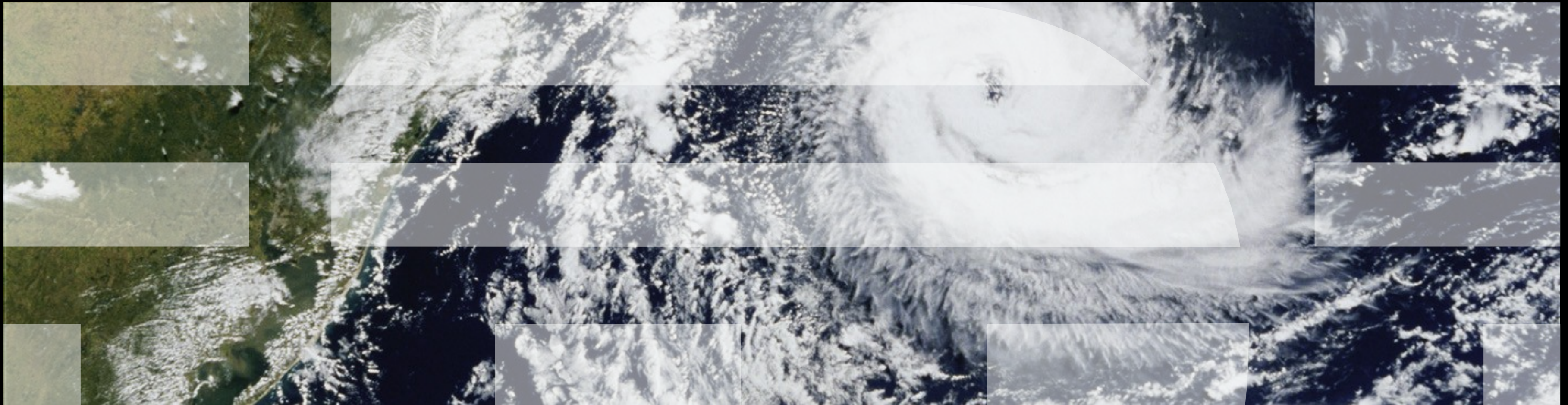
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center
Member, IBM Academy of Technology
linux.conf.au, Auckland, New Zealand, January 15, 2015

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

(Adventures in Ubiquity)



© 2015 IBM Corporation

IBM

Group Effort: NO_HZ_FULL Acknowledgments

- Josh Triplett: First prototype (LPC 2009)
- Frederic Weisbecker: Core kernel work and x86 port
- Steven Rostedt: Lots of code review and comments, tracing upgrades
- Christoph Lameter: Early adopter feedback, vmstat kthread fixes
- Li Zhong: Power port
- Geoff Levand, Kevin Hilman: ARM port, with Kevin continuing on with residual-tick-elimination work
- Peter Zijlstra: Scheduler-related review, comments, and work
- Paul E. McKenney: Read-copy update (RCU) work, full-system idle
- Thomas Gleixner, Paul E. McKenney: “Godfathers”
- Ingo Molnar: Maintainer
- Other contributors:
  - Avi Kivity, Chris Metcalf, Gilad Ben Yossef, Hakan Akkan, Lai Jiangshan, Max Krasnyansky, Namhyung Kim, Paul Gortmaker, Paul Mackerras, Zen Lin, Jason Furmanek, Paul Clarke, Mala Anand, Mike Galbraith, Oleg Nesterov, Viresh Kumar, Rik van Riel, Dave Jones, Andy Lutomirski, Russell King, Oleg Nesterov, Nicolas Pitre, Sasha Levin, Lai Jiangshan, Andrew Morton, Tejun Heo, Hugh Dickins, Breno Leitao, Srivatsa Bhat, Stephen Warren, Mike Galbraith, Steve Rostedt, Henrik Austad, and many others

© 2015 IBM Corporation

Overview

- Back in the old days...
- From dyntick-idle to NO_HZ_FULL
- Achieving ubiquity: Two of my adventures
- Lessons (Re)Learned
- Additional NO_HZ_FULL issues
- Cheat sheets

© 2015 IBM Corporation

Back In The Old Days...

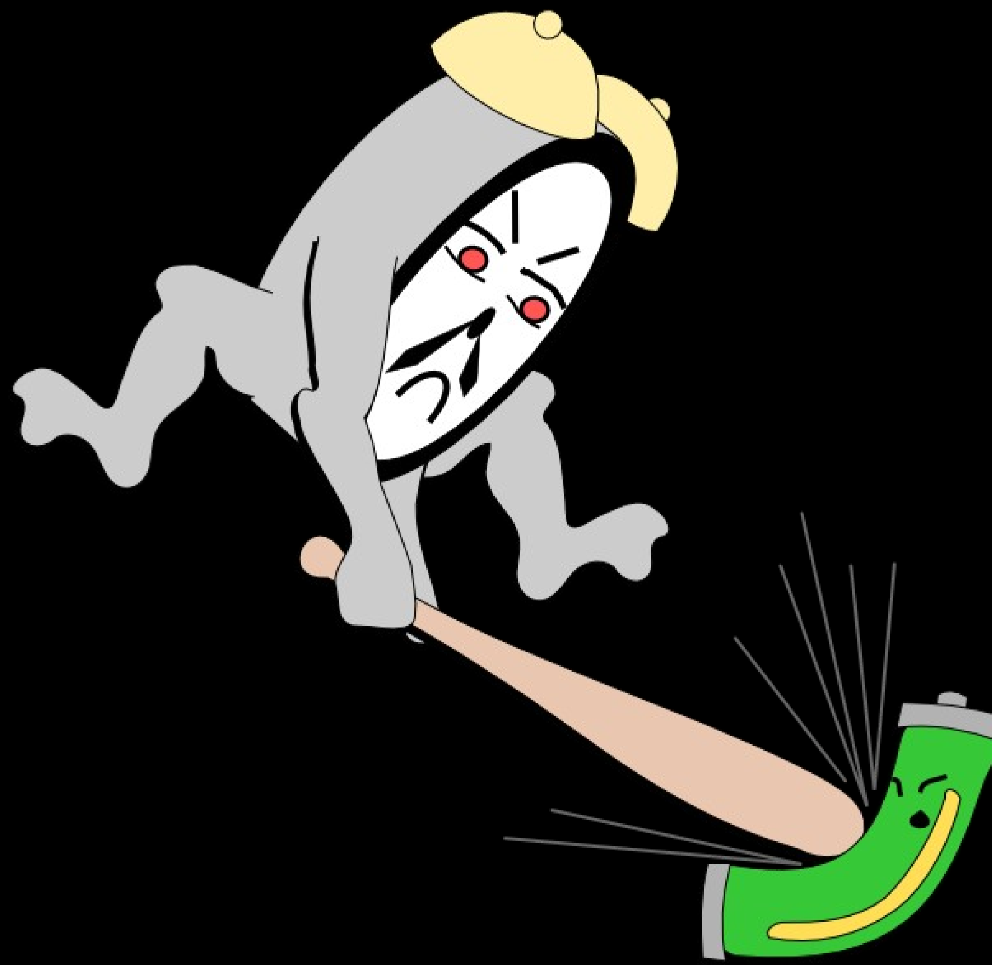
© 2015 IBM Corporation

Back In The Old Days...

- Mainstream CPUs had no energy-efficiency features
  - Which meant that idle state was often the least energy efficient
  - No cache misses, so full utilization of the power-hungry ALU
  - Which meant that scheduling-clock interrupts to idle CPUs actually *improved* energy efficiency
- Things have changed: Idle often means powered-off CPU
  - So scheduling-clock interrupts to an idle CPU are now very bad
    - Especially on battery-powered systems!!!

© 2015 IBM Corporation

There Used To Be Things You Could Count On...



© 2015 IBM Corporation

There Used To Be Things You Could Count On...

© 2015 IBM Corporation

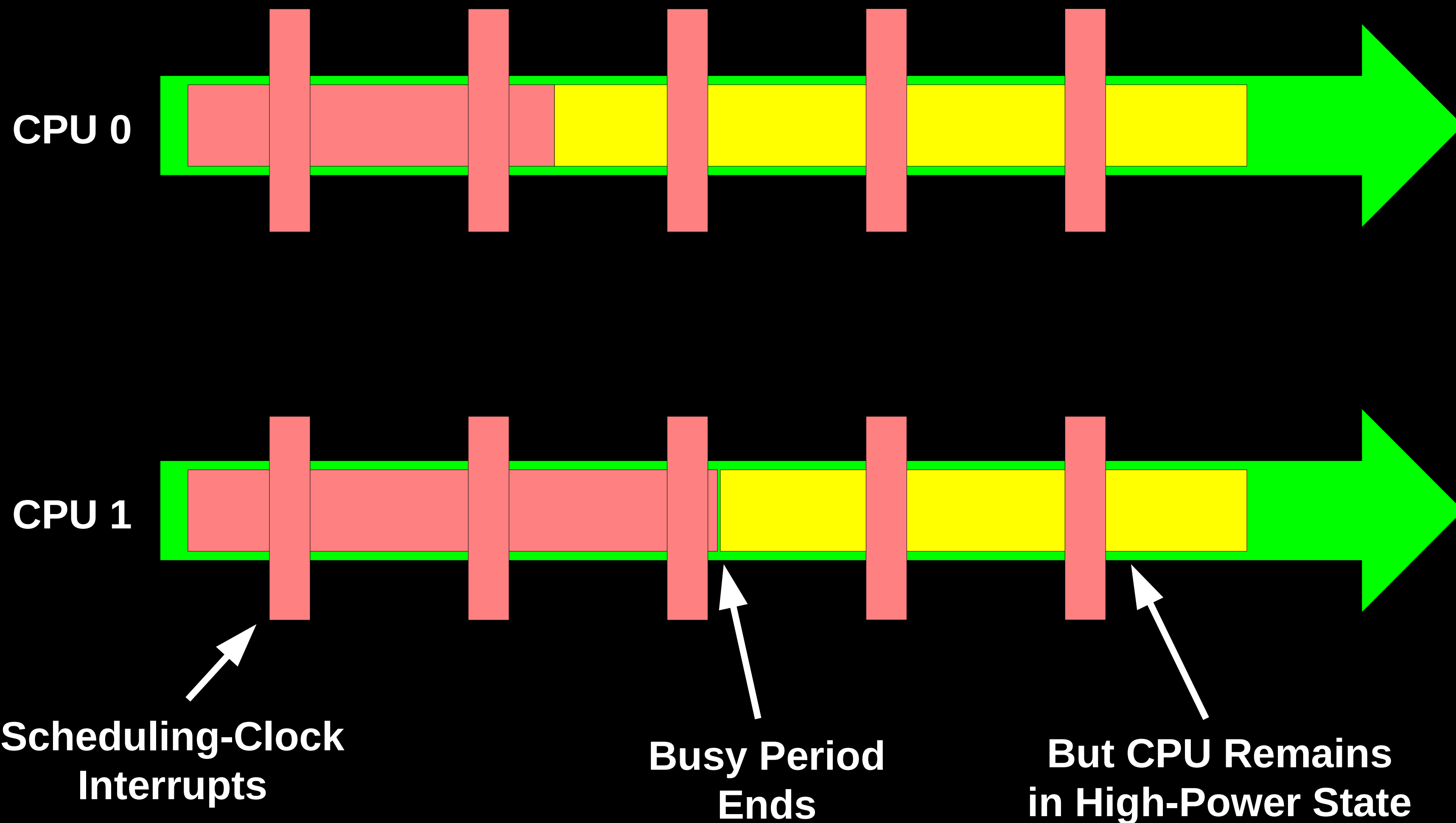
IBM

What We Need Instead...

