

Paul E. McKenney, Meta Platforms Kernel Team

Joel Fernandes, Google

Boqun Feng, Microsoft

Uladzislau Rezki, Sony

Frédéric Weisbecker, SUSE

Neeraj Upadhyay, AMD

Linux Plumbers Conference RCU Office Hours BoF, September 20, 2024



# Recent Linux-Kernel RCU Changes

---

# But First...

---

Your Questions?  
This is Your Time!!!

# Otherwise: RCU Changes

Yes, RCU is still under active development!

# RCU Review

---

# RCU Review (At Speed)

- Purpose
- Core API
- Semantics and Restrictions
- Spatio-Temporal Synchronization

# RCU Review: Purpose

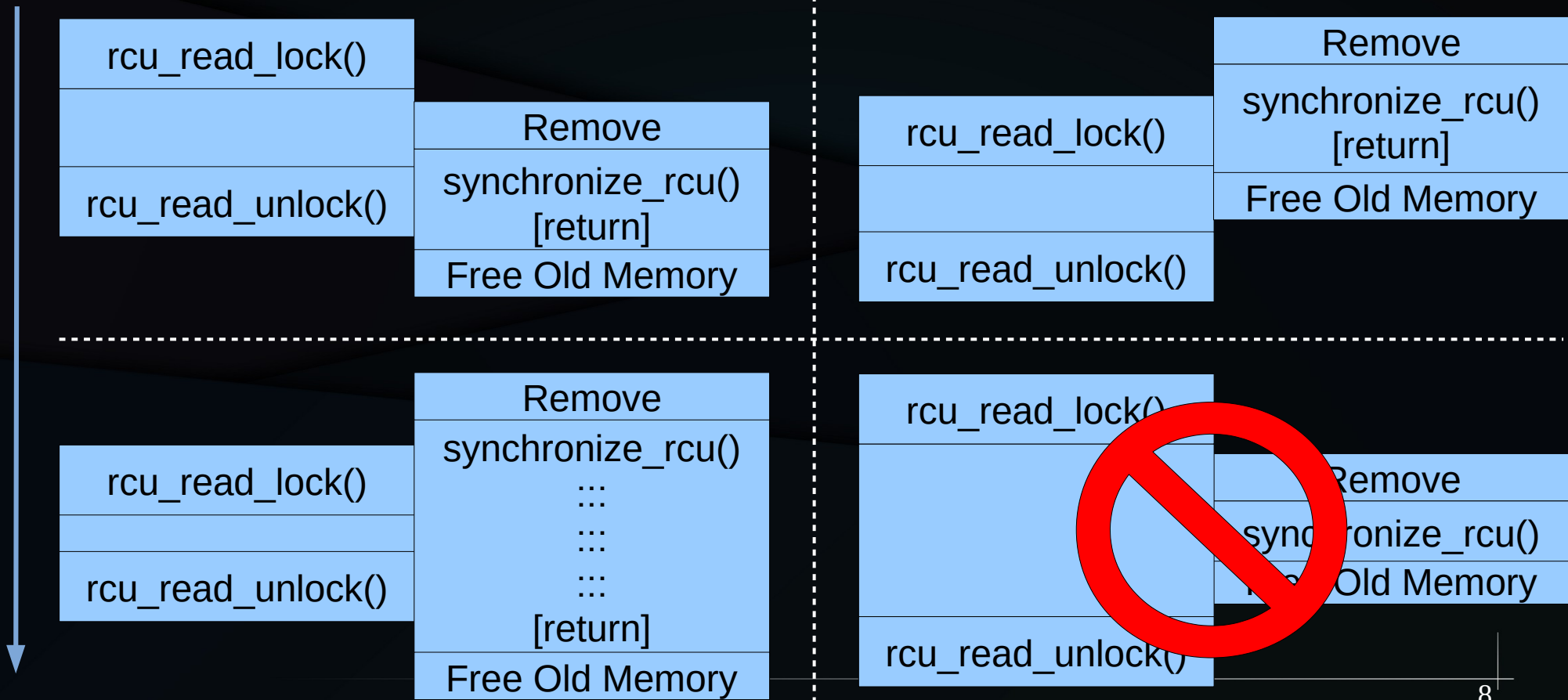
- Global agreement is expensive
  - Finite speed of light and non-zero-sized atoms...
- So use both spatial & temporal synchronization
- RCU is one way to do this
  - Hazard pointers provide another way

# RCU Review: Core API (Space/Time)

- `rcu_read_lock()`: Begin reader
- `rcu_read_unlock()`: End reader
- `synchronize_rcu()`: Wait for pre-existing readers
- `call_rcu()`: Invoke function after pre-existing readers complete
- `rcu_dereference()`: Load RCU-protected pointer
- `rcu_dereference_protected()`: Ditto, but update-side locked
- `rcu_assign_pointer()`: Update RCU-protected pointer

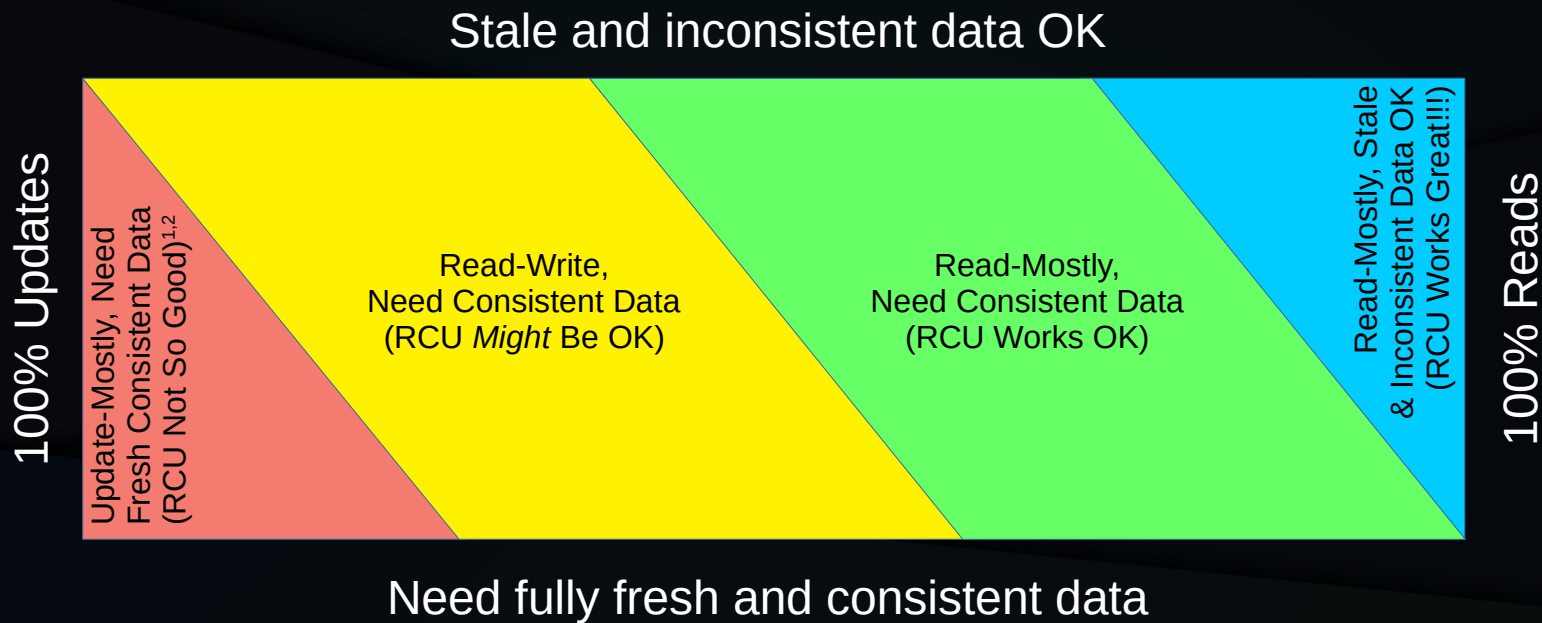
# RCU Review: Semantics (Graphical)

