CPU Scheduling

Yi Shi

Fall 2018

Xi’an Jiaotong University

Goals for Today

- CPU Schedulers
- Scheduling Evaluation Metrics
- Scheduling Algorithms
- Others
  - Multiple-Processor Scheduling
  - Thread Scheduling
  - Real-Time Scheduling

Process Model

- Process alternates between CPU and I/O bursts
  - CPU-bound jobs: Long CPU bursts

  - I/O-bound: Short CPU bursts

  - I/O burst = process idle, switch to another “for free”
  - Problem: don’t know job’s type before running

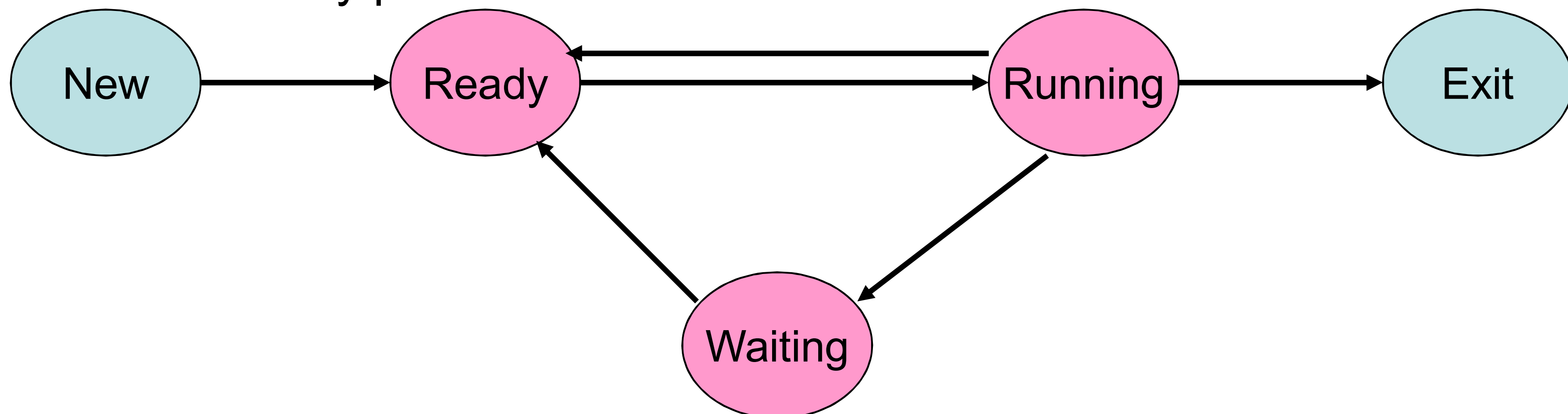
Schedulers

- Process migrates among several queues
  - Device queue, job queue, ready queue
- Scheduler selects a process to run from these queues
- Long-term scheduler:
  - load a job in memory
  - Runs infrequently
- Short-term scheduler:
  - Select ready process to run on CPU
  - Should be fast
- Middle-term scheduler (aka swapper)
  - Reduce multiprogramming or memory consumption

Process Scheduling

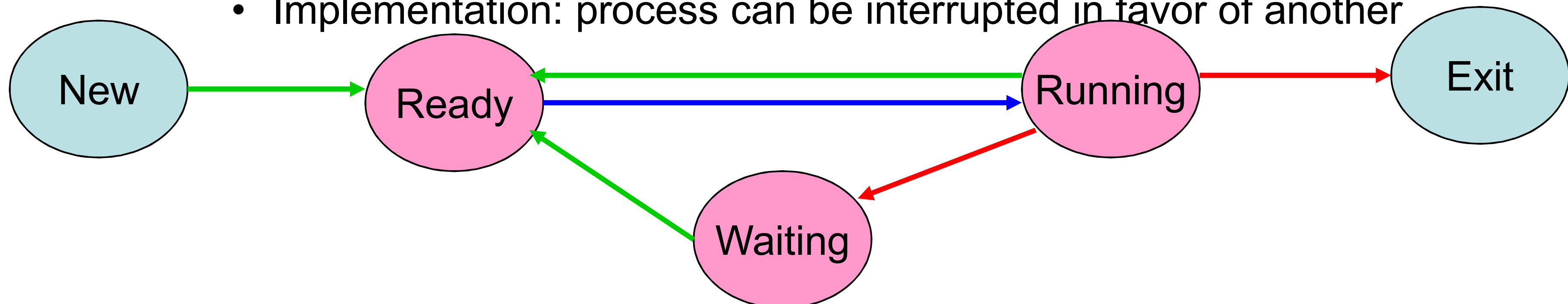
- “process” and “thread” used interchangeably
- Which process to run next

- Many processes in “ready” state
- Which ready process to pick to run on the CPU?
  - 0 ready processes: run idle loop
  - 1 ready process: easy!
  - > 1 ready process: what to do?



When does scheduler run?

- Non-preemptive minimum
  - Process runs until voluntarily relinquish CPU
    - process blocks on an event (e.g., I/O or synchronization)
    - process terminates
    - process yields
- Preemptive minimum
  - All of the above, plus:
    - Event completes: process moves from blocked to ready
    - Timer interrupts
    - Higher priority process has just been created
    - Implementation: process can be interrupted in favor of another



Goals for Today

- CPU Schedulers
- Scheduling Evaluation Metrics
- Scheduling Algorithms
- Others
  - Multiple-Processor Scheduling
  - Thread Scheduling
  - Real-Time Scheduling

Scheduling Evaluation Metrics

- Many quantitative criteria for evaluating scheduling algorithm**:**
  - CPU utilization: percentage of time the CPU is not idle
  - Throughput: completed processes per time unit
  - Turnaround time: submission to completion
  - Waiting time: time spent on the ready queue
  - Response time: response latency
  - Predictability: variance in any of these measures
- The right metric depends on the context

- An underlying assumption:
  - “response time” most important for interactive jobs (I/O bound)

"The perfect CPU scheduler"

