CMPT 300
Operating System I

3.2 –Process 2
Chapter 3

Dr. Hazra Imran

Admin notes

- Quiz 1 grades released ✓
- No class on June 3 (next Friday)

Multiprogramming

P1 P2 P3

OS requirements for multiprogramming

1. Mechanism
   - To switch between processes — Process Dispatcher
   - To protect processes from one another

2. Policy
   - To decide which process to schedule — Process Scheduler

Decision maker

Optimize workload performance metrics

Dispatch Mechanism

Context switch
Interupts

OS runs **dispatch loop**

```
while (1) {
        run process A for some time-slice
        stop process A and save its context
        load context of another process B
    }
```

Context-switch

Dispatcher must track context of process when not running

- Save context in **process control block (PCB)** (or, process descriptor)

OS control

Pr A P1 P2 P3

How does OS get control?

1) Synchronous interrupts, or traps
   - Event internal to a process that gives control to OS
   - Examples: System calls, page faults (access page not in main memory), or errors (illegal instruction or divide by zero)

2) Asynchronous interrupts
   - Events external to a process, generated by hardware
   - Examples: Characters typed, or completion of a disk transfer

Interrupt ⟷ ISR

How does Dispatcher run?

Option 1: Cooperative Multi-tasking

- Trust process to hand over CPU through traps
  - Disadvantages: Processes can misbehave
  - By avoiding all traps and performing no I/O, can take over entire machine
  - Only solution: Reboot!
- Not performed in modern operating systems

Option 2: True Multi-tasking

- Guarantee OS can obtain control periodically
- Enter OS by enabling periodic alarm clock
- Hardware generates timer interrupt (CPU or separate chip). Example: Every 10ms User must not be able to mask timer interrupt
- Dispatcher counts interrupts between context switches. Example: Waiting 20 timer ticks gives 200 ms time slice
- Common time slices range from 10 ms to 200 ms

Context Switching

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch

- **Context** of a process represented in the PCB

5 process

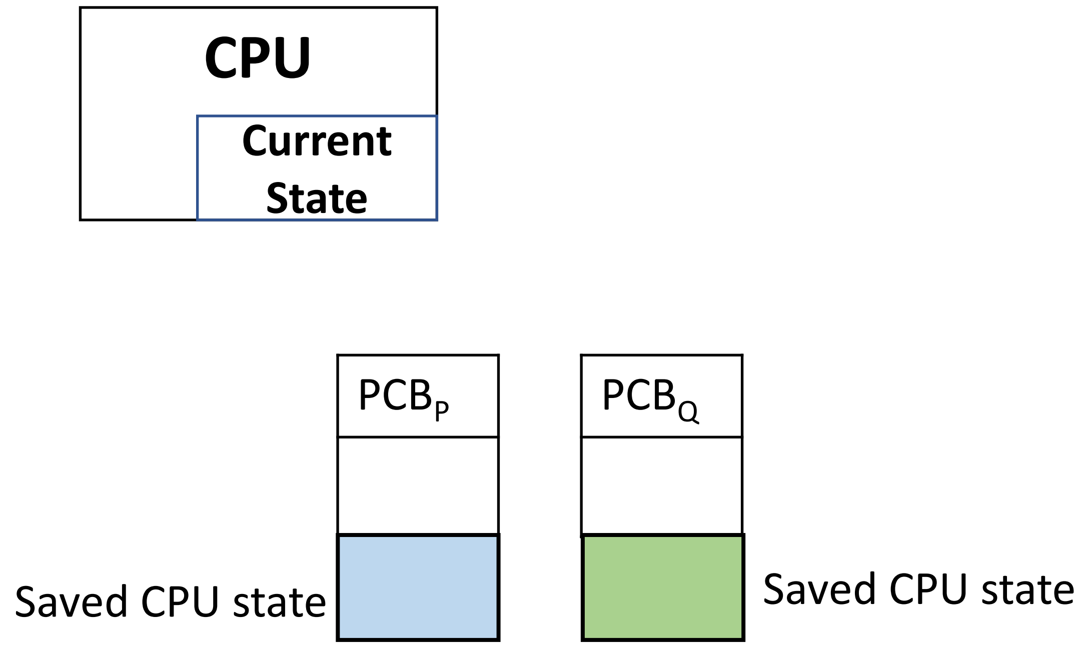
running - one process at a time

Multiple process single CPU (turn)

