

Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center

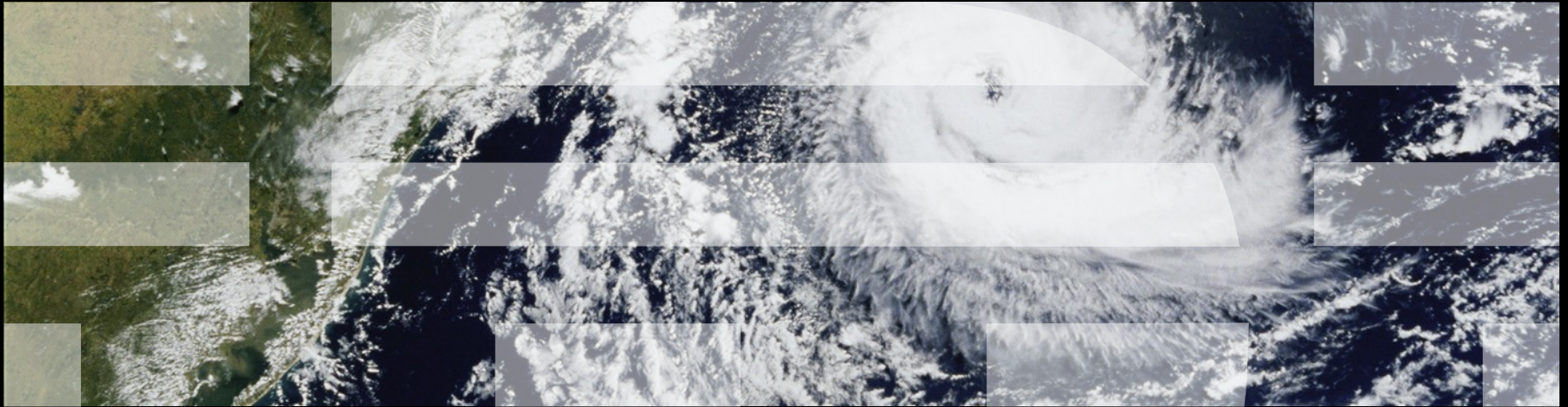
Member, IBM Academy of Technology

linux.conf.au, January 23, 2019



RCU's First-Ever CVE

And How I Lived to Tell the Tale

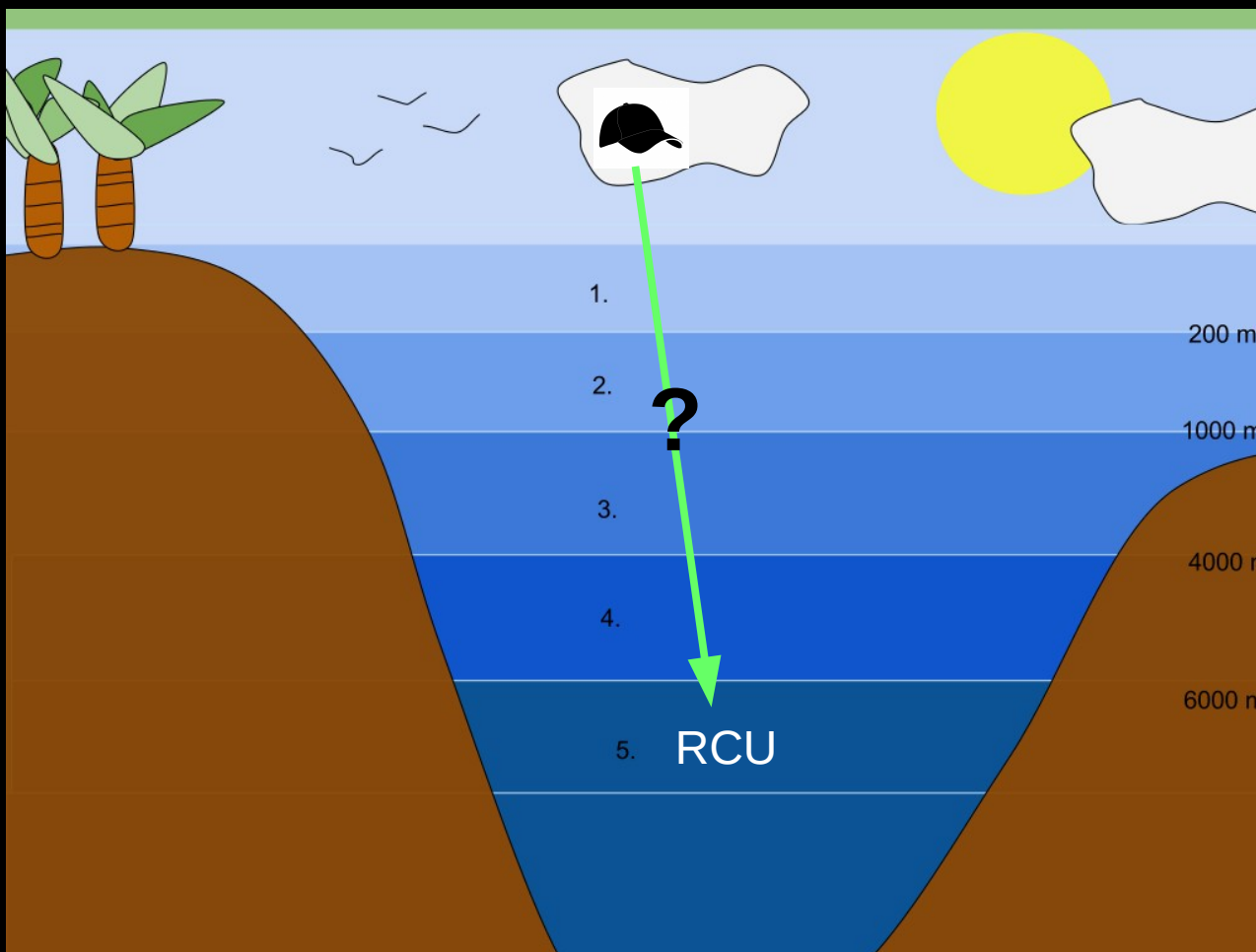


Overview

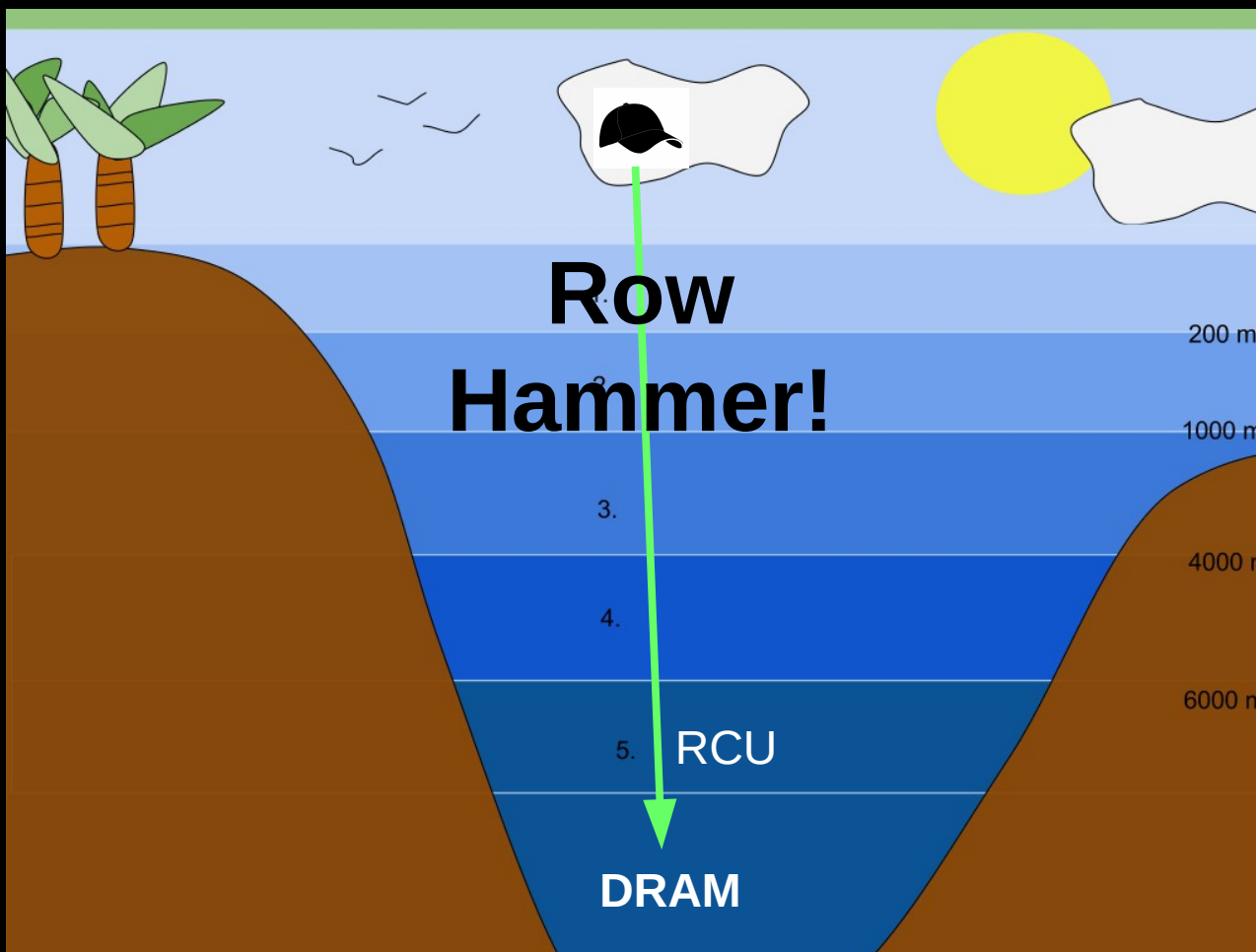
- Isn't RCU a bit low-level to be involved in a CVE?
- What is the CVE number?
- What was the real problem?
- What would a fix even look like???
- Possible solutions
- Other consequences
- Summary

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

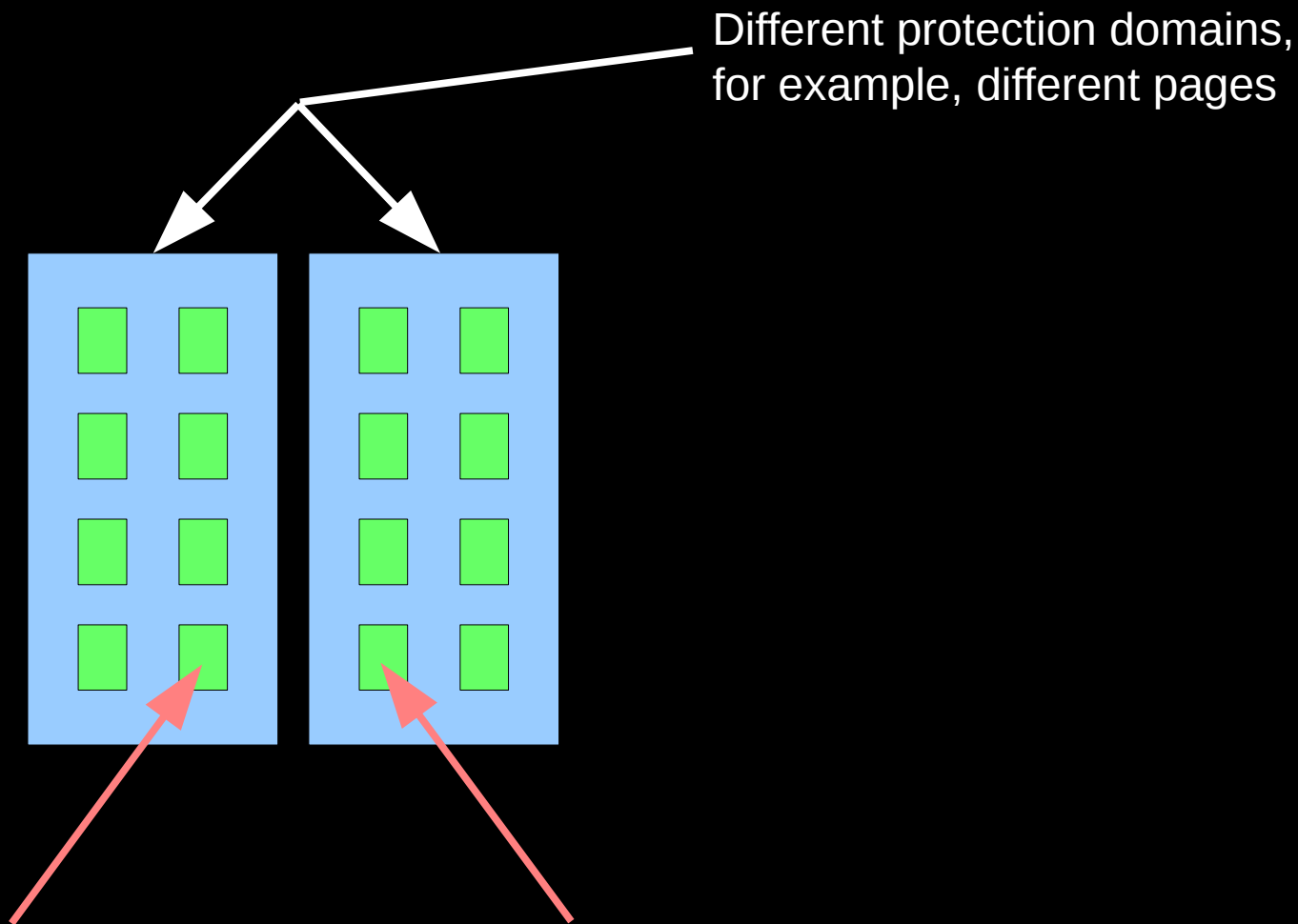
Isn't RCU a Bit Low-Level to be Involved in a CVE?



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



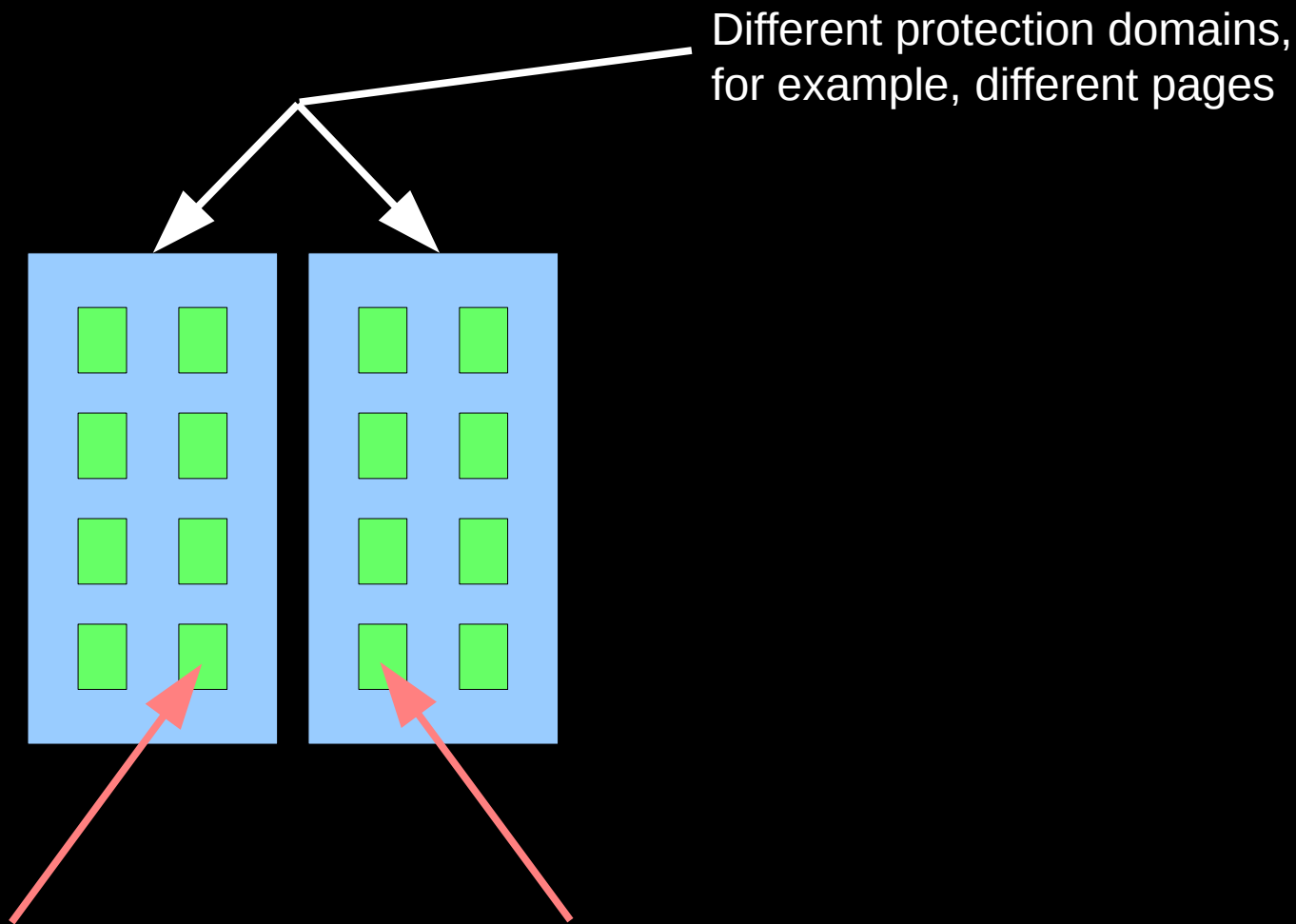
Obligatory Row Hammer Diagram



Manipulating this bit...

