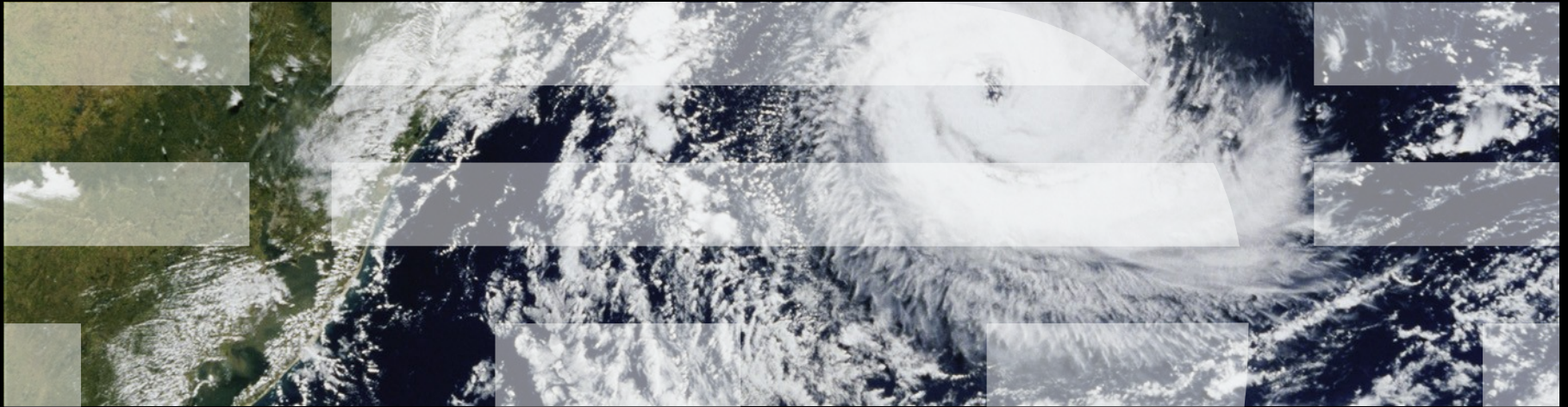Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center
Member, IBM Academy of Technology
Oregon State University BarCamp,  May April 20, 2013

IBM

# Bare-Metal Multicore Performance
# in a General-Purpose Operating System

**IBM**

# Group Effort: Acknowledgments

- Josh Triplett: First prototype (LPC 2009)

- Frederic Weisbecker: Core kernel work and x86 port

- Steven Rostedt: Lots of code review and comments

- Christoph Lameter: Early adopter feedback

- Li Zhong: Power port

- Geoff Levand, Kevin Hilman: ARM port

- Peter Zijlstra: Scheduler-related review, comments, and work

- Paul E. McKenney: Read-copy update (RCU) work

- Thomas Gleixner, Paul E. McKenney: "Godfathers"

- Ingo Molnar: Maintainer

- Other contributors:
    - Avi Kivity, Chris Metcalf, Geoff Levand, Gilad Ben Yossef, Hakan Akkan, Lai Jiangshan, Max Krasnyansky, Namhyung Kim, Paul Gortmaker, Paul Mackerras, Peter Zijlstra, Steven Rostedt, Zen Lin (and probably many more)

# What Do Database, HPC, and RT Developers Want?

# What Do Database, HPC, and RT Developers Want?

**Get The #@#$#*!!! Kernel Out
Of Our #@#$#*!!! Way!!!**

4

# What Do Database, HPC, and RT Developers Want?

**Get The #@#$#*!!! Kernel Out**
**Of Our #@#$#*!!! Way!!!**

But we need device drivers.

5

# What Do Database, HPC, and RT Developers Want?

## Get The #@#$#*!!! Kernel Out Of Our #@#$#*!!! Way!!!

But we need device drivers.
And file systems.

# What Do Database, HPC, and RT Developers Want?

## Get The #@#$#*!!! Kernel Out
## Of Our #@#$#*!!! Way!!!

But we need device drivers.
And file systems.
And memory protection.

7

# What Do Database, HPC, and RT Developers Want?

## Get The #@#$#*!!! Kernel Out Of Our #@#$#*!!! Way!!!

But we need device drivers.
And file systems.
And memory protection.
And...

# So What Are Us Poor Kernel Developers To Do???

Oregon State University BarCamp,  May April 20, 2013
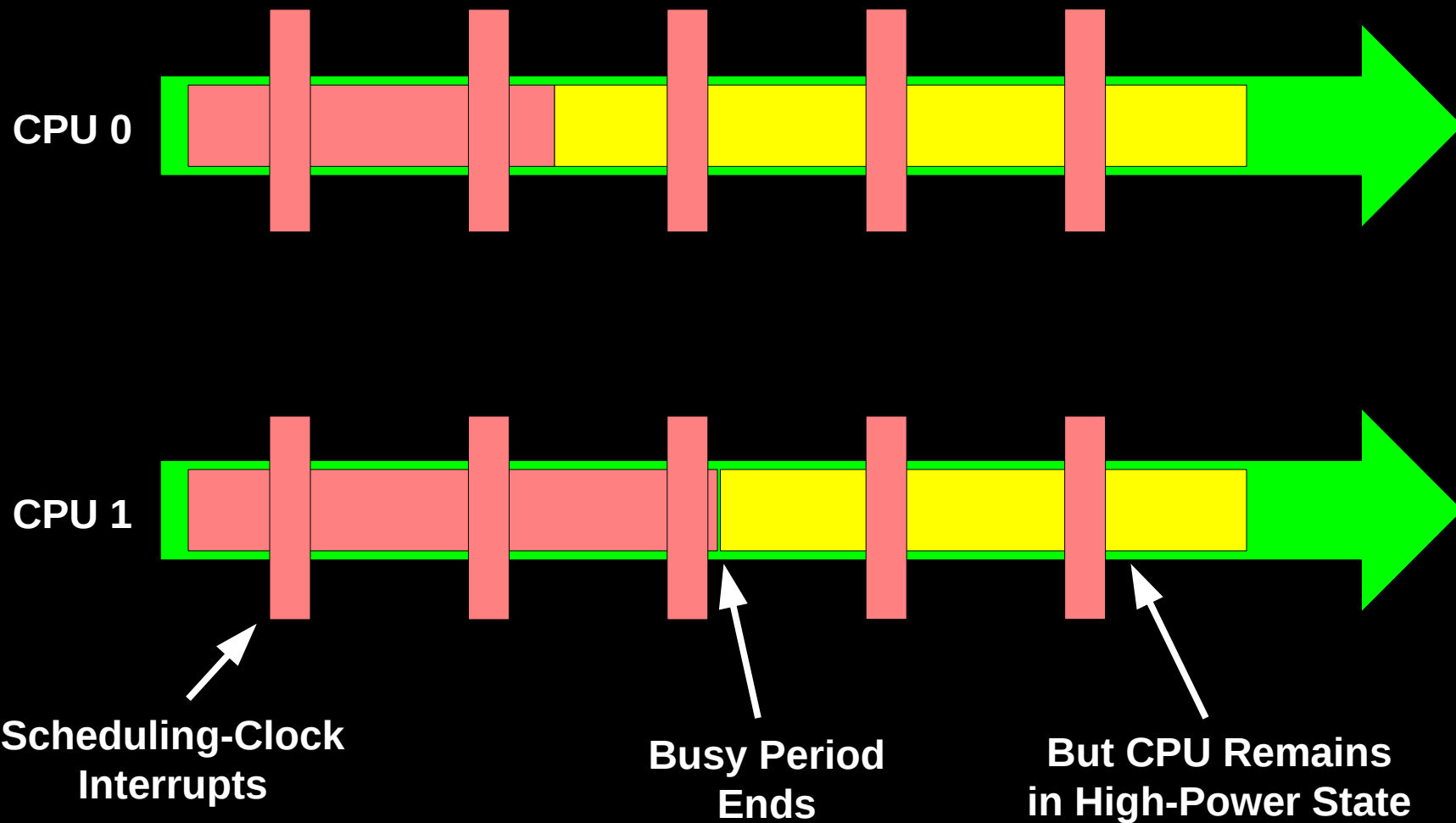
## So What Are Us Poor Kernel Developers To Do???

- For almost 20 years, my response was "Yeah, right, you really do want the whole kernel, just admit it already!!!"

© 2009 IBM Corporation

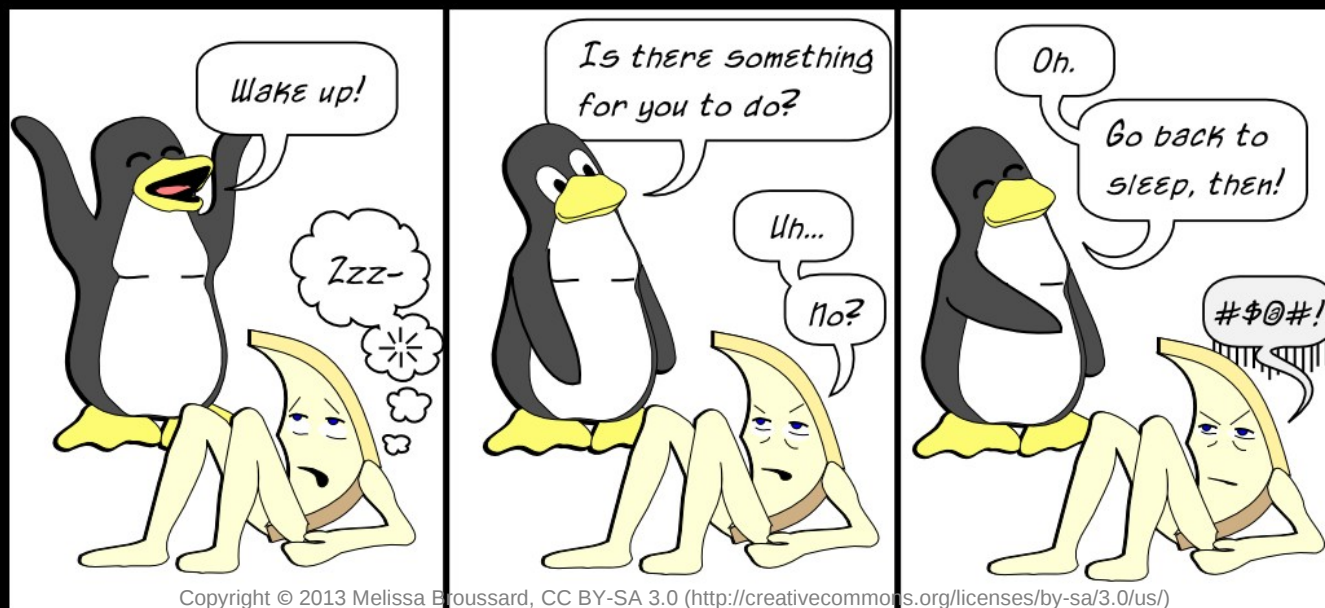# So What Are Us Poor Kernel Developers To Do???

- For almost 20 years, my response was "Yeah, right, you really do want the whole kernel, just admit it already!!!"

- My first clue otherwise was Linux's dyntick-idle system
  - (Used in battery-powered systems for years prior to Linux's use.)

