CSE 325-Operating System

Lecture: 1-3

Computer Science and Engineering

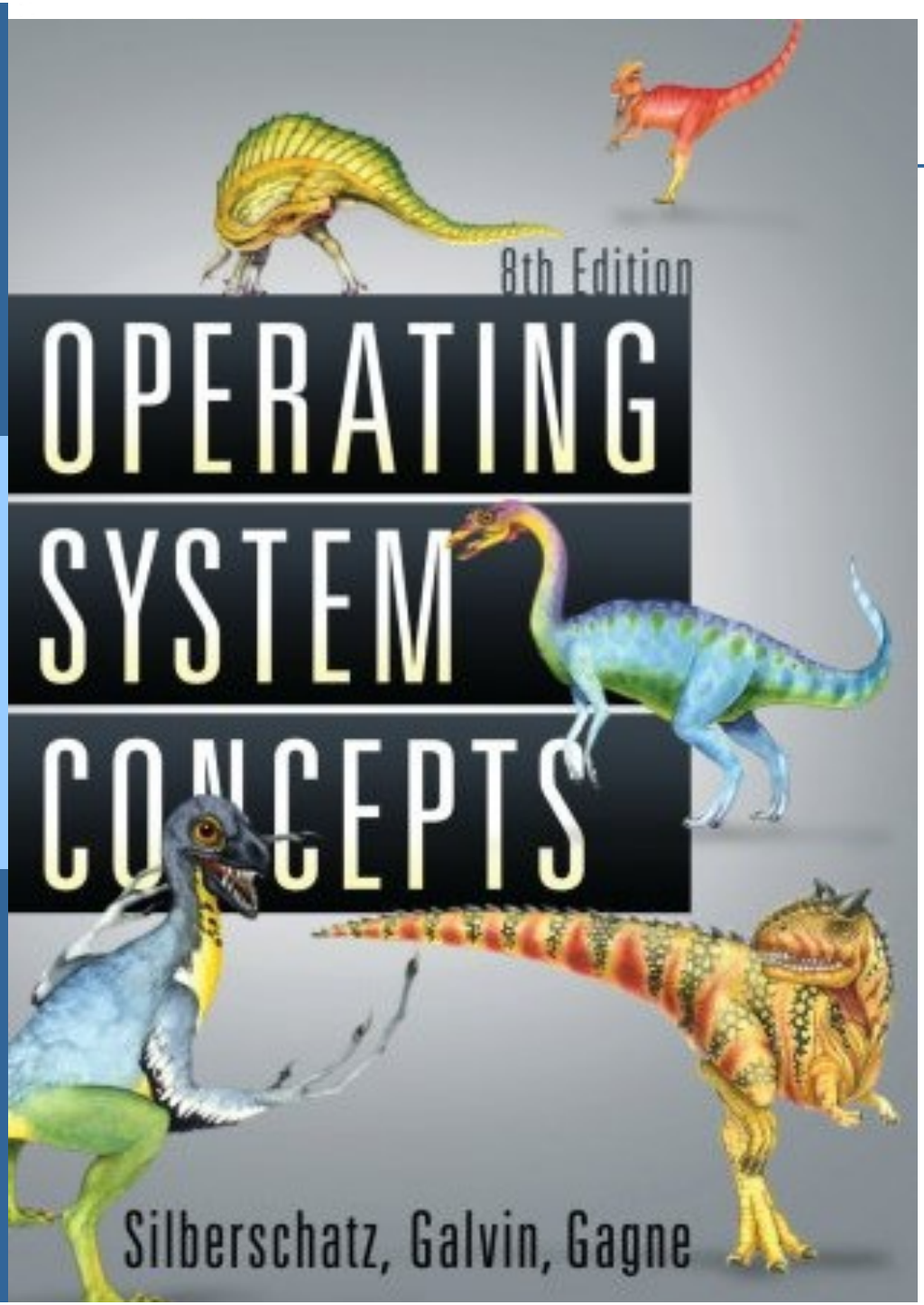
Course Description

- This course introduces the principles and techniques for the **design and implementation of operating systems**:
  - **interrupts,**
  - **computer resource management**
    - memory management,
    - processor management,
    - I/O management,
    - file management,
    - process management and
    - security management,
  - **inter process communication.**
- Additionally topics are: Multithreaded OS, and Concurrent computations.
- This course includes a project implementation.

Course Outcomes

Successful completion of this course, you will be able to:

- Identify different components of Operating System.
- Describe different methods of resource management.
- Apply resource management techniques for resource constrained problems.
- Expose memory organization and I/O management techniques.



Text book

Lecture 1: Introduction

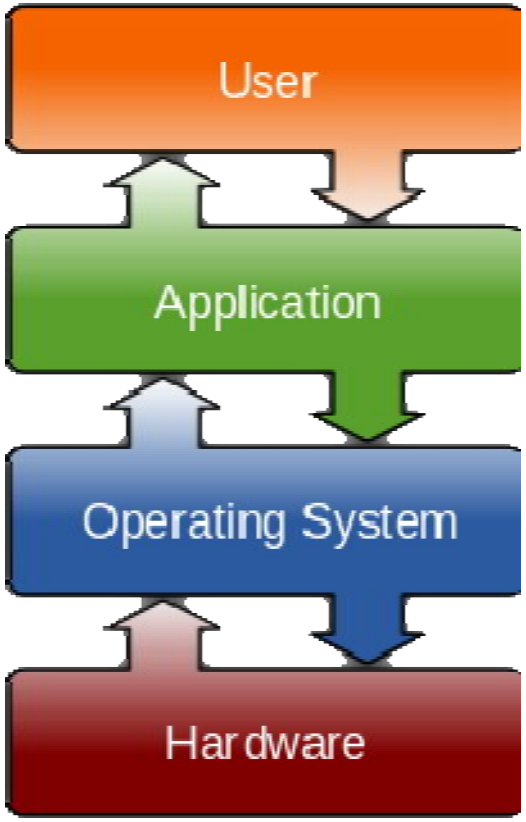
Objective

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



Tanenbaum@Modern OS

Microarchitecture- Computer Organization

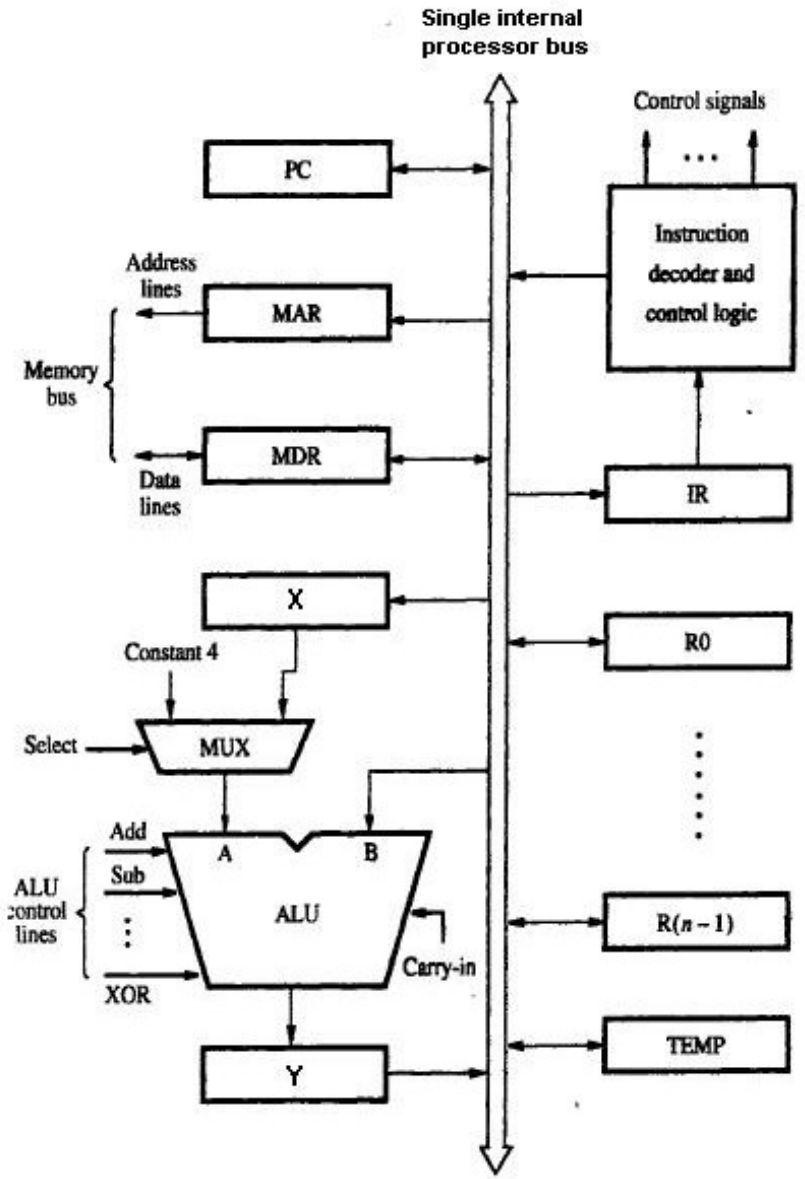
- The way a given instruction set architecture (ISA) is implemented on a processor

The ISA includes :

- the execution model,
- processor registers,
- address and data formats.

