Chapter 3: Processes

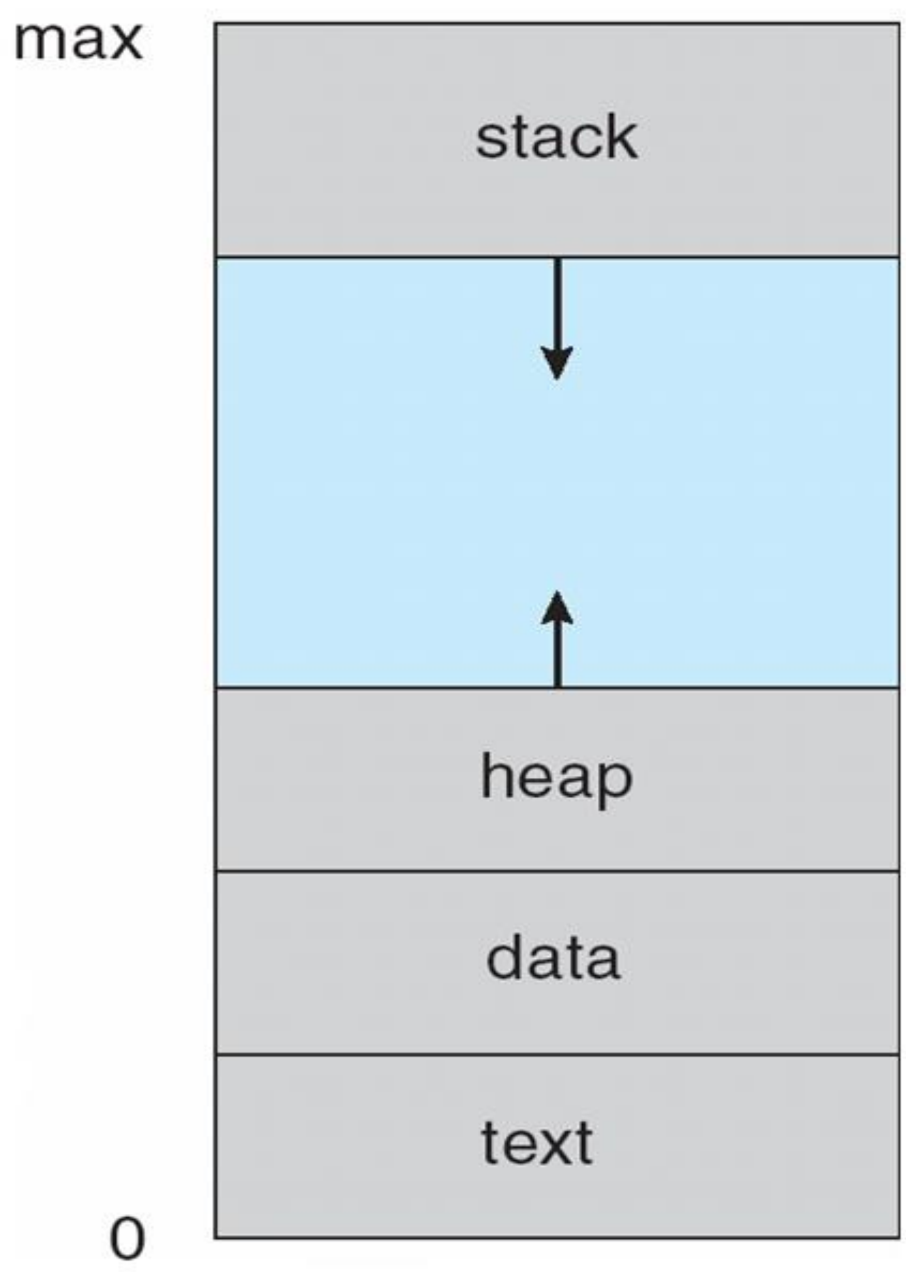
Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Terms *job* and *process* almost interchangeably
- Process – a program in execution; process execution must progress in sequential fashion
  - It is a unit of work in a modern time-sharing system.
- A process includes:
  - program counter
  - Stack
  - Data section
  - Heap

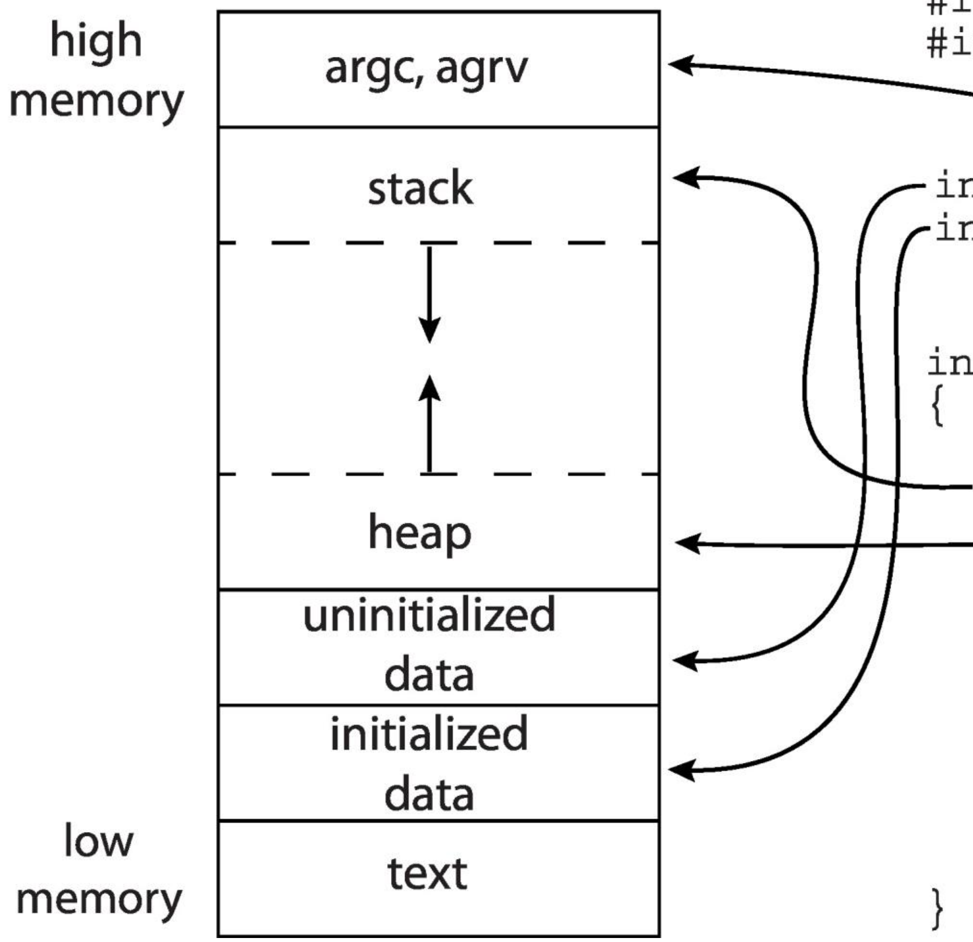
The Process

- Multiple parts
  - The program code, also called **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time
- Program is passive entity, process is active
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program

