CSE 325-Operating Systems

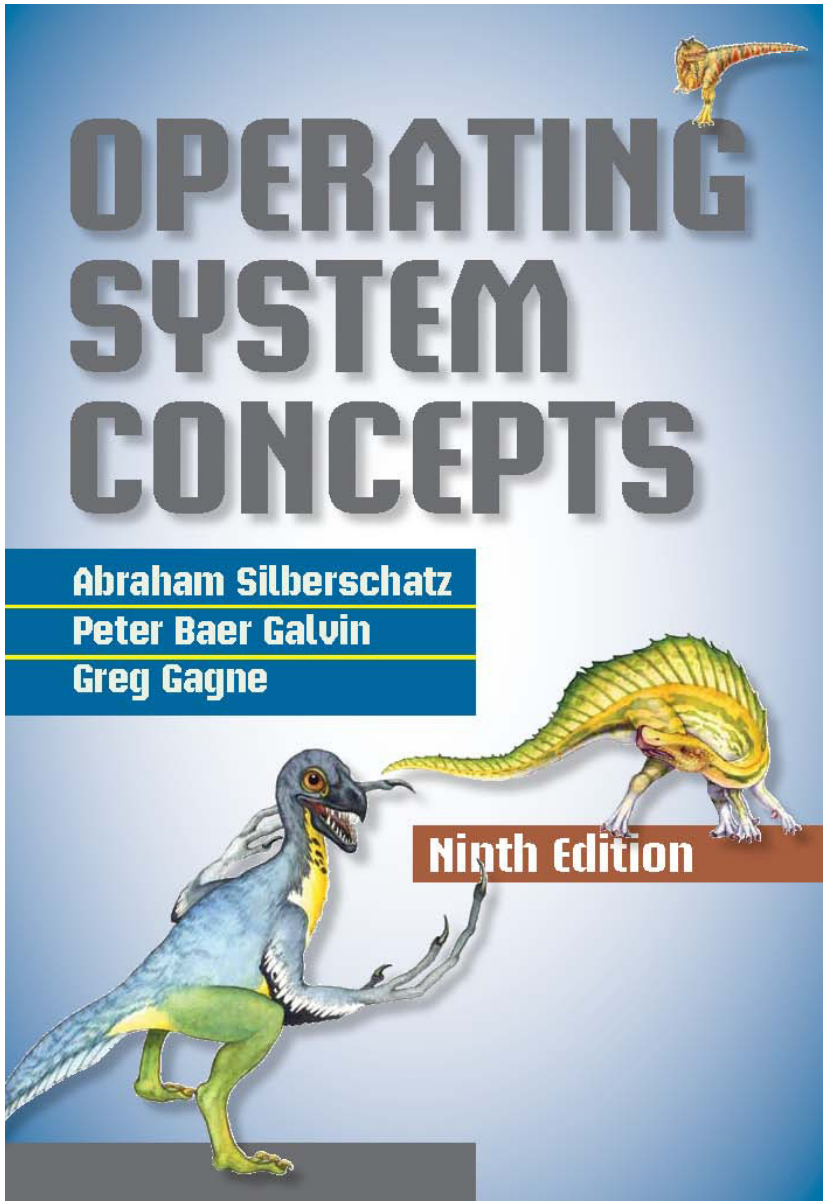
Lecture: 1-4

Dr. Shamim Akhter

Associate Professor

Computer Science and Engineering

EAST WEST UNIVERSITY


OPERATING SYSTEM CONCEPTS
Abraham Silberschatz
Peter Baer Galvin
Greg Gagne
Ninth Edition

Text book

Lecture 1: Introduction

Objective

Overview of the major OS components.

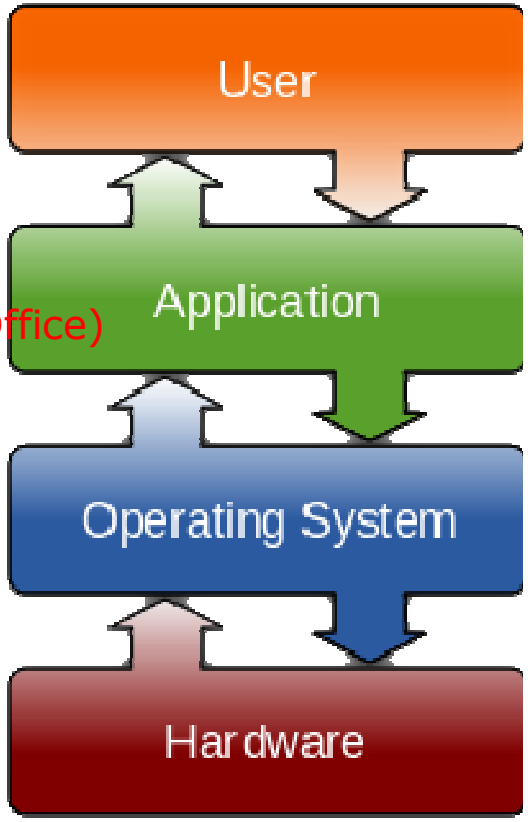
Components of a Computer System

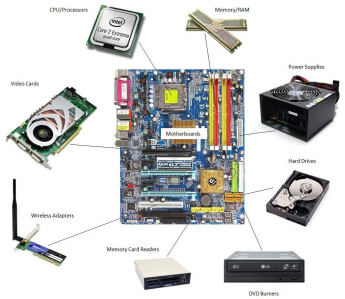
Banking system	Airline reservation	Web browser
Compilers	Editors	Command interpreter
Operating system		
Machine language		
Microarchitecture		
Physical devices		

Application programs
services to the user
(e.g. Web Browser. MS Office)

System programs
services to the hardware
(Disk Driver, assembler, Linker, loader)

Hardware



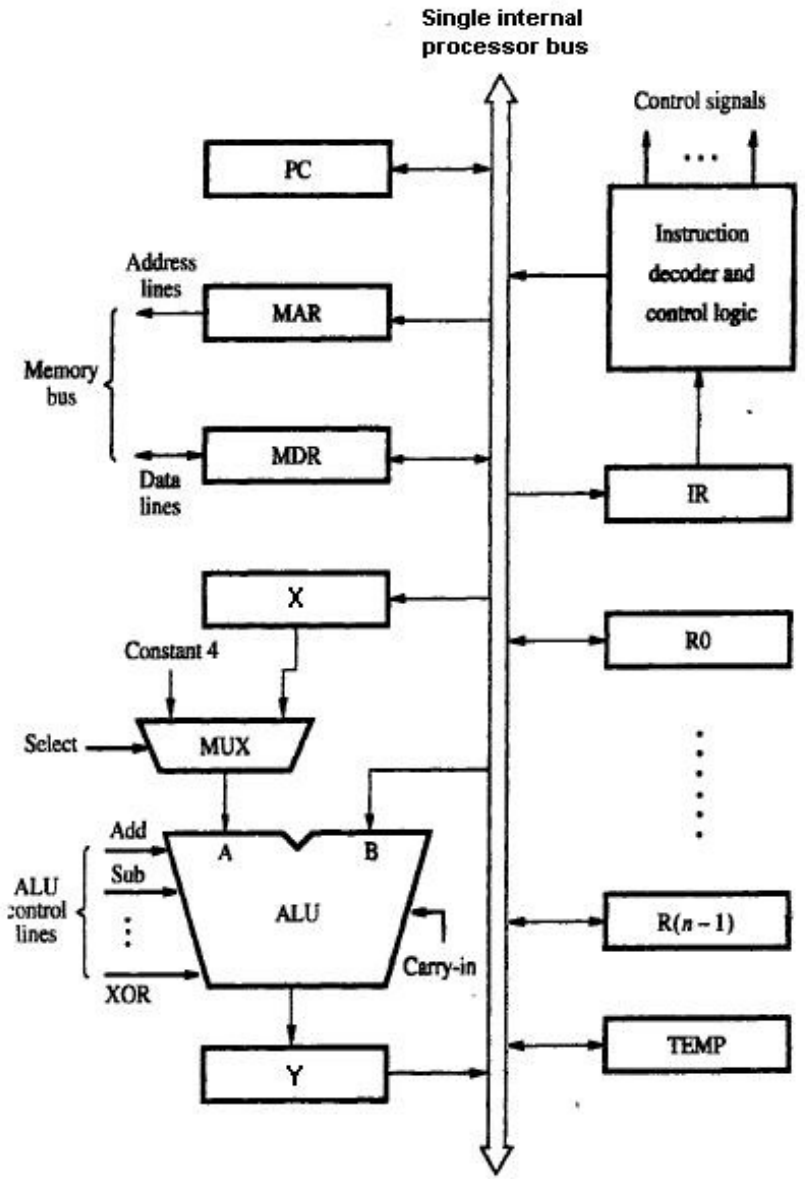


Microarchitecture- Computer Organization

- The way a given instruction set architecture (ISA) is implemented on a processor

The ISA includes :
- the execution model,
- processor registers,
- address and data formats.

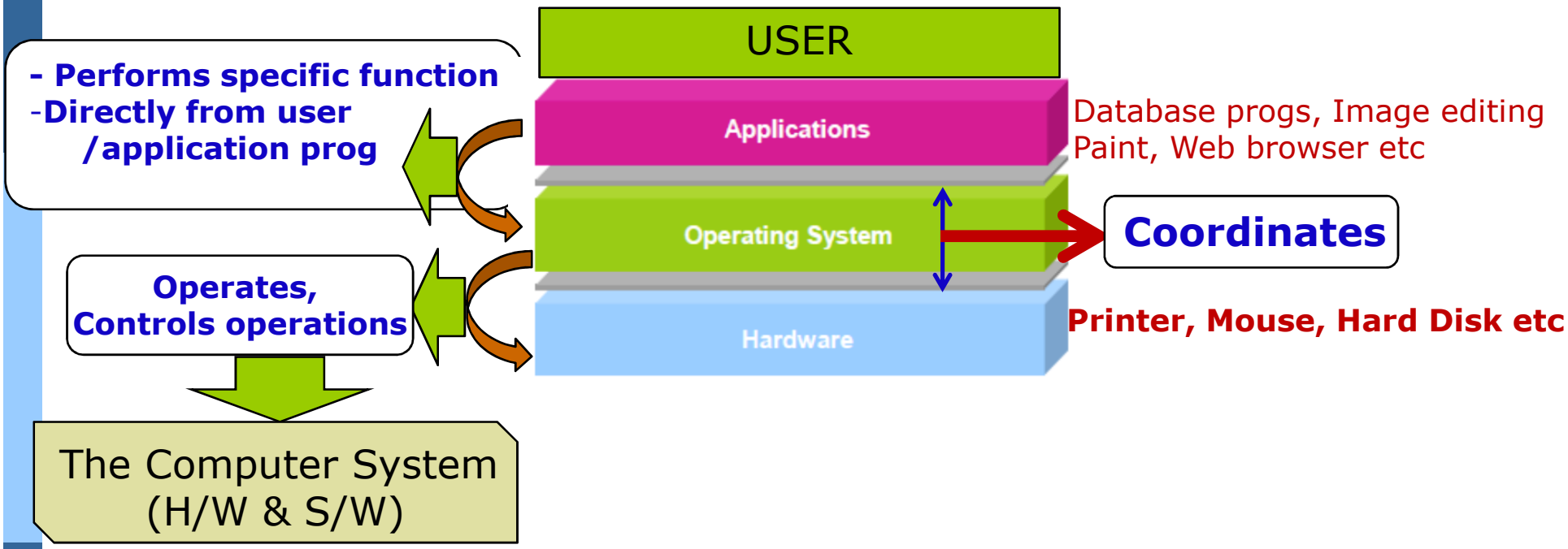
The microarchitecture includes:
- the parts of the processor and
- how these interconnect and
- interoperate to implement ISA.



Single Bus Organization

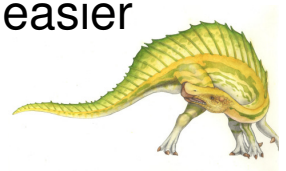
What is an Operating System?