... can change this bit, protection domain notwithstanding.

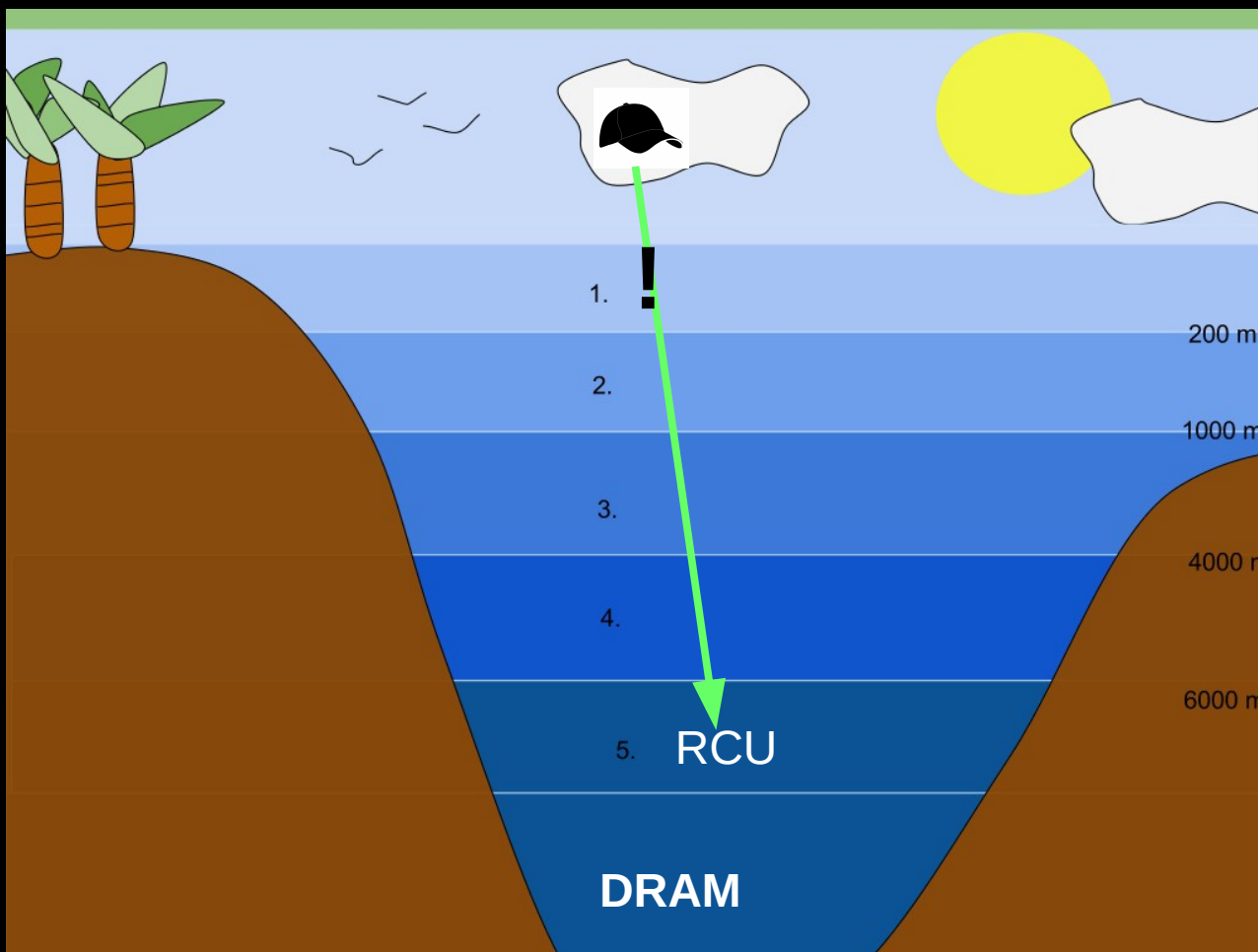
Obligatory Row Hammer Diagram: Cannot Virtualize EMI Out of Existence!!!



Manipulating this bit...

... can change this bit, protection domain notwithstanding.

If Black Hats Can Hit DRAM, 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 <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 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.

Linus

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 <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:

[. . .]

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.

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 <jannah@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 <jannah@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.



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 <jannah@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 different and confusing versions of it. It clearly causes real problems.

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

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 <jannah@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 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

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 <jannah@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

[. . .]

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 fix in practice or mis-use, but still.

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

Linus

But First, What is the CVE Number???

But First, What is the CVE Number???

I have no idea.

But First, What is the CVE Number???

I have no idea.

I am not cleared for embargoed security issues.

But First, What is the CVE Number???

I have no idea.

**I am not cleared for embargoed security issues.
Is there really a CVE number for this issue?**

But First, What is the CVE Number???

I have no idea.

I am not cleared for embargoed security issues.

Is there really a CVE number for this issue?

Again, I have no idea.

But First, What is the CVE Number???

I have no idea.

I am not cleared for embargoed security issues.

Is there really a CVE number for this issue?

Again, I have no idea.

But there was an exploitable bug.

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();
}
```

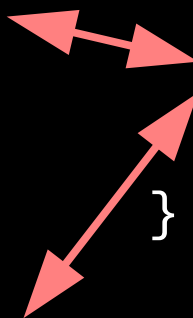
```
void updater(void)
{
    /* Remove old data. */
    synchronize_rcu();
    /* Free old data. */
}
```

Why is This an Abuse of RCU???

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!!!

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

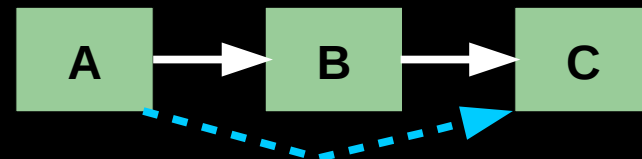
```
rcu_read_lock_sched();
```

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

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

Still using B!!!

```
rcu_read_unlock_sched()
```



```
list_del_rcu(B);
```

```
synchronize_rcu();
```

```
kfree(B);
```

**What are developers
supposed to do
instead?**

What Are Developers Supposed to do Instead?

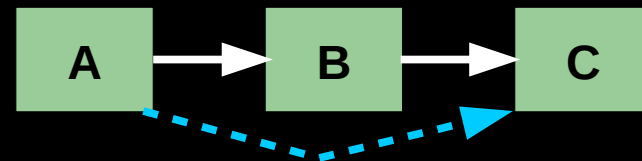
```
rcu_read_lock();
```

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

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

```
Still using B,  
but that's OK!
```

```
rcu_read_unlock()
```



```
list_del_rcu(B);
```

```
synchronize_rcu();
```

```
kfree(B);
```

Or, Alternatively, Adjust the Updater:

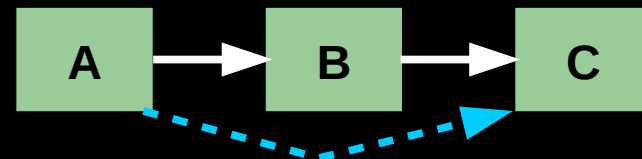
```
rcu_read_lock_sched();
```

```
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();
```



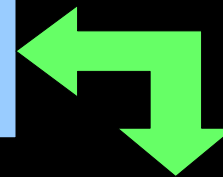
```
list_del_rcu(B);
```

```
synchronize_sched();
```

```
kfree(B);
```

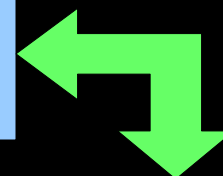
Consistency is Required, But That is a Problem!

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



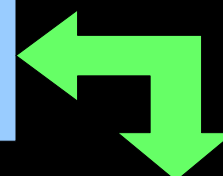
```
synchronize_rcu();
```

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



```
synchronize_rcu_bh();
```

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



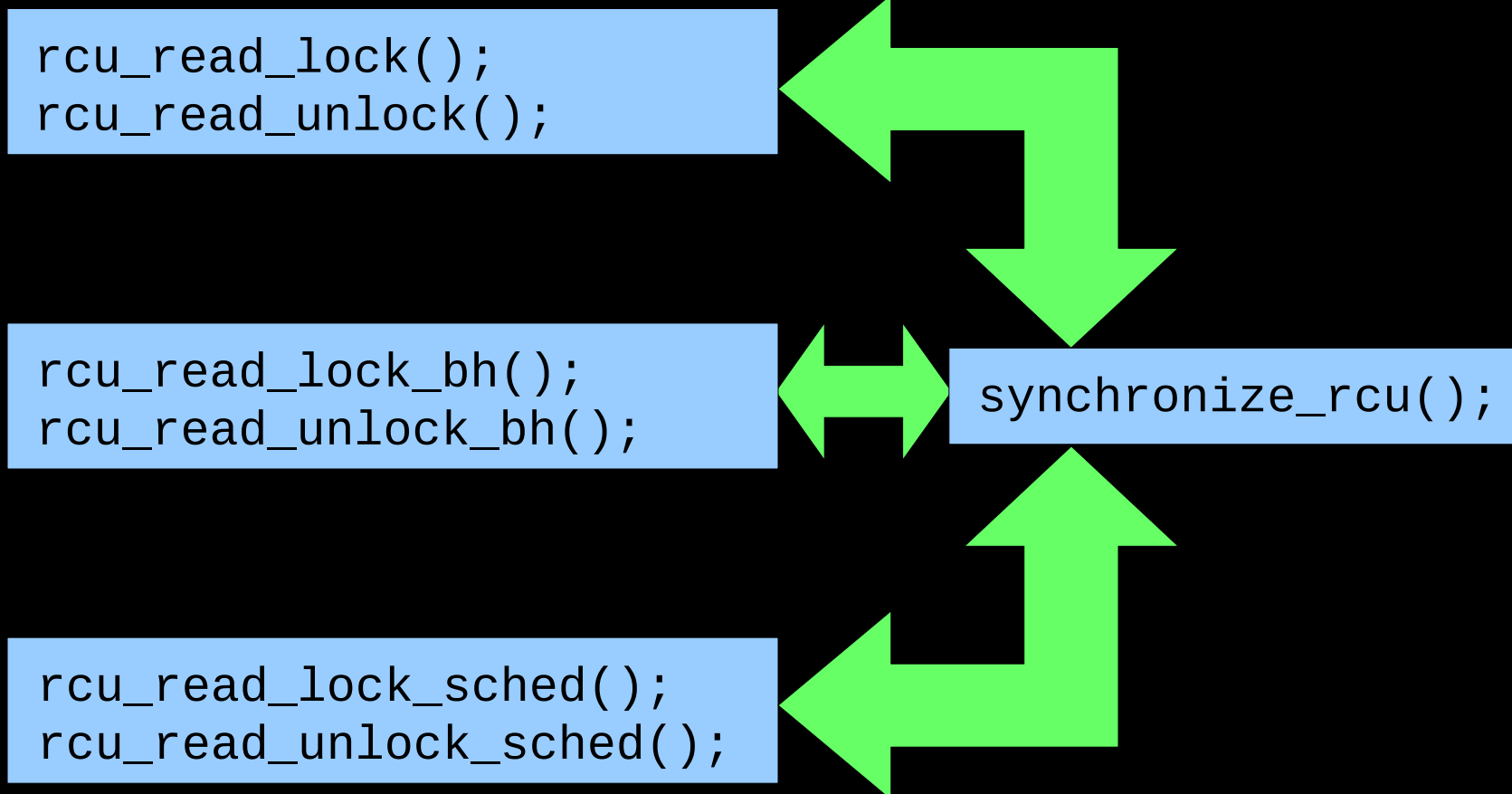
```
synchronize_sched();
```

To err is human...

Plus userspace controls content of much kernel data!!!

What Would A Fix Even Look Like???

Desired State From Usability/Security Viewpoint:



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

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

```
rcu_read_lock_bh();  
rcu_read_unlock_bh();  
local_bh_disable();  
local_bh_enable();  
. . .
```

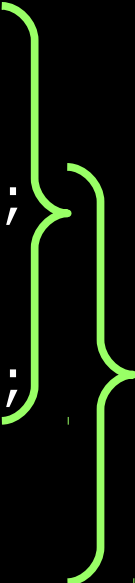
```
rcu_read_lock_sched();  
rcu_read_unlock_sched();  
preempt_disable();  
preempt_enable();  
local_irq_disable();  
local_irq_enable();  
. . .
```

```
synchronize_rcu();
```




Elaborations on Desired State

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

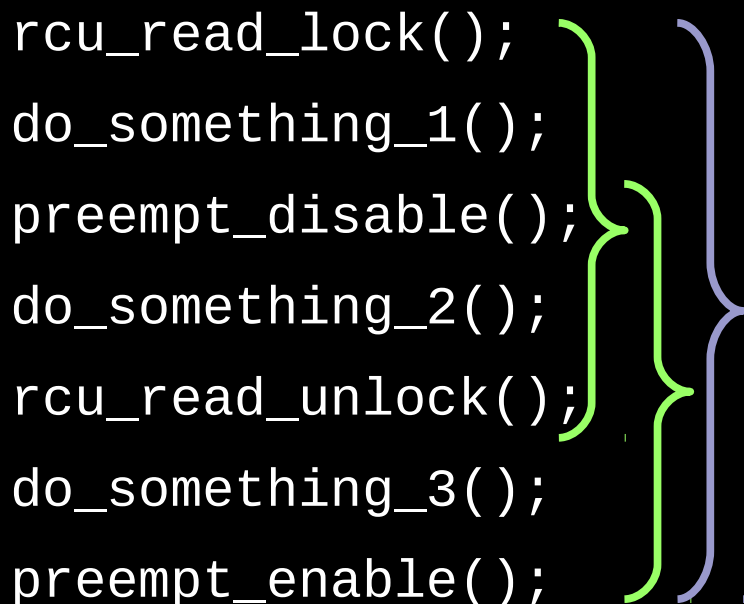