The microarchitecture includes:

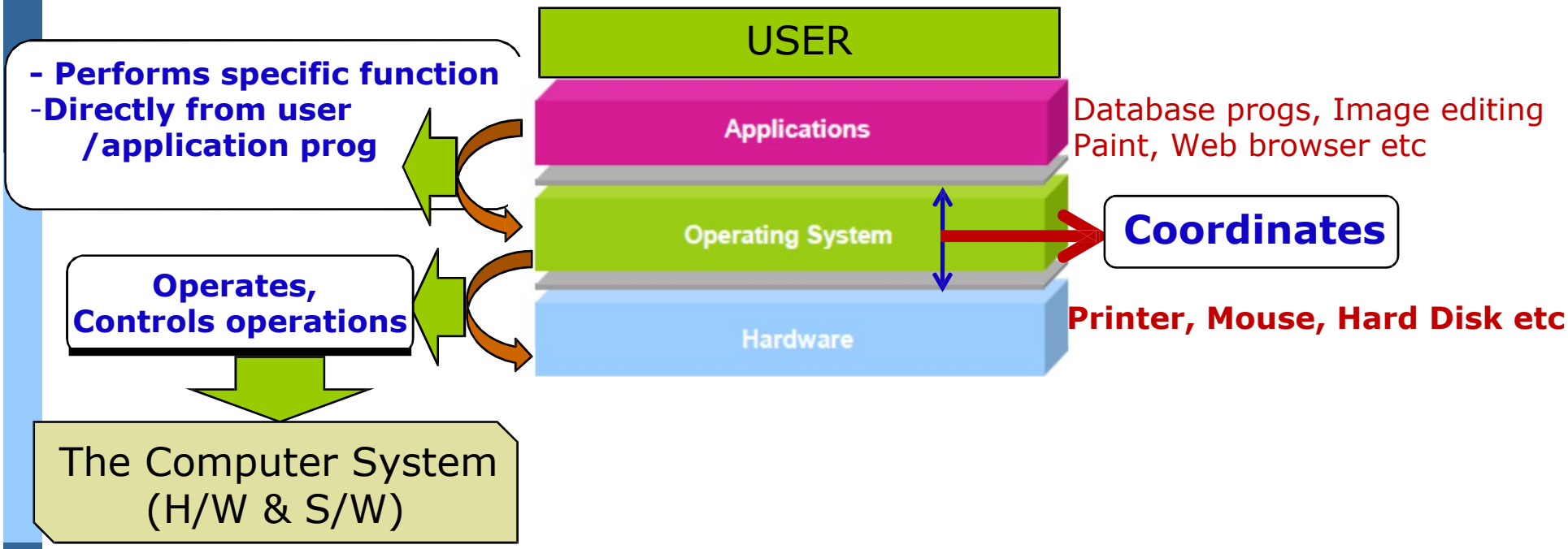
- the parts of the processor and
- how these interconnect and
- interoperate to implement ISA.



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

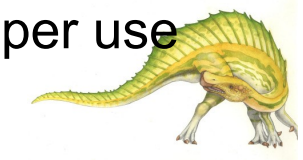
- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use.

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.



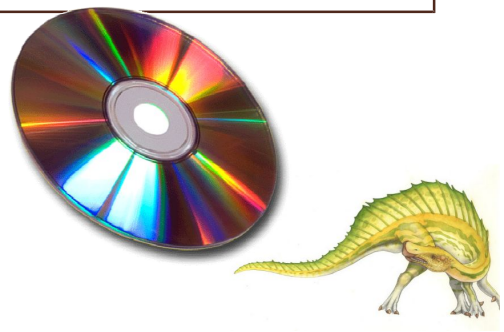
An example comparing life with/without OS

