Memory Systems
Lec08 – Memory Hierarchy

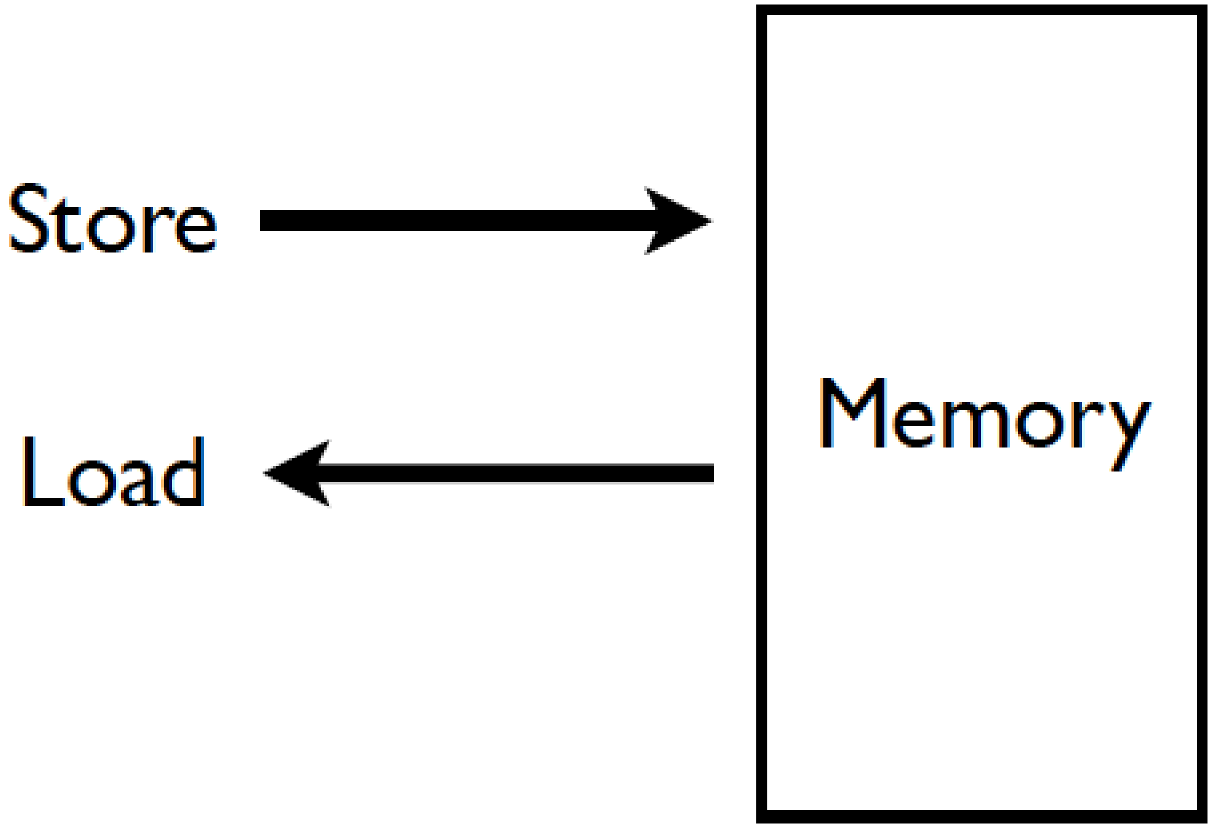
Chin-Fu Nien (粘儆夫)

The content of this part is mainly from:
Randal E. Bryant and David R. O'Hallaron, “Computer Systems: A Programmer's Perspective,” 3/e.

(本節內容改自Prof. Randal E. Bryant and David R. O’Hallaron 8th Lectures課程講義)

Module 2: System & Software (con’t)

Memory (Programmer's View)



Abstraction: Virtual vs. Physical Memory

- Programmer sees virtual memory
  - Can assume the memory is "infinite"
- Reality: Physical memory size is much smaller than what the programmer assumes
- The system (system software + hardware, cooperatively) maps virtual memory addresses to physical memory
  - The system automatically manages the physical memory space transparently to the programmer

+ Programmer does not need to know the physical size of memory nor manage it → A small physical memory can appear as a huge one to the programmer → Life is easier for the programmer

-- More complex system software and architecture

A classic example of the programmer/(micro)architect tradeoff

(Physical) Memory System

- You need a larger level of storage to manage a small amount of physical memory automatically
  $\rightarrow$ Physical memory has a backing store: disk

- We will first start with the physical memory system

- For now, ignore the virtual$\rightarrow$physical indirection
  - We will get back to it later

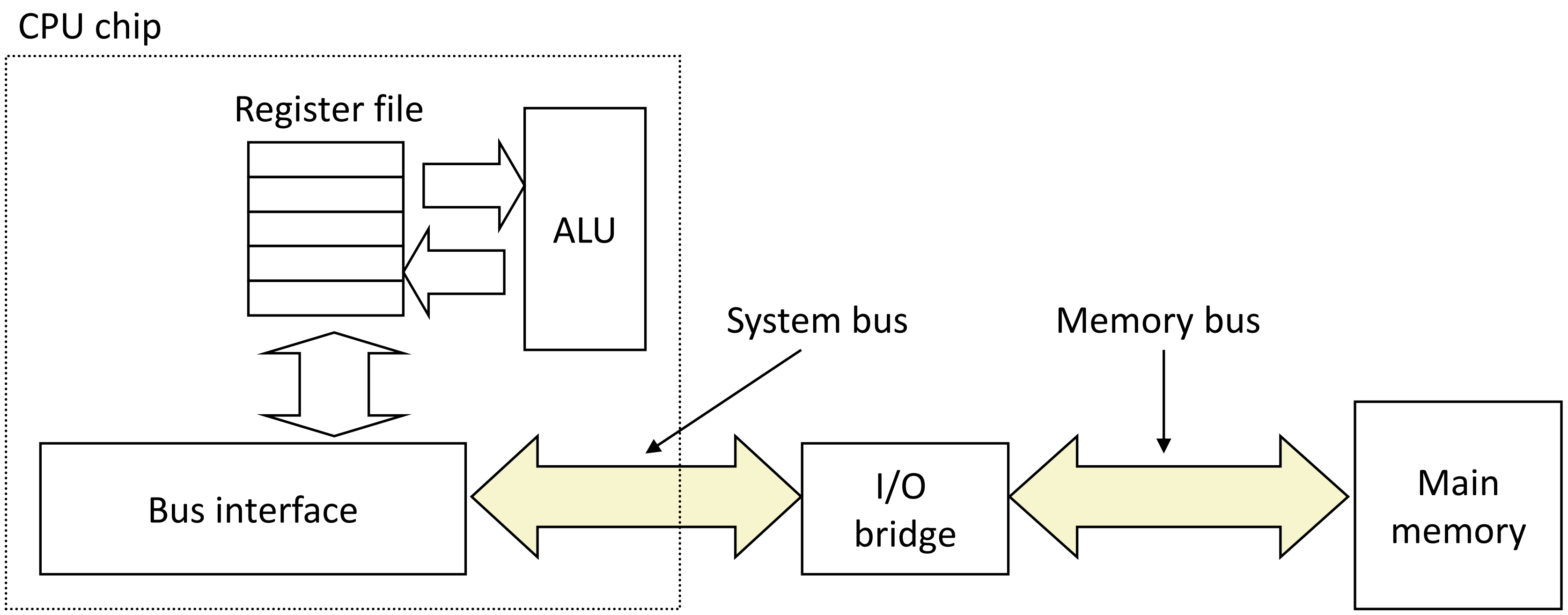
How Can We Store Data?

- Flip-Flops (or Latches)
  - Very fast, parallel access
  - Very expensive (one bit costs tens of transistors)
- Static RAM (we will describe them in a moment)
  - Relatively fast, only one data word at a time
  - Expensive (one bit costs 6+ transistors)
- Dynamic RAM (we will describe them in a moment)
  - Slower, one data word at a time, reading destroys content (refresh), needs special process for manufacturing
  - Cheap (one bit costs only one transistor plus one capacitor)
- Other storage technology (flash memory, hard disk, tape)
  - Much slower, access takes a long time, non-volatile
  - Very cheap (one transistor stores many bits or no transistors involved)

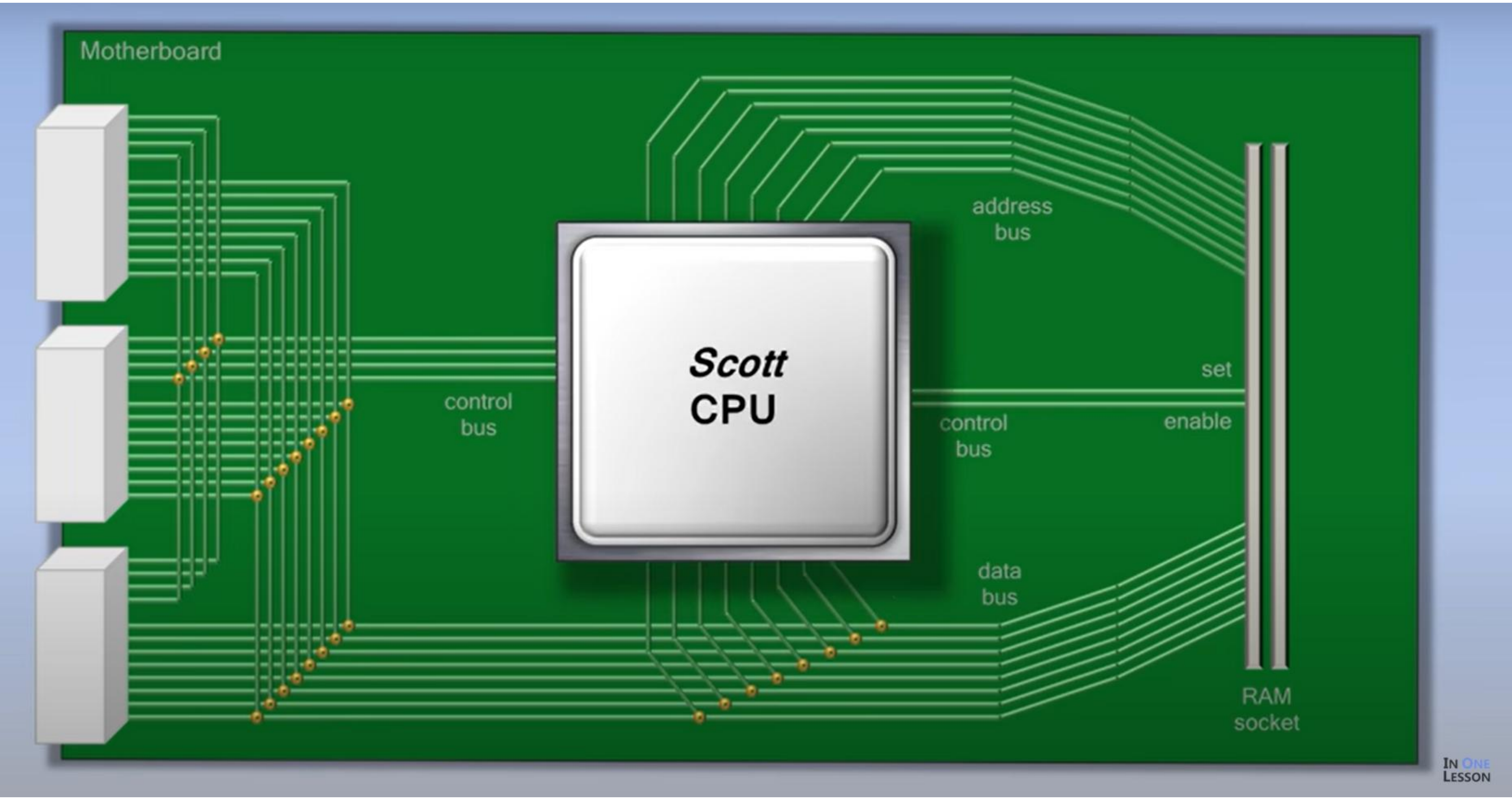
Main Memory System

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.



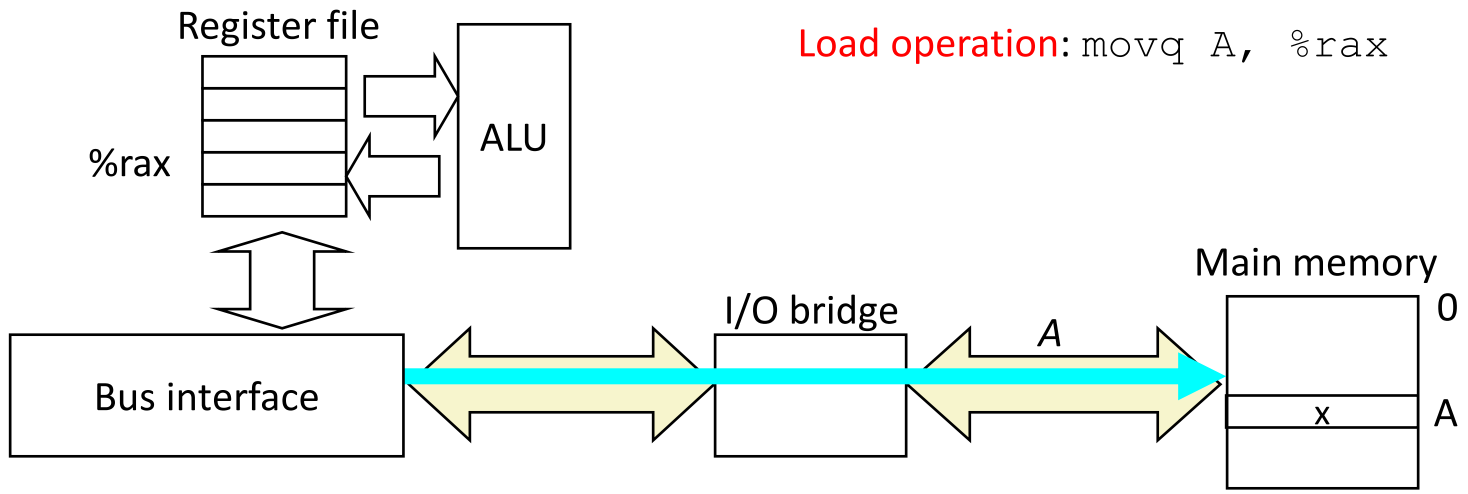
A Modern CPU Layout



[Picture Source] https://www.youtube.com/watch?v=cNN_tTXABUA

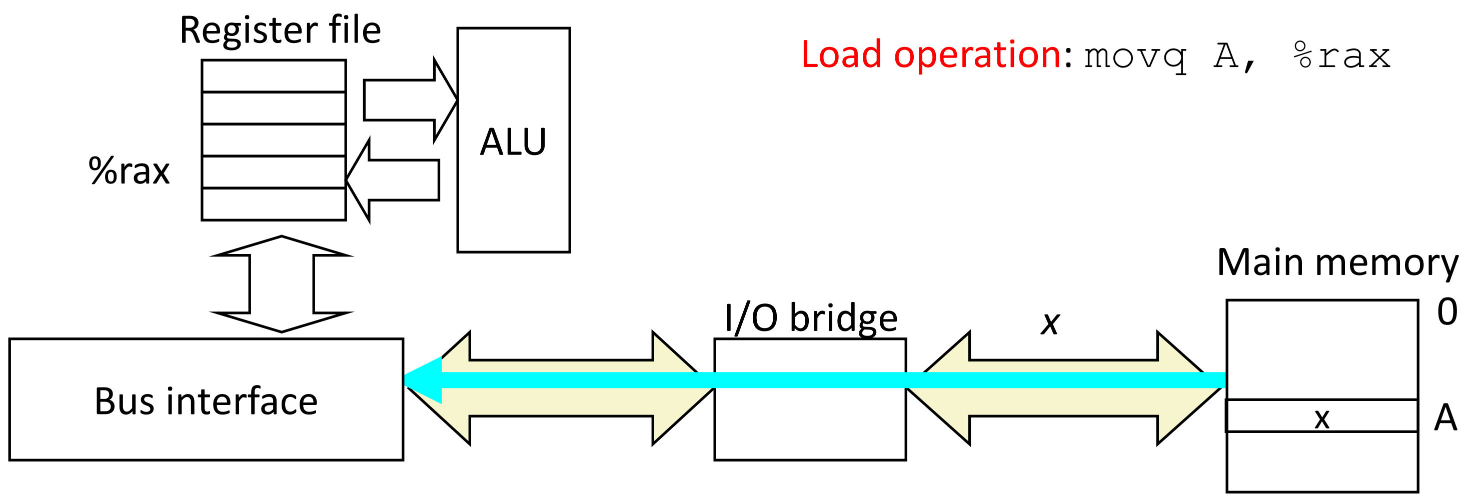
Memory Read Transaction (1)

- CPU places address A on the memory bus.



