CSE 451: Operating Systems

Lab Section: Week 3

Today

- Last week's quiz
- Project 2
- Scheduling
  (this will hopefully be useful for tomorrow's quiz ☺)

Last week's quiz

1) Define terms: (a) exception, (b) fault,
(c) interrupt, (d) trap

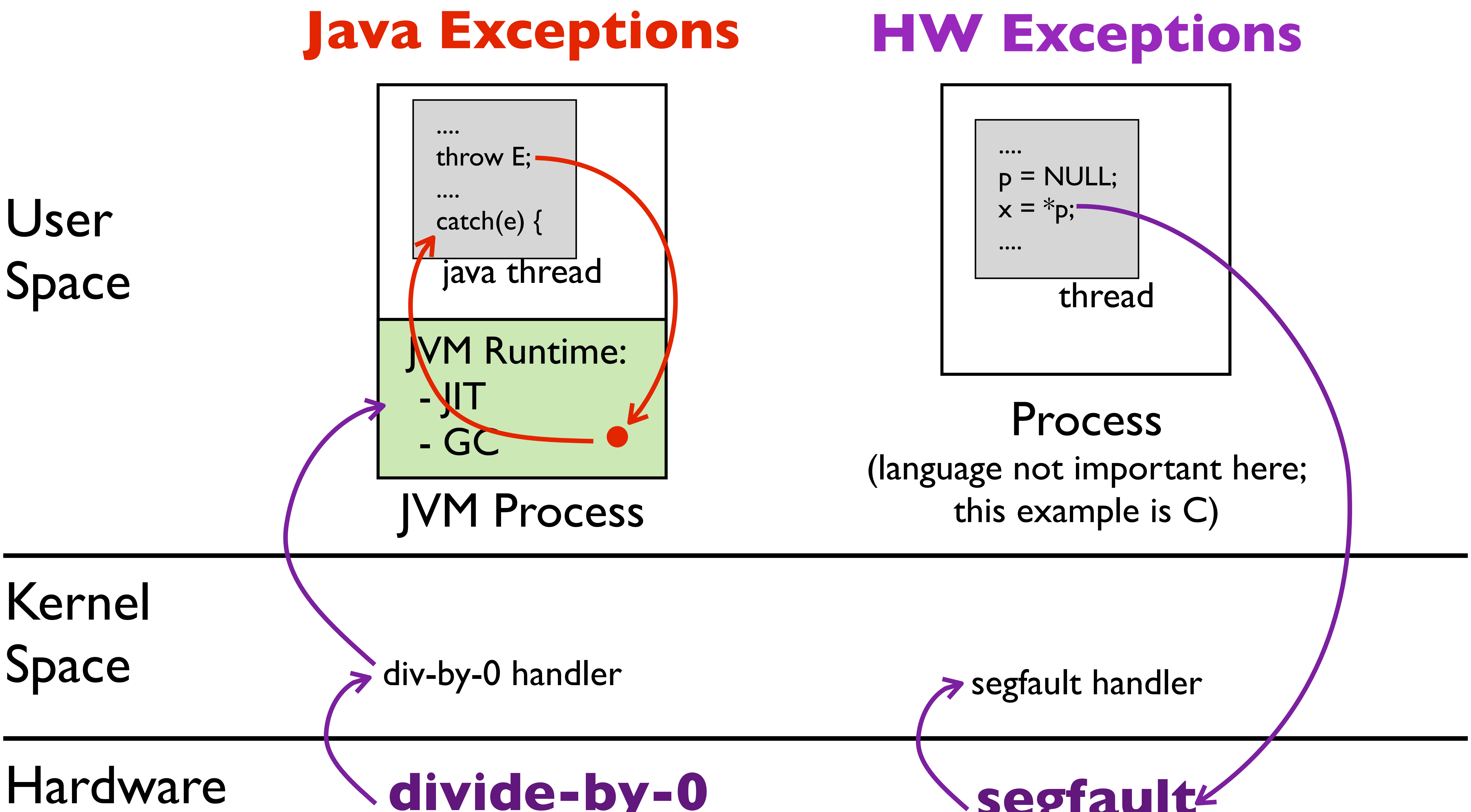
- see answer in slides from last week

Two reasons for losing points:

- didn't mention system calls!
- these are different from Java exceptions ...

