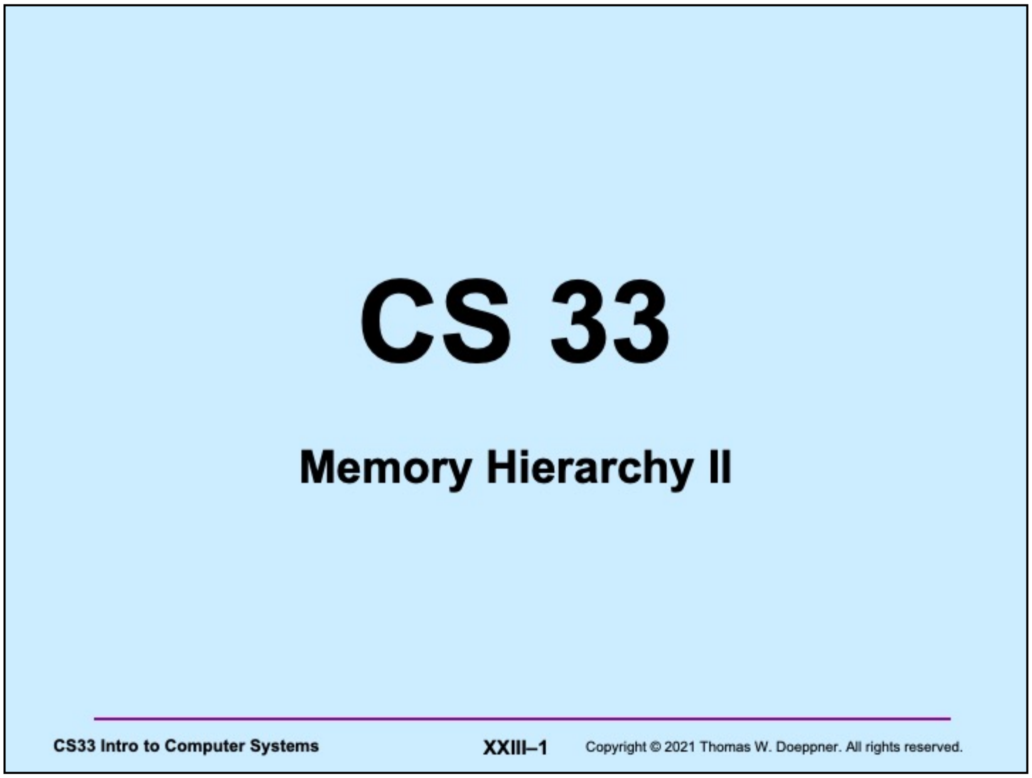
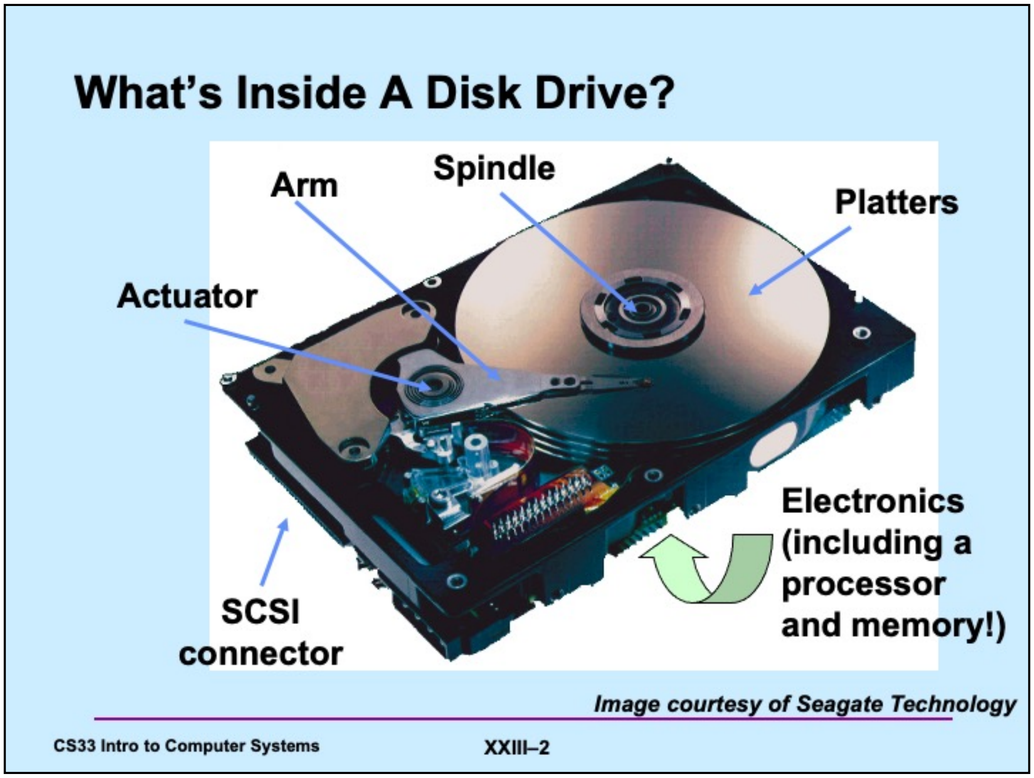


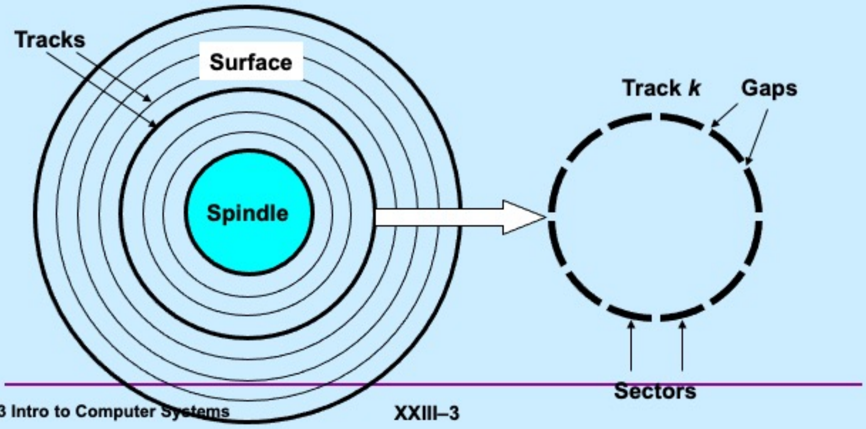
Most of the slides in this lecture are either from or adapted from slides provided by the authors of the textbook "Computer Systems: A Programmer's Perspective," 2nd Edition and are provided from the website of Carnegie-Mellon University, course 15-213, taught by Randy Bryant and David O'Hallaron in Fall 2010. These slides are indicated "Supplied by CMU" in the notes section of the slides.



Supplied by CMU.

