Threads and Processes

Peter J. Denning

© Copyright 2019, Peter J Denning

Process

- A program in execution on virtual machine with its own address space and CPU
  - "Trace of instruction pointer through instruction sequence"
  - "Thread of control"
  - Process: dynamic. Program: static.
  - Abstraction of dynamics of executing program.

© Copyright 2019, Peter J Denning

Thread

- Trace of instruction pointer through instruction sequence in an address space
  - "Thread of control"
  - Many threads can share one address space, their shared memory

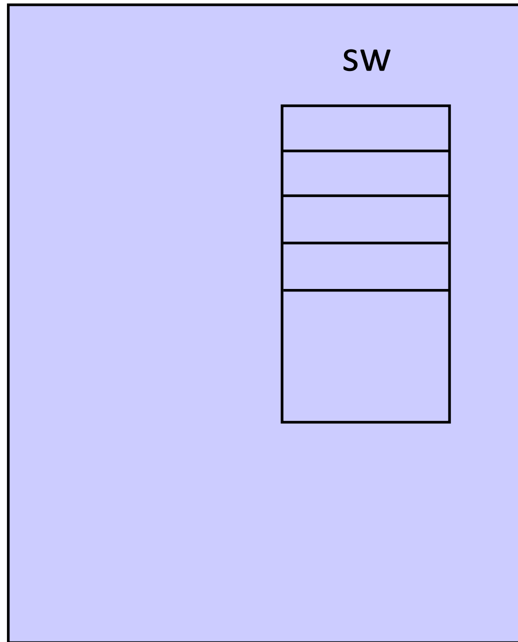
© Copyright 2019, Peter J Denning

Time Sharing

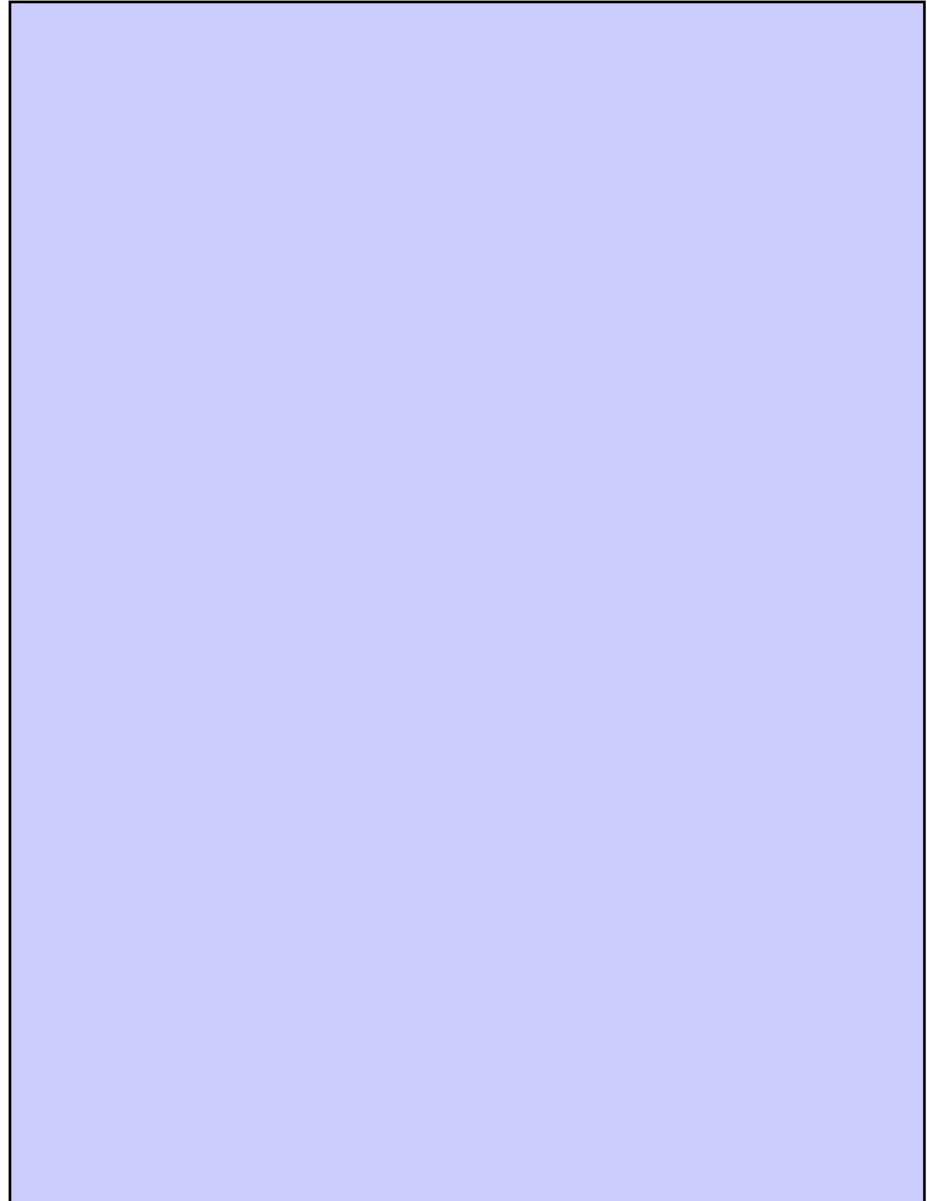
- A method of implementing multiple threads on a computer
- One CPU multiplexed among processes, for one time slice at a time.
- Creates illusion of independent concurrent processes all running at slower speeds than the CPU.