Time (really ordering)





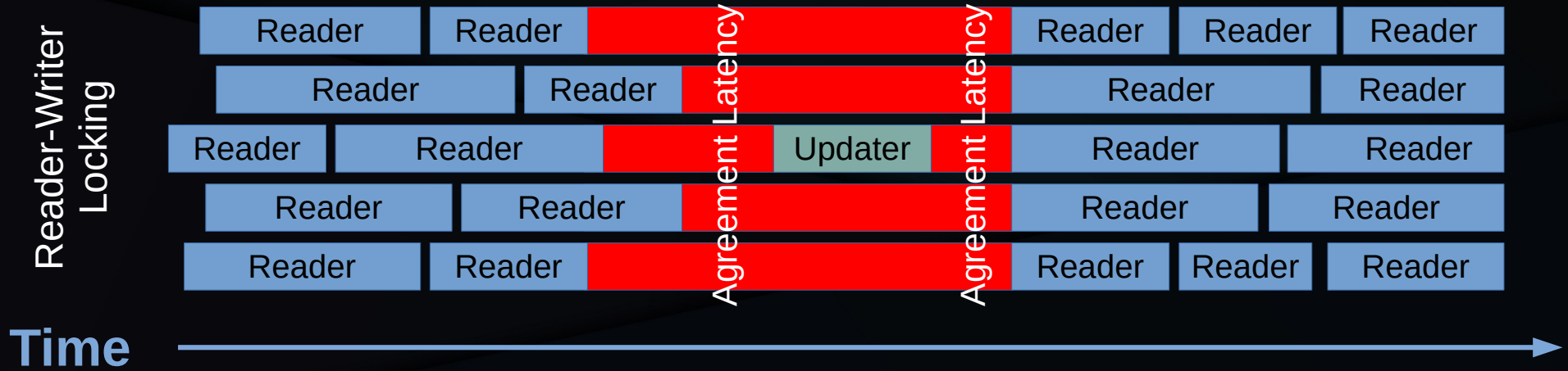
# RCU Review: Restrictions



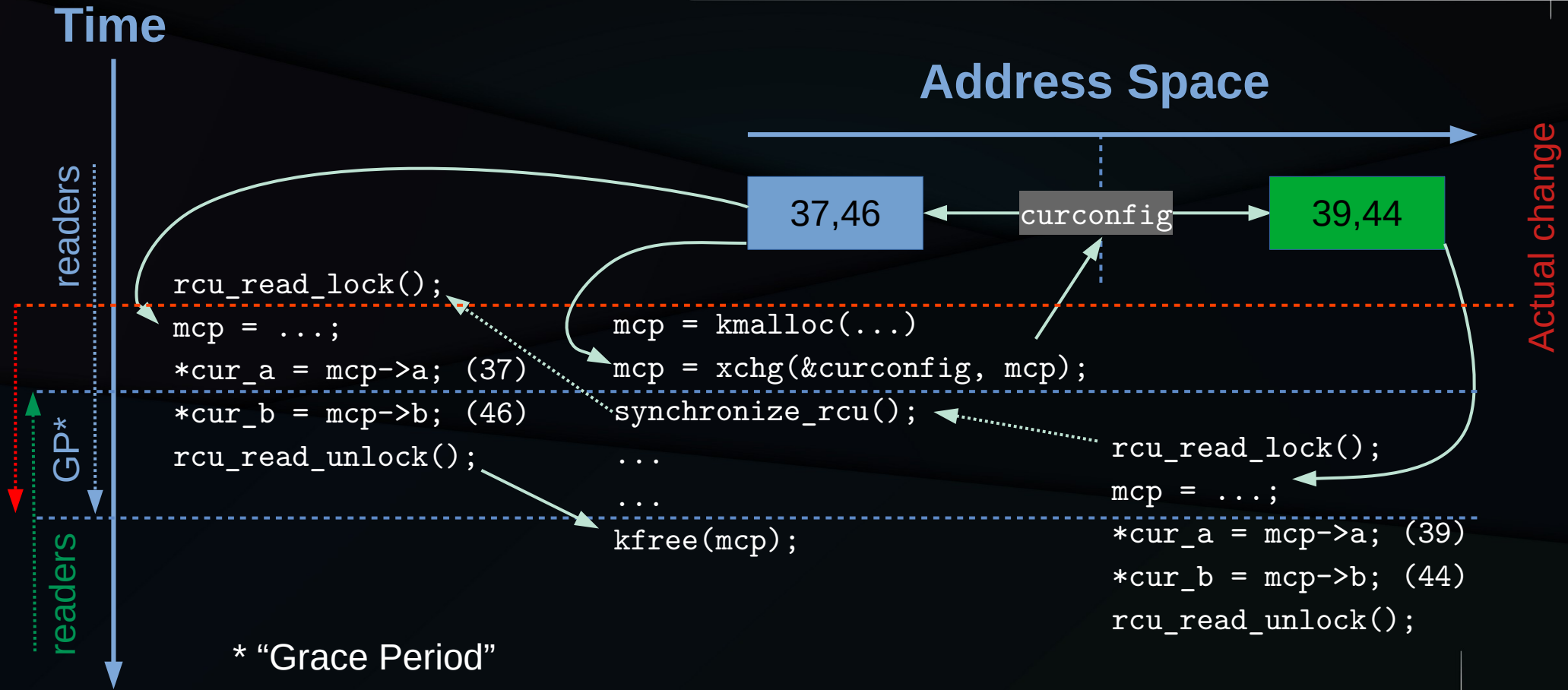
1. RCU provides ABA protection for update-friendly mechanisms (light-weight garbage collector)
2. RCU provides bounded wait-free read-side primitives for real-time use

And RCU is most frequently used for linked data structures.

