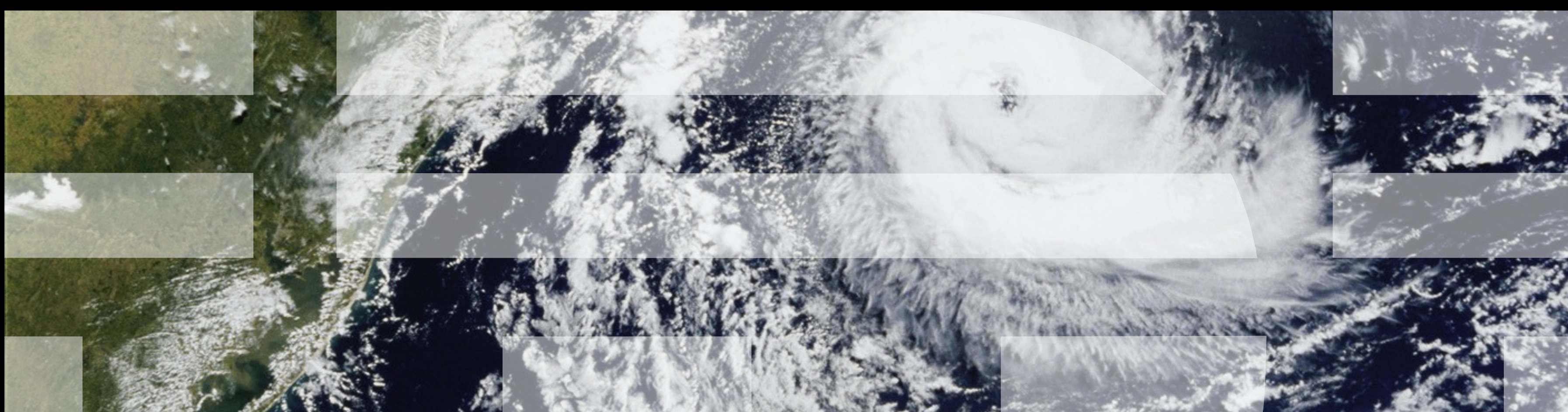
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center
Member, IBM Academy of Technology
Real Time Linux Workshop, Lugano, Switzerland October 30, 2013

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

(Now With Added Energy Efficiency!)



© 2013 IBM Corporation

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
- 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 (fun with “Hotel California” interrupts!)
- 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)

© 2013 IBM Corporation

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

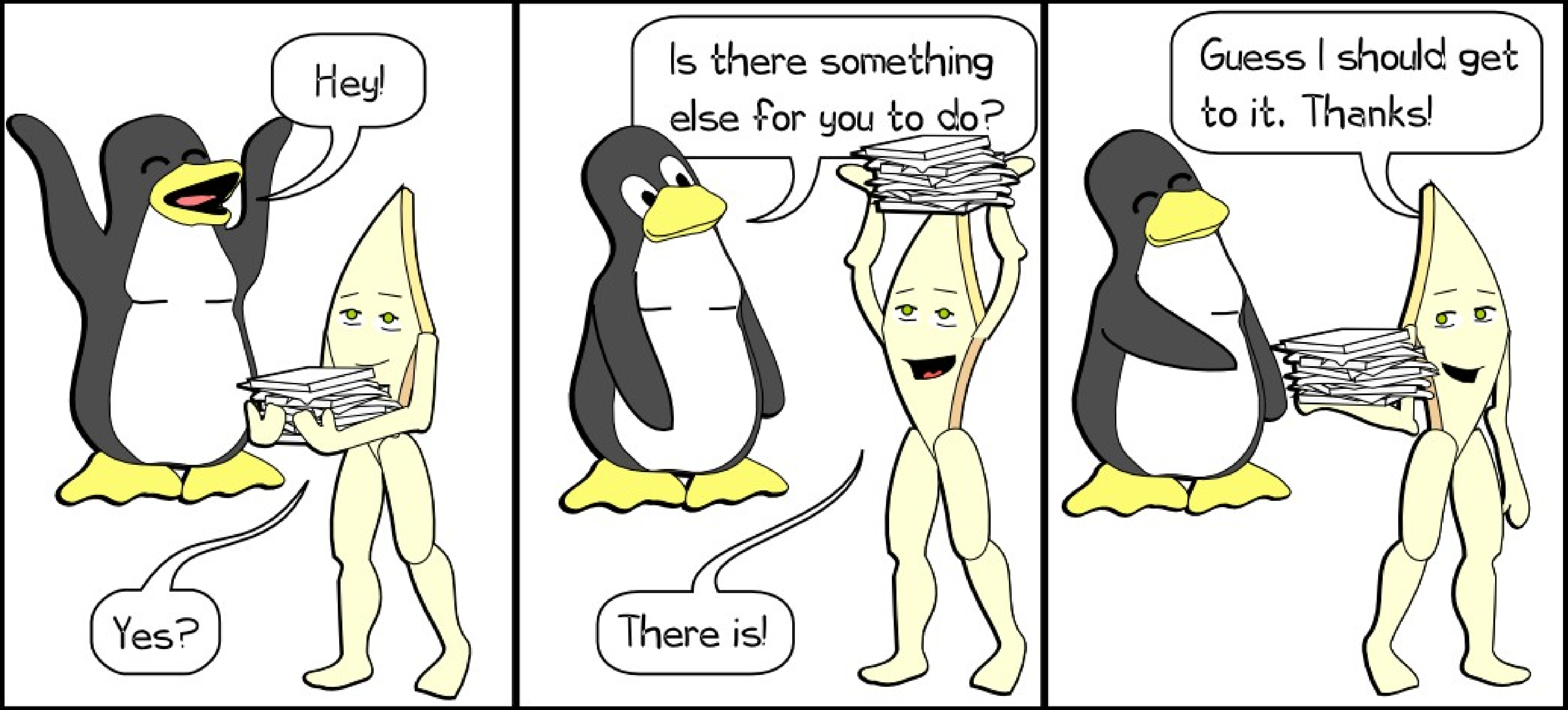
© 2013 IBM Corporation

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

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

© 2013 IBM Corporation

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



© 2013 IBM Corporation

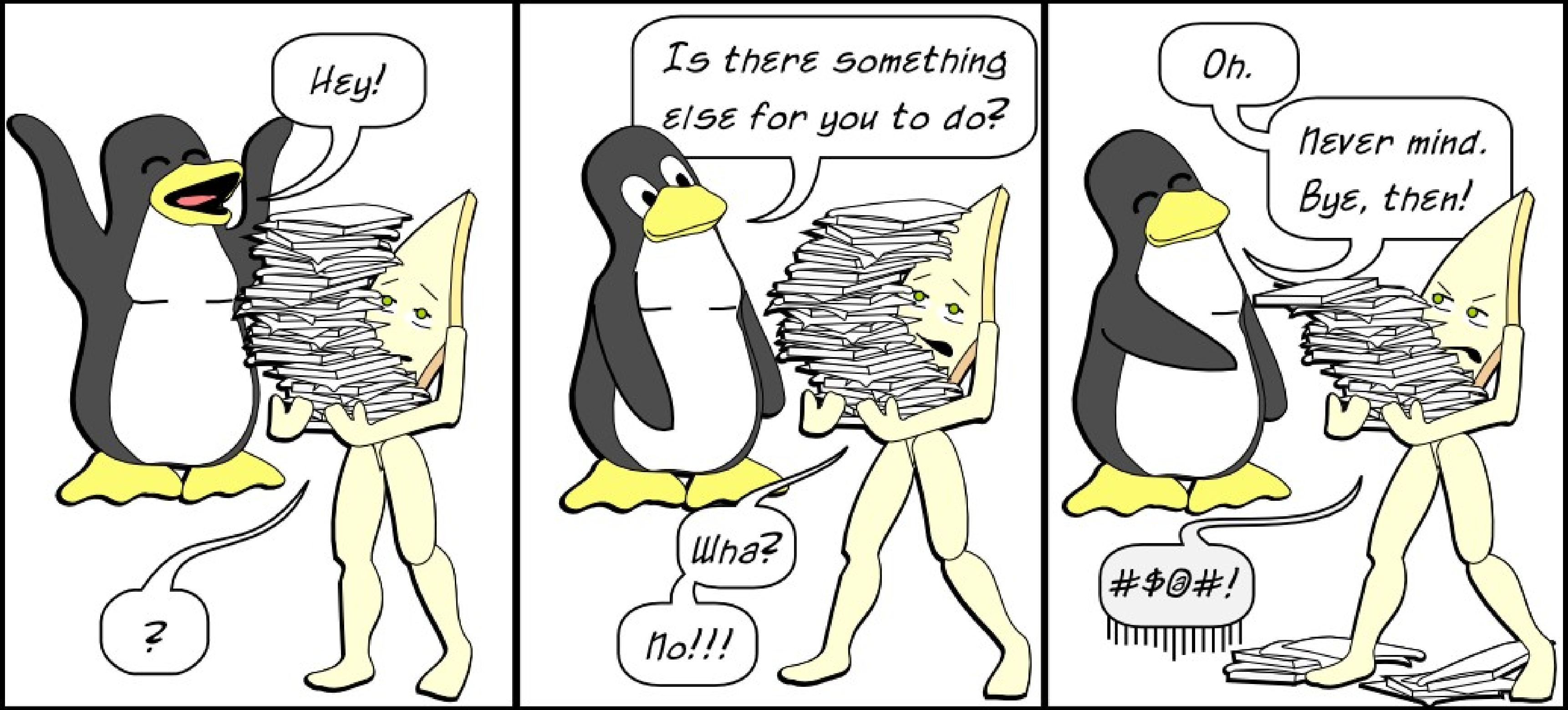
There Used To Be Things You Could Count On...