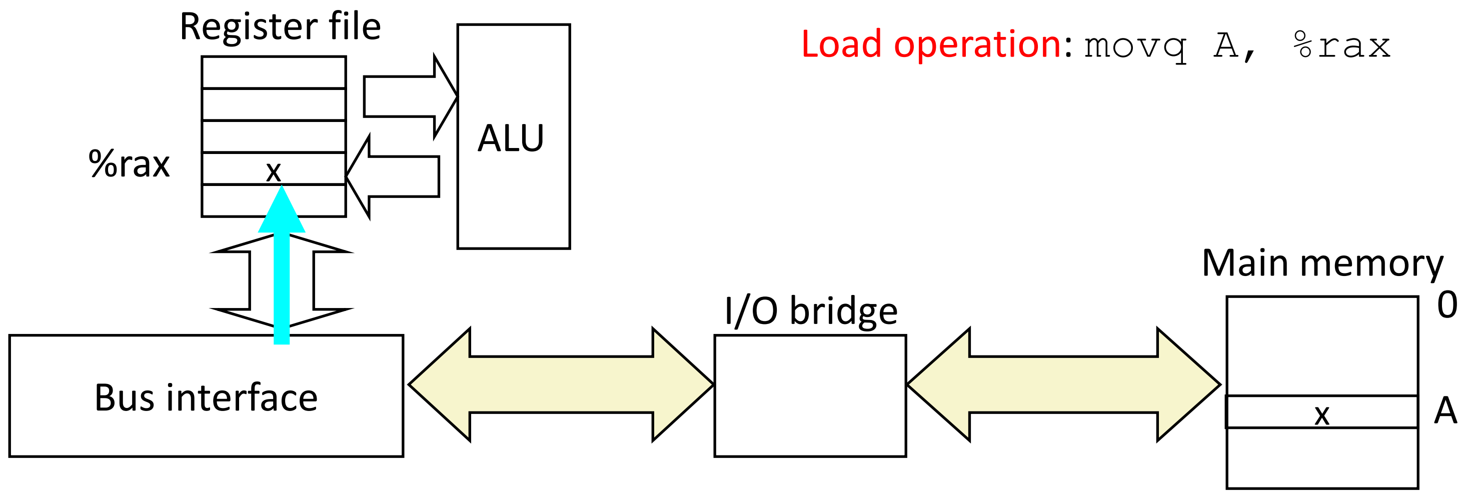
Memory Read Transaction (2)

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

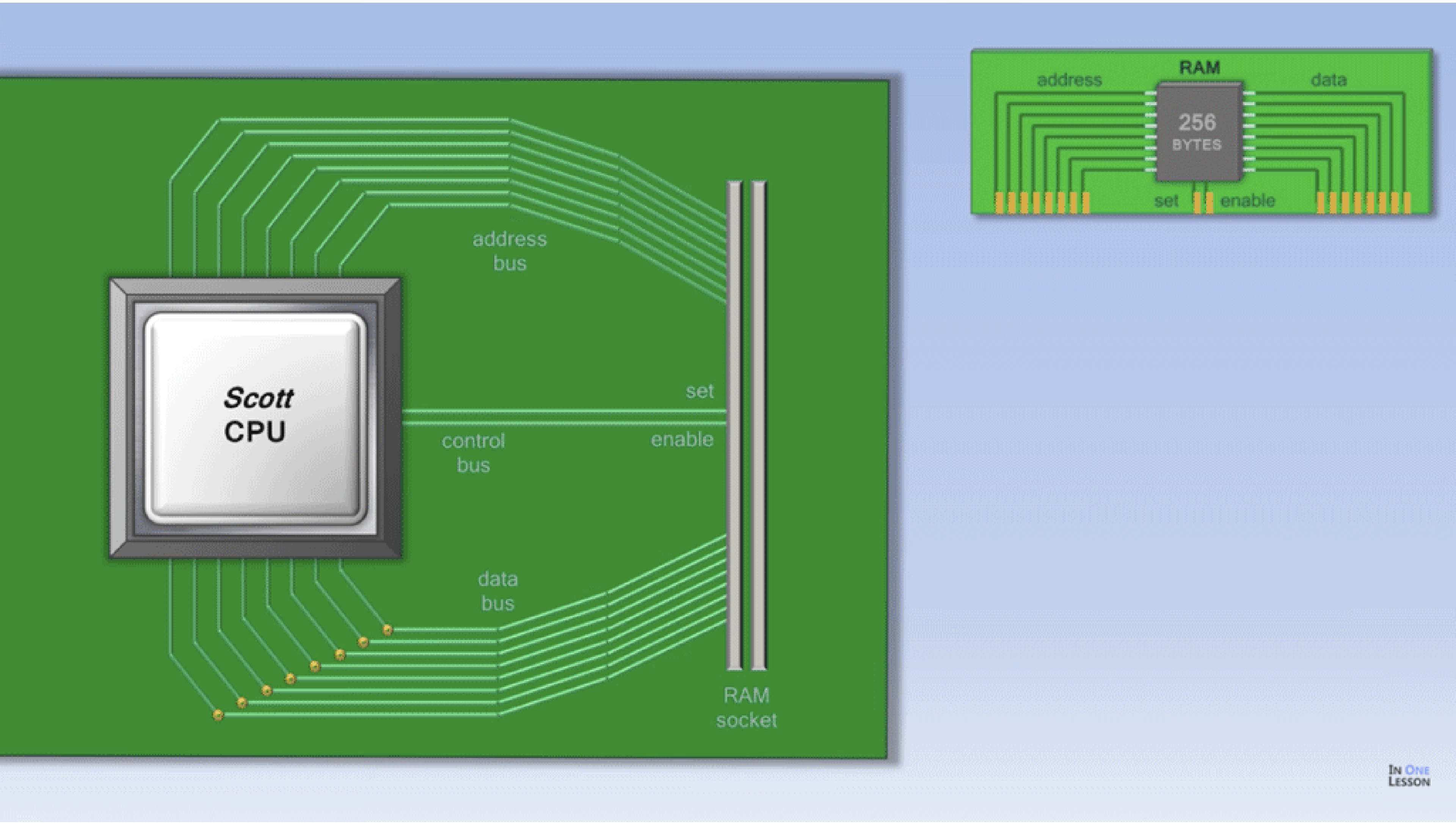


Memory Read Transaction (3)

- CPU read word x from the bus and copies it into register %rax.



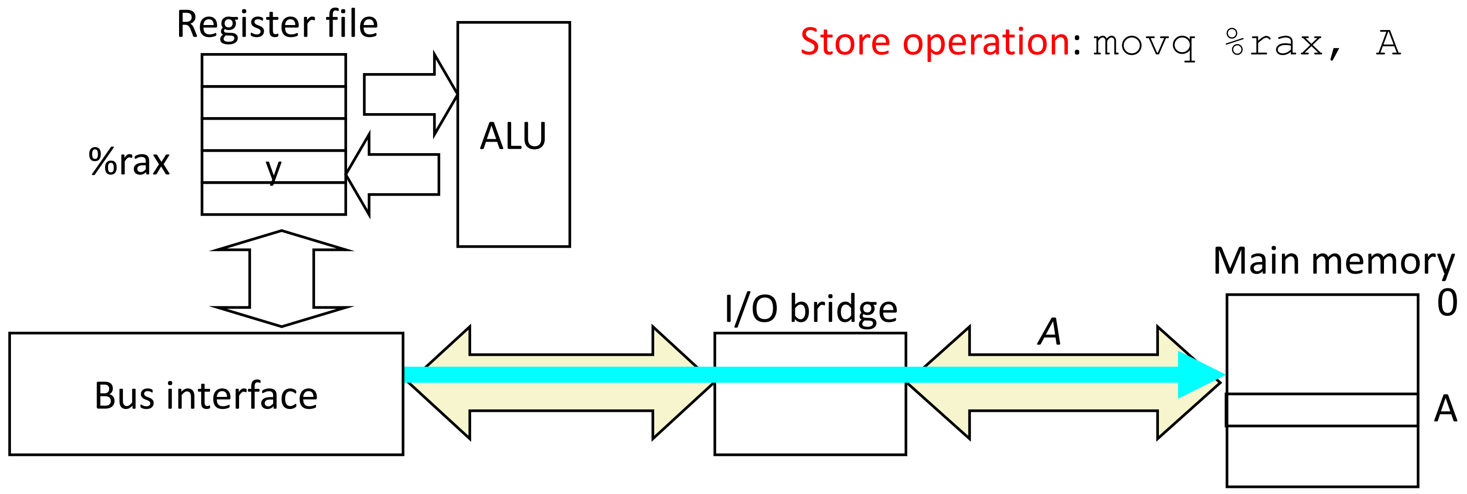
Animation for CPU Read



[Picture Source] https://notes.andywu.tw/2017/cpu、ram、bus之間的互動圖解/

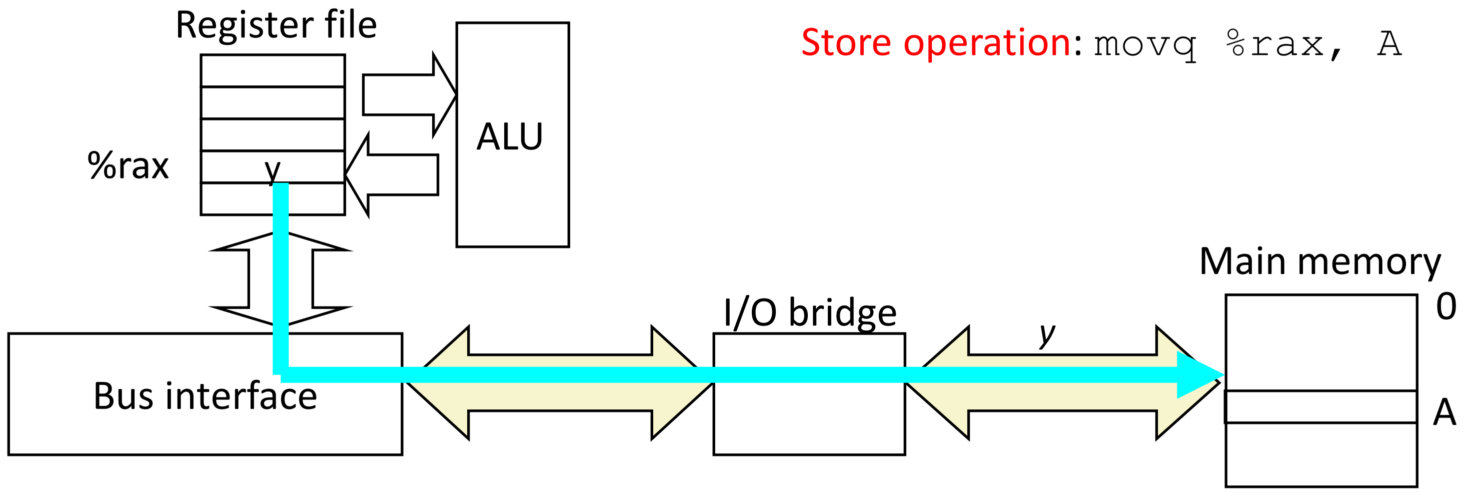
Memory Write Transaction (1)

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



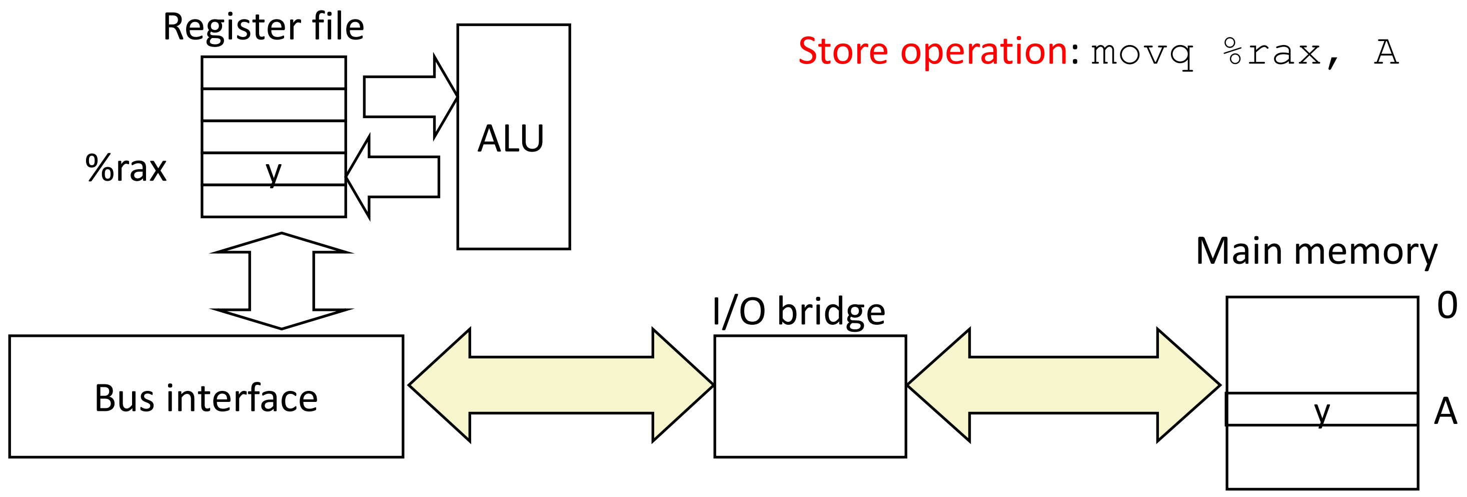
Memory Write Transaction (2)

- CPU places data word y on the bus.



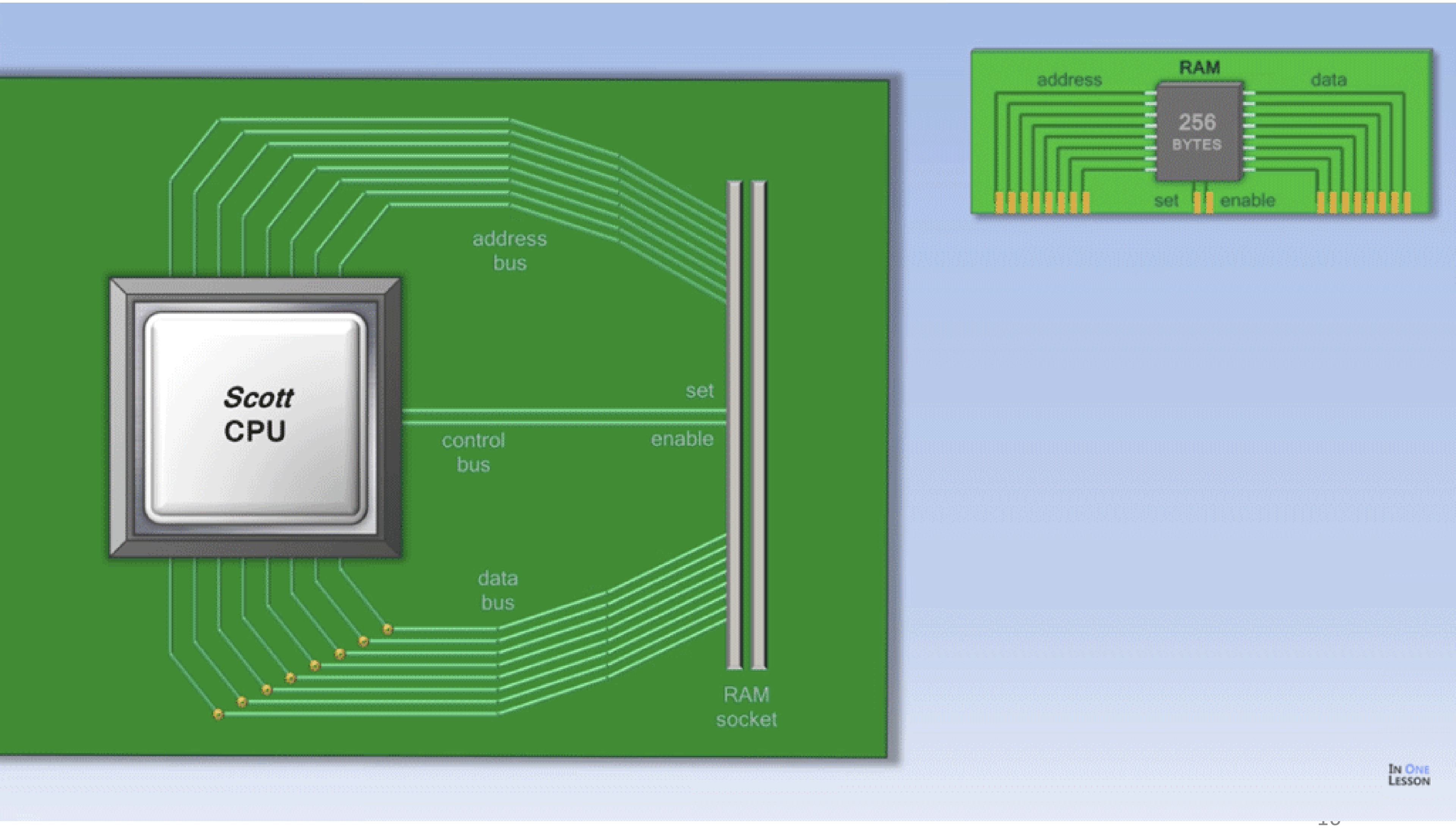
Memory Write Transaction (3)

