



# COS 318: Operating Systems

## CPU Scheduling

Kai Li and Andy Bavier  
Computer Science Department  
Princeton University

<http://www.cs.princeton.edu/courses/archive/fall13/cos318/>



# Quiz 1

---

- ◆ Avg score: 10.7
- ◆ Suggested solutions posted shortly
- ◆ Common issues:
  - Q1 (system call): Forgot to save/restore user context
  - Q2 (context switch): Forgot to flush cache & TLB
  - Q3 (microkernel): Didn't give a disadvantage
  - Didn't submit plain ASCII file
- ◆ Reminder: Quiz 2 due tomorrow @11:55pm



# Today's Topics

---

- ◆ CPU scheduling basics
- ◆ CPU Scheduling algorithms



# When to Schedule?

---

- ◆ Process/thread creation
- ◆ Process/thread exit
- ◆ Blocking on I/O or synchronization
- ◆ I/O interrupt
- ◆ Clock interrupt (pre-emptive scheduling)



# Preemptive vs. Non-Preemptive Scheduling

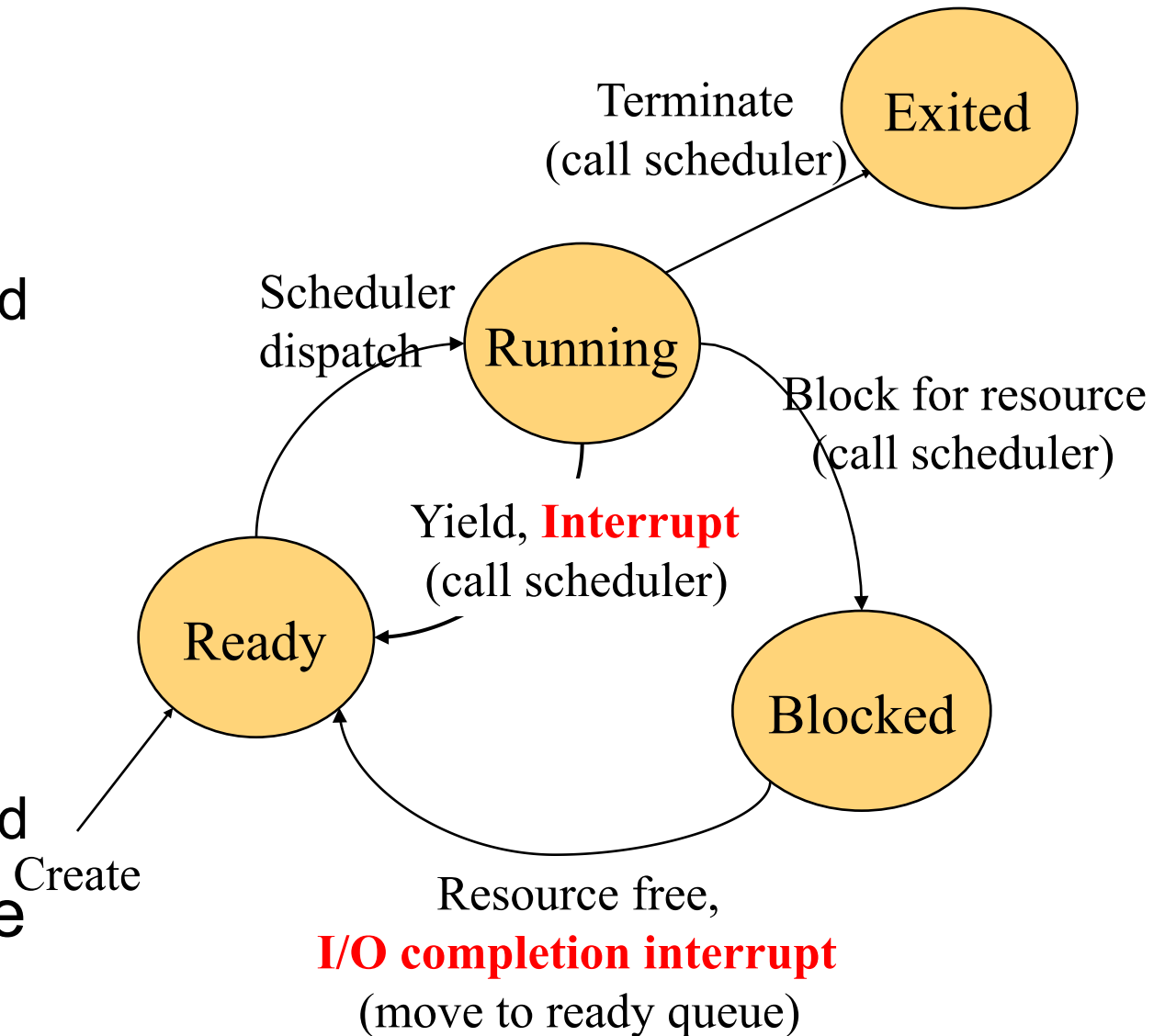
## ◆ Preemptive scheduling

- Running  $\Rightarrow$  ready
- Blocked  $\Rightarrow$  ready
- Running  $\Rightarrow$  blocked
- Terminate

## ◆ Non-preemptive scheduling

- Running  $\Rightarrow$  ready
- Blocked  $\Rightarrow$  ready
- Running  $\Rightarrow$  blocked

## ◆ Batch vs interactive vs real-time



# Scheduling Criteria

---

## ◆ Assumptions

- One program per user and one thread per program
- Programs are independent

## ◆ Goals for batch and interactive systems

- Provide fairness
- Everyone makes some progress; no one starves
- Maximize CPU utilization
  - Not including idle process
- Maximize throughput
