The Memory Hierarchy

COMP400727: Introduction to Computer Systems

Danfeng Shan
Xi’an Jiaotong University

Today

- **The memory abstraction**
- RAM : main memory building block
- Locality of reference
- The memory hierarchy
- Storage technologies and trends

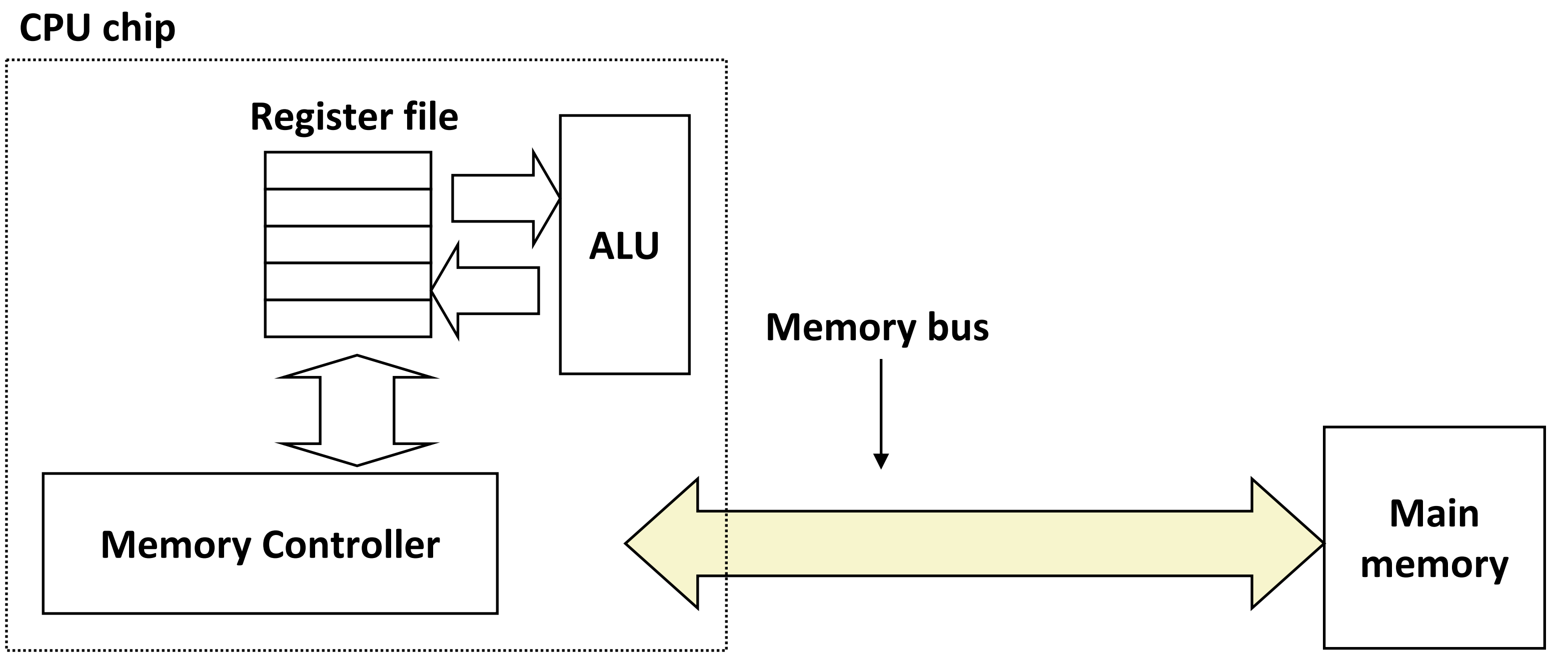
Writing & Reading Memory

- **Write**
  - Transfer data from CPU to memory
    `movq %rax, 8(%rsp)`
  - “Store” operation
- **Read**
  - Transfer data from memory to CPU
    `movq 8(%rsp), %rax`
  - “Load” operation

From 5th lecture

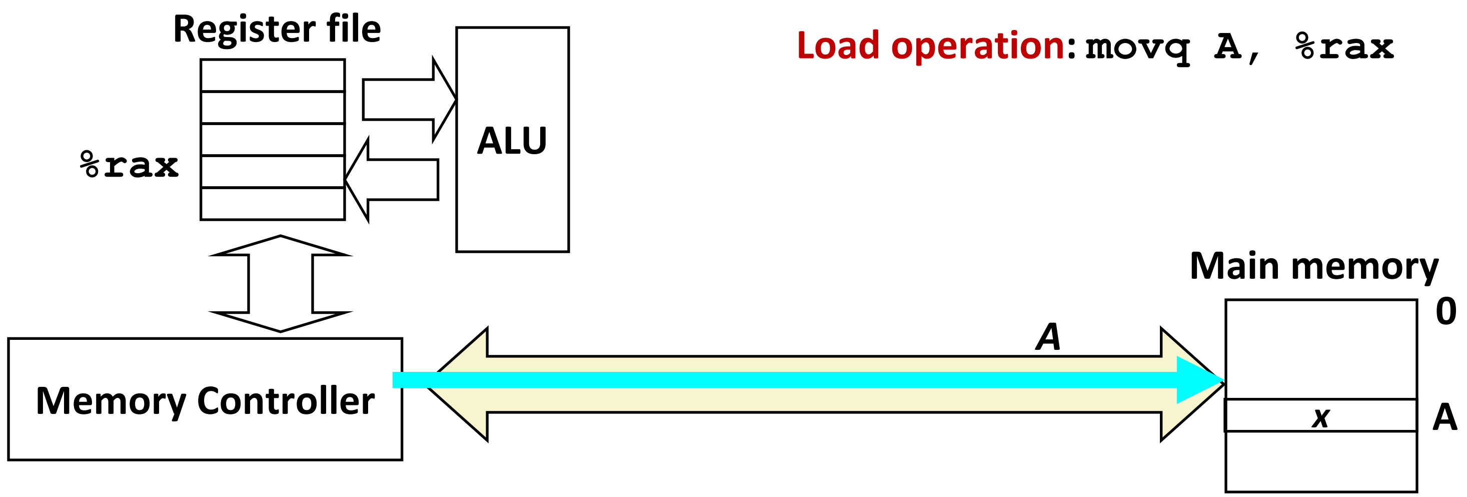
Modern Connection between CPU and Memory

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



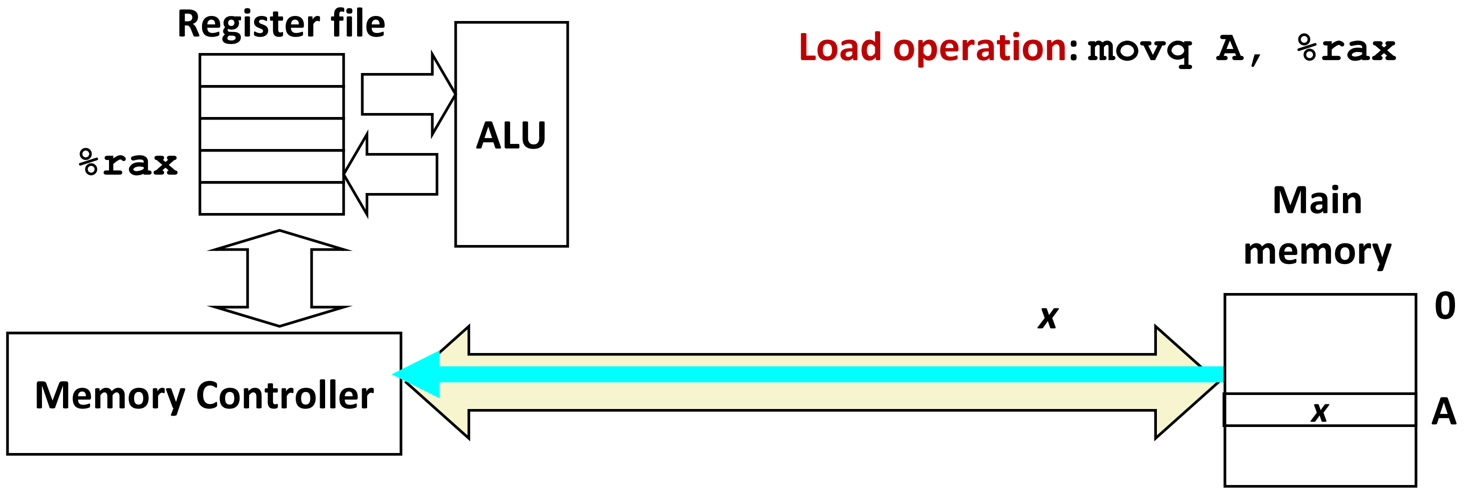
Memory Read Transaction (1)

- CPU places address A on the memory bus.



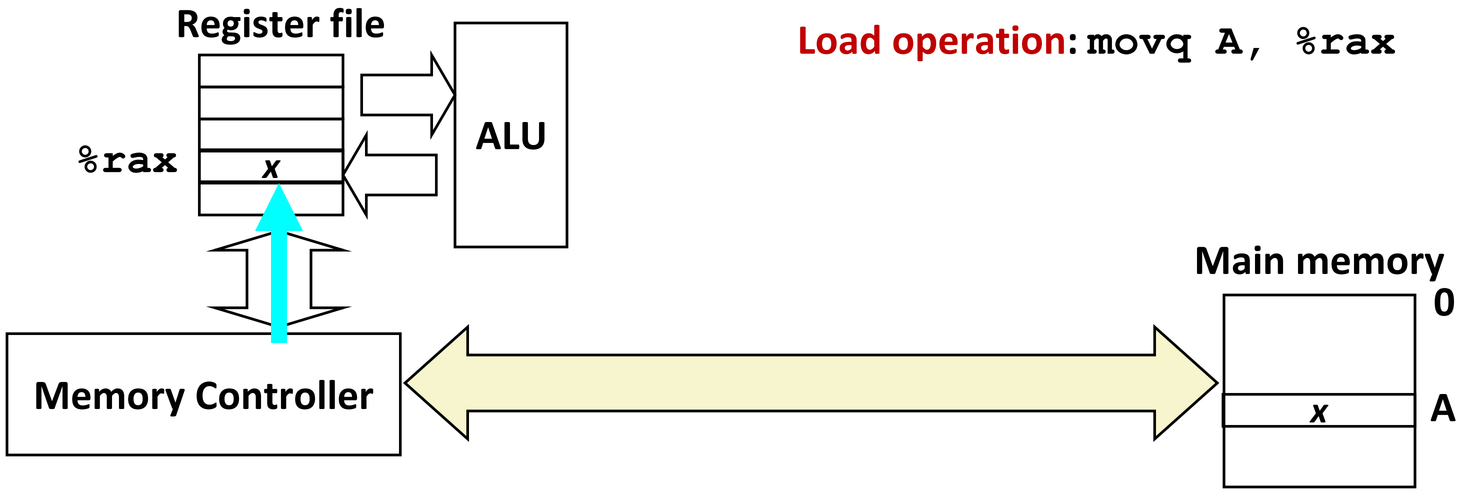
Memory Read Transaction (2)

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



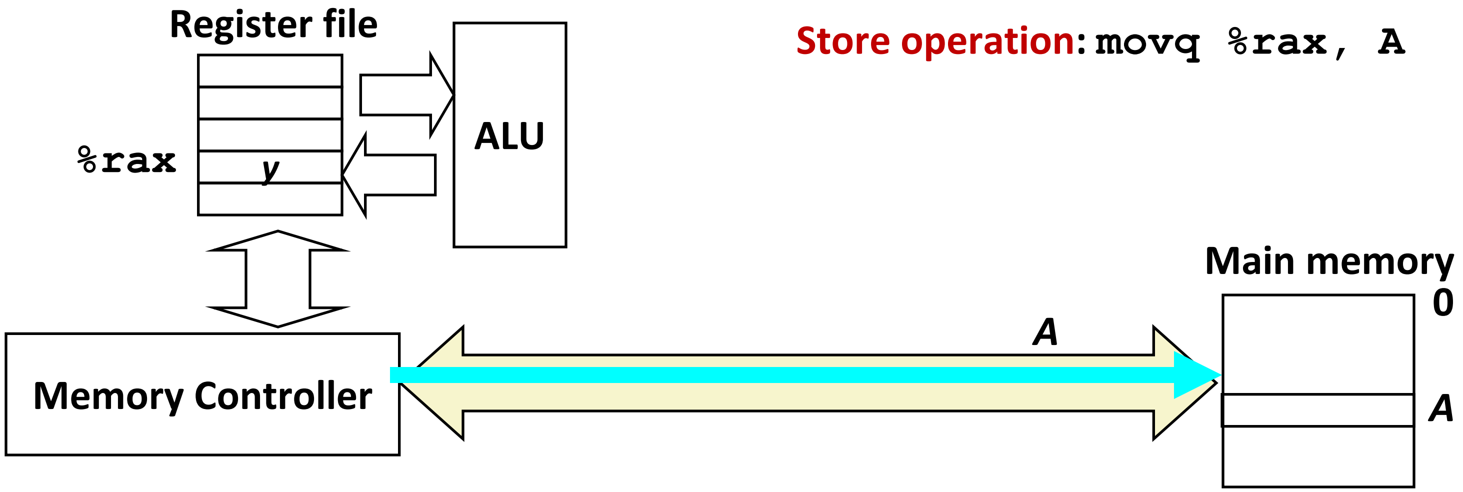
Memory Read Transaction (3)

- CPU reads word x from the bus and copies it into register **`%rax`**.



