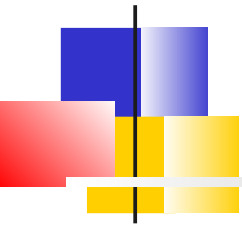
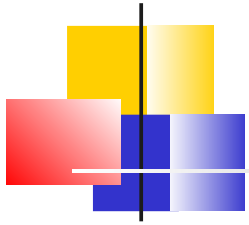


CAS CS 552
Intro to Operating Systems

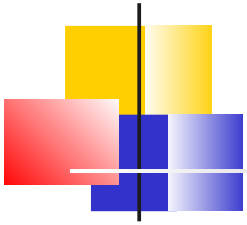
Richard West

Computer System Structures



Bootstrap Program

- Initial program run when system starts
  - Initializes system, CPU registers, memory, devices…
  - Loads OS kernel into main memory
  - E.g. GRUB, LILO, Das U-Boot, etc
  - OS then executes 1st process (in UNIX/Linux this is typically “init”) and then waits for events to occur
  - Events -> interrupts
    - h/w interrupts from devices to CPU
    - s/w interrupts, or traps, via syscalls and/or faults



The “Old” PC Disk Boot Sector

PC architecture assumes boot sector (or MBR) is 1st sector (S) on 1st track of a cylinder (C), under the first head (H)

- CHS geometry: 001 = logical block address (LBA) 0
- For CHS block address, LBA =

  (C * #heads+H) * #sectors/track + (S-1)
- At power on, the basic I/O system (BIOS) performs a power-on self-test (or POST)
  - 80386: CPU starts executing BIOS at 0xF000:FFF0
  - Modern x86: Starts reset vector at 0xFFFFFFF0 then jumps to 0xF0000
  - Drives (e.g., floppy, disk, cdrom) are checked for a valid boot sector having a signature 0AA55h at offset 510
  - NOTE: Sectors are 512 bytes

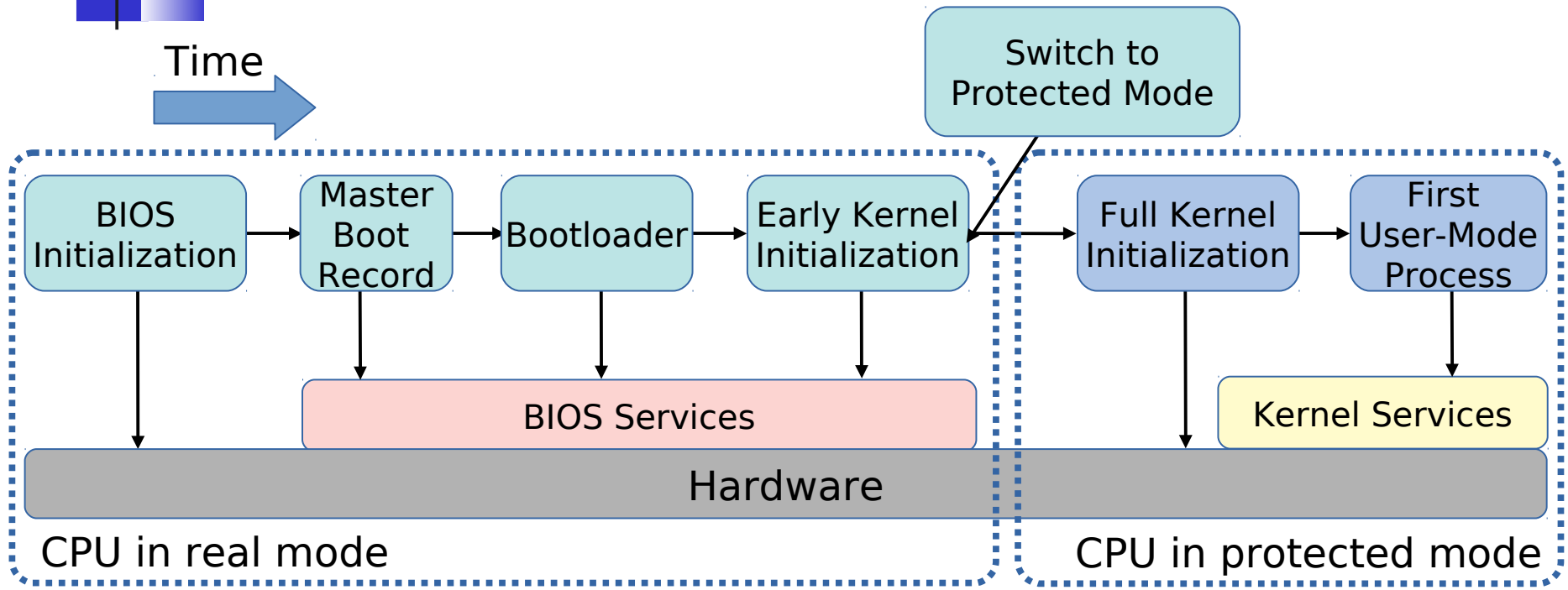
- BIOS reads valid boot sector from disk into memory at address 0:7C00h
  - Chainloading might cause boot sector (MBR) to relocate to another region e.g., 0:0600h, replacing 0:7C00h with new boot sector
- Boot sector code + data can then load kernel
  - Must be less than 510 bytes – size of space for master partition table having up to four primary partition entries

Example DOS Boot Sector

```
OFFSET 0 1 2 3  4 5 6 7  8 9 A B  C D E F  *0123456789ABCDEF*
000000 eb3c904d 53444f53 352e3000 02010100 *.<.MSDOS5.0.....*
000010 02e00040 0bf00900 12000200 00000000 *...@............*
000020 00000000 0000295a 5418264e 4f204e41 *......)ZT.&NO NA*
000030 4d452020 20204641 54313220 2020fa33 *ME    FAT12   .3*
000040 c08ed0bc 007c1607 bb780036 c5371e56 *.....|...x.6.7.V*
000050 1653bf3e 7cb90b00 fcf3a406 1fc645fe *.S.>|.........E.*
000060 0f8b0e18 7c884df9 894702c7 073e7cfb *....|.M..G...>|.*
000070 cd137279 33c03906 137c7408 8b0e137c *..ry3.9..|t....|*
000080 890e207c a0107cf7 26167c03 061c7c13 *.. |..|.&.|...|.*
000090 161e7c03 060e7c83 d200a350 7c891652 *..|...|....P|..R*
0000a0 7ca3497c 89164b7c b82000f7 26117c8b *|.I|..K|. ..&.|.*
0000b0 1e0b7c03 c348f7f3 0106497c 83164b7c *..|..H....I|..K|*
0000c0 00bb0005 8b16527c a1507ce8 9200721d *......R|.P|...r.*
0000d0 b001e8ac 0072168b fbb90b00 bee67df3 *.....r........}.*
0000e0 a6750a8d 7f20b90b 00f3a674 18be9e7d *.u... .....t...}*
0000f0 e85f0033 c0cd165e 1f8f048f 4402cd19 *._.3...^....D...*
000100 585858eb e88b471a 48488a1e 0d7c32ff *XXX...G.HH...|2.*
000110 f7e30306 497c1316 4b7cbb00 07b90300 *....I|..K|......*
000120 505251e8 3a0072d8 b001e854 00595a58 *PRQ.:.r....T.YZX*
000130 72bb0501 0083d200 031e0b7c e2e28a2e *r.........|....*
000140 157c8a16 247c8b1e 497ca14b 7cea0000 *.|..$|..I|.K|...*
000150 7000ac0a c07429b4 0ebb0700 cd10ebf2 *p....t).........*
000160 3b16187c 7319f736 187cfec2 88164f7c *;..|s..6.|....O|*
000170 33d2f736 1a7c8816 257ca34d 7cf8c3f9 *3..6.|..%|.M|...*
000180 c3b4028b 164d7cb1 06d2e60a 364f7c8b *.....M|.....6O|.*
000190 ca86e98a 16247c8a 36257ccd 13c30d0a *.....$|.6%|.....*
0001a0 4e6f6e2d 53797374 656d2064 69736b20 *Non-System disk *
0001b0 6f722064 69736b20 6572726f 720d0a52 *or disk error..R*
0001c0 65706c61 63652061 6e642070 72657373 *eplace and press*
0001d0 20616e79 206b6579 20776865 6e207265 * any key when re*
0001e0 6164790d 0a00494f 20202020 20205359 *ady...IO      SY*
0001f0 534d5344 4f532020 20535953 000055aa *SMSDOS   SYS..U.*
```