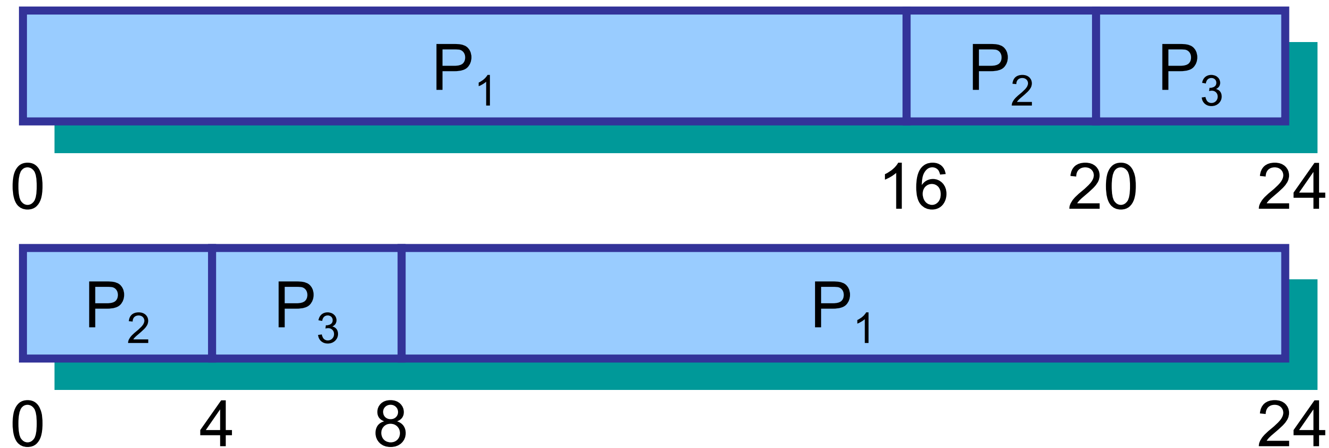
- Minimize latency: response or job completion time
- Maximize throughput: Maximize jobs / time.
- Maximize utilization: keep CPU and I/O devices busy.
- Fairness: everyone makes progress, no one starves

Goals for Today

- CPU Schedulers
- Scheduling Evaluation Metrics
- Scheduling Algorithms
- Others
  - Multiple-Processor Scheduling
  - Thread Scheduling
  - Real-Time Scheduling

Scheduling Algorithms FCFS

- **First-come First-served (FCFS) (FIFO)**
  - Jobs are scheduled in order of arrival
  - Non-preemptive
- **Problem:**
  - Average waiting time depends on arrival order



- **Advantage:** really simple!

Convoy Effect

- A CPU bound job will hold CPU until done,
  - or it causes an I/O burst
    - rare occurrence, since the thread is CPU-bound

  $\Rightarrow$ long periods where no I/O requests issued, and CPU held
  - Result: poor I/O device utilization
- Example: one CPU bound job, many I/O bound
    - CPU bound runs (I/O devices idle)
    - I/O bound jobs waiting for CPU
    - CPU bound I/O
    - I/O bound job(s) run, quickly block on I/O (CPU idle)
    - CPU bound runs again
    - I/O completes
    - CPU bound still runs while I/O devices idle (continues…)

Scheduling Algorithms LIFO

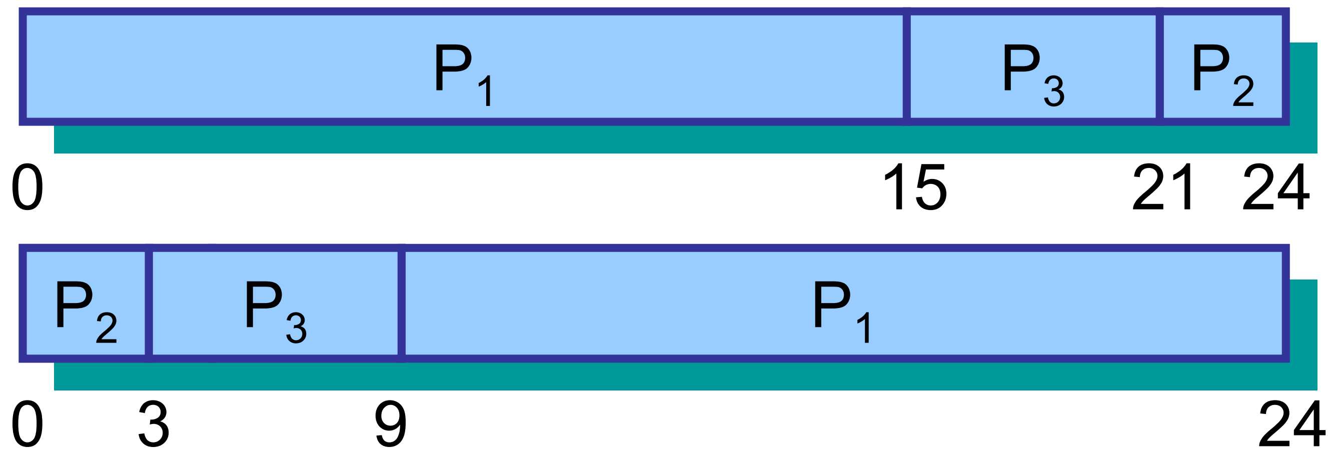
- **Last-In First-out (LIFO)**
  - Newly arrived jobs are placed at the head of ready queue
  - Improves response time for newly created threads
- **Problem:**
  - May lead to starvation – early processes may never get CPU

Problem

- You work as a short-order cook
  - Customers come in and specify which dish they want
  - Each dish takes a different amount of time to prepare
- Your goal:
  - minimize average time the customers wait for their food
- What strategy would you use ?
  - Note: most restaurants use FCFS.