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 —Assemble→ First.o

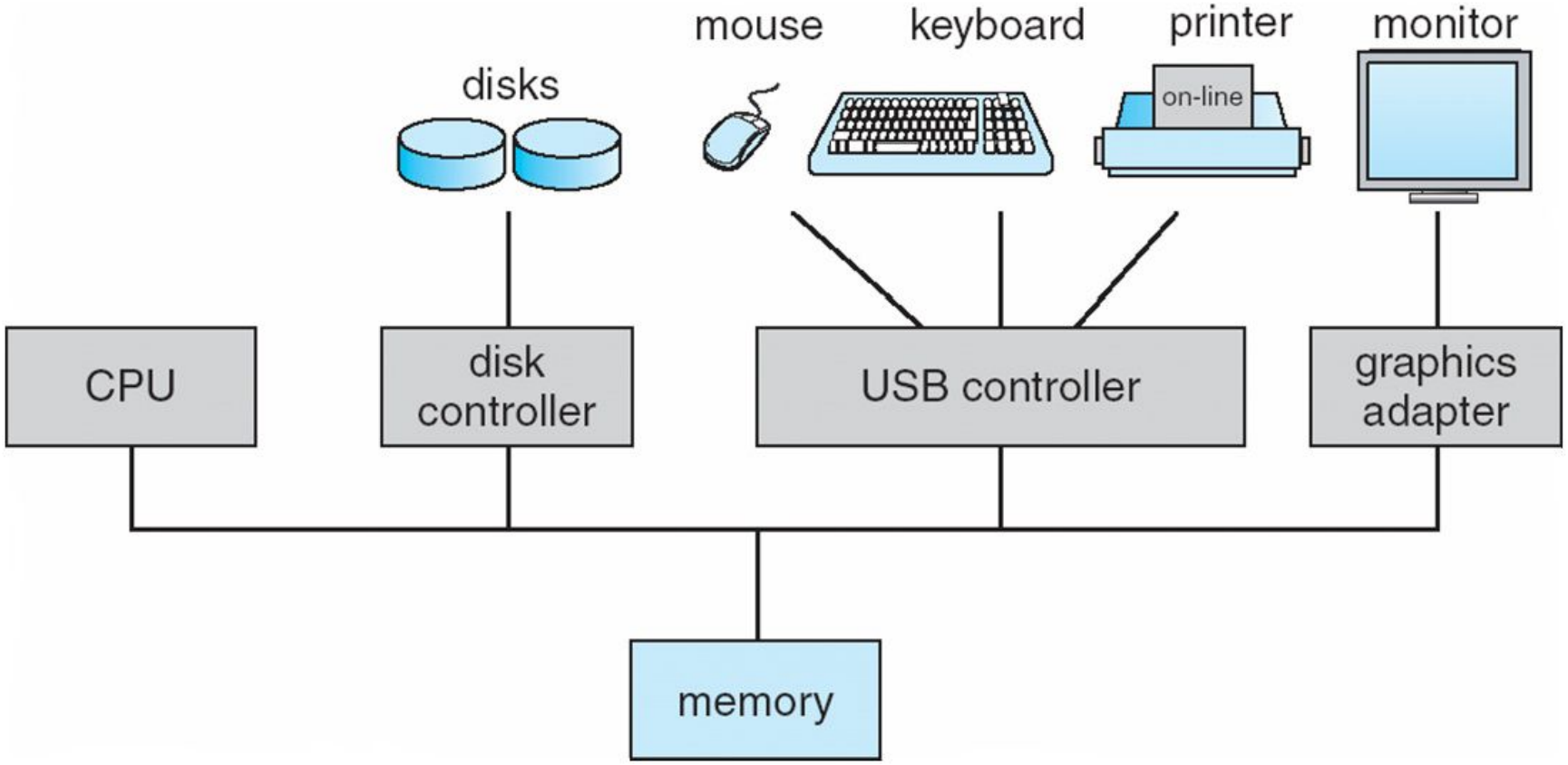
- Memory

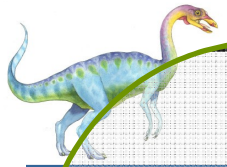
PC=0

IR= 0001……..00

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

Computer System





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
```

Memory mapped I/O: No Special Instruction load/store instructions

Controller registers/memory maps in physical address space. I/O accomplished with load and store instructions

Example: **load 0xFFFF0000, AL**

(Control) load 0xFFFF0004, BX

1-Ready
0-not

32 bits	IE	R

32 bits Data

Polling VS Interrupt

Both are methods to notify processor that I/O device needs attention

Polling

simple, but slow

processor check status of I/O device regularly to see if it needs attention

similar to checking a telephone without bells!

Interrupt

fast, but more complicated

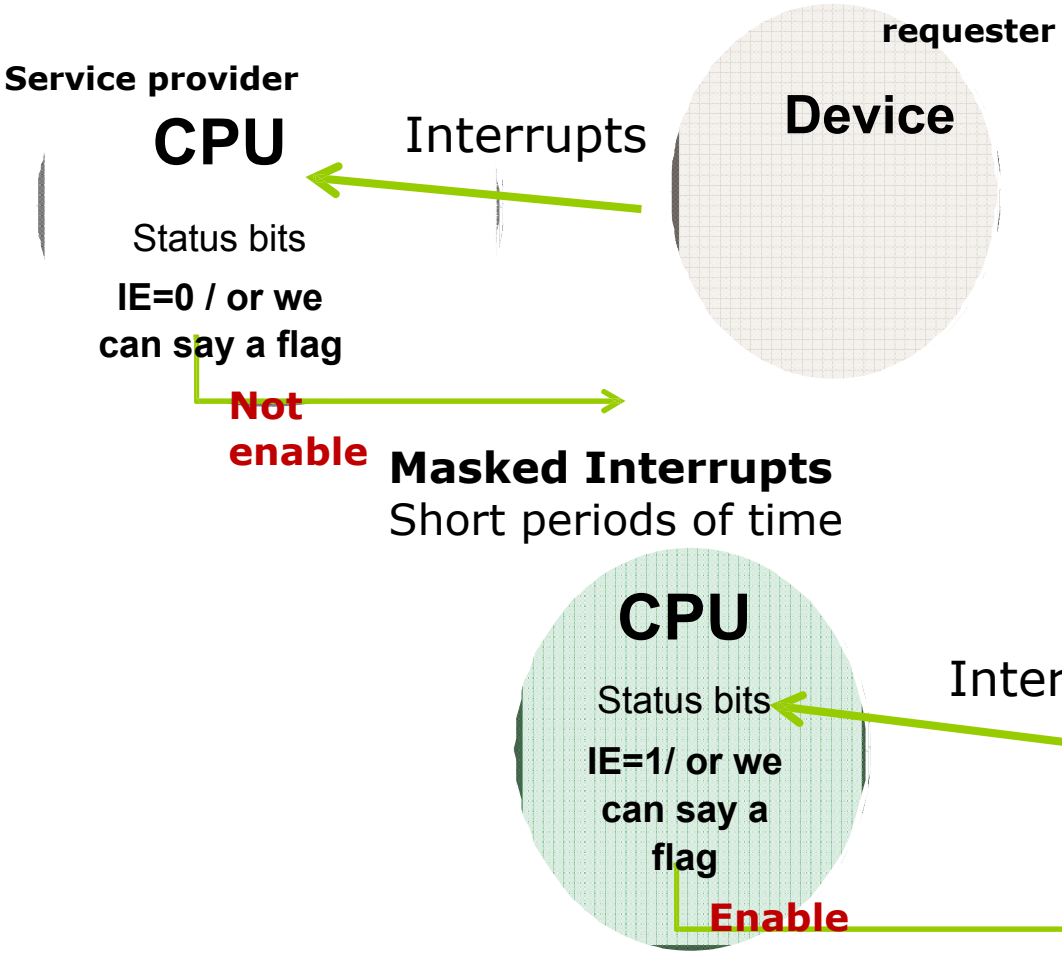
Interrupt is better if processor has other work to do

processor is notified by I/O device (interrupted) when device needs attention

similar to a telephone with bells

In brief: Interrupt is asynchronous, and polling is synchronous

