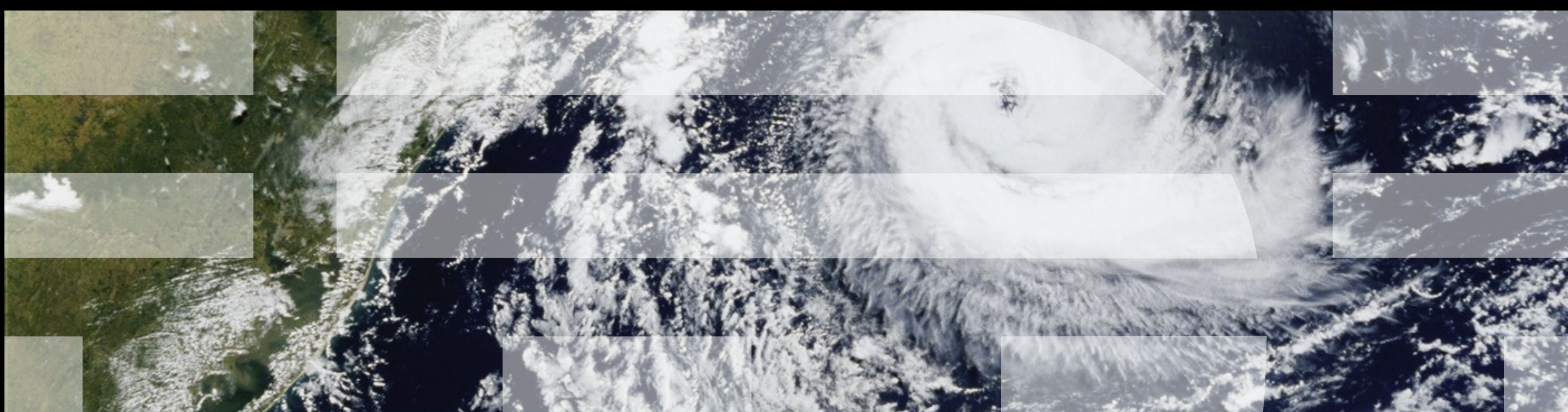
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center
Member, IBM Academy of Technology

Systor2019, June 5, 2019

A Critical RCU Safety Property Is...

Ease of Use!!!

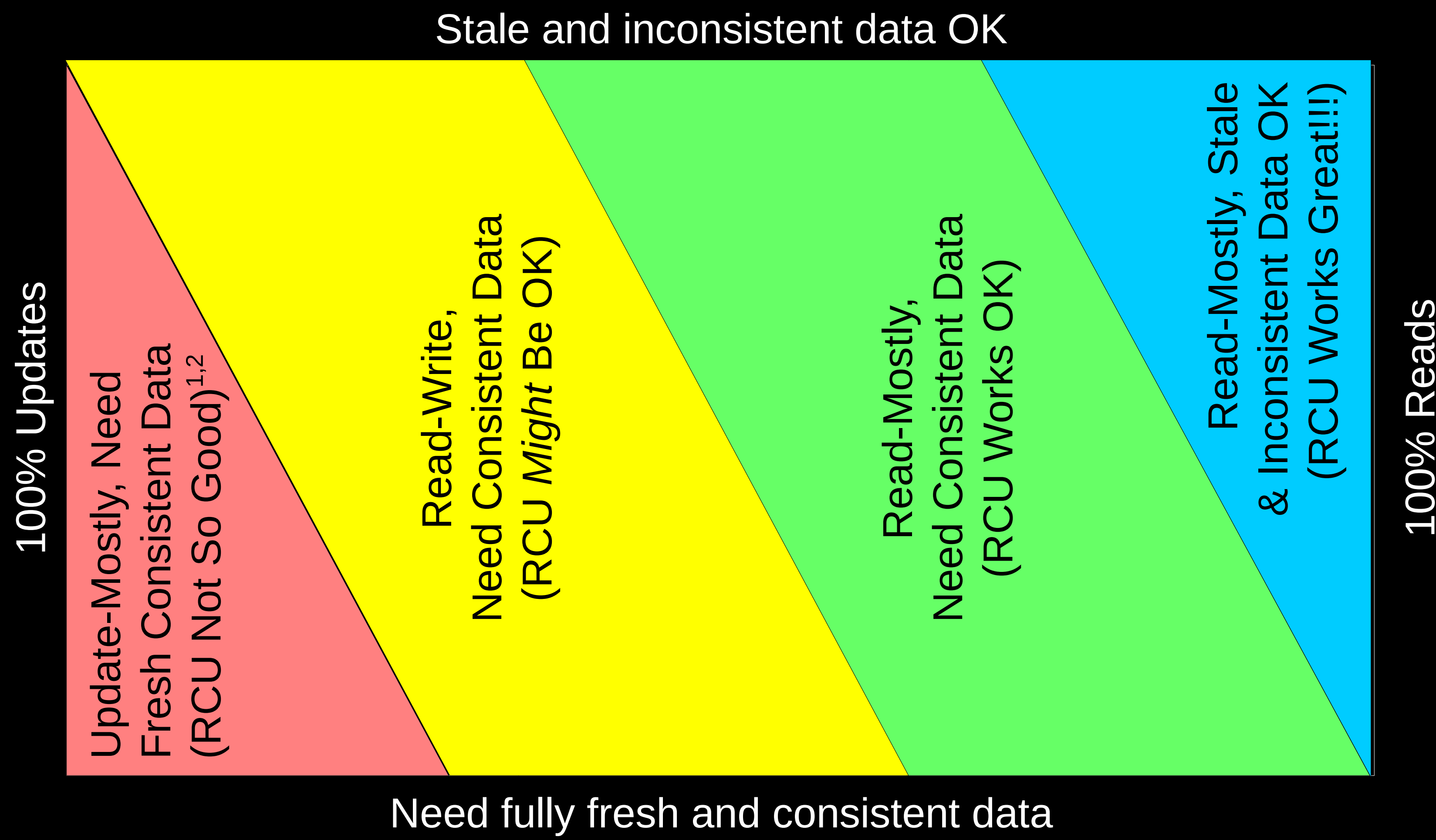


Overview

- Quick RCU overview
- Isn't RCU a bit low-level to be involved in an exploit?
- What was the real problem?
- What would a fix even look like???
- Possible solutions
- Other consequences
- Summary

Quick RCU Overview

Primary Use Case: Read-Mostly Linked LIsts



1. RCU provides ABA protection for update-friendly mechanisms
2. RCU provides bounded wait-free read-side primitives for real-time use

Summary of RCU's Deep Core Primitives

- **Read-side primitives:**

  `rcu_read_lock()`
  – Start an RCU read-side critical section

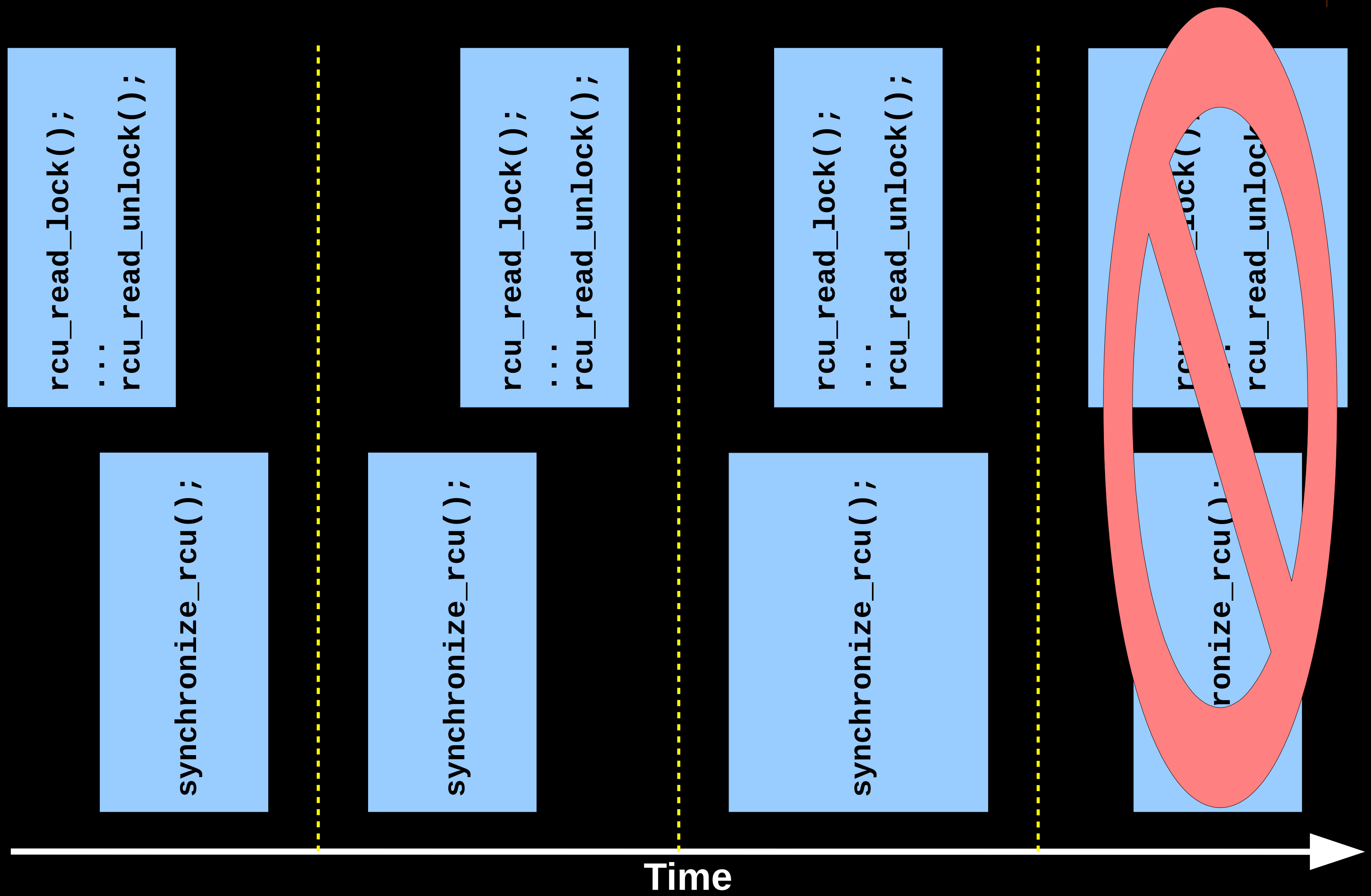
  `rcu_read_unlock()`
  – End an RCU read-side critical section

- **Update-side primitive**

  `void synchronize_rcu(void)`
  – Wait for pre-existing RCU read-side critical sections to complete

The RCU API, 2019 Edition: https://lwn.net/Articles/777036/

RCU Execution Constraints



Toy Implementation of QSBR-Style RCU: 11 Lines of Code, Full Read-Side Performance!!!

- **Read-side primitives:**

```
#define rcu_read_lock()     __asm__ __volatile__("": : :"memory")
#define rcu_read_unlock()   __asm__ __volatile__("": : :"memory")
#define rcu_dereference(p) READ_ONCE(p)
```

- **Update-side primitives**

```
#define rcu_assign_pointer(p, v) smp_store_release(&(p), (v))
void synchronize_rcu(void)  /* PREEMPT=n Linux kernel. */
{
    int cpu;

    for_each_online_cpu(cpu)
        sched_setaffinity(current->pid, cpumask_of(cpu));
}
```

Only 9 of which are needed on sequentially consistent systems...
And some people still insist that RCU is complicated... ;-)

Linux Kernel RCU Has More Than 11 Lines Because:

- Systems with 1000s of CPUs
- Sub-20-microsecond real-time response requirements
- CPUs can come and go (“CPU hotplug”)
- If you disturb idle CPUs. you enrage low-power embedded folks
- Forward progress requirements: callbacks, network DoS attacks
- RCU grace periods must provide extremely strong ordering
- RCU uses the scheduler, and the scheduler uses RCU
- Firmware sometimes lies about the number and age of CPUs
- RCU must work during early boot, even before initialization
- Preemption can happen, even when interrupts are disabled (vCPUs!)
- RCU should identify errors in client code (maintainer self-defense!)