Disk Geometry

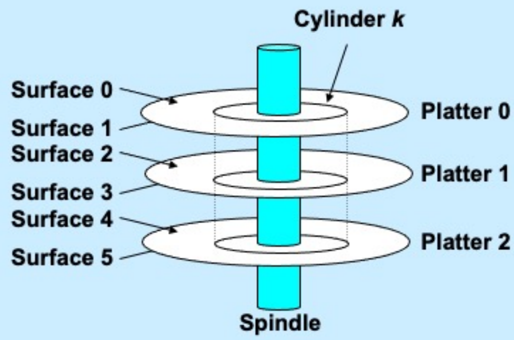
- Disks consist of **platters**, each with two **surfaces**
- Each surface consists of concentric rings called **tracks**
- Each track consists of **sectors** separated by **gaps**



Supplied by CMU.

Disk Geometry (Multiple-Platter View)

- **Aligned tracks form a cylinder**



Supplied by CMU.

Disk Capacity

- **Capacity**: maximum number of bits that can be stored
  - capacity expressed in units of gigabytes (GB), where 1 GB = 2^{30} Bytes ≈ 10^9 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
- 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

Supplied by CMU.

Computing Disk Capacity

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

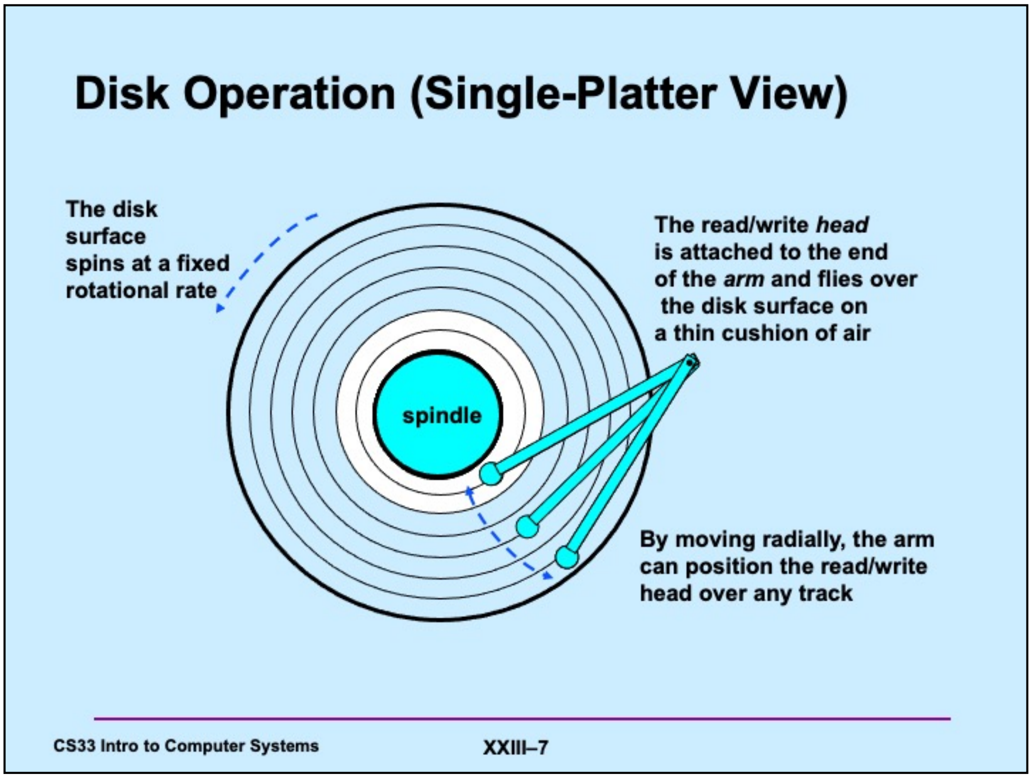
Example:

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

Capacity = 512 x 600 x 40000 x 2 x 5
= 122,880,000,000
= 113.88 GB

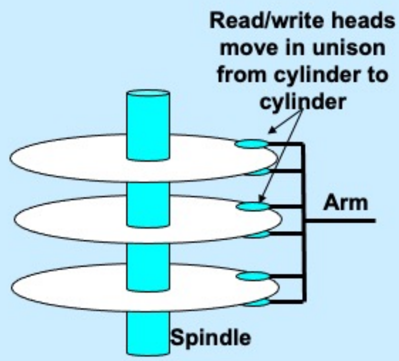
Supplied by CMU.

Note that 1GB = 2^{30} bytes.



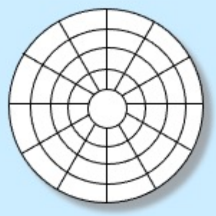
Supplied by CMU.

Disk Operation (Multi-Platter View)



Supplied by CMU.

Disk Structure: Top View of Single Platter

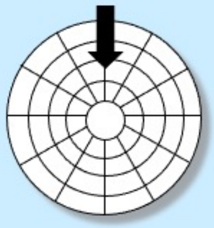


Surface organized into tracks

Tracks divided into sectors

Supplied by CMU.

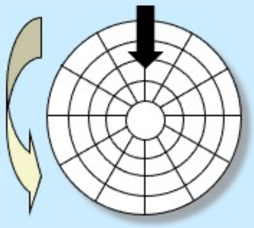
Disk Access



Head in position above a track

Supplied by CMU.

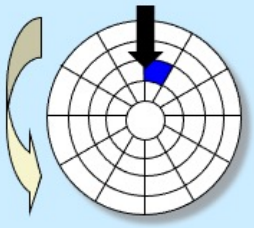
Disk Access



Rotation is counter-clockwise

Supplied by CMU.

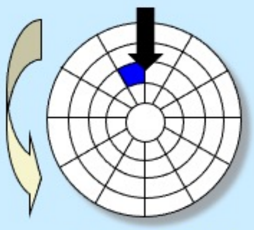
Disk Access – Read



About to read blue sector

Supplied by CMU.

