Operating Systems (CS3000)

Lecture – 8

(Multilevel Scheduling)

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING, KANCHEEPURAM

Dr. Jaishree Mayank

Assistant Professor

Department of Computer Sc. and Engg.

Multilevel Queue

- Ready queue is partitioned into separate queues – Priority Class
  - Foreground (interactive) (Word processor, game application)
  - Background(batch) (System update)

- Process permanently in a given queue
- Each queue has its own scheduling algorithm

–foreground – RR

–background – FCFS

Scheduling must be done between Queues

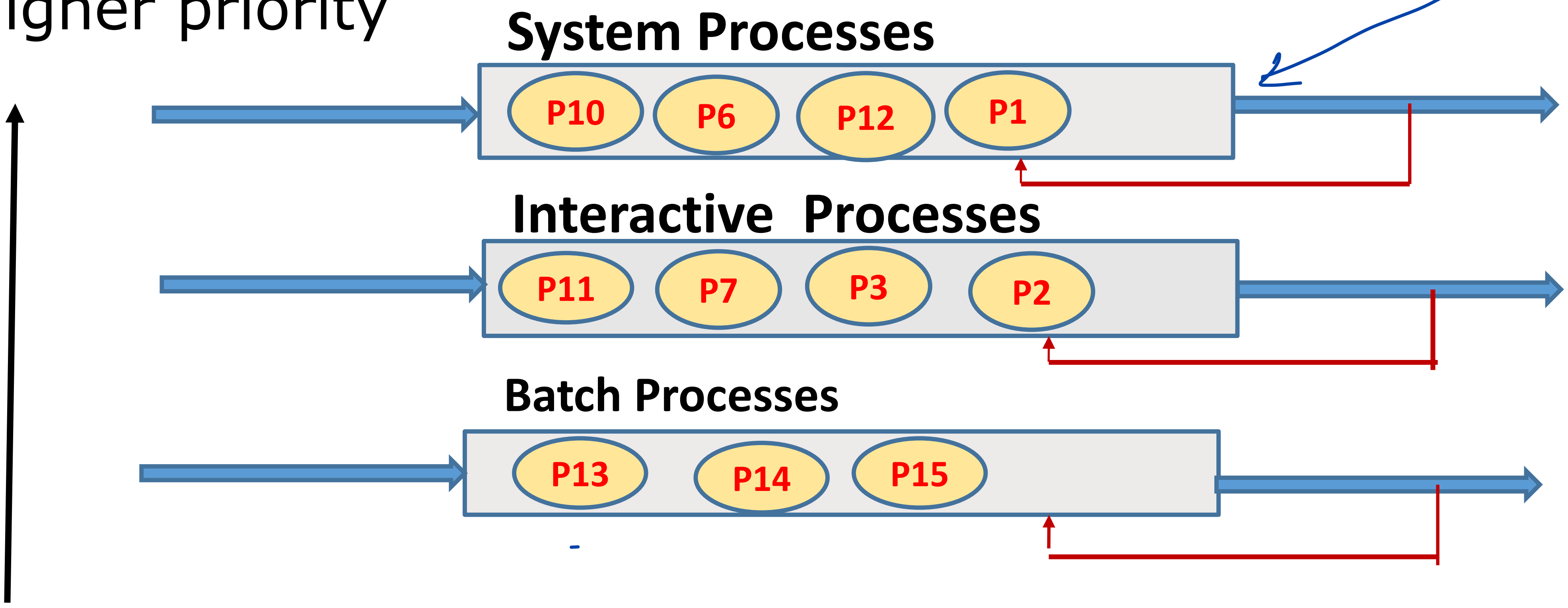
- **Strategy 1: Using Fixed Priority Scheduling**
  - serve all from foreground then from background
  - Issue:- Possibility of Starvation

- **Strategy 2: Using Time Slice**
  - each queue gets a certain amount of CPU time which it can schedule amongst its processes;
  - 80% to foreground in RR
  - 20% to background in FCFS

Multilevel Queue Scheduling

CPU

- Higher priority



preemptiv
fixed
priority
Schedulin

- Lower Priority

Multilevel Feedback Queue

- A process can move between the various queues processes
- Does not demand a knowledge of running time of jobs
- Balances both turnaround time (optimized by SJF/PSJF by selecting the smallest job) and response time (optimized by RR by alternating)
- Multilevel feedback queue defined by the following parameters
  - Numbers of queues
  - Scheduling algorithm for each queue
  - Methods used to determine when to upgrade a process
  - Methods used to determine when to demote a process
  - Methods used to determine which queue a process will enter when a process needs services

Multilevel Feedback Queue

- Rule 1: If Priority(A) > Priority(B), A runs (B doesn't)
- Rule 2: If Priority(A) = Priority(B), A & B run in RR
- Rule 3: When a job enters the system, it is placed at the highest priority (the topmost queue)
- Rule 4a: If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue)

–Intuition – CPU-bound job

- Rule 4b: If a job gives up the CPU before the time slice is up, it stays at the same priority level

