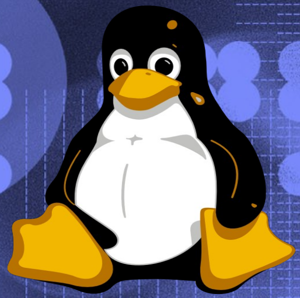
Real-Time Technologies in the Linux Kernel

Paul E. McKenney
IBM Distinguished Engineer & CTO Linux
Linux Technology Center



ACACES July 16, 2009

Course Objectives and Goals

- Introduction to Performance, Scalability, and Real-Time Issues on Modern Multicore Hardware: Is Parallel Programming Hard, And If So, Why?
- Performance and Scalability Technologies in the Linux Kernel
- Creating Performant and Scalable Linux Applications
- Real-Time Technologies in the Linux Kernel
- Creating Real-Time Linux Applications

Overview

- Why Parallel Real-Time Programming?
- Towards a Real-Time Linux Kernel
- Real-Time Linux Technologies
- Priority-Boosting Reader-Writer Locks
- Conclusions

Why Parallel Real-Time Programming?

Advent of SMP Embedded Realtime Systems

Traditional Systems

Traditional Realtime:
Few CPUs
Latency Guarantees
Non-Standard

OR

Traditional SMP:
Many CPUs
No Guarantees
Standard (and OSS)

But Not Both!!!

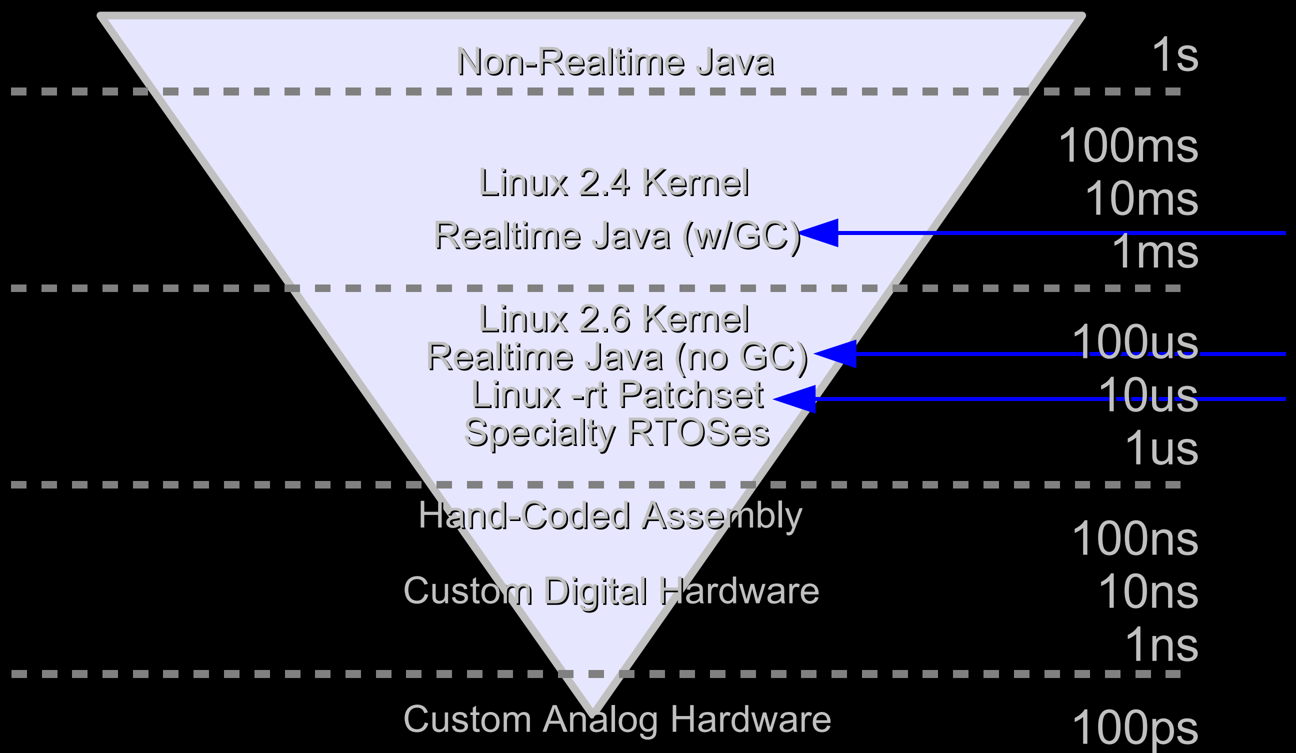
Convergence

Emerging Systems

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

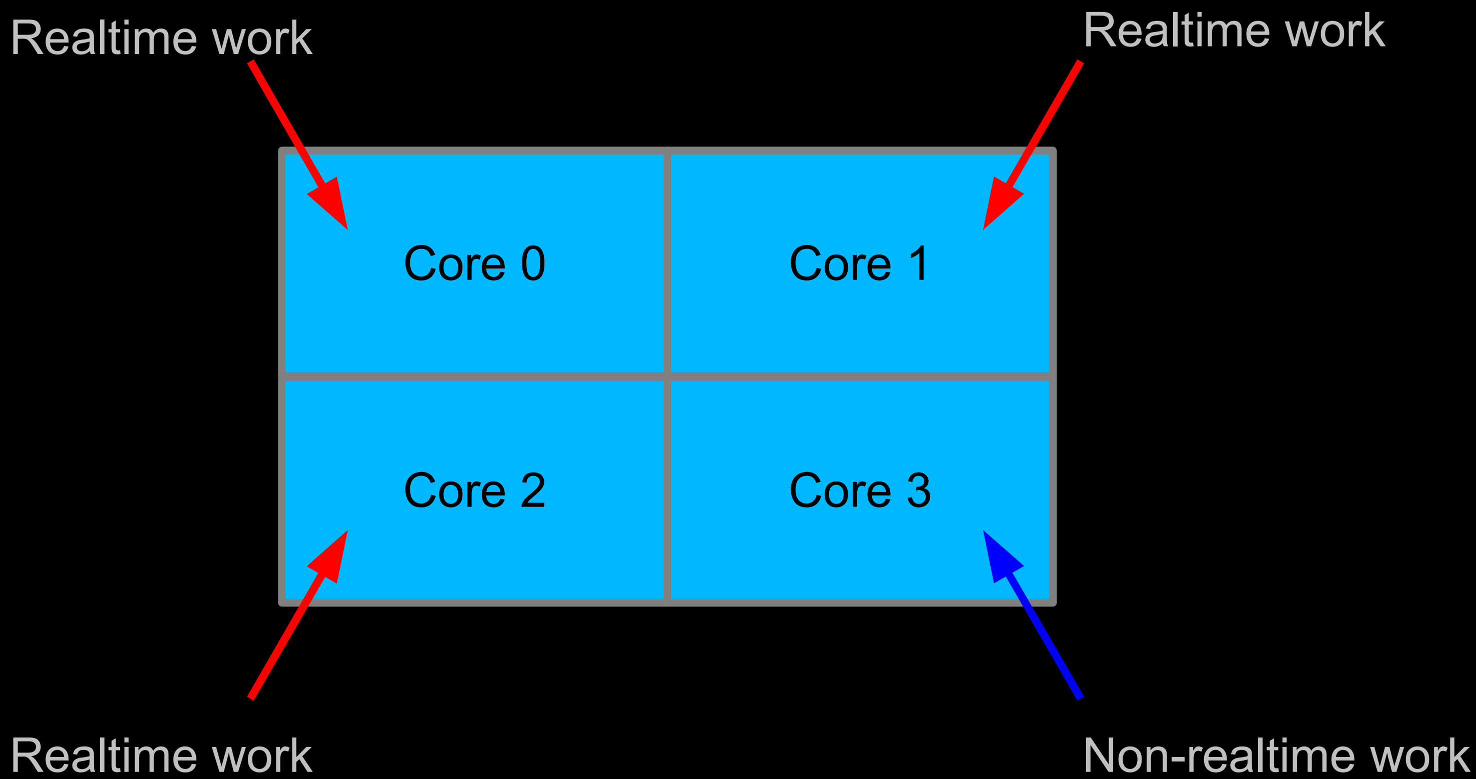
User Demand (DoD, Financial, Gaming, ...)
Techological Changes Leading to Commodity SMP
- Commodity Hardware Multithreading
- Commodity Multi-Core Dies
- Tens to Hundreds of CPUs per Die – Or More

Regimes of SMP Embedded Realtime Systems



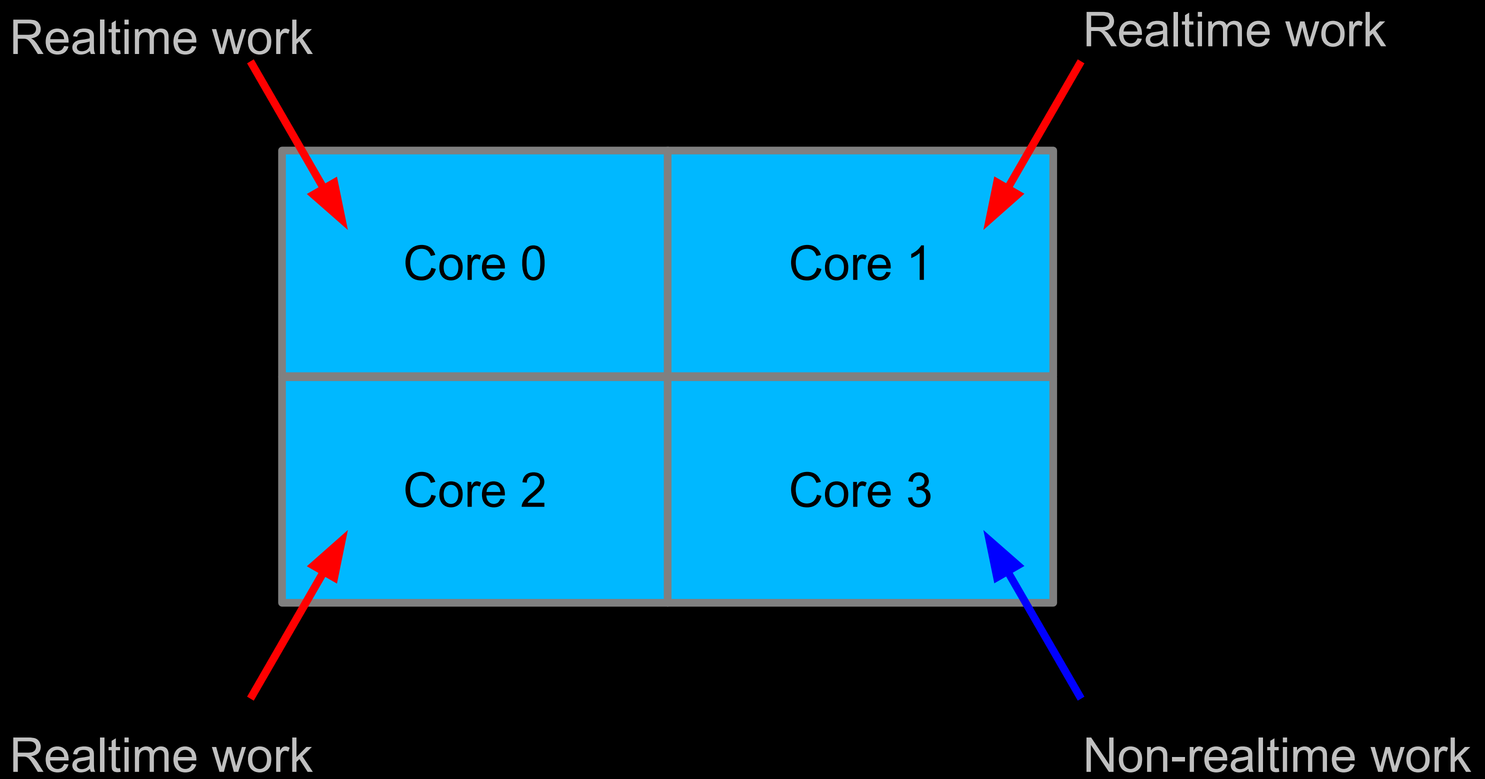
Towards a Real-Time Linux Kernel

2004: Prototype Multi-Core ARM Chip!!!



