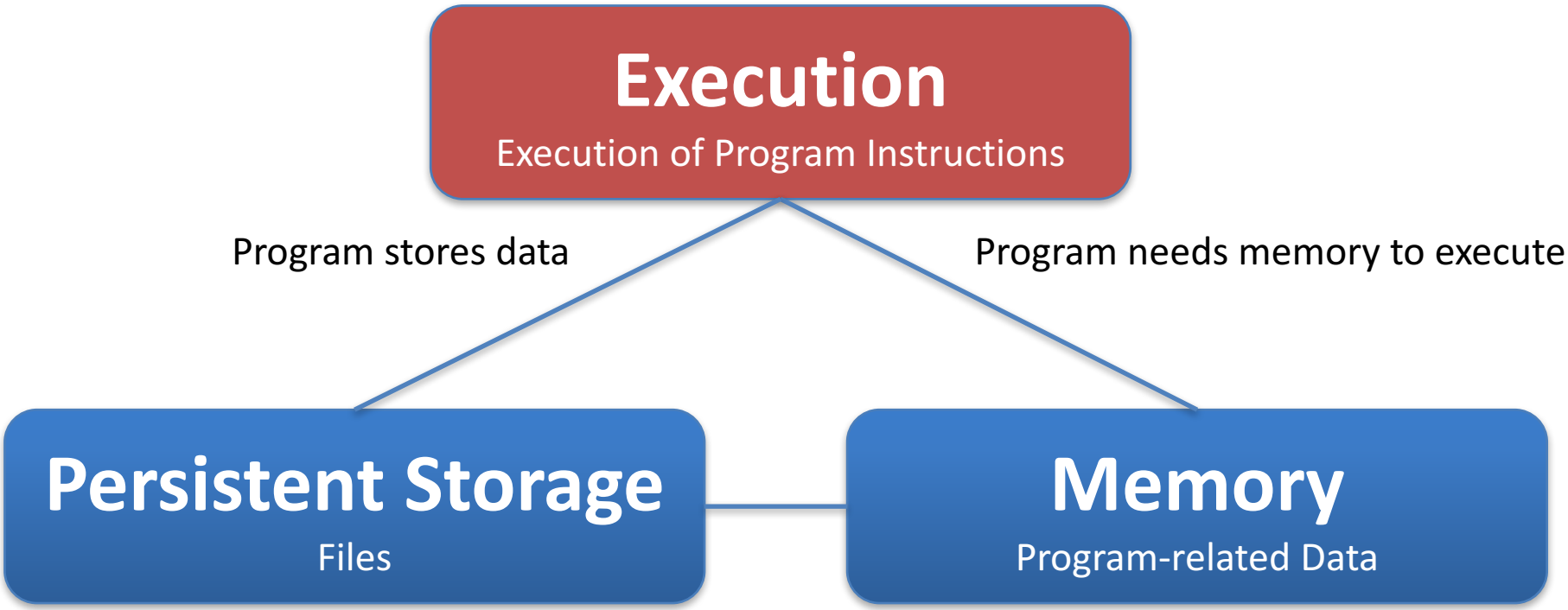
Scheduling

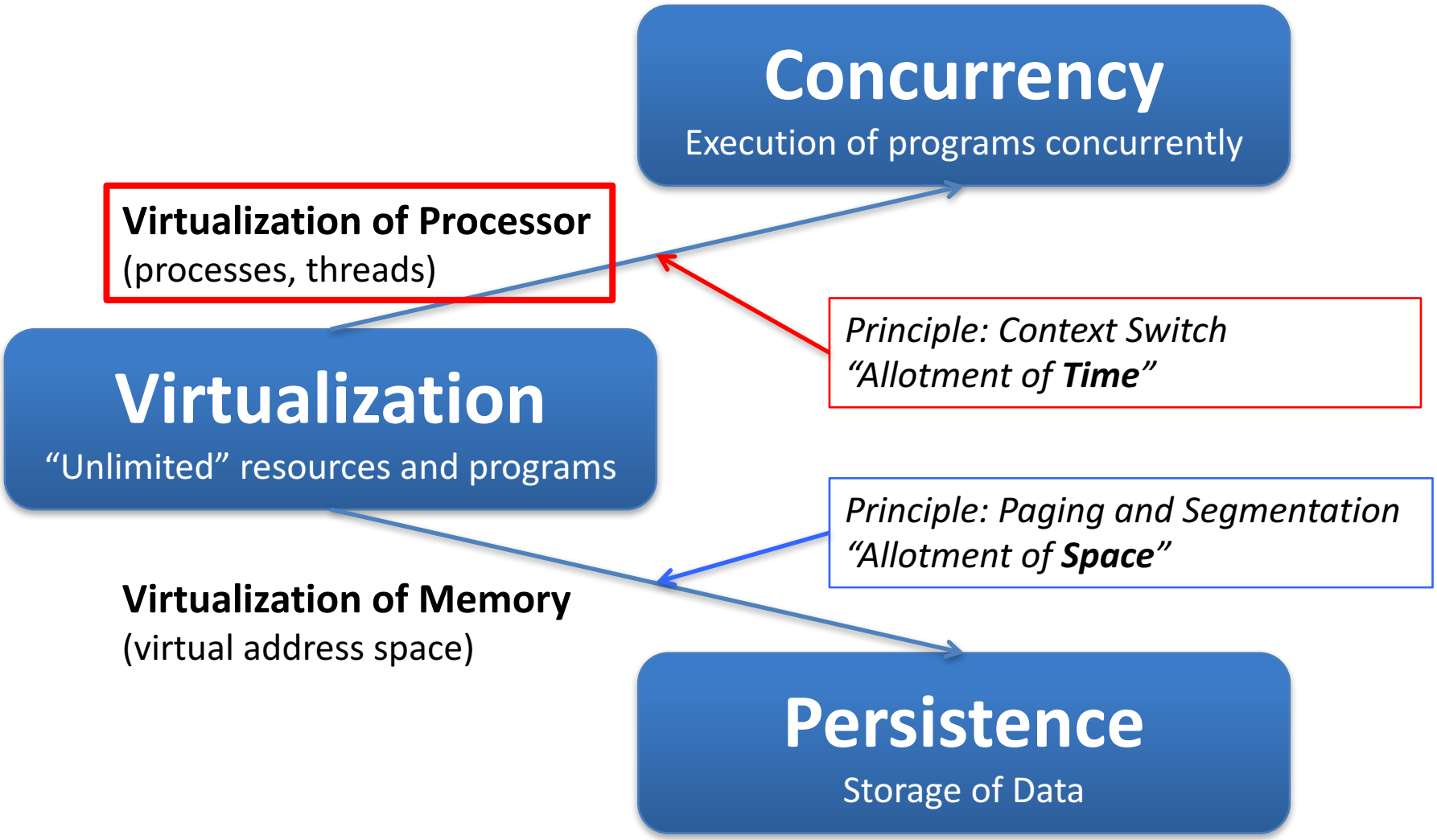
CS3026 Operating Systems

Lecture 13

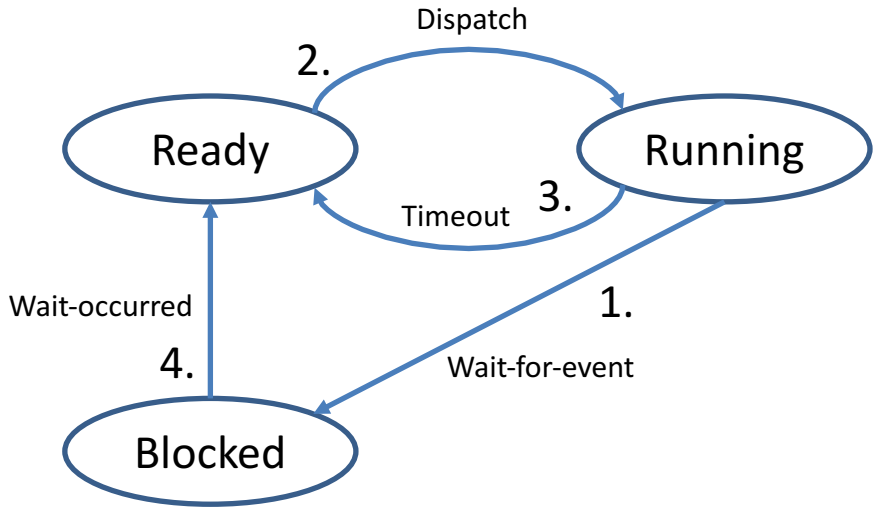
Program Execution



Core Concepts



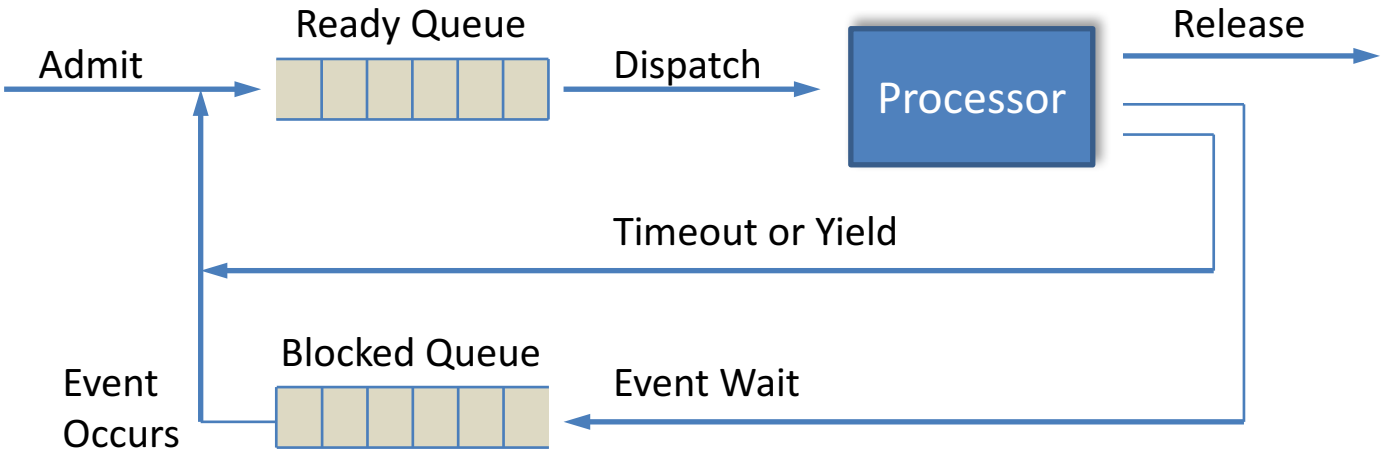
Process Execution (Dispatch)



1. Process blocked for I/O
2. Dispatcher schedules another process
3. Dispatcher interrupts process because its time slice expired
4. Input becomes available, blocked process made ready