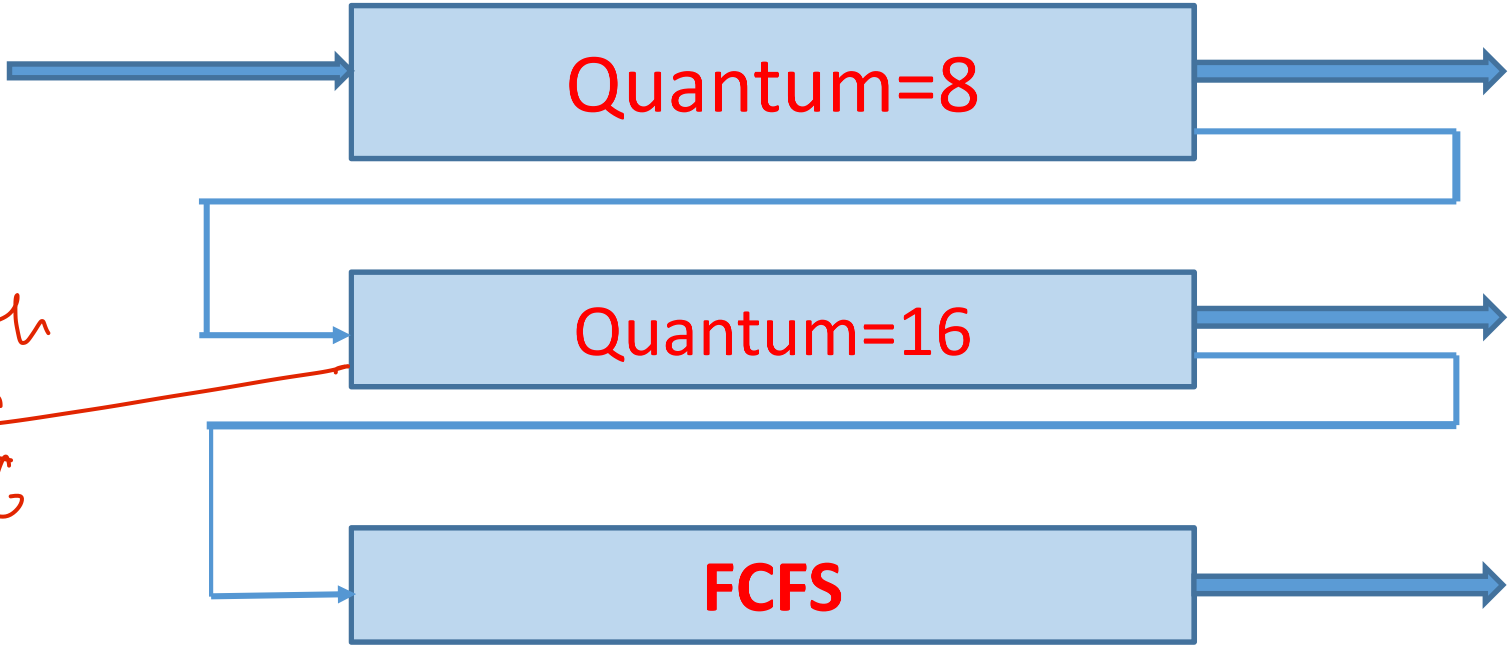
–Intuition – I/O-bound (interactive job)

Multilevel Feedback Queue

- Example: Consider 3 queues
  - Q_0 - time quantum 8 milliseconds
  - Q_1 –time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling:
  - A new job enters queue Q_0 which is served *for 8 ms*.
  - When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served RR and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.
  - At Q2 it executes as FCFS

Multilevel Feedback Queue

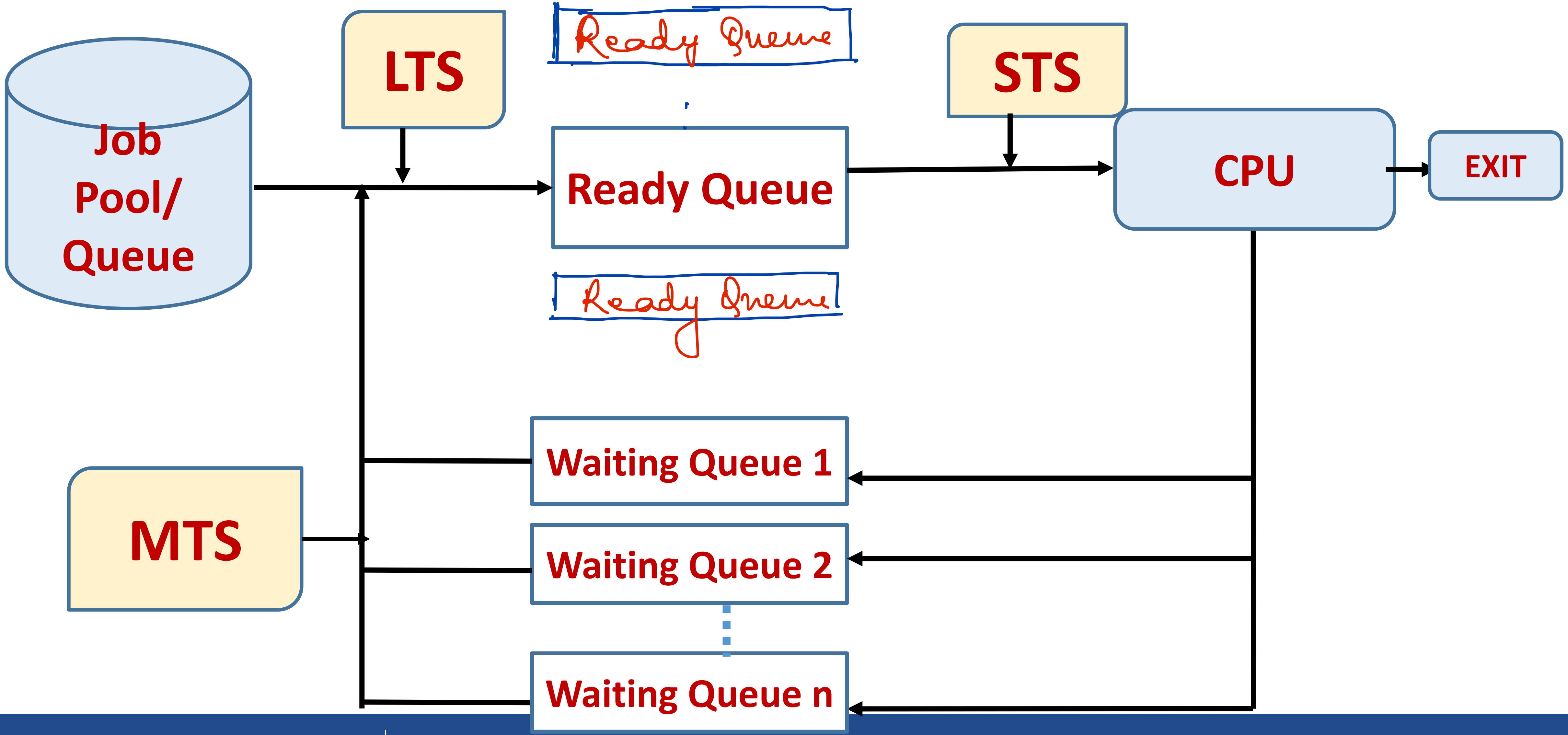
RR with quantum = 8



RR with quantum 16

among Queues preemptive fixed priority scheduling.

