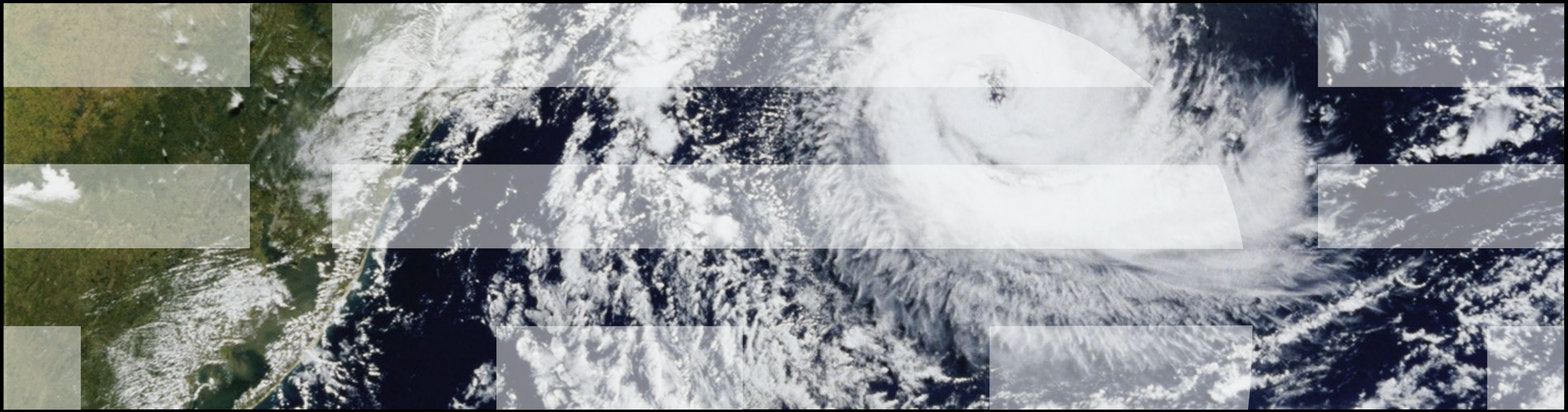
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center

Member, IBM Academy of Technology

Linux Plumbers Conference Real-Time Microconference, September 11, 2019

RCU Configuration, Operation, and Upcoming Changes for Real-Time Workloads



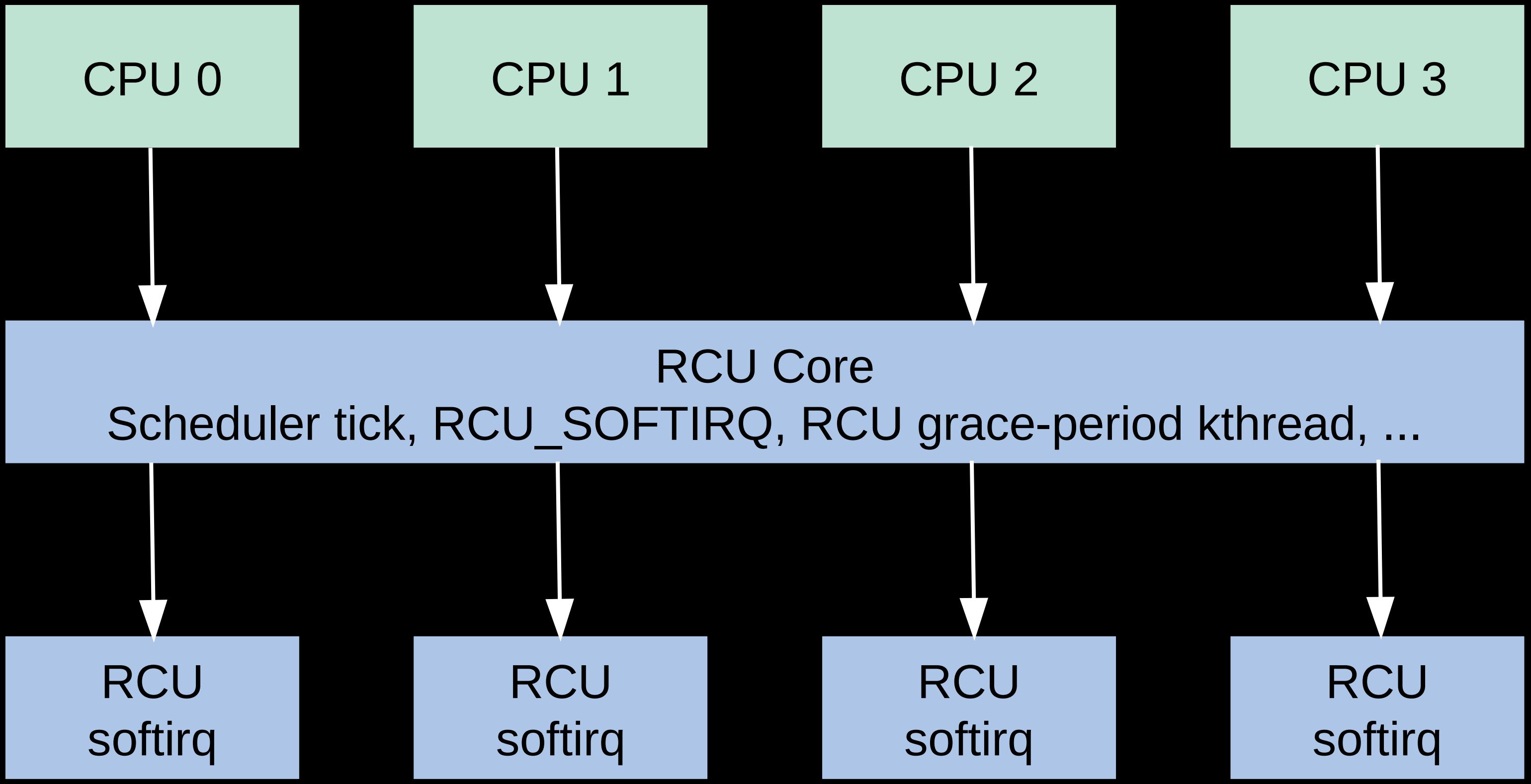
Recent Changes in RCU

- Improving forward progress for offloaded RCU callbacks
- RCU flavor consolidation
- Kernel boot parameters
- Other requests

Improving Forward Progress for Offloaded Callbacks

Default RCU Callback Flow: Self-Throttling!!!