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

Whether you needed it or not.

© 2013 IBM Corporation

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



© 2013 IBM Corporation

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.

© 2013 IBM Corporation

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

© 2013 IBM Corporation

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

© 2013 IBM Corporation

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...

© 2013 IBM Corporation

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...

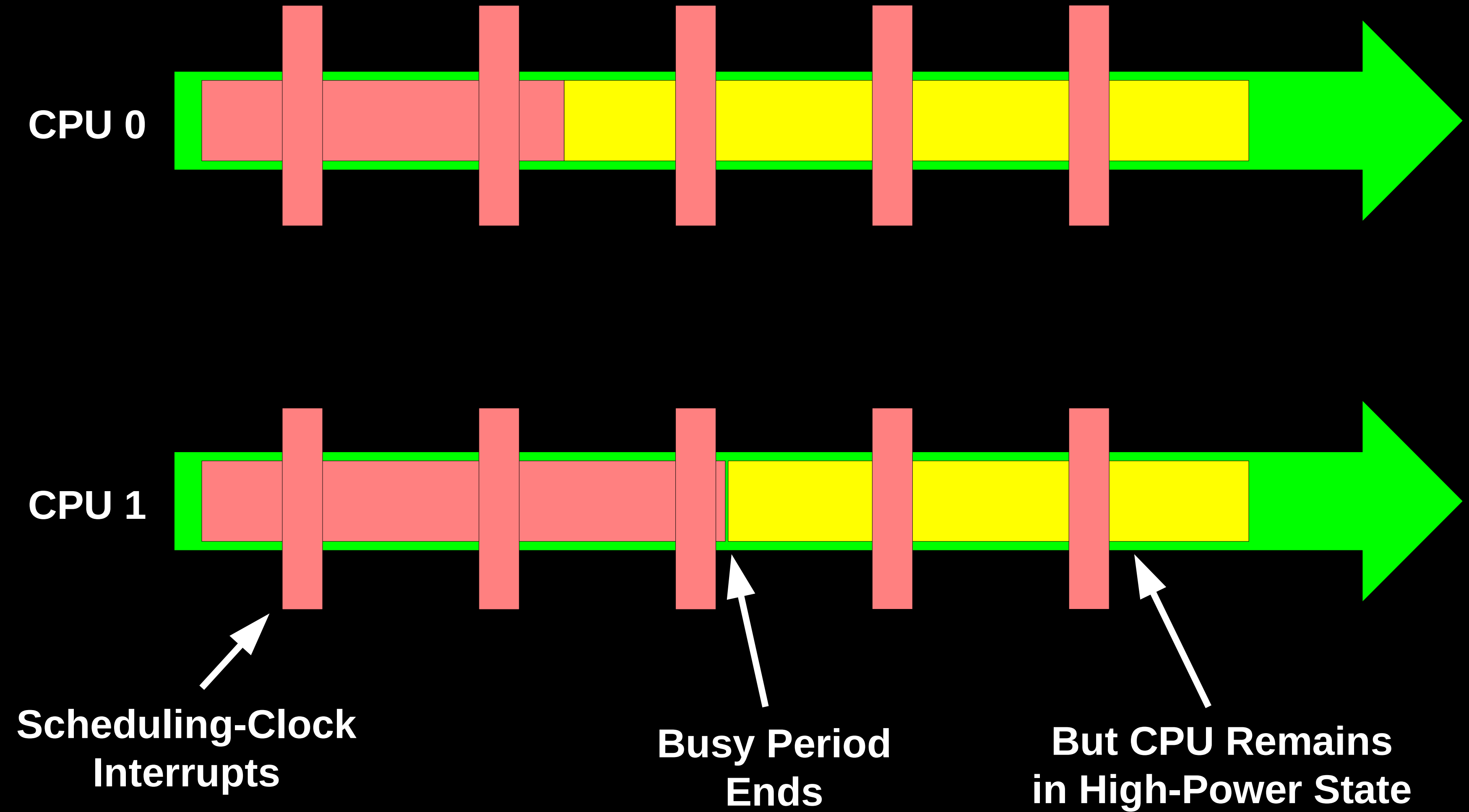
And, if your system was portable, a forklift.

© 2013 IBM Corporation

What We Need Instead...

