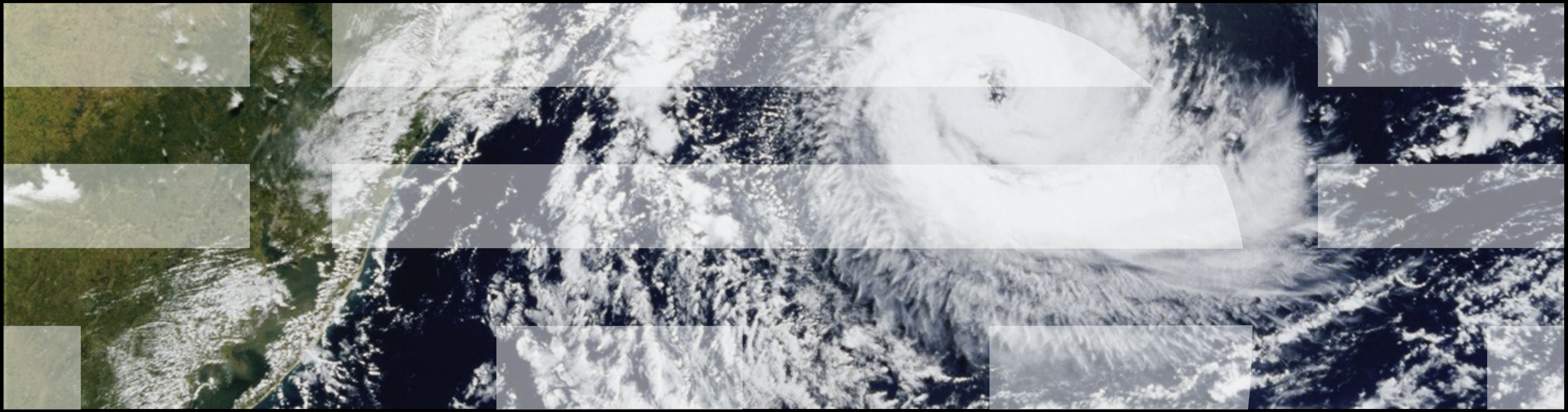
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center

linux.conf.au January 30, 2013

Making RCU Respect Your Device's Battery Lifetime

On-The-Job Energy-Efficiency Training For RCU Maintainers

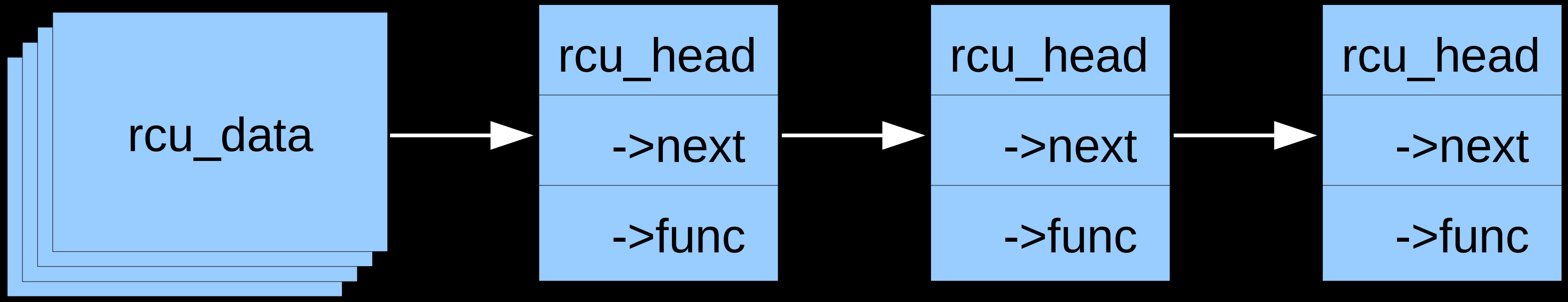


Overview

- What is RCU?
- “The Good Old Days”
- Overview of RCU's many variants of energy efficiency
- Current state of RCU energy efficiency
- Future directions

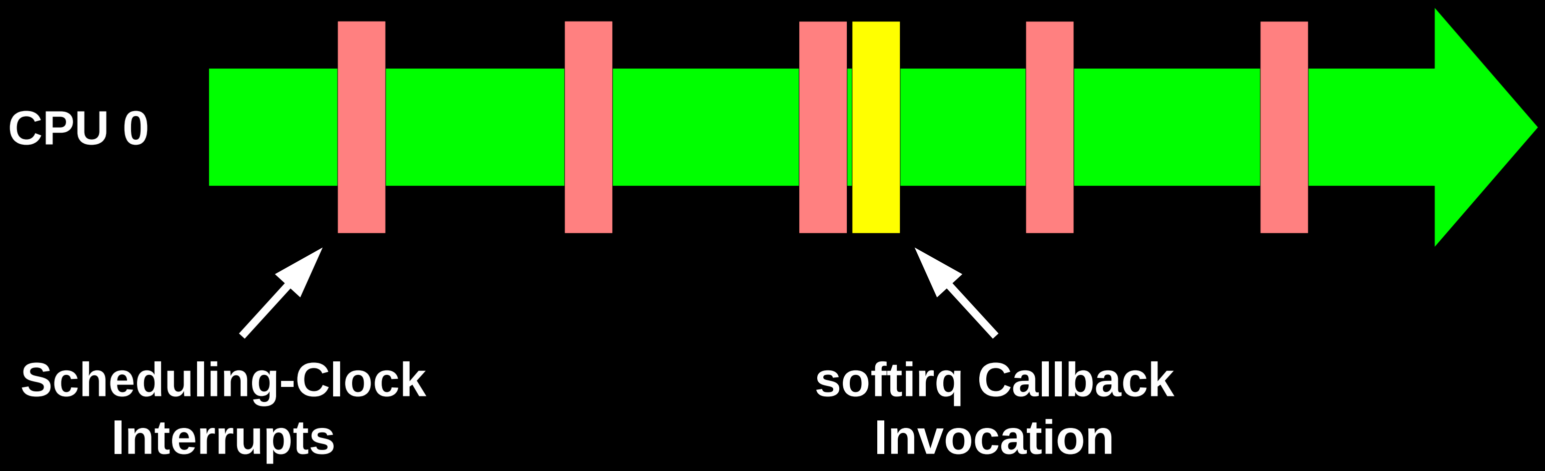
What is RCU?

- For an overview, see http://lwn.net/Articles/262464/
- For the purposes of this presentation, think of RCU as something that defers work, with one work item per callback
  - Each callback has a function pointer and an argument
  - Callbacks are queued on per-CPU lists, invoked after grace period
    - Invocation can result in OS jitter and real-time latency
  - Deferring the work a bit longer than needed is OK, deferring too long is bad – but failing to defer long enough is fatal

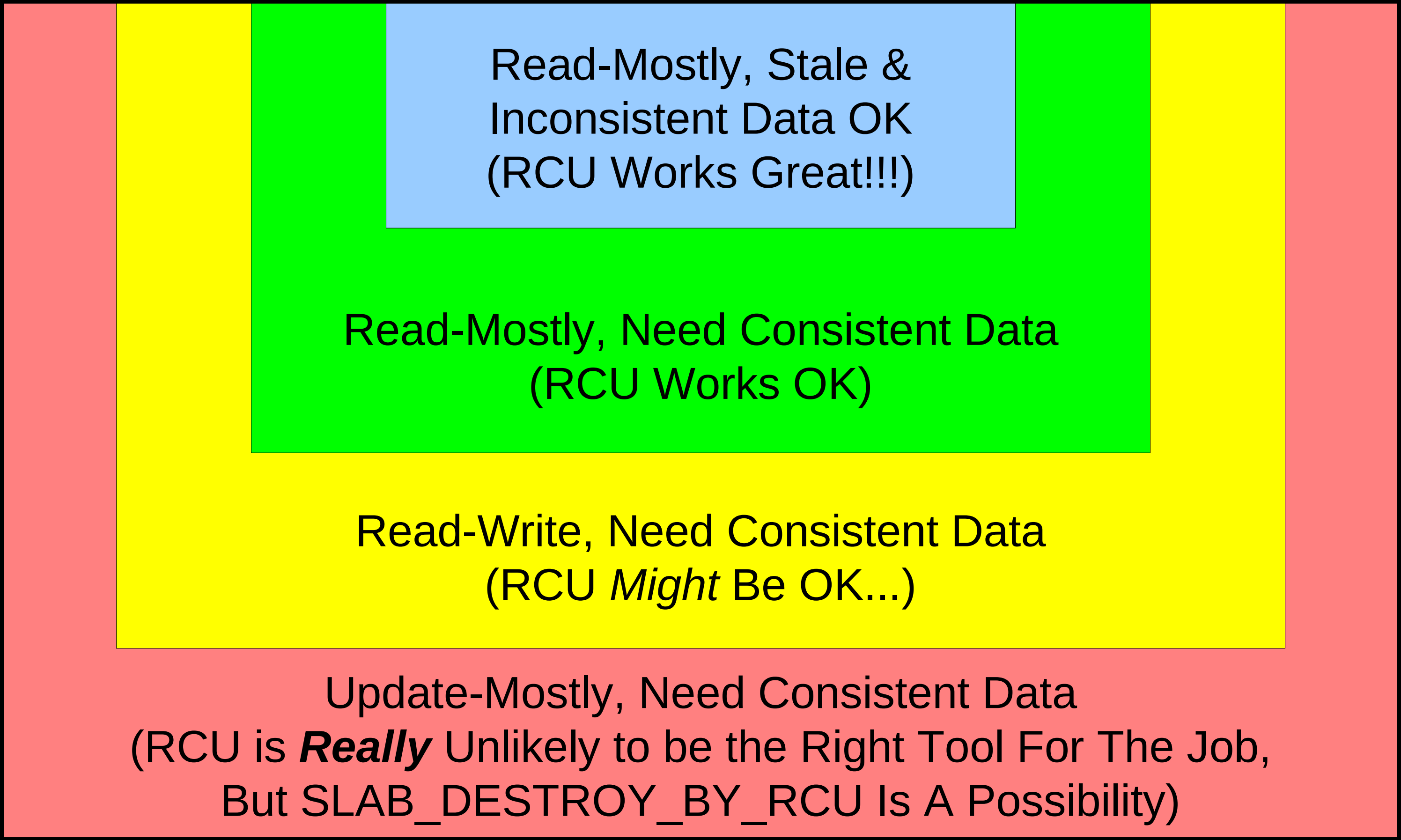


What is RCU?

- RCU uses a state machine driven out of the scheduling-clock interrupt to determine when it is safe to invoke callbacks
- Actual callback invocation is done from softirq



RCU Area of Applicability



Read-Mostly, Stale &
Inconsistent Data OK
(RCU Works Great!!!)

Read-Mostly, Need Consistent Data
(RCU Works OK)

Read-Write, Need Consistent Data
(RCU *Might* Be OK...)

Update-Mostly, Need Consistent Data
(RCU is ***Really*** Unlikely to be the Right Tool For The Job,
But SLAB_DESTROY_BY_RCU Is A Possibility)

Use the right tool for the job!!!

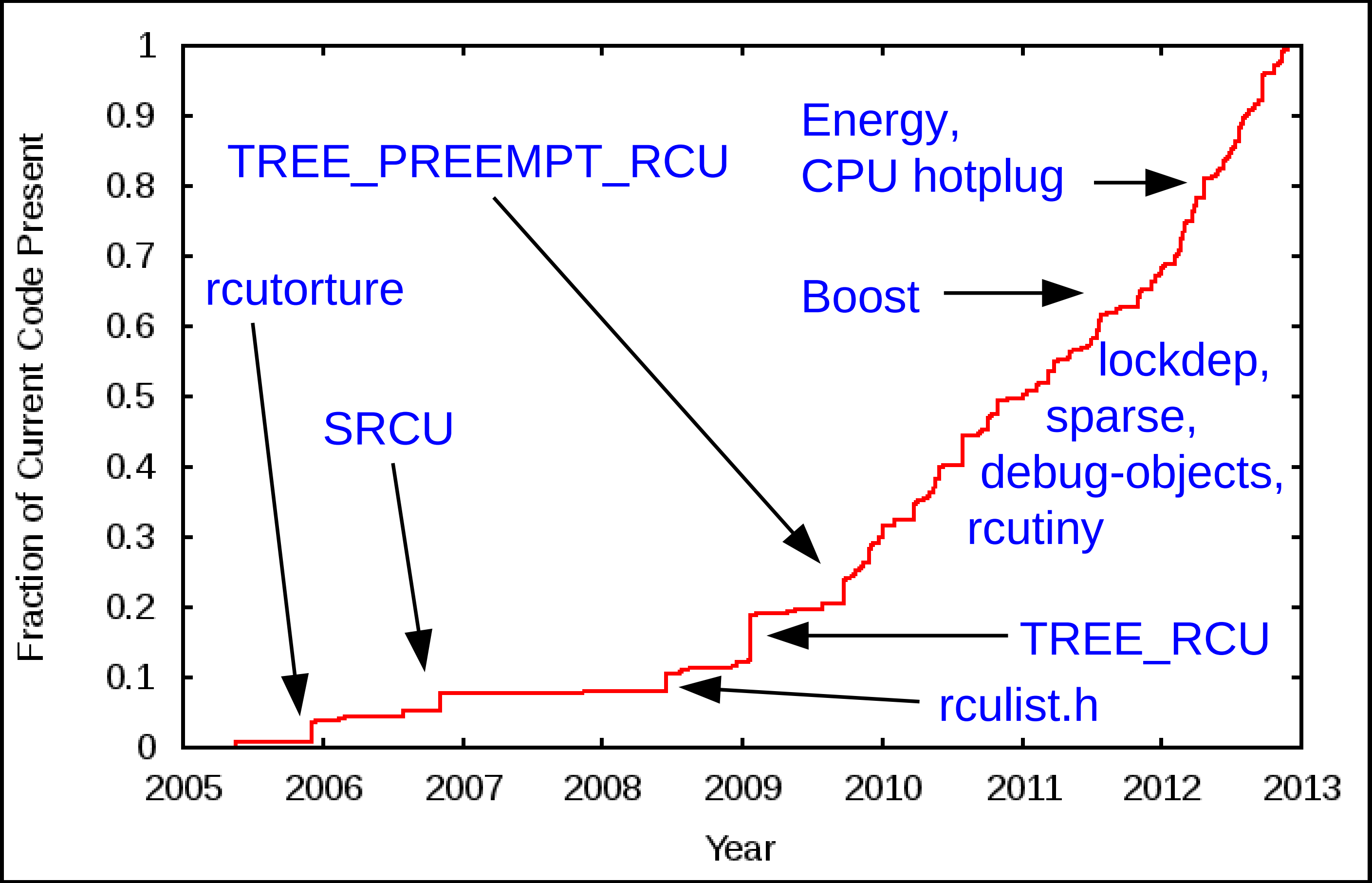
For More Information on RCU...

- Documentation/RCU in the Linux® kernel source code
- “User-Level Implementations of Read-Copy Update” (Mathieu Desnoyers et al.)
  - http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.159
- “The RCU API, 2010 Edition”
  - http://lwn.net/Articles/418853/
- “What is RCU” LWN series
  - http://lwn.net/Articles/262464/ (What is RCU, Fundamentally?)
  - http://lwn.net/Articles/263130/ (What is RCU's Usage?)
  - http://lwn.net/Articles/264090/ (What is RCU's API?)
- “Introducing technology into the Linux kernel: a case study”
  - http://doi.acm.org/10.1145/1400097.1400099
- “Meet the Lockers” (Neil Brown)
  - http://lwn.net/Articles/453685/
- “Read-Copy Update” (2001 OLS paper, still used in a number of college courses)
  - http://www.linuxsymposium.org/2001/abstracts/readcopy.php
- Plus more at: http://www.rdrop.com/users/paulmck/RCU

RCU:
Tapping The Awesome Power of Procrastination For Two Decades!!!