© 2015 IBM Corporation

IBM

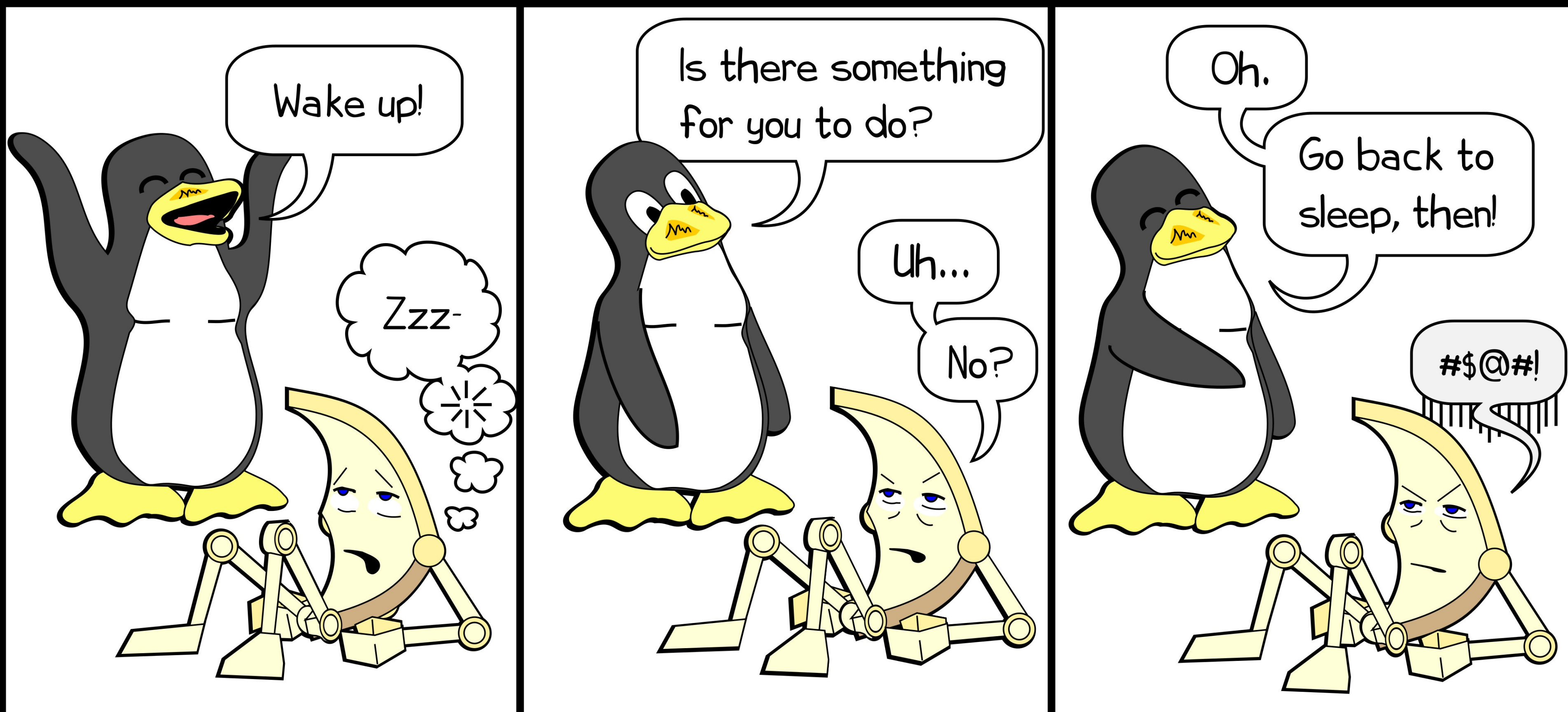
Before Linux's dyntick-idle System



© 2015 IBM Corporation

IBM

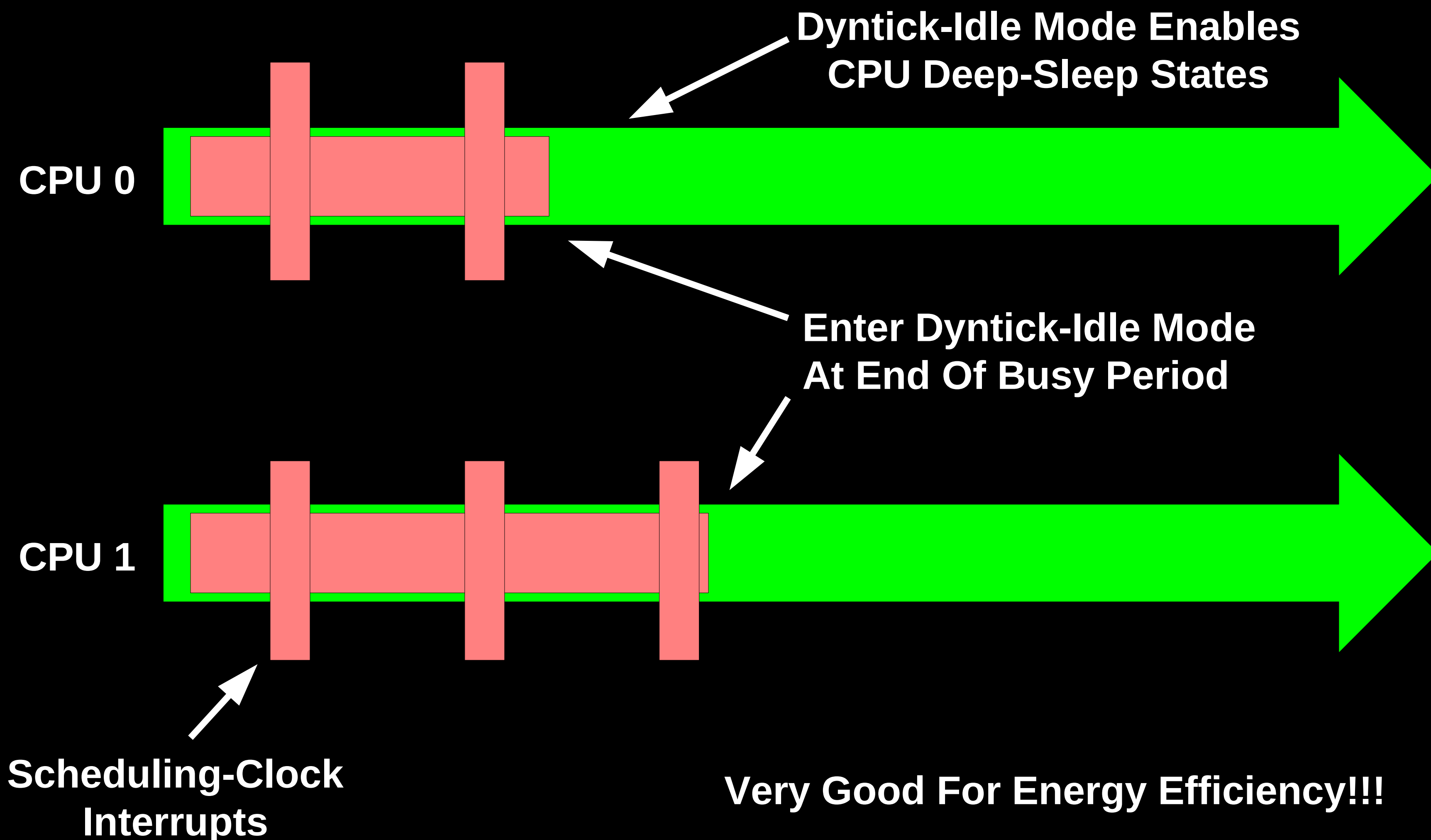
Before Linux's dyntick-idle System



© 2015 IBM Corporation

IBM

Linux's dyntick-idle System: NO_HZ



Very Good For Energy Efficiency!!!

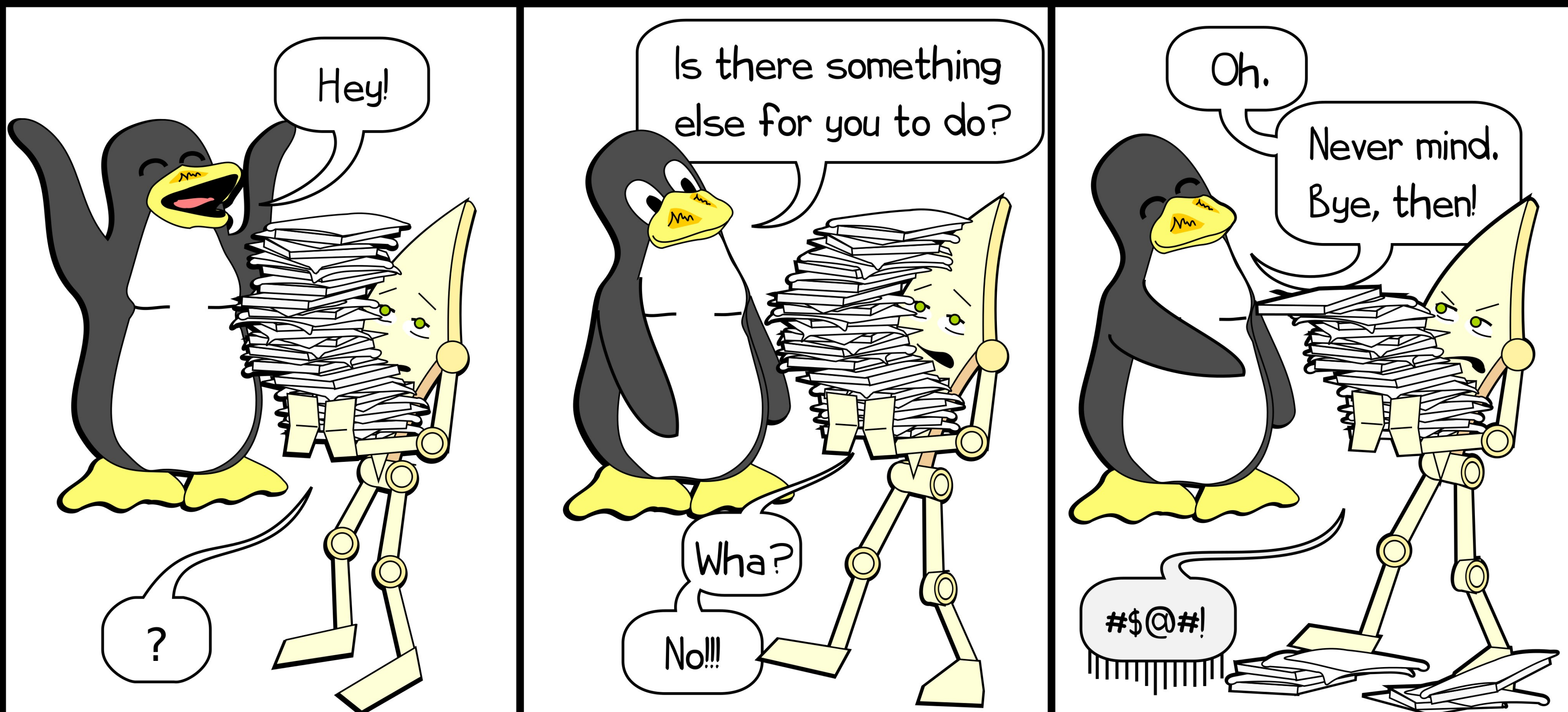
© 2015 IBM Corporation

Also: Avoid Unnecessary Usermode Interrupts

- HPC and real-time applications can increase performance if unnecessary scheduling-clock interrupts are omitted
- And if there is only one runnable task on a given CPU, why interrupt it?
- If another task shows up, ***then*** we can interrupt the CPU
- Until then, interrupting it only slows it down

