Operating Systems
CS220

Lecture 5

Process Management

19th April 2021

By: Dr. Rana Asif Rehman

Process

- A computer program in execution on a machine is a process

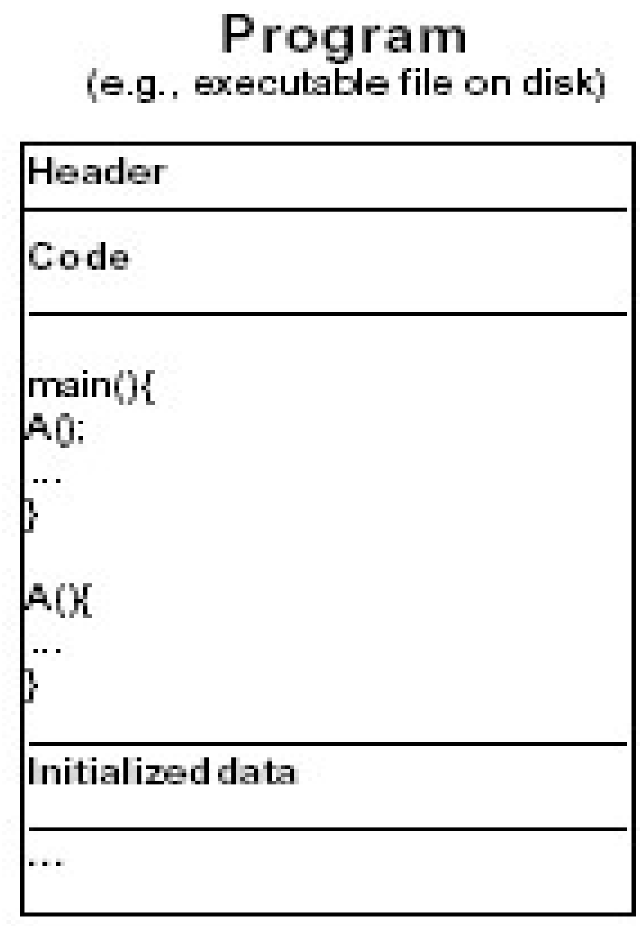
- More formally:
  - A Sequential stream of Execution in its own address space

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



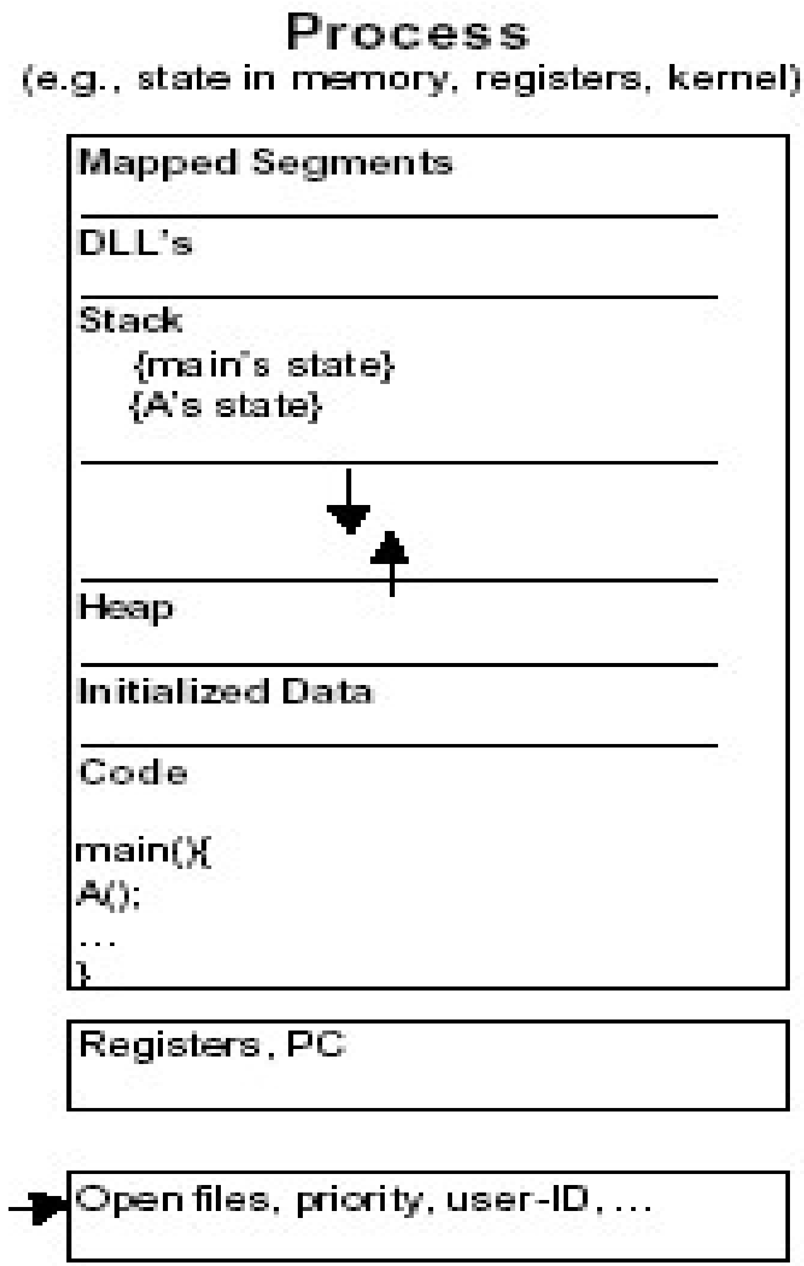
- Program: series of commands (e.g. C statements, assembly commands, shell commands)

Process =? Program

- Anatomy of a process
  - More to a process than just a program
    - program is part of process state
    - I run dir, you run dir – same program, different processes
  - Less to a process than a program
    - A program can invoke more than one process to get the job done

Process

- A process consists of
  - Code (text) section
  - Data section
  - Stack
  - Heap
  - CPU State (program counter, etc.)
  - Environment
  - Process control block (PCB)


Process
(e.g., state in memory, registers, kernel)
Mapped Segments
DLL's
Stack
{main's state}
{A's state}
Heap
Initialized Data
Code
main(){
A();
...
}
Registers, PC
Open files, priority, user-ID, ...

CPU State

- CPU registers contain the current state
  - Program Status Word (PSW): includes bits
  - Instruction Register (IR):
  - Program Counter (PC):
  - Stack Pointer (SP):
  - General purpose registers:

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
  - Data
  - Heap
  - Stack
- 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 (PCB)

- 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 Control Block (PCB)

- PCB is "the manifestation of a process in an operating system”.

- **Data Structure** defined in the operating system kernel containing the information needed to manage a particular process.

- It must be kept in an area of memory protected from normal user access.

Process Control Block (PCB)

