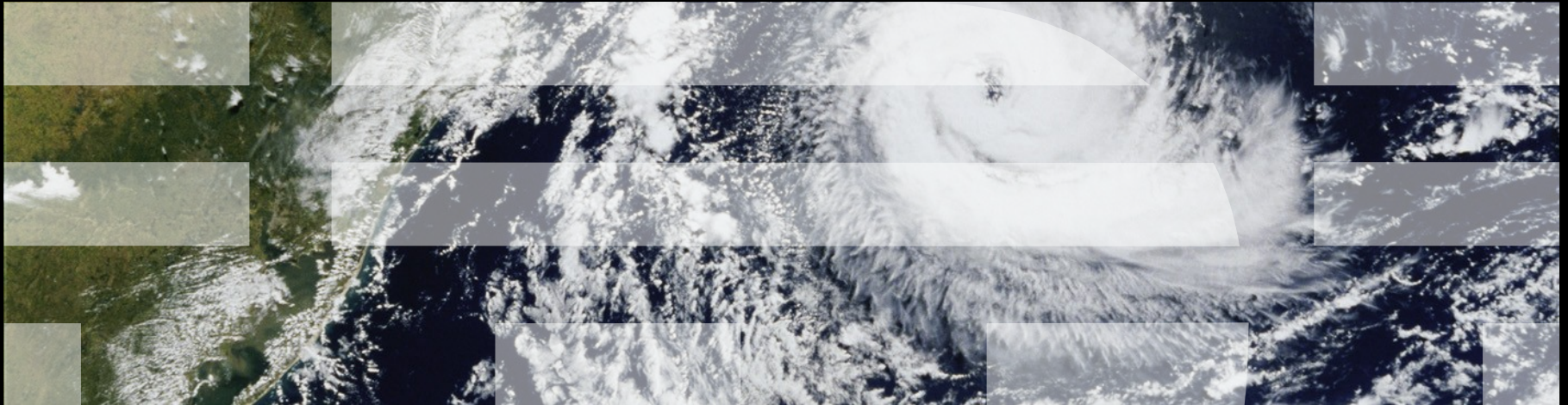


Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center
Member, IBM Academy of Technology
Beaver BarCamp, Corvallis, OR, USA, April 12, 2014



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

(Now With Added Energy Efficiency!)



Group Effort: Acknowledgments

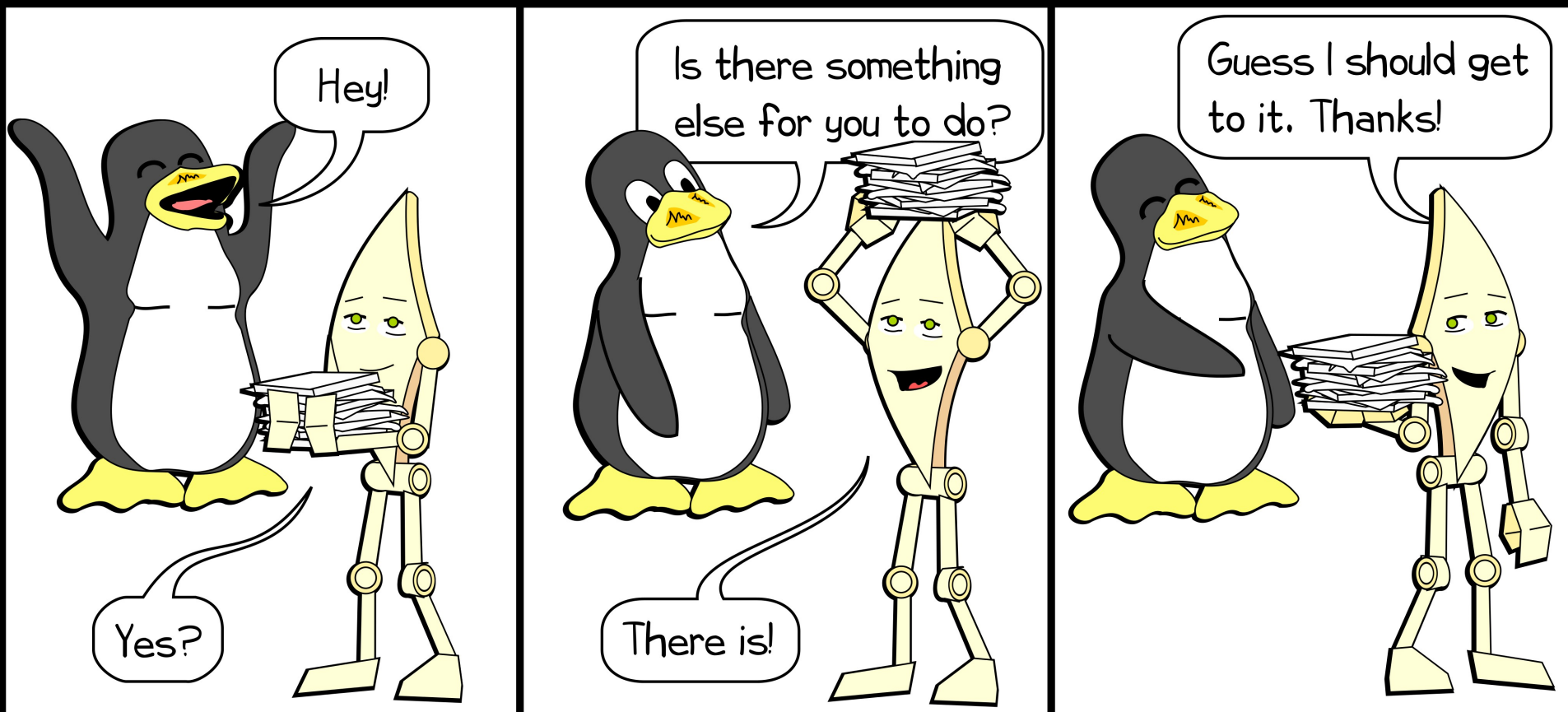
- Josh Triplett: First prototype (LPC 2009)
- Frederic Weisbecker: Core kernel work and x86 port
- Steven Rostedt: Lots of code review and comments, tracing upgrades
- Christoph Lameter: Early adopter feedback, work on vmstat kthread
- Li Zhong: Power port
- Geoff Levand, Kevin Hilman: ARM port, with 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, and probably many others

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

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

Like a scheduling-clock interrupt every jiffy on every CPU.

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



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

Like a scheduling-clock interrupt every jiffy on every CPU.
Whether you needed it or not.

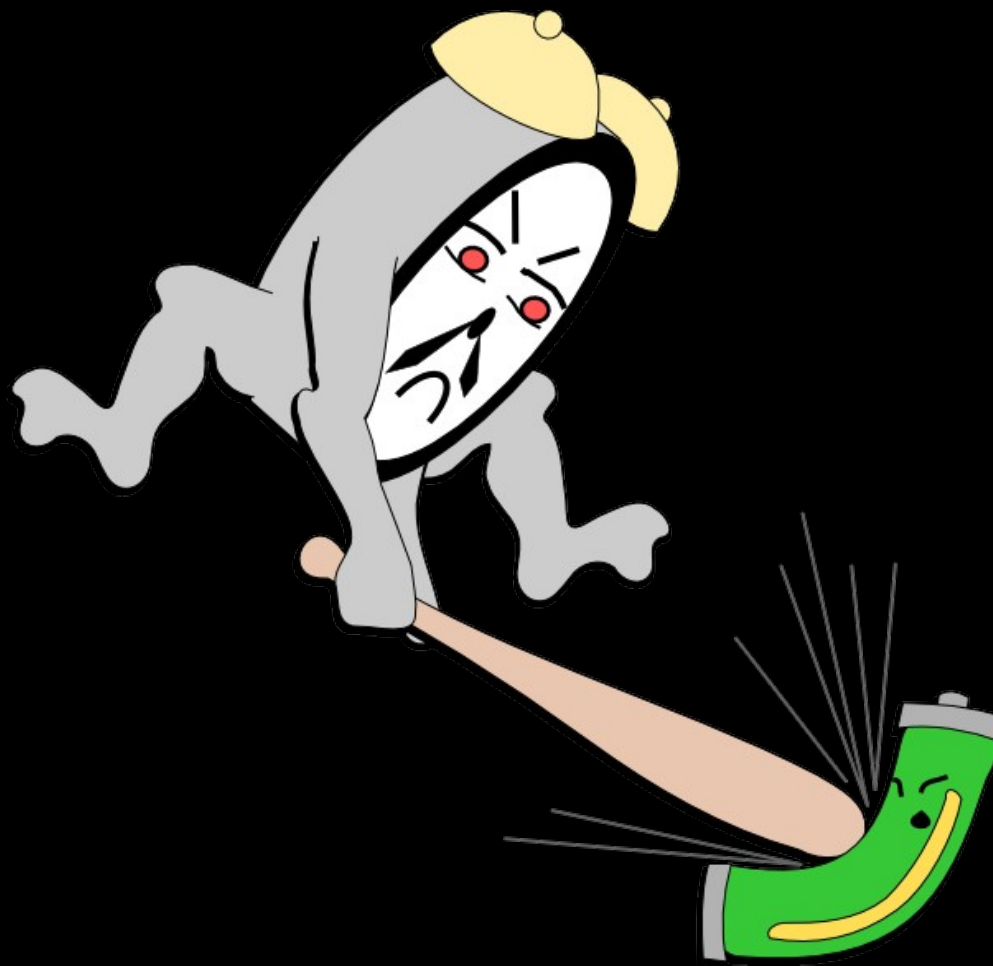
There Used To Be Things You Could Count On...

Like a scheduling-clock interrupt every jiffy on every CPU.

Whether you needed it or not.

And especially, whether your battery needed it or not.

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



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



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

Like a scheduling-clock interrupt every jiffy on every CPU.

Whether you needed it or not.

And especially, whether your battery needed it or not.

Of course, back then you needed a somewhat larger battery...

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

Like a scheduling-clock interrupt every jiffy on every CPU.

Whether you needed it or not.

And especially, whether your battery needed it or not.

