Lecture 10:
I/O, Storage, Performance

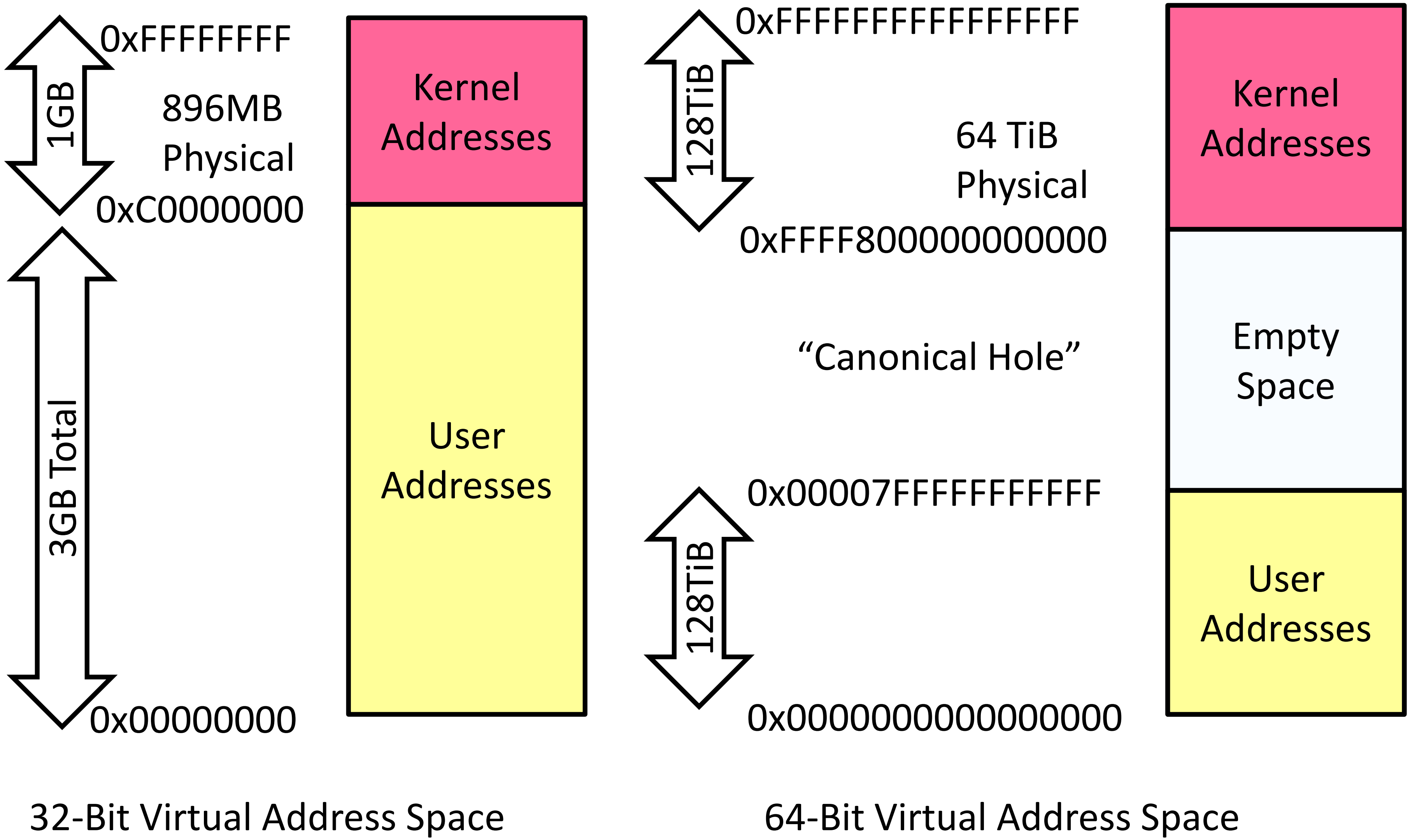
Bo Tang @ 2020, Spring

Memory Management

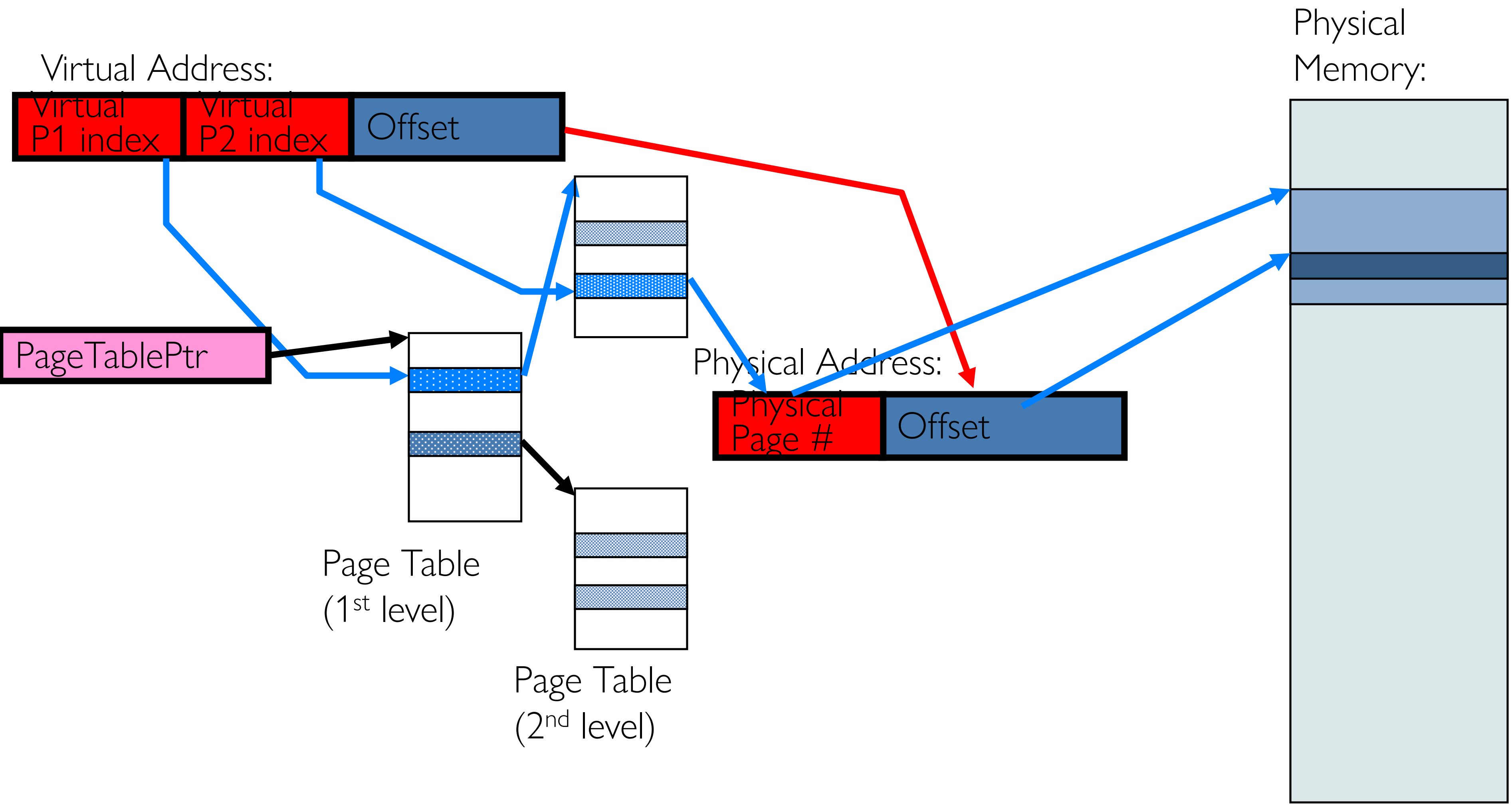
Linux Memory Details?

- Memory management in Linux considerably more complex that the previous indications
- Memory Zones: physical memory categories
  - ZONE_DMA: < 16MB memory, DMAable on ISA bus
  - ZONE_NORMAL: 16MB $\rightarrow$ 896MB (mapped at 0xC0000000)
  - ZONE_HIGHMEM: Everything else (> 896MB)
- Each zone has 1 freelist, 2 LRU lists (Active/Inactive)
- Many different types of allocation
  - SLAB allocators, per-page allocators, mapped/unmapped
- Many different types of allocated memory:
  - Anonymous memory (not backed by a file, heap/stack)
  - Mapped memory (backed by a file)
- Allocation priorities
  - Is blocking allowed/etc

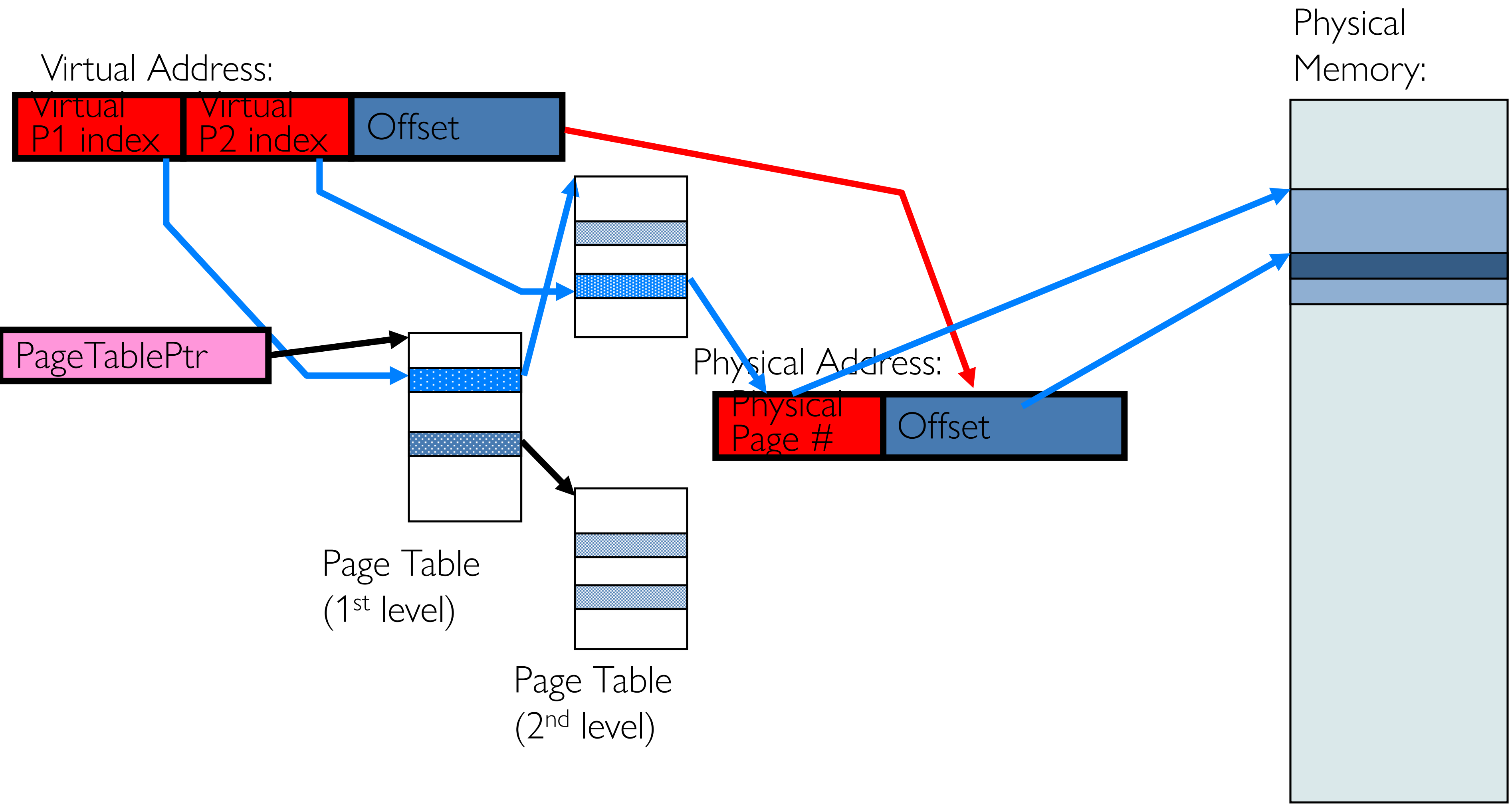
Linux Virtual memory map



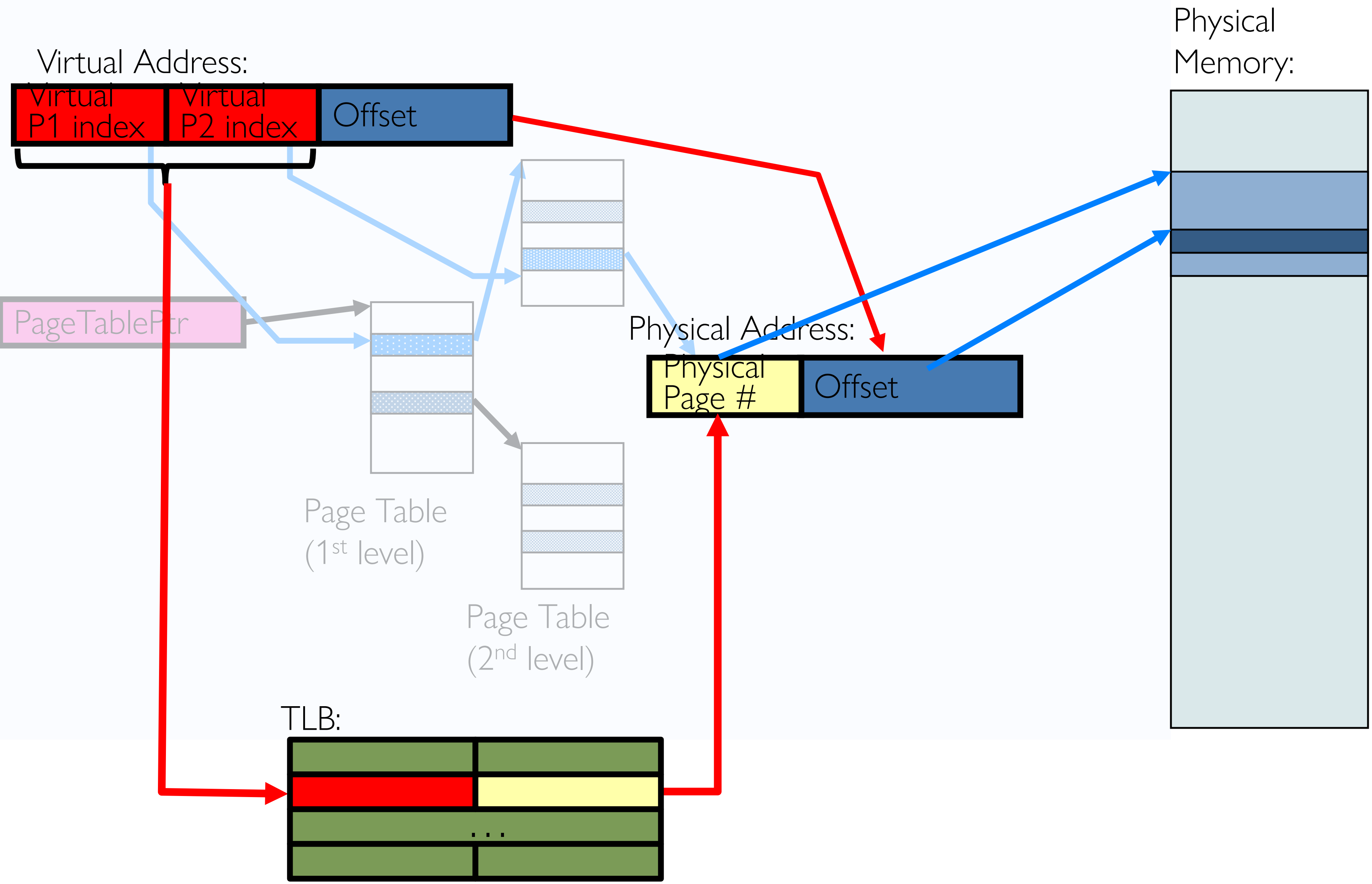
Putting All Together: Address Translation