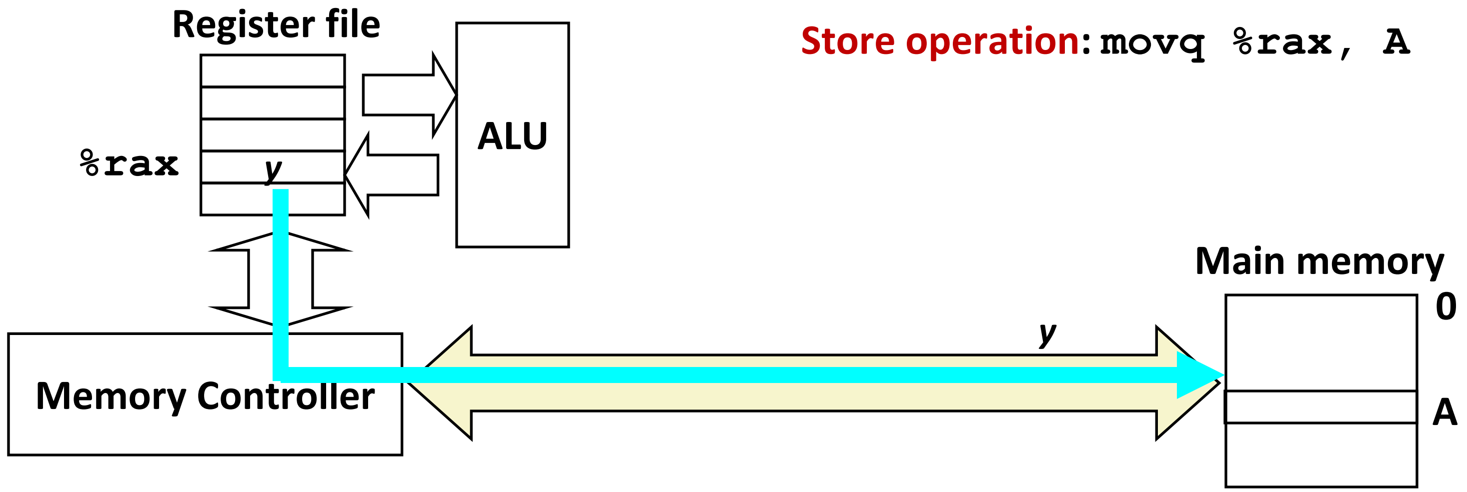
Memory Write Transaction (1)

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



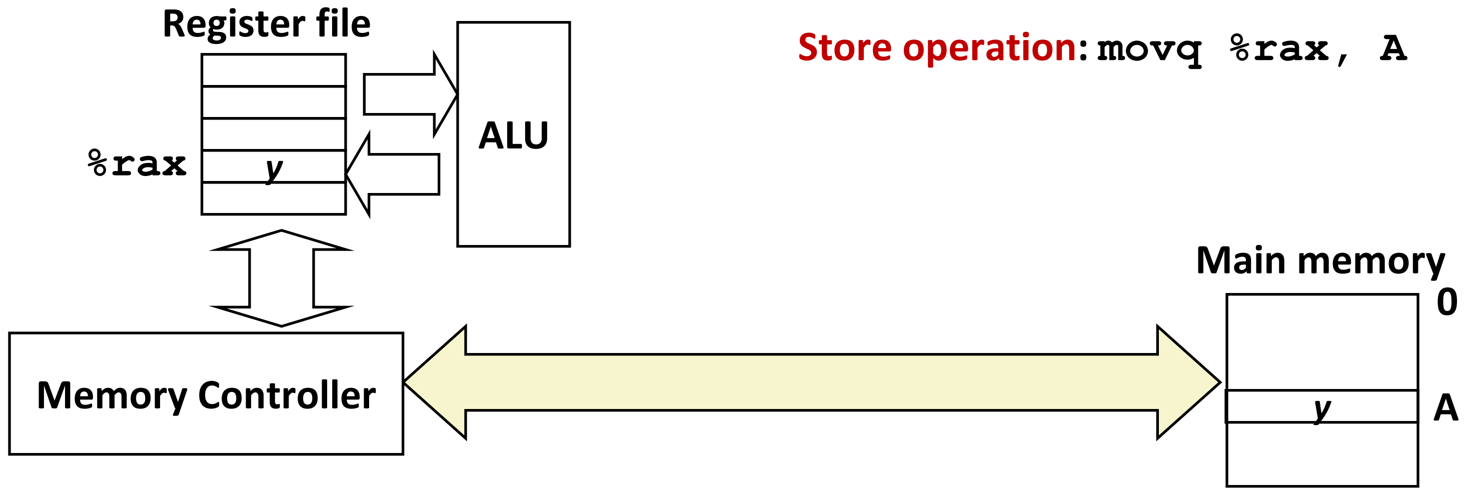
Memory Write Transaction (2)

- 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.



Today

- The memory abstraction
- **RAM : main memory building block**
- Locality of reference
- The memory hierarchy
- Storage technologies and trends

Random-Access Memory (RAM)

- **Key features**
  - RAM is traditionally packaged as a chip.
    - or embedded as part of processor chip
  - Basic storage unit is normally a cell (one bit per cell).
  - Multiple RAM chips form a memory.

- **RAM comes in two varieties:**
  - SRAM (Static RAM)
  - DRAM (Dynamic RAM)

SRAM vs DRAM Summary

	Trans. per bit	**Access time**	**Needs refresh?**	**Needs EDC?**	**Cost**	**Applications**
SRAM	6 or 8	1x	No	Maybe	100x	Cache memories
DRAM	1	10x	Yes	Yes	1x	Main memories, frame buffers

EDC: Error detection and correction

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
  - Electrically eraseable PROM (EEPROM): electronic erase capability
  - Flash memory: EEPROMs, with partial (block-level) erase capability
    - Wears out after about 100,000 erasings
  - 3D XPoint (Intel Optane) & emerging NVMs
    - New materials

- **Uses for Nonvolatile Memories**
  - Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,…)
  - Solid state disks (replacing rotating disks)
  - Disk caches

Today

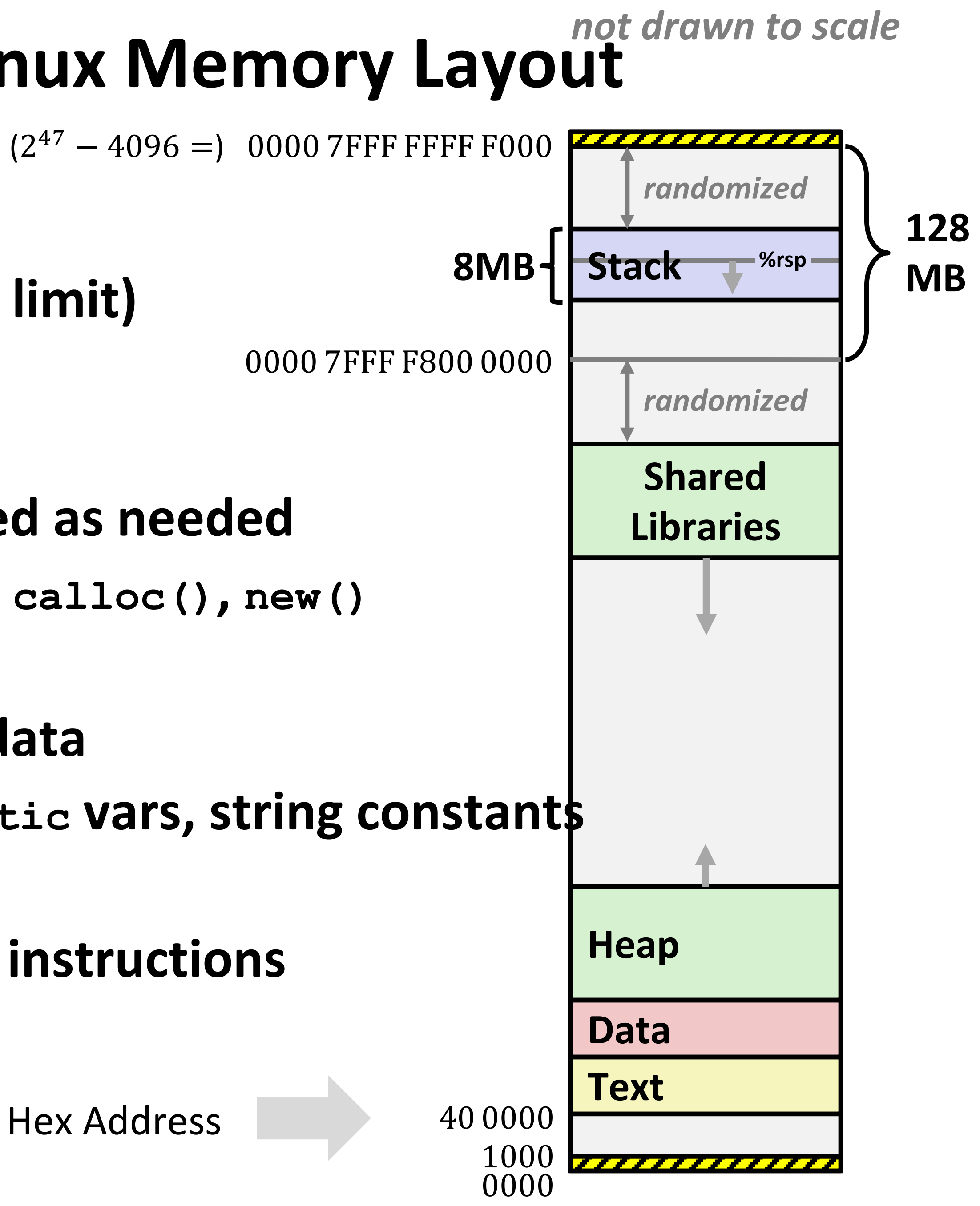
- The memory Abstraction
- DRAM : main memory building block
- **Locality of reference**
- The memory hierarchy
- Storage technologies and trends

Announcement

- Bomb Lab Dues this Thursday
- Cache Lab Comes out Soon
- Out-of-scope sections
  - 1.7, 1.8, 1.9
  - 2.4
  - 3.9.2, 3.11
  - 6.1

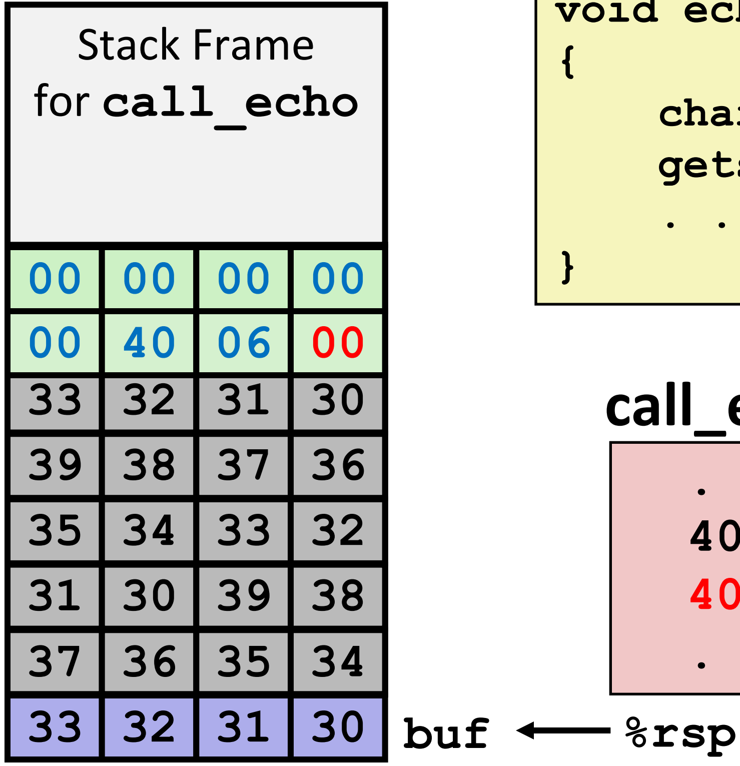
Reminder: x86-64 Linux Memory Layout

- **Stack**
  - **Runtime stack (8MB limit)**
  - **e.g., local variables**
- **Heap**
  - **Dynamically allocated as needed**
  - **When call `malloc()`, `calloc()`, `new()`**
- **Data**
  - **Statically allocated data**
  - **e.g., global vars, `static` vars, string constants**
- **Text / Shared Libraries**
  - **Executable machine instructions**
  - **Read-only**



Reminder: Buffer Overflow Stack Example #2

After call to gets



```
void echo()
{
      char buf[4];
      gets(buf);
      . . .
}
```

```
echo:
  subq   $0x18, %rsp
  movq   %rsp, %rdi
  call   gets
  . . .
```

call_echo:

```
  . . .
  4006be:   callq   4006cf <echo>
  4006c3:   add     $0x8,%rsp
  . . .
```