Here is Your Elegant Synchronization Mechanism:

Photo by "Golden Trvs Gol twister", CC by SA 3.0

Here is Your Elegant Synchronization Mechanism Equipped To Survive In The Linux Kernel:

Photo by Луц Фишер-Лампрехт, CC by SA 3.0

Linux Kernel RCU Has More Than 11 Lines Because:

- Systems with 1000s of CPUs
- Sub-20-microsecond real-time response requirements
- CPUs can come and go (“CPU hotplug”)
- If you disturb idle CPUs. you enrage low-power embedded folks
- Forward progress requirements: callbacks, network DoS attacks
- RCU grace periods must provide extremely strong ordering
- RCU uses the scheduler, and the scheduler uses RCU
- Firmware sometimes lies about the number and age of CPUs
- RCU must work during early boot, even before initialization
- Preemption can happen, even when interrupts are disabled (vCPUs!)
- RCU should identify errors in client code (maintainer self-defense!)
- *Multiple “flavors” of RCU*

Multiple “Flavors” of RCU

- Generic use cases:
```
rcu_read_lock()
rcu_read_unlock()
synchronize_rcu()
```
This flavor reviewed on past few slides

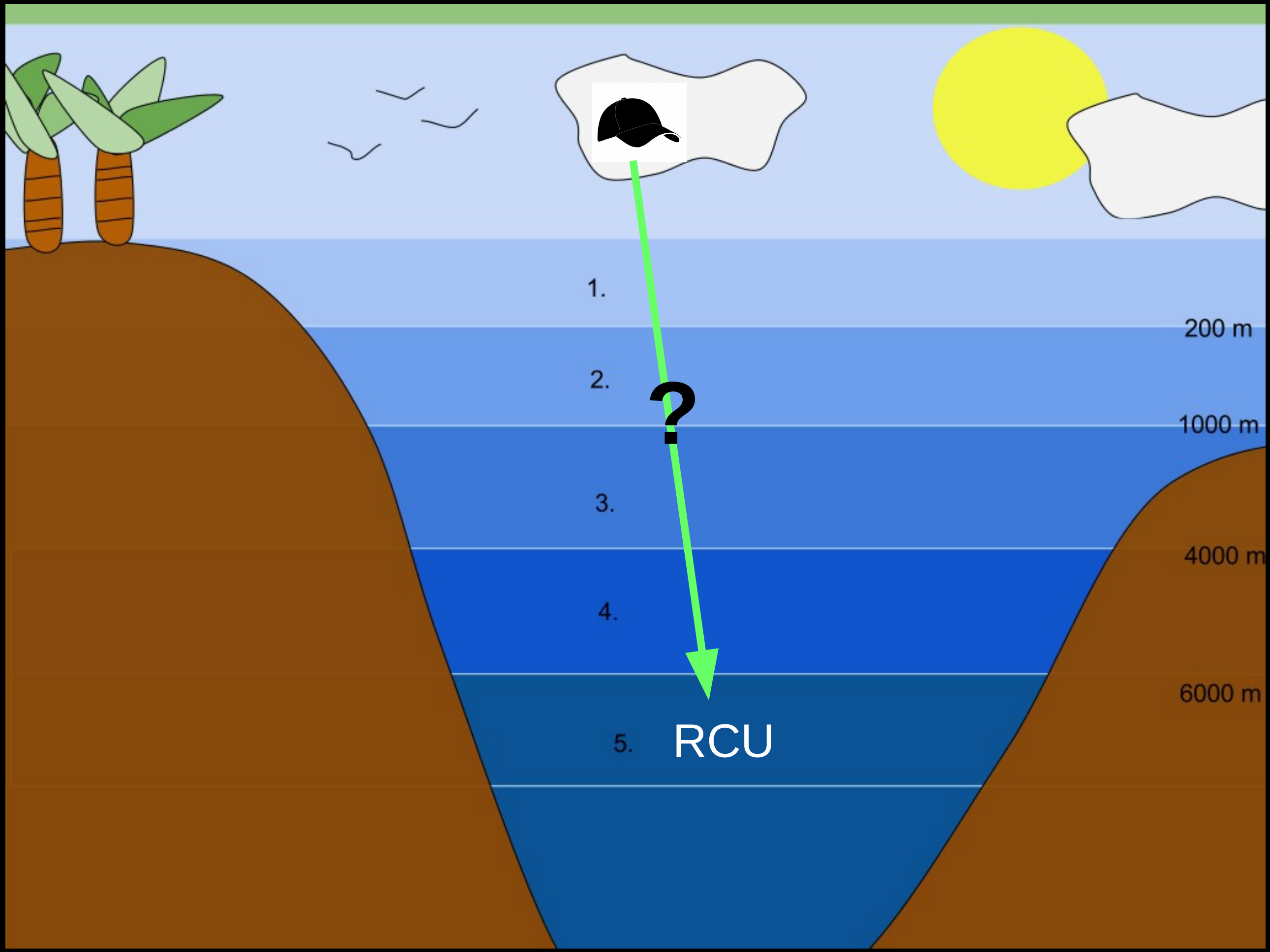
- Code subject to denial-of-service attacks:
```
rcu_read_lock_bh()
rcu_read_unlock_bh()
synchronize_rcu_bh()
```

- Interactions with non-realtime preempt-disable regions:
```
rcu_read_lock_sched()
rcu_read_unlock_sched()
synchronize_sched()
```

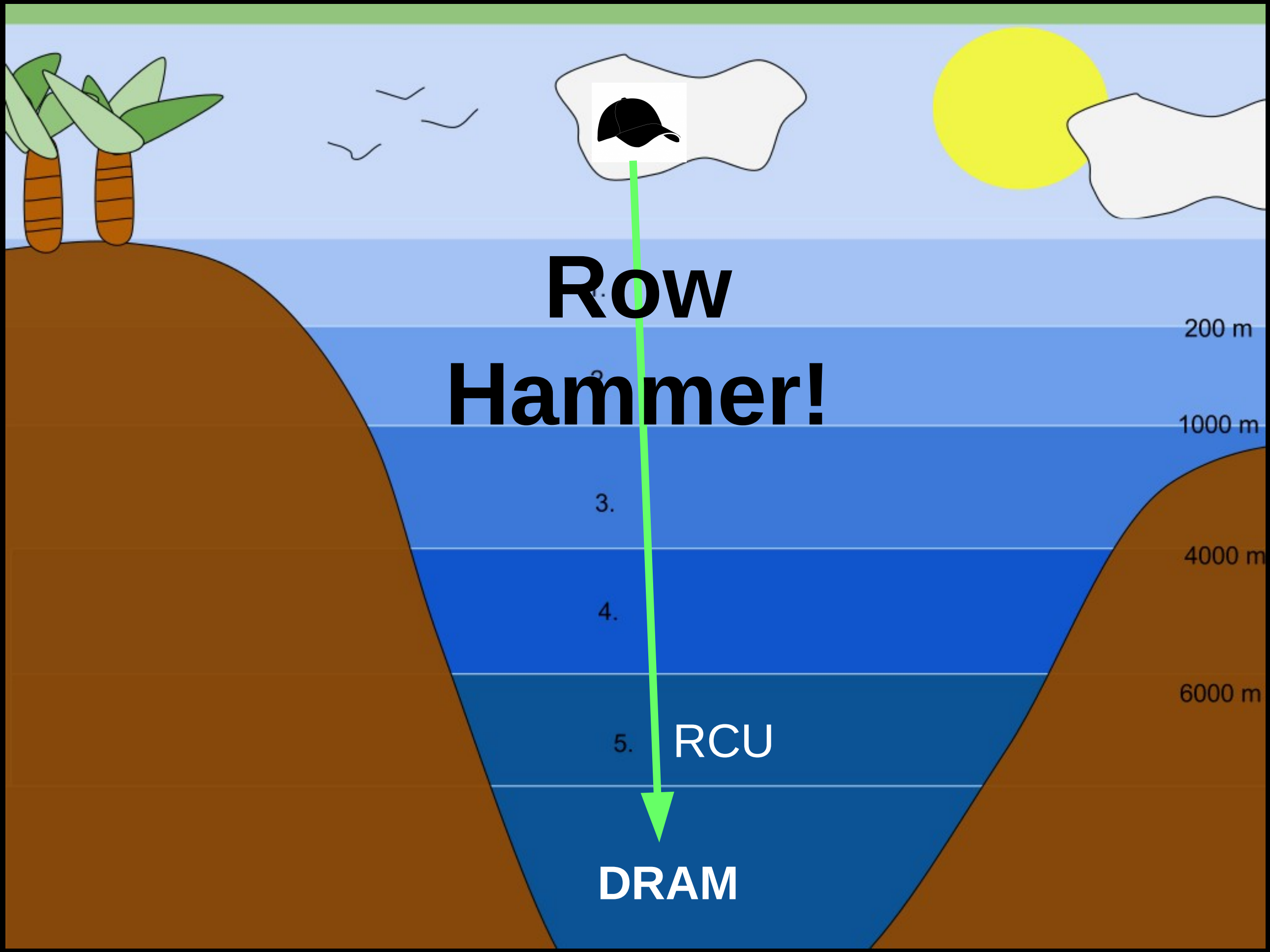
There is a Lot More to RCU Implementation and Use

- RCU has been used in production for more than 25 years
  - And has antecedents going back to 1980 or perhaps even 1963
- There is therefore a huge body of RCU-related practice:
  - Simple/scalable/real-time/energy-efficient/... implementations
  - Combined use of RCU with locking, sequence locking, transactional memory, non-blocking synchronization, …
  - Complex atomic-to-readers updates via transactional memory
  - Complex atomic-to-readers updates via Issaquah Challenge
  - Interactions with hardware features (interrupts, complex instructions...)
  - Formal semantics from several viewpoints
- But the preceding slides do provide a few RCU basics
  - The paper goes into more detail and contains citations

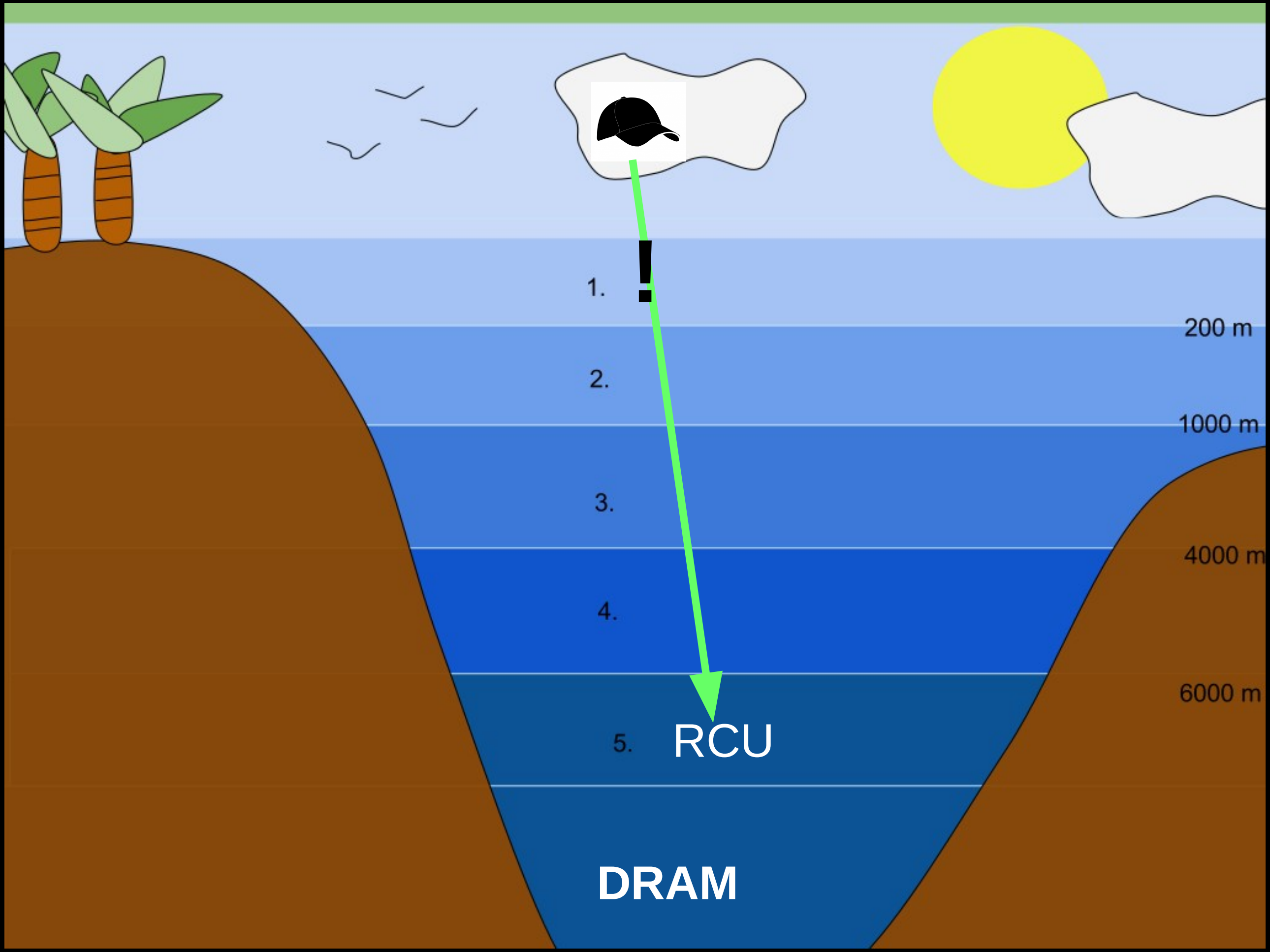
Isn't RCU a Bit Low-Level to be Involved in a Exploit?



Isn't RCU a Bit Low-Level to be Involved in a CVE?



If Black Hats Can Hit DRAM (Saying Nothing of Firmware), They Can Hit RCU!!!



This is No Longer Strictly Theoretical...

Minding My Own Business When This Email Arrived

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 <bcrl@kvack.org>, security@kernel.org, Al Viro <viro@zeniv.linux.org.uk>
Subject: Re: AIO locking bug in lookup_ioctx()
From linus971@gmail.com Sat Mar 3 17:54:39 2018

[Add the cc too for various reasons]

Linus Torvalds

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

[. . .]

security@kernel.org

> 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.

Linus

A Prototype RCU-Usage Fix, And Then This Email

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 <bcrl@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

[. .

Linus Torvalds

Which is the topic of this talk!

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

I repeat: I really love the traditional RCU, but I *despise* there
are a million different and confusing versions of it. It clearly
causes real pro

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?

Linus

What Was The Real Problem???

What Was The Real Problem???
Abuse of RCU...

```
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. */
}
```

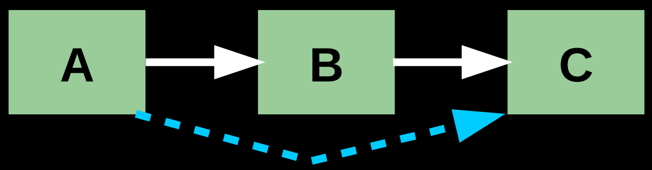
What Was The Real Problem???

```
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. */
}
```

This is about as healthy for your kernel as acquiring the wrong lock!!!
Or accessing the wrong variable.
Or calling the wrong function.
Or...

Why is This a Problem??? Pictorial Form...



`rcu_read_lock_sched();`

`list_for_each_entry_rcu(...)`
Get reference to B

`list_del_rcu(B);`

`synchronize_rcu();`

`rcu_read_lock()` in effect?
No, so report quiescent state!

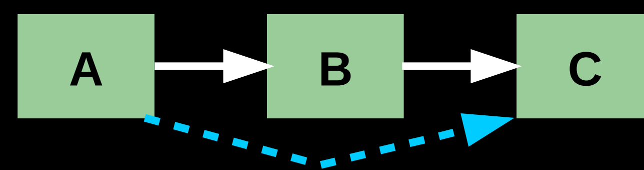
`kfree(B);`

Still using B!!!

`rcu_read_unlock_sched()`

What are developers supposed to do instead?

What Are Developers Supposed to do Instead?