© Copyright 2019, Peter J Denning

CPU

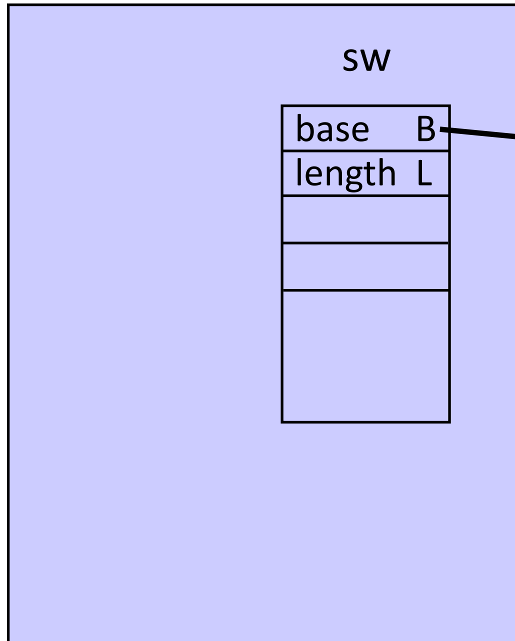


RAM

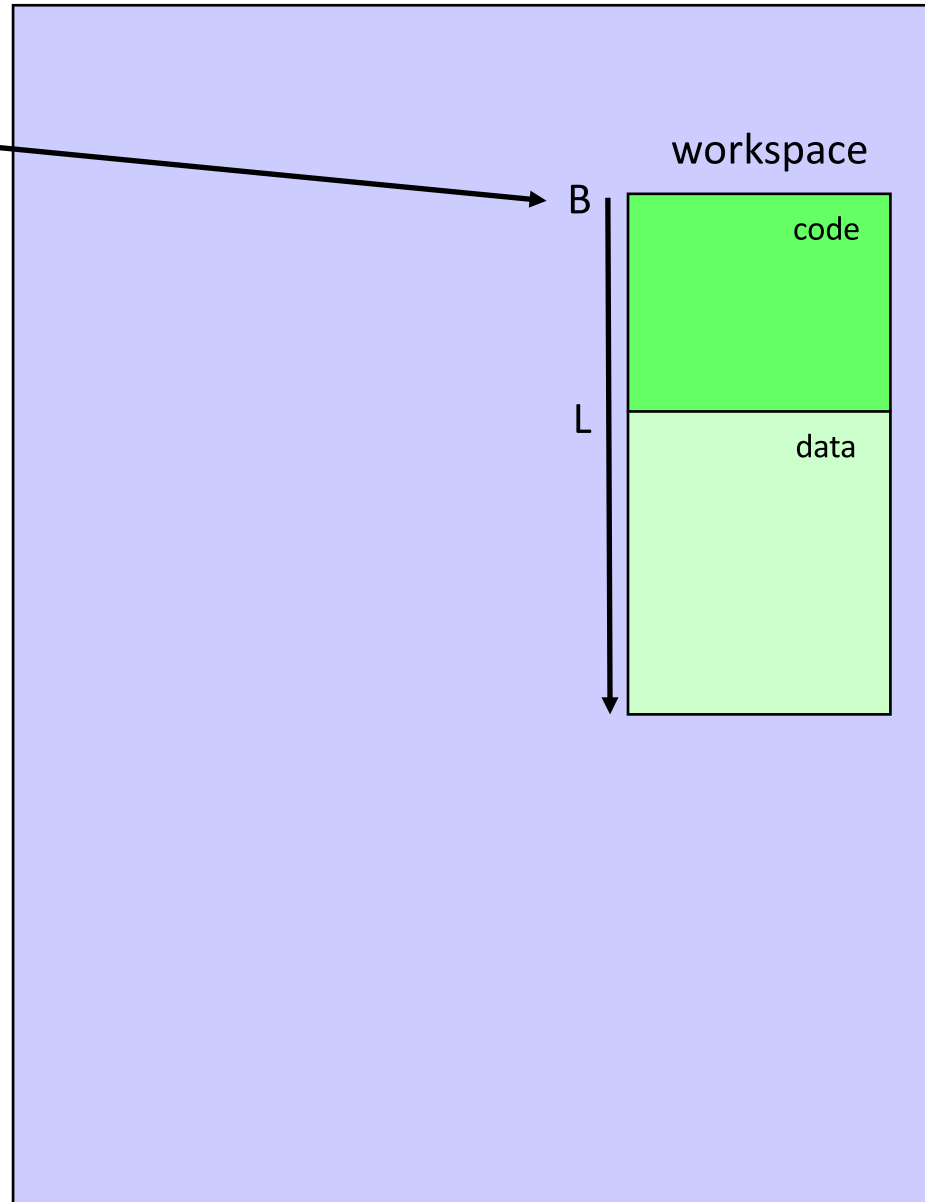


© Copyright 2019, Peter J Denning

CPU



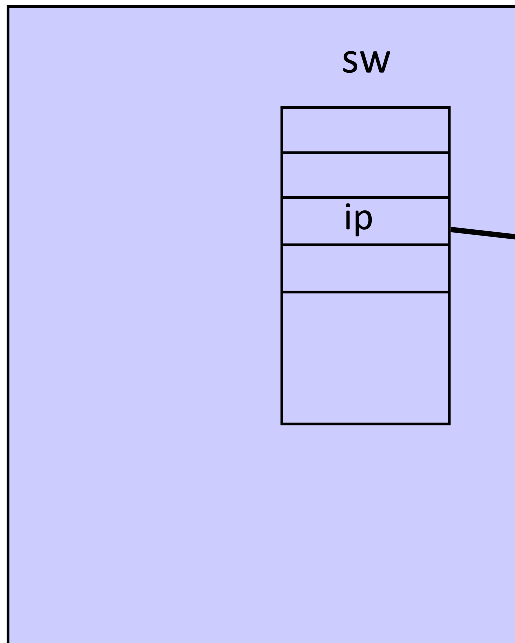
RAM

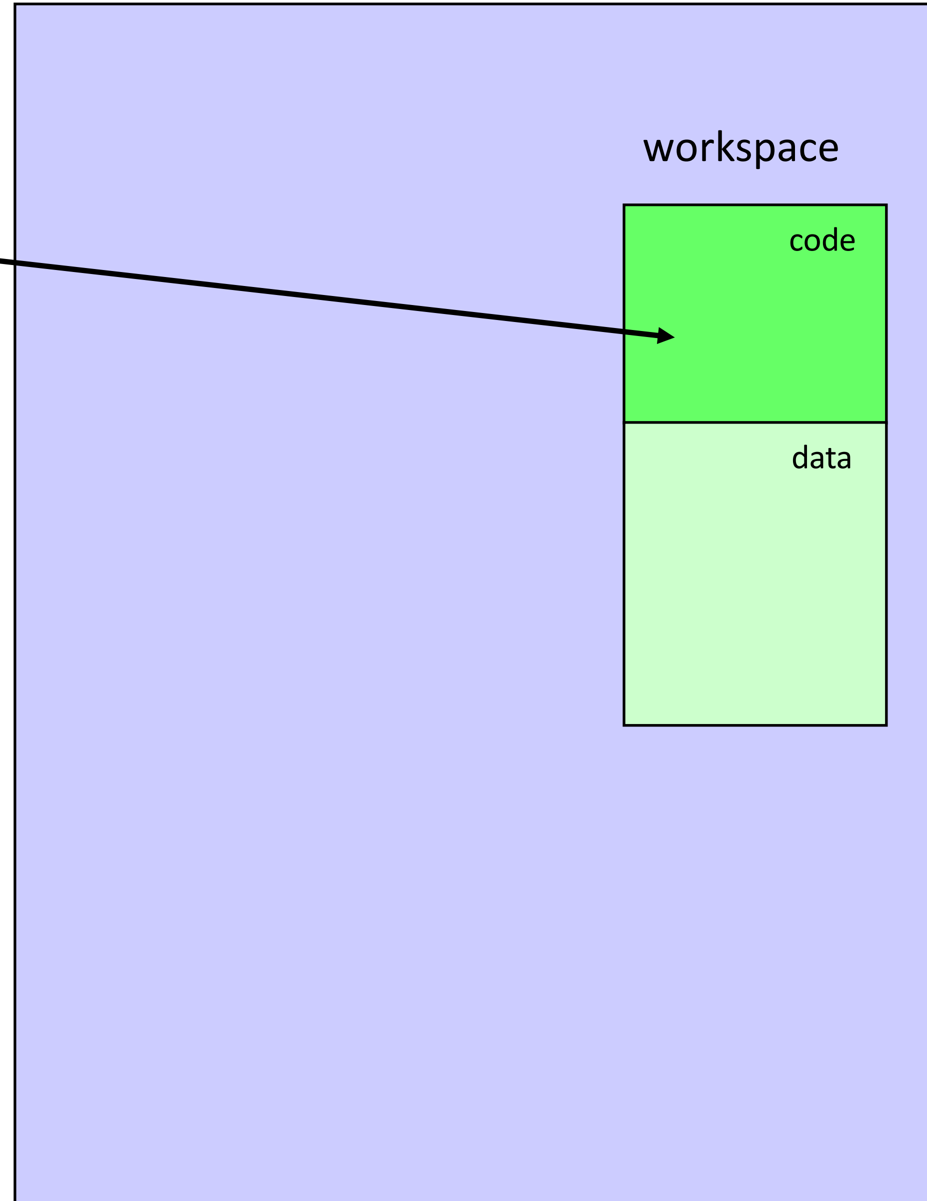


CPU can only access RAM locations B,…,B+L-1

Otherwise, MEMORY BOUND ERROR

© Copyright 2019, Peter J Denning



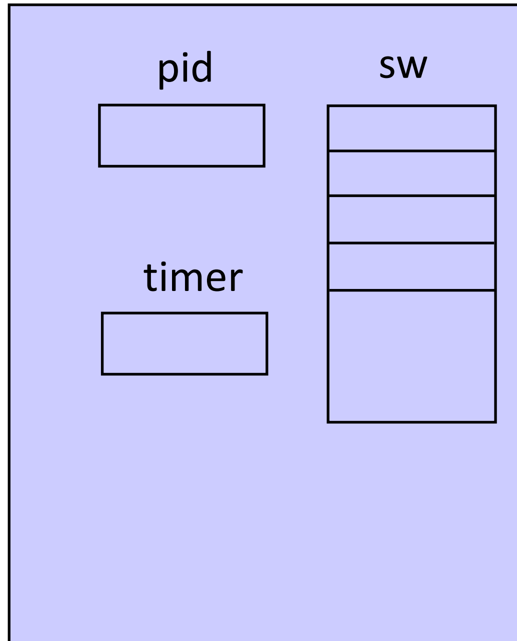


