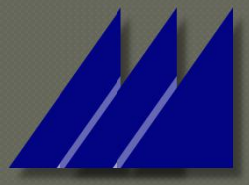


**Department of Physics,
Computer Science & Engineering**

CPSC 410 – Operating Systems I

Process Description & Control

Keith Perkins

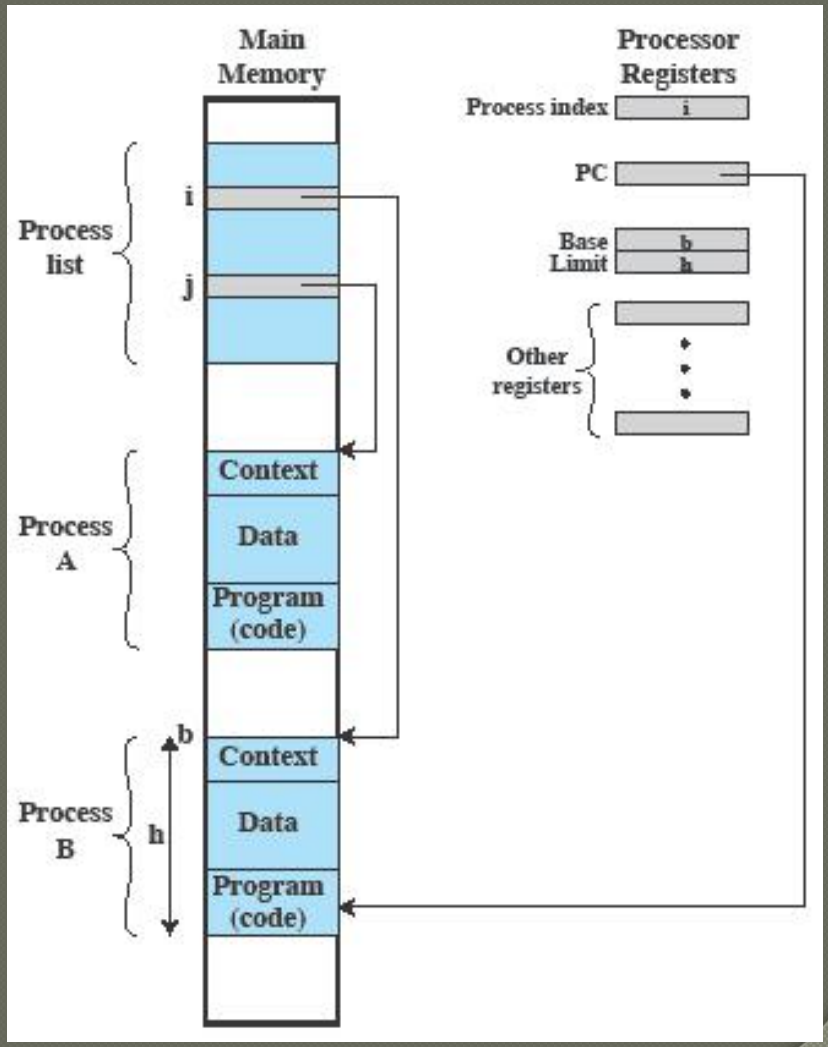
Adapted from original slides by Dr. Roberto A. Flores
Also from “CS 537 Introduction to Operating Systems” Arpaci-Dusseau

Topics

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

Revisit - Process Management

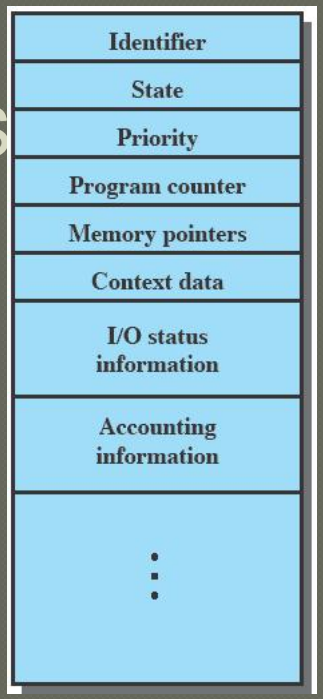
- Scheduler chooses a process to run (more later)
- Dispatcher runs it
- How? What's in the Process List?
- BTW this list is a simplification



Processes

Control Blocks

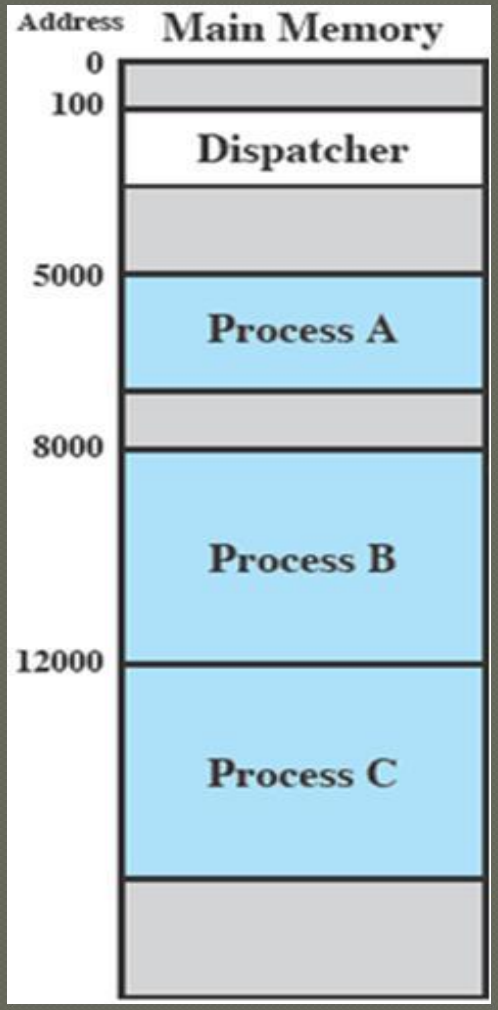
- data structure created & managed by OS
  - Identifier: unique ID
  - State: (e.g., running, blocked)
  - Priority: relative to other processes
  - Program counter: address of next instruction
  - Memory pointers: to code & data
  - I/O status: I/O in use/pending
  - Accounting: CPU time used, IDs,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

Q2: what state must be saved?

Q3: where to find processes?

Processes

States

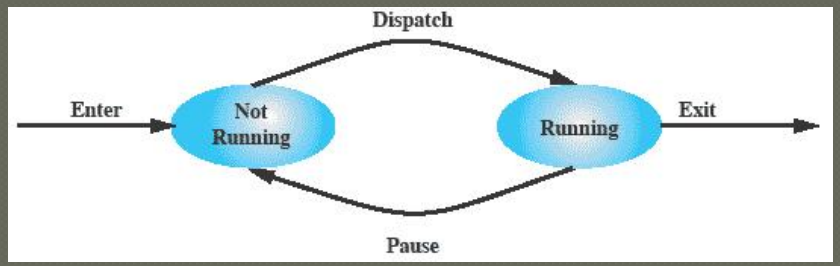
- Trace
  - Instructions executed by a process
  - In multiprogramming:
    - interleaving of instructions as processes alternate using the CPU
- The pale blue lower right is dispatcher code
- Process switches because of Interrupts (timer, I/O)

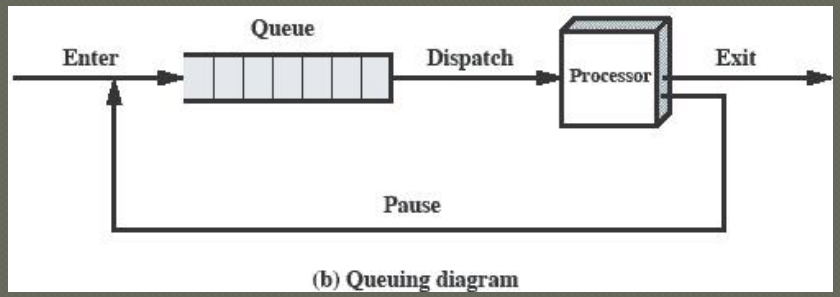
(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