`rcu_read_lock();`

`list_for_each_entry_rcu(...)`
Get reference to B

`list_del_rcu(B);`

`rcu_read_lock()` in effect?
Yes, so no quiescent state.

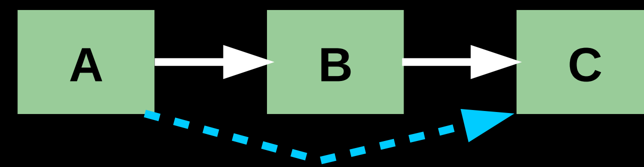
`synchronize_rcu();`

Still using B,
but that's OK!

`rcu_read_unlock()`

`kfree(B);`

Or, Alternatively, Adjust the Updater:




```
rcu_read_lock_sched();
```

```
list_for_each_entry_rcu(...)
```
Get reference to B

Preemption disabled?
Yes, so no quiescent state.

Still using B,
but that's OK!

```
rcu_read_unlock_sched();
```

```
list_del_rcu(B);
```

```
synchronize_sched();
```

```
kfree(B);
```

Consistency is Required, But That is a Problem!

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

```
synchronize_rcu();
```

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

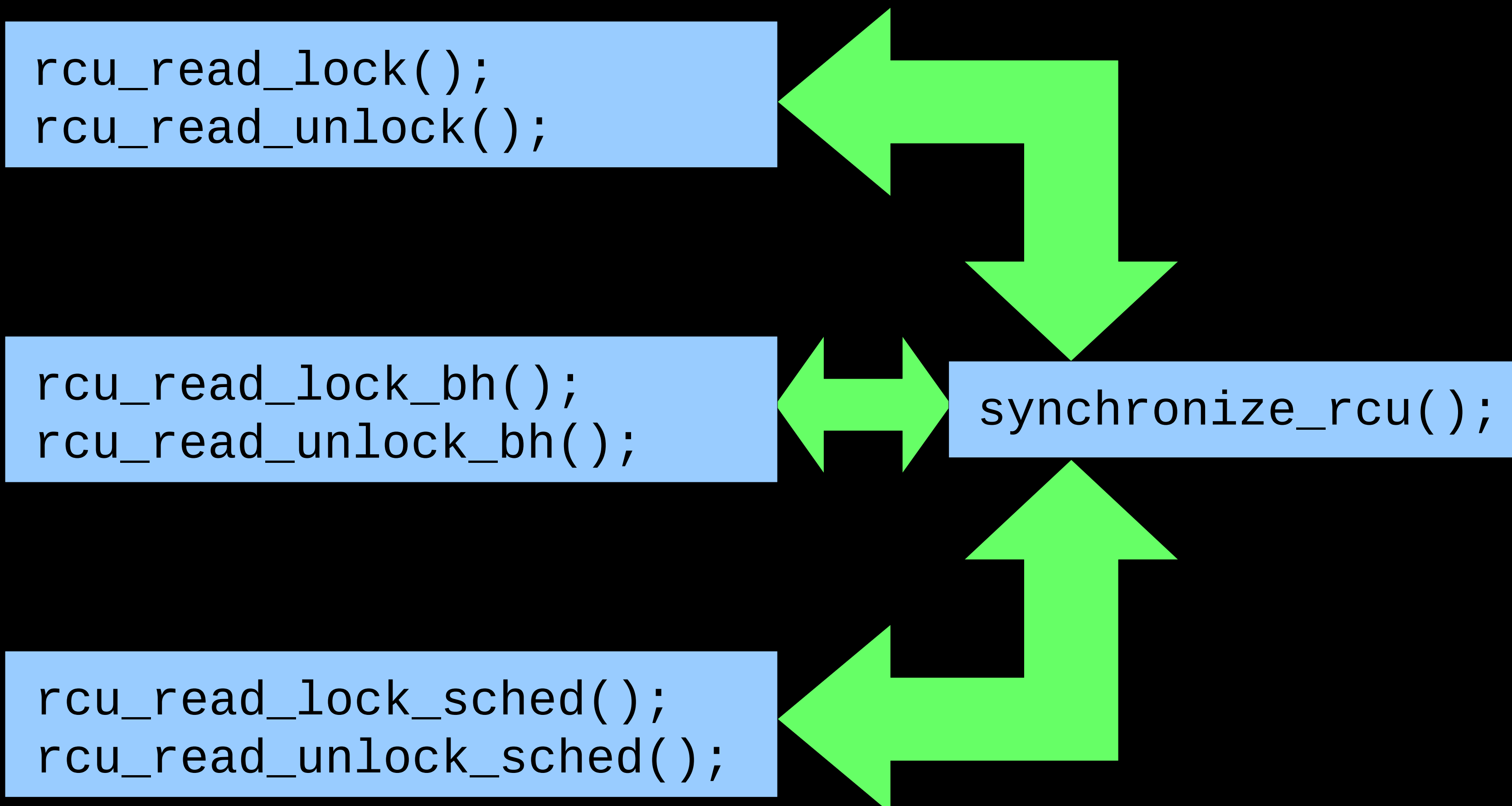
```
synchronize_rcu_bh();
```

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

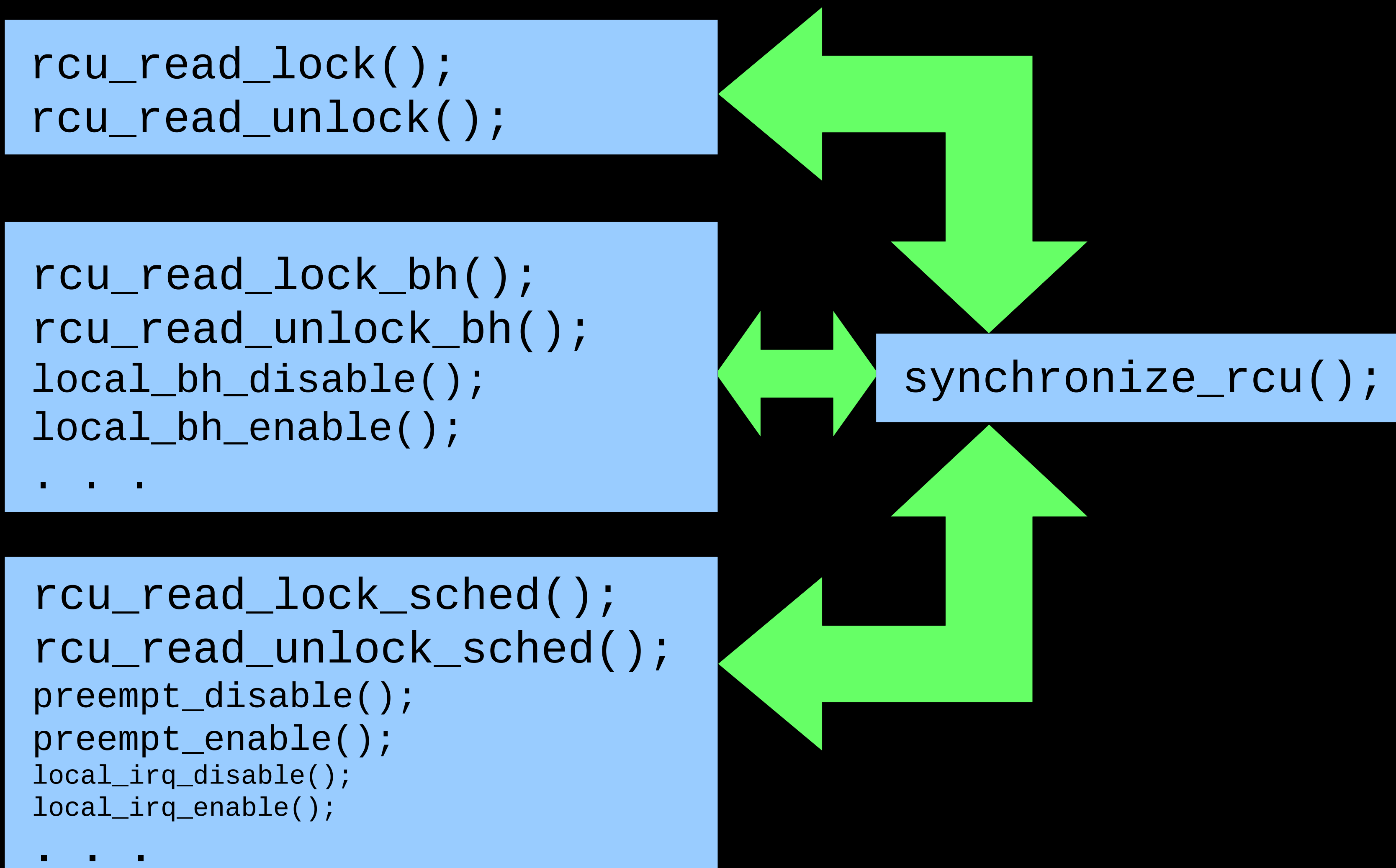
```
synchronize_sched();
```

To err is human...
Plus userspace controls content of much kernel data!!!

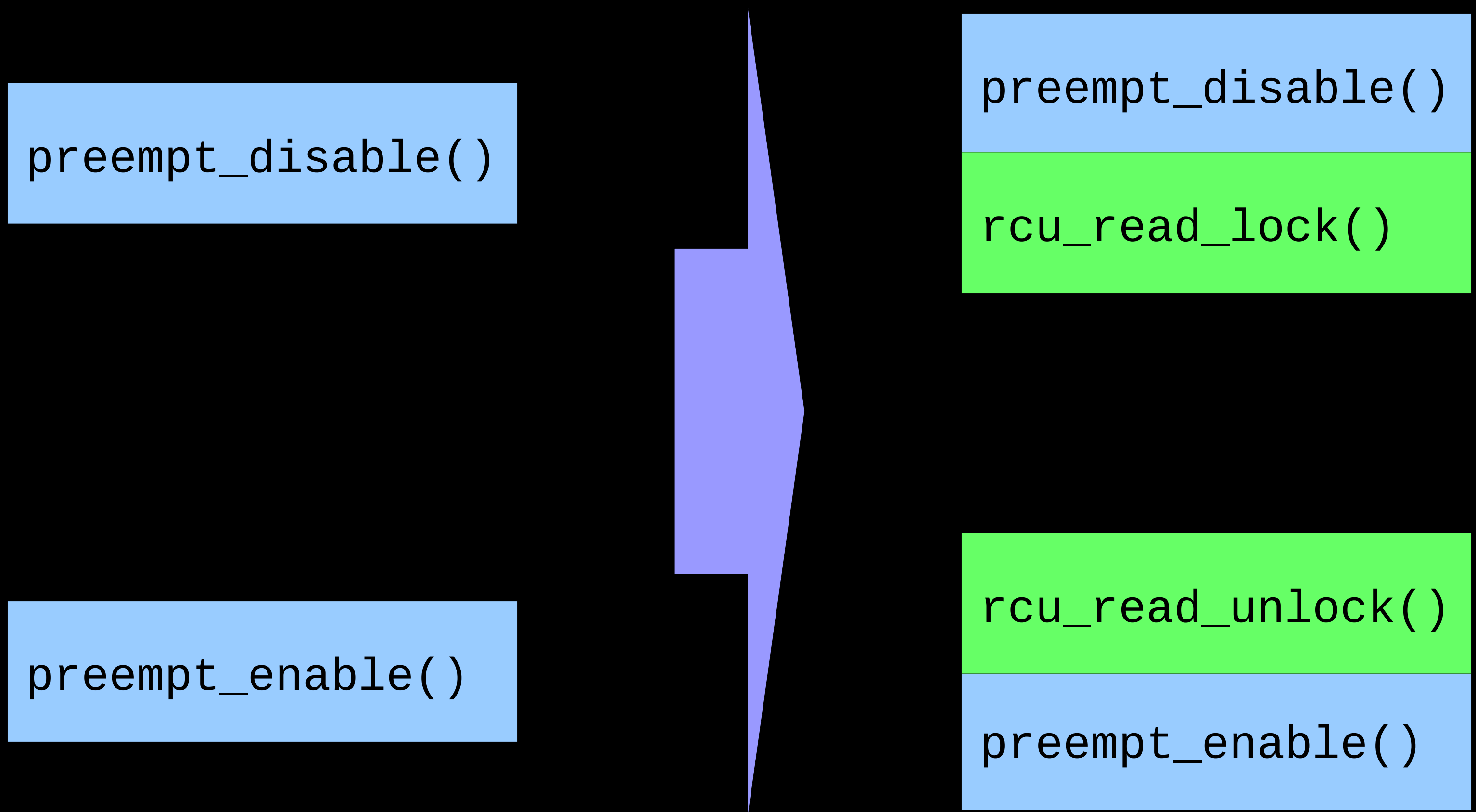
Desired State From Usability/Security Viewpoint:



Desired State From Usability/Security Viewpoint Except That Things Are Never Quite That Simple...



Possible Solution: Add Explicit RCU Readers
Example: preempt_disable() and preempt_enable()



For more detail, see paper and linux.conf.au presentation:
Slides: http://www.rdrop.com/users/paulmck/RCU/cve.2019.01.23e.pdf
Video: https://www.youtube.com/watch?v=hZX1aokdNiY

Possible Solution: Add Explicit RCU Readers
Too Bad About All That Fastpath Assembly Code...