- A program that acts as an intermediary betⁿ a user & H/W



GOALS:

- Convenient to use
- Execute user programs & make solving user problems easier
- Use H/W in an efficient manner



Operating System Definition

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use.

One goal of the operating system is to increase the utilization of resources.

Utilization =useful time/total time

For example, the OS should avoid wasting CPU time because the disk is rotating or wasting switching among tasks. Waiting for I/O.

- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer. e.g. Memory Protection.

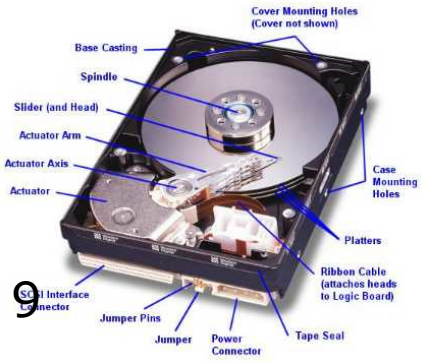
An example comparing life with/without OS

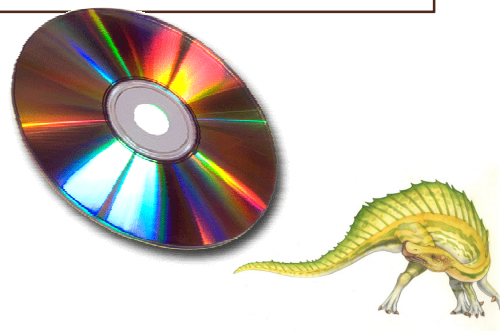
Life with an OS

```
file = open ("test.txt",
   O_WRONLY);
write (file, "test", 4);
close (file);
```

Life without an OS

- Blocks, platter, track, and sector
- Where is this file on disk? Which platter, track, and sectors?
- Code needs to change on a different system





How does a program execute?

- C Program

```
int A[2]; int C[2]; int B[2];
P=A[0];
Q=B[0];
C[0]=P+Q;
```

- Assembly

```
Load $t0, [A]
Load $t1, [B]
ADD $t3, $t0, $t1
STORE $t3, [C]
```

- Executable

```
0001……..00
0000……..01
00011……11
100……11
```

First.c → **Compile** → First.asm → First.o → **Linker** → Executable

Loader

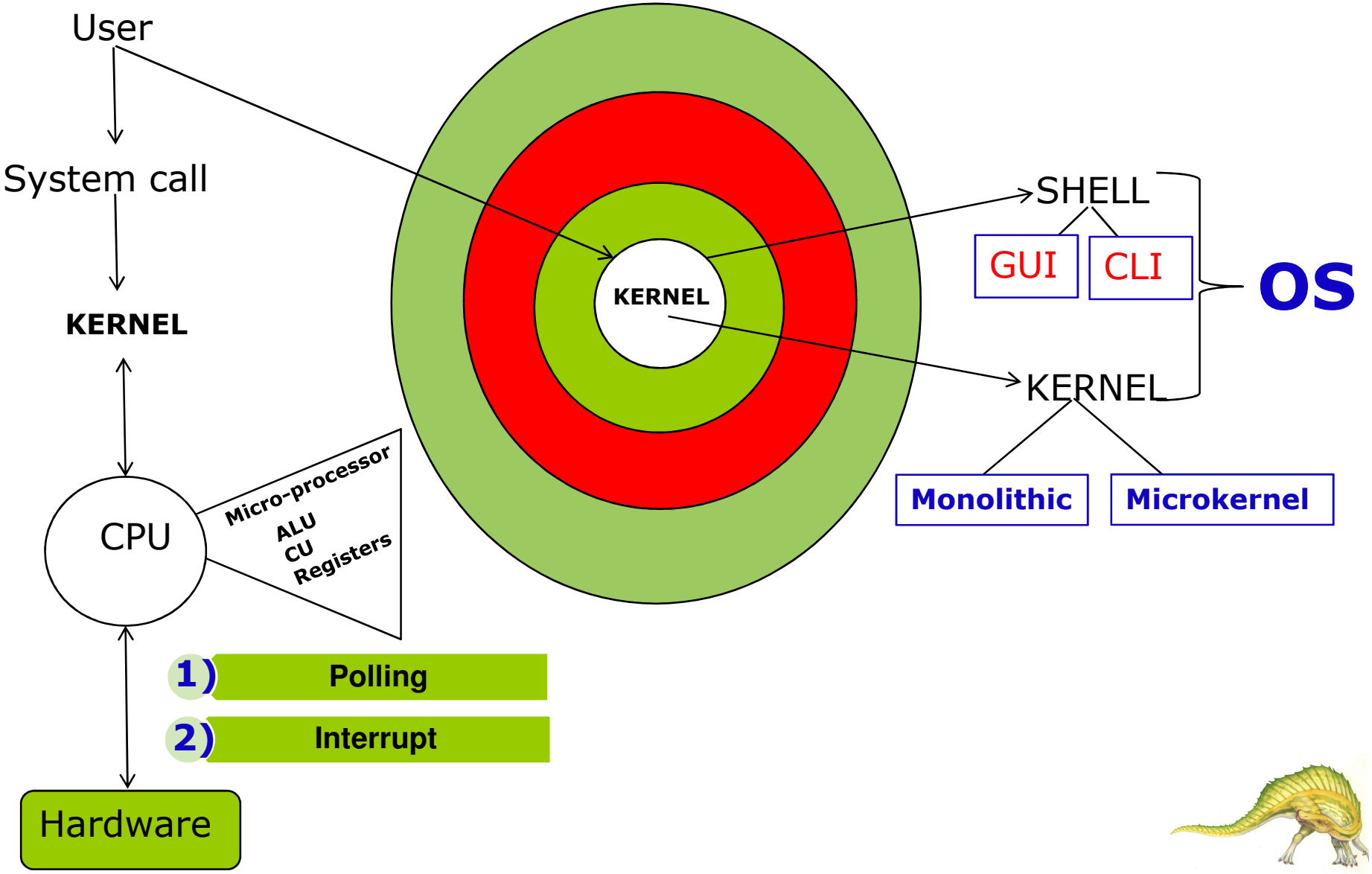
- Memory

0	0001……..00
4	0000……..01
8	00011……11
12	100……11

PC=0

IR= 0001……..00

How does user communicate with OS

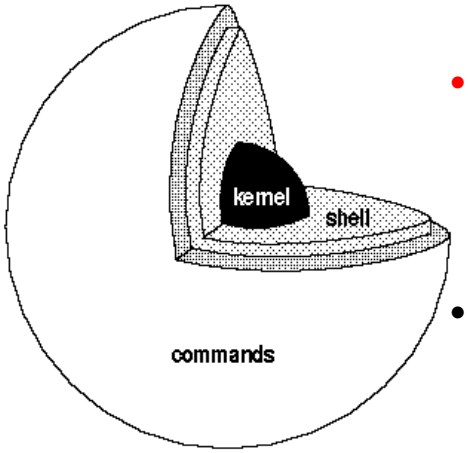


The Kernel and The Shell

An Operating system has two parts: kernel and shell

KERNEL (resides in main memory):

- is the heart and soul of an OS. Useful applications and utilities are added over the kernel, then the complete package becomes an OS.
- directly controls the computer hardware by performing OS services **using specific system calls or requests or hardware interrupts.**
- hides the hardware details from the user and application programs.



Example,

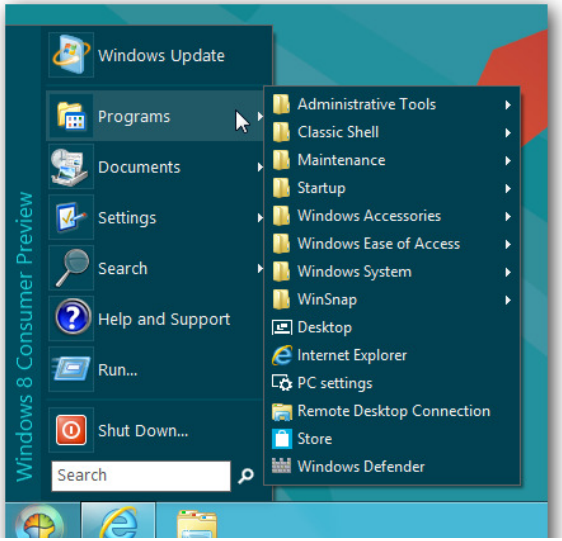
- Linux is a kernel as it does not include applications-file-system utilities, windowing systems and graphical desktops, system administrator commands, text editors, compilers etc.
- So, various companies added these kinds of applications over Linux kernel and provide their operating system like **ubuntu, centOS, redHat** etc.

An Operating system has two parts: kernel and shell

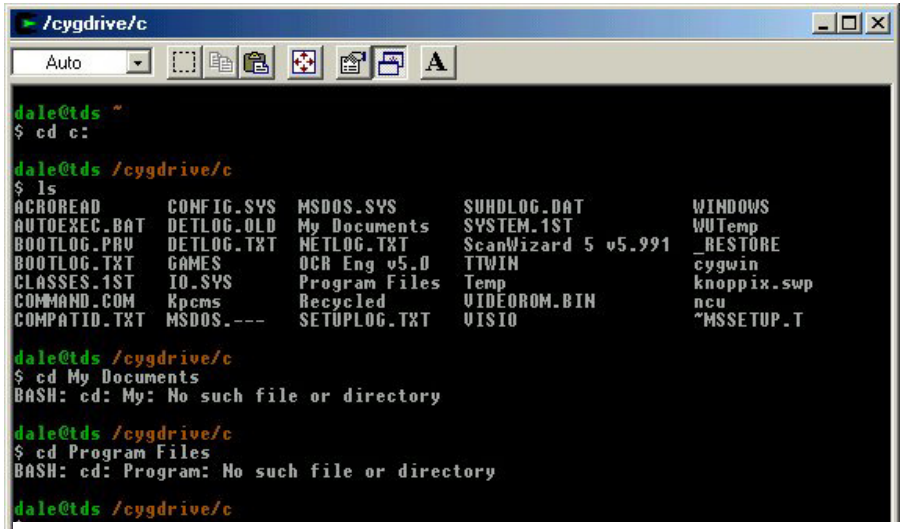
shell (command interpreter): Part of OS

- Serves as the interface between user and kernel.
- User enters commands via shell in order to use the kernel service.