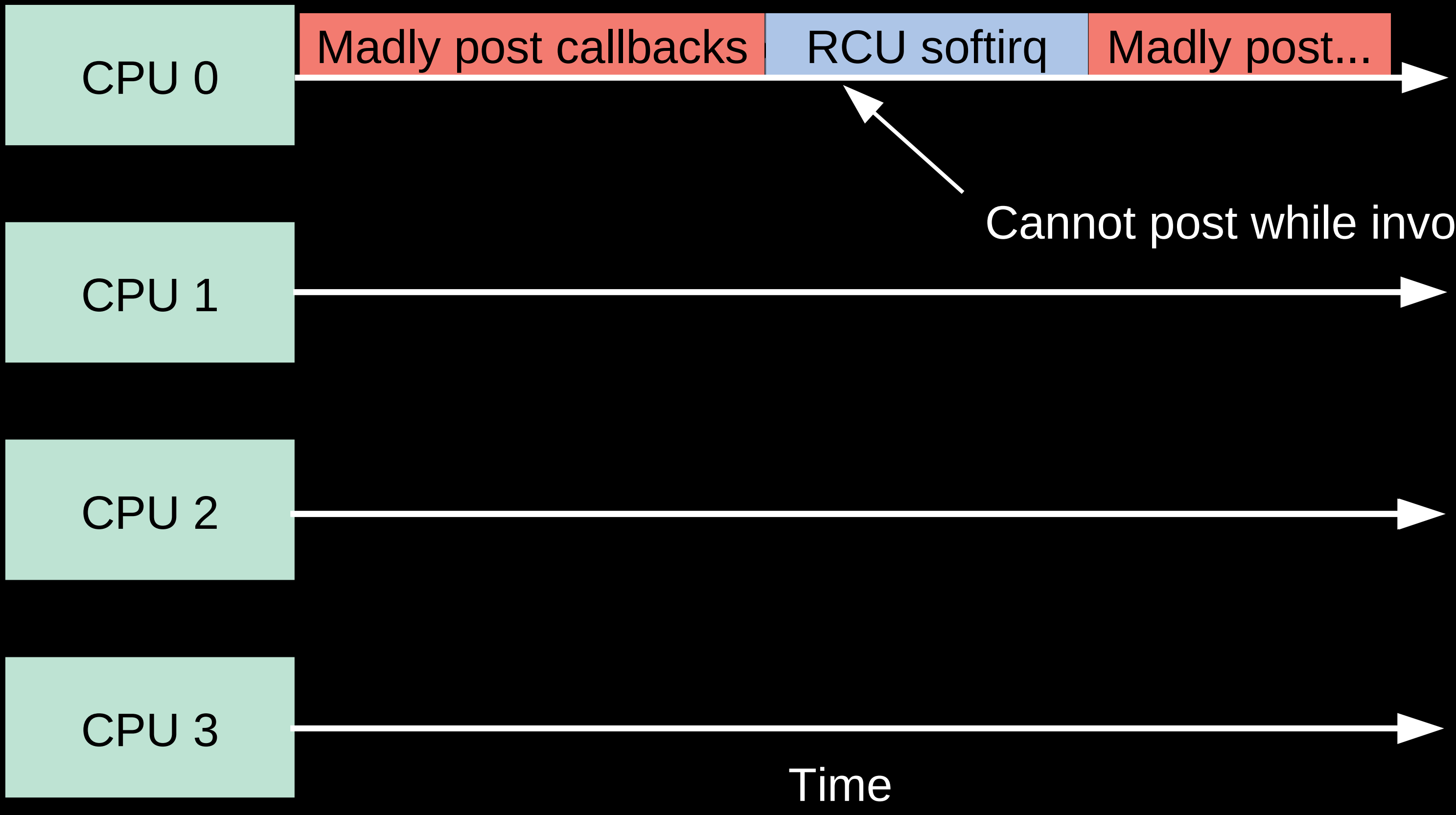
Queue RCU Callbacks



Invoke RCU Callbacks

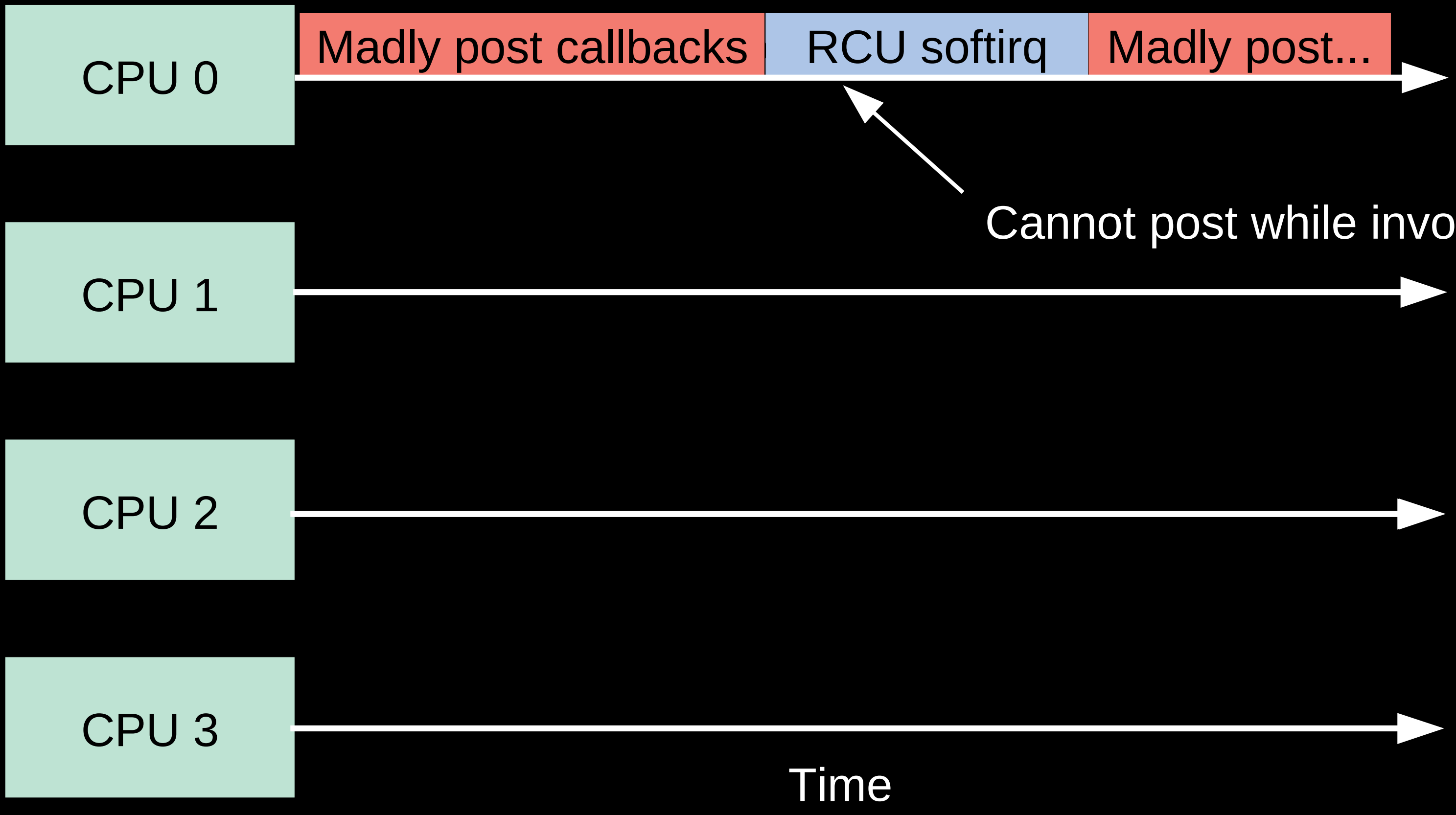
Why self throttling? See next slide...

