Processes

Process

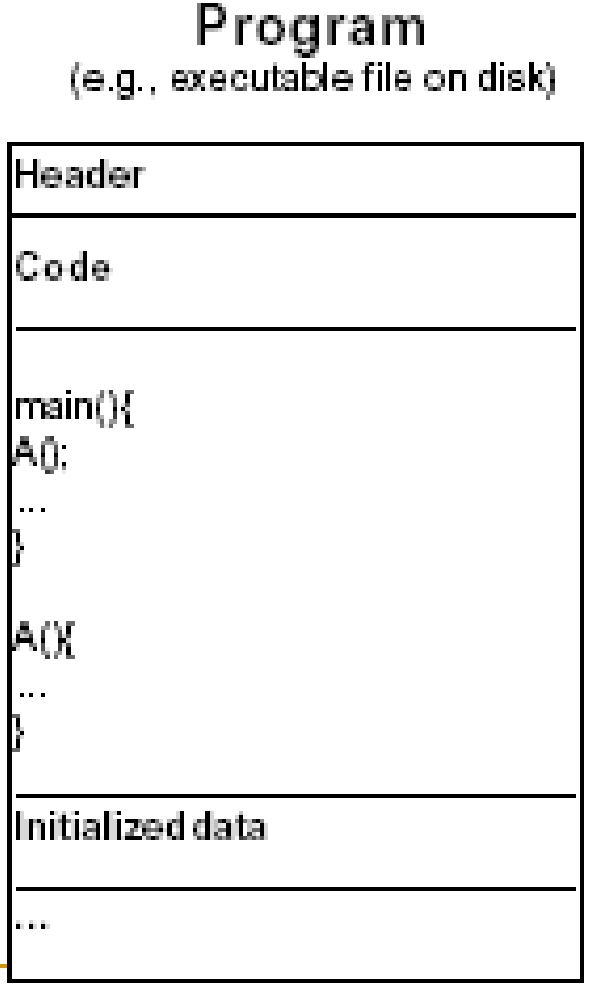
- A computer program in execution on a machine is a process
- More formally:
  - A **Sequential stream of Execution** in its own **address space**

Process Address Space

- A list of memory locations from some min (usually 0) to some max that a process can read and write.
- Contains
  - the executable program
  - program's data
  - Stack
  - Associated with a process is a set of registers e.g. PC,SP and other information to run the program.

Process =? Program

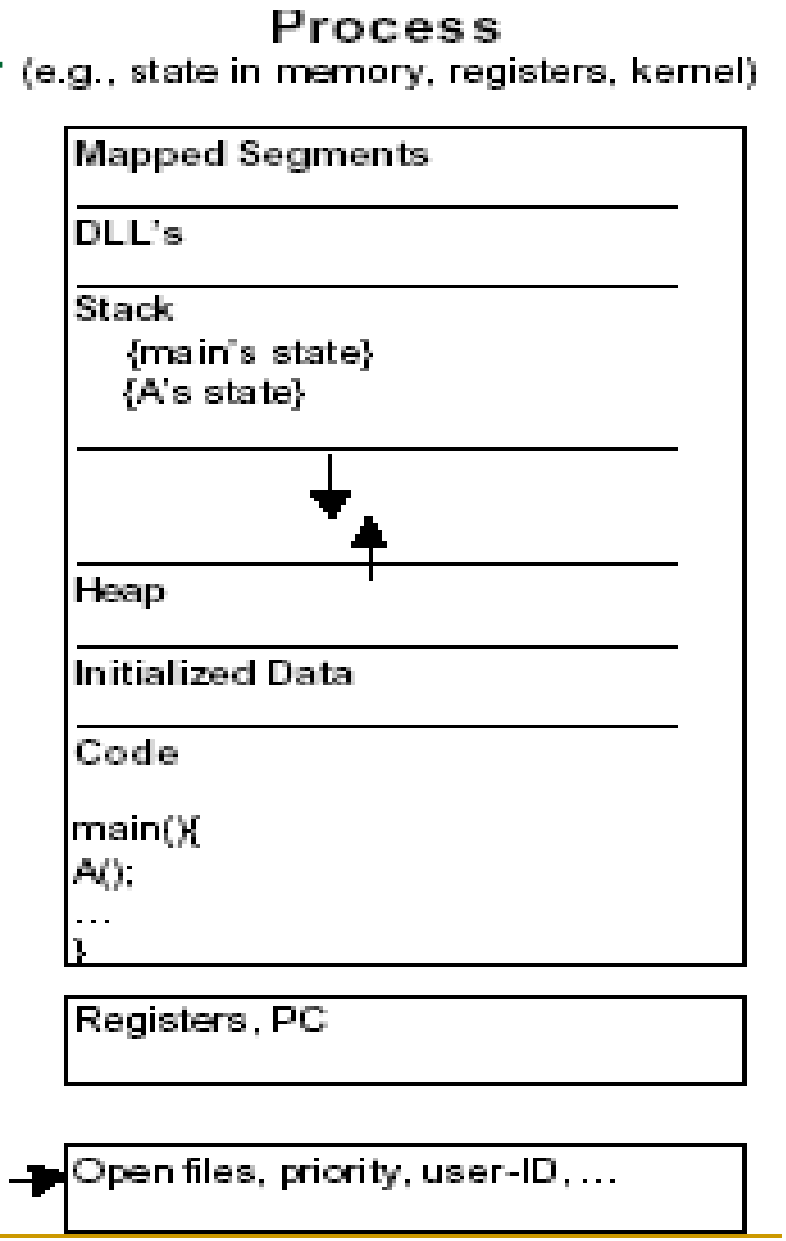
- **Program:** series of commands (e.g. C statements, assembly commands, shell commands)



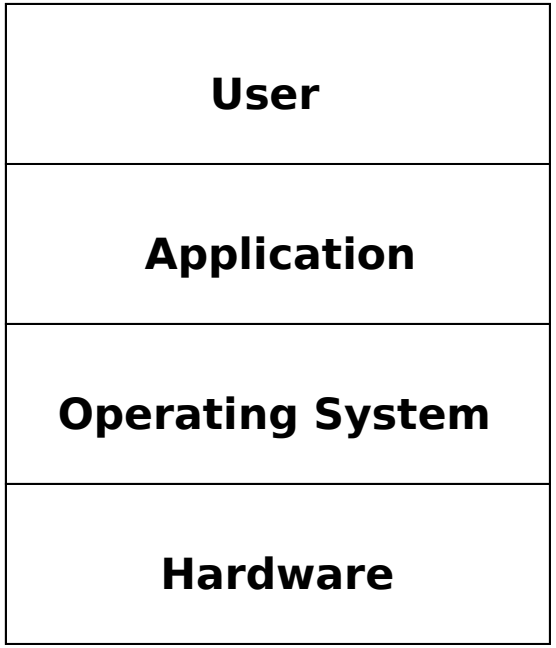
Process =? Program

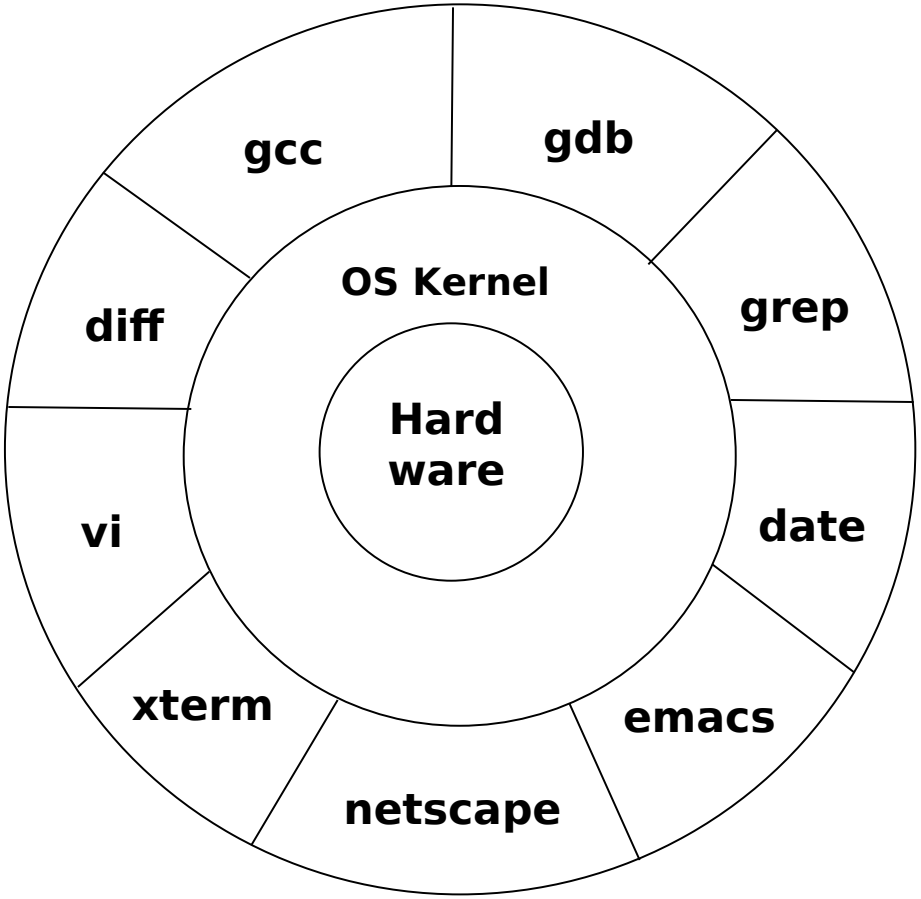
- **Anatomy of a process**
  - 1) More to a process than just a program
    - program is *part* of process state
    - I run *dir*, you run *dir* – same program, different processes
  - 2) Less to a process than a program
    - A program can invoke more than one process to get the job done
    - e.g. cc starts up cpp, cc1, cc2

