Operating Systems (A)
(Honor Track)

Lecture 12: Scheduling (I)

Yao Guo (郭耀)

Peking University

Fall 2021

Acknowledgements: Prof. Xiangqun Chen & Tao Wang at PKU and Prof. Yuanyuan Zhou at UCSD

This Lecture

Scheduling Overview

Single Processor Scheduling

Scheduling for batch systems

Interactive scheduling

Buzz Words

Scheduling

Long-term scheduler, or job scheduler

Short-term scheduler, or CPU scheduler

Dispatcher

Preemptive scheduling

Non-preemptive scheduling

Multiprogramming

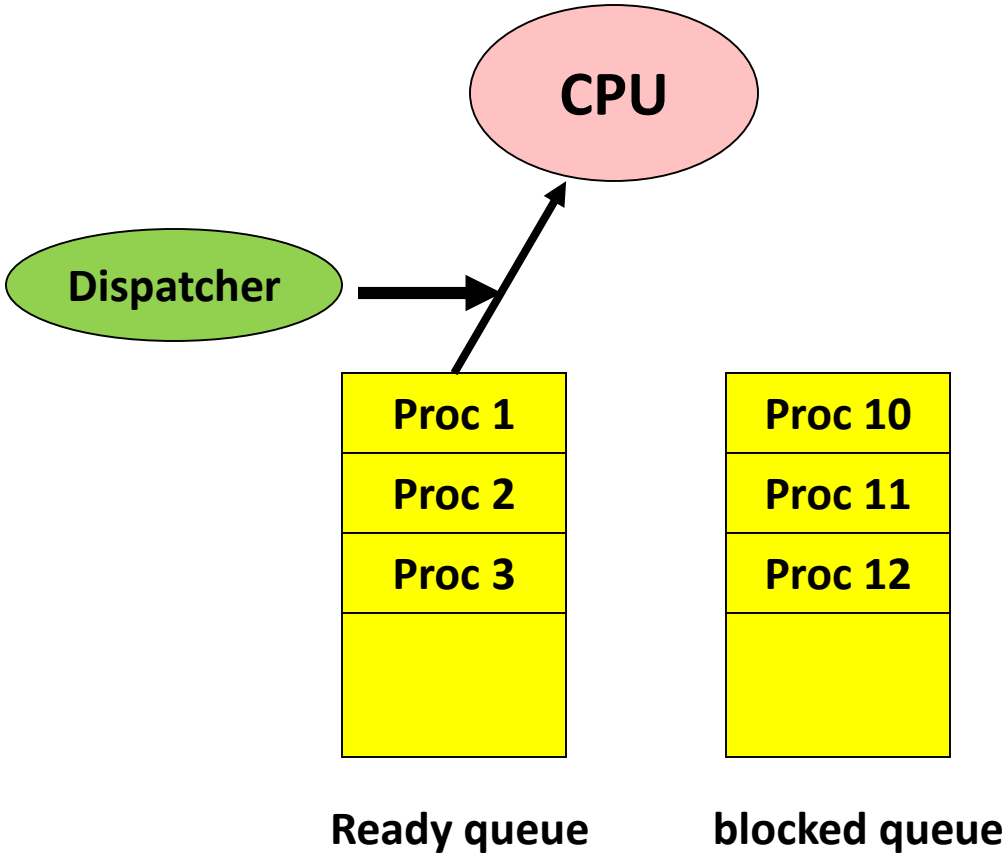
- In a multiprogramming system, we try to increase CPU utilization and job throughput by overlapping I/O and CPU activities
  - Requires a combination of mechanisms and policy
- We have covered the mechanisms
  - Context switching, how and when it happens
  - Process queues and process states
- Now we'll look at the policies
  - Which process (thread) to run, for how long, etc.
- Making this decision is called scheduling
- We'll refer to schedulable entities as jobs (standard usage) – could be processes, threads, people, etc.

Long-term vs. Short-term Schedulers

- Long-term scheduler, or job scheduler
  - In a batch system, some processes are spooled to a mass-storage device for later execution
  - Selects processes from this pool and loads them into memory (for execution)
- Short-term scheduler, or CPU scheduler
  - Selects those from among the processes that are ready to execute and allocates the CPU to one of them

Related Concepts of CPU Scheduler

- Dispatcher
- Preemptive vs. non-preemptive

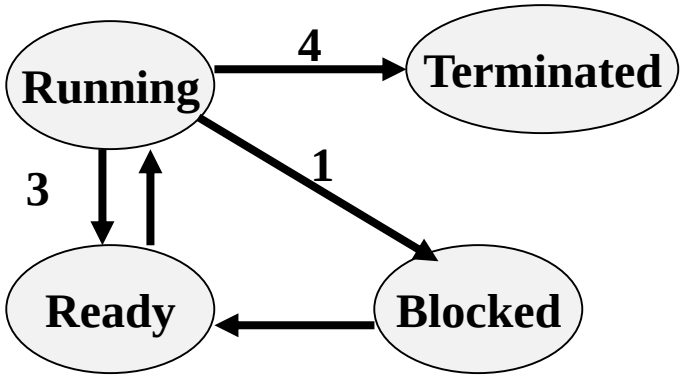


Dispatcher

- Gives the control of the CPU to the process, scheduled by the short-term scheduler
- Dispatcher Functions
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program
- *Dispatch Latency*: time to stop process and start another one
  - Pure overhead
  - Needs to be fast

Preemptive vs. Non-preemptive

- **Non-preemptive scheduling**
  - The running process keeps the CPU until it voluntarily gives up the CPU
    - process exits
    - switches to blocked state
    - 1 and 4 only (no 3)



- **Preemptive scheduling**
  - The running process can be interrupted and must release the CPU (can be forced to give up CPU)

When to Schedule?

- A new process starts
- The running process exits
- The running process is blocked
- I/O interrupt (some processes will be ready)
- Clock interrupt (e.g. every 10 milliseconds)

Scheduling Objectives

- Performance
- Fair
- Priority
- Encourage good behavior
- Support heavy loads
- Adapt to different environments
  - interactive, real-time, multi-media

Performance Criteria

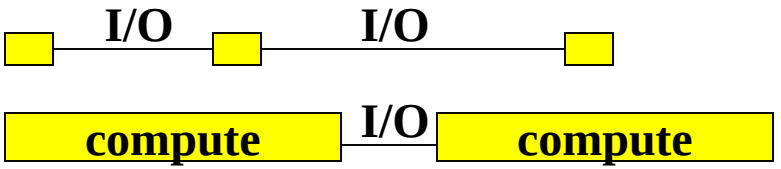
- Efficiency: keep resources as busy as possible
- Throughput: # of processes that completes in unit time
- Turnaround Time
  - The interval from the time of submission of a process to the time of completion
- Waiting Time
  - The sum of the periods spent waiting in the ready queue.
- Response Time
  - The amount of time from when a request was first submitted until first response is produced.
  - Predictability and variance (Why is this important?)
- Meeting Deadlines: avoid losing data

Different Systems, Different Focuses

- For all
  - Fairness, policy enforcement, resource balance
- Batch Systems
  - Max throughput, min turnaround time, max CPU utilization
- Interactive Systems
  - Min Response time, …
- Real-Time Systems
  - Predictability, meeting deadlines

Program Behaviors Considered in Scheduling

- Is it I/O bound? Example?
- Is it CPU bound? Example?



