Lecture 21

COP3402 FALL 2015 – DR. MATTHEW GERBER – 12/2/2015

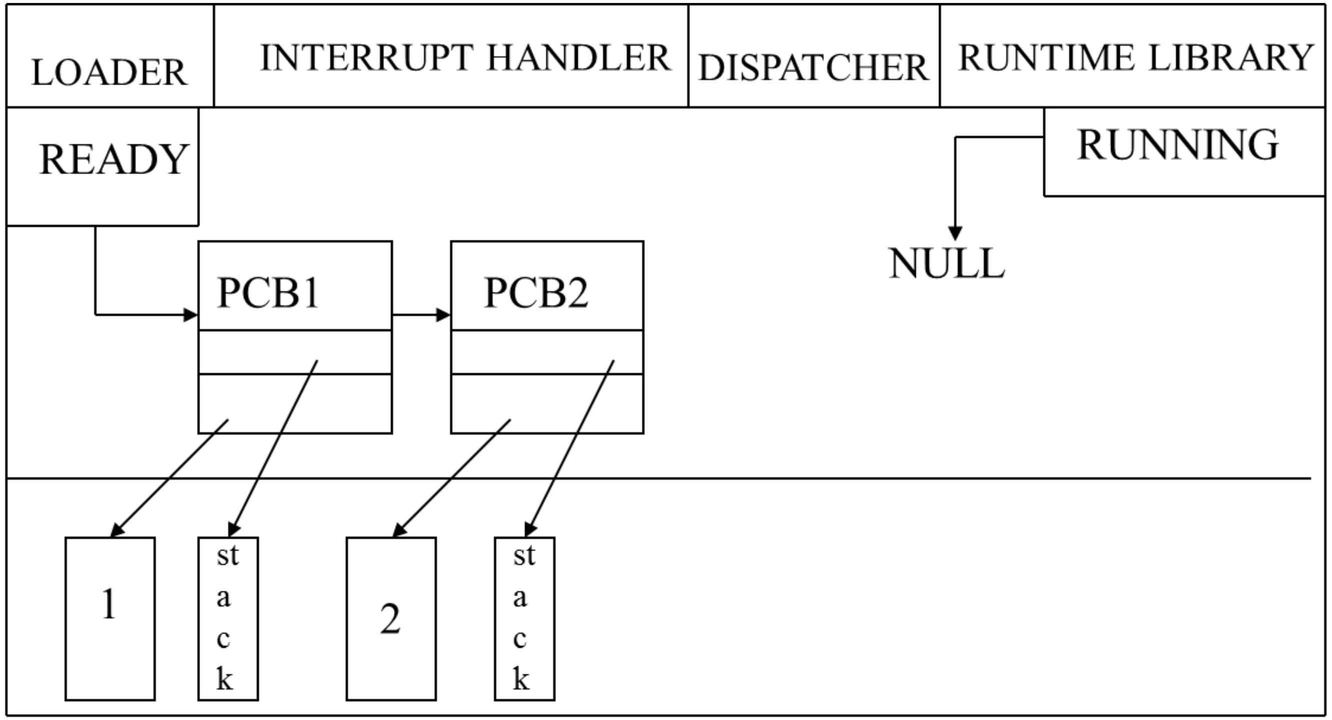
FROM EURIPIDES MONTAGNE, FALL 2014

Tonight

Process Transitions

Ready Processes

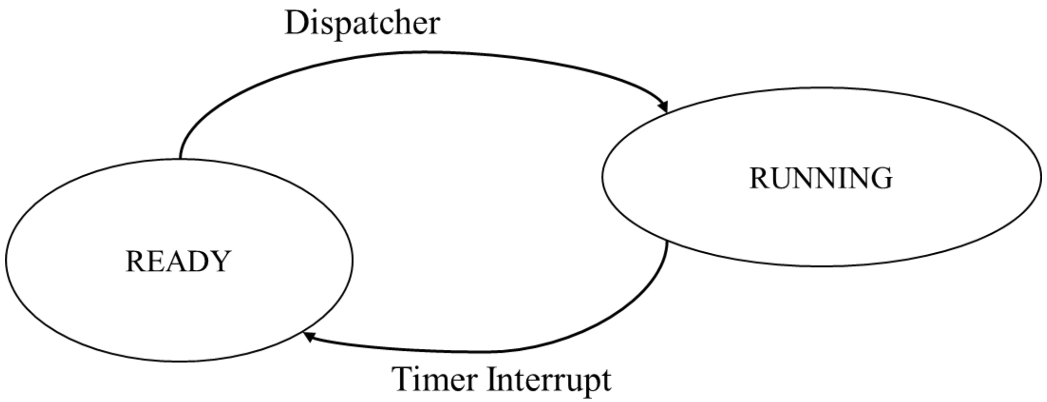
Here we have two processes both ready to be selected to run on the CPU. The resulting **ready queue** is stored as a linked list.



Ready and Running States

Processes compete for CPU time.

