Operating Systems

Lecture 2.2 - Process Scheduling

Golestan University

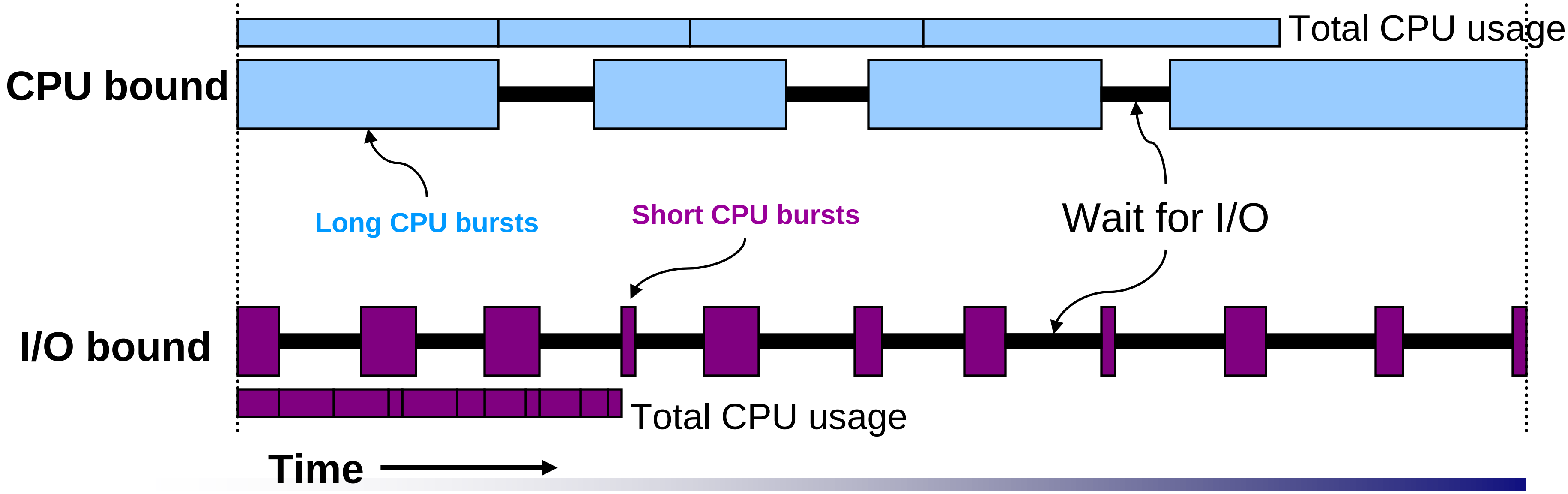
Hossein Momeni
momeni@iust.ac.ir

Scheduling

- What is scheduling?
  - Goals
  - Mechanisms
- Scheduling on batch systems
- Scheduling on interactive systems
- Other kinds of scheduling
  - Real-time scheduling

Why schedule processes?

- Bursts of CPU usage alternate with periods of I/O wait
- Some processes are *CPU-bound*: they don't make many I/O requests
- Other processes are *I/O-bound* and make many kernel requests



Scheduling goals

- All systems
  - Fairness: give each process a fair share of the CPU
  - Enforcement: ensure that the stated policy is carried out
  - Balance: keep all parts of the system busy
- Batch systems
  - Throughput: maximize jobs per unit time
  - Turnaround time: minimize time users wait for jobs
  - CPU utilization: keep the CPU as busy as possible
- Interactive systems
  - Response time: respond quickly to users' requests
  - Proportionality: meet users' expectations
- Real-time systems
  - Meet deadlines: missing deadlines is a system failure!
  - Predictability: same type of behavior for each time slice

Measuring scheduling performance

- Throughput
  - Amount of work completed per second (minute, hour)
  - Higher throughput usually means better utilized system
- Response time
  - Response time is time from when a command is submitted until results are returned
  - Can measure average, variance, minimum, maximum, …
- Turnaround time
  - amount of time to execute a process (from delivery to execute = waiting time for entry to memory + waiting time for entry to ready queue + run time + I/O time)
- Usually not possible to optimize for *all* metrics with the same scheduling algorithm

Optimization Criteria

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

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

- Context-switch time is overhead; the system does no useful work while switching