Current instruction is at address ip, within the code segment of the workspace.

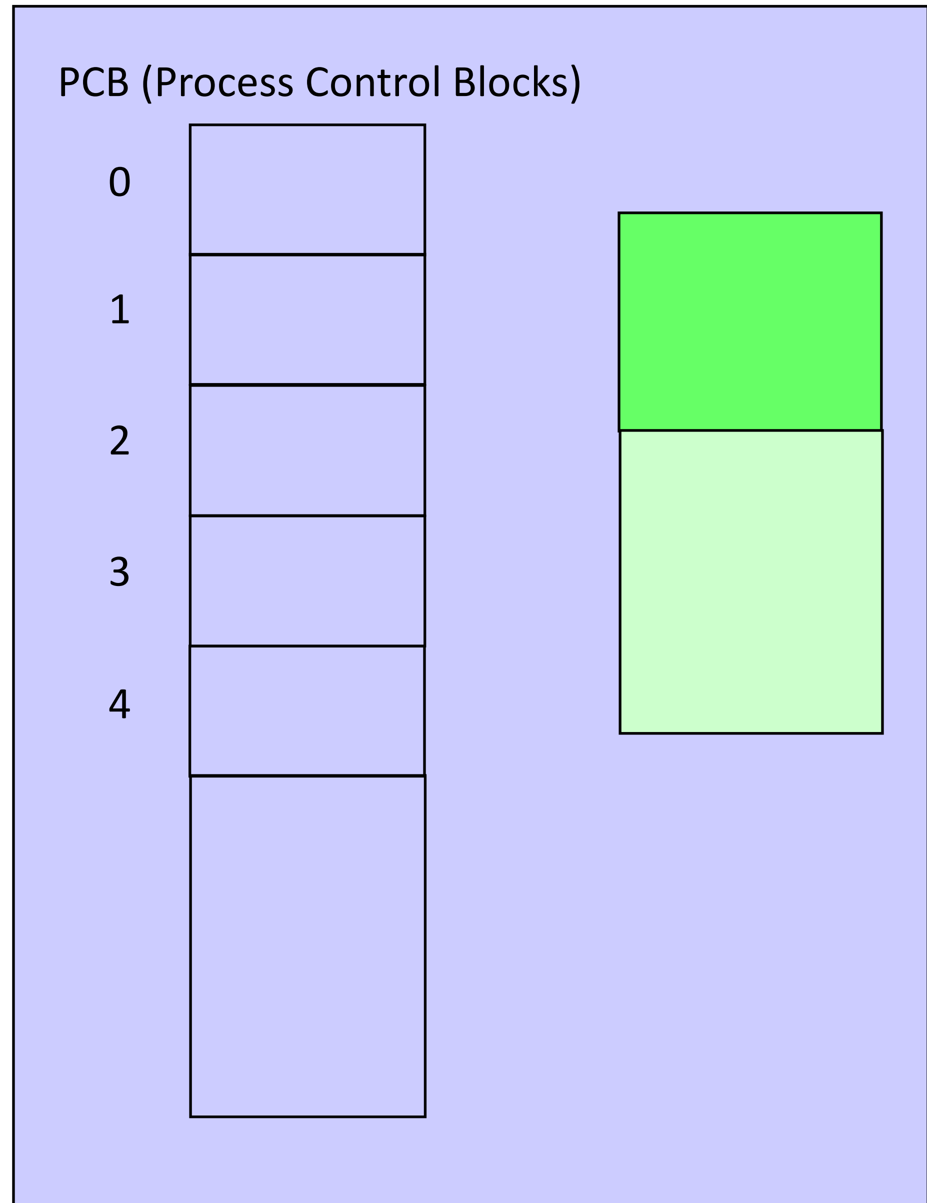
Next instruction is at ip+1, except if branch

© Copyright 2019, Peter J Denning

CPU



RAM



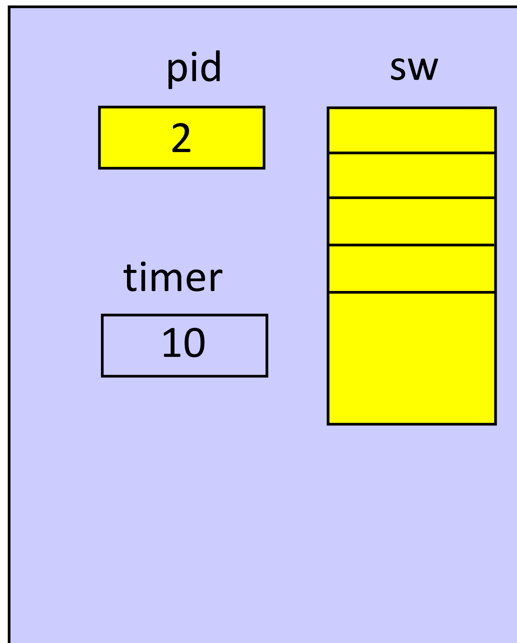
pid = ID of running process

timer = time remaining to time slide end

PCB = snapshot of CPU state at last context switch (in kernel private memory)

