Scheduling

Module 13

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
- Fundamentals: Unicore policies
  - FIFO, round robin, Optimal
  - multilevel feedback as approximation to optimal
- Multicore policies
  - Affinity scheduling, gang scheduling
- Queueing theory
  - Can you understand/predict/improve a system's response time?

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

Performance Metrics

- Throughput
  - average tasks completed per time unit
- Response Time
  - average time required to complete a task
- Fairness
  - ?
- Unfairness
  - Priorities

Policy: First In First Out (FIFO)

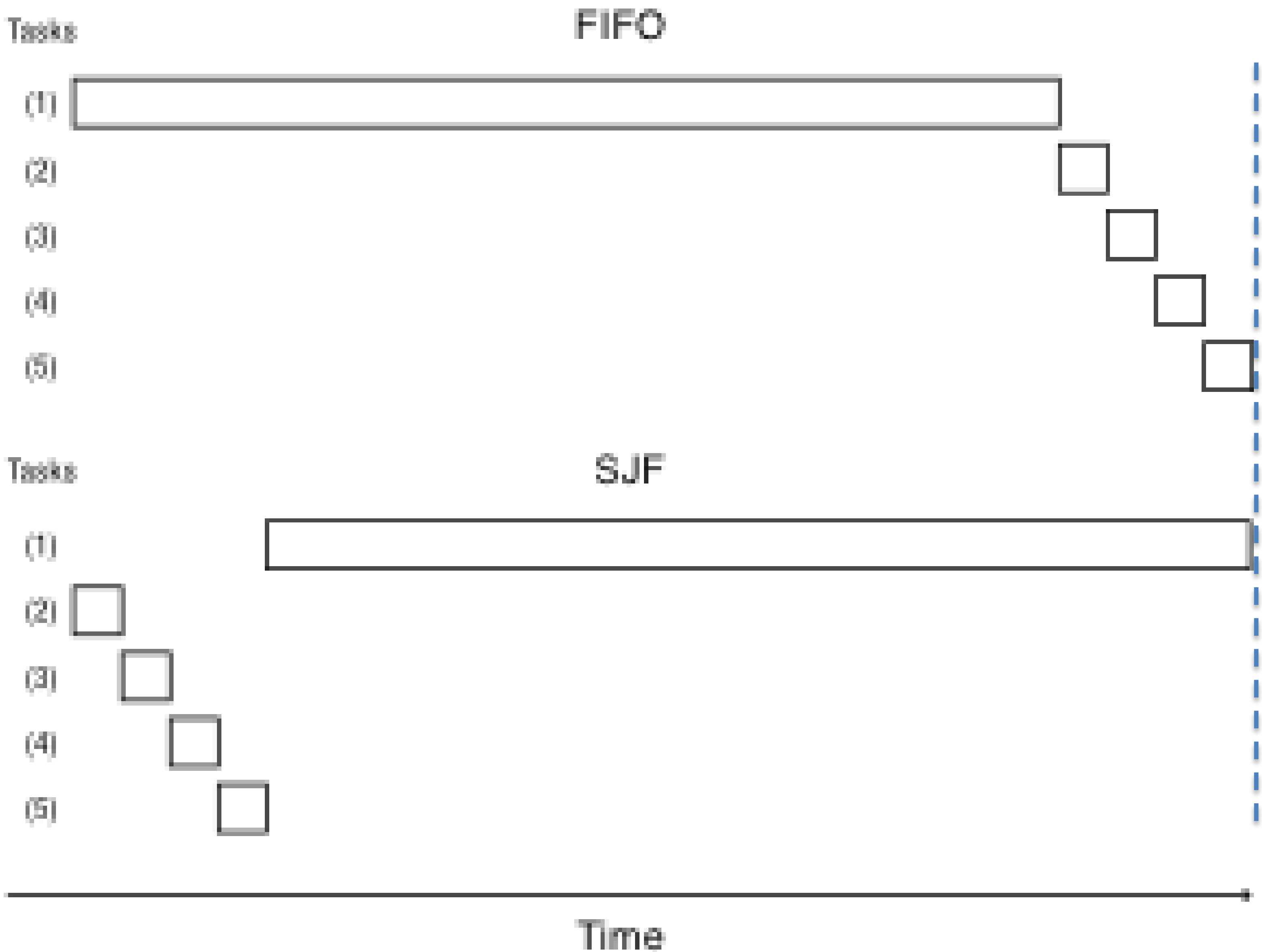
- Schedule tasks in the order they arrive
  - Continue running them until they complete or give up the processor
- On what workloads is FIFO particularly bad?

Policy: 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



Question

- Claim: SJF is optimal for average response time
  - Why?

- Does SJF have any downsides?