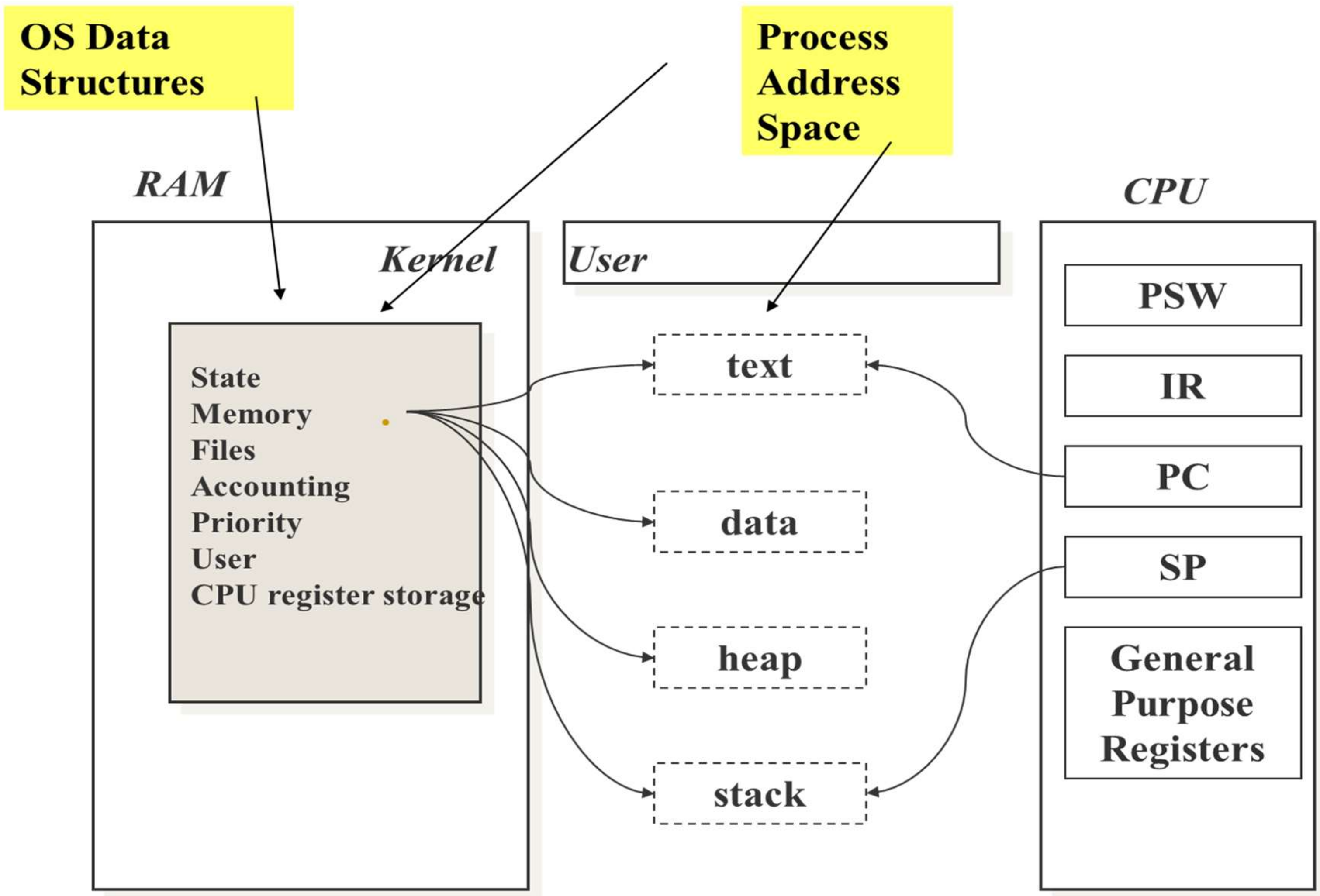
- PCB contains the Process information and attributes
  - Process state
  - Program counter
  - CPU registers
  - CPU scheduling information
  - Memory management information
  - Accounting information
  - I/O status information
  - Per process file table
  - Process ID (PID)
  - Parent PID, etc.

Process Identification

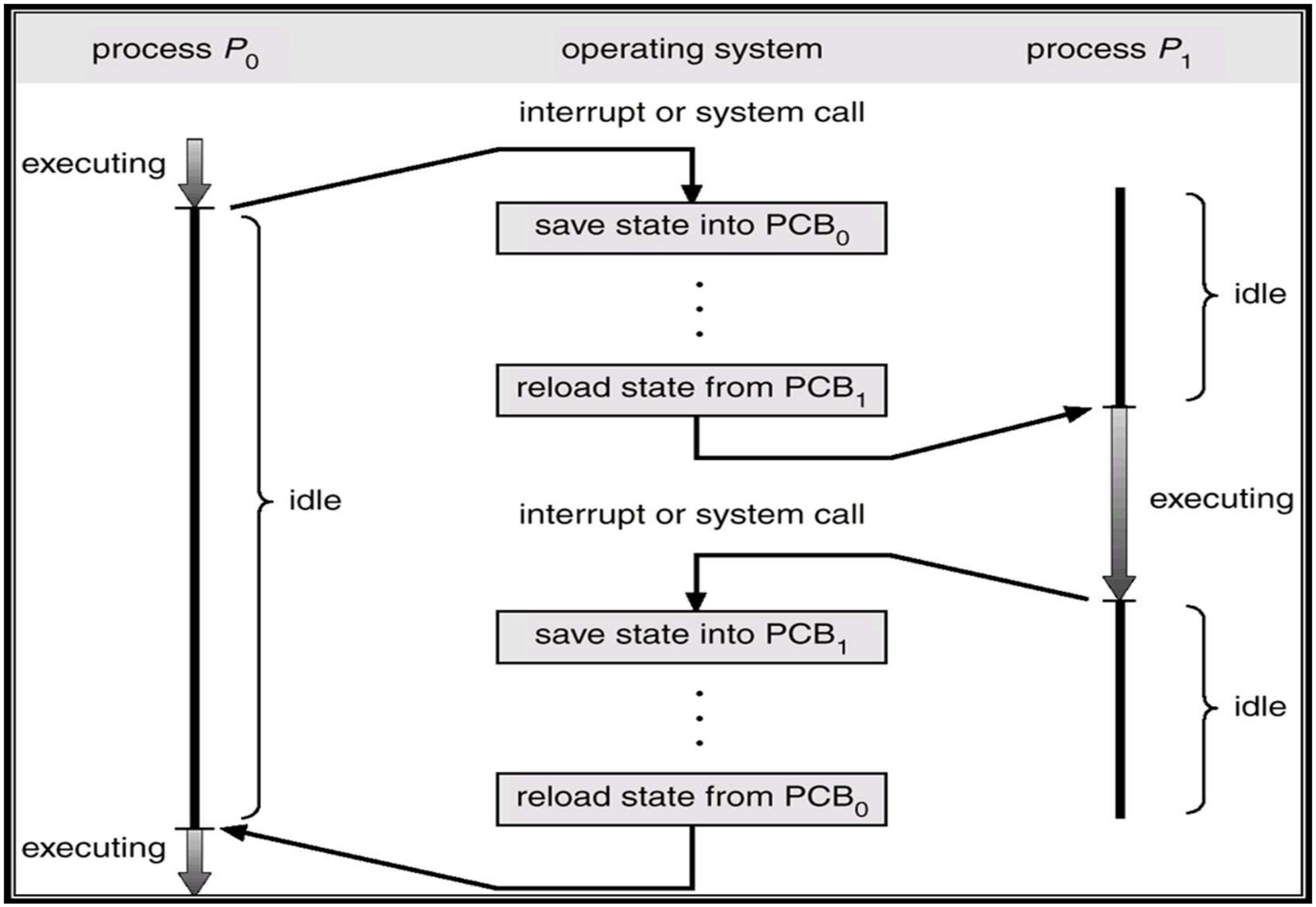
- Process ID, a unique numeric identifier

- User ID
  - Who runs the process. Why?
  - Used to determine what access rights the process has

