Processor Multiplexing

Peter J. Denning

Thread

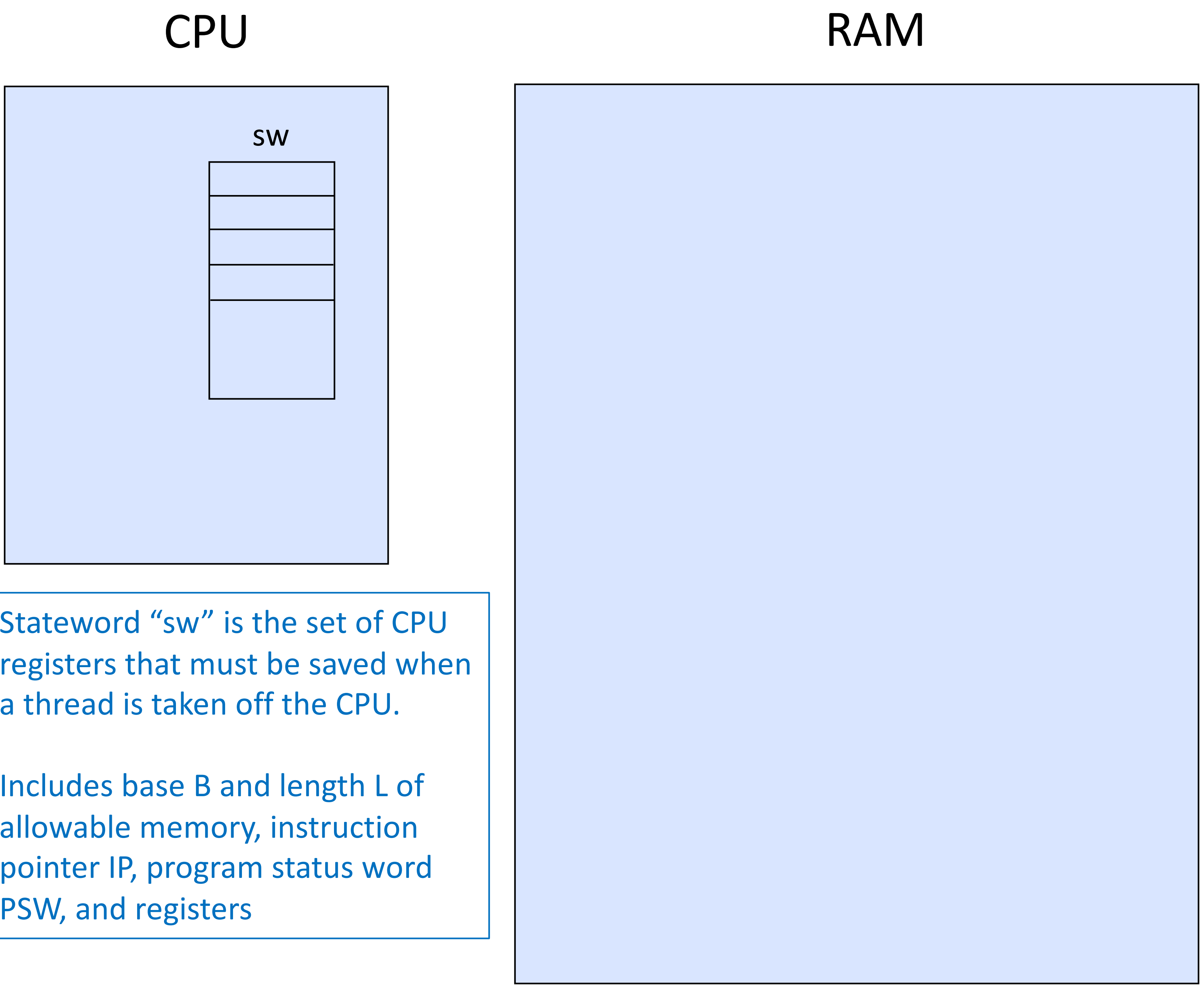
- Trace of instruction pointer through instruction sequence in an address space

Process

- A program in execution on virtual machine with its own address space and simulated (virtual) CPU
- Process always contains at least one thread
- Process can contain many threads, all sharing the same address space

