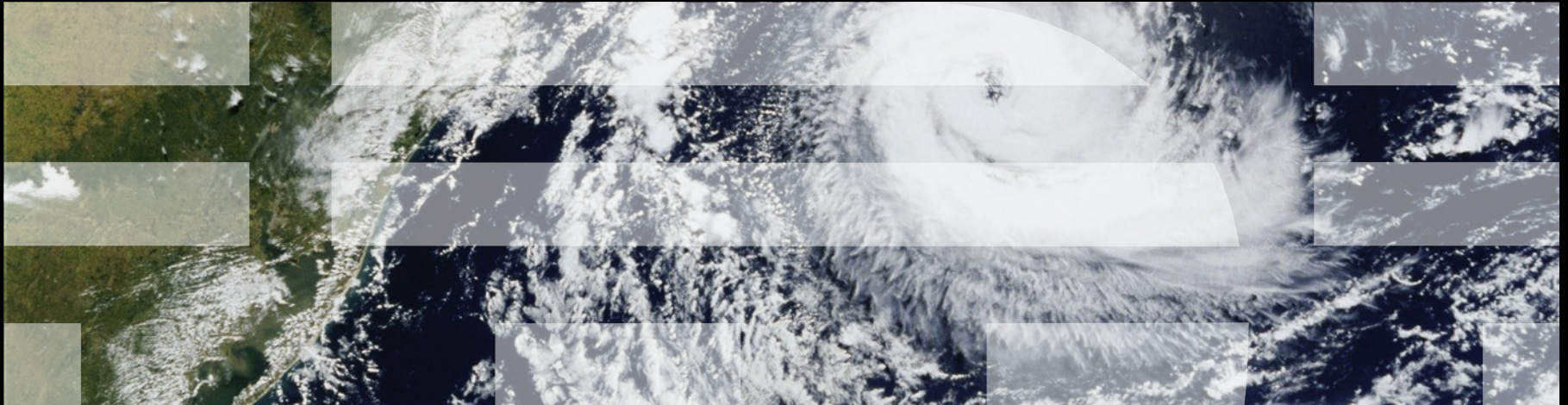


Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center (Linaro)

10 July 2012



Real-Time Response on Multicore Systems: *It is Bigger Than You Think*



Experience With Real-Time Computing

Experience With Real-Time Computing

- Early 1980s: soft real-time on 8-bit and 16-bit systems
 - Building-control/energy-management system (bare metal z80, single processor, deadlines of 1-2 seconds, penalty: exploding transformers)
 - Card-key security system (RT-11 on PDP-11, single processor, deadlines of a few seconds, penalty: user gives up)
 - Acoustic navigation system (BSD 2.8 on PDP-11, single processor, deadlines of a few seconds, penalty: random transponder commands)

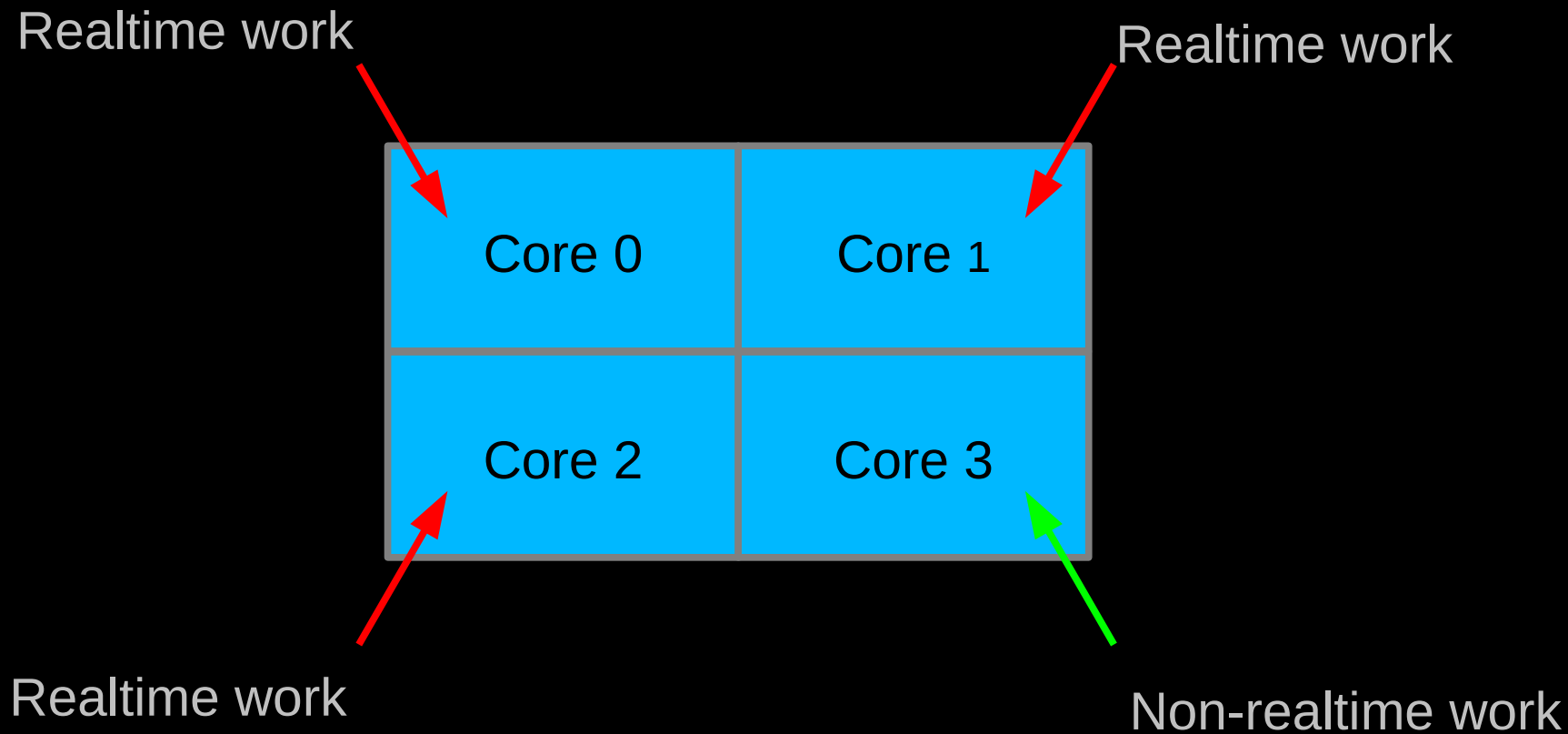
Experience With Real-Time Computing

- Early 1980s: soft real-time on 8-bit and 16-bit systems
 - Building-control/energy-management system (bare metal z80, single processor, deadlines of 1-2 seconds, penalty: exploding transformers)
 - Card-key security system (RT-11 on PDP-11, single processor, deadlines of a few seconds, penalty: user gives up)
 - Acoustic navigation system (BSD 2.8 on PDP-11, single processor, deadlines of a few seconds, penalty: random transponder commands)
- So what have I done with real time ... lately?

Experience With Real-Time Computing

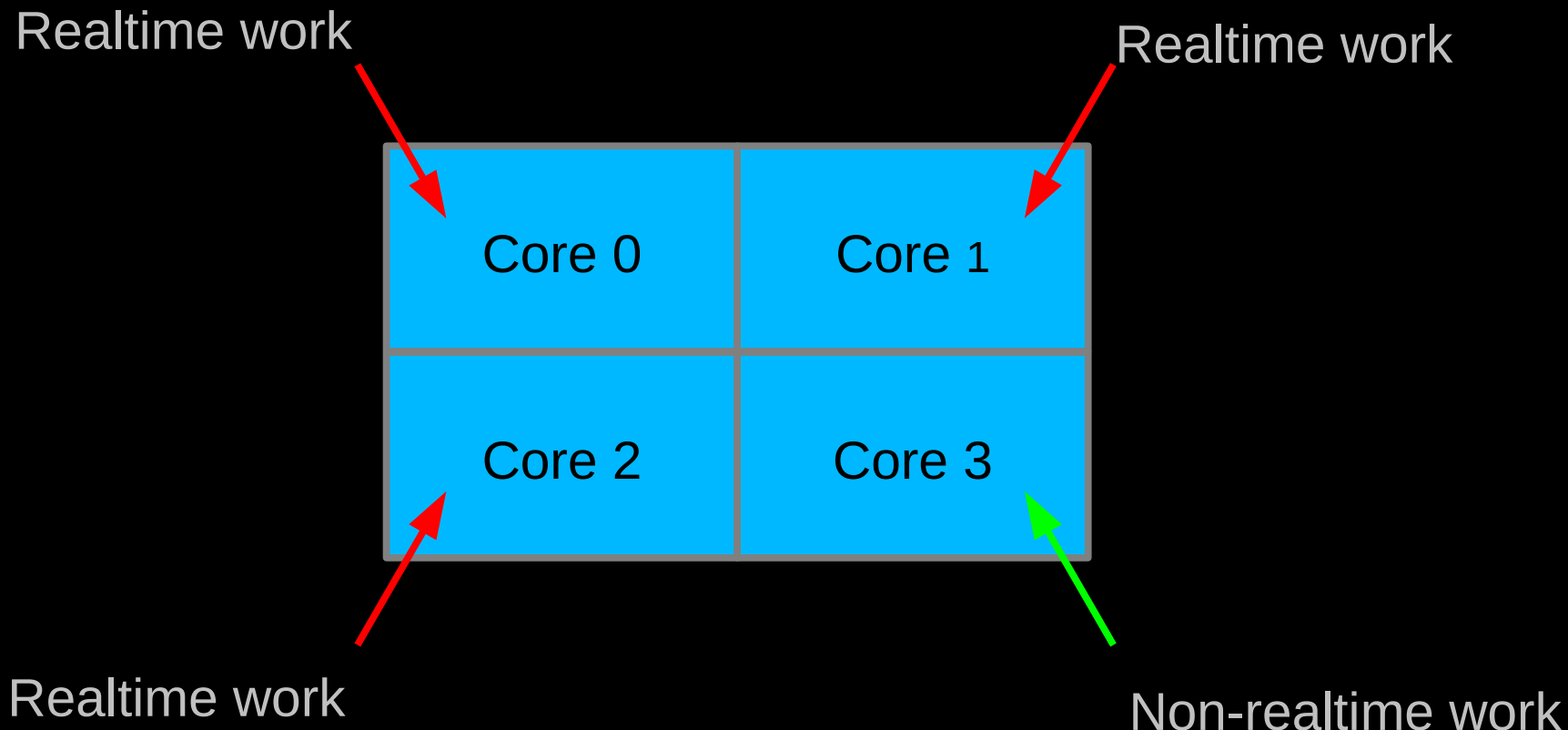
- Early 1980s: soft real-time on 8-bit and 16-bit systems
 - Building-control/energy-management system (bare metal z80, single processor, deadlines of 1-2 seconds, penalty: exploding transformers)
 - Card-key security system (RT-11 on PDP-11, single processor, deadlines of a few seconds, penalty: user gives up)
 - Acoustic navigation system (BSD 2.8 on PDP-11, single processor, deadlines of a few seconds, penalty: random transponder commands)
- So what have I done with real time ... lately?
- Early 2000s: Lots of requests for “real-time Linux”
 - IBM response: Linux does not meet your requirements. No bid.

2004: Prototype Multi-Core ARM Chip!!!