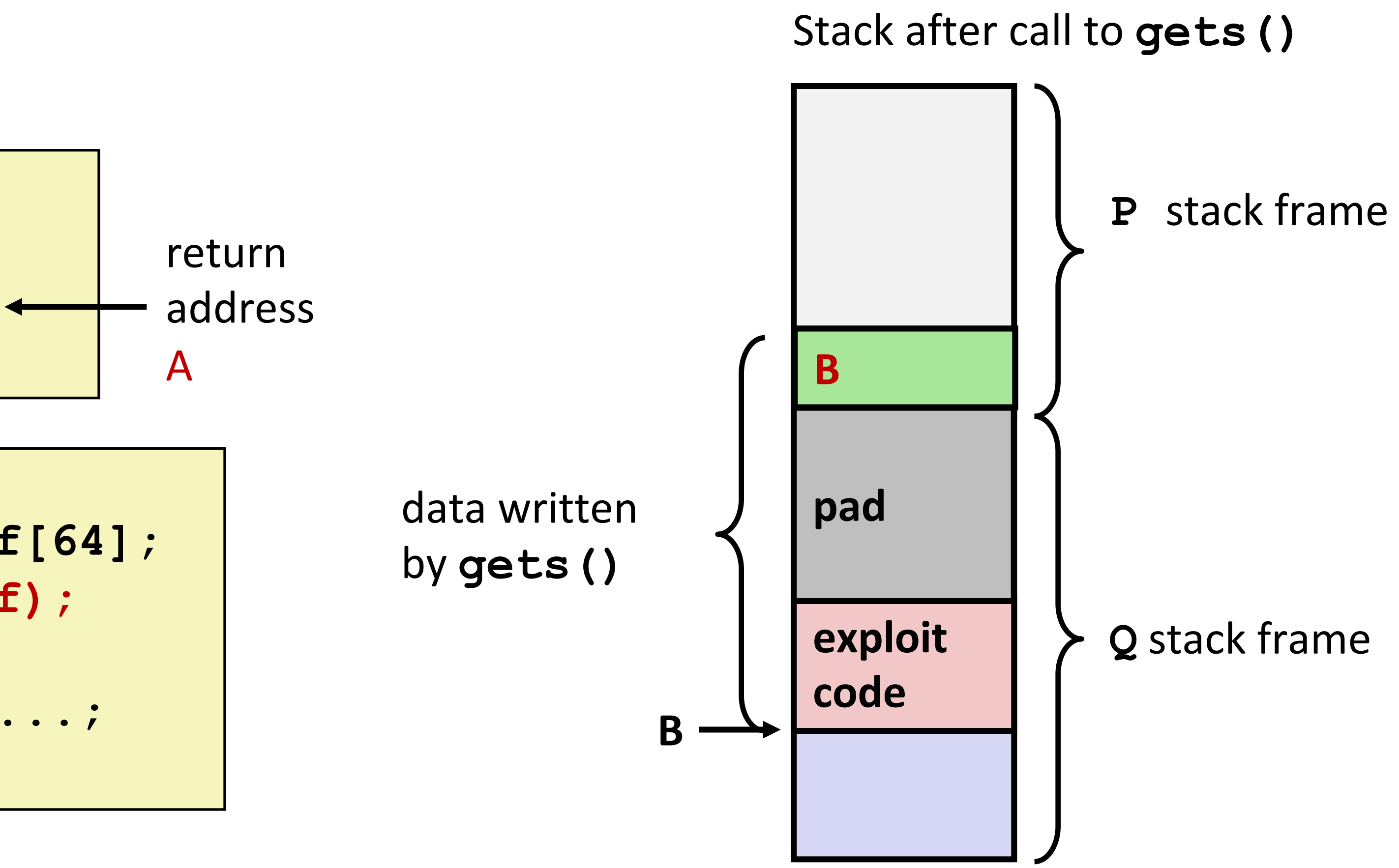
```
unix>./bufdemo-nsp
Type a string:0123456789012345678901234567890123
0123456789012345678901234567890123
Segmentation fault
```

Program "returned" to 0x0400600, and then crashed.

Reminder: Code Injection Attacks

```
void P(){
  Q();
  ...
}
```

```
int Q() {
  char buf[64];
  gets(buf);
  ...
  return ...;
}
```



- Input string contains byte representation of executable code
- Overwrite return address A with address of buffer B
- When Q executes `ret`, will jump to exploit code

Reminder: Avoid Overflow Vulnerabilities in Code

```
/* Echo Line */
void echo()
{
    char buf[4];
    fgets(buf, 4, stdin);
    puts(buf);
}
```

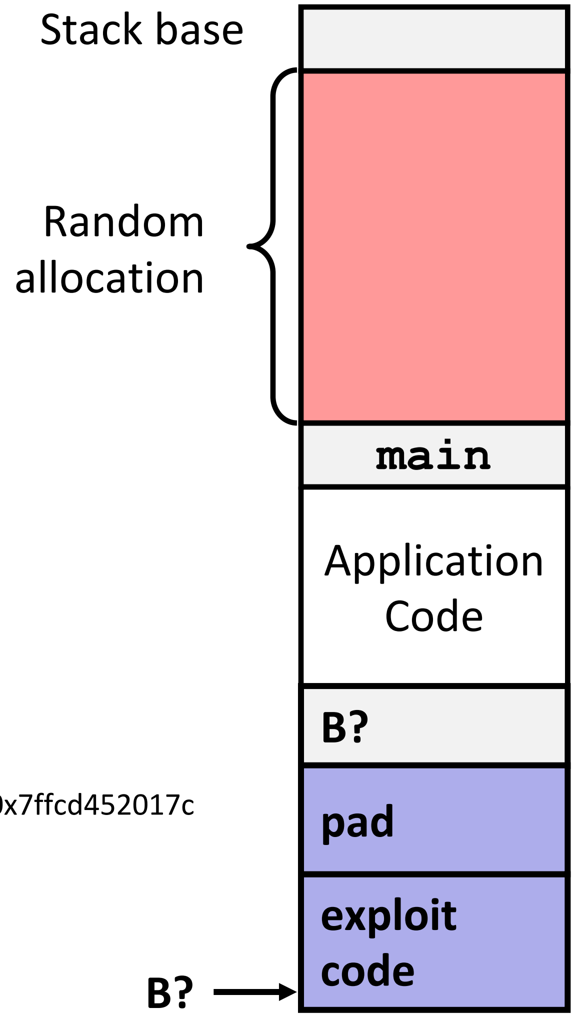
- For example, use library routines that limit string lengths
  - `fgets` instead of `gets`
  - `strncpy` instead of `strcpy`
  - Don't use `scanf` with `%s` conversion specification
    - Use `fgets` to read the string
    - Or use `%ns` where `n` is a suitable integer

Reminder: System-Level Protections Can Help

- Randomized stack offsets
  - At start of program, allocate random amount of space on stack
  - Shifts stack addresses for entire program
  - Makes it difficult for hacker to predict beginning of inserted code
  - e.g., 5 executions of memory allocation code

    - Stack repositioned each time program executes

local	0x7ffe4d3be87c	0x7fff75a4f9fc	0x7ffeadb7c80c	0x7ffeaea2fdac	0x7ffcd452017c



Reminder: Stack Canaries Can Help

- Idea
  - Place special value (“canary”) on stack just beyond buffer
  - Check for corruption before exiting function
- GCC Implementation
  - `-fstack-protector`
  - Now the default (disabled earlier)

```
unix>./bufdemo-sp
Type a string:0123456
0123456
```

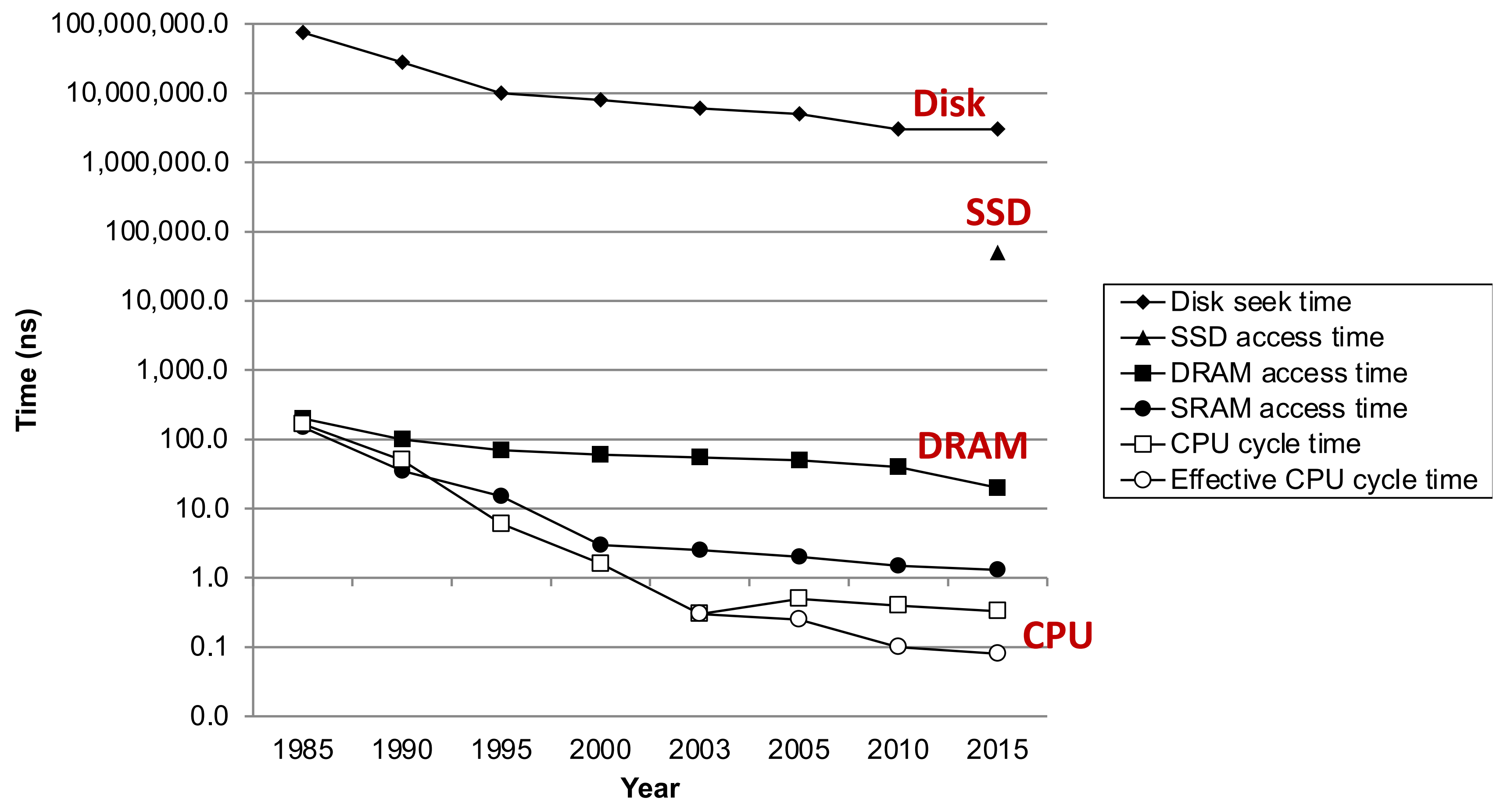
```
unix>./bufdemo-sp
Type a string:012345678
*** stack smashing detected ***
```

Today

- The memory Abstraction
- DRAM : main memory building block
- **Locality of reference**
- The memory hierarchy
- Storage technologies and trends

The CPU-Memory Gap

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



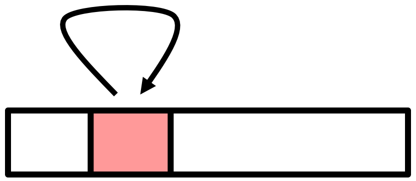
Locality to the Rescue!

The key to bridging this CPU-Memory gap is an important property of computer programs known as locality.

