EECS-343 Operating Systems
Lecture 14:
I/O and Disks

Steve Tarzia

Spring 2019

Northwestern

Announcements

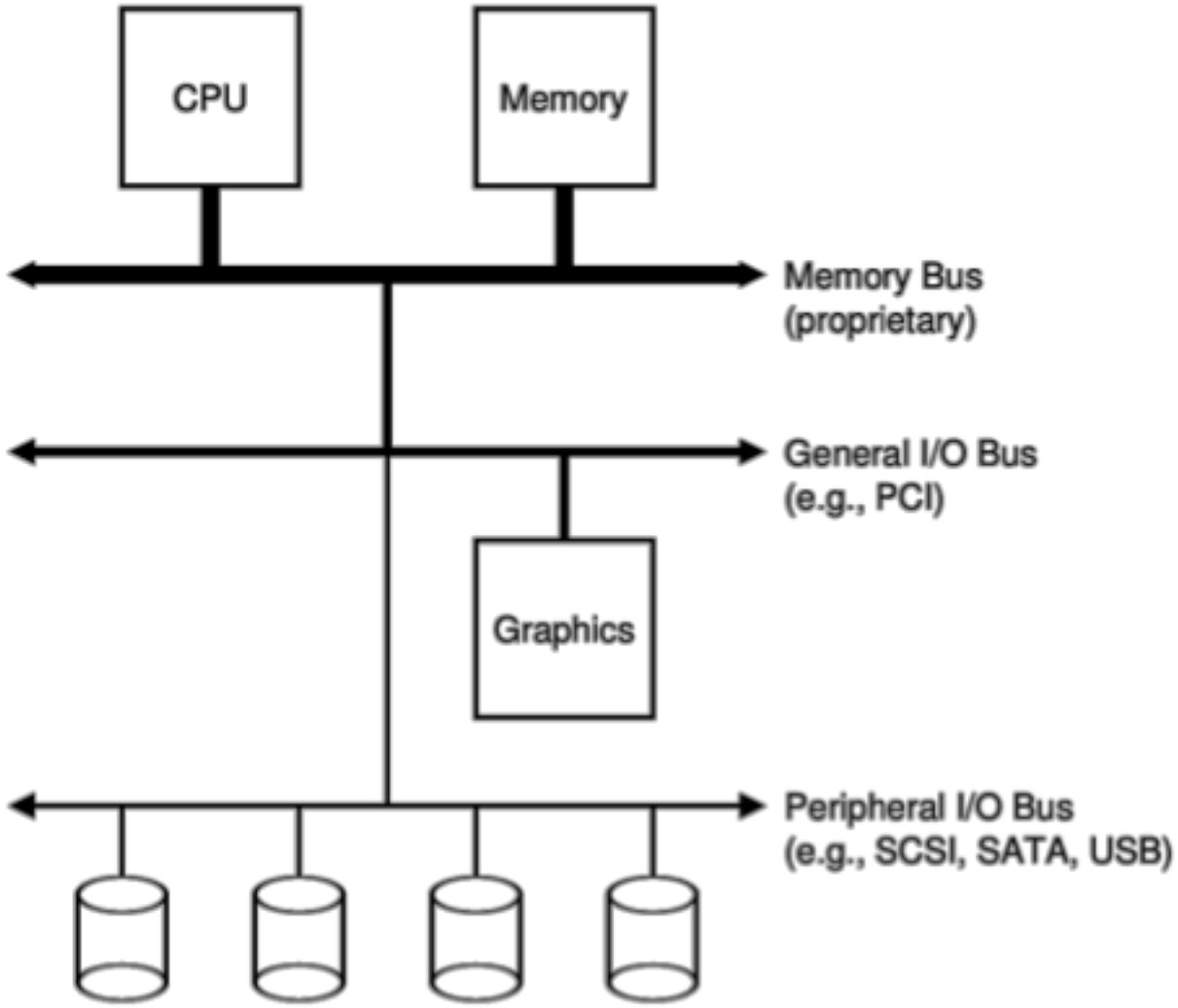
- Project 4 will be posted soon.

Last Lecture – Synchronization Bugs

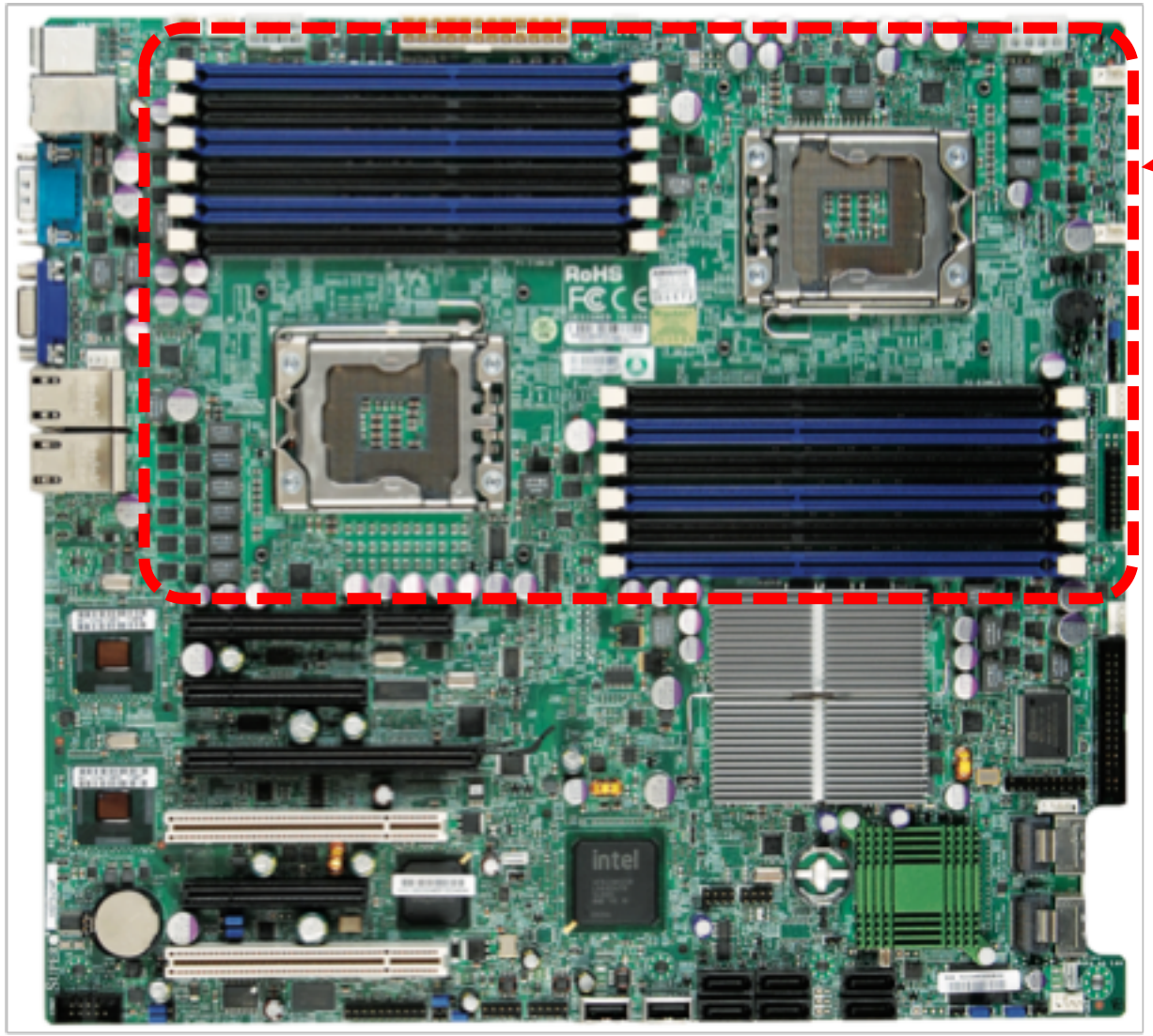
- ***Semaphore*** (up/down) is an all-purpose synchronization primitive
- ***Reader-writer*** lock allows multiple readers, but one writer.
- Adding too many locks can lead to ***deadlock***, which requires:
  - Mutual exclusion (avoid locks to avoid deadlock)
  - Hold and wait (use ***trylock*** to release first lock to before deadlocking)
  - No preemption
  - Circular wait (always acquire locks in the same order to avoid deadlock)
- Dining philosophers was an example of deadlock
  - Circular wait can be avoided by making one philosopher grab right-hand side instead of left first.