Last week's quiz:

our "exceptions" cross user/kernel/hw boundary



Last week's quiz

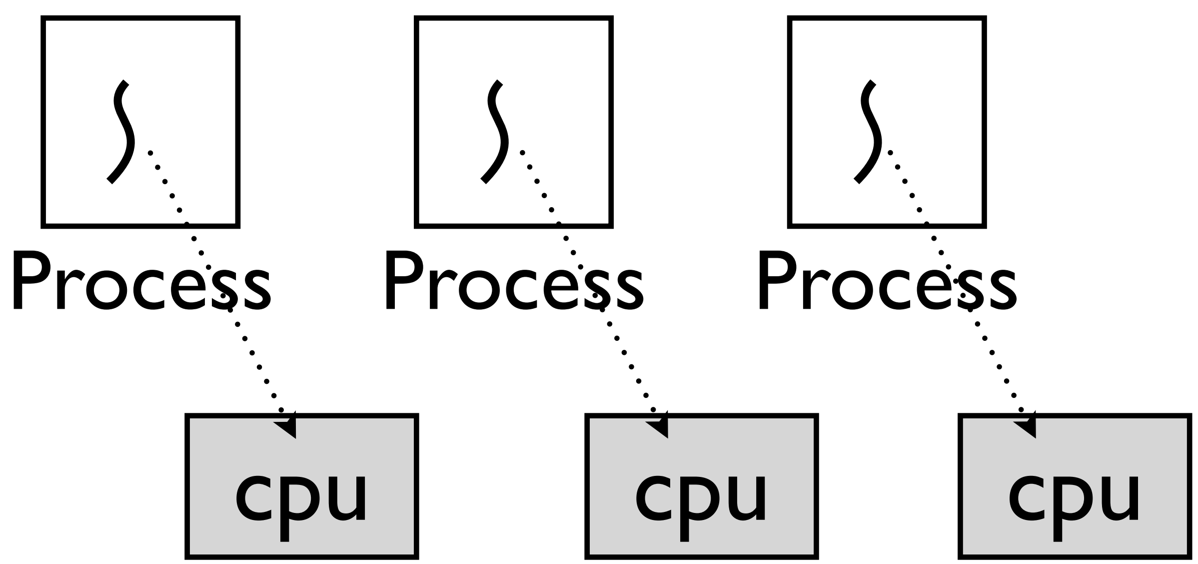
2) Explain how kernel/user modes are related to privileged machine instructions
    - everyone got this right!

Last week's quiz

3) What's the importance of separating threads from processes?

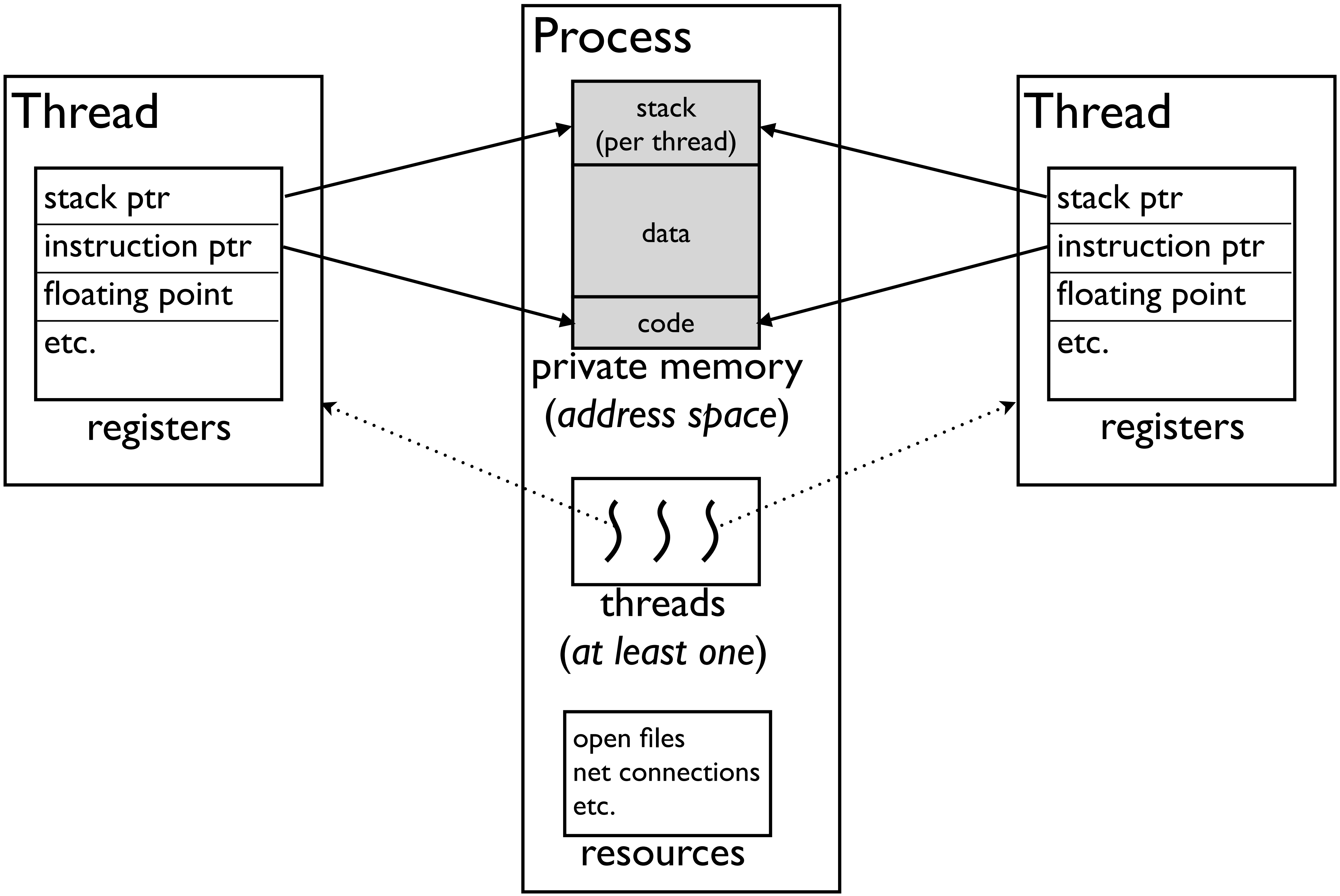
✘ .... concurrency?