- We can distinguish three basic process states during execution
  - Running: actually using the CPU
  - Ready: being runnable, temporarily stopped (time-out) to let another process execute
  - Blocked/Waiting: unable to run until some external event happens, such as I/O completion

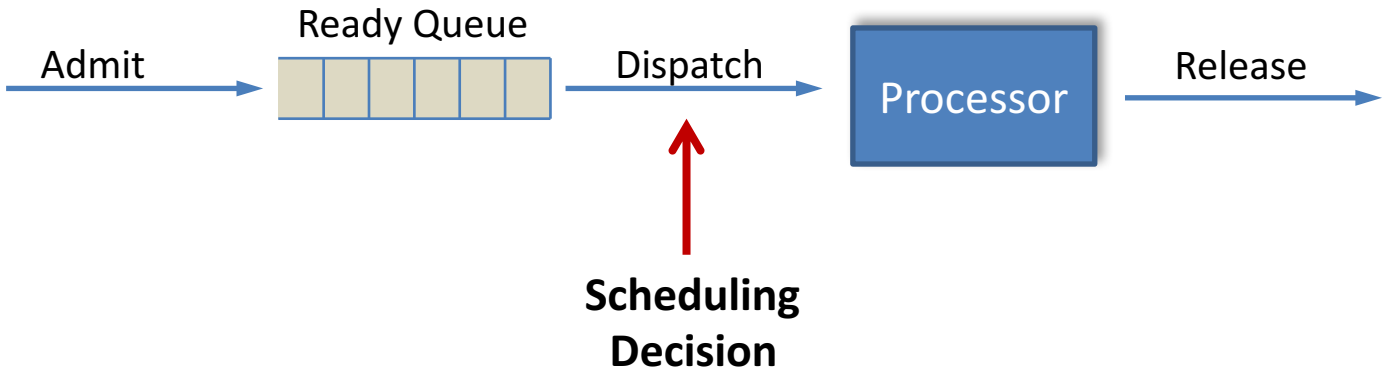
Dispatcher



- Scheduling:
  - Which process is the next to be dispatched from the Ready Queue?