Process in Memory



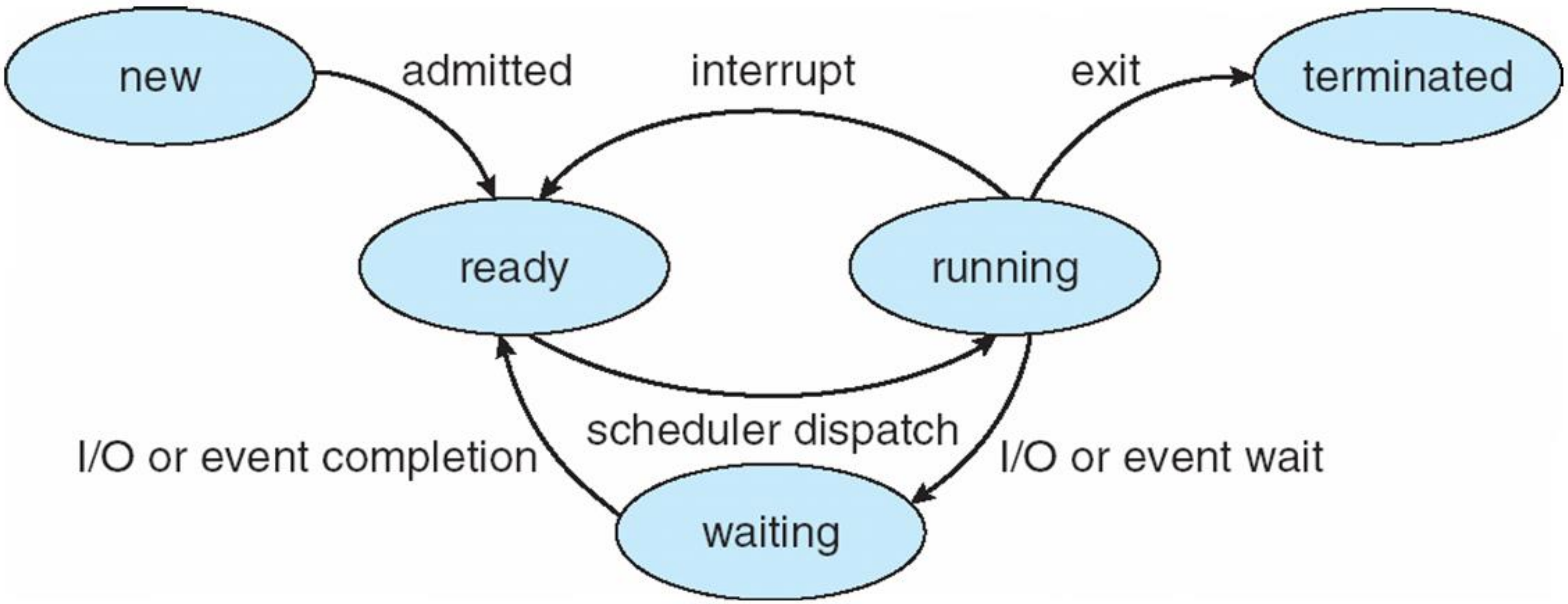
Memory Layout of a C Program




```
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 15;

int main(int argc, char *argv[])
{
    int *values;
    int i;

    values = (int *)malloc(sizeof(int)*5);

    for(i = 0; i < 5; i++)
        values[i] = i;

    return 0;
}
```

Process State

- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution



Only one process can be running on any processor at any instant

Process Control Block (PCB)

Also called as a Task Control Block

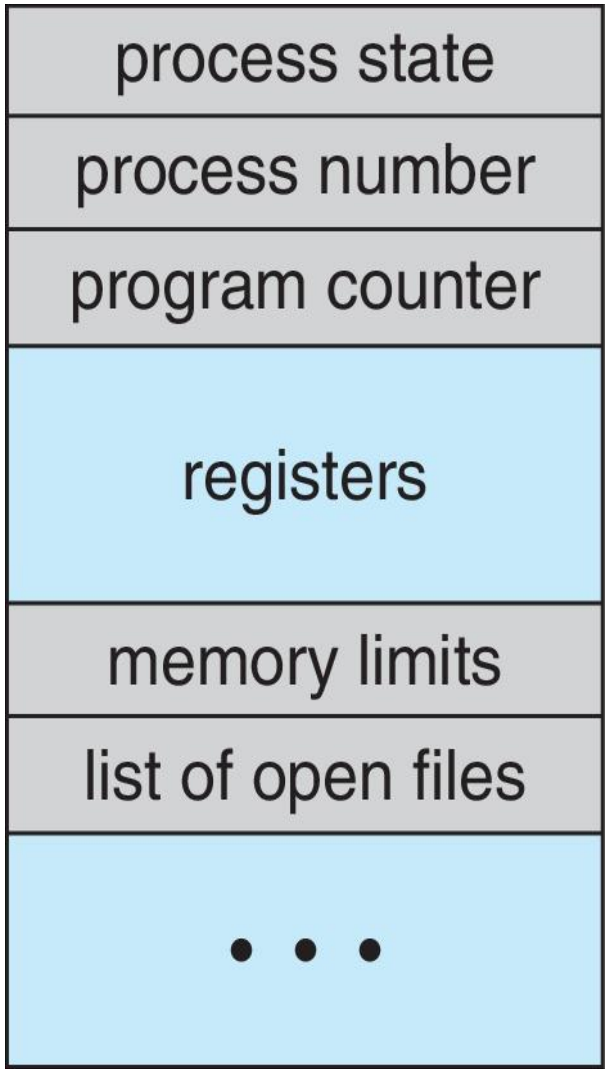
Information associated with each process

- Process state- **ready, running, etc.**
- Program counter- **keeps track of the next instruction to be executed**
- CPU registers- **accumulators, index registers, general purpose registers**
- CPU scheduling information- **process priority and scheduling queues**
- Memory-management information- **values of base-index registers, page table, segment tables**
- Accounting information- **amount of CPU time used, time limits, etc.**
- I/O status information- **list of I/O devices allocated to the process etc**.

Process Control Block (PCB)

Information associated with each process(also called **task control block**)

- Process state – running, waiting, etc.
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



