15-410

"Arbitrarily Bad"

Real Time Systems

Oct 24, 2008

Roger B. Dannenberg

Dave Eckhardt

Additional material by Vishakha Gupta

Scheduling on Mars

What happened on Mars?

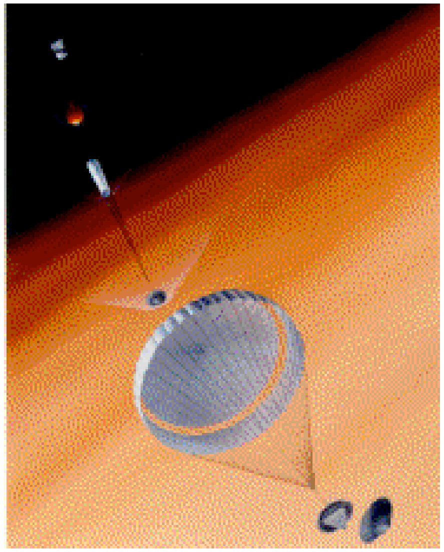


Photo credits: NASA

What Happened On Mars?

Mars Pathfinder probe (1997)

Nice launch

Nice transit

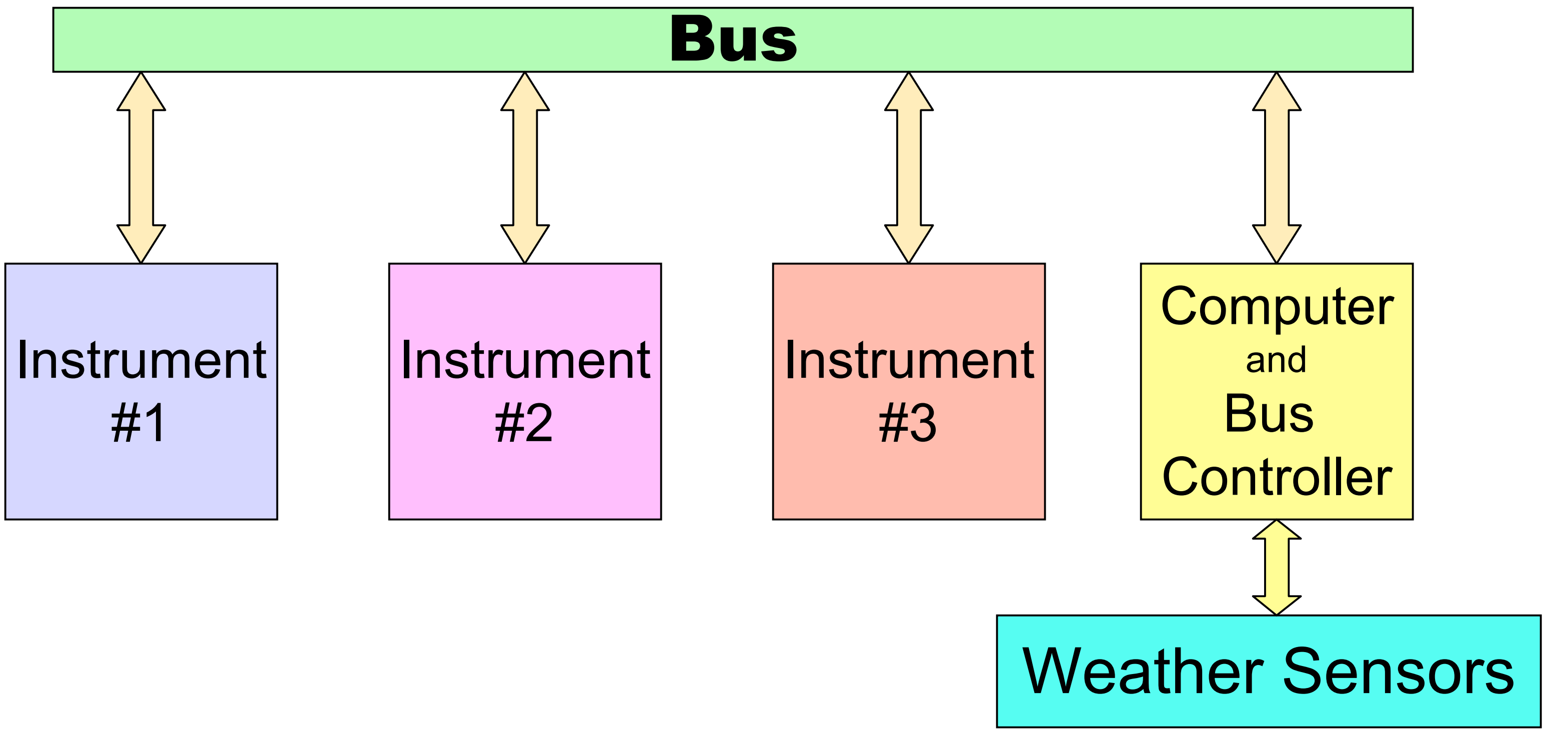
Nice de-orbit

Nice thump-down (inflatable air-bag)