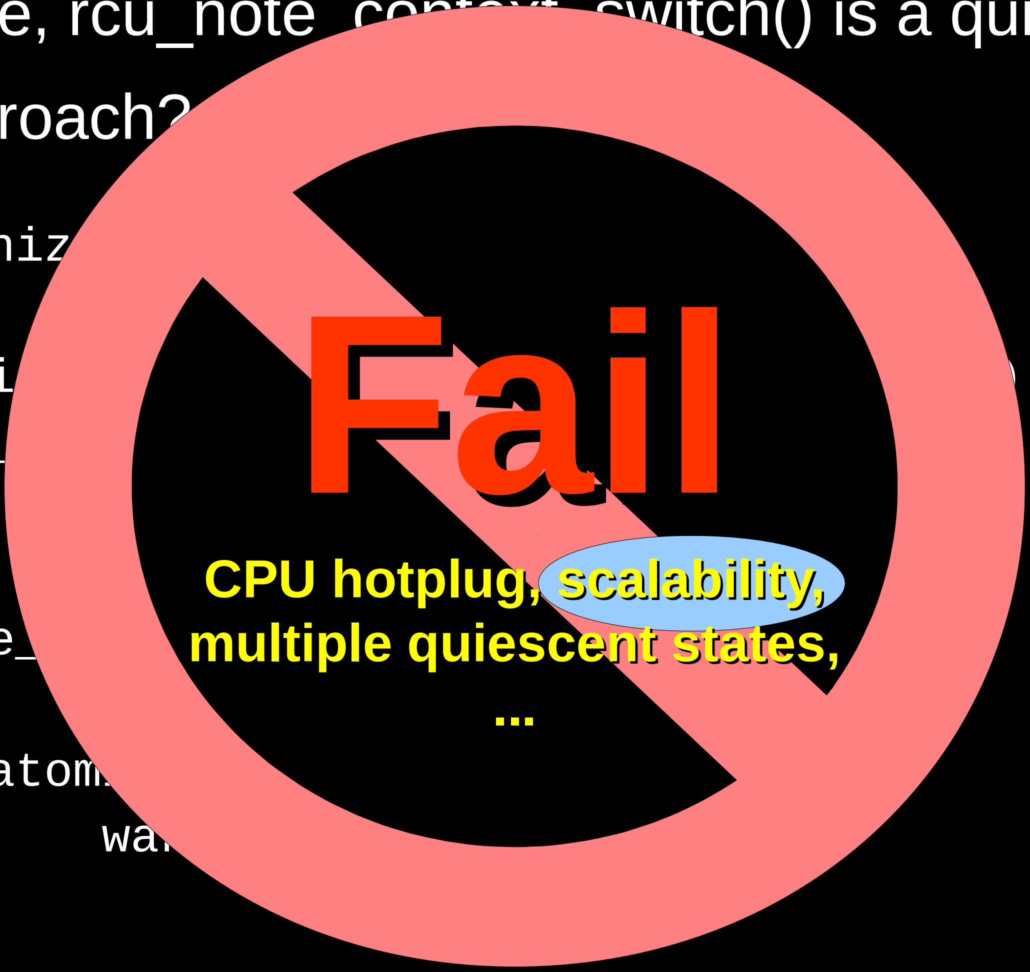
- Try easy approaches ... readers:
  - Make local_bh_di... before returning and local_bh_enab... being called
  - Make preemp... ore returning and preempt_en... ng called
  - Make local_... e returning and local_irq_e... g called
    - And sam...
- How many p...
- So test it first: ... ement counter and instead of r... nt same counter
  - Complain if counter ... ng is enabled

Just Globally Count Deferral Reasons!

- For example, rcu_note_context_switch() is a quiescent state
- Simple approach?

```
void synchroniz
{
        atomi                                          );
        wait_
}

void rcu_note_
{
        If (atom
                wa
}
```



First bug report against RCU on 512-CPU system in 2004...

Defer Reporting of Quiescent States at Reader End

Debuggi… s writing the cod… refore, if you… ly as poss… n, not s… t.

Fail

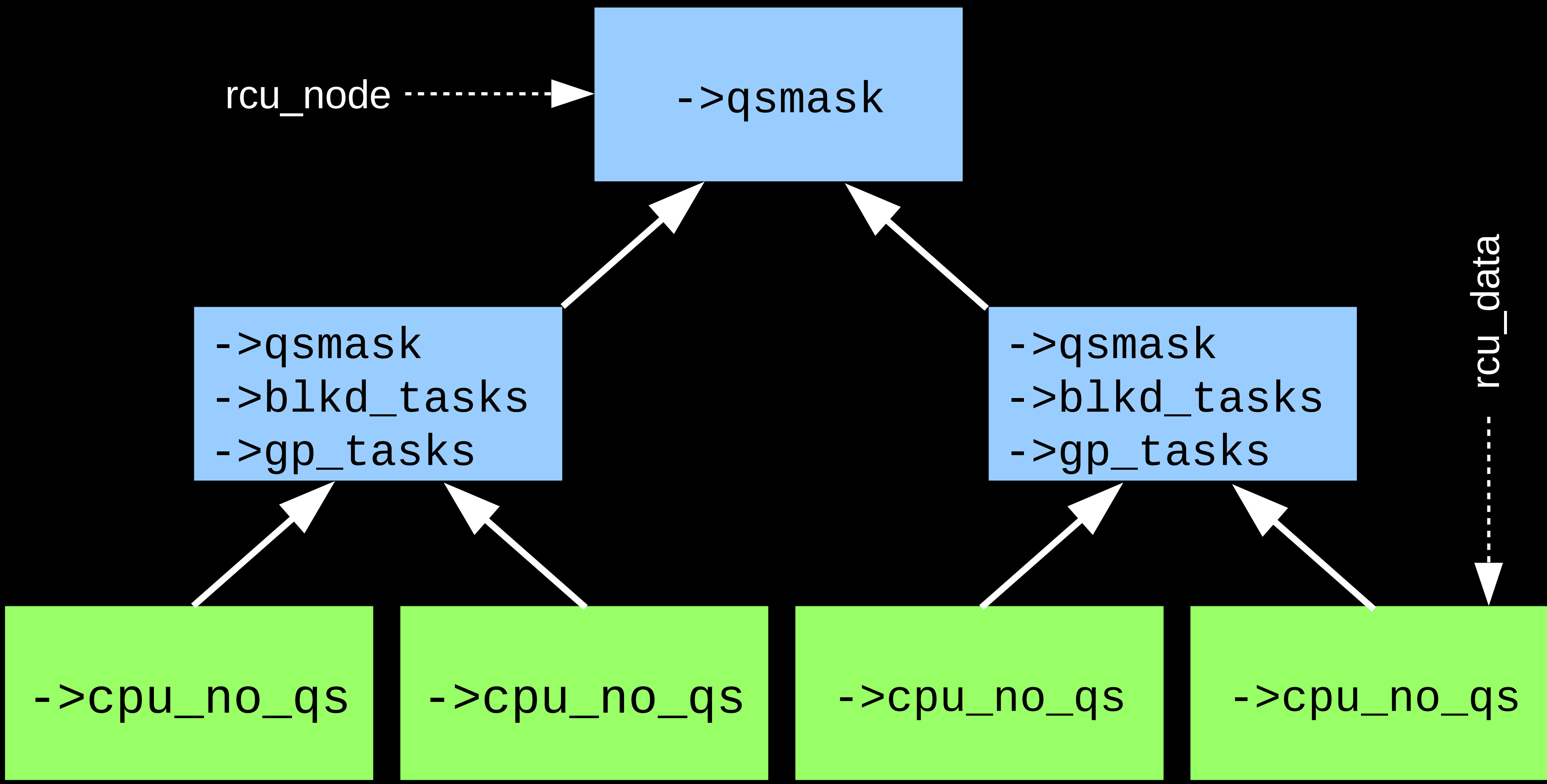
RCU read-side critical sections linked by preempt-disable... Excessive complexity!!!

For that matter, am I even smart enough to test it???

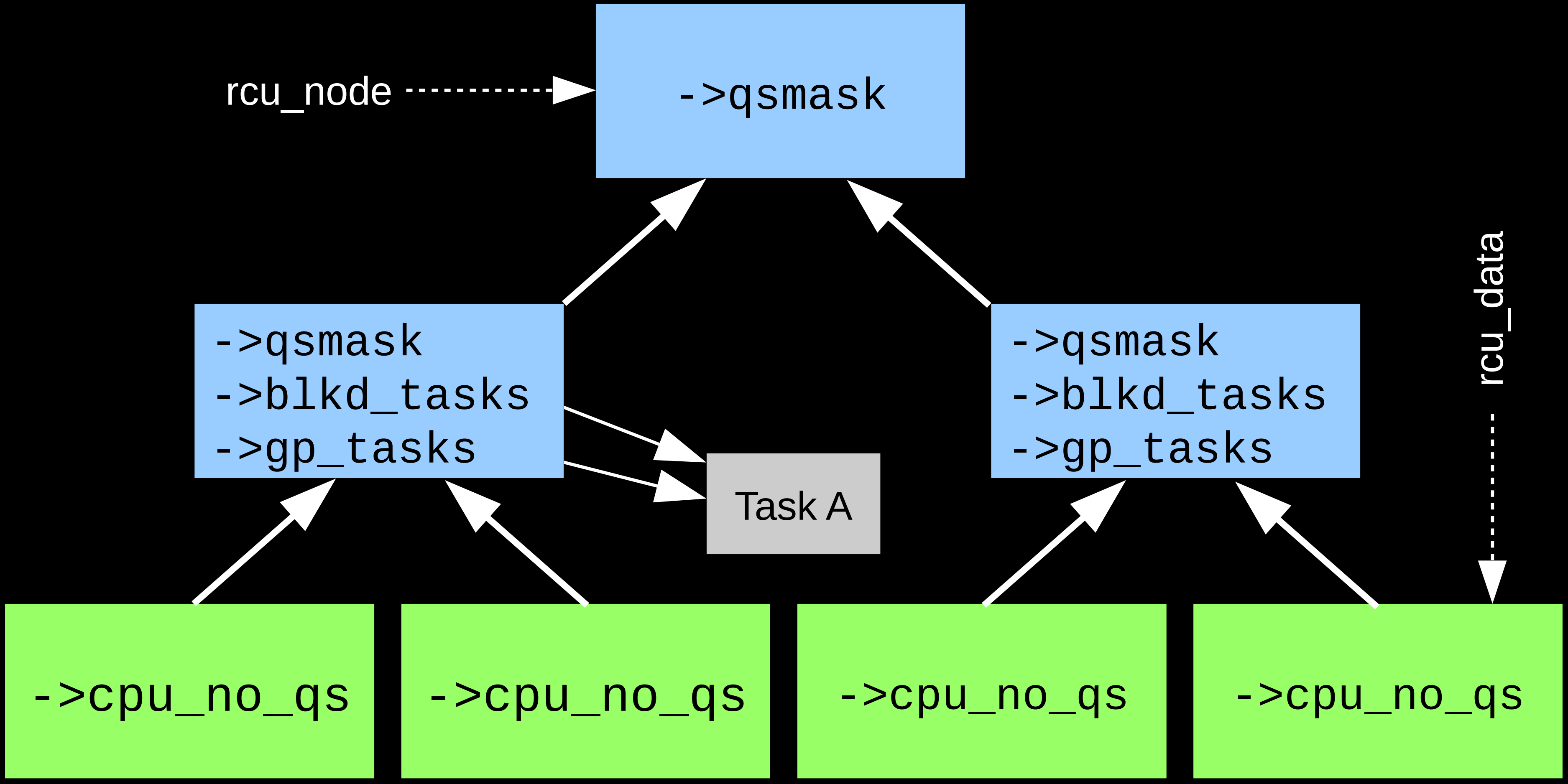
Back to the drawing board...

Possible Solution: Defer rcu_read_unlock() Dequeue

Preempted Tasks Queued on Leaf rcu_node Structure
Task A Preempted, Blocks Current Grace Period

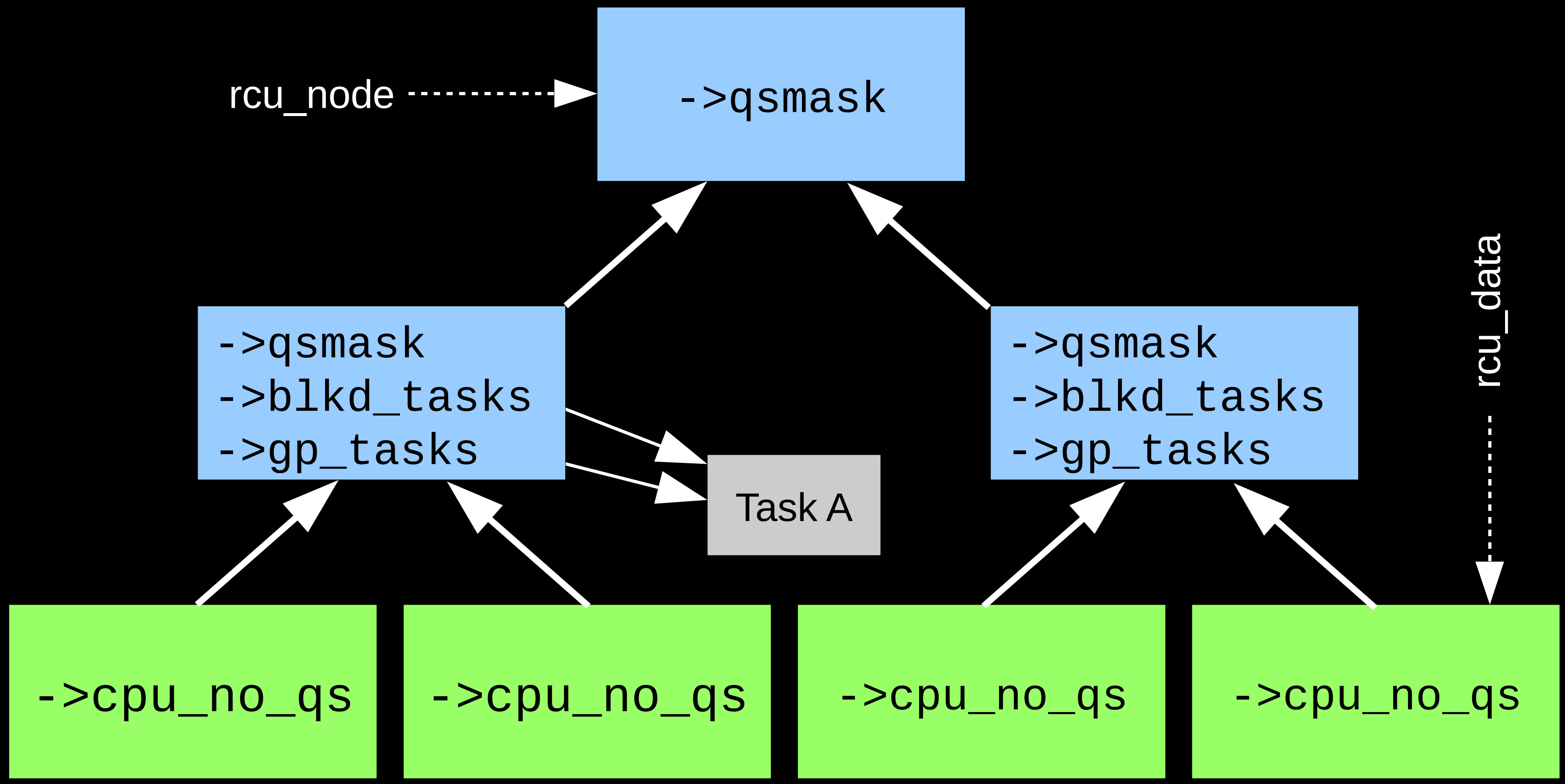


Preempted Tasks Queued on Leaf rcu_node Structure Task A Preempted, Blocks Current Grace Period