- Batch or interactive environment
- Urgency
- Priority
- Frequency of page faults
- Frequency of preemption
- How much execution time it has already received
- How much execution time it needs to complete

This Lecture

Scheduling Overview

Single Processor Scheduling

Scheduling for batch systems

Interactive scheduling

Buzz Words

- First come first serve (FCFS)
- Shortest job first (SJF)
- Starvation
- Round-robin (RR)
- Priority inversion
- Priority inheritance
- Multiple-level feedback queues (MLFQ)

This Lecture

Scheduling Overview

Single Processor Scheduling

Scheduling for batch systems

Interactive scheduling

First Come First Serve (FCFS)

- Process that requests the CPU FIRST is allocated the CPU FIRST
  - Also called FIFO
- Non-preemptive
- Real life analogy: Fast food restaurant
- Implementation: FIFO queues
  - A new process enters the tail of the queue
  - The scheduler selects from the head of the queue
- Performance metric: **Average Waiting Time** (AWT)
- Given Parameters:
  - Duration(in ms), Arrival Time and Order

FCFS Example

Process	Duration	Order	Arrival Time
P1	24	1	0
P2	3	2	0
P3	4	3	0

The final schedule:

P1 waiting time: 0
P2 waiting time: 24
P3 waiting time: 27

The average waiting time:
(0+24+27)/3 = 17

Problems with FCFS

- Non-preemptive
- Not optimal AWT (average waiting time)
  - Average waiting time can be large if small jobs wait behind long ones
  - E.g., You have a basket, but you're stuck behind someone with a full shopping cart?
- Solution?
  - Express lane (5 items or less)

Shortest Job First (SJF)

- Schedule the job with the shortest duration first
- Two types: Non-preemptive & Preemptive
- Requirement: the lengths of the jobs need to be known in advance
- Optimal if all the jobs are available simultaneously (provable). Why?
  - Gives the best possible AWT (average waiting time)
- Real life analogy?
  - Express lane in supermarket
  - Shortest important task first

--The 7 Habits of Highly Effective People

Non-preemptive SJF: Example

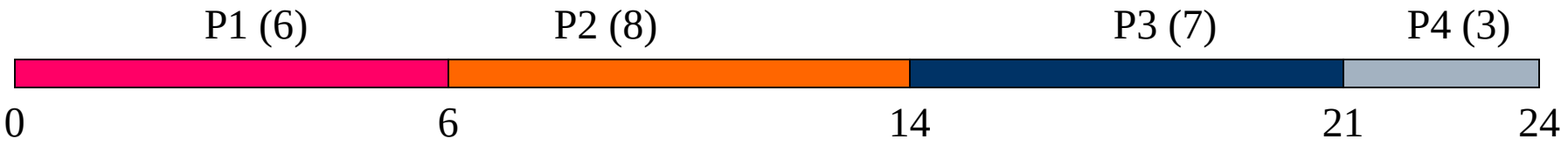
Process	Duration	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0

P4 waiting time: 0
P1 waiting time: 3
P3 waiting time: 9
P2 waiting time: 16

The total time is: 24
The average waiting time (AWT):
(0+3+9+16)/4 = 7

Comparing to FCFS

Process	Duration	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0



P1 waiting time: 0
P2 waiting time: 6
P3 waiting time: 14
P4 waiting time: 21

The total time is the same (why?)

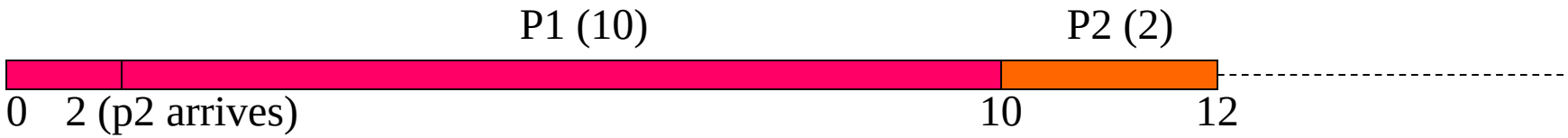
The average waiting time (AWT):
(0+6+14+21)/4 = 10.25

(7 in SJF)

SJF is Not Always Optimal

Process	Duration	Order	Arrival Time
P1	10	1	0
P2	2	2	2

- Is SJF optimal if all the jobs are not available simultaneously?

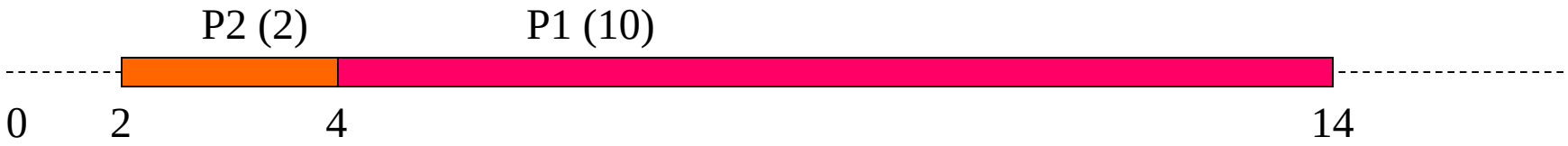


P1 waiting time: 0
P2 waiting time: 8

The average waiting time (AWT):
(0+8)/2 = 4

What if the Scheduler Waits for 2 Time Units?

Process	Duration	Order	Arrival Time
P1	10	1	0
P2	2	2	2



P1 waiting time: 4
P2 waiting time: 0

The average waiting time (AWT):
(0+4)/2 = 2

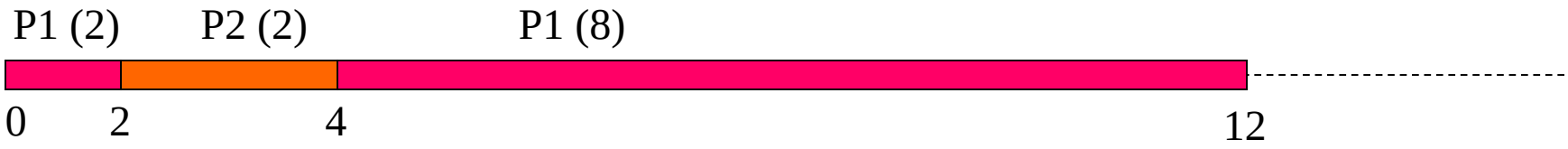
However: waste 2 time units of CPU

Preemptive SJF

- Also called Shortest Remaining Time First
  - Schedule the job with the shortest remaining time required to complete
- Requirement: the lengths of the jobs need to be known in advance

Preemptive SJF: Same Example

Process	Duration	Order	Arrival Time
P1	10	1	0
P2	2	2	2



P1 waiting time: 4-2 =2
P2 waiting time: 0

The average waiting time (AWT):
(0+2)/2 = 1
No CPU waste!!!

