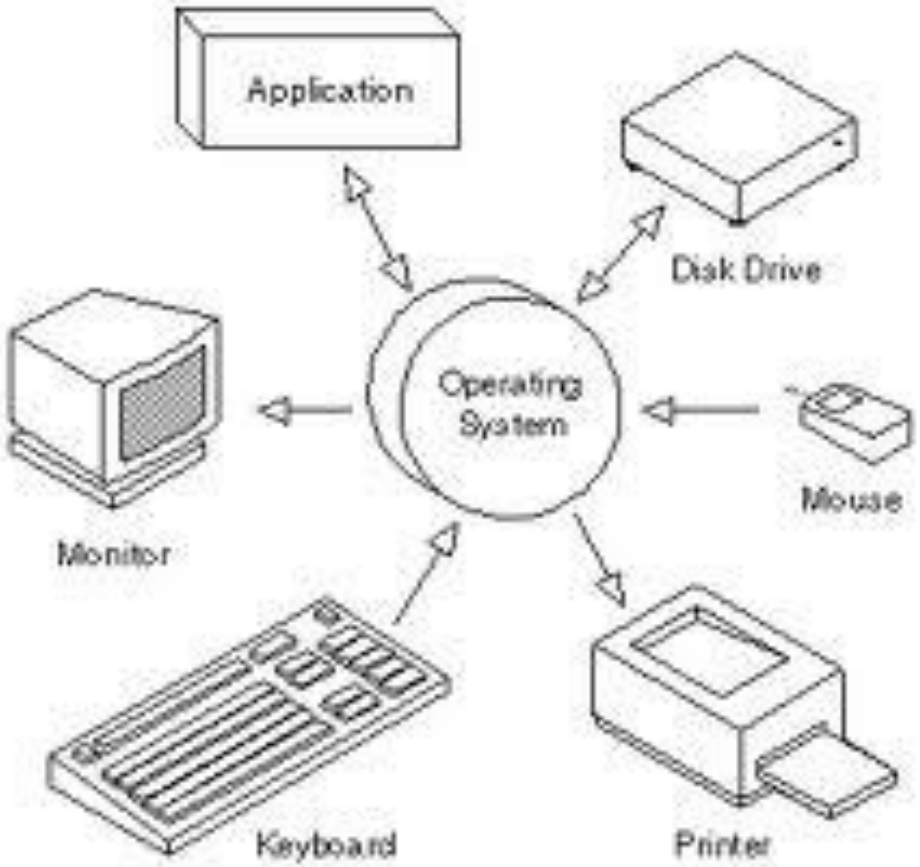
CSCI-GA.2250-001

Operating Systems

Input / Output → I/O

Hubertus Franke
frankeh@cs.nyu.edu


Application
Disk Drive
Operating System
Monitor
Mouse
Keyboard
Printer

Categories of I/O Devices

External devices that engage in I/O with computer systems can be grouped into three categories:

Human readable

- suitable for communicating with the computer user
- printers, terminals, video display, keyboard, mouse

Machine readable

- suitable for communicating with electronic equipment
- disk drives, USB keys, sensors, controllers

Communication

- suitable for communicating with remote devices
- modems, digital line drivers

A Simple Definition

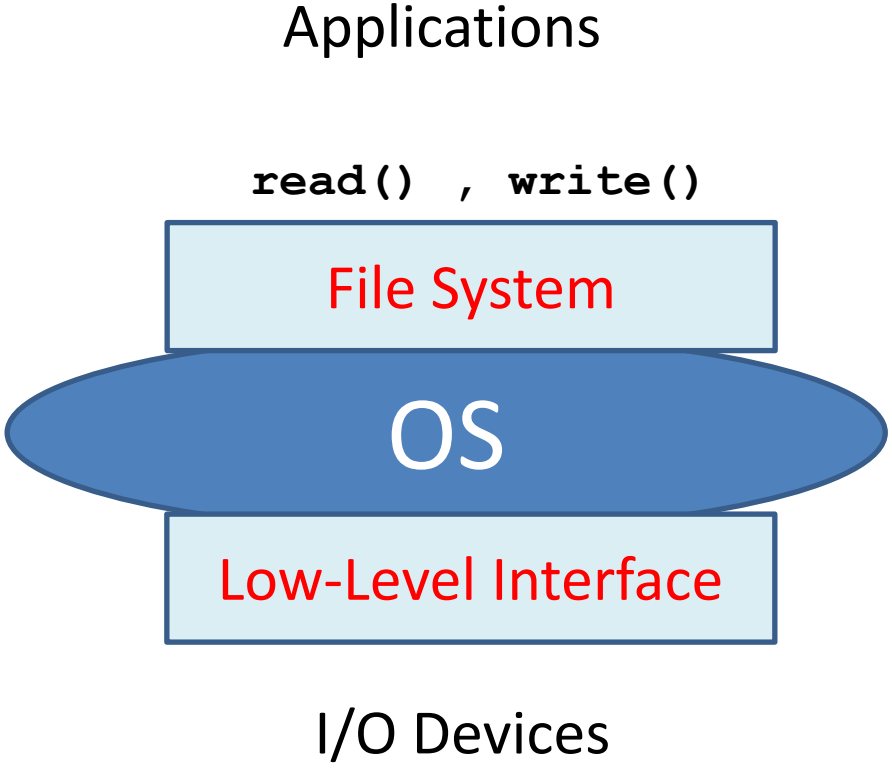
- The main concept of I/O is to move data from/to I/O devices to the processor using some modules (code) and buffers.
- This is the way the processor deals with the outside world.
- Examples: mouse, display, keyboard, disks, network, scanners, speakers, accelerators, GPUs, etc.

The OS and I/O

- The OS controls all I/O devices
  - Issue commands to devices
  - Catch interrupts
  - Handle errors
- Provides an interface between the devices and the rest of the system
- A few exceptions are processor built-in accelerators (encryption, compression engines) that often can be invoked by applications.
  - But think of these less as I/O but as a different kind of functional units in the CPU

Simplified View

- Applications rarely communicate directly with Devices
- Would be too hardware specific
- OS responsible of translating filesystem requests (read/write) into low level device specific I/O requests

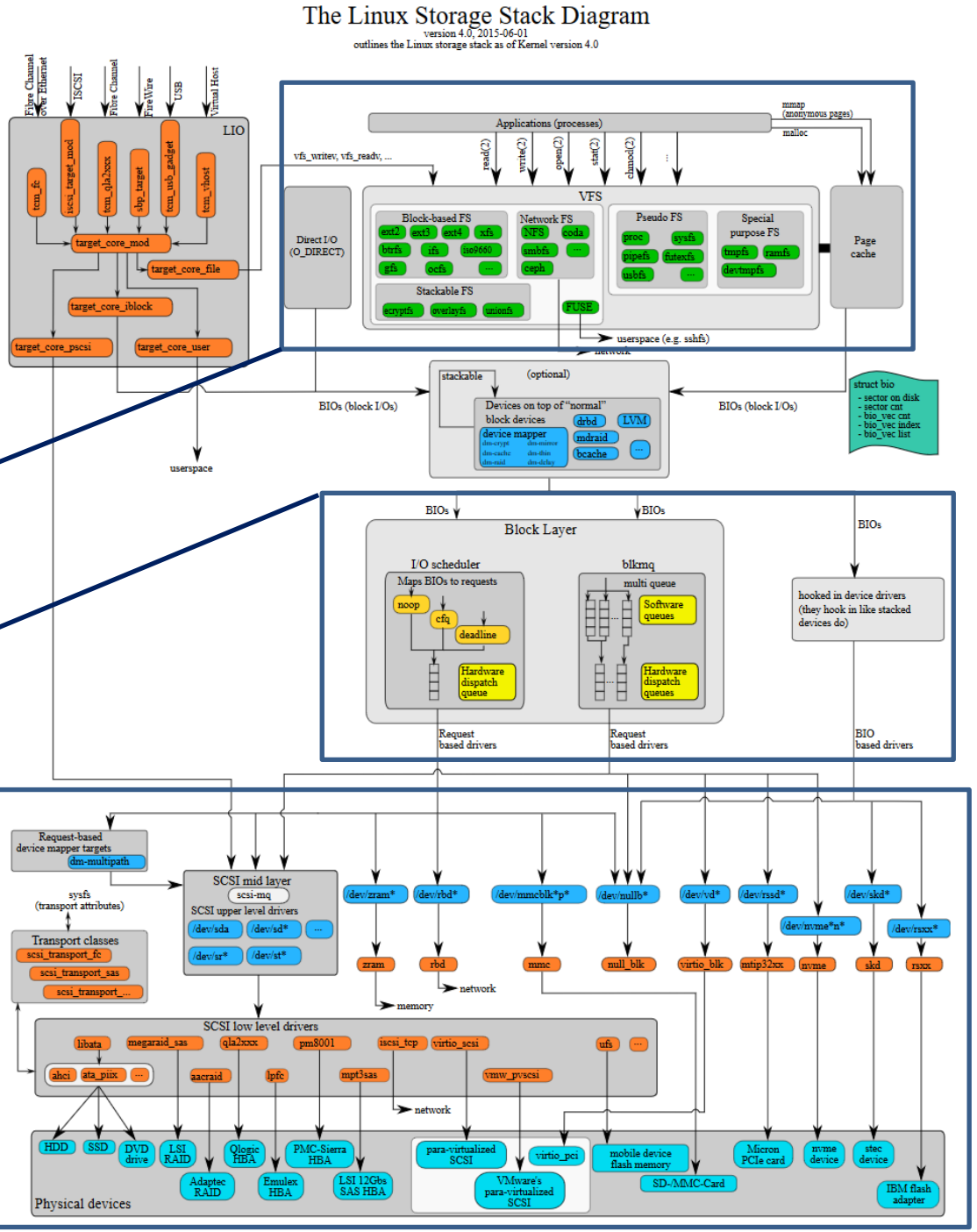


Full Linux I/O Stack

File System

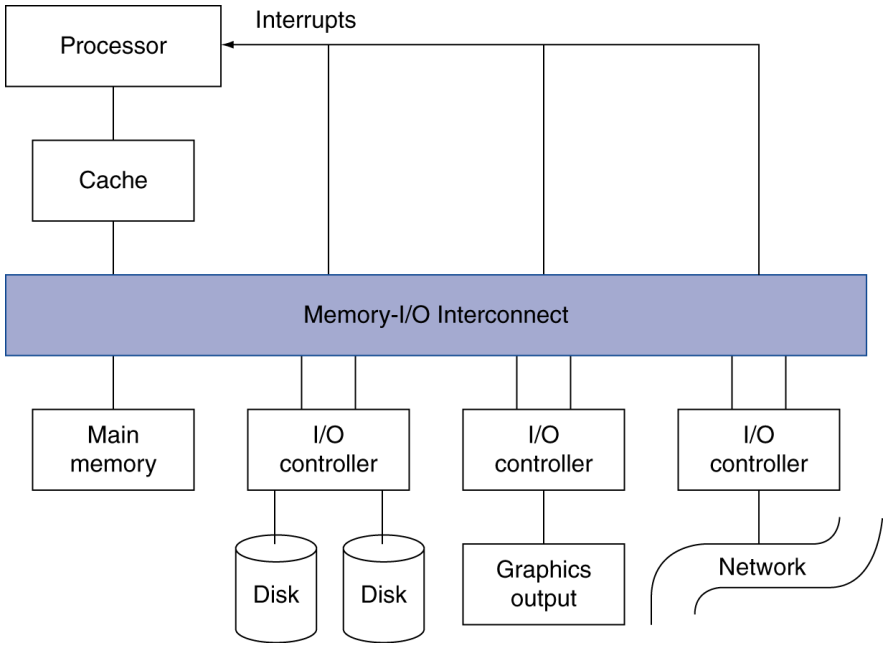
IO Schedulers

Device Drivers



I/O Devices: Challenges

- Very diverse devices
  - behavior (i.e., input vs. output vs. storage)
  - partner (who is at the other end?)
  - data rate
- I/O Design affected by many factors (expandability, resilience)
- Performance:
  - access latency
  - throughput
  - connection between devices and the system
  - the memory hierarchy
  - the operating system
- A variety of different users

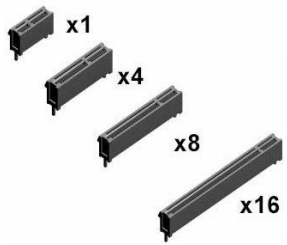


I/O Devices

- Block device
  - Stores information in fixed-size blocks
  - Each block has its own address
  - Transfers in one or more blocks
  - Example: Hard-disks, CD-ROMs, USB sticks
- Character device
  - Delivers or accepts stream of character
  - Is not addressable
  - Example: mice, printers, network interfaces

Devices (Feed and Speed) !!!

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner	400 KB/sec
Digital camcorder	3.5 MB/sec
802.11g Wireless	6.75 MB/sec
52x CD-ROM	7.8 MB/sec
Fast Ethernet	12.5 MB/sec
Compact flash card	40 MB/sec
FireWire (IEEE 1394)	50 MB/sec
USB 2.0	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
SATA disk drive	300 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec



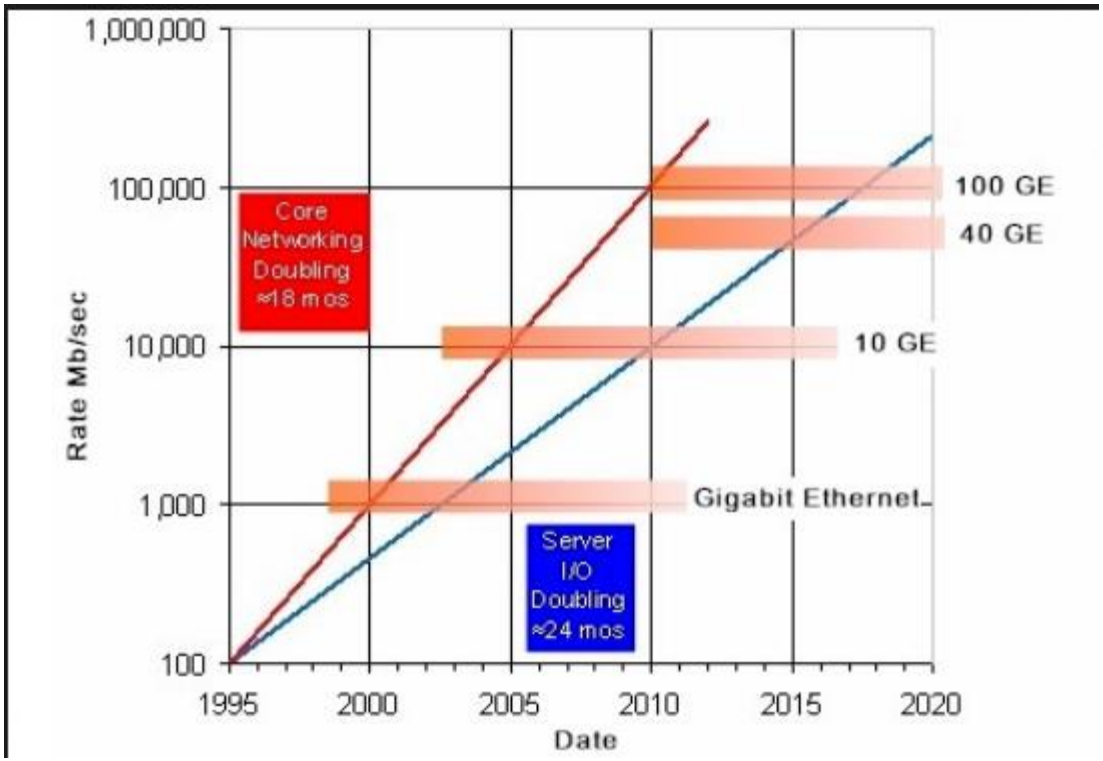
peripheral component interconnect express

PCIe Bandwidth & Frequency

Year	Bandwidth	Frequency/Speed
1992	133MB/s (32 bit simplex)	33 Mhz (PCI)
1993	533MB/s (64 bit simplex)	66 Mhz (PCI 2.0)
1999	1.06GB/s (64 bit simplex)	133 Mhz (PCI-X)
2002	2.13GB/s (64 bit simplex)	266 Mhz (PCI-X 2.0)
2002	8GB/s (x16 duplex)	2.5 GHz (PCIe 1.x)
2006	16GB/s (x16 duplex)	5.0 GHz (PCIe 2.x)
2010	32GB/s (x16 duplex)	8.0 GHz (PCIe 3.x)
2017	64GB/s (x16 duplex)	16.0 GHz (PCIe 4.0)
2019	128GB/s (x16 duplex)	32.0 GHz (PCIe 5.0)

Devices change !!!

- Network speed doubling every 18-24 month
- Approaching 400Gbps (we already have 200)