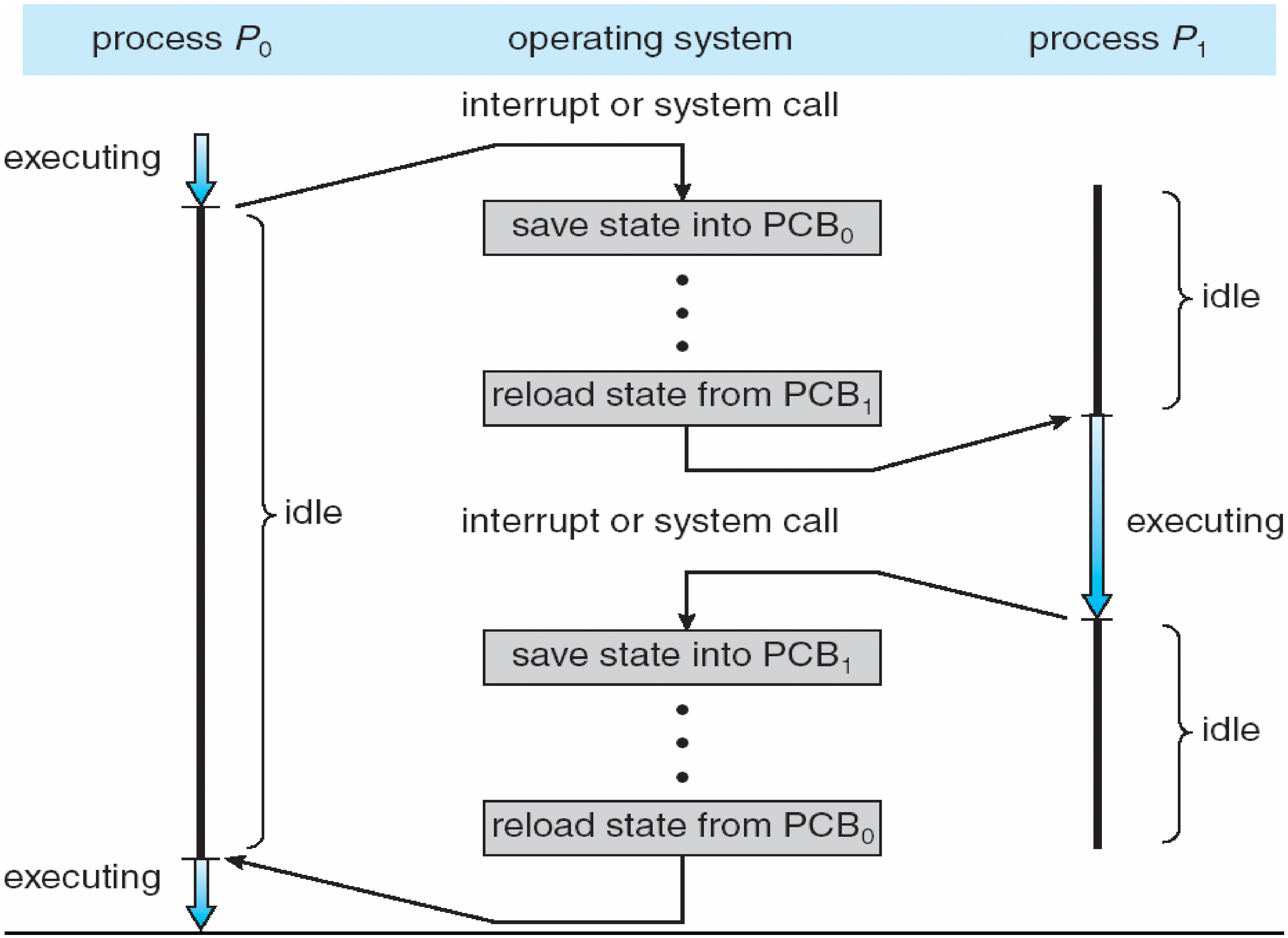
Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a **context switch**.

- **Context** of a process represented in the PCB

- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB -> longer the context switch

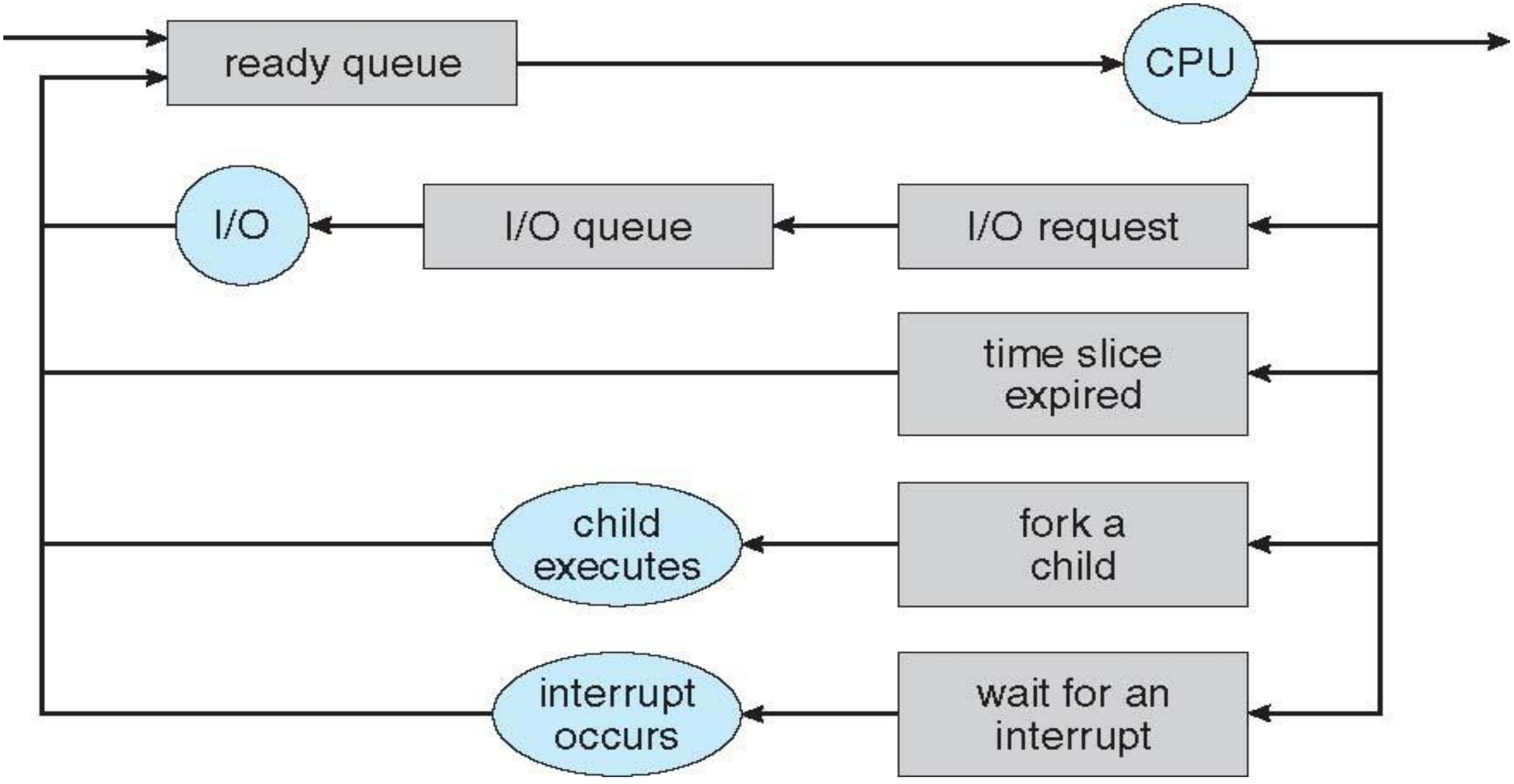
CPU Switch From Process to Process



Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
    - Each device has its own device queue
  - Processes migrate among the various queues

Representation of Process Scheduling



Queueing Diagram

Schedulers

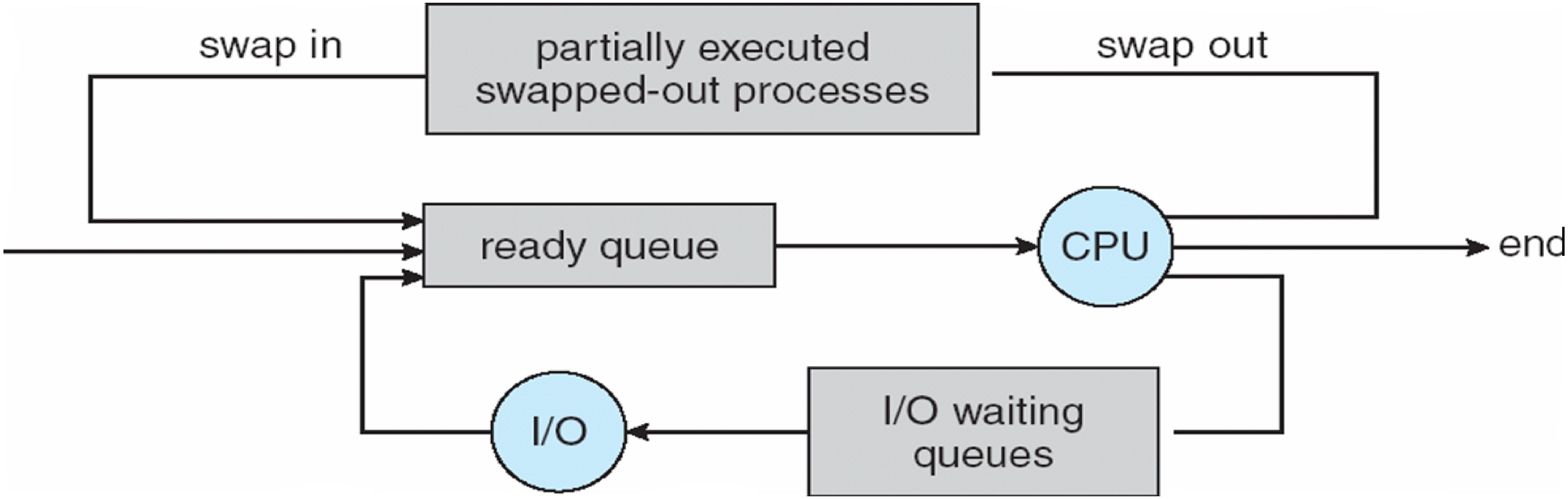
- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue

- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system

Schedulers (Cont.)

