

# CS 33

## Signals part 3; Memory Hierarchy II

# Interrupted System Calls

- What if a signal is handled before the system call completes?

~~1) invoke handler, then resume system call~~  
~~• not clear if system call should be resumed~~  
or

- 2) invoke handler, then return from system call prematurely
- if one or more pieces were completed, return total number of bytes transferred
  - otherwise return “interrupted” error

# Interrupted System Calls: Non-Lengthy Case

```
while (read(fd, buffer, buf_size) == -1) {  
    if (errno == EINTR) {  
        /* interrupted system call – try again */  
        continue;  
    }  
    /* the error is more serious */  
    perror("big trouble");  
    exit(1);  
}
```

# Quiz 1

```
int ret;  
char buf[128];  
  
fillbuf(buf);  
  
ret = write(1, buf, 128);
```

- **The value of ret is:**
  - a) either -1 or 128
  - b) either -1, 0, or 128
  - c) any integer in the range [-1, 128]

# Interrupted System Calls: Lengthy Case

```
char buf[BSIZE];
fillbuf(buf);
long remaining = BSIZE;
char *bptr = buf;
for ( ; ; ) {
    long num_xfrd = write(fd,
        bptr, remaining);
    if (num_xfrd == -1) {
        if (errno == EINTR) {
            /* interrupted early */
            continue;
        }
        perror("big trouble");
        exit(1);
    }
    if (num_xfrd < remaining) {
        /* interrupted after the
           first step */
        remaining -= num_xfrd;
        bptr += num_xfrd;
        continue;
    }
    /* success! */
    break;
}
```

# Asynchronous Signals (1)

```
main( ) {  
    void handler(int);  
    signal(SIGINT, handler);  
  
    ...    /* long-running buggy code */  
  
}  
  
void handler(int sig) {  
    ...    /* clean up */  
    exit(1);  
}
```

# Asynchronous Signals (2)

```
computation_state_t  state;    long_running_procedure( ) {  
                                while (a_long_time) {  
main( ) {                      update_state(&state);  
    void handler(int);         compute_more( );  
                                }  
    signal(SIGINT, handler);    }  
  
    long_running_procedure( );  void handler(int sig) {  
}                               display(&state);  
                                }  
                                }
```

# Asynchronous Signals (3)

```
main( ) {  
    void handler(int);  
  
    signal(SIGINT, handler);  
  
    ... /* complicated program */  
  
    myput("important message\n");  
  
    ... /* more program */  
  
}  
  
void handler(int sig) {  
    ... /* deal with signal */  
  
    myput("equally important "  
        "message\n");  
}
```



# Asynchronous Signals (4)

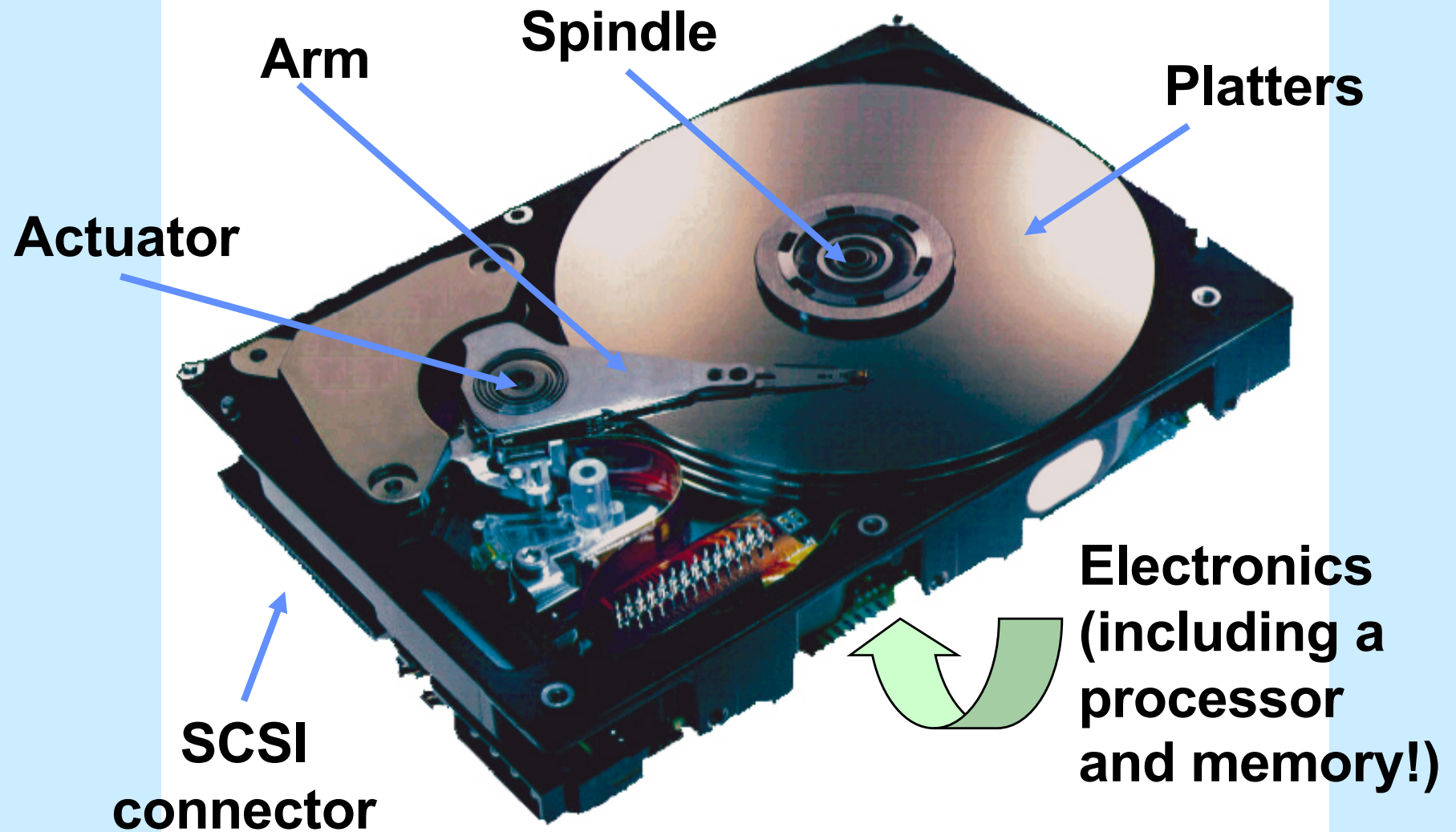
```
char buf[BSIZE];
int pos;
void myput(char *str) {
    int len = strlen(str);
    for (int i=0; i<len; i++, pos++) {
        buf[pos] = str[i];
        if ((buf[pos] == '\n') || (pos == BSIZE-1)) {
            write(1, buf, pos+1);
            pos = -1;
        }
    }
}
```

# Async-Signal Safety

- Which library functions are safe to use within signal handlers?

– abort	– dup2	– getppid	– readlink	– sigemptyset	– tcgetpgrp
– accept	– execl	– getsockname	– recv	– sigfillset	– tcsendbreak
– access	– execve	– getsockopt	– recvfrom	– sigismember	– tcsetattr
– aio_error	– _exit	– getuid	– recvmsg	– signal	– tcsetpgrp
– aio_return	– fchmod	– kill	– rename	– sigpause	– time
– aio_suspend	– fchown	– link	– rmdir	– sigpending	– timer_getoverrun
– alarm	– fcntl	– listen	– select	– sigprocmask	– timer_gettime
– bind	– fdatsync	– lseek	– sem_post	– sigqueue	– timer_settime
– cfgetispeed	– fork	– lstat	– send	– sigsuspend	– times
– cfgetospeed	– fpathconf	– mkdir	– sendmsg	– sleep	– umask
– cfsetispeed	– fstat	– mkfifo	– sendto	– sockatmark	– uname
– cfsetospeed	– fsync	– open	– setgid	– socket	– unlink
– chdir	– ftruncate	– pathconf	– setpgid	– socketpair	– utime
– chmod	– getegid	– pause	– setsid	– stat	– wait
– chown	– geteuid	– pipe	– setsockopt	– symlink	– waitpid
– clock_gettime	– getgid	– poll	– setuid	– sysconf	– write
– close	– getgroups	– posix_trace_event	– shutdown	– tcdrain	
– connect	– getpeername	– pselect	– sigaction	– tcflow	
– creat	– getpgrp	– raise	– sigaddset	– tcflush	
– dup	– getpid	– read	– sigdelset	– tcgetattr	