Scheduling Algorithms: SJF

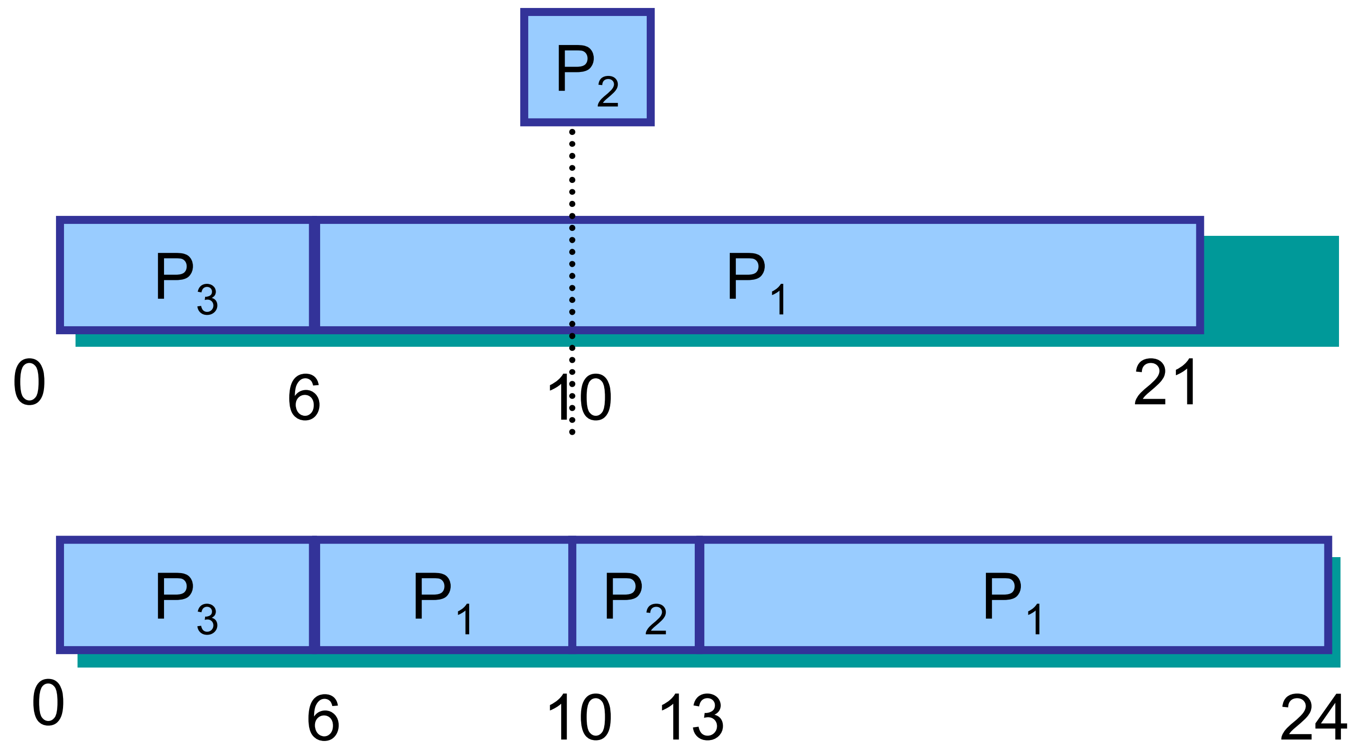
- **Shortest Job First (SJF)**
  - Choose the job with the shortest next CPU burst
  - Provably optimal for minimizing average waiting time



- **Problem:**
  - Impossible to know the length of the next CPU burst

Scheduling Algorithms SRTF

- SJF can be either preemptive or non-preemptive
  - New, short job arrives; current process has long time to execute
- Preemptive SJF is called *shortest remaining time first*



Shortest Job First Prediction

- Approximate next CPU-burst duration
  - from the durations of the previous bursts
    - The past can be a good predictor of the future
- No need to remember entire past history
- Use exponential average:

t_n duration of the nth CPU burst

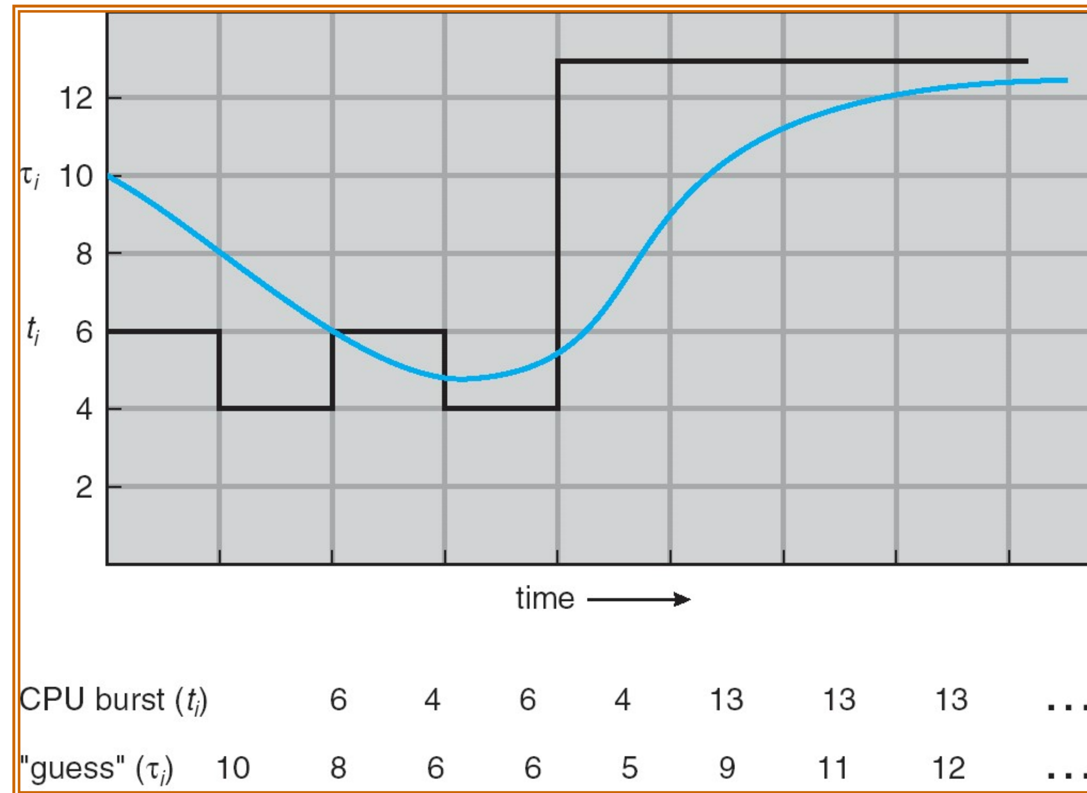
τ_{n+1} predicted duration of the $(n+1)^{st}$ CPU burst

$$\tau_{n+1} = \alpha\, t_n + (1- \alpha)\, \tau_n$$

where $0 \leq \alpha \leq 1$

α determines the weight placed on past behavior

Prediction of the Length of the Next CPU Burst



Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j \alpha\, t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