Process =? Prog



The Operating System controls the machine





CPU State

- CPU registers contain the current state
  1. Processor Status Word (PSW): includes bits
     - Privileged or normal
     - Outcome of the last arithmetic operation (zero,-ve, +ve, overflow, carry)
     - Which interrupts are allowed and which are not
  2. Instruction Register (IR):
     - The current instruction being executed
  3. Program Counter (PC):
     - Address of the next instruction to be executed
  4. Stack Pointer (SP):
     - the address of the current stack frame, including function's local variables and return information.
  5. General purpose registers:
     - used to store addresses and data values as directed by the compiler.

Memory Contents

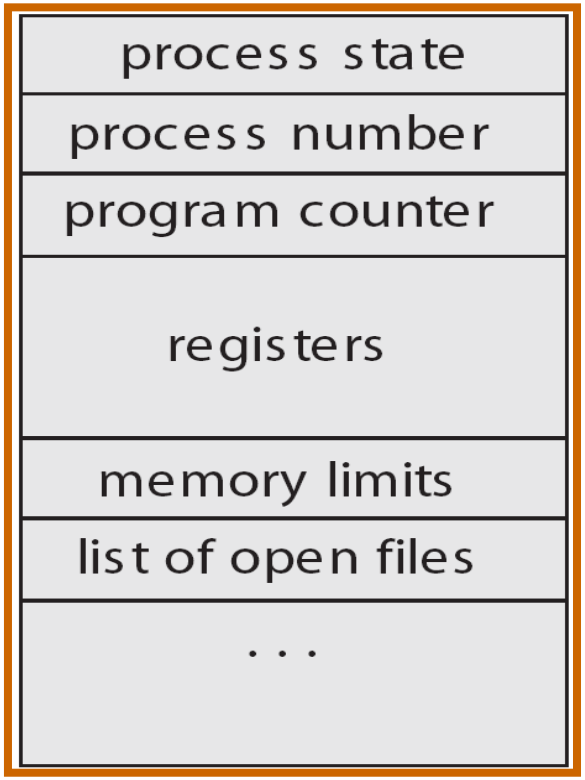
- Only a small part of an application's data can be stored in registers. The rest is in memory.
- Typically divided into a few segments:
  - Text/application code
    - read-only
    - might be shared by a number of processes?
  - Data
    - The application's predefined data structures
  - Heap
    - An area from which space can be allocated dynamically at runtime, using functions like **`new`** or **`malloc`**.
  - Stack
    - Where register values are saved
    - local variables allocated
    - Return Address of subroutine calls
    - All the addressable memory together is called?
- The process's address space.

Environment

- Contains the relationships with other entities
- A process does not exist in a vacuum
- It typically has connections with other entities, such as
  - A terminal where the user is sitting.
  - Open files
  - Communication channels to other processes, possibly on other machines.

Process Control Block

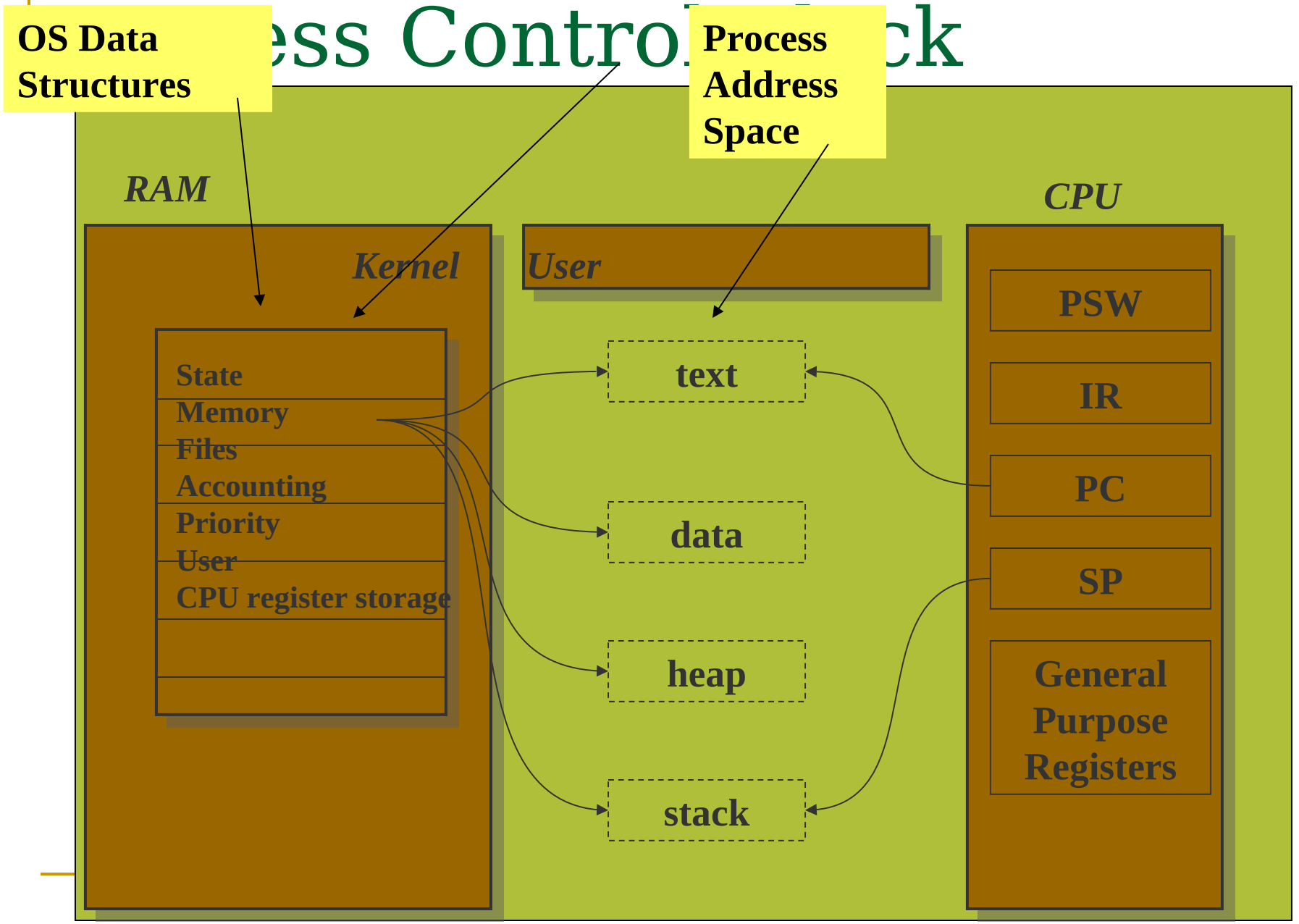
- The OS keeps all the data it needs about a process in the process control block (PCB)
- Thus another definition of a process:
  - “the entity described by a PCB”
- This includes many of the data items described above, or at least pointers to where they can be found
  - e.g. for the address space