I/O is a major responsibility of the kernel

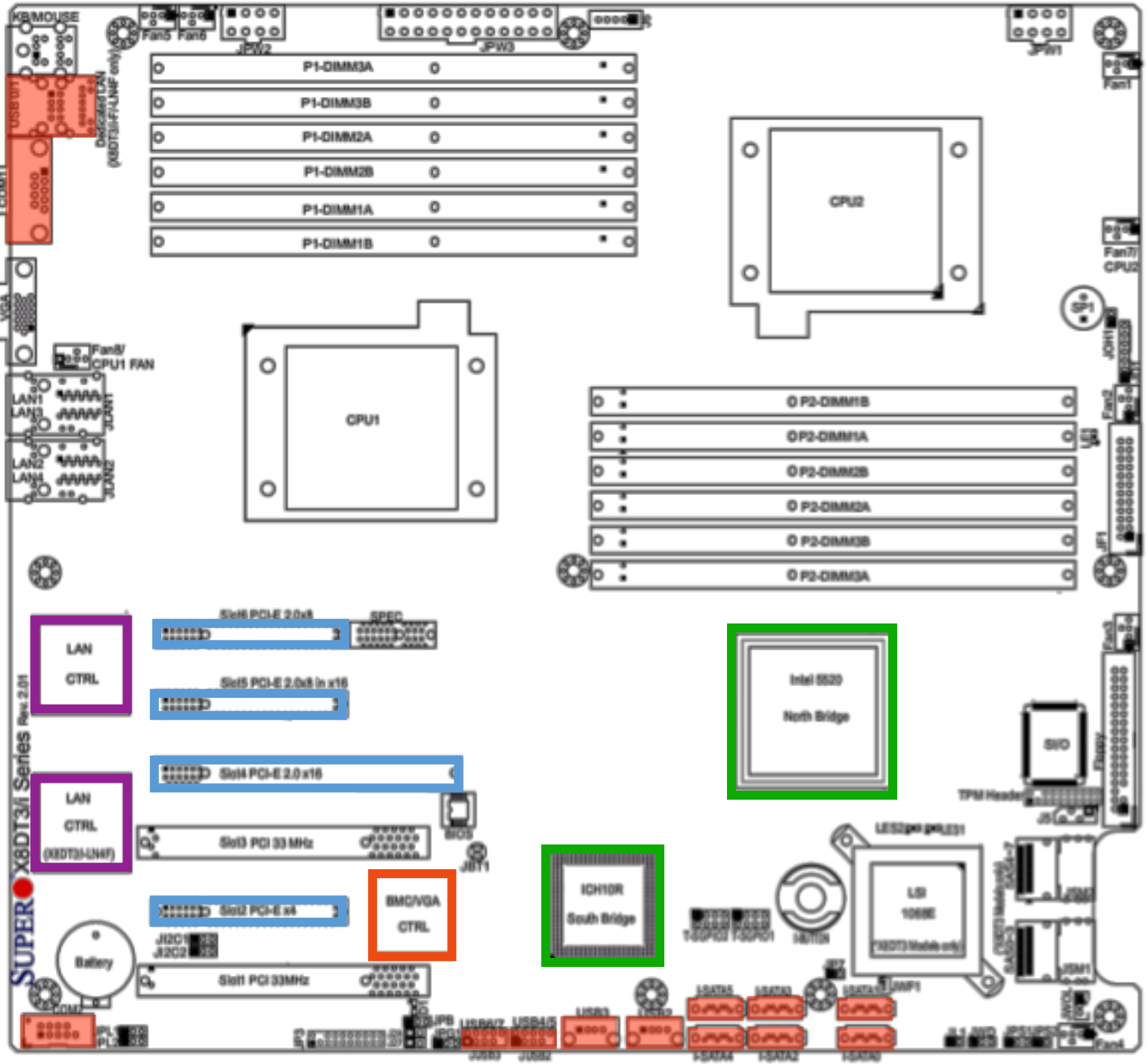
- I/O = Input/Output
- Peripheral hardware is a shared resource, and not directly accessible by user programs.
- Includes:
  - Disks
  - Graphics
  - Network
  - Keyboard/mouse
  - Audio
  - Webcam
  - Printer



A look at a motherboard (Supermicro X8DT3, ~2011)



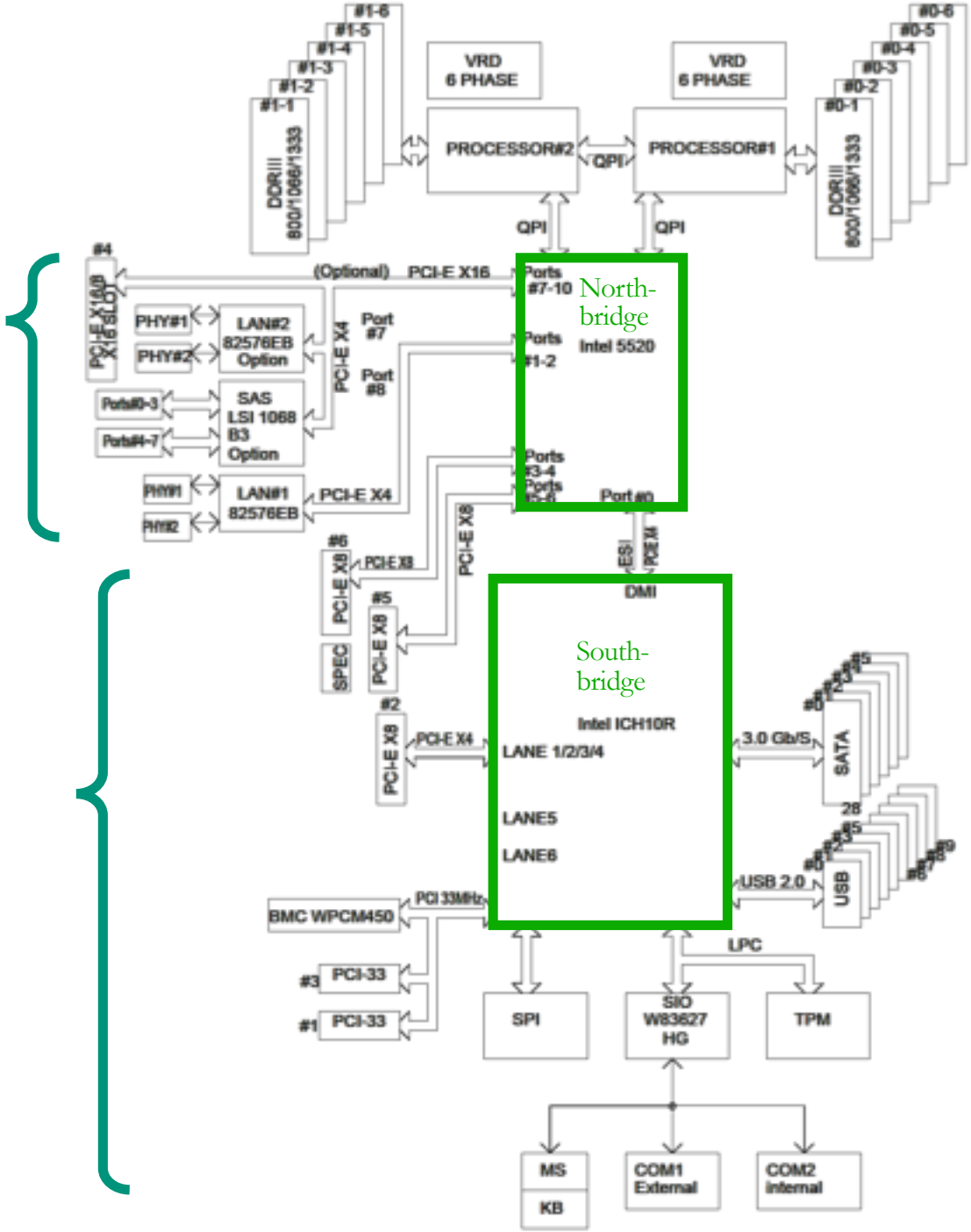
- 2 CPU sockets and 12 RAM slots in upper right.
- This is what we've dealt with so far.
- The rest of the board is dedicated to I/O controllers and devices.
  - This part varies dramatically from machine to machine, and the OS must somehow handle different HW.



- Northbridge & Southbridge
  (serving many functions)
- Graphics (VGA)
- 2 Network controllers
  (with 4 ports)
- PCI-Express slots for expansion cards
  - Eg., a high-powered graphics card.
- Ports for:
  - SATA disks
  - USB
  - RS232 serial port
- This physical view shows *where to plug in peripheral HW*

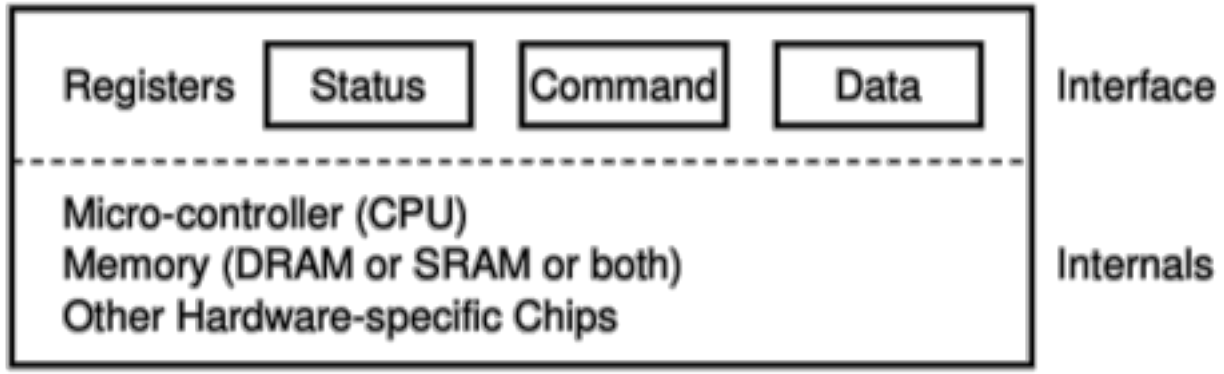
OS's view of this machine

- **Northbridge** handles high-speed I/O devices on PCI-Express bus
  - Two network controllers and (optional) LSI-brand RAID (disk) controller is soldered onto board
  - Additional high-speed devices can be installed in PCI-E slots.
- **Southbridge** handles slower/legacy devices. In this case:
  - A crappy VGA card (BMC WPCM450)
  - USB controller
  - SATA disk controllers
  - Keyboard, mouse, rs232 serial ports