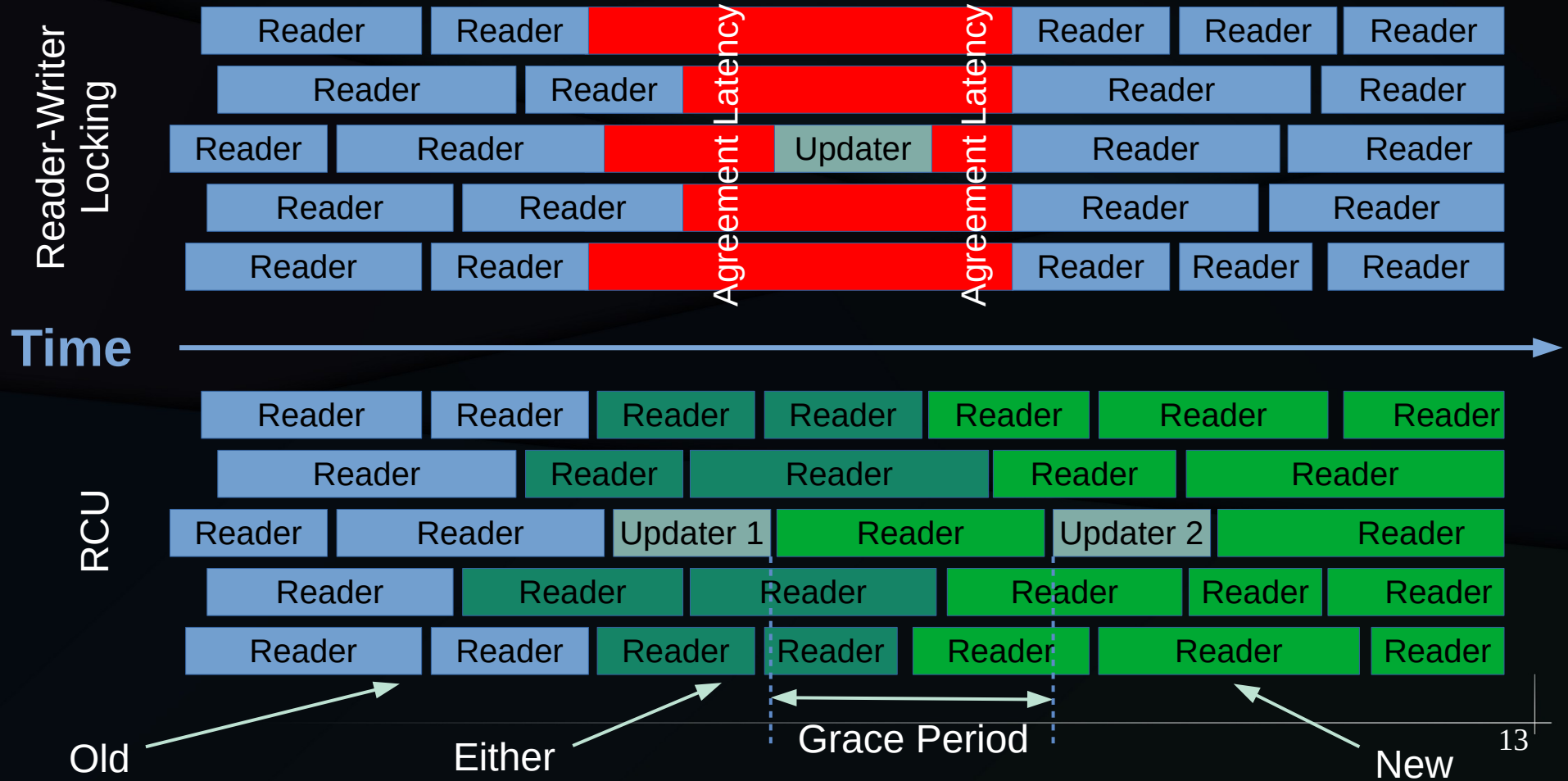
# RCU Review: Global Agreement Cost



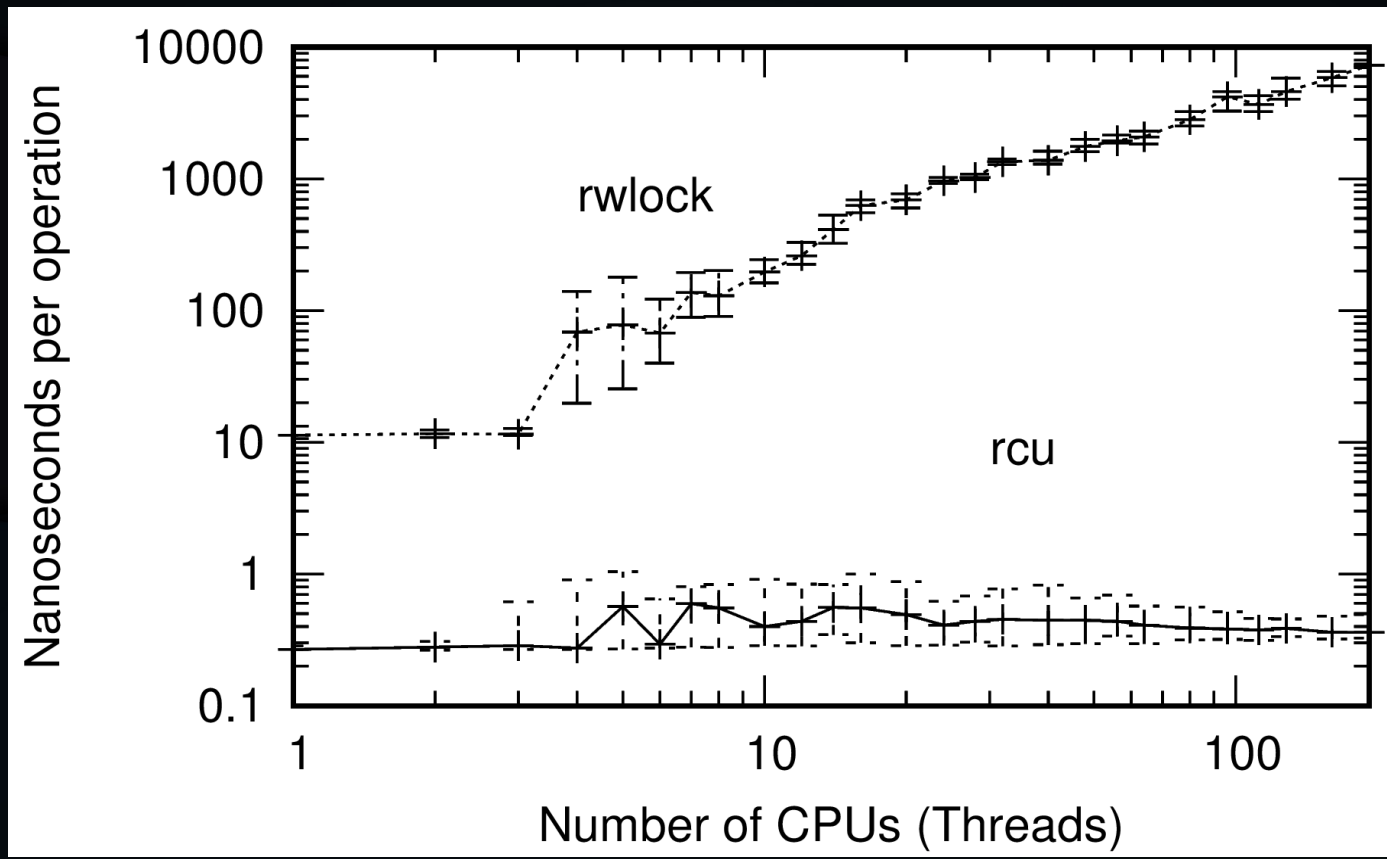
# RCU Review: Code Animation