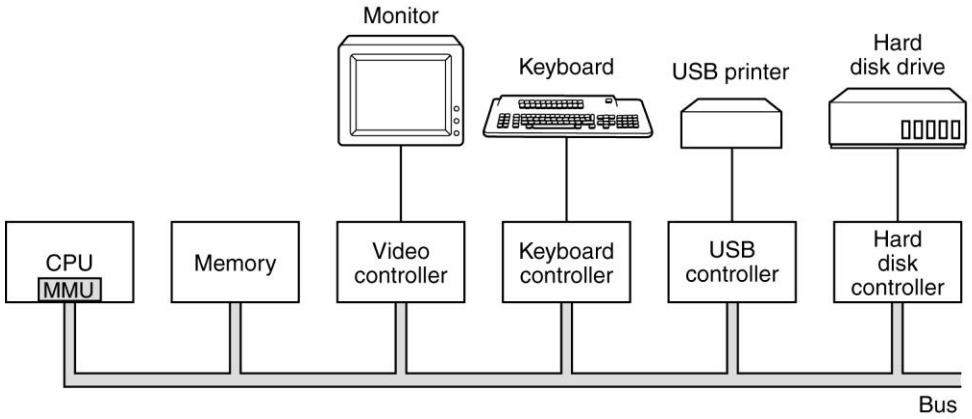
I/O Units

"Mechanical" component *

Electronic Component

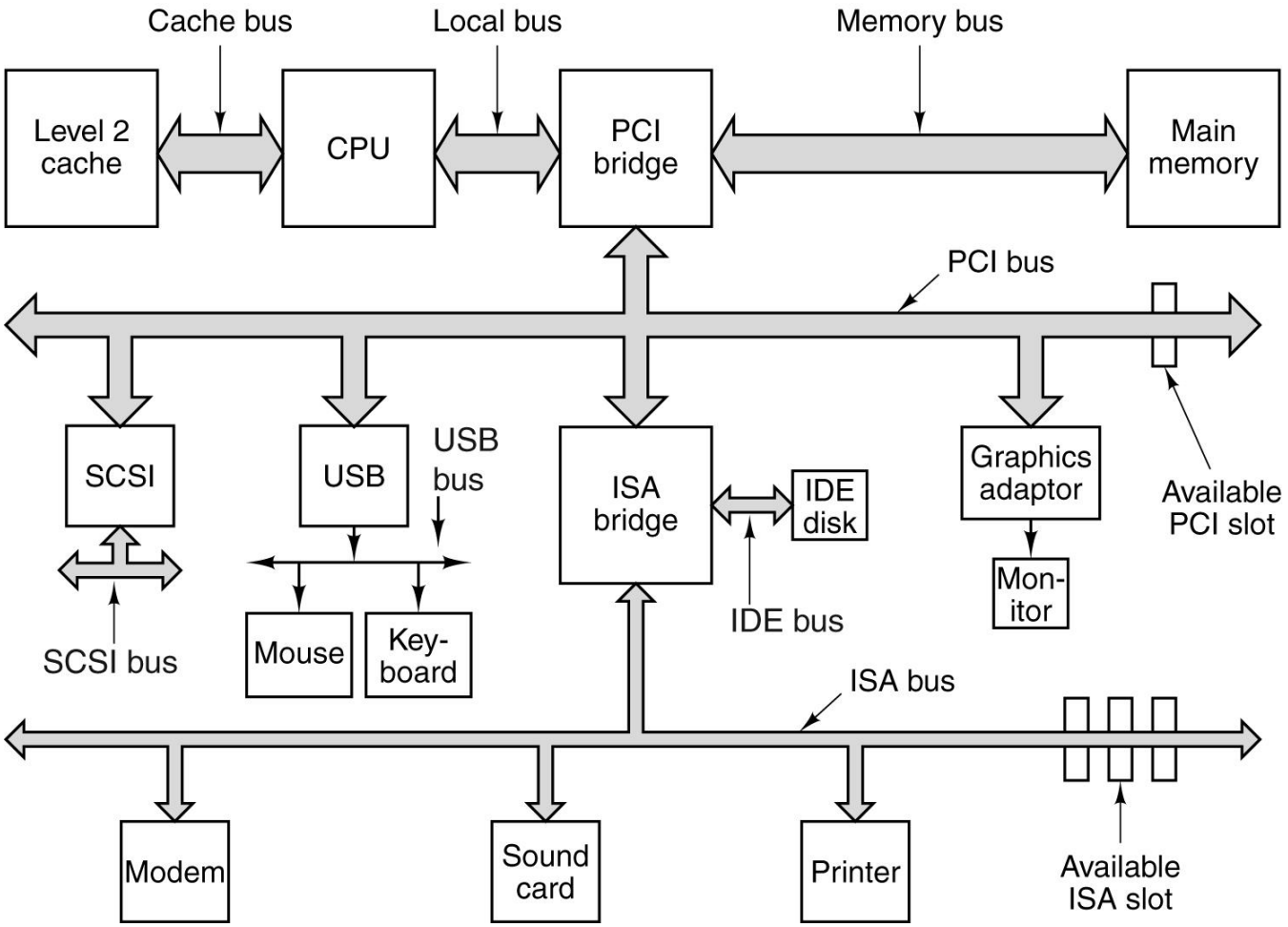
The Device Itself

Device Controller



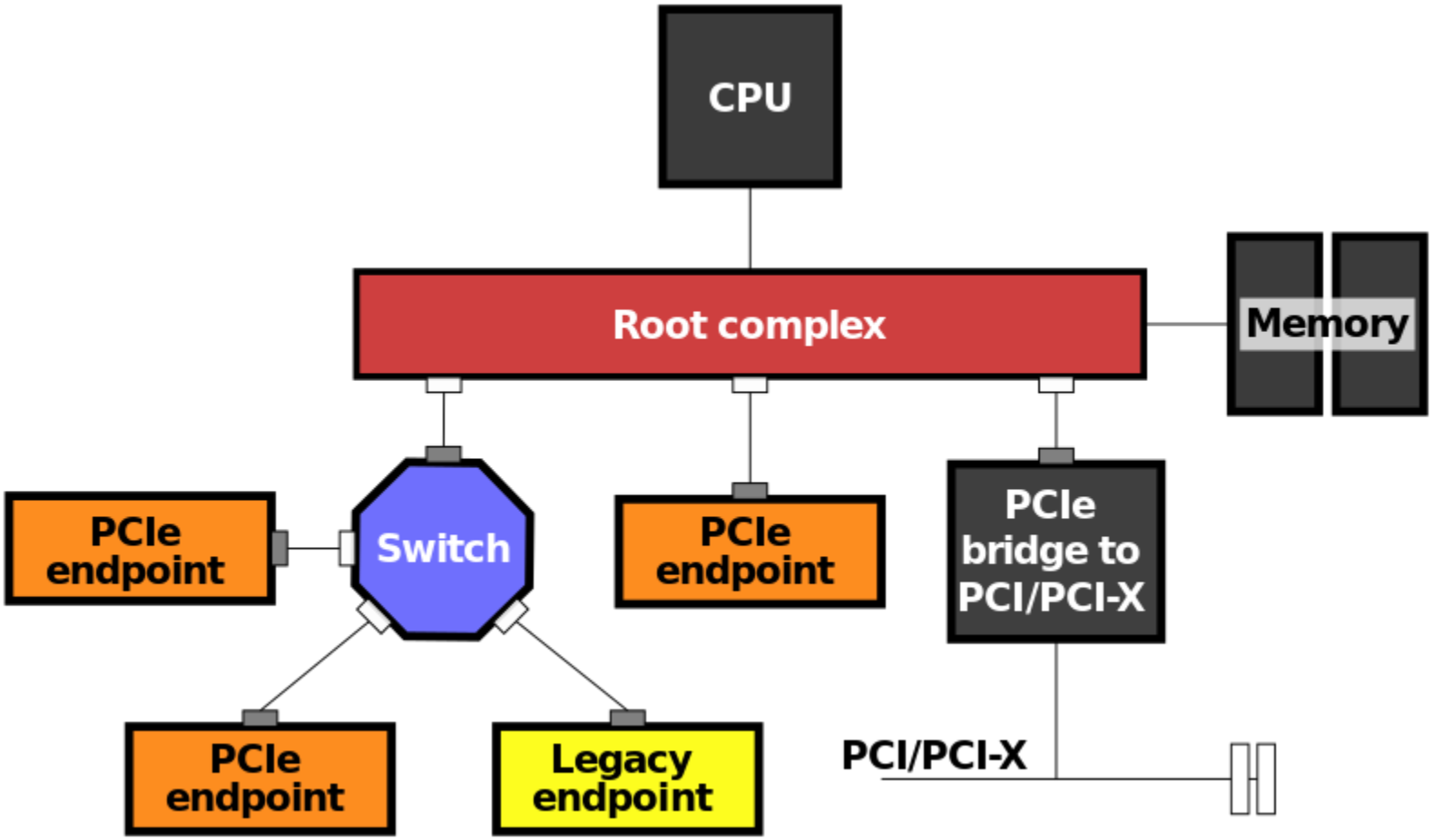
* Many devices don't have mechanical components anymore

A more realistic hierarchy of I/O



PCI-e

Peripheral Component Interconnect Express



Standard means to connect modern devices to the system

Controller and Device

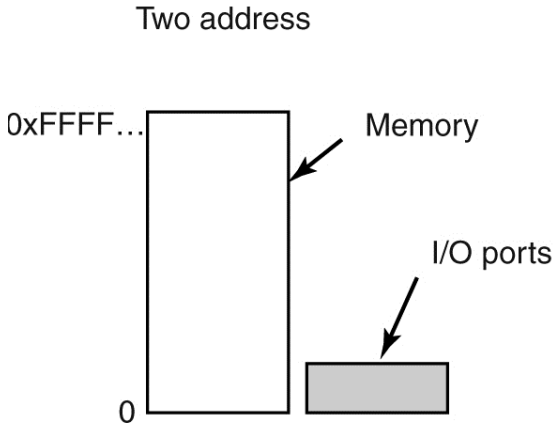
- Each controller has few registers used to communicate with CPU
- By writing/reading into/from those registers, the OS can control the device.
- (1) There are also data buffers in the device that can be read from or written to by the OS
  - Location/address of buffer
  - Location/address of I/O "object"
  - Size of data to transfer
- (2) cmd buffers to start/stop operations

How does CPU communicate with control registers and data buffers?

Two main approaches:

- I/O port space
- Memory-mapped I/O

I/O Port Space



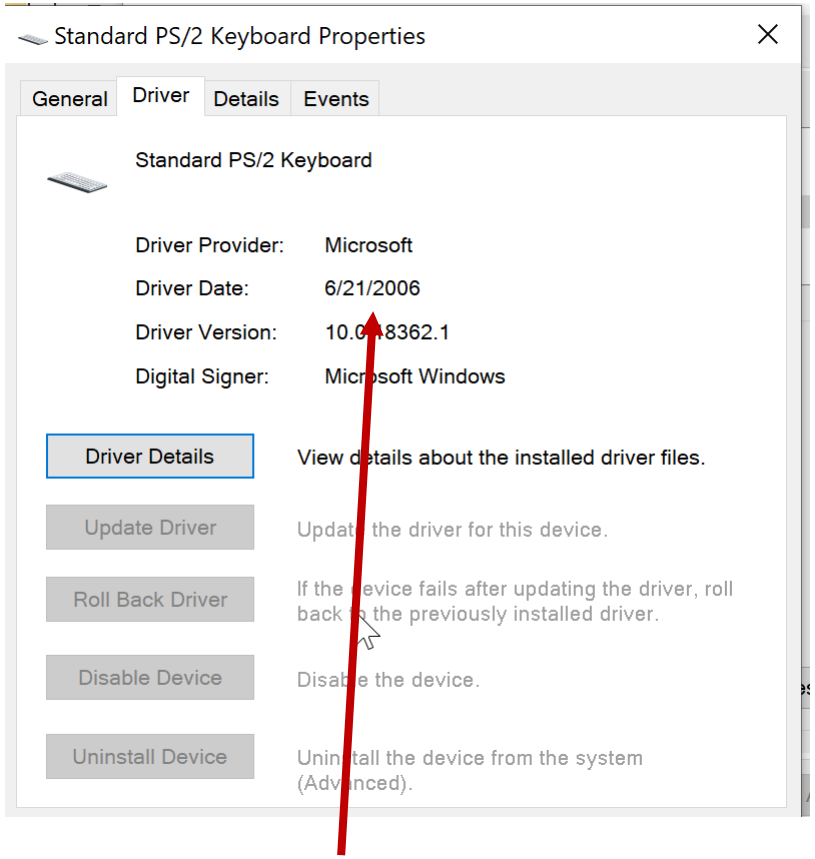
- Each control register is assigned an I/O port number
- The set of all I/O ports form the I/O port space
- I/O port space is protected → only kernel can access

  execute privileged instructions:
  - `in portnum, target`
  - `out portnum, source`

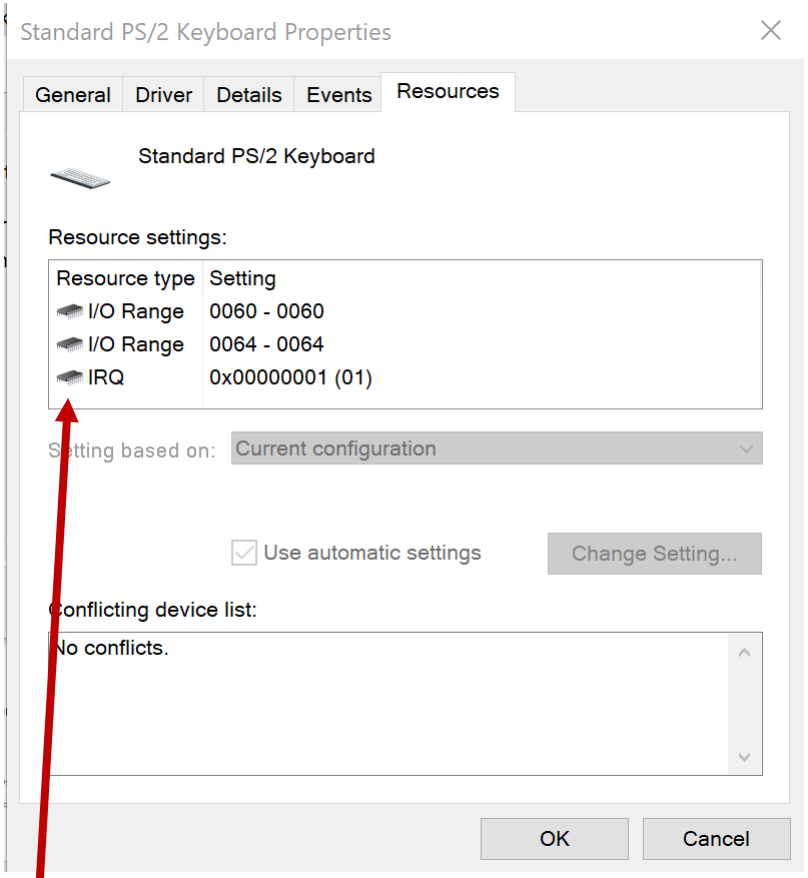
Opcode	Mnemonic	Description
E4 ib	IN AL,imm8	Input byte from imm8 I/O port address into AL.
E5 ib	IN AX,imm8	Input byte from imm8 I/O port address into AX.
E5 ib	IN EAX,imm8	Input byte from imm8 I/O port address into EAX.
EC	IN AL,DX	Input byte from I/O port in DX into AL.
ED	IN AX,DX	Input word from I/O port in DX into AX.
ED	IN EAX,DX	Input doubleword from I/O port in DX into EAX.

- Classical x86 way
- Largely done only for "legacy" devices

Example Win Keyboard



Talk about legacy software



two io-ports and one interrupt

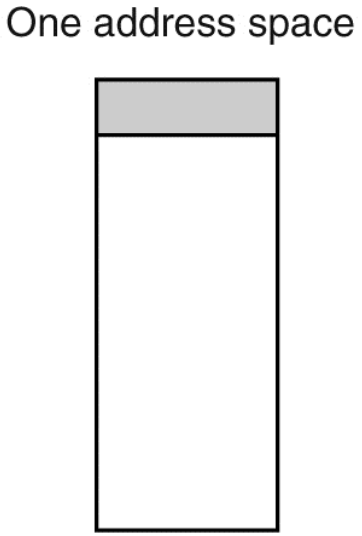
Example: Serial and Parallel Ports

COM Port #	IRQ	I/O Port Address
1	4	3F8-3FFh
2	3	2F8-2FFh
3	4	3E8-3EFh
4	3	2E8-2EFh

LPT Port #	IRQ	I/O Port Address Range
LPT1	7	378-37Fh or 3BC-38Fh
LPT2	5	278-27Fh or 378-37Fh
LPT3	5	278-27Fh

Memory-Mapped I/O

- Map control registers into the memory space
- Each control register is assigned a unique physical memory address
- These physaddr are mapped through the pagetable into the kernel address space and can now be operated on via memory load/store
- Load/stores to the memory are "snooped" on by device and acted upon, PCI device is told the range to snoop
- Every modern architecture basically does it this way



Advantages of Memory-Mapped I/O

- Device drivers can be written entirely in C (since no special instructions are needed)
- No special protection is needed from OS, just refrain from putting that portion of the address space in any user's virtual address space
- Every instruction that can reference memory can also reference control registers

PCI-Configuration Space

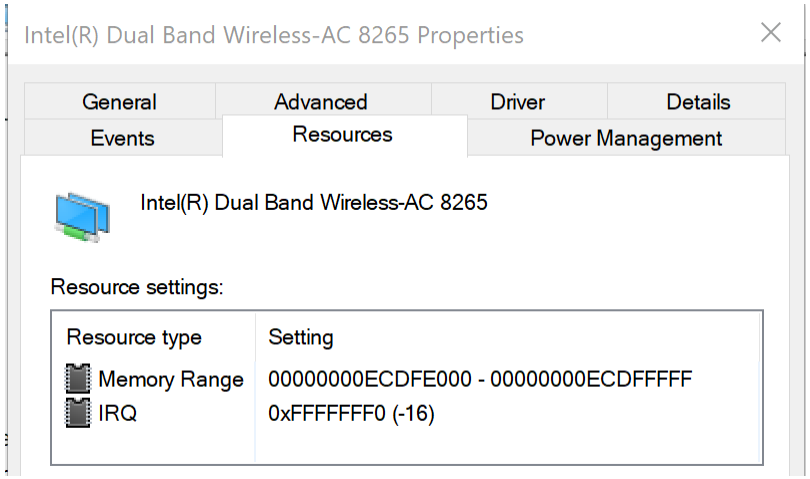
Table 2-22: **Type 0 PCI Configuration Space Header**

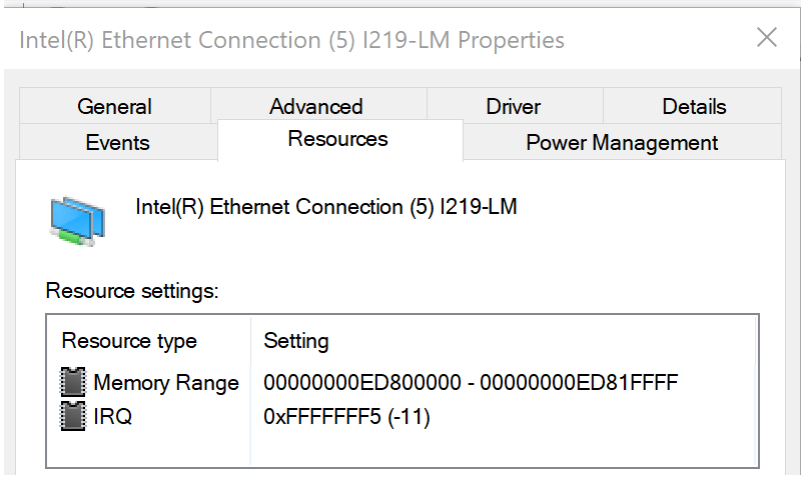
31	16	15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Rev ID	08h
BIST	Header	Lat Timer	Cache Ln	0Ch
Base Address Register 0				10h
Base Address Register 1				14h
Base Address Register 2				18h
Base Address Register 3				1Ch
Base Address Register 4				20h
Base Address Register 5				24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved			CapPtr	34h
Reserved				38h
Max Lat	Min Gnt	Intr Pin	Intr Line	3Ch

```
struct pci_config {
      uint16 devid;
      uint16 vendorid;
      :
      uint32 bar0;
      :
      uint32 bar1;
      :
      uint8 maxlat;
      :
      uint8 intr;
};
```

Standard PCI Control Space

Examples for network devices





Range: 0x2000 == 8KB

```
struct wlac_8265 {
        struct pci_config pcie_config;
        <specific structure for 8265 dev>
};
```

```
struct wlac_8265 *wldev =
    (struct wlac_8265*) kern_addr(0xECDFE000);
wldev->memberelem =  …..
```

Range: 0x20000 == 128KB

```
struct lan_I219 {
        struct pci_config pcie_config;
        <specific structure for I219ev>
};
```

virtual kernel address

physical bus address

```
struct lan_I219 *landev =
    (struct lan_I219*) kern_addr(0xED800000);
landev->memberelem = …..
```

Inspecting your PCI subsystem

What devices are connected to my system ?

```
[frankeh@linserv1 ~]$ lspci | grep -v "^00"
01:00.0 Ethernet controller: Broadcom Inc. and subsidiaries NetXtreme II BCM5709 Gigabit Ethernet (rev 20)
01:00.1 Ethernet controller: Broadcom Inc. and subsidiaries NetXtreme II BCM5709 Gigabit Ethernet (rev 20)
02:00.0 Ethernet controller: Broadcom Inc. and subsidiaries NetXtreme II BCM5709 Gigabit Ethernet (rev 20)
02:00.1 Ethernet controller: Broadcom Inc. and subsidiaries NetXtreme II BCM5709 Gigabit Ethernet (rev 20)
03:00.0 PCI bridge: PLX Technology, Inc. PEX 8624 24-lane, 6-Port PCI Express Gen 2 (5.0 GT/s) Switch [ExpressLane] (rev bb)
04:00.0 PCI bridge: PLX Technology, Inc. PEX 8624 24-lane, 6-Port PCI Express Gen 2 (5.0 GT/s) Switch [ExpressLane] (rev bb)
04:01.0 PCI bridge: PLX Technology, Inc. PEX 8624 24-lane, 6-Port PCI Express Gen 2 (5.0 GT/s) Switch [ExpressLane] (rev bb)
04:04.0 PCI bridge: PLX Technology, Inc. PEX 8624 24-lane, 6-Port PCI Express Gen 2 (5.0 GT/s) Switch [ExpressLane] (rev bb)
04:05.0 PCI bridge: PLX Technology, Inc. PEX 8624 24-lane, 6-Port PCI Express Gen 2 (5.0 GT/s) Switch [ExpressLane] (rev bb)
05:00.0 RAID bus controller: Broadcom / LSI MegaRAID SAS 2108 [Liberator] (rev 05)
0a:03.0 VGA compatible controller: Matrox Electronics Systems Ltd. MGA G200eW WPCM450 (rev 0a)
20:00.0 Host bridge: Advanced Micro Devices, Inc. [AMD/ATI] RD890 Northbridge only dual slot (2x8) PCI-e GFX Hydra part (rev 02)
20:00.2 IOMMU: Advanced Micro Devices, Inc. [AMD/ATI] RD890S/RD990 I/O Memory Management Unit (IOMMU)
20:02.0 PCI bridge: Advanced Micro Devices, Inc. [AMD/ATI] RD890/RD9x0/RX980 PCI to PCI bridge (PCI Express GFX port 0)
20:03.0 PCI bridge: Advanced Micro Devices, Inc. [AMD/ATI] RD890/RD9x0 PCI to PCI bridge (PCI Express GFX port 1)
20:0b.0 PCI bridge: Advanced Micro Devices, Inc. [AMD/ATI] RD890/RD990 PCI to PCI bridge (PCI Express GFX2 port 0)
```

```
[frankeh@linserv1 ~]$ lspci -n | grep -v "^00"
01:00.0 0200: 14e4:1639 (rev 20)
01:00.1 0200: 14e4:1639 (rev 20)
02:00.0 0200: 14e4:1639 (rev 20)
02:00.1 0200: 14e4:1639 (rev 20)
03:00.0 0604: 10b5:8624 (rev bb)
04:00.0 0604: 10b5:8624 (rev bb)
04:01.0 0604: 10b5:8624 (rev bb)
04:04.0 0604: 10b5:8624 (rev bb)
04:05.0 0604: 10b5:8624 (rev bb)
05:00.0 0104: 1000:0079 (rev 05)
0a:03.0 0300: 102b:0532 (rev 0a)
20:00.0 0600: 1002:5a12 (rev 02)
20:00.2 0806: 1002:5a23
20:02.0 0604: 1002:5a16
20:03.0 0604: 1002:5a17
20:0b.0 0604: 1002:5a1f
```