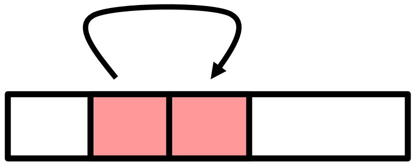
Locality

- **Principle of Locality: Many 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;
```

Spatial or Temporal Locality?

- **Data references**
  - Reference array elements in succession (stride-1 reference pattern). **spatial**
  - Reference variable `sum` each iteration. **temporal**
- **Instruction references**
  - Reference instructions in sequence. **spatial**
  - Cycle through loop repeatedly. **temporal**

Qualitative Estimates of Locality

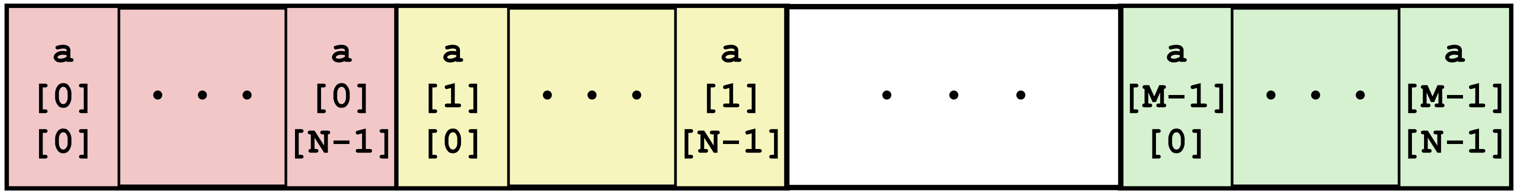
- **Claim:** Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.
- **Question:** Does this function have good locality with respect to array `a`?

Hint: array layout is row-major order

Answer: yes

```
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

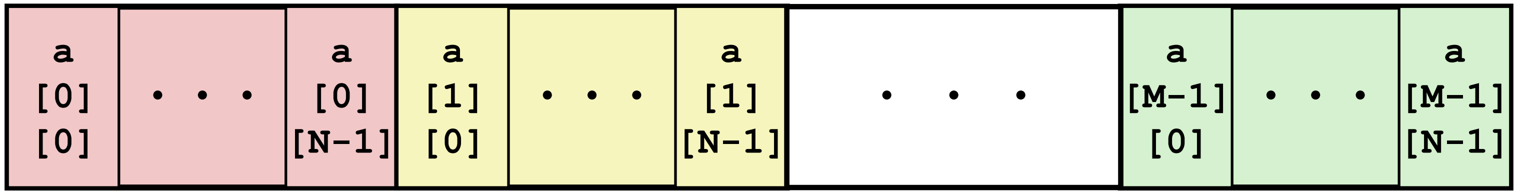
- **Question:** Does this function have good locality with respect to array a?

```
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;
}
```

