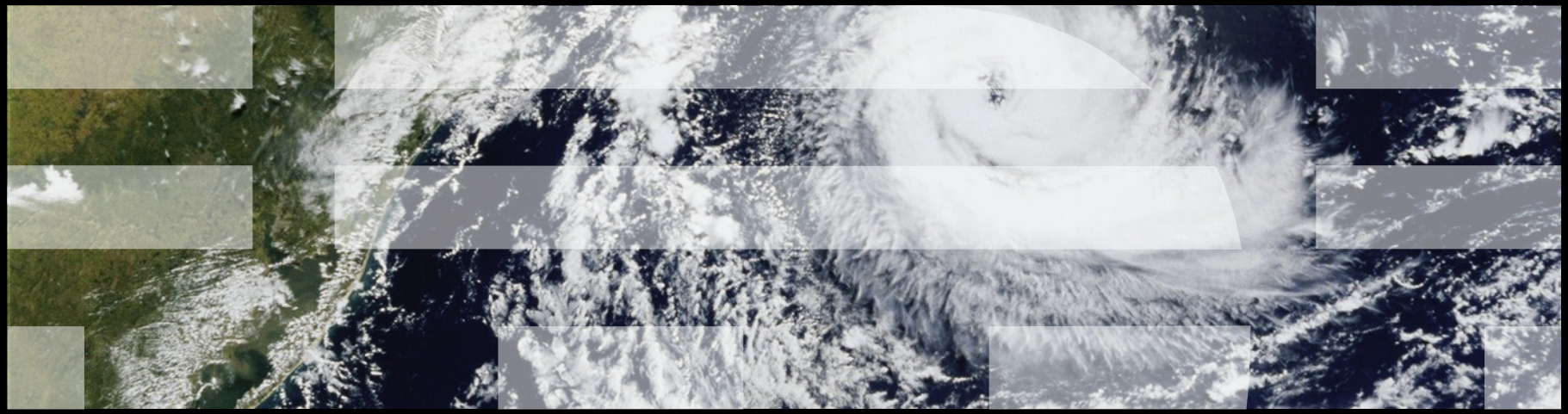
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center
Member, IBM Academy of Technology
Real Time Linux Workshop, Lugano, Switzerland October 29, 2013



Configuring RCU In Your Linux Kernel

An Expert Embedded System Integrator's Guide



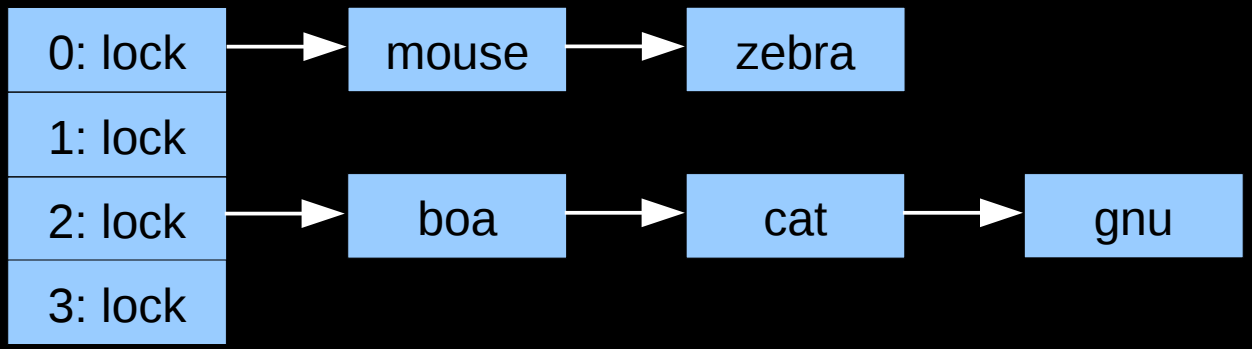
© 2013 IBM Corporation

Brief Introduction to RCU

© 2013 IBM Corporation

Brief Introduction to RCU: Example Application

- Schrödinger wants to construct an in-memory database for the animals in his zoo (example in July CACM article)
  - Births result in insertions, deaths in deletions
  - Queries from those interested in Schrödinger's animals
  - Lots of short-lived animals such as mice: High update rate
  - Great interest in Schrödinger's cat (perhaps queries from mice?)
- Simple approach: chained hash table with per-bucket locking

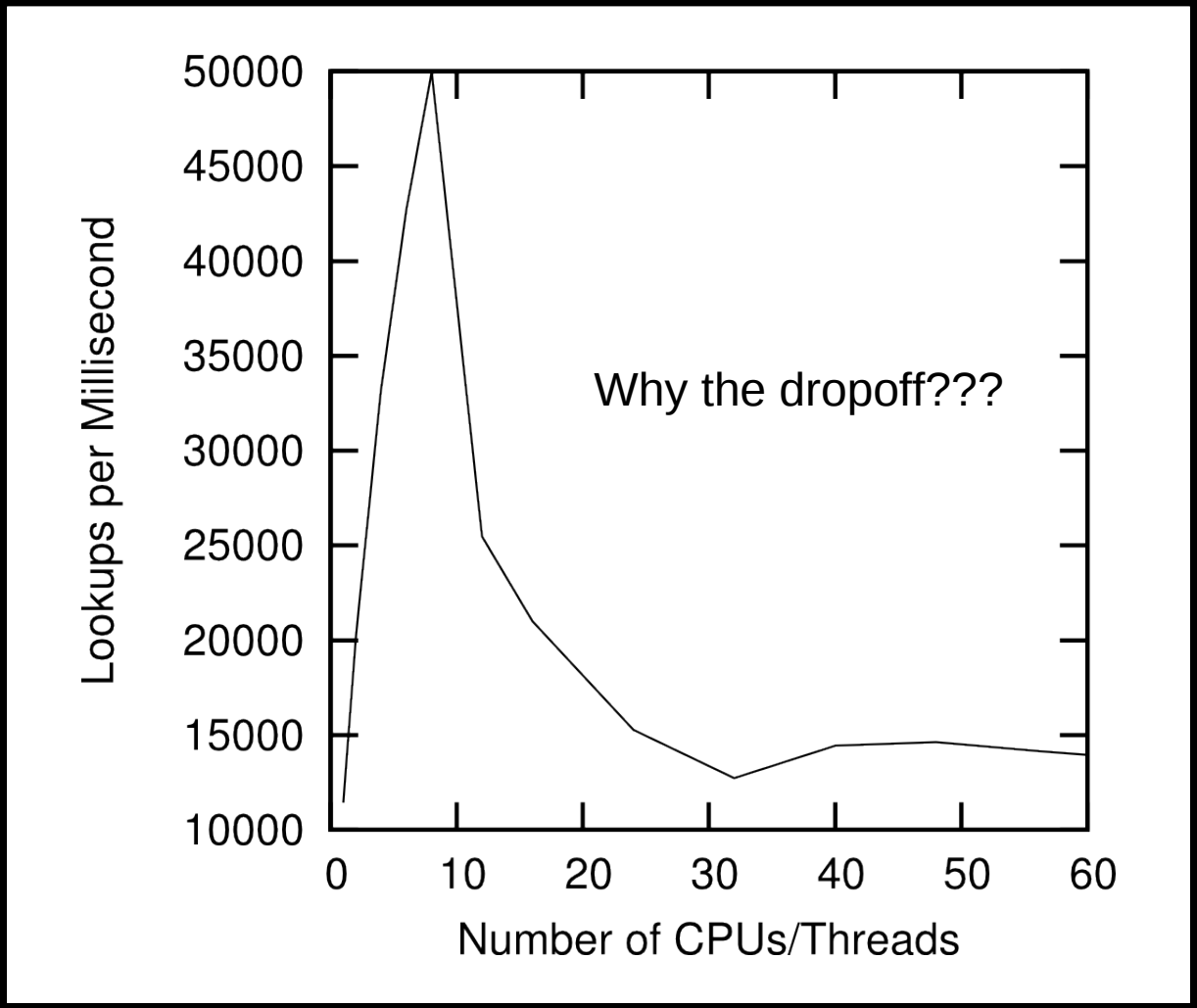


© 2013 IBM Corporation

Perfect Partitioning, Thus Good Performance!

© 2013 IBM Corporation

Read-Only Bucket-Locked Hash Table Performance



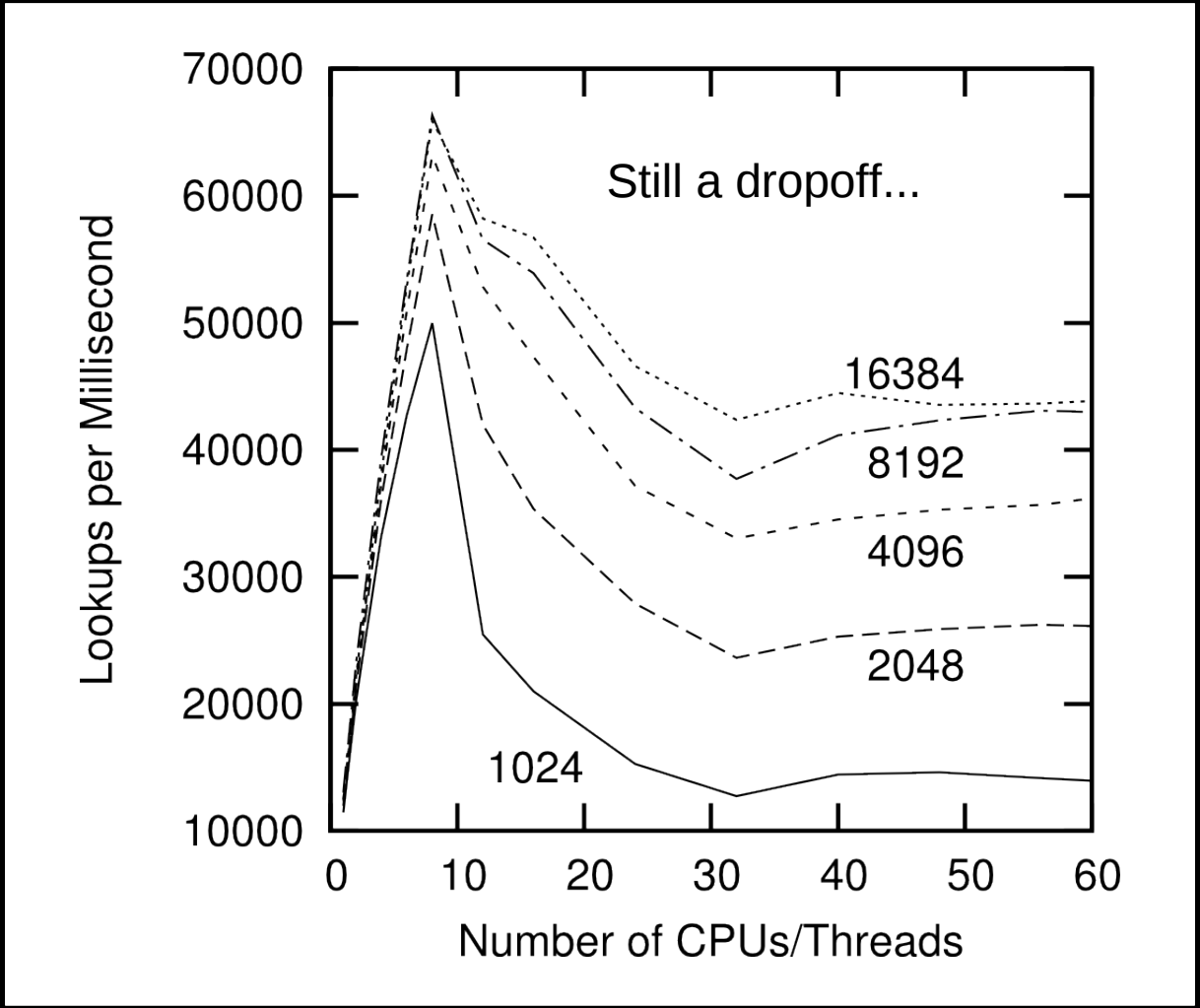
2GHz Intel Xeon Westmere-EX, 1024 hash buckets

© 2013 IBM Corporation

What If We Added More Hash Buckets?