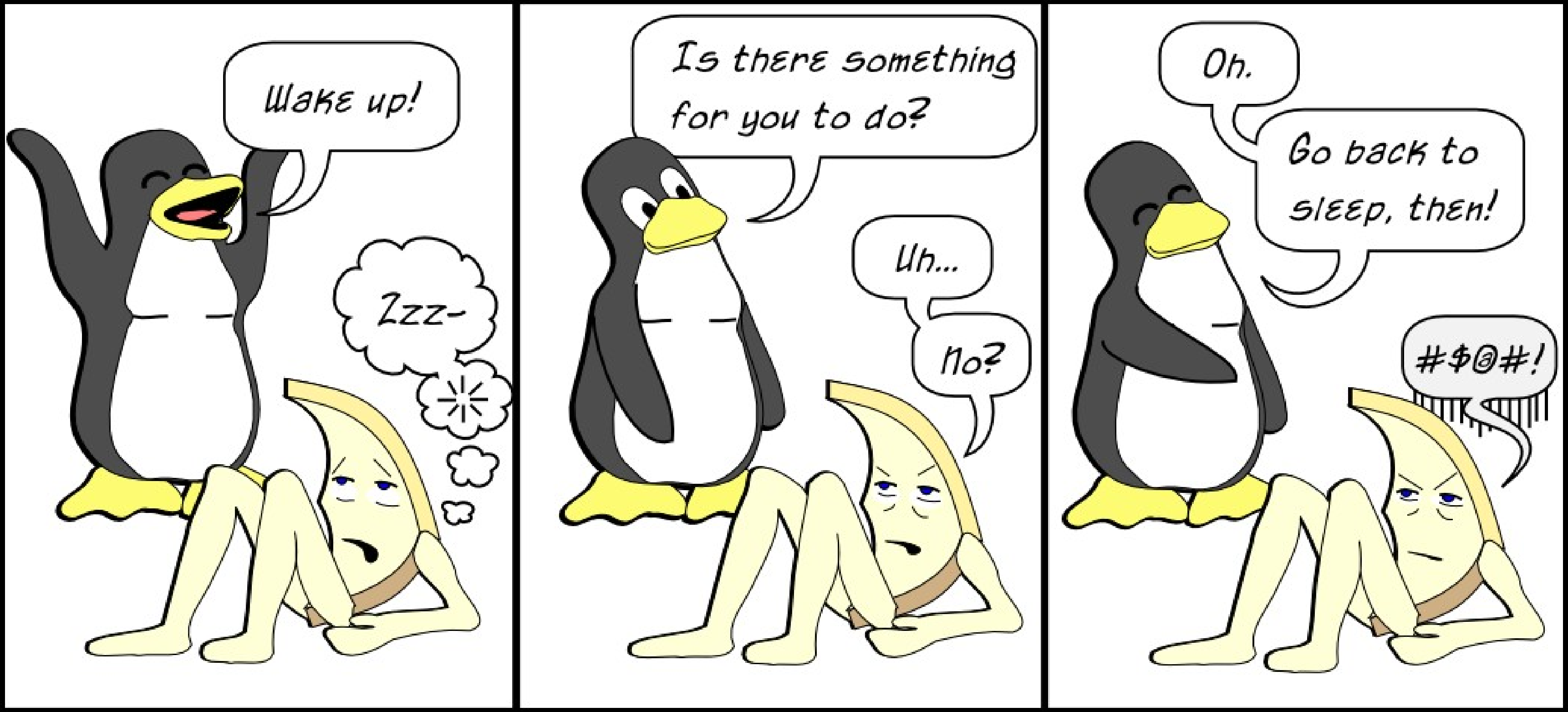
© 2013 IBM Corporation

Before Linux's dyntick-idle System



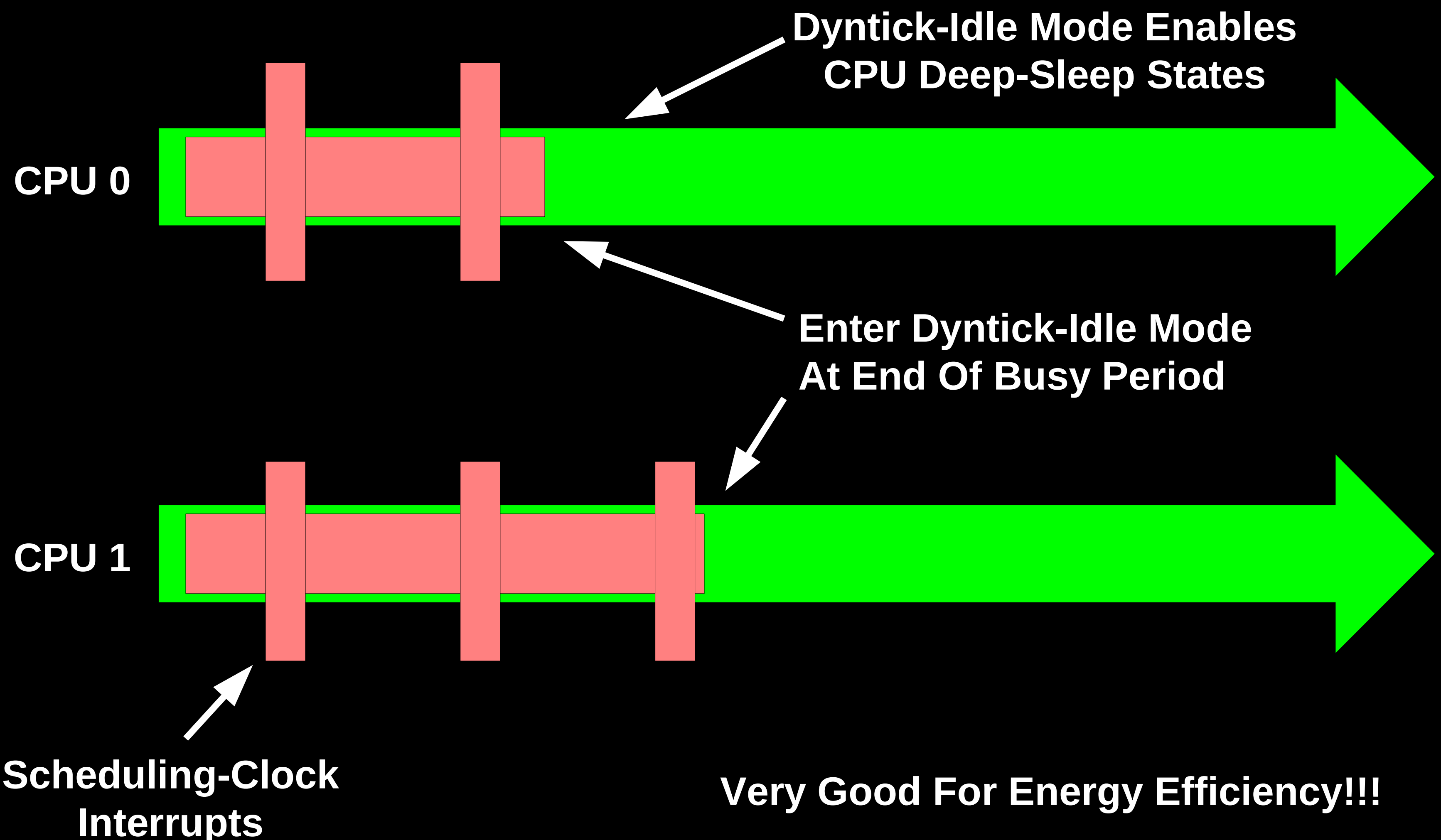
© 2013 IBM Corporation

Before Linux's dyntick-idle System



© 2013 IBM Corporation

Linux's dyntick-idle System



Very Good For Energy Efficiency!!!

