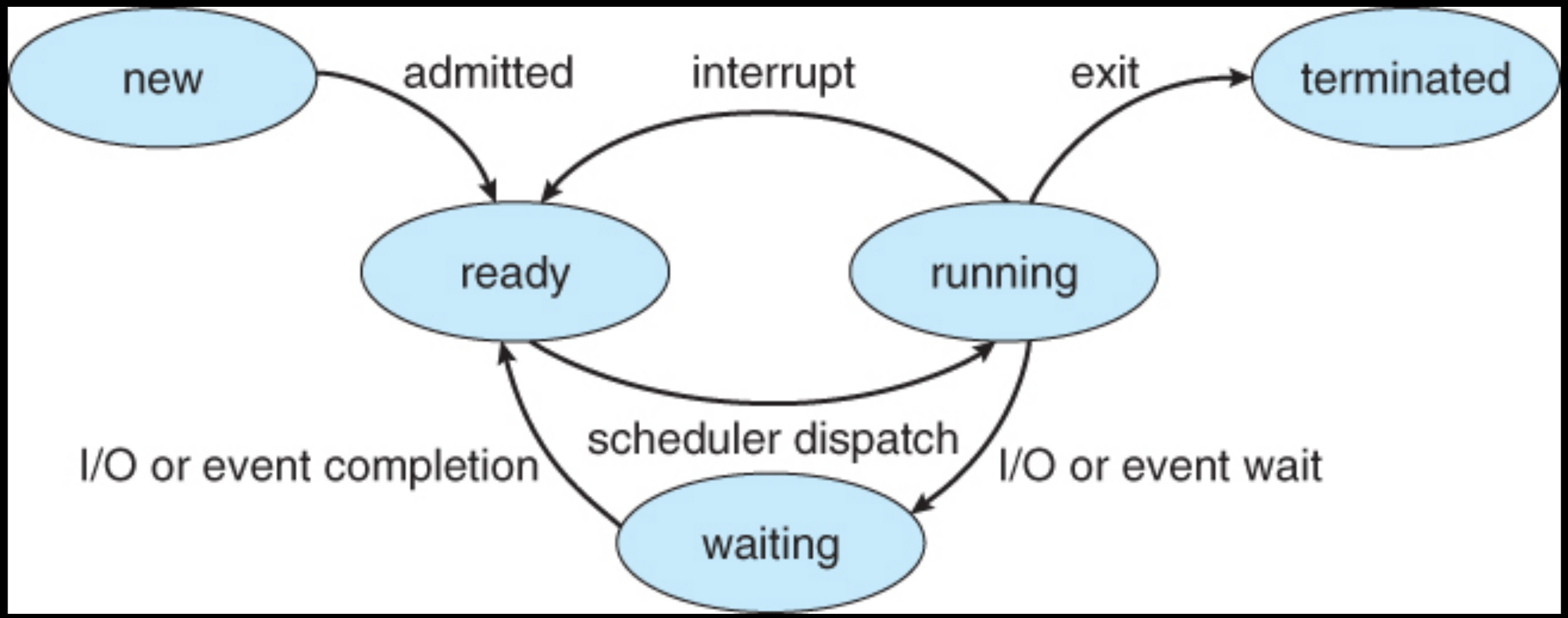