```
local_bh_disable();  
do_something_1();  
rcu_read_lock();  
local_bh_enable();  
do_something_2();  
preempt_disable();  
rcu_read_unlock();  
do_something_3();  
preempt_enable();
```

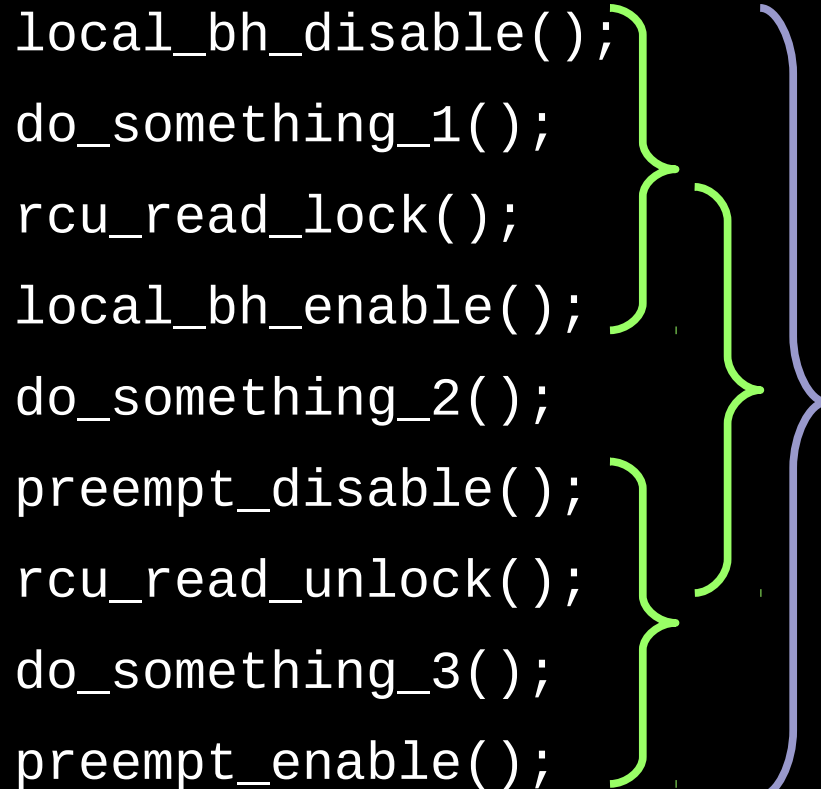


Elaborations on Desired State

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



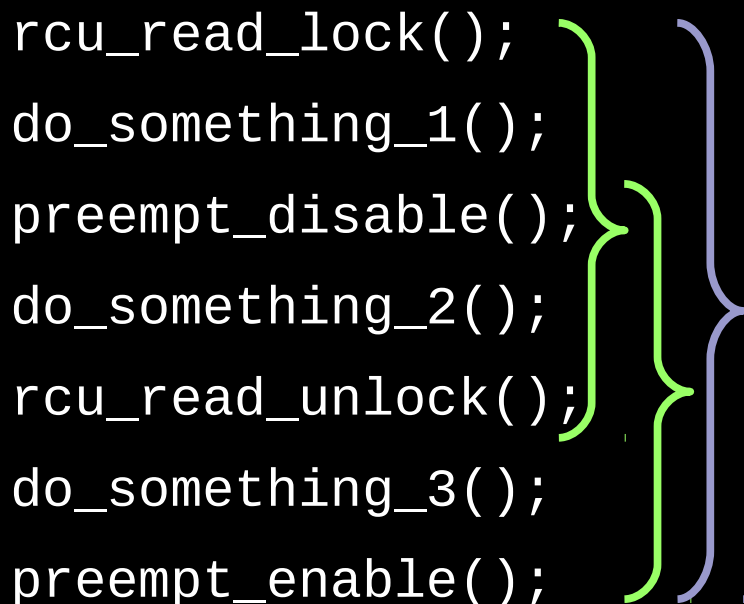
```
local_bh_disable();  
do_something_1();  
rcu_read_lock();  
local_bh_enable();  
do_something_2();  
preempt_disable();  
rcu_read_unlock();  
do_something_3();  
preempt_enable();
```



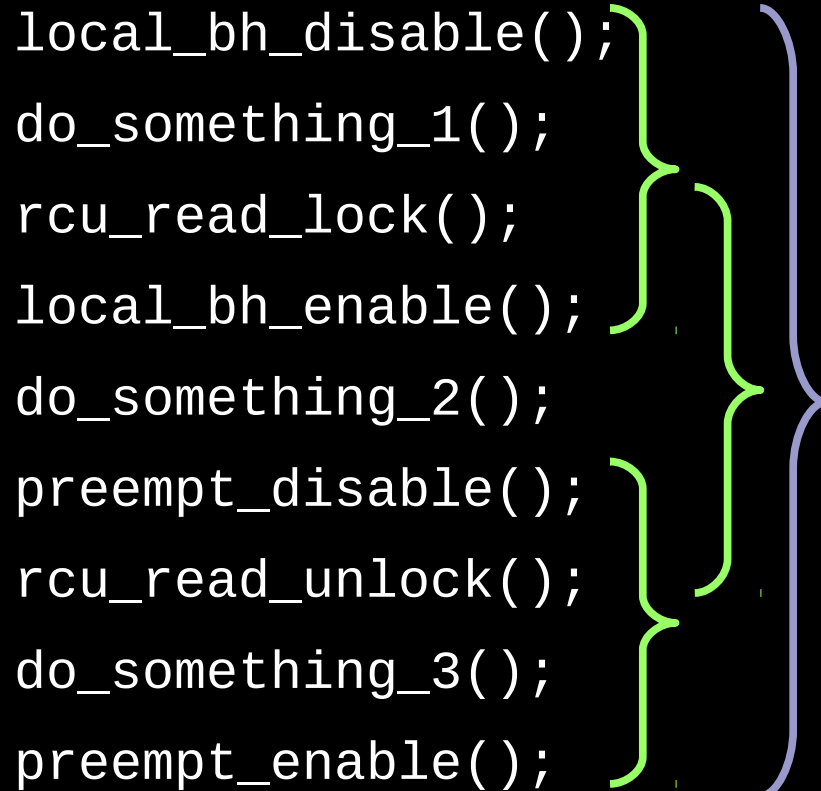
Elaborations on Desired State...

But Please Keep This to a Minimum in Kernel Code!!!

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



```
local_bh_disable();  
do_something_1();  
rcu_read_lock();  
local_bh_enable();  
do_something_2();  
preempt_disable();  
rcu_read_unlock();  
do_something_3();  
preempt_enable();
```



There are a *lot* of possible combinations of elaborations!!!

So rcutorture does up to eight randomly selected elaborations per reader

Thankfully, There Is Some Good News

Thankfully, There Is Some Good News For PREEMPT=n, RCU Already Handles This!!! *

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

The code block on the left is annotated with two sets of curly braces. A green brace on the right side groups the lines from `rcu_read_lock();` to `rcu_read_unlock();`. A blue brace on the right side groups the lines from `preempt_disable();` to `preempt_enable();`. The `do_something_1();` and `do_something_2();` lines are nested within both the green and blue braces.

```
local_bh_disable();
do_something_1();
rcu_read_lock();
local_bh_enable();
do_something_2();
preempt_disable();
rcu_read_unlock();
do_something_3();
preempt_enable();
```

The code block on the right is annotated with two sets of curly braces. A green brace on the right side groups the lines from `local_bh_disable();` to `local_bh_enable();`. A blue brace on the right side groups the lines from `preempt_disable();` to `preempt_enable();`. The `do_something_1();` and `do_something_2();` lines are nested within both the green and blue braces.

* Give or take RCU-bh quiescent states

But too bad about PREEMPT=y kernels...

Possible Solutions

Possible Solution: Add Explicit RCU Readers

Possible Solution: Add Explicit RCU Readers

Example: `preempt_disable()` and `preempt_enable()`

`preempt_disable()`

`preempt_enable()`

`preempt_disable()`

`rcu_read_lock()`

`rcu_read_unlock()`

`preempt_enable()`

Adds some overhead on some fastpaths, but **security!!!**

Possible Solution: Add Explicit RCU Readers

- Try easy approaches first!!! Add RCU readers:
 - Make `local_bh_disable()` do `rcu_read_lock()` just before returning and `local_bh_enable()` do `rcu_read_unlock()` just after being called
 - Make `preempt_disable()` do `rcu_read_lock()` just before returning and `preempt_enable()` do `rcu_read_unlock()` just after being called
 - Make `local_irq_disable()` do `rcu_read_lock()` just before returning and `local_irq_enable()` do `rcu_read_unlock()` just after being called
 - And same for the many other disable/enable functions

Possible Solution: Add Explicit RCU Readers

- Try easy approaches first!!! Add RCU readers:
 - Make `local_bh_disable()` do `rcu_read_lock()` just before returning and `local_bh_enable()` do `rcu_read_unlock()` just after being called
 - Make `preempt_disable()` do `rcu_read_lock()` just before returning and `preempt_enable()` do `rcu_read_unlock()` just after being called
 - Make `local_irq_disable()` do `rcu_read_lock()` just before returning and `local_irq_enable()` do `rcu_read_unlock()` just after being called
 - And same for the many other disable/enable functions
- How many people find this a bit scary?

Possible Solution: Add Explicit RCU Readers

- Try easy approaches first!!! Add RCU readers:
 - Make `local_bh_disable()` do `rcu_read_lock()` just before returning and `local_bh_enable()` do `rcu_read_unlock()` just after being called
 - Make `preempt_disable()` do `rcu_read_lock()` just before returning and `preempt_enable()` do `rcu_read_unlock()` just after being called
 - Make `local_irq_disable()` do `rcu_read_lock()` just before returning and `local_irq_enable()` do `rcu_read_unlock()` just after being called
 - And same for the many other disable/enable functions
- How many people find this a bit scary?
- So test it first: Instead of `rcu_read_lock()`, increment counter and instead of `rcu_read_unlock()`, decrement same counter
 - Complain if counter non-zero where everything is enabled

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

- Try easy approaches to add explicit readers:
 - Make `local_bh_disable` call `rcu_read_lock` before returning and `local_bh_enable` call `rcu_read_unlock` before being called
 - Make `preempt_disable` call `rcu_read_lock` before returning and `preempt_enable` call `rcu_read_unlock` before being called
 - Make `local_irq_disable` call `rcu_read_lock` before returning and `local_irq_enable` call `rcu_read_unlock` before being called
 - And same for `spin_lock` and `spin_unlock`
- How many pointers to `rcu_read_lock` and `rcu_read_unlock` are there?
- So test it first:
 - Add `rcu_read_lock` and `rcu_read_unlock` to increment counter
 - Add `rcu_read_lock` and `rcu_read_unlock` to decrement counter
 - Complain if counter is not zero when `rcu_read_lock` is enabled

Possible Solution: Defer Reporting Quiescent States

Possible Solution: Defer Reporting Quiescent States

- If preemption/irq/bh disabled across `rcu_read_unlock()`, don't report the quiescent state until everything is enabled

Possible Solution: Defer Reporting Quiescent States

- If preemption/irq/bh disabled across `rcu_read_unlock()`, don't report the quiescent state until everything is enabled
- The algorithm can be described in one sentence, so the implementation cannot possibly be all that hard, right?

Possible Solution: Defer Reporting Quiescent States

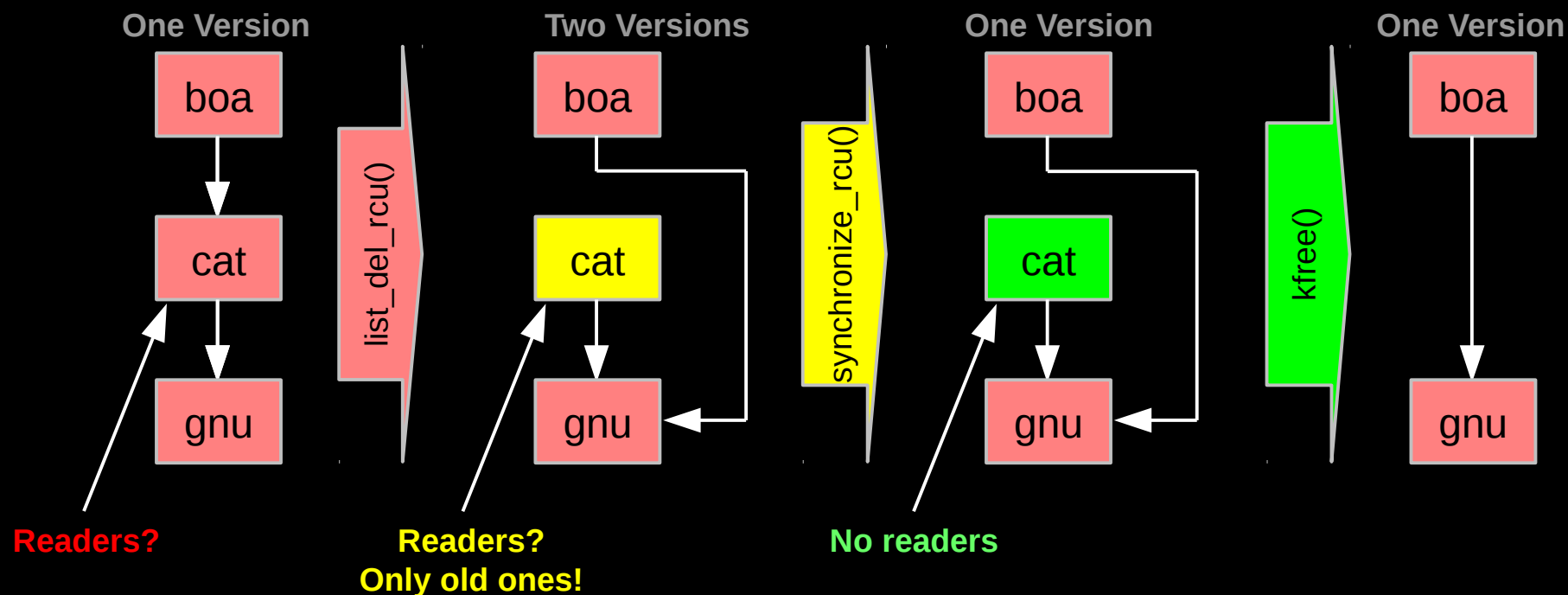
- If preemption/irq/bh disabled across `rcu_read_unlock()`, don't report the quiescent state until everything is enabled
- The algorithm can be described in one sentence, so the implementation cannot possibly be all that hard, right?
 - Just plumb it into RCU's quiescent-state reporting infrastructure!!!

Possible Solution: Defer Reporting Quiescent States

- If preemption/irq/bh disabled across `rcu_read_unlock()`, don't report the quiescent state until everything is enabled
- The algorithm can be described in one sentence, so the implementation cannot possibly be all that hard, right?
 - Just plumb it into RCU's quiescent-state reporting infrastructure!!!
- But first, what on earth is an RCU quiescent state???
 - Let's review RCU deletion from a linked list...

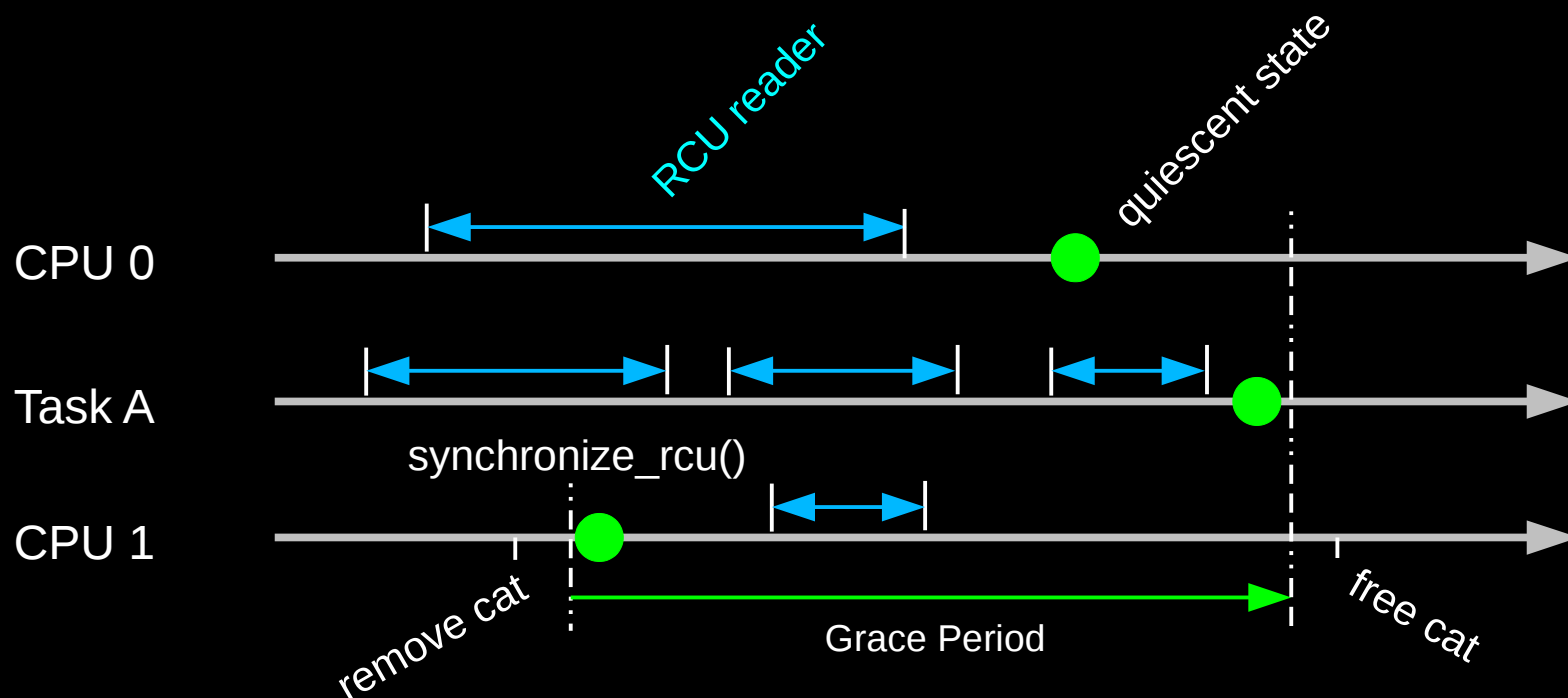
RCU Removal From Linked List

- 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()`)



RCU Removal From Linked List: Quiescent States

- CPU *quiescent state* means all that CPU's readers are done
 - Quiescent states include context switch, idle, offline, `cond_resched()`:
Special cases of enabled code not in an RCU read-side critical section
- *Grace period* ends after everything passes through a quiescent state



From Quiescent States to Grace Periods

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

