

CPU Inheritance Scheduling

Bryan Ford Sai Susarla

*Computer Systems Laboratory
Department of Computer Science
University of Utah*

`flux@cs.utah.edu`

`http://www.cs.utah.edu/projects/flux/`

October 30, 1996

Key Concepts

Threads schedule *each other* by donating the CPU using a *directed yield* primitive.

One *root scheduler thread* per processor sources all CPU time.

Kernel *dispatcher* manages threads, events, and CPU donation without making any scheduling policy decisions.

The Dispatcher

Implements thread sleep, wakeup, schedule, etc.

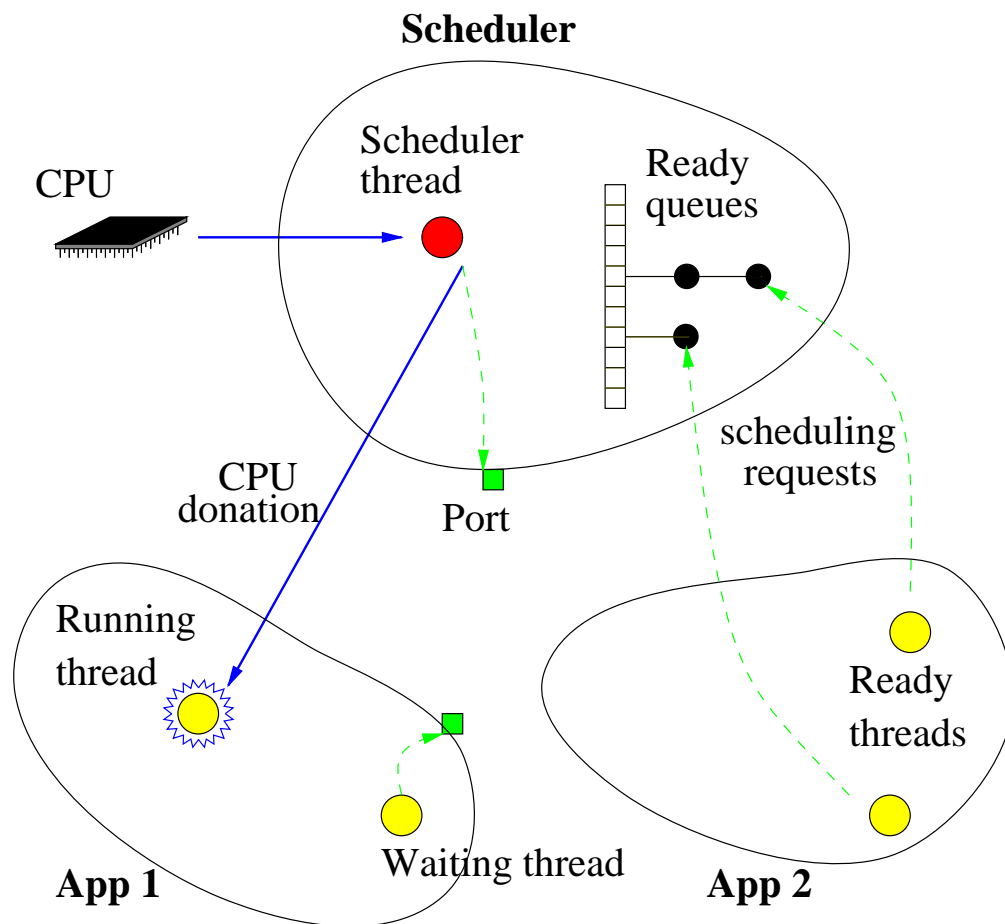
Runs in the context of currently running thread.

Has no notion of thread priority, CPU usage, clocks, or timers.

Dispatcher wakes a scheduler thread when:

- Scheduler's client blocks.
- Event of interest to the scheduler occurs.

Scheduling Example



The `schedule()` operation

`schedule(thread, port, sensitivity)`

Sensitivity levels:

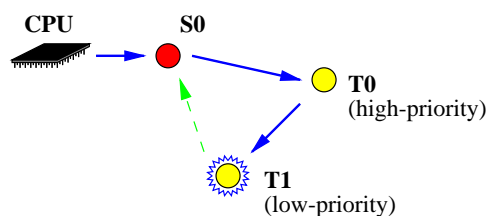
- `ON_BLOCK`: Wake the scheduler *any* time its client thread blocks.
- `ON_SWITCH`: Wake the scheduler only when a *different* client is requesting the CPU.
- `ON_CONFLICT`: Wake the scheduler only when *two or more* clients are runnable at the same time.

