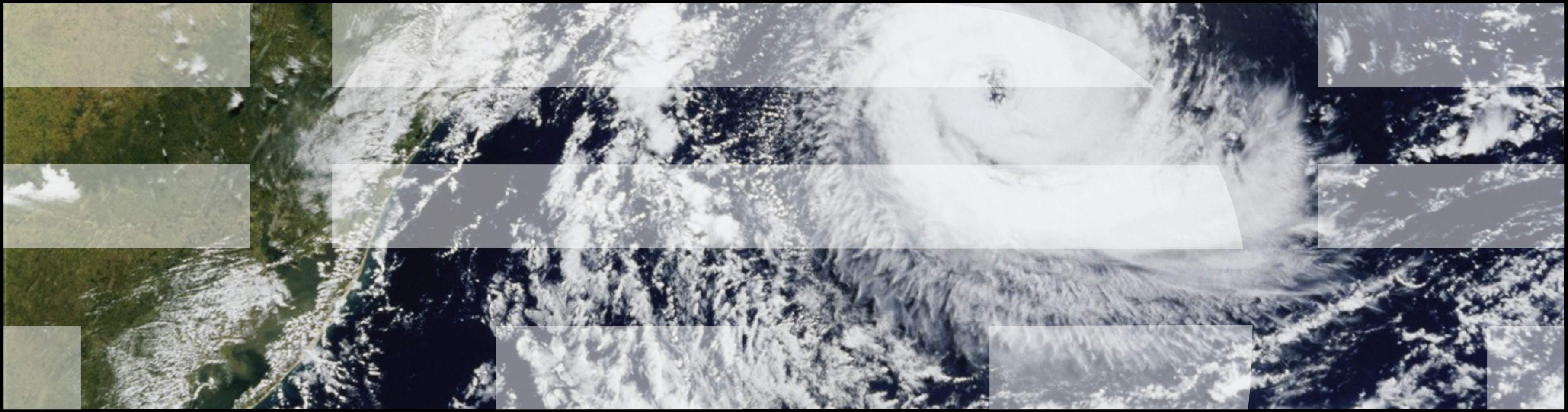
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center

Member, IBM Academy of Technology

linux.conf.au, January 25, 2018

Can RCU and CPU Hotplug Survive the Attack of the Killer Virtual Environments?

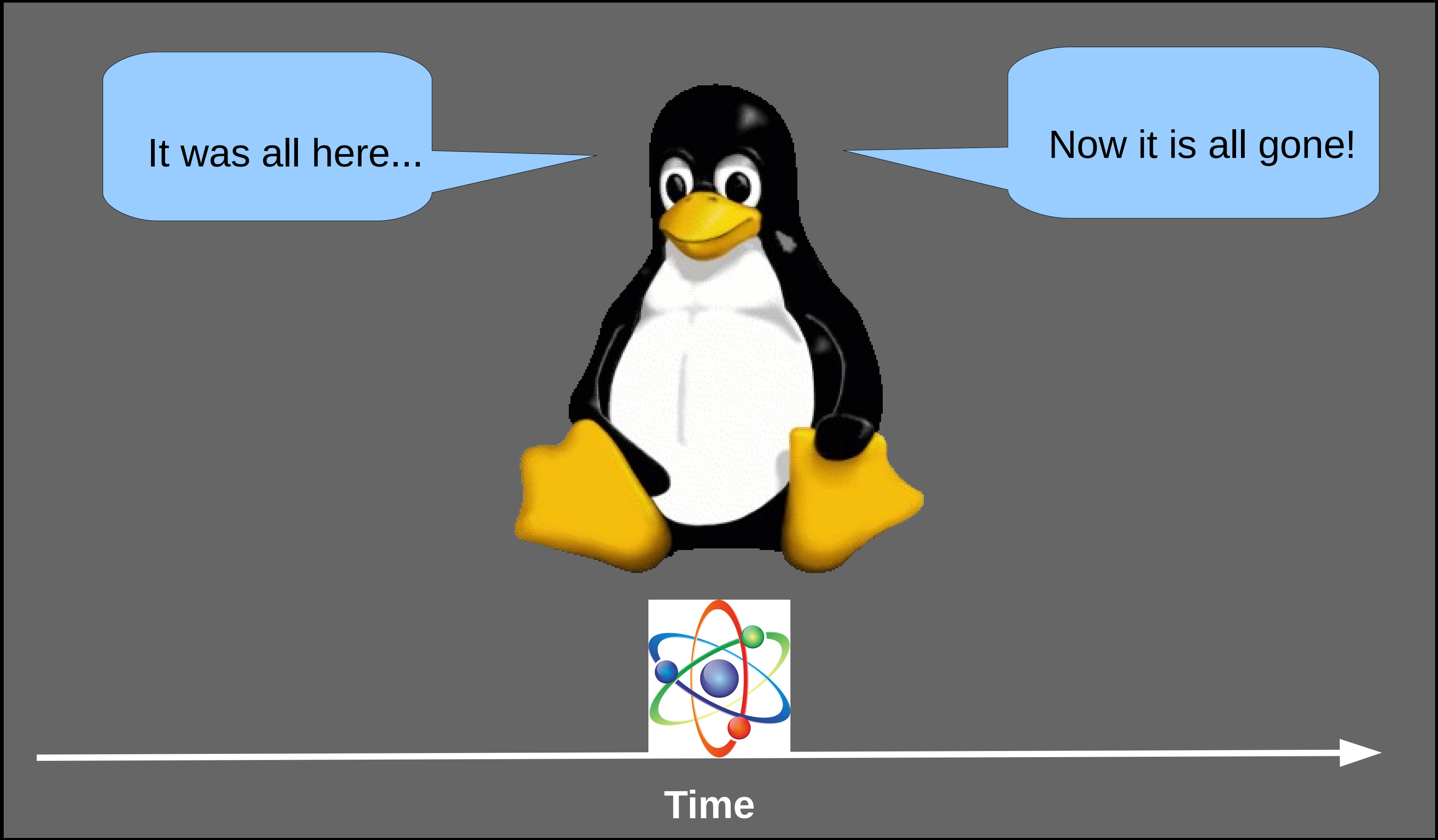


Overview

- Why would CPU hotplug be a problem?
- What is the big deal with RCU and CPU hotplug?
- Why would virtualization be a problem?
- More fun with RCU and virtualization
- Can RCU and CPU hotplug survive the attack of the killer virtual environments?

Why Would CPU Hotplug be a Problem?

CPU Hotplug Would Not be a Problem...
If it Could be Atomic!

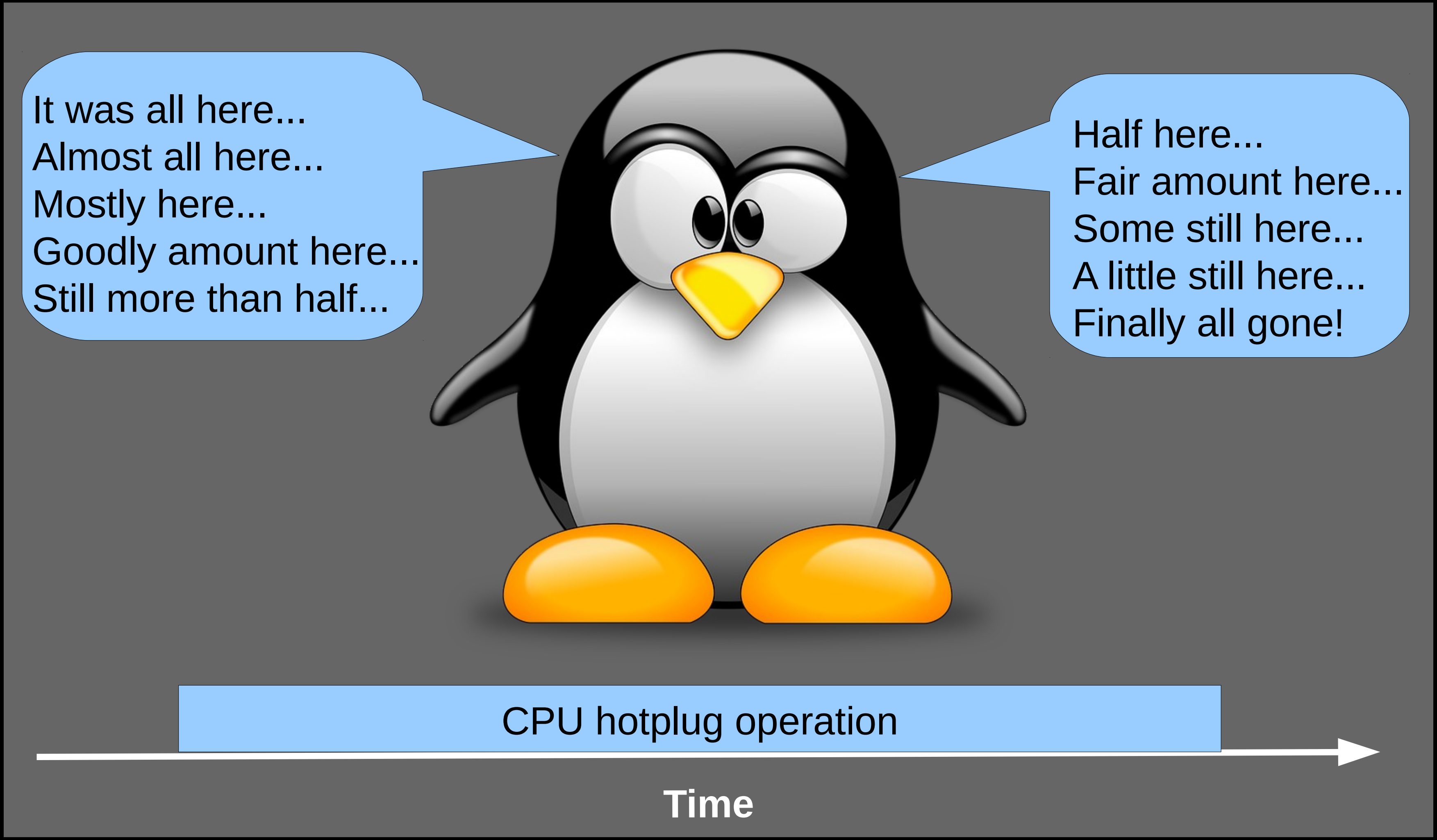


It was all here...

Now it is all gone!

Time

CPU Hotplug is Definitely *Not* Atomic!



It was all here...
Almost all here...
Mostly here...
Goodly amount here...
Still more than half...

Half here...
Fair amount here...
Some still here...
A little still here...
Finally all gone!

CPU hotplug operation

Time

CPU Hotplug is Definitely *Not* Atomic! Many Steps...