```
void synchronize_rcu(void)
{
    atomic_set(&nqsneeded, num_online_cpus());
    wait_event(gp_wait, nqsneeded == 0);
}
```

```
void rcu_note_context_switch(bool preempt)
{
    If (atomic_dec_and_test(&nqsneeded))
        wake_up(&gp_wait);
}
```

From Quiescent States to Grace Periods

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

```
void synchronize_rcu_expedited()
{
    atomic_notifier_chain_notify(&rcu_notifier_chain);
    wait_for_completion(&rcu_completion);
}

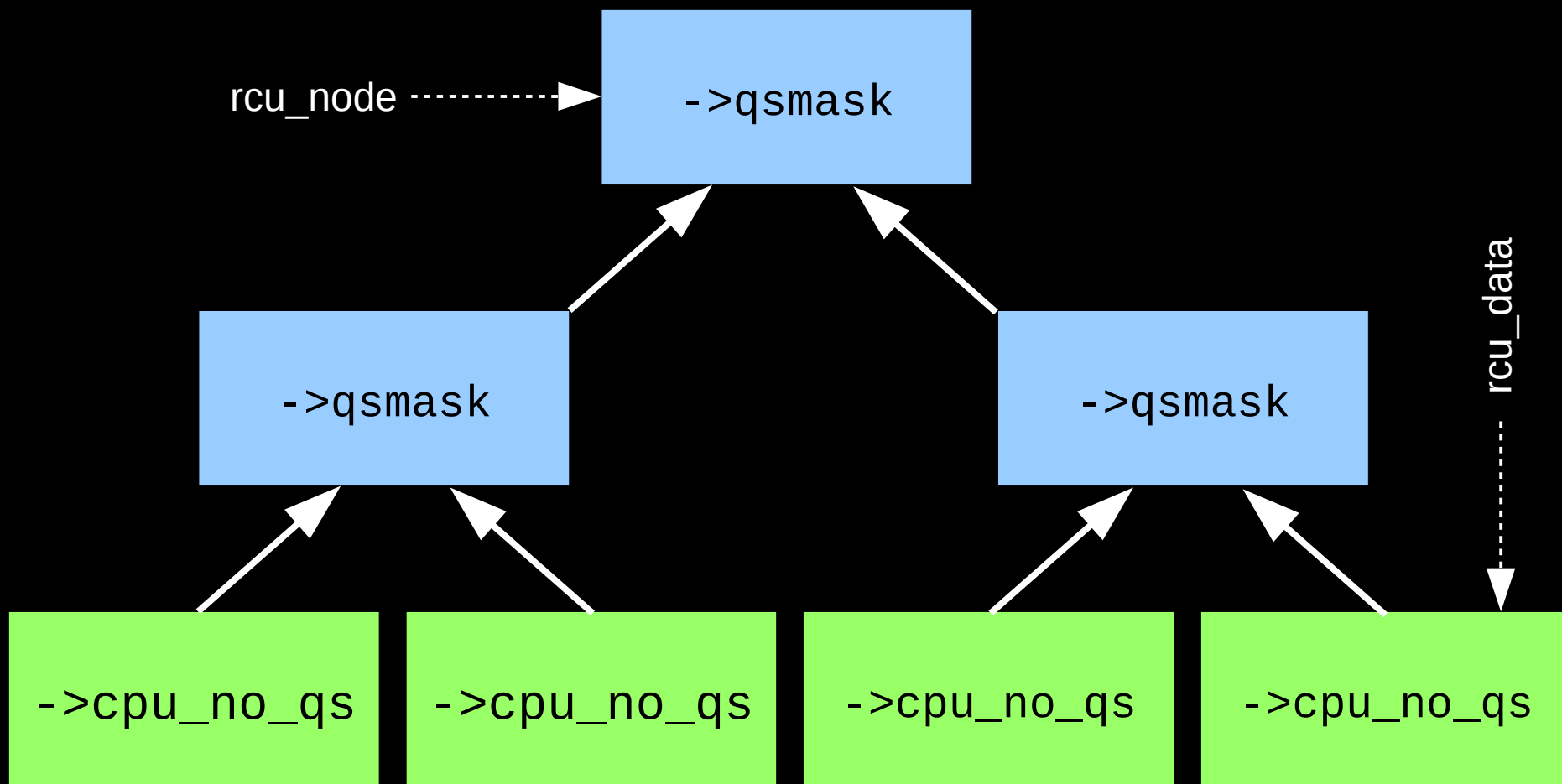
void rcu_note_context_switch(int cpu)
{
    If (atomic_notifier_chain_notify(&rcu_notifier_chain));
    wait_for_completion(&rcu_completion);
}
```



Fail

**CPU hotplug, scalability,
multiple quiescent states,
...**

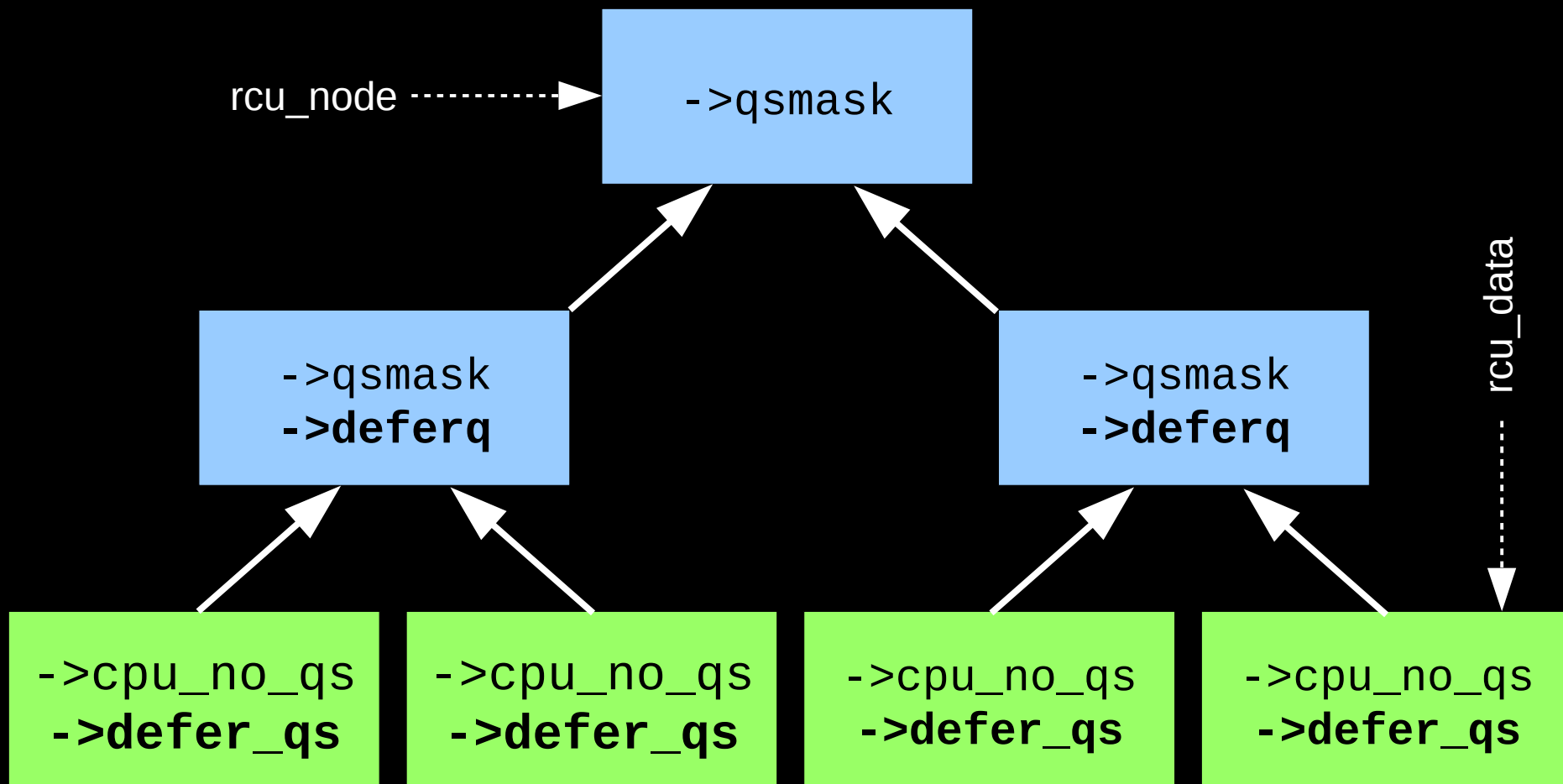
Improve Scalability With Combining Tree (Tree RCU)



Improve Scalability With Combining Tree (Tree RCU)

- Start grace period: Set all `->cpu_no_qs` flags and `->qsmask` bits corresponding to online CPUs
- Quiescent state:
 - If CPU's `rcu_data` structure's `->cpu_no_qs` flag is set, clear it and proceed to leaf `rcu_node` structure
 - If CPU's bit in leaf `rcu_node` structure's `->qsmask` is set, clear it and if all bits are now clear, proceed to root `rcu_node` structure
 - Protected by leaf `rcu_node` structure's `->lock` field
 - If corresponding bit in root `rcu_node` structure's `->qsmask` is set, clear it, and if all bits are now clear, end of grace period!
 - Protected by root `rcu_node` structure's `->lock` field
- Constant lock contention no matter how deep the tree!!!

Defer Quiescent States With Combining Tree: Atomic Count of CPUs w/Deferred Quiescent States



Improve Scalability With Combining Tree, Including Deferred Quiescent States (1/2)

- Start grace period:
 - Set all `->cpu_no_qs` flags and `->qsmask` bits per online CPUs
 - Clear all `->deferq` and `->defer_qs` flags**
- Quiescent state:
 - If CPU's `rcu_data's ->cpu_no_qs` flag is set **and if `->defer_qs` is clear**, clear `->cpu_no_qs` and proceed to leaf `rcu_node` structure
 - If CPU's bit in leaf `rcu_node's ->qsmask` is set, clear it and if all bits **and `->deferq`** are now clear, proceed to root `rcu_node` structure
 - If corresponding bit in root `rcu_node` structure's `->qsmask` is set, clear it, and if all bits are now clear, end of grace period!
- **Disabled during `rcu_read_unlock()`:**
 - Set `->defer_qs` and atomically increment `->deferq`**
- **When enabled... (Next slide)**

Improve Scalability With Combining Tree, Including Deferred Quiescent States (2/2)

- ***When enabled and rcu_data's ->defer_qs is set:***
 - ***Clear rcu_data's ->defer_qs***
 - ***Atomically decrement rcu_node's ->deferq, and if zero proceed to the leaf rcu_node***
 - ***If CPU's bit in leaf rcu_node's ->qsmask is set, clear it and if all bits and ->deferq are now clear, proceed to root rcu_node structure***
 - ***If corresponding bit in root rcu_node structure's ->qsmask is set, clear it, and if all bits are now clear, end of grace period!***

Improve Scalability With Combining Tree, Including Deferred Quiescent States (2/2)

- ***When enabled and rcu_data's ->defer_qs is set:***
 - ***Clear rcu_data's ->defer_qs***
 - ***Atomically decrement rcu_node's ->deferq, and if zero proceed to the leaf rcu_node***
 - ***If CPU's bit in leaf rcu_node's ->qsmask is set, clear it and if all bits and ->deferq are now clear, proceed to root rcu_node structure***
 - ***If corresponding bit in root rcu_node structure's ->qsmask is set, clear it, and if all bits are now clear, end of grace period!***

- Time to start coding!!!

Combining Tree With Deferred Quiescent States (Part of a Page, Five Pages Total)

```

void rcu_preempt_deferred_qs(struct rcu_data *rdp)
{
    unsigned long flags;
    struct rcu_node *rnp;

    lockdep_assert_irqs_disabled();
    if ((!rdp->defer_norm_qs && !rdp->defer_exp_qs) || in rcu RS-CS)
        return;
    if (rdp->defer_norm_qs) {
        rnp = rdp->defer_norm_qs->node;
        raw_spin_lock_irqsave_rcu_node(rnp, flags);
        rdp->defer_norm_qs = false;
    }
}

```

Combining Tree With Deferred Quiescent States (Part of a Page, Five Pages Total)

```

void rcu-preempt-deferred-gs (struct rcu_data *rdp)
{
    unsigned long flags;
    struct rcu_node *rnp;

    lockdep_assert_irqs_disabled();
    if ((!rdp->defer-norm-gs && !rdp->defer-exp-gs) || in-rcu RS-CS)
        return;
    if (rdp->defer-norm-gs) {
        rnp = rdp->defer-norm-gs-node;
        raw_spin_lock_irqsave_rcu_node(rnp, flags);
        rdp->defer-norm-gs = false;
    }
}

```

But wait! Preempted task queuing, stall warnings, expedited grace periods, ...

Combining Tree With Deferred Quiescent States (Part of a Page, Five Eight Pages Total)

```

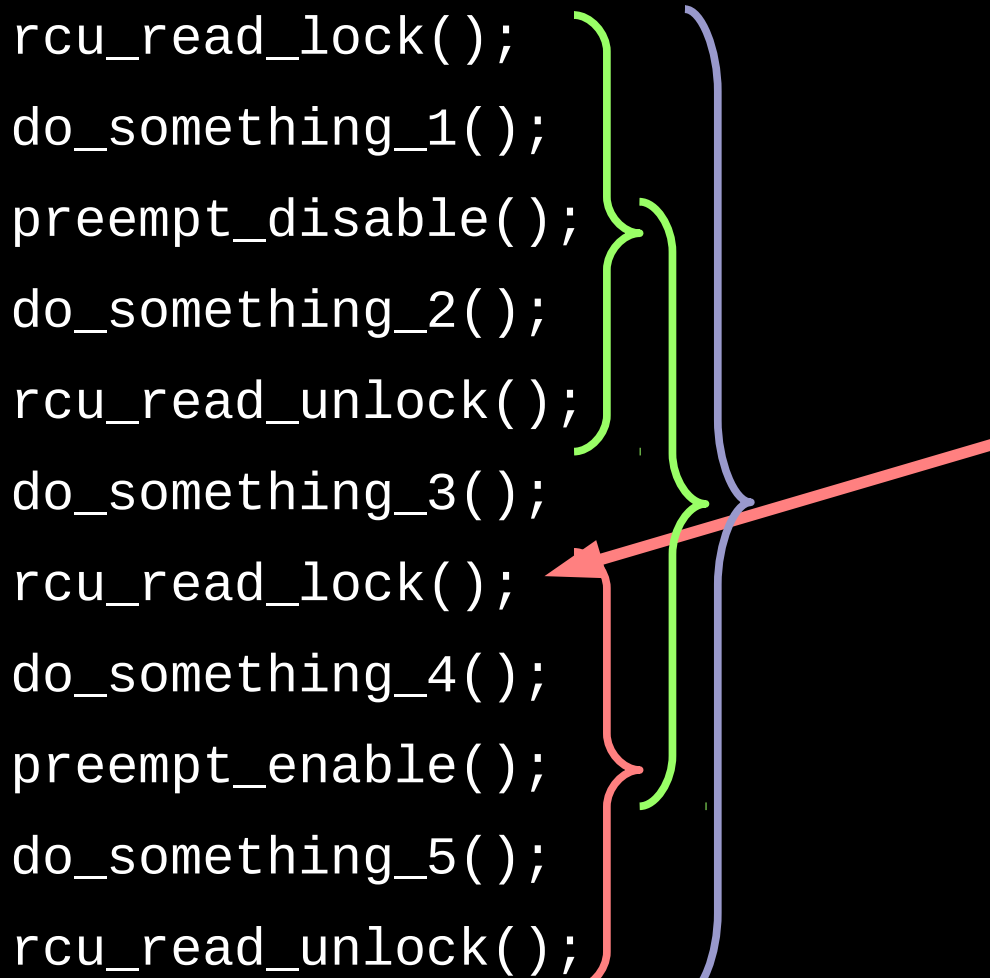
X print_other_cpu_stall(): rcu_for_each_leaf_node() to for_each_leaf_node_possible_cpu() loop:
    ...
    struct task_struct *t;
    int idx;
    ... ← lock acq ←
    idx = rcu_seq_cur(rnp->gp_seq) & 0x1;
    ...

    t = cpu_curr(cpu);
    if ((rnp->gsmask & leaf_node_cpu_bit(rnp, cpu)) ||

```

But Eight Pages is Still Imperfect for PREEMPT=y!!!