Scheduling

- CPU Scheduling
  - Decides which process to execute next
  - is the activity of selecting the next process in the Ready queue for execution on a processor

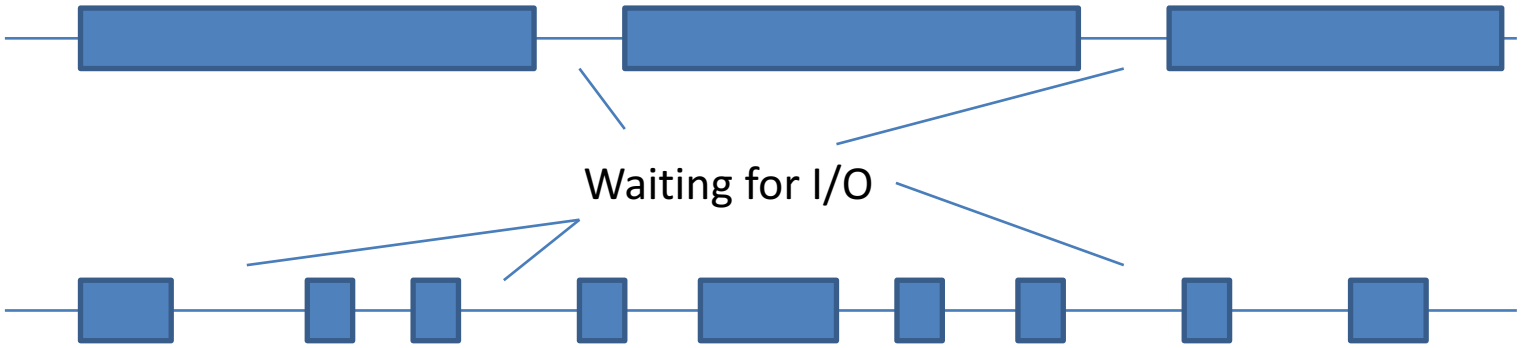


Scheduling

- A scheduling decision takes place when an event occurs that interrupts the execution of a process
- Possible events
  - Clock interrupts
  - I/O interrupts
  - Operating system calls
  - Signals (e.g., semaphores)

Characteristics of Process Execution

CPU – intensive Execution: long CPU bursts

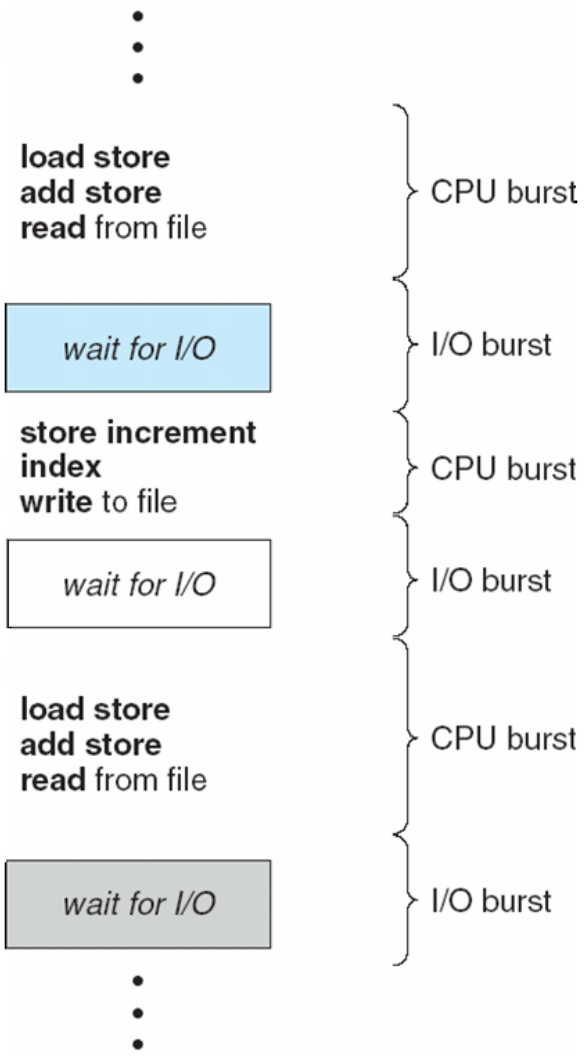


I/O – intensive Execution: short CPU bursts

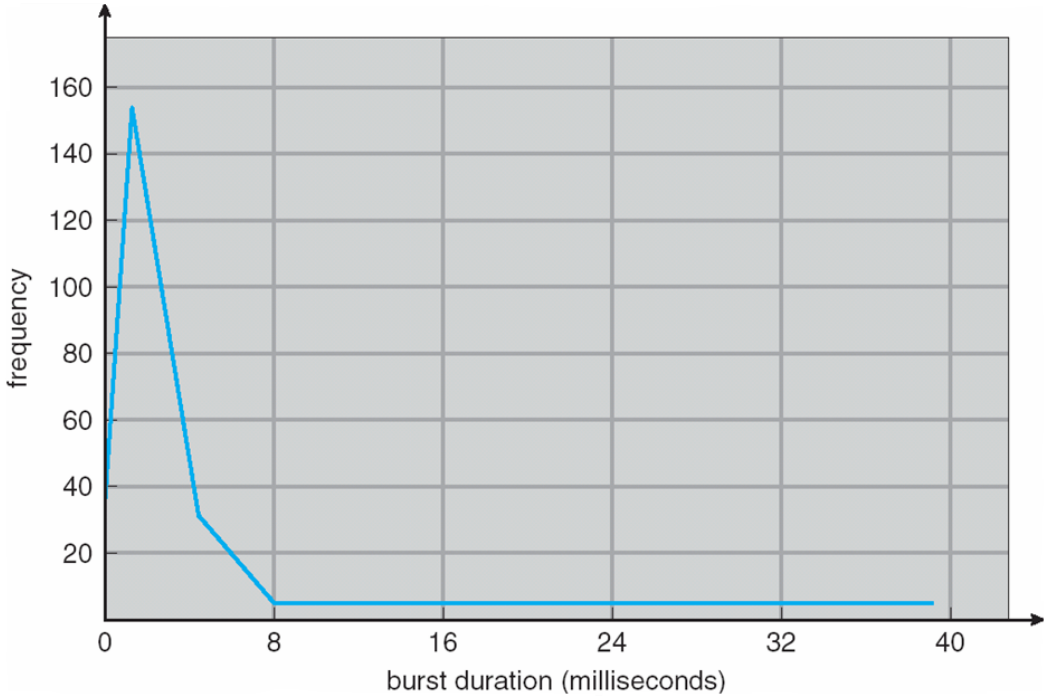
- Processes can be categorized as
  - CPU bound: uses mainly CPU over long periods of time
    - the faster a CPU, the earlier such a process is finished
  - I/O bound: are limited by the speed of I/O devices
    - The faster a device, the earlier such a process is finished