# Before Linux's dyntick-idle System



**CPU 0**

**CPU 1**

**Scheduling-Clock Interrupts**

**Busy Period Ends**

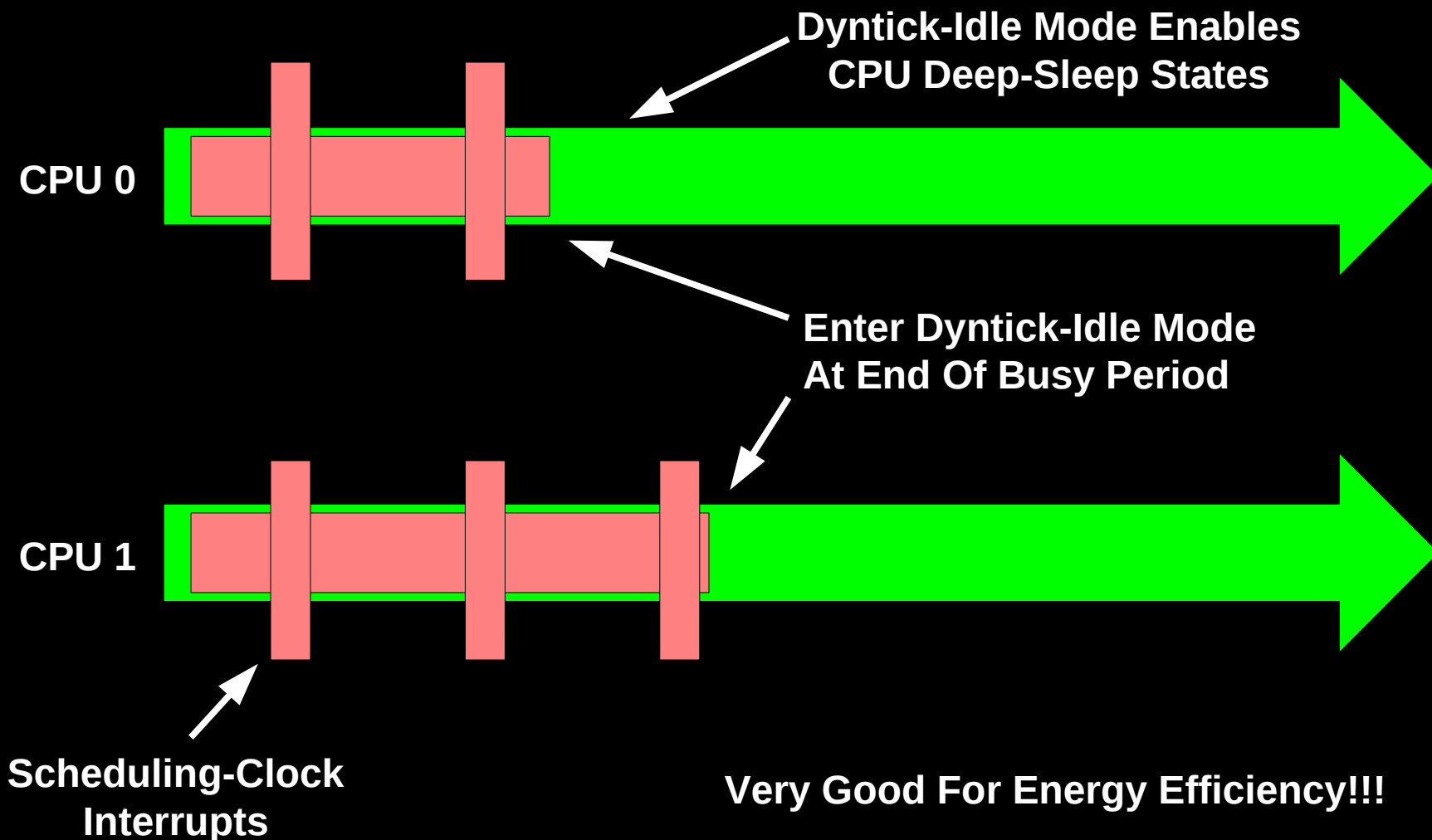**But CPU Remains in High-Power State**

# Scheduling-Clock Interrupts Really Optional???

- Scheduling-clock interrupt purpose:
  - Check for other work from time to time
  - Prevent a given process from monopolizing the CPU

- But if the CPU is idle, there is nothing for it to do anyway!!!

Copyright © 2013 Melissa Broussard, CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0/us/)

13

# Linux's dyntick-idle System

**Dyntick-Idle Mode Enables
CPU Deep-Sleep States**

**CPU 0**

**Enter Dyntick-Idle Mode
At End Of Busy Period**

**CPU 1**

**Scheduling-Clock
Interrupts**

**Very Good For Energy Efficiency!!!**

# Linux Kernel Is Now Out Of The Idle Loop's Way...

IBM

# Linux Kernel Is Now Out Of The Idle Loop's Way...
# So Can We Get It Out Of The Application's Way?

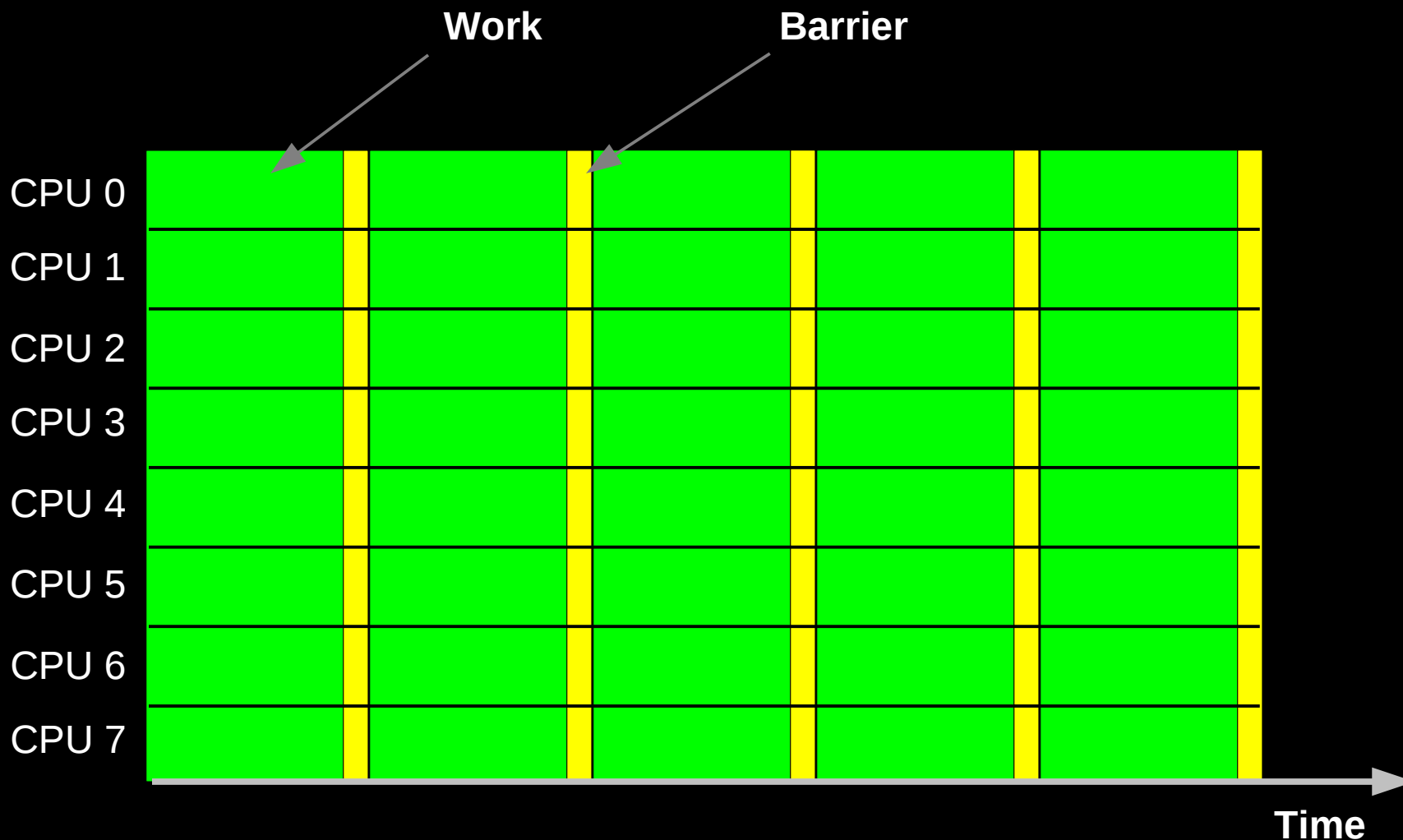# Is The Kernel Being In The Way Really A Problem?

# Is The Kernel Being In The Way Really A Problem?

- For aggressive real-time workloads, scheduling clock tick does add measurable latency
  - Some insane people really are getting sub-20-microsecond real-time interrupt latencies out of the Linux kernel...
  - And I happen to strongly believe in encouraging that sort of insanity!!!

# Is The Kernel Being In The Way Really A Problem?

- For aggressive real-time workloads, scheduling clock tick does add measurable latency
  - Some insane people really are getting sub-20-microsecond real-time interrupt latencies out of the Linux kernel...
  - And I happen to strongly believe in encouraging that sort of insanity!!!

- Some HPC workloads are sensitive to "OS jitter"
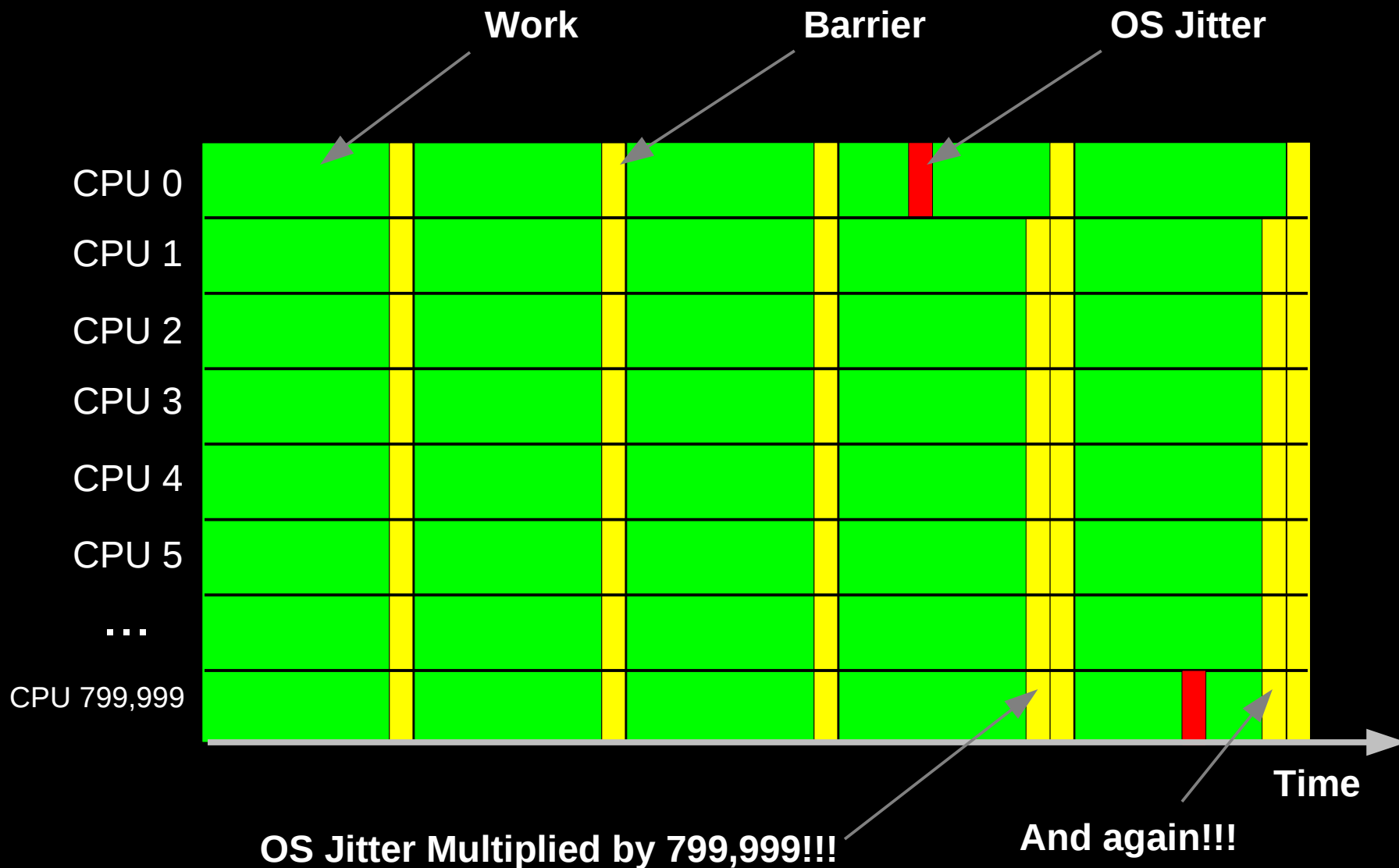  - Especially iterative workloads with short iterations