Process Control Block



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 in Memory

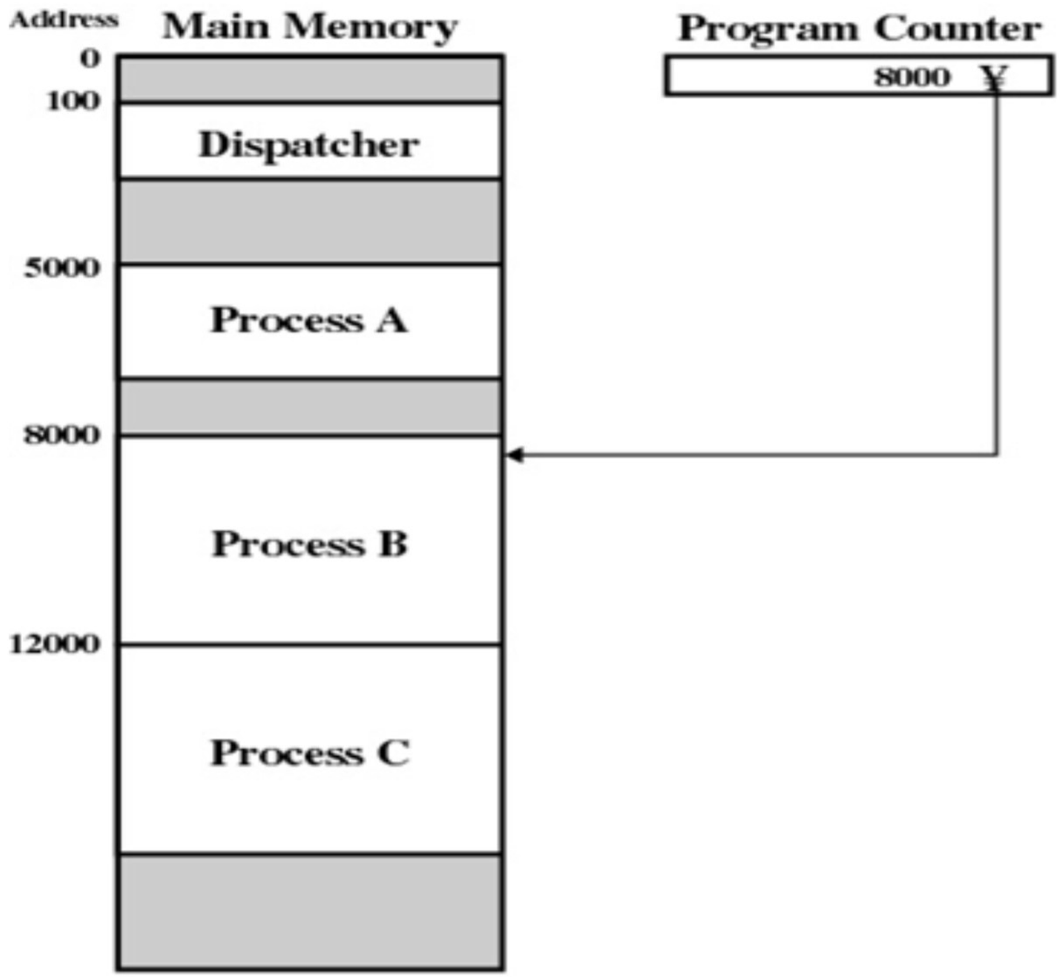
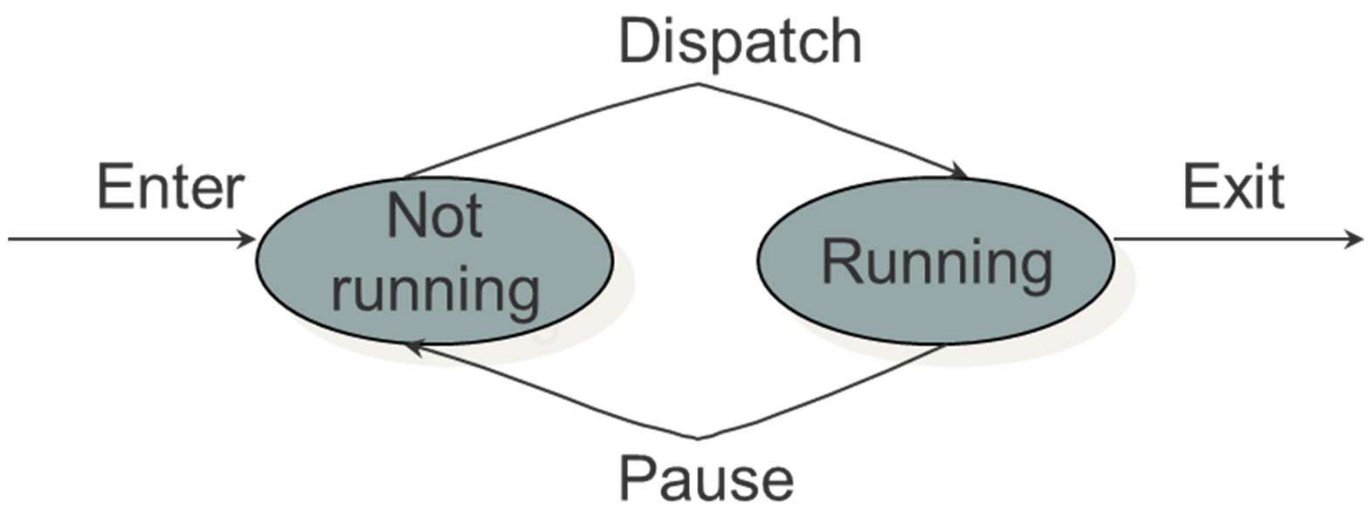


Figure 3.1 Snapshot of Example Execution

Process States

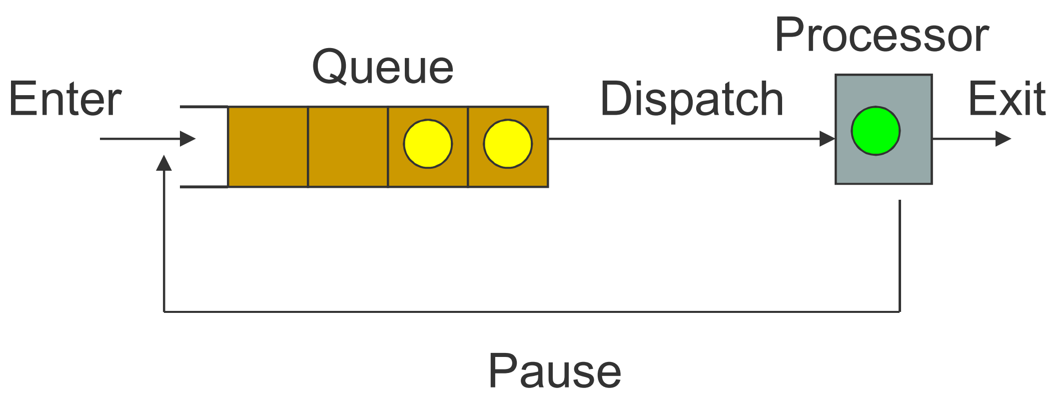
- At any given time a process is either running or not running
- Number of states
  - Running
  - Not Running
- When the OS creates a process, the process is entered into Not Running state

Two-state process model



- 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 changes_1

Ready

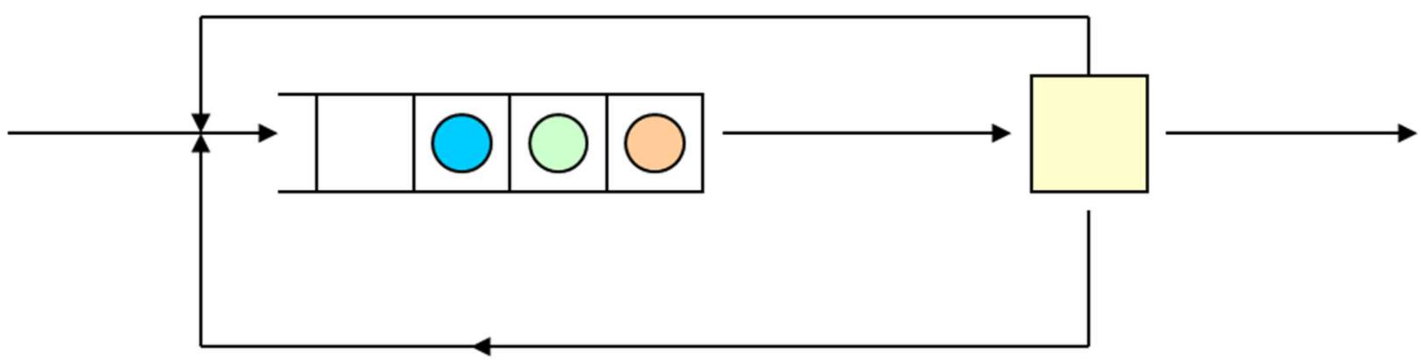
```
→ a := 1
  b := a + 1
  c := b + 1
  read a file
  a := b - c
  c := c * b
  b := 0
```

Ready

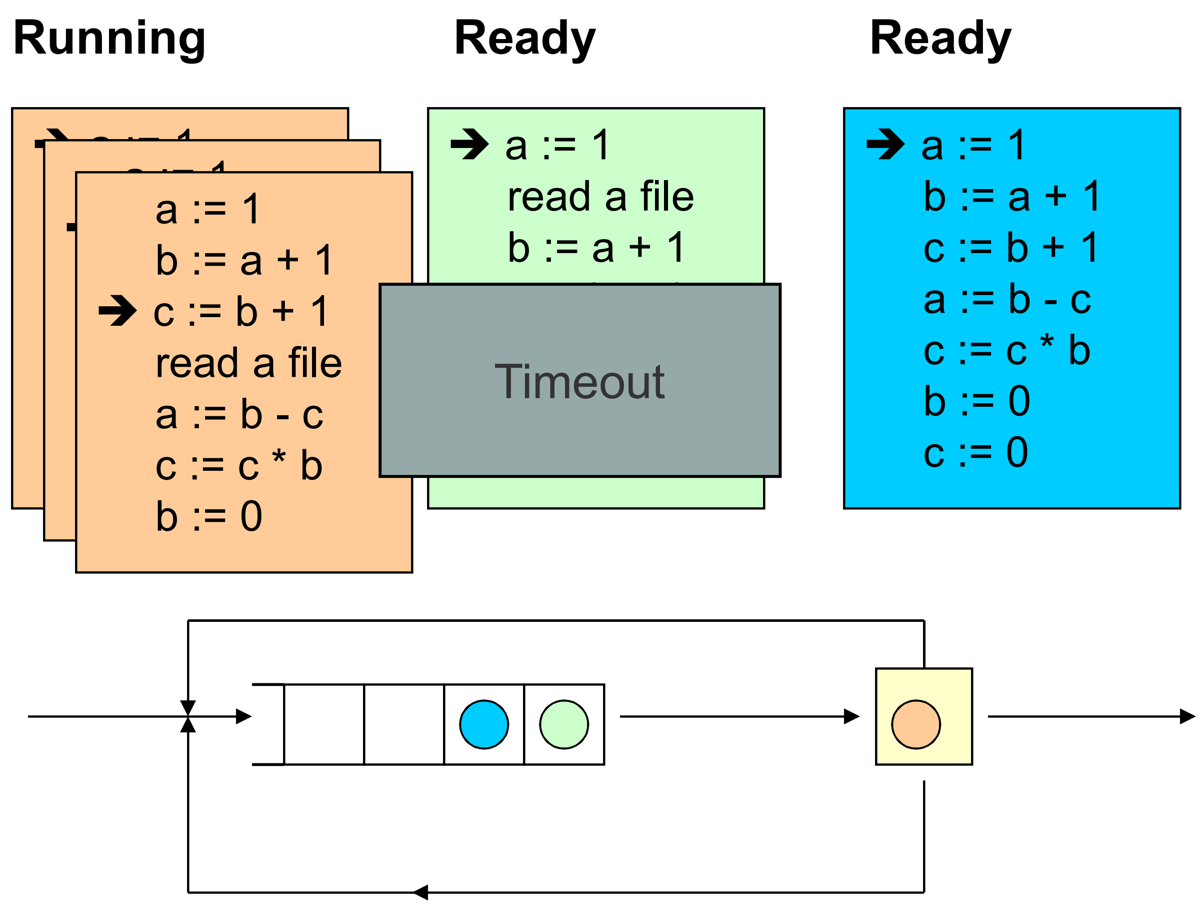
```
→ a := 1
  read a file
  b := a + 1
  c := b + 1
  a := b - c
  c := c * b
  b := 0
```