A Problem with (Preemptive) SJF

- Starvation
  - In some condition, a job is waiting forever
  - Example: SJF
    - Process *A* with length of 1 hour arrives at time 0
    - But every 1 minute, a short process with length of 2 minutes arrives
    - Result of SJF: *A* never gets to run

This Lecture

Scheduling Overview

Single Processor Scheduling

Scheduling for batch systems

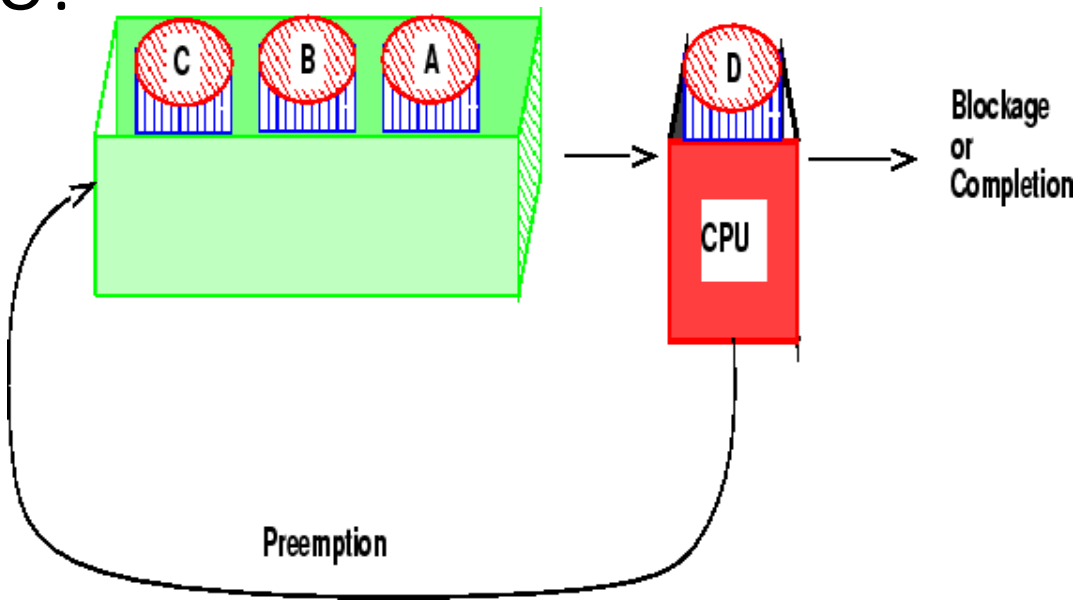
Interactive scheduling

Interactive Scheduling Algorithms

- Usually preemptive
  - Time is sliced into quantum (time intervals)
  - Scheduling decision is also made at the beginning of each quantum
- Performance criteria
  - Min response time
  - AWT is also important
- Representative algorithms:
  - Round-robin
  - Priority-based
    - Extra: what happened on Mars ?
  - …

Round-robin

- One of the oldest, simplest, most commonly used scheduling algorithm
- Select process/thread from ready queue in a round-robin fashion (take turns)
- Difference with FIFO?



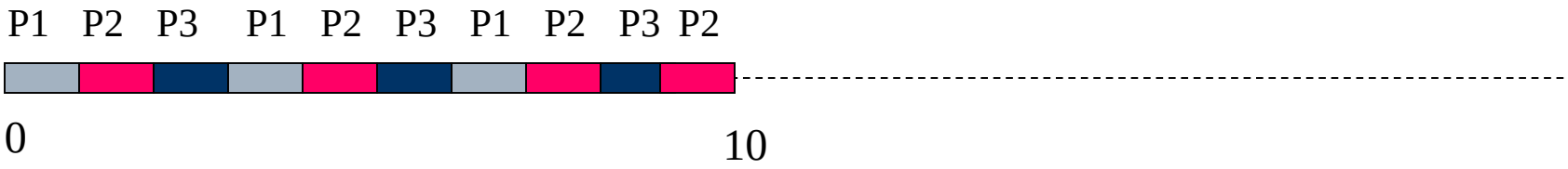
Problem:

- Do not consider priority
- Context switch overhead

Round-robin: Example

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is: 1 unit, P1, P2 & P3 never block



P1 waiting time: 4
P2 waiting time: 6
P3 waiting time: 6

The average waiting time (AWT):
(4+6+6)/3 = 5.33

Time Quantum

- Time slice too large
  - FIFO behavior
  - Poor response time
- Time slice too small
  - Too many context switches (overheads)
  - Inefficient CPU utilization
- Heuristic: 70-80% of jobs block within time-slice
- Typical time-slice 10 to 100 ms
- Time spent in system depends on the size of the job

Simple Calculation (1 minute)

- Suppose a context switch takes 1 msec
- Suppose the scheduling quantum is 100 msec

- What is the percentage overhead of context switches in round-robin scheduling?

- What if the scheduling quantum is 10 msec?

Exercise (Ch 2-43)

43. Measurements of a certain system have shown that the average process runs for a time T before blocking on I/O. A process switch requires a time S, which is effectively wasted (overhead). For round-robin scheduling with quantum Q, give a formula for the CPU efficiency for each of the following:

(a) $Q = \infty$
(b) $Q > T$
(c) $S < Q < T$
(d) $Q = S$
(e) Q nearly 0

Priority Scheduling

- Each job is assigned a priority
- FCFS/RR within each priority level
- Select highest priority job over lower ones
  - Motivation: higher priority jobs are more mission-critical
    - Example: video player vs. send email
  - Note: the priority of a process may be changed over time
- Real life analogy?
  - Boarding at airports
- Problems:
  - May not give the best AWT
  - Indefinitely blocking or “starving” a process

Set Priority

- Two approaches
  - Static (for system with well known and regular application behaviors)
  - Dynamic (otherwise)
- Priority may be based on:
  - Cost to user
  - Importance of user
  - Aging
  - Percentage of CPU time used in last X hours