Implicit Donation

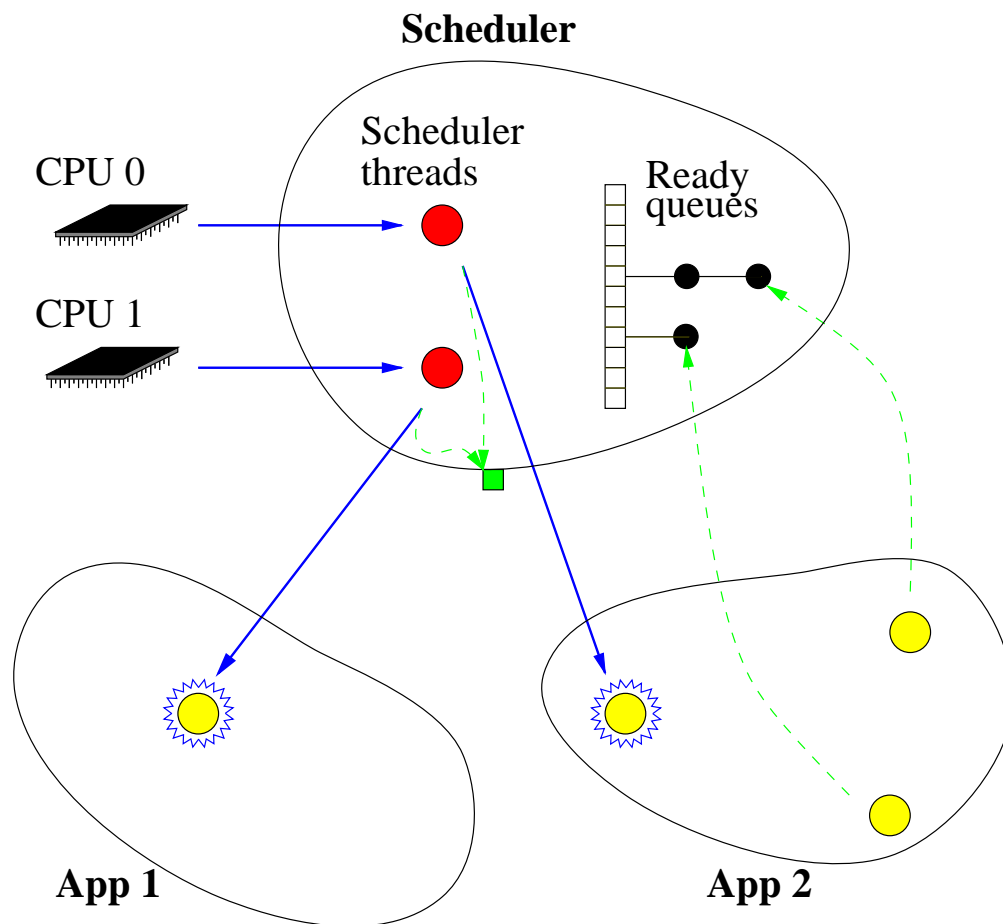
Works like `schedule()`, except done implicitly;
e.g.:

- Thread attempting to lock a held mutex donates to current owner
- Client thread donates to server thread for the duration of an RPC

Analogous to priority inheritance in traditional systems.



Multiprocessor Scheduling



Benefits

- Hierarchical, stackable scheduling policies
- Application-specific scheduling policies
- Modular CPU usage control
- Automatic priority inheritance
- Accurate CPU usage accounting
- Naturally extends to multiprocessors
- Supports processor affinity policies and scheduler activations