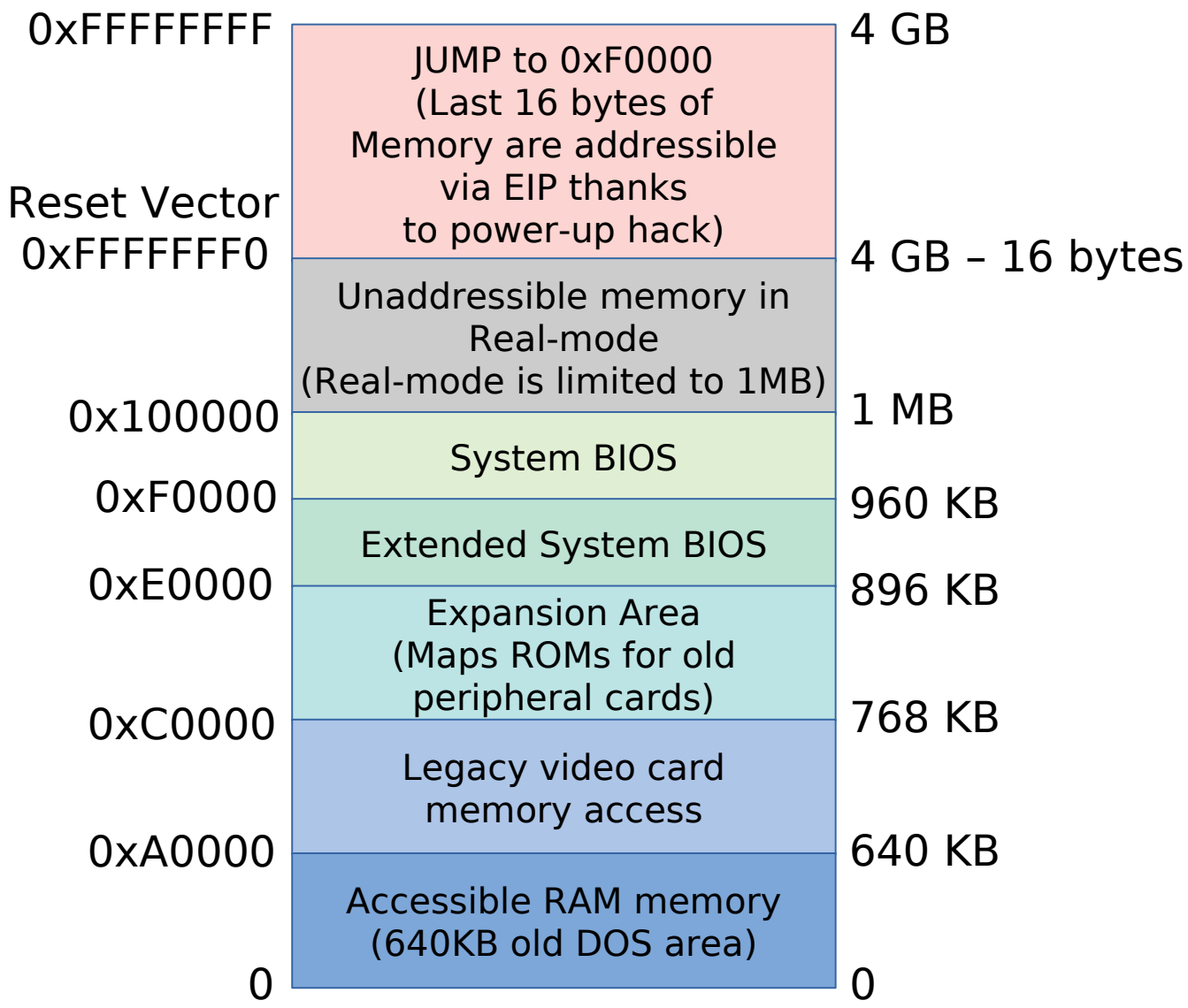
Boot signature

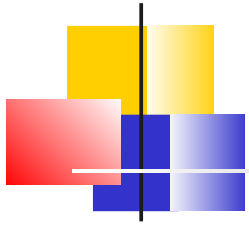
BIOS Booting Procedure



- Example BIOS functions using real-mode s/w interrupts:
  - INT 0x10 = Video display functions (including VESA/VBE)
  - INT 0x13 = mass storage (disk, floppy) access
  - INT 0x15 = memory size functions
  - INT 0x16 = keyboard functions