Add item
Discuss
Help
ID syntax

The PCI ID Repository

The home of the pci.ids file

Log in

Main -> PCI Devices -> **Vendor 14e4**

Name: Broadcom Inc. and subsidiaries

Discussion

Name: Broadcom Corporation

mj

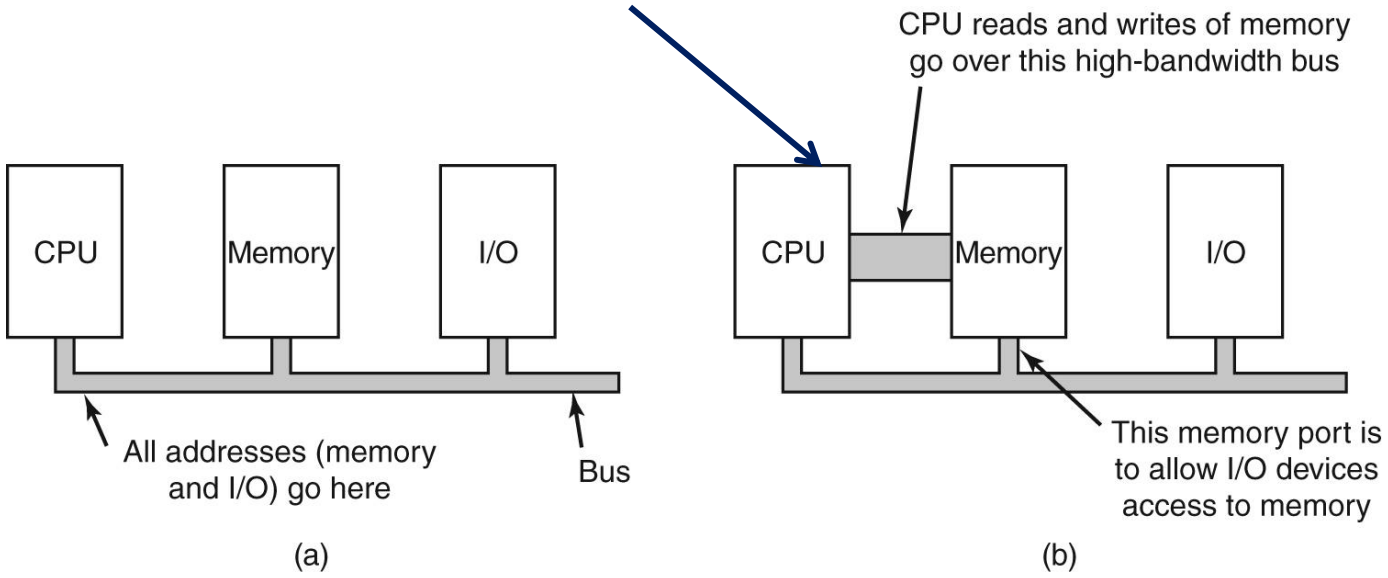
2002-08-14 18:16:25

Executed on “linserv1.cims.nyu.edu”

Something to be conscious about with Memory-Mapped I/O

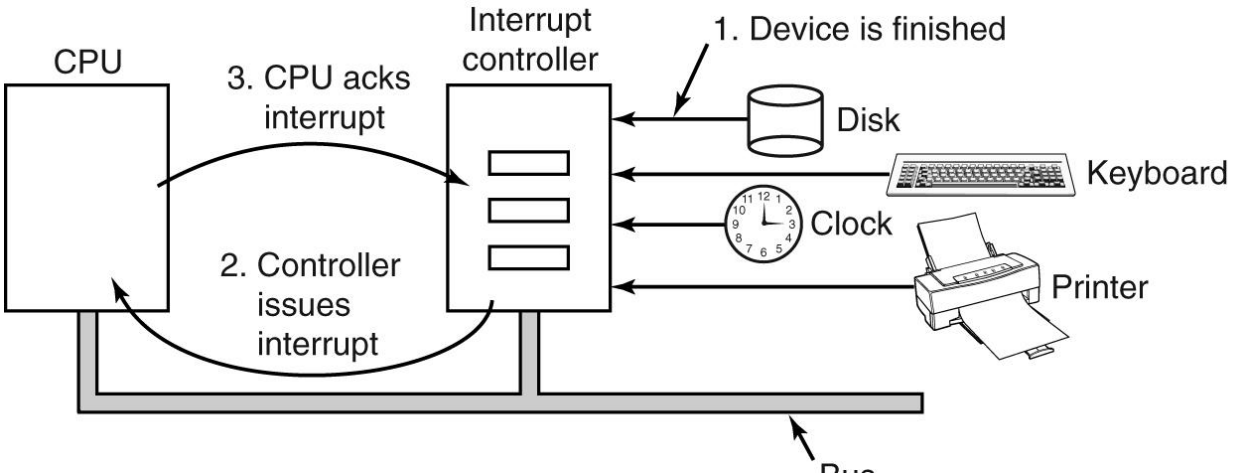
- Caching a device control register can be disastrous, after all it is just a "memory load"

- Remember the "cache disabled bit" in PTE ?

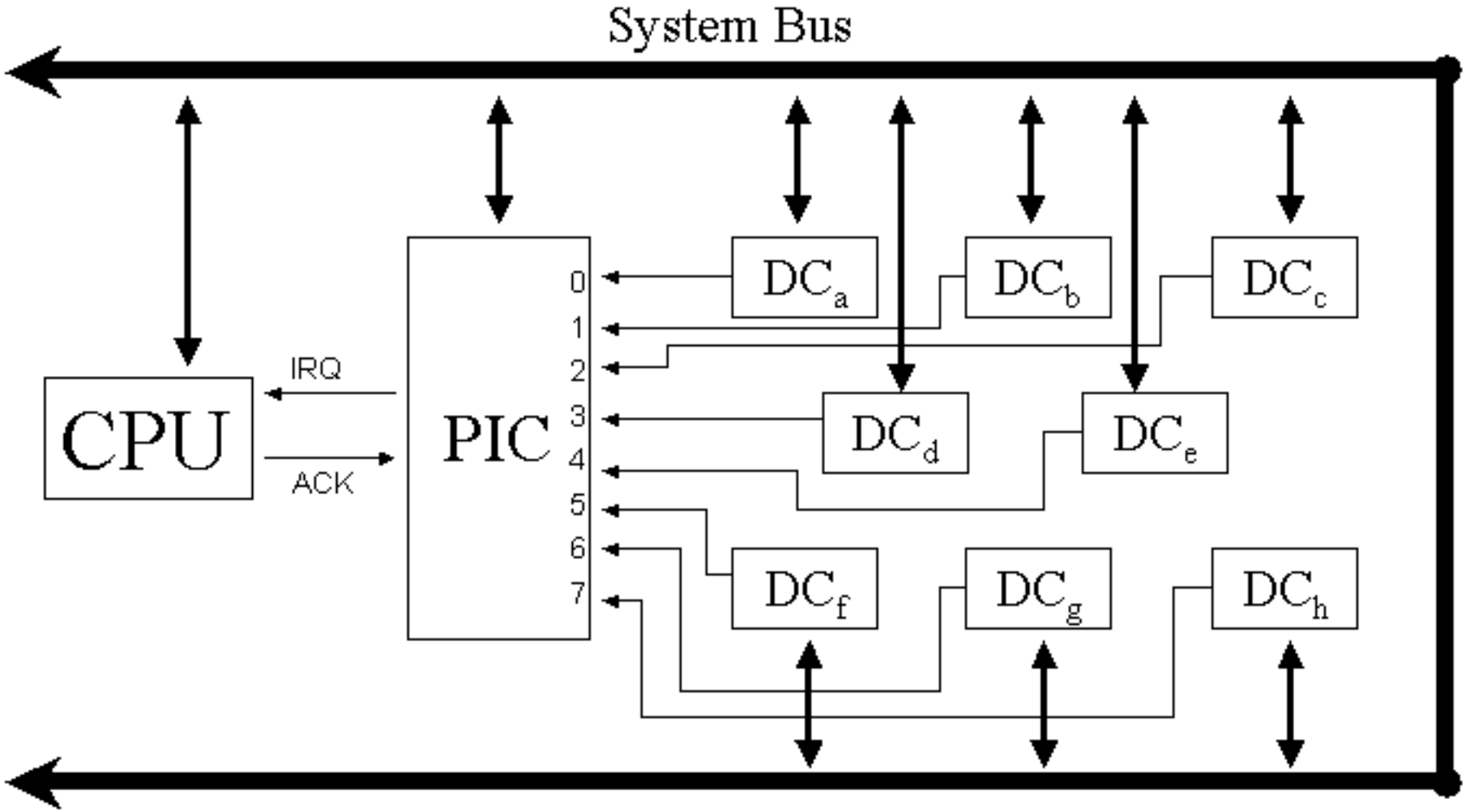


Interrupts

- When an I/O device has finished the work given to it, it causes an interrupt.
- More generally, anytime a device wants attention it causes an interrupt.



(Advanced) Programmable Interrupt Controllers [APIC]



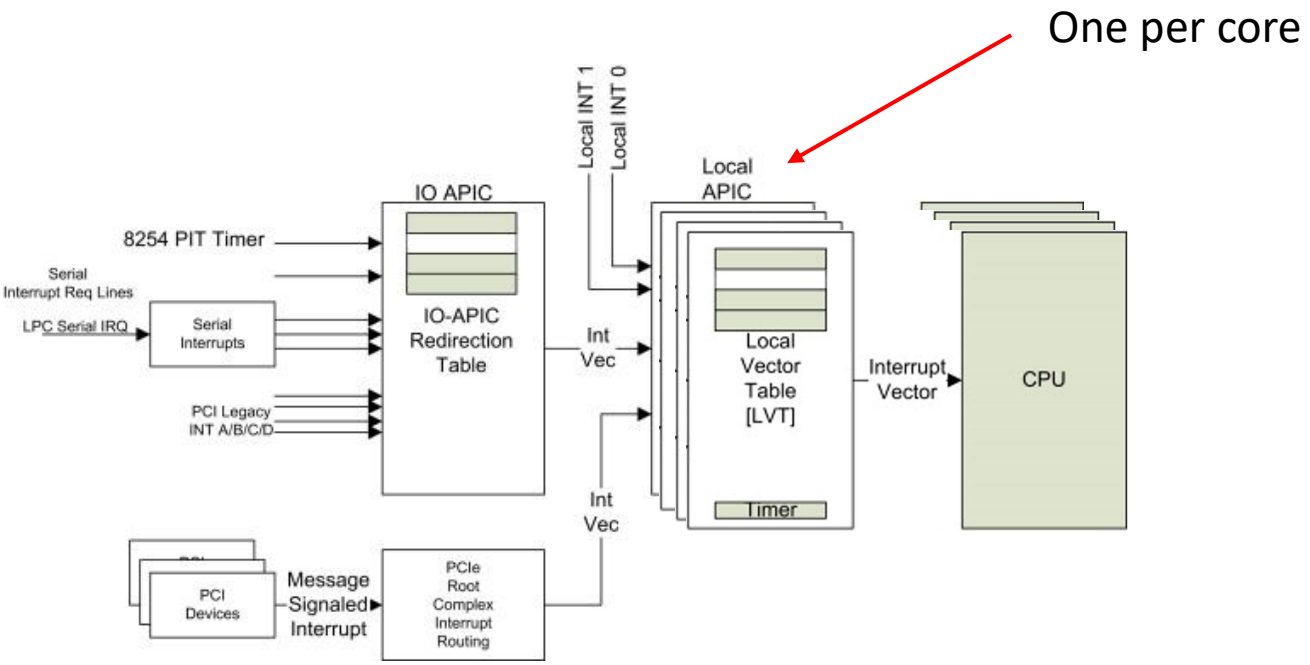
DC == Device Controller

Message Signaled Interrupts

- Hardware lines from device to APIC too limited in numbers → pin count restrictions
- MSI allows the device to write a small amount of interrupt-describing data to a special memory-mapped I/O address
- Writing this "value" to a particular memory address triggers an interrupt.
- The PCI chipset delivers the corresponding interrupt to a processor.
- PCI-2.2 defined MSI with upto 32 interrupts
- PCI-3.0 defines MSI-X with upto 2048 interrupts
- Highly programmable

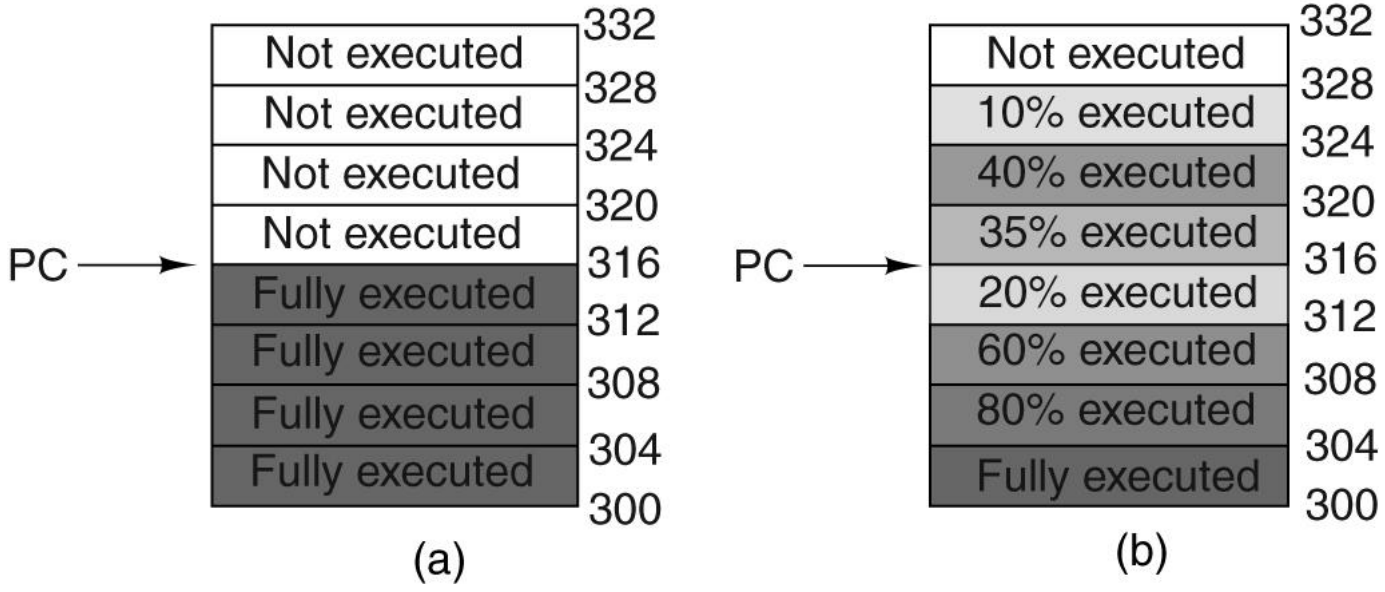
Complete Interrupt System

- Split Architecture
  - Global: IO APIC and PCI-based MSI Interrupt Routing
  - Local: per core Local APIC



Issues with Interrupt processing

- Architectures are pipelined and OOO
- What does it then mean to "switch the next instruction to enter the kernel at __entry ?



Precise Interrupt

Imprecise Interrupt

Precise Interrupts

- Makes handling interrupts much simpler
- Has 4 properties
  - The program counter (PC) is saved in known place (typically a special register only accessible through kernel mode)
  - All instructions before the one pointed by PC have fully executed
  - No instruction beyond the one pointed by PC has been executed (or has any sideeffects)
  - The execution state of the instruction pointed to by the PC is known
- Hugely important with out of order / superscalar procesors

I/O Software

- Device independence
- Uniform naming
- Error handling
  - Should be handled as close to the hardware as possible
- Synchronous vs asynchronous (interrupt-driven)
- Buffering
- Sharable versus dedicated devices

Three Ways of Performing I/O

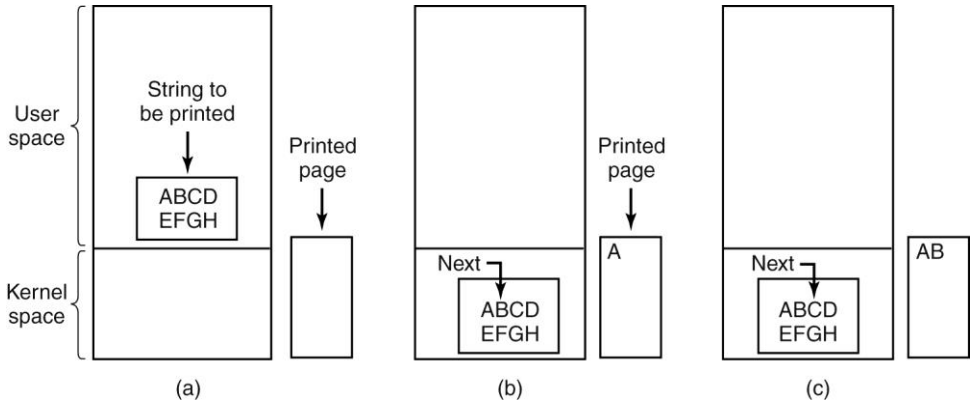
- Programmed I/O
- Interrupt-driven I/O
- I/O Using DMA

Programmed I/O

- CPU does all the work
- Busy-waiting (polling)

Example:

```
copy_from_user(buffer, p, count);                    /* p is the kernel buffer */
for (i = 0; i < count; i++) {                        /* loop on every character */
        while (*printer_status_reg != READY) ;       /* loop until ready */
        *printer_data_register = p[i];               /* output one character */
}
return_to_user( );
```



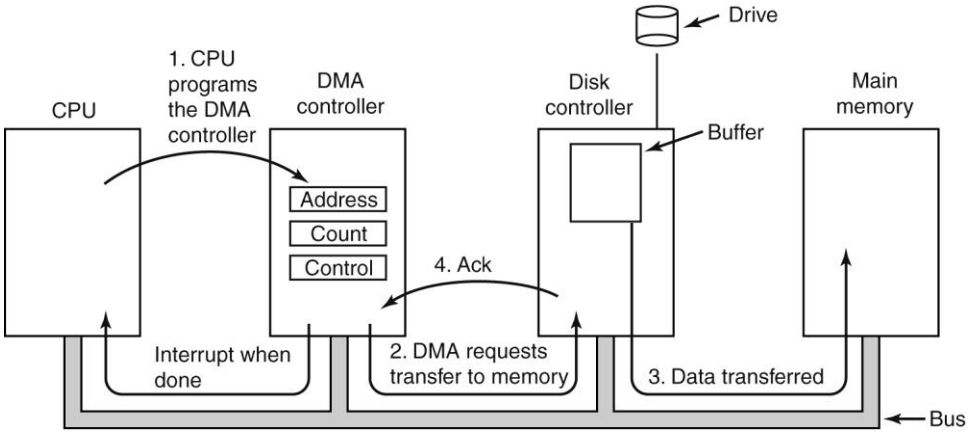
Interrupt-Driven I/O

- Waiting for a device to be ready, the process is blocked and another process is scheduled.
- When the device is ready it raises an interrupt.
- Then data is copied by CPU as previous (avoids polling)
- Upon completion of the transfer the blocked process can be made ready again.

