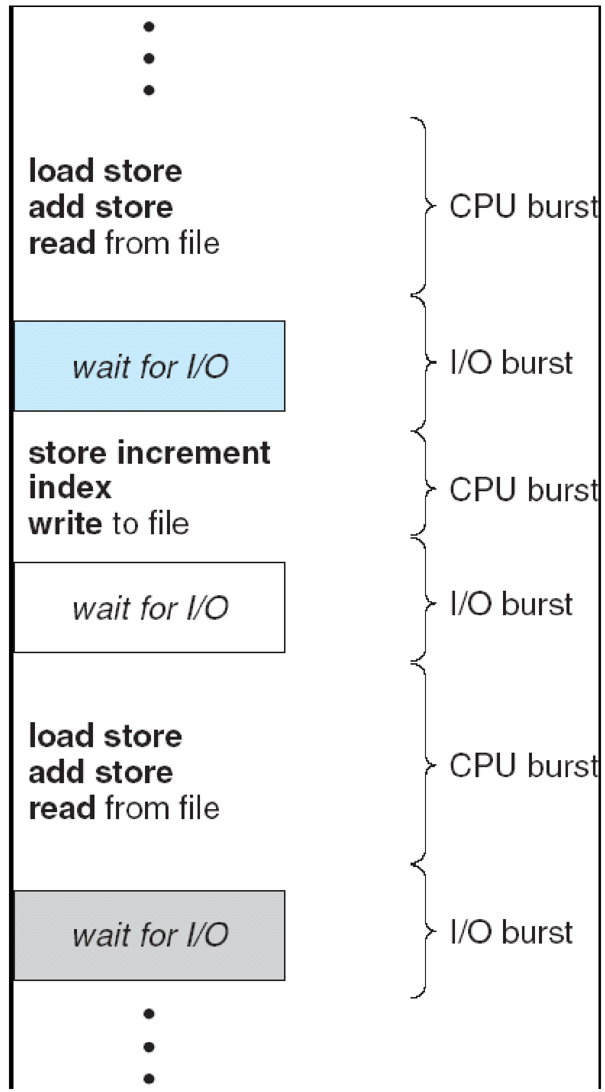
Part 3: Process Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
  – Uniprocessor (single-core) scheduling
  – Multiprocessor (multi-core) scheduling

Basic Concepts

- CPU–I/O Bursts: Process execution alternates between CPU executions and I/O operations
  - CPU burst: Duration of one CPU execution cycle
  - I/O burst: Duration of one I/O operation (wait time)
- CPU scheduling objective: Keep the CPU busy as much as possible (i.e., using multiprogramming)
- For simplicity, we focus on the concepts of **short-term (ready queue) scheduler for both uni- and multi- processor CPUs**

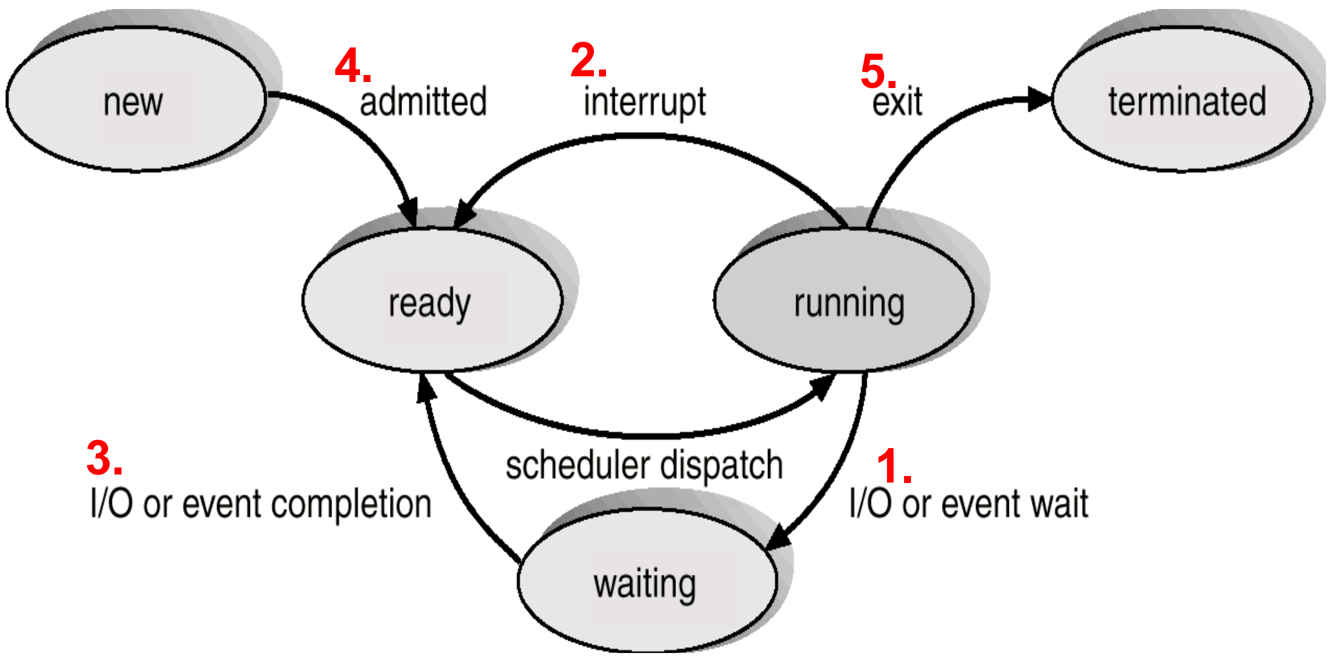
Alternating Sequence of CPU And I/O Bursts