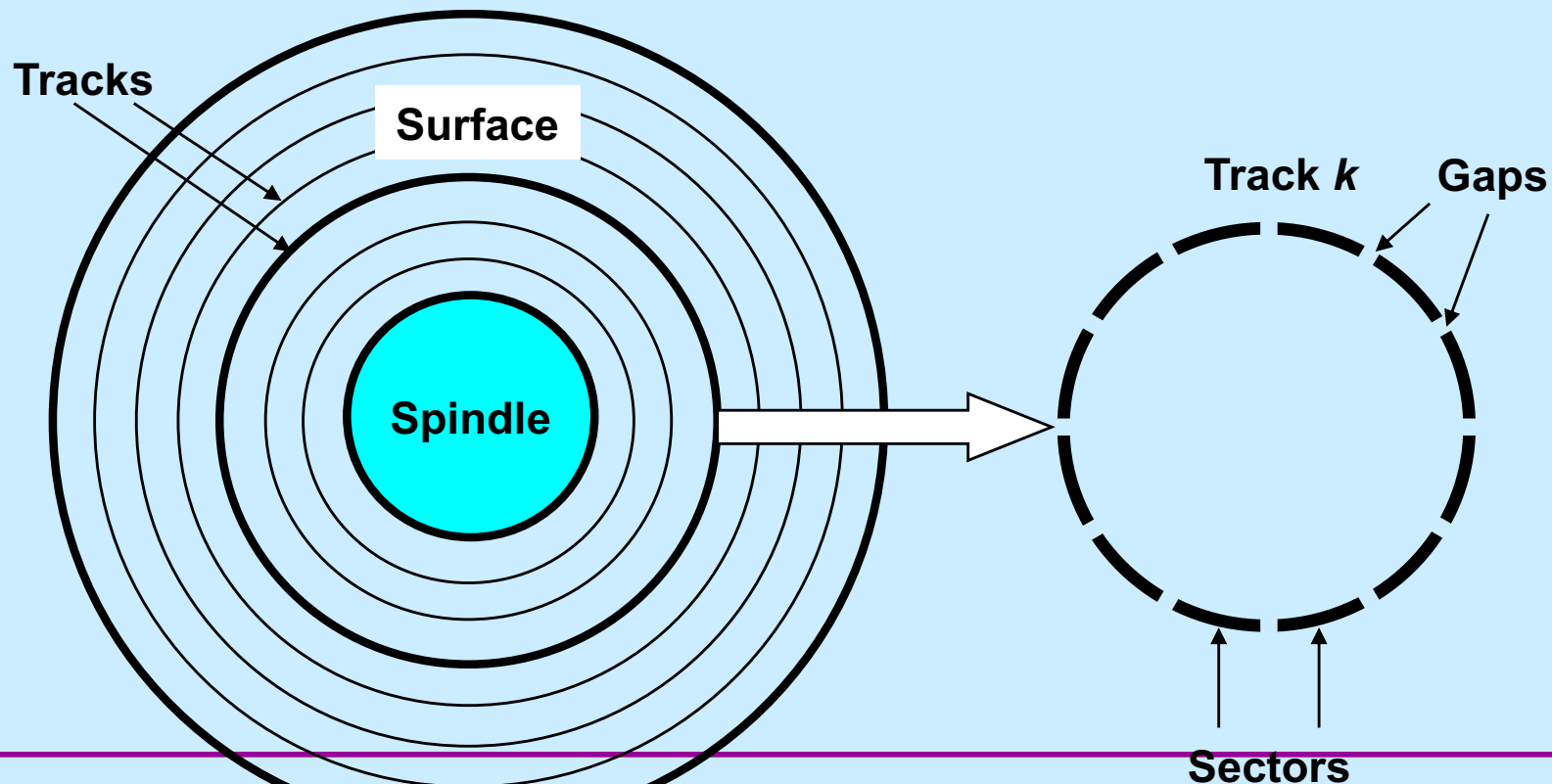
# What's Inside A Disk Drive?



*Image courtesy of Seagate Technology*

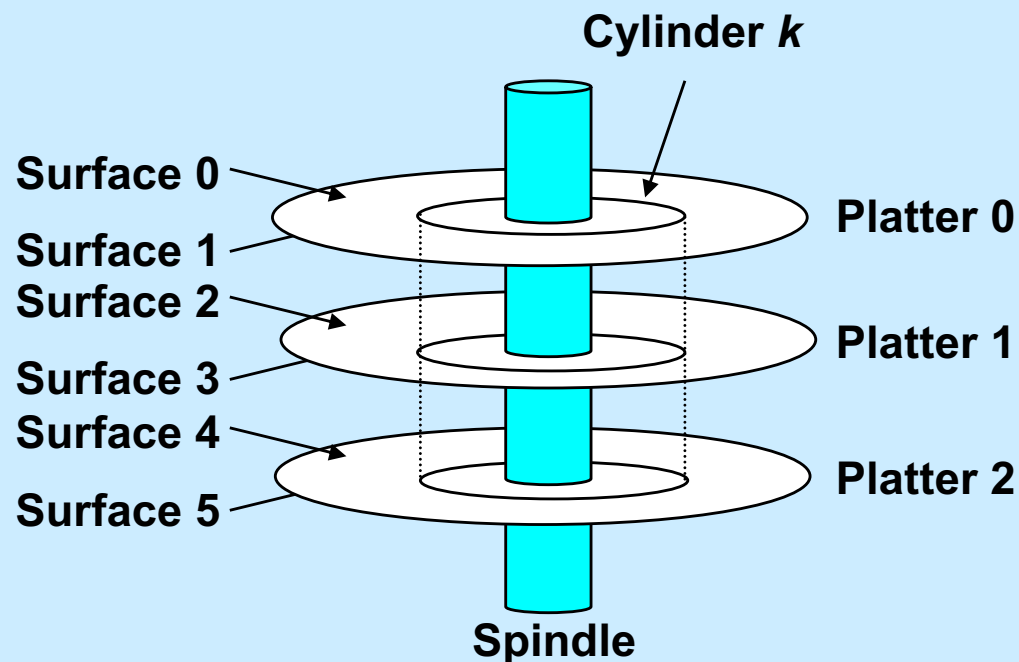
# Disk Geometry

- 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
  - capacity expressed in units of gigabytes (GB), where  $1 \text{ GB} = 2^{30} \text{ Bytes} \approx 10^9 \text{ Bytes}$
- 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<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

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

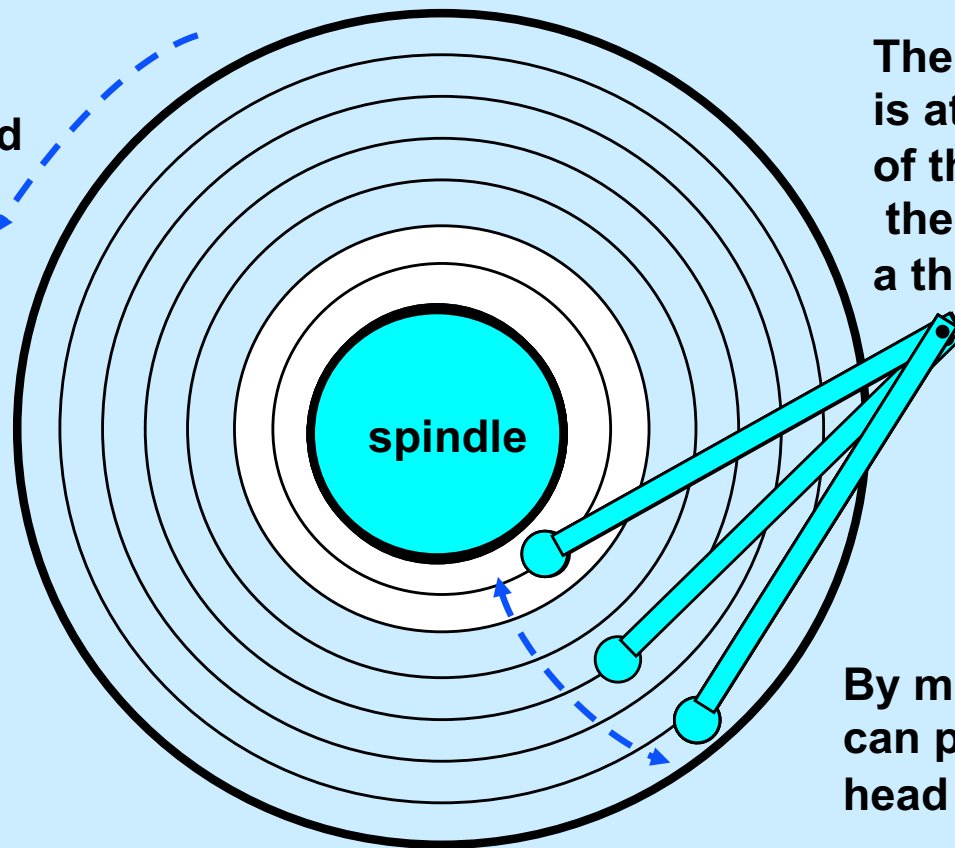
**Example:**

- 512 bytes/sector
- 600 sectors/track (on average)
- 40,000 tracks/surface
- 2 surfaces/platter
- 5 platters/disk

$$\begin{aligned}\text{Capacity} &= 512 \times 600 \times 40000 \times 2 \times 5 \\ &= 122,880,000,000 \\ &= 113.88 \text{ GB}\end{aligned}$$