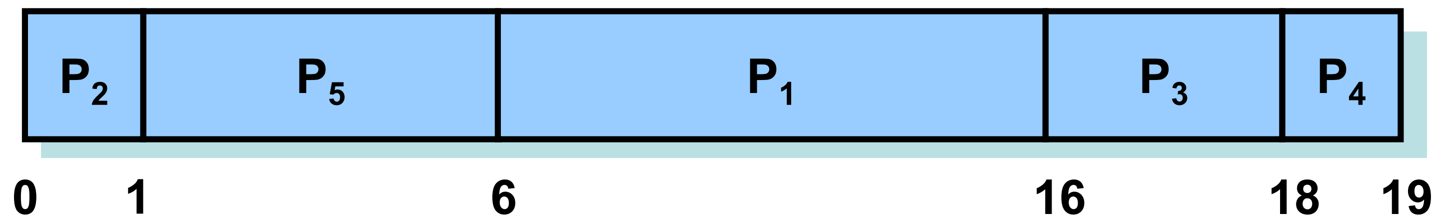
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Priority Scheduling

- **<u>Priority Scheduling</u>**
  - Choose next job based on priority
  - For SJF, priority = expected CPU burst
  - Can be either preemptive or non-preemptive
- **Problem:**
  - Starvation: jobs can wait indefinitely
- **Solution to starvation**
  - Age processes: increase priority as a function of waiting time

Priority Scheduling

Time 0:	Process	Burst Time	Priority
	P_1	10	3
	P_2	1	1
	P_3	2	4
	P_4	1	5
	P_5	5	2



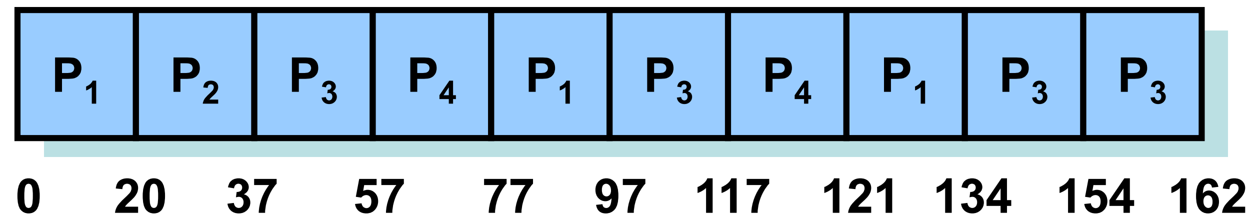
Round Robin

- **Round Robin (RR)**
  - Often used for timesharing
  - Ready queue is treated as a circular queue (FIFO)
  - Each process is given a time slice called a *quantum*
  - It is run for the quantum or until it blocks
  - RR allocates the CPU uniformly (fairly) across participants.
  - If average queue length is n, each participant gets 1/n

RR with Time Quantum = 20

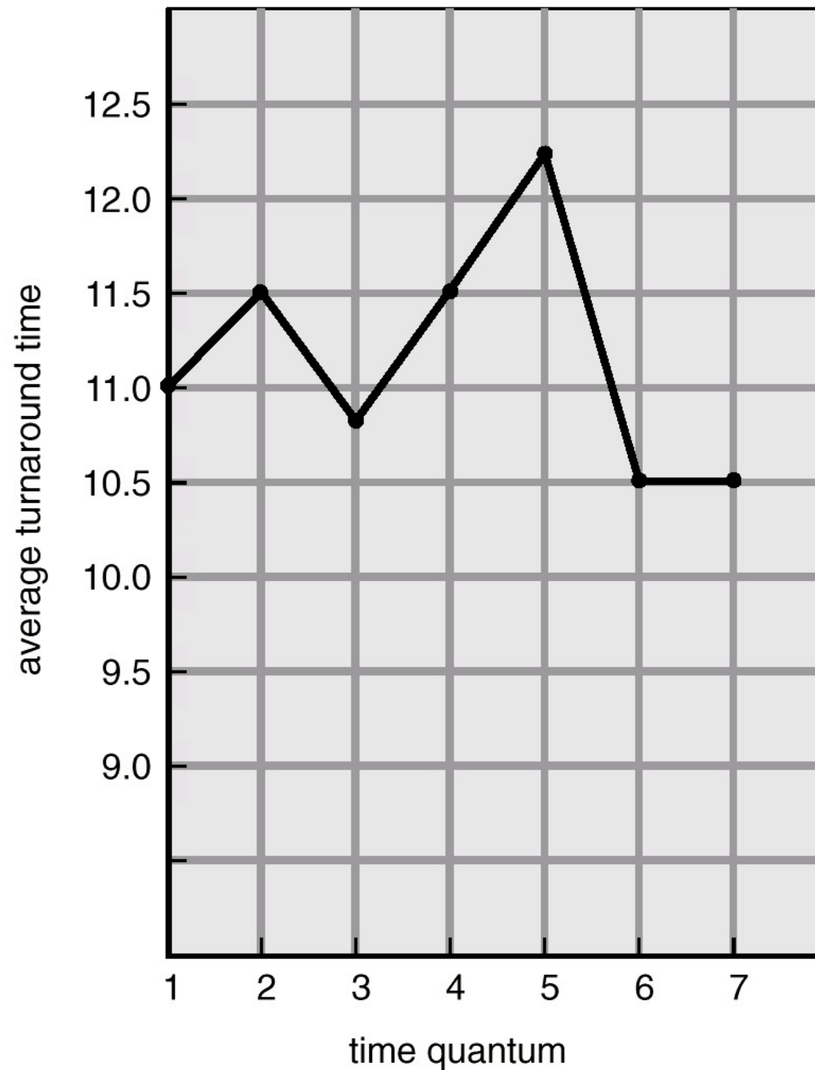
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