- Boot CPU:
  - offline
  - threads:prepare
  - perf:prepare
  - workqueue:prepare
  - hrtimers:prepare
  - smpcfd:prepare (call function)
  - relay:prepare
  - slab:prepare
  - RCU/tree:prepare
  - timers:dead
  - cpu:bringup
  - smpcfd:dying
  - cpu:teardown

- Application CPU
  - sched:starting
  - RCU/tree:dying
  - ap:online
  - smpboot/threads:online
  - irq/affinity:online
  - perf:online
  - workqueue:online
  - RCU/tree:online
  - sched:active
  - online

kernel/cpu.c

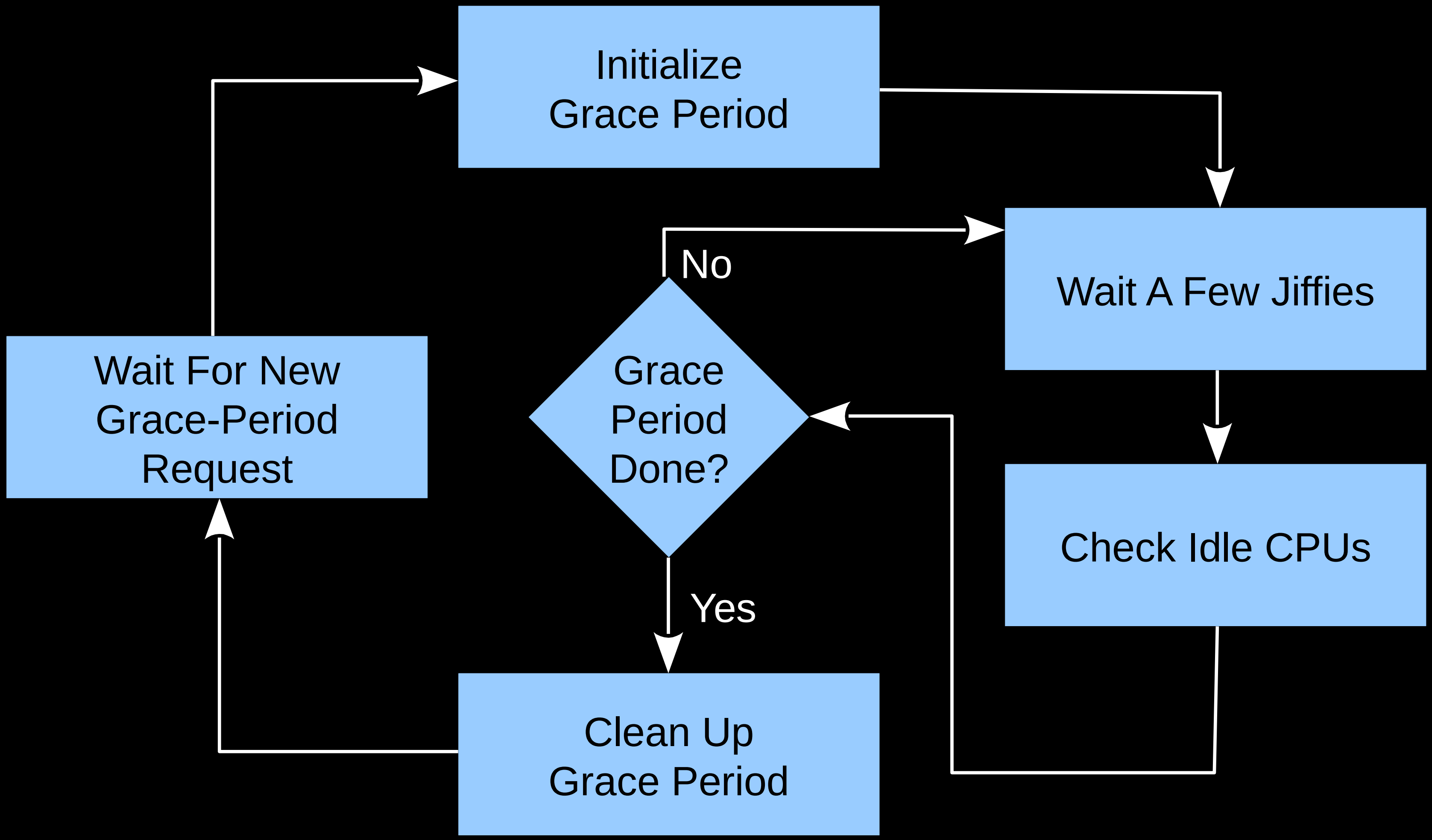
CPU Hotplug is Definitely *Not* Atomic! Many Steps... Towards CPUs as Sets of Services That Come and Go

- Boot CPU:
  - offline
  - threads:prepare
  - perf:prepare
  - workqueue:prepare
  - hrtimers:prepare
  - smpcfd:prepare (call function)
  - relay:prepare
  - slab:prepare
  - RCU/tree:prepare
  - timers:dead
  - cpu:bringup
  - smpcfd:dying
  - cpu:teardown
- Application CPU
  - sched:starting
  - RCU/tree:dying
  - ap:online
  - smpboot/threads:online
  - irq/affinity:online
  - perf:online
  - workqueue:online
  - RCU/tree:online
  - sched:active
  - online

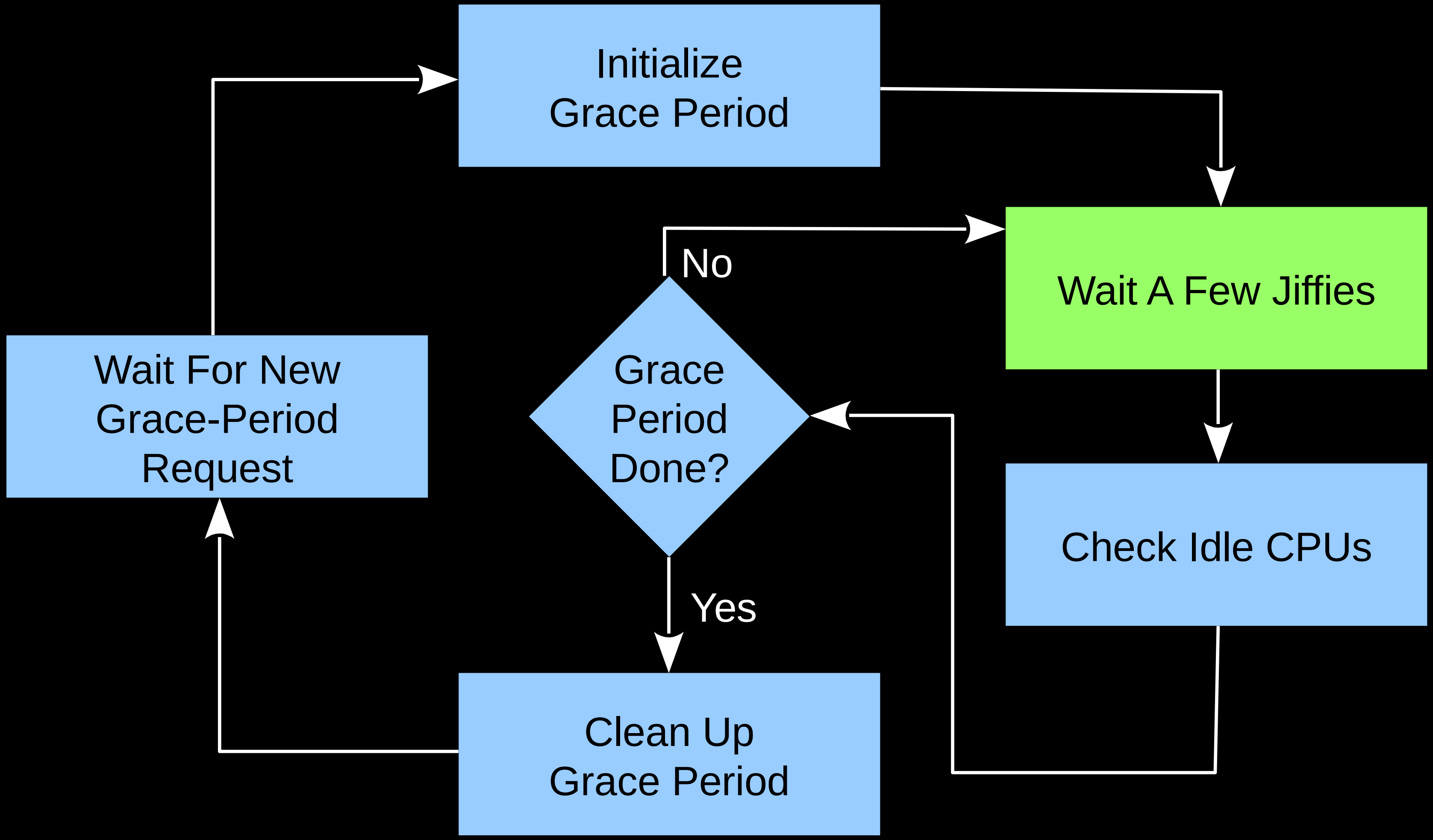
kernel/cpu.c

What is the Big Deal with RCU and CPU Hotplug?

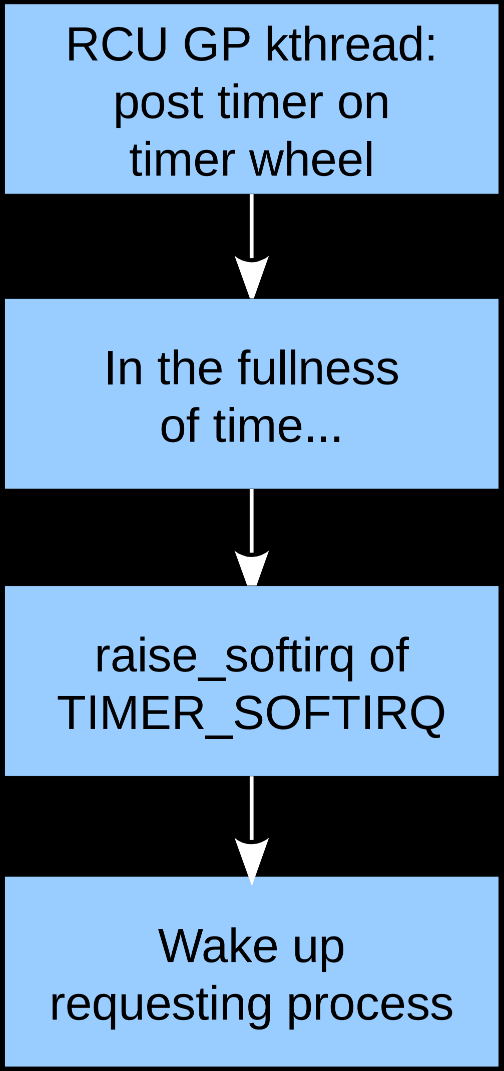
High-Level RCU Grace-Period Processing



High-Level RCU Grace-Period Processing



Wait a Few Jiffies: High-Level Timer Processing



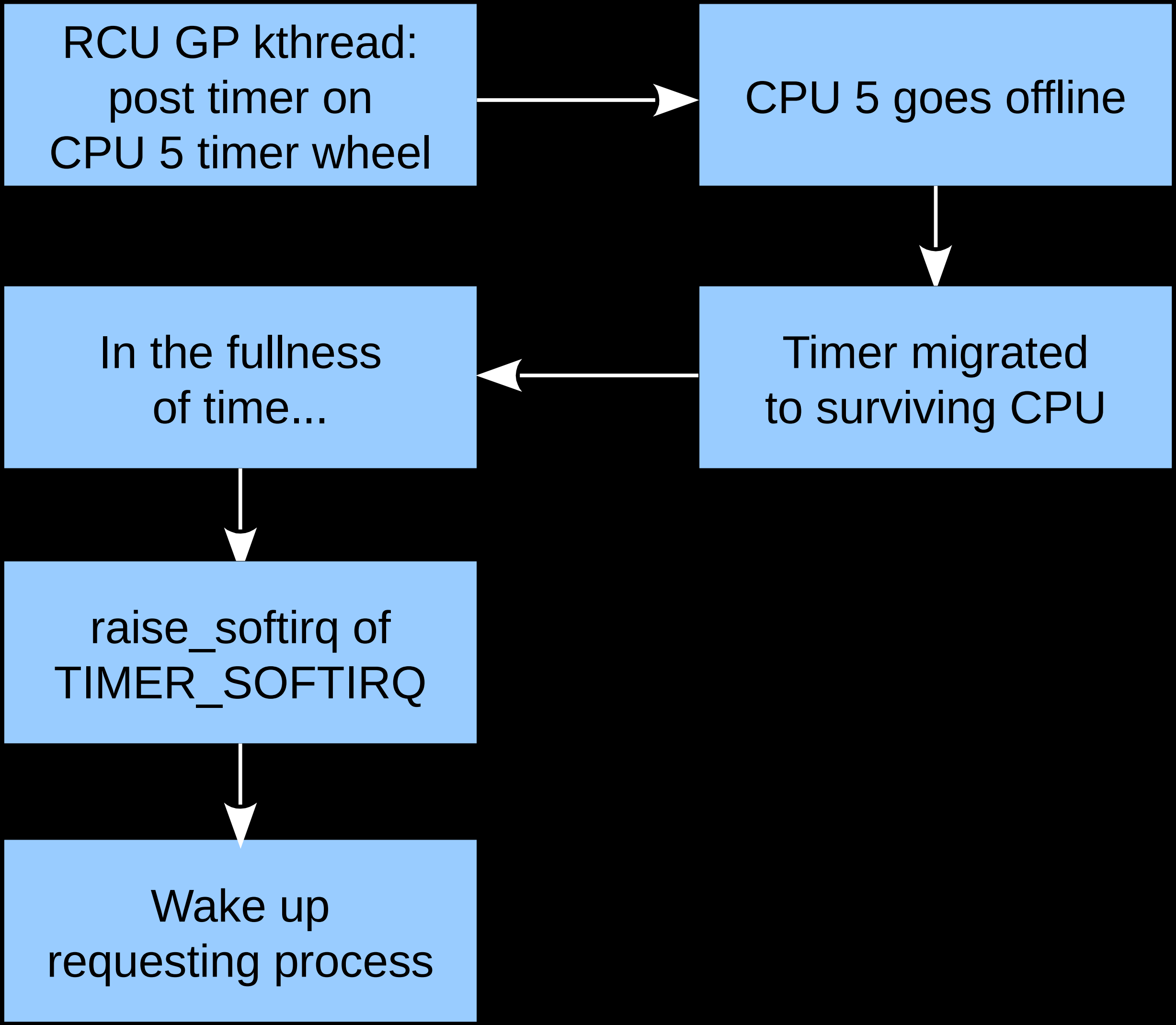
RCU GP kthread: post timer on timer wheel

In the fullness of time...