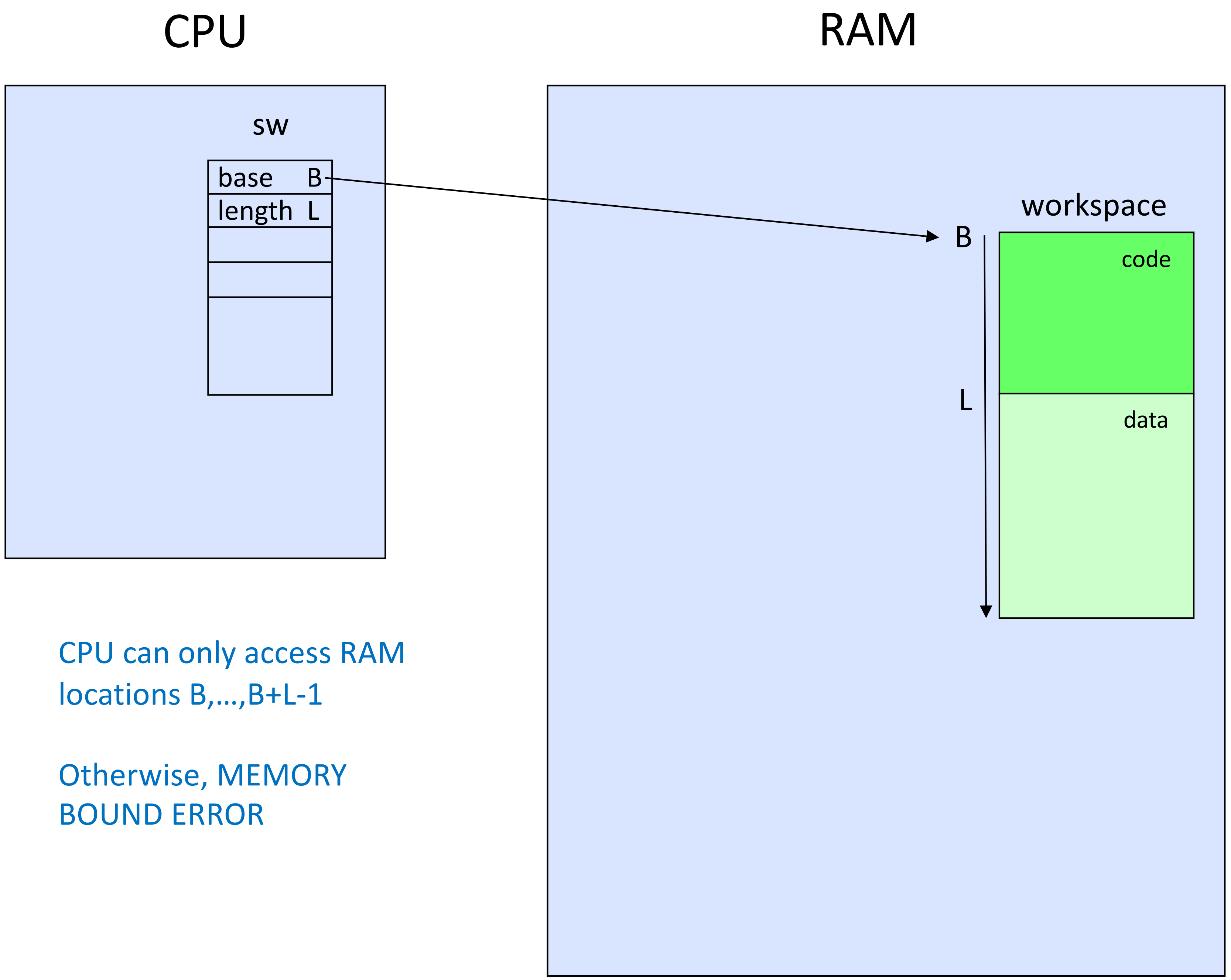
Time Sharing

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



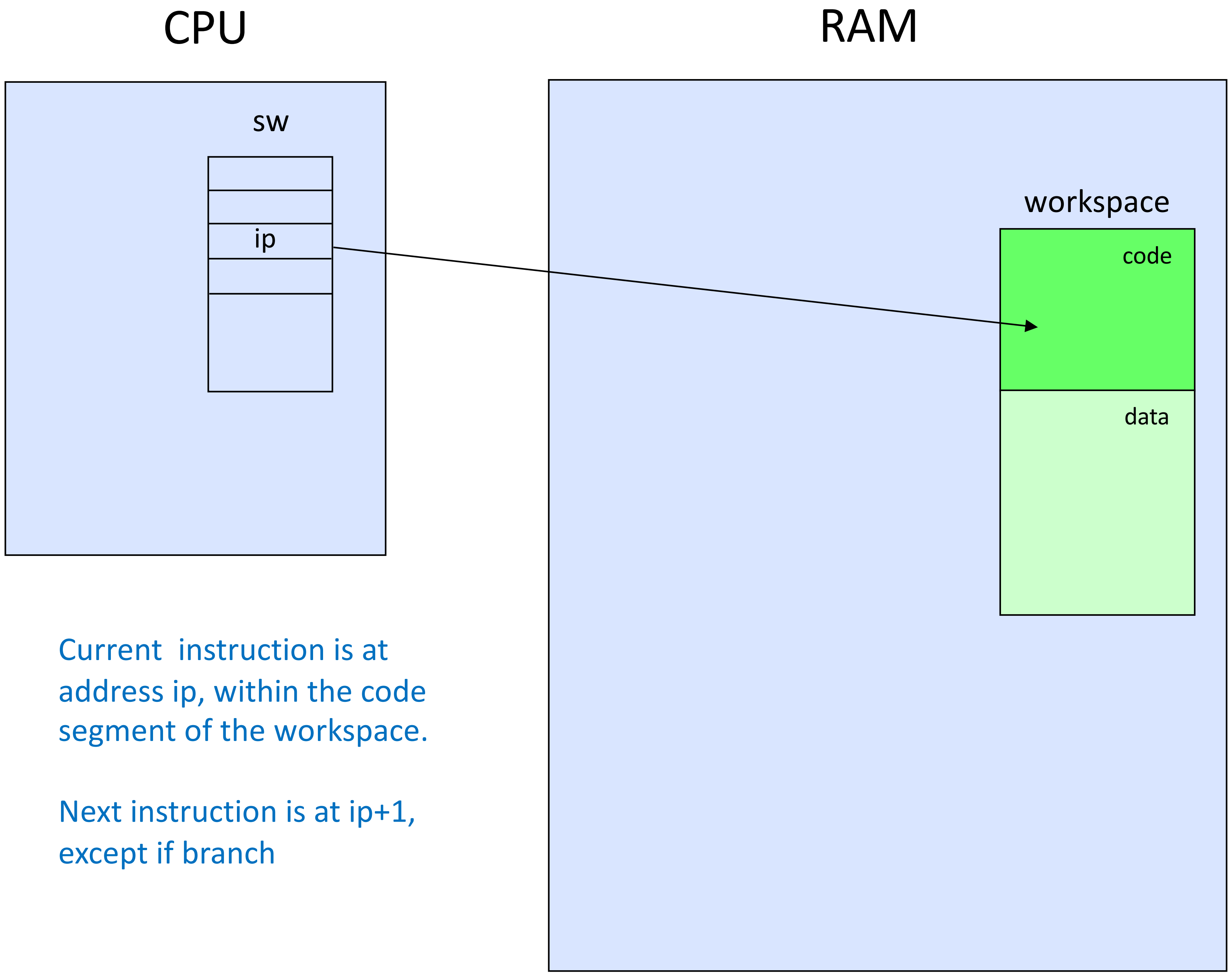
Stateword "sw" is the set of CPU registers that must be saved when a thread is taken off the CPU.

Includes base B and length L of allowable memory, instruction pointer IP, program status word PSW, and registers



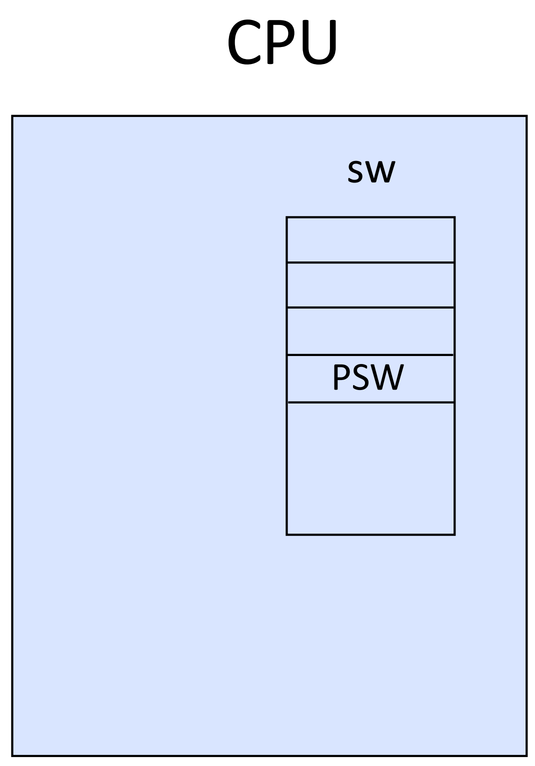
CPU can only access RAM locations B,…,B+L-1