We don't need threads for concurrency!



✔ **sharing**

Threads share Process' resources



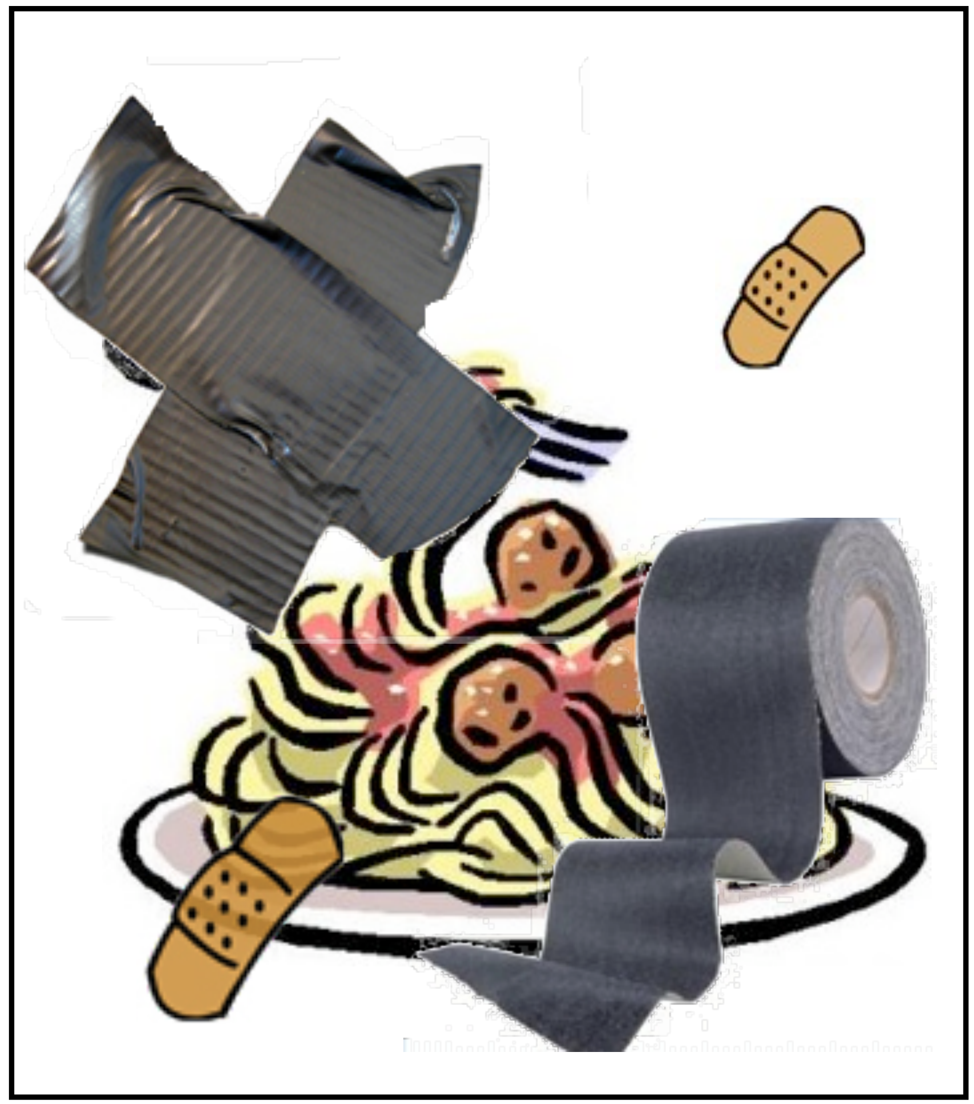
Project 2

- Due January 26, 11:59 pm
  - same time as Project 1
  - you can resubmit until then
- Questions?