© 2013 IBM Corporation

Varying Number of Hash Buckets



2GHz Intel Xeon Westmere-EX

© 2013 IBM Corporation

But What About NUMA Effects???

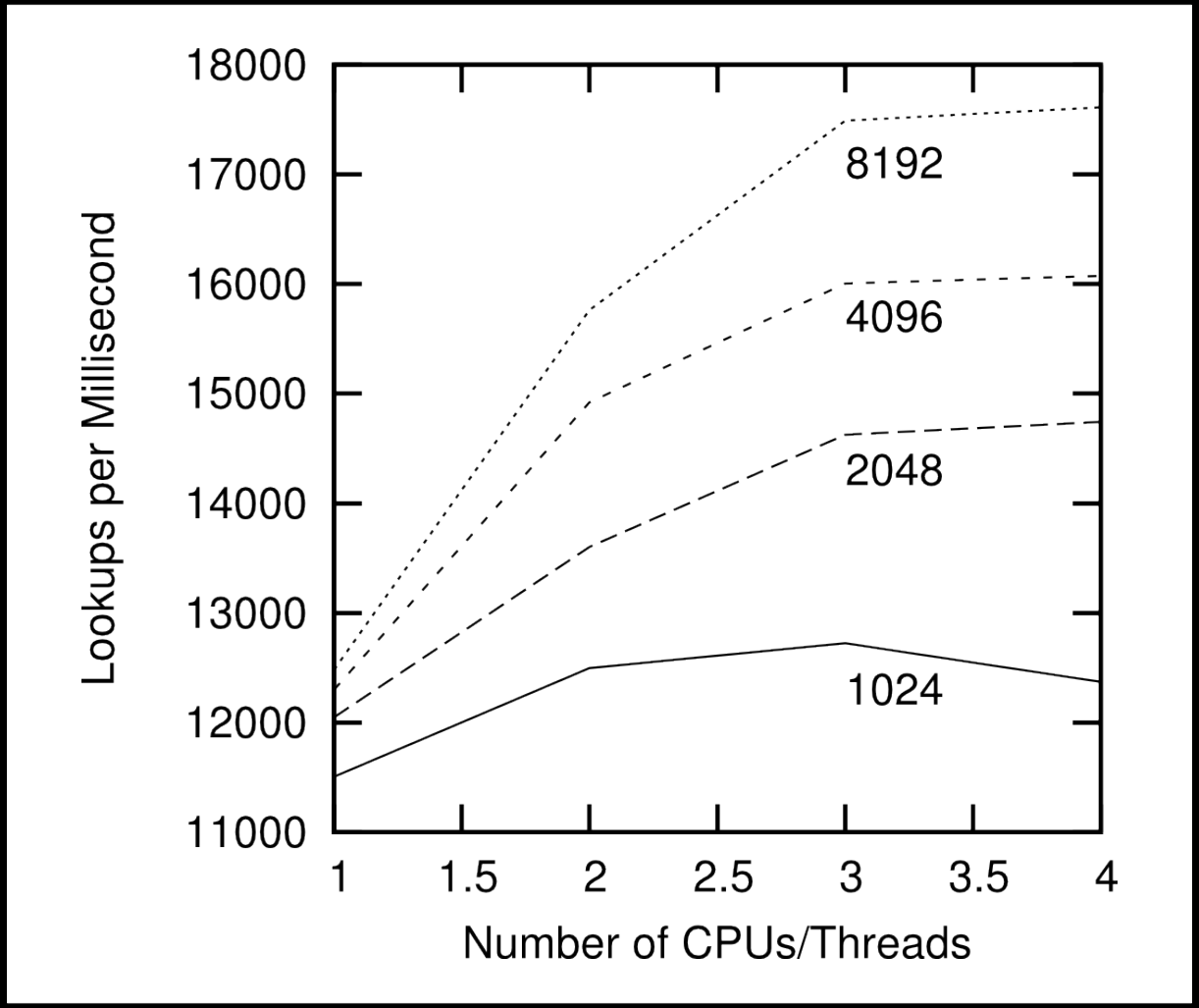
© 2013 IBM Corporation

Check System's NUMA Layout

- /sys/devices/system/cpu/cpu0/cache/index0/shared_cpu_list:
  - 0,32
- /sys/devices/system/cpu/cpu0/cache/index1/shared_cpu_list:
  - 0,32
- /sys/devices/system/cpu/cpu0/cache/index2/shared_cpu_list:
  - 0,32
- /sys/devices/system/cpu/cpu0/cache/index3/shared_cpu_list:
  - **0-7,32-39**
- Two hardware threads per core, eight cores per socket
- Try using only one CPU per socket: CPUs 0, 8, 16, and 24

© 2013 IBM Corporation

Bucket-Locked Hash Performance: 1 CPU/Socket



This is not the sort of scalability that Schrödinger requires!!!

© 2013 IBM Corporation

Why So Slow????

© 2013 IBM Corporation

Performance of Synchronization Mechanisms

Need to be here! (Partitioning/RCU)

16-CPU 2.8GHz Intel X5550 (Nehalem) System

Operation	Cost (ns)	Ratio
Clock period	0.4	1
“Best-case” CAS	12.2	33.8
Best-case lock	25.6	71.2
Single cache miss	12.9	35.8
CAS cache miss	7.0	19.4
Single cache miss **(off-core)**	31.2	86.6
CAS cache miss **(off-core)**	31.2	86.5
Single cache miss **(off-socket)**	92.4	256.7
CAS cache miss **(off-socket)**	95.9	266.4

Heavily optimized reader-writer lock might get here for readers (but too bad about those poor writers...)

Typical synchronization mechanisms do this a lot

© 2013 IBM Corporation

Just Don't Use Heavyweight Instructions!

© 2013 IBM Corporation

Implementing Read-Copy Update (RCU)

- Lightest-weight conceivable read-side primitives

© 2013 IBM Corporation

Implementing Read-Copy Update (RCU)

- Lightest-weight conceivable read-side primitives
  - /* Assume non-preemptible (run-to-block) environment. */
  - #define rcu_read_lock()
  - #define rcu_read_unlock()

© 2013 IBM Corporation

Implementing Read-Copy Update (RCU)

- Lightest-weight conceivable read-side primitives
  - /* Assume non-preemptible (run-to-block) environment. */
  - #define rcu_read_lock()
  - #define rcu_read_unlock()
- Best possible performance, scalability, real-time response, wait-freedom, and energy efficiency

© 2013 IBM Corporation

Implementing Read-Copy Update (RCU)

- Lightest-weight conceivable read-side primitives
  - /* Assume non-preemptible (run-to-block) environment. */
  - #define rcu_read_lock()
  - #define rcu_read_unlock()
- Best possible performance, scalability, real-time response, wait-freedom, and energy efficiency
- How can something that does not affect machine state possibly be used as a synchronization primitive???

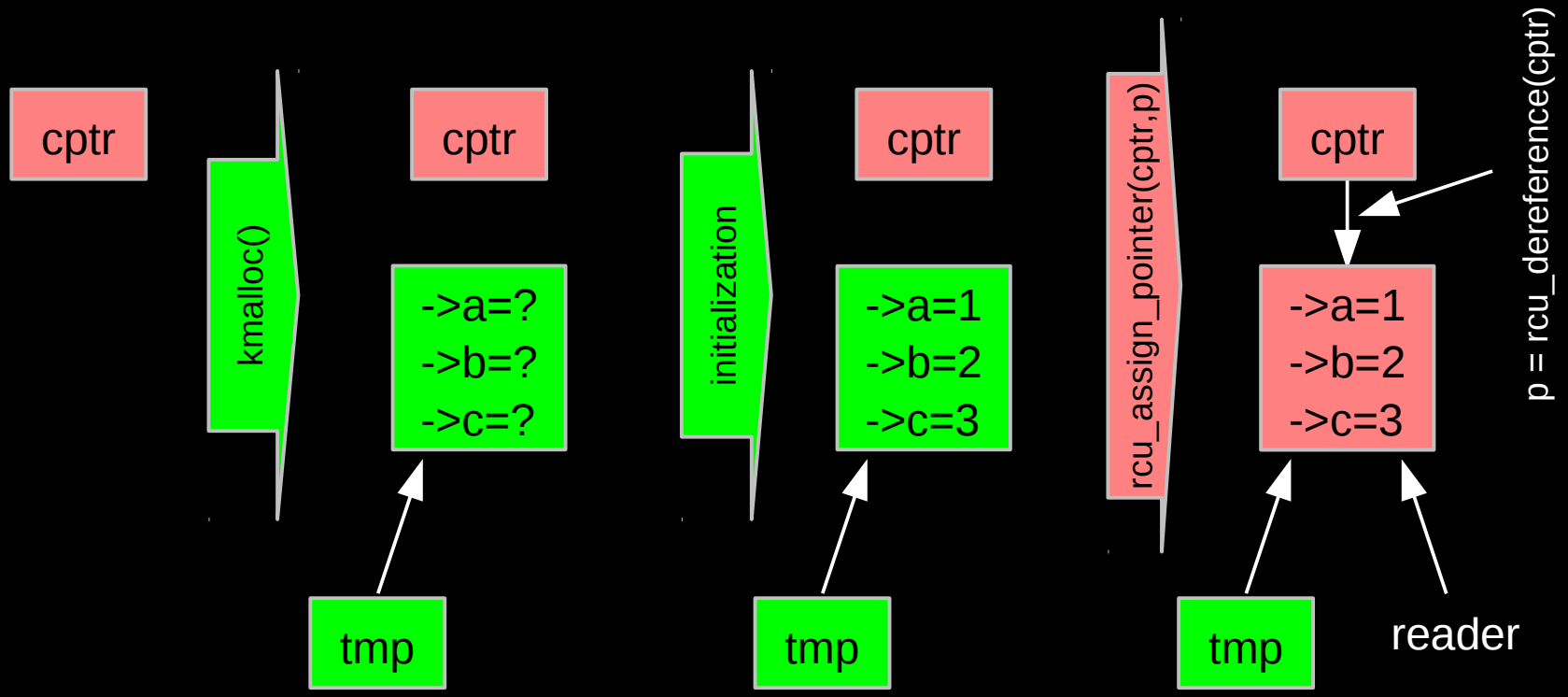
© 2013 IBM Corporation

RCU: Publish New Data, Remove Old Data

© 2013 IBM Corporation

Publication of And Subscription to New Data

Key:
- Dangerous for updates: all readers can access
- Still dangerous for updates: pre-existing readers can access (next slide)
- Safe for updates: inaccessible to all readers