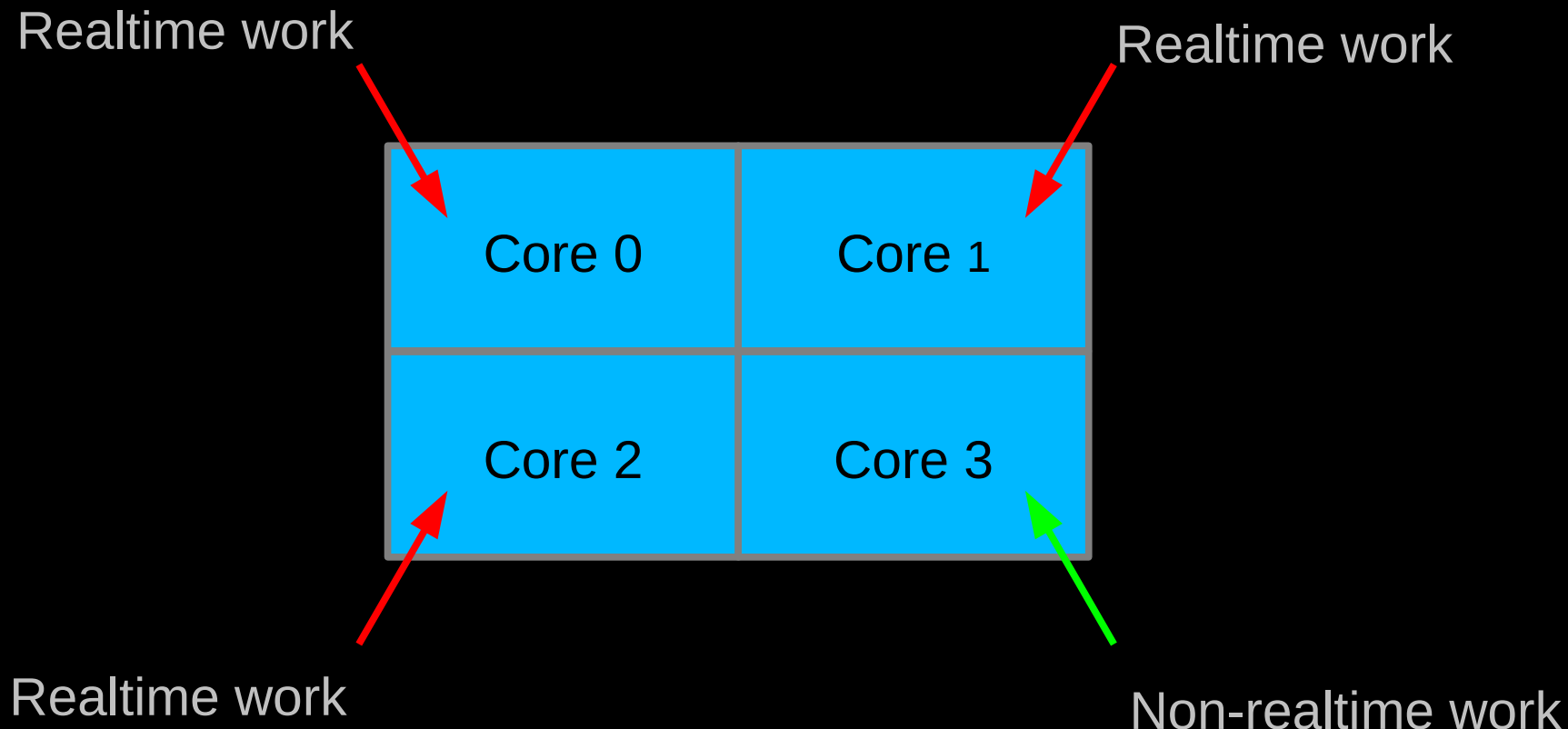
Submitted simple patch to Linux-kernel mailing list in 2004...

2004: Prototype Multi-Core ARM Chip!!!



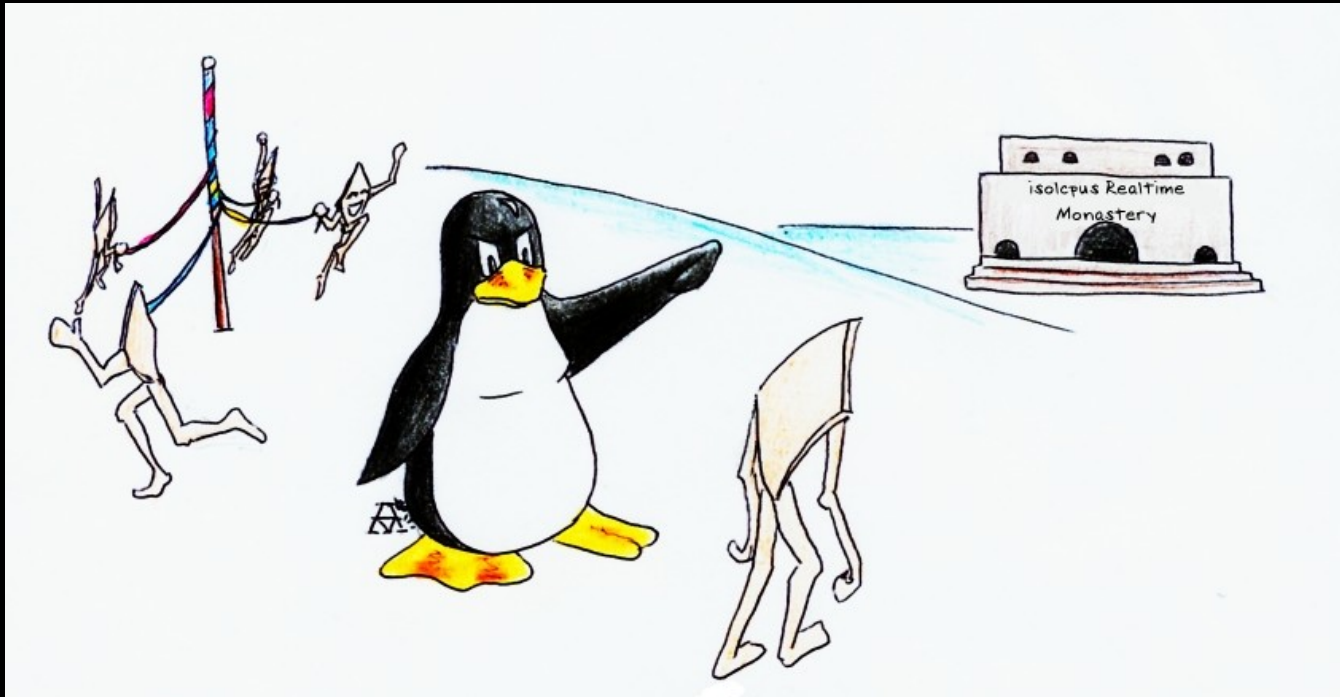
*Submitted simple patch to Linux-kernel mailing list in 2004...
The reception was not positive: PREEMPT_RT had started.*

2004: Prototype Multi-Core ARM Chip!!!

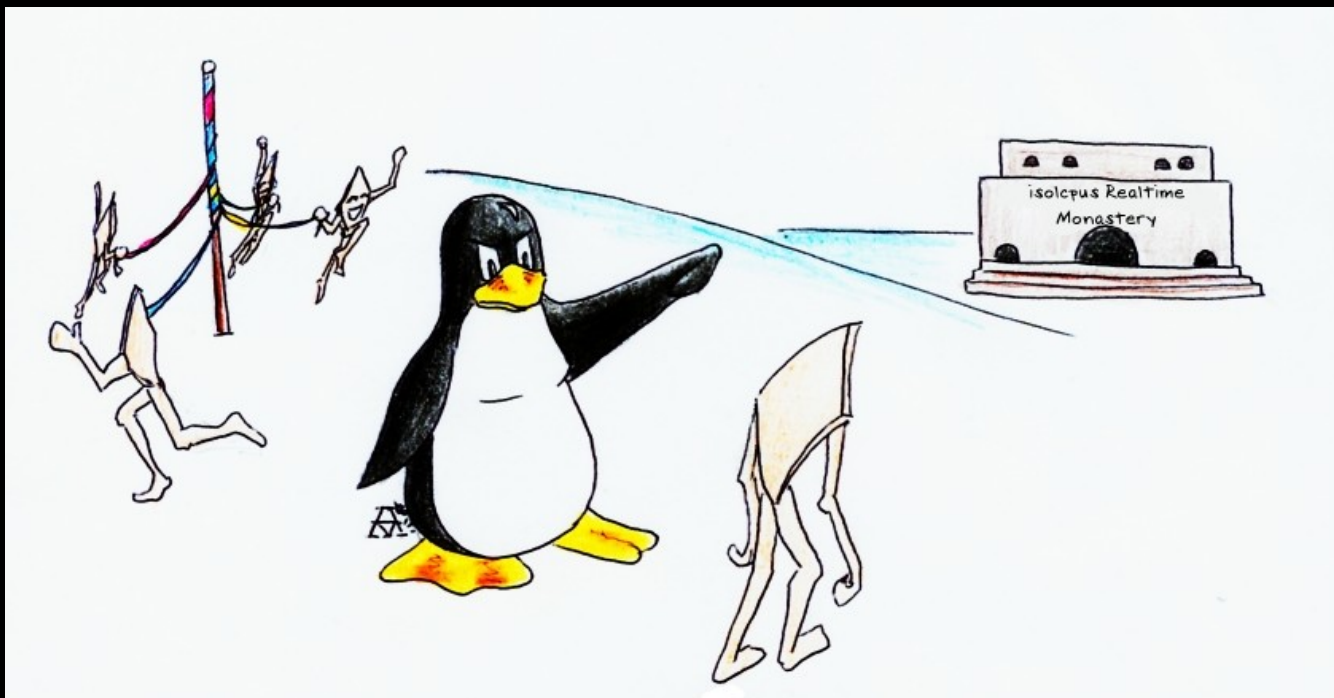


*Submitted simple patch to Linux-kernel mailing list in 2004...
The reception was not positive: PREEMPT_RT had started.
But I did convince my VP that real-time Linux was feasible.*

Resulting in This Situation for Real-Time Linux



Proposed This Approach to a Real Real-Time User

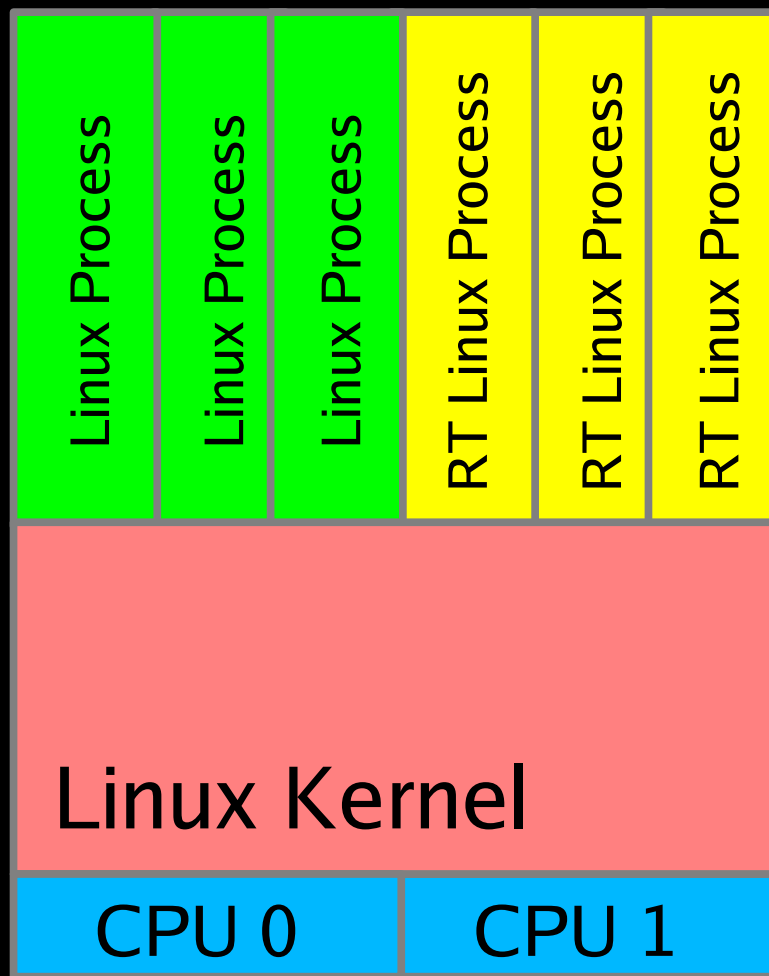


But my clever scheme failed to survive first contact with a real user...
Why?

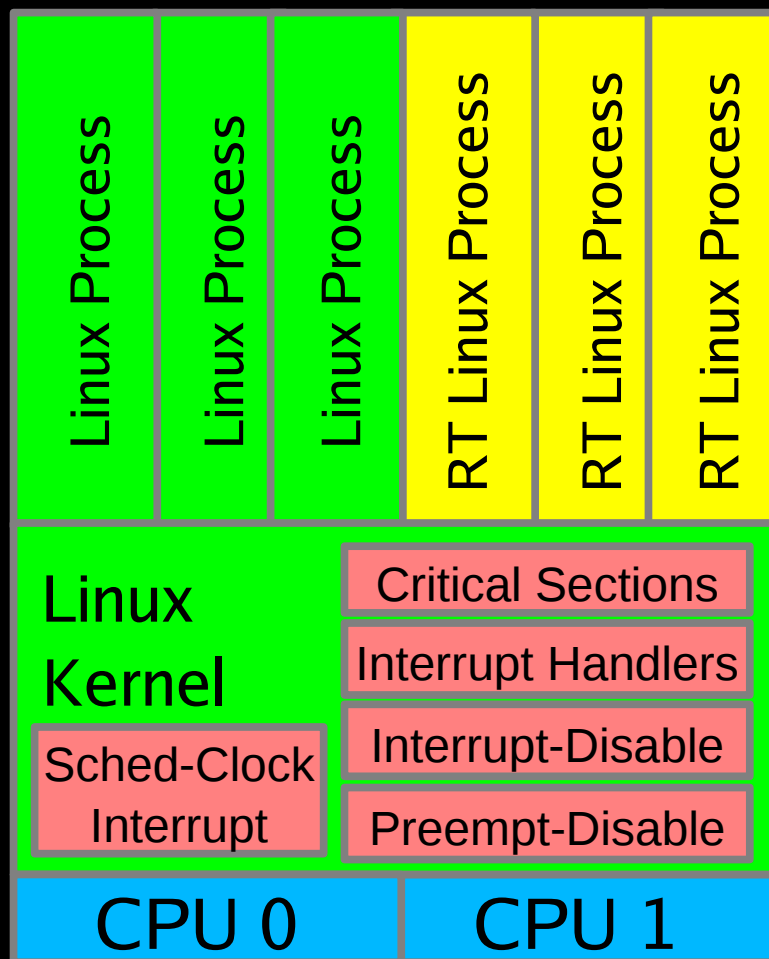
Fortunately, There Was This PREEMPT_RT Project...