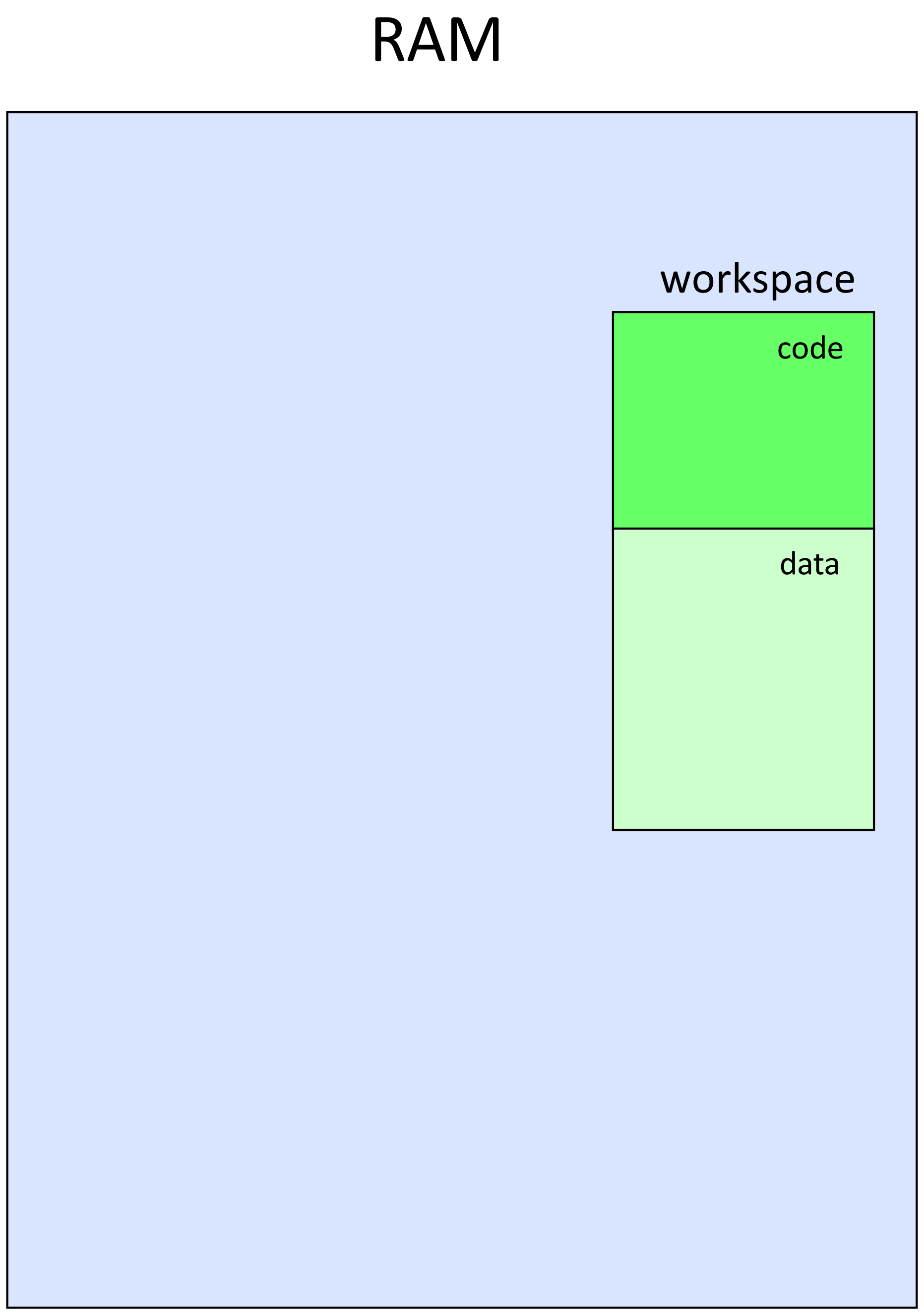
Otherwise, MEMORY BOUND ERROR



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

Next instruction is at ip+1, except if branch

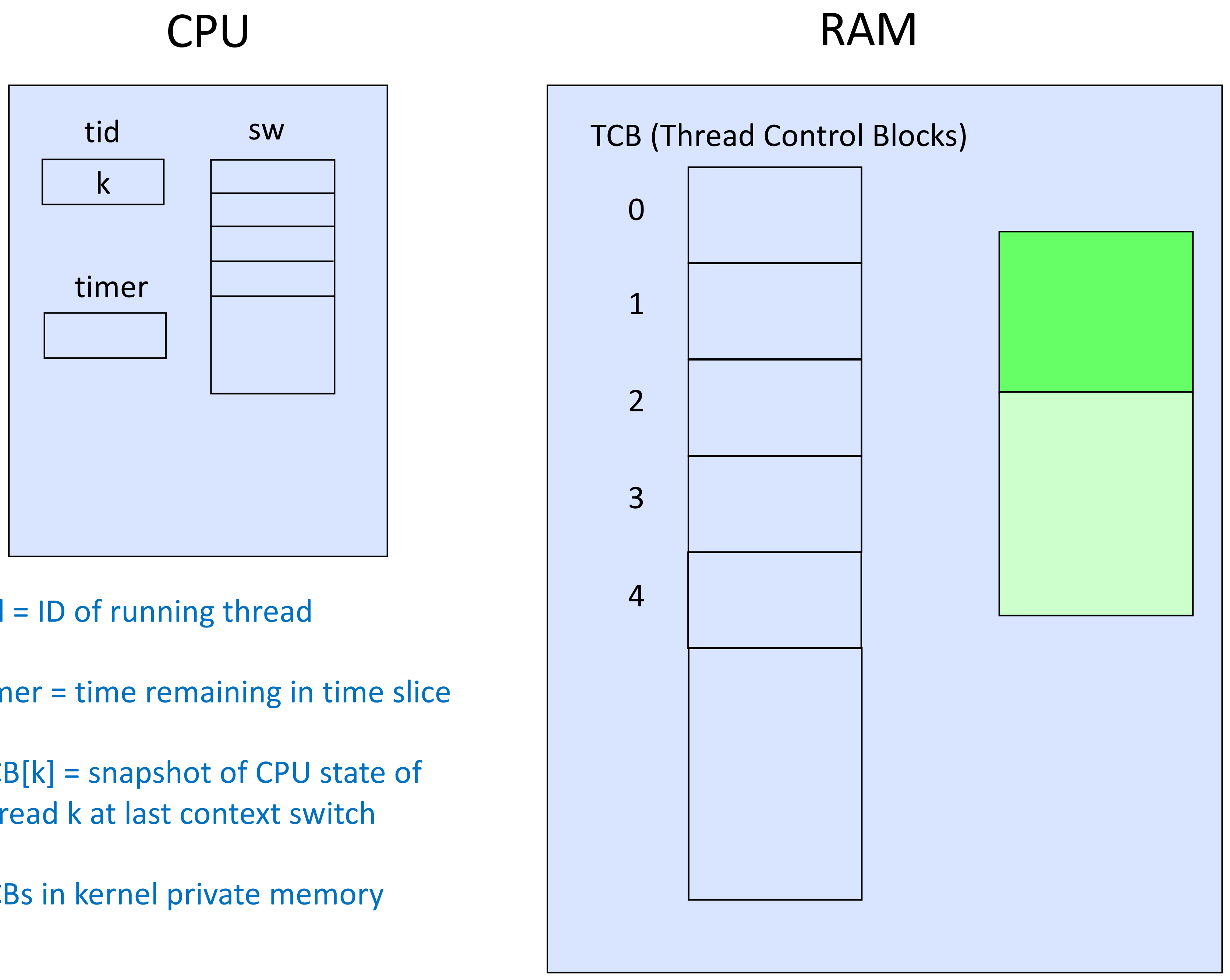




PSW contains CPU control bits.

PSW includes the kernel mode bit: only in kernel mode can CPU access data marked “kernel private” or execution “kernel only” instructions

PSW includes the interrupt mask, which specifies which interrupts the CPU will listen to.

