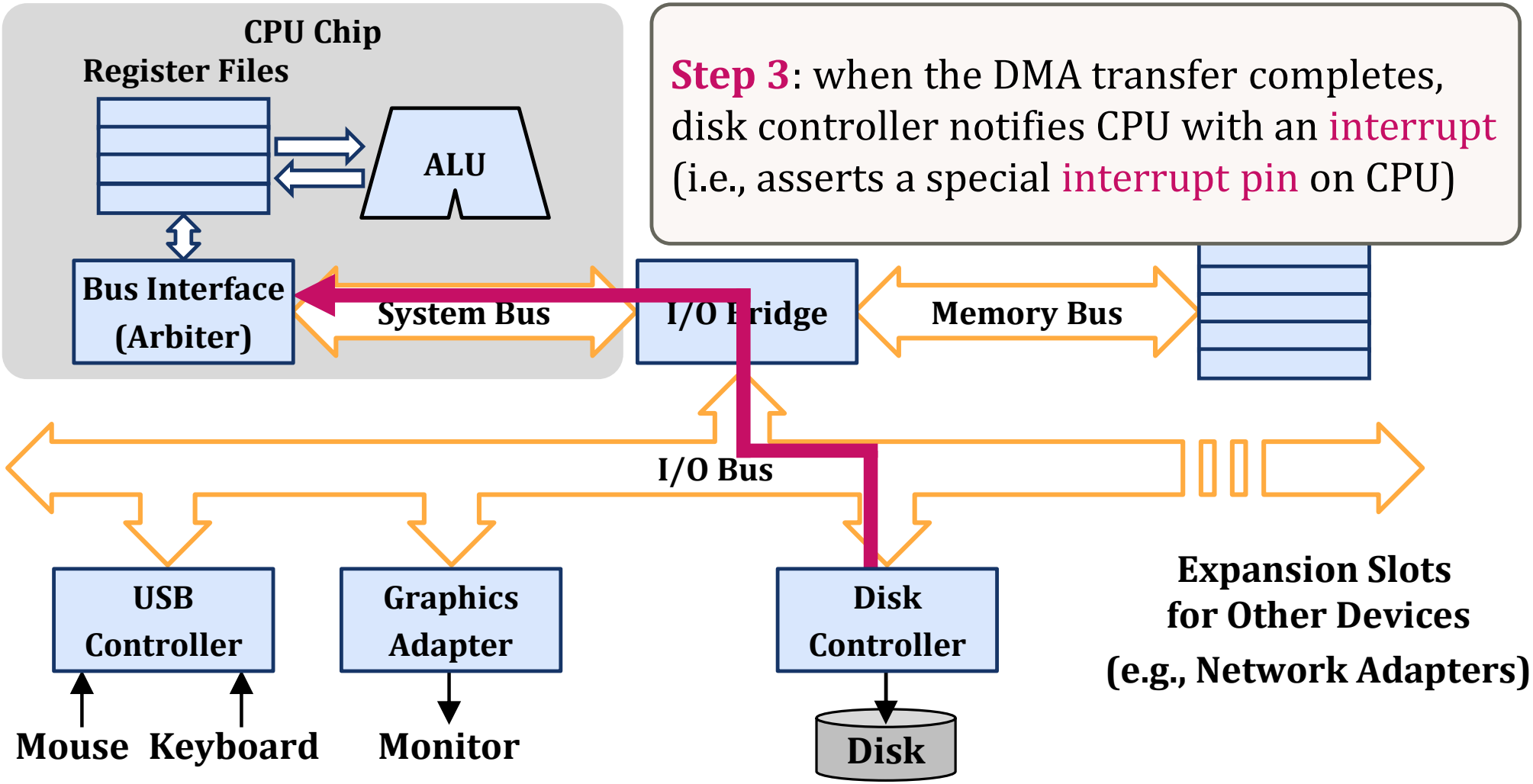


Step 1: CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

Reading a Disk Sector

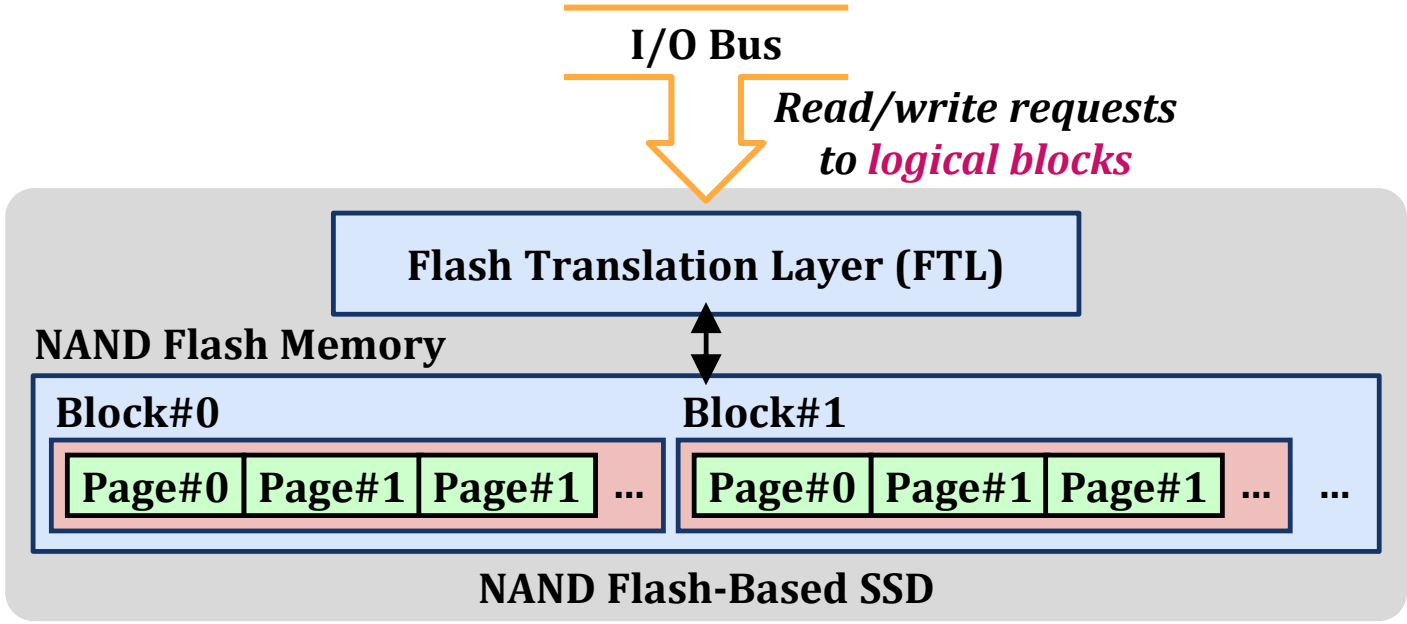


Reading a Disk Sector



Step 3: when the DMA transfer completes, disk controller notifies CPU with an interrupt (i.e., asserts a special interrupt pin on CPU)

Solid-State Drives (SSDs)



- Pages: 4~16 KiB (from 256 B), Blocks: > 1,000 pages (from 32 pages)
- A flash cell must be first erased before writing it (erase-before-write)
- Reads/writes in a page granularity, erase in a block granularity
- A cell has limited endurance: e.g., it cannot reliably store data after 1K writes

SSD Performance Characteristics

- Sequential access faster than random access
  - Common theme in the memory hierarchy
  - The difference is not as large as in HDDs
- Random writes are somewhat slower
  - Erasing a block takes a long time (e.g., 3 ms)
  - Erasing a block requires all valid pages to be copied to a new block
  - In earlier SSDs, the read/write gap was much larger

