CIS 5512 - Operating Systems
I/O Systems & Secondary Storage

Professor Qiang Zeng

Fall 2017

Previous class…

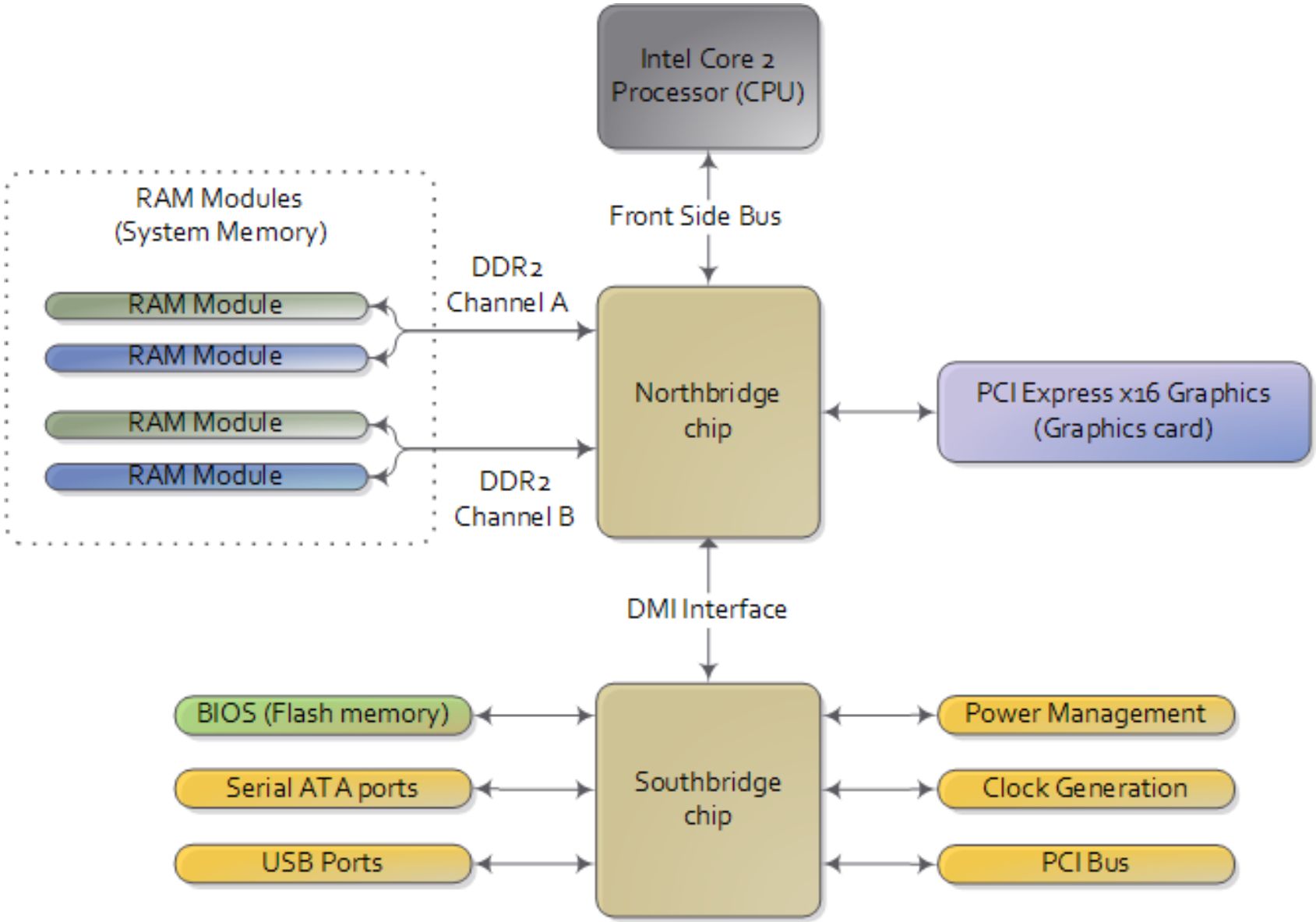
- Memory subsystem
  - How to allocate physical memory?
  - How to do address translation?
  - How to be quick?
    - CPU cache such as TLB
    - Page cache
  - How to be lazy?
    - Copy-on-write
    - Demand paging
    - Memory-mapped file I/O

Outline

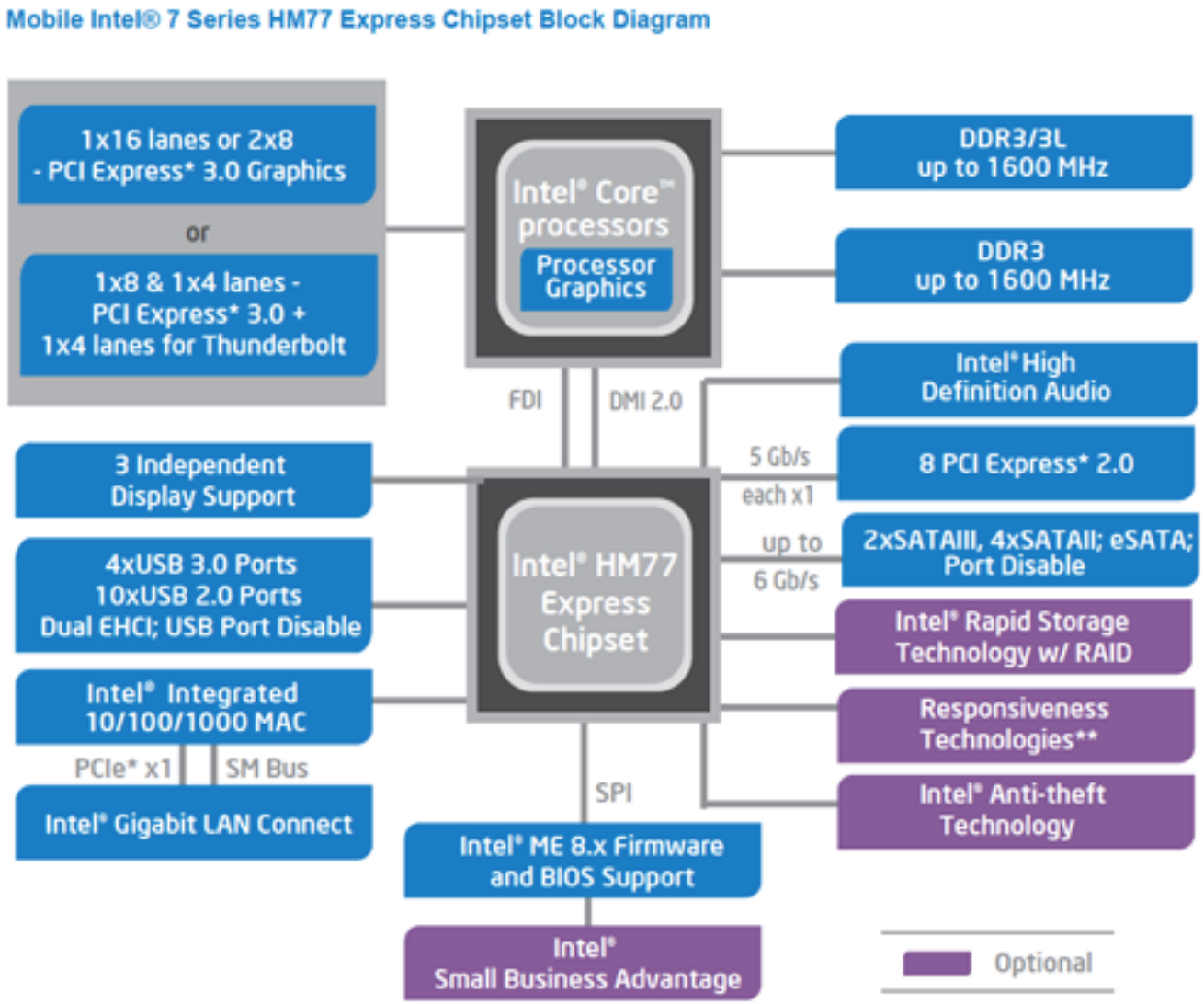
- I/O subsystem: hardware aspect
  - Terms: controller, bus, port
  - How are the device registers and memory addressed?
- I/O subsystem: software aspect
  - Device drivers programmer's perspective: polling, interrupt-driven, and DMA
  - User-space programmer's perspective: blocking, non-blocking, & asynchronous
- I/O subsystem: performance aspect
  - Caching, Buffering and Spooling
  - I/O scheduling