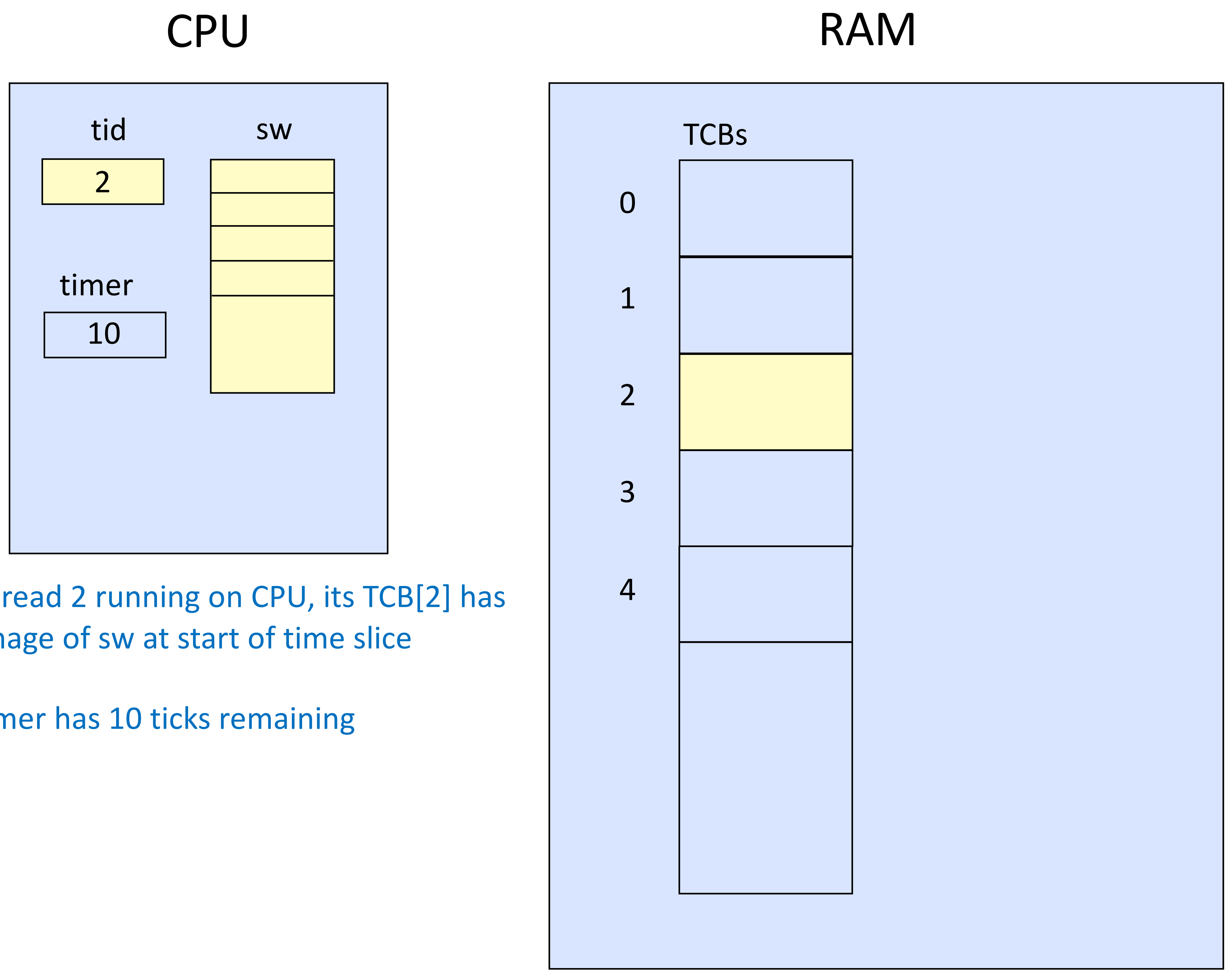


tid = ID of running thread

timer = time remaining in time slice

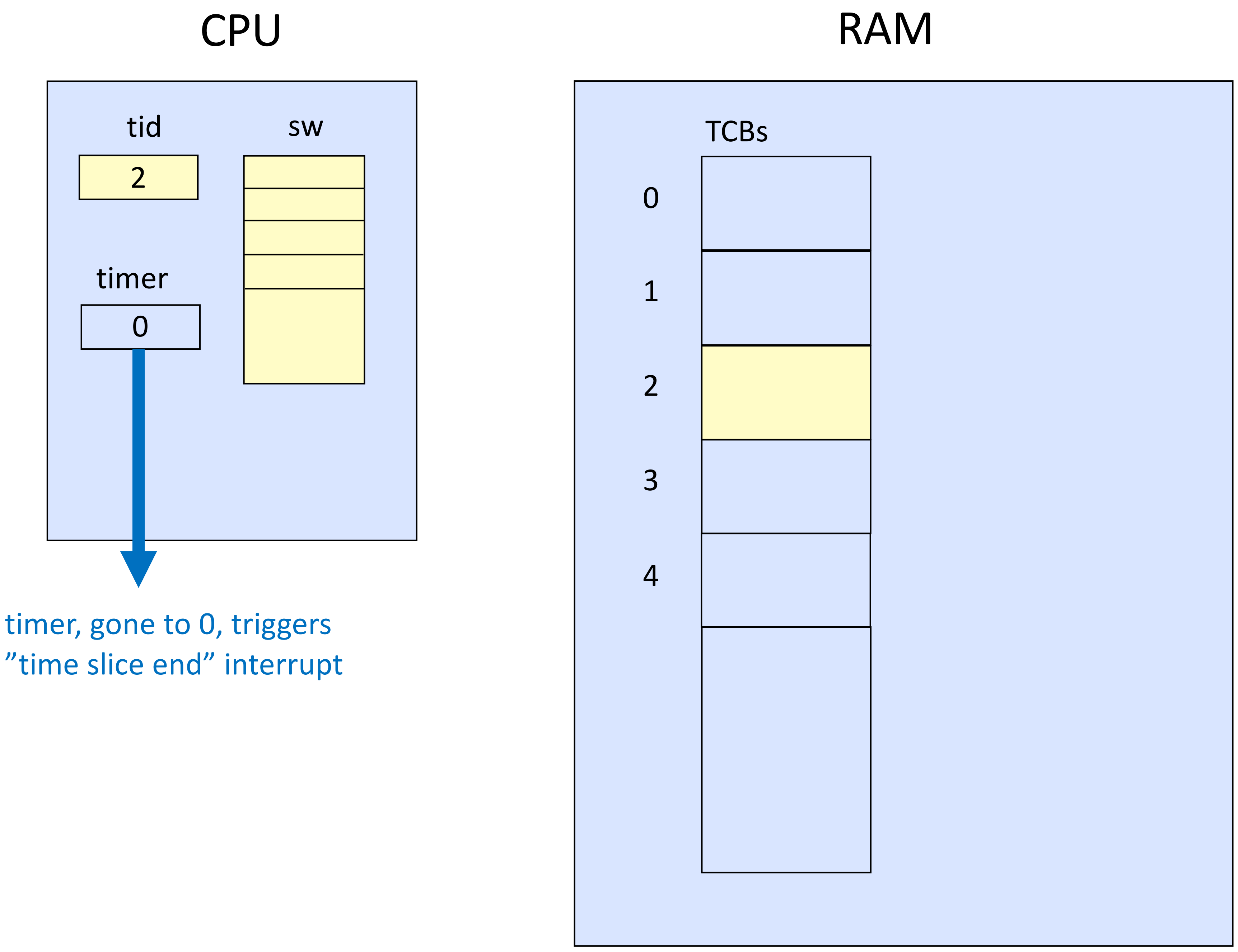
TCB[k] = snapshot of CPU state of thread k at last context switch