Execution in CPU bursts

- All processes execute in CPU bursts
  - CPU is not permanently used by one process, process may wait for I/O
- A process typically alternates between two states:
  - **CPU burst**: perform calculations on the CPU (state Running)
  - **I/O burst**: process waits for data transfer in or out of system (state Blocked)
- Scheduler can schedule another process during I/O wait



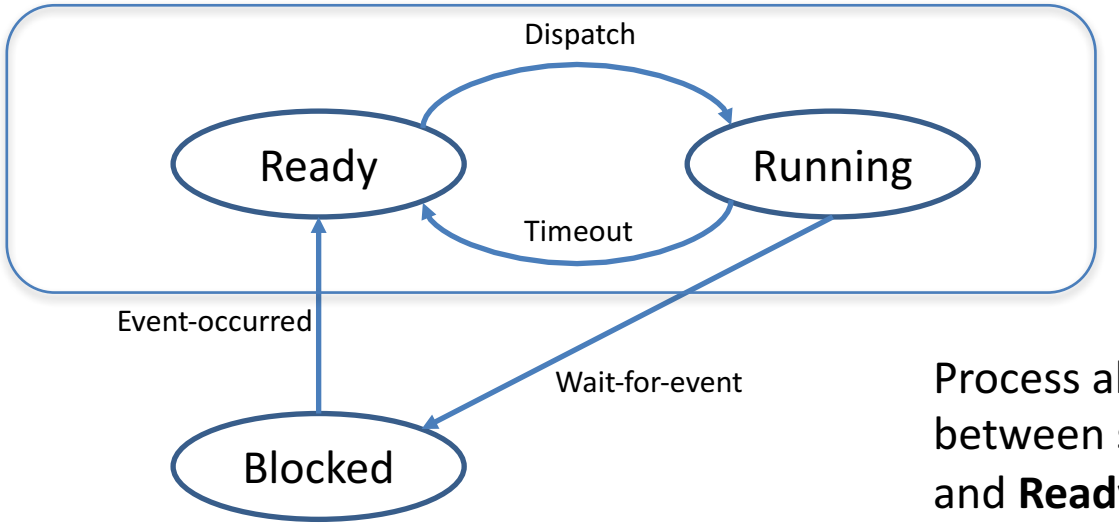
CPU-burst Times



- Histogram shows that most processes have many very short CPU bursts (high frequency of short bursts)
  - Interactive systems: very short CPU bursts
  - Scientific calculations: very long CPU bursts