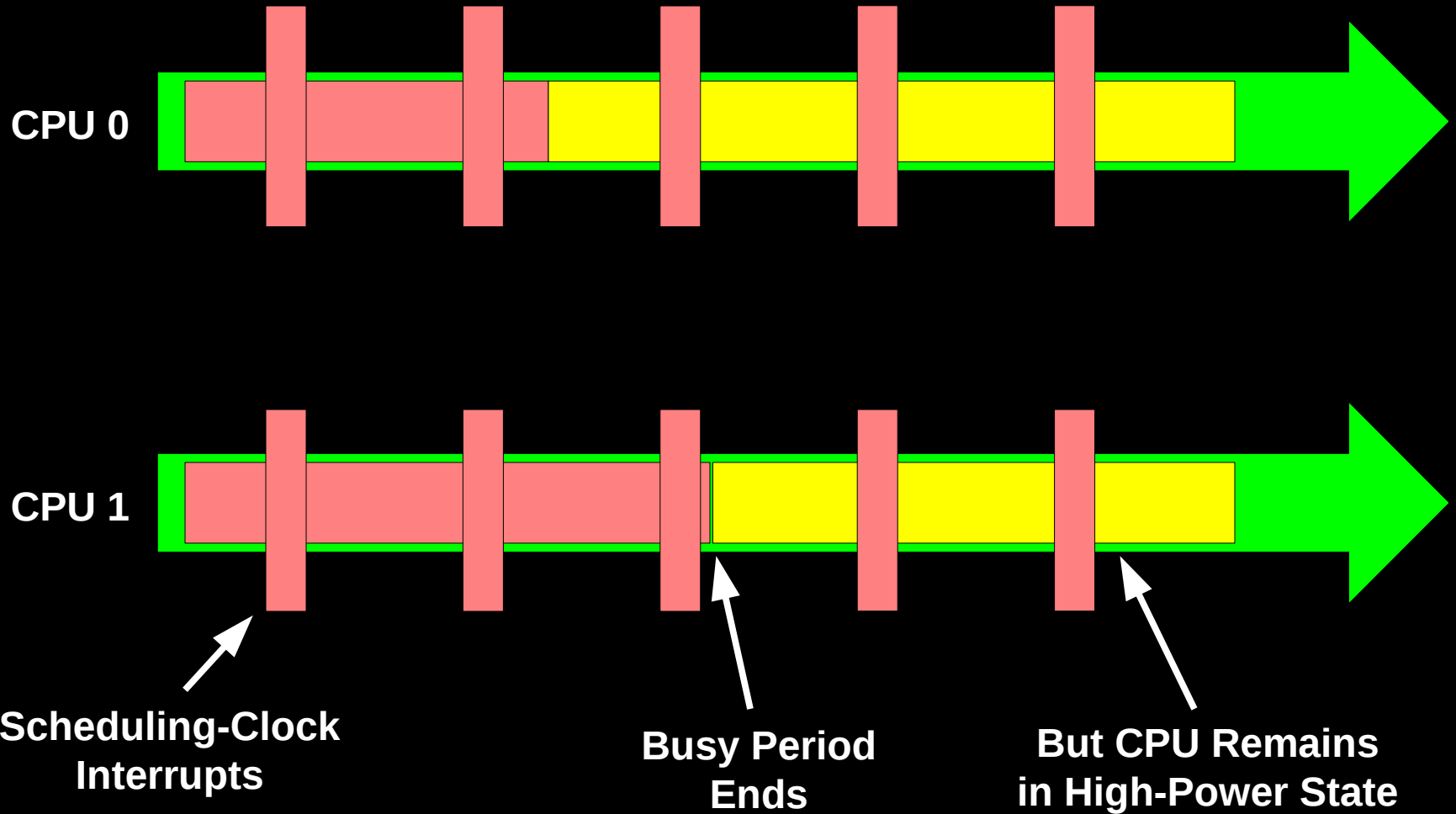
Of course, back then you needed a somewhat larger battery...

And, if your system was portable, a forklift.

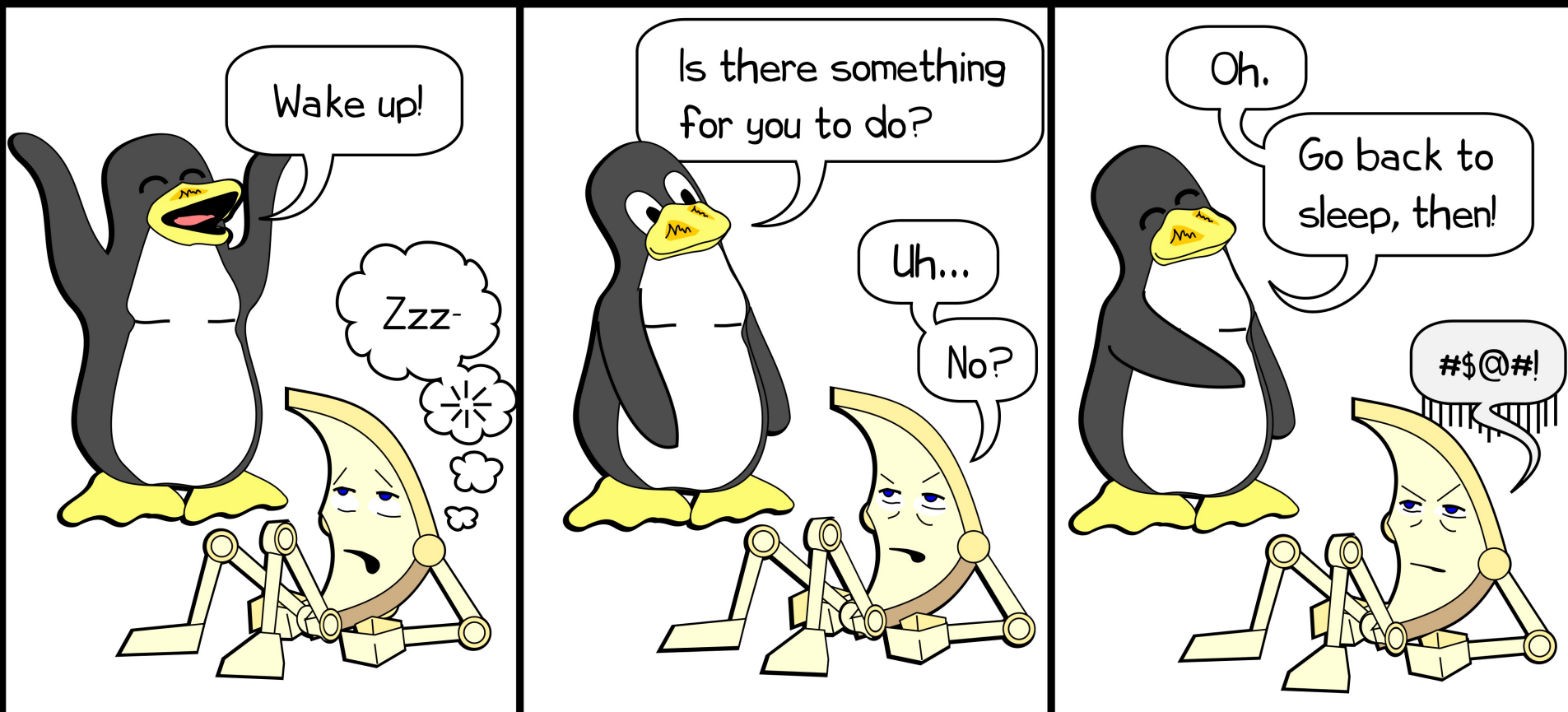
What We Need Instead...



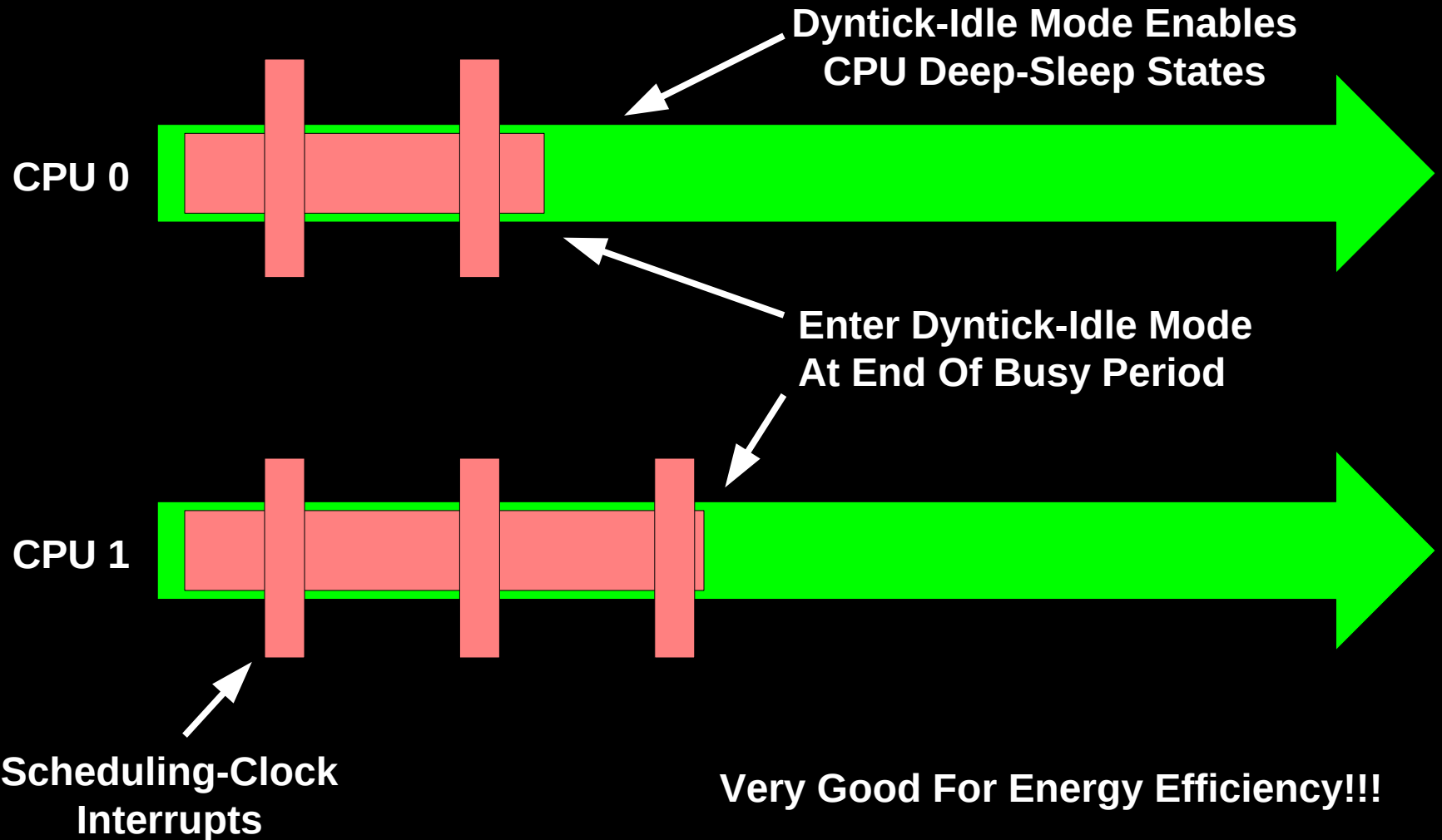
“Good Old Days” Before Linux's dyntick-idle System



