CS 524 A

Introduction to Cloud Computing

Lecture 3: Virtual Machines (Part 1)

OUTLINE

- Virtual vs. Real
- Computer organization
- OS recap
  - A process as a unit of computation
  - Multiprocessing
  - Virtual views: How a process gets to think that it owns
  - the CPU
  - **infinite** memory
  - all devices to itself

American Heritage Dictionary Definition

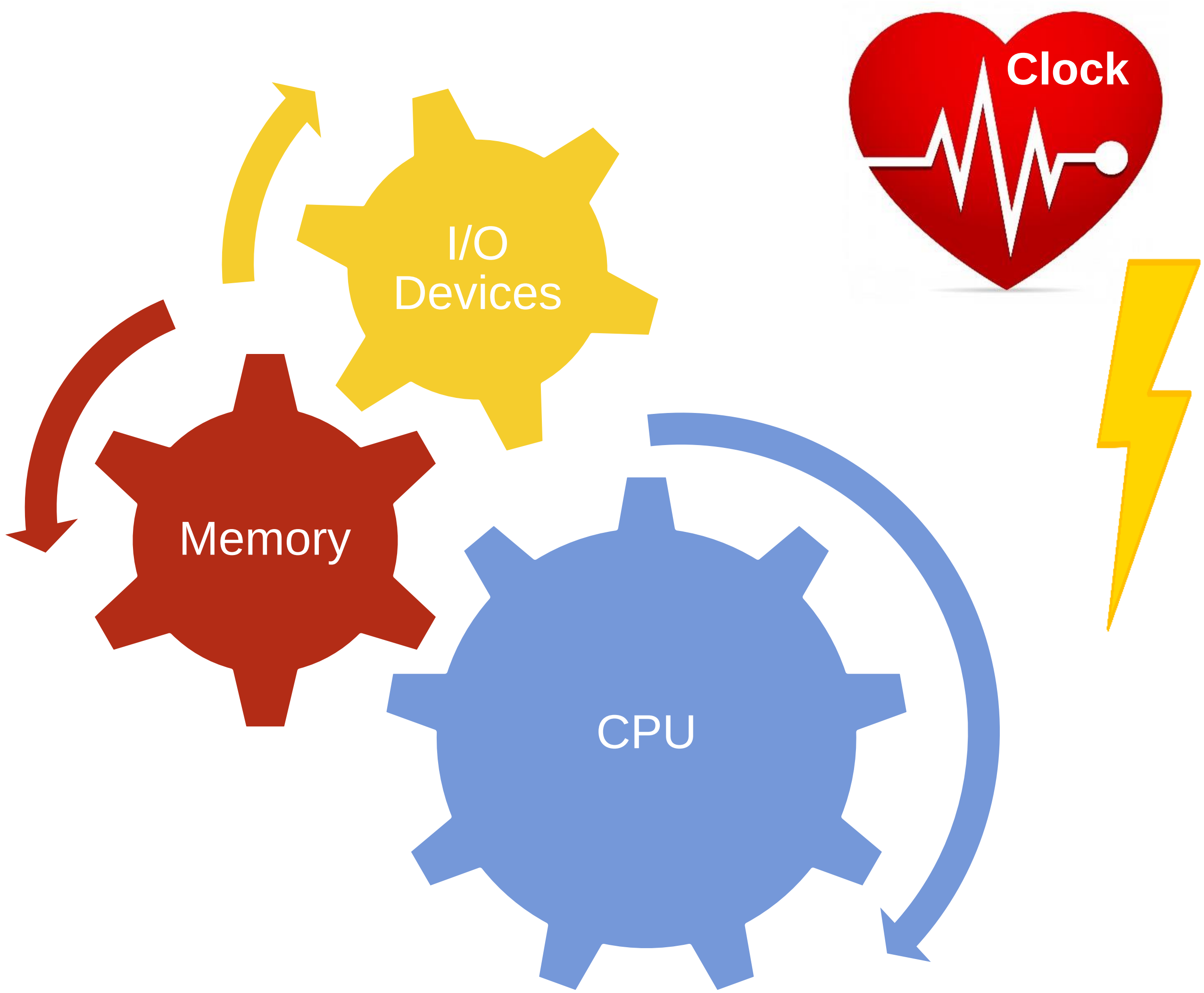
vir·tu·al (vûr□ch□-əl) : □

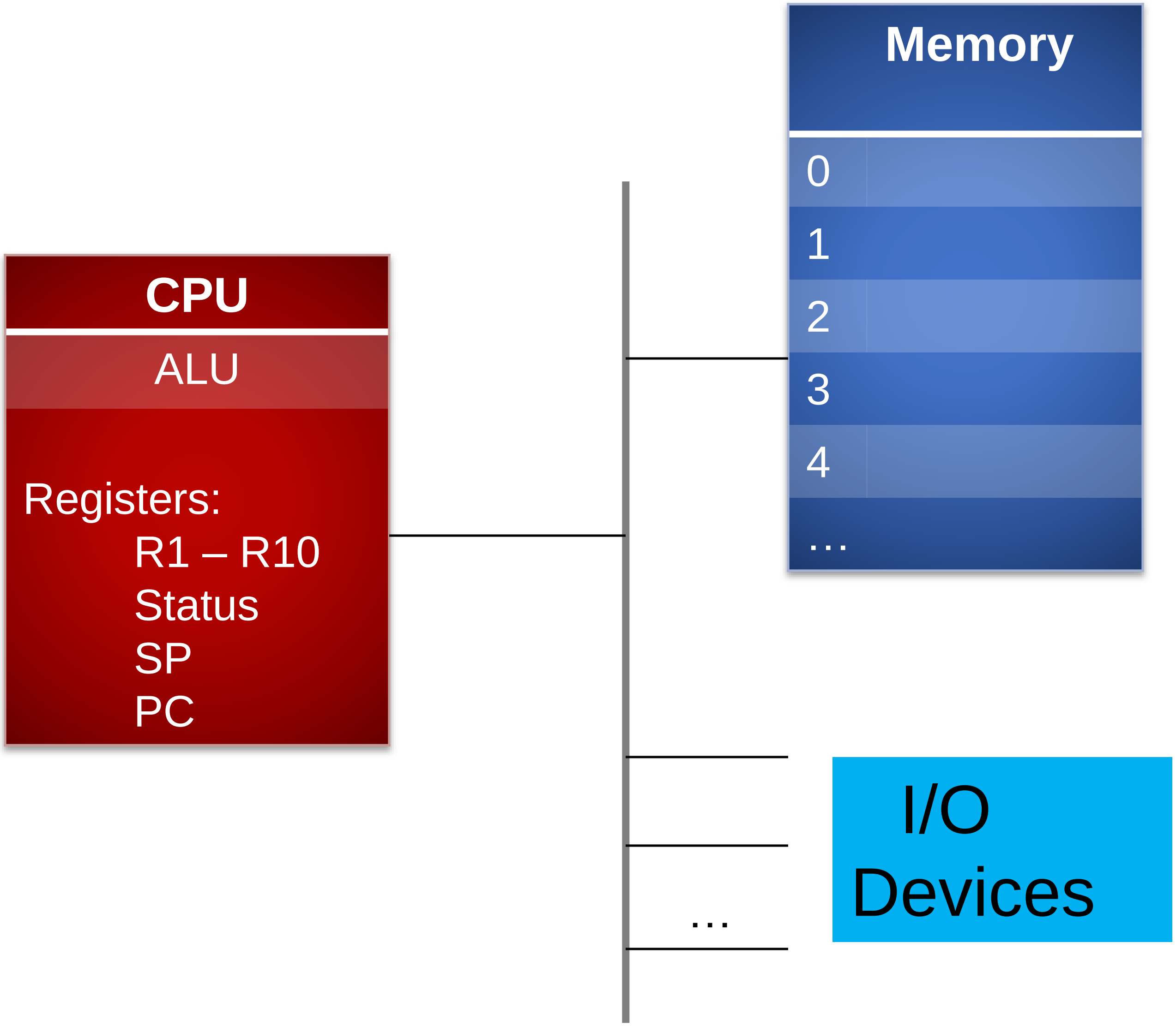
adj. **1.** Existing or resulting in essence or effect though not in actual fact, form, or name: *the virtual extinction of the buffalo.*

2. Existing in the mind, especially as a product of the imagination. Used in literary criticism of a text.

3. *Computers* Created, simulated, or carried on by means of a computer or computer network: *virtual conversations in a chatroom.*

