CPU Scheduling

Main Points

- Scheduling policy: what to do next, when there are multiple threads ready to run
  - Or multiple packets to send, or web requests to serve, or …
- Definitions
  - response time, throughput, predictability
- Uniprocessor policies
  - FIFO, round robin, optimal

Example

- You manage a web site, that suddenly becomes wildly popular. Do you?
  - Buy more hardware?
  - Implement a different scheduling policy?
  - Turn away some users? Which ones?
- How much worse will performance get if the web site becomes even more popular?

Definitions

- Task/Job
  - User request: e.g., mouse click, web request, shell command, …
- Latency/response time
  - How long does a task take to complete?
- Throughput
  - How many tasks can be done per unit of time?
- Overhead
  - How much extra work is done by the scheduler?
- Fairness
  - How equal is the performance received by different users?
- Predictability
  - How consistent is the performance over time?

More Definitions

- Workload
  - Set of tasks for system to perform
- Preemptive scheduler
  - If we can take resources away from a running task
- Work-conserving
  - Resource is used whenever there is a task to run
  - For non-preemptive schedulers, work-conserving is not always better
- Scheduling algorithm
  - takes a workload as input
  - decides which tasks to do first
  - Performance metric (throughput, latency) as output
  - Only preemptive, work-conserving schedulers to be considered

First In First Out (FIFO)

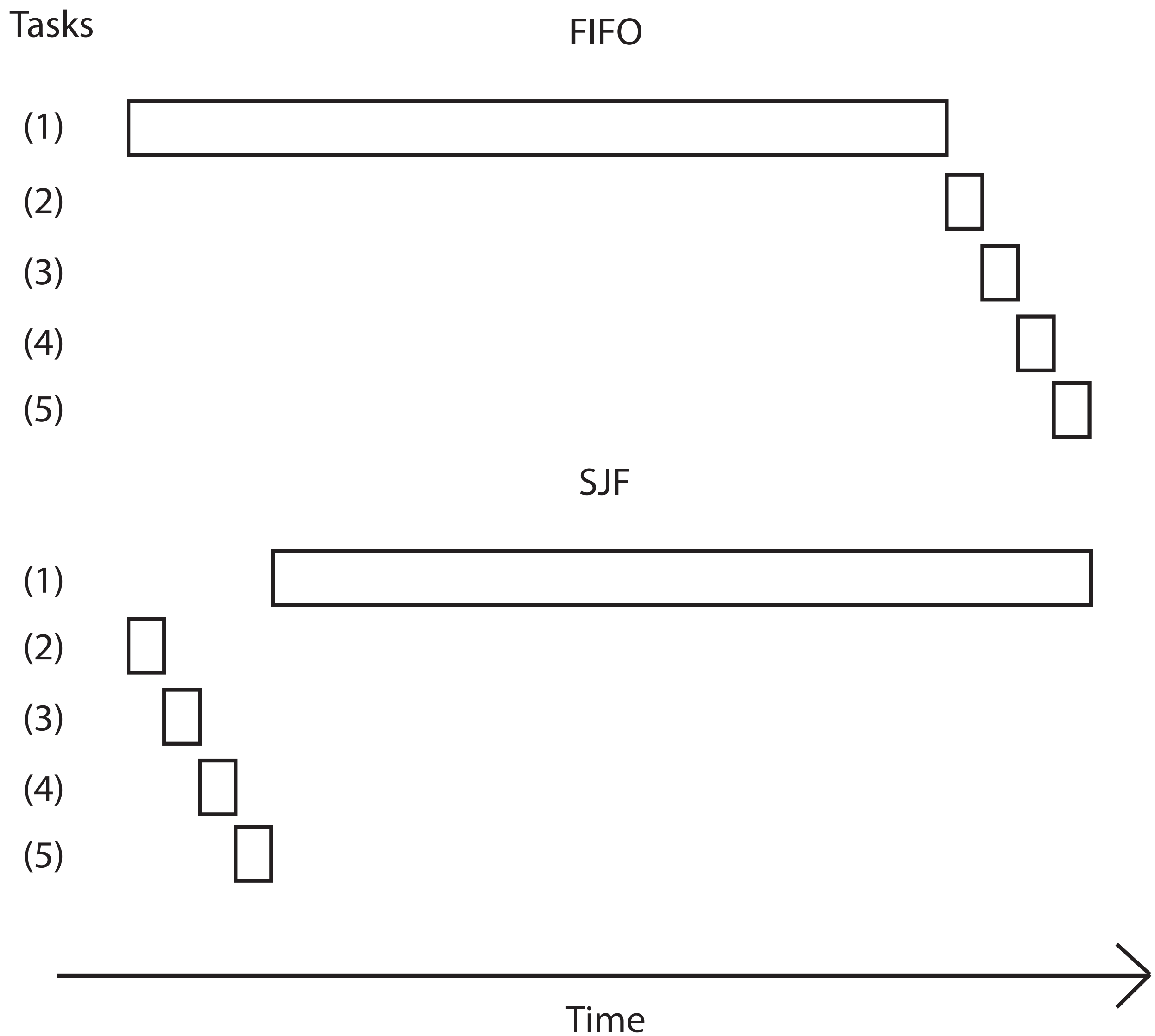
- Schedule tasks in the order they arrive
  - Continue running them until they complete or give up the processor
- Example: memcached
  - Facebook cache of friend lists, …
- On what workloads is FIFO particularly bad?