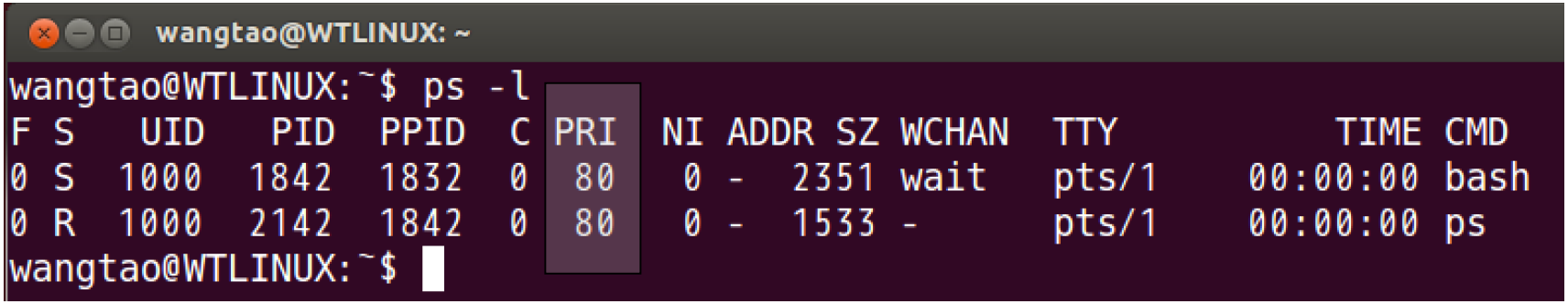
Priority Scheduling: Example

Process	Duration	Priority	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0

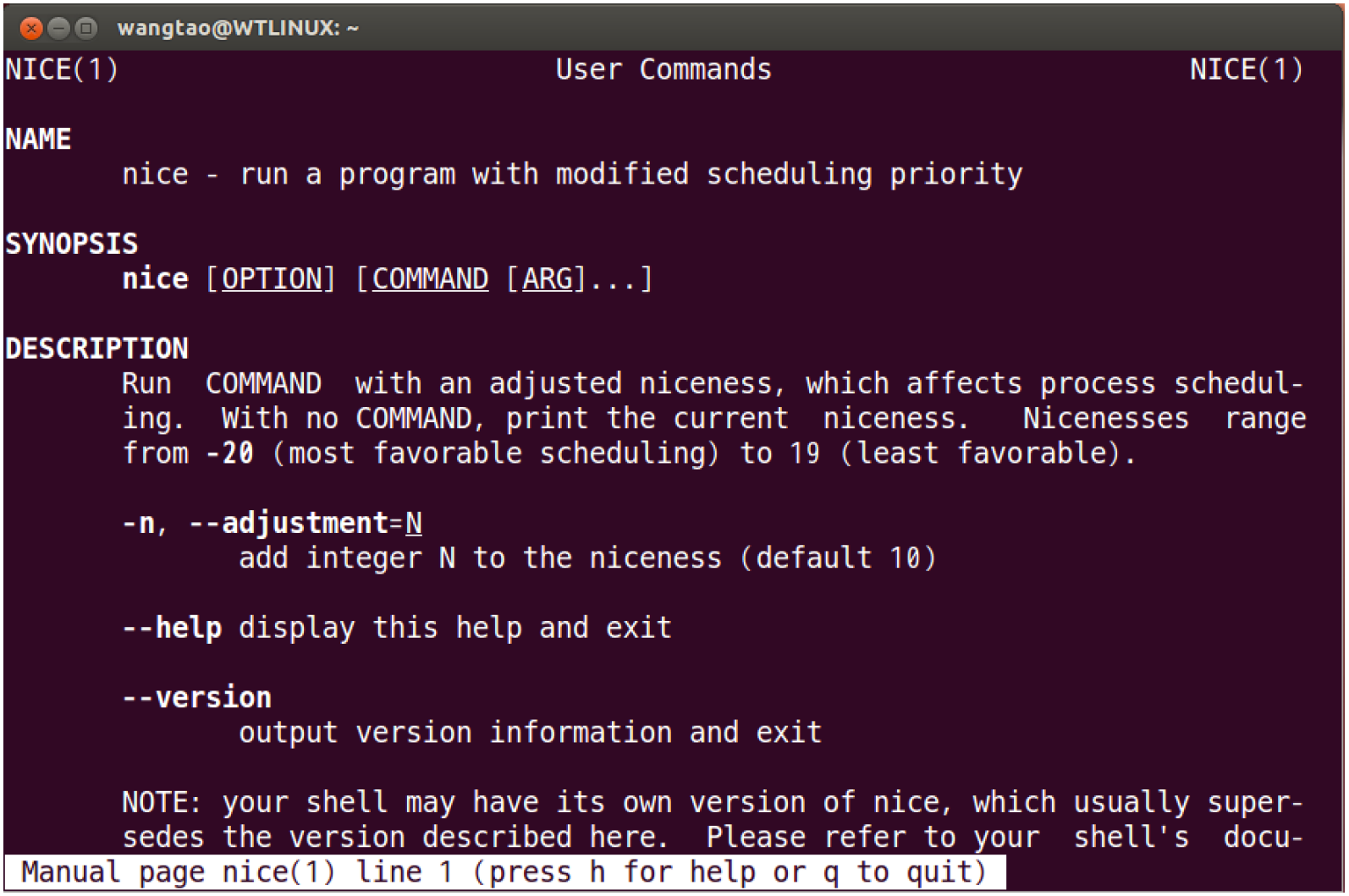
P2 waiting time: 0
P4 waiting time: 8
P3 waiting time: 11
P1 waiting time: 18

The average waiting time (AWT):
(0+8+11+18)/4 = 9.25
(worse than SJF)

