Operating Systems

Dr. Shu Yin

Part II: Process Management

- Processes
- Threads
- Process Synchronization
- CPU Scheduling
- Deadlocks

Goals

- Scheduling Intro
- CPU-Scheduling Algorithms
- Evaluation Criteria for Selecting Scheduling Algorithms

Scheduling Objectives

- Enforcement of fairness
- Enforcement of priorities
- Make best use of available system resources
- Give preference to processes holding key resources
- Give preference to processes exhibiting good behavior
- Degrade gracefully under heavy loads

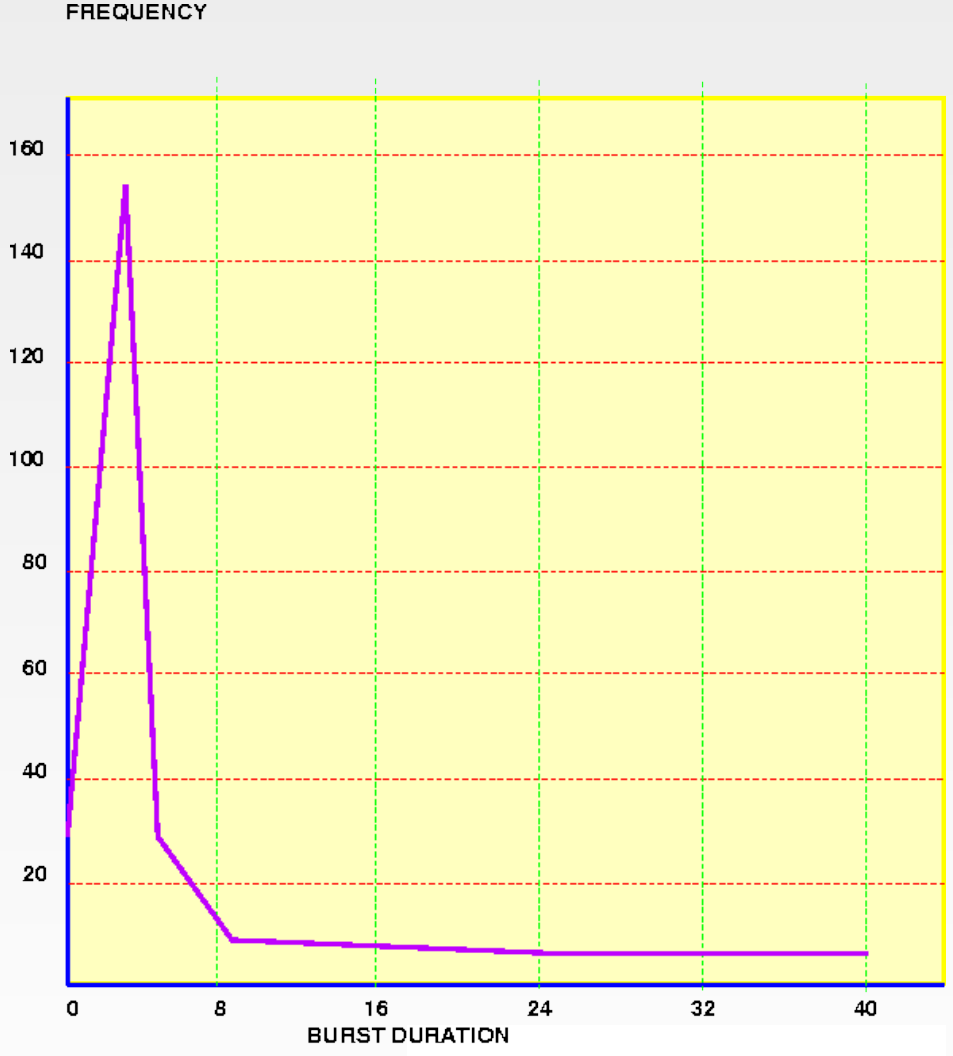
上海科技大学
ShanghaiTech University

Program Behavior Issues

- I/O boundedness
- CPU boundedness
- Urgency and priorities
- Frequency of preemption
- Process execution time
- Time Sharing