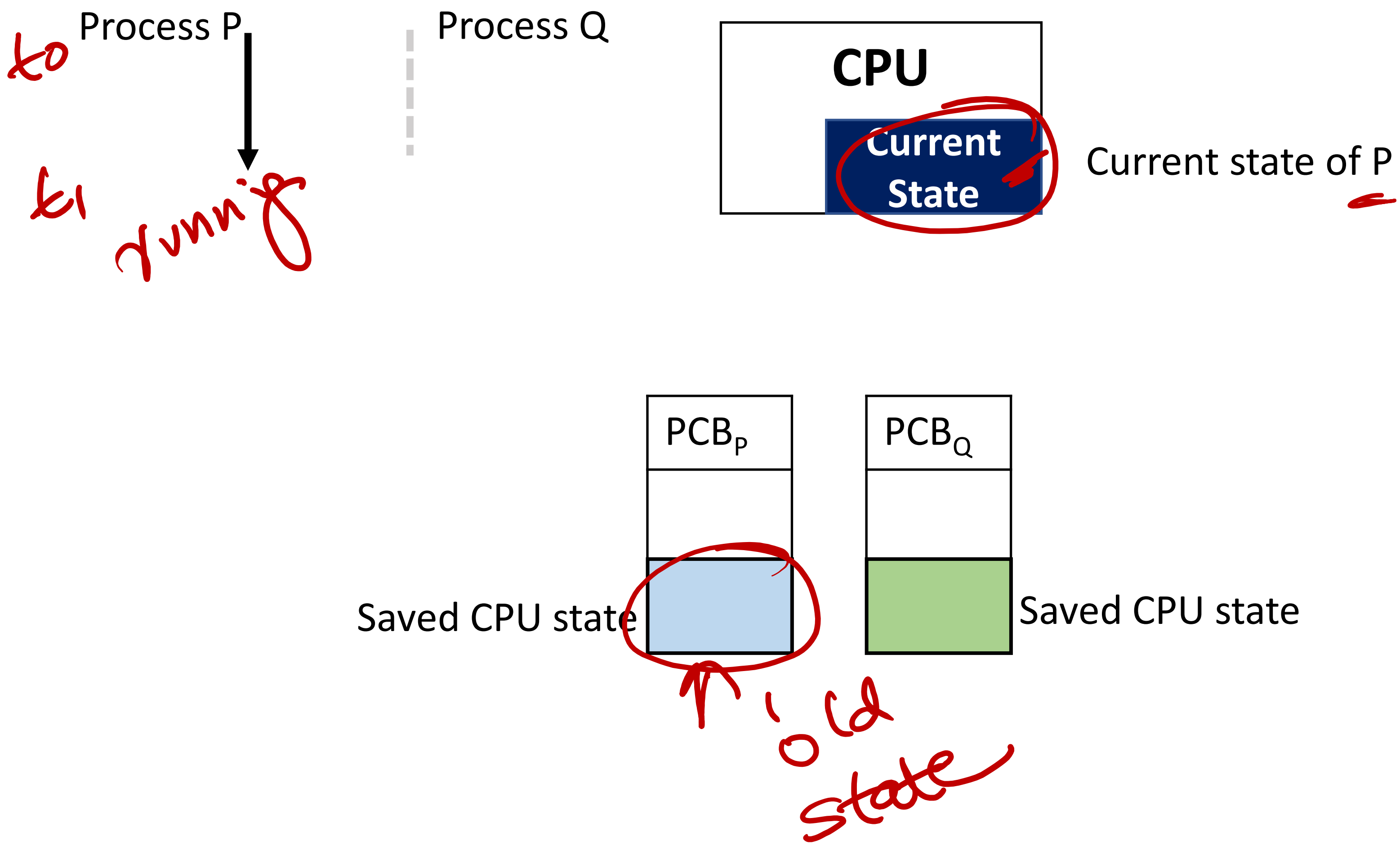
Context Switching

Process P

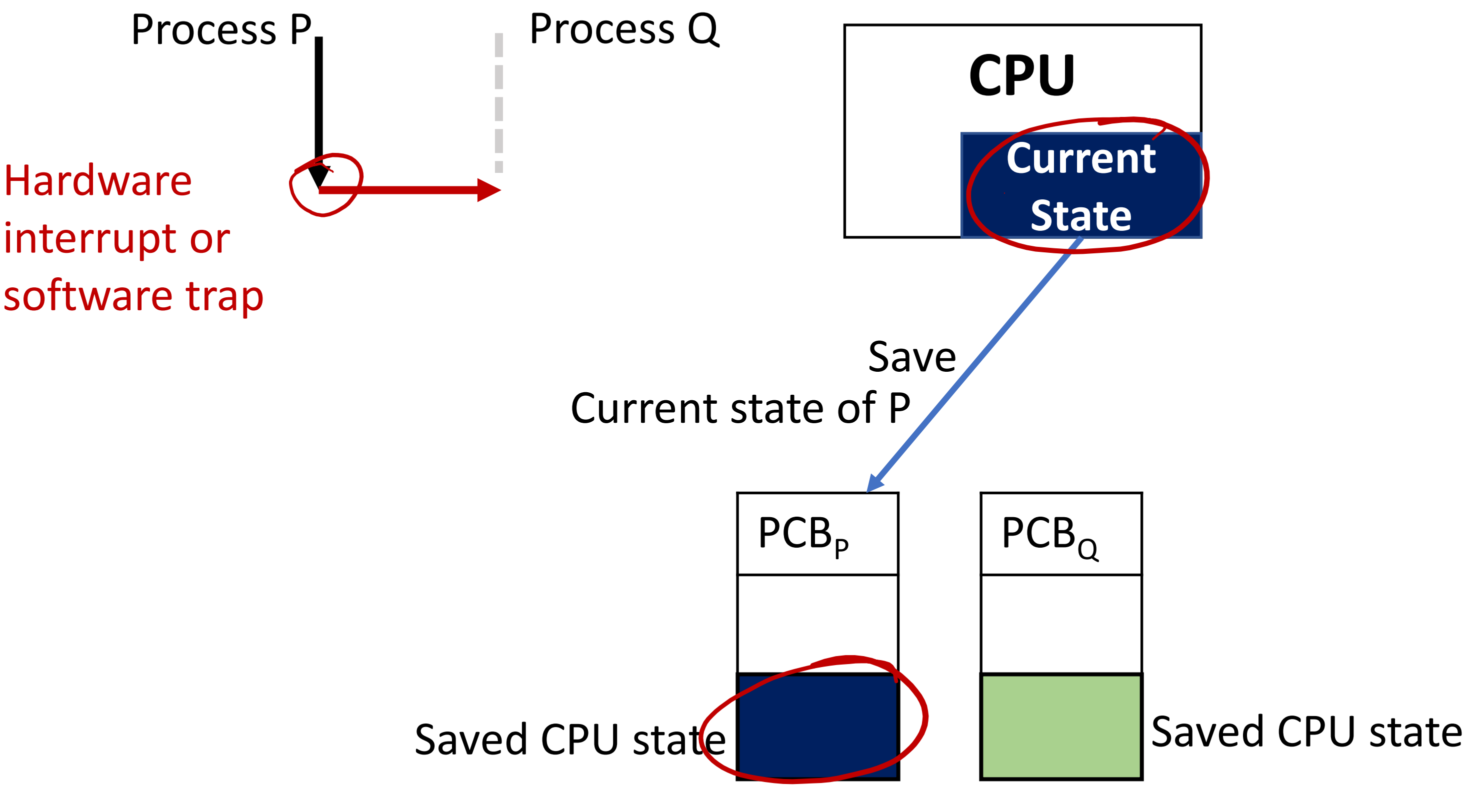
Process Q