Terminology: Preemptive vs. non-Preemptive

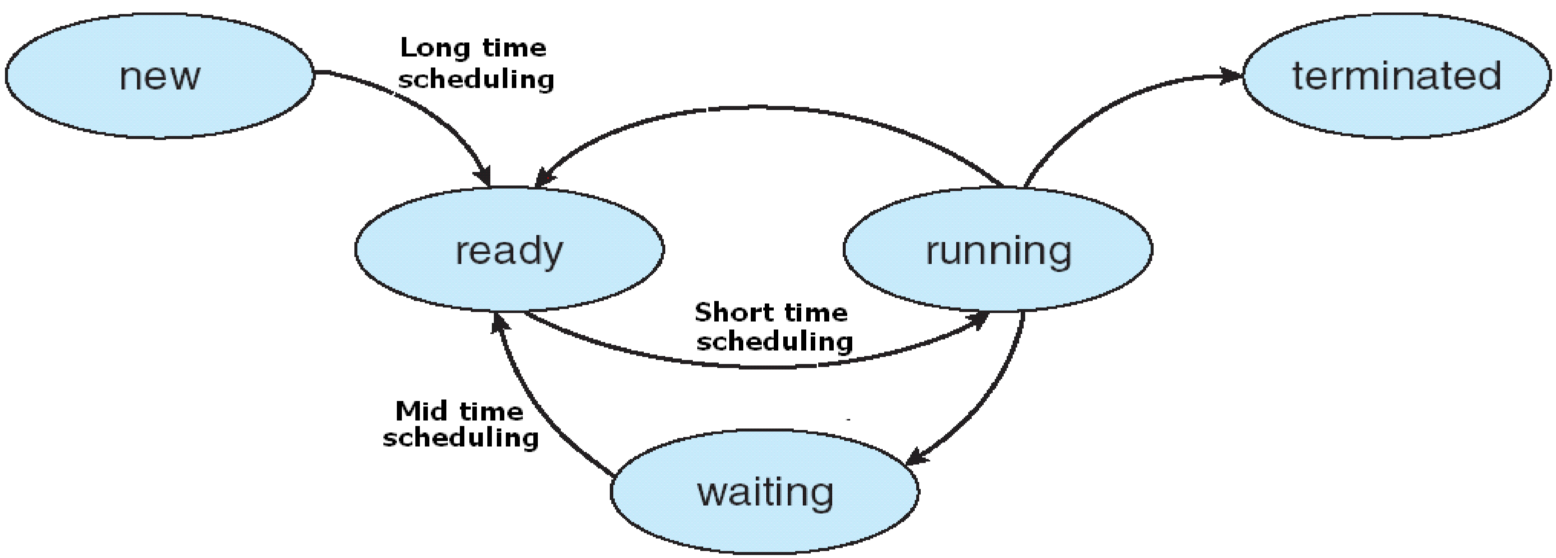
- **Preemptive:** A Process can be suspended and resumed
- **Non-preemptive:** A process runs until it voluntarily gives up the CPU (waiting on I/O or terminate).

- Most modern OSs use preemptive CPU scheduling, implemented via timer interrupts.

- Non-preemptive is used when suspending a process is impossible or very expensive: e.g., can't "replace" a flight crew in middle of flight.

Scheduling type:

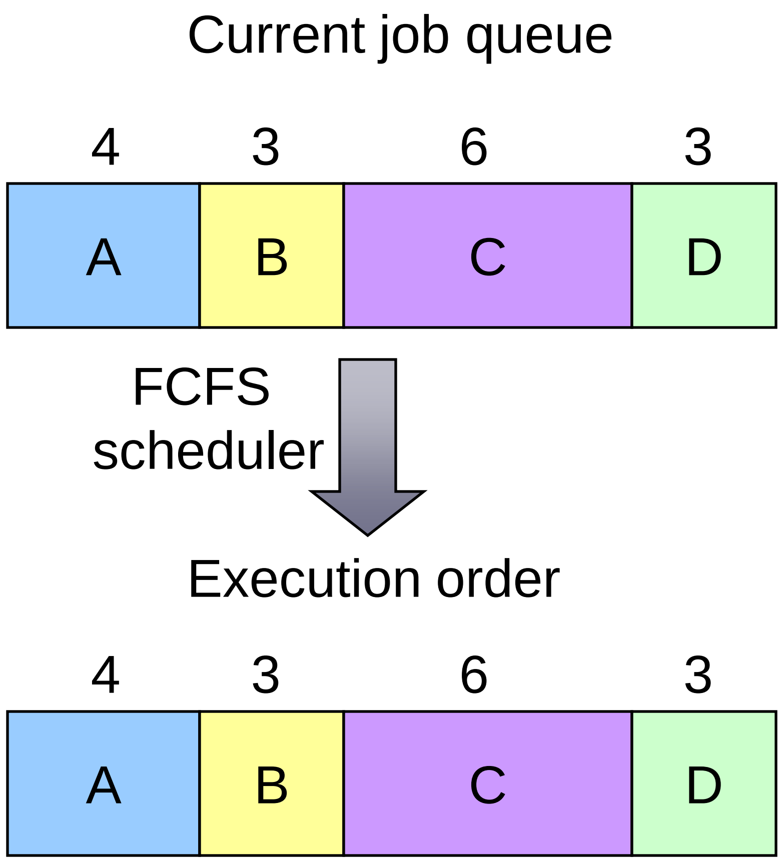
- I/O Scheduling
- Short time scheduling
  - Switch Ready to Running
- Long time scheduling
  - Switch New to Ready
- Mid time scheduling
  - Switch Waiting to Ready



Scheduling Policies

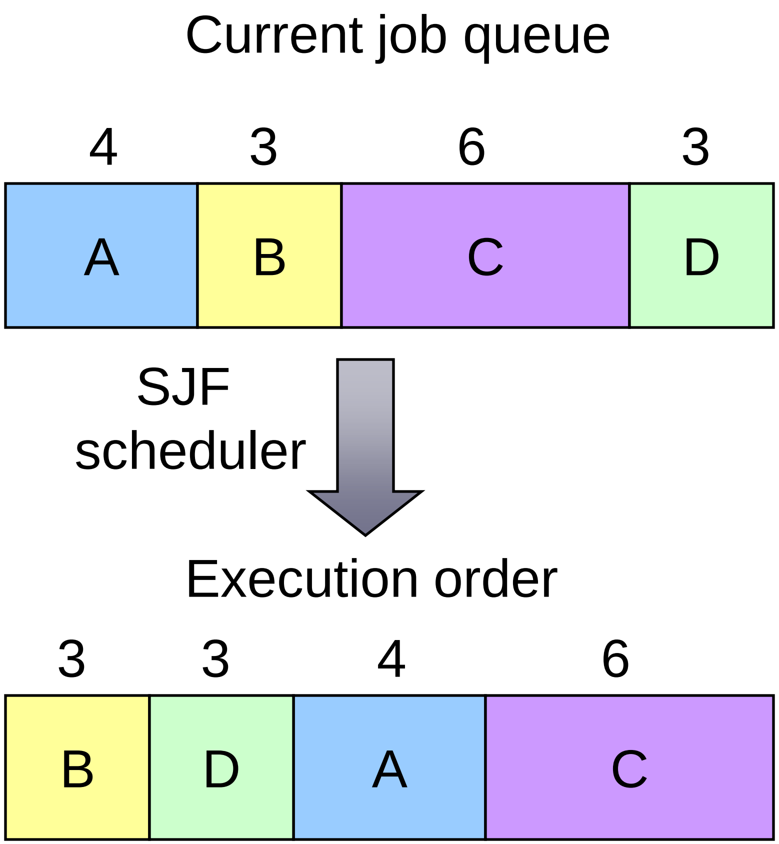
- Batch systems:
  - First Come First Served
  - Shorted Job First
  - Shortest Remaining Time Next
- Interactive systems:
  - Round Robin
  - Priority Scheduling
  - Multiple Queues
  - Shortest Process Next
  - Guaranteed Scheduling
  - Lottery Scheduling
  - Fair-share Scheduling
- Real-time systems:
  - Static vs. dynamic

First Come, First Served (FCFS)



- Goal: do jobs in the order they arrive
  - Fair in the same way a bank teller line is fair
- Privilege:
  - Simple algorithm!
  - Fairness, No Starvation, low Overhead
- Problem: long jobs delay every job after them
  - Many processes may wait for a single long job
  - Priority is not supported
  - High waiting time and TAT
  - Not suitable for I/O bound process