- Higher average turnaround than SJF,
- But better response time

Turnaround Time w/ Time Quanta



process	time
P_1	6
P_2	3
P_3	1
P_4	7

RR: Choice of Time Quantum

- Performance depends on length of the timeslice
  - Context switching isn't a free operation.
  - If timeslice time is set too high
    - attempting to amortize context switch cost, you get FCFS.
    - i.e. processes will finish or block before their slice is up anyway
  - If it's set too low
    - you're spending all of your time context switching between threads.
  - Timeslice frequently set to 10~100 milliseconds
  - Context switches typically cost < 1 millisecond

Moral:

Context switch is usually negligible (< 1% per timeslice) otherwise you context switch too frequently and lose all productivity

Multi-level Queue Scheduling

- Implement multiple ready queues based on job “type”
  - interactive processes
  - CPU-bound processes
  - batch jobs
  - system processes
  - student programs
- Different queues may be scheduled using different algorithms
- Intra-queue CPU allocation is either strict or proportional
- Problem: Classifying jobs into queues is difficult
  - A process may have CPU-bound phases as well as interactive ones

Multilevel Queue Scheduling

Highest priority

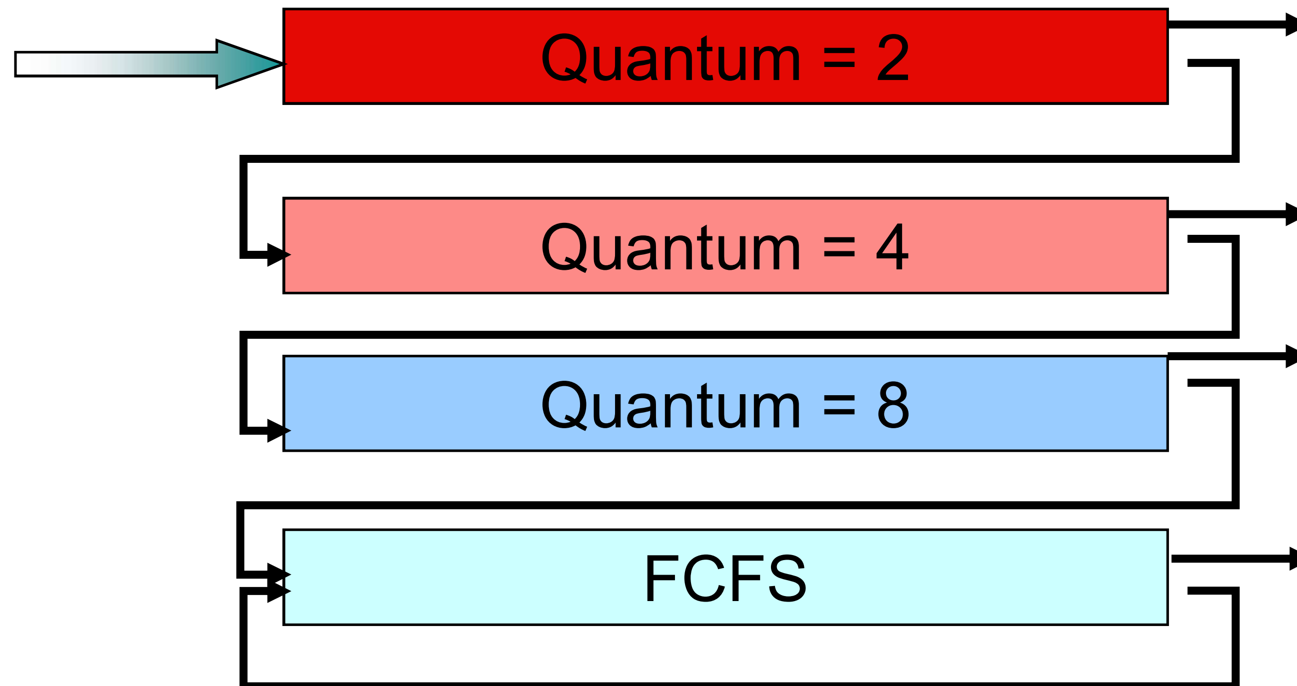
Lowest priority

Multi-level Feedback Queues