Direct Memory Access (DMA)

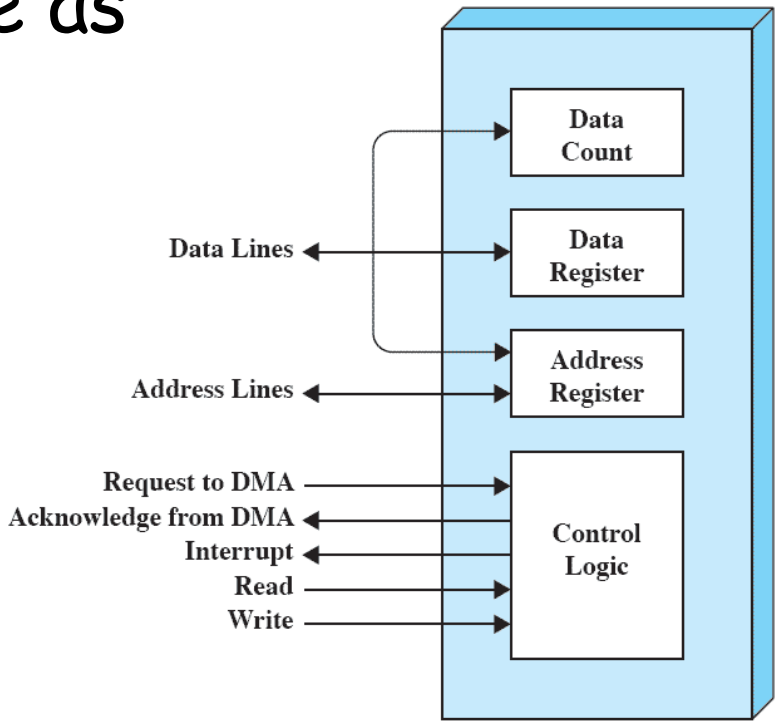
- It is not efficient for the CPU to request data from I/O one byte/word at a time

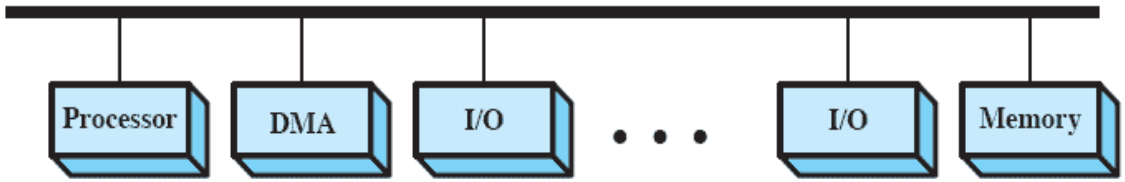
- DMA controller has access to the system bus independent of the CPU



I/O Using DMA

- DMA does the work instead of the CPU
- Think of DMA engine as a simple "copy-core"
- Let the DMA do its work and then use interrupts to notify CPU

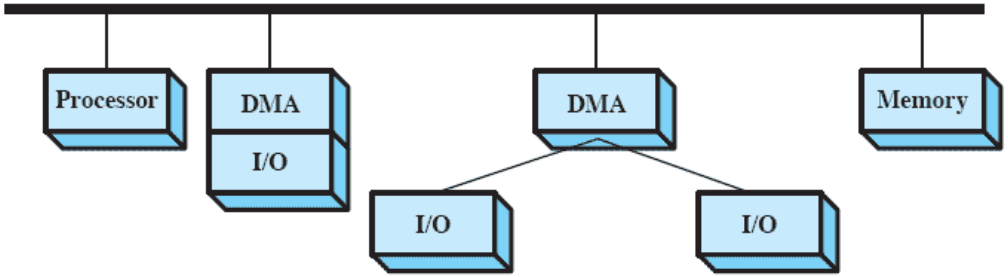




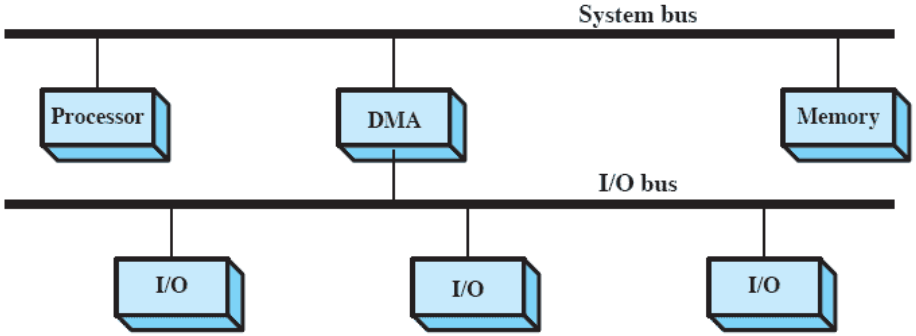
(a) Single-bus, detached DMA

DMA

Alternative



(b) Single-bus, Integrated DMA-I/O

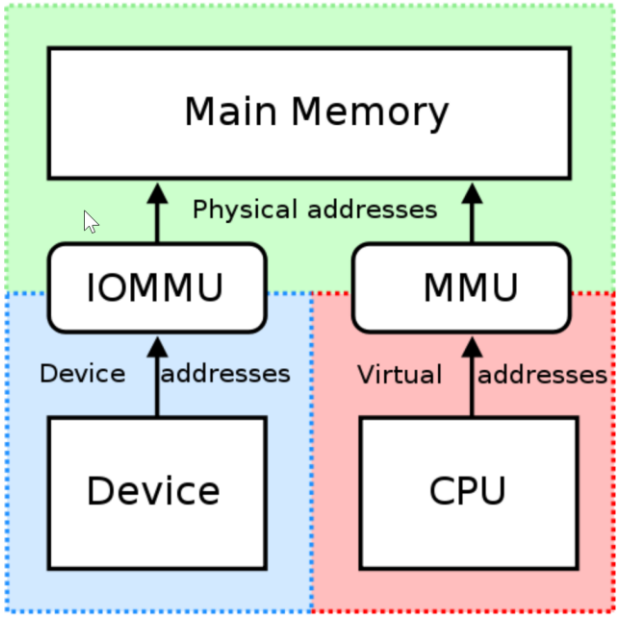


(c) I/O bus

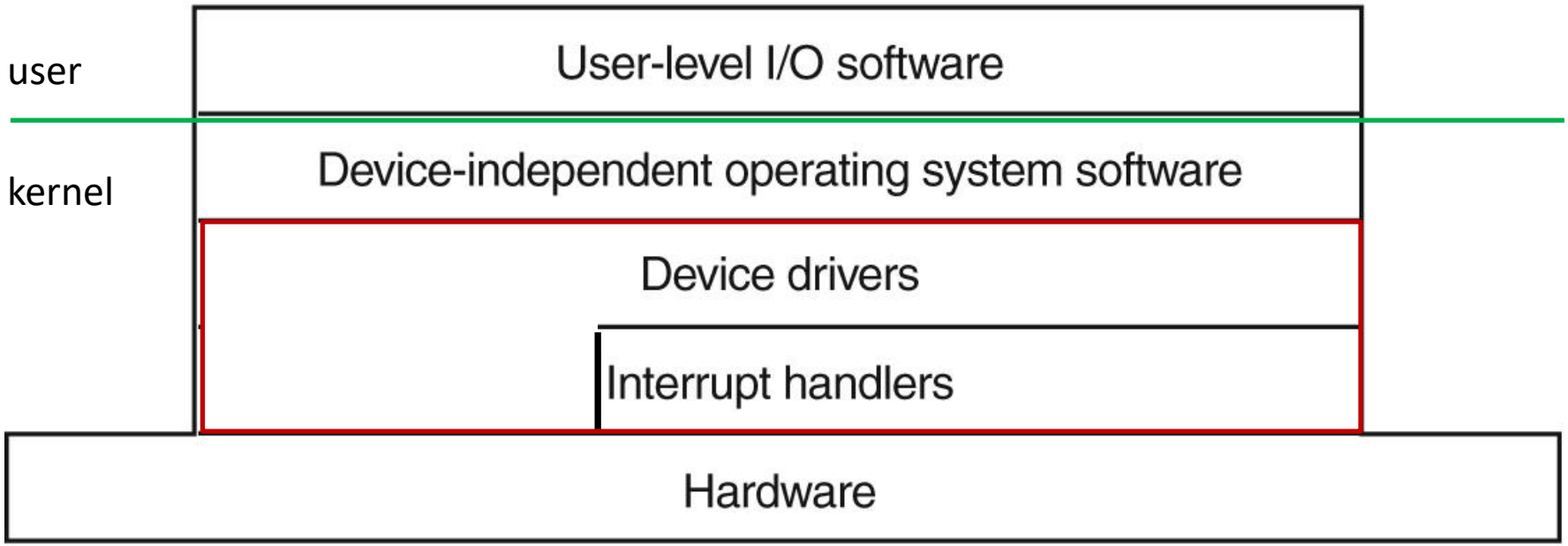
Configurations

DMA and Memory Protection

- Since DMA allows devices access to memory it can be a source of BUGs (and considerable headaches)
- System and Operating System protects itself with another translation table → IOMMU
- Enables OS to install only buffer addresses in IOMMU
- If Device attempts DMA to an address and there is no translation, an interrupt/exception is raised
- It is similar how apps are restricted to what memory they can access using a pagetable (MMU).



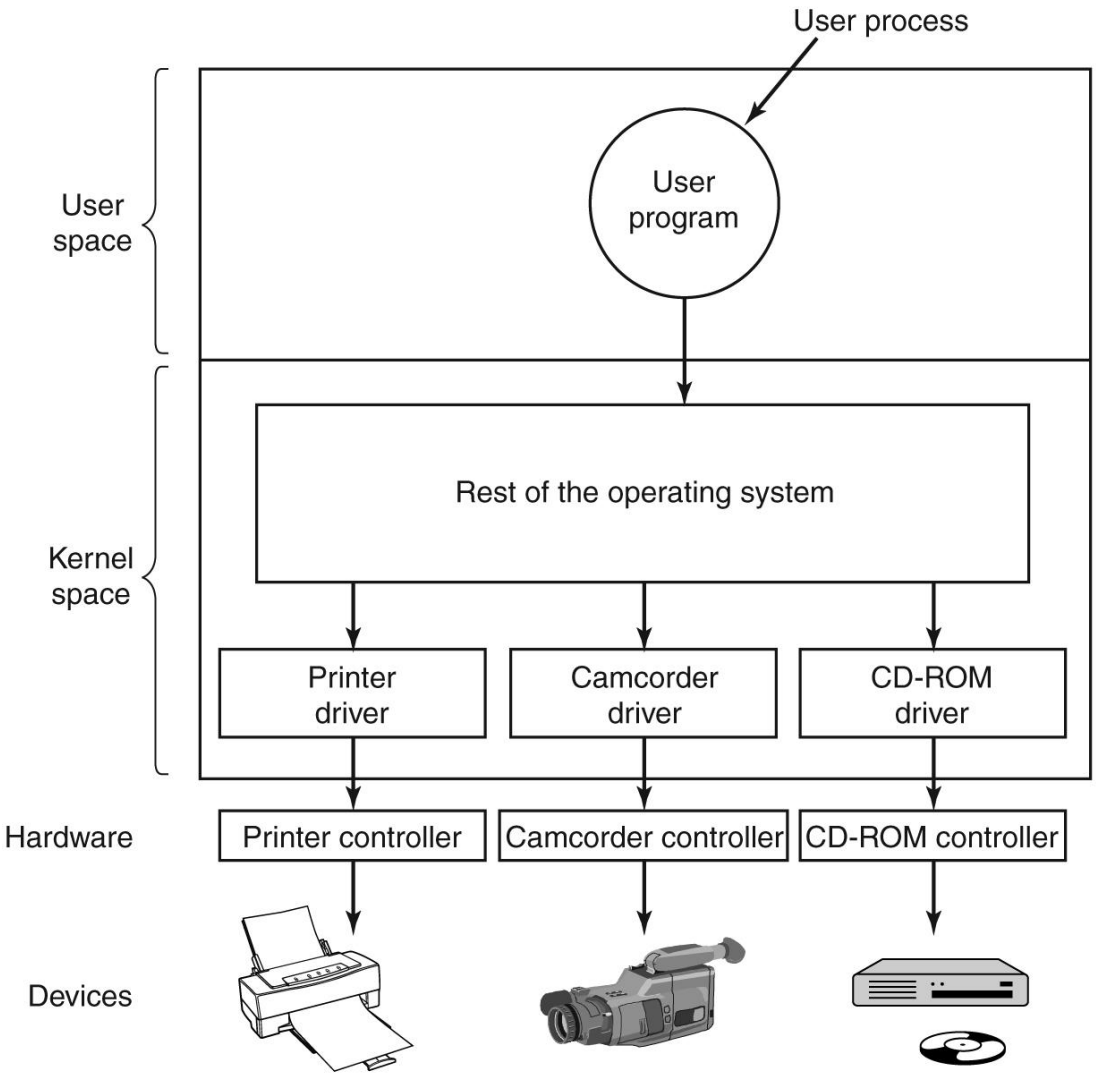
OS Software Layers for I/O



Device Drivers

- Device specific code for controlling the device
  - Read device registers from controller
  - Write device registers to issue commands
  - Handle Interrupts
- Usually supplied by the device manufacturer
- Can be part of the kernel or at user-space (with system calls to access controller registers)
- OS defines a standard interface that drivers for block devices must follow and another standard for driver of character devices

Device Drivers

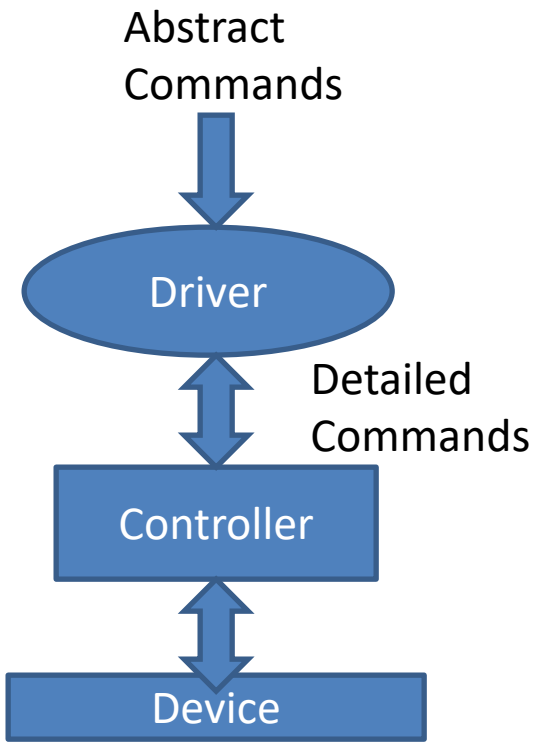


Device Drivers

- Main functions:
  - Initialize the device
  - Receive abstract read/write from layer above and execute them
  - Handle Interrupts
  - Log events
  - Manage power requirements

- Driver Code must be reentrant
  - Deal with multiple devices of same type
  - As a result they always pass a struct dev *devobj object to each function all state must be maintained in this object

- Drivers must deal with events such as a device removed or plugged



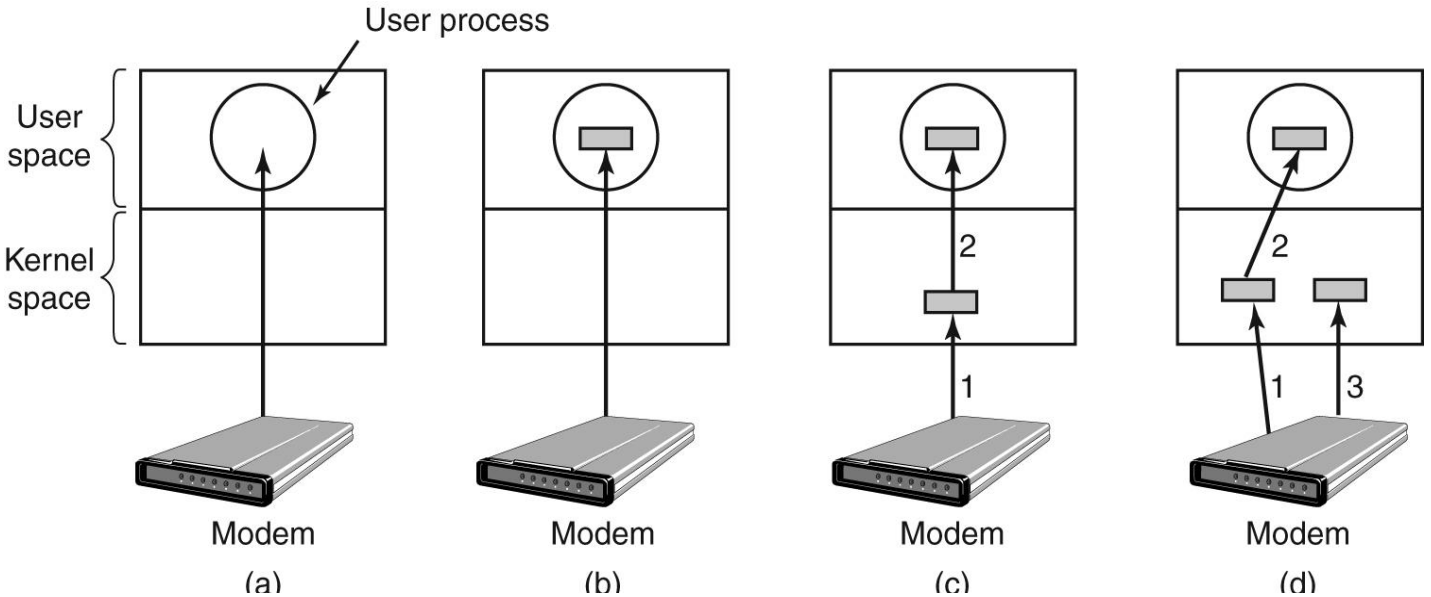
Device Independent I/O Software

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicated devices
Providing a device-independent block size

Device Independent I/O Software

- Uniform interfacing for device drivers
  - Trying to make all devices look the same
  - For each class of devices, the OS defines a set of functions that the driver must supply.
  - This layer of OS maps symbolic device names onto proper drivers

Device Independent I/O Software



Interrupt with every character

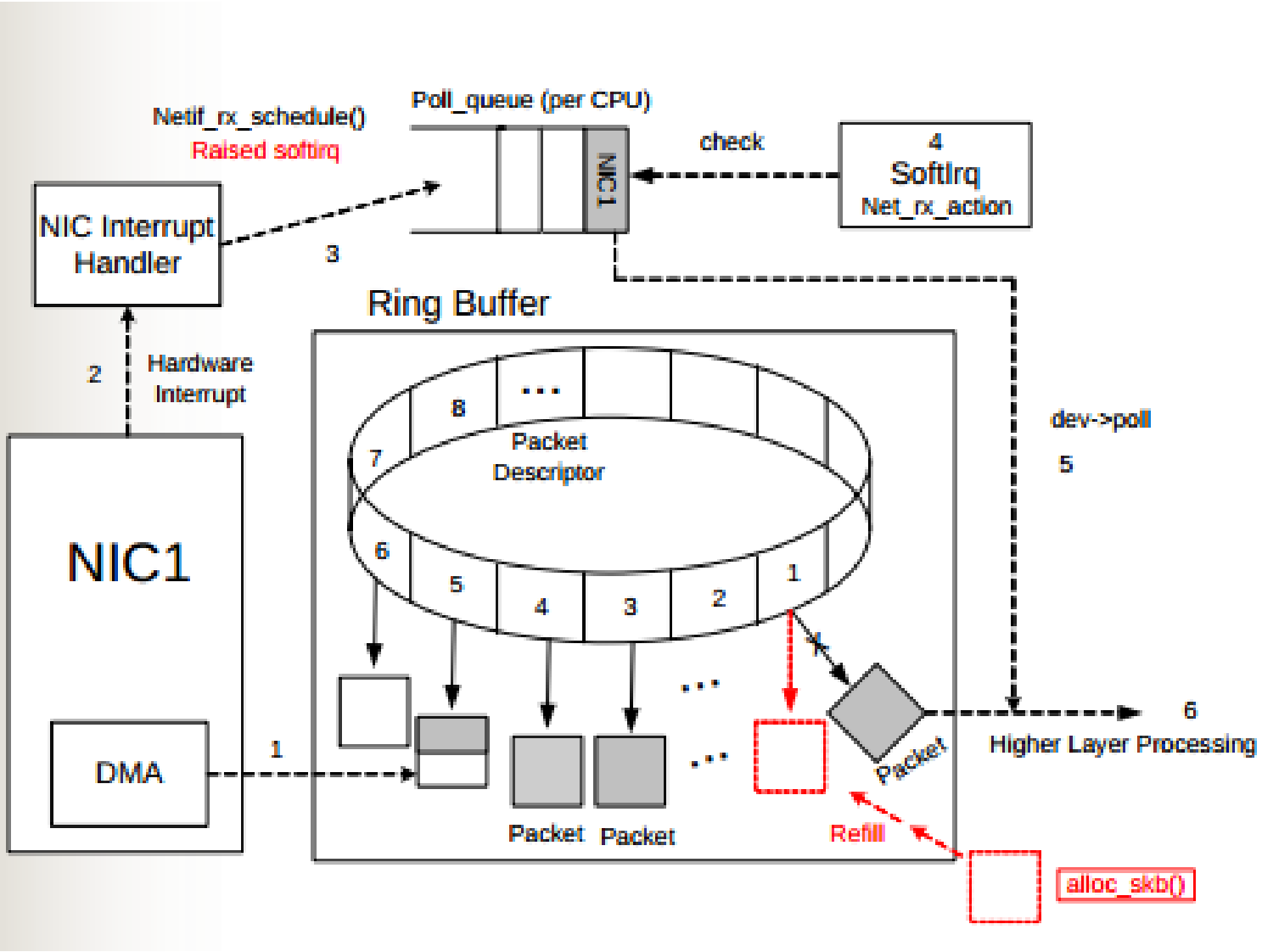
Very inefficient

Buffering n char.

What if buffer is paged out?

