CPU Scheduling (cont)

ECE469, Feb 9

Yiying Zhang

Readings

- Dinosaur Chapter 5
- Comet Chapter 7, 8, 9

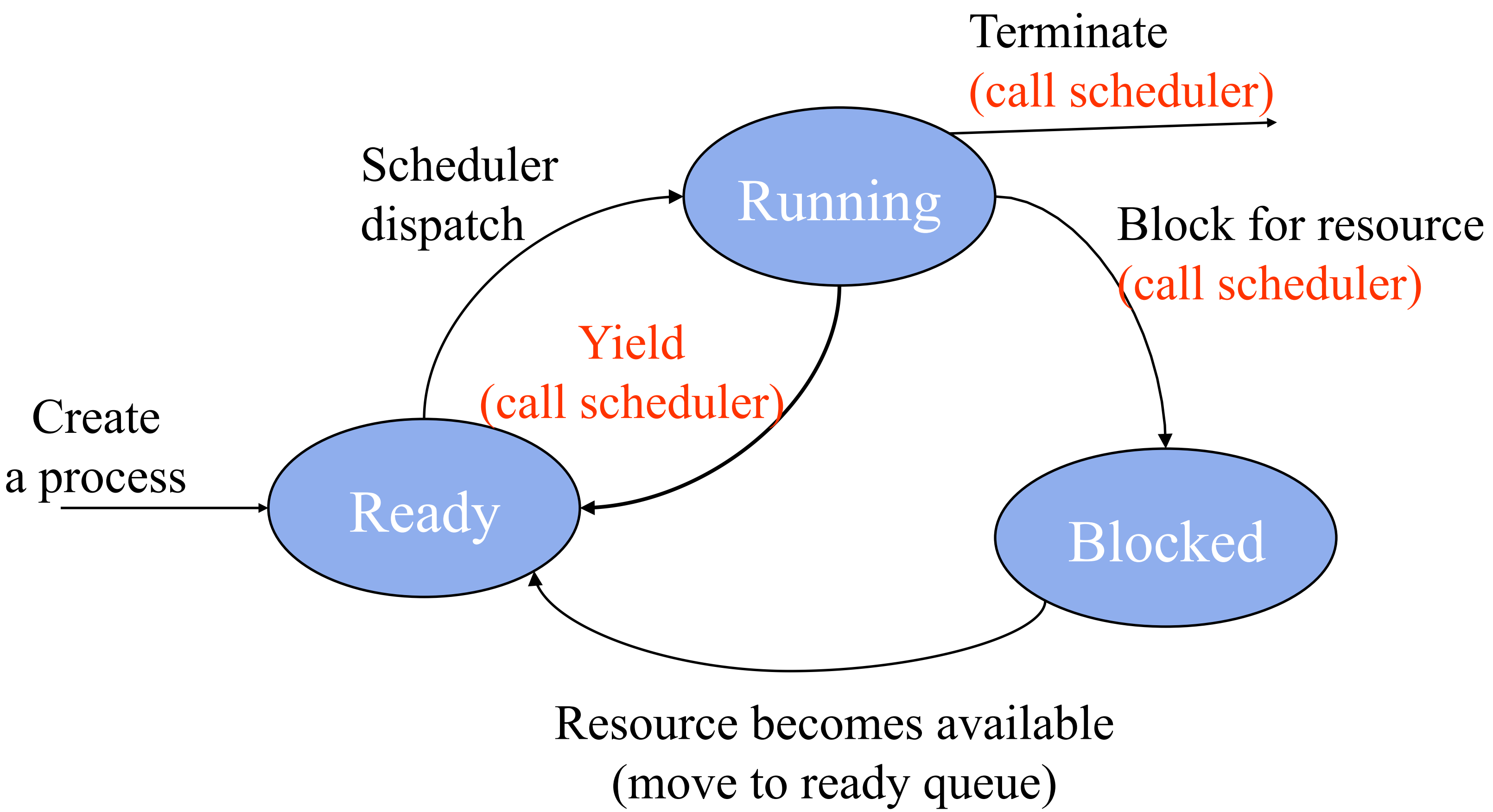
True or false?

- “A CPU scheduling algorithm that minimizes avg turnaround time cannot lead to starvation.”
- “Among all CPU scheduling algorithms, Round Robin always gives the worse average turnaround time.”