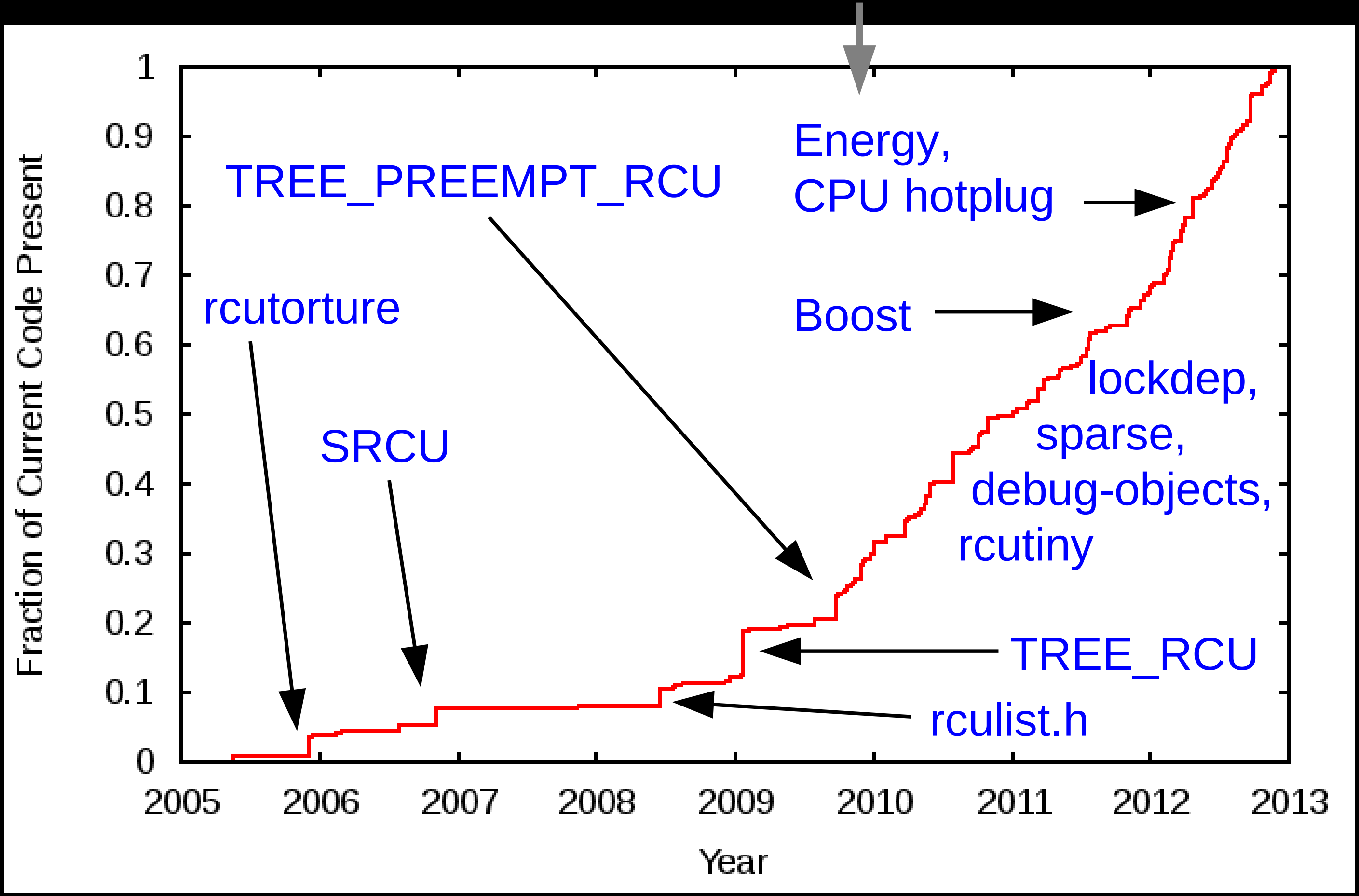
“The Good Old Days”

Not Much “Good Old Days” Code Left in RCU



Not Much “Good Old Days” Code Left in RCU

Why did I wait so long to conserve energy???



Why Did I Wait Until 2011 to Conserve Energy?

- The fact is that I didn't wait that long!!!
- But RCU's energy-efficiency code is unusual in that it has been rewritten a great many times
  - RCU has been concerned about energy efficiency for about ten years
  - Not much energy-efficiency code in RCU in the 1990s: Why?
- Other minor changes:
  - Expedited grace periods
  - Additions to rcutorture
  - Additional list-traversal primitives
  - Upgrading real-time response
  - Plus the usual list of fixes, improvements, and adaptations

“The Good *Really* Old Days”

- RCU used by DYNIX/ptx: Heavy database servers
- Used for a number of applications:
  - Fraud detection in large financial systems
  - Inventory monitoring/control for large retail firms
  - Rental car tracking/billing
  - Manufacturing coordination/control
    - Including manufacturing of airliners

Airliner Manufacturing Plants Had Lots of These:

Author: William M. Plate Jr. (Public Domain)

Airliner Manufacturing Plants Had Lots of These

At About 40KW Each

Author: William M. Plate Jr. (Public Domain)

And if You Think That *Welders* Are Power-Hungry...

GE90-115B turbofan - front {{Le Bourget 2005}} Copyright © 2005 David Monniaux {{GFDL}} {{cc-by-sa-2.0}} {{cc-by-sa-2.0-fr}}

If You Are Running a Bunch of Welders or Turbines...

- Not only are you not going to care much about RCU's contribution to power consumption...

If You Are Running a Bunch of Welders or Turbines...

- Not only are you not going to care much about RCU's contribution to power consumption...
- You are not going to care much about the whole server's contribution to power consumption!
- But of course things look very different for small battery-powered devices...

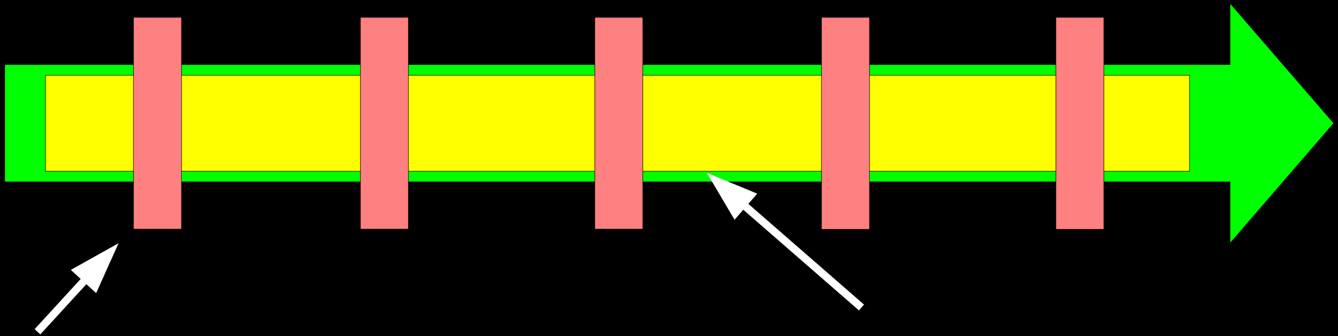
RCU's Many Energy-Efficiency Implementations

Initial RCU Did Have One Energy-Efficiency Feature

- Initial DYNIX/ptx RCU had light-weight read-side primitives
  - “Free” is a *very* good price!!!
- This meant that the system returned to idle more quickly than it would with heavier-weight synchronization primitives
  - But 1990s systems consumed more power idle than when running!
  - This was because the idle loop fit into cache, thus allowing the CPU to execute useless instructions at a very high rate
- But today's CPUs have many energy-efficiency features
  - And have very low idle power, especially for long-duration idle periods
- So why does RCU need to worry about energy efficiency???
  - After all, it is just a synchronization primitive!!!

RCU Driven From Scheduling Clock Interrupt

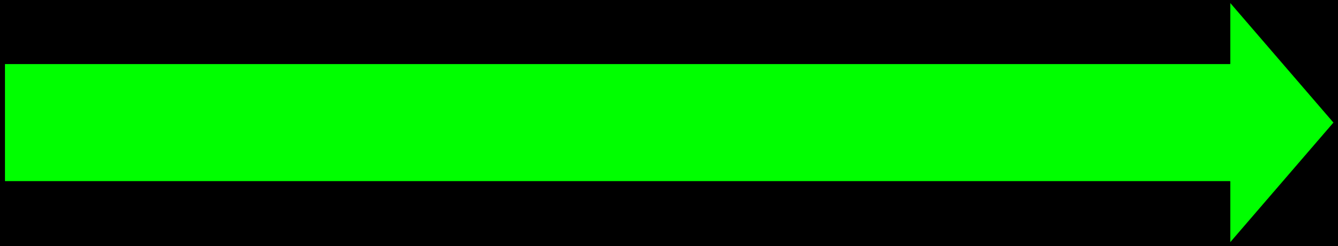
What RCU Did (2003)



Scheduling-Clock Interrupts

RCU's Use of Scheduling-Clock Interrupts Wastes Power and Prevents Deep CPU Sleep States

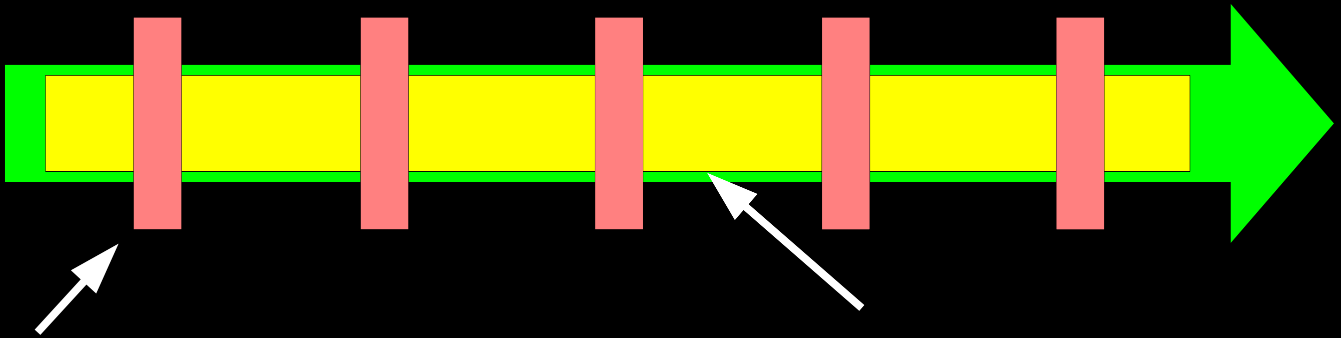
What Is Required



No Scheduling-Clock Interrupts, CPU Enters Deep Sleep

RCU Driven From Scheduling Clock Interrupt

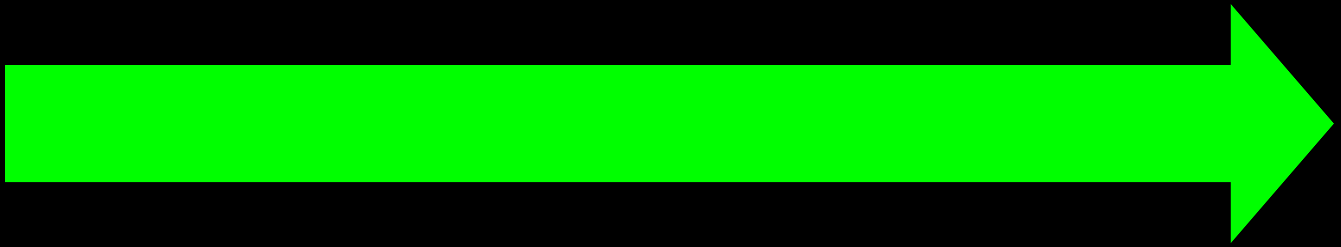
What RCU Did (2003)



Scheduling-Clock Interrupts

RCU's Use of Scheduling-Clock Interrupts Wastes Power and Prevents Deep CPU Sleep States

What Is Required



No Scheduling-Clock Interrupts, CPU Enters Deep Sleep

Which is why RCU has a dyntick-idle subsystem!

RCU and Dyntick Idle (AKA CONFIG_NO_HZ=y)

- List of implementations:
  - 2004: Dyntick-idle bit vector
    - Manfred Spraul locates theoretical bug

RCU and Dyntick Idle (AKA CONFIG_NO_HZ=y)

- List of implementations:
  - 2004: Dyntick-idle bit vector
    - Manfred Spraul locates theoretical bug
    - A few months before the mainframe guys encounter it

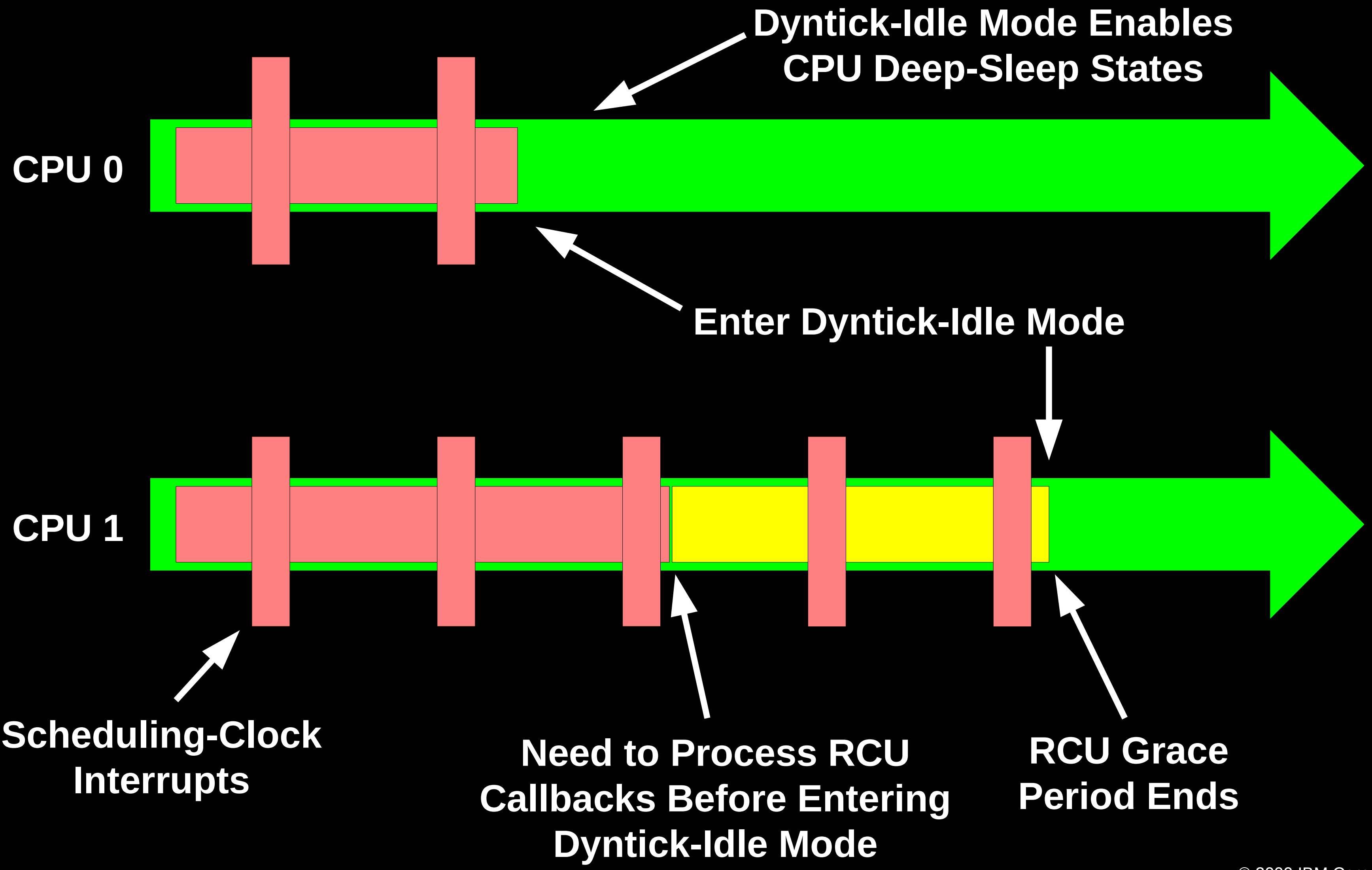
RCU and Dyntick Idle (AKA CONFIG_NO_HZ=y)