© 2015 IBM Corporation

Also: Avoid Unnecessary Usermode Interrupts



© 2015 IBM Corporation

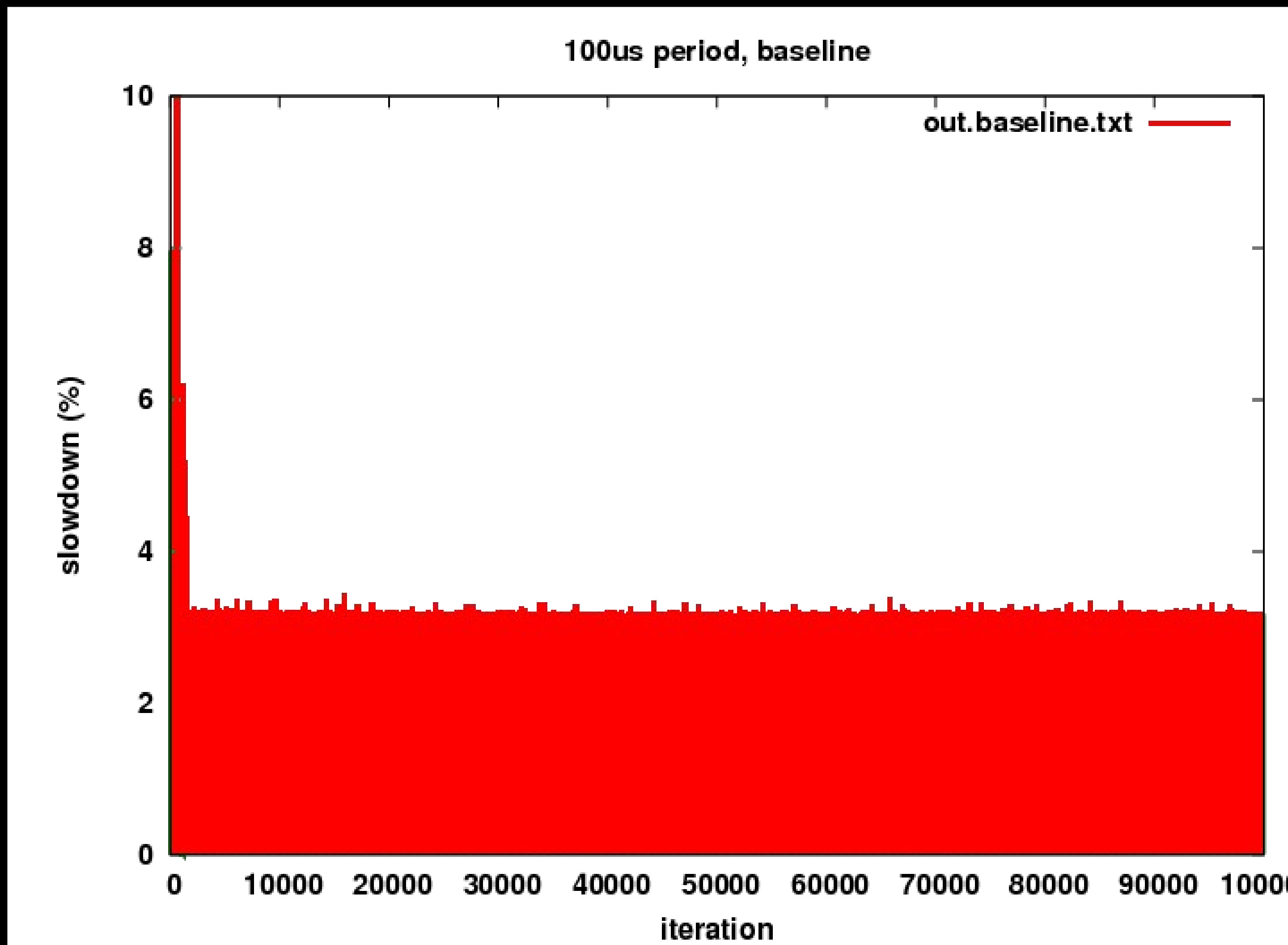
Also: Avoid Unnecessary Usermode Interrupts

- HPC and real-time applications can increase performance if unnecessary scheduling-clock interrupts are omitted
- And if there is only one runnable task on a given CPU, why interrupt it? All that will do is slow things down!!!
- If another task shows up, ***then*** we can interrupt the CPU
- Josh Triplett prototyped CONFIG_NO_HZ_FULL in 2009

© 2015 IBM Corporation

IBM

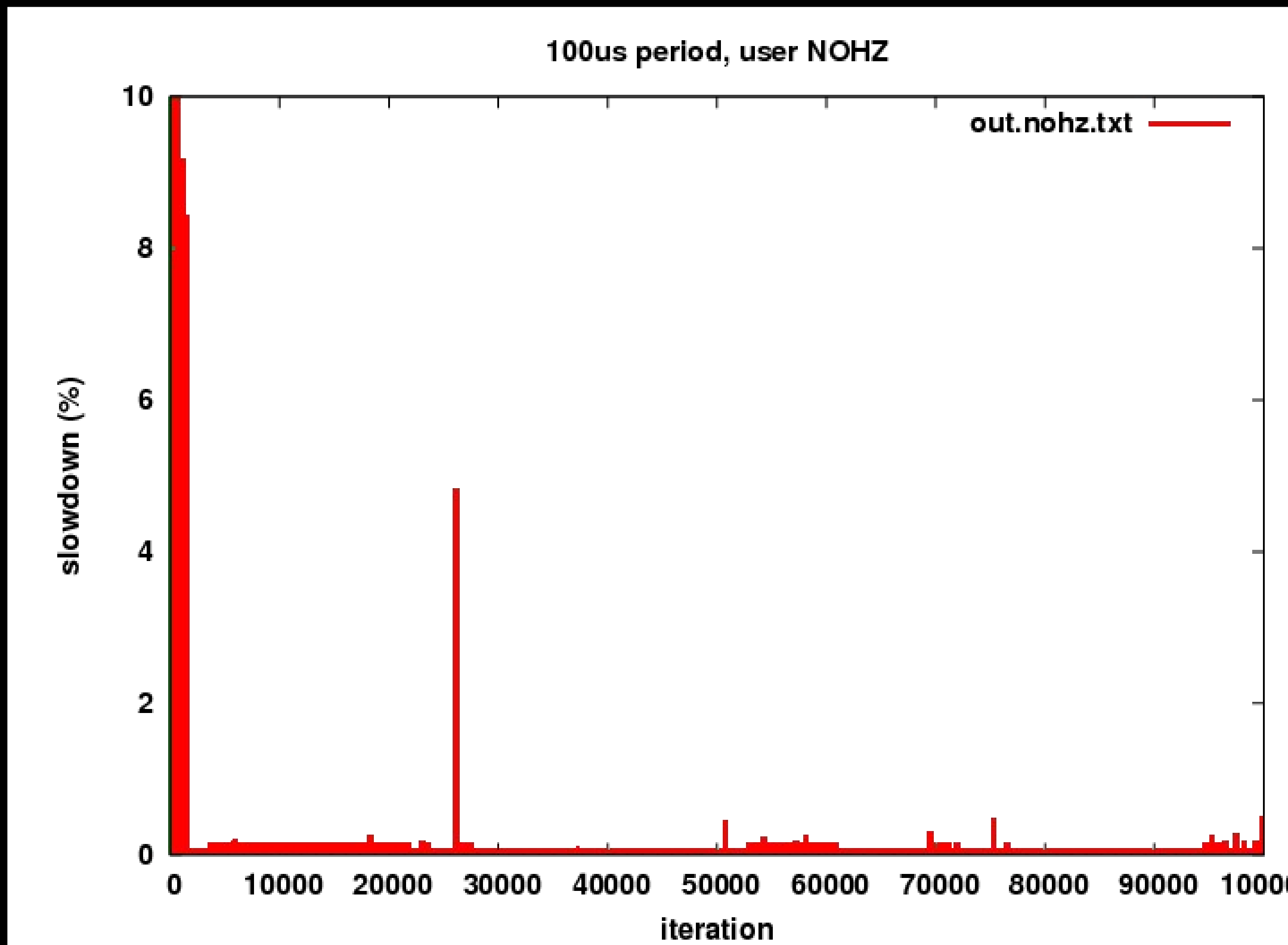
Benchmark Results Before (Anton Blanchard)



© 2015 IBM Corporation

IBM

Benchmark Results After (Anton Blanchard)



Well worth going after...

© 2015 IBM Corporation

IBM

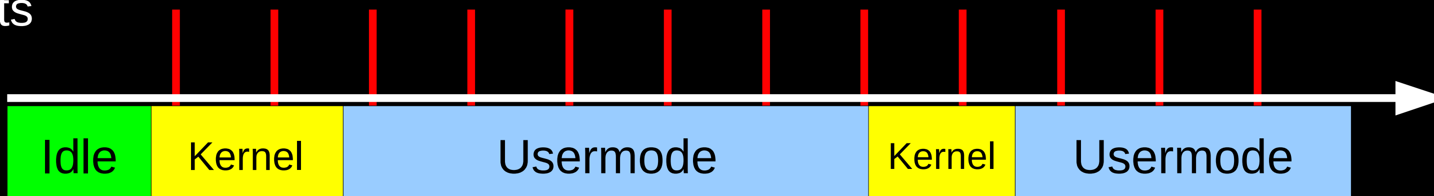
But There Were A Few Small Drawbacks...

- 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 ticks 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-ticks usermode execution so that it ignores adaptive-ticks user-mode CPUs, similar to its handling of dyntick-ticks CPUs
- Frederic Weisbecker took on the task of fixing this (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
  - We now have NO_HZ_FULL!

© 2015 IBM Corporation

How Well Does NO_HZ_FULL Work?

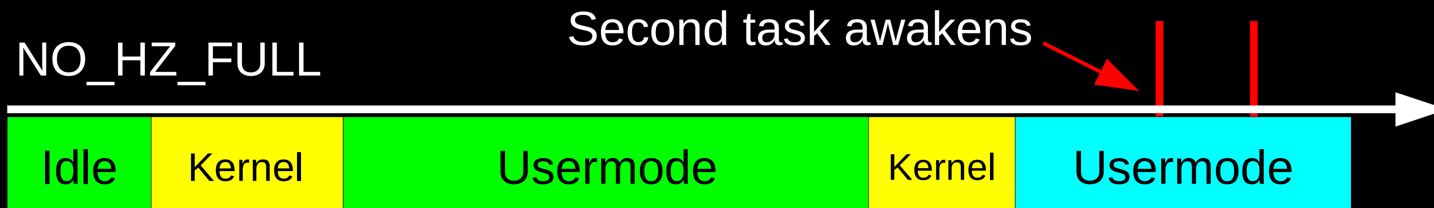
Scheduling clock interrupts

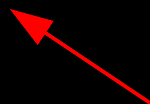


NO_HZ

NO_HZ_FULL

Second task awakens





One task per CPU

© 2015 IBM Corporation

IBM

Accepted Into Linux-Kernel Mainline v3.10

© 2015 IBM Corporation

Accepted Into Linux-Kernel Mainline v3.10
Enabled by Default in RHEL7

© 2015 IBM Corporation

IBM

Accepted Into Linux-Kernel Mainline v3.10
Enabled by Default in RHEL7

Thus Used By Everyone, Not Just HPC and RT

And So The *Real* Validation Begins!!!

© 2015 IBM Corporation

Rik van Riel: rcu_sched at More Than 40% CPU!!!

© 2015 IBM Corporation

IBM

Rik van Riel: rcu_sched at More Than 40% CPU!!!

Say *What*???

© 2015 IBM Corporation

Rik van Riel: rcu_sched at More Than 40% CPU!!!