# Iterative Workloads With Short Iterations: Ideal

# Iterative Workloads With Short Iterations: OS Jitter



Work

Barrier

OS Jitter

CPU 0

CPU 1

CPU 2

CPU 3

CPU 4

CPU 5

CPU 6

CPU 7

Time

**OS Jitter Multiplied by 7!!!**

**And again!!!**

# Now Try This With 800,000 CPUs In A Cluster...



**Work**

**Barrier**

**OS Jitter**

CPU 0
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
...
CPU 799,999

**Time**

**OS Jitter Multiplied by 799,999!!!**

**And again!!!**

22

# Yes, This Is A Real Problem For Some Workloads

# Linux Kernel Is Now Out Of The Idle Loop's Way...
# So Can We Get It Out Of The Application's Way?

# Josh Triplett's First Prototype, 2009

- Always turn off scheduling-clock interrupt for user code

- Good demonstration of feasibility and benefit
  - 2009 Linux Plumbers Conference presentation
  - http://linuxplumbersconf.org/ocw/proposals/103
  - See next two slides for performance comparison

25

# Benchmark Results Before (Anton Blanchard)



100us period, baseline
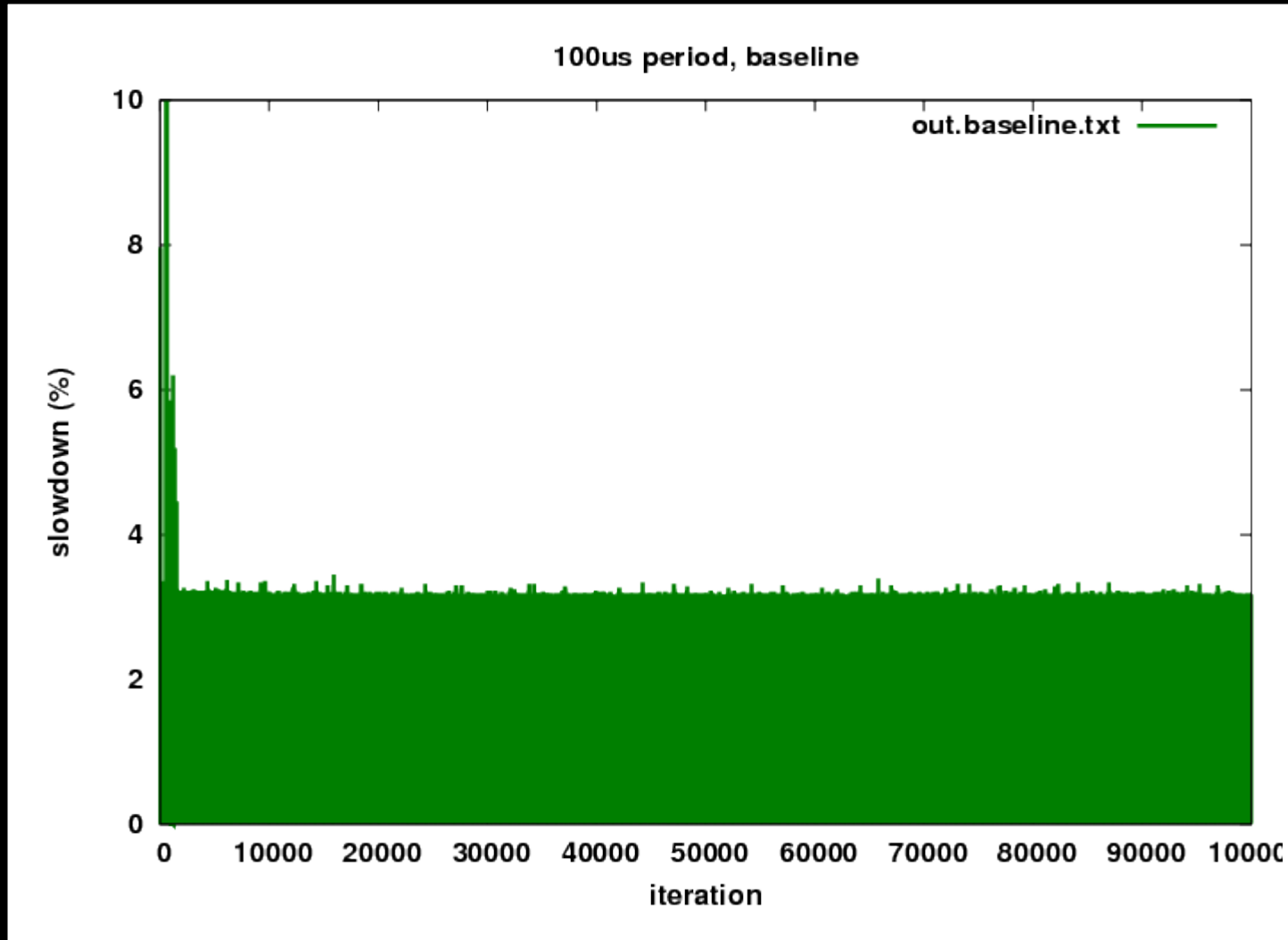
# Benchmark Results After (Anton Blanchard)
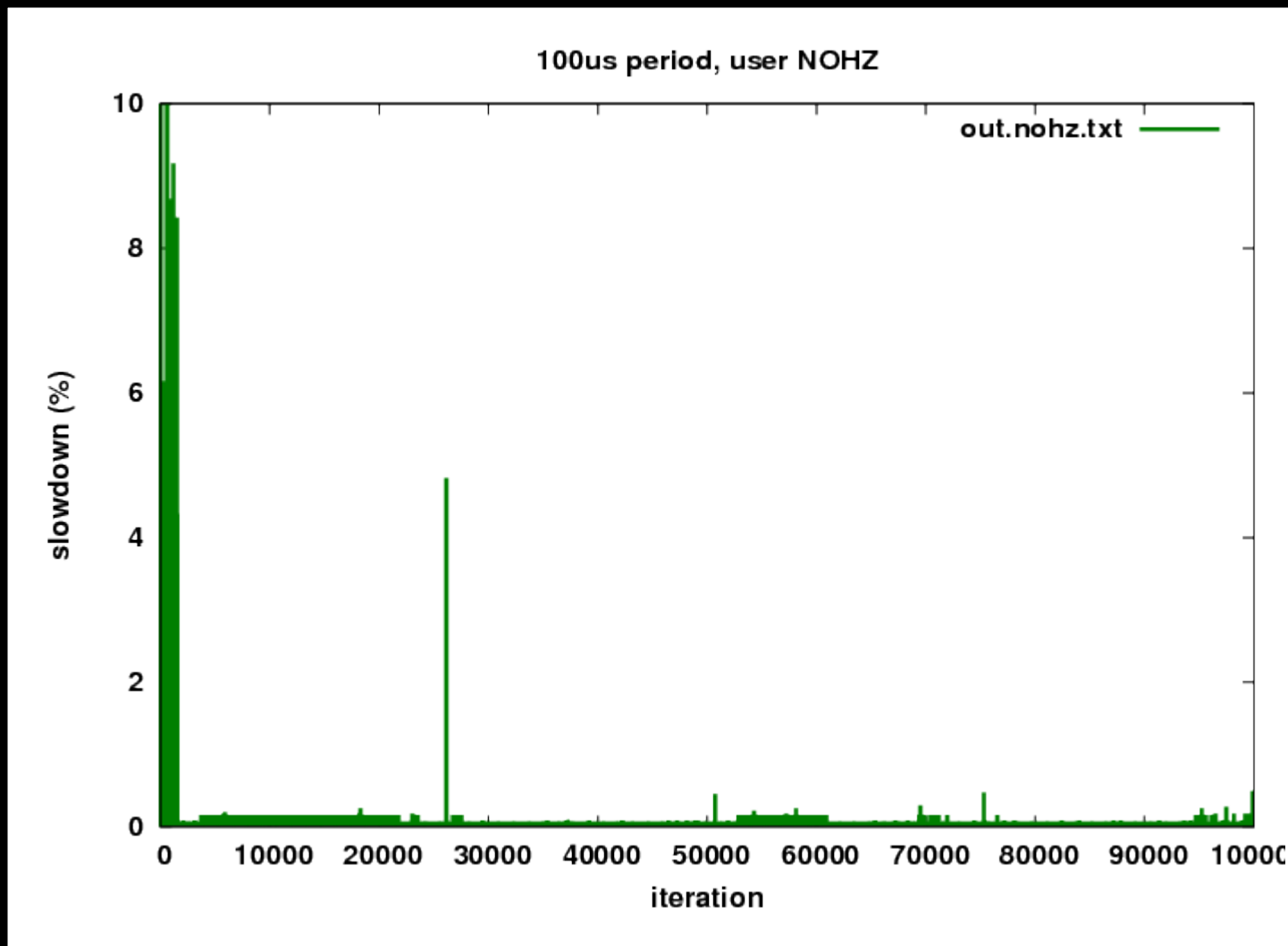


Well worth going after...

# But There Were A Few Small Drawbacks...

- No process accounting

- User applications can monopolize CPU

- RCU grace periods go forever, running system out of memory
  - More on this later

# Can We Do Something About The Drawbacks? (Discussion at 2010 Linux Plumbers Conference)

- User applications can monopolize CPU
  - But if there is only one runnable task, so what???
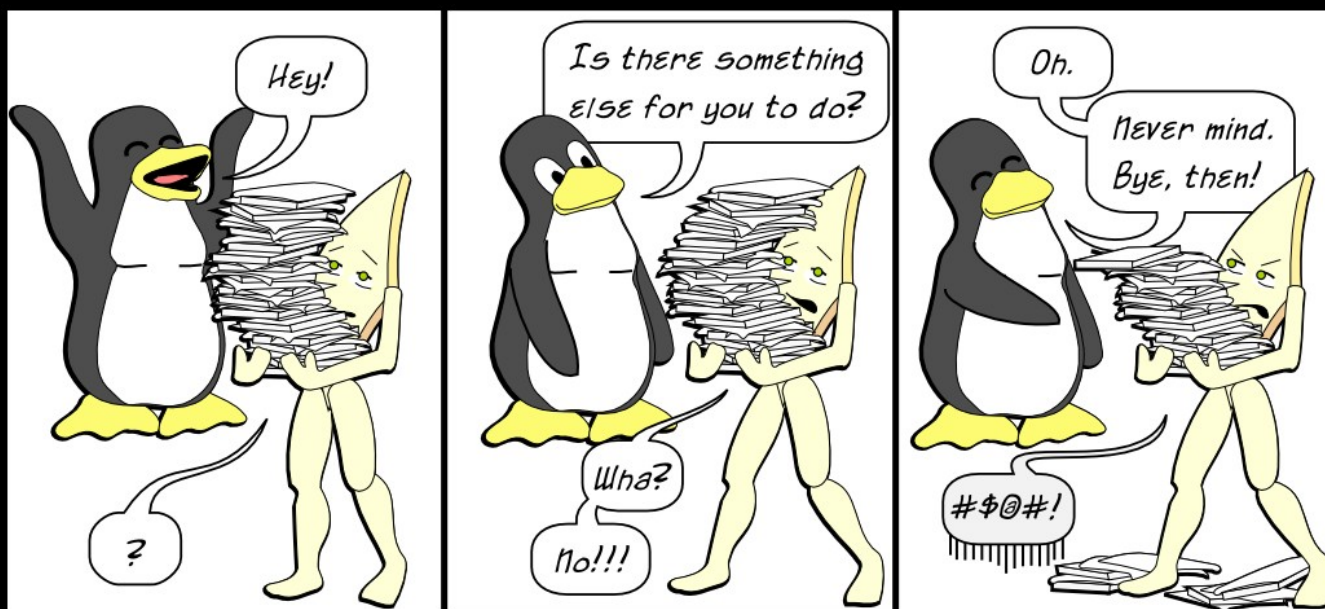
# So Another Look At The Drawbacks... (Discussion at 2010 Linux Plumbers Conference)