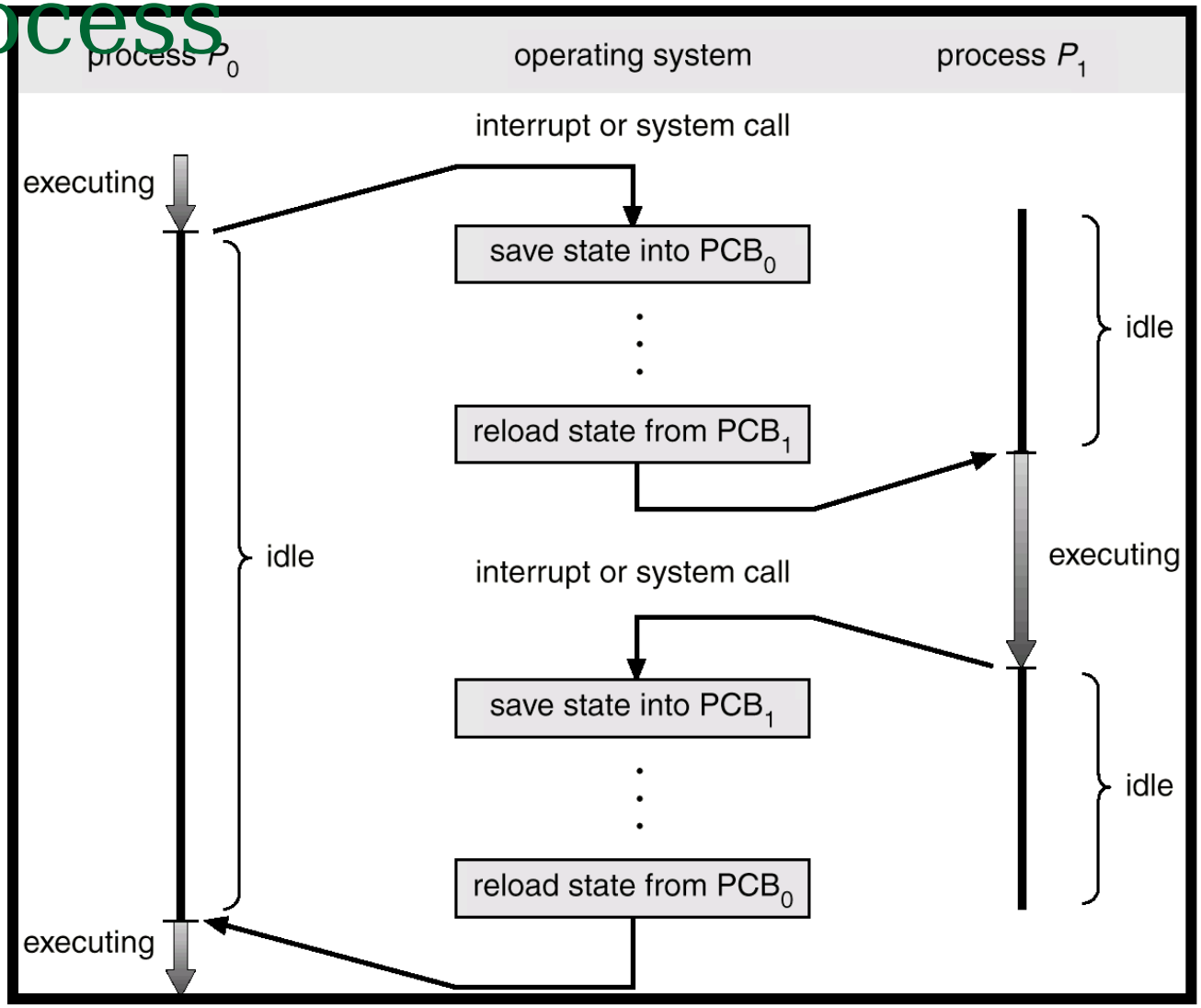
Process Identification

- Process ID, a unique numeric identifier
- User ID
  - Who runs the process. Why?
  - Used to determine what access rights the process has

ess Control ck



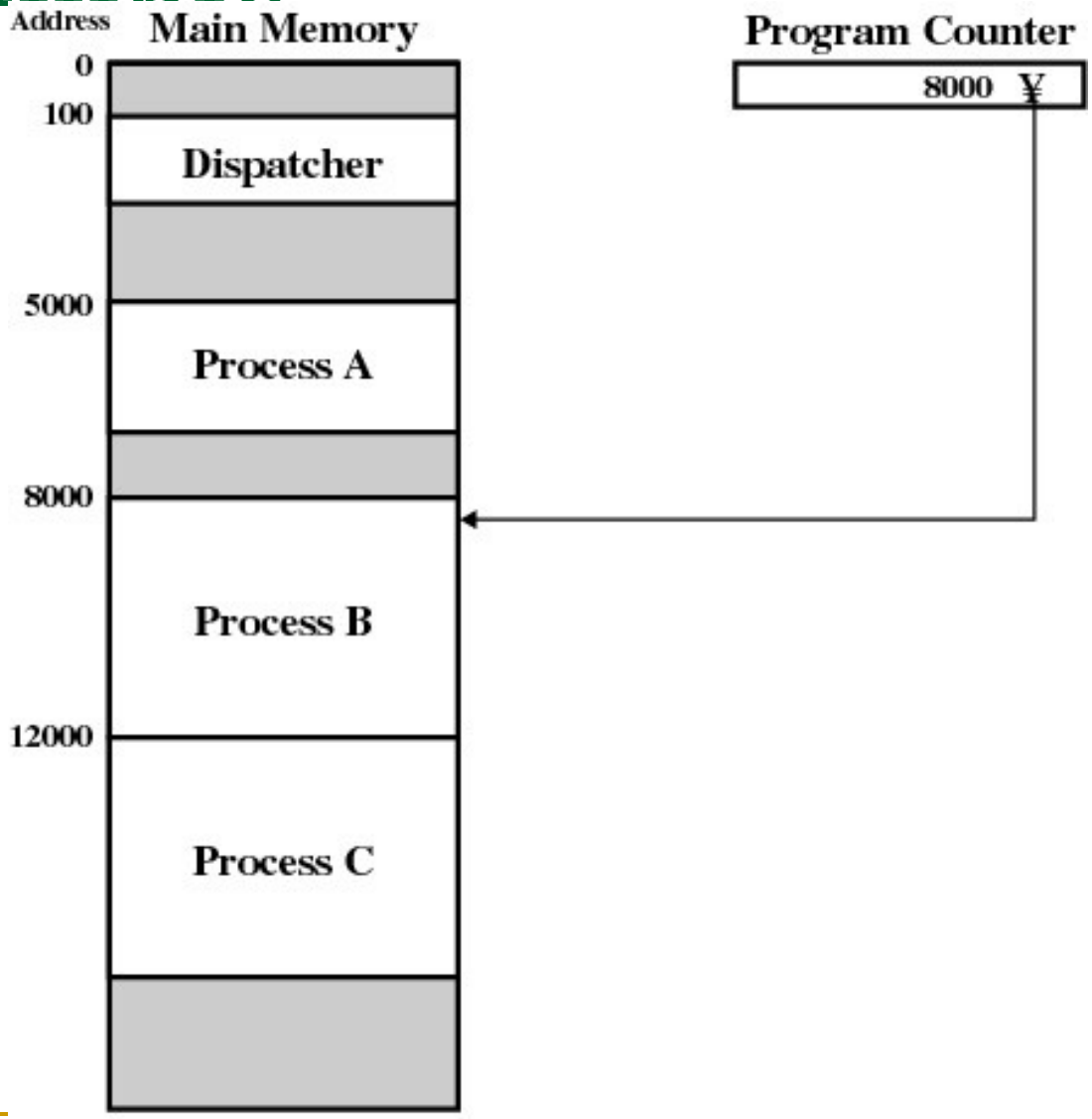
CPU Switch From Process to Process



CPU Switch From Process to Process

- Switching a process requires
  - Saving the state of old process
  - Loading the saved state of the new process
- This is called **Context Switch**
- Part of OS responsible for switching the processor among the processes is called **Dispatcher**

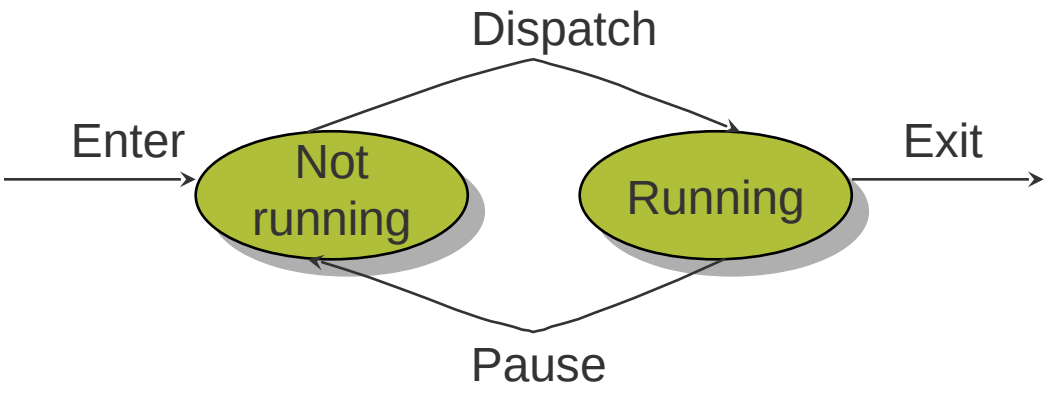
Process Example



Process States

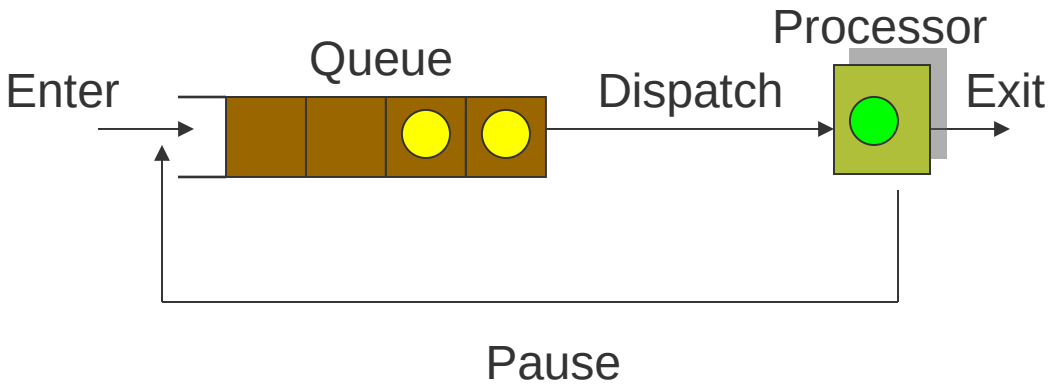
- At any given time a process is either running or not running
- Number of states
  - Running
  - Not Running
- When the OS creates a process, the process is entered into which state?
  - Not Running