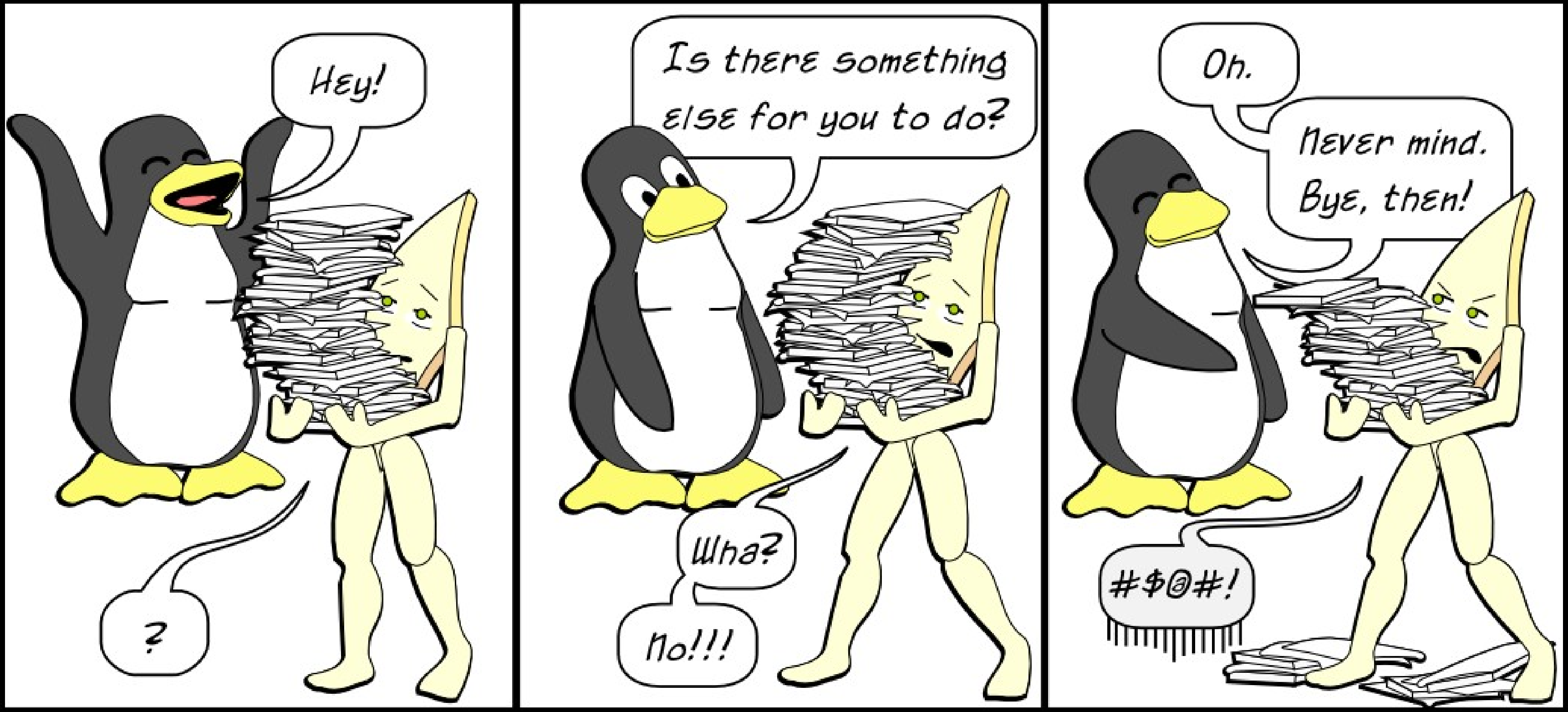
© 2013 IBM Corporation

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

© 2013 IBM Corporation

Also: Avoid Unnecessary Usermode Interrupts



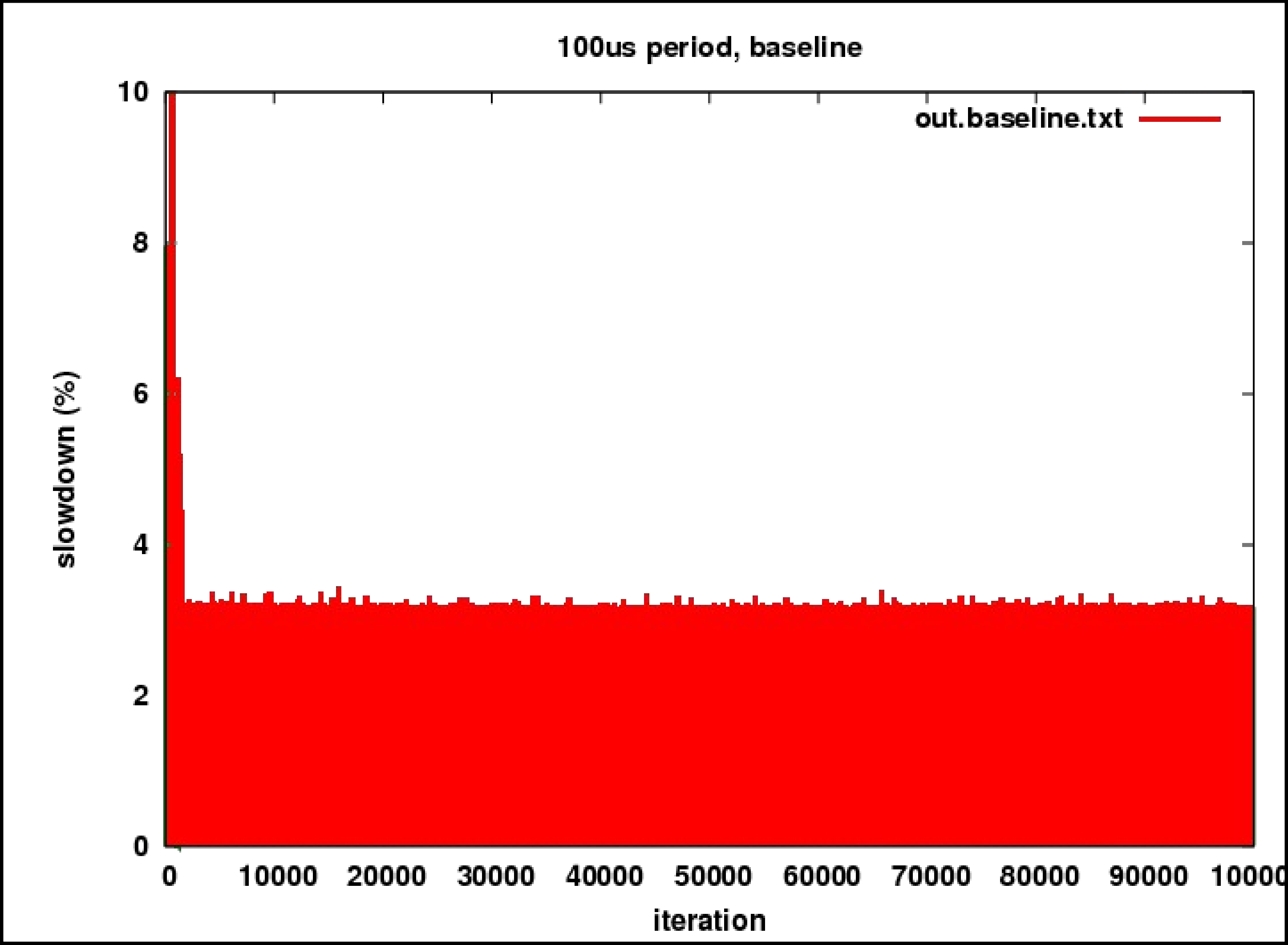
© 2013 IBM Corporation

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
- Josh Triplett prototyped CONFIG_NO_HZ_FULL 2009

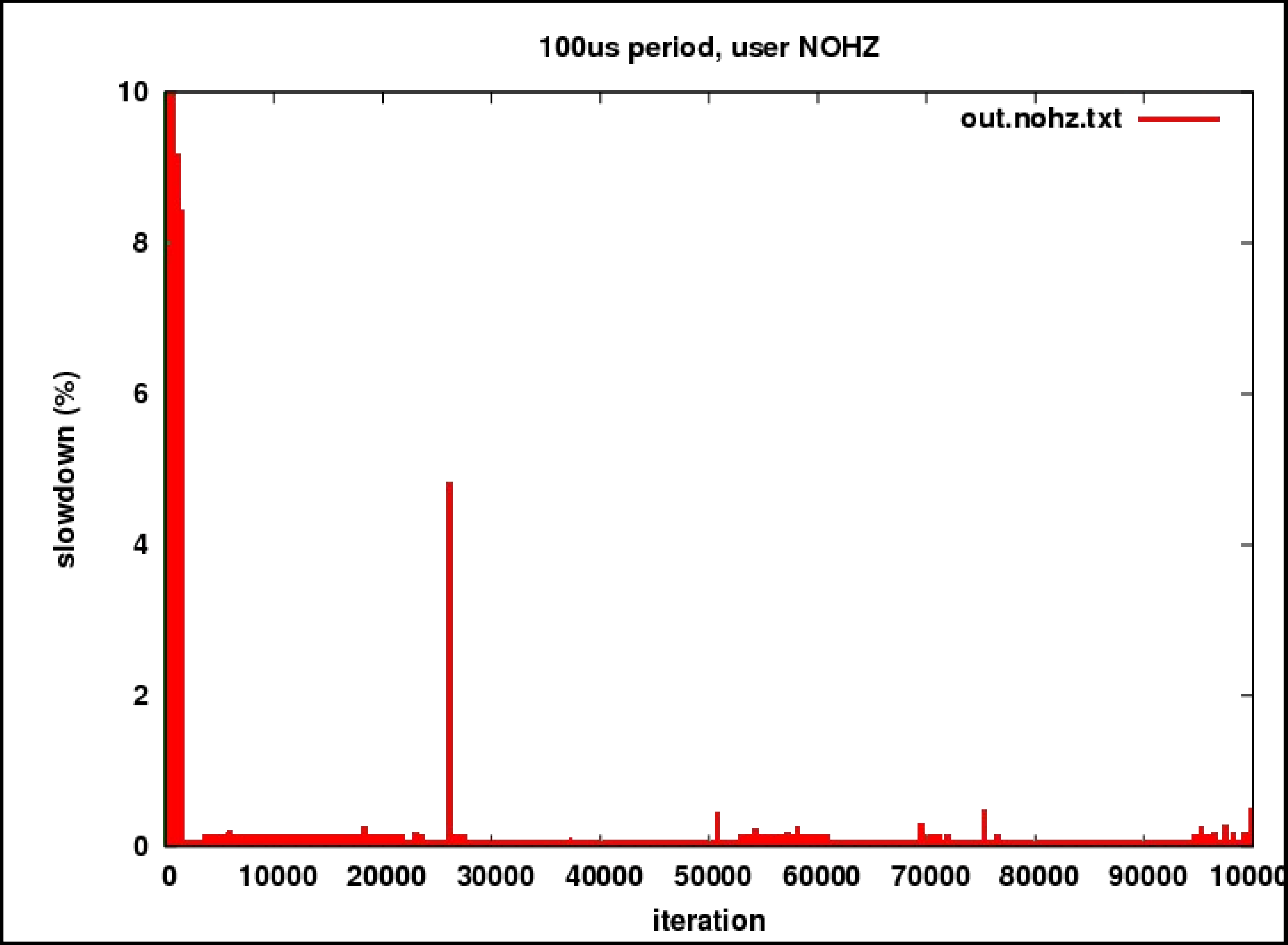
© 2013 IBM Corporation

Benchmark Results Before (Anton Blanchard)



© 2013 IBM Corporation

Benchmark Results After (Anton Blanchard)



Well worth going after...