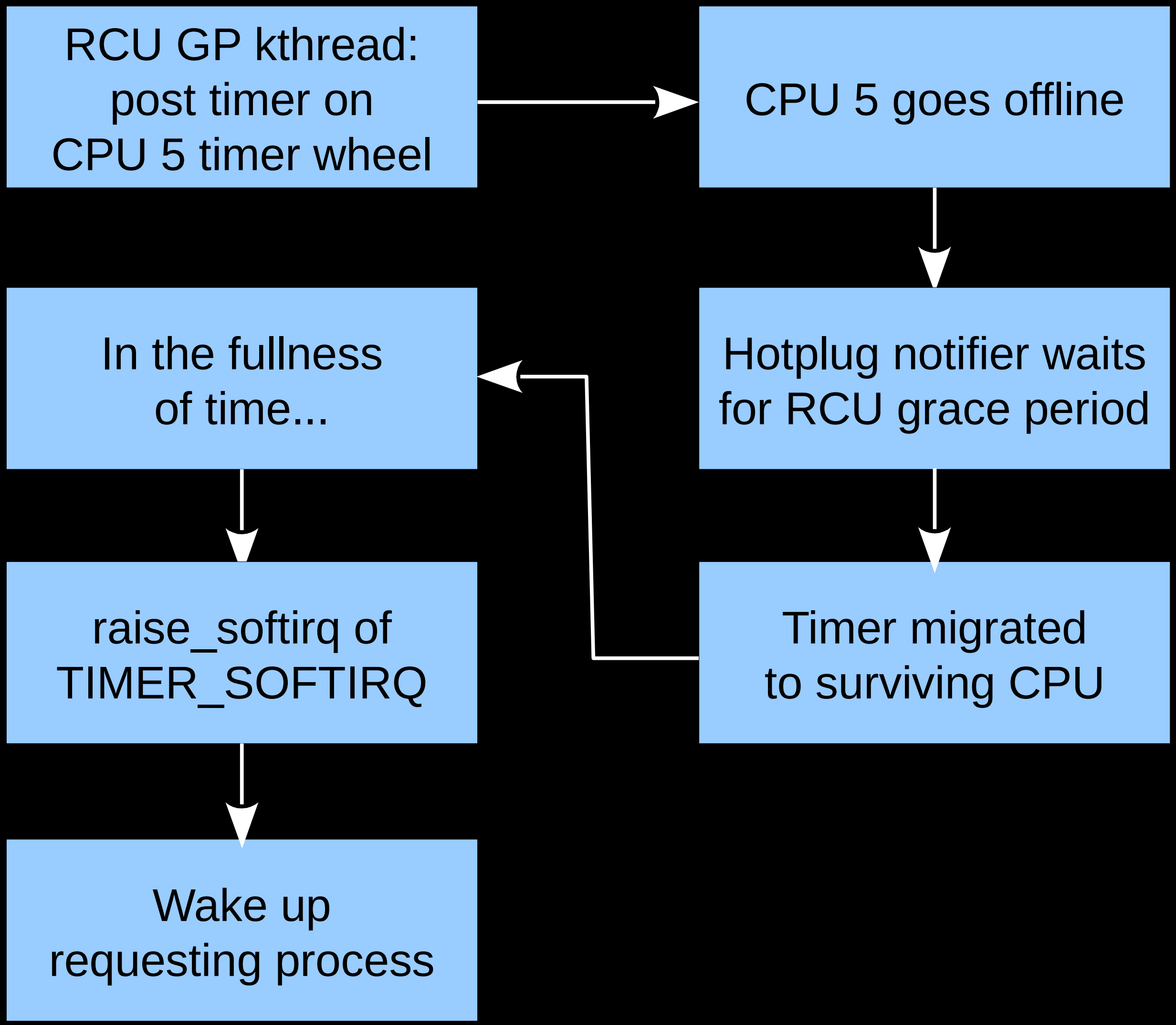
raise_softirq of TIMER_SOFTIRQ

Wake up requesting process

High-Level Timer Processing, CPU Offline


RCU GP kthread: post timer on CPU 5 timer wheel
CPU 5 goes offline
Timer migrated to surviving CPU
In the fullness of time...
raise_softirq of TIMER_SOFTIRQ
Wake up requesting process

High-Level Timer Processing, CPU Offline, RCU



RCU GP kthread: post timer on CPU 5 timer wheel

CPU 5 goes offline

Hotplug notifier waits for RCU grace period

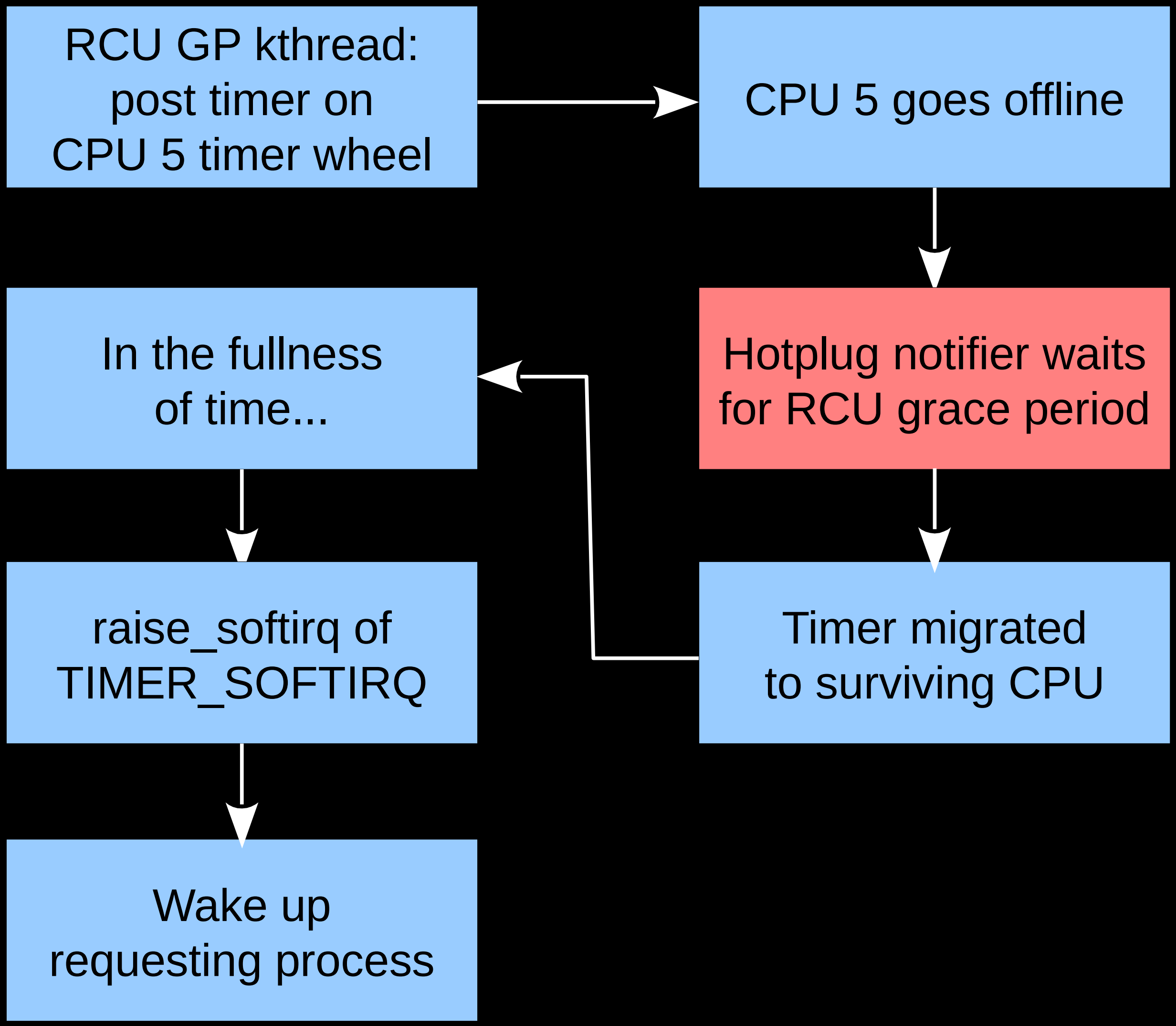
Timer migrated to surviving CPU

In the fullness of time...

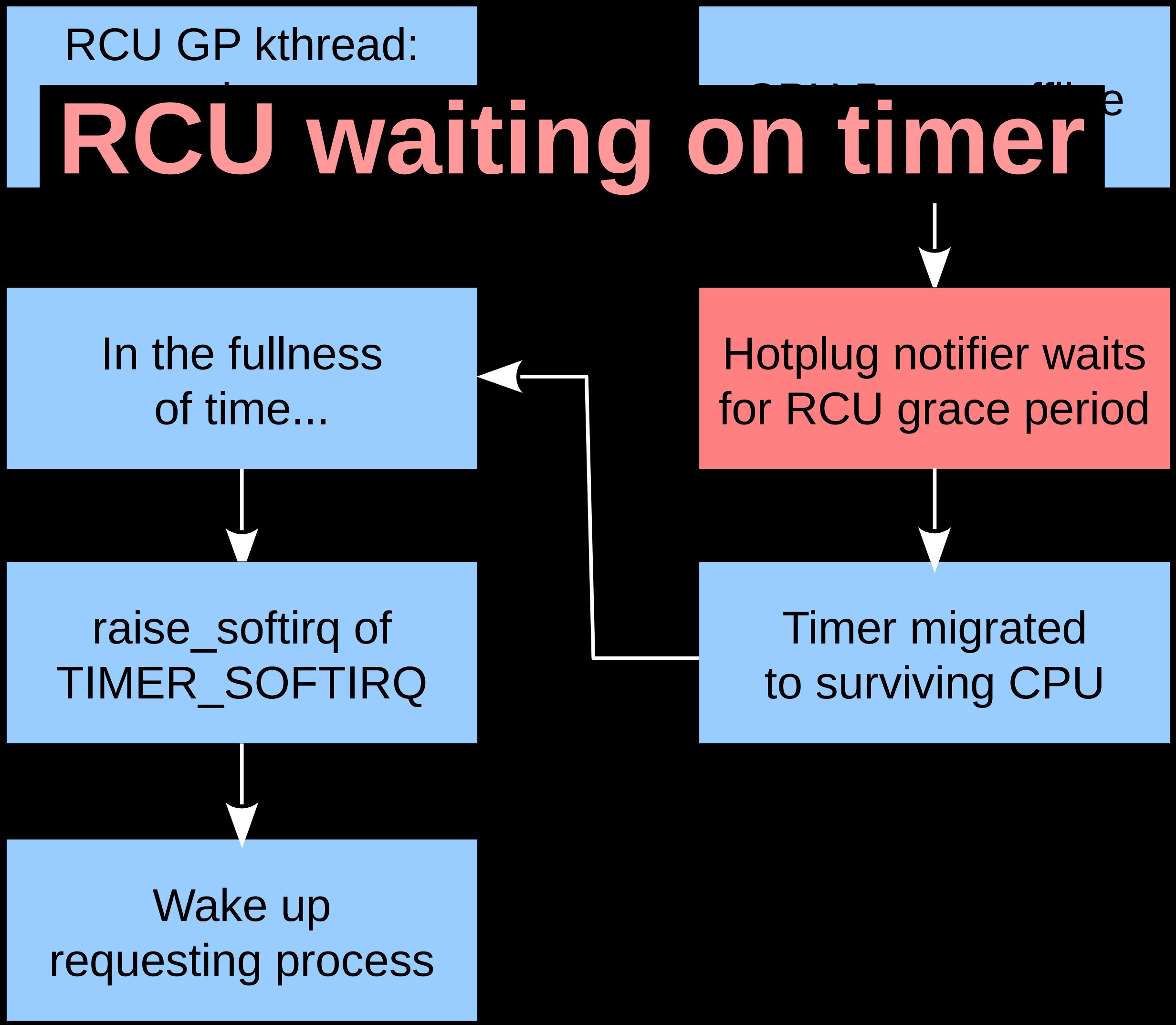
raise_softirq of TIMER_SOFTIRQ

Wake up requesting process

High-Level Timer Processing, CPU Offline, RCU


RCU GP kthread: post timer on CPU 5 timer wheel
CPU 5 goes offline
Hotplug notifier waits for RCU grace period
Timer migrated to surviving CPU
In the fullness of time...
raise_softirq of TIMER_SOFTIRQ
Wake up requesting process

High-Level Timer Processing, CPU Offline, RCU



RCU GP kthread:

RCU waiting on timer

In the fullness of time...