Interrupts Work

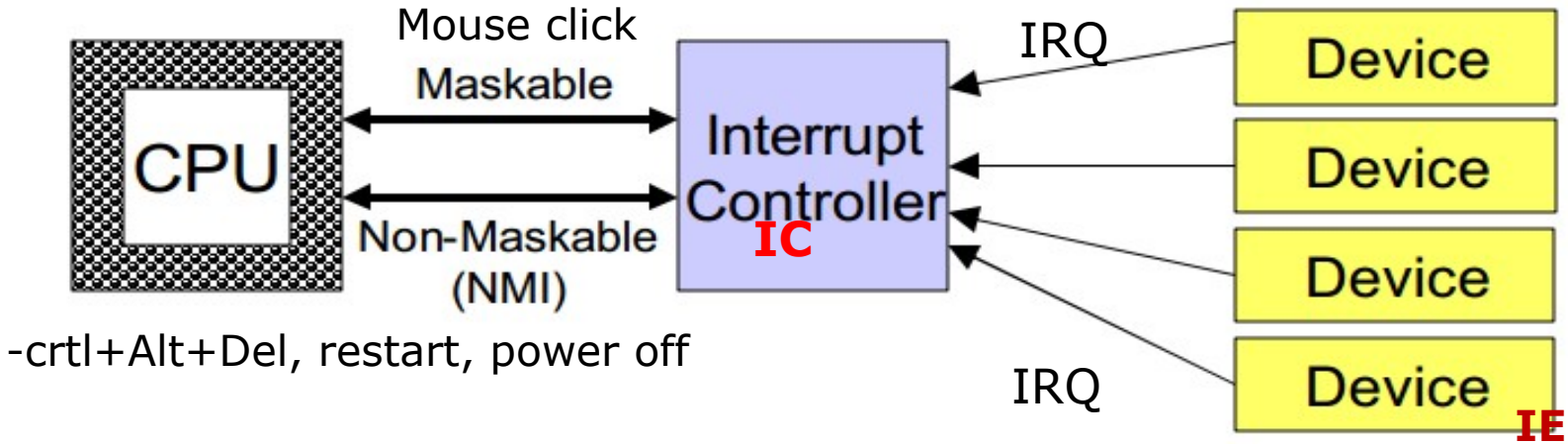


Sends acknowledgement, disable interrupts IE=0, and calls Interrupt Service rou[1]t[2]i[0]ne

Alternative: Interrupt Vector

- Give each device a wire (interrupt line) that it can use to signal the processor
  - When interrupt signaled, processor executes a routine called an interrupt handler to deal with the interrupt
  - No overhead when no requests pending

IE =1



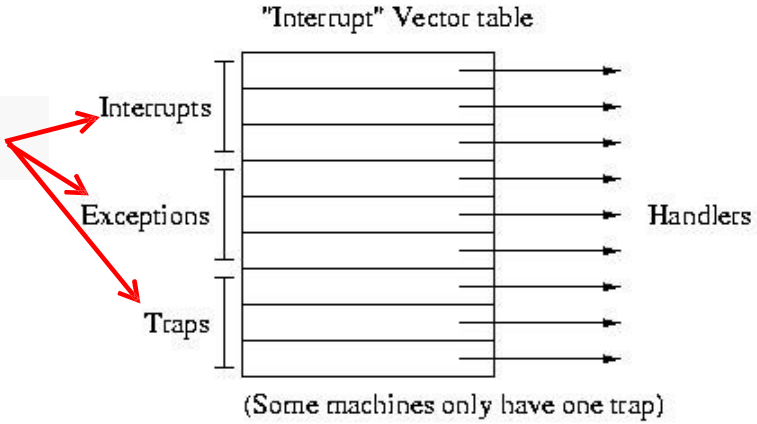
-crtl+Alt+Del, restart, power off

IE =1

CPU sets **CPU IE =1, Device can send interrupt (Not Masked)**

Different Interrupts

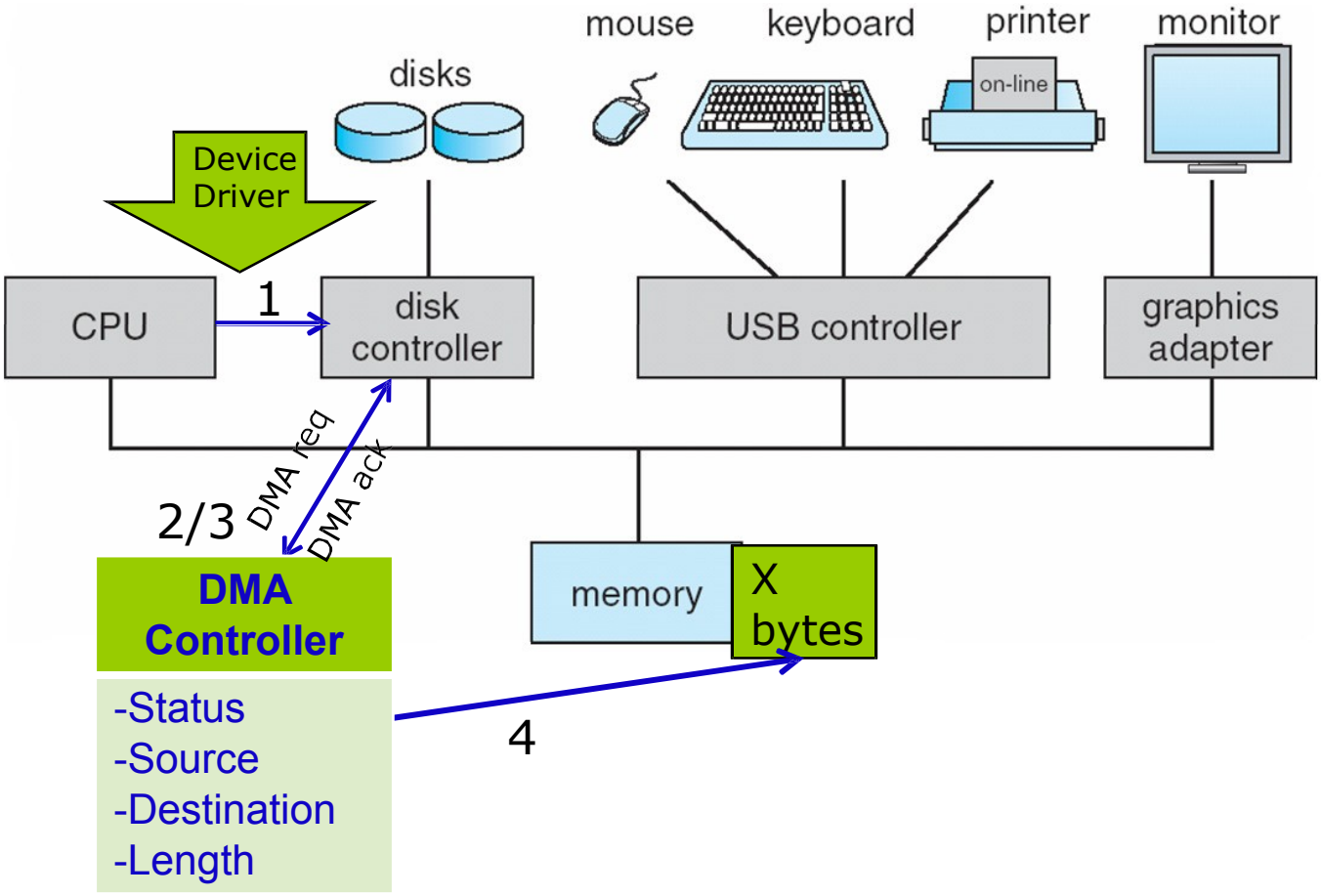
An operating system is interrupt driven



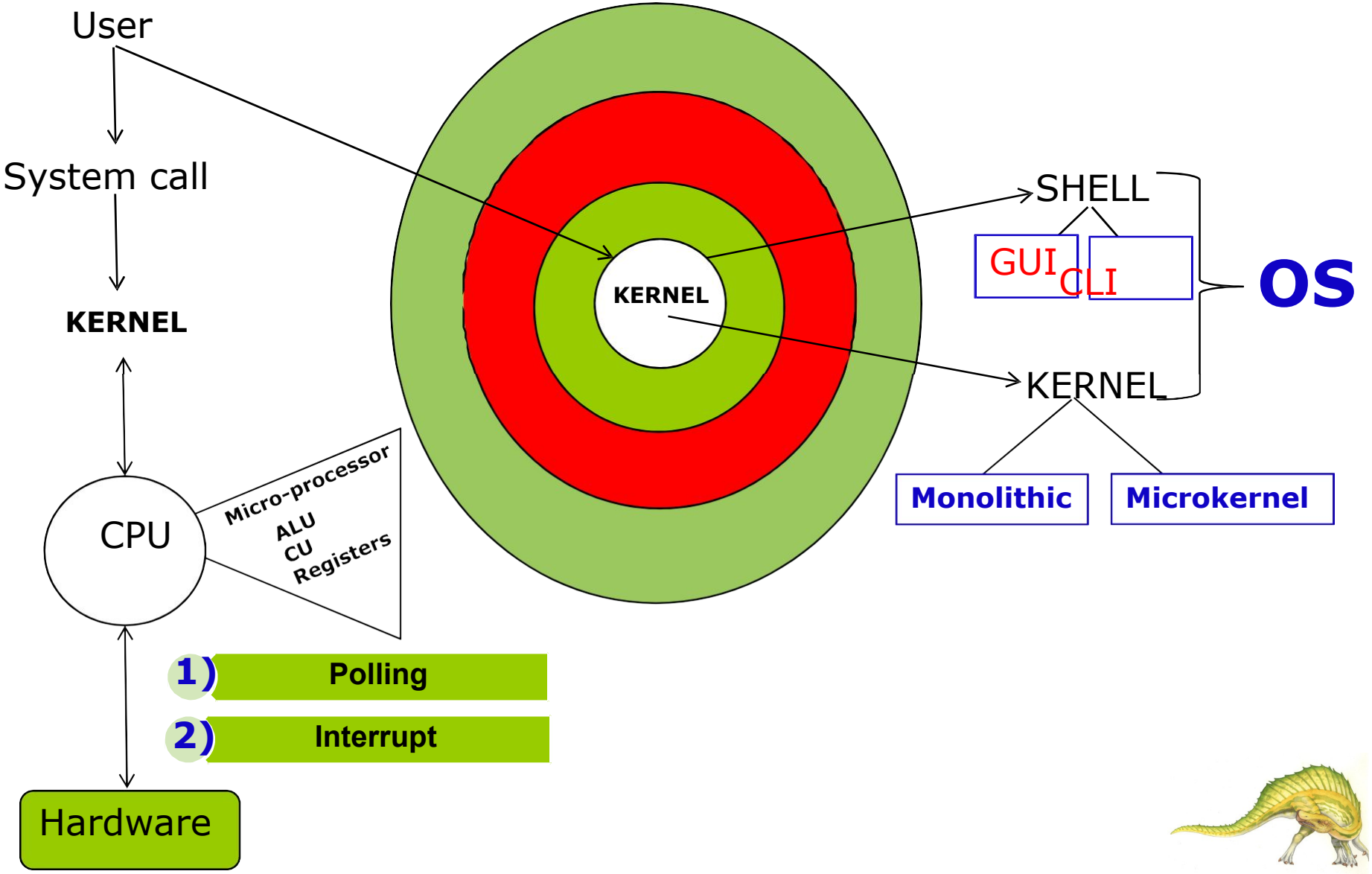
Non-Maskable (hardware)	• **It does not check IE flag or priority.**
Maskable (hardware)	• **It checks IE flag or priority. If IE=1, the processor acknowledges the interrupt..**
Normal interrupt (Software Interrupt)	• a programmer initiated and expected transfer of control to a special handler routine. • **user program can ask for an operating system service** • **unconditional –control always transfer to pre-defined procedure.**