Default RCU Callback Flow: Self-Throttling!!!



While a CPU is invoking callbacks, it cannot be posting any additional callbacks: Again, self-throttling!!!

Default RCU Callback Flow: Self-Throttling!!!



But horrible real-time properties!!!

(Especially prior to applying Eric Dumazet’s patch)

Which is Why RCU Callbacks Can be Offloaded!!!

RCU callback offloading is intended for tightly controlled embedded systems running highly disciplined applications.

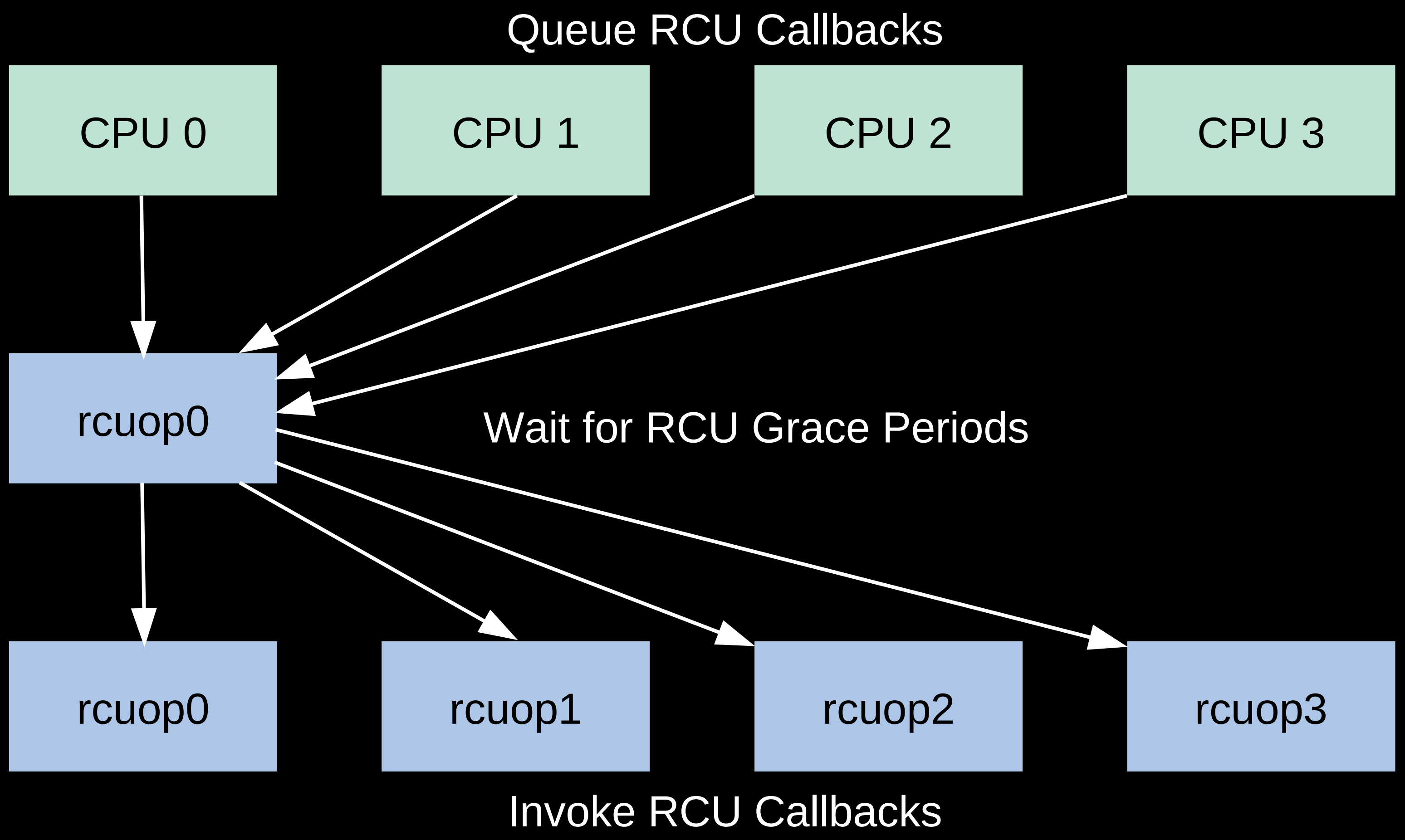
Which is Why RCU Callbacks Can be Offloaded!!!

RCU callback offloading is intended for tightly controlled embedded systems running highly disciplined applications.

As a result, RCU can safely assume a sanely low rate of queuing of RCU callbacks.