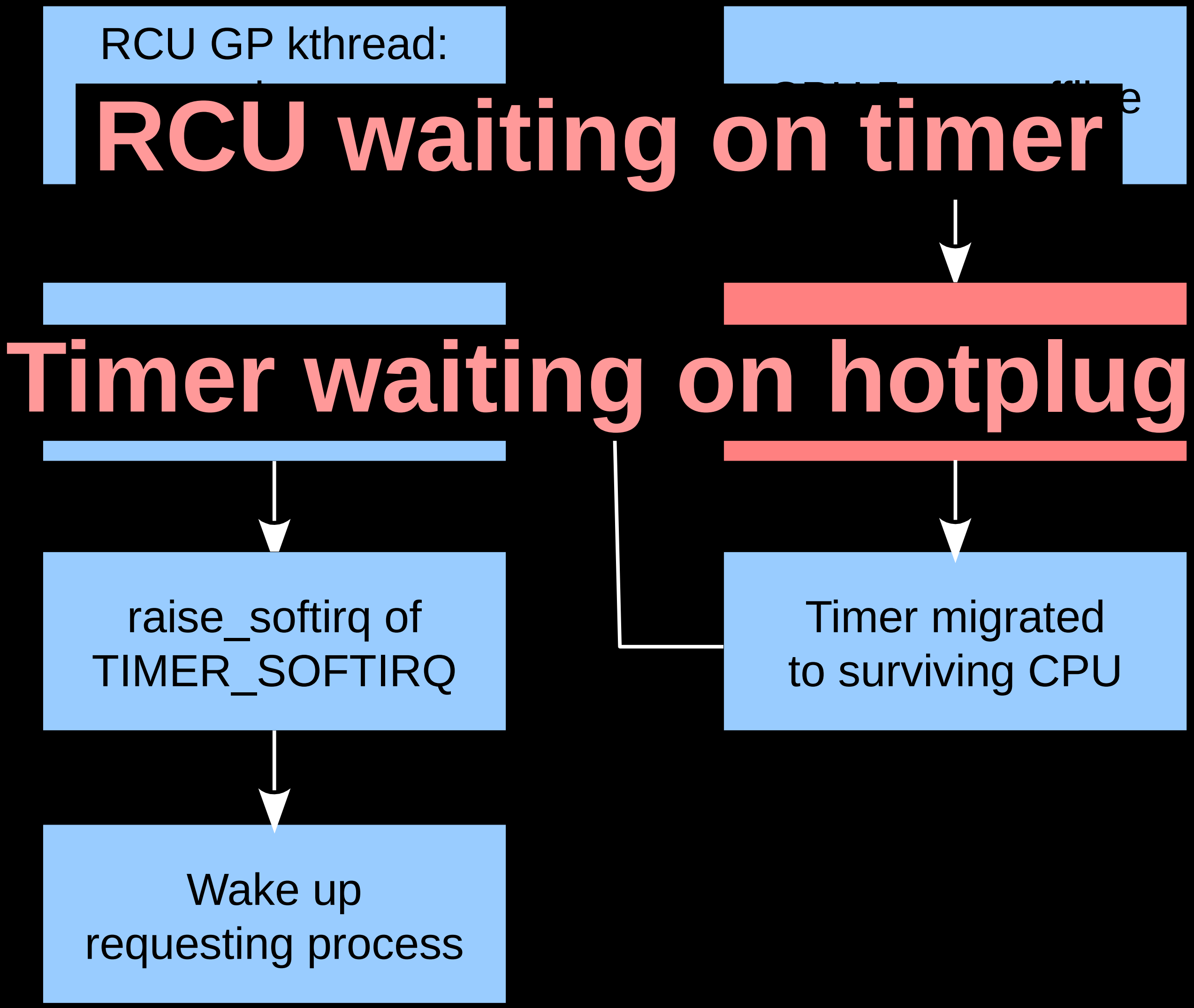
raise_softirq of TIMER_SOFTIRQ

Wake up requesting process

Hotplug notifier waits for RCU grace period

Timer migrated to surviving CPU

High-Level Timer Processing, CPU Offline, RCU



RCU GP kthread:

RCU waiting on timer

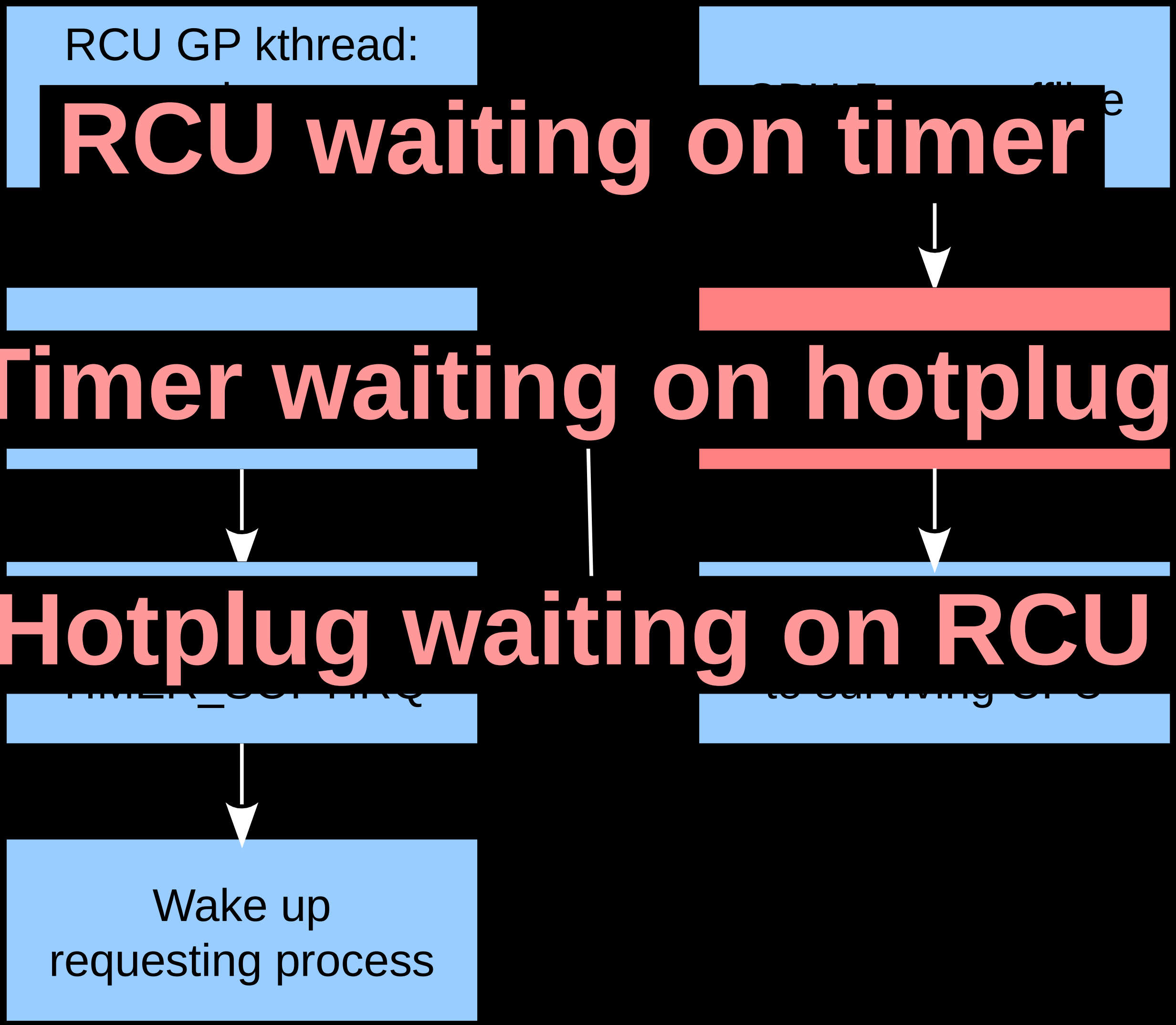
Timer waiting on hotplug

raise_softirq of TIMER_SOFTIRQ

Timer migrated to surviving CPU

Wake up requesting process

High-Level Timer Processing, CPU Offline, RCU

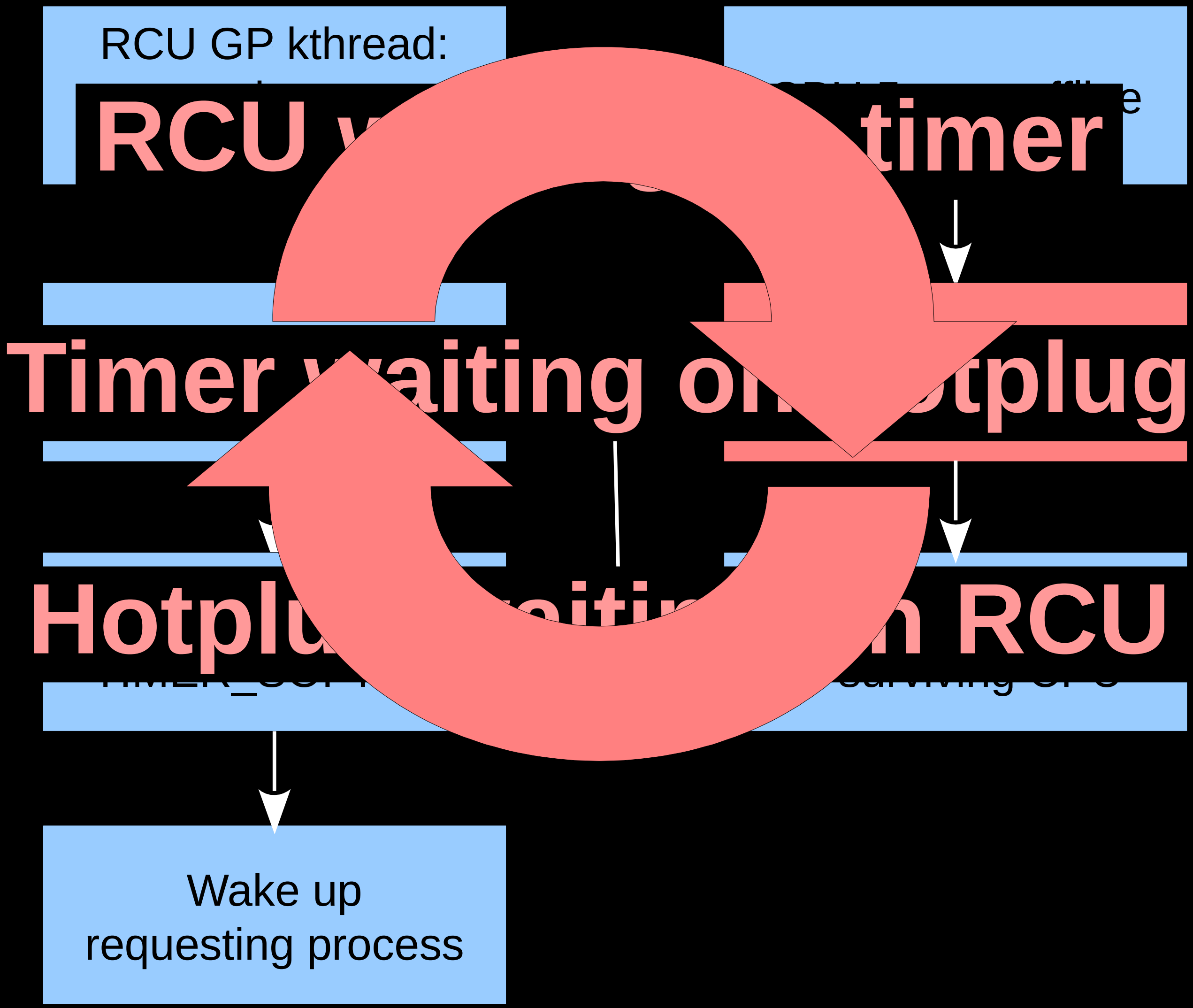


RCU waiting on timer

Timer waiting on hotplug

Hotplug waiting on RCU

High-Level Timer Processing, CPU Offline, RCU



Time Waits For No One, But It Can Deadlock With CPU-Hotplug Offline and RCU Grace Periods!!!

```
/*
 * On the tear-down path, timers_dead_cpu() must be invoked
 * before blk_mq_queue_reinit_notify() from notify_dead(),
 * otherwise a RCU stall occurs.
 */
```