© Copyright 2019, Peter J Denning

CPU

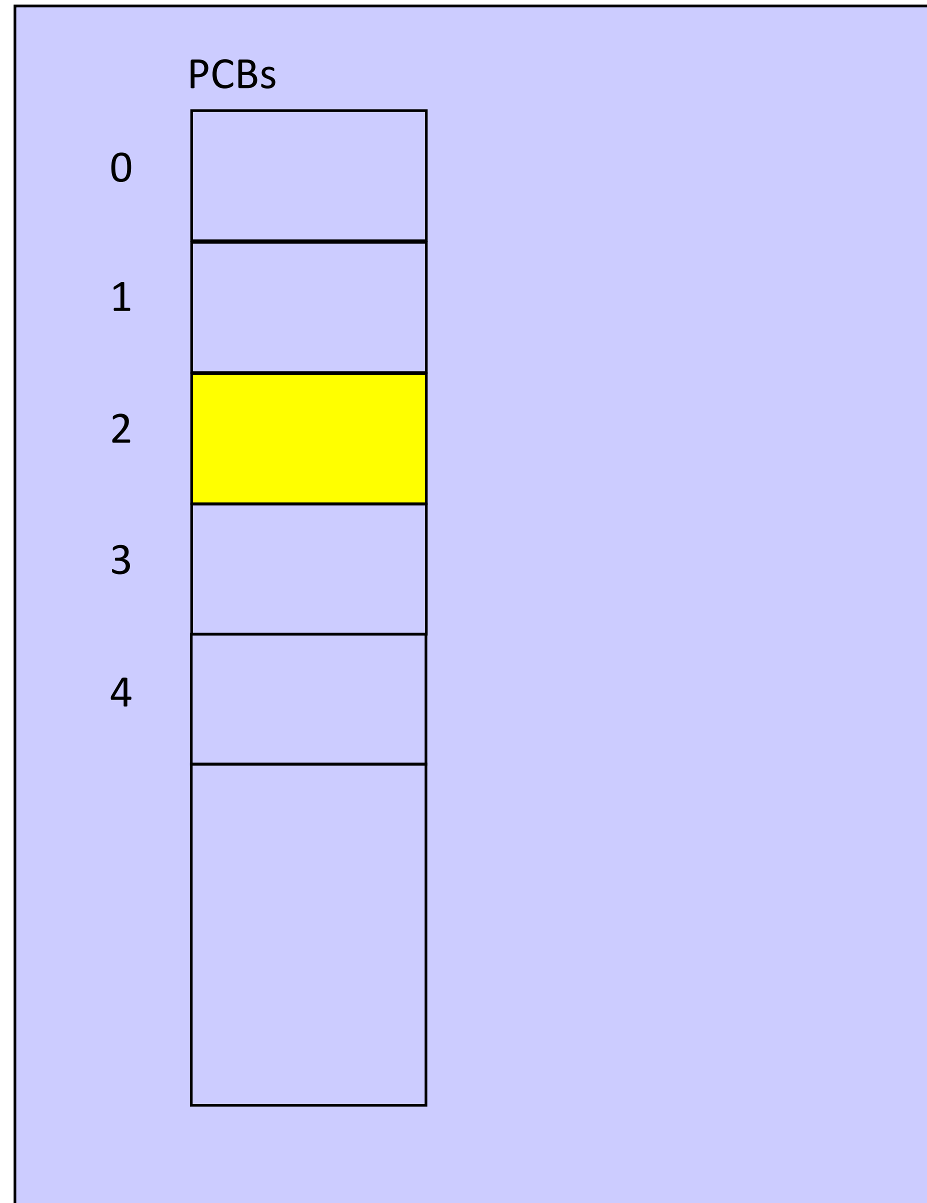


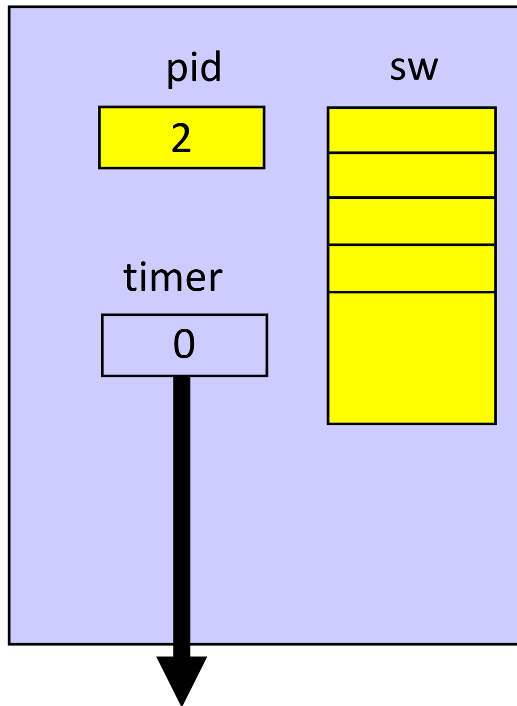
process 2 running on CPU, its PCB has image of sw at start of time slice

timer has 10 ticks remaining

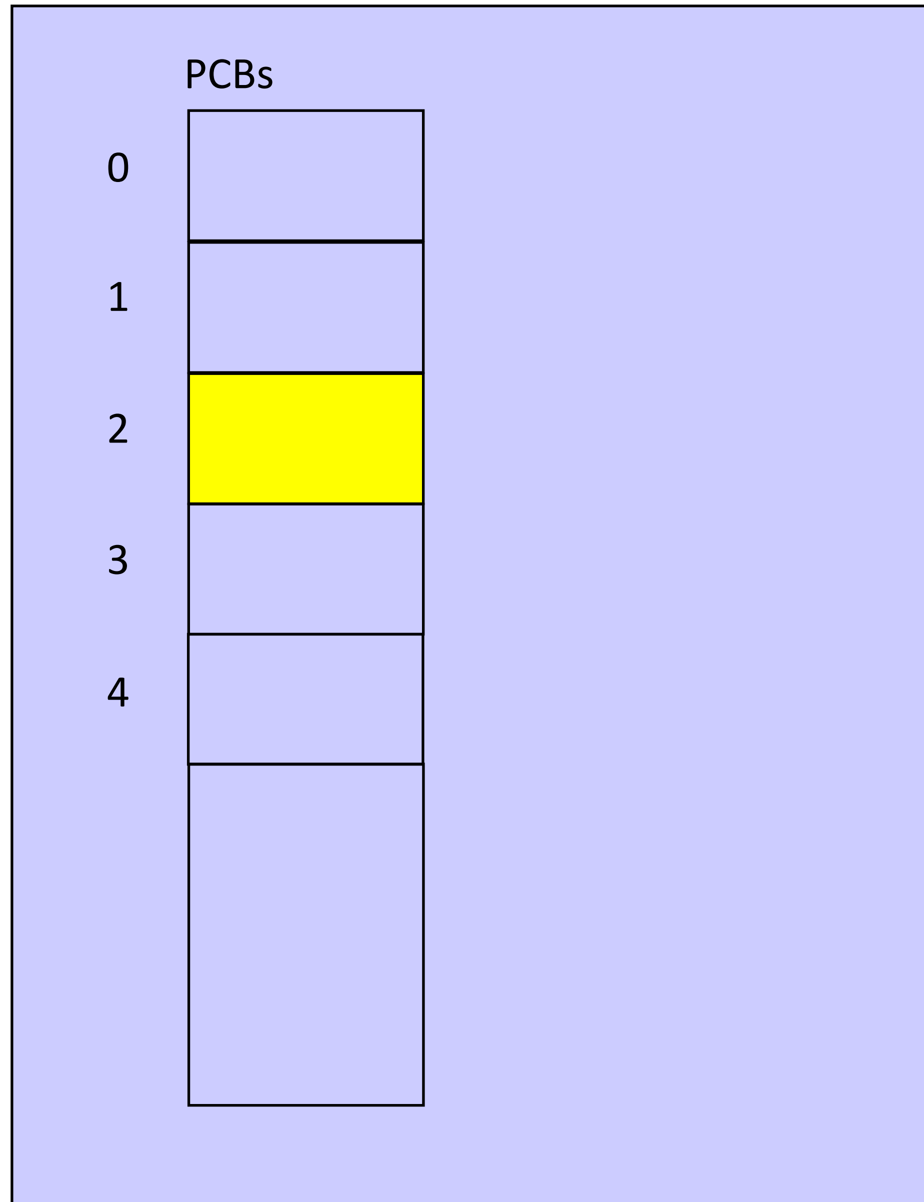
© Copyright 2019, Peter J Denning

RAM





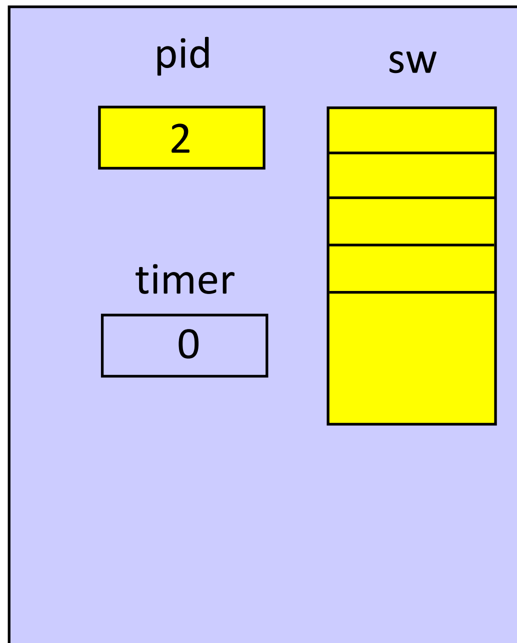
timer, gone to 0, triggers
”time slice end” interrupt