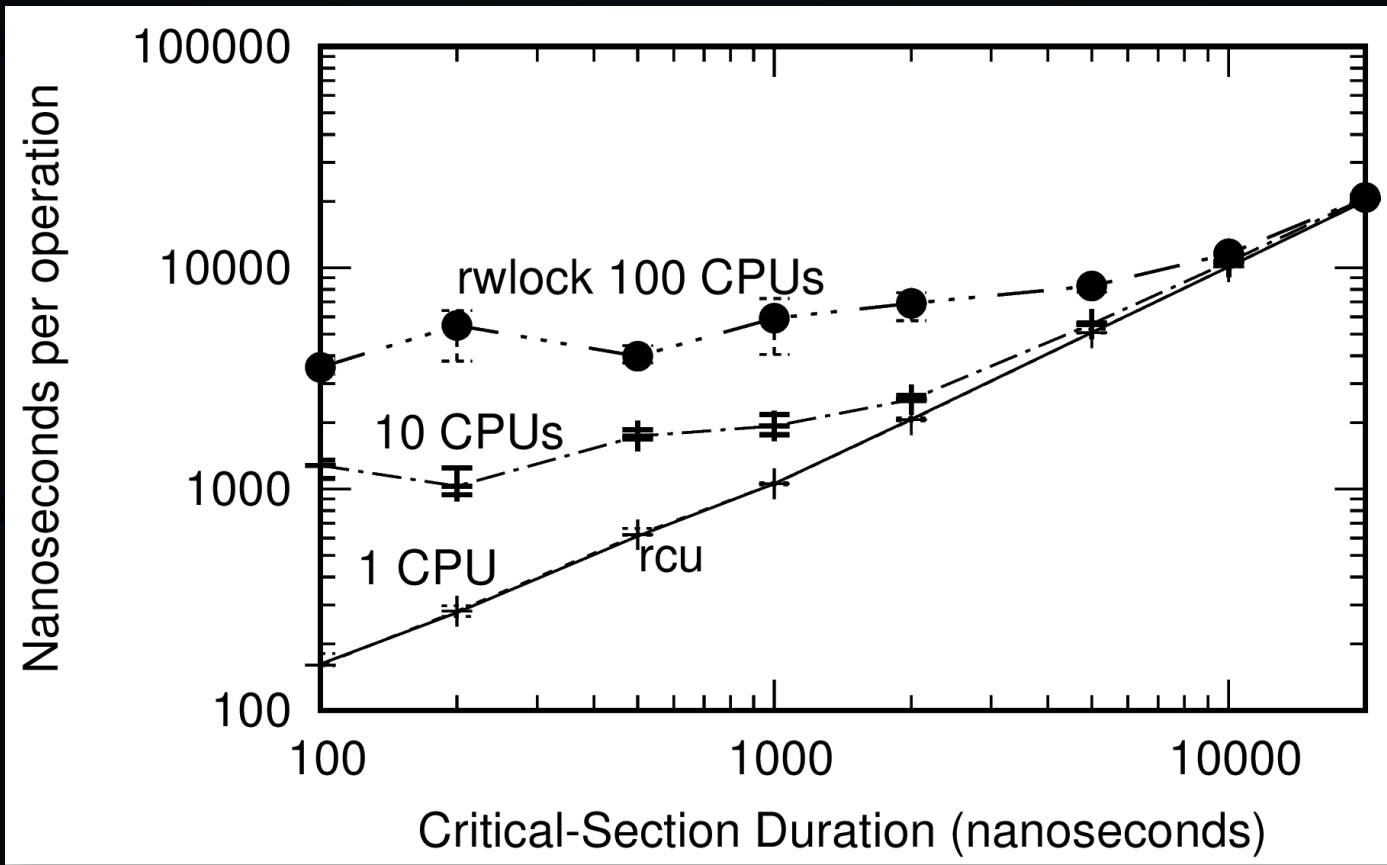
# RCU Review: Code Animation



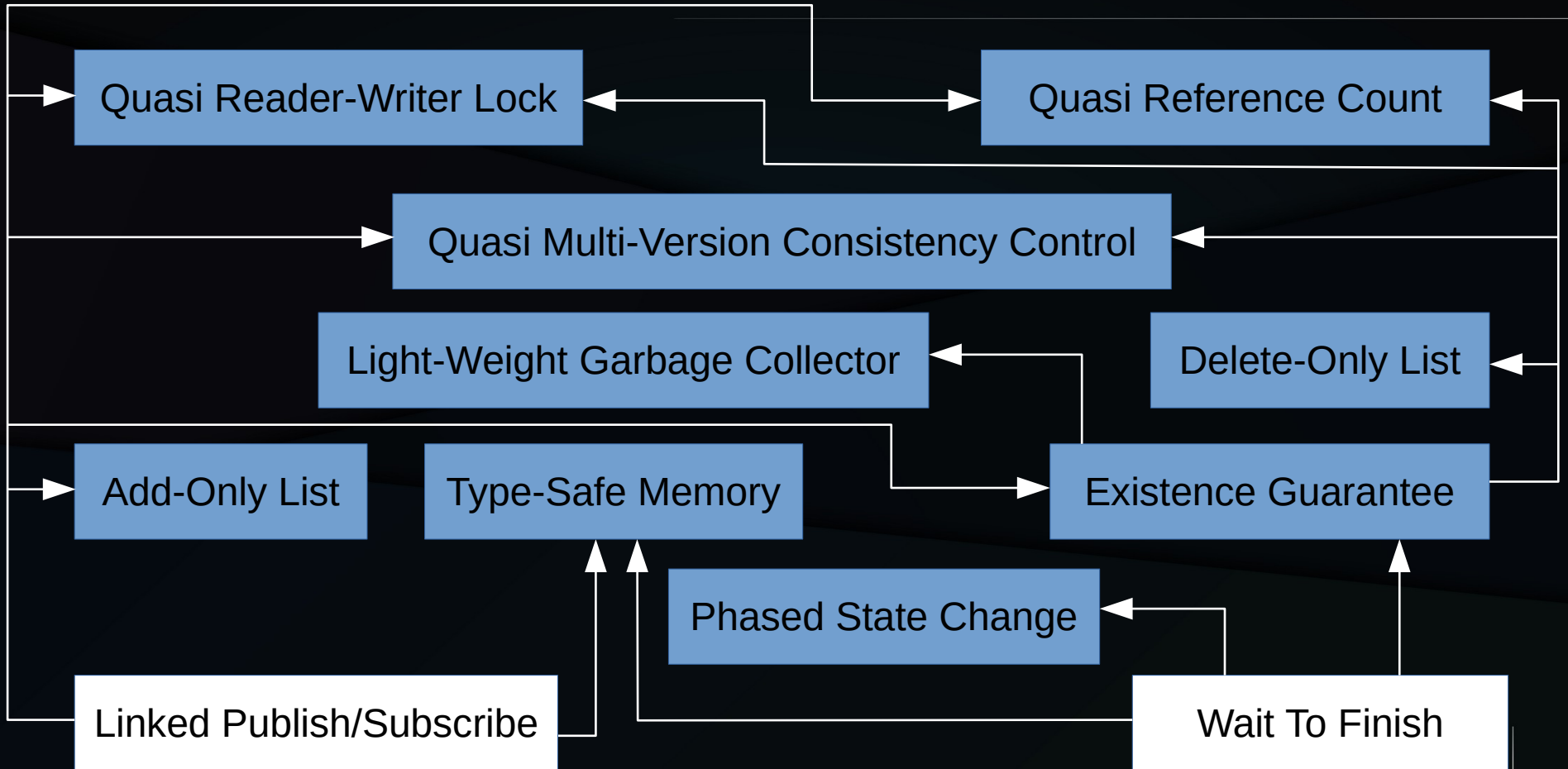
# RCU Review: Scalability (Empty)