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU-I/O Burst Cycle

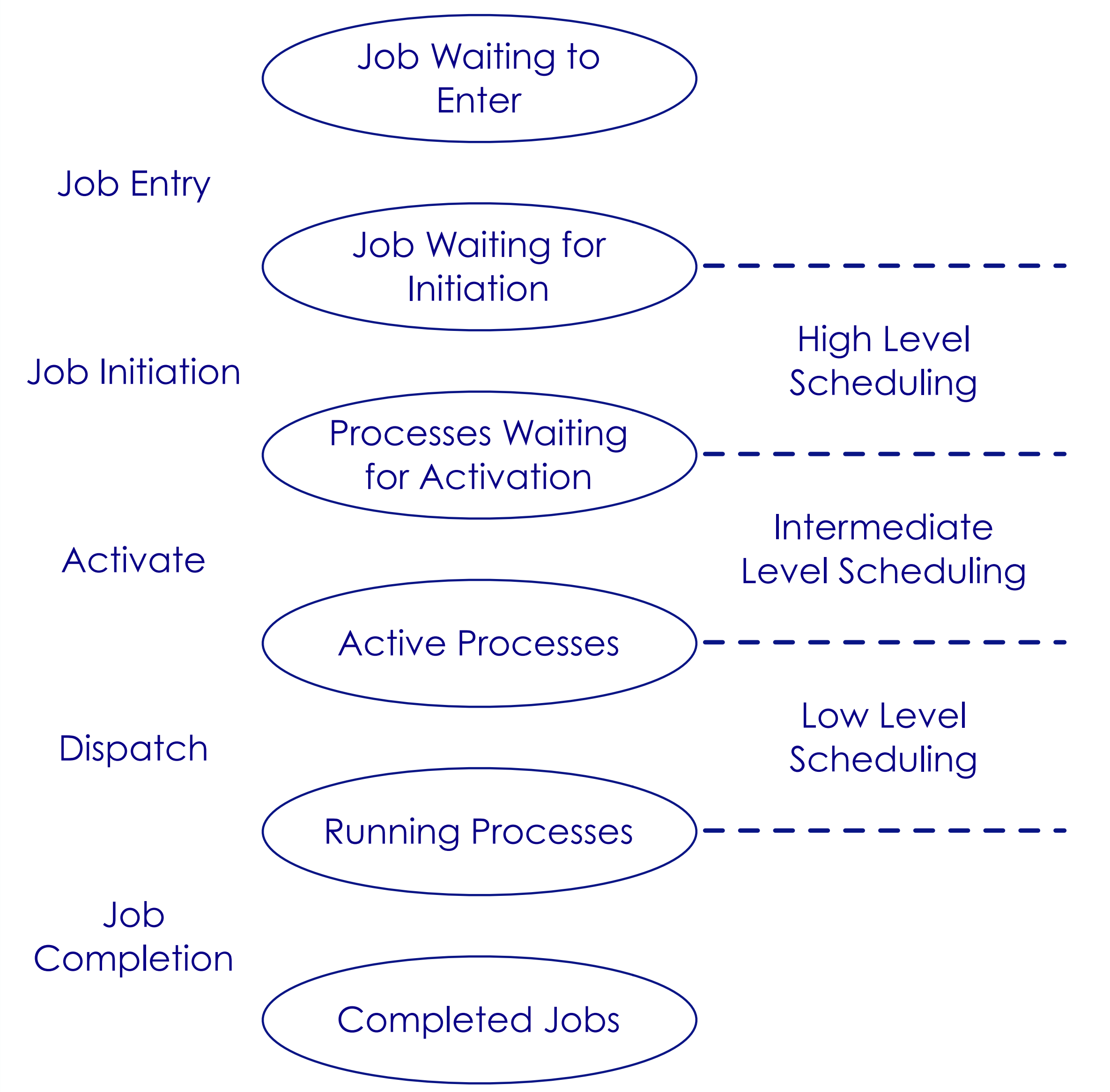


上海科技大学
ShanghaiTech University

Levels of Scheduling

- High level scheduling
  - Job scheduling
- Intermediate level scheduling
  - Medium term scheduling
- Low level scheduling
  - CPU scheduling

Levels of Scheduling (cont.)



Levels of Scheduling

CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
  - No-preemptive Scheduling
    - Process keeps CPU until
      - Process exits OR
      - Process switches to waiting state
  - Preemptive Scheduling
    - Process can be interrupted
      - Need to coordinate access to shared data

CPU Scheduling Decisions

- CPU scheduling decisions may take place when a process
  - Switches from running state to waiting state
  - Switches from running state to ready state
  - Switches from waiting state to ready state
  - Terminates

上海科技大学 ShanghaiTech University

CPU Scheduling Decisions

- CPU scheduling decisions may take place when a process
  - Switches from running state to waiting state
  - Switches from running state to ready state
  - Switches from waiting state to ready state
  - Terminates

Preemptive

CPU Scheduling Decisions

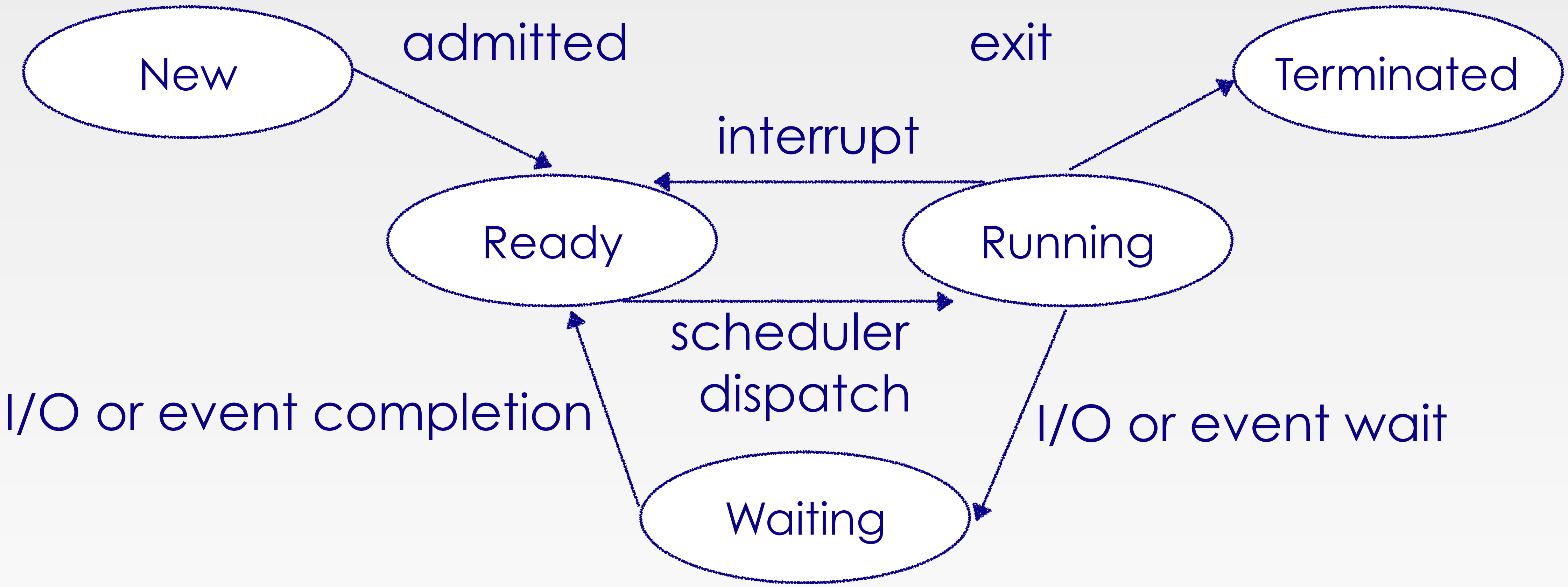
- CPU scheduling decisions may take place when a process
  - Switches from running state to waiting state
  - Switches from running state to ready state
  - Switches from waiting state to ready state
  - Terminates

Preemptive

Non-preemptive

上海科技大学 ShanghaiTech University

CPU Scheduling Decisions (cont.)



Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler. Involving:
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program

Dispatcher (cont.)