- User applications can monopolize CPU
  - But if there is only one runnable task, so what???
  - If new task awakens, interrupt the CPU, restart scheduling-clock interrrupts
  - In the meantime, we have an "adaptive idle usermode" CPU

- No process accounting
  - Use delta-based accounting, based on when process started running
  - One CPU retains scheduling-clock interrupts for timekeeping purposes

- RCU grace periods go forever, running system out of memory
  - Inform RCU of adaptive-idle usermode execution so that it ignores adaptive-idle user-mode CPUs, similar to its handling of dyntick-idle CPUs

- Frederic Weisbecker took on this task (for x86-64)
  - Geoff Levand and Kevin Hilman: Port to ARM
  - Li Zhong: Port to PowerPC
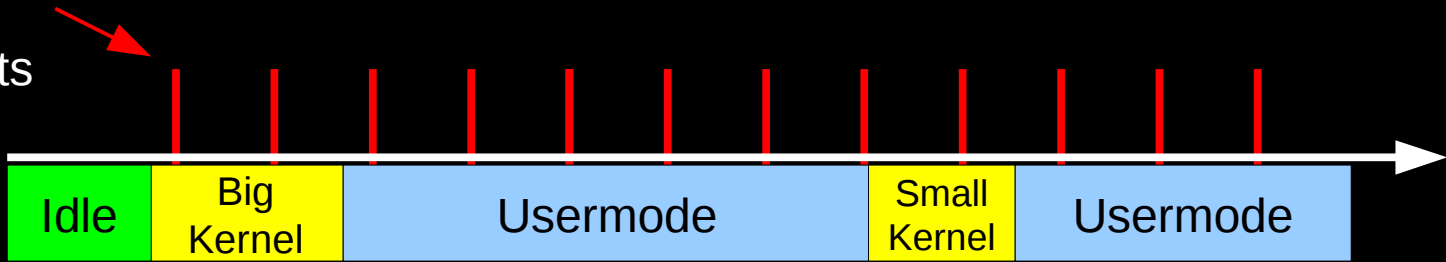  - I was able to provide a bit of help with RCU

30

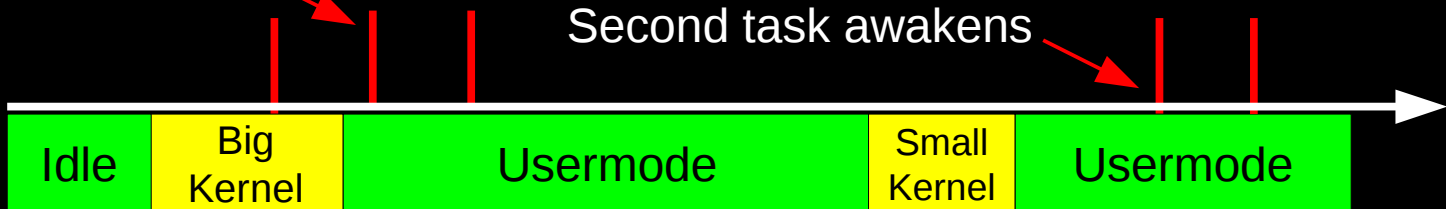# How Well Does It Work?

# How Well Does It Work?

- Preliminary results look good
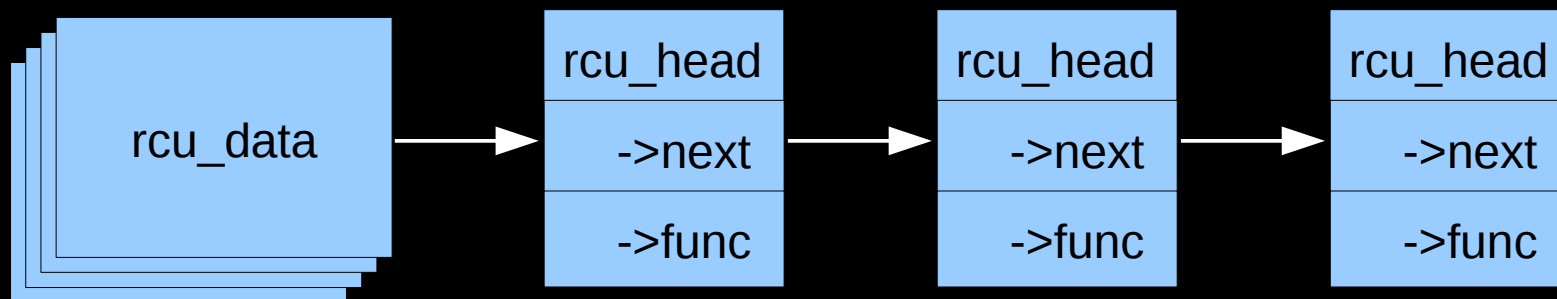
# How Well Does It Work?

# Other Than RCU, Looks Great!!!

- Need to fix RCU

- But first, what is RCU?

# What Is RCU?

# What Is RCU? (AKA Read-Copy Update)

- 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"
  - Deferring the work a bit longer than needed is OK, deferring too long is bad – but failing to defer long enough is fatal
  - RCU allows extremely fast & scalable read-side access to shared data

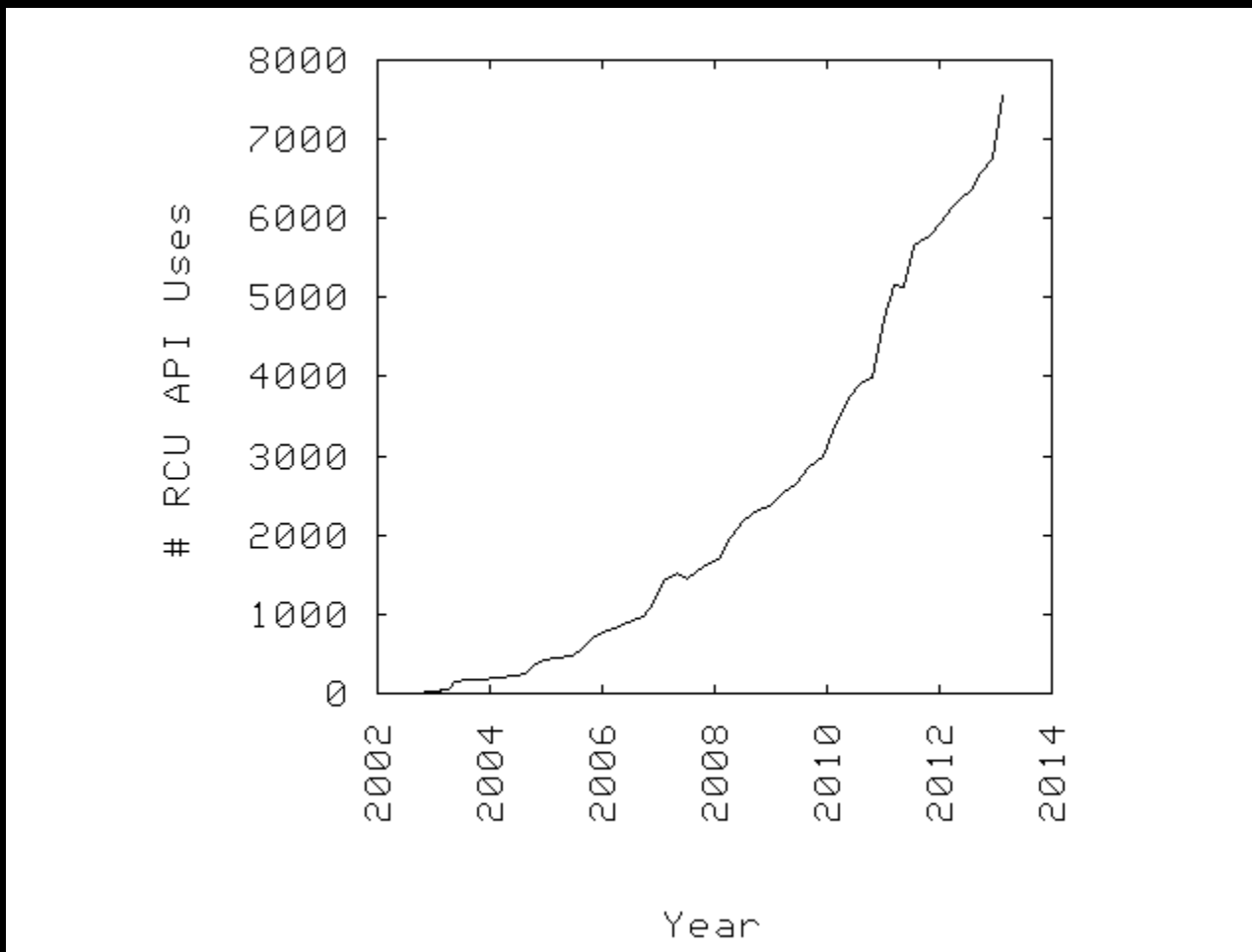| rcu_data | | rcu_head | | rcu_head | | rcu_head |
|----------|---|----------|---|----------|---|----------|
| | → | ->next | → | ->next | → | ->next |
| | | ->func | | ->func | | ->func |