Ready

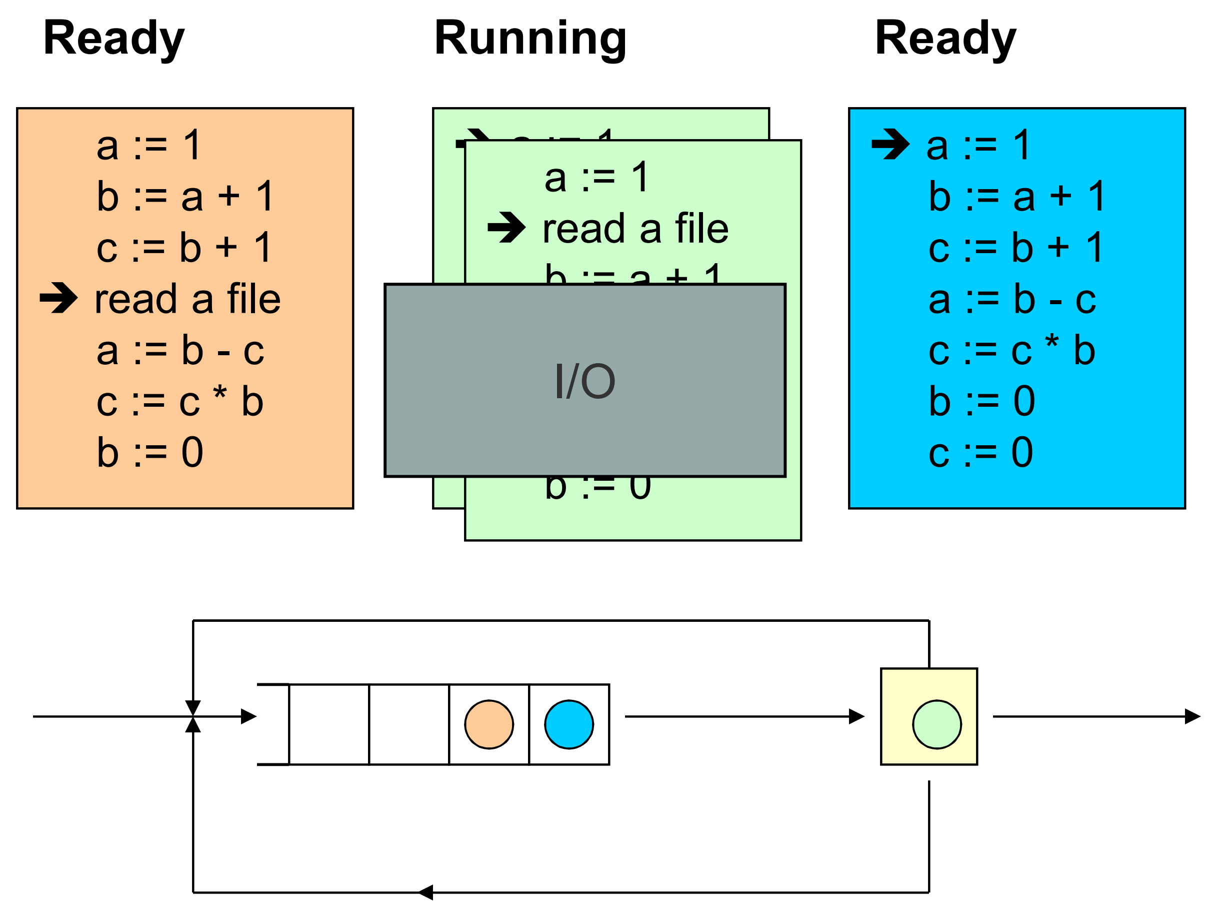
```
→ a := 1
  b := a + 1
  c := b + 1
  a := b - c
  c := c * b
  b := 0
  c := 0
```