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

Processes

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

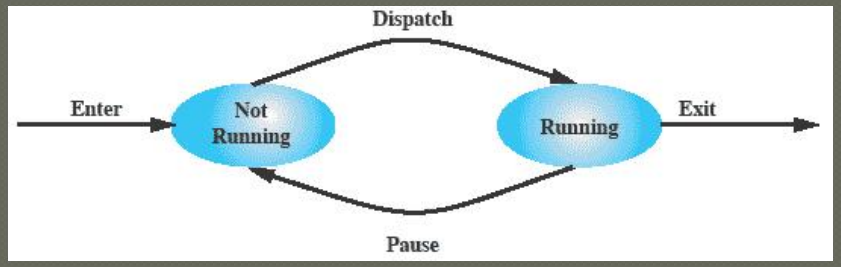


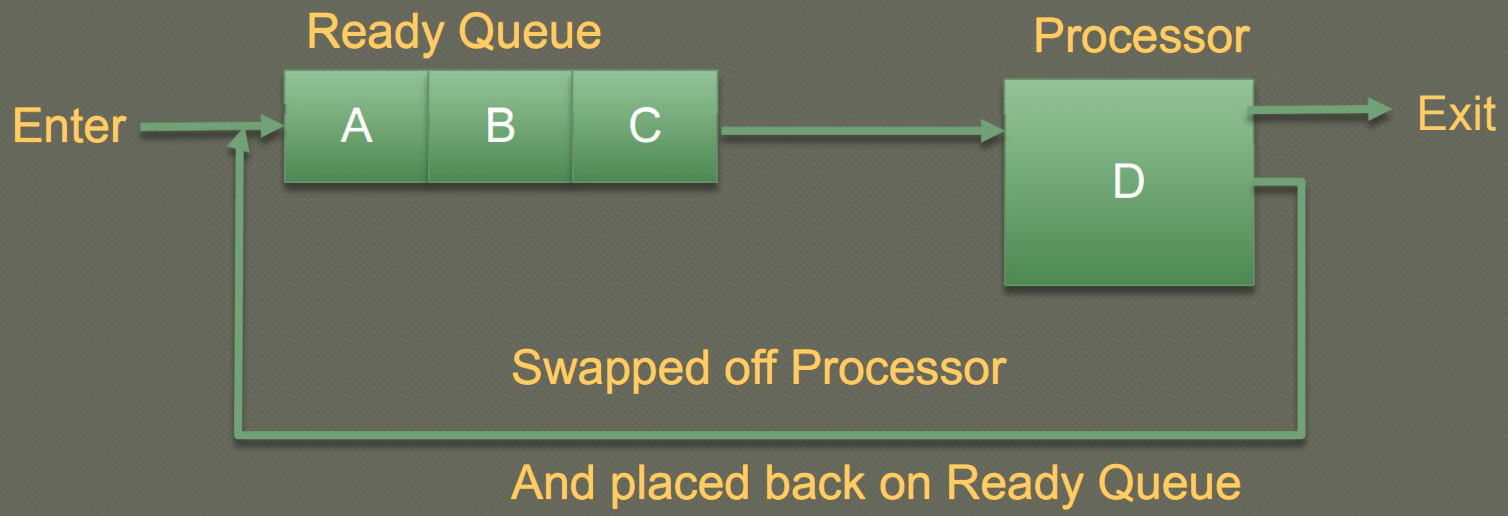


(b) Queuing diagram

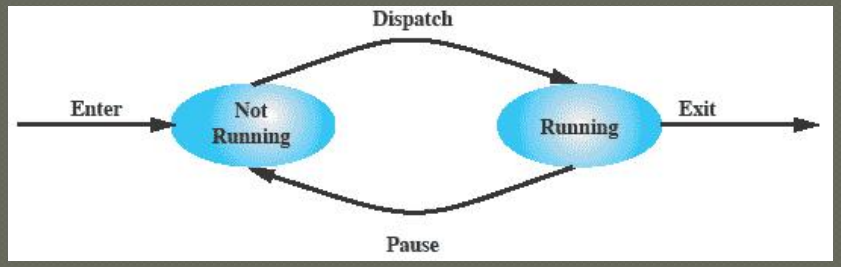
- Where do processes come from?
- When do they stop?