Queues during process execution



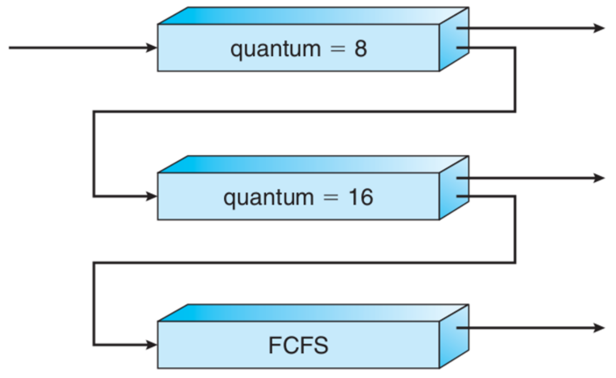
Multilevel Feedback Queue

- Rule 1: If Priority(A) > Priority(B), A runs (B doesn't)
- Rule 2: If Priority(A) = Priority(B), A & B run in RR
- Rule 3: When a job enters the system, it is placed at the highest priority (the topmost queue)
- Rule 4a: If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue)
- Rule 4b: If a job gives up the CPU before the time slice is up, it stays at the same priority level
- Issues

–Starvation – too many I/O jobs

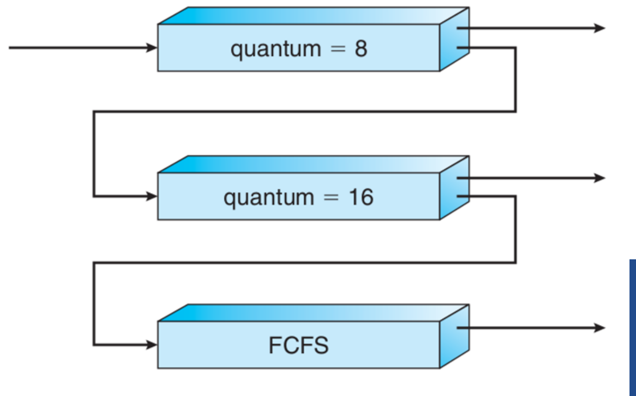
–Trick the scheduler – I/O just before the time slice is over

–A program may change its behavior over time



Multilevel Feedback Queue

- Rule 1: If Priority(A) > Priority(B), A runs (B doesn't)
- Rule 2: If Priority(A) = Priority(B), A & B run in RR
- Rule 3: When a job enters the system, it is placed at the highest priority (the topmost queue)
- Rule 4: Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue)
- Rule 5: After some time period S, move all the jobs in the system to the topmost queue



Comparison b/w different scheduling approach

Algorithm	Strategy	Average waiting time	Preemption	Starvation	Performance
FCFS	Acc. To arrival time	Large	No	NO	Slow performance/convoy effect
SJF	lowest cpu burst time	Smaller than FCFS	No	Yes	Mini Avg Waiting time
SRTF	Same as SJF but preemption allowed	Smaller than FCFS	YES	Yes	Min avg waiting time
RR	Fixed time quatum(TQ)	Large as compared to SJF and Priority scheduling.	YES	No	Each process has given a fairly fixed time
Priority Preemptive	Acc. To priority time with	Smaller than FCFS	Yes	Yes	Well starvation performance, but

Comparison b/w different scheduling approach

Algorithm	Strategy	Complexity	Average waiting time	Preemption	Starvation	Performance
Priority Non-Preemptive	Acc. To priority time with monitoring incomimg priority tasks	This type is less complex than priority preemptive	Smaller than FCFS	No	Yes	Most beneficial with batch systems
HRR	Response Ratio	Complex than FCFS	Smaller than FCFS	No	No	Helps for process with longer waiting time
MLQ	According to the process that resides in the higher priority queue	Complex than priority scheduling	Smaller than FCFS,	Yes/No	Yes	Good performance but contain a starvation problem
MLFQ	Longer burst time process moves to lower priority queue	Complex	Smaller than all scheduling types in many cases.	Yes/No	No	Good performance, no starvation

Other Advanced Scheduling Techniques

- Multiprocessor Scheduling
  - Symmetric
  - Asymmetric
- Real-Time Scheduling
- Distributed System Scheduling

Multiple-Processor Systems

- Multiprocessor systems are increasingly commonplace
  - In desktop machines, laptops, and even mobile devices

- Multicore processor
  - Multiple CPU cores are packed onto a single chip

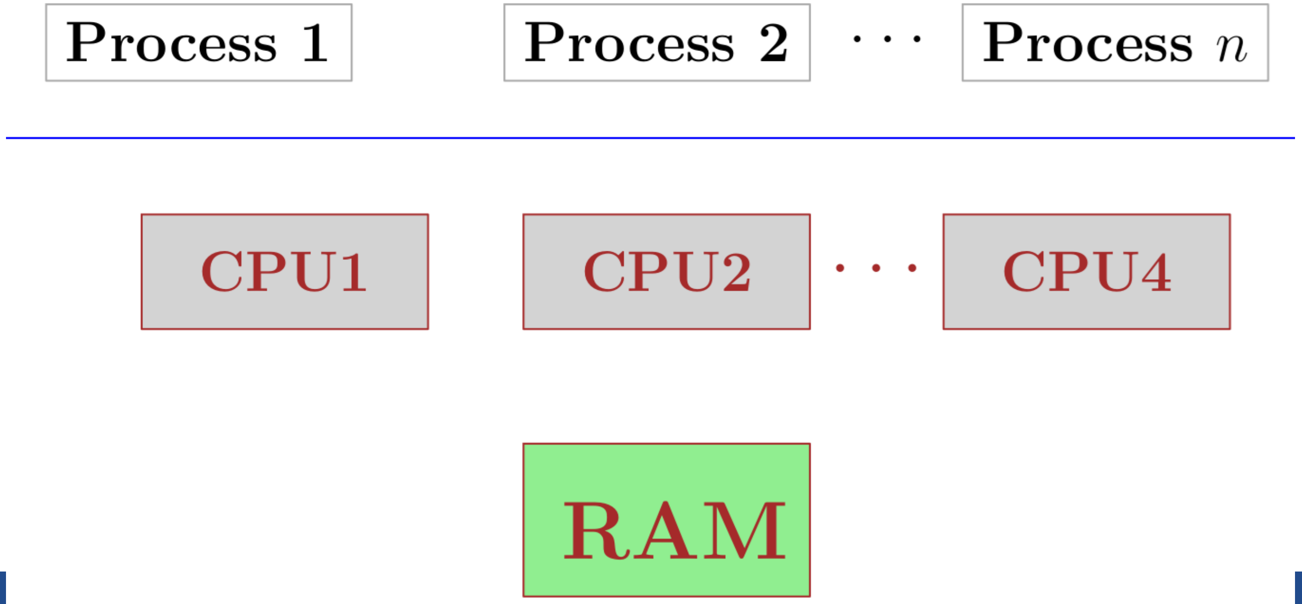
Multiple-Processor Scheduling - Challenges

- The same user program would not run faster
- CPU scheduling more complex when multiple CPUs are available – where + when

–Where to run scheduler?