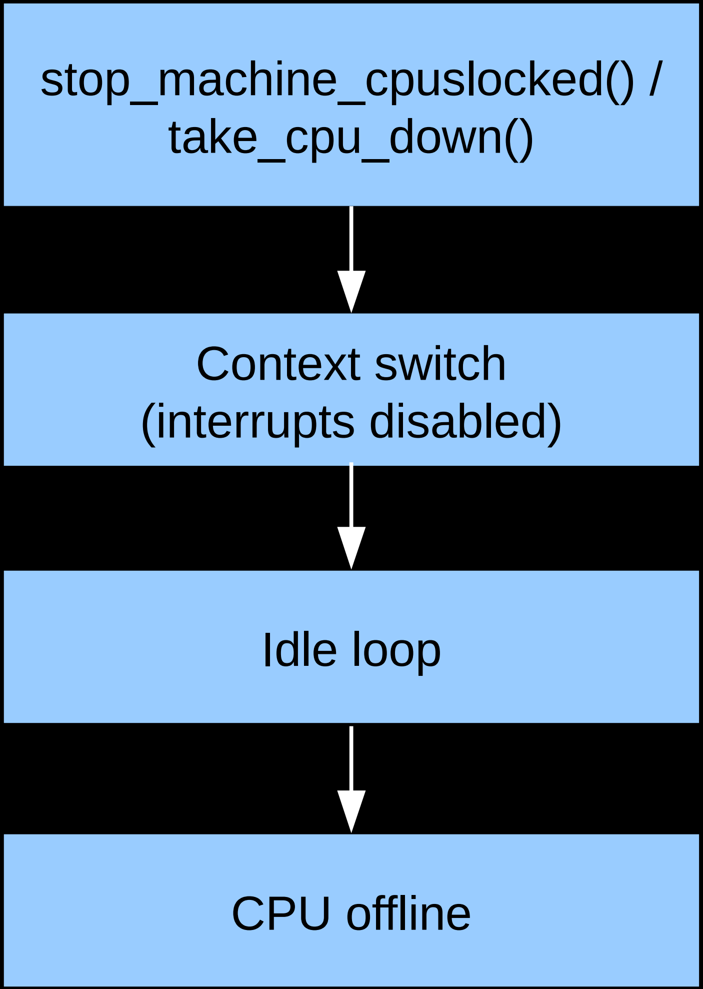
Time Waits For No One, But It Can Deadlock With CPU-Hotplug Offline and RCU Grace Periods!!!

```
/*
 * On the tear-down path, timers_dead_cpu() must be invoked
 * before blk_mq_queue_reinit_notify() from notify_dead(),
 * otherwise a RCU stall occurs.
 */
```

In addition, RCU migrates callbacks from outgoing CPUs earlier in the process

Why Would Virtualization be a Problem?

Why Would Virtualization be a Problem?
Last Gasps of An Outgoing CPU



stop_machine_cpuslocked() / take_cpu_down()

Clears CPU's cpu_online_mask bit

Context switch (interrupts disabled)

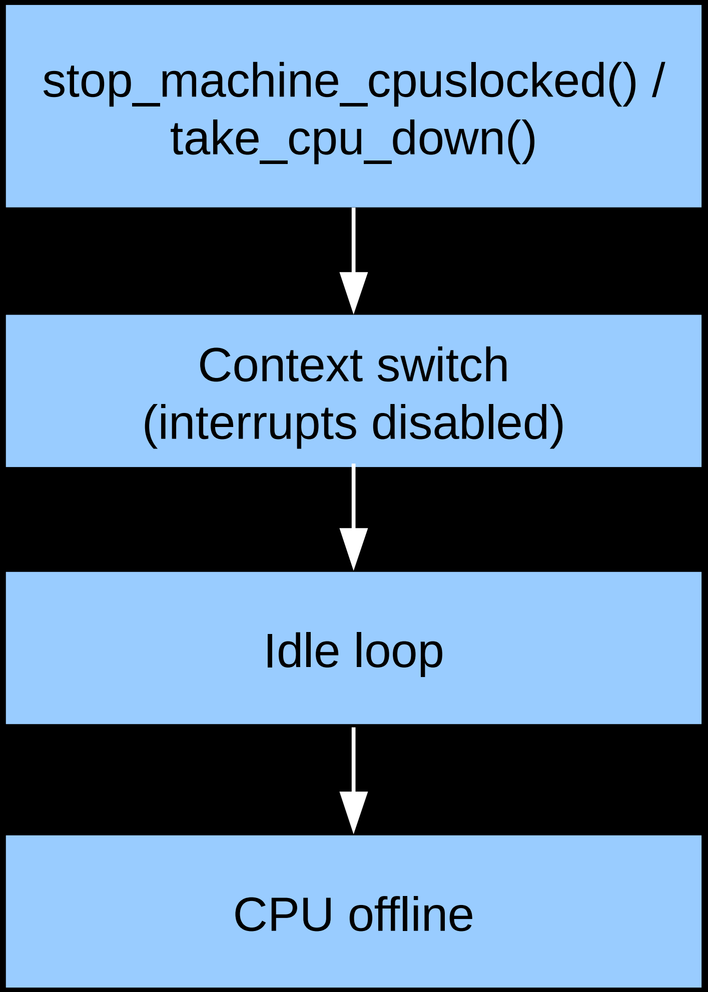
Final pass through the scheduler

Idle loop

Special check transitions CPU out

CPU offline

Why Would Virtualization be a Problem?
Last Gasps of An Outgoing CPU



stop_machine_cpuslocked() /
take_cpu_down()

Clears CPU's cpu_online_mask bit

Context switch
(interrupts disabled)

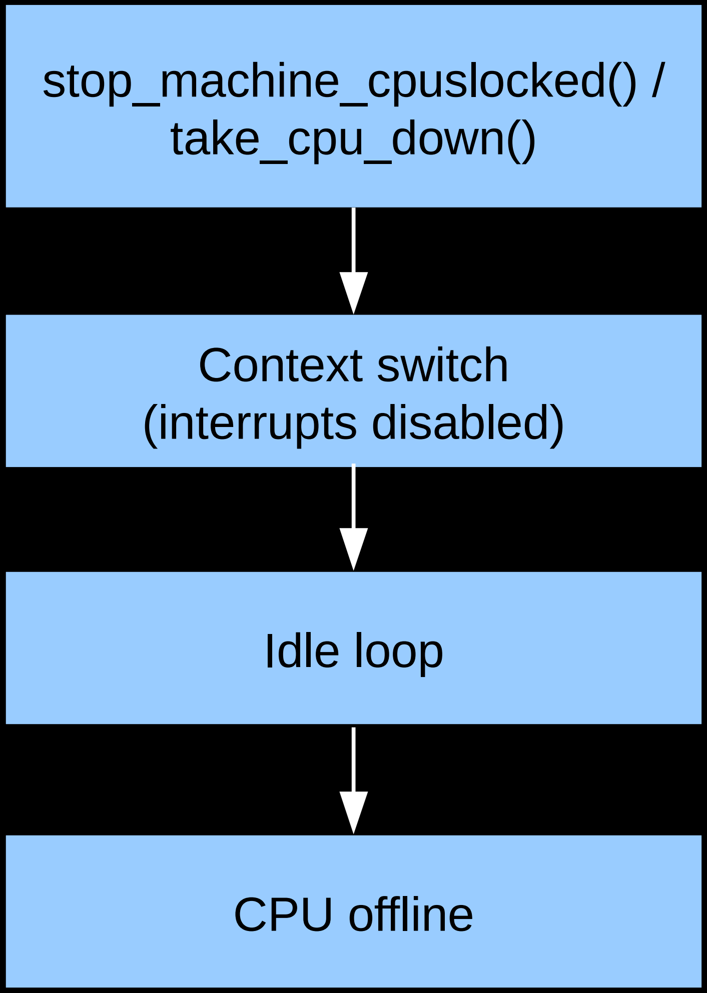
The scheduler uses RCU!!!
So RCU must watch this CPU!!!

Idle loop

Special check transitions CPU out

CPU offline

Why Would Virtualization be a Problem?
Last Gasps of An Outgoing CPU



stop_machine_cpuslocked() / take_cpu_down()

Context switch (interrupts disabled)

Idle loop

CPU offline

Clears CPU's cpu_online_mask bit

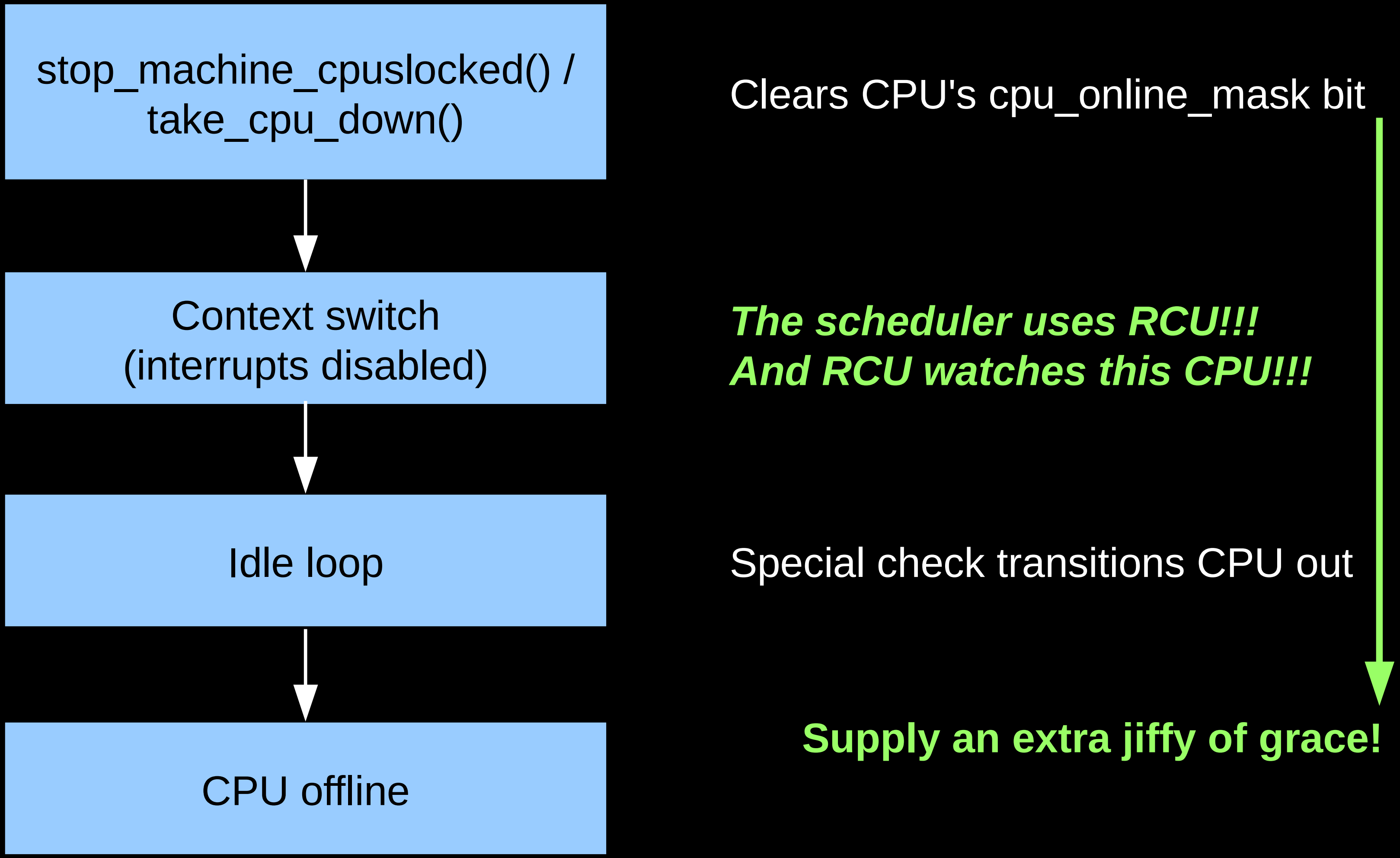
The scheduler uses RCU!!!
So RCU must watch this CPU!!!