32-bit PC Memory Regions in Boot Procedure





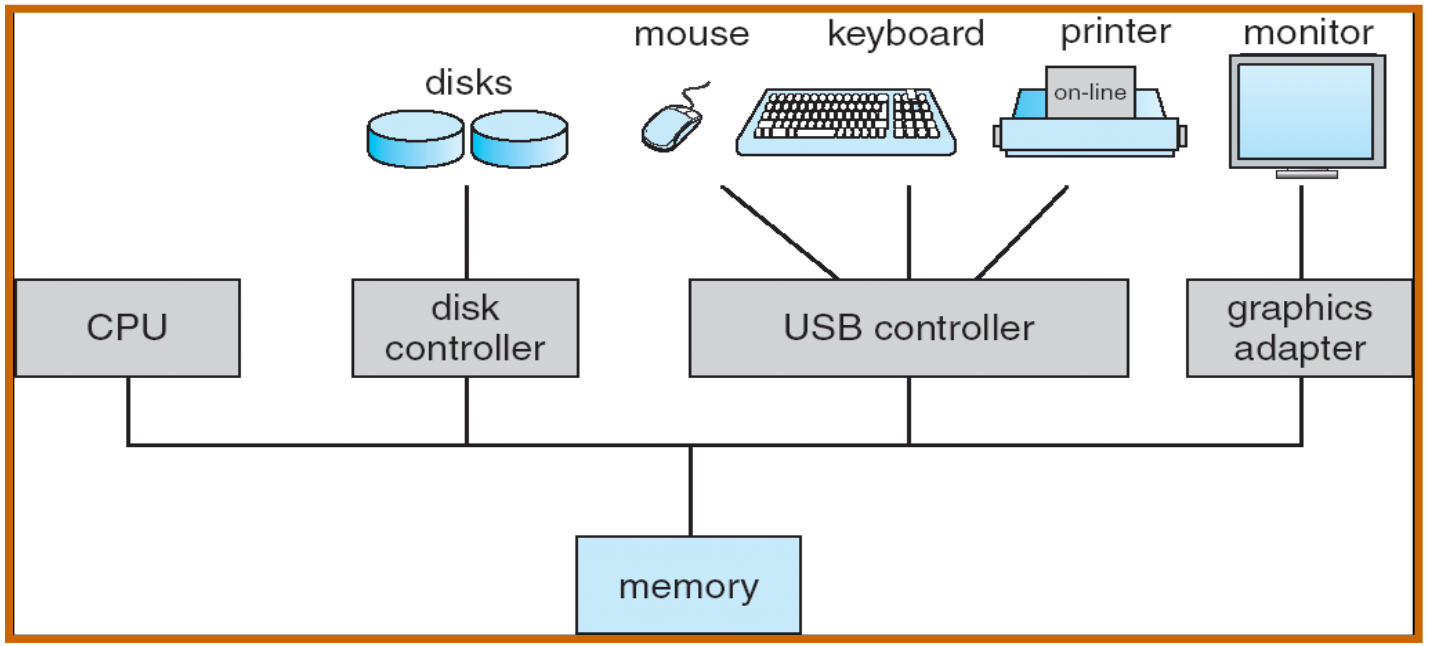
x86 Memory Map (Lower 1 MB)

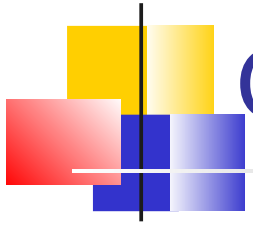
Start	End	Size	Type	Description
0x00000	0x003FF	1 KB	RAM	Real-mode IVT
0x00400	0x004FF	256 bytes	RAM	BIOS Data Area
0x00500	0x07BFF	~30 KB	RAM (Free for use)	Conventional memory
0x07C00	0x07DFF	512 bytes	RAM	OS boot sector
0x07E00	0x7FFFF	480.5 KB	RAM (Free for use)	Conventional memory
0x80000	0x9FFFF	128 KB	RAM (partially usable)	Extended BIOS Data Area
0xA0000	0xFFFFF	384 KB	Various (unusable)	Video memory, ROM area

See: https://wiki.osdev.org/Memory_Map_(x86)

Computer System Organization

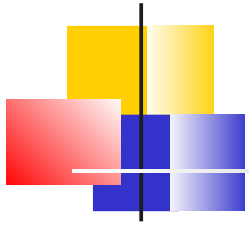
- Computer system: One or more CPUs, device controllers connected via a system bus to shared memory
- Memory accessed via memory controller (not shown)





Computer System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU (or DMA controller) moves data from/to main memory to/from local buffers
- I/O is between the device and main memory, via local buffers of controller
- Device controller informs CPU that it has finished its operation by causing an *interrupt*

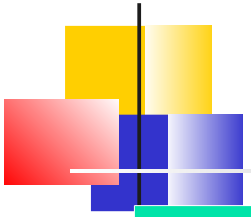


Common Causes of Interrupts

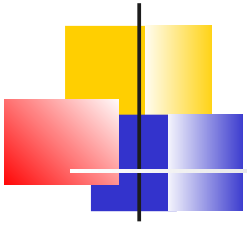
- I/O operation completes, but also…
- Program faults (divide by zero, floating-point error…)
- Bus error, address error, general protection fault, page fault etc
- Programmed exceptions such as system calls

- Syscalls:
  - Pass control to OS, to perform some “privileged” service on behalf of the caller

Traditional ISA Device Interrupts (IRQs) for IBM PC



IRQ	Description
0	Programmable Interrupt Timer Interrupt
1	Keyboard Interrupt
2	Cascade (used internally by the two PICs. never raised)
3	COM2 (if enabled)
4	COM1 (if enabled)
5	LPT2 (if enabled)
6	Floppy Disk
7	LPT1 / Unreliable spurious interrupt (usually)
8	CMOS real-time clock (if enabled)
9	Free for peripherals / legacy SCSI / NIC
10	Free for peripherals / SCSI / NIC
11	Free for peripherals / SCSI / NIC
12	PS2 Mouse
13	FPU / Coprocessor / Inter-processor
14	Primary ATA Hard Disk
15	Secondary ATA Hard Disk

