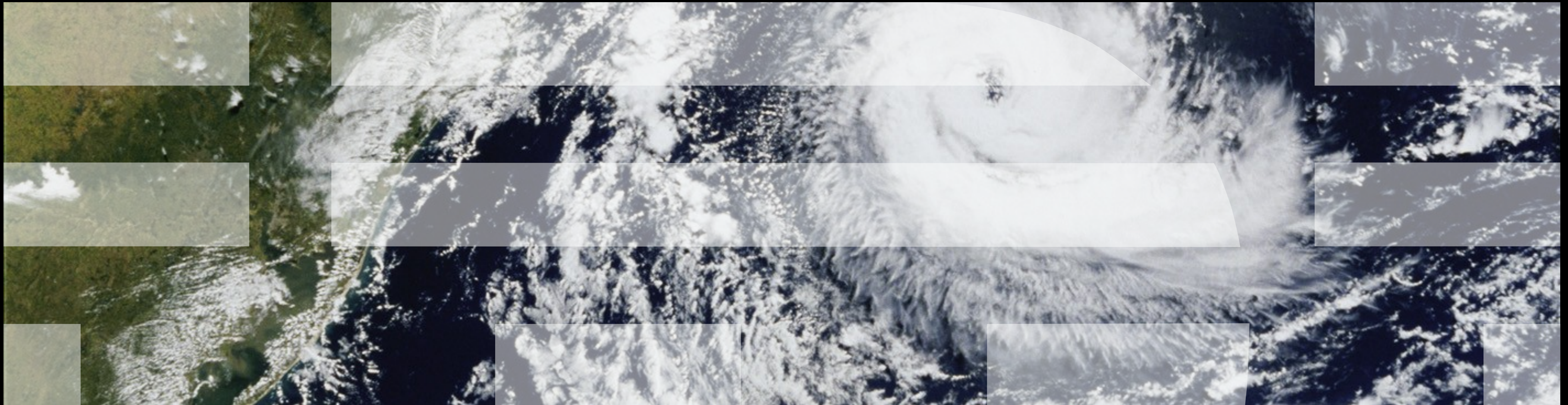
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center (Linaro)

31 August 2012

Getting RCU Further Out Of The Way



2012 Linux Plumbers Conference, Real Time Microconference

© 2009 IBM Corporation

RCU Was Once A Major Real-Time Obstacle

- rcu_read_lock() disabled preemption
- RCU processing happened every jiffy, whether needed or not
- Callback invocation could tie up a CPU forever

© 2009 IBM Corporation

RCU Was Once A Major Real-Time Obstacle

- rcu_read_lock() disabled preemption
- RCU processing happened every jiffy, whether needed or not
- Callback invocation could tie up a CPU forever
  - OK, there is an upper bound: the number of RCU-protected blocks of memory on the system

© 2009 IBM Corporation

RCU Was Once A Major Real-Time Obstacle

- rcu_read_lock() disabled preemption
- RCU processing happened every jiffy, whether needed or not
- Callback invocation could tie up a CPU forever
  - OK, there is an upper bound: the number of RCU-protected blocks of memory on the system
- But what the heck is RCU??? http://lwn.net/Articles/262464/

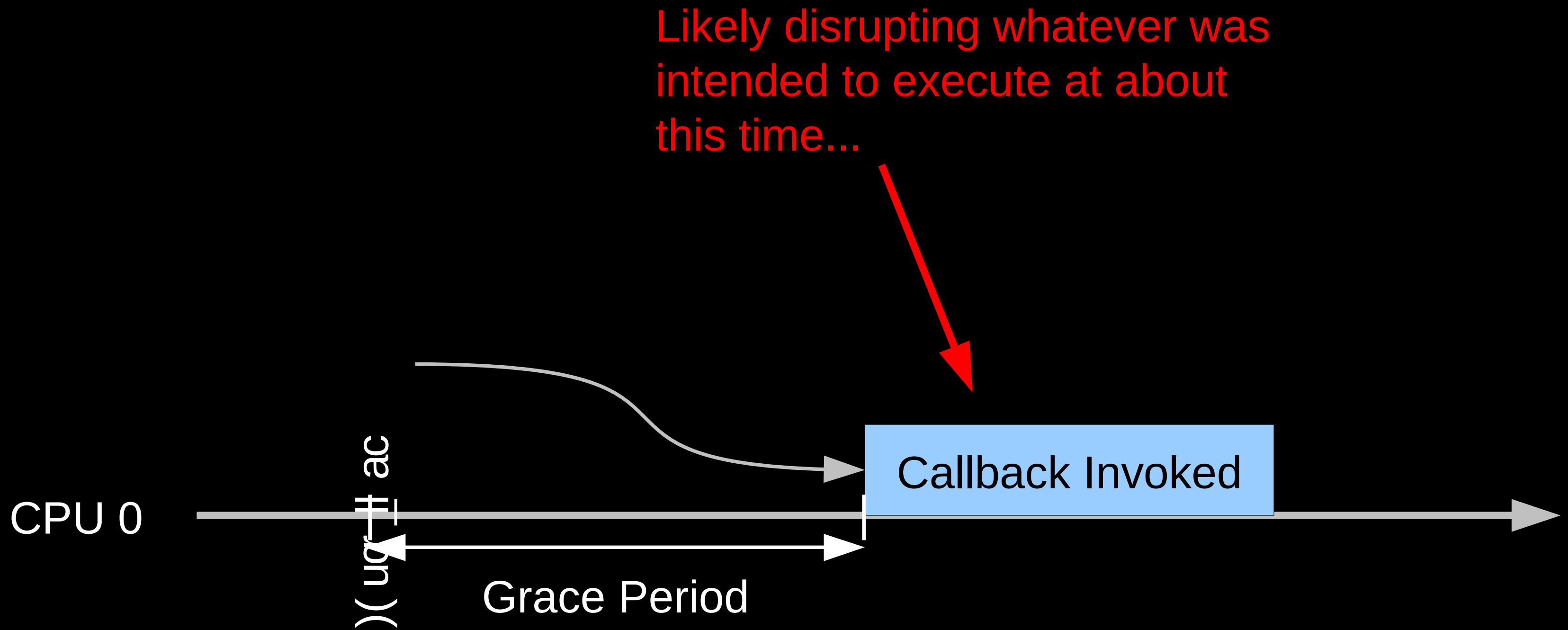
© 2009 IBM Corporation

RCU Was Once A Major Real-Time Obstacle

- rcu_read_lock() disabled preemption
- RCU processing happened every jiffy, whether needed or not
- Callback invocation could tie up a CPU forever
  - OK, there is an upper bound: the number of RCU-protected blocks of memory on the system
- But what the heck is RCU??? http://lwn.net/Articles/262464/
- For the purposes of this presentation, think of RCU as something that defers work, with one work item per callback
  - Each callback has a function pointer and an argument
  - Callbacks are queued on per-CPU lists, invoked after grace period
    - Invocation can result in OS jitter and real-time latency
  - Global list handles callbacks from offlined CPUs: adopted quickly

© 2009 IBM Corporation

The Problem With RCU Callbacks



© 2009 IBM Corporation

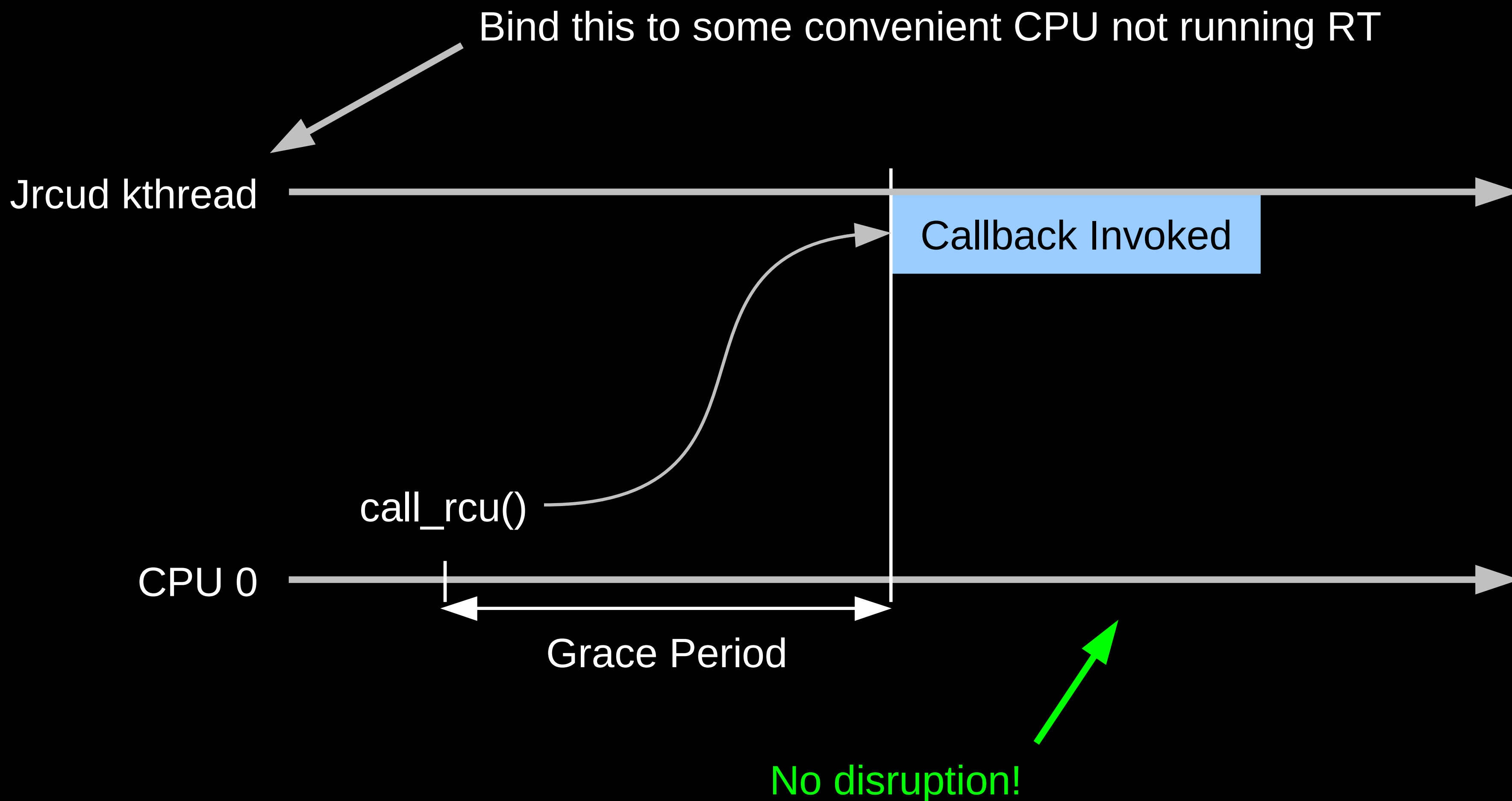
IBM

RCU Has Reformed Considerably

- 2002-onwards: Dyntick-idle RCU
  - Unfortunately, this only helps if the CPU is idle, not good for real-time
  - But Frederic's adaptive-tick work should clear this up
- 2004: RCU callback throttling (Dipankar Sarma)
  - Limits callback processing to bursts of 10 callbacks
- 2004: Jim Houston's RCU implementation
  - Since updated by Joe Korty: JRCU ***(out of tree)***
  - All callback processing happens in kthread: preemptible
    - Eliminates need for driving RCU from scheduling-clock interrupt
    - Allows callback processing to offloaded to some other CPU
  - But has heavyweight read-side primitives and poor scalability
- 2005-2009: Preemptible RCU read-side critical sections

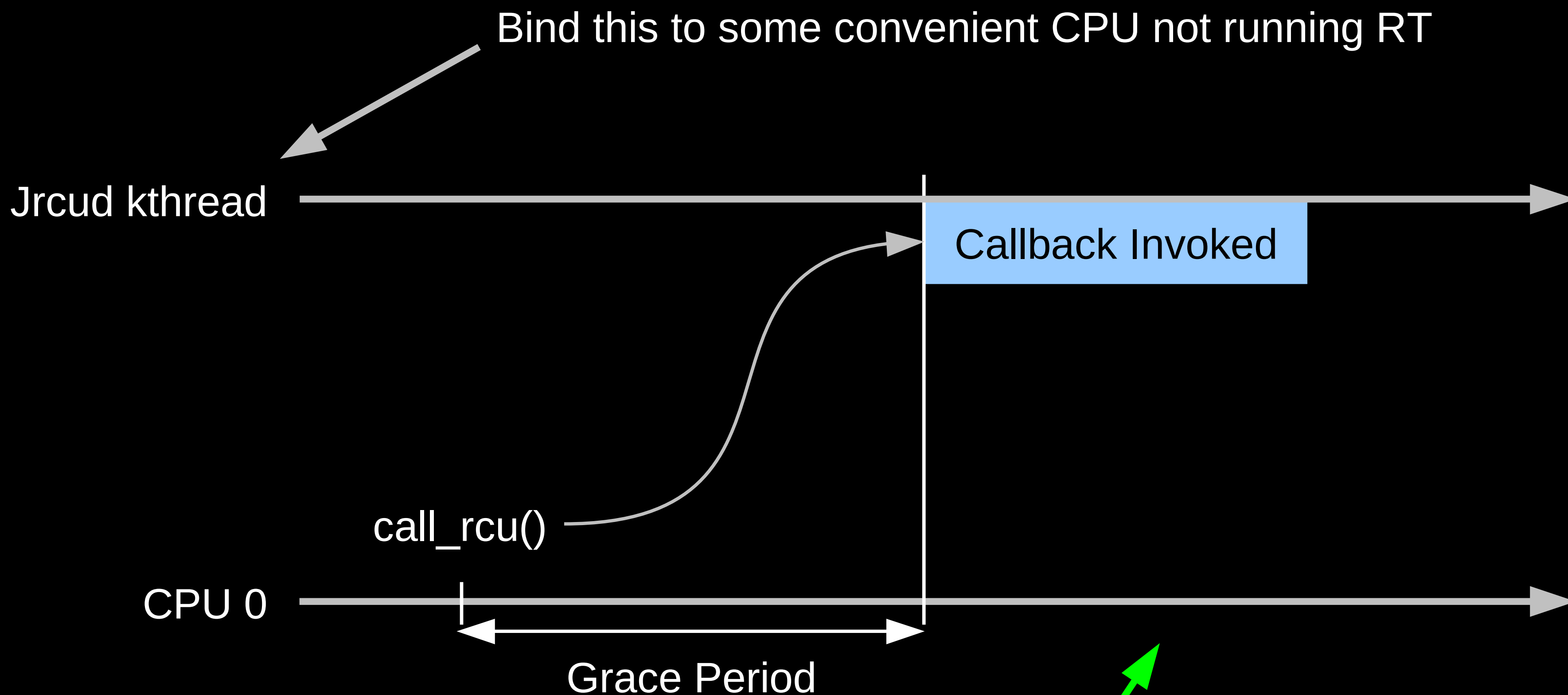
© 2009 IBM Corporation

RCU Callbacks, Houston/Korty Style



© 2009 IBM Corporation

RCU Callbacks, Houston/Korty Style



No disruption!
But also no scalability,
no energy efficiency,
expensive readers, ...

© 2009 IBM Corporation

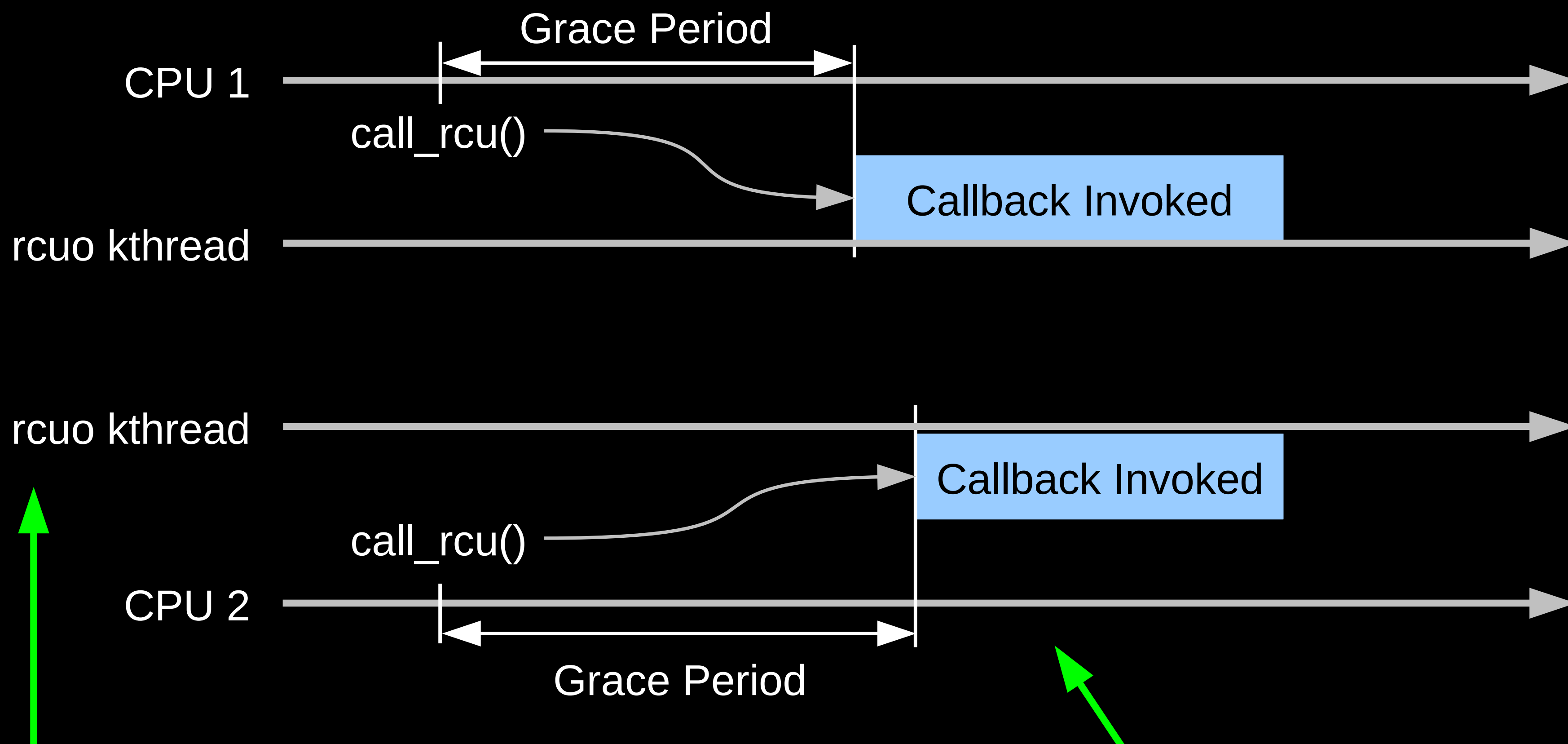
But Mainline RCU Still Does Not Offload Callbacks

- 2012: Time to remedy this situation!
  - And yes, -rt runs callbacks in kthread, but does not offload them
  - Also, recent mainline preferentially invokes callbacks during idle
  - But offloading is still the gold standard of real-time response
- Where to start? Prototype!!!
  - Designate no-callbacks (no-CBs) CPUs at boot time
    - rcu_nocbs accepts list of CPUs
  - One kthread per no-CBs CPU with “rcuoN” name, where “N” is the number of the CPU being offloaded
  - Must work reasonably with dyntick-idle, CPU hotplug, ...
  - OK to require at least one non-no-CBs CPU in the system (CPU 0)
  - Must run on large systems, but OK to limit number of no-CBs CPUs
  - User's responsibility to place kthreads, if desired

© 2009 IBM Corporation

IBM

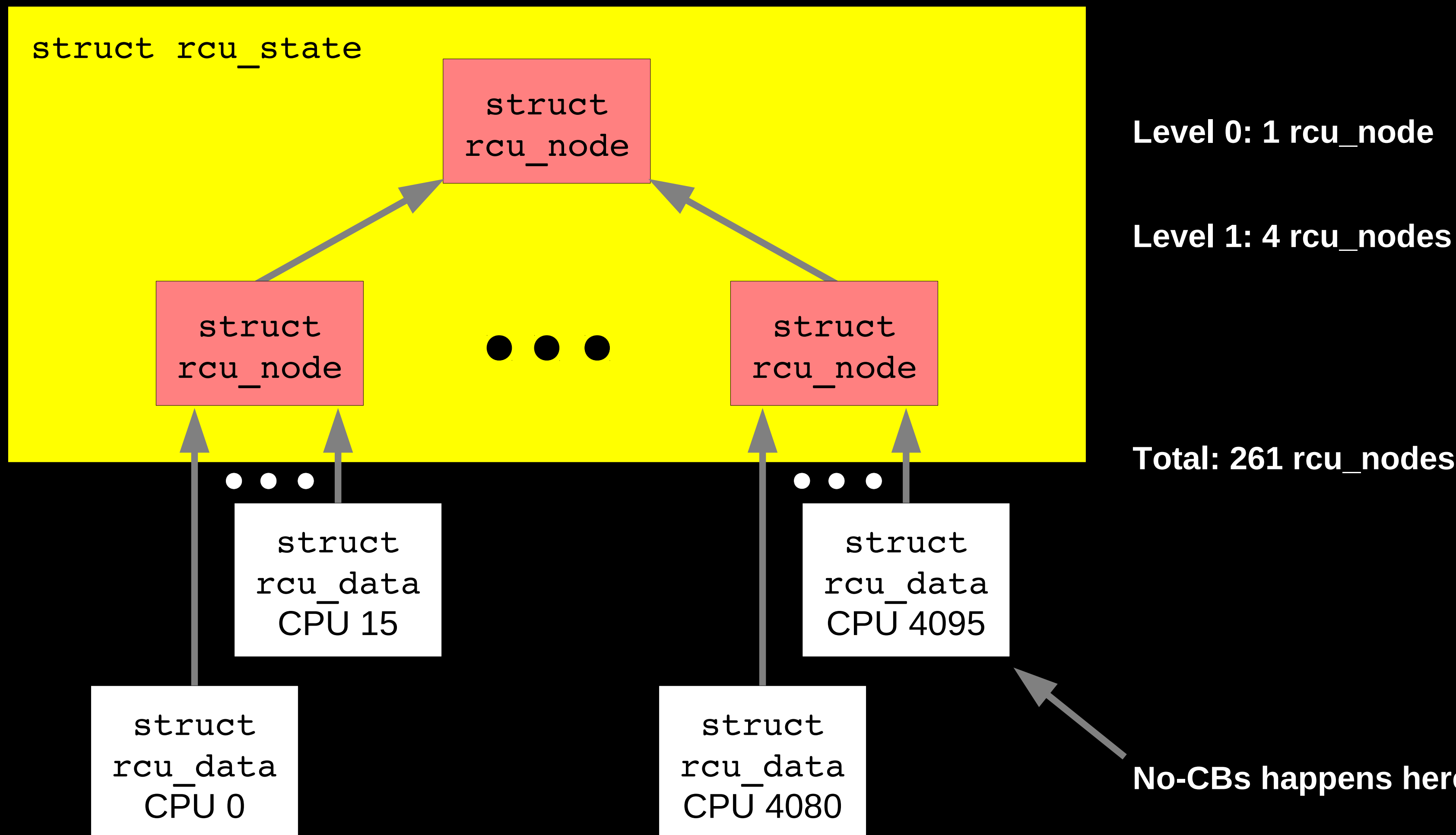
RCU Callbacks, Houston/Korty for TREE_RCU



© 2009 IBM Corporation

IBM

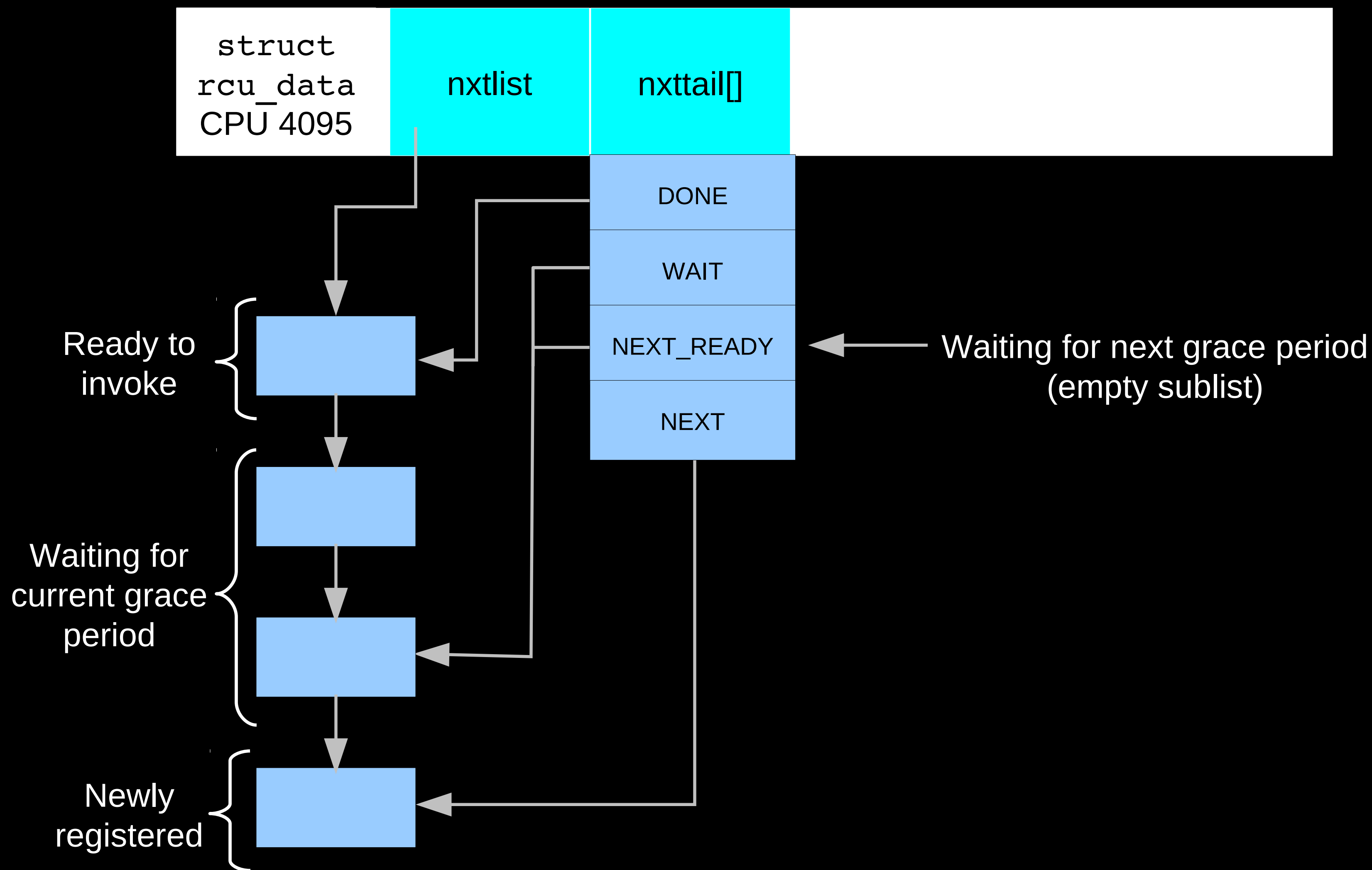
RCU Data Structures (One For Each Flavor)



© 2009 IBM Corporation

IBM

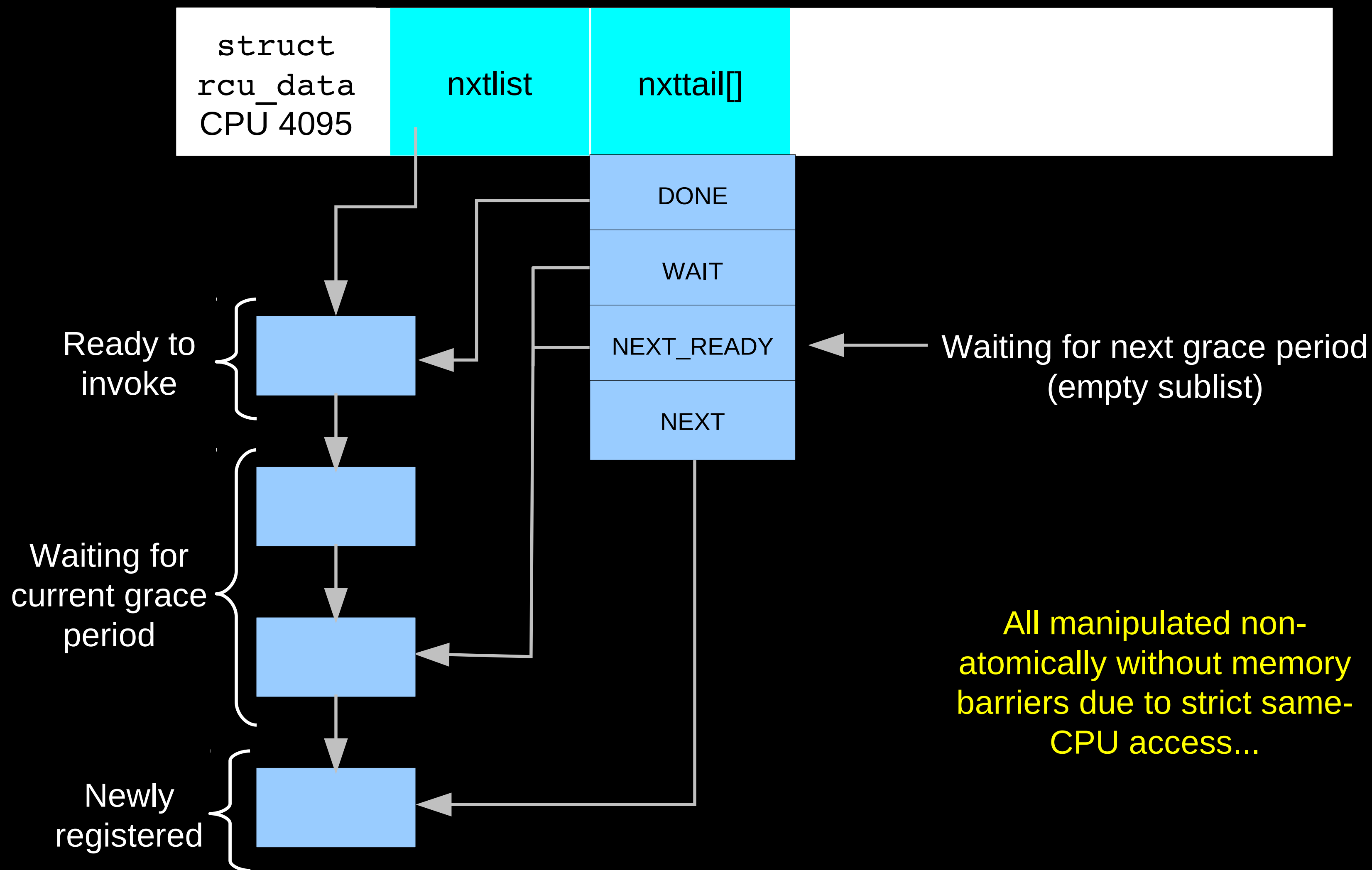
Existing Per-CPU Callback Lists With Tail Pointers



© 2009 IBM Corporation

IBM

Existing Per-CPU Callback Lists With Tail Pointers

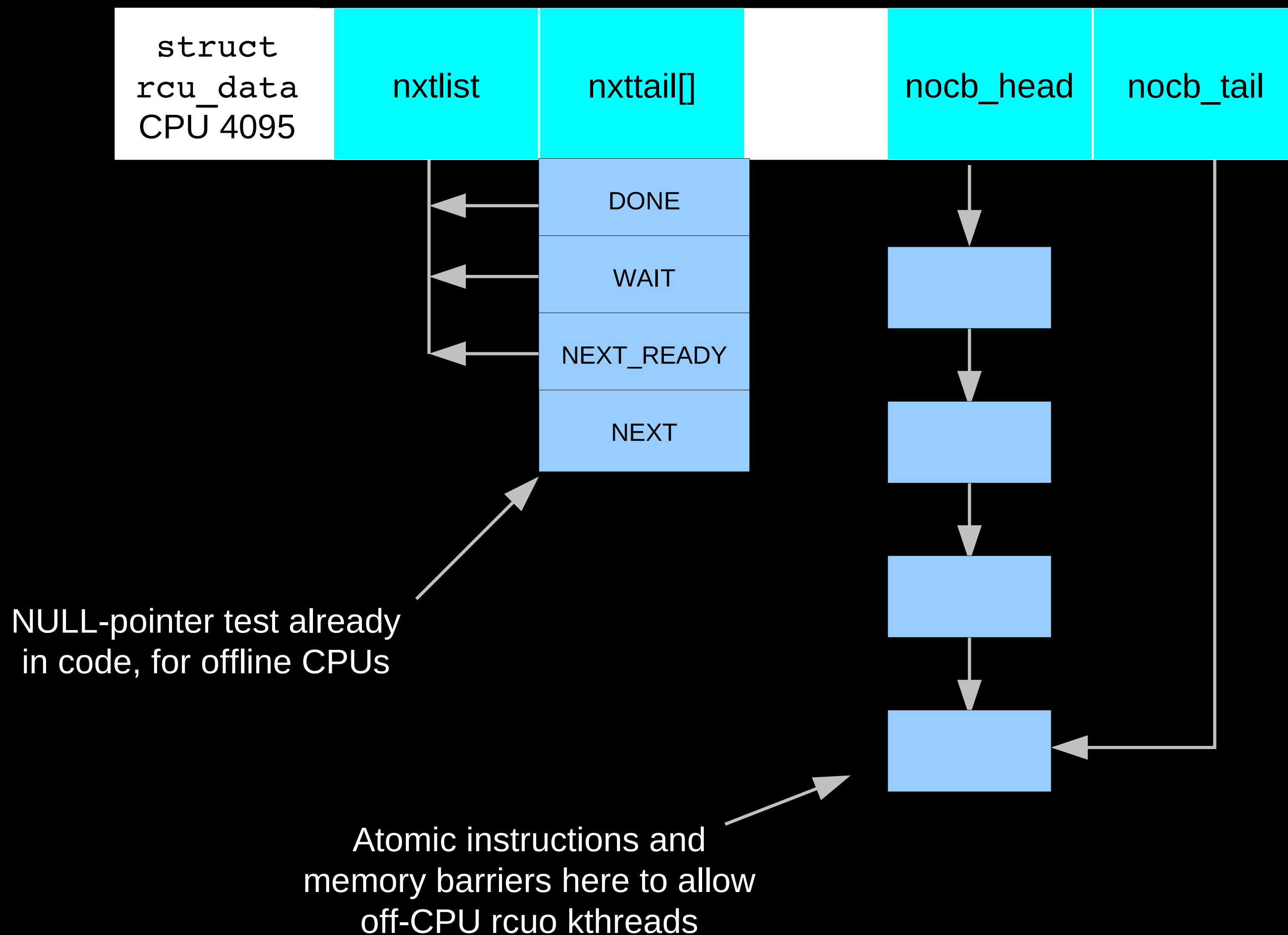


All manipulated non-atomically without memory barriers due to strict same-CPU access...

© 2009 IBM Corporation

IBM

No-CBs Per-CPU Callback Lists With Tail Pointer



© 2009 IBM Corporation

No-CBs Callbacks Setup

- "rcu_nocbs=" kernel boot parameter
  - Takes a list of no-CBs CPUs
  - CPU 0 cannot be no-CBs CPU: boot code kicks it out of list
- "rcu_nocb_poll" kernel boot parameter
  - If non-zero, "rcuo" kthreads poll for callbacks
  - Otherwise, call_rcu() does explicit wake_up() as needed
- Both are dumped to dmesg at boot time along with the usual RCU configuration messages

© 2009 IBM Corporation

Flow of Callbacks For No-CBs CPUs

- Get here when NEXT pointer is NULL
  - If CPU is not a no-CBs CPU, issue warning (offline CPU) and return
- Enqueue callback:

```
old_rhpp = xchg(&rdp->nocb_tail, rhtp);
ACCESS_ONCE(*old_rhpp) = rhp;
```

- If queue was empty (or way full), wake corresponding kthread
- The kthread will dequeue all callbacks:

```
list = ACCESS_ONCE(rdp->nocb_head);
ACCESS_ONCE(rdp->nocb_head) = NULL;
tail = xchg(&rdp->nocb_tail, &rdp->nocb_head);
```

- The “tail” variable is used to validate that full list is received:

```
while (next == NULL && &list->next != tail) {
        schedule_timeout_interruptible(1);
        next = list->next;
}
```


© 2009 IBM Corporation

But We Also Must Wait For An RCU Grace Period...

- Could just use synchronize_rcu()

© 2009 IBM Corporation

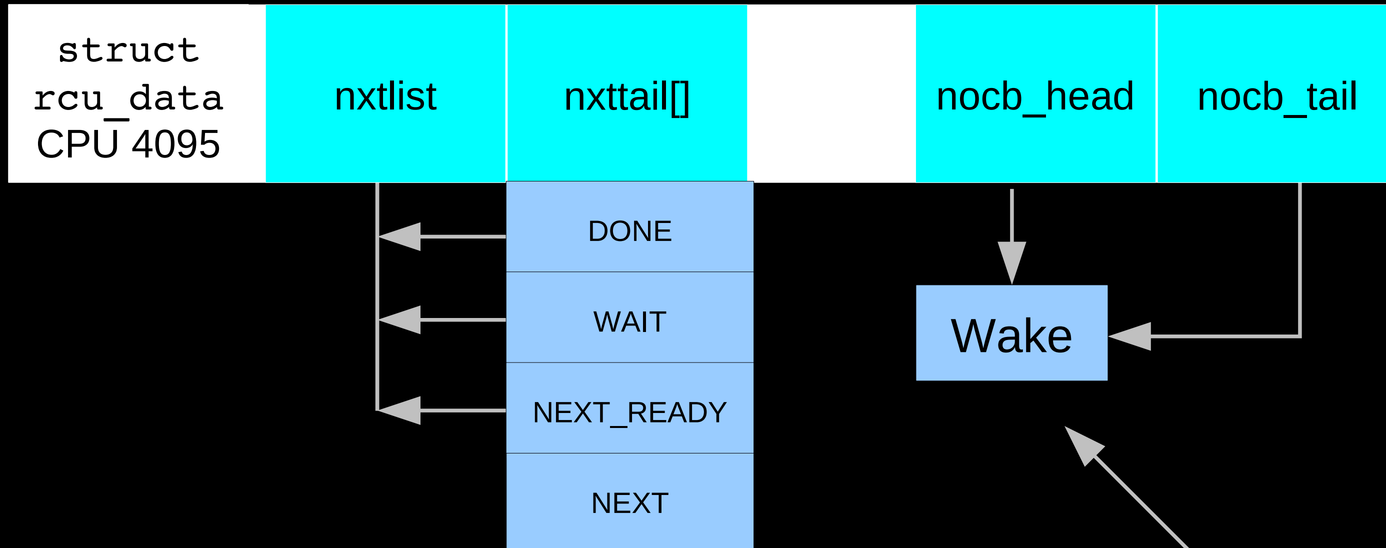
But We Also Must Wait For An RCU Grace Period...

- Could just use synchronize_rcu()
- But if this is an no-CBs CPU, then all that does is to queue the callback on the ->nocb_head queue
- Which won't be invoked until after the kthread invokes the callbacks it currently has
- Which the kthread won't do until after the newly queued callback is invoked
- Resulting in the situation shown on the next slide...

© 2009 IBM Corporation

IBM

No-CBs Callback-List Deadlock



NULL-pointer test already in code, for offline CPUs

Cannot execute until previous batch is invoked... Which won't happen until this callback is invoked. Deadlock!!!

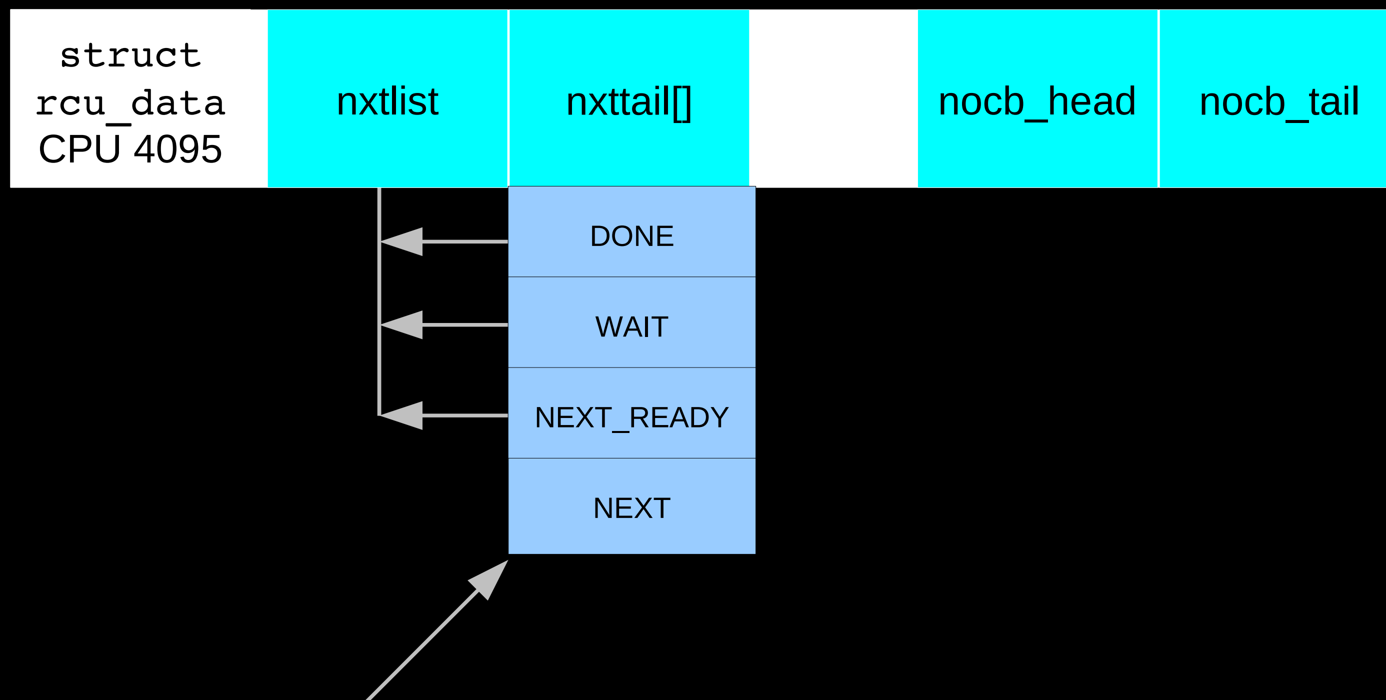
© 2009 IBM Corporation

But We Also Must Wait For An RCU Grace Period...

- Could just use synchronize_rcu()
- But if this is an no-CBs CPU, then that does is queue the callback on the ->nocb_head queue
- Which won't be accessed until after the grace period elapses
- Which won't end because the kthread won't access the callback
- So rely on the fact that CPU 0 is never a no-CBs CPU
  - smp_call_function_single() to make CPU 0 queue the callback
  - Which limits the number of no-CBs CPUs on large systems
  - Which will be fixed later: remember, this is a prototype

© 2009 IBM Corporation

No-CBs Callback-List Deadlock

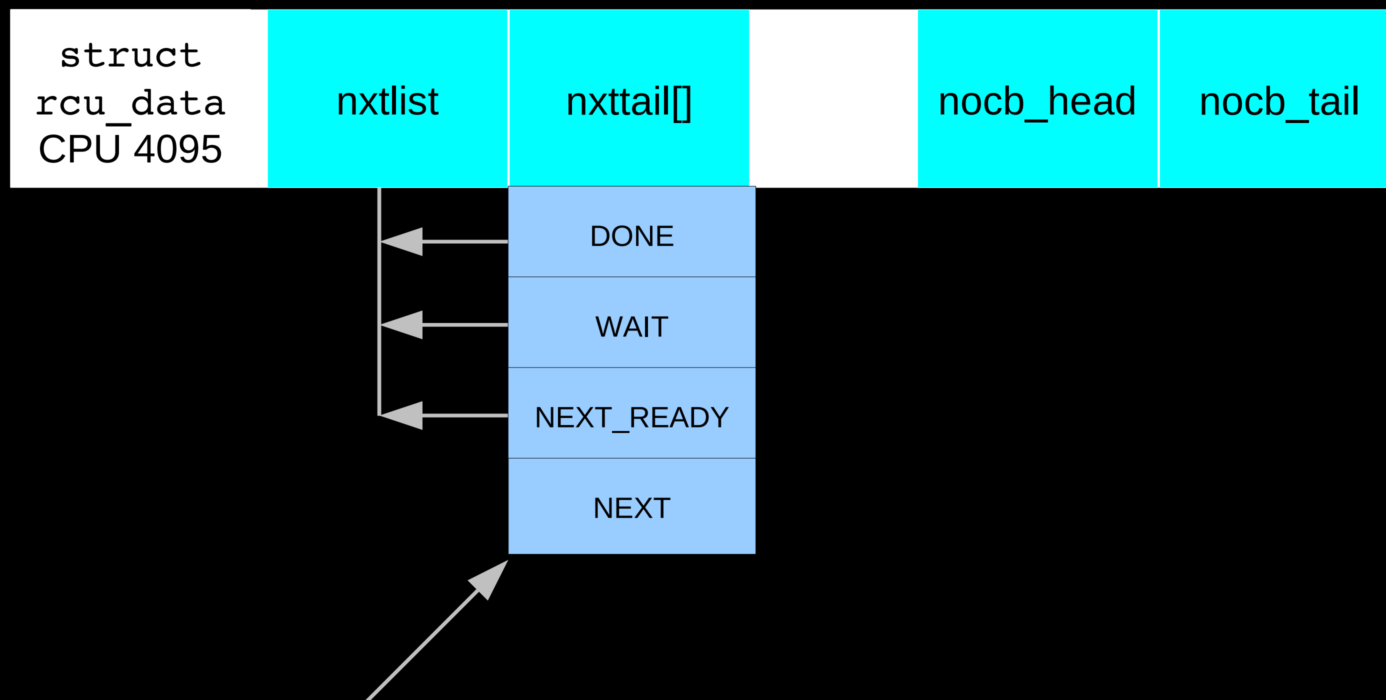


CPU 0 guaranteed to be using this list, so grace-period callback will proceed normally

© 2009 IBM Corporation

IBM

No-CBs Callback-List Deadlock



CPU 0 guaranteed to be using this list, so grace-period callback will proceed normally:

Which means that at least one CPU must remain non-no-CBs CPU!

© 2009 IBM Corporation

CPU Hotplug Considerations

- When a non-no-CBs CPU is offlined, its callbacks are adopted by some other CPU
- But we don't need to do this for no-CBs CPUs
  - The corresponding kthread will continue handling the callbacks regardless of the CPU being offline
- Three complications:
  - rcu_barrier() needs to worry about no-CBs CPUs, even if offline
  - No-CBs CPUs must adopt callbacks onto nocb_head rather than the usual nxtlist
  - Not permitted to offline the last non-no-CBs CPU
  - “Simple matter of code”

© 2009 IBM Corporation

Prototype Performance Tests

- Two-CPU x86 KVM runs
- Running TREE_PREEMPT_RCU implementation
  - Works fine with TREE_RCU as well
- Booted with “rcu_nocbs=1” (control run w/out no-CBs CPUs)
- In-kernel test code generates 10 self-spawning RCU callbacks, each spinning for a time period controlled by sysfs
  - All initiated on CPU 1
- Shell script counts to 100,000
  - Affinity to either CPU 0 or CPU 1
  - Measure how long the script takes to execute on each CPU

© 2009 IBM Corporation

IBM

Prototype Performance Tests: Crude Test Results

	rcu_nocbs=1		rcu_nocbs disabled	
Spin Duration	CPU 0	CPU 1	CPU 0	CPU 1
500 us	1.3 s	0.8 s	0.8 s	1.2 s
100 us	0.9 s	0.8 s	0.8 s	0.9 s
10 us	0.8 s	0.8 s	0.8 s	0.8 s

Callbacks offloaded from CPU 1

Callbacks remain on CPU 1

© 2009 IBM Corporation

Prototype Complexity According to diffstat

```
include/trace/events/rcu.h |    1
init/Kconfig               |   19 ++
kernel/rcutree.c           |   63 +++++--
kernel/rcutree.h           |   47 +++++
kernel/rcutree_plugin.h    |  397 ++++++++++++++++++++++++++++++++++++++++++++-
kernel/rcutree_trace.c     |   14 +
6 files changed, 524 insertions(+), 17 deletions(-)
```


© 2009 IBM Corporation

Limitations and Future Directions

- Need atomic_inc_long() and friends
  - Currently living dangerously with "int" counters on 64-bit systems
  - I cannot be the only one wishing for atomic_long_t!!!
- Must reboot to reconfigure no-CBs 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.)
- Scalability: 1,000 no-CBs CPUs would not do well
  - Should be able to improve this, but not an issue for prototype
- Must be at least one non-no-CBs CPU (e.g., CPU 0)
  - Scalability fixes would likely fix this as well.
- No energy-efficiency code: lazy & non-lazy CBs? Non-lazy!
  - But do real-time people even care about energy efficiency?
- No-CBs CPUs' kthreads not subject to priority boosting
  - Rely on configurations restrictions for prototype
- Setting all no-CBs CPUs' kthreads to RT prio w/out pinning them: bad!
  - At least on large systems: configuration restrictions
- Thus, I do not expect no-CBs path to completely replace current CB path

© 2009 IBM Corporation

Question From The Speaker...

- Is this approach to callback offloading useful?
  - Real time?
  - High-performance computing?
  - High-speed networking?

© 2009 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.

© 2009 IBM Corporation

Questions?

© 2009 IBM Corporation