CPU Scheduler (Short-Term)

- Goal: Select one or more of the processes in the ready queue, and allocate it to the CPU (one process per core)

- Scheduler may run whenever a process changes state



In which case(s) does CPU scheduling have to occur in order to keep the CPU cores busy?

Answer: Only 1 and 5

Types of CPU Schedulers

1. **Nonpreemptive** means that once the CPU has been allocated to a process, the process keeps the CPU until it voluntarily releases the CPU either by terminating or requesting I/O/event wait
   – Scheduling is nonpreemptive if it happens **only** for transitions 1 and 5 (also 4, when a CPU core is idle)

2. **Preemptive** means that the CPU can be taken away from a running process at any time by the scheduler
   – Scheduling is preemptive if it **also** happens for transitions 2, 3, 4 or at any other time instant

Scheduling Objectives

System-wide Objectives:

1. **Max. CPU Utilization:** Keep the CPU as busy as possible
   – Percentage of time during which the CPU cores are busy executing processes.
   – (Sum of execution time on each core) divided by (total time multiplied by number of cores)
2. **Max. Throughput:** Number of processes that complete their execution per unit of time
   – Number of "**exit**" **transitions** per unit of time

Scheduling Objectives (Cont.)

Individual Process Objectives:

1. **Min. Turnaround Time:** Amount of time to execute a particular process, i.e., from the time of creation to the time of termination
   - Time elapsed between **"admitted"** and **"exit" transitions** for a process
   - Average over all processes is denoted as average turnaround time

Scheduling Objectives (Cont.)

2. **Min. Waiting Time:** Amount of time a process has been waiting in the **"ready" state**
   - We do not consider time in "waiting" state; why?
   - Turnaround time components: CPU burst (running), I/O burst (waiting) and waiting time (ready)
   - If all processes have a single CPU burst (no I/O), then waiting time = turnaround time – CPU burst

3. **Min. Response Time:** Time from a request submission (**"admitted" transition**) until the first response is produced (we assume first response coincides with start of execution)

Scheduling Algorithms: Uniprocessor System

1. First-Come, First-Served (FCFS)
2. Shortest Job First (SJF)
3. Priority-based Scheduling
4. Round Robin (RR)
5. Multilevel Queue Scheduling

We assume a single CPU burst for each process, a single CPU core and focus on the average waiting time

First-Come, First-Served (FCFS) Scheduling

- Example:

Process	CPU Burst Length
P_1	24
P_2	3
P_3	3

- Suppose that processes arrive in the order P_1, P_2, P_3, all at time 0
- **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, again all at time 0
- The Gantt chart for the schedule is:

- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = \mathbf{3}$
  - Much better than the previous case

FCFS Scheduling: Properties

- Is FCFS preemptive or nonpreemptive?
  - Nonpreemptive, because processes have to voluntarily release the CPU once allocated

- Main problem with FCFS is the ***convoy effect***
  - Short processes suffer increased waiting times due to an earlier arrived long process
    - *We* saw this effect in the previous example - P1 increased the waiting times of P2 and P3

Shortest-Job-First (SJF) Scheduling

- Prioritize processes based on their CPU burst lengths
  - **Shorter burst implies higher priority**
  - Intuitive way to handle the convoy effect of FCFS