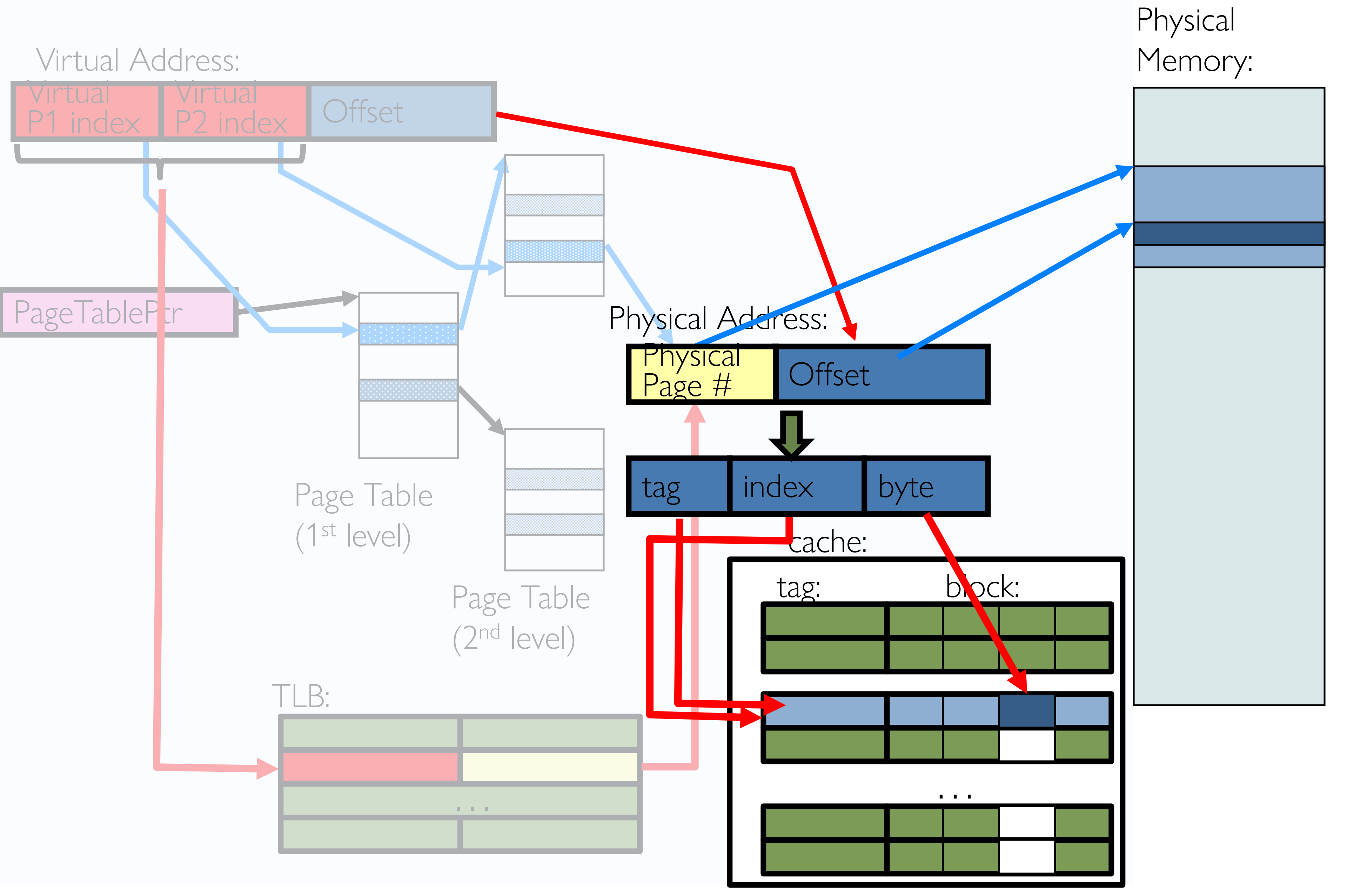
Putting All Together: Address Translation



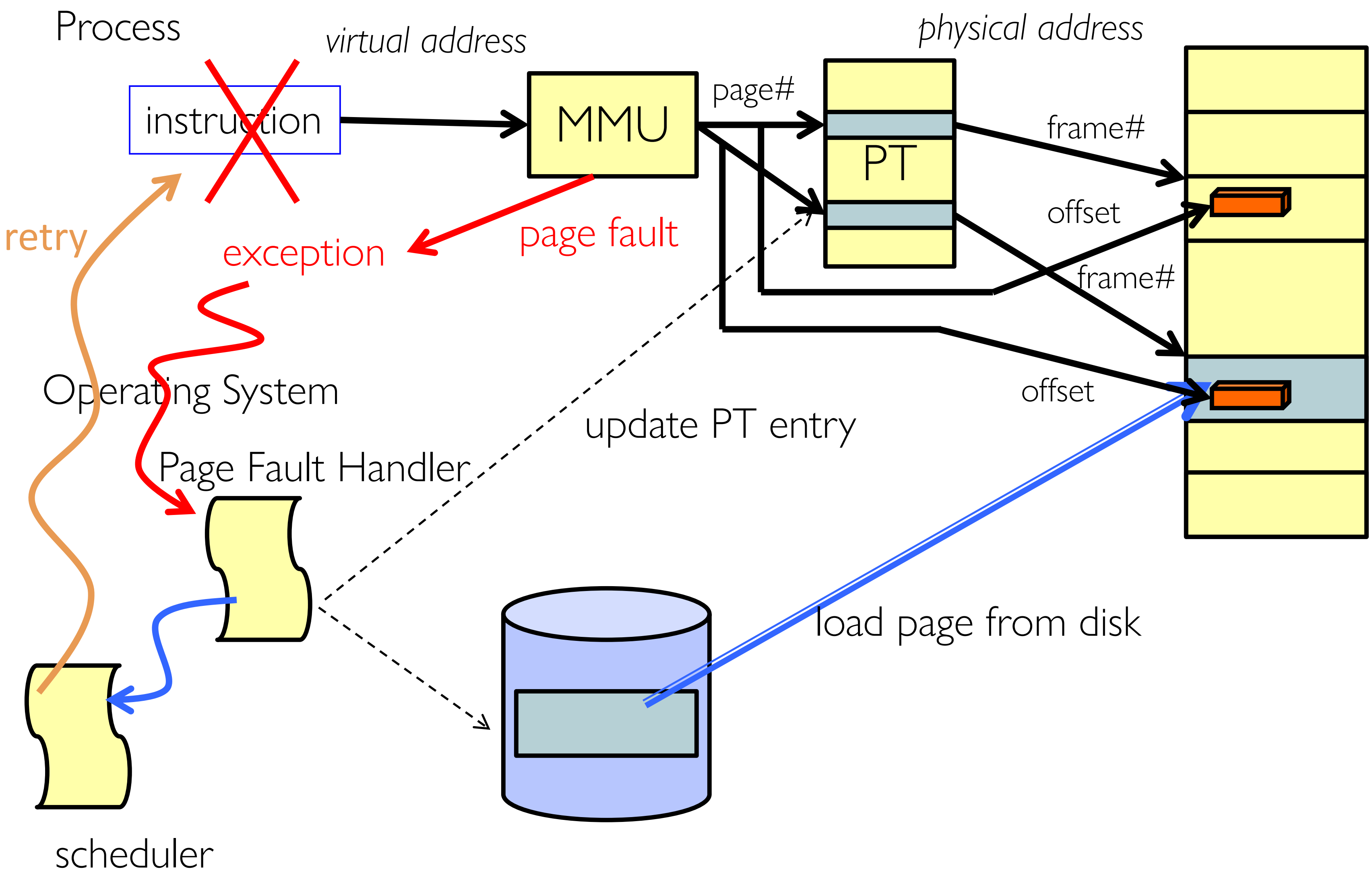
Putting Everything Together: TLB



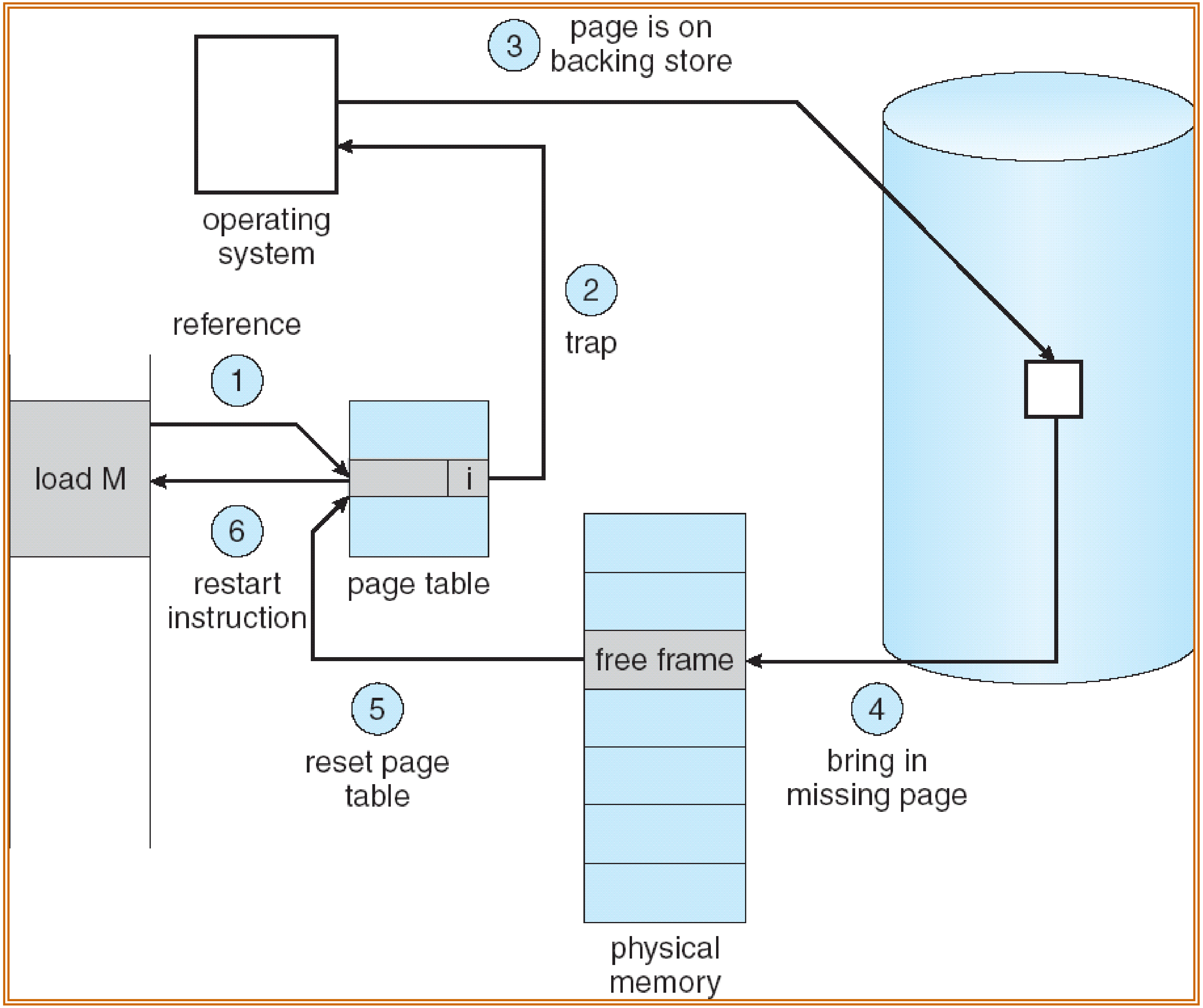
Putting Everything Together: Cache



In a picture

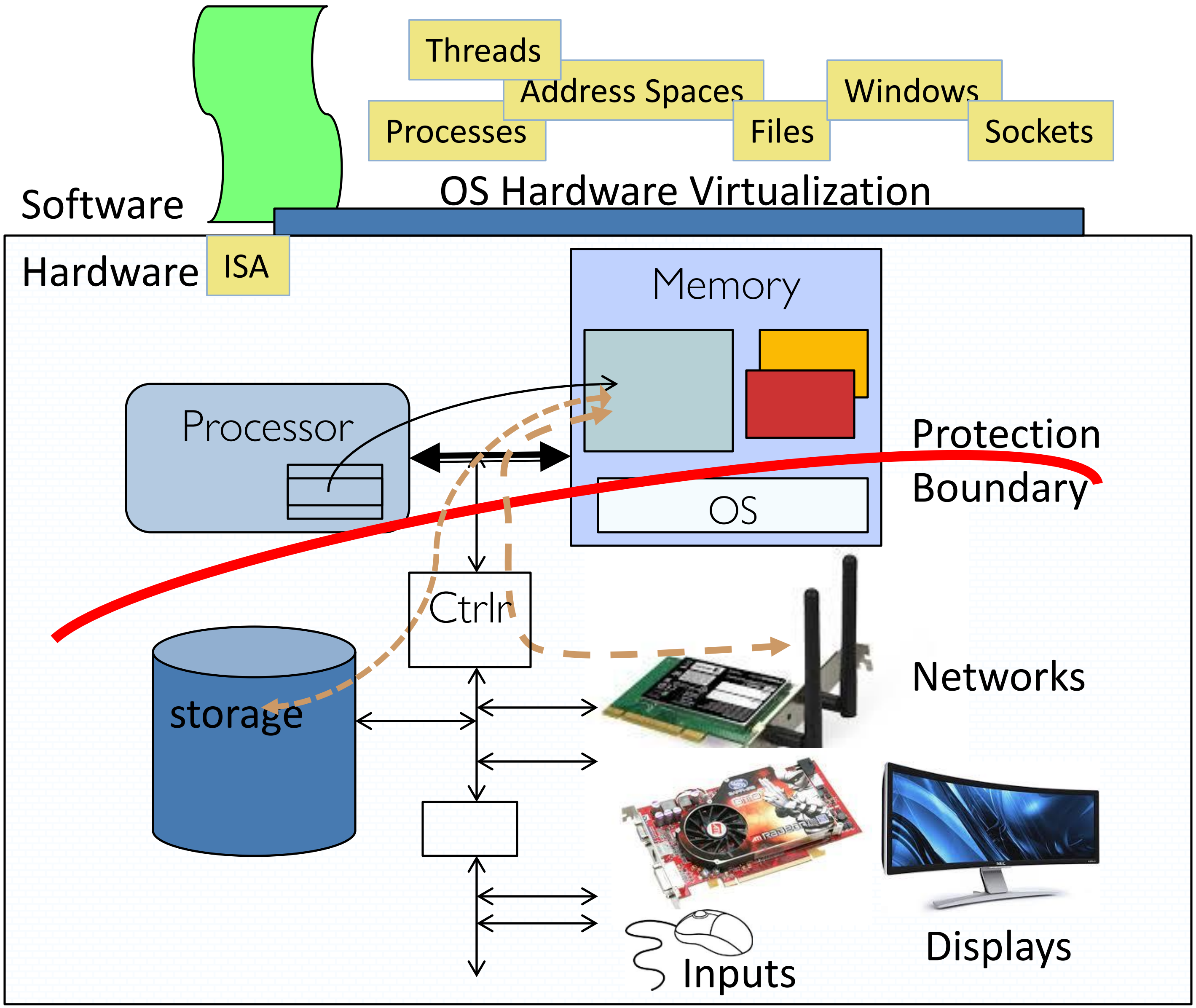

Process
virtual address
physical address
instruction
MMU
page#
PT
frame#
offset
frame#
offset
retry
exception
page fault
Operating System
update PT entry
Page Fault Handler
load page from disk
scheduler