Now called PREEMPT_RT_FULL
(But still the -rt patchset)

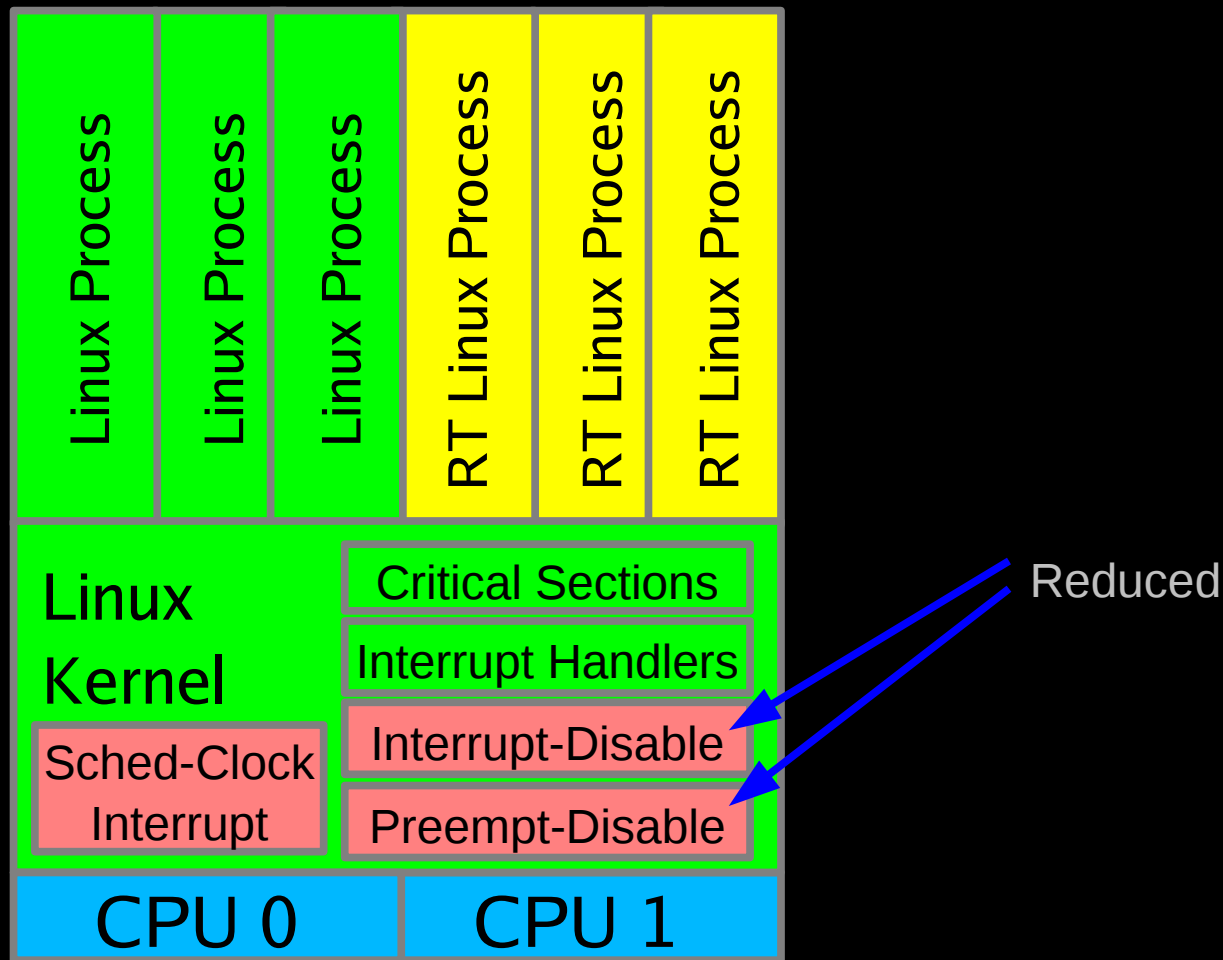
CONFIG_PREEMPT=n Kernel



CONFIG_PREEMPT=y Kernel

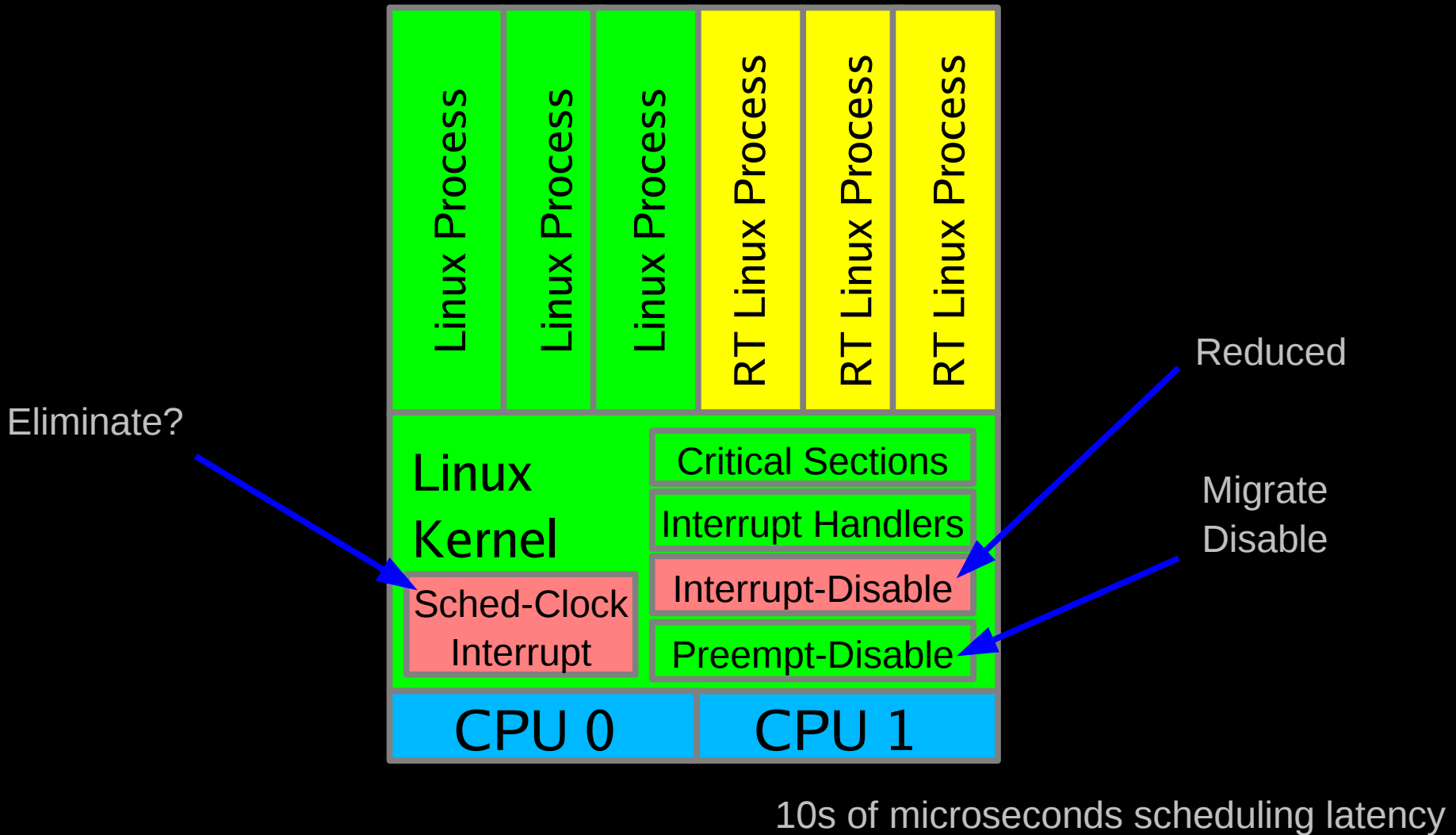


CONFIG_PREEMPT_RT=y Kernel



10s of microseconds scheduling latency

CONFIG_PREEMPT_RT_FULL=y Kernel



But 2004 PREEMPT_RT Had Problems With RCU...

- So I knew what my job had to be:

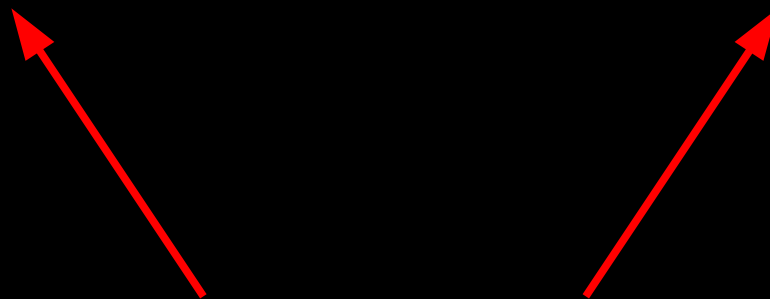
```
+      /*  
+      * PREEMPT_RT semantics: different-type read-locks  
+      * dont nest that easily:  
+      */  
+//    rcu_read_lock_read(&ptype_lock);
```

- Why is this a problem?

The Problem With 2004 PREEMPT_RT RCU

```
rcu_read_lock();  
spin_lock(&my_lock);  
do_something();  
spin_unlock(&my_lock);  
rcu_read_unlock();
```

```
spin_lock(&my_lock);  
rcu_read_lock();  
do_something_else();  
rcu_read_unlock();  
spin_unlock(&my_lock);
```

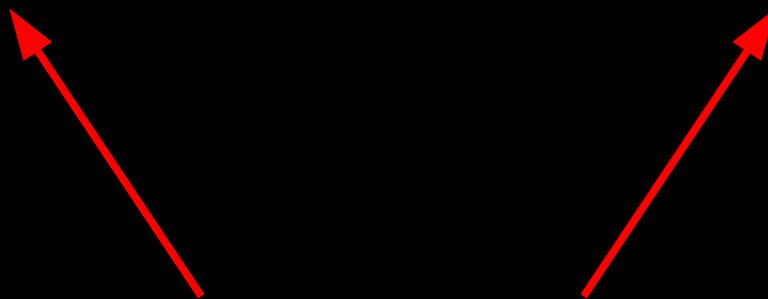


Deadlock!!!

The Problem With 2004 PREEMPT_RT RCU

```
rcu_read_lock();  
spin_lock(&my_lock);  
do_something();  
spin_unlock(&my_lock);  
rcu_read_unlock();
```

```
spin_lock(&my_lock);  
rcu_read_lock();  
do_something_else();  
rcu_read_unlock();  
spin_unlock(&my_lock);
```



Deadlock!!!

And there are a *lot* of these in the Linux kernel!

Preemptible RCU

- December 2004: realized that I fix RCU...
- March 2005: first hint that solution was possible
 - I proposed flawed approach, Esben Neilsen proposed flawed but serviceable approach
- May 2005: first design fixing flaws in Esben's approach
- June 2005: first patch submitted to LKML
- August 2005: patch accepted in -rt
- November 2006: priority boosting patch
- Early 2007: priority boosting accepted into -rt
- September 2007: preemptible RCU w/o atomics
- January 2008: preemptible RCU in mainline
- December 2009: scalable preemptible RCU in mainline
- July 2011: RCU priority boosting in mainline

The -rt Patchset Was Used in Production Early On

- 2006: aggressive real-time on 64-bit systems
 - Real-time Linux kernel (x86_64, 4-8 processors, deadlines down to 70 microseconds, measured latencies less than 40 microseconds)
 - I only did RCU. Ingo Molnar, Sven Dietrich, K. R. Foley, Thomas Gleixner, Gene Heskett, Bill Huey, Esben Nielsen, Nick Piggin, Lee Revell, Steven Rostedt, Michal Schmidt, Daniel Walker, and Karsten Wiese did the real work, as did many others joining the project later on.
 - Plus a huge number of people writing applications, supporting customers, packaging distros, ...

The -rt Patchset Was Used in Production Early On

- 2006: aggressive real-time on 64-bit systems
 - Real-time Linux kernel (x86_64, 4-8 processors, deadlines down to 70 microseconds, measured latencies less than 40 microseconds)
 - I only did RCU. Ingo Molnar, Sven Dietrich, K. R. Foley, Thomas Gleixner, Gene Heskett, Bill Huey, Esben Nielsen, Nick Piggin, Lee Revell, Steven Rostedt, Michal Schmidt, Daniel Walker, and Karsten Wiese did the real work, as did many others joining the project later on.
 - Plus a huge number of people writing applications, supporting customers, packaging distros, ...
- But some were not inclined to believe it, so...