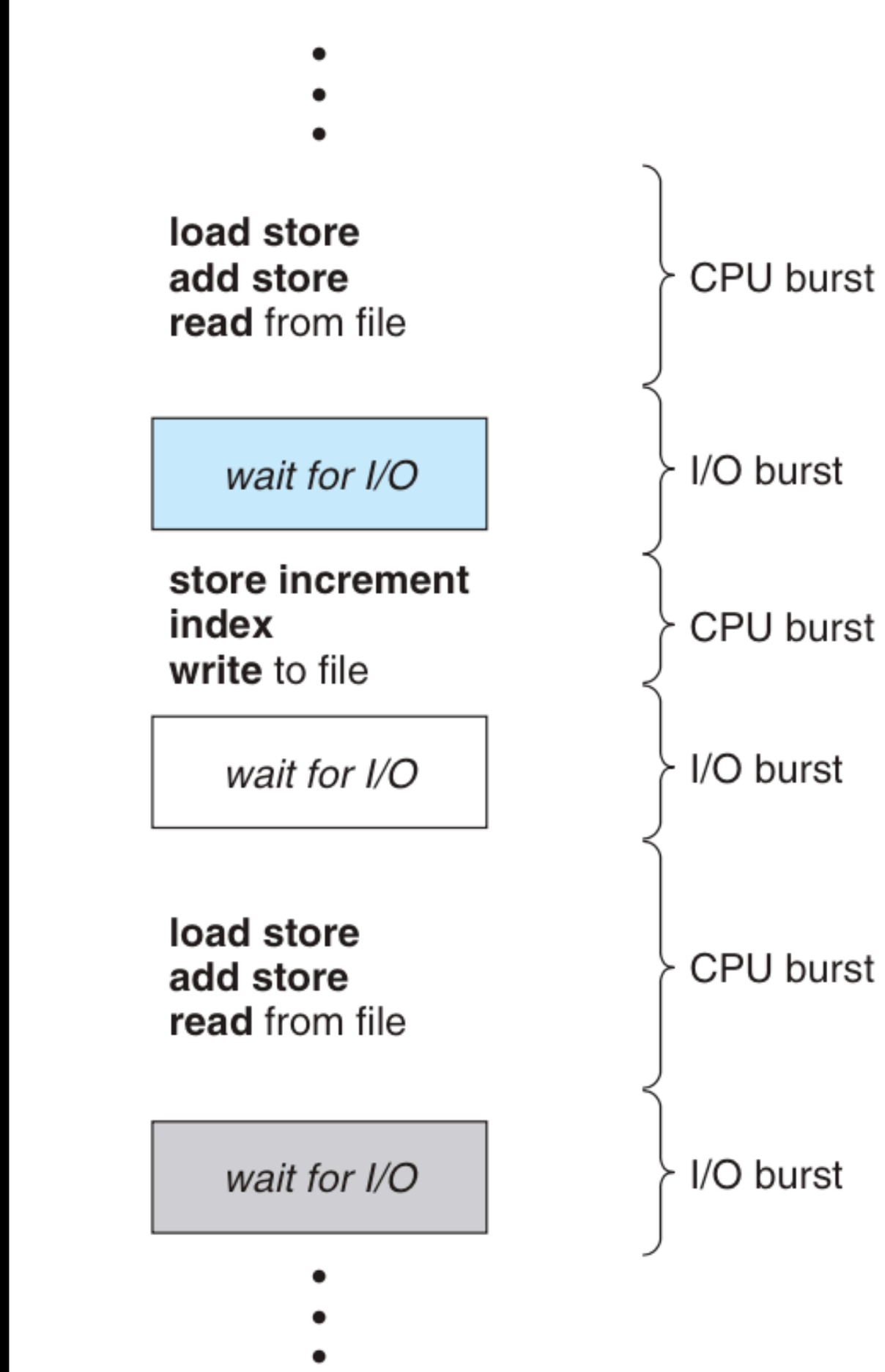
CPU Scheduling

