Exercise walk-through

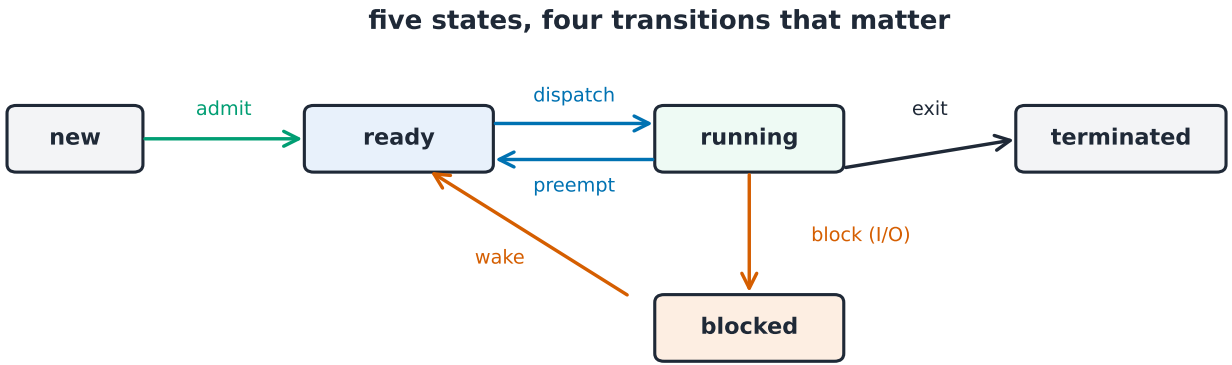
# 7.1
# Processes and threads

University Computer Science

# Processes and threads: five states, one PCB

1. **States** New, **ready**, **running**, **blocked**, terminated - dispatch and preempt between ready and running, block on I/O, wake on completion.
2. **Switch** The PCB saves registers, PC, page-table pointer, open files: 2000 bytes at 2 GB/s is 1 us, plus cache pollution.
3. **Threads** A thread per connection: about 50 us create-and-switch against 500 us as processes - sharing is the point and the danger.

# Process state diagram




blocked waits on a device; ready waits only for the CPU — the PCB carries everything a suspend must save

Five **states**, four transitions: dispatch, preempt, block, wake.

# Exercises 2.1·2.3

**2.1** A process waits for a disk read. Name its state, and the state it enters when the read completes. [3]

**2.2** Name the three things a thread owns that its process does not re-own per thread. [3]

**2.3** Compute the PCB switch time for 5000 bytes at 2.5 GB/s. [3]

# Exercises 2.1·2.3

**2.1** A process waits for a disk read. Name its state, and the state it enters when the read completes. [3]

**2.1**

- Blocked; it enters ready when the read completes ✓.

---

**2.2** Name the three things a thread owns that its process does not re-own per thread. [3]

**2.2**

- Its own stack, its own registers, its own program counter ✓.

---

**2.3** Compute the PCB switch time for 5000 bytes at 2.5 GB/s. [3]

**2.3**

- $5000/2.5 \times 10^{9} = 2$ µs ✓.

# Exercise 3.1

**3.1** A process runs on a machine where the timer interrupts every 10 ms; a disk read it issues takes 4 ms, and the switch costs 1 µs.

Trace the state changes from the moment the read is issued until the process runs again, naming each transition and its cause. Compute the CPU time lost to the switch as a percentage of the 4 ms wait. State why the faulting instruction can simply restart, and what would be lost if it could not. [8]

# Exercise 3.1

**3.1** A process runs on a machine where the timer interrupts every 10 ms; a disk read it issues takes 4 ms, and the switch costs 1 µs.

Trace the state changes from the moment the read is issued until the process runs again, naming each transition and its cause. Compute the CPU time lost to the switch as a percentage of the 4 ms wait. State why the faulting instruction can simply restart, and what would be lost if it could not. [8]

**3.1**

- Running → blocked (read issued) ✓.
- Scheduler dispatches another process (ready → running elsewhere) ✓.
- Read completes: blocked → ready (wake) ✓.
- Dispatcher picks it: ready → running; the instruction restarts ✓.
- Switch cost 1 µs of a 4 ms wait = 0.025% ✓.
- The instruction is restartable because the fault happens before any visible side effect; otherwise preemption would lose or duplicate work ✓.

# Exercise 3.2

**3.2** A server handles each connection with a thread (create plus switch 50 µs) or a process (500 µs).

Compute the setup cost for 20 connections under each design. Give one shared resource that makes threads cheaper to run, and the single failure that makes them riskier. State which design isolates a crash and why. [7]

# Exercise 3.2

**3.2** A server handles each connection with a thread (create plus switch 50 µs) or a process (500 µs).

Compute the setup cost for 20 connections under each design. Give one shared resource that makes threads cheaper to run, and the single failure that makes them riskier. State which design isolates a crash and why. [7]

**3.2**

- Threads: $20 \times 50$ µs $= 1$ ms ✓.
- Processes: $20 \times 500$ µs $= 10$ ms ✓.
- Shared: the address space (code and cache) – no per-connection copy ✓.
- Riskier: one thread's crash takes the whole process – one address space ✓.
- Processes isolate: separate spaces mean a crash cannot corrupt another connection's state ✓.