# Disk Operation (Single-Platter View)

The disk surface spins at a fixed rotational rate

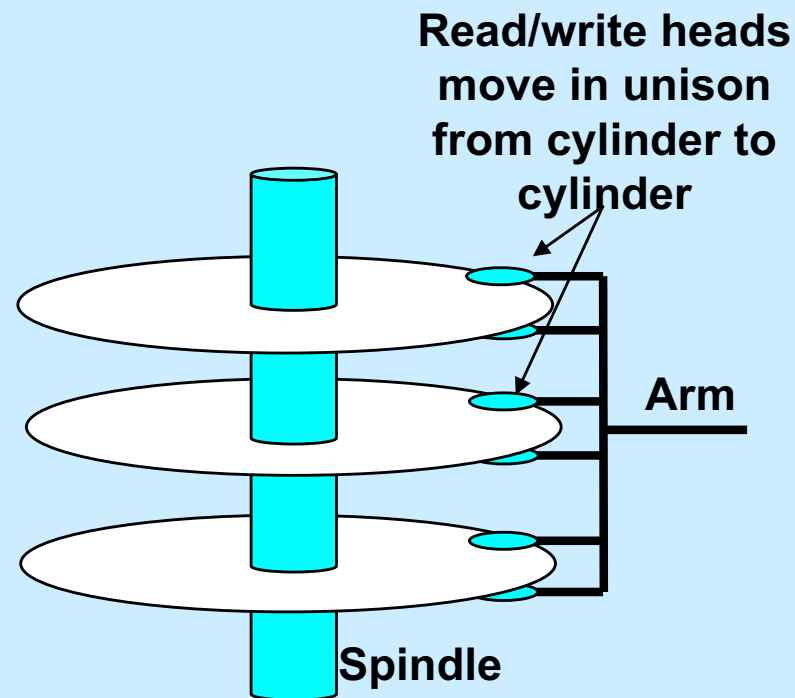


The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air

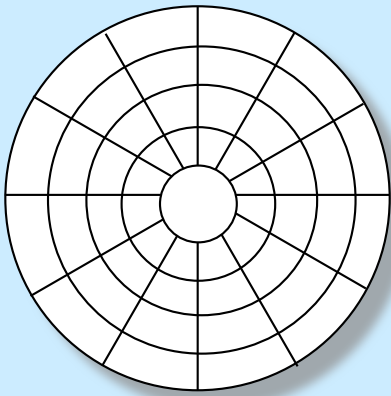
By moving radially, the arm can position the read/write head over any track



# Disk Operation (Multi-Platter View)



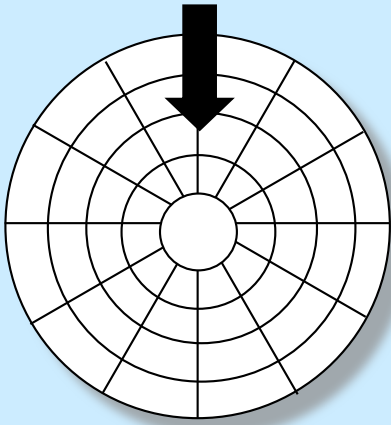
# Disk Structure: Top View of Single Platter