COMPUTER ORGANIZATION




Memory
0
1
2
3
4
…
CPU
ALU
Registers:
R1 – R10
Status
SP
PC
…
I/O Devices

The CPU Loop: first approximation

```
While TRUE
  {
     Fetch the instruction pointed to by the PC;
     Advance the PC to the next instruction;
     Execute the instruction;
   }
```

100000	**LOAD**	**R1**	**@20002**
100020	LOAD	R2	@20010

A Process

- Modern operating systems support *multiprogramming*—that is an ability to execute several programs concurrently on one CPU or simultaneously on several CPUs
- A *process* is a *program in execution:*
  - A program is a cookbook
  - A CPU is a cook
  - I/O devices are cooking utensils
  - A process is making a dish described in the *cook book (a program)*

Operating System Services To a Process

- Program execution (Process or Thread)
- Input/Output (I/O) operations
- File-system support
- Interprocess Communications
- Error detection
- Resource allocation
- Accounting
- Protection

The process's stack and the procedure call

Main line of the process code

100000	LOAD	R1	@20002
100010	LOAD	R2	@20010
100020	STORE	R1	@SP
100030	ADD	SP	#-4
100040	STORE	R2	@SP
100050	ADD	SP	#-4
100060	ADD	SP	#-8
100070	JPR	10000000	
100080	ADD	SP	#16
100090	…	…	…

Stack Frame

Saved PC (100080)
Internal variable 2
Internal variable 1
Parameter 2
Parameter 1

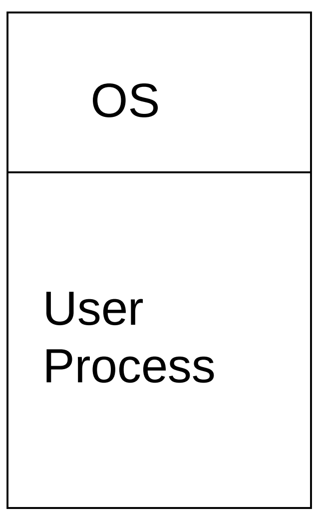
The procedure code

10000000	**LOAD**	**R1**	**@(SP + #20)**
…	…	…	…
10005000	**RTP**		

WHERE IS THIS PROCESS?

- In older systems (up until 1960), there was a place in memory for *one* process. The Operating System loaded it and ran it

Low memory



Problems:

1. Low CPU Utilization
2. A need to program device drivers in each process for the devices it uses
3. Inability to split a program into manageable independent concurrent pieces
4. Effectively, inability to support more than one interactive user

Multiprogramming

Operating System

Process 1

Process 2

...

Process n

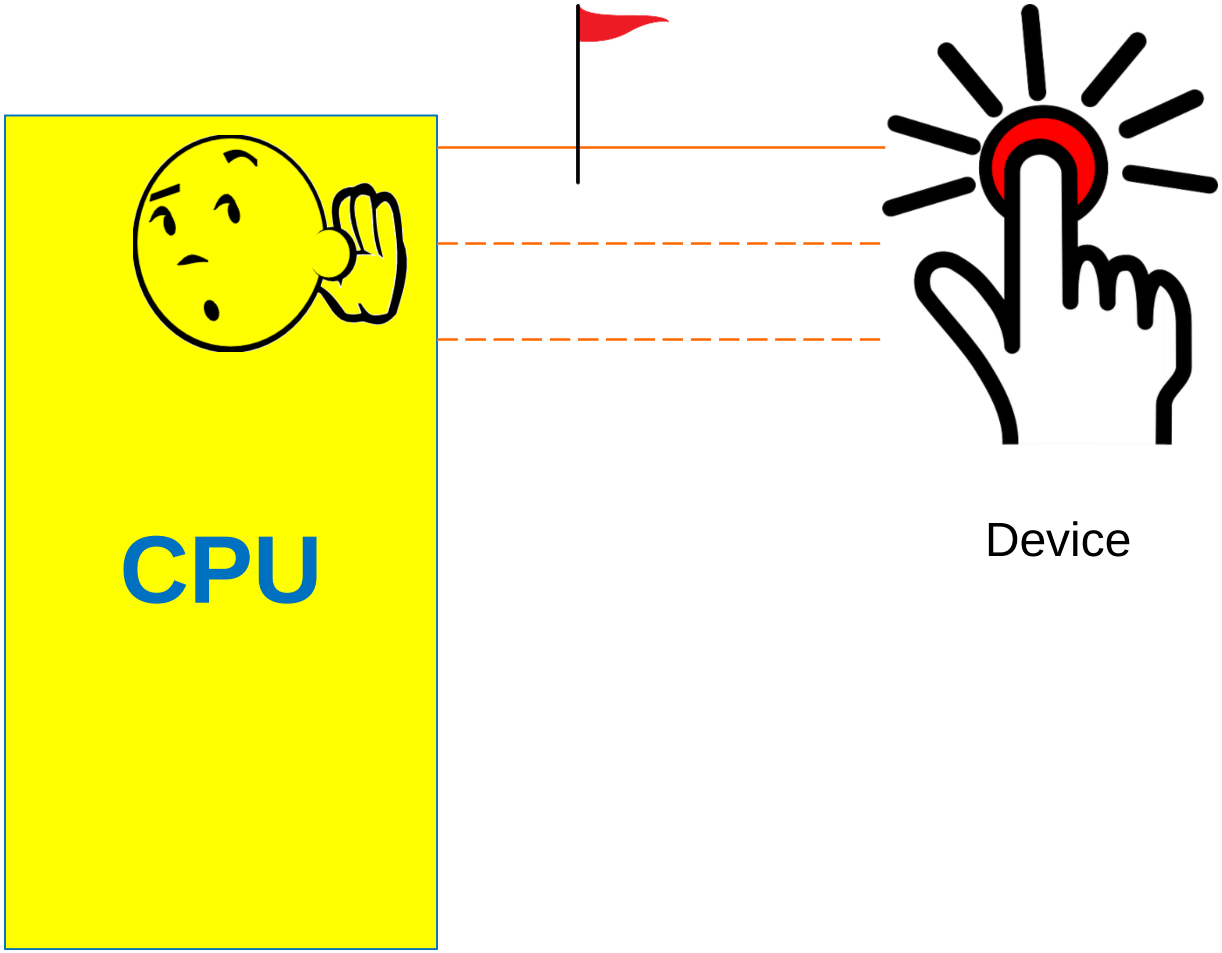
BUT WHILE ONE PROCESS IS WAITING…

- …Others had better use the CPU and run!

- Then the OS **must** get control of the CPU!

INTERRUPTS AND EXCEPTIONS

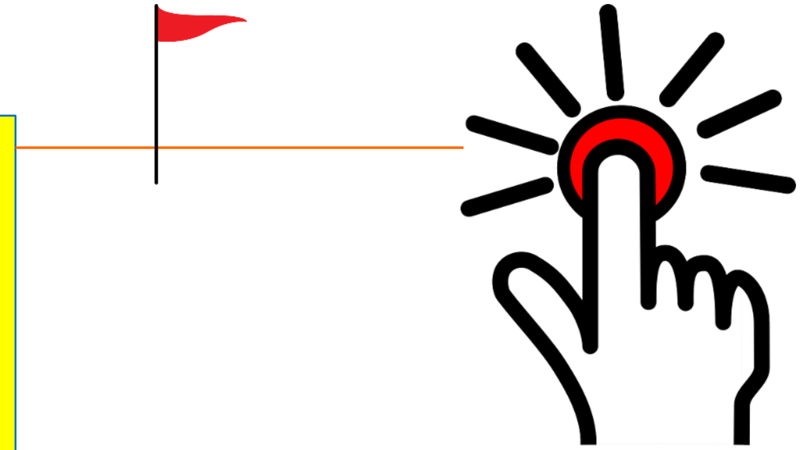
1. Interrupts

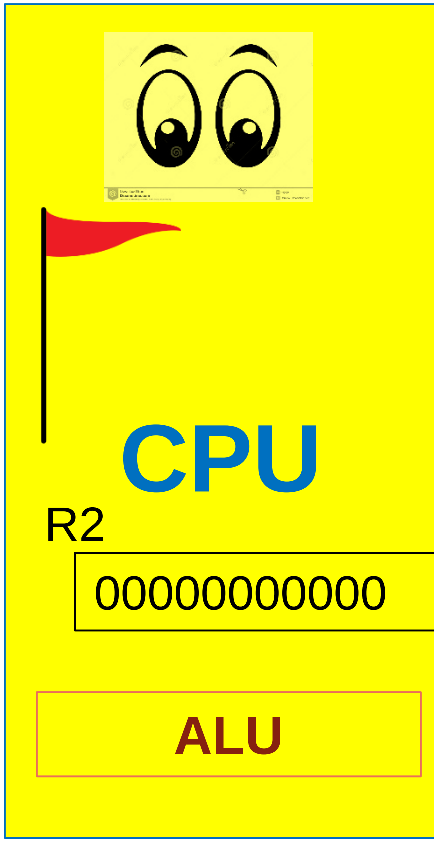


Interrupts occur *asynchronously* with the execution of the code

INTERRUPTS AND EXCEPTIONS

1 Exceptions (examples)





1) Memory: Bus error *"LOAD R1 70000000000"*

2) A non-executable instruction: "*DIV R1 R2*"