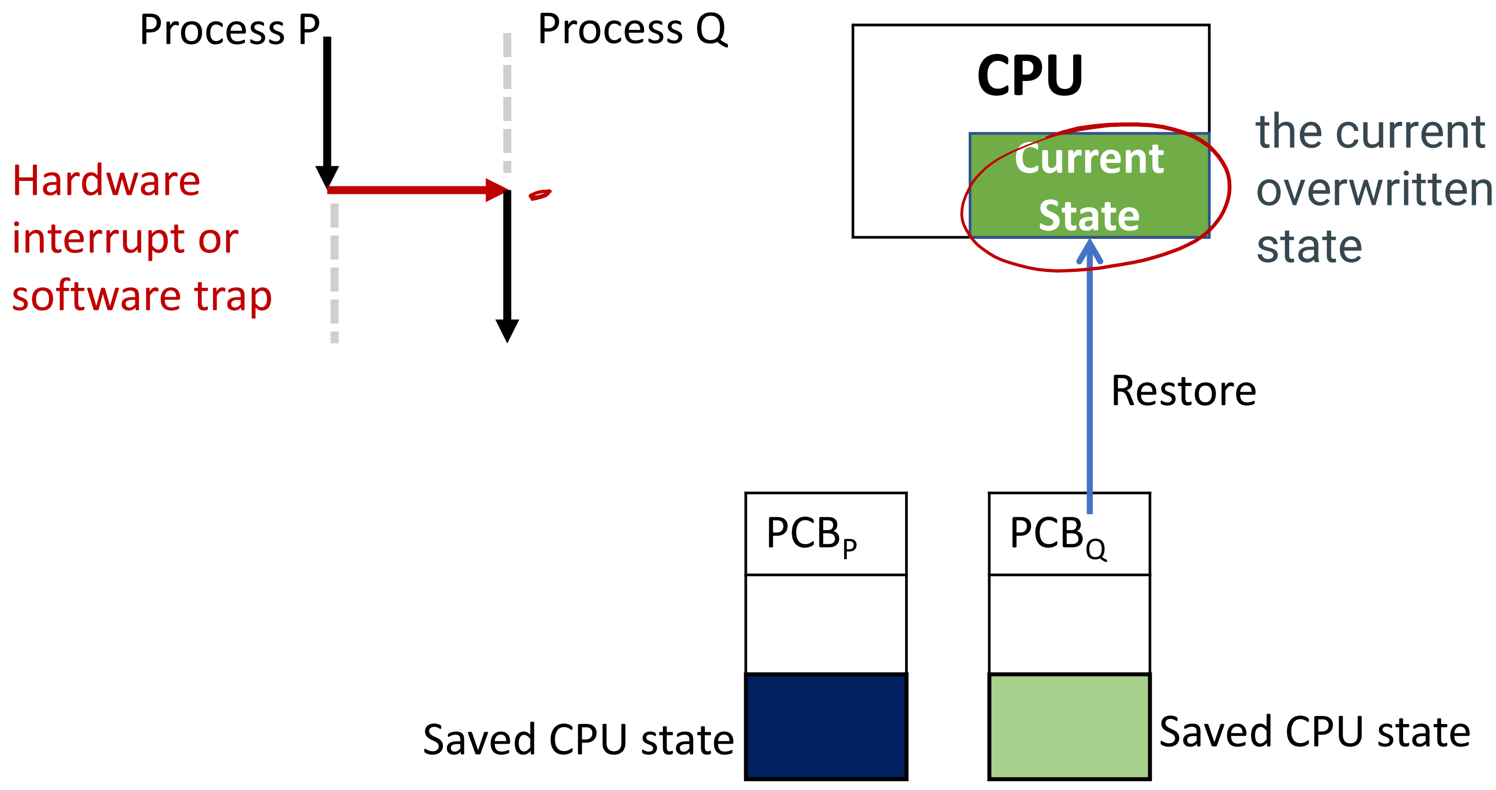
Context Switching



Context Switching

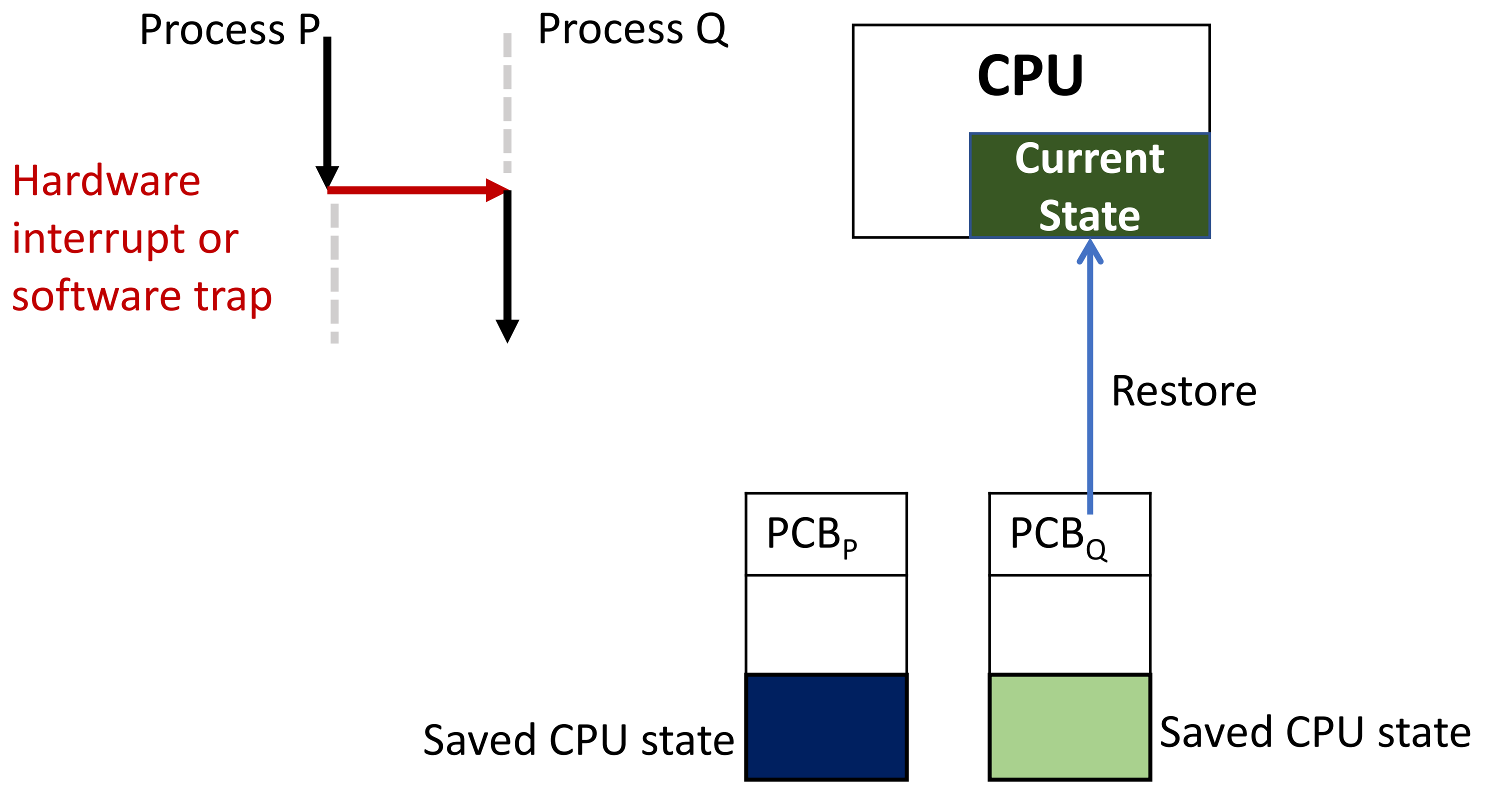


Context Switching