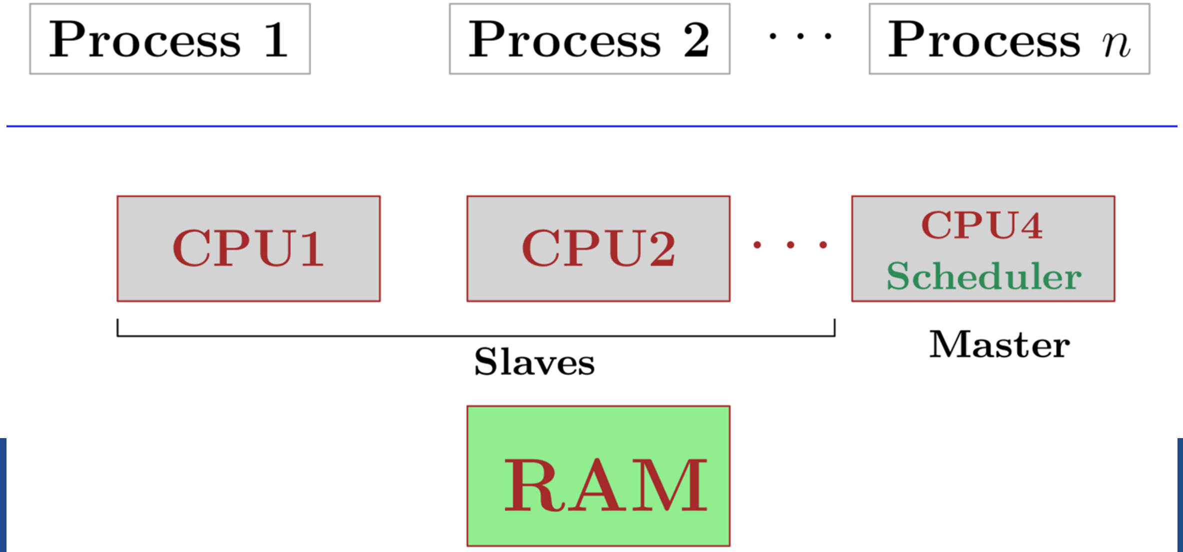
–Synchronization

- Shared data



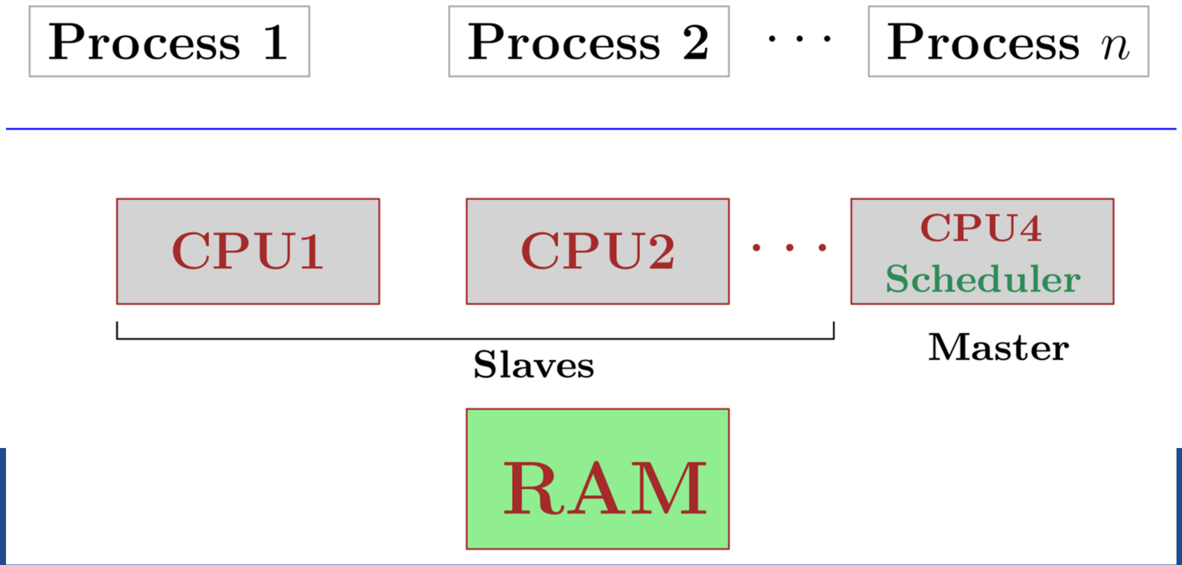
Asymmetric Multiprocessing (Master-Slave)

- All scheduling decisions, I/O processing, and other system activities handled by a single processor—the master
- The other processors execute the assigned program
- Only one processor accesses the system data structures, reducing the need for data sharing



