Make RCU do less (& later) !

Presenters:

Joel Fernandes (Google)

Uladzislau Rezki (Sony)

Rushikesh Kadam (Intel)

Intel power data courtesy: Sitanshu Nanavati.

Linux Plumbers Conference

Dublin, Ireland September 12-14, 2022

Overview

- Discuss what RCU does at high-level (not how it works!).
- Discuss the 2 main issues we found:
  - On a mostly idle system, RCU activity can disturb the idleness.
    - RCU blocking the scheduler tick ON when idle.
    - RCU constantly queuing callbacks on a lightly loaded system.
- Discuss possible solutions.

What RCU does?

Linux Plumbers Conference
Dublin, Ireland September 12-14, 2022

- RCU reader critical section protected by “**read** lock”
- RCU writer critical section protected by regular locks.
- Reader and writer execute concurrently.
- Writer creates **copy** of obj, writes to it and switches object pointer to new one (release ordered write).
- Writer GCs old object after waiting (**update**)

What RCU does?

- That’s just one use case, there are many uses of RCU.

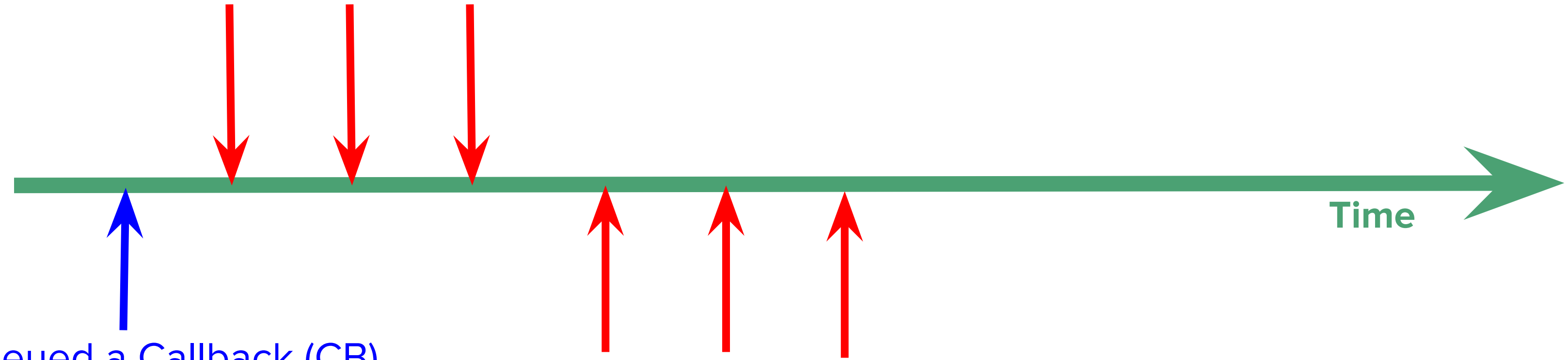
All use cases need same basic tools:

- Lock-less markers of a critical section (CS).
- Start waiting at some point in time (t = T0).
- Stop waiting until all readers that existed at T0 exited CS.

What RCU does?

- On a local CPU (running in kernel mode).

Upper red arrows are timer tick checking are there readers left? If not, report.



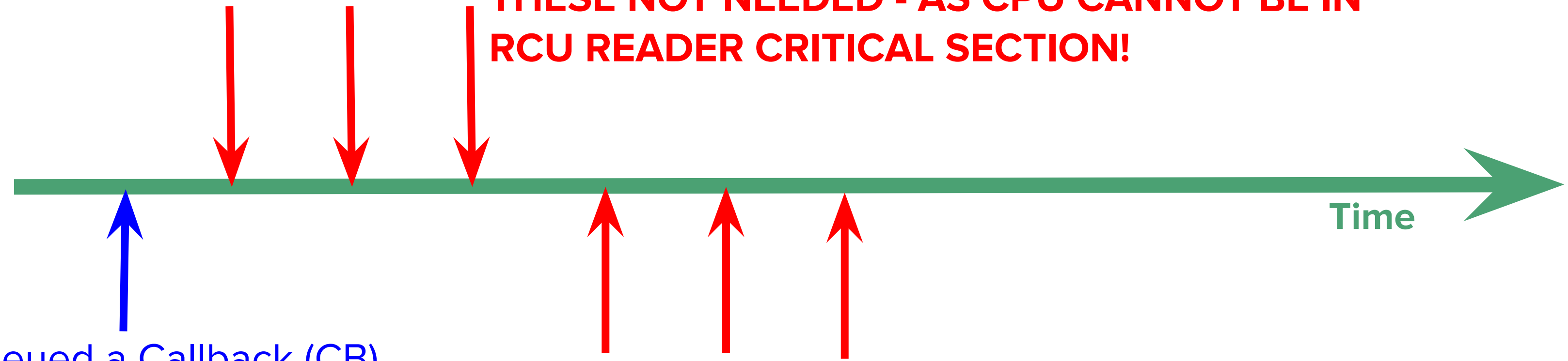
Queued a Callback (CB)

Lower red arrows are timer tick: have ALL CPUs reported? If yes, execute CB. If no, try again.

What RCU does?

- On a local CPU (running in **idle mode**).

Upper red arrows are timer tick checking are there readers left? If not, report.
THESE NOT NEEDED - AS CPU CANNOT BE IN RCU READER CRITICAL SECTION!



Queued a Callback (CB)

Lower red arrows are timer tick: have ALL CPUs reported? If yes, execute CB. If no, try again.
THESE STILL NEEDED - AS local CPU has queued CB.

What RCU does?

Linux Plumbers Conference Dublin, Ireland September 12-14, 2022

- You see the problem?
  - RCU can block the timer tick from getting turned off!
  - Negates power-savings of CONFIG_NOHZ_IDLE

What RCU does?