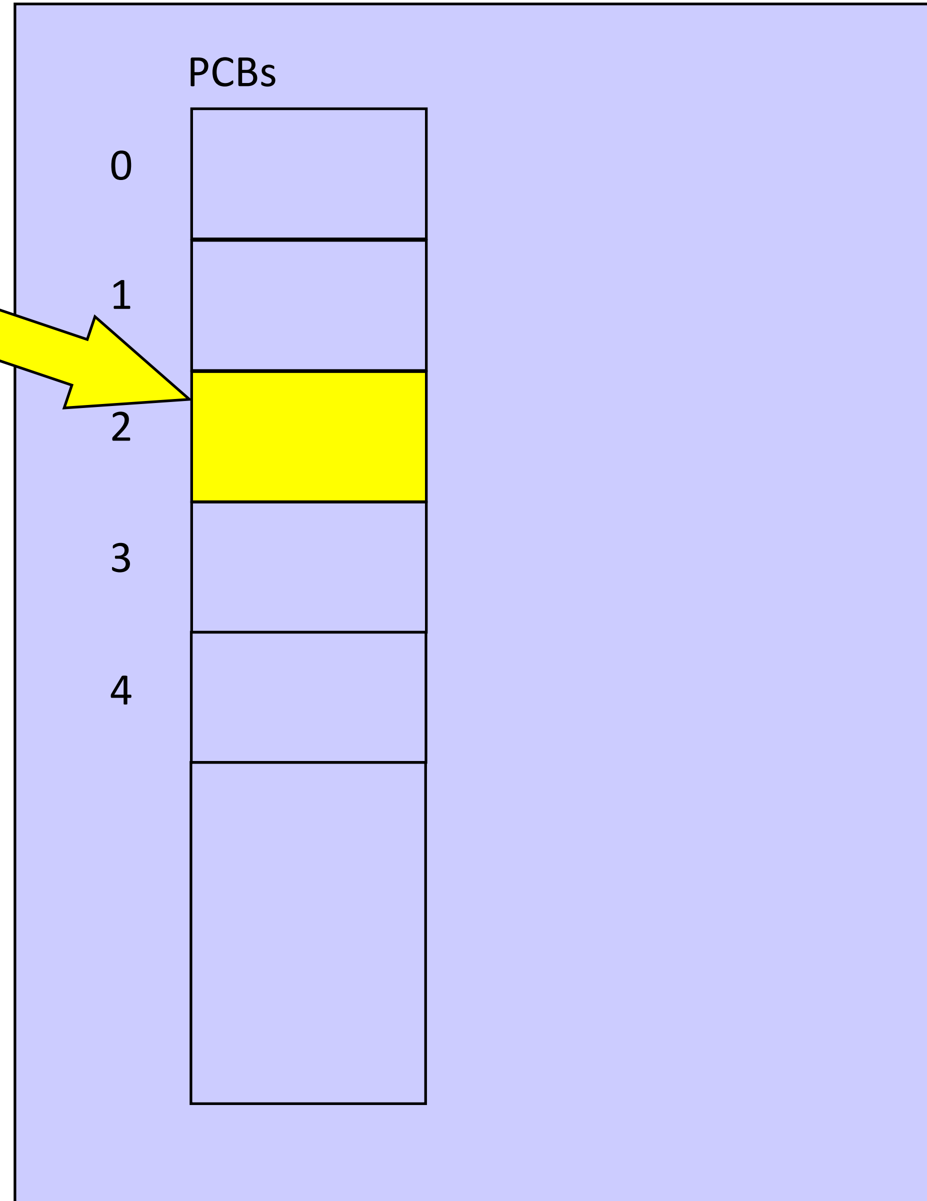


© Copyright 2019, Peter J Denning

CPU

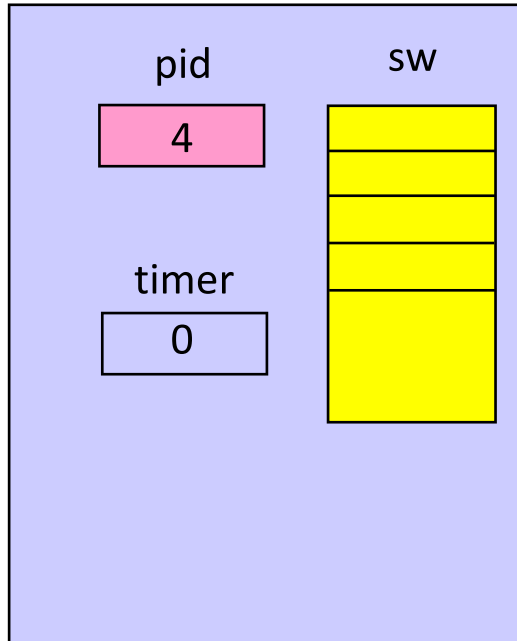
RAM

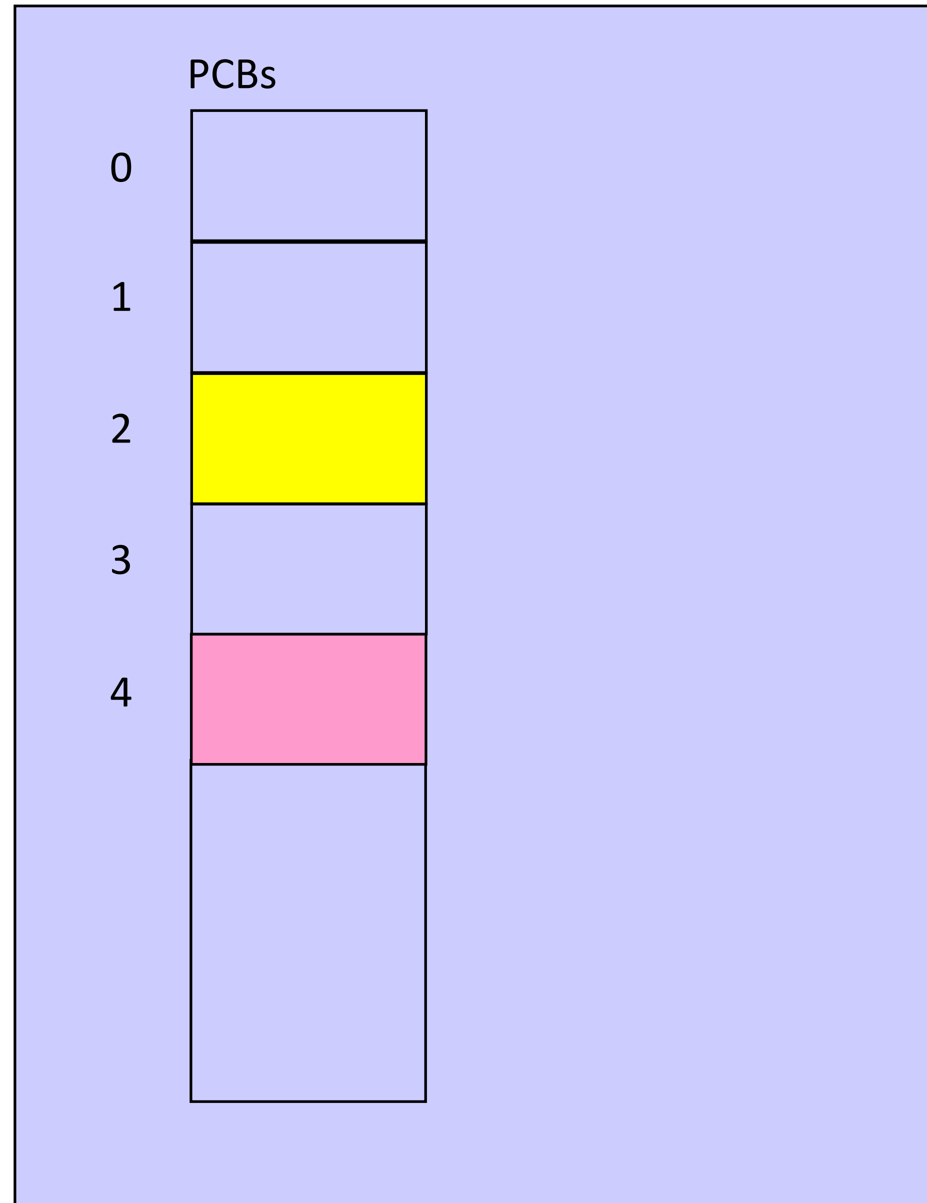




CPU executes SAVESW, which copies entire sw into PCB[2]