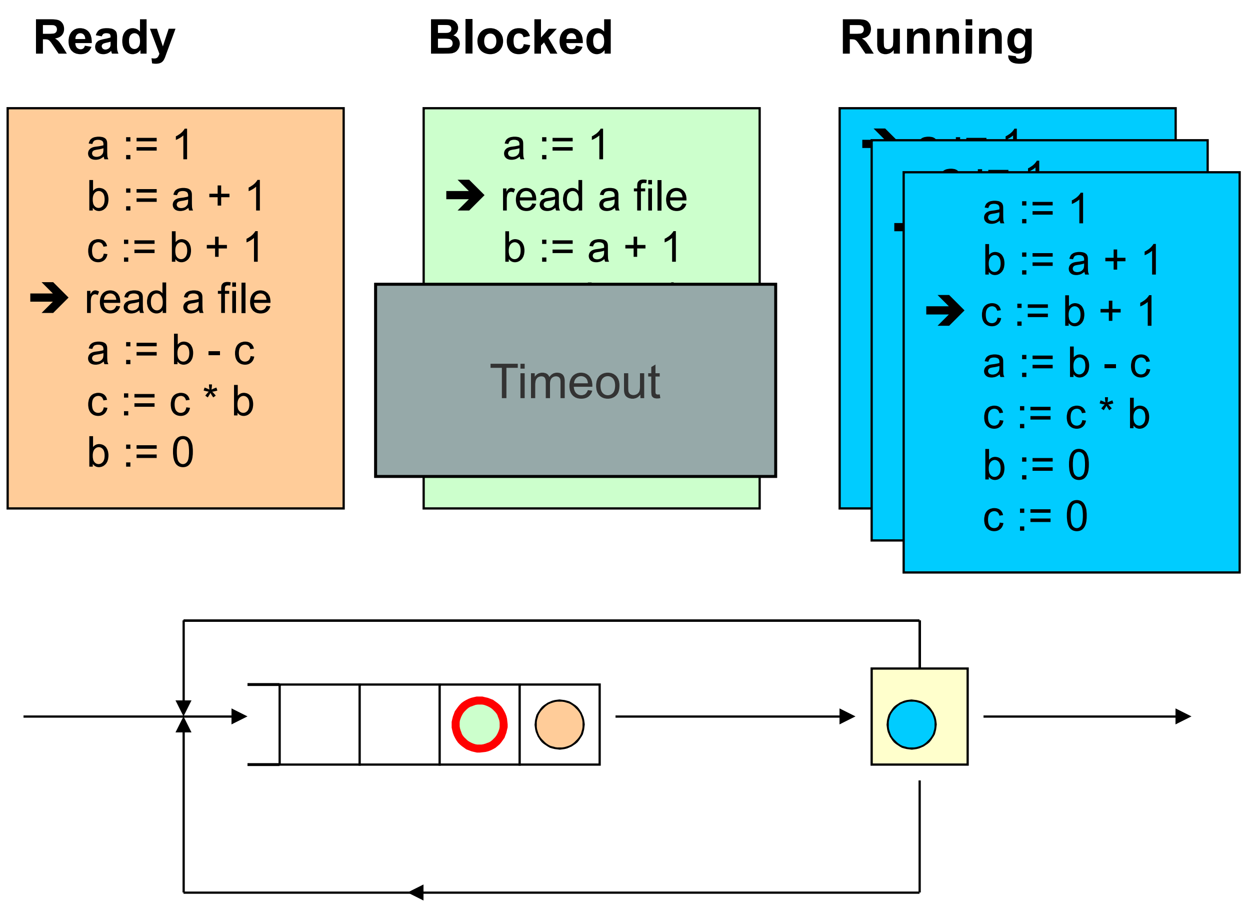
How process state_2



How process state changes_3



How process state changes_4



How process state changes_5

Running

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

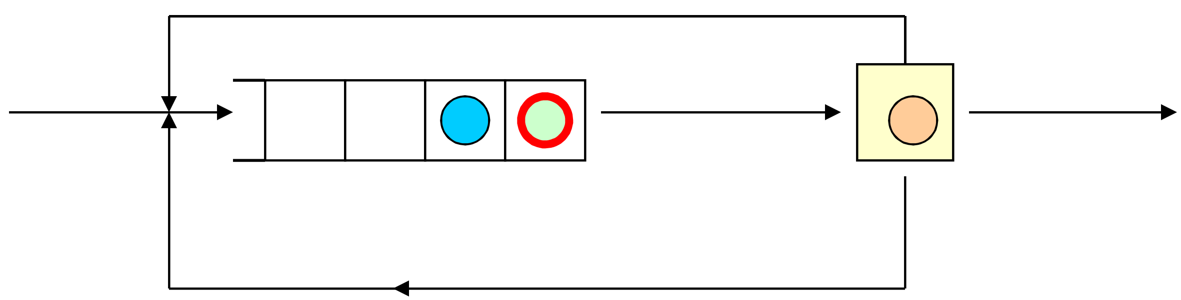
Blocked

```
   a := 1
➔ read a file
   b := a + 1
```

I/O

Ready

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



How process state changes_6

Blocked

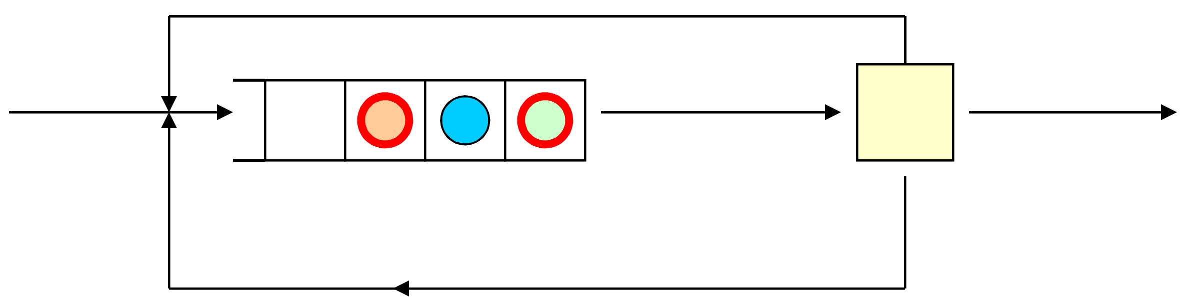
```
    a := 1
    b := a + 1
    c := b + 1
➔  read a file
    a := b - c
    c := c * b
    b := 0
```

Blocked

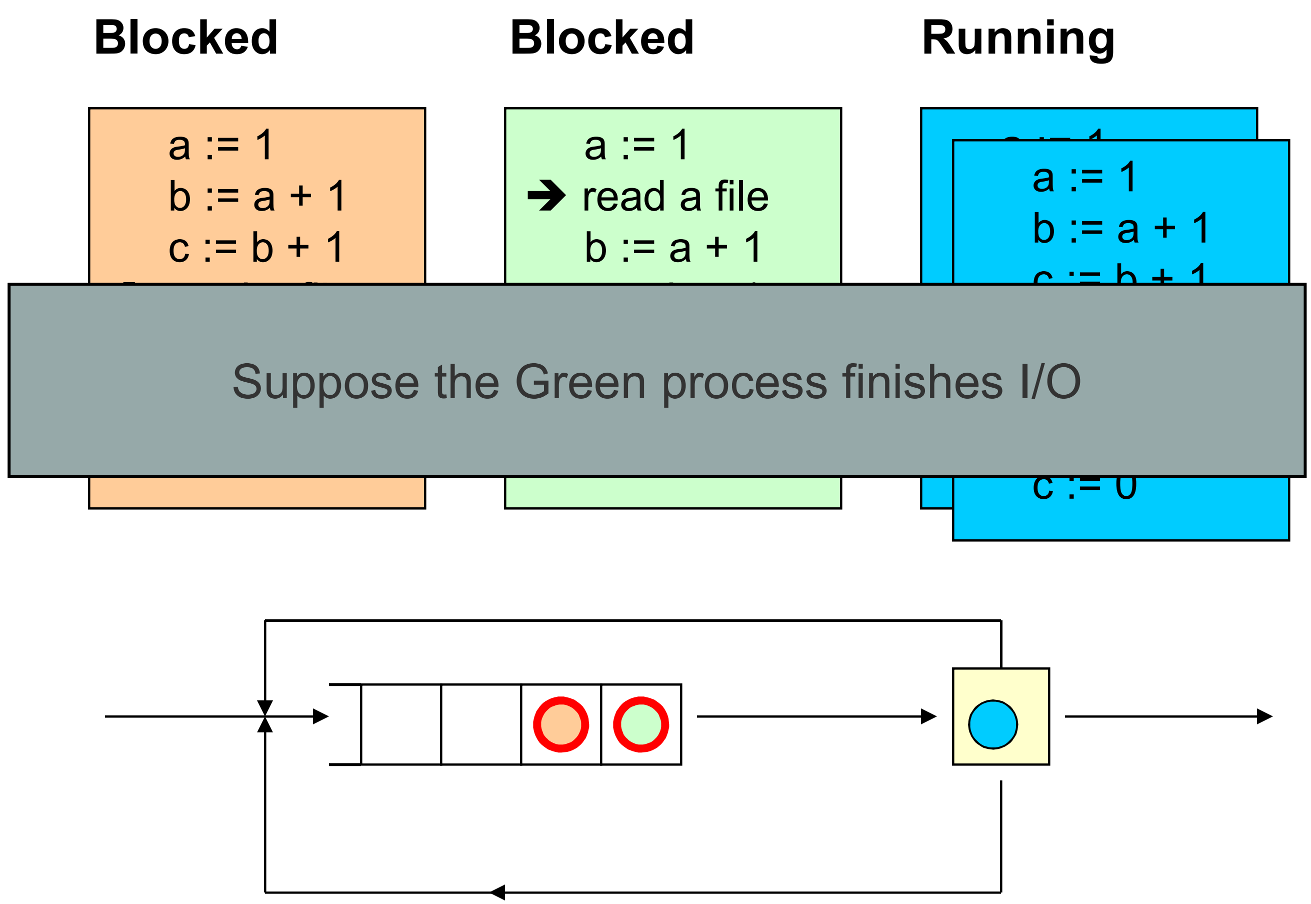
```
    a := 1
➔  read a file
    b := a + 1
    c := b + 1
    a := b - c
    c := c * b
    b := 0
```