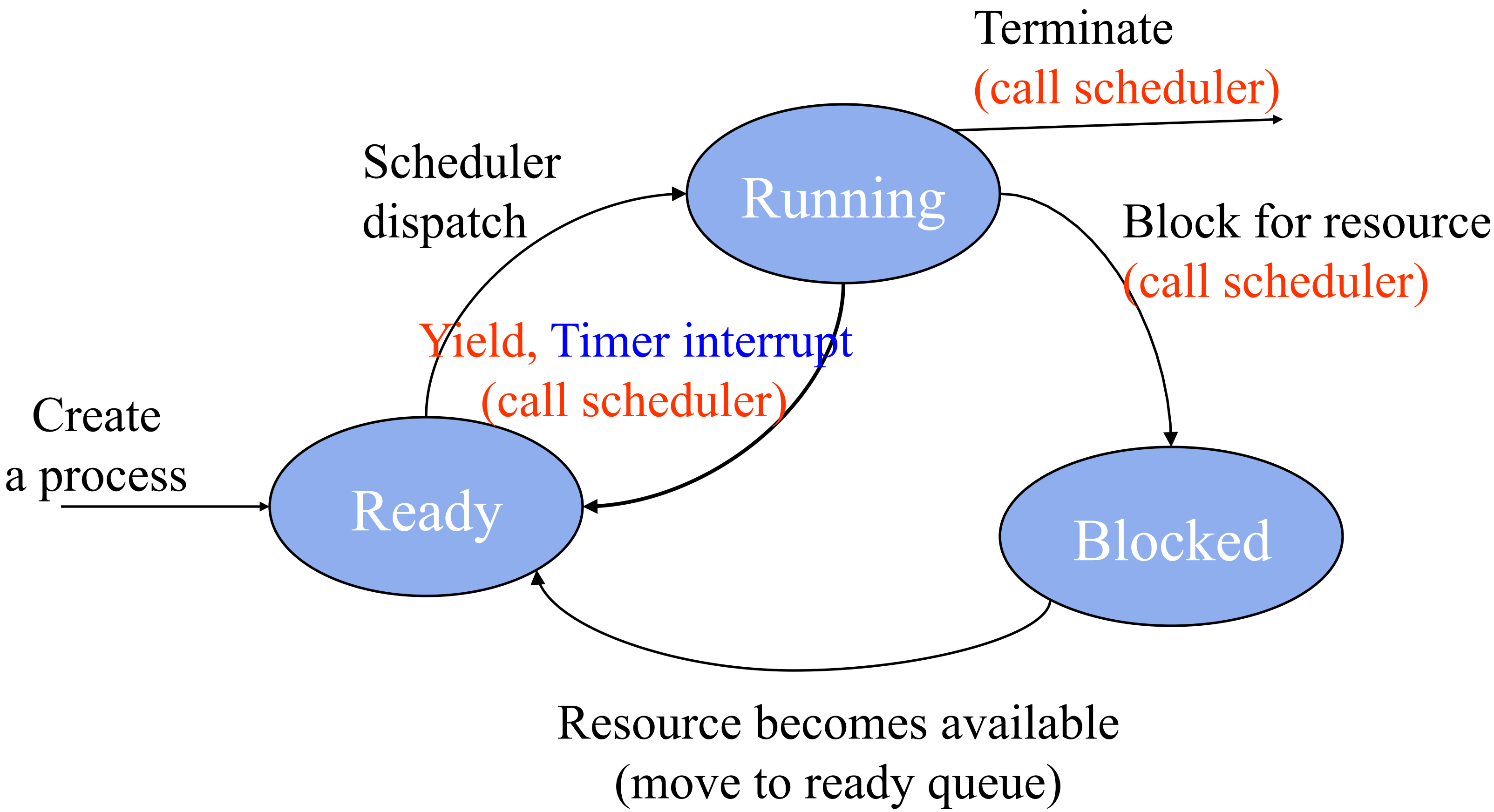
Timesharing Systems

- Timesharing systems support interactive use
  - each user feels he/she has the entire machine
    - CPU, memory, I/O devices
- CPU - How?
  - optimize response time
  - based on time-slicing

Process State Transition of Non-Preemptive Scheduling



Preemptive Scheduling



Review on CPU scheduling

- Mechanism is easy, policy is hard
  - Jobs have diverse characteristics
  - 4 performance metrics
  - Don’t know about future

- Hard to analyze even when narrowing down metric/job nature

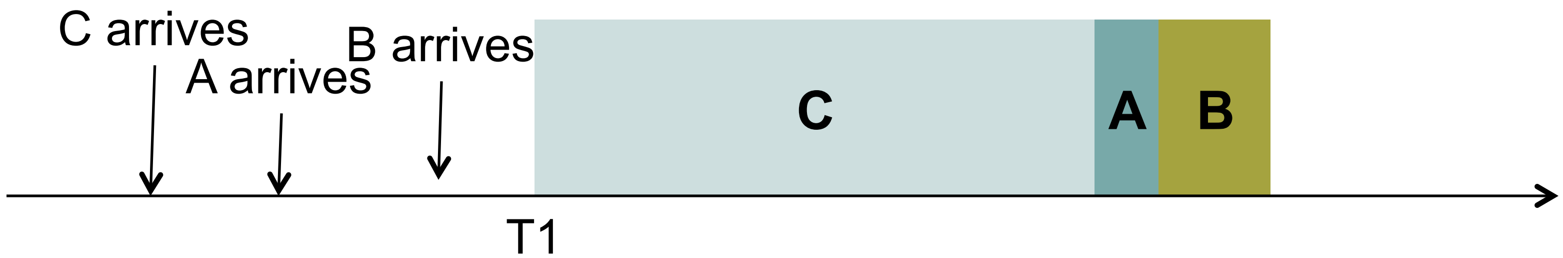
[lec 9] Scheduling policies

- FIFO
- Round Robin
- SJCF
- SRTCF

  - FIFO vs. Round Robin in average turnaround time
    - 10 procs, 100 sec each, who wins?
    - 10 procs, 100 sec for first, 10 sec other 9, who wins?

STCF (SJF) – Shortest Job First

- What shall we do if we care about turn-around time?
  - FIFO can be bad



- STCF/SJF
  - schedule shortest (total completion time) job first

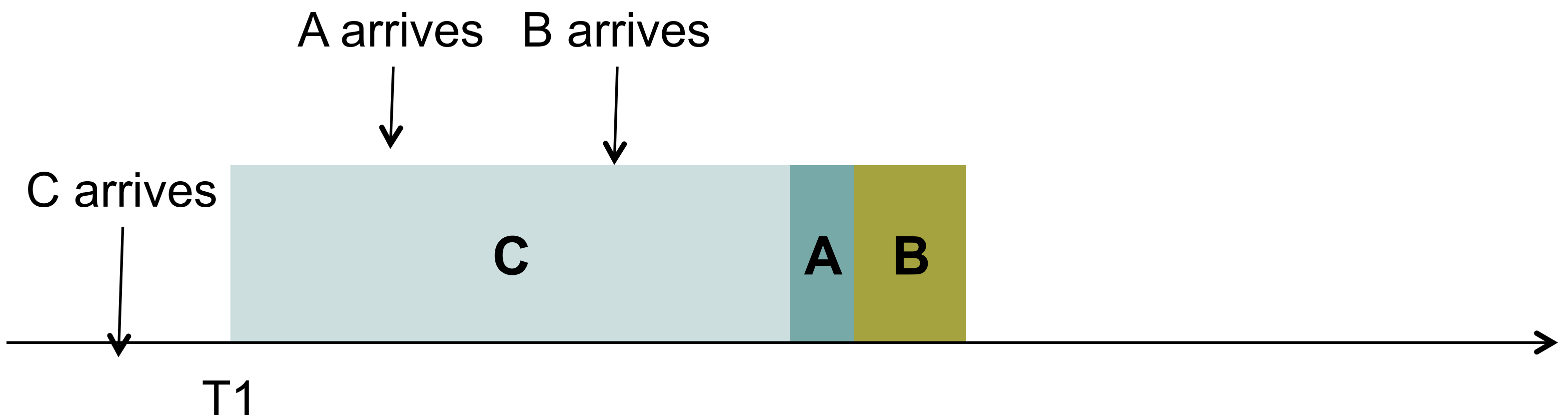
SJF – Pros and Cons

- Can we do better than Shortest Job First in terms of average turnaround time?
  - Assume all jobs arrive at the beginning
- In fact, SJF can be proved to be the optimal scheduling algorithm with the above assumption
  - But we are not going to prove it, since this is not a theory class ☺

