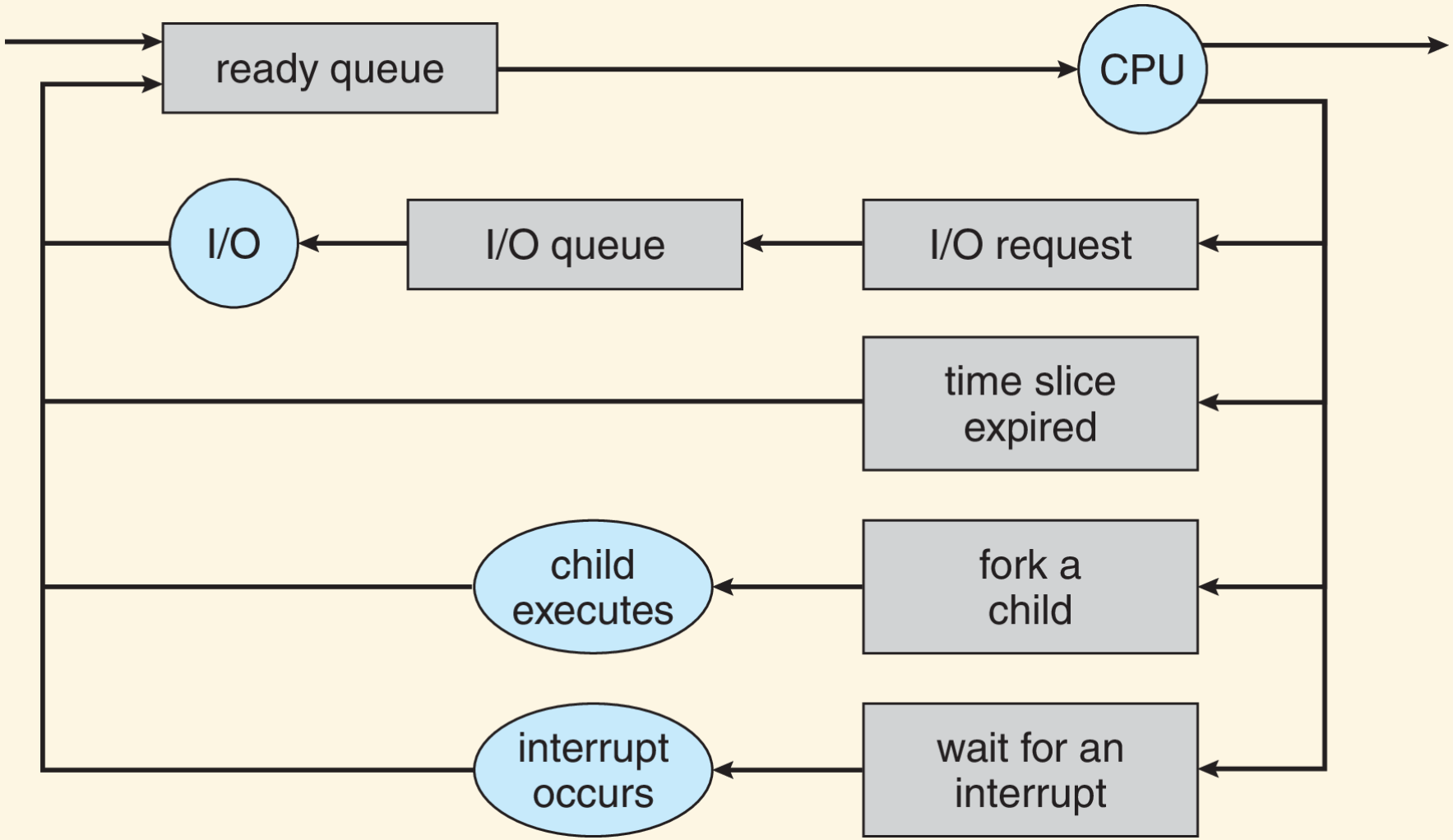
CIS 452 - Operating Systems Concepts

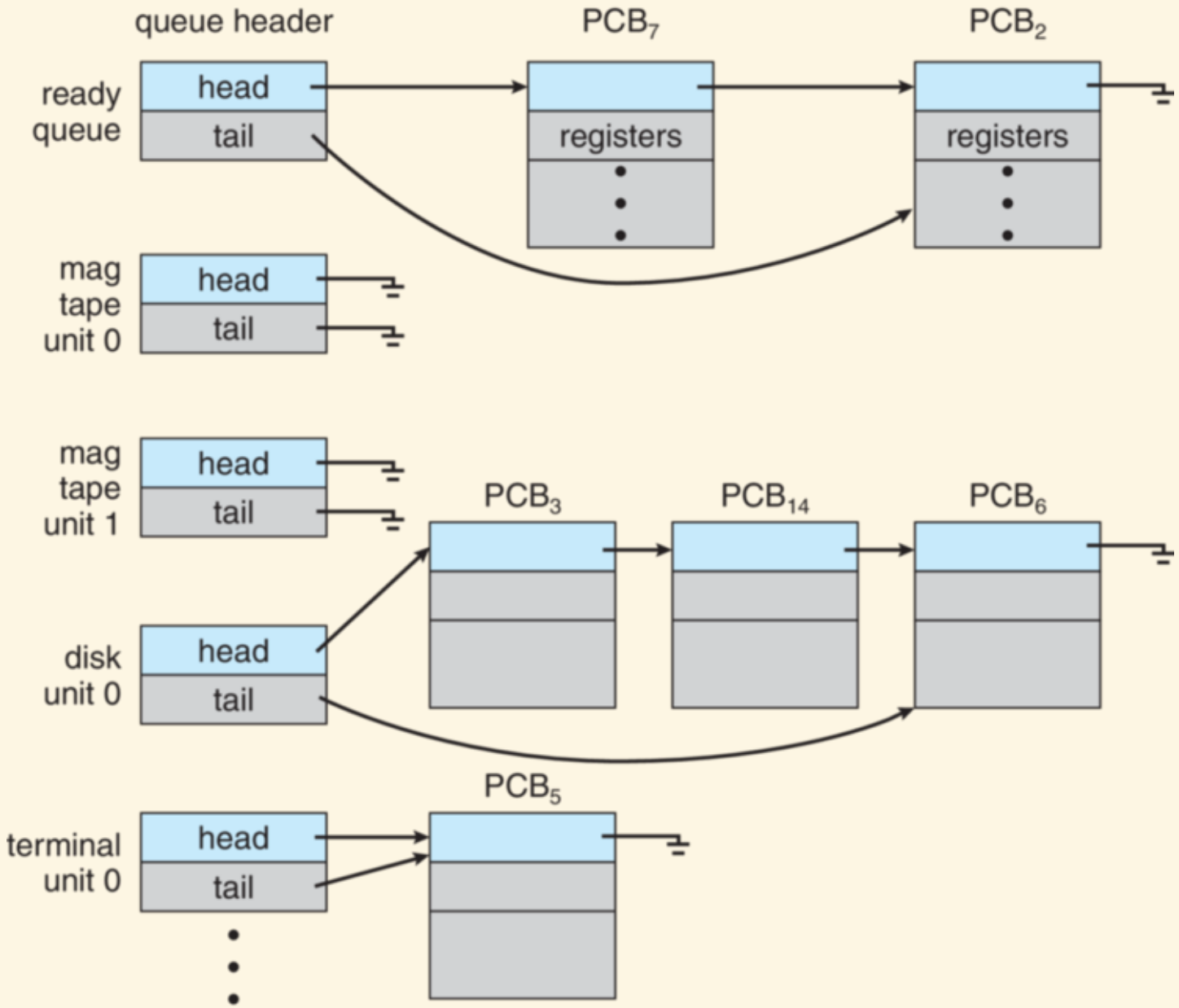
Nathan Bowman

Images taken from Silberschatz book

CPU Scheduling

We previously discussed the big picture of how processes are scheduled


ready queue
CPU
I/O
I/O queue
I/O request
time slice expired
child executes
fork a child
interrupt occurs
wait for an interrupt


queue header
PCB7
PCB2
ready queue
head
tail
registers
registers
mag tape unit 0
head
tail
mag tape unit 1
head
tail
PCB3
PCB14
PCB6
disk unit 0
head
tail
PCB5
terminal unit 0
head
tail

We will now focus specifically on the CPU (short-term) scheduler

This decides which process from the ready queue will be run next

The ready queue is not necessarily a FIFO queue

Although we are discussing processes, the same ideas apply to threads

We will discuss thread-specific issues later

We also focus on single-processor systems for now, with details of multi-processor systems coming later

CPU scheduling is fundamental to achieving good resource utilization and system responsiveness