Priority in Unix

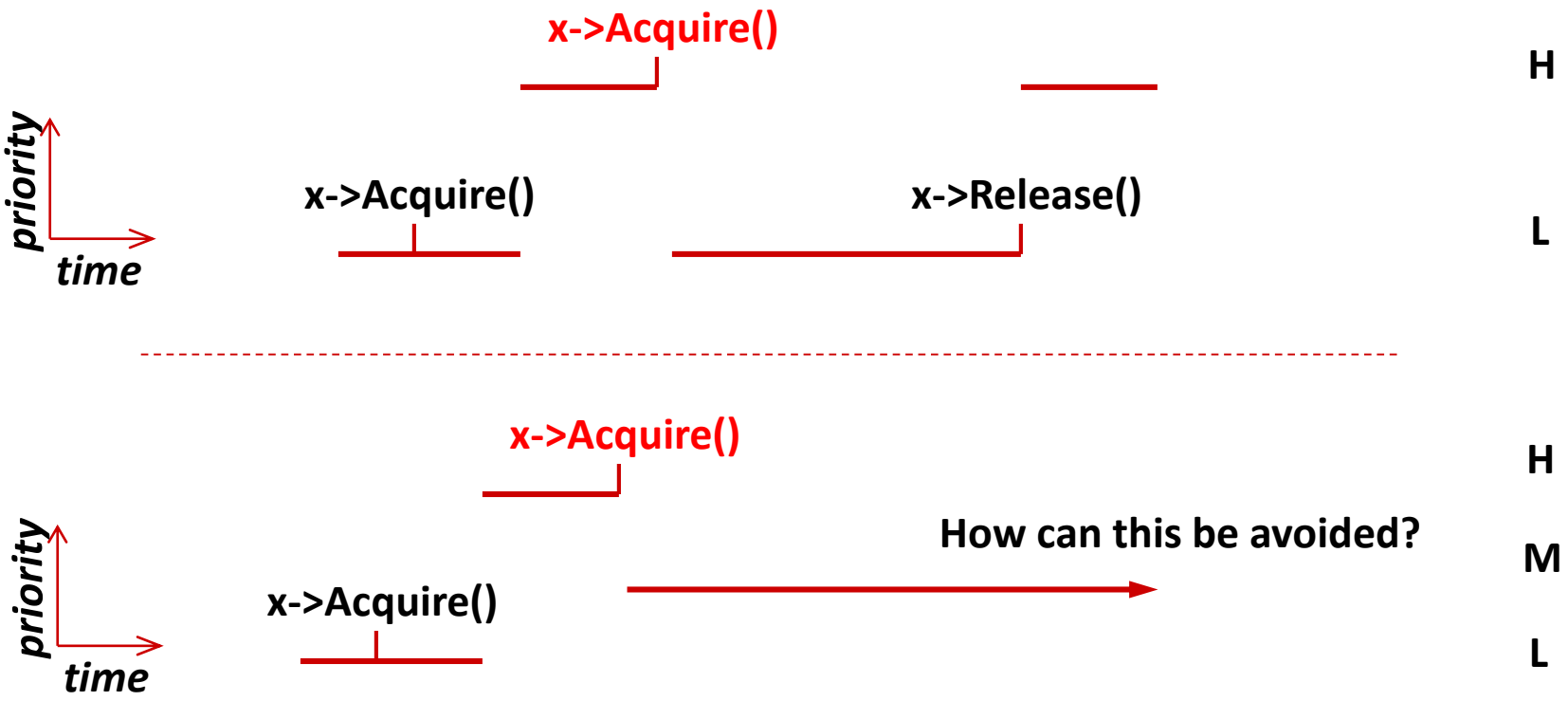


Be "nice" in Unix



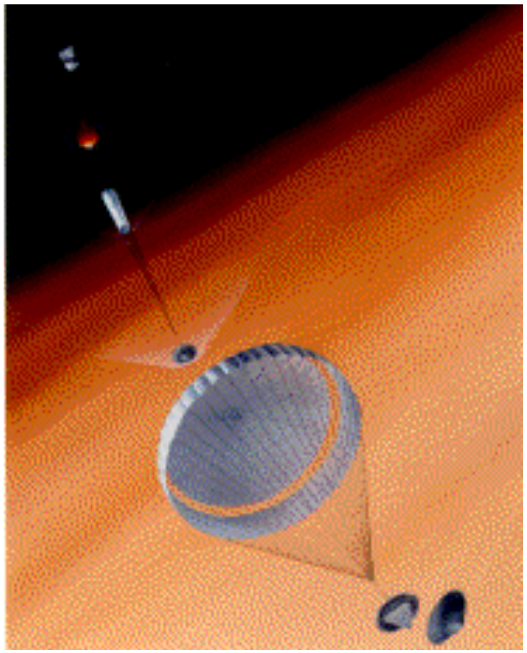
More on Priority Scheduling

- *Priority inversion* is a risk unless all resources are jointly scheduled



Scheduling on Mars

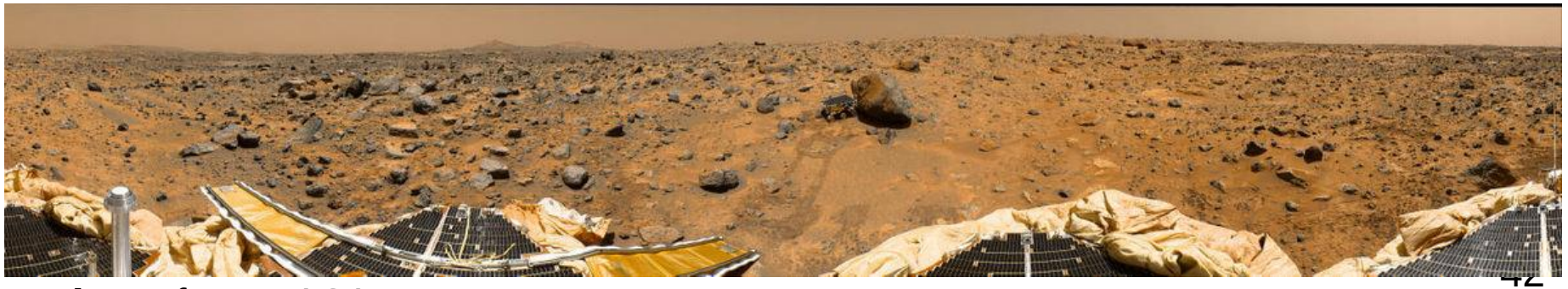
- What happened on Mars?



* Photos from NASA

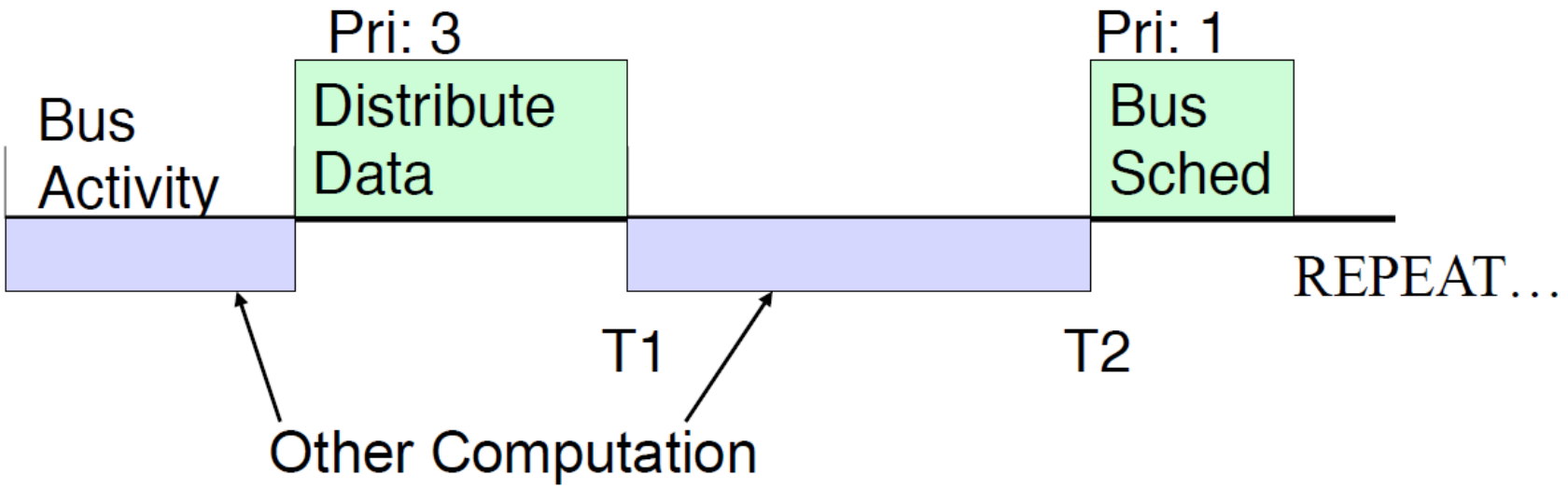
What Happened on Mars?