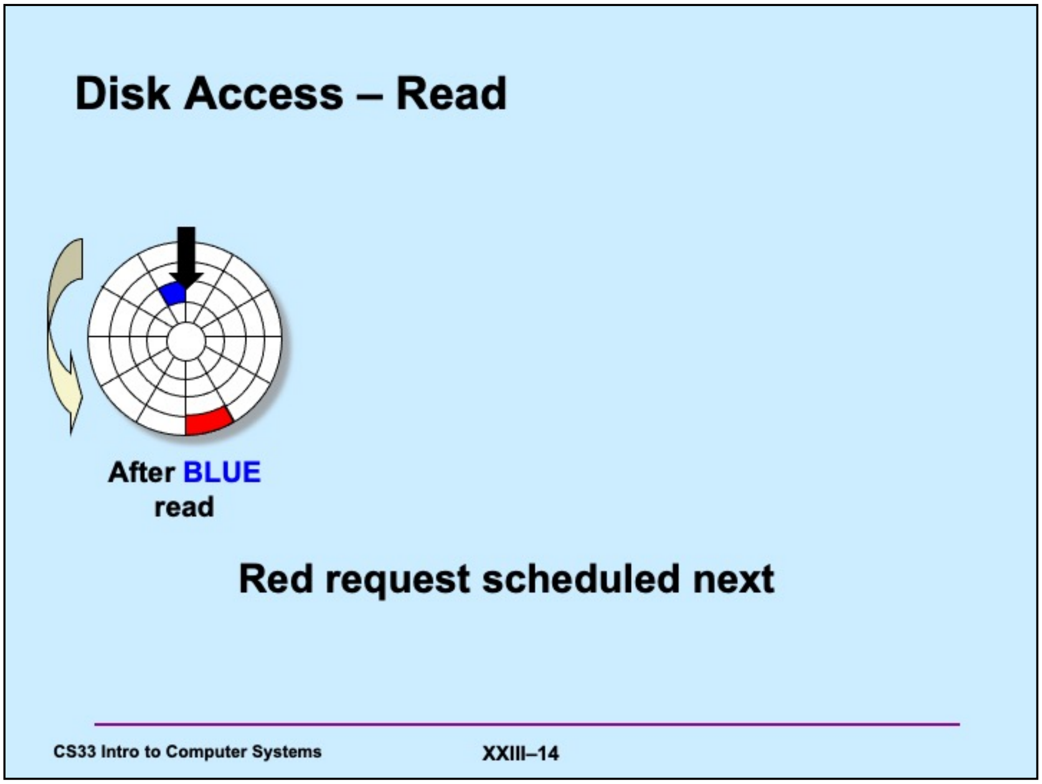
Disk Access – Read



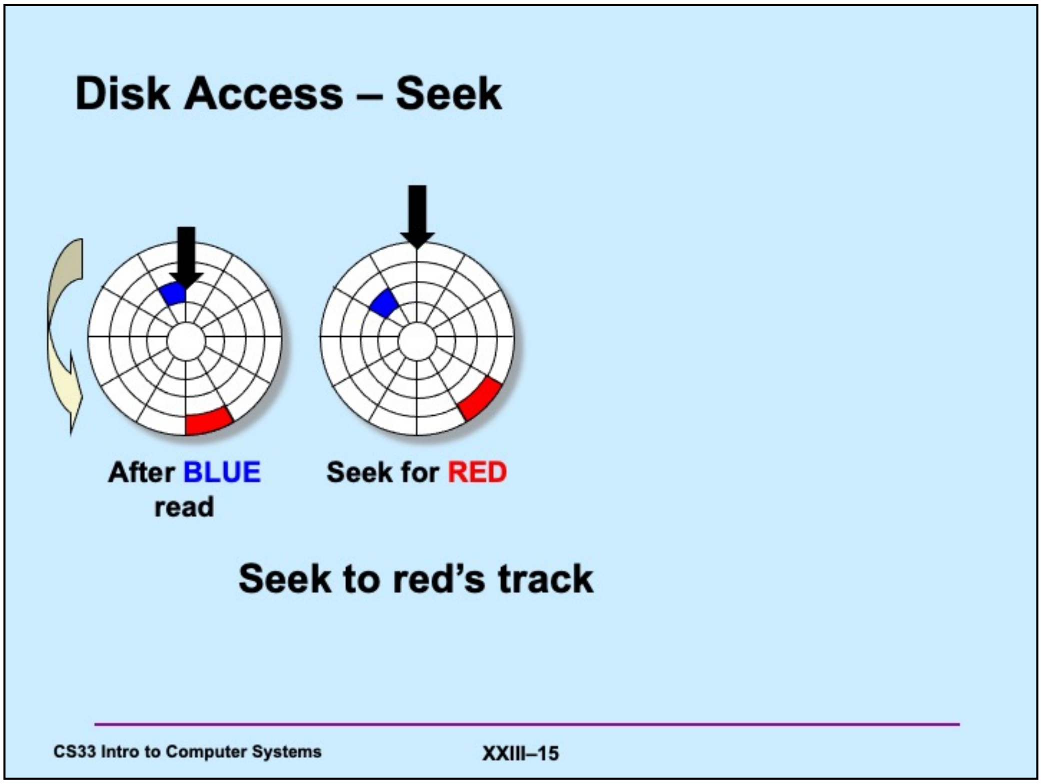
After BLUE read

After reading blue sector

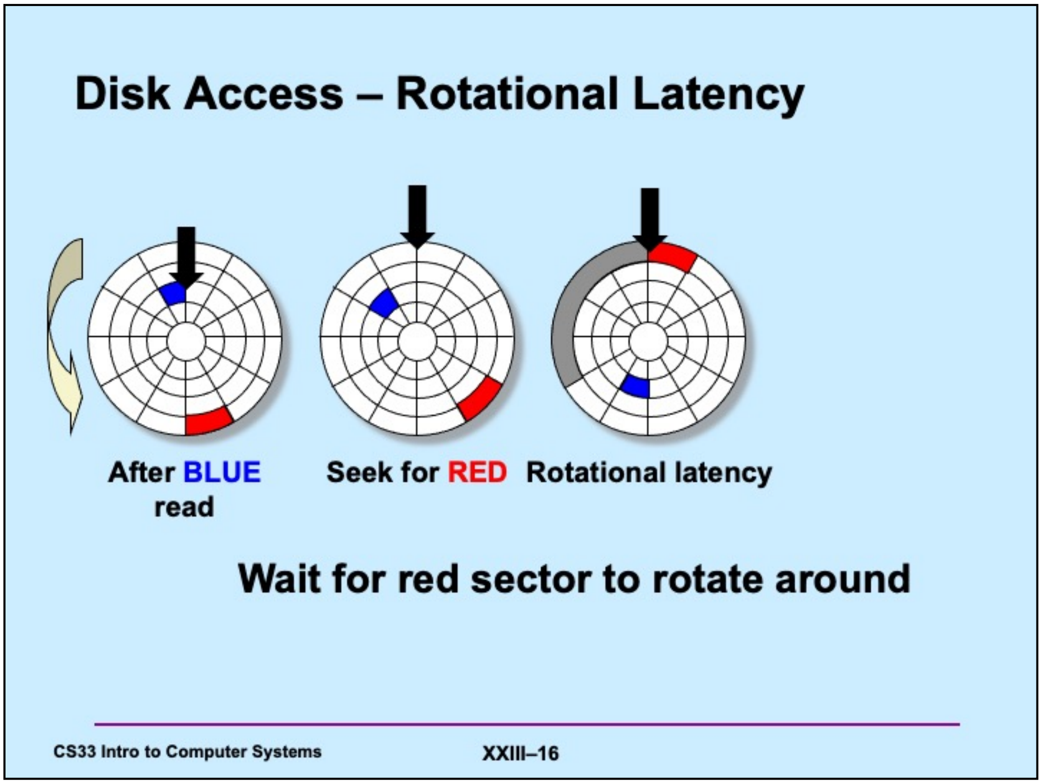
Supplied by CMU.