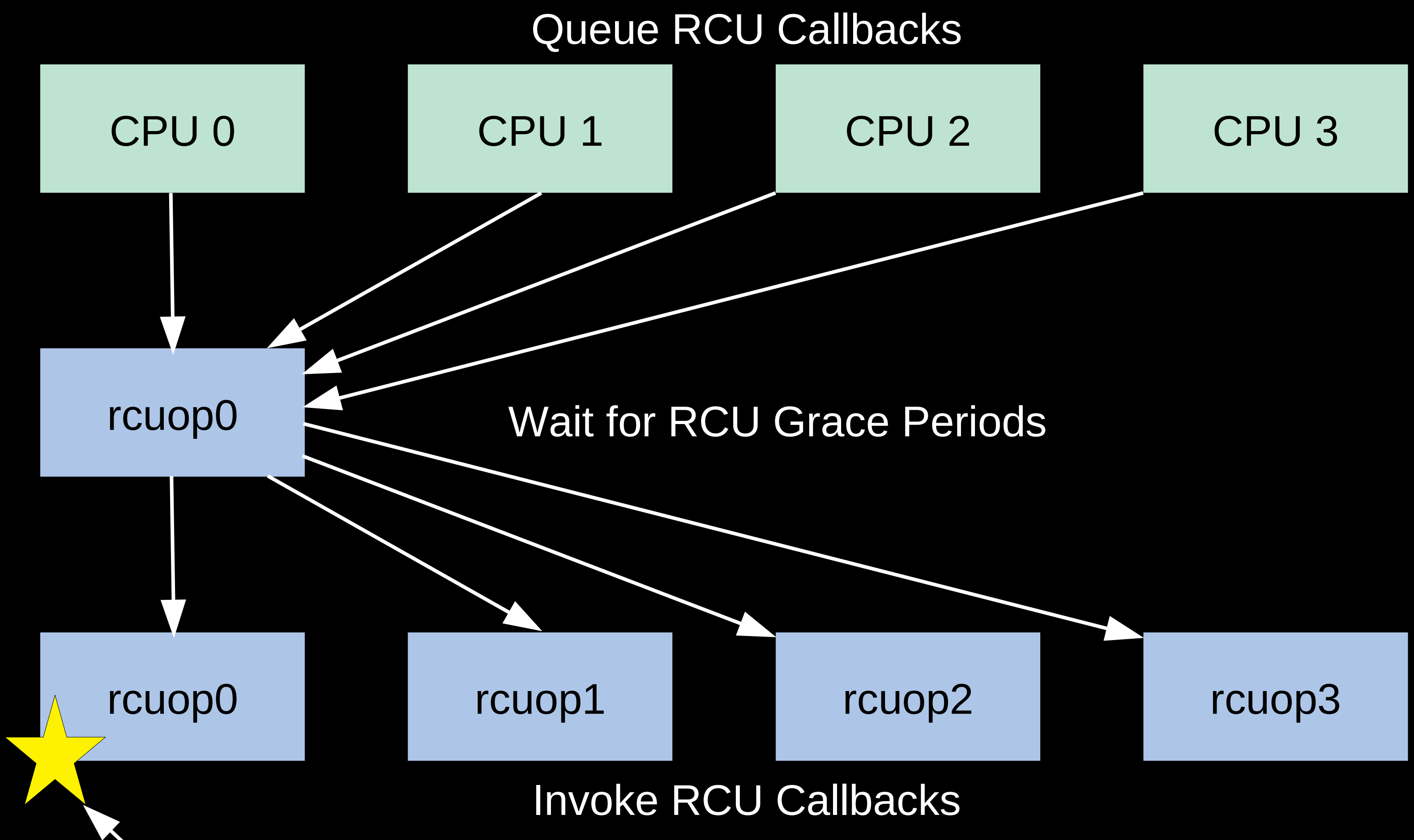
And RCU Callback Offloading Uses This Assumption

And RCU Callback Offloading Uses This Assumption



The administrator may choose rcuo kthread placement and priority

And RCU Callback Offloading Uses This Assumption



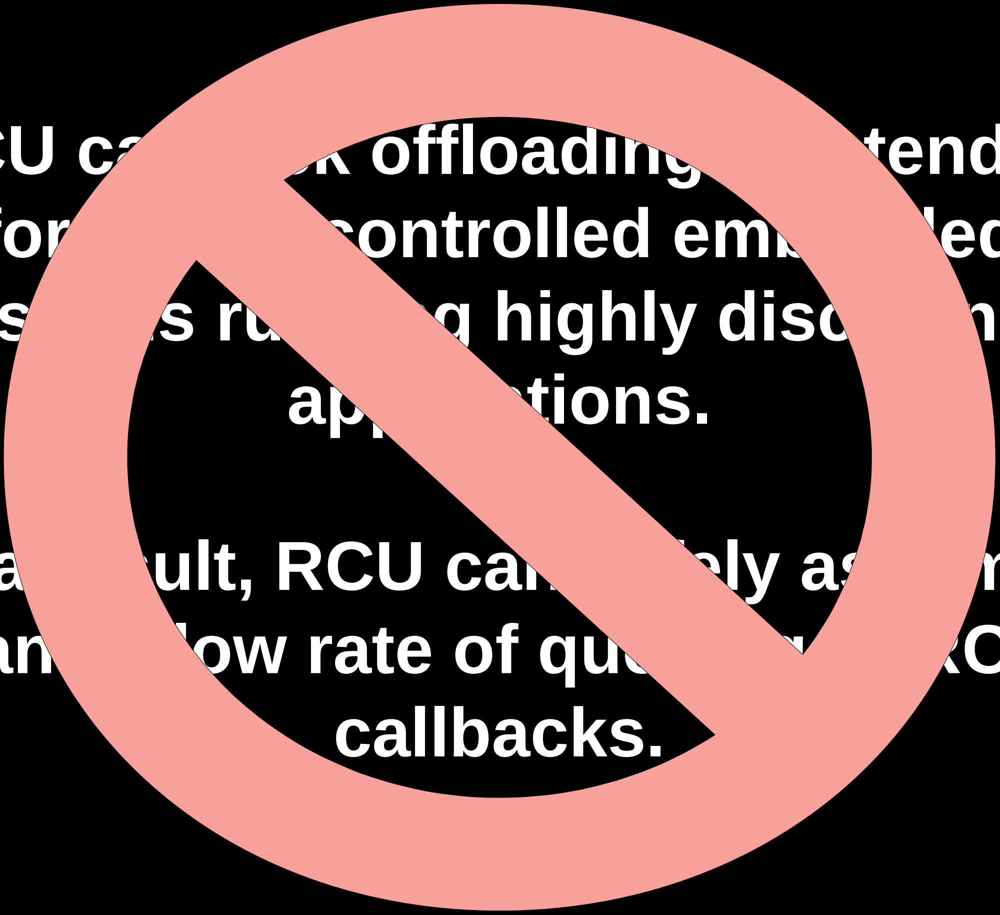
If this assumption is violated, rcuop0 can get stuck here!!!