Sequential read BW	**550 MB/s**	**Sequential write BW**	**470 MB/s**
Random read BW	**365 MB/s**	**Random write BW**	**303 MB/s**
Avg. read latency	**50 us**	**Avg. seq write latency**	**60 us**

Source: Intel SSD 730 product specification

SSD Performance Characteristics

- Sequential access faster than random access
  - Common theme in the memory hierarchy
  - The difference is not as large as in HDDs

- Random writes are somewhat slower
  - Erasing a block takes a long time (e.g., 3 ms)
  - Erasing a block requires all valid pages to be copied to a new block
  - In earlier SSDs, the read/write gap was much larger

Sequential read BW	**7,450 MB/s**	**Sequential write BW**	**6,900 MB/s**
Random read BW	**6,400 MB/s**	**Random write BW**	**6,200 MB/s**
Avg. read latency	**45 us**	**Avg. seq write latency**	**1.25 us**

Source: Samsung SSD 990 Pro product specification

SSD Tradeoffs vs HDDs

- Advantages
  - No moving parts → faster, less power, more rugged
- Disadvantages
  - Have the potential to wear out
    - Many mitigations (e.g., wear-leveling logic) in flash translation layer
    - e.g., Intel X25 guarantees 1,000 terabytes (10^{15} bytes) written (TBW)
  - ~~In 2010, about 100× more expensive per byte~~
  - In 2023, about 2~3× more expensive per byte
- Applications
  - ~~MP3 players~~, smart phones, laptops
  - ~~Beginning to appear~~ Rapidly replacing HDDs in desktops and servers

Summary on I/O

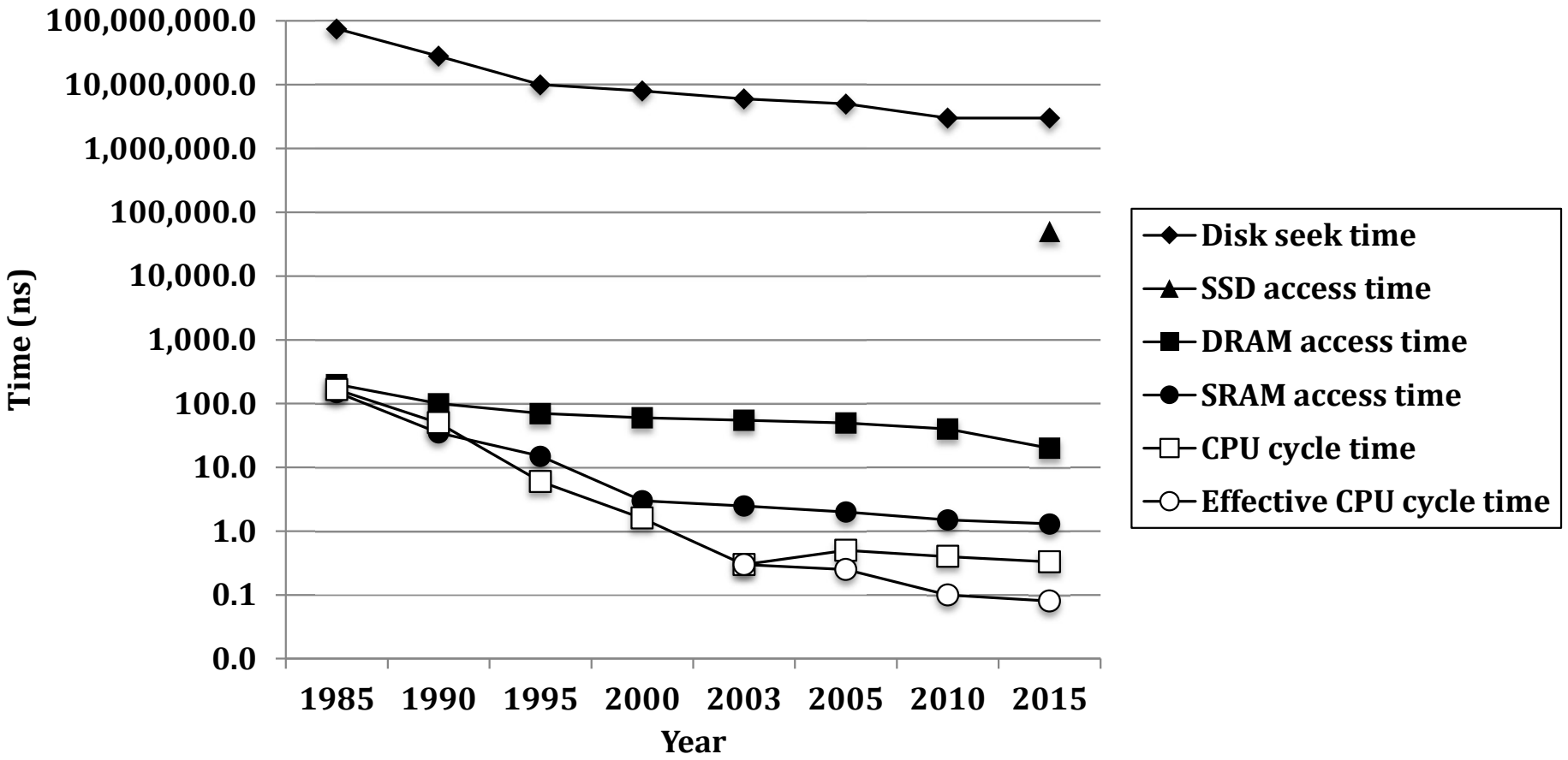
- Buses
  - System bus, memory bus, I/O bus
  - Who controls the bus?
    - CPU: old design
    - Bus arbiter
  - Data transfer between I/O device and Memory
    - CPU-controlled bus : cycle stealing or direct memory access
    - Bus arbiter: direct memory access, packet-based data transfer (PCI-e bus)
- I/O control method
  - Memory mapped I/O: one consolidated address for memory and I/O devices
  - Channel I/O: separate address space for memory and I/O
  - How to know: existence of special I/O instructions