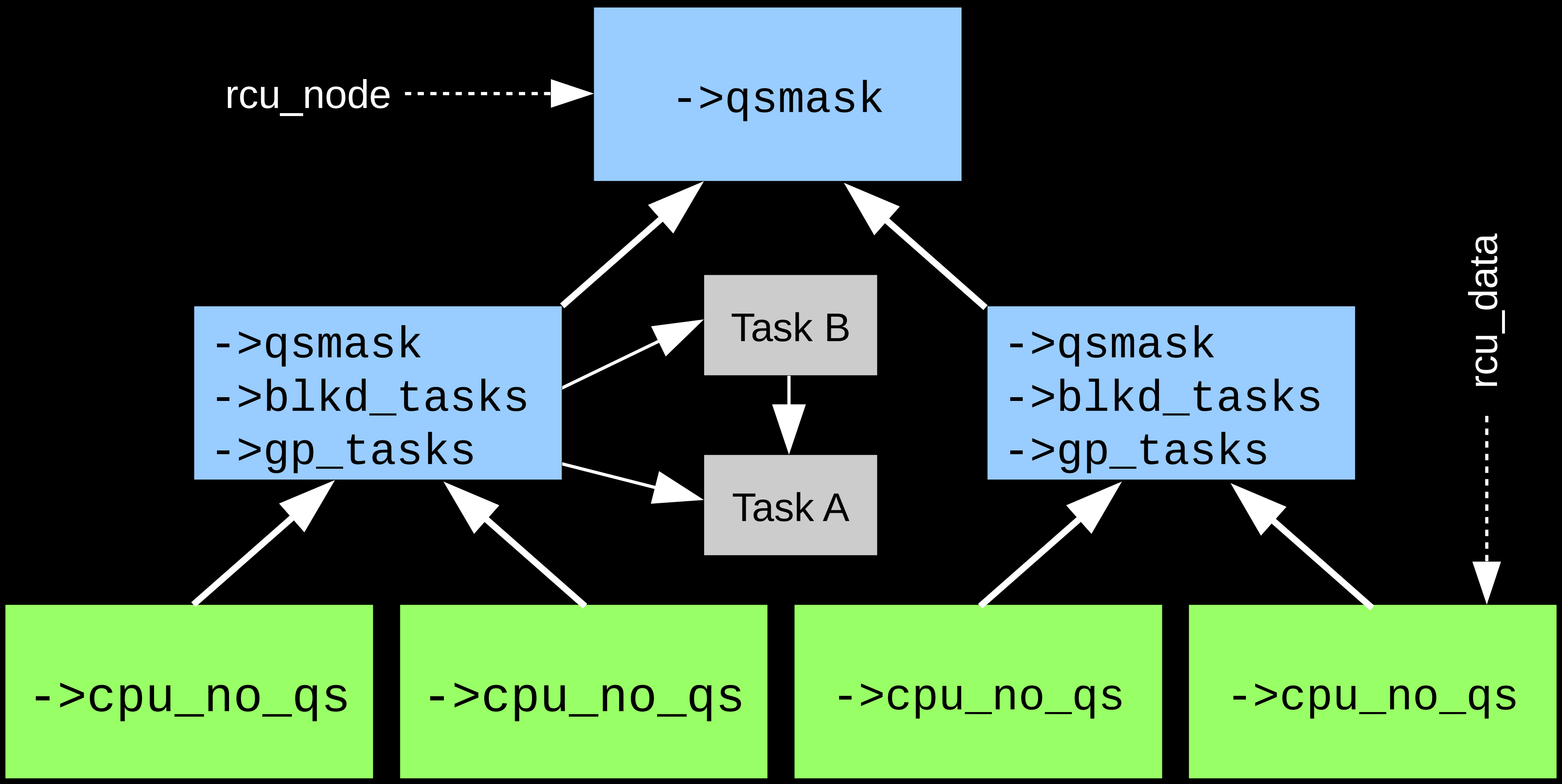


CPU switches to Task B

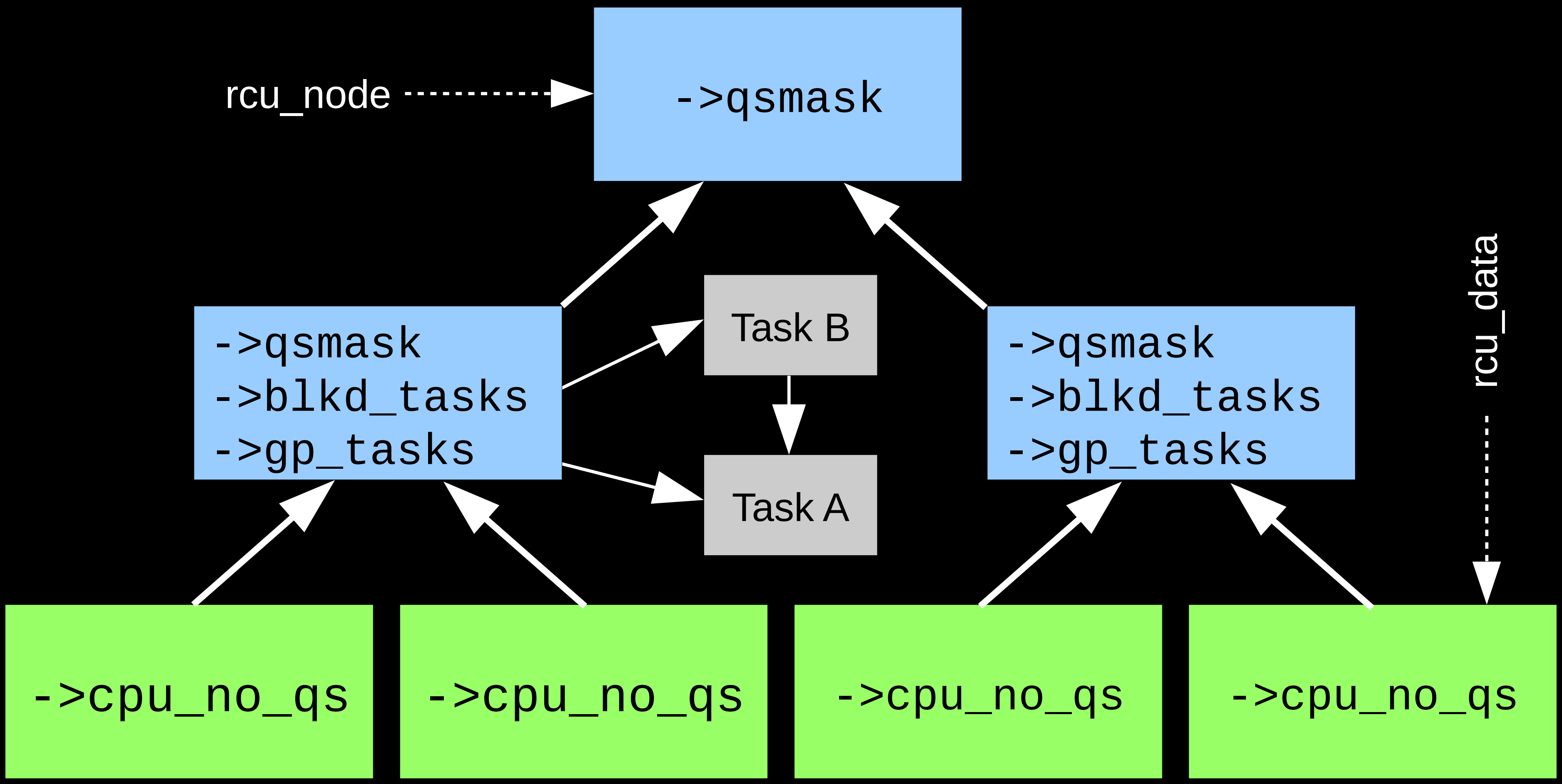
Preempted Tasks Queued on Leaf rcu_node Structure
Task B's priority is lowered, Task A resumes



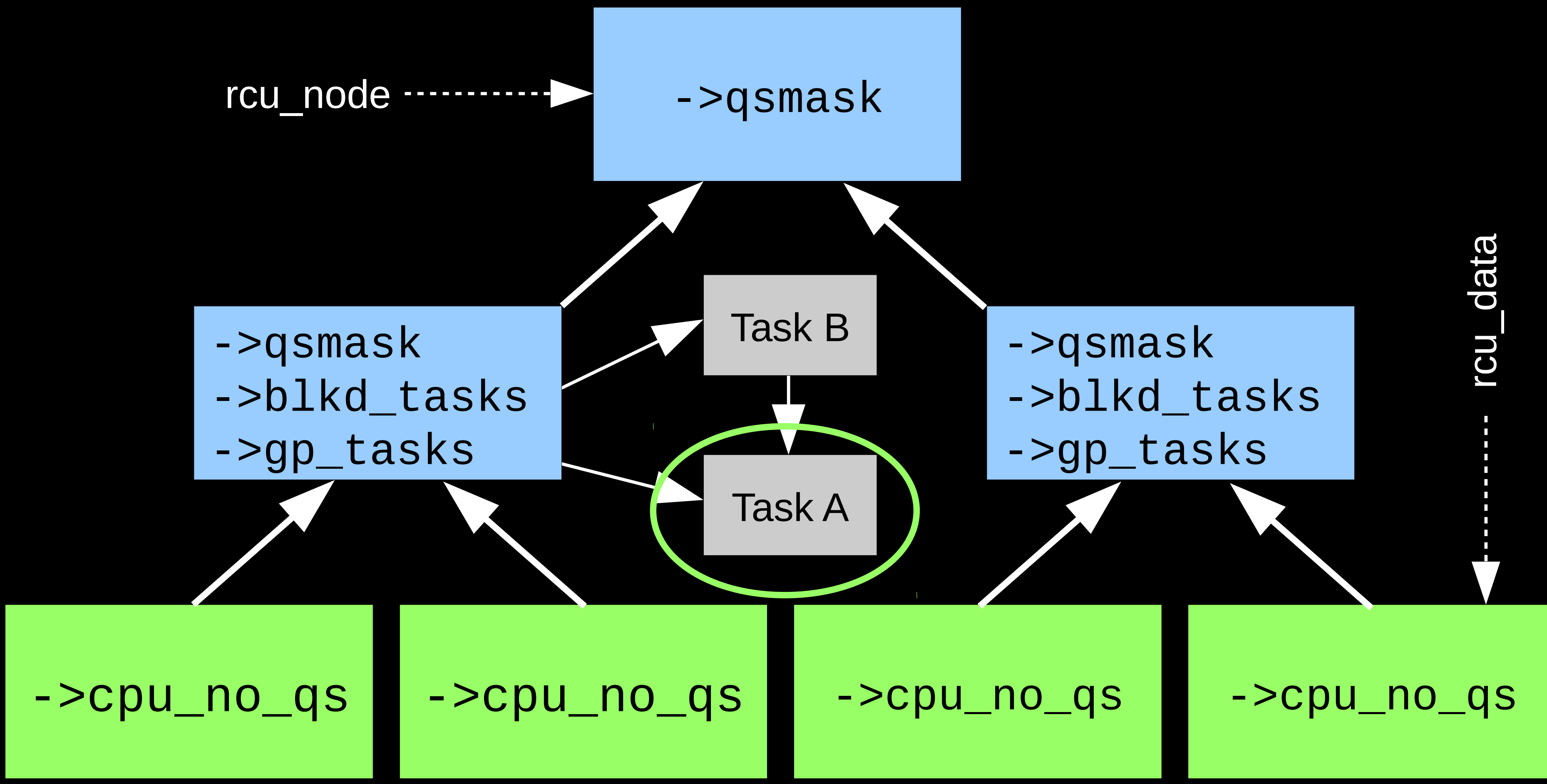
Preempted Tasks Queued on Leaf rcu_node Structure Task A Blocks Current Grace Period, Task B Does Not



Preempted Tasks Queued on Leaf rcu_node Structure Task A Executes rcu_read_unlock()