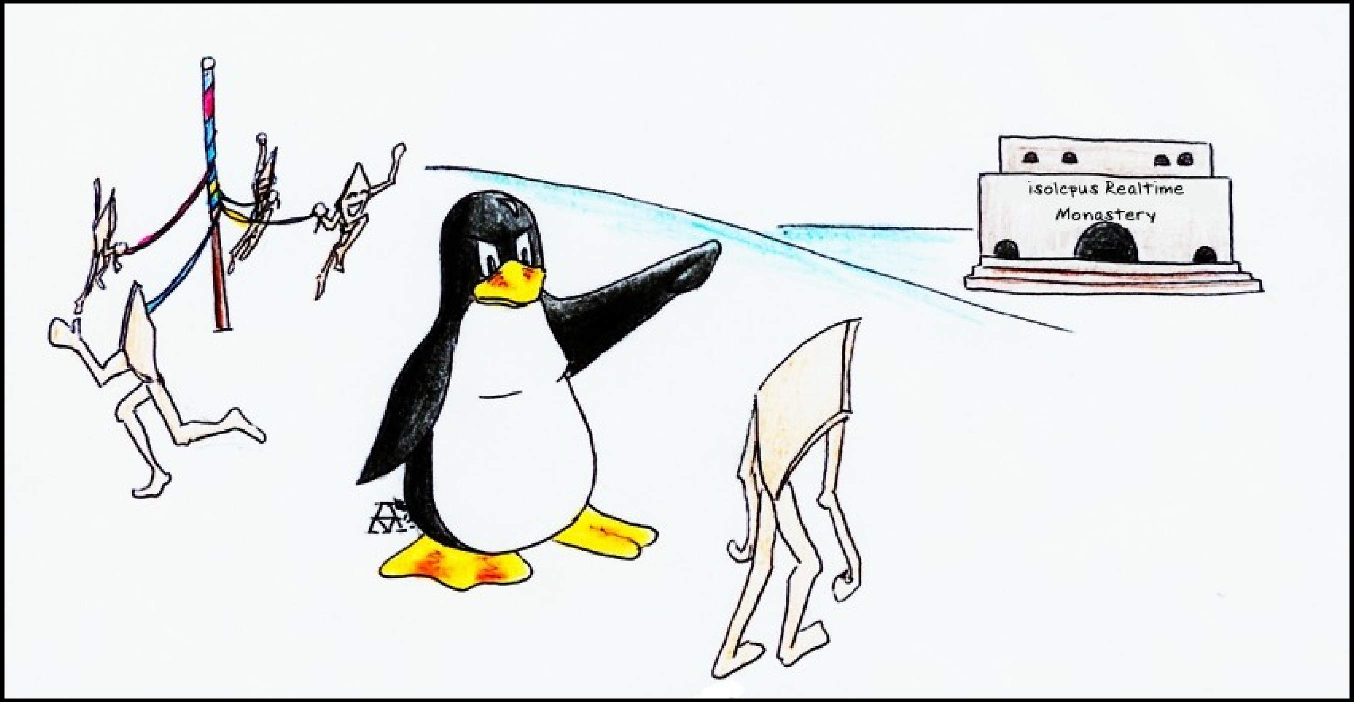
Submitted simple patch to Linux-kernel mailing list in 2004...

2004: Prototype Multi-Core ARM Chip!!!



Submitted simple patch to Linux-kernel mailing list in 2004...
And convinced my VP that real-time Linux was feasible.

Leveraging SMP Systems for Realtime



Useful approach in many cases – but not so good if *all* CPUs must do realtime...

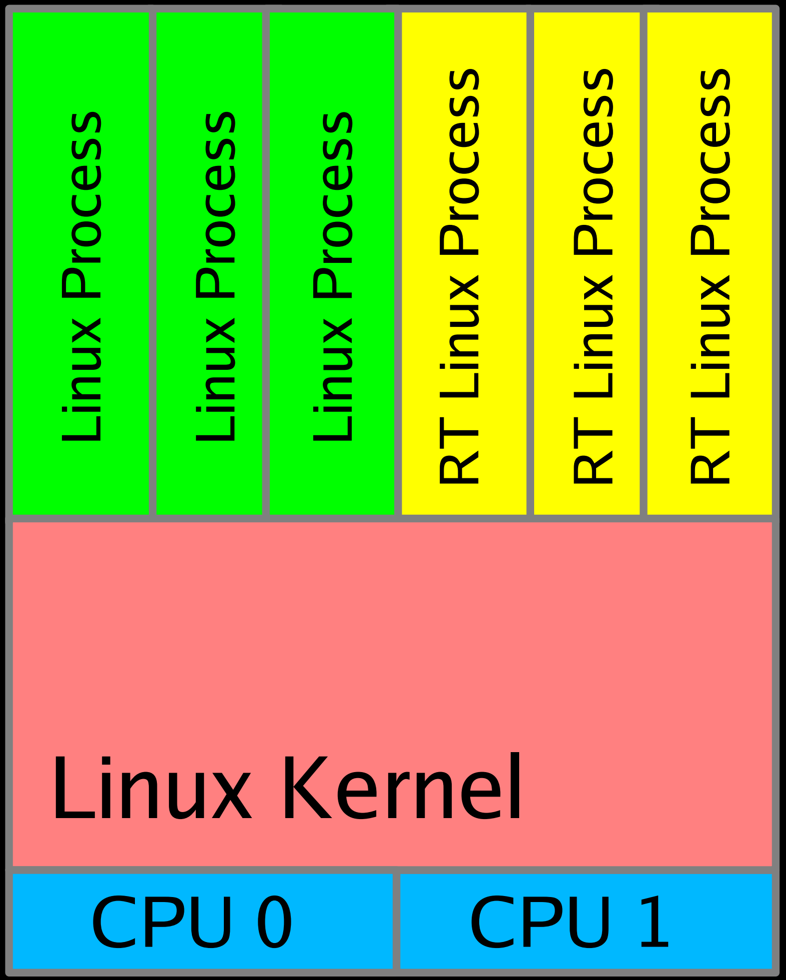
Therefore Joined Ingo Molnar's RT Linux Project

```
+       /*
+        * PREEMPT_RT semantics: different-type read-locks
+        * dont nest that easily:
+        */
+//     rcu_read_lock_read(&ptype_lock);
```

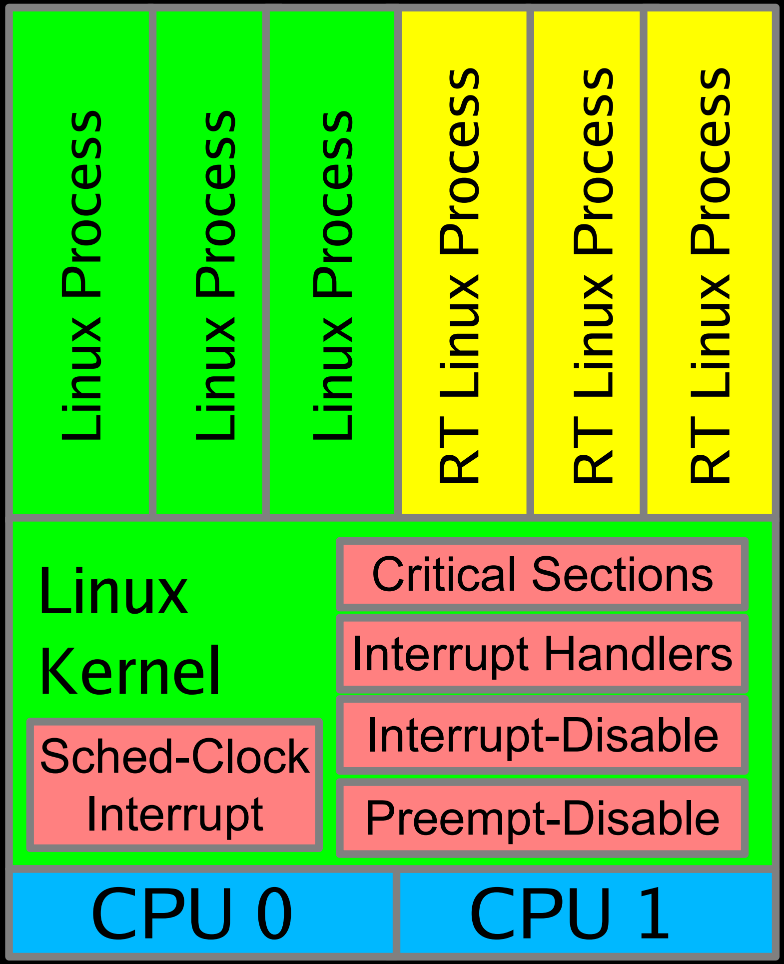
Preemptable RCU

- December 2004: realized that I fix RCU...
- March 2005: first hint that solution was possible
  - Esben Neilsen proposed flawed but serviceable approach
- May 2005: first design fixing Esben's flaws
- June 2005: first patch submitted to LKML
- August 2005: patch accepted in -rt
- November 2006: priority boosting patch
- Early 2007: priority boosting accepted into -rt
- September 2007: preemptable RCU w/o atomics
- January 2008: preemptable RCU in mainline
- Next: “evil plan” on later slide.