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

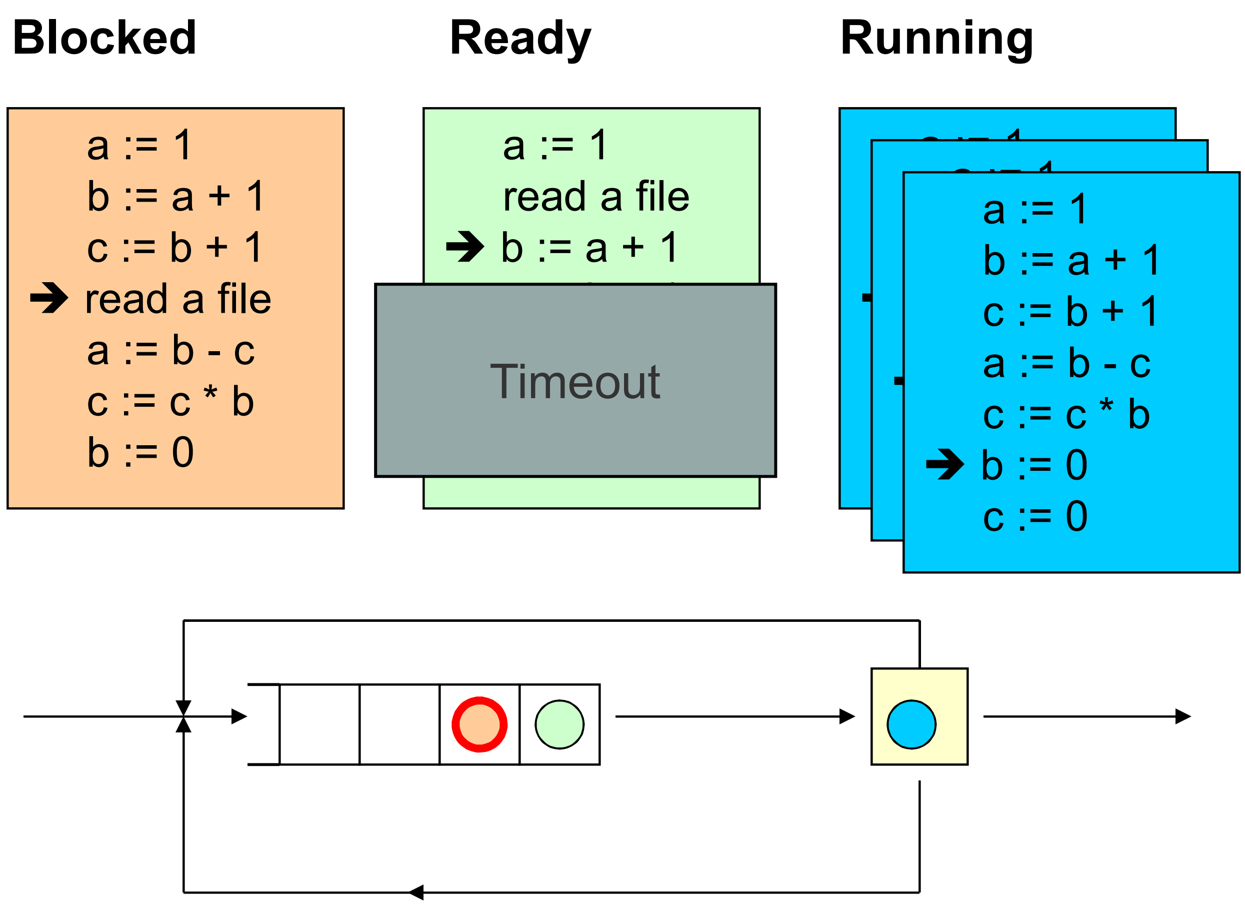
```
➔  a := b - c
    c := c * b
    b := 0
    c := 0
```