- List of implementations:
  - 2004: Dyntick-idle bit vector
    - Manfred Spraul locates theoretical bug
    - A few months before the mainframe guys encounter it
    - But after it had been in-tree for *four years*

RCU and Dyntick Idle (AKA CONFIG_NO_HZ=y)

- List of implementations:
  - 2004: Dyntick-idle bit vector
    - Manfred Spraul locates theoretical bug
    - A few months before the mainframe guys encounter it
    - But after it has been in-tree for *four years*
  - 2008: -rt version (with Steven Rostedt)
    - Very complex: http://lwn.net/Articles/279077/
  - 2009: Separate out NMI accounting
    - Greatly simplified: No proof of correctness required ;-)

RCU and Dyntick Idle as of Early 2010


Dyntick-Idle Mode Enables CPU Deep-Sleep States
CPU 0
Enter Dyntick-Idle Mode
CPU 1
Scheduling-Clock Interrupts
Need to Process RCU Callbacks Before Entering Dyntick-Idle Mode
RCU Grace Period Ends

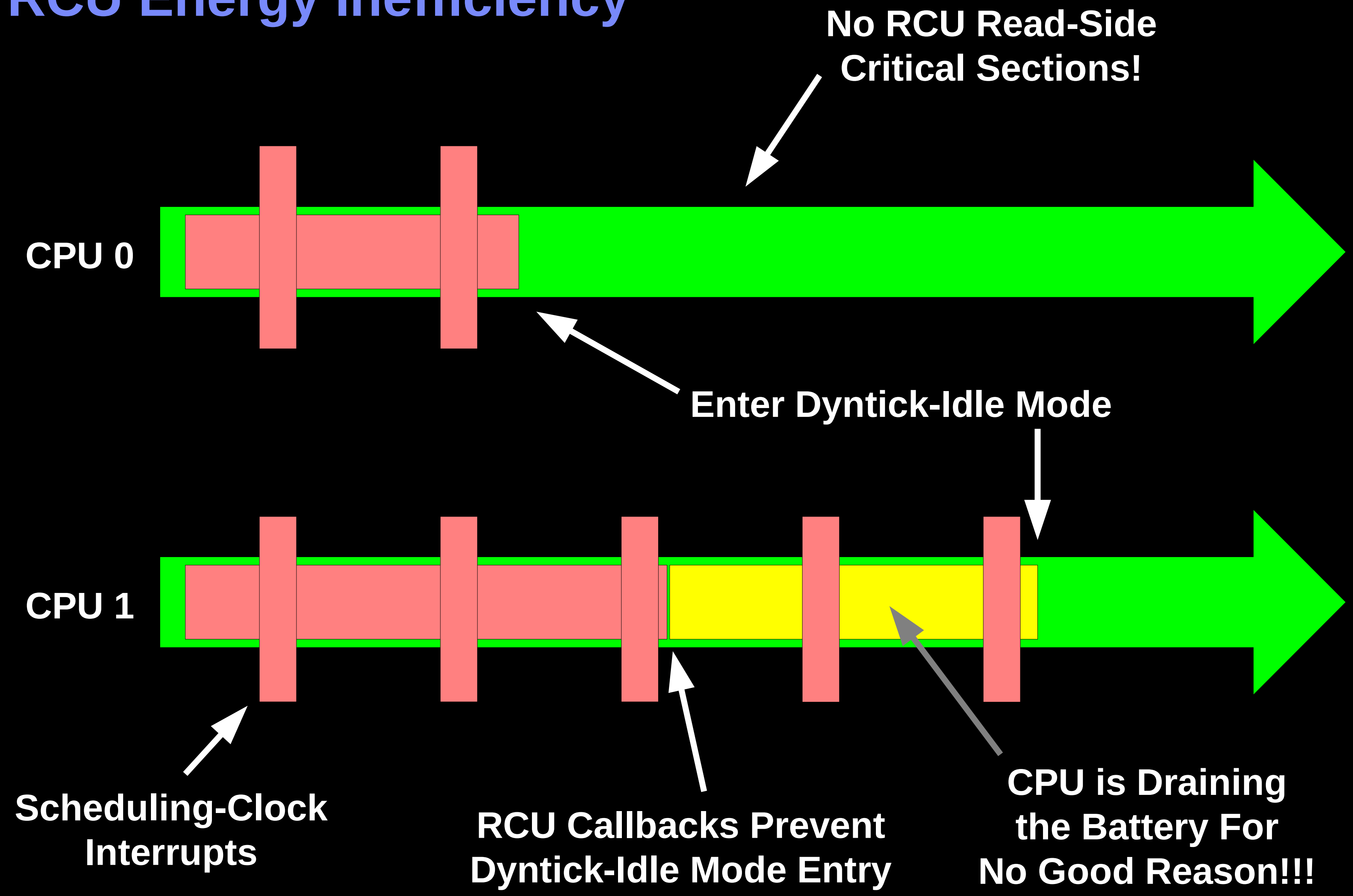
So RCU is Perfectly Energy Efficient, Right?

So RCU is Perfectly Energy Efficient, Right?

- Well, I thought that RCU was *very* energy efficient
- Then in early 2010 I got a call from someone working on a battery-powered multicore system
- And he was ***very*** upset with RCU

- Why?

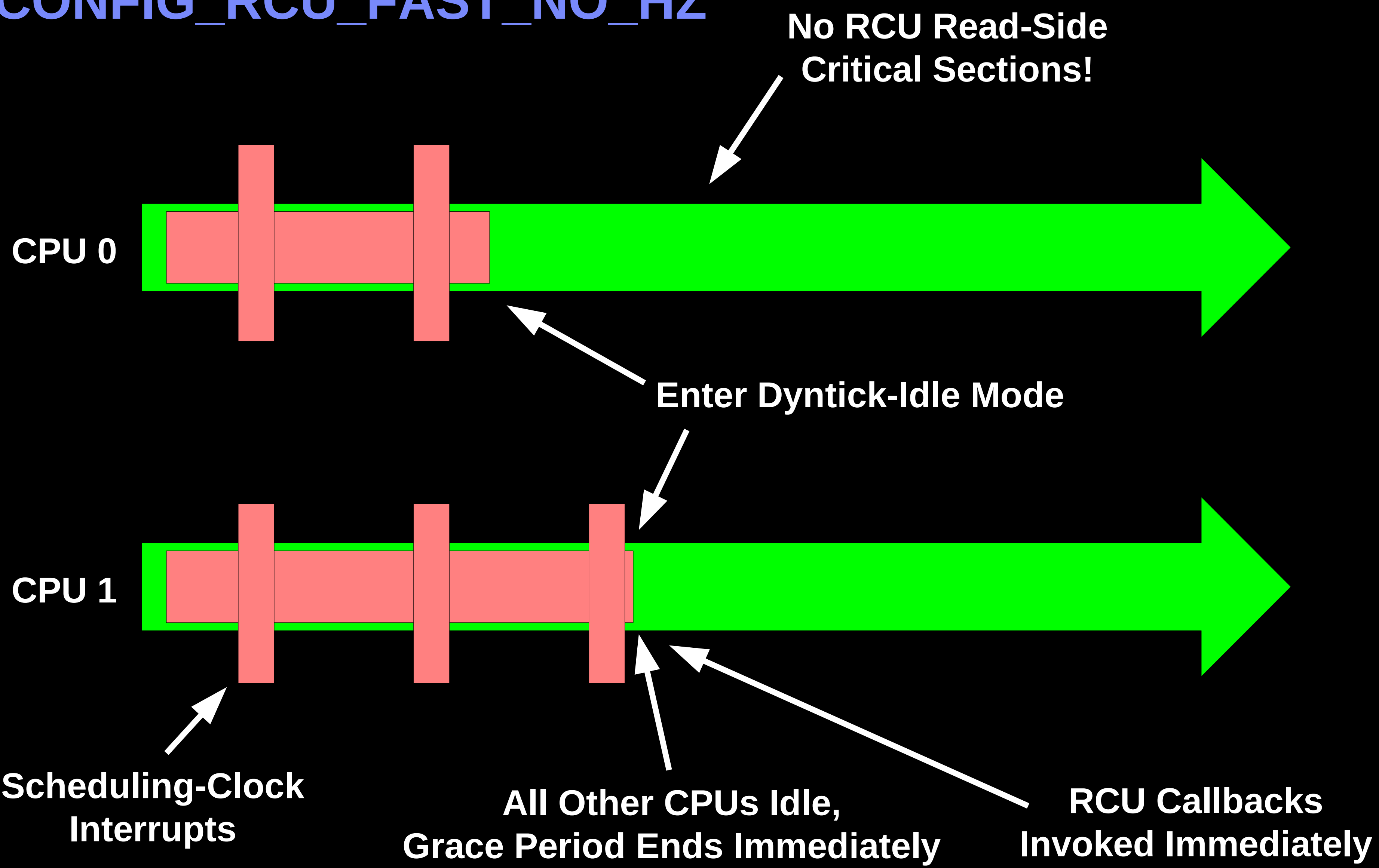
RCU Energy Inefficiency



RCU and Dyntick Idle (AKA CONFIG_NO_HZ=y)

- List of implementations:
  - 2004: Dyntick-idle bit vector
    - Manfred Spraul locates theoretical bug
    - A few months before the mainframe guys encounter it
    - But after it has been in-tree for *four* years
  - 2008: -rt version (with Steven Rostedt)
    - Very complex: http://lwn.net/Articles/279077/
  - 2009: Separate out NMI accounting
    - Greatly simplified: No proof of correctness required
  - 2010: CONFIG_RCU_FAST_NO_HZ for small systems
    - Force last CPU into dyntick-idle mode

CONFIG_RCU_FAST_NO_HZ



No RCU Read-Side Critical Sections!

Enter Dyntick-Idle Mode

Scheduling-Clock Interrupts

All Other CPUs Idle, Grace Period Ends Immediately

RCU Callbacks Invoked Immediately

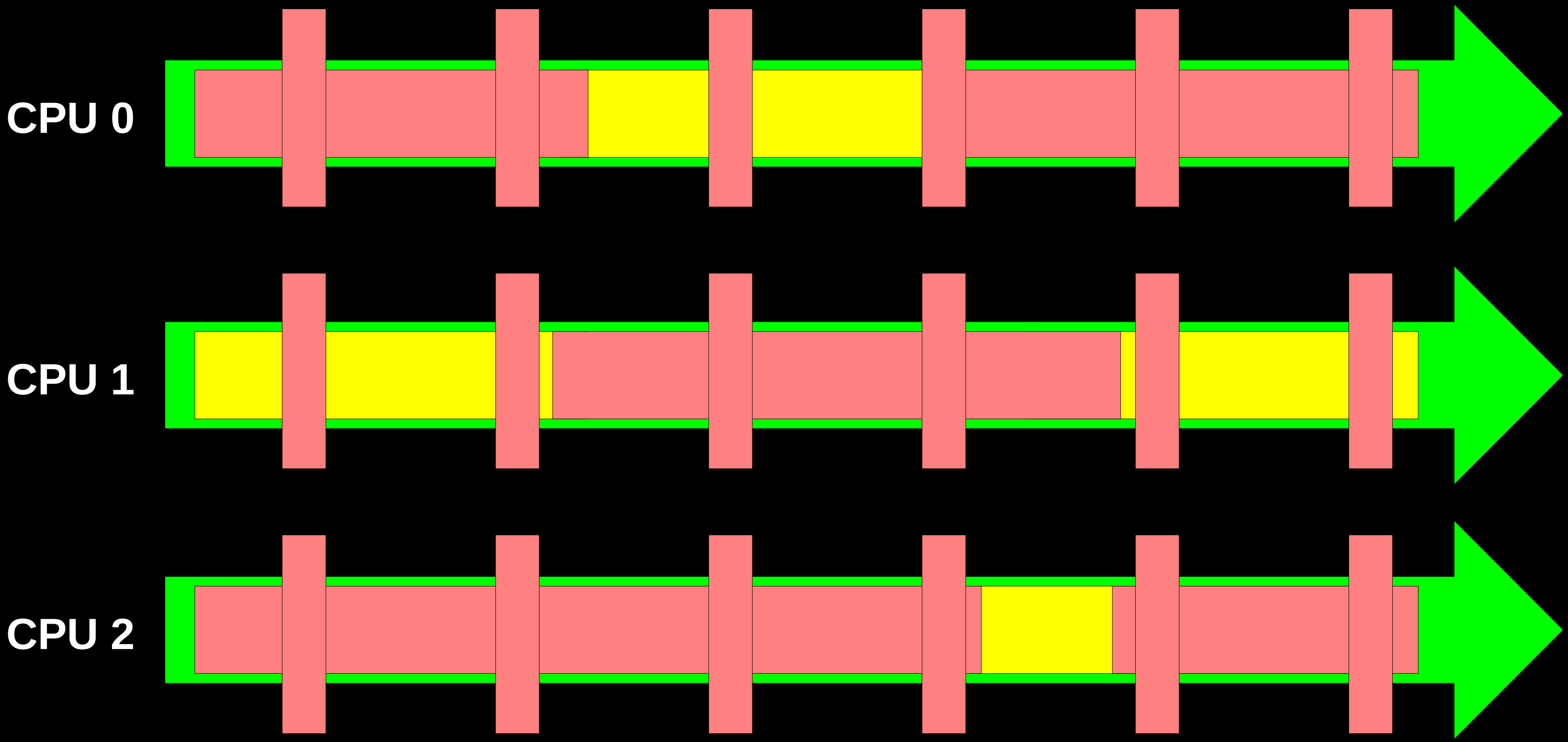
So RCU is Perfectly Energy Efficient, Right?

So RCU is Perfectly Energy Efficient, Right?

- This time, I was wiser:
  - I suspected CONFIG_FAST_NO_HZ needed on large systems
- And someone mentioned this to me in late 2011
- But some things never change: He was ***very*** upset with RCU

- Why?

Might *Never* Have All But One CPU Dyntick-Idled!!!