- Mars Pathfinder probe (1997)
- Nice launch (1996.12.4)
- Nice transit
- Nice de-orbit
- Nice thump-down with inflatable air-bag (1997.7.4)
- Nice rover disembarkation
- Nice *spontaneous reboots*! *(on July 5, 10, 11 and 14)*



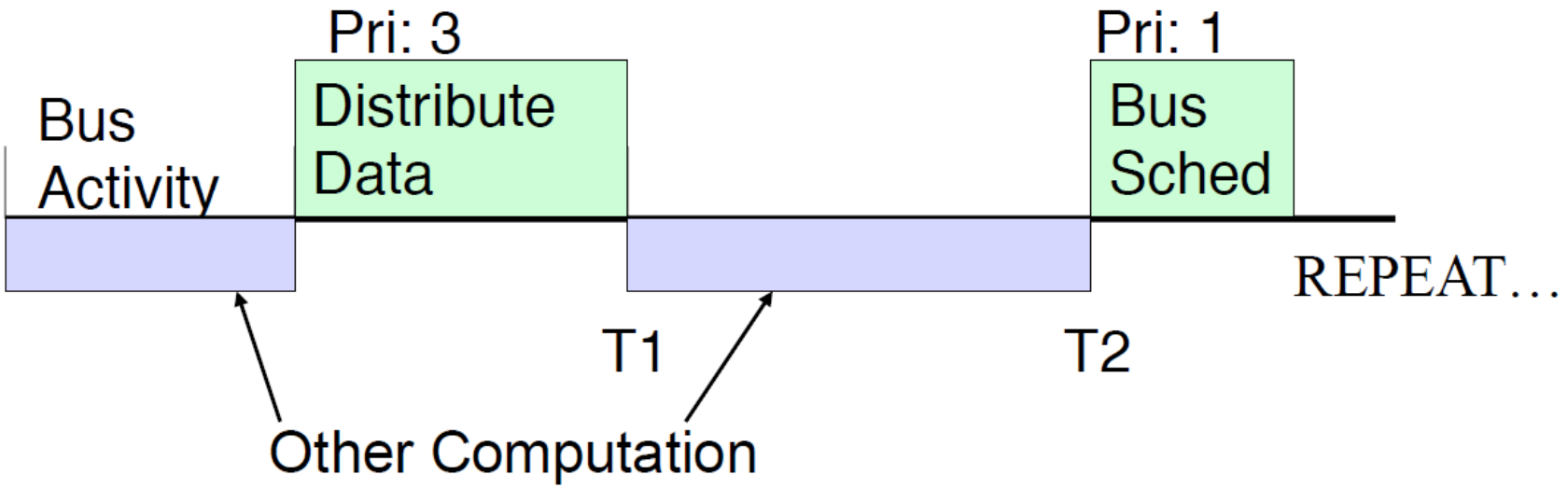
* Photos from NASA

Software Design



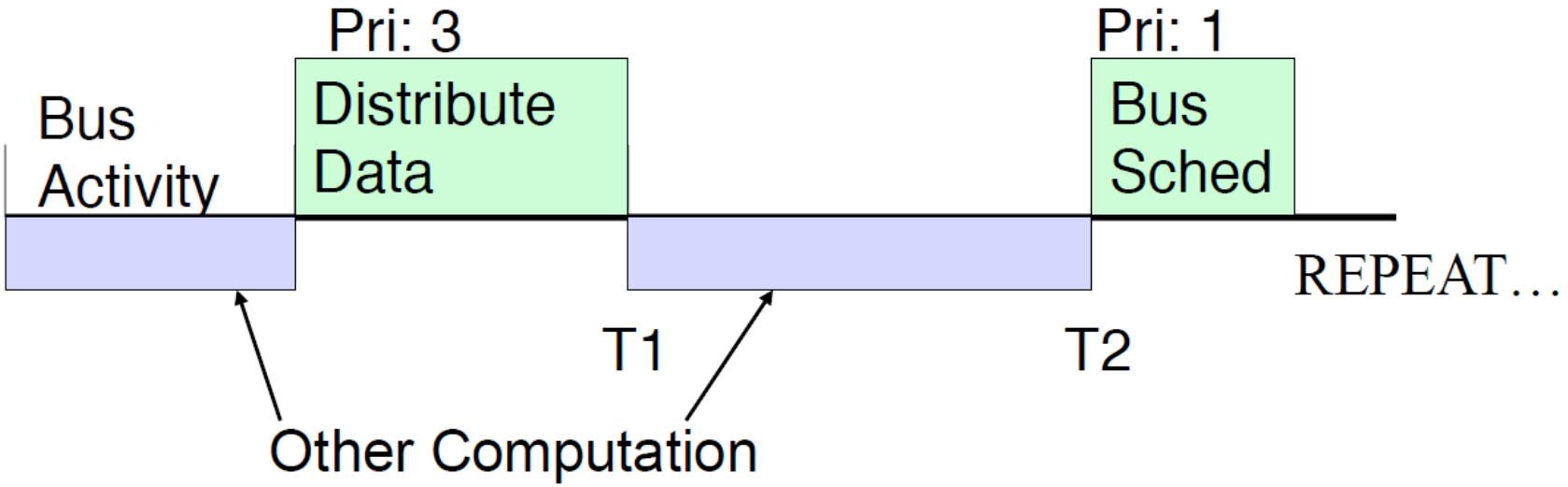
- T1 < T2 *or else system reboots!!!*

Software Design



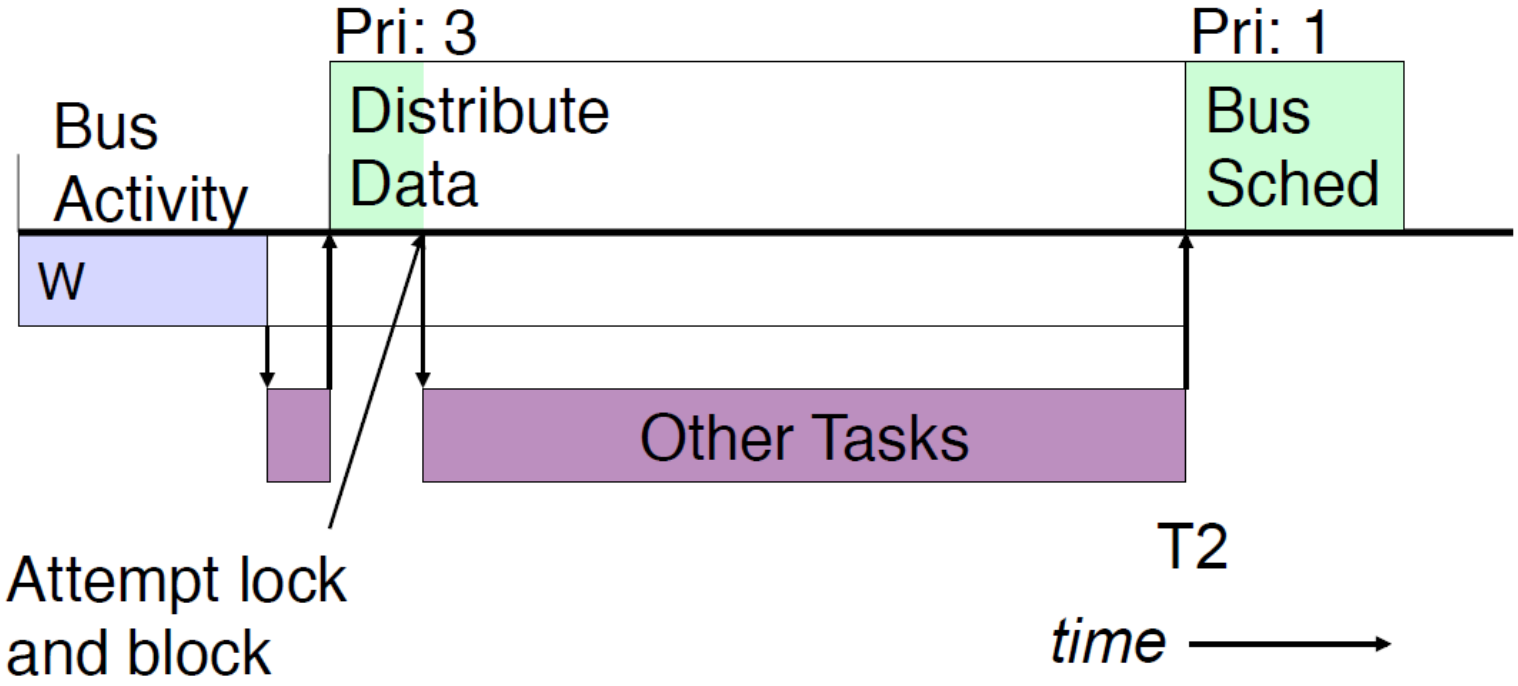
- Other computation (threads)
  - W (weather data thread): low priority
  - Many medium priority tasks