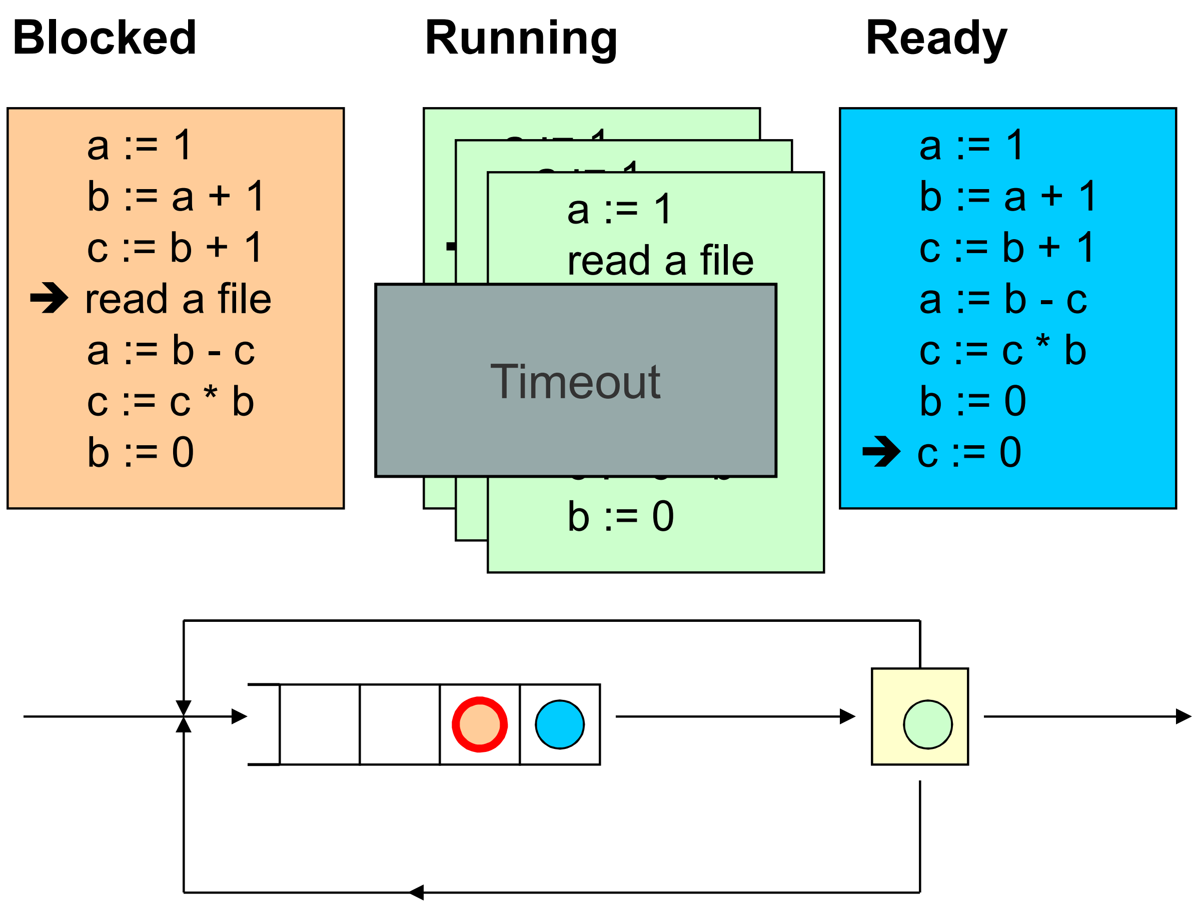
How process state changes_7



How process state changes_8



How process state changes_9



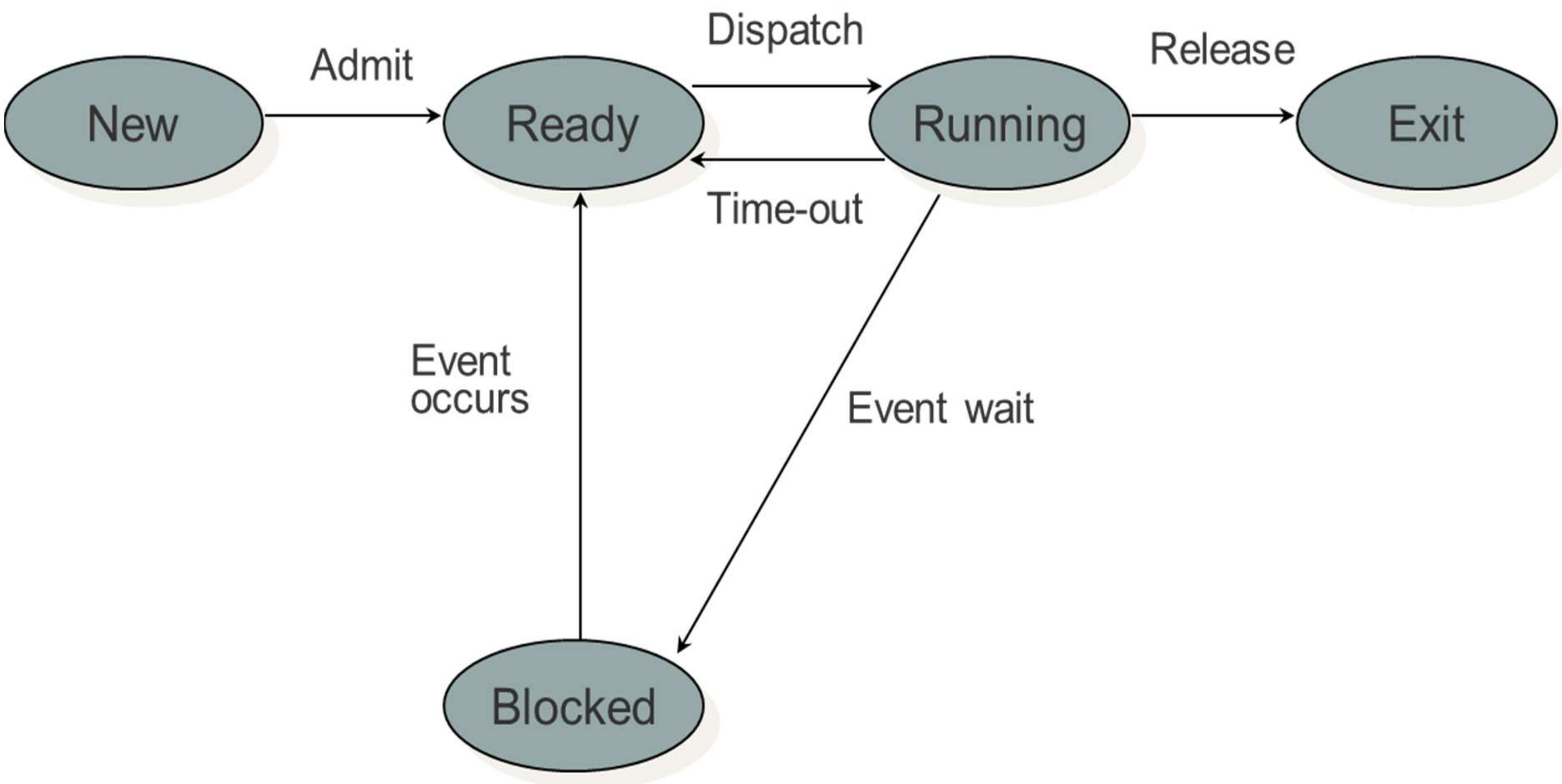
Problem in Two-state Process model

- 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

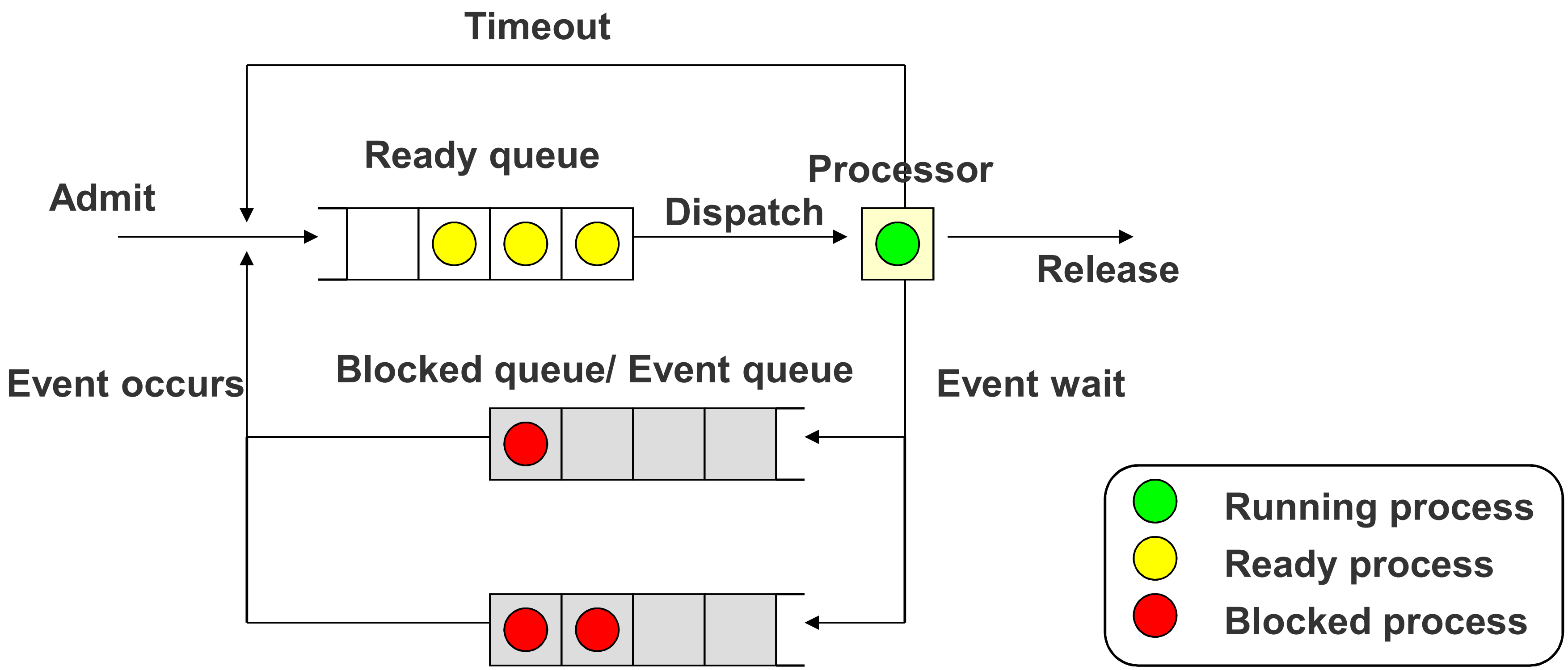
Five-state Process Model

- 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

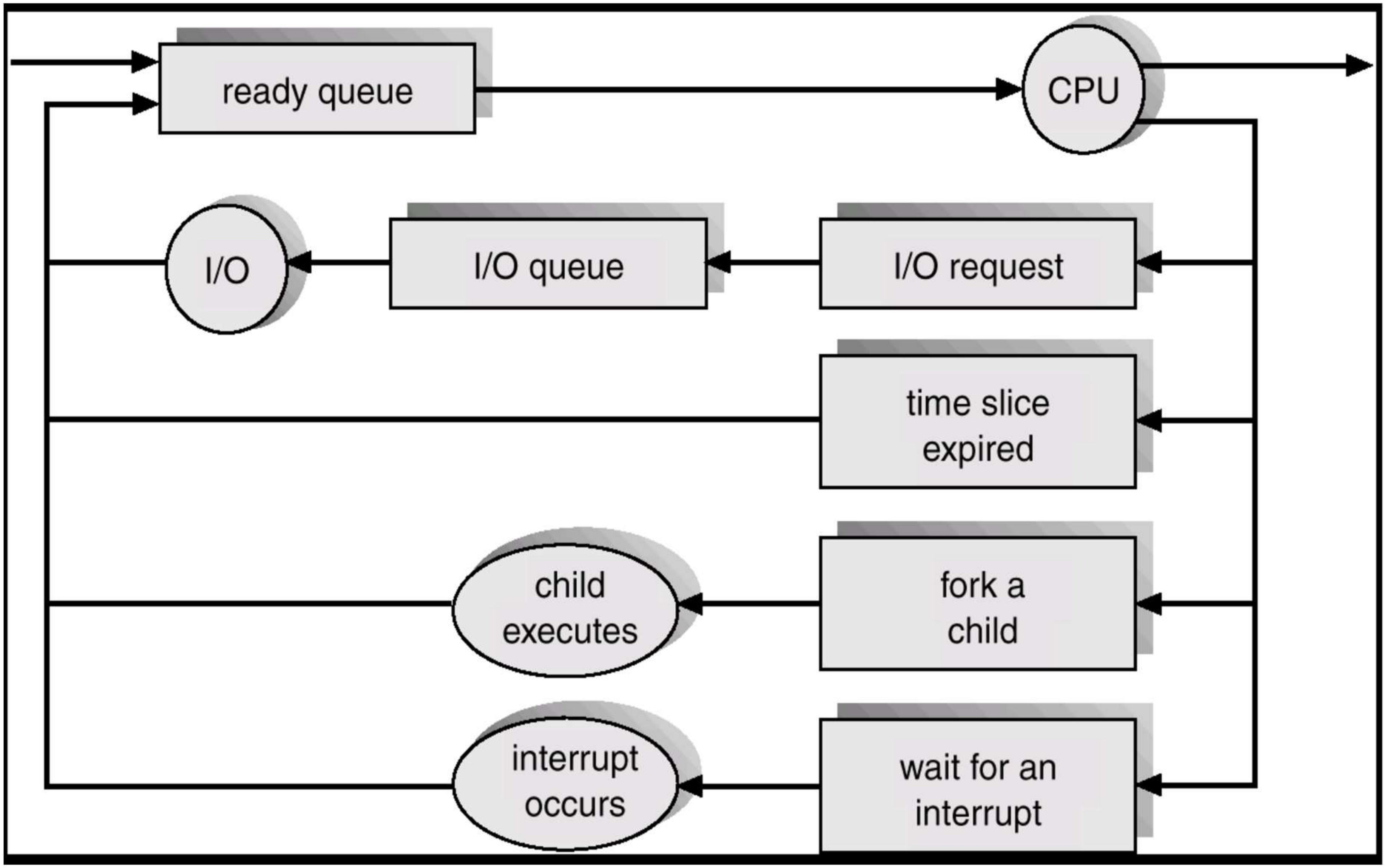


Further Enhancement:
A separate queue holds the processes waiting for different event.

Scheduling Queues

- Job queue – set of all processes in the system.
- Ready queue – set of all processes residing in main memory, ready and waiting to execute.
- Device queues – set of processes waiting for an I/O device.
- Process migration between the various queues.