The Writeup

“SMP and Embedded Real Time”

▪ Five Real-Time Myths:

- Embedded systems are always uniprocessor systems
- Parallel programming is mind crushingly difficult
- Real time must be either hard or soft
- Parallel real-time programming is impossibly difficult
- There is no connection between real-time and enterprise systems

“SMP and Embedded Real Time”

- Five Real-Time Myths:
 - Embedded systems are always uniprocessor systems
 - Parallel programming is mind crushingly difficult
 - Real time must be either hard or soft
 - Parallel real-time programming is impossibly difficult
 - There is no connection between real-time and enterprise systems
- This message was not well-received in all quarters

“SMP and Embedded Real Time”

- Five Real-Time Myths:
 - Embedded systems are always uniprocessor systems
 - Parallel programming is mind crushingly difficult
 - Real time must be either hard or soft
 - Parallel real-time programming is impossibly difficult
 - There is no connection between real-time and enterprise systems
- This message was not well-received in all quarters
- Let's start with “Real time must be either hard or soft”:
 - What is hard real time? A system that always meets its deadlines!

The Limits of Hard Real Time in the Hard Real World



You show me a hard real-time system,
and I will show you a hammer that will cause it to miss its deadlines.

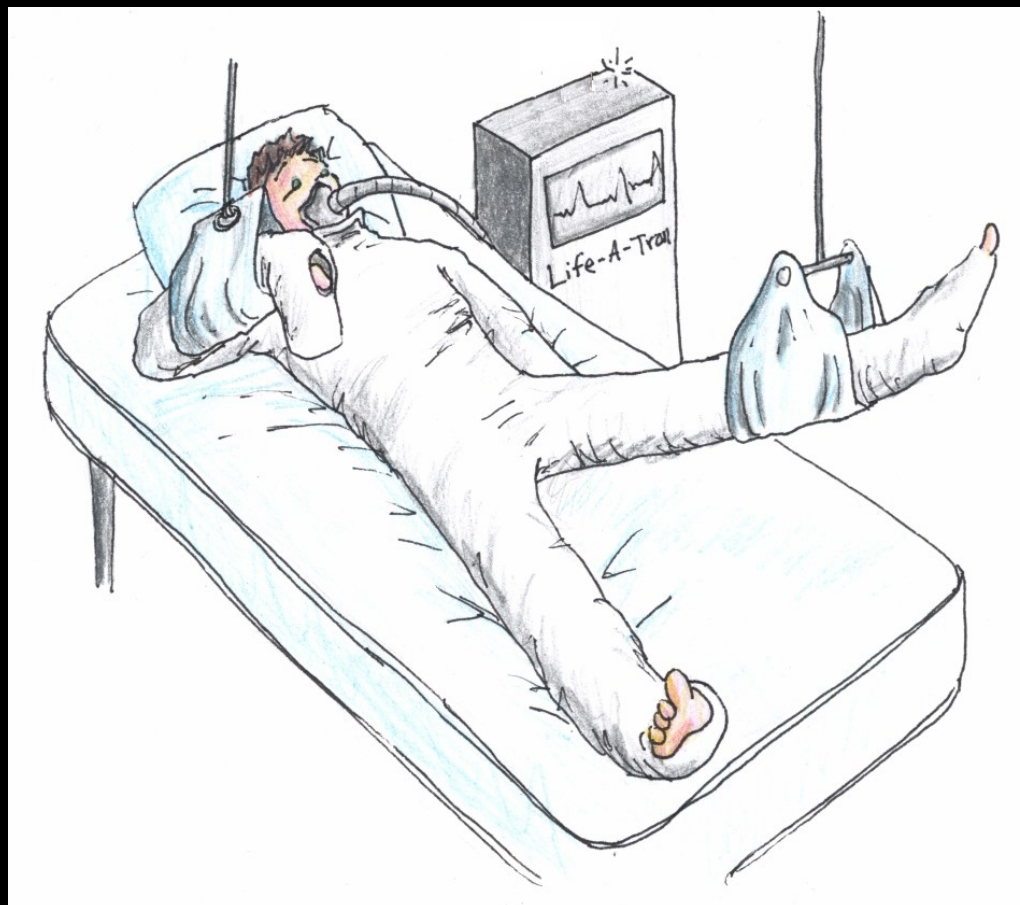
The Limits of Hard Real Time in the Hard Real World



You can make your system more robust,
but I can get a bigger hammer.

But Do Hardware Failures Count?

But Do Hardware Failures Count?



Rest assured, sir, that should there be a failure,
it will not be due to software!

The Reality of Hard and Soft Real Time?

- Hard real time is a point in a multidimensional continuum of possible real-time requirements
- Soft real time is much of the remainder of the continuum
- The reality is that we almost always need to design a much more sophisticated specification of real-time behavior:
 - What operations?
 - For each operation, what deadlines?
 - What constraints on the environment?
 - What is to happen if a given environmental constraint is violated?
 - What degradation of non-real-time performance, throughput, and scalability can be tolerated?

“SMP and Embedded Real Time”

- Five Real-Time Myths:
 - Embedded systems are always uniprocessor systems
 - **Parallel programming is mind crushingly difficult**
 - **Real time must be either hard or soft**
 - **Parallel real-time programming is impossibly difficult**
 - There is no connection between real-time and enterprise systems
- This message was not well-received in all quarters
- Just for fun, let's focus on the most controversial two of them

Parallel Programming Is Mind Crushingly Difficult???

- On the theory that there is no example quite like a good counter-example...

Parallel Programming Is Mind Crushingly Difficult???

- On the theory that there is no example quite like a good counter-example...

```
#!/bin/sh
./do_something &
./do_something_else &
wait
```

Parallel Programming Is Mind Crushingly Difficult???

- On the theory that there is no example quite like a good counter-example...

```
#!/bin/sh
./do_something &
./do_something_else &
wait
```

- As more parallel open-source projects appear, there will be more examples to learn from

Parallel Programming Is Mind Crushingingly Difficult???

- On the theory that there is no example quite like a good counter-example...

```
#!/bin/sh
./do_something &
./do_something_else &
wait
```

- As more parallel open-source projects appear, there will be more examples to learn from
 - Without the benefit of parallel-programming experience, the smarter you are, the deeper a hole you dig for yourself before you realize that you are in trouble!!!

Parallel Programming Is Mind Crushingly Difficult???

- On the theory that there is no example quite like a good counter-example...

```
#!/bin/sh
./do_something &
./do_something_else &
wait
```

- As more parallel open-source projects appear, there will be more examples to learn from
 - Without the benefit of parallel-programming experience, the smarter you are, the deeper a hole you dig for yourself before you realize that you are in trouble!!!
 - Not parallel programming's fault if you do hard things the hard way!!!

Parallel Programming Is Mind Crushingly Difficult???

- In addition, the Linux kernel is starting to see bugs that appear only in UP kernels
 - For two recent example:
 - Patch that failed to provide definitions used in UP kernels
 - Patch that livelocked on UP kernels
 - Perhaps the Linux kernel community is becoming all too comfortable with parallel programming ;-)
- Parallelism is primarily a performance optimization
 - It is one optimization of many, each with an area of applicability
 - Is parallelism the best optimization for the problem at hand?
 - Not parallel programming's fault if you make a poor choice of optimization!!!

Parallel Real Time: Impossibly Difficult?

- Again, on the theory that there is no example quite like a good counter-example...

Parallel Real Time: Impossibly Difficult?

- Again, on the theory that there is no example quite like a good counter-example...
 - Parallel real-time projects exist
 - Therefore, parallel real-time programming logically cannot be impossibly difficult

Parallel Real Time: Impossibly Difficult?

- Again, on the theory that there is no example quite like a good counter-example...
 - Parallel real-time projects exist
 - Therefore, parallel real-time programming logically cannot be impossibly difficult
 - Instead, it is merely mind-crushingly difficult

Parallel Real Time: Impossibly Difficult?

- Again, on the theory that there is no example quite like a good counter-example...
 - Parallel real-time projects exist
 - Therefore, parallel real-time programming logically cannot be impossibly difficult
 - Instead, it is merely mind-crushingly difficult
 - It will get easier as we gain experience with it, just as has been the case with each and every new technology that has been invented over the past several centuries

Parallel Real Time: Impossibly Difficult?

- Again, on the theory that there is no example quite like a good counter-example...
 - Parallel real-time projects exist
 - Therefore, parallel real-time programming logically cannot be impossibly difficult
 - Instead, it is merely mind-crushingly difficult
 - It will get easier as we gain experience with it, just as has been the case with each and every new technology that has been invented over the past several centuries
 - Consider the choices of university education for a 15th-century German merchant's son...

Parallel real-time programming: 15th Century Analogy

“There is a story of a German merchant of the fifteenth century, which I have not succeeded in authenticating, but it is so characteristic of the situation then existing that I cannot resist the temptation of telling it. It appears that the merchant had a son whom he desired to give an advanced commercial education. He appealed to a prominent professor of a university for advice as to where he should send his son. The reply was that if the mathematical curriculum of the young man was to be confined to adding and subtracting, he perhaps could obtain the instruction in a German university; but the art of multiplying and dividing, he continued, had been greatly developed in Italy which, in his opinion, was the only country where such advanced instruction could be obtained.”

Parallel real-time programming: 15th Century Analogy

“There is a story of a German merchant of the fifteenth century, which I have not succeeded in authenticating, but it is so characteristic of the situation then existing that I cannot resist the temptation of telling it. It appears that the merchant had a son whom he desired to give an advanced commercial education. He appealed to a prominent professor of a university for advice as to where he should send his son. The reply was that if the mathematical curriculum of the young man was to be confined to adding and subtracting, he perhaps could obtain the instruction in a German university; but the art of multiplying and dividing, he continued, had been greatly developed in Italy which, in his opinion, was the only country where such advanced instruction could be obtained.”

Perhaps parallel real-time programming is to the 21st century as multiplying and dividing was to the 15th century.

Boundary Between SMP and Real Time: A Good Place for Challenging New Research and Development



As with plate tectonics, the boundaries are where most of the action is!

I Believe That “SMP and Embedded Real Time” Has Stood the Test of Time

I Believe That “SMP and Embedded Real Time” Has Stood the Test of Time

However, I Did Make One Big Error in
“SMP and Embedded Real Time”

Large Error in “SMP and Embedded Real Time”

- February 8, 2012
 - Dimitri Sivanic reports 200+ microsecond latency spikes from RCU
 - My initial response, based on lots of experience otherwise:
 - “You must be joking!!!”

Large Error in “SMP and Embedded Real Time”

- February 8, 2012
 - Dimitri Sivanic reports 200+ microsecond latency spikes from RCU
 - My initial response, based on lots of experience otherwise:
 - “You must be joking!!!”
 - Further down in Dimitri's email: NR_CPUS=4096

Large Error in “SMP and Embedded Real Time”

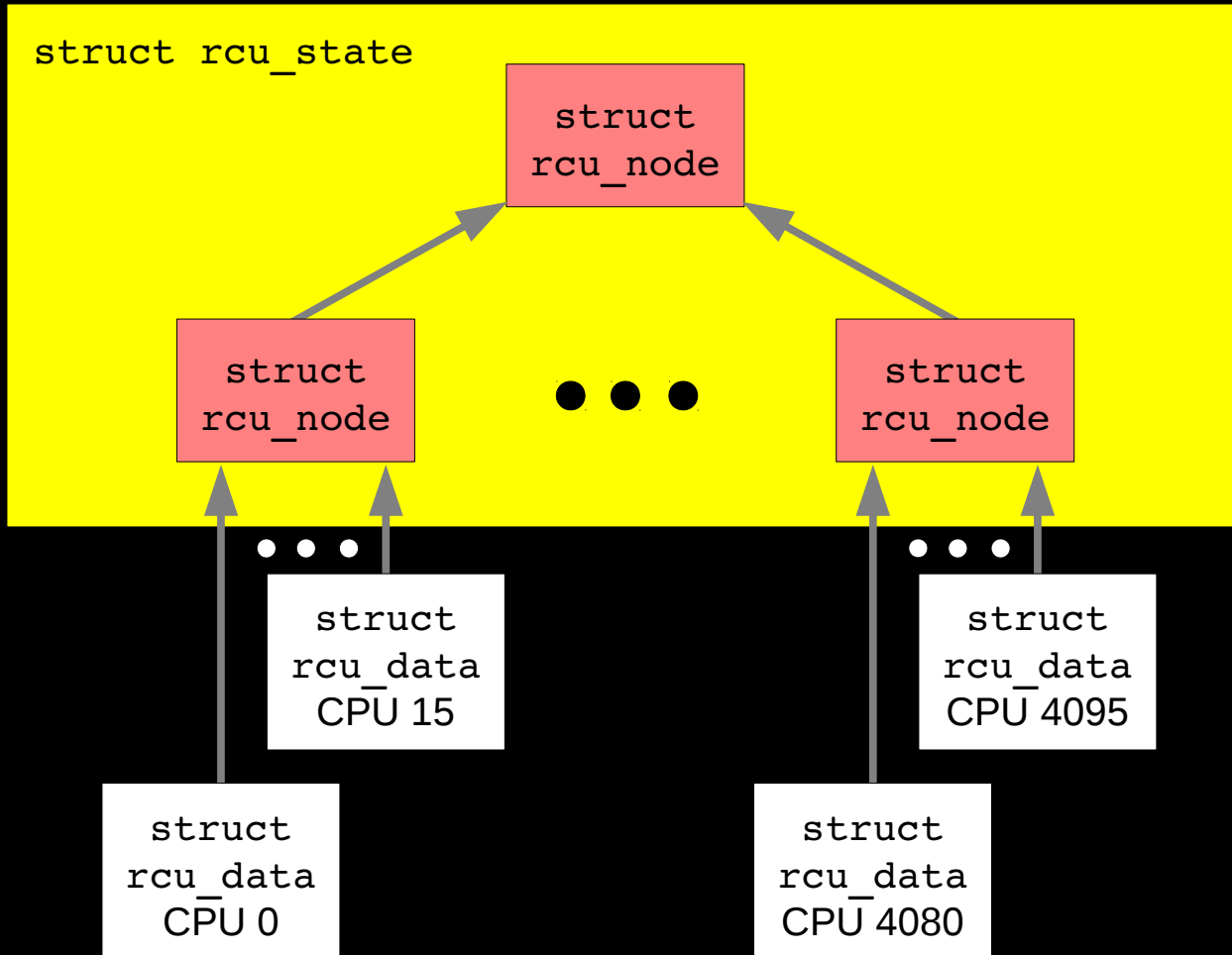
▪ February 8, 2012

- Dimitri Sivanic reports 200+ microsecond latency spikes from RCU
- My initial response, based on lots of experience otherwise:
 - “You must be joking!!!”
- Further down in Dimitri's email: NR_CPUS=4096
 - “You mean it took only 200 microseconds?”