TCBs in kernel private memory

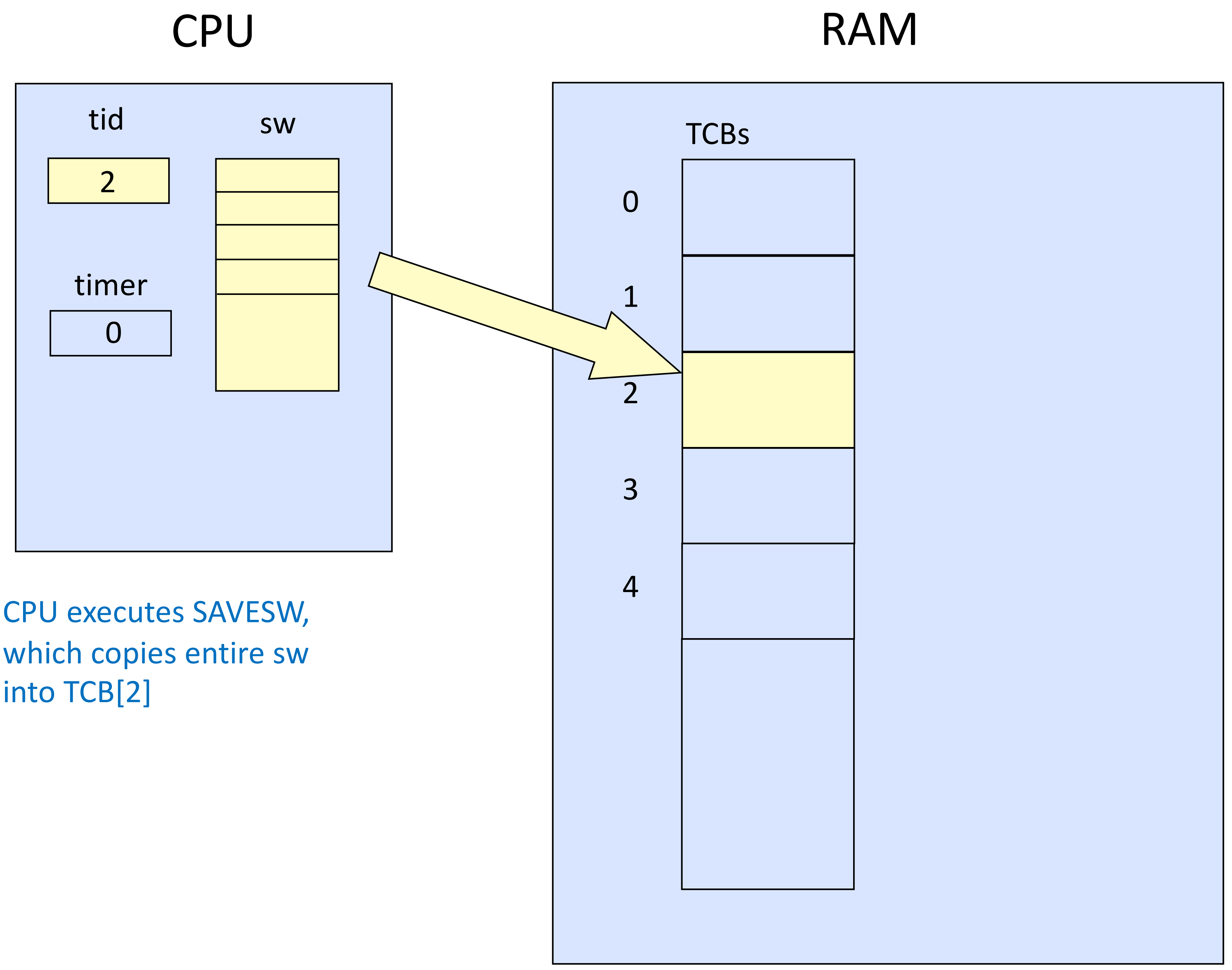


thread 2 running on CPU, its TCB[2] has image of sw at start of time slice