Today

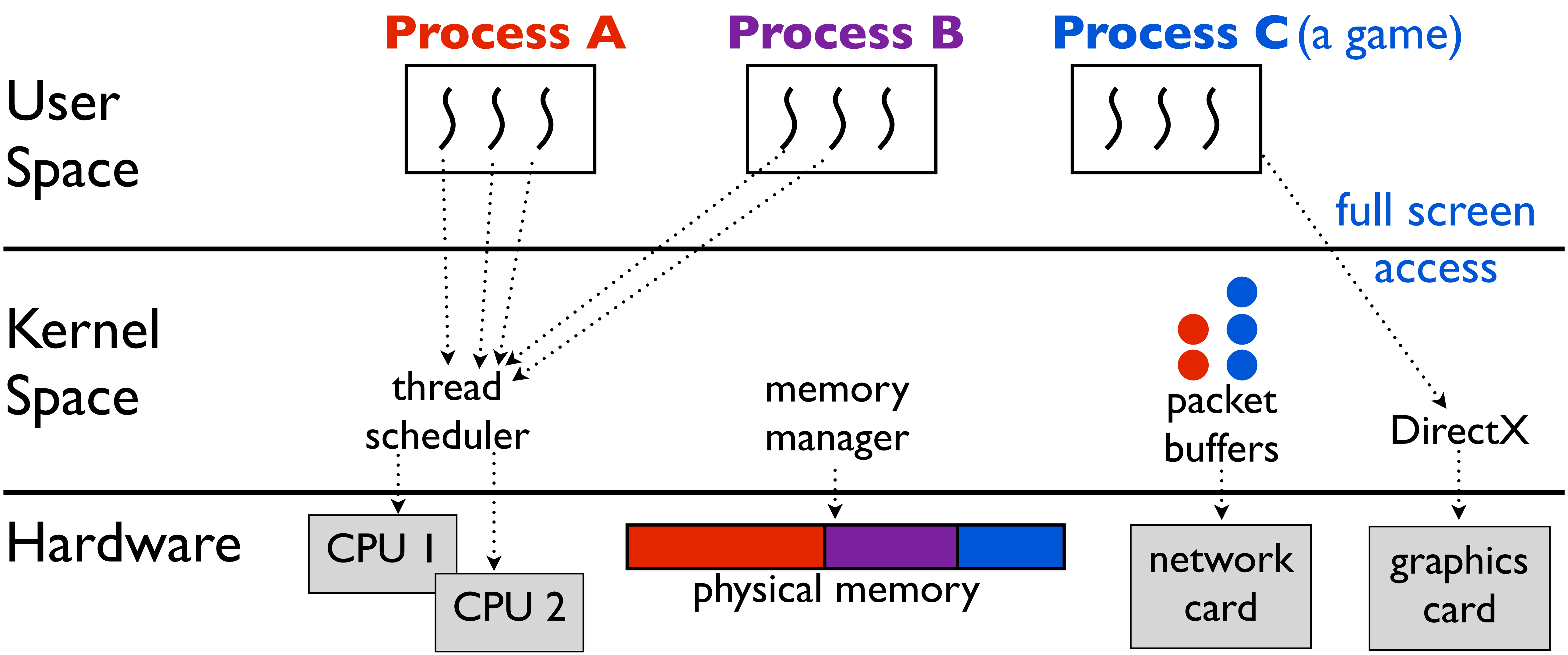
- ~~Last week’s quiz~~
- ~~Project 2~~
- Scheduling
  (this will hopefully be useful for tomorrow’s quiz ☺)

