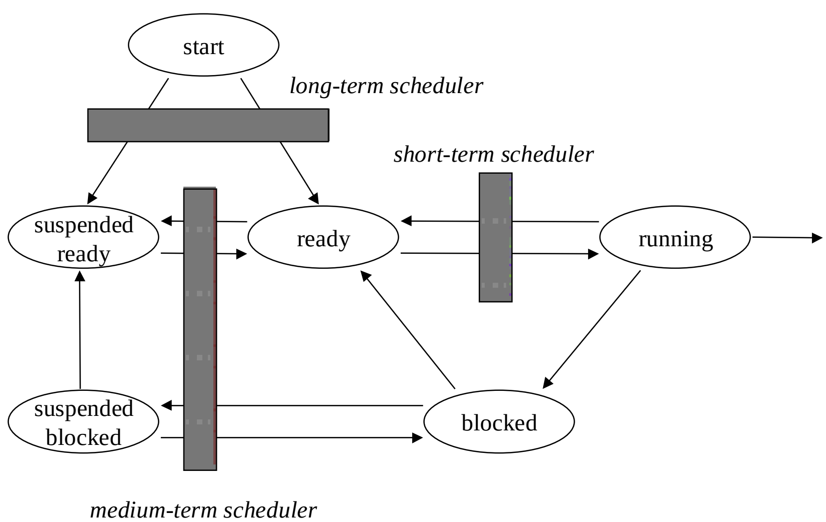
CPU Scheduling

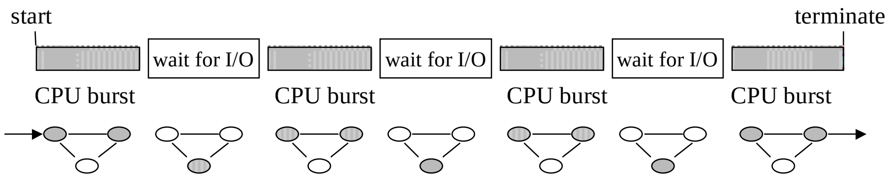
- Schedulers
- Structure of a CPU scheduler
- Criteria for scheduling
- Scheduling Algorithms
  - FCFS
  - SPN
  - SRT
  - MLFQ
- CPU scheduling in Unix

Schedulers



Short-Term Scheduling

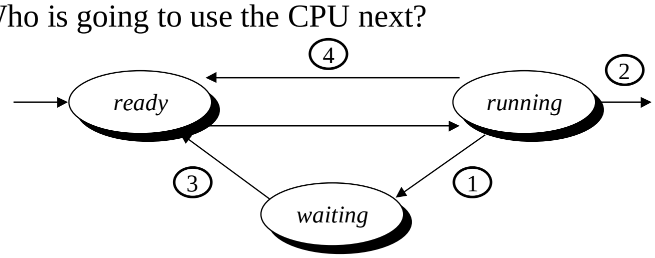
- Motivation for multiprogramming: Have multiple processes in memory to keep CPU busy.
- Typical execution profile of a process:



- **CPU scheduler** is managing the execution of CPU bursts, represented by processes in *ready* or *running* state.

Scheduling Decisions

- Who is going to use the CPU next?