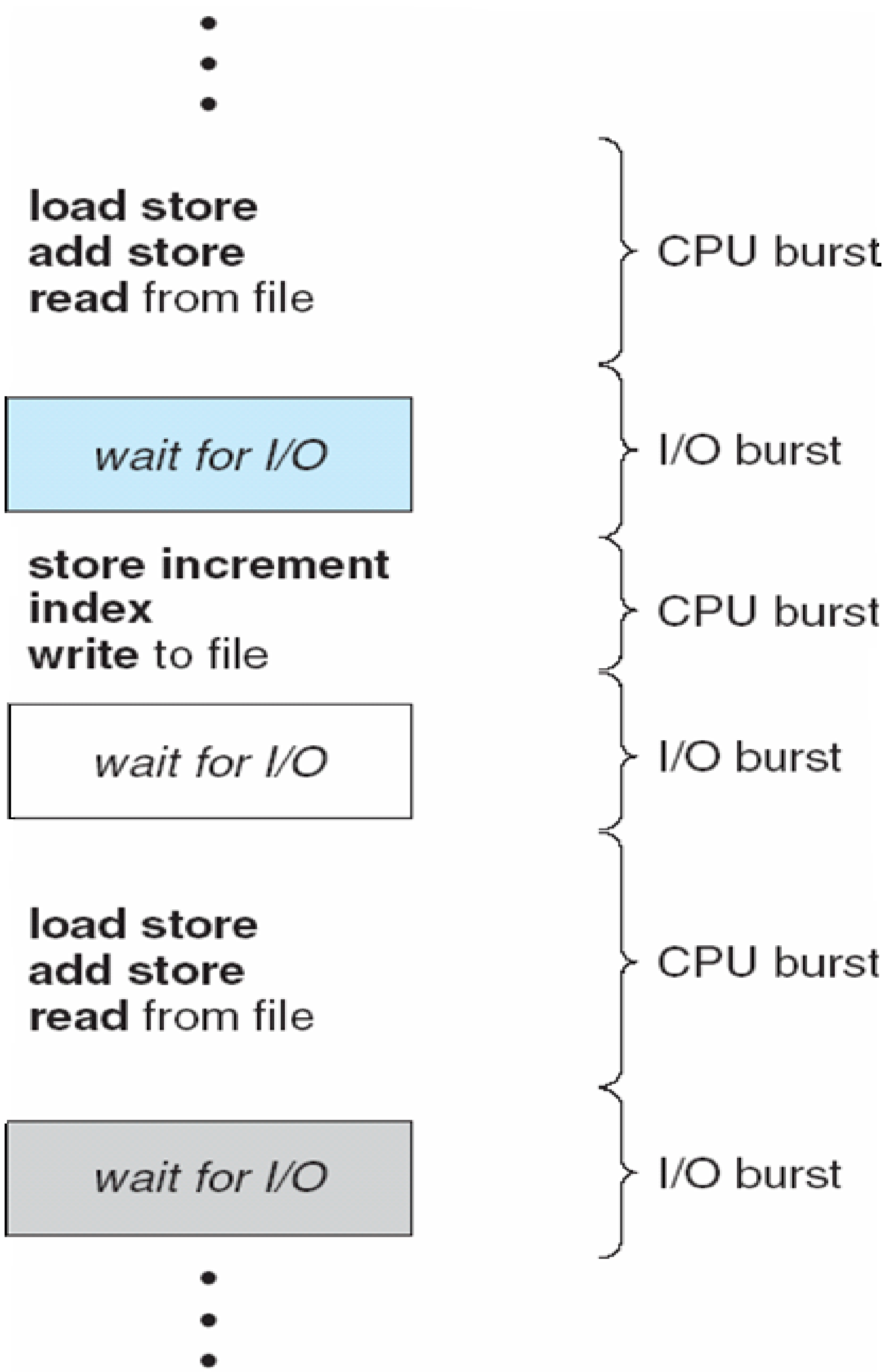
- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
- The long-term scheduler controls the *degree of multiprogramming*
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; ~~few very long,~~ might have a few long CPU bursts

Addition of Medium Term Scheduling



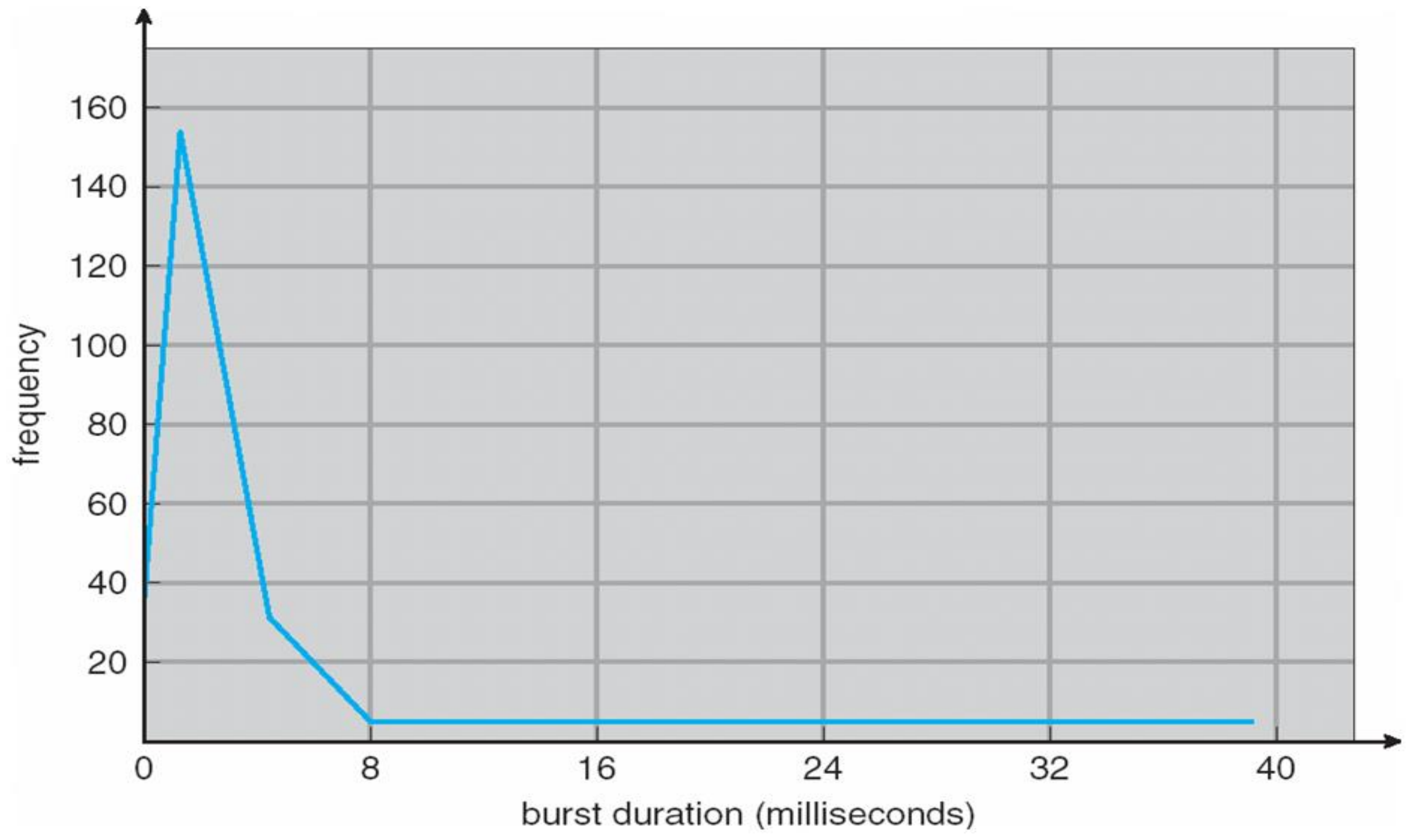
- Key Idea
  - Sometimes it is advantageous, to remove process from memory (active contention of CPU) to reduce the degree of multiprogramming.
  - The process can be reintroduced into the ready queue and its execution can be continued where it left off. The scheme is called swapping.
  - Swapping may be necessary to improve the process mix.