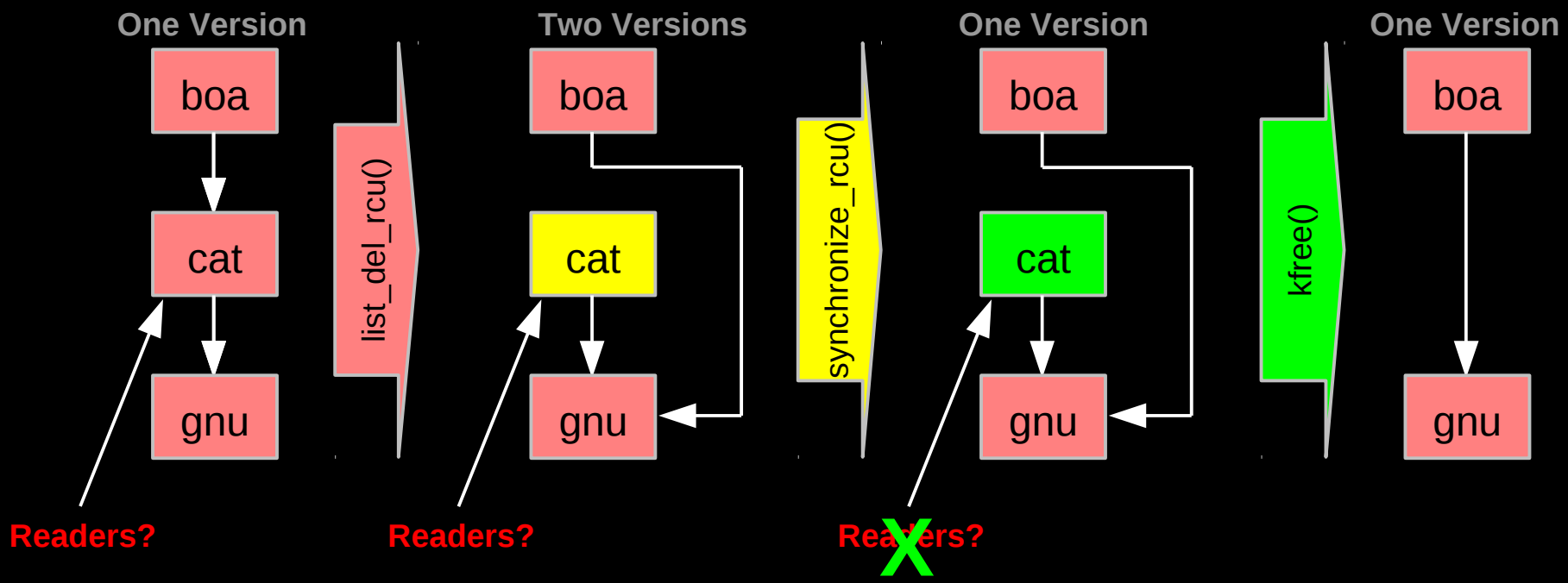


But if all we do is add, we have a big memory leak!!!

© 2013 IBM Corporation

RCU Removal From Linked List

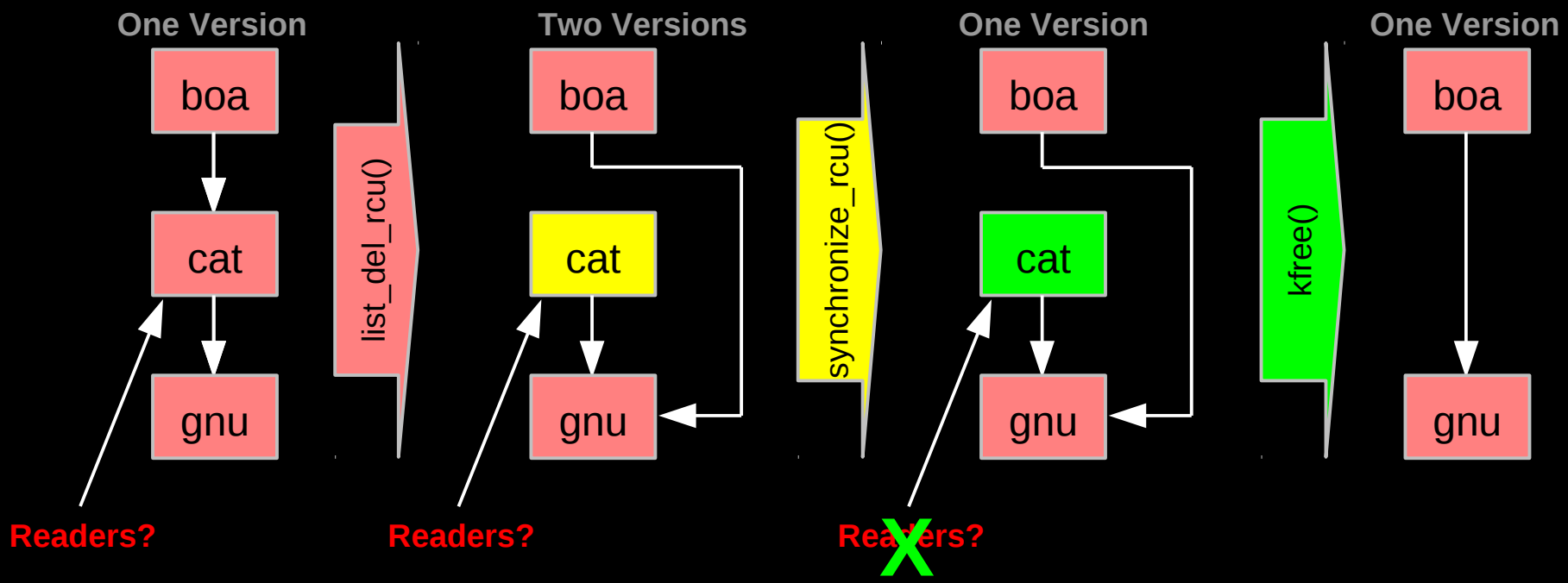
- Combines waiting for readers and multiple versions:
  - Writer removes the cat's element from the list (list_del_rcu())
  - Writer waits for all readers to finish (synchronize_rcu())
  - Writer can then free the cat's element (kfree())



© 2013 IBM Corporation

RCU Removal From Linked List

- Combines waiting for readers and multiple versions:
  - Writer removes the cat's element from the list (list_del_rcu())
  - Writer waits for all readers to finish (synchronize_rcu())
  - Writer can then free the cat's element (kfree())



But how can software deal with two different versions simultaneously???

© 2013 IBM Corporation

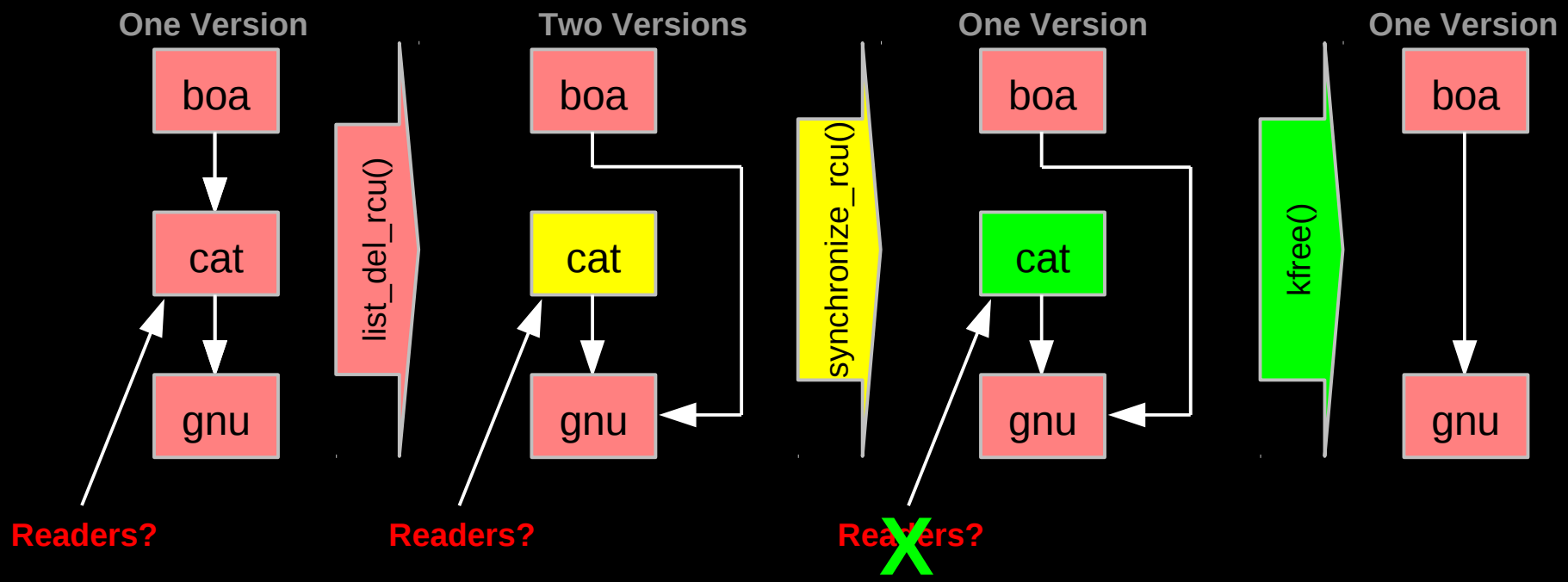
Two Different Versions Simultaneously???