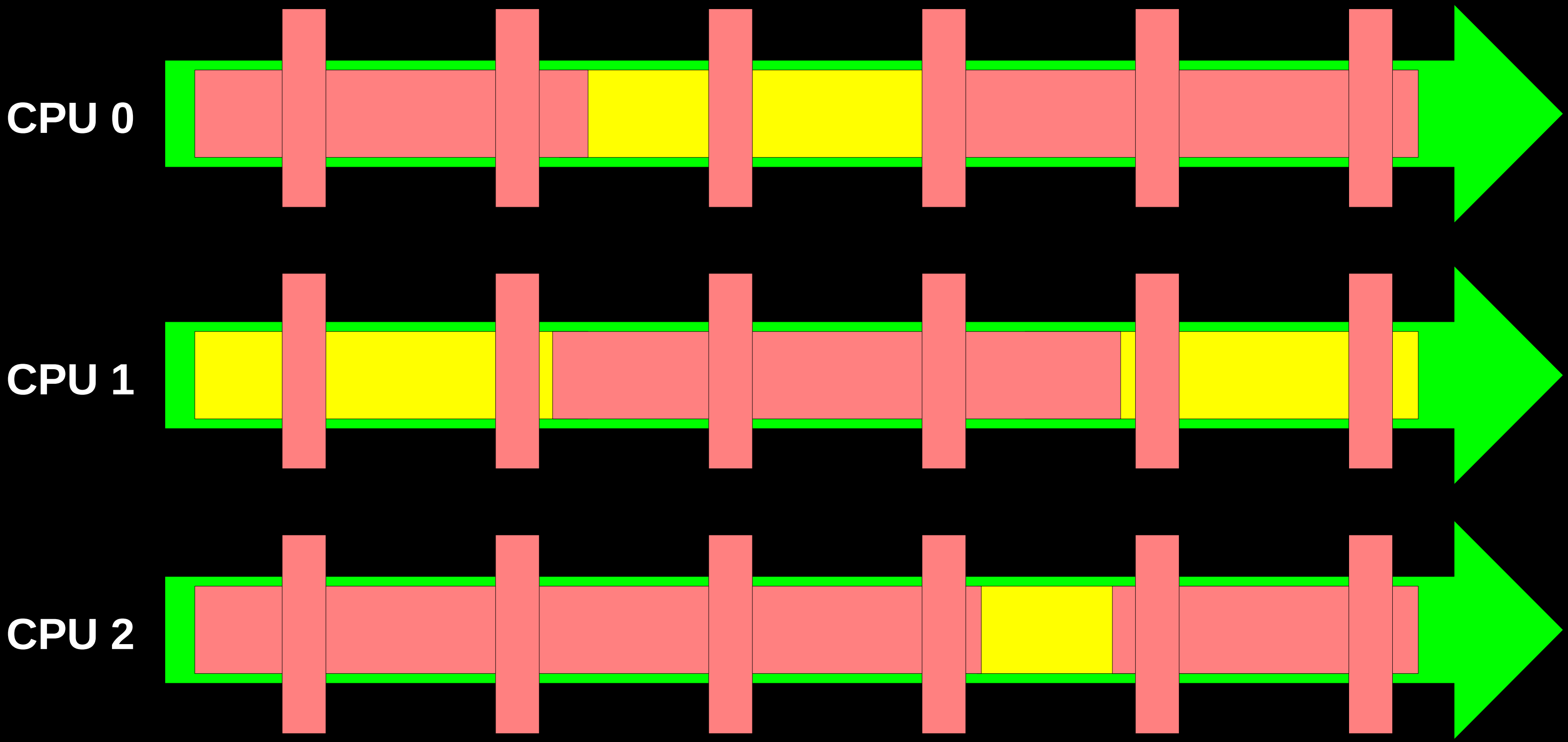


The more CPUs you have, the worse this effect gets

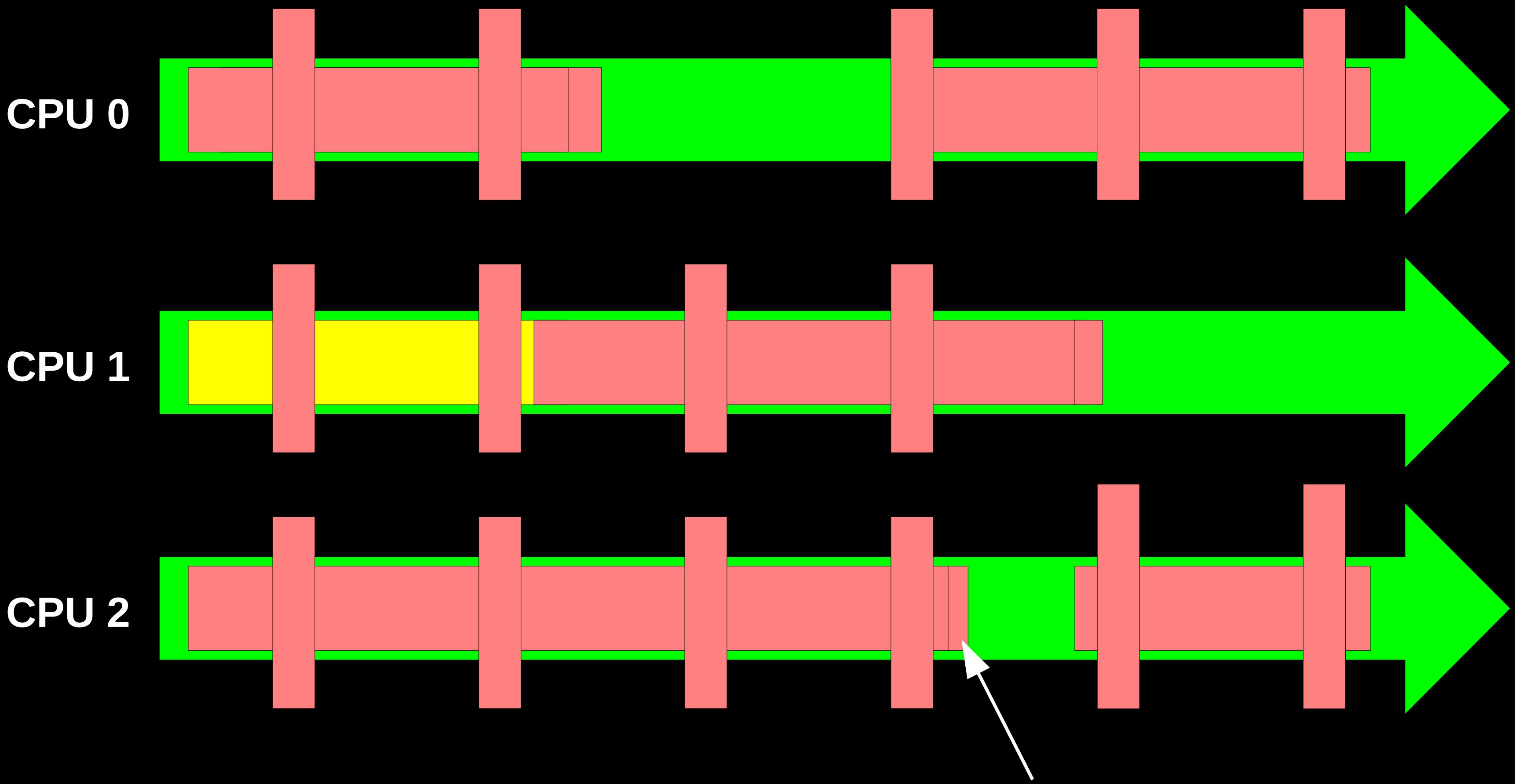
RCU and Dyntick Idle (AKA CONFIG_NO_HZ=y)

- List of implementations:
  - 2004: Dyntick-idle bit vector
    - Manfred Spraul locates theoretical bug
    - A few months before the mainframe guys encounter it
    - But after it has been in-tree for *four* years
  - 2008: -rt version (with Steven Rostedt)
    - Very complex: http://lwn.net/Articles/279077/
  - 2009: Separate out NMI accounting
    - Greatly simplified: No proof of correctness required
  - 2010: CONFIG_RCU_FAST_NO_HZ for small systems
    - Force last CPU into dyntick-idle mode
  - 2012: CONFIG_RCU_FAST_NO_HZ for large systems
    - Force CPUs with callbacks into dyntick-idle, but wake them up later
    - (See 2012 ELC presentation)

Large-System CONFIG_RCU_FAST_NO_HZ: Before



Large-System CONFIG_RCU_FAST_NO_HZ: After

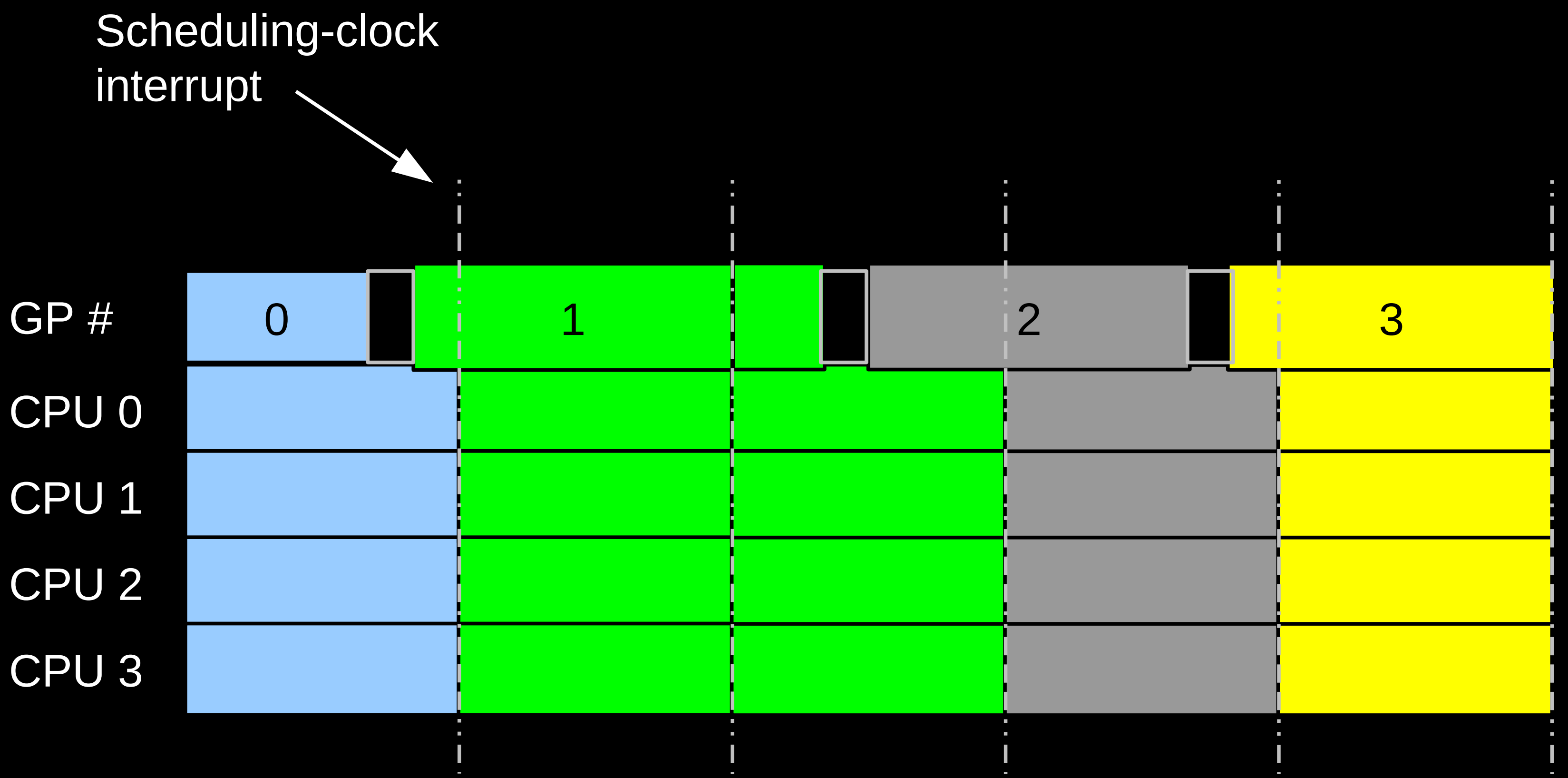


Extra work at idle entry might
(or might not) save work later

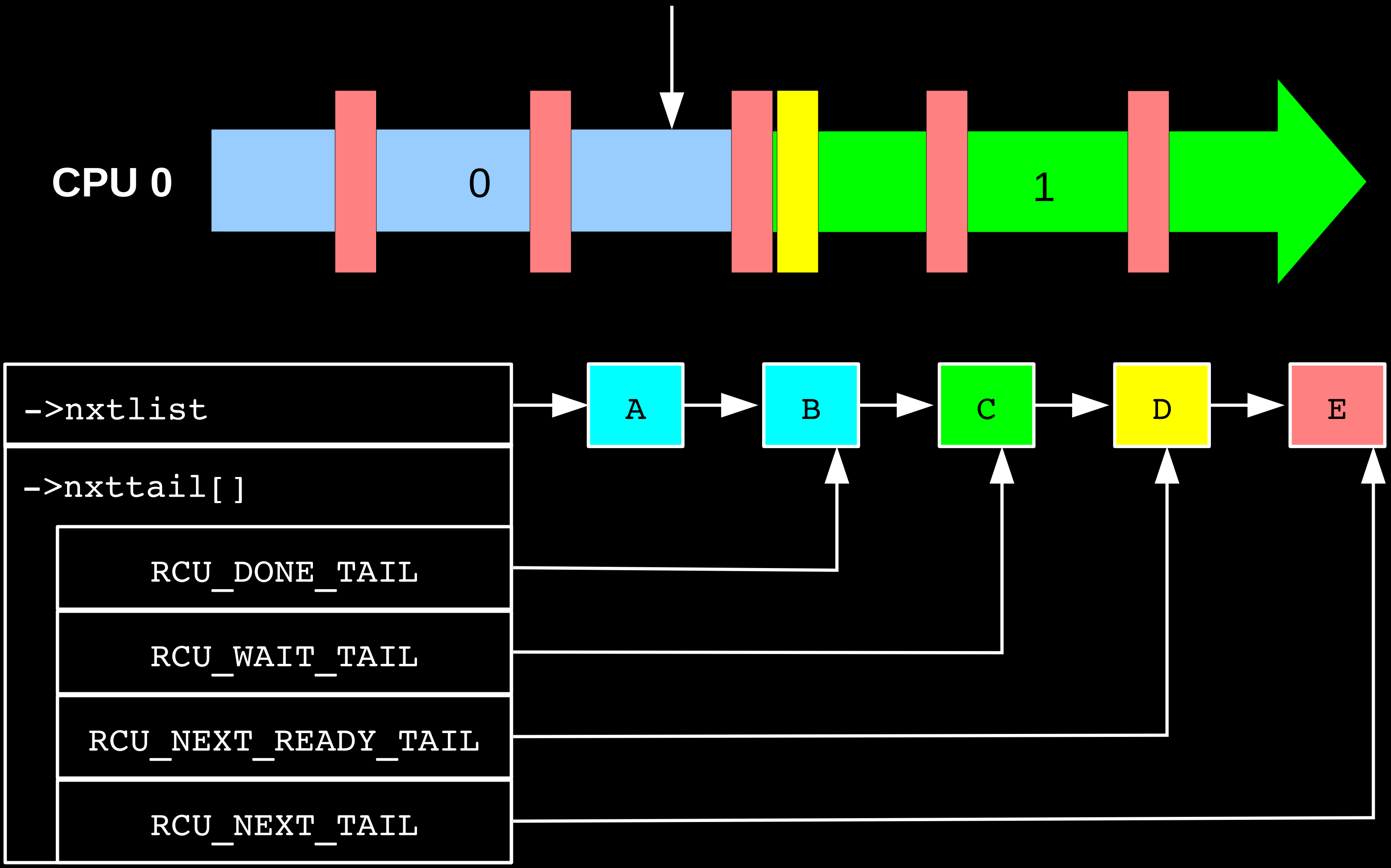
Large-System CONFIG_RCU_FAST_NO_HZ: Results

- Performance work showed equivocal results
- Often a great reduction in wakeups, but not always as large of energy savings as desired
- Repeated attempts to drain callbacks on idle entry do not seem to be as helpful as desired
- Can CONFIG_RCU_FAST_NO_HZ reduce scheduling-clock ticks with less idle-entry RCU-callback work?
  - To find out, let's look at RCU grace-period and callback handling
  - Grace period: The period of time that RCU defers work