36

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

37

# 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,
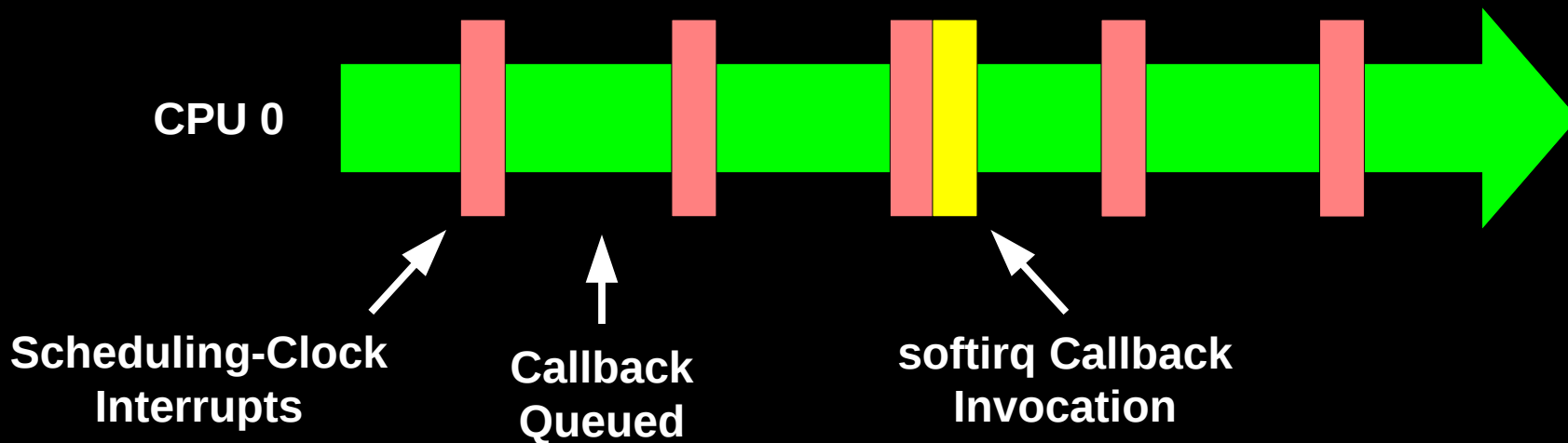But SLAB_DESTROY_BY_RCU Is A Possibility)
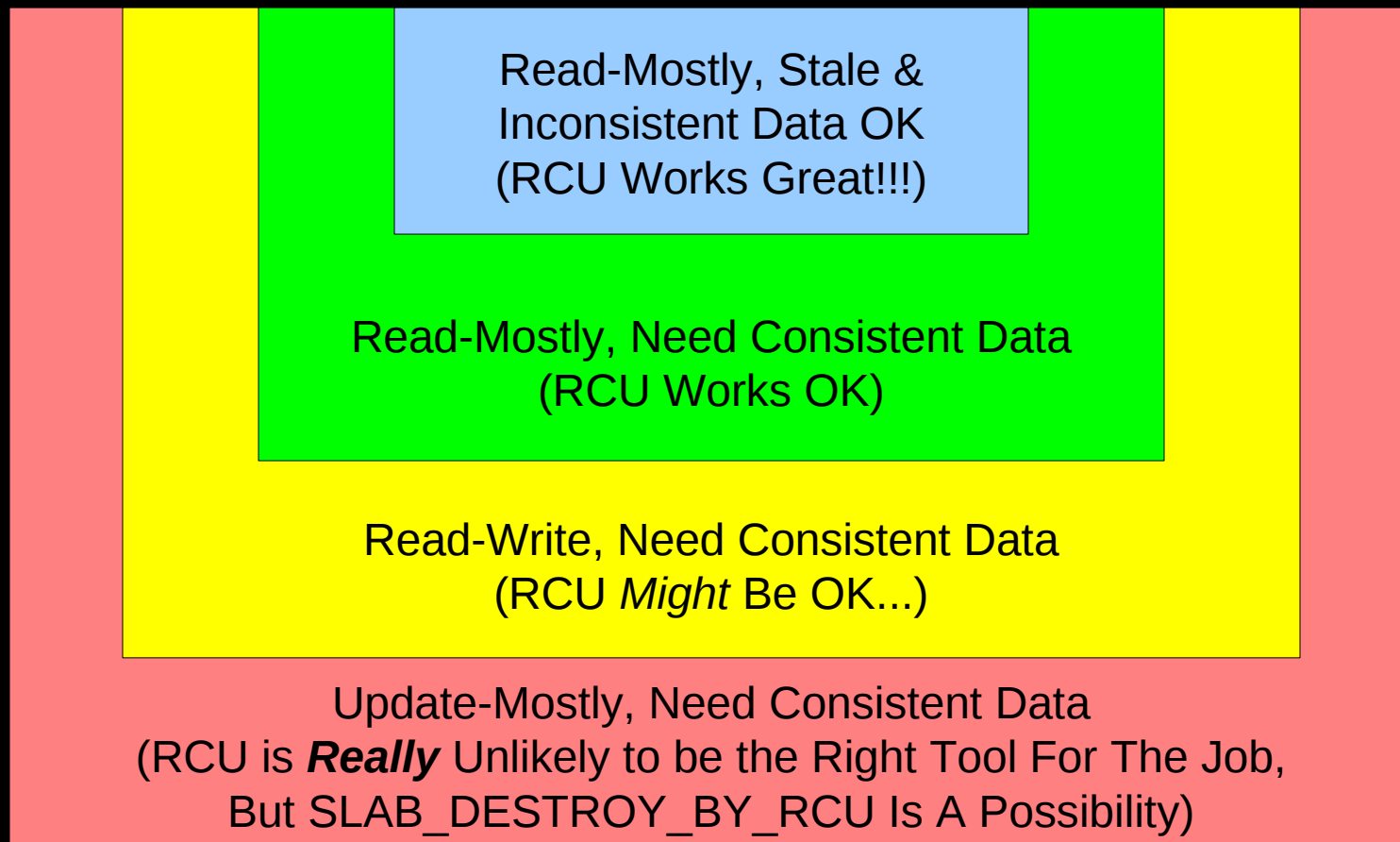
*Use the right tool for the job!!!*
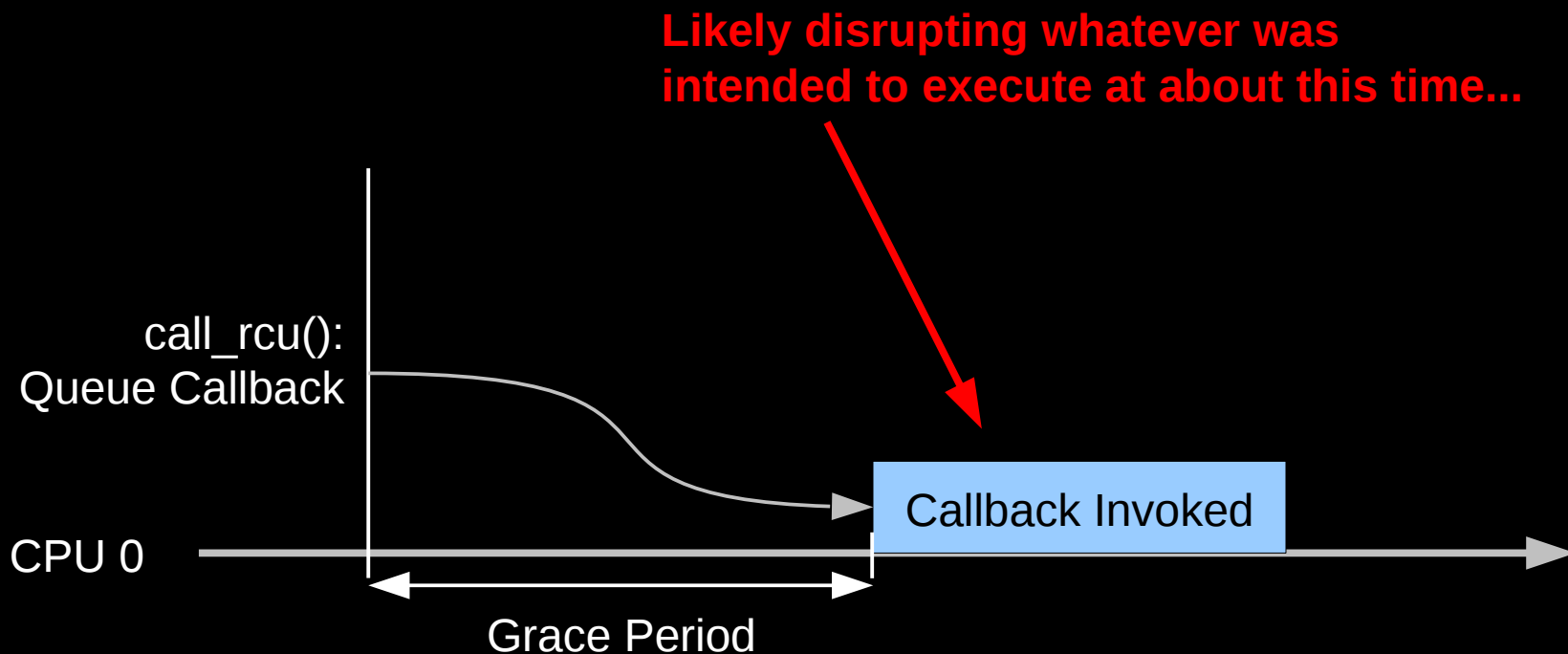
# Applicability To The Linux Kernel

# What Is RCU?  (AKA Read-Copy Update)

- 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

CPU 0

Scheduling-Clock
Interrupts

Callback
Queued

softirq Callback
Invocation

# Procrastination's Dark Side

# Procrastination's Dark Side: Eventually Must Do Work

**Likely disrupting whatever was intended to execute at about this time...**

call_rcu():
Queue Callback

Callback Invoked

CPU 0

Grace Period

# Why Not Offload RCU's Callbacks?

# Offload RCU Callbacks: Houston/Korty Approach

Grace Period

CPU 0

call_rcu()

RCU
(CPU 1)

Callback Invoked | Callback Invoked

call_rcu()

CPU 2

**No disruption!**

44

# Offload RCU Callbacks: Houston/Korty Approach

Grace Period

CPU 0

call_rcu()

RCU
(CPU 1)

Callback Invoked    Callback Invoked

call_rcu()

CPU 2
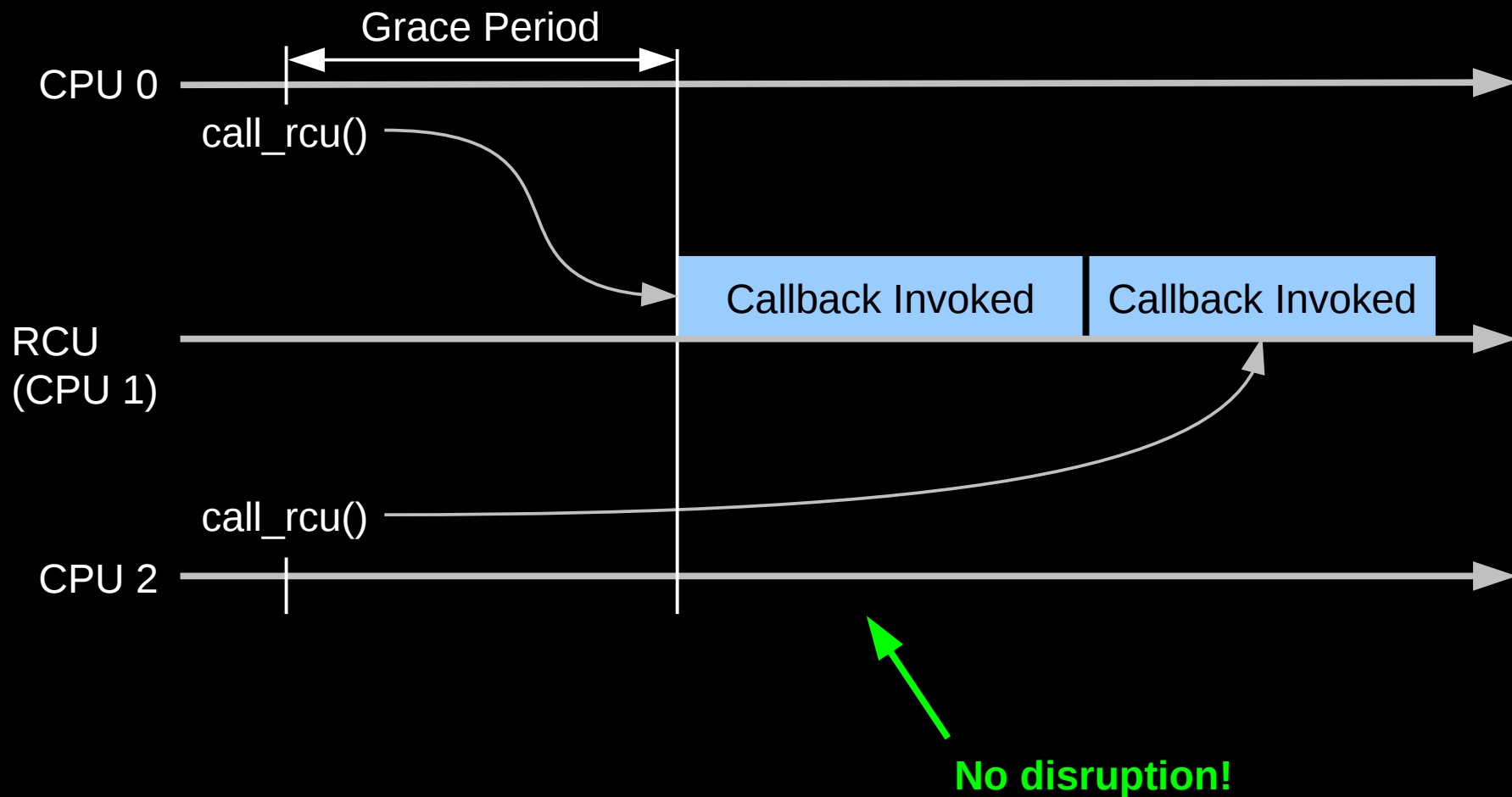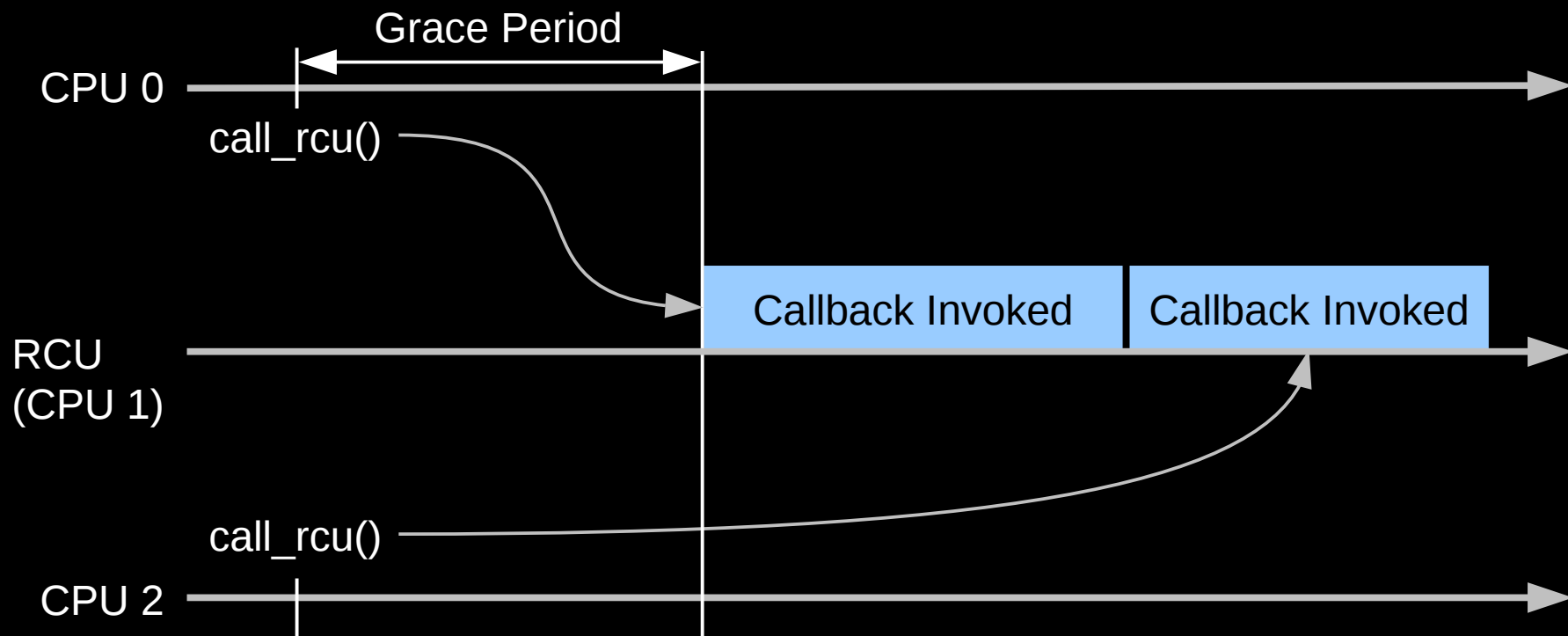
**No disruption!
(But also no scalability,
and Linux kernel must scale)**