Lecture Agenda

- Storage Technologies and Trends
- Locality of Reference
- Caching in the Memory Hierarchy

CPU-Memory Performance Gap

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



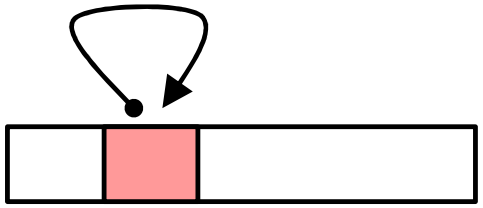
Locality to the Rescue!

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

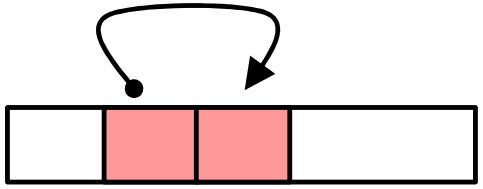
Locality

- 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;
}
```

Hint: array layout is row-major order

Answer: yes

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;
}
```

Answer: no, unless `M` is very small

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: can you permute the loops so that the function scans 3D 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;
}
```

Answer:
Make `j` in the inner loop

Memory Hierarchy

- Some fundamental and enduring properties of hardware and software:
  - Fast storage technologies have higher cost per bit and require more power (heat)
  - The gap between CPU and main memory speeds is widening
  - Well-written programs tend to exhibit good locality

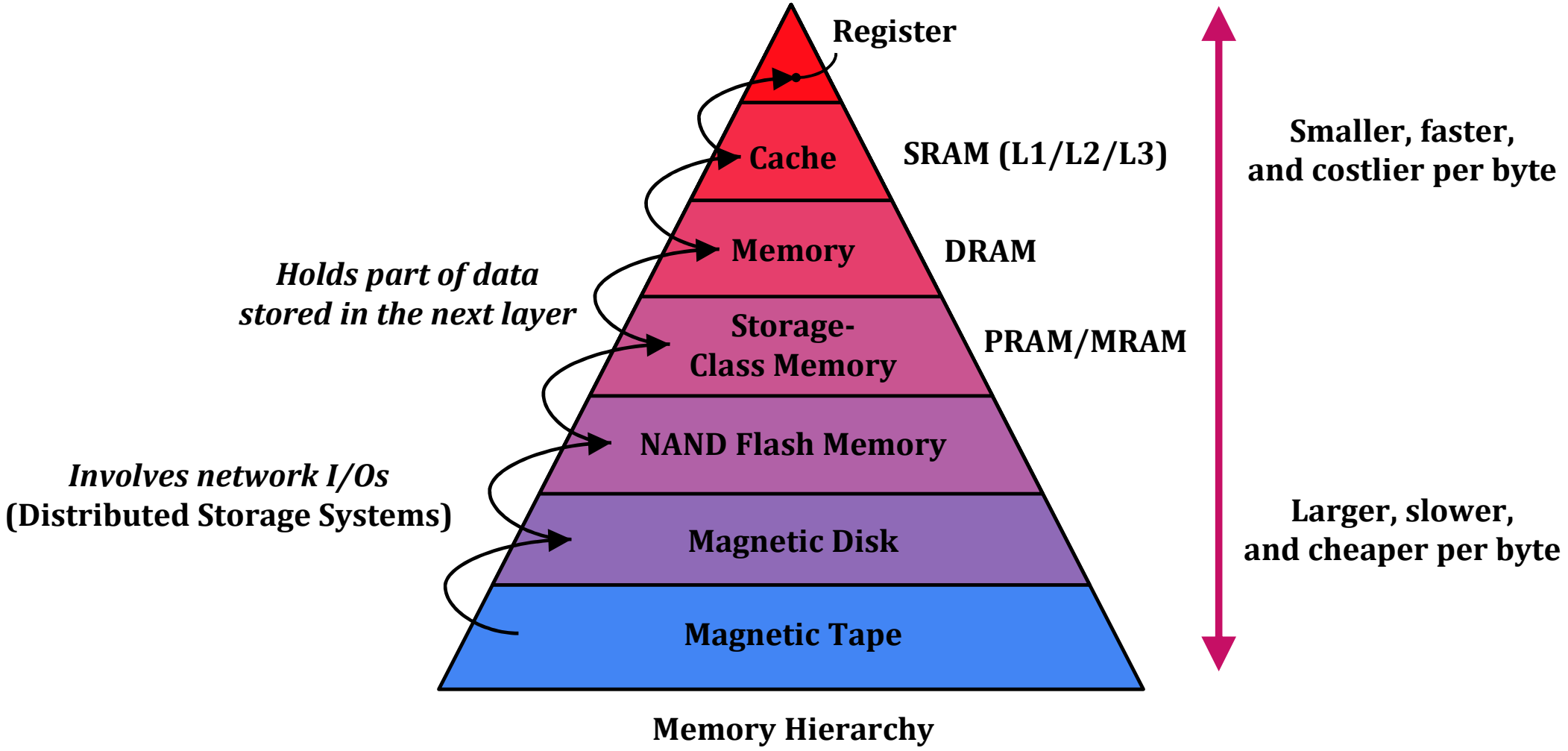
- These fundamental properties complement each other beautifully

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

Lecture Agenda

- Storage Technologies and Trends
- Locality of Reference
- Caching in the Memory Hierarchy

Example Memory Hierarchy

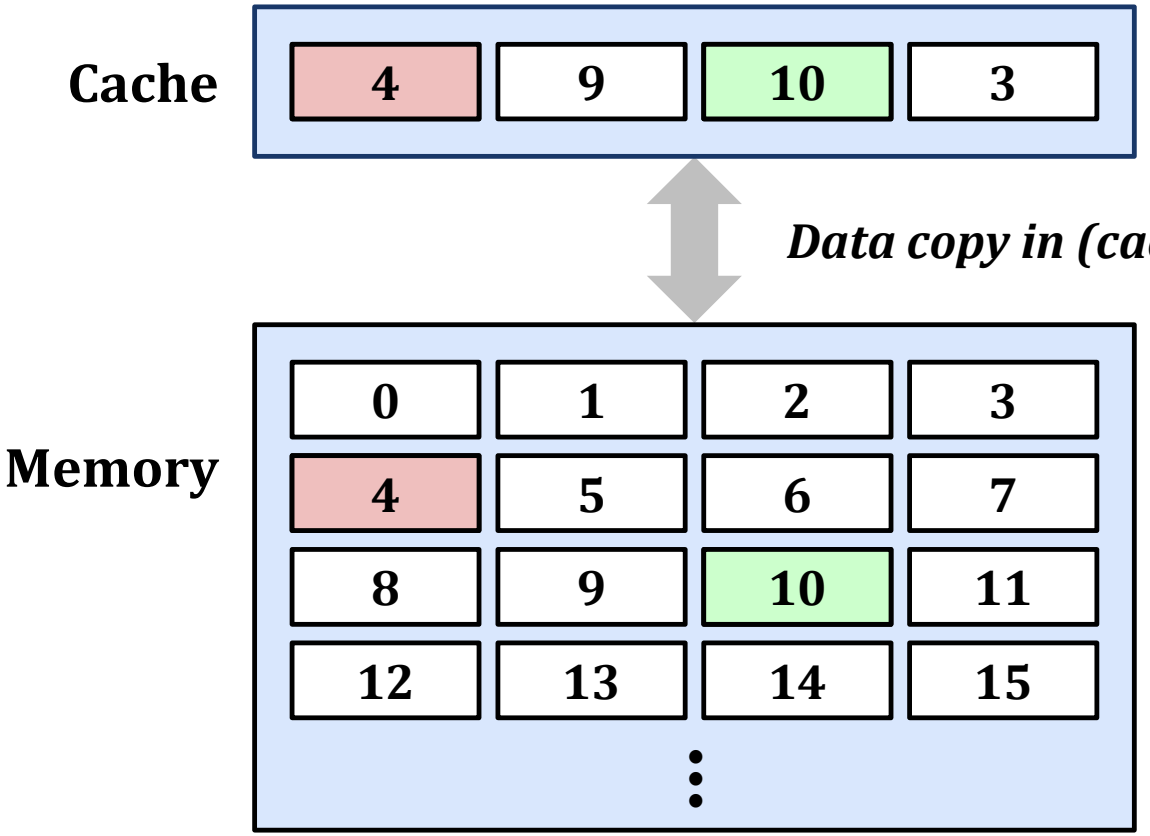


Memory Hierarchy

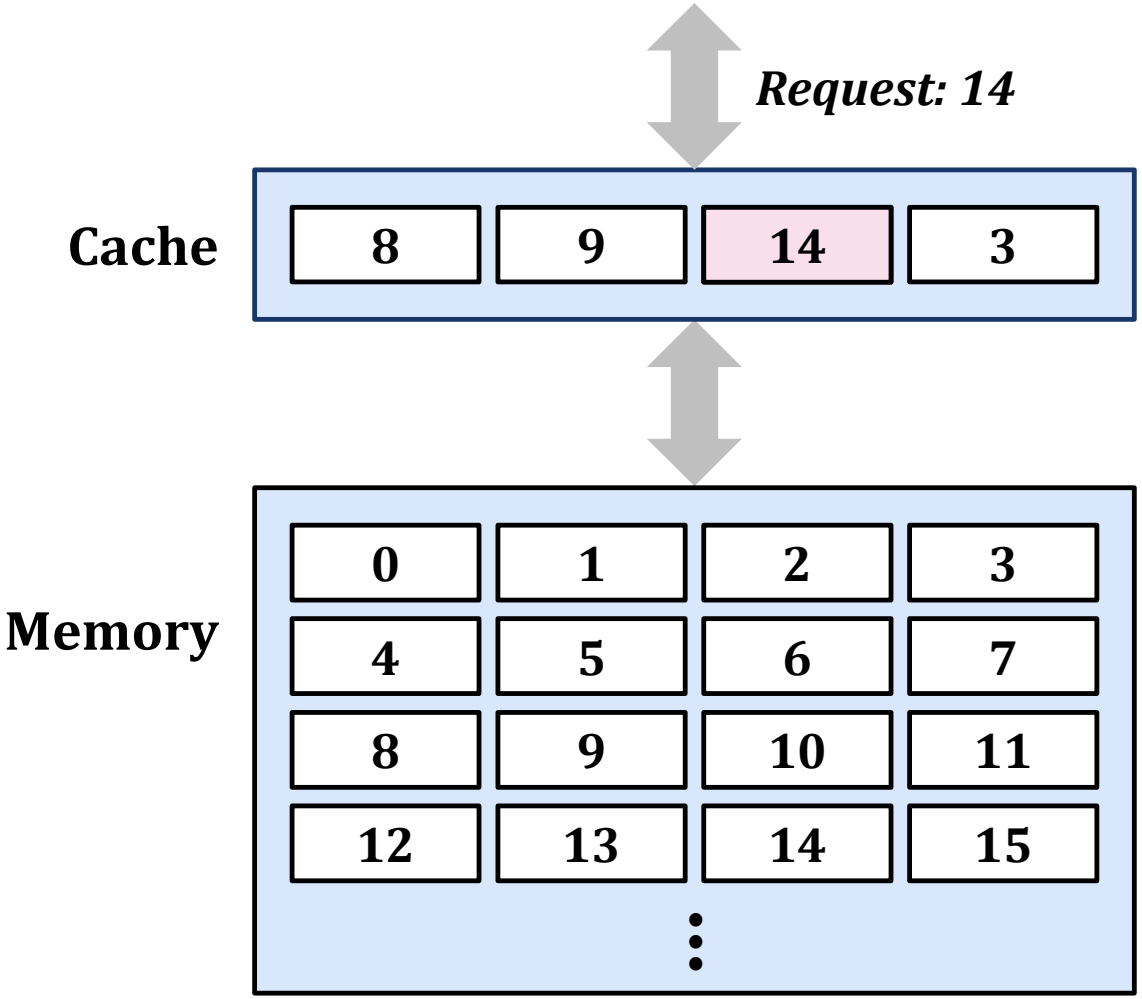
Caches

- 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 does it work?
  - Due to locality, programs tend to access the data at level *k* more frequently than 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