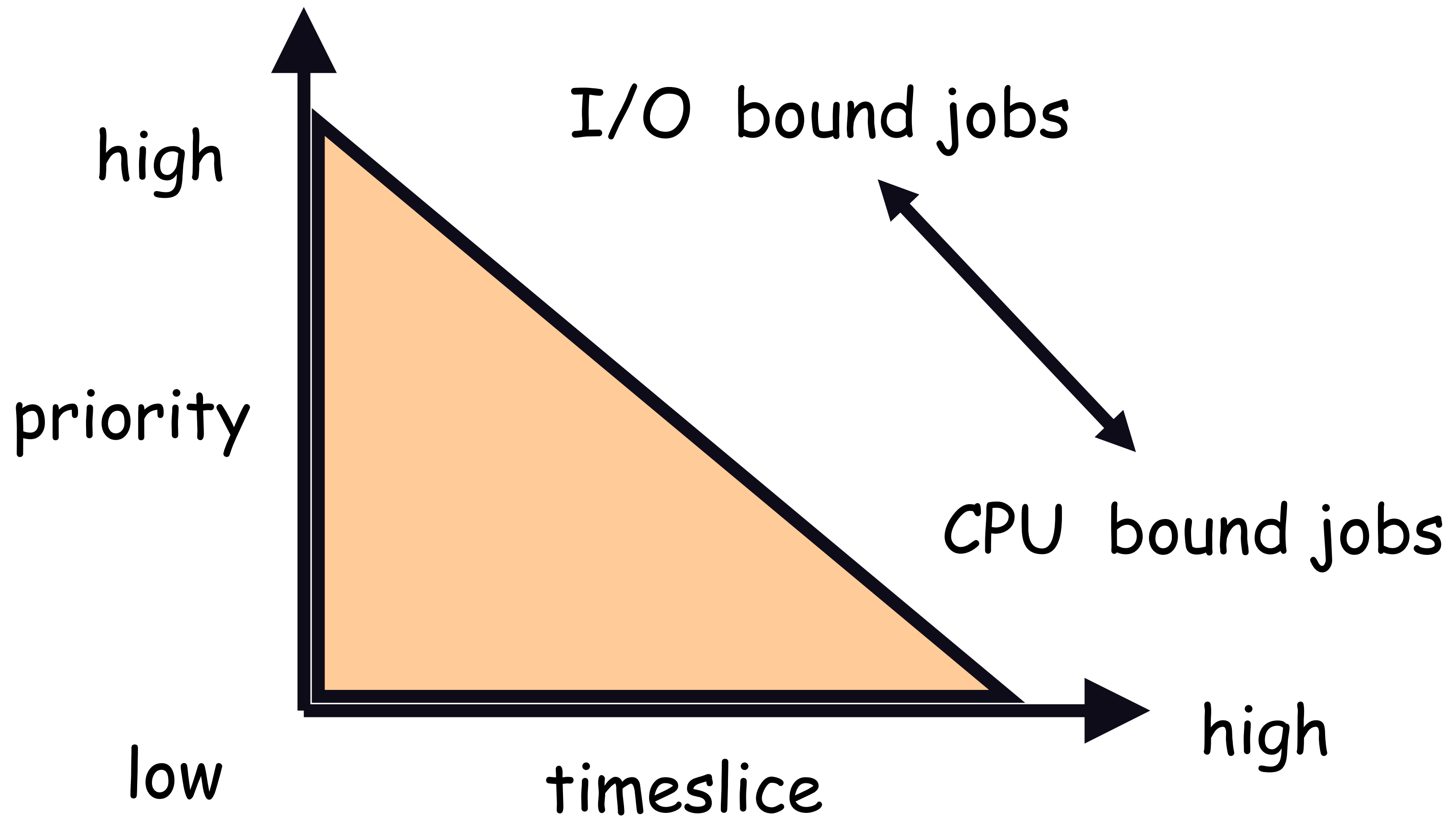
- Implement multiple ready queues
  - Different queues may be scheduled using different algorithms
  - Just like multilevel queue scheduling, but assignments are not static
- Jobs move from queue to queue based on feedback
  - Feedback = The behavior of the job,
    - e.g. does it require the full quantum for computation, or
    - does it perform frequent I/O ?

- Very general algorithm
- Need to select parameters for:
  - Number of queues
  - Scheduling algorithm within each queue
  - When to upgrade and downgrade a job

Multilevel Feedback Queues



A Multi-level System



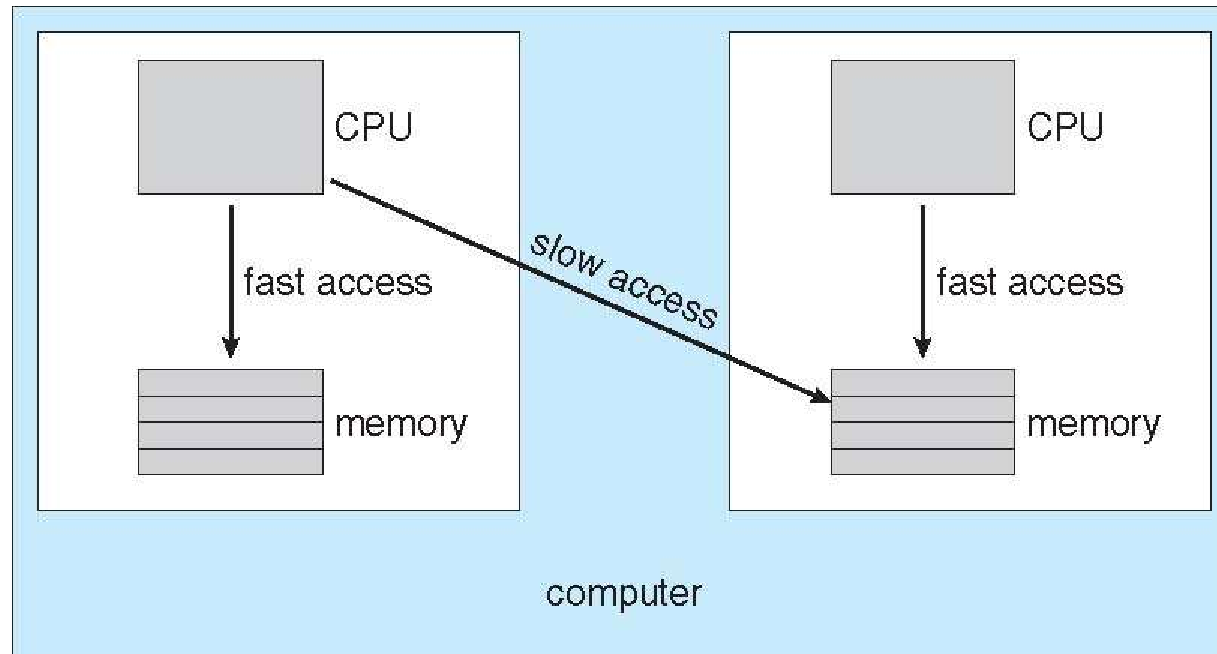
Goals for Today

- CPU Schedulers
- Scheduling Evaluation Metrics
- Scheduling Algorithms
- Others
  - Multiple-Processor Scheduling
  - Thread Scheduling
  - Real-Time Scheduling

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- **Homogeneous processors** within a multiprocessor
- **Asymmetric multiprocessing** – only one processor accesses the system data structures, alleviating the need for data sharing
- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
- **Processor affinity** – process has affinity for processor on which it is currently running
  - **soft affinity**
  - **hard affinity**
- **Load Balancing**
  - **push migration**
  - **pull migration**