What real schedulers look like



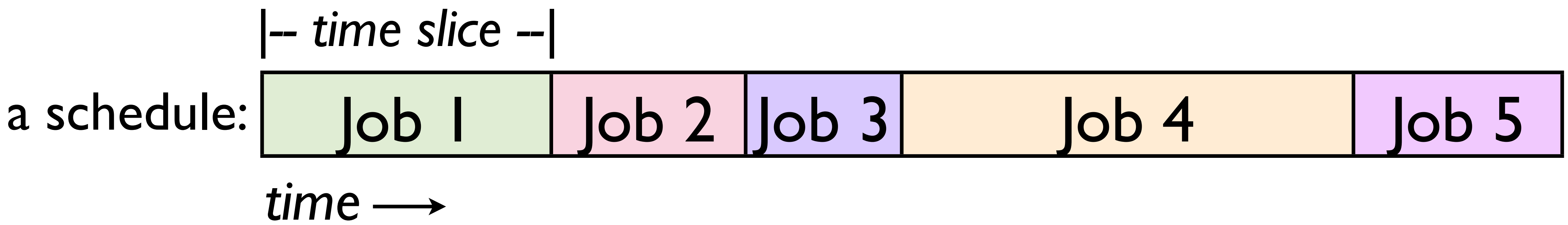
scheduler.c

Scheduling happens throughout the kernel



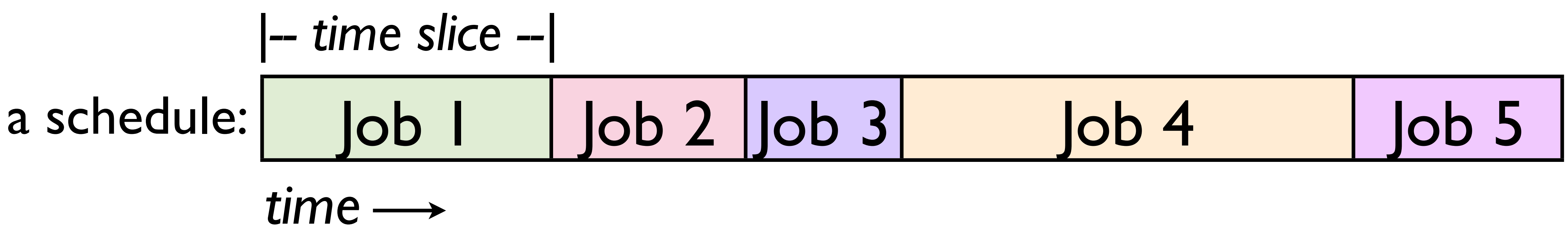
We'll focus on CPU scheduling

CPU scheduling in the abstract



- What might we want in a good schedule?
  - fairness (every job gets a “fair” slice)
  - priority (some jobs are more important)
  - deadlines (some jobs must finish by a certain time)
  - thread locality
  - ....

CPU scheduling in the abstract



- Practical issues
  - new jobs are starting all the time
  - how do we know how long a job will take?

    (can't plan ahead very far)

Two decisions a scheduler makes

- When do I reschedule the CPU?
  (i.e., how long is the next time slice?)
- Who gets the CPU next?

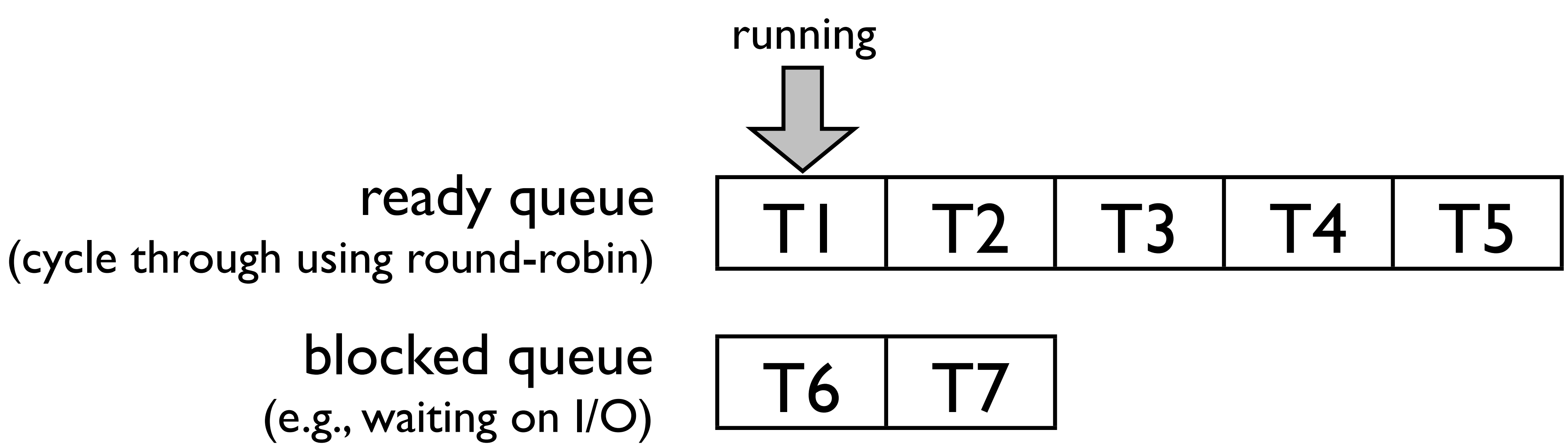
When do I reschedule the CPU?