# RCU Review: Scalability (Non-Empty)



# RCU Review: Use Cases



# RCU Changes



# RCU Changes Since 2019

- See <https://lwn.net/Articles/988638/>
  - Lazy RCU grace asynchronous periods
  - Reworking of `kfree_rcu()`
  - Polled RCU grace periods
  - Tasks Rude RCU and Tasks Trace RCU
  - New SRCU read-side critical sections
  - Read-side guards
  - RCU callback dynamic (de-)offloading
  - Miscellaneous (no more `CONFIG_RCU_FAST_NO_HZ`, etc.)
- RCU API reference: <https://lwn.net/Articles/988666/>

# RCU Changes

---

- Flavor consolidation
- Optional lockdep expression for `list_for_each_entry_rcu()`
- Single-argument `kfree_rcu()` and `kvfree_rcu()`
- Tasks Trace RCU and Tasks Rude RCU
- Polled grace-period APIs
- Runtime RCU callback (de-)offloading
- SRCU memory-footprint diet
- Real-time expedited grace periods

# Looking Back Two Years From Now ...

- 223 paulmck@kernel.org (45%)
- 67 frederic@kernel.org
- 24 torvalds@linux-foundation.org
- 21 urezki@gmail.com
- 21 vschneid@redhat.com
- 20 qiang1.zhang@intel.com
- 19 joel@joelfernandes.org
- 15 qiang.zhang1211@gmail.com
- 7 thunder.leizhen@huawei.com
- 6 kernelfans@gmail.com
- 3 kuba@kernel.org
- 3 peterz@infradead.org

# And Two Years From Last Year ...

```
503 paulmck@kernel.org (63%)
 78 frederic@kernel.org
 26 urezki@gmail.com
 20 joel@joelfernandes.org
 14 peterz@infradead.org
 13 neeraju@codeaurora.org
 10 qiang1.zhang@intel.com
  7 mchehab+huawei@kernel.org
  7 zhouzhouyi@gmail.com
  6 bigeasy@linutronix.de
  6 tglx@linutronix.de
  5 quic_neeraju@quicinc.com
```

# Backup

---

# RCU Flavor Consolidation

---

# RCU Flavor Consolidation: Impetus

Date: Sat, 3 Mar 2018 17:50:44 -0800  
From: Linus Torvalds <torvalds@linux-foundation.org>  
To: Jann Horn <jannh@google.com>, Tejun Heo <tj@kernel.org>, Paul McKenney  
<paulmck@linux.vnet.ibm.com>  
Cc: Benjamin LaHaise <bcr1@kvack.org>, security@kernel.org, Al Viro  
<viro@zeniv.linux.org.uk>  
Subject: Re: AIO locking bug in lookup\_iocx()  
From linus971@gmail.com Sat Mar 3 17:54:39 2018

[ Adding Al, Paul and Tejun and to the cc too for various reasons ]

On Fri, Mar 2, 2018 at 3:14 PM, Jann Horn <jannh@google.com> wrote:

[ . . . ]

> I'm not sending a patch because I'm not sure whether the intent here is to  
> use RCU, and if so, whether it should be RCU-sched or normal RCU.

It's meant to use regular RCU.

But then in commit a4244454df12 ("percpu-refcount: use RCU-sched  
insted of normal RCU") the percpu refcounts were changed to use  
RCU-sched.

.. and in the process apparently broke the AIO RCU locking.

Tejun, Paul, please tell me why I'm wrong.

**security@kernel.org**



# Root Cause Of Exploit

```
void reader(void)
{
    rcu_read_lock_sched();
    /*
     * Access RCU-
     * protected data.
     */
    rcu_read_unlock_sched();
}
```

```
void updater(void)
{
    /* Remove old data. */
    synchronize_rcu();
    /* Free old data. */
}
```



# Need Consistency, Except That...

```
rcu_read_lock();  
rcu_read_unlock();
```

```
synchronize_rcu();
```

```
rcu_read_lock_bh();  
rcu_read_unlock_bh();
```

```
synchronize_rcu_bh();
```

```
rcu_read_lock_sched();  
rcu_read_unlock_sched();
```

```
synchronize_sched();
```

**...to err is human!**

**Plus userspace controls content of much kernel data!!!**

# And the Call to Action

Date: Sun, 4 Mar 2018 10:53:54 -0800  
From: Linus Torvalds <torvalds@linux-foundation.org>  
To: Tejun Heo <tj@kernel.org>  
Cc: Jann Horn <jannh@google.com>, Paul McKenney <paulmck@linux.vnet.ibm.com>, Benjamin LaHaise <bcr1@kvack.org>, security@kernel.org, Al Viro <viro@zeniv.linux.org.uk>  
Subject: Re: AIO locking bug in lookup\_ioctx()  
From linus971@gmail.com Sun Mar 4 10:56:59 2018

[ . . . ]

I've been confused before, and this time it was an actual security bug. Admittedly one that is probably almost impossible to ever hit in practice or mis-use, but still.

I repeat: I really love the traditional RCU, but I *\*despise\** how there are a million of these things that causes real problems.

**Paul, is there really no way out of this mess?**

The only reason for rcu-sched to exist in the first place is that the regular RCU had been made so much slower with PREEMPT\_RCU. In other words, the proliferation of different insane RCU implementations ends up feeding on itself, and causing more and more of the proliferation.

Paul, is there really no way out of this mess?