Answer: no, unless…

M is very small



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;
}
```

Answer: make j the inner loop

Today

- The memory abstraction
- RAM : main memory building block
- Locality of reference
- **The memory hierarchy**
- Storage technologies and trends

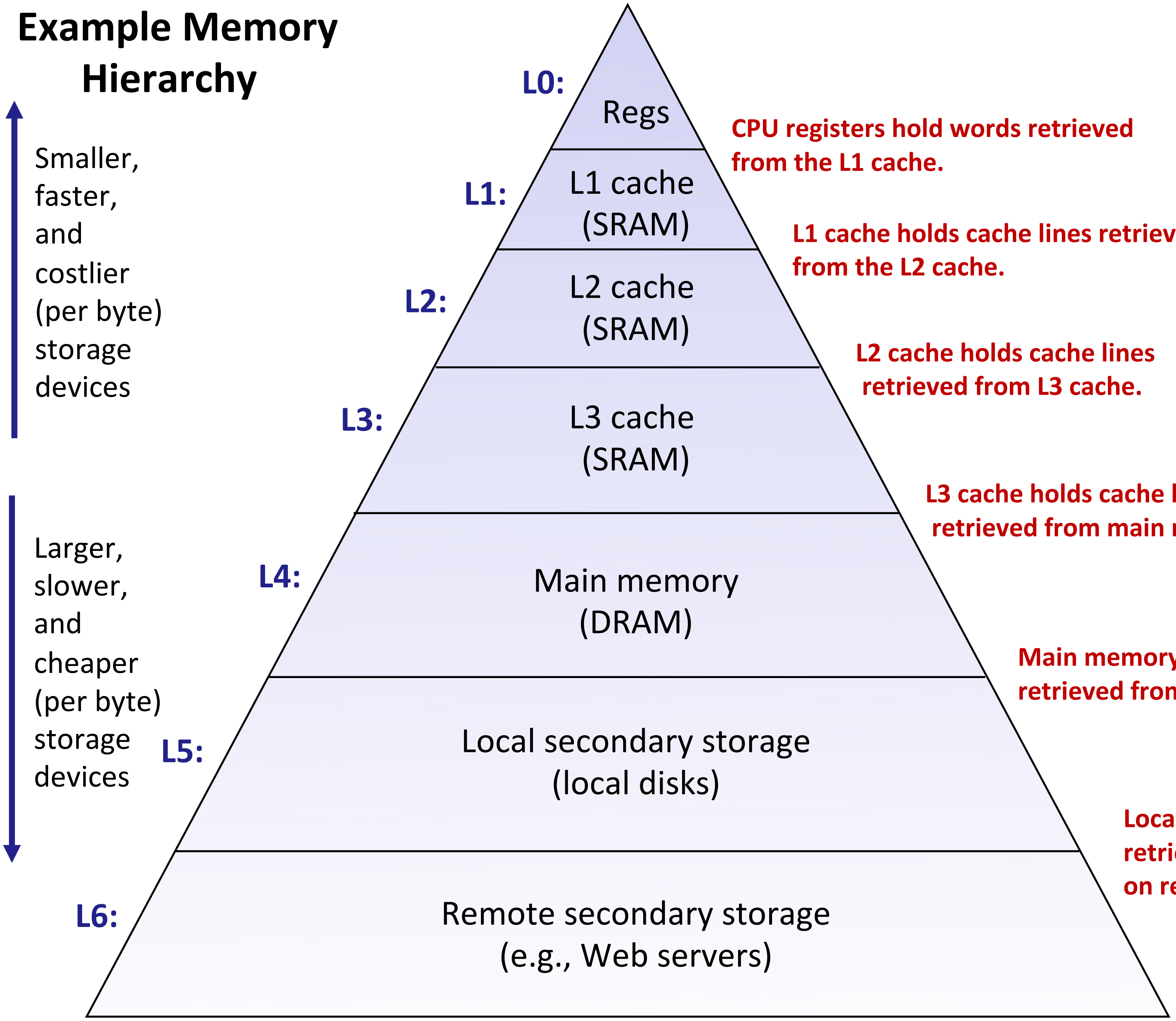
Memory Hierarchies

- **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 properties complement each other well for many types of programs.**

- **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.

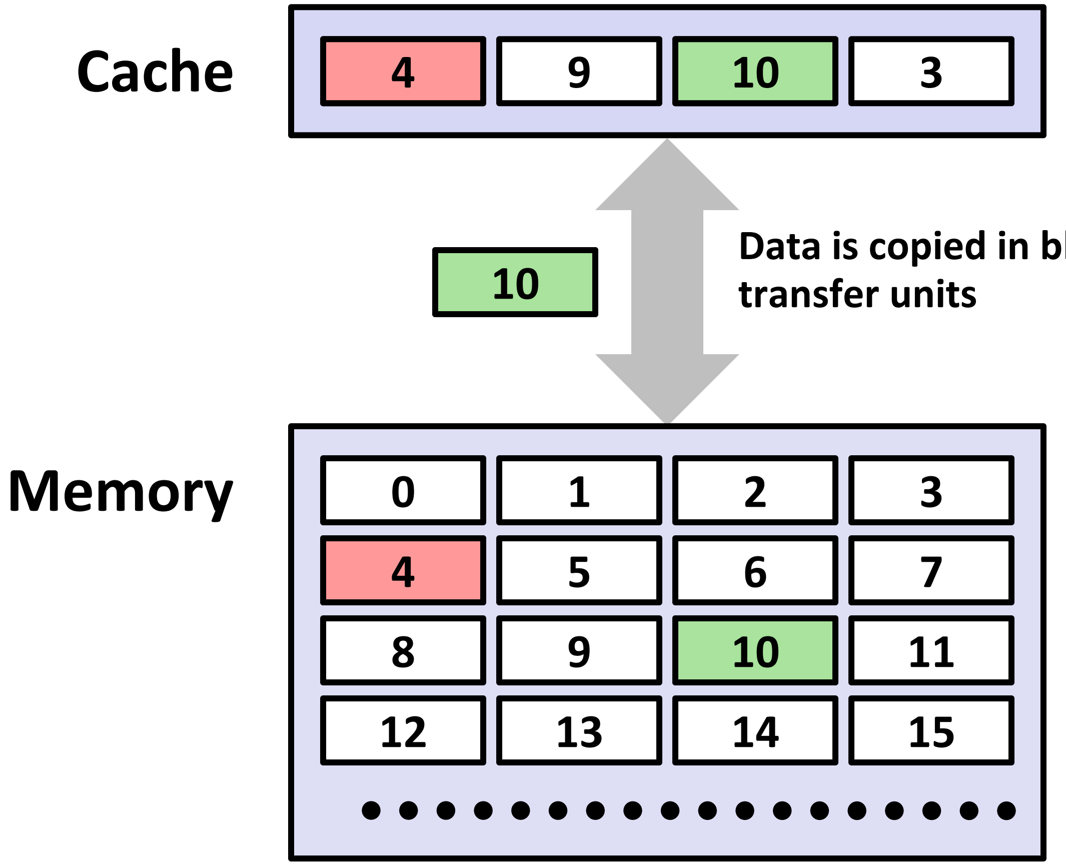
Main memory holds disk blocks retrieved from local disks.

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 (Ideal):* **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

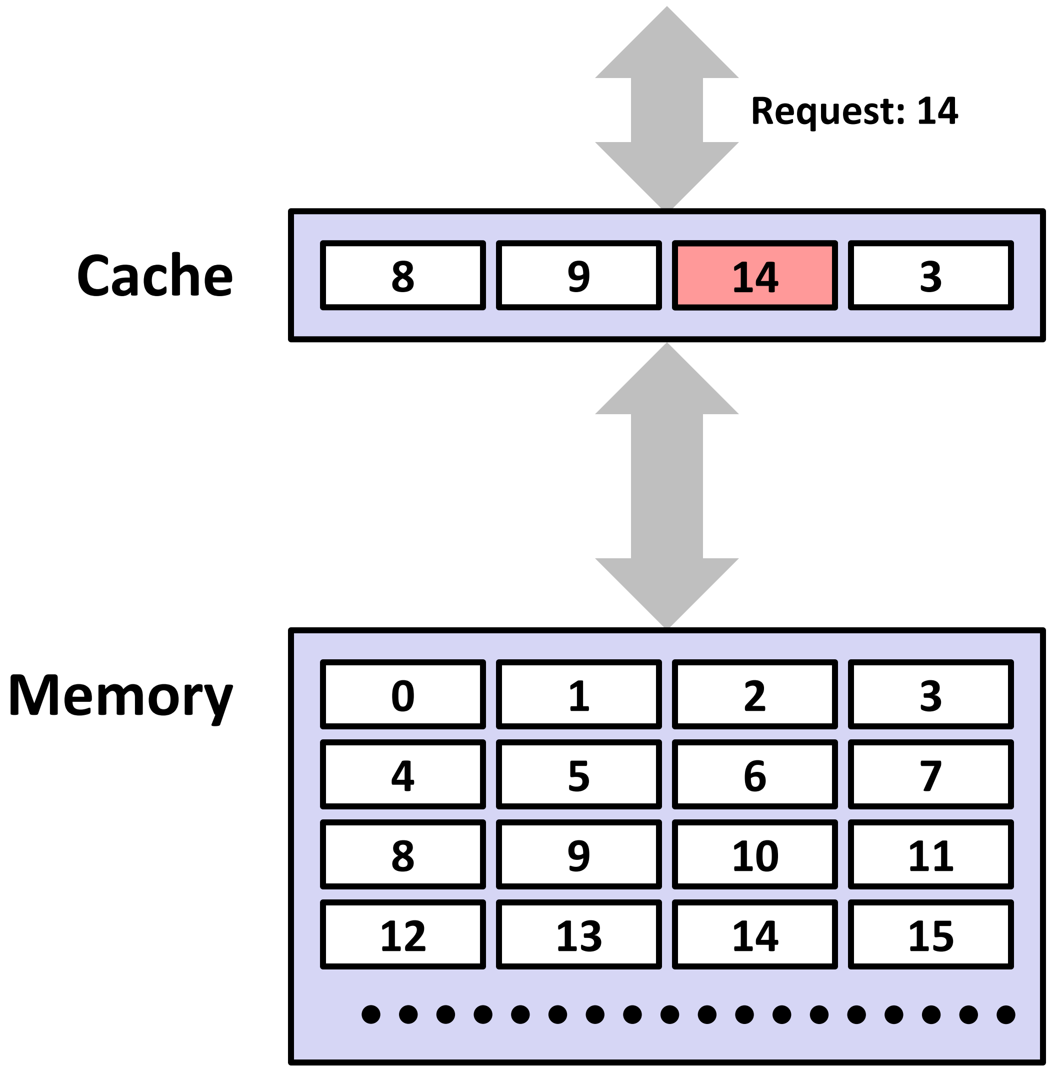


Smaller, faster, more expensive memory caches a subset of the blocks

Data is copied in block-sized transfer units

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

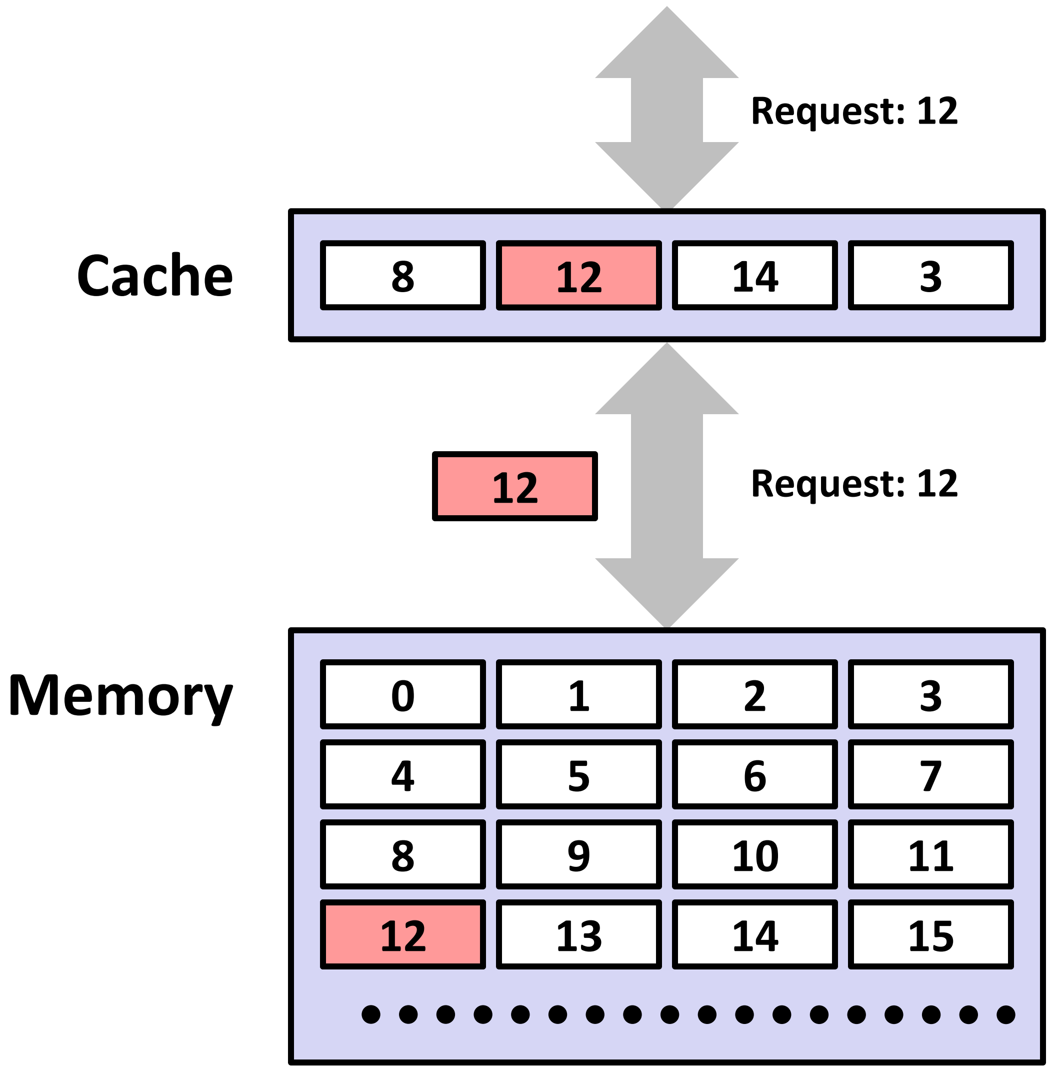
General Cache Concepts: Hit



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: 3 Types of Cache Misses

- **Cold (compulsory) miss**
  - Cold misses occur because the cache starts empty and this is the first reference to the block.
- **Capacity miss**
  - Occurs when the set of active cache blocks (working set) is larger than the cache.
- **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.

Examples of Caching in the Mem. Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 byte 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

Today

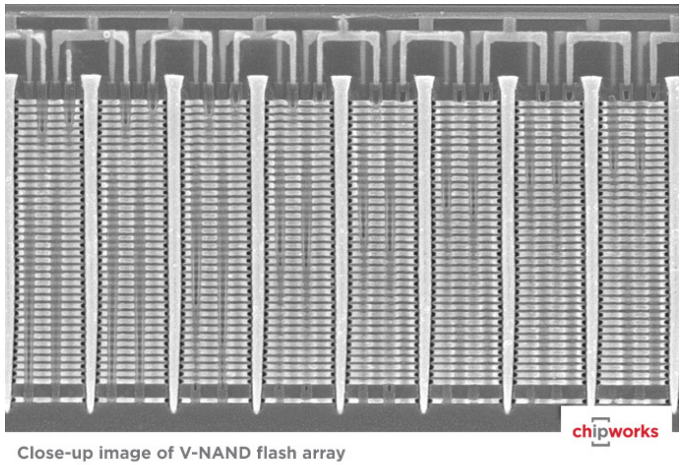
- The memory abstraction
- RAM : main memory building block
