

CPU SCHEDULING

- CPU Scheduling refers to a set of policies and mechanisms build into the operating system to determine which process will be executed next.

Long-Term Scheduling

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
- More processes, smaller percentage of time each process is executed

Medium-Term Scheduling

- selects which process should be removed or re-introduced into memory
- Part of the swapping function
- Based on the need to manage the degree of multiprogramming

Short-Term Scheduling

- Known as the dispatcher
- decision as to which available process will be executed
- Executes most frequently
- Invoked when an event occurs
 - Clock interrupts
 - I/O interrupts
 - Operating system calls

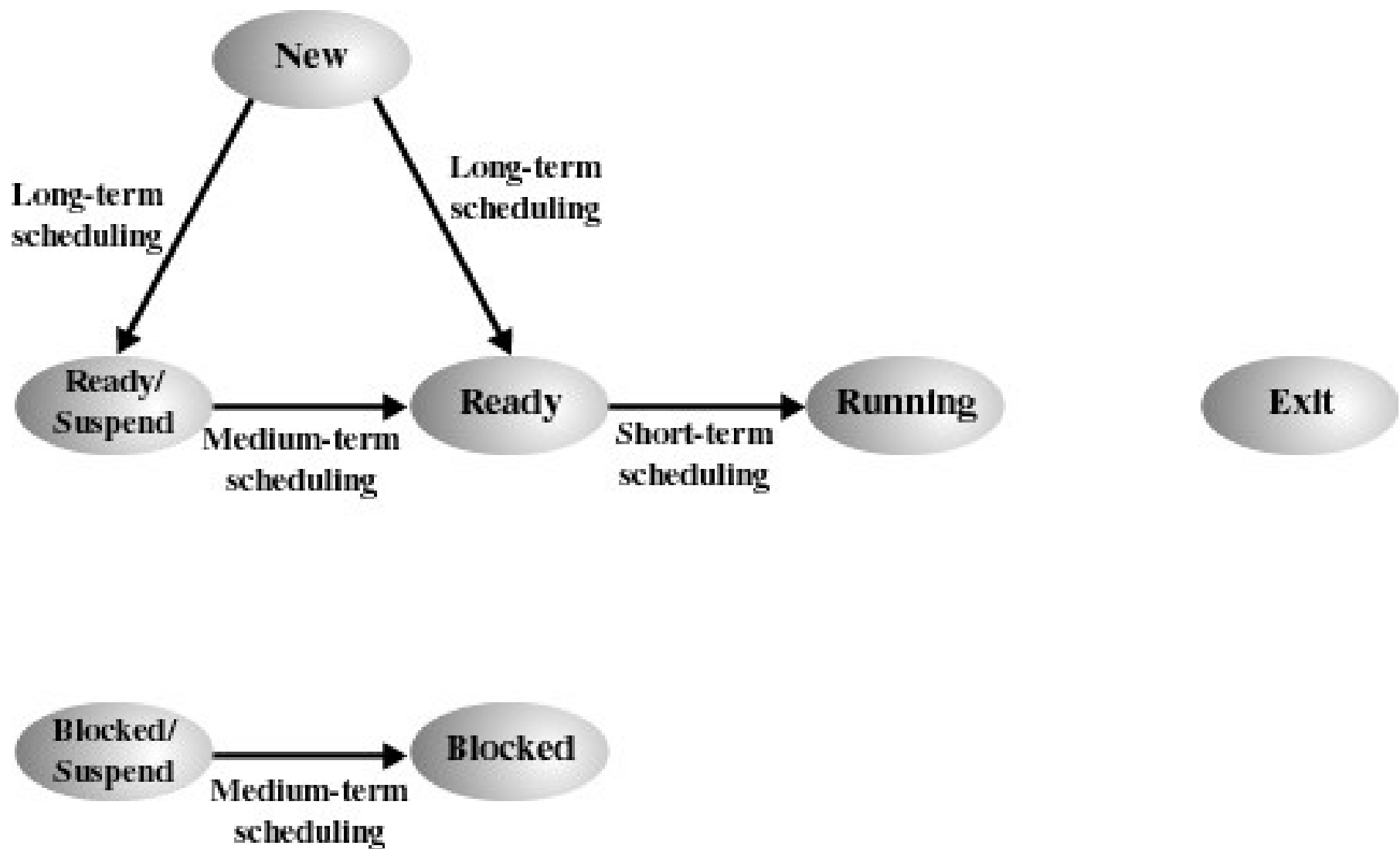
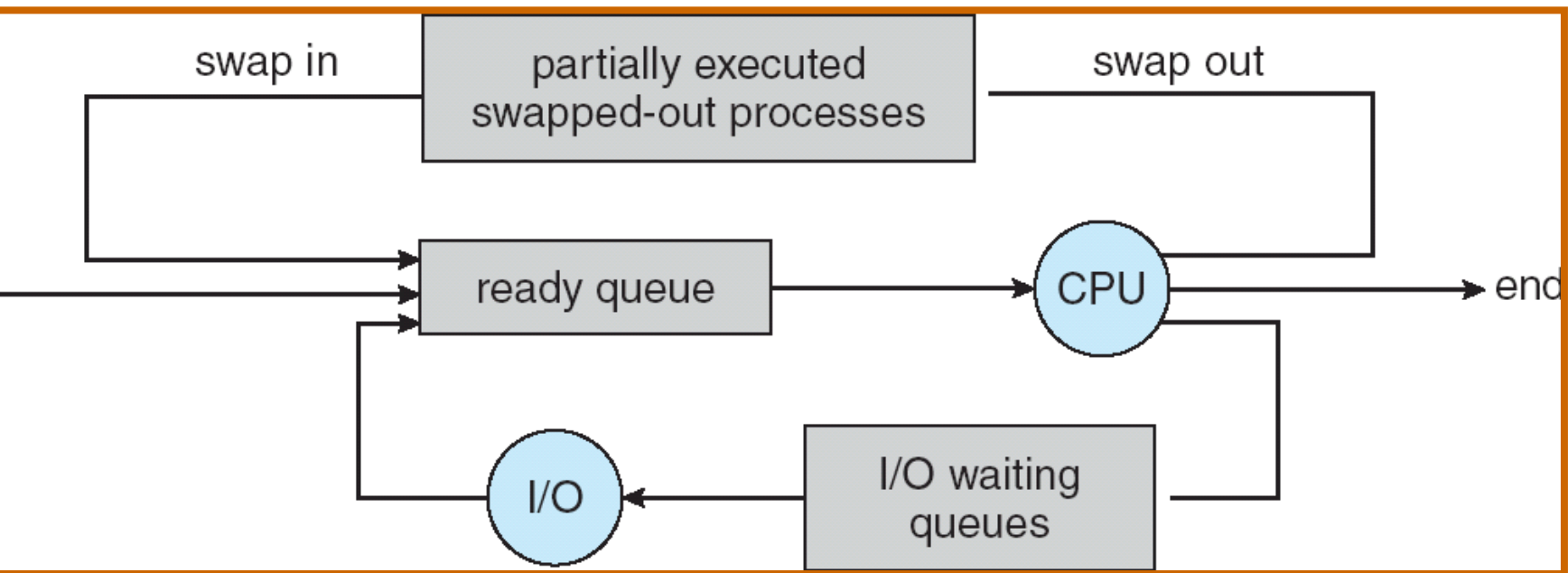