- Dispatcher Latency:
  - Time it takes to stop one process and start another
  - MUST be fast

Scheduling Criteria

- CPU utilization
  - As busy as possible
- Throughput
  - # of processes complete execution per time
- Turnaround Time
  - Time it takes to execute a process from its entry time

Scheduling Criteria (cont.)

- Waiting Time
  - Time it takes waiting in the ready queue
- Response Time
  - Time it takes from request submitted to the first response is produced

Optimization Criteria

- Maximum
  - CPU utilization
  - Throughput
- Minimum
  - Turnaround time
  - Waiting time
  - Response time

Observations: Scheduling Criteria

- Throughput vs. response time
  - Throughput related to responses time
  - But not identical
  - Minimizing responses time -> more context switching
- Fairness vs. response time
  - Share CPU among users in some equitable way
  - Fairness is not minimizing average response time
    - Less fair->better average response time

上海科技大学 ShanghaiTech University

Scheduling Algorithms

- First Come First Serve
- Shortest Job First
- Priority
- Round Robin
- Multilevel Queue
- Multilevel Feedback Queue
- Real-time Scheduling

FCFS Scheduling

- First Come First Serve
- Policy: Process that requests the CPU FIRST is allocated the CPU FIRST.
- Non-preemptive algorithm
- Implementation: FIFO queues
  - Incoming process is added to the tail of the queue
  - Process selected for execution is taken from head of queue

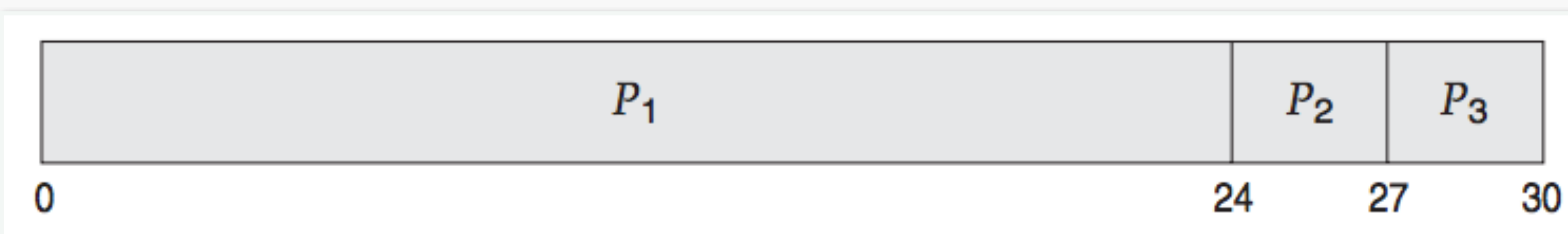
FCFS Scheduling (cont.)

- Performance metric: average waiting time in queue
- Gantt Charts are used to visualize schedules

FCFS Scheduling (cont.)

- Suppose the arrival order: P_1, P_2, P_3
- Waiting Time:
  - P_1=0, P_2=24, P_3=27
- Average Waiting Time:
  - (0+24+27)/3 = 17

Process	Burst Time
P_1	24
P_2	3
P_3	3



FCFS Scheduling (cont.)

- Suppose the arrival order: P_2, P_3, P_1
- Waiting Time:
  - $P_1=6$, $P_2=0$, $P_3=3$
- Average Waiting Time:
  - $(6+0+3)/3 = 3$
- Convoy Effect:
  - Short process behind long process
  - e.g. Single CPU bound process
  - e.g. I/O bound process

Process	Burst Time
P_1	24
P_2	3
P_3	3

FCFS Scheduling (cont.)

- Suppose the arrival order: P_2, P_3, P_1
- Waiting Time:
  - $P_1=6$, $P_2=0$, $P_3=3$
- Average Waiting Time:
  - $(6+0+3)/3 = 3$
- Convoy Effect:
  - Short process behind long process
  - e.g. Single CPU bound process
  - e.g. I/O bound process

Process	Burst Time
P_1	24
P_2	3
P_3	3

FCFS Scheduling (cont.)

- Pros: simple
- Cons: short job get stuck behind long ones

SJF Scheduling

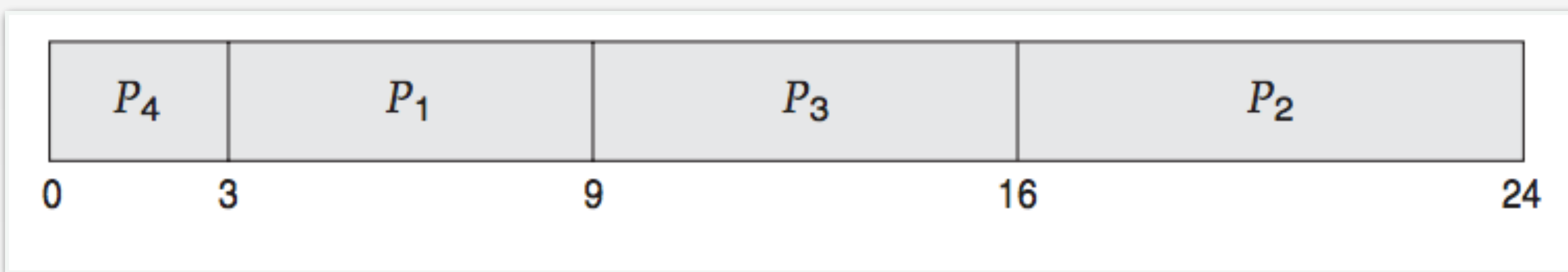
- Associate with each process the length of its next CPU burst
- Use these lengths to schedule the process with the shortest time
- Two Schemes
  - Non-preemptive
  - Preemptive
- SJF is optimal
  - Gives minimum average waiting time for a given set of processes