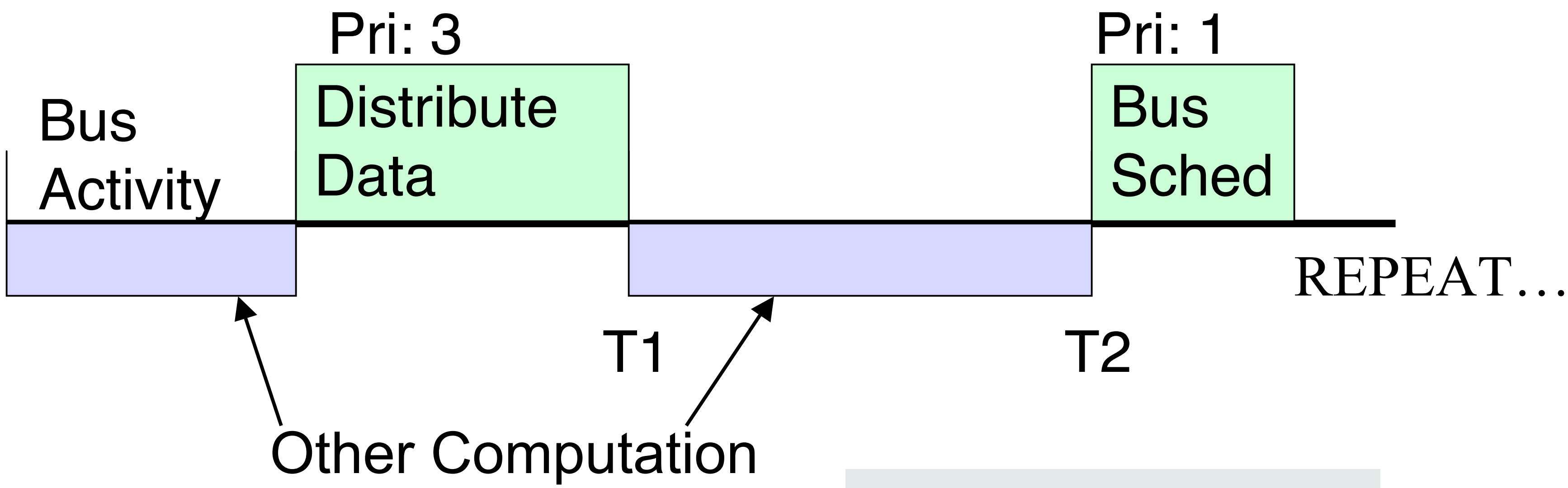
Nice rover disembarkation

Photo credits: NASA

Hardware Design

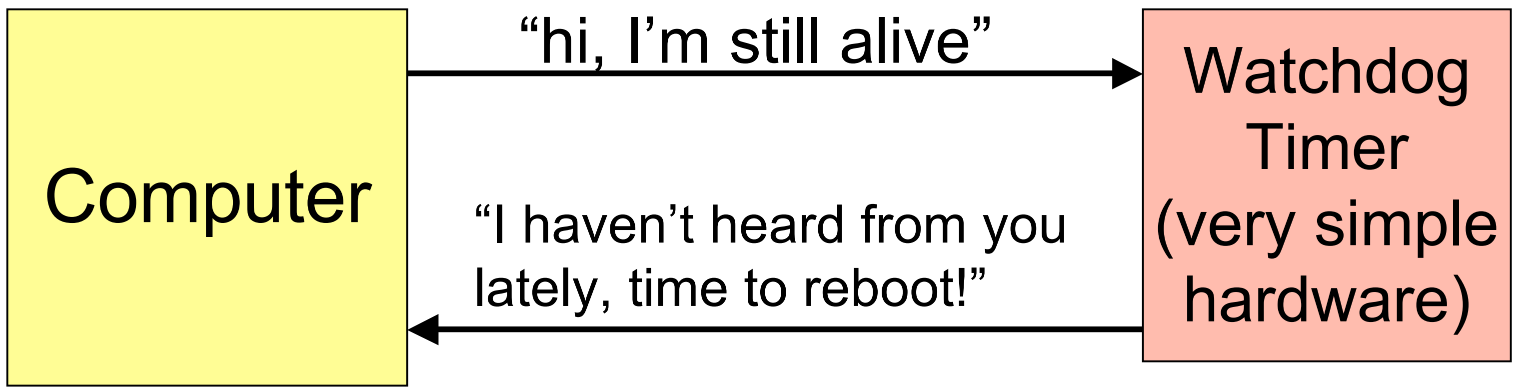


Software Design

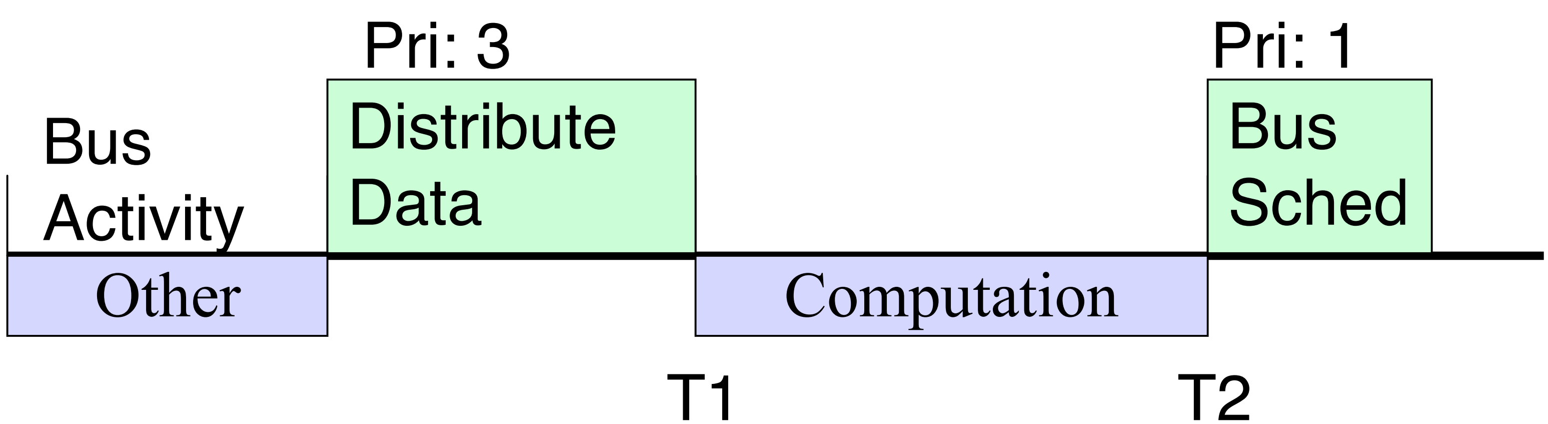


So: transfer data
compute
transfer data
compute
…