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



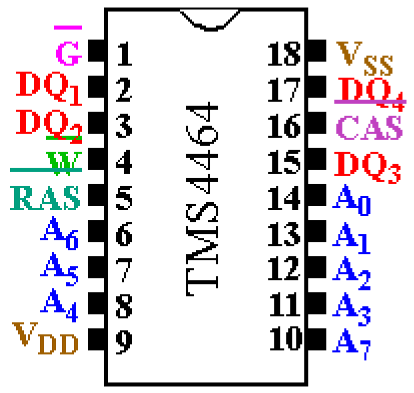
Store operation: `movq %rax, A`

Animation for CPU Write



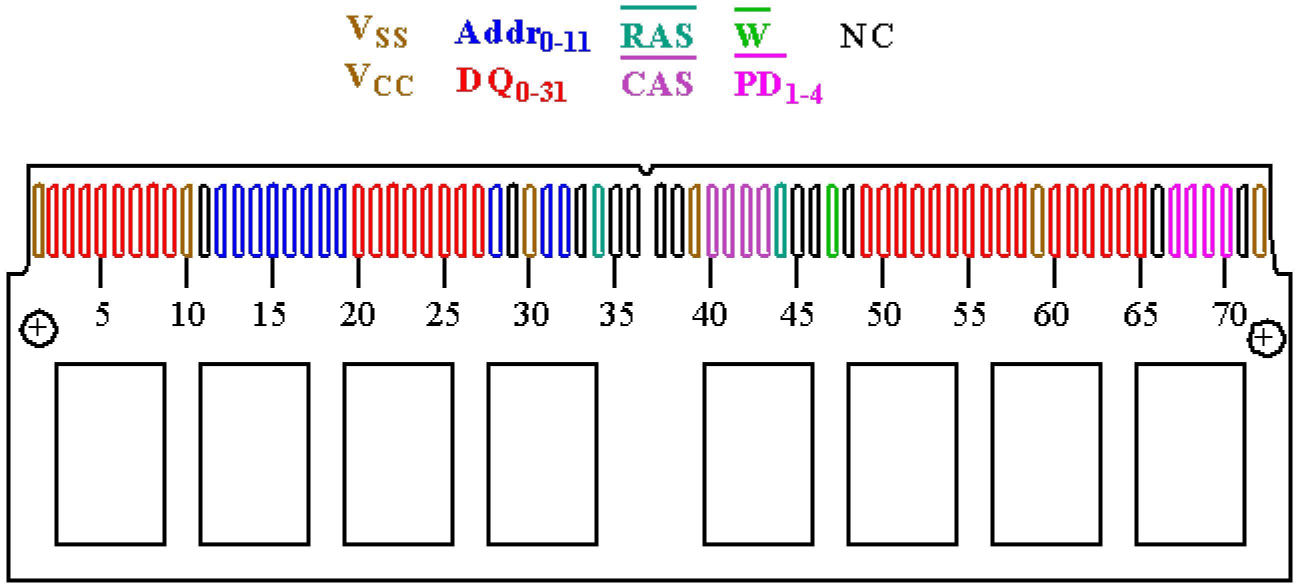
[Picture source] https://notes.andywu.tw/2017/cpu、ram、bus之間的互動圖解/

DRAM Chip and Dual-Inline-Memory Module (DIMM)



64K x 4 DRAM

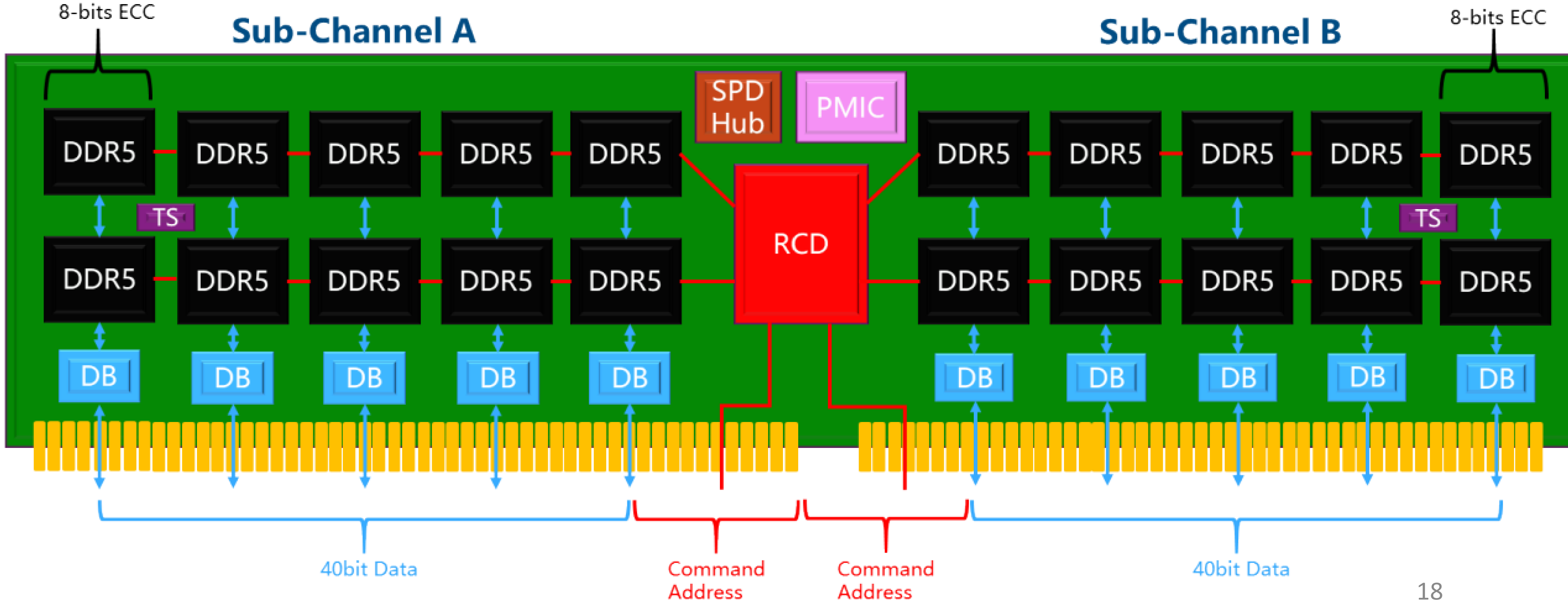
Pin(s)	Function
A_0-A_7	Address
DQ_0-DQ_4	Data In/Data Out
$\overline{RAS}$	Row Address Strobe
$\overline{CAS}$	Column Address Strobe
$\overline{G}$	Output Enable
$\overline{W}$	Write Enable



[Picture Source] https://ece-research.unm.edu/jimp/310/slides/8086_memory1.html

A Modern DIMM Module

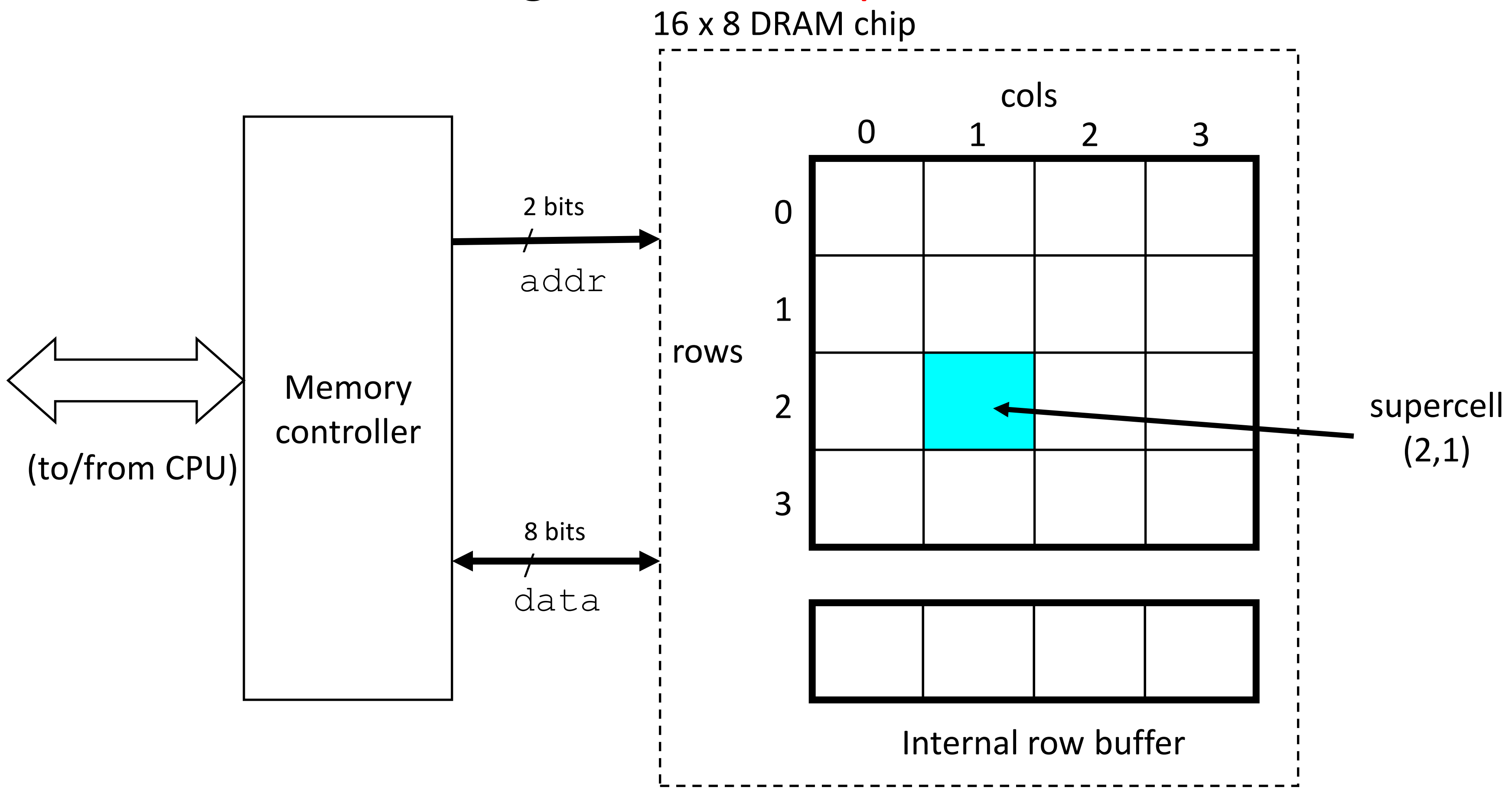
- On-Die ECC (Error-Correction Code): Checks DRAM data for errors and ensures data integrity.
- RCD (Registering Clock Driver): Handles commands, signals, and timing, then sends them to DRAM.
- SPD Hub (Serial Presence Detect EEPROM): Manages access between external controllers and DRAM.
- DB (Data Buffer): Buffers data to improve signal quality before reaching DRAM.
- TS (Temperature Sensor): Monitors DIMM temperature for optimal performance.
- PMIC (Power Management IC): Provides the required voltages to DRAM and I/O components.



[Picture Source] https://www.macnica.com/apac/galaxy/zh_tw/products-support/technical-articles/double-data-rate/

Conventional DRAM Organization

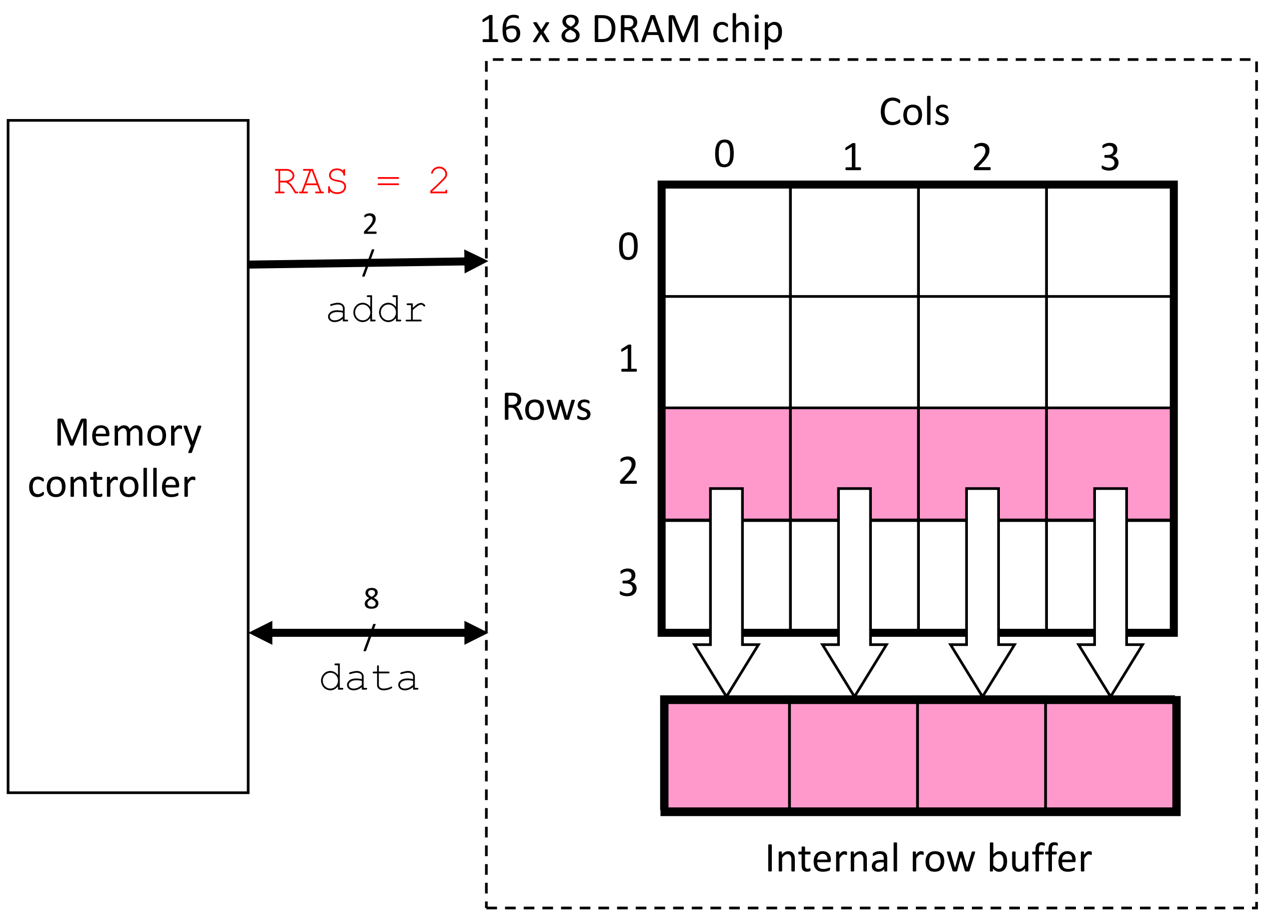
- d x w DRAM:
  - dw total bits organized as d supercells of size w bits



Reading DRAM Supercell (2,1)

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

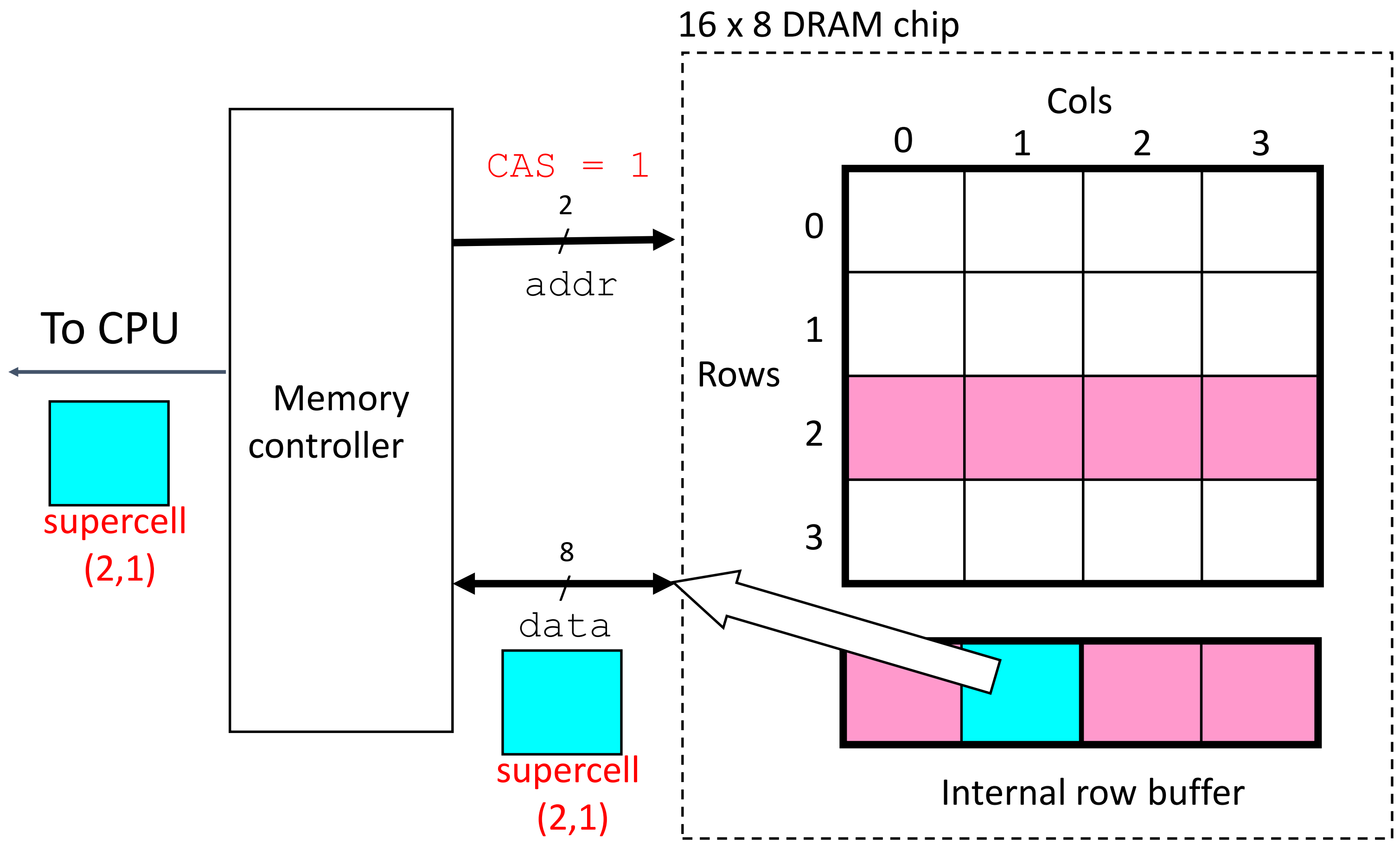
Step 1(b): Row 2 copied from DRAM array to row buffer.