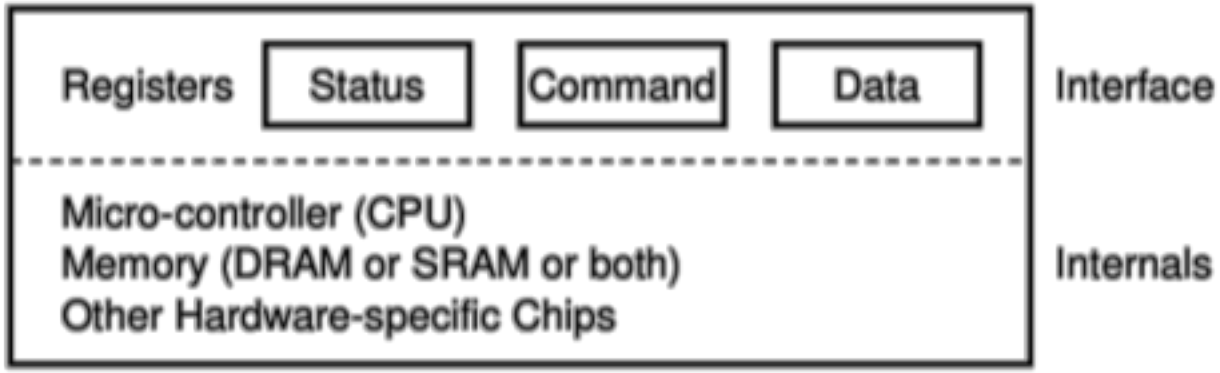
How does OS interact with I/O devices?

- A hardware device can be a complex mini-computer within the computer, but we just consider the interface it exposes:



- Generally, the OS sees just a few registers to read/write, and
- An instruction manual (spec) explaining how to use those registers.

A simple *polling* I/O device interaction




```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
// This starts the device and executes the command
While (STATUS == BUSY)
    ; // wait until device is done with the request
```

I/O modes

- ***Programmed I/O*** (PIO) is when the CPU transfers the data.
  - Works well for quick I/O
  - No need for a context switch/interrupt
- ***Direct Memory Access*** (DMA) is an alternative that frees the CPU from this "busy work"
  - DMA controller is programmed by the OS to handle the data transfer loop,
  - Or the device is given direct access to read/write RAM (a.k.a. bus mastering)
  - DMA is very common, even in old PC hardware (1980s)
- In both cases I/O can be ***asynchronous***:
  - When device is ready, generate an ***interrupt*** to let CPU/OS handle it.

How does OS interact with device registers?

x86 provides two options:

- **`in`**/**`out`** instructions read/write data to ***I/O ports***
  - Ports are like addresses for I/O registers
  - These are *privileged* instructions
- ***Memory-mapped I/O***
  - Hardware (northbridge) maps some physical memory addresses to I/O
  - Kernel uses simple **`mov`** instructions to read/write to devices.
  - Must be careful to disable CPU caching of these memory addresses

Functionally, there is not much difference between the two.

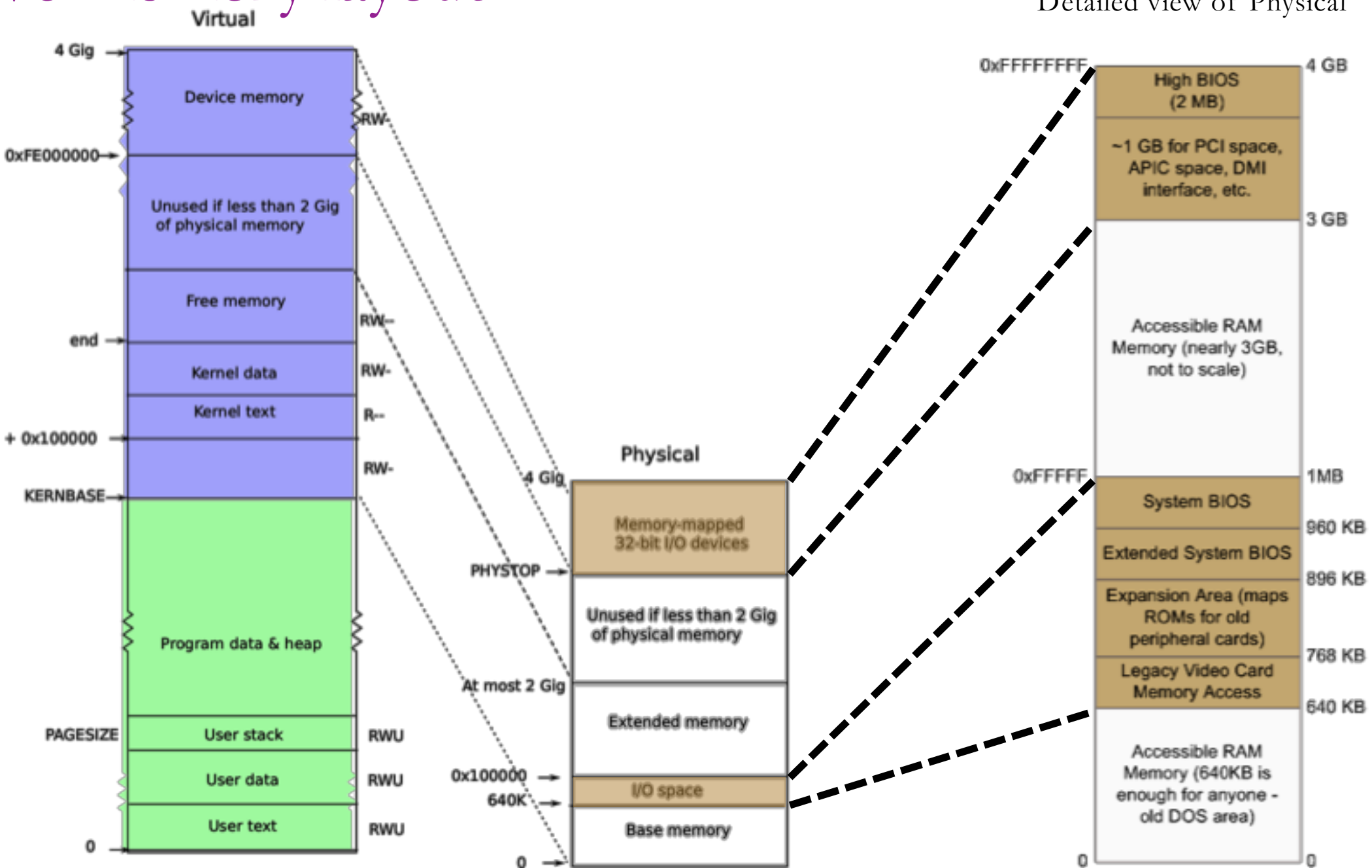
I/O ports

These are just typical values, not standardized.

https://wiki.osdev.org/Can_I_have_a_list_of_IO_Ports

Port range	Summary
0x0000-0x001F	The first legacy DMA controller, often used for transfers to floppies.
0x0020-0x0021	The first Programmable Interrupt Controller
0x0022-0x0023	Access to the Model-Specific Registers of Cyrix processors.
0x0040-0x0047	The PIT (Programmable Interval Timer)
0x0060-0x0064	The "8042" PS/2 Controller or its predecessors, dealing with keyboards and mice.
0x0070-0x0071	The CMOS and RTC registers
0x0080-0x008F	The DMA (Page registers)
0x0092	The location of the fast A20 gate register
0x00A0-0x00A1	The second PIC
0x00C0-0x00DF	The second DMA controller, often used for soundblasters
0x00E9	Home of the Port E9 Hack. Used on some emulators to directly send text to the hosts' console.
0x0170-0x0177	The secondary ATA harddisk controller.
0x01F0-0x01F7	The primary ATA harddisk controller.
0x0278-0x027A	Parallel port
0x02F8-0x02FF	Second serial port
0x03B0-0x03DF	The range used for the IBM VGA, its direct predecessors, as well as any modern video card in legacy mode.
0x03F0-0x03F7	Floppy disk controller
0x03F8-0x03FF	First serial port