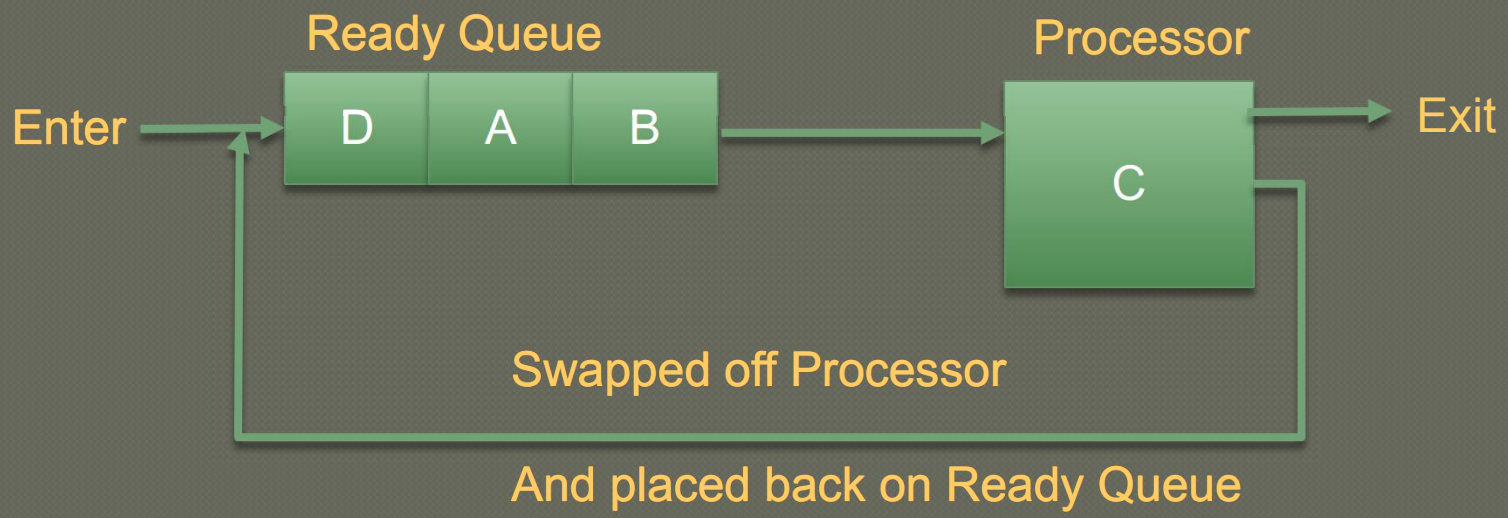
Swapping processes



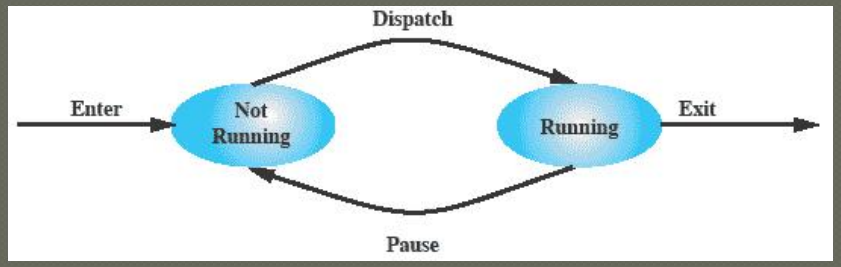


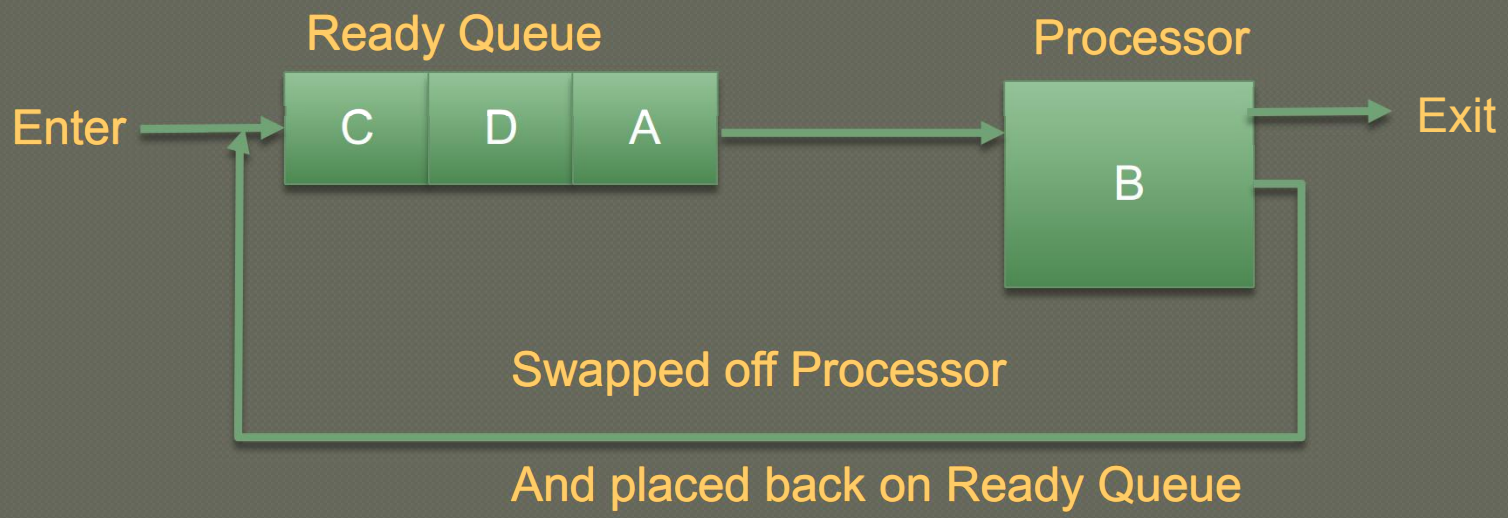
Swapping processes



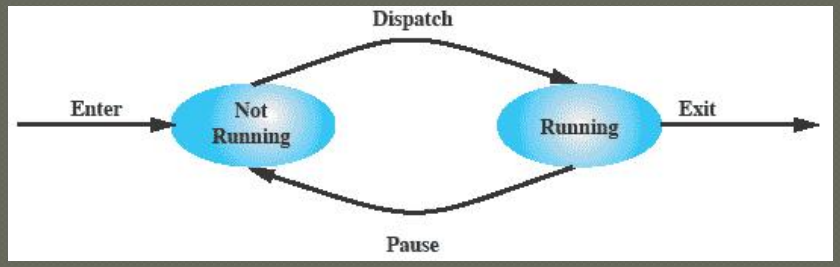


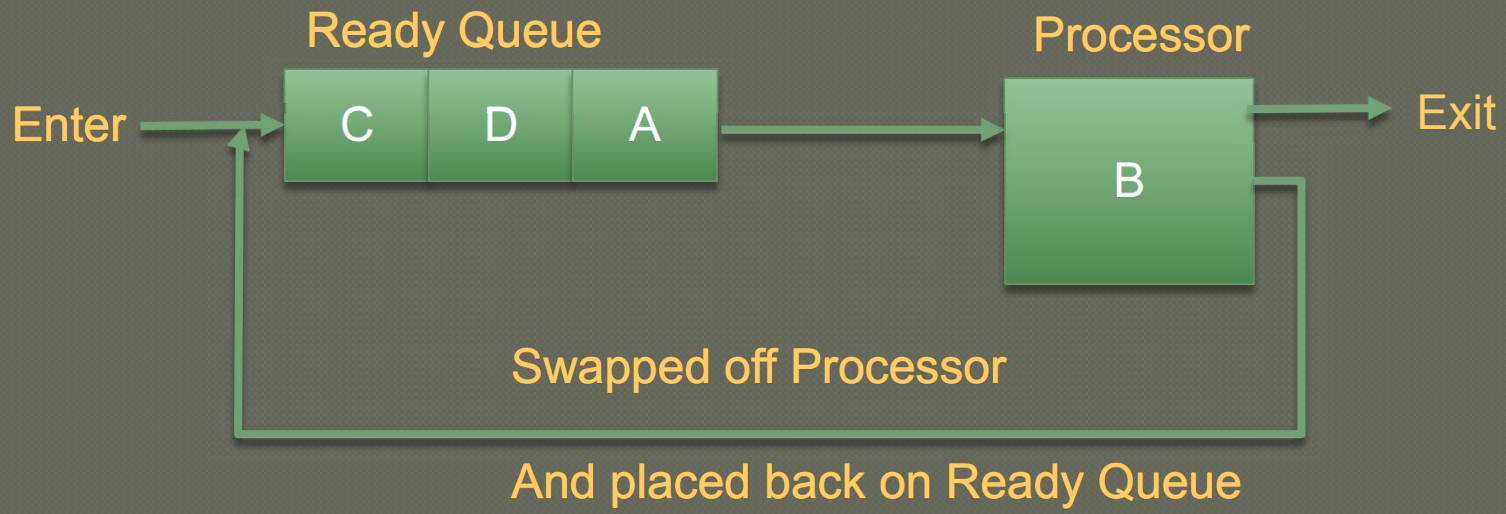
Swapping processes





Swapping processes





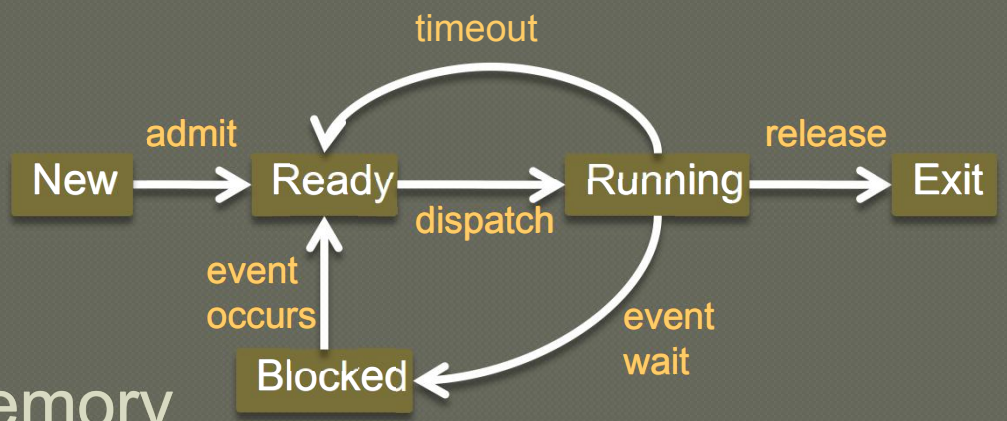
This cycle continues with some processes finishing and new proc taking their place in the Ready Queue

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
    - 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