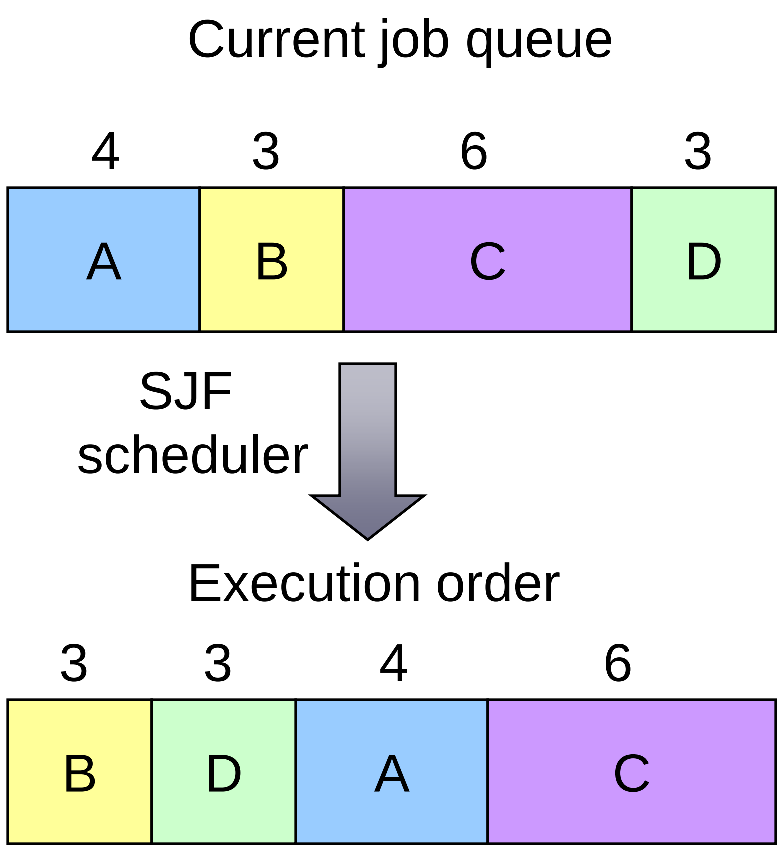
Shortest Job First (SJF)



- Goal: do the shortest job first
  - Short jobs complete first
  - Long jobs delay every job after them
- Jobs sorted in increasing order of execution time
  - Ordering of ties doesn't matter
- Advantages:
  - Minimum TAT and Waiting Time
  - Low overhead
- Disadvantages:
  - Job Execution Time Estimation
  - Starvation for long job

Commensurate with Batch Systems

Shortest Remaining Time First (SRTF)



- Shortest Remaining Time First (SRTF):
- preemptive form of SJF
- Problem: how does the scheduler know how long a job will take?
- Advantages:
  - Minimum TAT and Waiting Time
- Disadvantages:
  - Job Execution Time Estimation
  - Starvation for long job

Commensurate with Batch Systems

Highest Response Ration Next (HRRN)

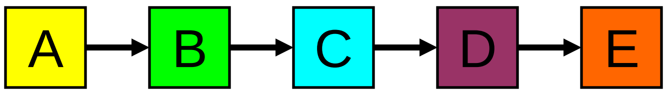
- Priority
- Non-preemptive
- Aging

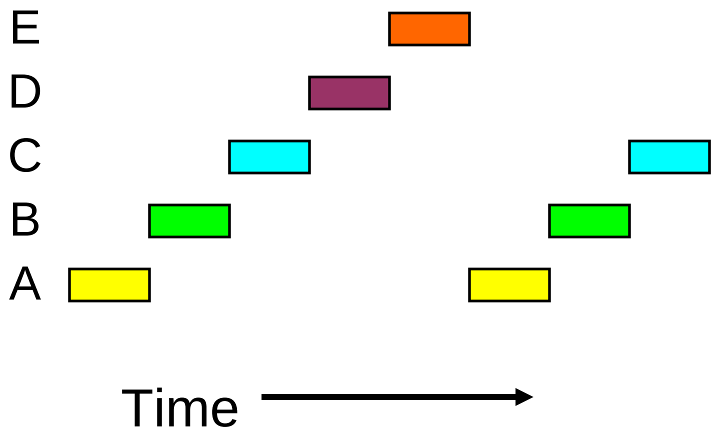
  Response Ratio= (waiting time + CBT)/CBT