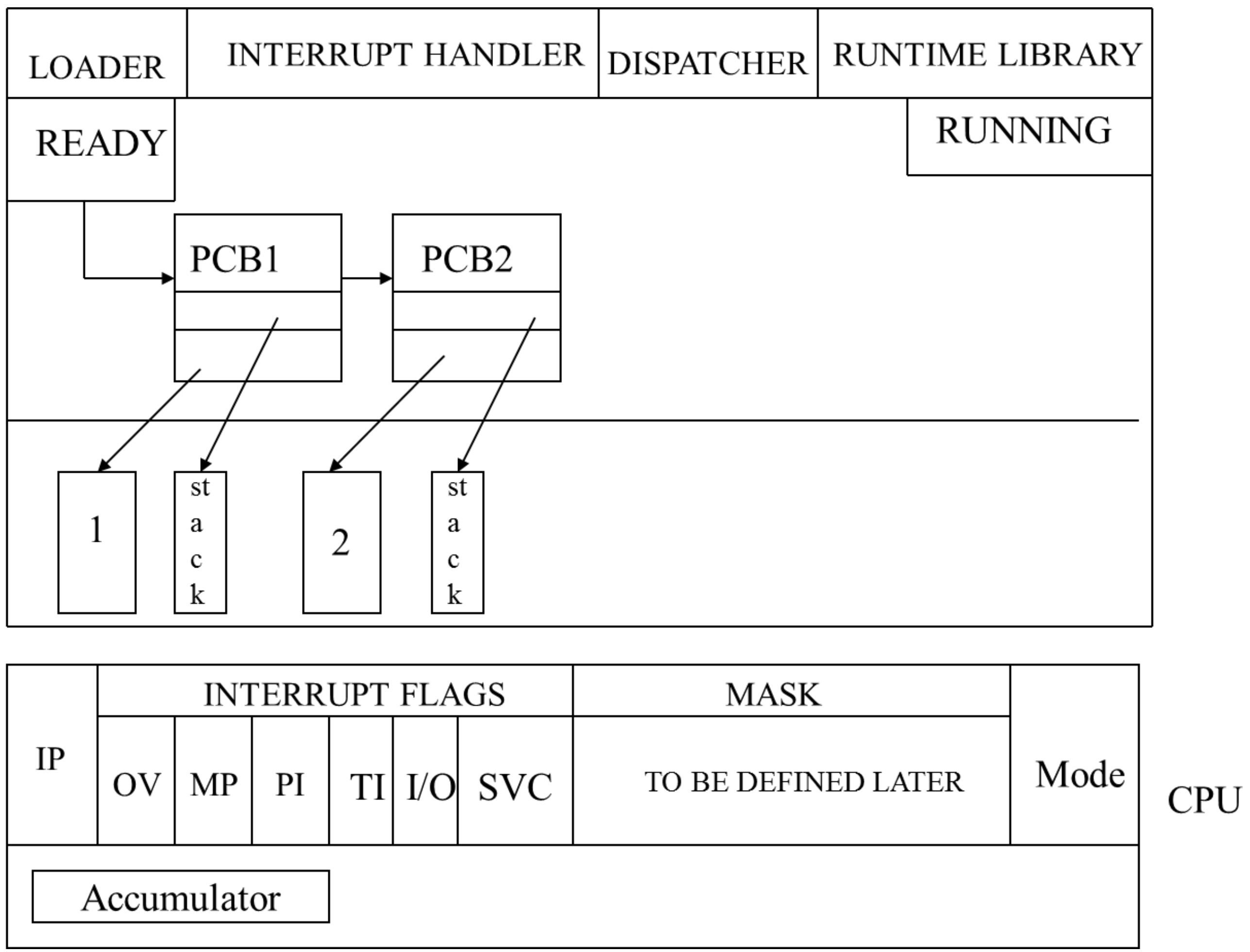
- The *dispatcher* is the OS component whose job it is to choose a running process from the ready queue
- In a typical modern operating system, a process will only be allowed a certain amount of time on the CPU before it is interrupted
- Being dispatched moves a process from **ready** to **running**
- Being timer-interrupted moves a process from **running** to **ready**



Whenever transitions occur, the OS has to save the state of the CPU in the process control block – and load the saved state of the CPU from the process control block it is letting run.