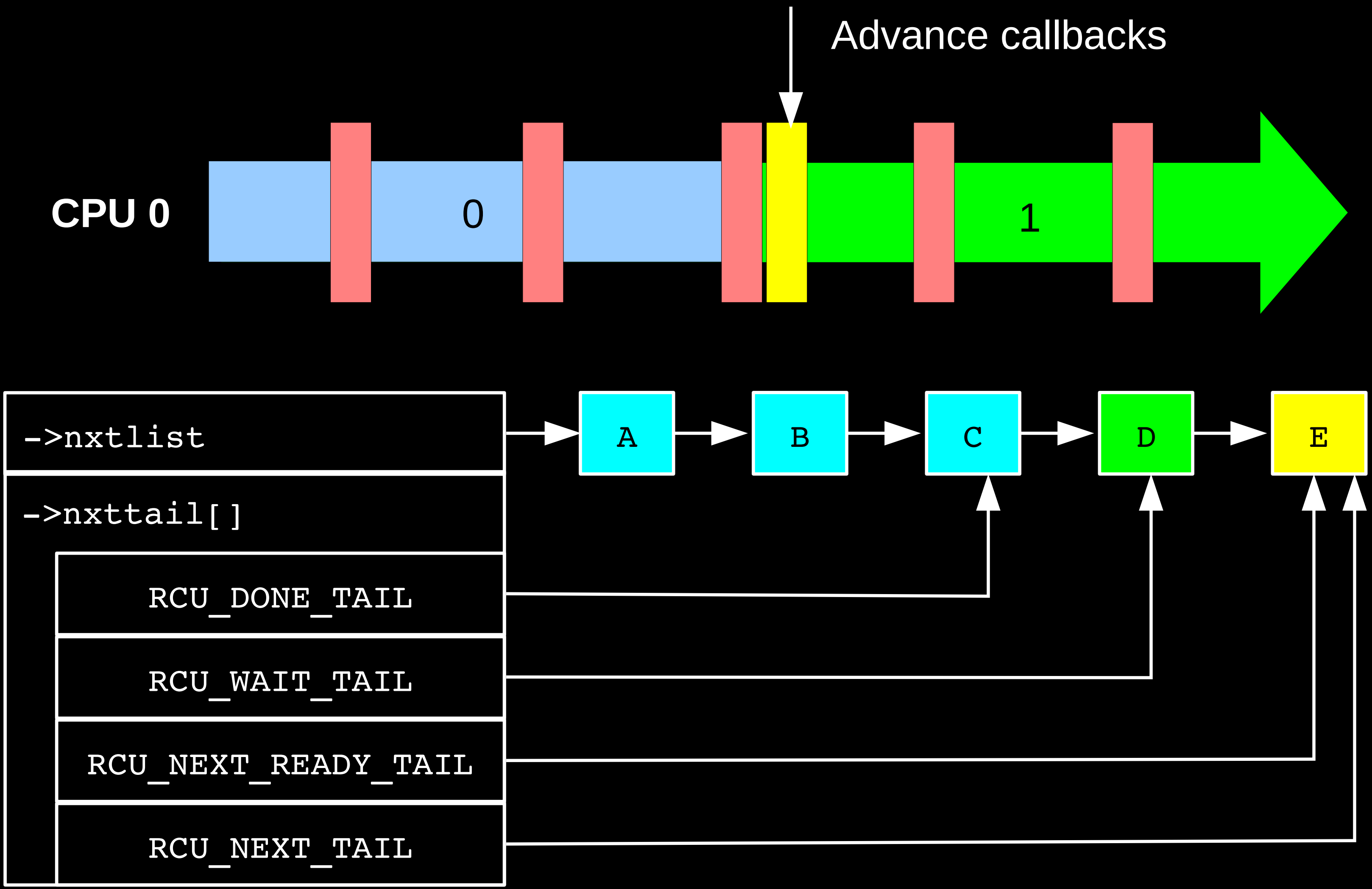
Grace-Period Handling In The Good Really Old Days



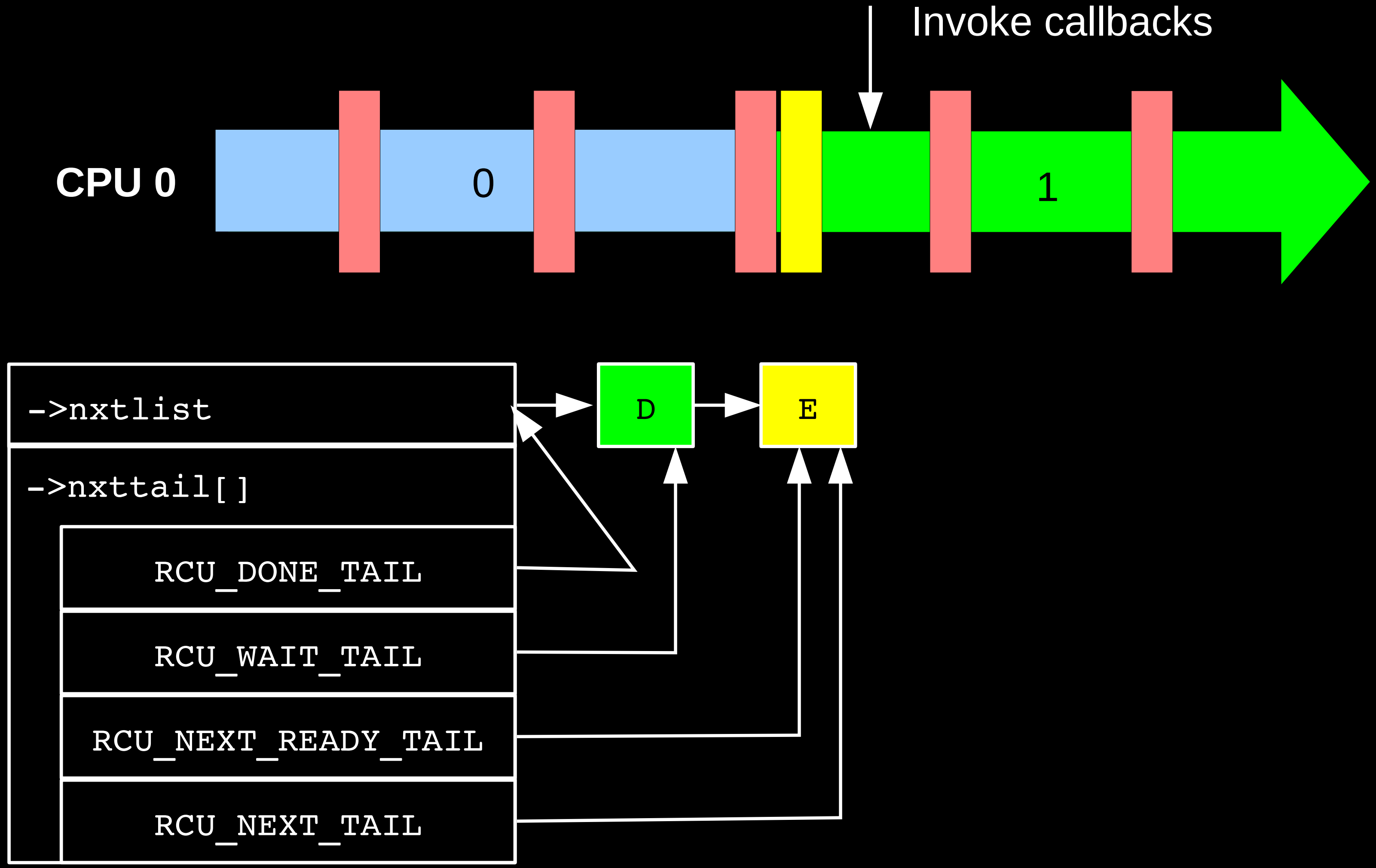
RCU Callback Handling In The Good Really Old Days



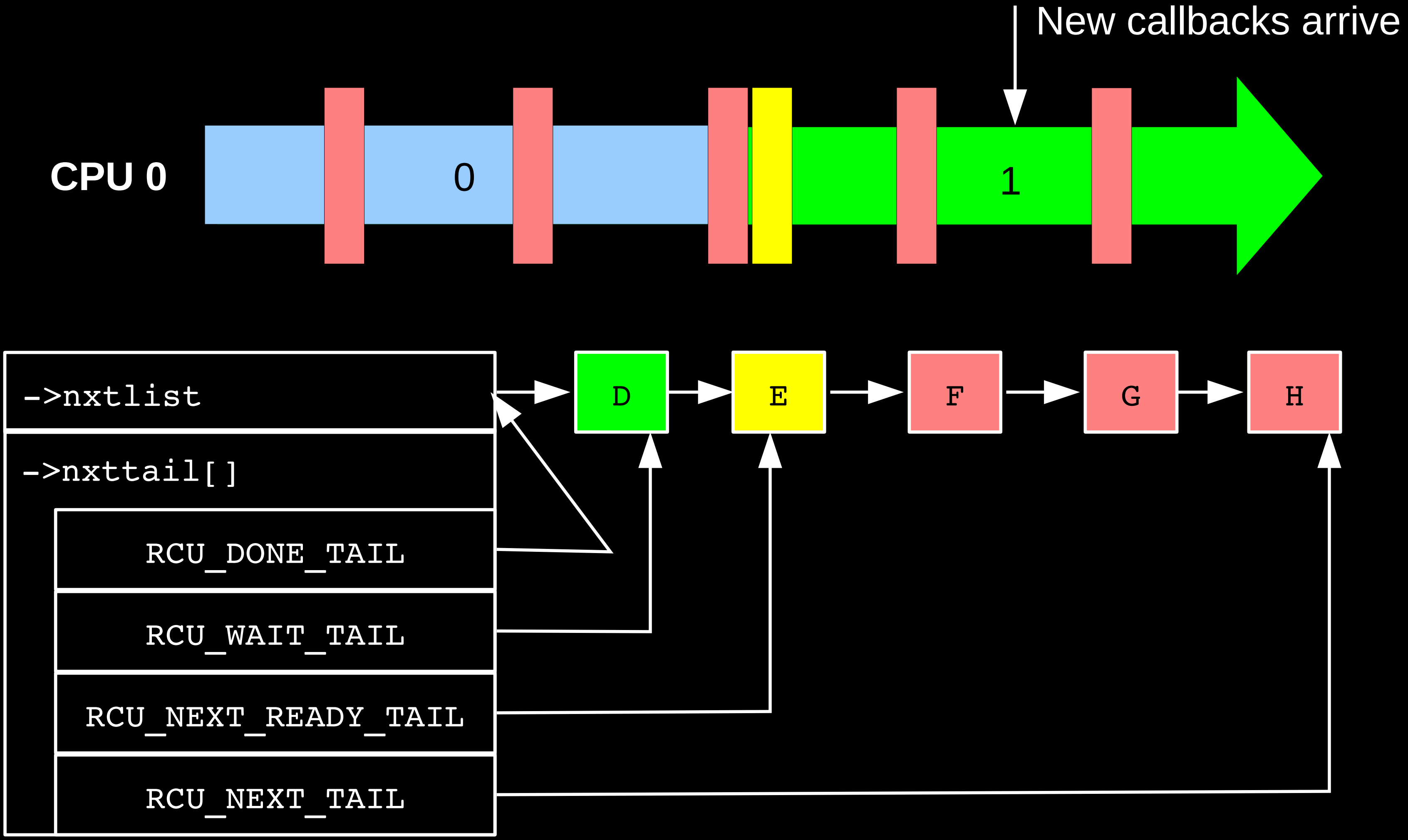
RCU Callback Handling In The Good Really Old Days



RCU Callback Handling In The Good Really Old Days



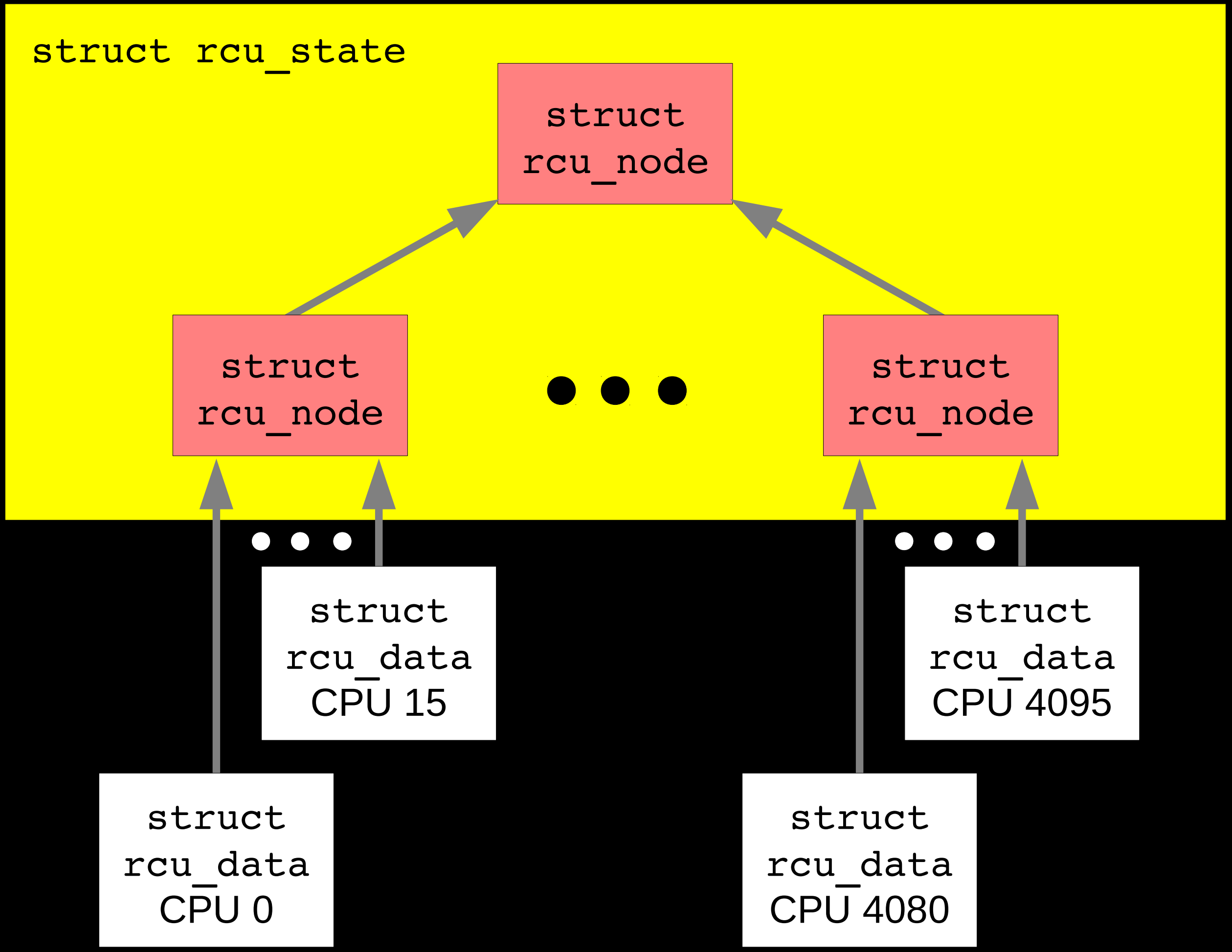
RCU Callback Handling In The Good Really Old Days