“Good Old Days” Before Linux's dyntick-idle System



Semi-Modern Innovation: Linux's dyntick-idle System

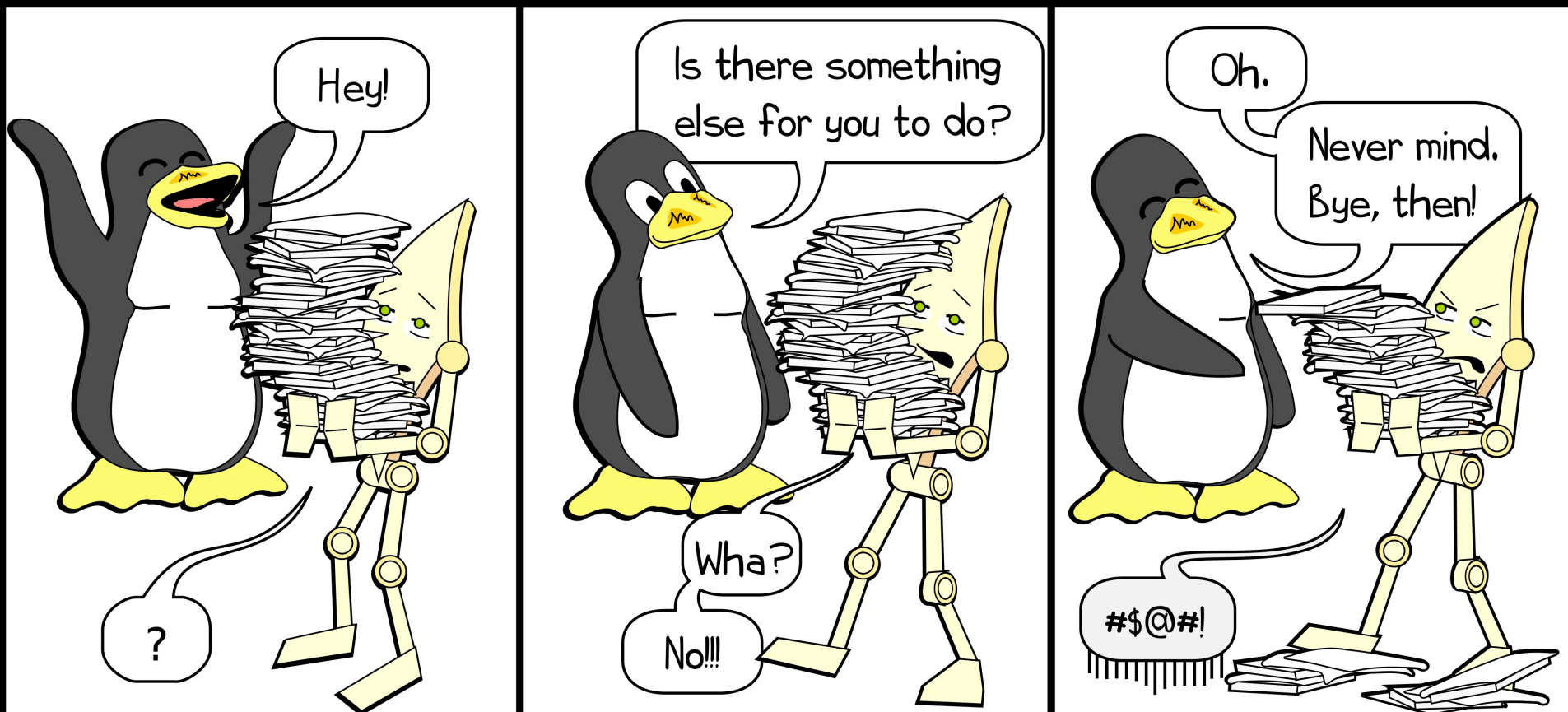


Very Good For Energy Efficiency!!!

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

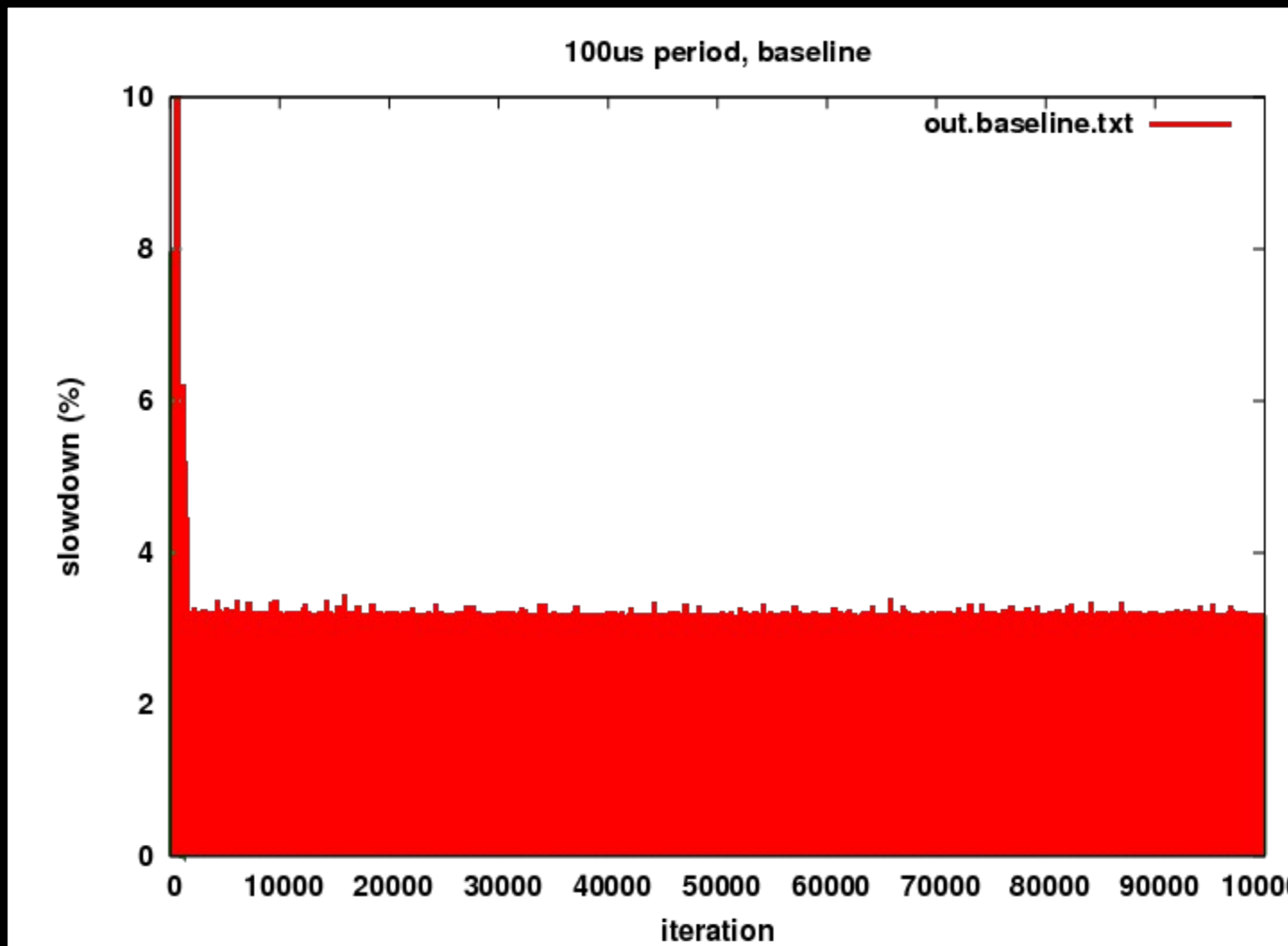
Also: Avoid Unnecessary Usermode Interrupts



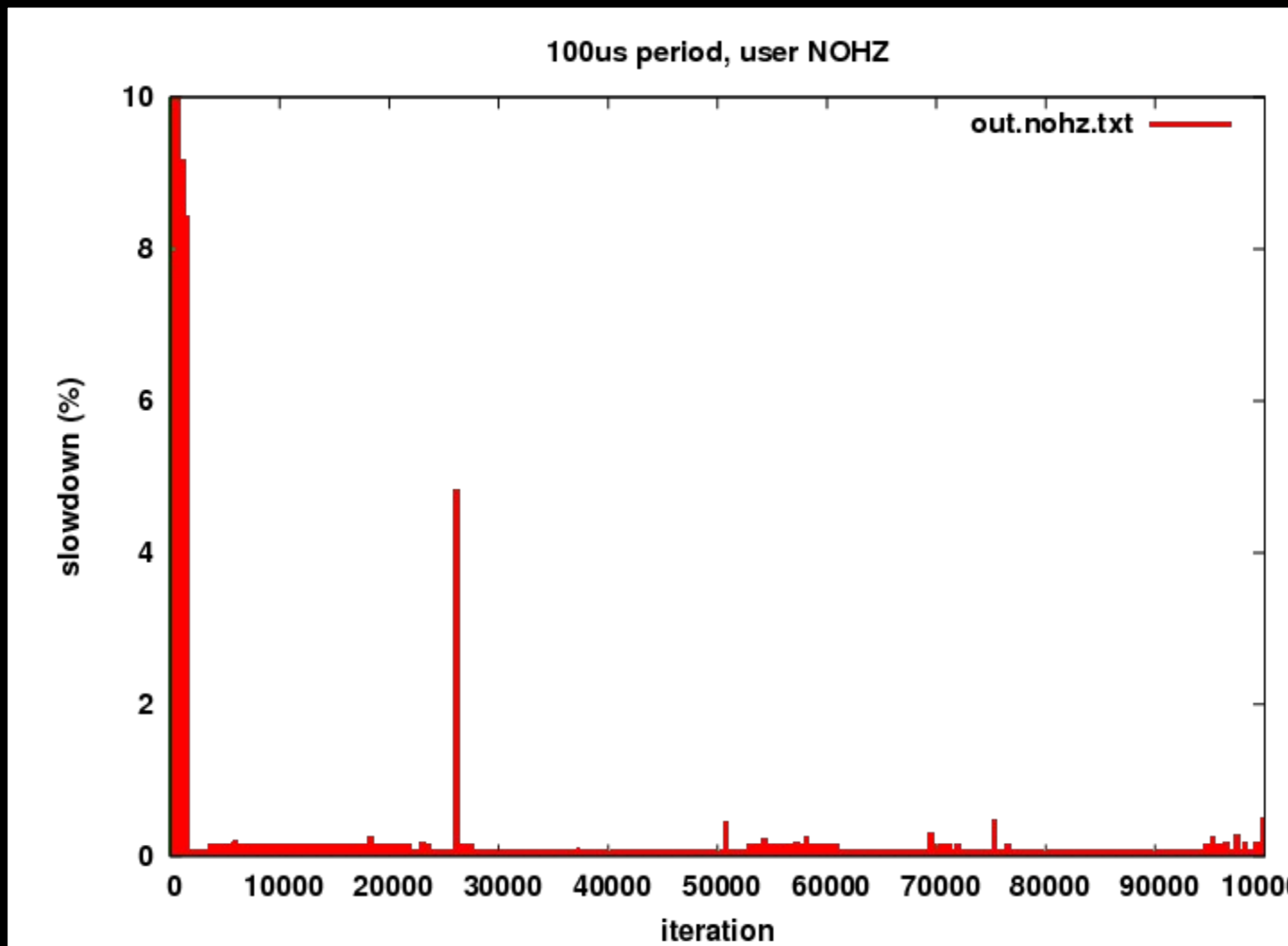
Also: Avoid Unnecessary Usermode Interrupts

