Linux Realtime Response

Challenges in Making Linux Ready for Real Time Computing

Paul E. McKenney
Distinguished Engineer
IBM Linux Technology Center

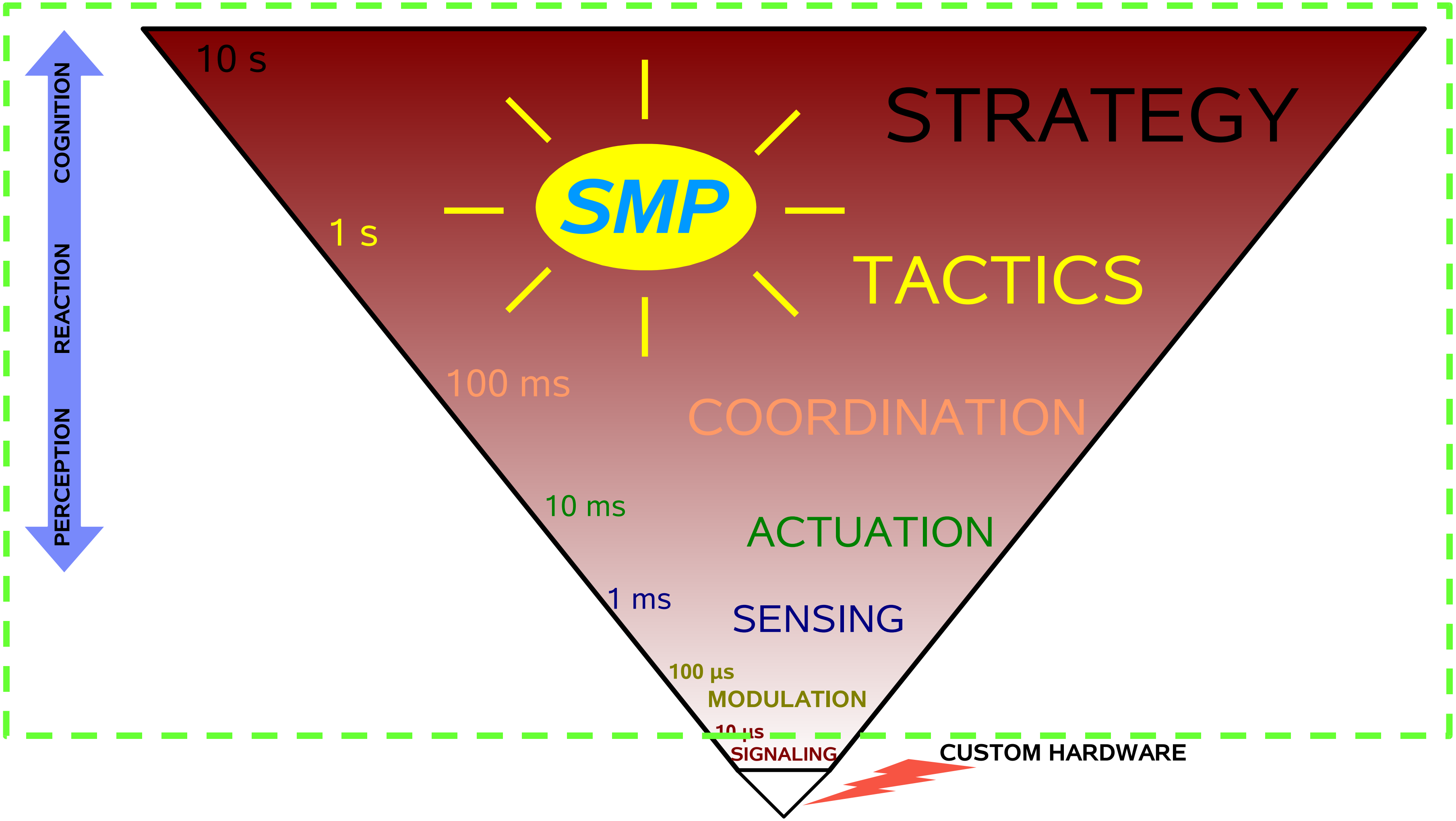
Overview

- Goals, Non-Goals, and Corollaries
- Overview of Linux Realtime Approaches
- Priority Inversion and Reader-Writer Lock
- Case Study: Signal-Delivery Latency
- Summary and Conclusions

Goals, Non-Goals, And Corollaries

- Goals
  - Realtime response on commodity mid-range multiprocessors
  - Common Linux-kernel code base
  - Merciless application of the 80-20 rule: do the 20% of the work required by 80% of the realtime applications now, more later
- Non-Goals
  - Provable “diamond-hard” realtime response (not yet, anyway)
  - Realtime response from *all* services: incrementalism instead
- Corollaries
  - Normal locking (priority inheritance)
  - Full POSIX semantics
  - Scalability and performance *in addition to* realtime response