Grace-Period Handling And TREE_RCU

- Problem: Lock contention
- Solution: Apply hierarchy in the form of TREE_RCU

Grace-Period Handling And TREE_RCU: 4096 CPUs



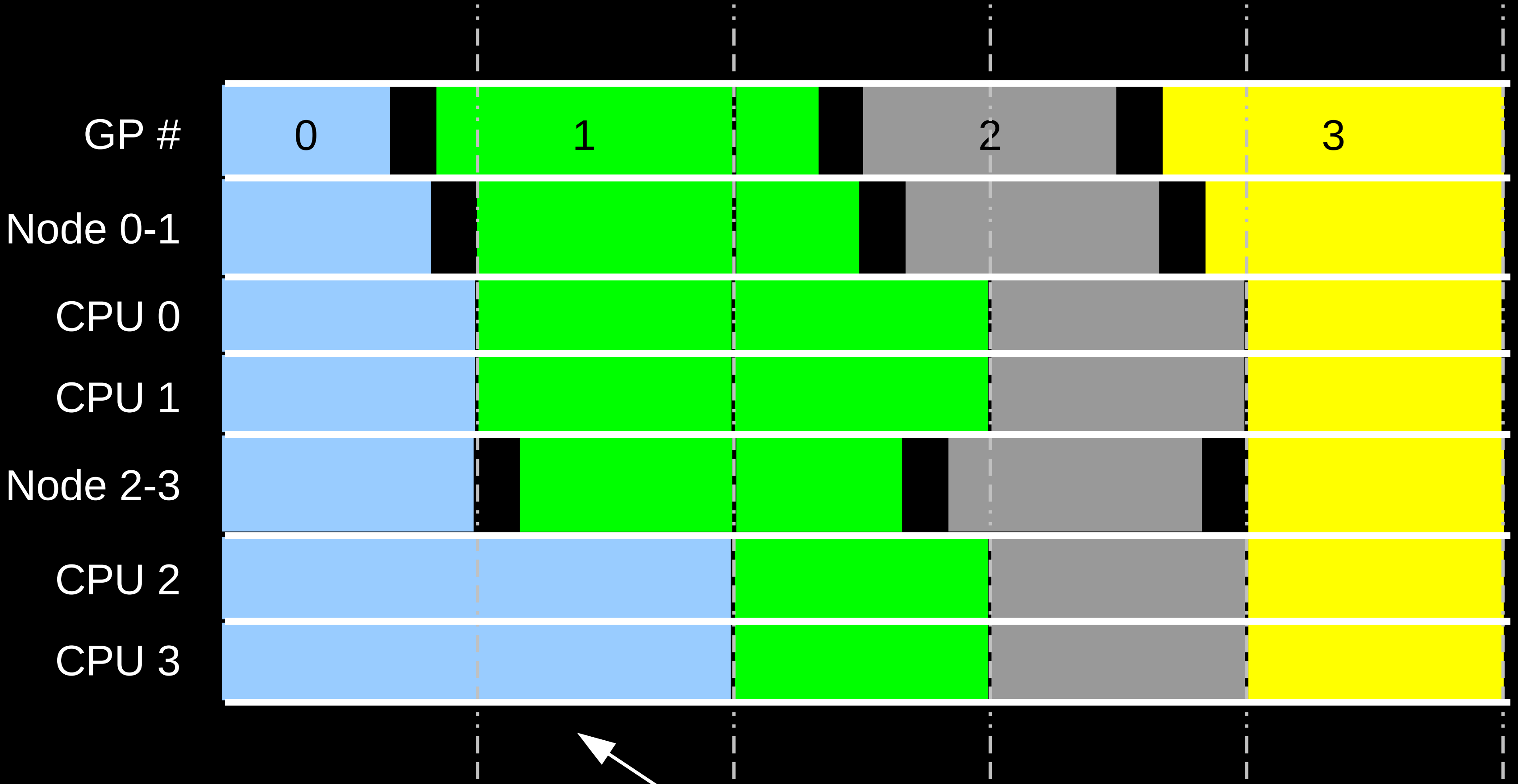
Level 0: 1 rcu_node

Level 1: 4 rcu_nodes

Level 2: 256 rcu_nodes

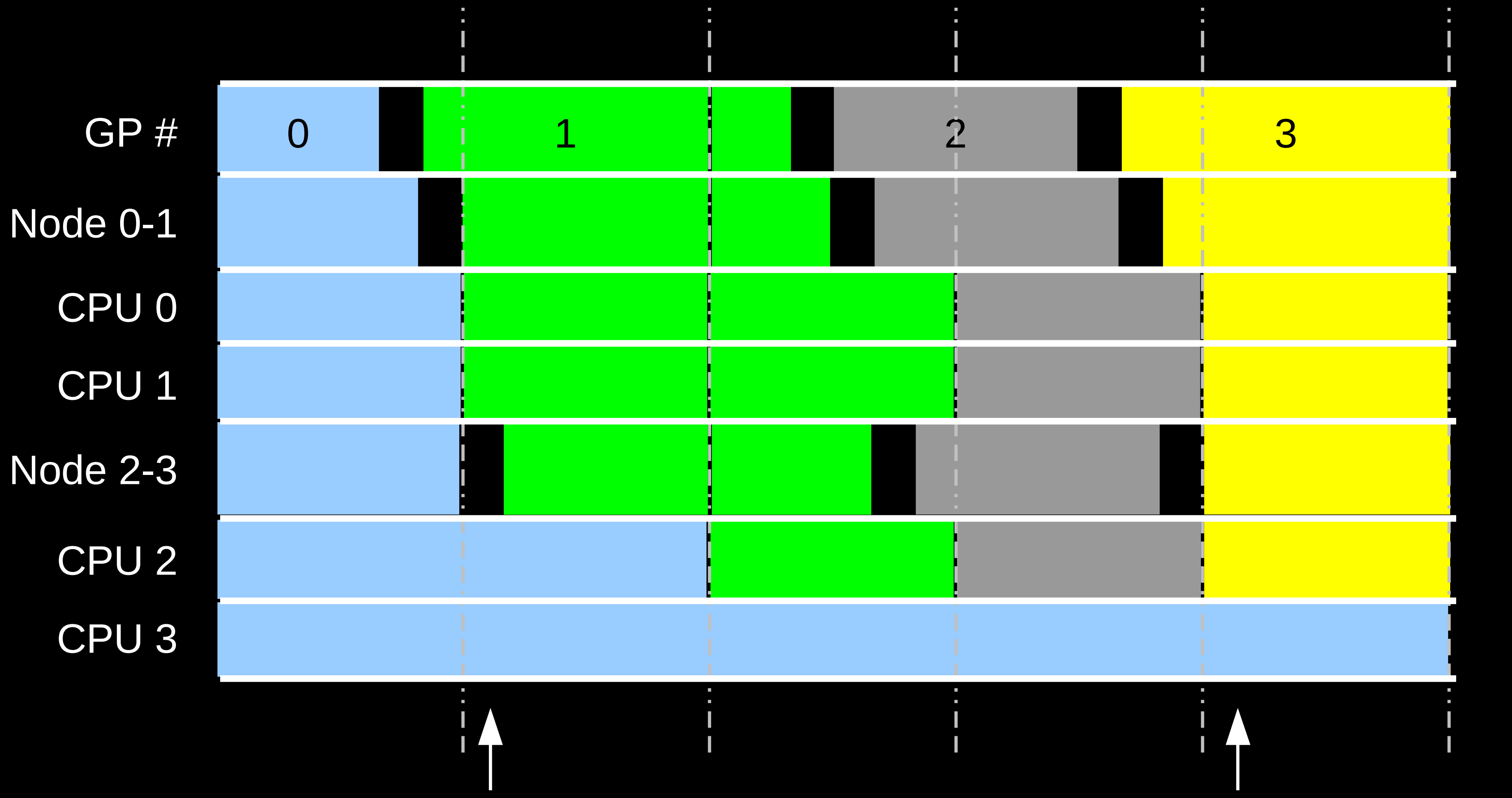
Total: 261 rcu_nodes

Grace-Period Handling And TREE_RCU: 4 CPUs



CPU 2 & 3 awareness
of race-period start delayed

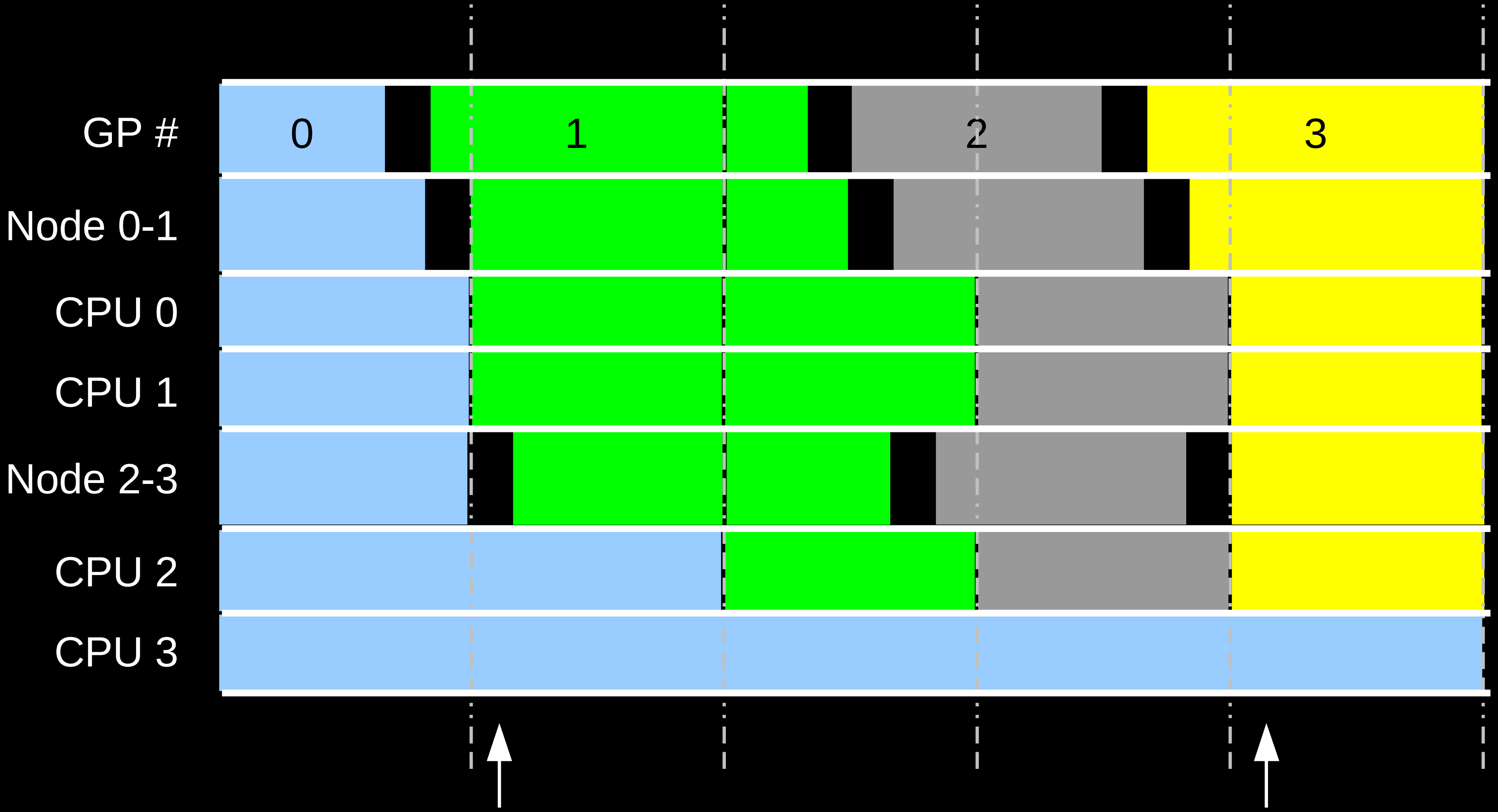
Grace-Period Handling, TREE_RCU, and dyntick-idle



Callbacks registered here ...

… are guaranteed done here

Grace-Period Handling, TREE_RCU, and dyntick-idle



Callbacks registered here ...

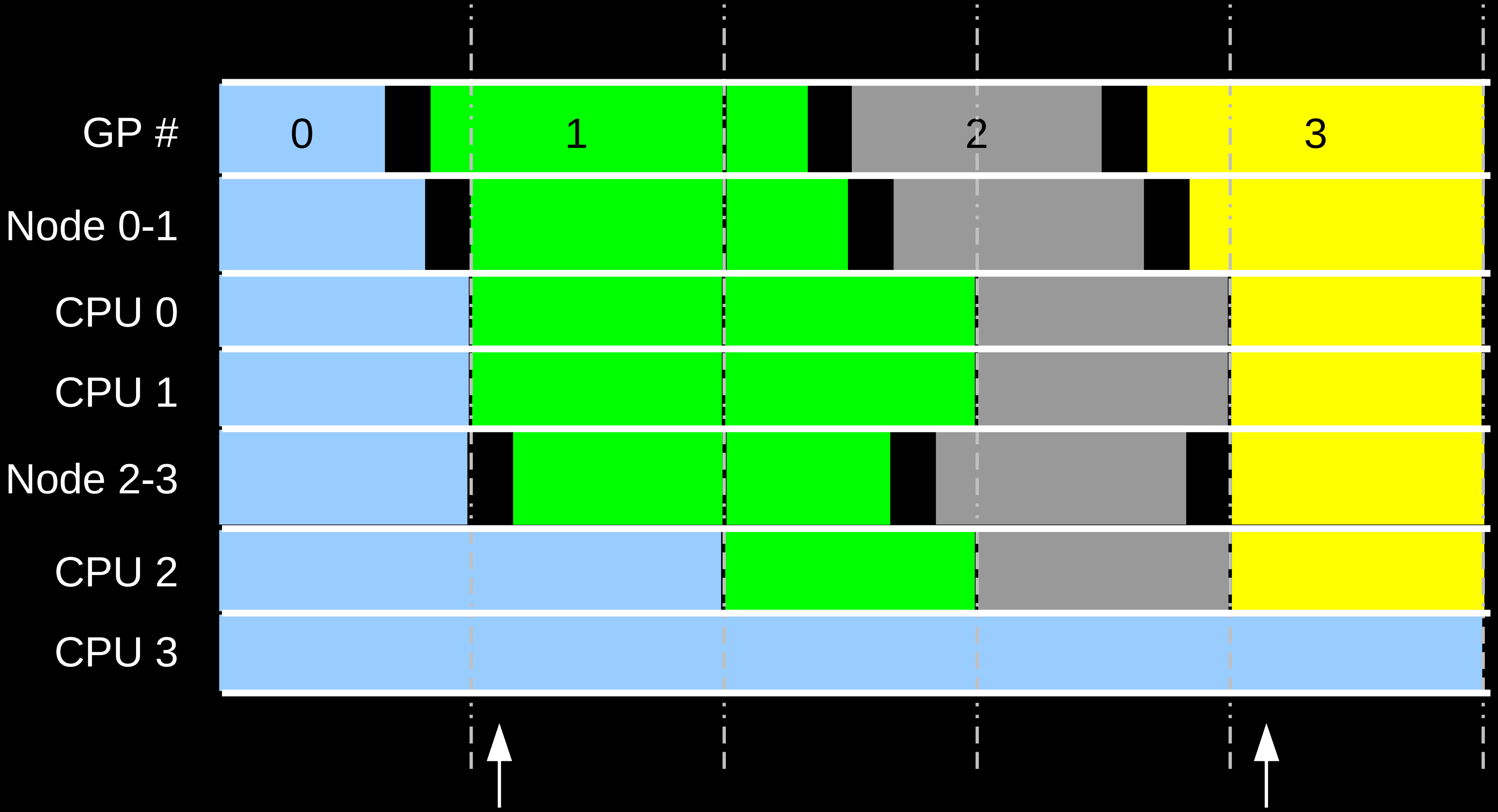
… are guaranteed done here

But CPU 3 is asleep and unaware!