Alternating Sequence of CPU and I/O Bursts



- Multiprogramming
  - Several processes are kept in memory at one time.
  - When one process has to wait for I/O.
  - The OS takes CPU away from that process and gives the CPU to another process.

Histogram of CPU-burst Times



- Multiprogramming
  - Large number of short CPU bursts
  - Small number of Long CPU bursts
  - An I/O bound process has many short CPU bursts

CPU Scheduler

- Selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways (Not in FIFO only)

- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates

- Scheduling under 1 and 4 is nonpreemptive
- All other scheduling is preemptive

- No-preemptive- Once the CPU has been allocated to a process, the process keeps the CPU until termination or switching to waiting state.

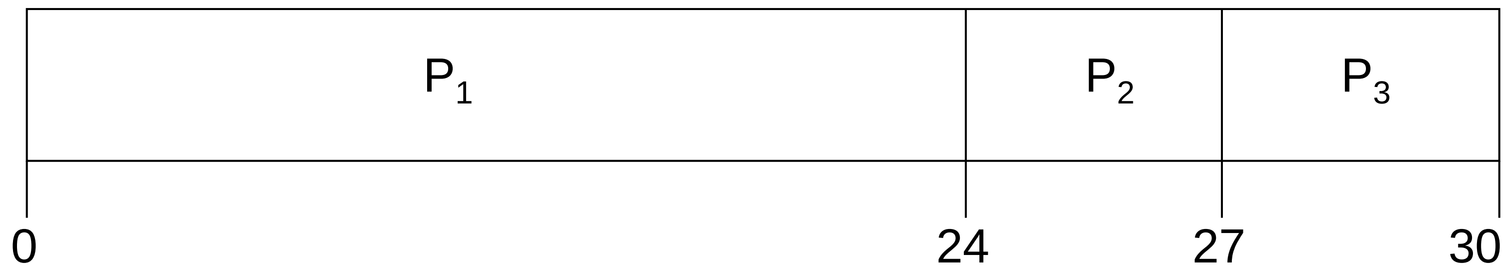
Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – #number of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process (Time from submission to completion)
- Waiting time – amount of time a process has been waiting in the ready queue
- Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



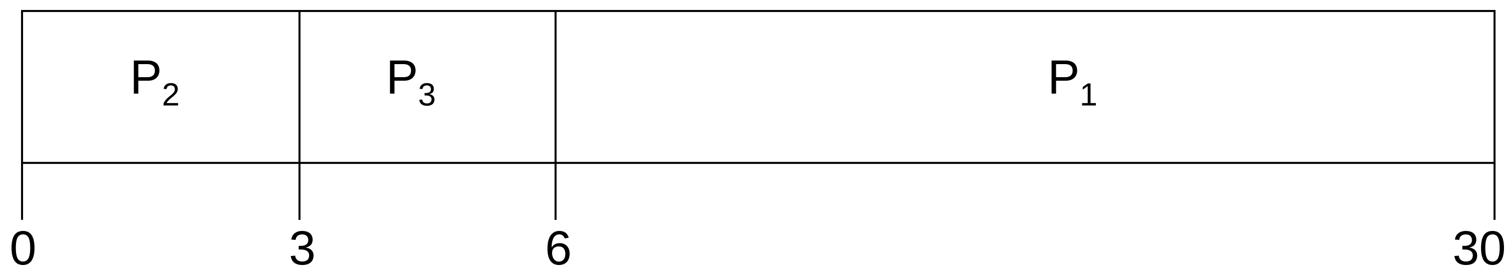
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- Convoy effect - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

FCFS Scheduling (Cont.)

- FCFS scheduling algorithm is non-preemptive
  - Once the CPU has been allocated to a process, it keeps the CPU until process terminates or by requesting I/O.
  - FCFS is therefore troublesome for time sharing systems.
    - It would be disastrous to allow one process to keep the CPU for an extended period.

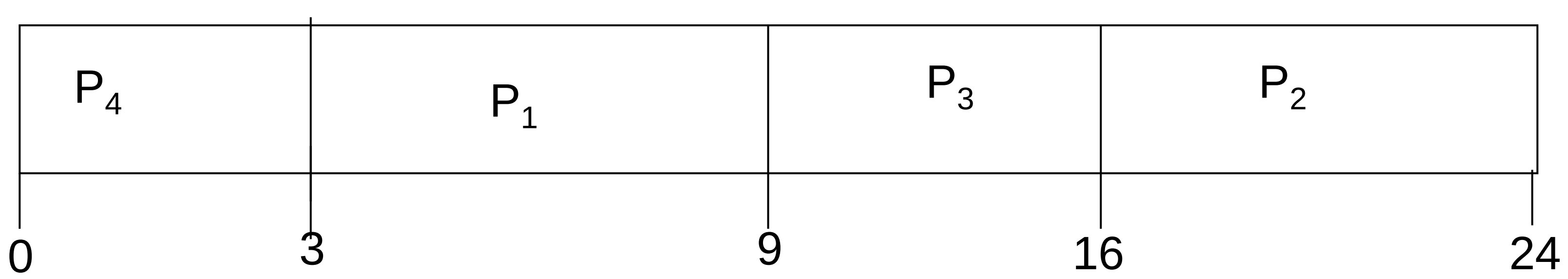
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
  - if CPU bursts of two processes are same, then FCFS scheduling is used for selection
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Knowing the length of the next CPU request?

Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst

- Can be done by using the length of previous CPU bursts, using exponential averaging

$$1.\ t_n = \text{actual length of } n^{th} \text{ CPU burst}$$
$$2.\ \tau_{n+1} = \text{predicted value for the next CPU burst}$$
$$3.\ \alpha,\ 0 \le \alpha \le 1$$
$$4.\ \text{Define:}$$

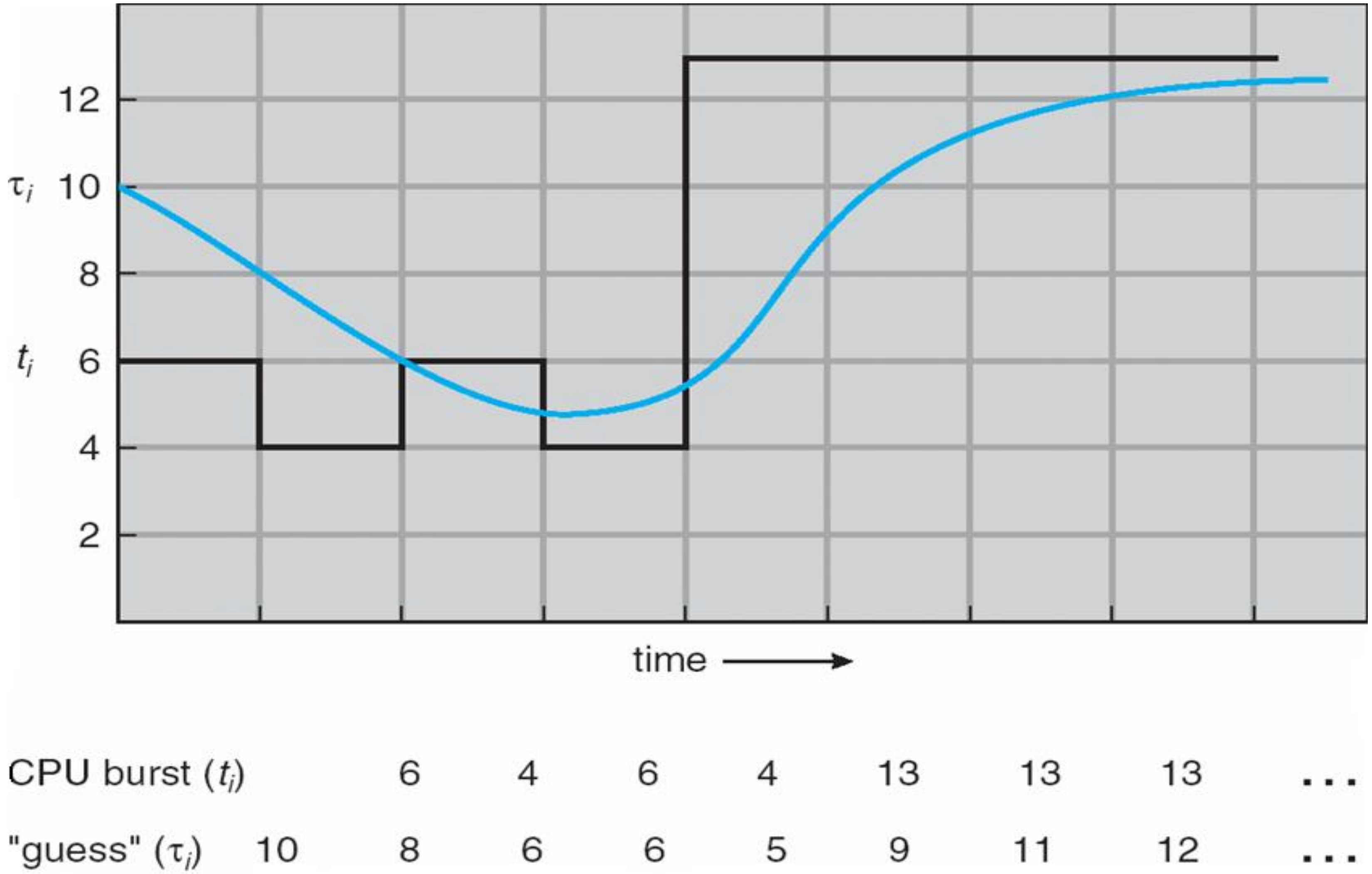
$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n.$$

- Commonly, α set to ½
- Preemptive version called shortest-remaining-time-first

Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts

Prediction of the Length of the Next CPU Burst



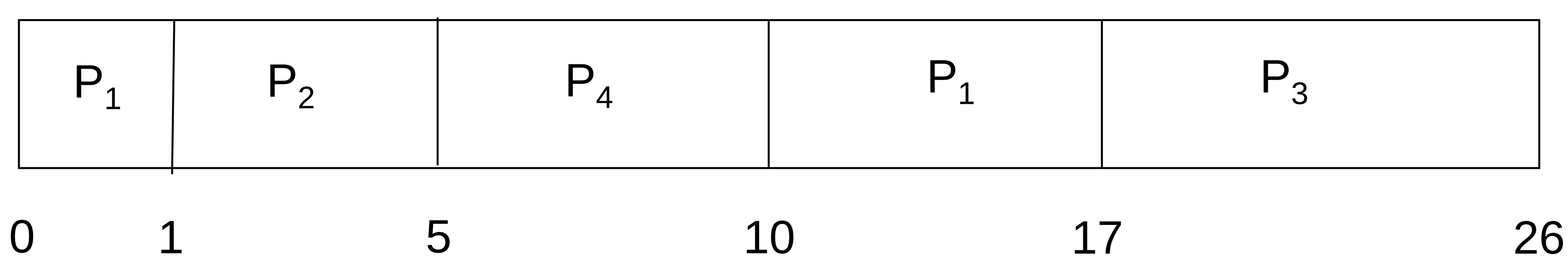
- Preemptive version called shortest-remaining-time-first

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

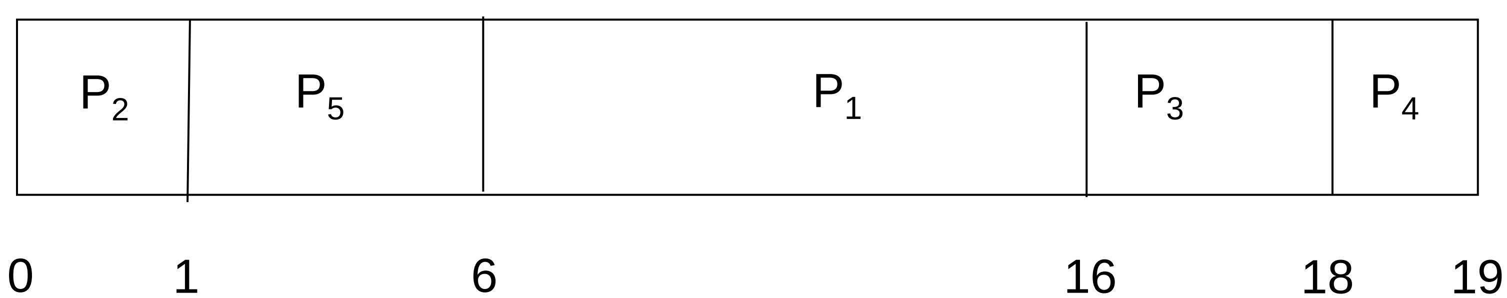
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ Starvation – low priority processes may never execute
- Solution ≡ Aging – as time progresses increase the priority of the process

Example of Priority Scheduling

<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

Round Robin (RR)

- Each process gets a small unit of CPU time (time quantum q), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.

- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.