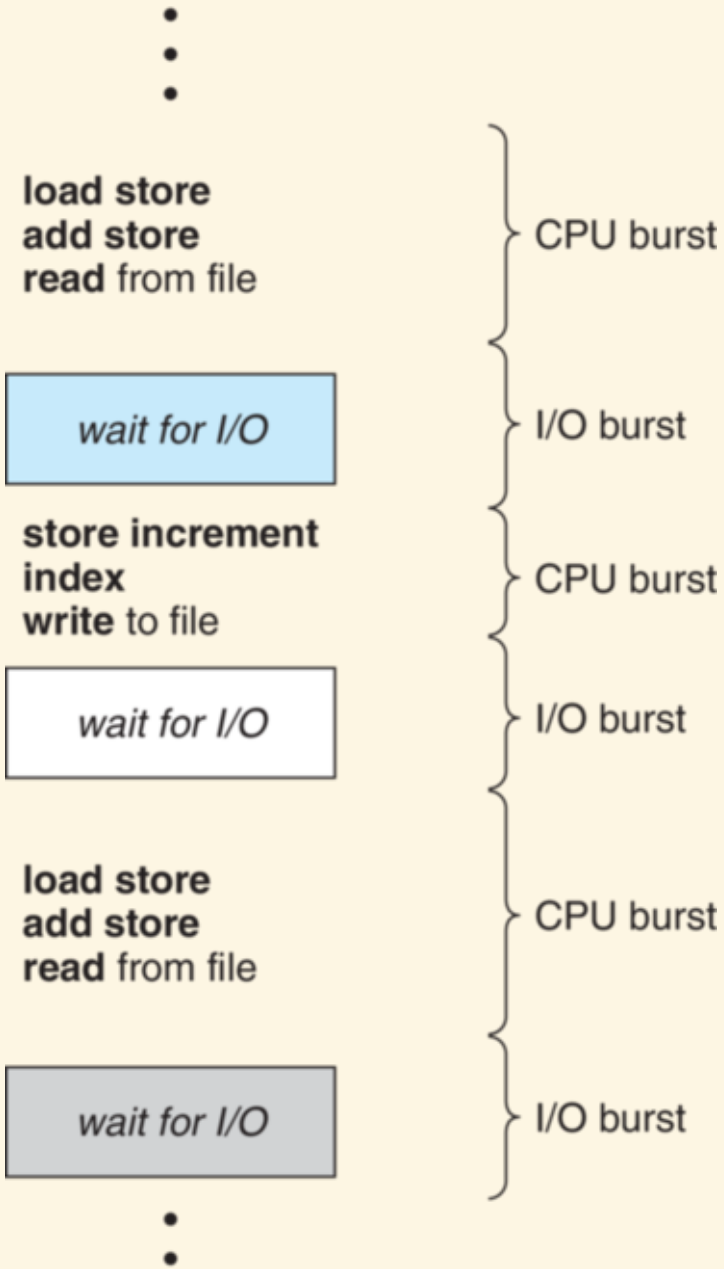
Want to ensure that there is some process using the CPU at all times

Scheduling objectives will depend on type of system, but some general rules apply