- 80-CPU x86 system with NO_HZ_FULL
- Context-switch-heavy workload
  - Which NO_HZ_FULL was ***not*** optimized for!!!
- So maybe grace periods are completing very quickly, resulting in high CPU load on RCU's grace-period kthread
  - So artificially slow down the loop in the rcu_sched task

© 2015 IBM Corporation

IBM

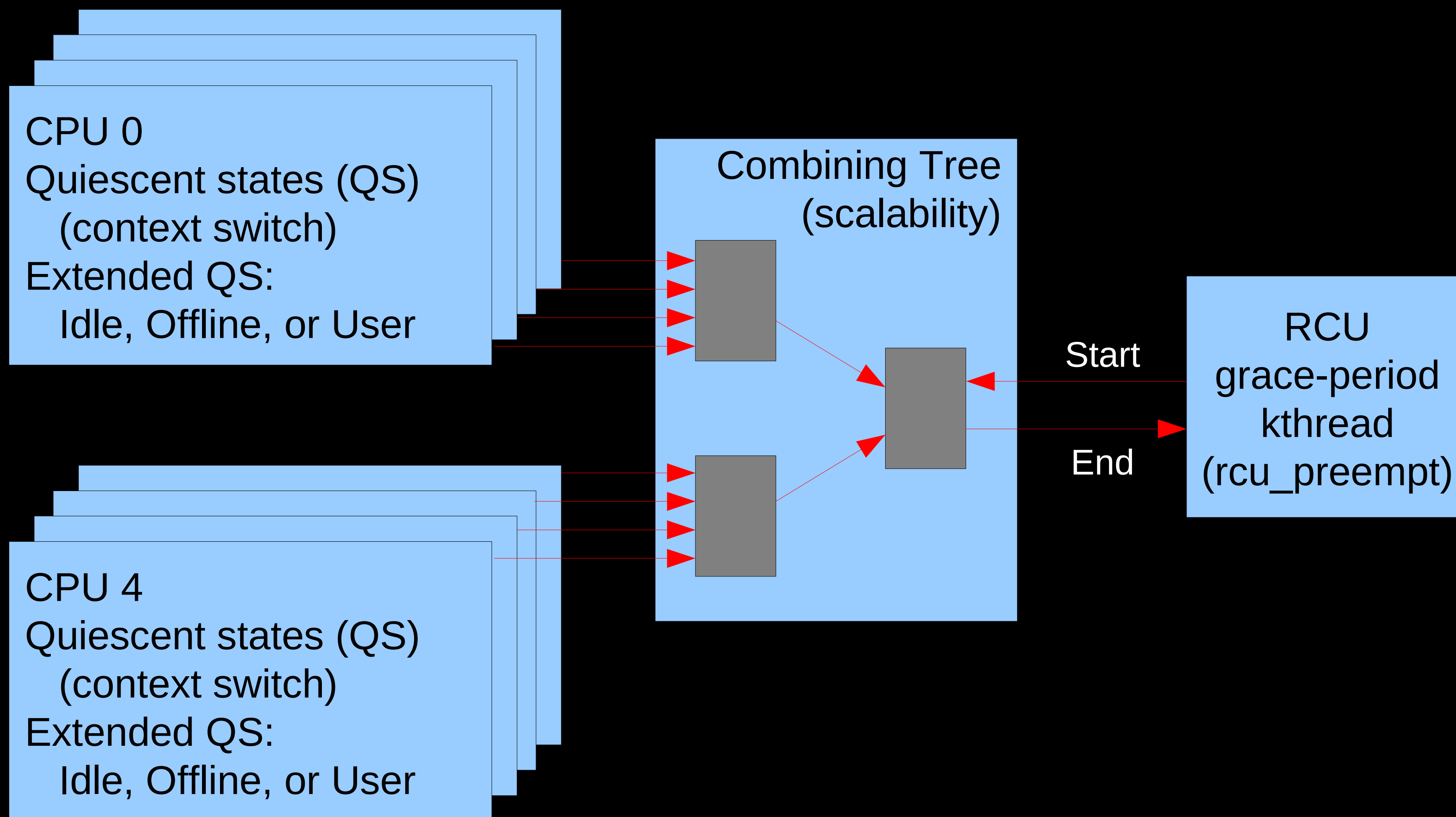
Rik van Riel: rcu_sched at More Than 40% CPU!!!

- 80-CPU x86 system with NO_HZ_FULL
- Context-switch-heavy workload
  - Which NO_HZ_FULL was ***not*** optimized for!!!
- So maybe grace periods are completing very quickly, resulting in high CPU load on RCU's grace-period kthread
  - So artificially slow down the loop in the rcu_sched task
  - Which unfortunately doesn't help
  - So need to actually analyze the problem! :-)

© 2015 IBM Corporation

IBM

Rough Diagram of Vanilla RCU Components Involved

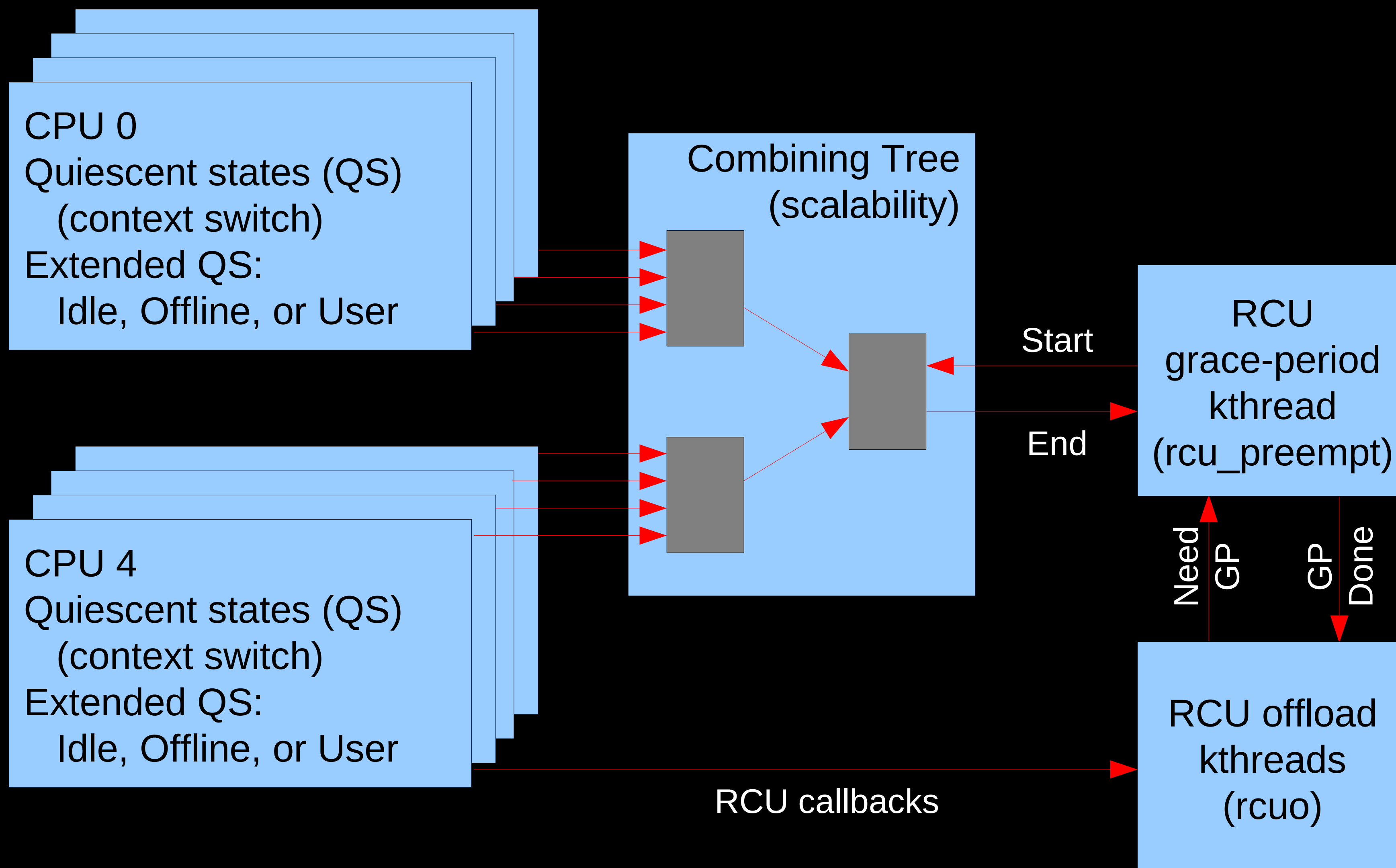


Low rcu_preempt CPU utilization

© 2015 IBM Corporation

IBM

Rough Diagram of RCU Components With Bare Metal

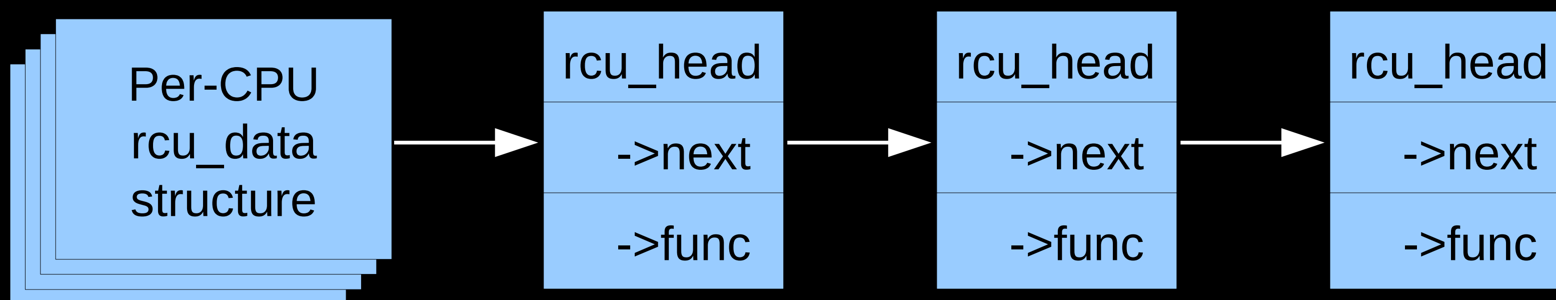


Excessive rcu_preempt CPU utilization

© 2015 IBM Corporation

But Why Offload RCU Callbacks???

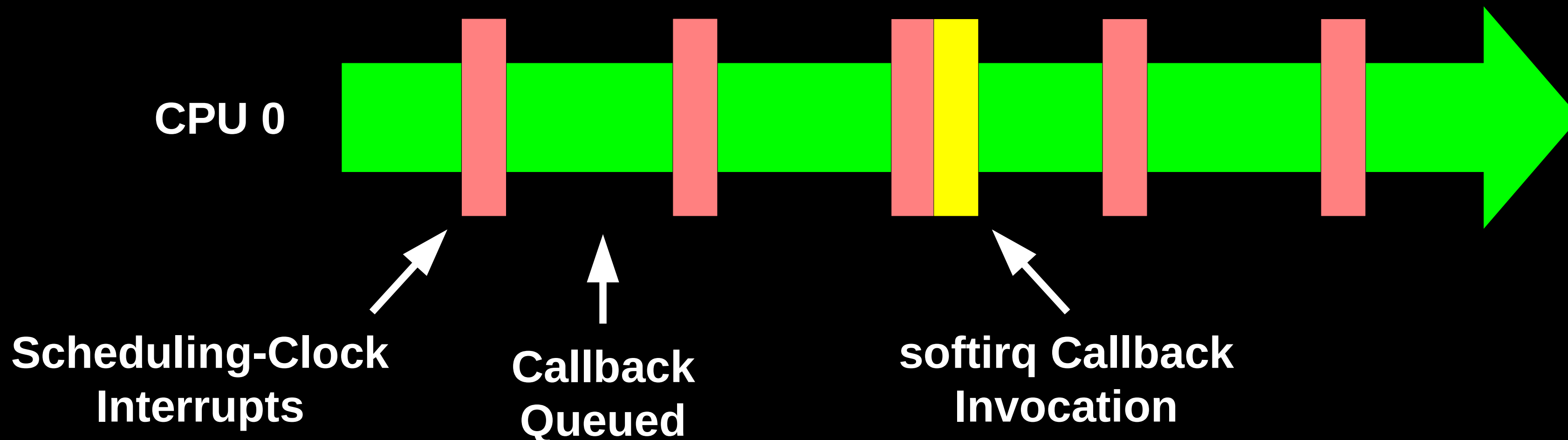
- But first, what are RCU callbacks and why are they needed?
  - For an overview, see http://lwn.net/Articles/262464/
- For the purposes of this presentation, think of RCU as something that defers work, with one work item per callback
  - Each callback has a function pointer and an argument
  - Callbacks 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
  - Allow extremely fast and scalable read-side access to shared data



© 2015 IBM Corporation

How Are RCU Callbacks Invoked?