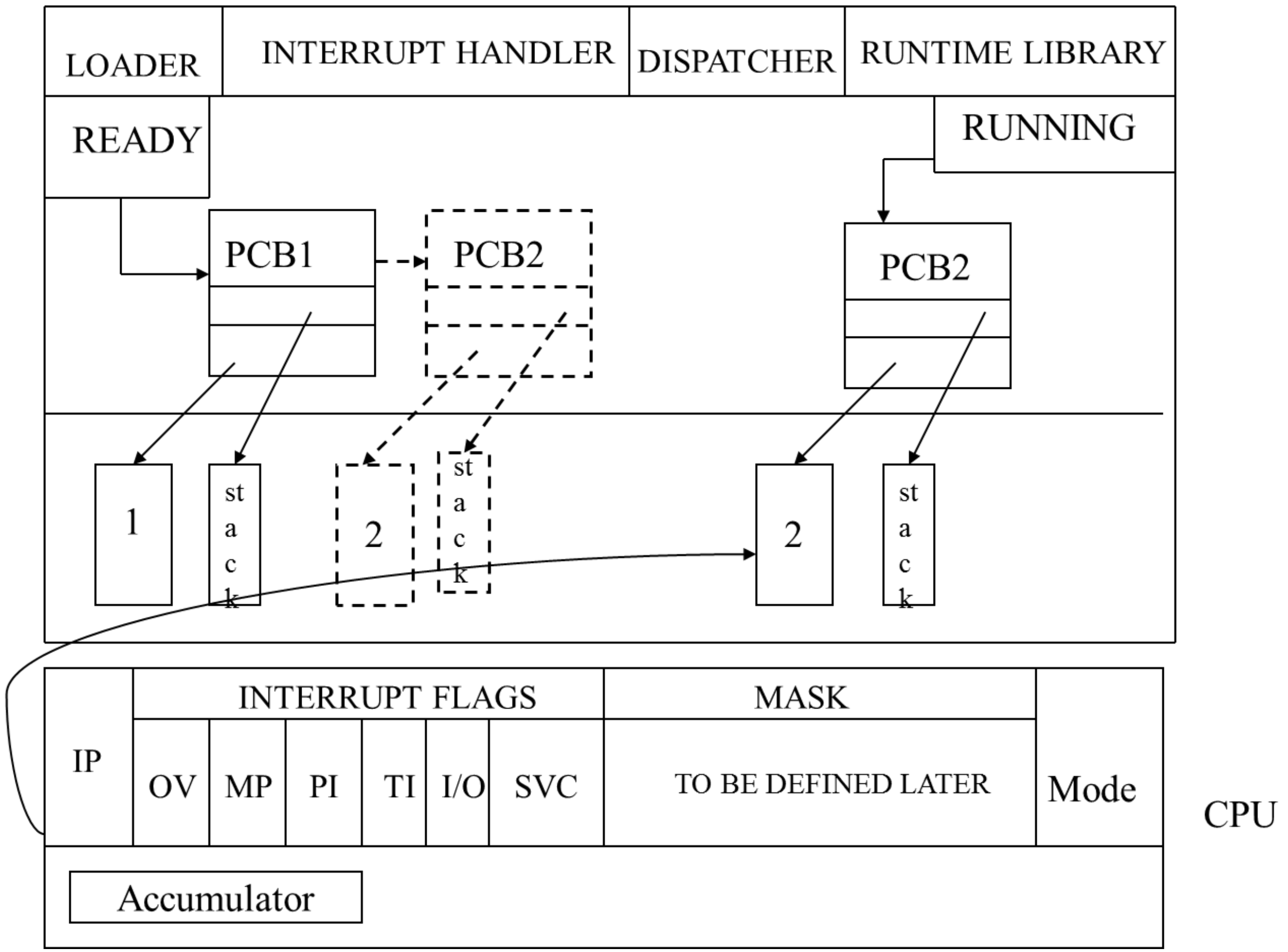
Ready to Running, Pt. 1

Process 2 has been selected, and is going to be moved from the ready queue to the running position.



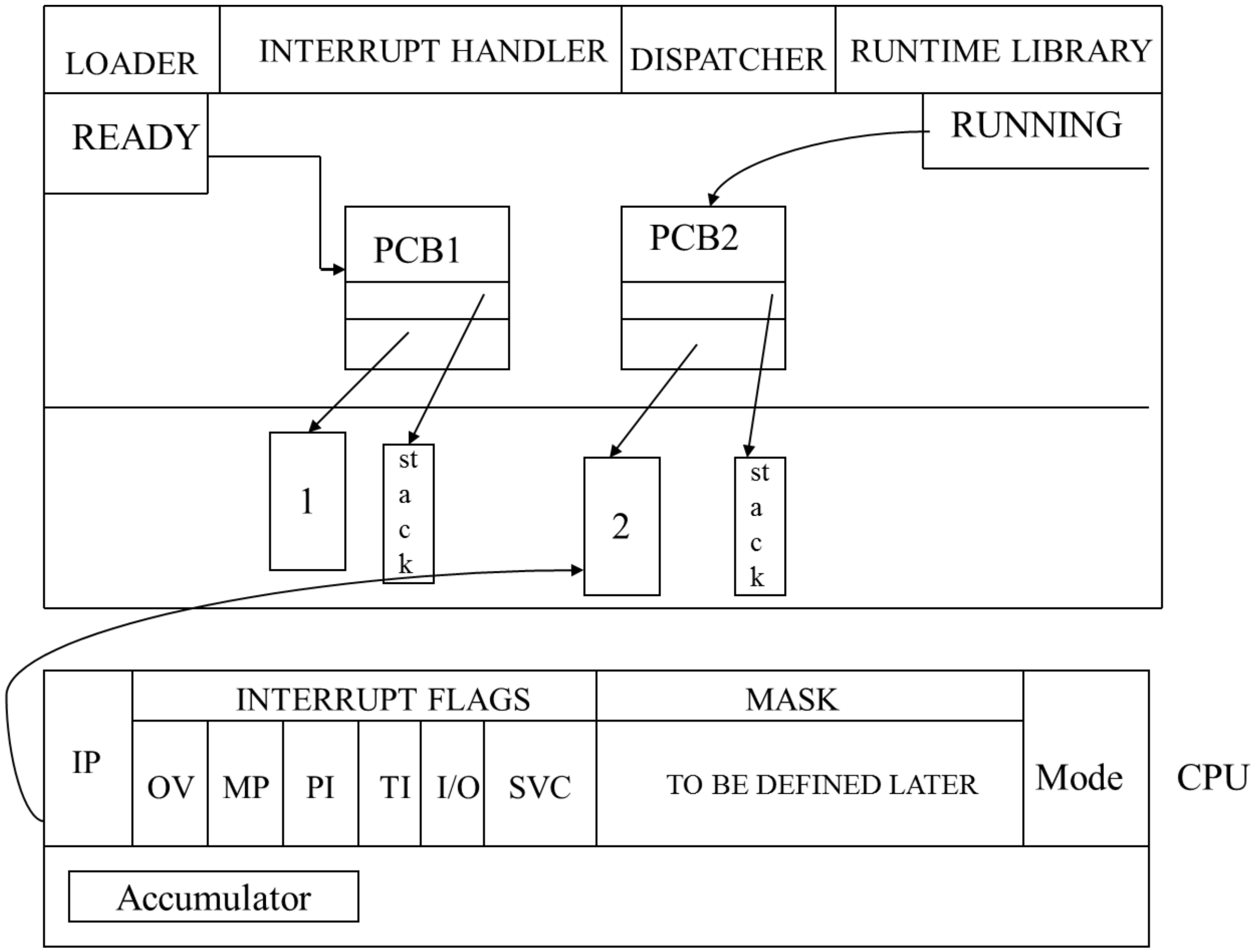
Ready to Running, Pt. 2

Process 2 is removed from the ready queue and placed in the running slot, and the CPU is assigned to it.