- Locality of reference
- The memory hierarchy
- **Storage technologies and trends**

Storage Technologies

- Magnetic Disks

- Store on magnetic medium
- Electromechanical access

Nonvolatile (Flash) Memory



Close-up image of V-NAND flash array

charge

Implemented with 3-D structure

- 100+ levels of cells
- 3-4 bits data per cell

What's Inside A Disk Drive?

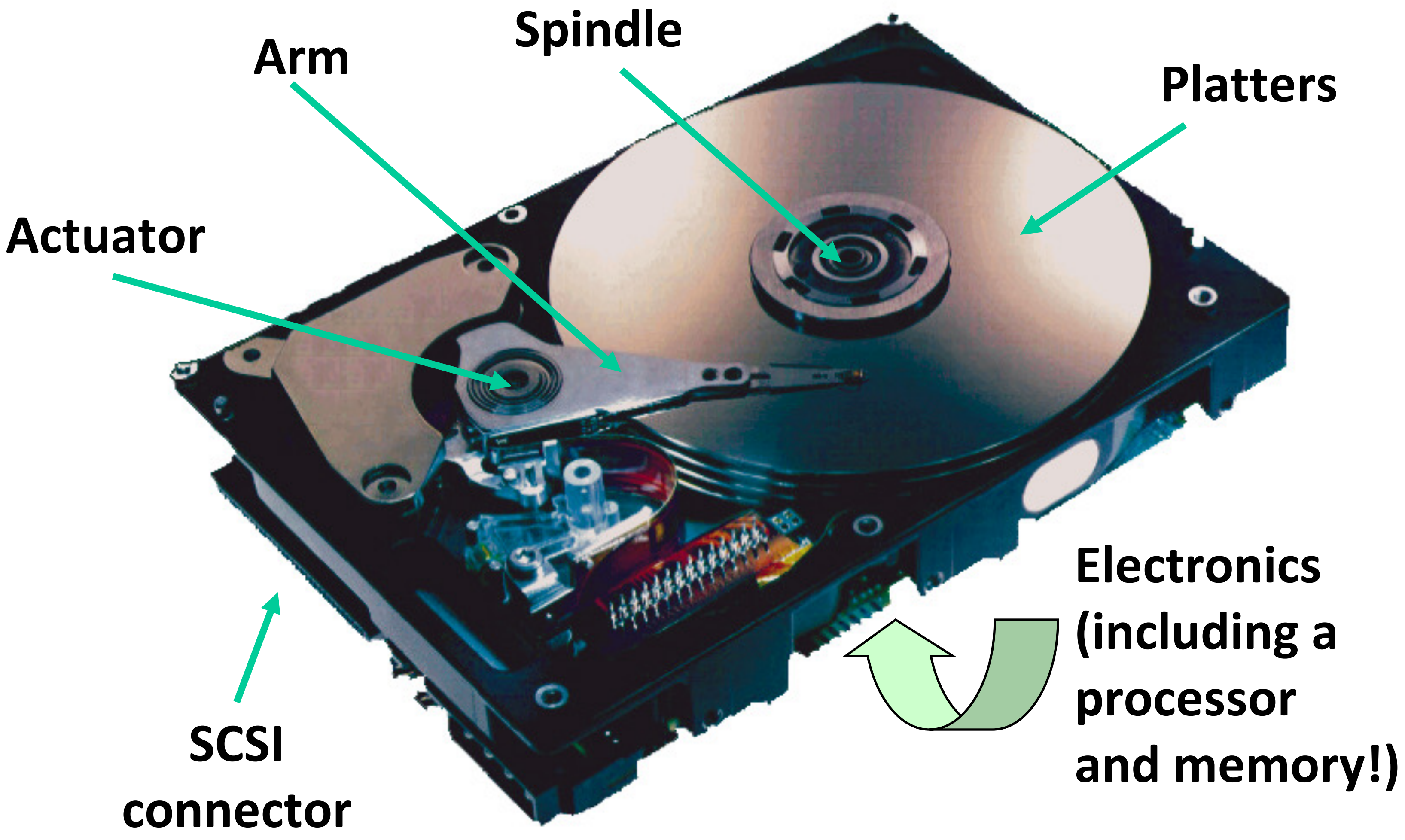
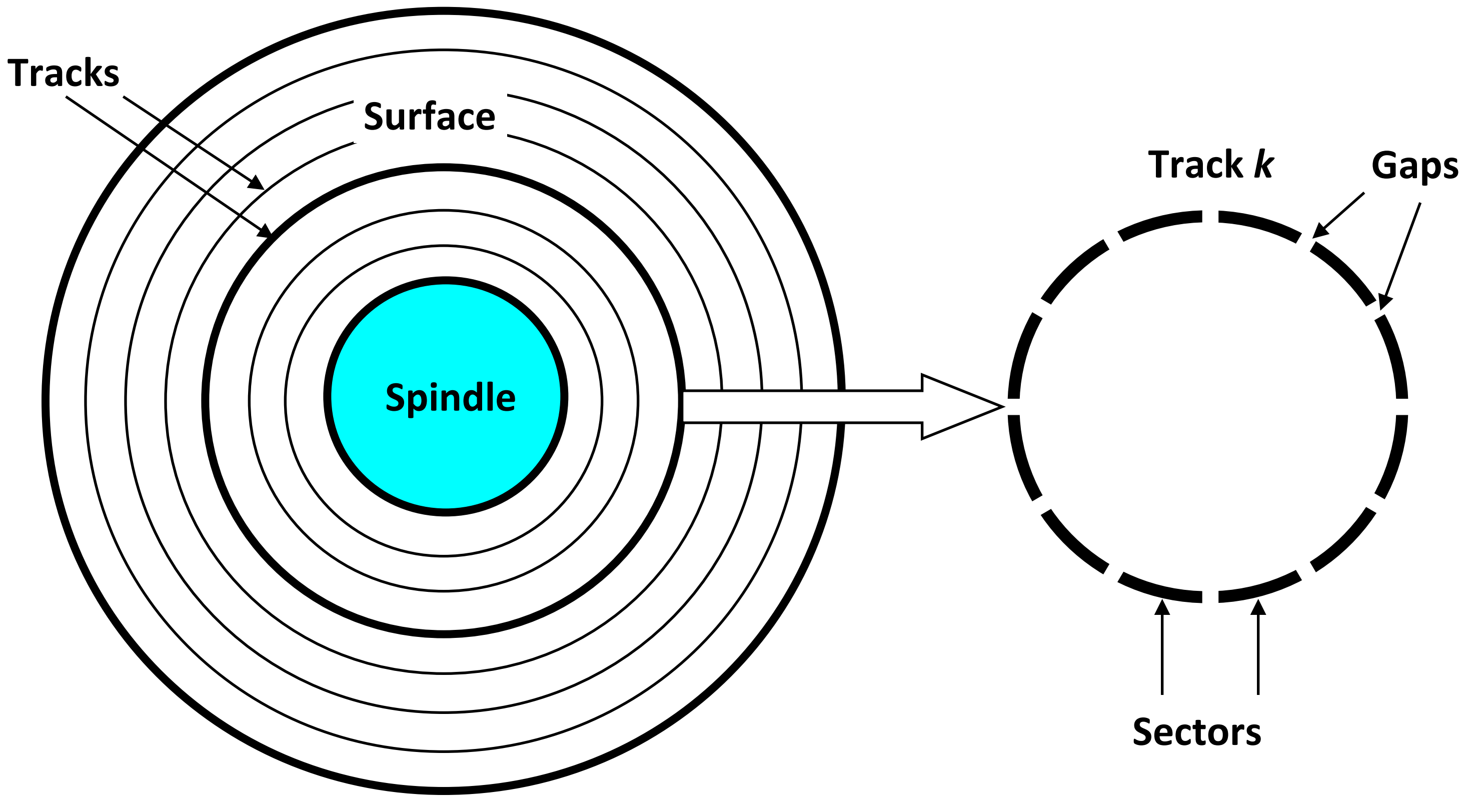


Image courtesy of Seagate Technology

哨芝士 bilibili

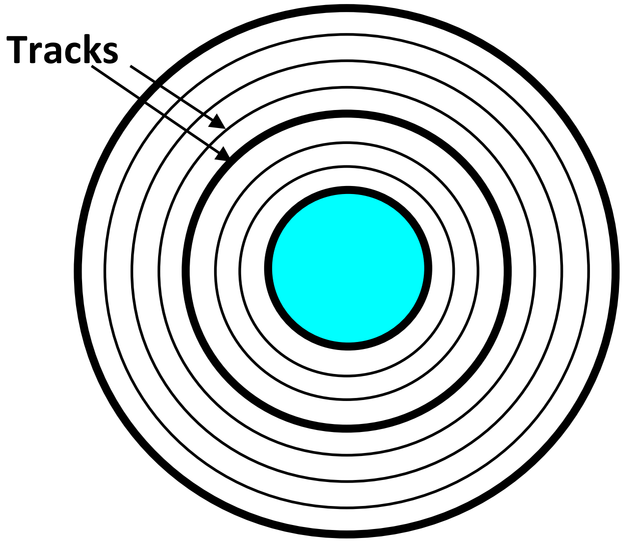
Disk Geometry

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



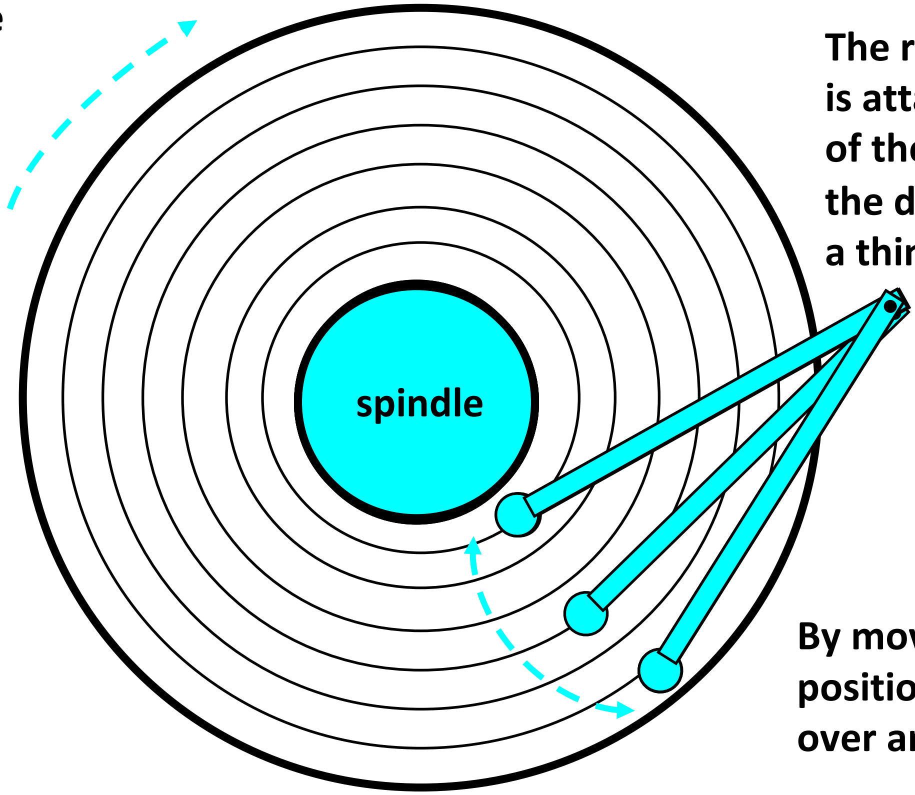
Disk Capacity

- **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 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 Operation (Single-Platter View)

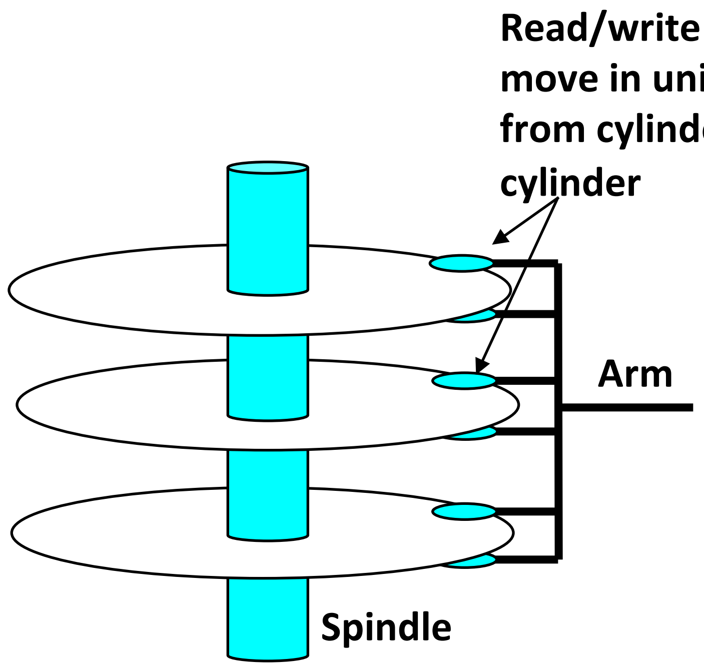
The disk surface spins at a fixed rotational rate



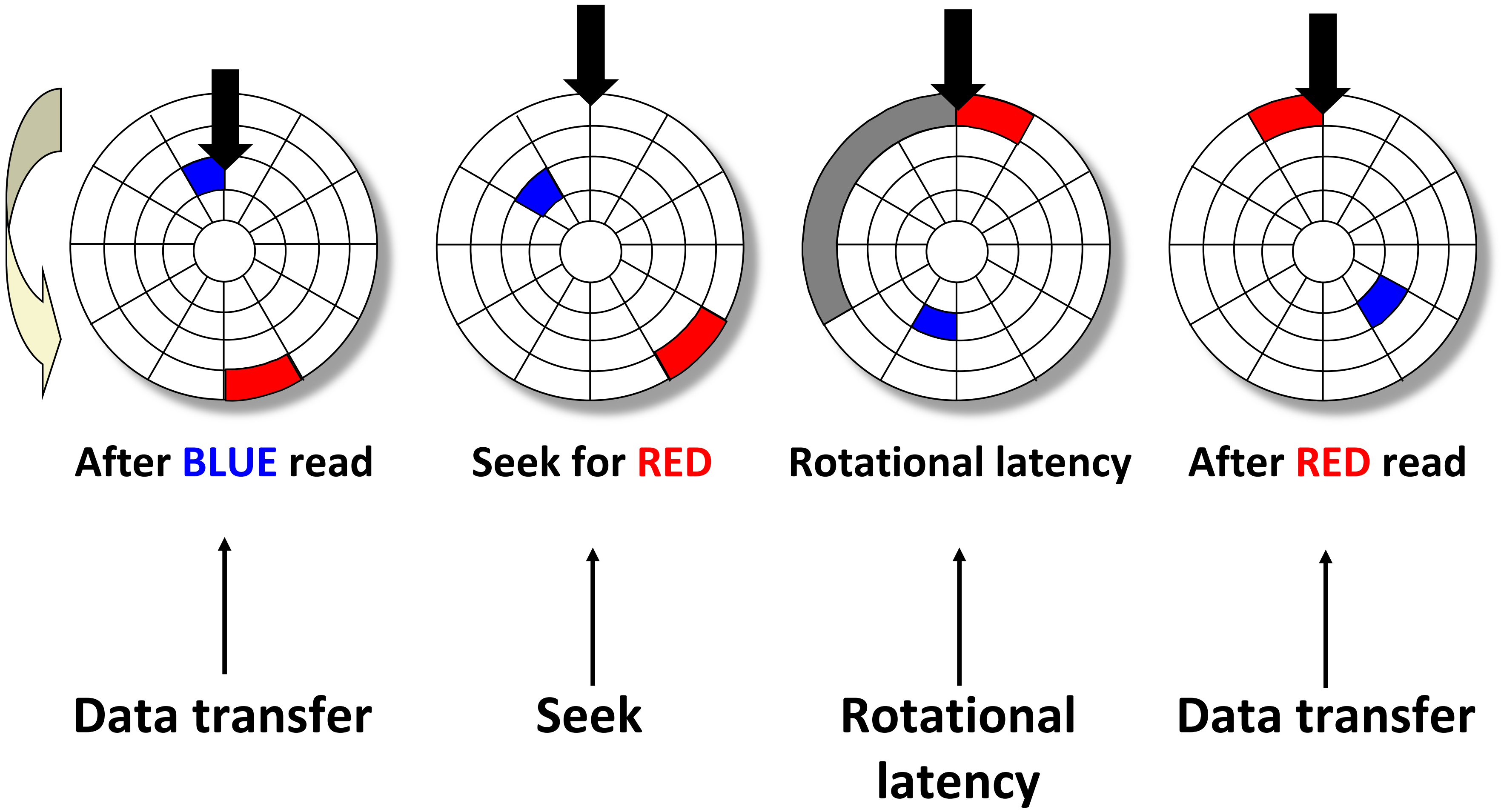
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

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

Disk Operation (Multi-Platter View)



Disk Access – Service Time Components



Disk Access Time

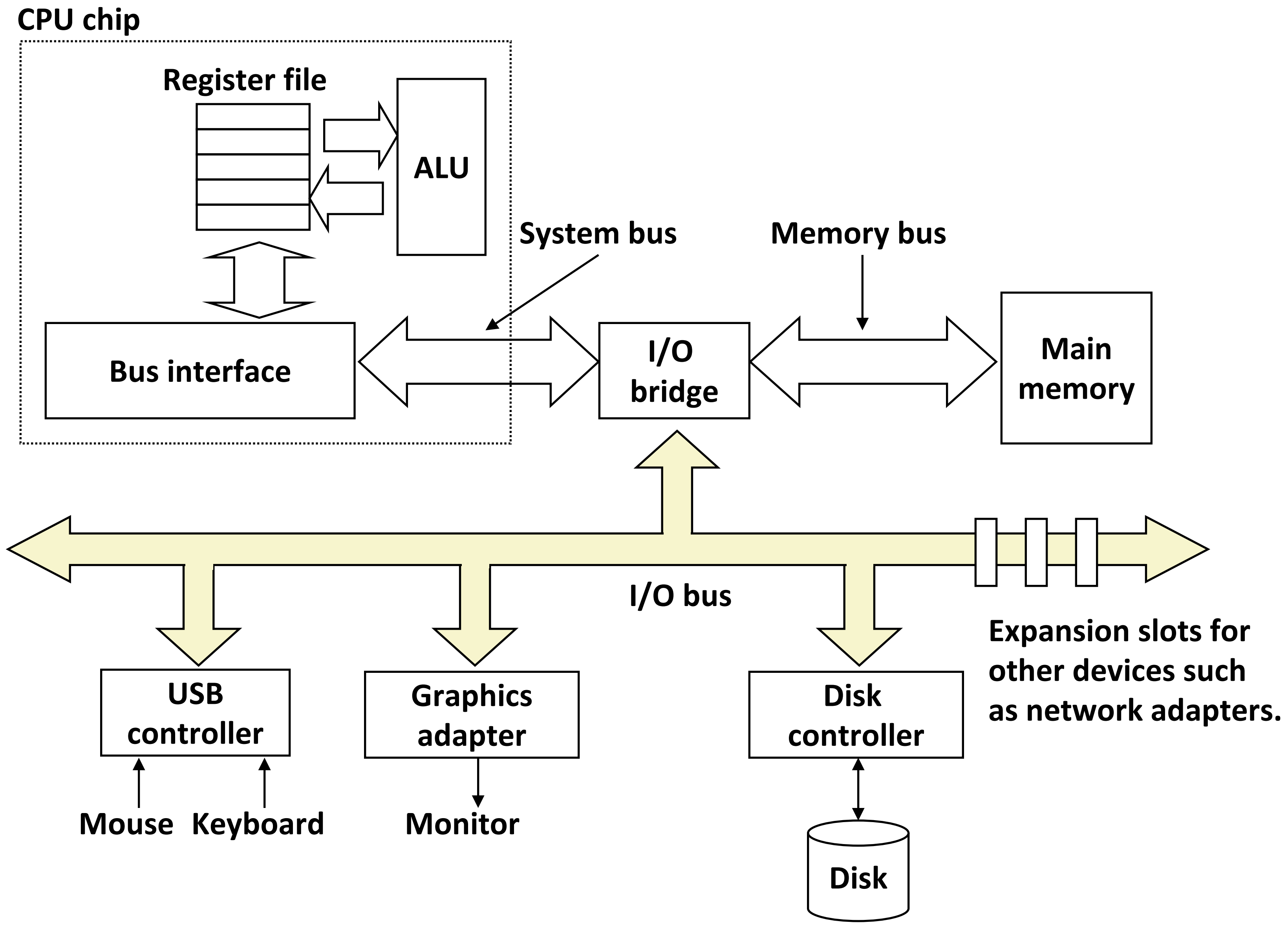
- **Average time to access some target sector approximated by:**
  - $T_{access} = T_{avg\ seek} + T_{avg\ rotation} + T_{avg\ transfer}$
- **Seek time ($T_{avg\ seek}$)**
  - Time to position heads over cylinder containing target sector.
  - Typical $T_{avg\ seek}$ is 3—9 ms
- **Rotational latency ($T_{avg\ rotation}$)**
  - Time waiting for first bit of target sector to pass under r/w head.
  - $T_{avg\ rotation}$ = 1/2 x 1/RPMs x 60 sec/1 min