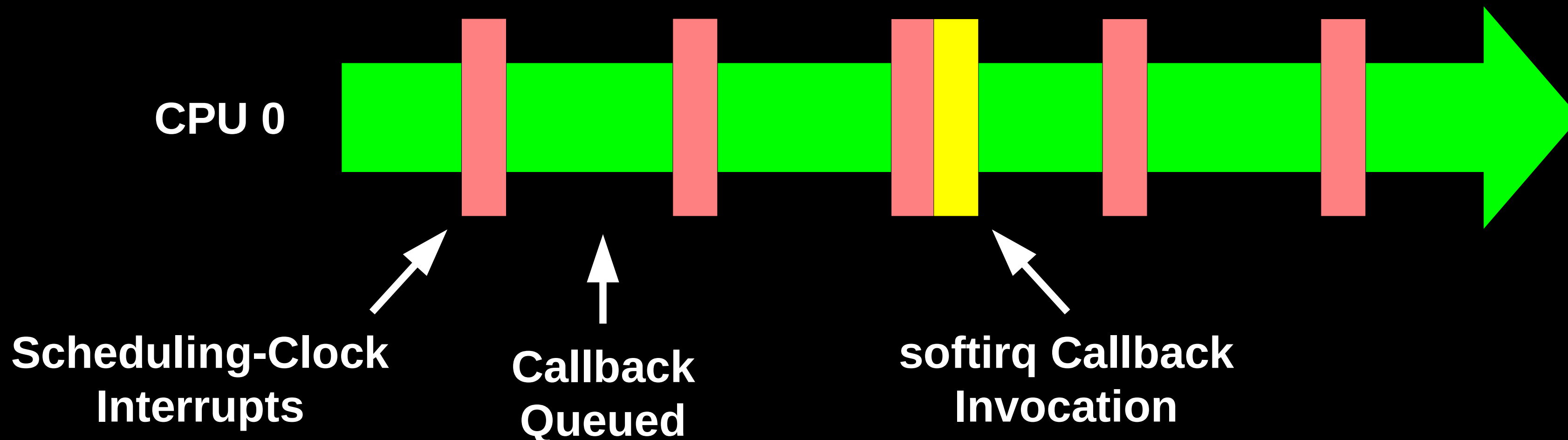
- RCU uses a state machine driven out of the scheduling-clock interrupt to determine when it is safe to invoke callbacks
- Actual callback invocation is done from softirq
- RCU: Tapping the awesome power of procrastination for more than two decades!!!



© 2015 IBM Corporation

How Are RCU Callbacks Invoked?

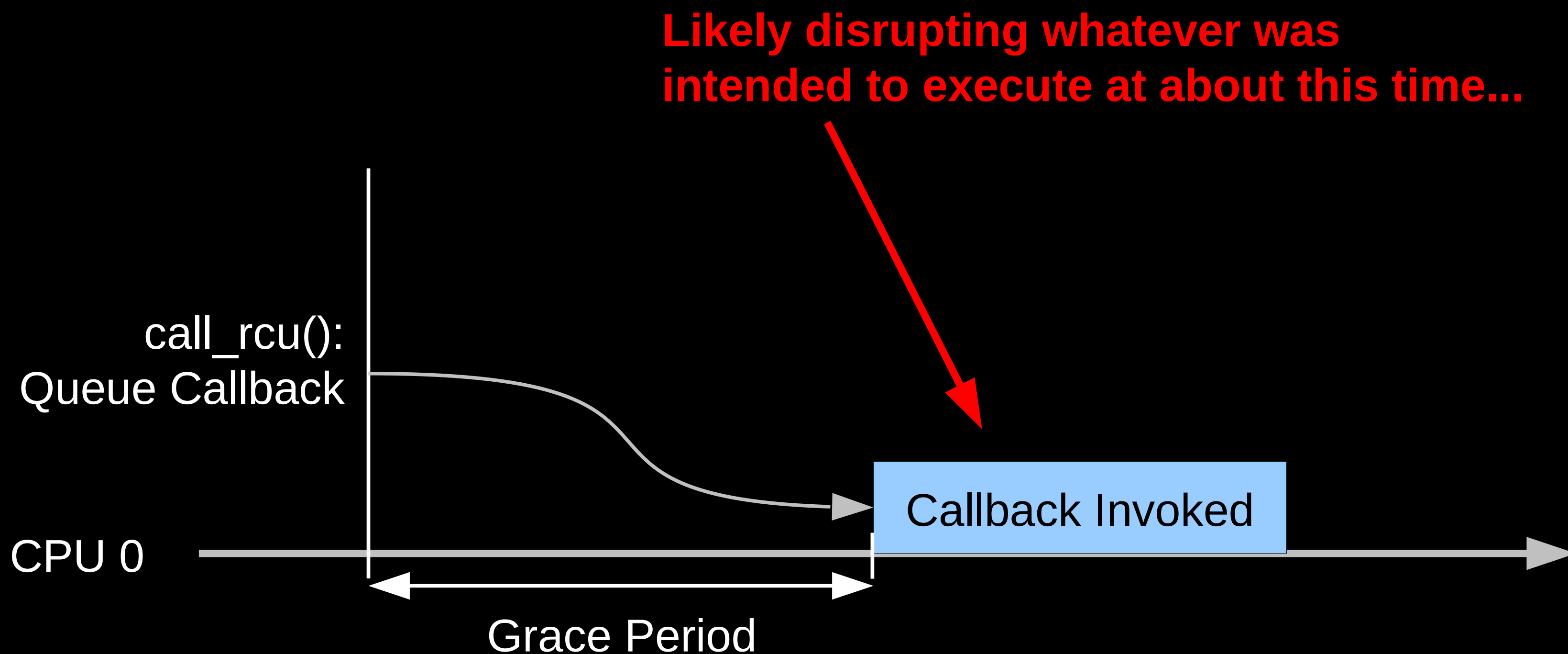
- RCU uses a state machine driven out of the scheduling-clock interrupt to determine when it is safe to invoke callbacks
- Actual callback invocation is done from softirq
- RCU: Tapping the awesome power of procrastination for more than two decades!!! **But...**



© 2015 IBM Corporation

IBM

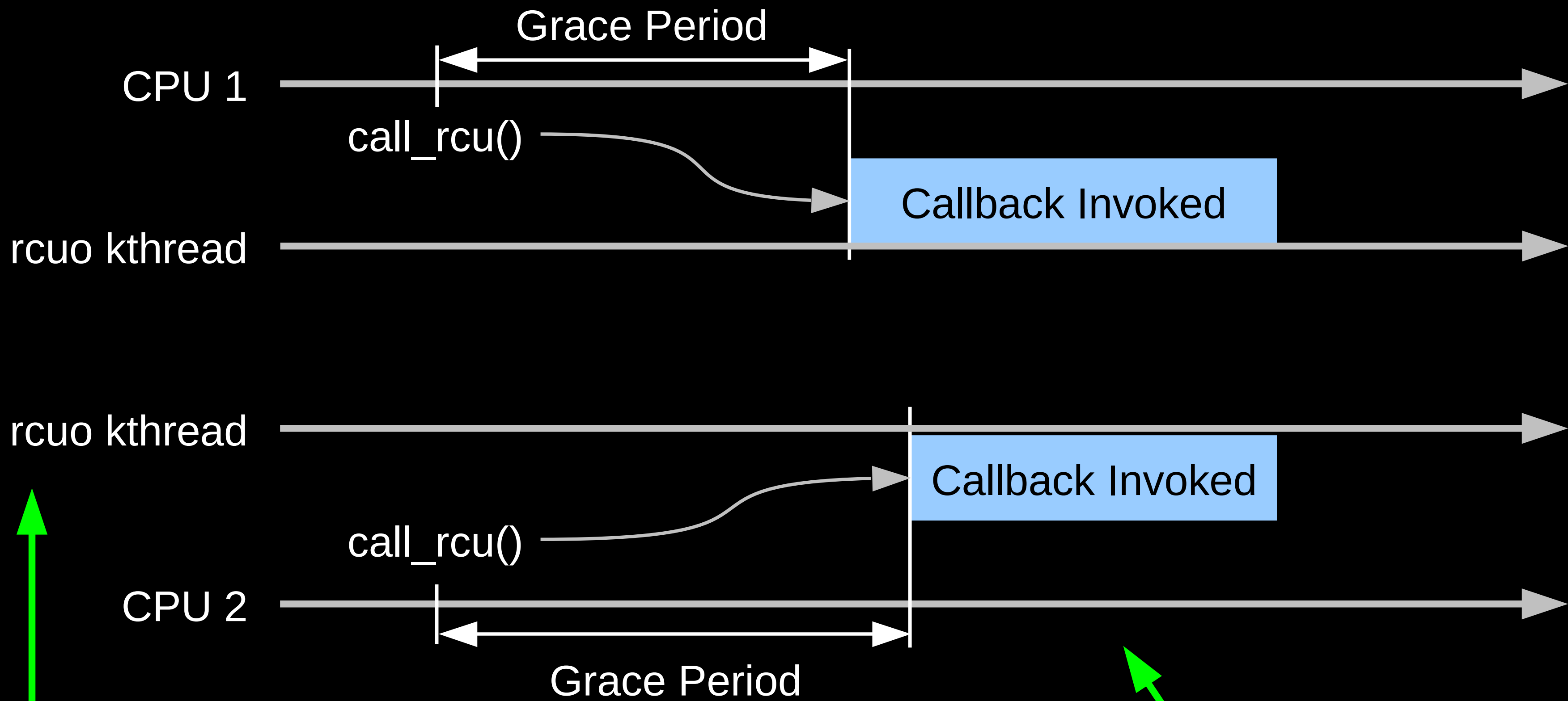
Procrastination's Dark Side: Eventually Must Do Work



© 2015 IBM Corporation

IBM

Offload RCU Callbacks: Inspired by Houston/Korty



Scheduler controls placement, or can place manually (NO_HZ_FULL now places on housekeeping CPUs)

No disruption!

© 2015 IBM Corporation

IBM

Reducing Disruption Great, But Not At 40% CPU!!!

© 2015 IBM Corporation

Reducing Disruption Great, But Not At 40% CPU!!! Especially Not For Users Not Caring About Disruption

Hey, I was hoping...

© 2015 IBM Corporation

First, Stop Bleeding For Innocent Bystanders!!!

- RCU's callback offloading is resulting in high CPU overhead for some workloads on large systems
- And Rik's setup enables callback offloading unconditionally
  - CONFIG_RCU_NOCB_CPU_ALL=y implied by CONFIG_NO_HZ_FULL=y, which is enabled by default
  - Each and every system uses expensive callback offloading, thus penalizing all users for something actually needed by only a few
  - Ubiquity strikes again!!!
- Initial bandaid:
  - Don't imply CONFIG_RCU_NOCB_CPU_ALL=y from CONFIG_NO_HZ_FULL=y
  - Offload only from CPUs specified by nohz_full= boot parameter
  - Stops the bleeding for normal users who don't care about bare metal
    - b58cc46c5f6b: Don't offload callbacks unless specifically requested)

© 2015 IBM Corporation

Industry Experience: 1 of 6 Fixes Introduces Bug

© 2015 IBM Corporation

Industry Experience: 1 of 6 Fixes Introduces Bug

- And this was in the “1” category!
- RCU is used ***very*** early in boot, including posting callbacks
  - Which was news to me, I figured call_rcu() didn't happen until later
    - And it didn't happen until later in ***my*** testing...
  - Ubiquity strikes again!!!
- Callbacks posted before RCU decides that a given CPU is to be offloaded are lost, which can result in hangs
  - Simple fix: decide earlier on which CPUs are to be offloaded
  - f4579fc57cf4: Fix attempt to avoid unsolicited offloading of callbacks
  - Thus stopping the bleeding caused by the earlier bandage...