Large Error in “SMP and Embedded Real Time”

- February 8, 2012
 - Dimitri Sivanic reports 200+ microsecond latency spikes from RCU
 - My initial response, based on lots of experience otherwise:
 - “You must be joking!!!”
 - Further down in Dimitri's email: NR_CPUS=4096
 - “You mean it took only 200 microseconds?”
- The large error: I was thinking in terms of 4-8 CPUs, maybe eventually as many as 16-32 CPUs
 - More than two orders of magnitude too small!!!

RCU Initialization



Level 0: 1 rcu_node

Level 1: 4 rcu_nodes

Level 2: 256 rcu_nodes

Total: 261 rcu_nodes

But Who Cares About Such Huge Systems?

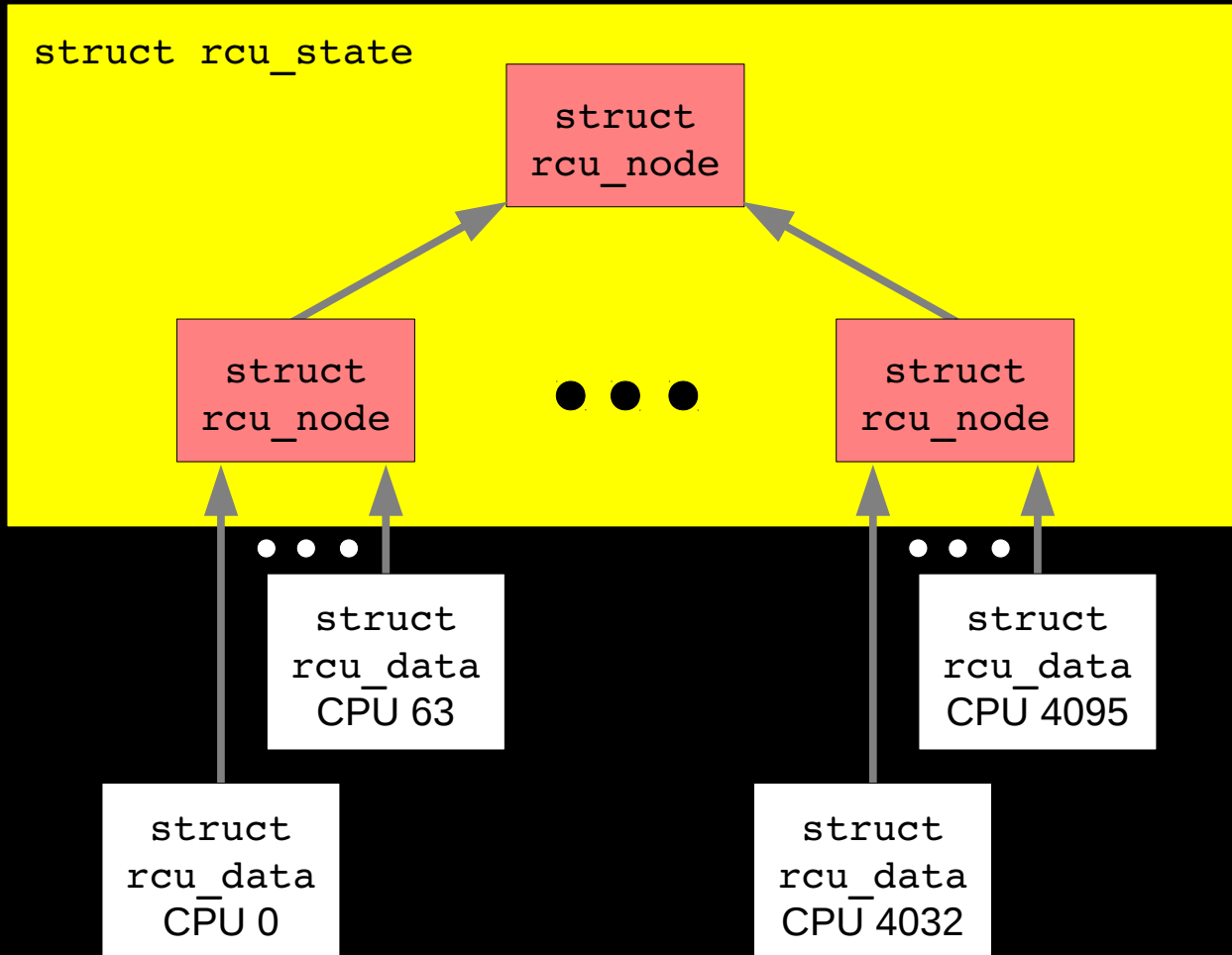
But Who Cares About Such Huge Systems?

- Their users do! :-)
- And you need to care about them as well

But Who Cares About Such Huge Systems?

- Their users do! :-)
- And you need to care about them as well
- Systems are still getting larger
 - I do remember 8-CPU systems being called “huge” only ten years ago
 - Today, *laptops* with 8 CPUs are readily available
 - And CONFIG_SMP=n is now inadequate for many *smartphones*
 - And the guys with huge systems provide valuable testing services
- Some Linux distributions build with NR_CPUS=4096
 - Something about only wanting to provide a single binary...
 - RCU must adjust, for example, increasing CONFIG_RCU_FANOUT

RCU Initialization, CONFIG_RCU_FANOUT=64



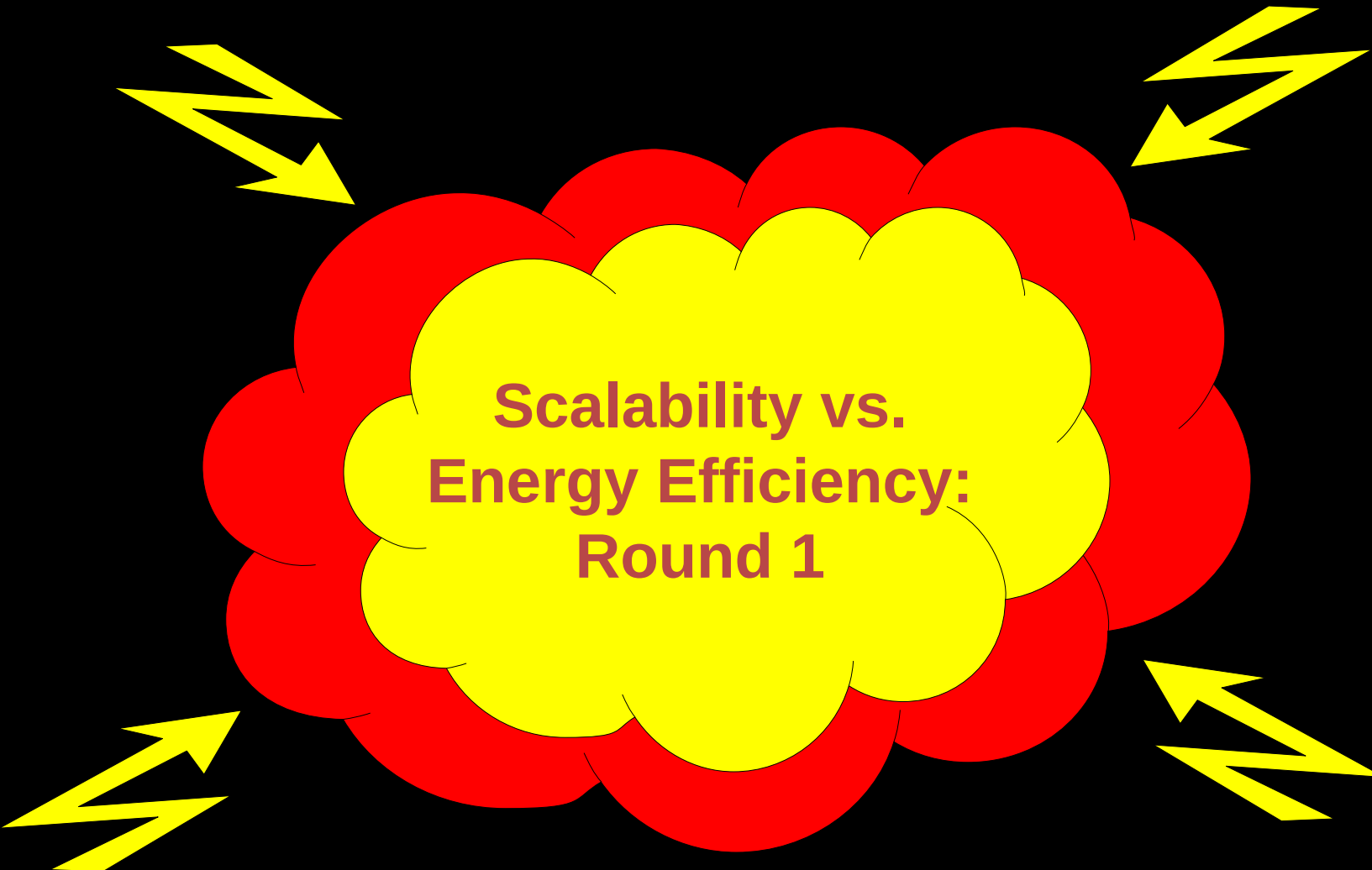
Level 0: 1 rcu_node

Level 2: 64 rcu_nodes

Total: 65 rcu_nodes

Decreases latency from 200+ to 60-70 microseconds. "Barely acceptable" to users. But...