- Cooperative scheduling ✔ Batch schedulers
  - reschedule when:
    ... a thread blocks on I/O
    ... a thread calls yield()
    ... a thread finishes
  - problem:
    ... must rely on threads to relinquish CPU (fairness)
- Preemptive scheduling ✔ Interactive schedulers
  - can reschedule at *any* time
  - usually at timer interrupts

Who gets the CPU next?

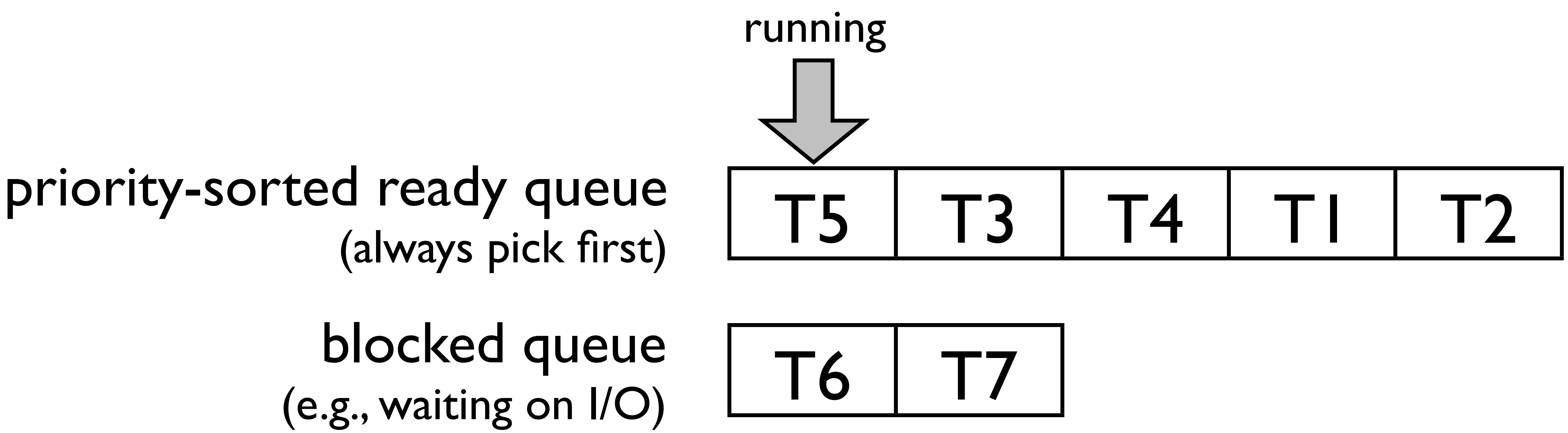
- Many algorithms ...
  Let's look at a few simple single-CPU algorithms