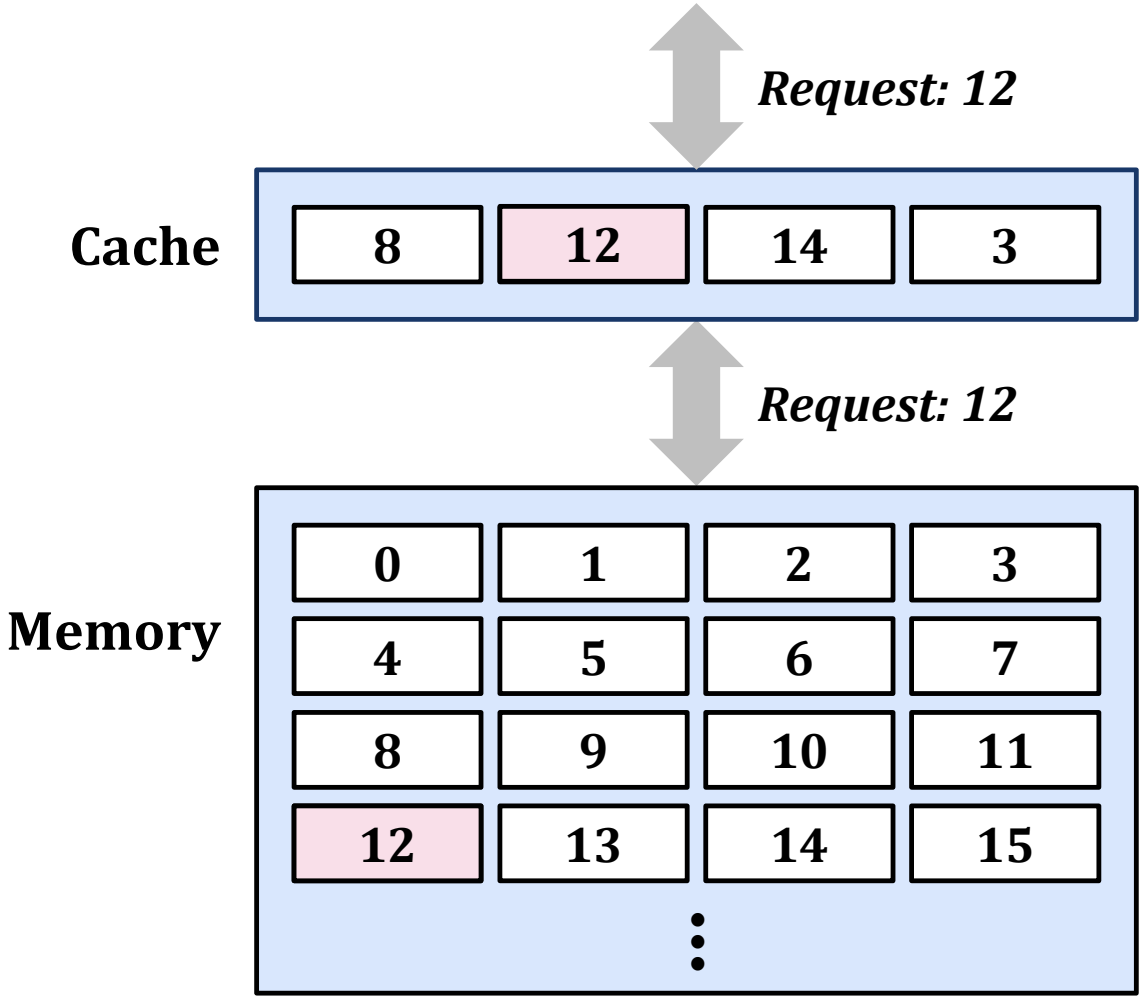
General Cache Concepts: Hit



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: **Miss!**

Block *B* is fetched from memory

Block *B* is stored in cache

Placement policy:
determines where b goes

Replacement policy:
determines which block gets evicted (victim)

General Caching Concepts: 3 Types of Cache Misses

- Compulsory miss (or cold miss)
  - Occurs because any cache starts empty: the first reference to the block
- Capcity miss
  - Occurs when the set of active cache blocks (working set) is larger than the cache
- Conflict miss
  - Occurs when the level-k cache is large enough, but multiple data objects all map to the same level-k block
    - Most caches limit blocks at level (k+1) to a small subset (sometimes a singleton) of the block positions at level k – set associative cache
  - e.g., Referencing blocks 0 → 8 → 0 → 8 → 0 → 8 → … would miss every time if block i at level (k+1) must be placed in block (i mod 4) at level (k+1)

Examples of Caching in the Mem. 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 MMU
L1 Cache	64-Bytes Blocks	On-Chip L1	4	Hardware
L2 Cache	64-Bytes Blocks	On-Chip L2	10	Hardware
Virtual Memory	4-KB Pages	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	NFS Client
Browser Cache	Web Pages	Local Disk	10,000,000	Web Browser
Web Cache	Web Pages	Remote Server Disk	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 hierarchy based on caching closes the gap by exploiting locality

[CSED211] Introduction to Computer Software Systems

Lecture 10: The Memory Hierarchy

Prof. Jisung Park

2023.11.06