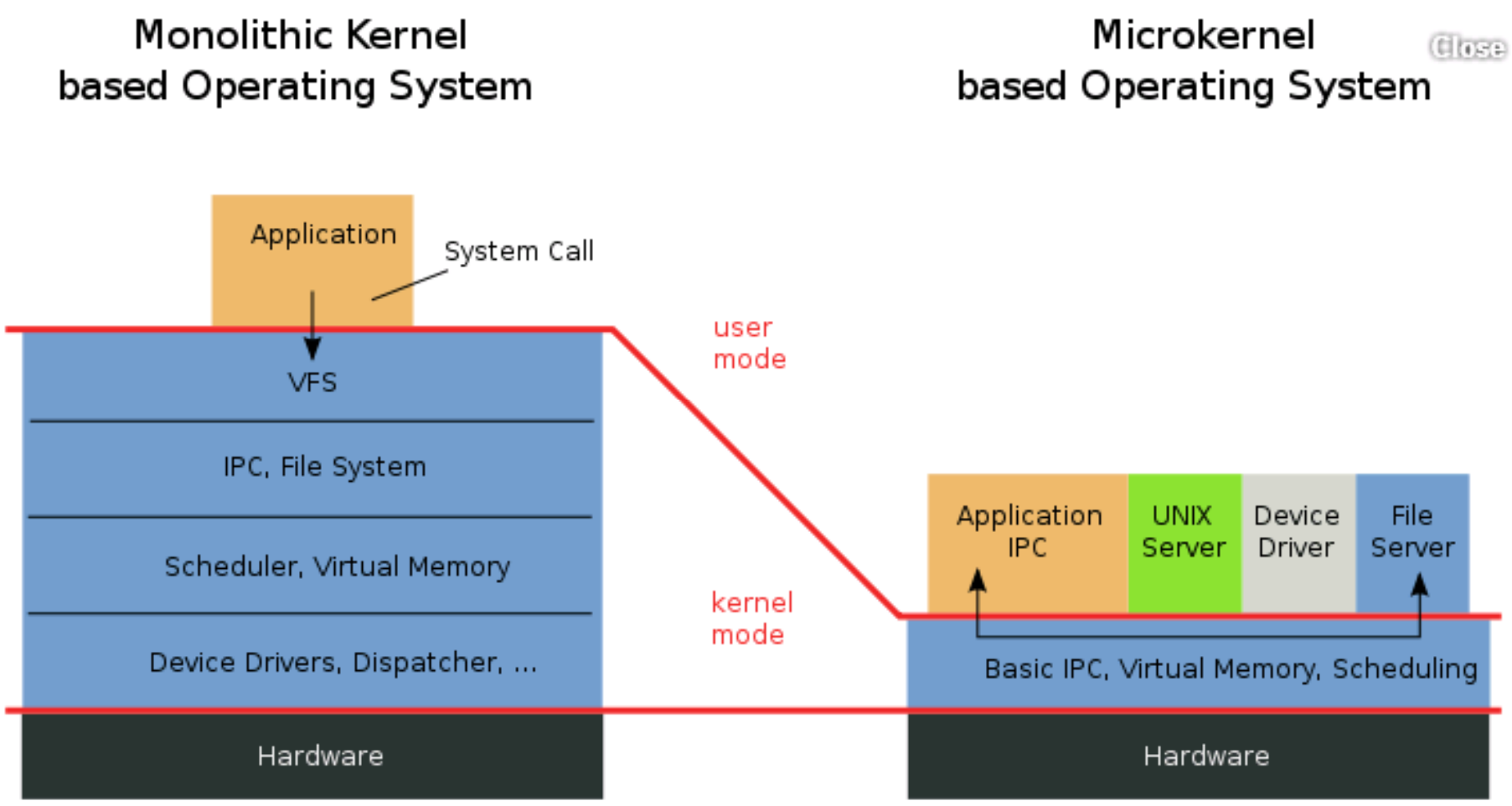
Type of shells



Graphical User Interface (GUI)
Windows/Mac OS



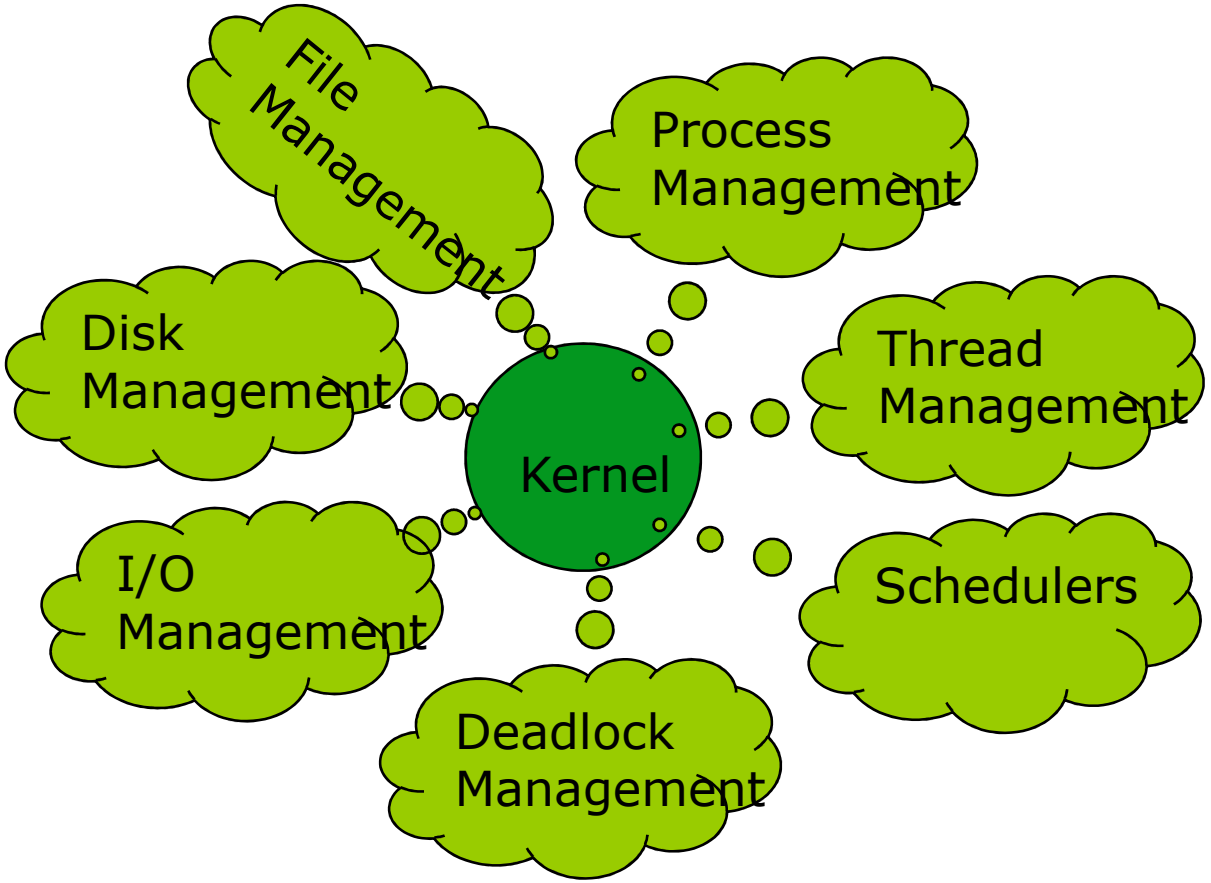
Command line interface (CLI)
Dos/Linux


Monolithic Kernel
based Operating System
Microkernel
based Operating System
Close
Application
System Call
VFS
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware
user
mode
kernel
mode
Application
IPC
UNIX
Server
Device
Driver
File
Server
Basic IPC, Virtual Memory, Scheduling
Hardware

Kernel Types

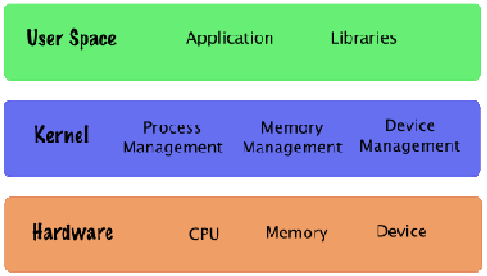
Monolithic Kernel	Micro Kernel
All the parts of a kernel Space - Scheduler, file system, memory, device drivers etc. are in one unit within the kernel.	Only the important parts are in kernel space - IPC, basic scheduler, basic memory management, basic I/O primitives etc. - Others (including device drivers) are in user space. Except clock driver.
Signals & sockets to ensure IPC	-Message Passing system to ensure IPC
Advantages - Faster processing	Advantages - Crash resistant - Smaller size - Portable - No need recompiling(for a new feature)
Disadvantages -Crash insecure -Porting Inflexibility -Add a new feature needs recompiling whole -Kernel size explosion	Disadvantages - slower processing due to additional message passing - 2%-4% slower
DOS, Unix, Linux	Symbian, Mac OS X, MINIX, Windows NT
Tanenbaum VS Torvalds Debate (Micro Kernel VS Monolithic Kernel) http://www.oreilly.com/openbook/opensources/book/appa.html, http://www.cs.vu.nl/~ast/reliable-os/	

OS Organization



System call and dual mode operation

Dual-mode operation allows OS to protect itself and other system components.

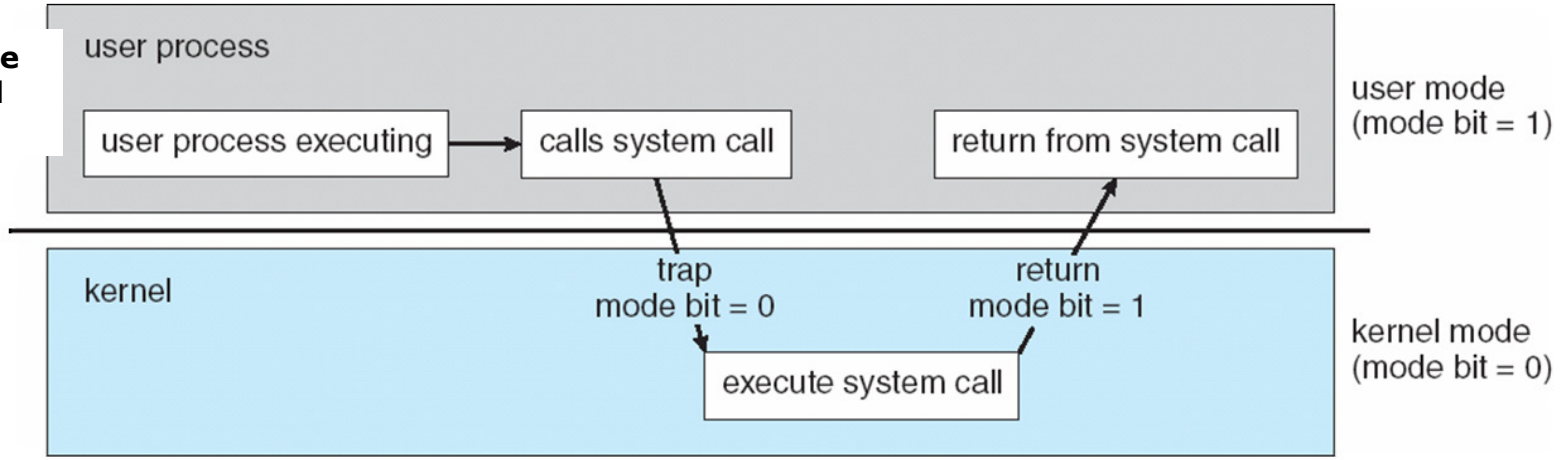


- User mode
- Kernel/Supervisor/Monitor/System Mode.

Mode bit @ Process Status Word (PSW)-Resister

- Provides ability to distinguish when system is running user code or kernel code.
- Some instructions designated as **privileged**, only executable in kernel mode. Example: load/save from protected memory, initiate I/O, timer management, and interrupt management etc.
- System call changes mode to kernel, return from call resets it to user.

System Boot time the H/W starts in kernel mode



Working with User and Kernel mode

scanf("%d",&a); // $t0 holds a value

PC

0	move $v0, 5
1	systemcall
2	store $v0, 0($t0)

1

Kernel I/O subsystem Service call handler (code)

2

mode bit is set to kernel mode(0).