Question

- Claim: SJF is optimal for average response time
  - Why?
    - Interchange argument
      - If a longer task precedes a shorter one in the schedule, swap them
      - The longer one's response time in the new schedule equals the shorter one's in the old schedule
      - The shorter one's response time in the new schedule is less than the longer one's in the old schedule
      - So, the average response time has decreased
- Does SJF have any downsides?
  - Fairness?
  - Starvation?

Question

- Is FIFO ever optimal?

- Pessimal?

Question

- Is FIFO ever optimal?
  - When it corresponds to SJF...
  - Including when all tasks are the same length
- Pessimal?
  - When it's longest job first

Evaluation Issues: Starvation and Sample Bias

- Suppose you want to compare two scheduling algorithms
  - Create some infinite sequence of arriving tasks
  - Start measuring
  - Stop at some point
  - Compute average response time as the average for completed tasks between start and stop
- Is this valid or invalid?

Evaluation Issues: Starvation and Sample Bias

- Is this valid or invalid?
  - Maybe yes, maybe no
  - The potential issue is that tasks discriminated against by one of the policies may not finish during the measurement interval evaluating that policy
    - The "bias" is that some kinds of tasks may be measured less frequently than they occur in the workload

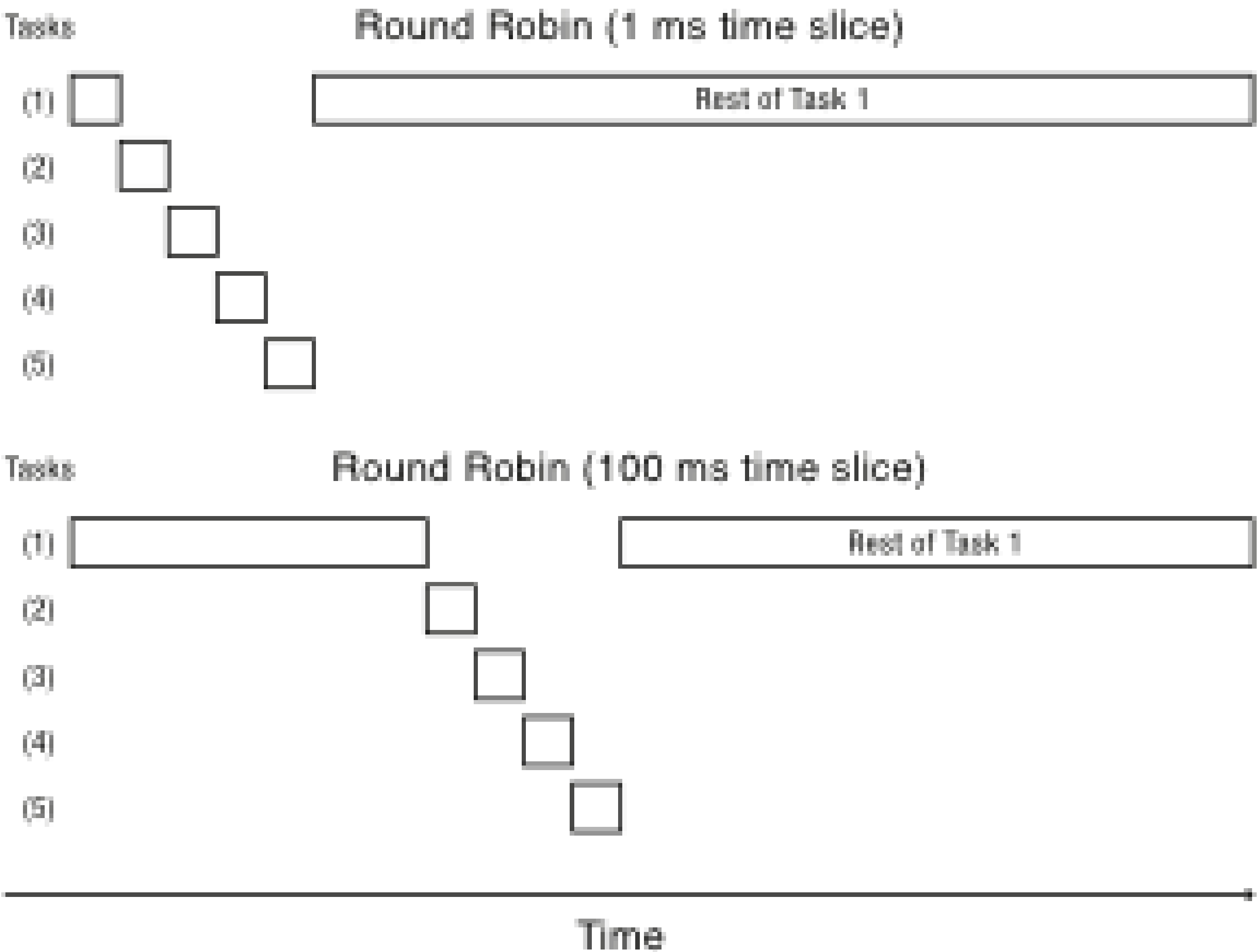
Sample Bias Solutions

- Measure for long enough that # of completed tasks >> # of uncompleted tasks
  - For both systems!
- Start and stop system in idle periods
  - Idle period: no work to do
  - If algorithms are work-conserving, both will complete the same tasks