- 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
- Josh Triplett prototyped CONFIG_NO_HZ_FULL 2009

Benchmark Results Before (Anton Blanchard)



Benchmark Results After (Anton Blanchard)



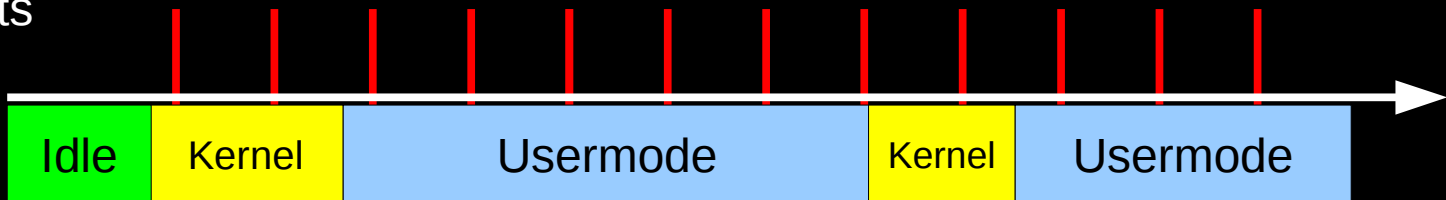
Well worth going after...

But There Were A Few Small Drawbacks...

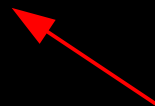
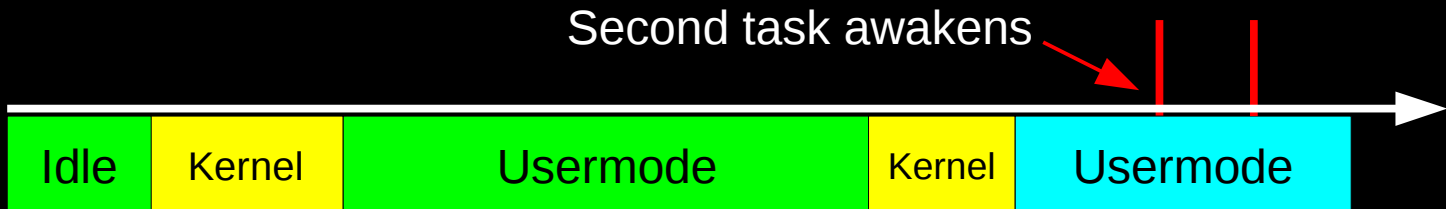
- User applications can monopolize CPU
 - But if there is only one runnable task, so what???
 - If new task awakens, interrupt the CPU, restart scheduling-clock interrupts
 - In the meantime, we have an “adaptive ticks usermode” CPU
- No process accounting
 - Use delta-based accounting, based on when 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

How Does It Work?

Scheduling
clock
interrupts



Second task awakens



One task per CPU

Other Than Energy Efficiency, Looks Great!!!

- But what is the problem with energy efficiency?

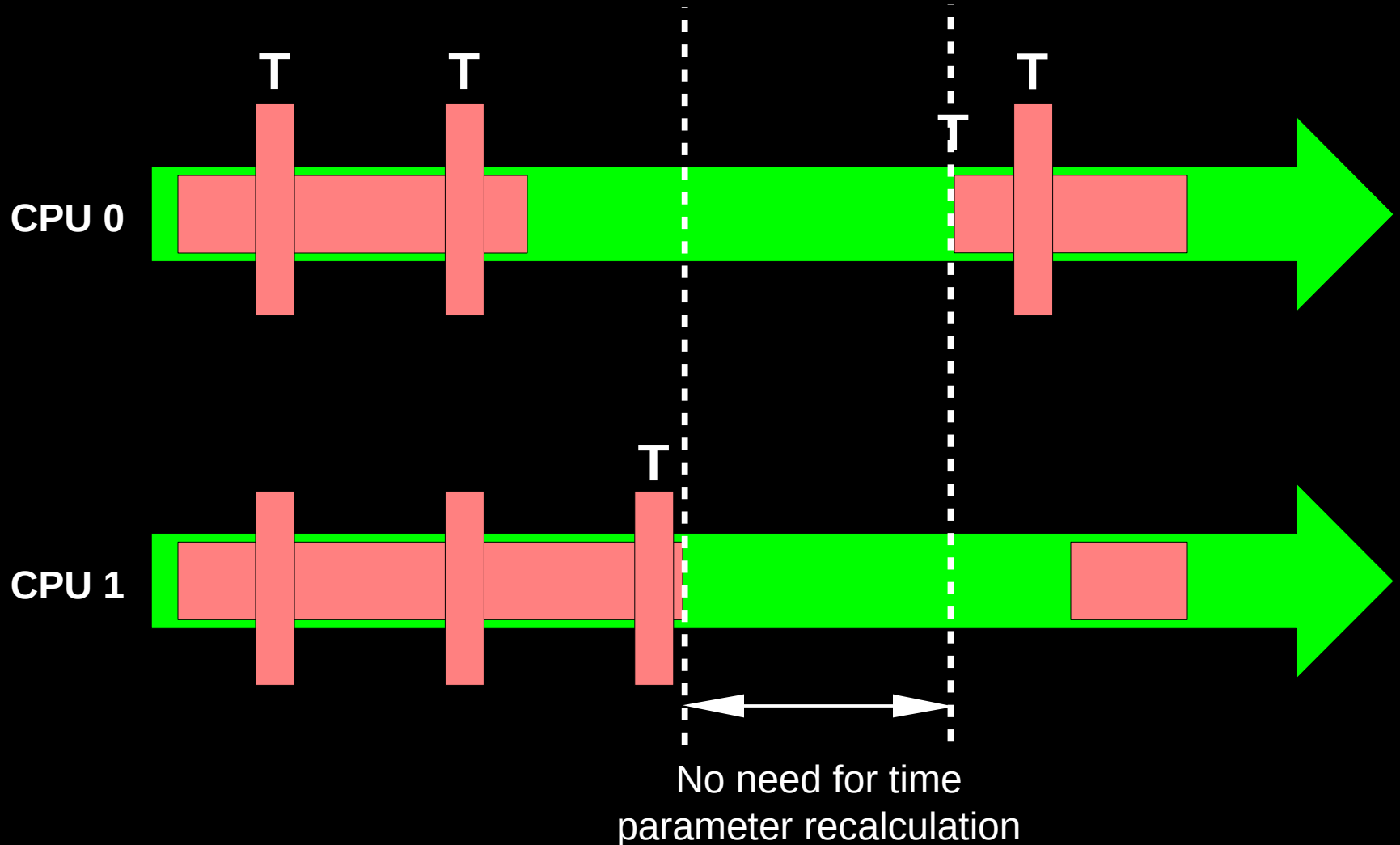
Energy Efficiency and Timekeeping

- Hardware oscillators drift
- Requires periodic recalculation of time conversion parameters, e.g., via NTP
 - Otherwise user applications get bad time data
- One special case

Energy Efficiency and Timekeeping

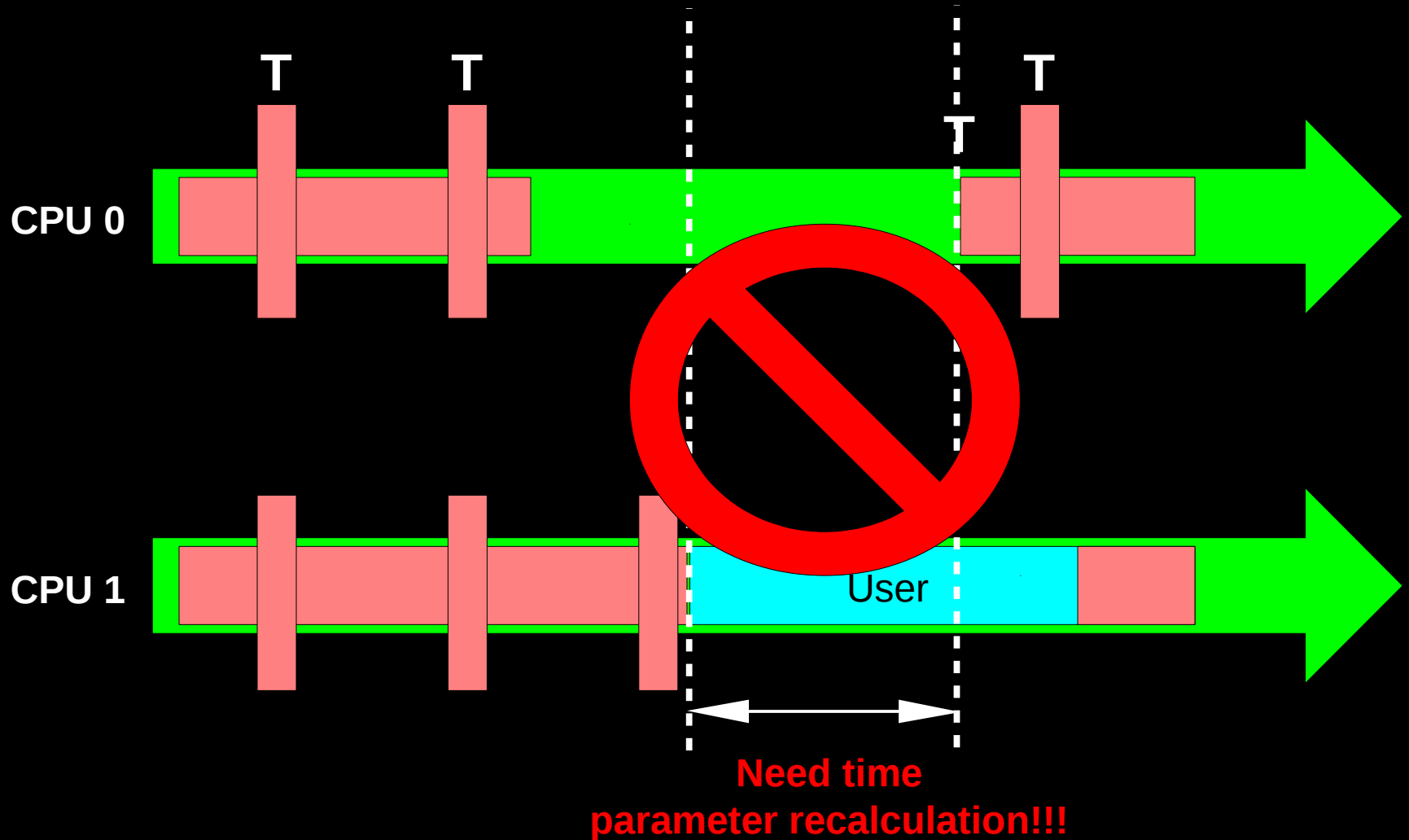
- Hardware oscillators drift
- Requires periodic recalculation of time conversion parameters, e.g., via NTP
 - Otherwise user applications get bad time data
- One special case:
 - If all CPUs are idle, none of them care about accurate timekeeping
 - Just need to recalculate time-conversion parameters when the first CPU goes non-idle