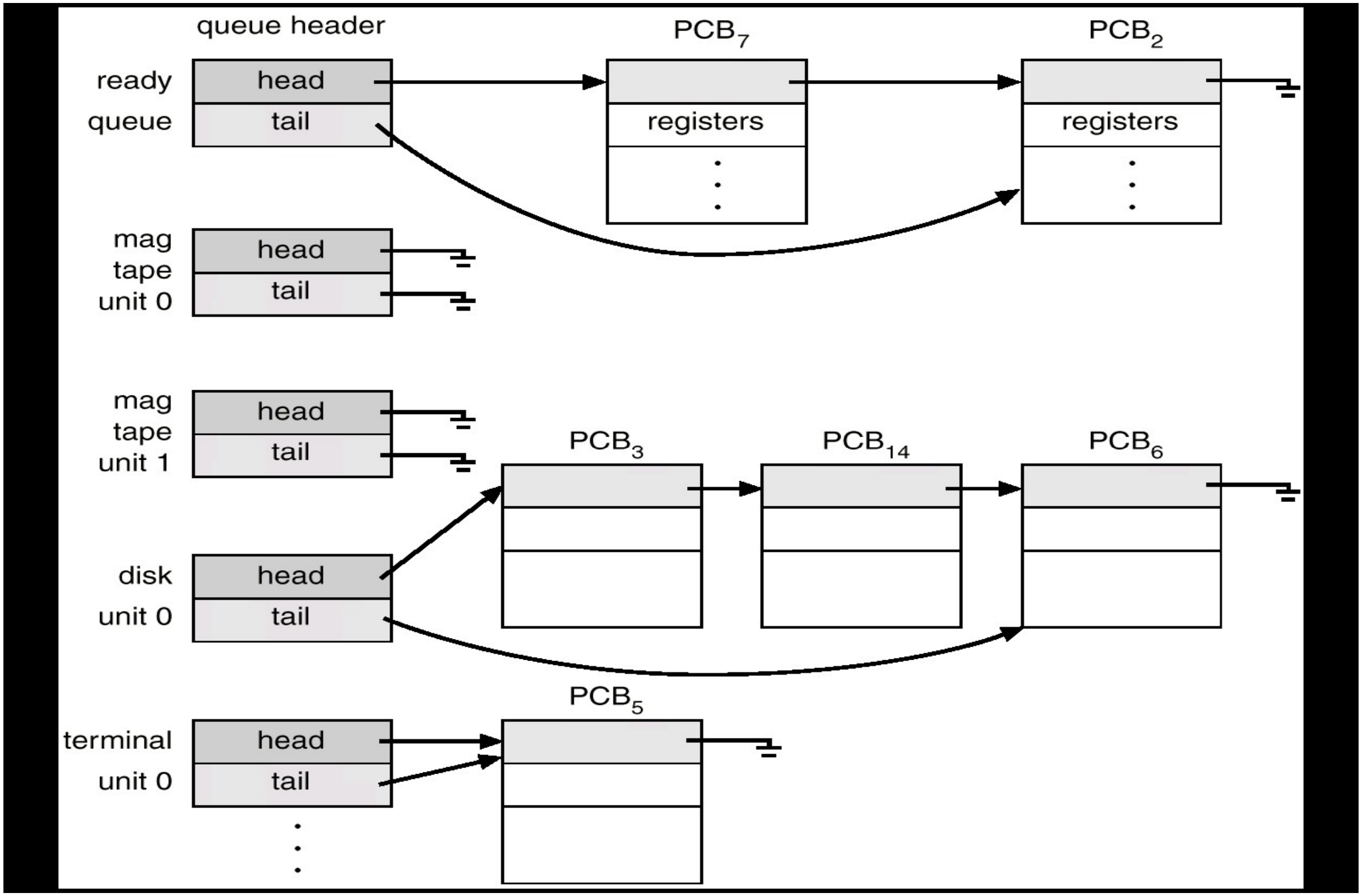
Process Scheduling View



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



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

Schedulers

- Chooses a ready process and activates it
- Policy – which process to choose
- Choice of policy depends on goals:
  - Maximize throughput – avoid idle time and overhead
  - Fairness – all jobs get same service
  - Minimum average waiting time – good service to some
  - Good service to “important” jobs

Schedulers

- Short term Scheduler or CPU Scheduling
- Long term Scheduler or Job Scheduler
- Medium Term Scheduler

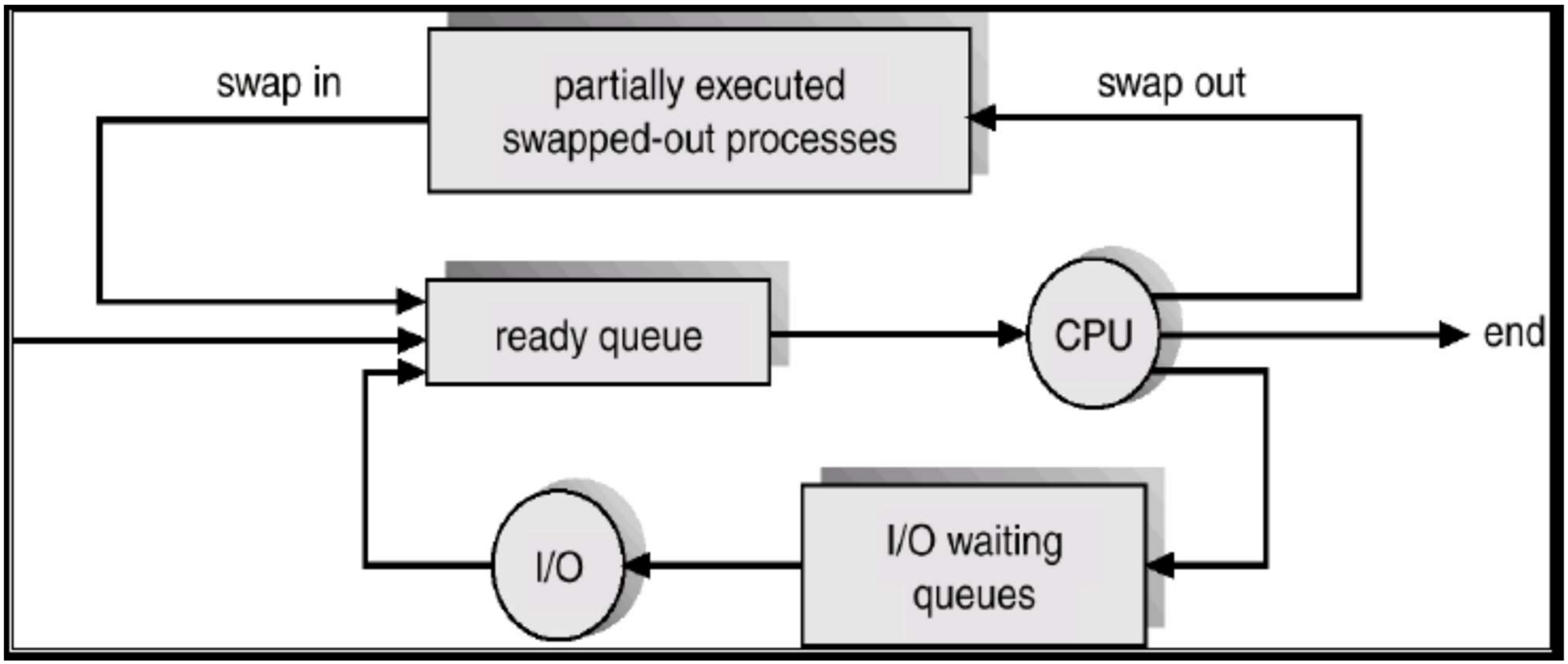
Schedulers

- 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

Schedulers

- Sometimes OS may swap a blocked process to disk to free up more memory
- Or to improve process mix
- This is also called **Swapping**
- This scheduler responsible for this task is known as **Intermediate or Medium Term Scheduler**

Addition of Medium Term Scheduling



Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ Design Objective? (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slower)
- The long-term scheduler controls the degree of *multiprogramming*

Policies without Priorities

- FIFO – run in order until complete or blocks
  - Non-pre-emptive (favors long, compute-bound jobs)
- Round-Robin – process for 1 time quantum
  - Pre-emptive (favors shorter jobs)
  - Reallocate CPU when:
    - Time quantum expires
    - Process blocks
    - Process initiates some slow-speed action, such as I/O
- Which one is fairer? Is there a downside?
- Which one might be used for an interactive system?

Policies with Priorities

- Jobs with highest priorities are scheduled first
  - Use FIFO or Round-Robin within same priority queue
- How might priorities be determined?

References

- Operating System Concepts (Silberschatz, 8th edition) Chapter 3
- http://siber.cankaya.edu.tr/OperatingSystems/ceng328/node87.html
- http://www.personal.kent.edu/~rmuhamma/OpSystems/Myos/processControl.htm
- http://www.tutorialspoint.com/operating_system/os_processes.htm
- http://wiki.answers.com/Q/Explain_process_control_block
- http://www.gitam.edu/eresource/comp/gvr(os)/4.2.htm