Two-state process model



- Number of processes **Running** at a particular time?
- Number of processes **Not Running** at a particular time?
- Data Structure?
- Processes that are **Not Running** at a particular time should be kept in some sort of a **queue**

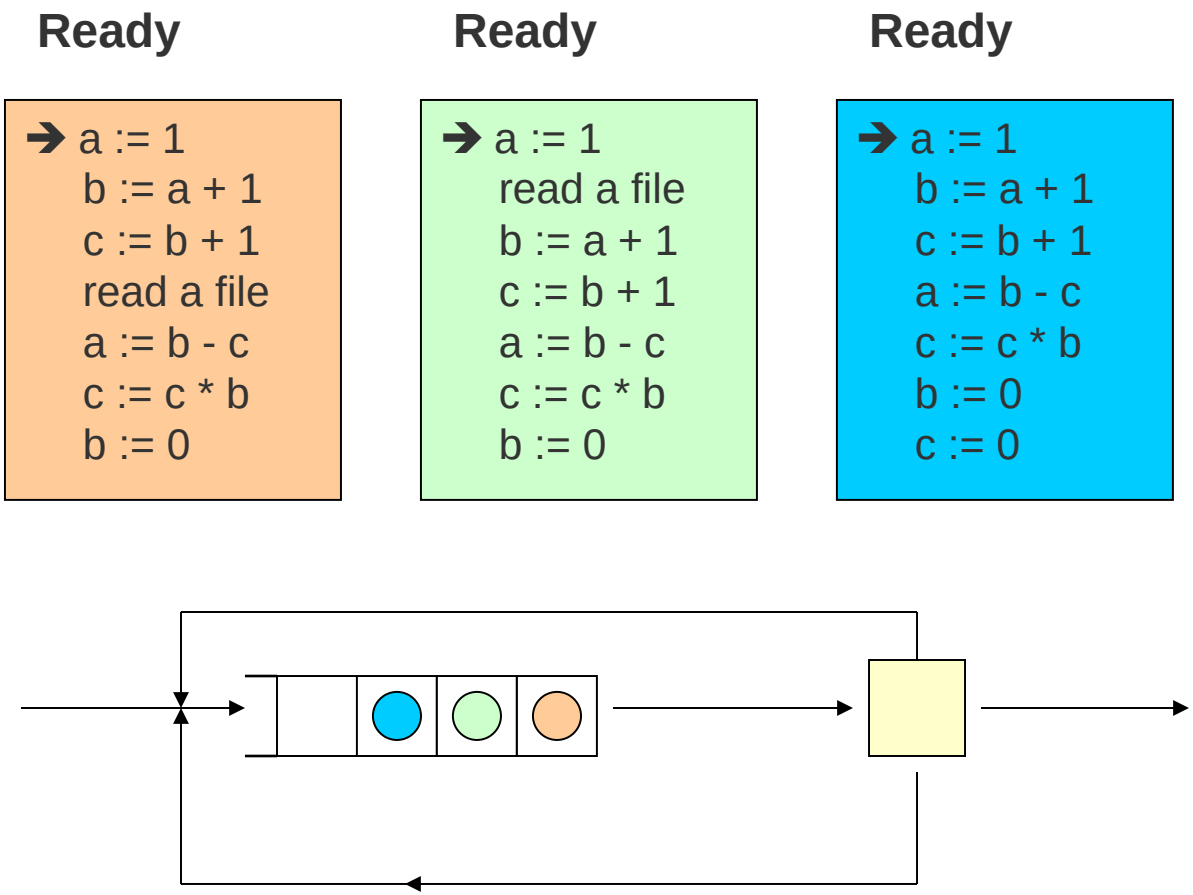
Two-state process model



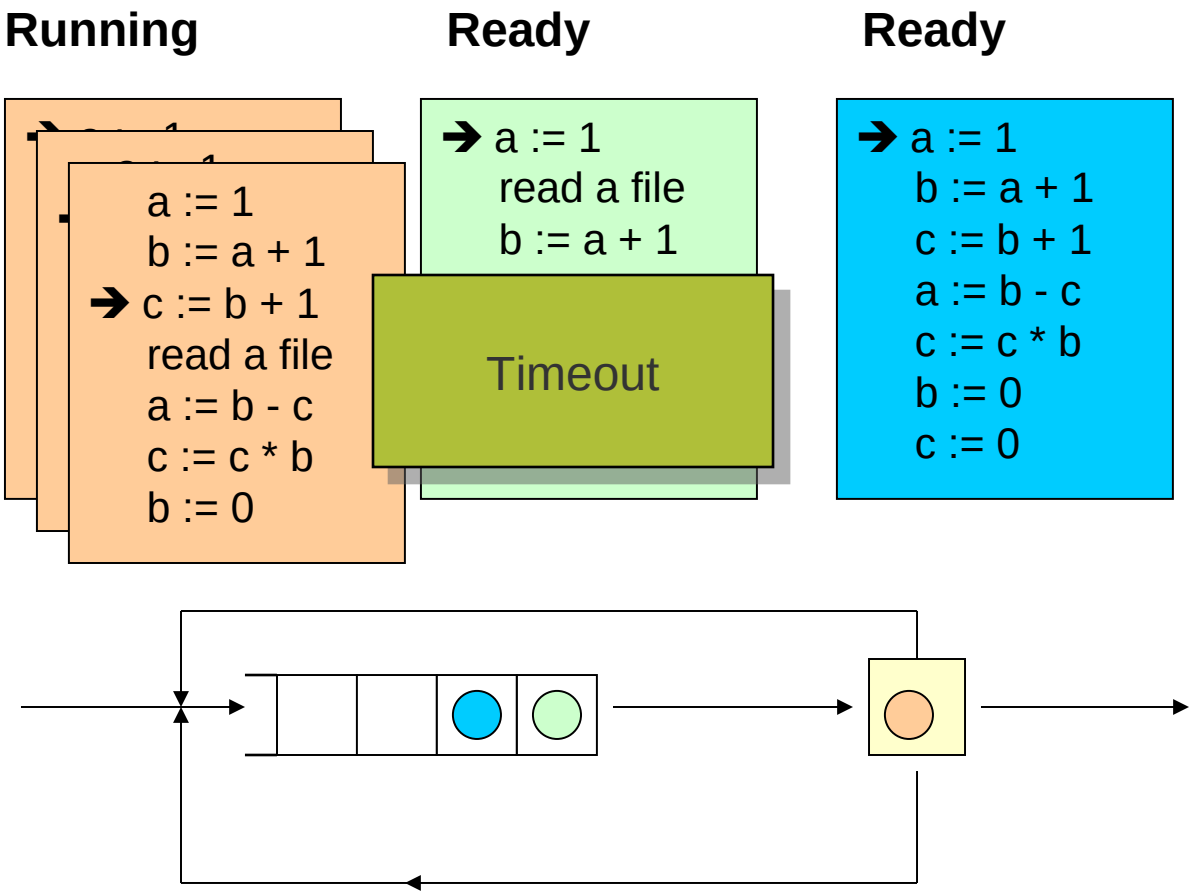
Dispatcher is now redefined:

- Moves processes to the waiting queue
- Remove completed/aborted processes
- Select the next process to run