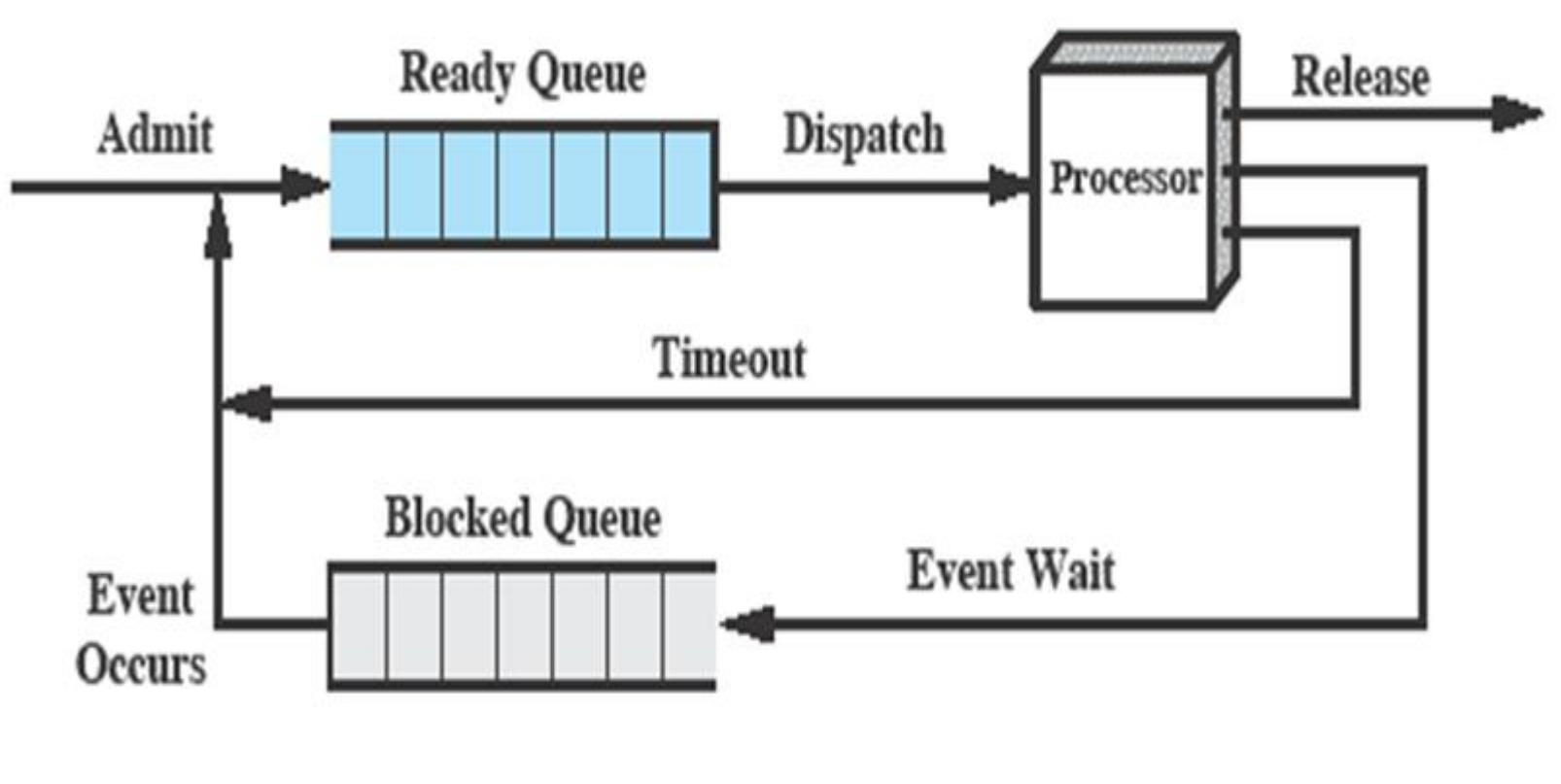
Processes

States (5 states)



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

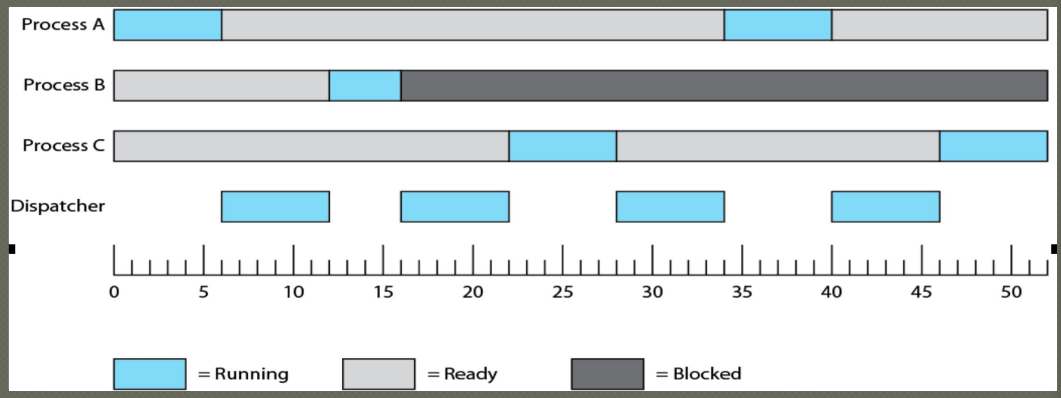
Using Two Queues



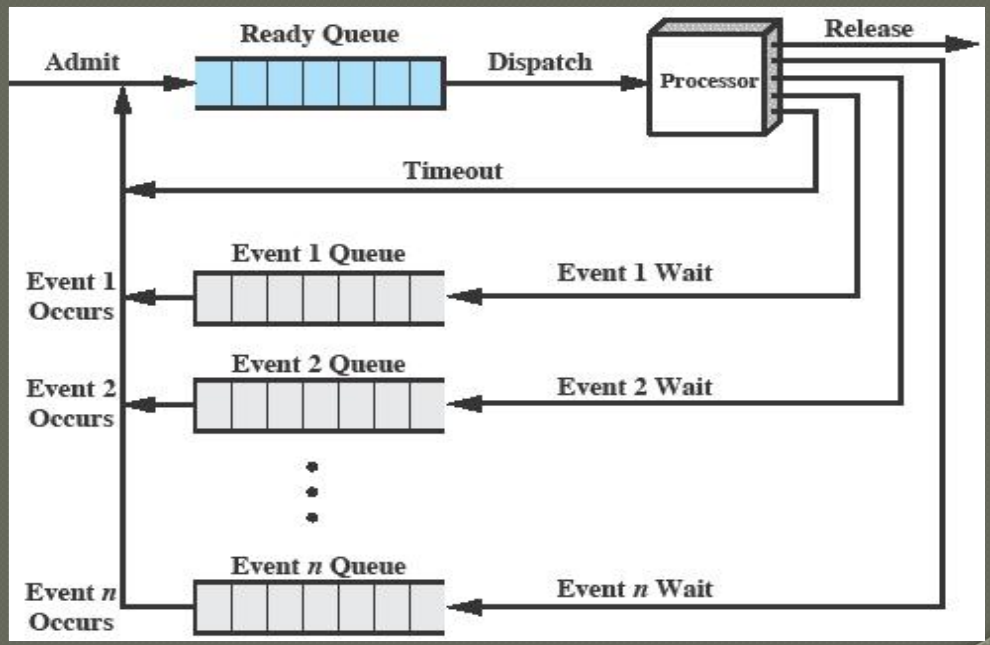
Processes

States (5 states)

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



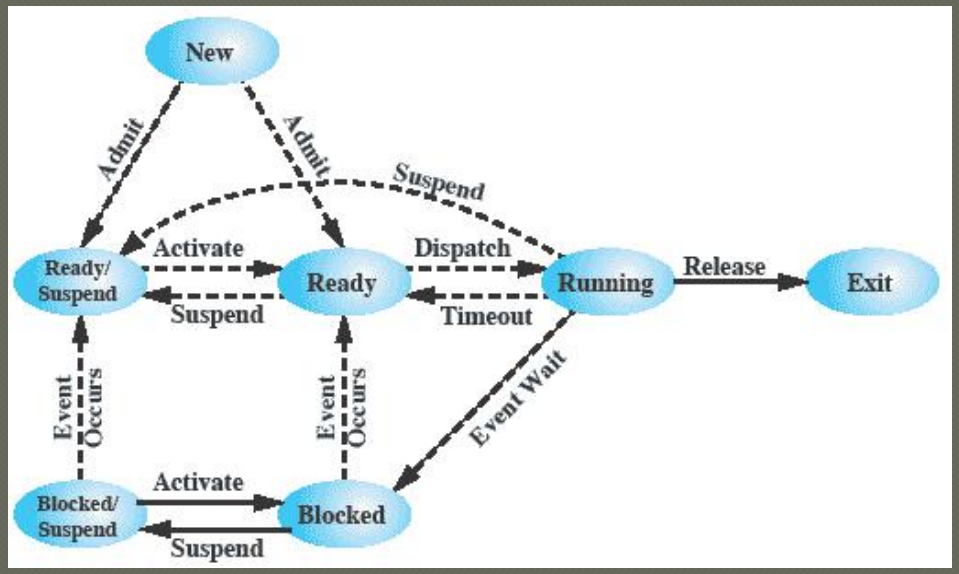
- Multiple block queues (1 per I/O device)