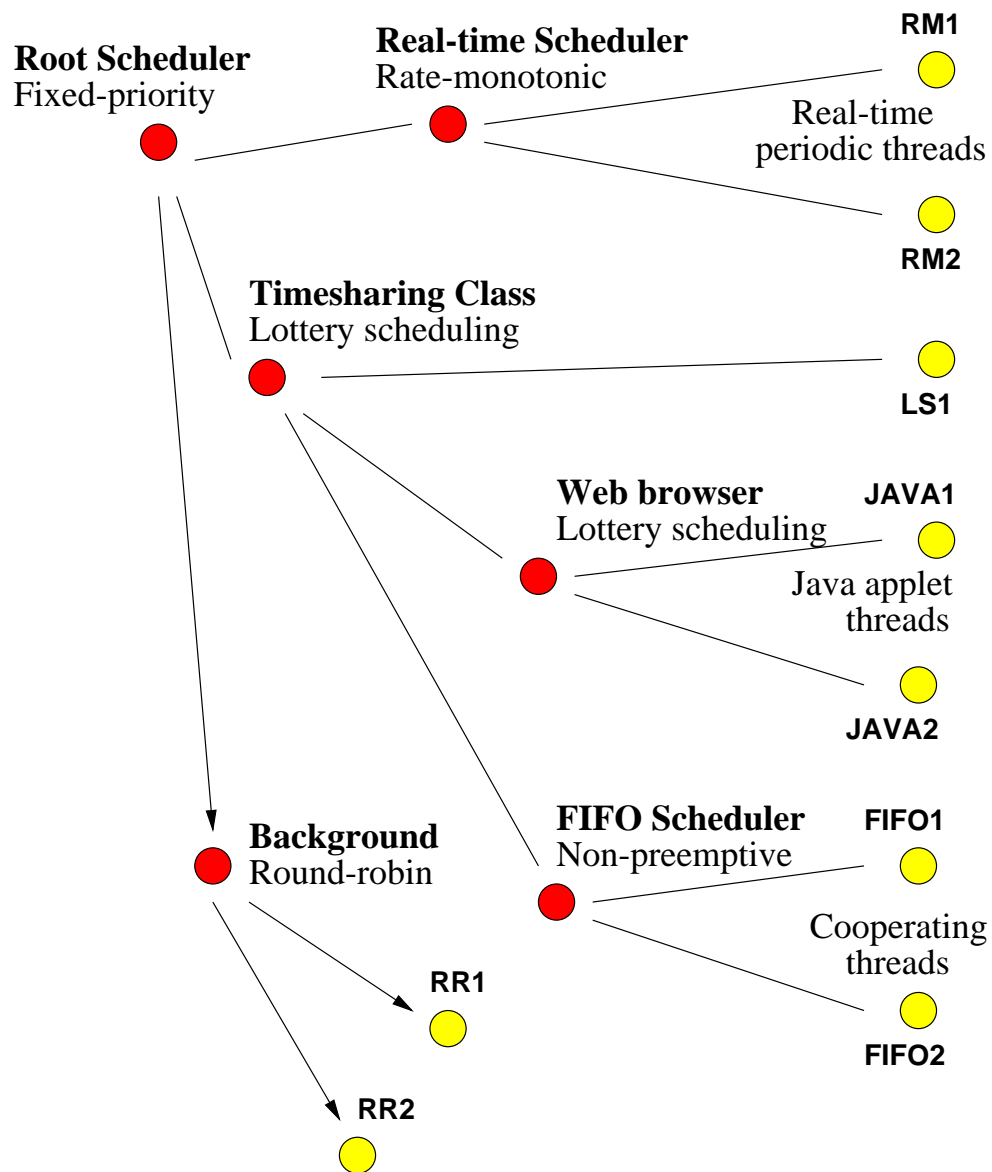
Prototype Implementation

Implemented as a fancy threads package in a BSD process.

Schedulers implemented:

- Fixed priority round-robin and FIFO
- Rate monotonic
- Lottery

Scheduling Hierarchy

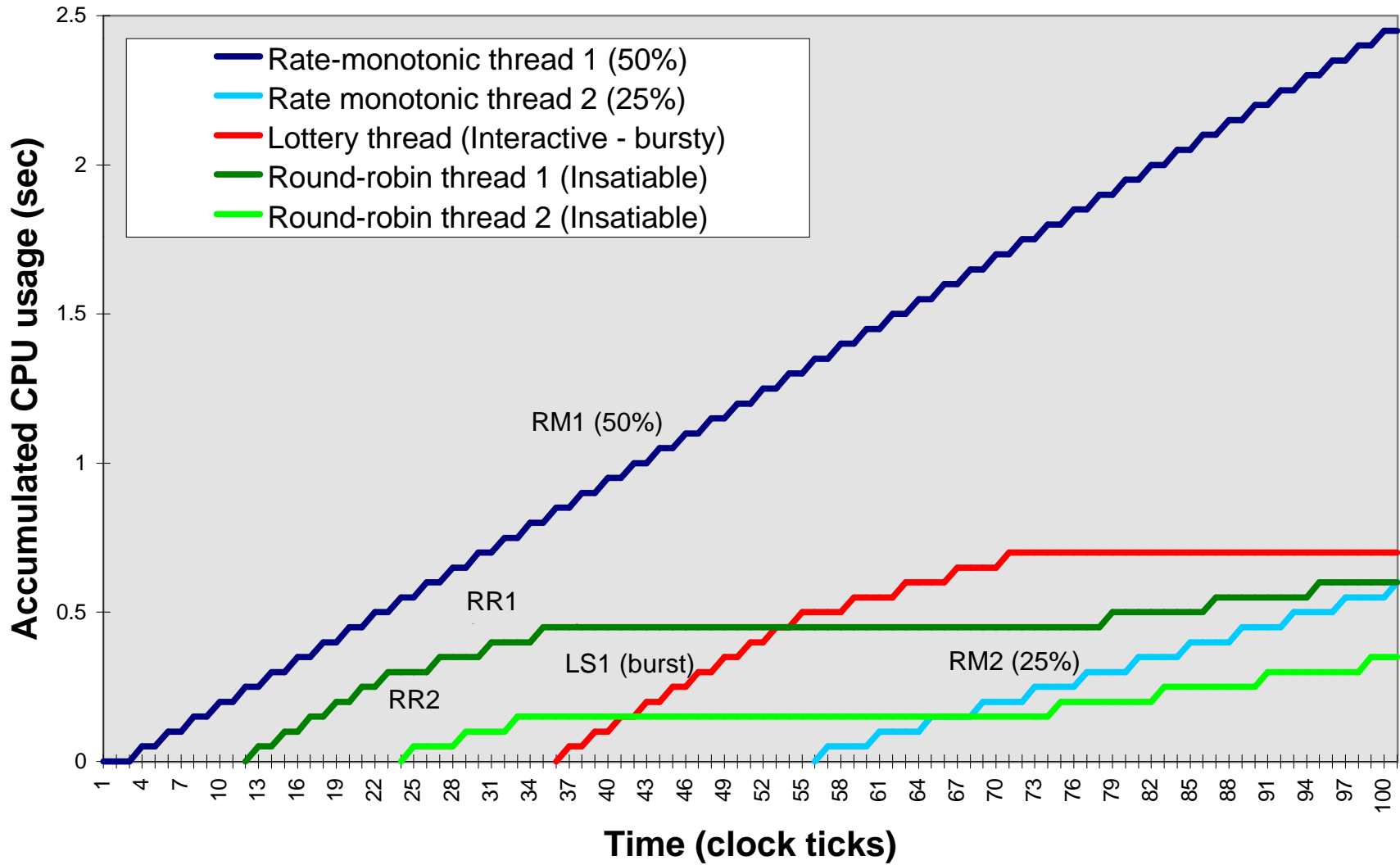


Results

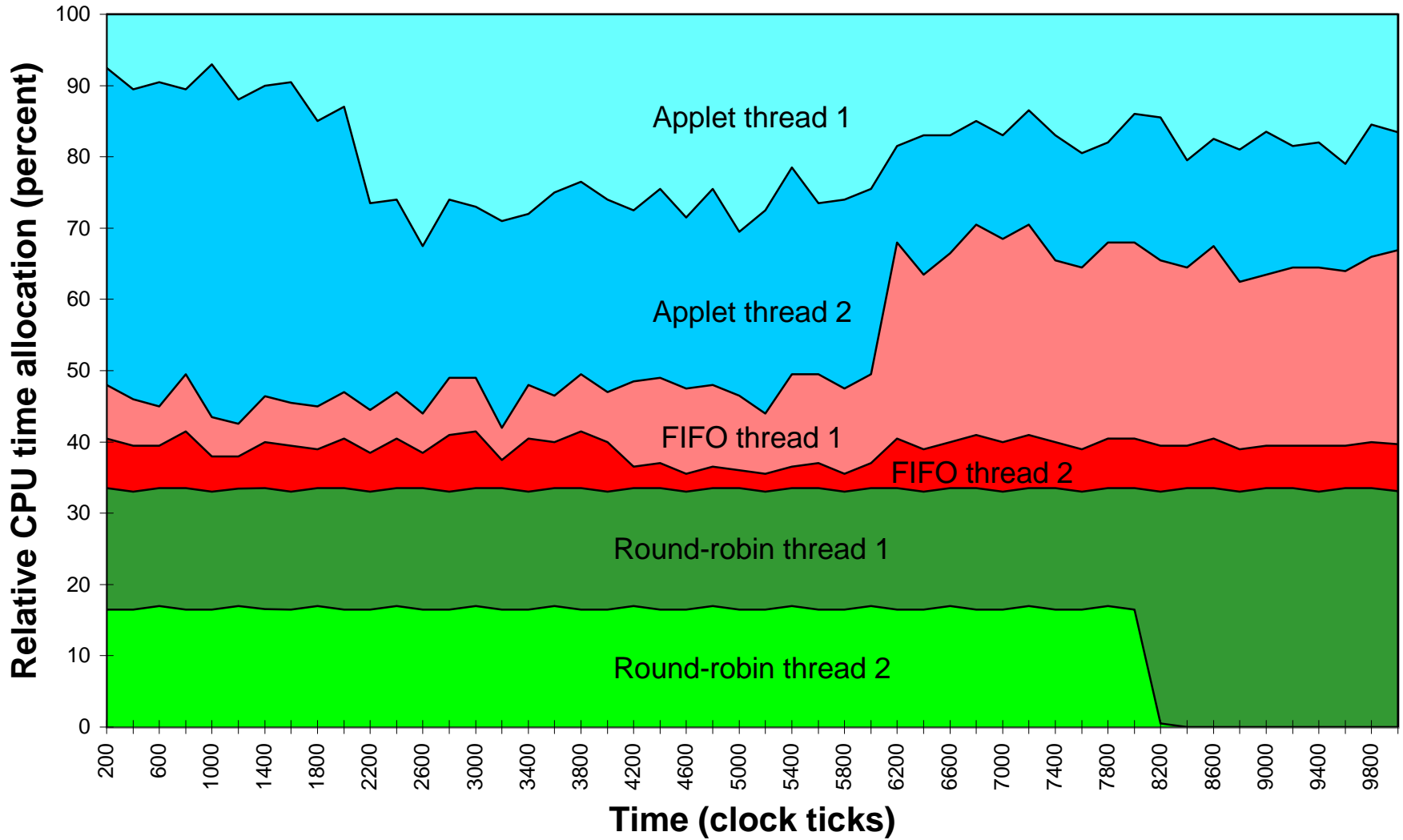
Three measures:

- Scheduling behavior (correctness)
- Overhead
- Implementation complexity

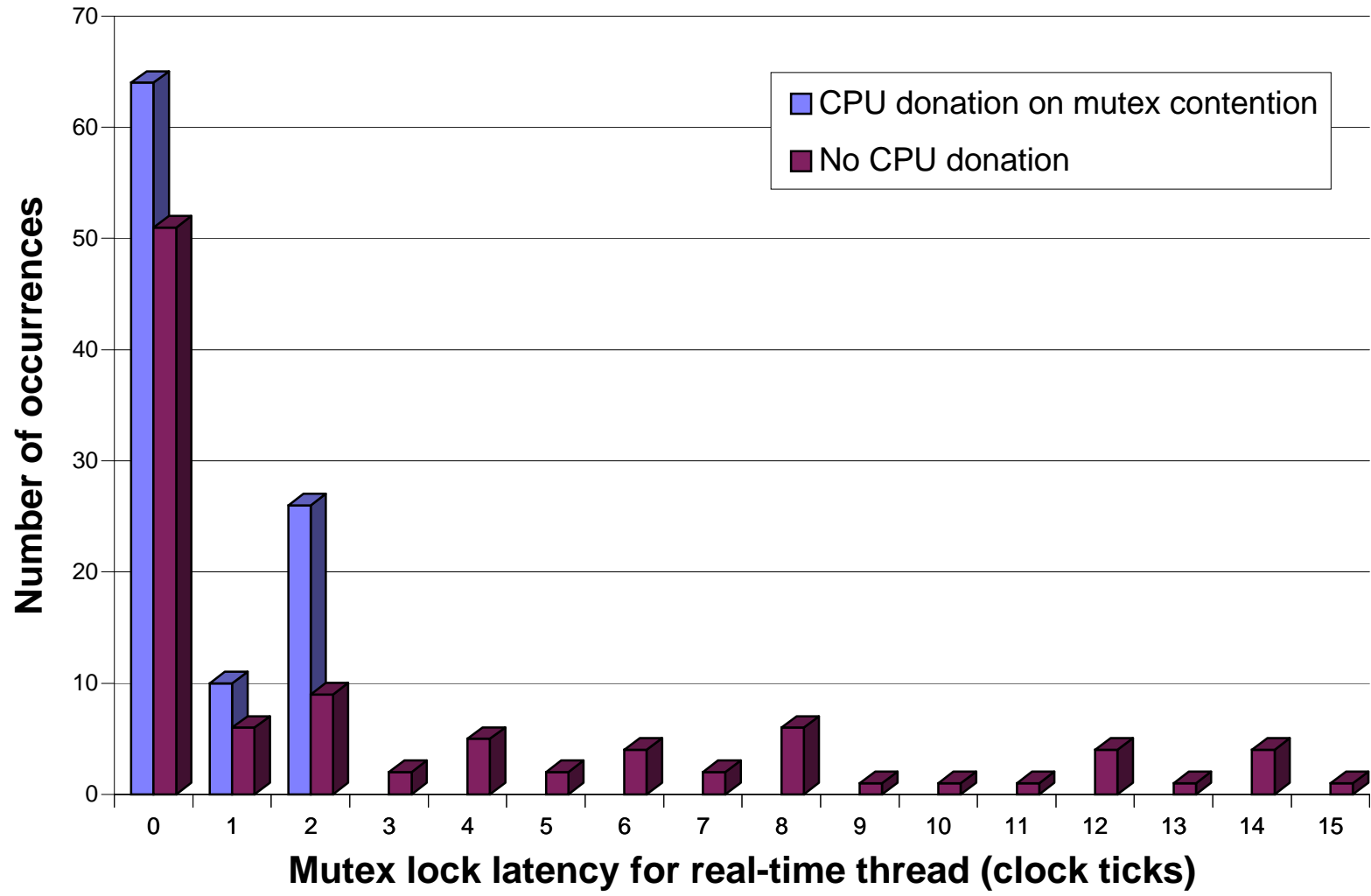
Multi-policy Scheduling Behavior



Modular Control of CPU Usage



Real-time Scheduling Behavior



Performance

- Dispatcher overhead
 - Base cost
 - Sensitivity to hierarchy depth
- Context switching overhead
 - Number of additional context switches
 - Cost of context switches

Dispatcher Micro-benchmarks

Scheduling Hierarchy Depth	Dispatch Time (μs)
Root scheduler only	8.0
2-level scheduling	11.2
3-level scheduling	14.0
4-level scheduling	16.2
8-level scheduling	24.4