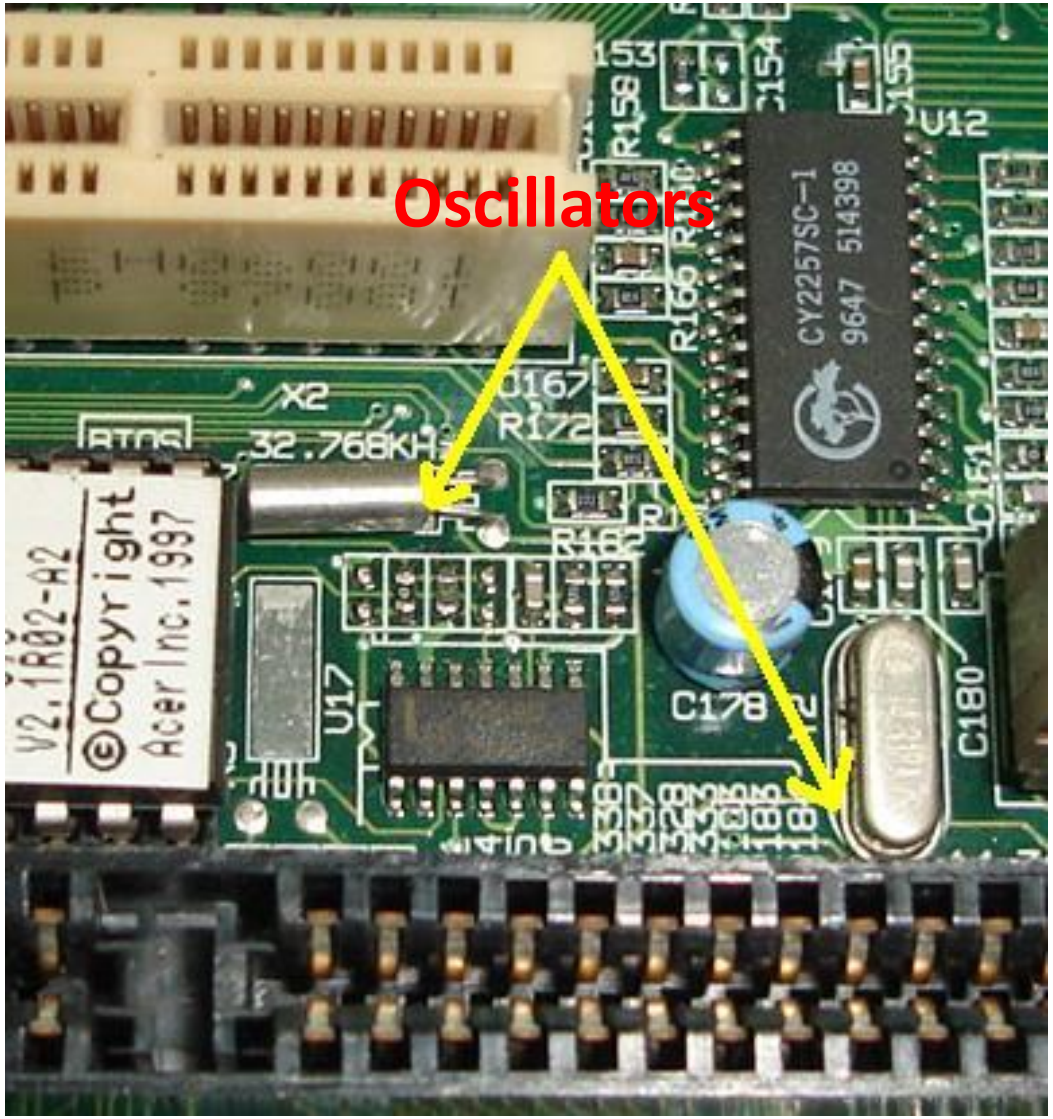
The need for buffering → buffering now is N buffers

Linux Networking Driver



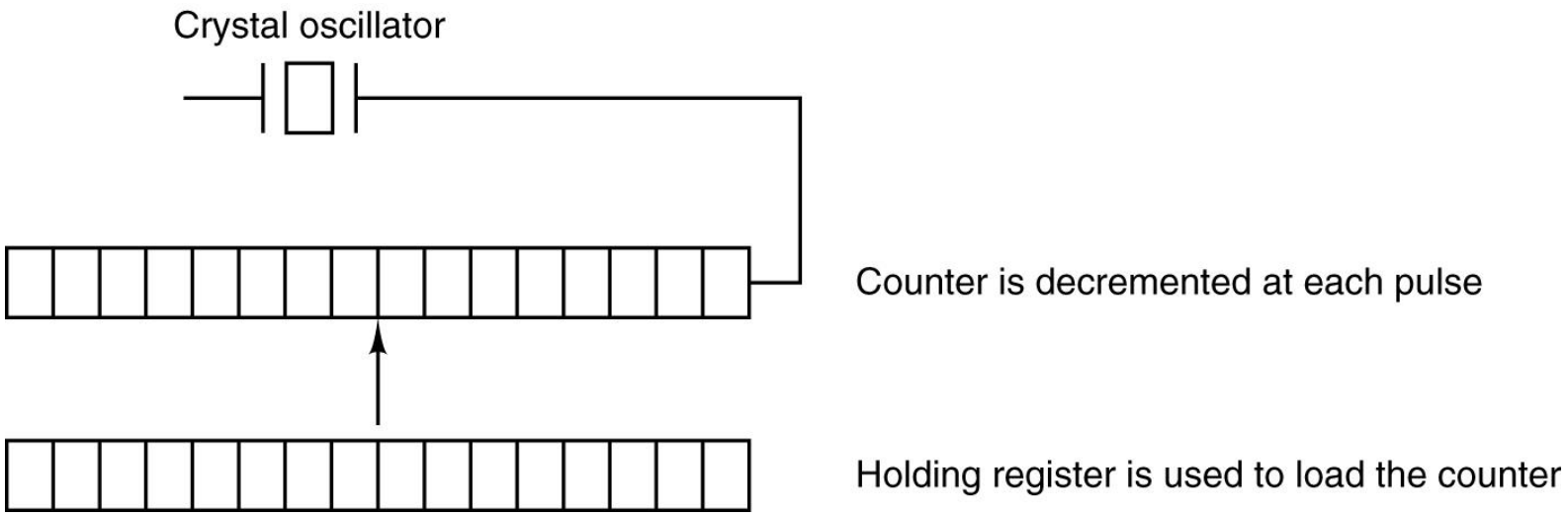
Example of Devices

Clocks


Oscillators
X2
32.768KH
CY2257SC-1
9647 514398
U12
U17
C178
C180
©Copyright
Acer Inc. 1997

Clock Hardware

- Old: tied to the power line and causes an interrupt on every voltage cycle
- New: Crystal oscillator + counter + holding register



Clock Software

- Maintaining the time of the day
- Preventing processes from running longer than they are allowed to
- Accounting for CPU usage
- Handling alarm system call made by user processes
- Providing watchdog timers for parts of the system itself
- Doing profiling, monitoring, and statistics gathering

User Interfaces

- Keyboard
- Mouse
- Monitor

As examples, we will take a closer look at the keyboard and mouse.

Keyboards

- Contains simplified embedded processor that communicates through a port with the controller at the motherboard
- An interrupt is generated whenever a key is struck and a second one whenever a key is released
- Keyboard driver extracts the information about what happens from the I/O port assigned to the keyboard
- The number in the I/O port is called the scan code (7 bits for code + 1 bit for key press/release)

Keyboards

Microprocessor of the keyboard

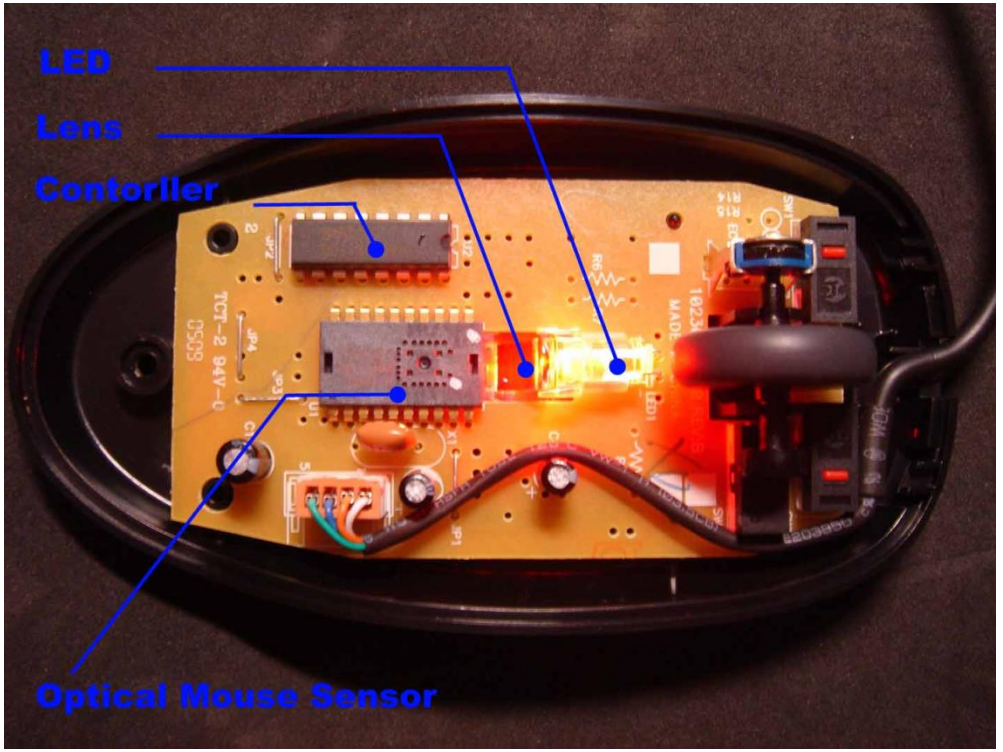
Key matrix

Keyboards

- There are two philosophies for programs dealing with keyboards
  1. The programs gets a raw sequence of ASCII codes (raw mode, or noncanonical mode)
  2. Driver handles all the intraline editing and just delivered corrected lines to the use programs (cooked mode or canonical mode)
- Either way, a buffer is needed to store characters

Mouse

- Mouse only indicates changes in position, not absolute position (delta_x and delta_y)



I/O Software Layer: Principle

- Interrupts are facts of life, but should be hidden away, so that as little of the OS as possible knows about them.
- The best way to hide interrupts is to have the driver starting an IO operation block until IO has completed and the interrupt occurs.
- OS registers assigned Interrupt with Device's interrupt handler and when interrupts happen, the associated interrupt handler is invoked.
- Once the handling of interrupt is done, the interrupt handler unblocks the thread in the device driver that started it.
- This model works if drivers are structured as kernel "threads/processes" with their own states, stacks and program counters.

More complex real device drivers

- Character Drivers
  - Discovery
  - Read / write data
- Disk storage
  - Discovery
  - Read block
  - Write block
  - Interrupt management
- Network Cards (will talk about it in last class)
  - Discovery
  - Send packet
  - Receive packet
  - Interrupt management
  - Throttling

Character Device Driver

- Let's do a life walk-through from the linux-kernel-labs

- https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html

Block Device Driver

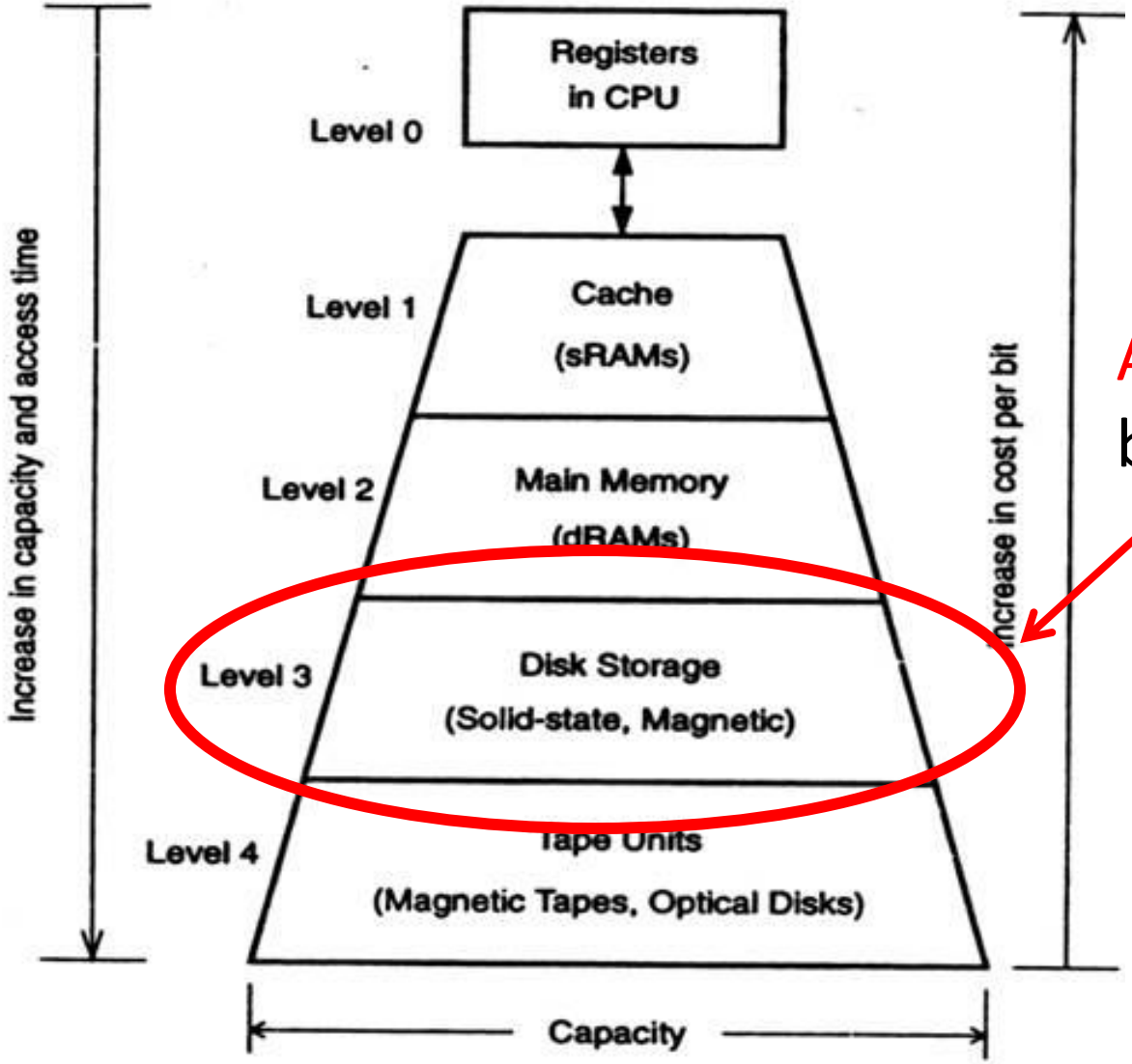
- Let's do a life walk-through from the linux-kernel-labs

- https://linux-kernel-labs.github.io/refs/heads/master/labs/block_device_drivers.html

Conclusions

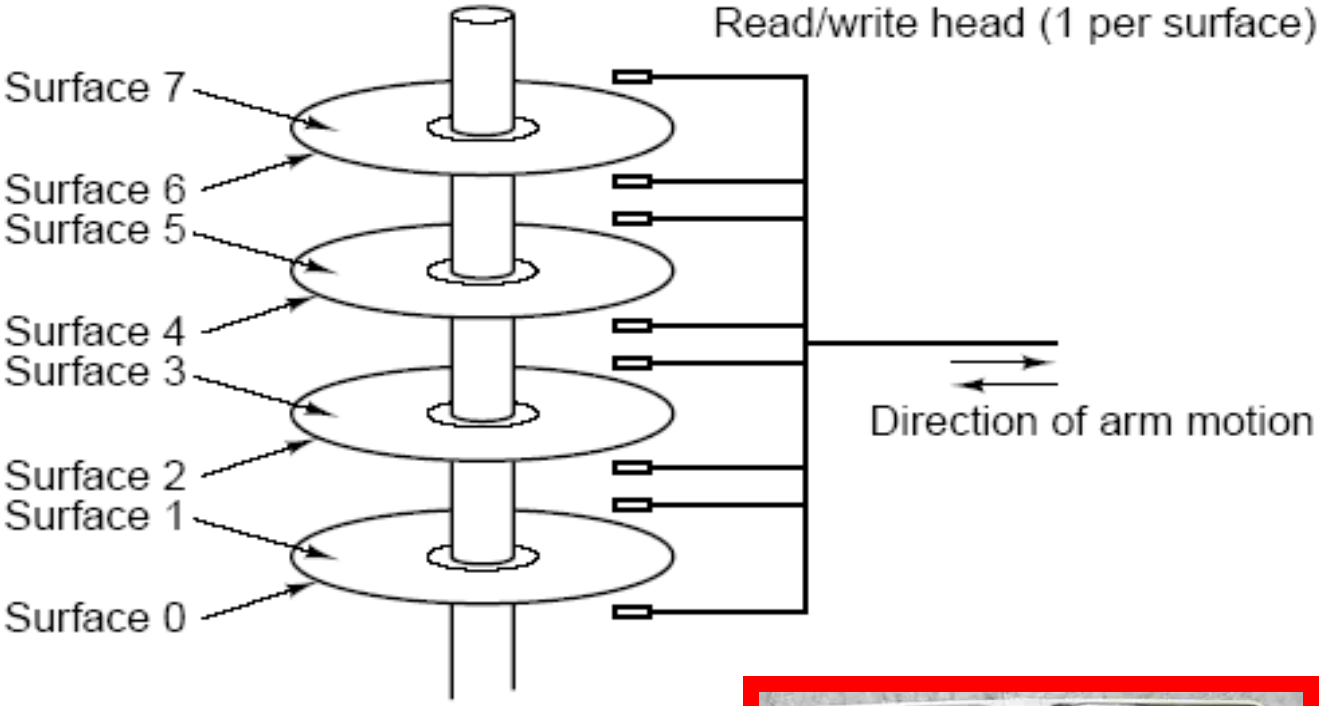
- The OS provides an interface between the devices and the rest of the system.
- The I/O part of the OS is divided into several layers.
- The hardware: CPU, programmable interrupt controller, DMA, device controller, and the device itself.
- OS must *expand* as new I/O devices are added

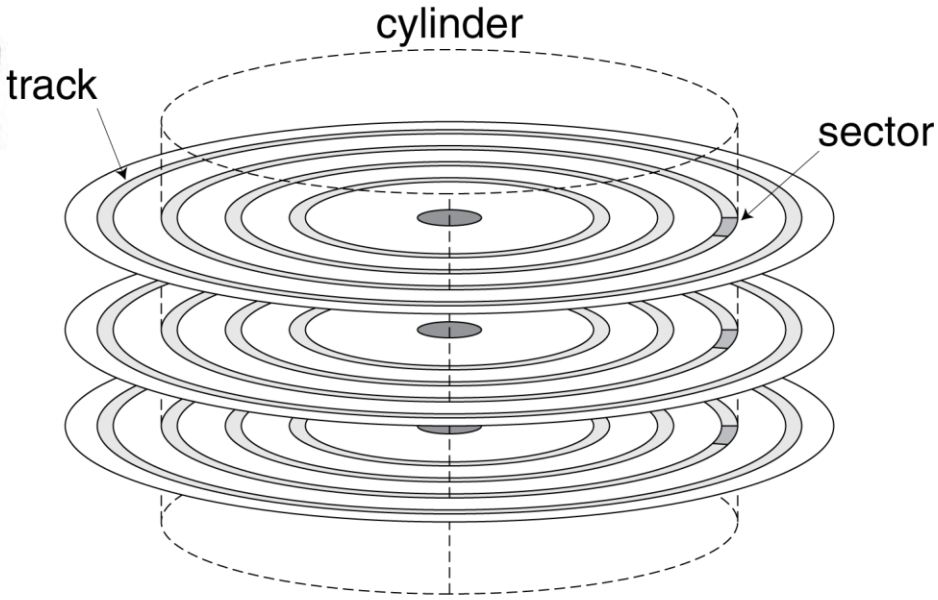
Disks Scheduling



Abstracted
by OS as files

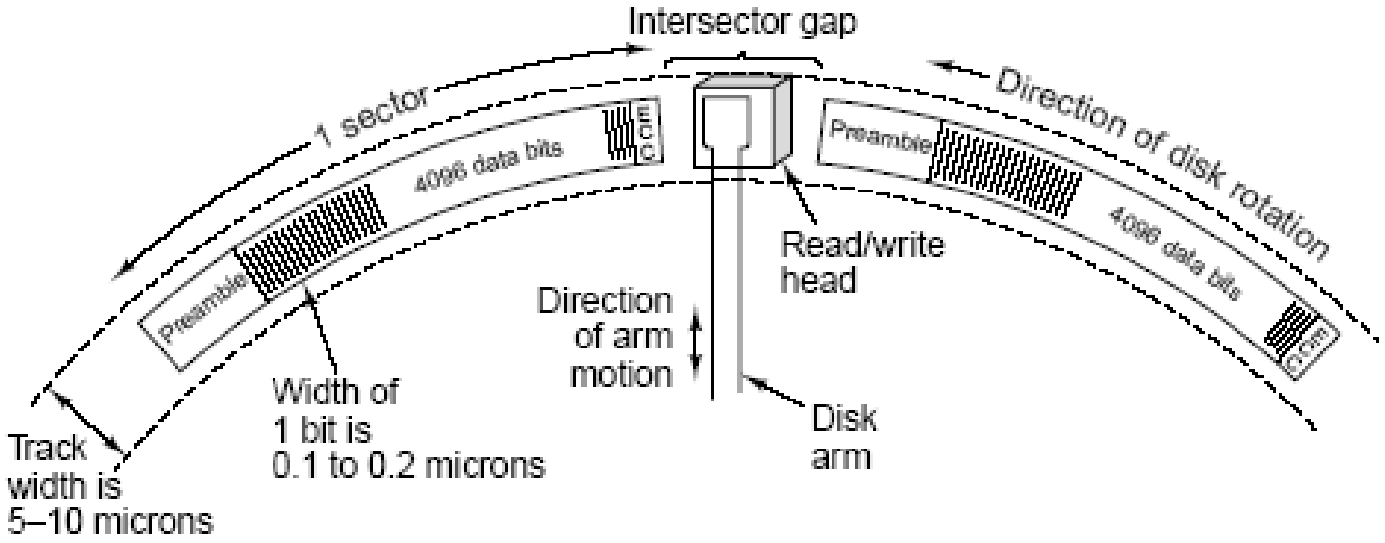
A Conventional Hard Disk (Magnetic) Structure




cylinder
track
sector

Hard Disk (Magnetic) Architecture

- **Surface** = group of tracks
- **Track** = group of sectors
- **Sector** = group of bytes
- **Cylinder**: several tracks on corresponding surfaces



Hard Disk Performance

- Seek
  - Position heads over cylinder, typically we assume 10msec avg:
    - 4 msecs for highend disk drives to
    - 15 msecs for mobile devices

- Rotational delay
  - Wait for a sector to rotate underneath the heads
  - Typically 7.14 – 2.0 ms (4,200 - 15,000 rpm)
    [or $\frac{1}{2}$ rotation]

HDD spindle speed [rpm]	Average rotational latency [ms]
4,200	7.14
5,400	5.56
7,200	4.17
10,000	3.00
15,000	2.00

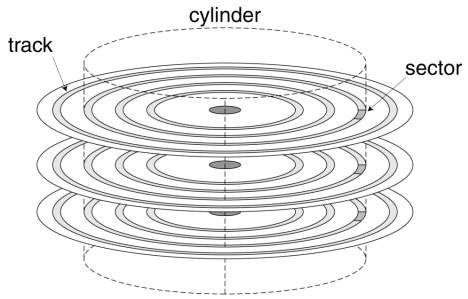
- Transfer bytes
  - Average transfer bandwidth (50-80 Mbytes/sec @4KB block sz)

Hard Disk Performance

- Performance of transfer 1 Kbytes
  - Seek (5.3 ms) + half rotational delay (3ms) + transfer (0.04 ms)
  - Total time is 8.34ms or 120 Kbytes/sec!
- What block size can get 90% of the disk transfer bandwidth?