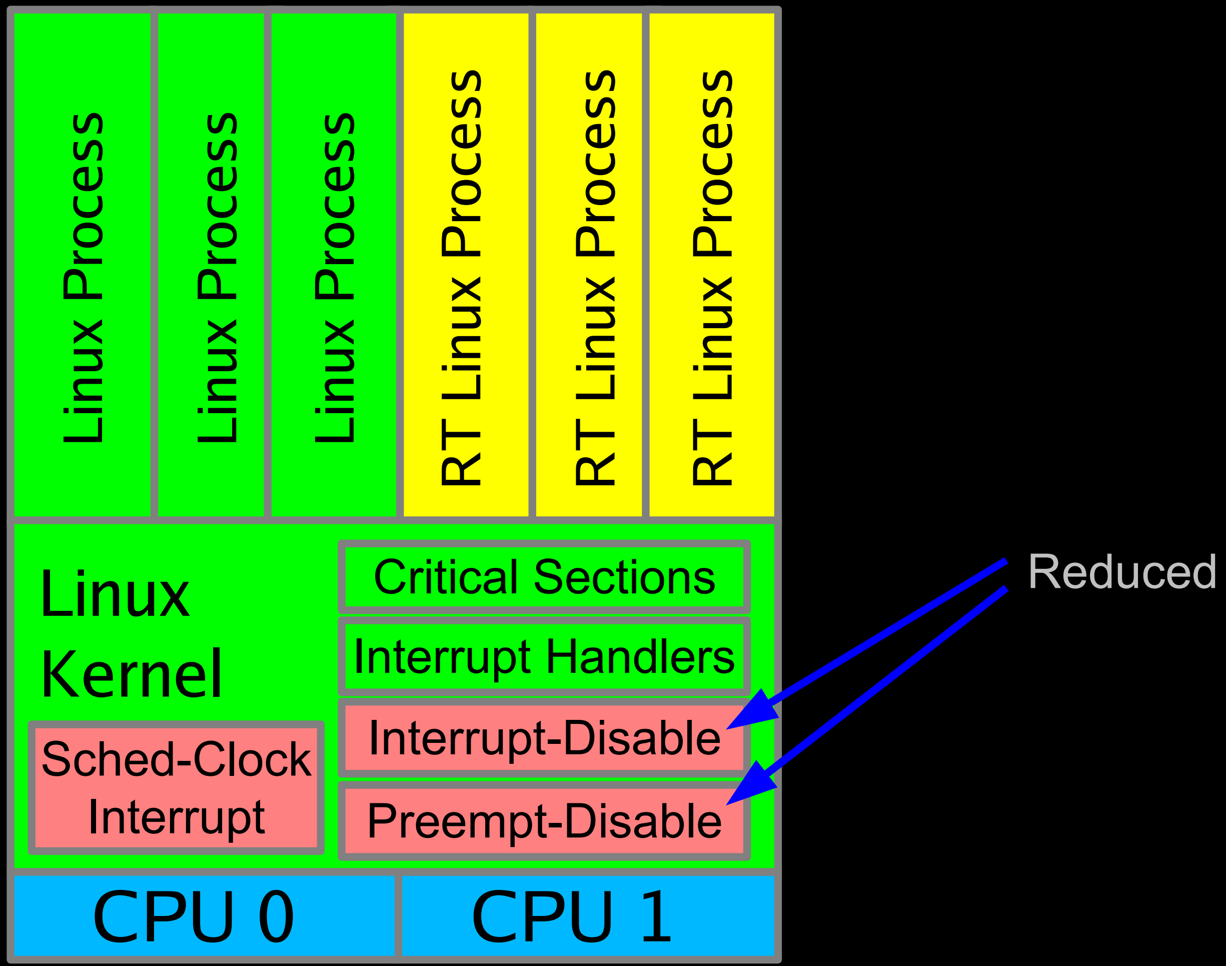
Vanilla Linux Kernel



Linux Kernel CONFIG_PREEMPT Build



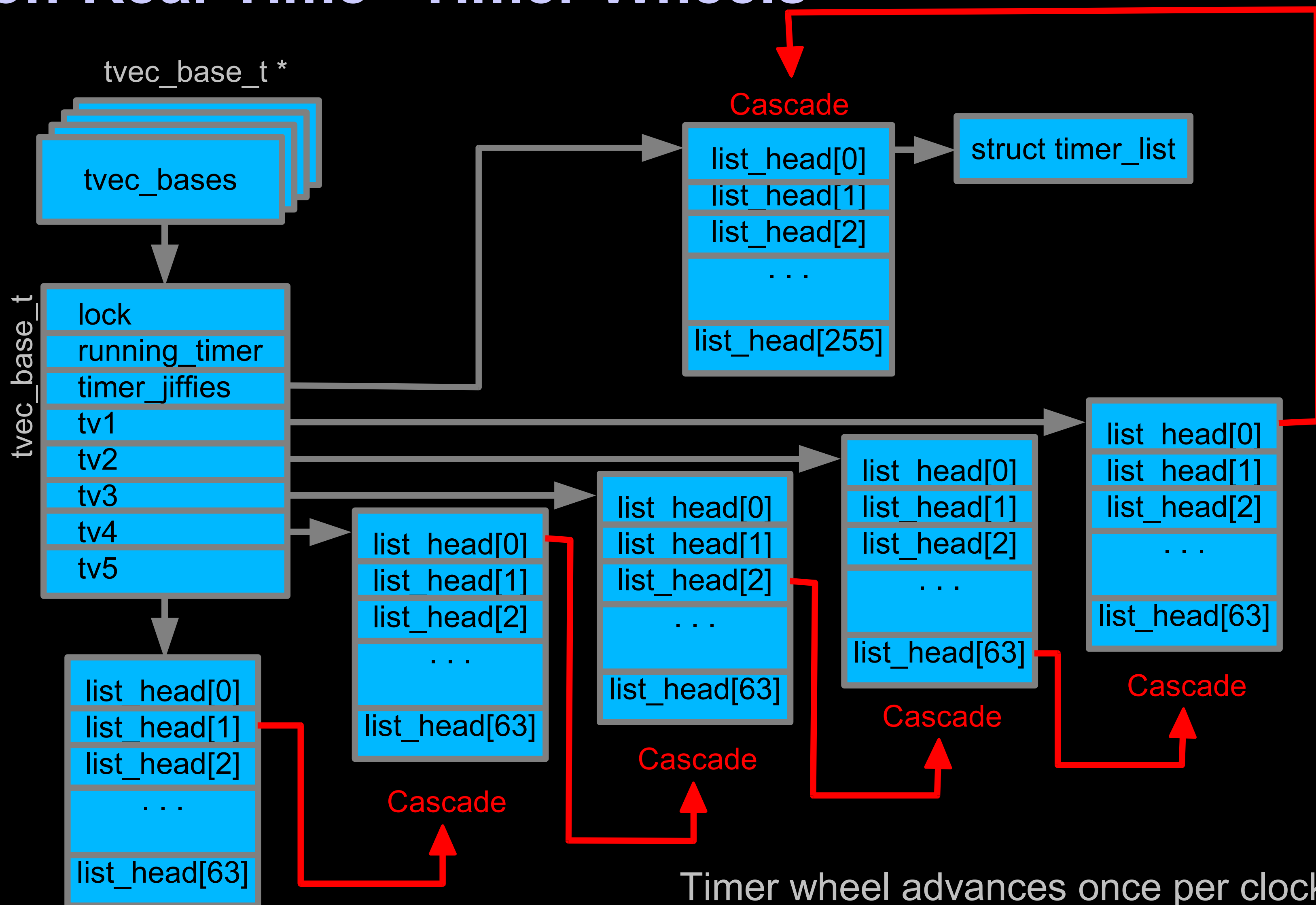
Linux Kernel CONFIG_PREEMPT_RT Build



10s of microseconds scheduling latency

Real-Time Linux Technologies

Non-Real-Time “Timer Wheels”



Timer wheel advances once per clock tick

Timer Wheels: Advantages and Disadvantages

- **Advantages:**
  - **O(1) insertion and removal operations**
  - **Batching of cascade operations improves throughput**
  - **Simple, well tested (both in Linux and elsewhere)**
- **Disadvantages:**
  - **Cascading operations *major*** latency hit!!!
  - Unforgiving tradeoff between accuracy and overhead
- But when you need tens-of-microseconds latencies for some applications...

Linux Timer Wheel at 1KHz

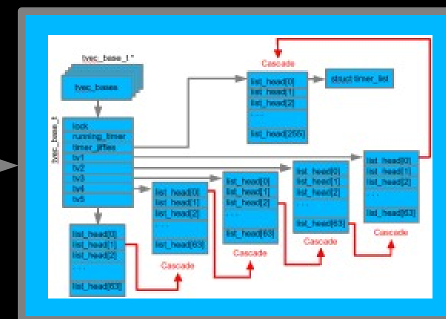
Linux Timer Wheel at 100KHz

Any Questions?

Solution: High-Resolution Timers

Timeouts: approximation OK, likely cancelled

add_timer(), mod_timer(), del_timer(), del_timer_sync(), ...



Timers: must be exact, rarely cancelled

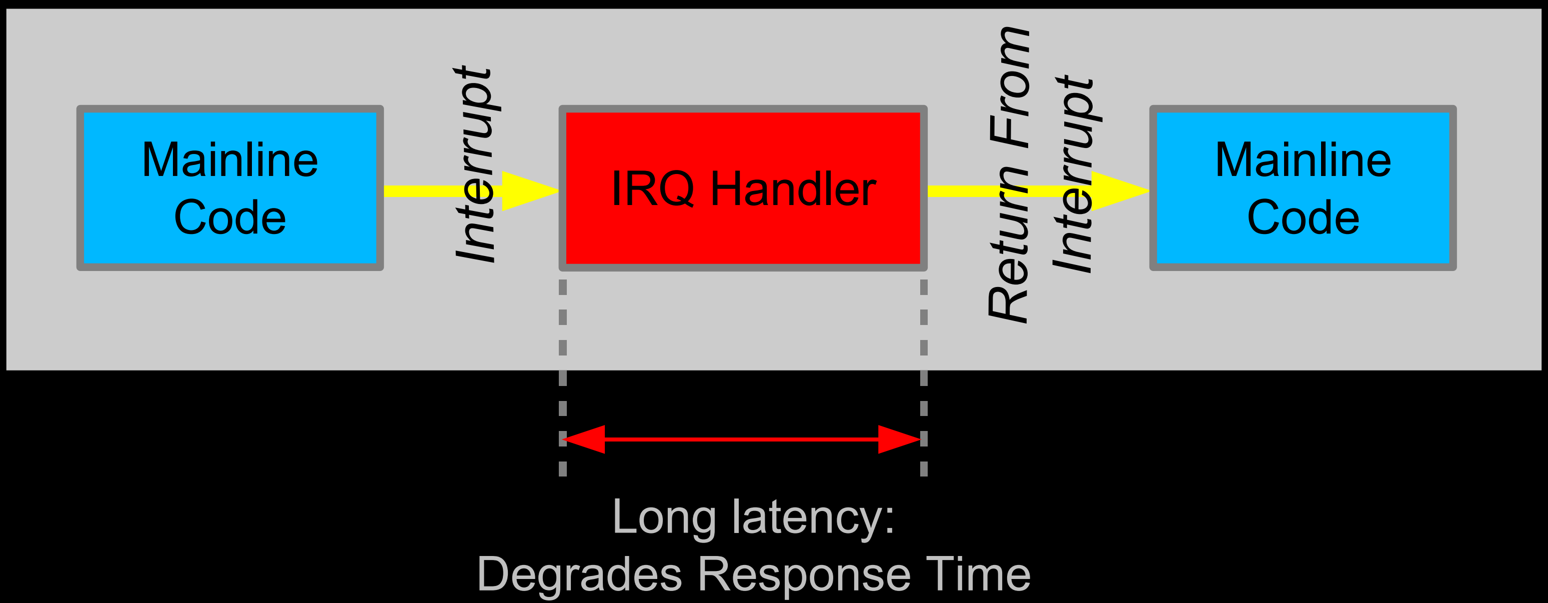
hrtimer_init(), hrtimer_init_sleeper(), hrtimer_start(), hrtimer_cancel(), hrtimer_forward(), ...