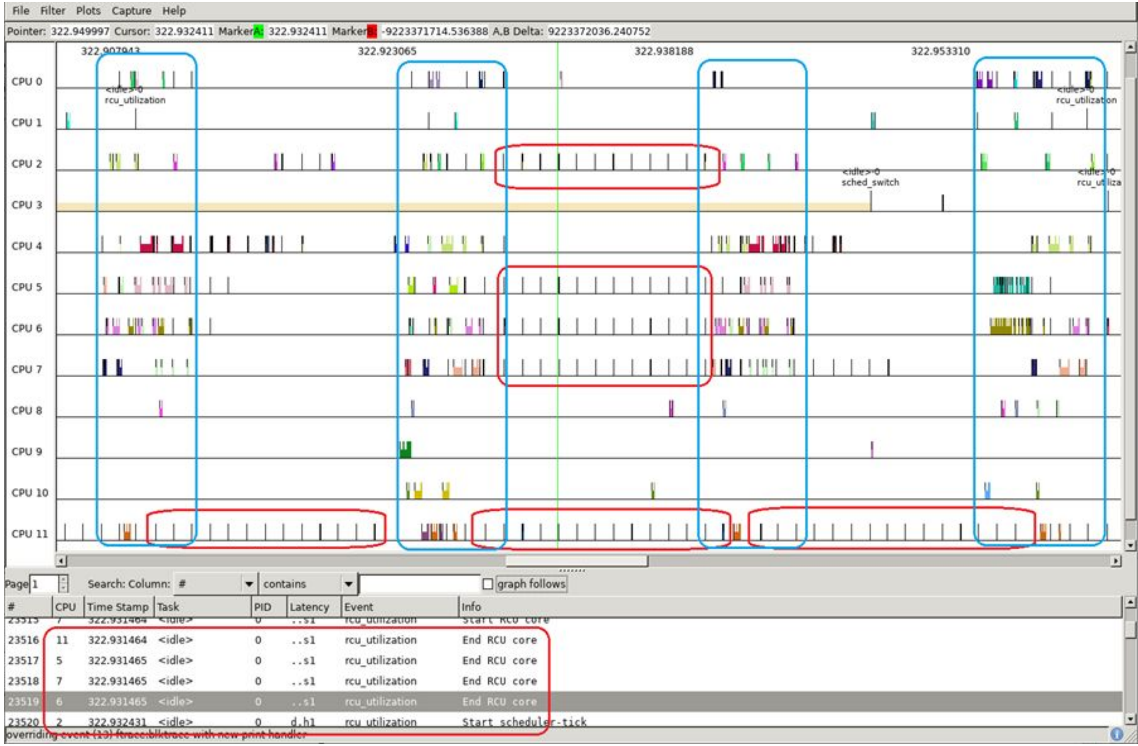
- This happens even in user mode
- NOHZ_FULL systems typically turn tick off. RCU can keep it on.

Issue 1: RCU keeping the scheduler tick ON when idle.

- “Local Video Playback” use-case has 2500+ wakes per second. A large chuck of the wakes result from RCU callbacks blocking the dynticks-idle mode

- RCU wakes are seen at HZ rate (red boxes) between graphics 16.6ms activity (blue boxes)

- Blocks deeper Package C-states. Impacts power



Why was RCU keeping the tick on?

- By default RCU executes callbacks on the same CPU that queued them, in a softirq.

- If there are Callbacks queued , keeping the tick on ensures the CBs are executed in a timely fashion.

Issue 1: RCU keeping the scheduler tick ON when idle.

Possible solution: Using `CONFIG_FAST_NOHZ` option

- This option permits CPUs to enter the dyntick-idle state (the state where the tick is turned off) even if they have CBs queued.

- Idle CPUs with callbacks are kept idle for a minimum number of jiffies before rechecking of the RCU state.

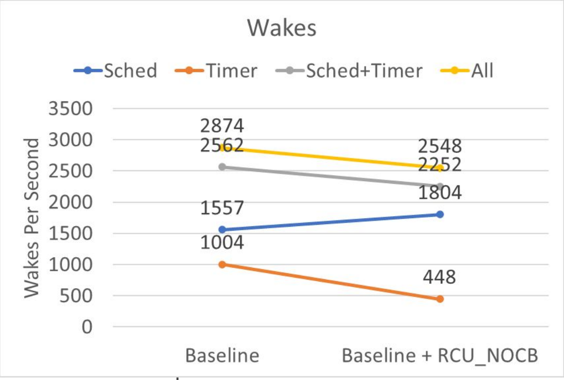
Issue 1: RCU keeping the scheduler tick ON when idle.

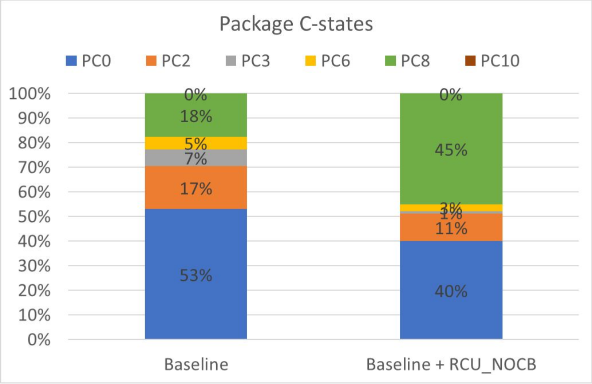
Solution for newer kernels:

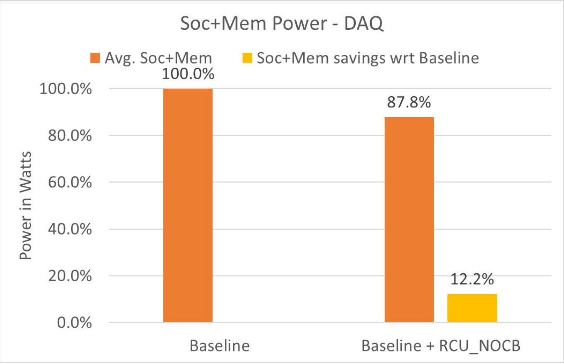
- `CONFIG_FAST_NO_HZ` is removed in recent kernels.
- `CONFIG_RCU_NOCB_CPU` : Execute RCU CBs in per-cpu threads.
- Scheduler may or may not move threads to non-idle CPUs and is in control of whether CPU needs to be idle or execute callbacks.
- **Both** starting of new grace periods, and executing CBs are moved out of the softirq context and into threads.

Issue 1: RCU keeping the scheduler tick ON when idle.