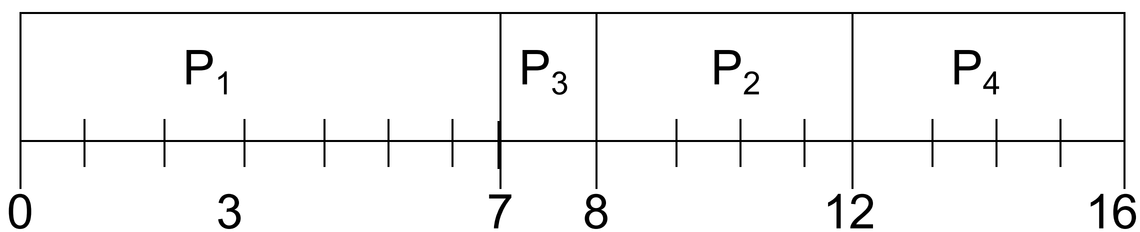
- Two schemes:
  1. Nonpreemptive: Once core is given to a process, it cannot be preempted until it completes its CPU burst
  2. Preemptive (also known as Shortest Remaining-Time First or SRTF): If a newly created process has CPU burst length less than the remaining CPU burst of currently running process, then preemption will occur

- SJF/SRTF is optimal for minimizing average waiting time

Example of Nonpreemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- **SJF (Nonpreemptive) Gantt Chart:**

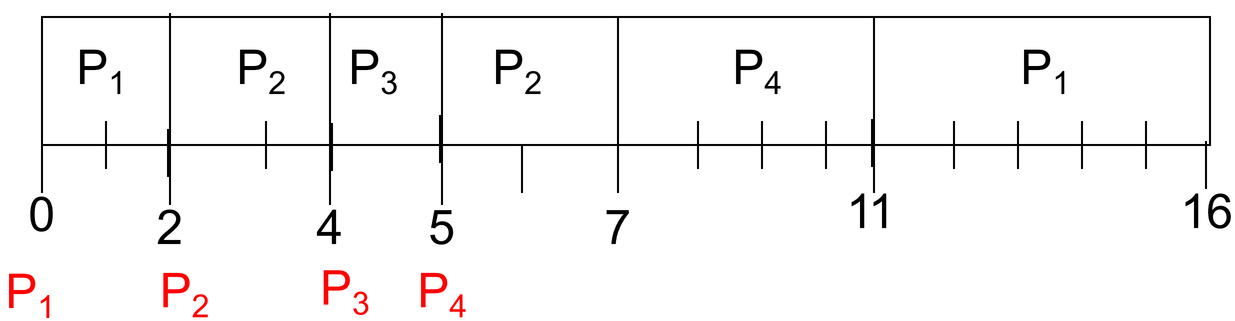


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Example of Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time
P_1	0.0	7 $\rightarrow$ 5 $\rightarrow$ 0
P_2	2.0	4 $\rightarrow$ 2 $\rightarrow$ 0
P_3	4.0	1 $\rightarrow$ 0
P_4	5.0	4 $\rightarrow$ 0

- **SJF (Preemptive) Gantt Chart:**



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

Priority-based Scheduling

- A priority number (integer) is associated with each process
- CPU is allocated to the process with the highest priority
  - Usually, smallest integer implies highest priority
  - Two schemes: Preemptive and nonpreemptive
  - FCFS: Priority based on arrival order
  - SJF: Priority based on CPU burst length
- Problem of Starvation: Lower priority processes may never execute in a heavily loaded system
  - Solution is Aging: As time progresses, slowly increase the priority of processes that are not able to execute

Round Robin (RR) Scheduling

- Fixed time quantum for scheduling (q): A Process is allocated CPU for q time units, preempted thereafter and inserted at the end of the ready queue
  - Processes are scheduled cyclically based on q
  - Usually q is small, around 10-100 milliseconds

- Performance:
  - *n processes in ready queue implies* waiting time is no more than (n-1)q time units
  - **Large q:** Degenerates to FCFS
  - **Small q:** *Many context switches; high overhead*