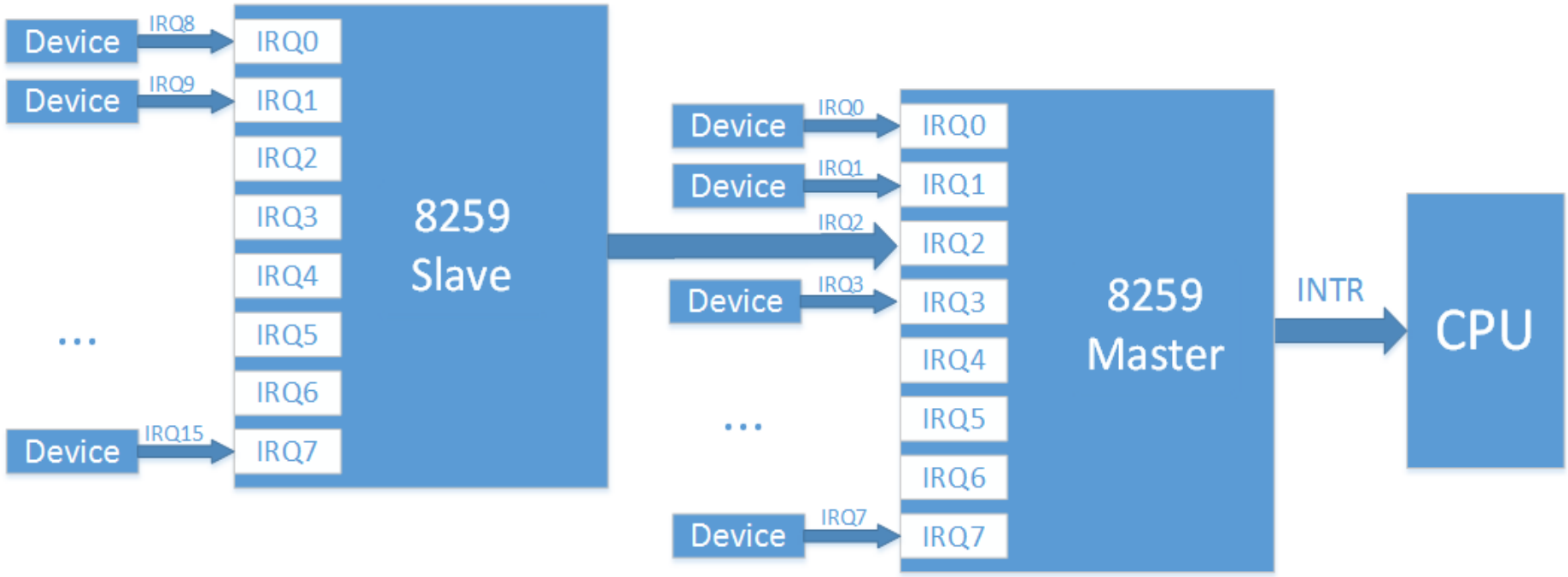


x86 Exceptions

Name	Vector nr.	Type	Mnemonic	Error code?
Divide-by-zero Error	0	Fault	#DE	No
Debug	1	Fault/Trap	#DB	No
Non-maskable Interrupt	2	Interrupt	-	No
Breakpoint	3	Trap	#BP	No
Overflow	4	Trap	#OF	No
Bound Range Exceeded	5	Fault	#BR	No
Invalid Opcode	6	Fault	#UD	No
Device Not Available	7	Fault	#NM	No
Double Fault	8	Abort	#DF	Yes
Coprocessor Segment Overrun	9	Fault	-	No
Invalid TSS	10	Fault	#TS	Yes
Segment Not Present	11	Fault	#NP	Yes
Stack-Segment Fault	12	Fault	#SS	Yes
General Protection Fault	13	Fault	#GP	Yes
Page Fault	14	Fault	#PF	Yes
...	31			

Example Interrupt Hardware

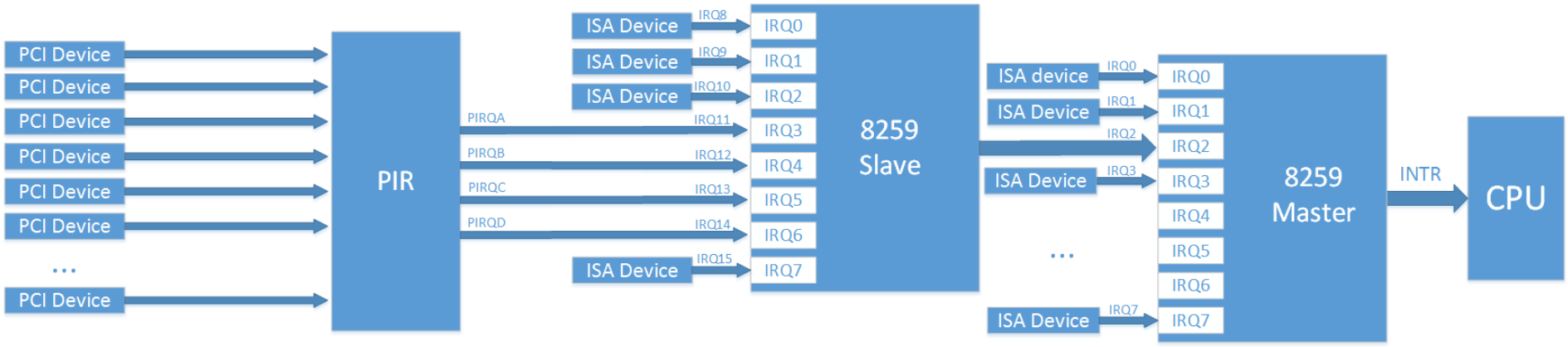
Two cascaded 8259 PICs for ISA device interrupts



Thanks to: https://habr.com/en/post/446312/

Example Interrupt Hardware

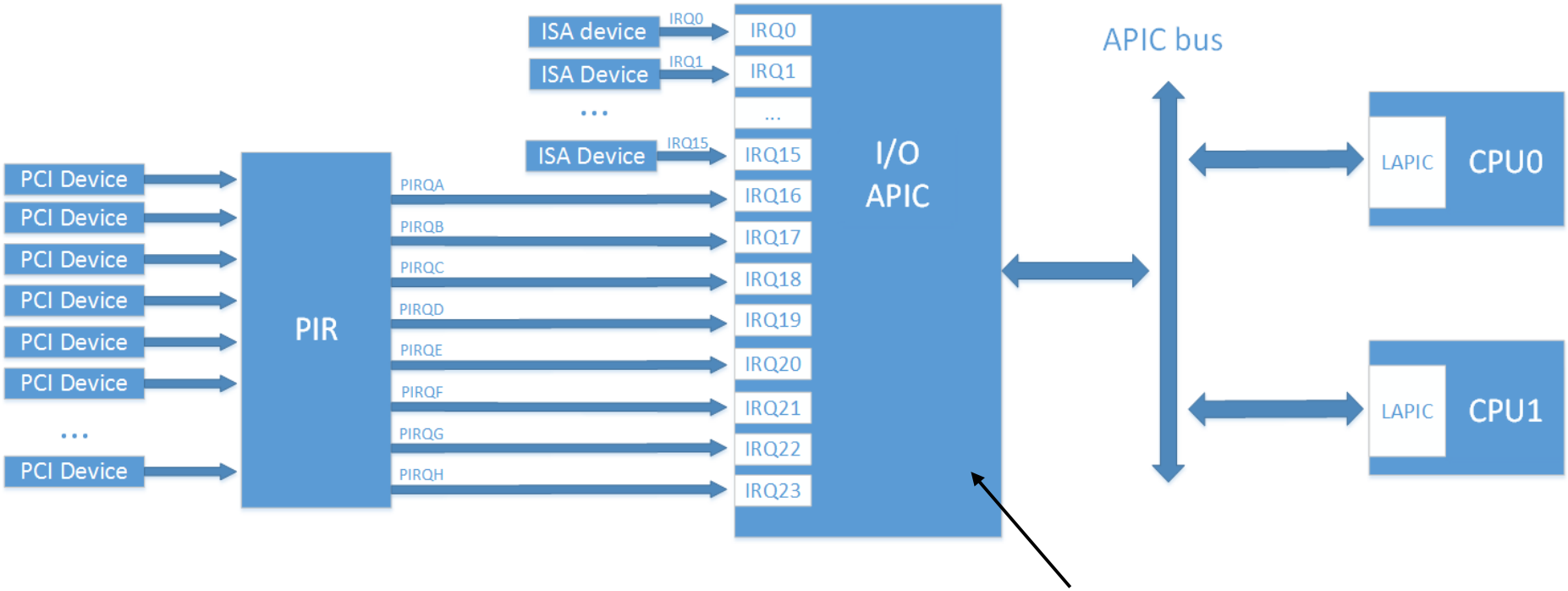
Extend with support for PCI device interrupt routing (PIR)



Thanks to: https://habr.com/en/post/446312/

Multiprocessor Interrupt Hardware (e.g., IOAPIC + LAPIC)

8259 PIC limited to one CPU! Need a new approach for multicore

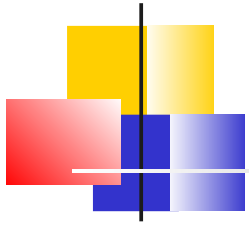


e.g., 82093AA
(up to 16 CPUs)

Thanks to: https://habr.com/en/post/446312/

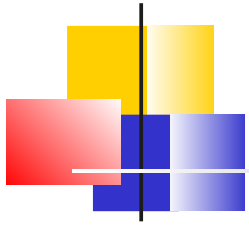
XAPIC + PCIe Architecture

- xAPIC backward compatible with APIC
- Supports 256 CPUs
  - Later, x2APIC extended support to 2^{32} CPUs
- PCI devices could have 4 interrupts (one per function), but later extended to 32 interrupts
- PCIe emulated PCI interrupts using MSIs