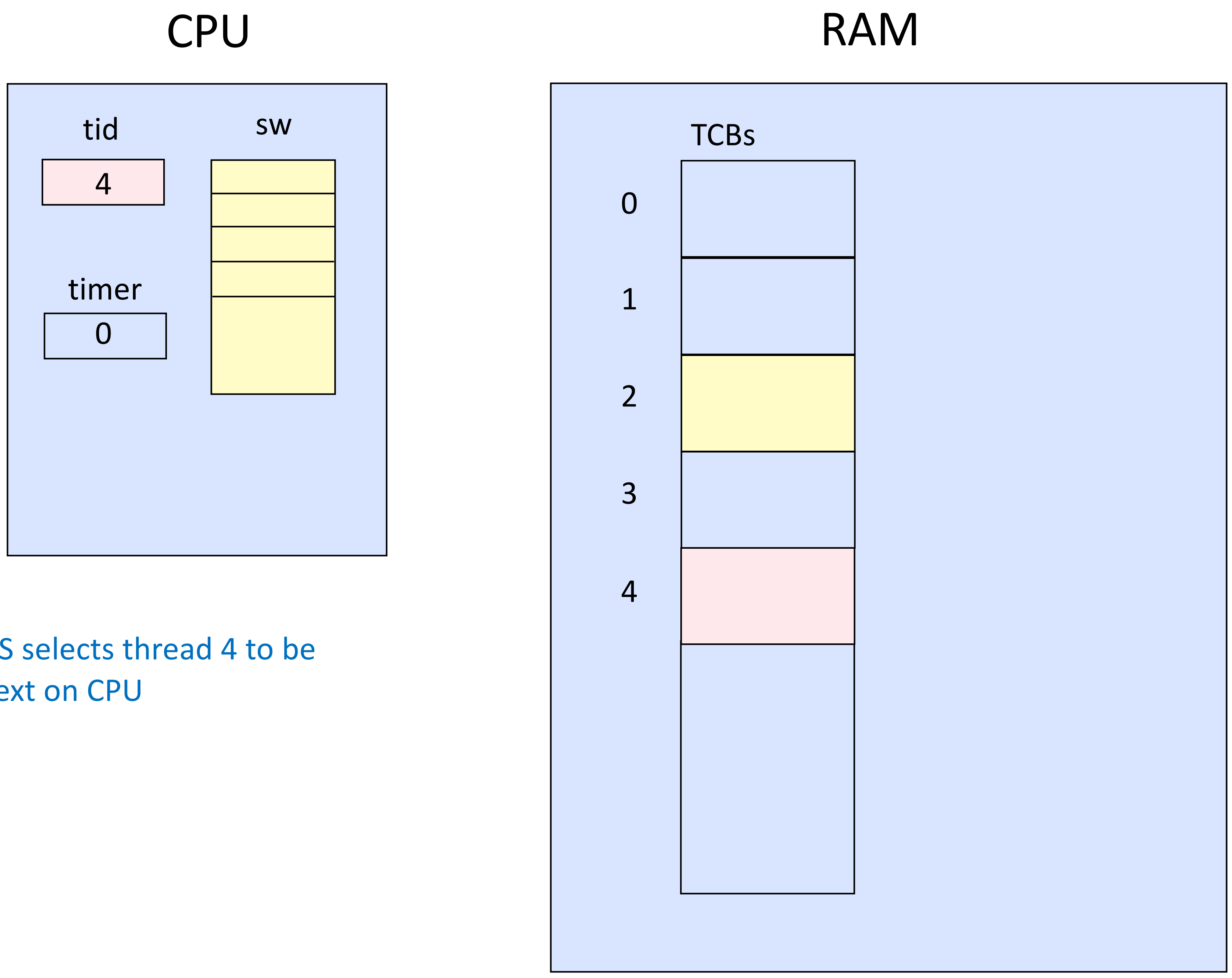
timer has 10 ticks remaining



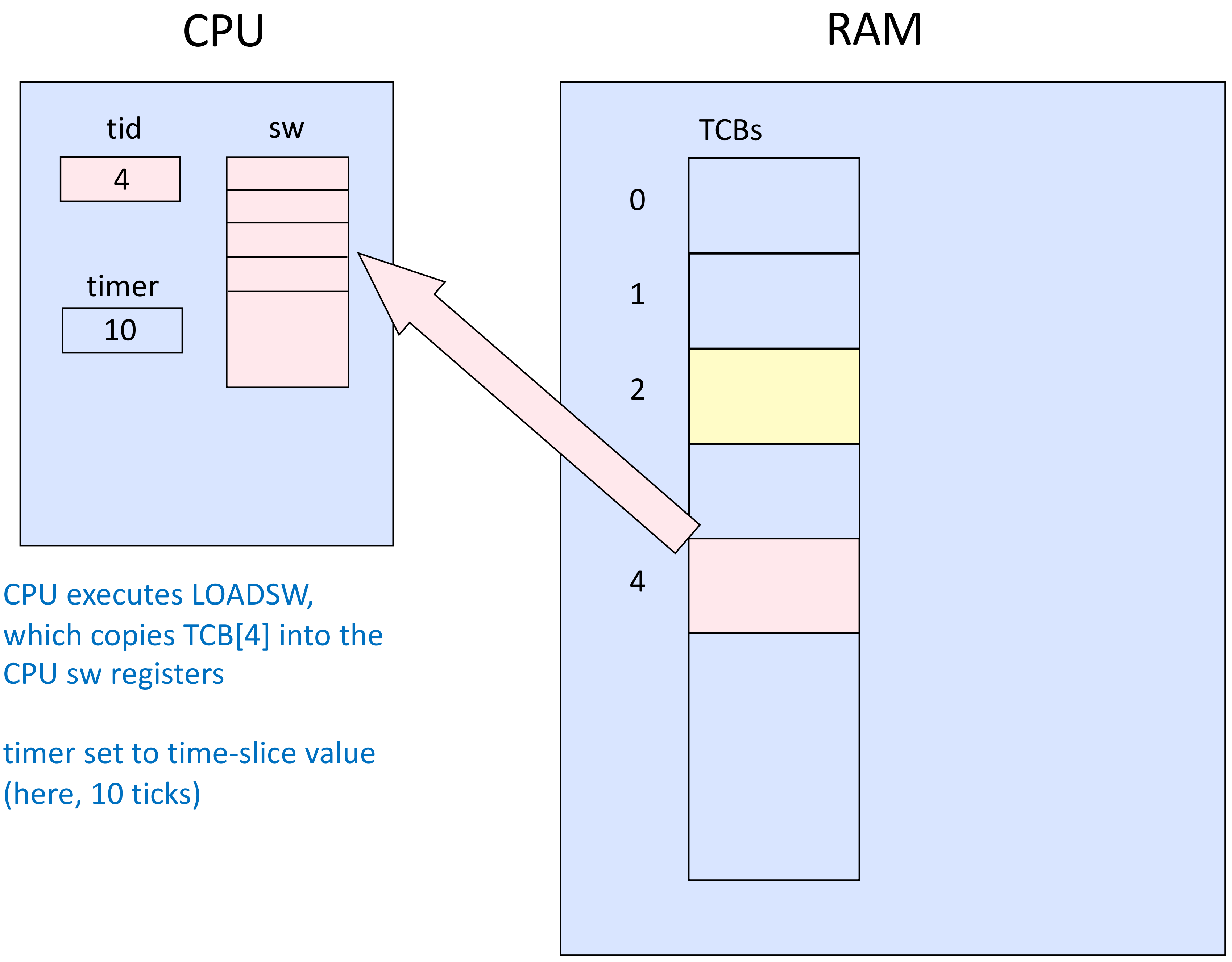
timer, gone to 0, triggers
”time slice end” interrupt