Process states



Process Execution

- Computation & I/O
- Computation uses CPU(Running)
- Interacts I/O (Waiting)
- I/O means network, file...



Scheduling Problem

- Have k jobs ready to run
- Have N CPUs

Which jobs to assign to which CPU, and how long?

Criterion

1. Throughput
2. Turnaround time
3. Response time

Throughput

"Number of jobs complete per unit of time"

Higher is better

Turnaround time

"Time for each job to complete after submit"

Lower is better

Response time

"Time from request to the first response"

Lower is better

Waiting time

"Total time that job waits in ready queue"

Users do not see it

Scheduling Goals

1. Maximine "throughput"
2. Minimize "turnaround time"
3. Minimize "response time"

Optimize average or bounded values?
For example, to guarantee that all users get good service, we may want to minimize the maximum response time.

Can reach goal if

1. Maximine CPU utilization
2. Maximine I/O utilization

Notes

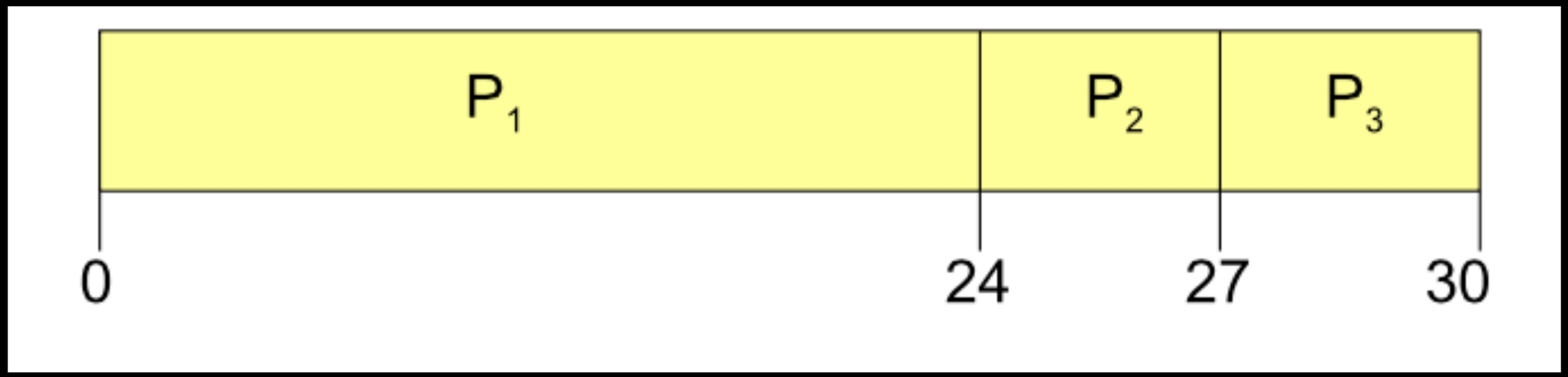
1. Overlap I/O & computation => increase utilization
2. I/O will be idle until triggered by "short CPU"

Scheduler

- First come first served
- Shortest Job First
- Round Robin

FCFS Scheduling

P1 needs 24 seconds, P2 & P3 need 3 seconds



- Throughput: 3/30 = 0.1 jobs/second
- Turnaround: (24 + 27 + 30)/ 3 = 27
- If order is "P2 P3 P1", turnaround = (3 + 6 + 30)/3

Convoy Example

Line of customers: C1, C2, C3, C4
- C1 needs 2 hours
- C2, C3 & C4 needs 30 seconds to have signatures then sends to other departments

Convoy Effect

- Long computation process hold the CPU
- Queued I/O processes needs "short CPU" to trigger I/O operations

Shortest Job First

Schedule the job whose next CPU burst is the shortest

(Preemptive => Shortest Remaining Time First)

Advantages of SJF

Minimize waiting time for set of given jobs

Limitation of SJF

- Doesn’t always minimize average turnaround time
- Can lead to unfairness or starvation

Round Robin

- Similar to FCFS but time is sliced
- Time slice, quantum: 10 - 100 milliseconds
- Ready queue is circular queue (FIFO)