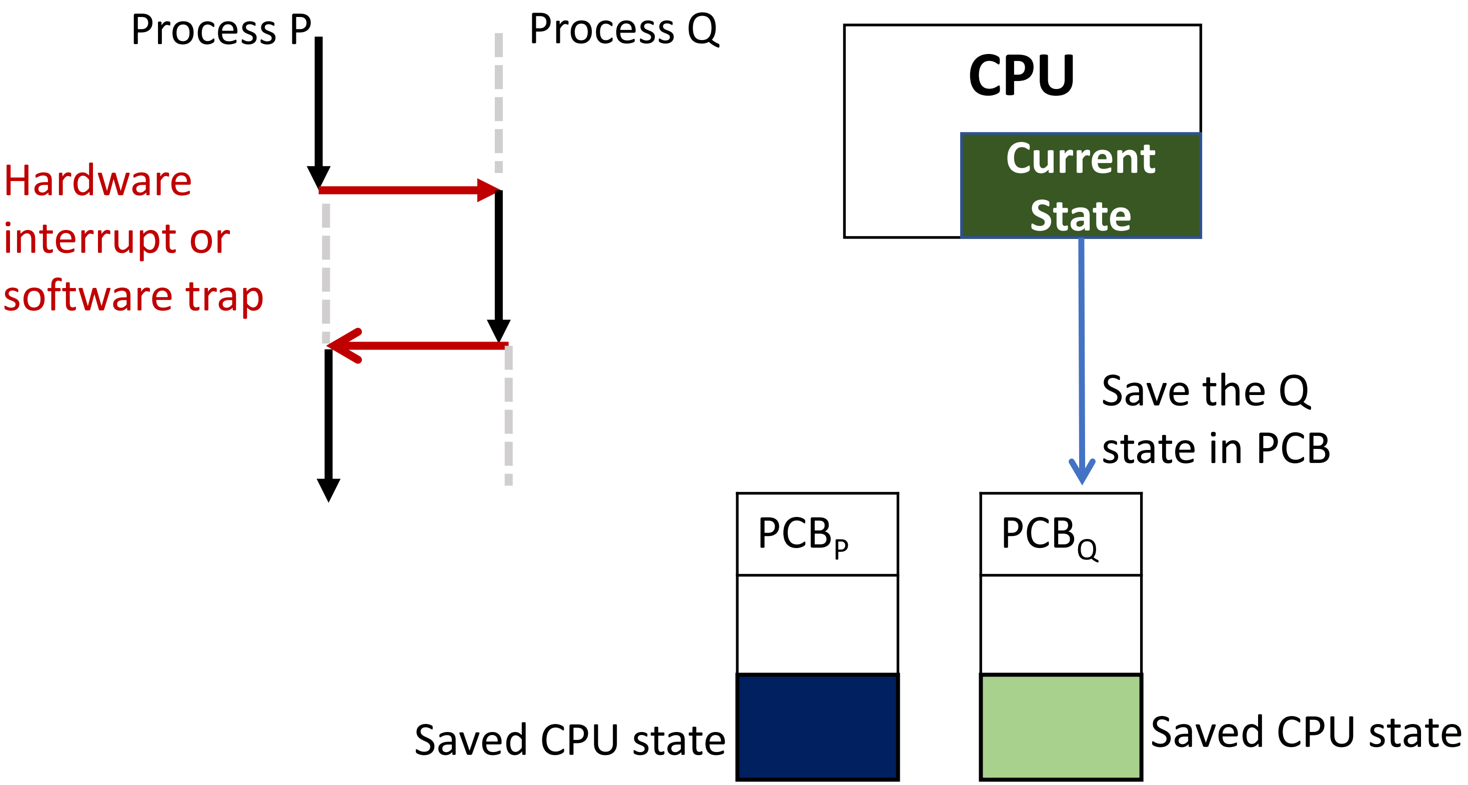


the current CPU state is overwritten with Q's last state

Context Switching



Context Switching