3) A *TRAP* instruction

Exceptions occur *synchronously* with the execution of the code

An example: Using a TRAP instruction to set a breakpoint

```
ADD R1, #1        2A 0001 01
STORE R1, @R2     B3 0001 02
ADD R2, #4        2A 0004 02
```

Save the instruction and replace it with *TRAP #1*

```
ADD R1, #1        2A 0001 01
TRAP #1           F1 0001 02
ADD R2, #4        2A 0004 02
```

The second approximation of the CPU loop

```
While TRUE
  {
     Fetch the instruction pointed to by the PC;
     Advance the PC to the next instruction;
     Execute the instruction;
     If  (an exception #x has been raised) OR
          ((an interrupt #x has been raised) AND interrupts are enabled)
       {
         If it is an exception
           Restore the previous PC value;
         Save the STATUS Register and the PC @SP;
         PC = Interrupt_Vector[#x];
       }
   }
```

Interrupt vectors

#1	**2300000**
#2	**7000000**
…	
#x	**30000000**
…	

An interrupt service routine

30000000	**DISI**
30000004	**SAVEREGS @SP**
…	…
30006000	**RESTREGS @SP**
30006004	**ENI**
30006008	**RTI**

An interrupt stack frame

…
Saved PC
Saved Status Register
Saved registers' image

A simple *hypervisor-debugger*

The *Go()* code

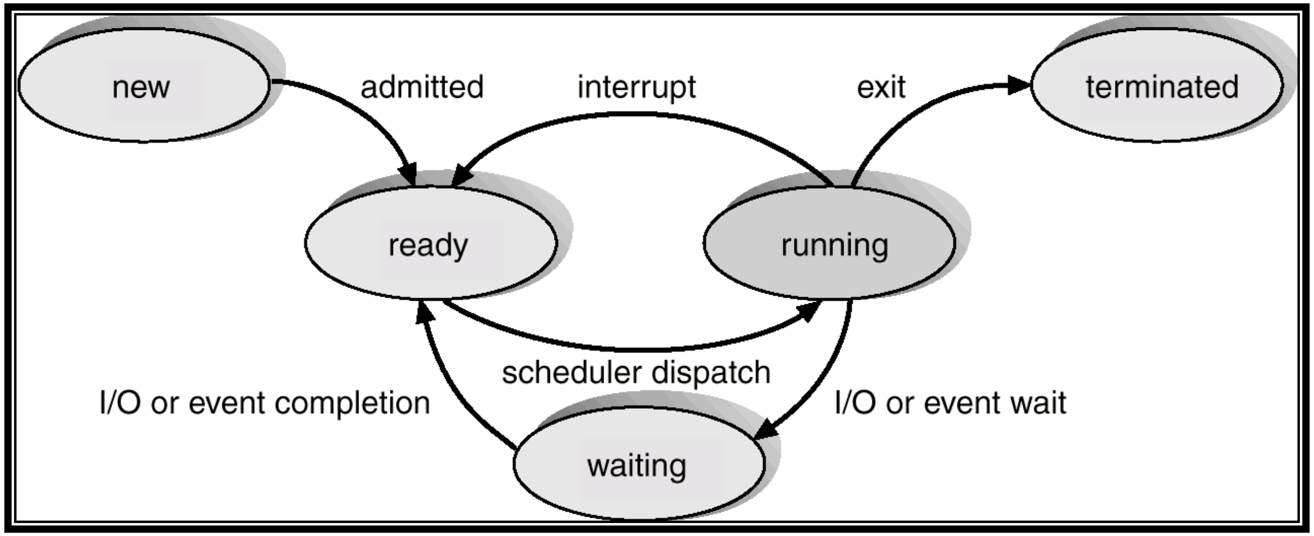
```
Go()
 {
    #DISI; /* disable interrupts */
    registers_struct.SP =
       registers_struct.SP +4;

   #RESTREGS @registers_struct;
   #STORE PC  @SP+8;
   #STORE STATUS @SP+4
     /* place the PC and Status in
        their proper place within
        the stack frame */

   #ENI; /* enable interrupts */
   #RTI;
  }
```

The service routine for *TRAP #1*

The TRAP #1 vector stores the address of the TRAP_1_Service Routine:

```
#DISI; /*disable interrupts*/
#SAVEREGS @registers_struct;
#ENI;  /*enable interrupts */
display(registers_struct, PC,
 STATUS);
command_line();
```

The process lives through these states

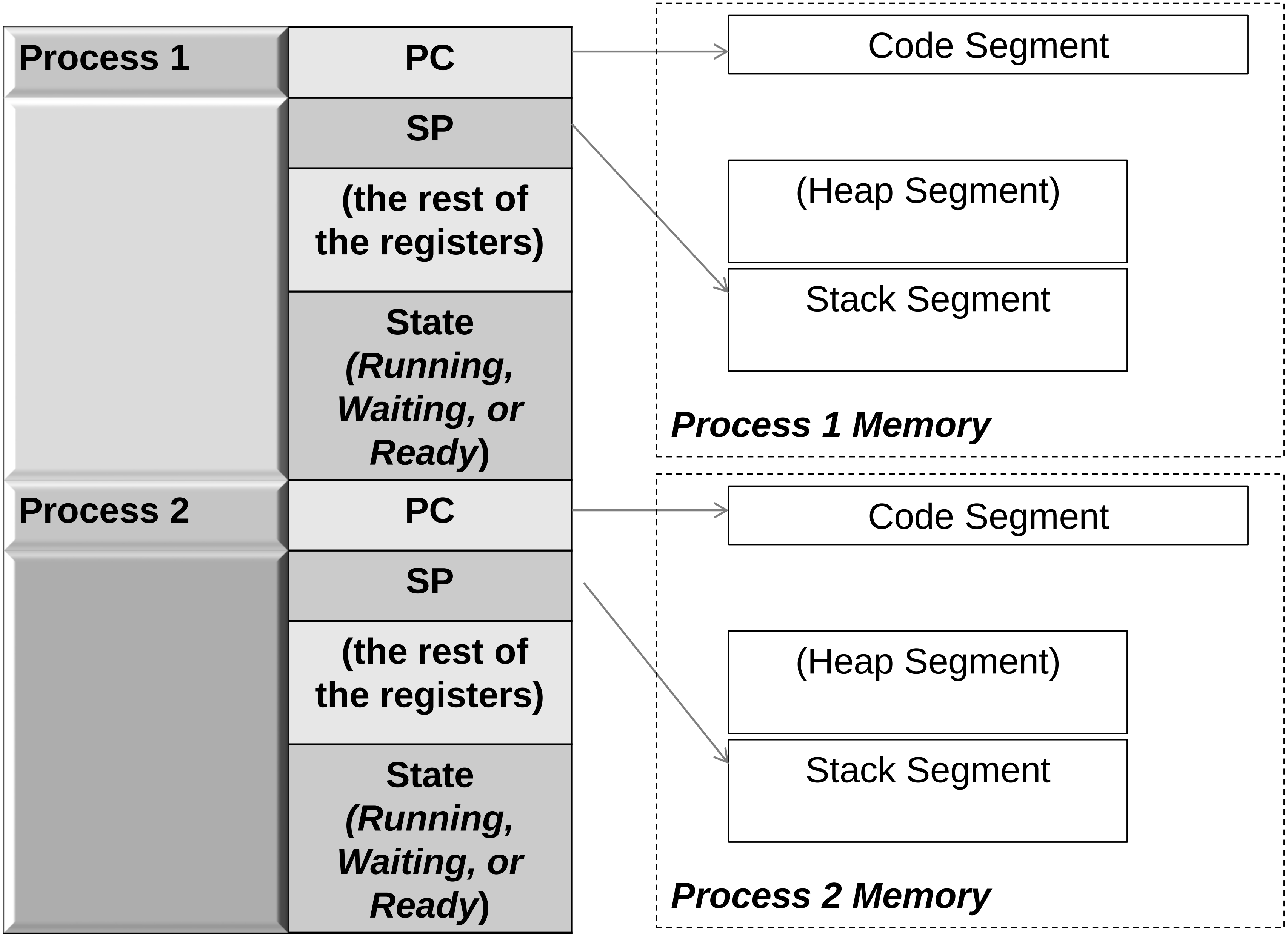


From Silberschatz & al.:
Operating Systems Concepts

The process information

Process ID
Process state (*ready*, *running*, etc.)
All the registers (including PC and SP)
Memory management information
File information
Children processes information
... many other things ....