Linux Realtime Goals



Linux Realtime Approaches (Violently Abbreviated)

Project	Quality of Service	Inspection	API	Complexity	Fault Isolation	HW/SW Configs
Vanilla Linux Kernel	10s of ms all services	All	POSIX + RT extensions	N/A	None	All
PREEMPT	100s of us Schd, Int	All spinlock critsect, preempt- & int-disable	POSIX + RT extensions	N/A	None	All
Nested OS	~10 us RTOS svs	RTOS + int-disable	RTOS	Dual environment	Good	All
Dual-OS / Dual-Core	<1 us RTOS svcs	All RTOS	RTOS	Dual environment	Excellent	Specialized
PREEMPT_RT	10s of us Schd, Int	All preempt- & int-disable (most ints in process ctxt)	POSIX + RT extensions	"Modest" patch	None	All (except some drivers)
Migration Between OSes	? us RTOS svcs	All RTOS + int-disable	RTOS (can be POSIX)	Dual env. (Fusion)	OK	All?
Migration Within OS	? us RTOS svcs	Scheduler + RT syscalls	POSIX + RT extensions	Small patch	None	All?

http://lwn.net/Articles/143323/ for additional detail.

Other Features That Might Appear. Someday.

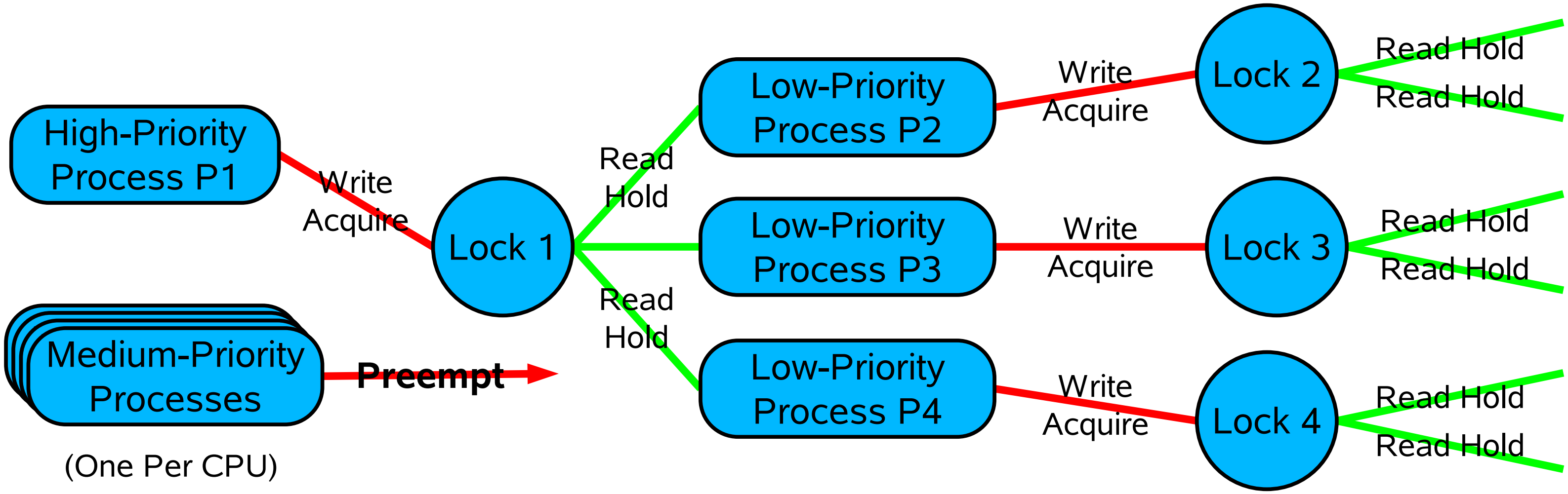
- Deterministic I/O
  - Disk I/O – or, more likely, Flash memory
  - Network protocols
    - Datagram protocols (UDP) relatively straightforward
    - “Reliable” protocols (TCP, SCTP) more difficult
    - Maintaining low network utilization is key workaround
    - Possible contender: Van Jacobson's lock-free Linux TCP/IP work
- Priority Inheritance Beyond Locking
  - Reader-writer locks with concurrent readers
    - Writer-to-reader boosting problematic
  - Across RCU, especially when low on memory
  - Across memory allocation
    - Boost priority of someone who is about to free?
  - Across networks

In Some Cases, Priority Boosting is Undesirable...