Context Switching

1 ms

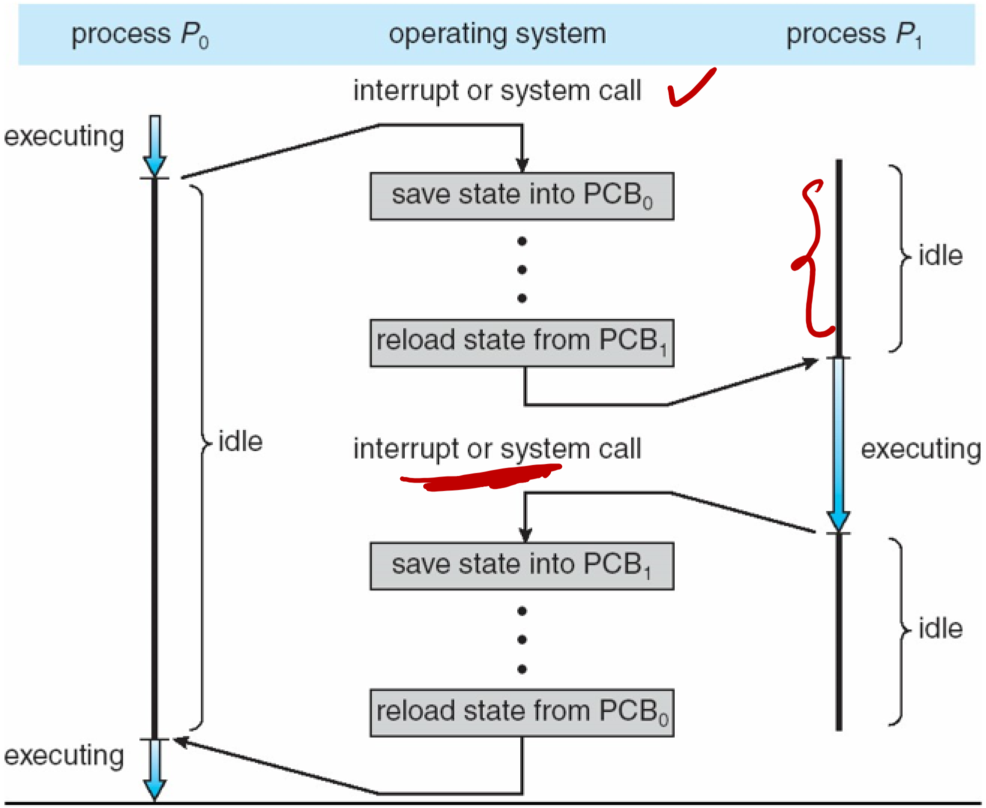
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB, the longer the context switch
  - Time is dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU
    - Allows multiple contexts to be loaded at once