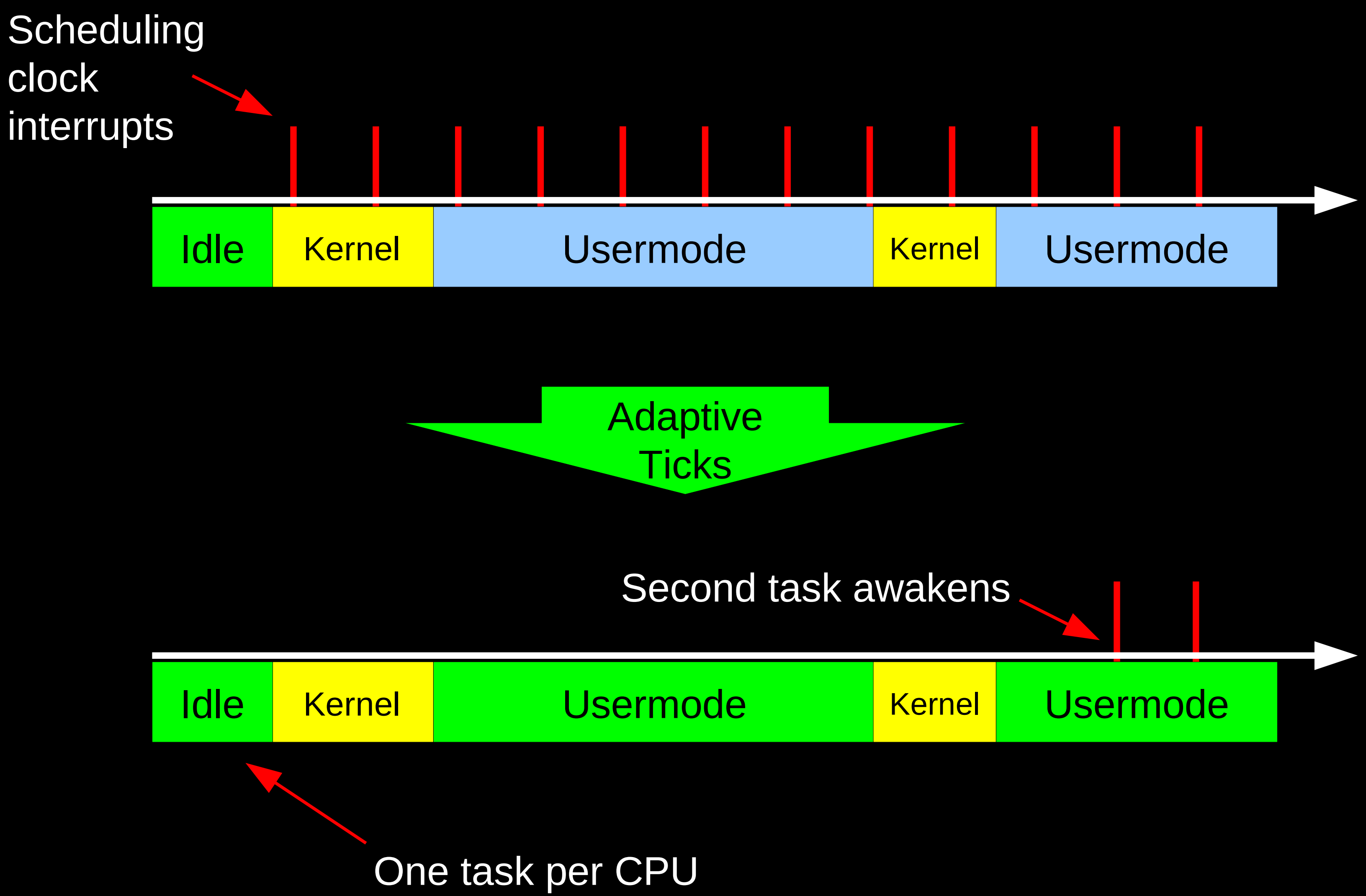
© 2013 IBM Corporation

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

© 2013 IBM Corporation

How Does It Work?



© 2013 IBM Corporation

Other Than Energy Efficiency, Looks Great!!!

- But what is the problem with energy efficiency?

© 2013 IBM Corporation

Energy Efficiency and Timekeeping

- Hardware oscillators drift
- Requires periodic recalculation of time conversion parameters
  - Otherwise user applications get bad time data
- One special case

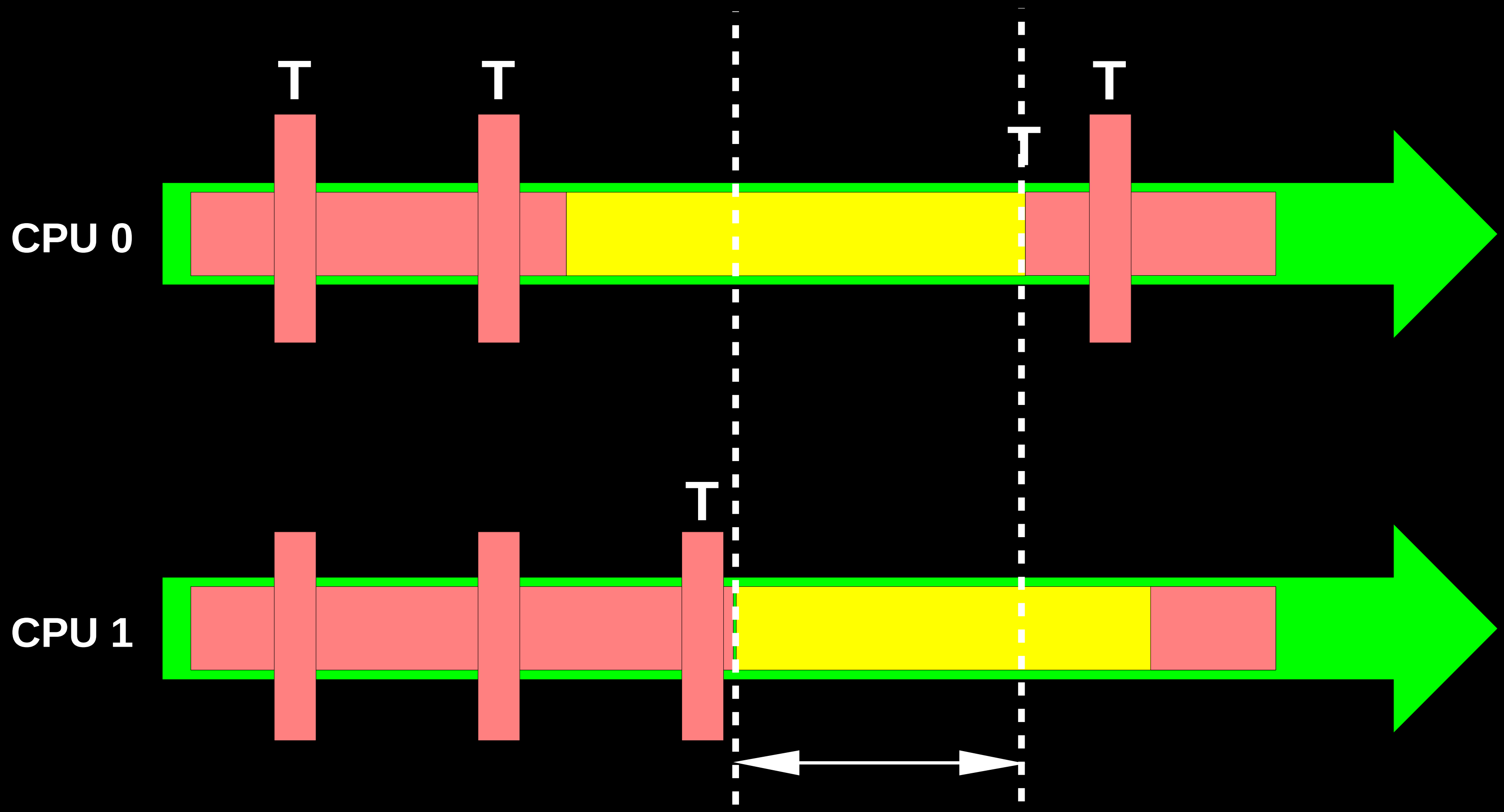
© 2013 IBM Corporation

Energy Efficiency and Timekeeping

- Hardware oscillators drift
- Requires periodic recalculation of time conversion parameters
  - 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