The process table

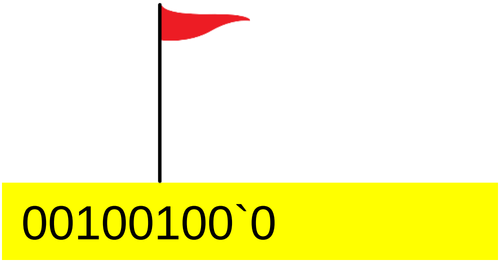


PRIVILEGED INSTRUCTIONS

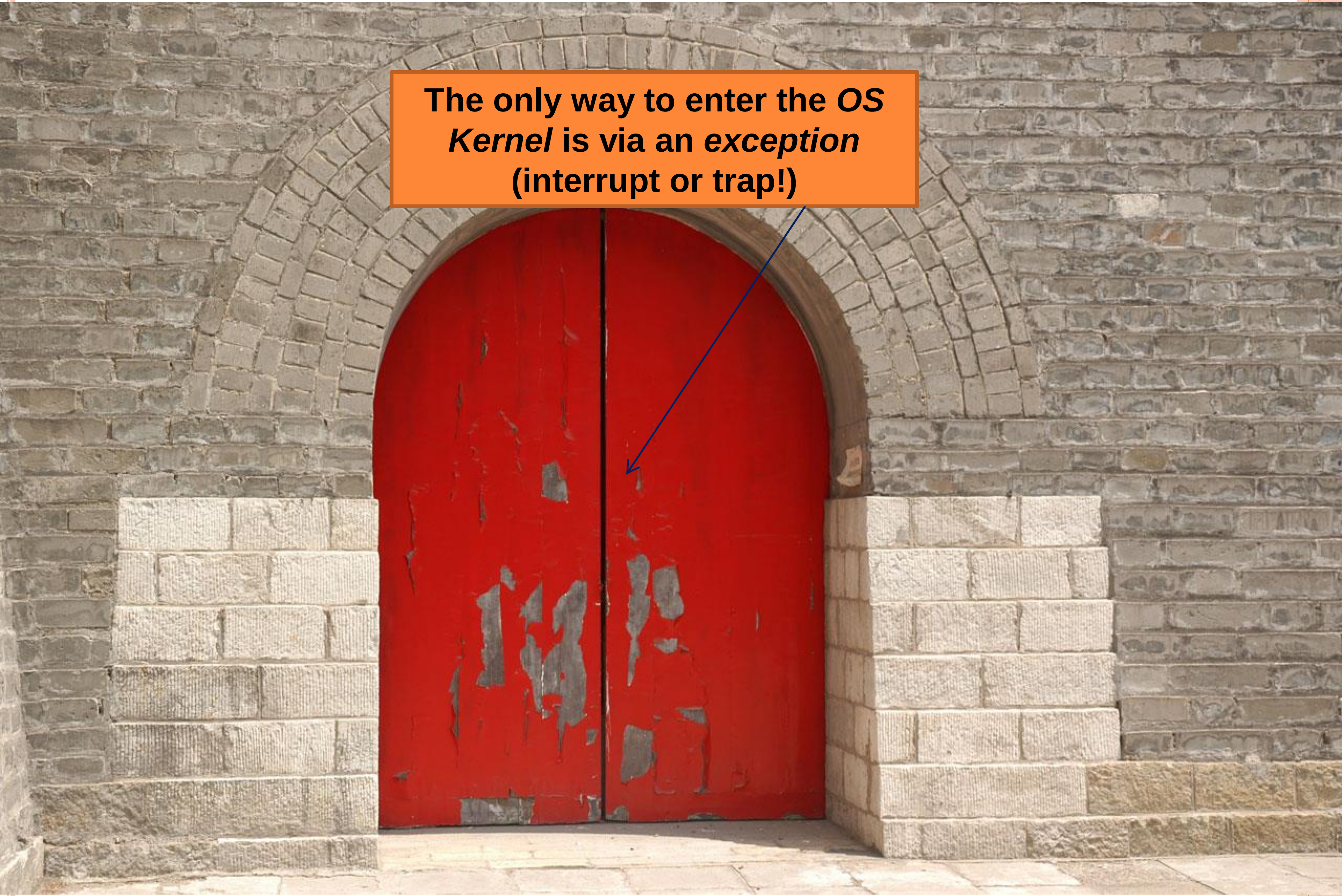
- Instructions that deal with processing interrupts, changing the status register, performing memory management, and the like are mission-critical.
- Critical instructions, by a long-established convention, require CPU processing in a special—***supervisory*** or ***system***—mode
- A CPU may also have a special set of registers, reserved for the system mode. A separate, **supervisory stack pointer** points to a separate stack
- All exception processing is performed **only** in supervisory mode**; an attempt to execute a privileged instruction in a user mode is either ignored [BAD!] or it forces an exception**
- (A *Unix* example) Each process therefore actually has two stacks: a ***user stack*** and a ***system stack***.

CPU Mode (an essential security feature)

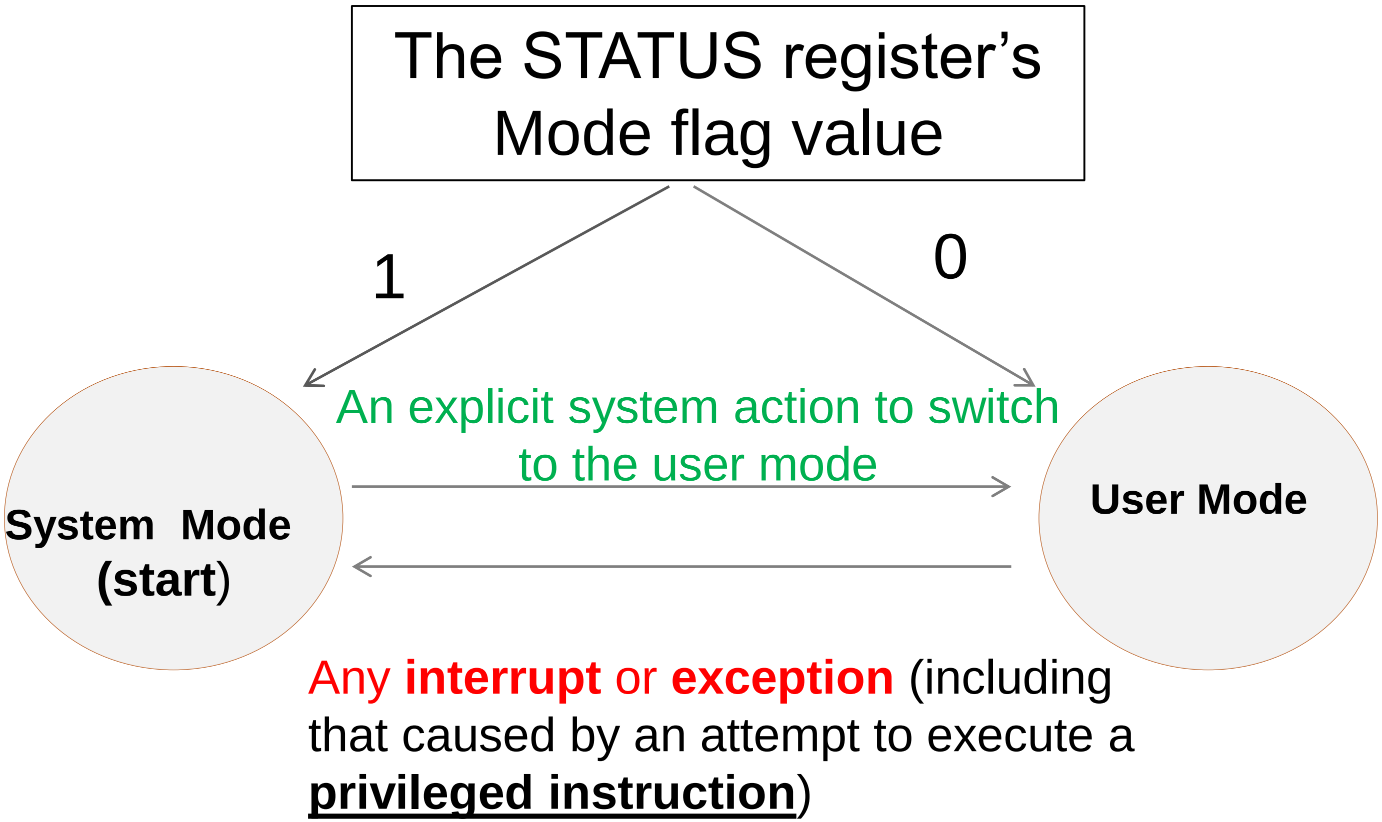
- To ensure that *only* the OS can execute system code (interrupt processing, memory manipulation, etc.), modern CPU execute the system and user code in different modes
- The mode is typically indicated by a flag in the STATUS register