- RCU callback offload unblocks dynticks-idle and hence reduces t wakes.
- RCU callback offload does increa scheduler wakes marginally, but total platform wakes.
- Improves Package C-states resid hence SoC + Memory power.







<u>Use-case</u>: Local video playback via Chrome browser, VP9 1080p @ 30 fps content

<u>Device</u>: Chrome reference device, AlderLake Hybrid CPU with 2 Cores (with Hyperthreading) + 8 Atoms

Observation: ChromeOS when idle

Linux Plumbers Conference
Dublin, Ireland September 12-14, 2022

- Several callbacks constantly queued.
- ChromeOS login + screenoff
- Device is on battery.

rcutop refreshing every 5 seconds

```
21:57:07 loadavg: 0.06 0.50 0.55 2/629 8945

Callback                           Queued  Executed
inode_free_by_rcu                  7       10
delayed_put_task_struct            7       15
k_itimer_rcu_free                  9       9
radix_tree_node_rcu_free           16      27
rcu_work_rcufn                     1       2
put_cred_rcu                       4       8
delayed_put_pid                    7       15
unbind_fence_free_rcu              4       5
dst_destroy_rcu                    4       10
__i915_gem_free_object_rcu         5       10
thread_stack_free_rcu              3       7
```

Observation: ChromeOS Display pipeline

Display pipeline in ChromeOS constantly opens/close graphics buffers.

```
VizCompositorTh-1999  [006]  1472.325451: sys_enter_close:      fd: 0x00000033
VizCompositorTh-1999  [006]  1472.325457: sys_enter_close:      fd: 0x00000046
ThreadPoolSingl-6857  [010]  1472.325734: sys_enter_close:      fd: 0x00000025
ThreadPoolSingl-6857  [010]  1472.325743: rcu_callback:         rcu_preempt rhp=0xffff9f3edc718480 func=file_free_rcu 1
          chrome-1975  [000]  1472.344365: sys_enter_close:      fd: 0x0000002d
       DrmThread-1993  [002]  1472.344627: sys_enter_close:      fd: 0x00000044
       DrmThread-1993  [002]  1472.344844: sys_enter_close:      fd: 0x00000044
          chrome-1975  [000]  1472.345019: sys_enter_close:      fd: 0x00000046
VizCompositorTh-1999  [006]  1472.345071: sys_enter_close:      fd: 0x00000046
VizCompositorTh-1999  [006]  1472.345088: sys_enter_close:      fd: 0x00000044
     kworker/10:2-2105  [010]  1472.346603: rcu_callback:       rcu_preempt rhp=0xffff9f41efa9f600 func=rcu_work_rcufn 1
      kworker/9:4-3546  [009]  1472.346603: rcu_callback:       rcu_preempt rhp=0xffff9f41efa5f600 func=rcu_work_rcufn 1
      kworker/0:4-3506  [000]  1472.346606: rcu_callback:       rcu_preempt rhp=0xffff9f41ef81f600 func=rcu_work_rcufn 1
       DrmThread-1993  [002]  1472.357990: sys_enter_close:      fd: 0x0000002e
       DrmThread-1993  [002]  1472.358005: rcu_callback:         rcu_preempt rhp=0xffff9f3eb9328000 func=file_free_rcu 1
          chrome-1975  [000]  1472.358200: sys_enter_close:      fd: 0x00000038
VizCompositorTh-1999  [006]  1472.358367: sys_enter_close:      fd: 0x0000002e
          chrome-1975  [000]  1472.358539: sys_enter_close:      fd: 0x00000044
          chrome-1975  [000]  1472.358546: sys_enter_close:      fd: 0x0000002e
          chrome-1975  [000]  1472.358548: sys_enter_close:      fd: 0x00000038
VizCompositorTh-1999  [006]  1472.358778: sys_enter_close:      fd: 0x0000002e
VizCompositorTh-1999  [006]  1472.358784: sys_enter_close:      fd: 0x00000046
ThreadPoolSingl-6857  [010]  1472.359008: sys_enter_close:      fd: 0x00000025
ThreadPoolSingl-6857  [010]  1472.359019: rcu_callback:         rcu_preempt rhp=0xffff9f3e8d28e300 func=file_free_rcu 1
          chrome-1975  [000]  1472.377594: sys_enter_close:      fd: 0x0000002d
       DrmThread-1993  [002]  1472.377825: sys_enter_close:      fd: 0x0000003f
       DrmThread-1993  [002]  1472.378043: sys_enter_close:      fd: 0x0000003f
          chrome-1975  [000]  1472.378227: sys_enter_close:      fd: 0x00000046
VizCompositorTh-1999  [006]  1472.378341: sys_enter_close:      fd: 0x00000046
VizCompositorTh-1999  [006]  1472.378356: sys_enter_close:      fd: 0x0000003f
      kworker/2:1-7250  [002]  1472.378524: rcu_callback:       rcu_preempt rhp=0xffff9f41ef89f600 func=rcu_work_rcufn 1
      kworker/0:4-3506  [000]  1472.379626: rcu_callback:       rcu_preempt rhp=0xffff9f41ef81f600 func=rcu_work_rcufn 1
     kworker/10:2-2105  [010]  1472.380627: rcu_callback:       rcu_preempt rhp=0xffff9f41efa9f600 func=rcu_work_rcufn 1
       DrmThread-1993  [002]  1472.391294: sys_enter_close:      fd: 0x00000033
       DrmThread-1993  [002]  1472.391306: rcu_callback:         rcu_preempt rhp=0xffff9f3eb9328600 func=file_free_rcu 1
```

Observation: Logging in Android (as example)

Example: Logging during static image (Android).

Static image is important use-case for power testing on Android. The system is mostly idle to minimize a power drain of the platform:

- Panel refresh-rate is zero, i.e. it is stopped and power collapsed
- CPUs spend most of their time in deepest C-state
- SoC bandwidth is minimal (memory bus, CPU/cache frequencies, etc.).

Logging does constant file open/close inducing RCU pressure when FDs get freed. As a side effect of such periodic light load, many wakeups happen due to frequent kicking an RCU-core for initializing a GP to invoke callbacks after it passes.