3

4

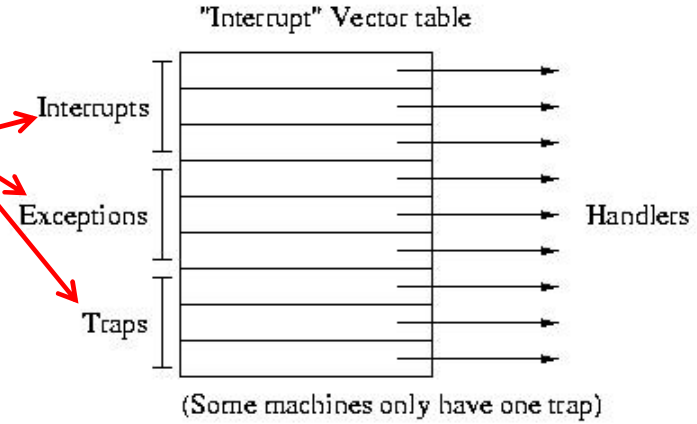
IVT- Interrupt Vector Table @ RAM-0000:0000H	
Service # ….	Code Address …..
005	0058
019	0090

	RAM
0150	systemcall (SWI)
0160	
0170	store $v0, 0($t0)
0058	Device driver (SWI)
0070	Allocates kernel buffer address To receive data
0074	GOTO RA=090
0078	GOTO RA=0160
0090	I/O command(control, test, read, write), Communication with H/W Controller
	GOTO RA=0078

Modified figure-13.13 @ galvin

Different Method of CPU Interruptions

An operating system is interrupt driven

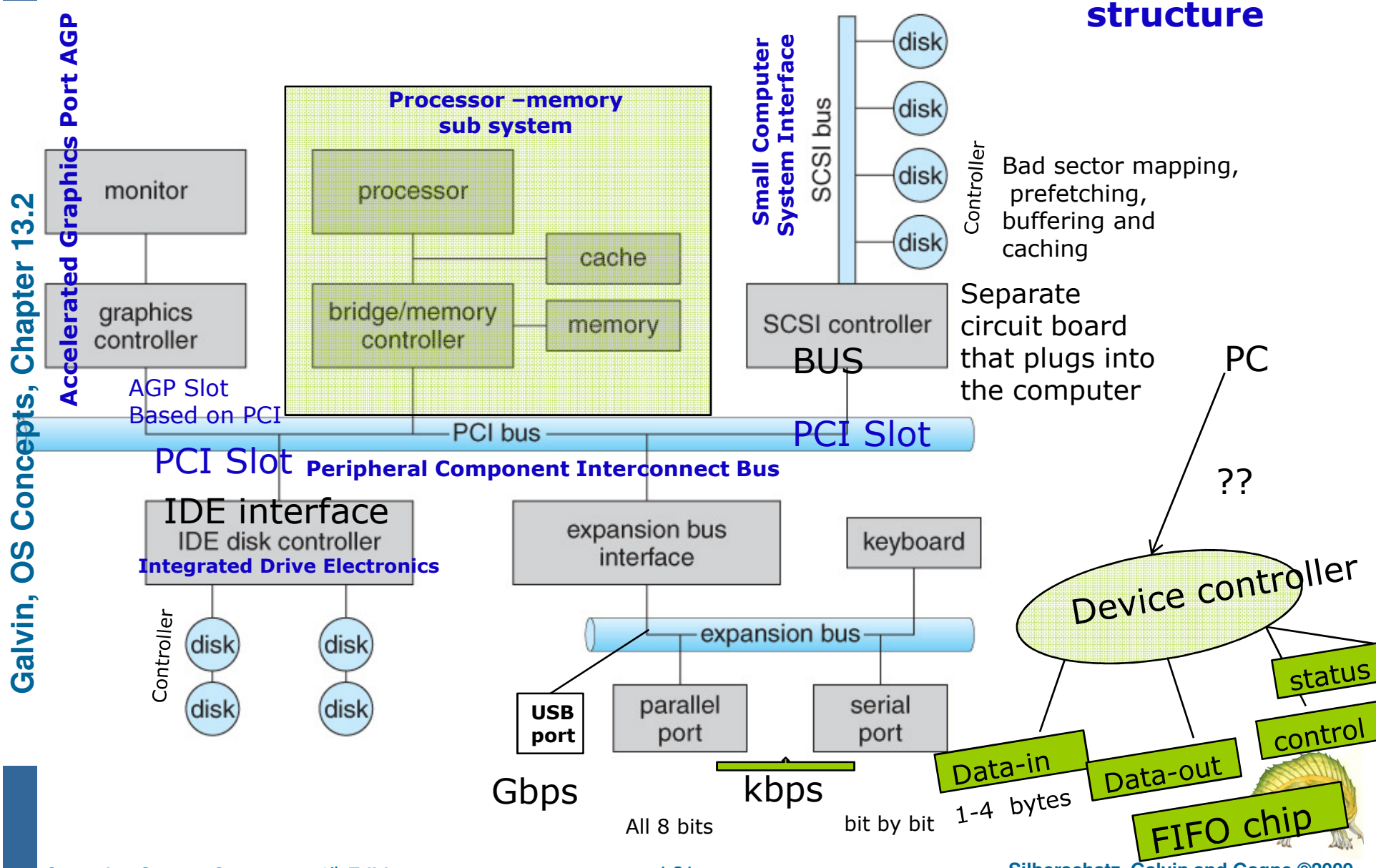


Interrupts (Hardware events)	• **external hardware event (external to the CPU).** • **Higher priority**
Exception (Software events)	• **automatically generated unexpected events occur in response to some exceptional condition.** • **illegal program actions that generate an interrupt.**
Traps (Software Interrupts)	• a programmer initiated and expected transfer of control to a special handler routine. • **user program can ask for an operating system service** • **unconditional –control always transfer to pre-defined procedure.**
System Call (Software Interrupts)	• simply sets up some registers with the needed system call/ OS service number, and execute the Trap/software interrupt.

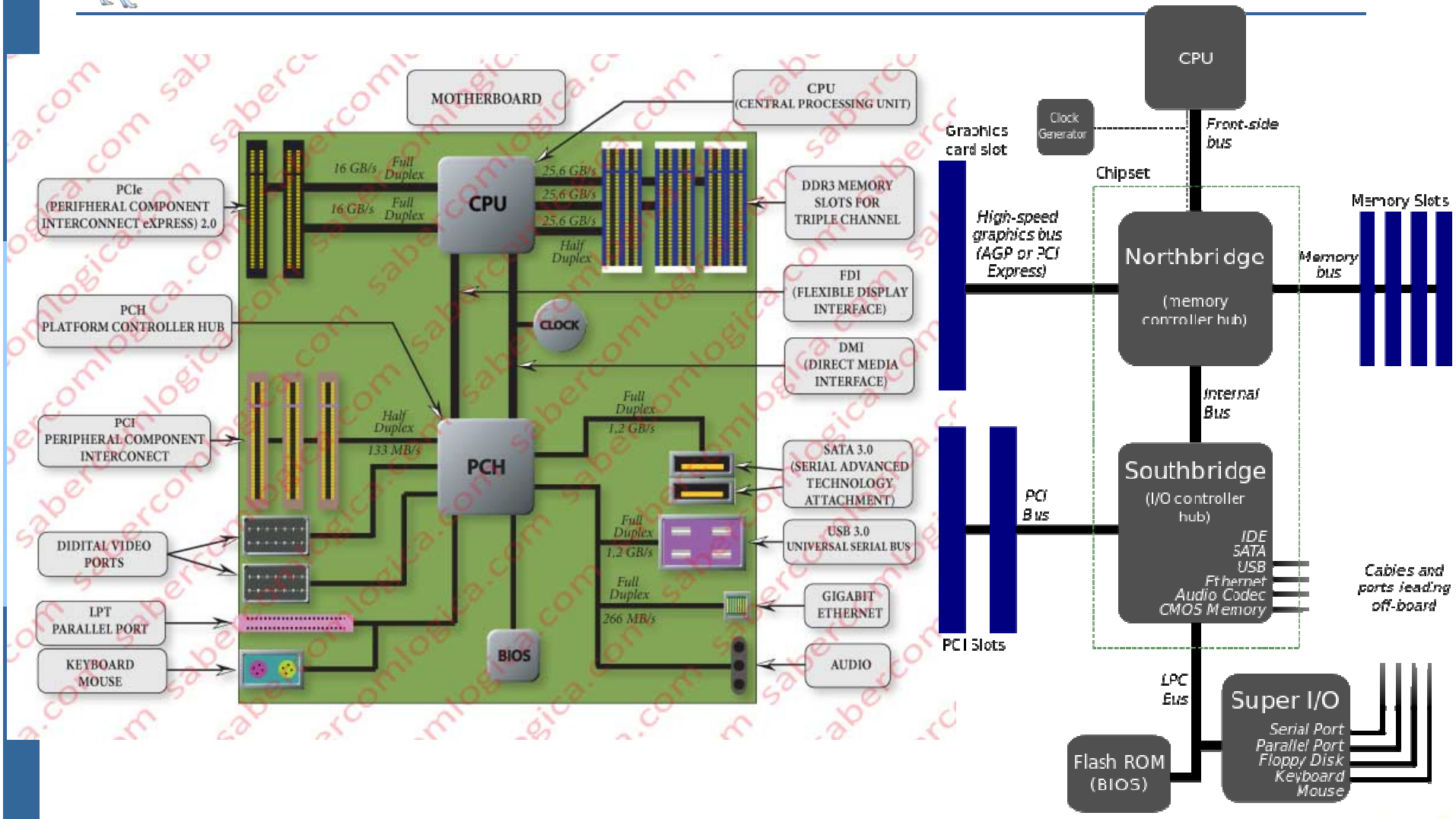
How does process pass data to controller?

I/O Hardware

Typical PC BUS structure



I/O Interface



Device I/O port location on PCs

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

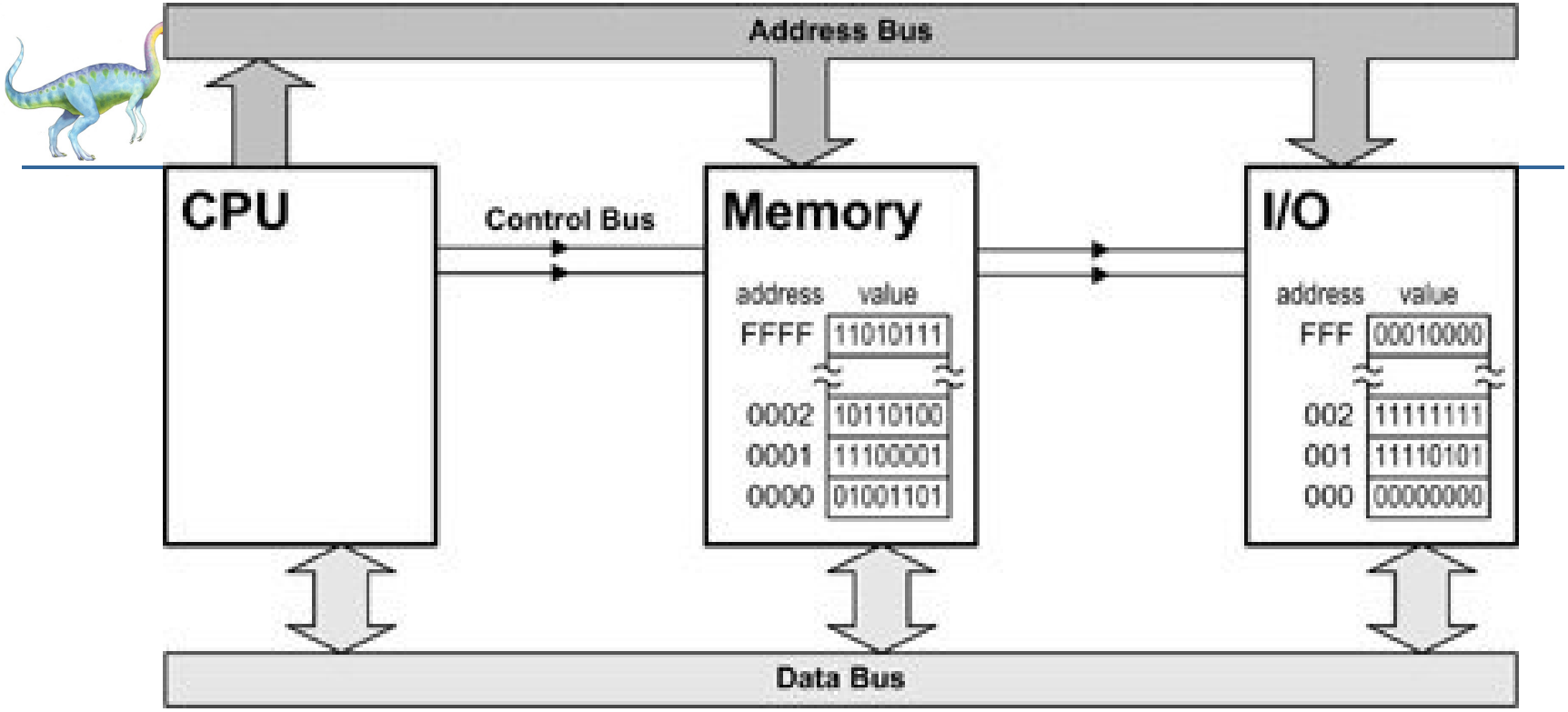
I/O Mapped: Special I/O instruction in/out instruction

Example from the Intel architecture: `out 0x21,AL`

```
000-00F DMA Controller
020-021 Interrupt Controller
040-043 Timer
3D0-3DF  Graphics Controller
```

Output to port

IN instruction reads from an I/O device, OUT writes.