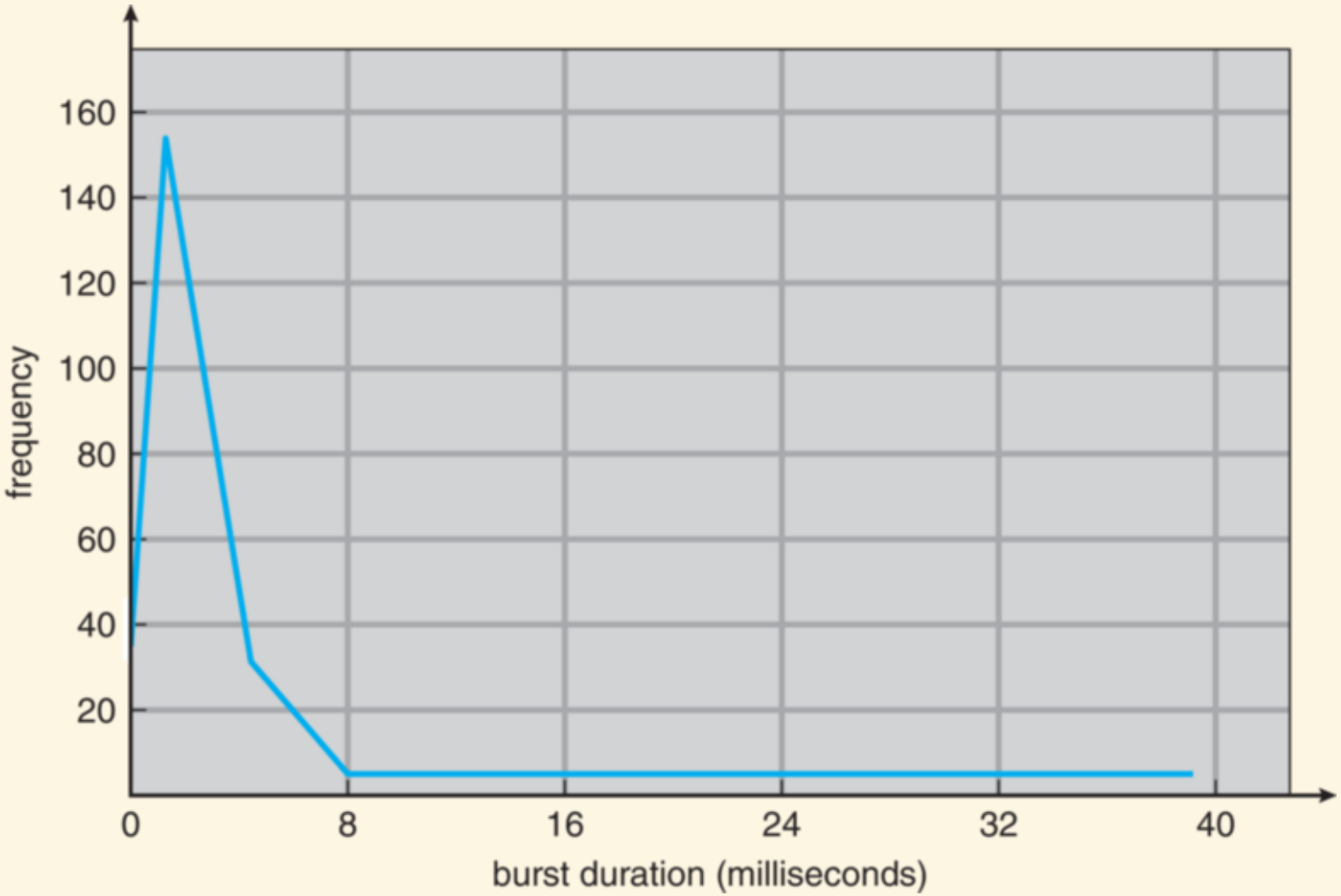
Key to effective scheduling: A given process will tend to alternate between **CPU bursts** and **I/O bursts**

CPU burst is when the program is actually performing useful work

Program will eventually need more data to operate on, so will make an I/O call and block


load store
add store
read from file
CPU burst
wait for I/O
I/O burst
store increment
index
write to file
CPU burst
wait for I/O
I/O burst
load store
add store
read from file
CPU burst
wait for I/O
I/O burst

CPU burst durations tend to be very short


160
140
120
100
80
60
40
20
frequency
0
8
16
24
32
40
burst duration (milliseconds)

New process may be assigned to CPU in any of four cases:

- running process switches from "running" to "waiting" (due to, e.g., an I/O call)
- running process switches from "running" to "ready" (due to, e.g., an interrupt)
- any process switches from "waiting" to "ready" (due to, e.g., completion of I/O)
- running process terminates

When running process blocks or terminates, new process should be chosen no matter what

In other two cases, different OSes may or may not even *consider* a context switch

Nonpreemptive scheduling (or cooperative scheduling) -- OS will run scheduler only when running processes blocks or terminates

Preemptive scheduling -- OS will run scheduler in any of four previously mentioned cases

Modern general-purpose OSes tend to use preemptive scheduling

Preemptive scheduling requires hardware support (a timer)

Preemptive scheduling is partly to blame for the race conditions we observed previously

Some OSes will disable preemption during a system call to ensure the OS itself does not get into an inconsistent state

The **dispatcher** is responsible for actually giving control to scheduled process

Dispatcher must

- switch context
- switch to user mode
- jump to user program

Summary

- CPU scheduler determines order of processes in ready queue
- Effective scheduler must account for CPU-I/O burst cycle
- Scheduling may be preemptive or nonpreemptive