...Or At Least Uncomfortable!!!

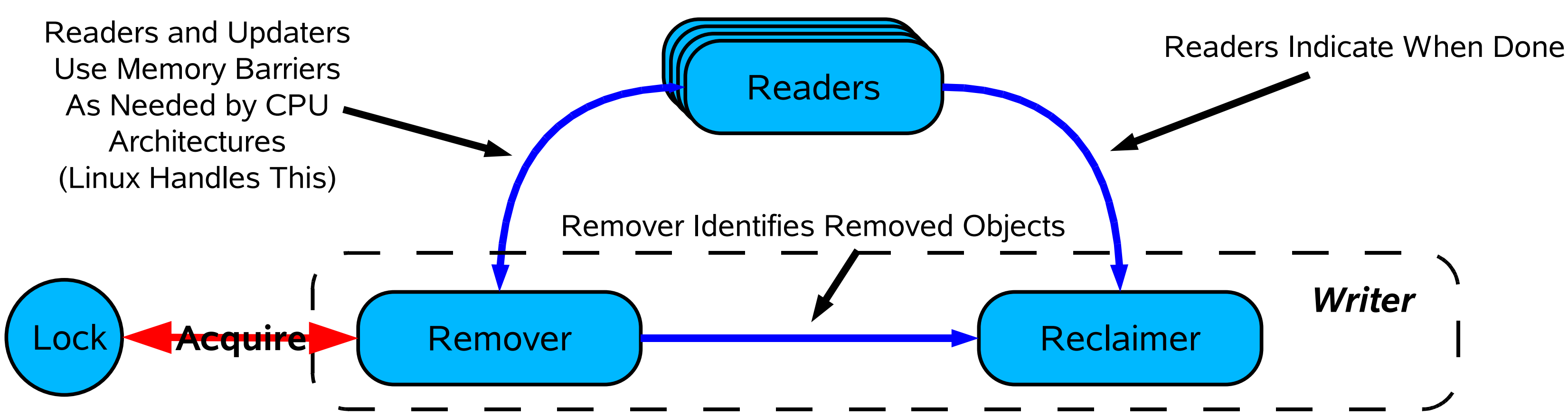
Priority Boosting and Reader-Writer Locking

- Process P1 needs Lock L1, held by P2, P3, and P4
  - Each of which is waiting on yet another lock
    - read-held by yet more low-priority processes
    - preempted by medium-priority processes
- Process P1 will have a long wait, despite its high priority
  - ***Even given priority inheritance: many processes to boost!***
  - Further degrading P1's realtime response latency
- Linux -rt approach: only one reading task...