© 2013 IBM Corporation

RCU Removal From Linked List

- Combines waiting for readers and multiple versions:
  - Writer removes the cat's element from the list (list_del_rcu())
  - Writer waits for all readers to finish (synchronize_rcu())
  - Writer can then free the cat's element (kfree())

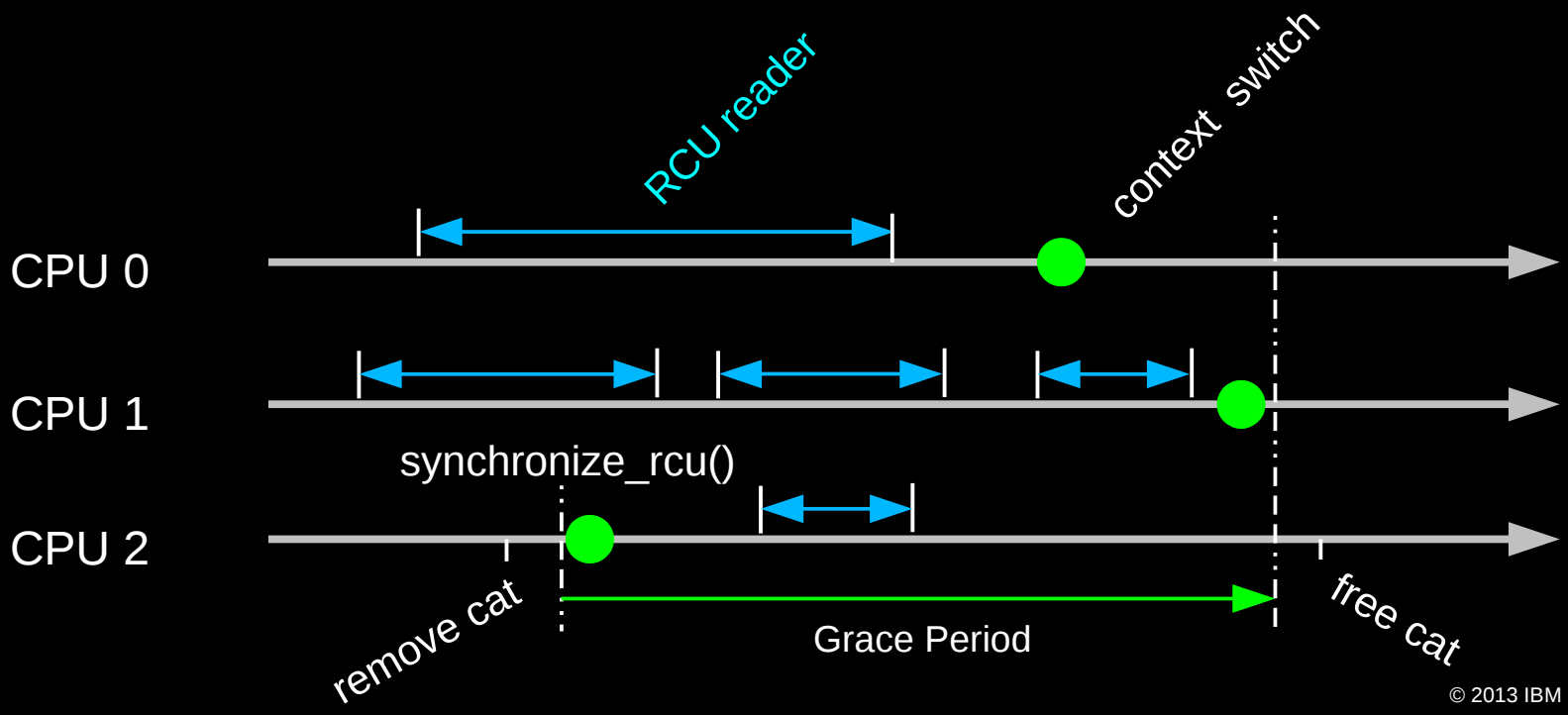


But if readers leave no trace in memory, how can we possibly tell when they are done???

© 2013 IBM Corporation

Waiting for Pre-Existing Readers: QSBR

- Non-preemptive environment (CONFIG_PREEMPT=n)
  - RCU readers are not permitted to block
  - Same rule as for tasks holding spinlocks
- CPU context switch means all that CPU's readers are done
- *Grace period* ends after all CPUs execute a context switch



© 2013 IBM Corporation

Synchronizing Without Machine-State Change

- But rcu_read_lock() does not need to change machine state
  - Instead, it acts on the developer, who must avoid blocking within RCU read-side critical sections
  - Or, more generally, avoid quiescent states within RCU read-side critical sections
- RCU is therefore synchronization via social engineering
- As do all other synchronization mechanisms:
  - "Avoid data races"
  - "Protect specified variables with the corresponding lock"
  - "Access shared variables only within transactions"

© 2013 IBM Corporation

RCU Usage: Readers

- Pointer to RCU-protected object guaranteed to exist throughout RCU read-side critical section

```
rcu_read_lock(); /* Start critical section. */
p = rcu_dereference(cptr);
/* *p guaranteed to exist. */
do_something_with(p);
rcu_read_unlock(); /* End critical section. */
/* *p might be freed!!! */
```

- The rcu_read_lock(), rcu_dereference() and rcu_read_unlock() primitives are very light weight
- However, updaters must take care...

© 2013 IBM Corporation

RCU Usage: Updaters

- Updaters must wait for an *RCU grace period* to elapse between making something inaccessible to readers and freeing it

```
spin_lock(&updater_lock);
q = cptr;
rcu_assign_pointer(cptr, new_p);
spin_unlock(&updater_lock);
synchronize_rcu(); /* Wait for grace period. */
kfree(q);
```

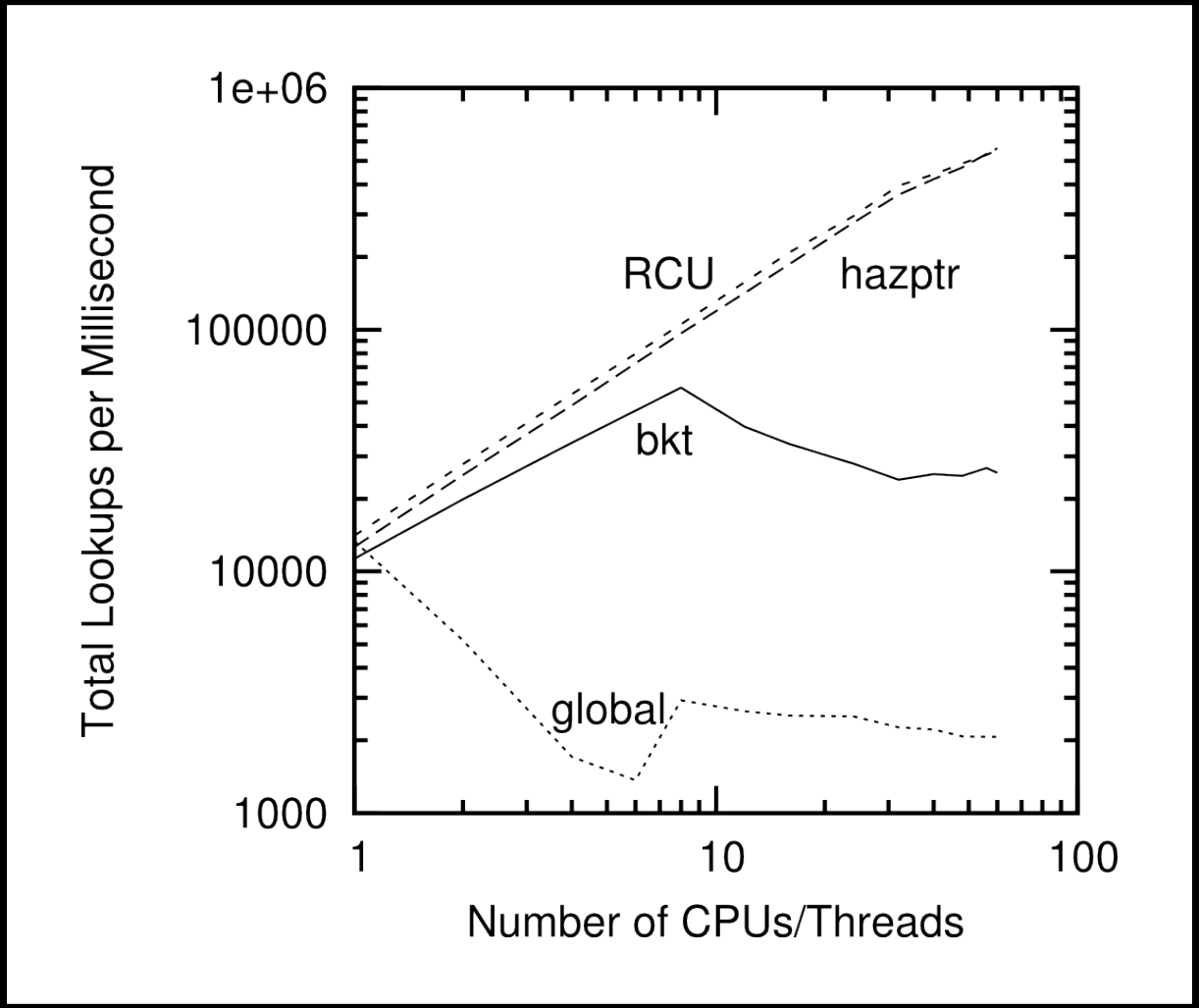
- RCU grace period waits for all pre-exiting readers to complete their RCU read-side critical sections