Shortest Job First (SJF)

- Always do the task that has the shortest remaining amount of work to do
  - Often called Shortest Remaining Time First (SRTF)

- Suppose we have five tasks arrive one right after each other, but the first one is much longer than the others
  - Which completes first in FIFO? Next?
  - Which completes first in SJF? Next?

FIFO vs. SJF



Shortest Job First

- Claim: SJF is optimal for average response time
  - Why?
- For what workloads is FIFO optimal?
- Pessimal?
- Does SJF have any downsides?

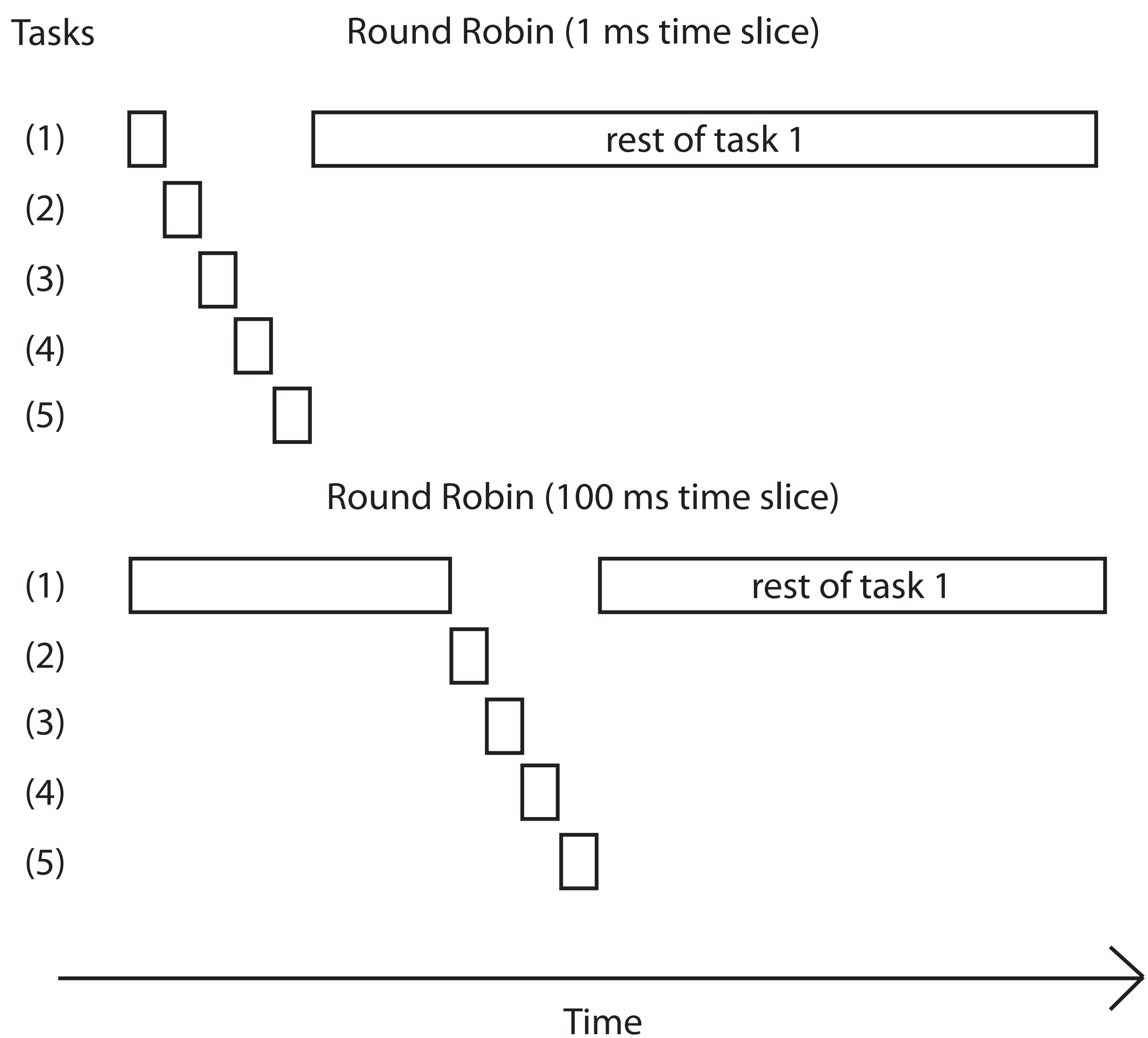
Starvation and Sample Bias

- Suppose you want to compare FIFO and SJF on some sequence of arriving tasks
  - Compute average response time as the average for tasks that start/end in some window
- Is this valid or invalid?

Round Robin

- Each task gets resource for a fixed period of time (time quantum)
  - If task doesn't complete, it goes back in line
- Need to pick a time quantum
  - What if time quantum is too long?
    - Infinite?
  - What if time quantum is too short?
    - One instruction?