Advantages of RR

- Fairness
- Low average waiting time
- Good responsiveness

Time slice value

- Too large becomes FCFS
- Too small "Context Switching" cost

Other schedulers

- Realtime scheduler
- Multiprocessors scheduler

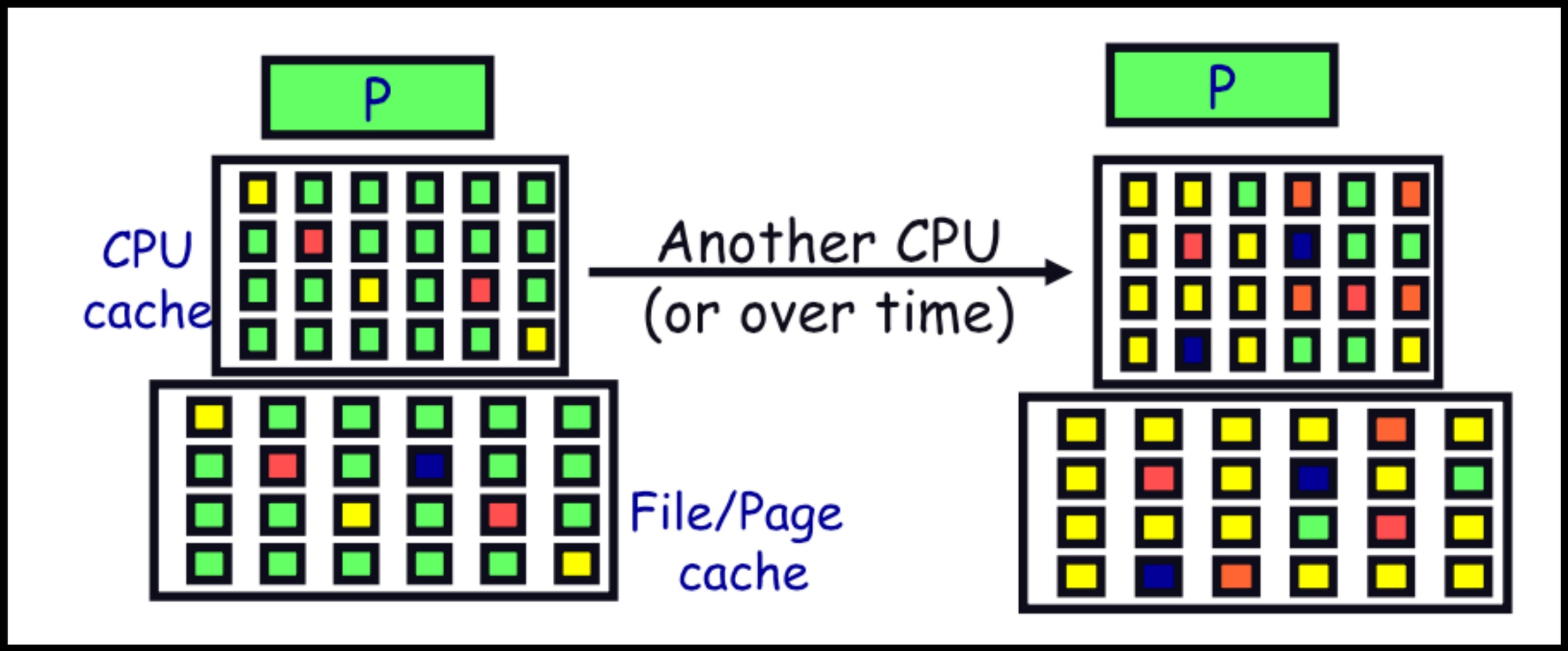
Context Switching

Happens when save current execution, restore previous execution

- Save & restore registers
- Switch address spaces (Expensive)