Special check transitions CPU out

But:

- Interrupts are disabled
- Nothing runnable on this CPU
- Only a few microseconds!!!

Why Would Virtualization be a Problem?
Last Gasps of An Outgoing CPU: Happy Hack!!!



stop_machine_cpuslocked() / take_cpu_down()

Clears CPU's cpu_online_mask bit

Context switch (interrupts disabled)

The scheduler uses RCU!!!
And RCU watches this CPU!!!

Idle loop

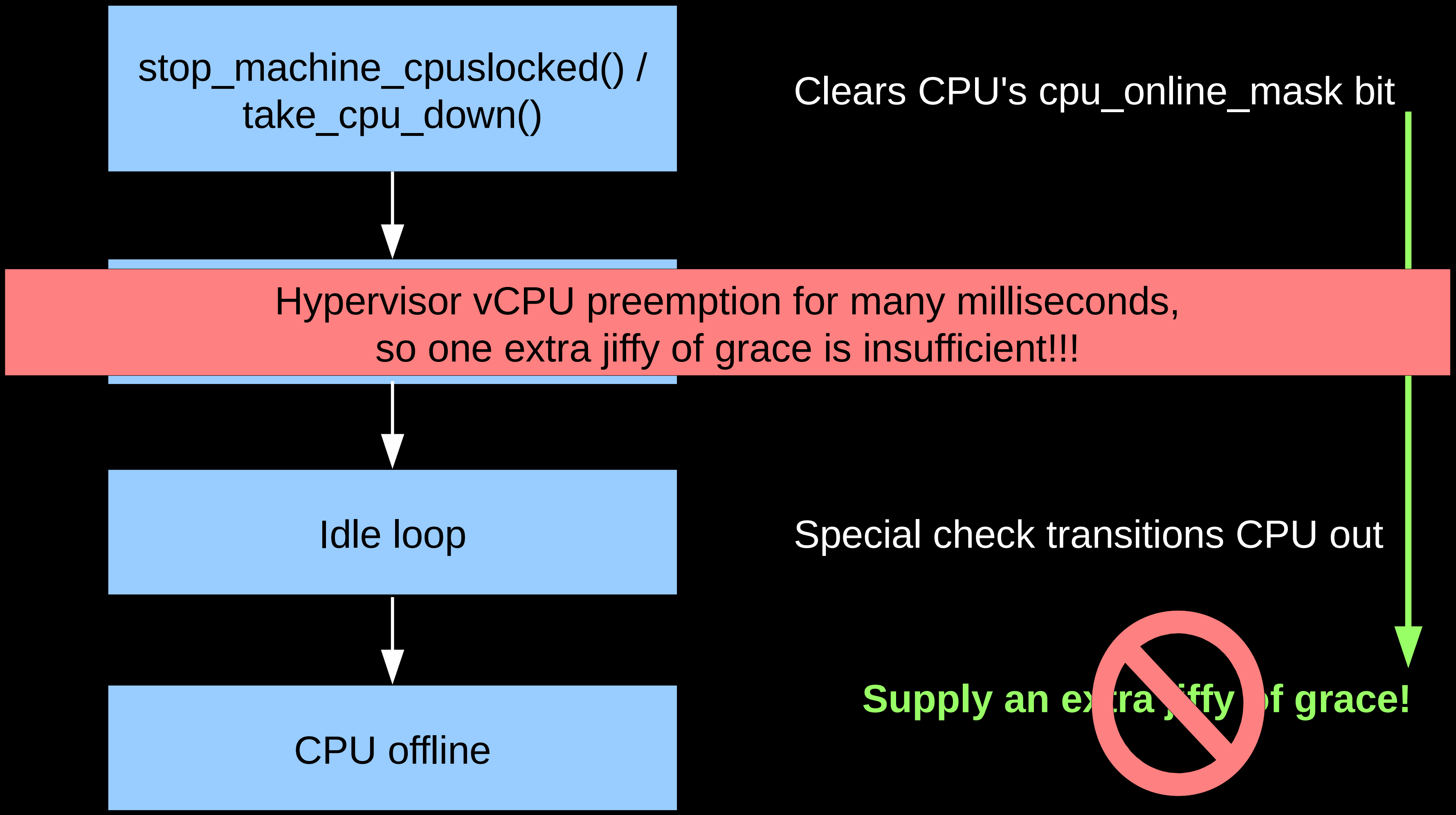
Special check transitions CPU out

Supply an extra jiffy of grace!

CPU offline

Hack Not So Happy On Hypervisors...

Why Would Virtualization be a Problem? Last Gasps of An Outgoing CPU With Hypervisor...



stop_machine_cpuslocked() / take_cpu_down()

Clears CPU's cpu_online_mask bit

Hypervisor vCPU preemption for many milliseconds, so one extra jiffy of grace is insufficient!!!

Idle loop

Special check transitions CPU out

CPU offline

Supply an extra jiffy of grace!

The Horrible Thing?

The Horrible Thing?
No Reported Failures in More Than 10 Years

Is This A Real Problem?

- This has not been a problem in the past, but:
  - Cloud providers are increasing utilizations
  - Higher utilization results in increased probability of preemption
- vCPU preemption really does happen!!!
- Cloud-computing economics seems likely to encourage heavy levels of overcommitment
  - A solution would therefore be a good thing

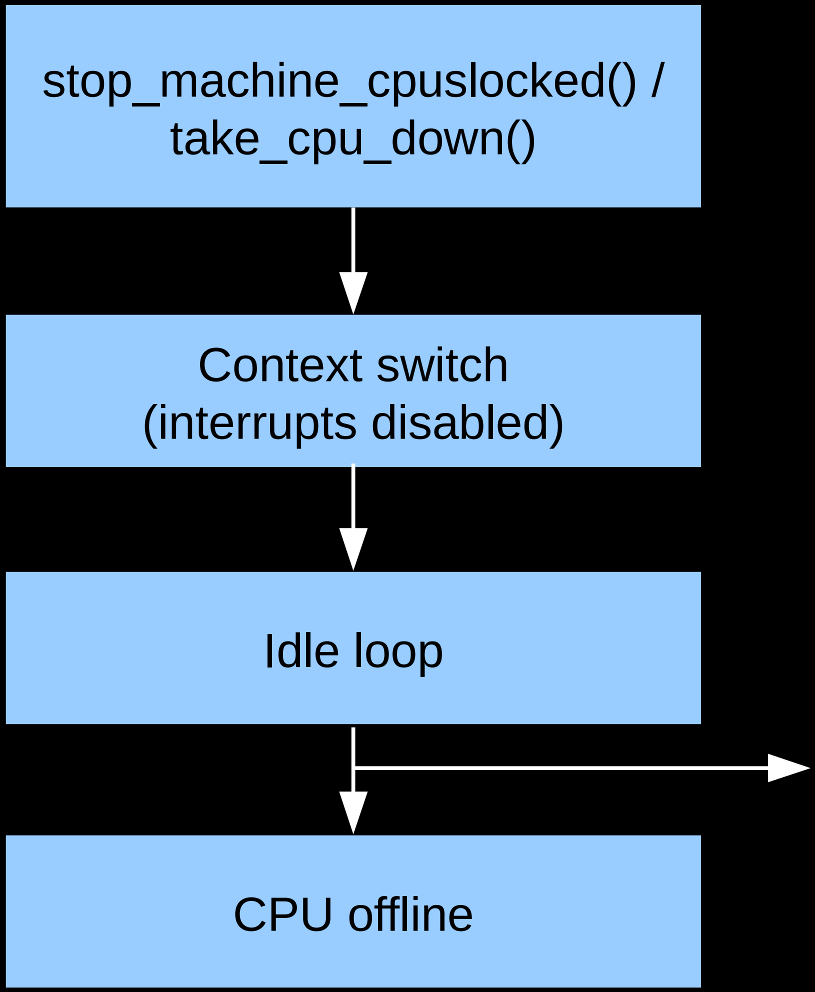
Non-Solutions

- Increase the number of jiffies of grace
  - Someone might do “kill -STOP” on a particular vCPU
  - Or perhaps someday even single-step it...
- Delay grace period until end of CPU hotplug operation
  - Some CPU-hotplug notifiers wait for grace periods
  - Deadlock!!!
- Detect the problem after the fact and fix it
  - ***Very*** hard to fix damage caused by too-short grace period
  - Such damage is also known as “random memory corruption”

Solution: RCU Ignores cpu_online_mask

Solution: RCU Ignores cpu_online_mask
Solve The Problem By Keeping Two Sets of Books

Solution: RCU Ignores cpu_online_mask



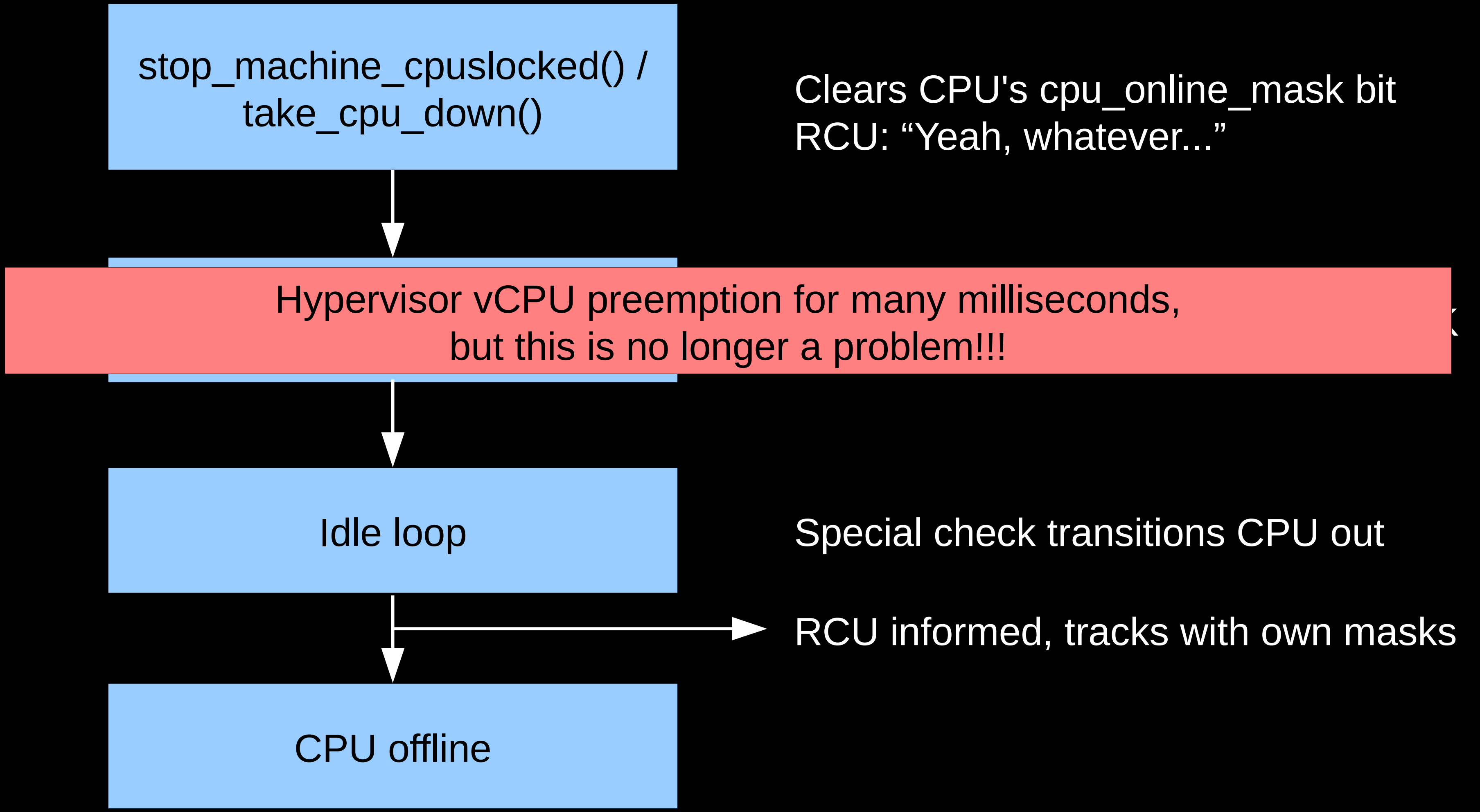
Clears CPU's cpu_online_mask bit
RCU: “Yeah, whatever...”