Energy Efficiency, Timekeeping, and Idle



But If Running Userspace, Need Recalculation

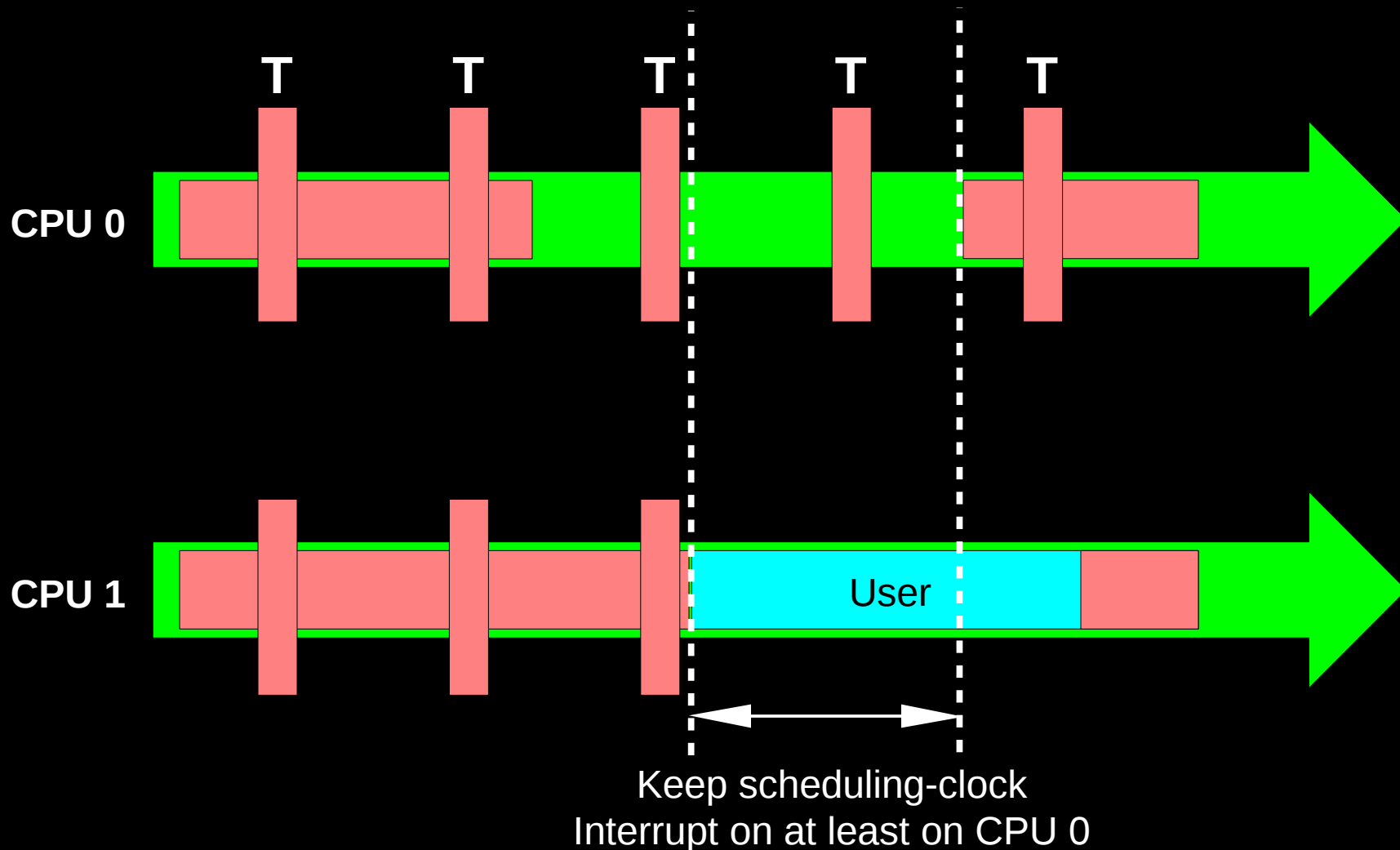
Energy Efficiency, Timekeeping, and Userspace



Shut Down *Almost All* Scheduling-Clock Interrupts

- If all CPUs are idle, we can shut down all CPUs' scheduling-clock interrupts
- If any CPU is non-idle, we need at least one CPU running the scheduling-clock interrupt
- Initial approach: Require that CPU 0 always run the scheduling-clock interrupt

Shut Down *Almost All* Scheduling-Clock Interrupts



The Battery-Powered Embedded Folks Not Happy...

The Battery-Powered Embedded Folks Not Happy... We Must Shut Down *All* Scheduling-Clock Interrupts

We Must Shut Down *All* Scheduling-Clock Interrupts: Two Simple (But Broken) Approaches

- Just count non-idle CPUs!!!
 - Maintain an atomic counter
 - When a CPU goes idle, atomically increment the counter
 - When a CPU goes non-idle atomically decrement the counter
 - This is a *really bad* idea on a system with lots of CPUs
 - Memory contention will degrade scalability and performance – and in extreme cases, hangs the system

- Just scan CPUs looking for non-idle ones!!!
 - Have the timekeeping kthread periodically scan CPUs: If all are idle, turn off the scheduling-clock tick
 - Vulnerable to race conditions, see next slide

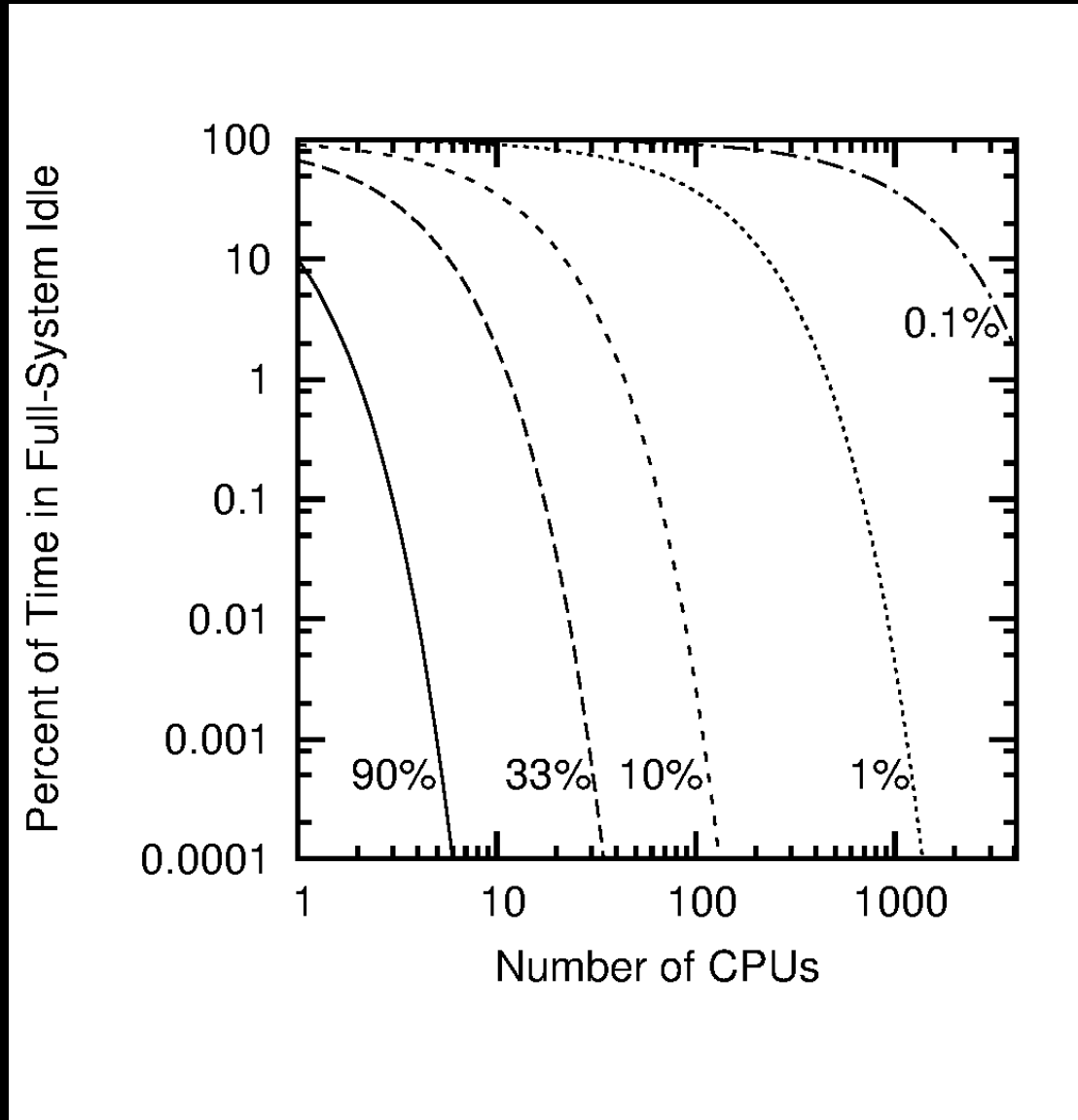
Scanning For Full-System Idle is Broken!