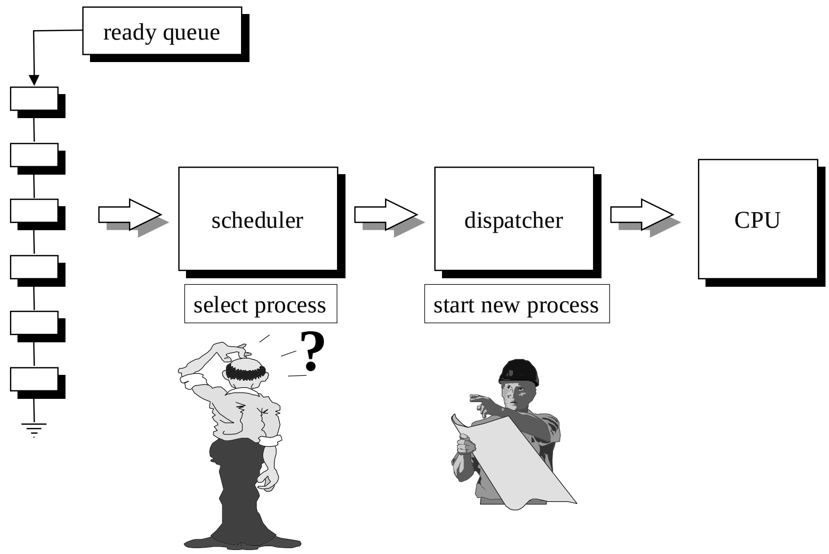


- Scheduling decision points:
  - **1**. The running process changes from *running* to *waiting* (current CPU burst of that process is over).
  - **2**. The running process terminates.
  - **3**. A waiting process becomes ready (new CPU burst of that process begins).
  - **4**. The current process switches from *running* to *ready* .

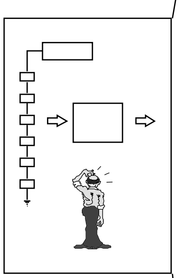
non-preemptive (1, 2)

preemptive (1–4)

Structure of a Scheduler (I)

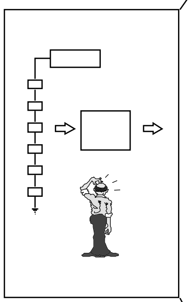


What Is a Good Scheduler? Criteria



- User oriented:
  - **Turnaround time** : time interval from submission of job until its completion
  - **Waiting time** : sum of periods spent waiting in ready queue
  - **Response time** : time interval from submission of job to first response
  - **Normalized turnaround time**: ratio of turnaround time to service time
- System oriented:
  - **CPU utilization** : percentage of time CPU is busy
  - **Throughput** : number of jobs completed per time unit
- Any good scheduler should:
  - *maximize* CPU utilization and throughput
  - *minimize* turnaround time, waiting time, response time
- Huh?
  - maximum/minimum values *vs.* average values *vs.* variance

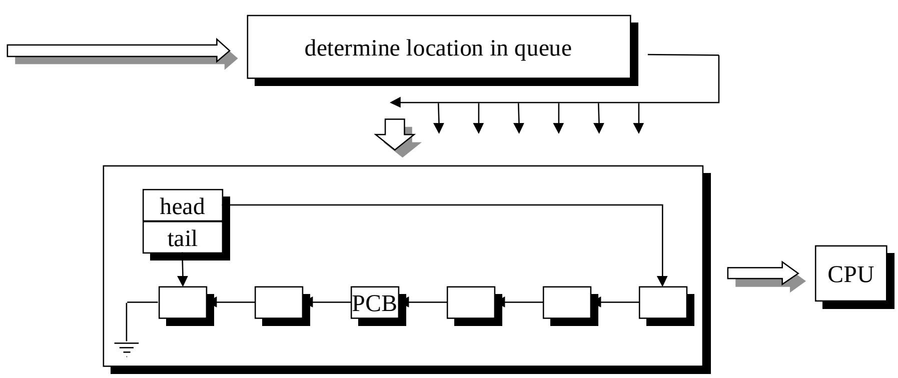
Scheduling Algorithms



- **FCFS** : First-come-first-served
- **SPN:** Shortest Process Next
- **SRT**: Shortest Remaining Time
- priority scheduling
- **RR** : Round-robin
- Multilevel feedback queue scheduling
- Multiprocessor scheduling

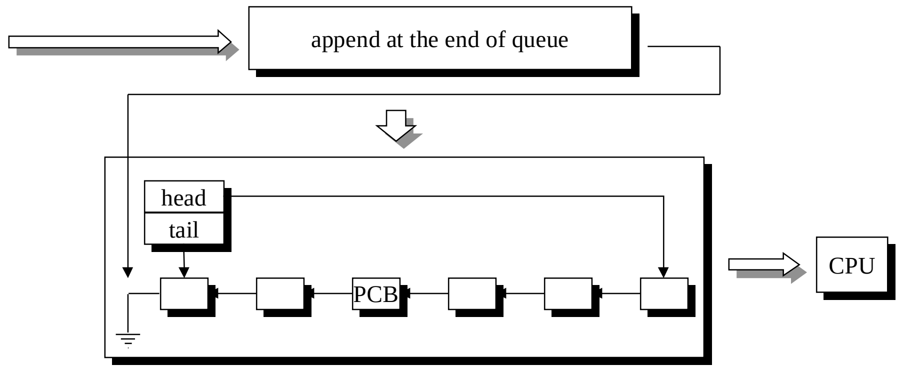
Structure of a Scheduler (II)

(conceptual structure)



- Incoming process is put into right location in ready queue.
- Dispatcher always picks first element in ready queue.