**Surface organized into tracks**

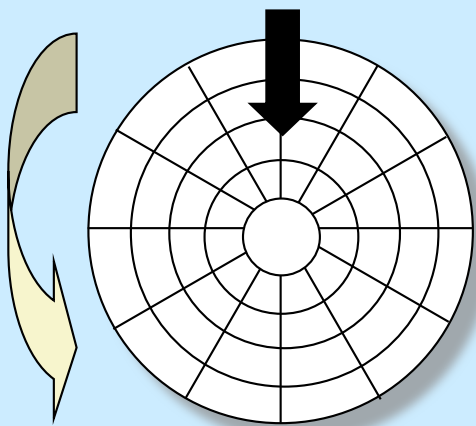
**Tracks divided into sectors**

# Disk Access



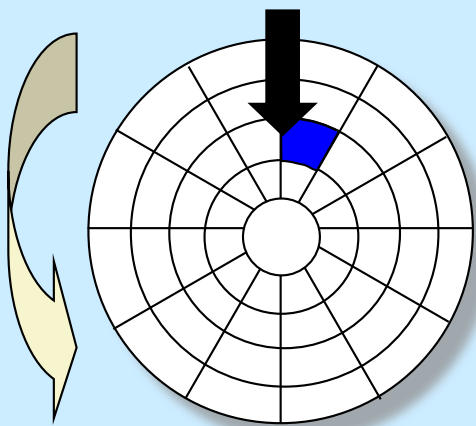
**Head in position above a track**

# Disk Access



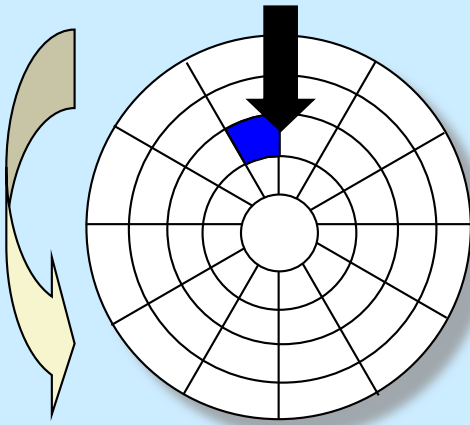
**Rotation is counter-clockwise**

# Disk Access – Read



**About to read blue sector**

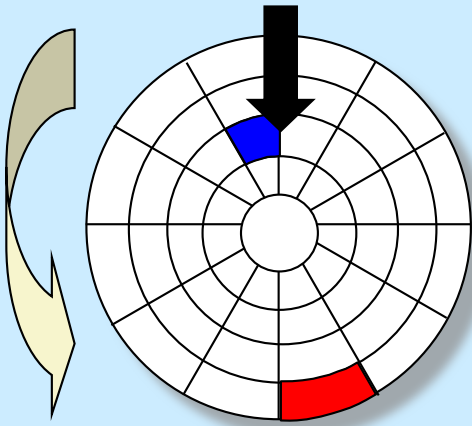
# Disk Access – Read



After **BLUE**  
read

After reading blue sector

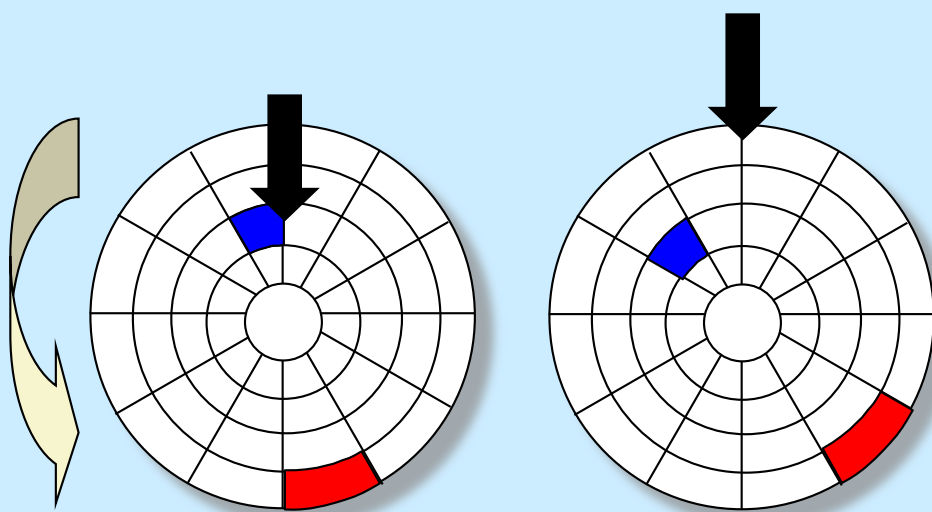
# Disk Access – Read



After **BLUE**  
read

**Red request scheduled next**

# Disk Access – Seek



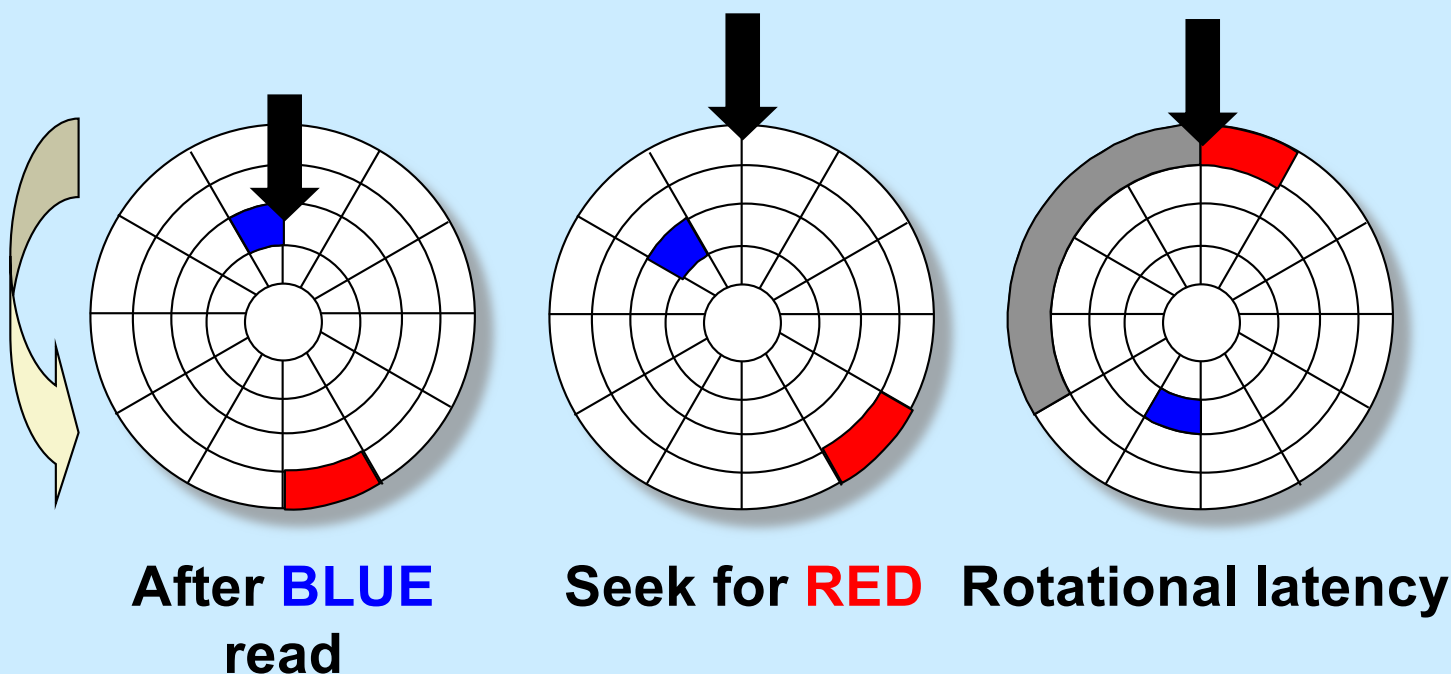
After **BLUE**  
read

Seek for **RED**

**Seek to red's track**

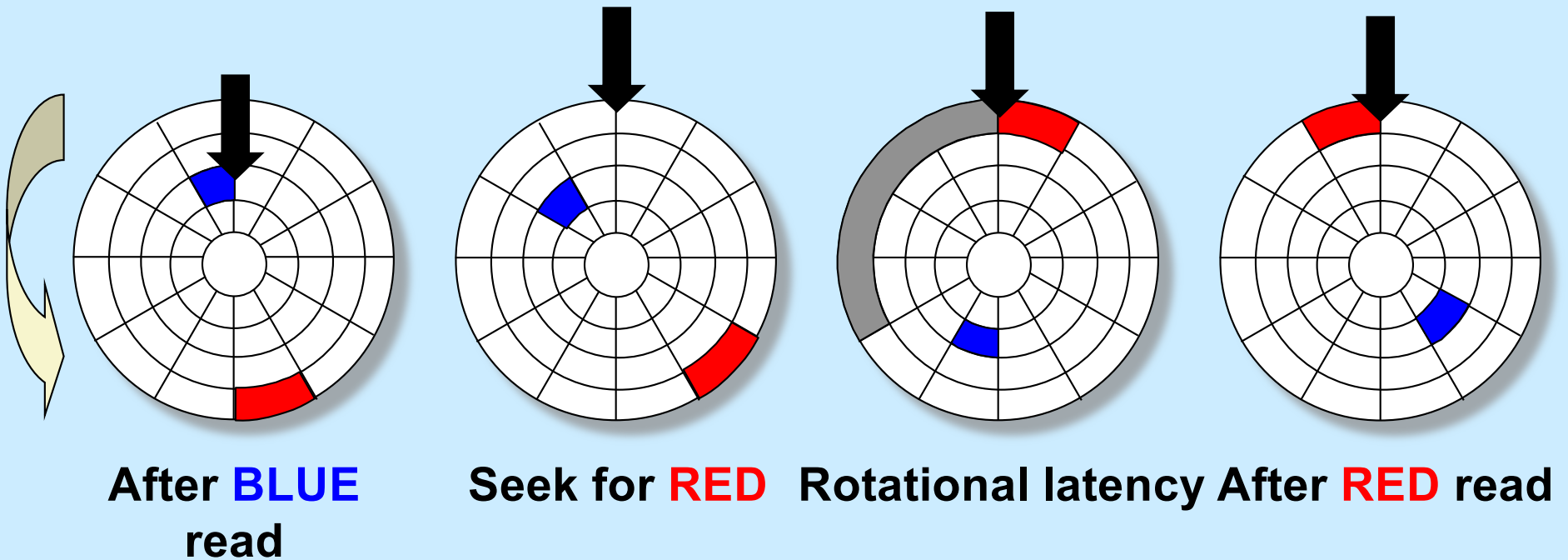


# Disk Access – Rotational Latency



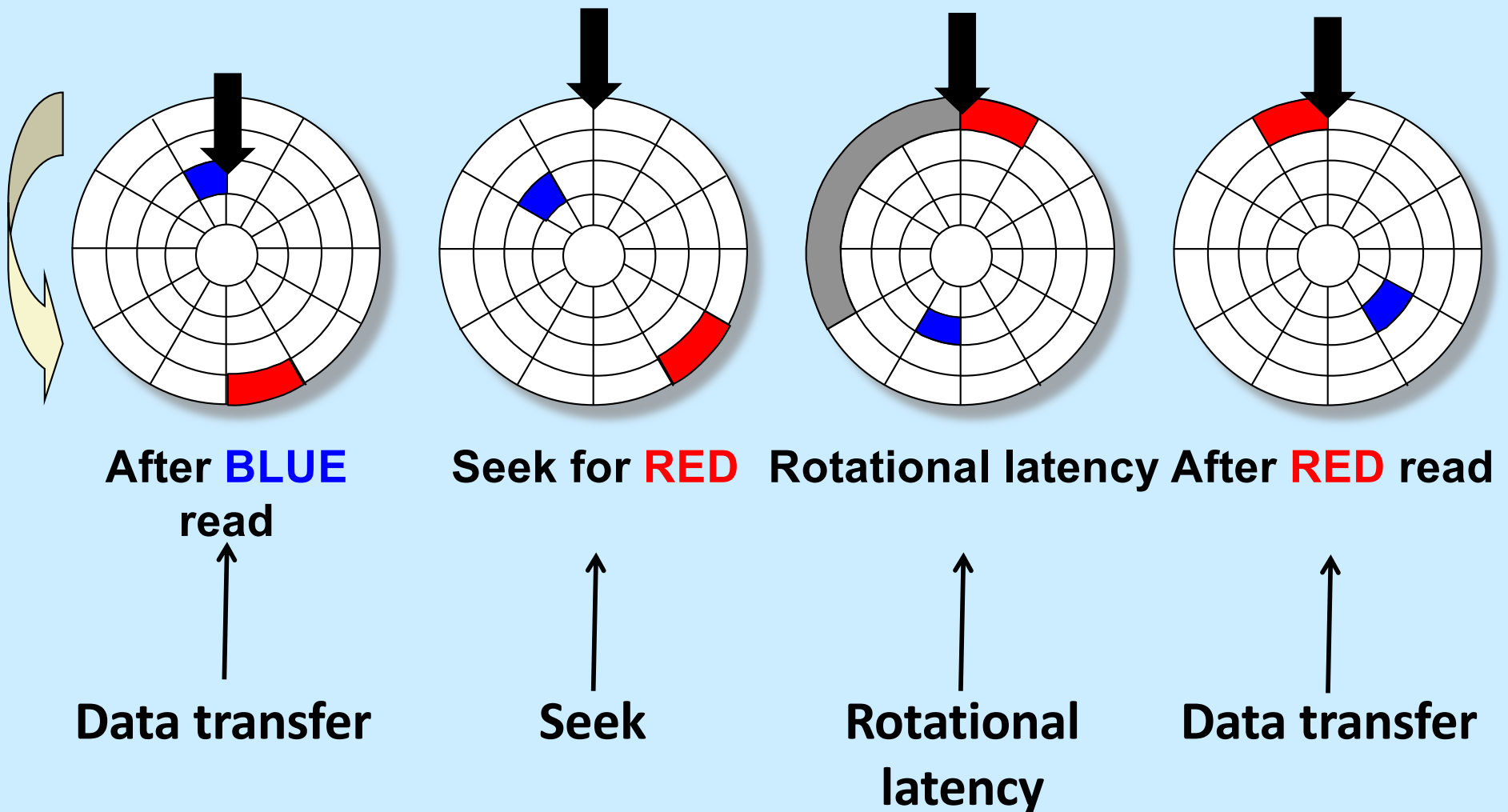
**Wait for red sector to rotate around**