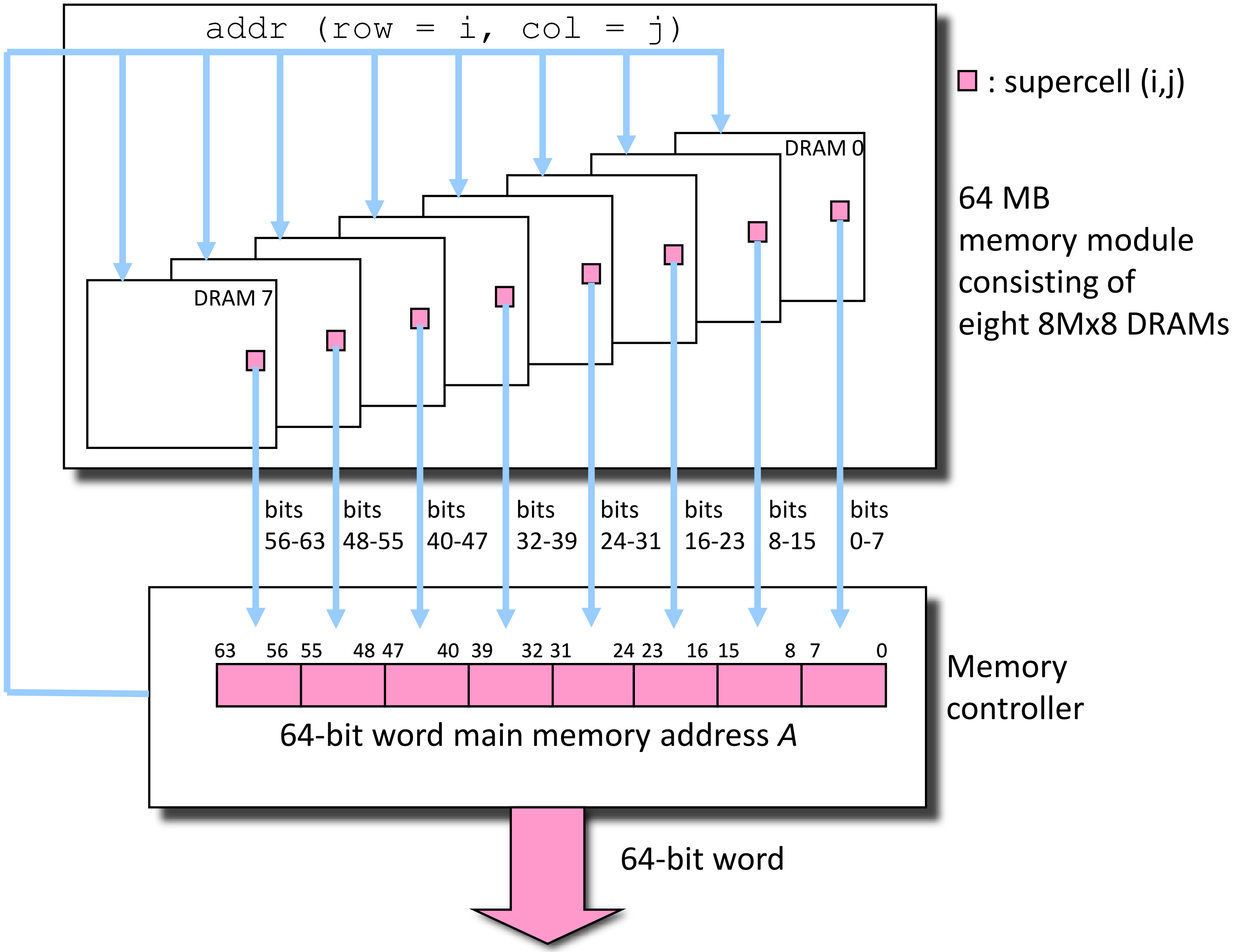
Reading DRAM Supercell (2,1)

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.

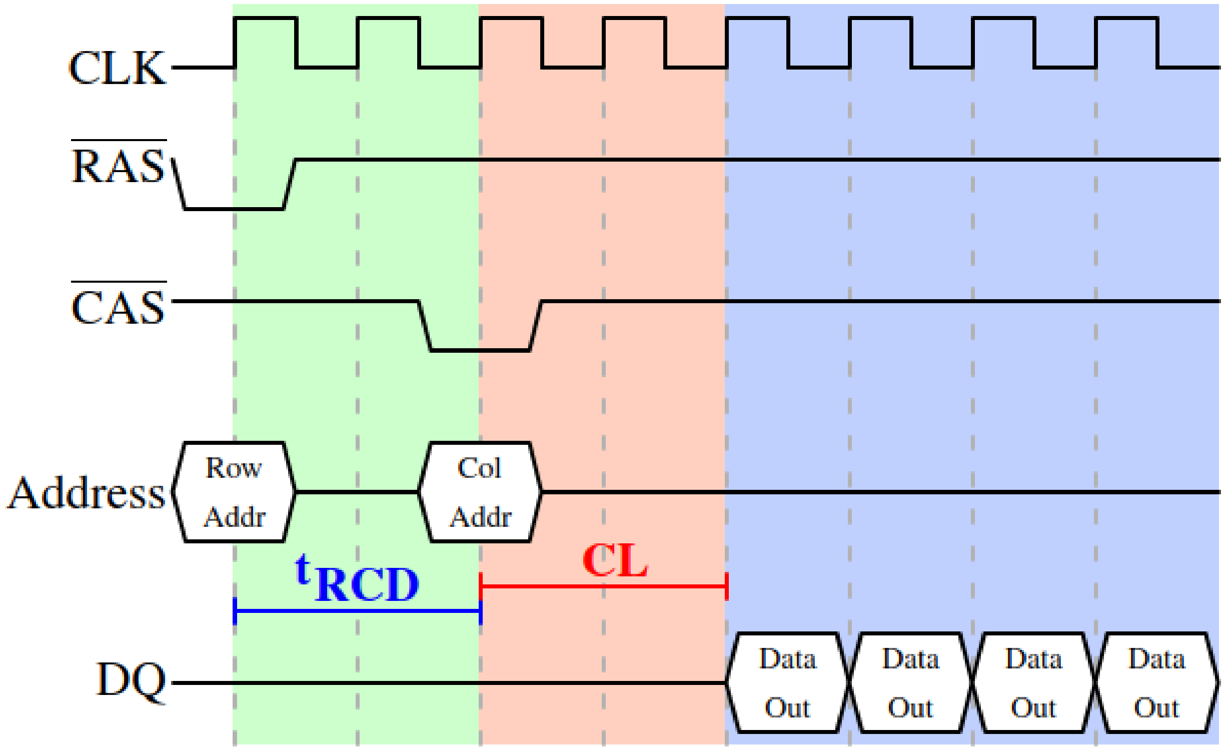


Memory Modules



Timing Constraints of DRAM

- t_{RCD}: RAS-to-CAS Delay (ACT→READ/WRITE)
- t_{CL}: CAS Latency (READ→*data*)

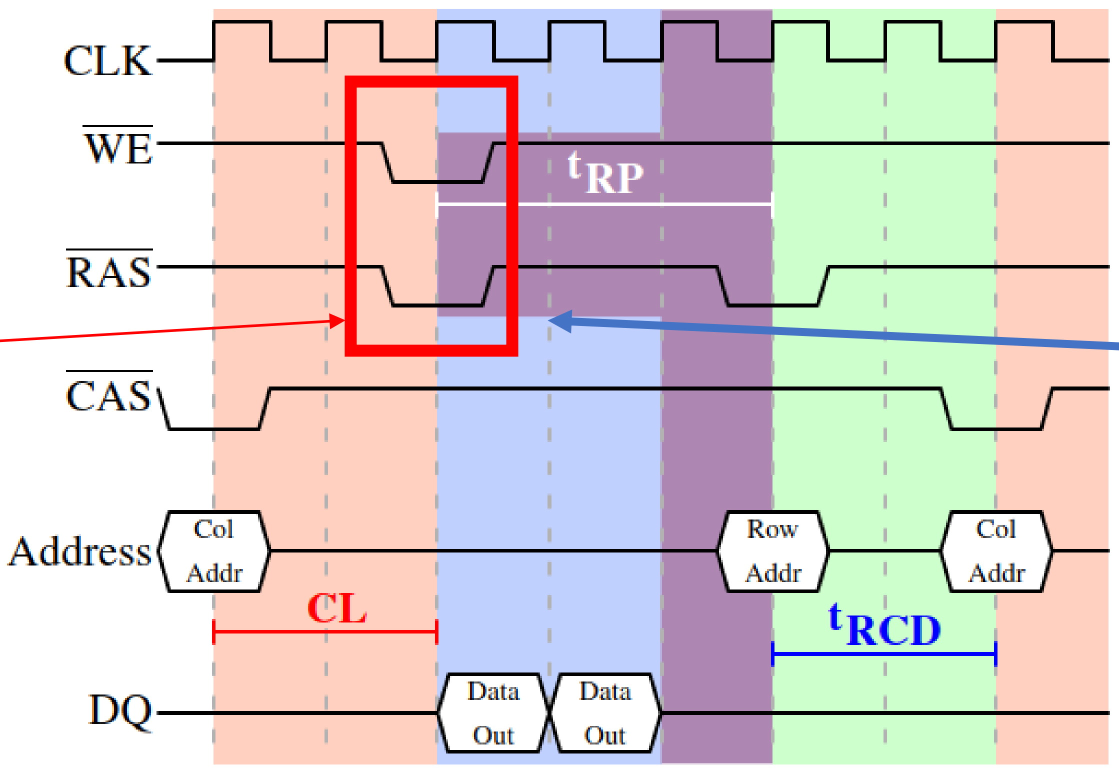


[Picture source] Ulrich Drepper, "What Every Programmer Should Know About Memory," 2007.

Timing Constraints of DRAM (con't)

- t_{RP}: Row Precharge time (PRE→ACT)
- t_{RAS}: Activate to Precharge delay (ACT→PRE)

Precharge

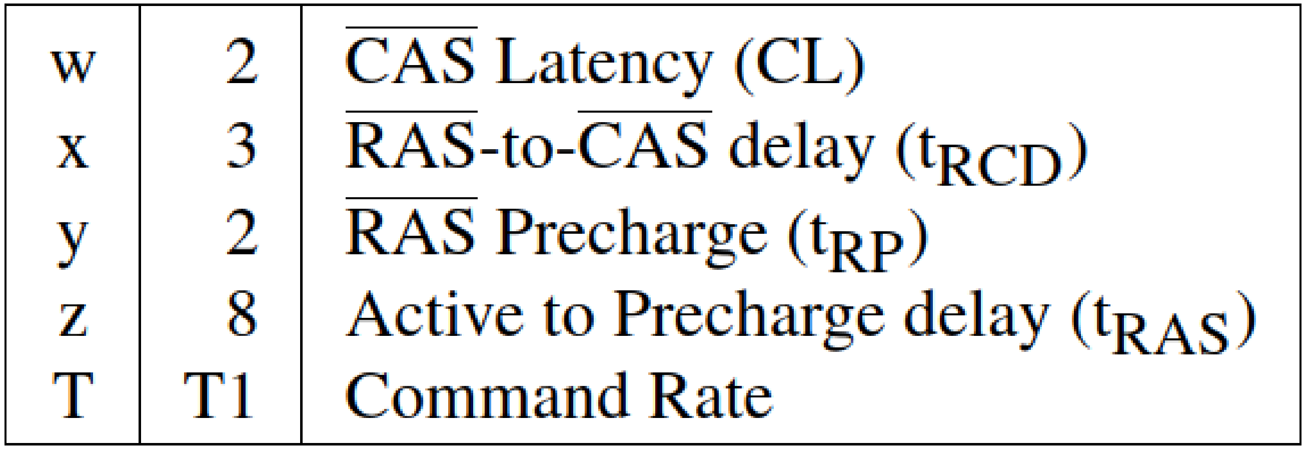


Has to be further delayed for 1 more cycles if the previous RAS only have one CAS with t_{RAS}=8 constraints!

[Picture source] Ulrich Drepper, "What Every Programmer Should Know About Memory," 2007.

DRAM Module Convention

- DRAM Module are often described using the following format:
  - w-x-y-z-T
  - E.g., For a DDR module with 2-3-2-8-T1:



w	2	$\overline{\text{CAS}}$ Latency (CL)
x	3	$\overline{\text{RAS}}$-to-$\overline{\text{CAS}}$ delay (t_{RCD})
y	2	$\overline{\text{RAS}}$ Precharge (t_{RP})
z	8	Active to Precharge delay (t_{RAS})
T	T1	Command Rate

[Picture source] Ulrich Drepper, “What Every Programmer Should Know About Memory,” 2007.

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

SDR/DDR Types

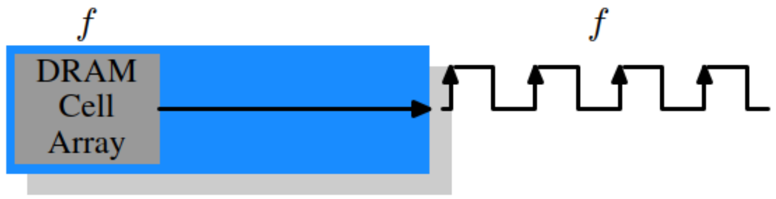


Figure 2.10: SDR SDRAM Operation

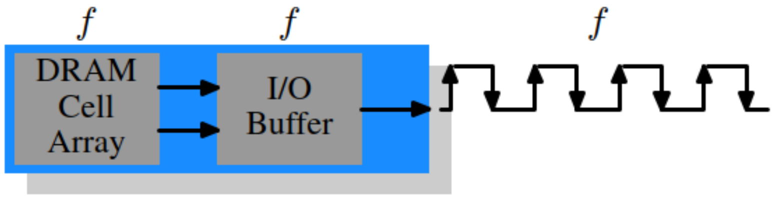


Figure 2.11: DDR1 SDRAM Operation

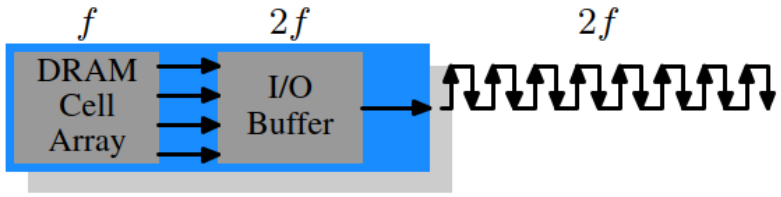


Figure 2.12: DDR2 SDRAM Operation

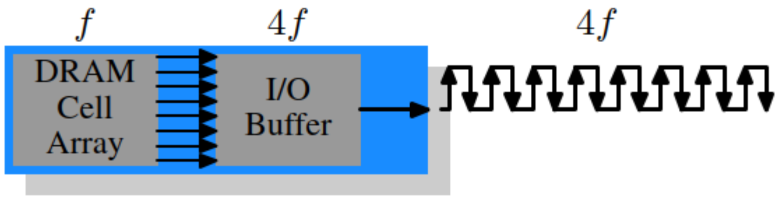


Figure 2.13: DDR3 SDRAM Operation

[Picture source] Ulrich Drepper, “What Every Programmer Should Know About Memory,” 2007.

Storage Devices: Hard Disk Drive (HDDs) and Solid-State Drive (SSDs)

What's Inside A Disk Drive?