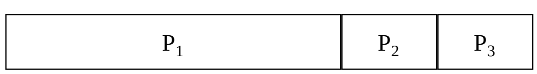
First-Come-First-Served (FCFS)



- Advantages:
  - very simple
- Disadvantages:
  - long average and worst-case waiting times
  - poor dynamic behavior (convoy effect)

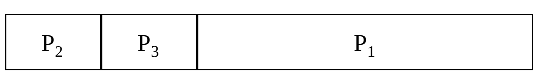
Waiting Times for FCFS

- Example: $P_1 = 24$, $P_2 = 6$, $P_3 = 6$



$W_{awg} = (24+30)/3 = 18$
$W_{wc} = 30$

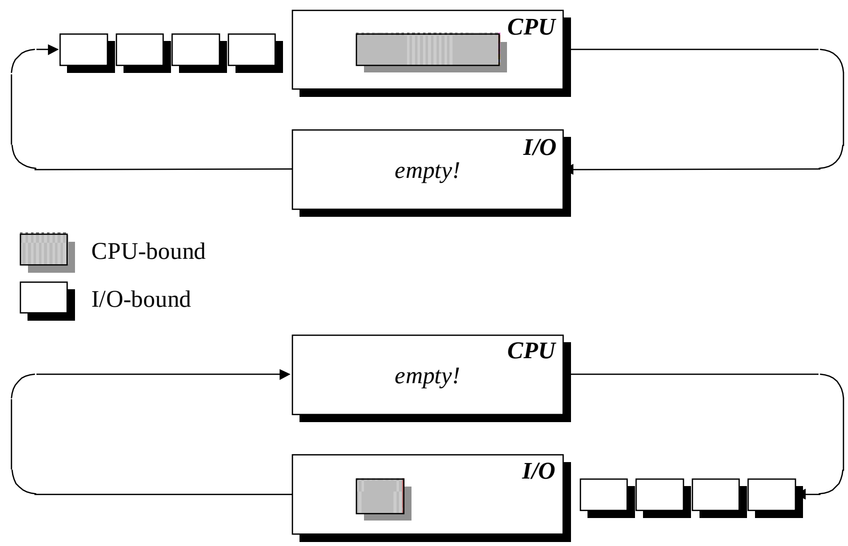
Different arrival order:



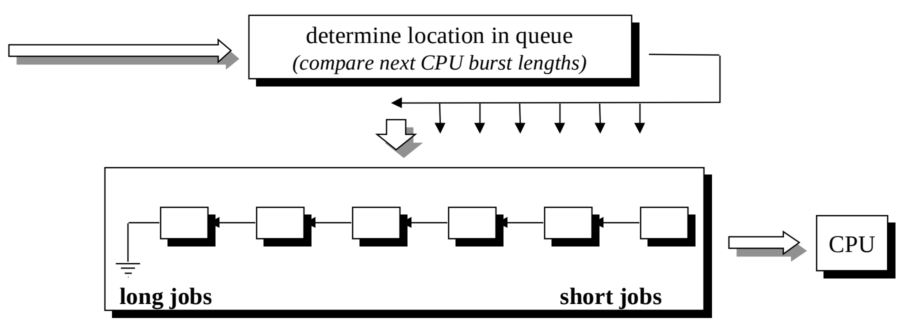
$W_{awg} = (6+12)/3 = 6$
$W_{wc} = 12$

- Average waiting times is not minimal.
- Waiting times may substantially vary over time.
- Worst-case waiting times can be very long.

Convoy Effects



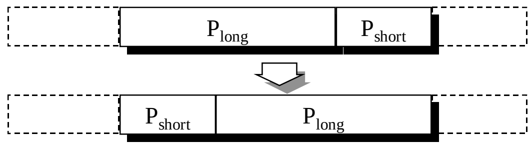
Shortest Process Next



- Whenever CPU is idle, picks process with *shortest next CPU burst*.
- Advantages: minimizes average waiting times.
- Problem: How to determine length of *next* CPU burst?
- Problem: starvation of jobs with long CPU bursts.