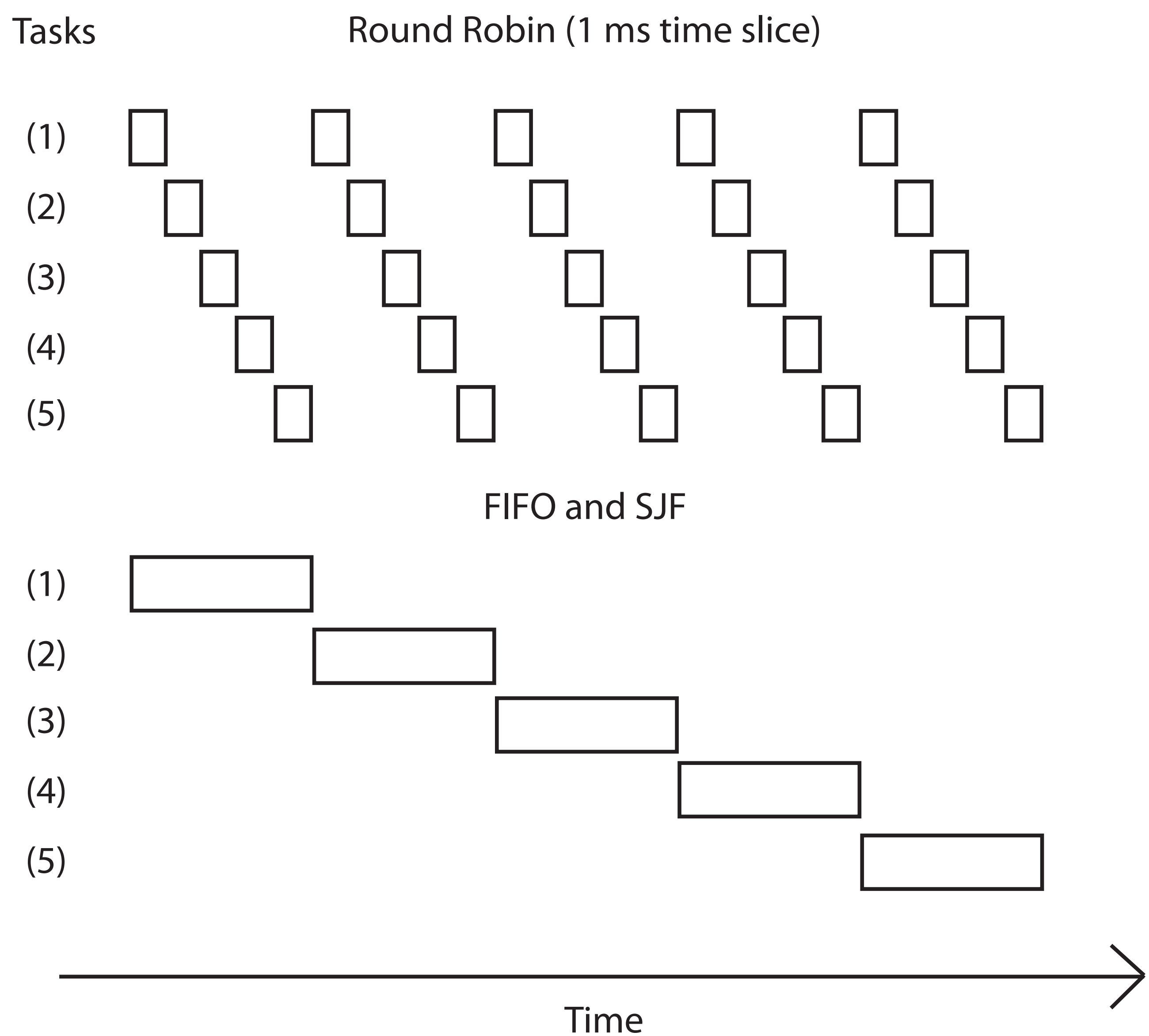
Round Robin


Tasks
Round Robin (1 ms time slice)
(1)
rest of task 1
(2)
(3)
(4)
(5)
Round Robin (100 ms time slice)
(1)
rest of task 1
(2)
(3)
(4)
(5)
Time

Round Robin vs. FIFO

- Assuming zero-cost time slice, is Round Robin always better than FIFO?