Some slides are courtesy of Dr. Abraham Silberschatz and Dr. Thomas Anderson

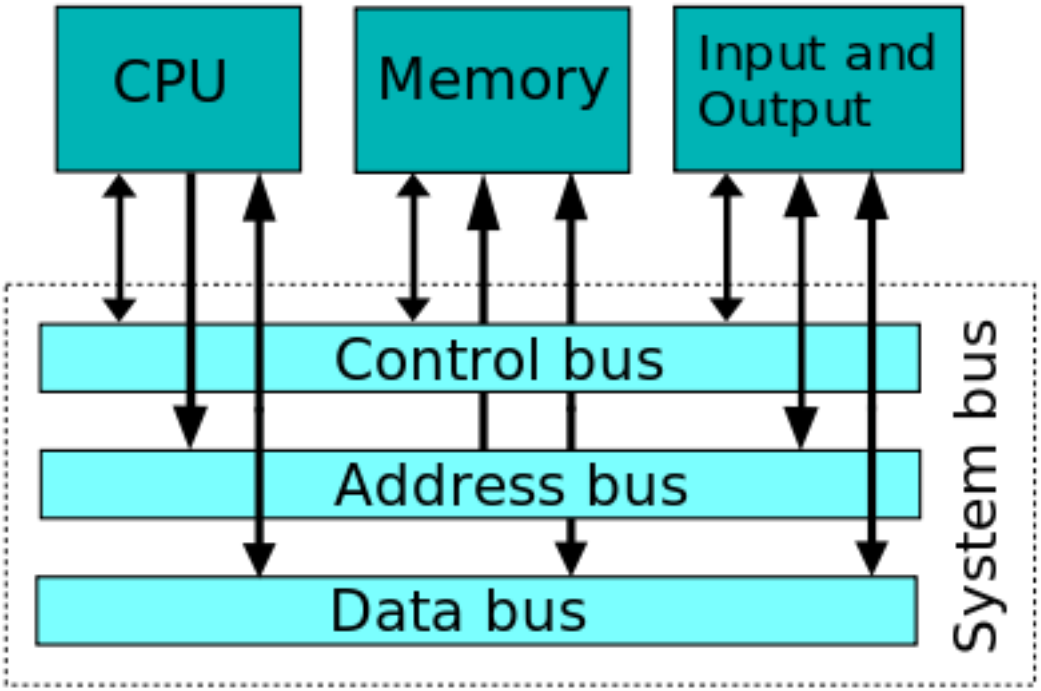
Northbridge/Southbridge architecture



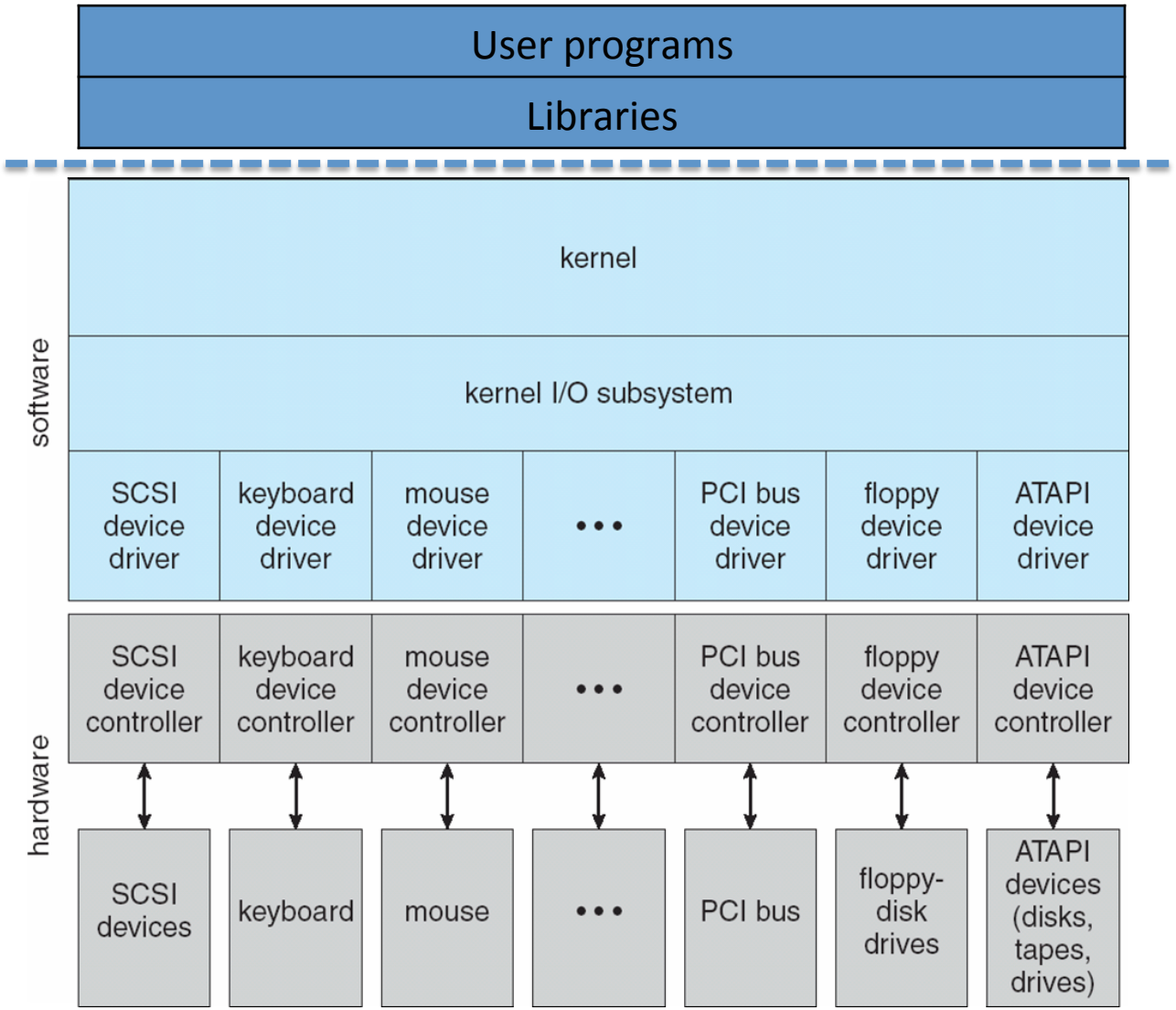
Intel Ivy Bridge



Simplified architecture



Layers:
User program-> libs -> core kernel -> Kernel I/O subsystem -> driver->controller->drive

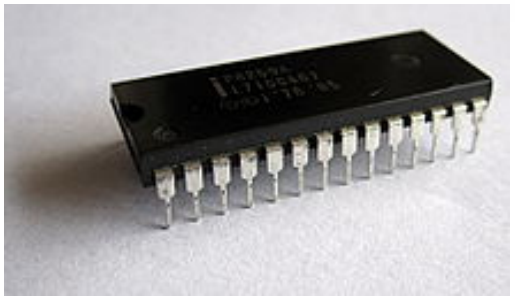


Controller

- A hardware module, inside or outside computer, that controls (or, interfaces with) a peripheral component or device.
  - Inside: e.g., a memory controller that manages access to memory, a Network Interface Controller that connects a computer to a network
  - Outside: e.g., the controller in an external hard disk
- Controller contains *registers* and *memory* that bridge the device and the computer
  - E.g., a keyboard controller's memory contains the chars that the user keyed in