CPU Switching from One Process to Another (Context Switch)

When switching occurs, kernel

- Saves **state** of P_0 in PCB_0 (in memory)
- Loads **state** of P_1 from PCB_1 i registers

State = values of the CPU registers, including the program counter, stack pointer



multiple register
↓
fast switching

Clicker

- A context switch for a currently running process P may be caused by ______

system call, P - unavailable resource

(A) Process P itself ✓

(B) OS ✓

(C) Some other process Q ✗

timer, OS let other Q to execute

Q cannot cause a context switch for P

Why the concept of Processes?

Structuring an application as processes allows independence from the:

- Number of CPUs
  - Physical CPU
  - Virtual CPU
- Type of CPU

Program

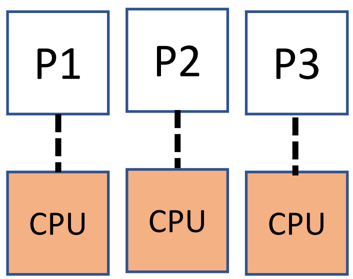
PA

PB

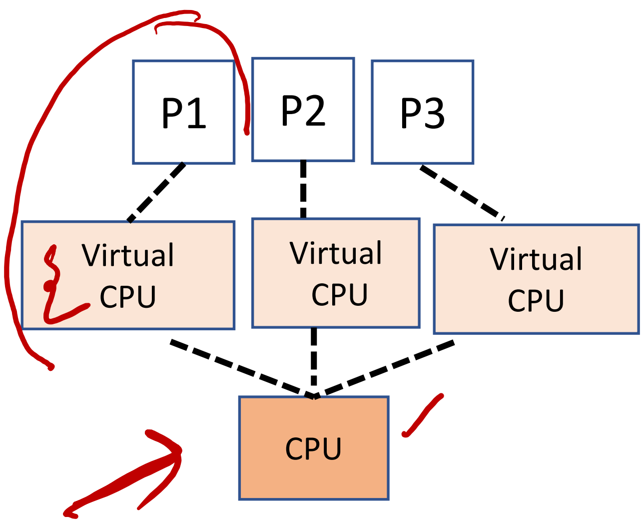
PC

Why the concept of Processes?

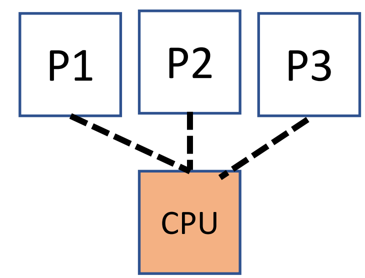
Different set up



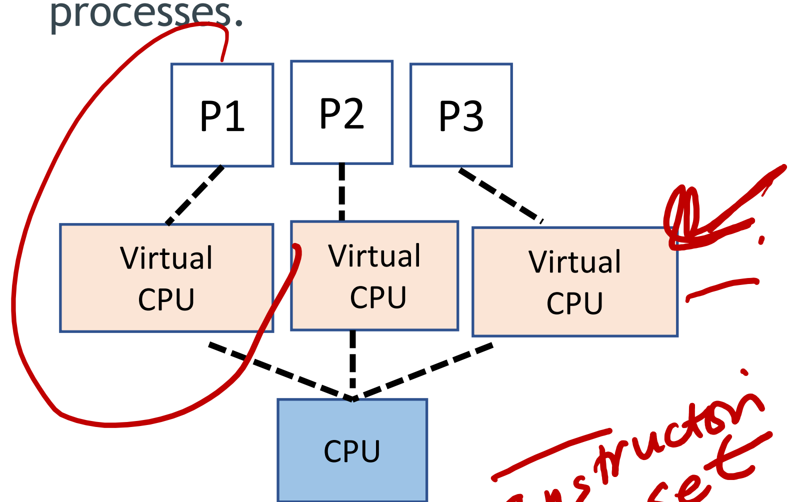
Each process can have a separate physical CPU.



time slice



All processes can time-share the same CPU, which is repeatedly switched among the processes.



Instruction set

Why Virtual CPUs?

Independence from the number and type of CPUs provides several crucial benefits:

- **Multi-user support**

- Multiple users, each represented by one or more separate processes, can share the same machine without being aware of each other.