- Advantages:
  - Not bad Waiting time and TAT
  - No Starvation
  - Fairness
  - Low overhead
- Disadvantage:
  - Estimation
  - Increase cost

Round Robin (RR) scheduling

- Preemptive
- Give each process a fixed Time Slice (***quantum***)
- Rotate through "ready" processes
- Each process makes some progress
- No starvation, Fairness, Simple
- Context switching overhead
- High RT an WT
- Useful for CPU bound





- What's a good quantum?
  - Too short: many process switches hurt efficiency
  - Too long: poor response to interactive requests
  - Typical length: 10–50 ms

Commensurate with Interactive Systems

Example:

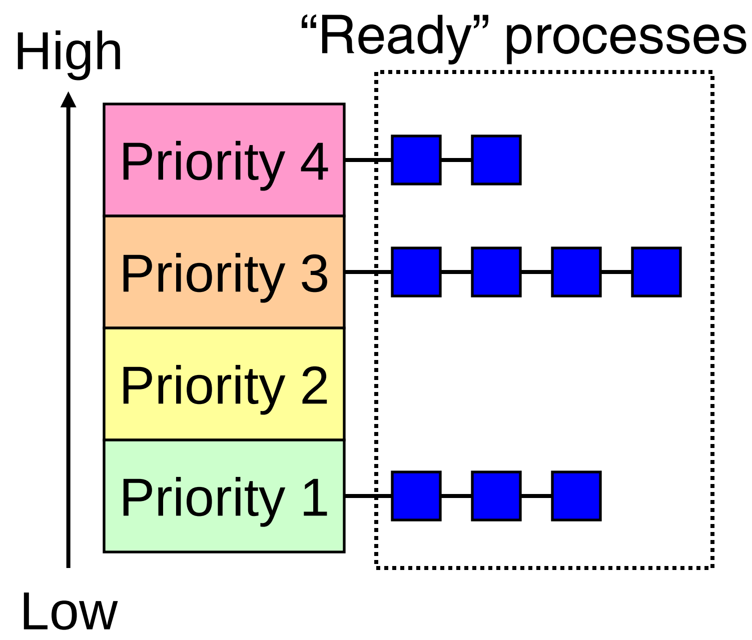
- Quantum time: 20 ms
- Context switch time: 5 ms
- Overhead cost time: 5/25 = 20%

- Quantum time: 100 ms
- Context switch time: 5 ms
- Overhead cost time: 5/105 < 5%

- Short Quantum: Low performance
- Long Quantum: High performance & bad response time

Priority scheduling

- Assign a priority to each process
  - "Ready" process with highest priority allowed to run
  - Running process may be interrupted after its quantum expires
- Priorities may be assigned dynamically
  - Reduced when a process uses CPU time
  - Increased when a process waits for I/O (may be set priority to 1/f, f is used time in last quantum)
- Often, processes grouped into multiple queues based on priority, and run round-robin per queue