- Scanning is subject to race conditions!
- Example race scenario on a four-CPU system:
 - CPU 0 goes idle (3 CPUs non-idle)
 - Timekeeping kthread checks CPU 0, sees it idle
 - CPU 1 goes idle (2 CPUs non-idle)
 - CPU 0 goes non-idle (3 CPUs non-idle)
 - Timekeeping kthread checks CPU 1, sees it idle
 - CPU 2 goes idle (2 CPUs non-idle)
 - CPU 1 goes non-idle (3 CPUs nonidle)
 - Timekeeping kthread checks CPU 2, sees it idle
 - CPU 3 goes idle (2 CPUs non-idle)
 - CPU 2 goes non-idle (3 CPUs nonidle)
 - Timekeeping kthread checks CPU 3, sees it idle
 - **Timekeeping kthread wrongly concludes that the entire system is idle!!!**
 - **Despite the fact that there has never been fewer than two non-idle CPUs**

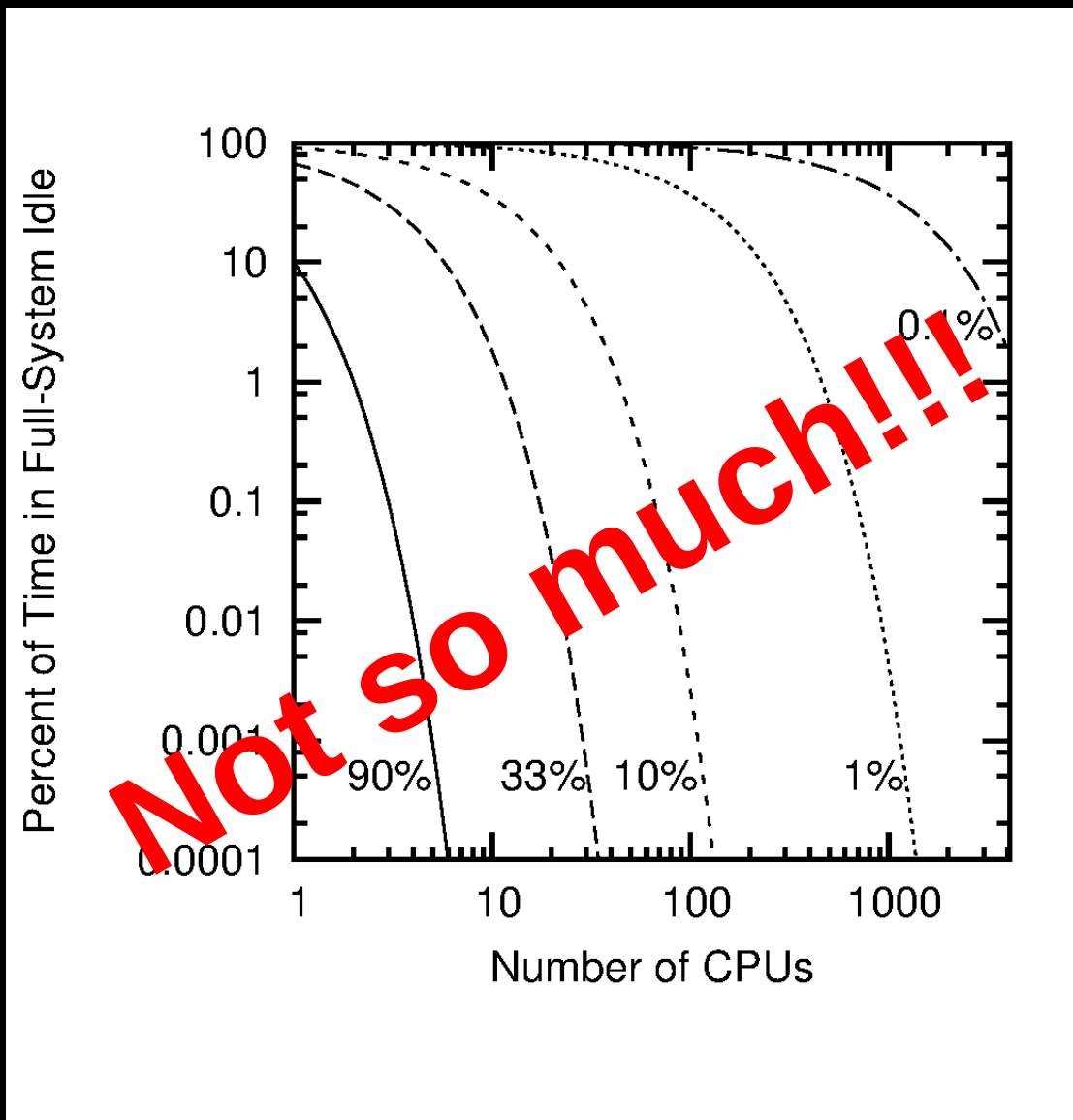
How To Efficiently Detect Full-System Idle?

- We have to give up something:
 - Energy efficiency
 - Scalability
 - Determinism
 - Full-system idle detection latency
 - Sanity

Opportunistic Idle on Large Systems?



Opportunistic Idle on Large Systems?



How To Efficiently Detect Full-System Idle?

- We have to give up something:
 - ~~Energy efficiency~~
 - ~~Scalability~~
 - ~~Determinism~~
 - **Full-system idle detection latency**
 - ~~Sanity~~

How To Efficiently Detect Full-System Idle?

- We have to give up something:
 - ~~Energy efficiency~~
 - ~~Scalability~~
 - ~~Determinism~~
 - **Full-system idle detection latency**
 - ~~Sanity~~: You cannot give up something that you do not have!!!

How To Efficiently Detect Full-System Idle?

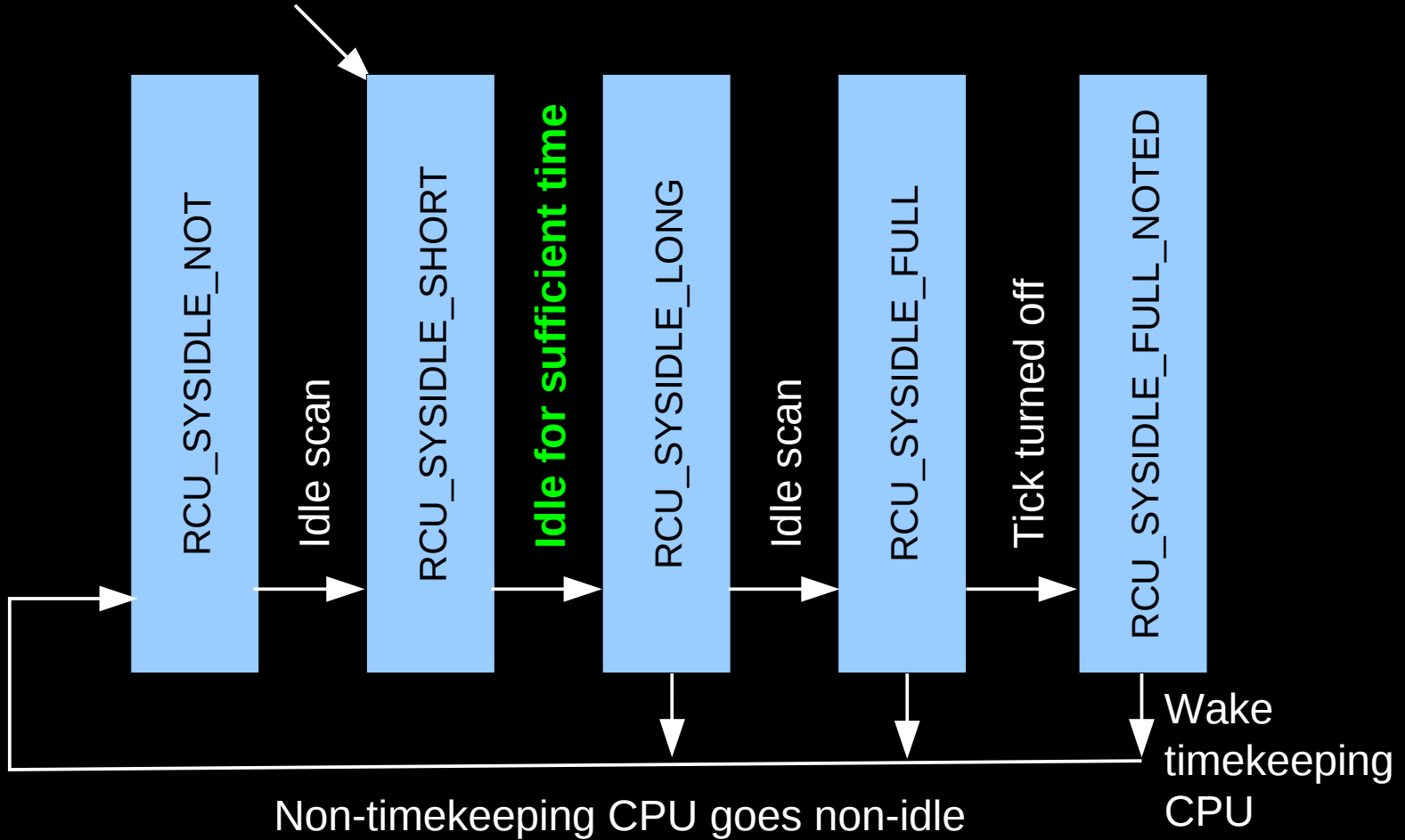
- We have to give up something:
 - **Full-system idle detection latency**
- Use a state machine
- Enter full-system-idle state more slowly on larger systems
 - Forces more time between atomic updates of global state variable on large systems, maintaining a constant level of memory contention
- Leverage RCU's pre-existing scan of idle CPUs
 - If a CPU is idle, it does not respond to RCU grace periods
 - RCU therefore scans CPUs, report quiescent states on their behalf
 - Also track last time each CPU went idle

Full-System-Idle State Machine

- Added twist: A timekeeping CPU being non-idle must not prevent the system from entering full-system-idle state!
- States:
 - RCU_SYSIDLE_NOT: Some CPU is not idle.
 - Return to this state any time a non-timekeeping CPU goes non-idle from RCU_SYSIDLE_LONG or later state.
 - RCU_SYSIDLE_SHORT: All CPUs idle for brief period.
 - Advance here if scan finds all non-timekeeping CPUs idle.
 - RCU_SYSIDLE_LONG: All CPUs idle for “long enough”.
 - Advance here if in RCU_SYSIDLE_SHORT state long enough, and if all CPUs remained idle throughout that time.
 - RCU_SYSIDLE_FULL: All CPUs idle, ready for sysidle.
 - Advance here from RCU_SYSIDLE_LONG if still idle on next scan.
 - RCU_SYSIDLE_FULL_NOTED: Scheduling-clock tick disabled globally.
 - Advance here when timekeeping kthreads sees RCU_SYSIDLE_FULL state.