# 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
  - typical rotation speed  $R = 7200$  RPM
  - $T_{\text{avg rotation}} = 1/2 \times 1/R \times 60 \text{ sec/1 min}$
- **Transfer time** ( $T_{\text{avg transfer}}$ )
  - time to read the bits in the target sector
  - $T_{\text{avg transfer}} = 1/R \times 1/(\text{avg \# sectors/track}) \times 60 \text{ secs/1 min}$

# Disk Access Time Example

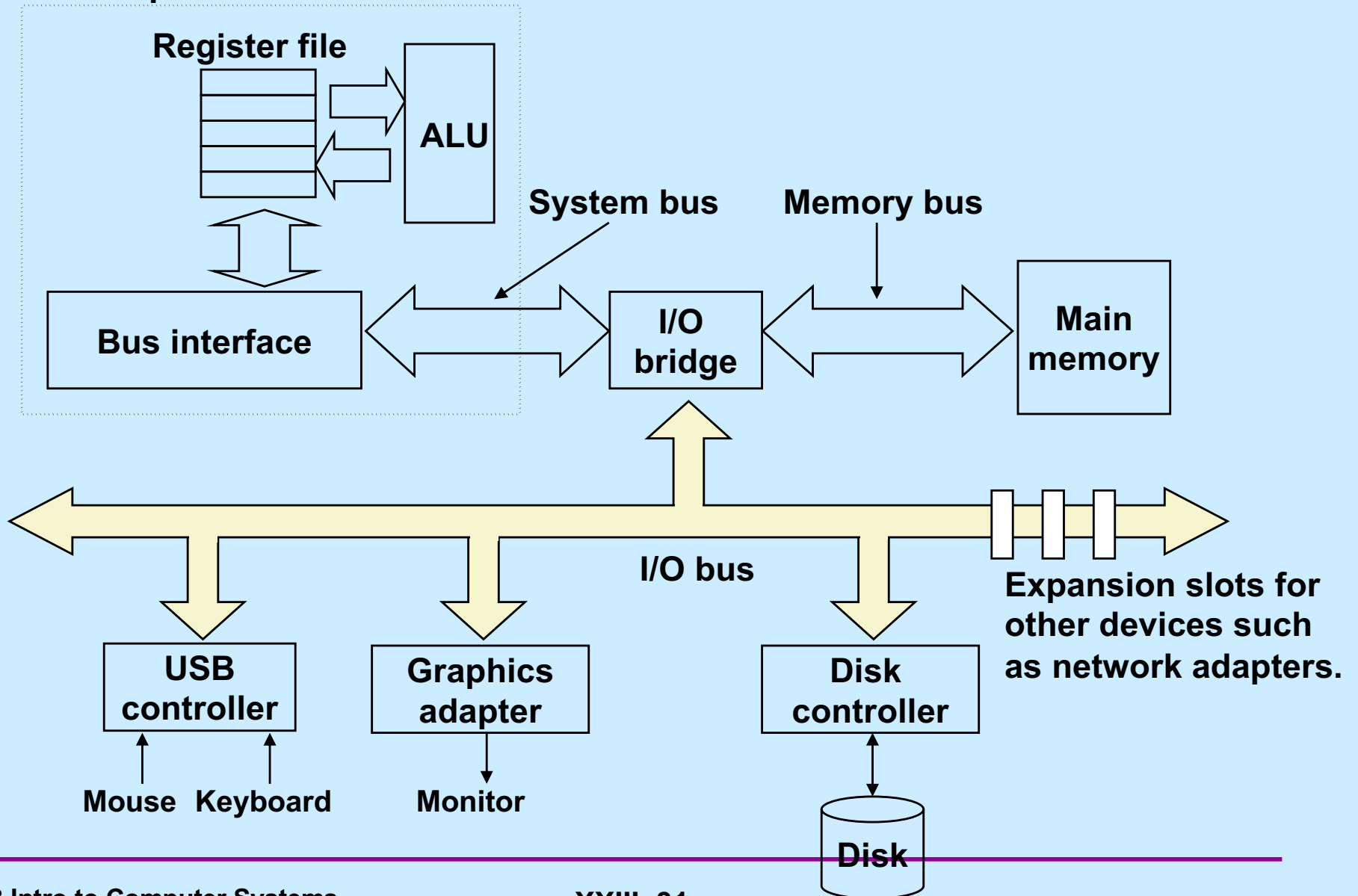
- **Given:**
  - rotational rate = 7,200 RPM
  - average seek time = 9 ms
  - avg # sectors/track = 600
- **Derived:**
  - $T_{\text{avg rotation}} = 1/2 \times (60 \text{ secs}/7200 \text{ RPM}) \times 1000 \text{ ms/sec} = 4 \text{ ms}$
  - $T_{\text{avg transfer}} = 60/7200 \text{ RPM} \times 1/600 \text{ sects/track} \times 1000 \text{ ms/sec} = 0.014 \text{ ms}$
  - $T_{\text{access}} = 9 \text{ ms} + 4 \text{ ms} + 0.014 \text{ ms}$
- **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”
-