  - Distribute Data sends data to W via a software pipe facility

What Could Go Wrong?



- Weather thread (W) locks pipe to read data
- High-priority Distribute Data must wait to write data
- Interrupt makes other tasks runnable
  - Higher priority, so preempt W
  - W does not release lock for a long time…

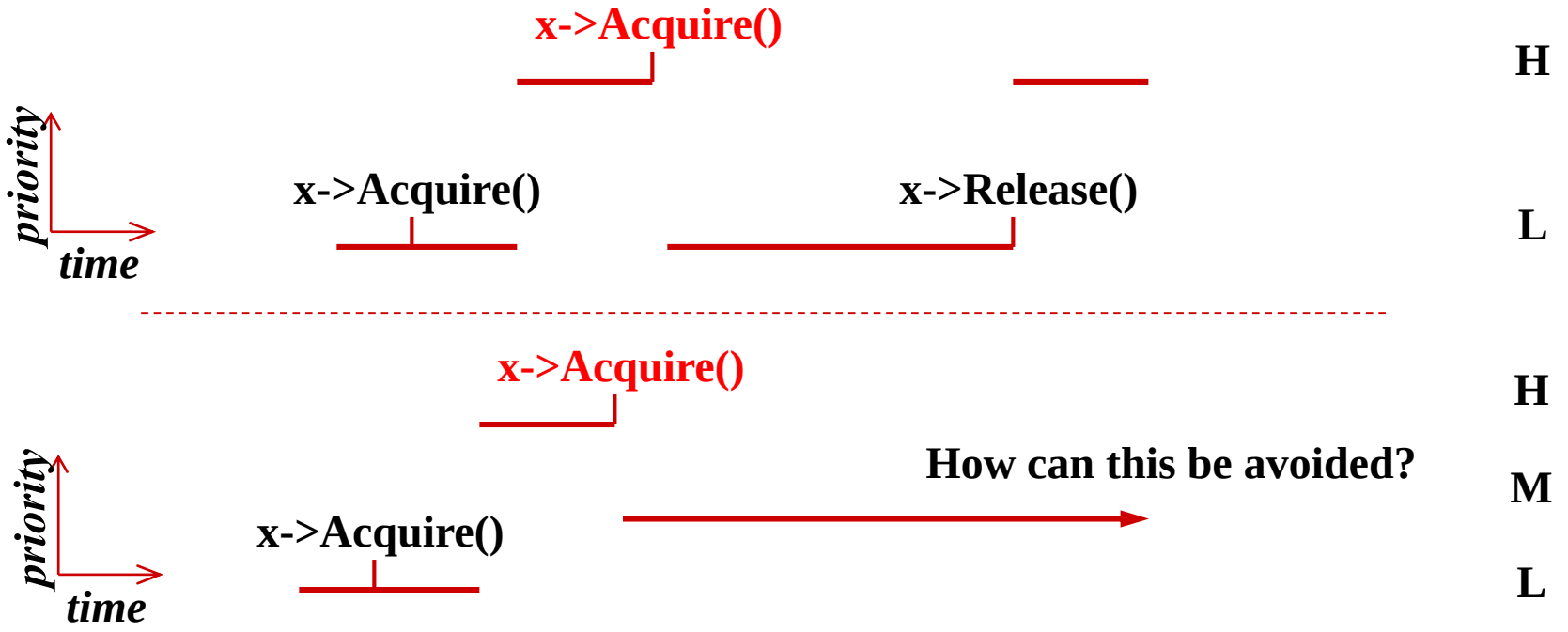
Priority Inversion



- T1 >= T2 : Oh no! *system reboots!!!*

A Solution: Priority Inheritance

- *Priority inversion*



- Priority inheritance
  - Lower-priority job inherits the priority of any higher priority job pending on a resource they share