Preemptive Scheduling

Process may be interrupted during its CPU burst

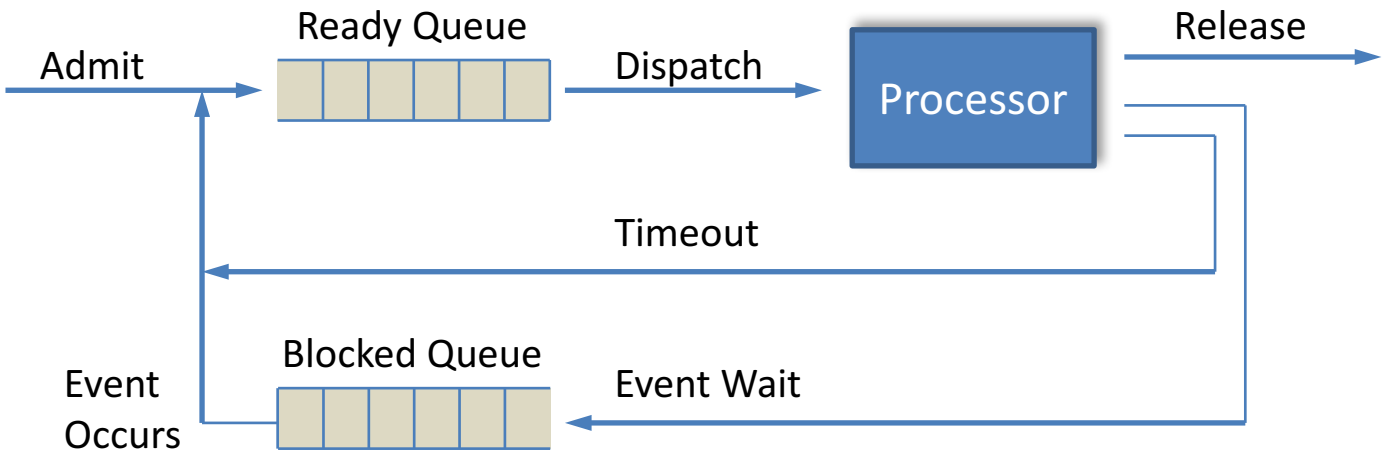


Process alternates between states **Running** and **Ready**

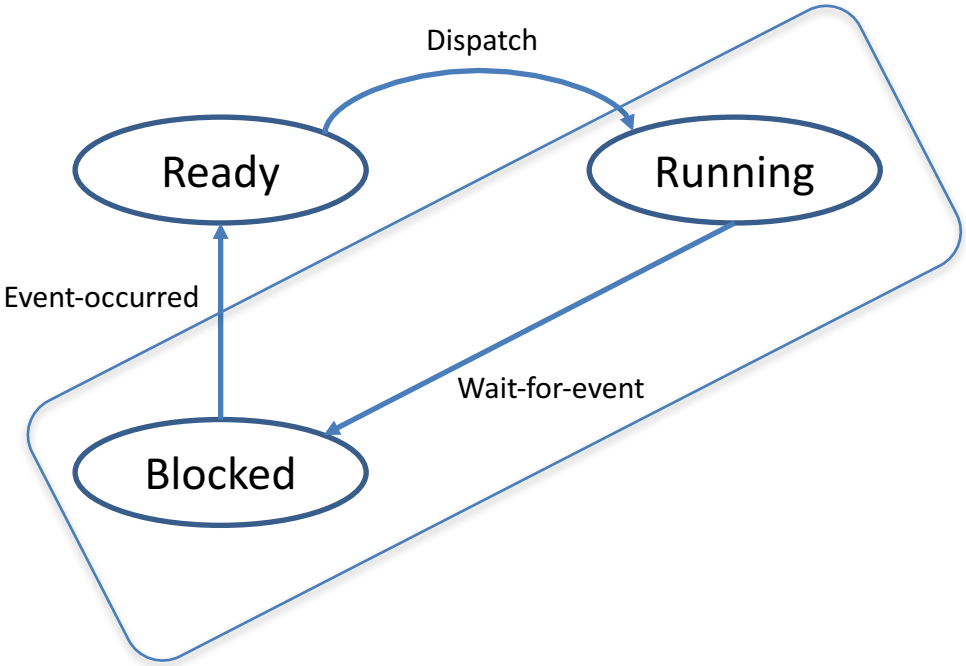
- Preemptive scheduling
  - A process in state **Running** can be interrupted by OS to give another process CPU time
  - Preempted process is transitioned directly into state **Ready**
  - CPU burst times are much shorter, processes much more interleaved in their execution

Preemptive Scheduling

- The operating system interrupts processes
  - A scheduled process executes, until its time slice (called a **time quantum**) is used up
  - Clock interrupt returns control of CPU back to scheduler at end of time slice
    - Current process is suspended and placed in the Ready queue
    - New process is selected from Ready queue and executed on CPU
  - When a process with higher priority becomes ready
- Used by most modern operating systems



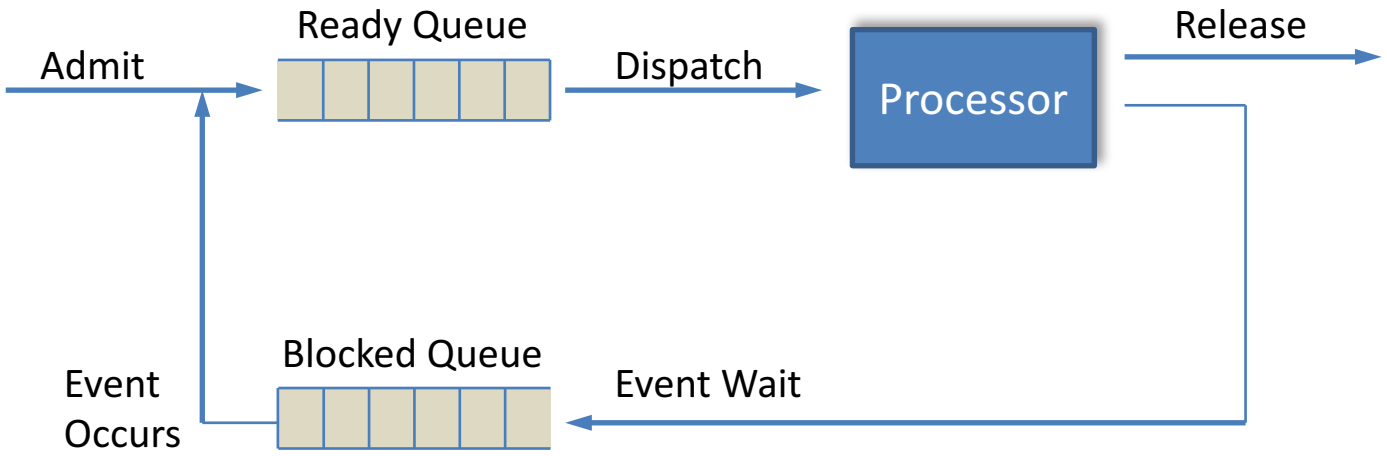
Non-Preemptive Scheduling



- Non-preemptive scheduling:
  - A process in state **Running** completes its CPU burst, until it transitions into state **Blocked** to wait for I/O, Running state cannot be interrupted by OS
  - If the process is in state Blocked, another process can be transferred into state Running (and will complete its CPU burst)

Non-premptive Scheduling

- Once a process is scheduled, it continues to execute on the CPU, until
  - it is finished (terminates)
  - It releases the CPU voluntarily (cooperative scheduling behaviour), or is interrupted by clock
  - It blocks due to an event:
    - I/O interrupts, waits for another process
- OS cannot interrupt process execution



Scheduling Goals

- Scheduling is important
  - To improve the performance of single processes (process perspective)
    - As few interruptions as possible, as much processing time as possible, large time slices
  - To improve the responsiveness of the overall system (user perspective)
    - Concurrent execution of as many processes as possible, small time slices
- A selection of goals
  - Improve throughput: process as many processes as possible
  - Be fair: no process should wait forever to be processed
  - Improve responsiveness of a system: no user should experience waiting times
  - Meet deadlines: real-time scheduling abilities of OS
  - Etc.