- Timer interrupts every quantum to schedule next process

- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high
  - if time quantum is extremely small,
    - processor sharing

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

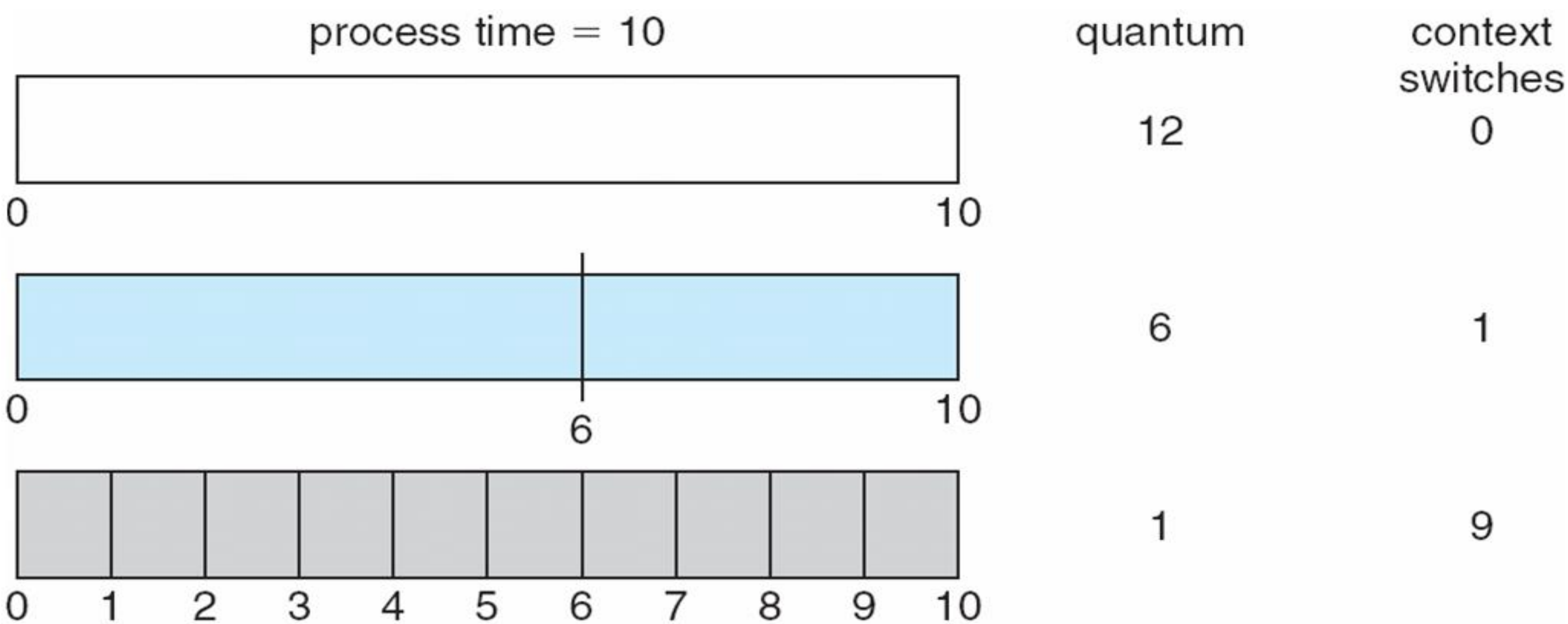
P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

- Typically, higher average turnaround than SJF, but better *response*
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Time Quantum and Context Switch Time

Typically, higher average turnaround than SJF,
but better *response*

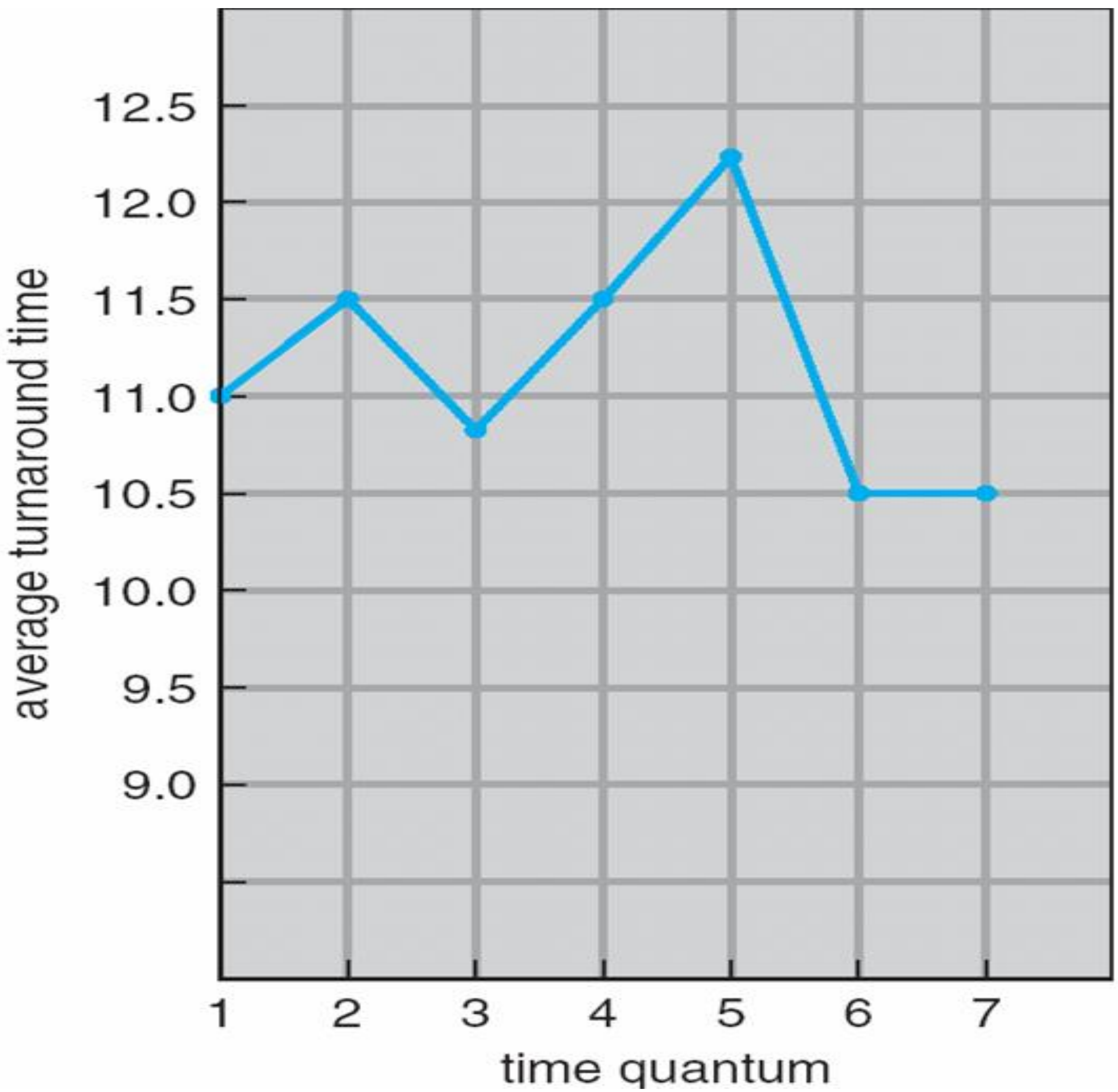


Turnaround Time Varies With The Time Quantum

process	time
P_1	6
P_2	3
P_3	1
P_4	7

Turnaround time with quantum =5

Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than q

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

- Typically, higher average turnaround than SJF, but better *response*
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