Interrupt Handling 1/2

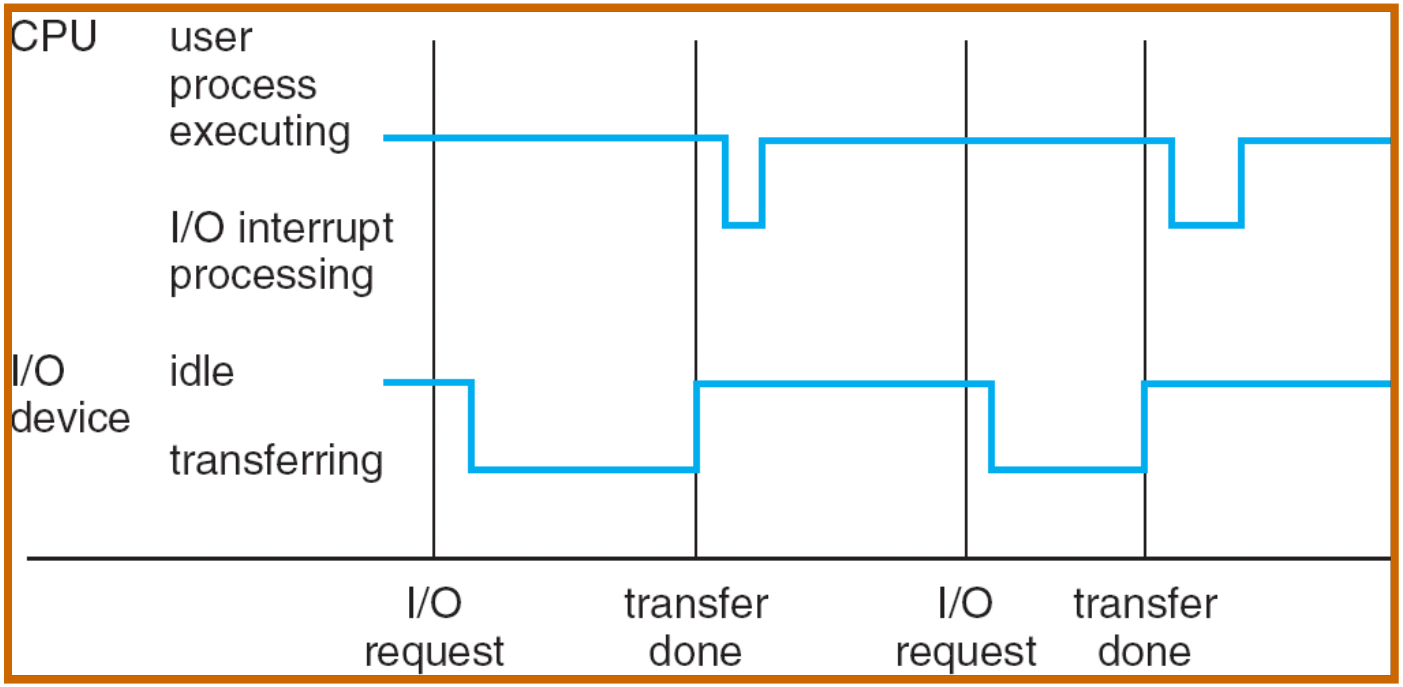
- When CPU is interrupted it stops current execution path and transfers control to the interrupt service routine
  - generally, via an *interrupt vector table*, which contains addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- Subsequent interrupts may be *disabled* while another interrupt is being processed to prevent a *lost interrupt*
- After servicing interrupt, CPU resumes old program execution (or “flow of control”)

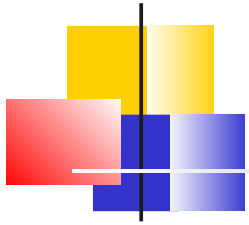


Interrupt Handling 2/2

- The operating system preserves the state of the CPU by storing registers and the program counter (a.k.a. instruction pointer)
  - Where is state saved?
- System determines which type of interrupt has occurred
  - E.g., via interrupt controller (e.g., 8259A) placing vector on data bus for CPU to access
- Separate segments of code determine what action should be taken for each type of interrupt
- After handling interrupt, saved return address is loaded into program counter, and old CPU state is restored, so that interrupted computation can resume

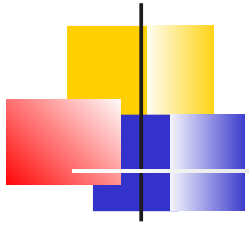
Interrupt Timeline





I/O Structure

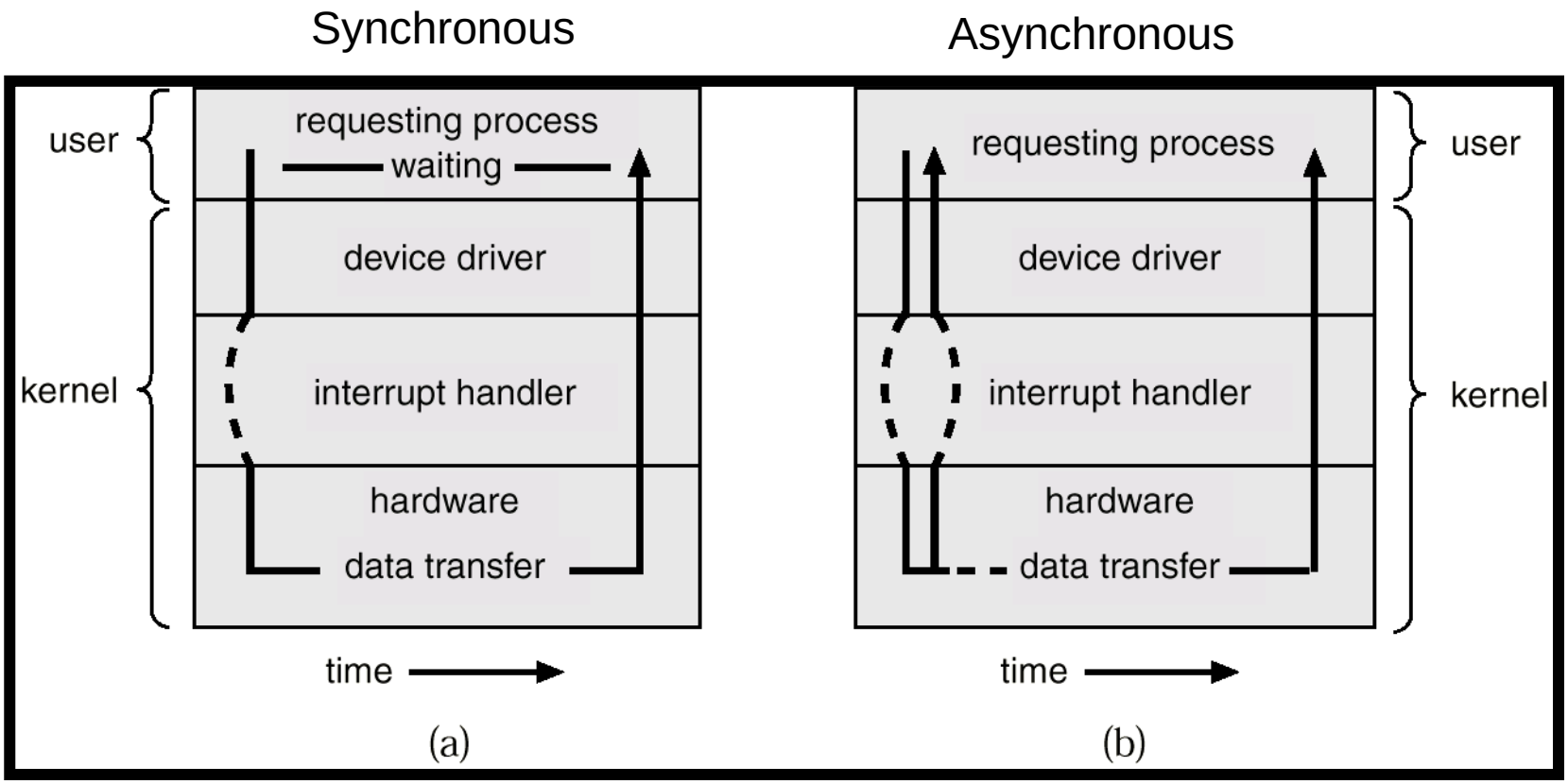
- To start an I/O operation, CPU loads data into device controller registers
  - Device controller reads registers to determine necessary action (e.g., read request from device to device controller buffer)
  - On completion of action, device controller interrupts CPU