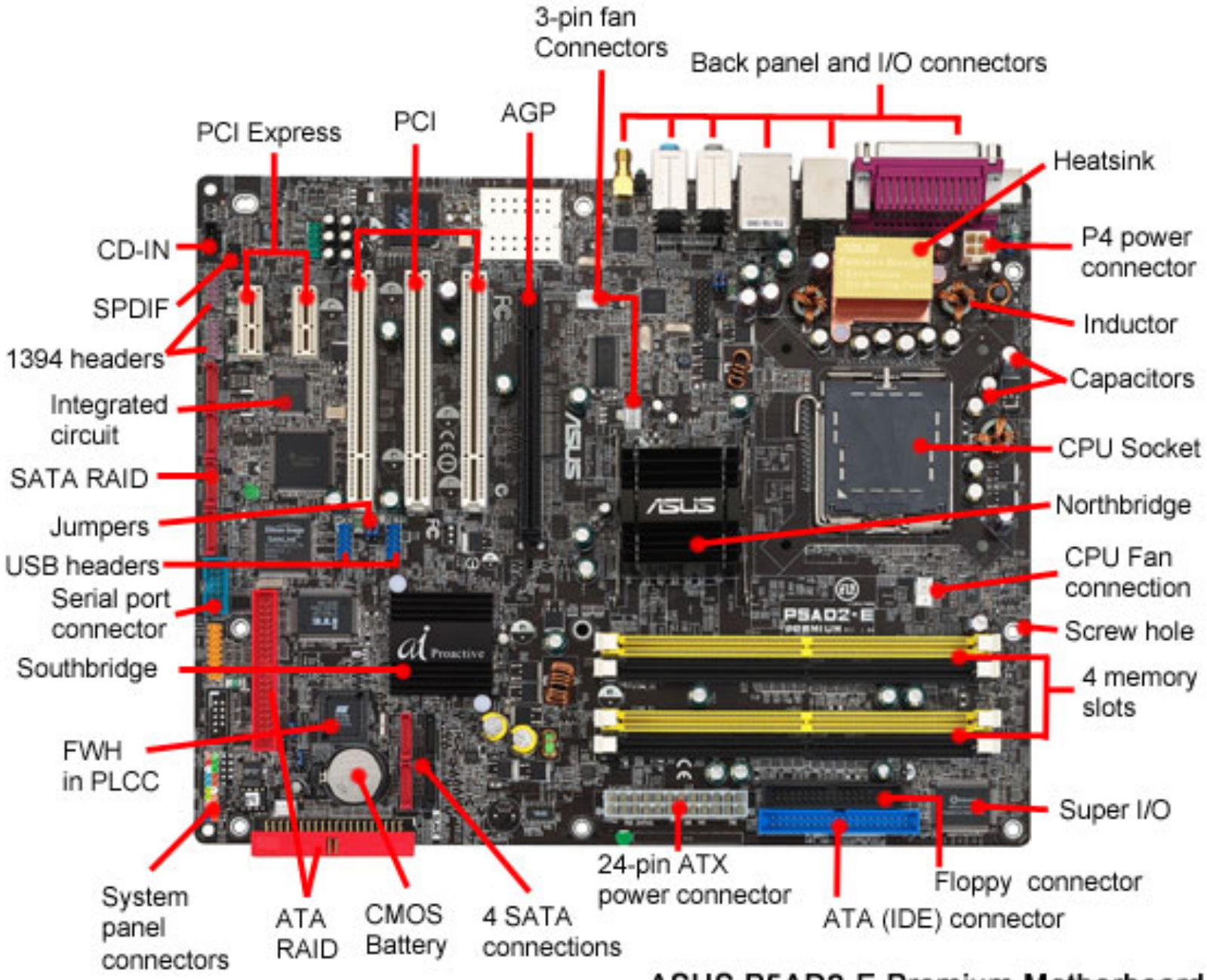
Controller: two forms

- Controller boards
  - Network interface controller
  - Graphics controller
- Chips
  - Northbridge: contains mem controller
  - Southbridge: hub of I/O controllers
  - Keyboard controller
  - Interrupt Controller



Bus and port

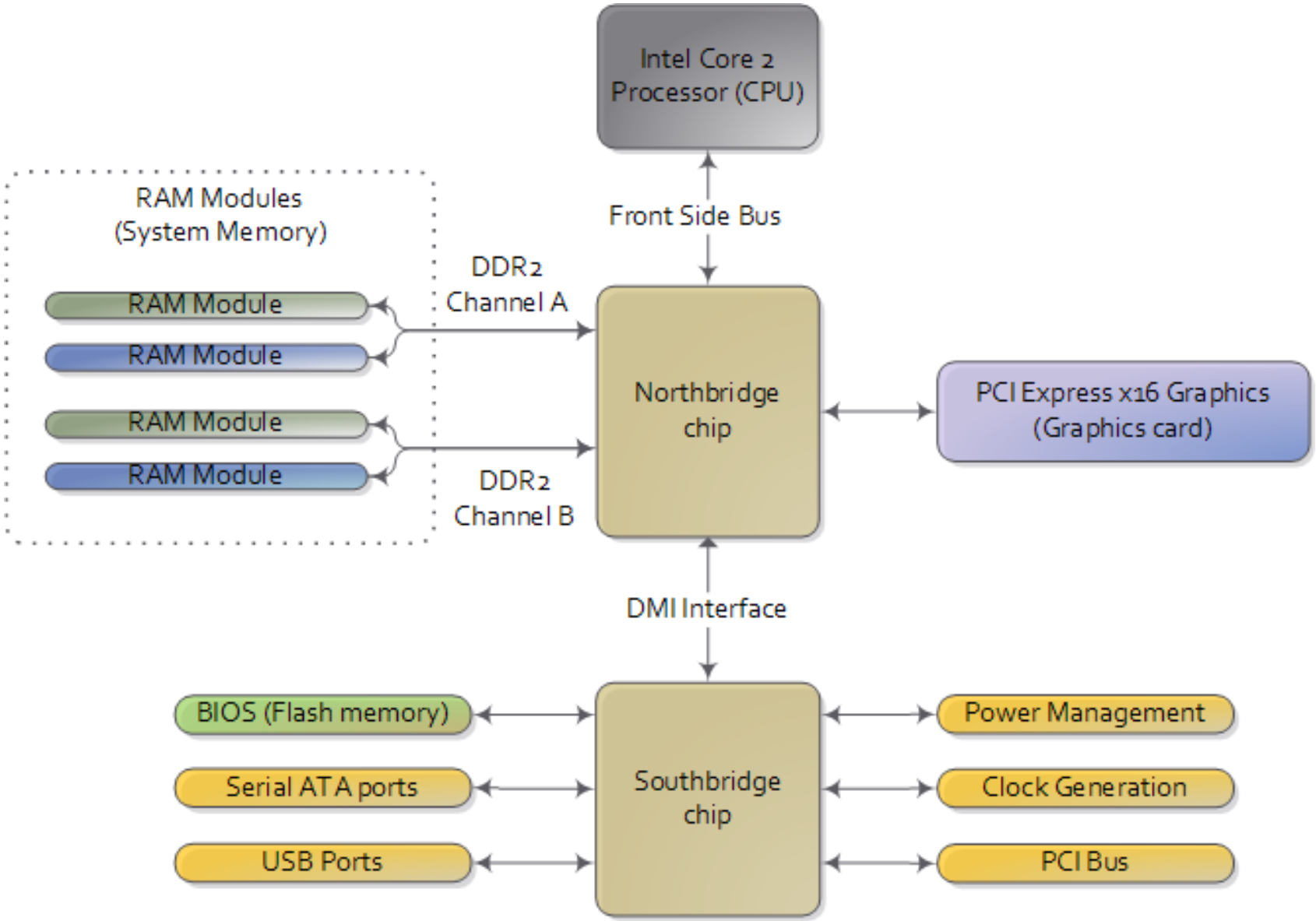
- Bus: logically, a set of wires used to pass information between computer components. E.g.,
  - **PCI** (Peripheral Component Interconnect)
  - **PCIe** (PCI express)
- Port: an interface to connect a computer with other computers or peripheral devices
  - Many controllers have ports to connect I/O devices
  - E.g., VGA, DVI, HDMI, Thunderbolt, PS/2, USB ports
  - Not to be confused with TCP/UDP ports
  - Not to be confused with I/O ports (addresses)