Figure 9.1 Scheduling and Process State Transitions



CPU scheduling

- Determination of which process will be executed next by the CPU
- The **scheduler** is responsible for choosing a process from the ready queue. The **scheduling algorithm** implemented by this module determines how process selection is done.
- The scheduler hands the selected process off to the **dispatcher** which gives the process control of the CPU.

CPU and I/O Bursts

- Typical process execution pattern: use the CPU for a while (CPU burst), then do some I/O operations (IO burst).
- CPU bound processes perform I/O operations infrequently and tend to have **long** CPU bursts.
- I/O bound processes spend less time doing computation and tend to have **short** CPU bursts.

CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- 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 *non-preemptive*
- All other scheduling is *preemptive*

Preemptive & Non-preemptive Scheduling

- SO what is Preemptive & Non-preemptive scheduling?
- Non-preemptive
 - Once CPU is allocated to a process, it holds it till it
 - Terminates (exits)
 - Blocks itself to wait for I/O
 - Requests some OS service
 - Used by Windows 3.x

Preemptive & Non-preemptive Scheduling

- Preemptive
 - Currently running process may be interrupted and moved to the ready state by the OS
 - Windows 95 introduced preemptive scheduling
 - Used in all subsequent versions of windows
 - Mac OS X uses it

Criteria & Objectives

- CPU utilization – keep the CPU as busy as possible (40 – lightly loaded, 90 – heavily loaded)
- Throughput – of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process (total time spent on the system)
- 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)

Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

SCHEDULING ALGORITHMS

- An algorithm refers to the sequence of steps followed in performing a given task
- Scheduling algorithms determine the order in which processes are granted the CPU.

Scheduling Algorithms

FIRST-COME, FIRST SERVED:

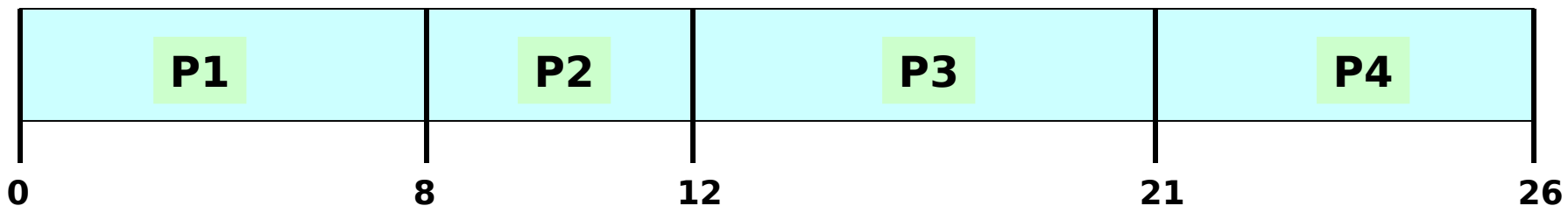
- (FCFS) same as FIFO
- Process granted the CPU using the arrival order
- Simple, fair, but poor performance. Average queueing time may be long.

FIFO Gantt Chart

EXAMPLE DATA:

Process	Arrival Time	Service Time
1	0	8
2	1	4
3	2	9
4	3	5

FCFS



$$\text{Average wait} = ((8-0) + (12-1) + (21-2) + (26-3)) / 4 = 61/4 = 15.25$$

**Residence Time
at the CPU**

Priority Scheduling

- Algorithm that gives preferential treatment to important
- jobs
- Each process is associated with a priority and the one with the
- highest priority is granted the CPU
- Equal priority processes are scheduled in FCFS order
- Priorities can be assigned to processes by a system
- administrator (e.g. staff processes have higher priority
- than student ones) or determined by the Processor
- Manager on characteristics such as:
 - Memory requirements
 - Peripheral devices required
 - Total CPU time
 - Amount of time already spent processing

Scheduling Algorithms

- **Shortest Job First (SJF)**
 - Special case of priority scheduling (priority = expected length of CPU burst)
 - Non-preemptive version: scheduler chooses the process with the least amount of work to do (shortest CPU burst).

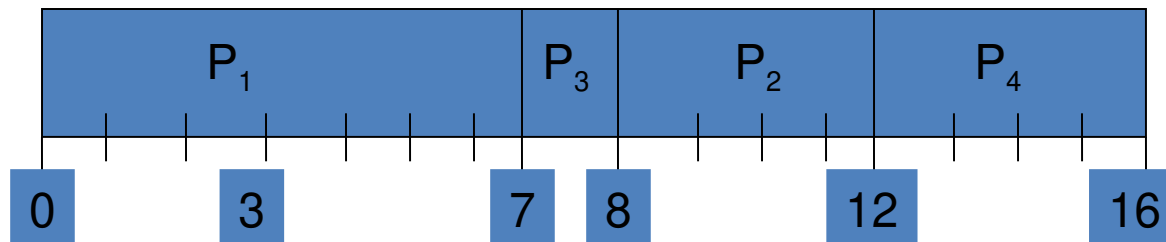
Scheduling Algorithms

- **Shortest Job First (SJF)**
- SJF pros & cons:
 - + Better average response time.
 - + It's the best you can do to minimize average response time-- can prove the algorithm is optimal.
 - Difficult to predict the future.
 - Unfair-- possible starvation (many short jobs can keep long jobs from making any progress).