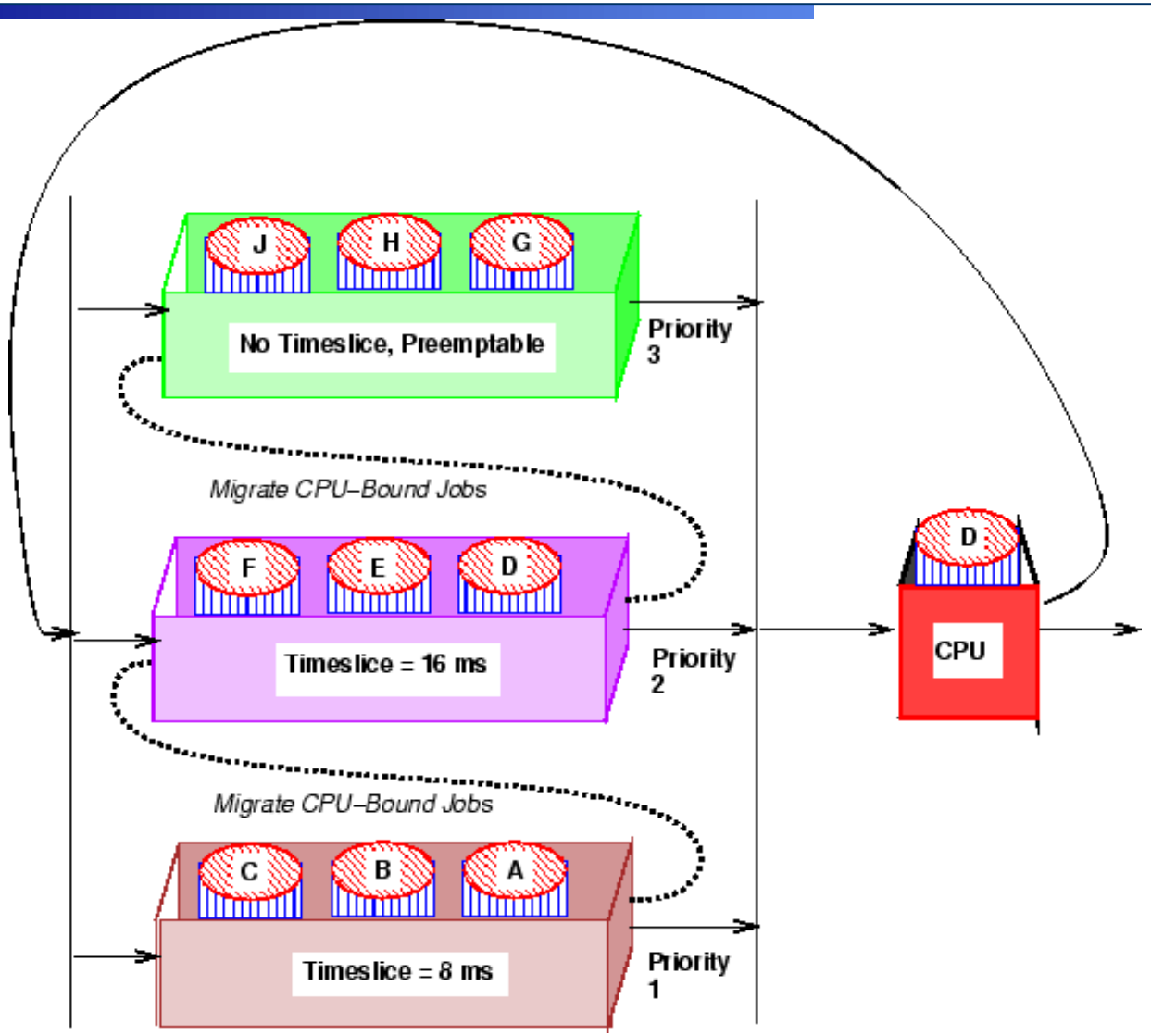
History of the Idea

- Priority Inheritance Protocols: An Approach to Real-Time Synchronization
  - IEEE Transactions on Computers 39:9 (1990.9)
    - 1987.12 “Manuscript” received
  - Lui Sha (CMU SEI), Ragunathan Rajkumar (IBM Research ⇒ CMU ECE), John Lehoczky (CMU Statistics)
  - Implemented in Wind River VxWorks RTOS
  - Rescues Mars Pathfinder (1997)
- History courtesy of Mike Jones and Glen Reeves (Debate)
  - http://www.cs.cmu.edu/~rajkumar/mars.html
  - http://www.cs.duke.edu/~carla/mars.html

Combining Algorithms

- Scheduling algorithms can be combined
  - Have multiple queues
  - Use a different algorithm for each queue
  - Move processes among queues
- Example: multiple-level feedback queues (MLFQ)
  - Multiple queues representing different job types
    - Interactive, CPU-bound, batch, system, etc.
  - Queues have priorities, jobs on same queue scheduled RR
  - Jobs can move among queues based upon execution history
    - Feedback: Switch from interactive to CPU-bound behavior

Multi-level Feedback Algorithm: Example



MLFQ in Unix

- The canonical Unix scheduler uses a MLFQ
  - 3-4 classes spanning ~170 priority levels (the higher the better)
    - Timesharing: first 60 priorities
    - System: next 40 priorities
    - Real-time: next 60 priorities
    - Interrupt: next 10 (Solaris)
- Priority scheduling across queues, RR within a queue
  - The process with the highest priority always runs
  - Processes with the same priority are scheduled RR
- Processes dynamically change priority
  - Increases over time if process blocks before end of quantum
  - Decreases over time if process uses entire quantum

Motivation of the Unix Scheduler

- The idea behind the Unix scheduler is to reward interactive processes over CPU hogs
- Interactive processes (shell, editor, etc.) typically run using short CPU bursts
  - They do not finish quantum before waiting for more input
- Want to minimize response time
  - Time from keystroke (putting process on ready queue) to executing keystroke handler (process running)
  - Don't want editor to wait until CPU hog finishes quantum
- This policy delays execution of CPU-bound jobs
  - But that's ok

Summary

- Scheduling overview
- Scheduling for batch systems
  - First Come First Serve (FCFS)
  - Shortest Job First (SJF)
- Interactive scheduling
  - Round Robin (RR)
  - Priority Scheduling
    - Priority inversion
    - Priority inheritance
  - Multiple-Level Feedback Queues (MLFQ)

- Next lecture: Multiprocessor and Real-Time Scheduling