上海科技大学 ShanghaiTech University

SJF Scheduling: Non-Preemptive

- Average waiting time:
  - (3+16+9+0)/4 = 7

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

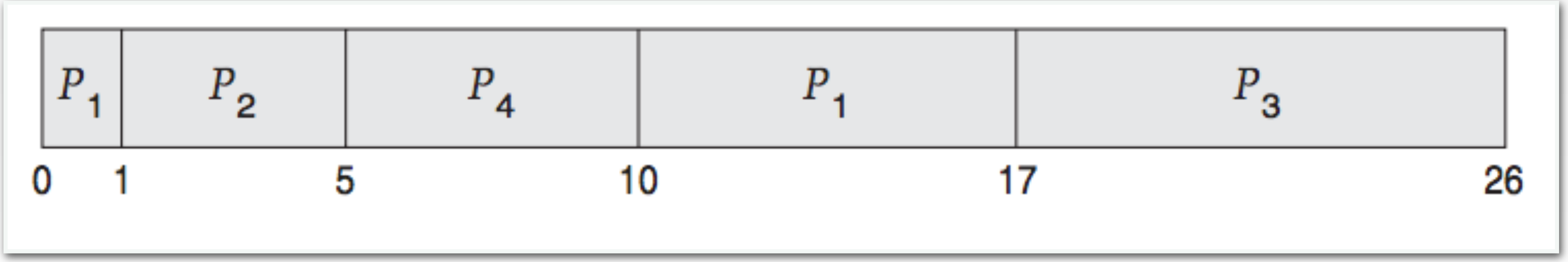


上海科技大学
ShanghaiTech University

SJF Scheduling: Preemptive

- Average waiting time:
  - ((10-1)+(1-1)+(17-2)+(5-3))/4 = 6.5
  - What if Non-preemptive?

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



SJF/SRTF Discussion

- The best you can do at minimizing average response time
  - Probably optimal
  - SRTF always at least as good as SJF
- Comparison of SRTF with FCFS
  - What if all jobs the same length
  - What if all jobs have varying length
- Starvation
  - Many small jobs, Large jobs never run

上海科技大学
ShanghaiTech University

SRTF Further Discussion

- Need to predict the future
  - How can we do this?
  - Some systems ask the user
  - BUT: have trouble predicting runtime of jobs
- Bottom line, can’t really know how long job will take
  - SRTF as a yardstick, so can’t do any better

SRTF Further Discussion (cont.)

- Pros.
  - Optimal in average response time
- Cons.
  - Hard to predict the future
  - Unfair

Priority Scheduling

- A priority value (int.) is associated with each process.
- Based on
  - Cost to user
  - Importance to user
  - Aging
  - %CPU time used in last XX hours

Priority Scheduling (cont.)

- CPU is allocated to process with the highest priority
  - Preemptive
  - Non-preemptive
- SJN is a priority scheme
  - The priority is the predicted next CPU burst time

Priority Scheduling (cont.)

- Problem
  - Starvation
  - Low priority processes may never execute
- Solution
  - Aging
  - As time progresses, increase the priority of the process

Round Robin Scheduling

- Each process gets a small unit of CPU time
  - Time quantum usually 10-100 ms
  - After this time has elapsed, the process is preempted and added to the end of the ready queue

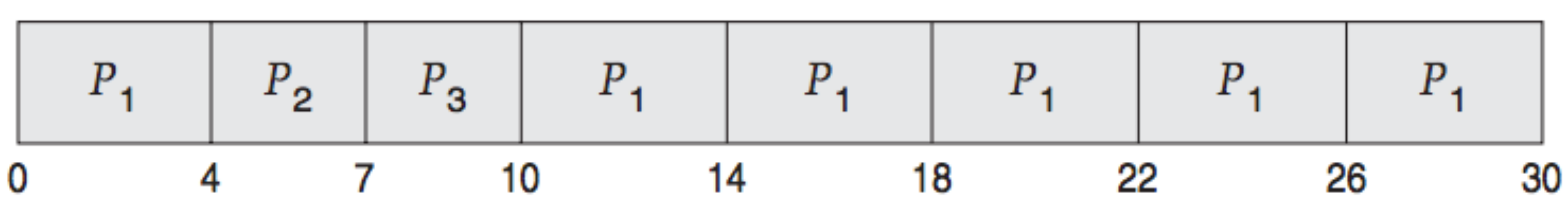
Round Robin Scheduling (cont.)

- *n* processes, time quantum= *q*
  - Each process gets *1/n* CPU time
  - At most *q* time units at a time
  - No process waits more than *(n-1)q* time units
  - Performance
    - Time slice *q* too large - response time poor
    - Time slice *q* too small - context switch overhead

Example of RR

- Time quantum = 4 ms
-

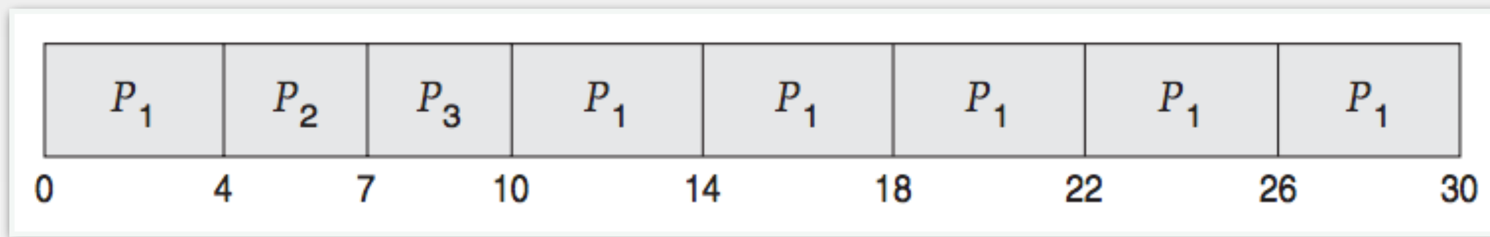
Process	Burst Time
P_1	24
P_2	3
P_3	3



Example of RR (cont.)

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Waiting time:
  - P_1: (10-4) = 6
  - P_2: (4-0) = 4
  - P_3: (7-0) = 7



- Average waiting time = (6+4+7)/3 = 5.66ms
- Pros: Better for short jobs, Fair
- Cons: Context switching overhead for long jobs

Round Robin Discussion

- How to choose time slice?
  - What if too big?
    - Response time suffers
  - What if infinite?
    - Get back to FCFS (FIFO)
  - What if too small?
    - Throughput suffers

上海科技大学 ShanghaiTech University

Round Robin Discussion (cont.)