```
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()`!!!

OK, Maybe Not A Hard Failure, But...

OK, Maybe Not A Hard Failure, But...

```
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 might not because of the prior `rcu_read_unlock()`!!!

So we need to carry more state.

OK, Maybe Not A Hard Failure, But...

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

Brian W. Kernighan

OK, Maybe Not A Hard Failure, But...

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

Brian W. Kernighan

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

OK, Maybe Not A Hard Failure, But...

Debugging... writing
the code... before, if
you... as
poss... n, not
t.

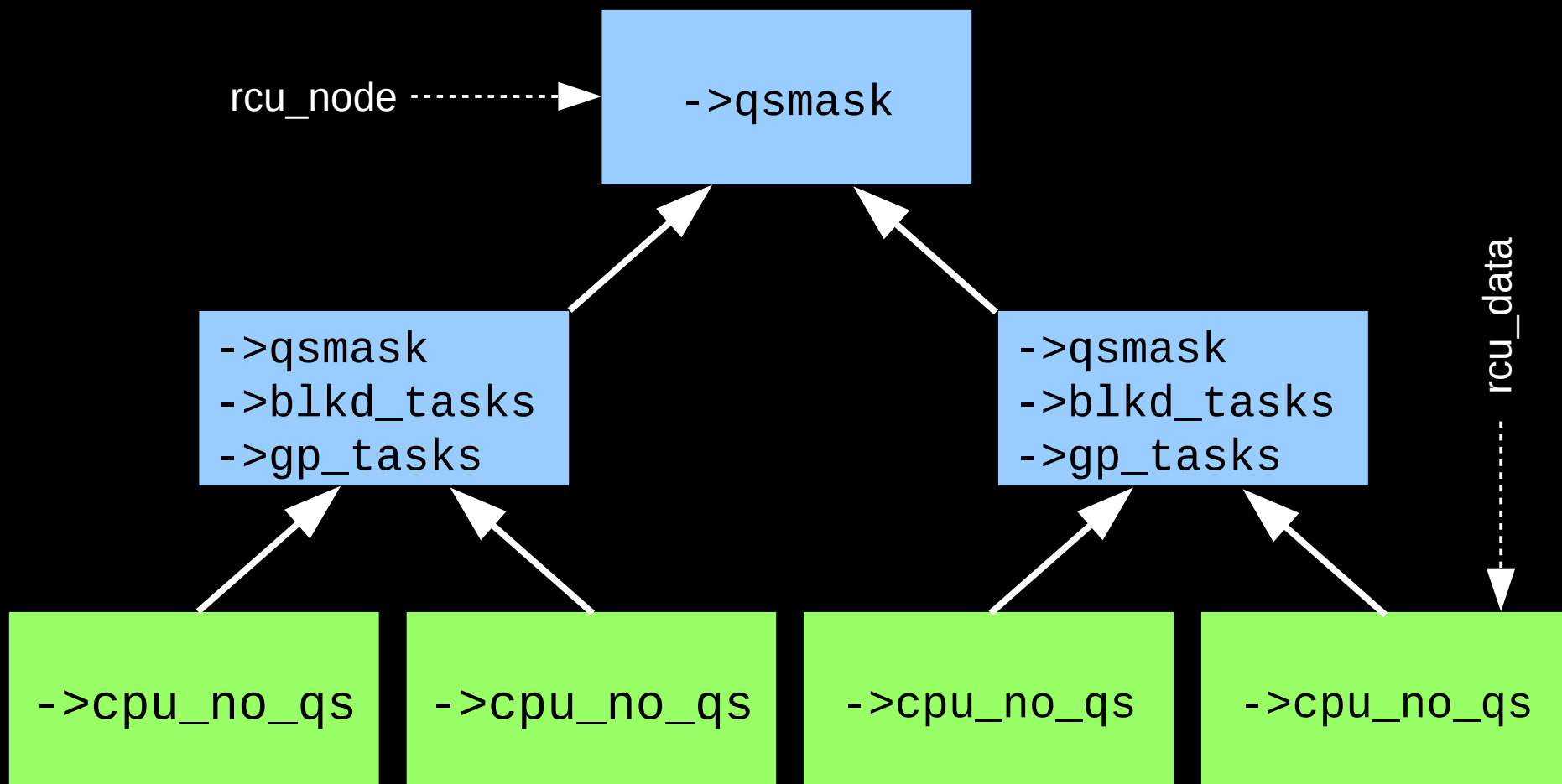
Fail
RCU read-side critical sections
linked by preempt-disable...
Excessive complexity!!!

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

Back to the drawing board...

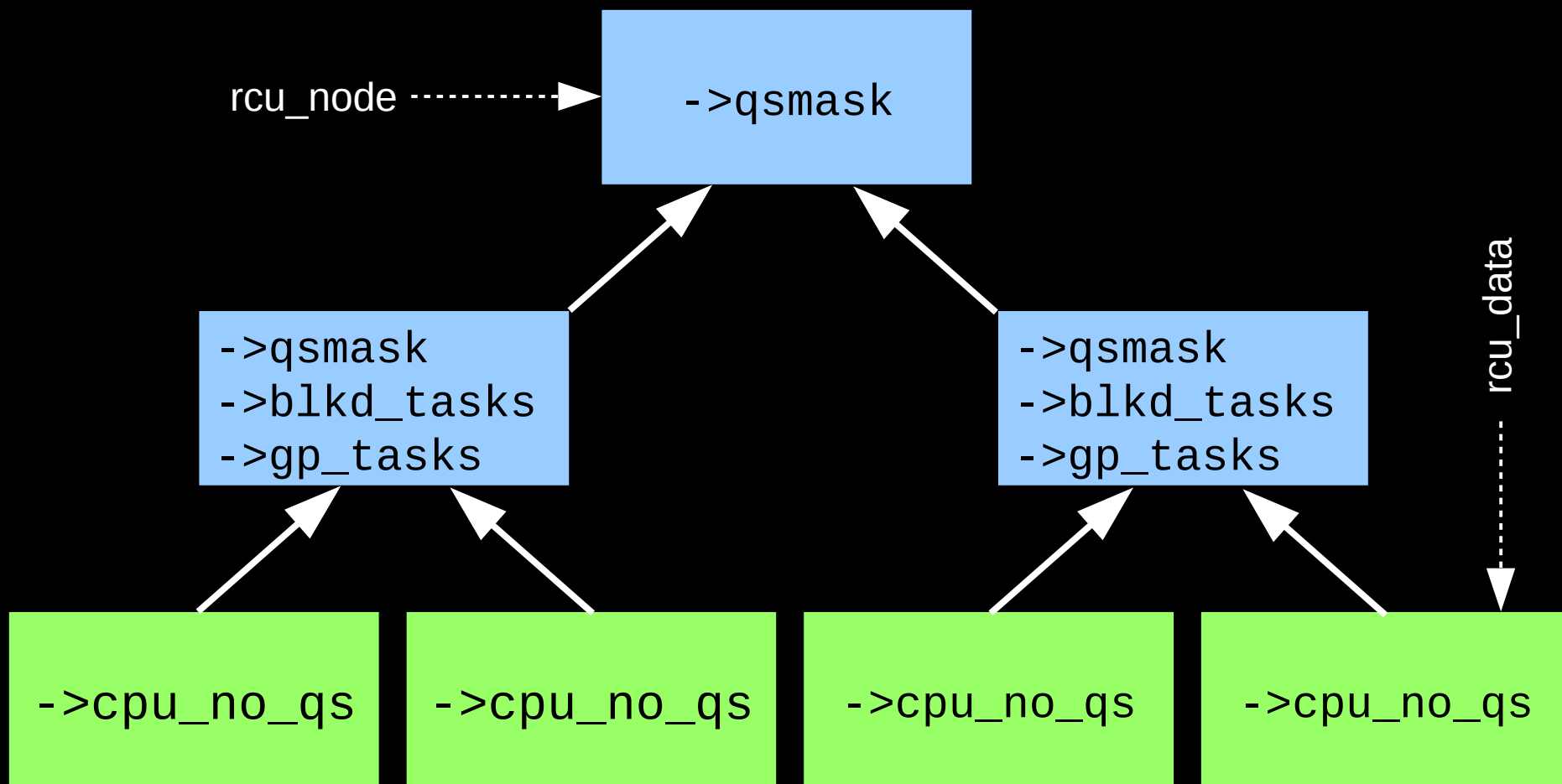
Possible Solution: Defer `rcu_read_unlock()` Dequeue

Preempted Tasks Queued on Leaf rcu_node Structure Running Task A



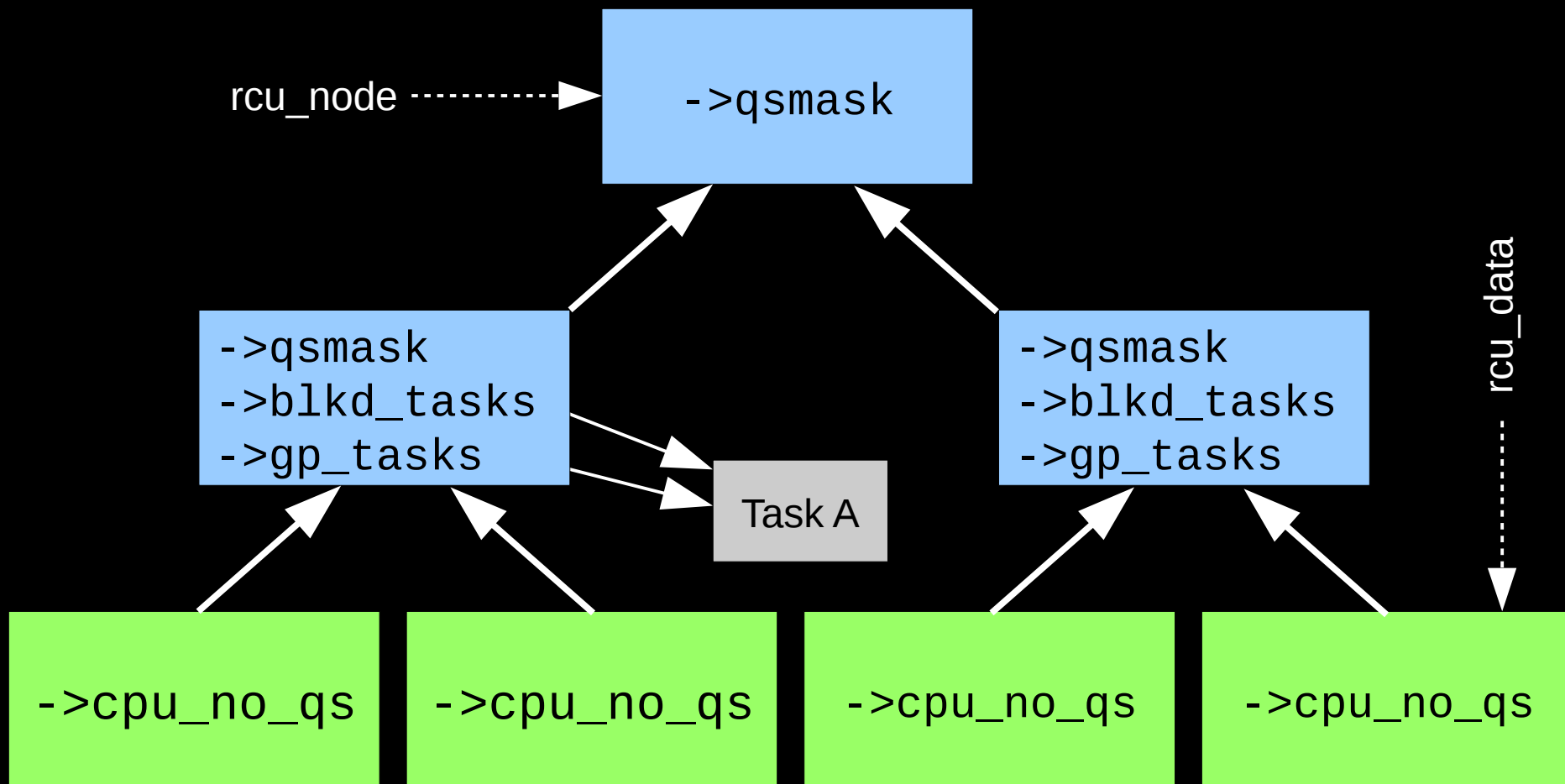
Preempted Tasks Queued on Leaf rcu_node Structure

Task A Preempted, Blocks Current Grace Period



Preempted Tasks Queued on Leaf rcu_node Structure

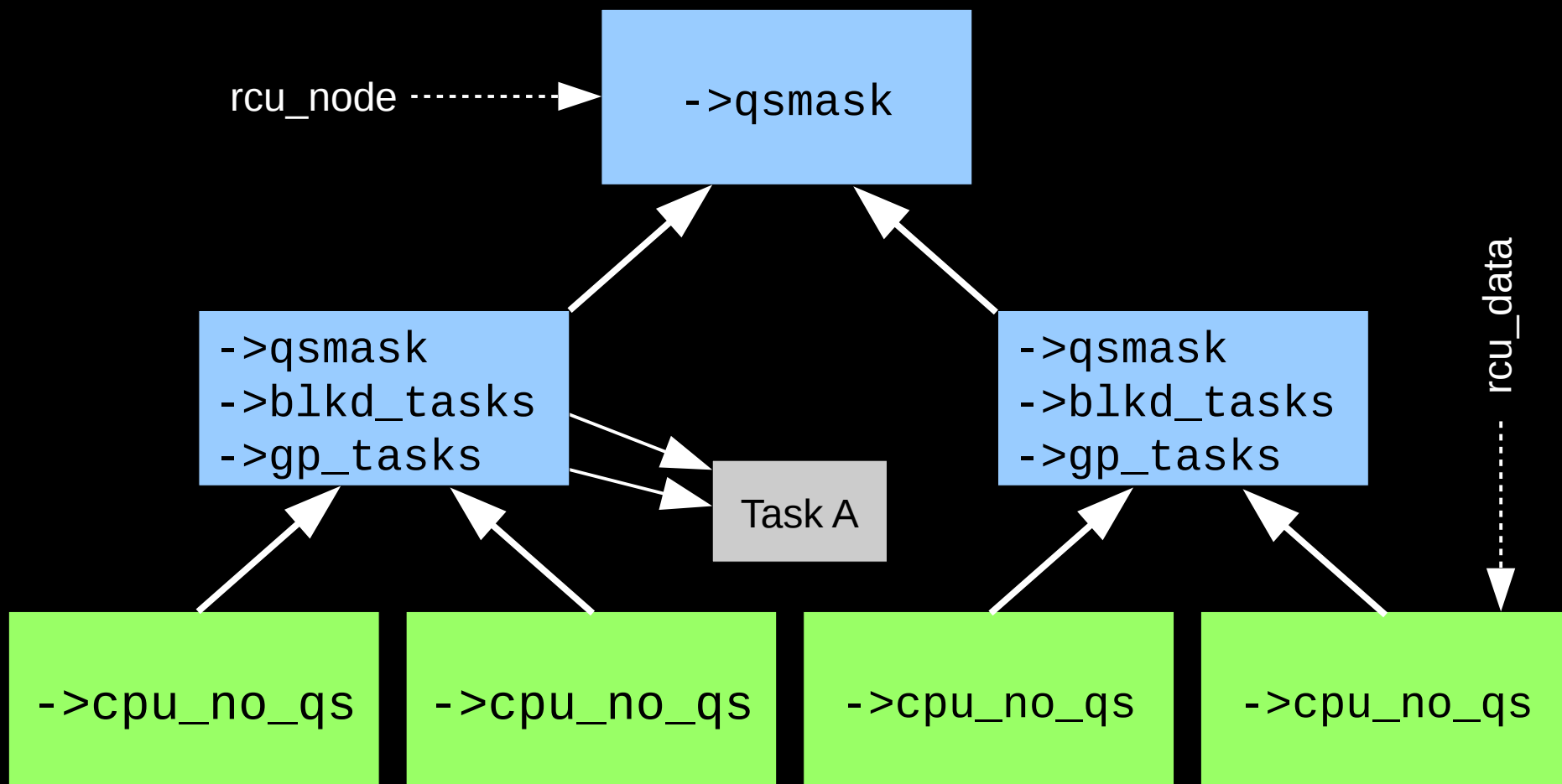
Task A Preempted, Blocks Current Grace Period



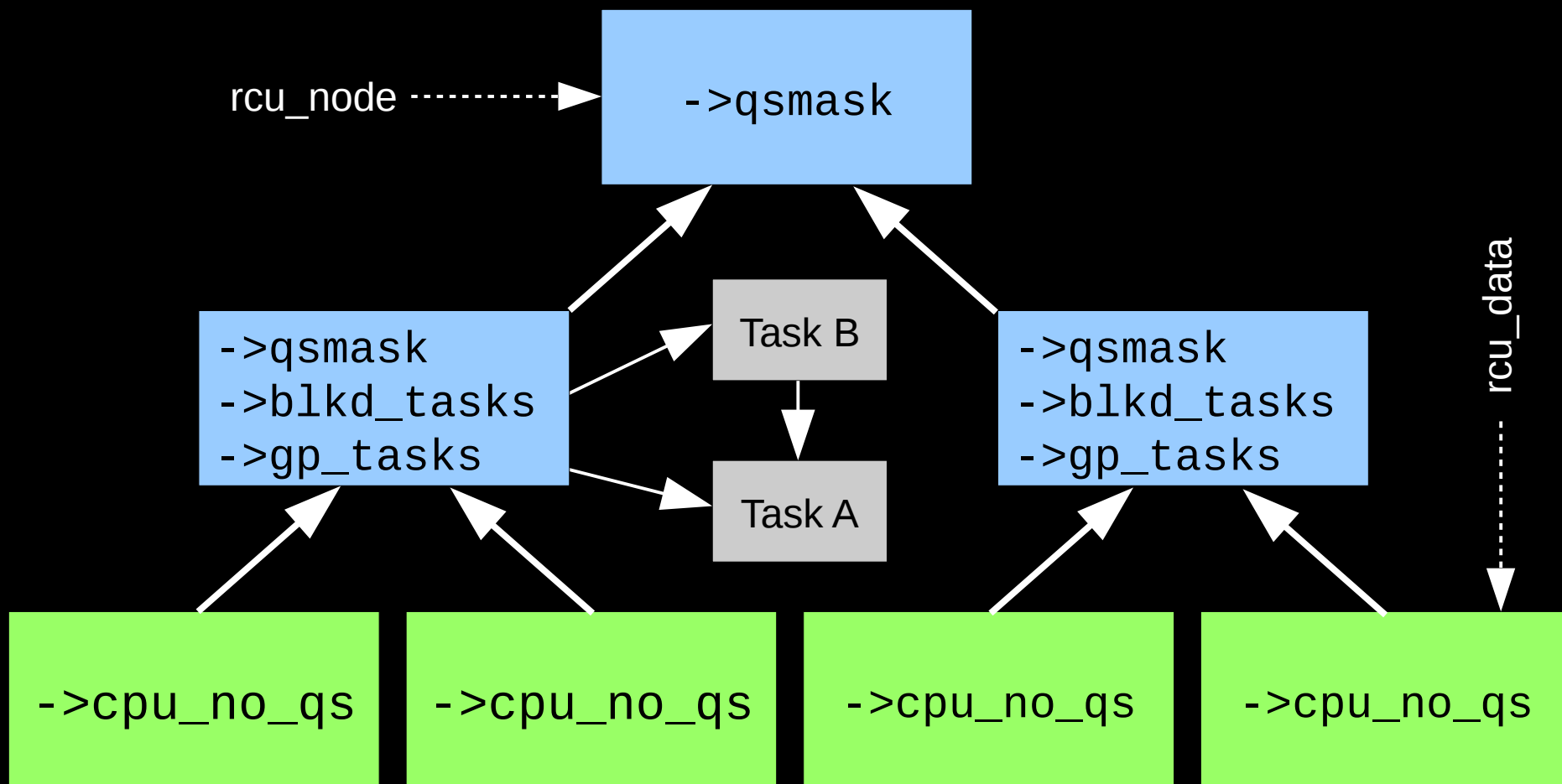
CPU switches to Task B

Preempted Tasks Queued on Leaf `rcu_node` Structure

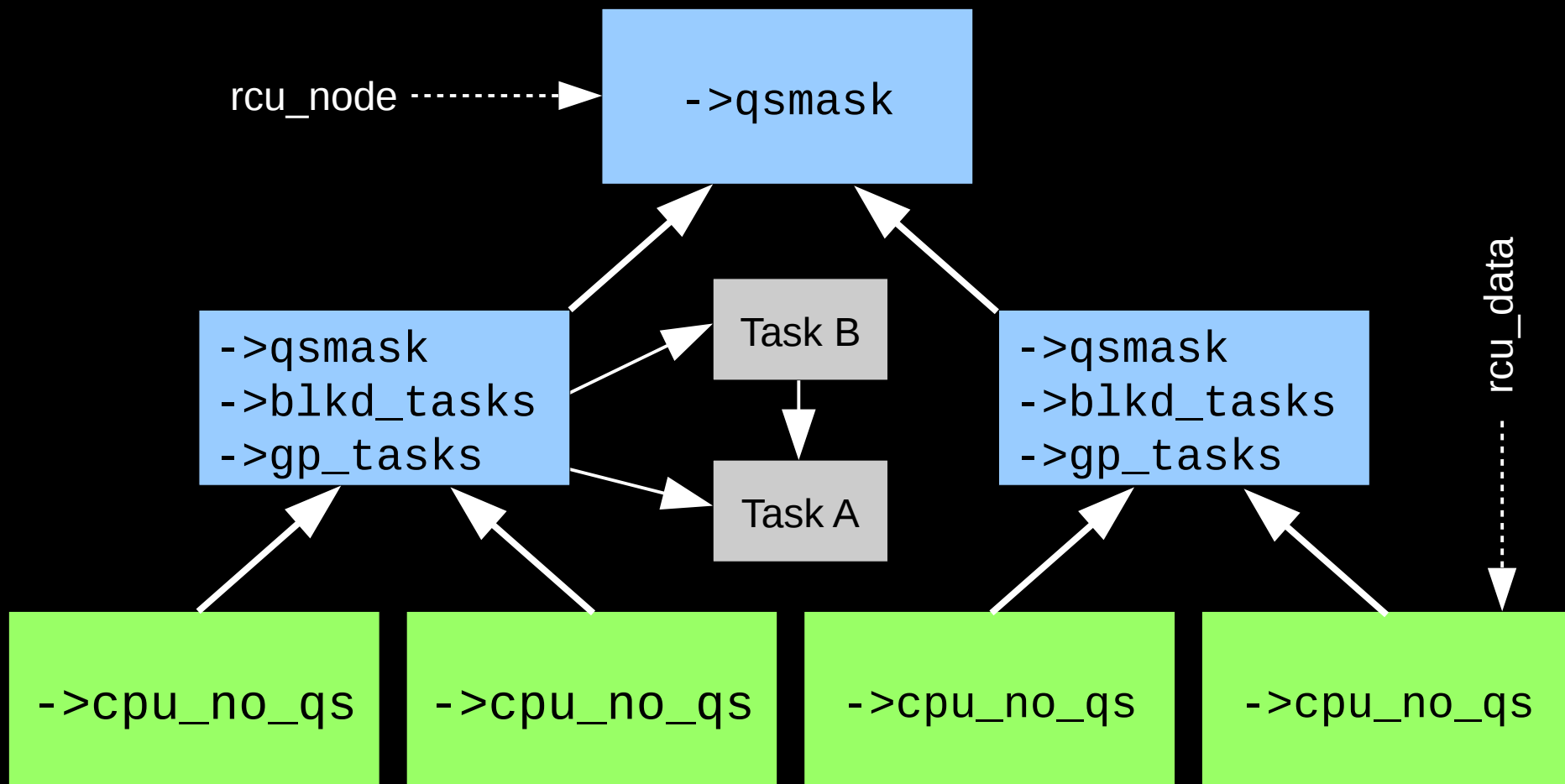
Task B's priority is lowered, Task A resumes



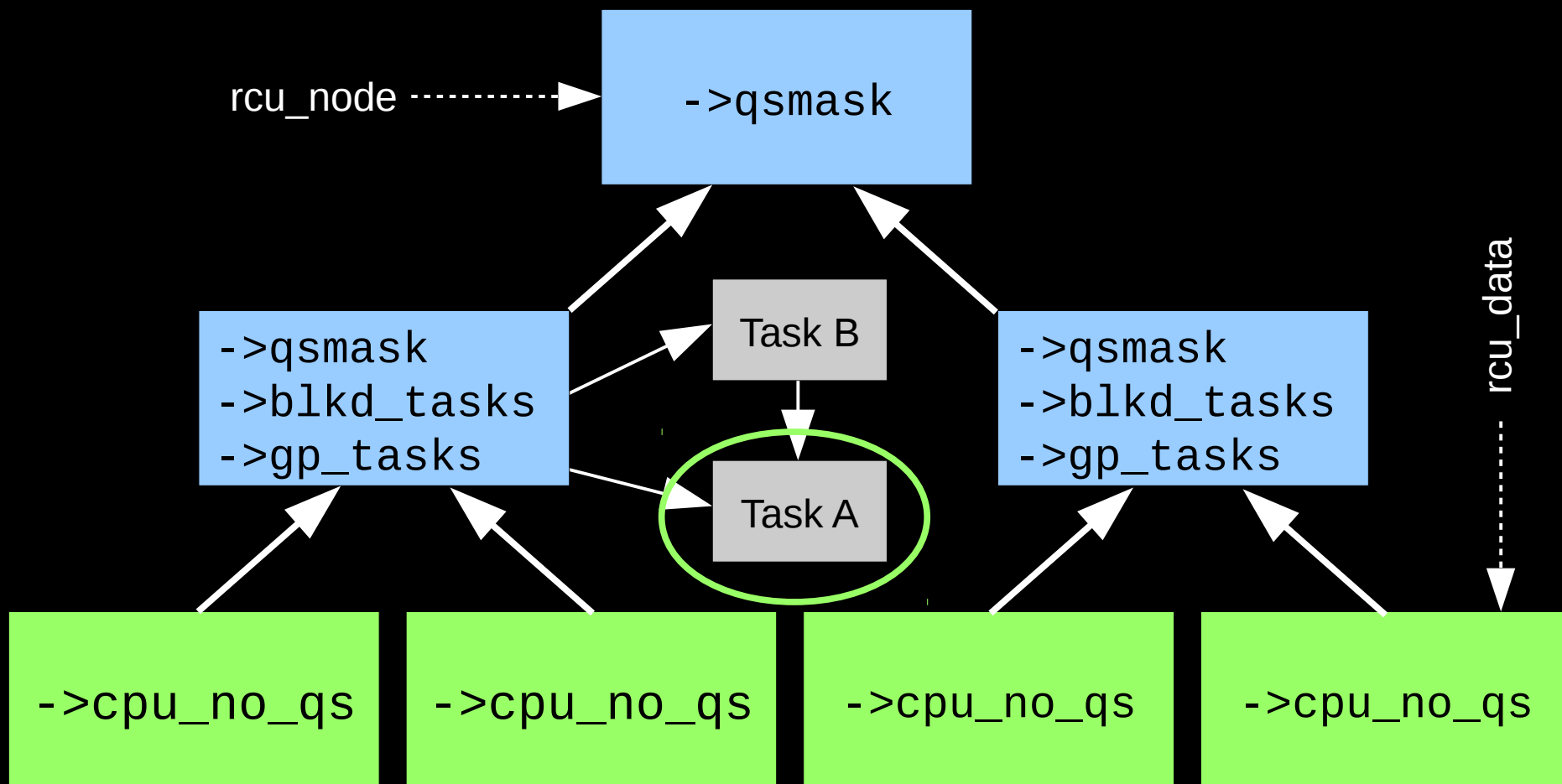
Preempted Tasks Queued on Leaf rcu_node Structure Task A Blocks Current Grace Period, Task B Does Not



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

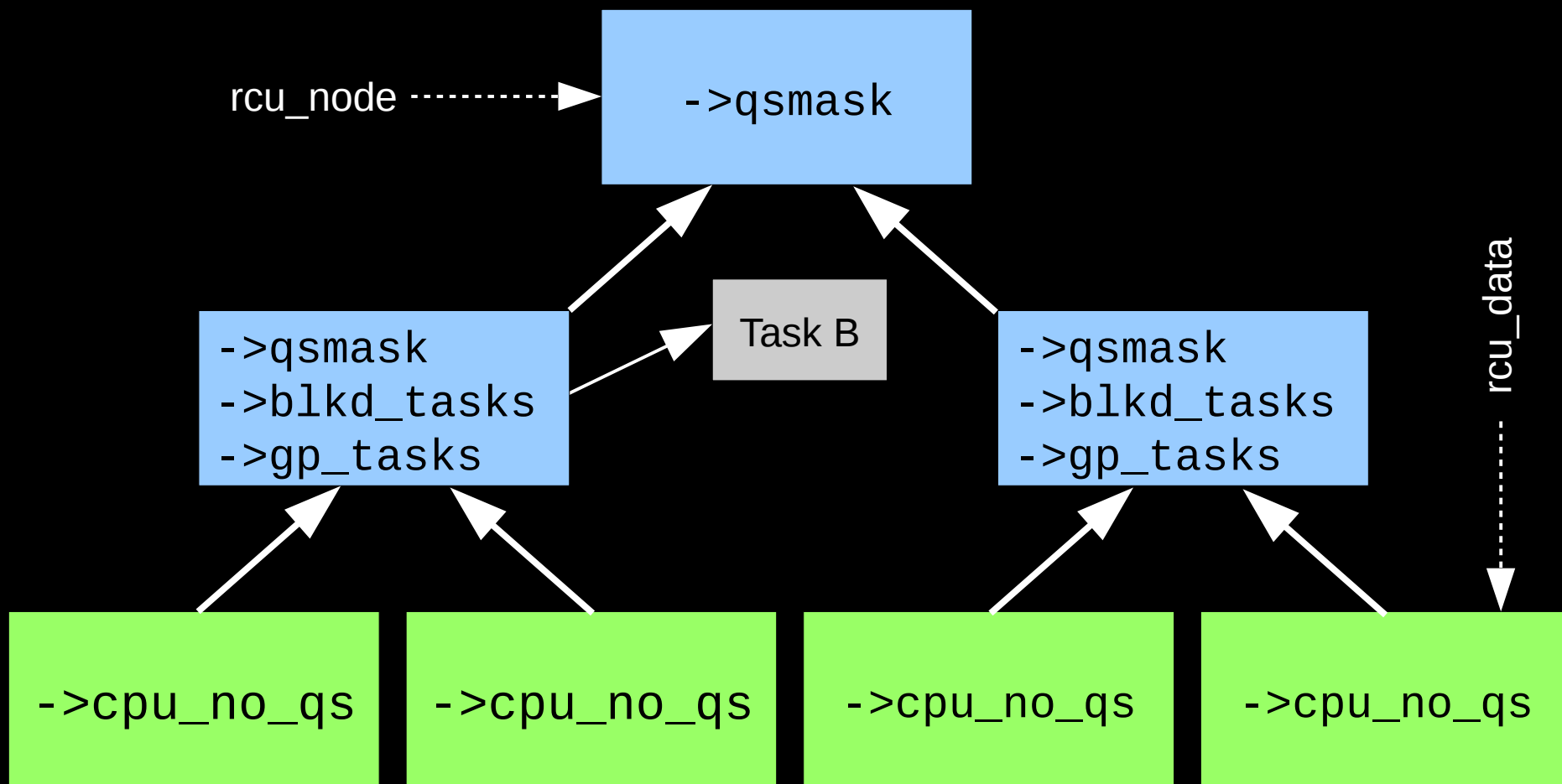


Preempted Tasks Queued on Leaf rcu_node Structure Task A No Longer Blocks Current Grace Period



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`

More Detail on Current 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():***
 - 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***

More Detail on Current 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():***
 - 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***

Key point: RCU already knows all about ->blkd_tasks and ->gp_tasks
So defer ->blkd_tasks removal until enabled!!!

Defer Dequeuing to Defer 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???

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 leave
task queued on `->blkd_tasks[]...`

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

One big reader, as required!!!

Scheduling Things For Later

- Leverage `local_bh_enable()`
 - If bottom halves disabled, `raise_softirq()`
 - Very cheap from interrupt handler
 - `RCU_SOFTIRQ` will be invoked shortly after `local_bh_enable()`
 - But preemption might still be disabled...
- Leverage `preempt_enable()` and/or interrupt enabling
 - Tell the scheduler to force a context switch
 - Even if no context switch is needed
 - Context switch informs RCU: Cannot happen until after fully enabled
- This can be expensive!!!
 - Fortunately, only happens when RCU reader is long and/or preempted

Scheduling Things For Later

- Leverage `local_bh_enable()`
 - If bottom halves disabled, `raise_softirq()`
 - Very cheap from interrupt handler
 - `RCU_SOFTIRQ` will be invoked shortly after `local_bh_enable()`
 - But preemption might still be disabled...
- Leverage `preempt_enable()` and/or interrupt enabling
 - Tell the scheduler to force a context switch
 - Even if no context switch is needed
 - Context switch informs RCU: Cannot happen until after fully enabled
- This can be expensive!!!
 - Fortunately, only happens when RCU reader is long and/or preempted
- Time to start coding!!!

Scheduling Things For Later

- Leverage `local_bh_enable()`
 - If bottom halves disabled, `raise_softirq()`
 - Very cheap from interrupt handler
 - `RCU_SOFTIRQ` will be invoked shortly after `local_bh_enable()`
 - But preemption might still be disabled...
- Leverage `preempt_enable()` and/or interrupt enabling
 - Tell the scheduler to force a context switch
 - Even if no context switch is needed
 - Context switch informs RCU: Cannot happen until after fully enabled
- This can be expensive!!!
 - Fortunately, only happens when RCU reader is long and/or preempted
- Time to start coding!!! (Again!)

Defer rcu_read_unlock() Current-Task Dequeue (Part of a Page, Down to Three 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;

    (x)
    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

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



Adjusts for RCU-bh

The Full Set of Commits: In Theory, Anyway...

- 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



Adjusts for RCU-bh

Merge Grace-Period Counters (1/2)

1. de30ad512a66 rcu: Introduce grace-period sequence numbers
2. dee4f42298bb rcu: Move rcu_gp_slow() to ->gp_seq
3. 17ef2fe97c8c rcu: Make rcutorture's batches-completed API use ->gp_seq
4. 78c5a67f1788 rcu: Convert rcu_check_gp_kthread_starvation() to GP sequence number
5. c9a24e2d0c7d rcu: Make quiescent-state reporting use ->gp_seq
6. e4be81a2ed3a rcu: Convert conditional grace-period primitives to ->gp_seq
7. 67e14c1e39d2 rcu: Move RCU's grace-period-change code to ->gp_seq
8. a66ae8ae35de rcu: Convert rcu_gpnum_ovf() to ->gp_seq
9. e05720b0977b rcu: Move rcu_implicit_dynticks_qs() to ->gp_seq
10. 03c8cb765a74 rcu: Move rcu_try_advance_all_cbs() to ->gp_seq
11. e0da2374c388 rcu: Move rcu_nocb_gp_get() to ->gp_seq
12. ba04107fc901 rcu: Move rcu_gp_in_progress() to ->gp_seq
13. 8aa670cdacc1 rcu: Convert ->rcu_iw_gpnum to ->gp_seq
14. d43a5d32e125 rcu: Convert ->completedqs to ->gp_seq
15. 29365e563b1e rcu: Convert grace-period requests to ->gp_seq
16. 471f87c3d91b rcu: Make RCU CPU stall warnings use ->gp_seq
17. ae8c82644b2c rcutorture: Convert rcutorture_get_gp_data() to ->gp_seq
18. 7a1d0f23ad70 rcu: Move from ->need_future_gp[] to ->gp_seq_needed