Scheduling Performance Criteria User Concerns

- Turnaround time
  - Total amount of time to execute one process to its completion
- Waiting time
  - Total amount of time a process has been waiting in the Ready queue
- Response time (Interactive systems)
  - The time it takes from when a request was submitted until the first response is produced by the operating system (latency), important for interactive systems
- Meeting Deadlines (Real-time systems)
  - An action has to occur until a specified deadline, important for real-time systems
- Predictability
  - The execution time of a process should be independent of the actual system load

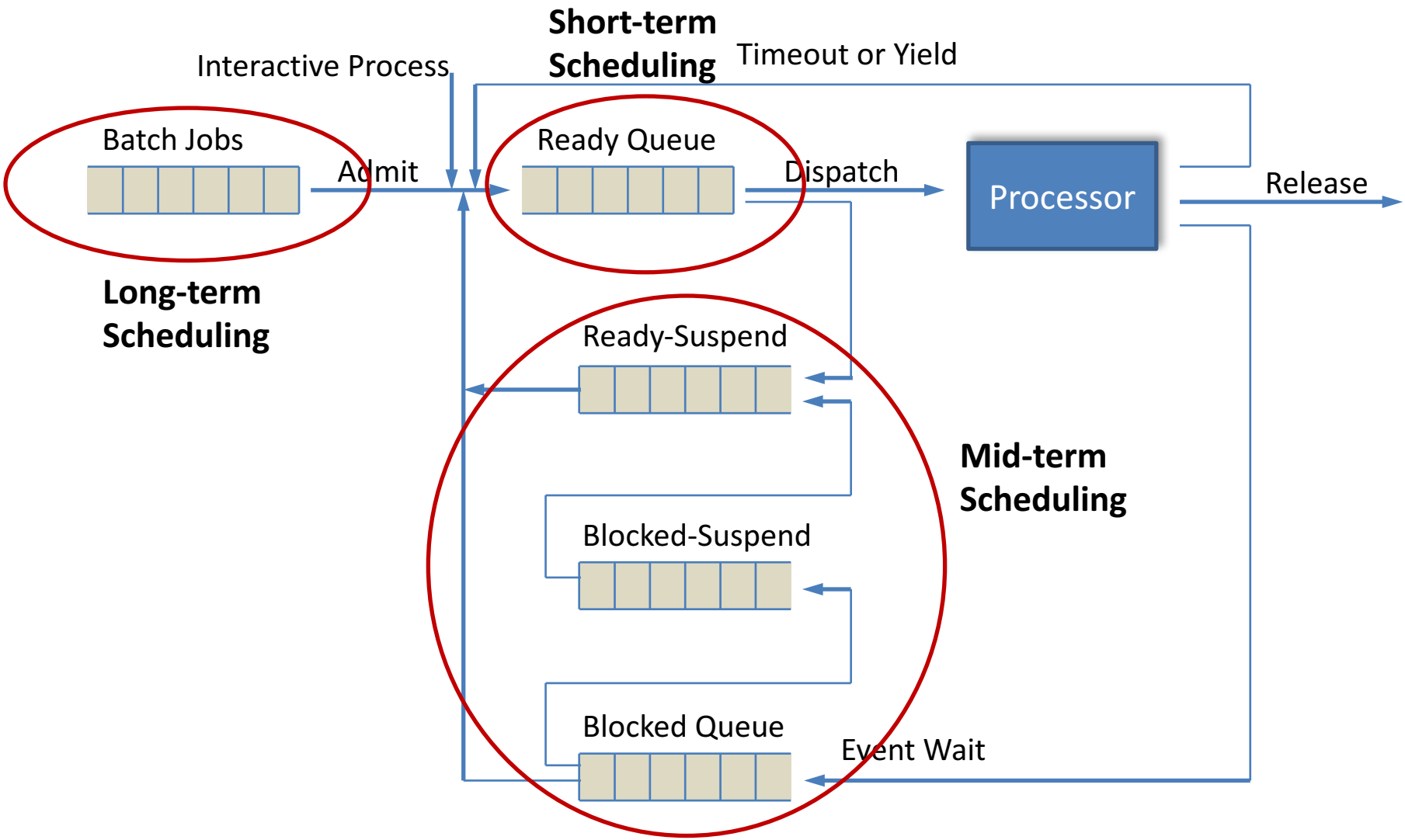
Scheduling Performance Criteria System / Process Concerns

- Processor utilisation
  - Keep the CPU as busy as possible
- Throughput
  - Number of processes that complete their execution per time unit
- Fairness
  - No process should suffer starvation, all processes to be treated equally
- Enforcing Priorities
  - When processes are assigned priorities, then scheduling should prefer higher-priority processes
- Balance
  - Balanced utilisation of resources

Optimization

- Maximize (operating system concerns)
  - Processor utilisation
  - Throughput
- Minimize (user concerns)
  - Turnaround time
  - Waiting time
  - Response time

Levels of Scheduling



Levels of Scheduling

- Long-term scheduling
  - Determines which programs are admitted to the system for processing and added to the Ready queue of the short-term scheduler (batch processing)
- Medium-term scheduling
  - Part of the swapping function, decides when to swap in or out processes (how many processes can be held in memory?)
- Short-term scheduling
  - Is invoked whenever an event occurs that blocks current process
    - Clock interrupt, I/O interrupt, system calls, signals

Scheduling Strategies

- Selection Function of Scheduler
  - Non-priority
    - Sequence of arrival (FCFS, RR)
  - Priority Scheduling
    - Fixed priority
    - Last CPU burst time
    - Accumulated processing time so far
    - Deadline
- Decision Mode
  - Pre-emptive: OS can interrupt process execution
  - Non-preemptive: OS cannot interrupt process execution

Scheduling Algorithms

Scheduling Algorithms

- Non-priority
  - First Come First Serve scheduling (FCFS)
  - Round-Robin Scheduling
- Priority scheduling
  - Shortest Job first scheduling
  - Multilevel Queue scheduling
  - Multilevel Feedback Queue scheduling

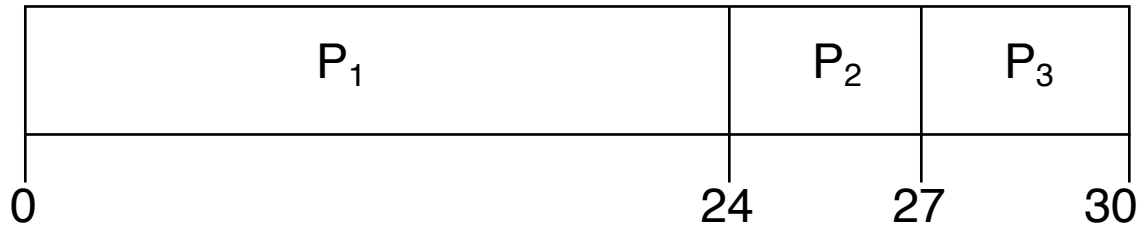
First Come First Serve Scheduling (FCFS)