© 2015 IBM Corporation

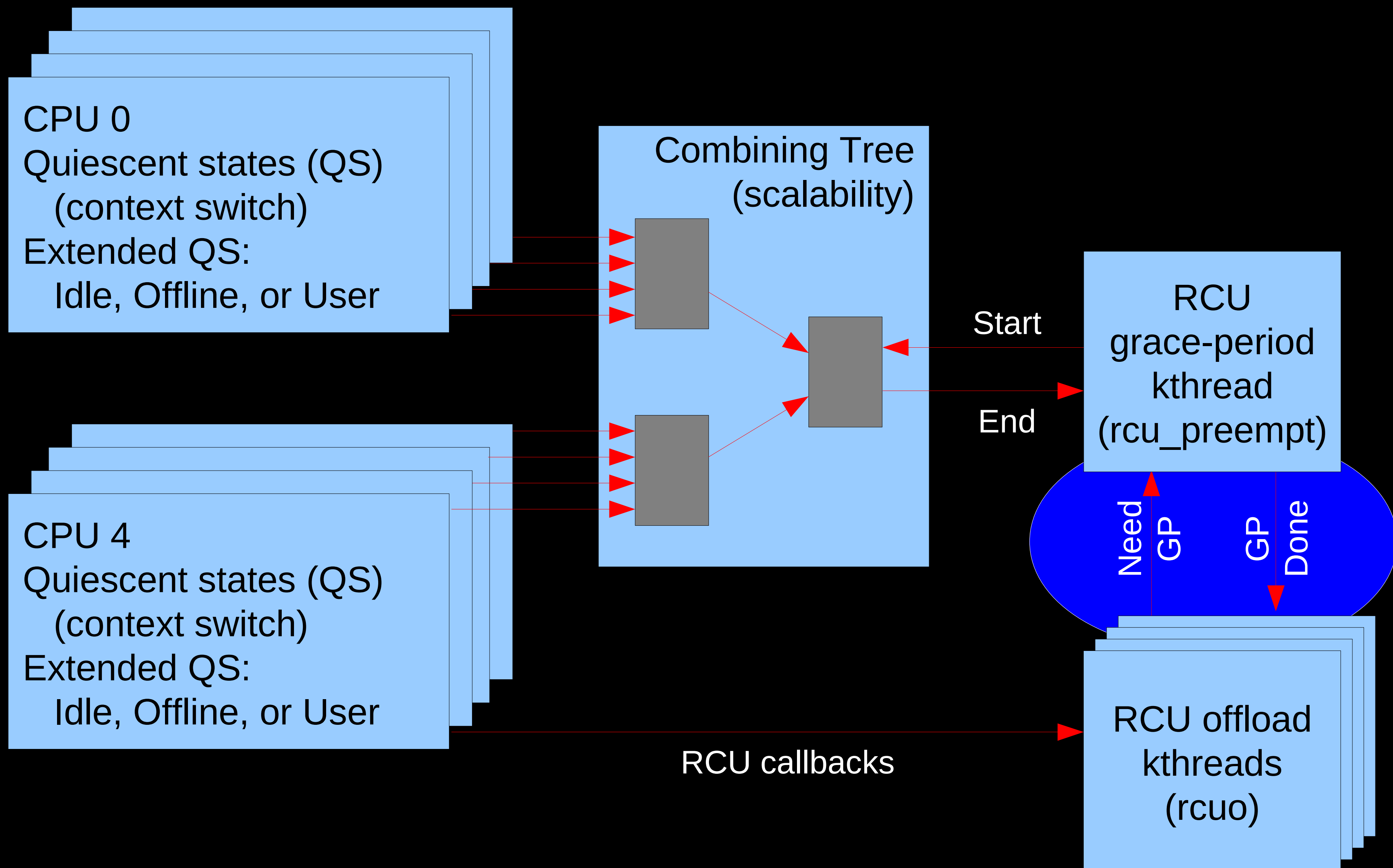
OK, Stopped the Bleeding, Now Fix the Bug!

- Rik van Riel determined CPU overhead due to grace-period kthread waking up rcuo callback-offload kthreads
- And this will just get worse with increasing numbers of CPUs
  - If wakeups on 80 CPUs consume 40% of a CPU, for 4096 CPUs wakeups will consume 2048% of the grace-period kthread's CPU
    - This will mean gross delays of grace-period completion, unacceptable
- One solution: Make some of the rcuo kthreads wake up the rest, thus spreading (AKA “hiding”) the overhead
  - And parallelizing the wakeups
  - fbce7497ee5a: Parallelize and economize NOCB kthread wakeups

© 2015 IBM Corporation

IBM

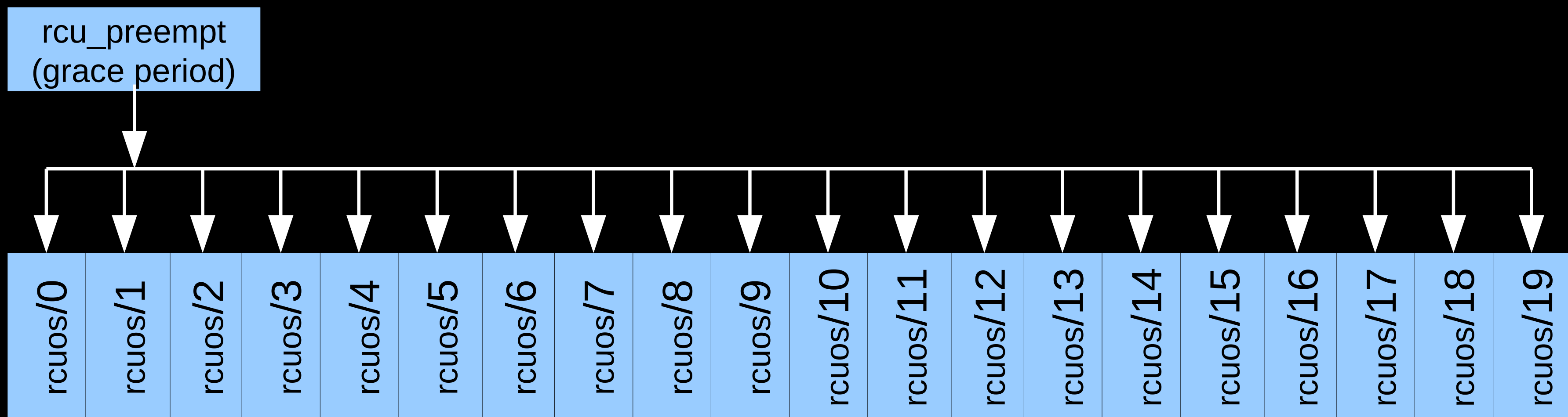
Back to the Rough Diagram of RCU Components



© 2015 IBM Corporation

IBM

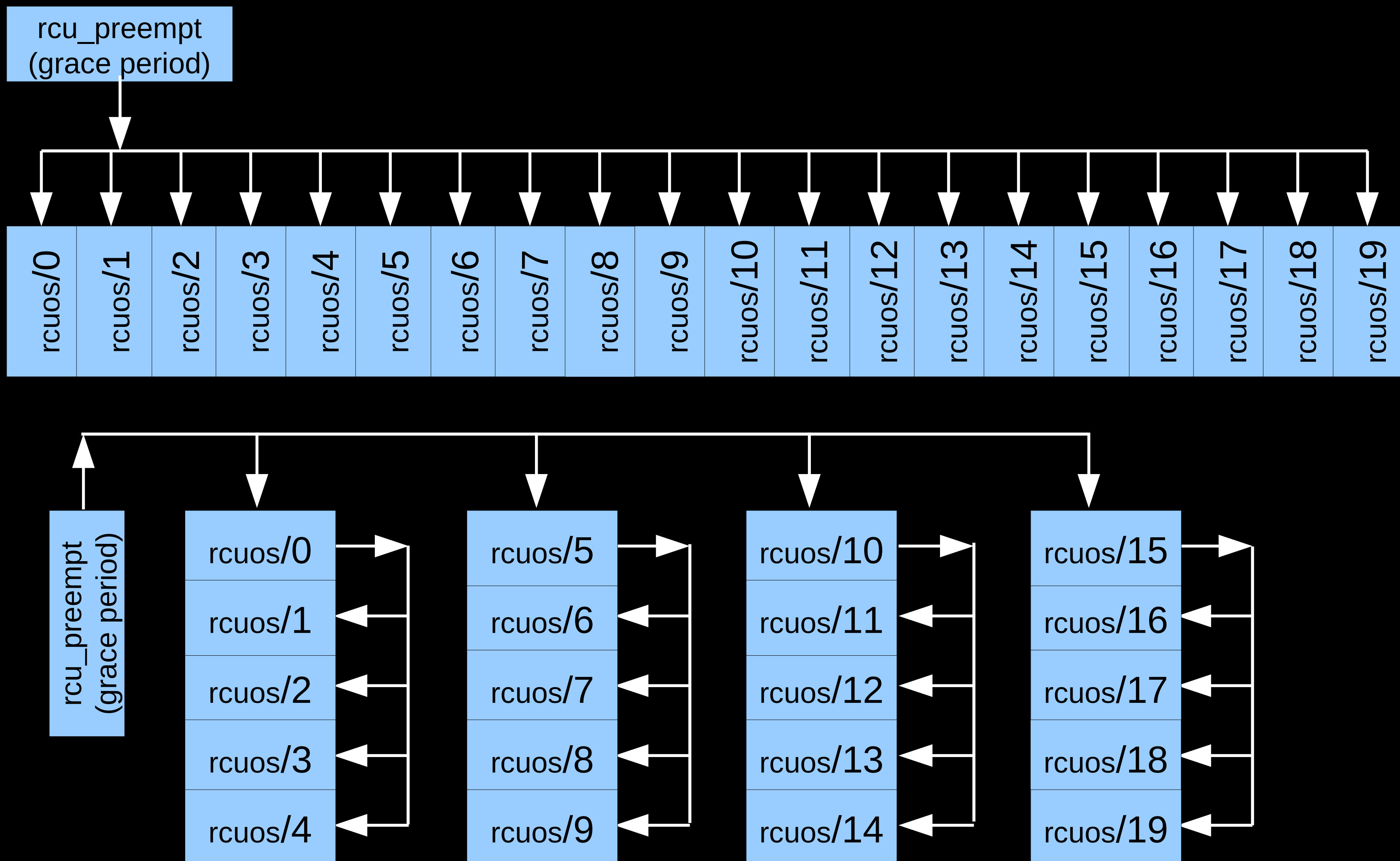
Rough Diagram Focusing on rcuo Wakeups, 20 CPUs