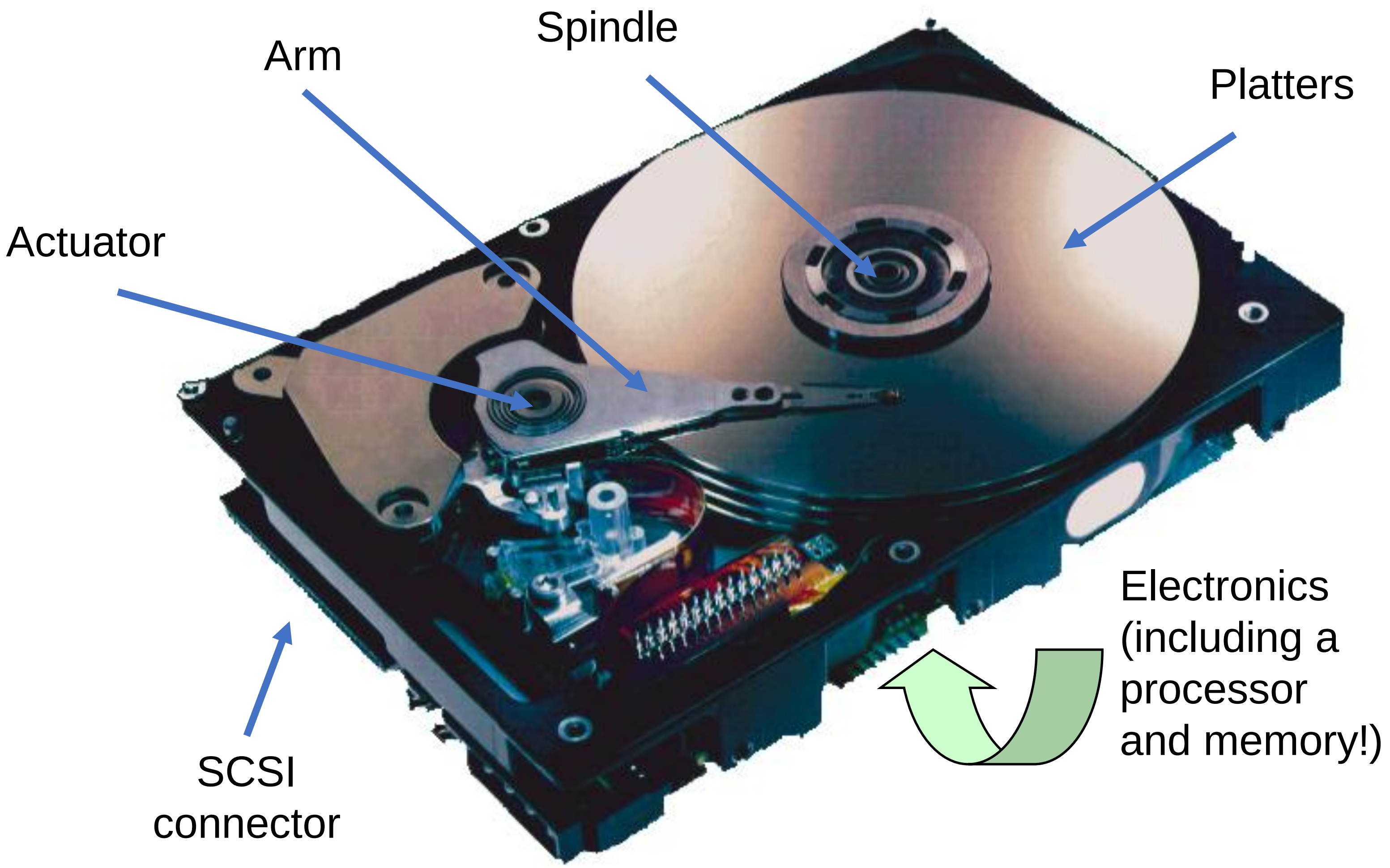
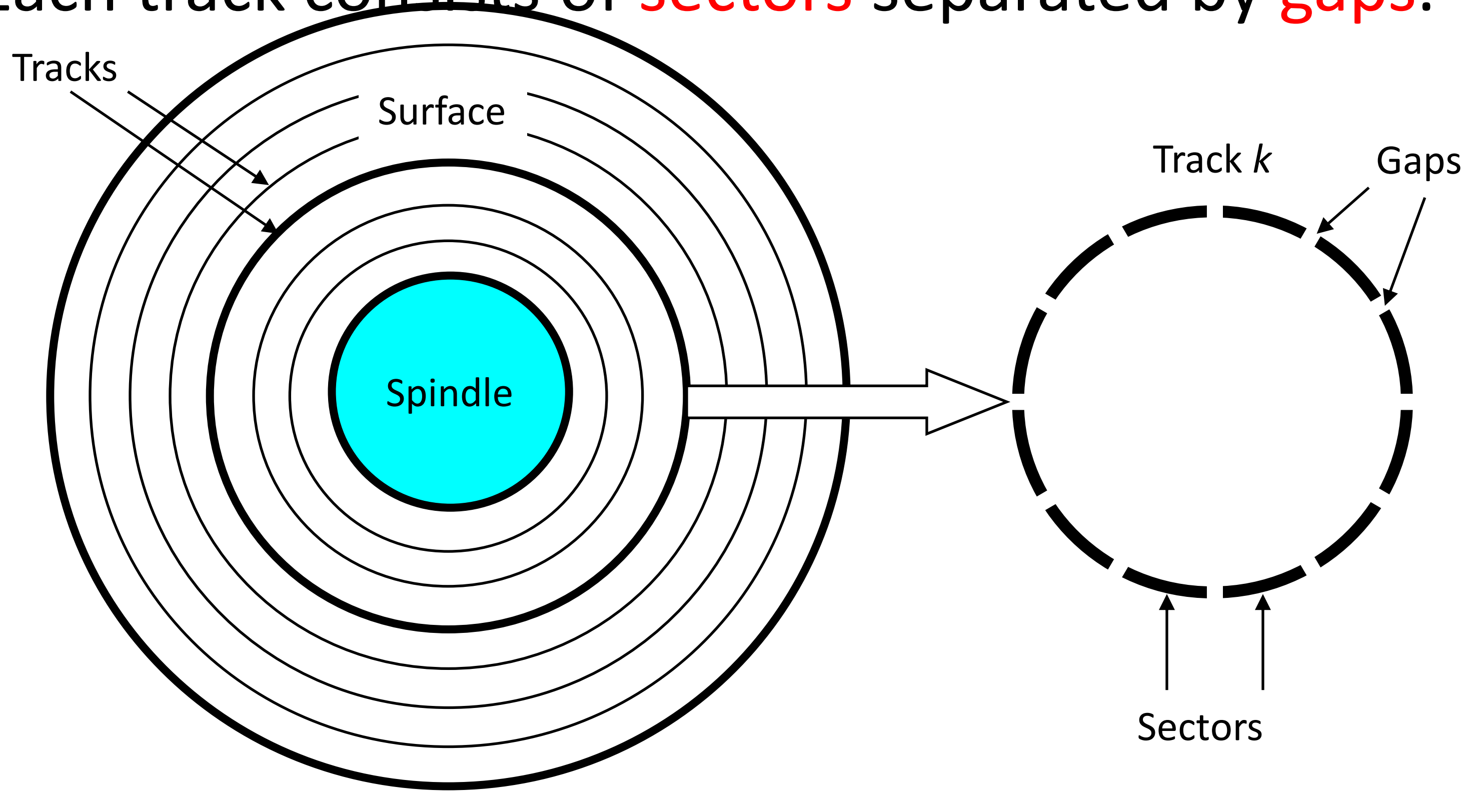


Image courtesy of Seagate Technology

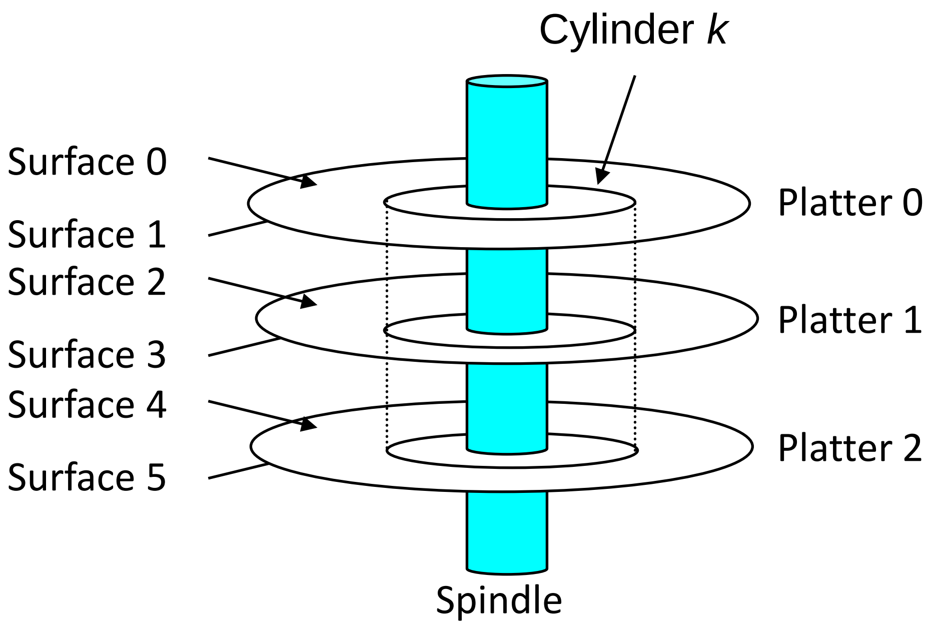
Disk Geometry

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



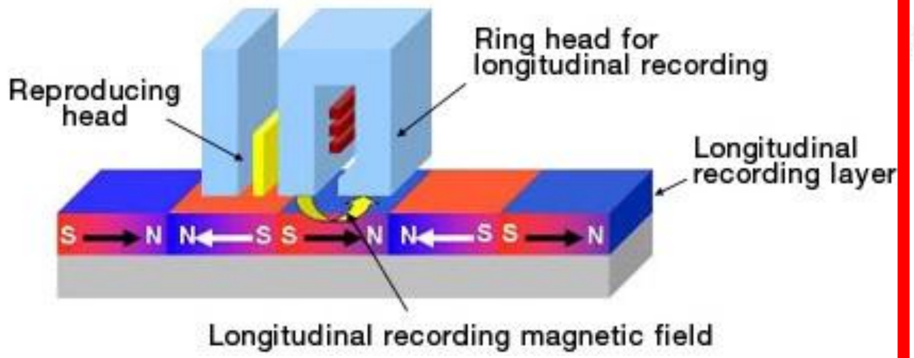
Disk Geometry (Muliple-Platter View)

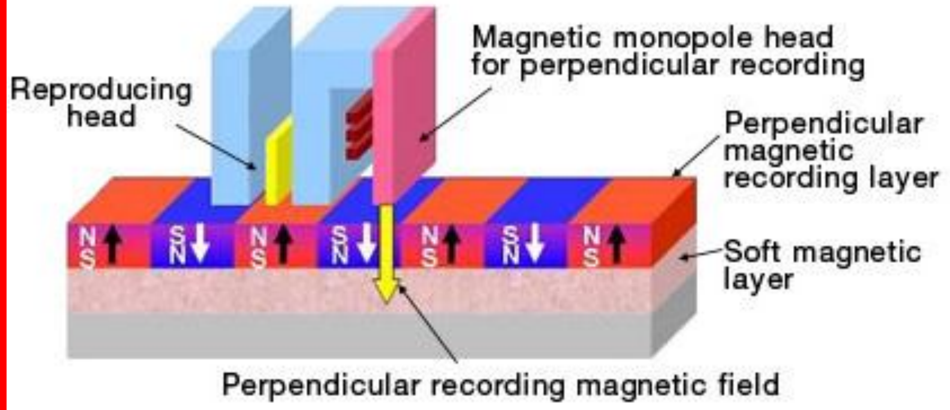
- Aligned tracks form a cylinder.



Longitudinal vs. Perpendicular Recording

- Perpendicular recording has become the mainstream of the HDD since the 2000s


Reproducing head
Ring head for longitudinal recording
Longitudinal recording layer
S N N S S N N S S N
Longitudinal recording magnetic field


Reproducing head
Magnetic monopole head for perpendicular recording
Perpendicular magnetic recording layer
N S S N N S S N N S S N N S
Soft magnetic layer
Perpendicular recording magnetic field

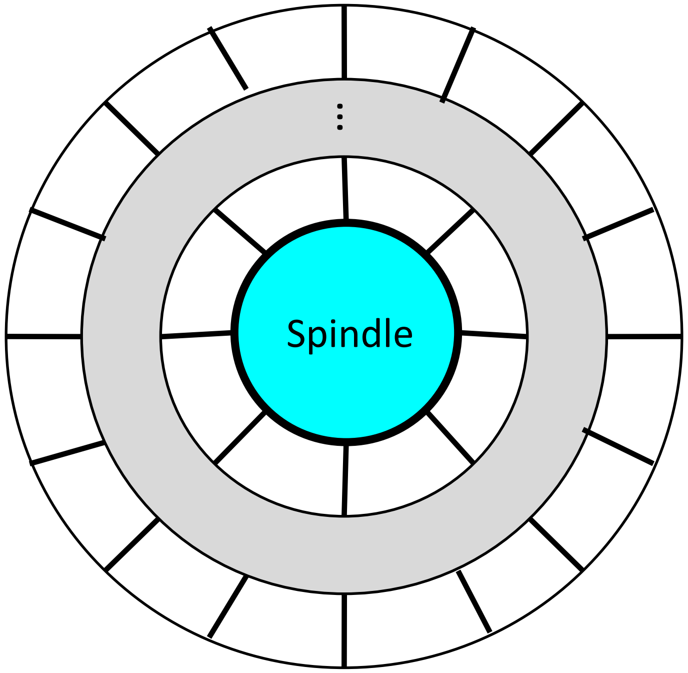
[Picture source] https://www.sdd.toshiba.com.tw/chinese/products/article02.aspx?ccid=4

Disk Capacity

- Capacity: maximum number of bits that can be stored.
  - Vendors express capacity in units of gigabytes (GB) and even terabytes (TB), where 1 GB = 10^9 Bytes, 1TB = 10^{12} 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/in2): product of recording and track density.

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 than inner zones.
    - So we use **average** number of sectors/track when computing capacity.



Computing Disk Capacity

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

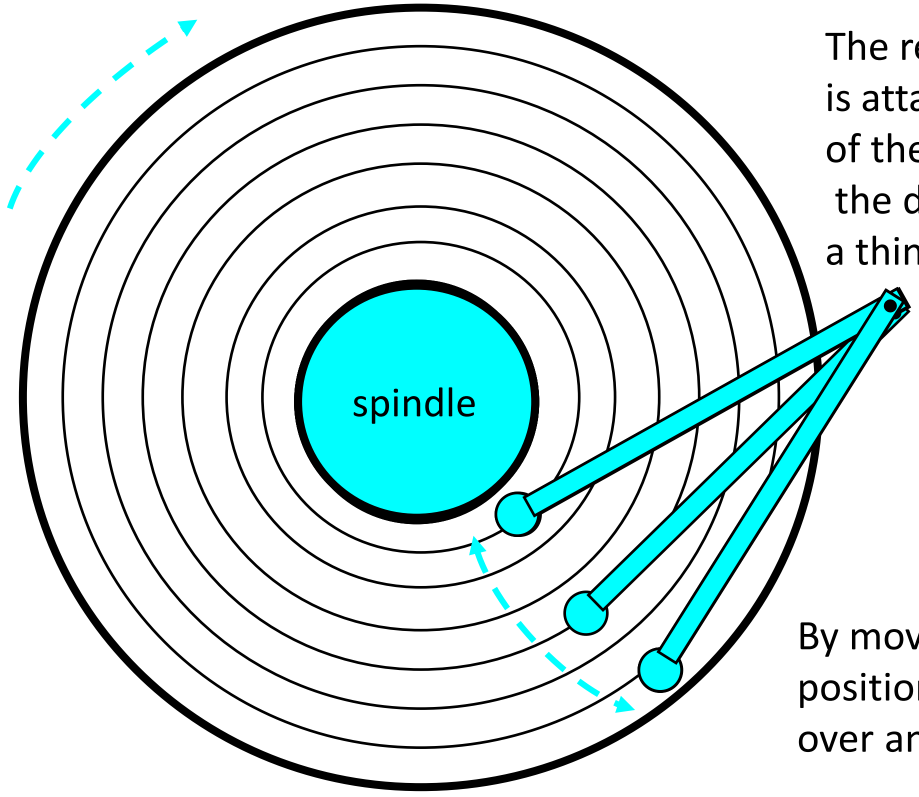
Example:

- 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)

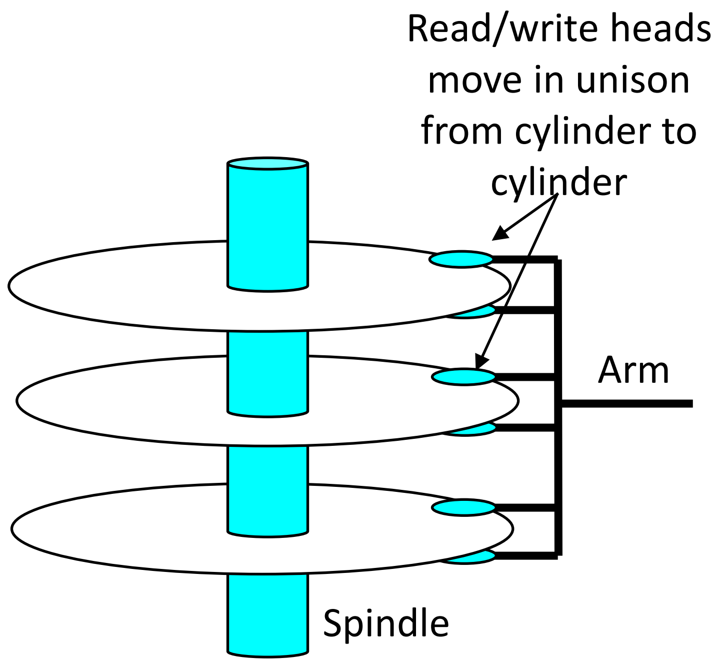
The disk surface spins at a fixed rotational rate



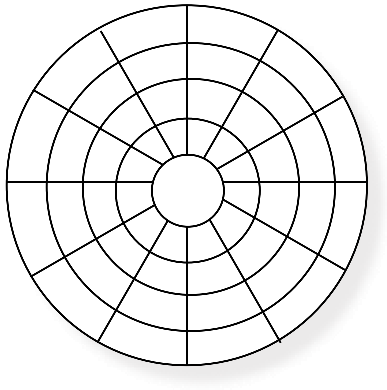
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

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

Disk Operation (Multi-Platter View)



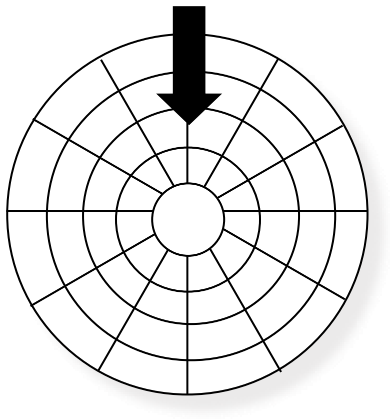
Disk Structure - top view of single platter