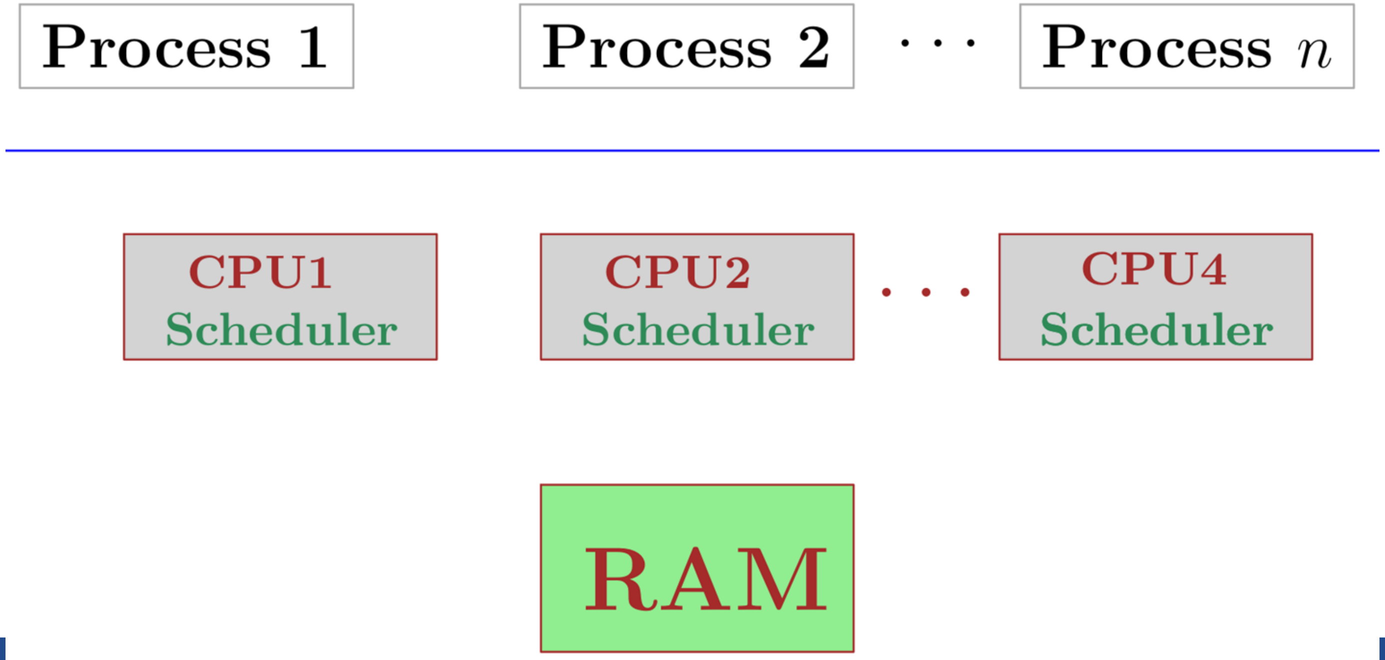
Asymmetric Multiprocessing - Issues

- Contention
  - –Slave processors waiting for Master to make Scheduling decisions