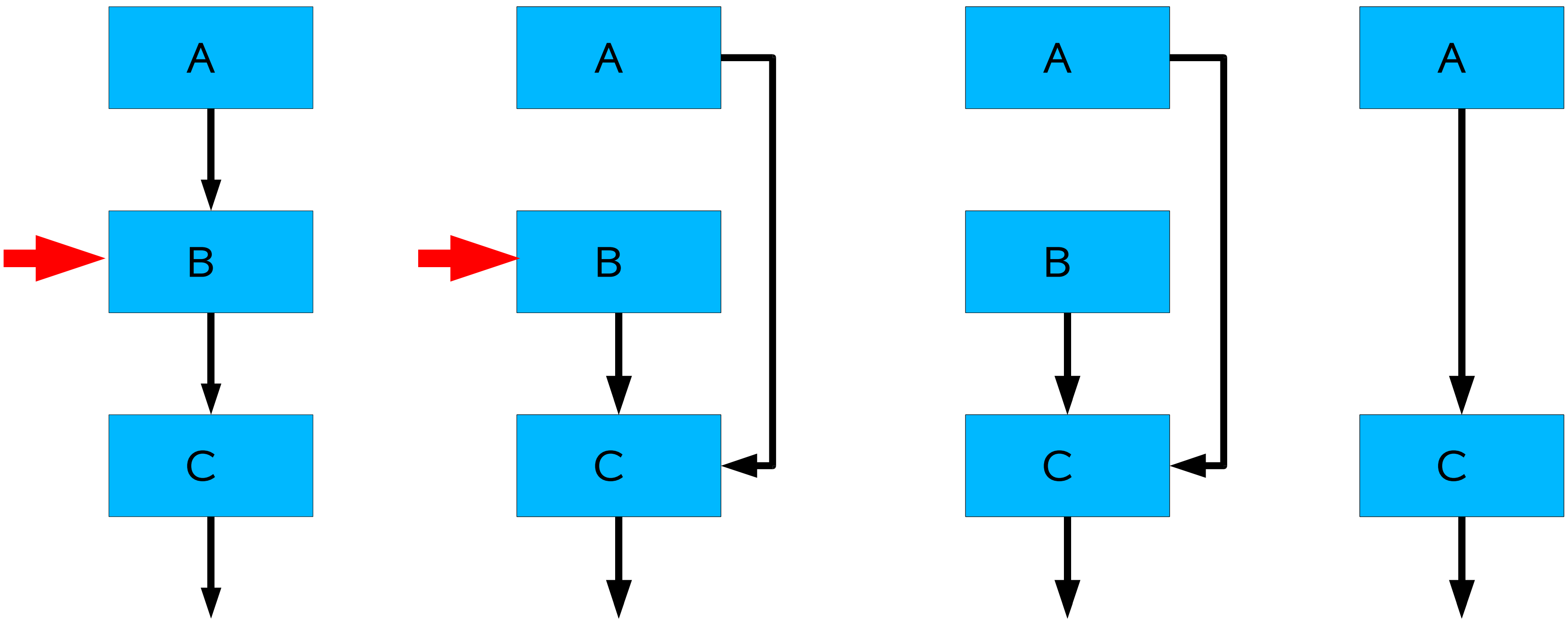


Priority Inversion and RCU: What is RCU?

- Analogous to reader-writer lock, but readers acquire no locks
  - Readers therefore cannot block writers
  - Reader-to-writer priority inversion is therefore impossible
- Writers break updates into “removal” and “reclamation” phases
  - Removals do not interfere with readers
  - Reclamations deferred until all readers drop references
    - Readers cannot obtain references to removed items
- RCU used in production for over a decade by IBM (and Sequent)
- IBM recently adapted RCU for realtime use in Linux



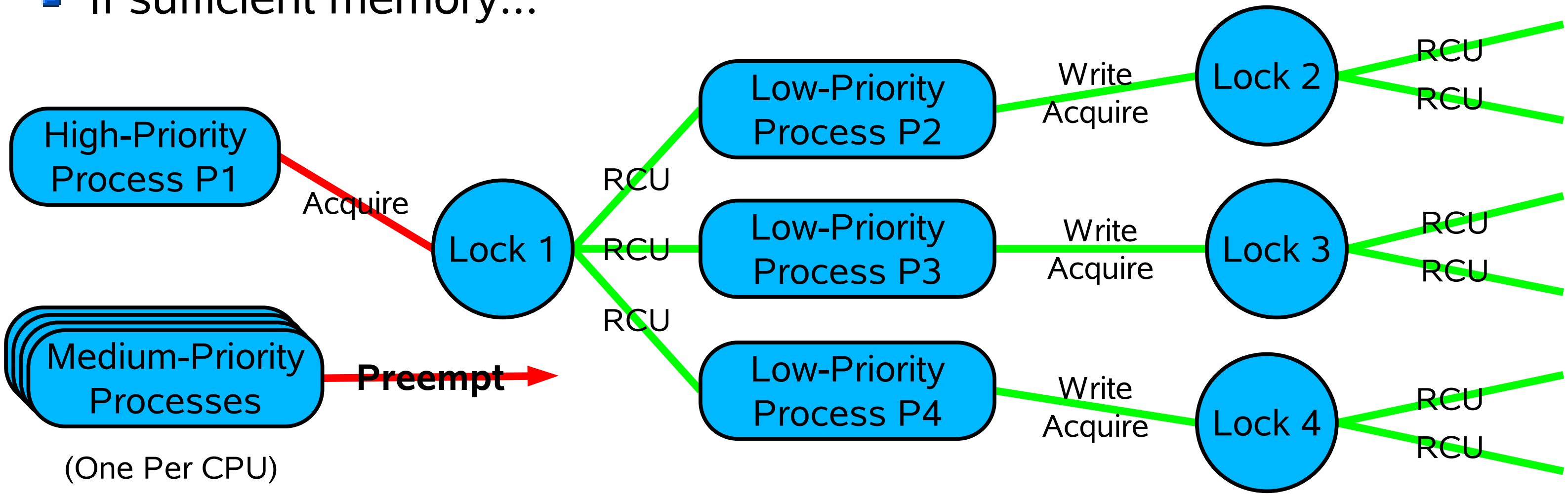
RCU Example: Removal From Linked List



Determine when RCU readers are done by observing states forbidden to RCU readers

Priority Inversion and RCU

- Process P1 needs Lock L1, but P2, P3, and P4 now use RCU
  - P2, P3, and P4 therefore need not hold L1
  - Process P1 thus immediately acquires this lock
  - Even though P2, P3, and P4 are preempted by the per-CPU medium-priority processes
- No priority inheritance required
  - Except if low on memory: permit reclaimer to free up memory
- Excellent realtime latencies: medium-priority processes can run
  - High-priority process proceeds despite low-priority process preemption
  - If sufficient memory...