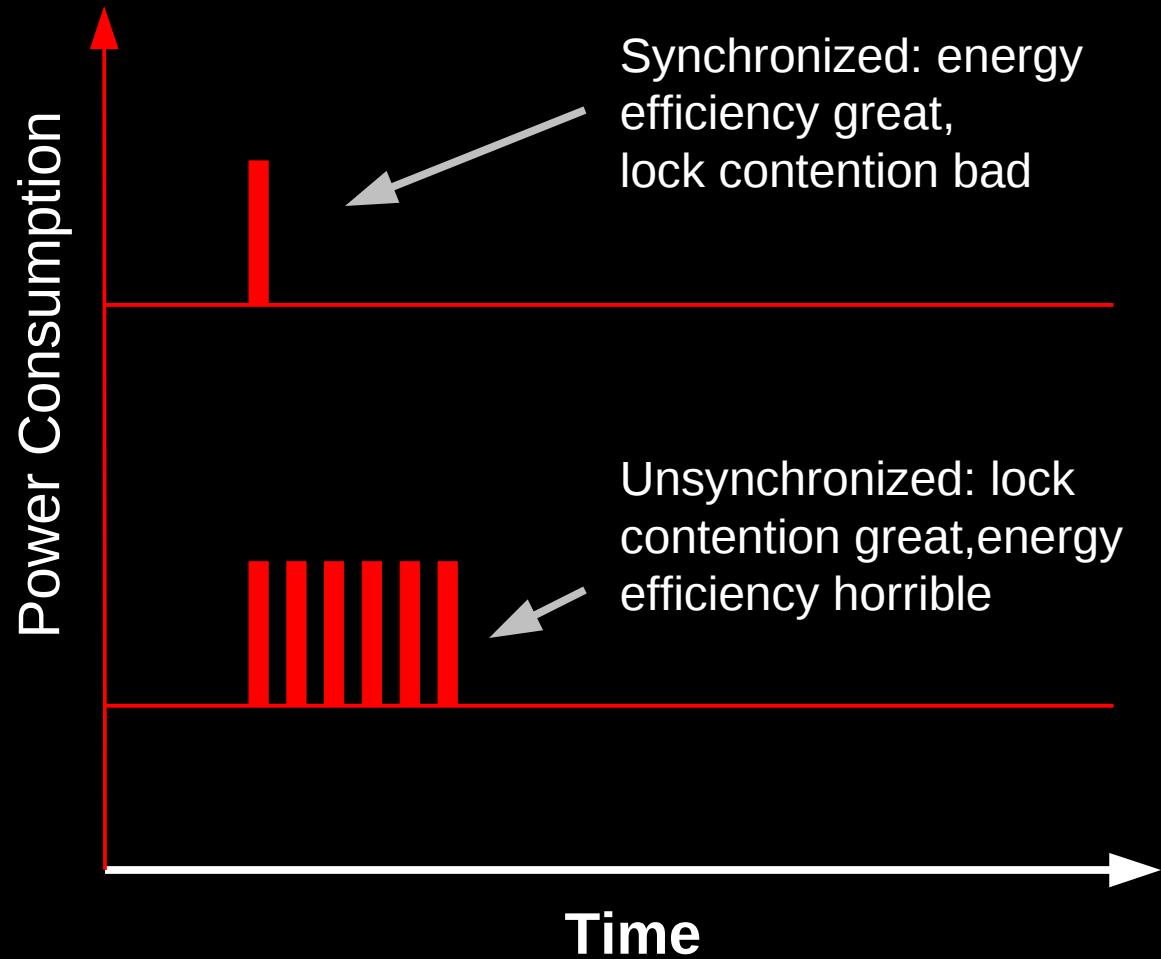
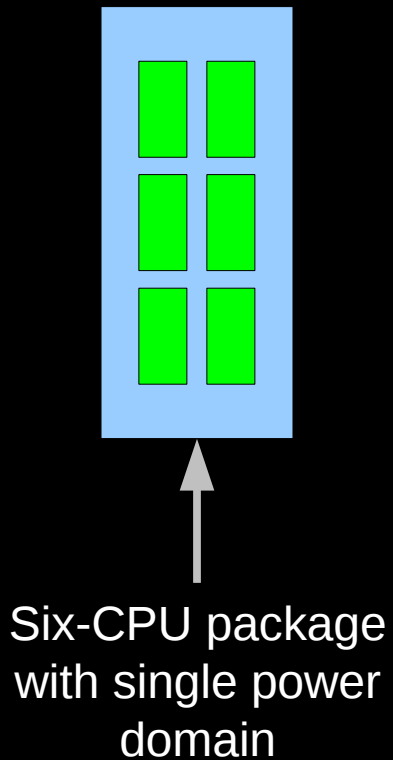
CONFIG_RCU_FANOUT=64 Consequences



CONFIG_RCU_FANOUT=64 Consequences

- Huge systems want 64 CPUs per leaf rcu_node structure
- Smaller energy-efficient systems want scheduling-clock interrupts delivered to each socket simultaneously
 - Reduces the number of per-socket power transitions under light load
- If all 64 CPUs attempt to acquire their leaf rcu_node structure's lock concurrently: Massive lock contention

Issues With Scheduler-Clock Synchronization



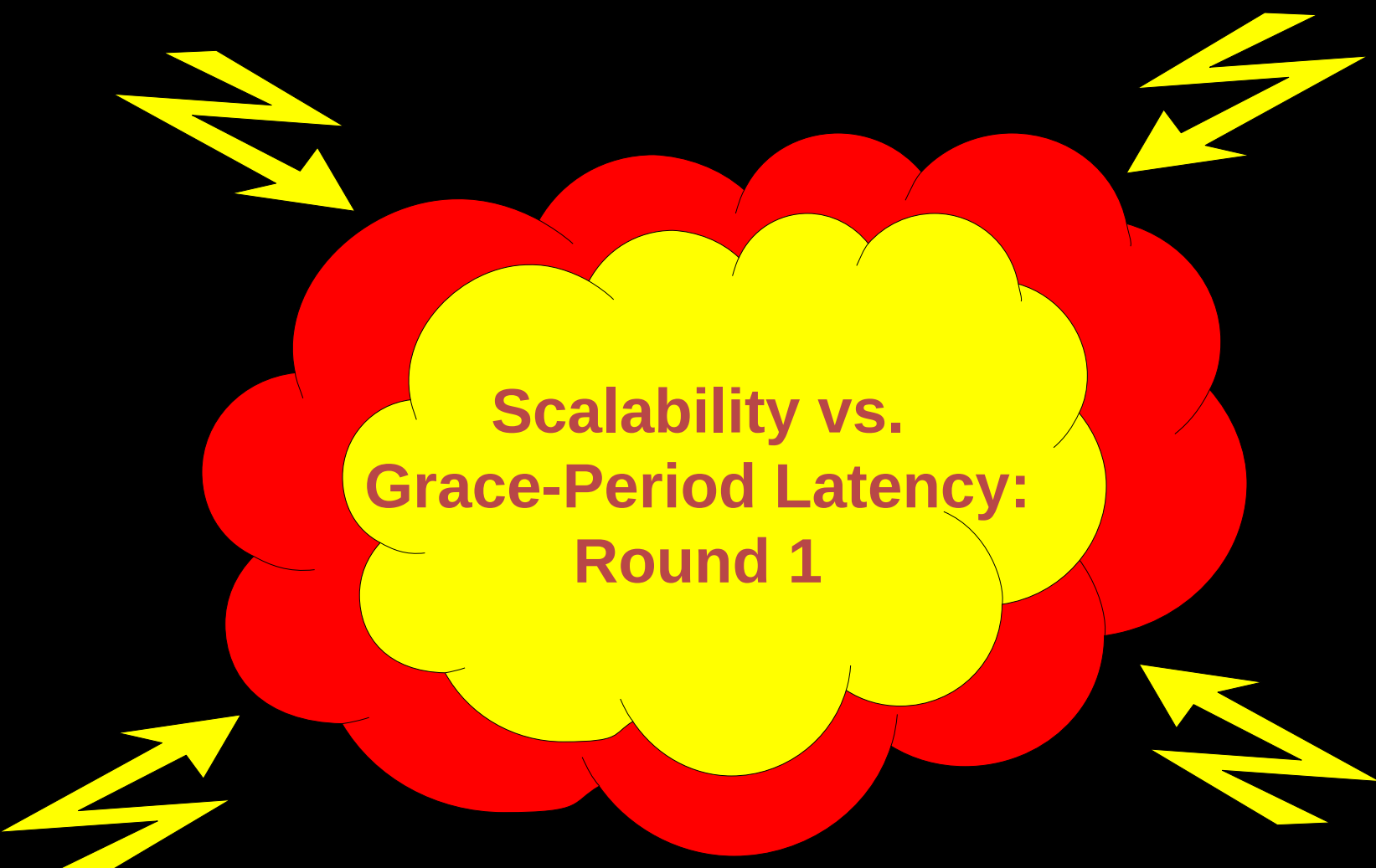
CONFIG_RCU_FANOUT=64 Consequences

- Huge systems want 64 CPUs per leaf rcu_node structure
- Smaller energy-efficient systems want scheduling-clock interrupts delivered to each socket simultaneously
 - Reduces the number of per-socket power transitions under light load
- If all 64 CPUs attempt to acquire their leaf rcu_node structure's lock concurrently: Massive lock contention
- Solution: Mike Galbraith added a boot parameter controlling scheduling-clock-interrupt skew
 - Later, Frederic Weisbecker's patch should help, but still have the possibility of all CPUs taking scheduling-clock interrupts
- Longer term: schedule events for energy and scalability

Unintended Consequences

- RCU polls CPUs to learn which are in dyntick-idle mode
 - `force_quiescent_state()` samples per-CPU counter
- Only one `force_quiescent_state()` at a time per RCU flavor
 - Mediated by trylock
- When 4096 CPUs trylock the same lock simultaneously, the results are not pretty: massive memory contention
- Immediate solution (Dimitri Sivanic):
 - Better mapping of `rcu_state` fields onto cachelines
 - Longer delay between `force_quiescent_state()` invocations, but...

Longer Polling Delay Consequences



**Scalability vs.
Grace-Period Latency:
Round 1**

Increased Polling Interval Consequences

- Increasing the polling interval increases the expected grace-period latency
- And people are already complaining about the grace periods taking too long!

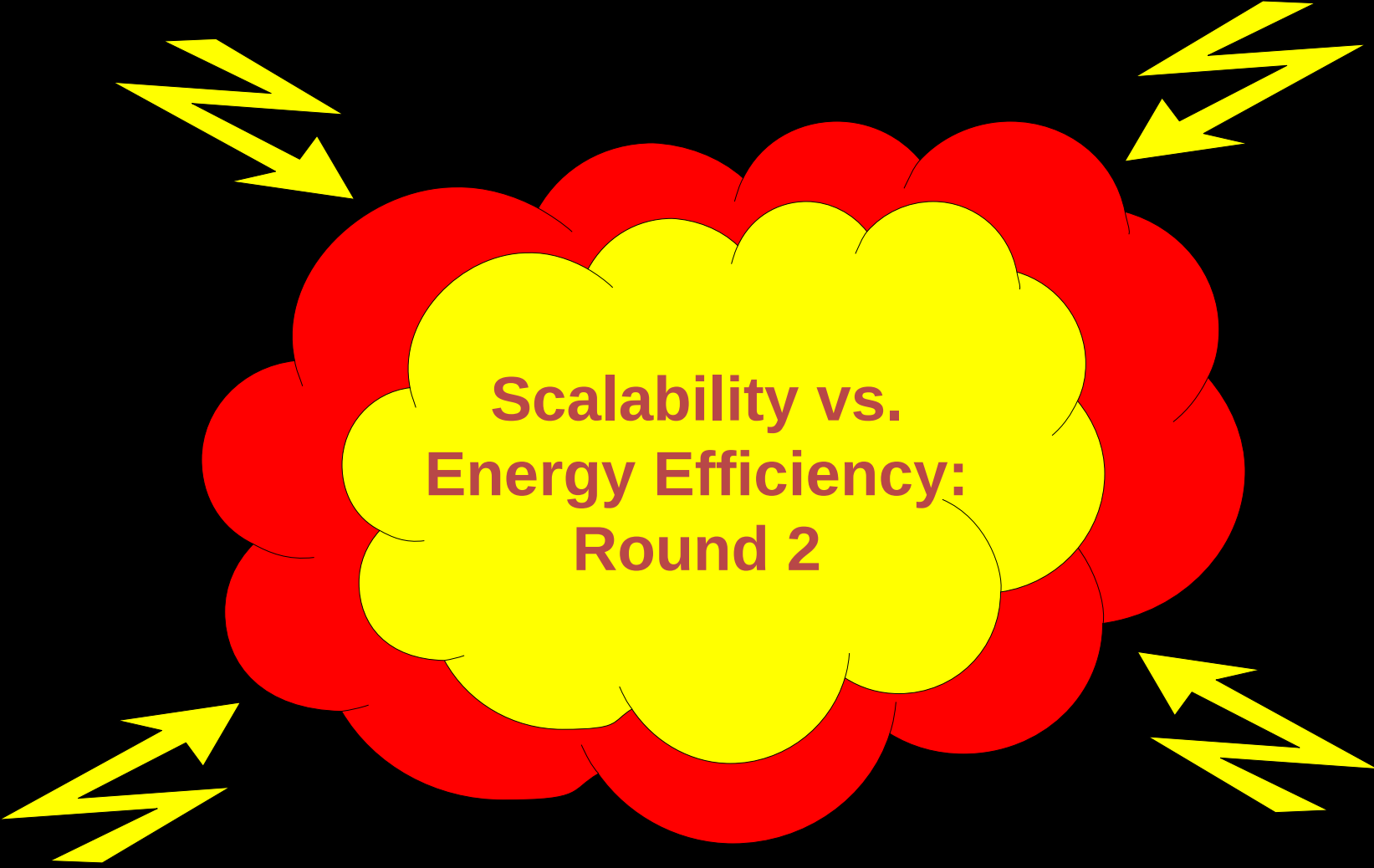
Increased Polling Interval Consequences

- Increasing the polling interval increases the expected grace-period latency
- And people are already complaining about the grace periods taking too long!
- Short-term solution: Control polling interval via boot parameter/sysfs; people can choose what works for them

Increased Polling Interval Consequences

- Increasing the polling interval increases the expected grace-period latency
- And people are already complaining about the grace periods taking too long!
- Short-term solution: Control polling interval via boot parameter/sysfs; people can choose what works for them
- Longer-term solution: Move grace period startup, polling, and cleanup to kthread, eliminating `force_quiescent_state()`'s lock
 - But this does not come for free...