Why Virtual CPUs?

Independence from the number and type of CPUs provides several crucial benefits:

- **Multi-CPU transparency**

  - **Multi-CPU transparency**: An application written to utilize multiple CPUs will run correctly, although perhaps more slowly, if only one CPU is available.

Why Virtual CPUs?

Independence from the number and type of CPUs provides several crucial benefits:

- **Portability**

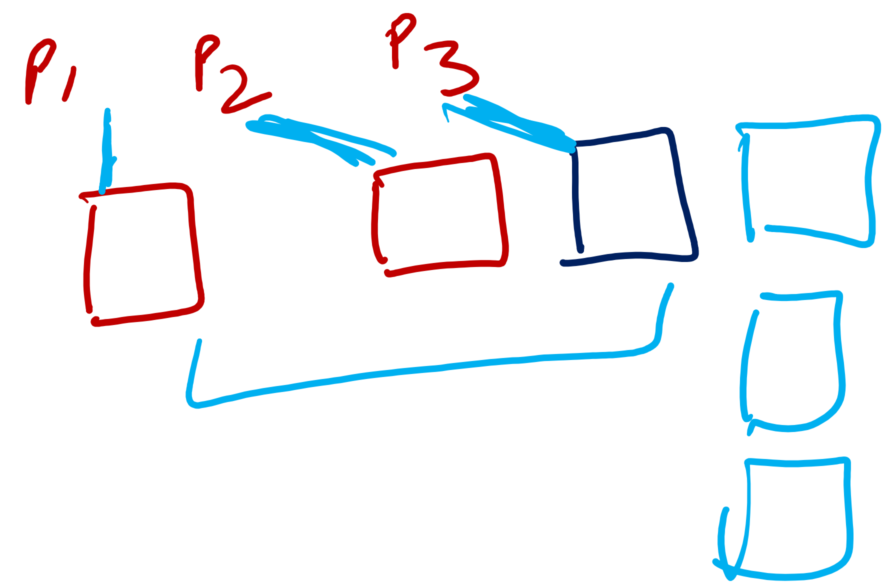
  An application compiled for one type of CPU can run on a different CPU without being modified or even recompiled.

Clicker

We have 3 independent processes executing on 2 physical CPUs.

If we increase the number of physical CPUs from 2 to 3 and then to 4, then the speed of execution will ____________?

A. An increase
B. Decrease
C. No effect



Kernel Data Structure: Process Control Block

- Process-specific details are managed in a data structure called the **process control block** (PCB), that is created and managed by the OS.

Kernel Data Structure: Process Control Block

- Process-specific details are managed in a data structure called the **process control block** (PCB), that is created and managed by the OS.

- PCB contains sufficient information so that it is possible to interrupt a running process and later resume execution as if the interruption had not occurred.

Kernel Data Structure: Process Control Block

- Process-specific details are managed in a data structure called the **process control block** (PCB), that is created and managed by the OS.

- PCB contains sufficient information so that it is possible to interrupt a running process and later resume execution as if the interruption had not occurred.

- The process control block is the key tool that enables the OS to support multiple processes and to provide for multiprocessing.

Kernel Data Structure: Process Control Block

- Process-specific details are managed in a data structure called the **process control block** (PCB), that is created and managed by the OS.
- PCB contains sufficient information so that it is possible to interrupt a running process and later resume execution as if the interruption had not occurred.
- The process control block is the key tool that enables the OS to support multiple processes and to provide for multiprocessing.
- What happen when process is interrupted?

Context switching.

Kernel Data Structure: Process Control Block

1. CPU_state

current p's state

2. Process_state

area of memory assigned to P

3. Memory

4. Scheduling information

when to start P?
time slice.

Kernel Data Structure: Process Control Block

5. Accounting information

billing

6. Open_files

currently open files by 'p'

7. other_resources

printer, p → request

8. Parent

(P) Parent process

9. Child

Child process C1

C2

C3

Kernel Data Structure: Process Control Block

An empty PCB structure is created for a new process.

CPU_state	Set of integers
Process_state	Integer or char
memory	Pointers
Scheduling information	Set of integers
Accounting information	Set of integers
Open_files	Start of linked list
Other_resources	Start of linked list
parent	Pointer or index
children	Start of linked list

table

Clicker

What is the minimum number of bits needed to represent the process_state field, if three states are supported: running, ready, and waiting?

2 bit - 4 state

A) 1 bit
B) 2 bit
C) 3 bit

N C