Improving Forward Progress for Offloaded Callbacks

RCU callback offloading is intended for tightly controlled embedded systems running highly disciplined applications.

As a result, RCU can safely assume a sanely low rate of queuing of RCU callbacks.

Improving Forward Progress for Offloaded Callbacks



RCU ca k offloading tended for controlled emb ded sys s ru g highly disc ned ap tions.

As a ult, RCU ca ely as me a san ow rate of que CU callbacks.

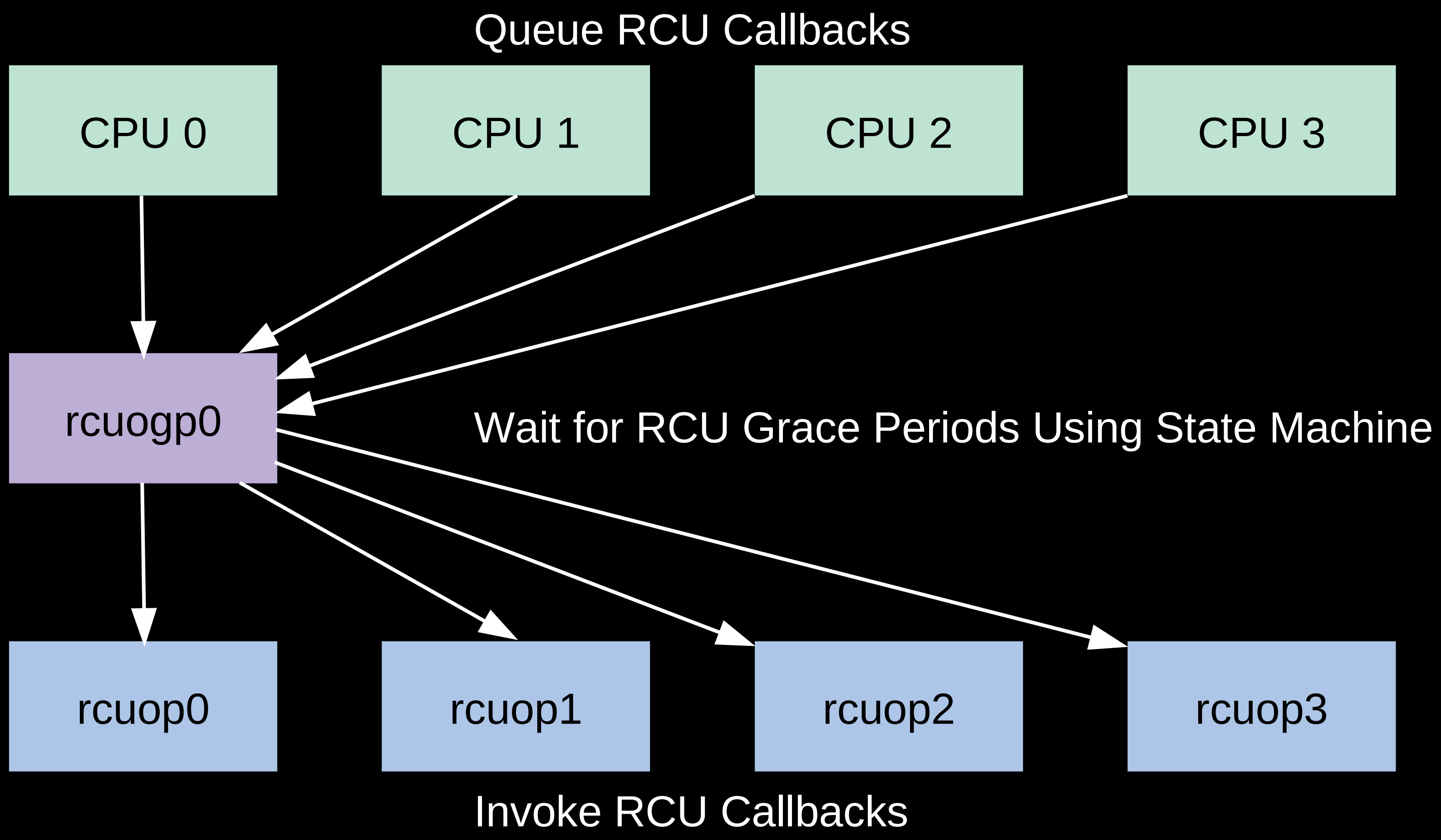
Improving Forward Progress for Offloaded Callbacks

RCU ca k offloading tended
for controlled emb led
sys s ru g highly disc ned
ap tions.

access() ACLs, anyone?

As a ult, RCU ca ely as me a
san low rate of que g CU
callbacks.