Policy: (Pre-emptive) Round Robin

- Each task gets resource for a fixed period of time (time quantum)
  - If task doesn’t complete, it goes back in line
- Does this sound familiar?
- 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

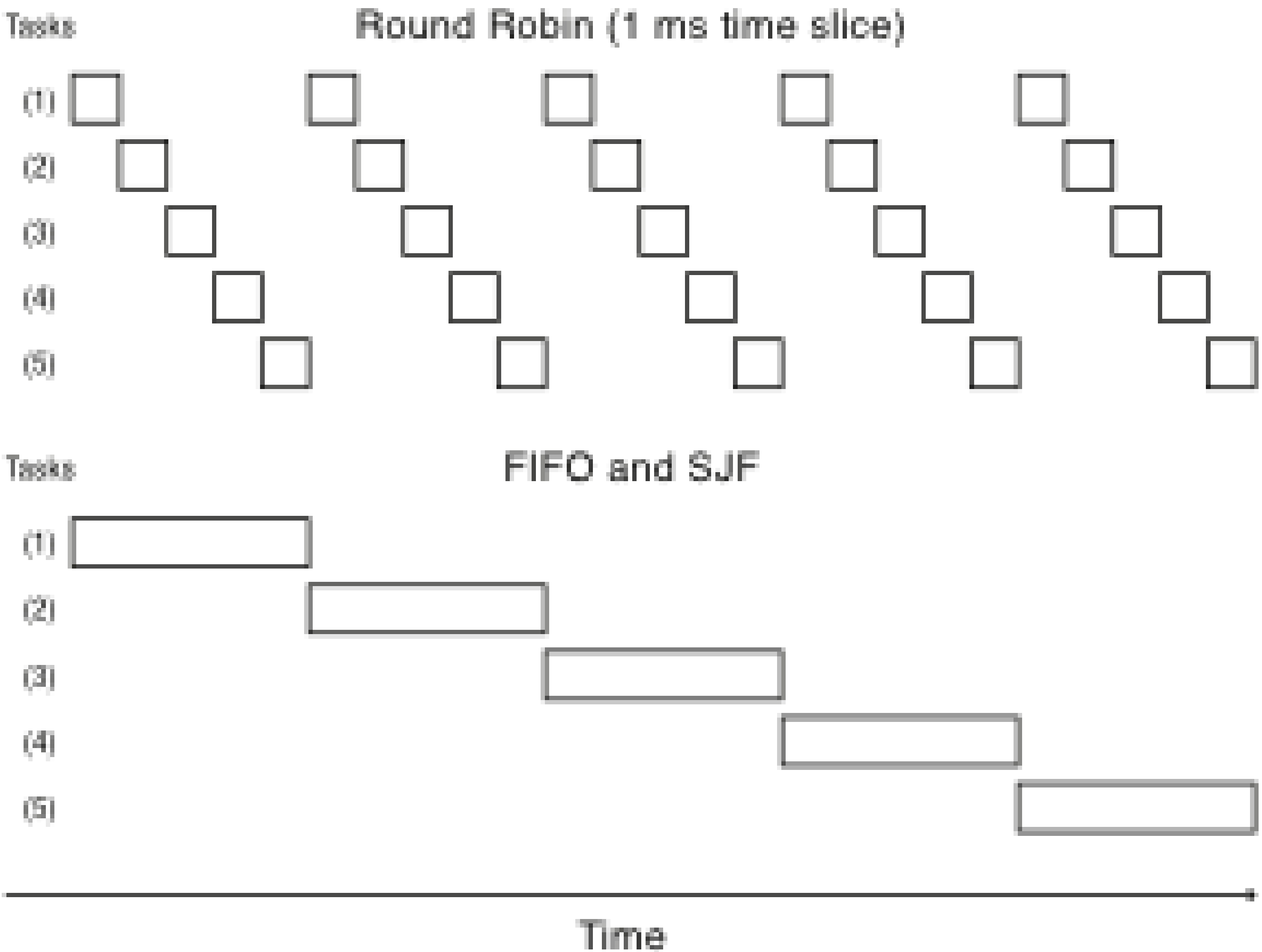


Tasks
Round Robin (1 ms time slice)
(1)
(2)
(3)
(4)
(5)
Rest of Task 1
Tasks
Round Robin (100 ms time slice)
(1)
(2)
(3)
(4)
(5)
Rest of Task 1
Time