RCU Realtime Scorecard

	Reliable	Callable From IRQ	Preemptible Read Side	Small Memory Footprint	Sync-Free Read Side	Indpt of Memory Blocks	Nestable Read Side	Uncond R-W Upgrade	Compatible API
Classic RCU			N	N					
rcu-preempt				X	N				
Jim Houston Patch			N		N				
Reader-Writer Locking					N		N	N	n
Unconditional Hazard Pointers				X	n	N			
Hazard Pointers: Failure				n	n	N			N
Hazard Pointers: Panic	N			n	n	N			
Hazard Pointers: Blocking		N		n	n	N			
Reference Counters				N	n	N			
rcu_donereference()					n	N			N
Lock-Based Deferred Free	N				N				
Read-Side Counter GP Suppression				N	n				
Read-Side Counters w/ “Flipping”					(n)				

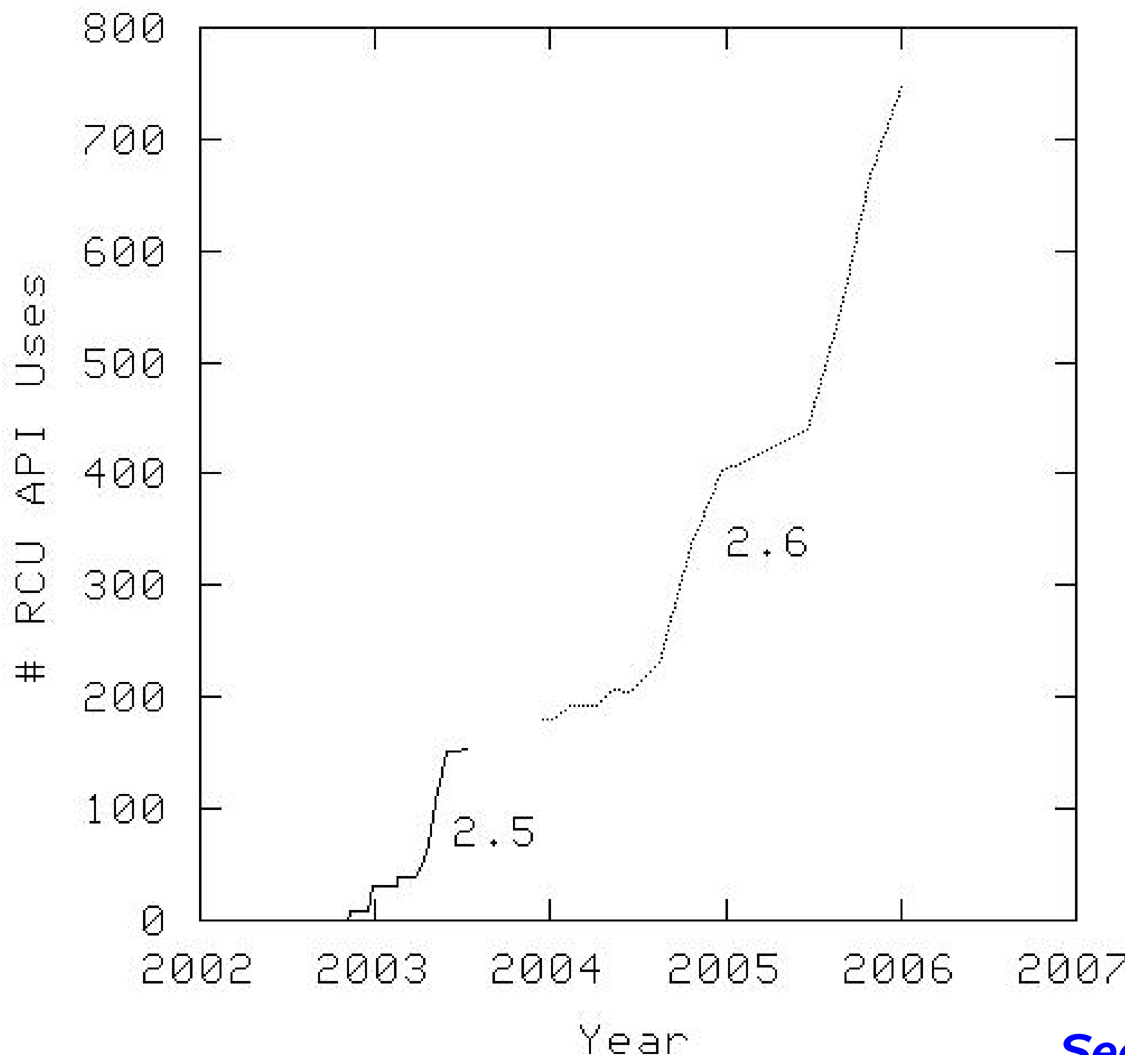
Case Study: kill() System-Call Latency

- Current concern: Latency of signal transmission
  - Reduce latency effect on sending process
  - Transmission-to-reception latency not yet a problem
- kill() read-holds on tasklist_lock for mutual exclusion
  - Prevent processes and threads from changing state
- Updates to process/thread state write-hold tasklist_lock
  - fork(), exec(), exit(), change process group, setuid, ...
- But most state-changes do not affect signal delivery
  - Traditional approach: fine-grained locking or non-blocking synchronization
  - But these approaches introduce high complexity
- Alternative: use RCU instead of read-acquiring tasklist_lock
  - 2x-3x reduction in latency, small code change
  - Now in Linus's mainline kernel source tree

Summary

- Linux is making great progress in realtime latency
- Modest technical goals, striving for widespread usefulness
  - Tens of microseconds scheduling/interrupt latency
  - Similar latencies for selected operations and system calls
  - Single source base (this may take awhile)
  - Simplicity, scalability, and performance minimally degraded
  - No provable latencies – perhaps SW tools will help?
- Using old (preemption) and new (RCU) techniques
  - Preemption of RCU read-side critical sections requires innovation in RCU implementation (ongoing work)