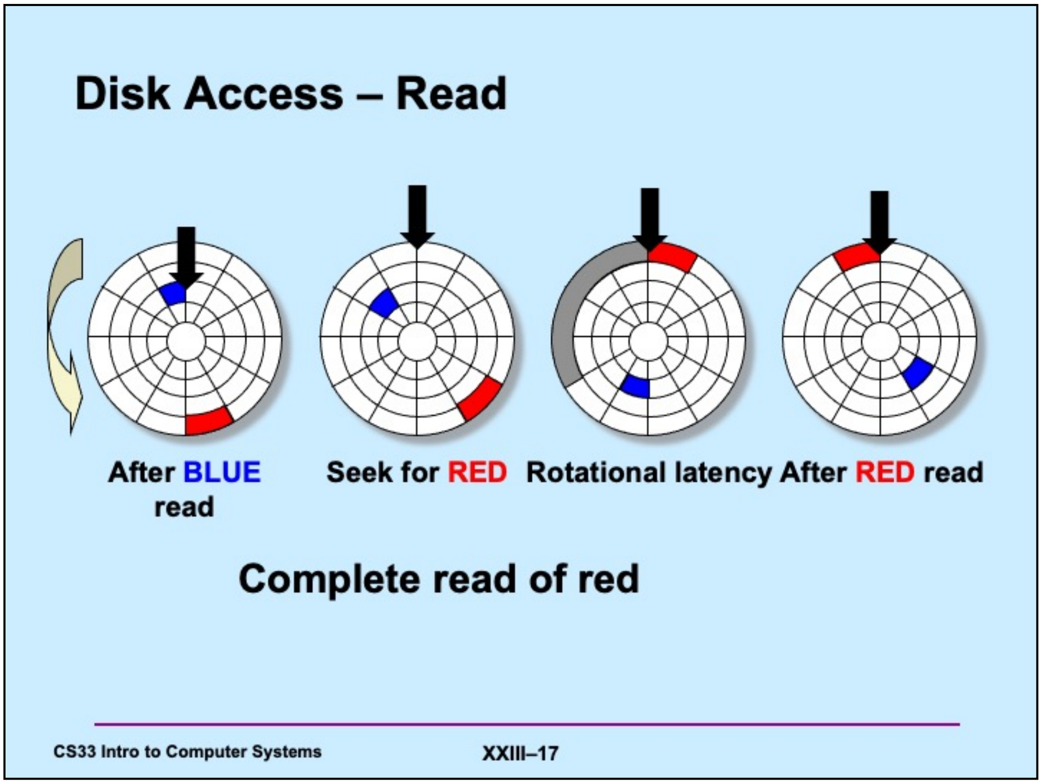
Supplied by CMU.



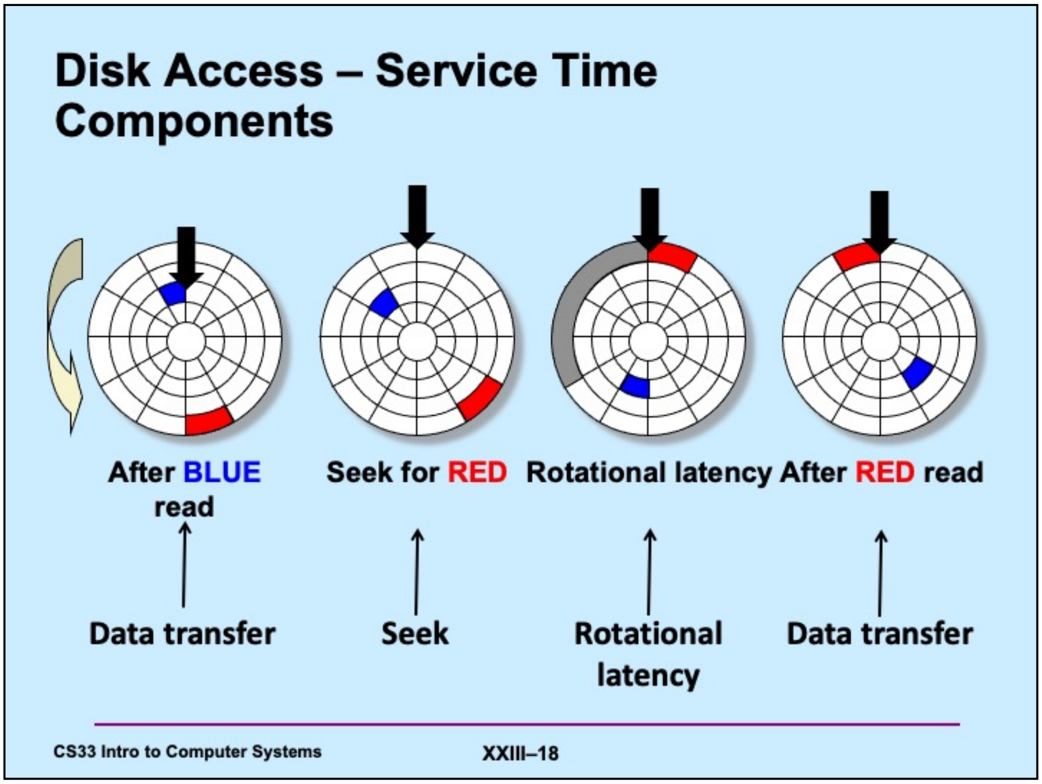
Supplied by CMU.



Supplied by CMU.



Supplied by CMU.



Supplied by CMU.

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
  - typical rotation speed R = 7200 RPM
  - Tavg rotation = 1/2 x 1/R x 60 sec/1 min
- **Transfer time** (Tavg transfer)
  - time to read the bits in the target sector
  - Tavg transfer = 1/R x 1/(avg # sectors/track) x 60 secs/1 min

Supplied by CMU.

Disk Access Time Example

- **Given:**
  - rotational rate = 7,200 RPM
  - average seek time = 9 ms
  - avg # sectors/track = 600
- **Derived:**
  - Tavg rotation = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms
  - Tavg transfer = 60/7200 RPM x 1/600 sects/track x 1000 ms/sec = 0.014 ms
  - Taccess = 9 ms + 4 ms + 0.014 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

Supplied by CMU.

Quiz 1

Assuming a 5-inch diameter disk spinning at 10,000 RPM, what is the approximate speed at which the outermost track is moving?

a) faster than a speeding bullet (i.e., supersonic)
b) roughly the speed of a pretty fast car (150 mph)
c) roughly the speed of a pretty slow car (50 mph)
d) roughly the speed of a world-class marathoner (13.1 mph)

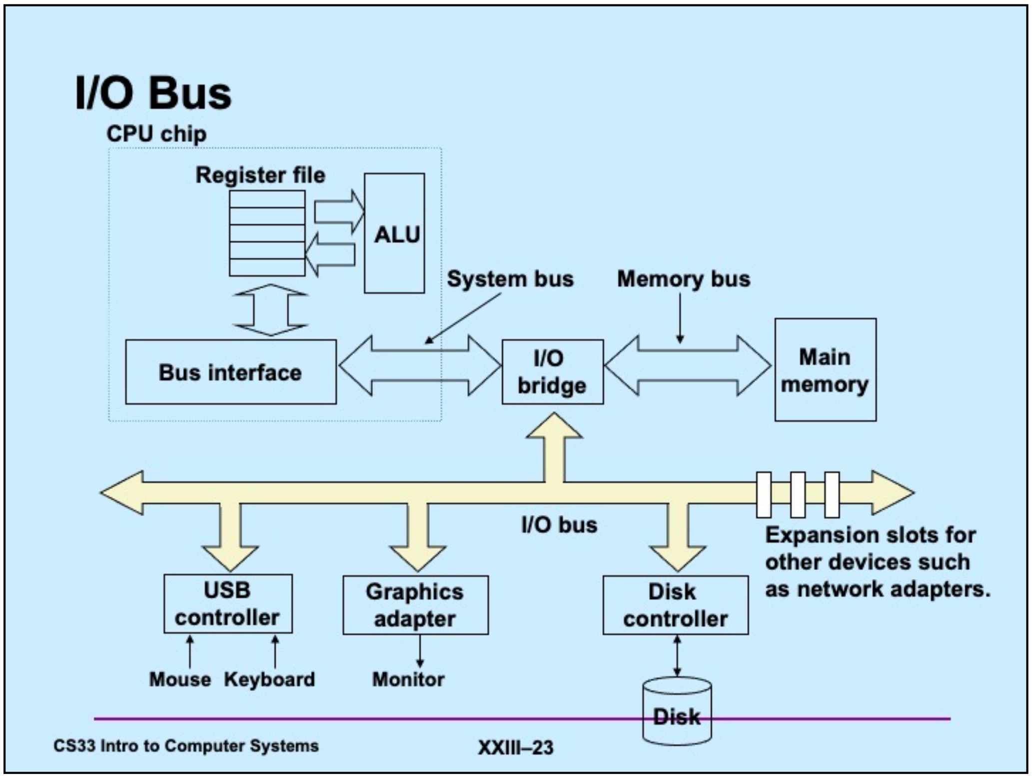
For the purposes of this quiz, you may assume **π** is 3 and thus the circumference of the disk is 15 inches or 1.25 feet. There are 5280 feet in a mile. Thus, if the disk were a wheel, it would have to rotate a bit over 4200 times to traverse a mile.