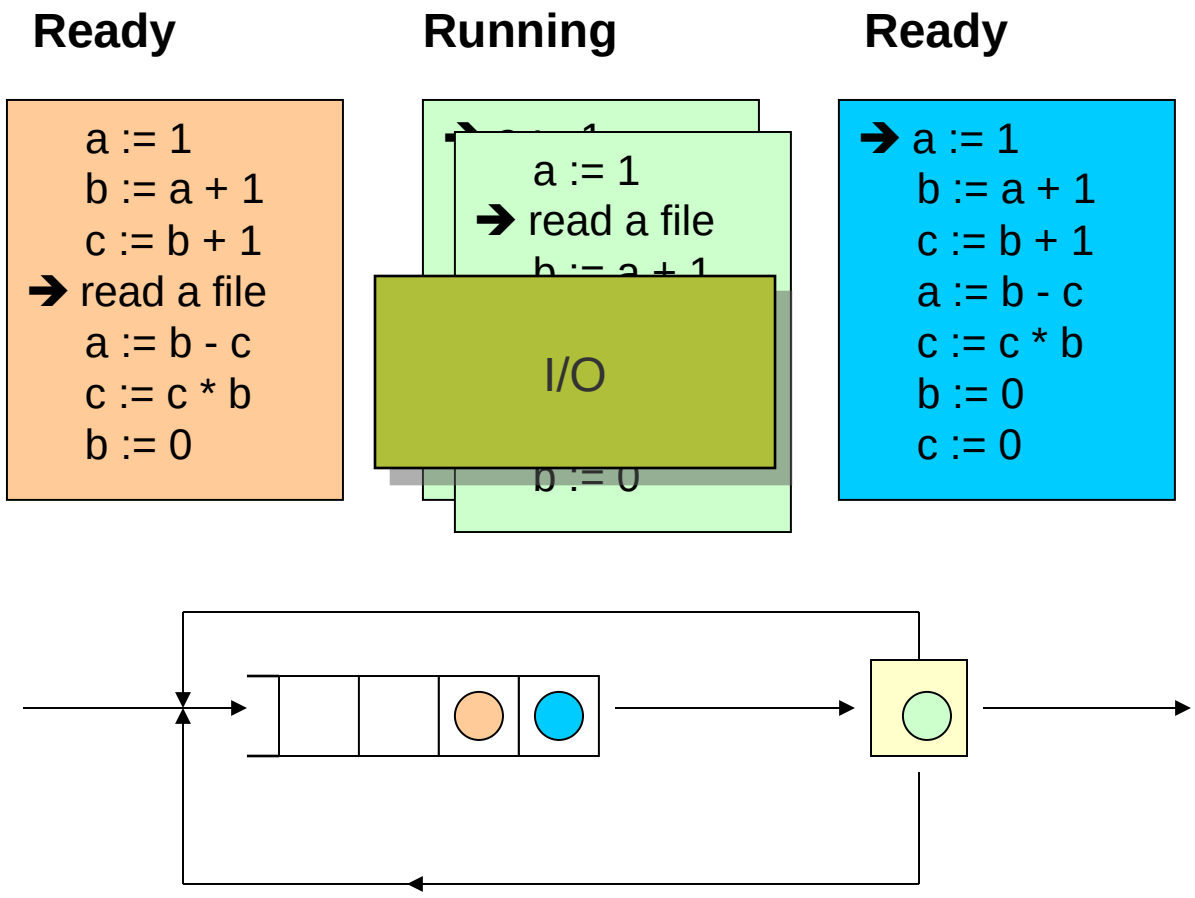
How process state changes1



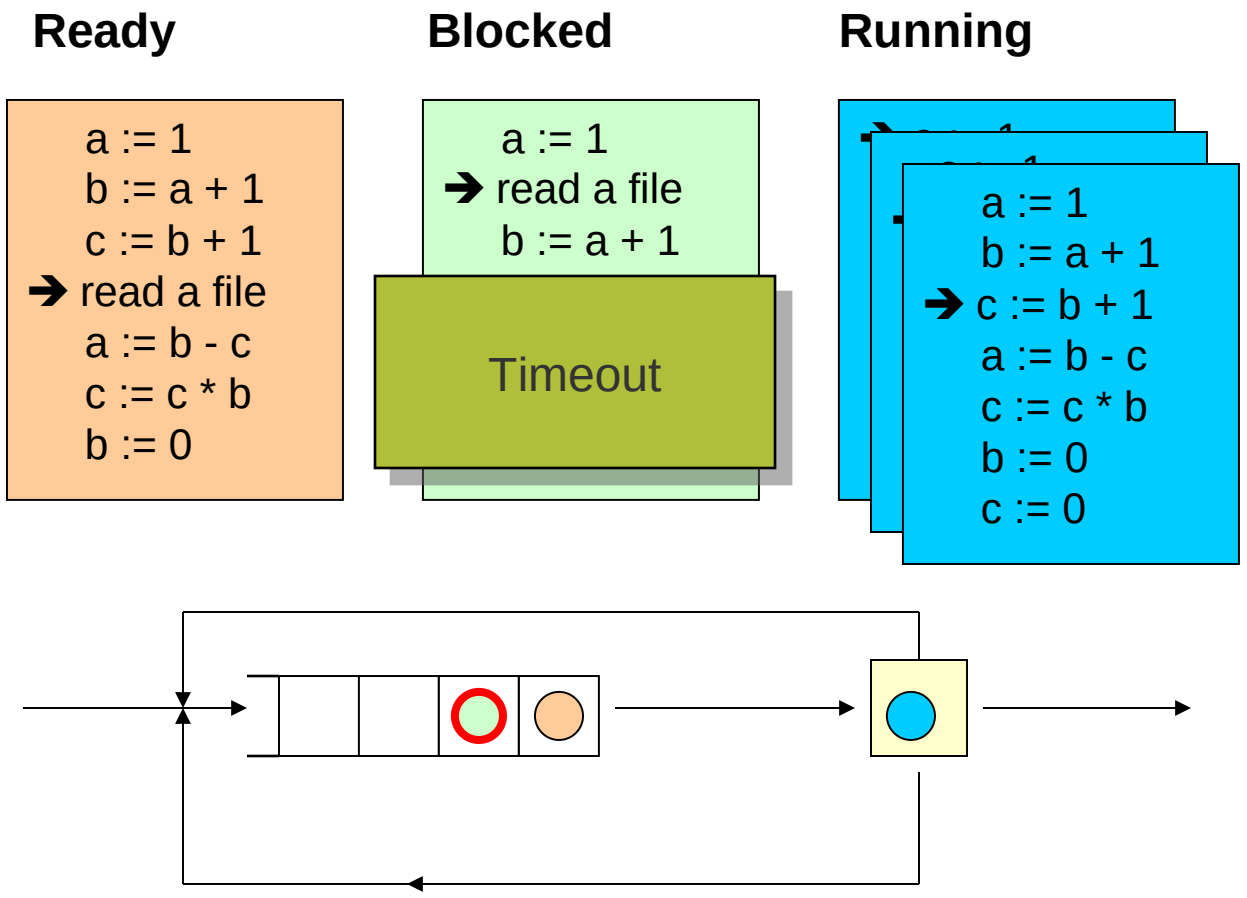
How process state changes2



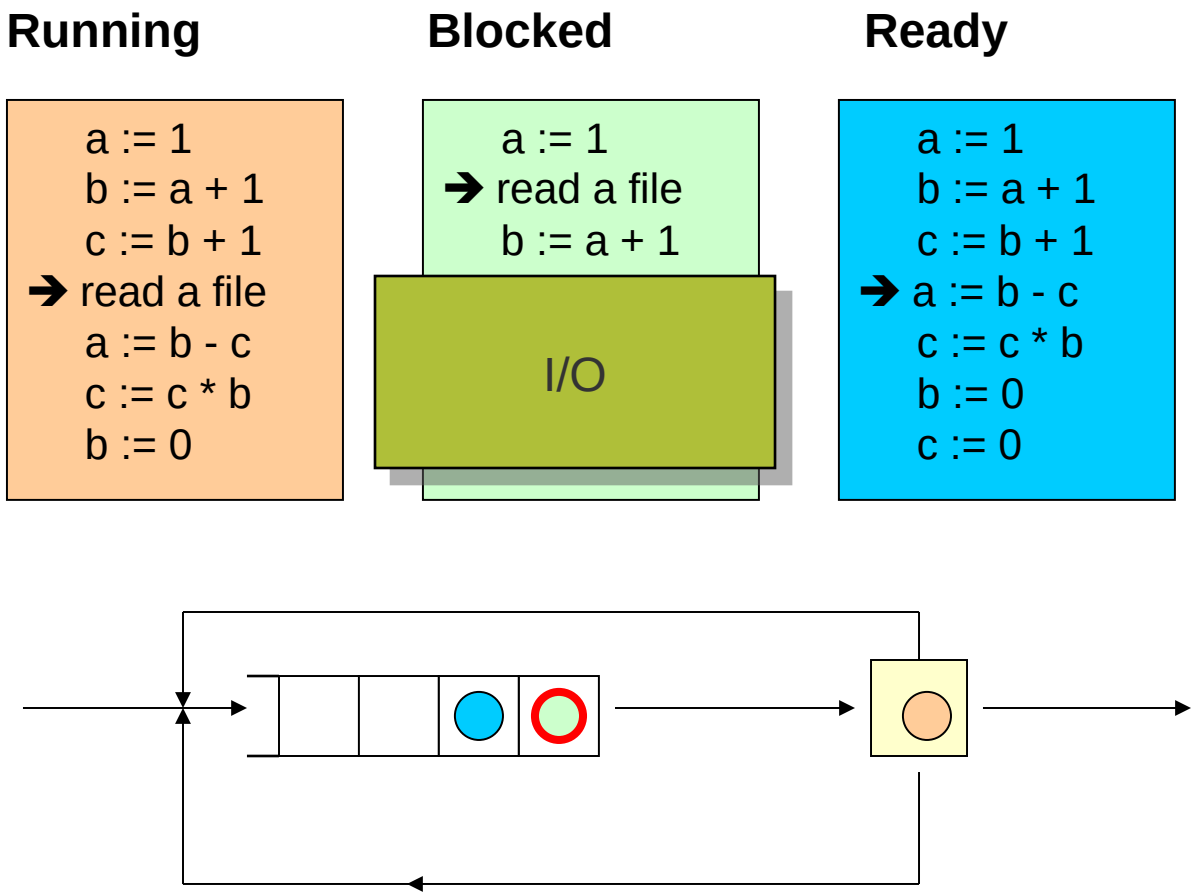
How process state changes3



How process state changes4



How process state changes5



How process state changes6

Blocked

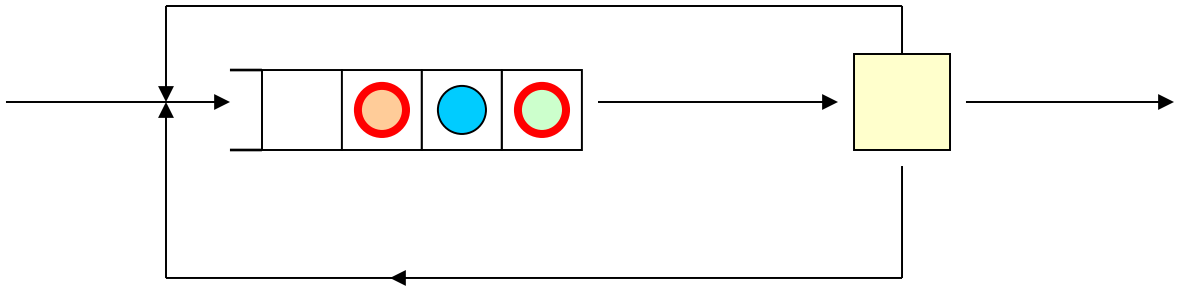
```
    a := 1
    b := a + 1
    c := b + 1