The scheduler uses RCU, but now OK

Special check transitions CPU out

RCU informed, tracks with own masks

Solution: RCU Ignores cpu_online_mask



stop_machine_cpuslocked() / take_cpu_down()

Clears CPU's cpu_online_mask bit
RCU: “Yeah, whatever...”

Hypervisor vCPU preemption for many milliseconds,
but this is no longer a problem!!!

Idle loop

Special check transitions CPU out

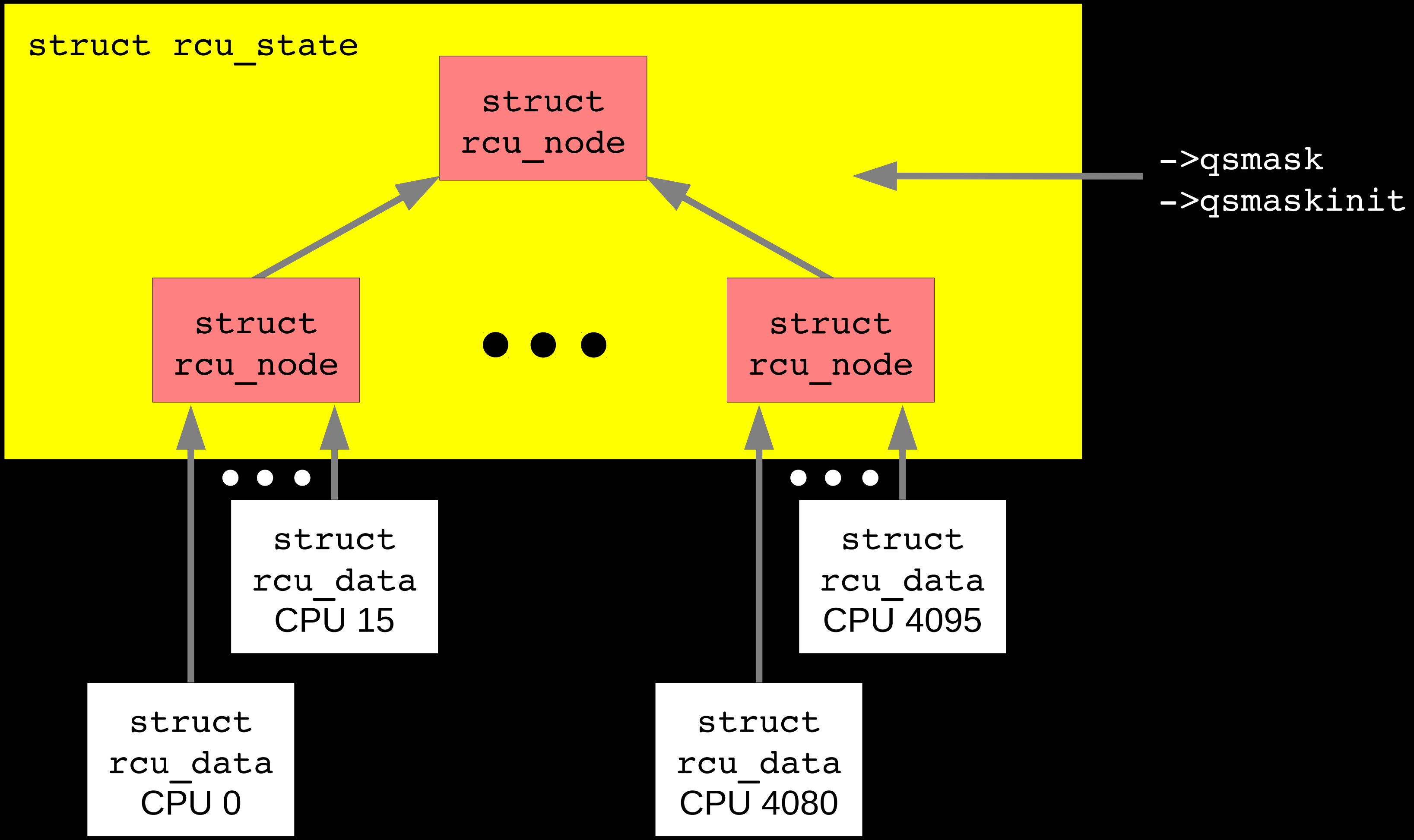
RCU informed, tracks with own masks

CPU offline

Ignore cpu_online_mask: Issues and Tricks

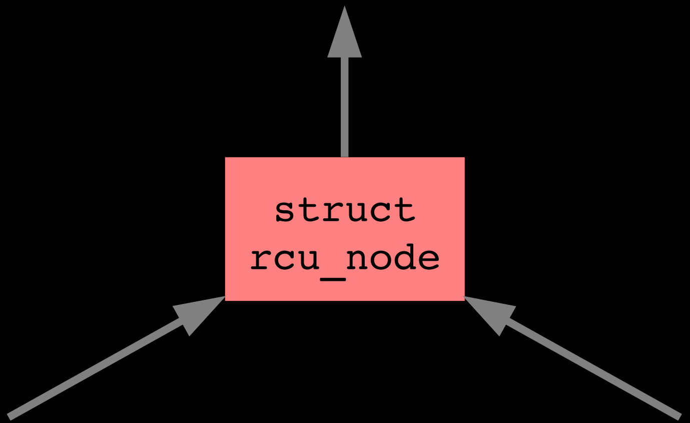
- Issue: RCU needs consistent snapshot of online CPUs when initializing a grace period
  - And blocking CPU hotplug during this time is no longer acceptable
  - RCU must permit waiting on grace periods during hotplug operations
- Trick: RCU only needs to pay attention to CPUs that were online when the grace period started
  - CPUs coming online mid-grace-period may be ignored
- Trick: RCU separately checks for CPUs going offline
  - CPUs going offline mid-grace-period needn't interact with grace period

RCU Major Data Structures Hold Bit Masks



Each node covers CPUs in its subtree
Initialization proceeds breadth-first from root node

Bit Masks Back In The Day...



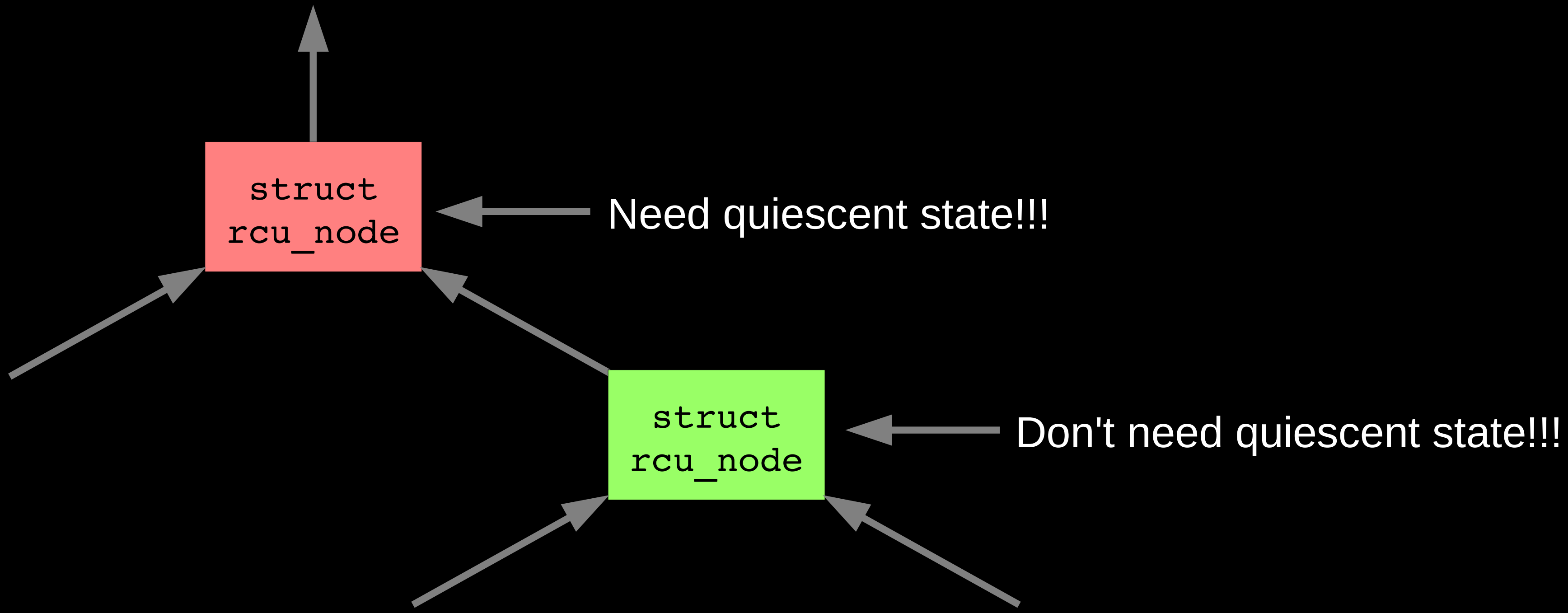
`->qsmask`

CPUs below needing to pass through a quiescent state? Initialized from ->qsmaskinit at start of each grace period, cleared by CPUs after quiescent state

`->qsmaskinit`

Value of ->qsmask for next grace period, set and cleared by CPU hotplug

Problem With Bit Masks Back In The Day...
(Avoided by Blocking Hotplug During GP Init)



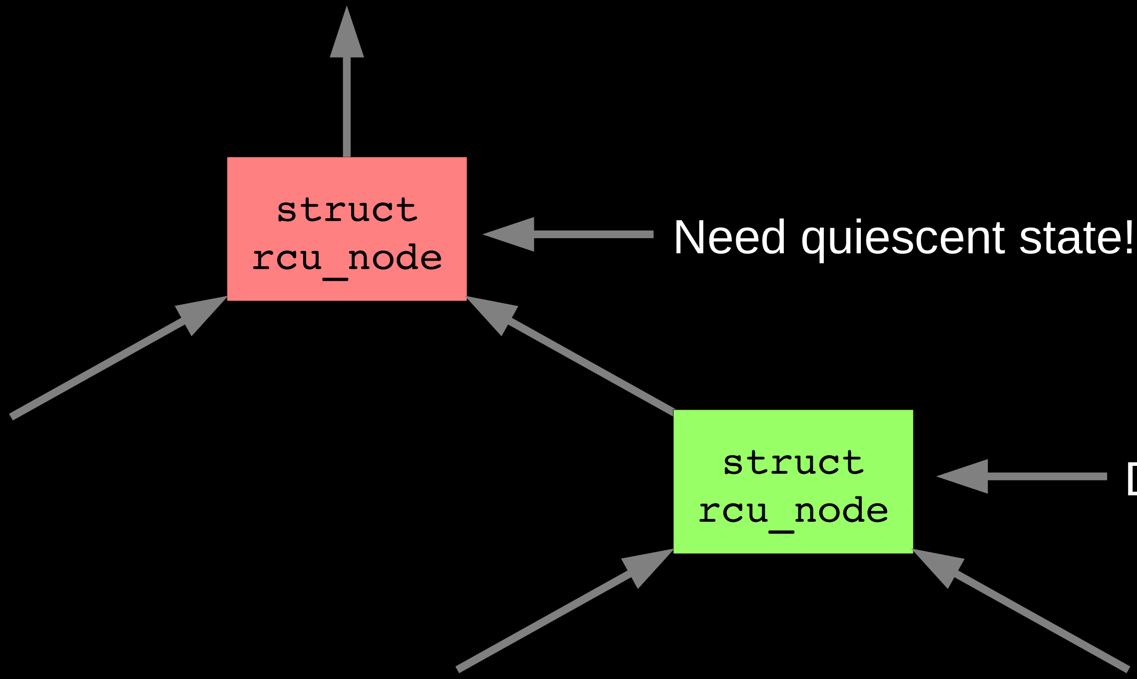
struct rcu_node

Need quiescent state!!!

struct rcu_node

Don't need quiescent state!!!

