Department of Physics,
Computer Science & Engineering

CPSC 410 – Operating Systems I

Process Description & Control

Keith Perkins

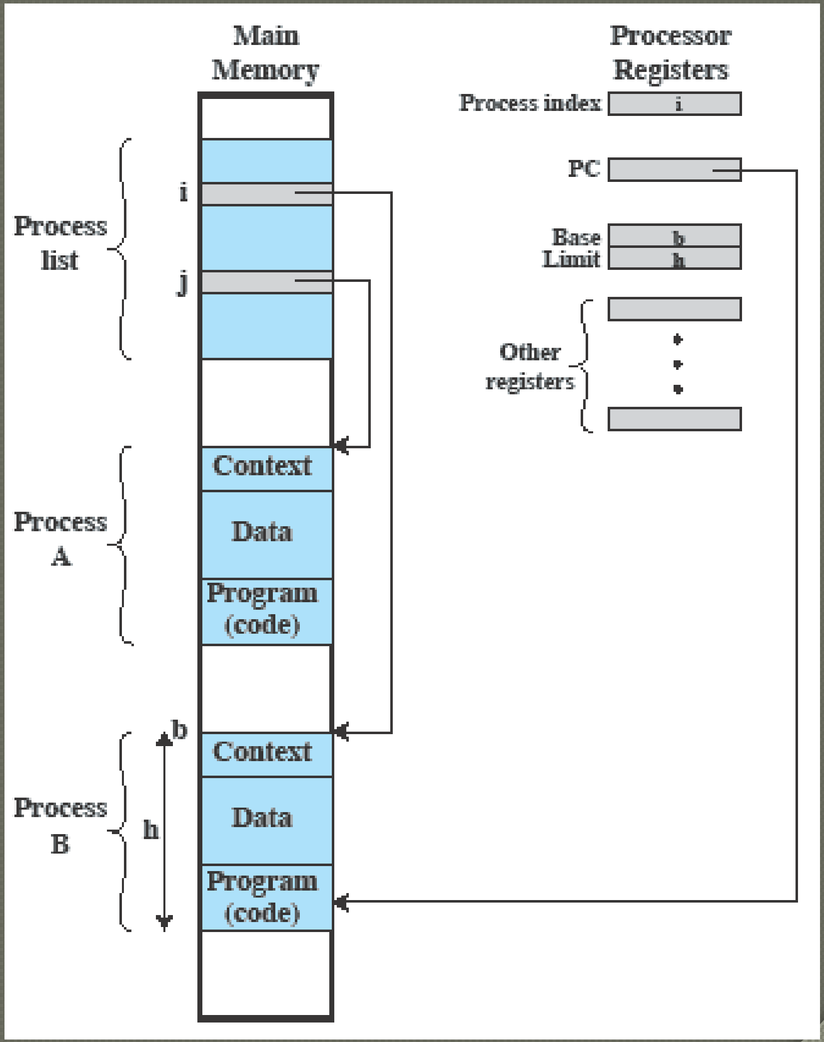
Adapted from original slides by Dr. Roberto A. Flores
Also from “CS 537 Introduction to Operating Systems” Arpaci-Dusseau

Topics

- Everything about Processes
  - Control blocks
  - States
  - Description
  - Control

Revisit - Process Management

- Scheduler chooses a process to run (more later)
- Dispatcher runs it
- How? What's in the Process List?
- BTW this list is a simplification

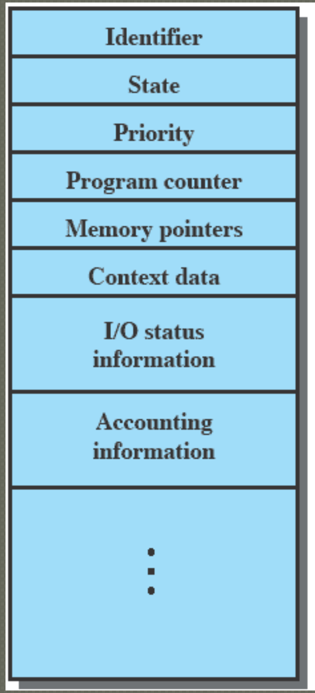


Control blocks
States
Description
Control

Processes

Control Blocks

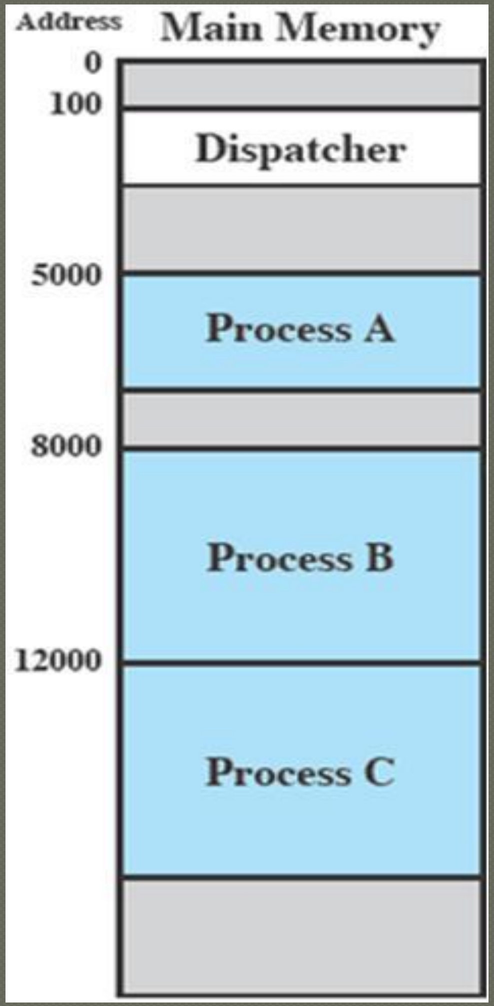
- data structure created & managed by OS
  - Identifier: unique ID
  - State: (e.g., running, blocked)
  - Priority: relative to other processes
  - Program counter: address of next instruction
  - Memory pointers: to code & data
  - I/O status: I/O in use/pending
  - Accounting: CPU time used, IDs, …
- data to hold/restore process state on interrupt/resume
  - key to support multiprocessing



Control blocks
States
Description
Control

Processes

- Dispatcher
  - Program that switches processes in/out of the CPU



Process dispatching mechanism

OS dispatching loop:

```
while(1) {
    run process for a while;
    save process state;
    next process = schedule (ready processes);
    load next process state;
}
```

Q1: how to gain control?

Q3: where to find processes?

Q2: what state must be saved?