Symmetric Multiprocessing

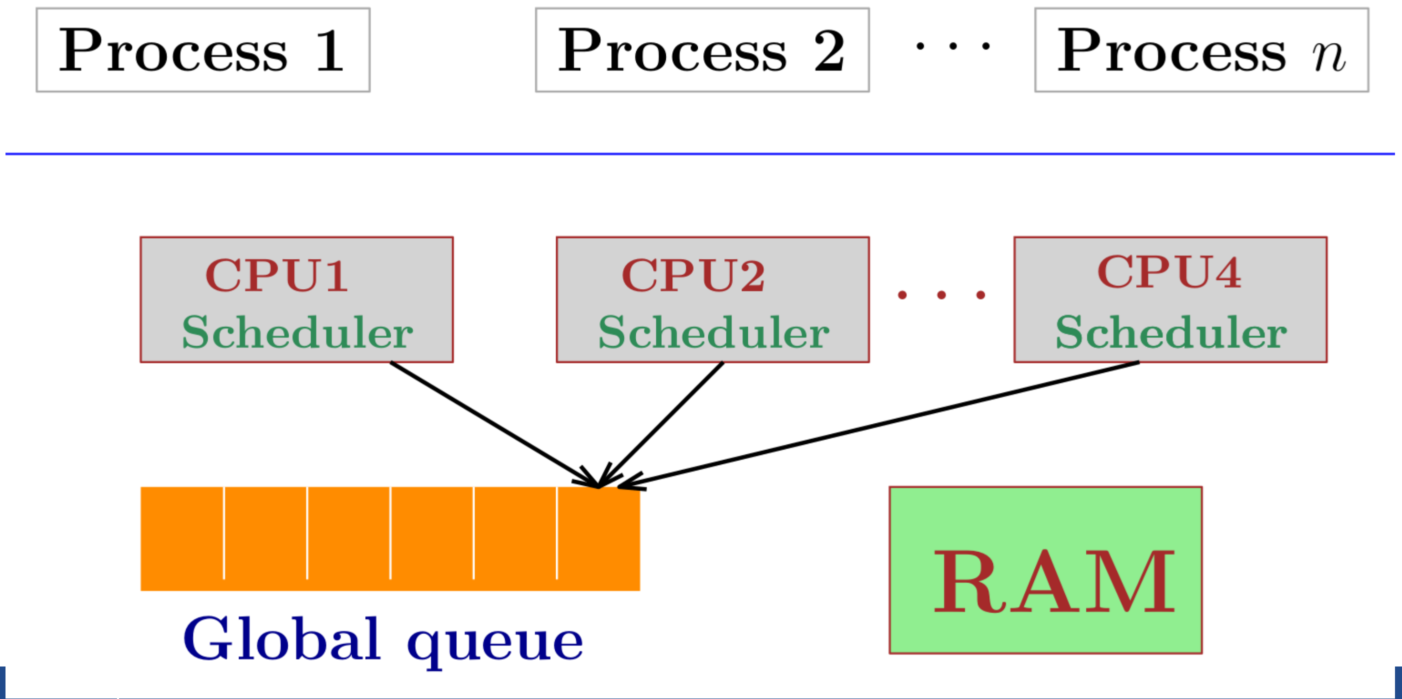
- Each processor runs its own scheduler



Symmetric Multiprocessing

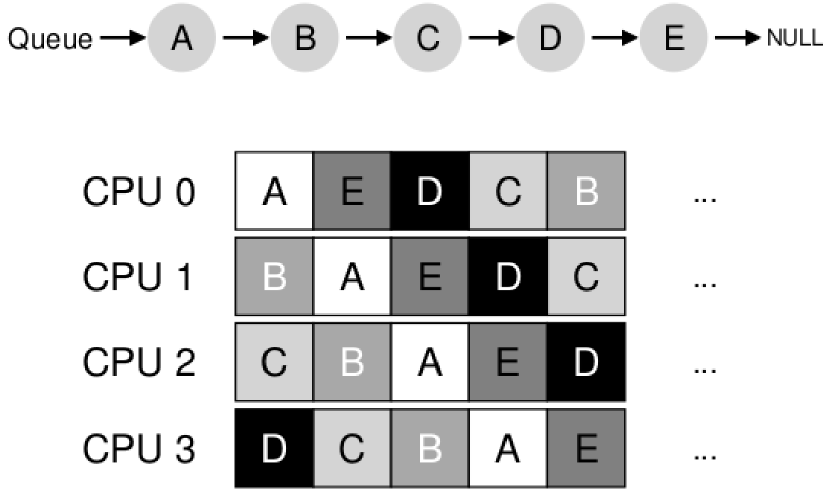
- Each processor runs its own scheduler

–**Global queue:** All processes in common ready queue



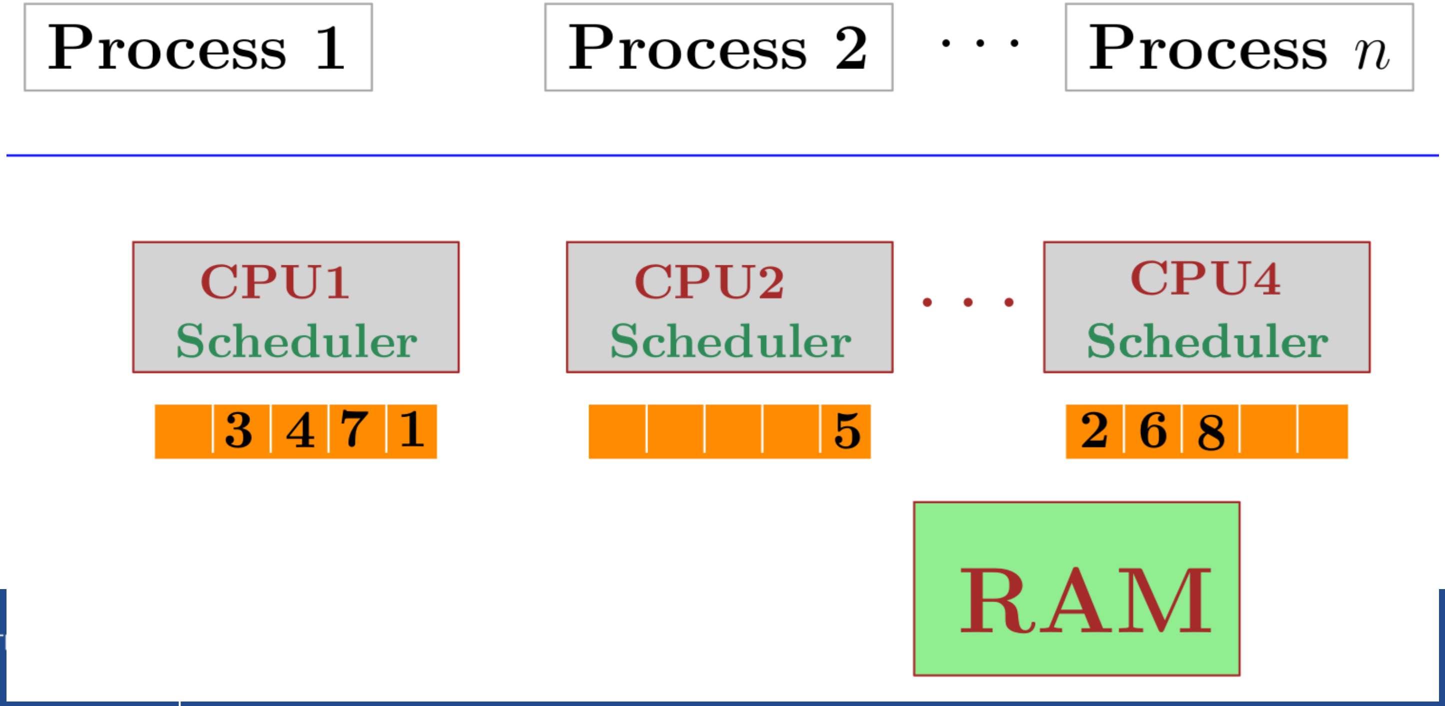
Symmetric Multiprocessing

- Each processor runs its own scheduler
  - **Global queue:** All processes in common ready queue
  - Advantages
    - Good CPU utilization
    - Fair to all processes
  - Issues
    - Scalability – contention for global queue
    - Synchronization – locking needed by scheduler
      - Scheduler should be as light as possible
    - Processor affinity not achieved



Symmetric Multiprocessing

- Each processor runs its own scheduler
  - **Partitioned queue:** per-processor a private queue
- Static partitioning of processes

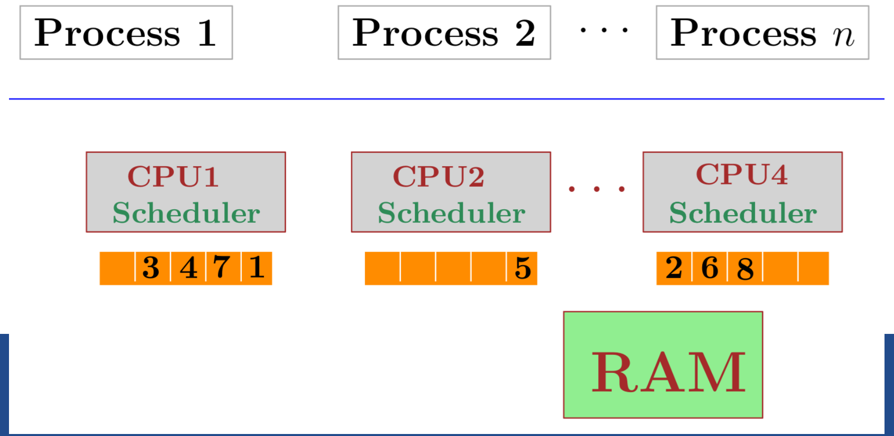


Symmetric Multiprocessing

- Each processor runs its own scheduler
  - **Partitioned queue:** per-processor a private queue
    - Static partitioning of processes
  - Advantages
    - Easy to implement – no need of synchronization
    - Scalable
    - Processor affinity implicitly guaranteed

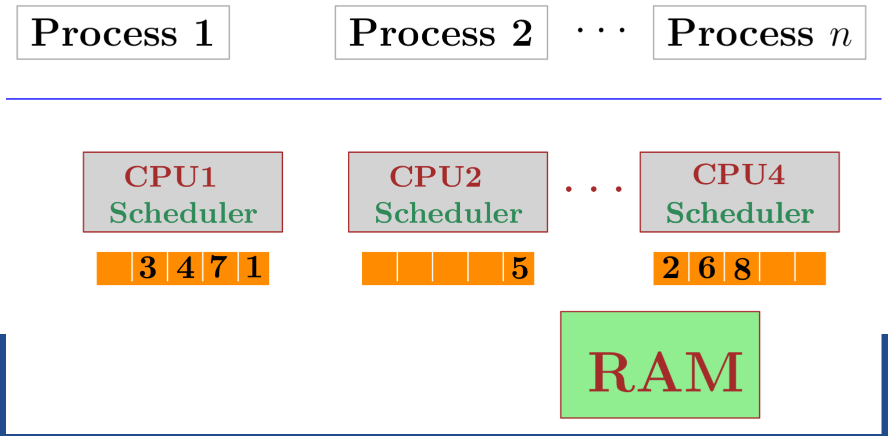
Assiging the process to a particular CPU only