Problem With Bit Masks Back In The Day...
(Avoided by Blocking Hotplug During GP Init)

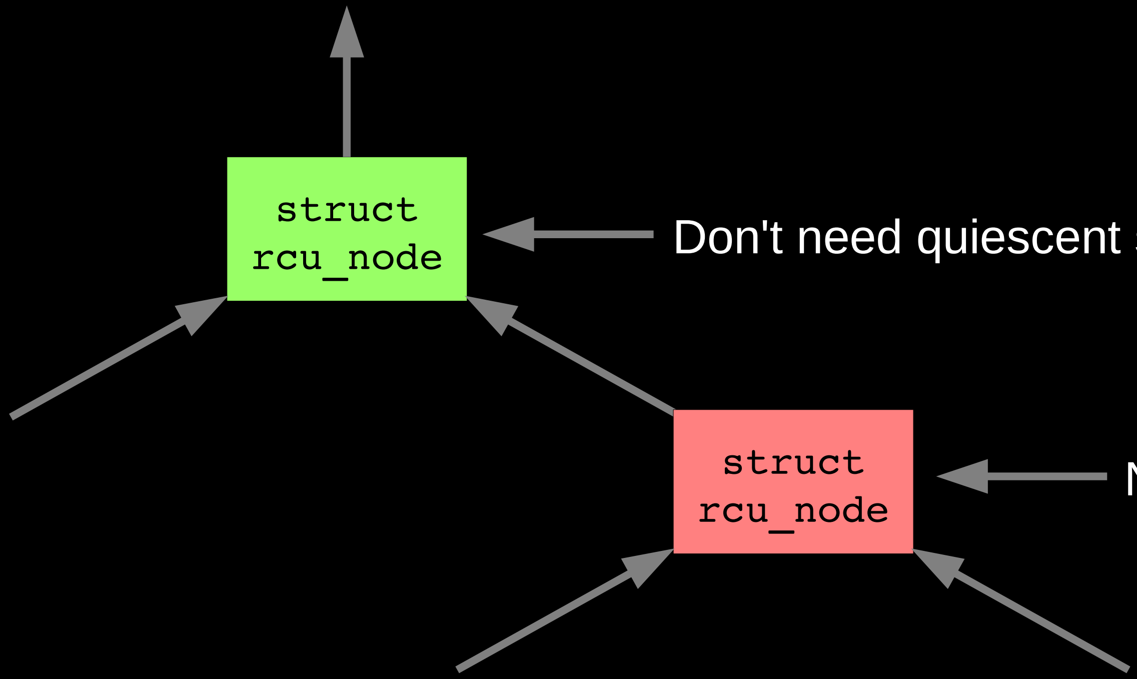


Need quiescent state!!!

Don't need quiescent state!!!

Grace-period hang!!!

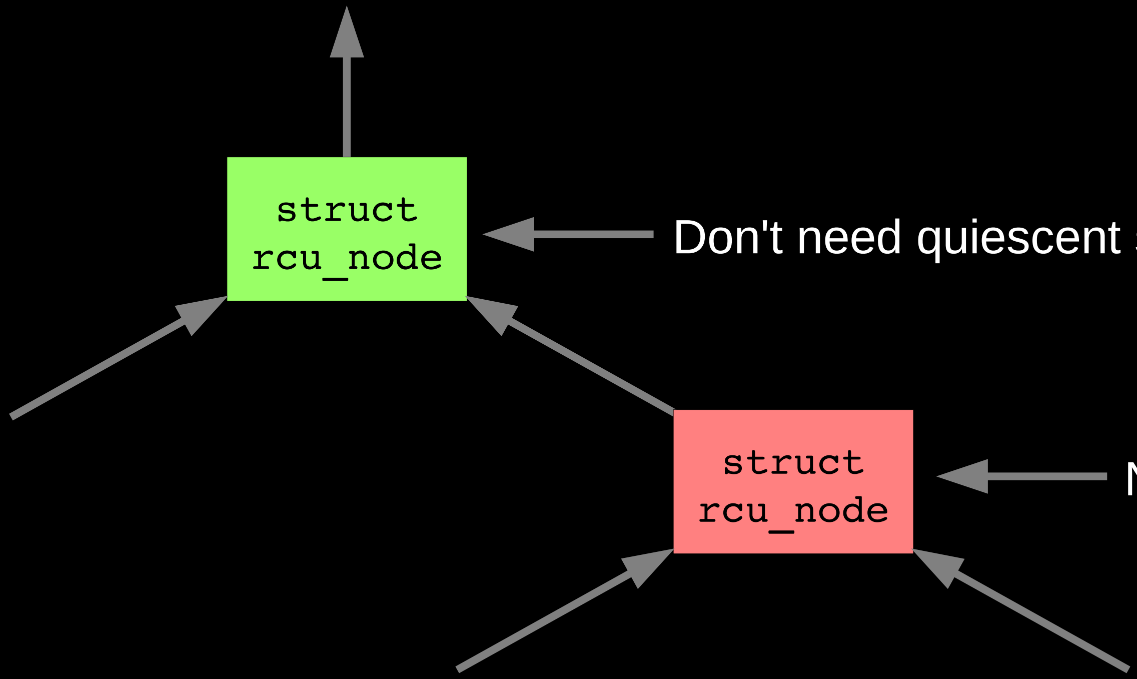
Another Problem With Bit Masks Back In The Day... (Avoided by Blocking Hotplug During GP Init)



Don't need quiescent state!!!

Need quiescent state!!!

Another Problem With Bit Masks Back In The Day... (Avoided by Blocking Hotplug During GP Init)

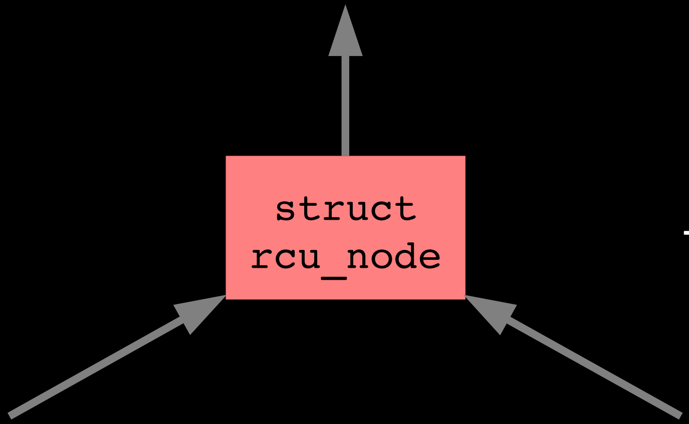


Don't need quiescent state!!!

Need quiescent state!!!

Too-short grace-period!!!
Can result in arbitrary memory corruption...

Solution: Add Another Bit Mask to Keep the Second Set of Books!!!



`->qsmask`

CPUs below needing to pass through a quiescent state? Initialized at start of each grace period, cleared by CPUs

`->qsmaskinit`

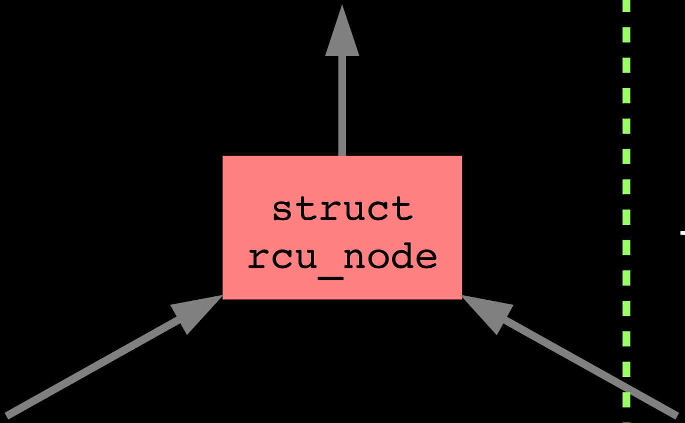
Value of ->qsmask for next grace period, copied from ->qsmaskinitnext at start of each grace period while holding ->lock

`->qsmaskinitnext`

Value of ->qsmaskinit for next grace period, set and cleared by CPU hotplug while holding ->lock

Second set of books

Solution: Add Another Bit Mask to Keep the Second Set of Books!!!



Now guaranteed consistent!!!

`->qsmask`

CPUs below needing to pass through a quiescent state? Initialized at start of each grace period, cleared by CPUs

`->qsmaskinit`

Value of ->qsmask for next grace period, copied from ->qsmaskinitnext at start of each grace period while holding ->lock

`->qsmaskinitnext`

Value of ->qsmaskinit for next grace period, set and cleared by CPU hotplug while holding ->lock

Additional Benefits of Ignoring cpu_online_mask

- RCU need not block CPU hotplug during grace-period setup
- RCU expedited grace periods avoid blocking CPU hotplug
- Now OK to wait for grace periods in CPU-hotplug notifiers
  - But please keep CPU-hotplug latency down to a dull roar...
- The rcu_barrier() primitive, alas, still blocks CPU hotplug
  - Fixing this is on my list...

More Fun with RCU and Virtualization

More Fun with RCU and Virtualization

```
rcu_read_lock();
p = rcu_dereference(gp);
do_something(p->a);
rcu_read_unlock();
```

Nice short RCU read-side critical section

More Fun with RCU and Virtualization

```
rcu_read_lock();
p = rcu_dereference(gp);
```

Hypervisor vCPU preemption for a very long time...

```
do_something(p->a);
rcu_read_unlock();
```

Nice short RCU read-side critical section nevertheless stalls grace period,
with help from the hypervisor!!!
Prasad et al., “The RCU-Reader Preemption Problem in VMs” 2017 USENIX ATC
https://www.usenix.org/conference/atc17/technical-sessions/presentation/prasad

Is This A Real Problem?

- This has not been a problem in the past, but:
  - Cloud providers are increasing utilizations
  - Higher utilization results in increased probability of preemption
- It can be forced to happen in real experiments
  - 2x CPU overcommit: About 50% increase in peak memory footprint
  - (See USENIX ATC paper)
- Cloud-computing economics seems likely to encourage heavy levels of overcommitment
  - A solution would therefore be a good thing

Potential Solution

```
rcu_read_lock();
p = rcu_dereference(gp);
```

Hypervisor vCPU preemption for a very long time...

```
do_something(p->a);
rcu_read_unlock();
```

RCU CPU stall-warning code detects problem and sends hint to the hypervisor.
Experiments ongoing...

Can RCU and CPU Hotplug Survive the Attack of the Killer Virtual Environments?

Can RCU and CPU Hotplug Survive the Attack of the Killer Virtual Environments?

- RCU can't ignore the attack of the killer virtual environments
  - And there have already been RCU changes
  - Brings many hazards of user-mode code into the kernel!
    - In particular, you cannot rely on consistent execution rates
    - Even when you have interrupts diisabled
- Scorecard:
  - RCU, CPU hotplug, and timers:
    - Fixed in v4.8 (4fae16dffb812) and v4.14 (a58163d8ca2c)
  - RCU, CPU hotplug, and virtualization: Fixed except for rcu_barrier()
    - v4.1 (528a25b00e1f) and v4.9 (7ec99de36f40)
  - RCU readers and virtualization: Work in progress
- Survival outlook: Good, but more work needed!
  - Might be worth checking your own code for similar issues...

Can RCU and CPU Hotplug Survive the Attack of the Killer Virtual Environments?

- RCU can't ignore the attack of the killer virtual environments
  - And there have already been RCU changes
  - Brings many hazards of user-mode code into the kernel!
    - In particular, you cannot rely on consistent execution rates
    - Even when you have interrupts diisabled
- Scorecard:
  - RCU, CPU hotplug, and timers:
    - Fixed in v4.8 (4fae16dffb812) and v4.14 (a58163d8ca2c)
  - RCU, CPU hotplug, and virtualization: Fixed except for rcu_barrier()
    - v4.1 (528a25b00e1f) and v4.9 (7ec99de36f40)
  - RCU readers and virtualization: Work in progress
- Survival outlook: Good, but more work needed!
  - Might be worth checking your own code for similar issues...
- RCU continues to spare its maintainer from boredom!!!

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?