© 2013 IBM Corporation

Energy Efficiency, Timekeeping, and Idle

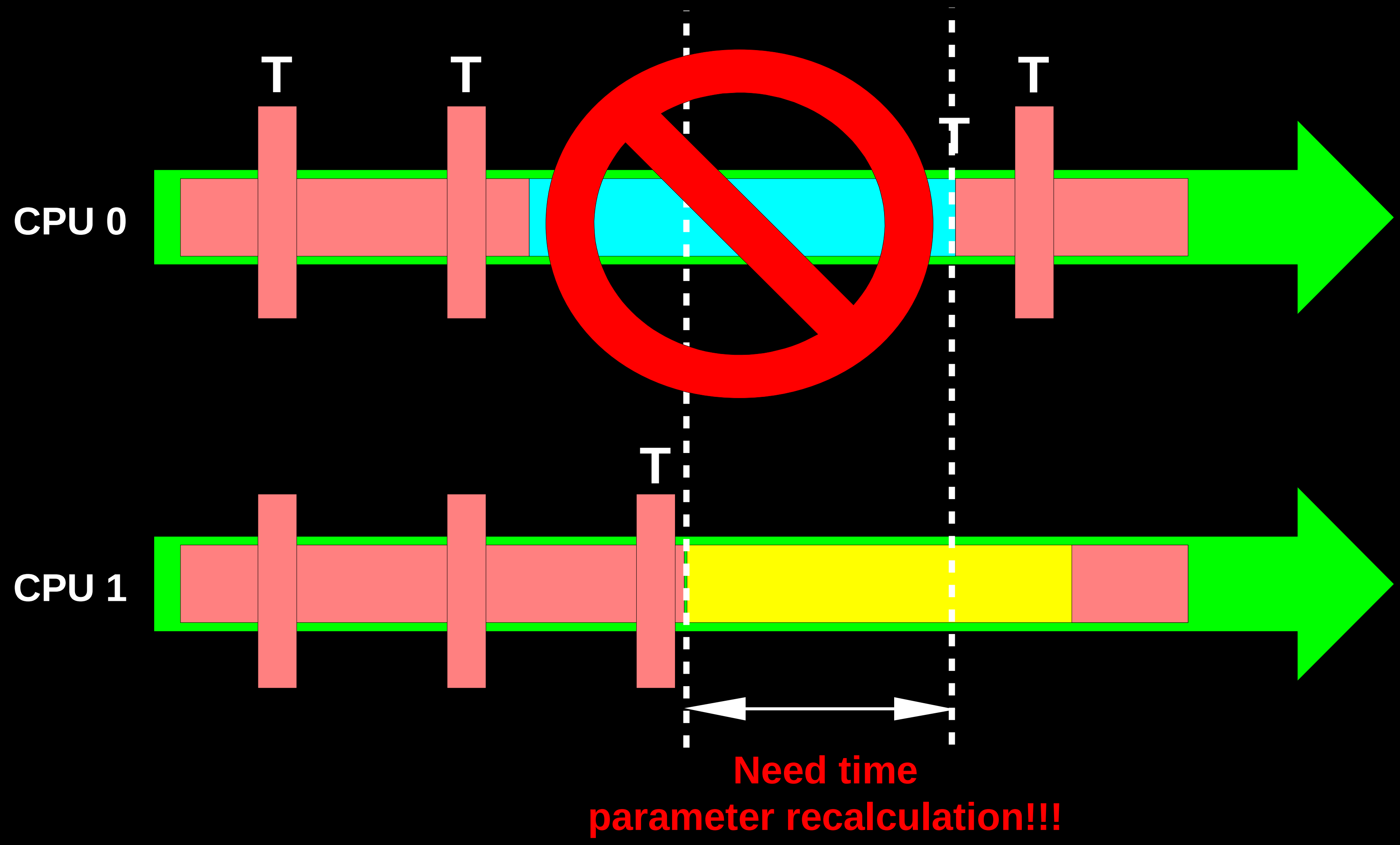


© 2013 IBM Corporation

But If Running Userspace, Need Recalculation

© 2013 IBM Corporation

Energy Efficiency, Timekeeping, and Userspace



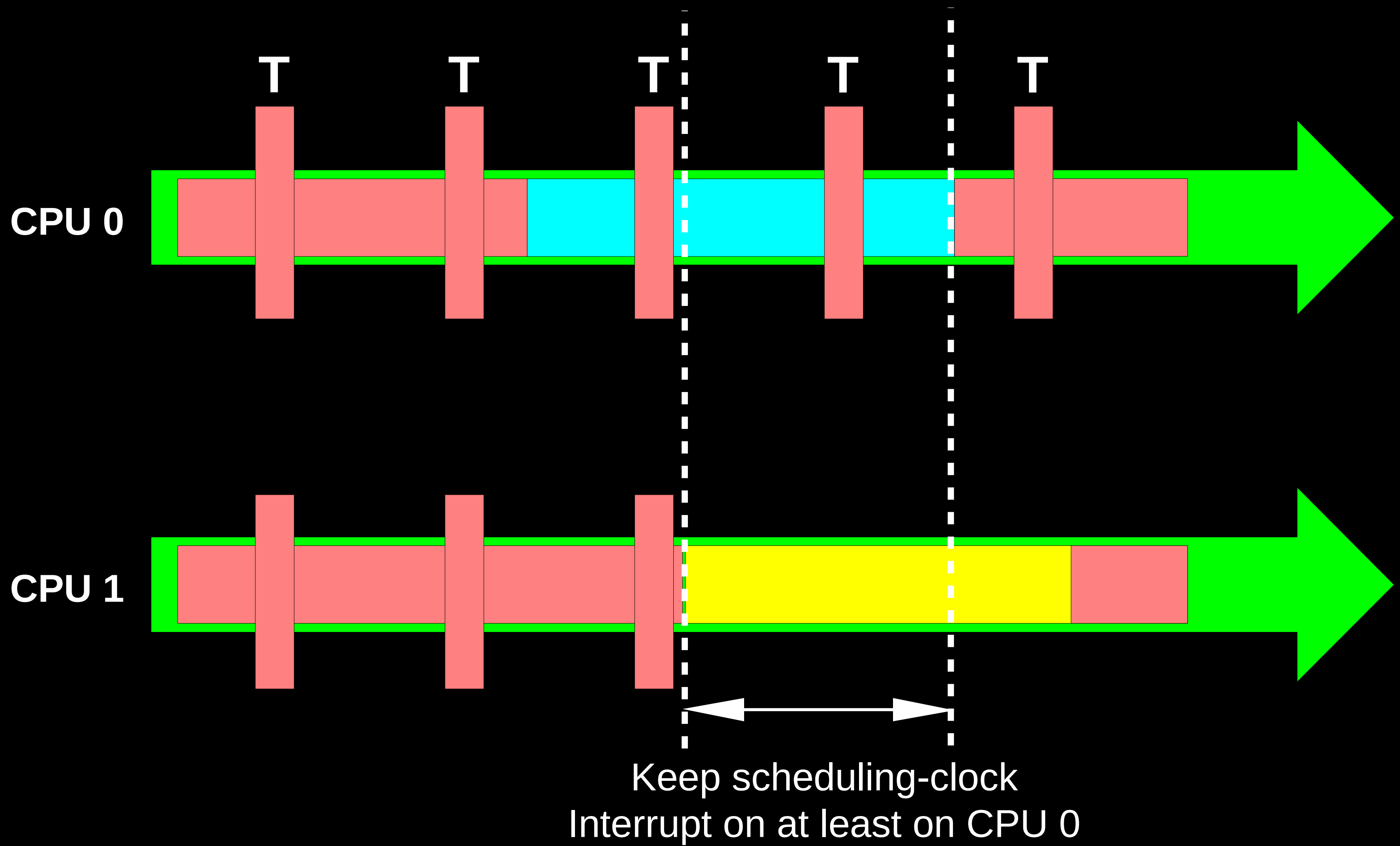
© 2013 IBM Corporation

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

© 2013 IBM Corporation

Shut Down *Almost* All Scheduling-Clock Interrupts



© 2013 IBM Corporation

The Battery-Powered Embedded Folks Not Happy...