- Round robin order



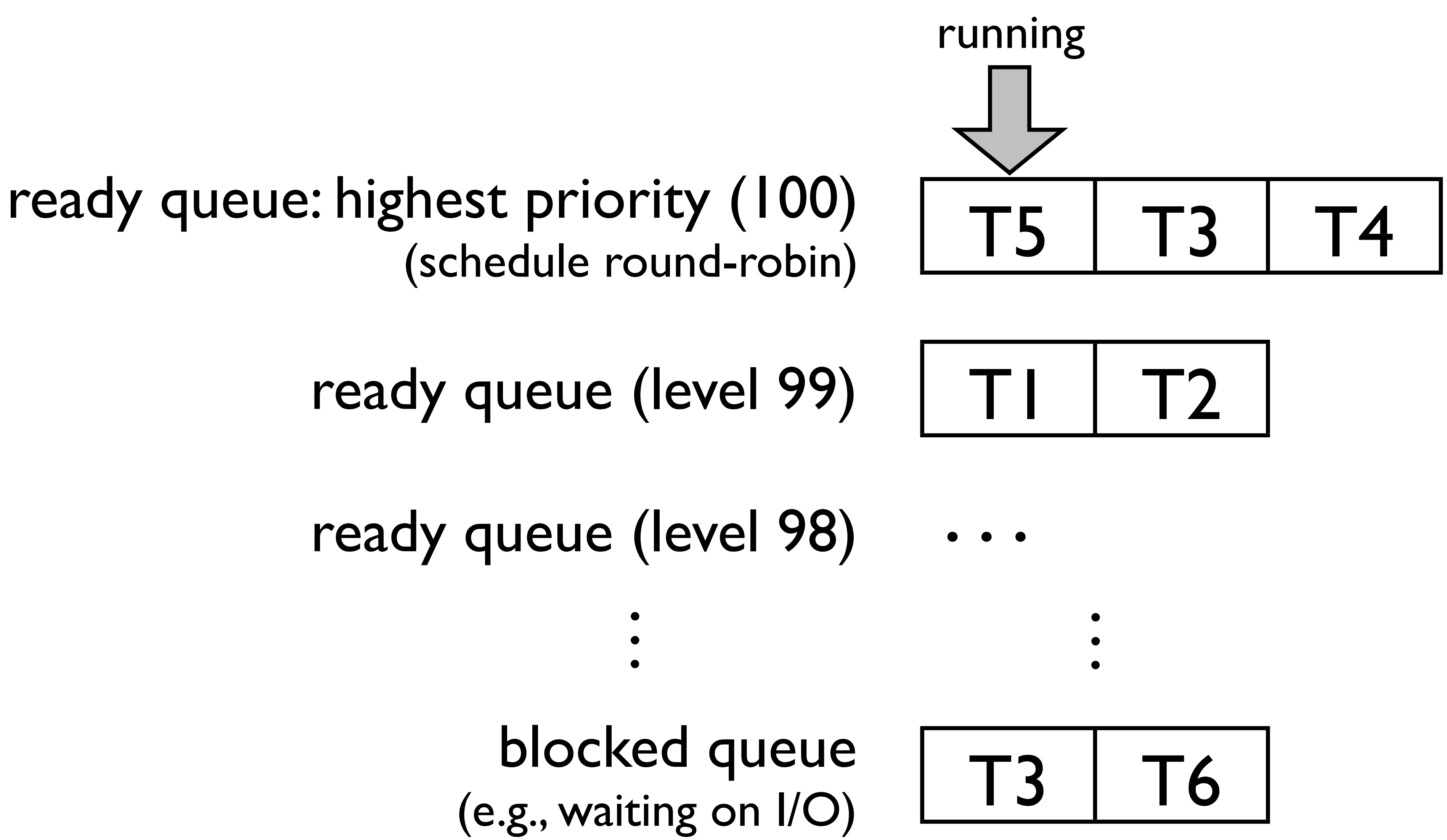
Who gets the CPU next?

- Priority order



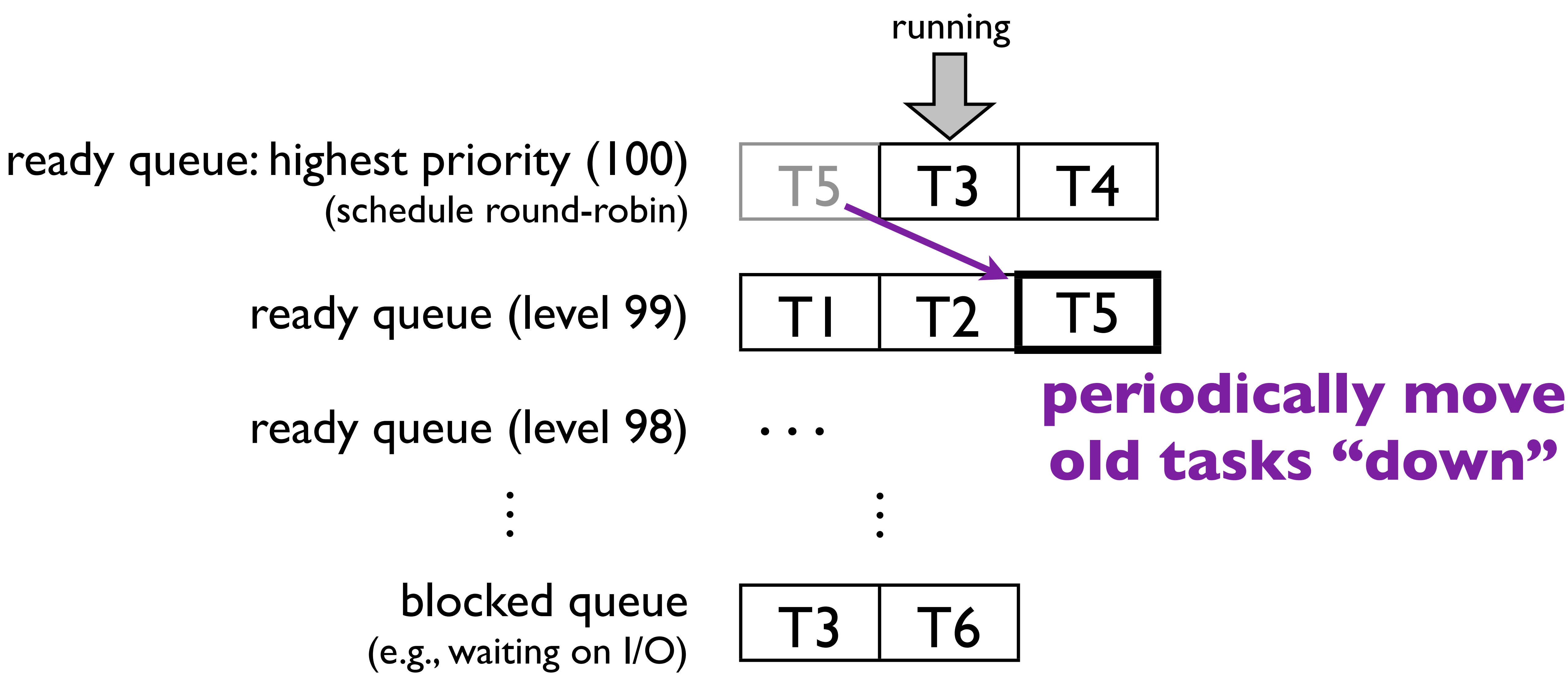
Who gets the CPU next?