Logical Disk Blocks

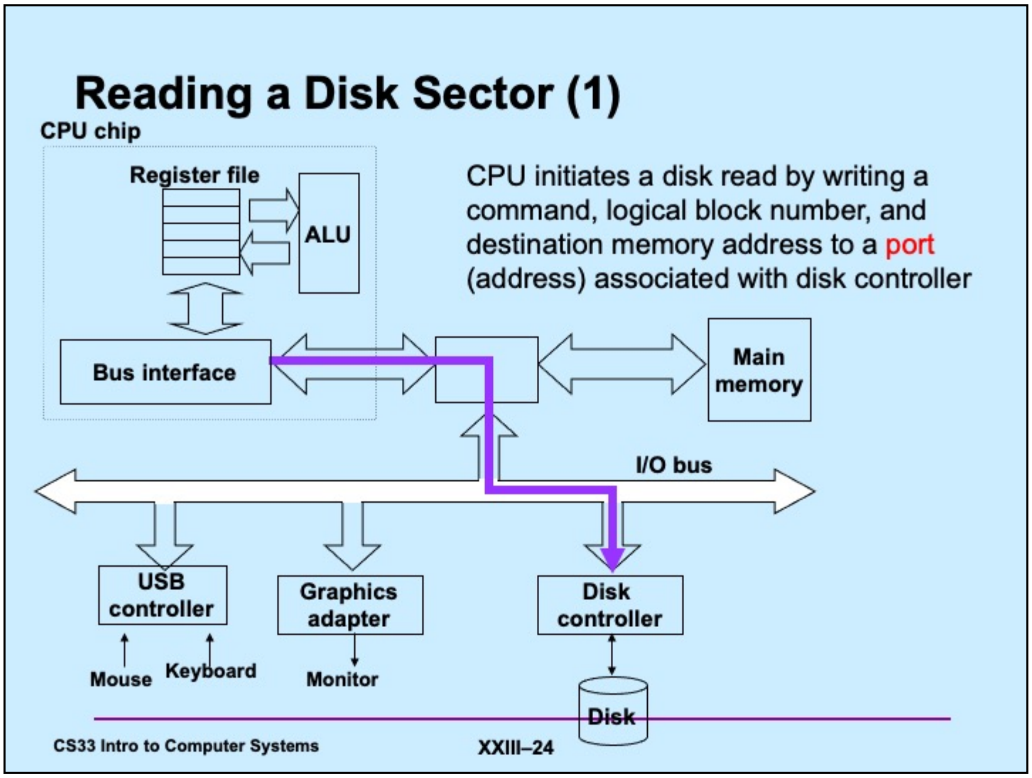
- **Modern disks present a simple abstract view of the complex sector geometry:**
  - the set of available sectors is modeled as a sequence of b-sized **logical blocks** (0, 1, 2, ...)
- **Mapping between logical blocks and actual (physical) sectors**
  - maintained by hardware/firmware device called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

Supplied by CMU.

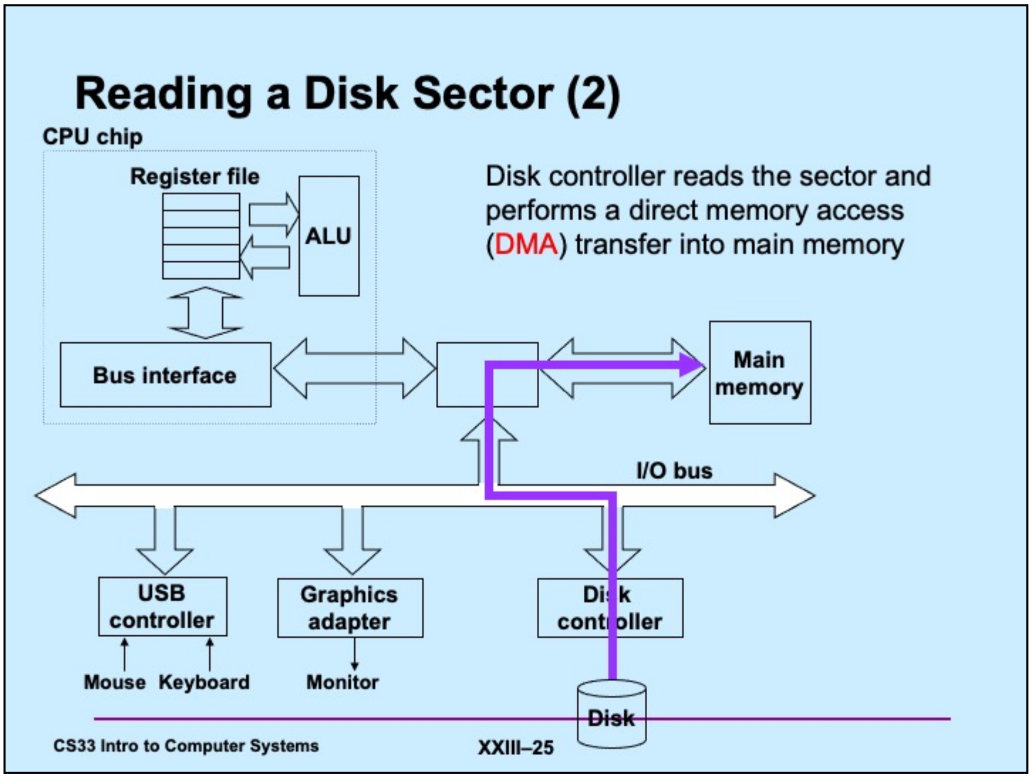
The purpose of the spare cylinders is to provide additional space in case certain sectors go bad (which is not uncommon).