© 2013 IBM Corporation

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

© 2013 IBM Corporation

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

© 2013 IBM Corporation

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 CPU wrongly concludes that the entire system is idle!!!**

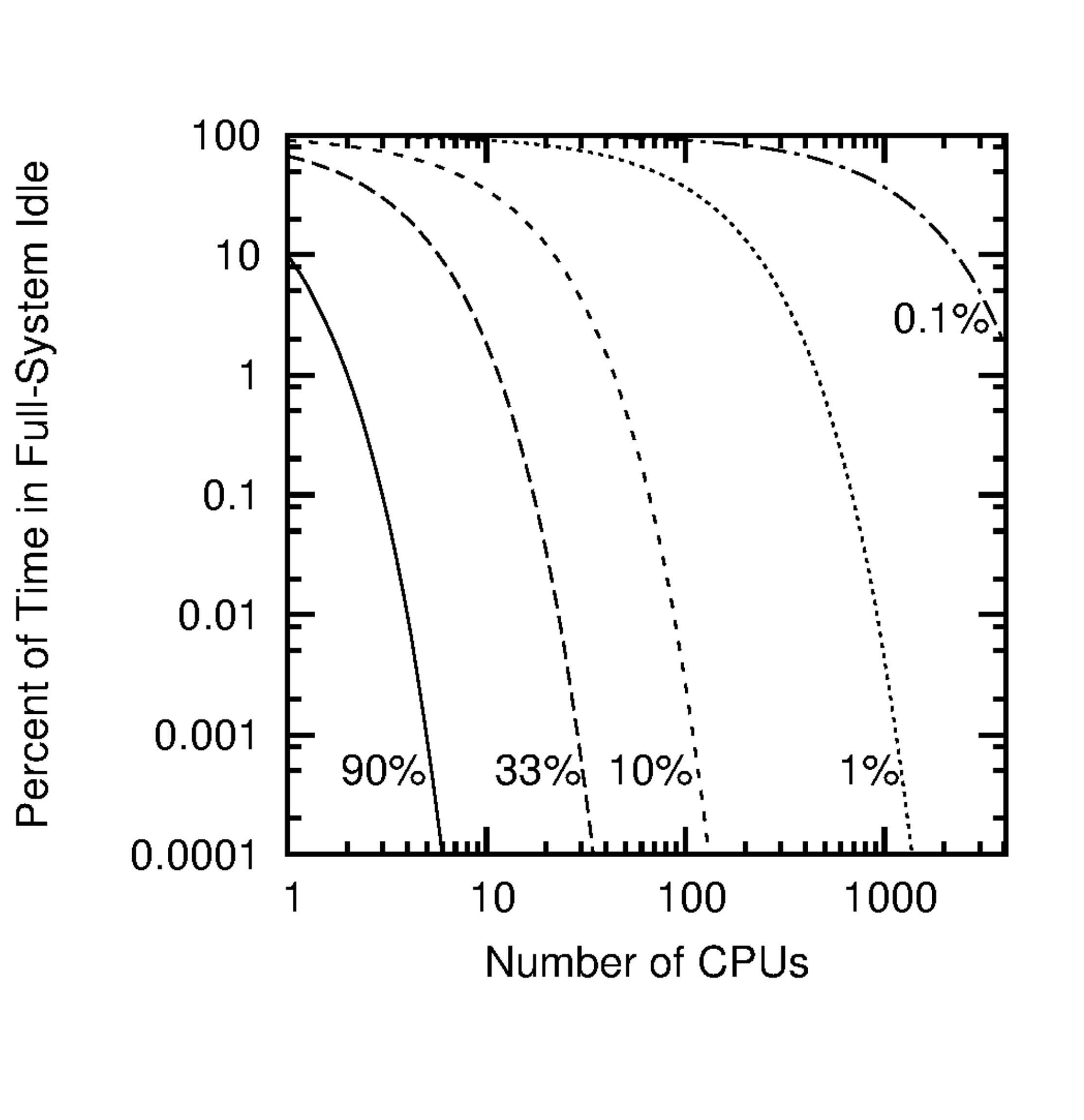
© 2013 IBM Corporation

How To Efficiently Detect Full-System Idle?

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

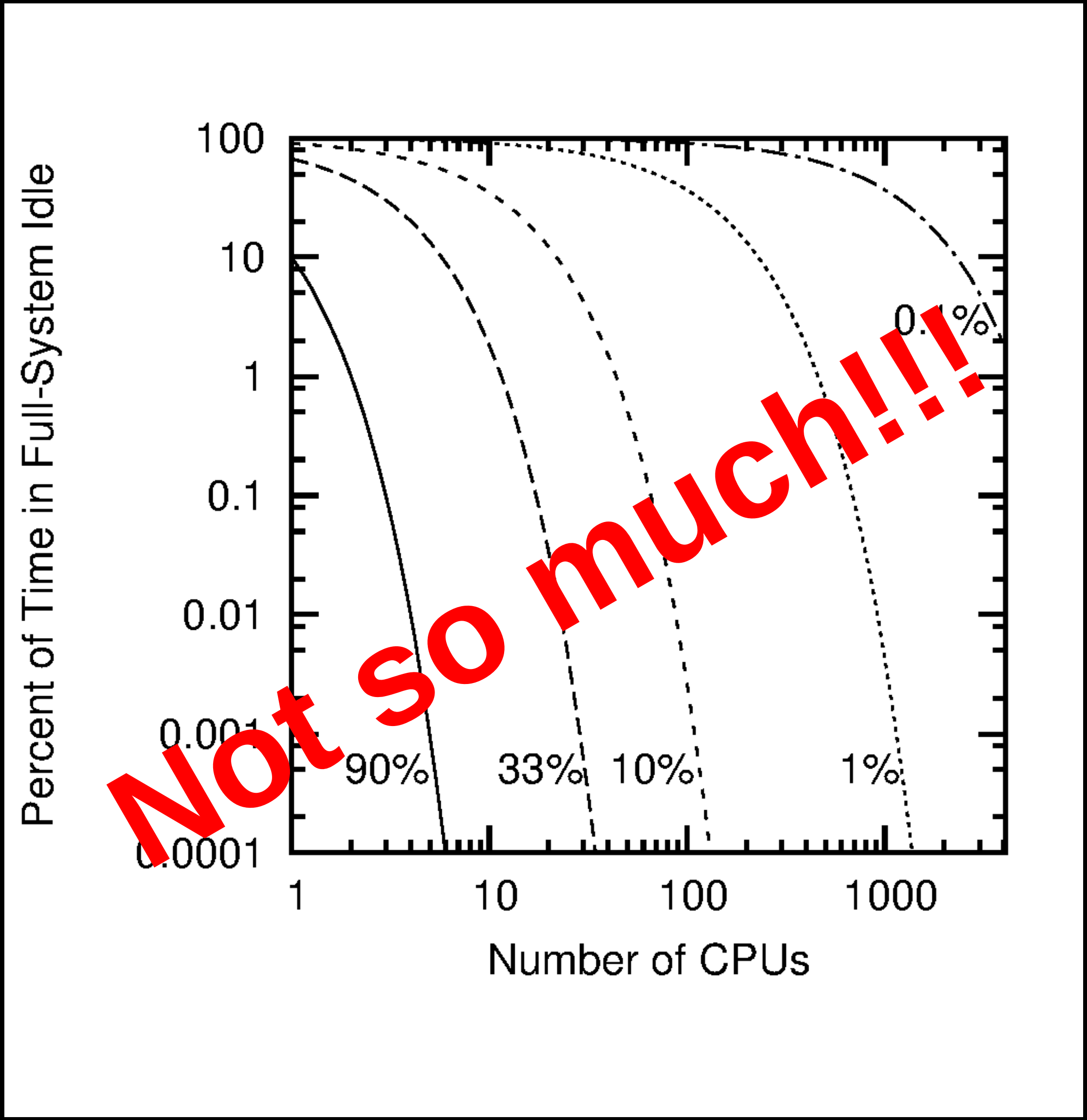
© 2013 IBM Corporation

Opportunistic Idle on Large Systems?



© 2013 IBM Corporation

Opportunistic Idle on Large Systems?



© 2013 IBM Corporation

How To Efficiently Detect Full-System Idle?

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

© 2013 IBM Corporation

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!!!