19. ab5e869c1f7a rcu: Make rcu_nocb_wait_gp() check if GP already requested
20. 477351f7829d rcu: Convert rcu_grace_period tracepoint to gp_seq

Merge Grace-Period Counters (2/2)

1. abd13fdd9516 rcu: Convert rcu_future_grace_period tracepoint to gp_seq
2. 598ce09480ef rcu: Convert rcu_preempt_task tracepoint to ->gp_seq
3. 865aa1e08d8a rcu: Convert rcu_unlock_preempted_task tracepoint to ->gp_seq
4. db023296f011 rcu: Convert rcu_quiescent_state_report tracepoint to ->gp_seq
5. fee5997c1756 rcu: Convert rcu_fqs tracepoint to ->gp_seq
6. ff3bb6f4d062 rcu: Remove ->gpnum and ->completed
7. e44e73ca47b4 rcu: Make simple callback acceleration refer to rdp->gp_seq_needed
8. 5b55072f22ba rcu: Produce last "CleanupMore" trace only if late-breaking request
9. 5ca0905f6787 rcu: Fix cpustart tracepoint gp_seq number
10. 2e3e5e550101 rcu: Make rcu_start_this_gp() check for grace period already started
11. d72193123c81 rcutorture: Correctly handle grace-period sequence wrap
12. 3d18469a2bb3 rcu: Regularize resetting of rcu_data wrap indicator
13. b73de91d6a4c rcu: Rename the grace-period-request variables and parameters
14. 2ee5aca54622 rcu: Make rcu_seq_diff() more exact
15. adbccddb4a16 rcu: Fix rcu_{node,data} comments about gp_seq_needed

Funnel-Lock Grace-Period Start

1. a2165e416878 rcu: Don't funnel-lock above leaf node if GP in progress
2. df2bf8f7f776 rcu: Use better variable names in funnel locking loop
3. 226ca5e76692 rcu: Identify grace period is in progress as we advance up the tree

Fix Pre-Existing rcutorture Failures

1. 962aff03c315 rcu: Clean up handling of tasks blocked across full-rcu_node offline
2. c50cbe535c97 rcu: Fix an obsolete ->qsmaskinit comment
3. 5554788e1d42 rcu: Suppress false-positive offline-CPU lockdep-RCU splat
4. fece27760ff5 rcu: Suppress false-positive preempted-task splats
5. 99990da1b3c0 rcu: Suppress more involved false-positive preempted-task splats
6. 0b107d24d936 rcu: Suppress false-positive splats from mid-init task resume
7. ec2c29765a4a rcu: Fix grace-period hangs from mid-init task resume
8. 1e64b15a4b10 rcu: Fix grace-period hangs due to race with CPU offline
9. c7cd161ecb21 rcu: Assign higher prio to RCU threads if rcutorture is built-in
10. 450efca7182a rcutorture: Disable RT throttling for boost tests
11. 3b745c8969c7 rcutorture: Make boost test more robust
12. 4babd855fd61 rcutorture: Add support to detect if boost kthread prio is too low
13. e746b558572e rcutorture: Warn on bad torture type for built-in tests
14. f0288064425f rcuperf: Warn on bad perf type for built-in tests
15. 894d45bbf7e7 rcu: Convert rcu_state.ofl_lock to raw_spinlock_t

Add Debugging Code

1. 4bc8d55574dd rcu: Add debugging info to assertion
2. 26d950a94513 rcu: Diagnostics for grace-period startup hangs
3. c74859d1eb2d rcu: Make rcu_report_unblock_qs_rnp() warn on violated preconditions
4. 77cfc7bf24ba rcu: Fix typo and add additional debug
5. 1f3e5f51b933 rcu: Add RCU-preempt check for waiting on newly onlined CPU
6. f34f2f5852e5 rcu: Move grace-period pre-init delay after pre-init
7. ff3cee39088b rcu: Add up-tree information to dump_blk_d_tasks() diagnostics
8. 577389423187 rcu: Add CPU online/offline state to dump_blk_d_tasks()
9. fea3f222d352 rcu: Record ->gp_state for both phases of grace-period initialization
10. f2e2df59786d rcu: Add diagnostics for offline CPUs failing to report QS
11. b06ae25a1e2b rcu: Use RCU CPU stall timeout for rcu_check_gp_start_stall()
12. 47199a081253 rcu: Add diagnostics for rcutorture writer stall warning
13. 89b4cd4b9ebf rcu: Print stall-warning NMI dyntick state in hexadecimal
14. 028be12b294e rcutorture: Change units of onoff_interval to jiffies
15. 691960197e8d rcu: Add state name to show_rcu_gp_kthreads() output
16. c669c014d1da rcu: Add jiffies-since-GP-activity to show_rcu_gp_kthreads()
17. 7ae47dfb7e2a rcu: Improve diagnostics for failed RCU grace-period start

Add rcutorture Quiescent-State Deferral Tests (1/3)

1. 6b06aa723ed7 rcutorture: Extract common code from rcu_torture_reader()
2. 8da9a59523b6 rcutorture: Use atomic increment for n_rcu_torture_timers
3. 3025520ec424 rcutorture: Use per-CPU random state for rcu_torture_timer()
4. 241b42522abb rcutorture: Make rcu_torture_timer() use rcu_torture_one_read()
5. 2397d072f76b rcutorture: Handle extended read-side critical sections
6. bf1bef50bee1 rcutorture: Emphasize testing of single reader protection type
7. 444da518fd55 rcutorture: Force occasional reader waits
8. 1b27291b1ea4 rcutorture: Add forward-progress tests for RCU grace periods
9. 119248bec9d3 rcutorture: Also use GP sequence to judge forward progress
10. 152f4afb58 rcutorture: Avoid no-test complaint if too few forward-progress tries
11. 08a7a2ec6834 rcutorture: Vary forward-progress test interval
12. 9fdcb9afe082 rcutorture: Add self-propagating callback to forward-progress testing
13. 3cff54a830f7 rcutorture: Increase rcu_read_delay() longdelay_ms
14. 1e69676592ed rcutorture: Limit reader duration if irq or bh disabled
15. fecad5091f35 rcutorture: Reduce priority of forward-progress testing
16. c04dd09bd38c rcutorture: Adjust number of reader kthreads per CPU-hotplug operations
17. f4de46ed5bbc rcutorture: Print forward-progress test interval on error
18. 474e59b476b3 rcutorture: Check GP completion at stutter end
19. 7c590fcca66b rcutorture: Maintain self-propagating CB only during forward-progress test
20. c0335743c5d8 rcutorture: Test extended "rcu" read-side critical sections

Add rcutorture Quiescent-State Deferral Tests (2/3)

1. 2ceebc035082 rcutorture: Add RCU-bh and RCU-sched support for extended readers
2. 72ce30dd1f9b rcu: Stop testing RCU-bh and RCU-sched
3. c770c82a2382 rcutorture: Remove the "rcu_bh" and "sched" torture types
4. 620d246065cd rcuperf: Remove the "rcu_bh" and "sched" torture types
5. de3875d30233 rcu: Remove now-unused rcutorture APIs
6. c116dba68d19 rcutorture: Dump reader protection sequence if failures or close calls
7. 4871848531af rcutorture: Add call_rcu() flooding forward-progress tests