Exception (Software Interrupt)	• **automatically generated unexpected events occur in response to some exceptional condition.** • **illegal program actions that generate an interrupt.**

DMA – Direct Memory Access

- High Speed, I/O device



How does user communicate with OS

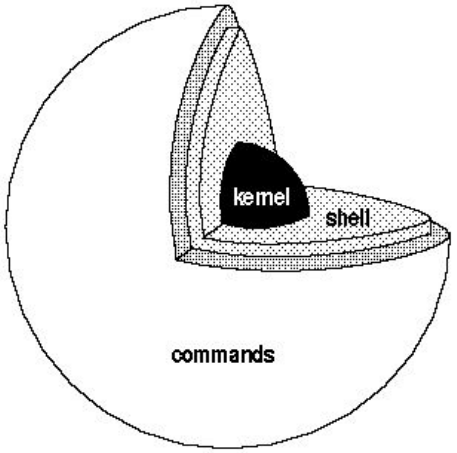


The Kernel and The Shell

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,

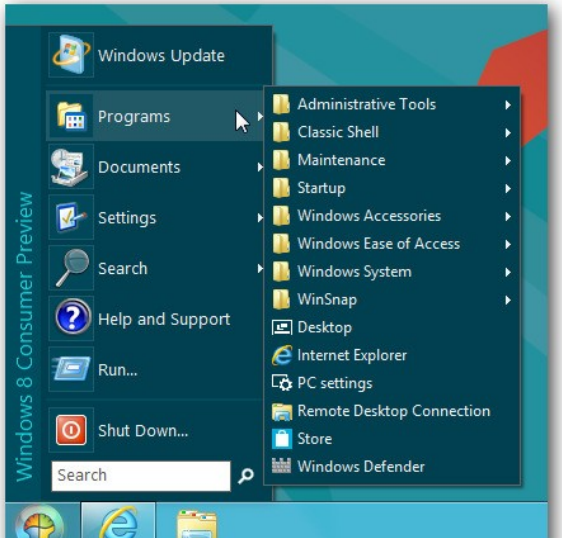
- Linux is a kernel as it does not include applications-file-system utilities, windowing systems and graphical desktops, system administrator commands, text editors, compilers etc.
- So, various companies add these kind of applications over Linux kernel and provide their operating system like **ubuntu,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

An Operating system has two parts: kernel and shell

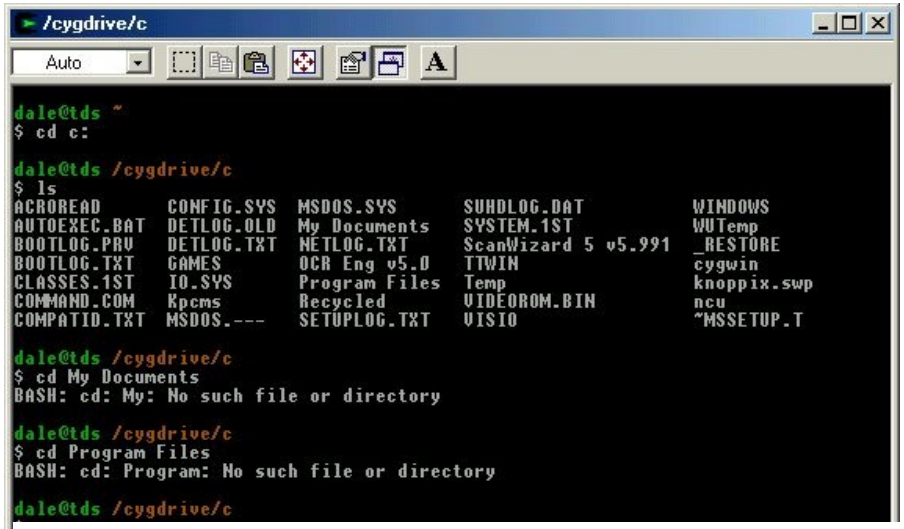
shell (command interpreter): Part of OS

- 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

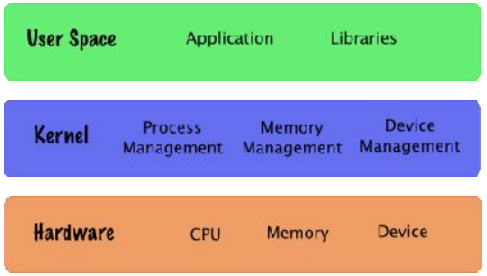


Command line interface (CLI)
Dos/Linux

OS Protections

Transition from User to Kernel Mode

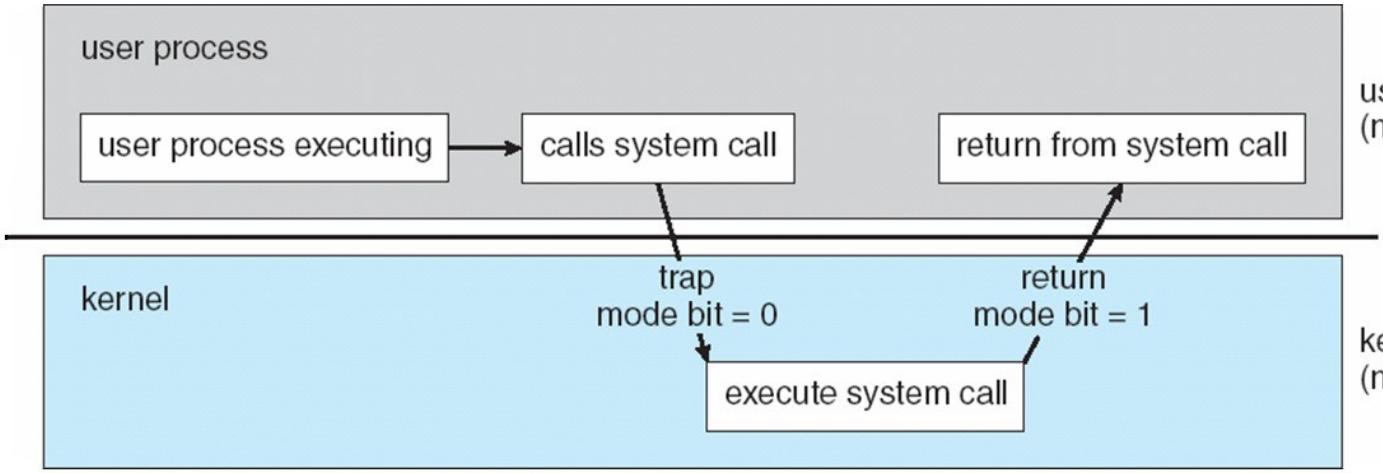