Steps in Handling a Page Fault

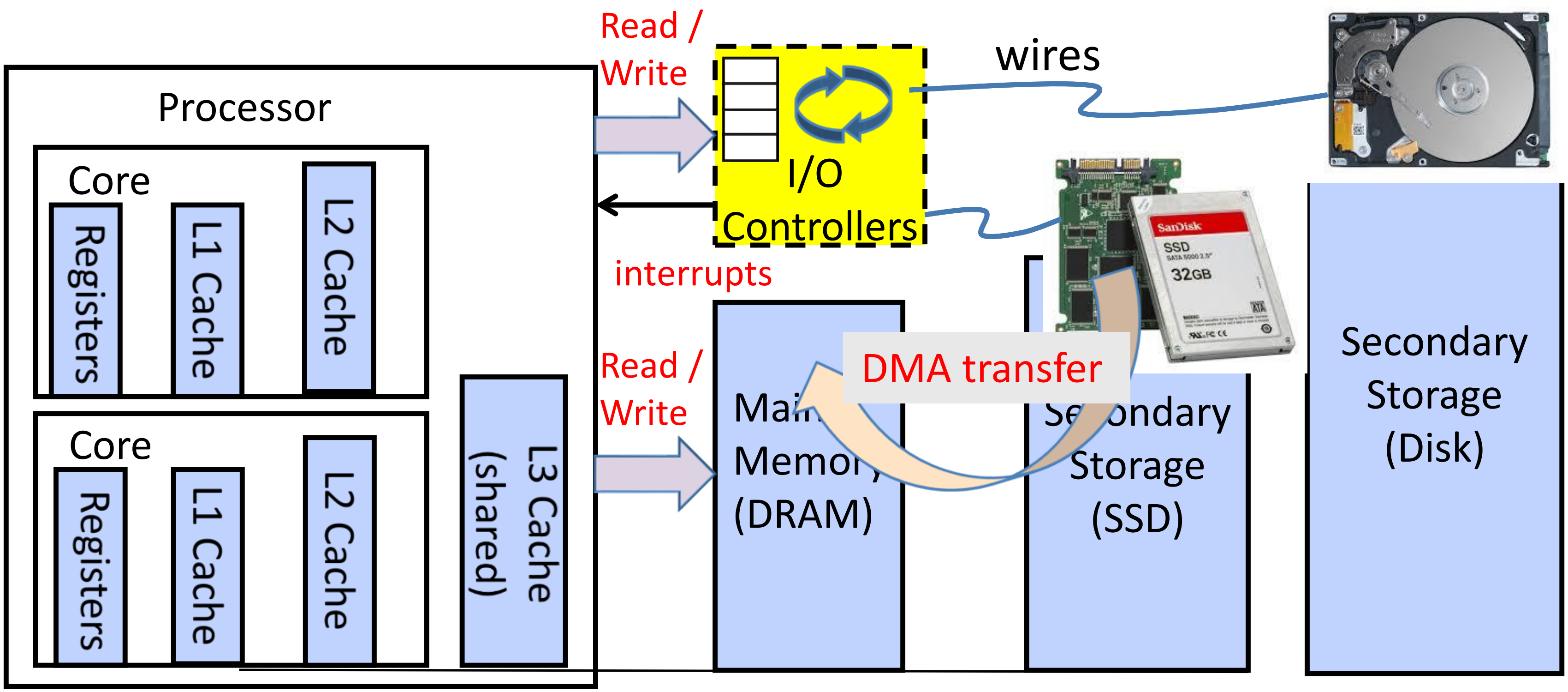


General I/O

OS Basics: I/O


Threads
Address Spaces
Windows
Processes
Files
Sockets
OS Hardware Virtualization
Software
Hardware
ISA
Memory
Processor
OS
Protection
Boundary
Ctrlr
storage
Networks
Displays
Inputs

In a Picture



- I/O devices you recognize are supported by I/O Controllers
- Processors accesses them by reading and writing IO registers as if they were memory
  - Write commands and arguments, read status and results

The Requirements of I/O

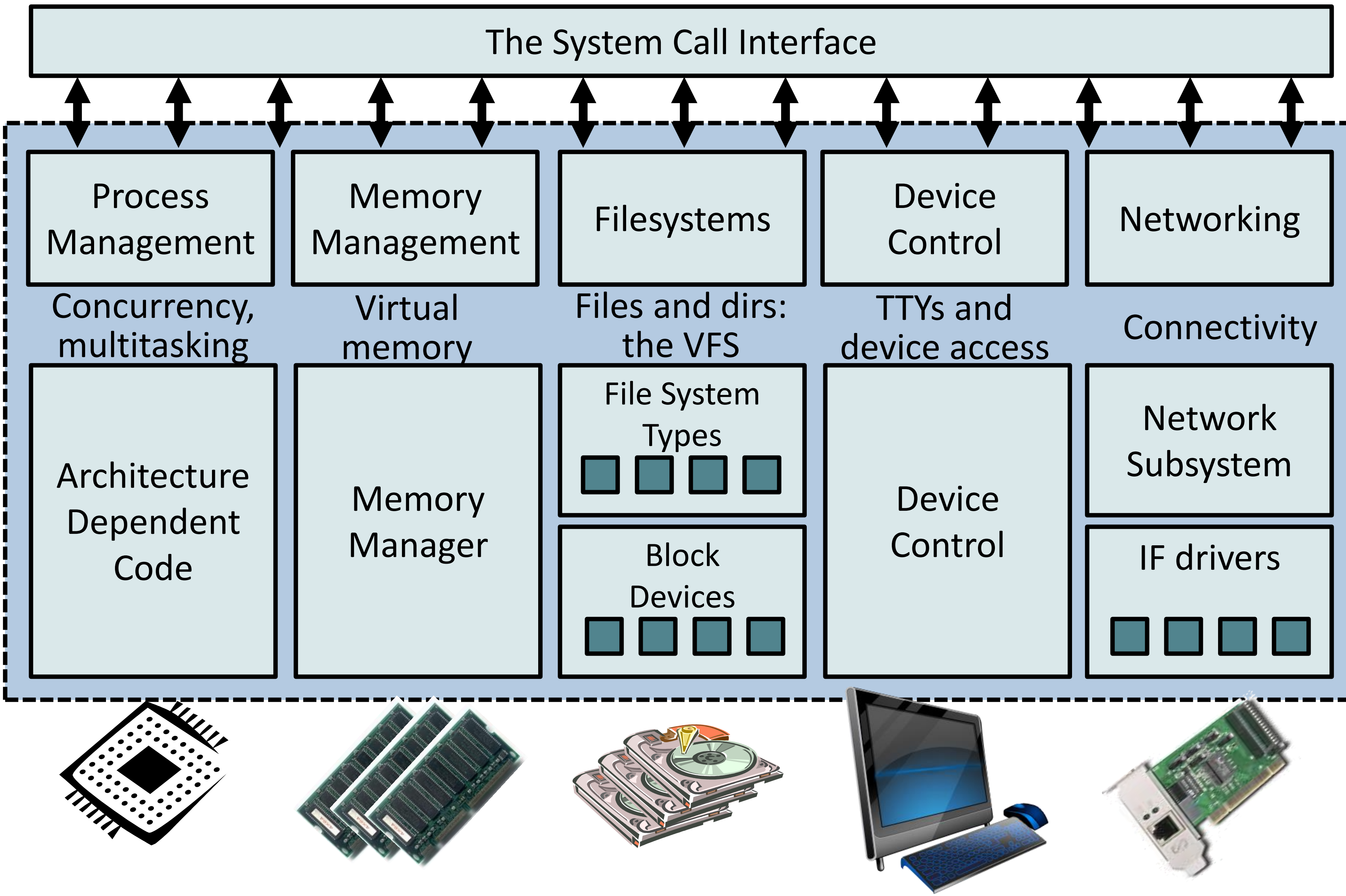
- So far in this course:
  - We have learned how to manage CPU and memory

- What about I/O?
  - Without I/O, computers are useless
  - But… thousands of devices, each slightly different
    - How can we standardize the interfaces to these devices?
  - Devices unreliable: media failures and transmission errors
    - How can we make them reliable?
  - Devices unpredictable and/or slow
    - How can we manage them if we do not know what they will do or how they will perform?

Operational Parameters for I/O

- Data granularity: Byte vs. Block
  - Some devices provide single byte at a time (e.g., keyboard)
  - Others provide whole blocks (e.g., disks, networks, etc.)
- Access pattern: Sequential vs. Random
  - Some devices must be accessed sequentially (e.g., tape)
  - Others can be accessed “randomly” (e.g., disk, cd, etc.)
    - Fixed overhead to start transfers
  - Some devices require continual monitoring
  - Others generate interrupts when they need service
- Transfer Mechanism: Programmed IO and DMA

Kernel Device Structure



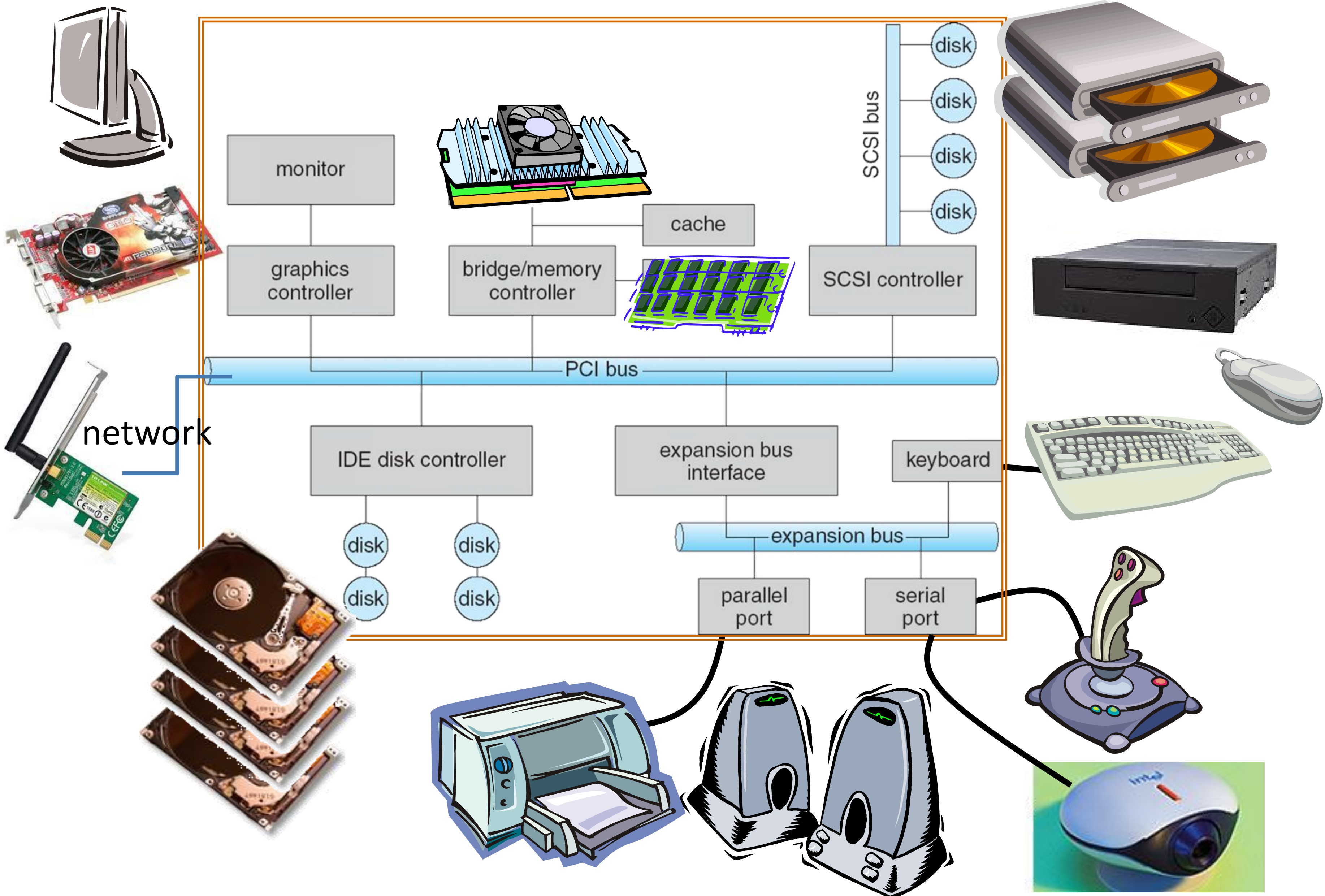
The Goal of the I/O Subsystem

- Provide uniform interfaces, despite wide range of different devices
  - This code works on many different devices:

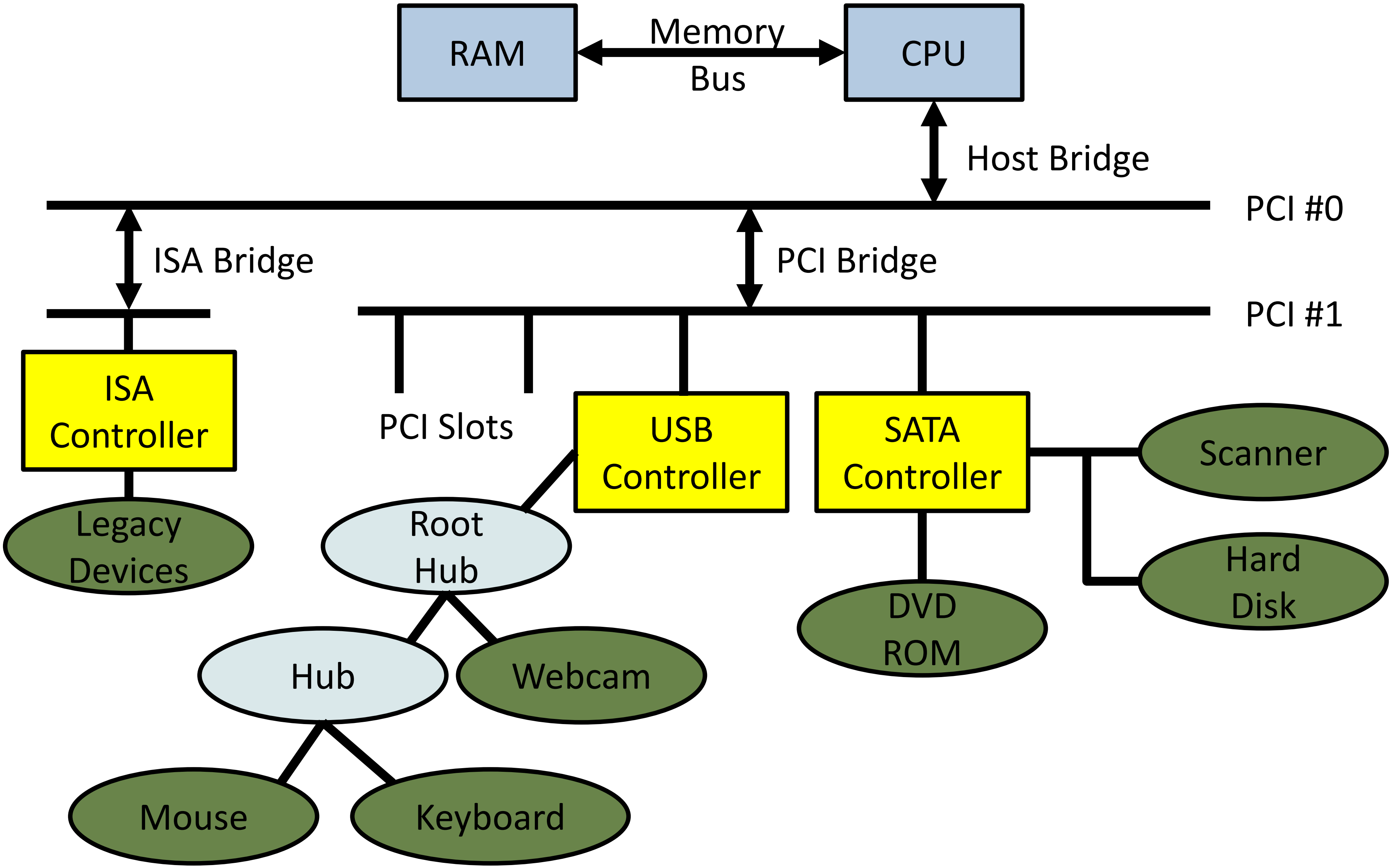
```
FILE fd = fopen("/dev/something", "rw");
for (int i = 0; i < 10; i++) {
  fprintf(fd, "Count %d\n", i);
}
close(fd);
```

  - Why? Because code that controls devices ("device driver") implements standard interface
- We will try to get a flavor for what is involved in actually controlling devices in rest of lecture
  - Can only scratch surface!