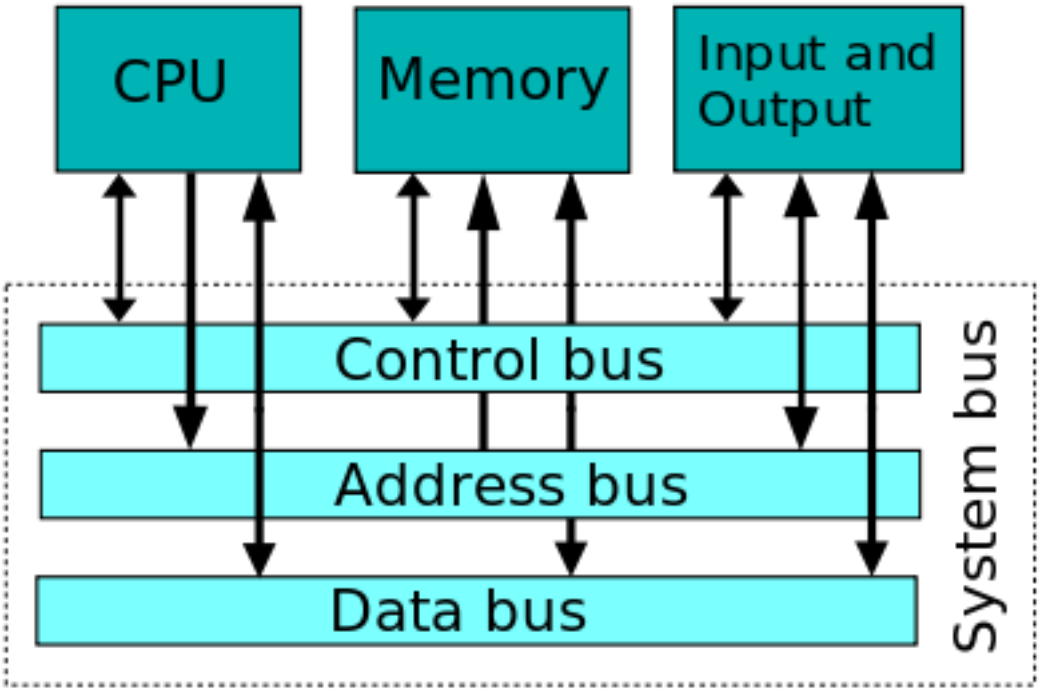
ASUS P5AD2-E Premium Motherboard

http://www.computerhope.com

Northbridge/Southbridge architecture



Simplified architecture



Controller registers

- Control register
  - can be written to start a command or to change the mode of a device
- Status register
  - Whether the current command has completed
  - Whether it has data to read (e.g., network packets)
  - Whether there is an error
- Data-in register
  - Read to get input
- Data-out register
  - Written to send output

How are the controller registers and memory addressed

- Port-mapped I/O
- Memory-mapped I/O

Port-mapped I/O

- Special instructions and a dedicated address space
  - Disadvantage 1: it increases the complexity of CPU for the support of special instructions
  - Disadvantage 2: those special instructions are very limited; e.g., in X86 they are *in* and *out*
  - Advantage: it doesn't occupy the physical address space
  - Try "cat /proc/ioports"

```
qiang@ubuntu:~/tmp/racecondition$ cat /proc/ioports | head -10
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-0060 : keyboard
0064-0064 : keyboard
0070-0071 : rtc0
0080-008f : dma page reg
00a0-00a1 : pic2
```

Port-mapped I/O example

```
MOV DX,12345h
IN AL,DX          ;reads I/O port 12345h
```

Memory-mapped I/O

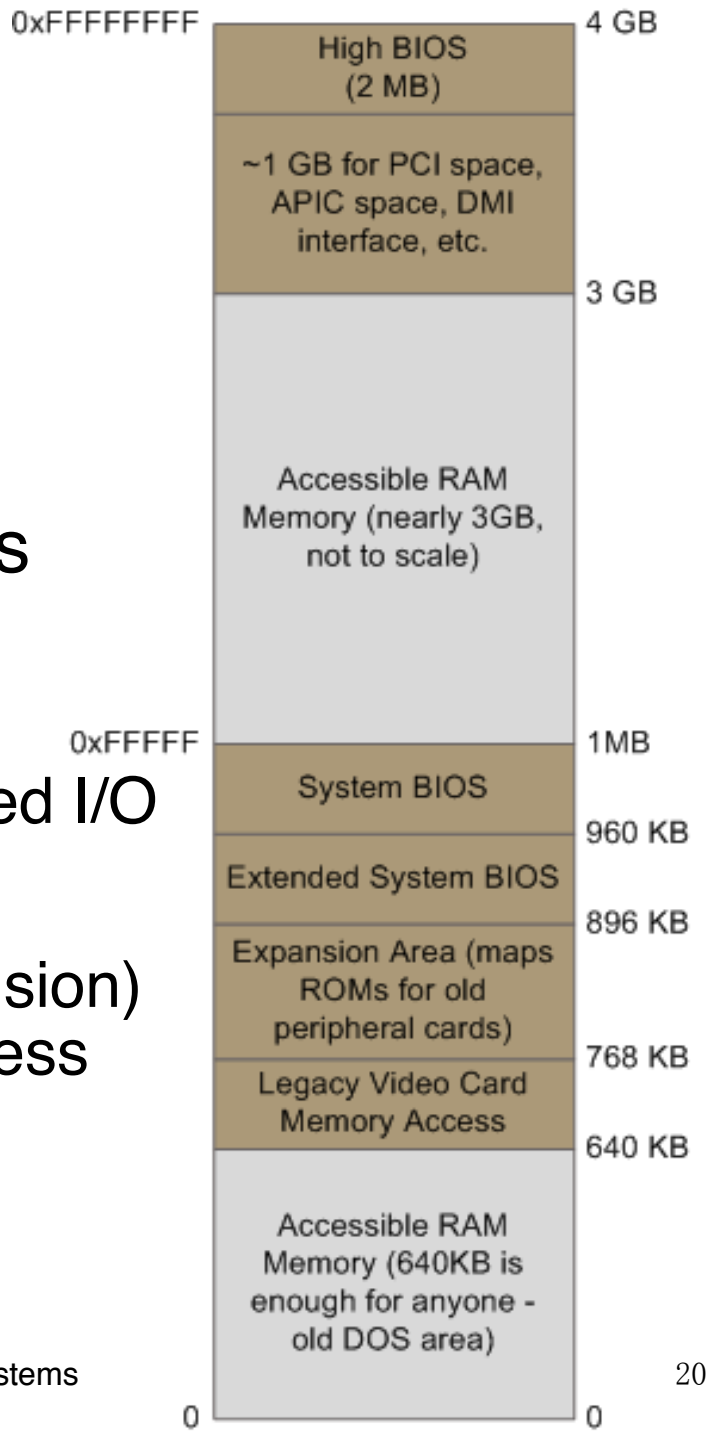
- Not to be confused with memory-mapped file, which maps a disk file to the virtual address space of a process
- Controller register and memory are mapped into the same physical address space as RAM, and are accessed the same way as RAM
  - Advantage: CPU is simpler; all instructions, e.g., *MOV* and *ADD* and addressing modes used for accessing RAM can be used for I/O
  - Disadvantage: controller's memory occupies portion of the physical address space

Port-mapped vs. memory-mapped I/O

```
MOV DX,12345h
IN AL,DX          ;reads I/O port 12345h
MOV AL,[DX]       ;virtual address 12345h is
first translated to a physical address,
which may point to controller memory,
BIOS, RAM or nothing
```

Questions

- On 32-bit system, why the system reports around 3GB memory even if 4GB RAM is installed?
  - Called “3G Barrier” issue
  - Mainly due to memory-mapped I/O
- What is the Solution?
  - PAE (Physical Address Extension) supports 36-bit physical address space
  - 64-bit systems



cat /proc/iomem

```
qiang@ubuntu:~/tmp/racecondition$ cat /proc/iomem
00000000-0000ffff : reserved
00010000-0009ebff : System RAM
0009ec00-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000ca000-000cafff : Adapter ROM
000dc000-000fffff : reserved
  000f0000-000fffff : System ROM
00100000-bbedffff : System RAM
  00100000-005983e6 : Kernel code
  005983e7-007acdc7 : Kernel data
  00858000-008e7ef7 : Kernel bss
```

Outline

- I/O subsystem: hardware aspect
  - Terms: controller, bus, port
  - How is the device memory addressed?
- I/O subsystem: software aspect
  - Device drivers programmer's perspective: polling, interrupt-driven, and DMA
  - User-space programmer's perspective: blocking, non-blocking, & asynchronous
- I/O subsystem: performance aspect
  - Caching, Buffering and Spooling
  - I/O scheduling

Some slides are courtesy of Dr. Abraham Silberschatz and Dr. Thomas Anderson

How to deal with various devices?