© 2013 IBM Corporation

RCU Performance

© 2013 IBM Corporation

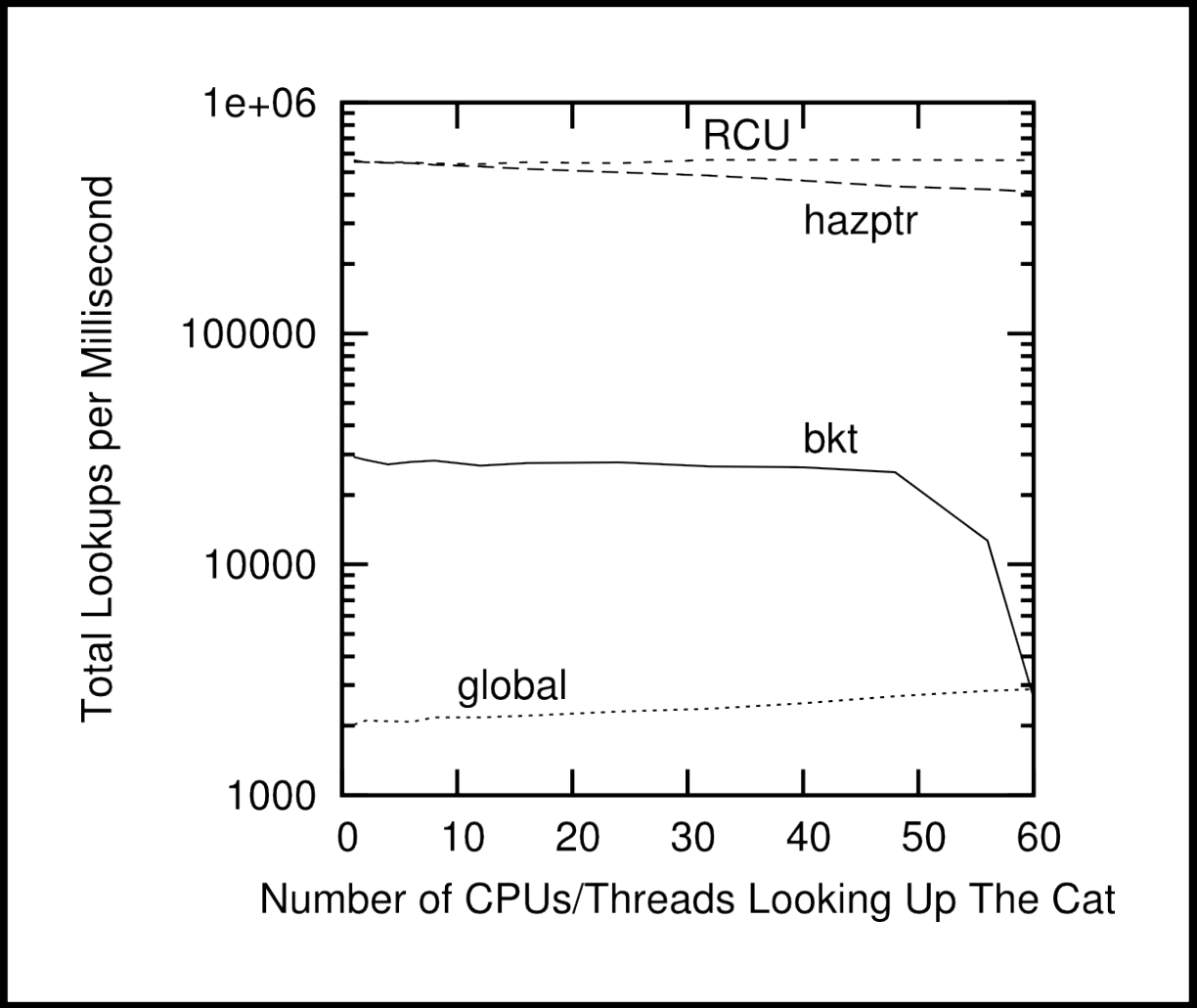
Schrödinger's Zoo: Read-Only



RCU and hazard pointers scale quite well!!!

© 2013 IBM Corporation

Schrödinger's Zoo: Read-Only Cat-Heavy Workload



Per-bucket locking doesn't help if all accesses are to one bucket!

© 2013 IBM Corporation

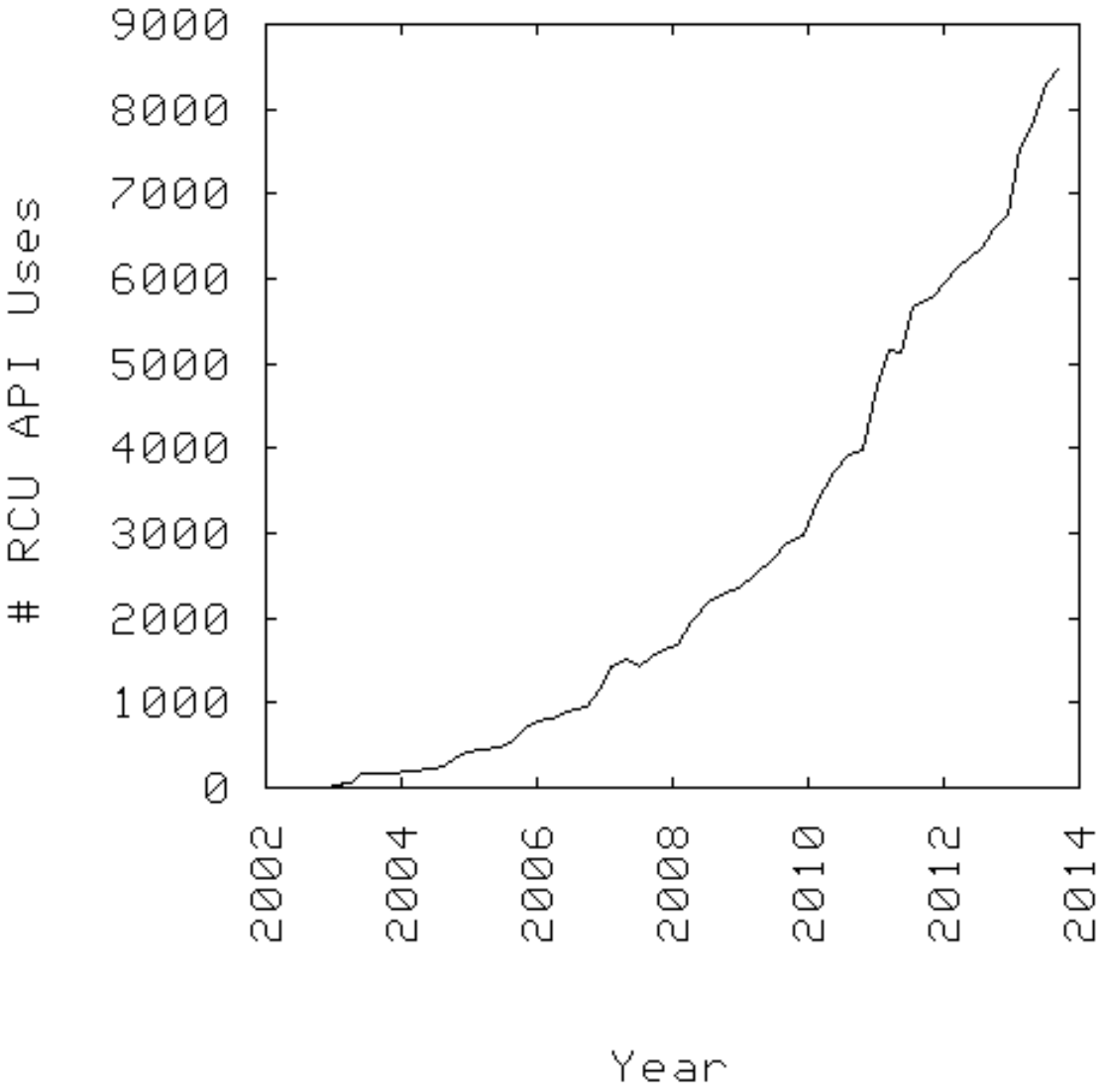
RCU Area of Applicability



Schrodinger's zoo is in blue: Can't tell exactly when an animal is born or dies anyway! Plus, no lock you can hold will prevent an animal's death...

© 2013 IBM Corporation

RCU Applicability to the Linux Kernel



© 2013 IBM Corporation

Configuring RCU

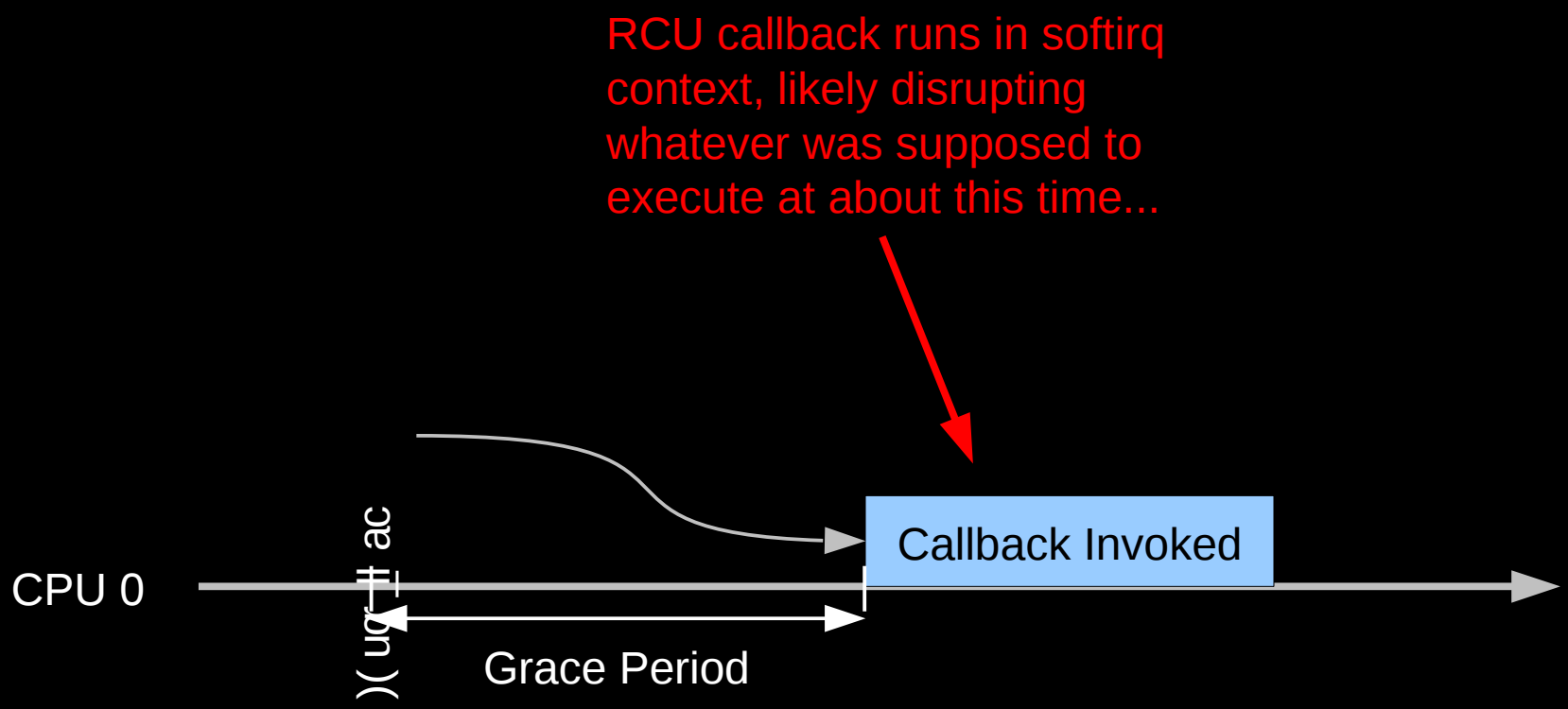
© 2013 IBM Corporation

Configuring RCU: Kconfig

- PREEMPT=y implies TREE_PREEMPT_RCU=y
  - Needed to avoid RCU readers degrading scheduling latency
  - TINY_PREEMPT_RCU is gone as of v3.11.
- NO_HZ_FULL=y implies RCU_USER_QS
  - More on this from Frederic Weisbecker
- RCU_NOCB_CPU=y and RCU_NOCB_CPU_ALL=y to offload callback invocation
  - Can pin resulting kthreads to “housekeeping” CPUs
  - But run at SCHED_OTHER, so less disruptive regardless of pinning
  - See next slide...