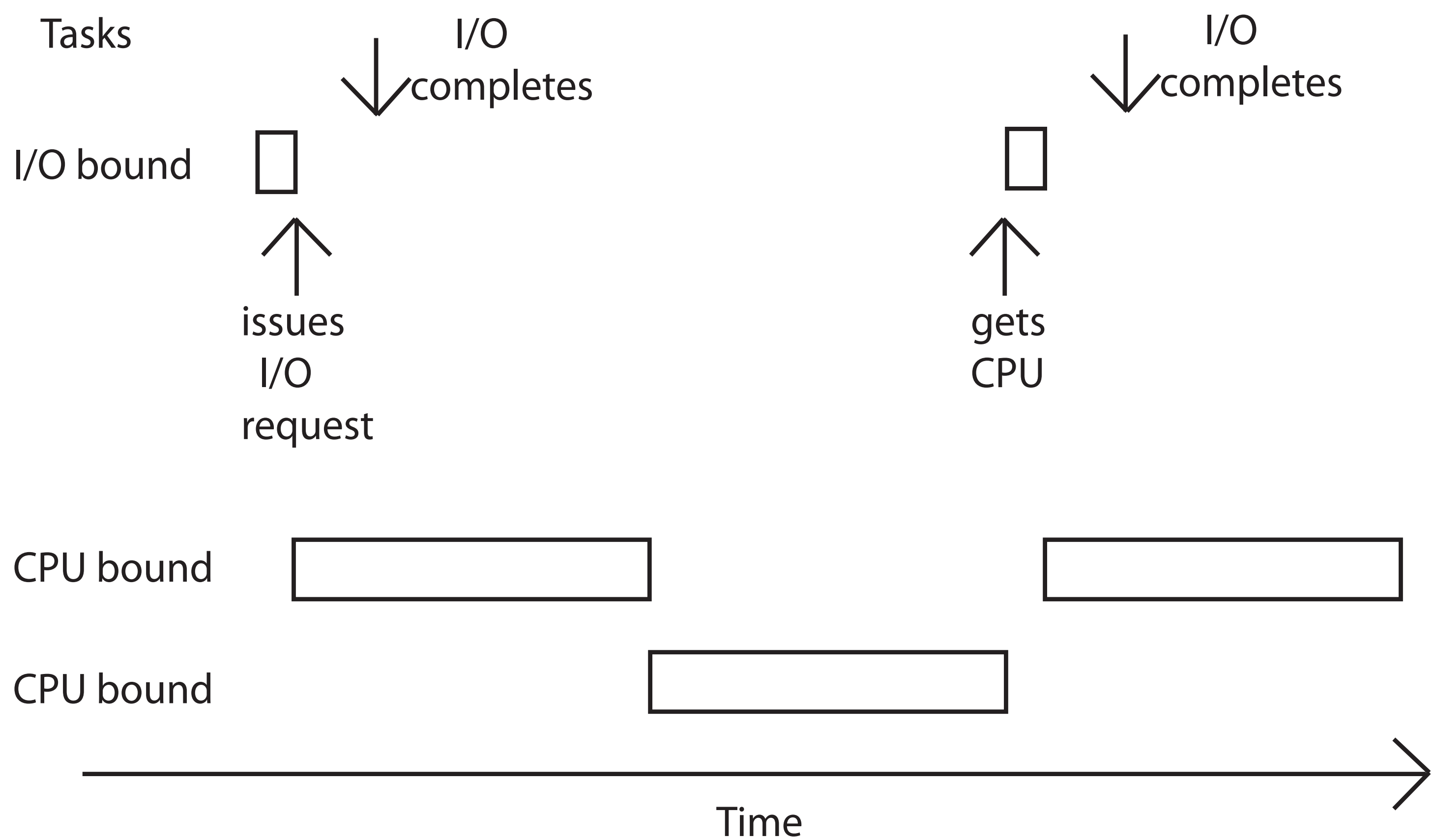
Round Robin vs. FIFO



Round Robin vs. Fairness

- Is Round Robin always fair?

Mixed Workload



Max-Min Fairness

- How do we balance a mixture of repeating tasks:
  - Some I/O bound, need only a little CPU
  - Some compute bound, can use as much CPU as they are assigned
- One approach: maximize the minimum allocation given to a task
  - Schedule the smallest task first, then split the remaining time using max-min