Disk Behaviors (example)

- There are more sectors on outer tracks than inner tracks
  - Read outer tracks: 37.4MB/sec
  - Read inner tracks: 22MB/sec
- Seek time and rotational latency dominate the cost of small reads
  - A lot of disk transfer bandwidth is wasted
  - Need algorithms to reduce seek time



Block Size (Kbytes)	% of Disk Transfer Bandwidth
1Kbytes	0.5%
8Kbytes	3.7%
256Kbytes	55%
1Mbytes	83%
2Mbytes	90%

Observations

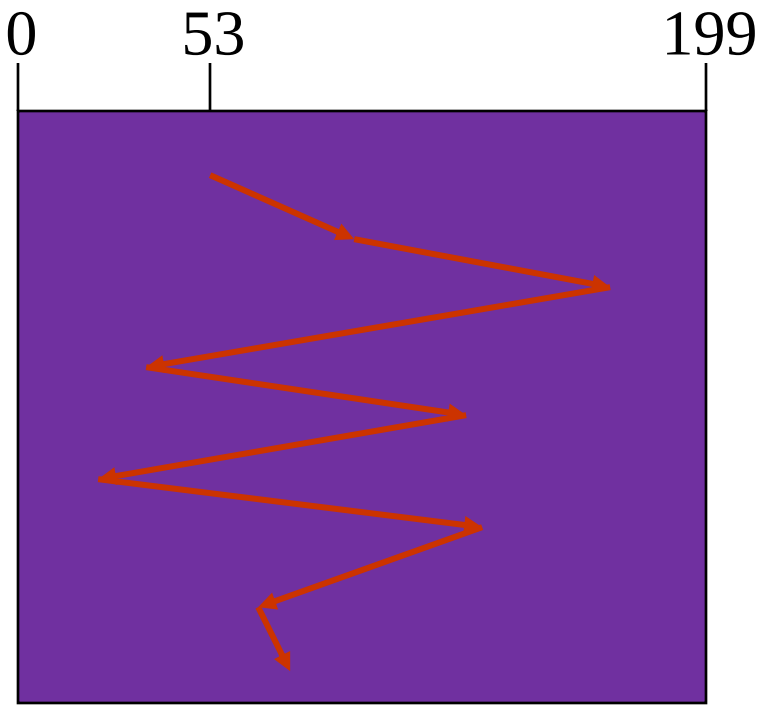
- Getting first byte from disk read is slow
  - high latency
- Peak bandwidth high, but rarely achieved
- Need to mitigate disk performance impact
  - Do extra calculations to speed up disk access
    - Schedule requests to shorten seeks
  - Move some disk data into main memory – file system caching

Disk Scheduling

- Which disk request is serviced first?
  - FCFS (FIFO)
  - Shortest Seek Time First (SSTF)
  - Elevator (SCAN)
  - LOOK
  - C-SCAN (Circular SCAN)
  - C-LOOK
- Looks familiar?

FIFO (FCFS) order

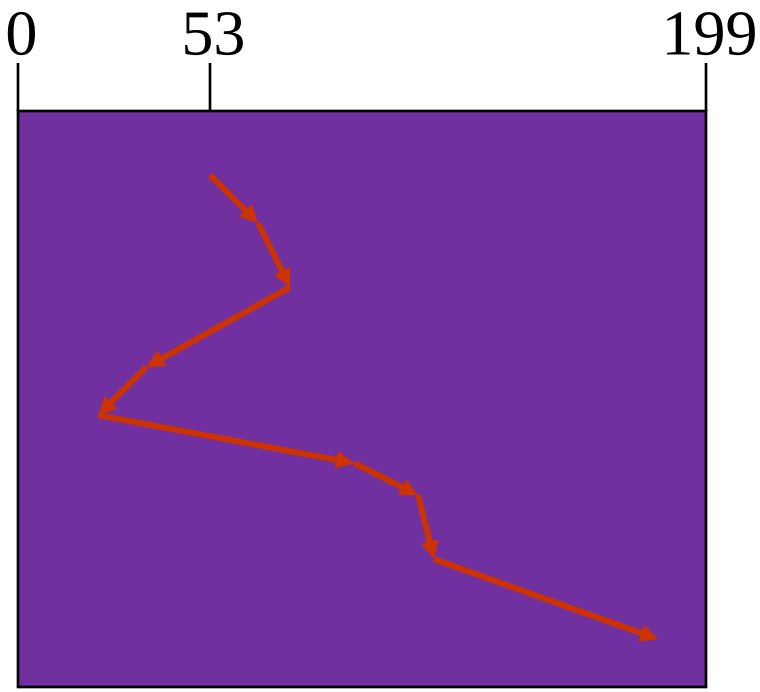
- Method
  - First come first serve
- Pros
  - Fairness among requests
  - In the order applications expect
- Cons
  - Arrival may be on random spots on the disk (long seeks)
  - Wild swing can happen
- Analogy:
  - Can elevator scheduling use FCFS?



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

SSTF (Shortest Seek Time First)

- Method
  - Pick the one closest on disk
  - Rotational delay is in calculation
- Pros
  - Try to minimize seek time
- Cons
  - Starvation
- Question
  - Is SSTF optimal?
  - Can we avoid starvation?
- Analogy: elevator

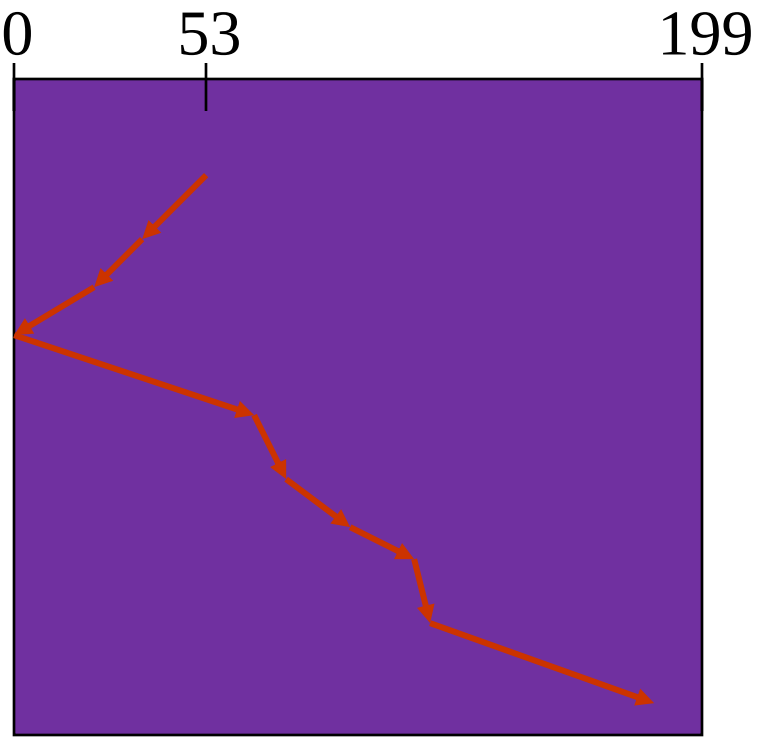


98, 183, 37, 122, 14, 124, 65, 67

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

Elevator (SCAN)

- Method
  - Take the closest request in the direction of travel
  - Real implementations do not go to the end (called LOOK)
- Pros
  - Bounded time for each request
- Cons
  - Request at the other end will take a while

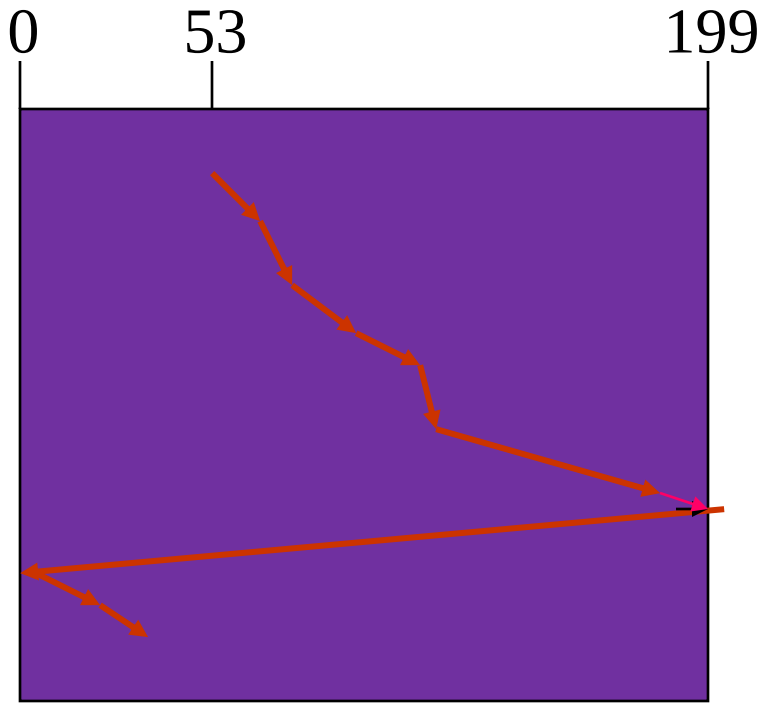


98, 183, 37, 122, 14, 124, 65, 67

(37, 14, 0, 65, 67, 98, 122, 124, 183)

C-SCAN (Circular SCAN)

- Method
  - Like SCAN
  - But, wrap around
  - Real implementation doesn't go to the end (C-LOOK)
- Pros
  - Uniform service time
- Cons
  - Do nothing on the return



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

(65, 67, 98, 122, 124, 183, 199, 0, 14, 37)

LOOK and C-LOOK

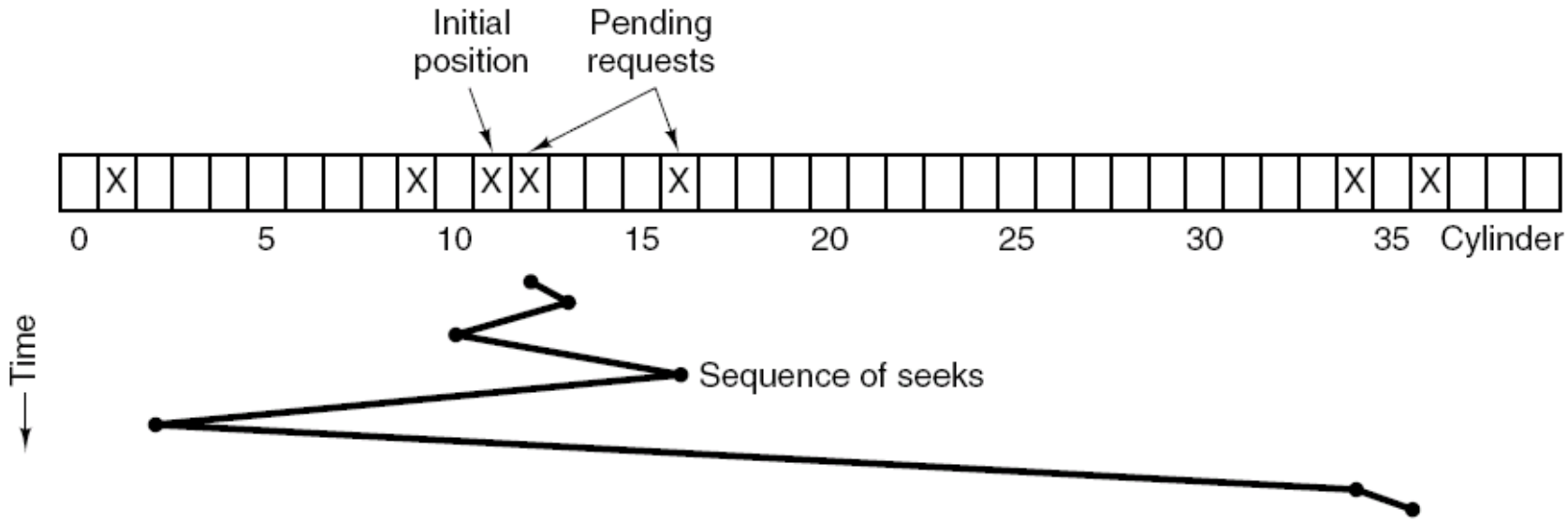
- SCAN and C-SCAN move the disk arm across the full width of the disk
- In practice, neither algorithm is implemented this way
- More commonly, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without first going all the way to the end of the disk.
- These versions of SCAN and C-SCAN are called LOOK and C-LOOK

FSCAN / FLOOK

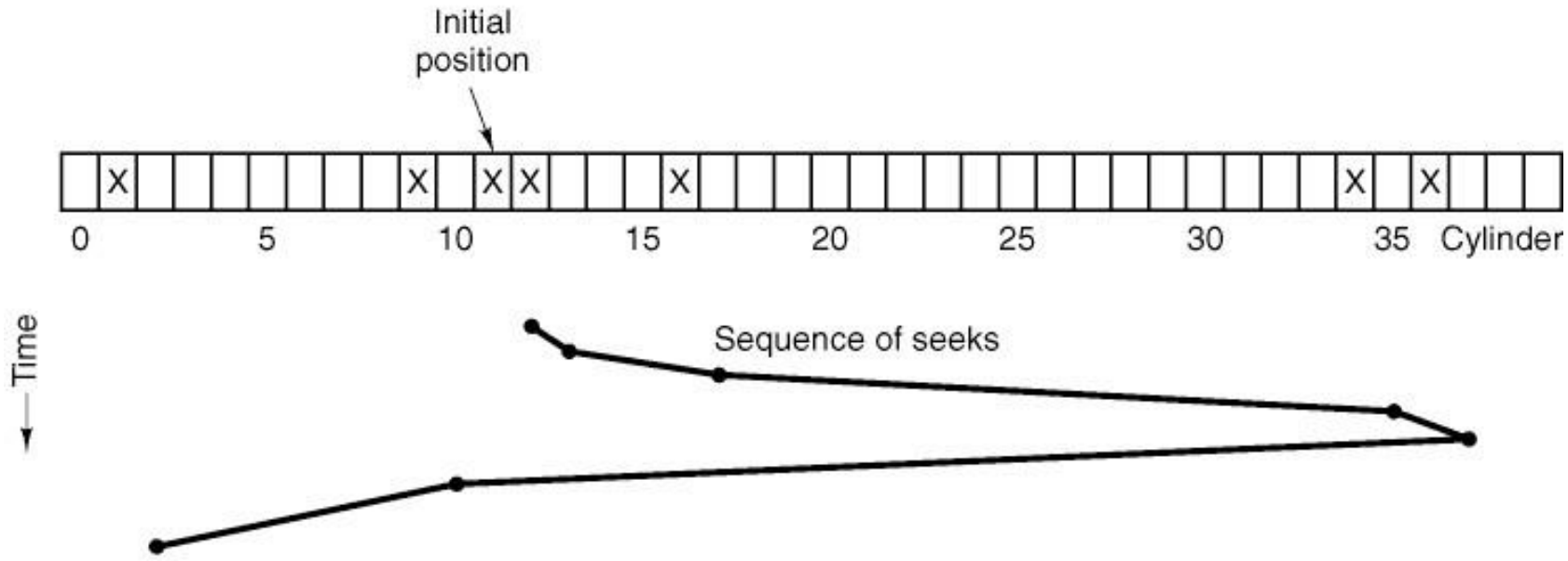
- Like SCAN / LOOK respectively… but
- Operates with two queues:
  - `active-queue`
  - `add-queue`
- any new I/O request is entered into add-queue.
- I/O is only scheduled from active-queue, when active-queue is empty, the queues are swapped (pointers !!) and tried one more time.

Which algorithm is this?

(assume I/O arrived at random)



Which algorithm is this?



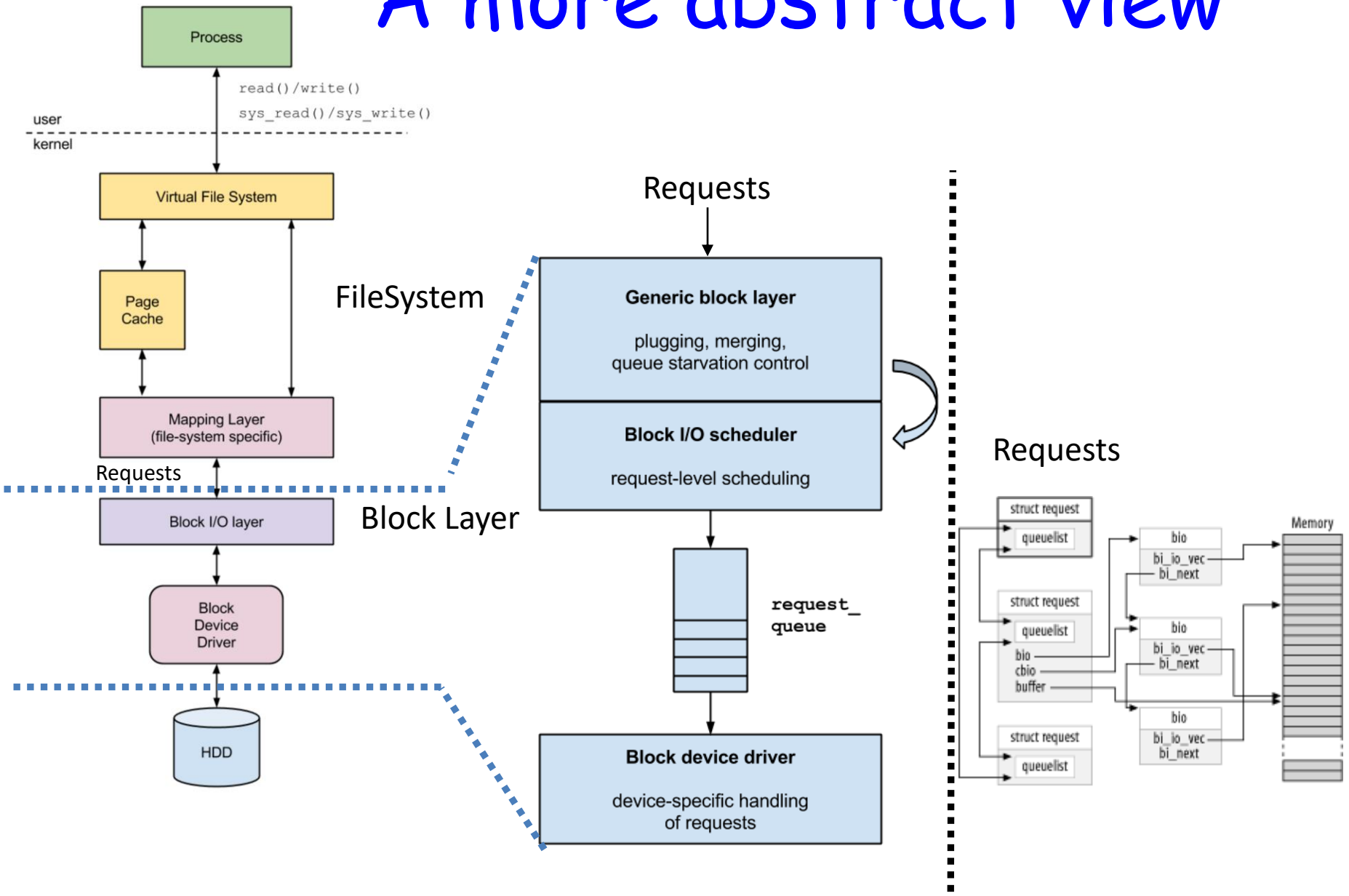
Other modern schedulers (1)

- Deadline Scheduler
  - Guarantees a start time for each I/O requests → deadline
  - Read requests have an expiration time of 500 ms, write requests expire in 5 seconds.
  - Maintains read + write deadline queues + sorted (by sector) queue
  - If a deadline expired then it is served otherwise, a batch is served from sorted queue.