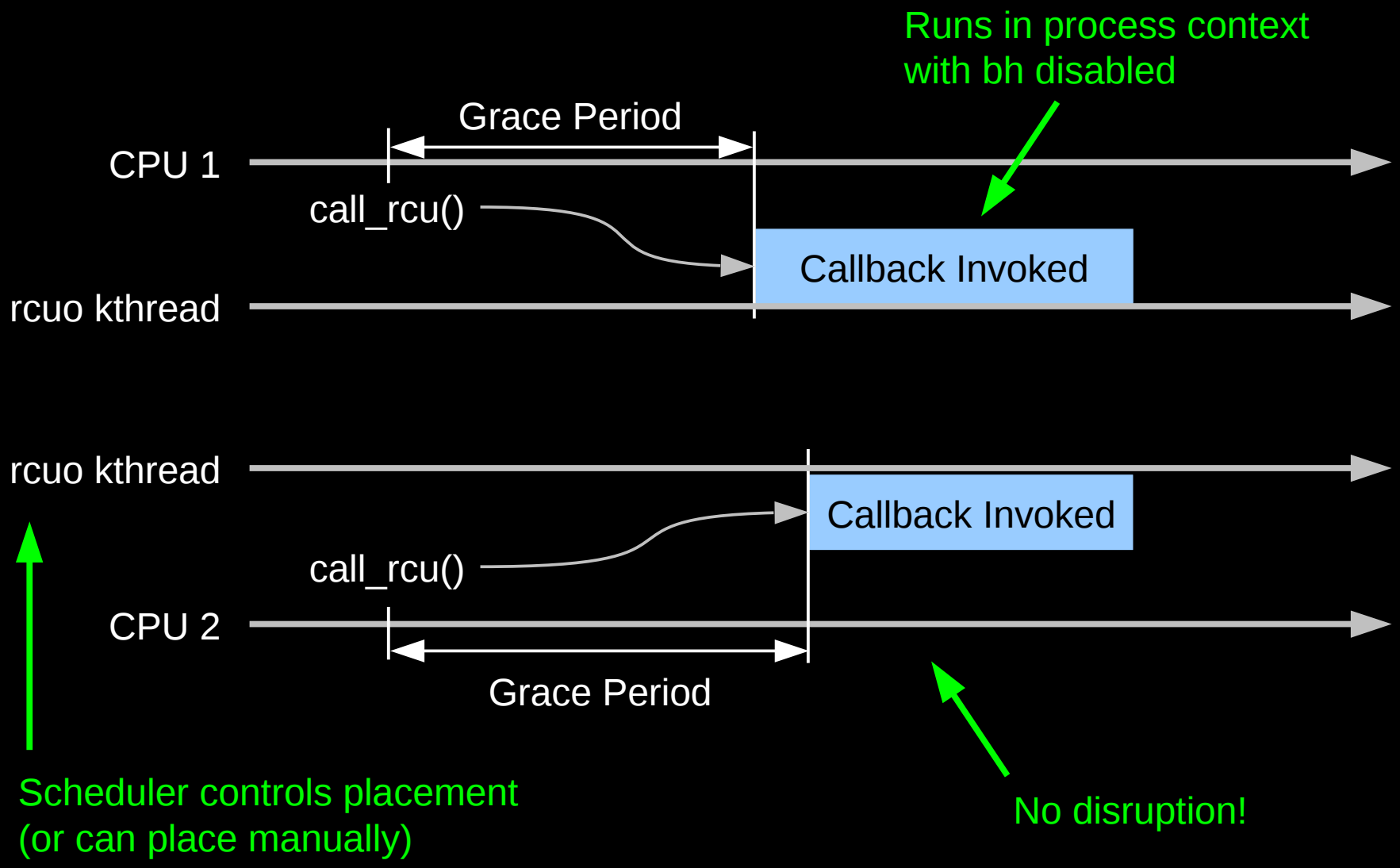
© 2013 IBM Corporation

The Problem With Non-Offloaded RCU Callbacks



© 2013 IBM Corporation

Offload RCU Callbacks (Inspired by Houston/Korty)



© 2013 IBM Corporation

Offloadable RCU Callbacks: Limitations and Futures

- Must reboot to reconfigure no-CBs CPUs
  - rcu_nocb_poll kernel command-line parameter gives list of no-CB CPUs
  - Races between reconfiguring, registering callbacks, rcu_barrier(), grace periods and who knows what all else are far from pretty! (But you can move the kthreads around w/out boot.)
  - Just offload all of them and don't worry about reconfiguring!!!
- No energy-efficiency code: lazy & non-lazy CBs? Non-lazy!
  - This can be fixed if it becomes a problem – but show me the problem first!
- No-CBs CPUs' kthreads not subject to priority boosting
  - Rely on configurations restrictions: priority boosting not heavily used anyway
- Setting all no-CBs CPUs' kthreads to RT prio w/out pinning them: bad!
  - So either leave them as SCHED_OTHER or pin them
  - Or run only on small systems where global scheduling works OK
- Thus, I do not expect no-CBs path to completely replace current CB path
  - At least not without a lot of experience and performance evaluation
  - Maybe in a few years if experience indicates that this simplification is safe for all workloads

© 2013 IBM Corporation

Configuring RCU: Kconfig Debugging Options (1/3)

- **DEBUG_OBJECTS_RCU_HEAD**(=n): Check for duplicate call_rcu() invocations
- **PROVE_RCU**(=n): Check for accessing RCU-protected pointers outside of RCU read-side critical sections
- **SPARSE_RCU_POINTER**(=n): When building with sparse, check usage of RCU-protected pointers.
- PROVE_RCU_REPEATEDLY(=n): Don't stop checking after the first PROVE_RCU splat
- PROVE_RCU_DELAY(=n): Insert delays to provoke allegedly-fixed RCU bugs

Please use the options in **green** when adding new kernel code that uses RCU.

© 2013 IBM Corporation

Configuring RCU: Kconfig Debugging Options (2/3)

- RCU_TORTURE_TEST=n: RCU torture tests, which may be built-in or built as module.
- RCU_TORTURE_TEST_RUNNABLE=n: If rcutorture is built in, start running at boot time.
- RCU_CPU_STALL_TIMEOUT=21: Number of seconds to wait before complaining about too-long RCU grace period.
- RCU_CPU_STALL_VERBOSE=y: Print per-task information for any tasks blocking a preemptible-RCU grace period.
- RCU_CPU_STALL_INFO=n: Print additional information for each CPU stalling the current grace period.

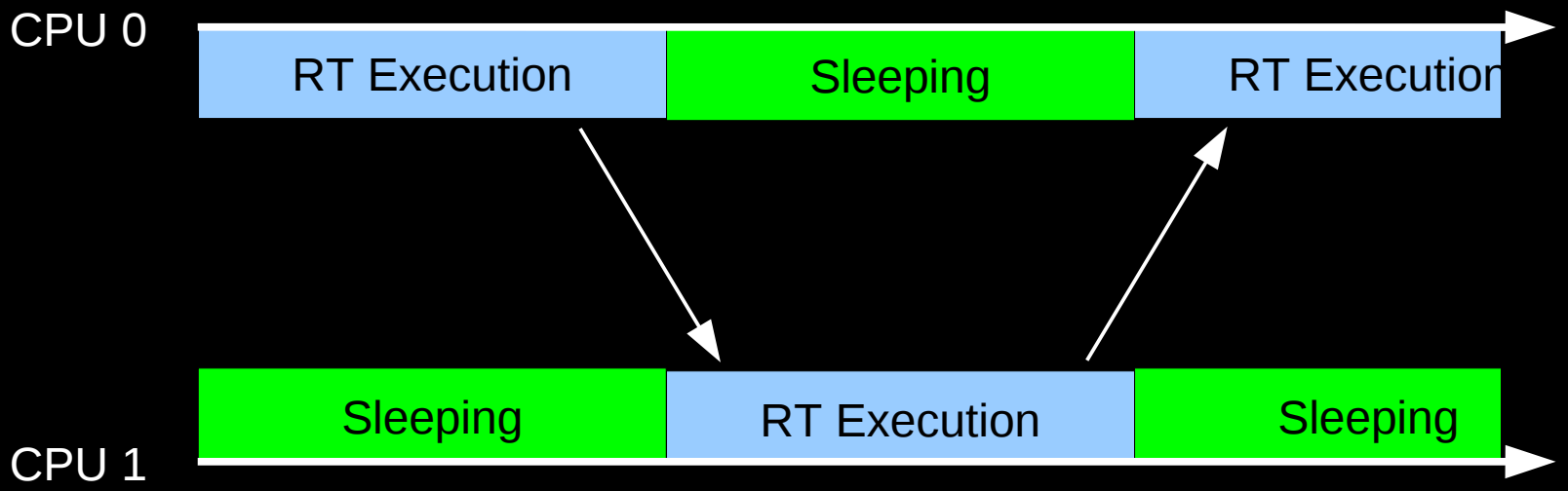
© 2013 IBM Corporation

Configuring RCU: Kconfig Debugging Options (3/3)

- TREE_RCU_TRACE=n to enable full event tracing
  - Also enables debugfs tracing
  - Usually used for debugging RCU itself
- RCU_BOOST=n in case of accidental infinite loops
  - Prevents preempted readers from indefinitely blocking grace periods
  - But consider instead the approach on the next slide
- RCU_BOOST_PRIO=1: Set higher than infinite loop
  - Otherwise, infinite loop will continue to block grace period
  - Defaults of 1 only helps with SCHED_OTHER infinite loops
- RCU_BOOST_DELAY=500: Wait this long before boosting
  - Maximum delay is 3000 (3 seconds)

© 2013 IBM Corporation

Alternative to CPU-Bound Execution



Use a pair of variables to mediate transitions:
1. am_awake: End of sleep
2. ready_sleep: Hand off to newly awakened thread.