  - Operations/second (min overhead, max resource utilization)
- Minimize turnaround time
  - Batch jobs: time to execute (from submission to completion)
- Shorten response time
  - Interactive jobs: time response (e.g. typing on a keyboard)
- Proportionality, or Principle of Least Astonishment
  - Meets user's expectations



# Scheduling Criteria

---

## ◆ Questions:

- What are the goals for PCs versus servers?
- Average response time vs. throughput
- Average response time vs. fairness



# Problem Cases

---

- ◆ Completely blind about job types
  - No CPU and I/O overlap.
- ◆ Optimization involves favoring jobs of type “A” over “B”
  - Lots of A’s? B’s starve.
- ◆ Interactive process trapped behind others
  - Response time bad for no good reason.
- ◆ Priorities: A depends on B and A’s priority  $>$  B’s
  - B never runs.





# Today's Topics

---

- ◆ CPU scheduling basics
- ◆ CPU Scheduling algorithms



# Scheduling Algorithms

---

- ◆ Simplified view of scheduling:
  - Save process state (to PCB)
  - **Pick which process to run next**
  - Dispatch process



# First-Come-First-Serve (FCFS) Policy

## ◆ What does it mean?

- Run to completion (old days)
- Run until blocked or yields

## ◆ Example 1

- P1 = 24sec, P2 = 3sec, and P3 = 3sec, submitted together
- Average response time =  $(24 + 27 + 30) / 3 = 27$



## ◆ Example 2

- Same jobs but come in different order: P2, P3 and P1
- Average response time =  $(3 + 6 + 30) / 3 = 13$



(Gantt Graph)