# Desired State (Usability/Security)

```
rcu_read_lock();  
rcu_read_unlock();
```

```
rcu_read_lock_bh();  
rcu_read_unlock_bh();
```

```
rcu_read_lock_sched();  
rcu_read_unlock_sched();
```

```
synchronize_rcu();
```

# Backport `synchronize_rcu()`

- v4.20 or later: Just backport with no change
- v4.19 or earlier:
  - Only waiting on `rcu_read_lock()`:
    - Use `synchronize_rcu()`, as in no change
  - Otherwise, use `synchronize_rcu_mult()`:  
`synchronize_rcu_mult(call_rcu, call_rcu_bh, call_rcu_sched);`
    - Using whichever `call_rcu*()` variants you need
    - Chain `call_rcu()`, `call_rcu_bh()`, and `call_rcu_sched()`

# lockdep & list\_for\_each\_entry\_rcu()

# Lockdep & list\_for\_each\_entry\_rcu()

- Non-lockdep style (still supported):
  - list\_for\_each\_entry\_rcu(pos, head, member)
  - hlist\_for\_each\_entry\_rcu(pos, head, member)
- With lockdep support:
  - list\_for\_each\_entry\_rcu(pos, head, member[, cond])
  - hlist\_for\_each\_entry\_rcu(pos, head, member[, cond])
    - Example “cond”: “lockdep\_is\_held(&event\_mutex)”

# Single-Argument kvfree\_rcu()

# Single-Argument `kfree_rcu()`

- Classic way: `kfree_rcu(p, rh)`
  - Requires an `rcu_head` field named “rh” in object
  - Never sleeps
  - Will continue to be supported
- New way: `kfree_rcu(p)`
  - No `rcu_head` required, but can sleep if OOM
- `kfree_rcu()` is a synonym for `kfree_rcu()`
  - Uses `kfree_bulk()` for cache locality (since v5.7)



# RCU Tasks {,Rude,Trace}

# RCU Tasks {,Rude,Trace}

- RCU Tasks and RCU Tasks Rude:
  - Used for tracing, e.g., freeing trampolines
- RCU Tasks Trace
  - Used for sleepable BPF trampolines
- All three variants are extremely specialized
  - Check with their other users before using them!!!

# RCU Tasks {,Rude,Trace}

## **Voluntary context switch (v3.18):**

synchronize\_rcu\_tasks()

call\_rcu\_tasks()

rcu\_barrier\_tasks()

## **Any context switch (v5.8):**

synchronize\_rcu\_tasks\_rude()

call\_rcu\_tasks\_rude()

rcu\_barrier\_tasks\_rude()

## **Explicit read-side markers (v5.8):**

rcu\_read\_lock\_trace()

rcu\_read\_unlock\_trace()

rcu\_read\_lock\_trace\_held()

synchronize\_rcu\_tasks\_trace()

call\_rcu\_tasks\_trace()

rcu\_barrier\_tasks\_trace()

# Polled Grace-Period APIs

---

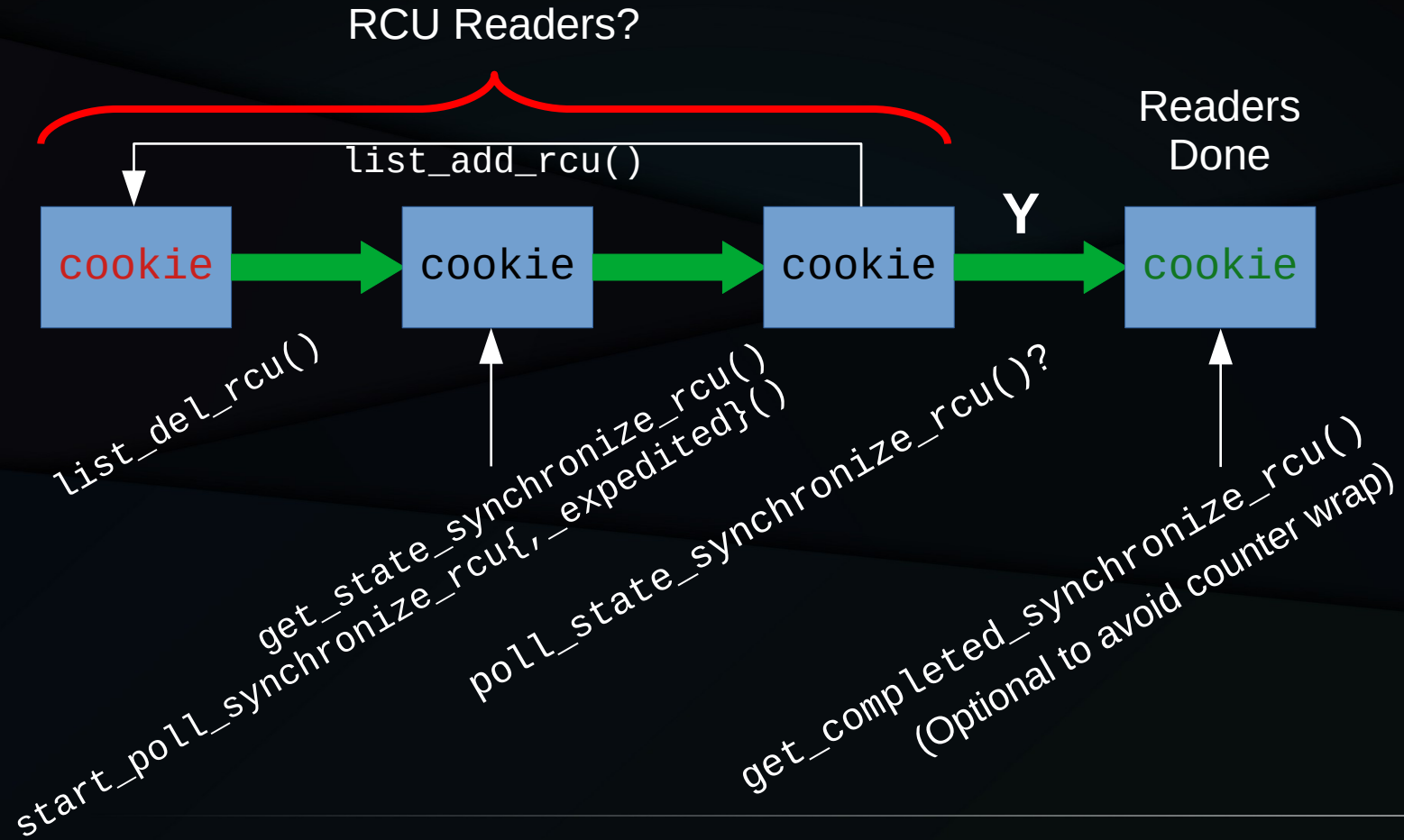
# Polled Grace-Period APIs

- `get_state_synchronize_rcu()`
- `cond_synchronize_rcu()`
  - v3.14 and later
- `start_poll_synchronize_rcu()`
- `poll_state_synchronize_rcu()`
  - v5.12 and later
- `get_completed_synchronize_rcu()`
- `start_poll_synchronize_rcu_expedited()`
- `cond_synchronize_rcu_expedited()`
  - Maybe v5.20 and later?
  