SJF Minimizes Average Waiting Time

- Provably optimal: Proof: swapping of jobs



$dW = t_{short} - t_{long} < 0$

- Example:

6	12	8	4	W = 6+18+26 = 50
6	8	12	4	W = 6+14+26 = 46
6	8	4	12	W = 6+14+18 = 38
6	4	8	12	W = 6+10+18 = 34
4	6	8	12	W = 4+10+18 = 32

- Question: How to determine execution time of *next* CPU burst ?!
  - wild guess?
  - code inspection?

- Forecasting (i.e. estimation)

$$S_{n+1} = F(T_n, T_{n-1}, T_{n-2}, T_{n-3}, T_{n-4}, ...)$$

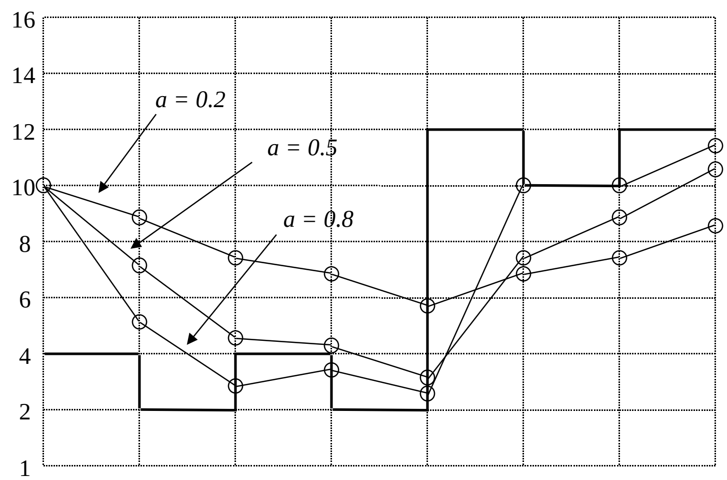
- Simple forecasting function: exponential average:

$$S_{n+1} = a\, T_n + (1-a)\, S_n$$

- Example: a = 0.8

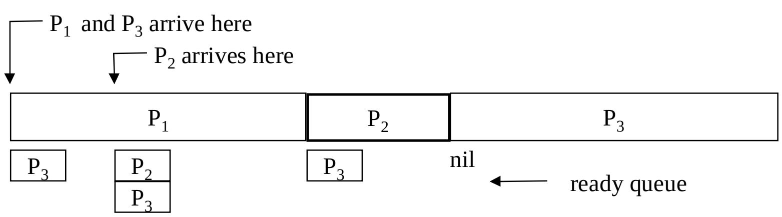
$$S_{n+1} = 0.8T_n + 0.16T_{n-1} + 0.032T_{n-2} + 0.0064T_{n-3} + ...$$

Exponential Averaging: Example

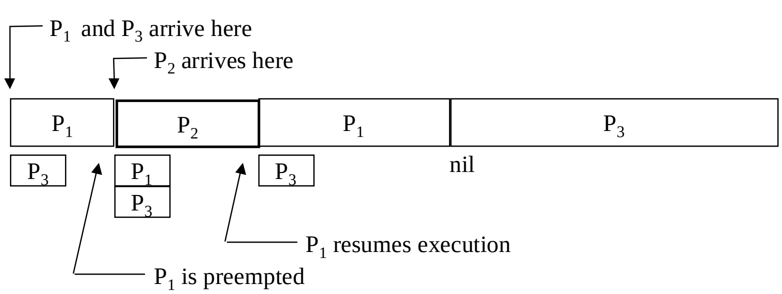


Preemptive SPN: Shortest-*Remaining*-*Time*-First

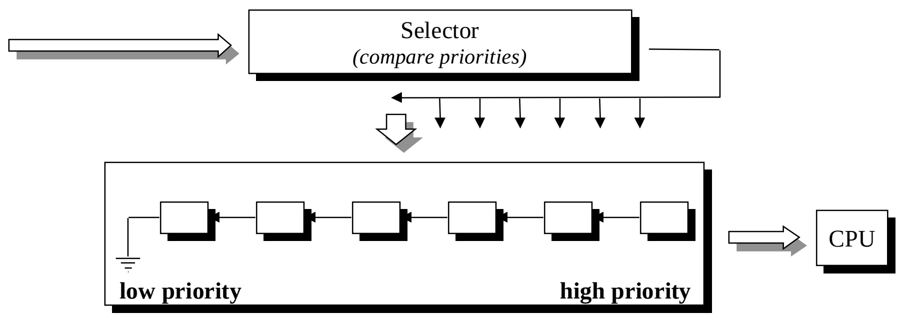
- SPN:



- SRT:

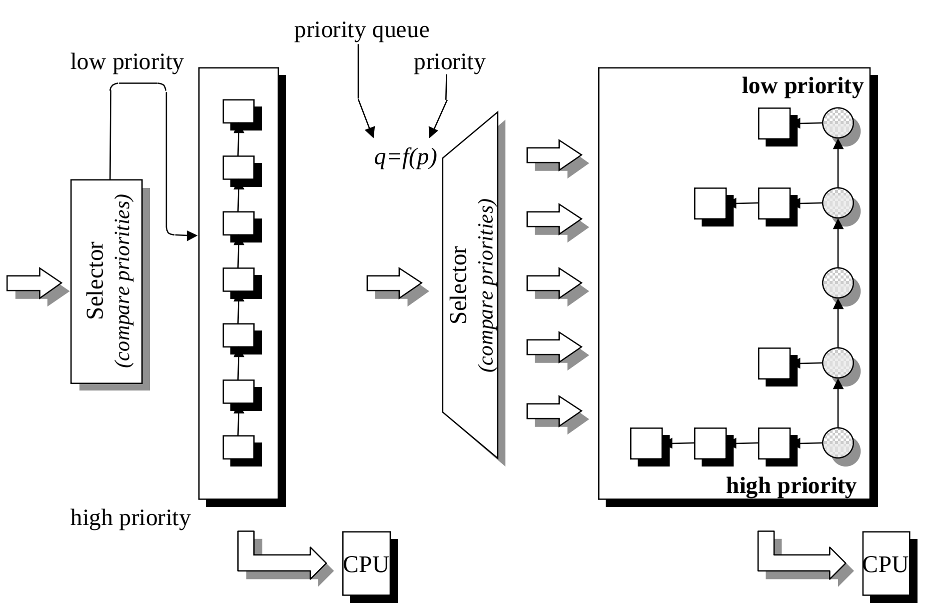


Priority Scheduling

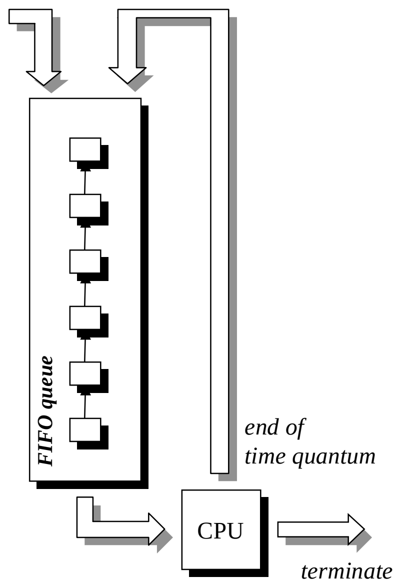


- Whenever CPU is idle, picks process with *highest priority*.
- Priority:
  - process class, burst length, urgency, pocket depth.
- Unbounded blocking: Starvation
  - Increase priority over time: aging