The only way to enter the *OS Kernel* is via an *exception* (interrupt or trap!)

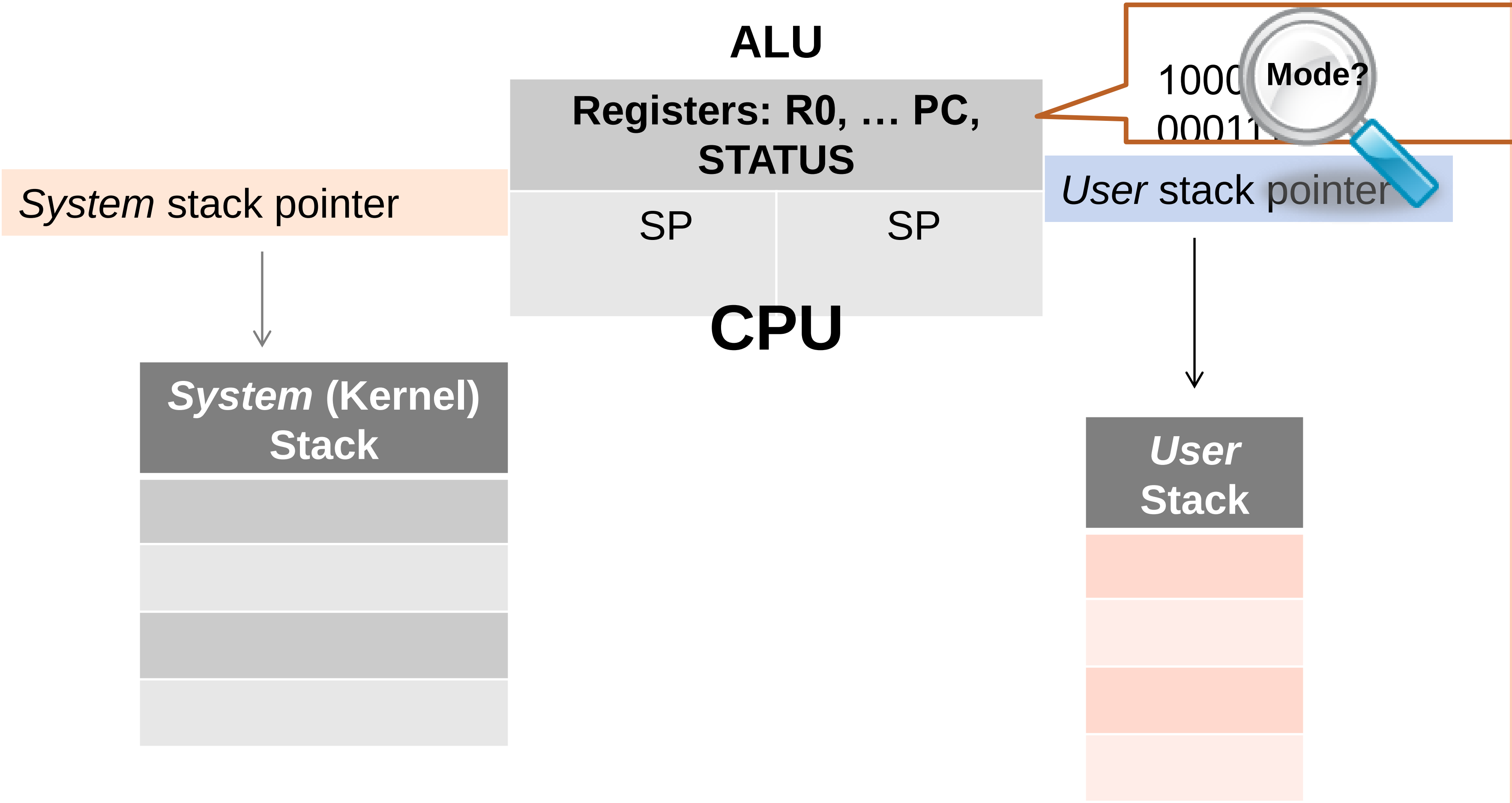


The *CPU mode* state machine



The System Mode and the User Mode are associated with separate stacks

The modified CPU and the two process stacks

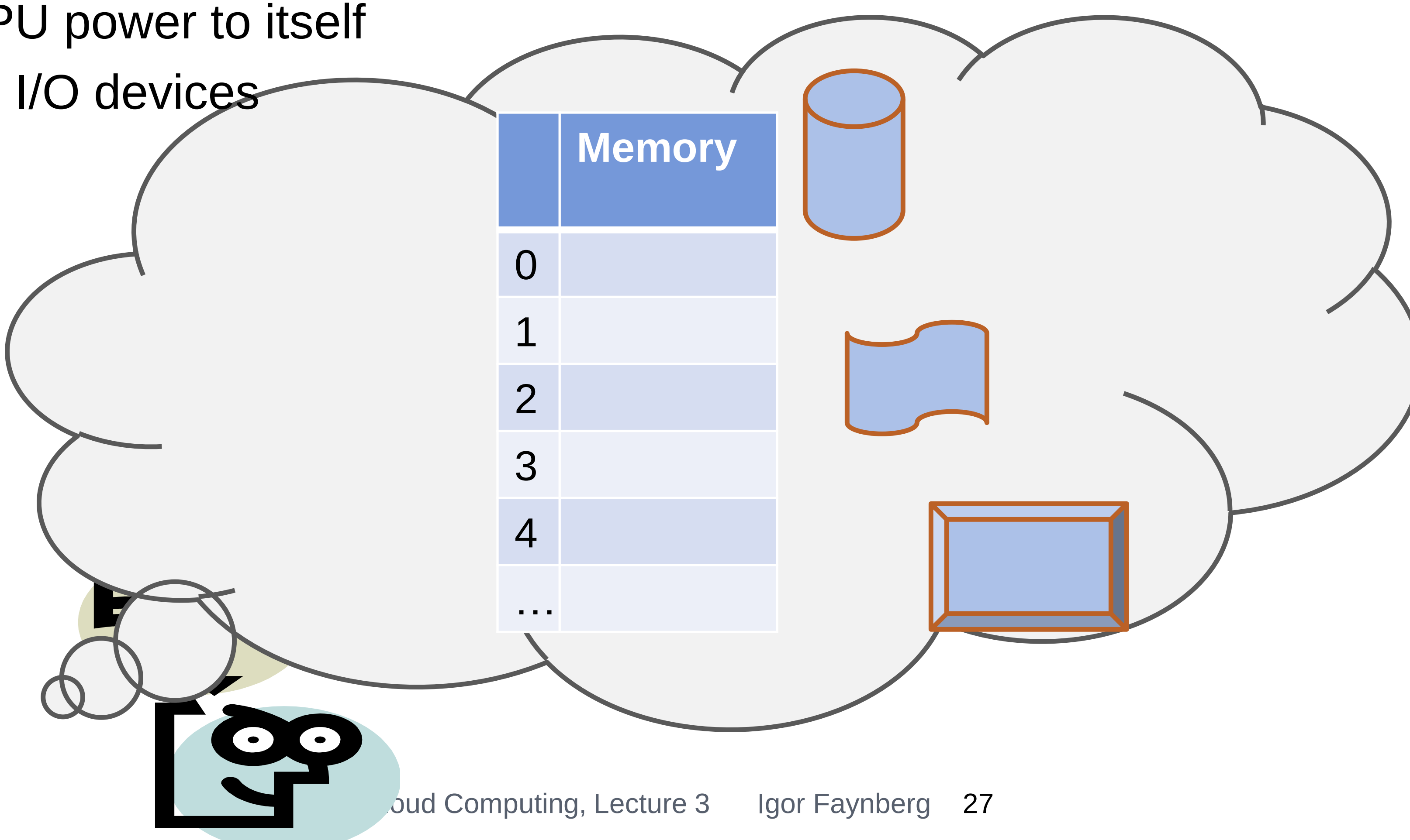


The CPU loop—the final version

```
While TRUE
  {
     Fetch an instruction pointed to by the PC;

     If the instruction is valid AND
        the instruction is appropriate for the present mode AND
        the parameters are valid for the operation
          {
             Advance the PC to the next instruction;
             Execute the instruction;
           }
      else
         raise an appropriate exception;