Surface organized into tracks

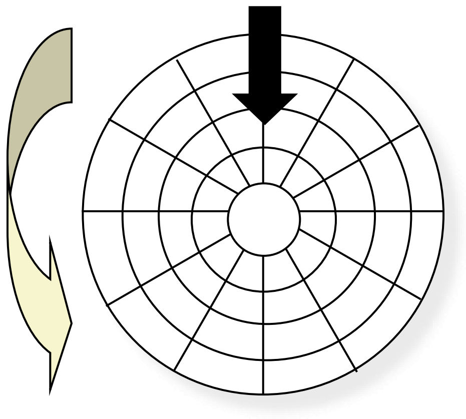
Tracks divided into sectors

Disk Access



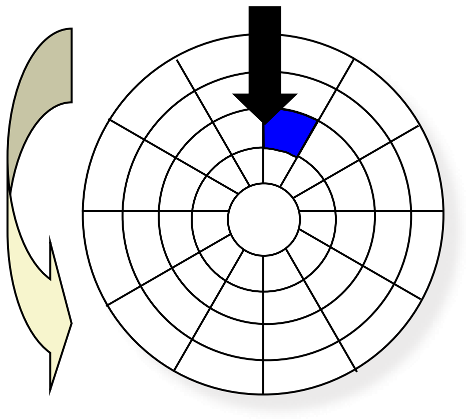
Head in position above a track

Disk Access



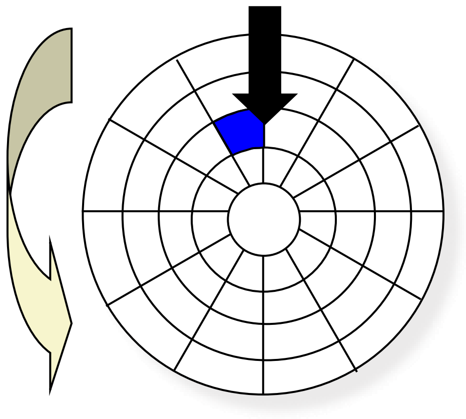
Rotation is counter-clockwise

Disk Access – Read



About to read blue sector

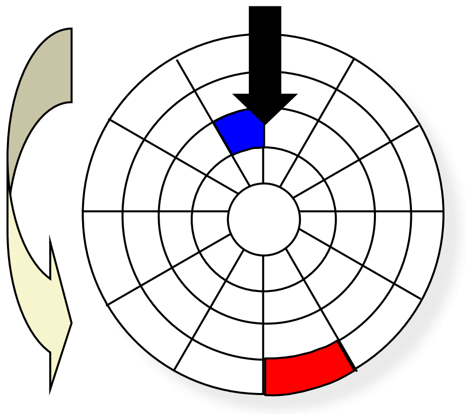
Disk Access – Read



After BLUE read

After reading blue sector

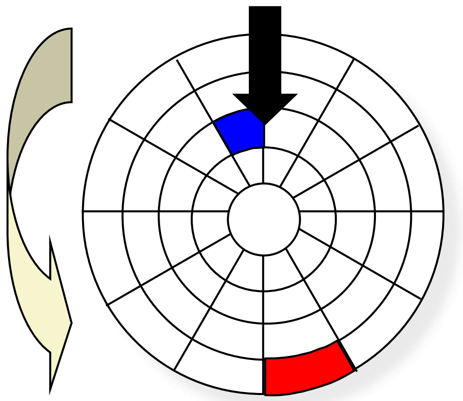
Disk Access – Read

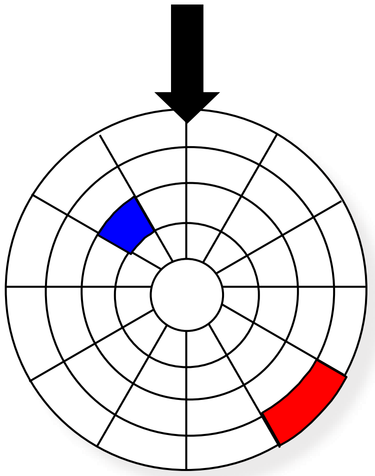


After BLUE read

Red request scheduled next

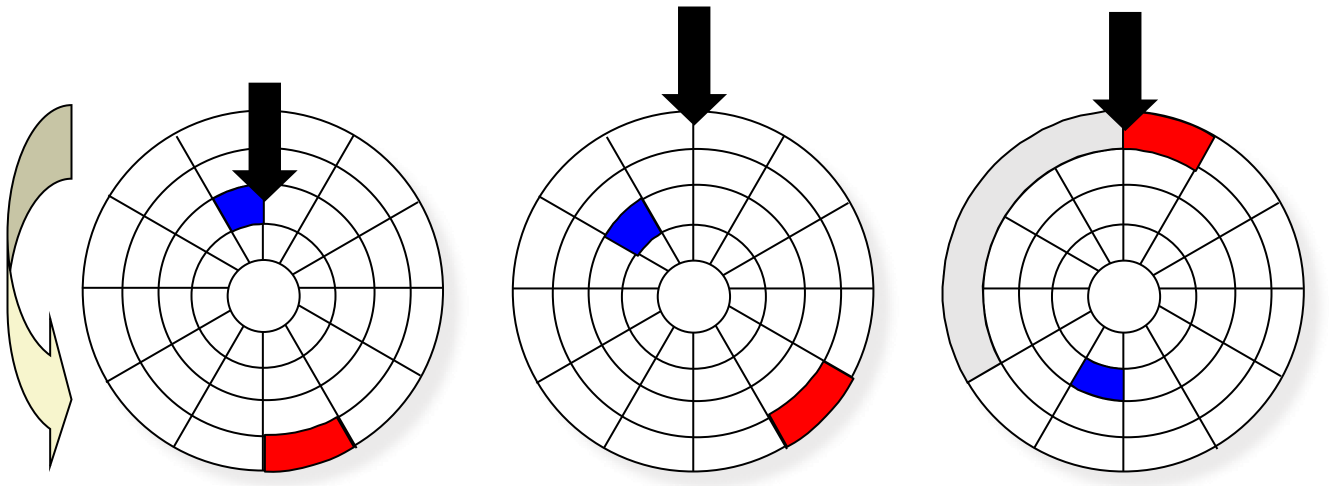
Disk Access – Seek


After BLUE read


Seek for RED

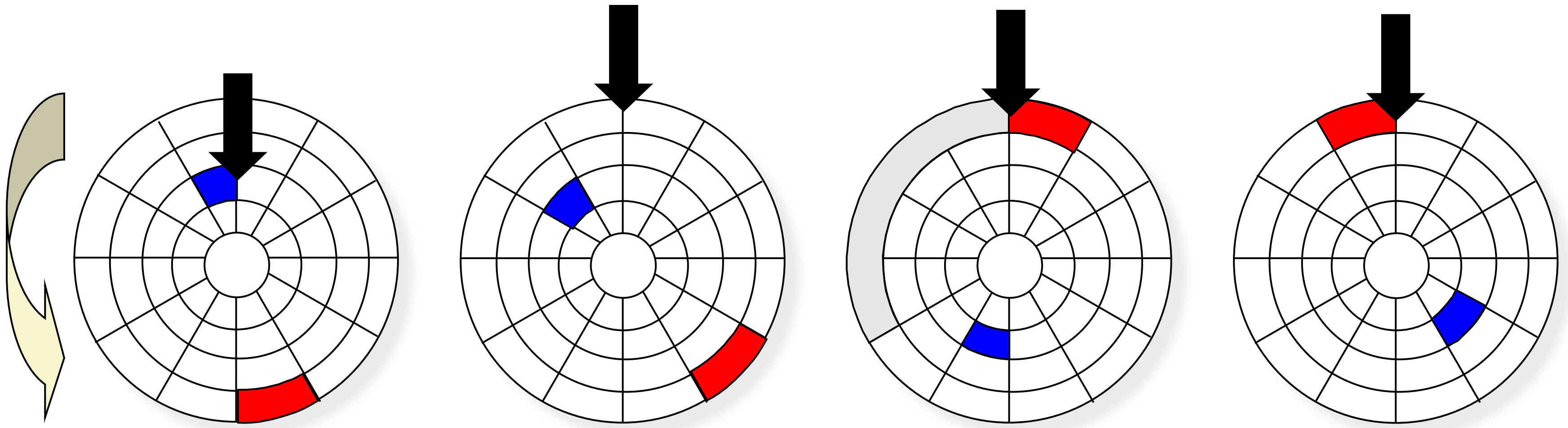
Seek to red's track

Disk Access – Rotational Latency



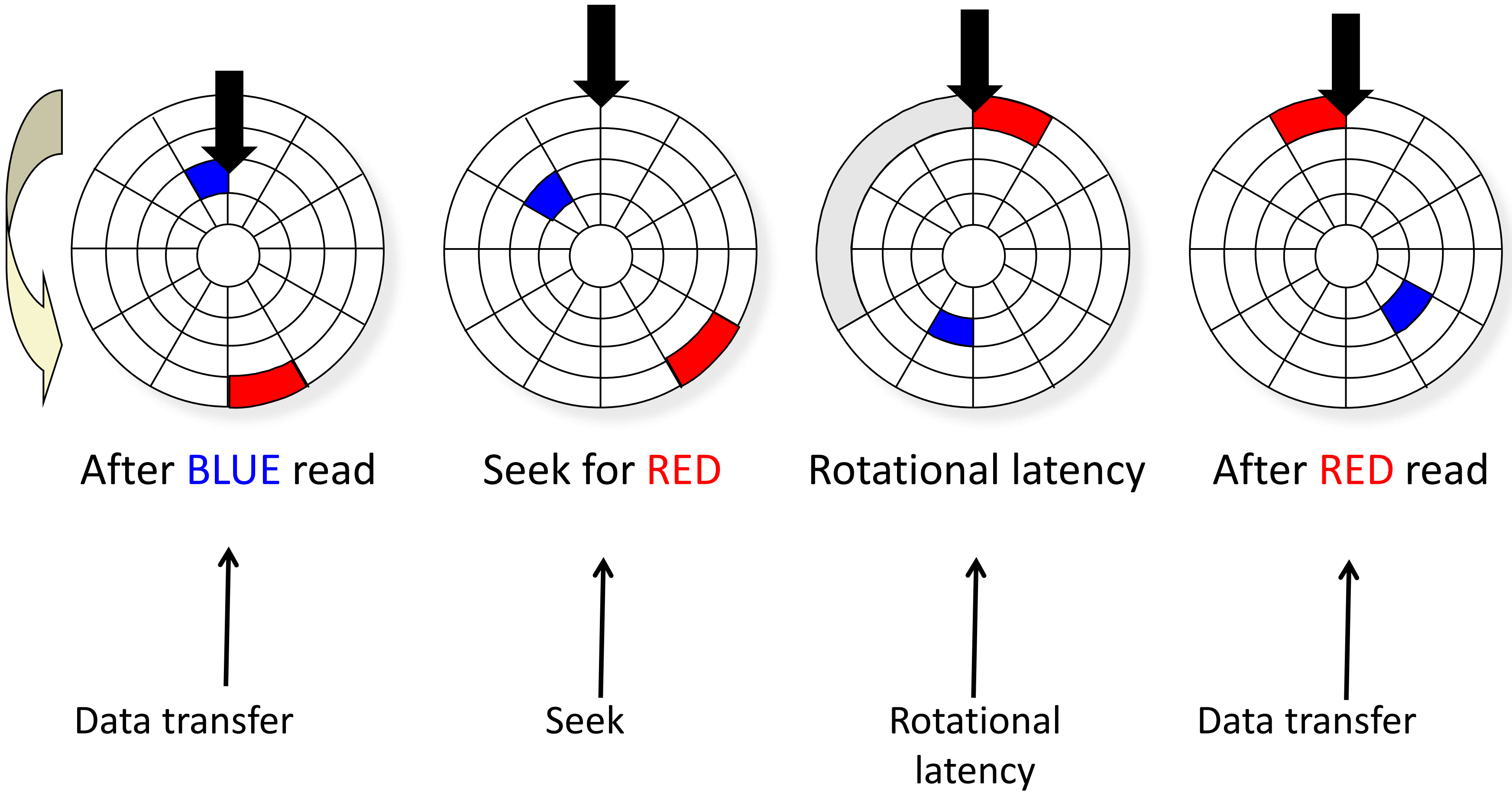
Wait for red sector to rotate around

Disk Access – Read



Complete read of red

Disk Access – Service Time Components



Disk Access Time

- Average time to access some target sector approximated by :
    - Taccess = Tavg seek + Tavg rotation + Tavg transfer
- Seek time (Tavg seek)
    - Time to position heads over cylinder containing target sector.
    - Typical Tavg seek is 3—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
    - Typical rotation = 7200 RPMs
- 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.

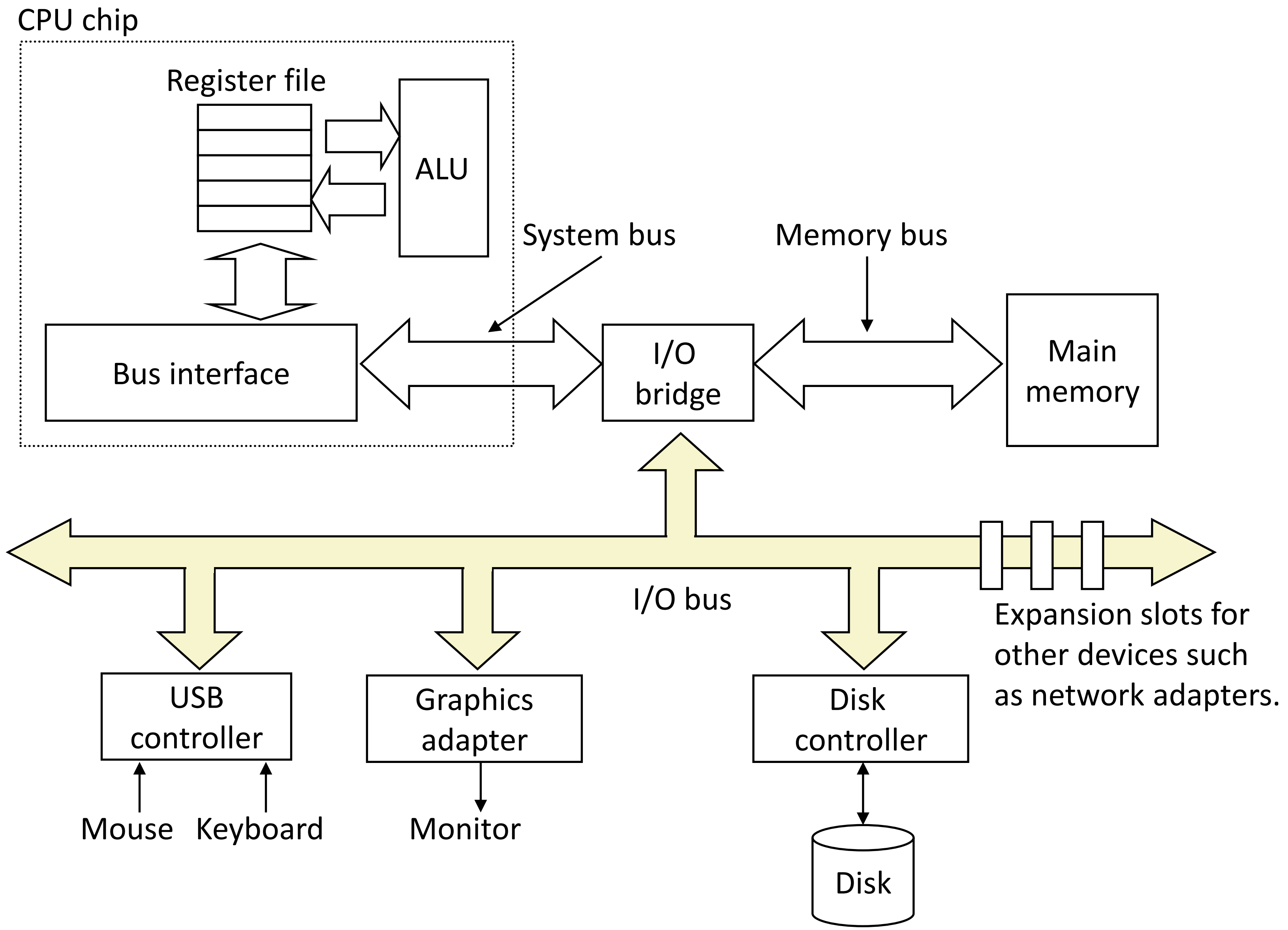
Disk Access Time Example

- 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.