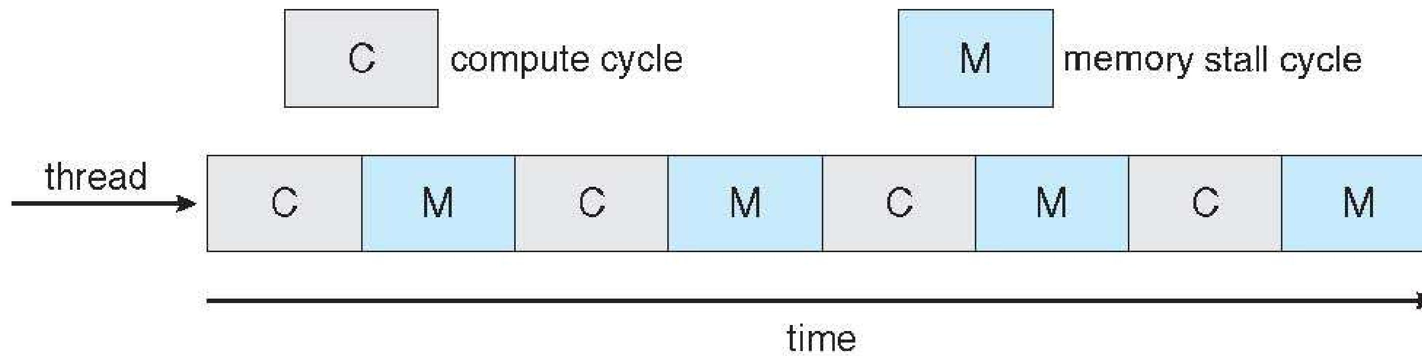
NUMA and CPU Scheduling



Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consume less power
- Multiple threads per core also growing: *symmetric multithreading(SMT)* or *hyperthreading*
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens

Multithreaded Multicore System



Thread Scheduling

Since all threads share code & data segments

- Option 1: Ignore this fact
- Option 2: Gang scheduling
  - run all threads belonging to a process together (multiprocessor only)
  - if a thread needs to synchronize with another thread
    - the other one is available and active
- Option 3: Two-level scheduling:
  - Medium level scheduler
  - schedule processes, and within each process, schedule threads
  - reduce context switching overhead and improve cache hit ratio

Real-time Scheduling

- Real-time processes have timing constraints
  - Expressed as deadlines or rate requirements
- Common RT scheduling policies
  - Rate monotonic
    - Just one scalar priority related to the periodicity of the job
    - Priority = 1/rate
    - Static
  - Earliest deadline first (EDF)
    - Dynamic but more complex
    - Priority = deadline
- Both require admission control to provide guarantees