Full-System-Idle State Machine Schematic

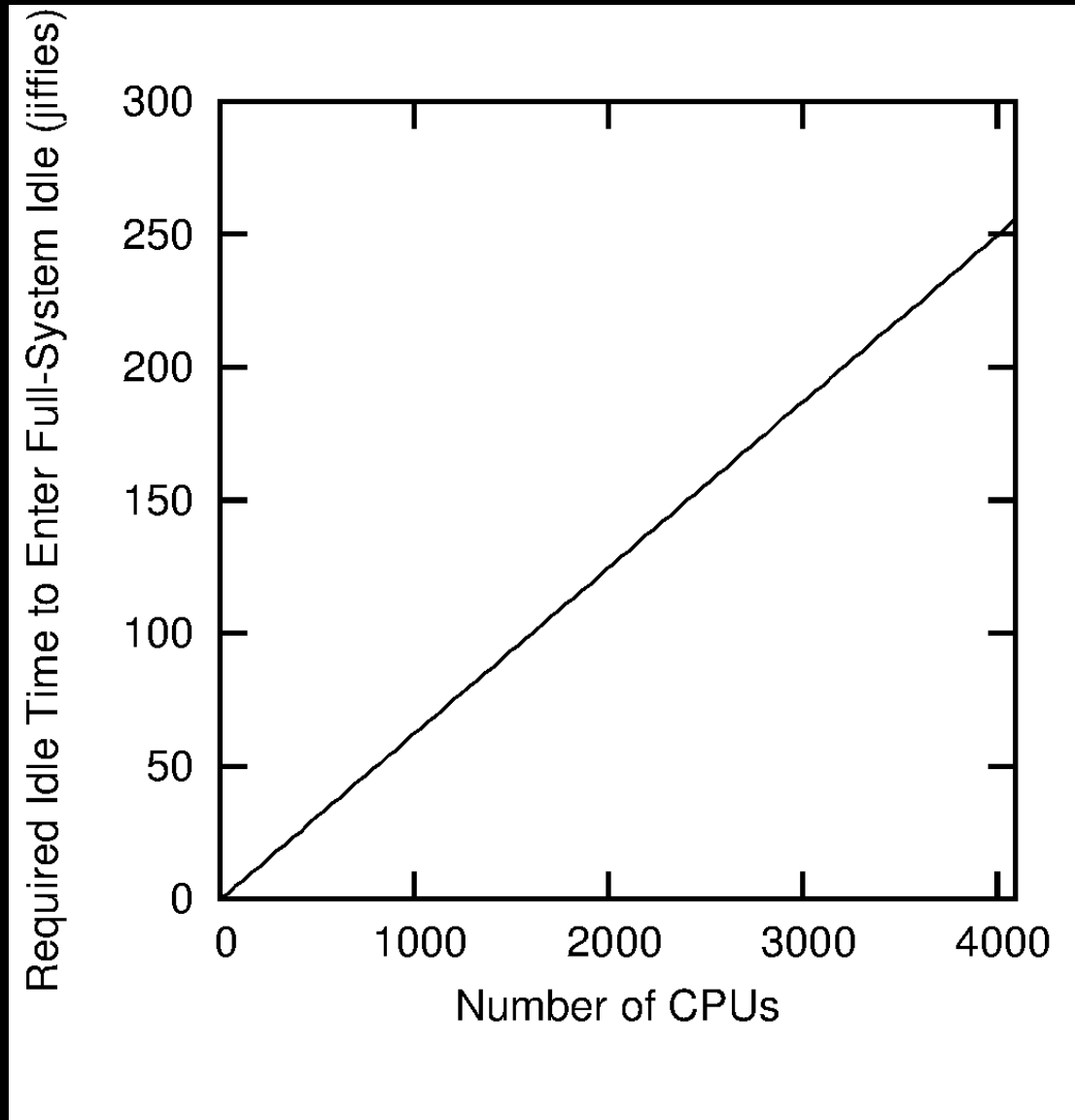
Protect against memory contention



How Long Idle in RCU_SYSIDLE_SHORT State?

- CPU going idle records the time
- RCU remembers most recent idle time when scanning CPUs
- Advance to RCU_SYSIDLE_LONG only if it has been sufficiently long since the last CPU went idle
 - Increases linearly with increasing numbers of CPUs
 - Adjusted by HZ and number of CPUs per rcu_node structure

How Long Is “Sufficiently Long”, Anyway?



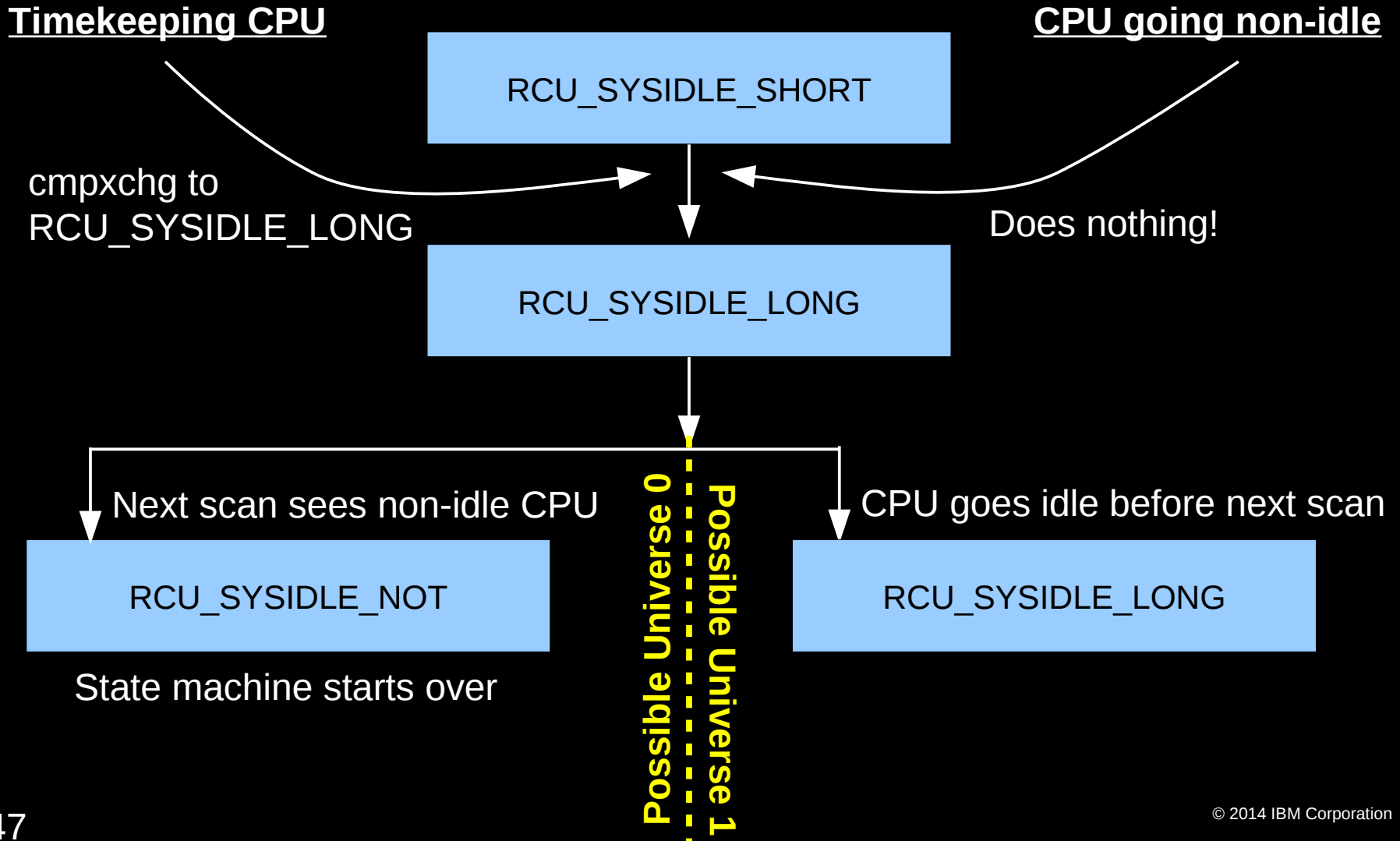
Avoiding Non-Idle Races

- Bad scenario: Timekeeping CPU turns off all scheduling-clock interrupts, then does not notice a CPU going non-idle
- Avoid this as follows:
 - CPU going non-idle in the `RCU_SYSIDLE_FULL_NOTED` state will force scheduling-clock interrupts back on.
 - CPU going non-idle will force the state to `RCU_SYSIDLE_NOT` unless the current state is `RCU_SYSIDLE_SHORT`
 - This means that there is at least one remaining scan (from `RCU_SYSIDLE_LONG` to `RCU_SYSIDLE_FULL`): During this scan, the CPU will be detected as non-idle, forcing state back to `RCU_SYSIDLE_NOT`
 - This requires careful use of atomic operations and memory barriers
- Be careful, getting this right is harder than it looks!