  - Replacement of reader-writer locks with RCU requires care due to RCU readers not blocking updates (ongoing work)
- No obvious *technological* barrier to scalable realtime Linux...
- But can the Linux community handle RCU?

Can the Linux Community Handle RCU?



Seems to be doing so!!!

Legal Statement

- The views expressed in this paper are the author's only, and should not be attributed to IBM.
- Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
- Other company, product, and service names may be trademarks or servicemarks of others.

Resources

- Discussion of realtime measures and goals
- Different approaches to Linux realtime
  - http://lwn.net/Articles/143323/
- Description of PREEMPT_RT patchset
  - http://lwn.net/Articles/146861/
- PREEMPT_RT patchset
  - http://www.redhat.com/~mingo/realtime-preempt/
- Victor Yodaiken dislikes priority inheritance; Doug Locke disagrees
  - http://www.linuxdevices.com/articles/AT7168794919.html
  - http://www.linuxdevices.com/articles/AT5698775833.html

BACKUP

Why Realtime Response???

- Moore's Law Now Generating Multithread/Multicore CPUs
- Consolidate Realtime Market: Improve software portability
- Customer Demand: DoD, Digital Media/Gaming, Financial
- “Nintendo Generation”
  - Grew up with sub-reflex response time from computers
  - Now are entering jobs controlling computer purchases
- Human-computer interaction changes when response time drops below about 100 milliseconds
  - Much more natural and fluid, much more productive
  - And can developed countries afford to continue to pay their people to stare at hourglasses???
    - But this problem extends far above the operating system...
- Delays accumulate across networks of machines

Isn't Realtime a Single-CPU Thing?

Today's Systems

Historical Realtime:
- ***Few CPUs***
- Latency Guarantees
- ***Non-Standard***

OR

Historical SMP:
- Many CPUs
- ***No Guarantees***
- Standard (and OSS)

But Not Both!!!

Convergence

Emerging Systems

SMP Realtime:
- Many CPUs
- Latency Guarantees
- Standard (and OSS)