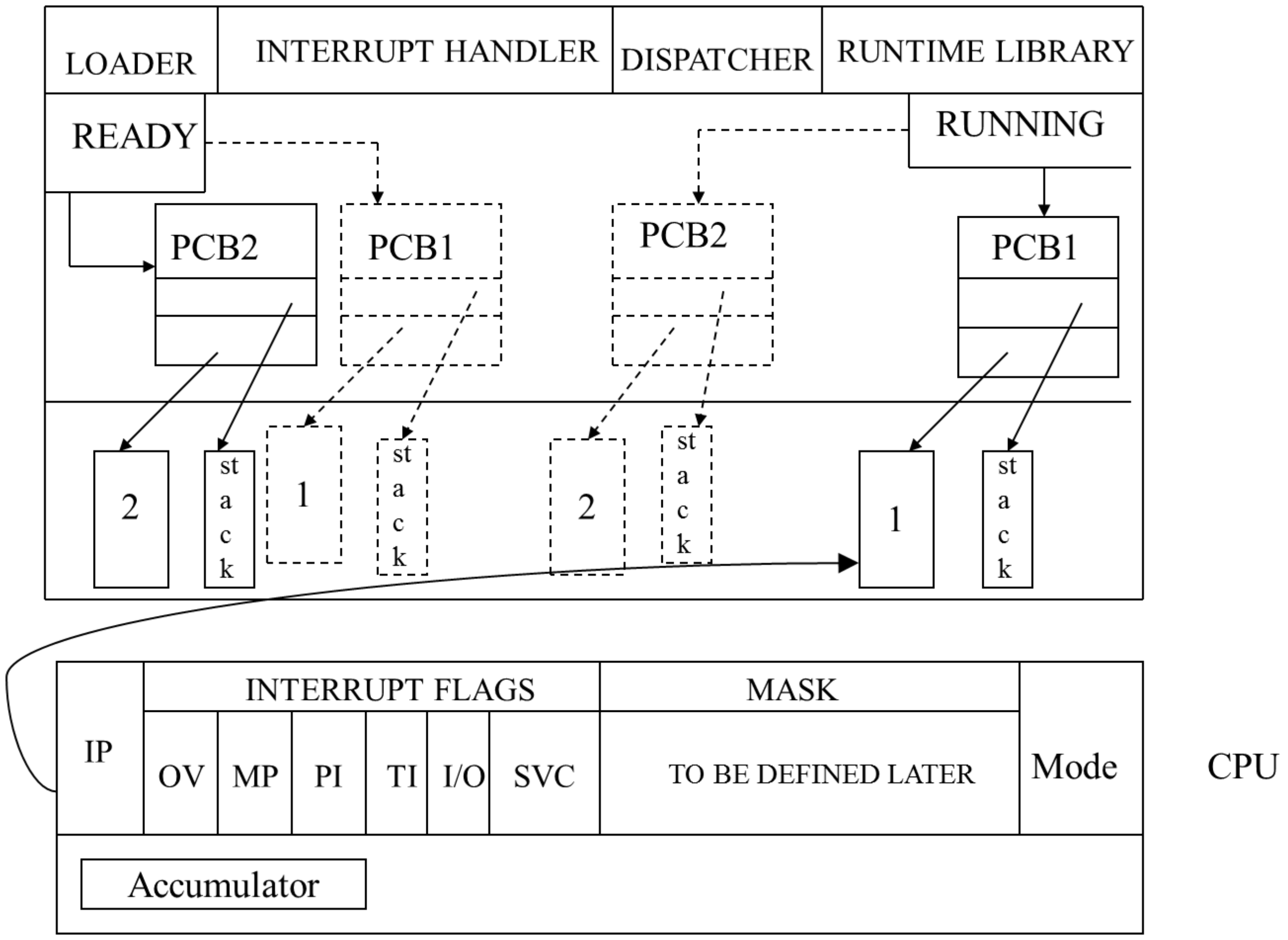
Back to Ready, Pt. 1

Process 2's time is up, and it is going to be moved from the running position back to the ready queue.



Back to Ready, Pt. 2

Here Process 2 has been removed from the running slot, while it has been given to Process 1 – which has in turn been removed from the ready queue.