Hidden cost

- Cache outdated
- Page fault



Context-Switch is expensive

Multithreading Questions

1. Does multiple cores make application run faster?

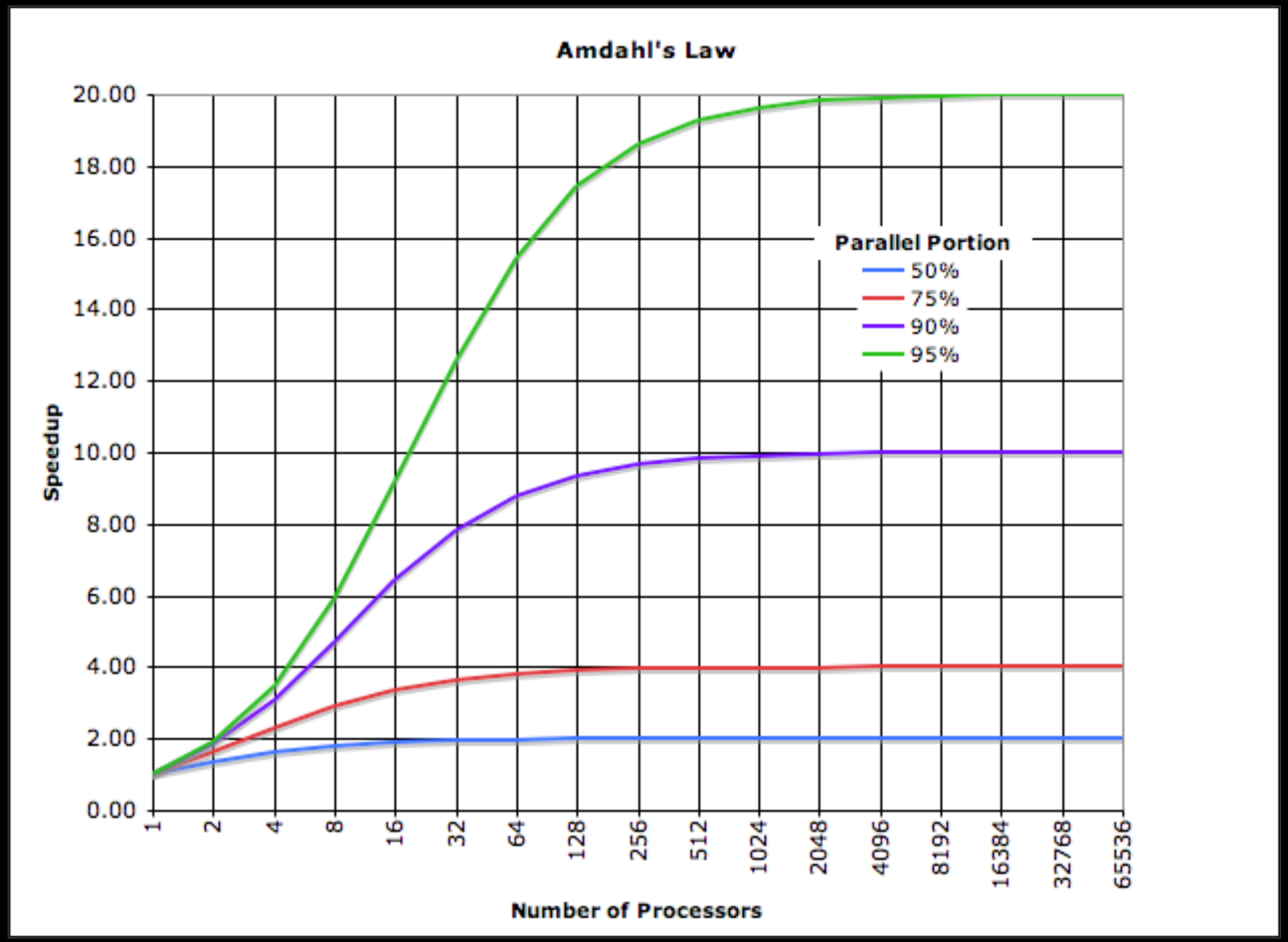
2. Why do we write multithreading applications?

Amdahl's Law

$$Speedup \leq \frac{1}{F + \frac{(1-F)}{N}}$$

- **N** is number of cores
- **F** is fraction of serial code

Amdahl's Law


Amdahl's Law
Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
Parallel Portion
50%
75%
90%
95%
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors

Why do we write multithreading applications

1. Faster. (No, not really)
2. More responsive. (Yes. When do you need it)

Multithread can slow down

Example:

1 CPU, 2 threads slower than 1 CPU, 1 thread

Reasons:

1. Context switching cost
2. Lock

More responsive

- Response time is reduced with multithreading
- GUI application requires small responsive time