- CPU communicates either memory or I/O device at a time, never both.
- CPU uses one of the control lines called "IO/MEM"
  - asserts (line=high) the CPU wants to work to memory
  - (line = low) for an I/O device.

- The IN instruction reads from an I/O device, OUT writes.
  - IO/MEM is not asserted (held low), so memory doesn't respond and the I/O chip does.

- For the memory-oriented instructions (Load/Store),
  - IO/MEM is asserted so CPU talks to the RAM, and IO devices stay out of the communication.

Memory mapped I/O:

1-Busy
0-not

32 bits	IE	B

32 bits Data

@ Physical memory

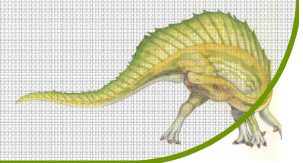
No Special Instruction load/store instructions

Controller registers maps to address space of the processor.
I/O accomplished with load and store instructions

Example: **load 0xFFFF0000, AL (Control)**
load 0xFFFF0004, BX (Data)

Address Decoding Circuitry

Hybrid : Graphics controller
Control operation via I/O port
Data-out memory mapped region



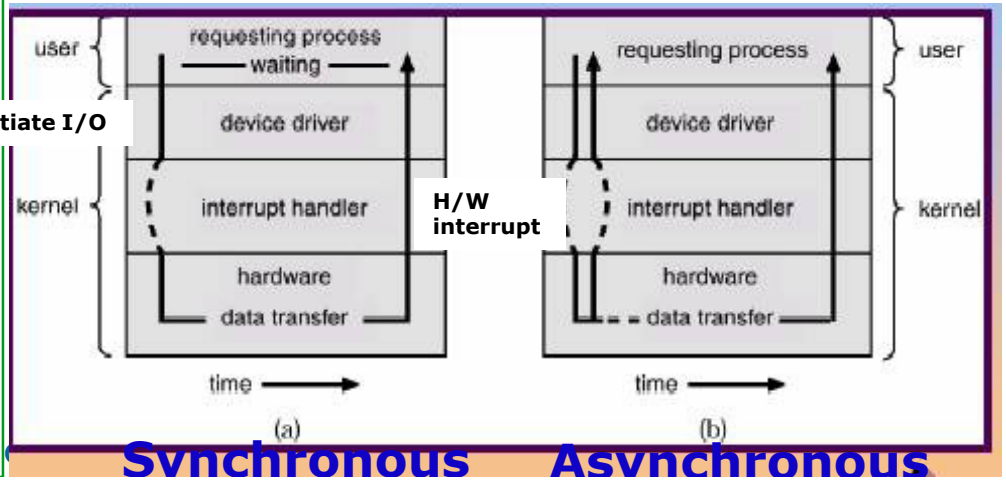
How does H/W communicate with Kernel/vice versa?

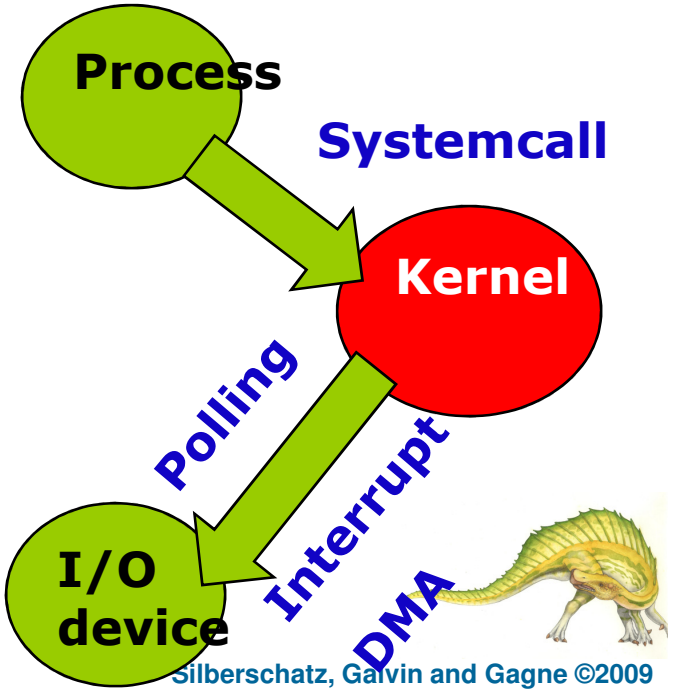
Process requests I/O

Synchronous

Asynchronous

H/W Interrupt After I/O completion





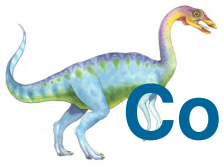
Communication Between CPU & H/W Controller

- Three (3) techniques

  Polling, Interrupt and Direct Memory Access

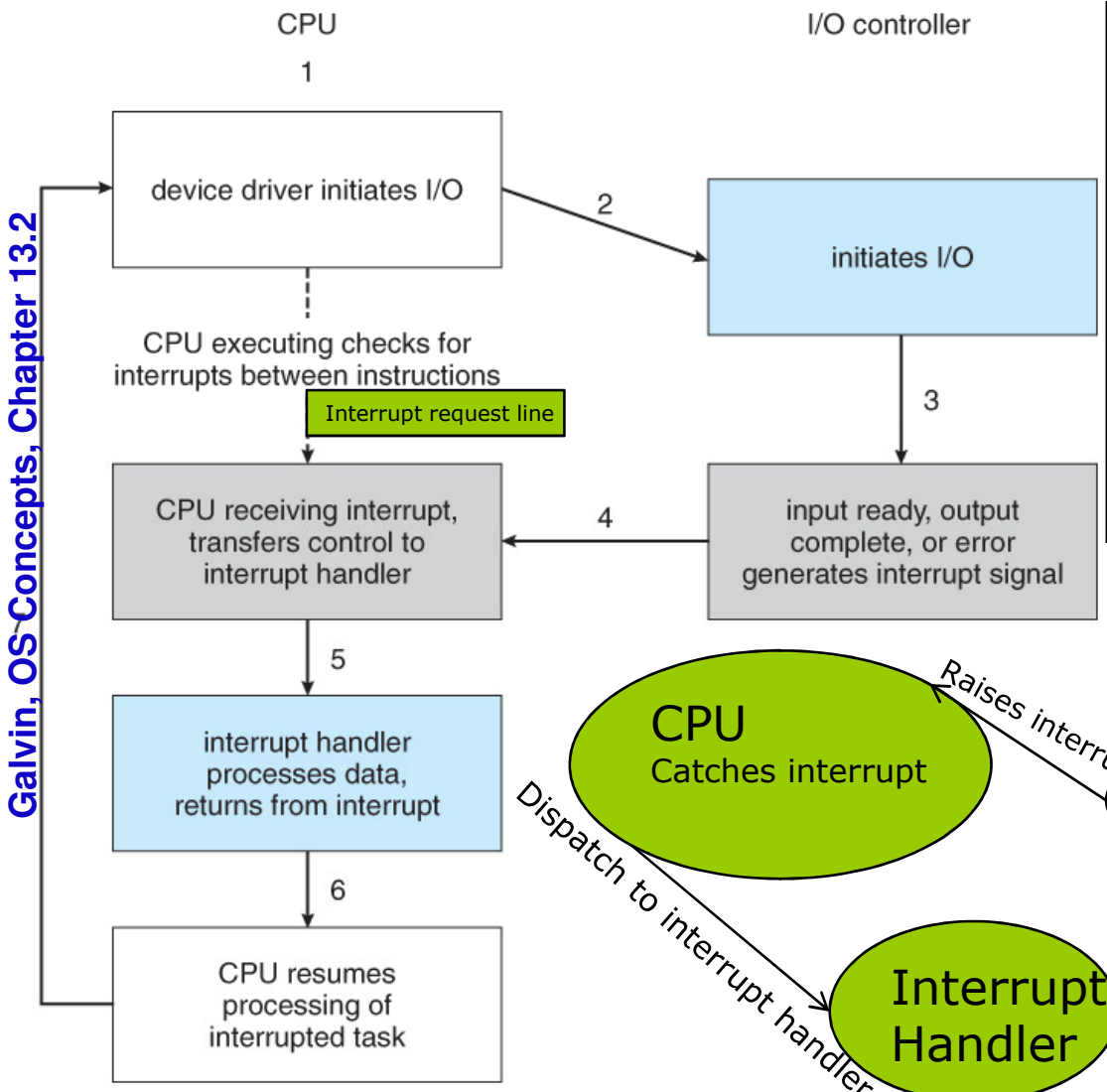
Polling (example : Mobile Phone)

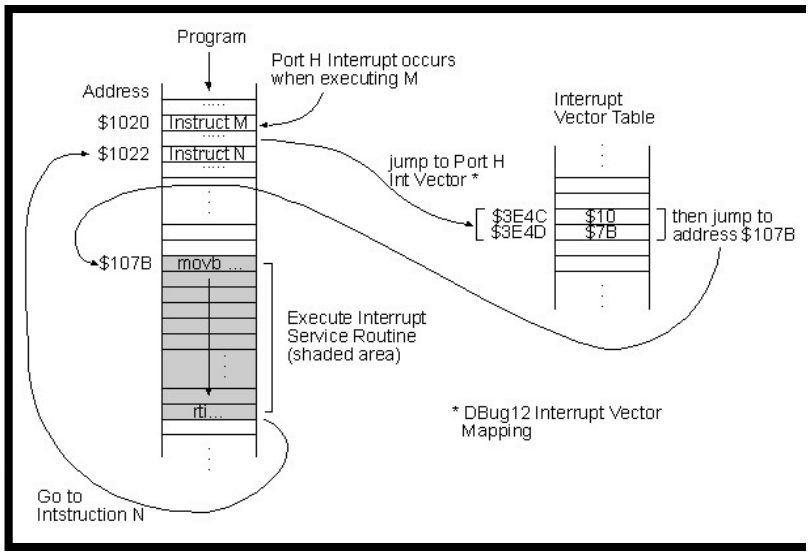
Processor	Controller
Sets the **write bit** in the command register and writes a **byte** into the **data-out** register	
Sets **command-ready bit** in the command register	Controller notices the **command-ready bit** is set
BUSY-WAITING	Sets busy bit
	Read command register and see write bit set/write command given
	Gets data from data-out register and does I/O to the device
Poll a device takes three (3) CPU cycles -read device register -do logical-and to extract a busy bit @ status register -bit =0 then submit job to device	Clears the command-ready bit, clears the error bit in the status register to indicate I/O succeed, and clears the busy bit to indicate that it is finished.