High-Resolution Timers Red-Black Tree

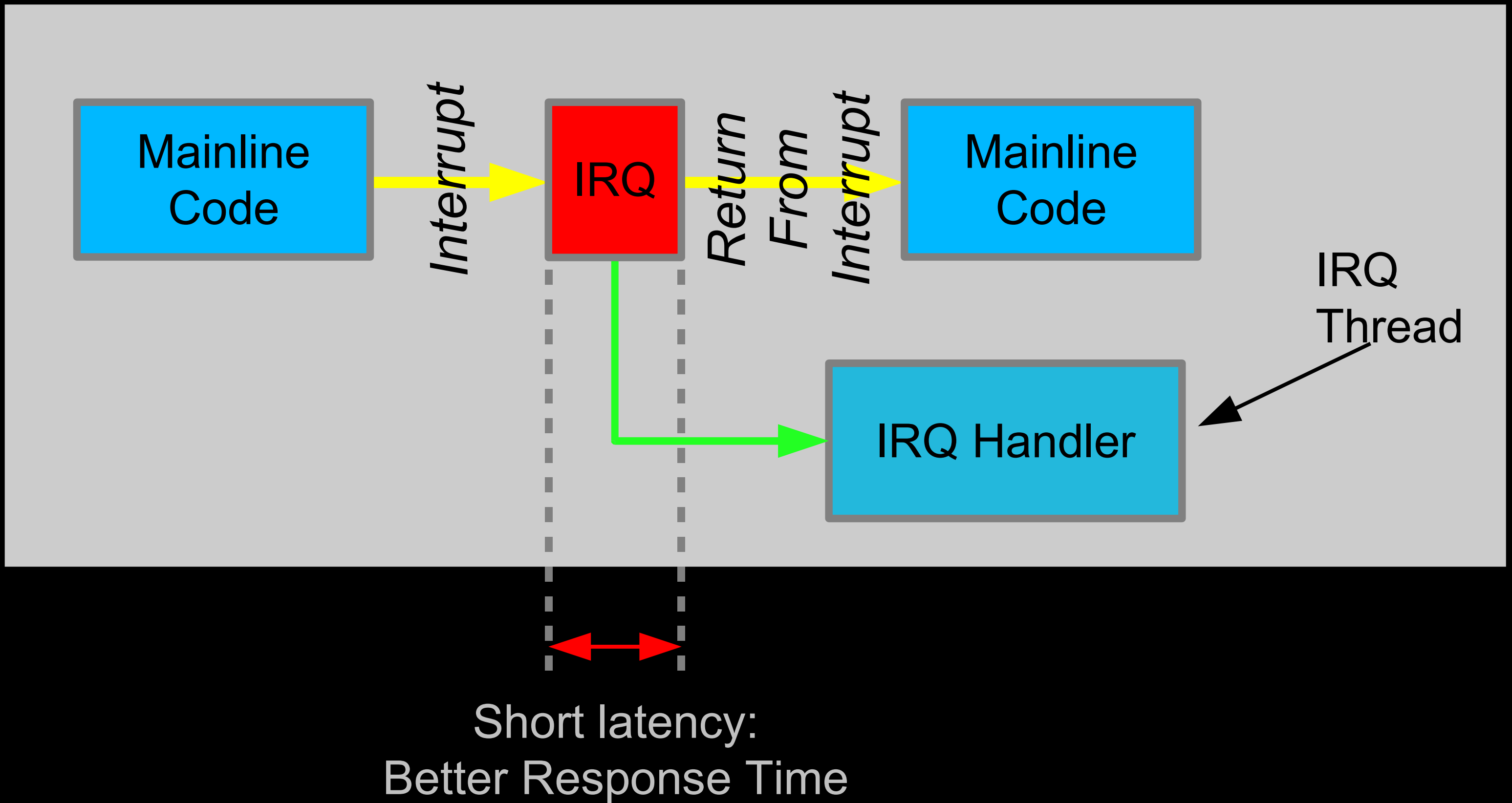
Preemptible Spinlocks

- Threads can be preempted while holding spinlocks
- Threads must therefore be permitted to block while acquiring spinlocks
  - Necessary to avoid self-deadlock scenario
- spinlock_t acquisition primitives can therefore block
- raw_spinlock_t provides “true spinlock” that disables preemption for special cases: scheduler, scheduling-clock interrupt
- Note that one uses the same primitives (e.g., spin_lock()) on both spinlock_t and raw_spinlock_t
- Requires threaded interrupt handlers...

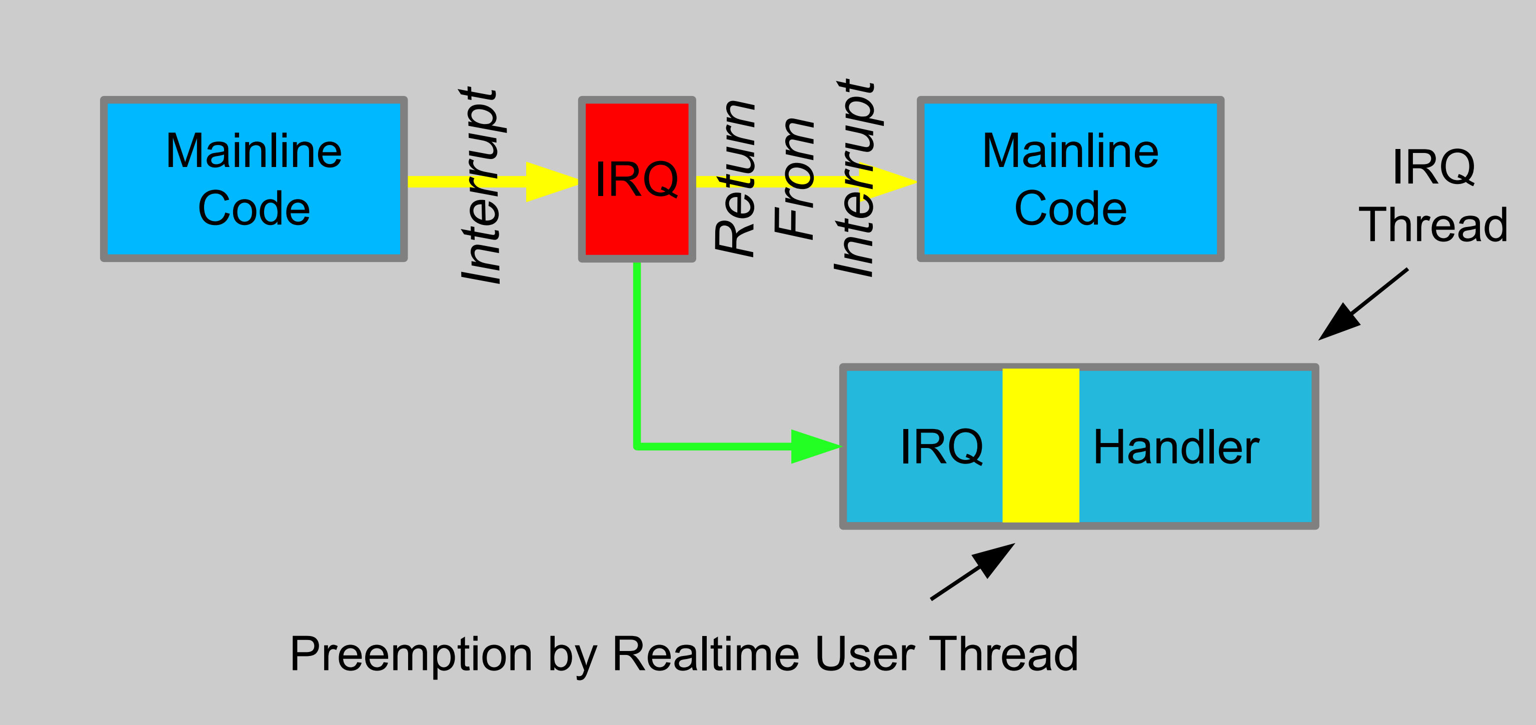
Linux's Non-Threaded Interrupt Handlers



-rt Patchset Threaded Interrupt Handlers



-rt Patchset Threaded Interrupt Handlers



Can get old hardirq behavior by specifying IRQ_NODELAY for given IRQ, but need very special handler: raw spinlocks, etc.

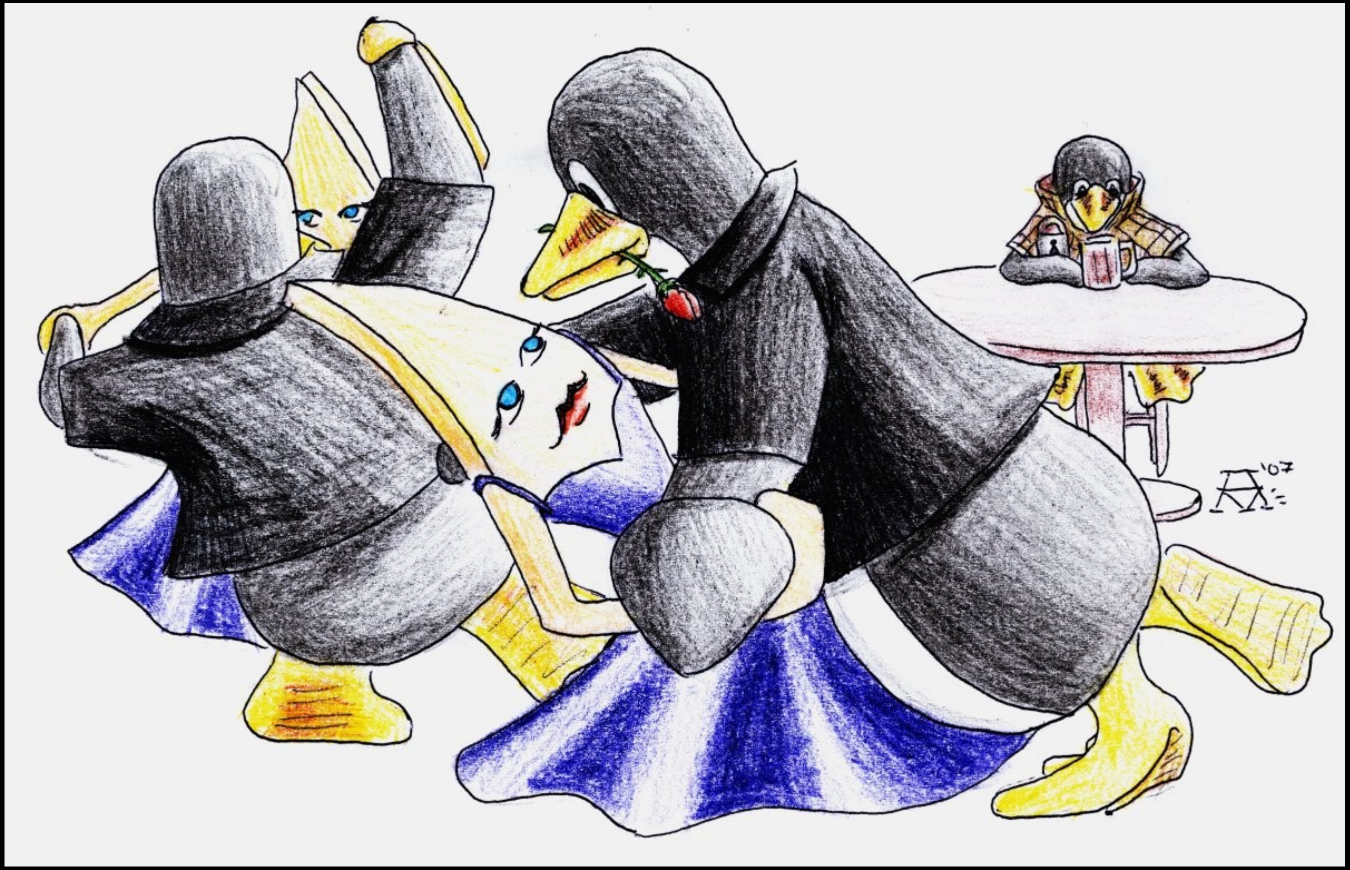
“Spiderman Principle”