Round Robin vs. FIFO

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

Round Robin vs. FIFO

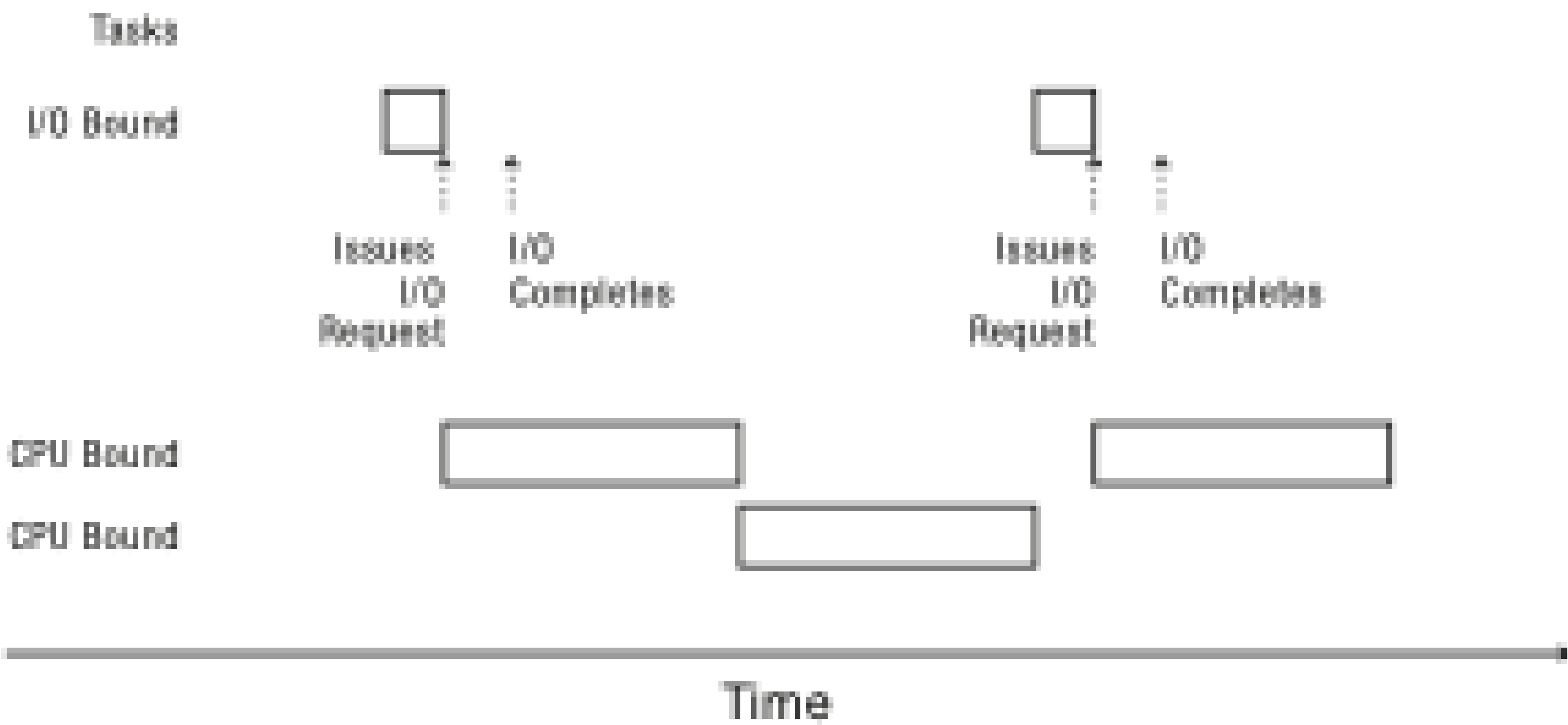


Round Robin == Fairness?

- Is Round Robin always fair?

- What is fair?
  - FIFO?
  - Equal share of the CPU?
  - What if some tasks don’t need their full share?
  - Minimize worst case divergence?
    - Time task would take if no one else was running
    - Time task takes under scheduling algorithm

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
  - If any task needs less than an equal share, schedule the smallest of these first
  - Split the remaining time using max-min
  - If all remaining tasks need at least equal share, split evenly

Linux Completely Fair Scheduler

- Each thread t has a weight, w(t)
- Each runnable thread t should acquire CPU time at rate $w(t) / \sum_j w(j)$
  - no reward while not runnable
- Keep track of accumulated weighted runtime vs. fair share amount
- Over a fixed interval, try to run each runnable thread at least once
  - Set timeslice according to its fair share of interval, based on weights
- Dispatch the thread whose accumulated runtime is most behind its fair share

Uniprocessor Summary (1)

- 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 (2)

- 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.