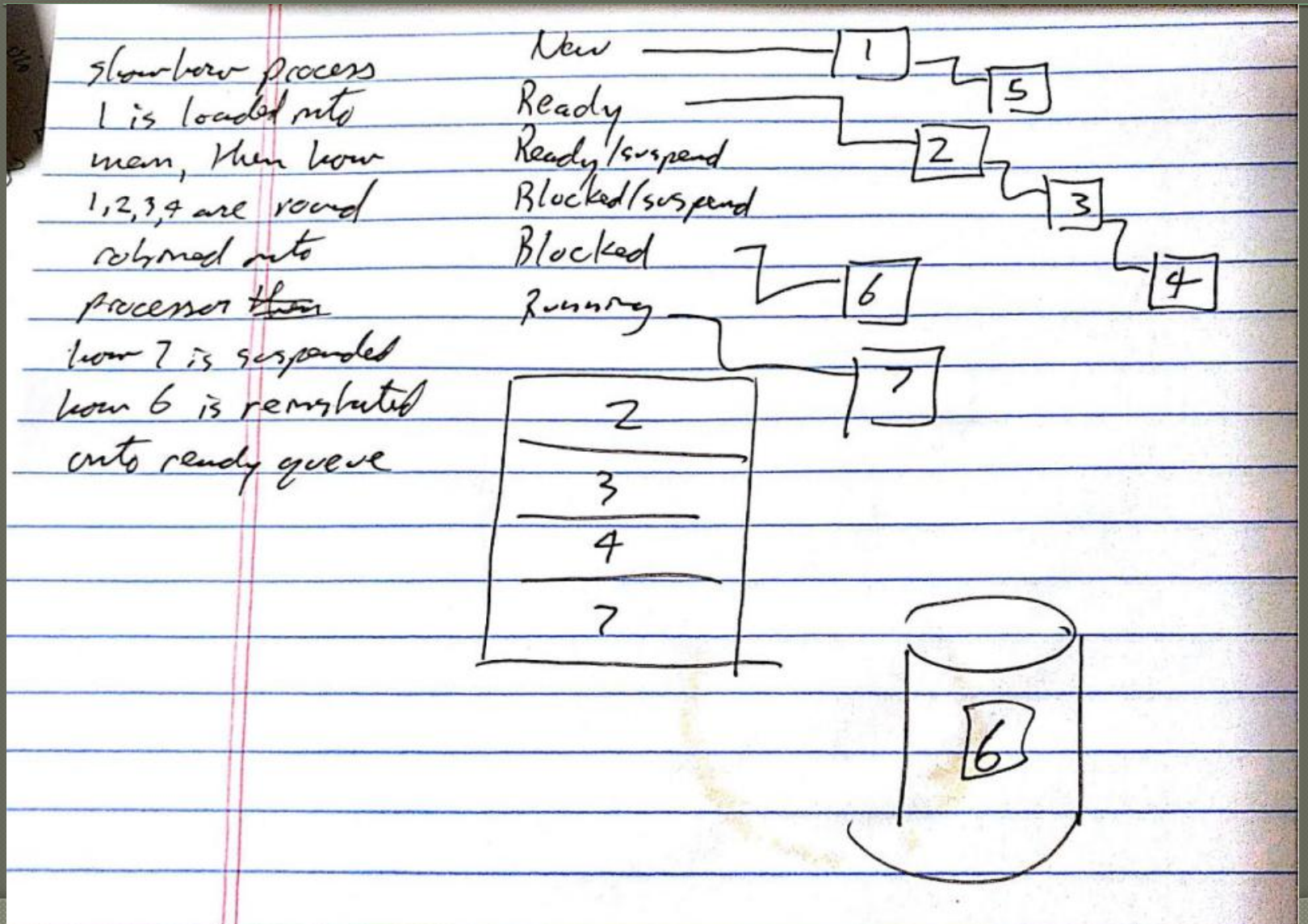
Processes

States (7 states)

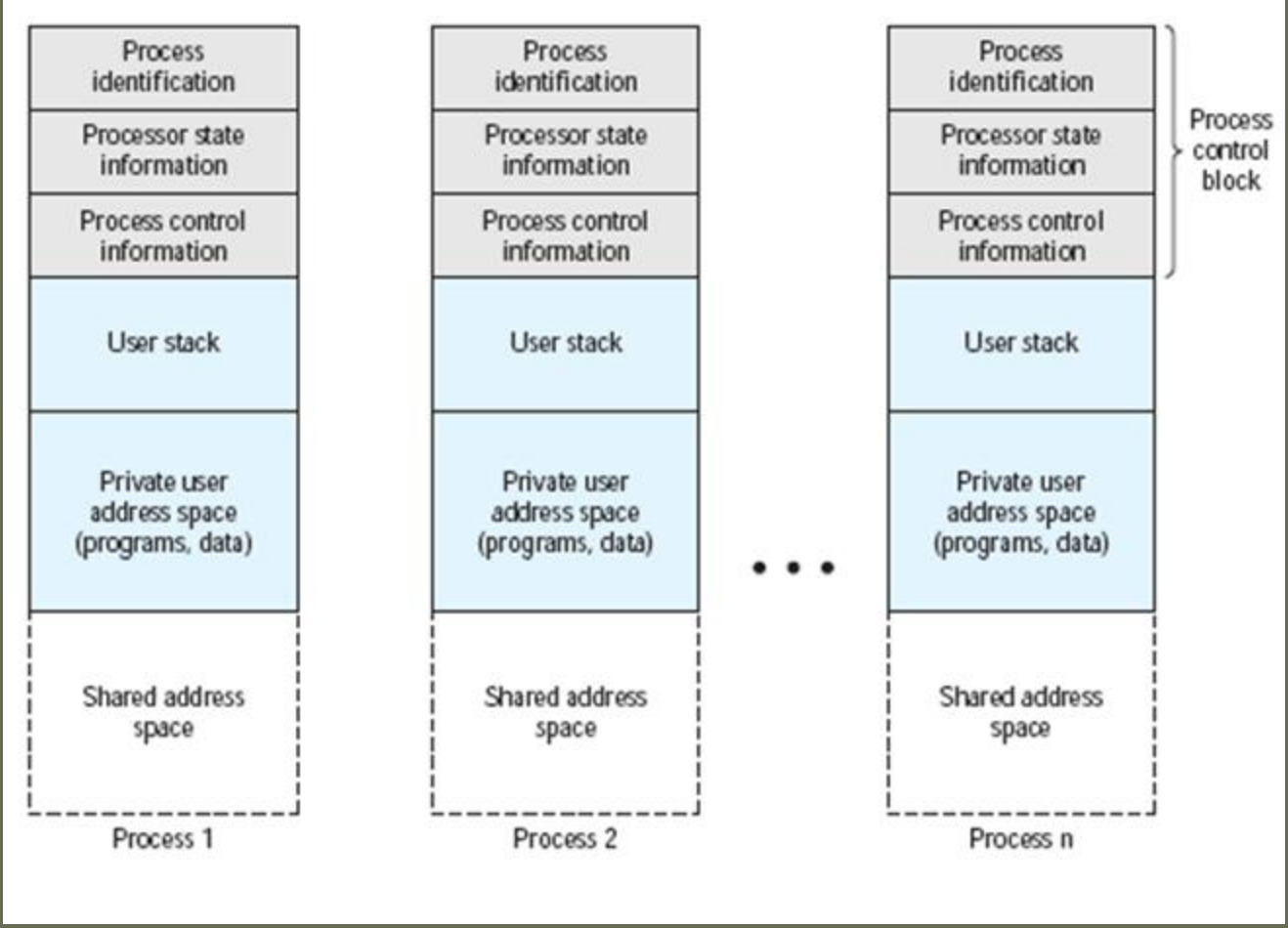
- What if running I/O intensive processes and all are waiting for I/O?
  - Solution: suspend blocked processes to disk, bring in new (from new or ready/suspend)