# STCF and SRTCF

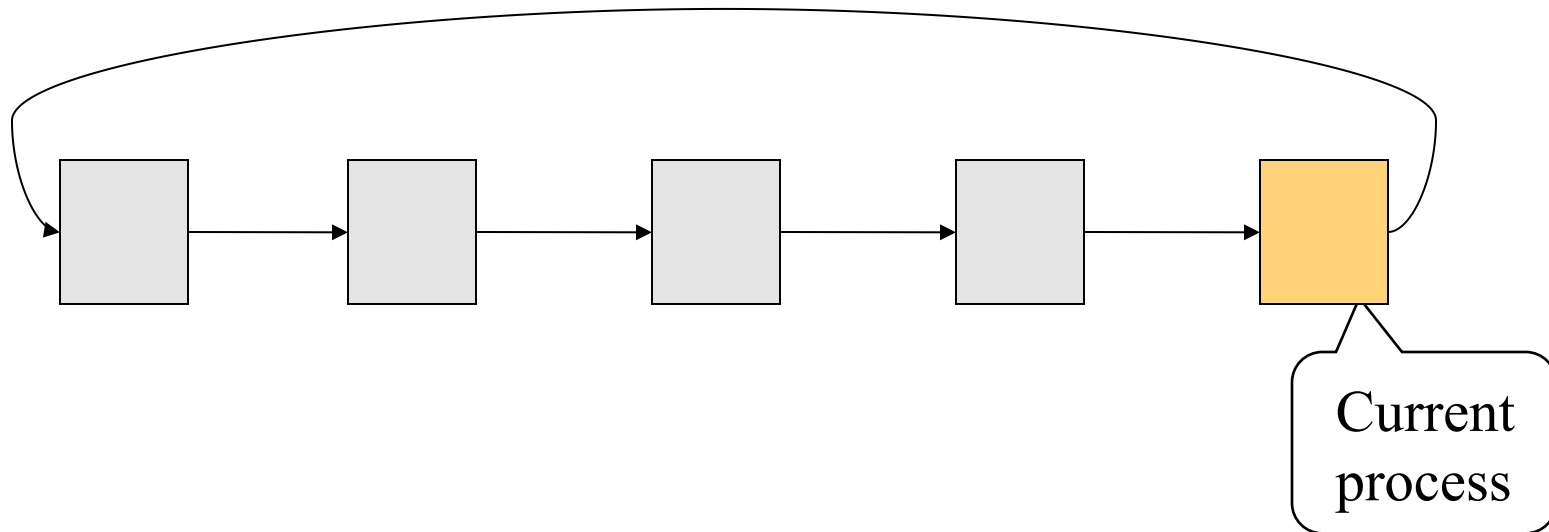
- ◆ Shortest Time to Completion First
  - Non-preemptive
- ◆ Shortest Remaining Time to Completion First
  - Preemptive version
- ◆ Example
  - P1 = 6sec, P2 = 8sec, P3 = 7sec, P4 = 3sec
  - All arrive at the same time



- ◆ Can you do better than SRTCF in terms of average response time?
- ◆ Issues with this approach?



# Round Robin



- ◆ Similar to FCFS, but add a time slice for timer interrupt
- ◆ FCFS for preemptive scheduling
- ◆ Real systems also have I/O interrupts in the mix
- ◆ How do you choose time slice?



# FCFS vs. Round Robin

---

## ◆ Example

- 10 jobs and each takes 100 seconds

## ◆ FCFS (non-preemptive scheduling)

- job 1: 100s, job2: 200s, ... , job10: 1000s

## ◆ Round Robin (preemptive scheduling)

- time slice 1sec and no overhead
- job1: 991s, job2: 992s, ... , job10: 1000s

## ◆ Comparisons

- Round robin is much worse (turnaround time) for jobs about the same length
- Round robin is better for short jobs