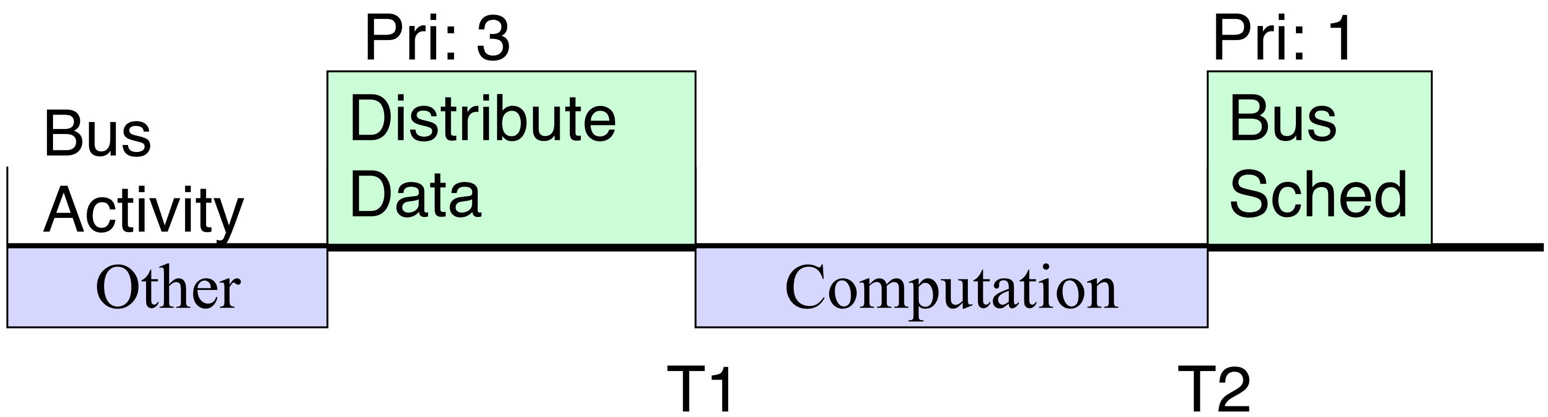
Watchdog Timer



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



Software Design



Other 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

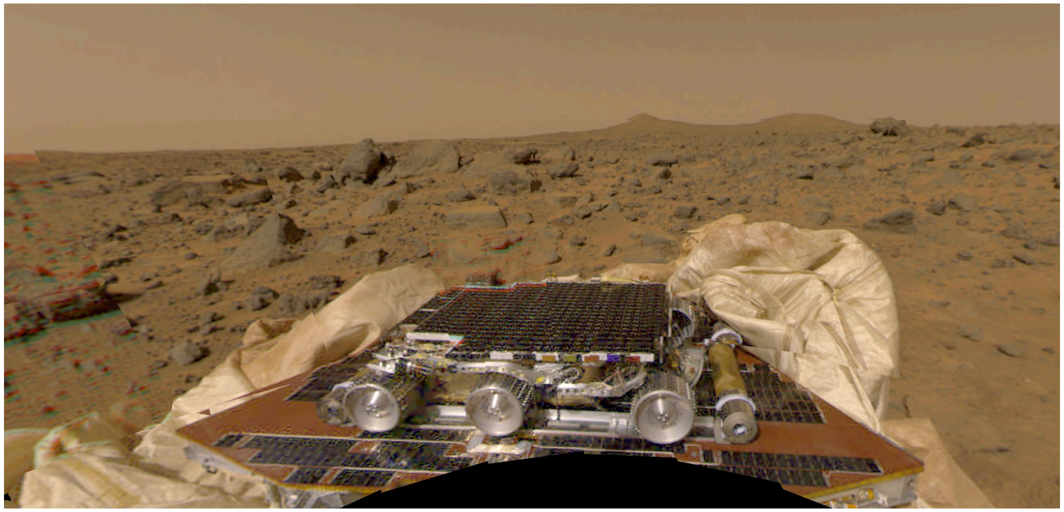


Photo credit: NASA

What could go wrong?

W locks pipe structure to read message

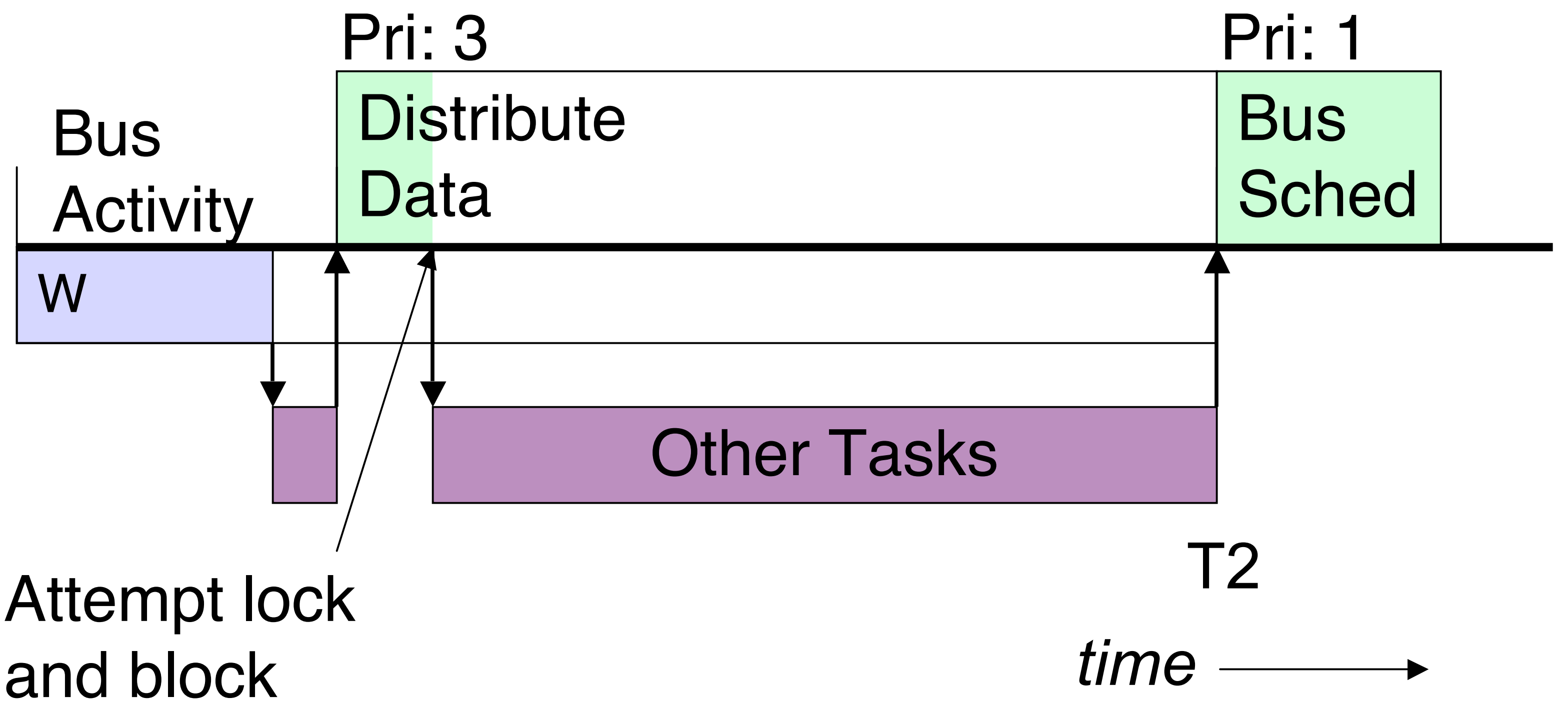
Interrupt makes other tasks runnable

- Higher priority, so preempt W
- W does not release lock for a long time…

Distribute Data becomes runnable

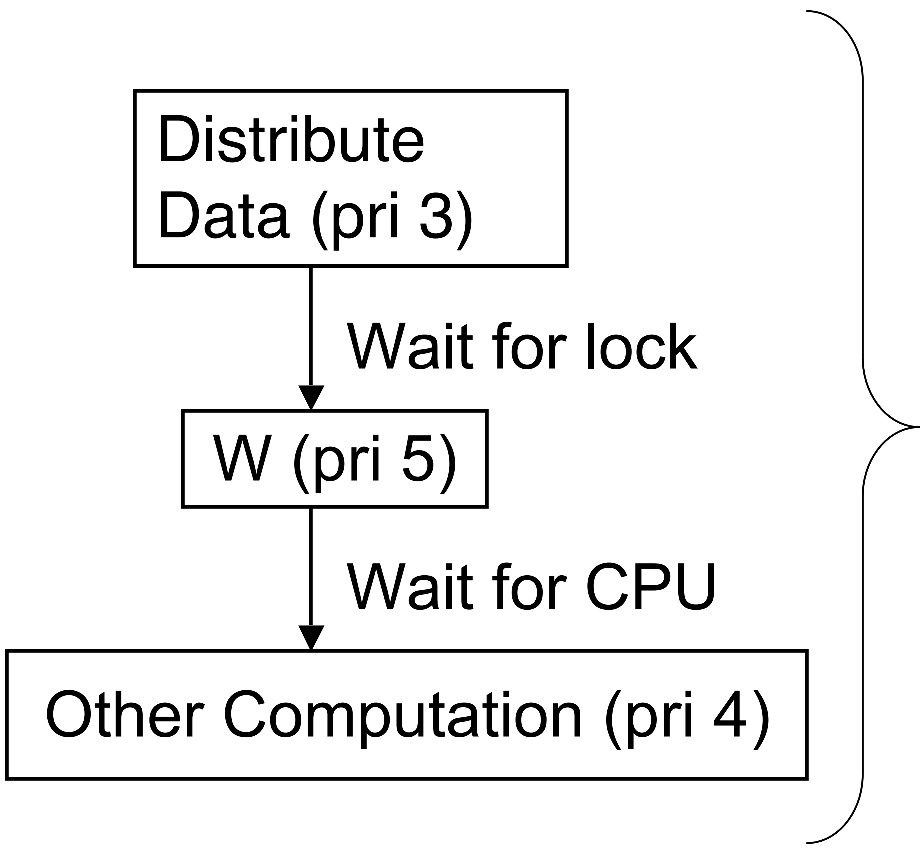
- Very high priority, so preempts other tasks
- Distribute Data tries to send data to W, but blocks
- Other tasks resume, run for a long time…

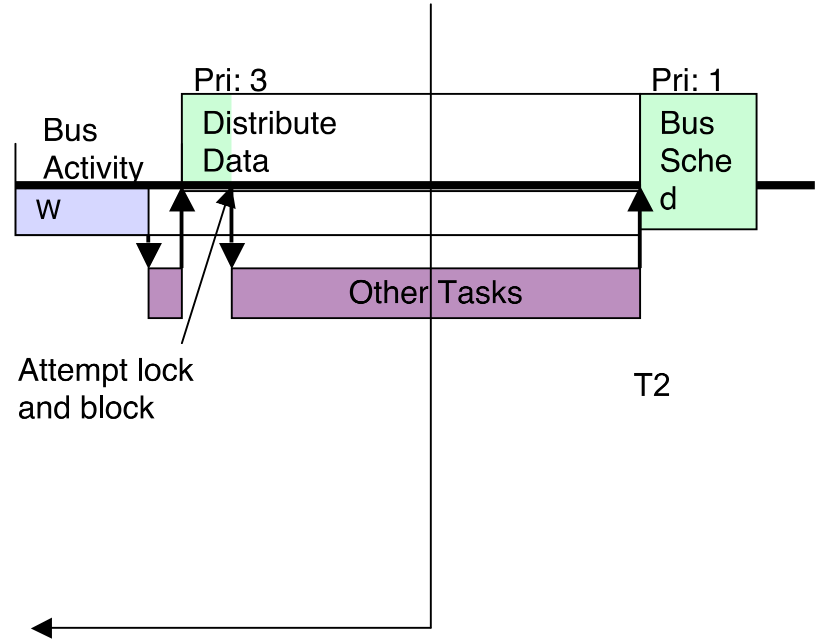
Priority Inversion



T1 ≮ T2 : Oh no! *system reboots!!!*

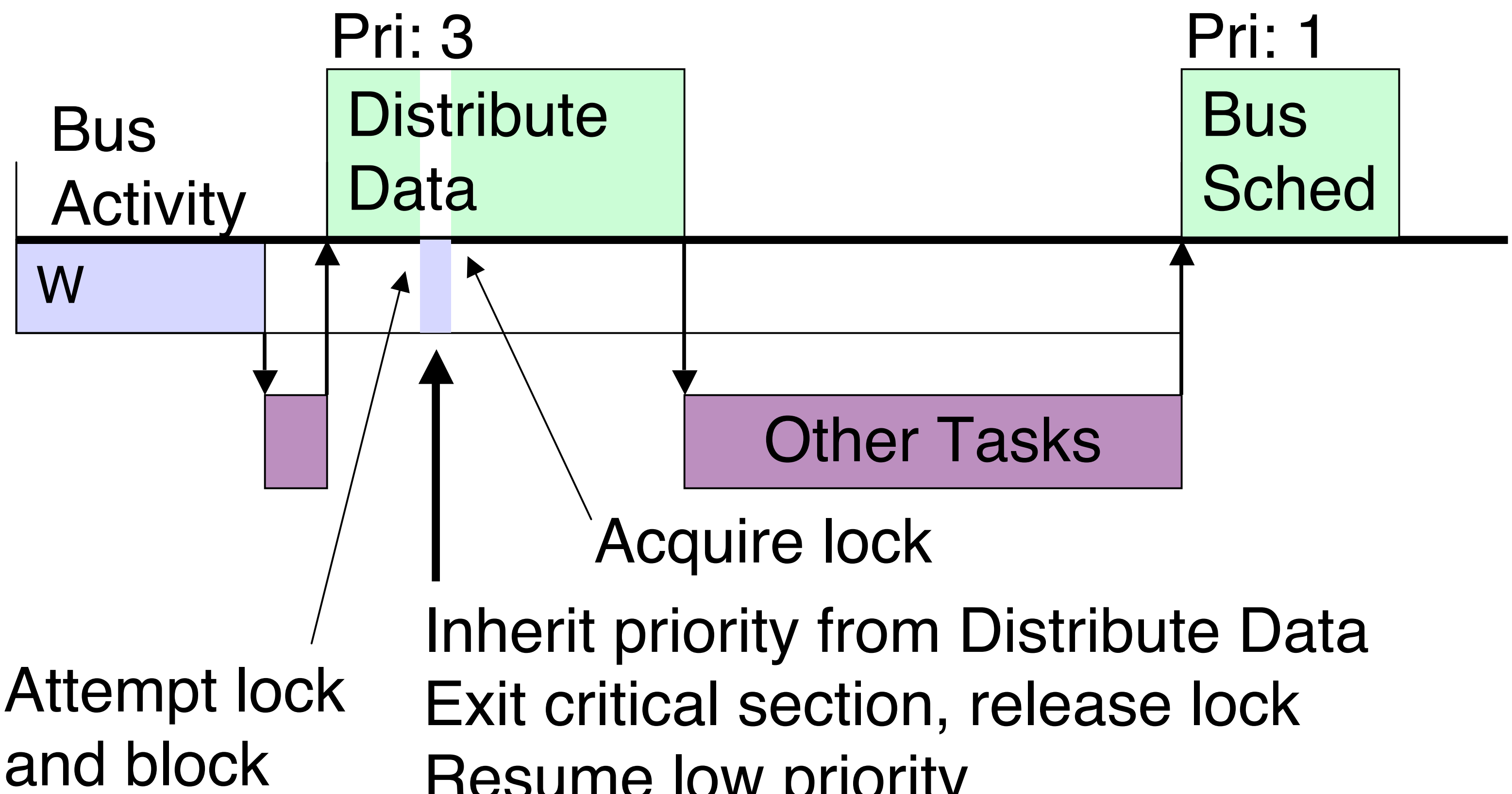
Priority Inversion





What if W could "borrow" Distribute Data's priority?

Priority Inheritance



History of an Idea

Priority Inheritance Protocols: An Approach to Real-Time Synchronization

- IEEE Transactions on Computers 39:9
  - Lui Sha (CMU SEI)
  - Ragunathan Rajkumar (IBM Research ⇒ CMU ECE)
  - John Lehoczky (CMU Statistics)

History of an Idea

Events

- 1987-12 “Manuscript” received
- 1988-05 Revised
- 1990-09 Published
- 1997-07 Rescues Mars Pathfinder

History courtesy of Mike Jones and Glen Reeves

- http://www.cs.cmu.edu/~rajkumar/mars.html
- http://www.cs.duke.edu/~carla/mars.html

Test Your Understanding

What could go wrong with an atomic exchange/spin lock?

Assume threads have fixed priorities.

Explain how priority inversion could arise from a call to malloc.

Real-Time Systems

Types of Systems

Rate Monotonic Scheduling

Earliest Deadline First Scheduling

Priority Inversion

Real-Time Audio Application/OS Interactions

Embedded Systems Scheduling

One Big Loop

- Polled I/O
- One thread: while (true) { task1(); task2(); … }

Time-driven: wait for next period at top of loop

Multiple threads

- Round-Robin, or
  Time-driven: run tasks at fixed frequencies
- Can incorporate interrupt-driven I/O

Static Priority-based Scheduling/Rate Monotonic

Deadline Scheduling

Rate Monotonic Scheduling

A method of assigning fixed priorities to a set of periodic processes

Higher rate (frequency) ⇒ Higher priority

Formal framework for reasoning about schedulability

Schedulable if:

preemption + execution + blocking < deadline

=> Schedulability is a key question for designers <=

Assumptions

Periodic tasks

Tasks become ready to execute at beginning of their periods

Tasks runnable until execution is complete (1 burst)

Task deadlines are always start of next period

No task is more important/critical than another

Tasks account for all execution time

- Task switching is instantaneous
- No interrupts

Schedulability Tests

Utilization Bound Test - fast, conservative

Response Time Test - slower, exact

Utilization

Computation time: C_i

Period: T_i,

Utilization: $U_i = C_i/T_i$

Total Utilization: $\sum U_i$

Note that $0 < \sum U_i < 1$

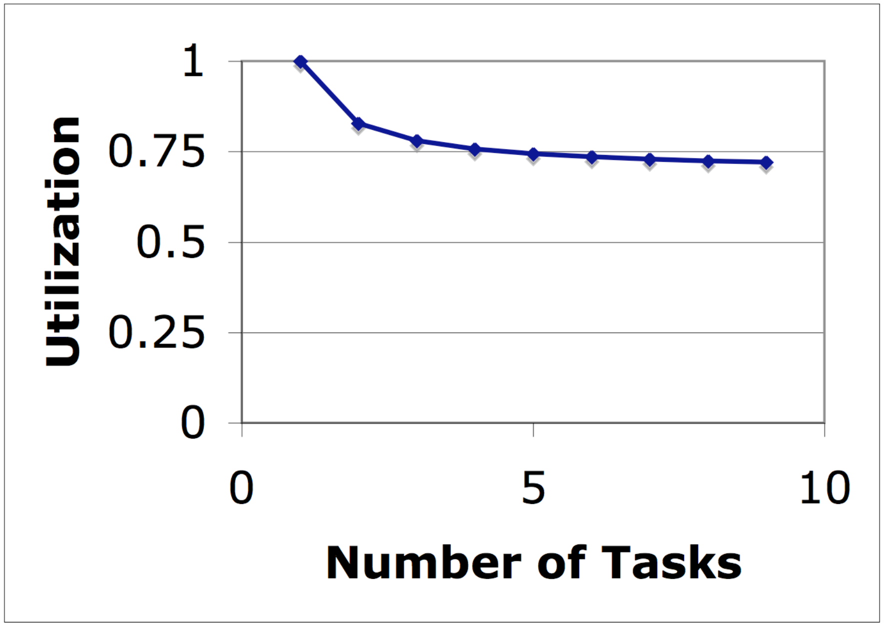
Example

	C	T	U
Task τ_1:	20	100	0.200
Task τ_2:	40	150	0.267
Task τ_3:	100	350	0.286

Example from 14342 – Fundamentals of Embedded Systems

Total utilization for 3 tasks is .200 + .267 + .286 = .753

Utilization Bound Test



Are all my tasks schedulable?

Rate Monotonic Scheduling:

- **Utilization for n tasks: $U(n) = n(2^{1/n} - 1)$**
- **This is a *worst case (lower)* bound**

Test: $(\sum U_i) < U(n)$

Example

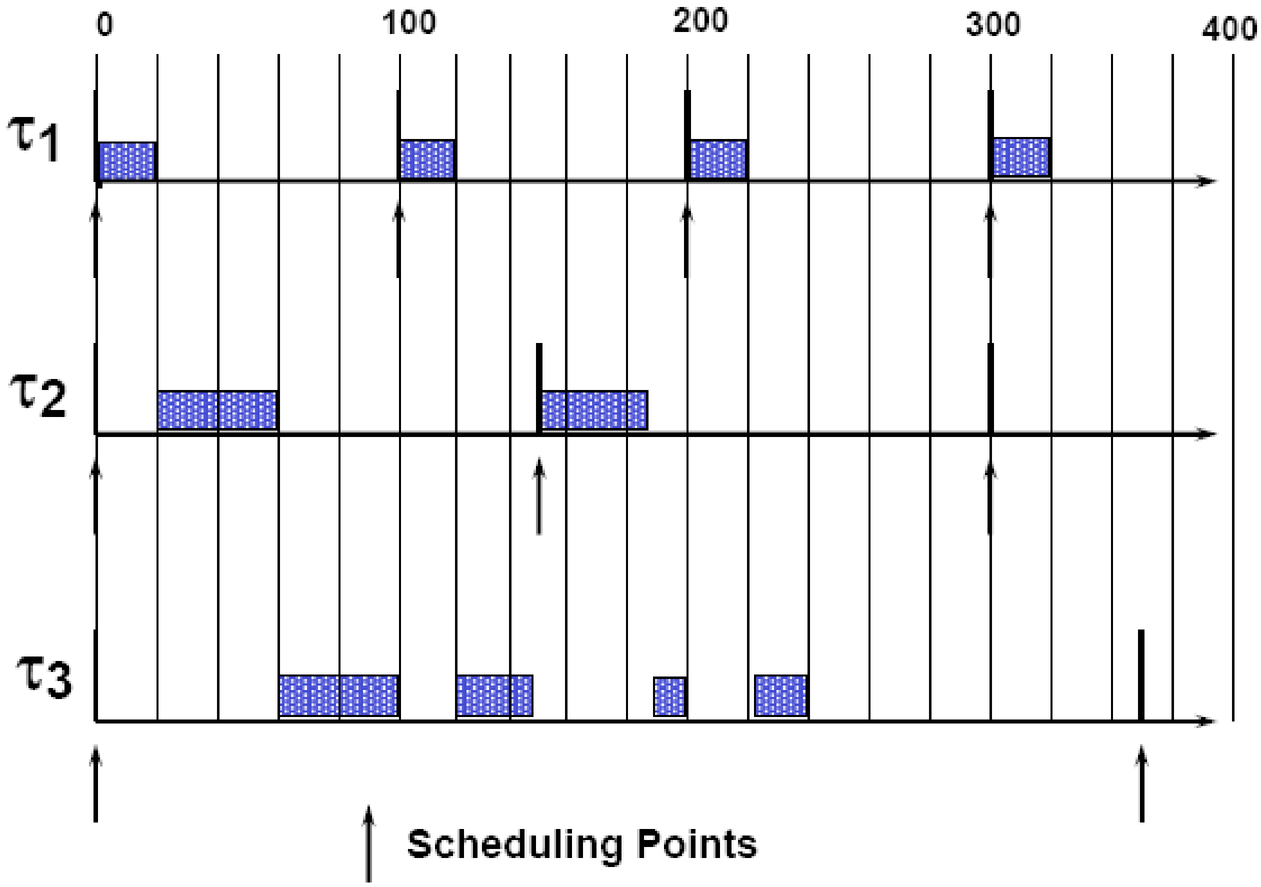
	C	T	U
Task τ_1:	20	100	0.200
Task τ_2:	40	150	0.267
Task τ_3:	100	350	0.286

Example from 14342 – Fundamentals of Embedded Systems

Total utilization for 3 tasks is .200 + .267 + .286 = .753
U(3) = .779

Total utilization for 3 tasks < U(3)
The periodic tasks in the sample problem are schedulable
According to the upper bound (UB) test

Timeline for the example



Response Time Test

Theorem: For a set of independent periodic tasks, if each task meets its deadline with worst case task phasing, the deadline will always be met

System *might* be schedulable with utilization > U(n), but it depends on the particular task mix

Rate Monotonic Extensions

Blocking:

- **preemption + execution + *blocking* < deadline**

Interrupt tasks

Addition/Deletion of tasks

Aperiodic tasks with computational budget

Earliest Deadline First

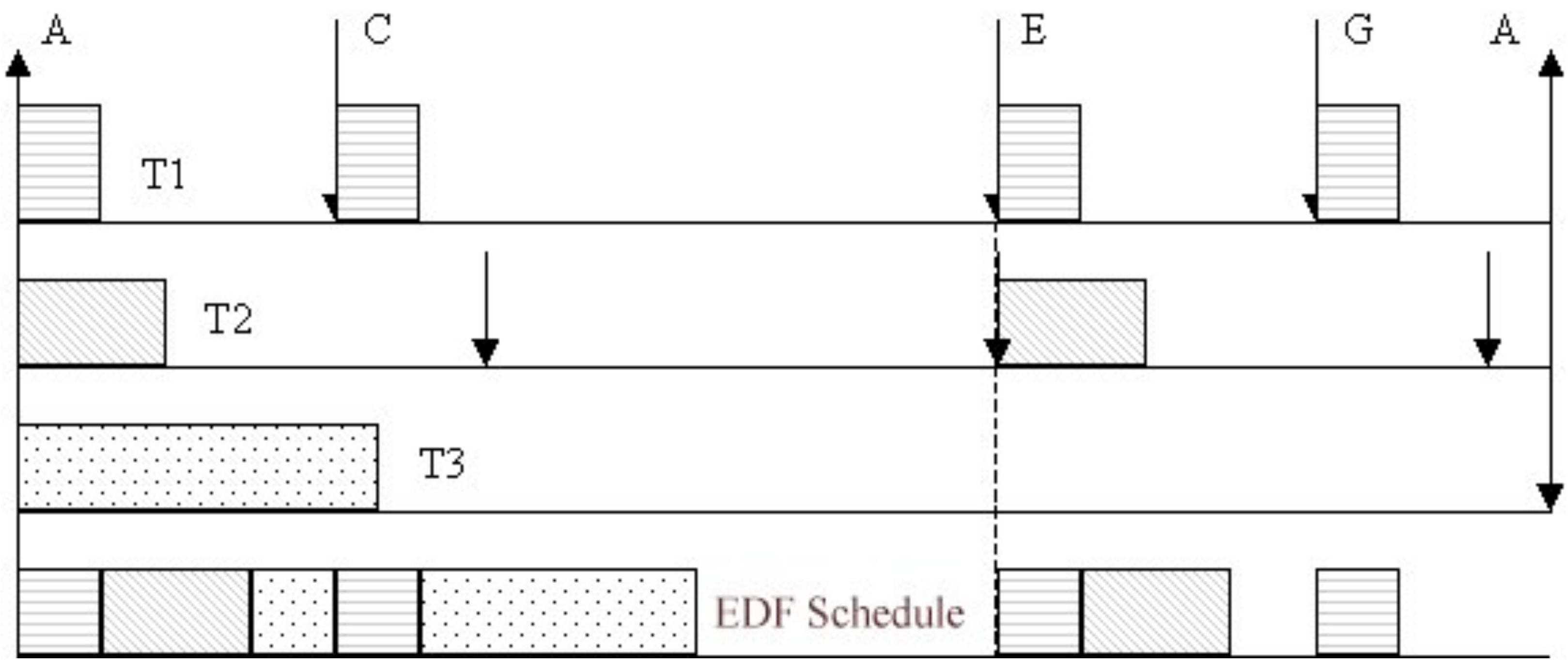
A dynamic scheduling principle

Assume independent tasks

Tasks in a priority queue, ordered by deadline

With periodic processes with deadlines = periods, EDF has a utilization bound of 100% (*optimal*)

Example



Pros and Cons

+ Optimal for schedulable task set
+ Task set need not be periodic
+ Deadlines need not equal periods
– Overload behavior can be arbitrarily bad
– Considered more difficult to implement than static priority schemes

Rate Monotonic vs. Earliest Deadline First

Rate Monotonic

- More widely supported
- Maps onto static priority schedulers (NT, CE, Linux, OS X)

Earliest Deadline First

- Sometimes higher utilization
- Less restrictive assumptions

Neither is really complicated

- If you have a well-defined problem, analysis is straightforward
- If not, think carefully about failure modes (which are different) and costs

Real World/Real Time Audio

What do you have to work with?

What are the implications?

Putting it together.

What performance can you get?

Audio: What Can You Assume?

Potential for priority inversion

System response time is an issue:

system_latency + preemption + execution + blocking < deadline

Static Priority Scheduling

(At least) two application classes:

- High audio latency (iTunes, sound effects, audio editor)
  - Compute audio well ahead (>100 ms)
  - Leave it to device driver to deliver samples on time
- Low audio latency (VoIP, Guitar Hero, real-time music synthesis)
  - Audio depends on real-time input
  - Only compute 1-10ms ahead of time
  - User-level application scheduling is critical

Low Audio Latency Implications

Need to use static priority scheduling ⇒

- 1ms to compute audio < 10ms to refresh display

Priority Inversion *is* a problem ⇒

No locking ⇒

No shared data structures ⇒

Threads communicate via lock-free FIFO

No malloc ⇒

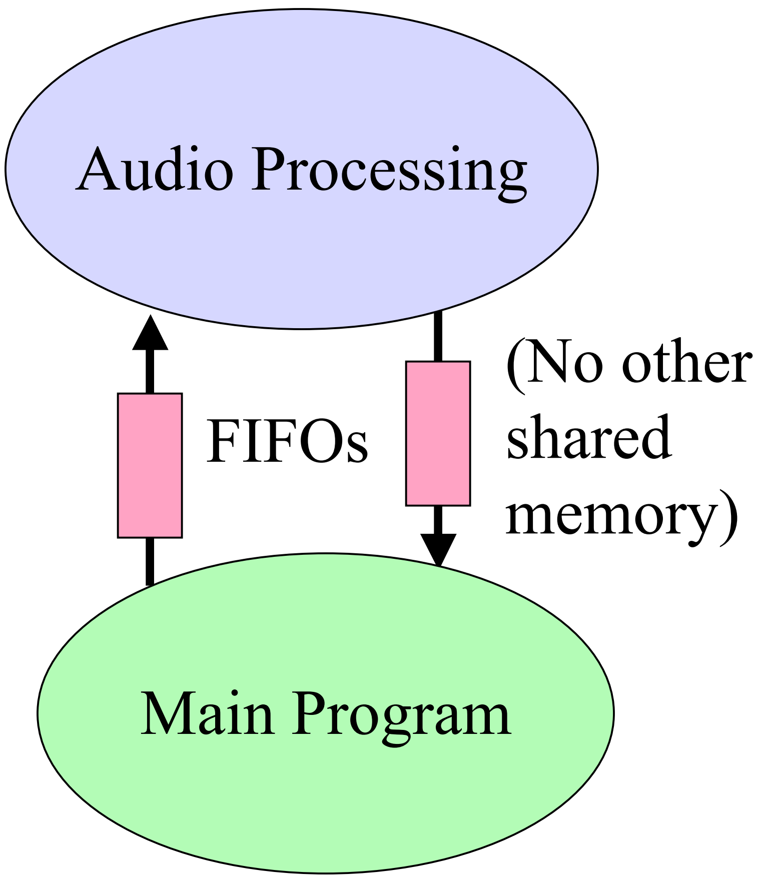
independent memory pool per thread

OR only lock-free shared structures ⇒

No malloc ⇒ write your own

+ lots of synchronous polling for I/O

Putting It Together




```
while (true) {
    audio_read(&buf);
    while (!input.empty())
        process_input();
    process_audio(&buf);
    // maybe send data to
    // output fifo
    audio_write(&buf);
}
```

What Performance Can You Get?

Current audio applications can deliver end-to-end latencies in the 3 to 10ms range.

Note: “native” windows audio is quite poor, but 3rd party (ASIO) drivers exist to improve performance.

A big issue is “system latency”:

- SGI/Irix was a leader: hard real-time kernel
- Linux has evolved rapidly (now <1ms)
- OS X: special real-time threads for audio
- Windows: worst-case system latency is high

Summary

Priority Inversion

Real-Time Scheduling

- Rate Monotonic
- Earliest Deadline First

Implications of Real-World OS on Real-Time Applications

- Polling
- Locks limited by Priority Inversion
- (Un)shared memory

Further Reading

Comparing Rate Monotonic to Earliest Deadline First:

- Giorgio C. Buttazzo, “Rate Monotonic vs. EDF: Judgment Day,” *Real Time Systems* 29(1) (Jan 2005), The Netherlands: Springer, pp 5-26.