© Copyright 2019, Peter J Denning

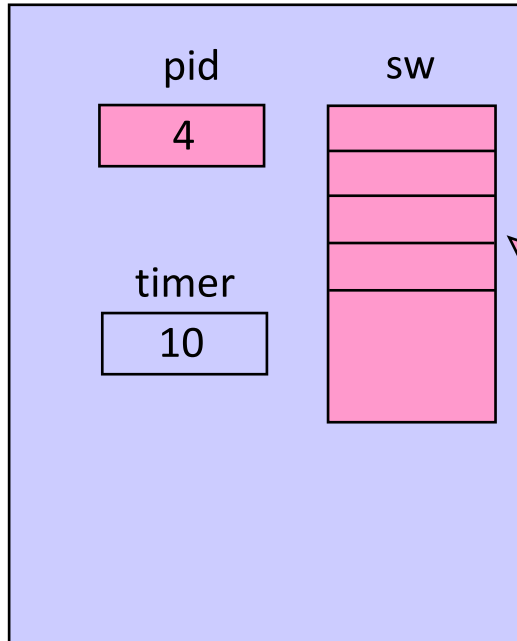




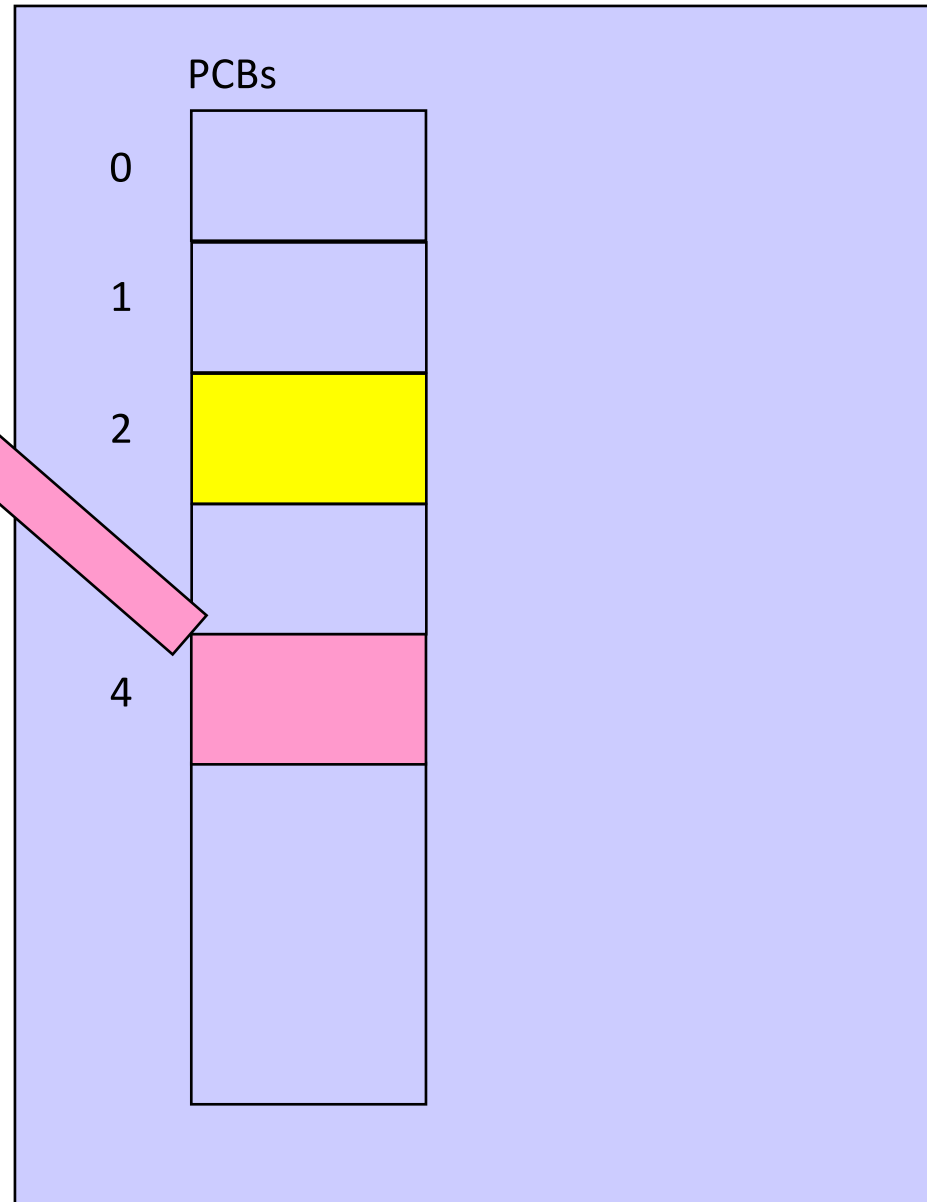
OS selects process 4 to be next on CPU

© Copyright 2019, Peter J Denning

CPU



RAM

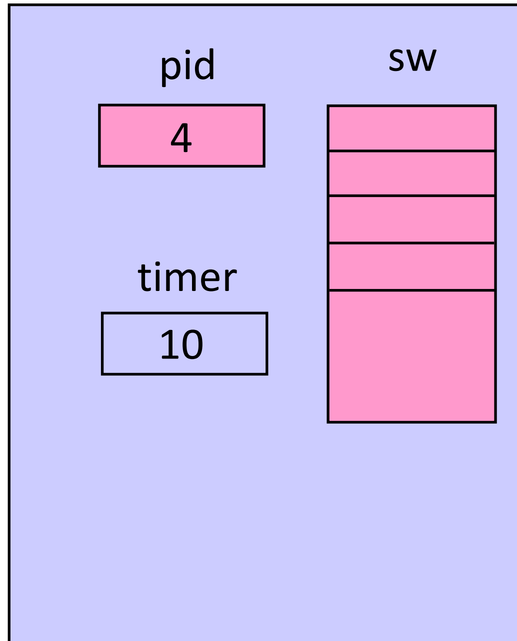


CPU executes LOADSW, which copies PDB[4] into the CPU sw registers

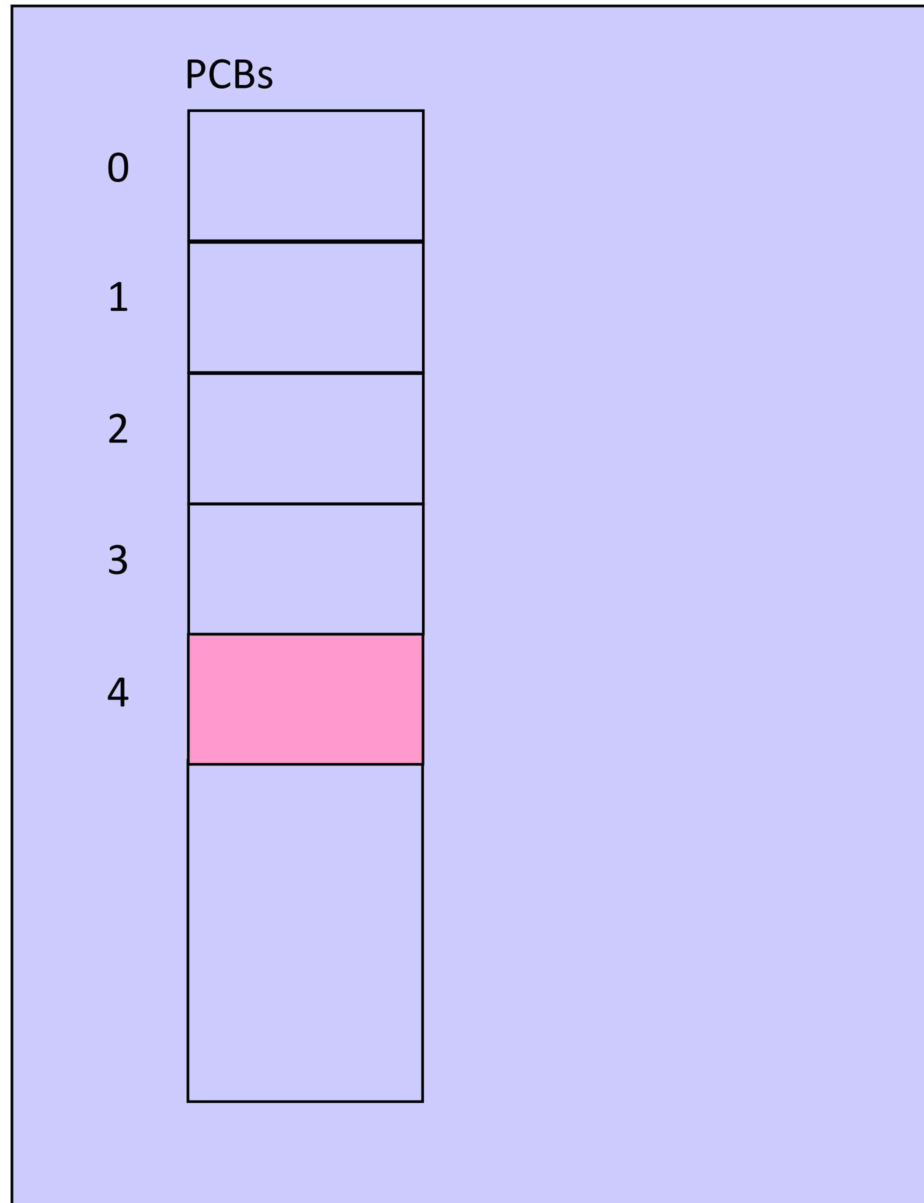
timer set to time-slice value

© Copyright 2019, Peter J Denning

CPU

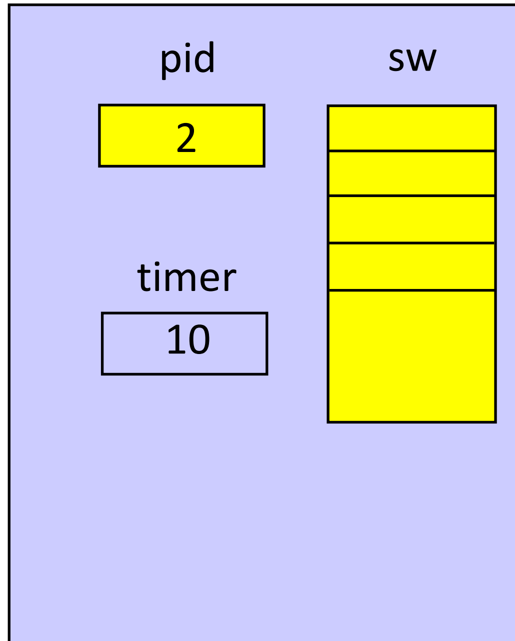


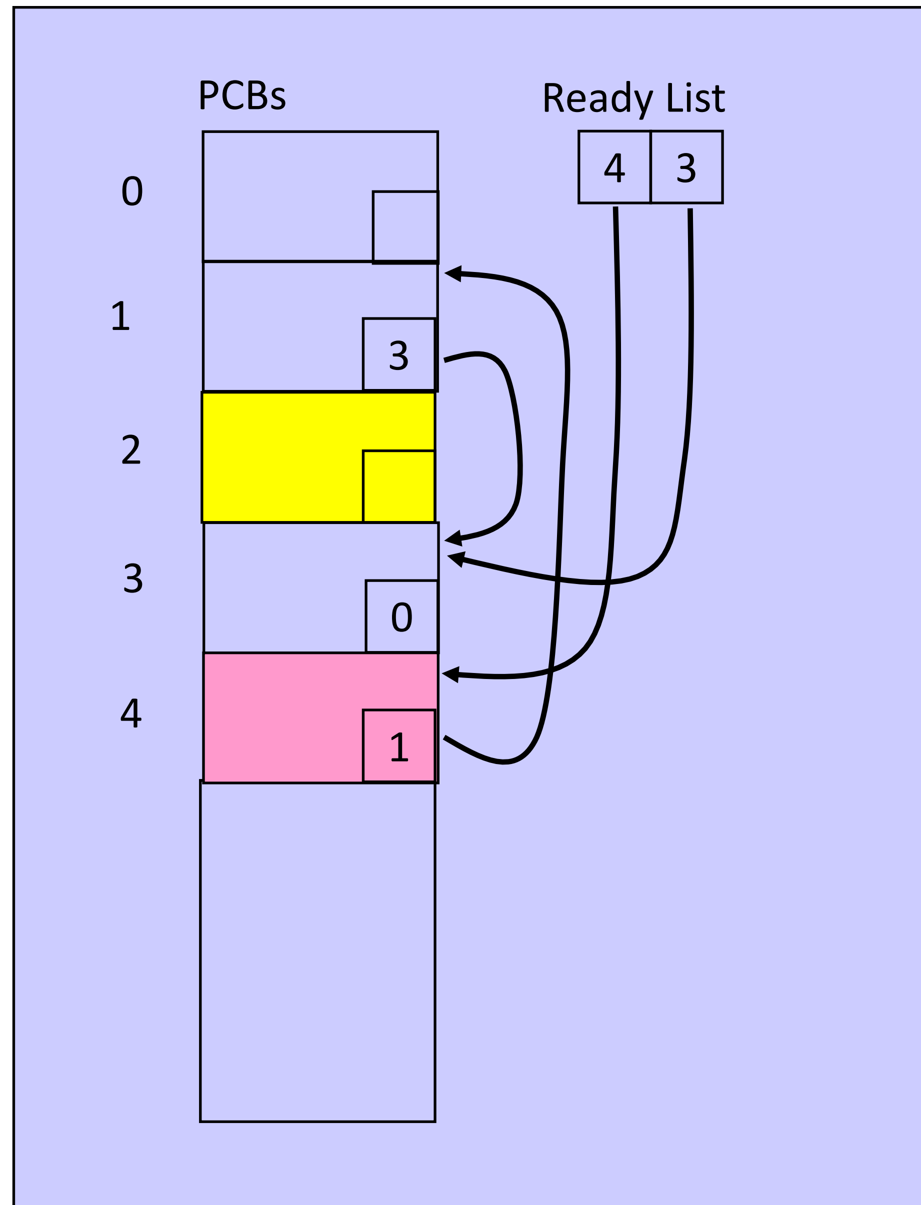
RAM



How did 4 come next after 2?

© Copyright 2019, Peter J Denning





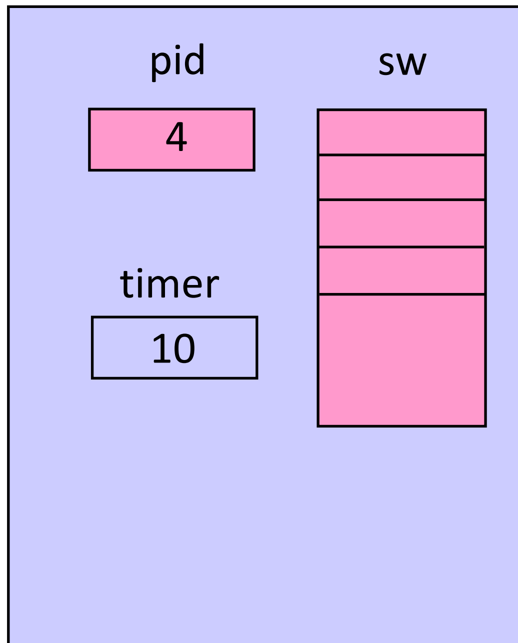
The RL (Ready List) links all processes waiting to run on the CPU