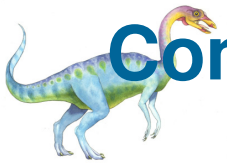
Communication with CPU & H/W Controller

Interrupt (example : Mobile Phone)



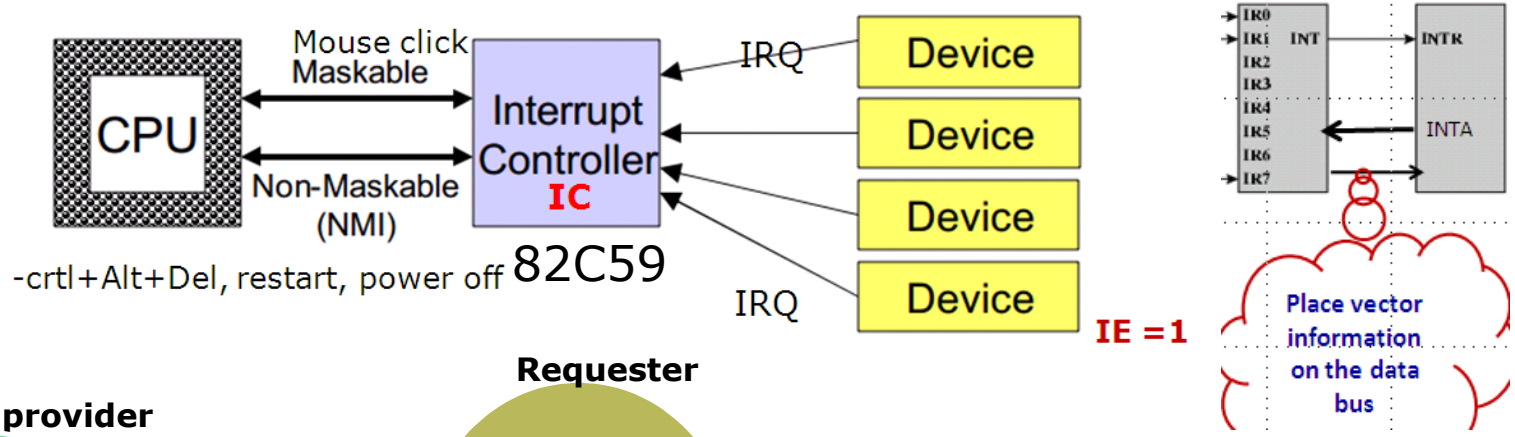


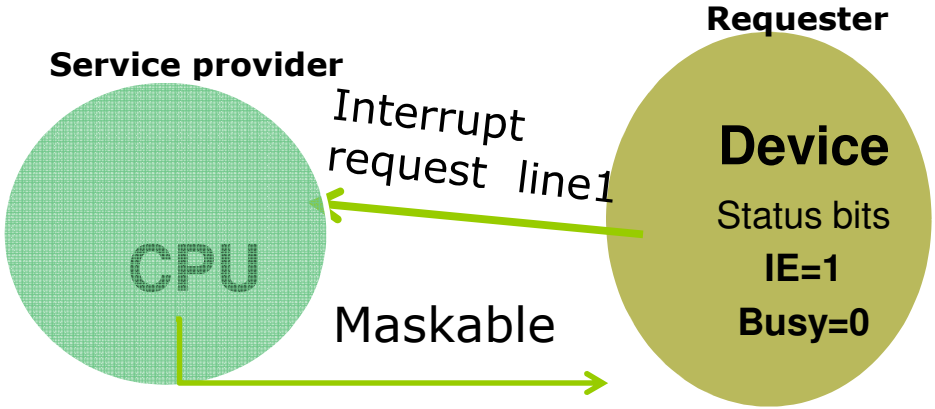
Program Execution with simple Interrupt



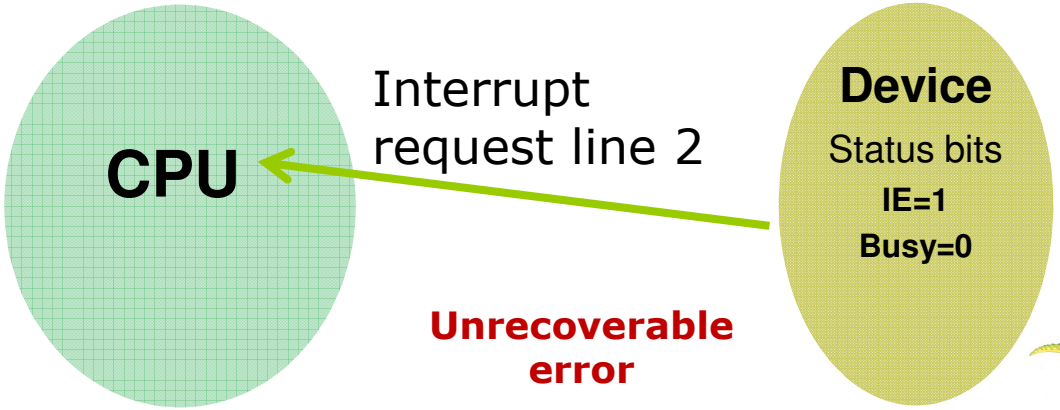
Communication with CPU & H/W Controller

Interrupts Work





Able to Turned off
Masked Interrupts
Short periods of time



Interrupt Vector Table

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Non-maskable

DMA – Direct Memory Access

- Device does large transfer – disk drive
  - CPU takes long time (checking status bit, feed data register one byte)

Programmed I/O-PIO

- Without burdening the CPU (only beginning and end)
  - some of this task transfers to special purpose processor Direct Memory Access (DMA) controller

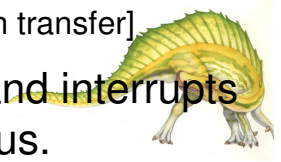
DMA Operation

- CPU wishes to read or write a block of data @memory
  - CPU provides command to DMA controller
  - Read/Write (IOR, IOW, MEMR, MEMW)
  - Device address
  - Starting address of memory block of data
  - Amount of data to be transferred

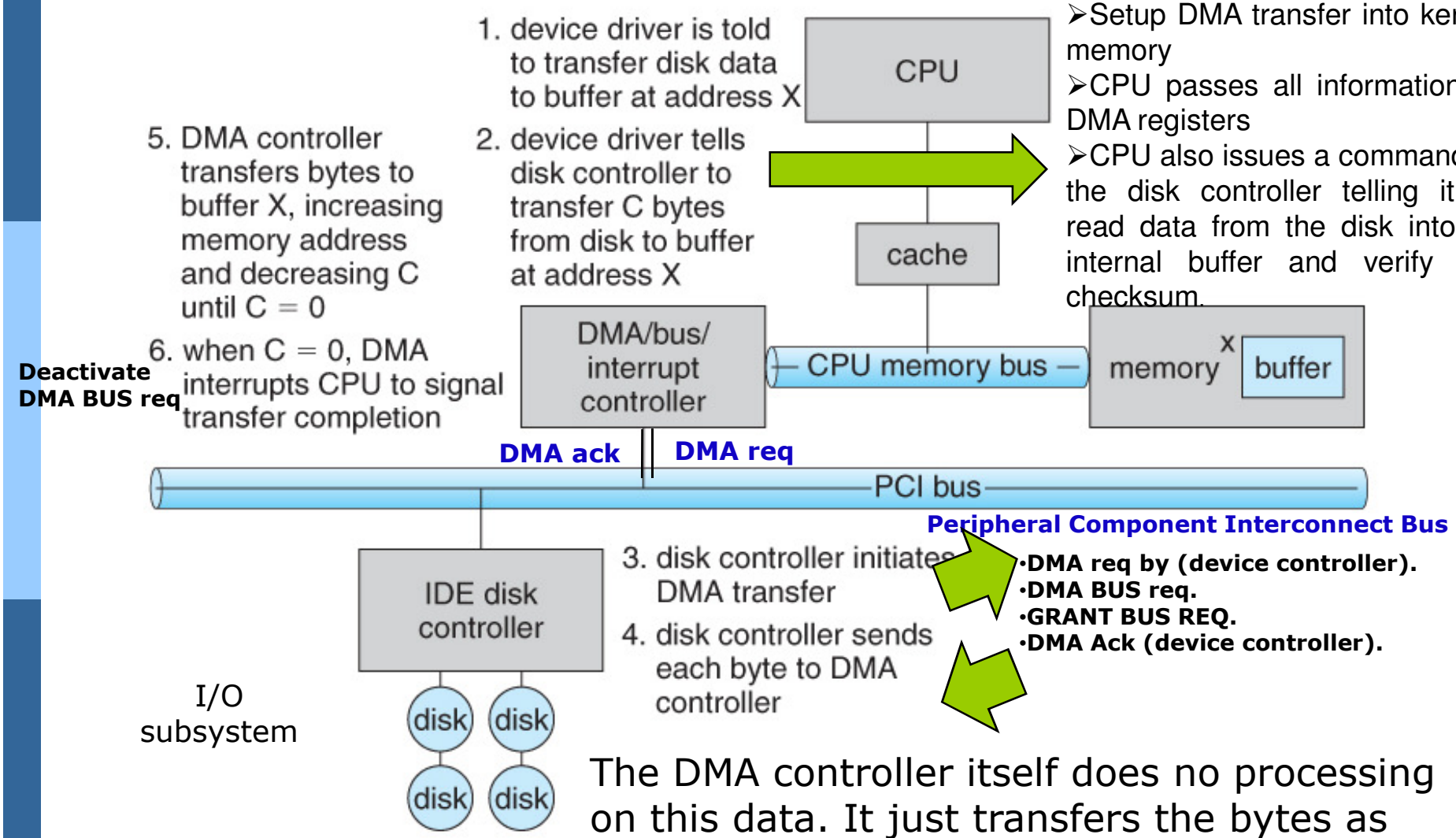
DMA Controller

-Status
-Source
-Destination
-Length

- DMA Takes control over bus
  - DMA places **bus request** to CPU
  - CPU accepts request by asserting the **bus grant signal** [Both asserts DMA can transfer]
  - When data transfer (words) completes, DMA **deasserts request signal** and interrupts to CPU,CPU then deasserts the bus grant signal and takes control over bus.



Handshaking: DMA and Disk Controller



- Setup DMA transfer into kernel memory
- CPU passes all information to DMA registers
- CPU also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum.

The DMA controller itself does no processing on this data. It just transfers the bytes as instructed in its configuration registers.

Cyclic Stealing :

- DMA forces CPU to stop its activities and handover the memory bus to DMA. Thus DMA steals a bus cycle from CPU.
- DMA controller transfers one word at a time after which it must return the control of the buses to the CPU.
- The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to "steal" one memory cycle.

DMA is capable to transfer data (Using 4 channels)between:

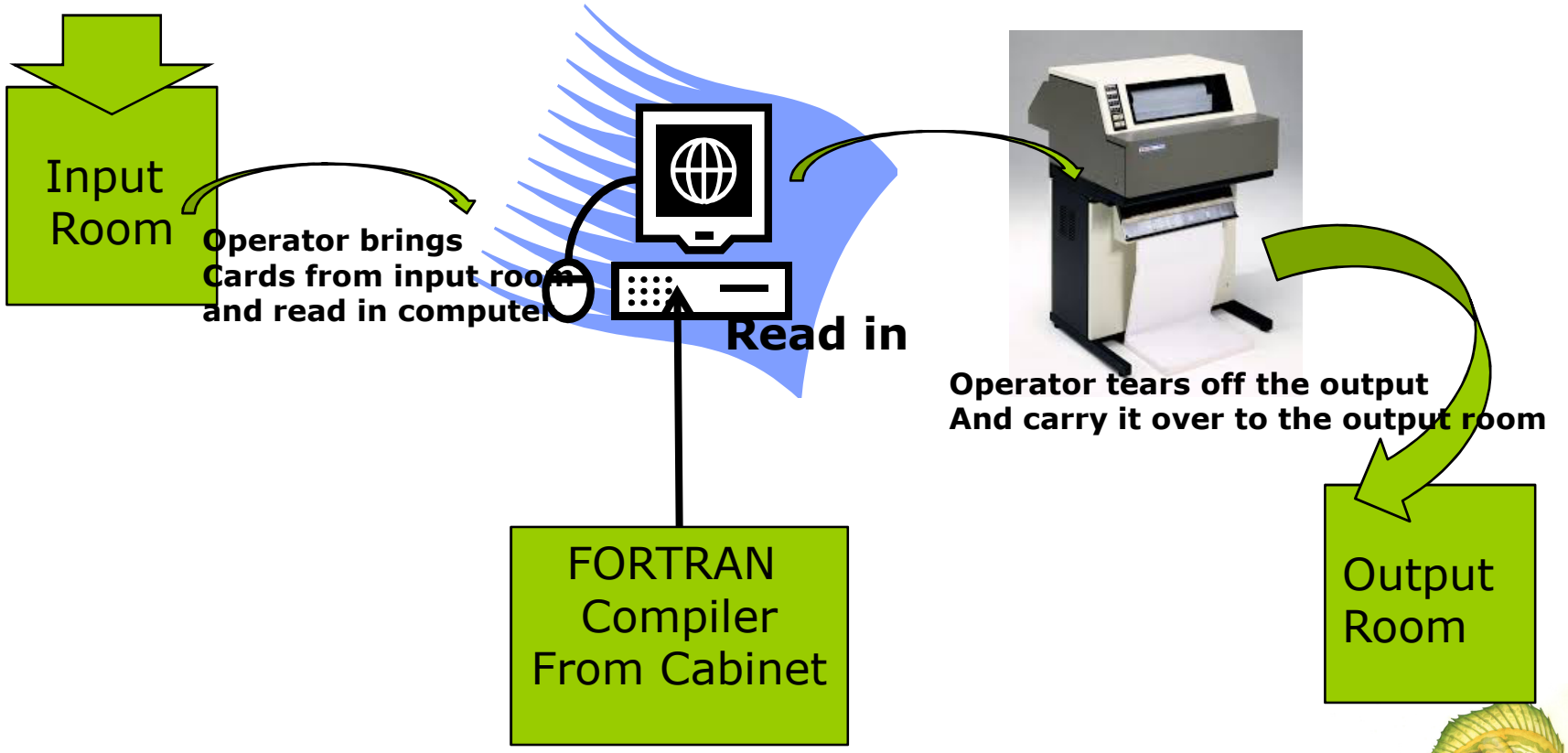
- Memory and I/O device or vice versa
- Memory to memory using (DMA control/command register loaded by CPU) Channel 0 is from memory to register and channel 1 is from register to memory.