- Actual choices of time slice
  - Initially, UNIX times slice one second
    - Worked ok when UNIX was used by 1 or 2 people
    - What if 3 compilation going on? 3 seconds to echo each keystroke
  - In practice, need to balance short job performance and long job throughput
    - Typical time slice today is between 10 - 100 ms
    - Typical context-switching overhead is 0.1 - 1 ms
    - Roughly 1% overhead due to context-switching

Comparison between FCFS and RR

- Assuming zero-cost context-switching time, is RR always better then FCFS?
- E.g., 10 jobs, each take 100s of CPU time, RR scheduler quantum of 1s, all jobs start at the same time?
  - Finish at the same time
  - Average response time RR worse

Job #	FCFS	RR
1	100	991
2	200	992
…	…	…
10	1000	1000

Comparison (cont.)

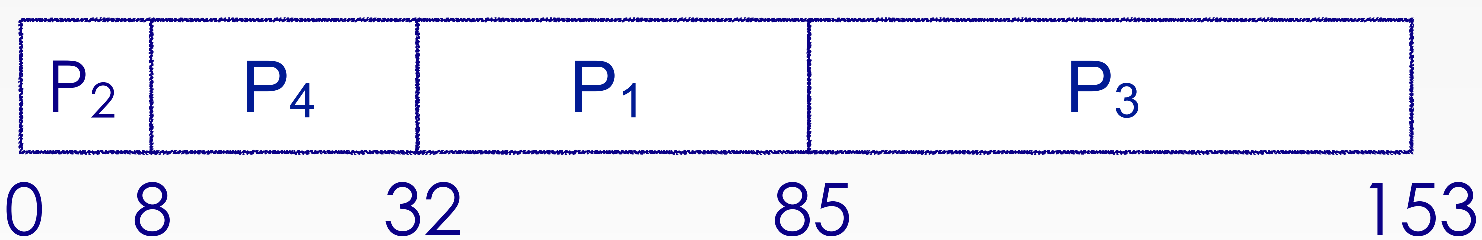
- Cache state must be shared among all jobs with RR but can be devoted to each job with FCFS
  - Total time for RR longer even for zero-cost switch

Example with Different Time Quantum

- Time Quantum = 20

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

- Best FCFS



上海科技大学
ShanghaiTech University

Example with Different Time Quantum

	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	Best FCFS	32	0	85	8	31.25
	Q=1	84	22	85	57	62
	Q=5	82	20	85	58	61.25
	Q=8	80	8	85	56	57.25
	Q=10	82	10	85	68	61.25
	Q=20	72	20	85	88	66.25
	Worst FCFS	68	145	0	121	83.5
Completion Time	Best FCFS	85	8	153	32	69.5
	Q=1	137	30	153	81	100.5
	Q=5	135	28	153	82	99.5
	Q=8	133	16	153	80	95.5
	Q=10	125	28	153	112	104.5
	Worst FCFS	121	153	68	145	121.75

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

上海科技大学 ShanghaiTech University

Multilevel Queue

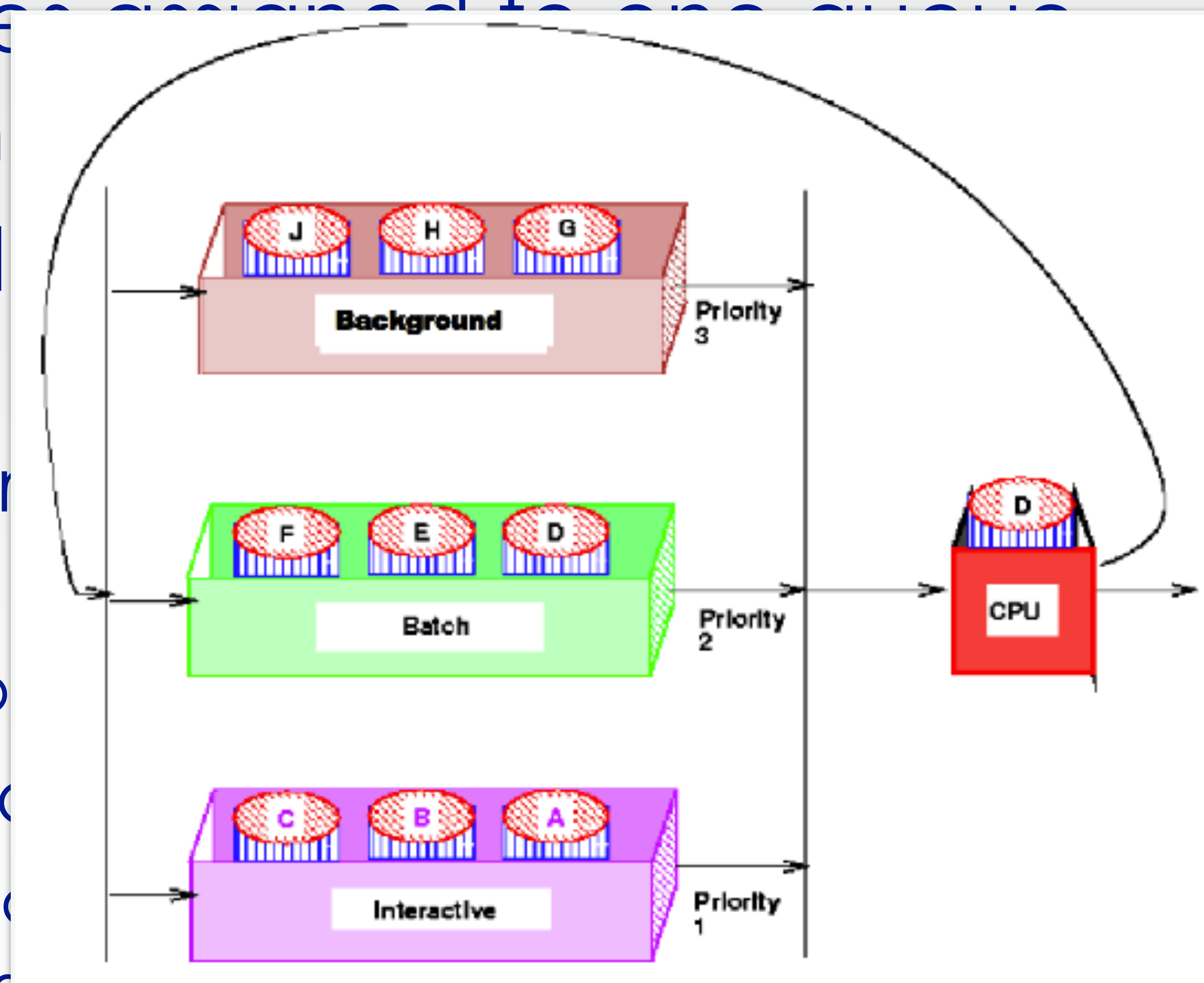
- Ready queue partitioned into separate queues
  - e.g., system processes, foreground(interactive), background(batch), student processes..
- Each queue has its own scheduling algorithm
  - e.g. foreground (RR), background (FCFS)

Multilevel Queue (cont.)