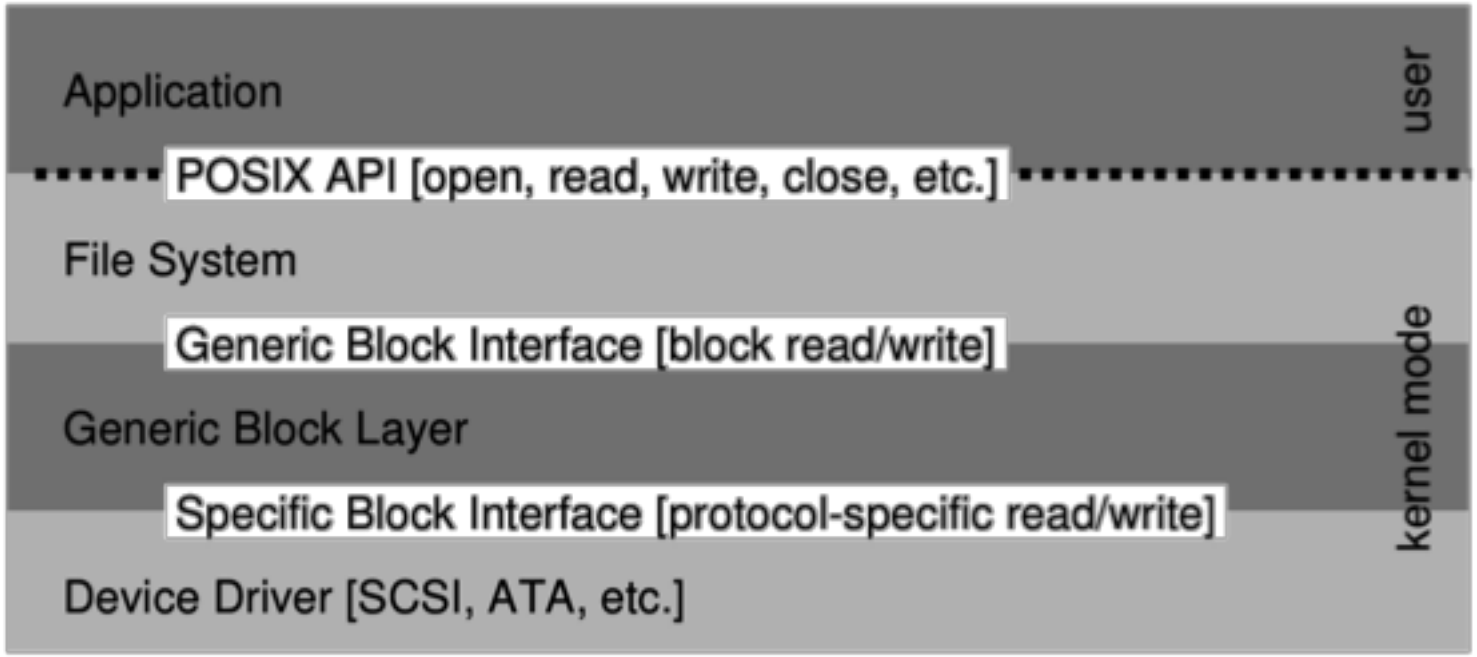
xv6 memory layout



Detailed view of Physical

Hardware abstraction

- OS creates a layered view of storage:
- Abstractions are used even within the kernel.



- Layered approach allows lower parts to be easily changed.
  - Eg., filesystem implementation is independent of the type of disk
- Code for managing I/O devices is in kernel ***device drivers***
  - 70% of Linux code is device drivers, because there are so many different devices, each is different! (15.3M lines of source code!)
  - Device drivers are the most common source of kernel bugs
  - A given driver may be used by just a small fraction of systems, so it's not highly exercised or scrutinized.

IDE disk drivers in Linux

```
steve-macbook-retina:linux steve$ ls drivers/ide/
Kconfig             dtc2278.c           ide-disk_ioctl.c    ide-lib.c           it8213.c            serverworks.c
Makefile            falconide.c         ide-disk_proc.c     ide-park.c          it821x.c            setup-pci.c
aec62xx.c           gayle.c             ide-dma-sff.c       ide-pci-generic.c   jmicron.c           sgiioc4.c
ali14xx.c           hpt366.c            ide-dma.c           ide-pio-blacklist.c macide.c            siimage.c
alim15x3.c          ht6560b.c           ide-eh.c            ide-pm.c            ns87415.c           sis5513.c
amd74xx.c           icside.c            ide-floppy.c        ide-pnp.c           opti621.c           sl82c105.c
atiixp.c            ide-4drives.c       ide-floppy.h        ide-probe.c         palm_bk3710.c       slc90e66.c
au1xxx-ide.c        ide-acpi.c          ide-floppy_ioctl.c  ide-proc.c          pdc202xx_new.c      tc86c001.c
buddha.c            ide-atapi.c         ide-floppy_proc.c   ide-scan-pci.c      pdc202xx_old.c      triflex.c
cmd640.c            ide-cd.c            ide-gd.c            ide-sysfs.c         piix.c              trm290.c
cmd64x.c            ide-cd.h            ide-gd.h            ide-tape.c          pmac.c              tx4938ide.c
cs5520.c            ide-cd_ioctl.c      ide-generic.c       ide-taskfile.c      q40ide.c            tx4939ide.c
cs5530.c            ide-cd_verbose.c    ide-io-std.c        ide-timings.c       qd65xx.c            umc8672.c
cs5535.c            ide-cs.c            ide-io.c            ide-xfer-mode.c     qd65xx.h            via82cxxx.c
cs5536.c            ide-devsets.c       ide-ioctls.c        ide.c               rapide.c
cy82c693.c          ide-disk.c          ide-iops.c          ide_platform.c      rz1000.c
delkin_cb.c         ide-disk.h          ide-legacy.c        it8172.c            sc1200.c
```

- 92 different drivers, just for IDE disk controllers
  - Each driver supports a family of related controllers
- 39k lines of code in total. ~426 lines each, on average
  - There is another 222k lines for SCSI disk controllers! (119 drivers)

Abstraction of a disk

- Each of the 92 different disk controller drivers on Linux provides the same simple abstraction to upper layers: a block device.
- The kernel adds a block buffering/caching layer (*covered later*)
- User code (if running with *root* permissions) and kernel filesystem code can access each disk as a virtual device file:
  - /dev/sda *the first disk*
  - /dev/sdb *the second disk*
  - /dev/sdc *and so on …*
- Files and disks are both just arrays of bytes (although files are byte addressable and disks are block addressable -- 512 byte chunks).

Abstraction of a sound card

- Unix systems abstract many devices as virtual files.
- **OSS** – the Open Sound System was an early Linux sound interface that uses the virtual file /dev/dsp.

- To play some white noise through the speaker:

```
cat /dev/random > /dev/dsp
```

- To read samples from the microphone to file a.a:

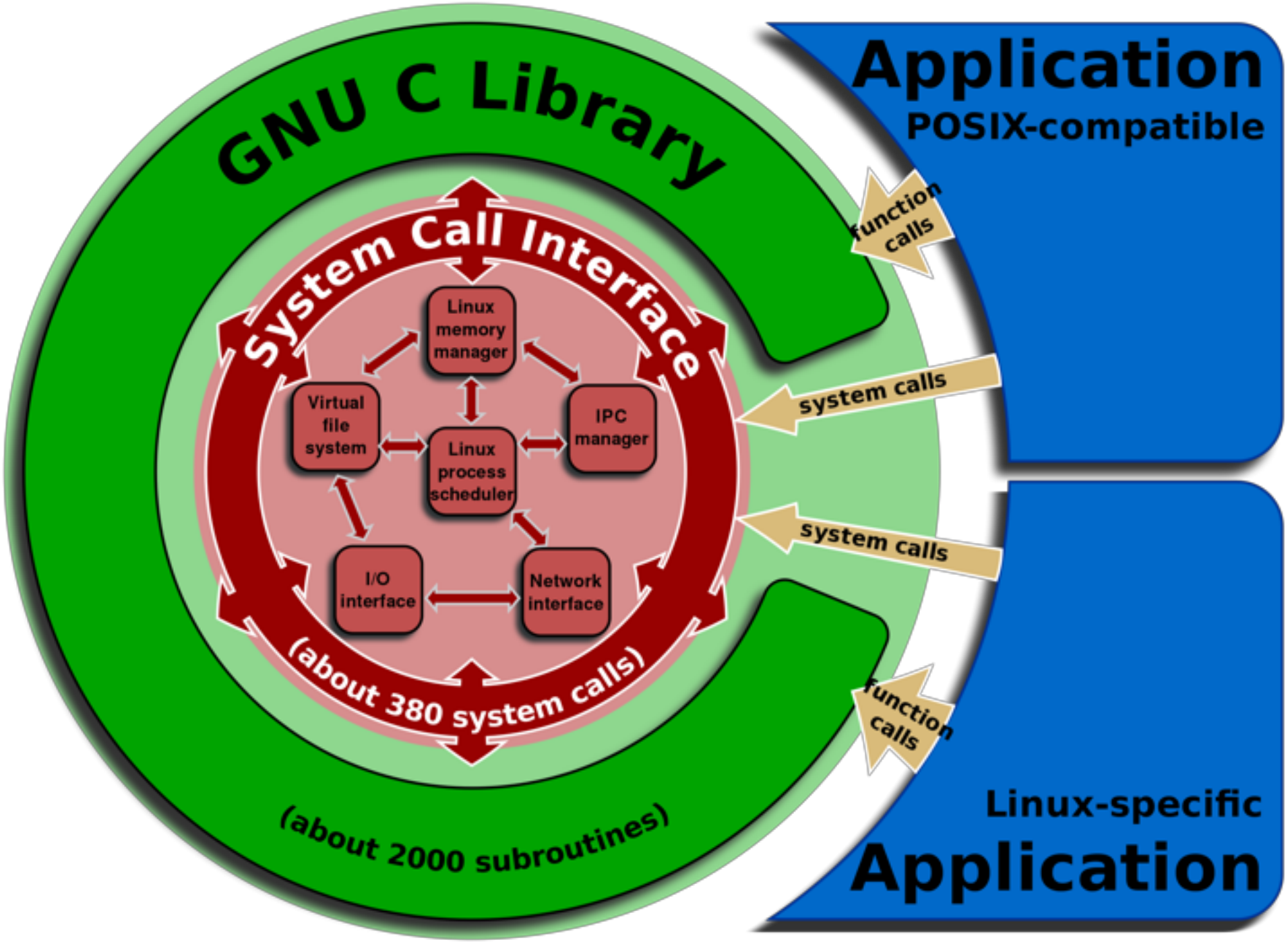
```
cat /dev/dsp > a.a
```

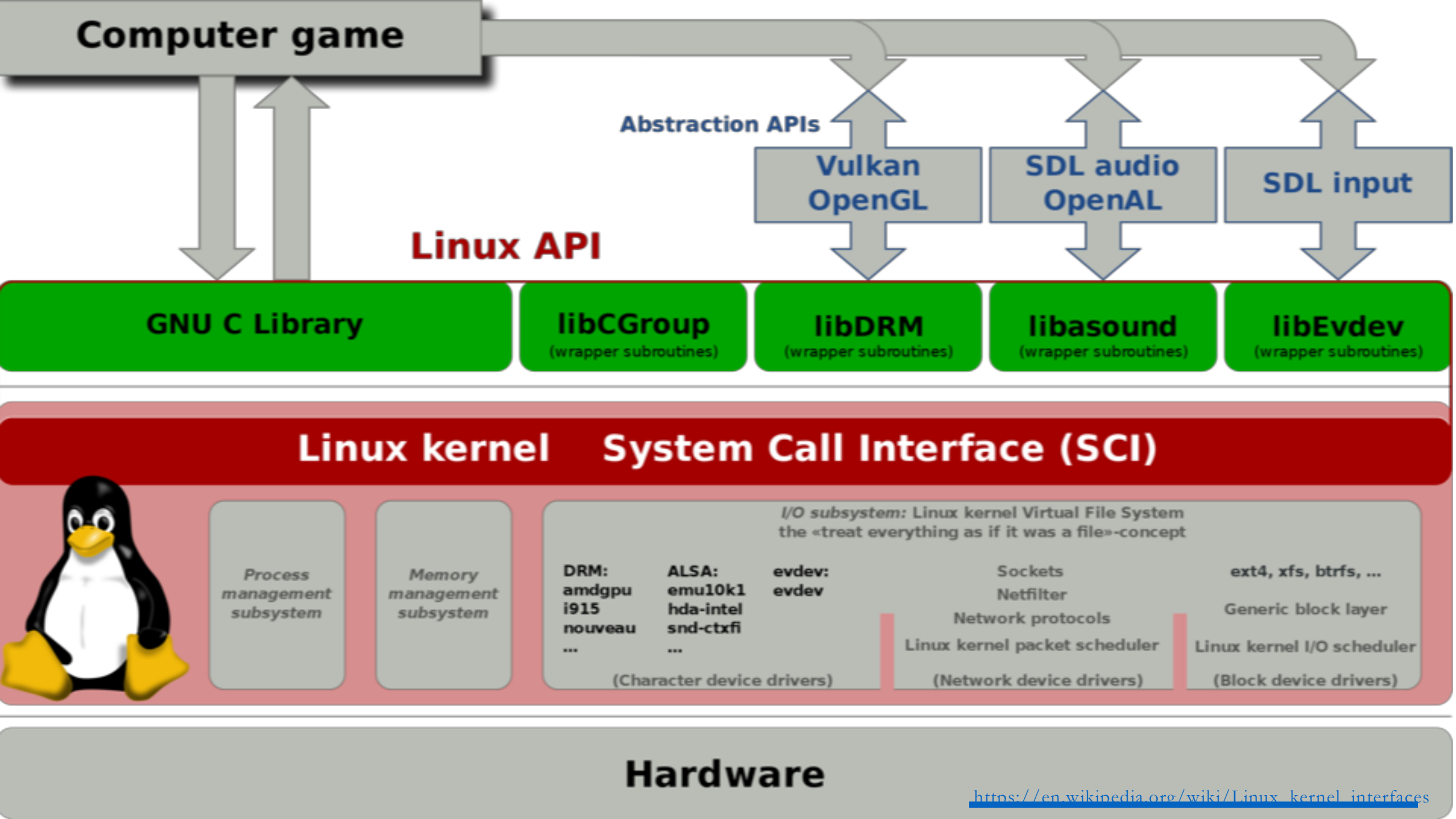
- Thus, programs can be written to use audio without any concern for the particular sound card's details, if the kernel has an OSS driver.

Kernel's interfaces for IO

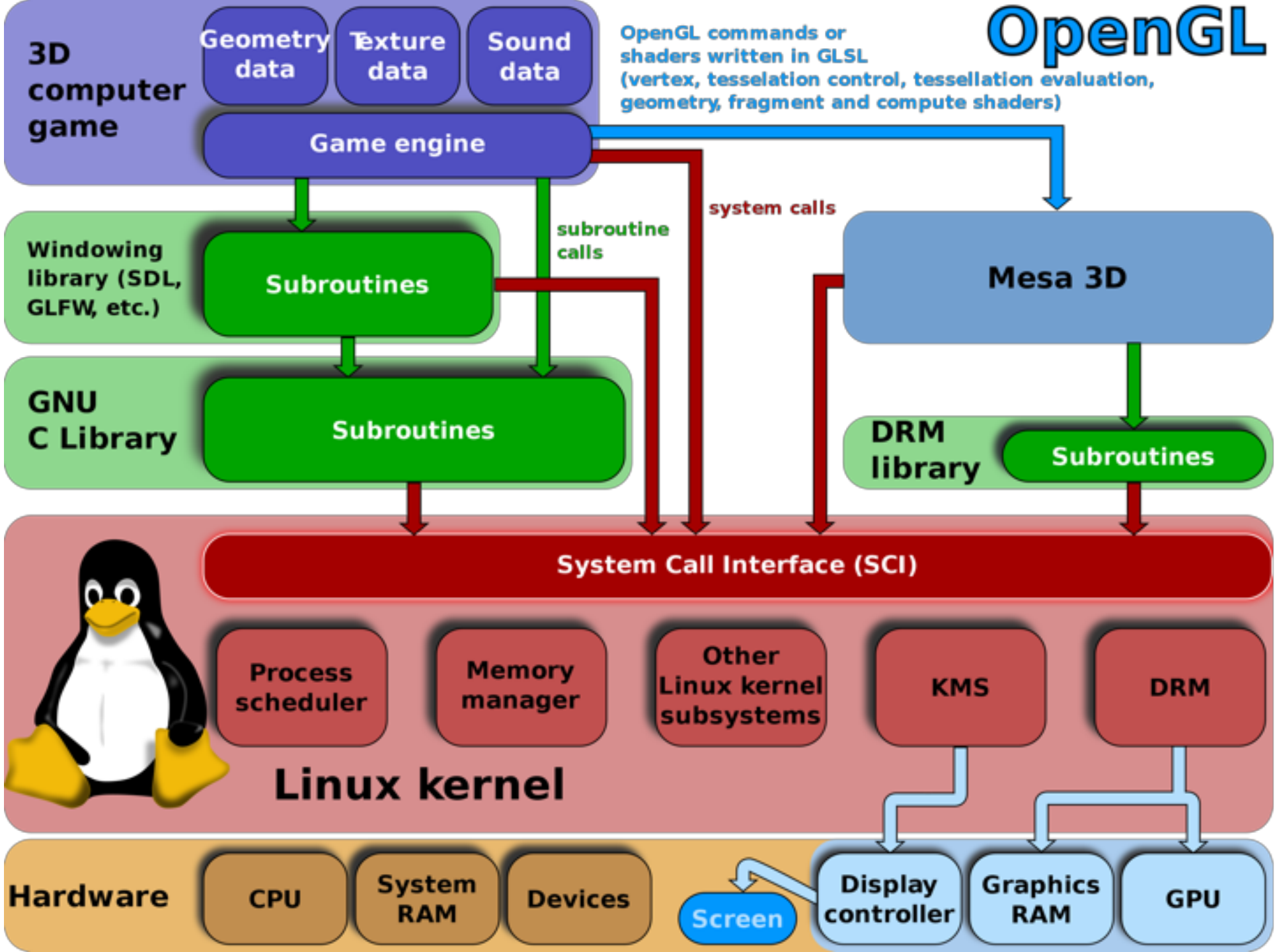
Kernels provide several different interfaces for user processes to do IO:

- **System calls**
  - eg., network connections (sockets) have dedicated system calls on POSIX.
- **Libraries** *(that use system calls)*
  - SDL (Standard DirectMedia Library)
  - glibc (printf, file open/write/read, etc.)
- **Virtual file system**
  - eg., disks, sound cards, serial ports
  - Would use syscalls or glibc to access the virtual files.