Processes

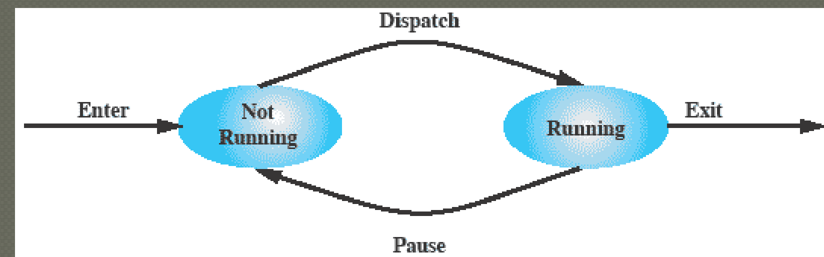
- **States**
  - Trace
    - Instructions executed by a process
    - In multiprogramming:
      - interleaving of instructions as processes alternate using the CPU
  - The pale blue lower right is dispatcher code
  - Process switches because of Interrupts (timer, I/O)

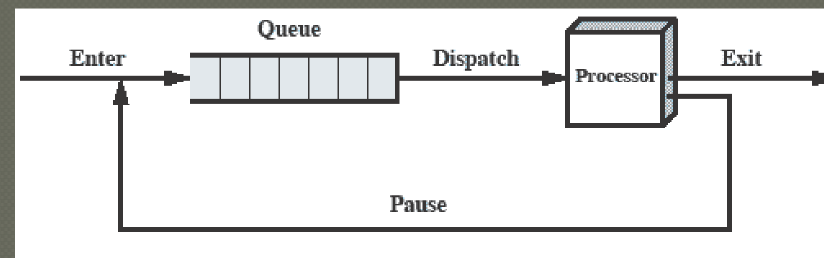
(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

1	5000
2	5001
3	5002
4	5003
5	5004
6	5005
------------------- Timeout	
7	100
8	101
9	102
10	103
11	104
12	105
13	8000
14	8001
15	8002
16	8003
--------------- I/O Request	
17	100
18	101
19	102
20	103
21	104
22	105
23	12000
24	12001
25	12002
26	12003
27	12004
28	12005
------------------- Timeout	
29	100
30	101
31	102
32	103
33	104
34	105
35	5006
36	5007
37	5008
38	5009
39	5010
40	5011
------------------- Timeout	
41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	12011
------------------- Timeout	

Processes

- States (2 states)
  - One CPU
  - Round-robin (timeout)
  - Running: CPU time!
  - Not running: or not

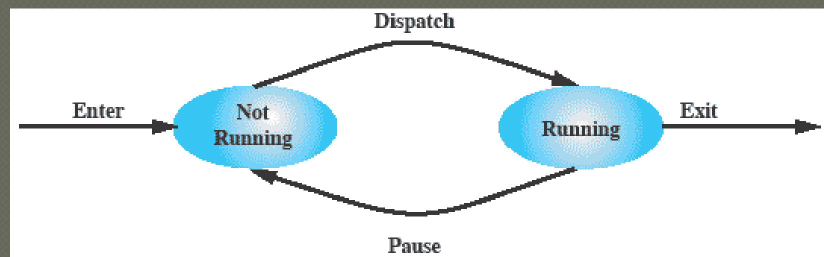


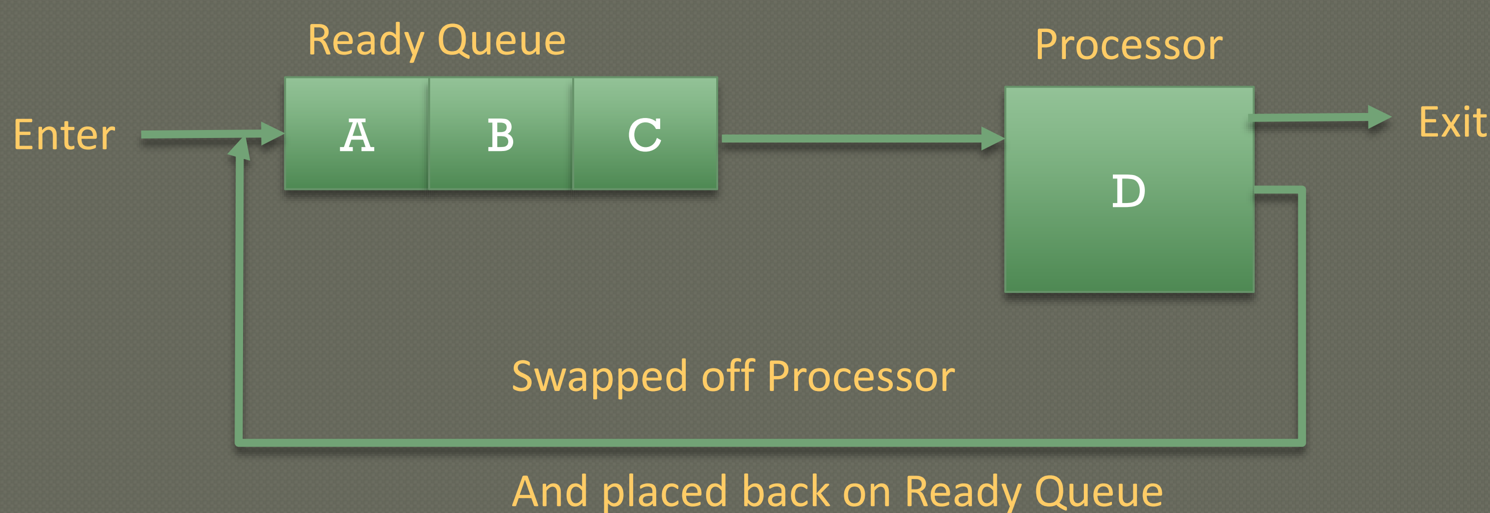


(b) Queuing diagram

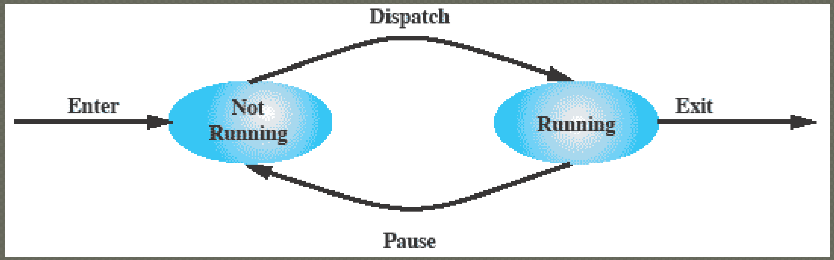
- Where do processes come from?
- When do they stop?