45

# Scalable RCU Callback Offloading

Grace Period

CPU 1

call_rcu()

Callback Invoked

rcuo kthread

rcuo kthread

Callback Invoked

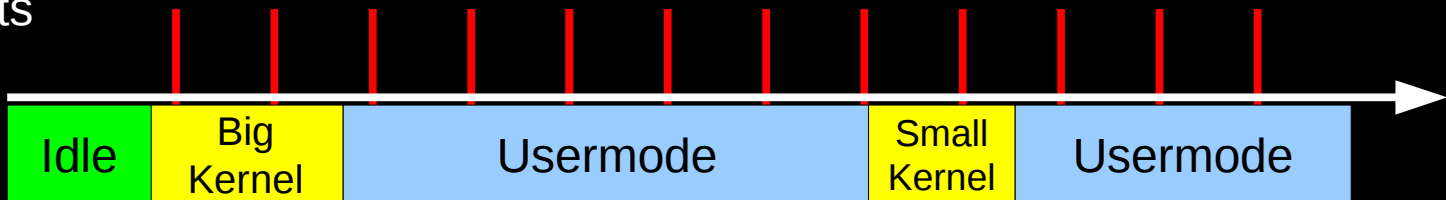call_rcu()

CPU 2

Grace Period
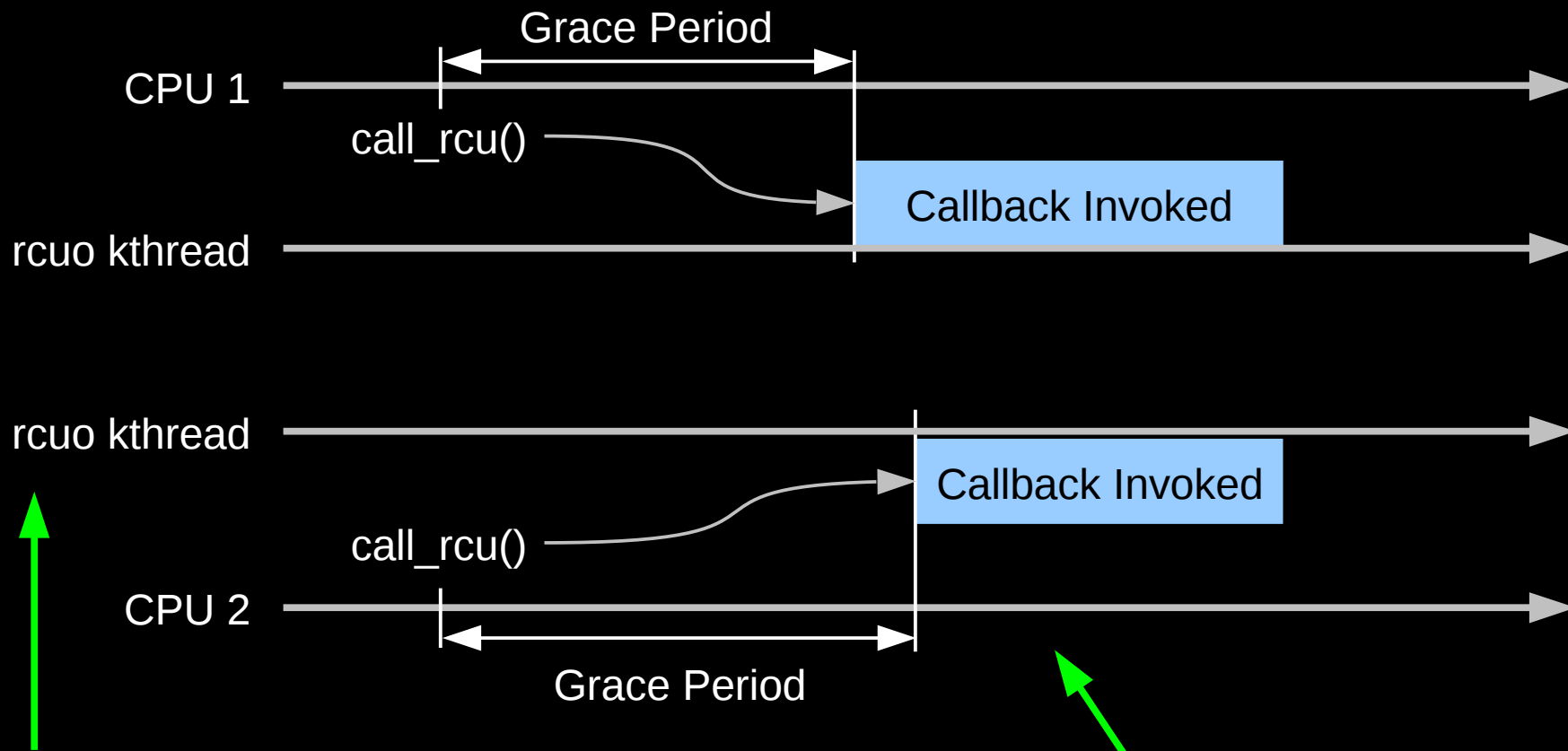
**Scheduler controls placement
(or can place manually)**

**No disruption!
Plus scalability!!!**

# Adaptive Ticks *And* Callback Offloading

Scheduling
clock
interrupts

Idle | Big Kernel | Usermode | Small Kernel | Usermode

**Adaptive Ticks**

RCU no longer causes
extra scheduling clock
interrupts

Second task awakens

Idle | Big Kernel | Usermode | Small Kernel | Usermode

One task per CPU

47

# Where To Run RCU Callbacks???

# Where To Run RCU Callbacks???

| | |
|---|---|
| **CPU 0** | **Interrupts, Management, Callbacks** |
| **CPU 1** | **(Massive Disruption for Housekeeping)** |
| **CPU 2** | |
| **CPU 3** | |
| **CPU 4** | **Worker Threads (HPC, Real Time)** |
| **CPU 5** | **(No Disruption for Real Work)** |
| **CPU 6** | |
| **CPU 7** | |
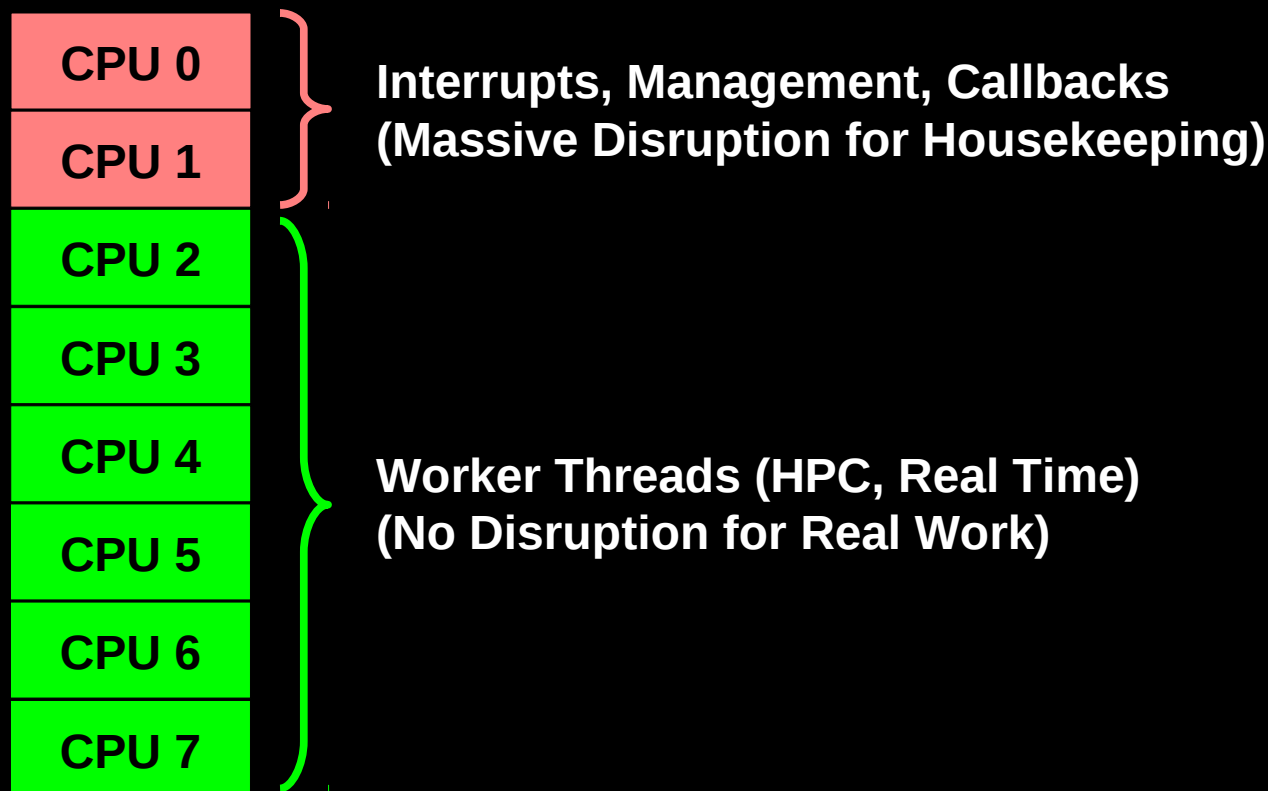
**Exact Layout Depends on Workload**