Example: RR with Time Quantum = 20

Process	Burst Time
P_1	53 → 33
P_2	17 → 0
P_3	68 → 48
P_4	24 → 4

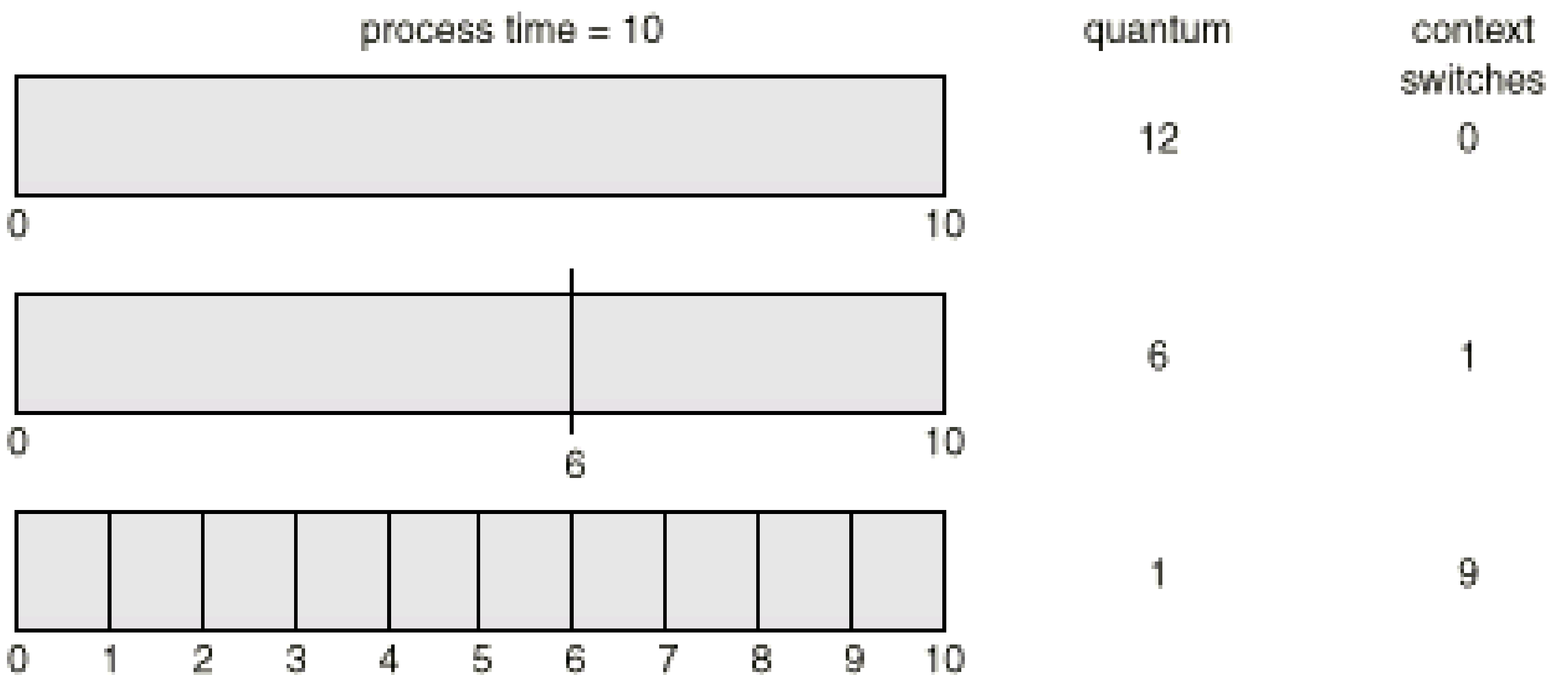
- All processes come at time 0 in the order of P1, P2, P3, P4
- **RR scheduling Gantt Chart:**