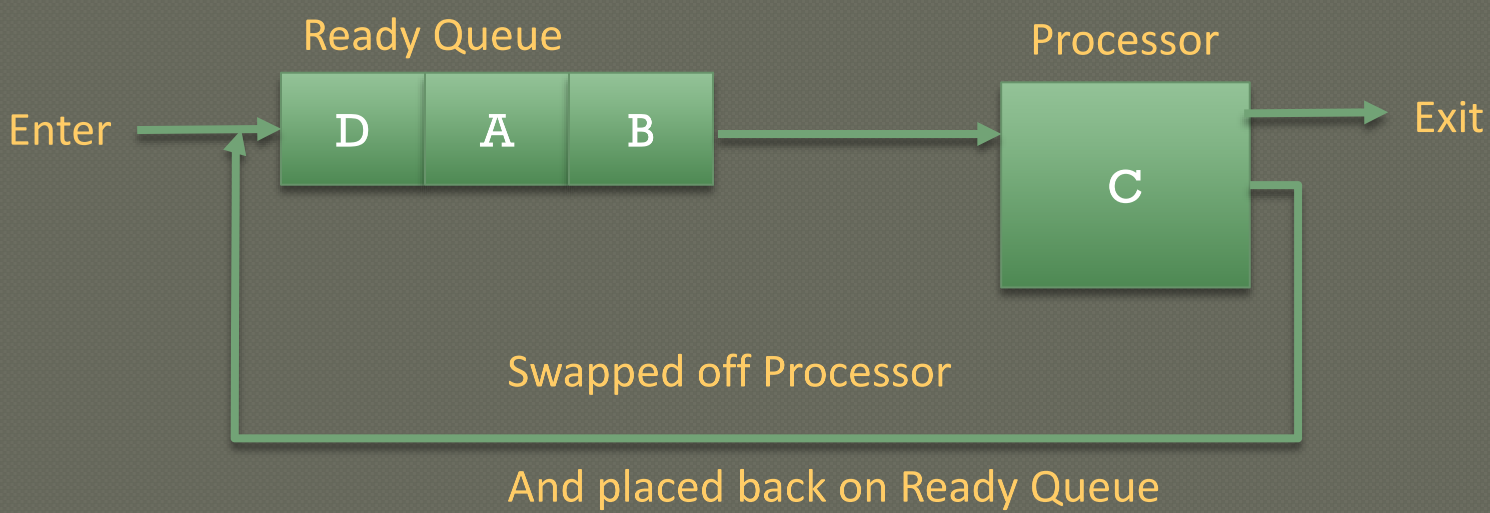
Swapping processes



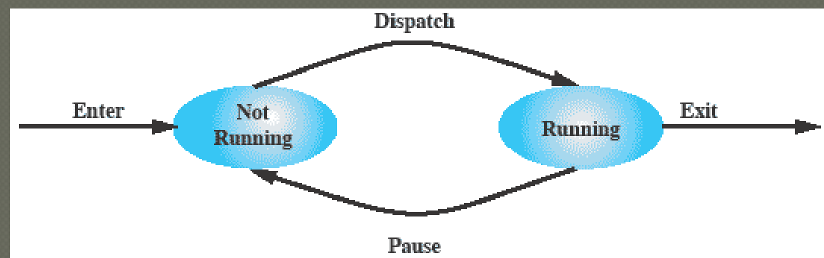


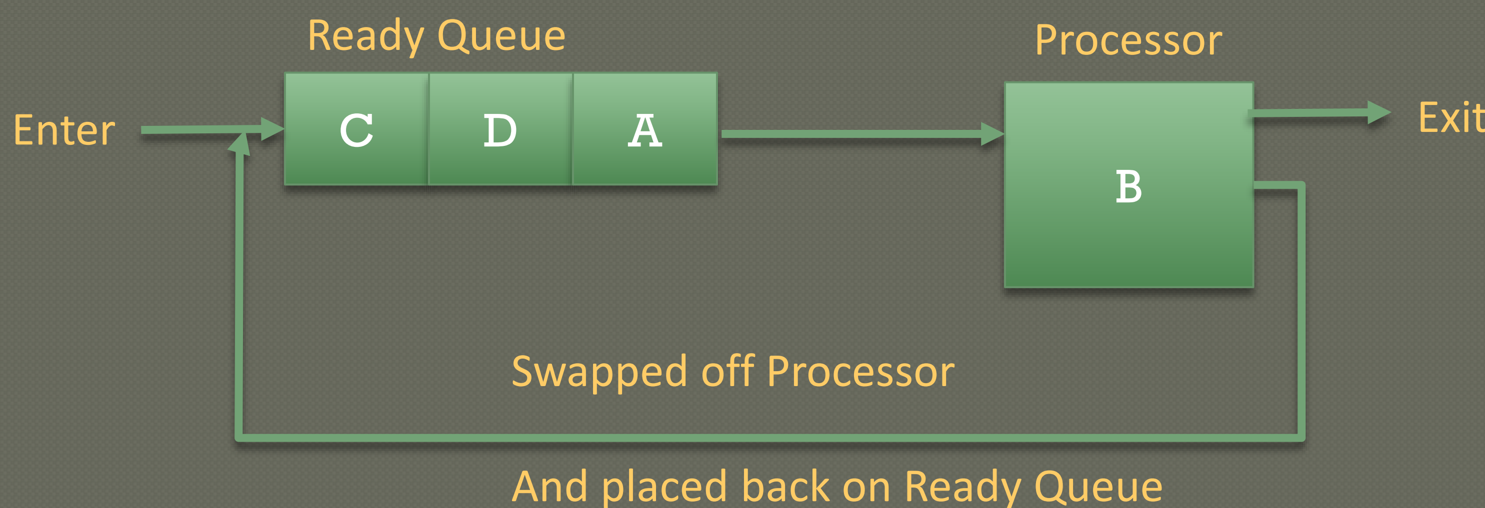
Swapping processes



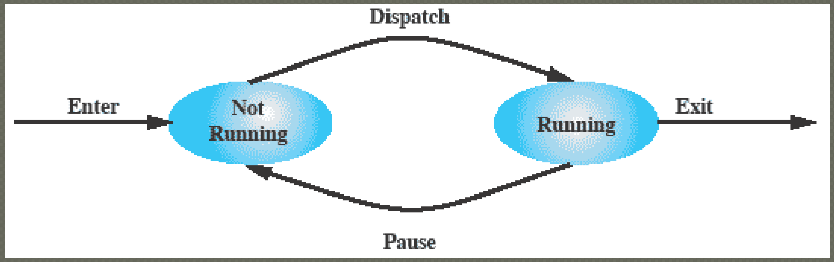


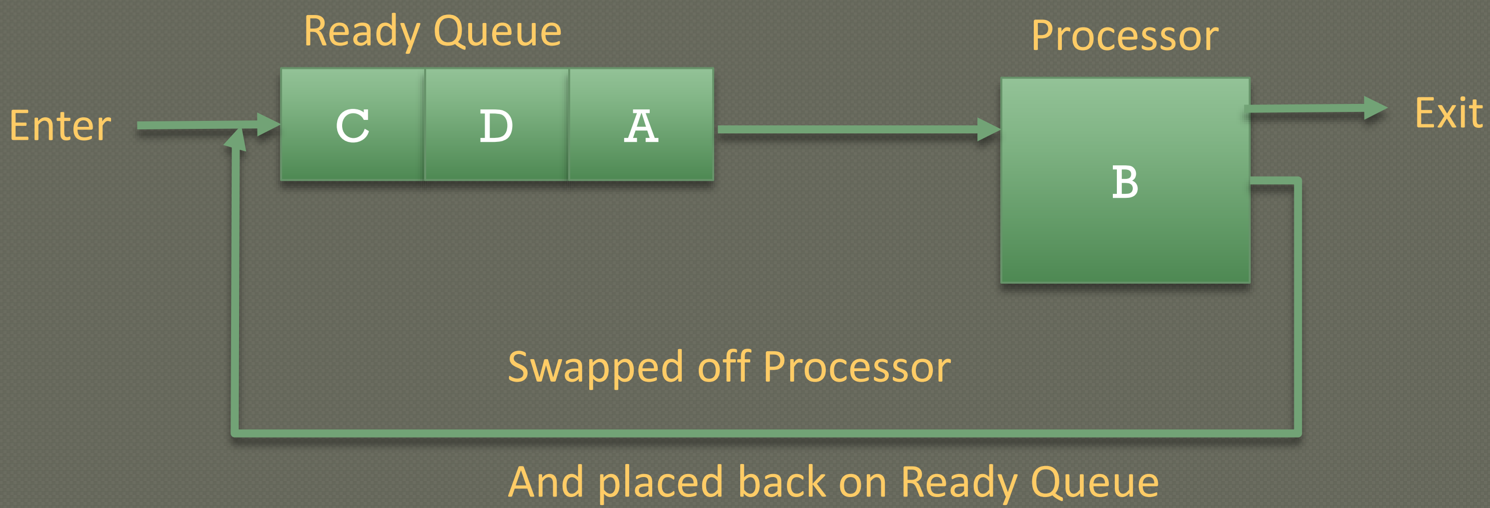
Swapping processes





Swapping processes





This cycle continues with some processes finishing and new processes taking their place in the Ready Queue

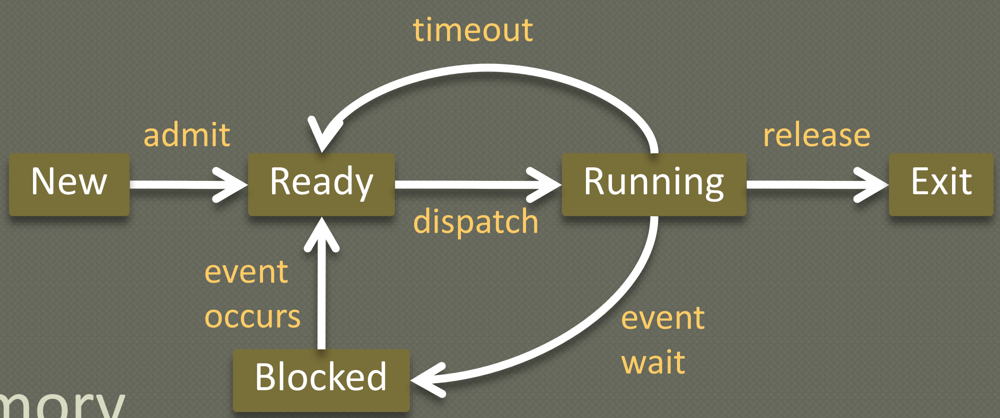
- Control blocks
- States
- Description
- Control

Processes

- Where do processes come from? (start)
  - Interactive logon: User in terminal logs in
  - OS service: OS-provided service (e.g., print spooler)
  - Spawned by process: uses parallelism (parent spawns child)
- When do they end? (termination)
  - Normal
    - Job finishes, user logs off, OS shutting down, etc.
  - Abnormal
    - Timeout: timeslice
    - Resource error: out of memory, I/O device unresponsive, deadlock
    - Runtime error: arithmetic operation, uninitialized variable
    - Authorization error: memory out of bounds, resource/instruction privilege