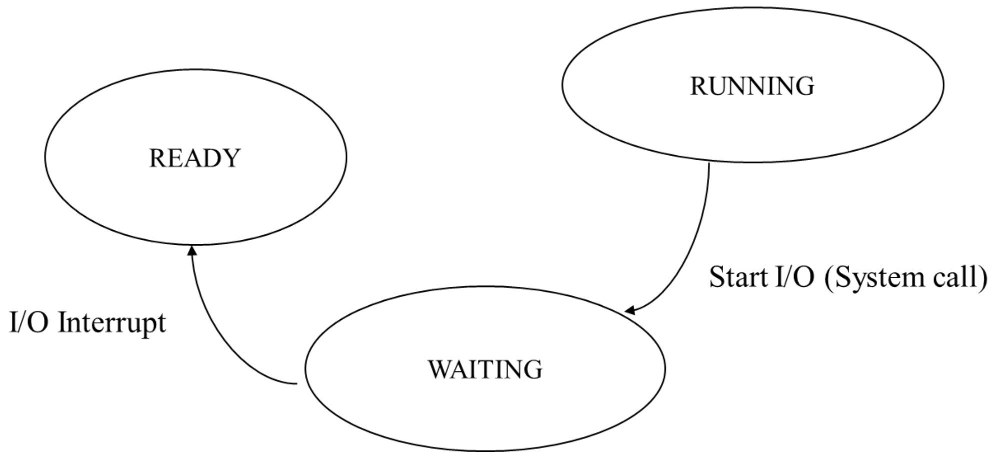


The Waiting State

Processes don't just use the CPU; they need I/O.

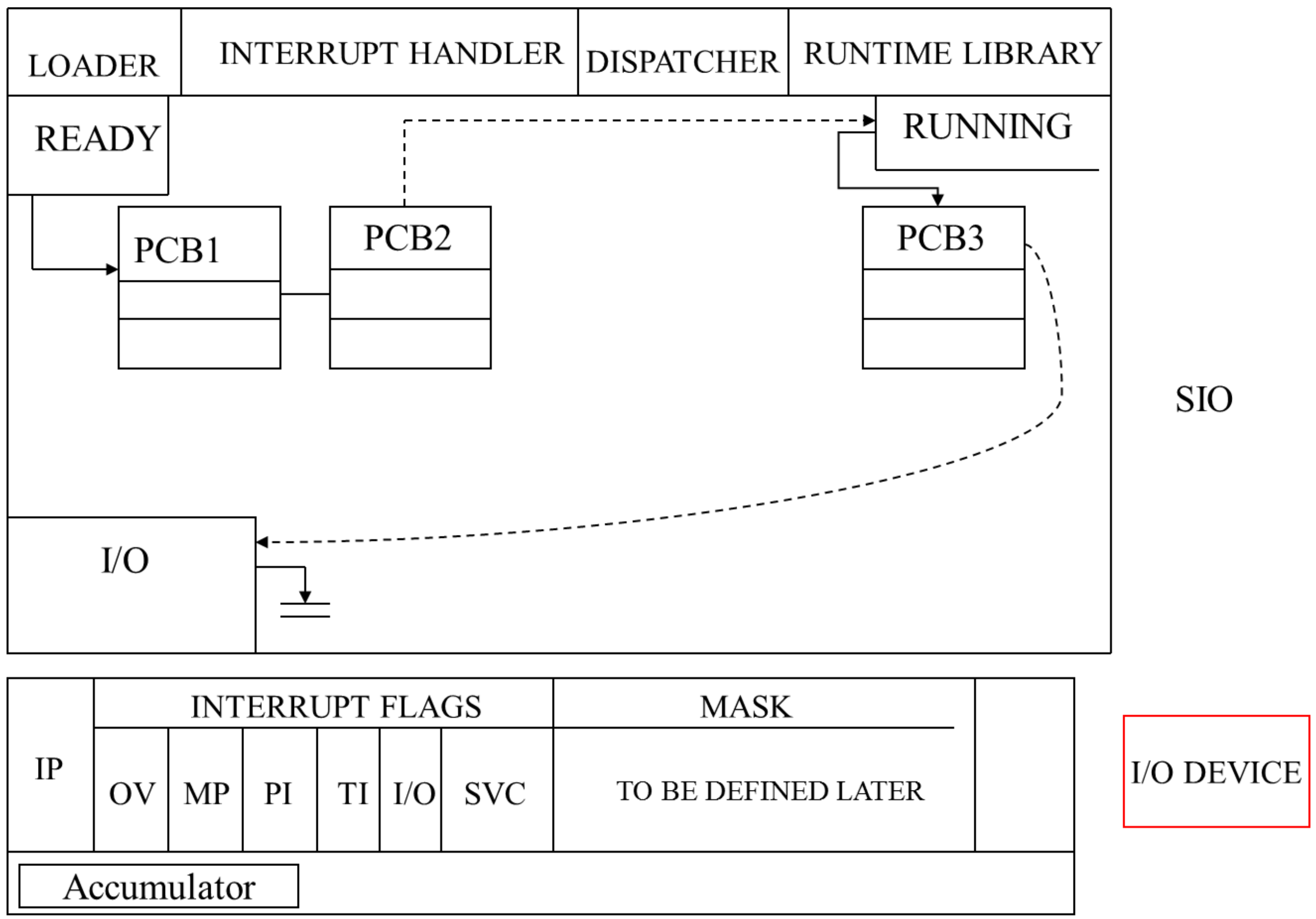
- When a process makes an I/O request, its status will be changed to **waiting**.
- When an I/O interrupt comes in to indicate that the process's I/O is complete, its status will be changed back to **ready**.

(As a historical note, I/O handling this clean and efficient is relatively new on small computers. It is, thankfully, now common.)