     If (an exception #x has been raised) OR
        (an interrupt #x has been raised) AND interrupts are enabled
       {
         Save the STATUS register and PC on the system stack (@SP);
         Switch to the system mode;
         PC = Interrupt_Vector[x];
       }
  }
```

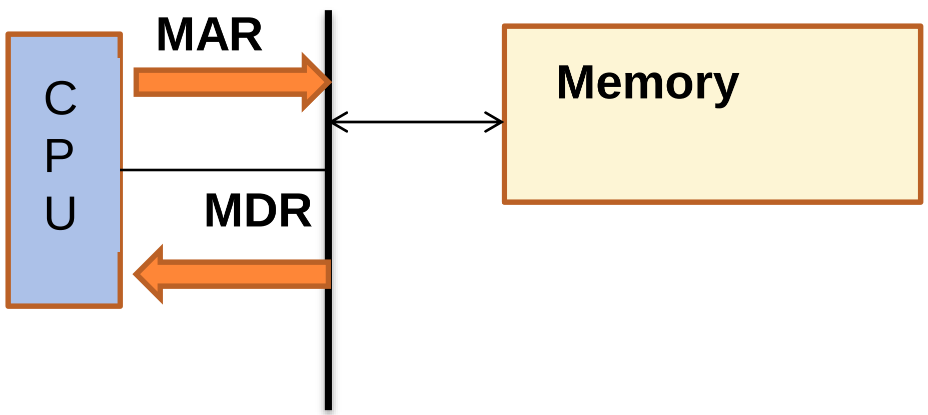
VIRTUALIZATION IN MODERN OPERATING SYSTEMS

- Each *process* thinks that it has
  - huge contiguous memory, starting from address *0*
  - CPU power to itself
  - All I/O devices

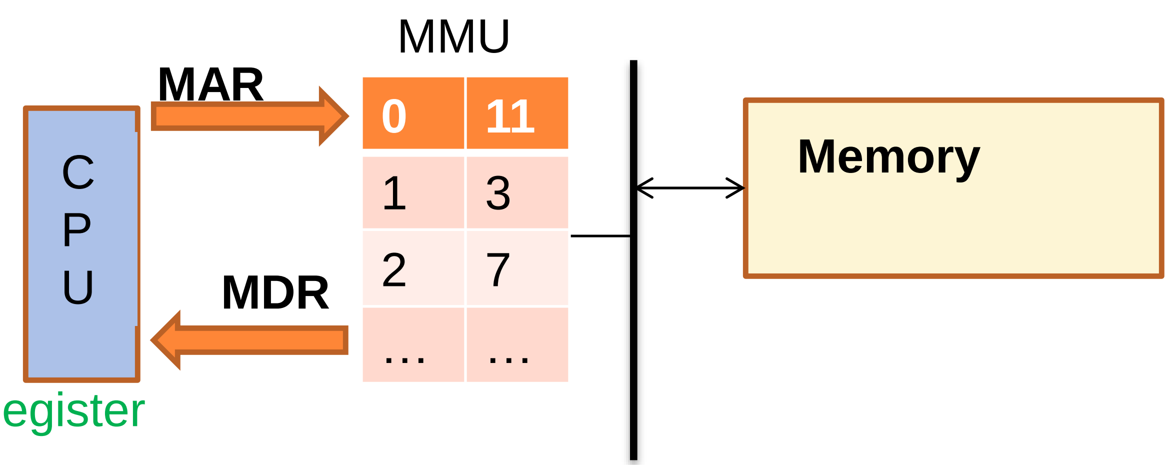


	Memory
0	
1	
2	
3	
4	
…	

HOW IS THIS MEMORY *VIRTUALIZATION* ACHIEVED?



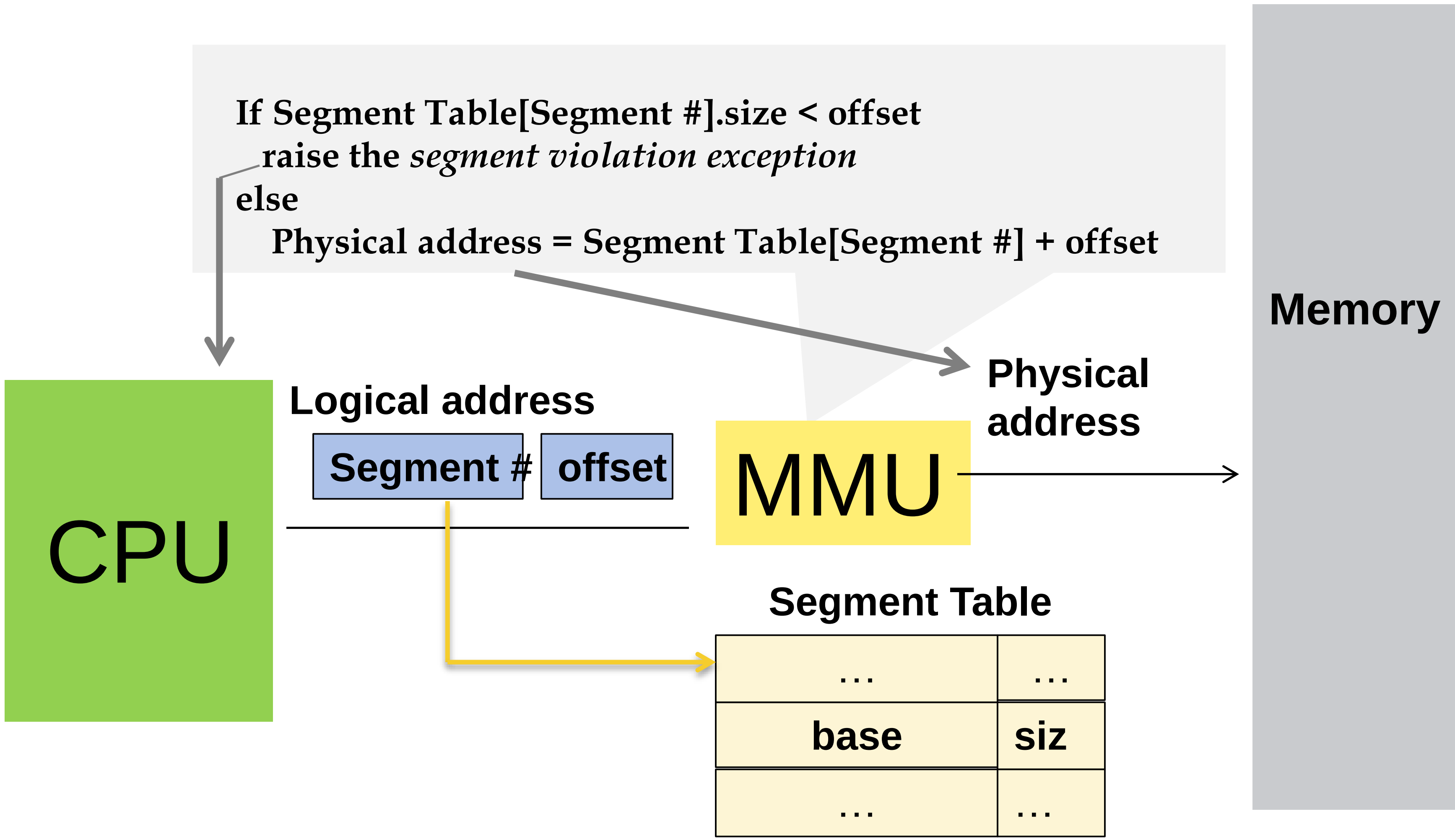
Memory Management Unit (MMU) translates a virtual address into a "real" (physical) address



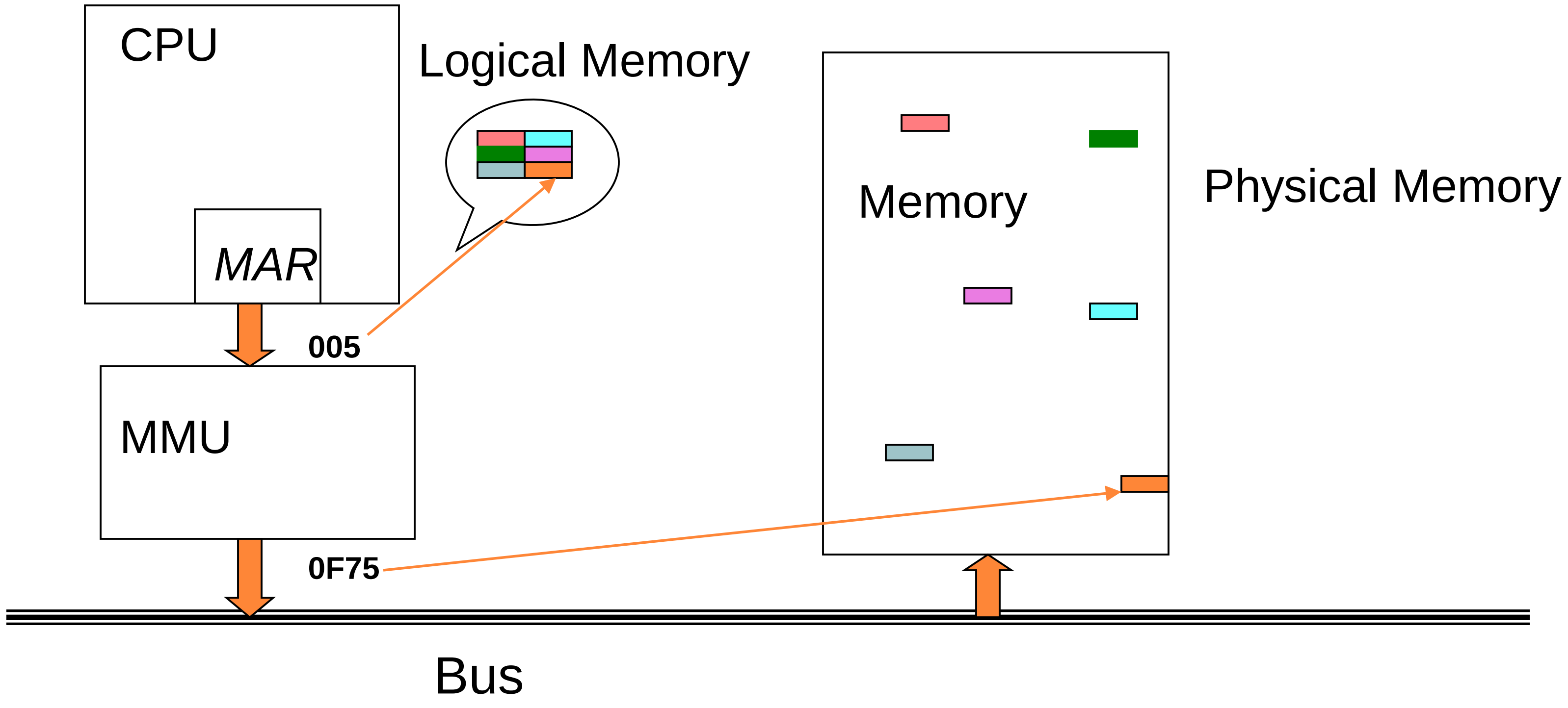
MAR: Memory Address Register

MDR: Memory Data Register

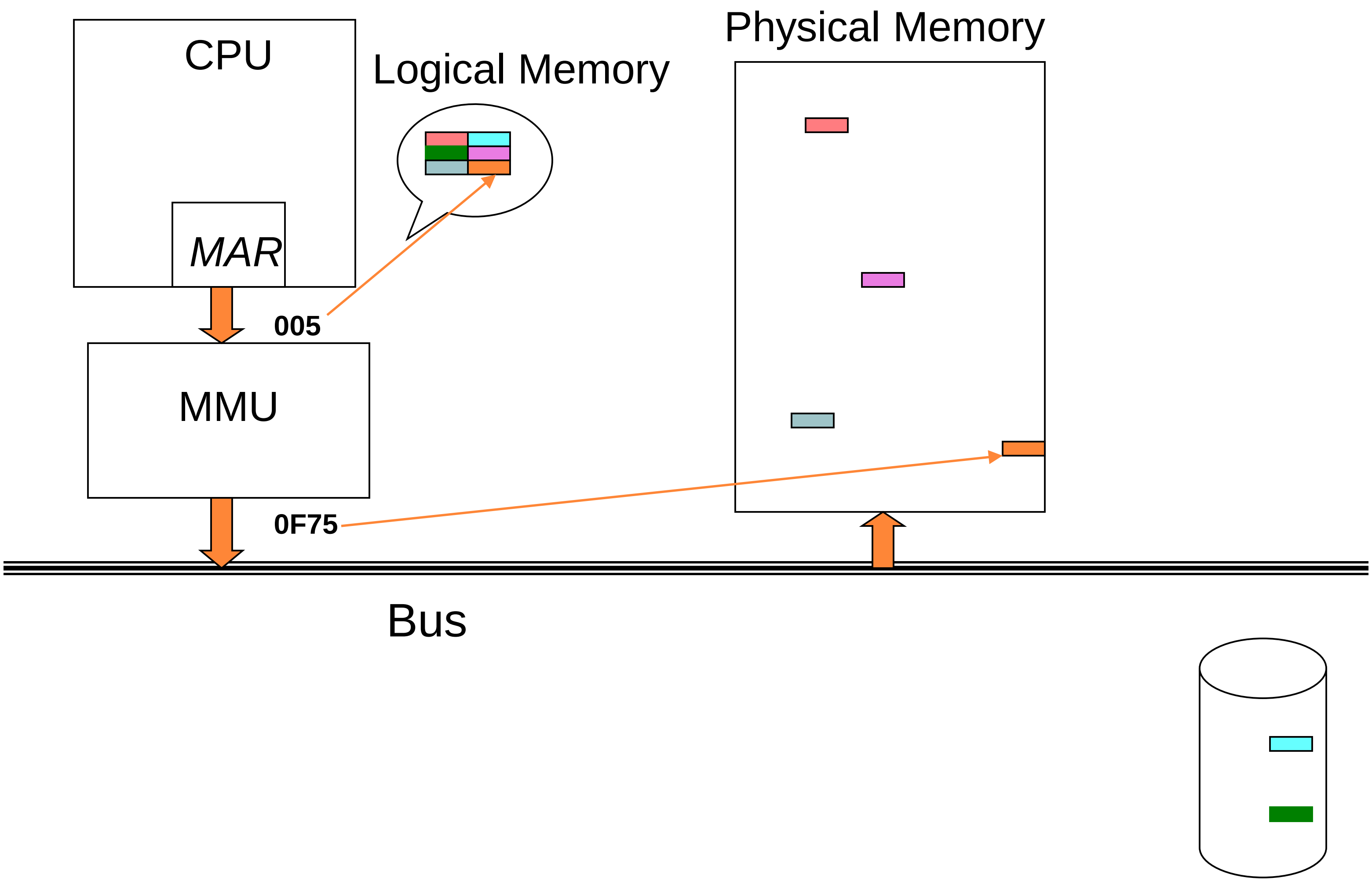
Segmentation: The MMU translation processing



Memory Reference Translation with the *Memory Management Unit (MMU)*



Virtual Memory: Contiguous and "Infinite"