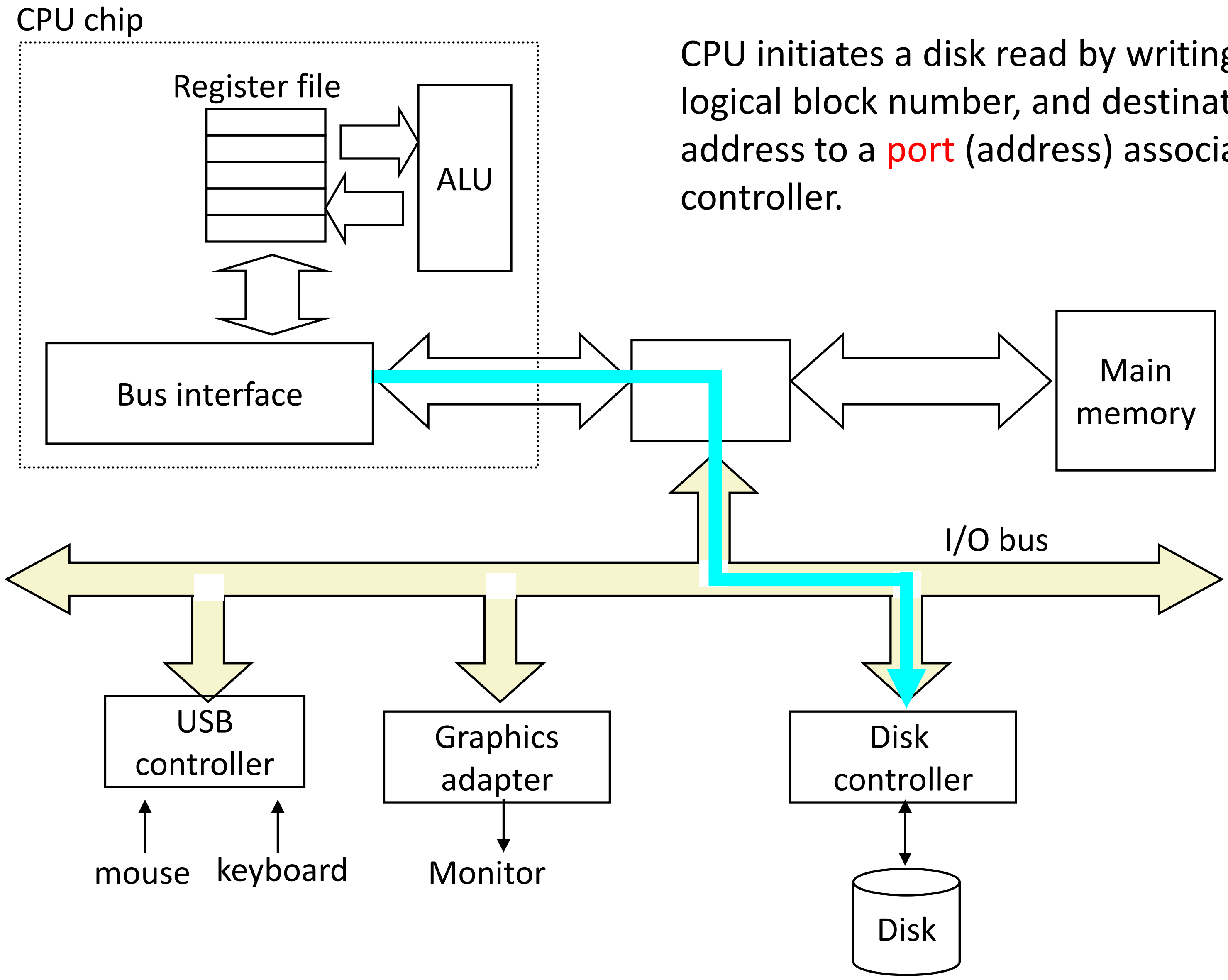
Logical Disk Blocks

- 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

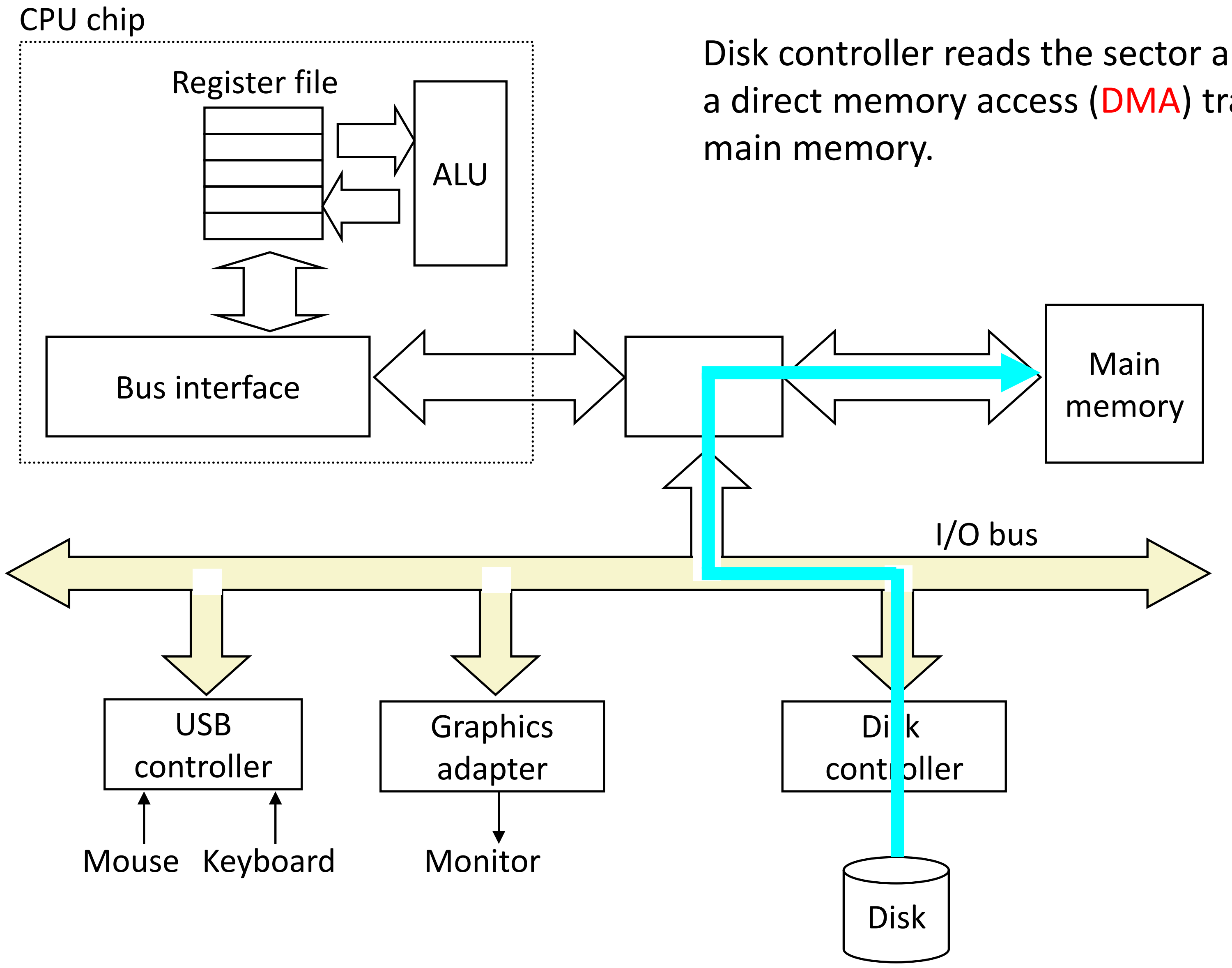
Reading a Disk Sector (1)



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

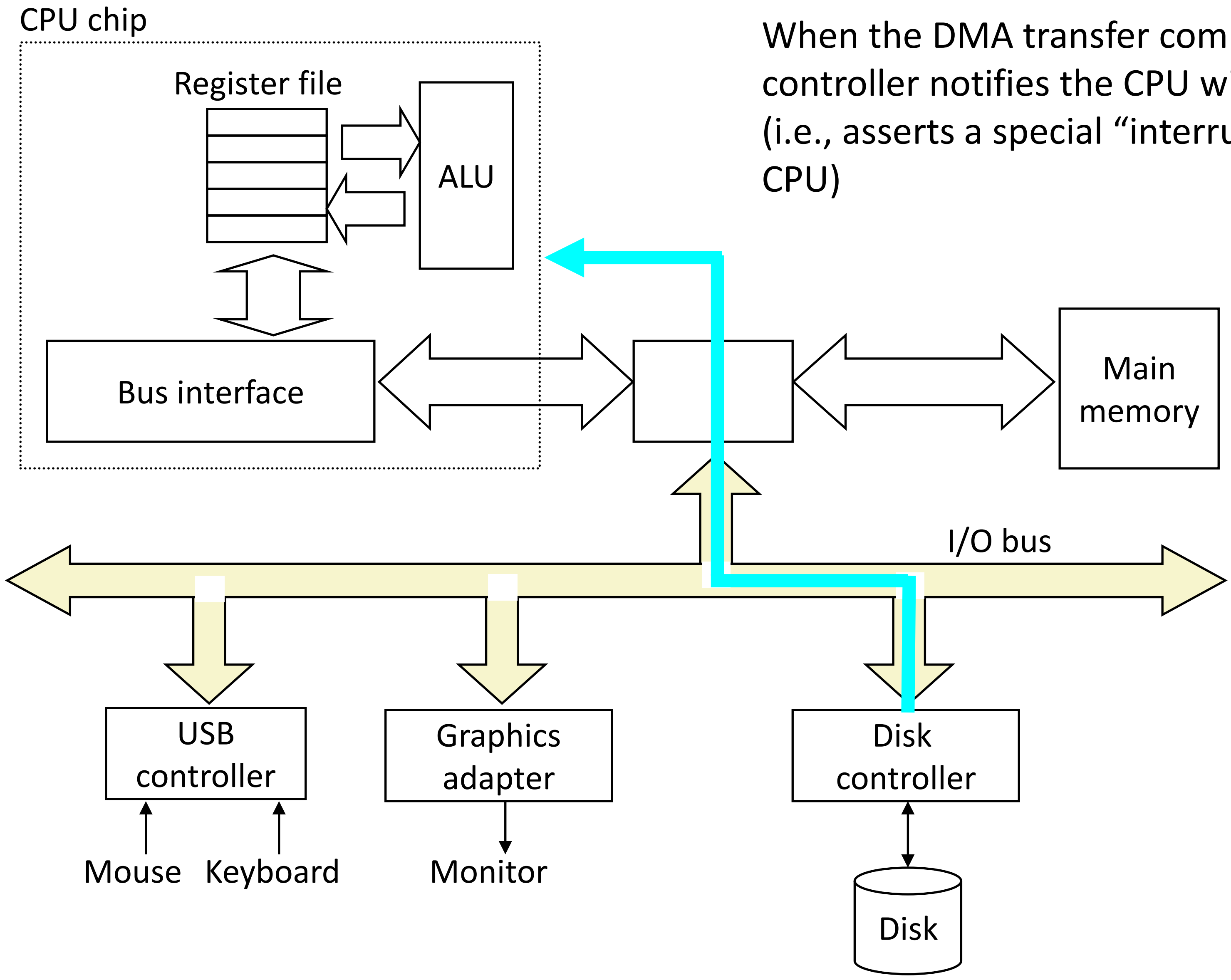
Reading a Disk Sector (2)

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

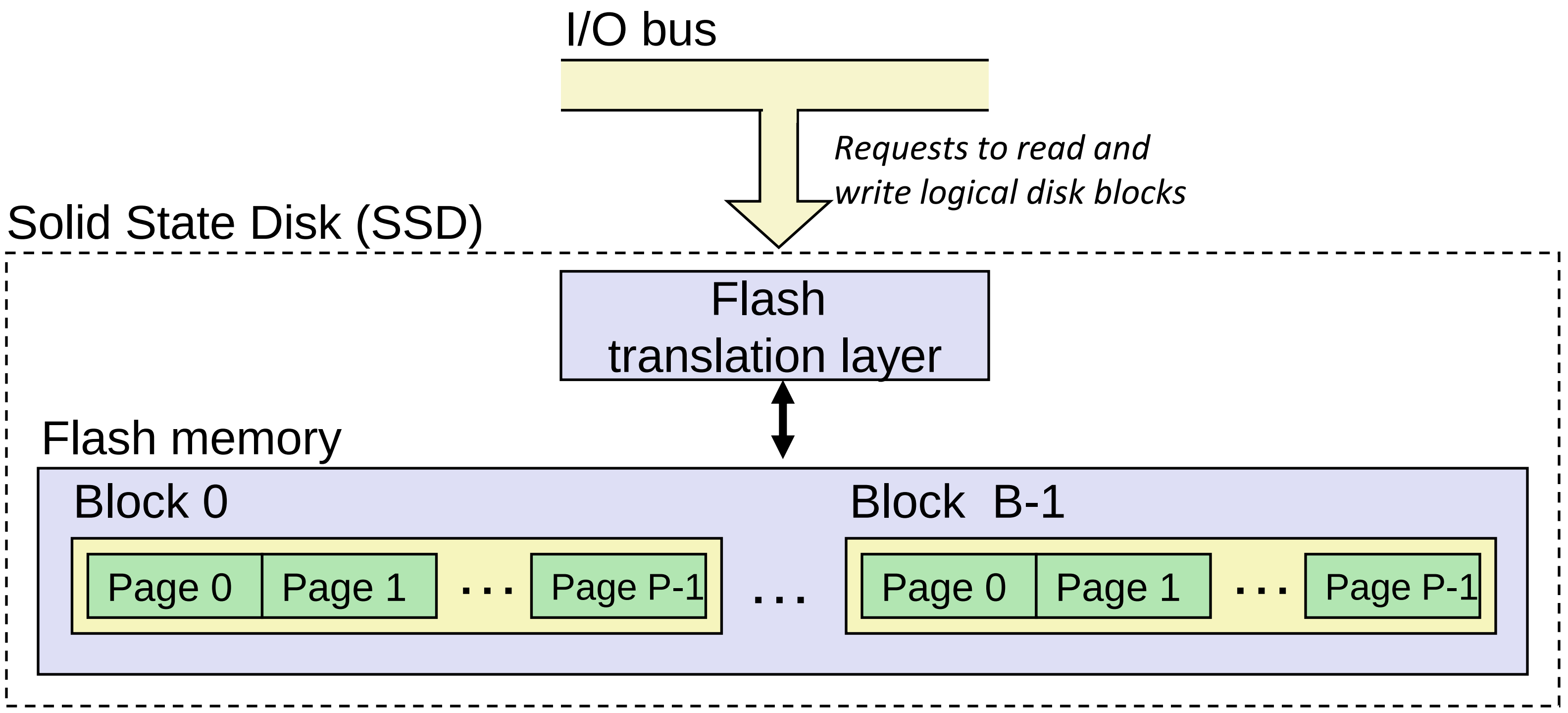


Reading a Disk Sector (3)

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



Solid State Disks (SSDs)



- Pages: 512KB 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 about 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	50 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.

Source: Intel SSD 730 product specification.

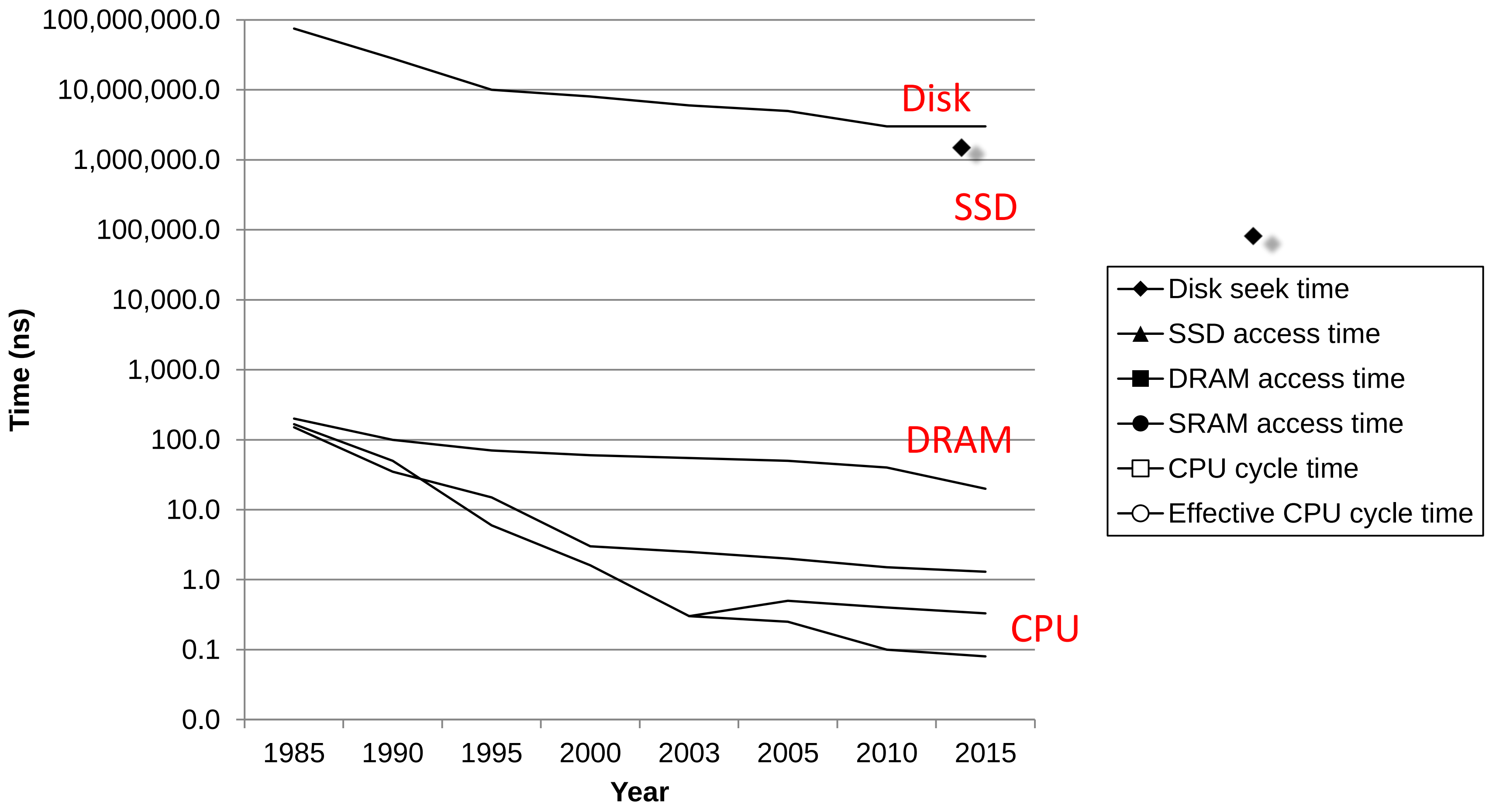
SSD Tradeoffs vs Rotating Disks

- 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 x 10^{15} bytes) of writes before they wear out
  - In 2015, about 30 times more expensive per byte
    - But now the gap is getting smaller, though HDD is still cheaper
- Applications
  - Smart phones, laptops, desktops and servers

The CPU-Memory Gap and Locality

The CPU-Memory Gap

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

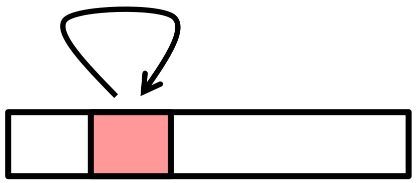


Locality to the Rescue!