Uniprocessor Summary (3)

- Max-Min fairness can improve response time for I/O-bound tasks.
- Round Robin and Max-Min fairness both avoid starvation.
- Max-min fairness / Completely Fair Scheduler attempt to roll performance, fairness, and IO behavior into one unified approach (with good success)

Multiprocessor Scheduling

- What new issues are there?
  - Contention for scheduler spinlock
  - Cache slowdown due to ready list data structure pinging from one CPU to another
  - Limited cache reuse: thread's data from last time it ran is often still in its old cache

Per-Processor Affinity Scheduling

- Each processor has its own ready list
  - Protected by a per-processor spinlock
- Put threads back on the ready list where it had most recently run
  - Ex: when I/O completes, or on Condition->signal
- Idle processors can steal work from other processors

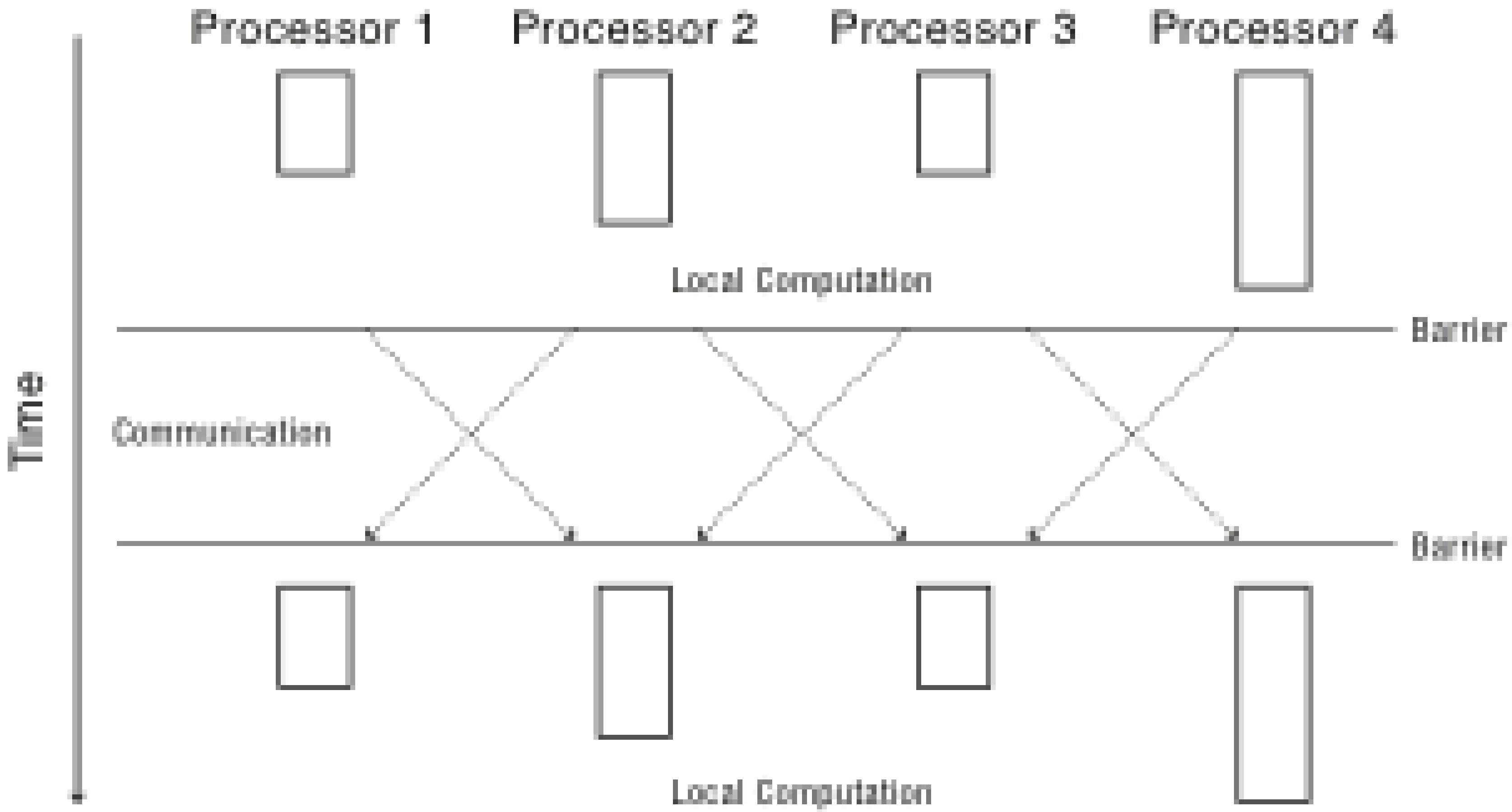
Scheduling Parallel Programs

- A parallel program has many, often fine-grained, threads that frequently synchronize
- What happens if one thread gets time-sliced while other threads from the same program are still running?
  - Assuming program uses locks and condition variables, it will still be correct
  - What about performance?