  - Typical rotational rate = 7,200 RPMs
- **Transfer time ($T_{avg\ transfer}$)**
  - Time to read the bits in the target sector.
  - $T_{avg\ transfer}$ = 1/RPM x 1/(avg # sectors/track) x 60 secs/1 min

time for one rotation (in minutes)   fraction of a rotation to be read

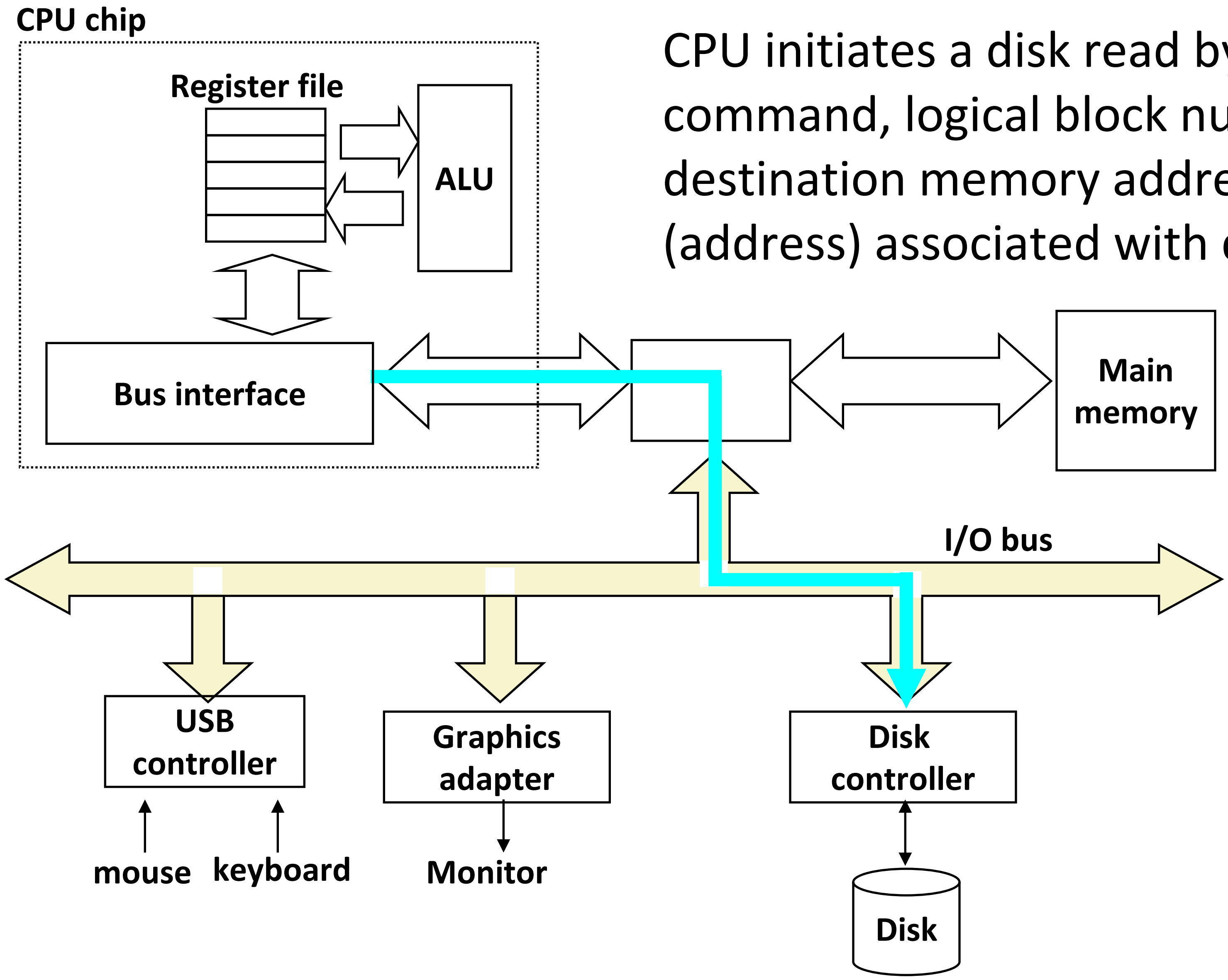
Disk Access Time Example

- **Given:**
  - Rotational rate = 7,200 RPM
  - Average seek time = 9 ms
  - Avg # sectors/track = 400
- **Derived:**
  - $T_{\text{avg rotation}}$ = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms
  - $T_{\text{avg transfer}}$ = 60/7200 x 1/400 x 1000 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.
  - ***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.***

I/O Bus

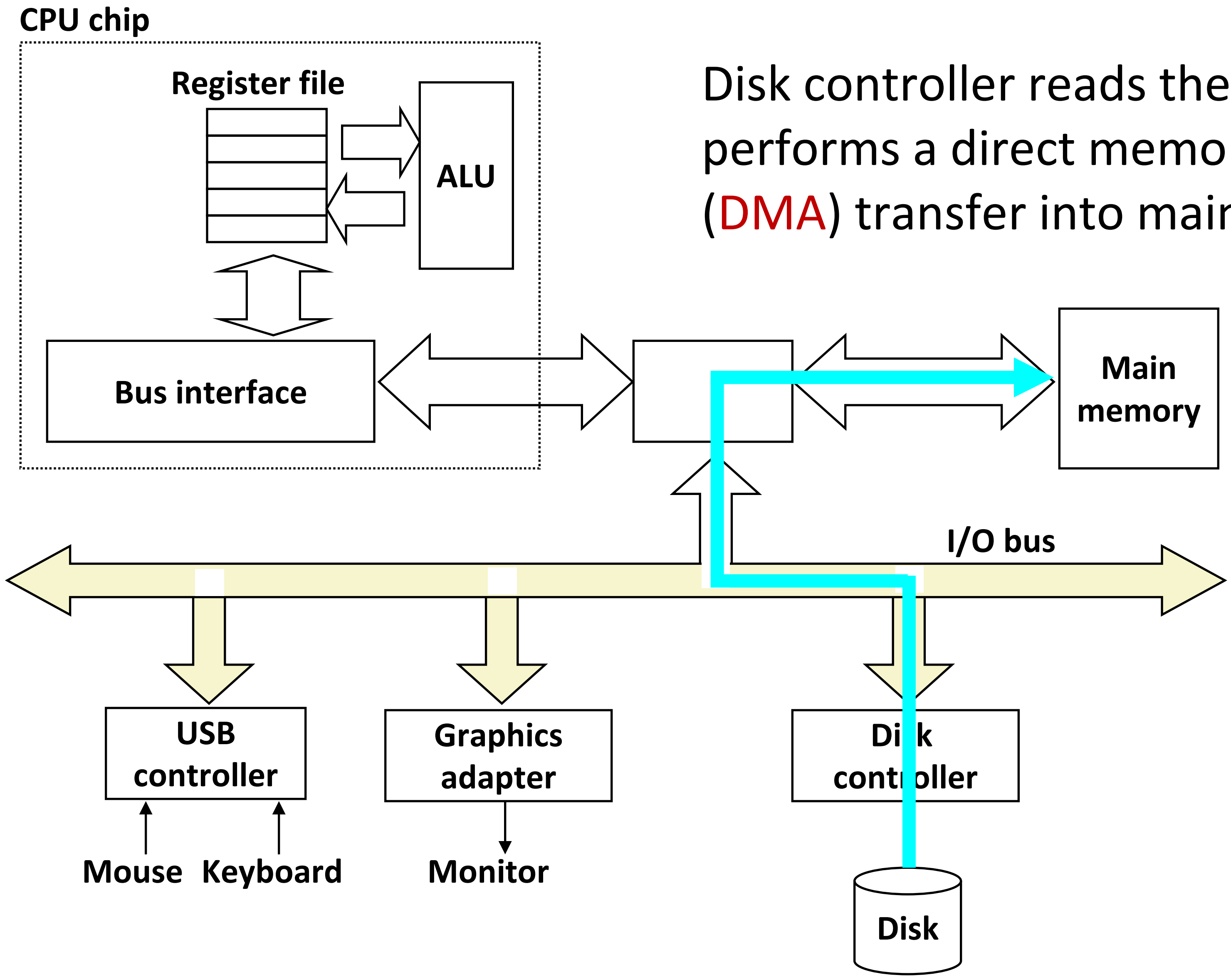

CPU chip
Register file
ALU
System bus
Memory bus
Bus interface
I/O bridge
Main memory
I/O bus
Expansion slots for other devices such as network adapters.
USB controller
Graphics adapter
Disk controller
Mouse
Keyboard
Monitor
Disk

Reading a Disk Sector (1)



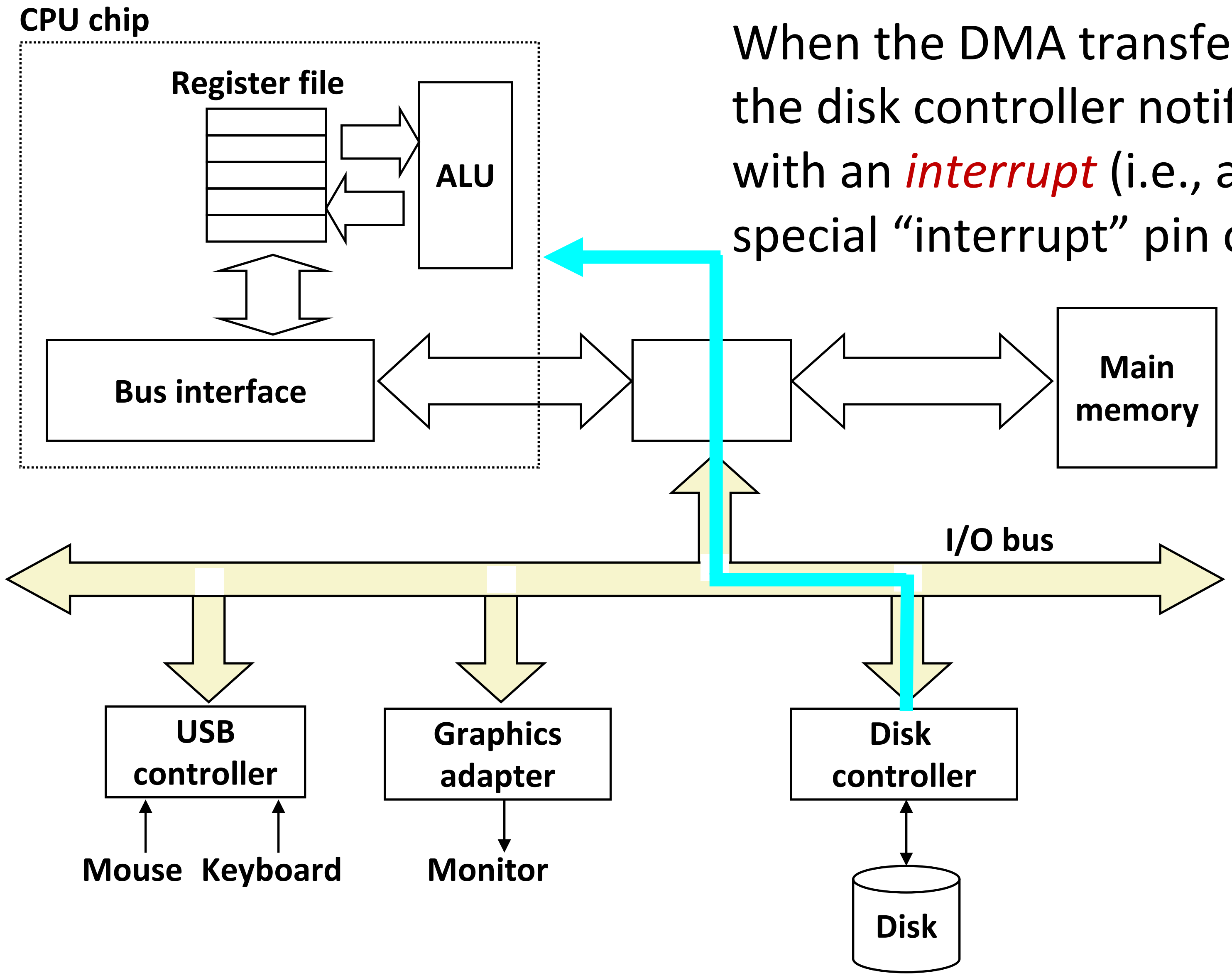
CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

Reading a Disk Sector (2)



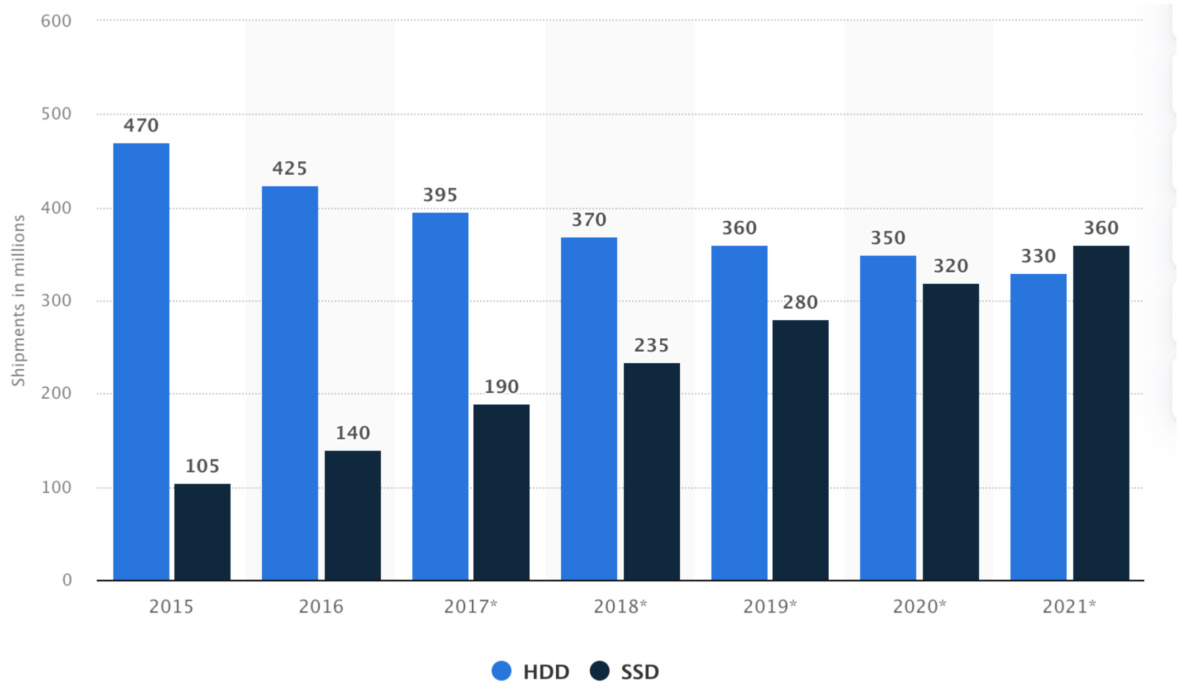
Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

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).

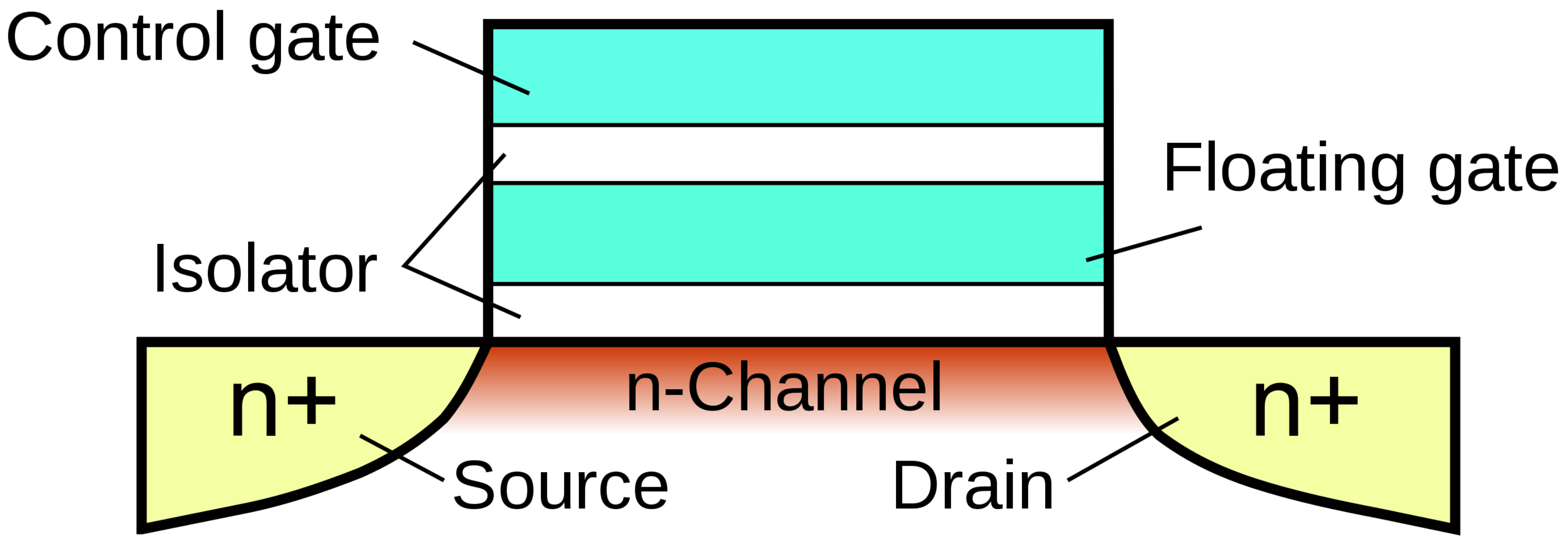
Solid State Disks (SSDs)



Shipments of hard and solid state disk (HDD/SSD) drives worldwide from 2015 to 2021

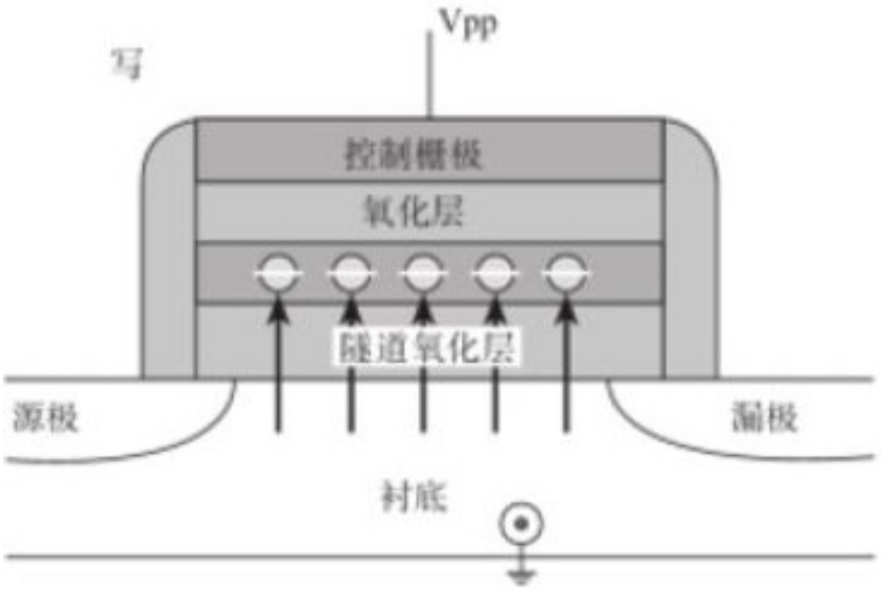
Solid State Disks (SSDs)

- **Memory Unit： Floating-gate MOSFET**

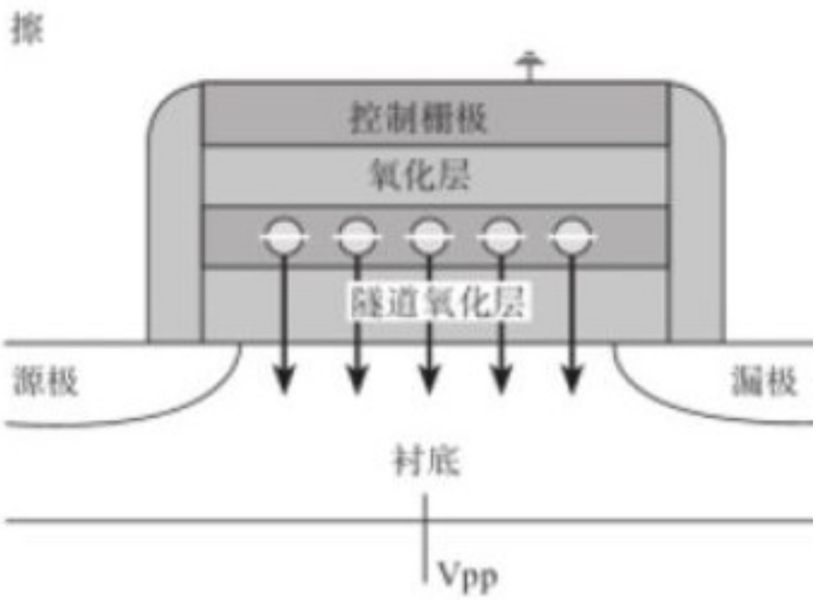


Solid State Disks (SSDs)

- Memory Unit： Floating-gate MOSFET