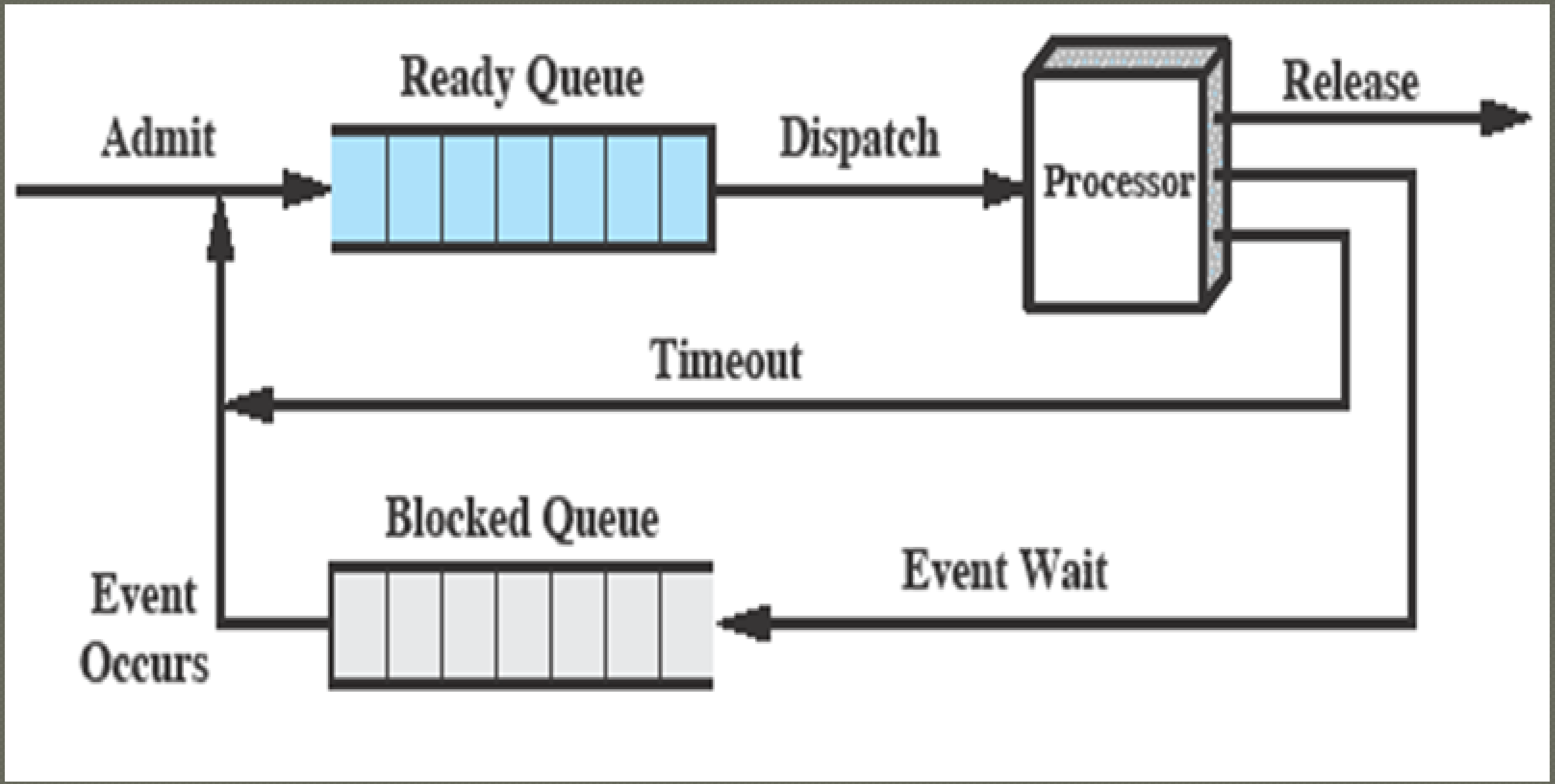
Processes

States (5 states)



- New: not yet in memory
- Ready: awaiting its turn
- Running: CPU time!
- Blocked: waiting for I/O
- Exit: done & gone

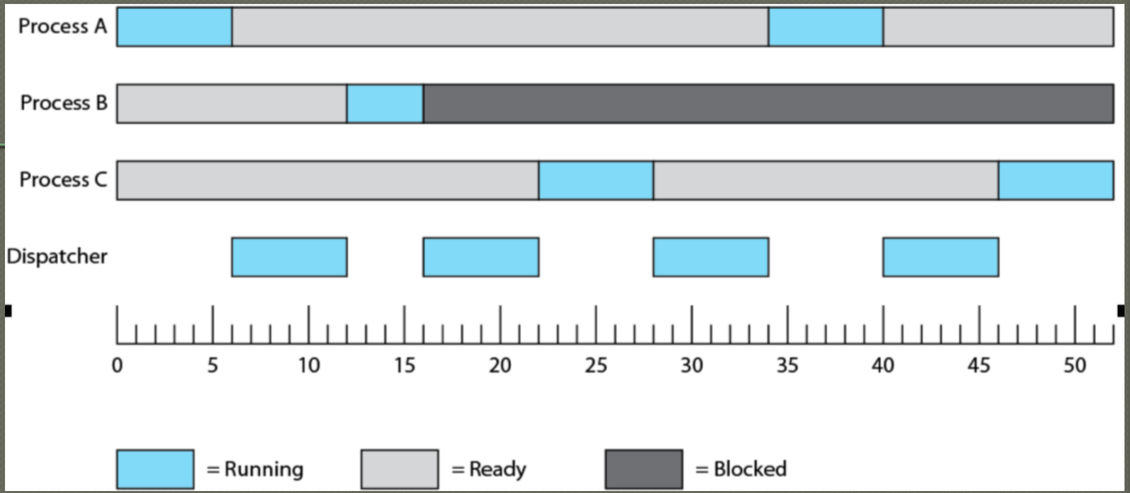
Using Two Queues



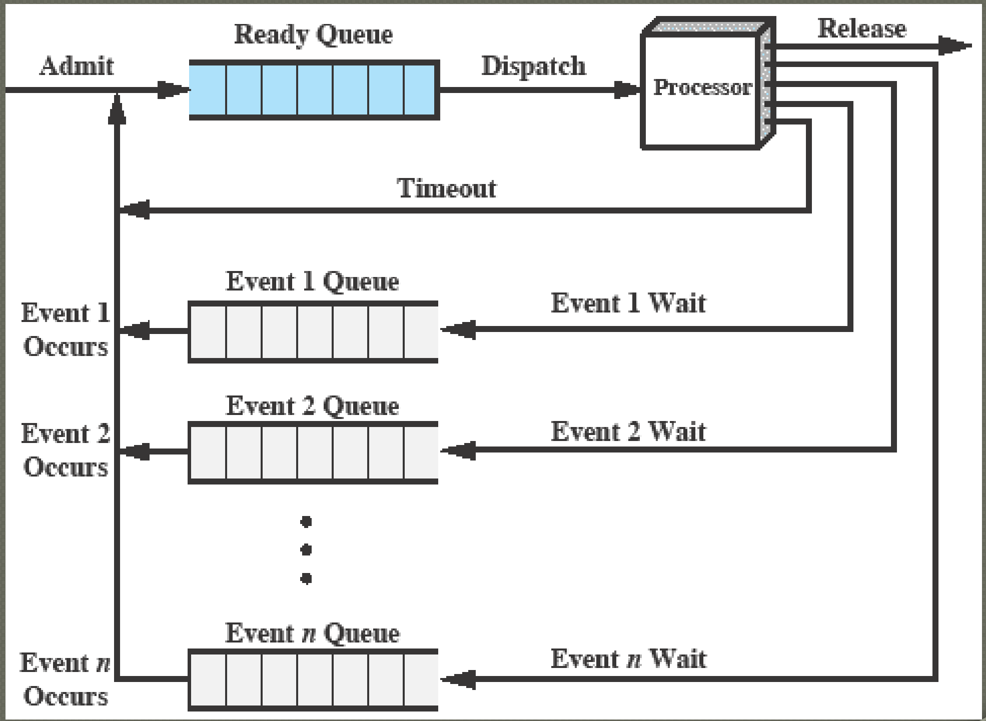
Control blocks
States
Description
Control