RCU_FAST_NO_HZ Consequences



**Scalability vs.
Energy Efficiency:
Round 2**

RCU_FAST_NO_HZ Consequences

- When a CPU enters idle, RCU_FAST_NO_HZ can invoke `force_quiescent_state()` several times in quick succession
 - It is attempting to flush callbacks from the CPU for dyntick-idle entry
 - (See ELCE 2012 presentation for more information.)
- If a large number of CPUs enter idle at about the same time, the results are not pretty
- Can just disable RCU_FAST_NO_HZ, but sooner or later huge systems are going to want to save energy
- But that is not all...

Grace-Period kthread Issues

- Increases binding between RCU and the scheduler
 - Working on this: “bigrt” patch set delayed from 3.6 to 3.7
- Single lock mediates kthread `wait_event()/wake_up()`
 - But preemption points reduce `PREEMPT=n` latency
 - So there is at least some potential benefit from taking this path

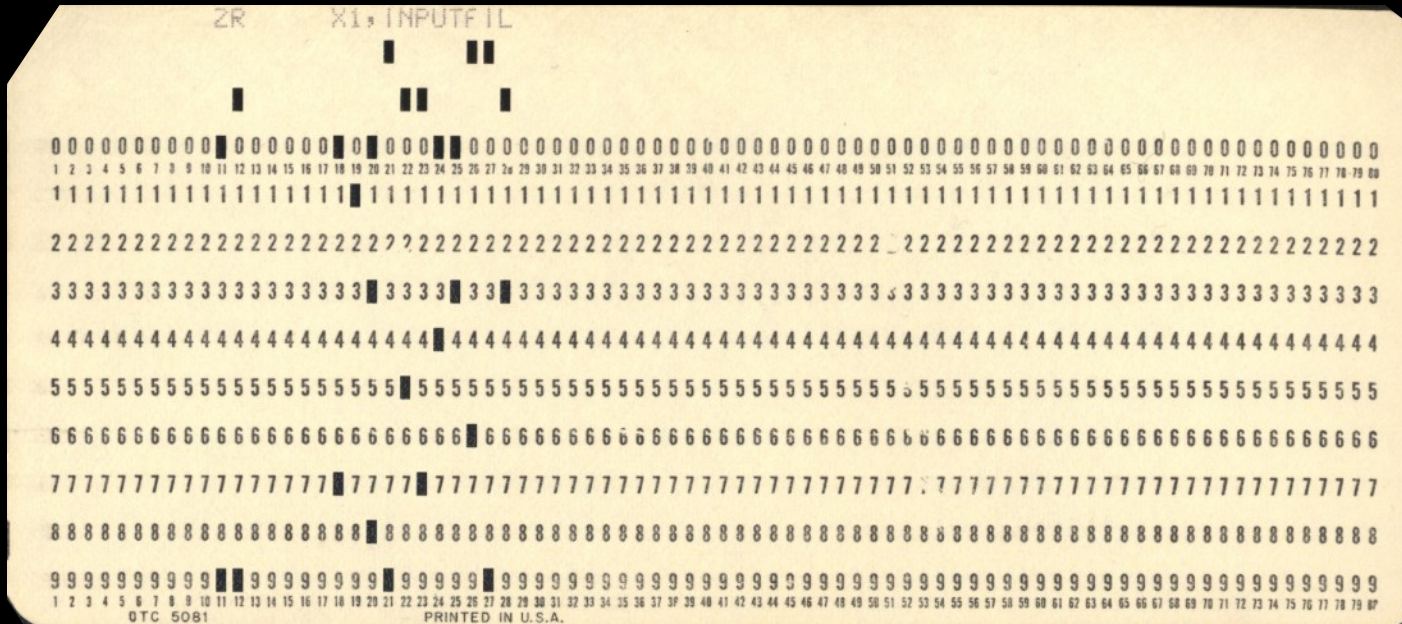
Grace-Period kthread Issues and Potential Benefits

- Increases binding between RCU and the scheduler
 - Working on this: “bigrt” patch set delayed from 3.6 to 3.7
- Single lock mediates kthread `wait_event()/wake_up()`
 - But preemption points reduce `PREEMPT=n` latency
 - So there is at least some potential benefit from taking this path
- Estimate of latency reduction:
 - Reducing `rcu_node` structures from 261 to 65 resulted in latency reduction from roughly 200 to 70 microseconds
 - Reducing `rcu_node` structures to *one* per preemption opportunity might reduce latency to about 30 microseconds (linear extrapolation)
 - But why not just run the test?

Grace-Period kthread Issues and Potential Benefits

- Increases binding between RCU and the scheduler
 - Working on this: “bigrt” patch set delayed from 3.6 to 3.7
- Single lock mediates kthread `wait_event()/wake_up()`
 - But preemption points reduce `PREEMPT=n` latency
 - So there is at least some potential benefit from taking this path
- Estimate of latency reduction:
 - Reducing `rcu_node` structures from 261 to 65 resulted in latency reduction from roughly 200 to 70 microseconds
 - Reducing `rcu_node` structures to *one* per preemption opportunity might reduce latency to about 30 microseconds (linear extrapolation)
 - But why not just run the test?
 - Because time on a 4096-CPU system is hard to come by
 - Fortunately, I have a very long history of relevant experience...

Coping With 4096-CPU System Scarcity



About That Single Global Lock...

About That Single Global Lock...

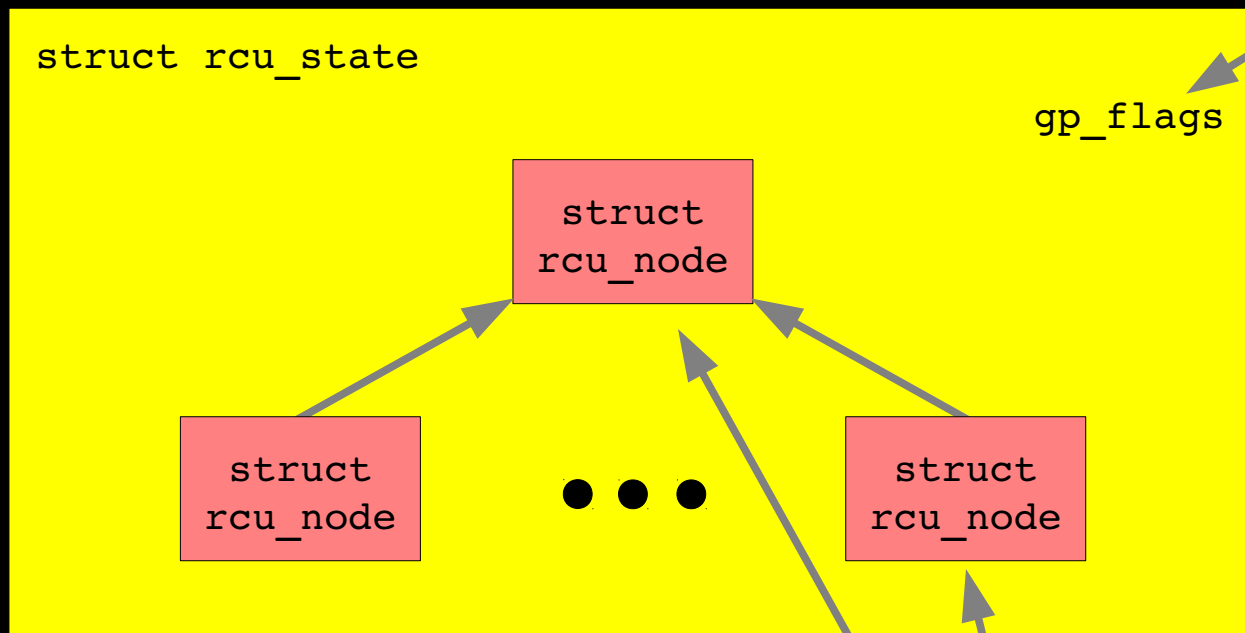
- Grace-period operations are global events
 - So if already running or being awakened, no action required
- This situation can be handled by a variation on a tournament lock (Graunke & Thakkar 1990)

About That Single Global Lock...

- Grace-period operations are global events
 - So if already running or being awakened, no action required
- This situation can be handled by a variation on a tournament lock (Graunke & Thakkar 1990)
 - A variation that does not share the poor performance noted by Graunke and Thakkar

Conditional Tournament Lock

Checked at each level



`spin_trylock()` at each level,
release at next level

Conditional Tournament Lock Code

```
1 rnp = per_cpu_ptr(rsp->rda, raw_smp_processor_id())->mynode;
2 for (; rnp != NULL; rnp = rnp->parent) {
3     ret = (ACCESS_ONCE(rsp->gp_flags) & RCU_GP_FLAG_FQS) ||
4         !raw_spin_trylock(&rnp->fqslock);
5     if (rnp_old != NULL)
6         raw_spin_unlock(&rnp_old->fqslock);
7     if (ret) {
8         rsp->n_force_qs_lh++;
9         return;
10    }
11    rnp_old = rnp;
12 }
```

Conditional Tournament Lock Code

```
1 rnp = per_cpu_ptr(rsp->rda, raw_smp_processor_id())->mynode;
2 for (; rnp != NULL; rnp = rnp->parent) {
3     ret = (ACCESS_ONCE(rsp->gp_flags) & RCU_GP_FLAG_FQS) ||
4         !raw_spin_trylock(&rnp->fqslock);
5     if (rnp_old != NULL)
6         raw_spin_unlock(&rnp_old->fqslock);
7     if (ret) {
8         rsp->n_force_qs_lh++;
9         return;
10    }
11    rnp_old = rnp;
12 }
```