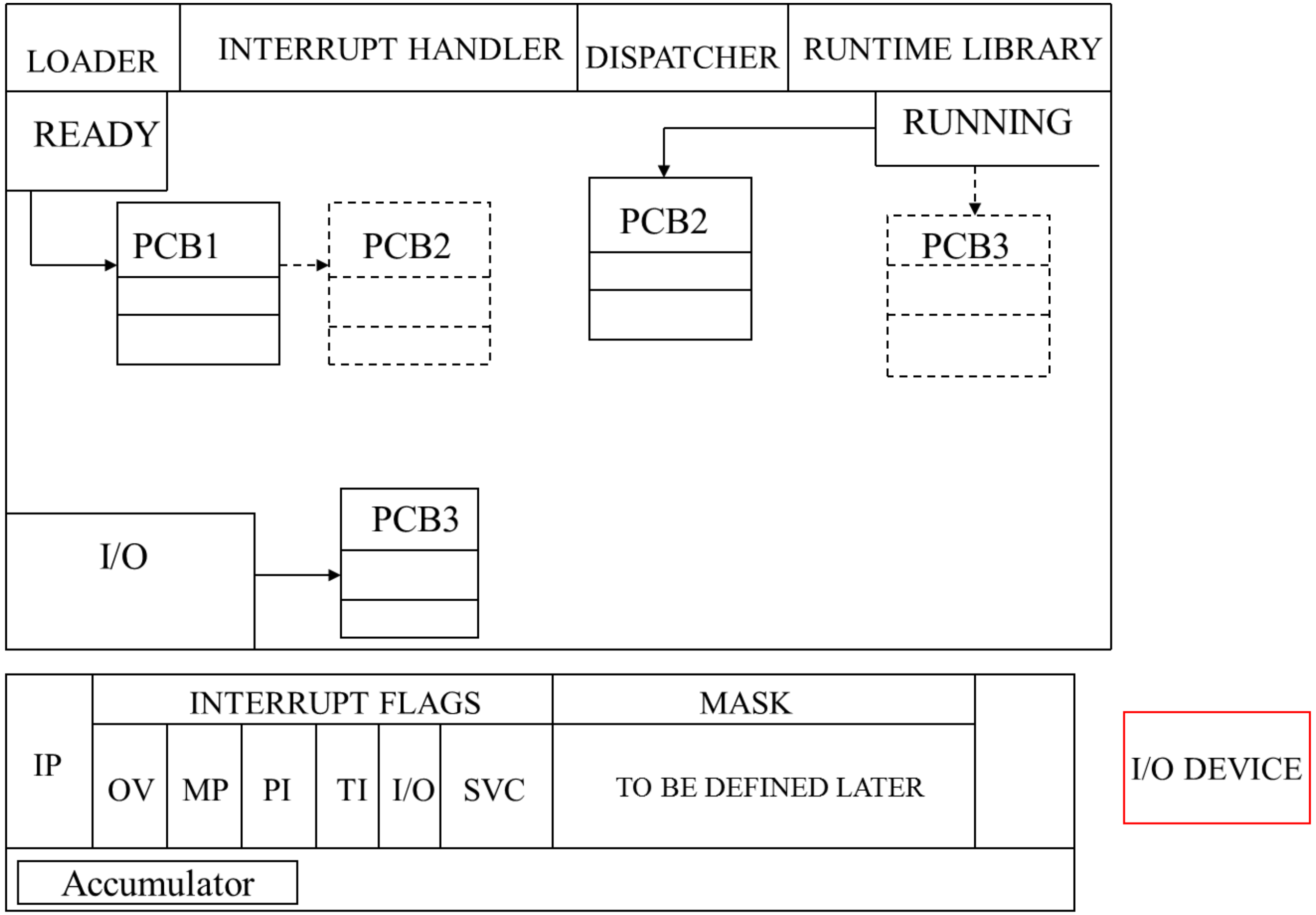
I/O System Call, Pt. 1

Here we see process 3 making an I/O system call. It will be removed from the running slot and added to the I/O wait queue, while another process will be allowed to use the CPU.