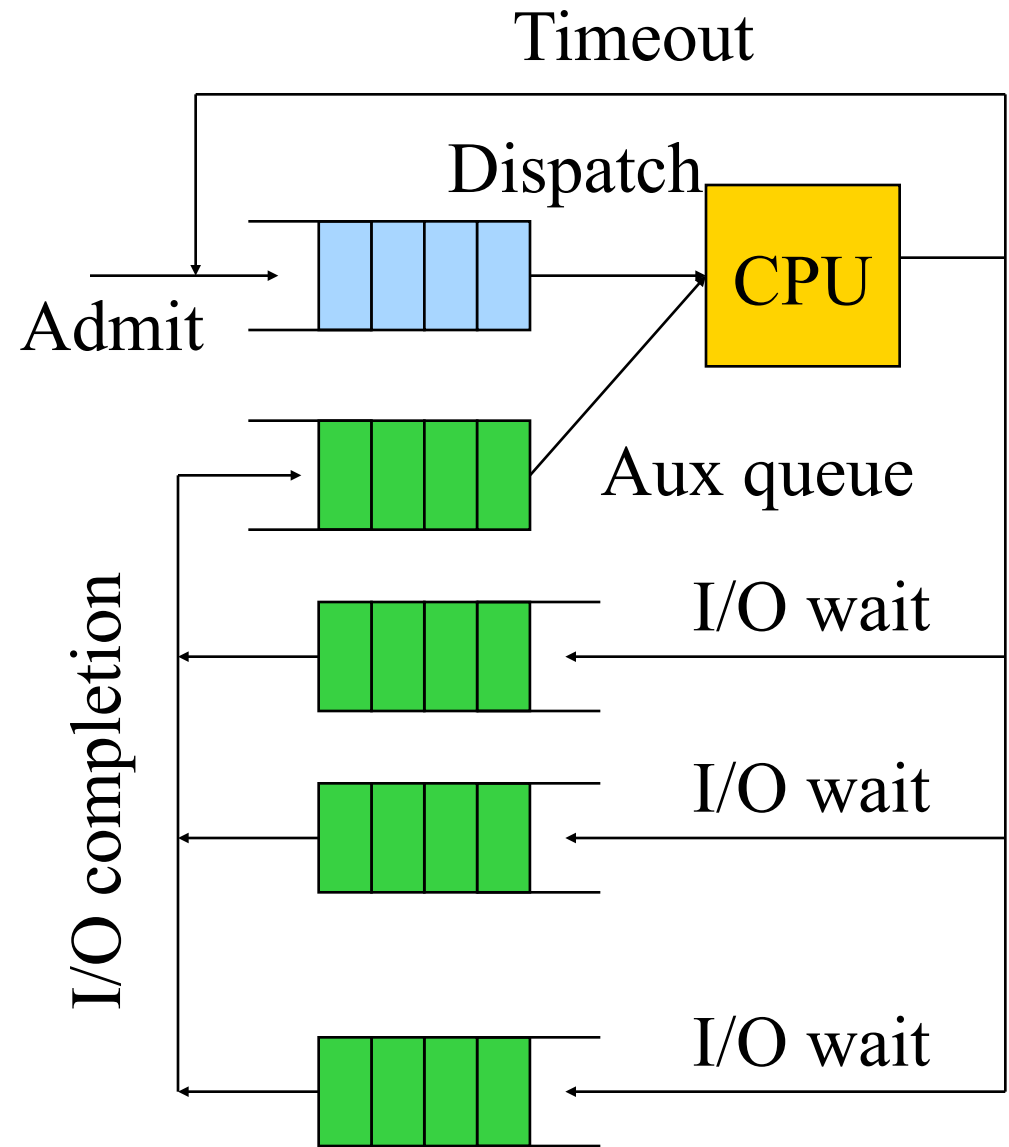
# Resource Utilization Example

- ◆ A, B, and C run forever (in this order)
  - A and B each uses 100% CPU forever
  - C is a CPU plus I/O job (1ms CPU + 10ms disk I/O)
- ◆ Time slice 100ms
  - A (100ms CPU), B (100ms CPU), C (1ms CPU + 10ms I/O),  
...
- ◆ Time slice 1ms
  - A (1ms CPU), B (1ms CPU), C (1ms CPU),  
A (1ms CPU), B (1ms CPU), C(10ms I/O) || A, B, ..., A, B
- ◆ What do we learn from this example?



# Virtual Round Robin

- ◆ Aux queue is FIFO
- ◆ I/O bound processes go to aux queue (instead of ready queue) to get scheduled
- ◆ Aux queue has preference over ready queue





# Priority Scheduling

## ◆ Obvious

- Not all processes are equal, so rank them

## ◆ The method

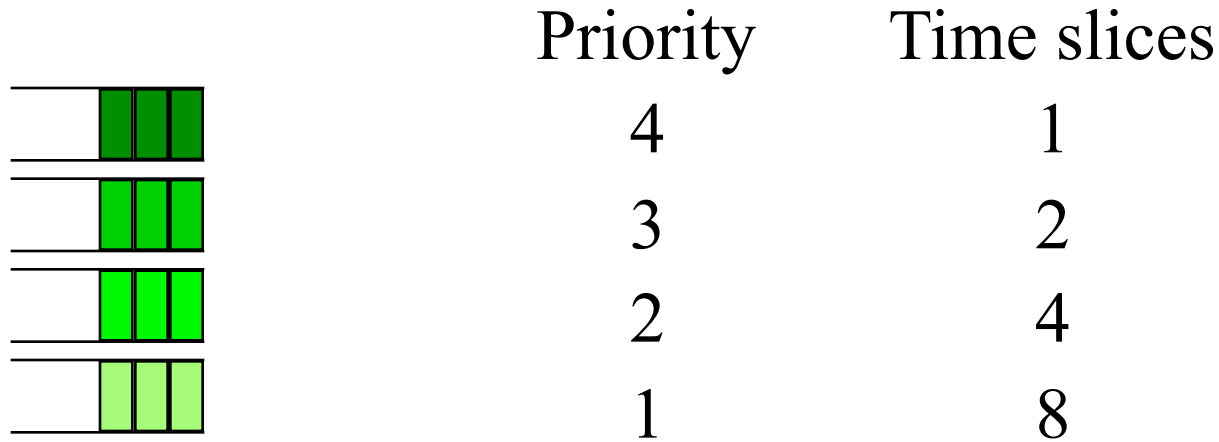
- Assign each process a priority
- Run the process with highest priority in the ready queue first
- Adjust priority dynamically (I/O wait raises the priority, reduce priority as process runs)