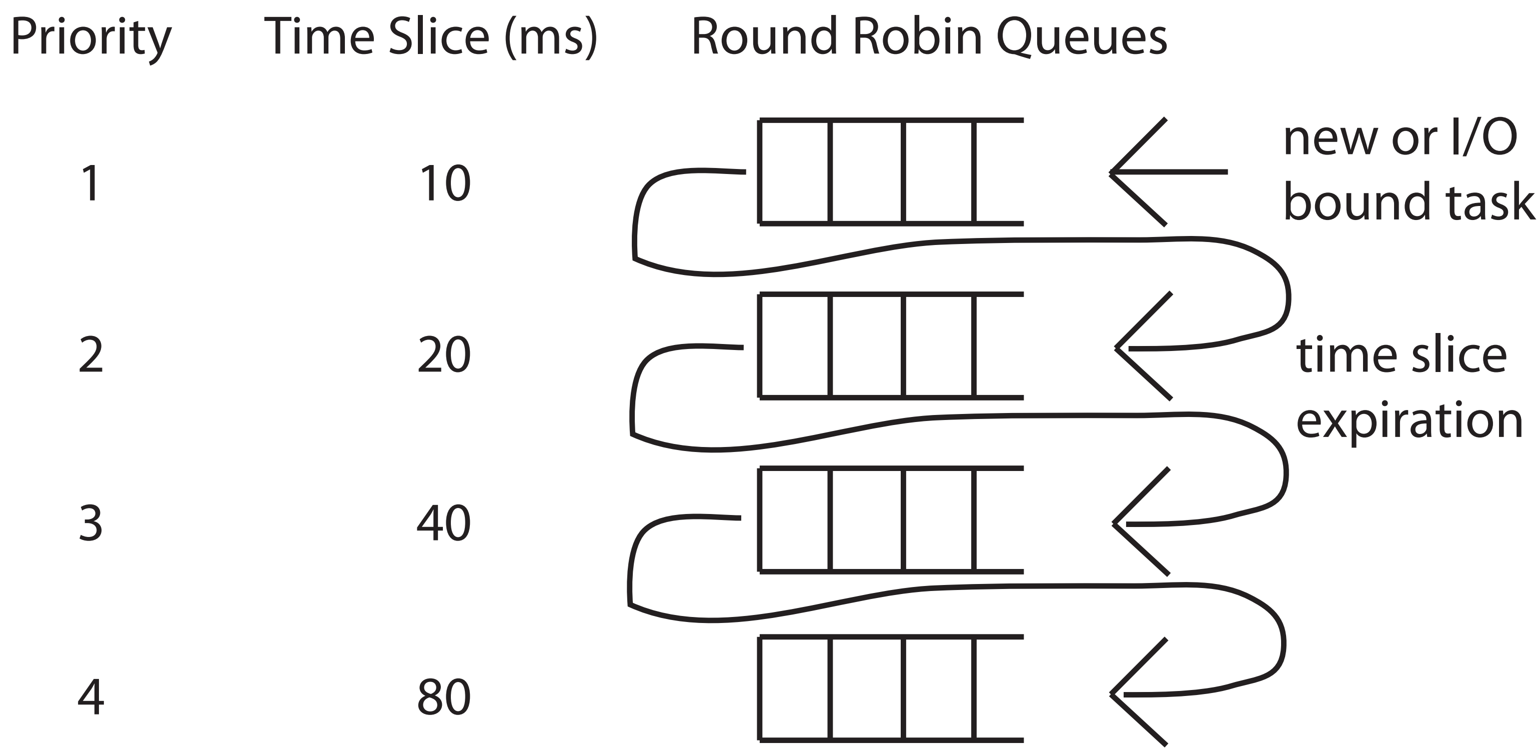
Multi-level Feedback Queue (MFQ)

- Goals:
  - Responsiveness
  - Low overhead
  - Starvation freedom
  - Some tasks are high/low priority
  - Fairness (among equal priority tasks)
- Not perfect at any of them!
  - Used in Linux (and probably Windows, MacOS)

MFQ

- Set of Round Robin queues
  - Each queue has a separate priority
- High priority queues have short time slices
  - Low priority queues have long time slices
- Scheduler picks first thread in highest priority queue
- Tasks start in highest priority queue
  - If time slice expires, task drops one level

MFQ


Priority
Time Slice (ms)
Round Robin Queues
1
10
new or I/O bound task
2
20
time slice expiration
3
40
4
80

Uniprocessor Summary

- FIFO is simple and minimizes overhead.
- If tasks are variable in size, then FIFO can have very poor average response time.
- If tasks are equal in size, FIFO is optimal in terms of average response time.
- Considering only the processor, SJF is optimal in terms of average response time.
- SJF is pessimal in terms of variance in response time.