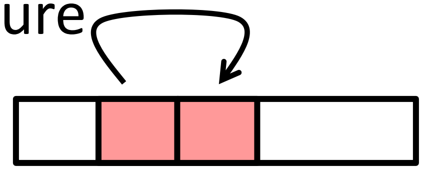
The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as locality

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;
}
```

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)?

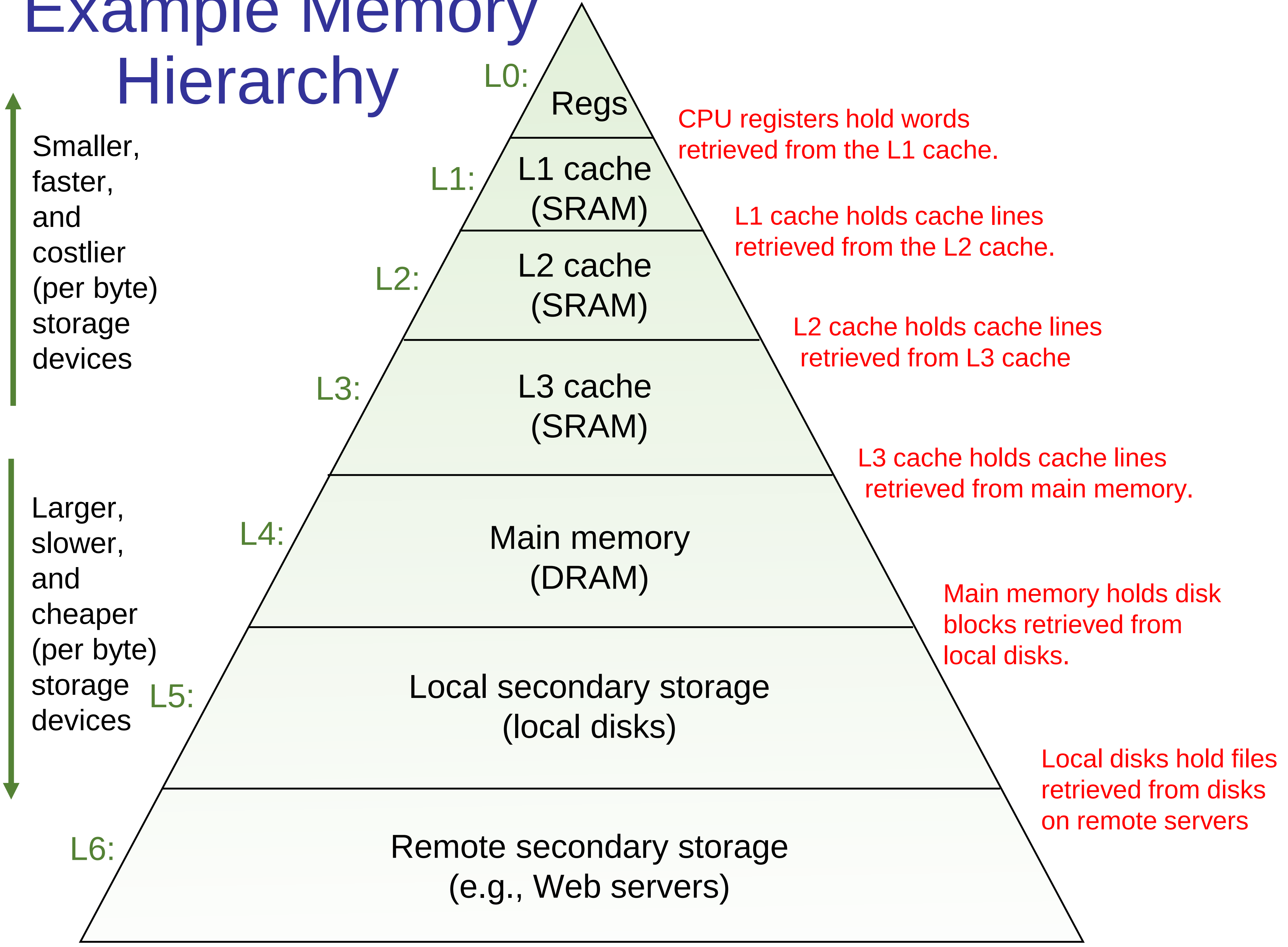
```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < 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

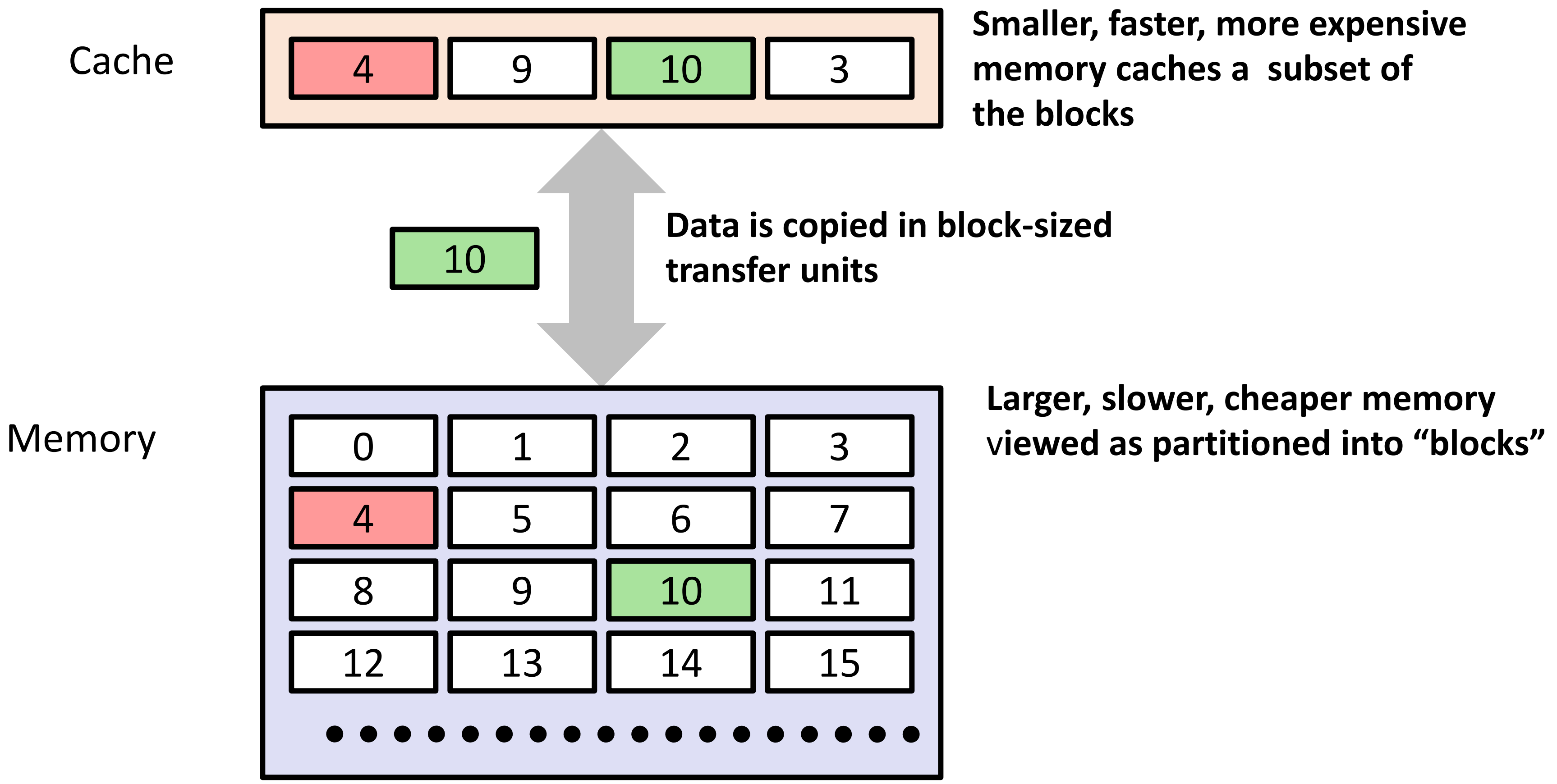


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
L1:
L1 cache (SRAM)
L2:
L2 cache (SRAM)
L3:
L3 cache (SRAM)
L4:
Main memory (DRAM)
L5:
Local secondary storage (local disks)
L6:
Remote secondary storage (e.g., Web servers)
CPU registers hold words retrieved from the L1 cache.
L1 cache holds cache lines retrieved from the L2 cache.
L2 cache holds cache lines retrieved from L3 cache
L3 cache holds cache lines retrieved from main memory.
Main memory holds disk blocks retrieved from local disks.
Local disks hold files retrieved from disks on remote servers

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

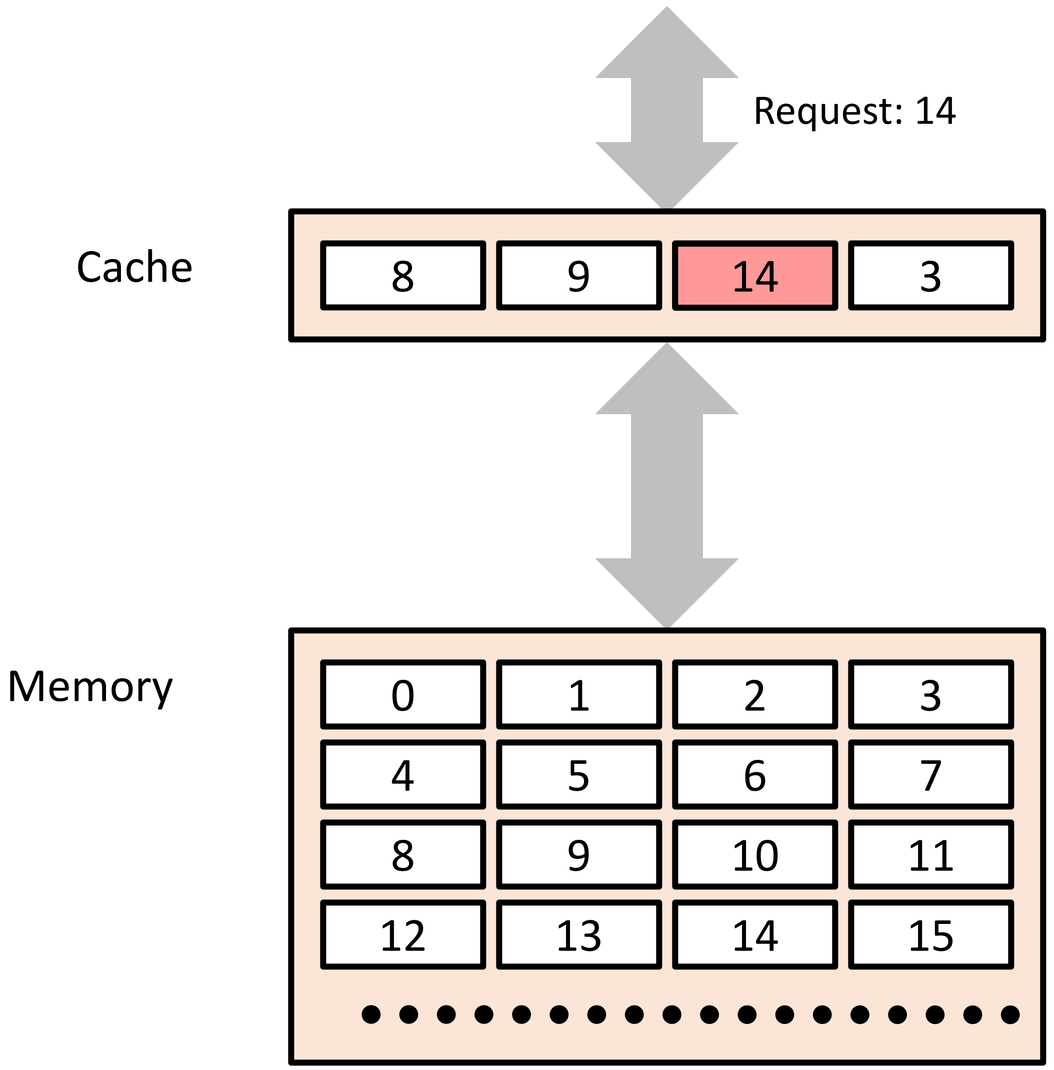


Smaller, faster, more expensive memory caches a subset of the blocks

Data is copied in block-sized transfer units

Larger, slower, cheaper memory viewed as partitioned into "blocks"

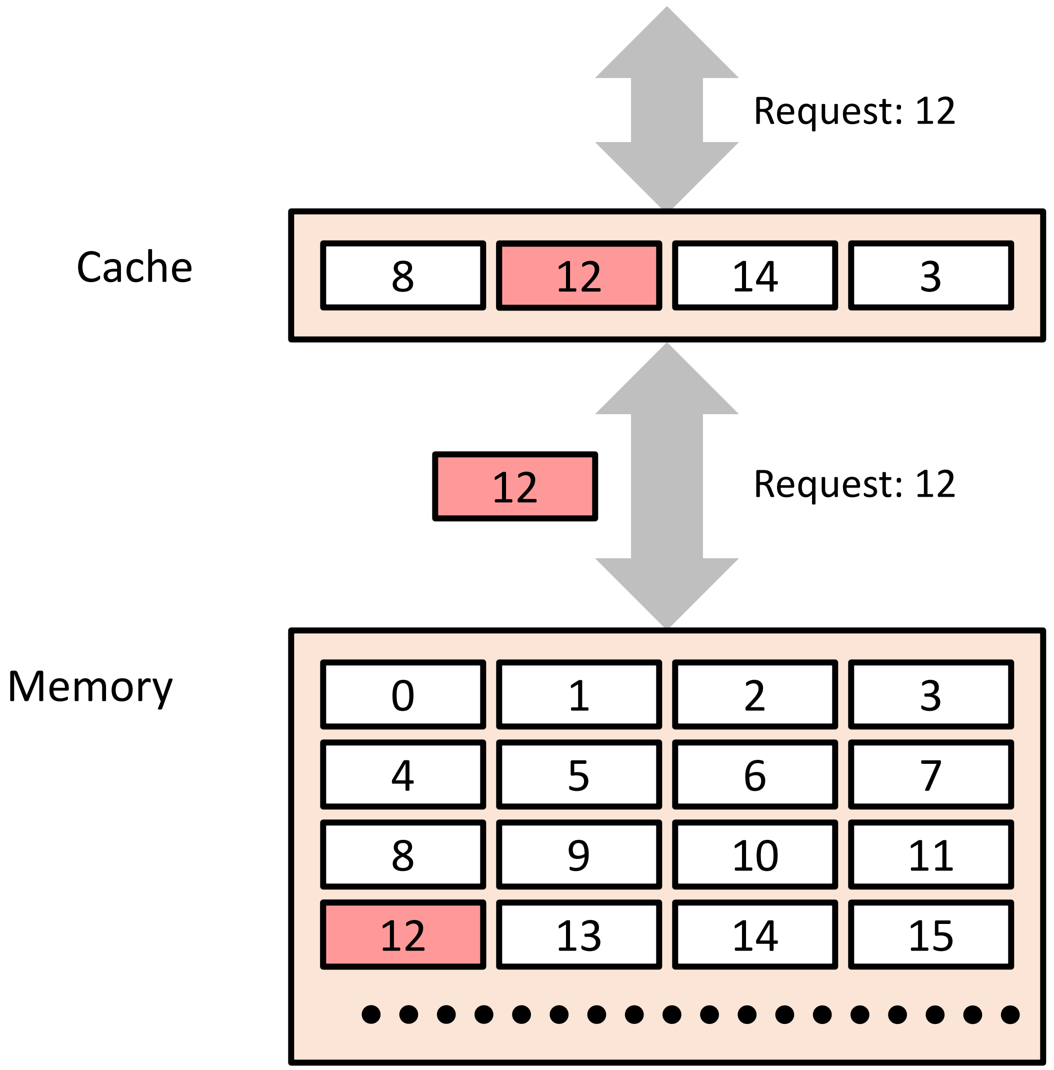
General Cache Concepts: Hit



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: 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 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-byte blocks	On-Chip L1	4	Hardware
L2 cache	64-byte 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 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.

Storage Trends

SRAM

Metric	1985	1990	1995	2000	2005	2010	2015	*2015:1985*
$/MB	2,900	320	256	100	75	60	*25*	*116*
access (ns)	150	35	15	3	2	1.5	*1.3*	*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