Observation: Logging in Android (as example)

Below is a wakeup trace log of static image use-case during 30 seconds.

```
<wake-up-trace-log>
 rcuop/2     pid:       33    woken-up    36709      interval: min   1320      max       71837          avg
9807
 rcuop/3     pid:       40    woken-up    36944      interval: min   1582      max       78649          avg
9744
 rcuop/0     pid:       15    woken-up    40570      interval: min   1520      max       80442          avg
8873
 rcuop/1     pid:       26    woken-up    40695      interval: min   1414      max       80043          avg
8846
 rcuog/0     pid:       14    woken-up    57907      interval: min     73      max       27855          avg
6217
 idd@1.0.    pid:     1116    woken-up    89498      interval: min    231      max    17442186          avg
4005
 rcu_preempt pid:       13    woken-up    90203      interval: min     39      max        8505          avg
3991
 iddd        pid:     1195    woken-up    250398     interval: min     92      max       16375          avg
1437
<wake-up-trace-log>
```

A trace was taken on the ARM big.LITTLE system. It is obvious that the biggest part belongs to the “iddd logger” whereas a second place is fully owned by the RCU-core subsystem marked as red

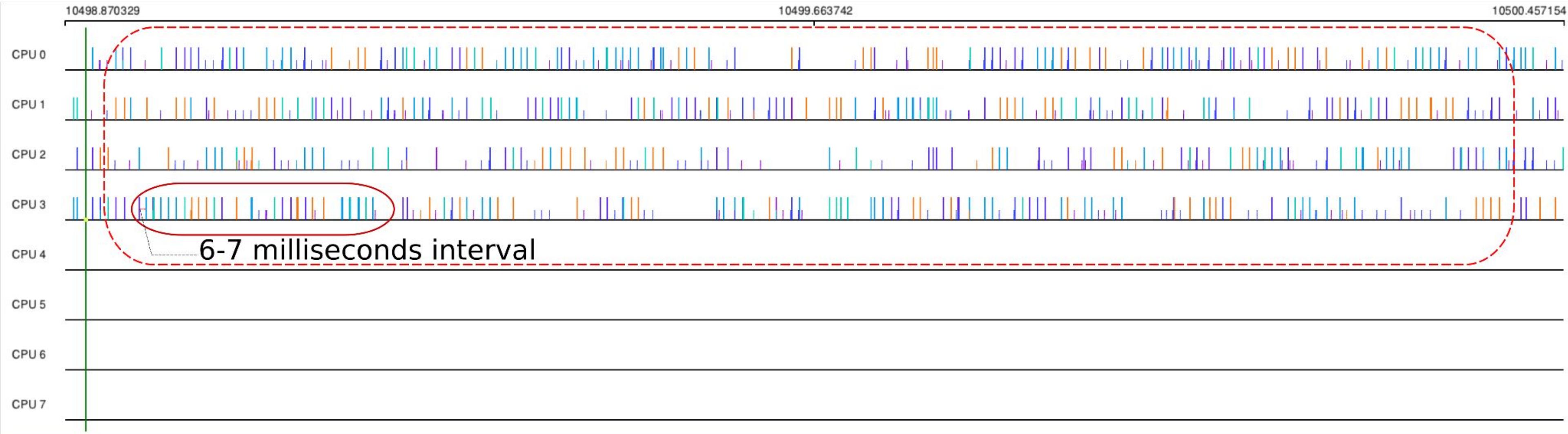
Observation: Logging in Android (as example)

RCU mostly invokes callbacks related to the VFS, SELinux subsystems during logging:

- `file_free_rcu()`
- `inode_free_by_rcu()`
- `i_callback()`
- `__d_free()`
- `avc_node_free()`

Since system is lightly loaded and a number of posted callbacks to be invoked are rather small, between 1-10, such pattern produce most of the wakeups (in static image use-case) to offload a CPU with __only__ few callbacks there.