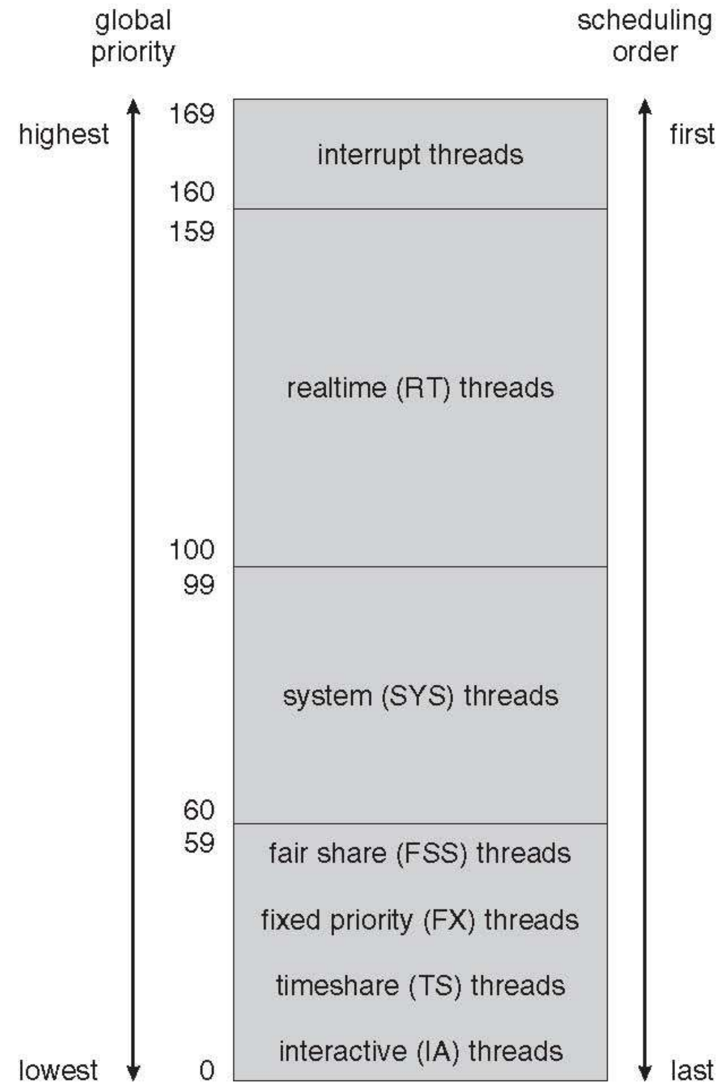
Operating System Examples

- Solaris scheduling
- Windows XP scheduling
- Linux scheduling

Solaris Dispatch Table

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

Solaris Scheduling



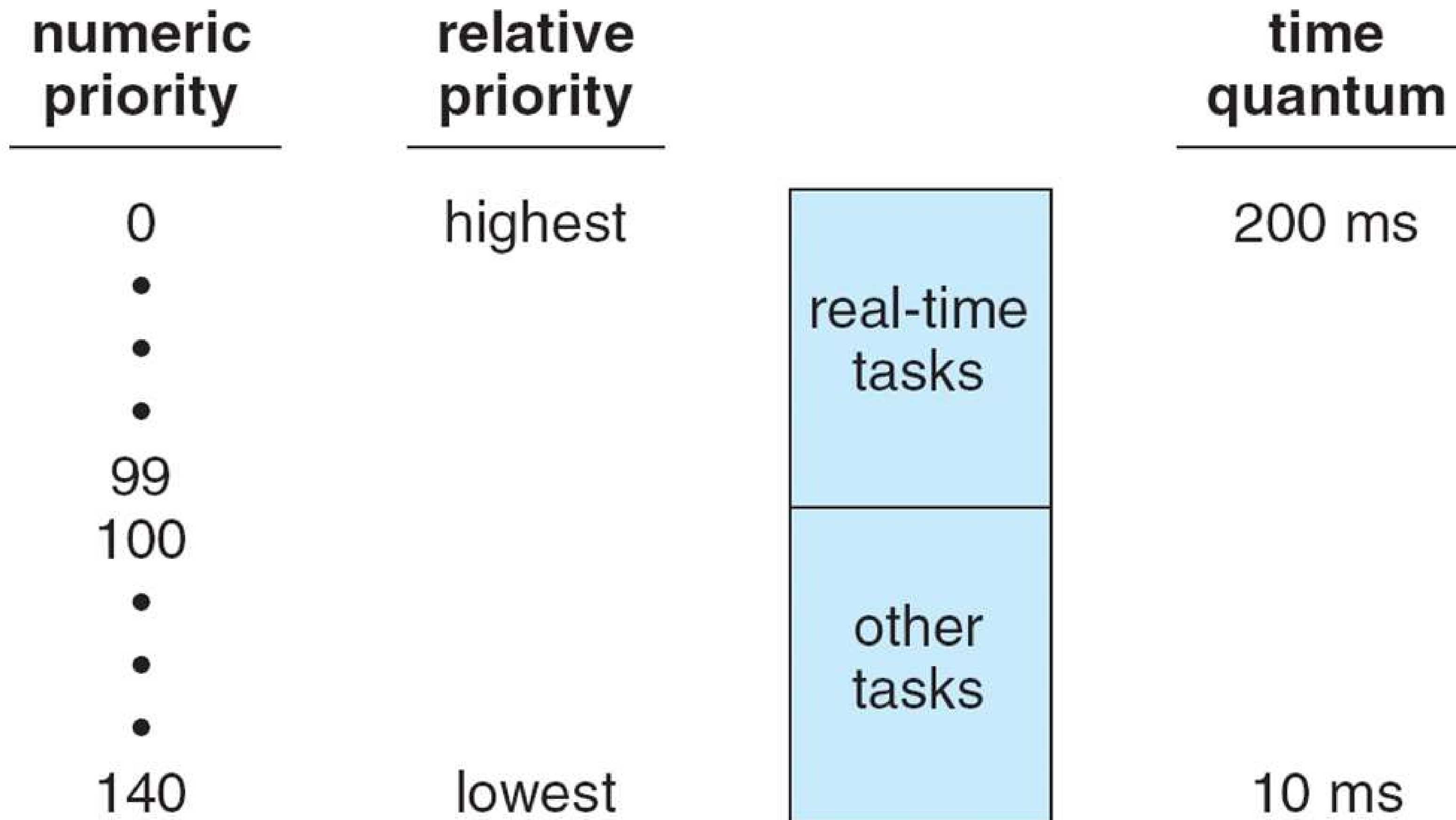
Windows XP Priorities

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

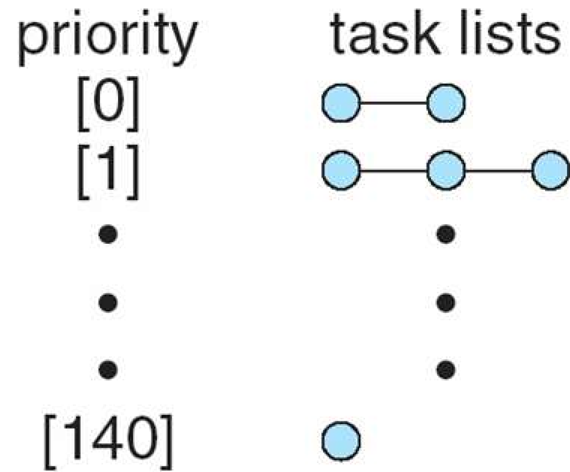
Linux Scheduling

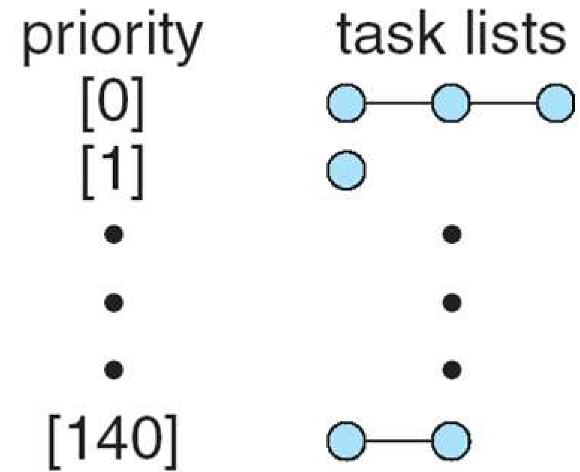
- Constant order $O(1)$ scheduling time
- Two priority ranges: time-sharing and real-time
- **Real-time** range from 0 to 99 and **nice** value from 100 to 140

Priorities and Time-slice length



List of Tasks Indexed According to Priorities





Summary

- Scheduling problem
  - Given a set of processes that are ready to run
  - Which one to select *next*
- Scheduling criteria
  - CPU utilization, Throughput, Turnaround, Waiting, Response
  - Predictability: variance in any of these measures
- Scheduling algorithms
  - FCFS, SJF, SRTF, RR
  - Multilevel (Feedback-)Queue Scheduling
- The best schemes are adaptive.
- To do absolutely best we'd have to predict the future.
  - Most current algorithms give highest priority to those that need the least!
- Scheduling become increasingly ad hoc over the years.
  - 1960s papers very math heavy, now mostly "tweak and see"

Assignments

- 5.2
- 5.4
- 5.5
- MiniProject