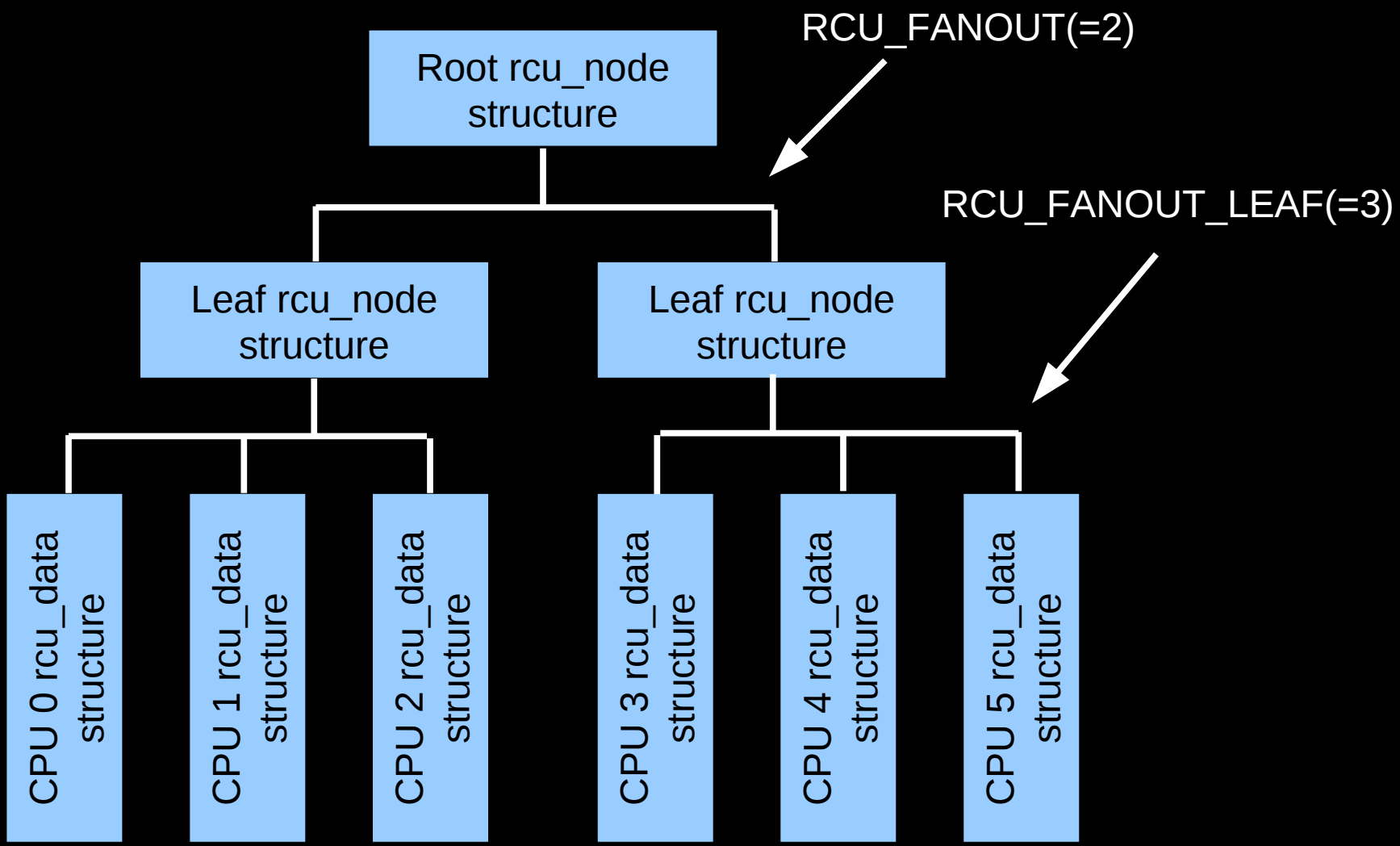
© 2013 IBM Corporation

Configuring RCU: Special-Purpose Kconfig Options

- RCU_FAST_NO_HZ=y: Reduce scheduling-clock interrupts while idle CPU has RCU callbacks. Increases from-idle scheduling latency, improves battery lifetime.
- RCU_FANOUT: Control fanout of non-leaf nodes of rcu_node combining tree. Default of 32 or 64 should work fine.
- RCU_FANOUT_LEAF: Control fanout of leaf nodes of rcu_node combining tree. Default of 16 should work fine.
  - Exception: Very large systems may need to set to 64, but only for kernels preceding v3.7.
- RCU_FANOUT_EXACT=n: Disabling rebalancing of rcu_node combining tree. Don't use pre-v3.14 if RCU_FANOUT and RCU_FANOUT_LEAF relatively prime...

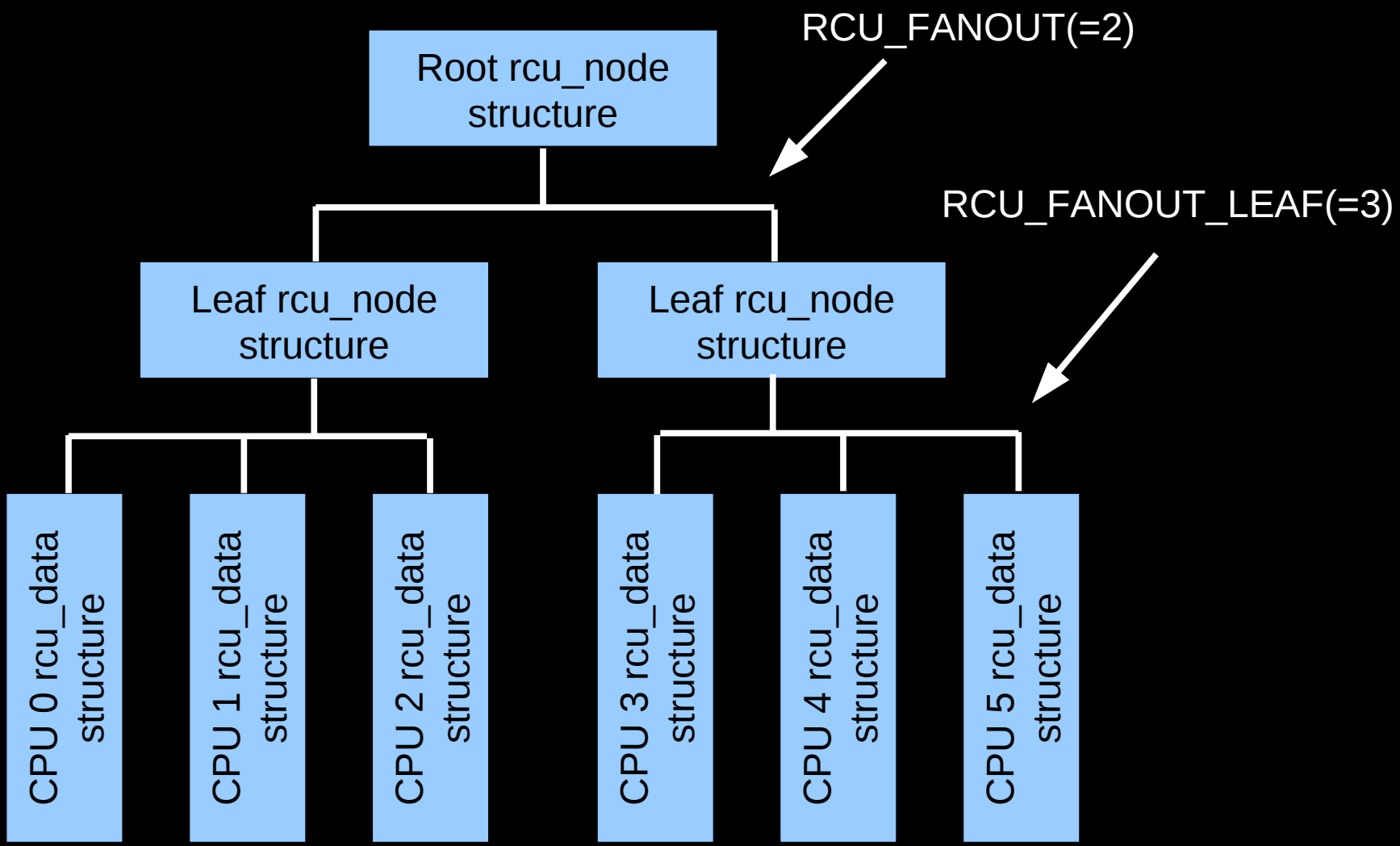
© 2013 IBM Corporation

Configuring RCU Combining-Tree Fanout



© 2013 IBM Corporation

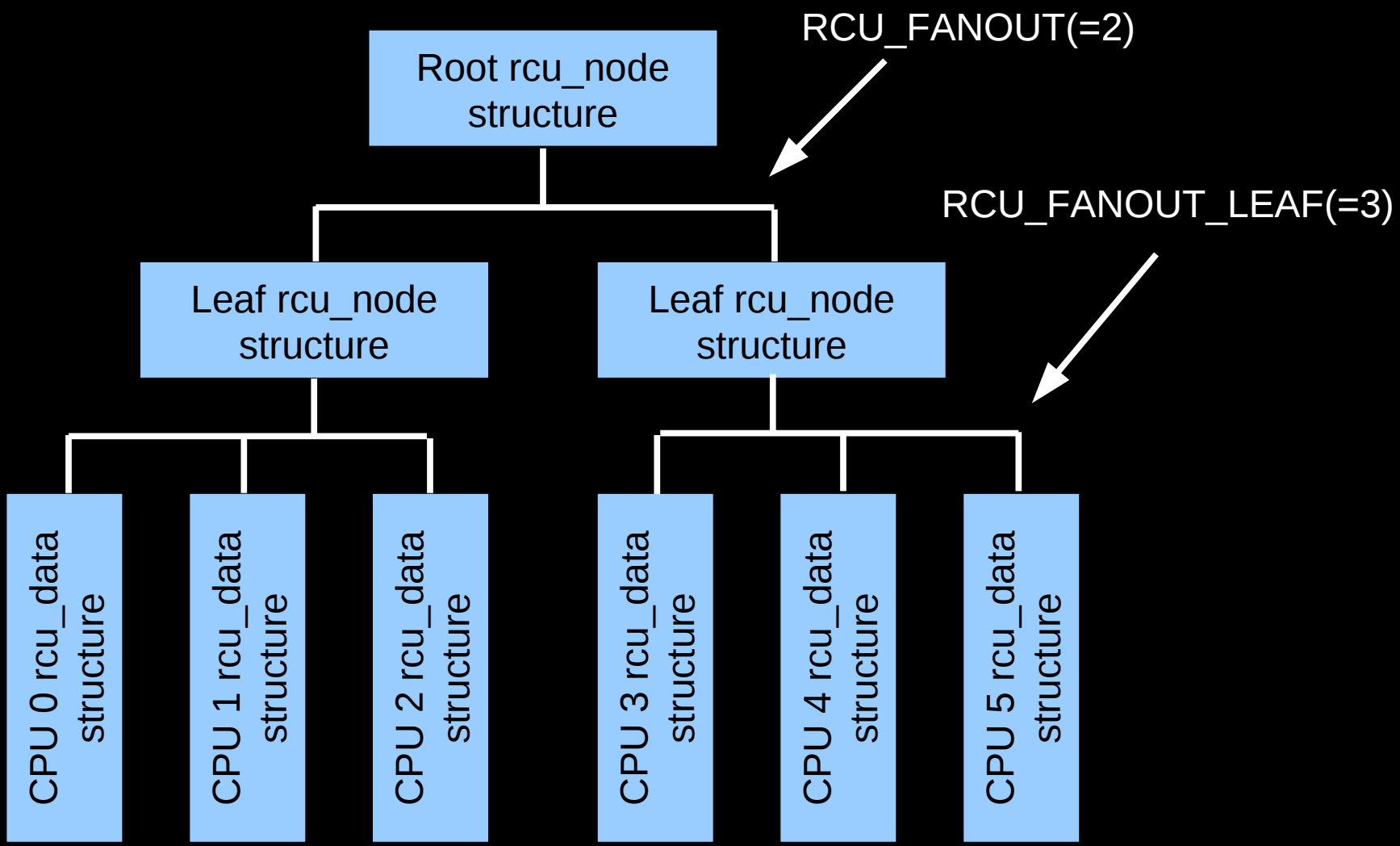
Configuring RCU Combining-Tree Fanout



This choice is a really bad idea unless you have commit #e9c0e577...