- `get_state_synchronize_srcu()`
- `start_poll_synchronize_srcu()`
- `poll_state_synchronize_srcu()`
  - v5.12 and later

# Using Polled Grace-Period APIs (1/2)

```
/* Get grace-period cookie. */  
cookie = get_state_synchronize_rcu();  
/* Do other work. */  
do_something();  
/* If grace period not done, wait for the rest of it. */  
cond_synchronize_rcu{,_expedited}(cookie);
```

# Using Polled Grace-Period APIs (2/2)



# Polled Grace Period Caveat

```
/* Get grace-period cookie. */
cookie = get_state_synchronize_rcu();
synchronize_rcu{,_expedited}();

/*
 * Can trigger!!! (1) Counter wrap. (2) Races that
 * result in partially overlapping normal and expedited
 * grace periods. Can eliminate #2, but with either
 * more storage (2 unsigned longs) or faster counter wrap.
 */
WARN_ON_ONCE(!poll_state_synchronize_rcu(cookie));
```



# Polled Grace Period Guarantee

```
/* Get grace-period cookie. */  
cookie = get_state_synchronize_rcu();  
  
/* Only one grace period can be lost due to race. */  
synchronize_rcu{,_expedited}();  
synchronize_rcu{,_expedited}();  
  
/* Can trigger!!! But only due to counter wrap. */  
WARN_ON_ONCE(!poll_state_synchronize_rcu(cookie));
```

# Polled Grace-Period APIs: Full Size

- `get_state_synchronize_rcu_full()`
- `cond_synchronize_rcu_full()`
- `start_poll_synchronize_rcu_full()`
- `poll_state_synchronize_rcu_full()`
- `get_completed_synchronize_rcu()`
- `get_completed_synchronize_rcu_full()`
  - Maybe v6.2 and later
- `NUM_ACTIVE_RCU_POLL_OLDSTATE`
- `NUM_ACTIVE_RCU_POLL_FULL_OLDSTATE`
  - Maybe v6.3 and later?

# Full Polled Grace Period Guarantee

```
struct rcu_gp_oldstate *rgos;

/* Get grace-period cookie. */
cookie = get_state_synchronize_rcu(&rgos);

/* Grace periods cannot be lost due to race. */
synchronize_rcu{,_expedited}();

/* Can trigger!!! But only due to counter wrap. */
WARN_ON_ONCE(!poll_state_synchronize_rcu_full(&rgos));
```

# Runtime Callback (De-)Offloading

# Runtime Callback (De-)Offloading

- Requires `CONFIG_RCU_NOCB_CPU=y`
  - Implied by `CONFIG_NO_HZ_FULL=y`
- Requires `rcu_nocbs` kernel boot parameter
  - Implied by `nohz_full` kernel boot parameter
- Available only in kernel for `rcu_nocbs` CPUs:
  - `rcu_nocb_cpu_offload(cpu)`
  - `rcu_nocb_cpu_deoffload(cpu)`

# Putting SRCU on a Memory Diet

# Putting SRCU on a Memory Diet

- Tree SRCU has a compile-time `srcu_node` tree
  - `NR_CPUS=4096` means 261 array elements
    - About 100 bytes each, which is not a big deal, but...
  - Requires bigger instructions to access later fields
  - And big `srcu_node` arrays wasted on real systems, which rarely have thousands of CPUs!!!
    - And most `srcu_struct` instances don't even need the `srcu_node` array at all!

# Putting SRCU on a Memory Diet

- Separately allocate srcu\_node tree
  - But only if requested or actually needed
    - srcutree.big\_cpu\_lim: # CPUs in “big” system (128)
    - srcutree.convert\_to\_big: When to allocate?
      - 0xX0: Never
      - 0xX1: At init\_srcu\_struct() time (AKA “always”)
      - 0xX2: When rcutorture so chooses
      - 0xX3: Decide 0/1 at boot based on big\_cpu\_lim (0x03 is default)
      - 0x1X: Above plus if lock contention is high and not OOM
  - Sized based on actual number of CPUs



# Putting SRCU on a Memory Diet

- Default to allocate on contention (0x13)?
  - No: This has not yet earned our trust
  - No useful way to report allocation failure
    - Contention continues if OOM
    - Which is *probably* OK...

# Real-Time Expedited Grace Periods

# Real-Time Expedited Grace Periods

- Brief history of RCU CPU stall warnings:
  - 1990s: Dynix/PTX 1.5 seconds
  - 2000s: Linux 60 seconds
  - 2010s: Linux 21 seconds
  - 2020s: 20 milliseconds proposed for expedited GPs
    - CONFIG\_RCU\_EXP\_CPU\_STALL\_TIMEOUT
      - Defaults to 20 ms if CONFIG\_ANDROID, 21000 ms otherwise
    - But how is this going to work???

# Real-Time Expedited Grace Periods

- Expedited grace periods driven by workqueues
  - Normal SCHED\_OTHER priority
- New CONFIG\_RCU\_EXP\_KTHREAD=y:
  - kthread\_create\_worker() @ SCHED\_FIFO
  - Less than 32 CPUs, no -rt, boosting enabled
  - Reduce max latency 3 OOM to ~2 ms!

# Real-Time Expedited Grace Periods

- There were a few remaining issues
  - Ran callback invocation at SCHED\_FIFO
  - Which results in unacceptable latency spikes
  - In v6.0, can run only expedited grace periods and priority boosting at SCHED\_FIFO
    - Let callbacks fight it out at SCHED\_OTHER

# Miscellaneous

---

# Miscellaneous

---

- If CONFIG\_PREEMPTION, RCU readers can be preempted
- No longer special restrictions on scheduler use of rcu\_read\_unlock() (Lai Jiangshan)
- RCU now watches almost all of the idle loop (Peter Zijlstra and Thomas Gleixner)
- Merge context tracking and RCU dyntick idle (Frederic Weisbecker)
- Faster Tasks Trace RCU grace periods