Observation: Logging in Android

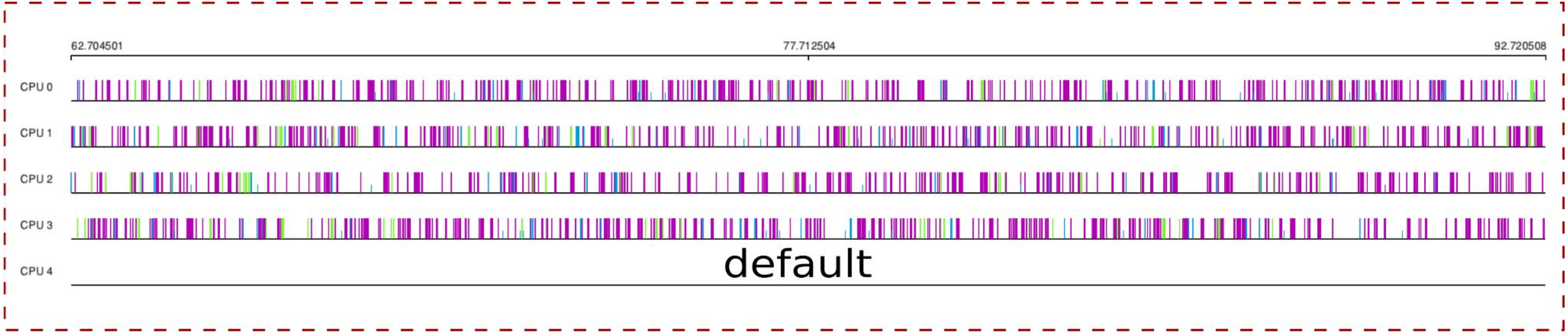


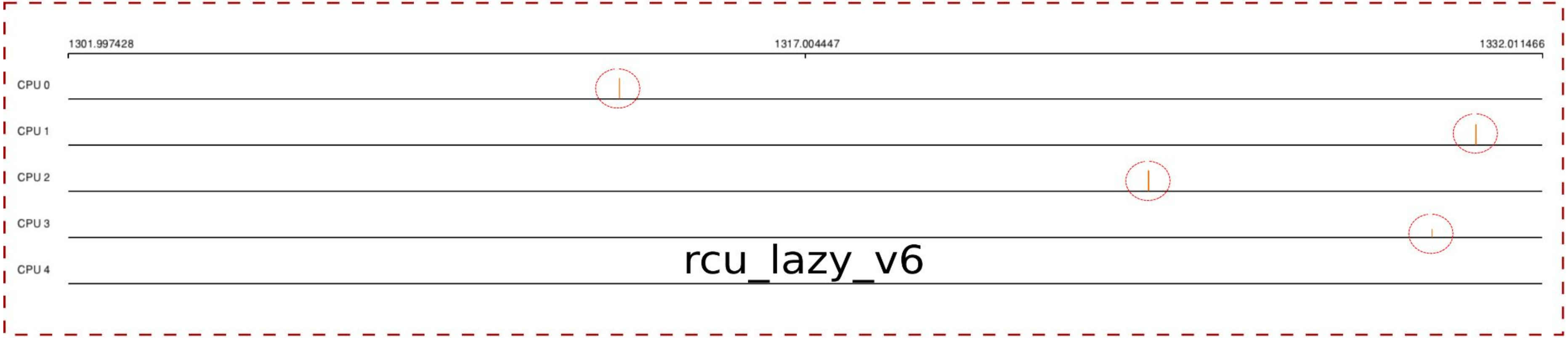
Search: Column # contains Next Prev ✓Graph follows

#	CPU	Time Stamp	Task	PID	Latency	Event	Info
	2	10498.892424	rcuop/2	33	d..1	rcu/rcu_batch_start	rcu_preempt CBs=32 bl=10
	3	10498.892637	rcuop/3	40	d..1	rcu/rcu_batch_start	rcu_preempt CBs=10 bl=10
	1	10498.900003	rcuop/1	26	d..1	rcu/rcu_batch_start	rcu_preempt CBs=18 bl=10
	0	10498.900056	rcuop/0	15	d..1	rcu/rcu_batch_start	rcu_preempt CBs=10 bl=10
	2	10498.900083	rcuop/2	33	d..1	rcu/rcu_batch_start	rcu_preempt CBs=10 bl=10
	3	10498.900177	rcuop/3	40	d..1	rcu/rcu_batch_start	rcu_preempt CBs=18 bl=10
	0	10498.908245	rcuop/0	15	d..1	rcu/rcu_batch_start	rcu_preempt CBs=10 bl=10
	2	10498.908385	rcuop/1	26	d..1	rcu/rcu_batch_start	rcu_preempt CBs=17 bl=10
	0	10498.908493	rcuop/2	33	d..1	rcu/rcu_batch_start	rcu_preempt CBs=11 bl=10
	3	10498.908536	rcuop/3	40	d..1	rcu/rcu_batch_start	rcu_preempt CBs=25 bl=10
	0	10498.916187	rcuop/0	15	d..1	rcu/rcu_batch_start	rcu_preempt CBs=6 bl=10
	2	10498.916369	rcuop/1	26	d..1	rcu/rcu_batch_start	rcu_preempt CBs=17 bl=10
	3	10498.916574	rcuop/3	40	d..1	rcu/rcu_batch_start	rcu_preempt CBs=29 bl=10

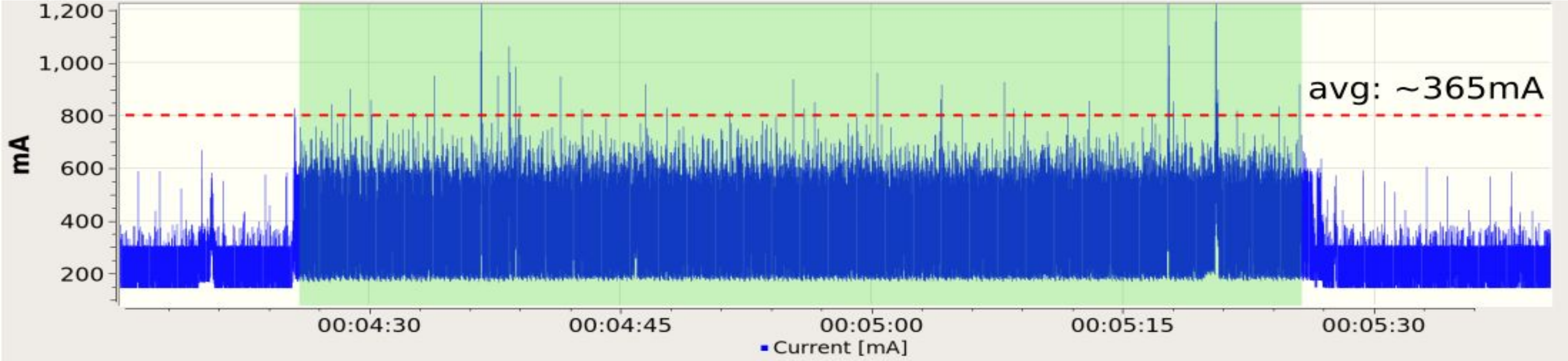
Only a few callbacks are invoked

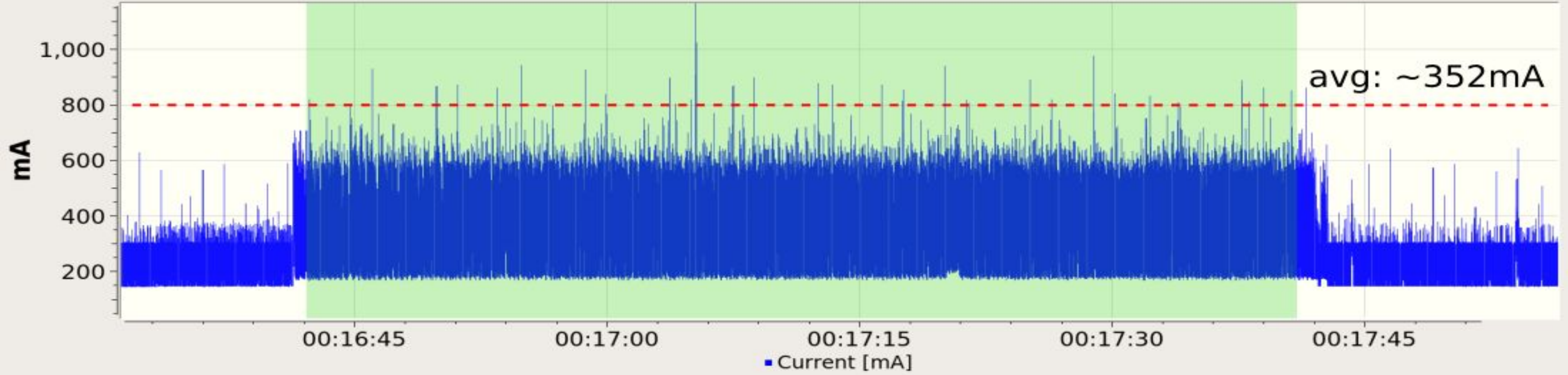
Home screen swipe(as example)