- Take the secondary storage as an example, it may be
  - A Western Digital RAID
  - A Seagate hard disk drive
  - An Intel solid state drive
  - An Amazon Elastic Block Store volume
- Building an operating system that handles each case individually would be too complex
- What should we do?
  - Layering

Device drivers

- What if all the second storage systems expose the same set of interface
  - Then the other part of the kernel doesn't need to care which storage drive is being used
  - That is exactly what device drivers do
- A device driver is a piece of code that operates a particular type of device, and exports some standard interface to the kernel (or other programs)
  - E.g., each manufacture (Intel, WD, and Seagate) provides a device driver that exports the same standard *block device* interface, so that the kernel can access the various devices based on the uniform interface

Device driver types

- Block devices
  - Provide buffered access (i.e., data is conveyed through buffer)
  - Usually seekable (i.e., supports random access)
- Character devices
  - Provide unbuffered access (i.e., raw access)
  - Usually not seekable (i.e., data is like a stream)
- Network devices
  - Sending and receiving data packets

Interfaces

```
$ ls -al /dev
crw-rw-rw-   1 root   wheel        4,  64 Oct 27 22:23 ttyt0
crw-rw-rw-   1 root   wheel       18,   3 Oct 27 22:23 Modem
brw-r-----   1 root   operator     1,   0 Oct 27 22:23 disk0
```

The network interface doesn't fit the Unix "*everything is a file*" principle, and does not exist in the /dev directory

Driver-device communication

- Polling
- Interrupt-driven I/O
- DMA (Direct Memory Access)

Polling

- For each byte of I/O
  1. **Polling** the busy bit in the status register until 0
  2. CPU sets read or write bit and, if write, copies data into data-out register
  3. CPU sets command-ready bit
  4. Controller sets busy bit, executes transfer
  5. Controller clears busy bit, error bit, and command-ready bit when transfer done
- Step 1 is **busy-wait** cycle to wait for I/O from device
  - It wastes a lot of CPU cycles
  - Recall the busy-wait lock in concurrent programming
  - Can someone notify CPU when the busy bit = 0?

Interrupt-driven I/O

- CPU **Interrupt-request line** is set by I/O device
  - Checked by processor after each instruction
- Dispatch handling to correct handler according to the interrupt number
  - Recall the Interrupt Descript Table
  - Interrupt chaining if more than one device uses the same interrupt number
- An interrupt is generated by devices when
  - Input is ready
  - Output is done

Direct Memory Access

- Contrast to **programmed I/O** (polling and interrupt-driven I/O), which relies on CPU to issue instructions to move each byte/word
- CPU initiates DMA command; once initiated, CPU is free to do other work and the DMA controller will take care of the I/O
- The initiation is costly, so DMA is not suitable for moving a small amount of data, for example
  - Keyboard and mouse
- DMA is now frequently used for
  - Network, graphics, disk I/O

Three steps of DMA

- CPU tells the DMA controller the following info.
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
- DMA controller arranges data transfer between device and memory according the info.
- When done, DMA controller issues an interrupt to signal completion

Addr. translation is important for security

- Each TLB and page table entry has a Supervisor bit
  - Entries for kernel memory have the bit set
  - If a user space program (ring 3) requests to translate a virtual address using these entries, a permission violation Page Fault will be triggered
  - This is how kernel (ring 0) memory gets protected
- Each process uses its own page table to translate addresses, Process A cannot access the content of Process B unless they share
- Finally, when a process writes to a read-only page, e.g., for libc code, the address translation again will trigger a permission violation Page Fault

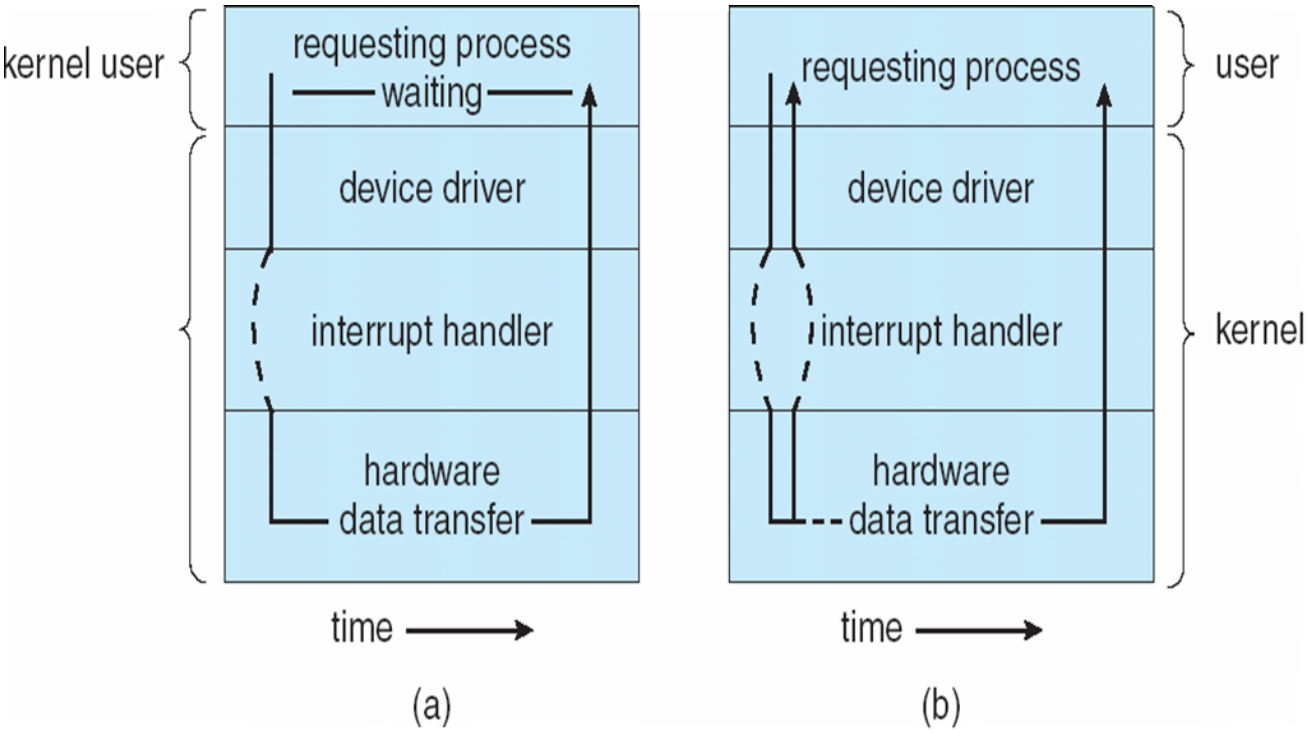
Insecure memory access with DMA

- DMA devices access memory using physical addresses, so they don't go through address translation
  - Do you sense something uncomfortable?
- When you leave your laptop (even you lock it), it may be attacked nearly arbitrarily
- DMA attack – a kind of physical attacks that allows you to read/write all physical memory
  - https://youtu.be/HDhpy7RpUjM
- Interception tool
  - https://github.com/carmaa/inception