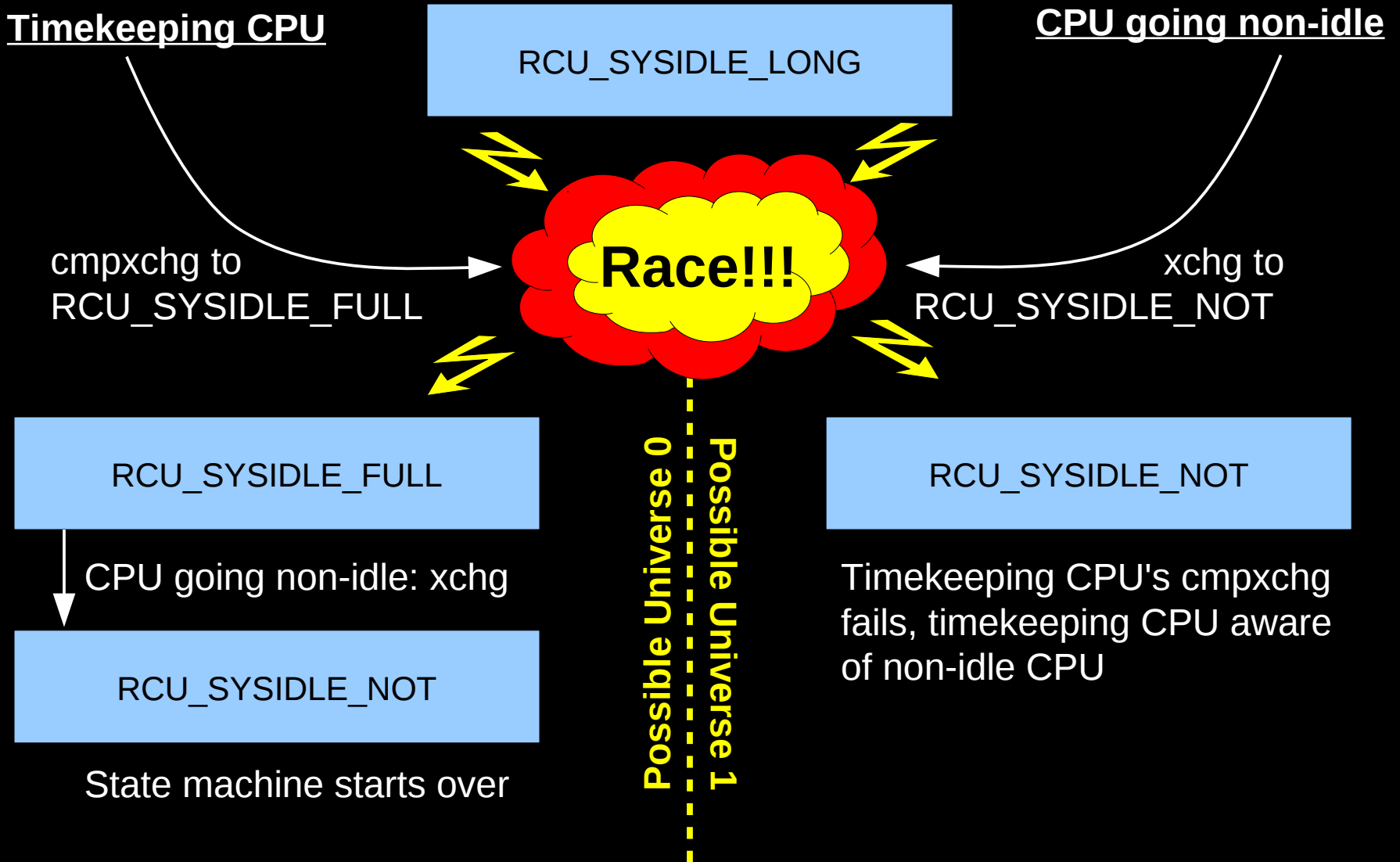
Avoiding Non-Idle Races

- Any CPU can drive state back down to `RCU_SYSIDLE_NOT`
 - It does so when it goes non-idle, but only if state has advanced to `RCU_SYSIDLE_LONG` or further
 - Uses atomic `xchg()` operation to update state: If returns `RCU_SYSIDLE_FULL_NOTED`, wakes up timekeeping CPU
- Only one task advances the state
 - After each scan finds all CPUs idle, with optional time constraint
 - Uses `cmpxchg()`, upon failure assumes CPU went non-idle
- If CPU going non-idle sees `RCU_SYSIDLE_SHORT`, state might advance to `RCU_SYSIDLE_LONG`
 - But memory barriers guarantee that next scan sees the non-idle CPU
- Diagrams on next three slides...

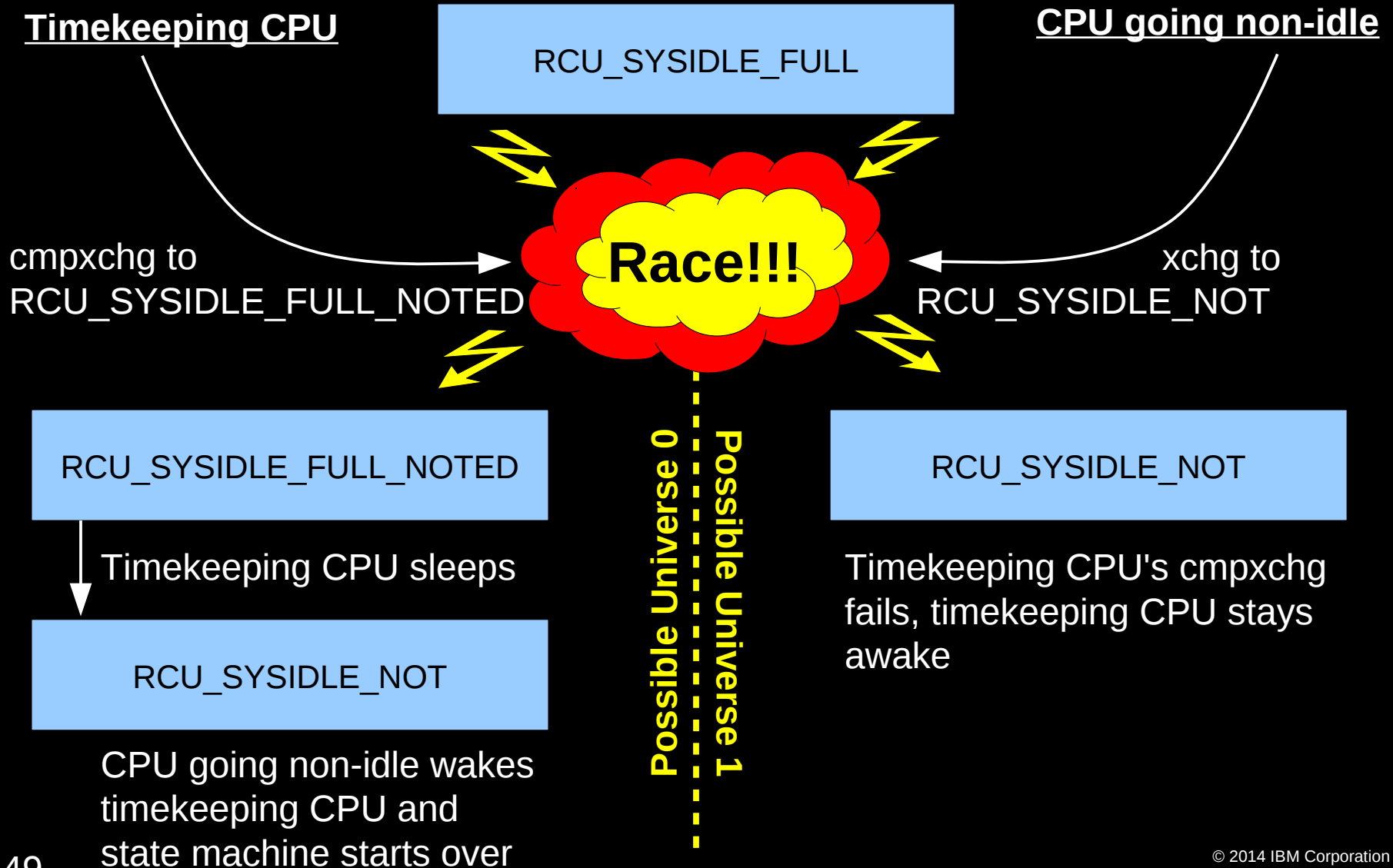
Handling Non-Idle Races: Scenario 1



Handling Non-Idle Races: Scenario 2



Handling Non-Idle Races: Scenario 3



Sounds Complex! Did You Mechanically Verify This?

Sounds Complex! Did You Mechanically Verify This?

- Well, I tried via goto-cc/goto-instrument/satabs:

Performing pointer analysis for concurrency-aware abstraction

```
satabs: value_set.cpp:1183: void value_sett::assign(const exprt&, const exprt&, const namespace_t&, bool): Assertion `base_type_eq(rhs.type(), type, ns)' failed.
```

Aborted (core dumped)

Bug report to the authors (who have been responsive)

- Array allocation problem, fix is still on the way...
- Used several other tools, all of which failed in interesting ways

- So I fell back on Promela/spin
 - 243 lines of Promela says it works!!! (Safety and forward progress)

Sounds Complex! Did You Mechanically Verify This?

- Well, I tried via goto-cc/goto-instrument/satabs:

Performing pointer analysis for concurrency-aware abstraction

```
satabs: value_set.cpp:1183: void value_sett::assign(const exprt&, const exprt&, const namespace_t&, bool): Assertion `base_type_eq(rhs.type(), type, ns)' failed.
```

Aborted (core dumped)

Bug report to the authors (who have been responsive)

- Array allocation problem, fix is still on the way...
- Used several other tools, all of which failed in interesting ways

- So I fell back on Promela/spin

- 243 lines of Promela says it works!!! (Safety and forward progress)
- Should I believe it? ;-)

Other Issues

- 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
 - Resulting in extreme latency blows
 - “Bandaïd” patches available: single-timer and no-timer cases
 - Thomas Gleixner working on more general solution
- vmstat kthread
 - Christoph Lameter has a prototype patch
- RCU_SOFTIRQ
 - Patches to avoid needlessly invoking this softirq handler
- “Whack-a-mole” with other problems, e.g., unbound workqueues

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/>

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, work in progress
- Also: CONFIG_HIGH_RES_TIMERS=y, CONFIG_DEBUG_PREEMPT=n, CONFIG_TRACING=n, CONFIG_DEBUG_LIST=n, ...

Patch Cheat Sheet 1/2 (Subject to Change!)

- Get the vmstat per-CPU kthreads out of the way:
 - <http://lwn.net/Articles/569669/>
- Turn off the residual 1Hz tick:
 - <http://www.spinics.net/lists/arm-kernel/msg273778.html>
- Remove residual RCU_SOFTIRQ jitter:
 - <https://lkml.org/lkml/2013/11/15/561> (in 3.14)
 - <https://lkml.org/lkml/2014/2/17/619>, <https://lkml.org/lkml/2014/2/17/678>, <https://lkml.org/lkml/2014/2/17/780> (expected in 3.15)
- Timer latency bandaids (to be obsoleted by rewrite):
 - <https://lkml.org/lkml/2014/1/15/924>, <https://lkml.org/lkml/2014/1/15/929>, <https://lkml.org/lkml/2014/1/15/928>, <https://lkml.org/lkml/2014/1/15/925>, <https://lkml.org/lkml/2014/1/15/930> (expected in 3.15)

Patch Cheat Sheet 2/2 (Subject to Change!)

- Workqueues:

- <https://lkml.org/lkml/2014/3/27/242>, <https://lkml.org/lkml/2014/3/27/237>,
<https://lkml.org/lkml/2014/3/27/236>, <https://lkml.org/lkml/2014/3/27/239>

- Dedicated timekeeper IPI:

- <https://lkml.org/lkml/2014/3/19/454>, <https://lkml.org/lkml/2014/3/19/457>,
<https://lkml.org/lkml/2014/3/19/456>

- Cleanups and fixes:

- <https://lkml.org/lkml/2014/1/25/44>, <https://lkml.org/lkml/2014/1/16/291>,
<https://lkml.org/lkml/2014/1/16/306>, <https://lkml.org/lkml/2014/1/16/292> (in 3.14)
 - <https://lkml.org/lkml/2014/2/14/234>

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

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 IPs, 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!

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 IPs, 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!!!

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?