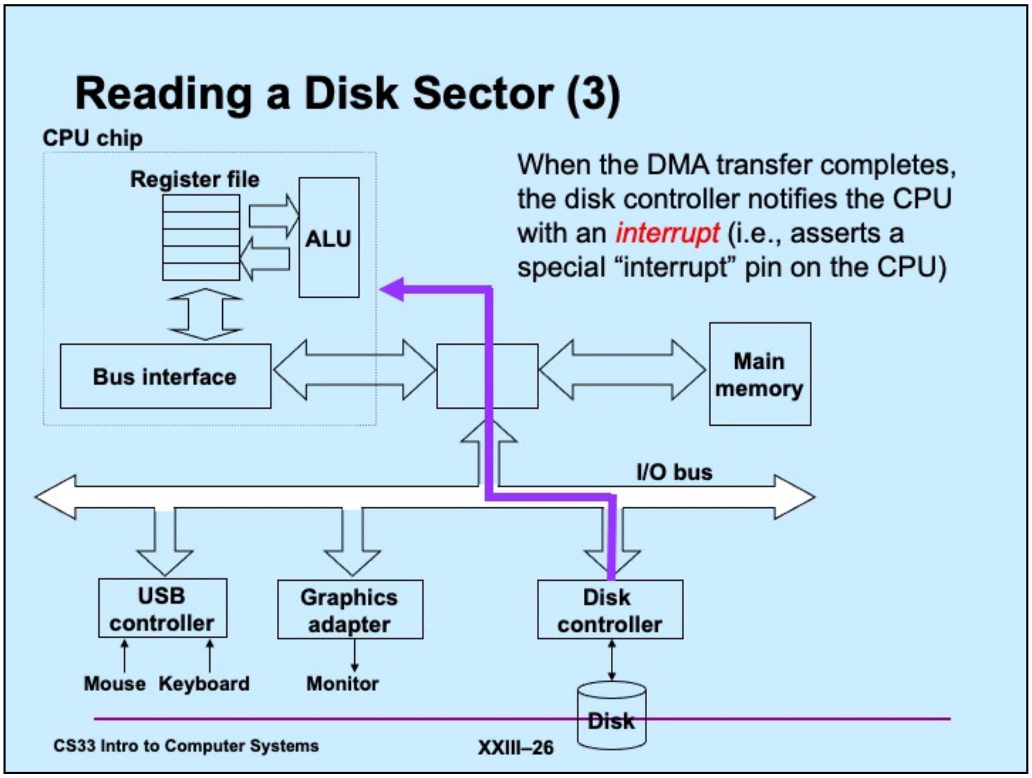
Supplied by CMU.



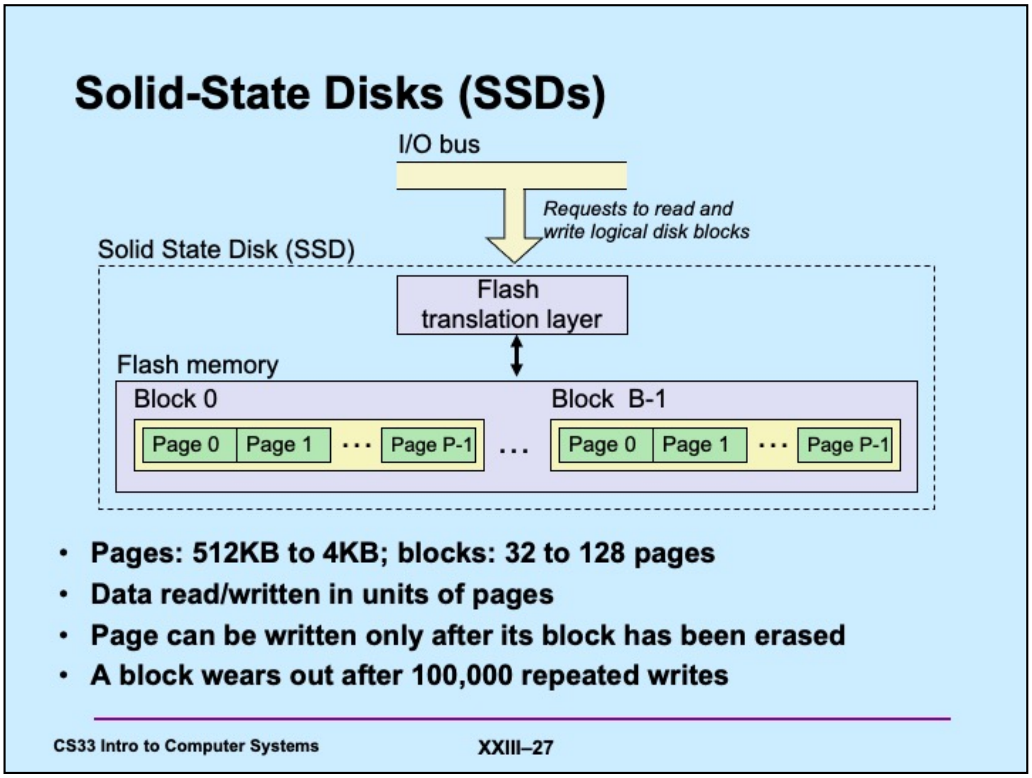
Supplied by CMU.



Supplied by CMU.



Supplied by CMU.



Supplied by CMU.

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

Supplied by CMU.

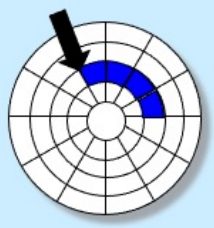
SSD Tradeoffs vs Rotating Disks

- 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
  - in 2021, about 2-3 times more expensive per byte
- Applications
  - smart phones, laptops, Apple “Fusion” drives

Adapted from a slide supplied by CMU.

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

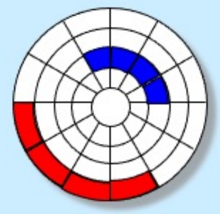
We have two files on the same (rotating) disk. The first file's data resides in consecutive sectors on one track, the second in consecutive sectors on another track. It takes a total of *t* seconds to read all of the first file then all of the second file.

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 (within a factor of 2)

c) much 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 (within a factor of 2)
c) much 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

Supplied by CMU.

2019prices for SRAM varied a fair amount. In October it could be had for around $9/MB, if you bought in quantities of 1000 or more.