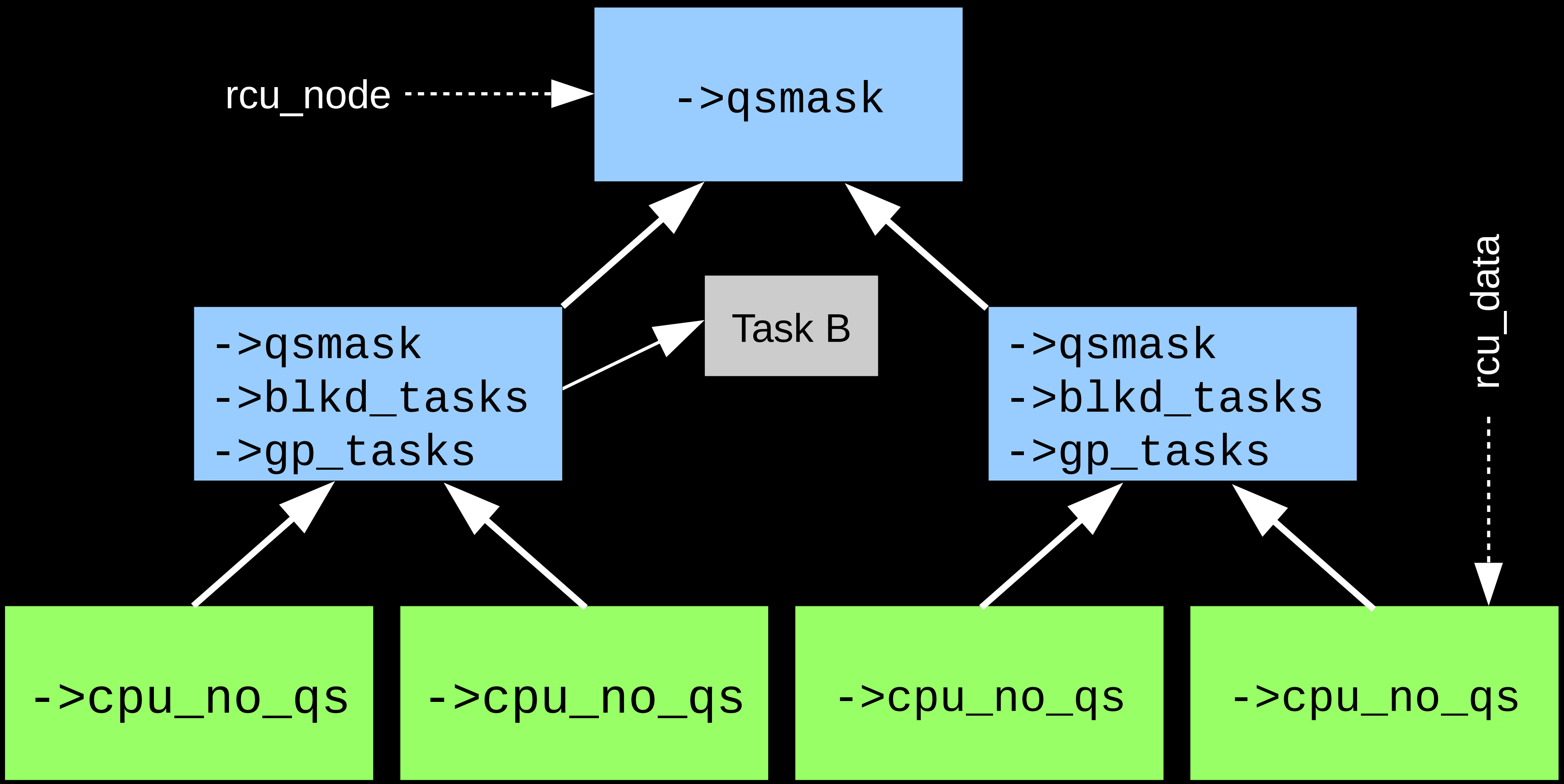


Preempted Tasks Queued on Leaf rcu_node Structure Task A No Longer Blocks Current Grace Period



Task A must remove itself from ->blkd_tasks and update ->gp_tasks
But there is no next task, so set ->gp_tasks to NULL

Preempted Tasks Queued on Leaf rcu_node Structure Grace Period No Longer Blocked by Preempted Task



Task A has removed itself from ->blkd_tasks and updated ->gp_tasks

Which Breaks This Larger Example!!!

```
rcu_read_lock();
do_something_1();
preempt_disable();
do_something_2();
rcu_read_unlock();
do_something_3();
rcu_read_lock();
do_something_4();
preempt_enable();
do_something_5();
rcu_read_unlock();
```

This rcu_read_lock() must block the grace period, but won't because of the prior rcu_read_unlock()!!!

Which Breaks This Larger Example!!!

```
rcu_read_lock();
do_something_1();
preempt_disable();
do_something_2();
rcu_read_unlock();
do_something_3();
rcu_read_lock();
do_something_4();
preempt_enable();
do_something_5();
rcu_read_unlock();
```

This rcu_read_lock() must block the grace period, but won't because of the prior rcu_read_unlock()!!!

Should the prior rcu_read_unlock() avoid dequeuing based on preemption having been disabled?

How Would Deferring Dequeuing Change Quiescent State Handling?

- Quiescent state:
  - If CPU's rcu_data structure's ->cpu_no_qs flag is set, clear it and proceed to leaf rcu_node
  - If CPU's bit in leaf rcu_node structure's ->qsmask is set, clear it and if all bits are clear and if ->gp_tasks is NULL, proceed to root rcu_node
  - If corresponding bit in root rcu_node's ->qsmask is set, clear it, and if all bits are now clear, end of grace period!
- "Special" situation in rcu_read_unlock():
  - ***Only if fully enabled,*** remove self from ->blkd_tasks, adjust ->gp_tasks if references self
  - If ->gp_tasks now NULL and all ->qsmask bits are clear, proceed to root rcu_node and handle it as above
- ***Periodically check for deferred quiescent states***
  - ***Dequeue task, if needed, and report deferred quiescent state***

Does This Really Work on That Example???

```
rcu_read_lock();
do_something_1();
preempt_disable();
do_something_2();
rcu_read_unlock();
do_something_3();
rcu_read_lock();
do_something_4();
preempt_enable();
do_something_5();
rcu_read_unlock();
```

Preemption disabled, so don't dequeue the task...

… which means that the task is still queued, and thus already blocking the grace period!!!

One big reader, as required!!!

Defer rcu_read_unlock() Current-Task Dequeue (Part of a Page, Down from 8+ to 3 Pages Total!!!)

```
static void rcu_read_unlock_special(struct task_struct *t)
{
	unsigned long flags;
	bool preempt_was_disabled = !!(preempt_count() && ~HARDIRQ_MASK);
	bool irqs_were_disabled;

	/* ——— */
	if (in_nmi())
		return;
	local_irq_save(flags);
	irqs_were_disabled = irqs_disabled_flags(flags);
	if (preempt_was_disabled || irqs_were_disabled) {
```

The Full Set of Commits

1. 3e3100989869 rcu: Defer reporting RCU-preempt quiescent states when disabled
2. 27c744e32a9a rcu: Allow processing deferred QSes for exiting RCU-preempt readers
3. fcc878e4dfb7 rcu: Remove now-unused ->b.exp_need_qs field from the rcu_special union
4. d28139c4e967 rcu: Apply RCU-bh QSes to RCU-sched and RCU-preempt when safe
5. ba1c64c27239 rcu: Report expedited grace periods at context-switch time
6. fced9c8cfe6b rcu: Avoid resched_cpu() when rescheduling the current CPU
7. 05f415715ce4 rcu: Speed up expedited GPs when interrupting RCU reader
8. 94fb70aa876b rcu: Make expedited IPI handler return after handling critical section

In Practice, Lots of Preparatory and Cleanup Work

- Merge grace-period counters: Reduce lock contention (35)
- Funnel-lock grace-period start: Reduce lock contention (3)
- Find and fix pre-existing intermittent rcutorture failures (15)
  - Want RCU squeaky clean before taking a meataxe to it
- Add quite a bit of debugging code (17)
- Add rcutorture quiescent-state deferral tests (42)
- Remove RCU-bh & RCU-sched and then simplify!!! (107)
  - And remove rcutorture scenarios testing RCU-bh and RCU-sched
- Drive-by optimizations (17)
- Additional cleanup as it becomes apparent (???)

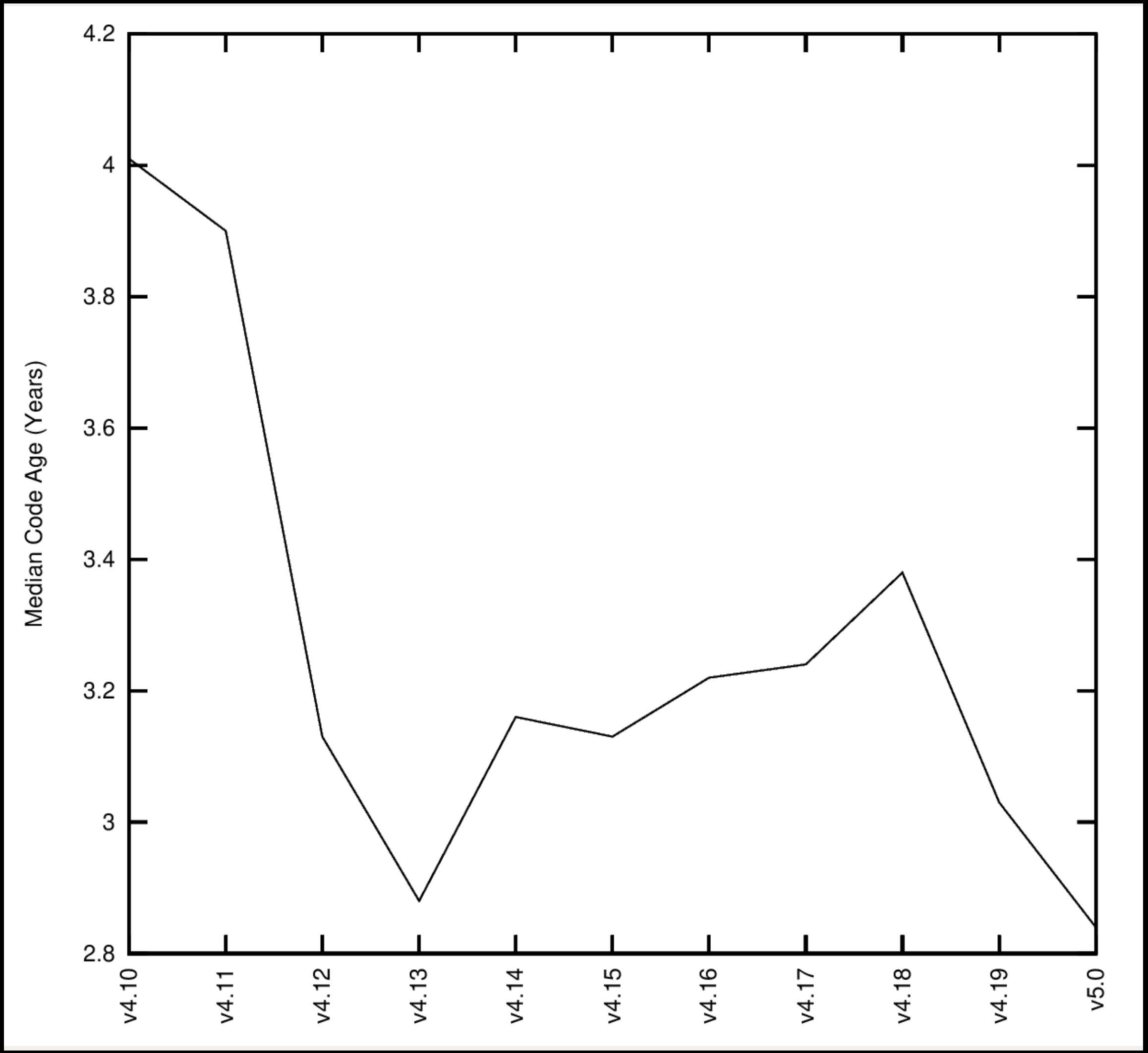
Near Misses: Saved by Community Processes!

- 0day finds a few issues
  - Build issue: Idle-loop entry change
  - Build issue: Definitions for 32-bit kernels
    - And many other fat-finger issues on various architectures
  - Boot-time issue: Infinite recursion through synchronize_rcu()
  - Runtime issue with rcu_read_unlock_special() recursion
    - Prompting a change in rcutorture testing scenarios
  - Runtime issue: Intermittent deadlock
  - Runtime issue: Intermittent spinlock recursion
  - Runtime issue: RCU readers from idle (several of these)
  - Runtime issue: Overly aggressive rcutorture testing
  - And much else besides
- Good review comments: Joel Fernandes now official reviewer

Other Consequences

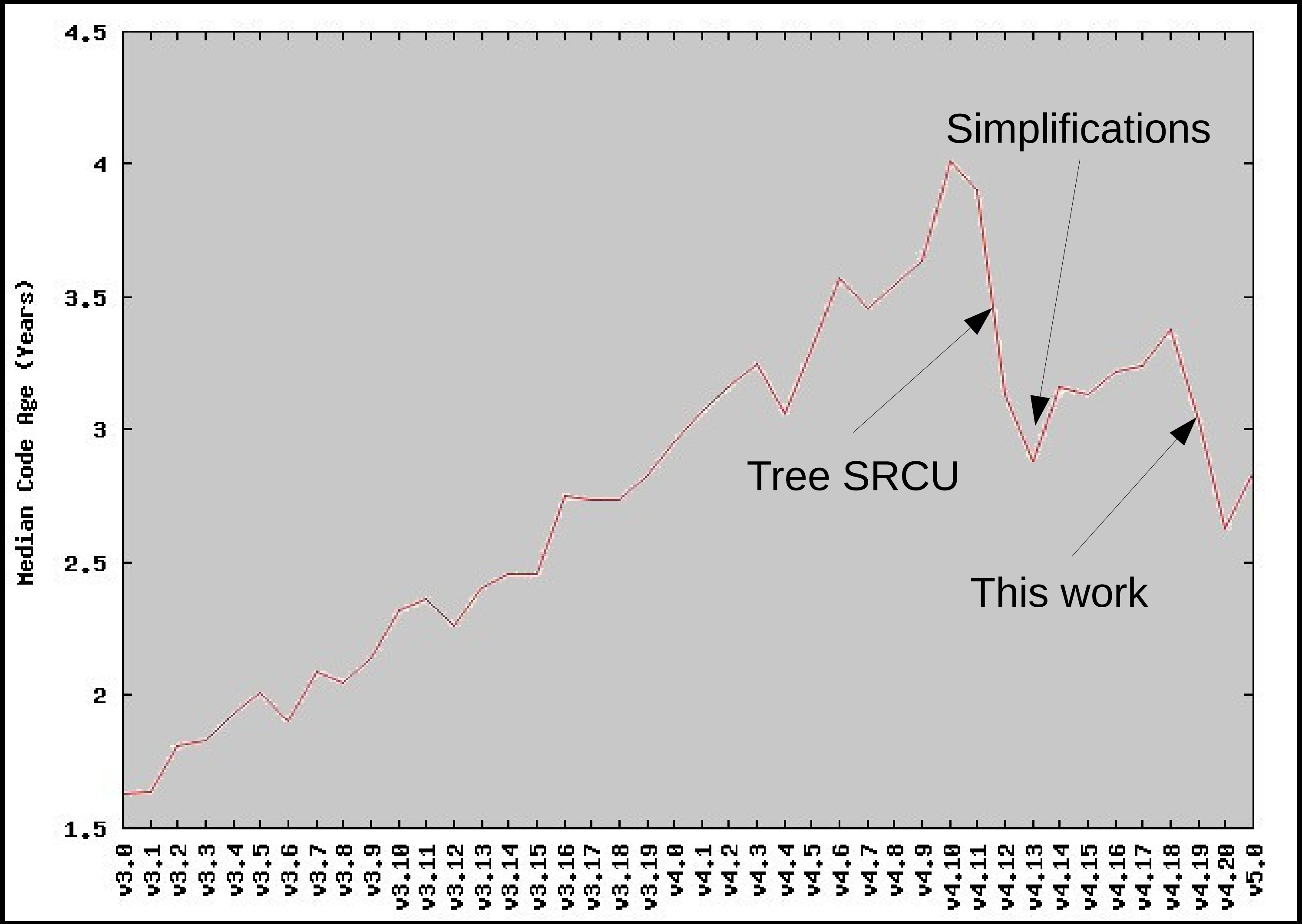
- What effect did this work have on RCU's reliability?
- According to rcutorture, it is actually ***more*** reliable
  - And rcutorture has become significantly more nasty
  - Which is a very good thing
- But this work did introduce some bugs
- Estimate reliability based on proxy: Median age of RCU code
  - One of those rare situations where older is usually more reliable...