Example of Non-Preemptive SJF

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



• Average waiting time = $(0 + 6 + 3 + 7)/4 = 4$

Problems with SJF

- Impossible to predict CPU burst times
- Schemes based on previous history (e.g. exponential averaging)
- SJF may lead to *starvation* of long jobs
- Solution to starvation- **Age** processes: increase **priority** as a function of waiting time

Shortest Remaining Time First(next)

- Preemptive version of the SJF algorithm
- CPU is allocated to the job that has the shortest CPU burst.
- Can be preempted if there is a newer job in the ready queue that is shorter than the one currently running

Round robin (RR)

- Often used for timesharing
- Ready queue is treated as a circular queue (FIFO)
- Each process is given a time slice called a *quantum* (q)
 - It is run for the quantum or until it finishes
 - RR allocates the CPU uniformly (fairly) across all participants.
- Definitions:
 - Context Switch Changing the processor from running one task (or process) to another. Implies changing memory.
 - Processor Sharing Use of a small quantum such that each process runs frequently at speed $1/n$.
 - Reschedule latency How long it takes from when a process requests to run, until it finally gets control of the CPU.

RR Gantt Chart

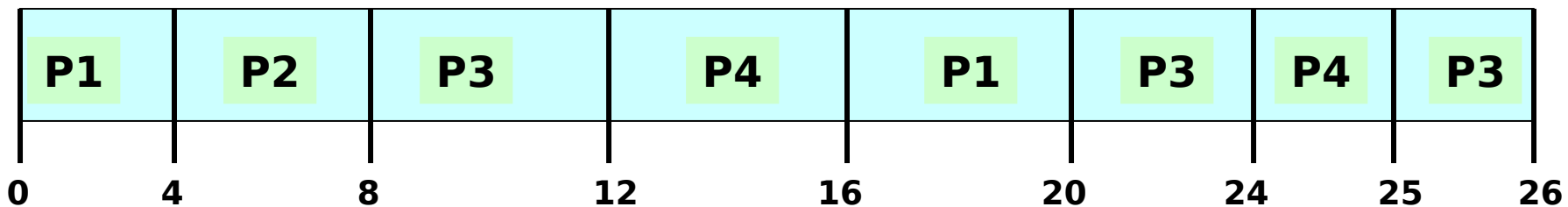
EXAMPLE DATA:

Process	Arrival Time	Service Time
1	0	8
2	1	4
3	2	9
4	3	5

Note:

Example violates rules for quantum size since most processes don't finish in one quantum.