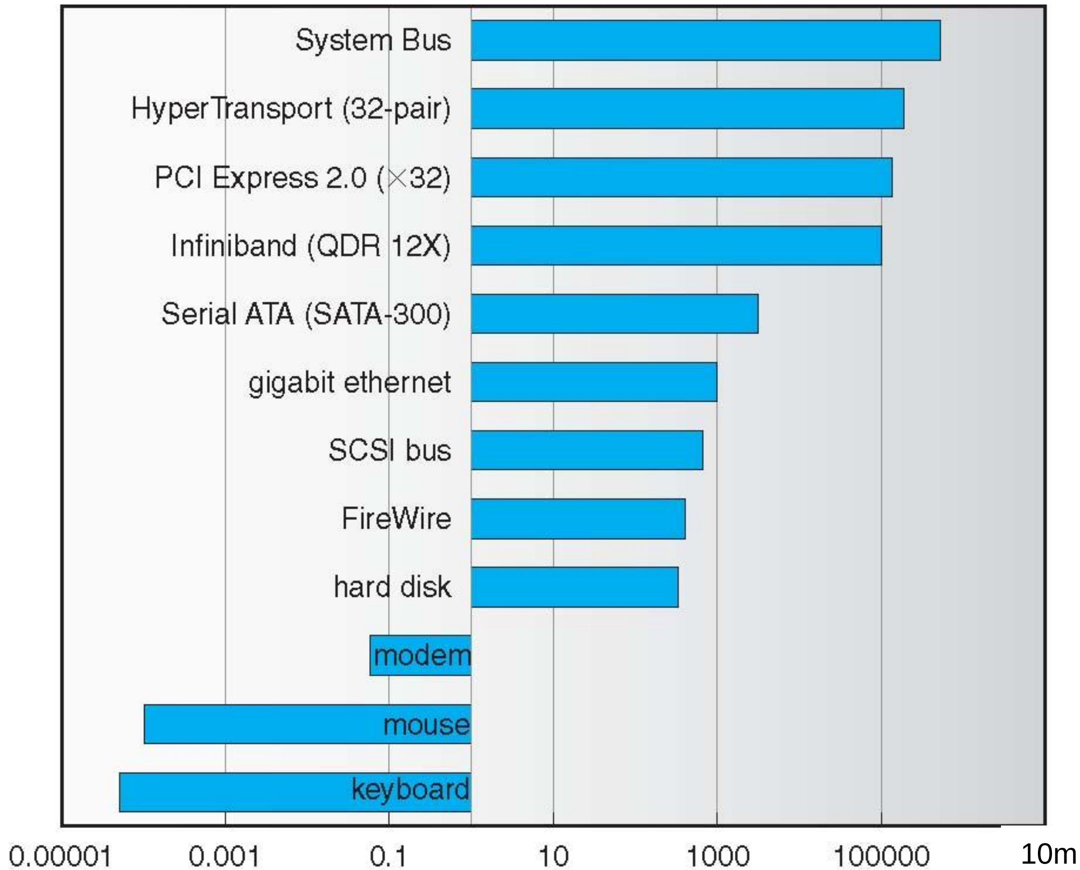
Modern I/O Systems



Example: PCI Architecture

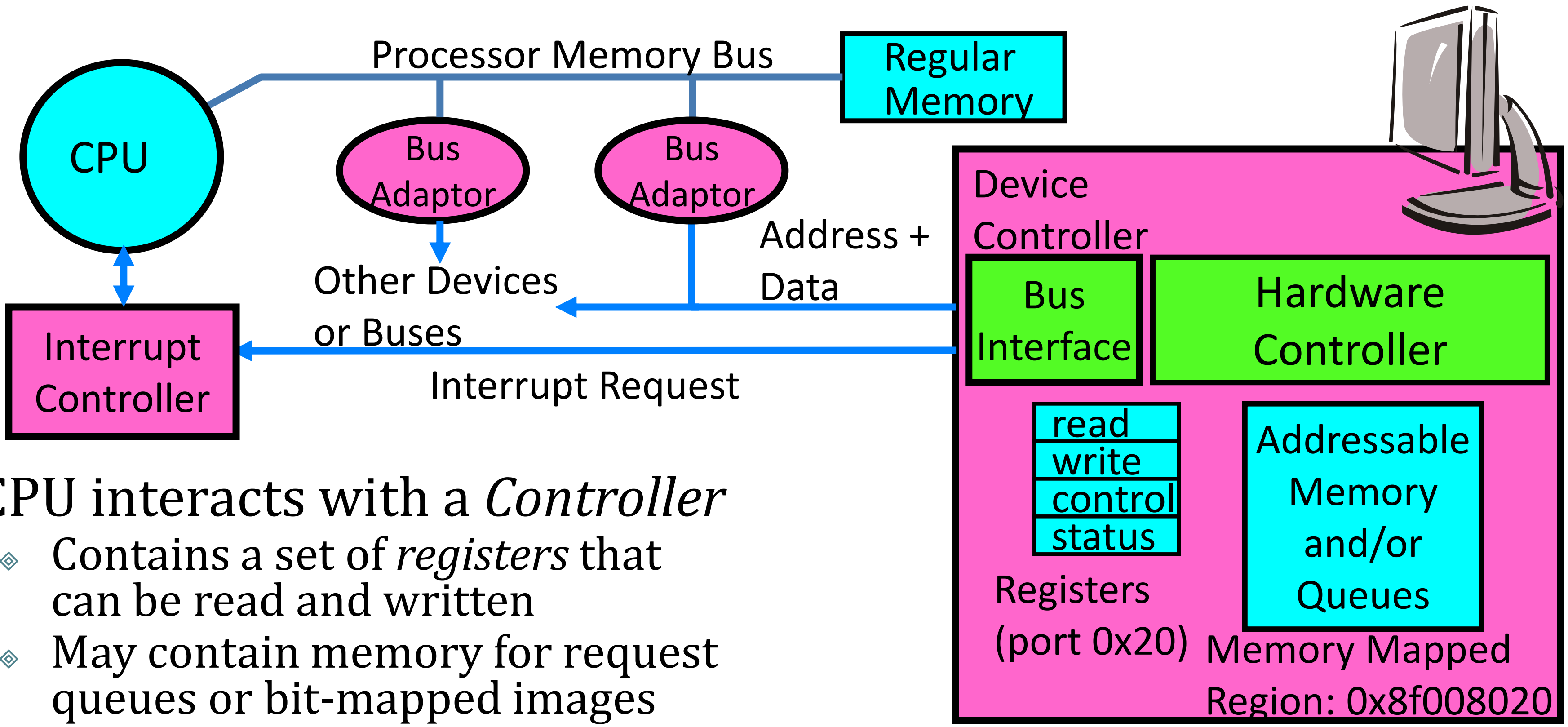


Example Device-Transfer Rates in Mb/s (Sun En. 6000)



- Device Rates vary over 12 orders of magnitude !!!
  - System better be able to handle this wide range
  - Better not have high overhead/byte for fast devices!
  - Better not waste time waiting for slow devices

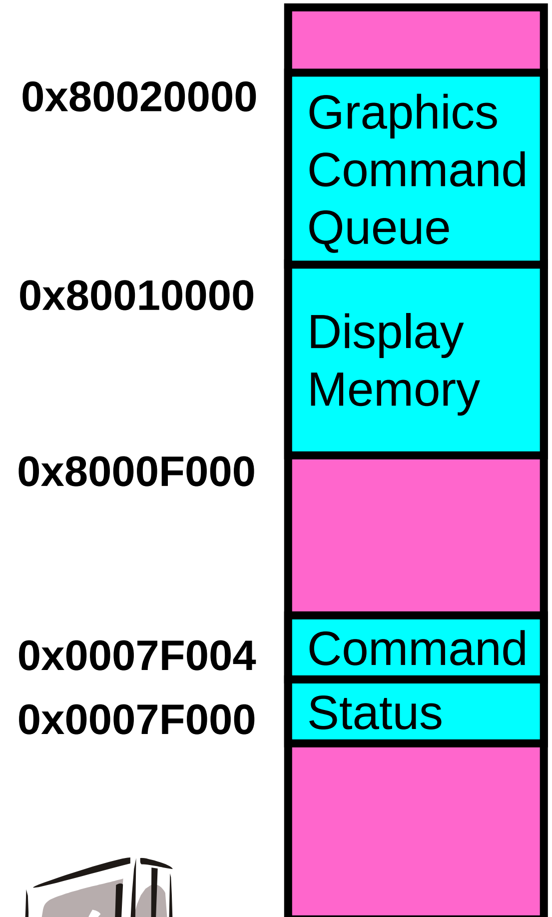
How does the processor talk to the device?

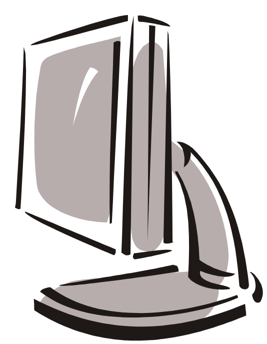


- CPU interacts with a *Controller*
  - Contains a set of *registers* that can be read and written
  - May contain memory for request queues or bit-mapped images
- Regardless of the complexity of the connections and buses, processor accesses registers in two ways:
  - I/O instructions: in/out instructions
    - Example from the Intel architecture: `out 0x21,AL`
  - Memory mapped I/O: load/store instructions
    - Registers/memory appear in physical address space
    - I/O accomplished with load and store instructions

Memory-Mapped Display Controller

- Memory-Mapped:
  - Hardware maps control registers and display memory into physical address space
    - Addresses set by HW jumpers or at boot time
  - Simply writing to display memory (also called the "frame buffer") changes image on screen
    - Addr: `0x8000F000 – 0x8000FFFF`
  - Writing graphics description to cmd queue
    - Say enter a set of triangles describing some scene
    - Addr: `0x80010000 – 0x8001FFFF`
  - Writing to the command register may cause on-board graphics hardware to do something
    - Say render the above scene
    - Addr: `0x0007F004`
- Can protect with address translation

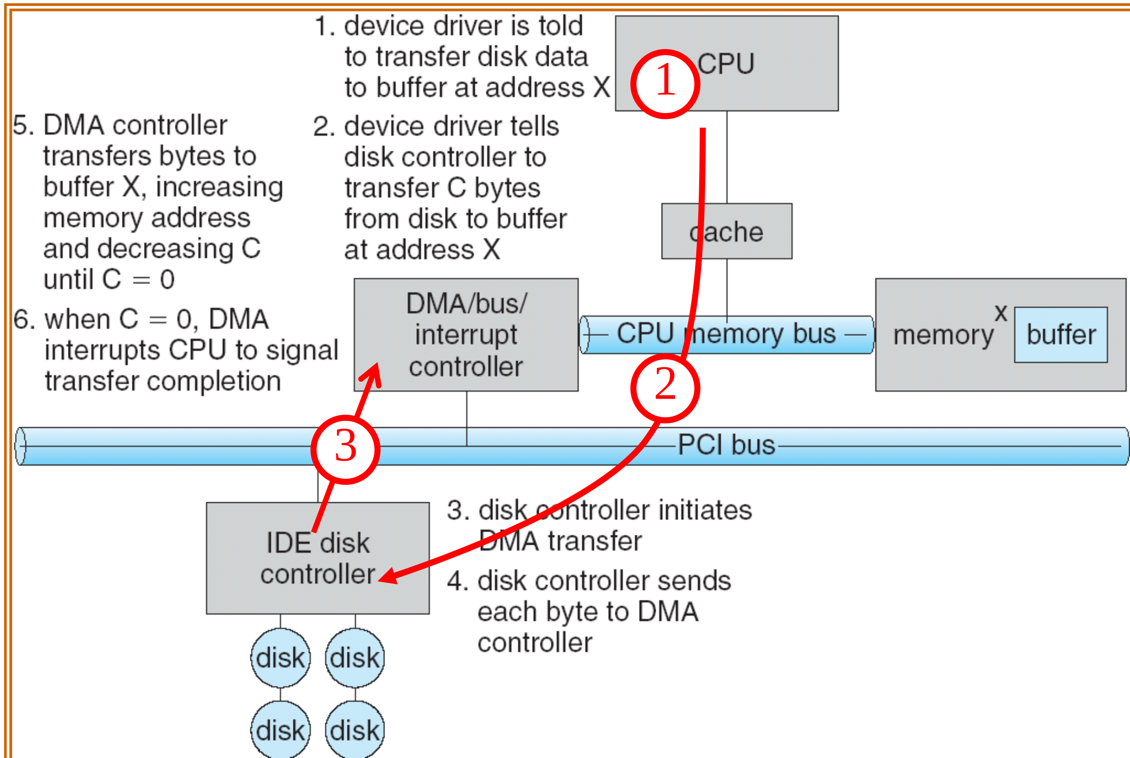




Physical Address Space

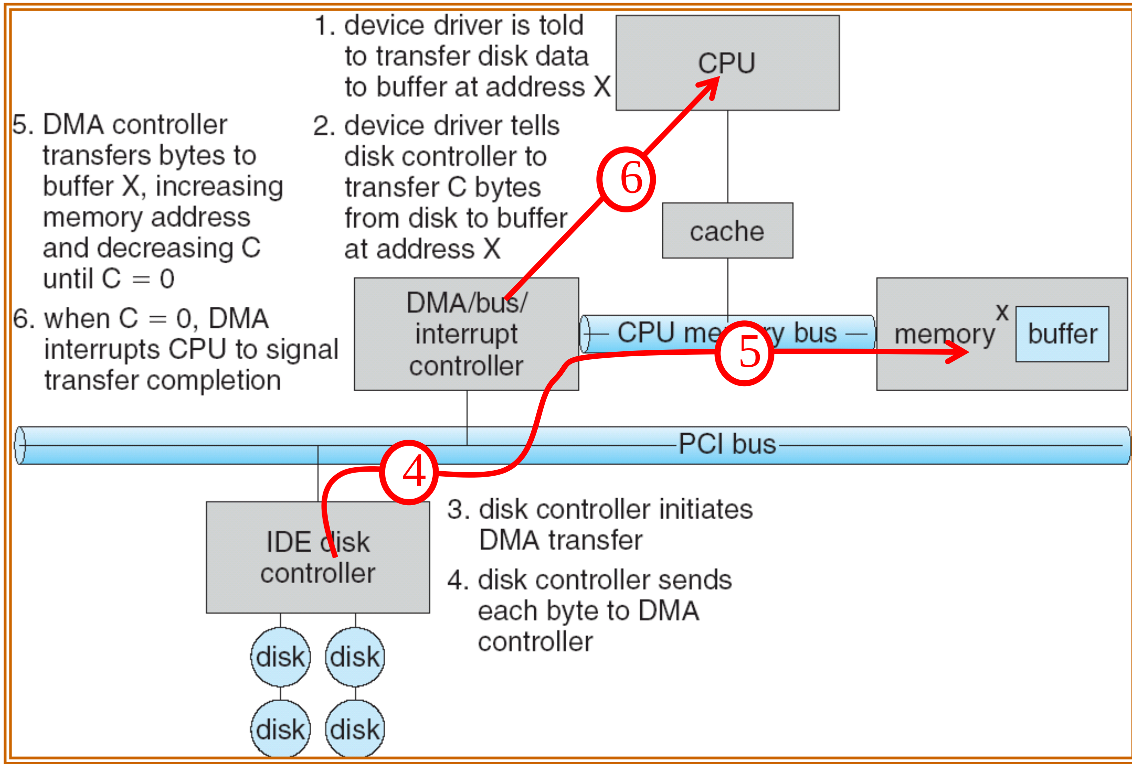
Transferring Data To/From Controller

- **Programmed I/O:**
  - Each byte transferred via processor in/out or load/store
  - Pro: Simple hardware, easy to program
  - Con: Consumes processor cycles proportional to data size
- **Direct Memory Access:**
  - Give controller access to memory bus
  - Ask it to transfer data blocks to/from memory directly
- Sample interaction with DMA controller



Transferring Data To/From Controller

- **Programmed I/O:**
  - Each byte transferred via processor in/out or load/store
  - Pro: Simple hardware, easy to program
  - Con: Consumes processor cycles proportional to data size
- **Direct Memory Access:**
  - Give controller access to memory bus
  - Ask it to transfer data blocks to/from memory directly
- Sample interaction with DMA controller



I/O Device Notifying the OS

- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- I/O Interrupt:
  - Device generates an interrupt whenever it needs service
  - Pro: handles unpredictable events well
  - Con: interrupts relatively high overhead
- Polling:
  - OS periodically checks a device-specific status register
    - I/O device puts completion information in status register
  - Pro: low overhead
  - Con: may waste many cycles on polling if infrequent or unpredictable I/O operations
- Actual devices combine both polling and interrupts
  - For instance – High-bandwidth network adapter:
    - Interrupt for first incoming packet
    - Poll for following packets until hardware queues are empty

Device Drivers

- Device Driver: Device-specific code in the kernel that interacts directly with the device hardware
  - Supports a standard, internal interface
  - Same kernel I/O system can interact easily with different device drivers
  - Special device-specific configuration supported with the `ioctl()` system call
- Device Drivers typically divided into two pieces:
  - Top half: accessed in call path from system calls
    - implements a set of standard, cross-device calls like `open()`, `close()`, `read()`, `write()`, `ioctl()`, `strategy()`
    - This is the kernel's interface to the device driver
    - Top half will *start* I/O to device, may put thread to sleep until finished
  - Bottom half: run as interrupt routine
    - Gets input or transfers next block of output
    - May wake sleeping threads if I/O now complete