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

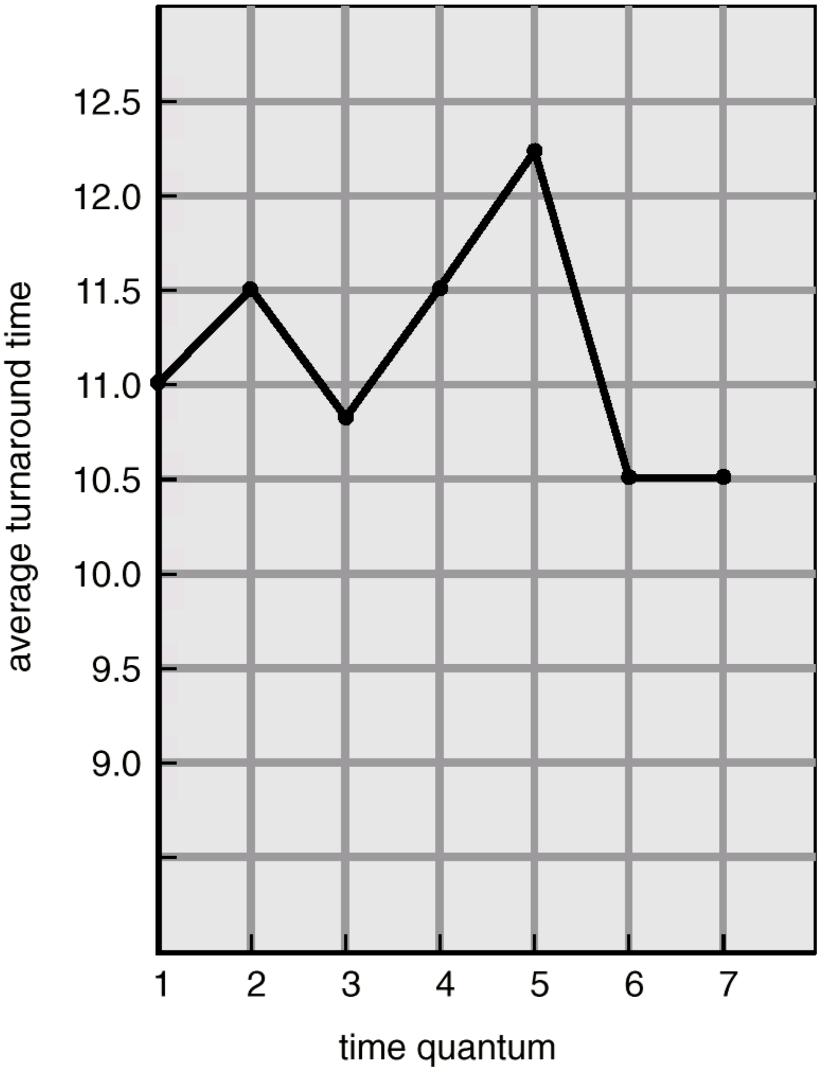
0 20 37 57 77 97 117 121 134 154 162

- Typically, higher average waiting/turnaround time than SJF, but better *response time*

Smaller Time Quantum Increases Context Switches



Turnaround Time Varies With Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Assume arrival order is P1, P2, P3, P4 all at time 0

- No well-defined relationship between average turnaround time and quantum size
- Average turnaround time remains fixed once quantum size is larger than maximum CPU burst length

Multilevel Queue Scheduling

- Different processes have different requirements
  - Foreground processes like those handling I/O need to be interactive (RR is preferred)
  - Background processes need NOT be interactive; scheduling overhead can be low (FCFS)

- Solution: Multi-level Queue Scheduling
  - Ready queue is partitioned into several queues
  - Each queue has its own scheduling algorithm
  - **How to schedule among the queues?**

Multilevel Queue Scheduling (Cont.)

- Two schemes for inter-queue scheduling
  - Fixed priority scheduling
    - Queues served in priority order (e.g., foreground before background)
    - Starvation for lower priority queues
  - Time-slice based scheduling
    - Each queue gets a fixed time quantum on the CPU (e.g., 80ms for foreground, 20ms for background, and then repeat)

Scheduling Algorithms: Multiprocessor System

1. Partitioned Scheduling (Asymmetric Multiprocessing - AMP)
2. Global Scheduling (Symmetric Multiprocessing - SMP)

We assume a multi-core CPU and that each process may only execute on one CPU core at any time instant

Partitioned Scheduling (AMP)

- Processes are partitioned apriori (at process creation time) among CPU cores
  - Each process is mapped to one core
  - Separate uniprocessor CPU scheduling for each core (**asymmetric scheduling**)
- Advantages
  - Core-specific scheduling strategy is feasible (FCFS, SJF, RR, multi-level, etc.)
  - Easy and simple extension of uniprocessor CPU scheduling to multiprocessor CPU scheduling

Partitioned Scheduling: Issues

- How to map cores to processes?
  - Poor mapping can lead to unequal loads: a core is idling while another is overloaded
    - May reduce system/process performance
    - **NP-Hard problem**: Similar to the well-known knapsack problem!
    - Heuristics such as best-fit could be used if CPU burst lengths are known

Global Scheduling (SMP)