# How Well Does It Work?

# How Well Does It Work?

- Preliminary data looks good: also helps save energy

- Some shortcomings, as always:
  - Adaptive-idle usermode slows user/kernel transitions slightly
    - Not a problem for computation-intensive workloads
  - One task per CPU for adaptive-idle usermode execution
    - Also not a problem for many computation-intensive workloads
    - Could generalize: 1 SCHED_FIFO + N SCHED_OTHER, for example
  - Must reboot to reconfigure adaptive idle and RCU callback offloading
  - Must configure interrupts and processes manually (see next slide)
  - Some remaining OS jitter sources (TLB, page faults, …)
    - Constrain workload to avoid these jitter sources
  - At least one CPU must keep scheduling-clock interrupt (timekeeping)
  - RCU callback-offloading kthreads (rcuo) not priority boosted
    - Rely on configuration restrictions leaving idle time on housekeeping CPUs
    - Or run rcu0 kthreads at real-time priority (not for the faint of heart!)
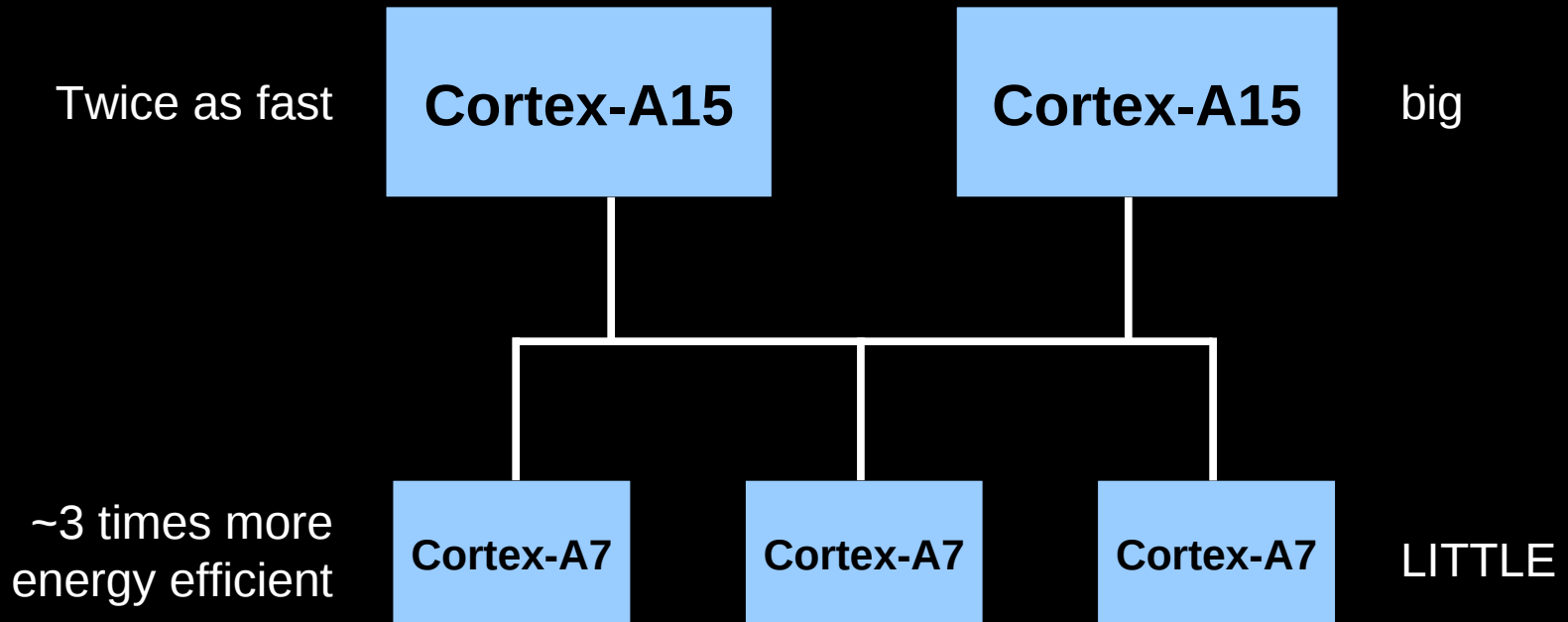  - Work in progress:  There are probably still a few bugs!

# Removing Other Sources of Disturbance

- Interrupts: /proc/irq/*/
  - One directory for each IRQ
  - smp_affinity file for hexadecimal specification (0x03)
  - smp_affinity_list for decimal CPU-list specification (0-1)
  - Verify via /proc/interrupts
  - Documentation/IRQ-affinity.txt in Linux kernel source for more info

- Timers: CPU hotplug remove, then reinsert

- Processes, daemons, and kthreads:
  - Per-task affinity (taskset command, sched_setaffinity() syscall)
  - cgroups or cpusets (Documentation/cgroups/*.txt)

- Global TLB-flush operations
  - Can be caused by kernel module unloading
    - So don't unload kernel modules on production systems!

- Cache and TLB misses are still with us: Use huge pages!!!

52

# RCU Callback Offloading: Energy Efficiency

- Preliminary data courtesy of Dietmar Eggemann and Robin Randhawa of ARM on early-silicon big.LITTLE system
- But what is big.LITTLE???

# ARM big.LITTLE Architecture

Twice as fast

**Cortex-A15**    **Cortex-A15**    big

~3 times more
energy efficient

**Cortex-A7**    **Cortex-A7**    **Cortex-A7**    LITTLE

# 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
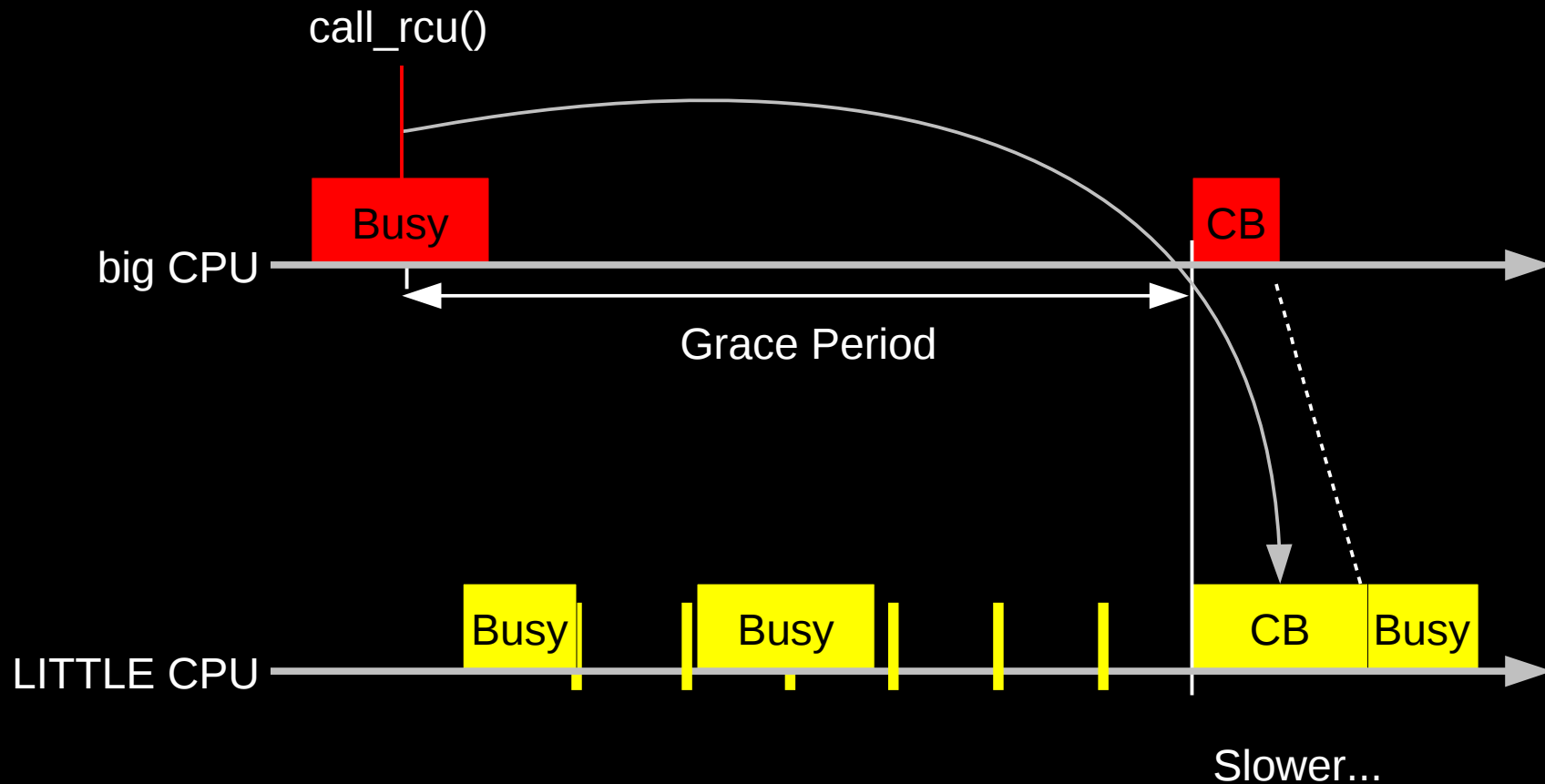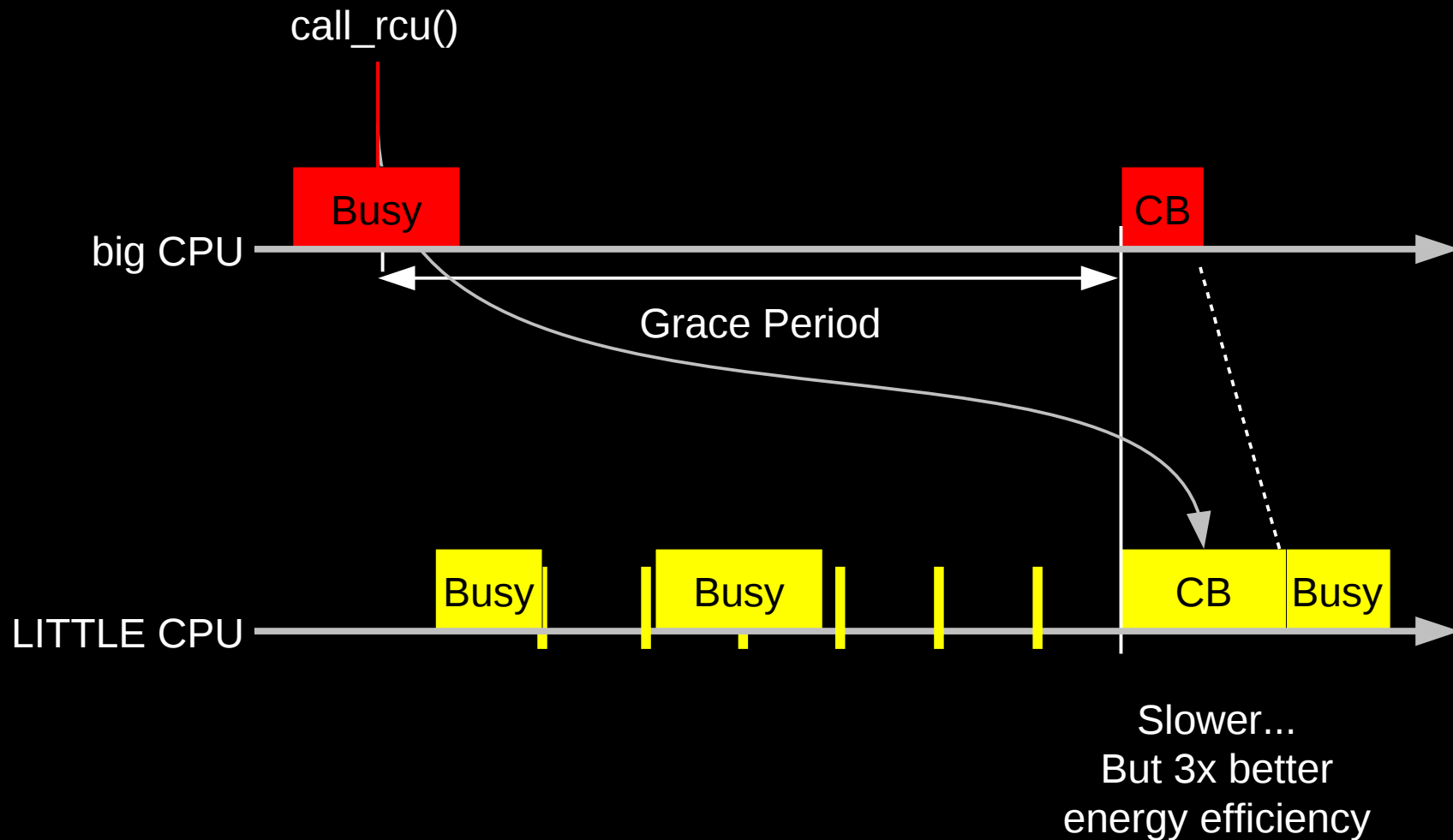  - This suggests that RCU callbacks should run on LITTLE CPUs