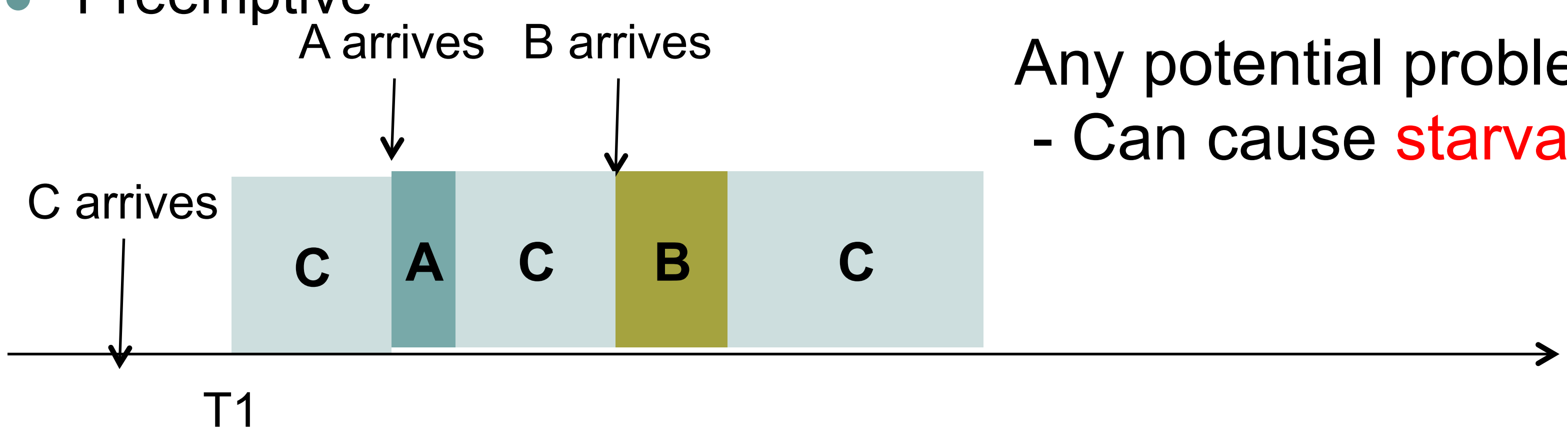
- SJF Advantage
  - Minimal average turnaround time
- Disadvantage
  - Difficult to know the future, has to run until finish

STCF vs. SRTCF

- Shortest time to completion first (shortest job first)
  - Non-preemptive



- Shortest *remaining* time to completion first
  - Preemptive

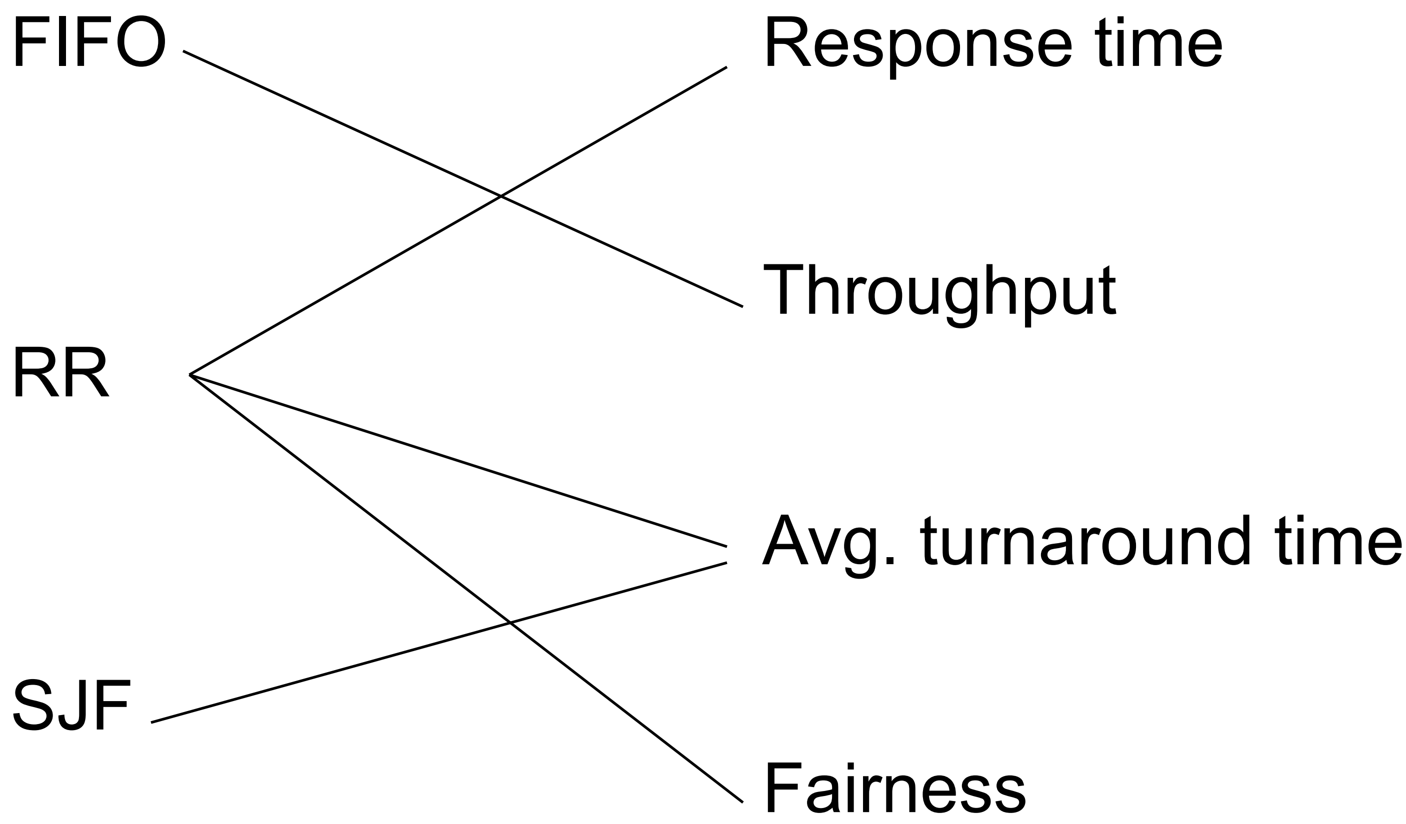


Any potential problems?
- Can cause starvation!

Observations so far

- Need to accommodate interactive jobs
  - Need some kind of RR
- Diversity in jobs – job length, I/O mix
  - RR also appears to help
- SJF also has virtue
  - Reduce avg. turnaround time
- Can we accommodate all?

Scheduling policies



What Issues are in Policy?