➔ read a file
    a := b - c
    c := c * b
    b := 0
```

Blocked

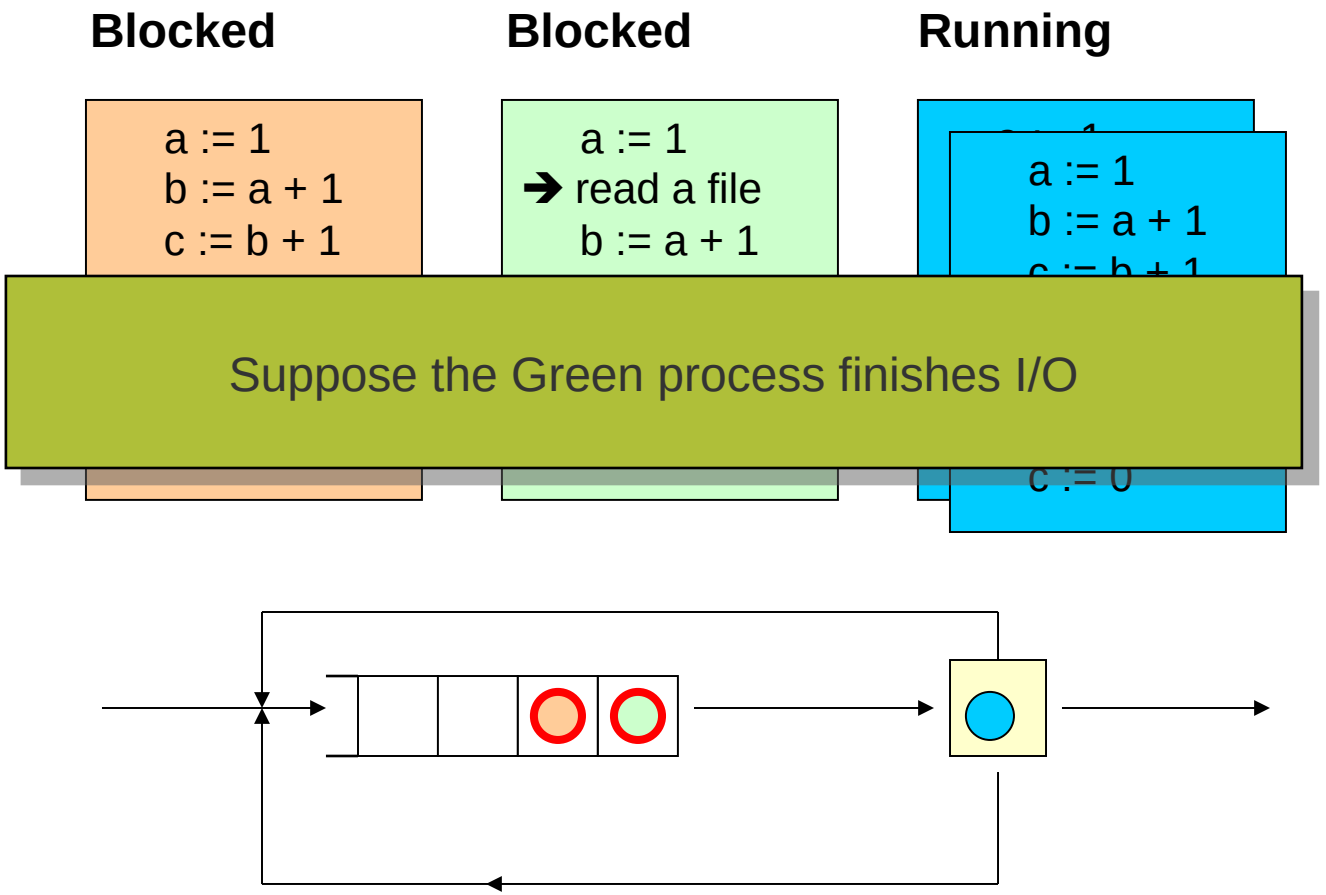
```
    a := 1
➔ read a file
    b := a + 1
    c := b + 1
    a := b - c
    c := c * b
    b := 0
```

The Next Process to Run cannot be simply selected from the front

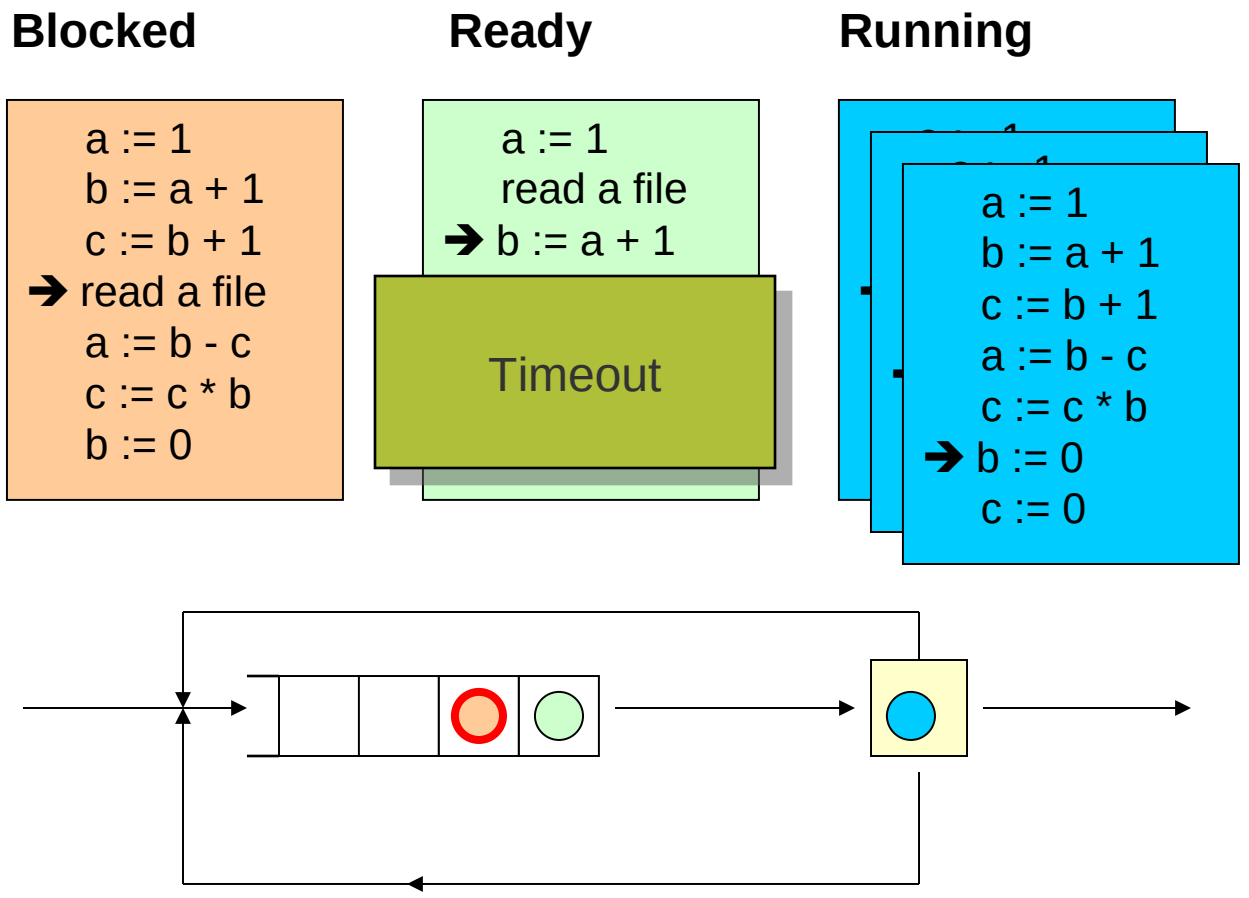
```
➔ a := b - c
    c := c * b
    b := 0
    c := 0
```