- Non-preemptive scheduling policy:
- Processes are scheduled for execution according to their arrival sequence
  - No other considerations such as how long a process may need to finish its execution (Turnaround time)
- Other processes with shorter turnaround time may have to wait
  - Impacts on the responsiveness of a computer system
  - When a process becomes ready, it is put at the end of the queue – FIFO strategy

First Come First Serve Scheduling (FCFS)

- Example:
  - Three processes arrive at the same time (arrival time t = 0)
  - They have different processing times
- We assume an arrival order **P1, P2, P3**

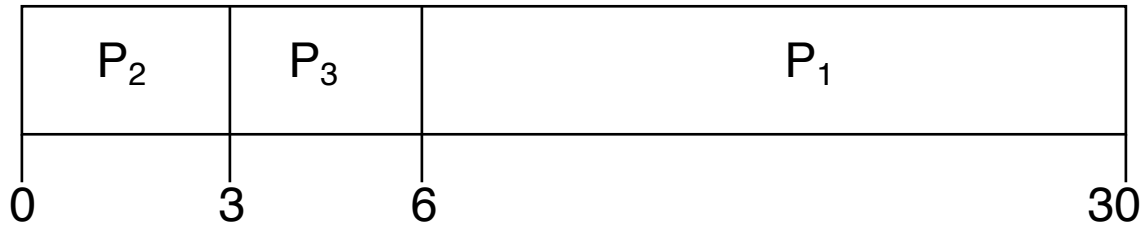
Process	Arrival	Processing Time
P1	**0**	**24**
P2	**0**	**3**
P3	**0**	**3**



- Waiting times
  - P1 has 0 time units waiting time
  - P2 has 24 time units waiting time
  - P3 has 27 time units waiting time
  - Average waiting time: (0 + 24 + 27)/3 = 17

First Come First Serve Scheduling (FCFS)

- We may get a better responsiveness with a different arrival order
- We assume the arrival order: **P2, P3, P1**
- The schedule of these three processes:



- Waiting times : P2 = 0, P3 = 3, P1 = 6
- Average waiting time: (0 + 3 + 6)/3 = 3

First Come First Serve Scheduling (FCFS)

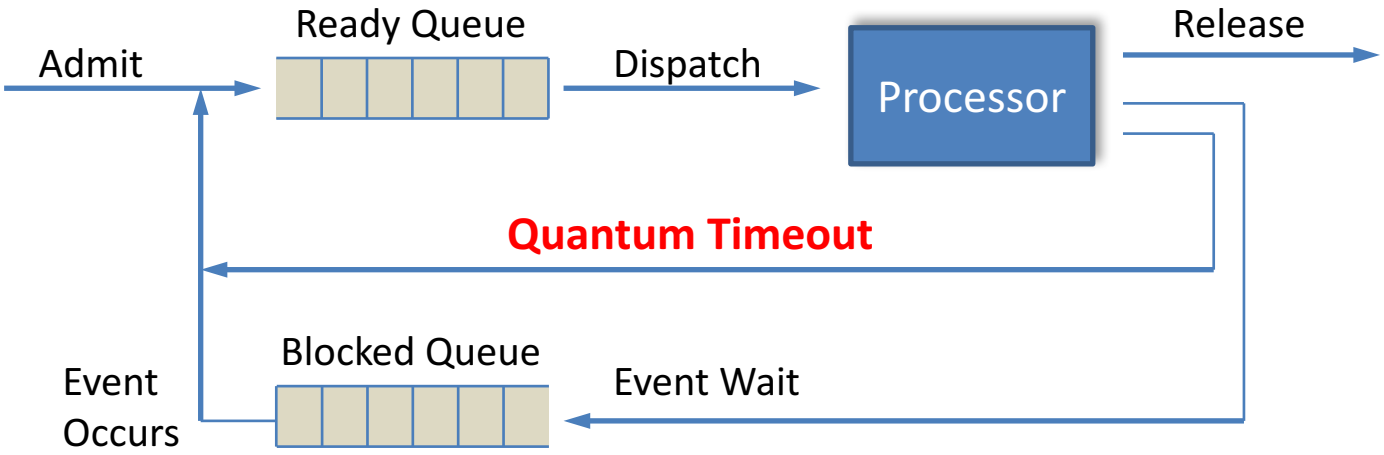
- Advantages
  - Favours long processes
  - Good for CPU-intensive processes (long burst times)
  - Is fair, no starvation
- Disadvantages
  - Leads to poor utilisation of CPU and I/O devices
  - Average waiting time is highly variable
    - Short jobs may wait behind large ones (convoy effect)

Round Robin (RR, Time Slicing)

- Is a preemptive scheduling policy, pre-emptive variant of FCFS
- Time quantum
  - Each process gets a fixed unit of CPU (usually 10-100 ms)
  - After time quantum has elapsed, the process is preempted and added to the end of the Ready queue
- Processes scheduled in a cyclical fashion
  - always same sequence of processes – round robin, FIFO strategy

Round Robin (RR, Time Slicing)

- Is a preemptive scheduling policy (pre-emptive variant of FCFS)
- Time quantum
  - Each process gets a fixed unit of CPU (usually 10-100 ms)



Round Robin (RR, Time Slicing)

- Simple to implement
  - Using FIFO strategy to take next process from Ready queue
- Fairness: Each process gets an equal time quantum
  - If the time quantum is q time units and if there are n processes in the Ready queue, each process get **1/n** of the CPU time, in time chunks with a size of at most **q** time units
  - No process waits more than **(n-1)q** time units
  - No starvation

Round Robin Scheduling

- Example
  - We assume:
    - N = 100 processes
    - Time quantum q = 10ms
  - Max waiting Time?
    - (n – 1)q = (100 – 1)10 = 990ms

Round Robin (RR, Time Slicing)

- Performance
  - Turnaround time varies with quantum size q
  - If q is large: round-robin becomes FCFS
    - Bad responsiveness, Good CPU utilisation, like FCFS
  - If q is small: fair, starvation-free, high responsiveness
    - Good responsiveness, Bad CPU utilisation
- Quantum should be large compared to context switch time, otherwise too much overhead