- High utilization (efficiency)
  - Lots of processes (want diff resources)
  - Lots of resources (want full parallelism)

- Issue?
  - How do you get the most ***useful*** work out of the system? (job throughput)

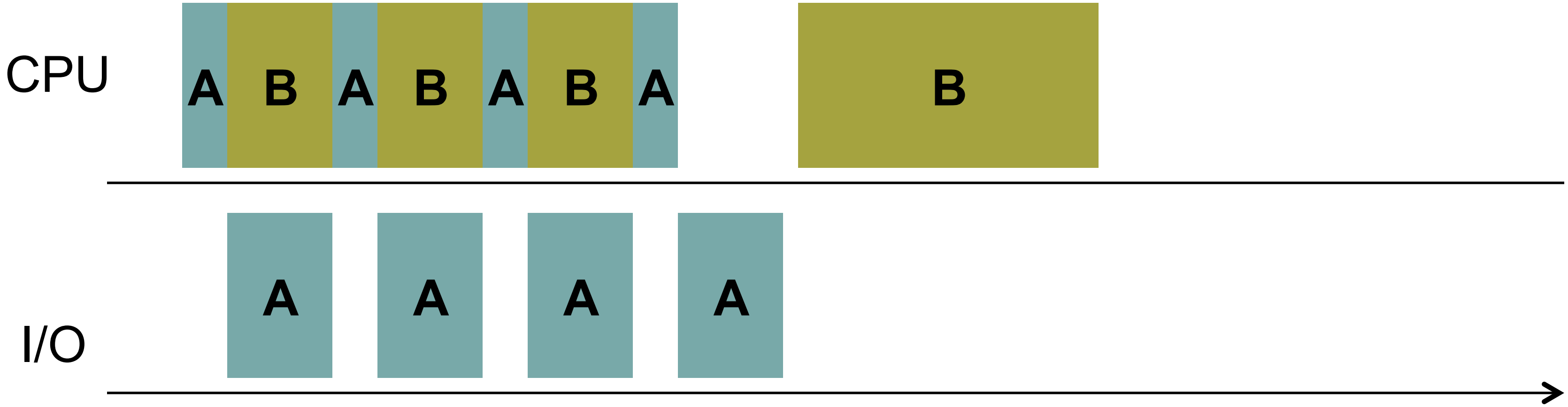
What Issues are in Policy?

- Fairness
- Flexibility
- High utilization (efficiency)
- Good response time
- Good turnaround time

Other issues?

- Smooth media playback, device handling

What about I/Os?

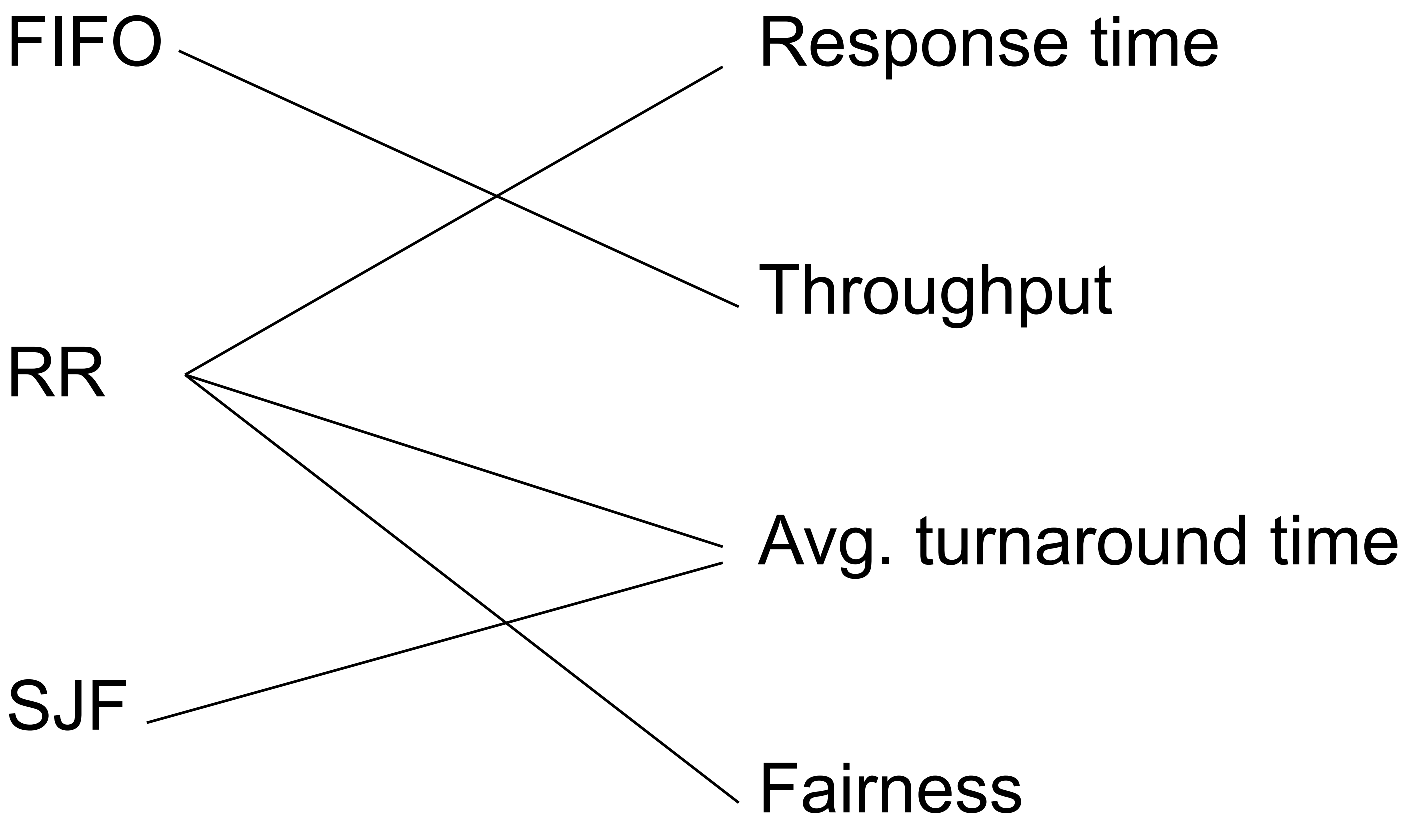


Adding I/O Into the Mix

- Resource utilization example
  - A and B each uses 100% CPU
  - C loops forever (1ms CPU and 10ms disk)
  - Time slice 99ms: nearly 30% of disk utilization with Round Robin and nearly 70% of CPU utilization
  - Time slice 1ms: nearly 90% of disk utilization with Round Robin and nearly 100% of CPU utilization

- What do we learn from this example?
  - *Small time slice can improve utilization / fairness to I/O jobs*

Scheduling policies



Priority Scheduling

- To accommodate the spirits of SJF/RR/FIFO
- The method
  - Assign each process a *priority*
  - Run the process with highest priority in ready queue first
    - Use FIFO for processes with equal priority
  - Adjust priority dynamically
    - To deal with *all* issues: e.g. aging, I/O wait raises priority

- Advantage
  - Flexibility: Not all processes are “born” equal
- Challenge?