Different styles of userspace I/O

- **Blocking** - process suspended until I/O completed
  - Easy to use and understand
  - Recall the first project: it is good to send the whole file
- **Nonblocking** - I/O call reads what is available, or writes what it can write until the device buffer is full, and then
  - Returns quickly with count of bytes read or written
  - `select()` to find if data ready then `read()` or `write()` to transfer
- **Asynchronous** - process runs while I/O executes
  - I/O subsystem signals process when I/O completed
  - Make use of multi-threading

Sync vs. Async



Outline

- I/O subsystem: hardware aspect
  - Terms: controller, bus, port
  - How are the device registers and memory addressed?
- I/O subsystem: software aspect
  - Device drivers programmer's perspective: polling, interrupt-driven, and DMA
  - User-space programmer's perspective: blocking, non-blocking, & asynchronous
- I/O subsystem: performance aspect
  - Caching, Buffering and Spooling
  - I/O scheduling

Some slides are courtesy of Dr. Abraham Silberschatz and Dr. Thomas Anderson

Vectored I/O

- **Vectored I/O** allows one system call to perform multiple I/O operations
- For example, Unix **`readv()`** and **`writev()`** accepts a vector of multiple buffers to read into or write from
- This **scatter-gather** method better than multiple individual I/O calls
  - Decreases the number of system calls

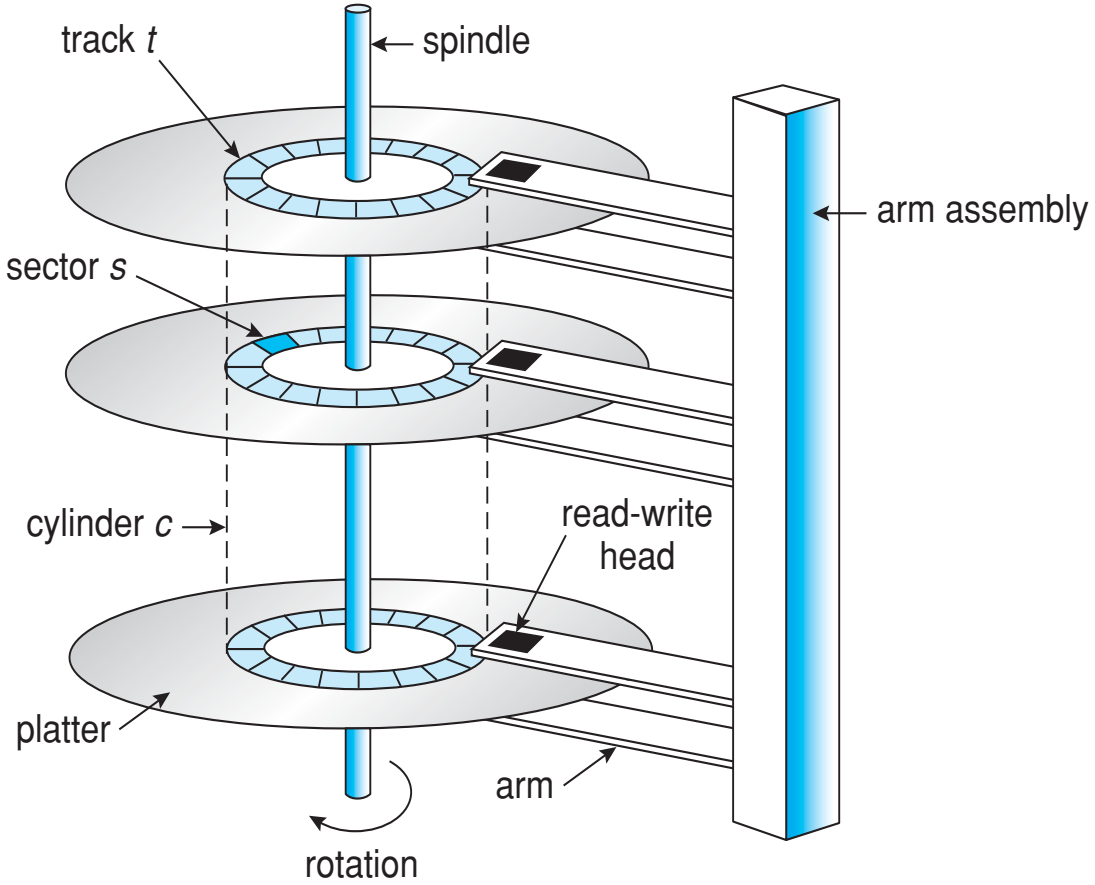
Several important generic ideas

- **Buffering** - store data in memory while transferring between devices
  - To cope with device speed mismatch – usually between producer and concumer
  - To enable I/O scheduling
  - To assemble multiple parts into a whole

Several important generic ideas

- **Spooling** - placing data in temporary working area for another program to process
  - Simultaneous Peripheral Operation On-Line
  - Process can access the resource without waiting. After writing the data on spool, process can perform other tasks. And a dedicated spooling process sends data on spool to the device
  - E.g: print spooling and Mail spooling etc.
  - Difference with buffering: Spooling is a special kind of buffering; it ensures data of different files/mails doesn't interleave
- **Caching** - faster device holding copy of data
  - Key to performance
  - Difference with buffering: data in cache is repetitively accessed, while data in buffer is consumed like a stream

Spinning disk



Disk Performance

Disk Latency =

Positioning time (= Seek Time + Rotation Time) + Transfer Time

- Seek Time: time to move disk arm over tracks (1-20ms)
- Rotation Time: time to rotate disk under disk head
  - Per rotation: 4 – 15ms (depending on price of disk)
  - On average, only need to wait half a rotation
- Transfer Time: time to transfer data onto/off of disk
  - Disk head transfer rate: 50-100MB/s (5-10 usec/sector)
  - Host transfer rate dependent on I/O connector (USB, SATA, …)

Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means minimizing the seek time
- Seek time ≈ seek distance

Disk Scheduling (Cont.)

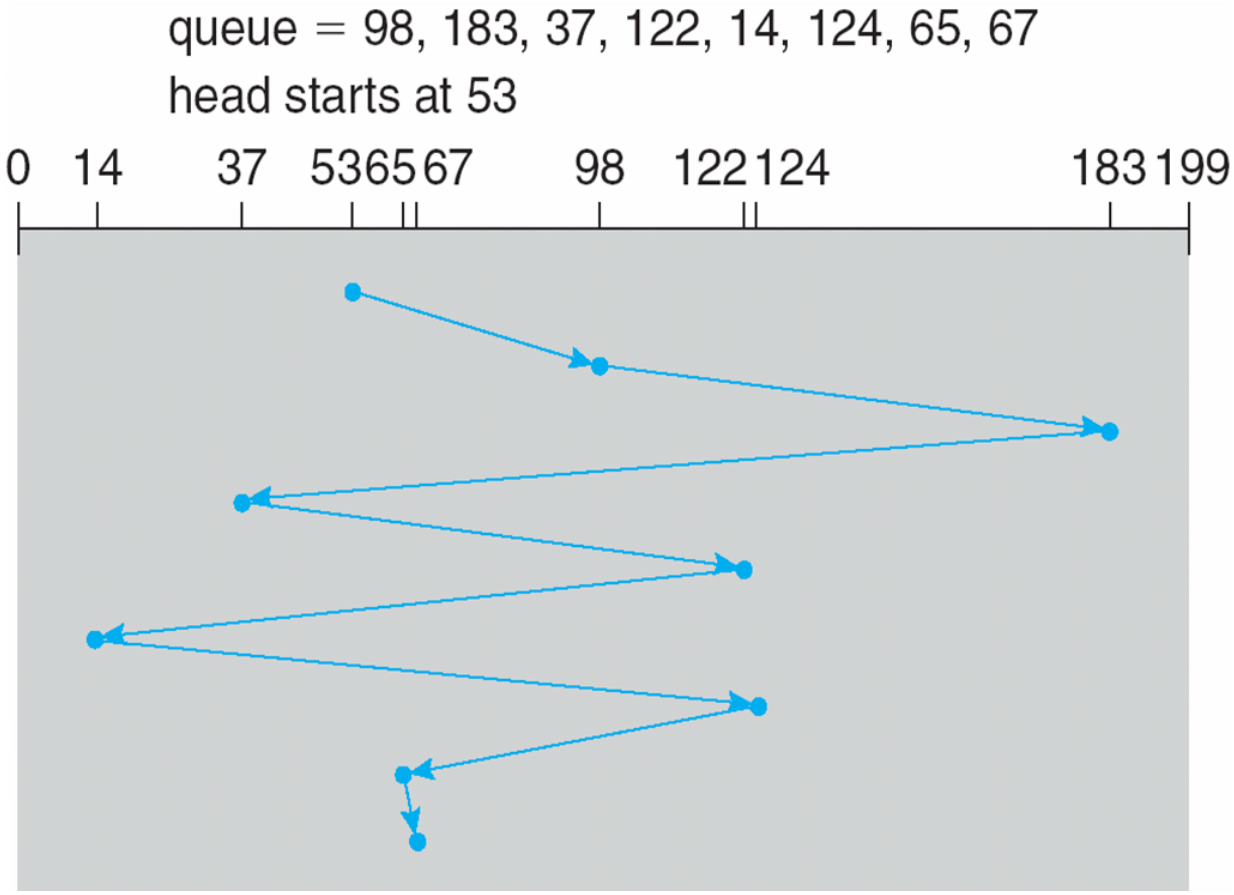
- Let's illustrate scheduling algorithms with a request queue; the numbers below are cylinder numbers:

  98, 183, 37, 122, 14, 124, 65, 67

  Current head pointer 53

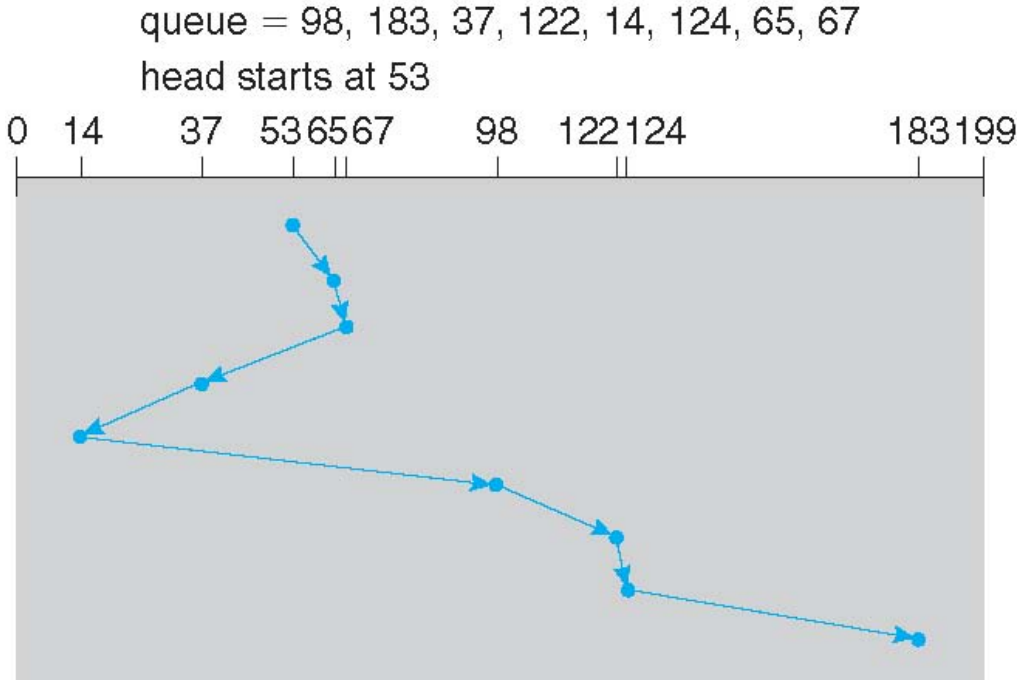
FCFS

Illustration shows total head movement of 640 cylinders



SSTF

- Shortest Seek Time First selects the request with the minimum seek time from the current head position
- It may cause starvation of some requests
- Illustration shows total head movement of 236 cylinders

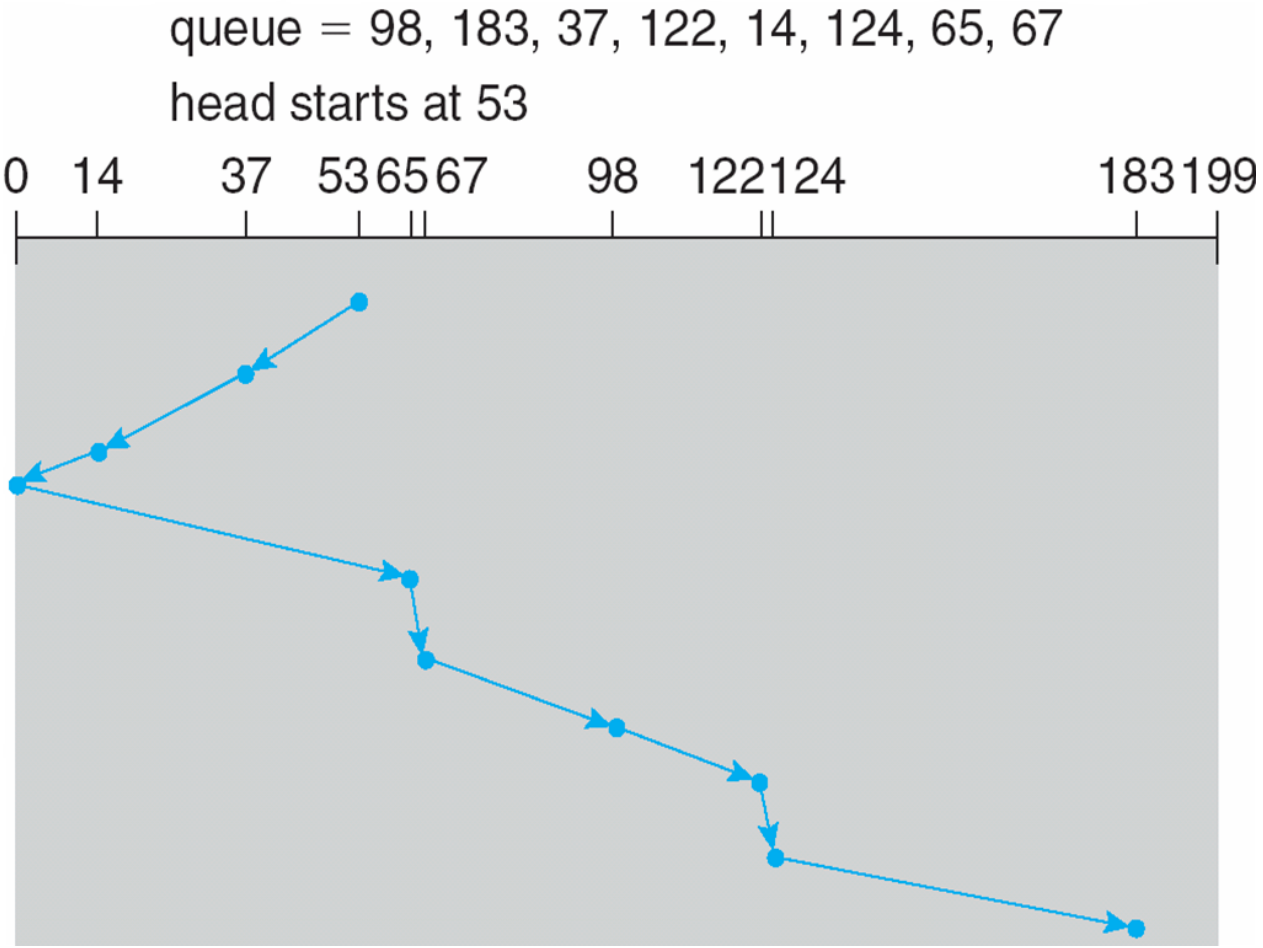


SCAN

- 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.
- **SCAN algorithm** Sometimes called the **elevator algorithm**
- Issue: after the arm reaches one end, and the requests near that end are probably sparse (since that area has been serviced just now), while the other end may have dense requests

SCAN (Cont.)