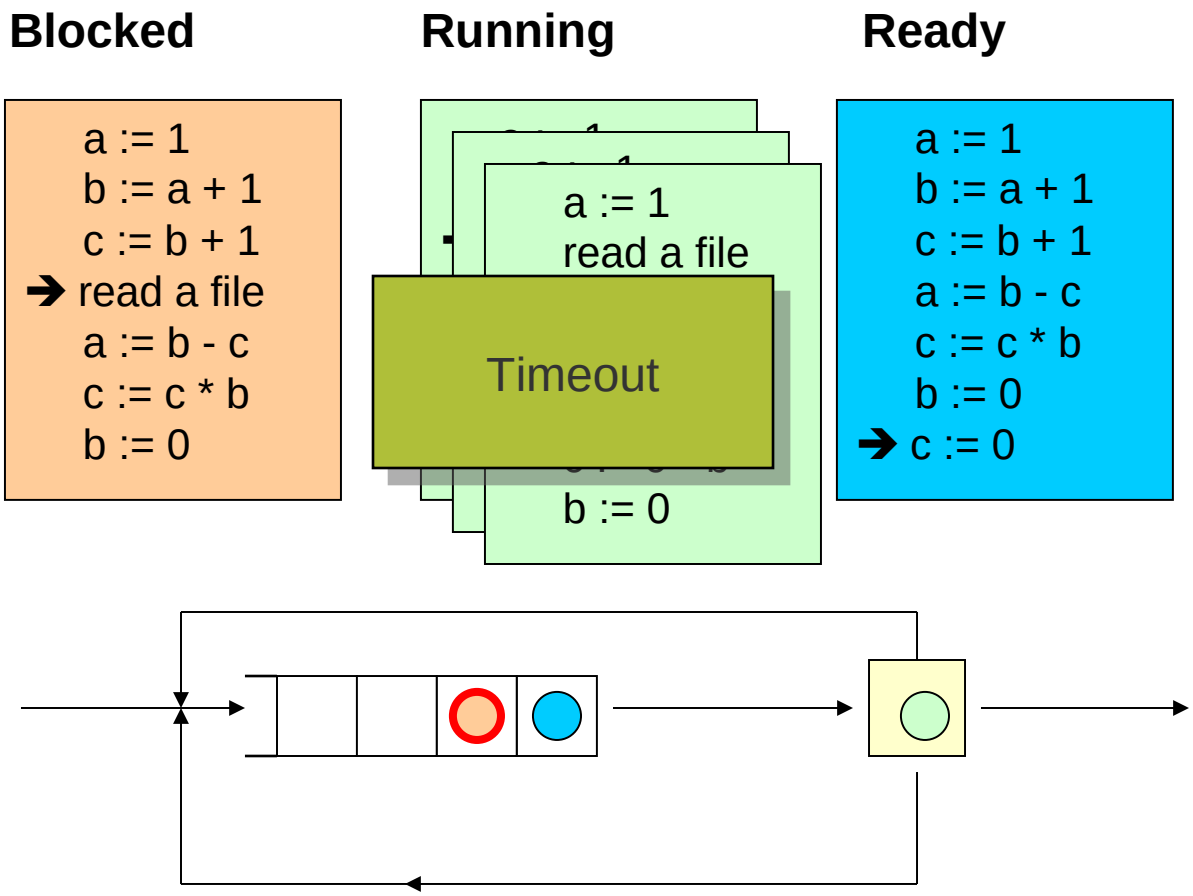
How process state changes7



How process state changes8



How process state changes9



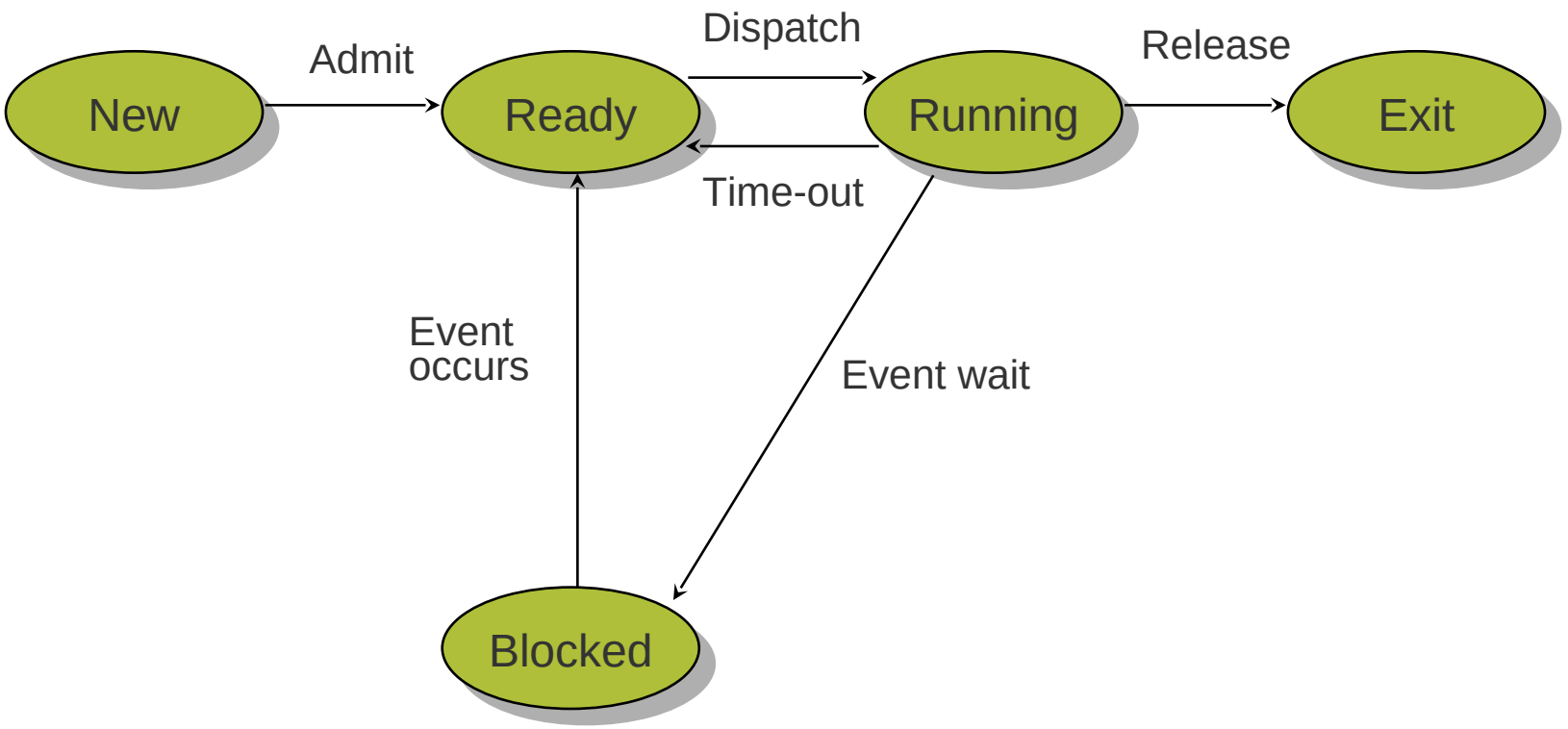
Problem in Two-state Process model

- A process may be waiting for I/O request
- A single queue for both the ready to run and waiting processes
- The dispatcher cannot simply select the process at the front, it can be a busy process
- In the worst case, it has to scan the whole queue to find the next process to run
- Solution?
- Split the Not Running state to:
  - Waiting
  - Ready