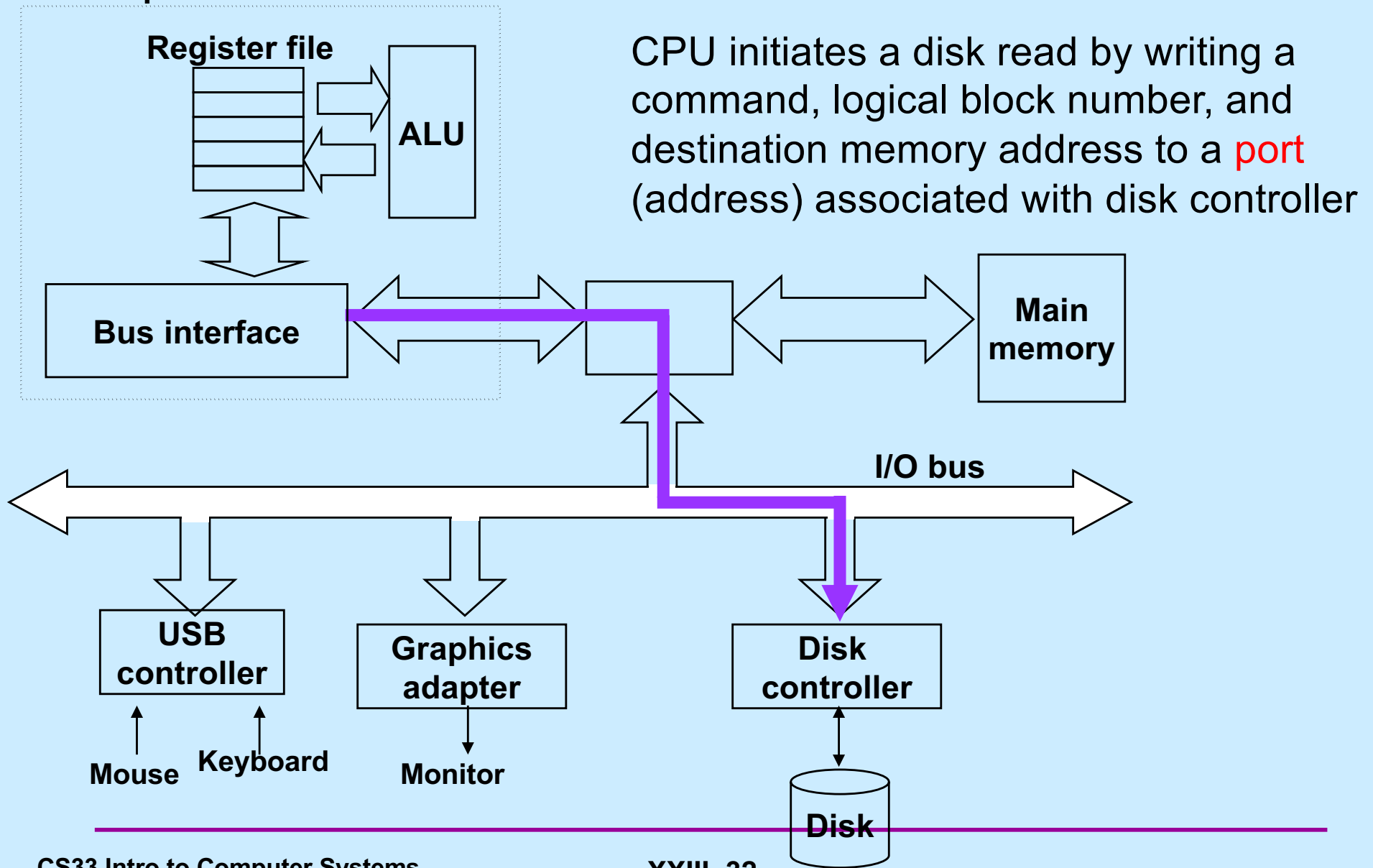
# I/O Bus

CPU chip



# Reading a Disk Sector (1)

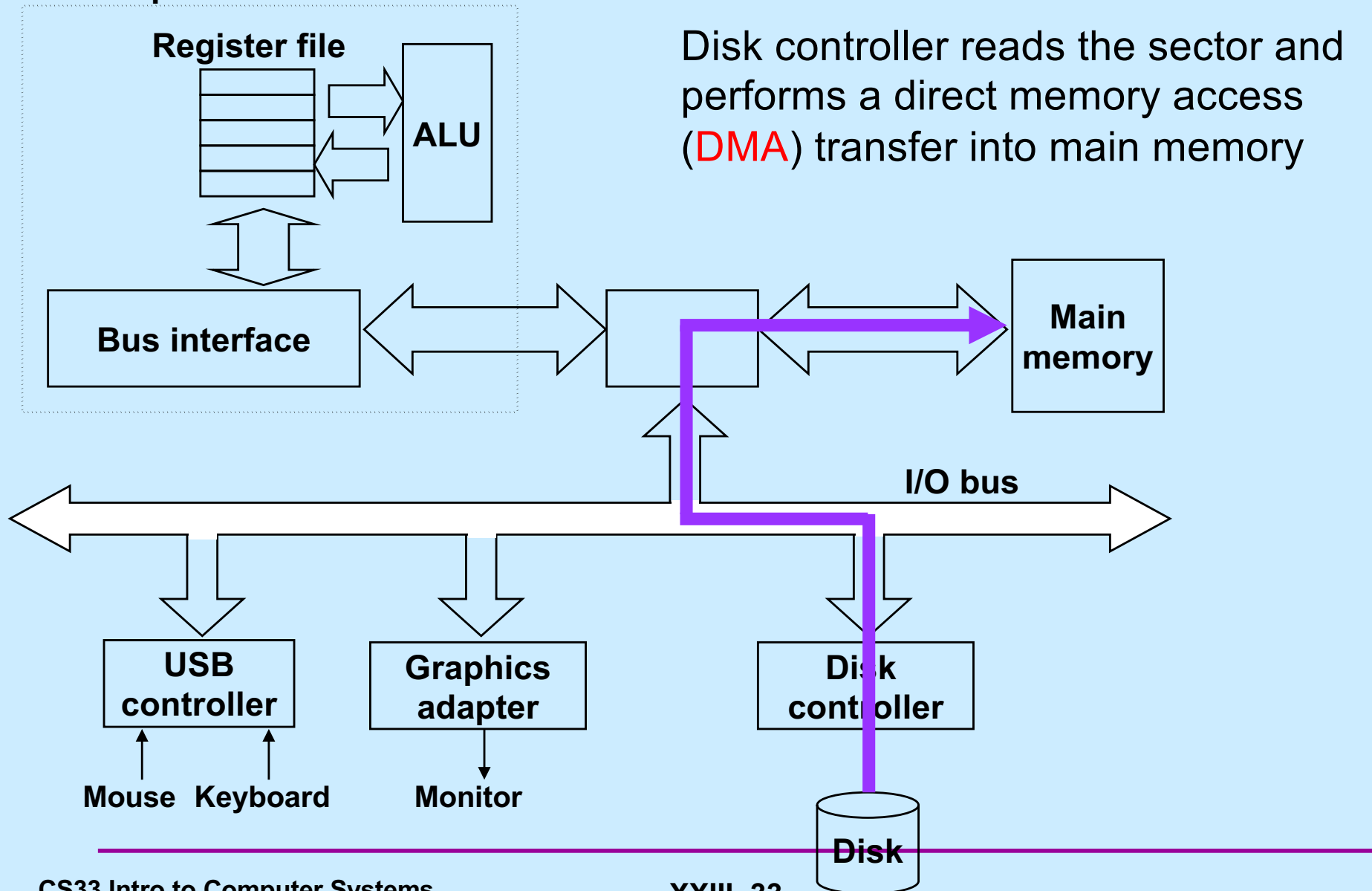
CPU chip





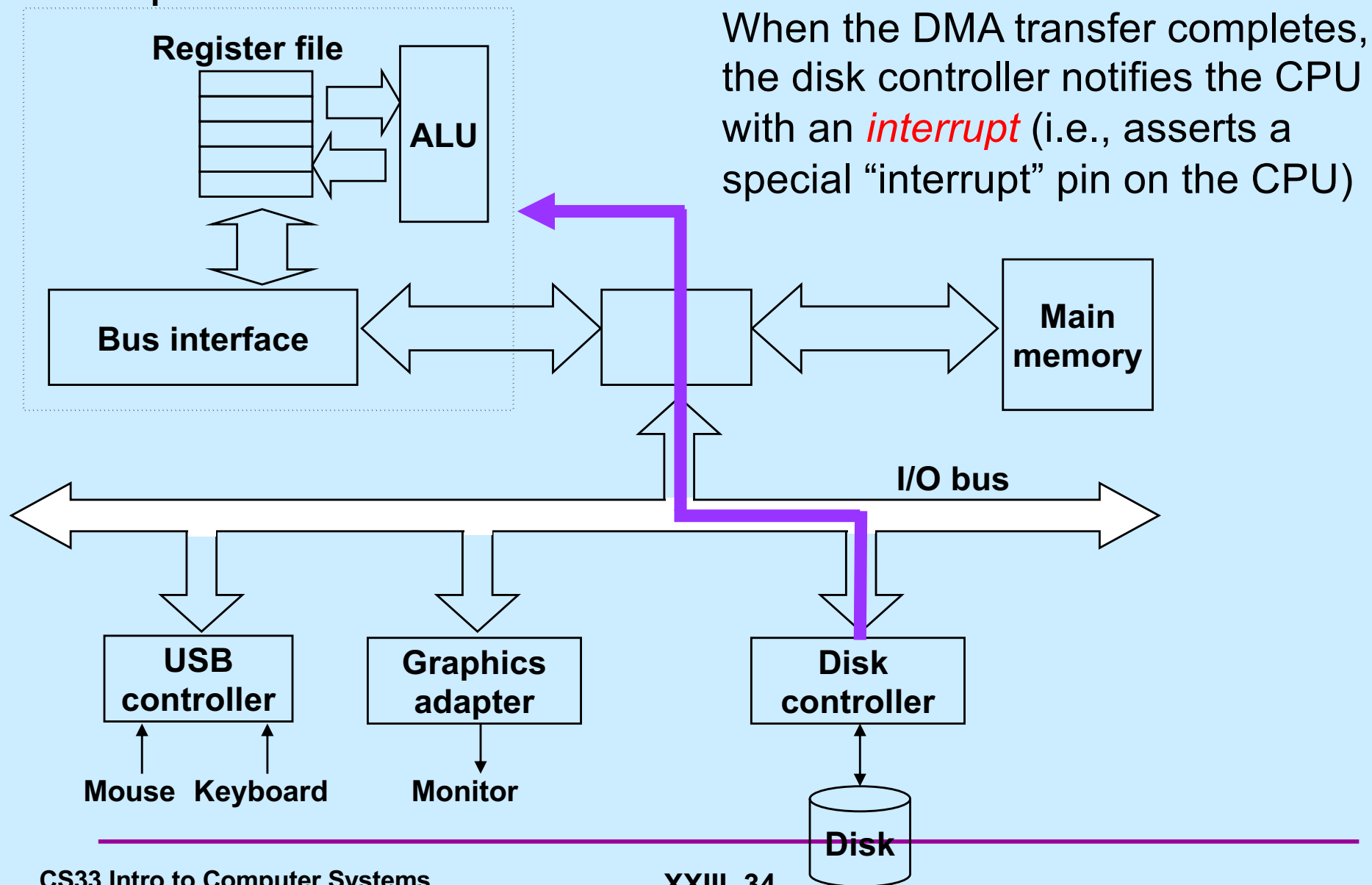
# Reading a Disk Sector (2)