Bulk Synchronous Parallelism

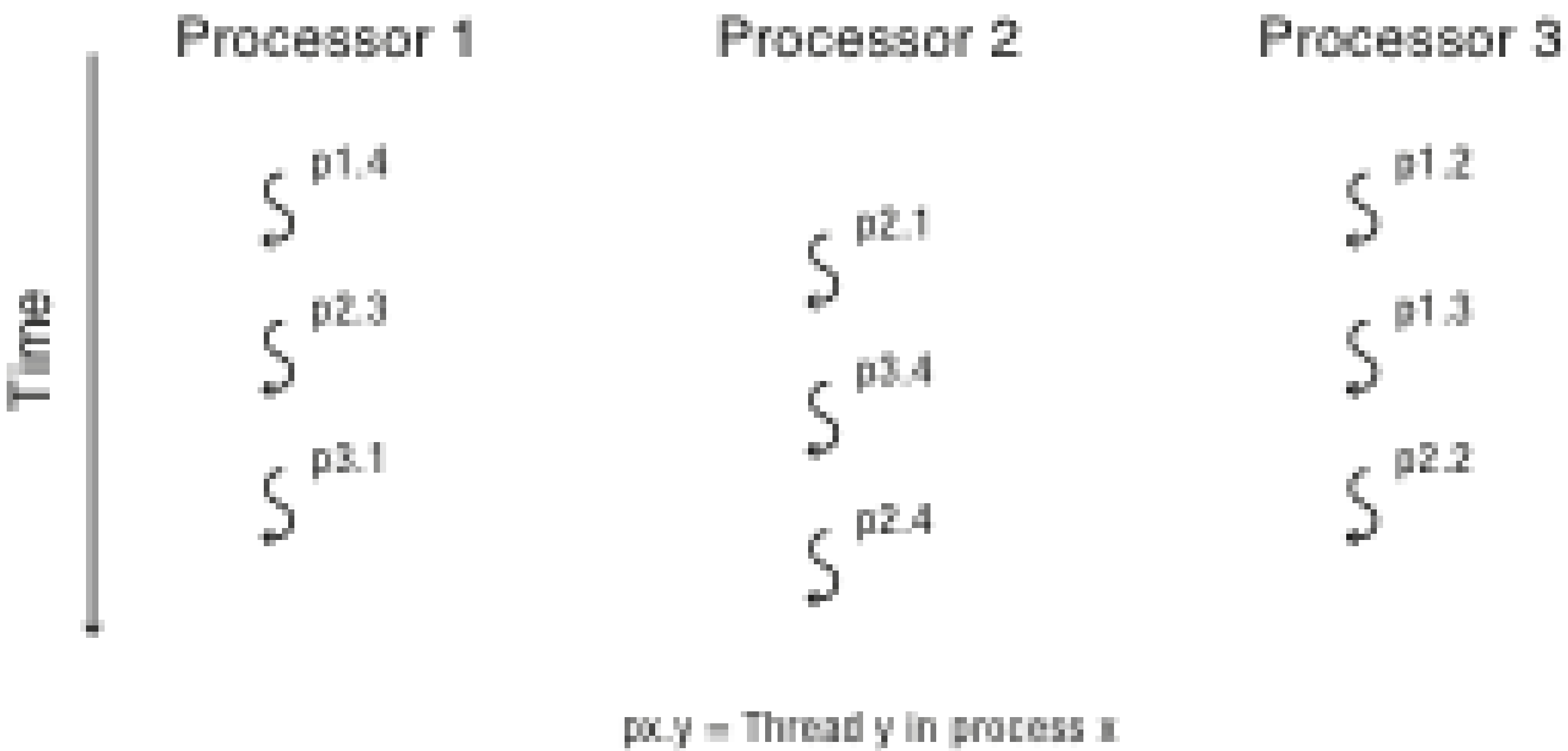
- Loop at each processor:
  - Compute on local data (in parallel)
  - Barrier
  - Send (selected) data to other processors (in parallel)
  - Barrier
- Examples:
  - MapReduce
  - Fluid flow over a wing
  - Most parallel algorithms can be recast in BSP
    - Sacrificing a small constant factor in performance