Dual-mode operation allows OS to protect itself and other system components.



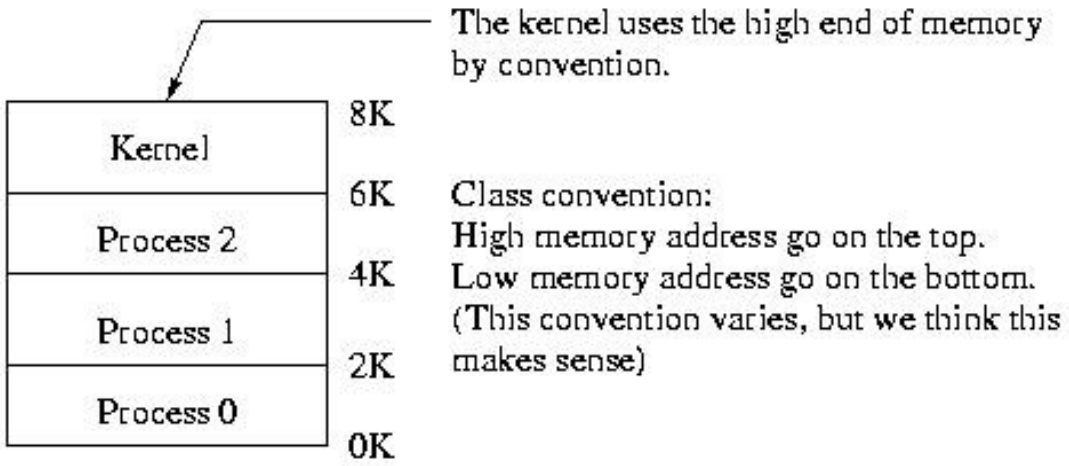
- User mode
- Kernel/Supervisor/Monitor/System Mode.

Mode bit @ Process Status Word (PSW)-Register

- 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 etc.
- System call changes mode to kernel, return from call resets it to user.



Simple Model for Memory Protection



To obtain true protection, need more hardware support.

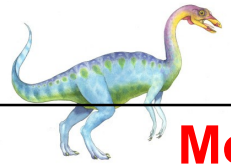
- The goal is to make sure that, for example, process 2 can't go outside of the 4K-6K region.
  - This requires hardware support -- we can't do it in software alone.
  - The simplest model is to add two registers: BASE & LIMIT (assigned by OS)
  - This is the simplest model, most computers are more complex.

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,
if BASE <= X < BASE + LIMIT,
where X is the actual memory address

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

more frequent scheme



Kernel Types

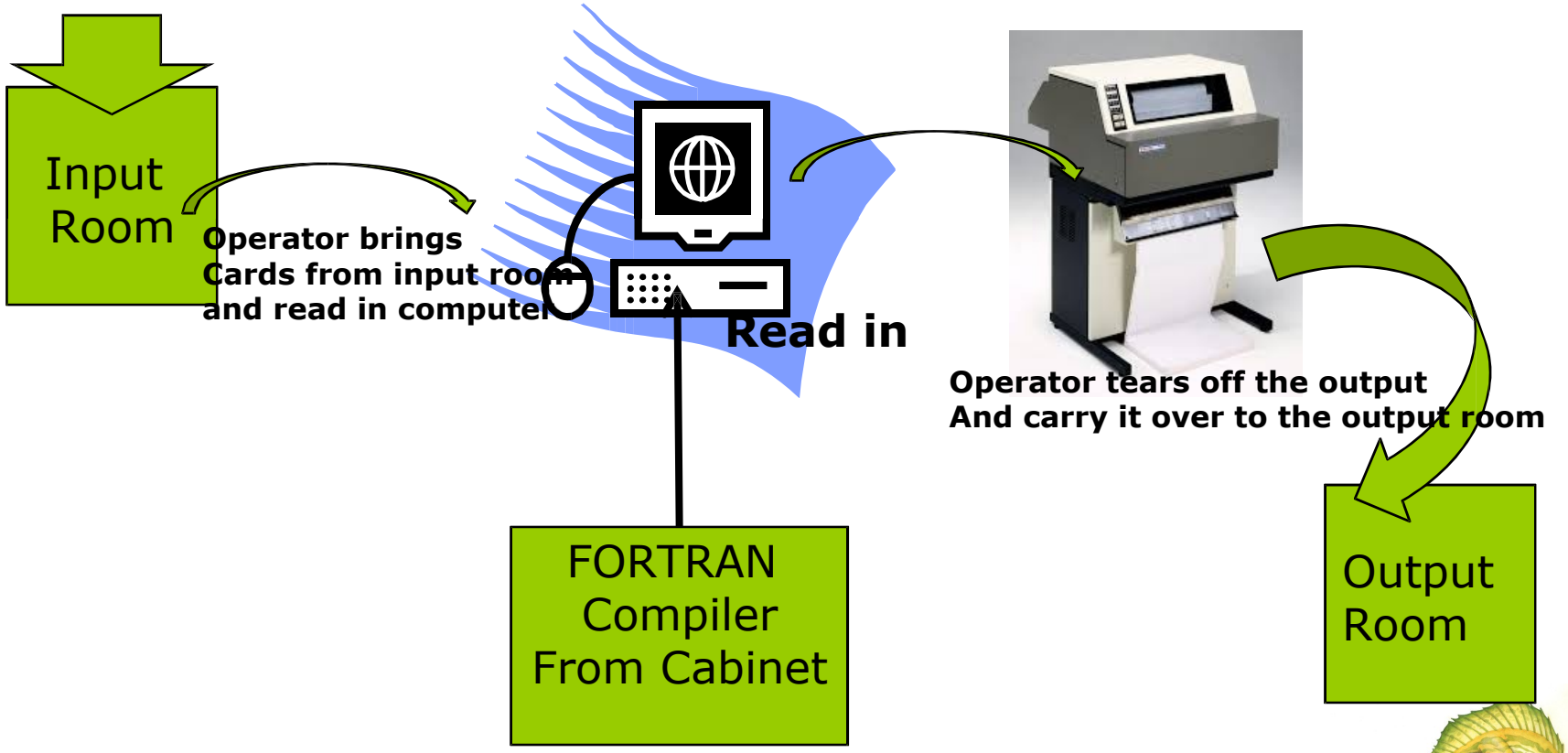
Monolithic Kernel	Micro Kernel