Commensurate with Interactive Systems

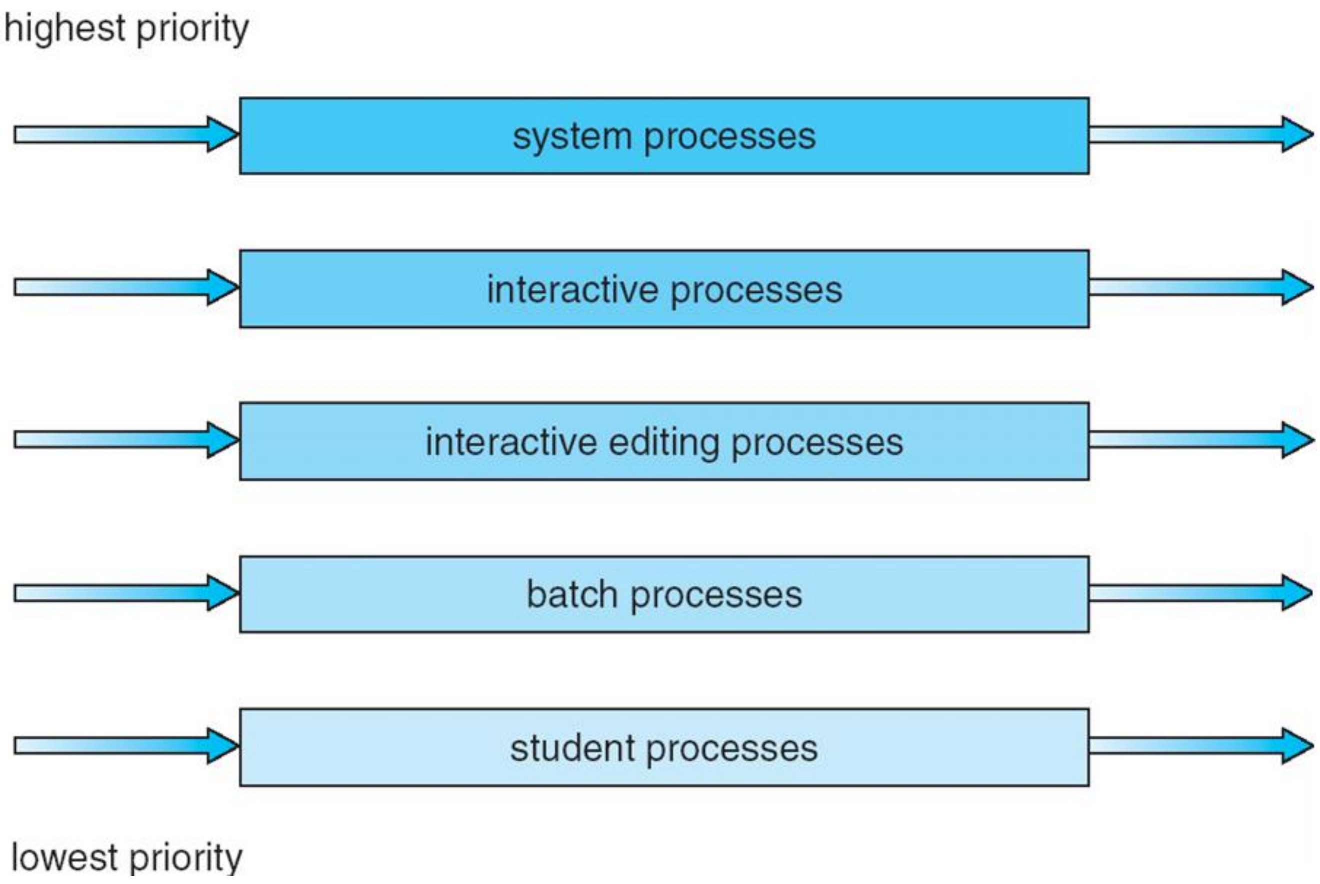
Problem ≡ **Starvation** – low priority processes may never execute

Solution ≡ **Aging** – as time progresses increase the priority of the process

Multilevel Queue

- Ready queue is partitioned into separate queues
- Each queue has its own scheduling algorithm
  - RR
  - FCFS
- Scheduling must be done between the queues