  - Issues
    - Load imbalance



Symmetric Multiprocessing

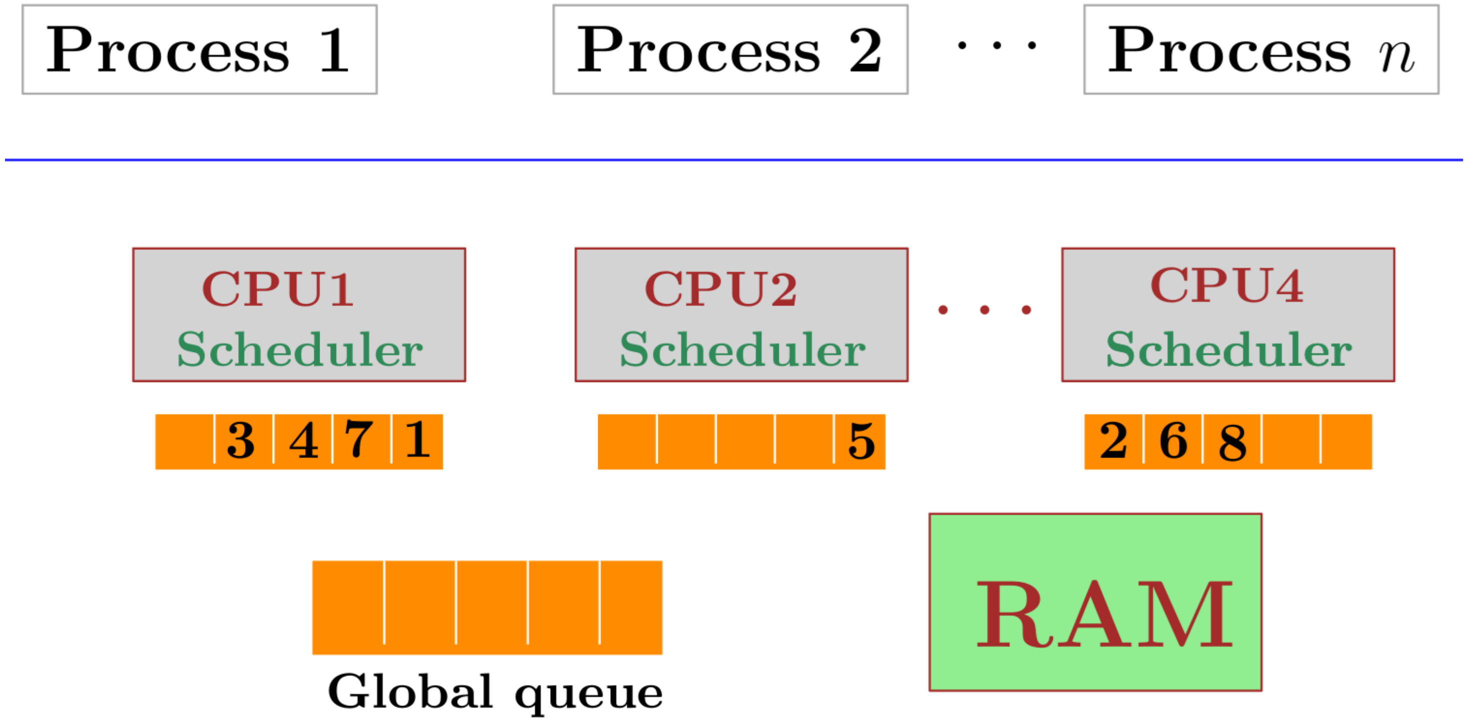
- Each processor runs its own scheduler
  - **Partitioned queue:** per-processor a private queue
    - Static partitioning of processes
  - Advantages
    - Easy to implement – no need of synchronization
    - Scalable
    - Processor affinity implicitly guaranteed
  - Issues
    - Load imbalance
    - **Solution??**



Symmetric Multiprocessing

- Each processor runs its own scheduler

–**Hybrid Approach:** global queue + partitioned queue



–Used in Linux kernel 2.6 onward

Real-Time Scheduling

- Real-time Systems

–Explicit timeliness requirement: deadline

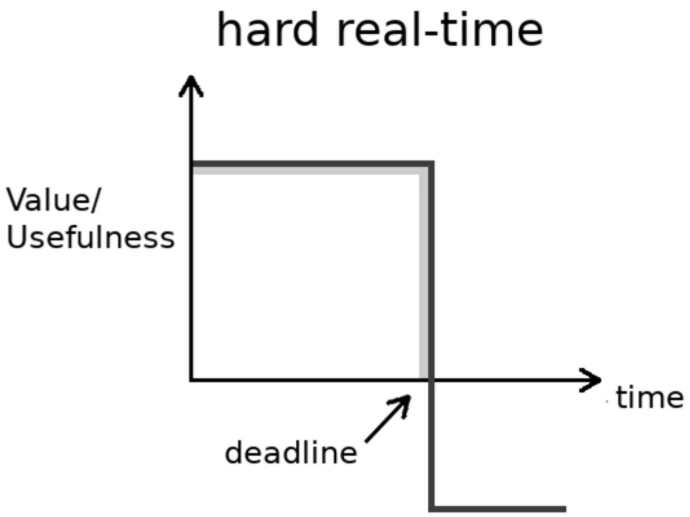
–The correctness of result depends on both functional correctness and time that the result is delivered

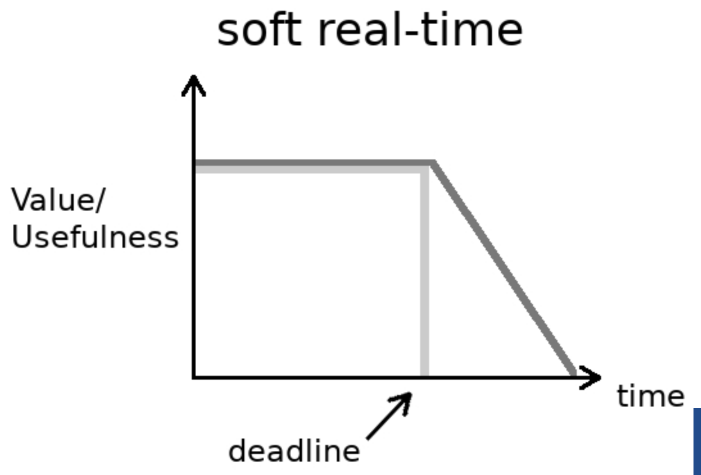
- Hard Real-time task

–Air traffic control, Vehicle subsystems control, Nuclear power plant control

- Soft Real-time Task

–Multimedia transmission and reception, Networking, telecom (cellular) networks, Web sites and services, Computer games