- **Conceptually:**
- **Priority Queues**

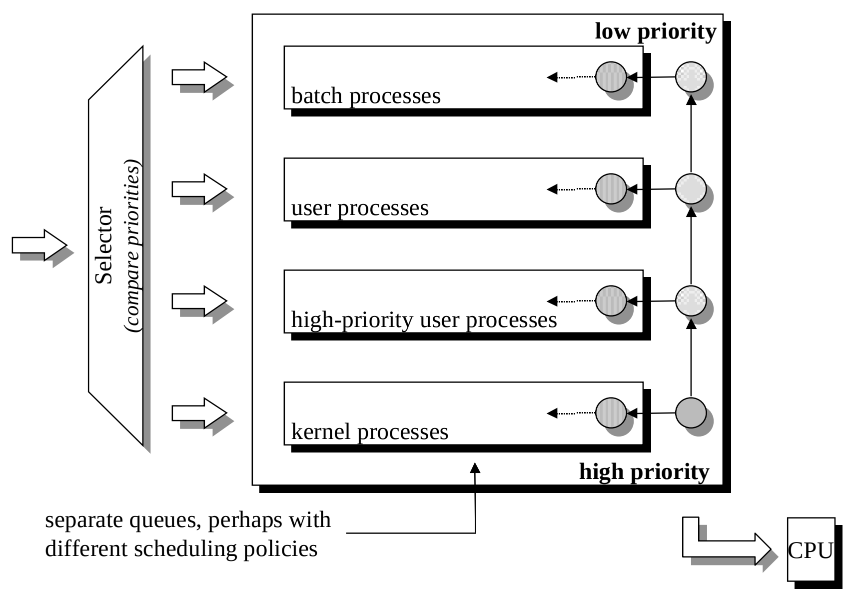


Round-Robin



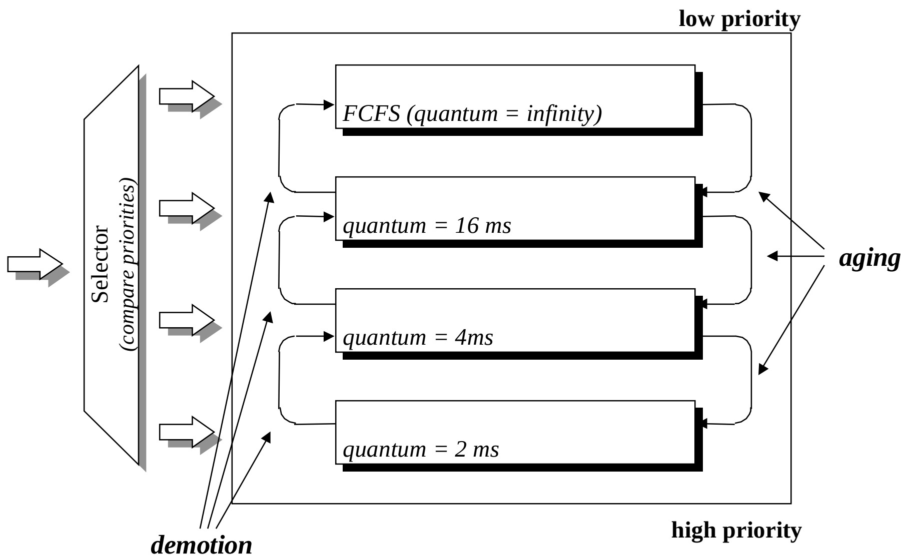
- FIFO with preemption after *time quantum*
- Method for time sharing
- Choice of time quantum:
  - large: FCFS
  - small: Processor sharing
- Time quantum also defines context-switching overhead
- Response time smaller with large time quantums

Multilevel Queue Scheduling



Multilevel Feedback Queue Scheduling

(conceptually)



MFBS: Implementation (Unix System V)

- Clock handler generates 60 clock ticks per second.
- Each PCB contains a field **CPU** ("recent CPU usage"), which is incremented on every clock tick while process is running.
- Every 60 ticks scheduler is awakened and
  - ajusts recent CPU usage according to a decay function:

    *decay(**CPU**) = **CPU**/2*
  - recalculates priorities according to following formula (higher priorities have lower priority values!):

    *priority = **CPU**/2 + base_priority*
- Decay rate controls aging.
- Priority recalculation controls demotion
- **Note**: This is a simplified view! (For a more detailed description refer to M.J.Bach, *The Design of the UNIX Operating System.*)

MFBS on UNIX System V: Example

- 3 processes, each with base priority 60:

	Process A		Process B		Process C	
time	priority	CPU count	priority	CPU count	priority	CPU count
	60	0 1 ... 60	60	0	60	0
1	75	30	60	0 1 ...	60	0
2	67	15	75	30	60	0 1 ...
3	63	7 8 ... 67	67	15	75	30
4	76	33	63	7 8 ... 67	67	15
5	68	16	76	33	63	7