Talk about processes moving on and off queues



Structure of Process Images in Virtual Memory



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

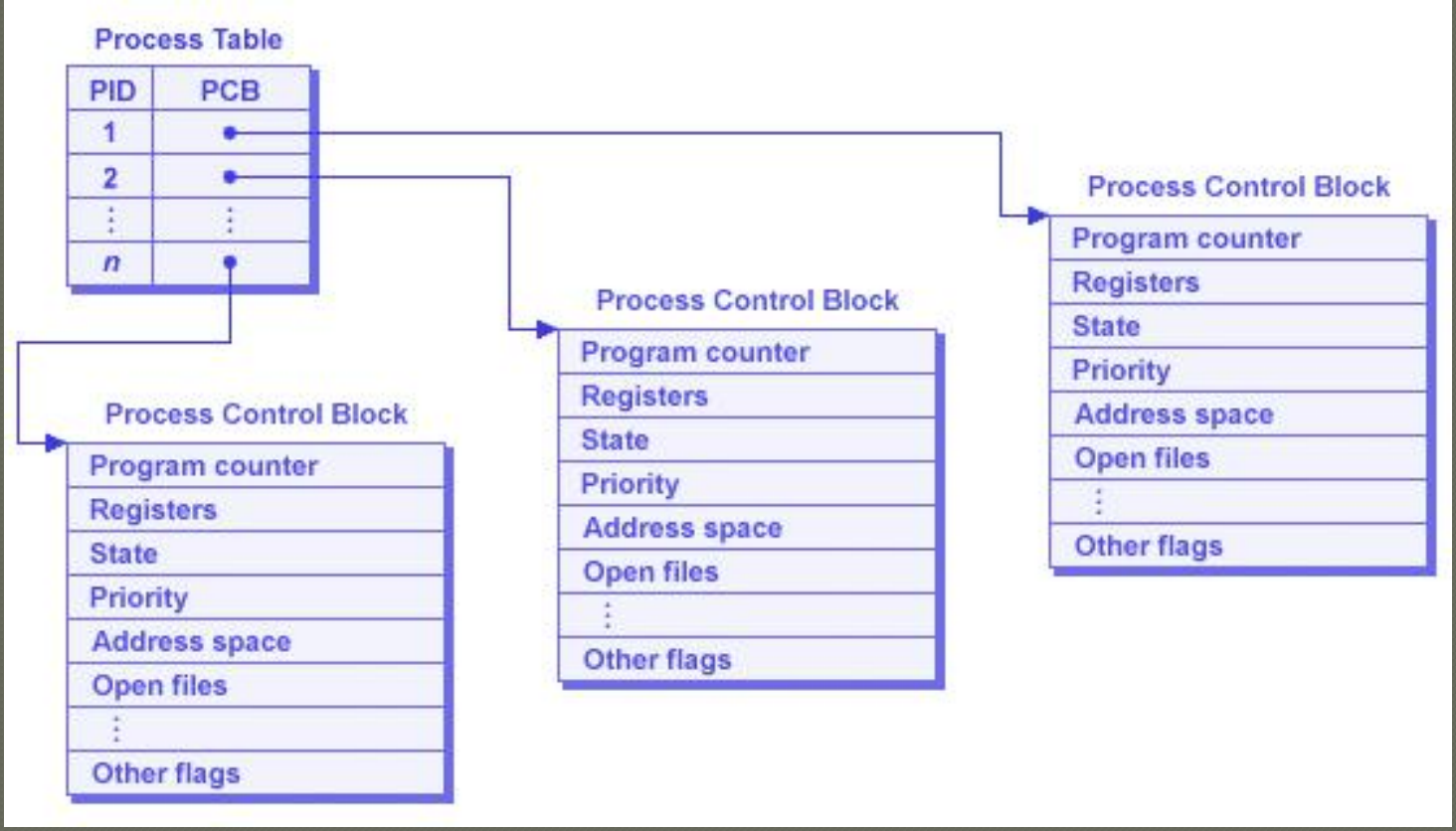
Process identification
- Each process has a unique ID
- IDs are used for reference:
  - in other tables
  - in inter-process communication
  - when a parent spawns a child process

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

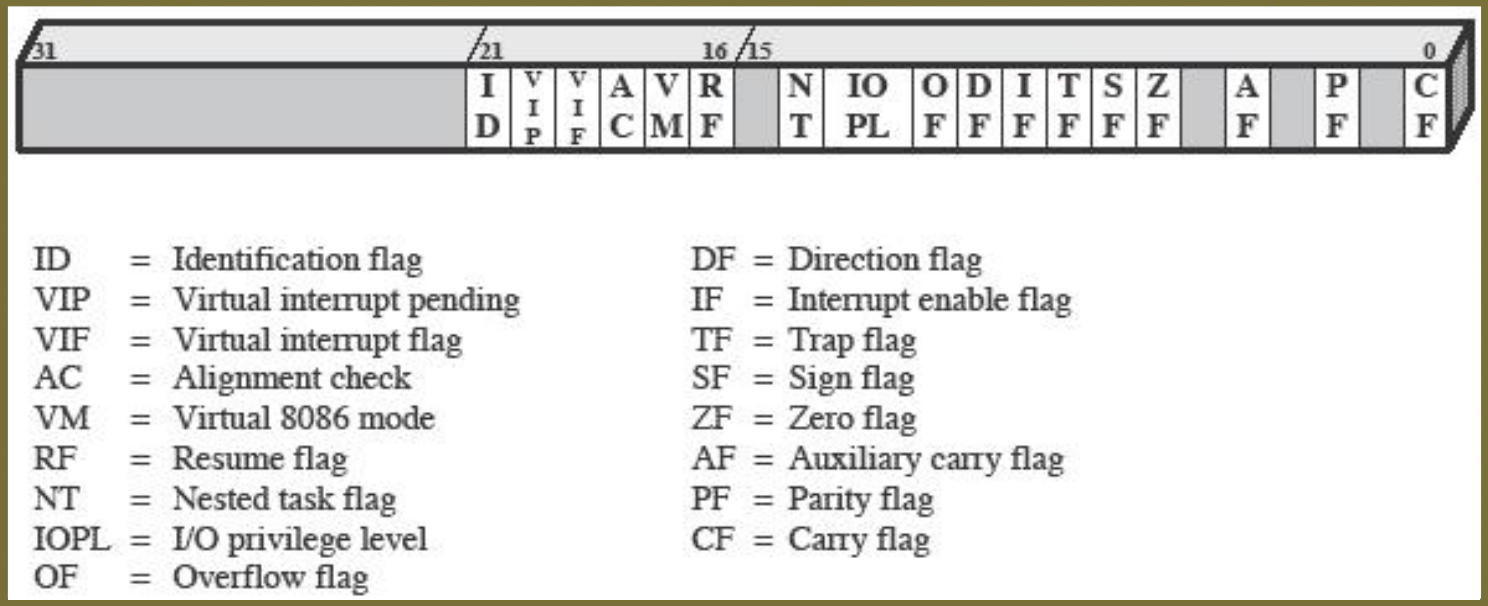
- reference to memory, I/O & file tables

Processes

- Process tables



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



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

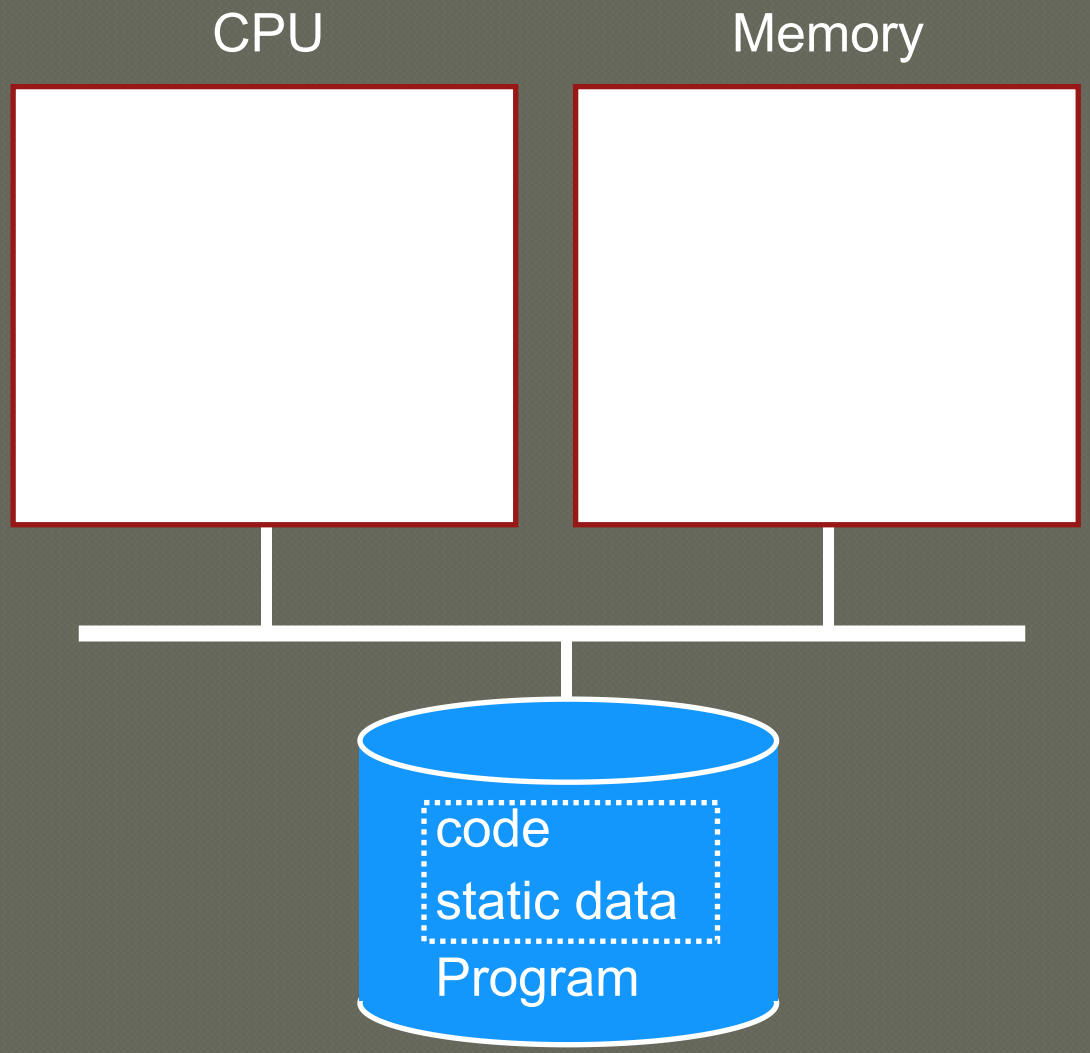
- reference (directly/indirectly) memory, I/O & file tables

Processes

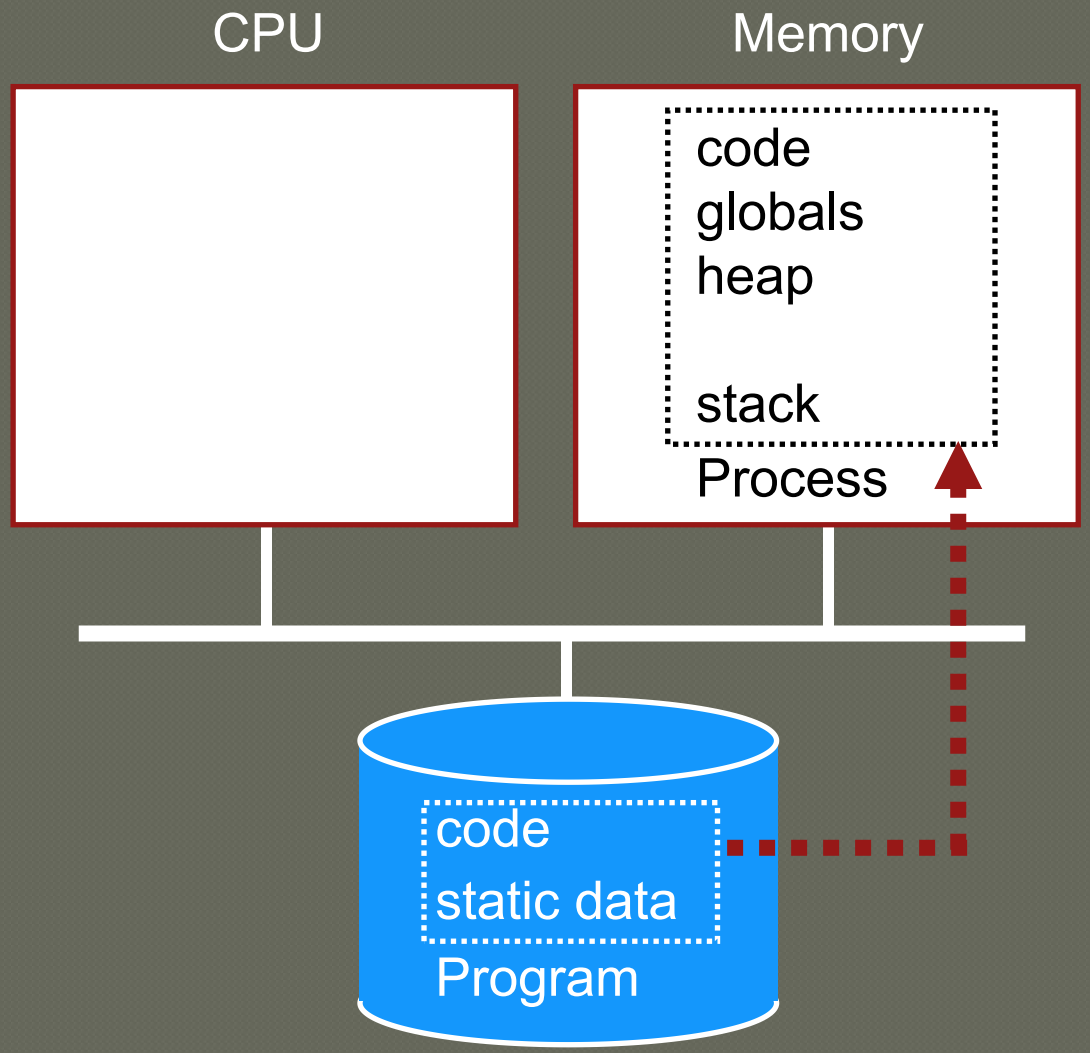
Control

- 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



Processes
Dispatch Mechanism

Process is running- how to switch to other process?

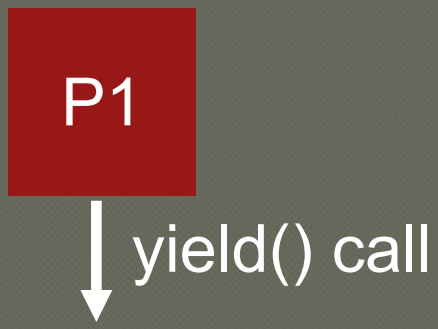
Processes

Q1: How does Dispatcher get CONTROL?

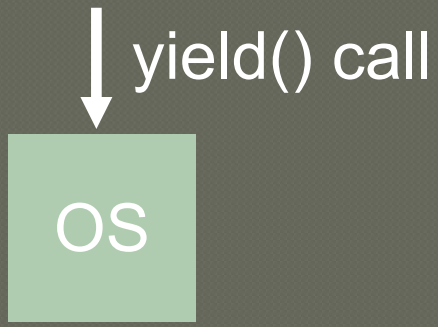
Option 1: Cooperative Multi-tasking

- 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

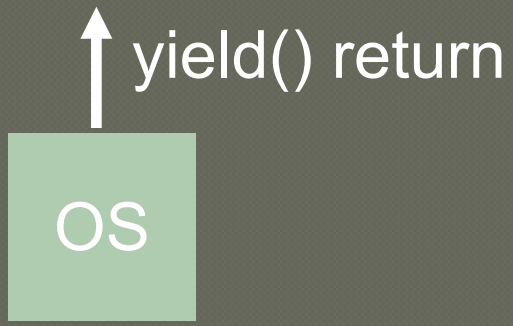
Cooperative Approach



Cooperative Approach

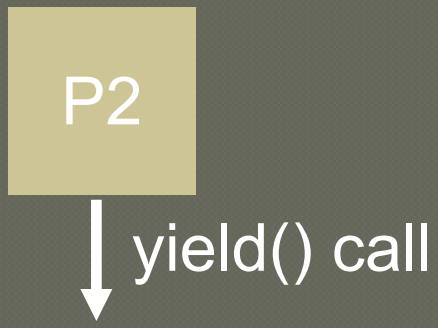


Cooperative Approach



Cooperative Approach

Cooperative Approach



Processes

Q1: How does Dispatcher get CONTROL?