Round Robin, quantum = 4, no priority-based preemption





$$\text{Average wait} = ((20-0) + (8-1) + (26-2) + (25-3)) / 4 = 74 / 4 = 18.5$$

RR Summary

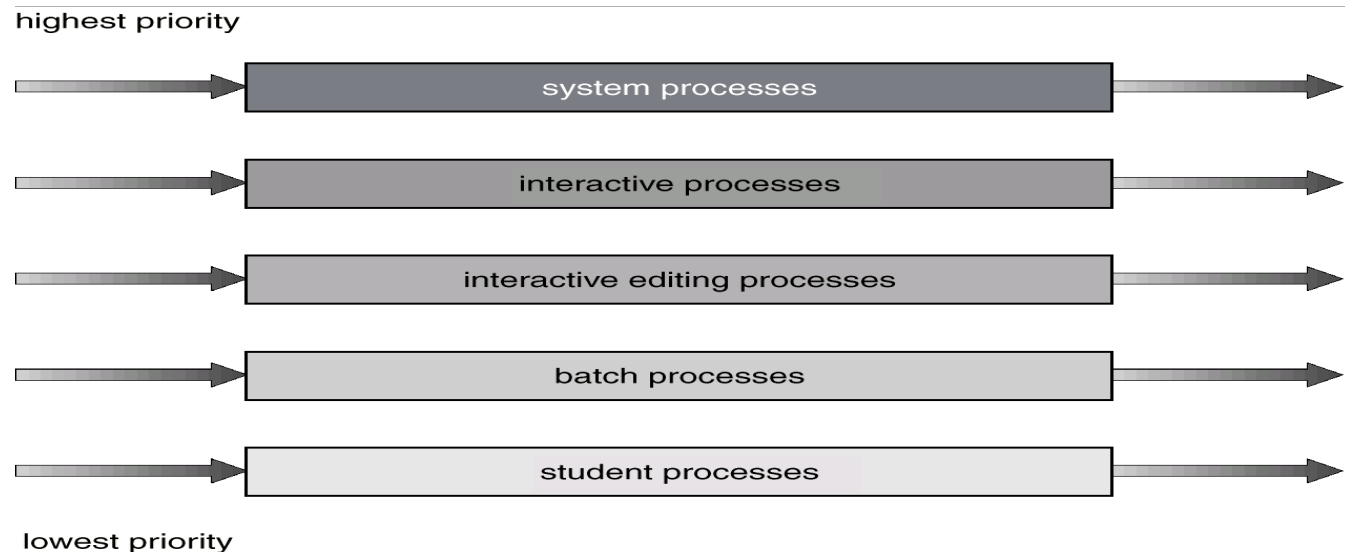
- Pros and Cons:
 - Better for short jobs (+)
 - Fair (+)
 - Context-switching time adds up for long jobs (-)
 - The previous examples assumed no additional time was needed for context switching – in reality, this would add to wait and completion time without actually progressing a process towards completion.
 - Remember: the OS consumes resources, too!
- If the chosen quantum is
 - too large, response time suffers
 - infinite, performance is the same as FIFO
 - too small, throughput suffers and percentage overhead grows
- Actual choices of timeslice:
 - UNIX: initially 1 second:
 - Worked when only 1-2 users
 - If there were 3 compilations going on, it took 3 seconds to echo each keystroke!
 - In practice, need to balance short-job performance and long-job throughput:
 - Typical timeslice **10ms-100ms**
 - Typical context-switch overhead **0.1ms – 1ms (about 1%)**

Multi-level Feedback Scheduling

- Another method for exploiting past behavior
 - Multiple queues, each with different priority
 - Higher priority queues often considered “foreground” tasks
 - Each queue has its own scheduling algorithm
 - E.g. foreground  RR, background  FCFS
 - Sometimes multiple RR priorities with quantum increasing exponentially (highest queue: 1ms, next: 2ms, next: 4ms, etc.)
 - Adjust each job’s priority as follows (details vary)
 - Job starts in highest priority queue
 - If entire CPU time quantum expires, drop one level
 - If CPU is yielded during the quantum, push up one level (or to top)

MULTI-LEVEL QUEUES:

- Each queue has its scheduling algorithm.
- Then some other algorithm (perhaps priority based) arbitrates between queues.
- Can use feedback to move between queues
- Method is complex but flexible.
- For example, could separate system processes, interactive, batch, favored, unfavored processes



- FCFS scheduling, FIFO Run Until Done:
 - Simple, but short jobs get stuck behind long ones
- RR scheduling:
 - Give each thread a small amount of CPU time when it executes, and cycle between all ready threads
 - Better for short jobs, but poor when jobs are the same length
- SJF/SRTF:
 - Run whatever job has the least amount of computation to do / least amount of remaining computation to do
 - Optimal (average response time), but unfair; hard to predict the future
- Multi-Level Feedback Scheduling:
 - Multiple queues of different priorities
 - Automatic promotion/demotion of process priority to approximate SJF/SRTF
- Priority Scheduling: