Evolving Real-Time Systems using Hierarchical Scheduling and Concurrency Analysis

John Regehr   Alastair Reid

Kirk Webb   Michael Parker   Jay Lepreau

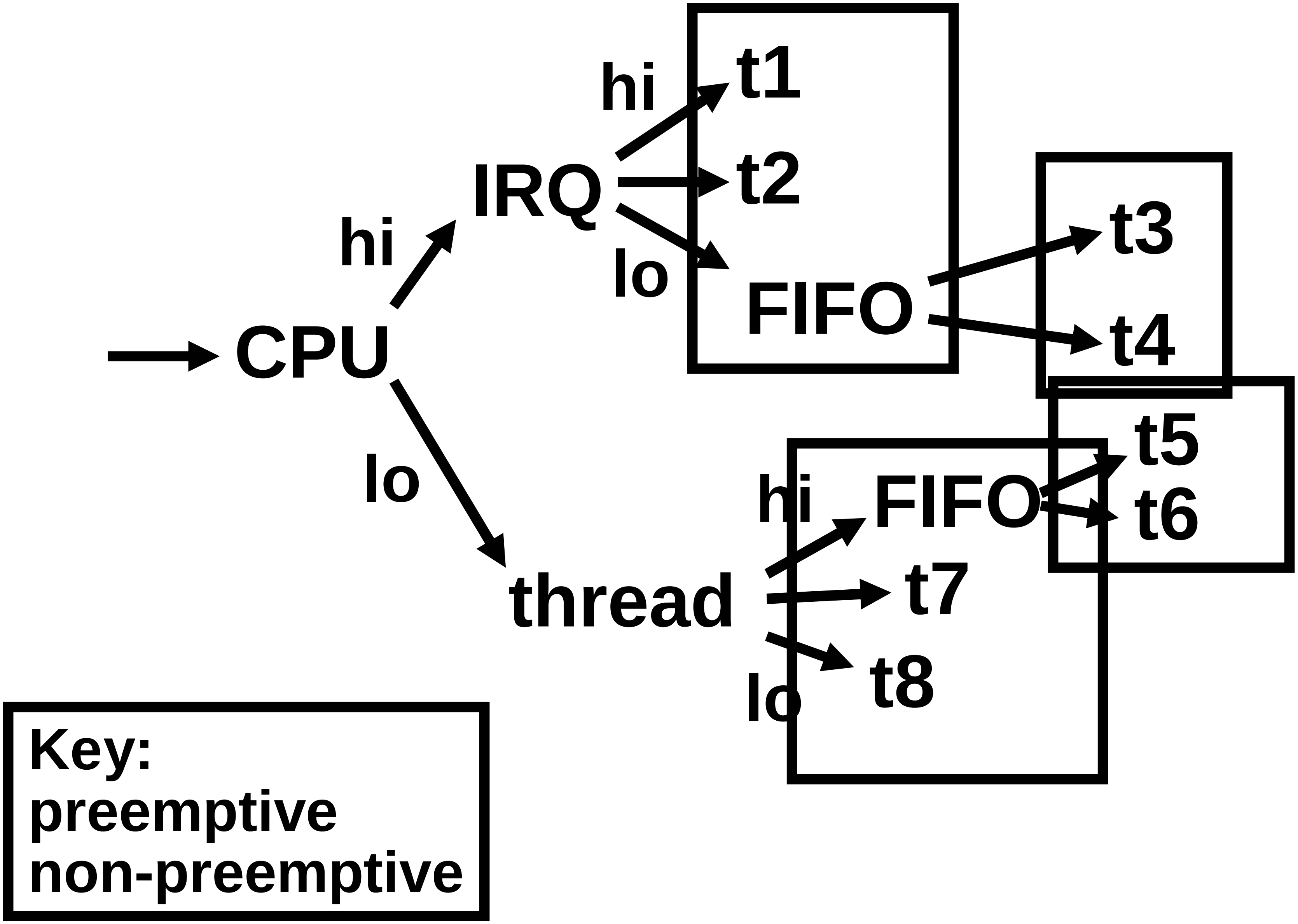
School of Computing, University of Utah

- Problem: Evolving real-time and embedded software is hard

- Problem: Concurrent software is hard to write and debug

- Problem: Traditional task models ignore some important details about real systems

A Task Set



CPU

hi

IRQ

hi

t1

t2

lo

FIFO

t3

t4

lo

thread

hi

FIFO

t5

t6

t7

lo

t8

Key:
preemptive
non-preemptive

This Talk

- **Introduces hierarchical execution environments to support analysis of:**
  - **Concurrency**
  - **Response times**
  - **Blocking terms**
  - **Dispatch overheads**
- **Results in solving real-time problems on sensor network nodes**

Execution Environments

- **A real-time or embedded system usually supports multiple execution environments**
  - **Interrupts**
  - **Bottom-half handlers**
    - **a.k.a. DPCs, tasklets, deferred handlers**
  - **Event handlers**
  - **Kernel threads**
  - **User threads**

An Execution Environment…

- **Occupies a place in the scheduling hierarchy**
- **Has particular performance characteristics**
- **Has rules:**
  - **About actions code running in it may take**
  - **About how to synchronize with code in other environments**

Related Work

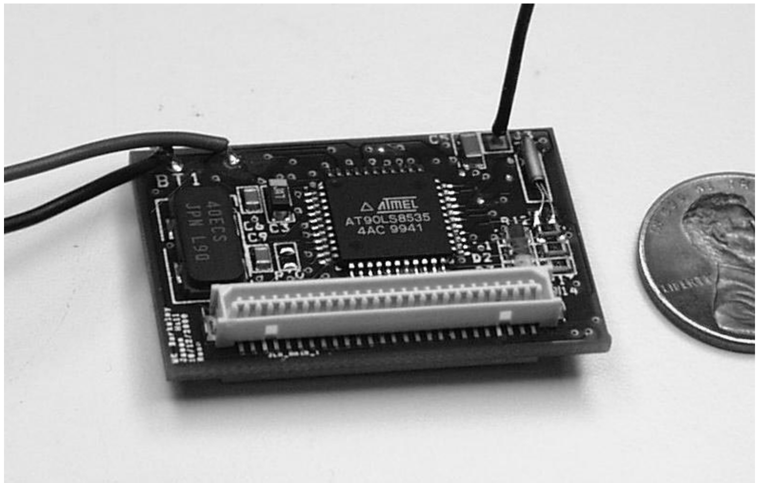
- Hierarchical scheduling
  - Lots of work: Deng et al., Feng & Mok, Lipari et al., Regehr & Stankovic, Saewong et al., Shin & Lee, …
- Multiple execution environments
  - Limited related work here
- Concurrency analysis
  - Lots of work in PL and formal methods communities – but none supporting multiple execution environments

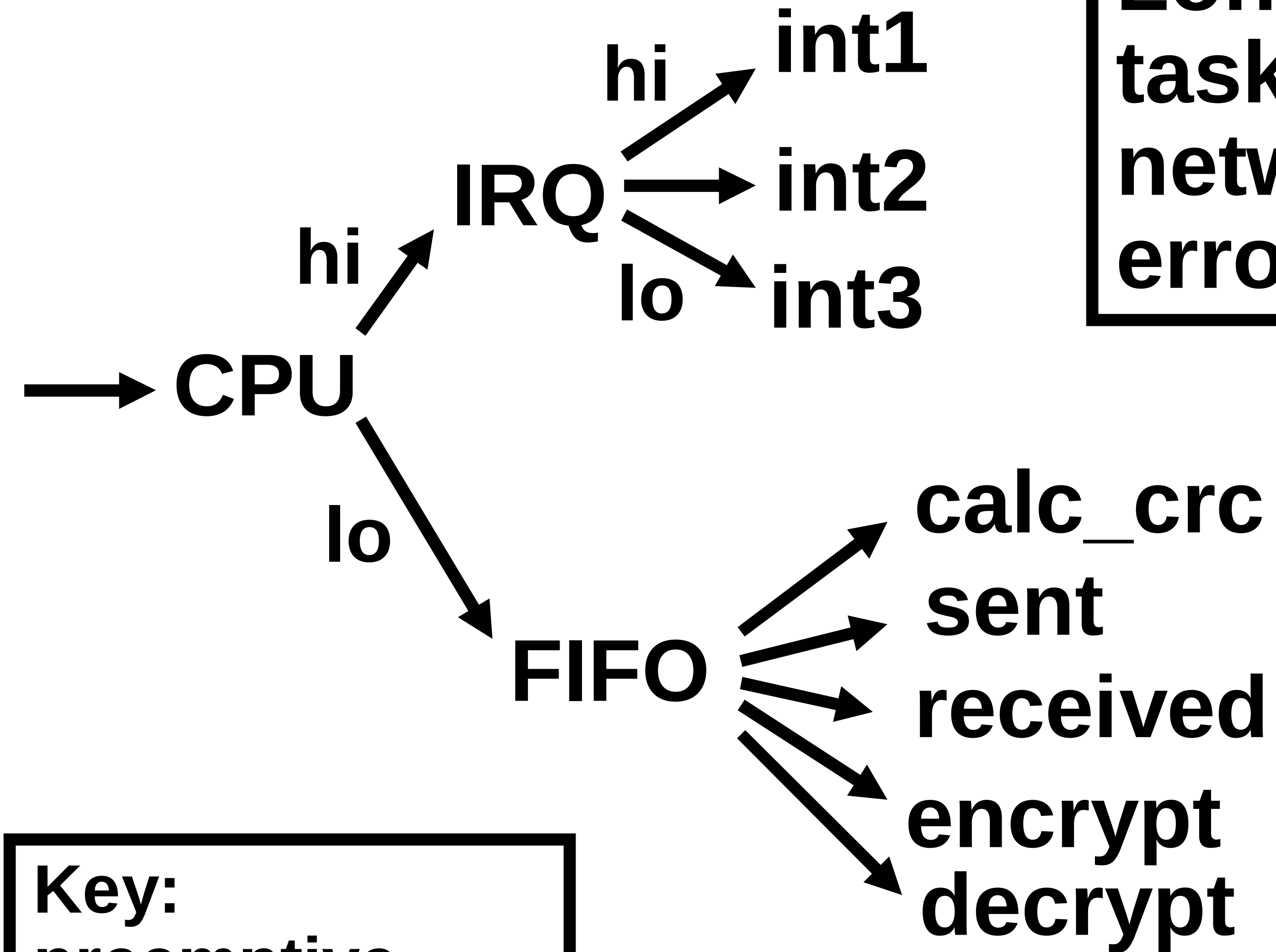
Goal: Evolving Systems

- Often desirable to move code between environments
  - "Promote" code to a higher priority environment
  - "Demote" code to a lower priority environment
- Problem: How do we know when to promote / demote code?
- Problem: Very easy to introduce concurrency errors this way
  - May be lots of code per environment

Example System: Motes

- Sensor network nodes
- Software based on TinyOS
  - Very simple “OS”
  - No threads!
- Motes are resource constrained
  - 4 MHz 8-bit RISC
  - 4 KB SRAM, 128 KB flash

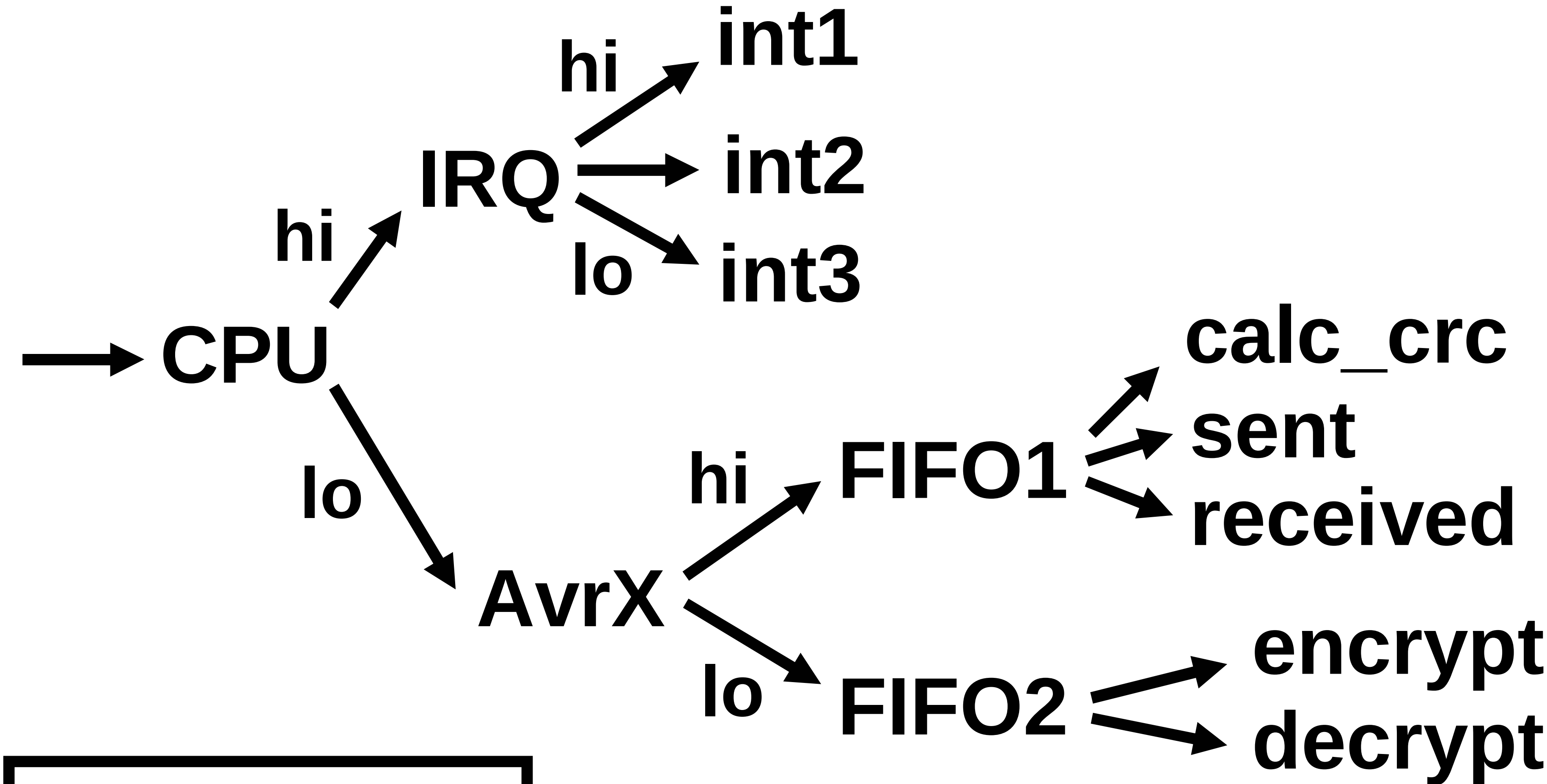




Problem 1: Long-running tasks cause network errors

Key:
preemptive
non-preemptive

Fixed TinyOS 1



CPU

hi

IRQ

hi

int1

int2

lo

int3

lo

AvrX

hi

FIFO1

calc_crc

sent

received

lo

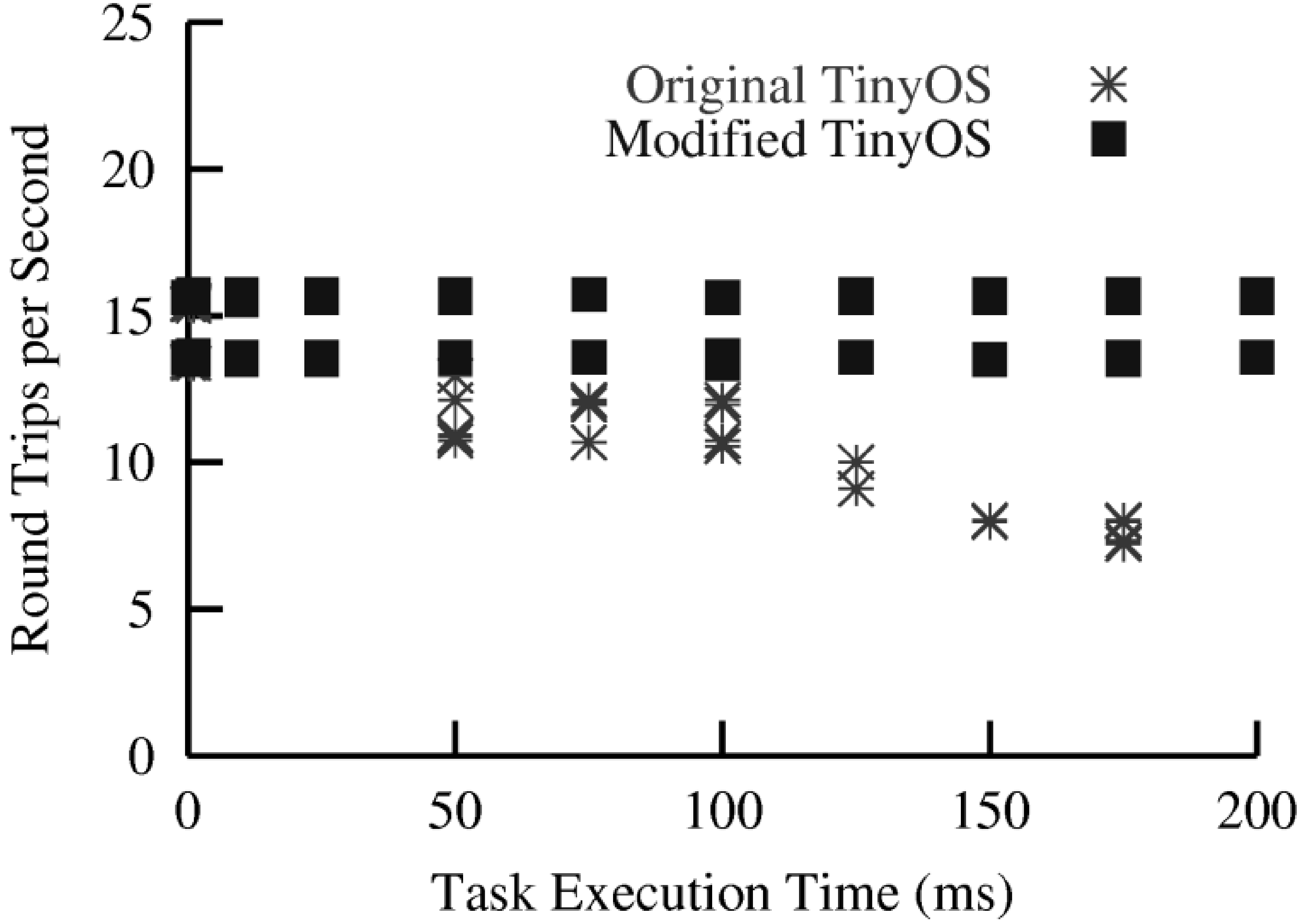
FIFO2

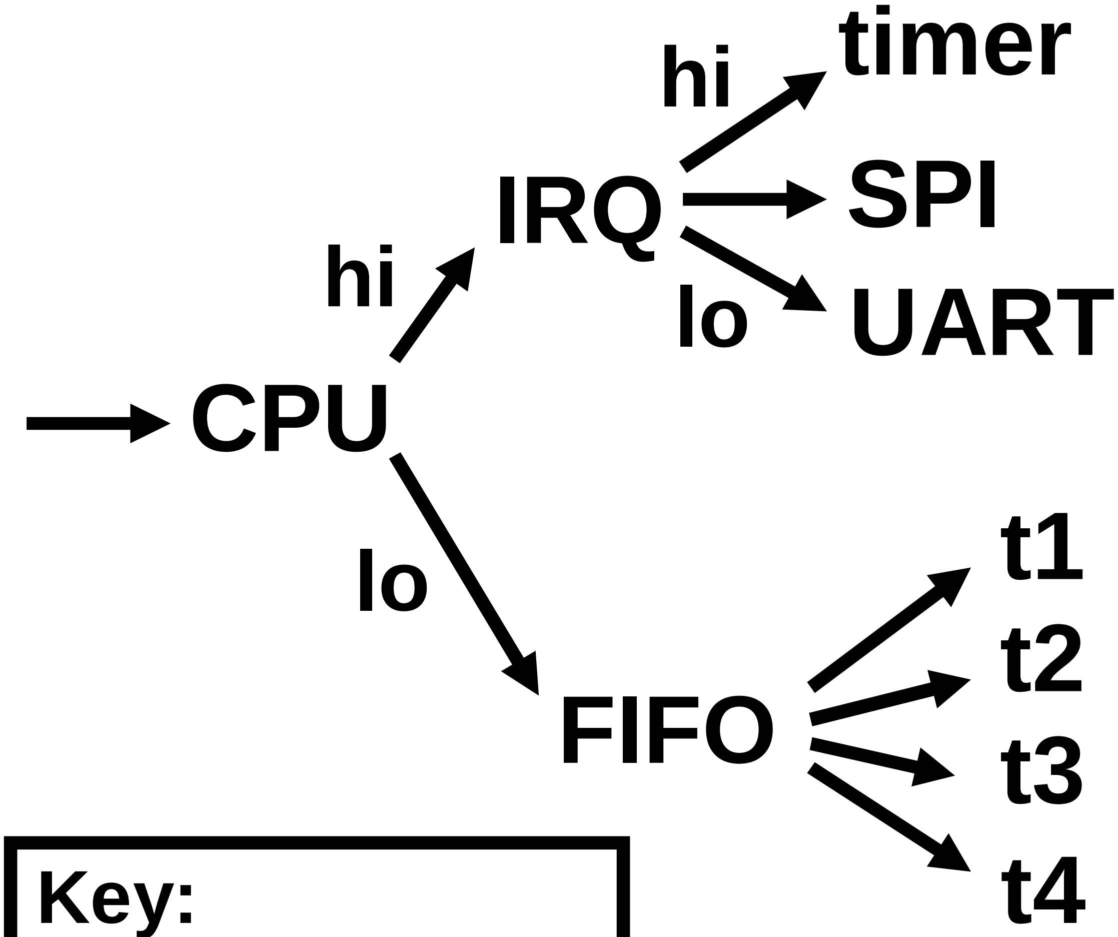
encrypt

decrypt

Key:
preemptive
non-preemptive

Results


Original TinyOS ✳
Modified TinyOS ■
Round Trips per Second
Task Execution Time (ms)
0
5
10
15
20
25
0
50
100
150
200



CPU

hi

IRQ

hi

timer

SPI

lo

UART

lo

FIFO

t1

t2

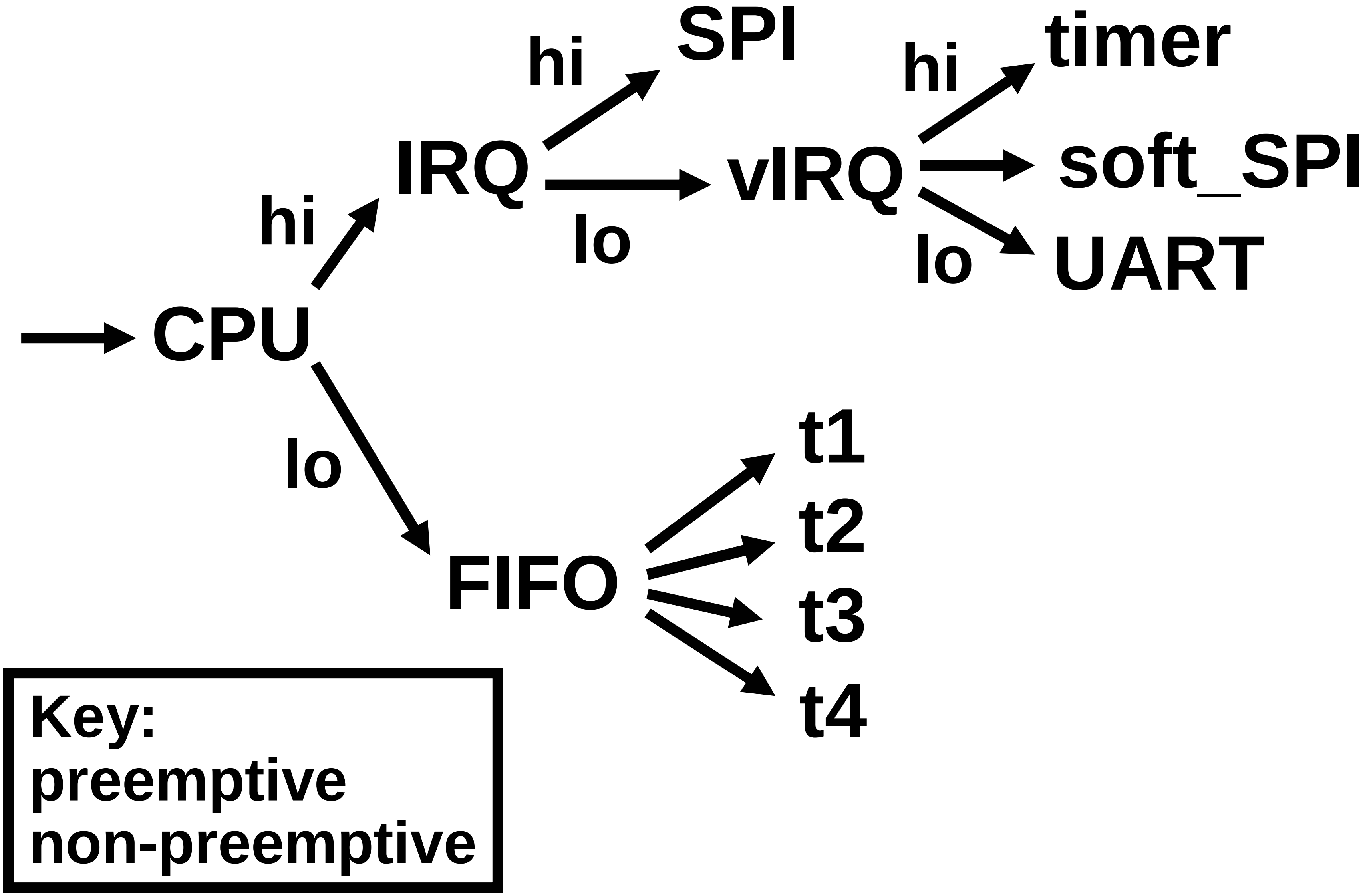
t3

t4

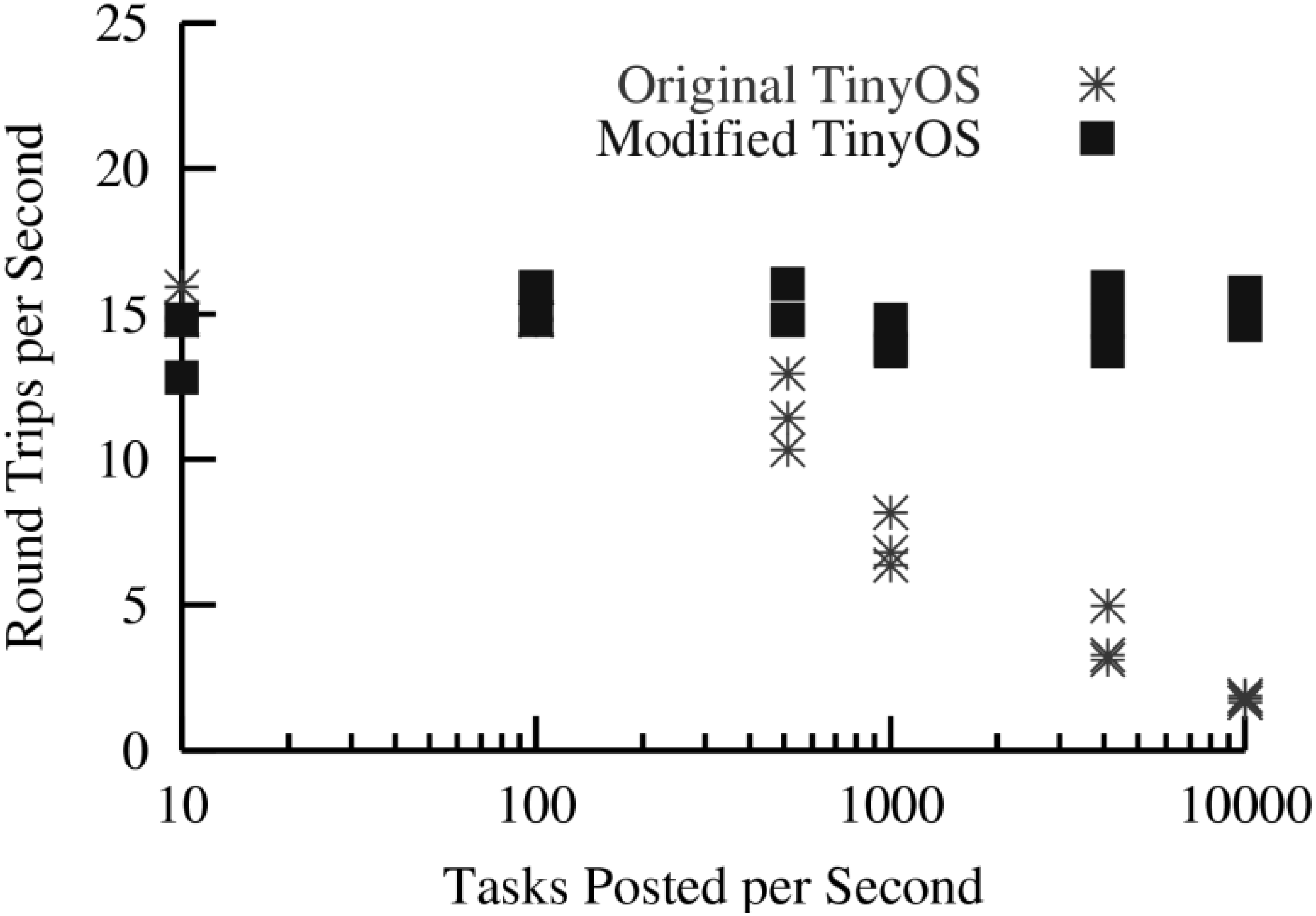
Problem 2: Missed SPI deadlines cause packet loss

Key:
preemptive
non-preemptive

Fixed TinyOS 2



Results


Original TinyOS
Modified TinyOS
Round Trips per Second
25
20
15
10
5
0
10
100
1000
10000
Tasks Posted per Second

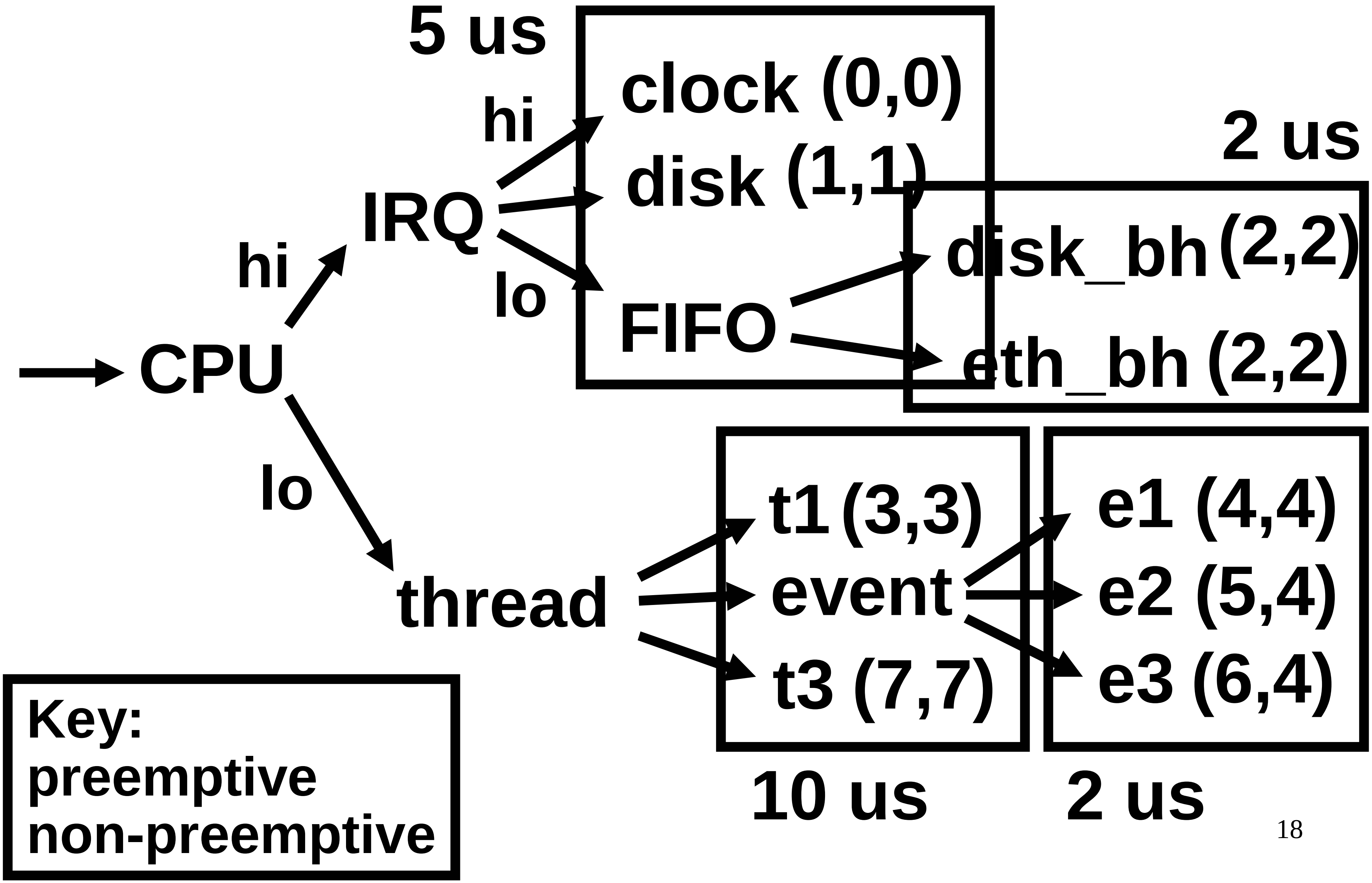
- Problem: How do we know when to promote / demote code?
- Solution: Response time analysis

- Problem: Very easy to introduce concurrency errors this way
- Solution: Concurrency analysis

Real-Time Analysis

- **Problem: How to analyze response times for hierarchies?**
- **Solution: Map to a problem that we know how to solve**
  - **Static priority scheduling**
  - **Preemption threshold scheduling**
- **Hierarchies restricted to:**
  - **Preemptive priority schedulers**
  - **Leaf schedulers can be non-preemptive FIFO or priority**

Real-Time Analysis



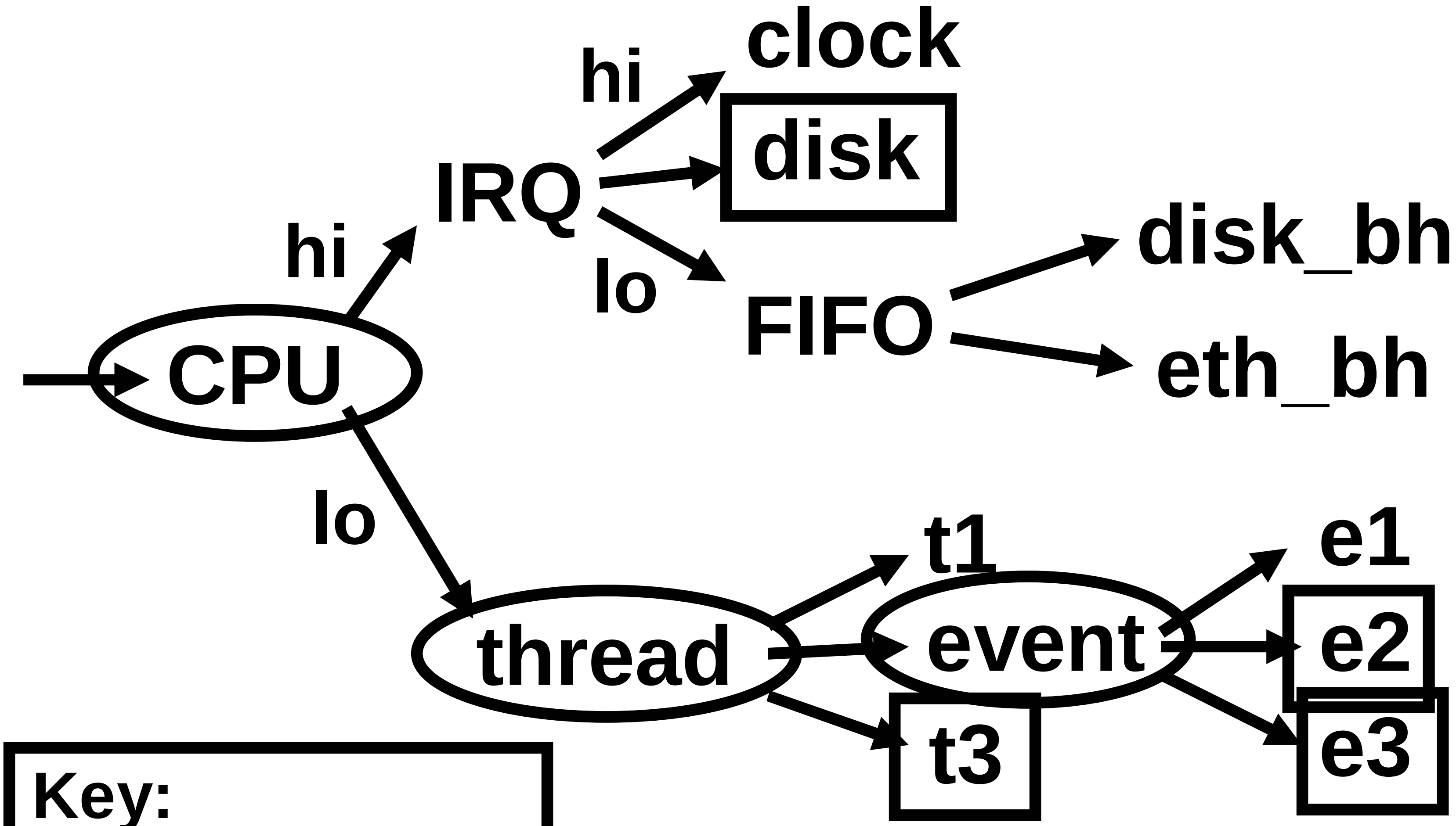
Concurrency Analysis

- **Problem: How to check for race conditions?**
- **Solution: Task scheduler logic**
  - **Static analysis of concurrency across a hierarchy of execution environments**

Task Scheduler Logic

- **Schedulers specify:**
  - **Preemption relations among things they schedule**
  - **Locks they provide**
- **Axioms propagate effects around the hierarchy**
- **TSL allows us to derive (potential) preemption relations for each pair of tasks in a system**
- **Details in paper…**

Concurrency Analysis



Key:
preemptive
non-preemptive

Contributions

- New notation for describing structure of systems software
- Heuristics for evolving systems
- Algorithms for hierarchical priority and FIFO schedulers:
  - Whole-program concurrency analysis
  - Response time analysis
- Experimental validation

Evolving Real-Time Systems using Hierarchical Scheduling and Concurrency Analysis

John Regehr   Alastair Reid

Kirk Webb   Michael Parker   Jay Lepreau

School of Computing, University of Utah