8. fc6f9c57787e rcutorture: Remove cbflood facility
9. 6b3de7a172bc rcutorture: Break up too-long rcu_torture_fwd_prog() function
10. 5ab7ab8362fa rcutorture: Affinity forward-progress test to avoid housekeeping CPUs
11. 61670adcb4a9 rcutorture: Prepare for asynchronous access to rcu_fwd_startat
12. e0aff9735557 rcutorture: Dump grace-period diagnostics upon forward-progress OOM
13. bfcfcffc5f23 rcu: Print per-CPU callback counts for forward-progress failures
14. 8dd3b54689d9 rcutorture: Print GP age upon forward-progress failure
15. 1a682754c7ed rcutorture: Print histogram of CB invocation at OOM time
16. c51d7b5e6c94 rcutorture: Print time since GP end upon forward-progress failure
17. 73d665b1410a rcutorture: Print forward-progress test age upon failure
18. 2667ccce9328 rcutorture: Recover from OOM during forward-progress tests
19. 2e57bf97a685 rcutorture: Use 100ms buckets for forward-progress callback histograms
20. 5ac7cdc29897 rcutorture: Don't do busted forward-progress testing

Add rcutorture Quiescent-State Deferral Tests (3/3)

1. da3d56fb3dd6 rcu: Add sysrq rcu_node-dump capability
2. a0a2c92a5543 rcutorture: Record grace periods in forward-progress histogram

Remove RCU-bh & RCU-sched and Simplify (1/6)

1. 65cfe3583b61 rcu: Define RCU-bh update API in terms of RCU
2. 82fcecfa8185 rcu: Update comments and help text for no more RCU-bh updaters
3. 45975c7d21a1 rcu: Define RCU-sched API in terms of RCU for Tree RCU PREEMPT builds
4. 709fdce7545c rcu: Express Tiny RCU updates in terms of RCU rather than RCU-sched
5. 358be2d3685c rcu: Remove RCU_STATE_INITIALIZER()
6. ec5dd444b678 rcu: Eliminate rcu_state structure's ->call field
7. da1df50d1617 rcu: Remove rcu_state structure's ->rda field
8. 16fc9c600b3c rcu: Remove rcu_state_p pointer to default rcu_state structure
9. 2280ee5a7d3e rcu: Remove rcu_data_p pointer to default rcu_data structure
10. b50912d0b5e0 rcu: Remove rsp parameter from rcu_report_qs_rnp()
11. aff4e9ede52b rcu: Remove rsp parameter from rcu_report_qs_rsp()
12. 139ad4da5ab5 rcu: Remove rsp parameter from rcu_report_unblock_qs_rnp()
13. 33085c469aea rcu: Remove rsp parameter from rcu_report_qs_rdp()
14. de8e87305a1a rcu: Remove rsp parameter from rcu_gp_in_progress()
15. 336a4f6c451e rcu: Remove rsp parameter from rcu_get_root()
16. ad3832e974eb rcu: Remove rsp parameter from record_gp_stall_check_time()
17. 8fd119b6522f rcu: Remove rsp parameter from rcu_check_gp_kthread_starvation()
18. 33dbdbf02538 rcu: Remove rsp parameter from rcu_dump_cpu_stacks()
19. e1741c69d427 rcu: Remove rsp parameter from rcu_stall_kick_kthreads()
20. a91e7e58b101 rcu: Remove rsp parameter from print_other_cpu_stall()

Remove RCU-bh & RCU-sched and Simplify (2/6)

1. 4e8b8e08f931 rcu: Remove rsp parameter from print_cpu_stall()
2. ea12ff2b7d97 rcu: Remove rsp parameter from check_cpu_stall()
3. 3481f2eab095 rcu: Remove rsp parameter from rcu_future_gp_cleanup()
4. 532c00c97f16 rcu: Remove rsp parameter from rcu_gp_kthread_wake()
5. 02f501423d0d rcu: Remove rsp parameter from rcu_accelerate_cbs()
6. c6e09b97b933 rcu: Remove rsp parameter from rcu_accelerate_cbs_unlocked()
7. 834f56bf54e8 rcu: Remove rsp parameter from rcu_advance_cbs()
8. c7e48f7ba382 rcu: Remove rsp parameter from __note_gp_changes()
9. 15cabdffbbf6 rcu: Remove rsp parameter from note_gp_changes()
10. 22212332c1f3 rcu: Remove rsp parameter from rcu_gp_slow()
11. 0854a05c9fa5 rcu: Remove rsp parameter from rcu_gp_kthread() and friends
12. 8087d3e3c453 rcu: Remove rsp parameter from rcu_check_quiescent_state()
13. 780cd590836f rcu: Remove rsp parameter from CPU hotplug functions
14. 5bb5d09cc4f8 rcu: Remove rsp parameter from rcu_do_batch()
15. e9ecb780fe7d rcu: Remove rsp parameter from force-quiescent-state functions
16. b96f9dc4fb64 rcu: Remove rsp parameter from rcu_check_gp_start_stall()
17. b049fdf8e3b9 rcu: Remove rsp parameter from __rcu_process_callbacks()
18. 5c7d89676bc5 rcu: Remove rsp parameter from __call_rcu() and friend
19. 98ece508b545 rcu: Remove rsp parameter from __rcu_pending()
20. 8344b871b1d5 rcu: Remove rsp parameter from _rcu_barrier() and friends

Remove RCU-bh & RCU-sched and Simplify (3/6)

1. 53b46303da84 rcu: Remove rsp parameter from rcu_boot_init_percpu_data() and friends
2. b8bb1f63cf9a rcu: Remove rsp parameter from rcu_init_one() and friends
3. a2887cd85f38 rcu: Remove rsp parameter from rcu_print_detail_task_stall()
4. 81ab59a3ad86 rcu: Remove rsp parameter from dump_blk_d_tasks() and friend
5. 6dbfdc1409cf rcu: Remove rsp parameter from rcu_spawn_one_boost_kthread()
6. b21ebed95101 rcu: Remove rsp parameter from print_cpu_stall_info()
7. 4580b0541bea rcu: Remove rsp parameter from no-CBs CPU functions
8. 63d4c8c97948 rcu: Remove rsp parameter from expedited grace-period functions
9. aedf4ba98416 rcu: Remove rsp parameter from rcu_node tree accessor macros
10. 88d1bead858d rcu: Remove rcu_data structure's ->rsp field
11. 564a9ae6046c rcu: Remove last non-flavor-traversal rsp local variable from tree_plugin.h
12. b97d23c51c9f rcu: Remove for_each_rcu_flavor() flavor-traversal macro
13. f7dd7d44fd2d rcu: Simplify rcutorture_get_gp_data()
14. 7cba4775ba79 rcu: Restructure rcu_check_gp_kthread_starvation()
15. 4c6ed43708bb rcu: Eliminate stall-warning use of rsp
16. 9cbc5b97029b rcu: Eliminate grace-period management code use of rsp
17. 3c779dfef2c4 rcu: Eliminate callback-invocation/invoke use of rsp
18. 67a0edbf3c4d rcu: Eliminate quiescent-state and grace-period-nonstart use of rsp
19. ec9f5835f74c rcu: Eliminate RCU-barrier use of rsp
20. eb7a6653887b rcu: Eliminate initialization-time use of rsp

Remove RCU-bh & RCU-sched and Simplify (4/6)

1. 8ff0b9078091 rcu: Fix typo in force_qs_rnp()'s parameter's parameter
2. 4e95020cdd34 rcu: Inline increment_cpu_stall_ticks() into its sole caller
3. 4c7e9c1434c6 rcu: Consolidate RCU-bh update-side function definitions
4. a8bb74acd8ef rcu: Consolidate RCU-sched update-side function definitions