Life Cycle of An I/O Request

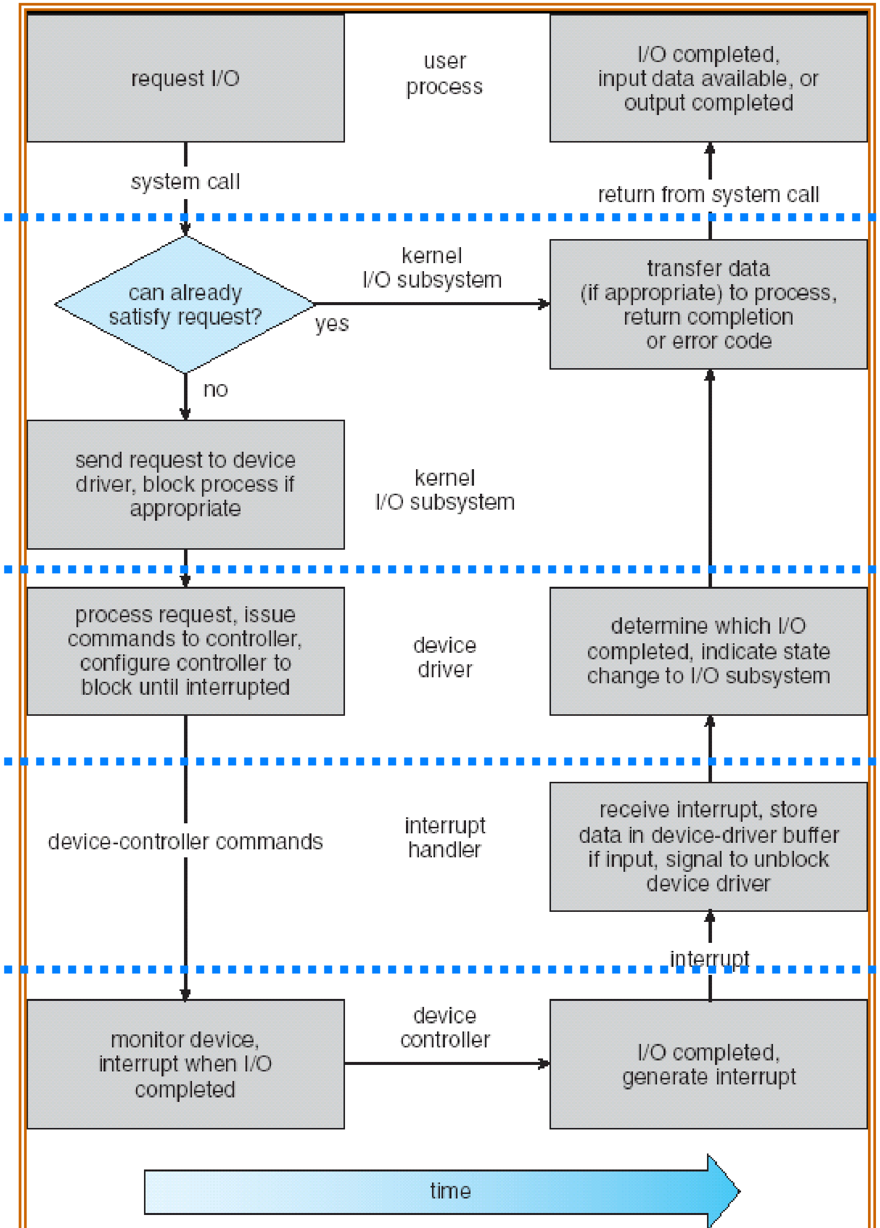
User Program

Kernel I/O Subsystem

Device Driver Top Half

Device Driver Bottom Half

Device Hardware

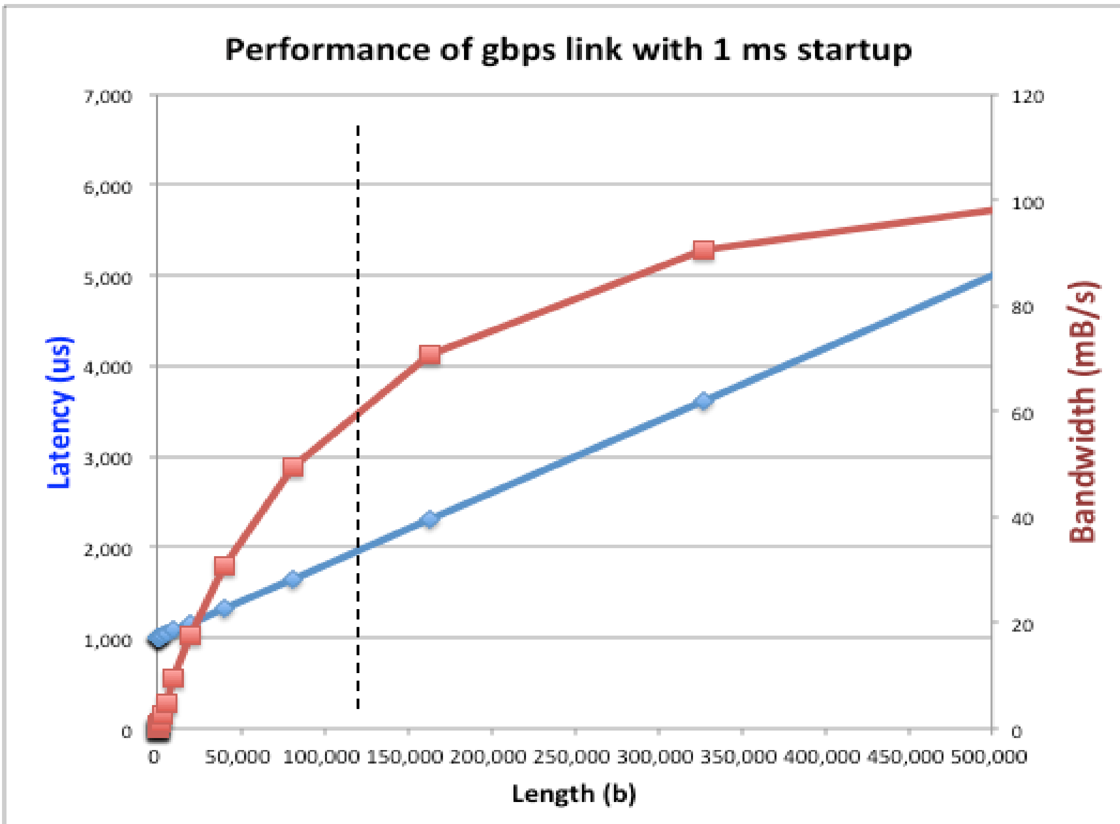


Basic Performance Concepts

- *Response Time* or *Latency*: Time to perform an operation(s)
- *Bandwidth* or *Throughput*: Rate at which operations are performed (op/s)
  - Files: MB/s, Networks: Mb/s, Arithmetic: GFLOP/s
- *Start up* or "Overhead": time to initiate an operation
- Most I/O operations are roughly linear in *n* bytes
  - Latency(n) = Overhead + n/TransferCapacity

Example (Fast Network)

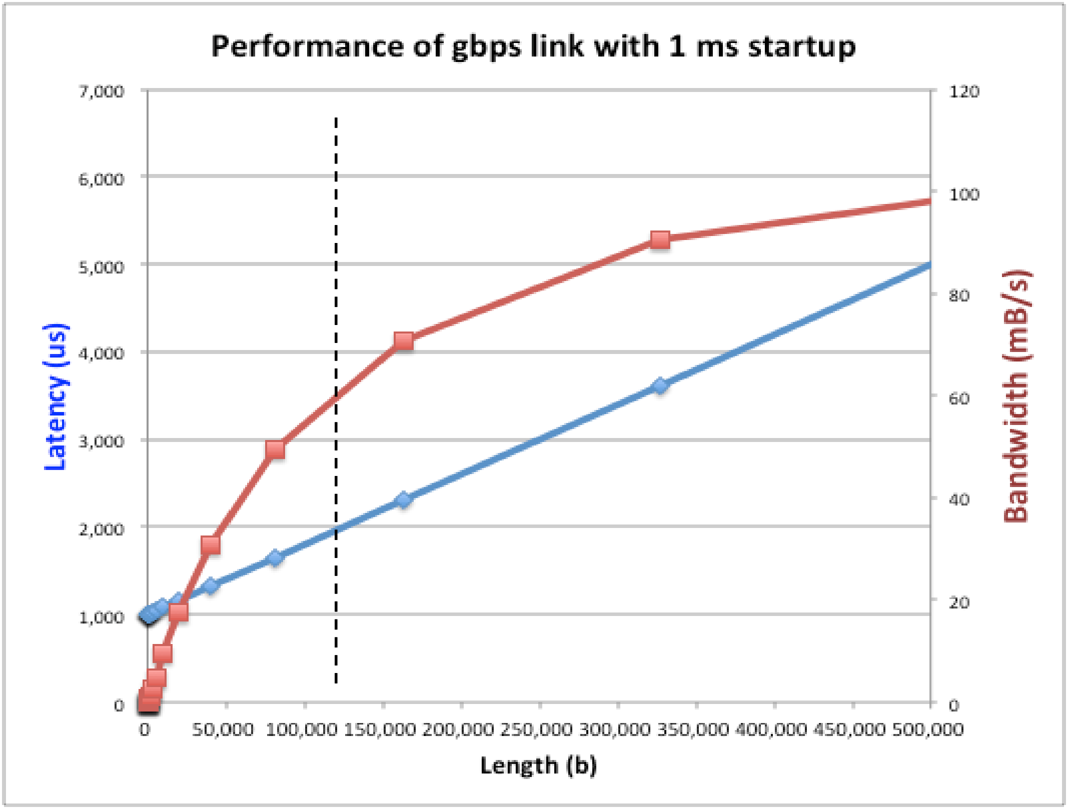
- Consider a 1 Gb/s link (Transfer capacity B = 125 MB/s)
  - With a startup cost S = 1 ms



  - Latency(n) = S + n/B
  - Bandwidth = n/(S + n/B) = B*n/(B*S + n) = B/(B*S/n + 1)

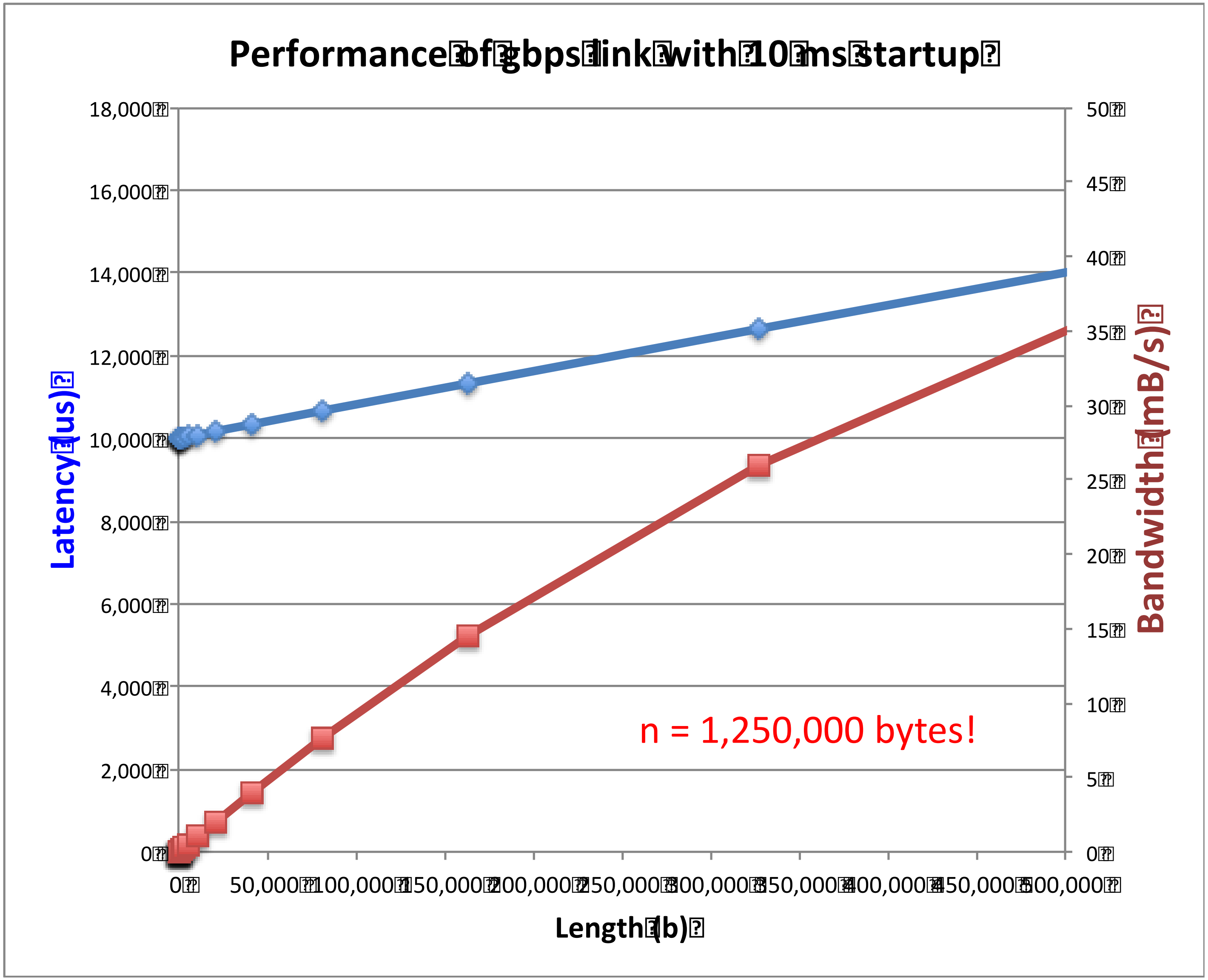
Example (Fast Network)

- Consider a 1 Gb/s link (B = 125 MB/s)
  - With a startup cost S = 1 ms



- Bandwidth = B/(B*S/n + 1)
- half-power point occurs at n=S*B → Bandwidth = B/2

Example: at 10 ms startup (like Disk)



What Determines Peak BW for I/O ?