GNU C Library
System Call Interface
Linux memory manager
Virtual file system
IPC manager
Linux process scheduler
I/O interface
Network interface
(about 380 system calls)
(about 2000 subroutines)
Application
POSIX-compatible
function calls
system calls
system calls
function calls
Linux-specific
Application



https://en.wikipedia.org/wiki/Linux_kernel_interfaces


3D computer game
Geometry data
Texture data
Sound data
Game engine
OpenGL commands or shaders written in GLSL (vertex, tesselation control, tessellation evaluation, geometry, fragment and compute shaders)
OpenGL
system calls
subroutine calls
Windowing library (SDL, GLFW, etc.)
Subroutines
Mesa 3D
GNU C Library
Subroutines
DRM library
Subroutines
System Call Interface (SCI)
Process scheduler
Memory manager
Other Linux kernel subsystems
KMS
DRM
Linux kernel
Hardware
CPU
System RAM
Devices
Screen
Display controller
Graphics RAM
GPU

Device driver example

IDE disk controller (for hard disk or CD-rom drive)

Example IDE registers for Programmed I/O

```
Control Register:
  Address 0x3F6 = 0x80 (0000 1RE0): R=reset, E=0 means "enable interrupt"

Command Block Registers:
  Address 0x1F0 = Data Port
  Address 0x1F1 = Error
  Address 0x1F2 = Sector Count
  Address 0x1F3 = LBA low byte
  Address 0x1F4 = LBA mid byte
  Address 0x1F5 = LBA hi  byte
  Address 0x1F6 = 1B1D TOP4LBA: B=LBA, D=drive
  Address 0x1F7 = Command/status
```

```
Status Register (Address 0x1F7):
    7      6     5      4     3     2     1      0
   BUSY  READY FAULT SEEK  DRQ  CORR IDDEX ERROR

Error Register (Address 0x1F1): (check when Status ERROR==1)
    7      6     5      4     3     2     1     0
   BBK    UNC   MC    IDNF  MCR  ABRT T0NF AMNF

   BBK  = Bad Block
   UNC  = Uncorrectable data error
   MC   = Media Changed
   IDNF = ID mark Not Found
   MCR  = Media Change Requested
   ABRT = Command aborted
   T0NF = Track 0 Not Found
   AMNF = Address Mark Not Found
```

- 9 one-byte registers for interacting with device
- Use x86 **`in/out`** instructions
- Sometimes individual bits in registers are used.
- 0x3F6 can enable interrupts
- 0x1F0 is a data buffer
- 0x1F7 is where you write command.
- Grab a lock before accessing device!

Using an IDE disk's I/O ports

```
static int ide_wait_ready() {
  while (((int r = inb(0x1f7)) & IDE_BSY) || !(r & IDE_DRDY))
    ;                                   // loop until drive isn't busy
}

static void ide_start_request(struct buf *b) {
  ide_wait_ready();
  outb(0x3f6, 0);                       // enable interrupt
  outb(0x1f2, 1);                       // how many sectors?
  outb(0x1f3, b->sector & 0xff);        // LBA goes here ...
  outb(0x1f4, (b->sector >> 8) & 0xff);    // ... and here
  outb(0x1f5, (b->sector >> 16) & 0xff);   // ... and here!
  outb(0x1f6, 0xe0 | ((b->dev&1)<<4) | ((b->sector>>24)&0x0f));
  if(b->flags & B_DIRTY){
    outb(0x1f7, IDE_CMD_WRITE);         // this is a WRITE
    outsl(0x1f0, b->data, 512/4);       // transfer data too!
  } else {
    outb(0x1f7, IDE_CMD_READ);          // this is a READ (no data)
  }
}
```

0x1f7 = command/status
0x3f6 = control register
0x1f3-0x1f6 = disk address
0x1f0 = data

Simplified xv6 IDE driver (continued)

```
void ide_rw(struct buf *b) {
  acquire(&ide_lock);
  for (struct buf **pp = &ide_queue; *pp; pp=&(*pp)->qnext)
    ;                                  // walk queue
  *pp = b;                             // add request to end
  if (ide_queue == b)                  // if q is empty
    ide_start_request(b);              // send req to disk
  while ((b->flags & (B_VALID|B_DIRTY)) != B_VALID)
    sleep(b, &ide_lock);               // wait for completion
  release(&ide_lock);
}

void ide_intr() {
  struct buf *b;
  acquire(&ide_lock);
  if (!(b->flags & B_DIRTY) && ide_wait_ready(1) >= 0)
    insl(0x1f0, b->data, 512/4);       // if READ: get data
  b->flags |= B_VALID;
  b->flags &= ~B_DIRTY;
  wakeup(b);                           // wake waiting process
  if ((ide_queue = b->qnext) != 0)     // start next request
    ide_start_request(ide_queue);      // (if one exists)
  release(&ide_lock);
}
```

Queue new requests

Interrupt handler

Break time

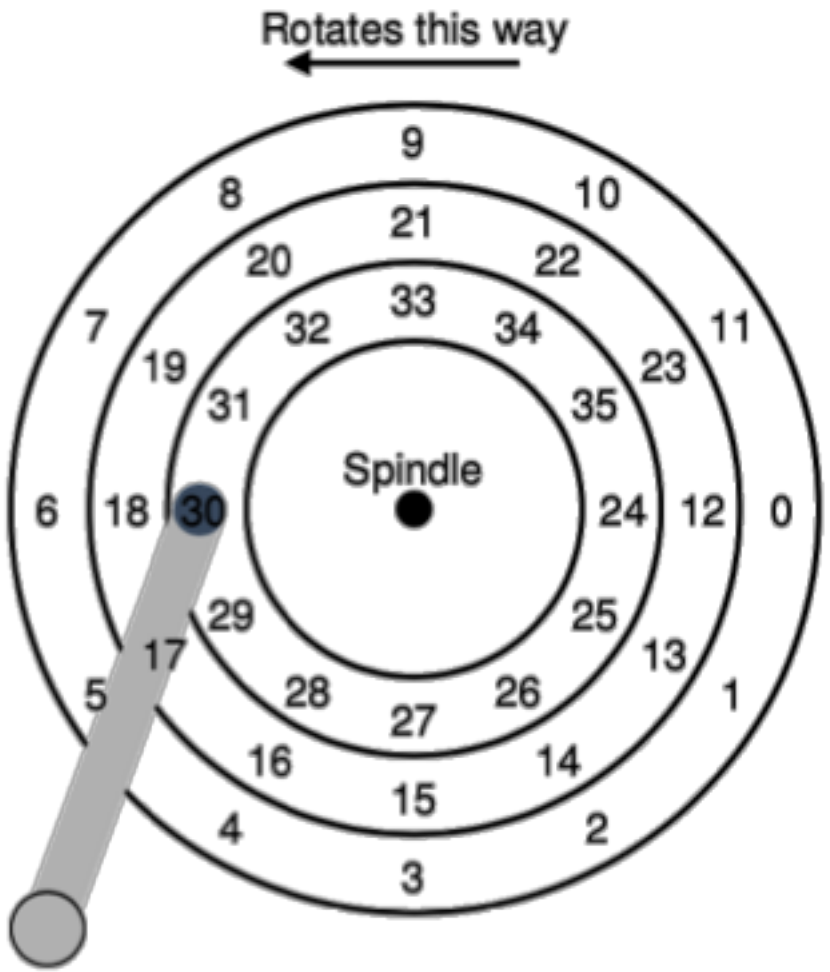
"The two things that really drew me to vinyl were the expense and the inconvenience."

"It's curiosity."