CPU chip

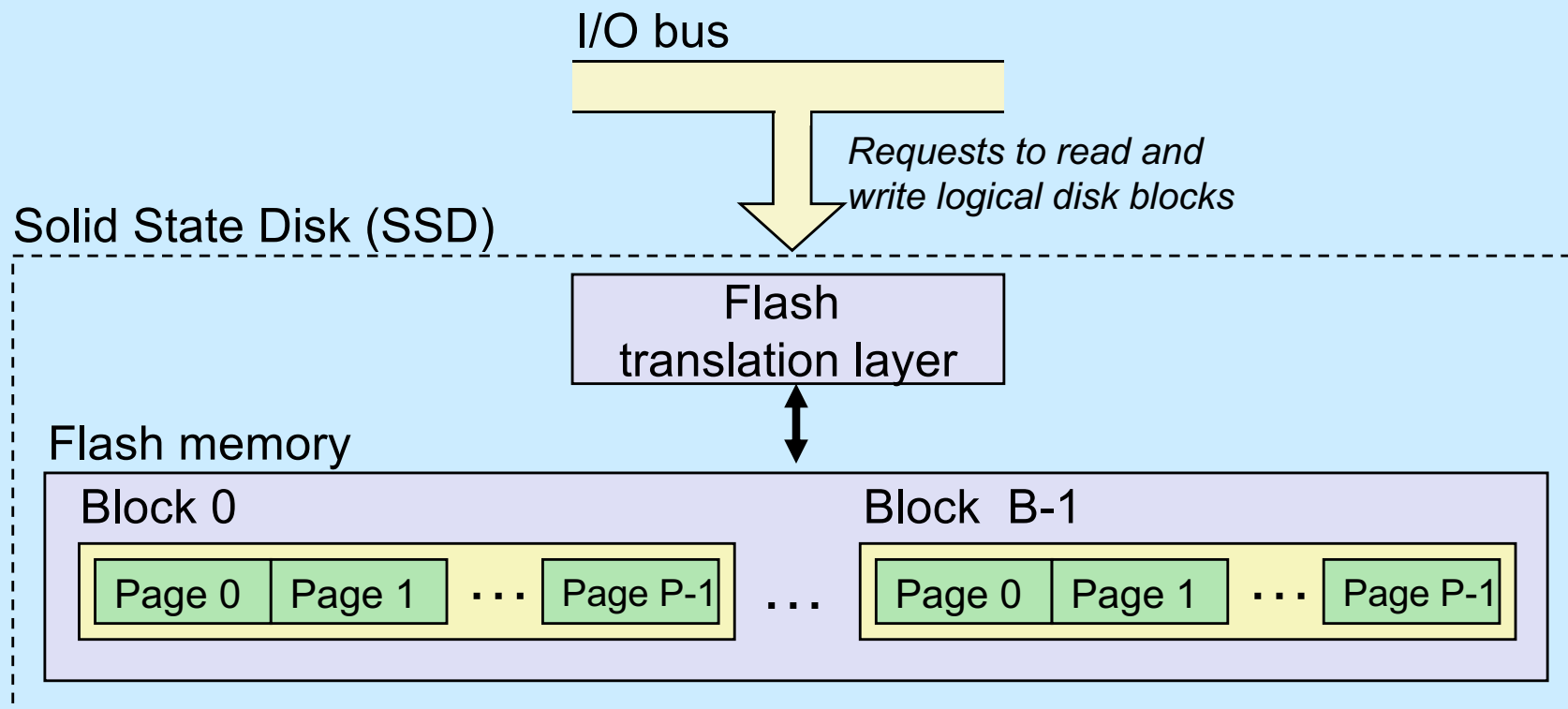


# Reading a Disk Sector (3)

CPU chip



# 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 100,000 repeated writes**

# SSD Performance Characteristics

Sequential read tput	250 MB/s	Sequential write tput	170 MB/s
Random read tput	140 MB/s	Random write tput	14 MB/s
Random read access	30 us	Random write access	300 us

- **Why are random writes so slow?**
  - erasing a block is slow (around 1 ms)
  - modifying a page triggers a copy of all useful pages in the block
    - » find a used block (new block) and erase it
    - » write the page into the new block
    - » copy other pages from old block to the new block

# SSD Tradeoffs vs Rotating Disks

- **Advantages**

- no moving parts → faster, less power, more rugged

- **Disadvantages**

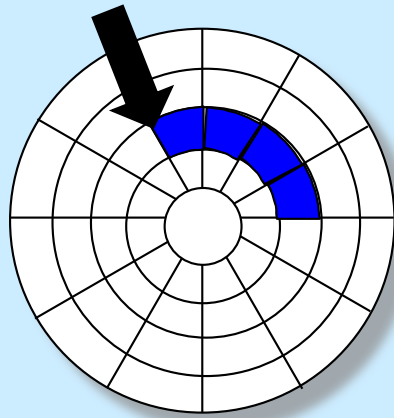
- have the potential to wear out
  - » mitigated by “wear-leveling logic” in flash translation layer
  - » e.g. Intel X25 guarantees 1 petabyte ( $10^{15}$  bytes) of random writes before they wear out
- in 2010, about 100 times more expensive per byte
- in 2017, about 6 times more expensive per byte

- **Applications**

- smart phones, laptops
- Apple “Fusion” drives

# Reading a File on a Rotating Disk

- **Suppose the data of a file are stored on consecutive disk sectors on one track**
  - **this is the best possible scenario for reading data quickly**
    - » **single seek required**
    - » **single rotational delay**
    - » **all sectors read in a single scan**

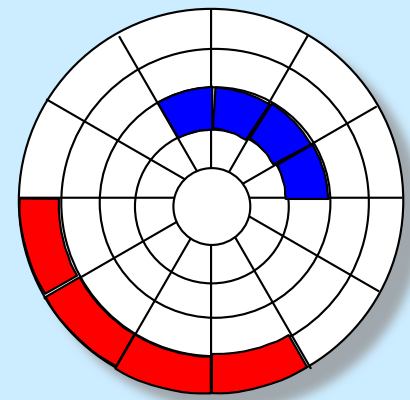


# Quiz 2

We have two files on the same (rotating) disk. The first file's data resides in consecutive sectors on one track, the second in consecutive sectors on another track. It takes a total of  $t$  seconds to read all of the first file then all of the second file.

Now suppose the files are read concurrently, perhaps a sector of the first, then a sector of the second, then the first, then the second, etc. Compared to reading them sequentially, this will take

- a) less time
- b) about the same amount of time
- c) more time



# Quiz 3

**We have two files on the same solid-state disk. Each file's data resides in consecutive blocks. It takes a total of  $t$  seconds to read all of the first file then all of the second file.**

**Now suppose the files are read concurrently, perhaps a block of the first, then a block of the second, then the first, then the second, etc. Compared to reading them sequentially, this will take**

- a) less time**
- b) about the same amount of time**
- c) more time**



# 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	.3	0.03	3,333,333
access (ms)	75	28	10	8	5	3	3	25
typical size (GB)	.01	.16	1	20	160	1,500	3,000	300,000