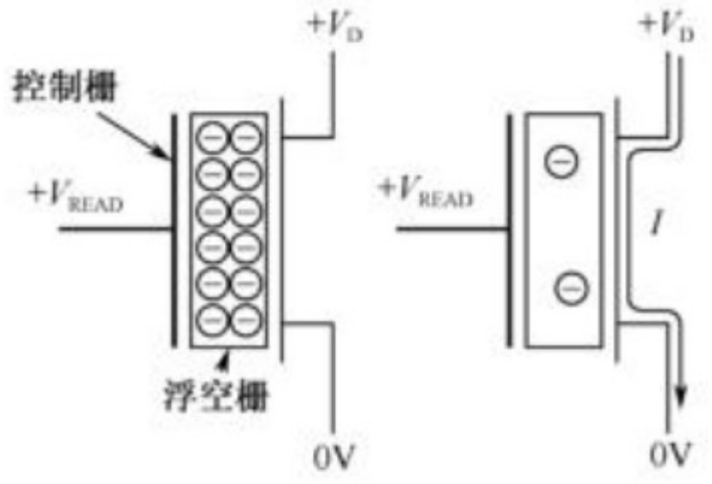
写入



擦除

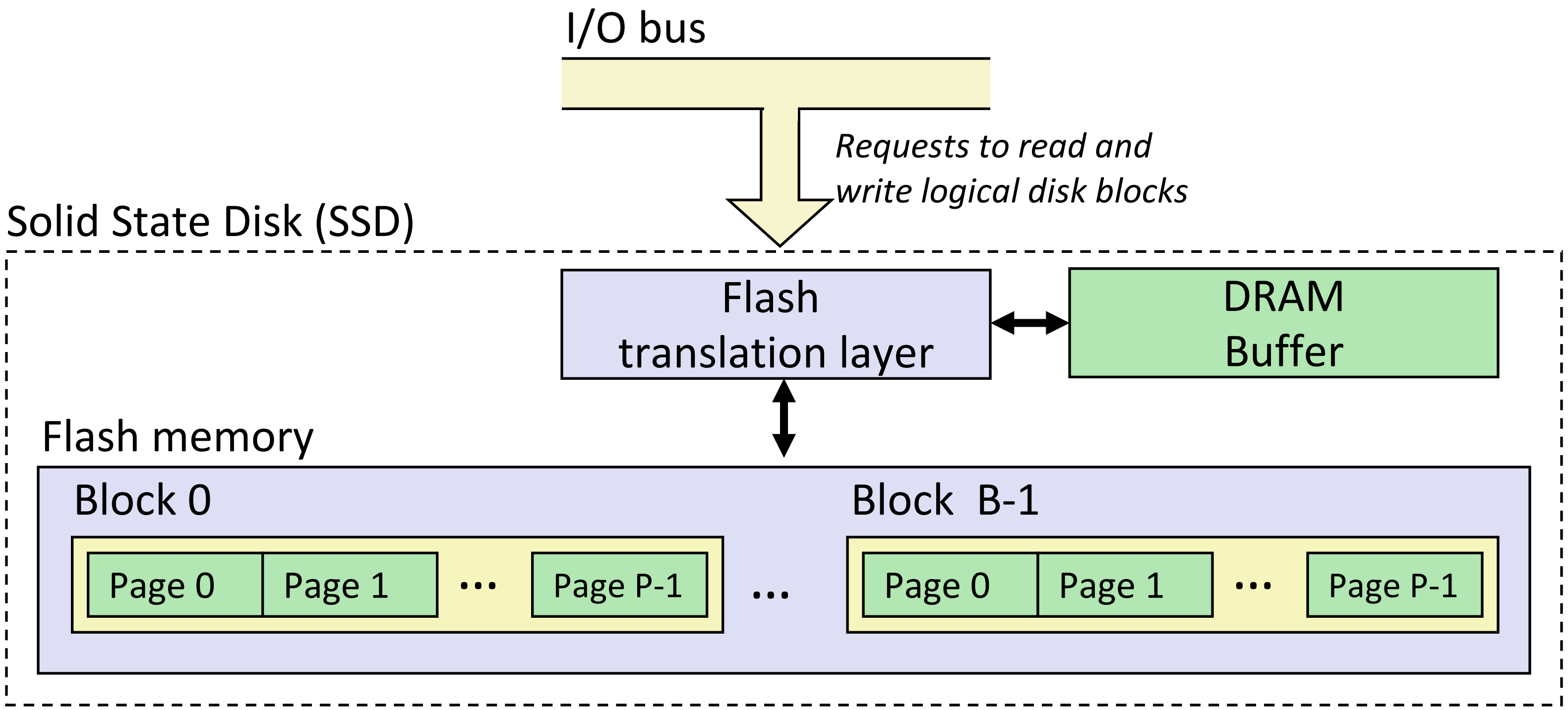
Solid State Disks (SSDs)

- Memory Unit: Floating-gate MOSFET



读取

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 10,000 repeated writes.

SSD Performance Characteristics

- **Benchmark of Samsung 970 EVO Plus**

  https://ssd.userbenchmark.com/SpeedTest/711305/Samsung-SSD-970-EVO-Plus-250GB

Sequential read throughput	2,221 MB/s	Sequential write tput	1,912 MB/s
Random read throughput	61.7 MB/s	Random write tput	165 MB/s
Random DQ throughput	947 MB/s	Random DQ write	1028 MB/s

  - Common theme in the memory hierarchy
  - DQ = deep queue, issuing many concurrent reads (latency hurts!)
- **Random writes are tricky**
  - Erasing a block takes a long time (~1 ms), but the SSD has a pool of pre-erased blocks
  - Modifying a block page requires all other pages to be copied to new block.
  - But the SSD has a write cache that it accumulates writes into…

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. Samsung 940 EVO Plus guarantees 600 writes/byte of writes before they wear out
    - Controller migrates data to minimize wear level
  - In 2022, about 4-5 times more expensive per byte
    - And, relative cost will keep dropping
- **Where are are rotating disks still used?**
  - Bulk storage – video, huge datasets / databases, etc.
  - Cheap storage – desktops.

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.**
- **Flash memory progress outpacing all other memory and storage technologies (DRAM, SRAM, magnetic disk)**
  - Able to stack cells in three dimensions

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