Tail Latency

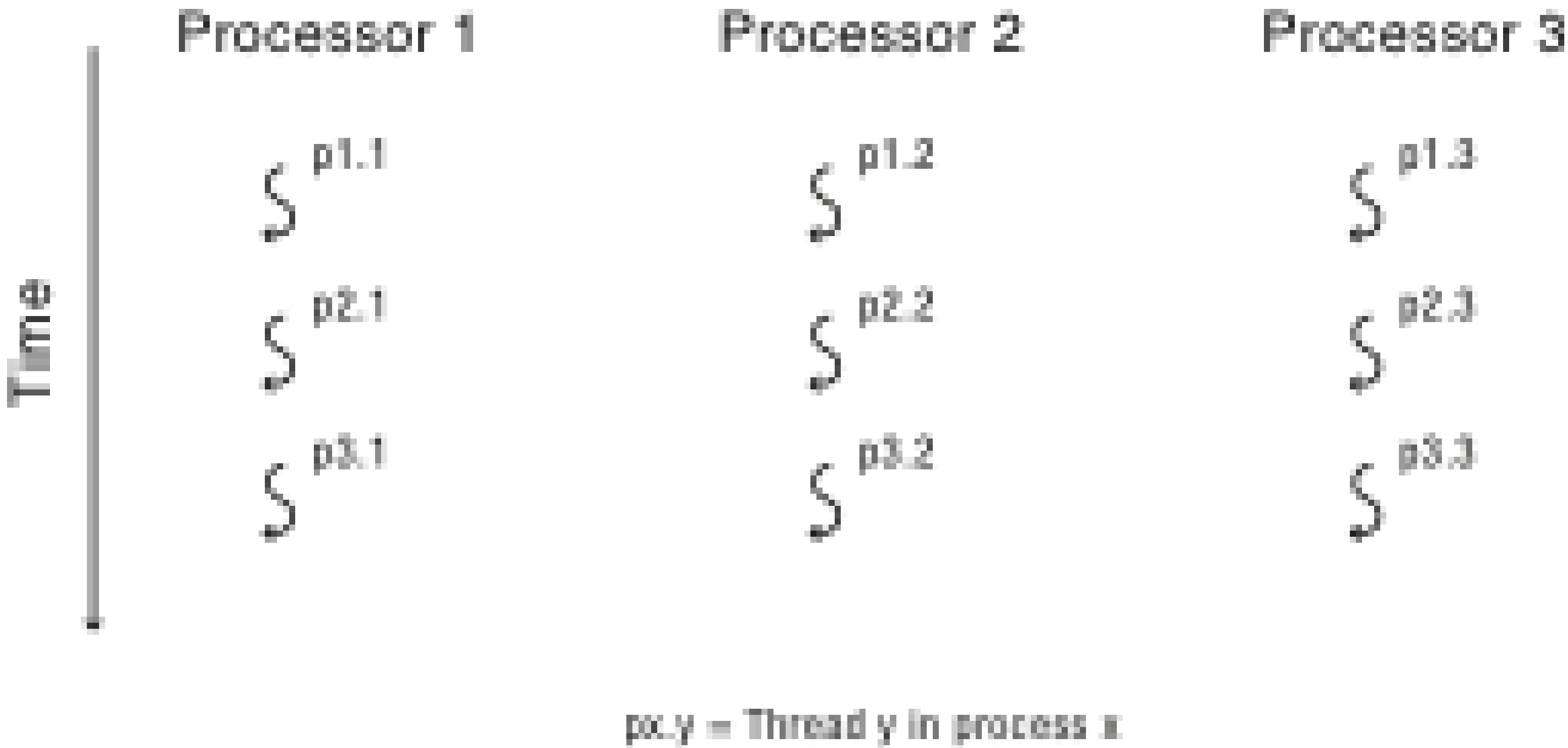


Scheduling Parallel Programs

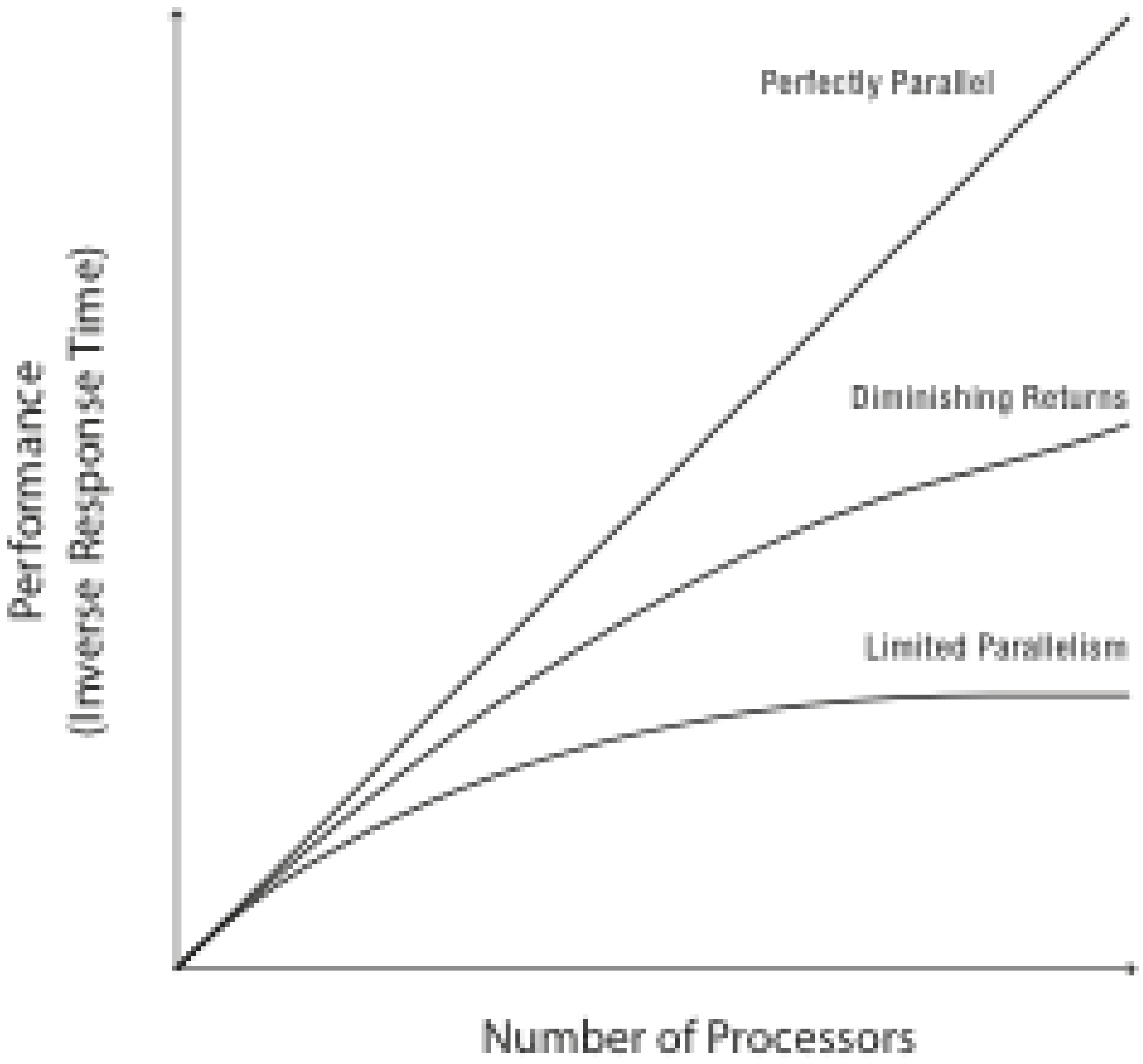
Oblivious: each processor time-slices its ready list independently of the other processors