Priority Scheduling (cont)

- Who sets the priorities
  - Internally by OS
    - I/O to computation ratio (can be dynamic)
    - Memory requirement (can be dynamic)
    - Time constraints (e.g. real-time systems)
  - Externally by users/sysadm
    - Importance
    - Funds paid for
    - Being nice
- How -- Dynamically adjustment is tricky

Break

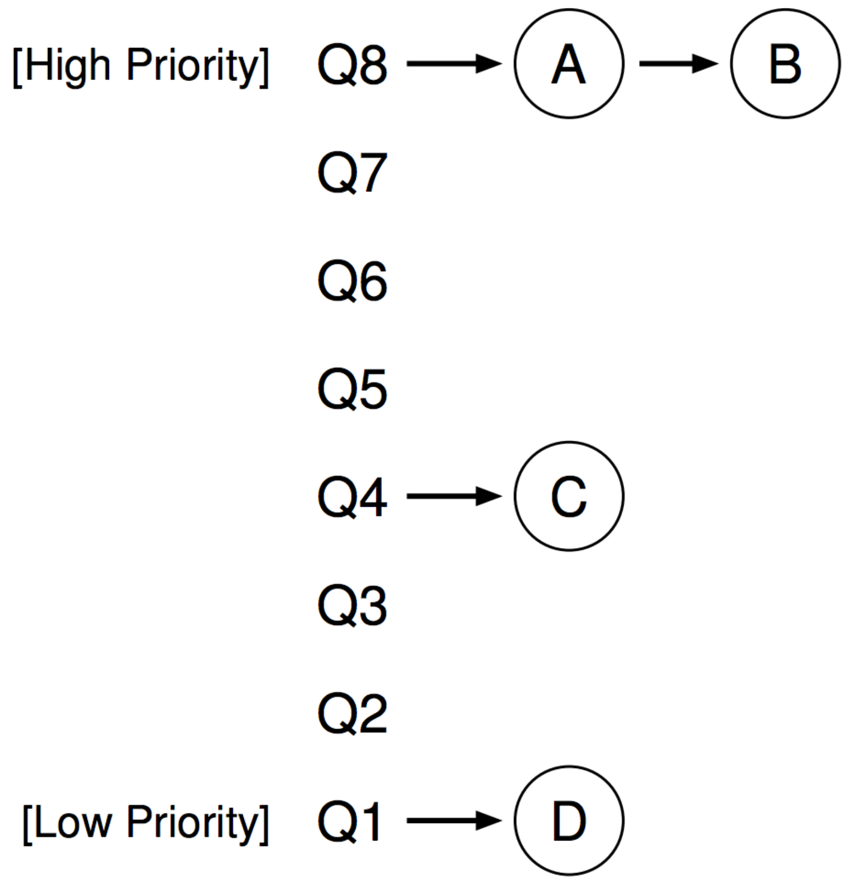
- Brain teaser time
  - There is a person who have two numbers, he tells sum to the person S and product of those numbers to P. Now there is this conversion between S and P.
    - S: I don’t know what are the numbers.
    - P: I also don’t know what are the numbers.
    - S: Now I know what are the numbers.
    - P: Now I also know what are the numbers.
  - Assuming S and P to be very wise and good in mathematics, What are those two numbers? Note: Numbers are greater than 0.

Approach 1: Multiple Queue Scheduling

- Motivation: processes may be of different nature and can be easily classified
  - e.g. foreground jobs vs. background jobs

- The method:
  - Processes permanently assigned to one queue, based on processes priority / type
    - Preference to jobs with higher priorities
  - Each queue can have its own scheduling algorithm
    - e.g. RR for foreground queue, FCFS for background queue
  - Need a scheduling among the queues
    - e.g. fixed priority preemptive scheduling (high-pri queue trumps other)
    - e.g. time-slice between queues

Multiple Queue Example



Pros/Cons of Multiple Queue Scheduling

- Pros:
  - Low scheduling overhead
    - Jobs do not move across queues
- Cons:
  - Processes permanently assigned to one queue – not flexible
    - Program behavior may change
    - E.g. can switch between I/O bound and CPU bound

    → Need some learning/adaptation at runtime
  - Starvation cannot be easily handled

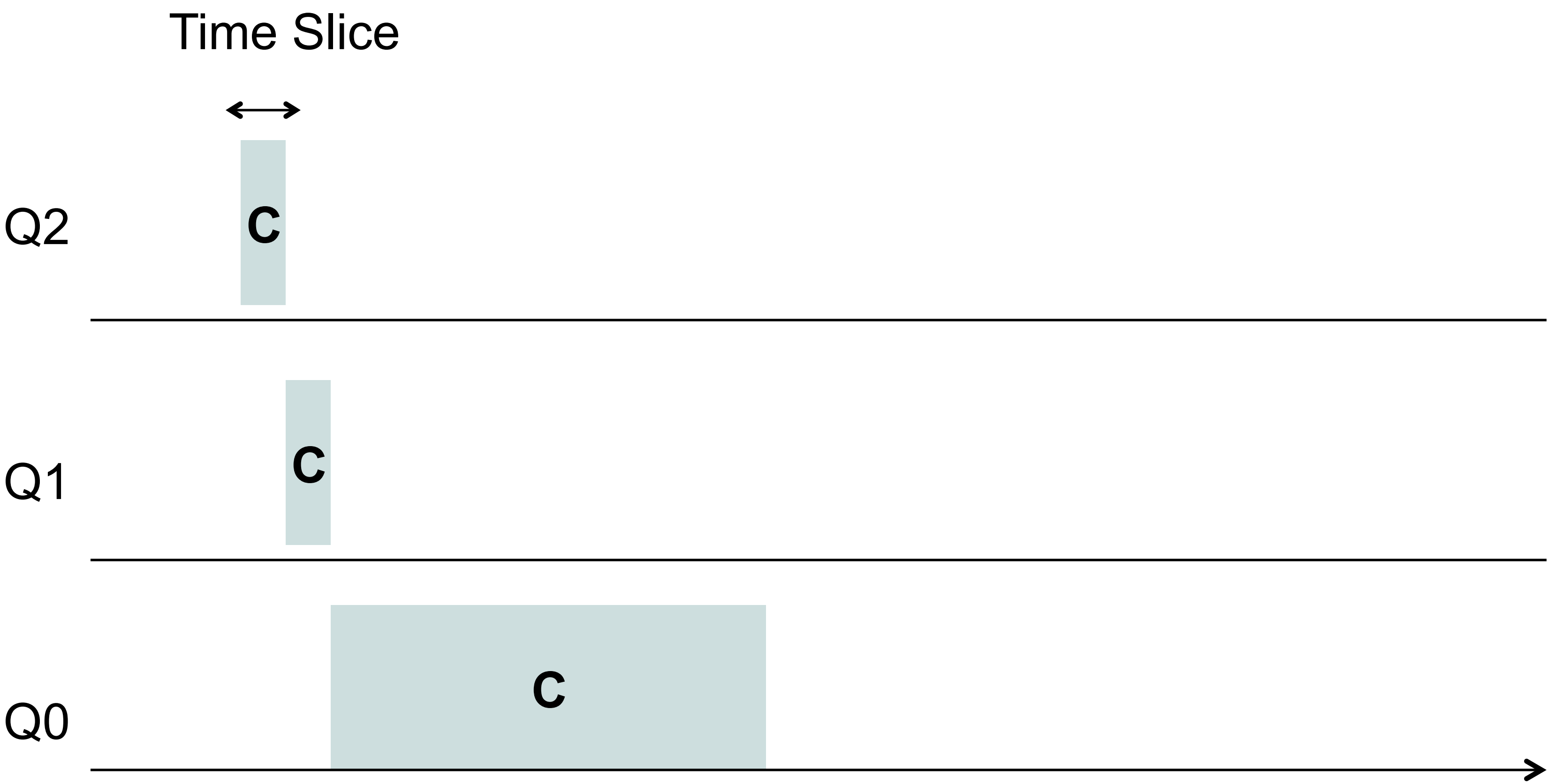
    → Need some learning/adaptation at runtime