- Bus Speed
  - PCI-X: 1064 MB/s = 133 MHz x 64 bit (per lane)
  - ULTRA WIDE SCSI: 40 MB/s
  - Serial Attached SCSI & Serial ATA & IEEE 1394 (firewire): 1.6 Gb/s full duplex (200 MB/s)
  - USB 3.0 – 5 Gb/s
  - Thunderbolt 3 – 40 Gb/s

- Device Transfer Bandwidth
  - Rotational speed of disk
  - Write / Read rate of NAND flash
  - Signaling rate of network link

Storage

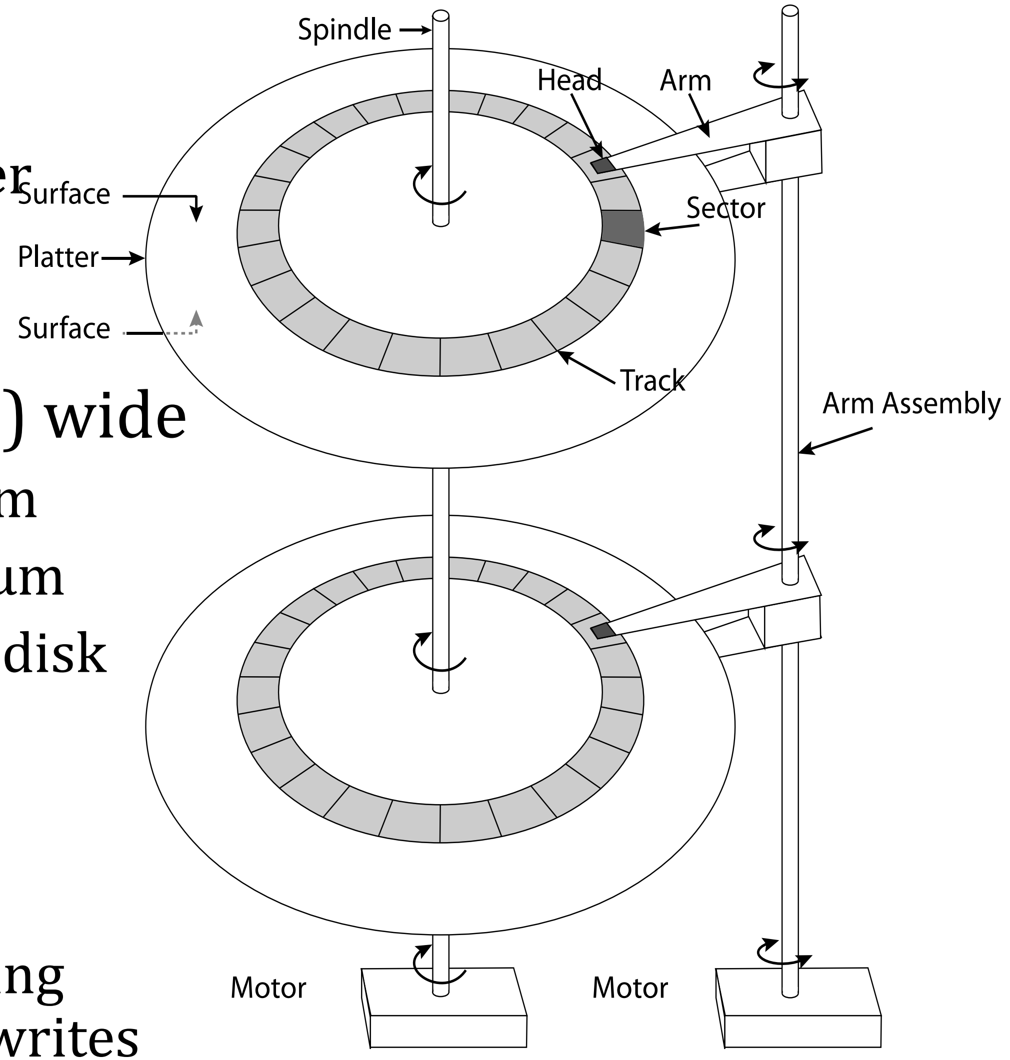
Storage Devices

- Magnetic disks
  - Storage that rarely becomes corrupted
  - Large capacity at low cost
  - Block level random access
  - Slow performance for random access
  - Better performance for sequential access

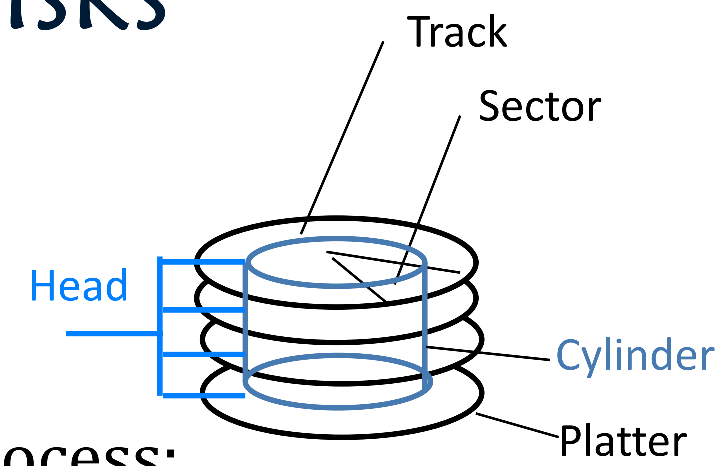
- Flash memory
  - Storage that rarely becomes corrupted
  - Capacity at intermediate cost (5-20x disk)
  - Block level random access
  - Good performance for reads; worse for random writes
  - Erasure requirement in large blocks

The Amazing Magnetic Disk

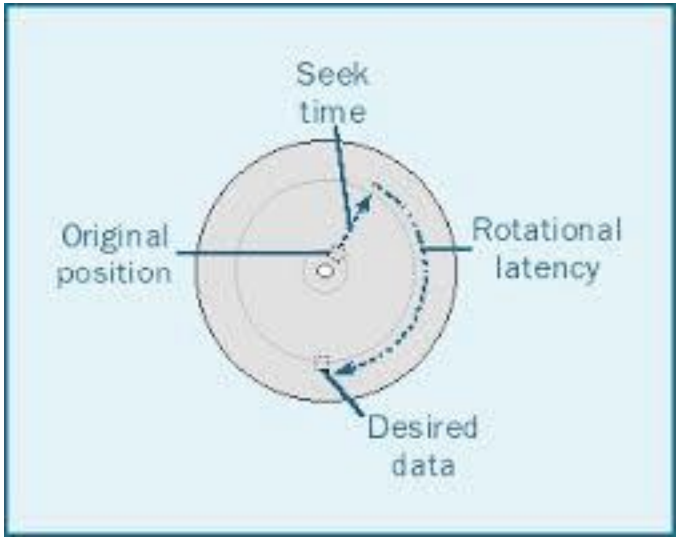
- Unit of Transfer: Sector
  - Ring of sectors form a track
  - Stack of tracks form a cylinder
  - Heads position on cylinders
- Disk Tracks ~ 1µm (micron) wide
  - Wavelength of light is ~ 0.5µm
  - Resolution of human eye: 50µm
  - 100K tracks on a typical 2.5” disk
- Separated by unused guard regions
  - Reduces likelihood neighboring tracks are corrupted during writes (still a small non-zero chance)



Magnetic Disks

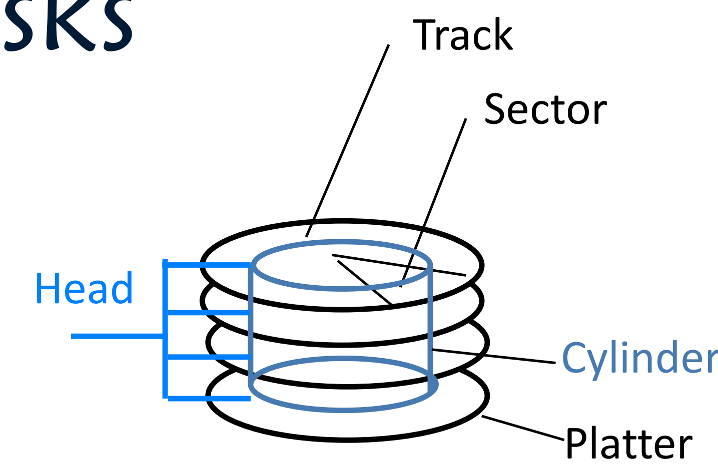


- **Cylinders:** all the tracks under the head at a given point on all surface
- Read/write data is a three-stage process:
  - **Seek time:** position the head/arm over the proper track
  - **Rotational latency:** wait for desired sector to rotate under r/w head
  - **Transfer time:** transfer a block of bits (sector) under r/w head



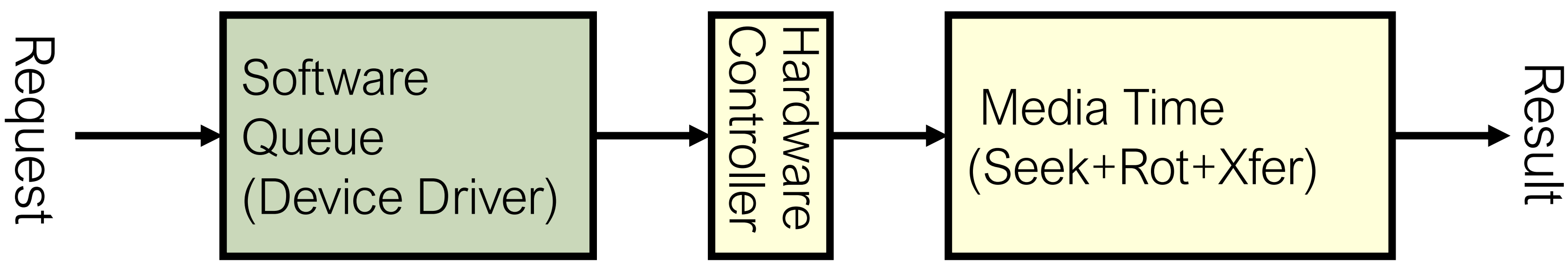
Seek time = 4-8ms
One rotation = 1-2ms
(3600-7200 RPM)

Magnetic Disks



- **Cylinders:** all the tracks under the head at a given point on all surface
- Read/write data is a three-stage process:
  - **Seek time:** position the head/arm over the proper track
  - **Rotational latency**: wait for desired sector to rotate under r/w head
  - **Transfer time**: transfer a block of bits (sector) under r/w head

Disk Latency = Queueing Time + Controller time +
Seek Time + Rotation Time + Xfer Time



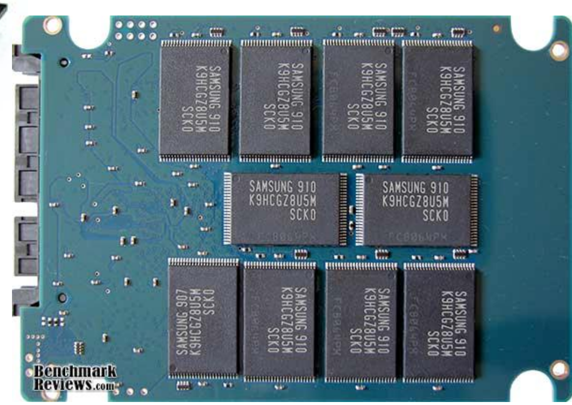
Disk Performance Example

- Assumptions:
  - Ignoring queuing and controller times for now
  - Avg seek time of 5ms,
  - 7200RPM $\Rightarrow$ Time for rotation: 60000 (ms/minute) / 7200(rev/min) ~= 8ms
  - Transfer rate of 4MByte/s, sector size of 1 Kbyte $\Rightarrow$ 1024 bytes/4×10^6 (bytes/s) = 256×10^{-6} sec $\cong$ .26 ms
- Read sector from random place on disk:
  - Seek (5ms) + Rot. Delay (4ms) + Transfer (0.26ms)
  - *Approx* 10ms to fetch/put data: 100 KByte/sec
- Read sector from random place in same cylinder:
  - Rot. Delay (4ms) + Transfer (0.26ms)
  - *Approx* 5ms to fetch/put data: 200 KByte/sec
- Read next sector on same track:
  - Transfer (0.26ms): 4 MByte/sec

Typical Numbers for Magnetic Disk

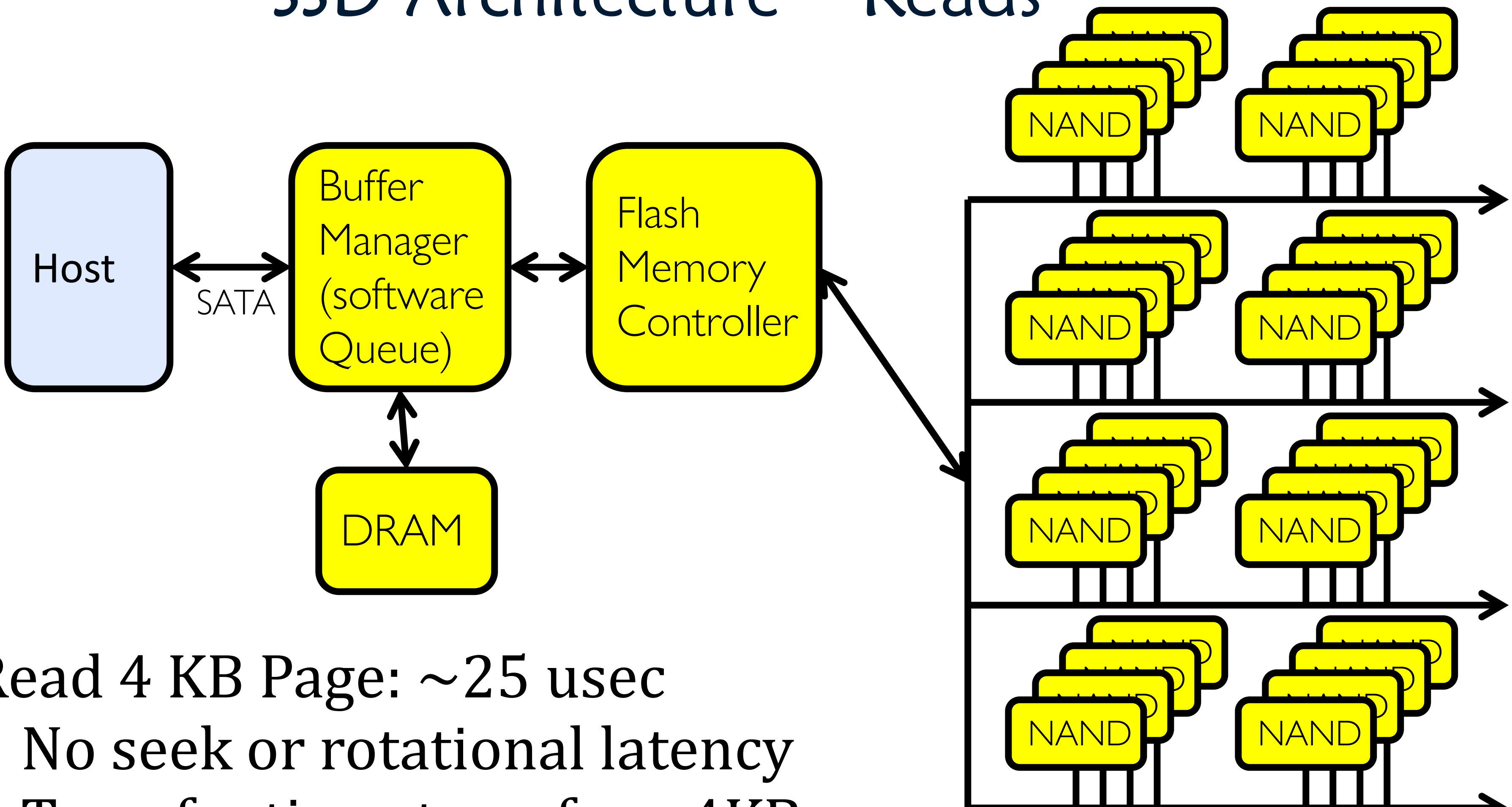
Parameter	Info / Range
Space/Density	Space: 8TB (Seagate), 10TB (Hitachi) in 3½ inch form factor! Areal Density: ≥ 1Terabit/square inch! (SMR, Helium, …)
Average seek time	Typically 5-10 milliseconds. Depending on reference locality, actual cost may be 25-33% of this number.
Average rotational latency	Most laptop/desktop disks rotate at 3600-7200 RPM (16-8 ms/rotation). Server disks up to 15,000 RPM. Average latency is halfway around disk so 8-4 milliseconds
Controller time	Depends on controller hardware
Transfer time	Typically 50 to 100 MB/s. Depends on: • Transfer size (usually a sector): 512B – 1KB per sector • Rotation speed: 3600 RPM to 15000 RPM • Recording density: bits per inch on a track • Diameter: ranges from 1 in to 5.25 in
Cost	Used to drop by a factor of two every 1.5 years (or even faster); now slowing down

Solid State Disks (SSDs)

- 1995 – Replace rotating magnetic media with non-volatile memory (battery backed DRAM)
- 2009 – Use NAND Multi-Level Cell (2 or 3-bit/cell) flash memory
  - Sector (4 KB page) addressable, but stores 4-64 "pages" per memory block
  - Trapped electrons distinguish between 1 and 0
- No moving parts (no rotate/seek motors)
  - Eliminates seek and rotational delay (0.1-0.2ms access time)
  - Very low power and lightweight
  - Limited "write cycles"
- Rapid advances in capacity and cost ever since!