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

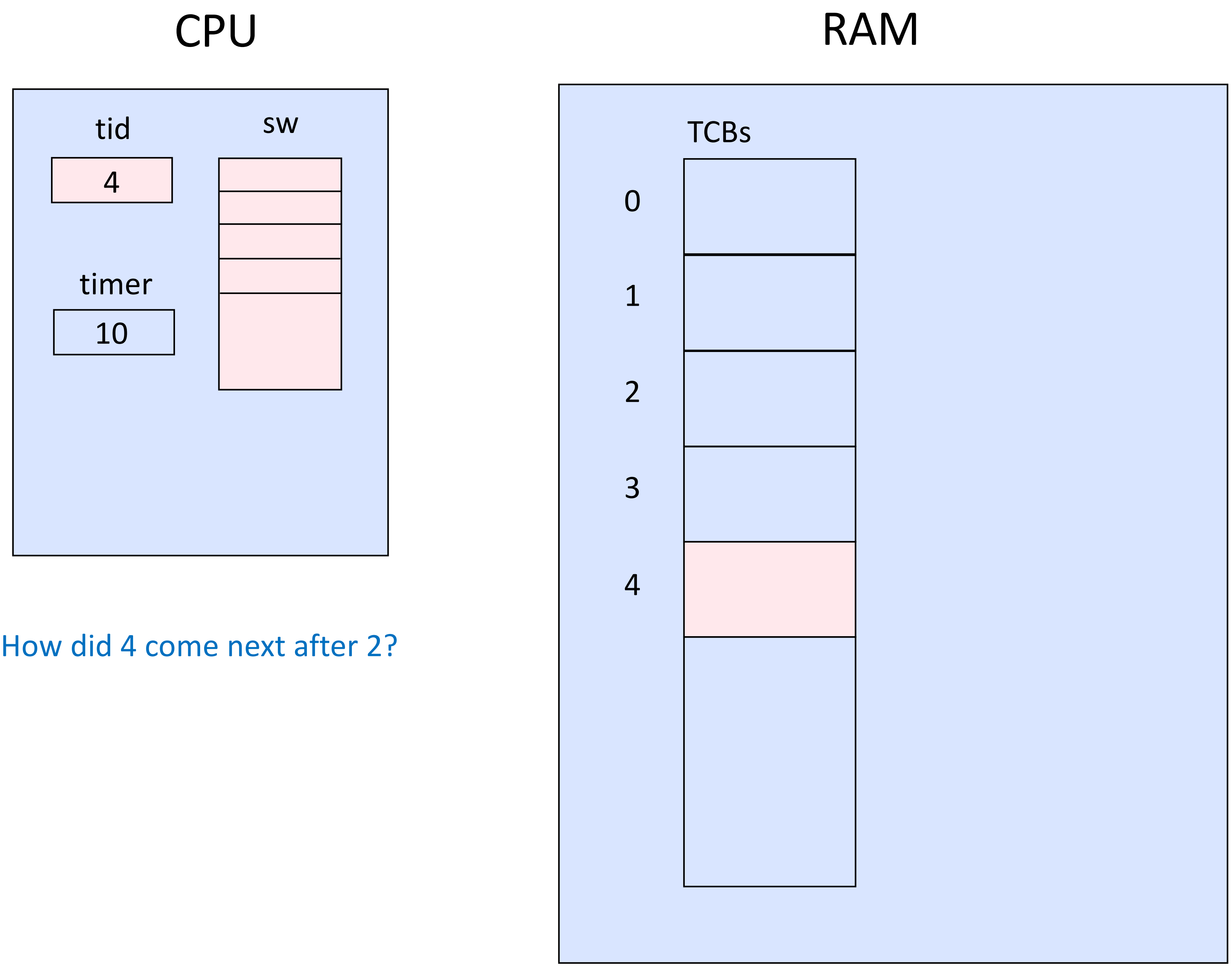


OS selects thread 4 to be next on CPU

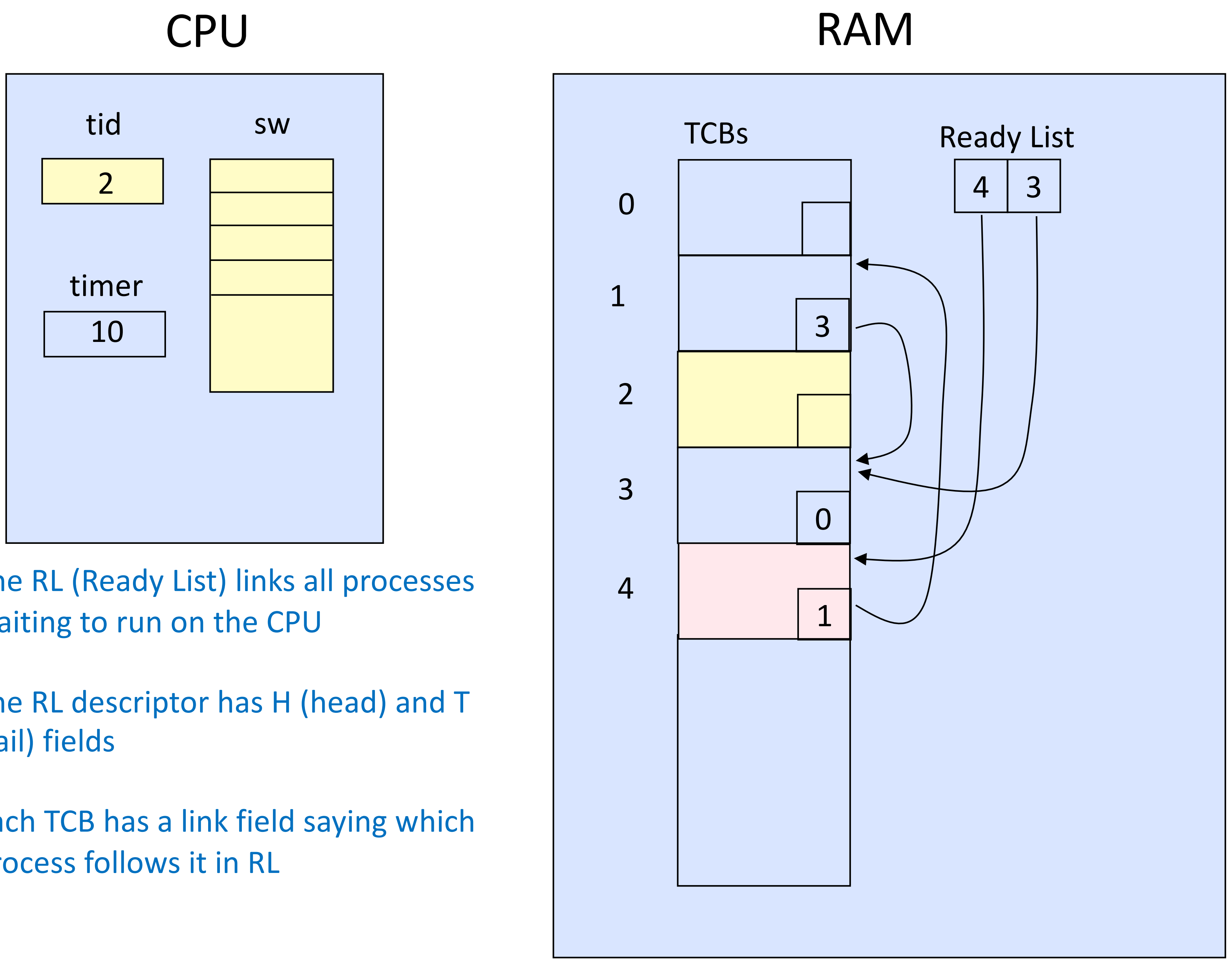


CPU executes LOADSW, which copies TCB[4] into the CPU sw registers

timer set to time-slice value (here, 10 ticks)



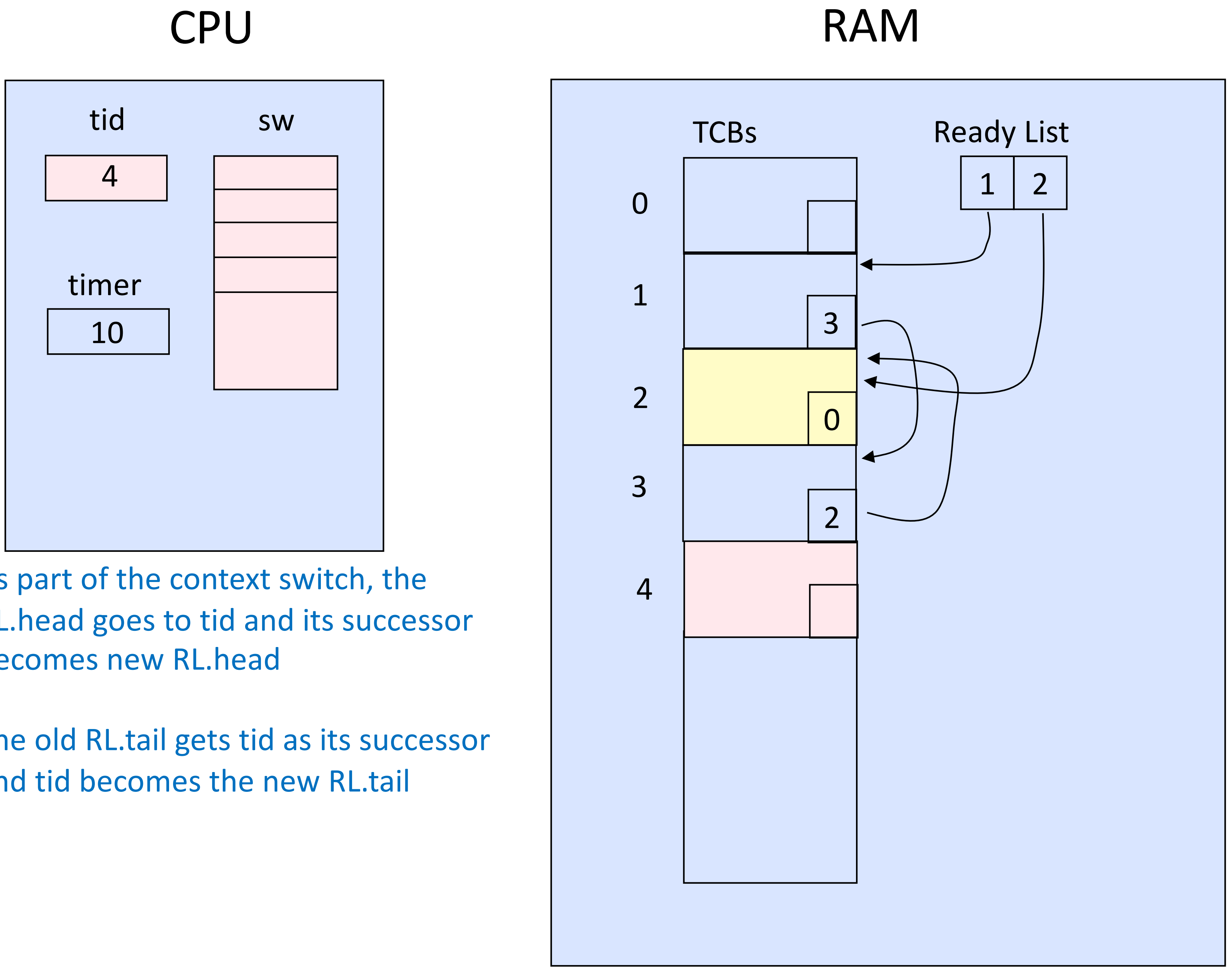
How did 4 come next after 2?



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

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

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



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

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

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
  tid=CYCLE-RL(tid)
LOADSW
```

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

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
tid=CYCLE-RL(tid)
LOADSW
enable
return
```

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.

Summary

- Each thread represented by its CPU stateword in a kernel array of TCBs
- Context switch saves CPU sw and replaces it with the sw of next ready thread
- Timer interrupt cycles the ready list to the next thread
- Process 0, always at end of RL, runs if RL empty