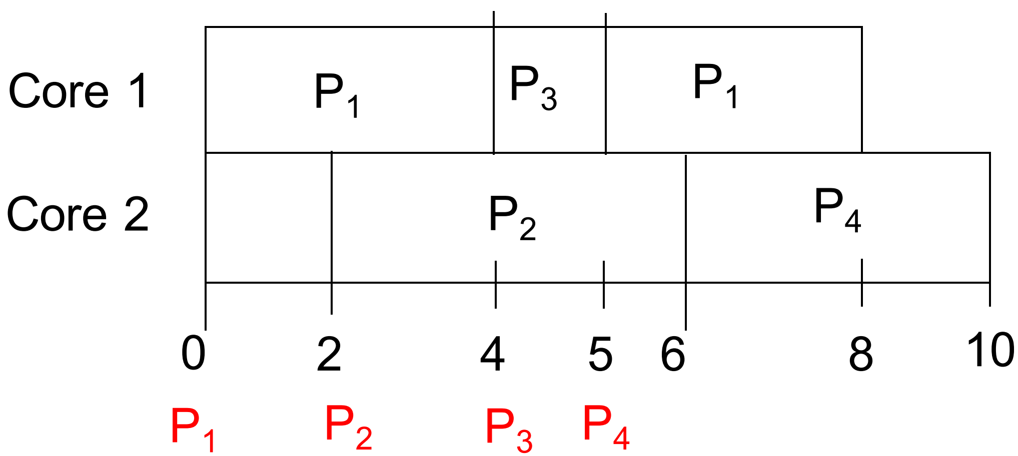
- One or more ready queues for the entire system (no mapping of queues to CPU cores)
  - When any core becomes idle, CPU scheduler runs and allocates the next process to execute
  - Process selection based on strategy applied globally or **symmetrically** across all cores (FCFS, SJF, etc.)
  - It is possible that the same process may execute on different cores at different times

No core-process mapping problem!

Example of SRTF (dual-core)

Process	Arrival Time	Burst Time
P_1	0.0	7 $\rightarrow$ 3 $\rightarrow$ 0
P_2	2.0	4 $\rightarrow$ 0
P_3	4.0	1 $\rightarrow$ 0
P_4	5.0	4 $\rightarrow$ 0

- **SJF (Preemptive) on dual-core Gantt Chart:**



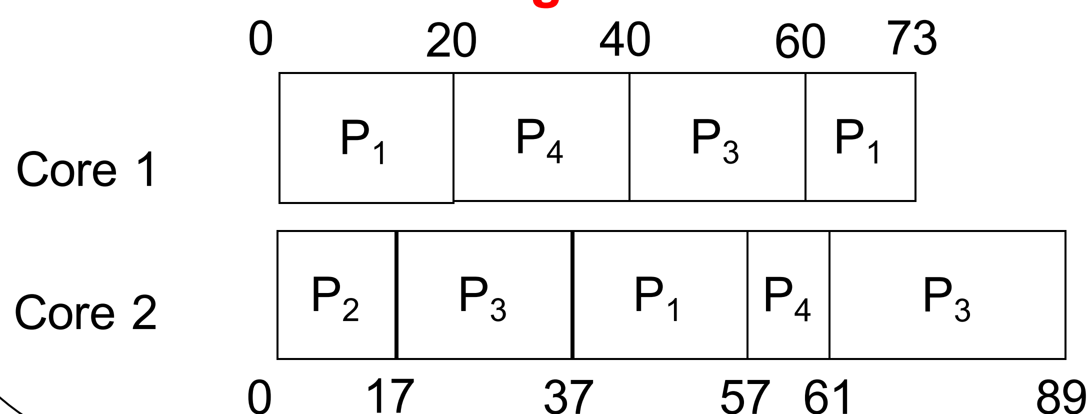
- Average waiting time = (1 + 0 + 0 + 1)/4 = 0.5

Example of RR with q = 20 (dual-core)

Process	Burst Time
P_1	53 → 33 → 13 → 0
P_2	17 → 0
P_3	68 → 48 → 28 → 0
P_4	24 → 4 → 0

- All processes come at time 0 in the order of P1, P2, P3, P4

- **RR scheduling dual-core Gantt Chart:**



Global Scheduling: Issues

- Implementation **complexity** is high when compared to partitioned scheduling
  - Need to synchronize clocks across cores
  - Global strategy for process selection combining all the cores
- Implementation **overhead** is high when compared to partitioned scheduling
  - Process could context switch from one core and be scheduled on another (private core-specific cache data needs to be migrated)