Priority Inheritance

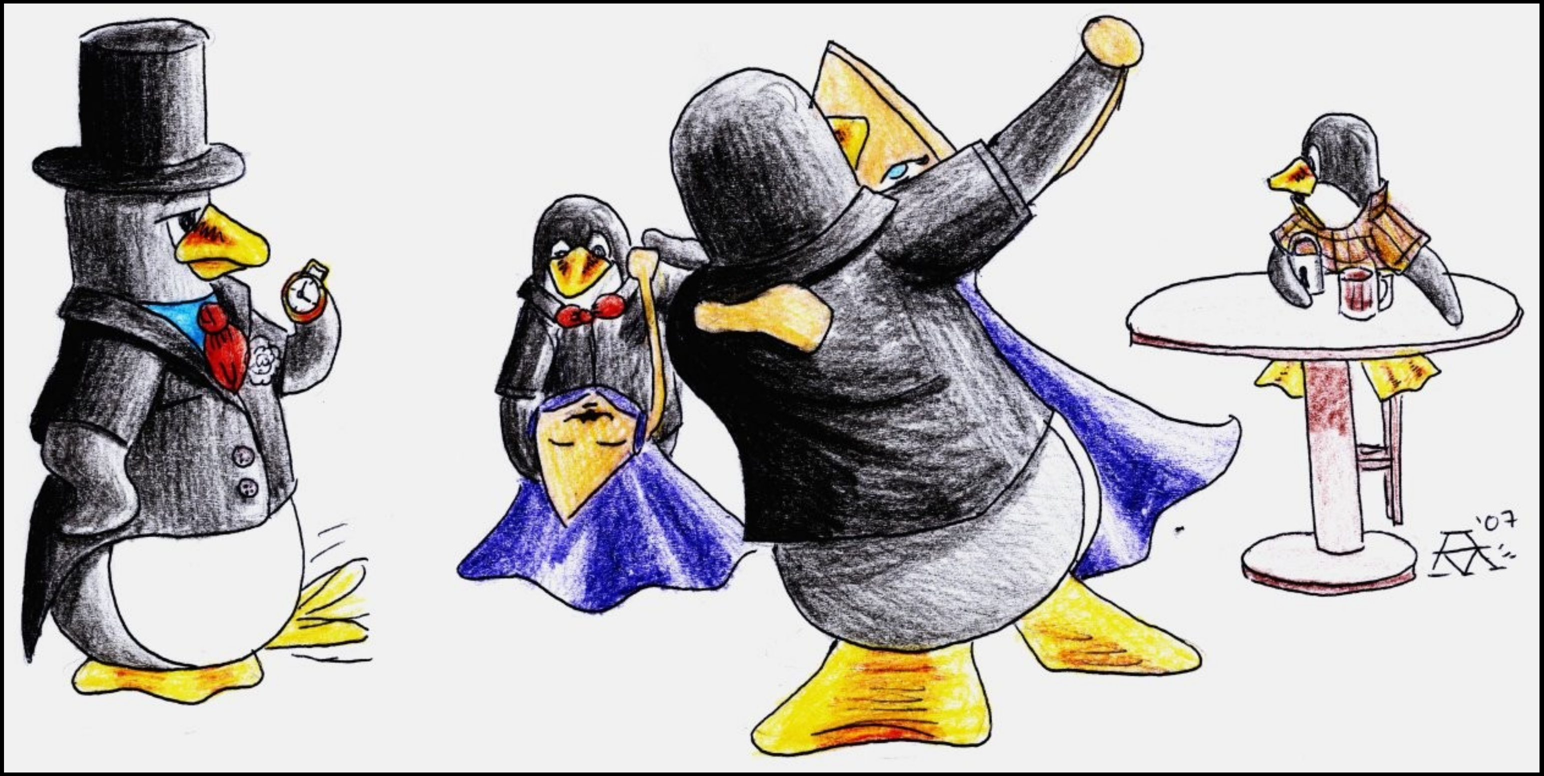
- "Trapdoor" Metaphor:
  - A dance floor...
    - CPUs dance with highest priority tasks (Tuxes)
  - Warning: any attempt to apply this metaphor in reverse will probably not end well...

Priority Inheritance

Priority Inheritance



Priority Inheritance

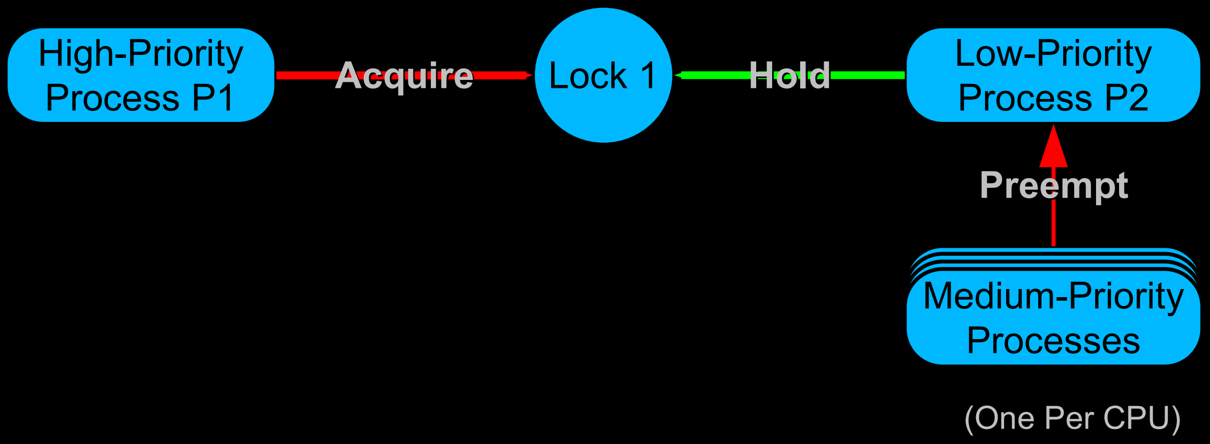


Priority Inheritance

Priority Inheritance

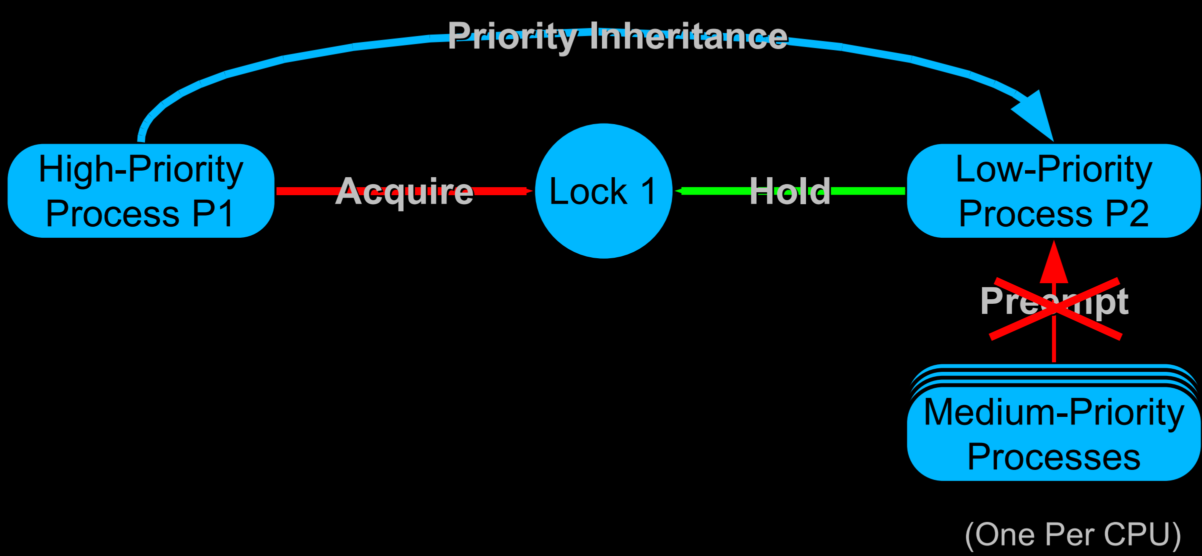
Priority Inversion Outside the Dance Hall

- Process P1 needs Lock L1, held by P2
- Process P2 has been preempted by medium-priority processes
  - Consuming all available CPUs
- Process P1 is blocked by lower-priority processes



Preventing Priority Inversion

- Trivial solution: Prohibit preemption while holding locks
  - But degrades latency!!! Especially for sleeplocks!!!!
- Simple solution: “Priority Inheritance”: P2 “inherits” P1's priority
  - But only while holding a lock that P1 is attempting to acquire
  - Standard solution, very heavily used
- Either way, prevent the low-priority process from being preempted



Limits to Priority Boosting

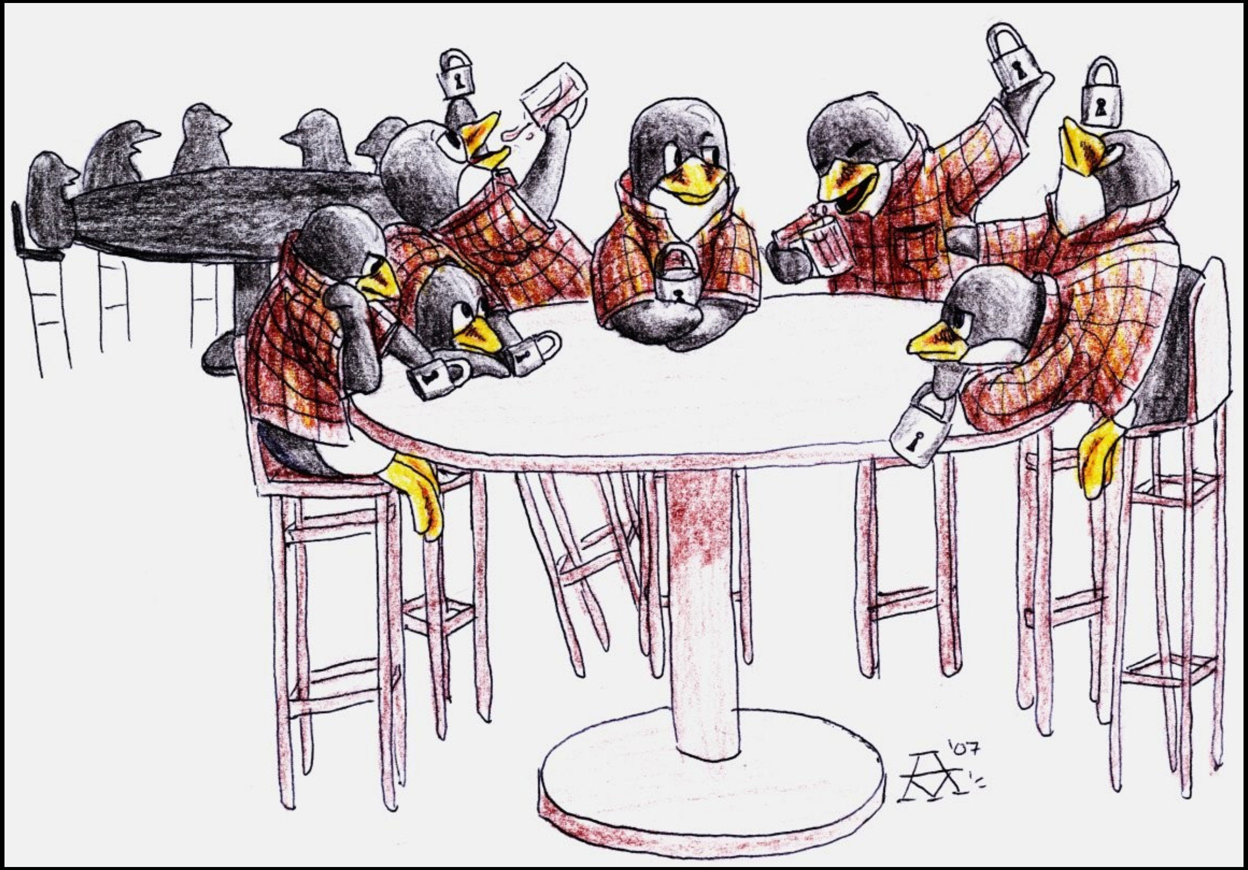
- **Inappropriate for ultimate in responsiveness**
  - **Then again, the same is true for digital hardware**
- **Does not work for events – who will raise the event?**
- **Does not work for memory exhaustion – who will free memory?**
- **Does not work for mass storage – make the disk spin faster???**
- **Does not work for network receives – boostee on other machine!**
  - ***Could*** do cross-system boosting
  - But there are limits (see next slide)
- Does not work for reader-writer locking
  - At least not very well (see following section)

In Some Cases, Priority Boosting Undesirable...