Median Age of RCU Code



30% decrease in median age: Should we be worried?

Median Age of RCU Code



But longer-term trend is not too bad...
But there are undoubtedly still many bugs to find!!!

Recently Fixed Bugs and RCU Versions

- Reported by Thomas Gleixner and Sebastian Andrzej Siewior
  - Unnecessary preempt_disable, unrelated bug (v4.19 in 2018)
- Reported by David Woodhouse and Marius Hillenbrand
  - RCU stalled by KVM, unrelated bug (v4.12 in 2017)
- Dennis Krein
  - SRCU omitted lock from Tree SRCU rewrite (v4.12 in 2017)
- Sebastian Andrzej Siewior
  - SRCU -rt issue from Tree SRCU rewrite (v4.12 in 2017)
- Jun Zhang, Bo He, Jin Xiao, and Jie A Bai
  - Unrelated self-wakeup bug (v3.16 in 2014)
- Reported by Sebastian Andrzej Siewior
  - Failure of rcutorture to test GP hangs after offline (v3.3 in 2011)

Expectations

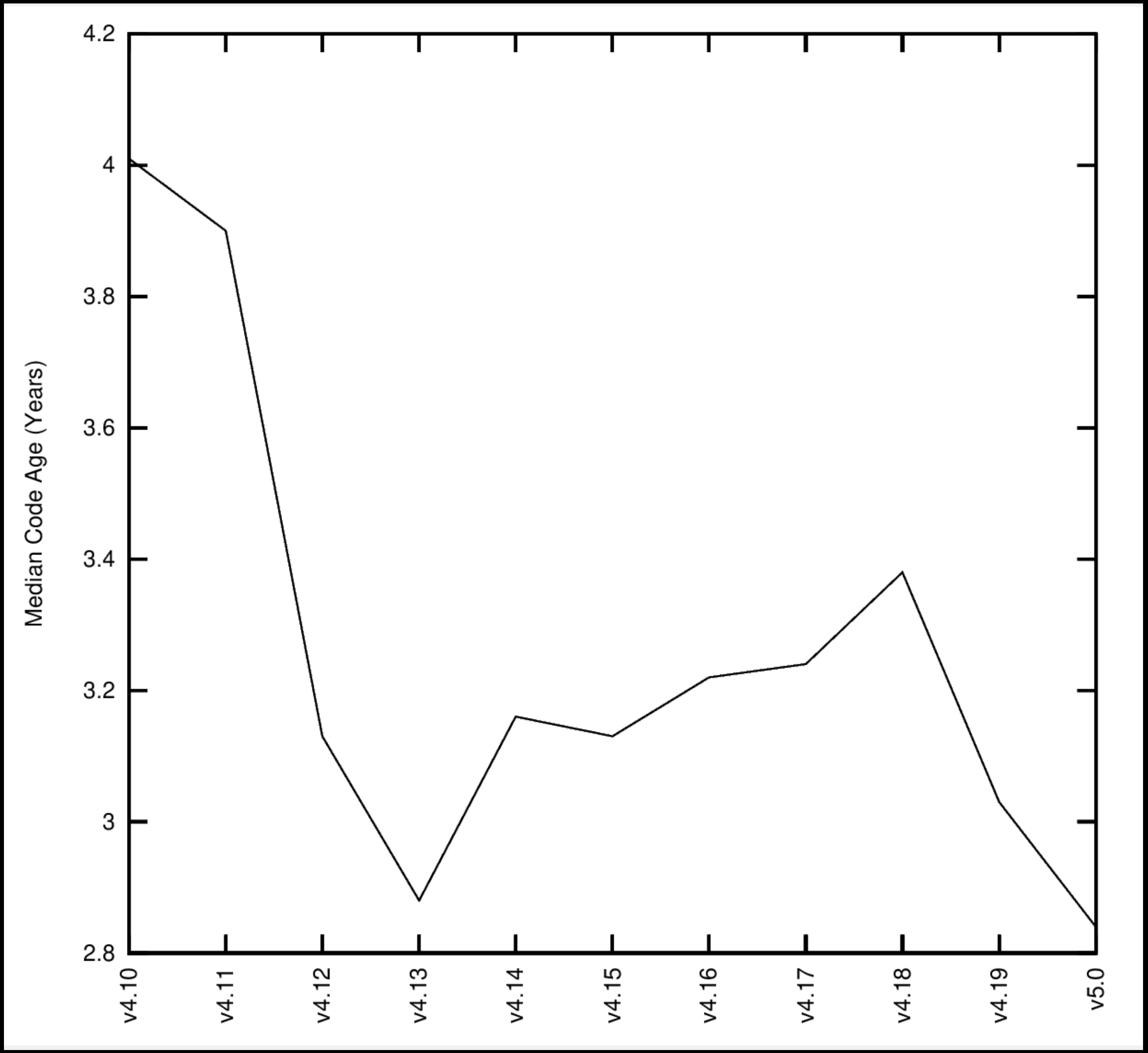
- More forward-progress bugs due to higher utilizations
  - But this is due to changes in workload, not RCU flavor consolidation
  - Nevertheless, area of current focus
- At least one more Tree SRCU bug
  - Tree SRCU seems to have doubled RCU's bug rate, give or take
- Several RCU flavor consolidation bugs
  - Not counting various nits
  - *Update:* Some changes required to accommodate -rt functionality
- The usual influx of bugs that I don't expect at all...

Expectations

- More forward-progress bugs due to higher utilizations
  - But this is due to changes in workload, not RCU flavor consolidation
  - Nevertheless, area of current focus
- At least one more Tree SRCU bug
  - Tree SRCU seems to have doubled RCU's bug rate, give or take
- Several RCU flavor consolidation bugs
  - Not counting various nits
  - *Update:* Some changes required to accommodate -rt functionality
- The usual influx of bugs that I don't expect at all...

Because Murphy Never Sleeps!!!

Why Not Be More Proactive for Expected RCU Bugs?



Why Not Be More Proactive for Expected RCU Bugs?

- Formal verification in RCU regression testing for the win?

Why Not Be More Proactive for Expected RCU Bugs?

- Formal verification in RCU regression testing for the win?
  - Lihao Liang et al., “Verification of the Tree-Based Hierarchical Read-Copy Update in the Linux Kernel”, https://arxiv.org/abs/1610.03052
    - Based on CBMC, which uses a SAT solver
  - Kokologiannakis et al., “Stateless Model Checking of the Linux Kernel’s Hierarchical Read-Copy-Update (Tree RCU)” https://michaliskok.github.io/papers/spin2017-rcu.pdf
    - Based on Nidhugg, which uses partial-order reduction
  - Roy, “rcutorture: Add CBMC-based formal verification for SRCU” Linux-kernel commit 418b2977b343
    - Based on CBMC
- How did these efforts work out?

How Did Formal Verification Work Out For RCU?

- Needed to configure RCU down to minimal code size
  - No CPU hotplug, no idle loop, no preemption, no callback offloading, ...
- Portions of RCU code extracted and placed into test harness
  - Both tools successfully ingested Linux-kernel C code: Very cool!!!
  - Both tools are just fine with non-linearizable concurrent algorithms
  - Both tools handle several weakish memory models
- Reported most—or even all—injected bugs
  - Yes, even formal verification tools must be validated!!!
  - We are all capable of writing printf(“Verified\n”), after all!!!

How Did Formal Verification Work Out For RCU?

- Needed to configure RCU down to minimal code size
  - No CPU hotplug, no idle loop, no preemption, no callback offloading, ...
- Portions of RCU code extracted and placed into test harness
  - Both tools successfully ingested Linux-kernel C code: Very cool!!!
  - Both tools are just fine with non-linearizable concurrent algorithms
  - Both tools handle several weakish memory models
- Reported most—or even all—injected bugs
  - Yes, even formal verification tools must be validated!!!
  - We are all capable of writing printf("Verified\n"), after all!!!
- But neither found any bugs that I was not already aware of!!!
  - That challenge is still open:
    - https://paulmck.livejournal.com/46993.html

Impressive Progress, But For FV Regression Testing:

(1) Either automatic translation or no translation required
- Automatic discarding of irrelevant portions of the code
- Manual translation provides opportunity for human error!

(2) Correctly handle environment, including memory model
- The QRCU validation benchmark is an excellent cautionary tale

(3) Reasonable memory and CPU overhead
- Bugs must be located in practice as well as in theory
- Linux-kernel RCU is 15KLoC (plus 5KLoC tests) and release cycles are short

(4) Map to source code line(s) containing the bug
- "Something is wrong somewhere" is not helpful: I already **know** bugs exist
- One bug reported just yesterday!!!

(5) Modest input outside of source code under test
- Preferably glean much of the specification from the source code itself (empirical spec!)
- Specifications are large bodies of software and can therefore have their own bugs

(6) Find relevant bugs
- Low false-positive rate, weight towards likelihood of occurrence (fixes create bugs!)
- For example, interesting recent work bounds number of preemptions

Summary

Summary

- Making your software do exactly what you want it to is a difficult undertaking
  - And it is insufficient: You might be confused about requirements
- Ease-of-use issues can result in security holes
  - Testing and reliability statistics are subject to misuse “Black Swans”
  - On the other hand, fixing these issues can simplify your code
- RCU currently seems to be in pretty good shape
  - But recent change means opportunity for formal verification
  - And there is some risk due to lack of synchronize_sched()
  - And real-time kernels don't like overlapping disable regions

Summary

- Making your software do exactly what you want it to is a difficult undertaking
  - And it is insufficient: You might be confused about requirements
- Ease-of-use issues can result in security holes
  - Testing and reliability statistics are subject to misuse “Black Swans”
  - On the other hand, fixing these issues can simplify your code
- ***RCU currently seems to be in pretty good shape***
  - But recent change means opportunity for formal verification
  - And there is some risk due to lack of synchronize_sched()
  - And real-time kernels don't like overlapping disable regions
- Famous last words...

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?

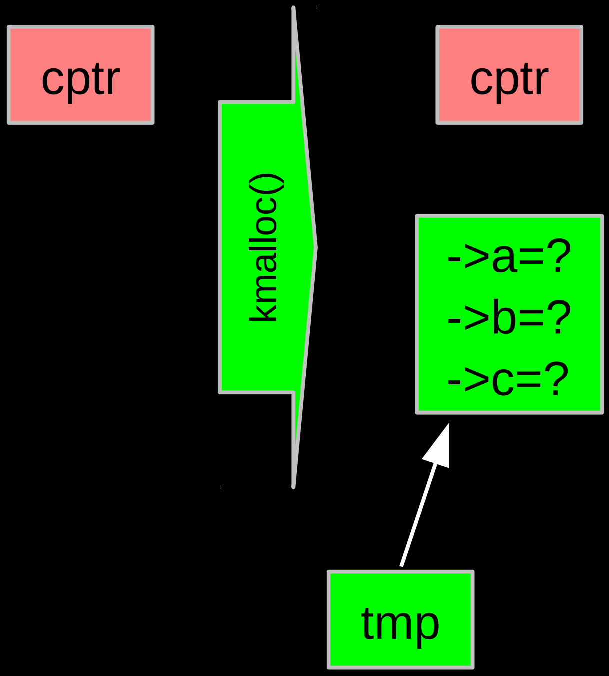
Publication of And Subscription to New Data

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

cptr

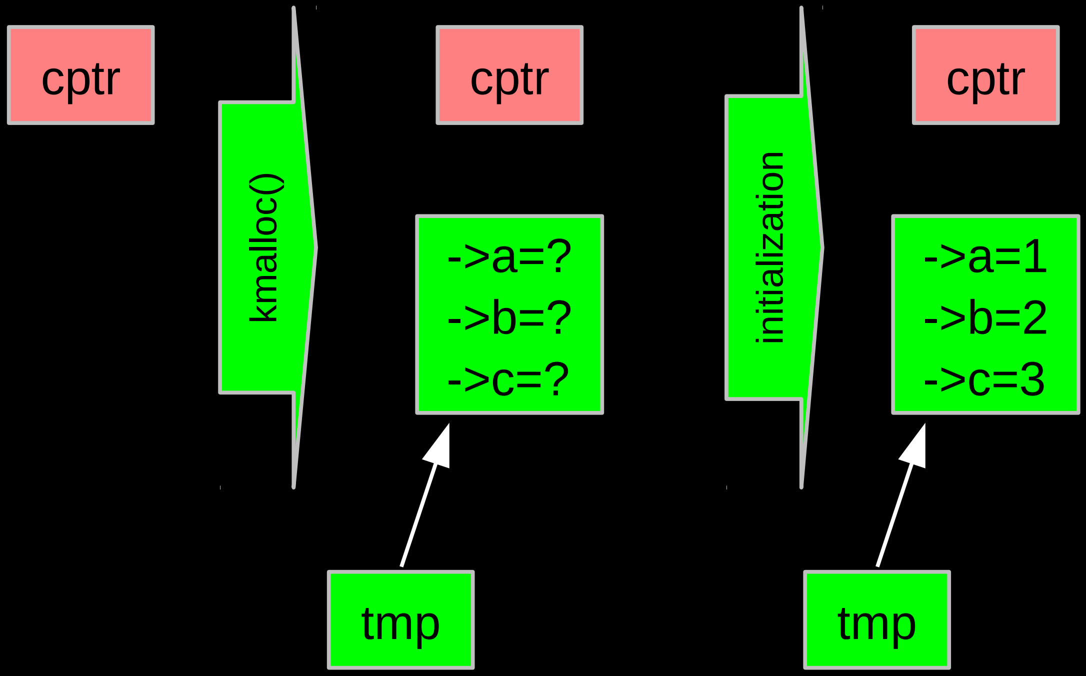
Publication of And Subscription to New Data