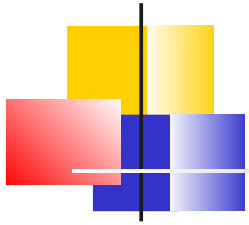


Synchronous vs Asynchronous I/O

- Synchronous I/O – user process blocks waiting for I/O to complete, after which it resumes execution
  - Kernel passes control back to user-level process on I/O completion
- Asynchronous I/O – user process continues execution while I/O request is being processed
  - Kernel passes control back to user-level process without waiting for I/O to complete

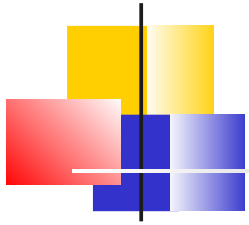
Two I/O Methods





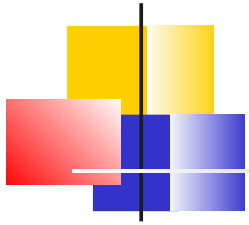
Polling

- Polled devices – CPU repeatedly checks the status of a device by e.g., checking a device register to see if the device is ready/busy (i.e., completed I/O request or not)
- Device status table – used by OS to keep track of I/O requests
  - Entries in table record device type, address and state (busy or idle…)
  - OS also maintains a wait queue for multiple requests to a specific device



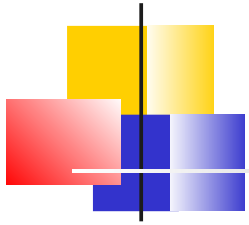
Direct Memory Access

- e.g., characters typed to a 9600-baud (bits/second) terminal
  - 9600 baud ~ 1000 chars/sec, or 1 char/millisecond
  - An interrupt service routine (ISR) may require 2µS/char to read each character into an input buffer
    - Therefore, 998µS every 1000µS can be used for other computations
    - Not much CPU overhead from interrupts!
- Faster devices (e.g., disks) would interrupt CPU too frequently
  - E.g., SATA can transfer at speeds of 3-6Gbps
  - CPU would spend most of its time processing interrupts and nothing else
  - Need for Direct Memory Access



Direct Memory Access Structure

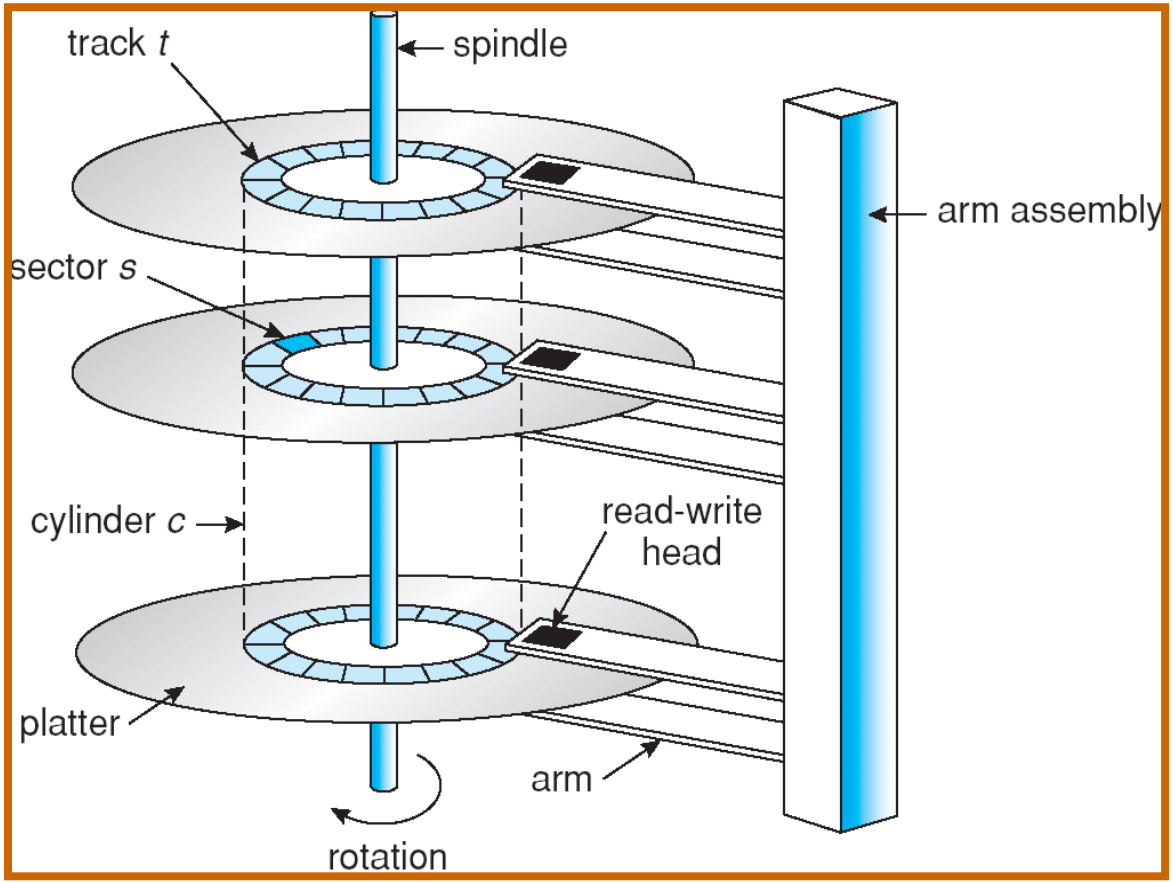
- Used for high-speed I/O devices able to transmit information at close to memory speeds
- After system (device driver) initializes memory location (for source/destination of DMA transfer) and count of bytes (or blocks) to transfer, device controller transfers data between buffer storage and main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte

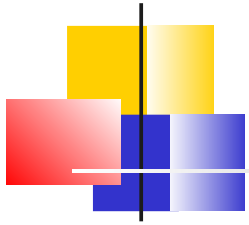


Storage Structure

- Main memory – only large storage media that the CPU can access directly
- Secondary storage – extension of main memory that provides large non-volatile storage capacity
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into *tracks*, which are subdivided into *sectors*
  - The *disk controller* determines the logical interaction between the device and the computer

Secondary Storage e.g., Magnetic Disks



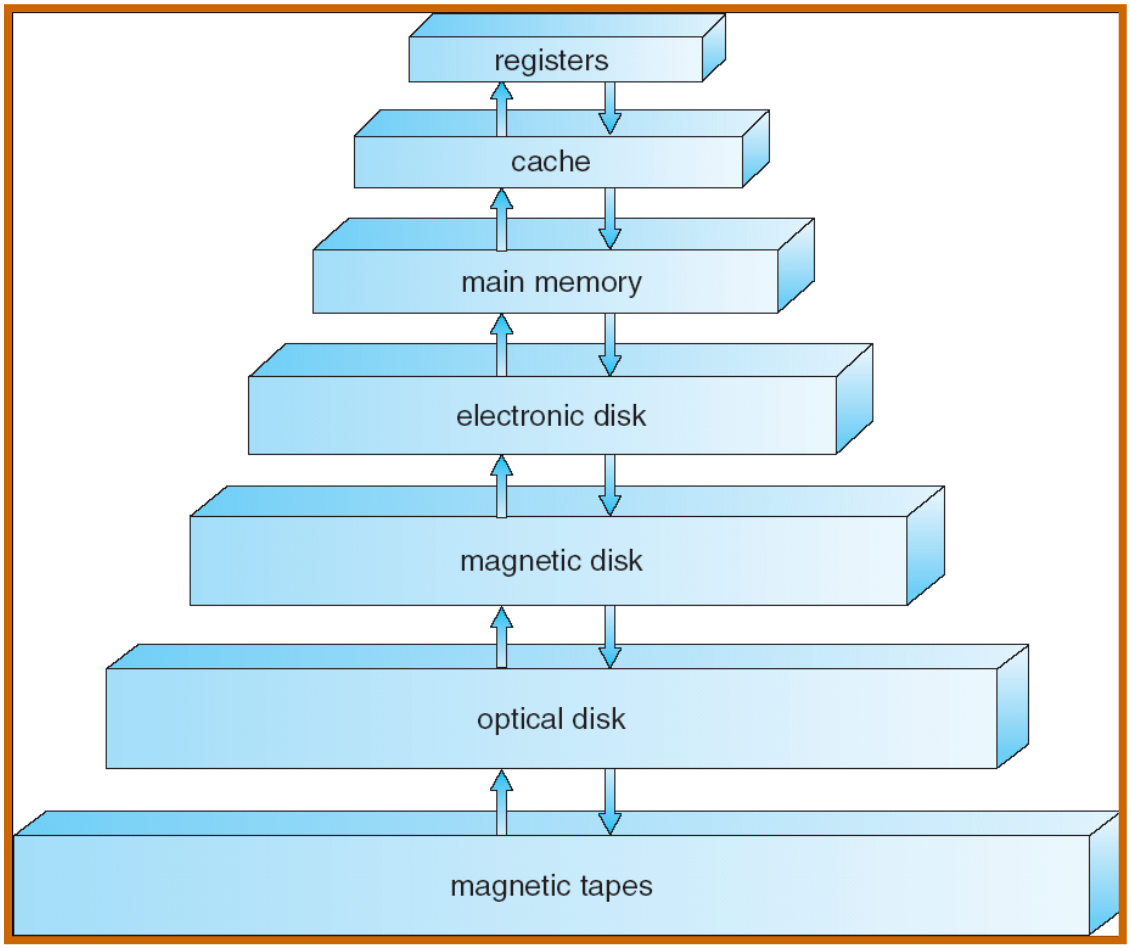


Storage Hierarchy

- Storage systems organized in a hierarchy according to:
  - Speed
  - Cost
  - Volatility
  - Capacity

- *Caching* – copying information into faster storage system
  - main memory can be viewed as a last *cache* for secondary storage

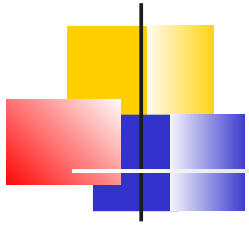
Storage-Device Hierarchy



Performance of Various Levels of Storage

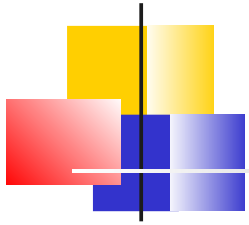
- Movement between levels of storage hierarchy can be explicit or implicit

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape



Caching 1/2

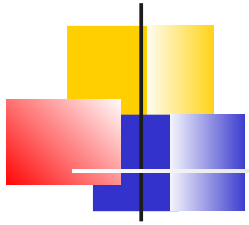
- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy



Caching 2/2

- Effectively a temporary, “fast” storage for information with high probability of access
  - Spatial and temporal locality
- Check cache before checking slower memory at lower level in the memory hierarchy
- Nowadays, CPUs have instruction and data caches

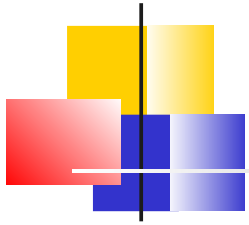
- On multiprocessor systems, each CPU has a cache
  - Data in one cache may differ from replica in another cache
  - Cache coherency is an issue – for program correctness may need all cached copies of a data item to have the same value (i.e., be consistent)



Hardware Protection

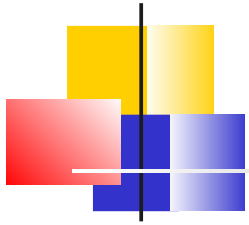
- OS must prevent one program from adversely affecting execution of another program
  - Must also protect itself from malicious programs
- We've seen that h/w can detect program errors e.g., illegal instructions, or attempts to access memory addresses outside process address space

  - Hardware can generate traps to the OS, which may terminate offending program
    - Signals are a way for programs to be informed of various kernel events triggered by interrupts



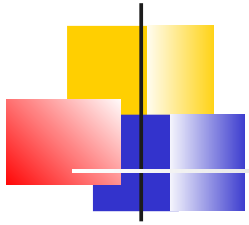
Dual-mode Protection

- Dual mode: user-level and kernel-level
  - Kernel-level often termed monitor/privileged/system/ supervisor mode
- Hardware mode bit keeps track of current mode bit during execution of a given sequence of instructions
  - IA32 architecture has TWO mode bits
- Traps (i.e., faults or programmed exceptions such as syscalls) switch the mode to kernel mode
  - Once in the kernel, code is executed at the most privileged “ring of protection”
  - Only kernel can execute most privileged instructions
  - ASIDE: MSDOS written for processors such as Intel 8088 which had no mode bit, so a user could potentially overwrite the (unprotected) OS



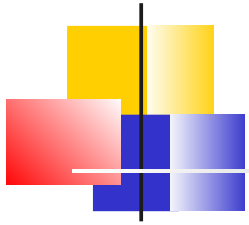
I/O Protection

- I/O instructions are typically executed by the OS
  - e.g., in/out instructions on various “ports” associated with devices
  - See example from Quest
  - i.e., they are privileged, although user-level can be granted access to certain I/O operations (e.g., by raising the I/O privilege level)
  - Why are I/O instructions usually privileged?