All the parts of a kernel - Scheduler, file system, memory, device drivers etc. are in one unit within the kernel.	Only the important parts are in kernel - IPC, basic scheduler, basic memory management, basic I/O primitives etc. -Others (including device drivers) are in user space.
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

Lecture 2: OS Evaluation

Computers were in Universities / Large companies

Programmers wrote programs (FORTRAN/ Assembly)on paper.

Punched the program into CARDS

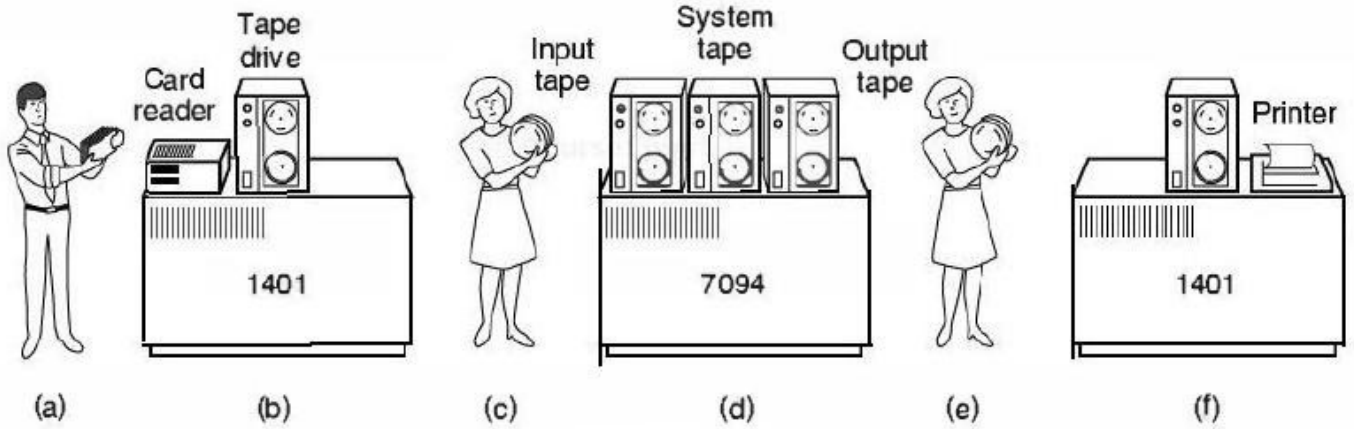


Wasting time while operators were walking around the machine room

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

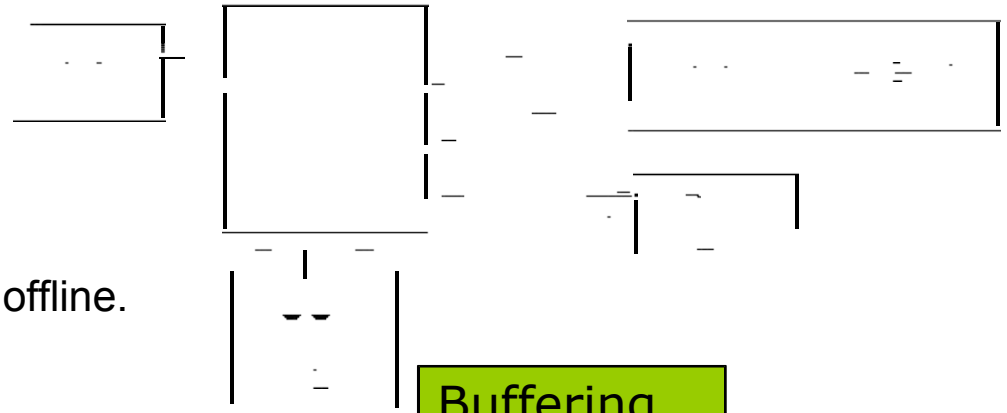
I= Input
C=Computation
O=Output

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?

Batch Systems - Issues

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



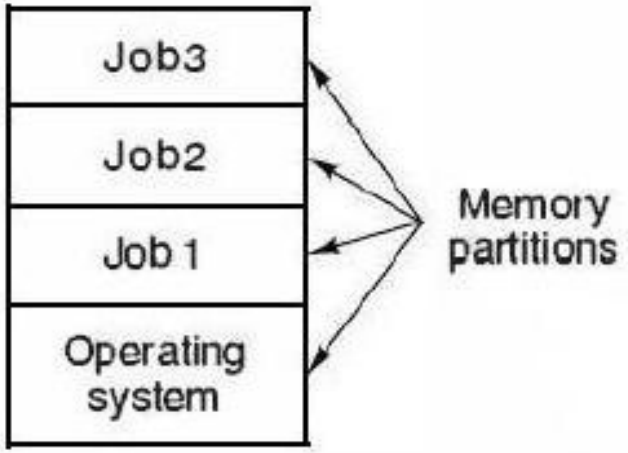
- Offline Processing (Previous)
  - load jobs into memory from tapes,
  - card reading and line printing are done offline.

Buffering

- 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

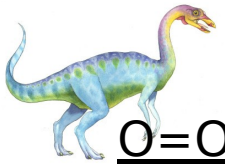
Multiprogramming

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