Pages: Virtual Memory, Physical Memory, Disc

0	A_1
1	A_2
2	A_3
3	A_4
4	A_5
5	A_6

The Process A Virtual Memory

0	B_1
1	B_2
2	B_3
3	B_4
4	B_5
5	B_6

The Process B Virtual Memory

0	A_1
1	B_4
2	A_6
3	A_2
4	B_3
5	B_5
6	x
7	x

Free Frame List: 6,7

Physical Memory

A_1	A_2	A_3	A_4
A_5	A_6	B_1	B_2
B_3	B_4	B_5	B_6

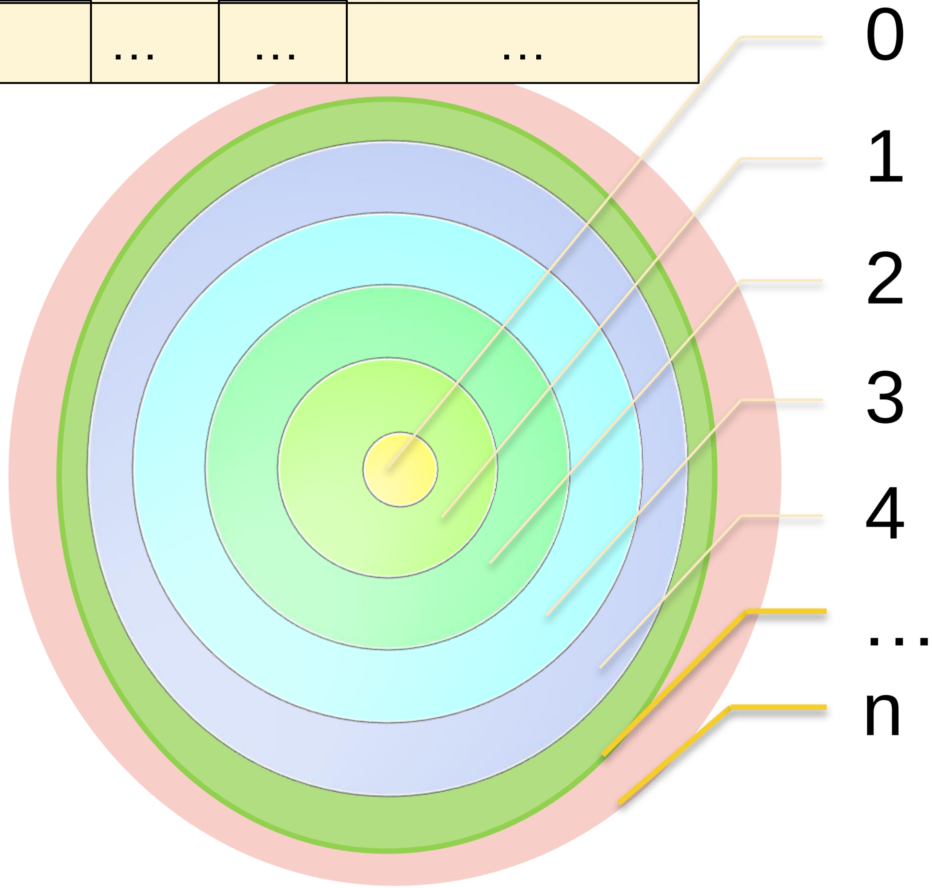
Disk

ACCESS PROTECTION RINGS

Segment Table revisited

…	…	…	…
Start address	size	**ring**	Access type
…	…	…	…

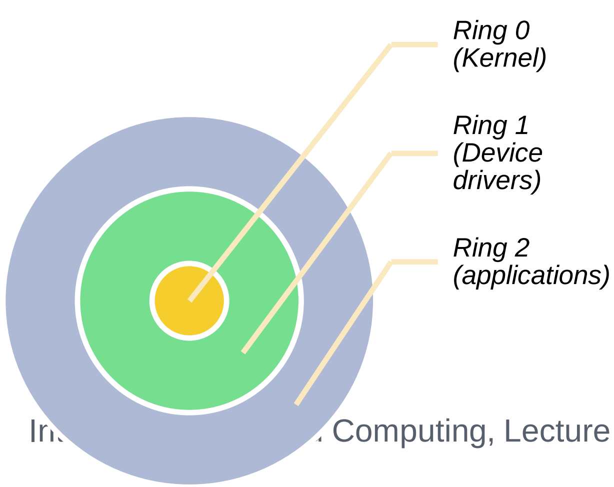
= {read, write, execute}



Code permitted to access ring *i* may access ring *j* if and only if *j* > *i*,

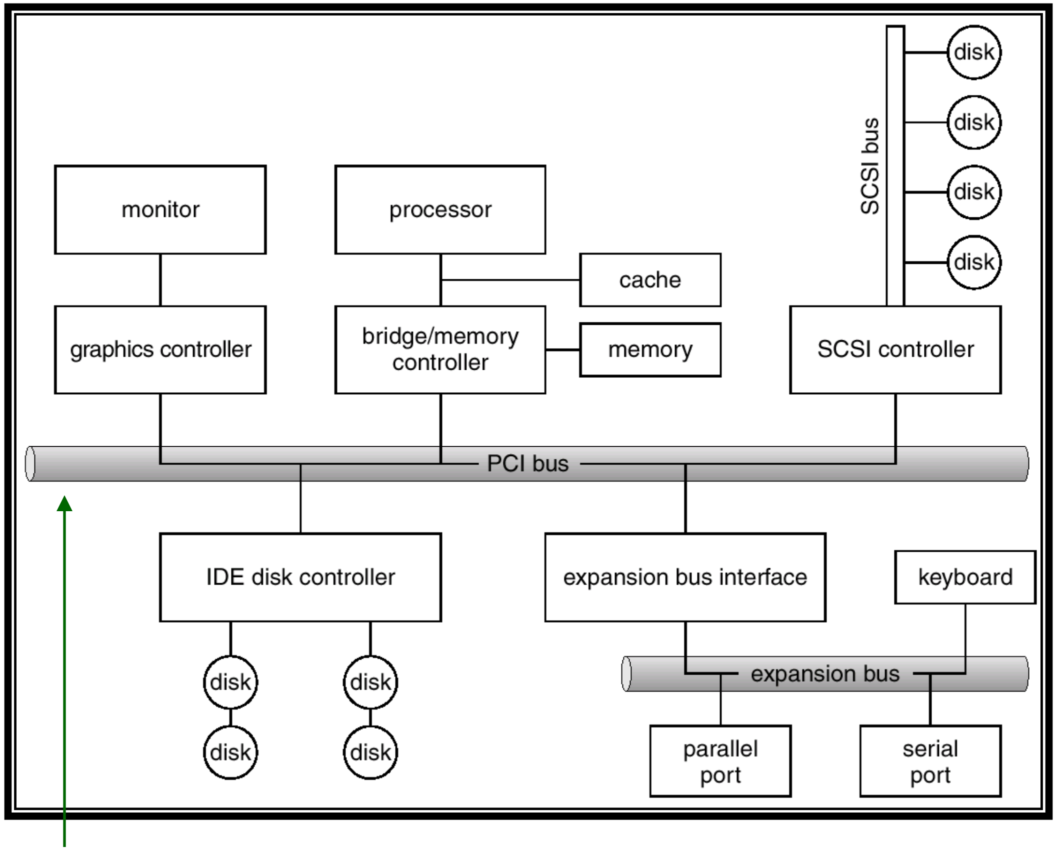
ANOTHER BIT OF HISTORY

- *Multics* had **eight** rings
- *OS/2* uses **three** rings: ring *0* for kernel code and device drivers; ring *2* for privileged code (user programs with I/O access permissions); and ring 3, for unprivileged access
- *UNIX and Microsoft Windows NT* support rings *0* (kernel) and *2 (*user programs)



Conclusion: An **OS kernel** by itself cannot create an isolated virtual machine

A PHYSICAL MACHINE



Peripheral Component Interconnection (PCI)

Note: *Integrated Disk Electronics (IDE)* is the least expensive current disk technology. IDE support is usually built into the main board.

Small Computer Systems Interface (SCSI) supports disks, tapes, and CD-ROM drives. While IDE disks provide up to one gigabyte of storage, SCSI disks are available with four to 9 gigabytes of storage.

Summary

- A process as a unit has its own virtual world of resources: A CPU, all I/O devices, an “infinite” memory
- A process relies on an operating system—the *government*, which ensures that all processes are treated fairly
- The operating system *kernel* is the only entity that can execute privileged instructions; the kernel is entered via *traps* (internally) and *interrupts* caused by other events
- A process is aware of other processes
- A *machine* (Hardware + OS) executes many processes
- <u>Question</u>: How can we *create* a virtual machine?