...Or At Least Uncomfortable!!!

Priority-Boosting Reader-Writer Locks

Back at the Dance Hall...

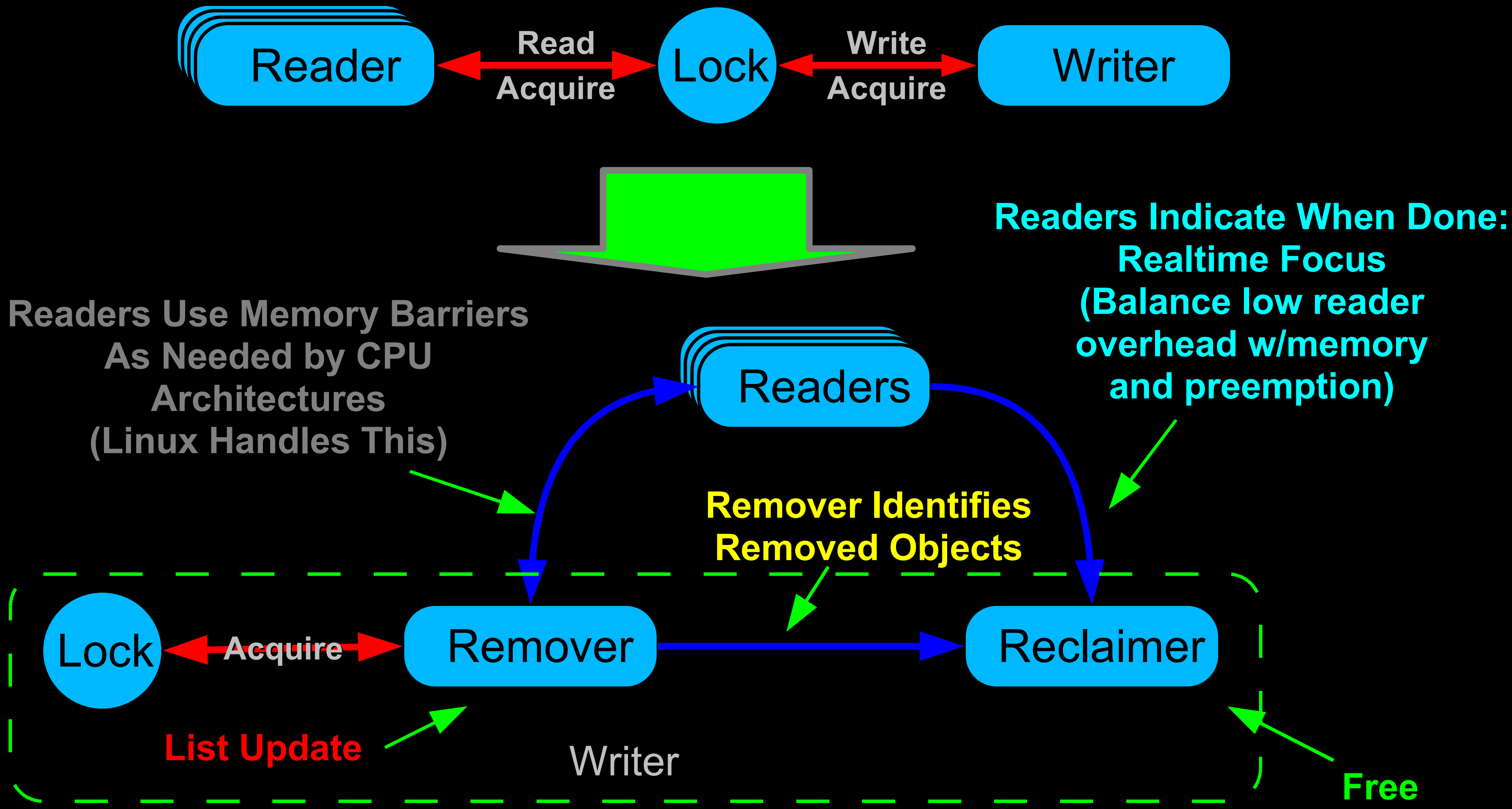


...Our High-Priority CPU May Have to Wait Awhile!!!

Priority Inheritance and Reader-Writer Locking

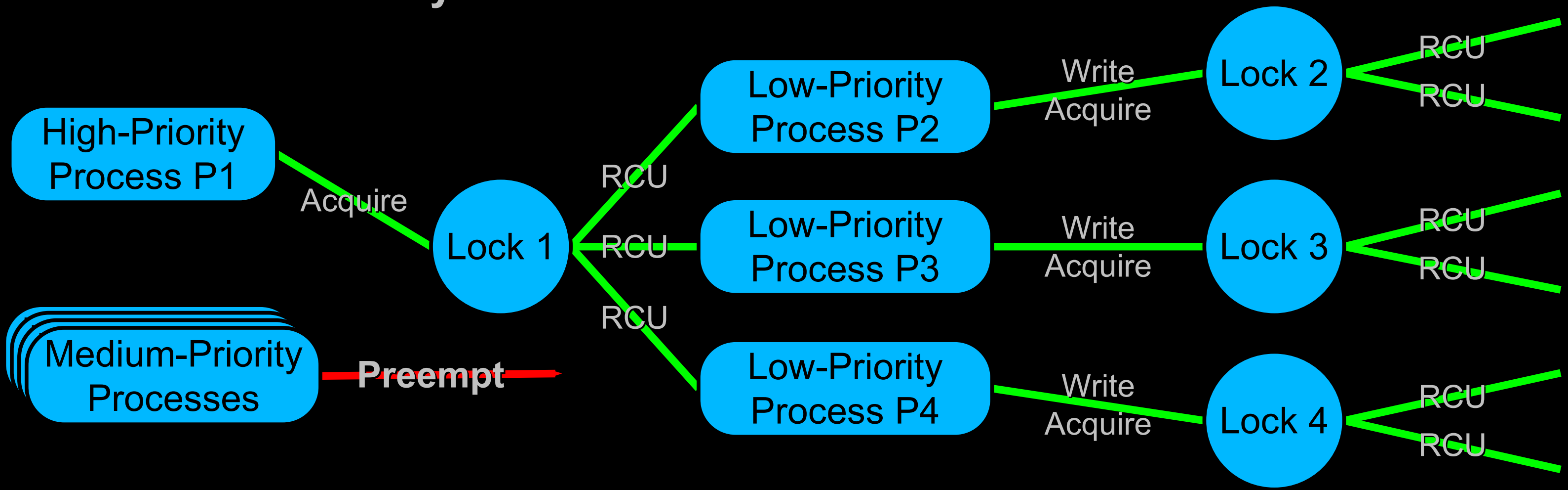
- Real-time operating systems have taken the following approaches to writer-to-reader priority boosting:
  - Boost only one reader at a time
    - Reasonable on a single-CPU machine, except in presence of readers that can block for other reasons.
    - Extremely ineffective on an SMP machine, as the writer must wait for readers to complete serially rather than in parallel
  - Boost a number of readers equal to the number of CPUs
    - Works well even on SMP, except in presence of readers that can block for other reasons (e.g., acquiring other locks)
  - Permit only one task at a time to read-hold a lock (PREEMPT_RT)
    - Very fast priority boosting, but severe read-side locking bottlenecks
- All of these approaches have heavy bookkeeping costs
  - Priority boost propagates transitively through multiple locks
  - Processes holding multiple locks may receive multiple priority boosts to different priority levels, actual boost must be to maximum level
  - Priority boost reduced (perhaps to intermediate level) when locks released
- So -rt patchset permits only one reading task at a time on a given lock
  - How to deal with this scalability limitation???