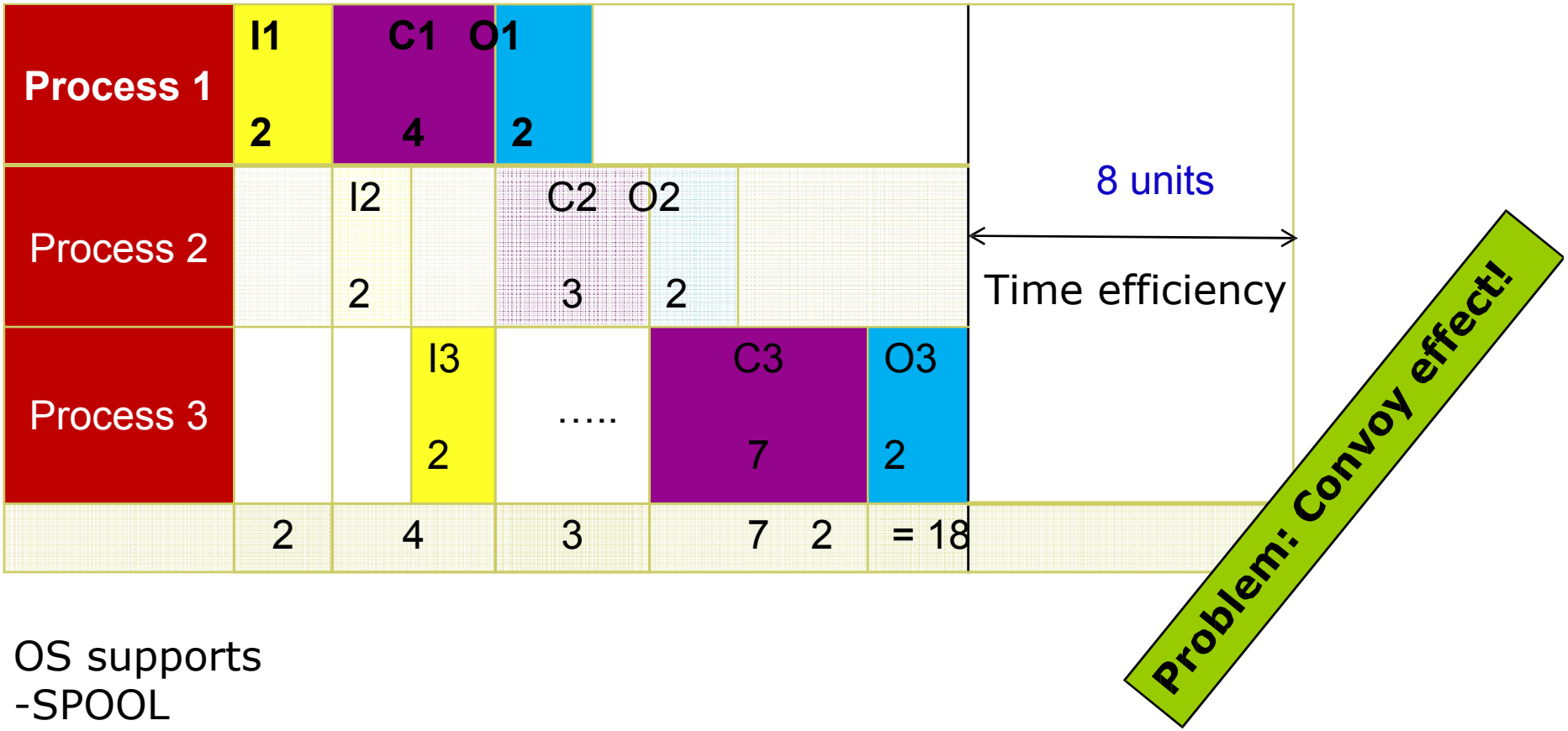
A multiprogramming system with three jobs in memory.

- Requires:
  - **secure memory [will learn @ final]and** I/O for each program.
  - **intervention, if program loops indefinitely**.
  - CPU scheduling to choose the next job to run.[will learn later]

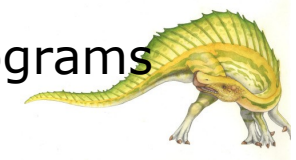


Multiprogrammed Processing

I= Input
C=Computation
O=Output



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

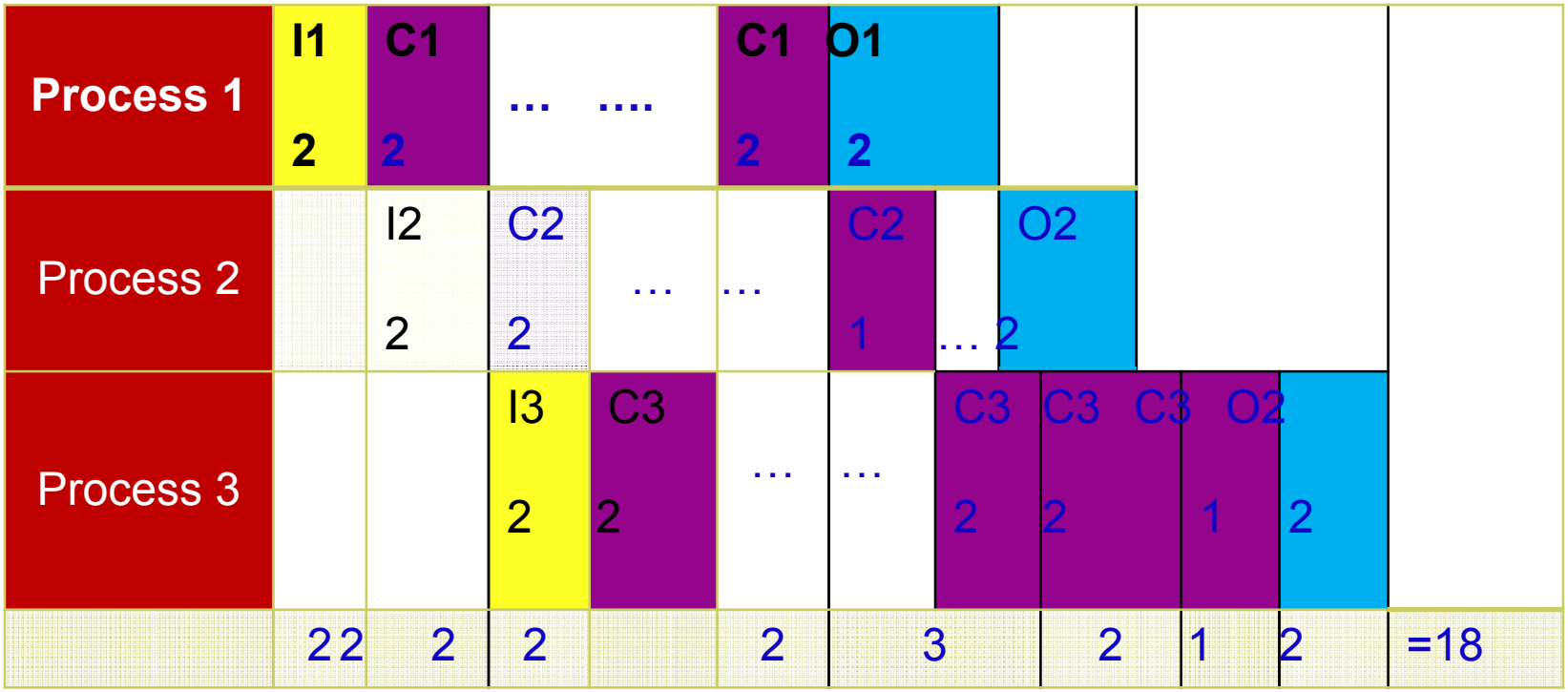


Timesharing

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

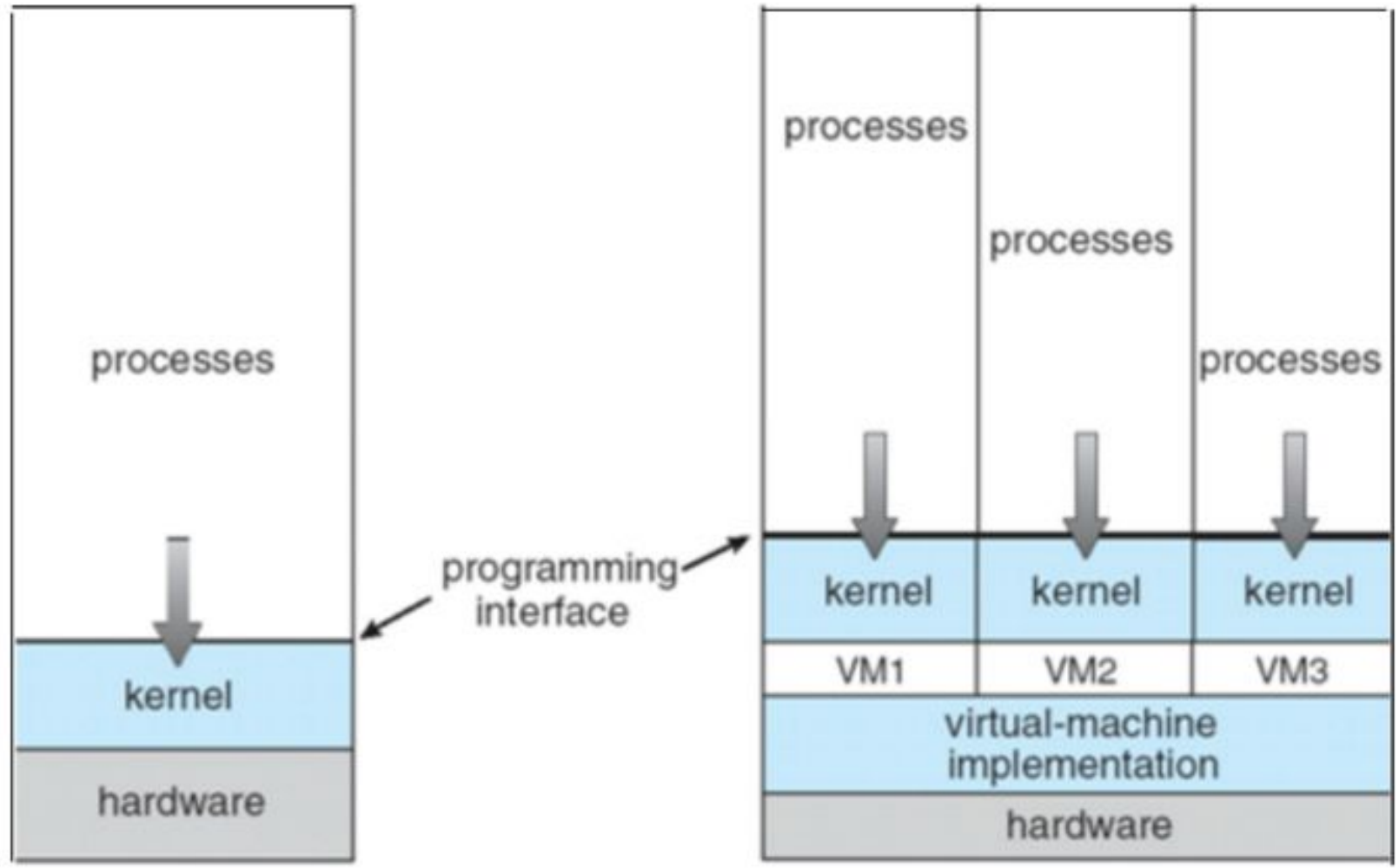
- **Like multiprogramming, but timer device interrupts** after a quantum (timeslice).
  - Interrupted program is returned to end of FIFO
  - Next program is taken from head of FIFO

Time Shared Processing

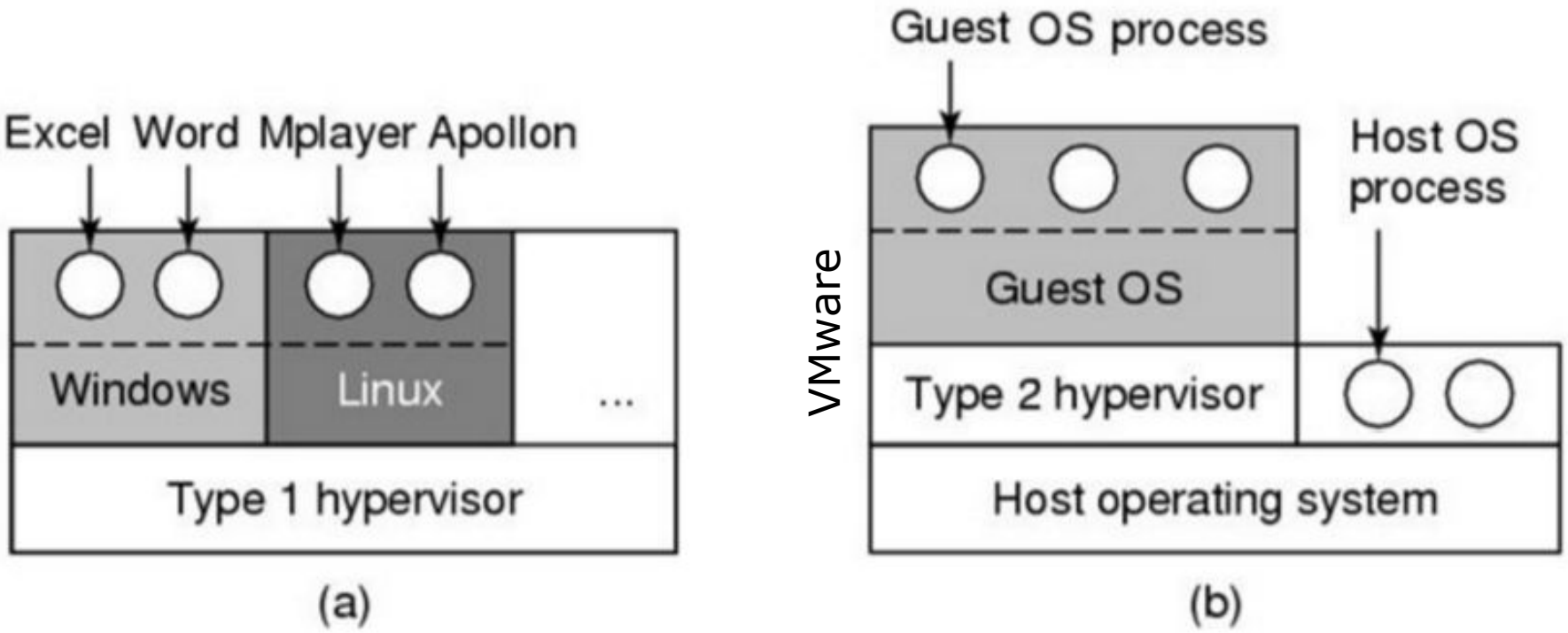


Concept of Virtual Processors
Time Slice =2

Virtual Machines



(a) Nonvirtual machine (b) virtual machine



(a) A type 1 hypervisor. (b) A type 2 hypervisor.

Reading Materials

- Galvin : 1.1, 1.2, 2.1, 2.2, 2.5, 3.3, 3.5.3,3.6
- Galvin: 13, 13.2.1,13.2.2

End of Lecture 1-3