© 2015 IBM Corporation

IBM

Rough Diagram Focusing on rcuo Wakeups, 20 CPUs



© 2015 IBM Corporation

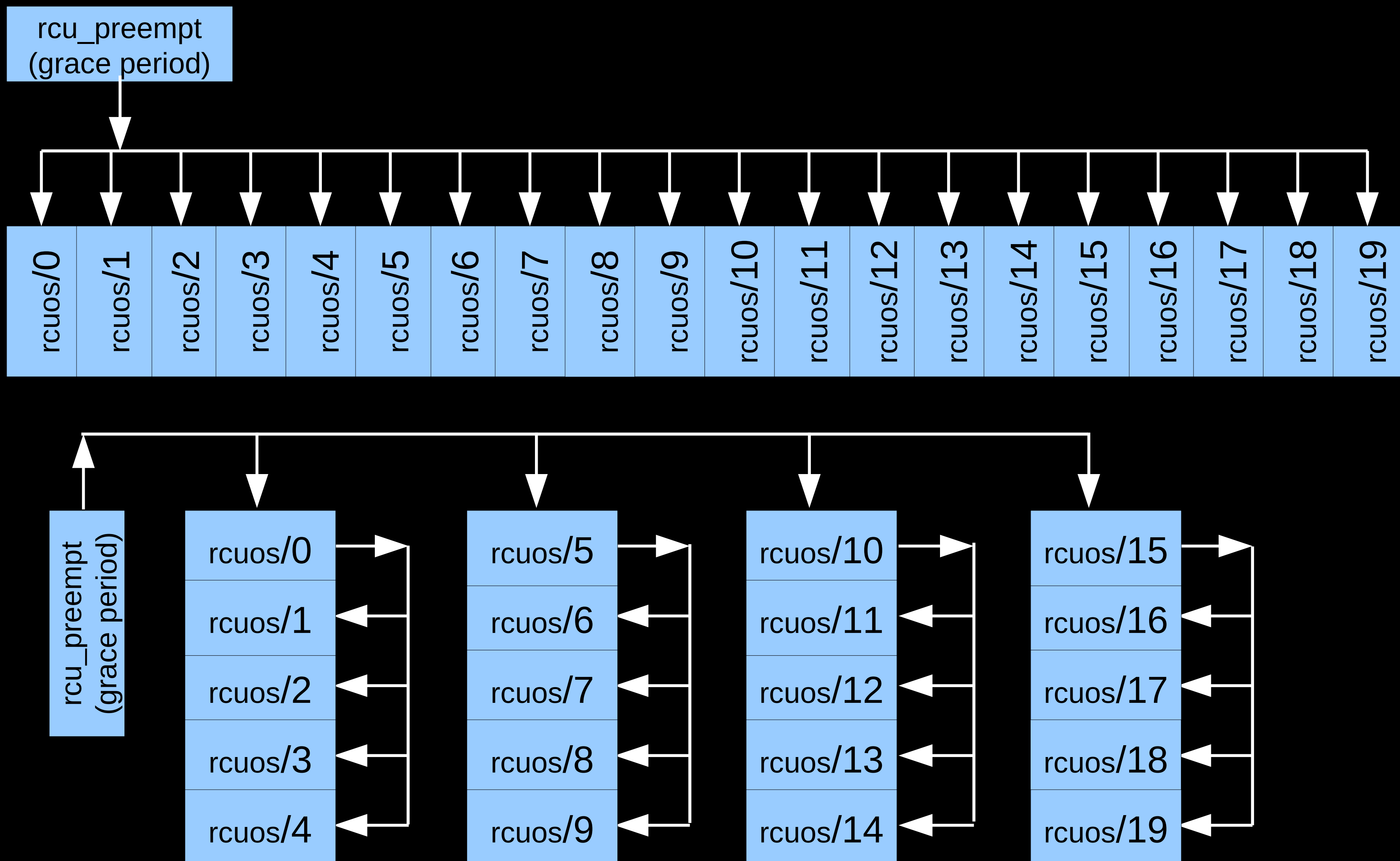
Serendipity: *Reduces* Number of Wakeups!

- Assume busy system, where each CPU has at least one callback per grace period
- Old way:
  - Each of rcuo/0 through rcuo/19 awakened when first callback posted
  - Each of rcuo/0 through rcuo/19 awakened when grace period ends
  - Total of 40 wakeups, 2 per rcuo kthread, 20 by grace-period kthread
- New way:
  - Each of rcuo/0, rcuo/5, rcuo/10, and rcuo/15 awakened when first callback posted
  - Each of rcuo/0 through rcuo/19 awakened when grace period ends
  - Total of only 24 wakeups, 4 by grace-period kthread
  - 40% reduction in total wakeups, and *80%* reduction in wakeups by grace-period kthread
- (See next slide)

© 2015 IBM Corporation

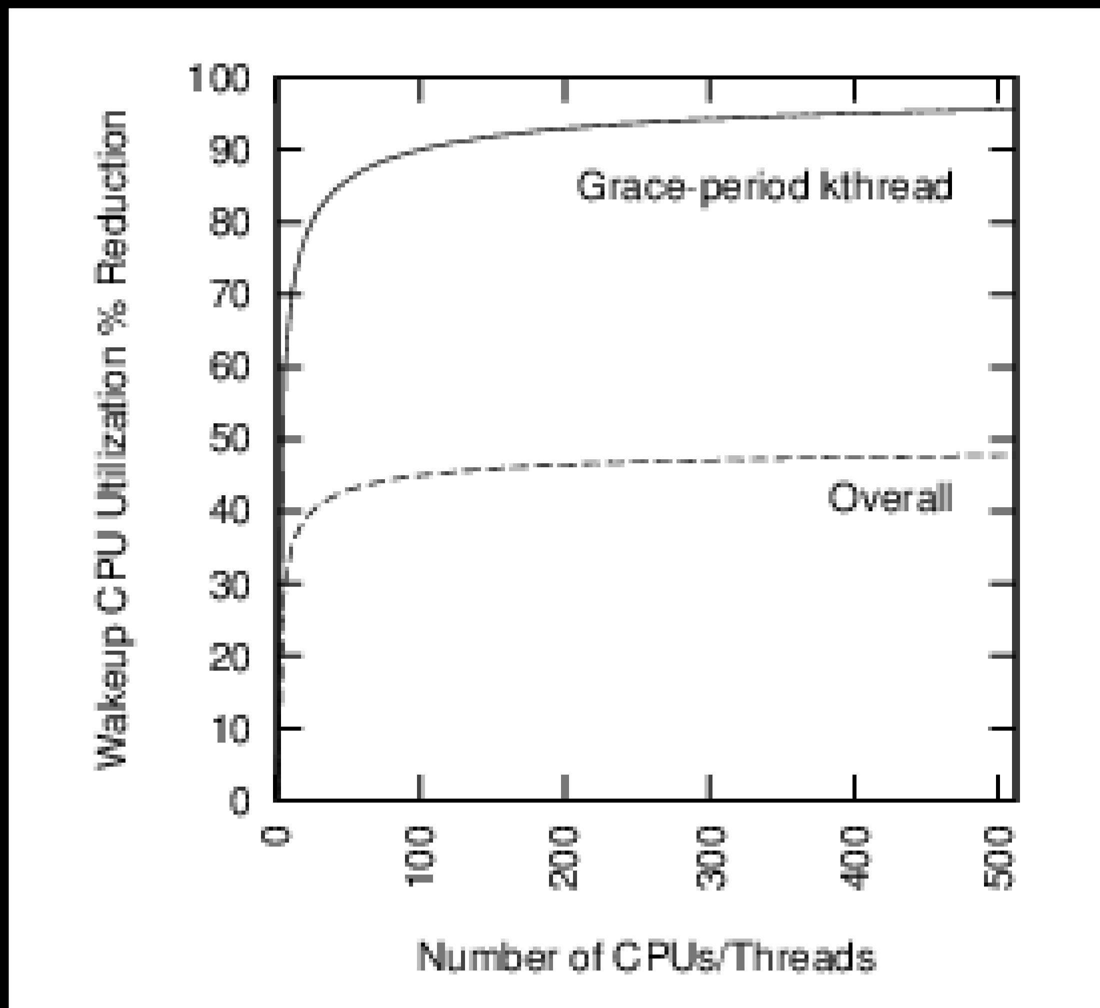
IBM

Rough Diagram Focusing on rcuo Wakeups, 20 CPUs



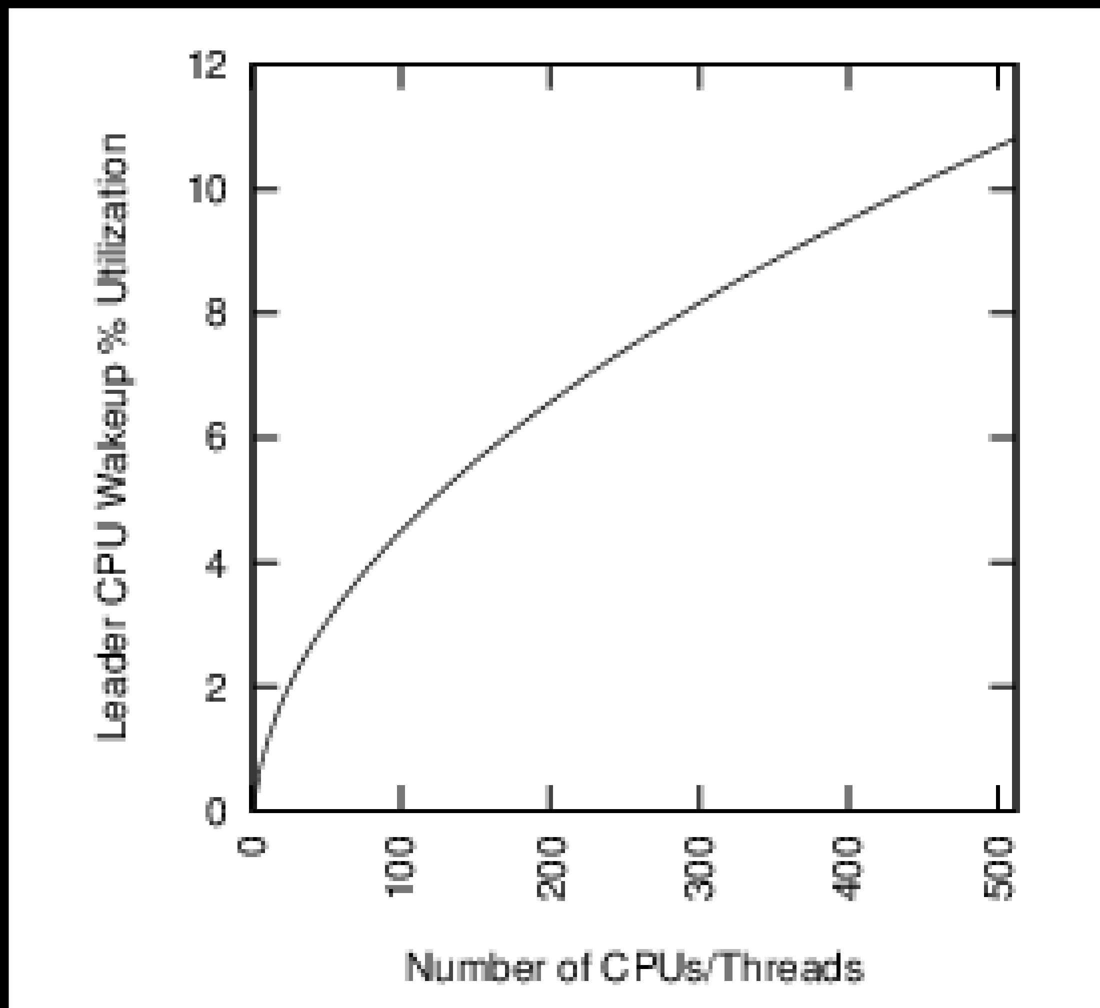
© 2015 IBM Corporation

More CPUs, Greater Reductions in Leader Wakeups!



© 2015 IBM Corporation

But More Leader CPU Load For “Switchy” Workloads



Might someday need an additional level of hierarchy, but other Linux scalability issues will strike first

© 2015 IBM Corporation

IBM

Overall, Looks Pretty Good!!!

- Not only spreads out the overhead, but also reduces it!
- Systems with lots of CPUs unlikely to run “switchy” workloads
- What is not to like?

© 2015 IBM Corporation

IBM

Overall, Looks Pretty Good!!!

- Not only spreads out the overhead, but also reduces it!
- Systems with lots of CPUs unlikely to run “switchy” workloads
- What is not to like?
- “Industry Experience: 1 of 6 Fixes Introduces Bug”
  - System hang, also due to callbacks being posted early
  - Reported by Amit Shah, fixed by Pranith Kumar
  - 11ed7f934cb8: Make nocb leader kthreads process pending callbacks after spawning

© 2015 IBM Corporation

IBM

Overall, Looks Pretty Good!!!

- Not only spreads out the overhead, but also reduces it!
- Systems with lots of CPUs unlikely to run "switchy" workloads
- What is not to like?
- "Industry Experience: 1 of 6 Fixes Introduces Bug"
  - System hang, also due to callbacks being posted early
  - Reported by Amit Shah, fixed by Pranith Kumar
  - 11ed7f934cb8: Make nocb leader kthreads process pending callbacks after spawning
- So it is all good, right?

© 2015 IBM Corporation

Paul Gortmaker: Why So Many rcuo kthreads???

© 2015 IBM Corporation

Paul Gortmaker: Why So Many rcuo kthreads???