## ◆ Why adjusting priorities dynamically

- T1 at priority 4, T2 at priority 1 and T2 holds lock L
- Scenario
  - T1 tries to acquire L, fails, blocks.
  - T3 enters system at priority 3.
  - T2 never gets to run!



# Multiple Queues



- ◆ Jobs start at highest priority queue
- ◆ If timeout expires, drop one level
- ◆ If timeout doesn't expires, stay or pushup one level
  
- ◆ What does this method do?



# Interlude

- ◆ What to do when you hit a brick wall?
- ◆ Meditate? Google thinks so...
  - <http://www.wired.com/business/2013/06/meditation-mindfulness-silicon-valley/all/>



For one who has conquered the mind,  
The mind is the best of friends,  
But for one who has failed to do so,  
His very mind will be the greatest enemy.  
-- Bhagavad-Gita

# Lottery Scheduling

---

## ◆ Motivations

- SRTCF does well with average response time, but unfair

## ◆ Lottery method

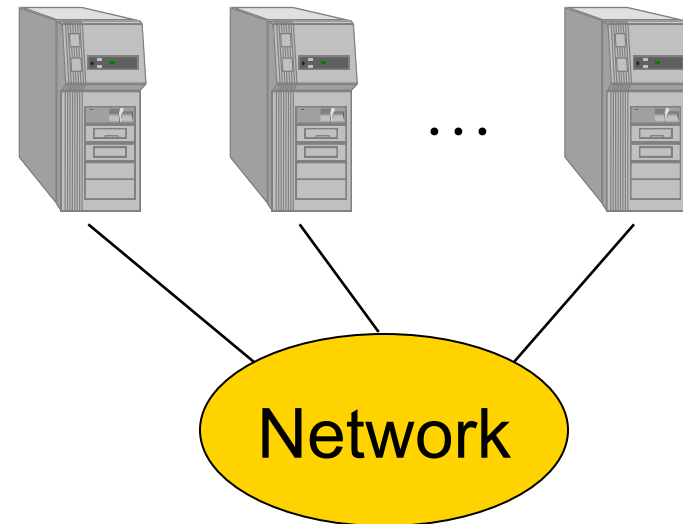
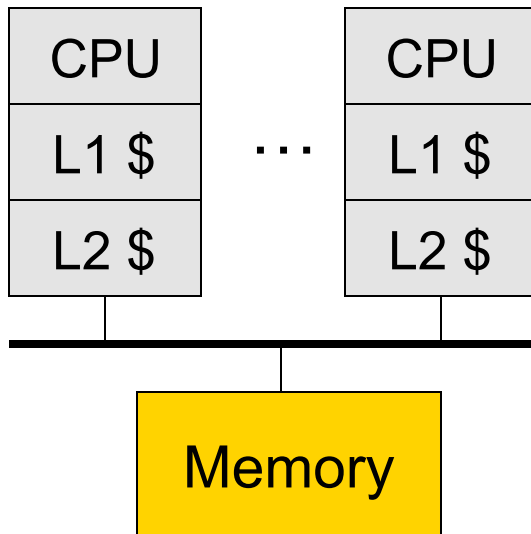
- Give each job a number of tickets
- Randomly pick a winning tickets
- To approximate SRTCF, give short jobs more tickets
- To avoid starvation, give each job at least one ticket
- Cooperative processes can exchange tickets

## ◆ Question

- How do you compare this method with priority scheduling?