- Processes assigned to one queue permanently
- Scheduling must be done between the queues
  - Fixed priority
    - Serve all from foreground, then from background
    - Possibility of starvation
  - Time Slice
    - Each queue get some CPU time that it schedules
    - E.g. 80% foreground, 20% background

Multilevel Queue (cont.)

- Processe
  perman
- Schedul ... he
  queues
  - Fixed pr
    - Serve ... round
    - Possib
  - Time Slic
    - Each ... edules
    - E.g. 80% foreground, 20% background



上海科技大学
ShanghaiTech University

Multilevel Feedback Queue

- Multilevel queue with priorities
- A process can move between the queues
  - Aging can be implemented this way

Multilevel Feedback Queue (cont.)

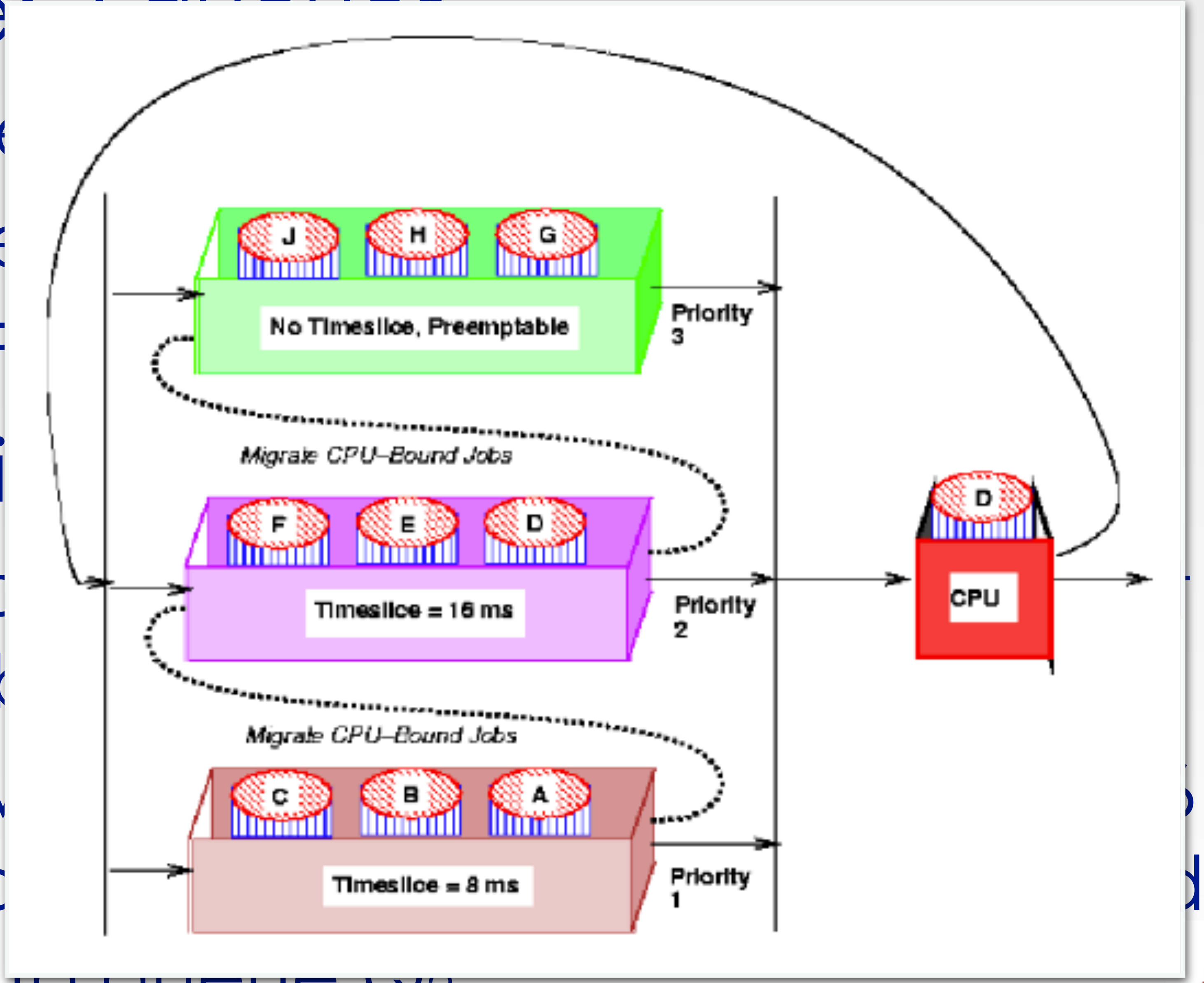
- Parameters for a multilevel feedback queue scheduler
  - Number of queues
  - Scheduling algorithm for each queue
  - Method used to determine
    - When to upgrade a process
    - When to demote a process
    - Which queue a process will enter when that process needs service

Multilevel Feedback Queue (cont.)

- Example: 3 queues
  - Q_0 - time quantum 8 ms (RR)
  - Q_1 - time quantum 16 ms (RR)
  - Q_2 - FCFS
- Scheduling
  - New job enters Q_0 - When it gains CPU, it receives 8 ms. If job does not finish, move it to Q_1
  - At Q_1, when job gains CPU, it receives 16 more ms. If job does not complete, it is preempted and moved to queue Q_2

Multilevel Feedback Queue (cont.)