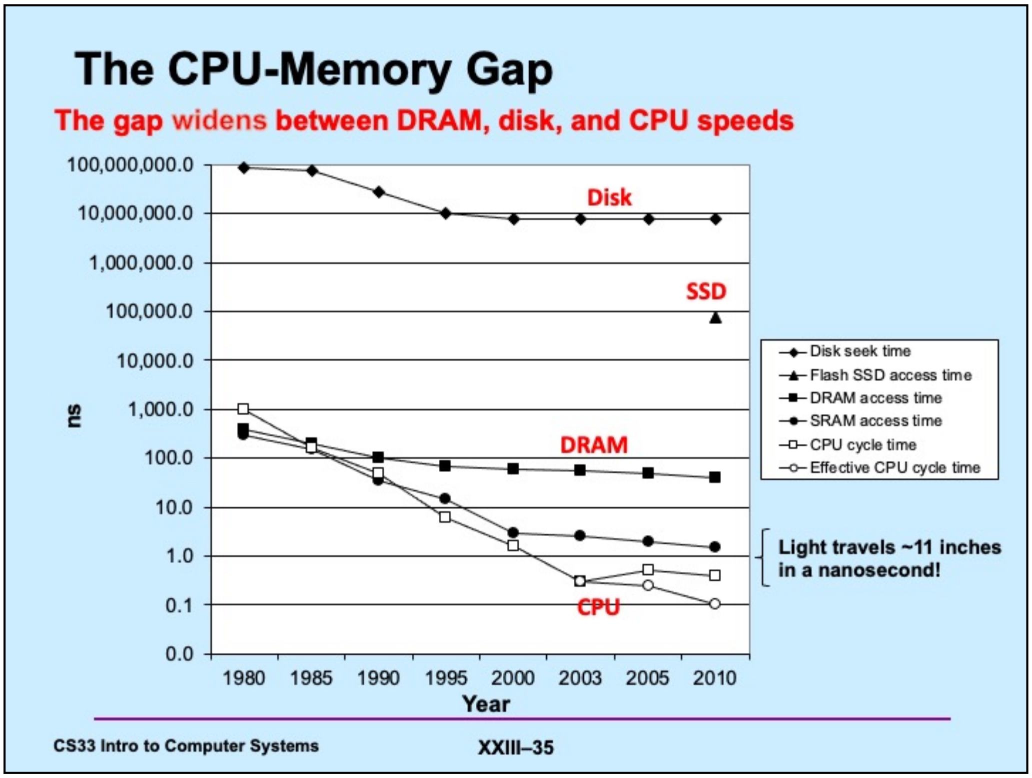
DRAM prices were as low as $.00075/MB, if bought in sufficient quantity. Today's prices are higher, due to Covid-related shortages.

CPU Clock Rates

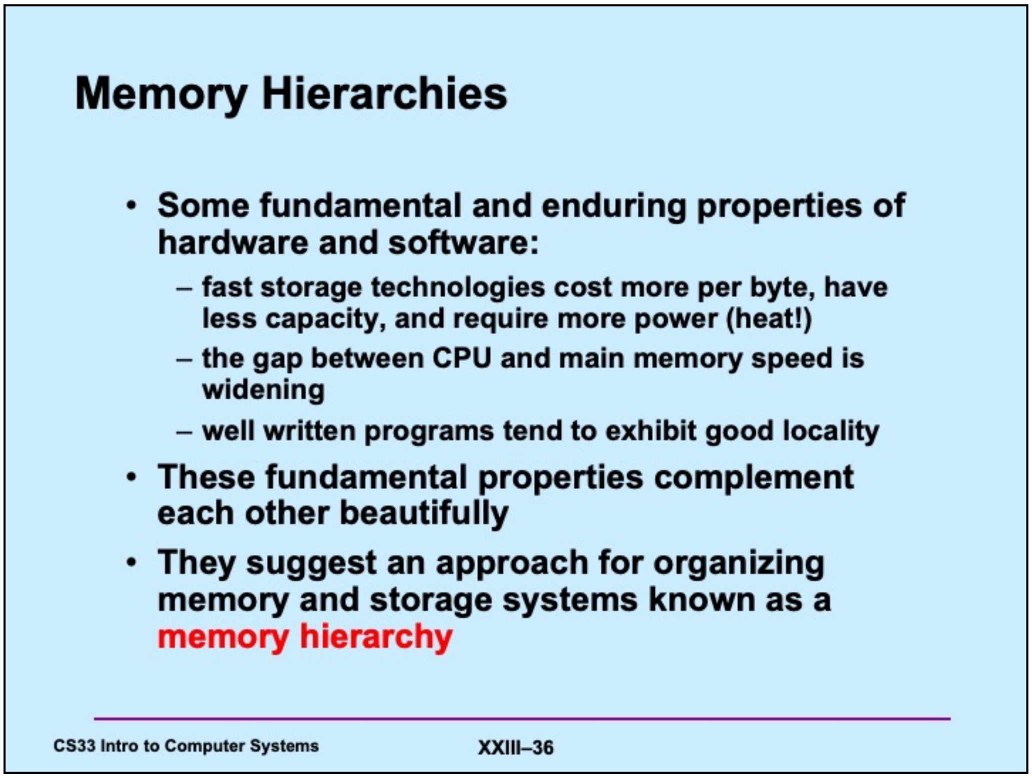
Inflection point in computer history
when designers hit the "Power Wall"

	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

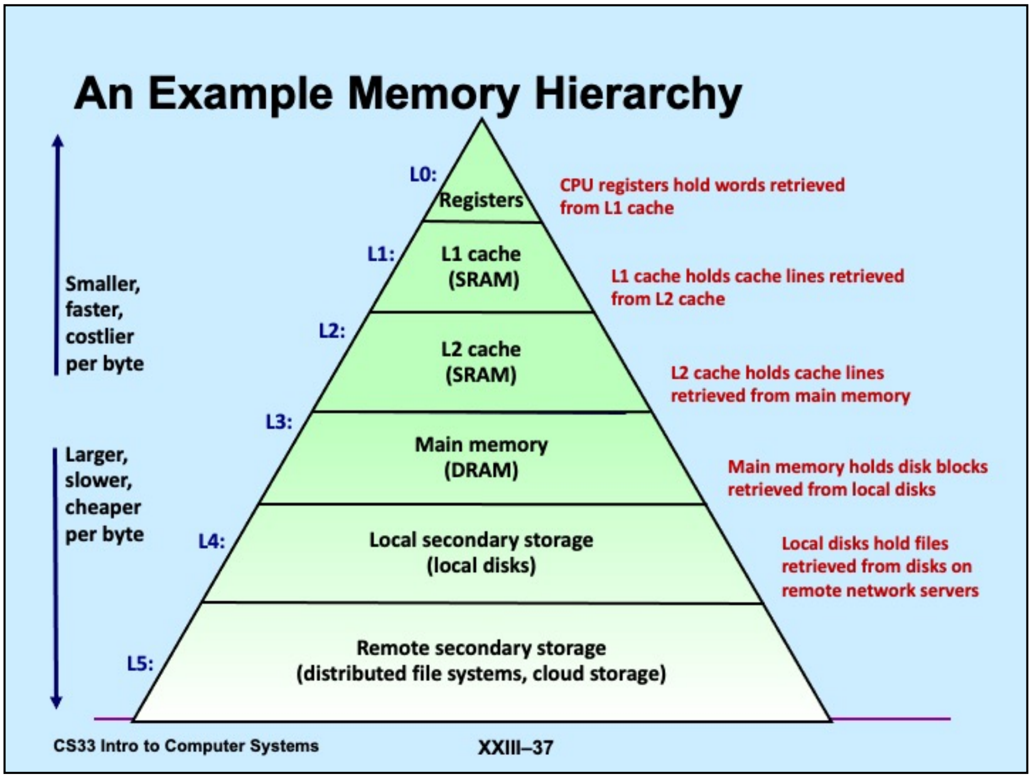
Supplied by CMU.



Supplied by CMU.



Supplied by CMU.



Supplied by CMU.

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

This analogy is from http://duartes.org/gustavo/blog/post/what-your-computer-does-while-you-wait (definitely worth reading!).