Uniprocessor Summary

- If tasks are variable in size, Round Robin approximates SJF.
- If tasks are equal in size, Round Robin will have very poor average response time.
- Tasks that intermix processor and I/O benefit from SJF and can do poorly under Round Robin.
- Max-min fairness can improve response time for I/O-bound tasks.
- Round Robin and Max-min fairness both avoid starvation.
- By manipulating the assignment of tasks to priority queues, an MFQ scheduler can achieve a balance between responsiveness, low overhead, and fairness.

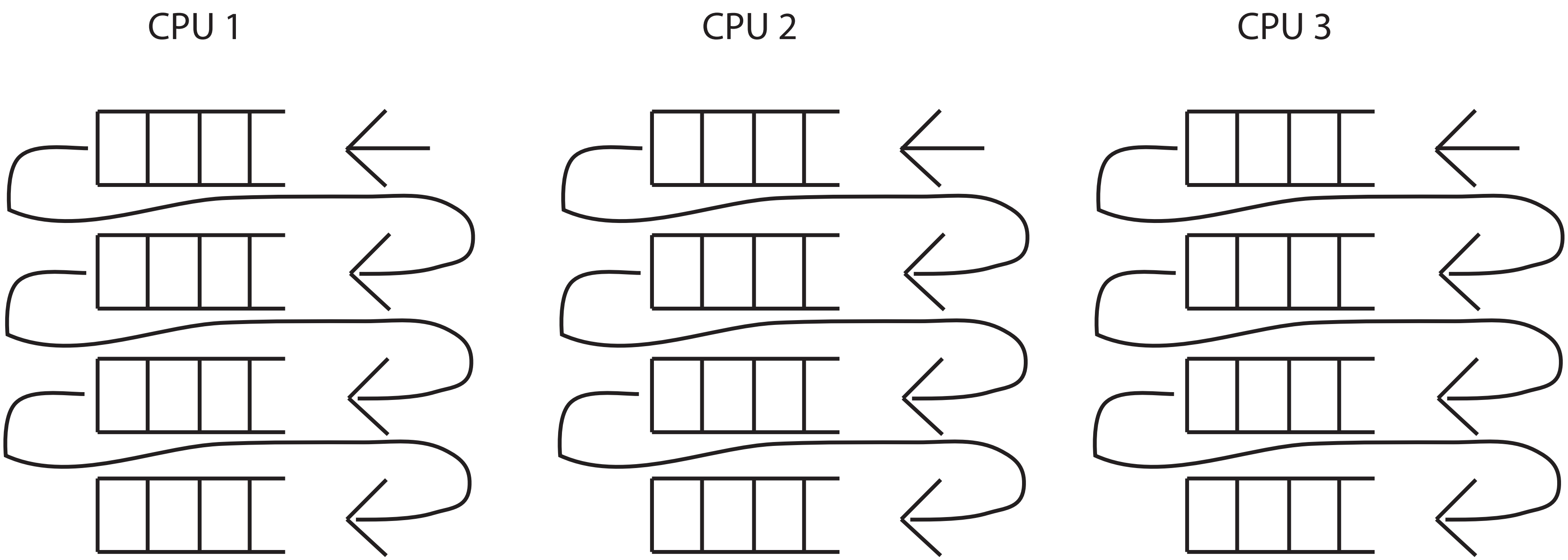
Multiprocessor Scheduling

- What would happen if we used MFQ on a multiprocessor?
  - Contention for scheduler spinlock
  - Programs will have more threads to take advantage of multiprocessor, so more contention
- Amdahl's Law
  - Speedup on a multiprocessor limited by whatever runs sequentially
  - Runtime >= Sequential portion + parallel/# procs