- Example: 3 queues
  - Q_0 - tim
  - Q_1 - time
  - Q_2 - FCF
- Scheduli
  - New job receives 8 ms. If job
  - At Q_1, w more ms. If job d and moved to queue Q_2



上海科技大学 ShanghaiTech University

Multiple-Processor Scheduling

- CPU scheduling becomes more complex when multiple CPUs are available
  - Have one ready queue accessed by each CPU
    - Self scheduled - each CPU dispatches a job from ready queue
    - Master_Slave - one CPU schedules the other CPUs

Multiple-Processor Scheduling (cont.)

- Homogeneous processors within multiprocessor
  - Permits load sharing
- Asymmetric multiprocessing
  - Only 1 CPU runs kernel
  - Others run users programs
  - Alleviates need for data sharing

Real-Time Scheduling

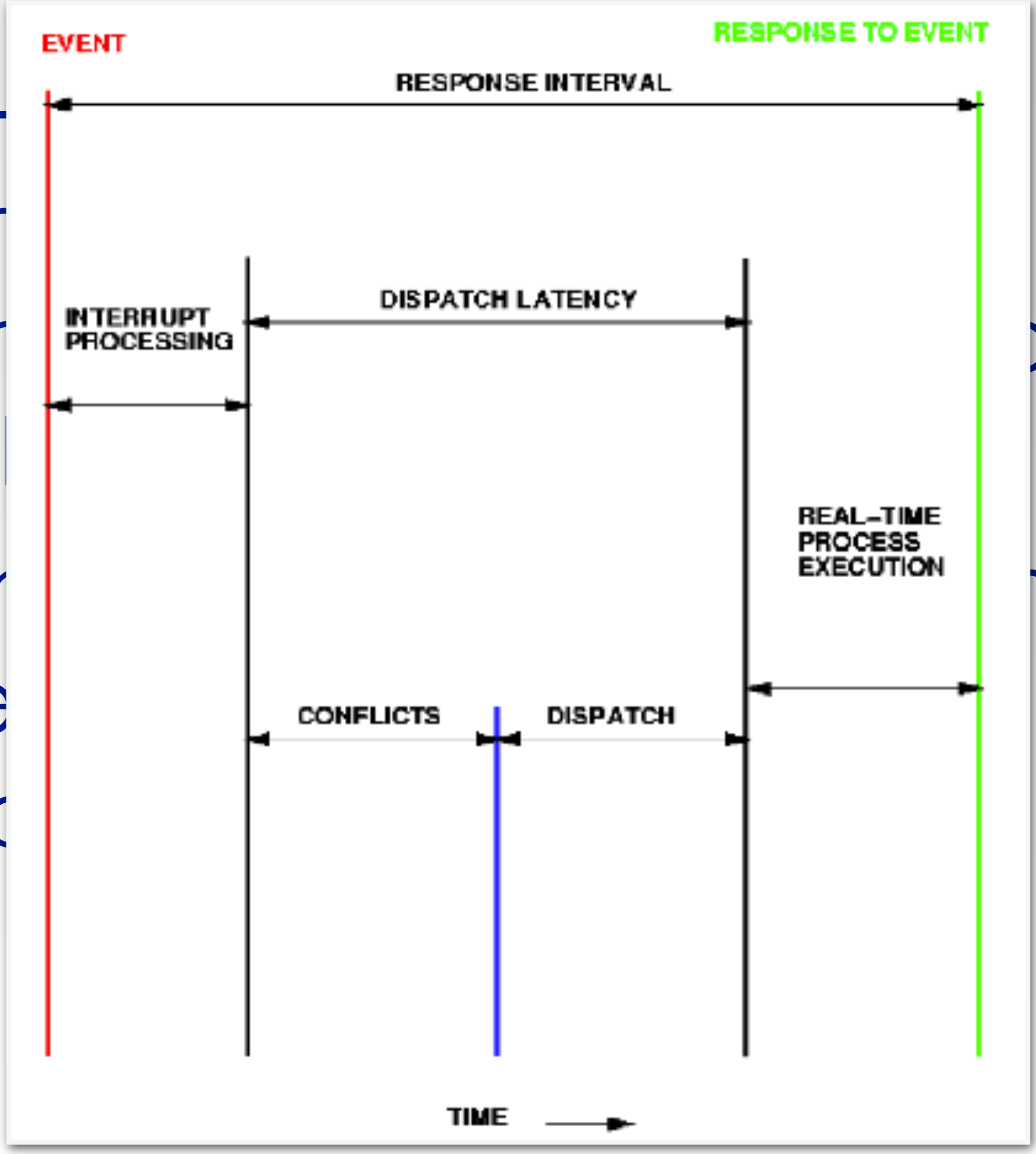
- Hard real-time computing
  - Require to complete a critical task within a guaranteed amount of time
- Soft real-time computing
  - Requires that critical processes receive priority over less fortunate ones
- Types of real-time schedulers
  - Periodic schedulers: fixed arrival rate
  - Demand-driven schedulers: variable arrival rate
  - Deadline schedulers: priority determined by deadline
  - …

Issues in Real-Time Scheduling

- Dispatch Latency
  - Problem: need to keep dispatch latency small, OS may enforce process to wait for system call or I/O to complete
  - Solution: make system calls preemptive, determine safe criteria such that kernel can be interrupt

Issues in Real-Time Scheduling

- Dispatch L
  - Problem: r latency small, OS r wait for system ca
  - Solution: m mptive, determine kernel can be interru



Issues in Real-Time Scheduling (cont.)

- Priority Inversion and Inheritance
  - Problem: priority inversion
    - Higher priority process needs kernel resource currently being used by another lower priority process
    - Higher priority process must wait
  - Solution: priority inheritance
    - Low priority process now inherits high priority until it has completed use of the resource in question.

上海科技大学 ShanghaiTech University

Algorithm Evaluation

- Deterministic modeling
- Queuing models and Queuing theory
  - Distributions of CPU and I/O bursts
  - Little's formula: $n = \lambda \times W$
    - n: average queue length
    - λ: average arrival rate
    - W: average waiting time in queue
- Other techniques
  - Simulation
  - Implementation

Case Study: Linux Scheduler

- O(1) CPU Scheduler
- Priority-based scheduler: 140 priorities
  - 40 for User Tasks
  - 100 for Real-time/Kernel
  - Lower priority value → Higher priority
  - All algorithms O(1): schedule *n* processes in constant time
    - compute time-slices/priorities/interactivity credits when job finishes time slice
    - 140-bit bit mask indicates presence or absence of job(s) at given priority level

Linux Scheduler (cont.)