Other modern schedulers (2)

- CFQ (completely fair queueing)
  - places requests submitted into a number of per-process queues
  - allocates timeslices for each queue to access the disk.
  - The length of the time slice and the number of requests a queue is allowed to submit depends on the I/O priority of the given process.
  - Implicitly provides "idle" time at end of I/O request to allow subsequent requests to be issued and bundled

A more abstract view



Other modern schedulers (3)

- Linux allows selection (dynamically) on a per I/O device
- LIVE DEMO

File: /sys/block/sda/queue/scheduler

Lab3 Release

- Discussion

RAID

Common Hard Drive Errors

1. Programming error
   - request for nonexistent sector
2. Transient checksum error
   - caused by dust on the head
3. Permanent checksum error
   - disk block physically damaged
4. Seek error
   - the arm sent to cylinder 6 but it went to 7
5. Controller error
   - controller refuses to accept commands

RAID

- *Redundant Array of Independent Disks* OR

  Redundant Array of Inexpensive Disks

- Use parallel processing to speed up CPU performance

- Use parallel I/O to improve disk performance, reliability (1988, Patterson)

- Design new class of I/O devices called RAID – Redundant Array of Inexpensive Disks (also Redundant Array of Independent Disks)

- Use the RAID in OS as a SLED (Single Large Expensive Disk), but with better performance and reliability

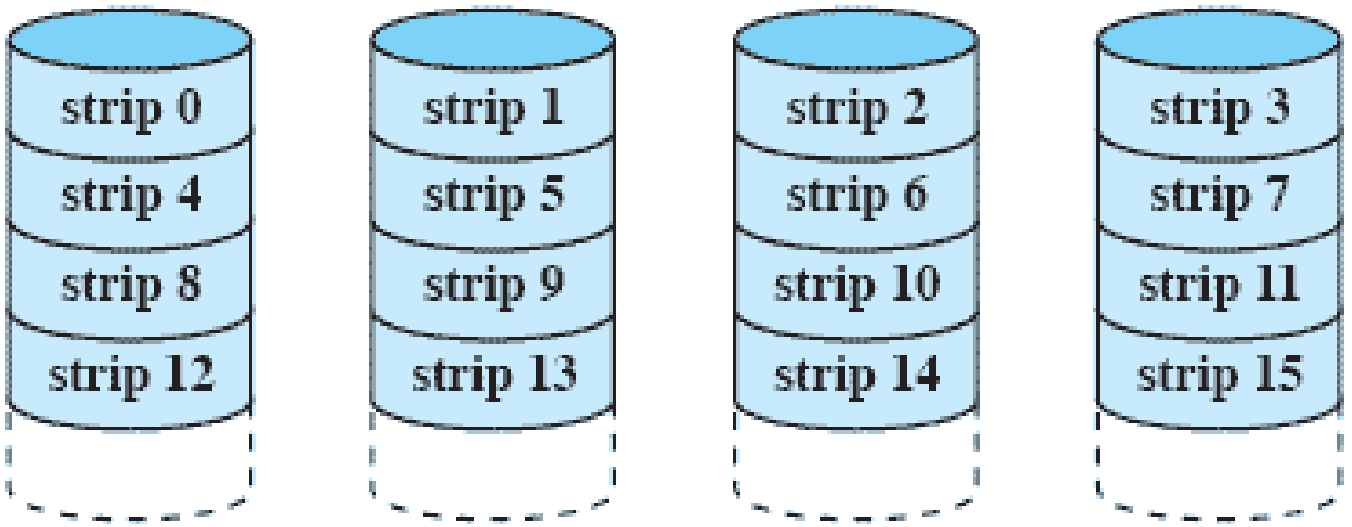
RAID

- RAID consists of RAID SCSI controller plus a box of SCSI disks
- Data are divided into strips and distributed over disks for parallel operation
- RAID 0 … RAID 6 levels
- RAID 0 organization writes consecutive strips over the drives in round-robin fashion – operation is called striping
- RAID 1 organization uses striping and duplicates all disks
- RAID 2 uses words, even bytes and stripes across multiple disks; uses error codes, hence very robust scheme
- RAID 3, 4, 5, 6 alterations of the previous ones

Small Computer System Interface (**SCSI**, /ˈskʌzi/ ***SKUZ****-ee*) is a set of standards for physically connecting and transferring data between computers and peripheral devices. The SCSI standards define commands, protocols, electrical and optical interfaces. (source Wikipedia)

RAID Level 0

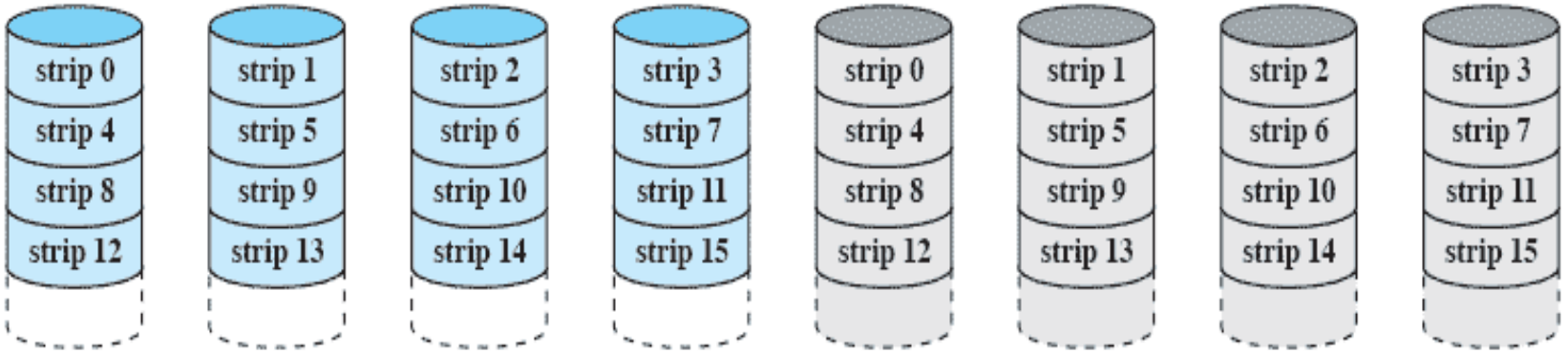
- Not a true RAID because it does not include redundancy to improve performance or provide data protection
- User and system data are distributed across all of the disks in the array
- Logical disk is divided into strips



(a) RAID 0 (non-redundant)

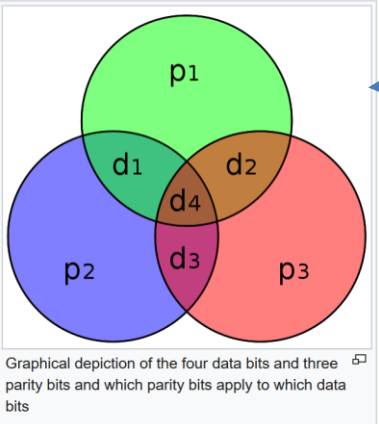
RAID Level 1

- Redundancy is achieved by the simple expedient of duplicating all the data
- There is no "write penalty"
- When a drive fails the data may still be accessed from the second drive
- Principal disadvantage is the cost



(b) RAID 1 (mirrored)

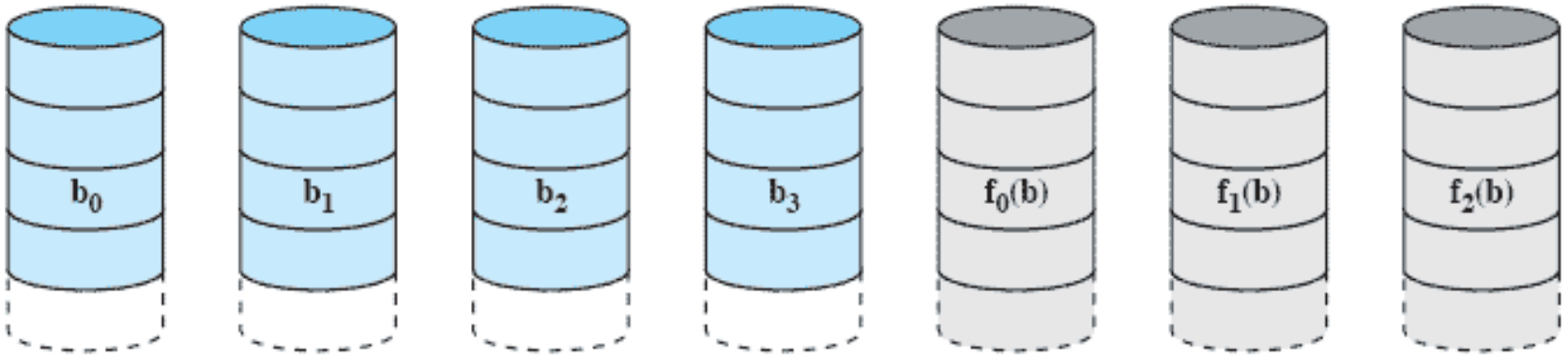
RAID Level 2



Graphical depiction of the four data bits and three parity bits and which parity bits apply to which data bits

Requires > log(N) redundant disks

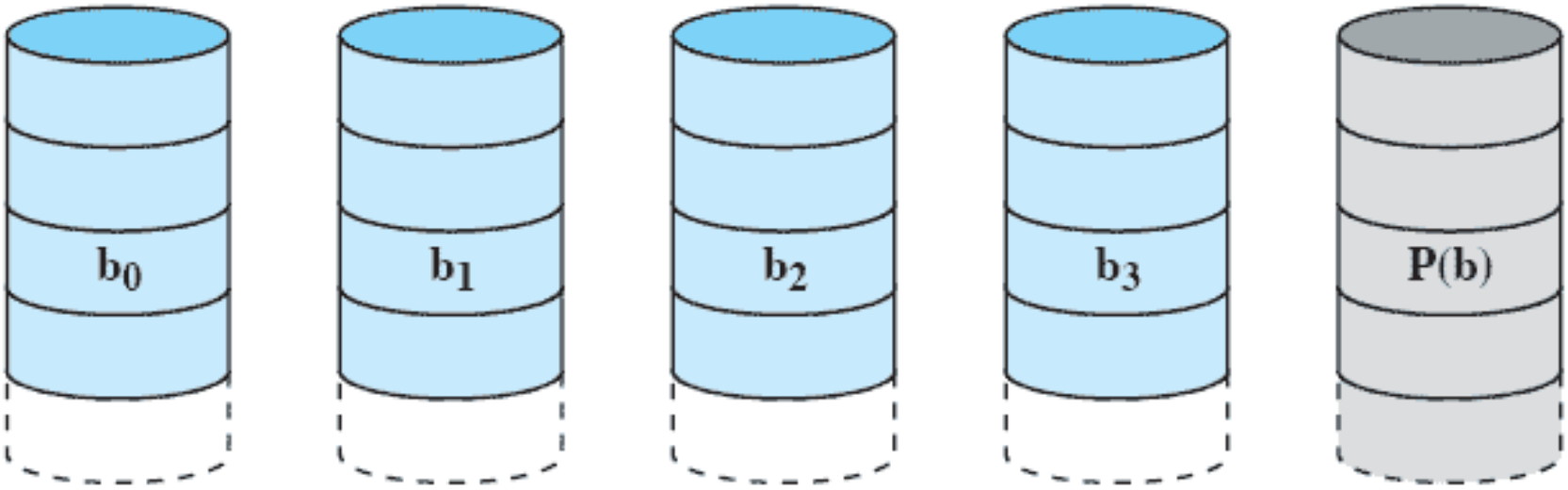
- Makes use of a parallel access technique, all disks participate
- Data striping is used, strips are very small, sometimes byte/words.
- Typically a Hamming code is used
- Effective choice in an environment in which many disk errors occur
- Can correct-single bit, detect 2-bit
- On read all disks are read



(c) RAID 2 (redundancy through Hamming code)

RAID Level 3

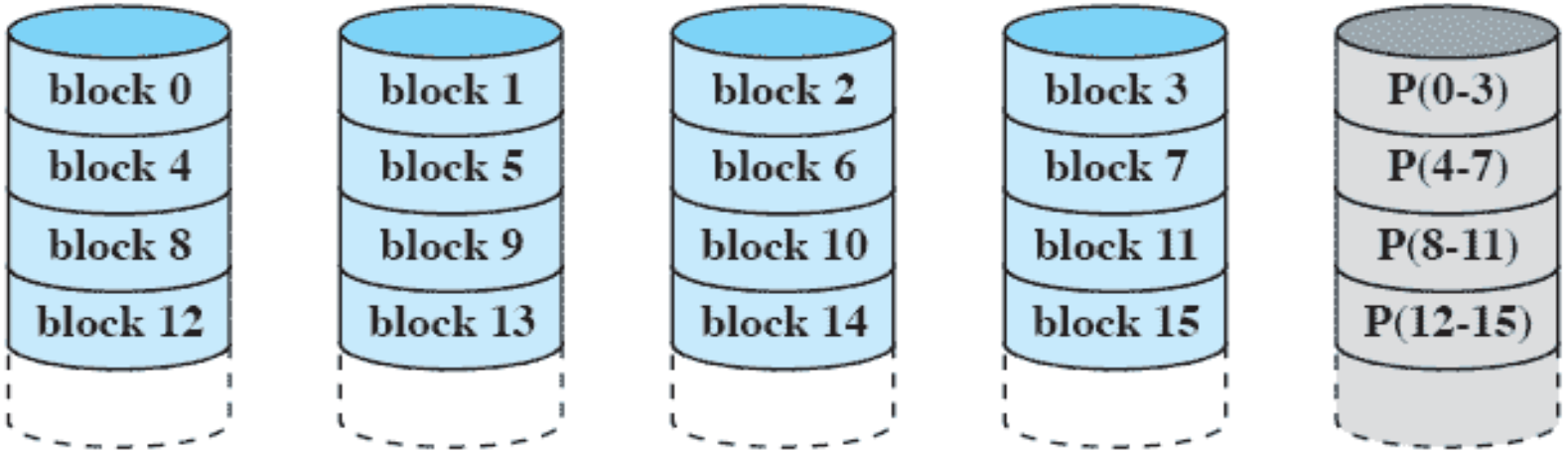
- Similar to Level2, but requires only a single redundant disk, no matter how large the disk array
- Employs parallel access, with data distributed in small strips
- Can achieve very high data transfer rates, but only one I/O at a time



(d) RAID 3 (bit-interleaved parity)

RAID Level 4

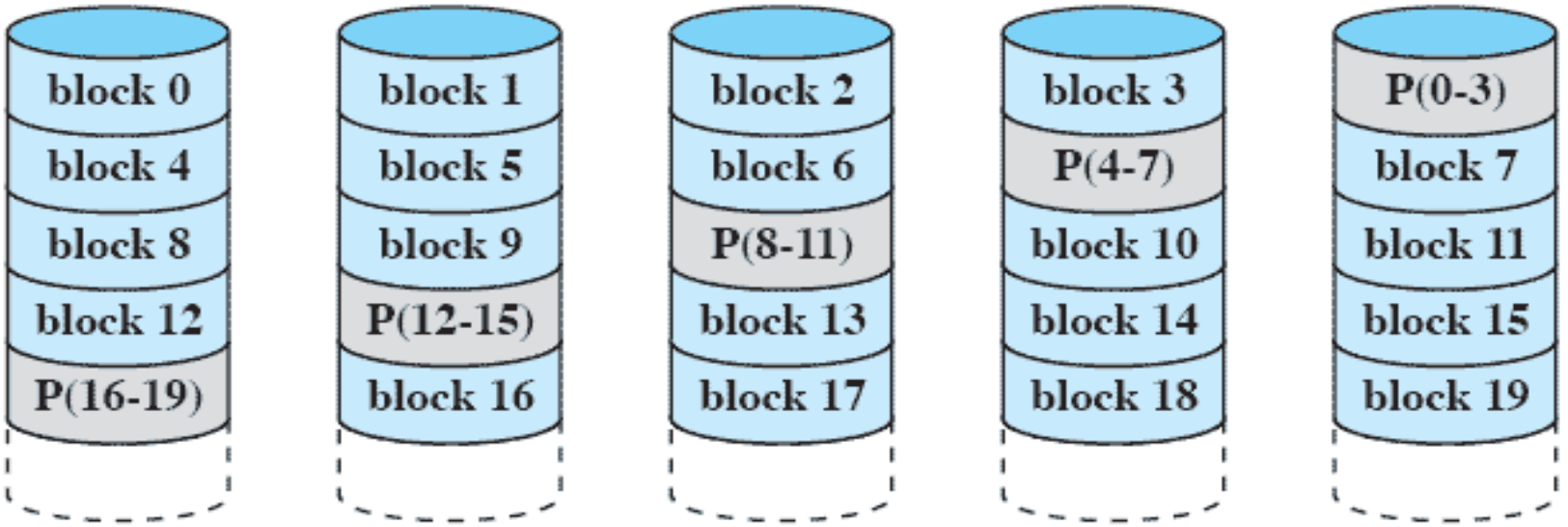
- Makes use of an independent access technique and strips are large.
- A bit-by-bit parity strip is calculated across corresponding strips on each data disk, and the parity bits are stored in the corresponding strip on the parity disk
- Involves a write penalty when an I/O write request of small size is performed



(e) RAID 4 (block-level parity)

RAID Level 5

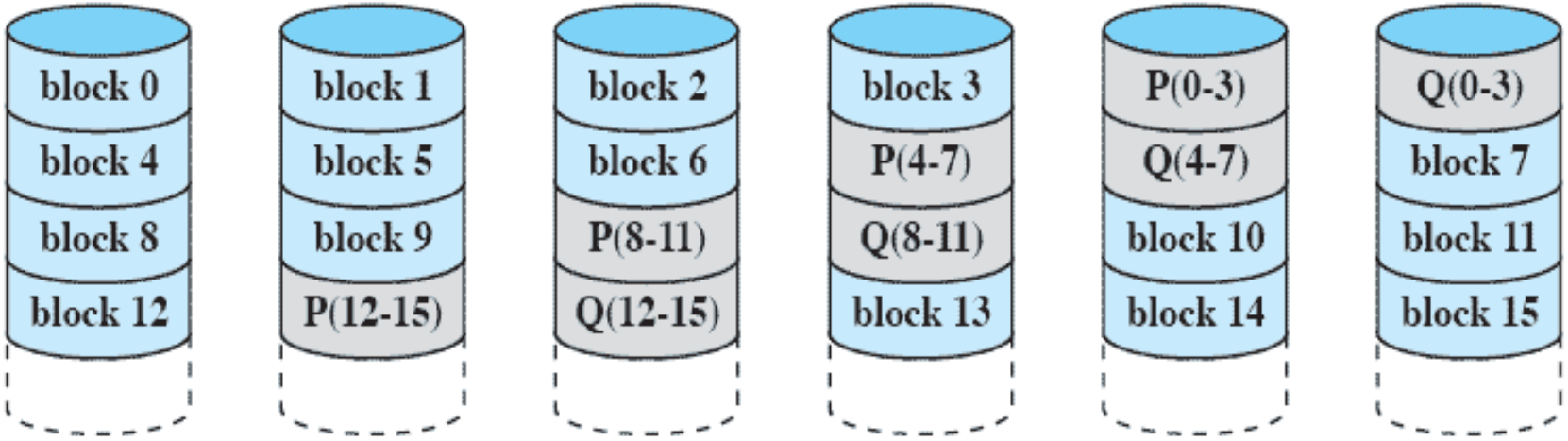
- Similar to RAID-4 but distributes the parity bits across all disks
- Typical allocation is a round-robin scheme
- Has the characteristic that the loss of any one disk does not result in data loss



(f) RAID 5 (block-level distributed parity)

RAID Level 6

- Two different parity calculations are carried out and stored in separate blocks on different disks
- Provides extremely high data availability
- Incurs a substantial write penalty because each write affects two parity blocks



(g) RAID 6 (dual redundancy)