SSD Architecture – Reads

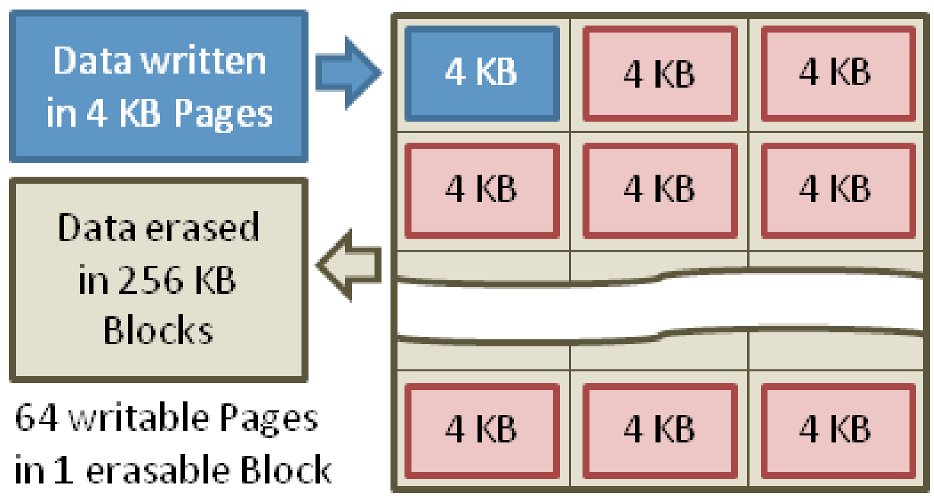


Read 4 KB Page: ~25 usec

- No seek or rotational latency
- Transfer time: transfer a 4KB page
  - SATA: 300-600MB/s => ~4 x10^3 b / 400 x 10^6 bps => 10 us
- Latency = Queuing Time + Controller time + Xfer Time
- Highest Bandwidth: Sequential OR Random reads

SSD Architecture – Writes

- Writing data is complex! (~200μs – 1.7ms)
  - Can only write empty pages in a block
  - Erasing a block takes ~1.5ms
  - Controller maintains pool of empty blocks by coalescing used pages (read, erase, write), also reserves some % of capacity
- Rule of thumb: writes 10x reads, erasure 10x writes



Typical NAND Flash Pages and Blocks

https://en.wikipedia.org/wiki/Solid-state_drive

Amusing calculation: is a full Kindle heavier than an empty one?

- Actually, "Yes", but not by much
- Flash works by trapping electrons:
  - So, erased state lower energy than written state
- Assuming that:
  - Kindle has 4GB flash
  - ½ of all bits in full Kindle are in high-energy state
  - High-energy state about 10^{-15} joules higher
  - Then: Full Kindle is 1 attogram (10^{-18}gram) heavier
    (Using $E = mc^2$)
- Of course, this is less than most sensitive scale can measure
  (it can measure 10^{-9} grams)
- Of course, this weight difference overwhelmed by battery discharge, weight from getting warm, ...
- According to John Kubiatowicz (New York Times, Oct 24, 2011)

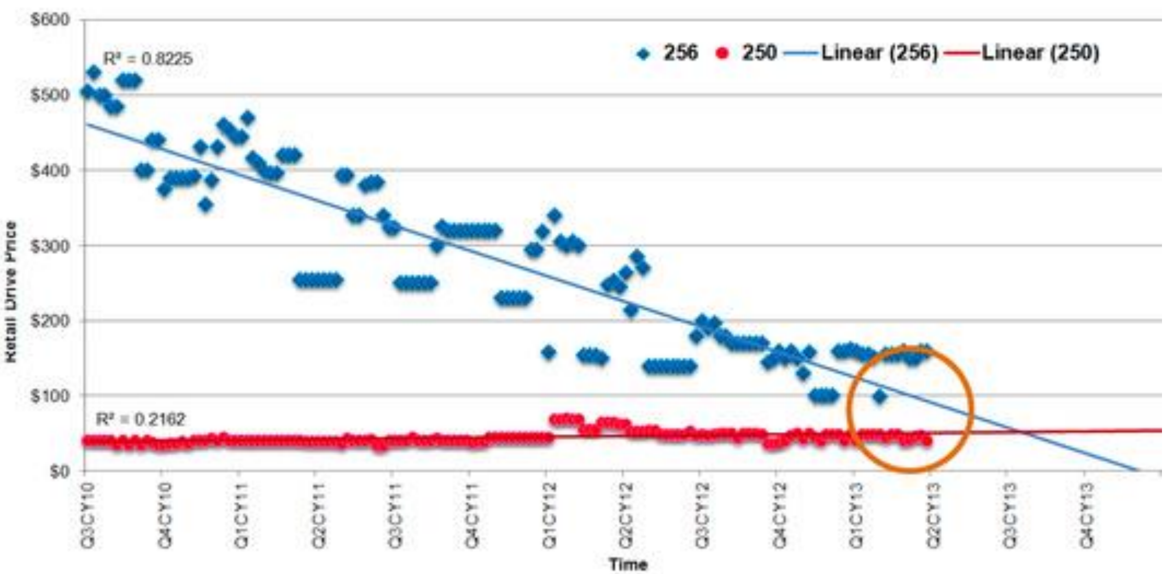
SSD Summary

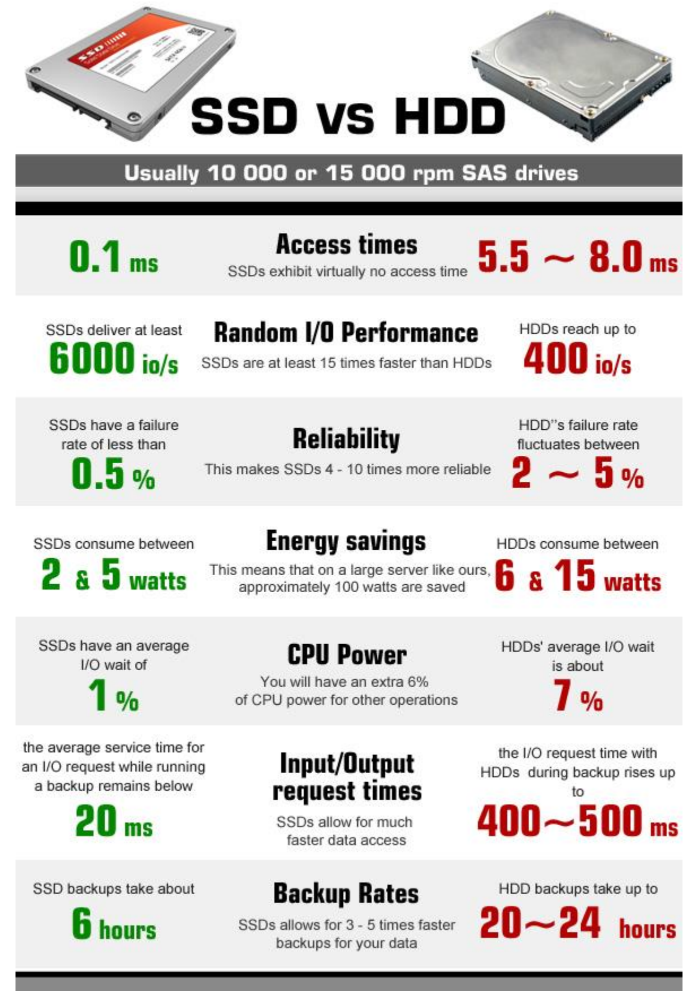
- Pros (vs. hard disk drives):
  - Low latency, high throughput (eliminate seek/rotational delay)
  - No moving parts:
    - Very light weight, low power, silent, very shock insensitive
  - Read at memory speeds (limited by controller and I/O b

No longer true!

- Cons
  - ~~Small storage (0.1-0.5x disk)~~, expensive (3-20x disk)
    - Hybrid alternative: combine small SSD with large HDD
  - Asymmetric block write performance: read pg/erase/write pg
    - Controller garbage collection (GC) algorithms have major effect on performance
  - Limited drive lifetime
    - 1-10K writes/page for MLC NAND
    - Avg failure rate is 6 years, life expectancy is 9–11 years
- These are changing rapidly!

HDD vs SSD Comparison





Price Crossover Point for HDD and SSD

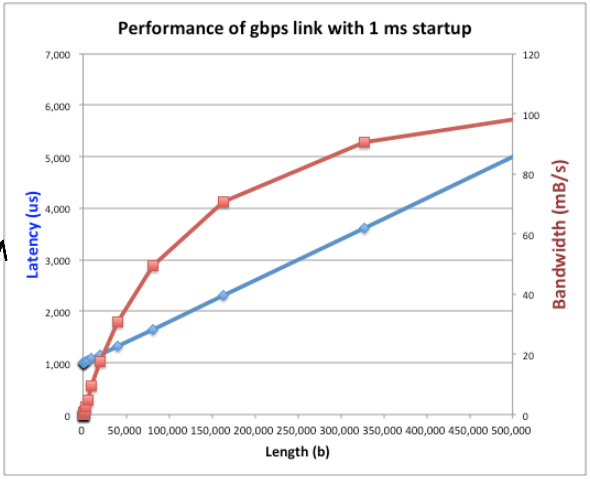
	2012	2013	2014	2015E	2016F	2017F
HDD	0.09	0.08	0.07	0.06	0.06	0.06
2.5" SSD	0.99	0.68	0.55	0.39	0.24	0.17

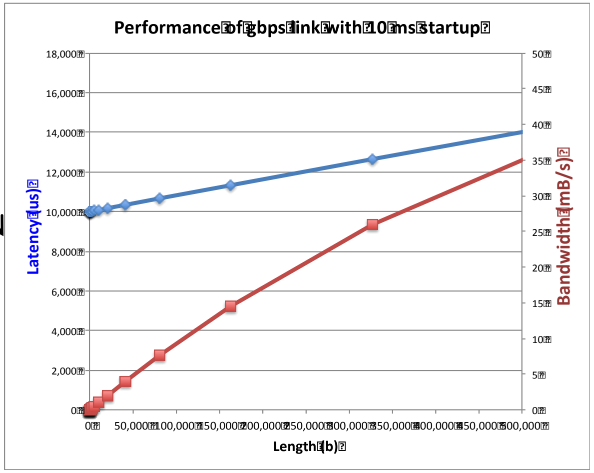
SSD prices drop much faster than HDD

What Goes into Startup Cost for I/O?

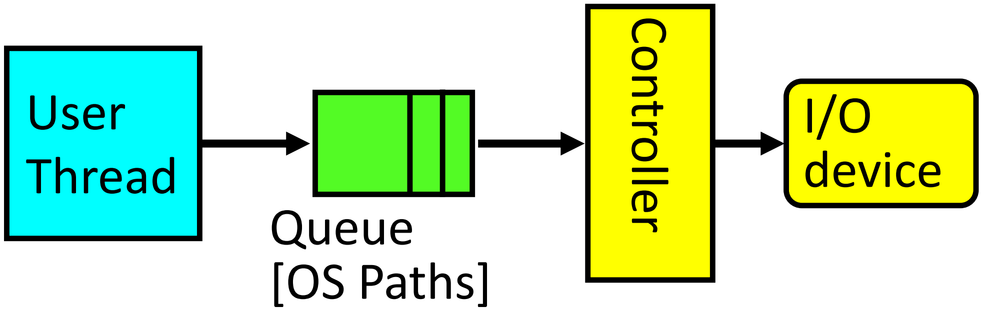
- Syscall overhead
- Operating system processing
- Controller Overhead
- Device Startup
  - Mechanical latency for a disk
  - Media Access + Speed of light + Routing for network
- Queuing (next topic)

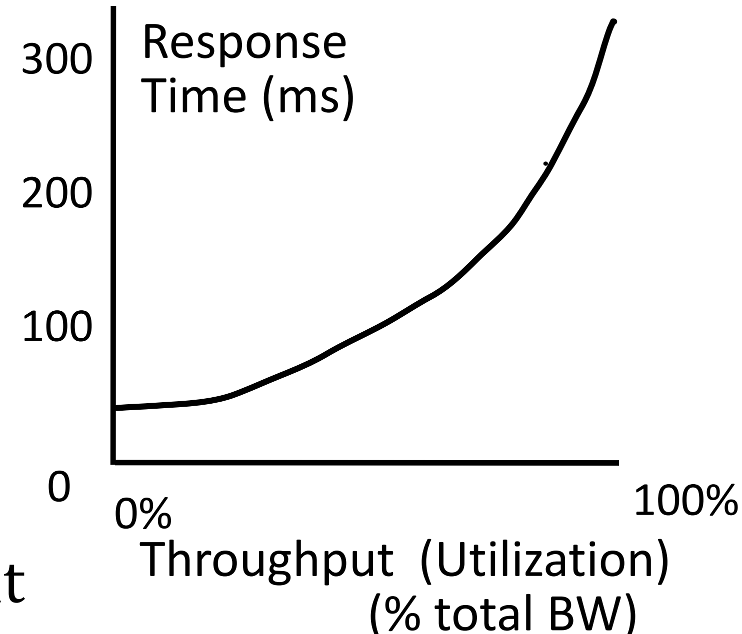
Startup cost
(fixed overhead)





I/O Performance





Response Time = Queue + I/O device service time

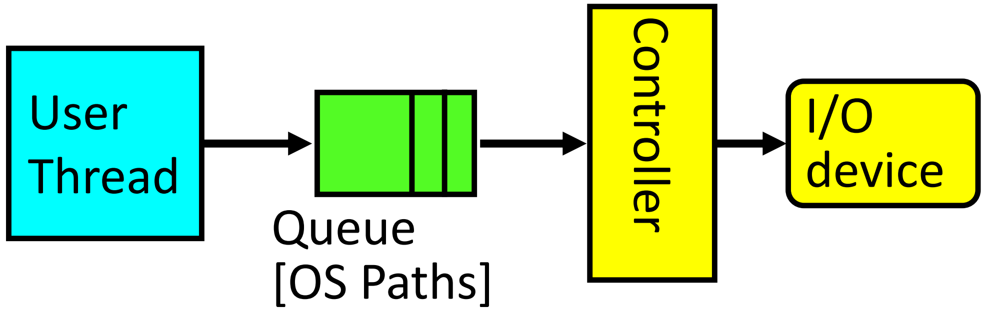
- Performance of I/O subsystem
  - Metrics: Response Time, Throughput
  - Effective BW per op = transfer size / response time
    - $EffBW(n) = n / (S + n/B) = B / (1 + SB/n)$

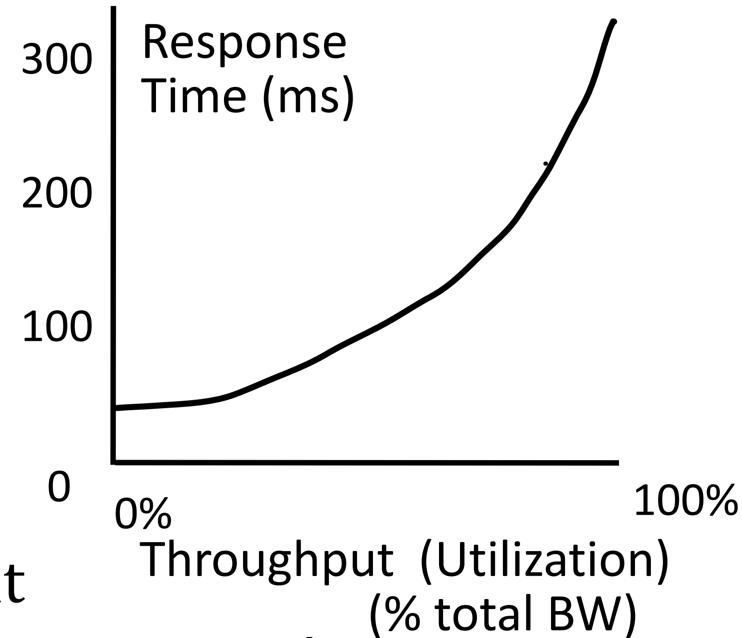
\# of ops

Fixed overhead

time per op

I/O Performance





Response Time = Queue + I/O device service time

- Performance of I/O subsystem
  - Metrics: Response Time, Throughput
  - Effective BW per op = transfer size / response time
    - EffBW(n) = n / (S + n/B) = B / (1 + SB/n)
  - Contributing factors to latency:
    - Software paths (can be loosely modeled by a queue)
    - Hardware controller
    - I/O device service time
- Queuing behavior:
  - Can lead to big increases of latency as utilization increases
  - FYI. we skip them in the main course, slides are attached

When is Disk Performance Highest?

- When there are big sequential reads, or
- When there is so much work to do that they can be piggy backed (reordering queues—one moment)

- OK to be inefficient when things are mostly idle
- Bursts are both a threat and an opportunity
- <your idea for optimization goes here>
  - Waste space for speed?