Memory Protection

- Interrupt vectors and ISRs should not be modified by untrusted user-level programs
- Address space of OS needs to be protected from untrusted user-level programs
- User-level programs need to be protected from each other
- Need memory protection!
- OS can leverage hardware support to establish ranges of memory addresses for separate user-level processes and the kernel
  - E.g., segmented memory support of IA32 allows base & limits values to be established on the ranges of memory addresses in a given memory region

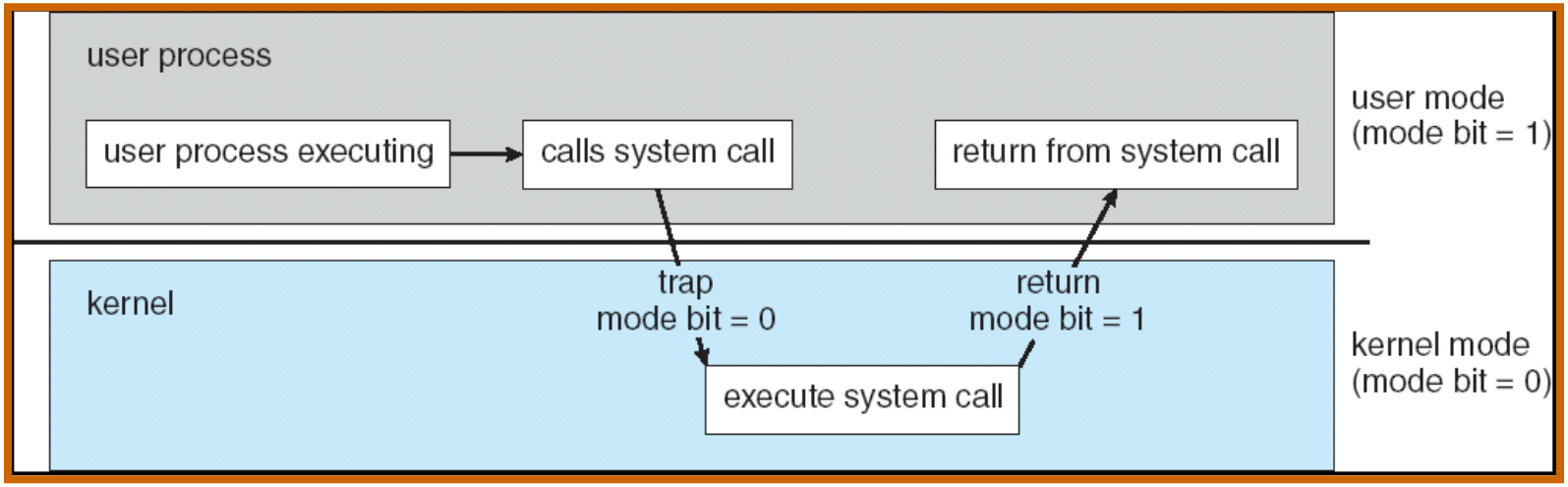


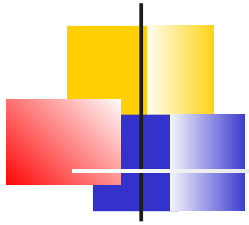
CPU Protection

- Need to prevent programs looping forever i.e., unduly hogging CPU
- A timer interrupt is set by OS to pass control back to the kernel after a program has executed for a specific quantum of time
  - i.e., a timeslice has expired
  - Timer interrupts effectively timeslice program execution, so that multiple programs can interleave their use of the CPU

Transition from User to Kernel Mode

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter is zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time



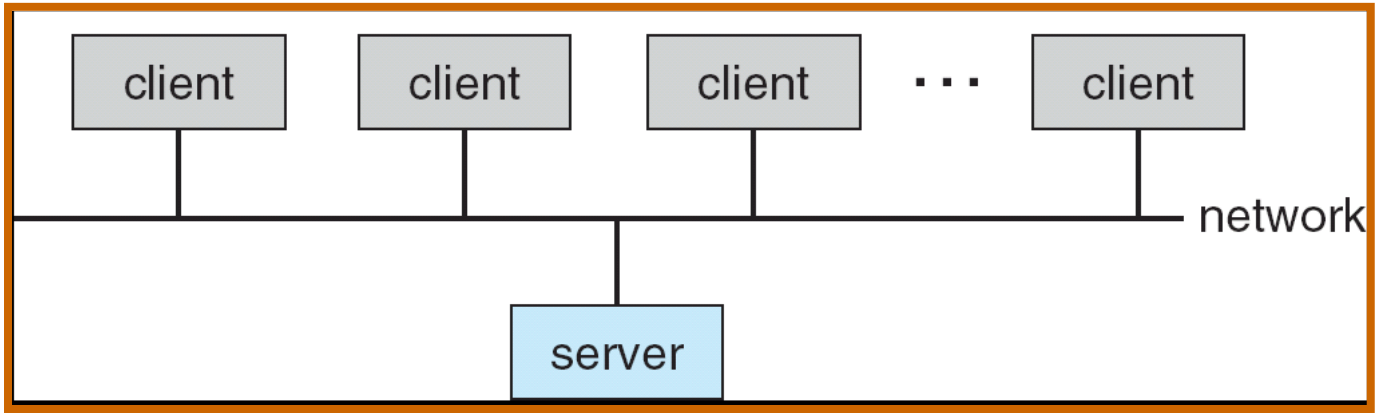


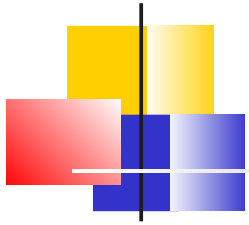
Computing Environments

- Traditional computer
  - Blurring over time
  - Office environment
    - PCs connected to a network, terminals attached to mainframe or minicomputers providing batch and timesharing
    - Now portals allowing networked and remote systems access to same resources
  - Home networks
    - Used to be single system, then modems
    - Now firewalled, networked

Computing Environments (Cont.)

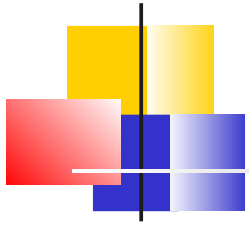
- Client-Server Computing
  - Dumb terminals supplanted by smart PCs
  - Many systems now **servers**, responding to requests generated by **clients**
    - **Compute-server** provides an interface to client to request services (i.e. database)
    - **File-server** provides interface for clients to store and retrieve files





Peer-to-Peer Computing

- Another model of distributed system
- P2P does not distinguish clients and servers
  - Instead all nodes are considered peers
  - May each act as client, server or both
  - Node must join P2P network
    - Registers its service with central lookup service on network, or
    - Broadcast request for service and respond to requests for service via *discovery protocol*
  - Examples include *Napster* and *Gnutella*



Web-Based Computing

- Web has become ubiquitous
- PCs most prevalent devices
- More devices becoming networked to allow web access
- New category of devices to manage web traffic among similar servers: **load balancers**
- Use of operating systems like Windows 95, client-side, have evolved into Linux and Windows XP, which can be clients and servers