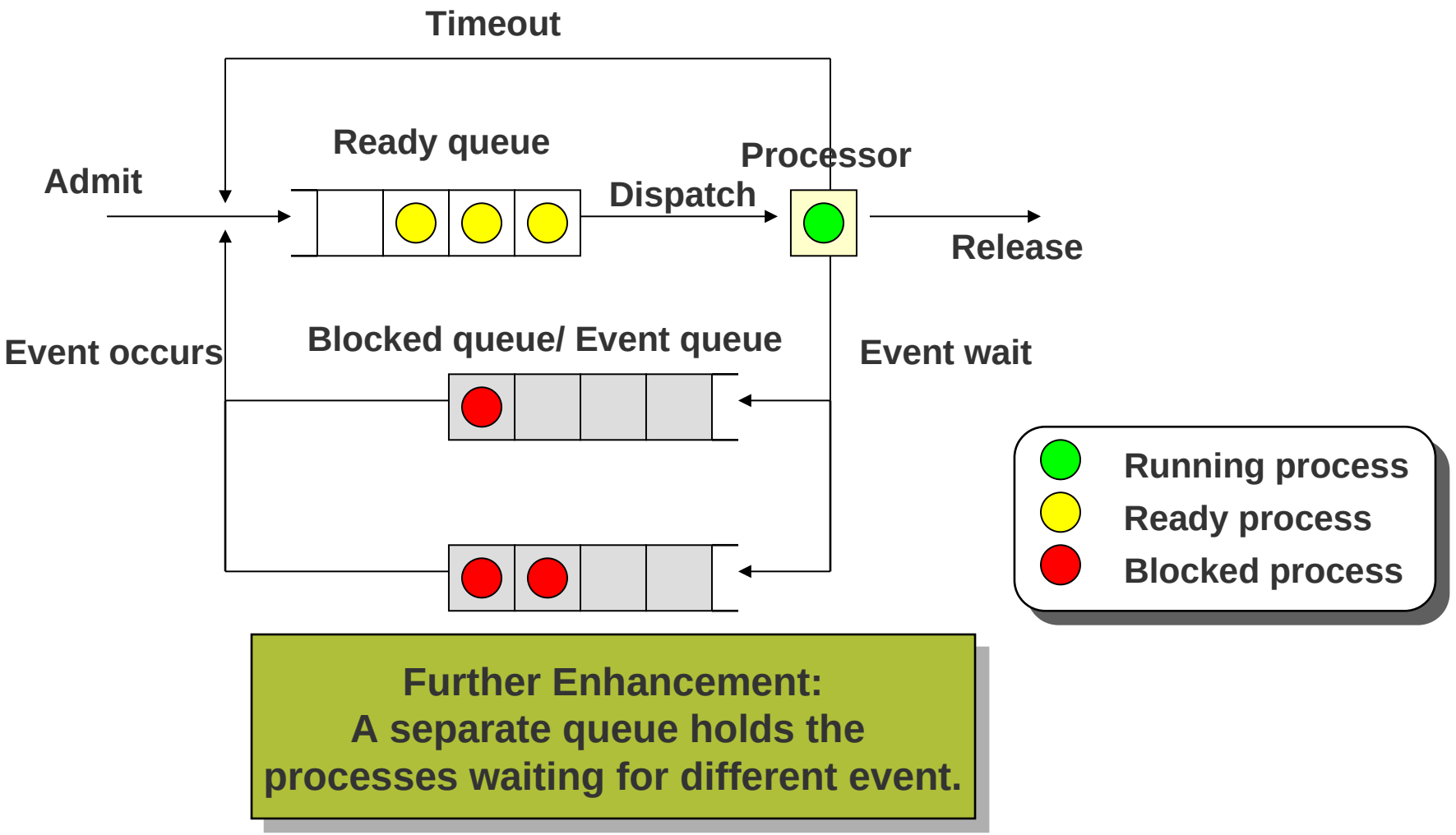
Five-state Process Model

- Running: currently being run
- Ready: ready to run
- Blocked: waiting for an event (I/O)
- New: just created, not yet admitted to set of run-able processes
- Exit: completed/error exit

Five-state Process Model



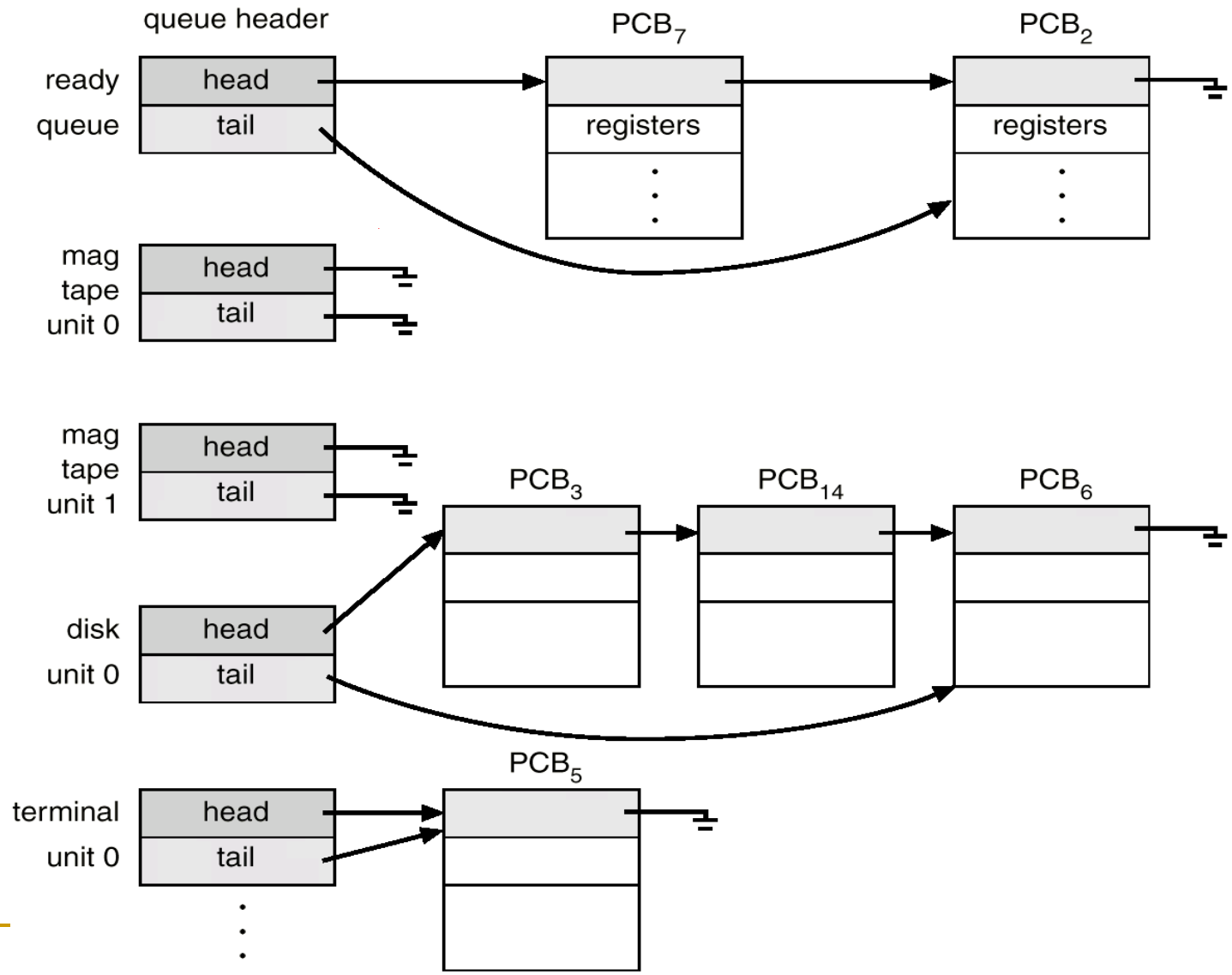
Blocked Queues



Scheduling Queues

- The queues are generally stored as linked lists
- A queue header points to the first and the final PCB's in the list
- We extend each PCB to include a pointer field that points to the next PCB in the ready queue

Scheduling Queues



Schedulers

- Short term Scheduler or CPU Scheduling
  - Which program is to be run next
- Long term Scheduler or Job Scheduler
  - Which ready jobs should be brought to memory
  - May need to invoke only when a process leaves the system
  - Must make a careful selection

Process Types

- Most processes can be described as either I/O bound or CPU bound
- I/O bound:
  - Spends more of its time doing I/O than doing computations
- CPU bound:
  - Spends more of its time doing computations than doing I/O
- If all processes are I/O bound,
  - The ready queue will almost always be empty
- If all processes are CPU bound,
  - The I/O waiting queue will almost always be empty, devices will go unused
  - System will again be unbalanced

Process Types

- Sometimes OS may swap a blocked process to disk to free up more memory
- Or to improve process mix
- This is called **Swapping**