Home screen swipe power(~3% delta)





Issue 2: RCU queuing CBs on lightly loaded system

- Observation: When a system is relatively idle, most CBs in the system don't need to be executed soon, we can in fact delay them as long as needed.

- Selectively identify frequently occurring CBs in the system that "trickle".

Issue 2: RCU queuing CBs on lightly loaded system

Solution 1: Delay RCU processing using `jiffies_till_{first,next}_fqs`

- Great power savings

`jiffies_till_first_fqs` & `jiffies_till_next_fqs`	= 3,3 (default)	= 8, 8	= 16, 16	= 24, 24	= 32, 32
SoC+Memory, power savings w.r.t Baseline	11.1%	13.1%	13.9%	14.4%	14.2%

- Problem:
  - Causes slow down in ALL call_rcu() users globally whether they like it or not.
  - Causes slow down in synchronize_rcu() users globally.
  - Significantly regresses boot time.

Issue 2: RCU queuing CBs on lightly loaded system

Solution 1: Jiffies causes massive synchronize_rcu() slowdown.

- ChromeOS tab switching autotest
  - Due to `synchronize_rcu()` latency increases quickly from 23 ms to 169 ms (with changing jiffies from 3 to 32)

- The same evaluation with `synchronize_rcu_expedited()` gives us a latency of < 1 msec at jiffies = 32

Issue 2: RCU queuing CBs on lightly loaded system
Solution 1: Jiffies increase causing function tracer issues

Several paths in ftrace code uses `synchronize_rcu()`:

For but 2 examples:

- `pid_write()` triggered by write to `/sys/kernel/tracing/debug/tracing/set_ftrace_pid`
- ring buffer code such as `ring_buffer_resize()`

End result is `trace-cmd record -p function_graph` can take several more seconds to start and stop recording, than it would otherwise.

Issue 2: RCU queuing CBs on lightly loaded system

Solution 1: Jiffies causing boot-time issues (SELinux)

SELinux enforcing during ChromeOS boot up invokes synchronize_rcu()

```
[   17.715904]  => __wait_rcu_gp
[   17.715904]  => synchronize_rcu
[   17.715904]  => selinux_netcache_avc_callback
[   17.715904]  => avc_ss_reset
[   17.715904]  => sel_write_enforce
[   17.715904]  => vfs_write
[   17.715904]  => ksys_write
[   17.715904]  => do_syscall_64
```

Issue 2: RCU queuing CBs on lightly loaded system

Solution 1: Jiffies causing per-cpu refcount regression

- RCU used to toggle atomic-mode and vice versa
- Can badly hurt paths that don't really want to free memory but use call_rcu() for some other purposes. Like suspend.
- `call_rcu()` slow down affects percpu refcounters
- These counters use RCU when switching to atomic-mode
  - `__percpu_ref_switch_mode() -> percpu_ref_switch_to_atomic_sync().`
- This call slows down for the per-cpu refcount users such as `blk_pre_runtime_suspend().`

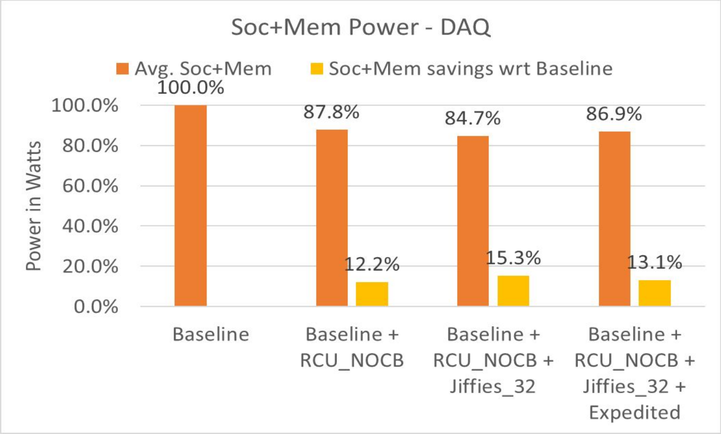
This is why, we cannot assume `call_rcu()` users will mostly just want to free memory. There could be cases just like this, and blanket slow down of call_rcu() might bite unexpectedly.

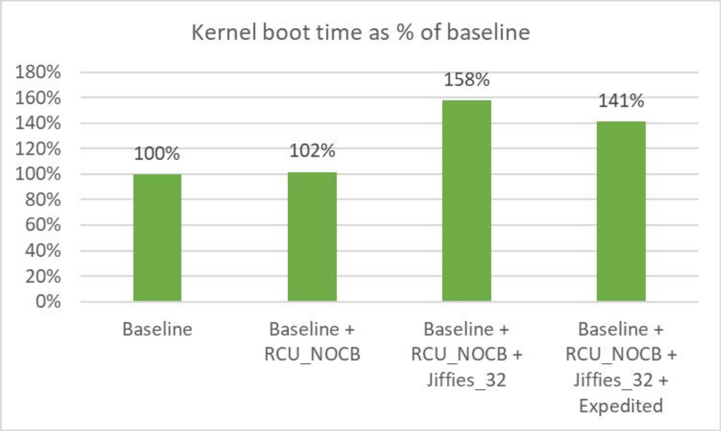
Issue 2: RCU queuing CBs on lightly loaded system
Solution 1: Jiffies with expedited option

- The previous synchronize_rcu() issues can be mitigated by usi while ensuring good power efficiency.