I/O System Call, Pt. 2

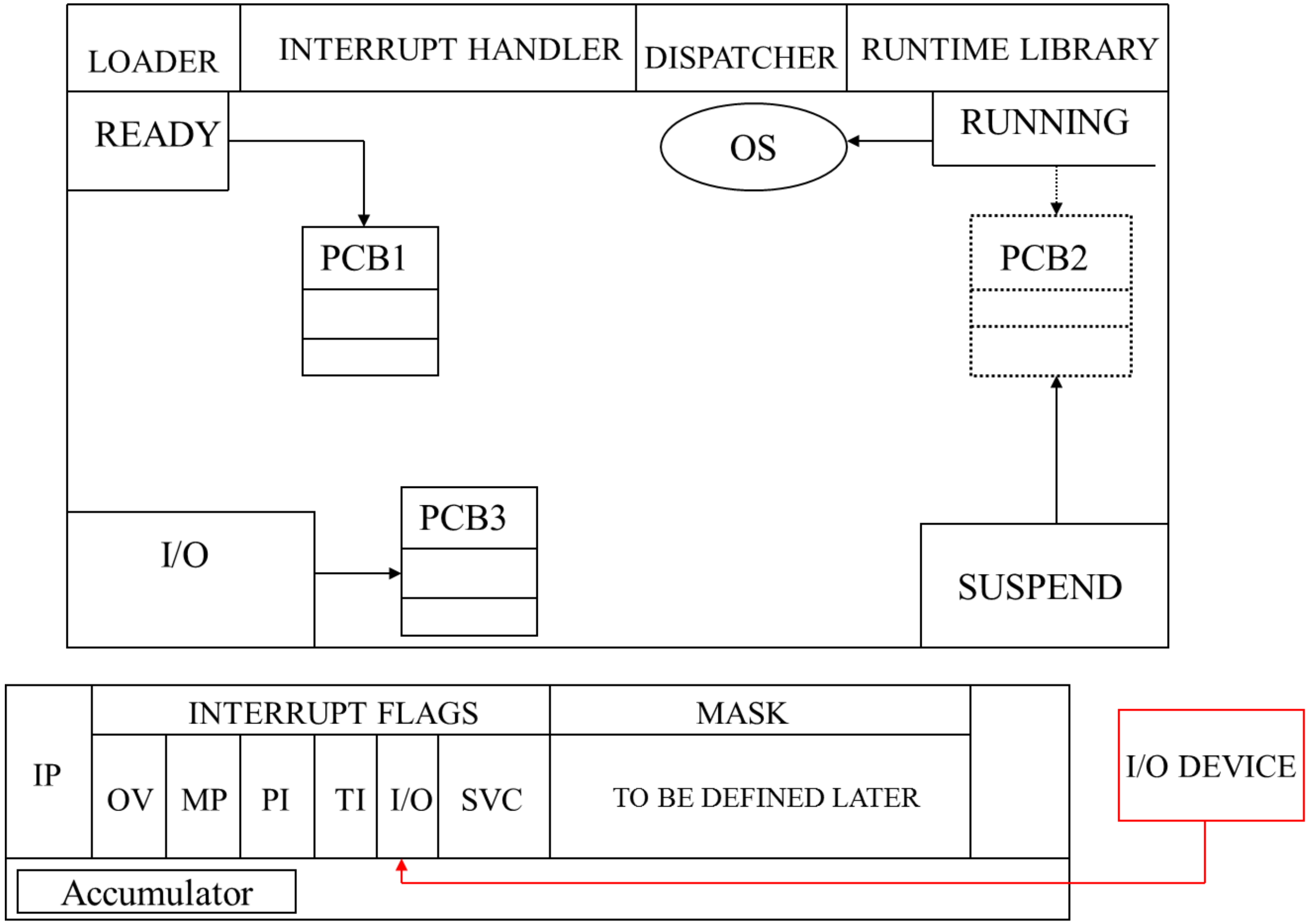
Process 2 is the lucky winner. It's going to get to use the CPU for a while.



I/O Interrupt, Pt. 1

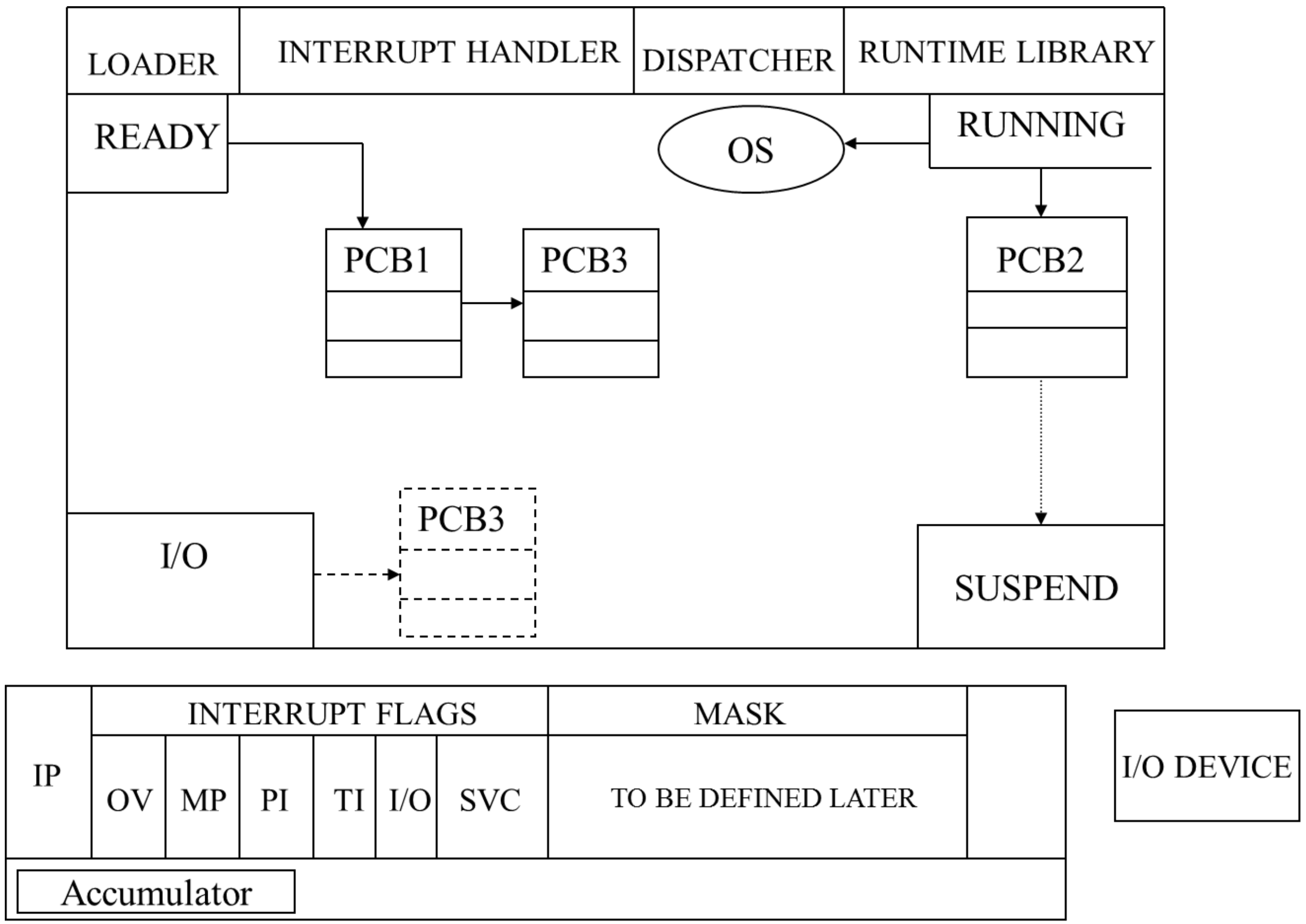
Process 3’s I/O is complete and the OS now has to figure out what to do about it. Process 2 is **suspended** while the OS does just that.