- A handful of CPUs, but ***hundreds*** of rcuo kthreads!!!
  - Supposed to be *at most* three rcuo kthreads per offloaded CPU
  - rcu_bh, rcu_sched, and if CONFIG_PREEMPT=y, rcu_preempt
    - (One for each “flavor” of RCU.)
- It seems that some firmware tells lies about number of CPUs
  - And RCU was stupid enough to believe these lies (http://paulmck.livejournal.com/37494.html)
  - FW said lots and lots of CPUs, so RCU created 100s of rcuo kthreads
    - Which just sat idle, consuming memory and clogging ps listings
- Easy fix: Just don't create rcuo until CPUs come online
  - 9386c0b75dda: Rationalize kthread spawning
  - 35ce7f29a44a: Create rcuo kthreads only for onlined CPUs
  - Passed tests with flying colors!!!

© 2015 IBM Corporation

But The Only Reason It Passed The Tests Was...

- Neither Paul Gortmaker or I had enabled modules
  - Plus my firmware doesn't lie about the number of CPUs!
- It turns out that module removal often uses rcu_barrier()...

© 2015 IBM Corporation

What The Heck is rcu_barrier()???

- Consider the following sequence of events:
  - Kernel module does call_rcu(&p->rcu, my_func)
  - This means that my_func() will be invoked after a grace period
    - If RCU is very busy on that CPU, maybe a ***long*** time after a grace period
  - Suppose that the kernel module is unloaded in the meantime
    - And that any grace periods use a lightly loaded CPU
  - The module might be completely unloaded by the time my_func() is finally invoked
  - Which would be an embarrassing and fatal surprise, because my_func() is no longer in memory!!!

© 2015 IBM Corporation

What The Heck is rcu_barrier()???

- Consider the following sequence of events:
  - Kernel module does call_rcu(&p->rcu, my_func)
  - This means that my_func() will be invoked after a grace period
    - If RCU is very busy on that CPU, maybe a ***long*** time after a grace period
  - Suppose that the kernel module is unloaded in the meantime
    - And that any grace periods use a lightly loaded CPU
  - The module might be completely unloaded by the time my_func() is finally invoked
  - Which would be an embarrassing and fatal surprise, because my_func() is no longer in memory!!!
- Use rcu_barrier() to prevent this by waiting for all previous callbacks to be invoked (see next slide)

© 2015 IBM Corporation

IBM

Using rcu_barrier() to Avoid Function Gone AWOL

- New sequence of events using rcu_barrier():
  - Kernel module does call_rcu(&p->rcu, my_func)
  - This means that my_func() will be invoked after a grace period
    - If RCU is very busy on that CPU, maybe a ***long*** time after a grace period
  - Suppose that the kernel module is unloaded in the meantime
    - But the module invokes rcu_barrier(), which means that my_func has been invoked by the time that rcu_barrier() has returned
  - The module may now be safely unloaded, because my_func() is no longer required
- Two principles behind rcu_barrier() design:
  - Post a callback on each CPU with callbacks, wait for all of them
  - Any given CPU's callbacks are ***always*** invoked in order

© 2015 IBM Corporation

Old rcu_barrier() Pseudocode, Before Offloading

- get_online_cpus()
- Set counter to one
- for_each_online_cpu()
  - If CPU has callbacks, post a callback and atomically increment counter
    - (Callback will atomically decrement counter, wake us if zero)
- put_online_cpus()
- Atomically decrement counter
- Wait for counter to reach zero

- But with offloading, offline CPUs can still have callbacks!!!

© 2015 IBM Corporation

IBM

New rcu_barrier() Pseudocode, Given Offloading

- get_online_cpus()
- Set counter to one
- for_each_online_cpu()
  - If CPU has callbacks ***or if CPU's callbacks are offloaded,*** post a callback and atomically increment counter
    - (Callback will atomically decrement counter, wake us if zero)
- put_online_cpus()
- Atomically decrement counter
- Wait for counter to reach zero

- **But never-onlined CPUs don't have rcuo kthread!!!**

© 2015 IBM Corporation

Fixed rcu_barrier() Pseudocode, Given Offloading

- get_online_cpus()
- Set counter to one
- for_each_online_cpu()
  - If CPU has callbacks on the one hand, or if CPU's callbacks are offloaded ***and the CPU has an rcuo kthread*** on the other, post a callback and atomically increment counter
    - (Callback will atomically decrement counter, wake us if zero)
- put_online_cpus()
- Atomically decrement counter
- Wait for counter to reach zero
- d7e29933969e: Make rcu_barrier() understand about missing rcuo kthreads

© 2015 IBM Corporation

But There Was Another Bug...

- If the CPUs came online out of order, the leader/follower lists could drop followers from the list
- Found by inspection, easy fix:
  - bbe5d7a93a39: Fix for rcuo online-time-creation reorganization bug

© 2015 IBM Corporation

But There Was Another Bug...

- If the CPUs came online out of order, the leader/follower lists could drop followers from the list
- Found by inspection, easy fix:
  - bbe5d7a93a39: Fix for rcuo online-time-creation reorganization bug
- Which prompted me to do an exhaustive usermode test
  - Which I should have done in the first place!

© 2015 IBM Corporation

IBM

Lessons (Re)Learned

- Limit the scope of any changes
  - Don't put innocent bystanders at risk
- Linux has an amazing range of workloads and hardware
  - You cannot hope to test them all
- Fixes can generate additional bugs
  - Murphy says that fixes ***will*** generate additional bugs
  - But sometimes serendipity happens!
- Fixes for bugs that are minor and that most users don't see require more caution
  - Unlike a deterministic boot-time panic, there is something to lose!
  - And this is why we have rules about what is accepted when

© 2015 IBM Corporation

IBM

Lessons (Re)Learned

- Limit the scope of any changes
  - Don't put innocent bystanders at risk
- Linux has an amazing range of workloads and hardware
  - You cannot hope to test them all
- Fixes can generate additional bugs
  - Murphy says that fixes ***will*** generate additional bugs
  - But sometimes serendipity happens!
- Fixes for bugs that are minor and that most users don't see require more caution
  - Unlike a deterministic boot-time panic, there is something to lose!
  - And this is why we have rules about what is accepted when
- People probably trust me more than they should
  - Though this experience might well have fixed that problem!

© 2015 IBM Corporation

Most Important Lesson (Re)Learned

- It is not enough to check your assumptions
- If you have a long-held assumption, you may have:
  - Built towers of logic on that assumption
  - Adopted processes based on those towers of logic
  - Formed habits based on those processes
    - Habits of thought, words, and deeds
- Fixing a false assumption requires also fixing logic, processes, and, hardest of all, habits
- False assumption: NO_HZ_FULL users build their kernels
  - Proven false when distros planned NO_HZ_FULL kernels
  - My habit was “dont' worry, but fix bugs if and when they arise”
  - Falsification should have motivated aggressive validation
    - And did, eventually...

© 2015 IBM Corporation

IBM

Additional Issues With Bare-Metal Operation

- 1Hz residual tick:
  - Just in case something we haven't found yet needs interrupts...
  - Kevin Hilman has a patch series that turns this off: Use at your own risk!
- Timer wheel:
  - Suppose that the application occasionally enters the kernel
  - Current timer-wheel code will proceed jiffy-by-jiffy to catch up: latency spikes!
  - “Bandaid” patches in mainline as of 3.15
  - Thomas Gleixner working on more general solution
- Precise delta-time process accounting causes some delays
  - Perhaps an option for coarse process accounting?
- “Whack-a-mole” with other problems as they arise!
  - Issues whacked thus far include workqueues, automating kthread placement, eliminating vmstat activity, interactions with time, …
  - Lots of work from lots of people, with Frederic doing much of the heavy lifting

© 2015 IBM Corporation

To Probe More Deeply Into Adaptive Ticks

- Documentation/timers/NO_HZ.txt
- Is the whole system idle?
  - http://lwn.net/Articles/558284/
- (Nearly) full tickless operation in 3.10
  - http://lwn.net/Articles/549580/
- "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/

© 2015 IBM Corporation

IBM

Configuration Cheat Sheet (Subject to Change!)

- CONFIG_NO_HZ_FULL=y Kconfig: enable adaptive ticks
  - Implies dyntick-idle mode (specify separately via CONFIG_NO_HZ_IDLE=y)
  - Specify which CPUs at compile time: CONFIG_NO_HZ_FULL_ALL=y
    - But boot CPU is excluded, used as timekeeping CPU
  - "full_nohz=" boot parameter: Specify adaptive-tick CPUs, overriding build-time Kconfig
    - "full_nohz=1,3-7" says CPUs 1, 3, 4, 5, 6, and 7 are adaptive-tick
    - Omitting "full_nohz=": No CPUs are adaptive-tick unless CONFIG_NO_HZ_FULL_ALL=y
    - Boot CPU cannot be adaptive-ticks, it will be used as timekeeping CPU regardless
  - 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 (specify at boot time)
    - RCU_NOCB_CPU_ZERO=y Kconfig: Offload CPU 0 (intended for randconfig testing)
    - RCU_NOCB_CPU_ALL=y Kconfig: Offload all CPUs
  - "rcu_nocbs=" boot parameter: Specify additional offloaded CPUs
- CONFIG_NO_HZ_FULL_SYSIDLE=y: enable system-wide idle detection
  - Still needs more plumbing from Frederic: https://lkml.org/lkml/2014/7/28/540
- Also: CONFIG_HIGH_RES_TIMERS=y, CONFIG_DEBUG_PREEMPT=n, CONFIG_TRACING=n, CONFIG_DEBUG_LIST=n, ...

© 2015 IBM Corporation

IBM

Boot/Doc Cheat Sheet (Subject to Change!)

- Boot:
  - nosoftlockup: Decrease soft-lockup checking overhead, and also remove the corresponding diagnostics. (Decisions, decisions!)
  - isolcpus=n-m: Tell the Linux kernel to isolate the specified CPUs. (Some consider this to be obsolete, others swear by it.)
  - elevator=noop: Disable complex block I/O schedulers. (Some prefer compiling with CONFIG_IOSCHED_NOOP=n, CONFIG_IOSCHED_DEADLINE=n, and CONFIG_IOSCHED_CFQ=n.)
- Documentation:
  - How-to info for kthreads: Documentation/kernel-per-CPU-kthreads.txt
  - Available in 3.10, see Documentation/timers/NO_HZ.txt for more info

© 2015 IBM Corporation

Summary

- General-purpose OS or bare-metal performance?
  - Why not both?
  - Work in progress gets us very close for CPU-bound workloads:
    - Adaptive ticks userspace execution (early version in mainline)
    - RCU callback offloading (version two in mainline)
    - Interrupt, process, daemon, and kthread affinity
    - Timer offloading
  - Some restrictions:
    - Need to reserve CPU(s) for housekeeping; 1-Hz residual tick
    - Adaptive-ticks and RCU-callback-offloaded CPUs specified at boot time
    - One task per CPU for adaptive-ticks usermode execution
    - Global TLB-flush IPIs, cache misses, and TLB misses are still with us
    - Whack-a-mole with various other issues, patches in flight
  - And can maintain energy efficiency as well!

© 2015 IBM Corporation

Summary

- General-purpose OS or bare-metal performance?
  - Why not both?
  - Work in progress gets us very close for CPU-bound workloads:
    - Adaptive ticks userspace execution (early version in mainline)
    - RCU callback offloading (version two in mainline)
    - Interrupt, process, daemon, and kthread affinity
    - Timer offloading
  - Some restrictions:
    - Need to reserve CPU(s) for housekeeping; 1-Hz residual tick
    - Adaptive-ticks and RCU-callback-offloaded CPUs specified at boot time
    - One task per CPU for adaptive-ticks usermode execution
    - Global TLB-flush IPIs, cache misses, and TLB misses are still with us
    - Whack-a-mole with various other issues, patches in flight
  - And can maintain energy efficiency as well!
- Extending Linux's reach further into extreme computing!!!

© 2015 IBM Corporation

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.

© 2015 IBM Corporation

IBM

Questions?

© 2015 IBM Corporation