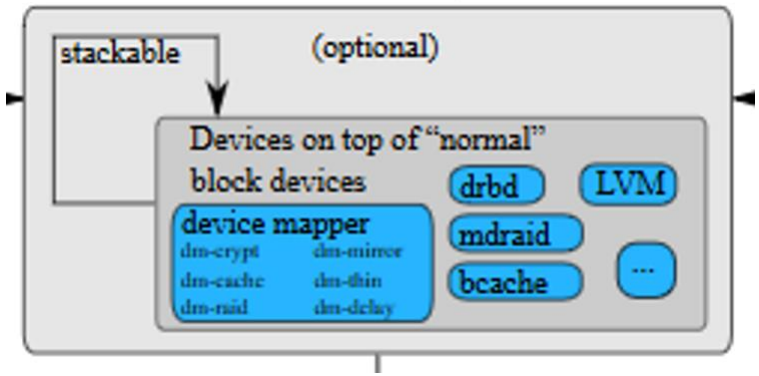
Table 11.4 RAID Levels

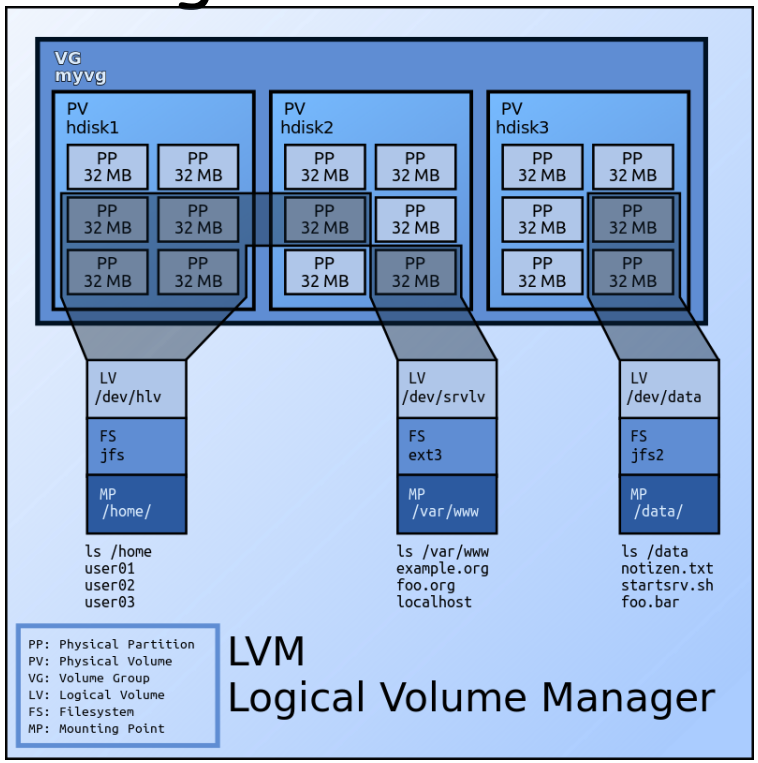
Category	Level	Description	Disks required	Data availability	Large I/O data transfer capacity	Small I/O request rate
Striping	0	Nonredundant	N	Lower than single disk	Very high	Very high for both read and write
Mirroring	1	Mirrored	$2N$	Higher than RAID 2, 3, 4, or 5; lower than RAID 6	Higher than single disk for read; similar to single disk for write	Up to twice that of a single disk for read; similar to single disk for write
Parallel access	2	Redundant via Hamming code	$N + m$	Much higher than single disk; comparable to RAID 3, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
	3	Bit-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
Independent access	4	Block-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 5	Similar to RAID 0 for read; significantly lower than single disk for write	Similar to RAID 0 for read; significantly lower than single disk for write
	5	Block-interleaved distributed parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 4	Similar to RAID 0 for read; lower than single disk for write	Similar to RAID 0 for read; generally lower than single disk for write
	6	Block-interleaved dual distributed parity	$N + 2$	Highest of all listed alternatives	Similar to RAID 0 for read; lower than RAID 5 for write	Similar to RAID 0 for read; significantly lower than RAID 5 for write

N = number of data disks; m proportional to log N

Logical Volume Manager (LVM)

- Physical Volume ~= HDD or SSD or partition
- LVM allows to combine a set of physical volumes to be treated as a volume group (via software)
- LVM allows flexible creation of logical volumes (think virtual disks) out of the volume group
- I/O requests are directed to correct physical disk

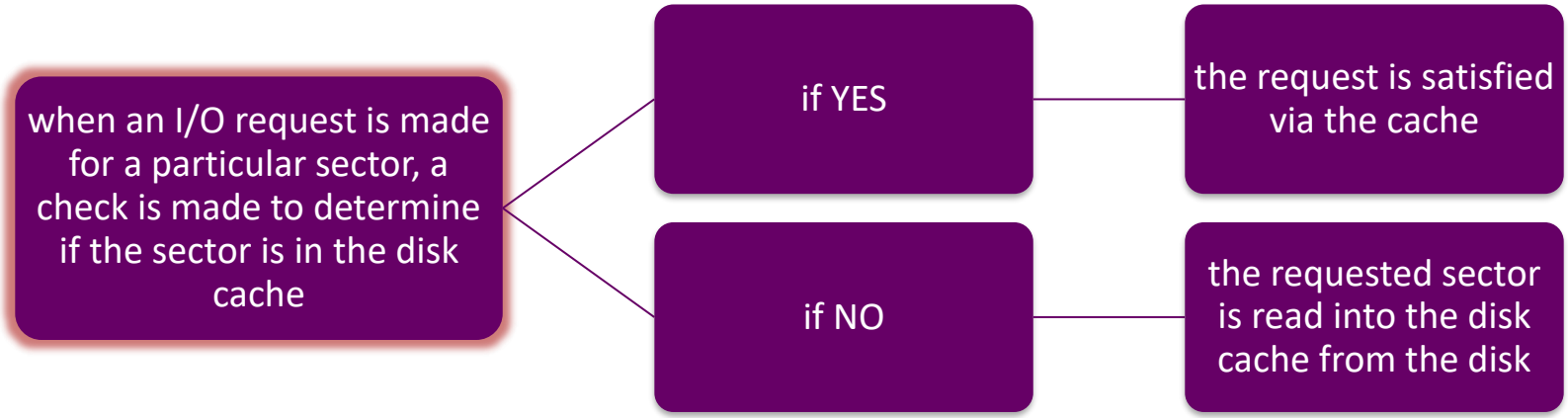




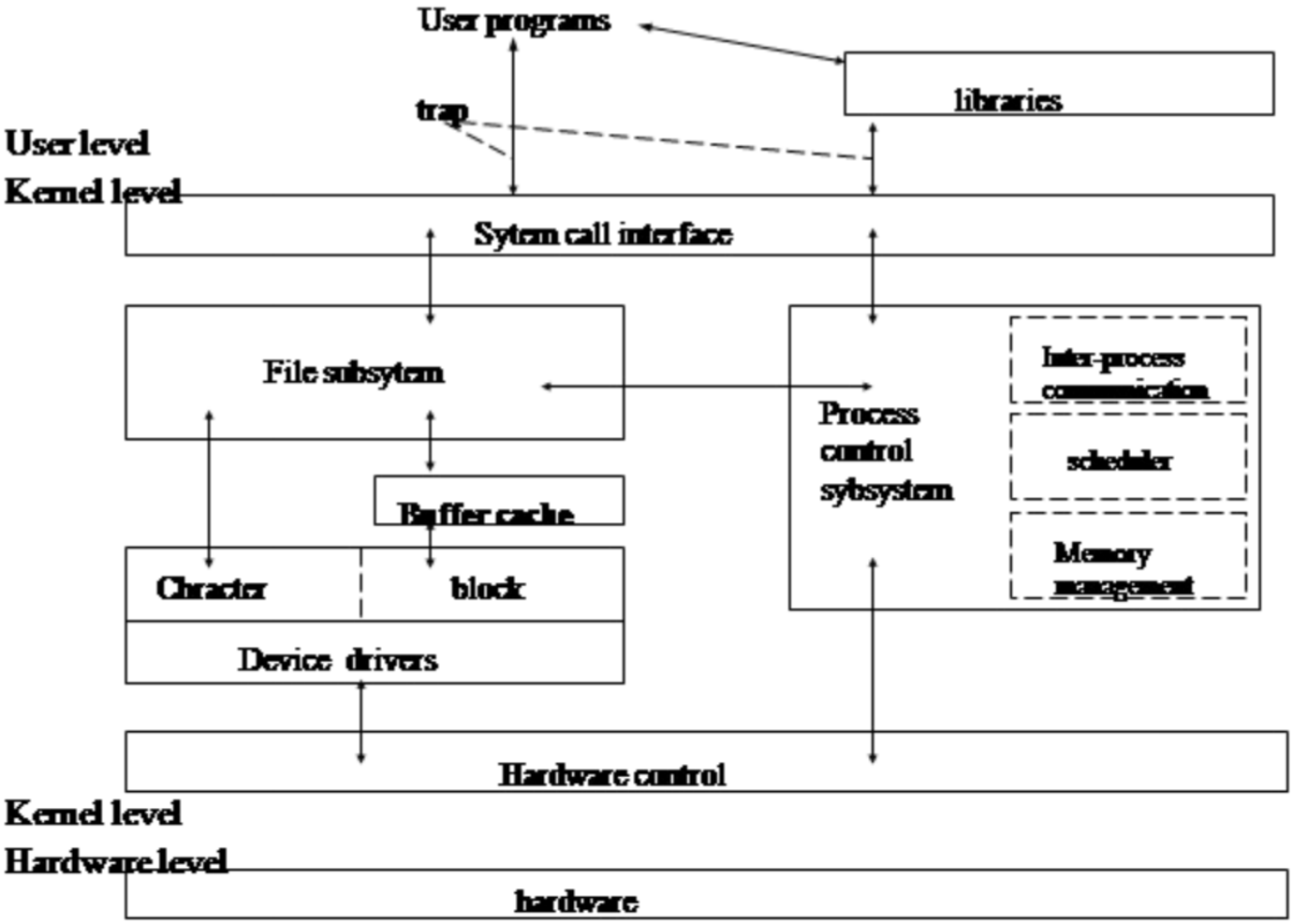
Other I/O related functions and Technologies

Disk Cache (aka buffer cache)

- *Cache memory* is used to apply to a memory that is smaller and faster than main memory and that is interposed between main memory and the processor
- Reduces average memory access time by exploiting the principle of locality
- *Disk cache* is a buffer in main memory for disk sectors
- Contains a copy of some of the sectors on the disk



Disk or Buffer Cache



Solid State Disks and FLASH Memory

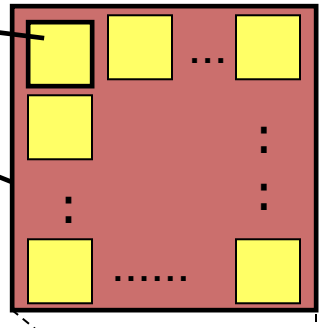
- Solid State Disks (all electronic , no mechanical parts)
  - Made out of FLASH Memory
  - Genealogy of FLASH
    - RAM, EPROM, EEPROM
      - RAM needs power
      - EPROM needs no power but could be programmed only once
      - EEPROM → erased and programmed, maintain the programmed value without power
      - 'ROM' → Read-Only-Memory: this term is used because although we could read any arbitrary location, entire 'block' needs to be erased at once.
    - Usage of EEPROMs
      - Historically, programs for embedded processors [which needed to be programmed once in a while]
      - Programming an EEPROM was not a time critical event in the past
      - Limitations on the number of times EEPROMs could be programmed ranged from 100s to 1000s and that was not a problem as typical embedded systems were programmed just a dozen times.
    - Evolution of Technology
      - Thousands of EEPROM circuits (called "blocks" in FLASH) are arrayed in order to randomly program and erase blocks

Organization of a Typical FLASH (Single-Layered) Chip

- 1 Page = 2KB

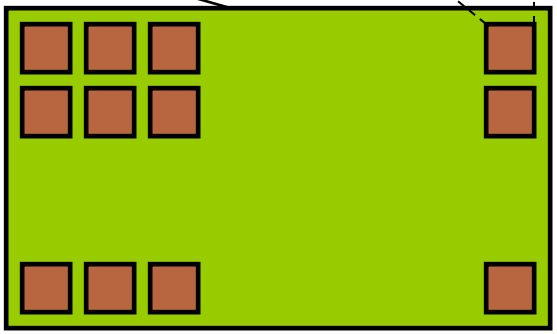
- 1 Block = 128KB

  (64 pages)



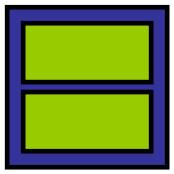
- **1 Plane = 2K Blocks**

  = 256MB



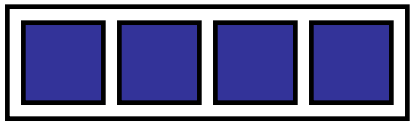
- **1 Die = 2 Planes**

  = 512MB



- **1 Flash Chip** = 2GB

  (4 Dies)



- **Functions supported**
  - Read
  - Erase
  - Program

FLASH Chip: A few details

- Operations on a FLASH Chip
  - 3 main operations: Read/Erase/Program. Write = Erase + Program.
  - Each set of dies in a chip-enable group operate independently as long as the shared data pins are not busy
  - With in a chip-enable group, only one of these operations can take place on a plane
  - Separate operations can take place in separate planes in parallel as long as they are submitted to the chip-enable group at once (this is because once a command is sent to the chip-enable group, it signals busy until the command is complete)
- Types of FLASH Chips
  - SLC: Single Layered Chip
    - Stores 0/1 in each cell [2 voltages]
    - Faster and more reliable
  - MLC: Multi Layered Chip
    - Stores 00/01/10/11 in each cell [4 voltages] ➔ 2 bits per cell
    - Fails 10 times sooner!
    - Cost Advantage of roughly 2 to 1.
  - Manufacturing process almost same for both SLC and MLC.

Quick Look at various operations on FLASH

	Read	Erase	Program
Granularity	Page	Block	Set of Pages with in a Block (entire Block need not be programmed at once.)
Access	Random	Random blocks within a Chip	❑ Any block in a chip can be programmed randomly ❑ Set of pages with in a block need to be programmed sequentially
Time	< 0.1ms	1.5ms	>= 0.3ms
'Wear Out'?	No	Yes	No (but you can program only once before erase)
Bit Change	-	Changes all Bits ➔'1'	Only op to change Bits ➔ '0'
Other Notes		❑ 'Stresses' the block and will eventually cause it to fail. SLC Chips: 98% of blocks will last at least 100,000 erase cycles ❑ Large systems have 'wear-leveling' algorithms built in so that 'system write' has high endurance than that of write to each cell	

Example: Sample Write Operation (source: Texas Memory Systems 2015)

- Writing one 8KB random write would require the following steps:

Step 1	Read 128KB (1 block)	**0.1ms + transfer time** **(trsfr time = 128KB/(20MB/s) = 6.25ms)**	6.4ms
Step 2	Erase 128KB Block		1.5ms
Step 3	Program 128KB Block (with 120KB of unchanged data and 8KB of new data)	**0.3ms + 6.25ms + (?)**	6.7ms
Data Transfer rate of this FLASH chip ~ 20MB/s		**Total Time**	**14.6ms**

- **Write Performance (of 14-15ms)**
  - Acceptable in consumer markets line thumb (USB) drives
  - Not Acceptable in Enterprise markets ➔ *one of the reasons for not deploying Flash drives in enterprises*

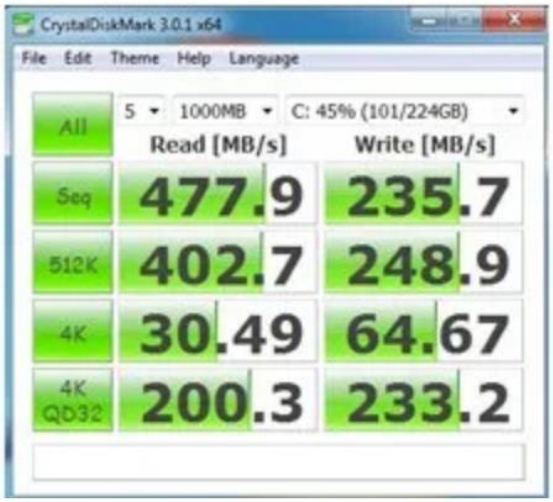
Difference between SSDs and traditional Hard Drives

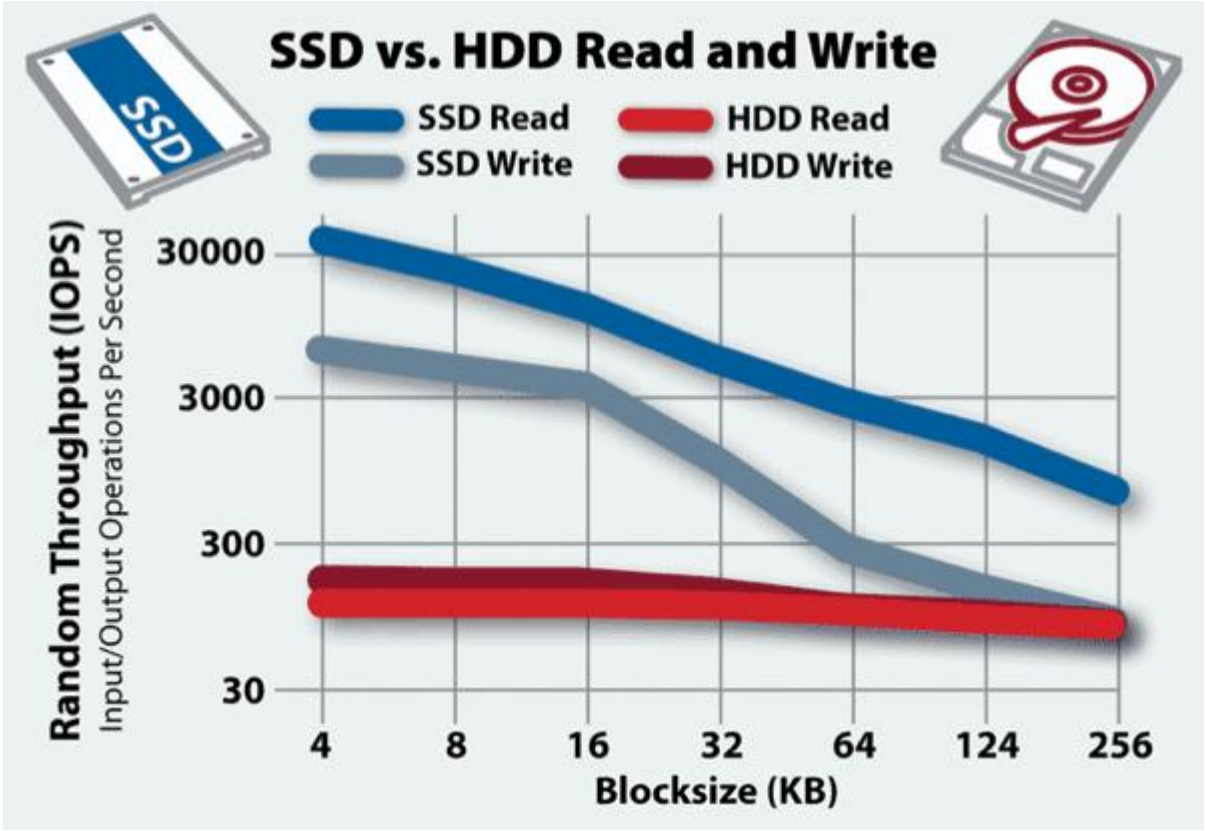
	Hard Drive	Solid State Drive
Construction/ Data Organization	Mechanical Motor/Spindle, Head, Cylinders. Data organized as Tracks and Sectors	FLASH Memory. Constructed using 'blocks'. No spinning. Electronic Reading and Writing.
Reads	❑ Slow Random Reads	Fast Random/Sequential Reads
	❑ Pre-fetching to help Sequential Reads (initiated by OS/present in Disk itself)	——
Writes	Slow Random Writes	Slow Random Writes (slower than current state-of-the-art Hard Disks) → 14-15ms for 128KB.
	Disk Cache/OS Coalescing could help buffer a few writes	Remapping blocks to convert random writes to sequential writes is one trend
Failures	Complete Disk Failure	Block Failure
Recovery/Failure prevention method	Implemented at an array level. RAID is the normal way to re-construct failed disks	Can be done at a disk level or array level ❑ Current methods at a disk level
	——	Wear-leveling algorithms to reduce possible failures of blocks
Striping for arrays	Typically Large Stripe Sizes preferred	Smaller Stripe Sizes preferred

Performance comparison (2018)

Hard Drive

SSD





I/O Scheduler Changes

- Since there is no seek time and I/O to any block is equal latency, I/O schedulers optimize for write reduction.
- Imagine writing a multiple consecutive 4KB blocks (since that's what the operating system will assume)
- 1st 4KB: read 128K, erase 128K, prog 128K
  2nd 4KB: read 128K, erase 128K, prog 128K
  :
- Reduces efficiency. I/O Scheduler should buffer requests for a "bit" assuming there might subsequent write request.
- Coalesce into a fewer "read/erase/program cycle"

Current Industry Trends…

- Increasing Performance (Read/Write)
  - Array of FLASH memories to utilize the parallel bandwidth and reduce the latencies
  - A cache (DDR RAM) in front of the array to further minimize the latencies
  - Optimum Stripe Sizes (transfer rate is important)
- Increasing Write Performance
  - Erase as a parallel background operation
- Increasing Endurance
  - A cache (DDR RAM) in front of the array to buffer writes (write-back cache)
  - Remapping for writes
  - Wear-leveling algorithms
  - Disks contain more space than advertised (of the order 20%)
- New technology hitting the market
  - NVMe : allows load/store operations (Intel Optane)
  - SSD behaves like memory → Storage class memory byte addressable