- Problem with cooperative approach? YES
- 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

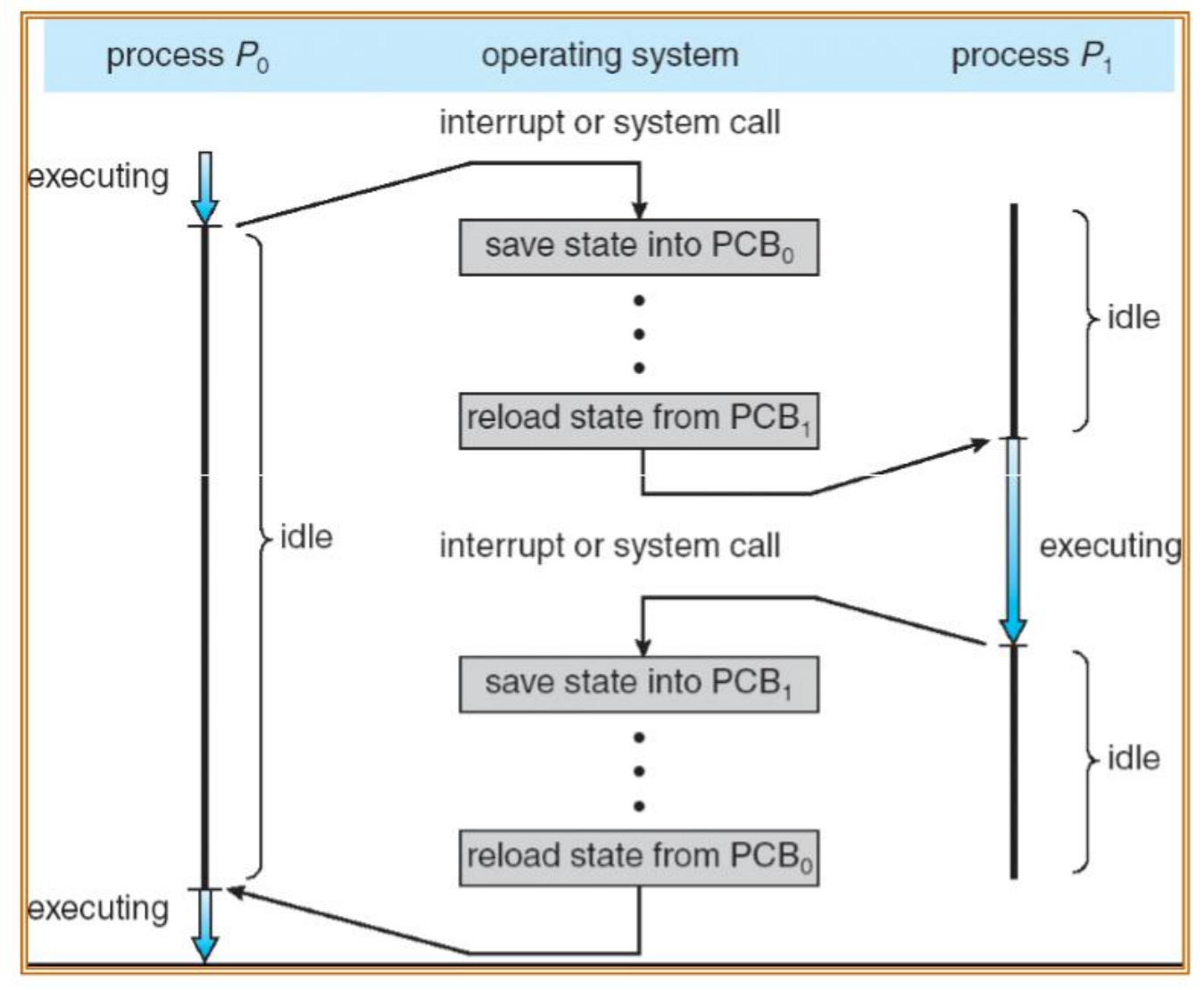
Processes

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 periodic timer interrupt (CPU or separate chip). OS uses this interrupt to measure amont of time process has been running
- User must not be able to mask timer interrupt
- Dispatcher counts interrupts between context switches
  - Example: if timer interrupt occurs every 10ms, then Waiting 20 timer interrupts yields a 200 ms time slice
  - Common time slices range from 10 ms to 200 ms

Process switch



Topics

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

Done!

Chapter 3 Topics

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

OS Execution

- OS is software, right?
  - How is it different from just another process?
  - How is it controlled?

a) Non-process Kernel

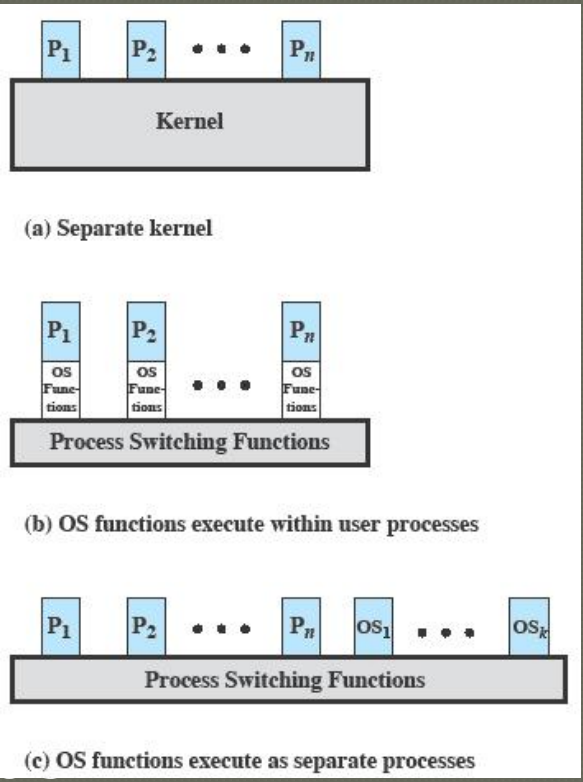
- Processes are processes. The kernel is the kernel.

b) Execution within user processes

- OS is a bare process switching mechanism
- OS routines are linked to user programs (OS data is shared)

c) Process-based OS

- OS routines run as independent process
- Modular approach for parallelism (e.g., in one CPU, user processes in another)



Execution *Within* User Processes

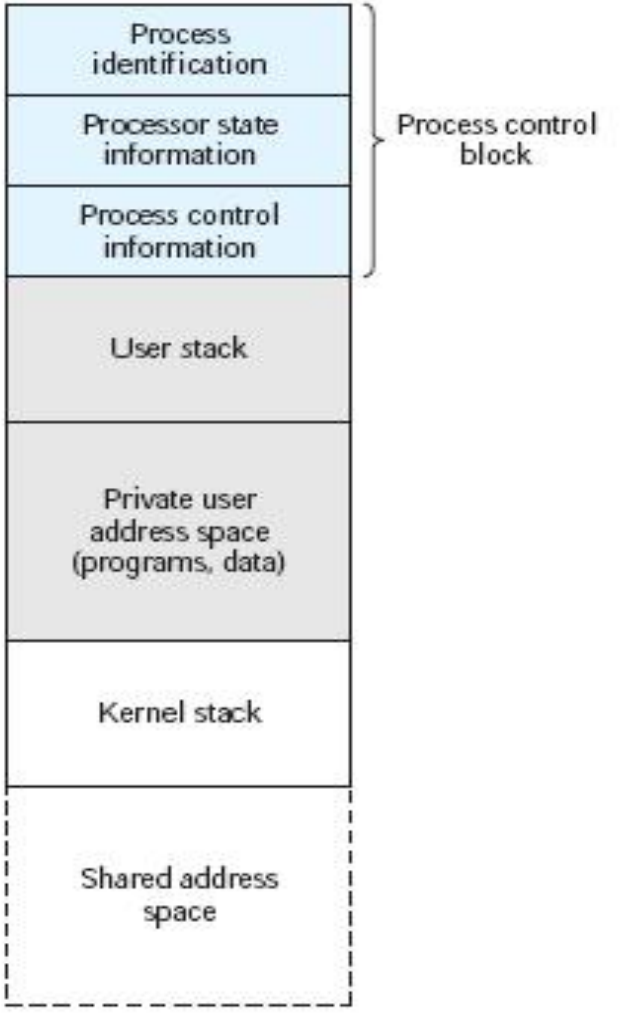


Figure 3.16 **Process Image: Operating System Executes within User Space**