Dealing With dyntick-idle Grace-Period Latency

- Don't allow CPUs with callbacks to go dyntick-idle
  - Which would unfortunately put us back where we started
- Try to force RCU state machine to drain callbacks
  - Already tried that, consumes too much CPU for too little benefit
- Impose time limit on dyntick-idle sojourns with callbacks
  - About 6 seconds if all lazy and about 4 jiffies if at least one non-lazy
  - Seems to work reasonably well: times can be adjusted at runtime
  - But still greatly degrades grace-period latency for dyntick-idle CPUs
- Mark callbacks with corresponding grace-period number

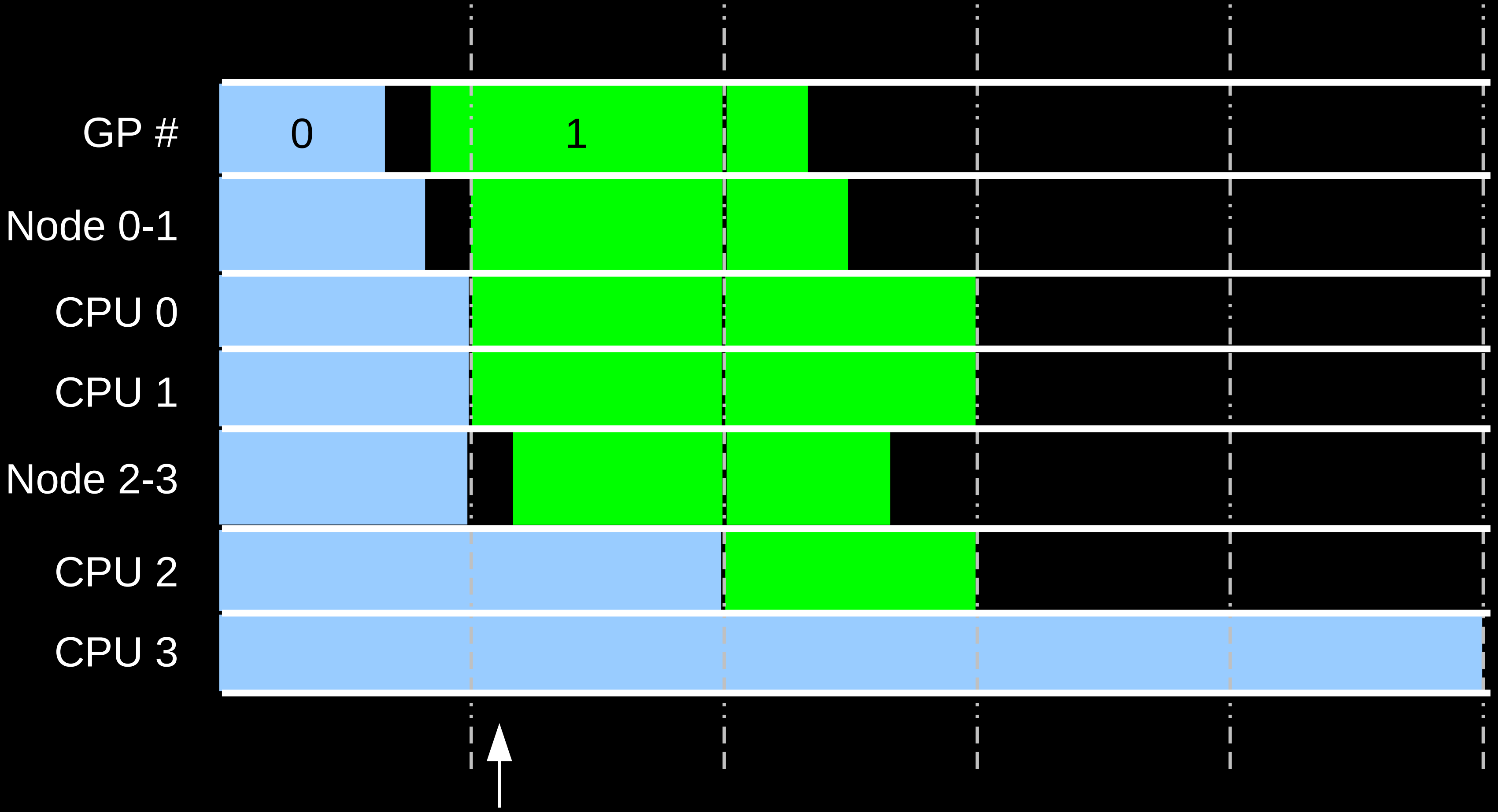
Grace-Period Handling, TREE_RCU, and dyntick-idle



Callbacks registered here are marked with grace period 2

And will be recognized as ready when CPU 3 awakens

But What If No Other CPU Needs Grace Period?

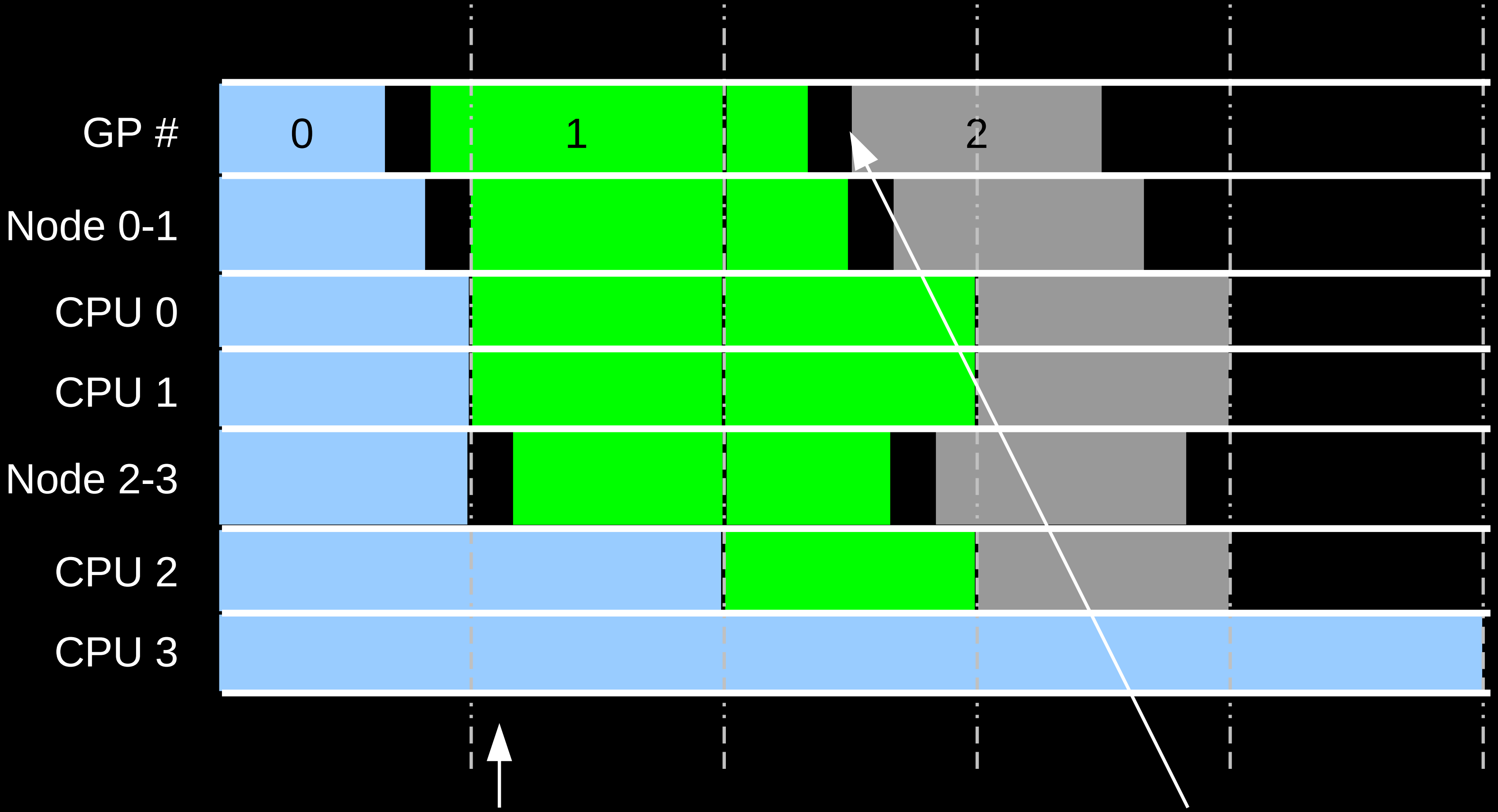


Callbacks registered and marked here, but grace period 2 never starts!!!

Dealing With dyntick-idle Grace-Period Latency

- Don't allow CPUs with callbacks to go dyntick-idle
  - **Which would unfortunately put us back where we started**
- Try to force RCU state machine to drain callbacks
  - **Already tried that, consumes too much CPU for too little benefit**
- Impose time limit on dyntick-idle sojourns with callbacks
  - About 6 seconds if all lazy and about 4 jiffies if at least one non-lazy
  - Seems to work reasonably well: times can be adjusted at runtime
  - But still degrades grace-period latency for dyntick-idle CPUs, so...
- Mark callbacks with corresponding grace-period number
  - But cannot start later grace periods, so...
- Register corresponding grace period with RCU core

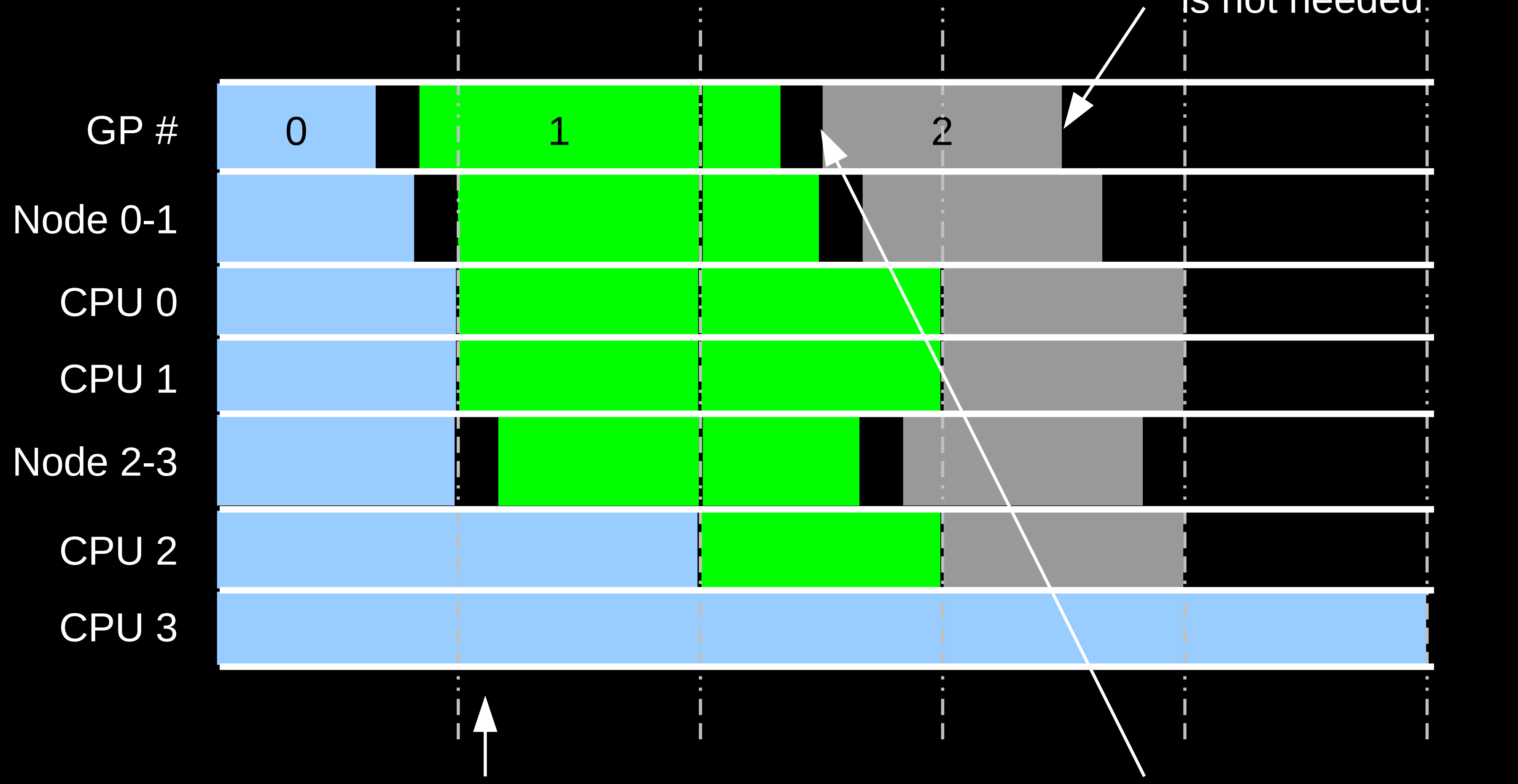
Grace-Period Handling, TREE_RCU, and dyntick-idle



Callbacks registered here are marked with grace period 2

And RCU knows to start grace period 2

Grace-Period Handling, TREE_RCU, and dyntick-idle



And that grace period 3 is not needed

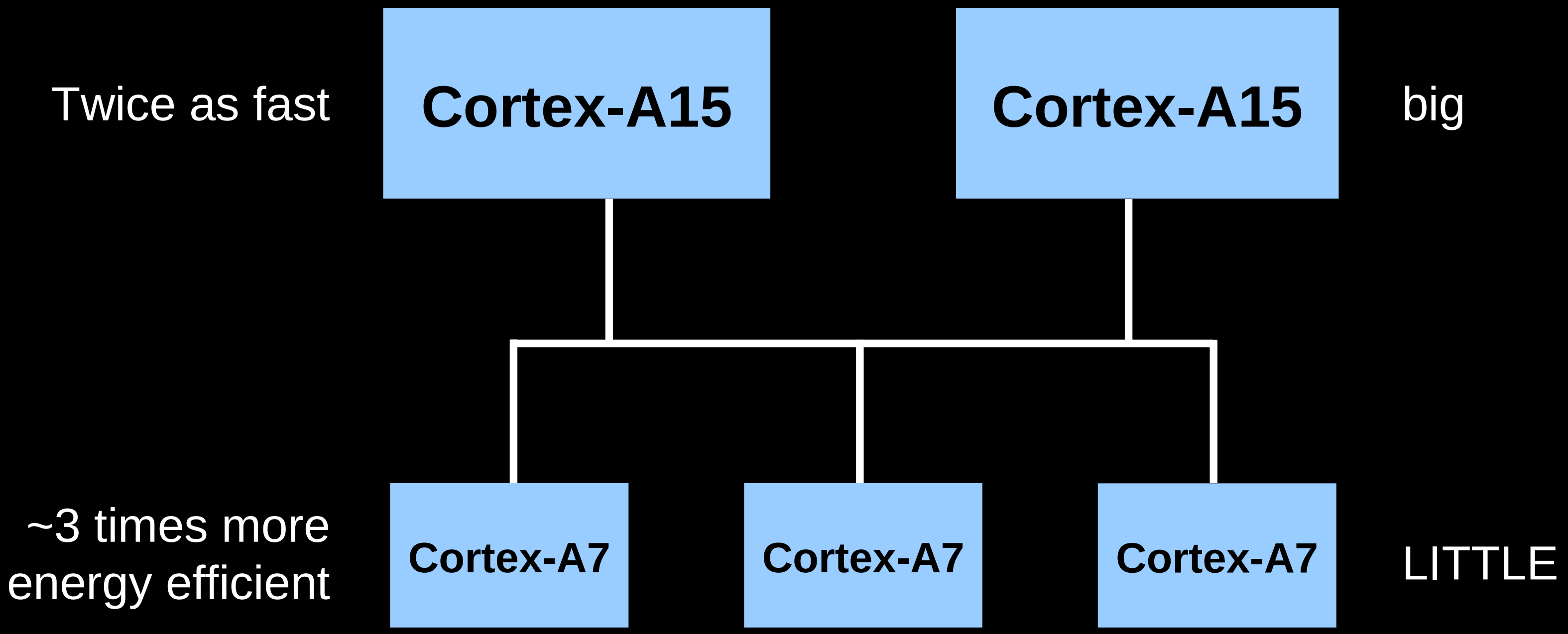
Callbacks registered here are marked with grace period 2

And RCU knows to start grace period 2

Preliminary Energy Efficiency Results

- Data courtesy of Dietmar Eggemann and Robin Randhawa of ARM on early-silicon big.LITTLE system
- Early results equivocal, but RCU_FAST_NO_HZ might not be helping much on big.LITTLE
  - Looking into kthread throttling and tuning
  - Also double-checking experiment setup
- Alternative approach: no-CBs CPUs!
- But what is big.LITTLE???