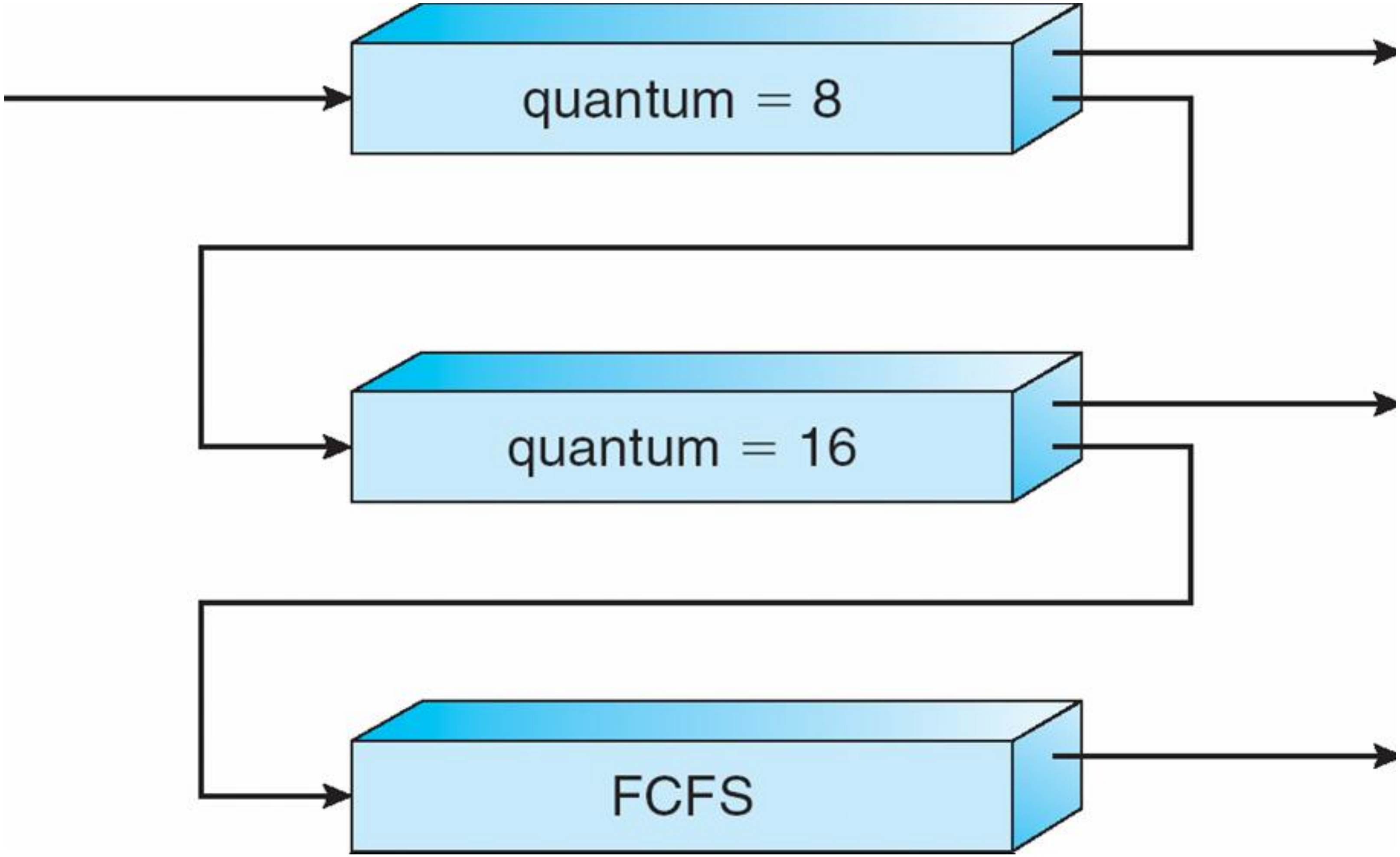
Multilevel Queue Scheduling



Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue (RR, FCFS)
  - method used to determine when to upgrade a process
- States of process at end of execution in a queue:
  - Terminate
  - I/O and the Upgrade
  - Not complete and Downgrade

Multilevel Feedback Queues



Lottery scheduling

- Give processes "tickets" for CPU time
  - More tickets => higher share of CPU
- Each quantum, pick a ticket at random
  - If there are *n* tickets, pick a number from 1 to *n*
  - Process holding the ticket gets to run for a quantum(1/n)
- Over the long run, each process gets the CPU *m*/*n* of the time if the process has *m* of the *n* existing tickets
- Tickets can be transferred
  - Cooperating processes can exchange tickets
  - Clients can transfer tickets to server so it can have a higher priority