Analogy: Reader-Writer Lock vs. RCU

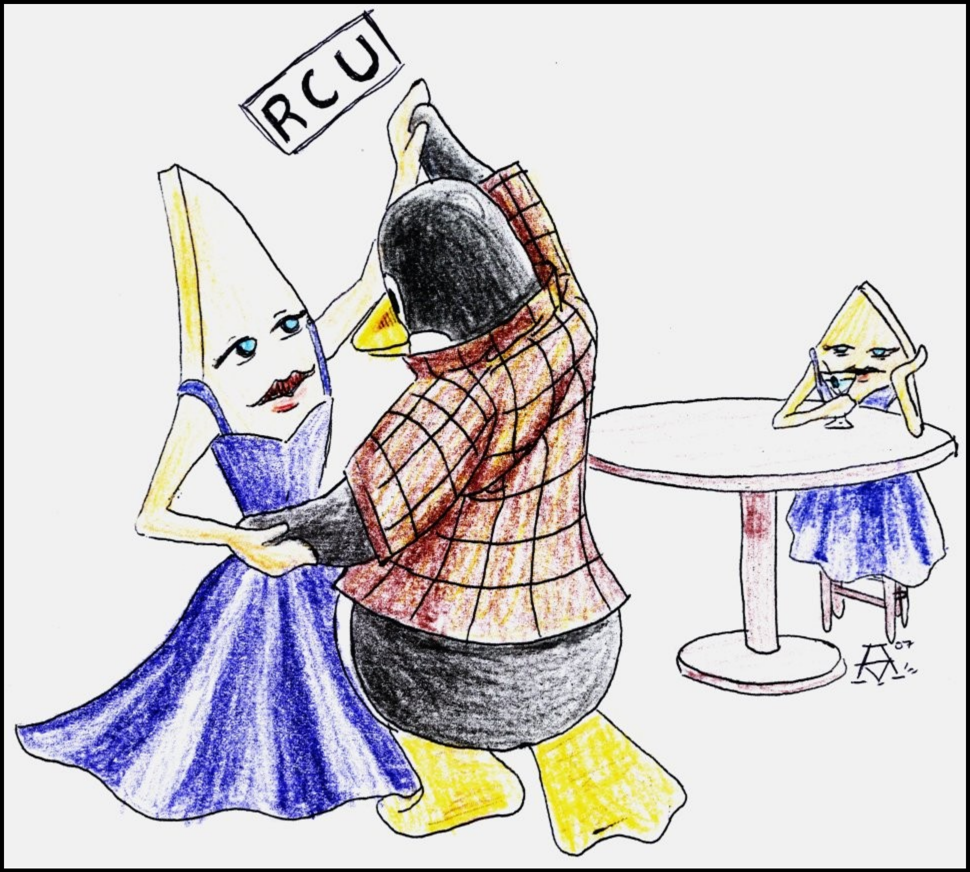


Priority Inversion and RCU

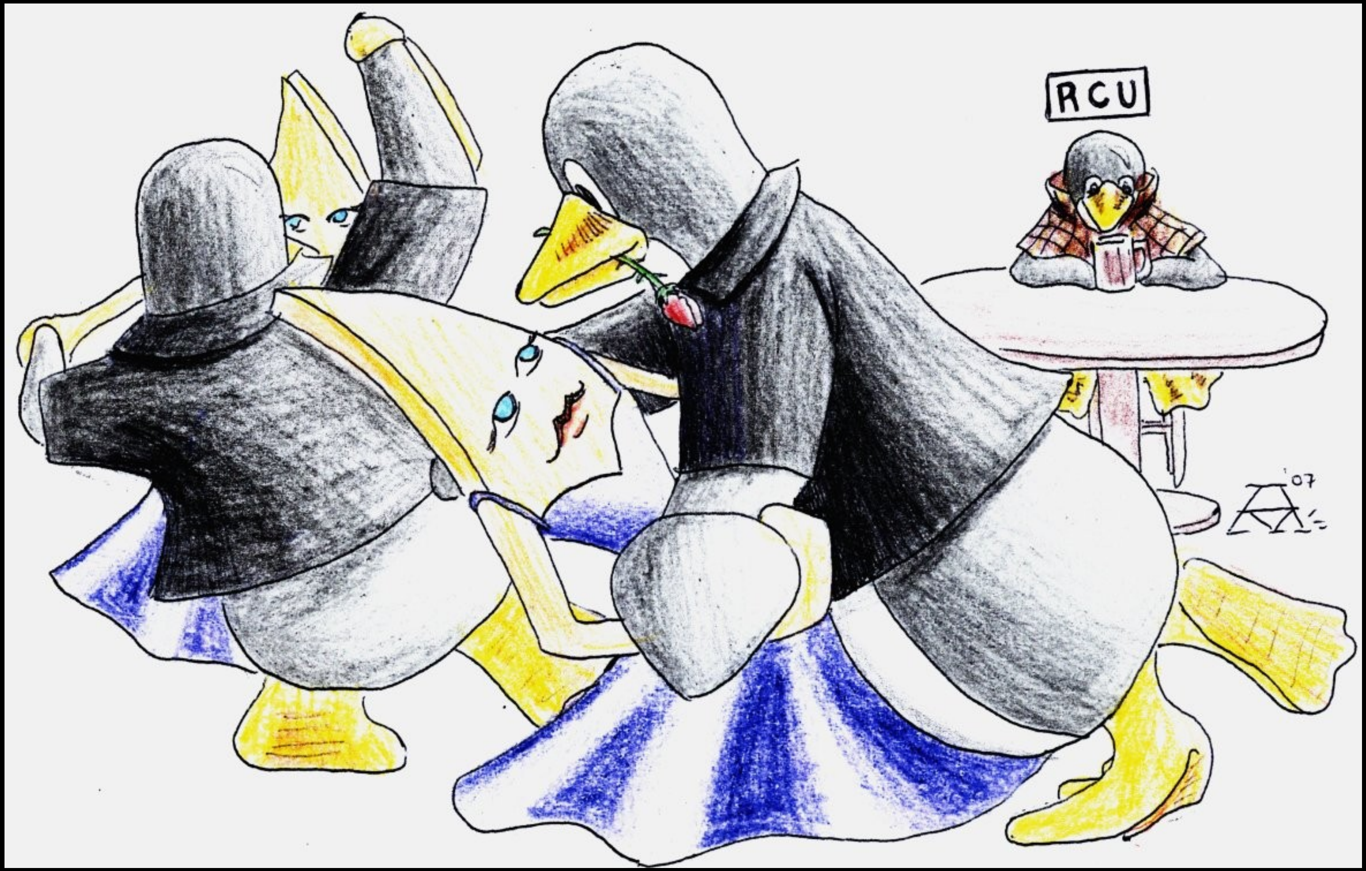
- Process P1 needs Lock L1, but P2, P3, and P4 now use RCU
  - P2, P3, and P4 therefore need not hold L1
  - Process P1 thus immediately acquires this lock
  - Even though P2, P3, and P4 are preempted by the per-CPU medium-priority processes
- No priority inheritance required
  - Except if low on memory: 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...



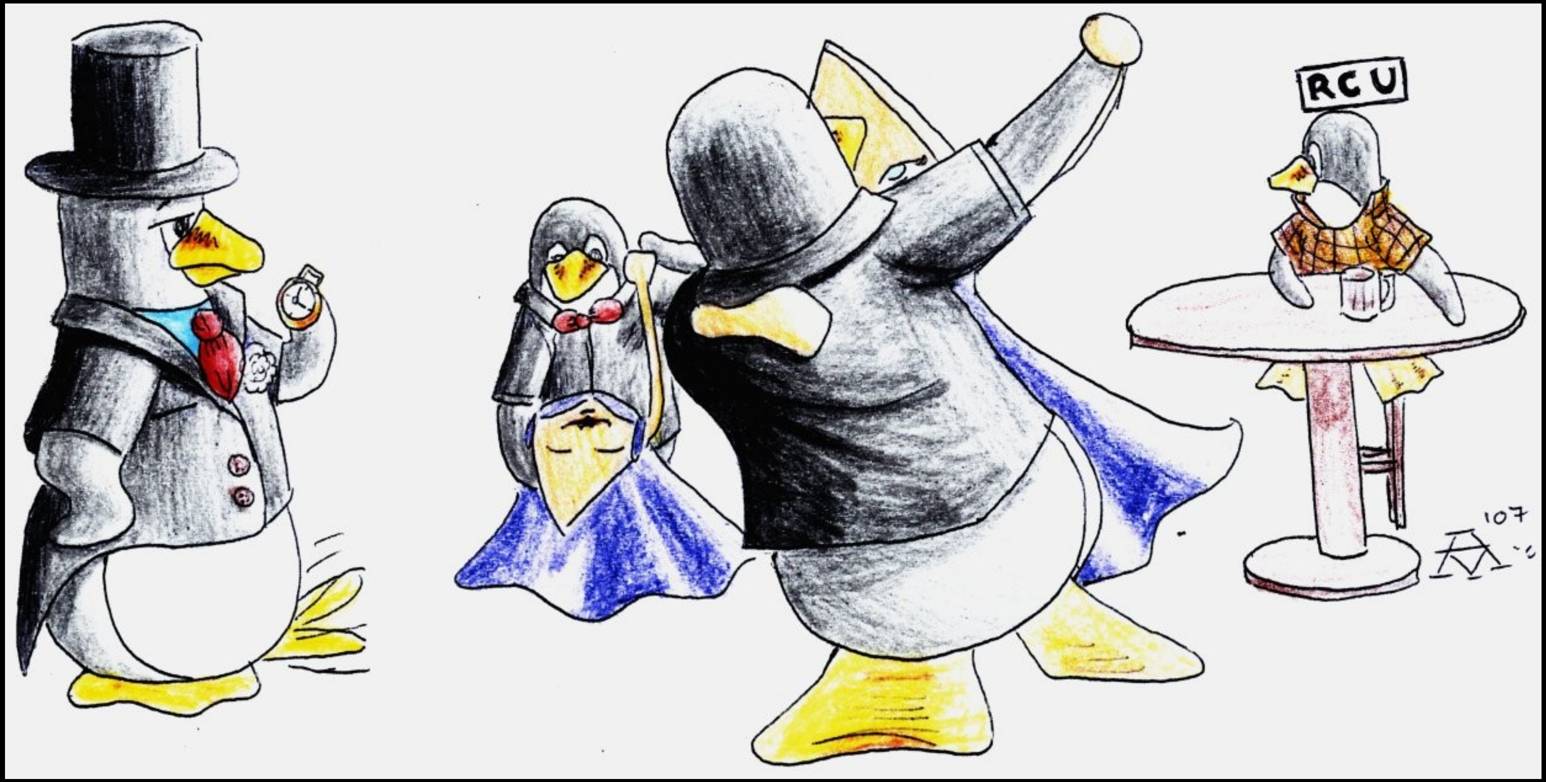
Priority Inversion and RCU



Priority Inversion and RCU



Priority Inversion and RCU



Priority Inversion and RCU

Realtime and RCU

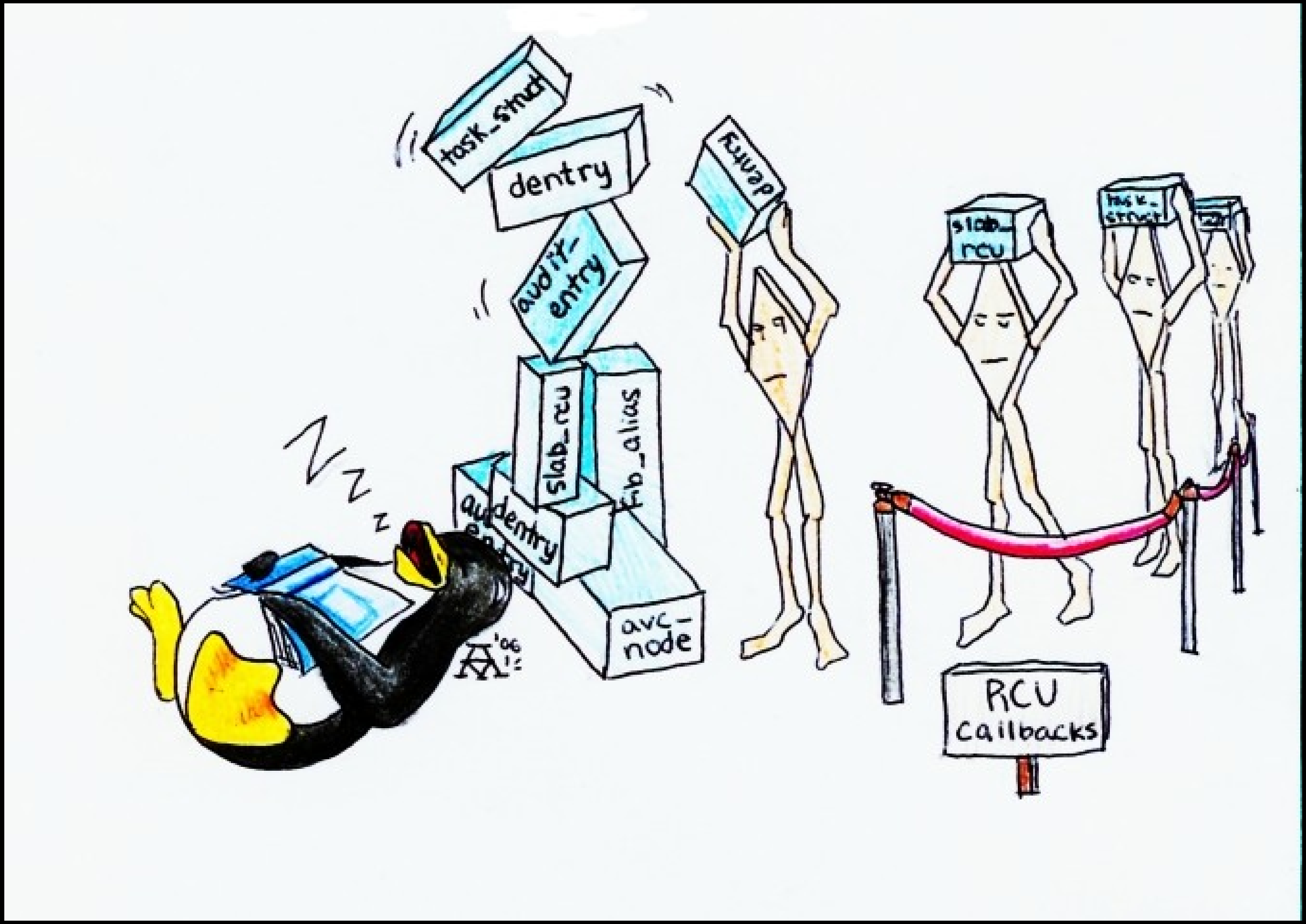
- **RCU exploited in PREEMPT_RT patchset to reduce latencies**
  - **"kill()" system-call RCU provided large reduction in latency**
  - **Expect similar benefits for pthread_cond_broadcast() and pthread_cond_signal()**
  - **Work ongoing in protocol stacks**
    - **Which is requiring an expedited-grace-period RCU implementation**
- **Current PREEMPT_RT realtime Linux provides relatively few realtime services**
  - **Process scheduling, interrupts, some signals**
- **Increasing the number of realtime services will likely require additional exploitation of RCU**
  - And will likely require that RCU readers be priority-boosted when low on memory
- But "Classic RCU" has realtime-latency problems of its own!!!
  - Classic RCU disables preemption across read-side critical sections...