States (5 states)

- e.g., Processes A, B & C

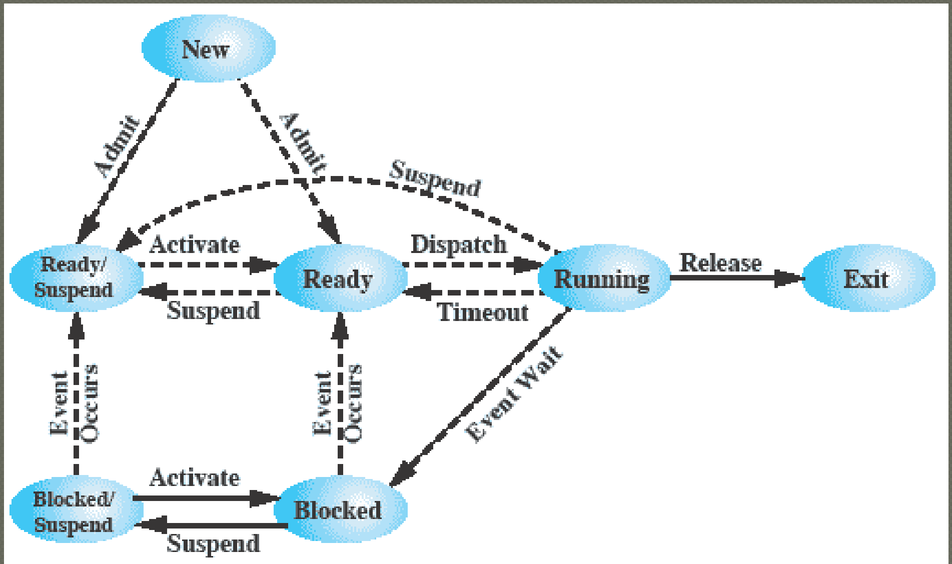


- Multiple block queues (1 per I/O device)



Processes

- States (7 states)
  - What if running I/O intensive processes and all are waiting for I/O?
    - Solution: suspend blocked processes to disk, bring in new (from new or ready/suspend)



Processes

- Process tables
  - OS keeps list of processes. Each entry tracks data about each process (process image)
    - Heap:
    - Globals:
    - Code: program to execute
    - stack: method call stack frames
    - process control block (PCB): data OS uses to control process
      - process identification: process/parent/user ID
      - processor state information: user/control registers, stack pointers
      - process control information: scheduling, inter-process comms, …
  - reference (directly/indirectly) memory, I/O & file tables

Processes

Process tables

Process identification

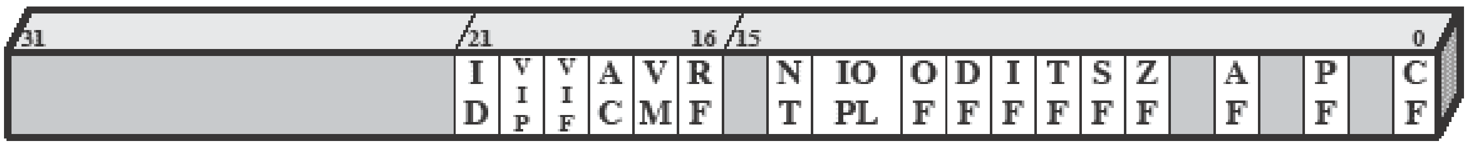
- Each process has a unique ID
- IDs are used for reference:
  - in other tables
  - in inter-process communication
  - when a parent spawns a child process

- process identification: process/parent/user ID
- processor state information: user/control registers, stack pointers
- process control information: scheduling, inter-process comms, …

- reference to memory, I/O & file tables

Process state information

- stack pointers
- user-visible registers
- control & status registers
  - program status word (PSW), e.g., EFLAGS in x86 processors



ID = Identification flag
VIP = Virtual interrupt pending
VIF = Virtual interrupt flag
AC = Alignment check
VM = Virtual 8086 mode
RF = Resume flag
NT = Nested task flag
IOPL = I/O privilege level
OF = Overflow flag

DF = Direction flag
IF = Interrupt enable flag
TF = Trap flag
SF = Sign flag
ZF = Zero flag
AF = Auxiliary carry flag
PF = Parity flag
CF = Carry flag

- processor state information: user/control registers, stack pointers
- process control information: scheduling, inter-process comms, …

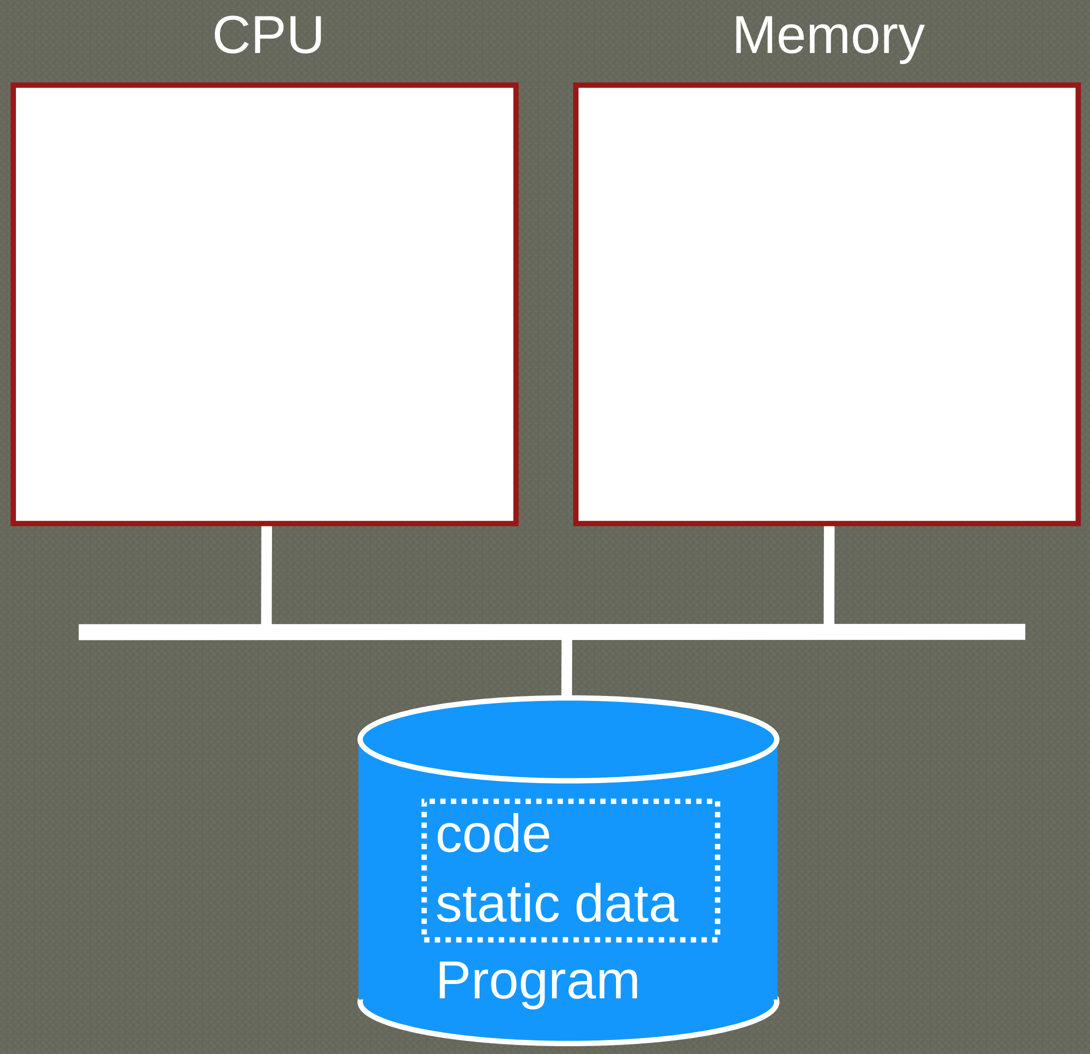
- reference (directly/indirectly) memory, I/O & file tables

Processes

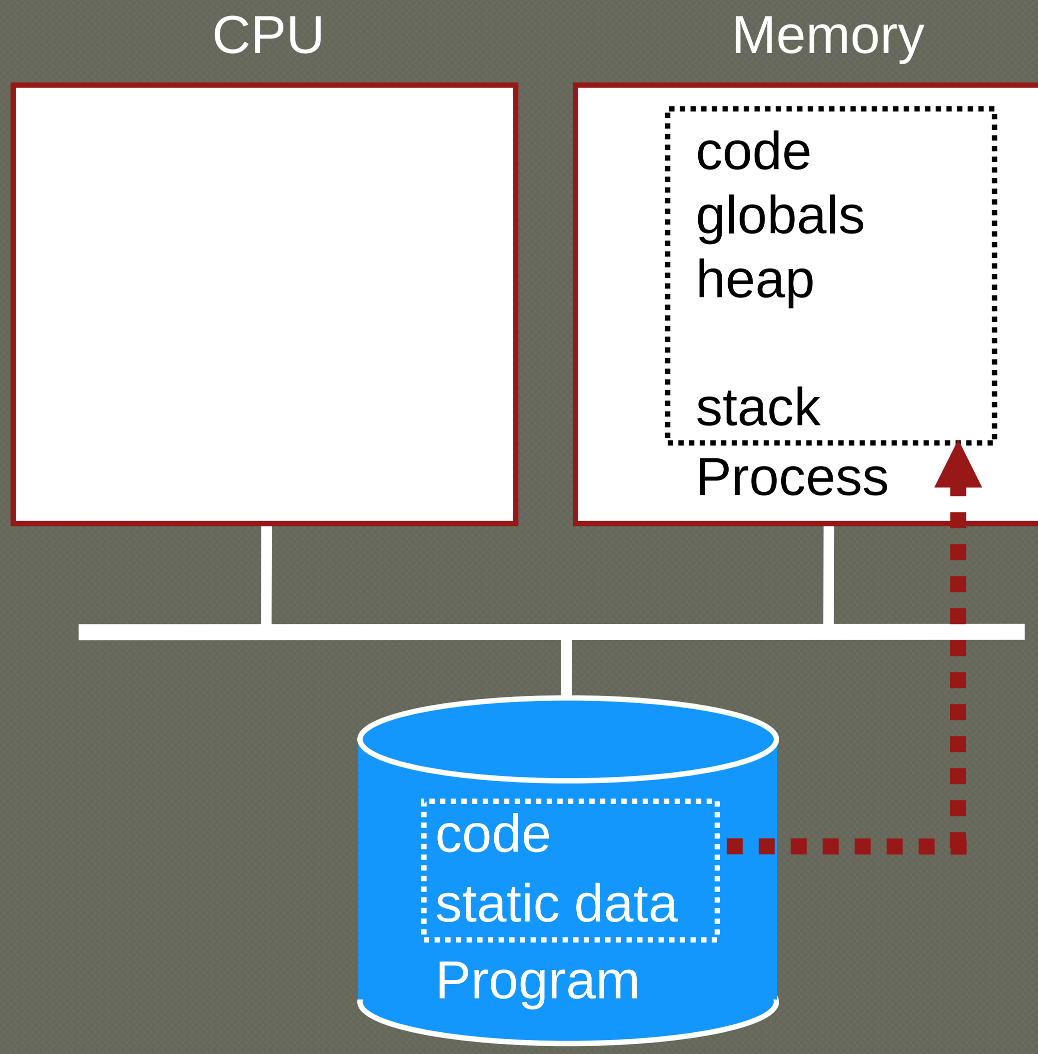
Control

- Process creation
  - What does OS do when a process is created?
    - assigns a new unique ID
    - allocates space for the process in memory
    - initializes its process control block & sets it in place (e.g. in process list)

Process Creation



Process Creation



Control blocks
States
Description
Control

Processes

Dispatch Mechanism

Process is running- how to switch to other process?

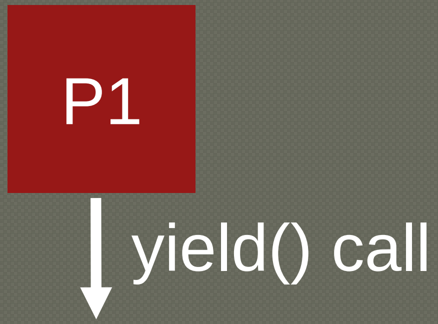
Processes

Q1: How does Dispatcher get CONTROL?

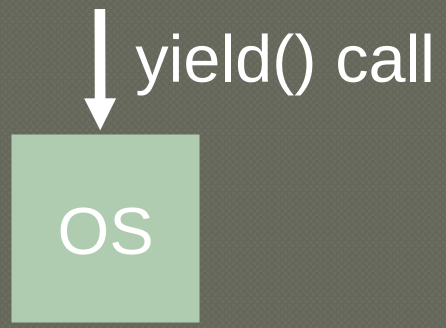
Option 1: Cooperative Multi-tasking

- Trust process to relinquish CPU to OS through traps
  - Examples: System call, page fault (access page not in main memory), or error (illegal instruction or divide by zero)
  - Provide special `yield()` system call

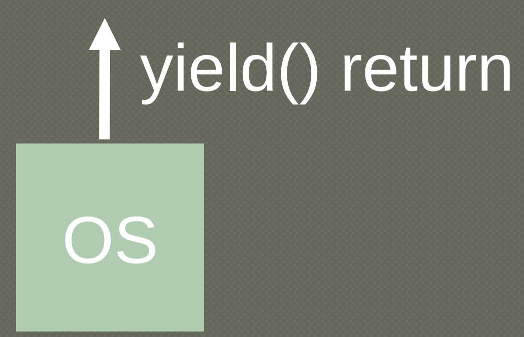
Cooperative Approach



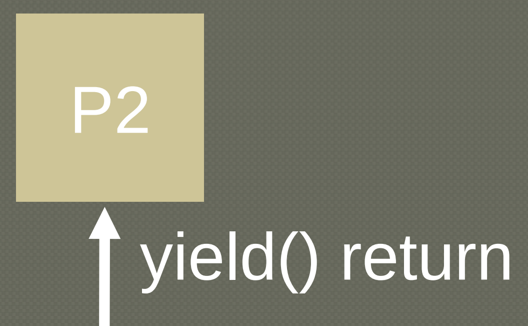
Cooperative Approach



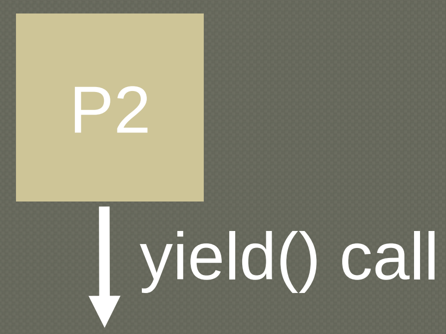
Cooperative Approach



Cooperative Approach



Cooperative Approach



Processes

Q1: How does Dispatcher get CONTROL?

- Problem with cooperative approach? YES
- Disadvantages: Processes can misbehave
  - By avoiding all traps and performing no I/O, can take over entire machine
  - Only solution: Reboot (like windows 95)!
- Not used in modern operating systems

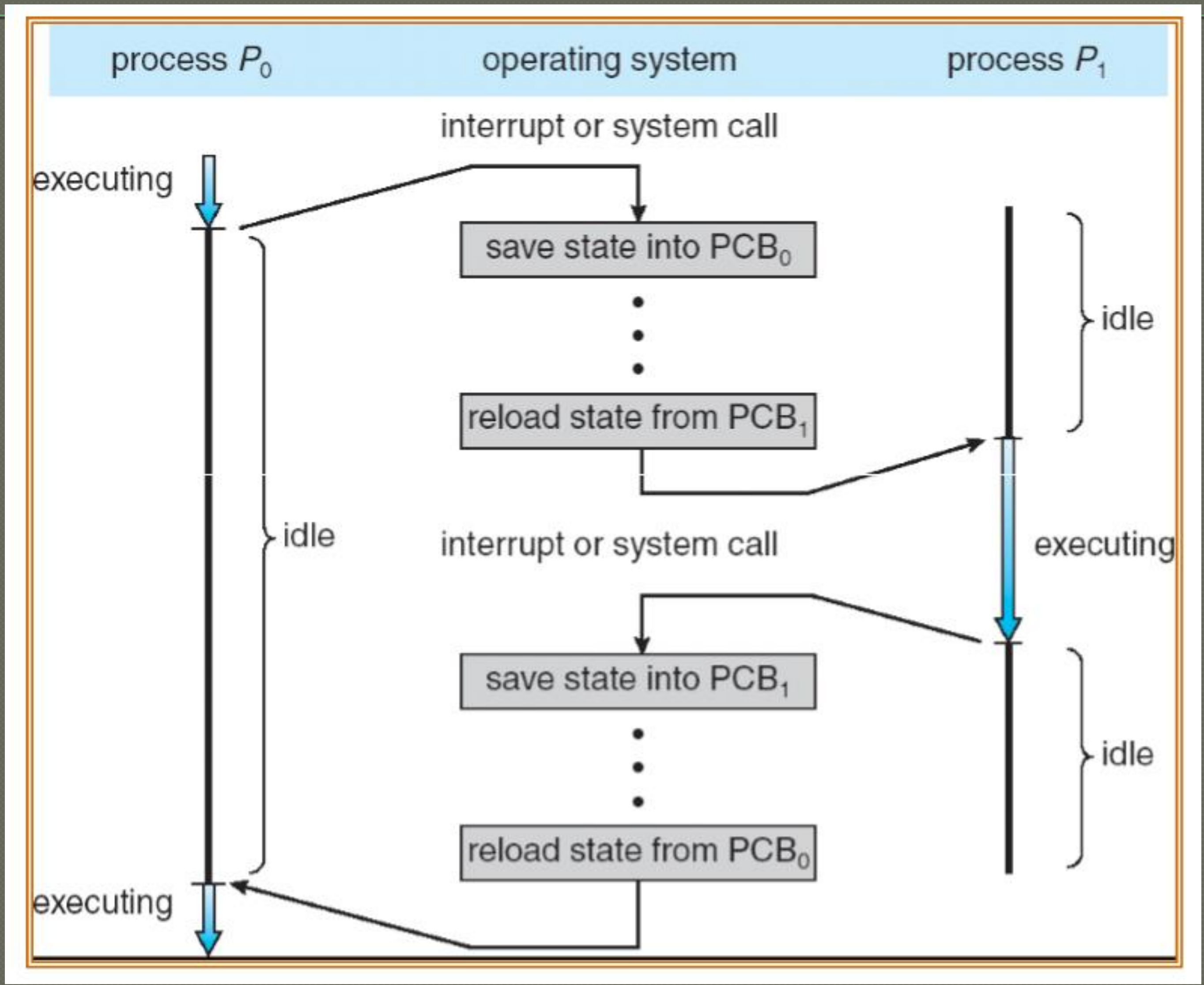
Processes

Q1: How does Dispatcher get CONTROL?

Option 2: Preemptive 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

Interrupts



process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
idle
reload state from PCB_1
idle
interrupt or system call
executing
save state into PCB_1
idle
reload state from PCB_0
executing

Topics

- Everything about Processes
  - Elements
  - Control blocks
  - States
  - Description
  - Control
- OS Execution

Done!