Round Robin (RR)

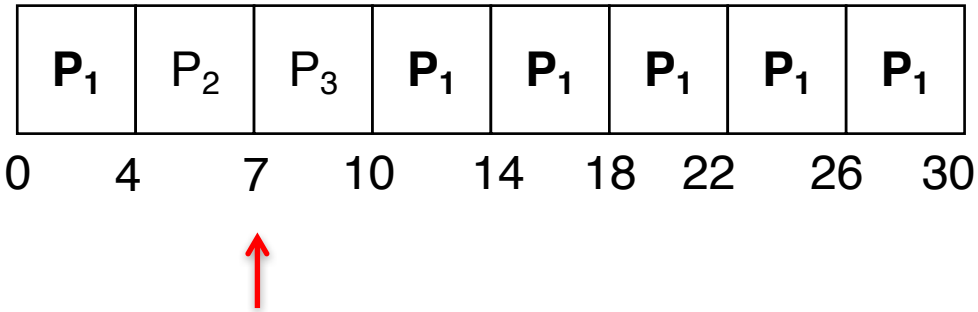
- Disadvantages:
  - CPU-intensive processes overtake I/O intensive processes
    - I/O interrupts a process executing its time quantum
    - I/O intensive processes may block early for I/O and may not use their full time quantum
  - CPU intensive processes usually use up their full time quantum, “get things done”, will overtake I/O intensive processes

Round Robin

- Example, time quantum q = 4

Process	CPU Time
P1	24
P2	3
P3	3

- Schedule



P2 only executes for 3 time units, doesn't use full time quantum

- RR has good response time behaviour, processes with short execution time are not disadvantaged

Priority Scheduling

Priority Scheduling

- Processes are scheduled according to a priority
- The CPU is allocated to the process with the highest priority
- There are pre-emptive and non-pre-emptive priority scheduling algorithms
- Policies
  - Shortest Job First (non-preemptive)
  - Shortest Remaining Time (pre-emptive)
  - Fixed Priority Scheduling (pre-emptive)
  - Multilevel Feedback Queue

Priority Scheduling

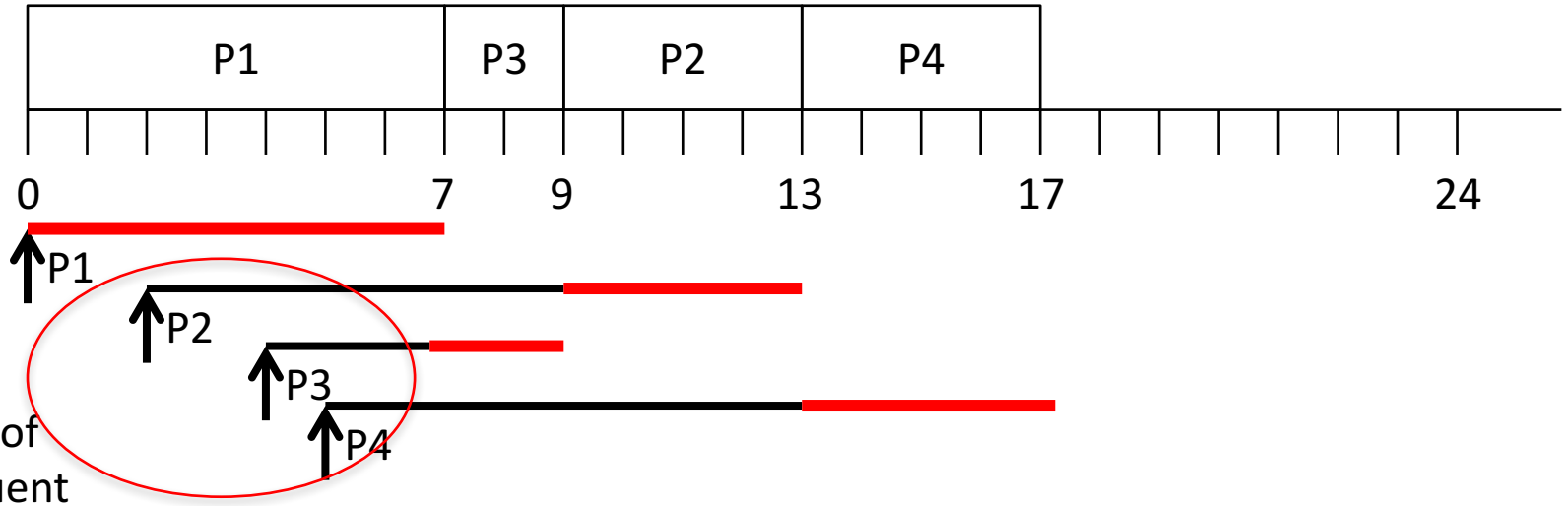
- Static Priorities
  - Priority specified at creation of a process
  - Used in real-time systems
- Dynamic Priorities
  - We analyse past performance to predict the future
  - Priority adapts to process behaviour (feedback scheduling)
  - E.g.: Shortest Job First SJF – priority determined by length of last CPU burst

Shortest Job First(SJF)

- Non-preemptive scheduling policy that tries to always schedule the process with the shortest expected processing time next
  - How do we know which process will have the shortest processing time?
  - Make it dependent on a predicted next CPU burst time
- Remember:
  - Non-preemptive scheduling:
    - A process in state **Running** **completes its CPU burst**, until it transitions into state **Blocked** to wait for I/O, **cannot be interrupted by OS**
    - If the process is in state **Blocked**, another process can be transferred into state Running (and will complete its CPU burst)

Shortest Job First(SJF)

Process	Arrival Time	Total CPU Time
P1	0	7
P2	2	4
P3	4	2
P4	5	4



Schedule sequence: [P1, P3, P2, P4]

Average Waiting Time: (0 + 7-4 + 9-2 + 13-5)/4 = = 18/4 = 4.5

Shortest Job First(SJF)

- Difficulty
  - How do we know which job has the shortest processing time?
- Necessity to estimate the required CPU processing time of each process
  - For batch jobs: past experience, specified manually
- Predict the length of the next CPU burst
  - For interactive jobs: analyse past bursts of CPU time, calculate exponential average