# Looking to the Future

---



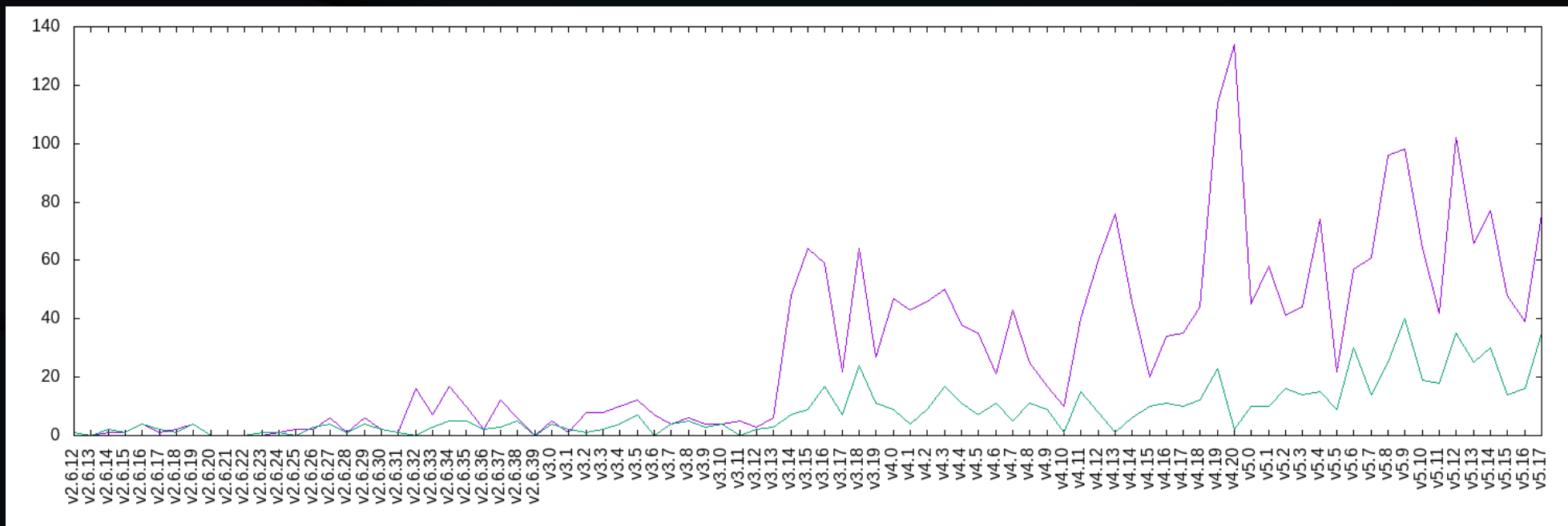
# Possible Future Work

- Even better handling of callback floods (battery)
- Callback-invocation cache locality
- Consolidate RCU Tasks & RCU Tasks Rude
- A very long list of more speculative items
  - For example, de-offload in response to callback overload
- Catch RCU Tasks CPU stall in `synchronize_srcu()`
- Additional NMI-safe SRCU read-side API (`printk()` just this week!)
- Common case: Stuff I don't see coming!!!

# But How Much Future?

---

# Longer-Term Trends



Upper trace: All RCU commits. Lower trace: Non-paulmck RCU commits. What happened at v3.14???

# Summary

---

# Summary

---

- RCU is still under active development:
  - Driven by the needs of its users
- RCU synchronizes in space as well as time
  - But the time and space aspects are deeply intertwined
  - Enables near-zero-cost read-side synchronization
- RCU's dirty little secret:
  - RCU is dead simple, but in order to make good use of it, you must change the way that you think about your problem

# For More Information

- Unraveling RCU Usage Mysteries
  - Part 1: <https://www.linuxfoundation.org/webinars/unraveling-rcu-usage-mysteries/>
  - Part 2: <https://linuxfoundation.org/webinars/unraveling-rcu-usage-mysteries-additional-use-cases/>
- “RCU Usage In the Linux Kernel: One Decade Later”:
  - <http://www.rdrop.com/~paulmck/techreports/survey.2012.09.17a.pdf>
  - <http://www.rdrop.com/~paulmck/techreports/RCUUsage.2013.02.24a.pdf>
  - 2020 update: <https://dl.acm.org/doi/10.1145/3421473.3421481>
- “Structured Deferral: Synchronization via Procrastination”: <http://doi.acm.org/10.1145/2488364.2488549>
- Linux-kernel RCU API, 2019 Edition: <https://lwn.net/Articles/777036/>
- “Stupid RCU Tricks: So you want to torture RCU?”: <https://paulmck.livejournal.com/61432.html>
- Documentation/RCU/\* in kernel source
- “Is Parallel Programming Hard, And, If So, What Can You Do About It?”, “Deferred Processing” chapter: <https://mirrors.edge.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html>
- Folly-library RCU implementation (also C-language user-space RCU)
- Large piles of information: <http://www.rdrop.com/~paulmck/RCU/>

# Questions?

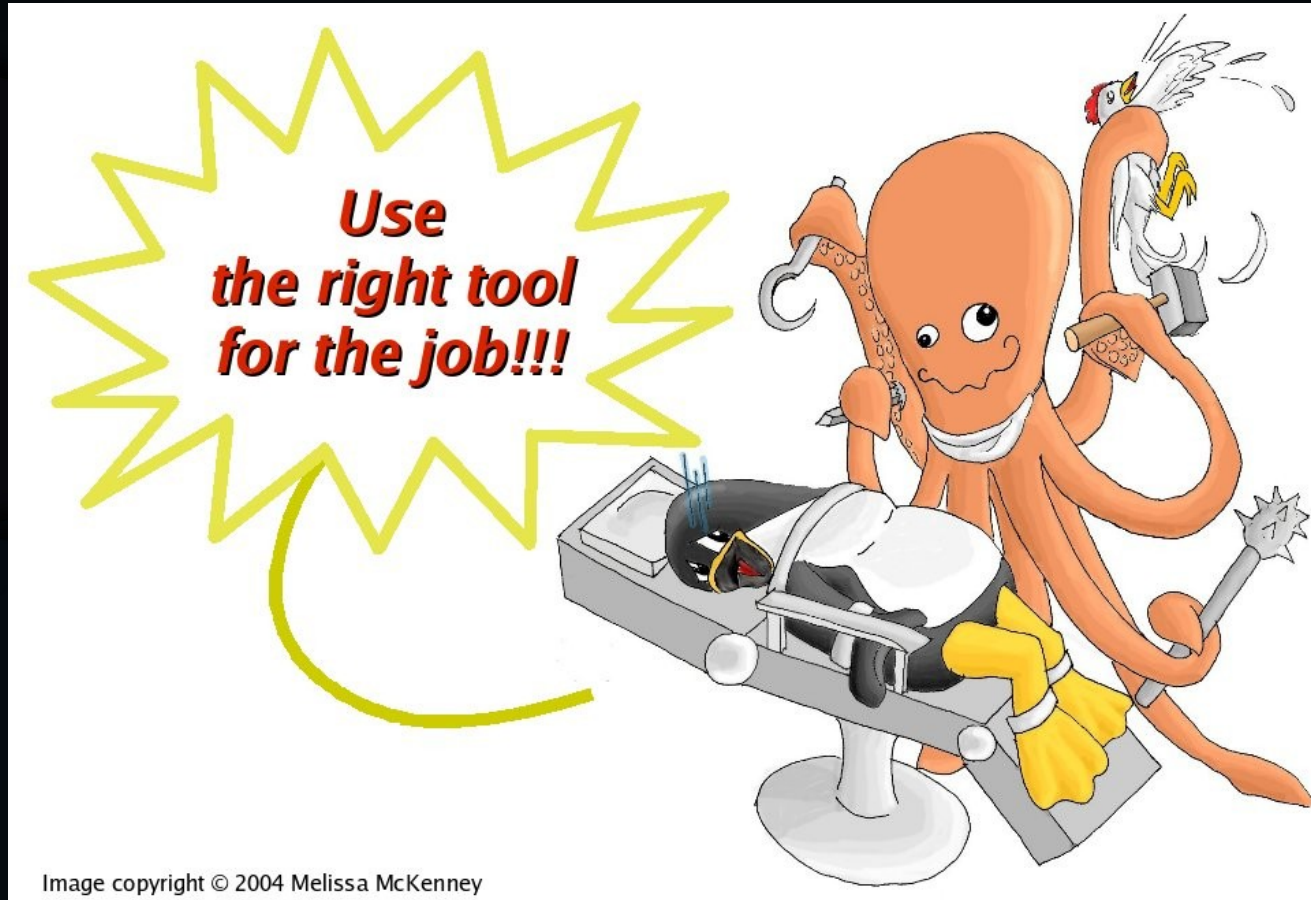


Image copyright © 2004 Melissa McKenney