OS Evaluation

Computers were in Universities / Large companies

Programmers wrote programs (FORTRAN/ Assembly)on paper.

Punched the program into CARDS

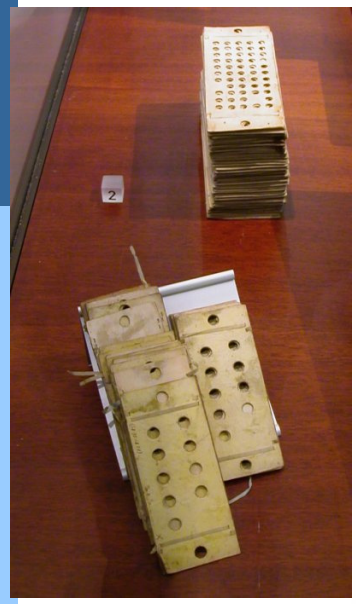


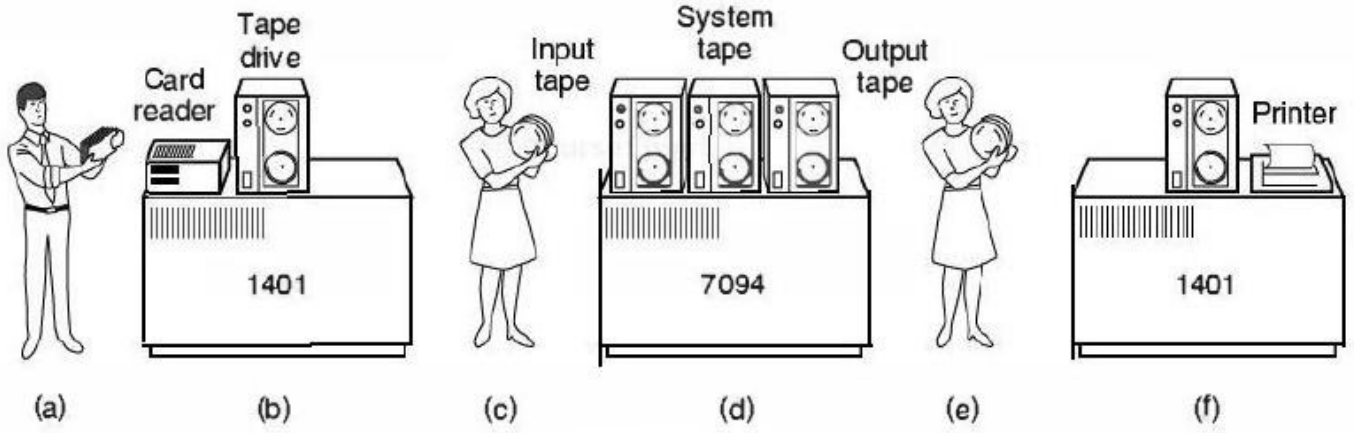
Wasting time while operators were walking around the machine room

Supervisor/Operator Control

CR ----> MT MT ----> CPU CPU ----> LP

IBM 7094





An early batch system. (a) Programmers bring cards to 1401. (b) 1401 reads batch of jobs onto tape. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

- Problems
  - Long turnaround time - up to 2 DAYS!!!
    - Debugging…
  - Low CPU utilization
    - I/O and CPU could not overlap;
    - slow mechanical devices.

From John Ousterhout slides

Offline Processing

BATCH Processing (One by one) Similar jobs

Process 1			Process 2			Process 3			
I1			I2			I3			
	C1			C2			C3		
		O1			O2			O3	
2	4	2	2	3	2	2	7	2	26

OS
User Program

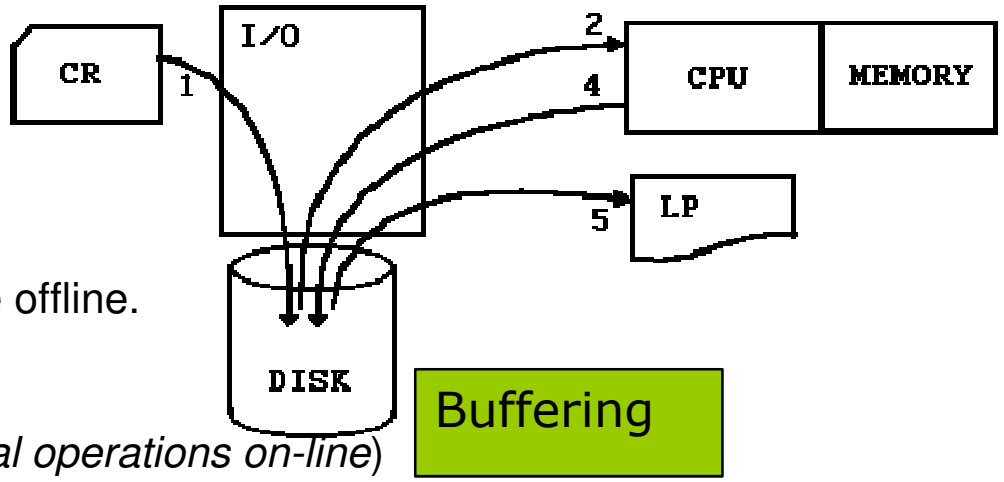
I= Input
C=Computation
O=Output

OS resides in memory and the task was simple
- transfer control from one job to another

Problem: Low CPU utilization;
- During I/O CPU is idle. WHY?
- @Class notes

Batch Systems – Upgrading Issues

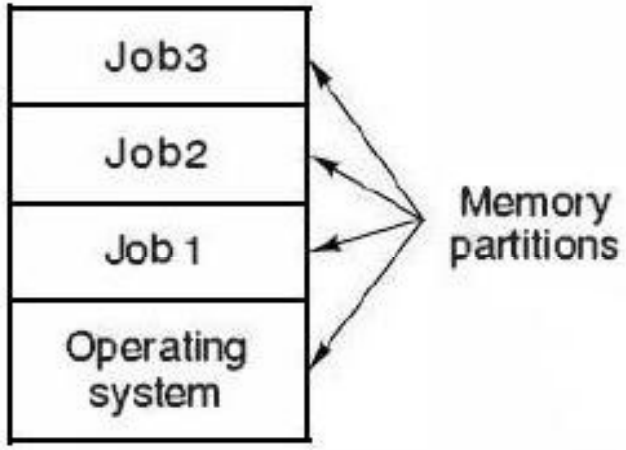
- **Solutions to speed up I/O:**



- Offline Processing (Previous)
  - load jobs into memory from tapes,
  - card reading and line printing are done offline.
- Online Spooling (*simultaneous peripheral operations on-line*)
  - A type of buffering, to disk (present)
    - Random access device, unlike tape
  - Use disk as large storage for reading as many input files as possible and storing output files until output devices are ready to accept them.
  - Allows overlap - I/O of one job with computation of another.
  - Introduces : job pool (Queues-job queue, ready queue)
    - allows OS to choose next job to run so as to increase CPU utilization

Multiprogramming

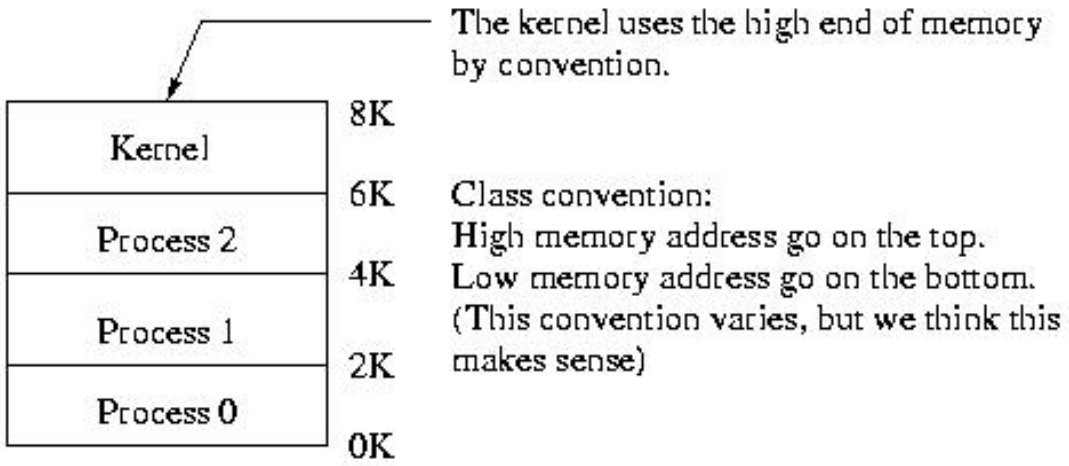
- Use interrupts to run multiple programs simultaneously
  - When a program performs I/O,
    - instead of polling, execute another program till interrupt is received.
  - Efficiency



A multiprogramming system with three jobs in memory.

- Requires:
  - **secure memory and** I/O for each program.
  - **intervention, if program loops indefinitely**.
  - CPU scheduling to choose the next job to run.

Simple Model for Memory Protection



To obtain true protection, need more hardware support.

- The goal is to make sure that, for example, process 2 can't go outside of the 4K-6K region.
  - This requires hardware support -- we can't do it in software alone.
  - The simplest model is to add two registers: BASE & LIMIT (assigned by OS)
  - This is the simplest model, most computers are more complex.