Context switch overhead

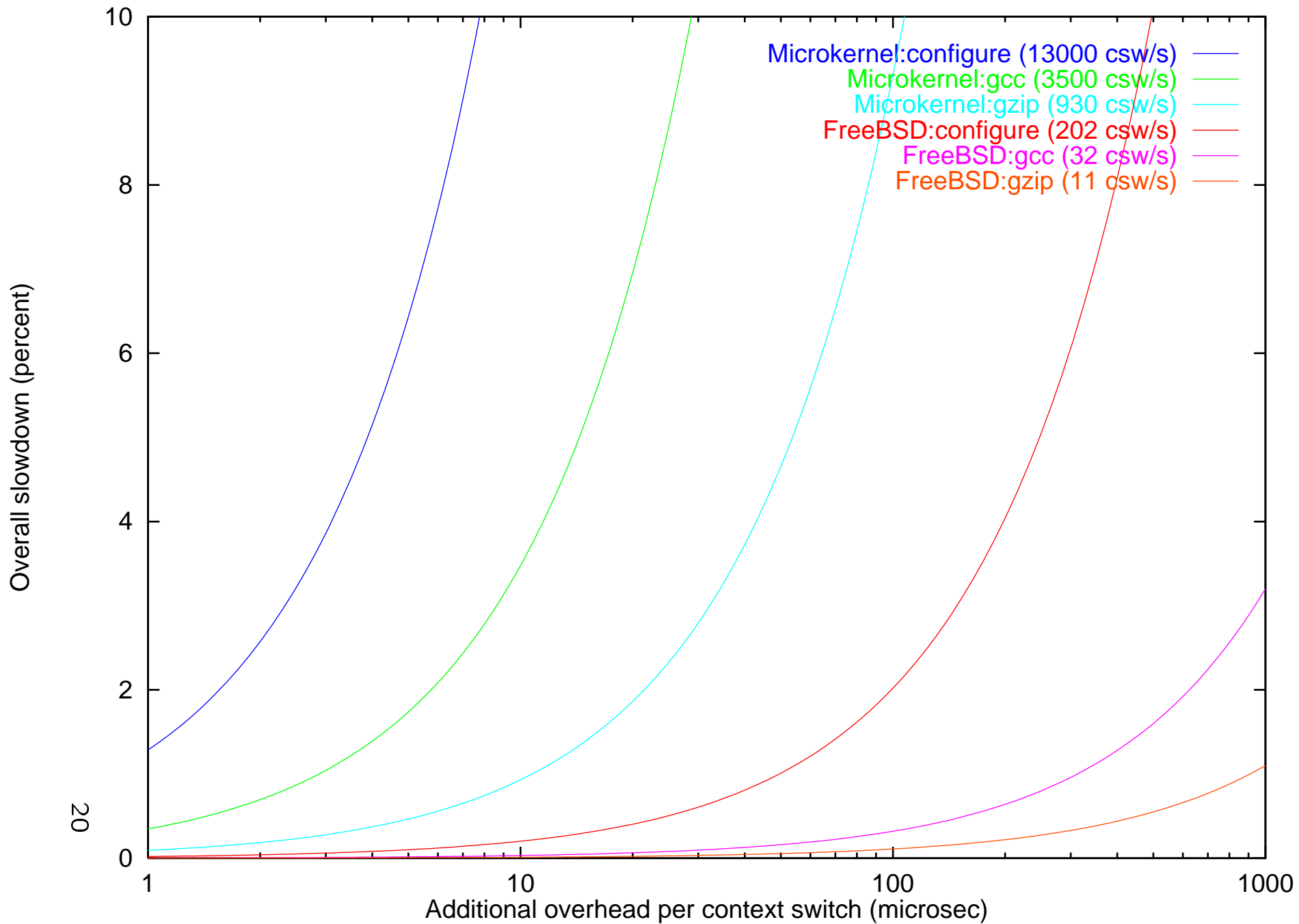
- In prototype, measure what proportion of context switches are to scheduler threads (i.e., extra)
- On a real OS, measure rate of context switches in various work loads
- Project slowdown in two OSs, based on expected rate and speed of context switches

Context Switches for Simple Tests

	Client/ Server	Parallel Database	Real- time	General
RM1	57		322	101
RM2	19			26
RM3	19			
LS1	25		622	17
JAVA1	46			
FIFO1	9			
RR1	114	238	249	7
RR2	3	242		14
RR3		234		
RR4		243		
User invocations	492	957	1193	165
Root scheduler	262	956	1237	142
Rate monotonic	43		1	65
Lottery scheduler	30		57	3
Applet scheduler	2			
FIFO scheduler	1			
Round-robin sched	8		8	8
Scheduler invoc.	346	956	1303	218
Total csw	838	1913	2496	383
Scheduler %	41%	50%	52%	56%

Statistics for Common Applications

	gzip	gcc	tar	configure
Run time (sec)	26.4	35.3	9.6	26.0
Context switches/sec	11	32	81	202
Traps/sec	10	562	22	3470
System calls/sec	23	651	517	1807
Device interrupts/sec	427	509	3337	1055



Code Complexity

- **Dispatcher:**
550 raw, 160 lines of semicolons
- **Example schedulers:**
each is 100–200 semicolons

Related Work

Existing multi-policy systems:

- Multi-class systems: Mach, NT
- Aegis Exokernel

Related Work

Existing hierarchical scheduling policies:

- KeyKOS meters
- Lottery/stride scheduling
- Start-time Fair Queuing (SFQ)

CPU inheritance scheduling is not a policy.

Status

Works, but needs to be tried in a real OS

Fluke kernel implementation in progress

Source for prototype will be available from the OSDI and Flux project web pages:

<http://www.cs.utah.edu/projects/flux/>

Conclusion

CPU inheritance scheduling:

- Provides flexible CPU scheduling, and supports many existing policies and mechanisms
- Is efficient enough for common uses
- Is straightforward to implement (in user mode)
- Supports the Fluke nested process model