What is Needed From Realtime RCU?

- **Reliable**
- **Callable from IRQ**
- ***Preemptible read-side critical sections***
- ***Small memory footprint***
- Synchronization-free read side
- Independent of memory-allocator data structures
- Freely nestable read side
- Unconditional read-to-write upgrade
- API compatible with “Classic RCU”

Why small memory footprint???

But Can't *Just* Make RCU Preemptible...



Small memory footprint means timely grace-period processing...

Overhead of RT RCU Read-Side....

- Heavier weight than the classic RCU implementations
- But still:
  - No locks
  - No loops
  - No atomic instructions
  - No memory barriers
- So still lightweight with *O(1)* worst-case execution time
  - And many implementations have *fixed* execution time

Accepted into 2.6.25 on January 25, 2008

Real-Time rcu_read_lock()

```
void rcu_read_lock(void)
{
        int idx;
        struct task_struct *t = current;
        int nesting;

        nesting = ACCESS_ONCE(t->rcu_read_lock_nesting);
        if (nesting != 0) {
                t->rcu_read_lock_nesting = nesting + 1;
        } else {
                unsigned long flags;

                local_irq_save(flags);
                idx = ACCESS_ONCE(rcu_ctrlblk.completed) & 0x1;
                ACCESS_ONCE(RCU_DATA_ME()->rcu_flipctr[idx])++;
                ACCESS_ONCE(t->rcu_read_lock_nesting) = nesting + 1;
                ACCESS_ONCE(t->rcu_flipctr_idx) = idx;
                local_irq_restore(oldirq);
        }
}
```

Real-Time rcu_read_unlock()

```
void __rcu_read_unlock(void)
{
        int idx;
        struct task_struct *t = current;
        int nesting;

        nesting = ACCESS_ONCE(t->rcu_read_lock_nesting);
        if (nesting > 1) {
                t->rcu_read_lock_nesting = nesting - 1;
        } else {
                unsigned long flags;

                local_irq_save(flags);
                idx = ACCESS_ONCE(t->rcu_flipctr_idx);
                ACCESS_ONCE(t->rcu_read_lock_nesting) = nesting - 1;
                ACCESS_ONCE(RCU_DATA_ME()->rcu_flipctr[idx])--;
                local_irq_restore(flags);
        }
}
```

Evil Plan for Real-Time rcu_read_{,un}lock()

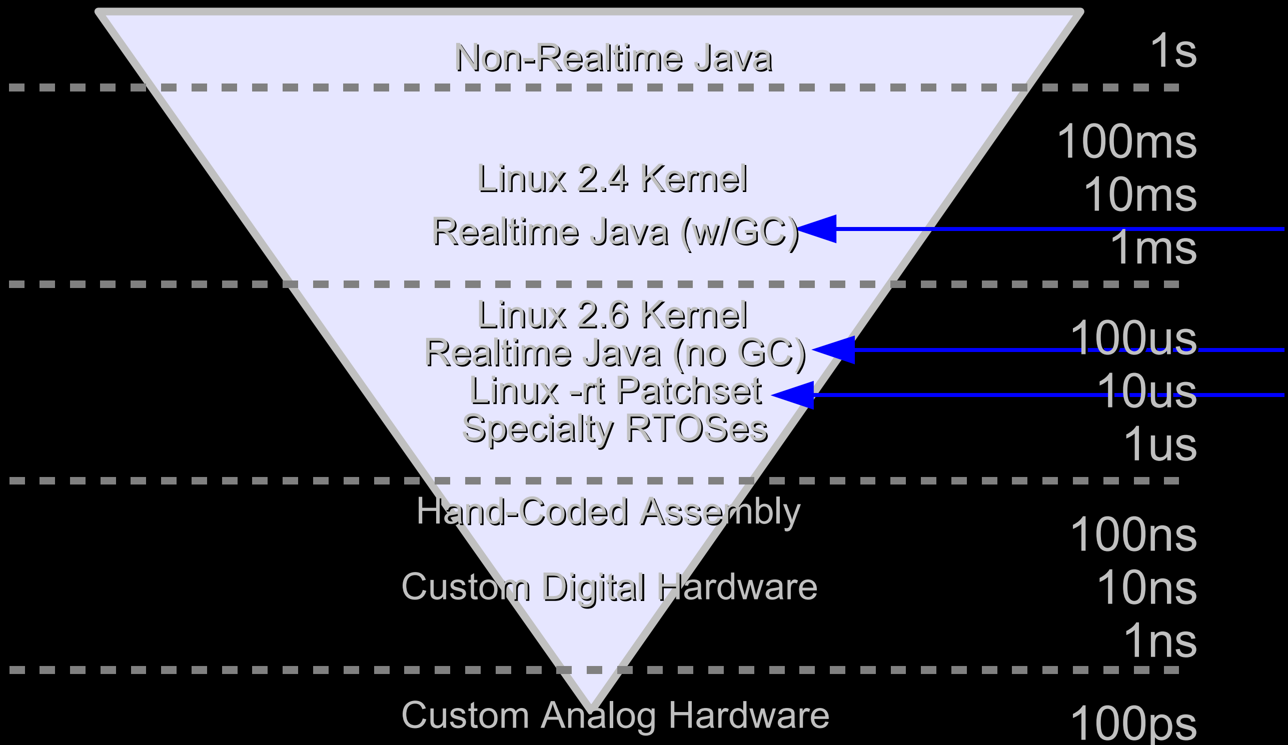
```
void _rcu_read_lock(void)
{
        ACCESS_ONCE(current->rcu_read_lock_nesting)++;
        barrier();
}

void _rcu_read_unlock(void)
{
        struct task_struct *t = current;

        barrier();
        if (--ACCESS_ONCE(t->rcu_read_lock_nesting) == 0 &&
            unlikely(ACCESS_ONCE(t->rcu_read_unlock_special)))
                _rcu_read_unlock_special(t);
}
```

Conclusions

Conclusions



Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM and IBM (logo) are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.

Questions?

To probe further:

- **General Information:**
  - **http://rt.wiki.kernel.org/index.php/Main_Page (-rt wiki)**
  - **http://www.kernel.org/pub/linux/kernel/projects/rt/ (-rt downloads)**
  - **http://lwn.net/Articles/310391/ (new -rt tree)**
- **Offerings:**
  - **people.redhat.com/bche/presentations/realtime-linux-summit08.pdf**
  - **http://news.com.com/Novell+to+launch+quick-response+Linux/2100-7344_3-6117479.html**
  - **http://www.mvista.com/products/realtime.html**
  - **http://www.linutronix.de/**
  - **http://www.ibm.com/software/webservers/realtime/**
- **Locking:**
  - **http://lwn.net/Articles/271817/ (Adaptive spinlocks)**
  - **http://lwn.net/Articles/267968/ (Ticket locks for determinism)**
  - **http://lwn.net/Articles/178253/ (Priority inheritance in the Linux kernel)**

"Controlling a laser with Linux is crazy, but everyone in this room is crazy in his own way. So if you want to use Linux to control an industrial welding laser, I have no problem with your using PREEMPT_RT." -- Linus Torvalds, July 2006

Questions?

To probe further:

- **Threaded Interrupt Handlers:**
  - **http://lwn.net/Articles/106010/ (Approaches, October 2004)**
  - **http://lwn.net/Articles/138174/ (Debate, June 2005)**
  - **http://lwn.net/Articles/139062/ (softirq splitting, June 2005)**
  - **http://lwn.net/Articles/302043/ (Moving interrupts to threads, October 2008)**
  - **http://lwn.net/Articles/321663/ (Threaded interrupts and lockdep, March 2009)**
- **Timers:**
  - **http://lwn.net/Articles/152363/ (rationale for timer/hrtimer split)**
  - **http://lwn.net/Articles/152436/ (timer implementation)**
  - **http://lwn.net/Articles/167897/ (high-resolution timer API – dated)**
  - **http://lwn.net/Articles/228143/ (deferrable timers)**

"Controlling a laser with Linux is crazy, but everyone in this room is crazy in his own way. So if you want to use Linux to control an industrial welding laser, I have no problem with your using PREEMPT_RT." -- Linus Torvalds, July 2006

Questions?

To probe further:

- Real-Time RCU:
  - http://www.rdrop.com/users/paulmck/RCU/realtimeRCU.2004.06.12a.pdf
    - Making RCU Safe for Deep Sub-Millisecond Response Realtime Applications, Sarma & McKenney
  - http://lkml.org/lkml/2004/8/30/87 (Jim Houston's implementation)
  - http://lwn.net/Articles/107269/ (Need for real-time RCU noted, October 2004)
  - http://lwn.net/Articles/129511/ (First limping real-time RCU, March 2005)
  - http://www.rdrop.com/users/paulmck/RCU/realtimeRCU.2005.04.23a.pdf
    - Towards Hard Realtime Response from the Linux Kernel on SMP Hardware, McKenney & Sarma
  - http://lwn.net/Articles/220677/ (RCU priority boosting, February 2007)
  - http://lwn.net/Articles/253651/ (Design of preemptible RCU, October 2007)
  - http://lwn.net/Articles/279077/ (dynticks and preemptible RCU)
  - The read-copy-update mechanism for supporting real-time applications on shared-memory multiprocessor systems with Linux, Guniguntala, McKenney, Triplett, and Walpole, IBM Systems Journal, April 2008