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 shell 1 implementation

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

CS 33

Linkers (1)

gcc Steps

1) **Compile**
   - to start here, supply .c file
   - to stop here: `gcc -S` (produces .s file)
   - if not stopping here, gcc compiles directly into a .o file, bypassing the assembler
2) **Assemble**
   - to start here, supply .s file
   - to stop here: `gcc -c` (produces .o file)
3) **Link**
   - to start here, supply .o file

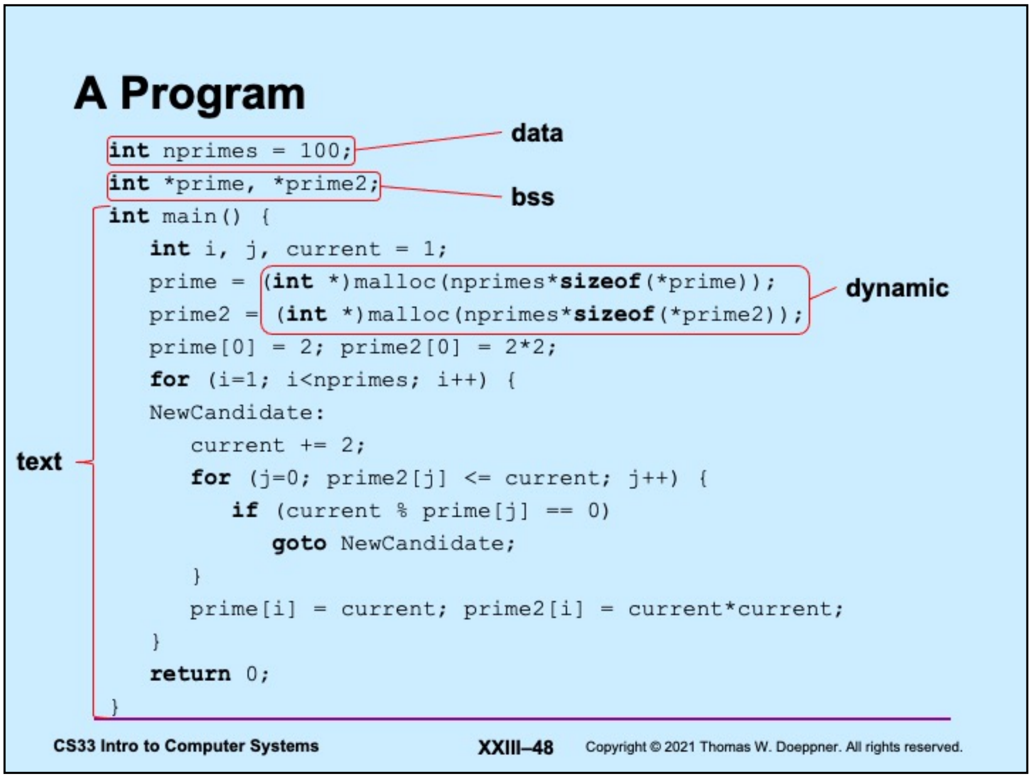
The Linker

- An executable program is one that is ready to be loaded into memory
- The linker (known as ld: /usr/bin/ld) creates such executables from:
  - object files produced by the compiler/assembler
  - collections of object files (known as libraries or archives)
  - and more we'll get to soon ...

The technology described here is current as of around 1990 and is known as static linking. We discuss static linking first, then move on to dynamic linking (in a few weeks), which is commonplace today.

Linker's Job

- Piece together components of program
  - arrange within address space
    - code (and read-only data) goes into text region
    - initialized data goes into data region
    - uninitialized data goes into bss region
- Modify address references, as necessary



The code is an implementation of the "sieve of Eratosthenes", an early (~200 BC) algorithm for enumerating prime numbers.

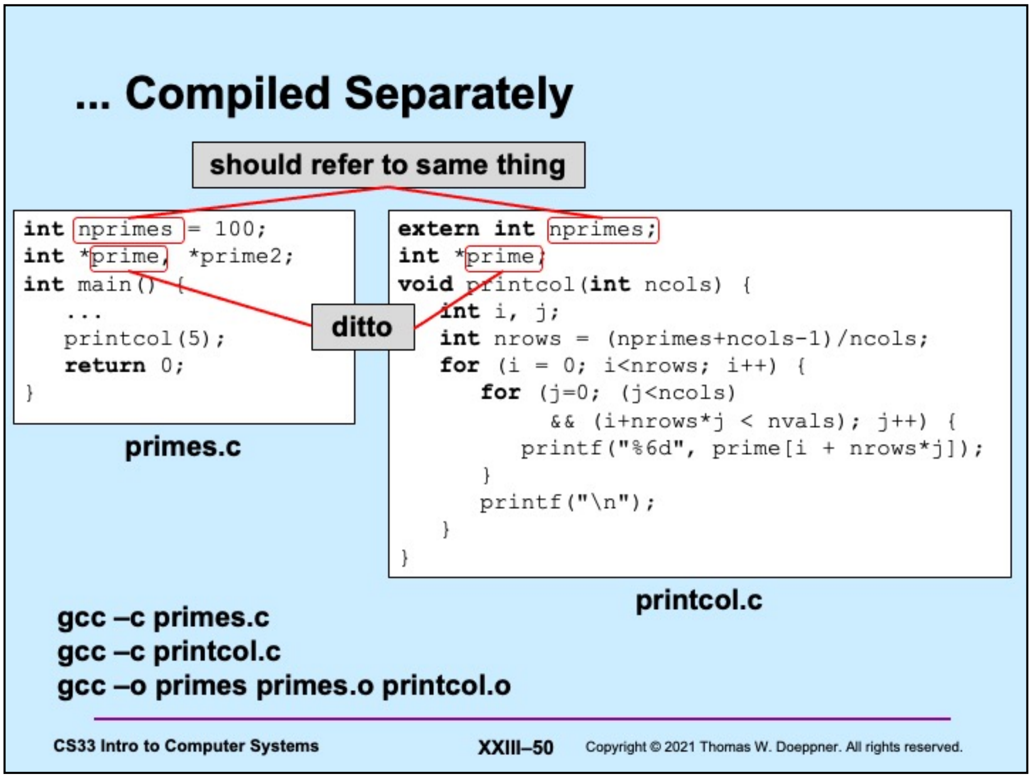
The *malloc* function allocates storage within the dynamic region. We discuss it in detail in an upcoming lecture.

... with Output

```
int nprimes = 100;
int *prime, *prime2;
int main() {
   ...
   printcol(5);
   return 0;
}

void printcol(int ncols) {
   int i, j;
   int nrows = (nprimes+ncols-1)/ncols;
   for (i = 0; i<nrows; i++) {
      for (j=0; (j<ncols) && (i+nrows*j < nvals); j++) {
         printf("%6d", prime[i + nrows*j]);
      }
      printf("\n");
   }
}
```


What this program actually does isn't all that important for our discussion. However, it prints out the vector of prime numbers in multiple columns.



In the first two invocations of gcc, the "-c" flag tells it to compile the C code and produce an object (".o") file, but not to go any further (and thus not to produce an executable program). In the third invocation, gcc invokes the ld (linker) program to combine the two object files into an executable program. As we discuss soon, it will also bring in code (such as printf) from libraries.