© 2013 IBM Corporation

Configuring RCU Combining-Tree Fanout



Or RCU_FANOUT_EXACT=n. Which fortunately is the default.

© 2013 IBM Corporation

RCU Kernel-Boot Parameters (1/3)

- rcu_nocbs=: List of CPUs to be NOCBs CPUs, but has no effect if RCU_NOCB_CPU_ALL.
- rcu_nocb_poll(=n): Avoid wakeup overhead in call_rcu() by making the rcu0 kthreads poll their CPU's callback lists.
- rcutree.blimit(=10): Maximum number of callbacks to invoke in a single batch from softirq.
- rcutree.qhimark(=10,000): Above this number of callbacks on a given CPU, ignore rcutree.blimit.
- rcutree.qlowmark(=100): Below this number of callbacks on a given CPU, revert to paying attention to rcutree.blimit.
- rcutree.rcu_fanout_leaf: Increase fanout vs. that specified by the RCU_FANOUT_LEAF Kconfig parameter.

© 2013 IBM Corporation

RCU Kernel-Boot Parameters (2/3)

- rcutree.jiffies_till_first_fqs: Number of jiffies in an RCU grace period until the first check for dyntick-idle CPUs. Default is computed from HZ and nr_cpu_ids.
- rcutree.jiffies_till_next_fqs: Number of jiffies in an RCU grace period between checks for dyntick-idle CPUs. Default is computed from HZ and nr_cpu_ids.
- rcutree.rcu_idle_gp_delay(=4): Number of jiffies till timed recheck for RCU_FAST_NO_HZ.
- rcutree.rcu_idle_lazy_gp_delay(=6*HZ): Number of jiffies until timed recheck for RCU_FAST_NO_HZ if only lazy callbacks.

© 2013 IBM Corporation

RCU Kernel-Boot Parameters (3/3)

- rcupdate.rcu_expedited(=n): Use expedited grace period even when non-expedited primitives are invoked. Does not affect call_rcu() and friends.
- rcupdate.rcu_cpu_stall_suppress(=n): Suppress reporting of any subsequent RCU CPU stall warning messages.
- rcupdate.rcu_cpu_stall_timeout(=21): Set timeout in seconds for RCU CPU stall warnings.
- Plus lots of rcutorture settings. See the rcutorture source code for these.

© 2013 IBM Corporation

Summary

- Synchronization overhead is a big issue for parallel programs
- Straightforward design techniques can avoid this overhead
  - Partition the problem: “Many instances of something good!”
  - Avoid expensive operations
  - Avoid mutual exclusion
- RCU is part of the solution
  - Excellent for read-mostly data where staleness and inconsistency OK
  - Good for read-mostly data where consistency is required
  - Can be OK for read-write data where consistency is required
  - Might not be best for update-mostly consistency-required data
  - Used heavily in the Linux kernel
- Careful RCU configuration helpful for real-time systems

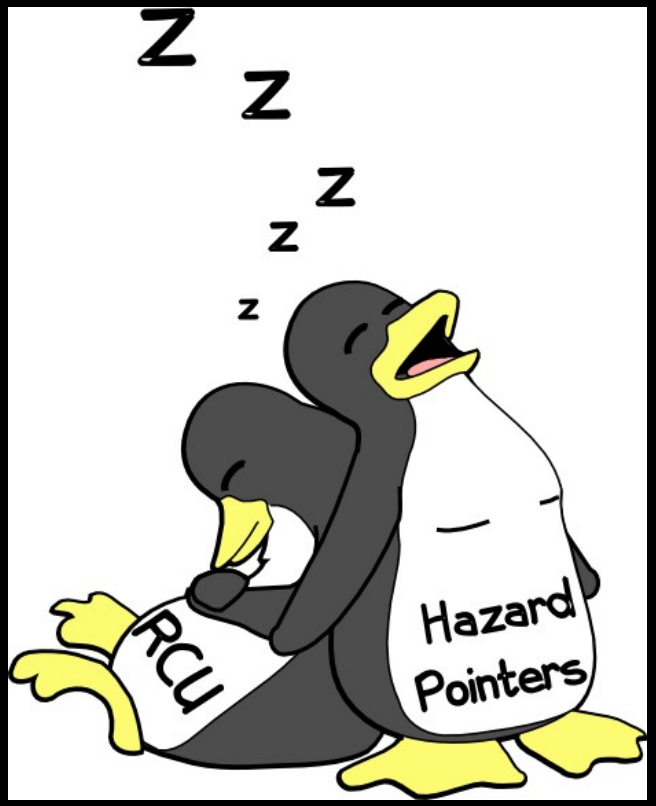
© 2013 IBM Corporation

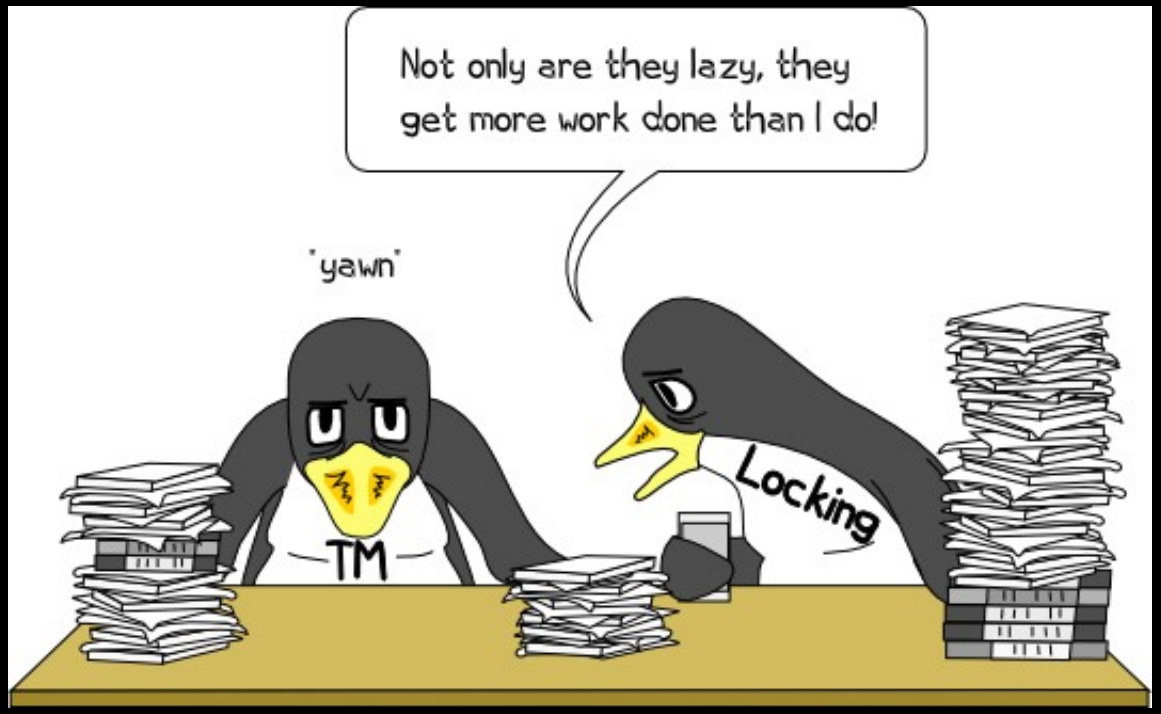
Configuration Summary

- RCU_NOCB_CPU=y and RCU_NOCB_CPU_ALL=y to offload callback invocation.
- After changing code that uses RCU, please test with:
  - DEBUG_OBJECTS_RCU_HEAD(=n): Check for duplicate call_rcu() invocations
  - PROVE_RCU(=n): Check for accessing RCU-protected pointers outside of RCU read-side critical sections
  - SPARSE_RCU_POINTER(=n): When building with sparse, check usage of RCU-protected pointers.

© 2013 IBM Corporation

Graphical Summary





© 2013 IBM Corporation

To Probe Further:

- https://queue.acm.org/detail.cfm?id=2488549
  - "Structured Deferral: Synchronization via Procrastination" (also in July 2013 CACM)
- http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.159 and http://www.computer.org/cms/Computer.org/dl/trans/td/2012/02/extras/ttd2012020375s.pdf
  - "User-Level Implementations of Read-Copy Update"
- git://lttng.org/userspace-rcu.git (User-space RCU git tree)
- http://people.csail.mit.edu/nickolai/papers/clements-bonsai.pdf
  - Applying RCU and weighted-balance tree to Linux mmap_sem.
- http://www.usenix.org/event/atc11/tech/final_files/Triplett.pdf
  - RCU-protected resizable hash tables, both in kernel and user space
- http://www.usenix.org/event/hotpar11/tech/final_files/Howard.pdf
  - Combining RCU and software transactional memory
- http://wiki.cs.pdx.edu/rp/: Relativistic programming, a generalization of RCU
- http://lwn.net/Articles/262464/, http://lwn.net/Articles/263130/, http://lwn.net/Articles/264090/
  - "What is RCU?" Series
- http://www.rdrop.com/users/paulmck/RCU/RCUdissertation.2004.07.14e1.pdf
  - RCU motivation, implementations, usage patterns, performance (micro+sys)
- http://www.livejournal.com/users/james_morris/2153.html
  - System-level performance for SELinux workload: >500x improvement
- http://www.rdrop.com/users/paulmck/RCU/hart_ipdps06.pdf
  - Comparison of RCU and NBS (later appeared in JPDC)
- http://doi.acm.org/10.1145/1400097.1400099
  - History of RCU in Linux (Linux changed RCU more than vice versa)
- http://read.seas.harvard.edu/cs261/2011/rcu.html
  - Harvard University class notes on RCU (Courtesy of Eddie Koher)
- http://www.rdrop.com/users/paulmck/RCU/ (More RCU information)

© 2013 IBM Corporation

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.

© 2013 IBM Corporation

Questions?



© 2013 IBM Corporation