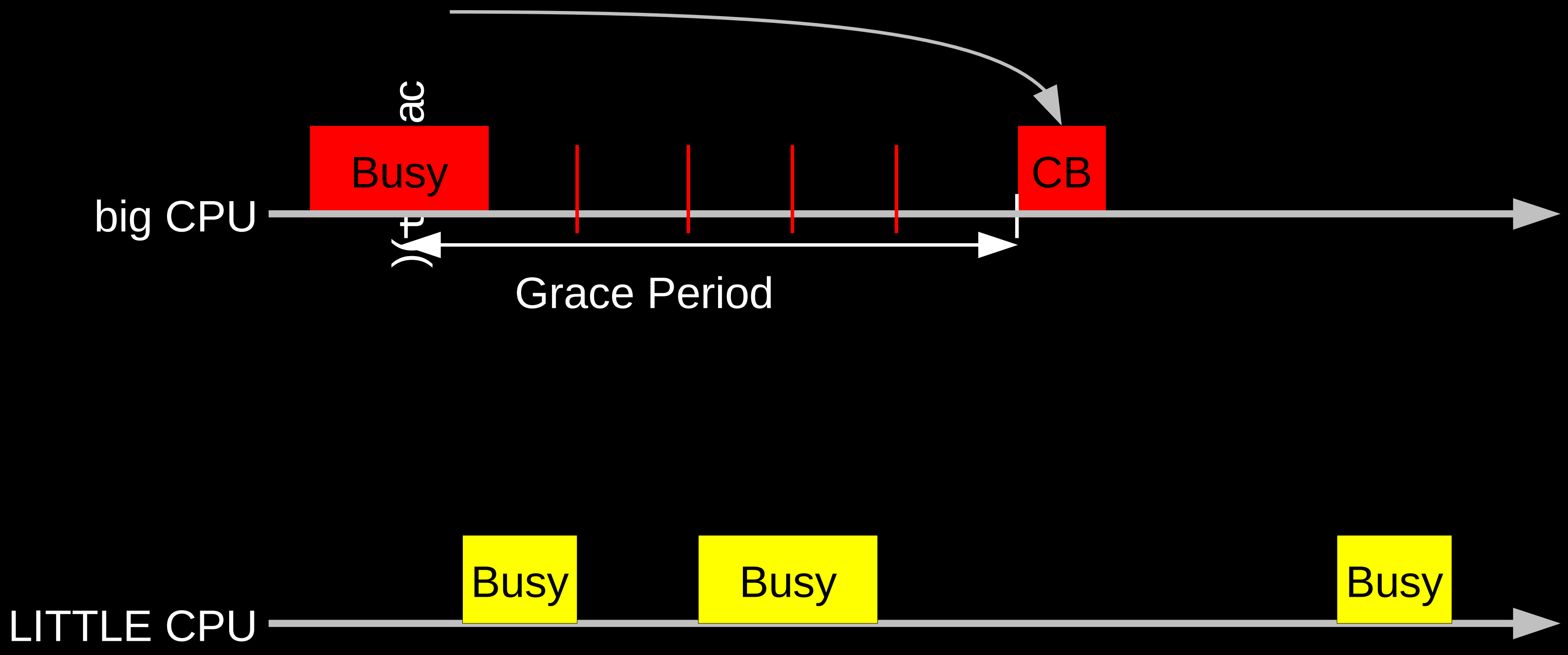
ARM big.LITTLE Architecture



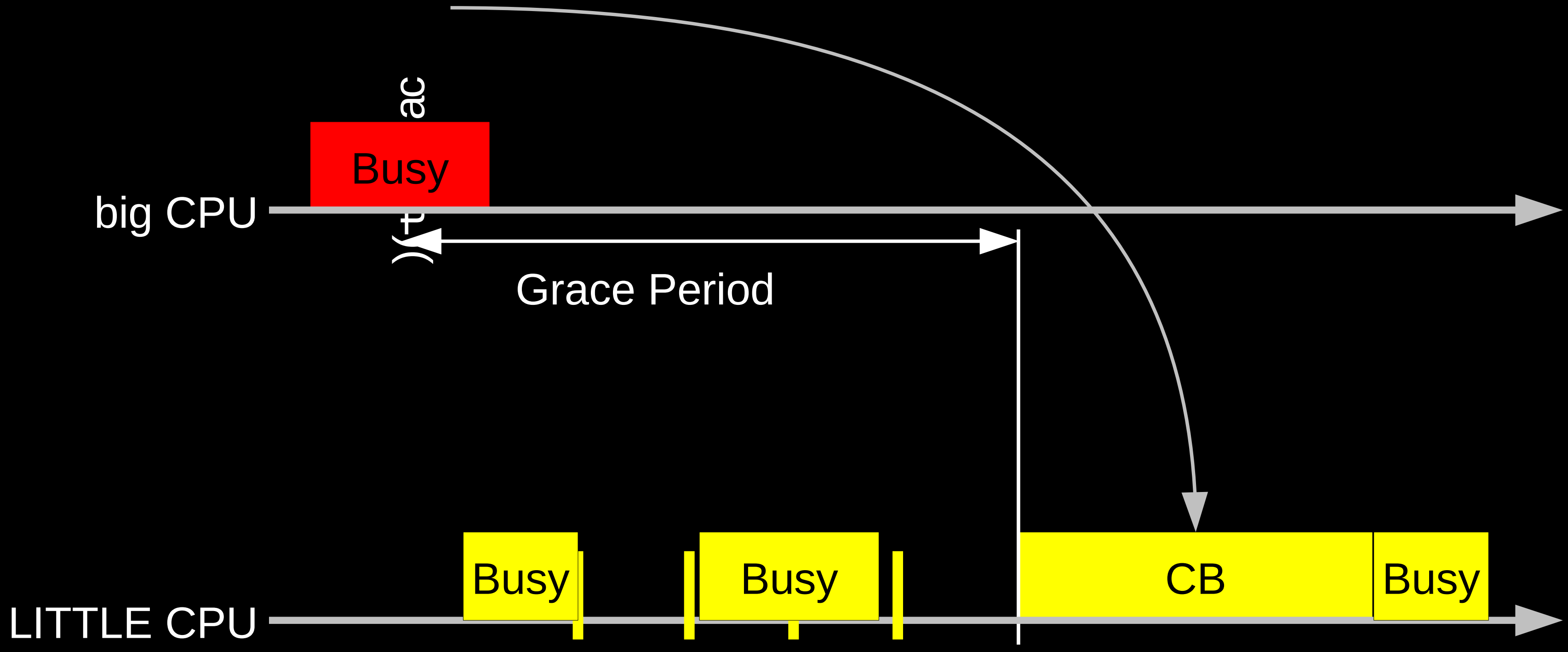
ARM big.LITTLE Architecture: Strategy

- Run on the LITTLE by default
- Run on big if heavy processing power is required
- In other words, if feasible, run on LITTLE for efficiency, but run on big if necessary to preserve user experience
  - This suggests that RCU callbacks should run on LITTLE CPUs

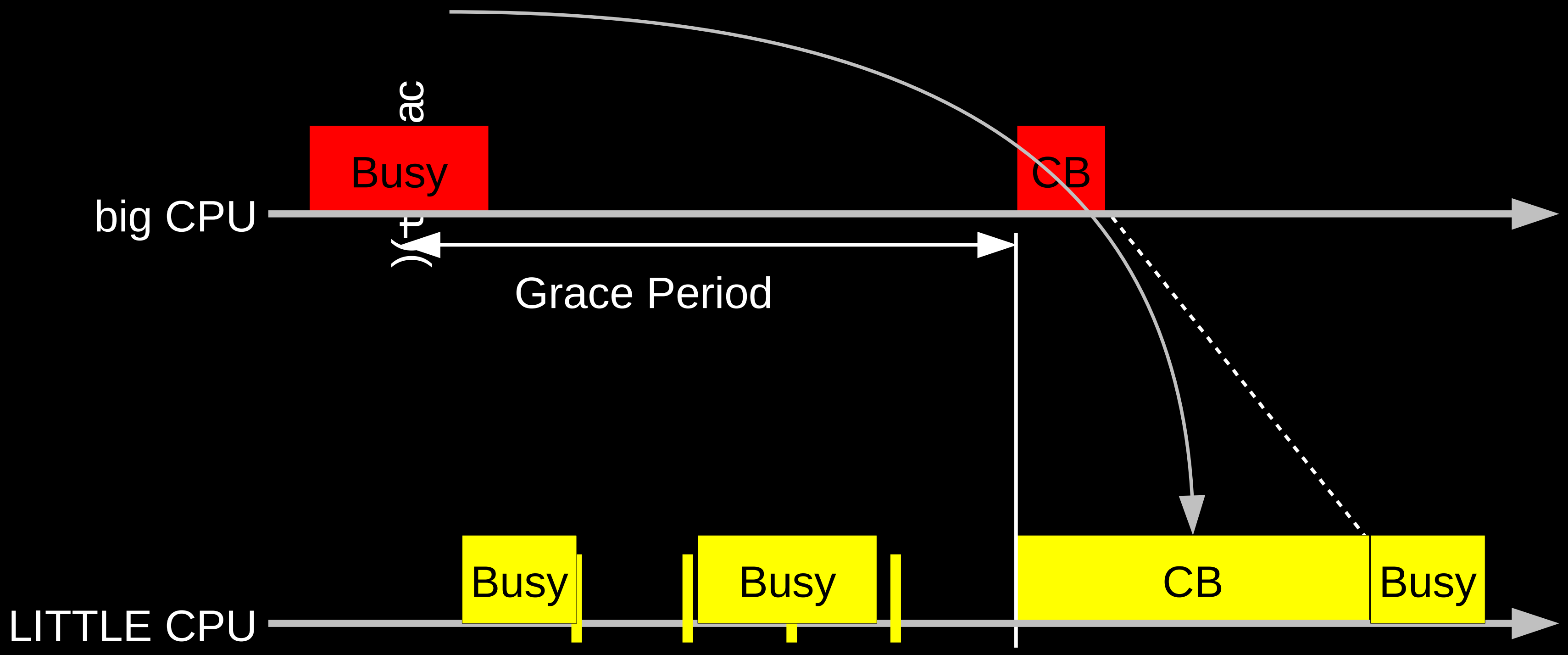
ARM big.LITTLE Without no-CBs CPUs



ARM big.LITTLE With no-CBs CPUs



ARM big.LITTLE With no-CBs CPUs: No Free Lunch



ARM big.LITTLE With no-CBs CPUs: Preliminary Results

- Reference System: RCU_NOCB_CPU=n
- Test System: RCU_NOCB_CPU=y, big CPUs offloaded, kthreads confined to LITTLE CPUs
- Approximate power savings:
  - cyclictest: 10%
  - andebench8: 2%
  - audio: 10%
  - bbench_with_audio: 5%
- Next steps:
  - Get no-CBs CPUs to production quality
  - More adjustment to RCU_FAST_NO_HZ

Offloadable RCU Callbacks: Limitations and Futures

- Probably several remaining bugs in no-CBs CPUs
  - Not yet production quality
- Must reboot to reconfigure no-CBs CPUs
  - Should be just fine for many uses
  - Hopefully also OK for HPC and real-time workloads
- No energy-efficiency code: lazy & non-lazy CBs? Non-lazy!
  - But non-lazy Cbs are common case, so deferring interpretation of laziness.
- No-CBs CPUs' kthreads not subject to priority boosting
  - Probably not a near-term problem
- Setting all no-CBs CPUs' kthreads to RT prio w/out pinning them: bad!
  - At least on large systems: Probably OK near-term, maybe long term as well
- Note: I do not expect no-CBs path to completely replace current CB path

To Probe More Deeply Into no-CBs CPUs...

- “Relocating RCU callbacks” by Jon Corbet
  - http://lwn.net/Articles/522262/
- “What Is New In RCU for Real Time (RTLWS 2012)”
  - http://www.rdrop.com/users/paulmck/realtime/paper/RTLWS2012occcRT.2012.10.19e.pdf
    - Slides 21-on
- “Getting RCU Further Out of the Way (Plumbers 2012)”
  - http://www.rdrop.com/users/paulmck/realtime/paper/nocb.2012.08.31a.pdf
- “Cleaning Up Linux’s CPU Hotplug For Real Time and Energy Management” (ECRTS 2012)
  - http://www.rdrop.com/users/paulmck/realtime/paper/hotplug-ecrts.2012.06.11a.pdf

Lessons Learned and Relearned

Lessons Learned, Old and New

- Workload matters!!!
  - Different workloads have different requirements
  - A given workload's requirements change over time
    - More important, one's understanding of requirements changes over time!
  - Supporting a single workload is easier than supporting many of them

Lessons Learned, Old and New

- Workload matters!!!
  - Different workloads have different requirements
  - A given workload's requirements change over time
    - More important, one's understanding of requirements changes over time!
  - Supporting a single workload is easier than supporting many of them
- Energy-efficiency and performance benchmarkers
  - You would never believe what either group will do for 5%...

Lessons Learned, Old and New

- Workload matters!!!
  - –Different workloads have different requirements
  - –A given workload's requirements change over time
    - More important, one's understanding of requirements changes over time!
  - –Supporting a single workload is easier than supporting many of them
- Energy-efficiency and performance benchmarkers
  - –You would never believe what either group will do for 5%...
- Median age of randomly chosen line of RCU code: < 2 years

Lessons Learned, Old and New

- Workload matters!!!
  - Different workloads have different requirements
  - A given workload's requirements change over time
    - More important, one's understanding of requirements changes over time!
  - Supporting a single workload is easier than supporting many of them
- Energy-efficiency and performance benchmarkers
  - You would never believe what either group will do for 5%...
- Median age of randomly chosen line of RCU code: < 2 years
- The guys who request an enhancement are rarely the guys who are willing to test your patches

Lessons Learned, Old and New

- Workload matters!!!
  - Different workloads have different requirements
  - A given workload's requirements change over time
    - More important, one's understanding of requirements changes over time!
  - Supporting a single workload is easier than supporting many of them
- Energy-efficiency and performance benchmarkers
  - You would never believe what either group will do for 5%...
- Median age of randomly chosen line of RCU code: < 2 years
- The guys who request an enhancement are rarely the guys who are willing to test your patches
- The importance of the community

A Brief History of RCU Issues

- ~1993: SMP scalability (30 CPUs) for RDBMS workloads
- 1996: NUMA (64 CPUs) for RDBMS workloads
- 2002: SMP scalability (~30 CPUs) for general workloads
- 2004: SMP scalability (~512 CPUs) for HPC workloads
  - And some concern about energy efficiency
- 2005: Real-time response (~4 CPUs)
- 2008: SMP scalability (>1024 CPUs) for HPC workloads
  - 100s of CPUs for more general workloads
- 2009: Real-time response (~30 CPUs) for general workloads
- 2010: Energy efficiency (~2 CPUs), real-time response when CPU-bound
- 2011: Energy efficiency (lots of CPUs)
- 2012: RCU causes 200-microsecond latency spikes...

And So I Owe The Linux Community Many Thanks

- Because of the many RCU-related challenges from the Linux community, some of my most important work and collaborations have been in the past ten years

And So I Owe The Linux Community Many Thanks

- Because of the many RCU-related challenges from the Linux community, some of my most important work and collaborations have been in the past ten years
- Not many people my age can truthfully say that

- Here is hoping for ten more years!!! ;-)

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