RCU Callback Offloading Avoids This Assumption


Queue RCU Callbacks

CPU 0

CPU 1

CPU 2

CPU 3

rcuogp0

Wait for RCU Grace Periods Using State Machine

rcuop0

rcuop1

rcuop2

rcuop3

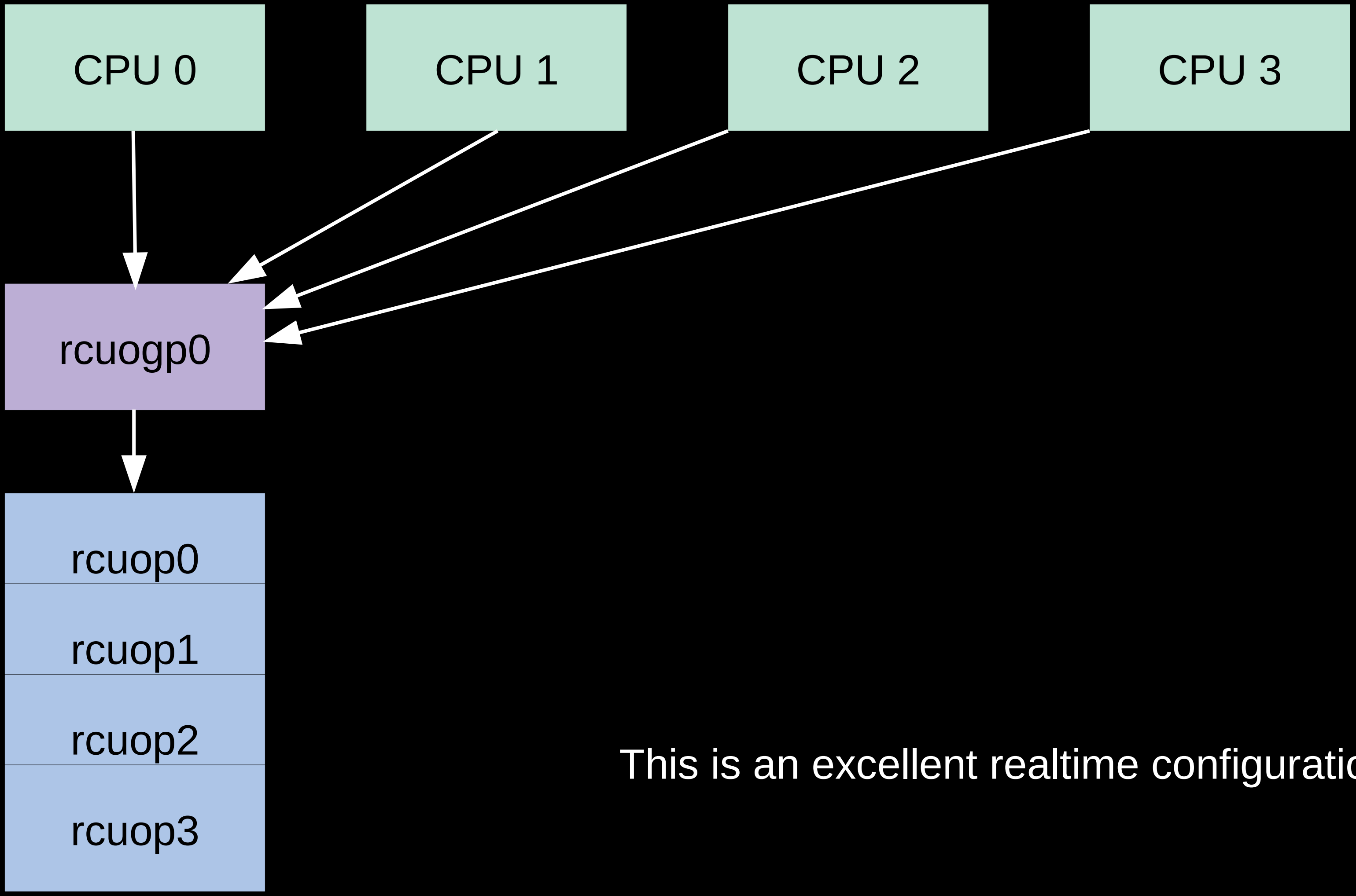
Invoke RCU Callbacks

Other RCU Callback Offloading Changes

- Segmented callback queue
  - Offloaded callbacks now take advantage of others’ grace periods
  - Under heavy load, callbacks can pass from CPU to rcuo kthreads without the rcuog kthread being involved at all
- Bypass queue to reduce lock contention
  - Plus crude contention avoidance by heavy user (the CPU)
  - If necessary, a lockless bypass queue can be used
    - But let’s hold off on any unnecessary complexity!!!
- Turn on scheduler tick for callback invocation
- Currently ~2-3x reduction in callback maximum queue length

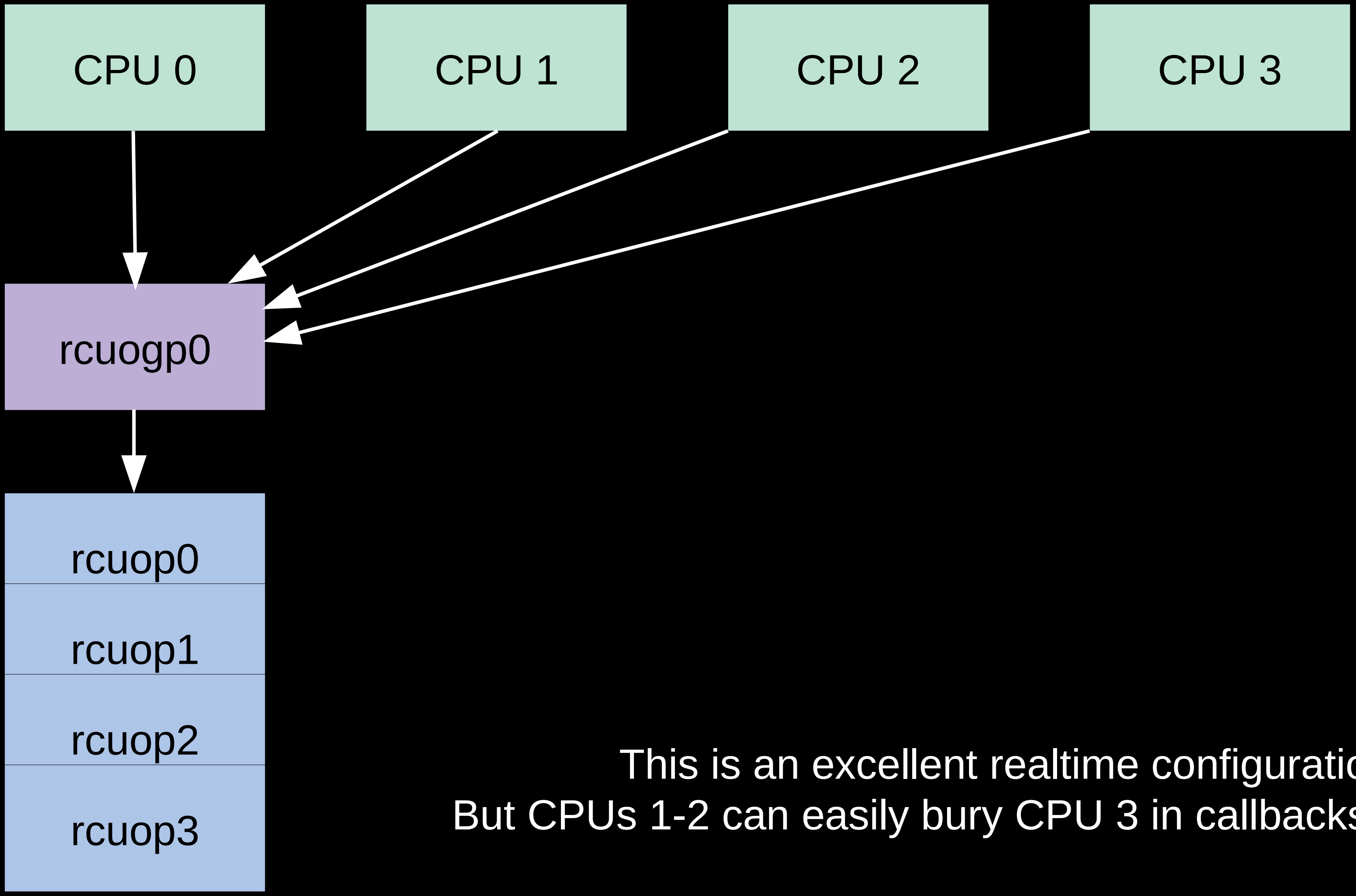
So RCU Callback Offloading is Now Perfect, Right?