# ARM big.LITTLE Without RCU Callback Offloading
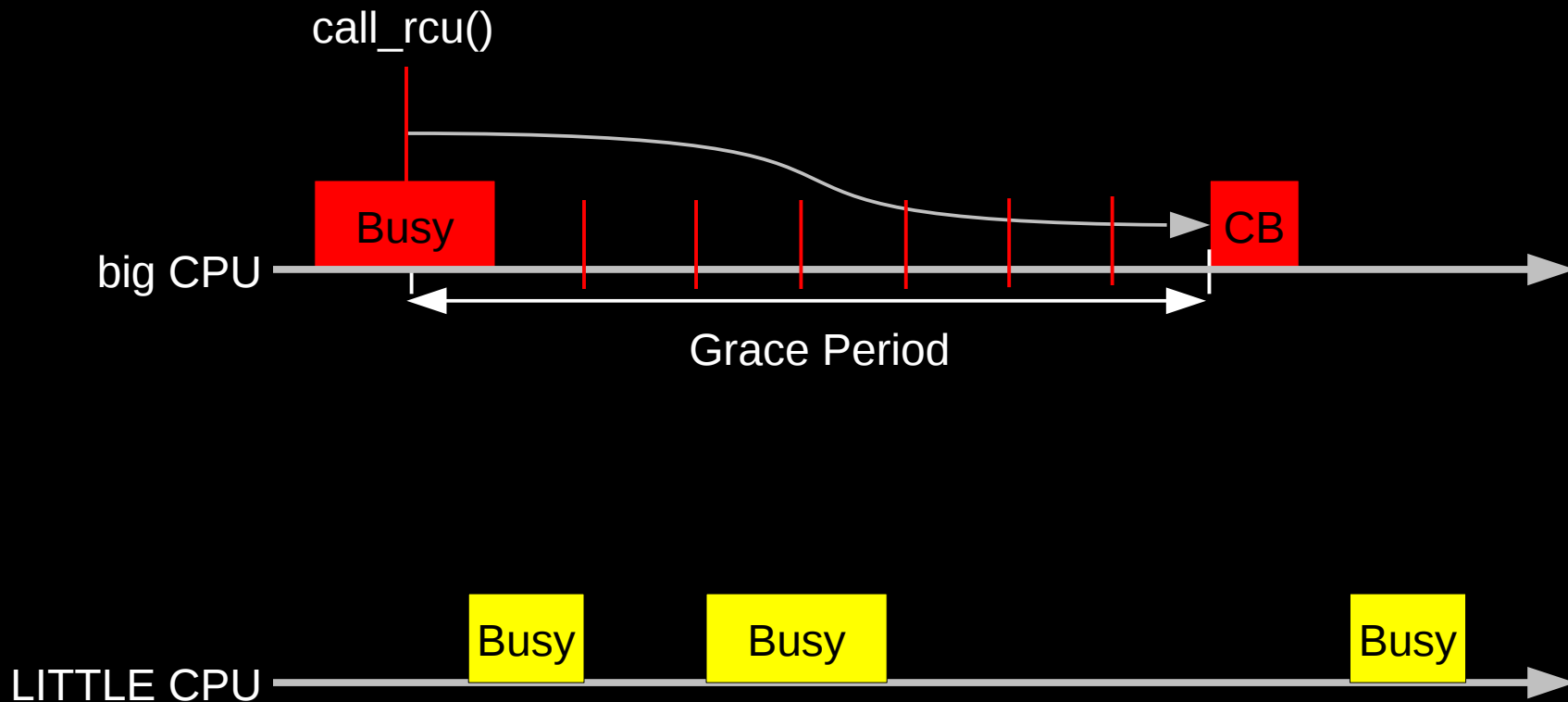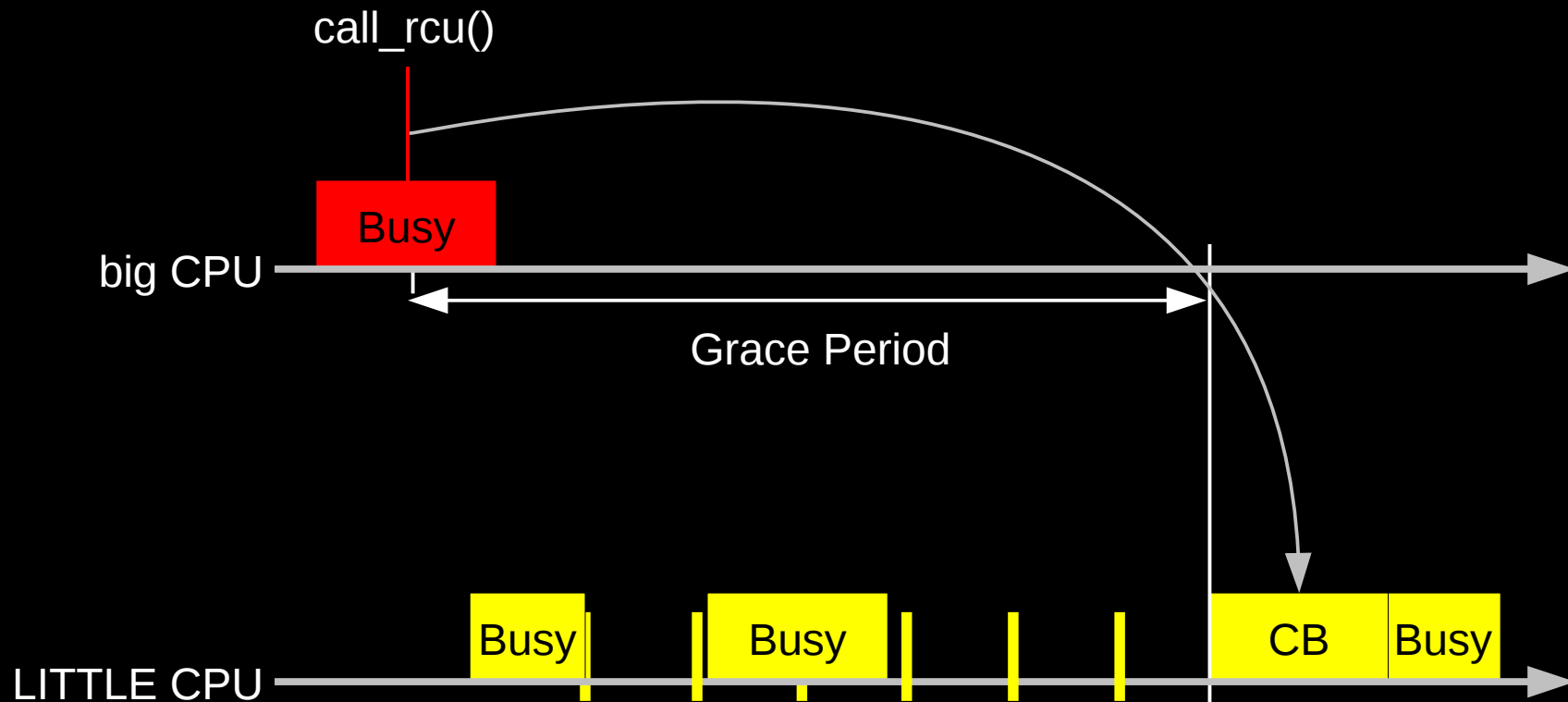
# ARM big.LITTLE With RCU Callback Offloading

# ARM big.LITTLE With RCU Callback Offloading

# ARM big.LITTLE With RCU Callback Offloading



call_rcu()

big CPU

Busy

CB

Grace Period

LITTLE CPU

Busy

Busy

CB

Busy

Slower...
But 3x better
energy efficiency

# ARM big.LITTLE With no-CBs CPUs: Preliminary Results (Randhawa and Eggemann, ARM)

- Reference System: No offloading

- Test System: big CPUs offloaded, kthreads on LITTLE CPUs

- Approximate power savings:
  - cyclictest: 10%
  - andebench8: 2%
  - audio: 10%
  - bbench_with_audio: 5%

# To Probe More Deeply Into Adaptive Idle

- Documentation/timers/NO_HZ.txt

- "The 2012 realtime minisummit" (LWN, CPU isolation discussion)
  - http://lwn.net/Articles/520704/

- "Interruption timer périodique" (Kernel Recipes, in French)
  - https://kernel-recipes.org/?page_id=410

- "What Is New In RCU for Real Time" (RTLWS 2012)
  - http://www.rdrop.com/users/paulmck/realtime/paper/RTLWS2012occcRT.2012.10.19e.pdf
    - Slides 31-32

- "TODO"
  - https://github.com/fweisbec/linux-dynticks/wiki/TODO

- "NoHZ tasks" (LWN)
  - http://lwn.net/Articles/420544/

IBM

# To Probe More Deeply Into RCU Callback Offloading

- "Making RCU Respect Your Device's Battery Lifetime: On-The-Job Energy-Efficiency Training For RCU Maintainers" (LCA 2013)
  - http://www.rdrop.com/users/paulmck/realtime/paper/RCUbattery.2013.01.30b.LCA.pdf

- "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

# Configuration Cheat Sheet

- CONFIG_NO_HZ_FULL=y Kconfig: enable adaptive ticks
  - Implies CONFIG_NO_HZ=y
  - "full_nohz=" boot parameter: Specify adaptive-tick CPUs
    - "full_nohz=1,3-7" says CPUs 1, 3, 4, 5, 6, and 7 are adaptive-tick
    - Omitting this parameter means no CPUs are adaptive-tick CPUs
    - Remember to leave at least one CPU for timekeeping!
  - PMQOS to reduce idle-to-nonidle latency
    - X86 can also use "idle=mwait" and "idle=poll" boot parameters, but note that these can cause thermal problems and degrade energy efficiency, especially "idle=poll"

- CONFIG_RCU_NOCB_CPU=y Kconfig: enable RCU offload
  - Specify which CPUs to offload at build time:
    - RCU_NOCB_CPU_NONE=y Kconfig: No offloaded CPUs
    - RCU_NOCB_CPU_ZERO=y Kconfig: Offload CPU 0
    - RCU_NOCB_CPU_ALL=y Kconfig: Offload all CPUs
  - "rcu_nocbs=" boot parameter: Specify additional offloaded CPUs

# Summary

- General-purpose OS or bare-metal performance?
  - Why not both?
  - Work in progress gets us very close for CPU-bound workloads:
    - Adaptive idle userspace execution (work in progress)
    - RCU callback offloading (early version in mainline)
    - Interrupt, process, daemon, and kthread affinity
    - Timer offloading
  - Some restrictions:
    - Need to reserve CPU(s) for housekeeping
    - Adaptive-idle and RCU-callback-offloaded CPUs specified at boot time
    - One task per CPU for adaptive-idle usermode execution
    - Global TLB-flush IPIs, cache misses, and TLB misses are still with us
  - Serendipity: Energy-efficiency benefits as well!

# Summary

- General-purpose OS or bare-metal performance?
  - Why not both?
  - Work in progress gets us very close for CPU-bound workloads:
    - Adaptive idle userspace execution (work in progress)
    - RCU callback offloading (early version in mainline)
    - Interrupt, process, daemon, and kthread affinity
    - Timer offloading
  - Some restrictions:
    - Need to reserve CPU(s) for housekeeping
    - Adaptive-idle and RCU-callback-offloaded CPUs specified at boot time
    - One task per CPU for adaptive-idle usermode execution
    - Global TLB-flush IPIs, cache misses, and TLB misses are still with us
  - Serendipity: Energy-efficiency benefits as well!

- Extending Linux's reach farther into extreme computing!!!

65

# 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?