5. 2bd8b1a2afc4 rcu: Clean up flavor-related definitions and comments in rcupdate.h
6. aff5f0369e31 rcu: Clean up flavor-related definitions and comments in rclist.h
7. df8561a0d7e4 rcu: Clean up flavor-related definitions and comments in rcupdate_wait.h
8. 8c1cf2da6f8a rcu: Clean up flavor-related definitions and comments in Kconfig
9. 7f87c036fea3 rcu: Clean up flavor-related definitions and comments in rcu.h
10. 62a1a945368f rcu: Clean up flavor-related definitions and comments in rcutorture.c
11. 6eb95cc4507a rcu: Clean up flavor-related definitions and comments in srcutree.h
12. 679d3f30923e rcu: Clean up flavor-related definitions and comments in tiny.c
13. 49918a54e63c rcu: Clean up flavor-related definitions and comments in tree.c
14. 8fa946d42855 rcu: Clean up flavor-related definitions and comments in tree_exp.h
15. 0ae86a272656 rcu: Clean up flavor-related definitions and comments in tree_plugin.h
16. 06462efc808c rcu: Clean up flavor-related definitions and comments in update.c
17. 4d232dfe1df3 rcu: Remove !PREEMPT code from rcu_note_voluntary_context_switch()
18. 395a2f097ebd rcu: Define rcu_all_qs() only in !PREEMPT builds
19. dd46a7882c2c rcu: Inline _rcu_barrier() into its sole remaining caller
20. 7e28c5af4ef6 rcu: Eliminate ->rcu_qs_ctr from the rcu_dynticks structure

Remove RCU-bh & RCU-sched and Simplify (5/6)

1. 31ab604bf323 rcu: Remove unused rcu_dynticks_snap() from Tiny RCU
2. cc72046cc3cc rcu: Merge rcu_dynticks structure into rcu_data structure
3. 0fd79e7521bc rcu: Switch ->tick_nohz_enabled_snap to rcu_data structure
4. 5998a75adbf4 rcu: Switch last accelerate/advance to rcu_data structure
5. c458a89e964d rcu: Switch lazy counts to rcu_data structure
6. 2dba13f0b6c2 rcu: Switch urgent quiescent-state requests to rcu_data structure
7. 4c5273bf2b5e rcu: Switch dyntick nesting counters to rcu_data structure
8. dc5a4f2932f1 rcu: Switch ->dynticks to rcu_data structure, remove rcu_dynticks
9. 8d8a9d0e7eda rcu: Remove obsolete ->dynticks_fqs and ->cond_resched_completed
10. 75a8f7224522 rcu: Remove unused rcu_state externs
11. 309ba859b950 rcu: Eliminate synchronize_rcu_mult()
12. d3ff3891b2ed rcu: Consolidate the RCU update functions invoked by sync.c
13. ee77e3c7a6e5 rcu: Rename and comment changes due to only one rcuo kthread per CPU
14. 8d72091d7397 rcu: Inline force_quiescent_state() into rcu_force_quiescent_state()
15. 841d84621ec5 rcu: Eliminate RCU_BH_FLAVOR and RCU_SCHED_FLAVOR
16. 8f9832faa774 rcu: Inline rcu_kthread_do_work() into its sole remaining caller
17. 9df74c360d7f rcu: Determine expedited-GP IPI handler at build time
18. e0cf0c15f468 rcu: Consolidate PREEMPT and !PREEMPT synchronize_rcu_expedited()
19. 8c705b1ca46a rcu: Consolidate PREEMPT and !PREEMPT synchronize_rcu()
20. b4f7db989227 rcu: Inline _synchronize_rcu_expedited() into synchronize_rcu_expedited()

Remove RCU-bh & RCU-sched and Simplify (6/6)

1. 004e0b8e9598 rcu: Discard separate per-CPU callback counts
2. f8e7680f01a2 rcu: Move rcu_cpu_kthread_task to rcu_data structure
3. c059f5df36fa rcu: Move rcu_cpu_kthread_status to rcu_data structure
4. dedda98c12b9 rcu: Remove unused rcu_cpu_kthread_loops per-CPU variable
5. 07c7c7c1370d rcu: Move rcu_cpu_has_work to rcu_data structure
6. 3de462dd756c rcu: Remove unused rcu_cpu_kthread_cpu per-CPU variable
7. e98376367759 rcu: Remove wrapper definitions for obsolete RCU update functions

Drive-By Optimizations

1. 18390aeae701 rcu: Make rcu_gp_cleanup() write only once to ->gp_flags
2. 8d672fa6bf68 rcu: Make rcu_init_new_rnp() stop upon already-set bit
3. 91f63ced7dc4 rcu: Replace smp_wmb() with smp_store_release() for stall check
4. 928164351e70 rcu: Prevent useless FQS scan after all CPUs have checked in
5. 17a8212b8de2 rcu: Remove failsafe check for lost quiescent state
6. e05121ba5b81 rcu: Remove CPU-hotplug failsafe from force-quiescent-state code path
7. 3949fa9bac09 rcu: Make rcu_read_unlock_special() static
8. 15651201fa05 rcu: Mark task as .need_qs less aggressively
9. 3b57a3994f33 rcu: Inline rcu_dynticks_momentary_idle() into its sole caller
10. 164ba3fc4864 rcu: Remove unused rcu_kick_nohz_cpu() function
11. ab6b82147f47 rcu: Remove unused local variable "cpu"
12. 95394e69c42f rcu: Remove "inline" from panic_on_rcu_stall() and rcu_blocking_is_gp()
13. eac45e586cd3 rcu: Remove "inline" from rcu_torture_print_module_parms()
14. 9622179519c5 rcu: Remove "inline" from rcu_perf_print_module_parms()
15. 51fbb910f52c rcu: Remove __maybe_unused from rcu_cpu_has_callbacks()
16. 117f683c6e01 rcu: Replace this_cpu_ptr() with __this_cpu_read()
17. f041d479a9cf rcu: Prevent needless ->gp_seq_needed update in __note_gp_changes()

And Murphy Will Always Be With Us!

Near Misses: Saved by Community Processes!

- Oday 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

Other Consequences

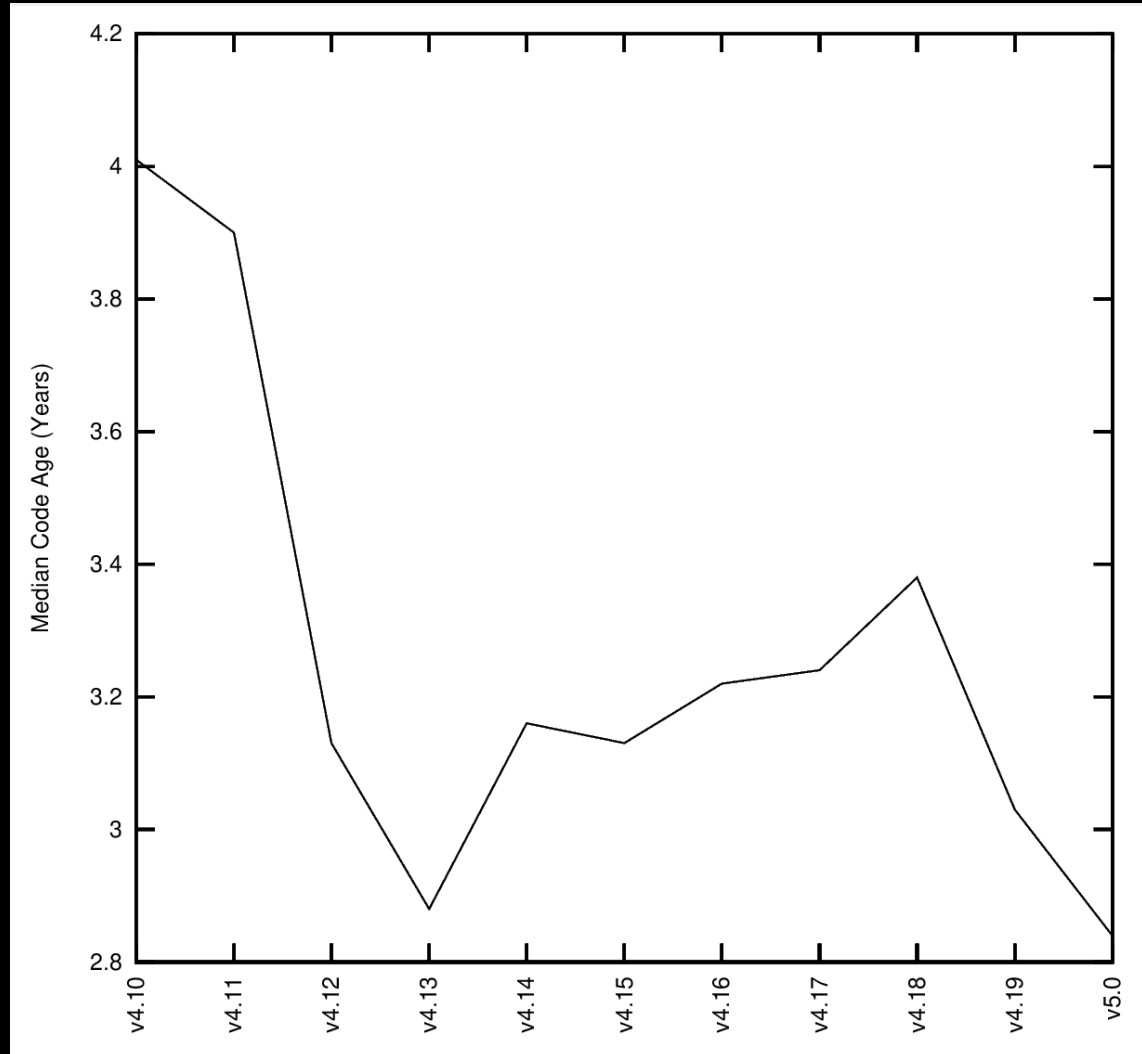
- What effect did this work have on RCU's reliability?
- According to rcutorture, it is actually *more* reliable
 - And rcutorture has become significantly more nasty
 - Which is a very good thing
- But this work did introduce some bugs

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...

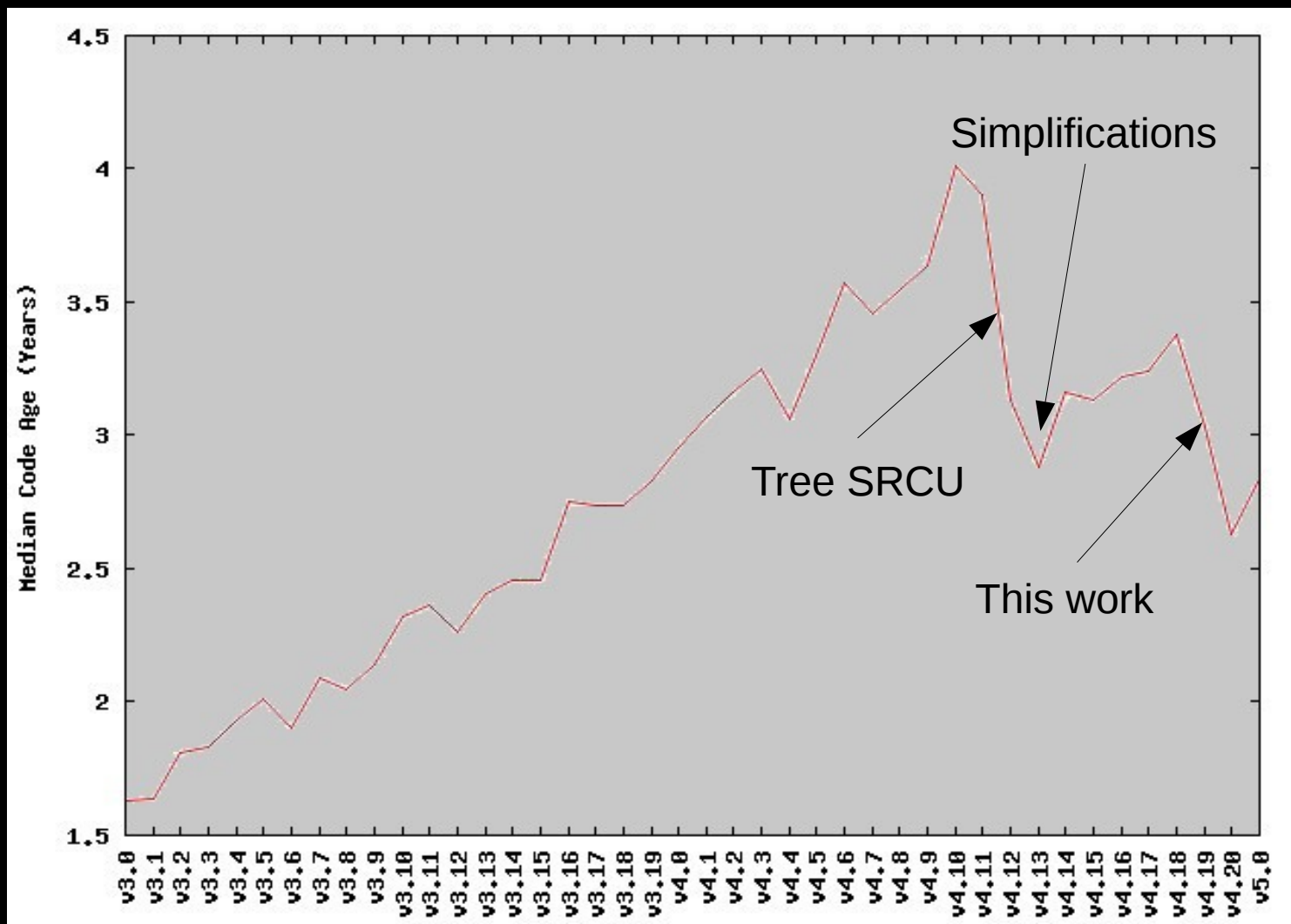
Median Age of RCU Code

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!!!

Other Consequences

- What effect did this work have on RCU's reliability?
- According to rcutorture, it is 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...
- And it is sometimes also interesting to look at fixes

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)

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)

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
- 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
- The usual influx of bugs that I don't expect at all...

Because Murphy Never Sleeps!!!

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()`

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()`
- 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?