Approach 2: Multilevel Feedback Queue (MLFQ)

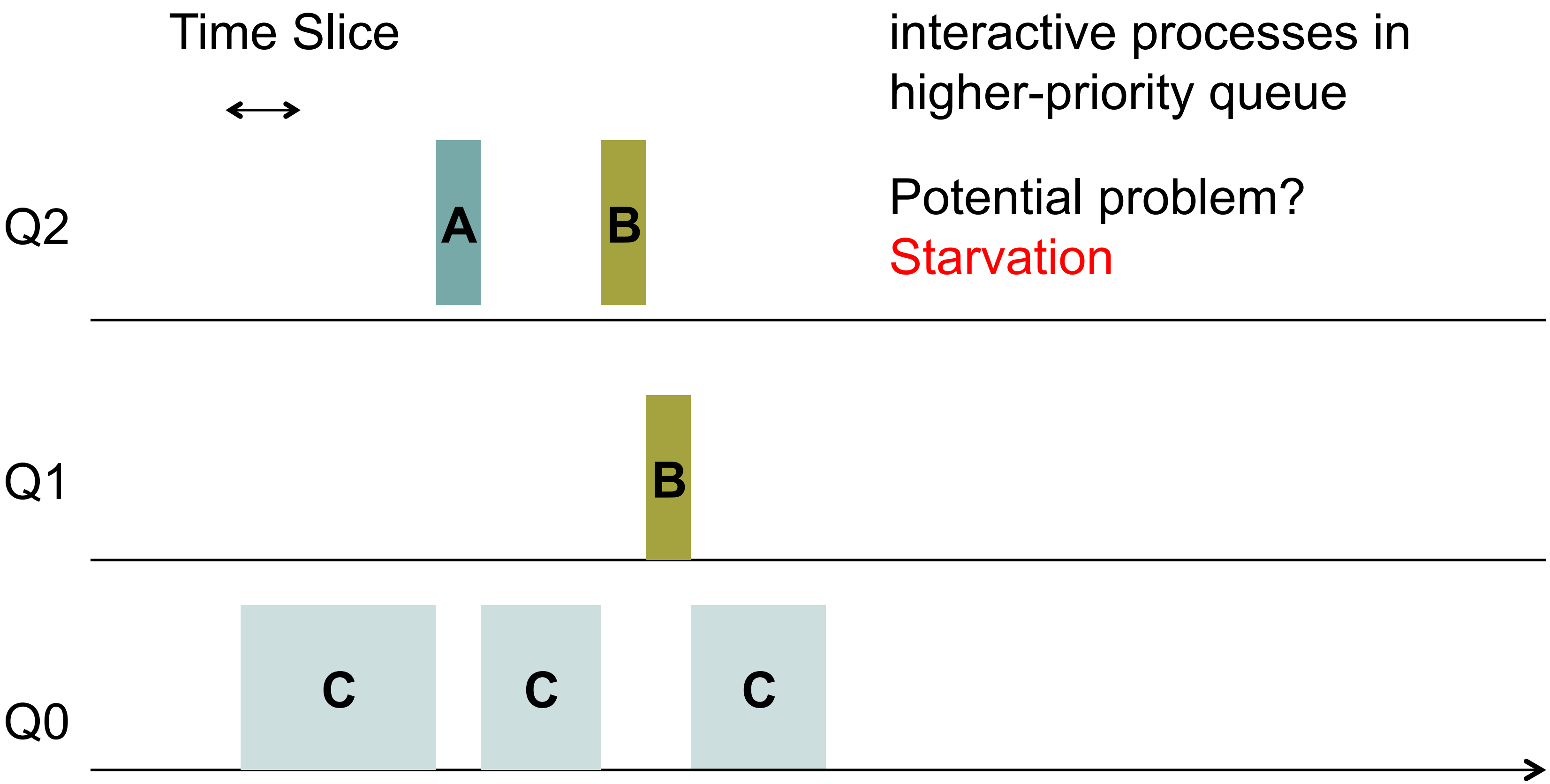
- Problem: how to change priority?

- Jobs start at highest priority queue
- Feedback
  - If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue).
  - If a job gives up the CPU before the time slice is up, it stays at the same priority level.
  - After a long time period, move all the jobs in the system to the topmost queue (aging)