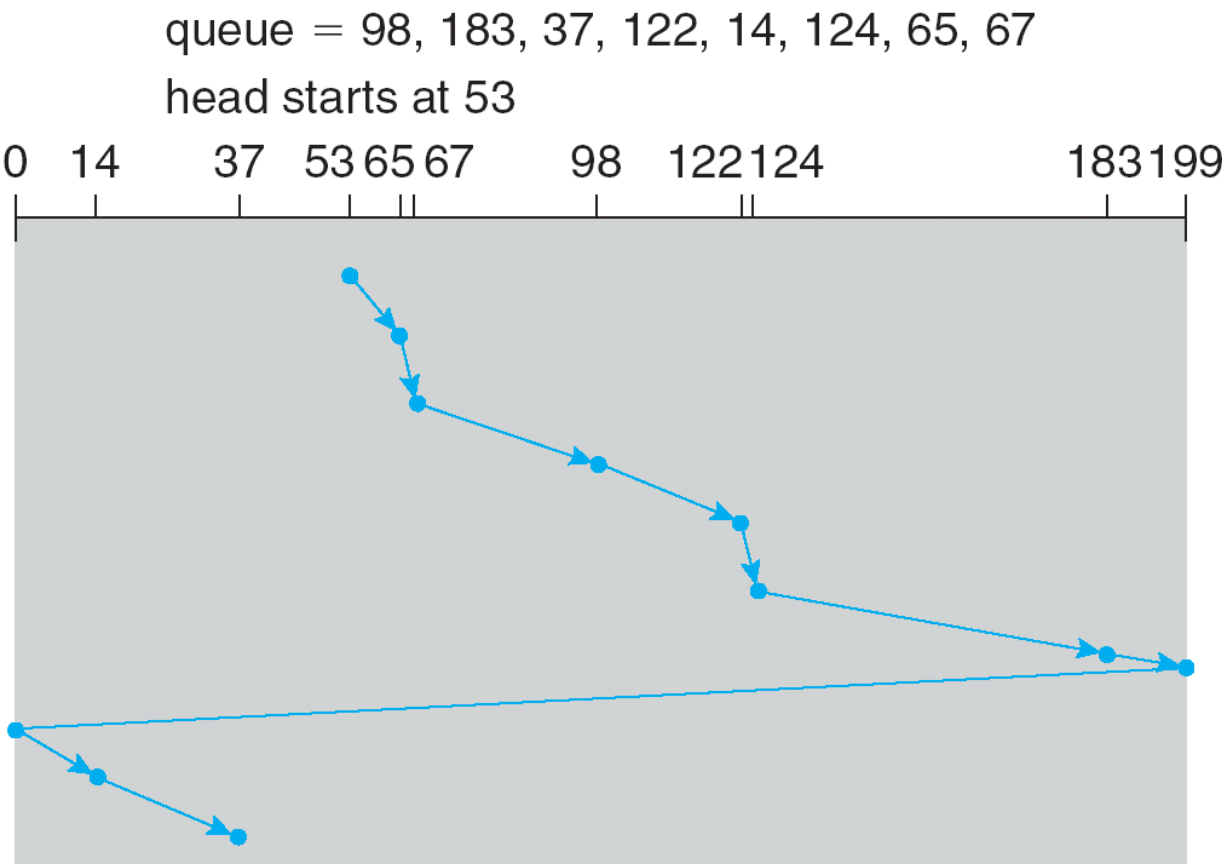
- Illustration shows total head movement of 208 cylinders



C-SCAN

- 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

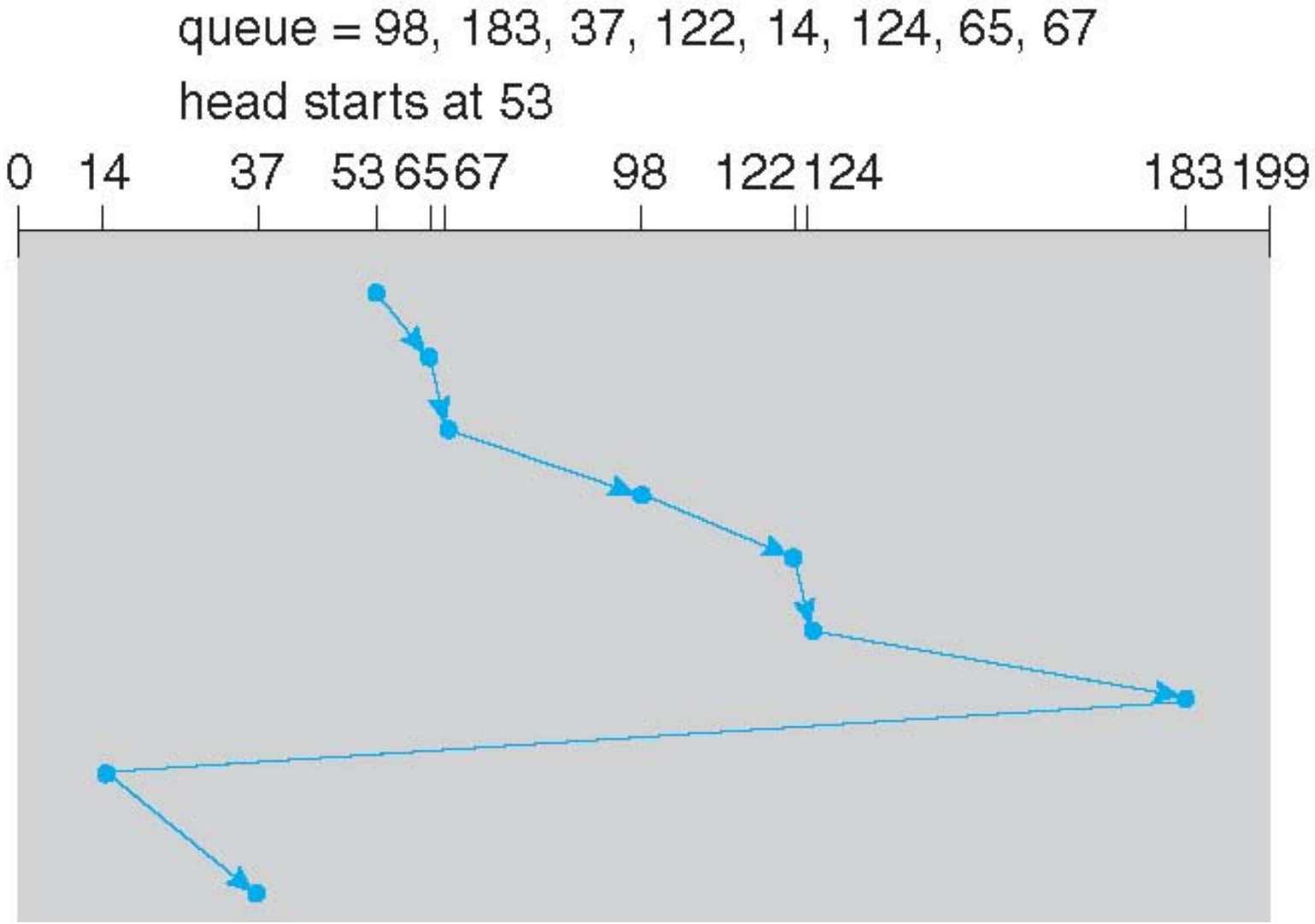
C-SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-LOOK

- 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 (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

Other second storage

- Solid state storage:
  - It has no moving parts and stores data using circuits; thus good bandwidth and lower power consumption
  - Flash
  - PCM (Phase Change Memory)
    - Faster and more scalable
  - Memristor
    - Much denser storage

Writing assignments

- Compare the ideas “buffering” and “caching”
- When to use “blocking I/O” and when to use “non-blocking I/O”
- What are the advantages and disadvantages of DMA
- What is the benefit of having the drivers to expose the same set of interfaces to the kernel?