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



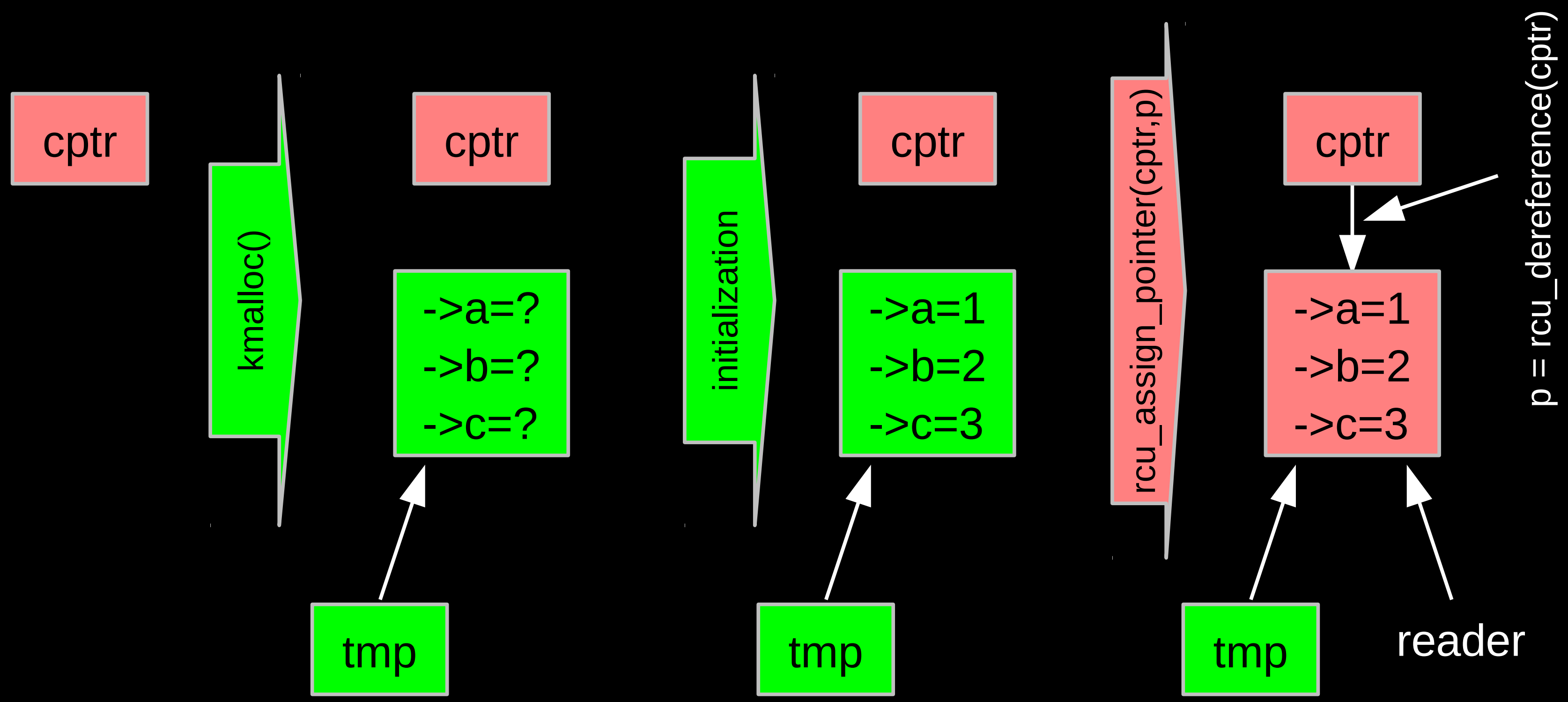
Publication of And Subscription to New Data

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



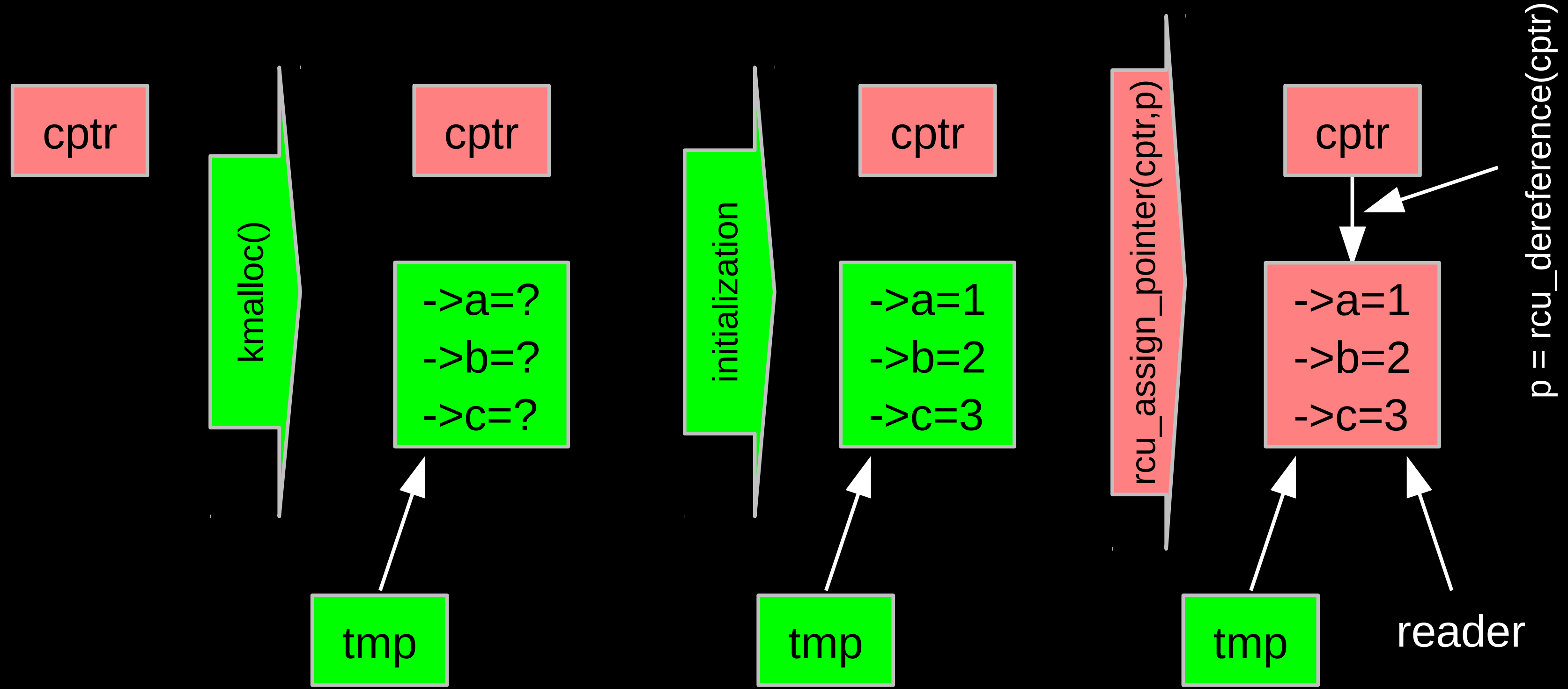
Publication of And Subscription to New Data

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



Publication of And Subscription to New Data

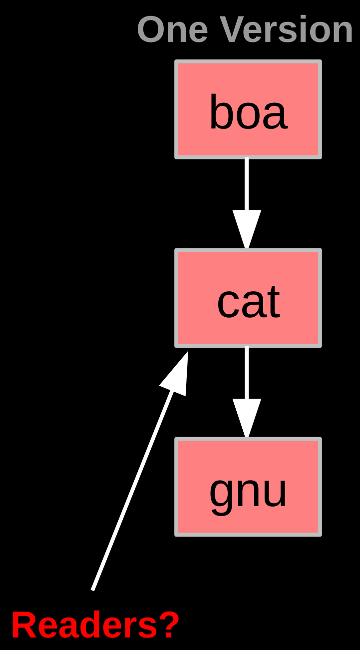
Key:
- Dangerous for updates: all readers can access
- Still dangerous for updates: pre-existing readers can access (next slide)
- Safe for updates: inaccessible to all readers



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

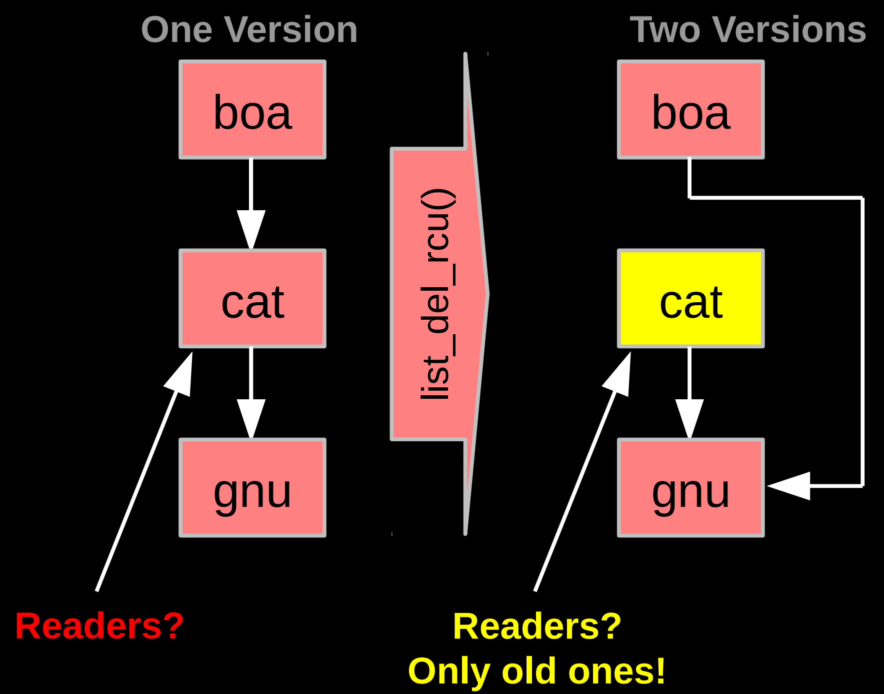
RCU Removal From Linked List

- Combines waiting for readers and multiple versions:



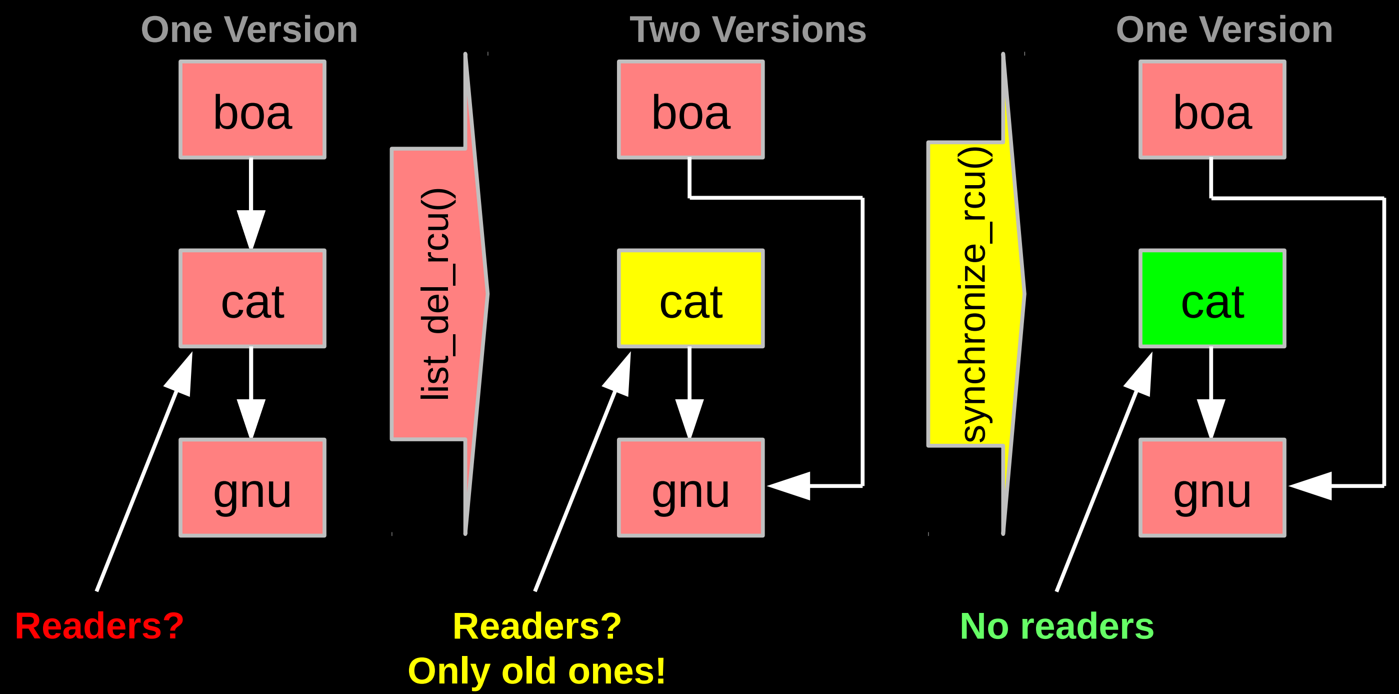
RCU Removal From Linked List

- Combines waiting for readers and multiple versions:
  - Writer removes the cat's element from the list (list_del_rcu())



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())



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())