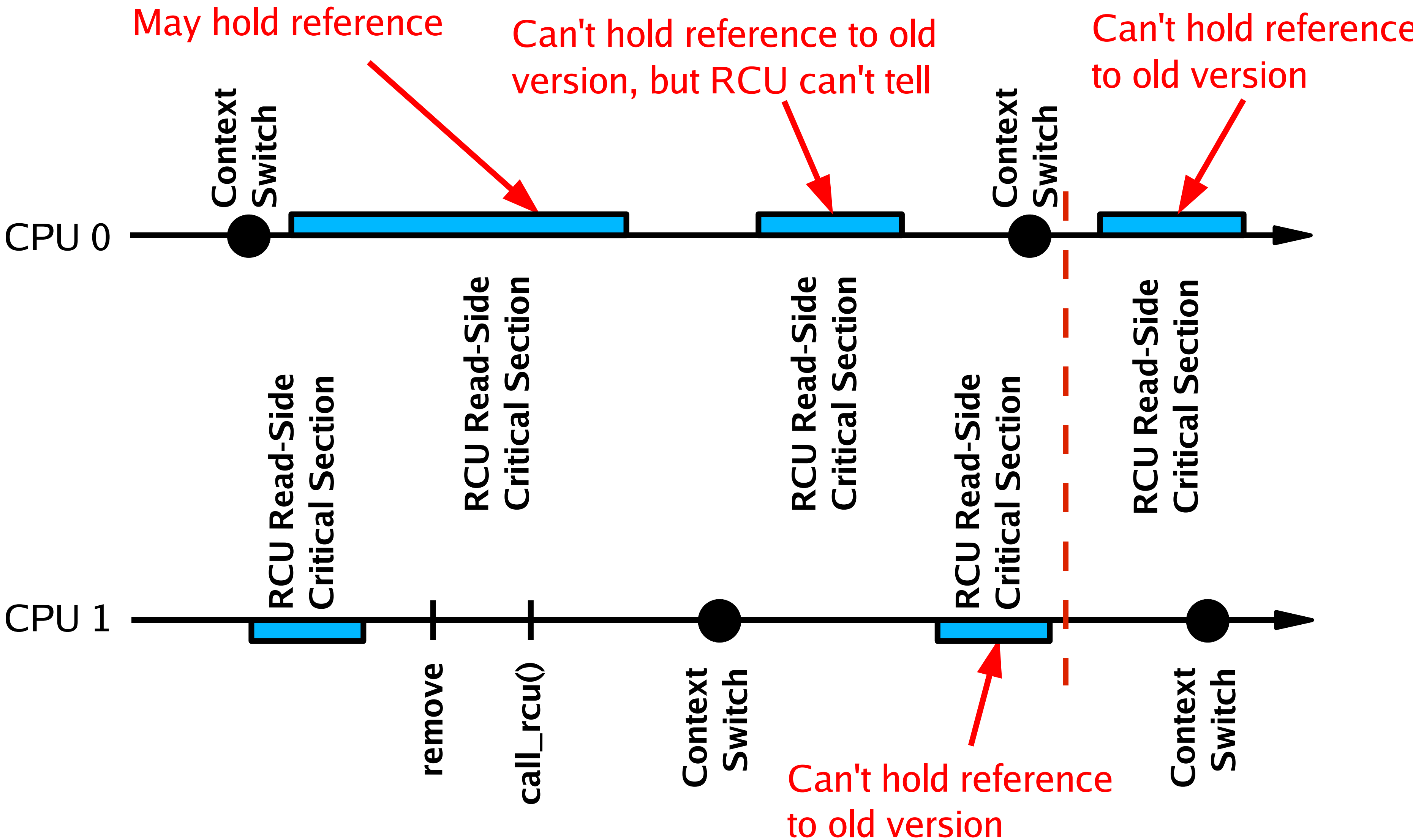
- User Demand (DoD, Financial, Gaming, ...)
- Techological Changes Leading to Commodity SMP
  - Hardware Multithreading
  - Multi-Core Dies
  - Tens to Hundreds of CPUs per Die – Or More

What Does Realtime Entail?

- Quality of Service (Beyond "Hard"/"Soft")
  - Services Supported
    - Probability of meeting deadline absent HW failure
    - Deadlines supported
  - Performance/Scalability for RT & non-RT Code
- Amount of Global Knowledge Required
- Fault Isolation
- HW/SW Configurations Supported

- "But Will People Use It?"

Classic RCU


May hold reference
Can't hold reference to old version, but RCU can't tell
Can't hold reference to old version
Context Switch
CPU 0
RCU Read-Side Critical Section
RCU Read-Side Critical Section
RCU Read-Side Critical Section
Context Switch
RCU Read-Side Critical Section
RCU Read-Side Critical Section
CPU 1
remove
call_rcu()
Context Switch
Can't hold reference to old version
Context Switch