Process	Burst Time	Priority
P_1	2	2
P_2	1	1
P_3	8	4
P_4	4	2
P_5	5	3

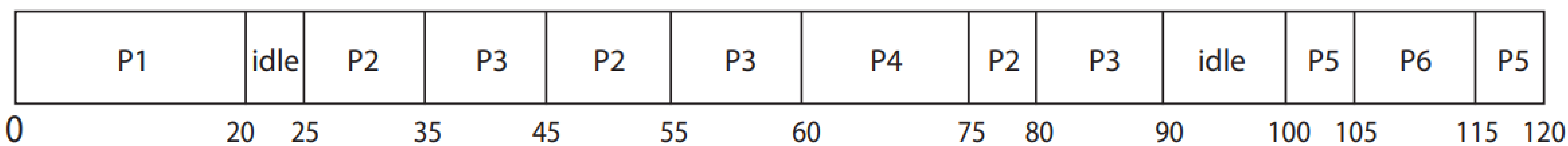
Process	Priority	Burst	Arrival
P_1	40	20	0
P_2	30	25	25
P_3	30	25	30
P_4	35	15	60
P_5	5	10	100
P_6	10	10	105

The length of a time quantum is 10 units

If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

P1	idle	P2	P3	P2	P3	P4	P2	P3	idle	P5	P6	P5

0 20 25 35 45 55 60 75 80 90 100 105 115 120



P1: 20-0 - 20, P2: 80-25 = 55, P3: 90 - 30 = 60, P4: 75-60 = 15, P5: 120-100 = 20, P6: 115-105 = 10

P1: 0, p2: 40, P3: 35, P4: 0, P5: 10, P6: 0

Algorithm Evaluation

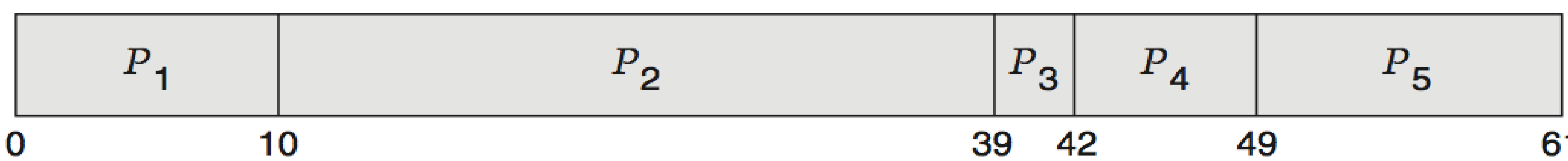
- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- Deterministic modeling
  - Type of analytic evaluation
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0:

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

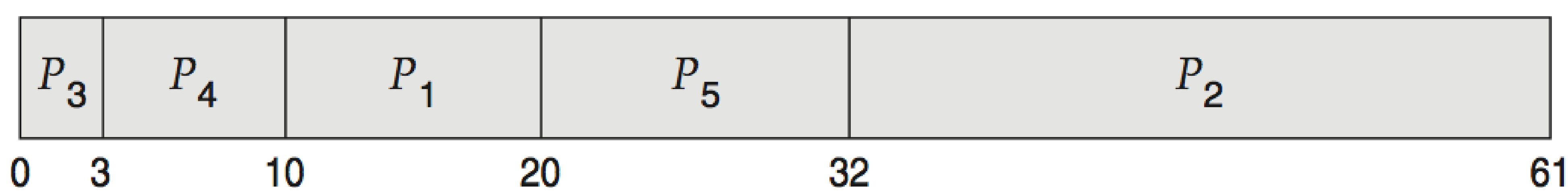
Deterministic Evaluation

- For each algorithm, calculate minimum average waiting time
- Simple and fast, but requires exact numbers for input, applies only to those inputs

  - FCS is 28ms:



  - Non-preemptive SFJ is 13ms:



  - RR is 23ms:

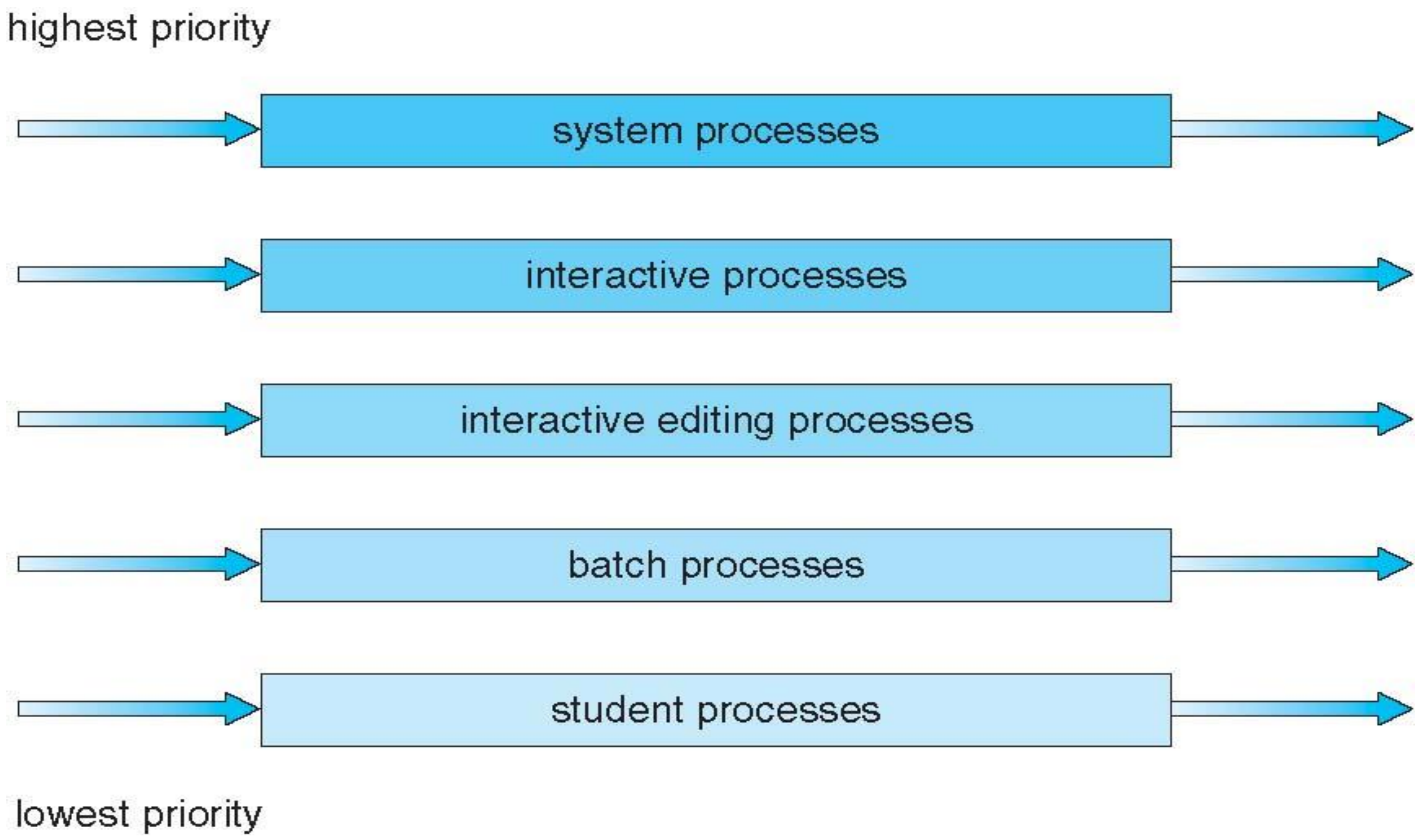
Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue based on the property of the process

- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS

- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR 20% to background in FCFS

Multilevel Queue Scheduling



Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
  - If process uses too much CPU time.
    - Moved to lower- priority queue
  - I/O bound and interactive process
    - Moved to higher priority queue

- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS

- Scheduling
  - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2

Multilevel Feedback Queues