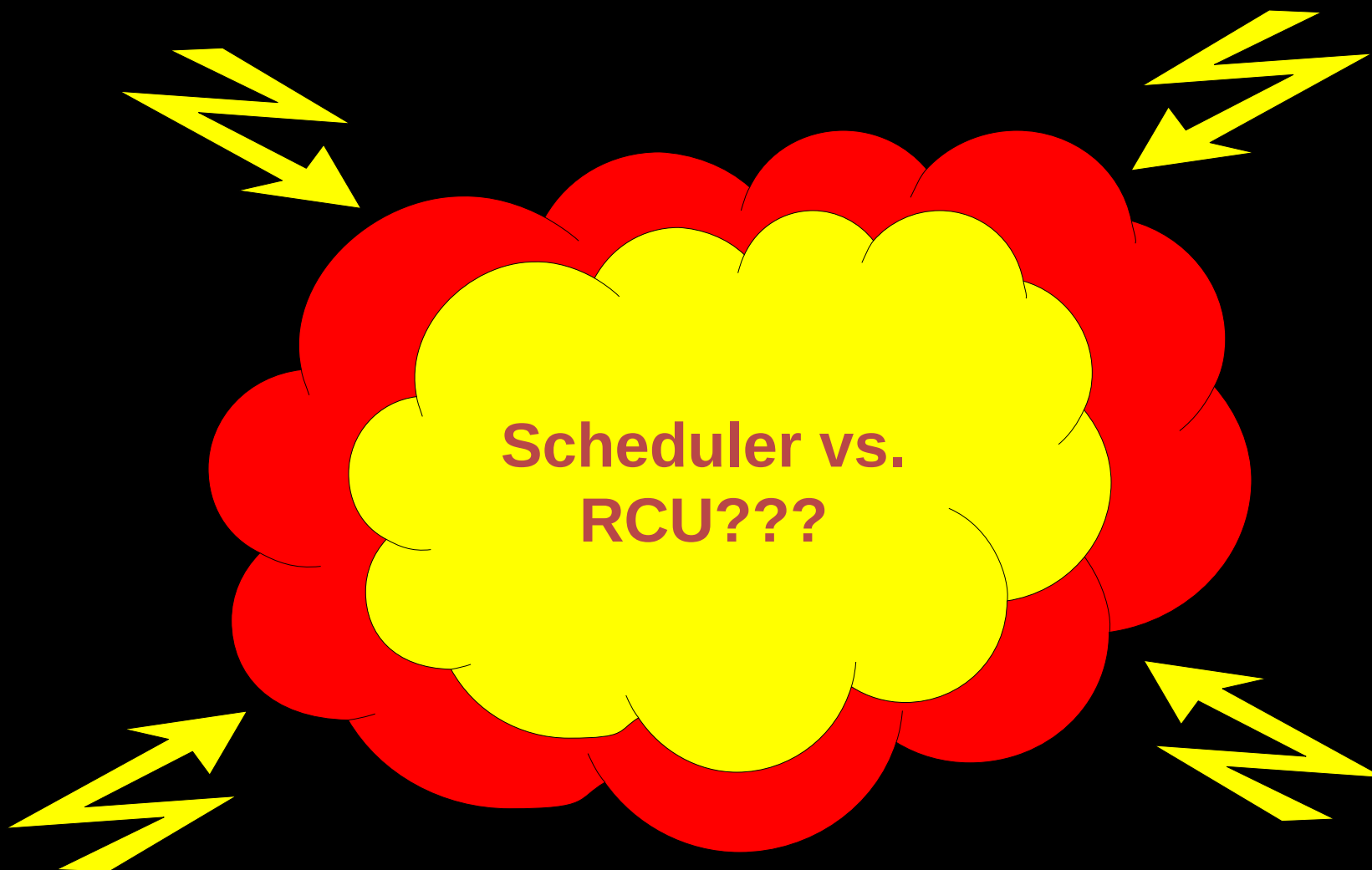
Effectiveness TBD

Other Possible Issues

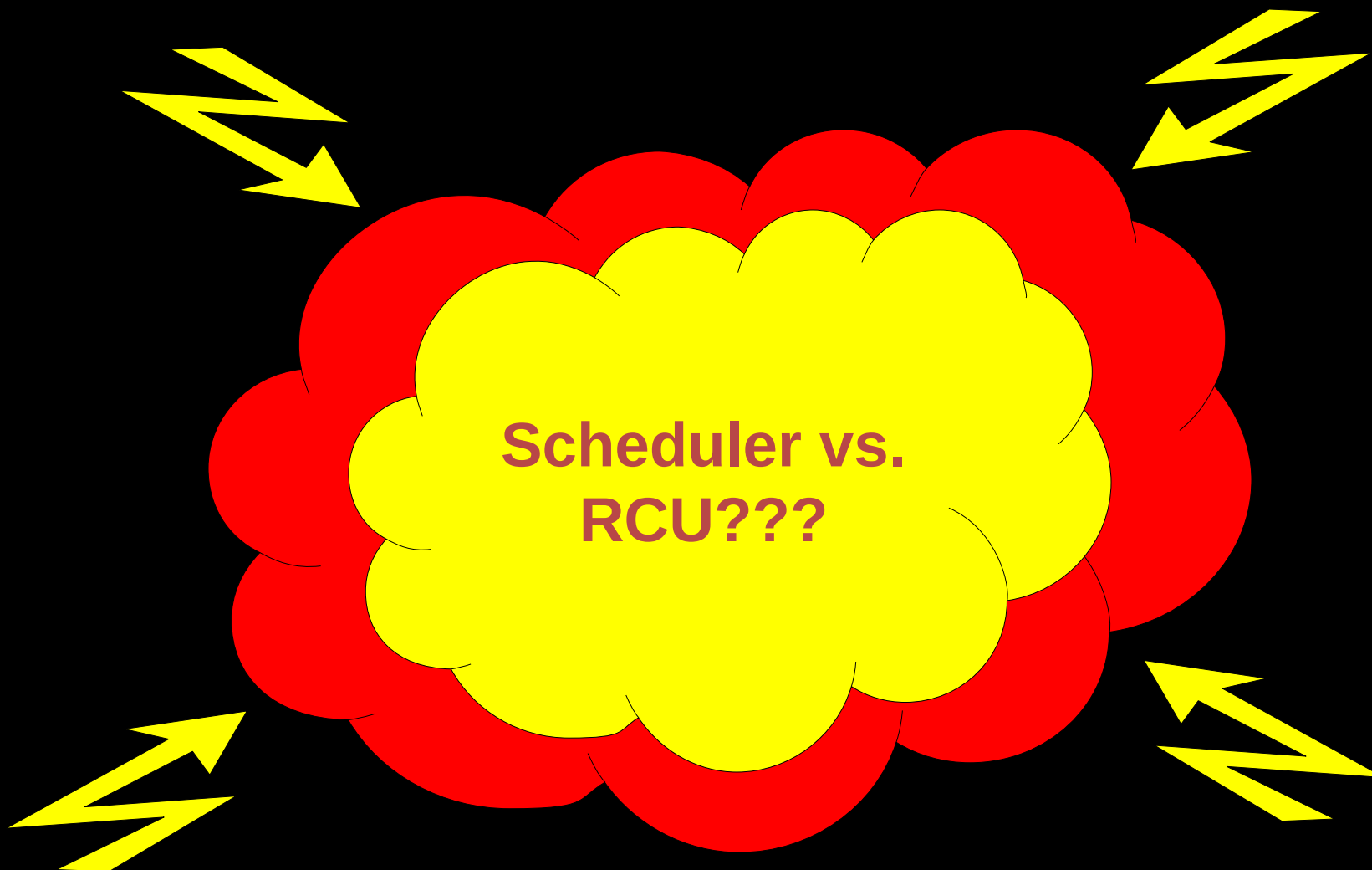
Other Possible Issues

- The `synchronize_*_expedited()` primitives loop over all CPUs
 - Parallelize? Optimize for dyntick-idle state?
- The `rcu_barrier()` primitives loop over all CPUs
 - Parallelize? Avoid running on other CPUs?
- Should `force_quiescent_state()` make use of state in non-leaf `rcu_node` structures to limit scan?
 - This actually degrades worst-case behavior
- Grace-period initialization and cleanup loops over all `rcu_node` structures
 - Parallelize?
- `NR_CPUS=4096` on small systems (RCU handles at boot)
- And, perhaps most important...

Possible Issue With RCU in a kthread

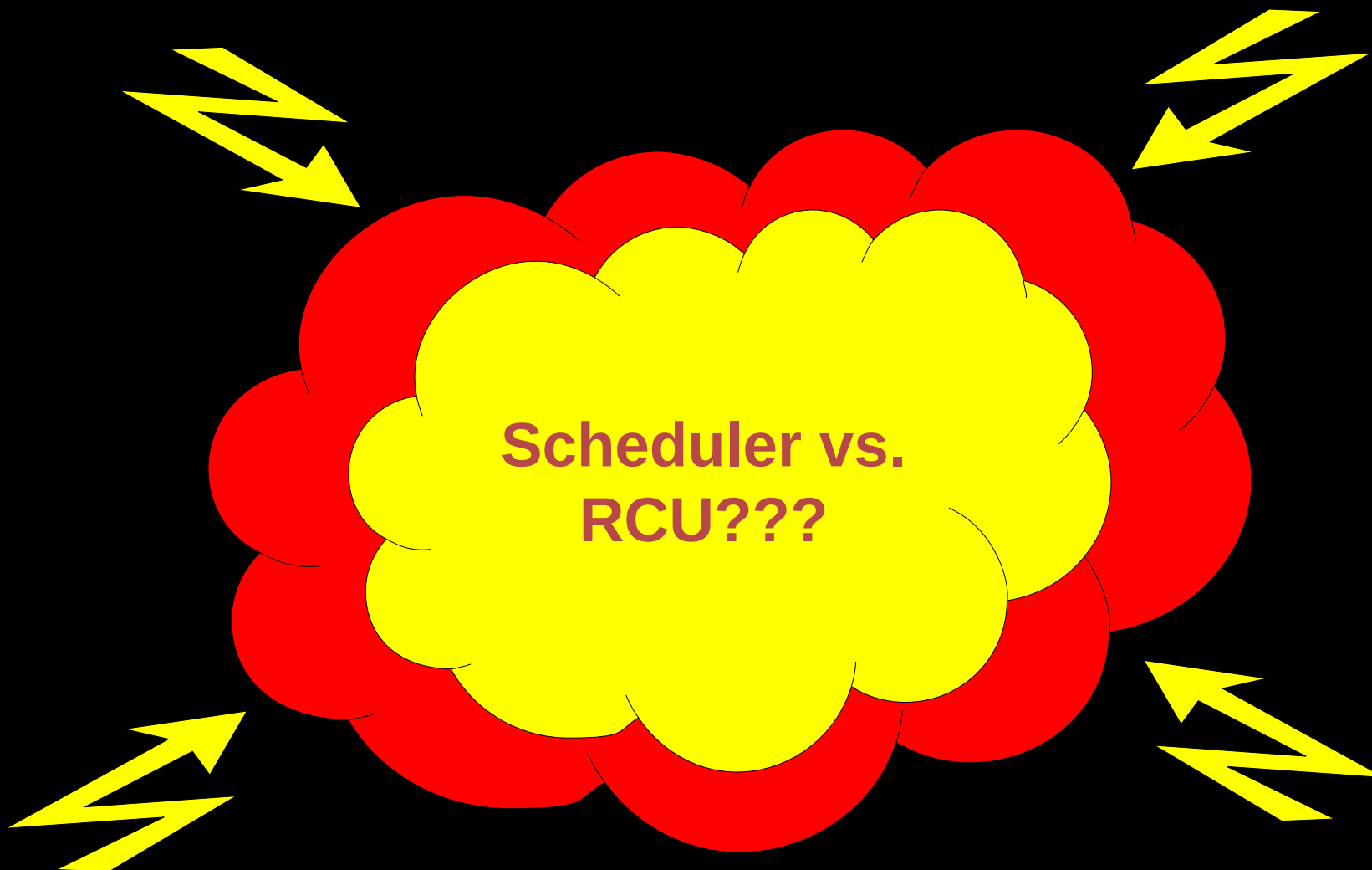


Possible Issue With RCU in a kthread



When these two fight, they both lose!

Possible Issue With RCU in a kthread



When these two fight, they both lose!
Much better if they both win!!!

The Linux Scheduler and RCU

- RCU uses the scheduler and the scheduler uses RCU
 - Plenty of opportunity for both RCU and the scheduler to lose big time!
 - See for example: <http://lwn.net/Articles/453002/>
 - Or this more-recent deadlock: <https://lkml.org/lkml/2012/7/2/163>

The Linux Scheduler and RCU

- RCU uses the scheduler and the scheduler uses RCU
 - Plenty of opportunity for both RCU and the scheduler to lose big time!
 - See for example: <http://lwn.net/Articles/453002/>
 - Or this more-recent deadlock: <https://lkml.org/lkml/2012/7/2/163>
- But driving RCU's grace periods from a kthread should be OK

The Linux Scheduler and RCU

- RCU uses the scheduler and the scheduler uses RCU
 - Plenty of opportunity for both RCU and the scheduler to lose big time!
 - See for example: <http://lwn.net/Articles/453002/>
 - Or this more-recent deadlock: <https://lkml.org/lkml/2012/7/2/163>
- But driving RCU's grace periods from a kthread should be OK
 - As long as the scheduler doesn't wait for a grace period on any of its wake-up or context-switch fast paths

The Linux Scheduler and RCU

- RCU uses the scheduler and the scheduler uses RCU
 - Plenty of opportunity for both RCU and the scheduler to lose big time!
 - See for example: <http://lwn.net/Articles/453002/>
 - Or this more-recent deadlock: <https://lkml.org/lkml/2012/7/2/163>
- But driving RCU's grace periods from a kthread should be OK
 - As long as the scheduler doesn't wait for a grace period on any of its wake-up or context-switch fast paths
 - Either directly or indirectly

The Linux Scheduler and RCU

- RCU uses the scheduler and the scheduler uses RCU
 - Plenty of opportunity for both RCU and the scheduler to lose big time!
 - See for example: <http://lwn.net/Articles/453002/>
 - Or this more-recent deadlock: <https://lkml.org/lkml/2012/7/2/163>
- But driving RCU's grace periods from a kthread should be OK
 - As long as the scheduler doesn't wait for a grace period on any of its wake-up or context-switch fast paths
 - Either directly or indirectly
 - And as long as the scheduler doesn't exit an RCU read-side critical section while holding a runqueue or pi lock if that RCU read-side critical section had any chance of being preempted

Conclusions

Conclusions

- They say that the best way to predict the future is to invent it

Conclusions

- They say that the best way to predict the future is to invent it
 - I am here to tell you that even this method is not foolproof

Conclusions

- They say that the best way to predict the future is to invent it
 - I am here to tell you that even this method is not foolproof
- SMP, real time, and energy efficiency are each well known
 - The real opportunities for new work involve combinations of them
- Some need for 10s-of-microseconds latency on 4096 CPUs
 - Translates to mainstream need on tens or hundreds of CPUs
 - Supporting this is not impossible

Conclusions

- They say that the best way to predict the future is to invent it
 - I am here to tell you that even this method is not foolproof
- SMP, real time, and energy efficiency are each well known
 - The real opportunities for new work involve combinations of them
- Some need for 10s-of-microseconds latency on 4096 CPUs
 - Translates to mainstream need on tens or hundreds of CPUs
 - Supporting this is not impossible
 - It will only require a little mind crushing ;-)

Conclusions

- They say that the best way to predict the future is to invent it
 - I am here to tell you that even this method is not foolproof
- SMP, real time, and energy efficiency are each well known
 - The real opportunities for new work involve combinations of them
- Some need for 10s-of-microseconds latency on 4096 CPUs
 - Translates to mainstream need on tens or hundreds of CPUs
 - Supporting this is not impossible
 - It will only require a little mind crushing ;-)
- There is still much work to be done on the Linux kernel
 - But even more work required for open-source applications
- The major large-system challenges are at the design level

Conclusions

- They say that the best way to predict the future is to invent it
 - I am here to tell you that even this method is not foolproof
- SMP, real time, and energy efficiency are each well known
 - The real opportunities for new work involve combinations of them
- Some need for 10s-of-microseconds latency on 4096 CPUs
 - Translates to mainstream need on tens or hundreds of CPUs
 - Supporting this is not impossible
 - It will only require a little mind crushing ;-)
- There is still much work to be done on the Linux kernel
 - But even more work required for open-source applications
- The major large-system challenges are at the design level
 - Pity that design issues receive little emphasis in the CS curriculum!!!

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?