Real-Time Scheduling

- Periodic tasks

–Job

- Example: Speed sensor

$T_i = (e_i,p_i)$

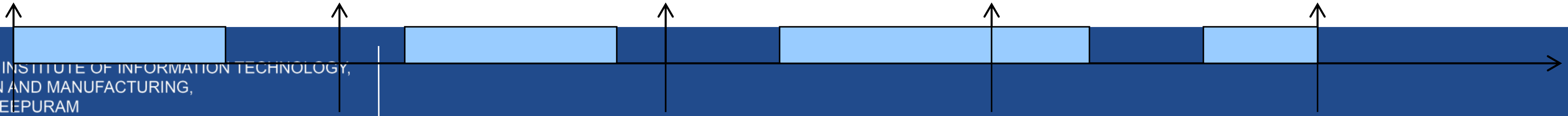
e_i = execution requirement and p_i = period

Utilization $u_i=e_i/p_i$

Necessary condition: $\Sigma u_i \leq 1$

* If not satisfied then there is definitely a deadline miss.

* If satisfied mean we may or may not get feasible schedule



Real-Time CPU Scheduling

- Rate Monotonic (RM)
- Earliest Deadline First (EDF)

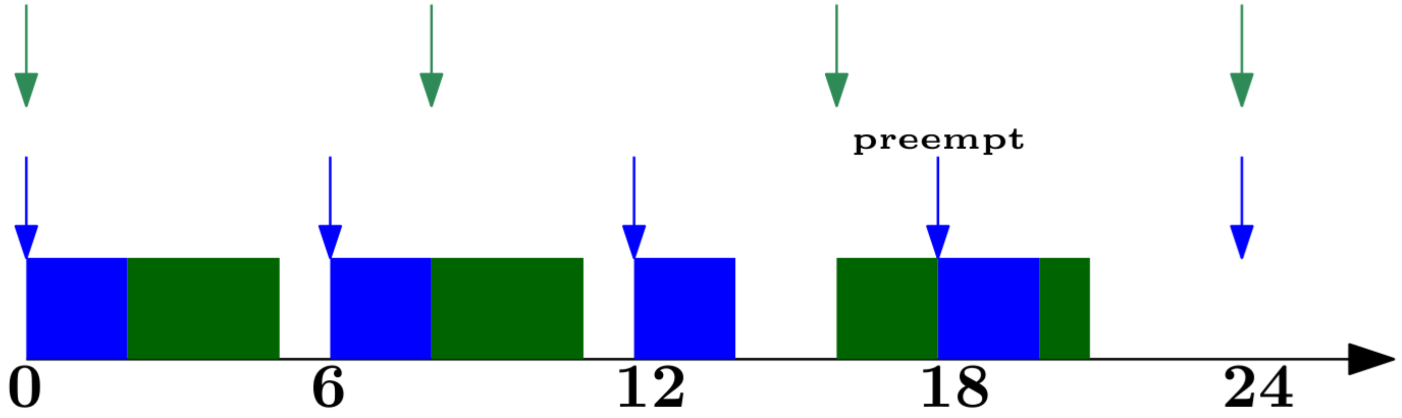
Rate Monotonic (RM) Scheduling

- Optimal Static Priority Real-time Task Scheduling Algorithm

→ Based on the time period (p)

Rate Monotonic (RM) Scheduling

- T1(2,6), T2(3,8)



- T1(2,6), T2(3,8), T3(3,12)

Rate Monotonic (RM) Scheduling

- Sufficient condition

If condition satisfy then only we can say tasks are schedulable without any misses.

$$\sum_{i=1}^{n} u_i \leq n(2^{\frac{1}{n}} - 1)$$

- For n → ∞, the righ… to $\log_e 2 = 0.692$

Earliest Deadline First (EDF) Scheduling

- Optimal Real-time Task Scheduling Algorithm (dynamic priority)

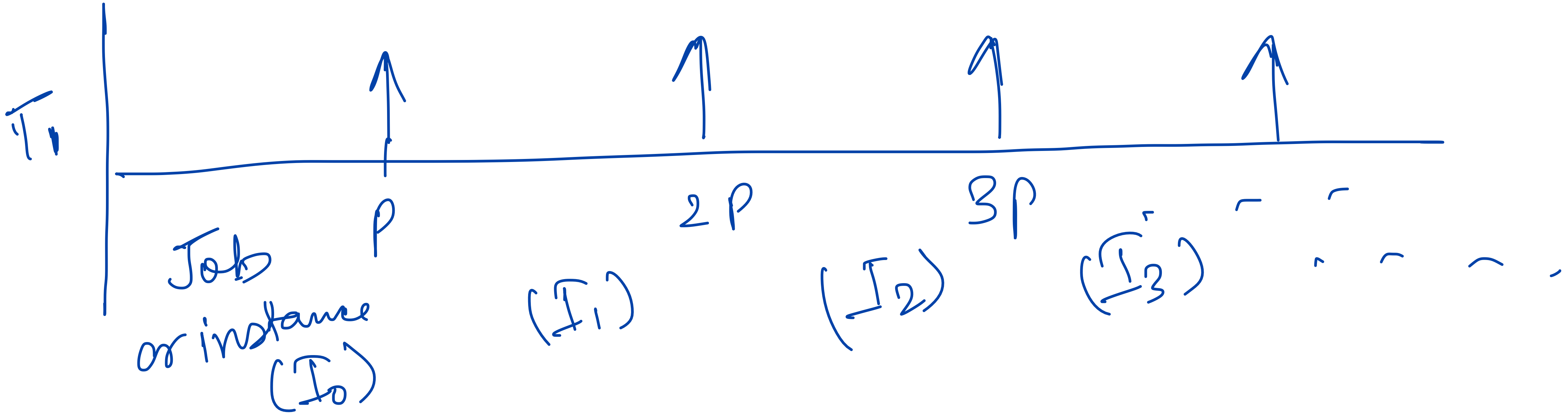
–Job-level fixed priority (Based on current deadline of job)

- At any scheduling point pick the job with the smallest deadline

Earliest Deadline First (EDF) Scheduling

- T1(2,6), T2(3,8)

- T1(2,6), T2(3,8), T3(3,12)



Earliest Deadline First (EDF) Scheduling

- Necessary and sufficient condition (preemptive version)

$$\sum_{i=1}^{n} u_i \leq 1$$

Thank You

Any Questions?