- However, experiments showed that using expedited RCU with

- Also, the expedited option is expensive, and can affect real-tim





Issue 2: RCU queuing CBs on lightly loaded system
Solution 2: Delay RCU CB processing (Lazy RCU)

Latest Patches:

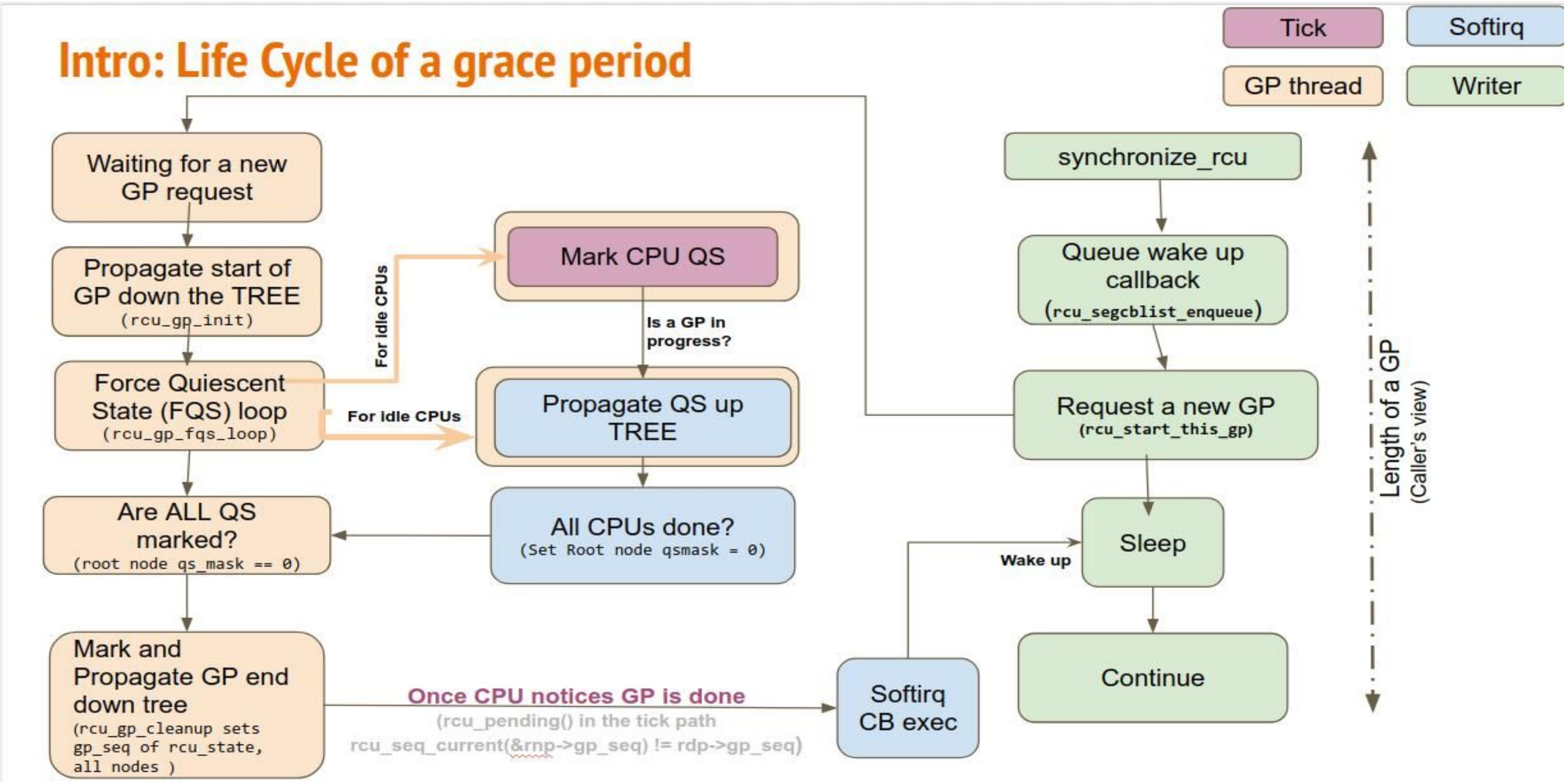
https://lore.kernel.org/all/20220819204857.3066329-1-joel@joelfernandes.org/

Summary:

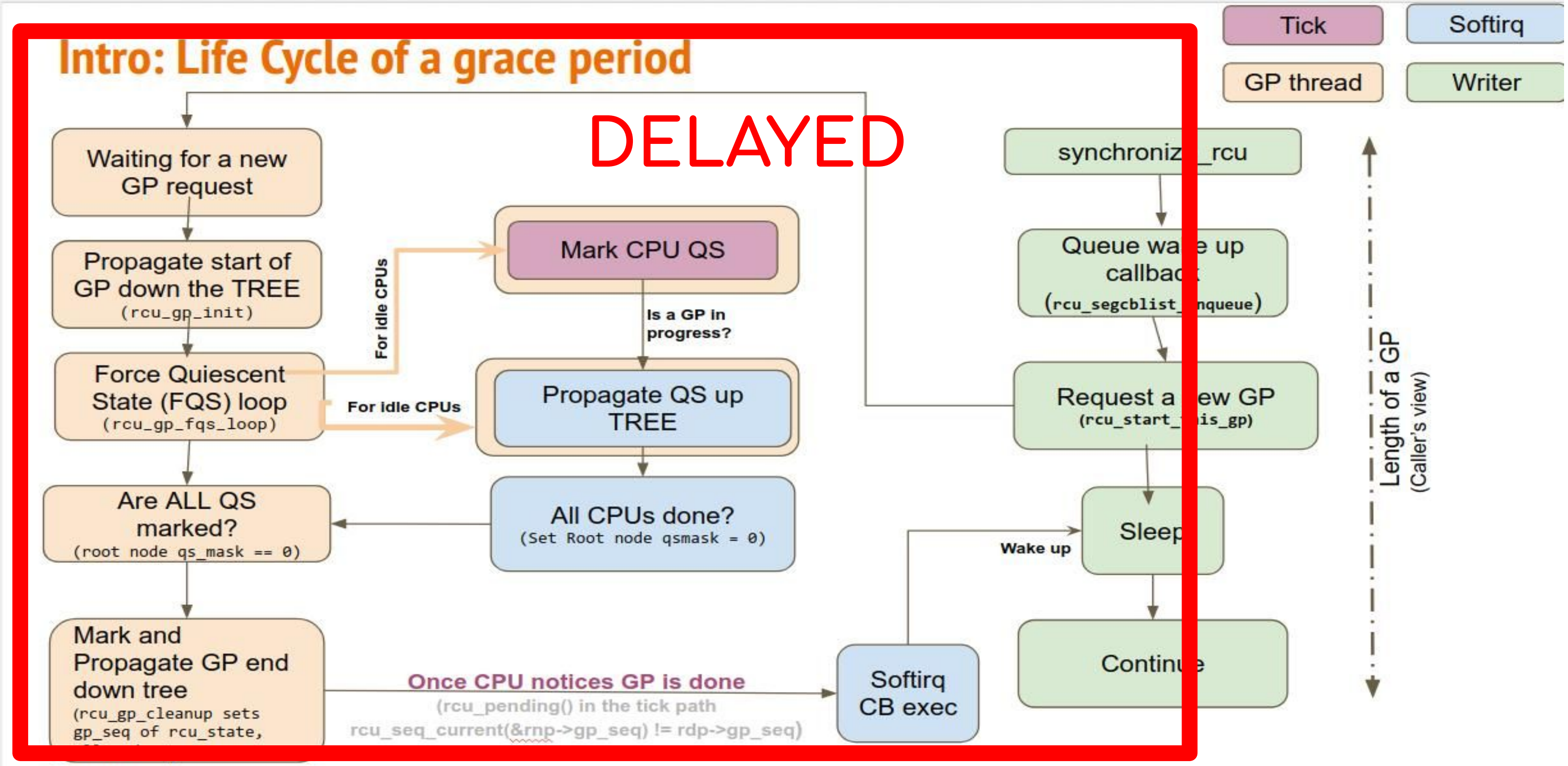
- Introduce new API for lazy-RCU (`call_rcu_lazy`).
- Queue CBs into the Bypass list.
- Flush the Bypass list when:
  - Non-Lazy CBs show up.
  - Bypass list grows too big.
  - Memory is low.
- Several corner cases now handled (`rcu_barrier`, CPU hotplug etc).

Issue 2: RCU queuing CBs on lightly loaded system

Solution 2: Delay RCU CB processing (Lazy RCU)



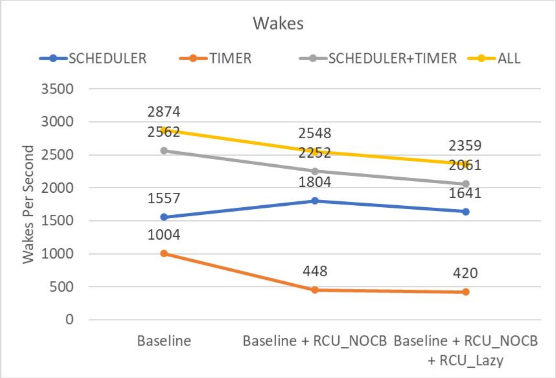
Issue 2: RCU queuing CBs on lightly loaded system
Solution 2: Delay RCU CB processing (Lazy RCU)

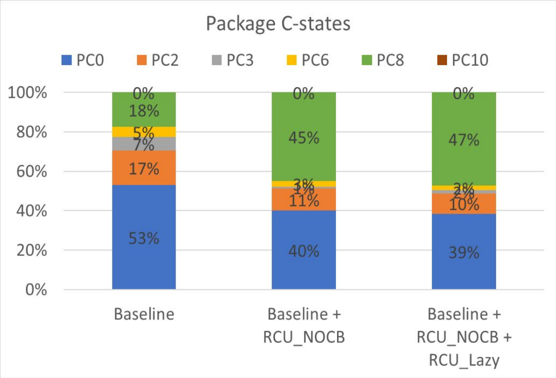


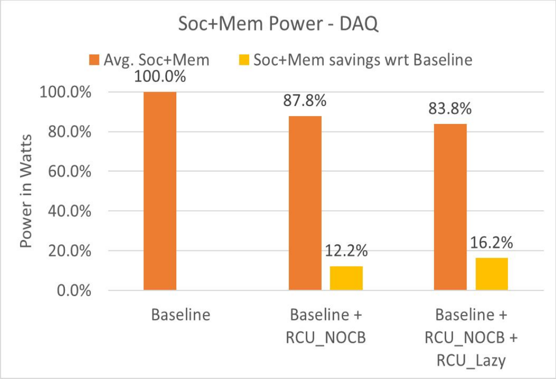
Issue 2: RCU queuing CBs on lightly loaded system

Solution 2: Delay RCU CB processing (Lazy RCU)

RCU lazy further reduces 300+ wakes per seconds, and improves SoC package C-states residency & Power







<u>Use-case</u>: Local video playback via Chrome browser, VP9 1080p @ 30 fps content

<u>Device</u>: Chrome reference device, AlderLake Hybrid CPU with 2 Cores (with Hyperthreading) + 8 Atoms

Issue 2: RCU queuing CBs on lightly loaded system

Solution 2: Delay RCU CB processing (Lazy RCU)

`rcutop` confirms callbacks are getting queued but not executed.

Callback	Queued	Executed
avc_node_free	41	0
k_itimer_rcu_free	5	0
thread_stack_free_rcu	23	0
file_free_rcu	576	0
delayed_put_pid	44	0
radix_tree_node_rcu_free	17	0
i_callback	55	0
__d_free	55	0
dst_destroy_rcu	2	0
epi_rcu_free	7	0
delayed_put_task_struct	44	0
inode_free_by_rcu	94	0

Drawbacks and considerations

- Depends on user of call_rcu() using lazy
  - If a new user of call_rcu() shows up, it would go unnoticed and negate the benefits.
- Risk of user using call_rcu_lazy() in a synchronous use case accidentally.
- Risks on memory pressure:
  - Protection is enough on extreme condition?
- Helping users choose the right API variant
  - Updates to docs may help: https://docs.kernel.org/RCU/whatisRCU.html#id11