How magnetic disks work

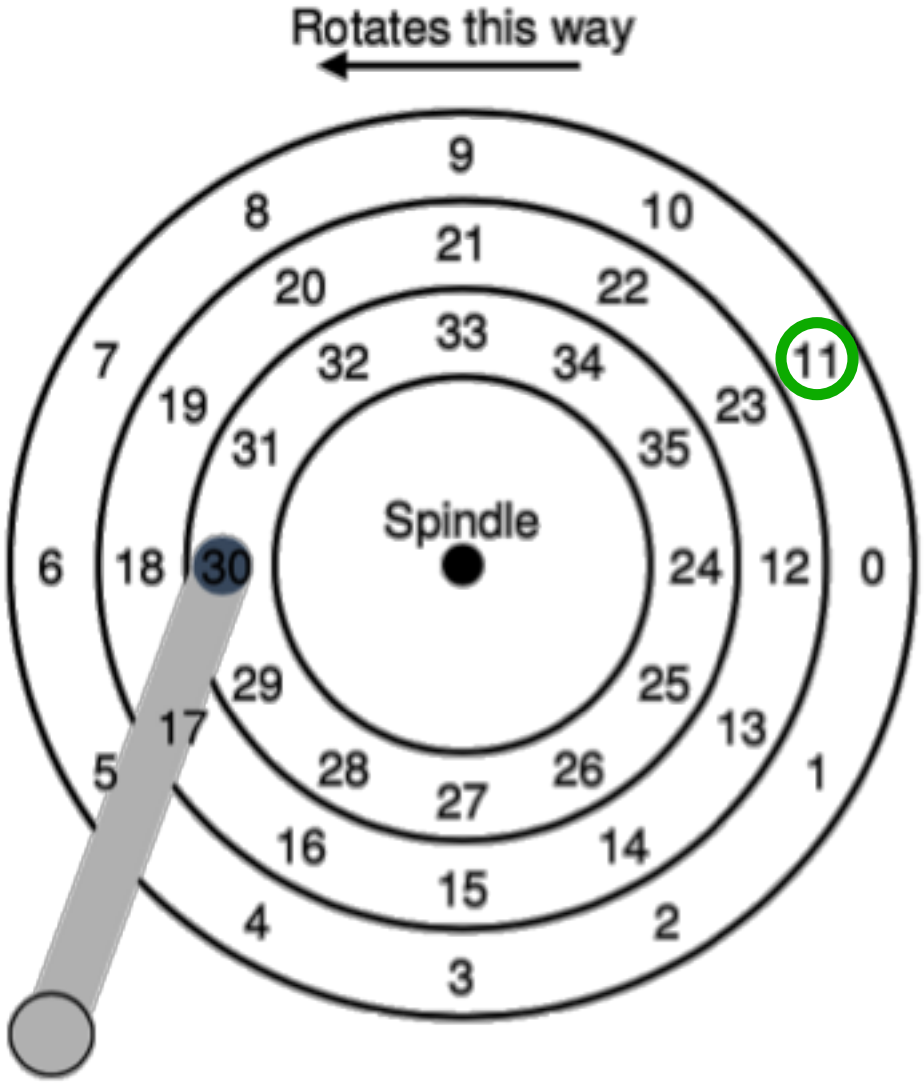
- All hard disk drives (magnetic or SSD) present a simple abstraction to the OS: an array of 512-byte ***sectors***.
  - Read or write a 512-byte sector
  - Number of sectors determines the disk capacity.
- Solid state disks (SSDs) are common on laptops/smartphones
  - Allow quick random access, like RAM.
  - But these have smaller capacity
- Magnetic disks, however, have more complex physical properties…

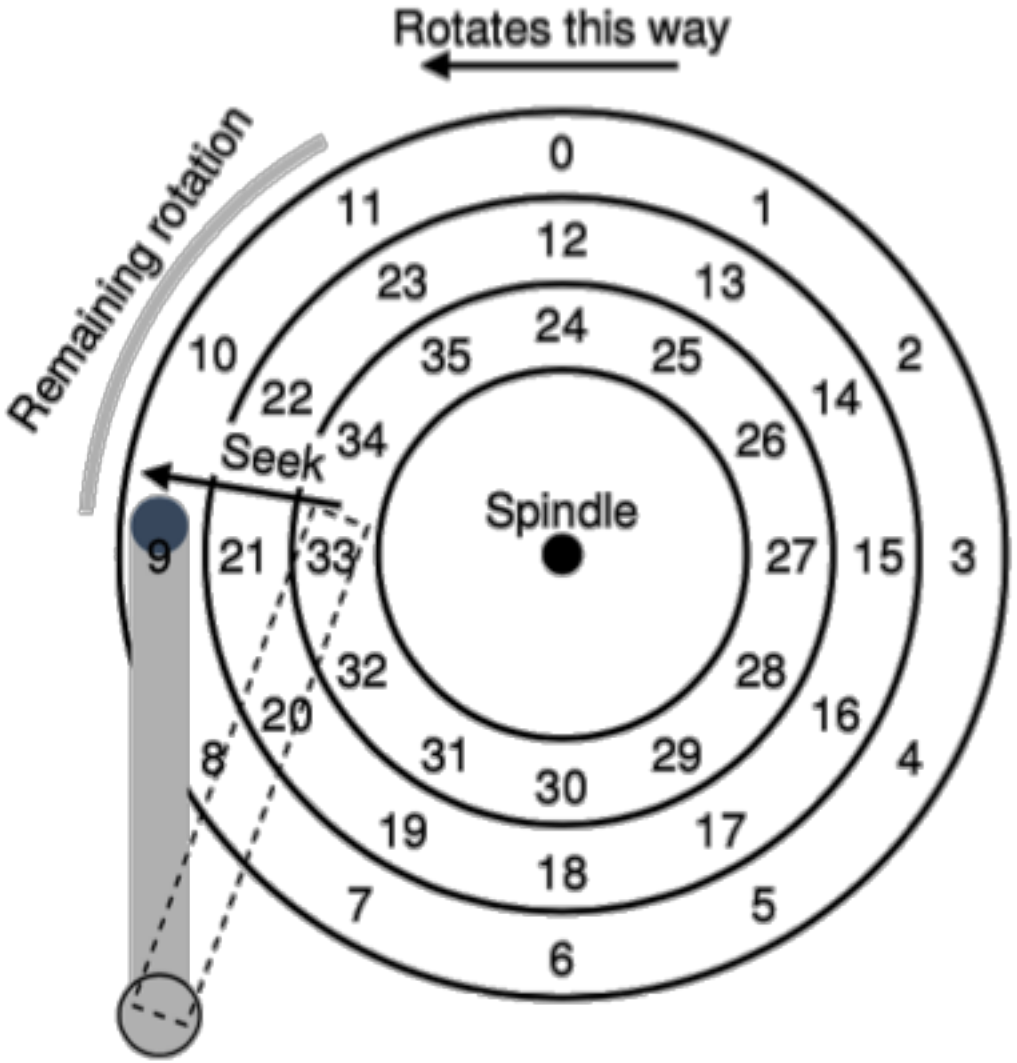
Magnetic disk geometry



- Sectors are arranged on circular ***tracks***
- A ***read/write head*** is attached to an arm
  - Head senses/magnetizes the disk's surface
  - Data can only be read/written under the head
  - Arm angle can change to reach different tracks (called ***seeking***).
- A motor rotates the disk at constant speed (4200-15k RPM)
  - ***Rotational delay*** is incurred as we wait for a given sector to rotate under the head.

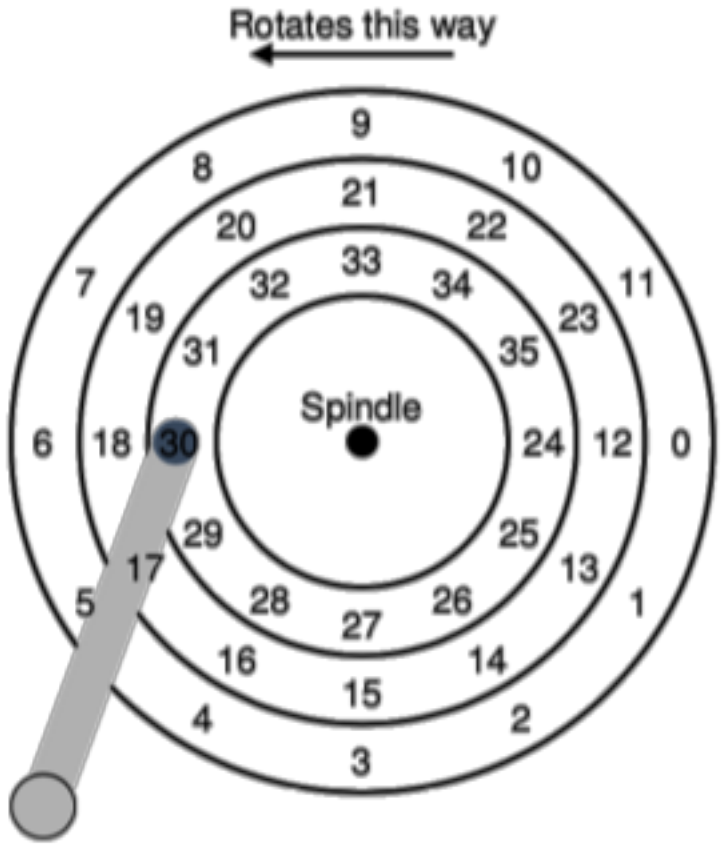
Reading sector **11** requires ***rotation*** and ***seek***