In order to keep process 2 within it's 2K box, we can do one of two things:

Option #1:
Set BASE = 4K
Set LIMIT=2K
A memory access is legal,
iff BASE <= X < BASE + LIMIT,
where X is the actual memory address

Option #2:
Set BASE = 4K
LIMIT = 2K
A memory access is legal,
iff 0 <= X < LIMIT,
where X is the actual memory address = BASE + X

more frequent scheme

I/O Protection

-Illegal I/O commands

- I/O instructions are privileged

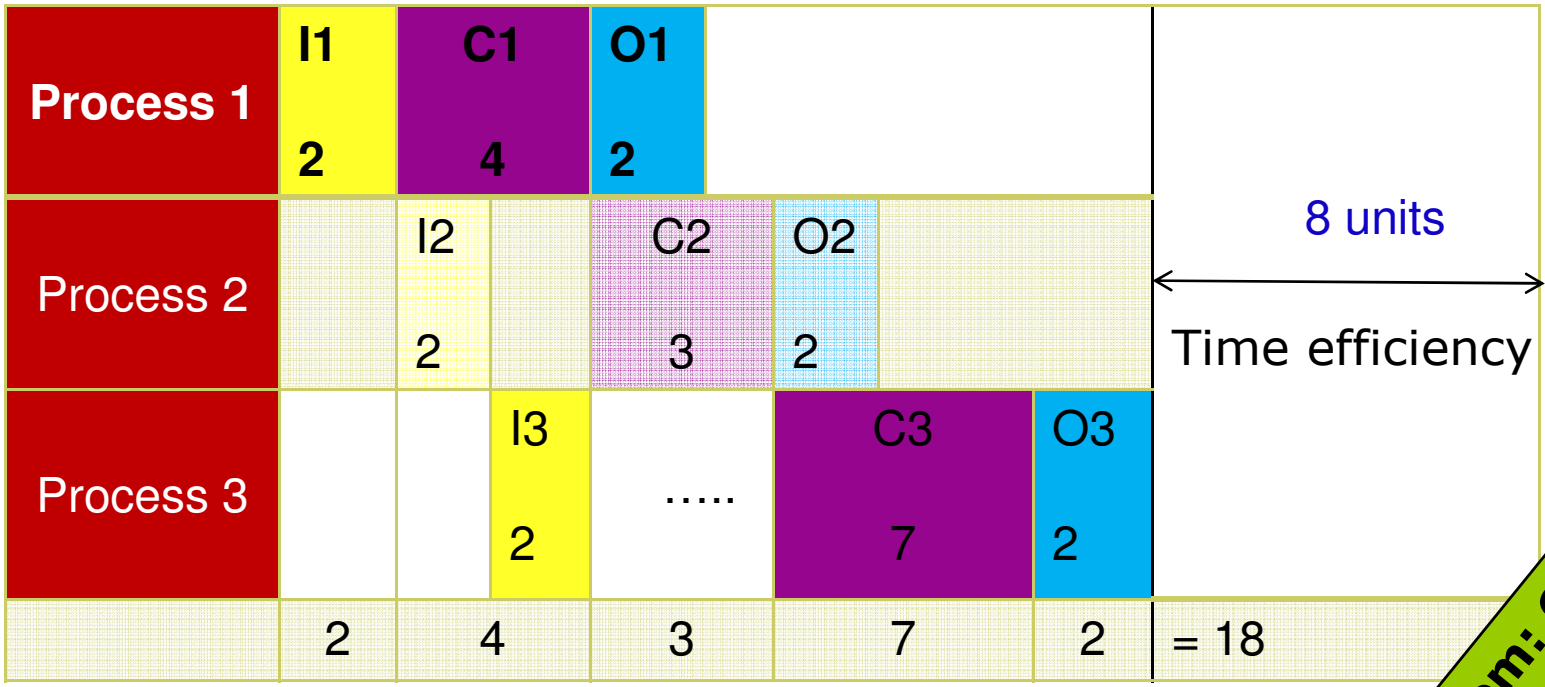
CPU Protection

-Timer device, can be set to interrupt the CPU after a specific periods

- Fixed and variable (counter decrement)

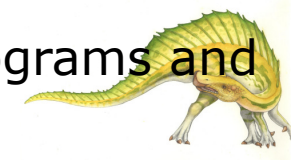
Multiprogrammed Processing

I= Input
C=Computation
O=Output



Problem: Convoy effect!

OS supports
-SPOOL
-Overlapping I/O and CPU [I/O of one job CPU of another job]
- Interleaving – mixed of CPU and I/O operations of several programs and alternating between them



Timesharing

- A variant of multiprogramming.
- Supports multiple users at the same time
- Programs queued for execution in FIFO order.
- A technique to protect CPU.

- **Like multiprogramming, but timer device interrupts** after a quantum (timeslice).
  - Interrupted program is returned to end of FIFO
  - Next program is taken from head of FIFO

Time Shared Processing

Process 1	I1 2	C1 2	…	C1 2	O1 2					
Process 2		I2 2	C2 2	…	…	C2 1	O2 2			
Process 3			I3 2	…	C3 2	…	C3 2	C3 2	C3 1	O2 2
	2	2	2	2	2	3		2	1	2 =18

Concept of Virtual Processors
Time Slice =2

Virtualization

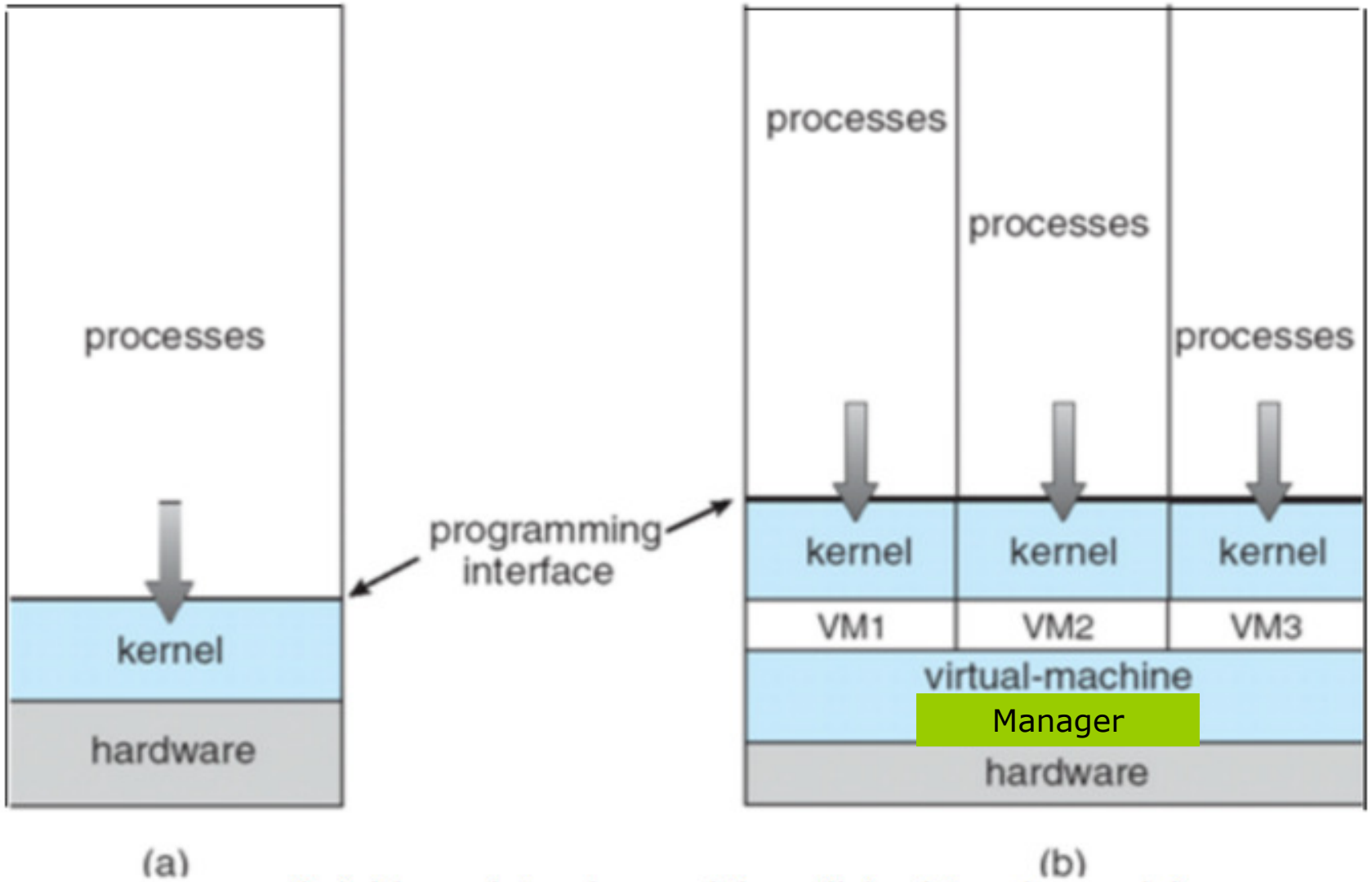
- Time sharing system provides
  - Multiprogramming
  - **Extend machine** with more convenient interface
- Virtualization
  - Not real but virtual existence
  - Two type virtualizations- H/W level abstraction & S/W level abstraction
  -

-Virtual H/W- processor, memory, chipset

-Interface
- Virtual environment
-Service virtualization – cloud
-Application- virtualization

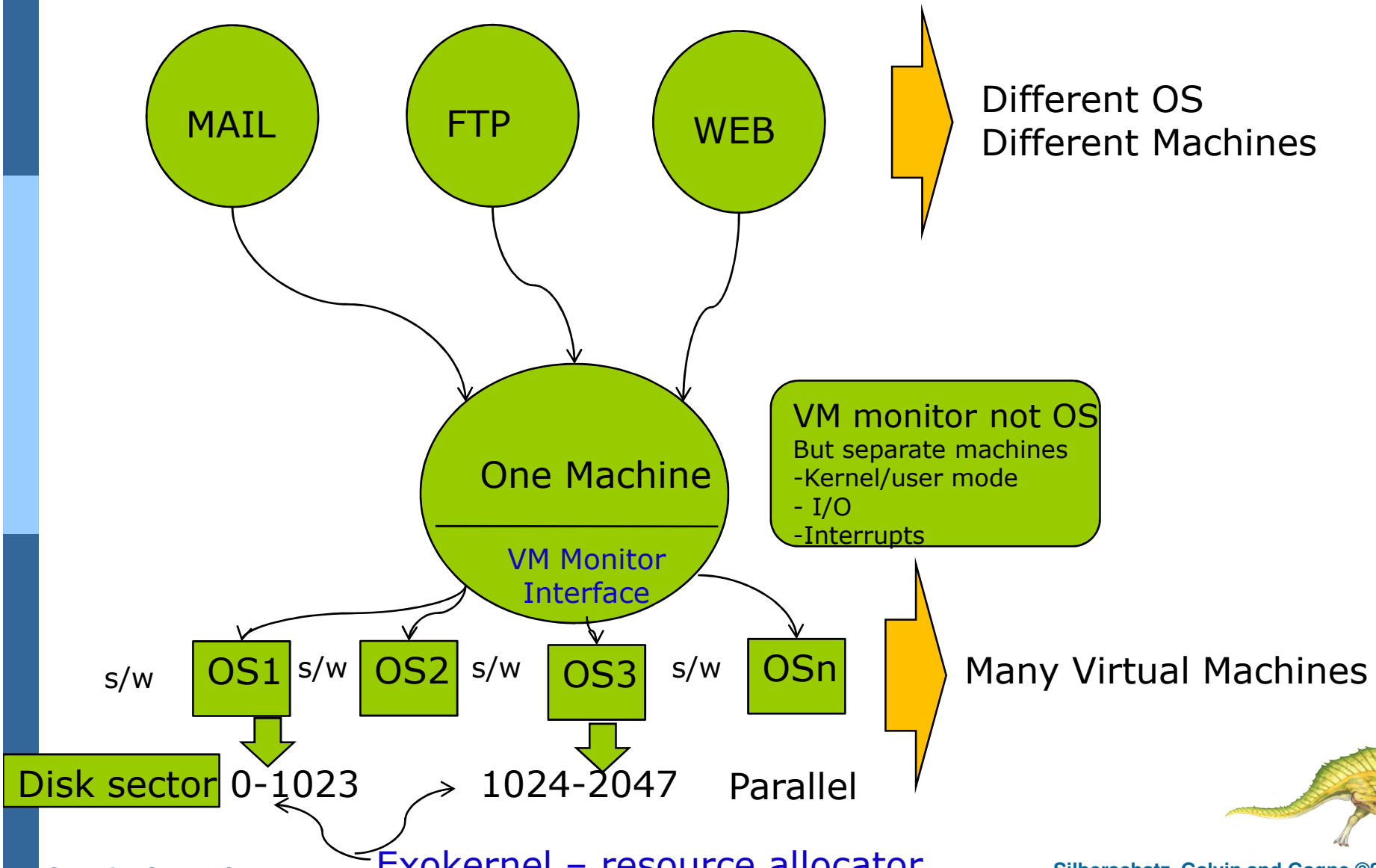
-OS itself running as application to another OS
-Emulation: Running an **operating system** on a hardware platform for which it was not originally engineered.

Virtual Machines



(a) Nonvirtual machine (b) virtual machine

Virtual Machines



Reading Materials

- Galvin : (Introduction) 1.1, 1.2, 1.4, 1.5, 1.11.6
- Galvin: (Operating System Structures) 2.1, 2.3, 2.4.1`, 2.5, 2.6, 2.7.1-2.7.3
- Galvin: (I/O Systems) 13.1-13.2,13.3.4
- Galvin: (Influential OS) 20.2

End of Lecture 1-4