- Multi-level Feedback Queues

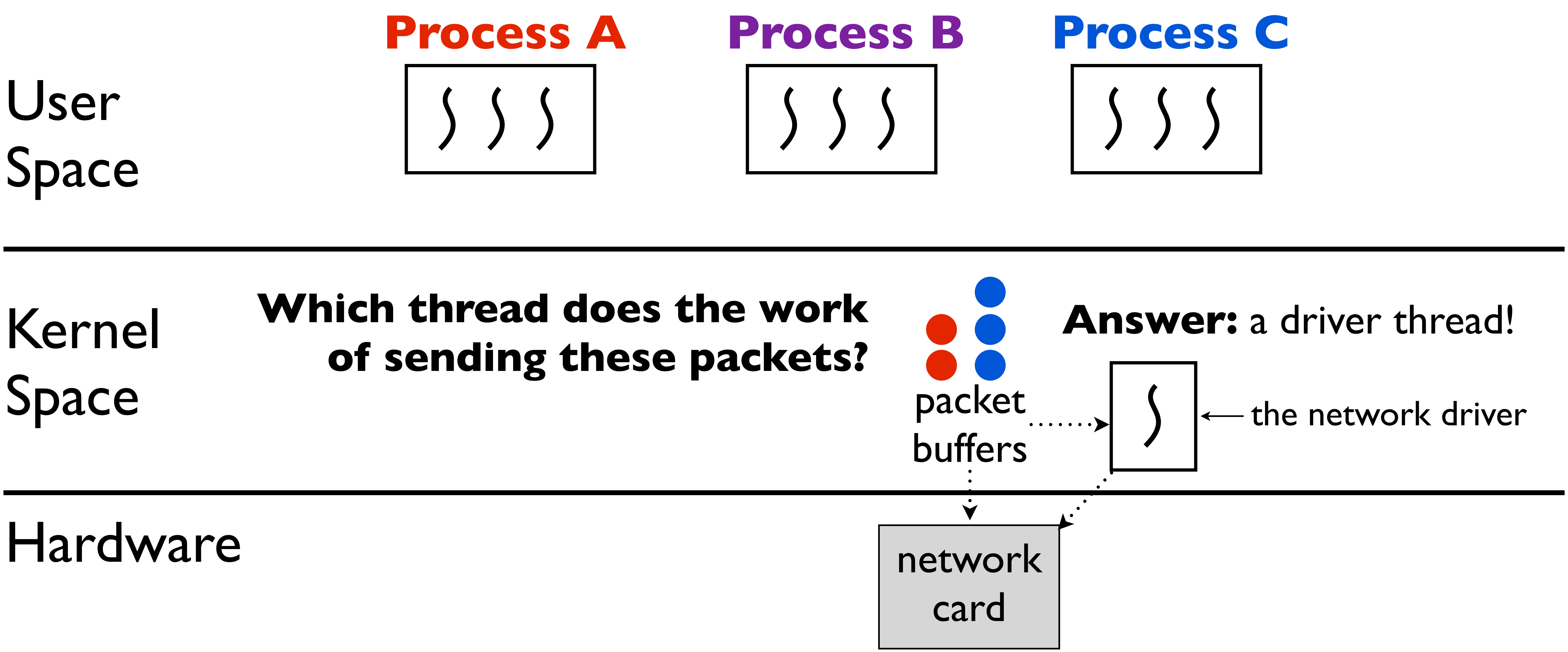


Who gets the CPU next?

- Multi-level Feedback Queues



The kernel needs CPU time too!



Driver threads get scheduled like any other thread