MLFQ Example – Single long job



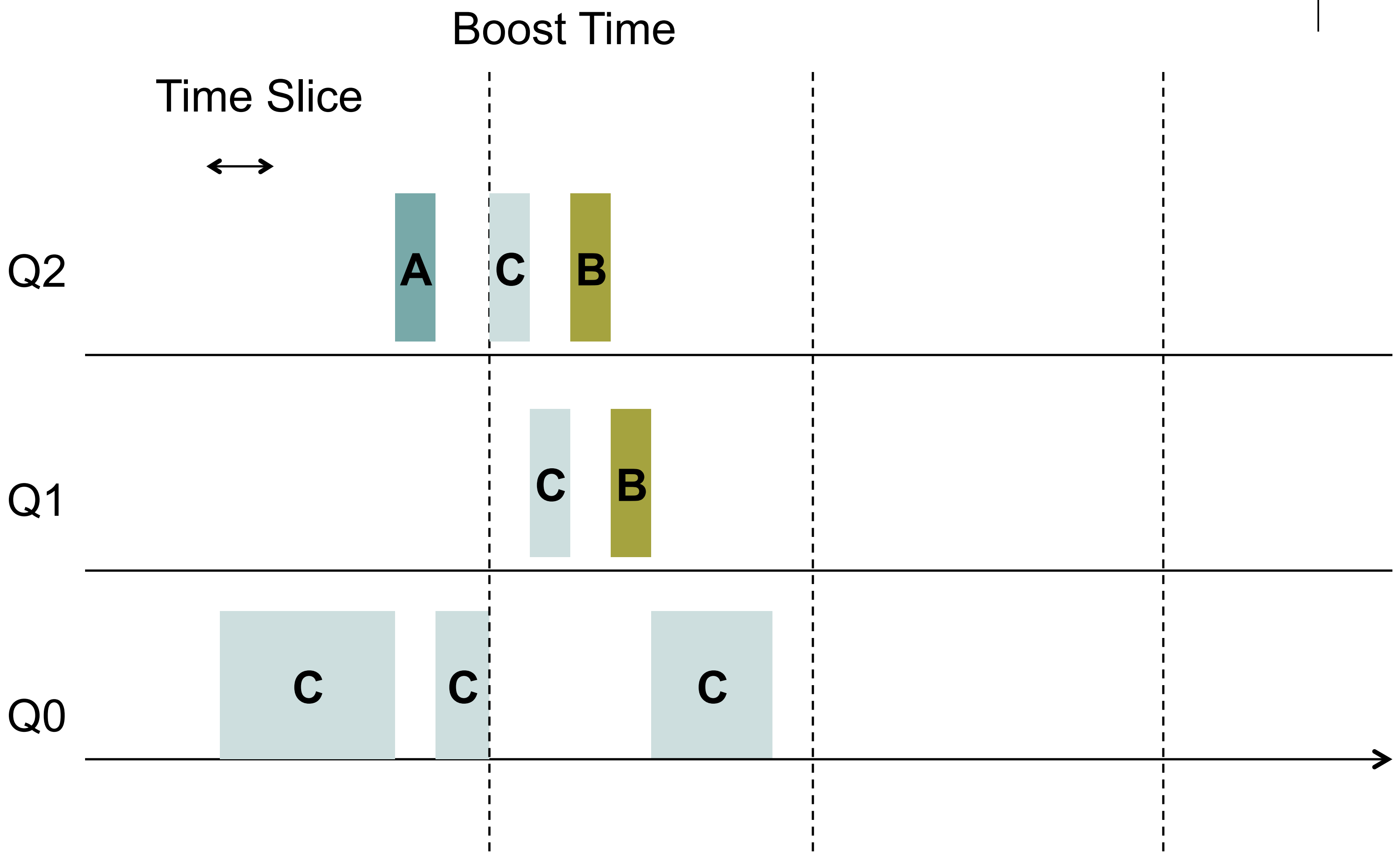
MLFQ Example – a long job + short jobs in between



Leave I/O bound and interactive processes in higher-priority queue

Potential problem?
Starvation

MLFQ Example – a long job +short jobs, with boost



Scheduling Algorithms in OSes

Operating System	Preemption	Algorithm
Windows 3.1x	None	Cooperative Scheduler
Windows 95, 98, Me	Half	Preemptive for 32-bit processes, Cooperative Scheduler for 16-bit processes
Windows NT (2000, XP, Vista, 7, and Server)	Yes	Multilevel feedback queue
Mac OS pre-9	None	Cooperative Scheduler
Mac OS 9	Some	Preemptive for MP tasks, Cooperative Scheduler for processes and threads
Mac OS X	Yes	Multilevel feedback queue
Linux pre-2.6	Yes	Multilevel feedback queue
Linux 2.6-2.6.23	Yes	O(1) scheduler
Linux post-2.6.23	Yes	Completely Fair Scheduler
Solaris	Yes	Multilevel feedback queue
NetBSD	Yes	Multilevel feedback queue
FreeBSD	Yes	Multilevel feedback queue

Cooperative Scheduling

- Early multitasking systems used applications that voluntarily ceded time to one another.
- This approach, which was eventually supported by many OSes, is known today as cooperative multitasking.
- Cooperative multitasking was once the scheduling scheme employed by Microsoft Windows (prior to Windows 95 and Windows NT) and Mac OS (prior to Mac OS X) ito enable multiple applications to be run simultaneously
- Now rarely used in larger systems

Let’s look at fairness again

- Proportional share: another view of fairness
  - Each job gets a (fair) proportional of CPU time
  - Goals here are not turnout time or response time

- How to share CPU proportionally?
  - Idea: proportional => probabilistic

Lottery Scheduling [OSDI 94]

- Motivations
  - SJF does well with avg turnaround time, but unfair
  - Priority scheduling is implemented by adjusting priorities, adjusting priority is a bit ad hoc.

- Lottery method: using probabilistic to assign CPU time
  - Give each job a number of tickets
  - Randomly pick a winning tickets => jobs with more tickets have higher chance to win (get CPU)
  - To approximate priority scheduling, high priority jobs get more tickets
  - To approximate SRTCF, short jobs get more tickets
  - To avoid starvation, give each job at least one ticket

Best thing about lottery scheduling

- Easy to implement!