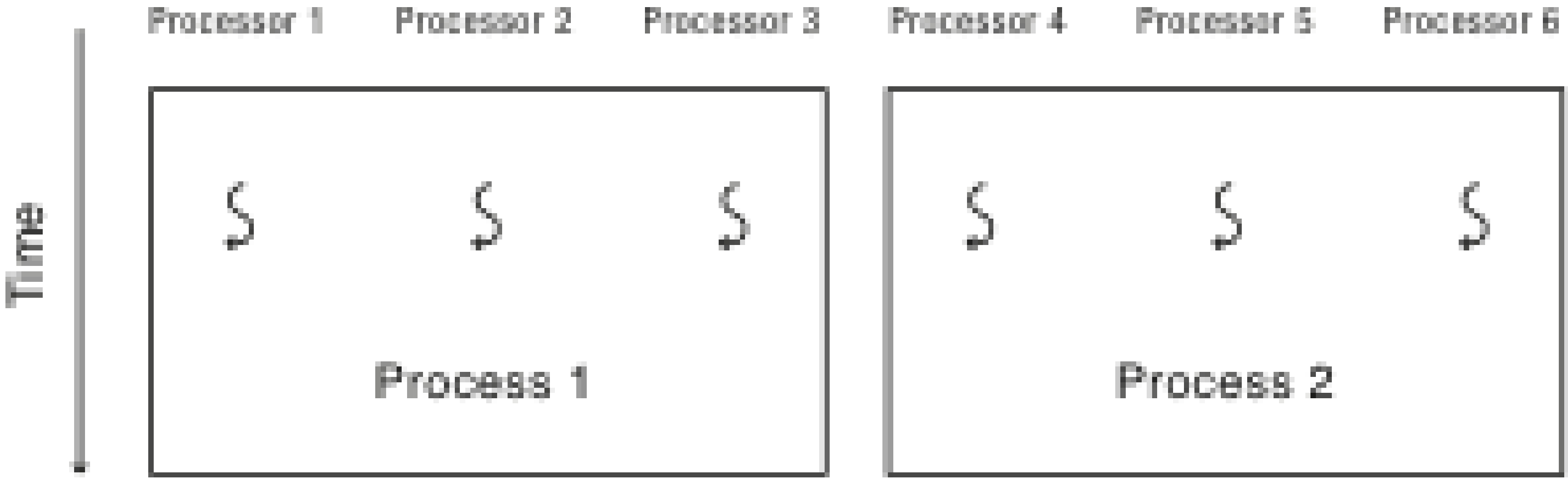
Gang Scheduling



Parallel Program Speedup



Space Sharing



Scheduler activations: kernel tells each application its # of processors with upcalls every time the assignment changes