The RL descriptor has H (head) and T (tail) fields

Each PCB has a link field saying which process follows it in RL

© Copyright 2019, Peter J Denning

CPU

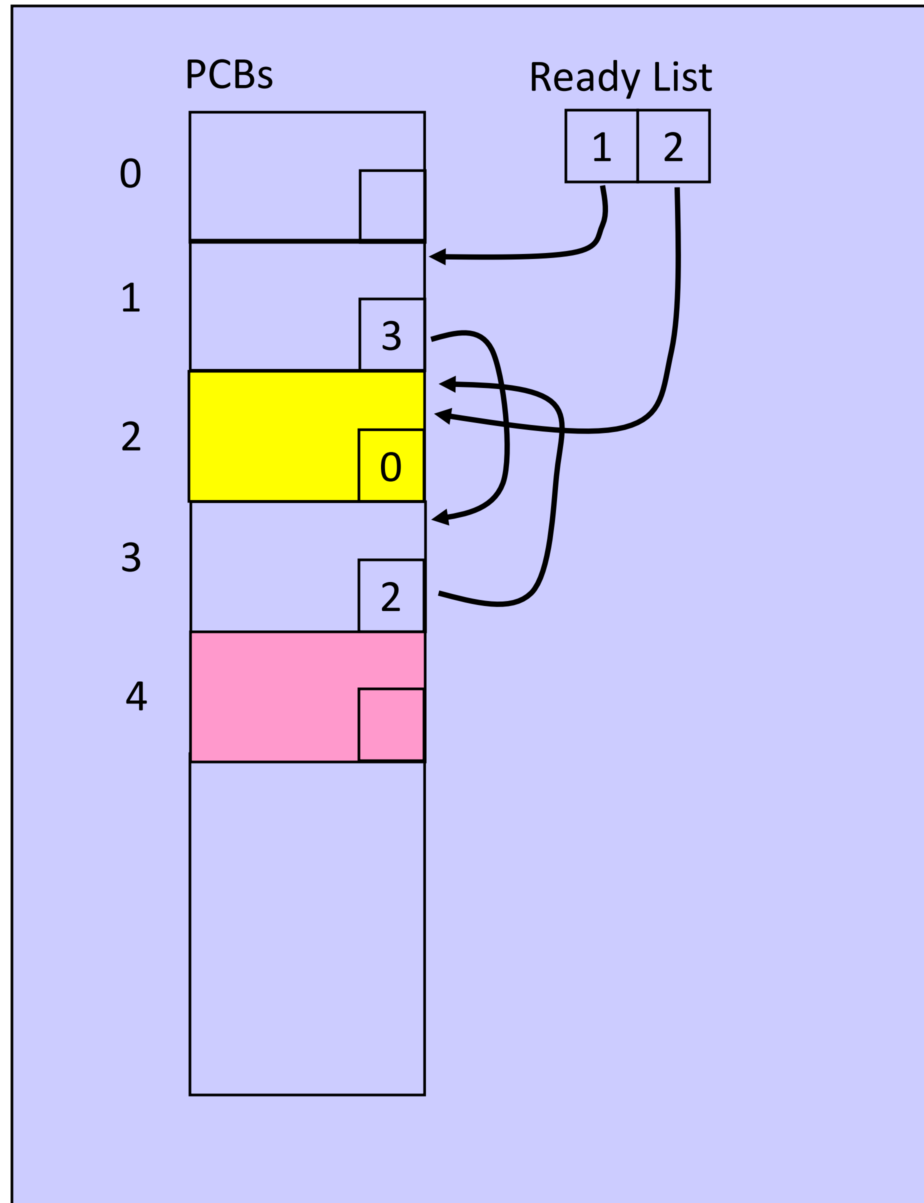


As part of the context switch, the RL.head goes to pid and its successor becomes new RL.head

The old RL.tail gets pid as its successor and pid becomes the new RL.tail

© Copyright 2019, Peter J Denning

RAM



Context Switching

- Save the current CPU stateword to the process's control block.
- Select next process from head of RL and update RL.
- Load that process's stateword into the CPU and start running.

```
SAVESW
  pid=CYCLE-RL(pid)
LOADSW
```

```
CYCLE-RL(A)
  PCB[RL.tail].link=A
  RL.tail=A
  PCB[A].link=0
  B=RL.head
  RL.head=PCB[RL.head].link
RETURN B
```

© Copyright 2019, Peter J Denning

Round Robin Scheduling

- Objective: time slice end interrupt switches CPU to next ready process
- T = time slice = max time until context switch
- Time slide end interrupt activates this routine:

```
disable
SAVESW
set timer = T
pid=CYCLE-RL(pid)
LOADSW
enable
return
```

© Copyright 2019, Peter J Denning

Process 0

- Think of a scenario that leaves RL empty. What happens?
- Our algorithms will leave RL(head,tail)=(0,0). Next context switch goes to process 0.
- Process 0 is a special idling process that runs when there are no others (e.g., a screensaver).
- When a process re-enters RL after wakeup, process 0 will be preempted.