So RCU Callback Offloading is Now Perfect, Right?



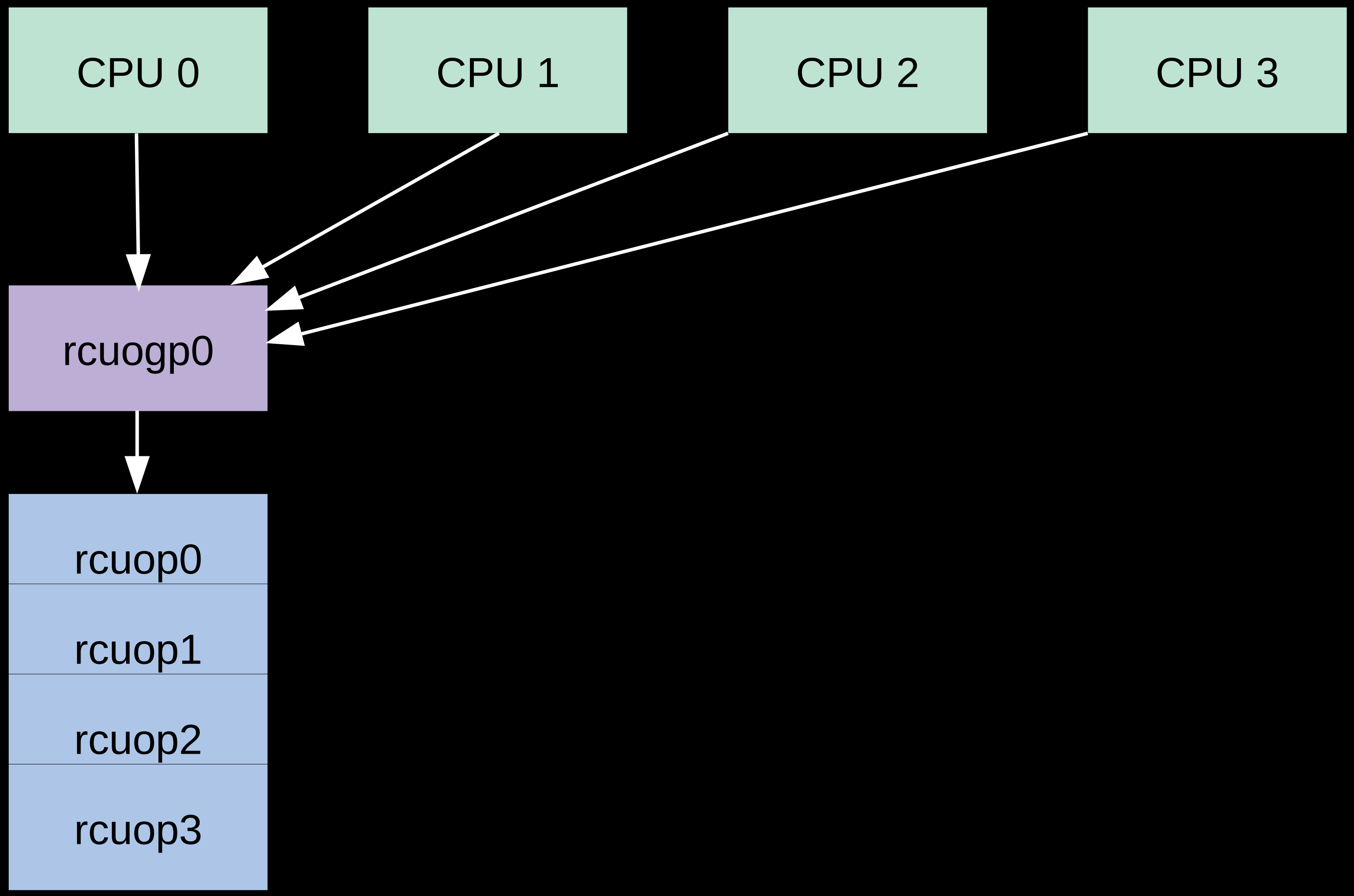
This is an excellent realtime configuration.

So RCU Callback Offloading is Now Perfect, Right?