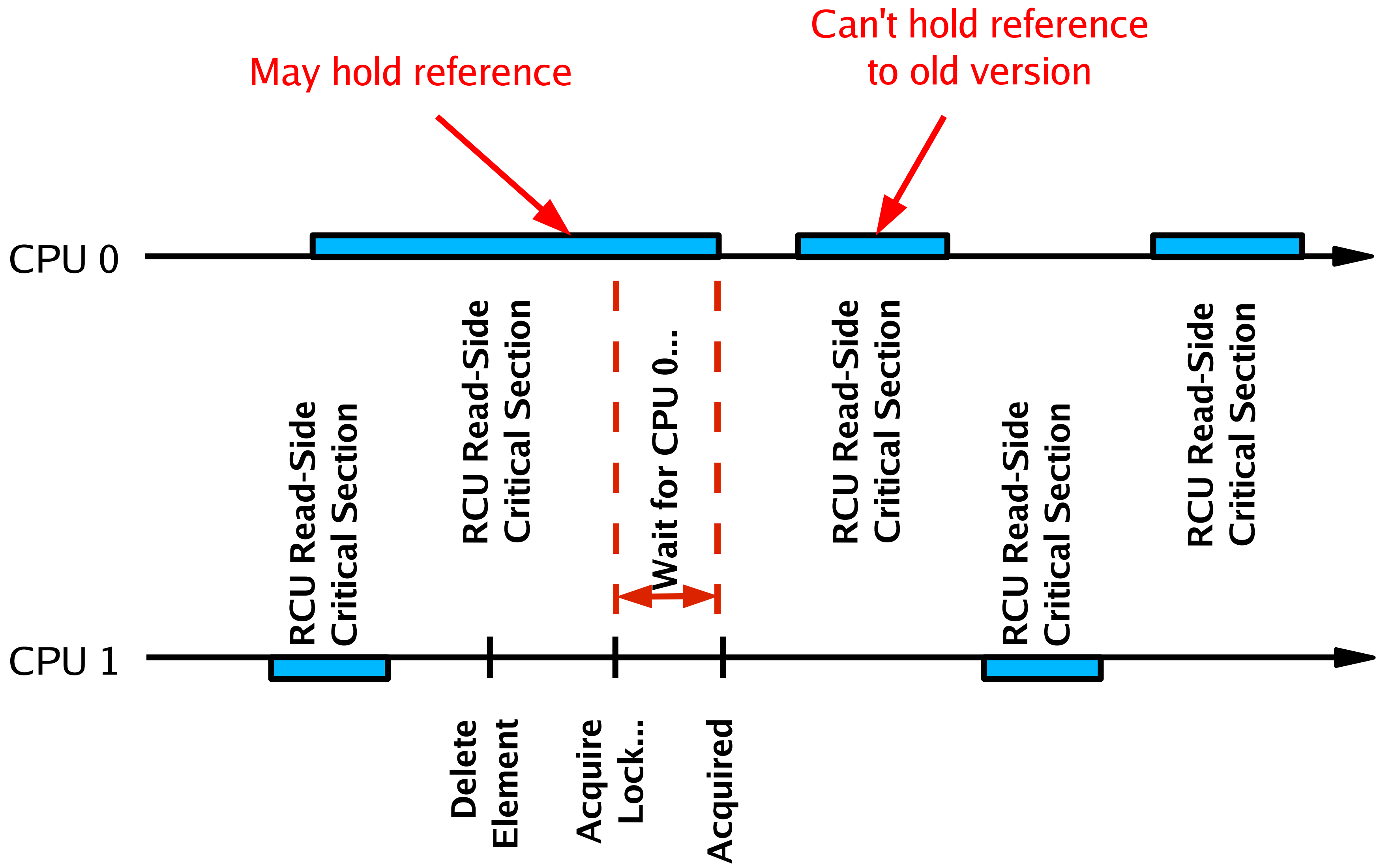
Simple Solution: Lock-Based Defer

```
void rcu_read_lock(void)
{
   read_lock(&rcu_ctrlblk.lock);
}

void rcu_read_unlock(void)
{
   read_unlock(&rcu_ctrlblk.lock);
}

void synchronize_kernel(void)
{
   write_lock_bh(&rcu_ctrlblk.lock);
   write_unlock_bh(&rcu_ctrlblk.lock);
}
```

Lock-Based Defer: Grace Periods



Problems With Lock-Based Deferral

- Latency can “bleed” from one reader to another via updater
  - Reader 1 read-holds lock
  - Updater blocked attempting to write-acquire lock
  - Reader 2 blocked attempting to read-acquire lock
    - Allowing Reader 2 to precede Updater results in starvation
- Use of RCU in interrupt handlers can result in self-deadlock
  - These deadlocks could be avoided by masking interrupts
  - But that would defeat the whole purpose: preemptible RCU read-side critical sections

- Solution: Counter-based scheme

Counter-Based Realtime RCU

	Current Count	Previous Count
CPU 0	0	1
CPU 1	2	0
CPU 2	1	0
CPU 3	1	0
CPU 4	0	0
CPU 5	3	1
CPU 6	0	1
CPU 7	0	0

Final Word...

From http://lwn.net/Articles/129511/

Realtime preemption and read-copy-update (Posted Apr 1, 2005 5:56 UTC (Fri) by subscriber bronson) (Post reply)
Wow. Just when I thought Linux was getting good enough, that it has all the features I need for the forseeable future, along comes something like this that makes me say, I want I want I want!