# Multiprocessor and Cluster



## Multiprocessor architecture

- ◆ Cache coherence
- ◆ Single OS

## Cluster or multicomputer

- ◆ Distributed memory
- ◆ An OS in each box

# Multiprocessor/Cluster Scheduling

---

- ◆ Design issue
  - Process/thread to processor assignment
- ◆ Gang scheduling (co-scheduling)
  - Threads of the same process will run together
  - Processes of the same application run together
- ◆ Dedicated processor assignment
  - Threads will be running on specific processors to completion
  - Is this a good idea?



# Real-Time Scheduling

## ◆ Two types of real-time

- Hard deadline
  - Must meet, otherwise can cause fatal error
- Soft Deadline
  - Meet most of the time, but not mandatory

## ◆ Admission control

- Take a real-time process only if the system can guarantee the “real-time” behavior of all processes
- The jobs are schedulable, if the following holds:

$$\sum \frac{C_i}{T_i} \leq 1$$

where  $C_i$  = computation time, and  $T_i$  = period



# Rate Monotonic Scheduling (Liu & Layland 73)

## ◆ Assumptions

- Each periodic process must complete within its period
- No process is dependent on any other process
- Each process needs the same amount of CPU time on each burst
- Non-periodic processes have no deadlines
- Process preemption occurs instantaneously (no overhead)

## ◆ Main ideas of RMS

- Assign each process a fixed priority = frequency of occurrence
- Run the process with highest priority
- Only works if CPU utilization is not too high

## ◆ Example

- P1 runs every 30ms gets priority 33 (33 times/sec)
- P2 runs every 50ms gets priority 20 (20 times/sec)





# Earliest Deadline First Scheduling

## ◆ Assumptions

- When a process needs CPU time, it announces its deadline
- No need to be periodic process
- CPU time needed may vary

## ◆ Main idea of EDF

- Sort ready processes by their deadlines
- Run the first process on the list (earliest deadline first)
- When a new process is ready, it preempts the current one if its deadline is closer
- Provably optimal

## ◆ Example

- P1 needs to finish by 30sec, P2 by 40sec and P3 by 50sec
- P1 goes first
- More in MOS 7.5.3-7.5.4



## 4.3 BSD Scheduling with Multi-Queue

- ◆ “1 sec” preemption
  - Preempt if a process doesn't block or complete within 1 second
- ◆ Priority is recomputed every second
  - $P_i = \text{base} + (\text{CPU}_{i-1}) / 2 + \text{nice}$ , where  $\text{CPU}_i = (U_i + \text{CPU}_{i-1}) / 2$
  - Base is the base priority of the process
  - $U_i$  is process utilization in interval  $i$
- ◆ Priorities
  - Swapper
  - Block I/O device control
  - File operations
  - Character I/O device control
  - User processes



# Linux Scheduling

## ◆ Time-sharing scheduling

- Two priority arrays: active and expired
- 40 priority levels, lower number = higher priority
- Priority = base (user-set) priority + “bonus”
  - Bonus between -5 and +5, derived from *sleep\_avg*
  - Bonus decremented when task sleeps, incremented when it runs
  - Higher priority gets longer timeslice
- Move process with expired quantum from active to expired
- When active array empty, swap active and expired arrays

## ◆ Real-time scheduling

- 100 static priorities, higher than time sharing priorities
- Soft real-time



# Windows Scheduling

## ◆ Classes and priorities

- Real time: 16 static priorities
- User: 16 variable priorities, start at a base priority
  - If a process has used up its quantum, lower its priority
  - If a process waits for an I/O event, raise its priority

## ◆ Priority-driven scheduler

- For real-time class, do round robin within each priority
- For user class, do multiple queue

## ◆ Multiprocessor scheduling

- For N processors, normally run N highest priority threads
- Threads have hard or soft affinity for specific processors
- A thread will wait for processors in its affinity set, if there are other threads available (for variable priorities)



# Summary

---

- ◆ Different scheduling goals
  - Depend on what systems you build
- ◆ Scheduling algorithms
  - Small time slice is important for improving I/O utilization
  - STCF and SRTCF give the minimal average response time
  - Priority and its variations are in most systems
  - Lottery is flexible
  - Real-time depends on admission control