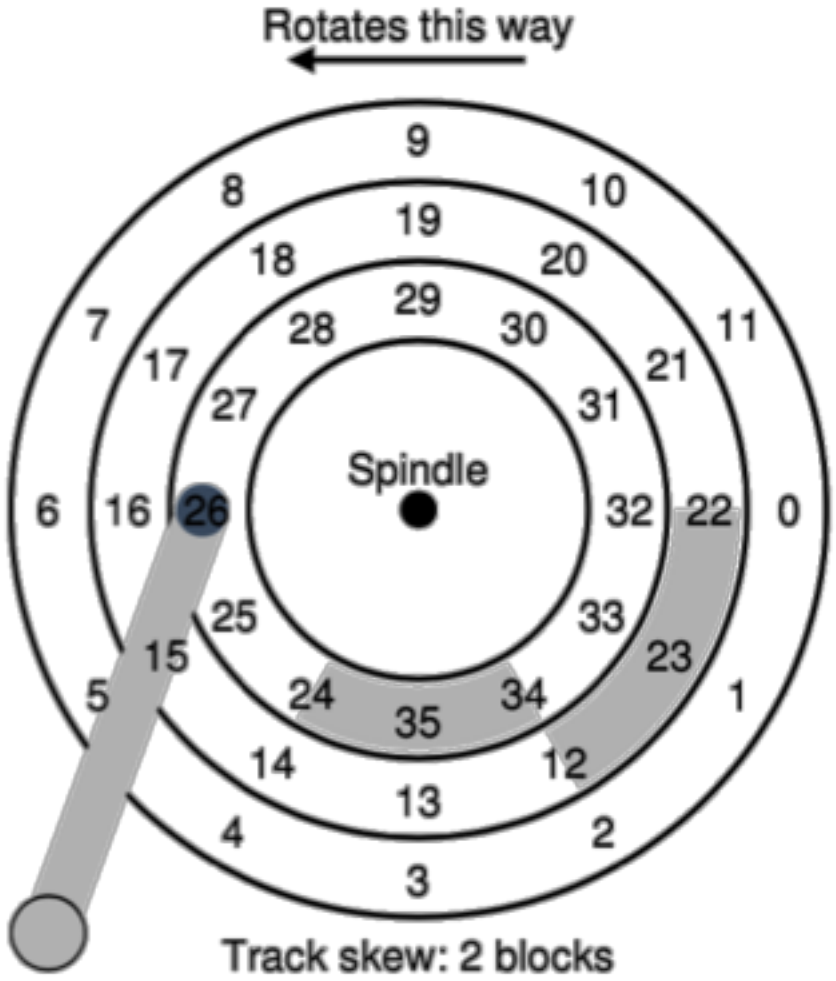
Average delays for random access

- On average half a rotation will be required
  - 10k RPM → full rotation requires 6ms
  - Average rotational latency ~**3ms**
- On average ***one third*** of a full seek is required to reach a random track
    - It would be one half if we always started at the edge, but we are often near the middle, which is close to many tracks.
  - Avg seek time ~= **6ms**
- These are big delays for a computer!
  - Seek time and rotational latencies have not improved much in the past 20 years.



Sequential disk performance

- Reads of sequential sectors are fast because little rotation and seek are required.
- Seek time can be minimized by choosing a ***track skew*** equal to the single-track seek time.
- Disk's maximum sustained throughput is often very high:
  - ~150 MB/sec = 3.4 *micro*seconds/sector
  - This is ~1000 times faster than random access delays due to seek/rotation.

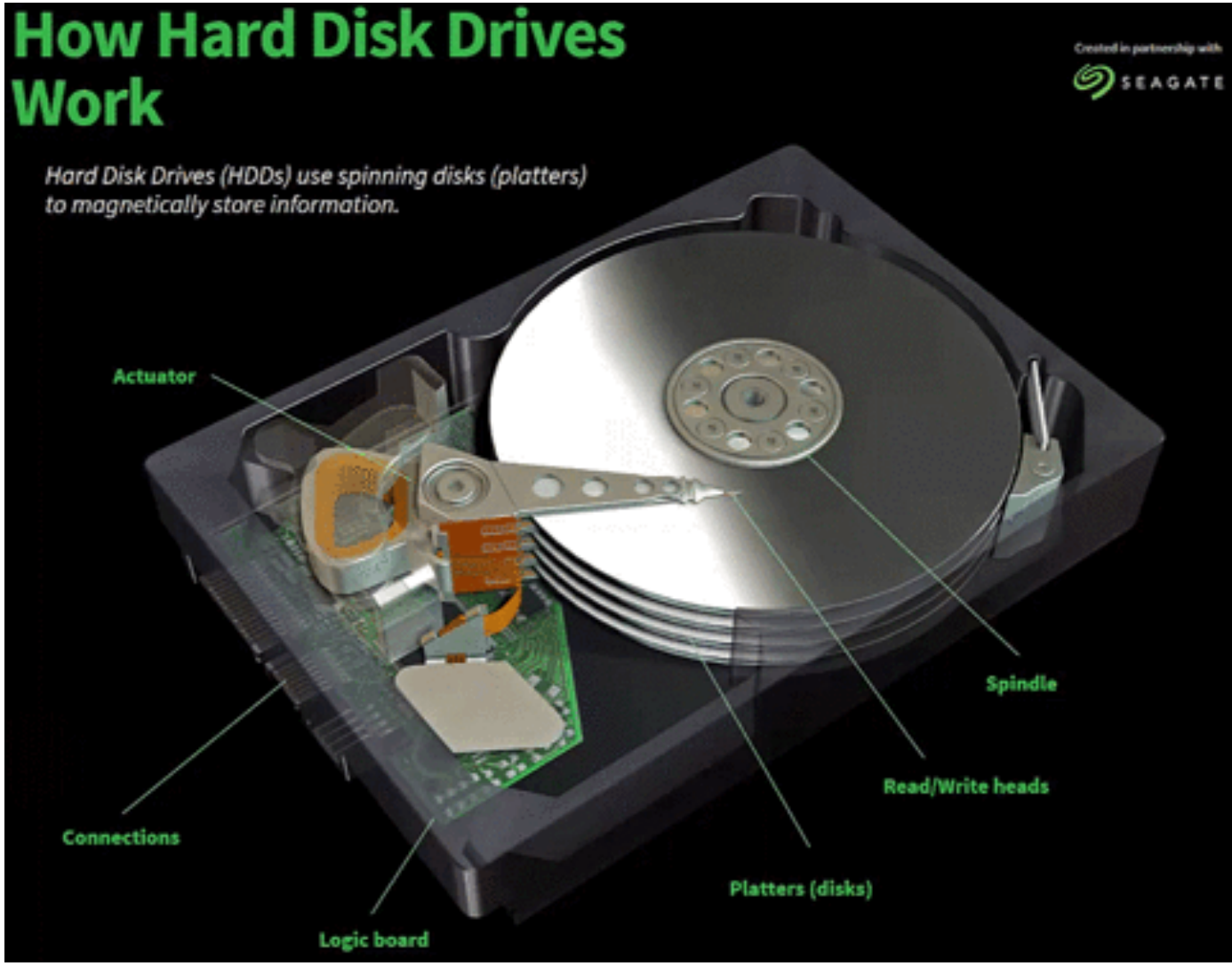


Scheduling disk requests is important

- The ordering of disk requests drastically affects performance.
- There is actually some benefit to delaying requests slightly in order to batch multiple requests and amortize the seek/rotation delays.
  - Ideal performance ~= 150 MB/sec
  - Worst case performance ~= 0.15 MB/sec
- OS does not really know the underlying geometry of the disk: doesn't know current head location nor the precise position of sectors.
  - Past OSes were very careful about disk request scheduling
  - Current practice is to just send many requests to the disk and let the disk schedule them. Disks nowadays are smarter and have large buffers.
- However, OS can at least assume that nearby sector numbers are nearby on the disk, and thus faster to access sequentially.

Real disks

- Disks actually have several double-sided platters, but there is a single arm that holds all the read/write heads.
  - Multiple disk surfaces provide parallelism
- The logic board has a RAM buffer to cache ~128MB of data temporarily.

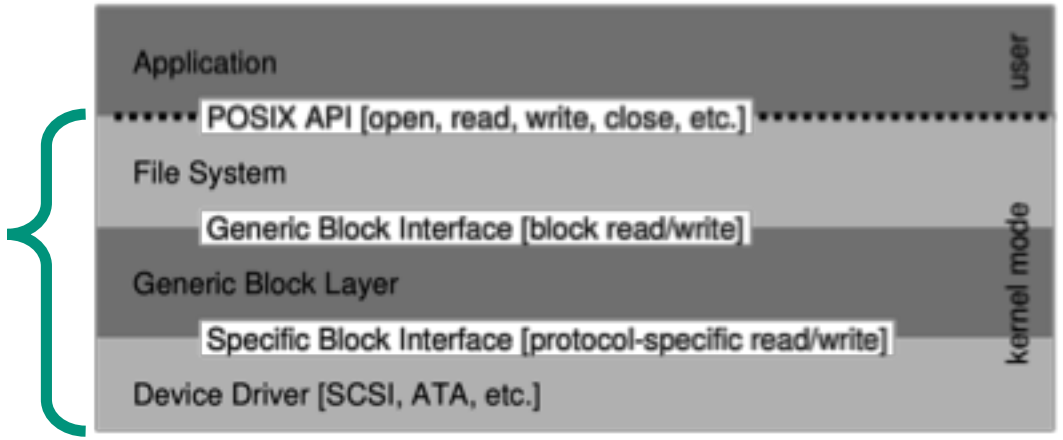


https://animagraffs.com/hard-disk-drive/

Recap– I/O and Disks

- OS interacts with devices by reading/writing ***device registers***
  - Each register has an ***I/O port*** address for in/out instructions, or
  - ***memory-mapped I/O*** uses special physical memory addresses (with `mov`)

- Storage is complex, so kernel functionality is divided into at least three layers:



- Random access to a magnetic disk is 1000x slower than sequential
  - Read head must ***seek*** and disk must ***rotate*** to reach a new sector