Multiprocessor Scheduling

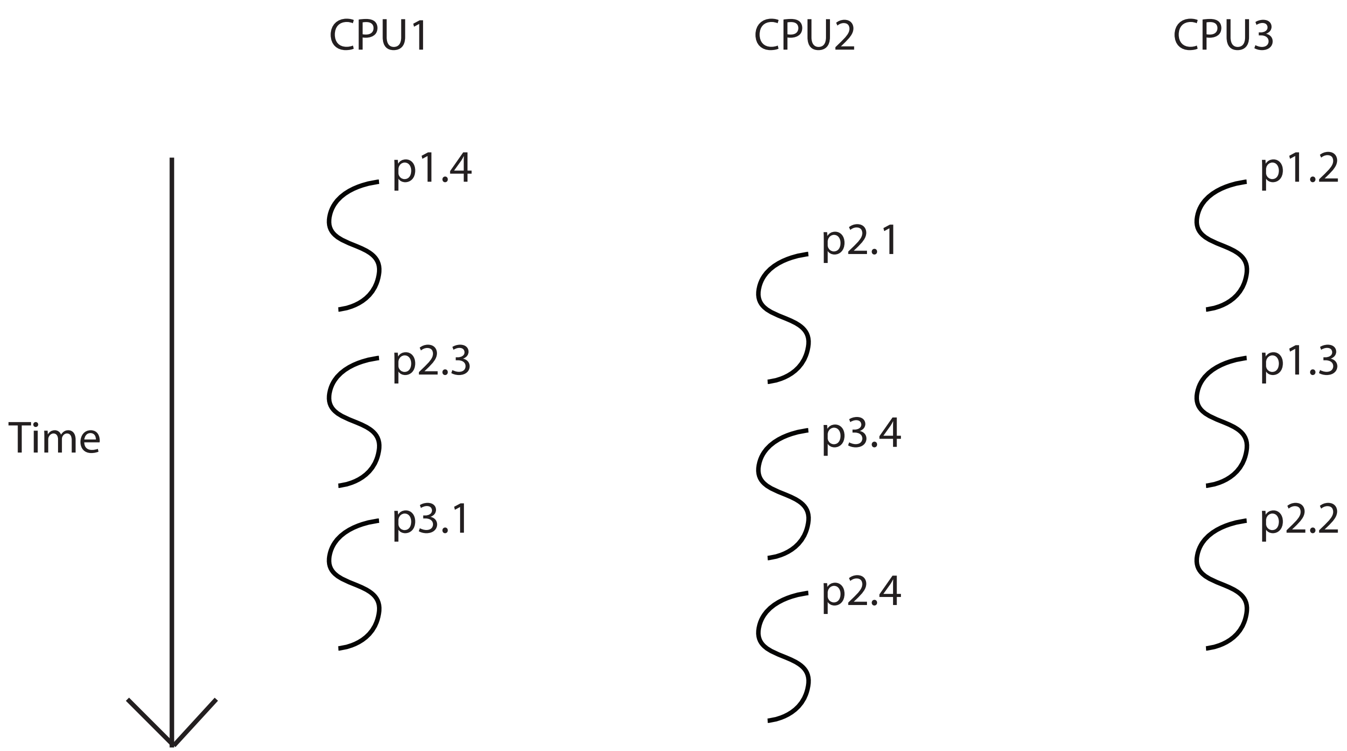
- Modern processor is 100x slower without a cache
- Cache effects of a single ready list:
  - Cache coherence overhead
    - MFQ data structure would ping between caches
    - Fetching data from other caches can be even slower than re-fetching from DRAM
  - Cache reuse
    - Thread's data from last time it ran is often still in its old cache

Per-Processor MFQ



Scheduling Parallel Programs

Oblivious: each processor time-slices its ready list independently of the other processors



px.y = thread y in process x

Scheduling Parallel Programs

- What happens if one thread gets time-sliced while other threads from the same program are still running?

Bulk Synchronous Parallel Program