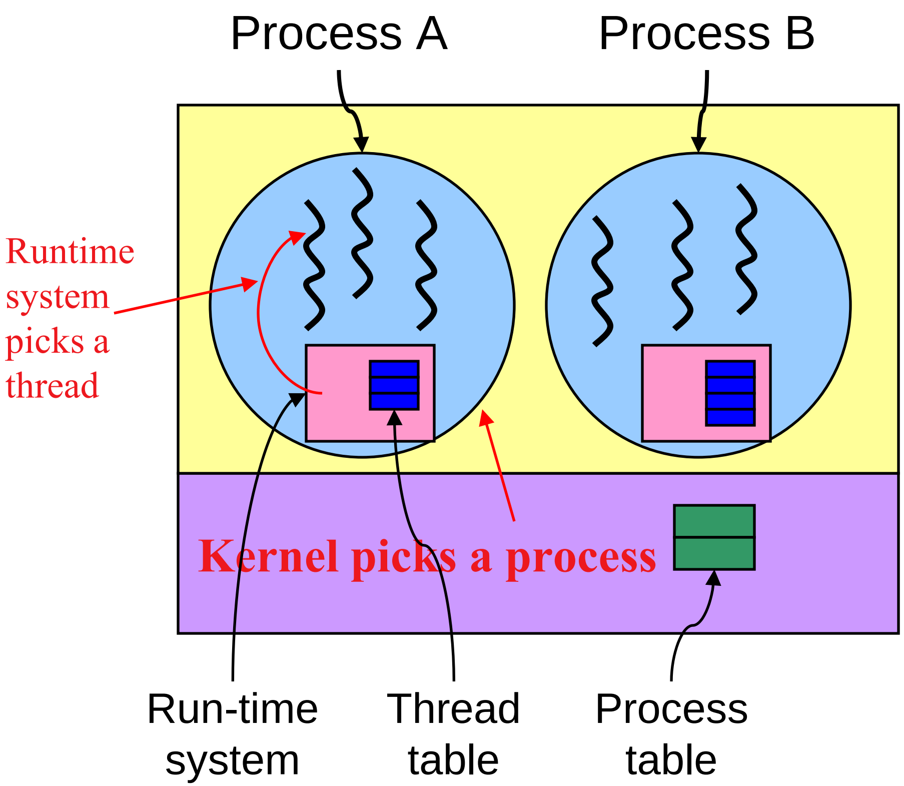
Scheduling in Real-Time Systems

Schedulable real-time system

- Given
  - *m* periodic events
  - event *i* occurs within period P_i and requires C_i seconds
- Then the load can only be handled if

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

Scheduling user-level threads

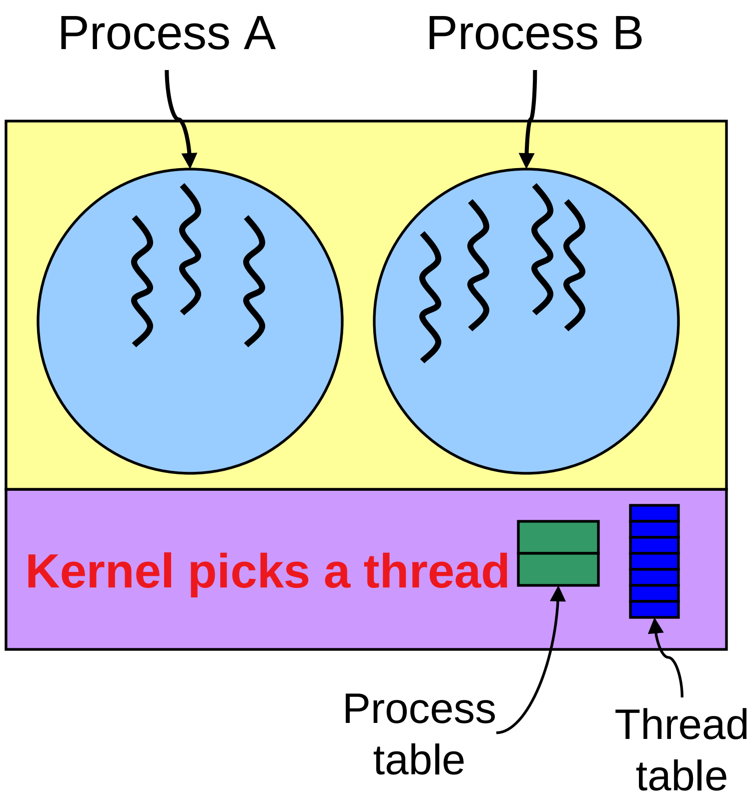


Possible scheduling of user-level threads

- 50-msec process quantum
- threads run 5 msec/CPU burst

- Kernel picks a process to run next
- Run-time system (at user level) schedules threads
  - Run each thread for less than process quantum
  - Example: processes get 50ms each, threads get 5ms each
- Example schedule: A1,A2,A3,A1,B1,B3,B2,B3
- Not possible: A1,A2,B1,B2,A3,B3,A2,B1

Scheduling kernel-level threads



- Kernel schedules each thread
  - No restrictions on ordering
  - May be more difficult for each process to specify priorities
- Example schedule: A1,A2,A3,A1,B1,B3,B2,B3
- Also possible: A1,A2,B1,B2,A3,B3,A2,B1