- Other techniques:
  - Reduce overhead through user level drivers
  - Reduce the impact of I/O delays by doing other useful work in the meantime

Performance: Disk Scheduling

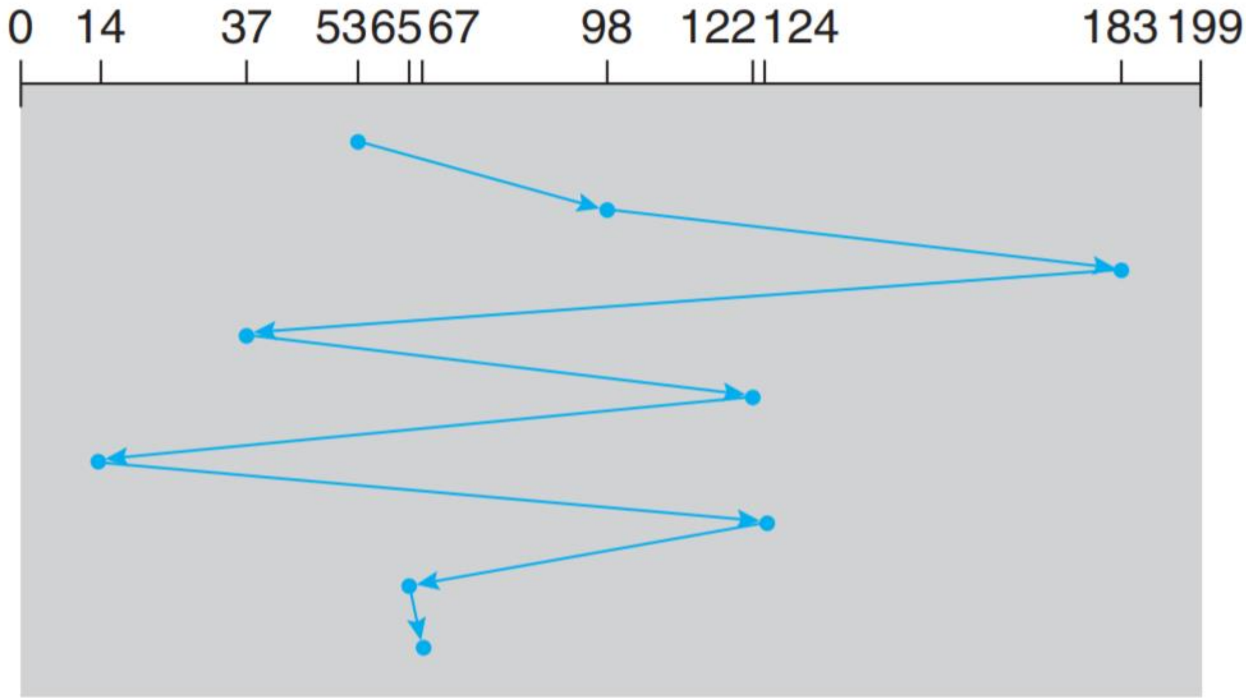
Disk Scheduling

- There are many sources of disk I/O request
  - OS
  - System processes
  - User processes
- OS should think how to use hardware efficiently?
  - Disk bandwidth
  - Access time
- Given a sequence of access pages in the HDD
  - 98, 183, 37, 122, 14, 124, 65, 67
  - Head point: 53
  - Pages: 0 ~ 199
- Minimize seek time
  - Seek time $\approx$ seek distance
- How to minimize the total head movement distance?
  - Minimize the total number of cylinders.

Disk Scheduling: FIFO

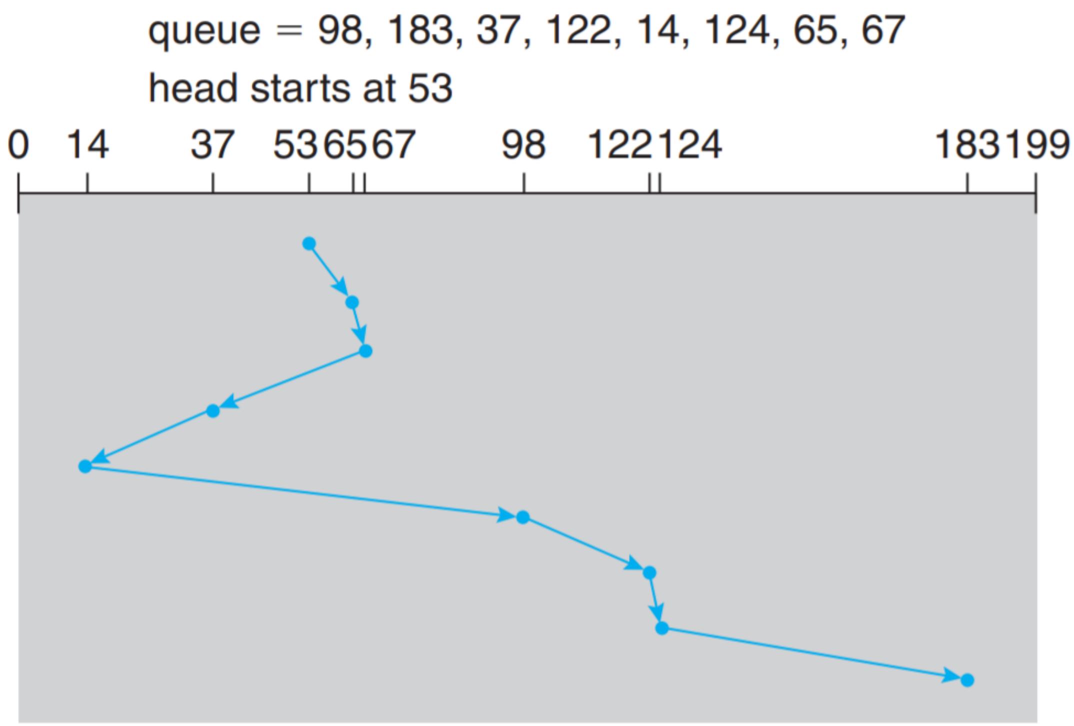
- FIFO Order
  - Fair among requesters, but order of arrival may be to random spots on the disk ⇒ Very long seeks
- The head movement distance = ?

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



Disk Scheduling: SSTF

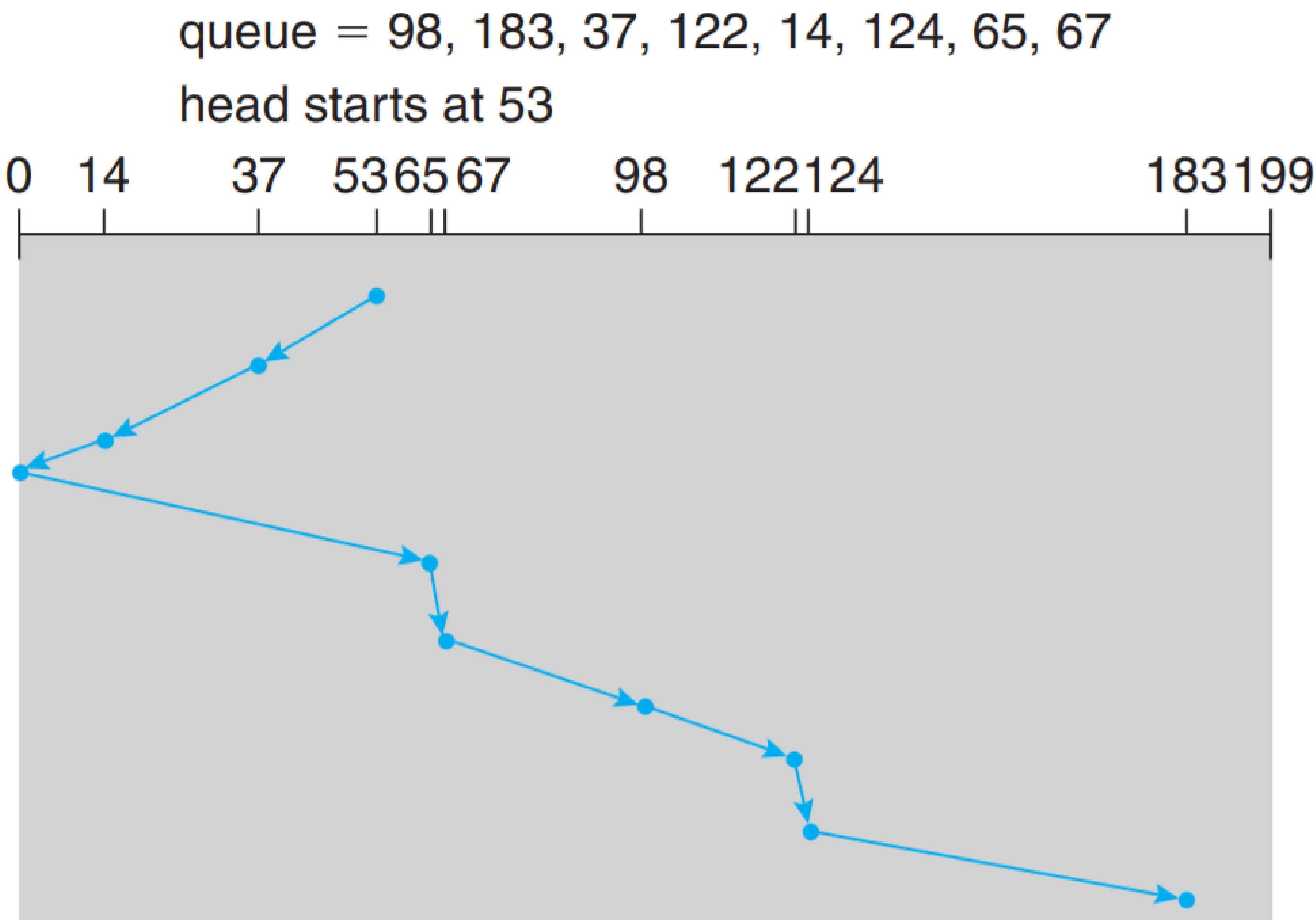
- SSTF order
  - Shortest Seek Time First selects the request with the minimum seek time from the current head position
  - SSTF scheduling is a form of **SJF** scheduling; may cause **starvation** of some requests
- The head movement distance = ?



Disk Scheduling: SCAN

- SCAN order
  - SCAN algorithm a.k.a., elevator algorithm
  - The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
  - But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest
- The head movement distance = ?

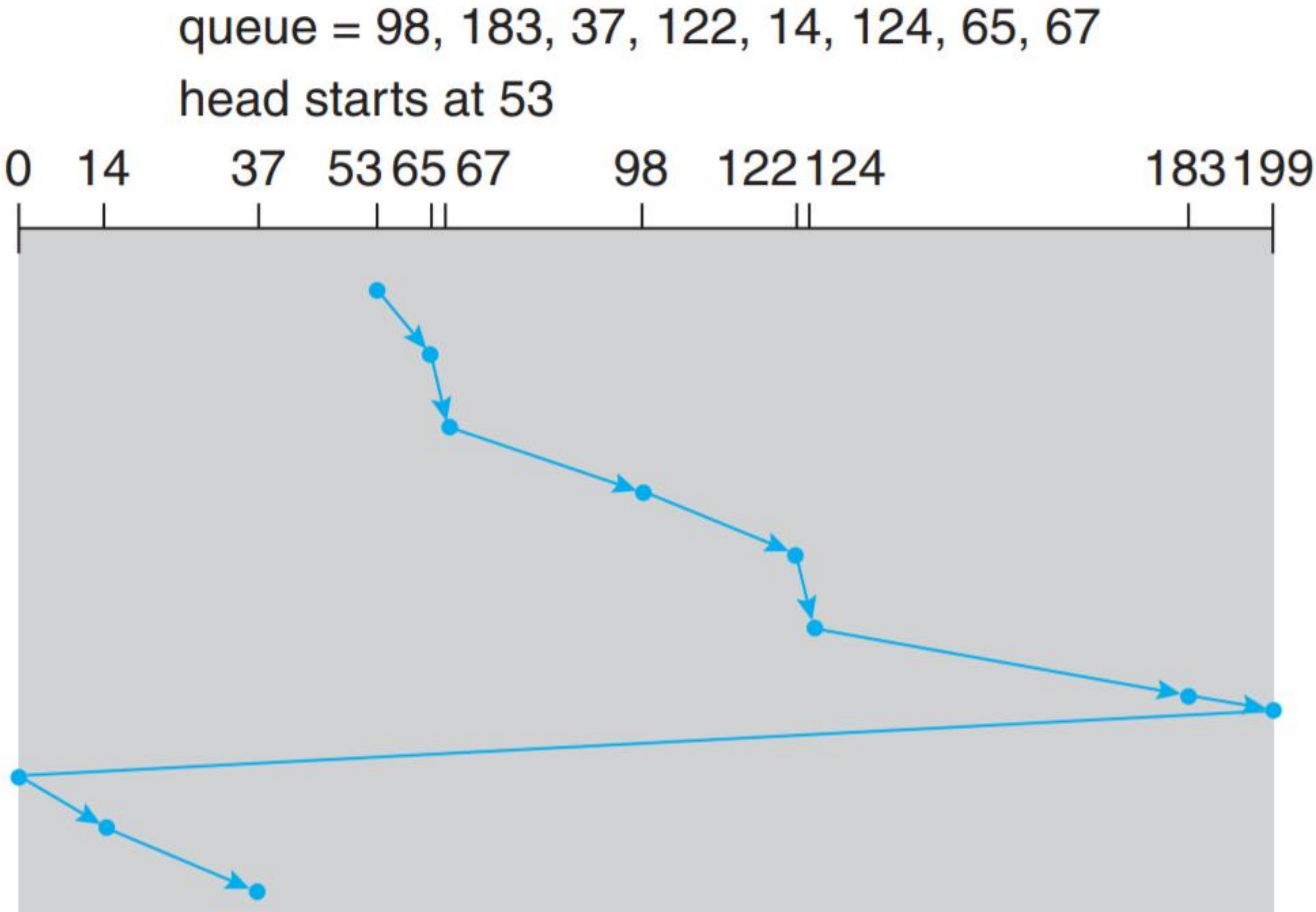
Disk Scheduling: SCAN



Disk Scheduling: C-SCAN

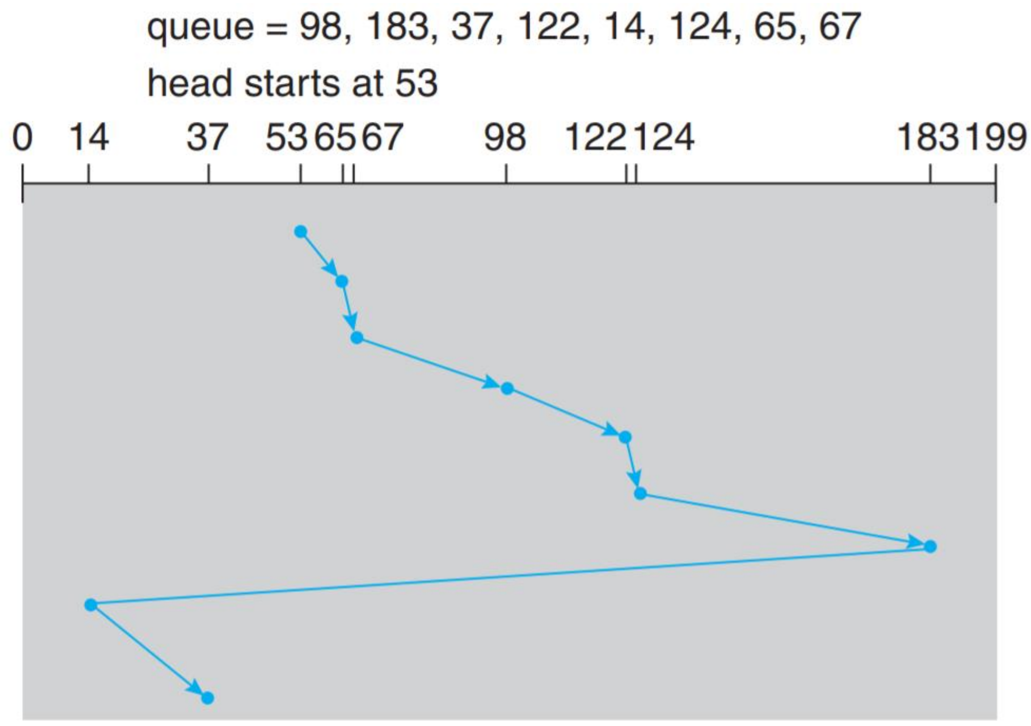
- C-SCAN order
  - Provides a more uniform wait time than SCAN
  - The head moves from one end of the disk to the other, servicing requests as it goes
    - When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
  - Treats the cylinders as a circular list that wraps around from the last cylinder to the first one
- The head movement distance =?

Disk Scheduling: C-SCAN



Disk Scheduling: LOOK, C-LOOK

- Look and C-LOOK order
  - LOOK a version of SCAN, C-LOOK a version of C-SCAN
  - Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk
- C-LOOK: the head movement distance = ?



Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
  - Less starvation
- Either SSTF or LOOK is a reasonable choice for the default algorithm

- Performance depends on the number and types of requests
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary

Thank You!