# CPU Clock Rates

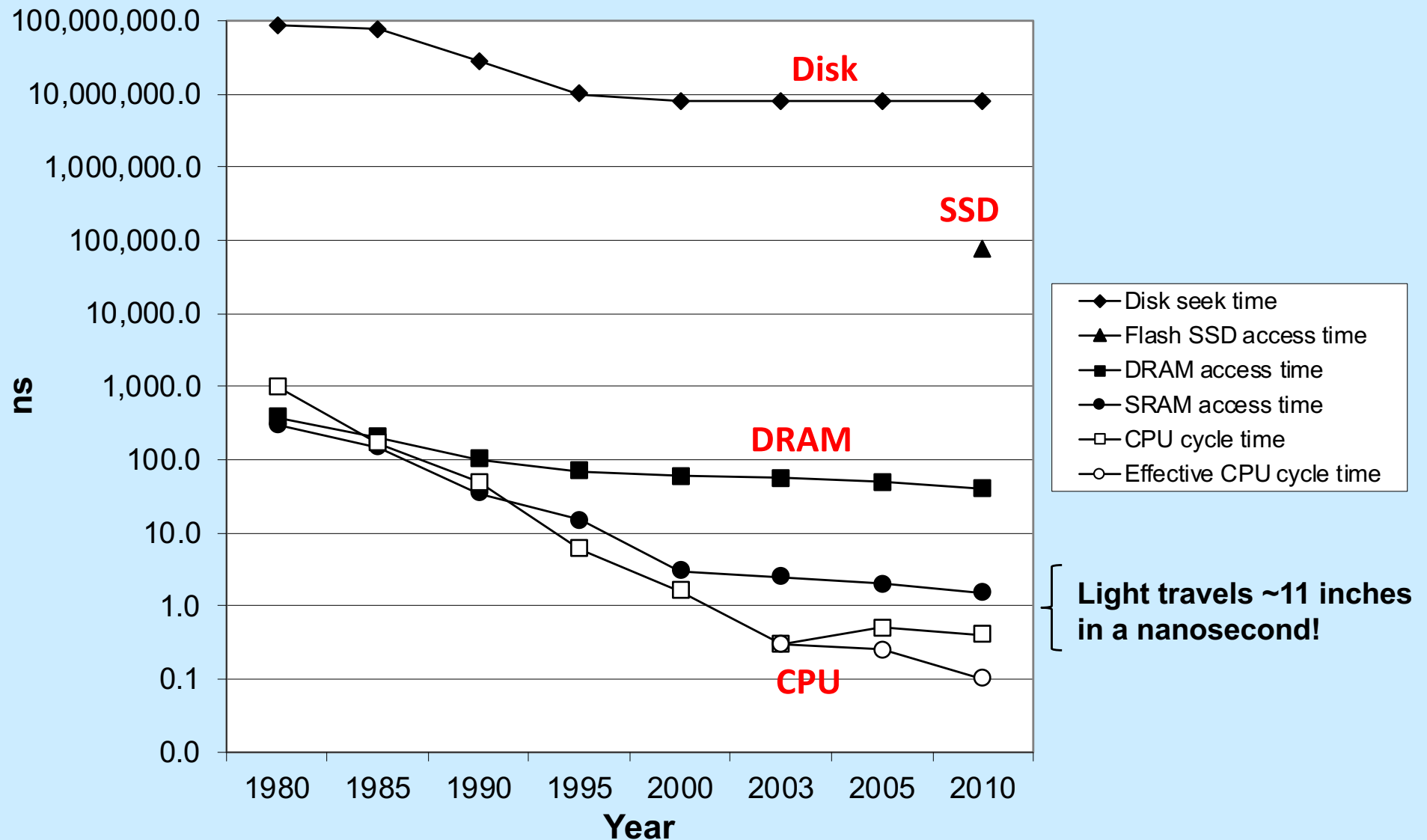
Inflection point in computer history  
when designers hit the “Power Wall”



	1985	1990	1995	2000	2003	2005	2015	2015:1985
CPU	286	386	Pentium	P-III	P-4	Core 2	Core i7	---
Clock rate (MHz)	6	20	150	600	3300	2000	3000	500
Cycle time (ns)	166	50	6	1.6	0.3	0.50	0.33	500
Cores	1	1	1	1	1	2	4	4
Effective cycle time (ns)	166	50	6	1.6	0.3	0.25	0.08	2075

# The CPU-Memory Gap

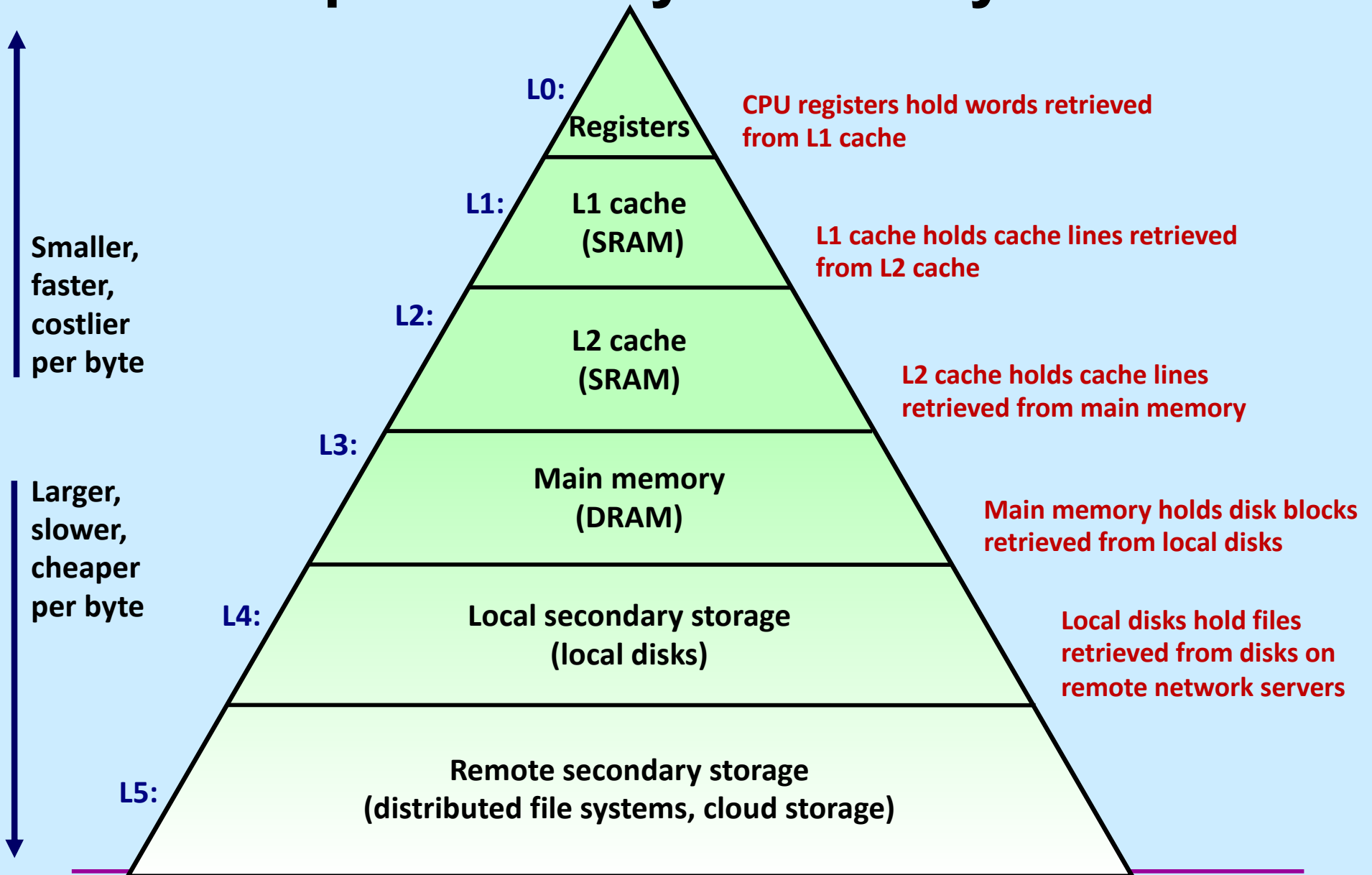
The gap widens between DRAM, disk, and CPU speeds



# 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****

# An Example Memory Hierarchy



# Putting Things Into Perspective ...

- **Reading from:**
  - ... the L1 cache is like grabbing a piece of paper from your desk (3 seconds)
  - ... the L2 cache is picking up a book from a nearby shelf (14 seconds)
  - ... main system memory is taking a 4-minute walk down the hall to talk to a friend
  - ... a hard drive is like leaving the building to roam the earth for one year and three months

# Disks Are Important

- **Cheap**
  - cost/byte much less than SSDs
- **(fairly) Reliable**
  - data written to a disk is likely to be there next year
- **Sometimes fast**
  - data in consecutive sectors on a track can be read quickly
- **Sometimes slow**
  - data in randomly scattered sectors takes a long time to read

# Abstraction to the Rescue

- Programs don't deal with sectors, tracks, and cylinders
- Programs deal with *files*
  - maze.c rather than an ordered collection of sectors
  - OS provides the implementation



# Implementation Problems

- **Speed**
  - **use the hierarchy**
    - » **copy files into RAM, copy back when done**
  - **optimize layout**
    - » **put sectors of a file in consecutive locations**
  - **use parallelism**
    - » **spread file over multiple disks**
    - » **read multiple sectors at once**

# Implementation Problems

- **Reliability**
  - **computer crashes**
    - » what you thought was safely written to the file never made it to the disk — it's still in RAM, which is lost
    - » worse yet, some parts made it back to disk, some didn't
      - you don't know which is which
      - on-disk data structures might be totally trashed
  - **disk crashes**
    - » you had backed it up ... yesterday
  - **you screw up**
    - » you accidentally delete the entire directory containing your strings/performance solution

# Implementation Problems

- **Reliability solutions**
  - **computer crashes**
    - » **transaction-oriented file systems**
    - » **on-disk data structures always in well defined states**
  - **disk crashes**
    - » **files stored redundantly on multiple disks**
  - **you screw up**
    - » **file system automatically keeps "snapshots" of previous versions of files**