In a modern operating system this interrupt handling time is very, very short, but it still has to happen.



I/O Interrupt, Pt. 2

And we're done. Process 3 is now on the ready queue. Process 2 still has the CPU – I/O being complete is *not* a guarantee of an immediate context switch.

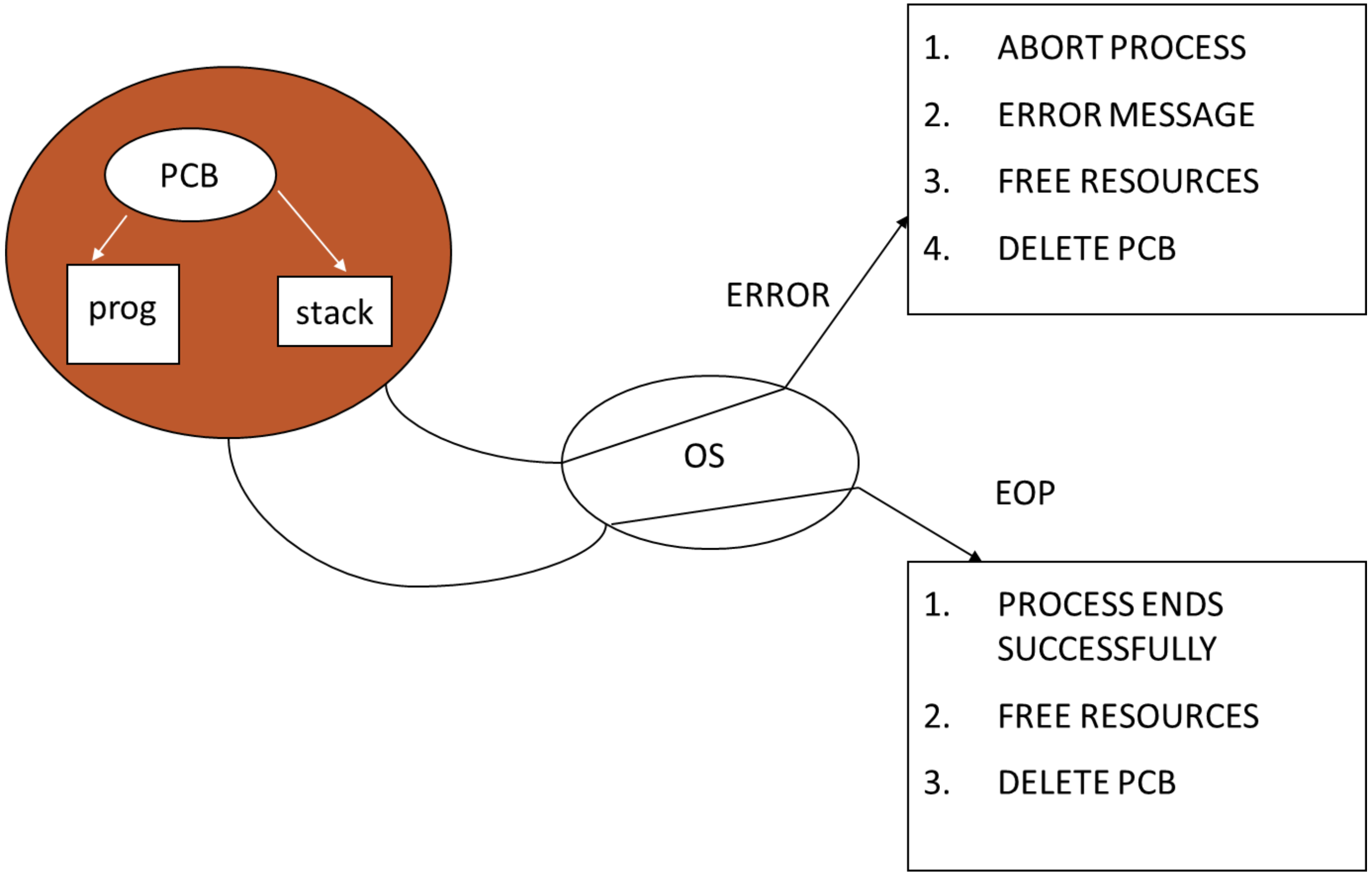


Termination

Processes can end one of two ways – and an OS actually does much the same thing for either one.

On an **abnormal end** due to an error, the OS aborts the process, typically provides some sort of error message, frees the process's resources and deletes the PCB.

On a normal end, it does the last two of those.



Next Time:
Review