© 2013 IBM Corporation

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 variables 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, noting quiescent states on their behalf
  - Also track last time each CPU went idle

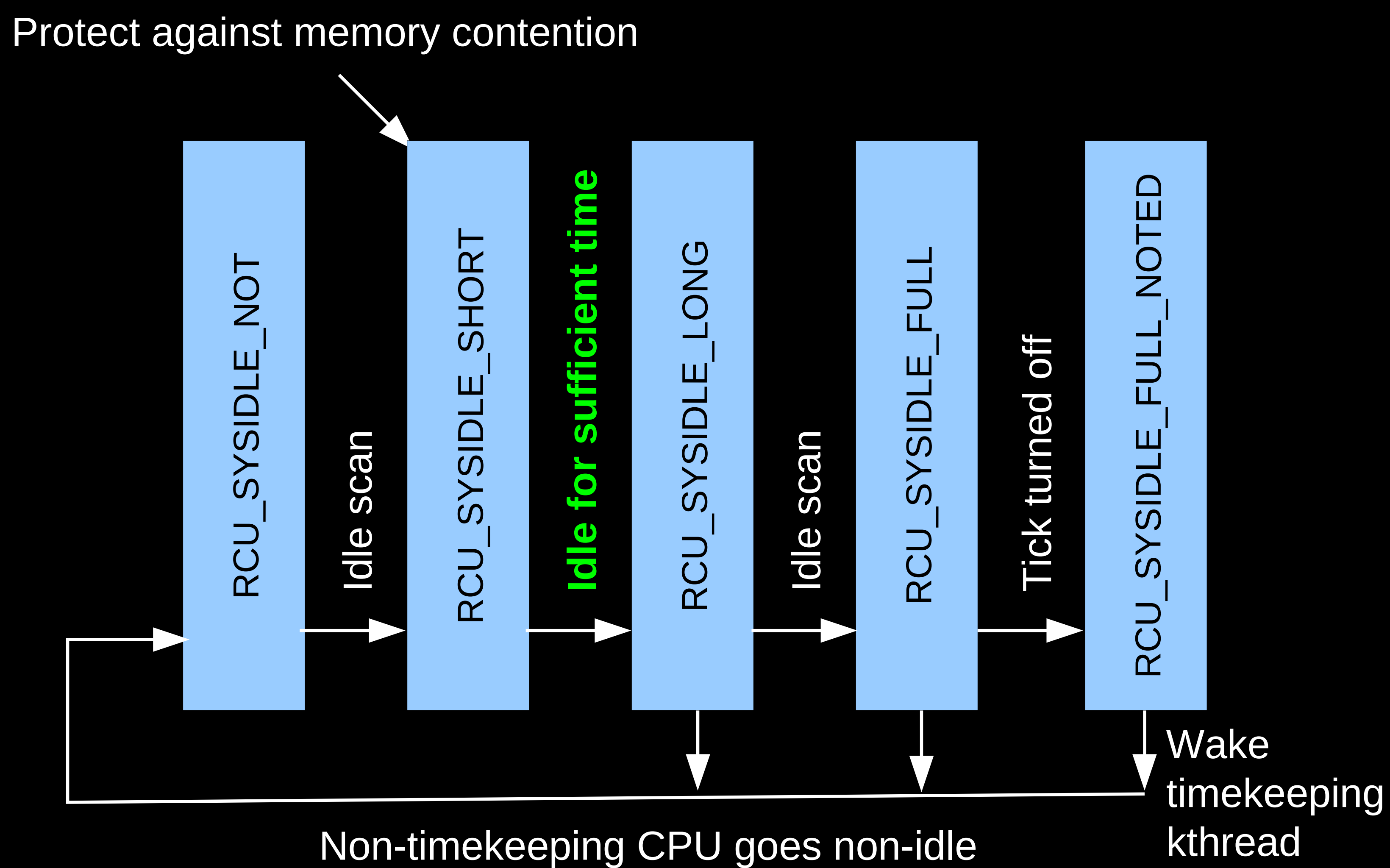
© 2013 IBM Corporation

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 ktheads sees `RCU_SYSIDLE_FULL` state.

© 2013 IBM Corporation

Full-System-Idle State Machine Schematic



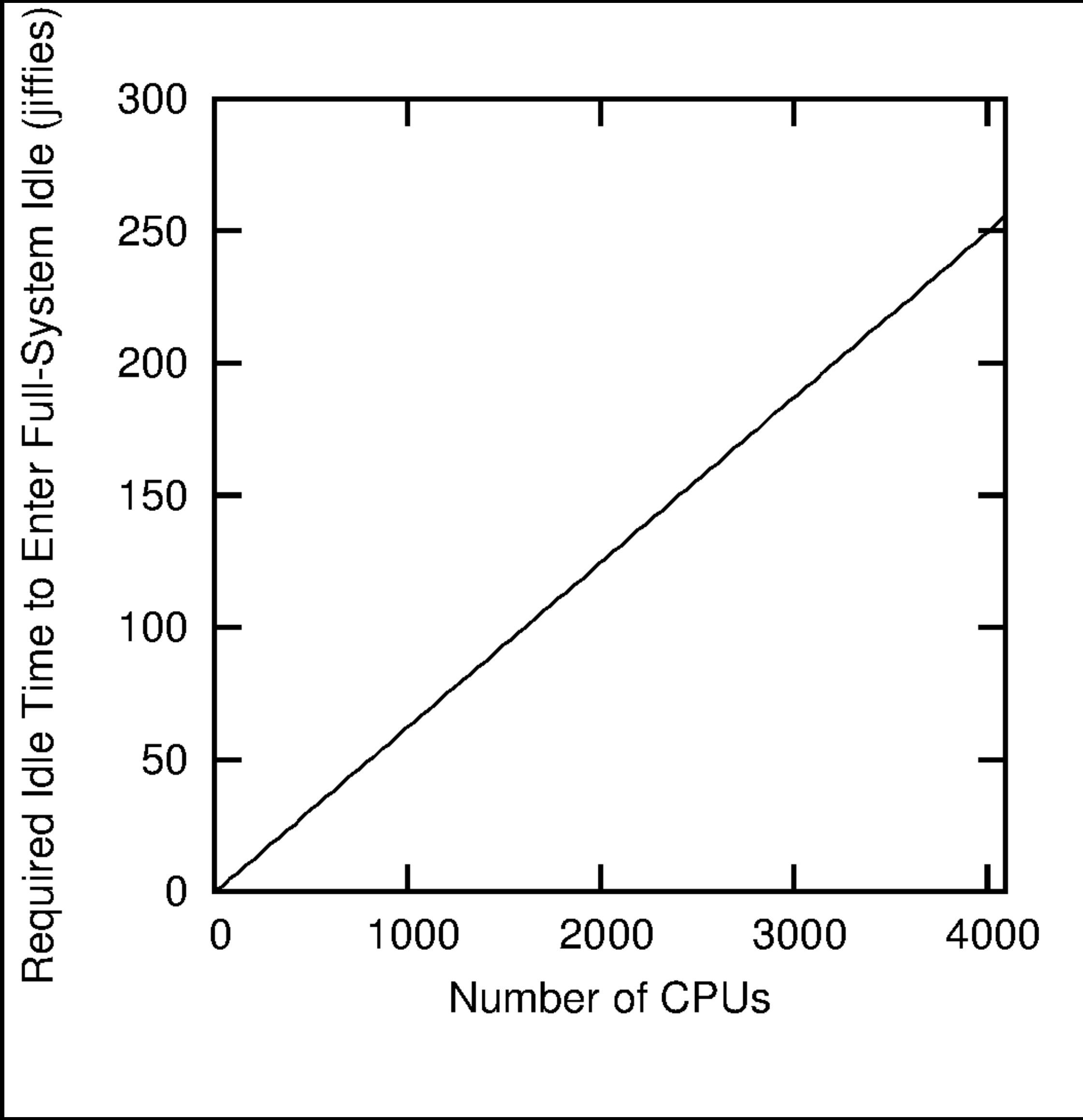
© 2013 IBM Corporation

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

© 2013 IBM Corporation

How Long Is “Sufficiently Long”, Anyway?



© 2013 IBM Corporation

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 will force scheduling-clock interrupts back on unless it sees the current state as either RCU_SYSIDLE_NOT or 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!

© 2013 IBM Corporation

Avoiding Non-Idle Races

- Any CPU can drive it 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 after updating 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 timekeeping (or grace-period) kthread will see nonidle on next scan

© 2013 IBM Corporation

Sounds Complex! Did You Mechanically Verify This?

© 2013 IBM Corporation

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 namespacet&, 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 on the way...

- Maybe fall back on Promela/spin
  - In addition to reviews, stress tests, and informal proof of correctness
- Murphy says that there are bugs!

© 2013 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/

© 2013 IBM Corporation

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
- How-to info for kthreads: Documentation/kernel-per-CPU-kthreads.txt
- Available in 3.10-3.12

© 2013 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
  - And can maintain energy efficiency as well!

© 2013 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
  - And can maintain energy efficiency as well!
- Extending Linux's reach further into extreme computing!!!

© 2013 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.

© 2013 IBM Corporation

Questions?

© 2013 IBM Corporation