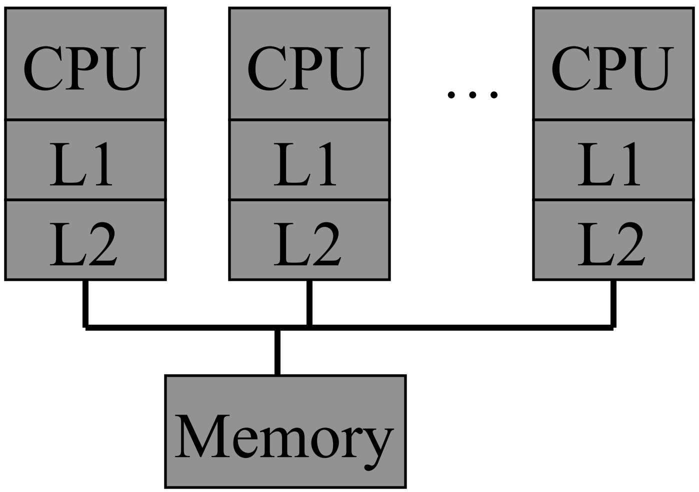
Real-Time Scheduling

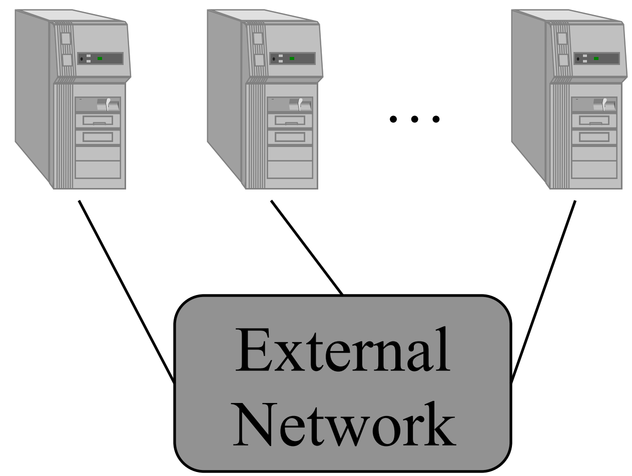
- Two types of real-time
  - Hard deadline: must meet, otherwise can cause fatal error
  - Soft headline: meet most of the time, but not mandatory
- Characteristics
  - User control: provide users with abilities to control and specify
  - Deterministic: upper bound on when to get services on an I/O
  - Responsive: how long does OS delay before ack an interrupt

Deadline Scheduling

- Admission control
  - Take a job only if the system can guarantee real-time
- Information needed
  - Ready time: time at which task becomes ready
  - Starting deadline: time by which a task must begin
  - Completion deadline: time by which a task must complete
  - Processing time: time required to execute the task to completion
  - Resource requirements
  - Priority
  - Subtask structure

Multiprocessor and Cluster





Multiprocessor architecture
- L2 cache coherence
- A single "image" OS

Cluster/Multicomputer
- Distributed memory
- An OS on each box

Multiprocessor/Cluster Scheduling

- New design issue: process/thread inter-dependence
  - Threads of the same process may synchronize
  - Processes of the same job may send/recv messages

Multiprocessor/Cluster Scheduling: Example Approach

- Gang scheduling (coscheduling)
  - Threads of same process will run together on multiprocessor
  - Processes of same application run together on cluster
- Dedicated processor assignment
  - Threads will be running on specific processors to completion
  - Pros / cons?
    - Good for reducing cache misses
    - Bad for load balance / fairness

Case Study: Windows NT Scheduling

- Classes and priorities
  - Real time: 16 static priorities
  - Variable: 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 variable class, multiple queue feedback
- Multiprocessor scheduling
  - For N processors, run N-1 highest priority threads on N-1 processors and run remaining threads on a single processor
  - A thread will wait for processors in its affinity set, if there are other threads available (for variable priorities)

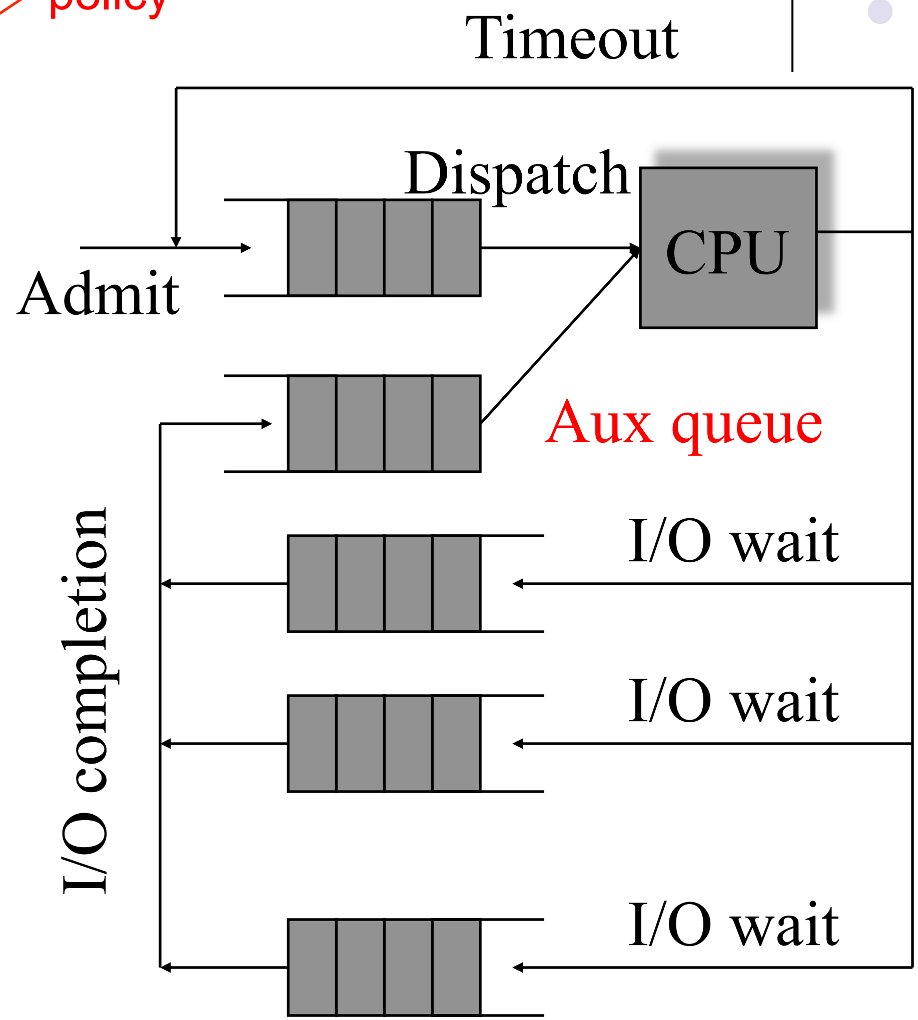
Backup

Virtual Round Robin

policy

- To improve fairness for I/O bound processes
- 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

mechanism



UNIX System V

- One of the first commercial versions of the Unix
- Originally developed by AT&T
- First released in 1983
- 4 major versions, termed R1 R2, R3, and 4
- SVR4, commercially most successful version, the result of Unix System Unification (with collaborations of major UNIX vendors)
  - IBM's AIX is based on SVR3
  - Sun ‘s Solaris and HP's HP-UX are based on SVR4
- 80s-early 90s, System V and BSD were the two major "flavors" of UNIX
  - Large multi-user systems vs. Desktop workstations

Case Study: Traditional Unix Scheduling (SVR3, 4.3 BSD)

Multilevel Feedback Queue with 1 sec preemption

- "1 sec" preemption
  - Preempt if a process doesn't block or complete within 1 sec
- Priority is recomputed every sec (low # is higher)

  P_i = base + CPU_{i-1}/ 2 + nice, where $CPU_i = (U_i + CPU_{i-1}) / 2$

  base is the base priority of the process

  U_i is process utilization in interval i, nice in [-20,20]
- Base priorities
  - Virtual memory Swapper
  - Block I/O device control
  - Character I/O device control
  - User processes