- Two separate priority queues (arrays)
  - Active
  - Expired
  - All tasks in the active queue use up their timeslices and get placed on the expired queue, after which queue swapped

Linux Scheduler (cont.)

- Timeslice depends on priority - linearly mapped onto timeslice range
  - Like multi-level queue (one queue per priority) with different timeslice at each level
  - Execution split into “Timeslice Granularity” chunks — RR through priority

Linux Scheduler (cont.)

- Heuristics
  - User-task priority adjusted ±5 based on heuristics
    - **`p -> sleep_avg = sleep_time - run_time`**
    - Higher **`sleep_avg`** → more I/O bound the task, more reward
  - Interactive Credit
    - Earned when task sleeps for "long" time, Spend when task runs for "long" time
    - IC is used to provide hysteresis to avoid changing interactivity for temporary changes in behavior
  - BUT, interactive tasks get special dispensation
    - To try to maintain interactivity
    - Placed back into active queue, unless another track has starved for too long..

上海科技大学 ShanghaiTech University

Linux Scheduler (cont.)

- Real-Time tasks
  - Always preempt non-RT tasks and no dynamic adjustment of priorities
  - Scheduling schemes
    - **`SCHED_FIFO`**: preempts other tasks, no timeslice limit
    - **`SCHED_RR`**: preempts normal tasks, RR scheduling amongst tasks of same priority

Linux Completely Fair Scheduler (CFS)

- First appeared in 2.6.23, modified in 2.6.24
- Inspired by Networking "Faire Queuing"
  - Each process given their fair share of resources
  - Models an "ideal multitasking processor"
    - *N* processes execute simultaneously as if they truly got *1/N* of the processor
  - Priorities reflected by weights such that increasing a task's priority by 1 always gives the same fractional increase in CPU time - regardless of current priority

CFS (cont.)

- Idea: track "virtual time" received by each process when it is running
  - Take real execution time, scale by weighting factor
    - Lower priority → real time divided by greater weight
    - Actually - multiple by sum of all weights/current weight
  - Keep virtual time advancing at same rate

上海科技大学 ShanghaiTech University

CFS (cont.)

- Red-Black tree holds all runnable processes sorted on *vruntime*
  - O(log n) time to perform insertions/deletions
    - Cache the item at far left (item with earliest *vruntime*)
  - Scheduler always takes process with smallest *vruntime* (far left time)

CFS Examples

- Suppose Targeted Latency = 20ms
- and Minimum Granularity = 1ms
- Two CPU bound tasks with same priorities
  - One task gets 5ms, another gets 15ms
- 40 tasks: each gets 1ms (no longer totally fair - miss target latency)

CFS Examples (cont.)

- One CPU bound task, one interactive task same priority
  - While interact task sleeps, CPU-bound task runs, increments *vruntime*
  - When interact task wakes up, runs immediately
- Group scheduling facilities (2.6.24)
  - Can give fair fractions to groups (user or other process group)
  - So, two users, one starts 1 process, other starts 40, each gets 50% CPU

Summary

- RR scheduling
  - Cycle among all ready processes
  - Pros: better for short jobs
- SJF/SRTF scheduling
  - Pros: optimal (average response time)
  - Cons: hard to predict the future, unfair
- Multi-Level Feedback Queue scheduling
  - Multiple queue of different priorities and algorithms
  - Automatic promotion/demotion

Summary (cont.)

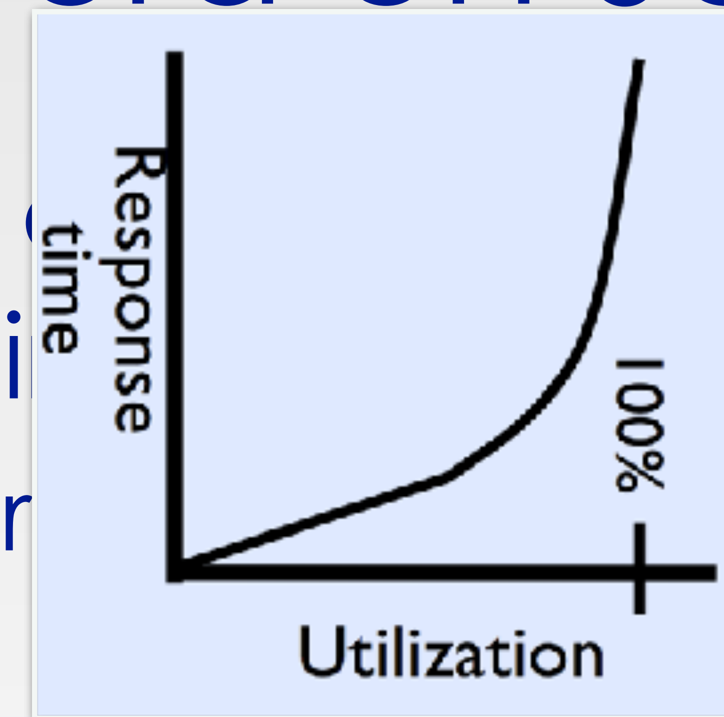
- Hard Real-Time
  - Attempt to meet all deadlines
    - EDF (earliest deadline first), LLF(least laxity first)
    - RMS (rate-monotonic scheduling), DM (deadline monotonic scheduling)
- Software Real-Time
  - Attempt to meet deadlines with high probability
    - Minimize miss ration/ Maximize completion ratio
    - Importance for multimedia applications
    - CBS (constant bandwidth server)

上海科技大学 ShanghaiTech University

A Final Word on Scheduling

- When do the details of the scheduling policy and fairness really matter?
  - When there aren't enough resources to go around
- An interesting implication of this curve:
  - Most scheduling algorithms work fine in the linear portion of the load curve
  - Response time goes to infinity when utilization is 100%

A Final Word on Scheduling



- When do the … e scheduling policy and fai… atter?
  - When there ar… esources to go around
- An interesting implication of this curve:
  - Most scheduling algorithms work fine in the linear portion of the load curve
  - Response time goes to infinity when utilization is 100%