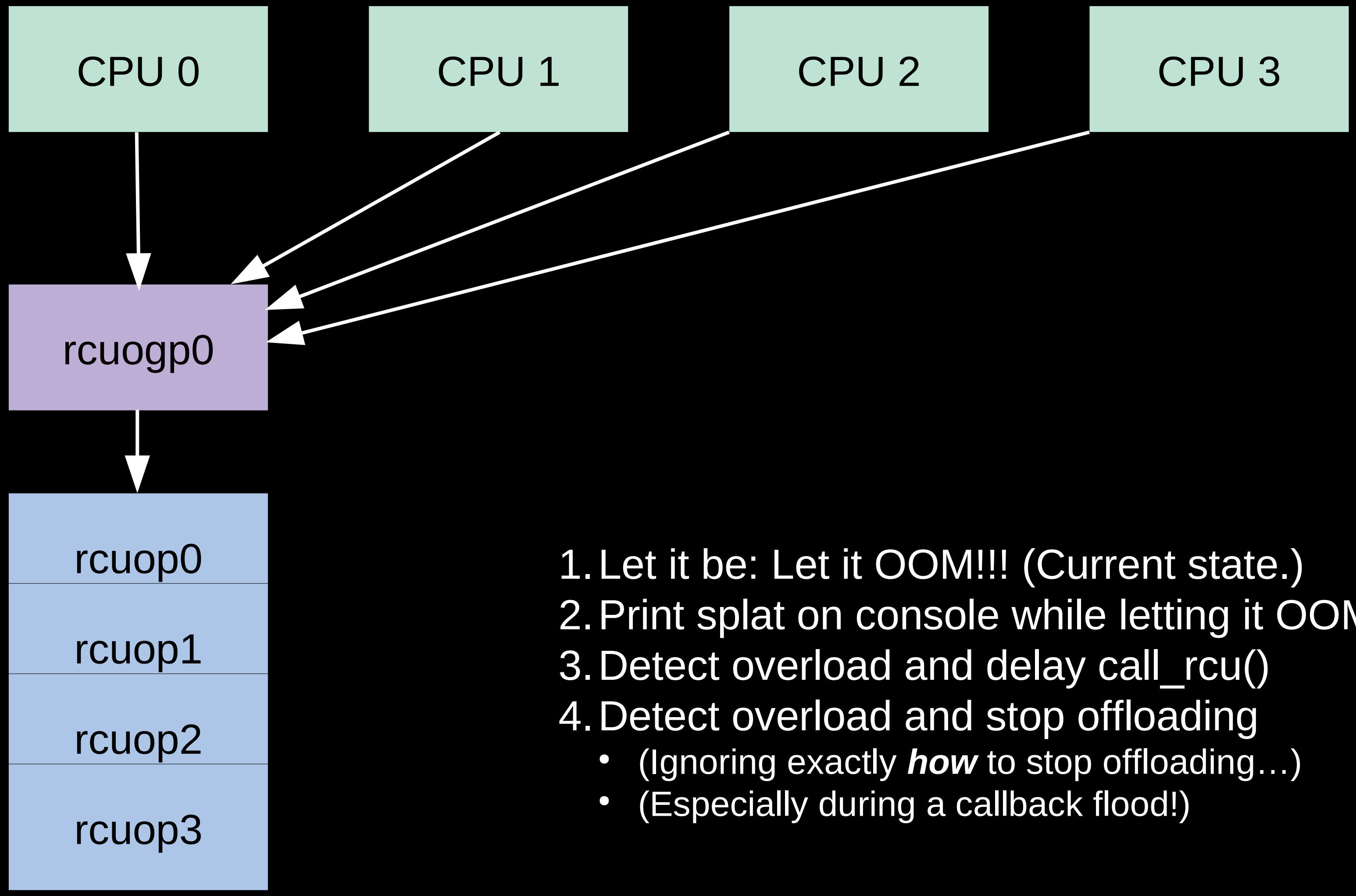


This is an excellent realtime configuration.
But CPUs 1-2 can easily bury CPU 3 in callbacks!!!

What Should RCU Do About Callback-Flooded CPUs?



What Should RCU Do About Callback-Flooded CPUs?



1. Let it be: Let it OOM!!! (Current state.)
2. Print splat on console while letting it OOM
3. Detect overload and delay call_rcu()
4. Detect overload and stop offloading
   - (Ignoring exactly ***how*** to stop offloading…)
   - (Especially during a callback flood!)

RCU Flavor Consolidation

RCU Flavor Consolidation

- Update-side _bh and _sched are no longer:
  - synchronize_rcu() instead of:
    - synchronize_rcu_bh(), ***synchronize_sched()***
  - synchronize_rcu_expedited instead of:
    - synchronize_rcu_bh_expedited, synchronize_sched_expedited()
  - call_rcu() instead of:
    - call_rcu_bh(), call_rcu_sched()
  - rcu_barrier() instead of:
    - rcu_barrier_bh(), rcu_barrier_sched()
  - Get_state_synchronize_rcu() and cond_synchronize_rcu() instead of:
    - get_state_synchronize_sched(), cond_synchronize_sched()
- Read-side _bh and _sched interfaces still work fine
- Greatly reduces the number of RCU offload kthreads (rcuo)

RCU Flavor Consolidation: Possible Issues

- The usual bugs…
  - My test setup is currently a bit lame
  - I expect to be able to fix this before year end
- There is no longer a way to wait for only RCU-sched
  - Because synchronize_rcu() also waits for preempted RCU readers
  - RCU priority boosting is a likely way out (famous last words!)
- Interactions between quiescent-state deferral and -rt
  - For one example, see Scott Wood’s rcutorture patch
  - CONFIG_PREEMPT_RT_BASE should take care of this
  - Perhaps some day mainline will match -rt or vice versa

New RCU Kernel Boot Parameters

New RCU Kernel Boot Parameters

- rcu_nocbs: Now “rcu_nocbs=all” specifies all CPUs
  - Trailing number followed by “-” to say remaining CPUs?
- rcupdate.rcu_cpu_stall_ftrace_dump: Dump ftrace on stall
- rcutree.rcu_kick_kthreads: Extra wakeup if GP kthread slow
- rcutree.rcu_nocb_gp_stride: Was rcutree.rcu_nocb_leader_stride
- rcutree.sysrq_rcu: Take over sysrq-y to dump rcu_node tree
- rcutree.use_softirq: Use rcuc kthreads instead of RCU_SOFTIRQ
- srcutree.counter_wrap_check: How often to check for wrap
- srcutree.exp_holdoff: Auto-expedite holdoff since last GP (ns)

Other RCU-Related Requests

Other RCU-Related Requests

- Warnings for insufficient callback forward progress (Linus)
- Improved RCU CPU stall warnings
  - Subsystem-specific diagnostics? How to determine which subsystem?
    - RCU CPU stall notifier?
  - Expand abbreviations? (Also expands amount of text dumped out!)
  - Other issues?
- rcu_barrier_expedited() – but need real-world use cases
- call_rcu_lazy() for energy efficiency – but need real use cases
  - I have ***never*** seen a CPU having only lazy callbacks queued
- Adapt rcu_node tree to arbitrary hardware layouts
  - I still need a clear demonstration of system-level benefit

Summary

Summary

- I thought that I had a fully functional RCU back in 1997
  - And before that, in 1994!
- Main current focus is forward progress
  - Especially for offloaded RCU callbacks
  - Thinking good thoughts for SCHED_DEADLINE and kthreads
- Some changes to kernel boot parameters
- And the usual miscellaneous requests

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.