RCU and Breakage

Paul E. McKenney
IBM Distinguished Engineer & CTO Linux
Linux Technology Center

September 20, 2009

Overview

- What the #$I#@(&!!! is RCU-bh for???
- RCU status in mainline
- Breakage for performance and scalability

What the #$I#@(&!!! is RCU-bh For???

What the #$I#@(&!!! is RCU-bh For???

- It is all Robert Olsson's fault!!!
  - Ran a DDoS workload that hung the system
  - ICMP redirects forced routing-table updates
    - Routing cache protected by RCU
    - Each update waits for a grace period before freeing
  - Load was so heavy that system never left irq!!!
    - No context switches, no quiescent states, no grace periods
    - Eventually, OOM!!!
  - Dipankar created RCU-bh
    - Additional quiescent state in softirq execution
    - Routing cache converted to RCU-bh, then withstood DDoS

RCU Status in Mainline

RCU Status in Mainline

- synchronize_sched_expedited() in mainline
  - Completes grace period in few tens of microseconds
  - By hammering all the CPUs with IPIs
  - Therefore, should be used sparingly
    - Boot-time and other infrequent updates
- CLASSIC_RCU and PREEMPT_RCU are gone
- TREE_RCU and TREE_PREEMPT_RCU instead
- TINY_RCU under test, not yet in mainline
  - Reports to the contrary notwithstanding

Breakage for Performance and Scalability

Performance of Synchronization Mechanisms

4-CPU 1.8GHz AMD Opteron 844 system

Operation	Cost (ns)	Ratio
Clock period	0.6	1
Best-case CAS	37.9	63.2
Best-case lock	65.6	109.3
Single cache miss	139.5	232.5
CAS cache miss	306.0	510.0

Need to be here! (Partitioning/RCU)

Heavily optimized reader-writer lock might get here for readers (but too bad about those poor writers...)

Typical synchronization mechanisms do this a lot

Performance of Synchronization Mechanisms

4-CPU 1.8GHz AMD Opteron 844 system

Operation	Cost (ns)	Ratio
Clock period	0.6	1
Best-case CAS	37.9	63.2
Best-case lock	65.6	109.3
Single cache miss	139.5	232.5
CAS cache miss	306.0	510.0

Need to be here! (Partitioning/RCU)

Heavily optimized reader-writer lock might get here for readers (but too bad about those poor writers...)

Typical synchronization mechanisms do this a lot

But this is an old system...

Performance of Synchronization Mechanisms

4-CPU 1.8GHz AMD Opteron 844 system

Operation	Cost (ns)	Ratio
Clock period	0.6	1
Best-case CAS	37.9	63.2
Best-case lock	65.6	109.3
Single cache miss	139.5	232.5
CAS cache miss	306.0	510.0

Need to be here! (Partitioning/RCU)

Heavily optimized reader-writer lock might get here for readers (but too bad about those poor writers...)

Typical synchronization mechanisms do this a lot

But this is an old system...

And why low-level details???

Why All These Low-Level Details???

- Would you trust a bridge designed by someone who did not understand strengths of materials?
  - Or a ship designed by someone who did not understand the steel-alloy transition temperatures?
  - Or a house designed by someone who did not understand that unfinished wood rots when wet?
  - Or a car designed by someone who did not understand the corrosion properties of the metals used in the exhaust system?
  - Or a space shuttle designed by someone who did not understand the temperature limitations of O-rings?
- So why trust algorithms from someone ignorant of the properties of the underlying hardware???

Performance of Synchronization Mechanisms

16-CPU 2.8GHz Intel X5550 (Nehalem) System

Operation	Cost (ns)	Ratio
Clock period	0.4	1
“Best-case” CAS	12.2	33.8
Best-case lock	25.6	71.2
Single cache miss	12.9	35.8
CAS cache miss	7.0	19.4

What a difference a few years can make!!!

Performance of Synchronization Mechanisms

16-CPU 2.8GHz Intel X5550 (Nehalem) System

Operation	Cost (ns)	Ratio
Clock period	0.4	1
“Best-case” CAS	12.2	33.8
Best-case lock	25.6	71.2
Single cache “miss”	12.9	35.8
CAS cache “miss”	7.0	19.4
Single cache miss (off-core)	31.2	86.6
CAS cache miss (off-core)	31.2	86.5

Not *quite* so good... But still a 6x improvement!!!

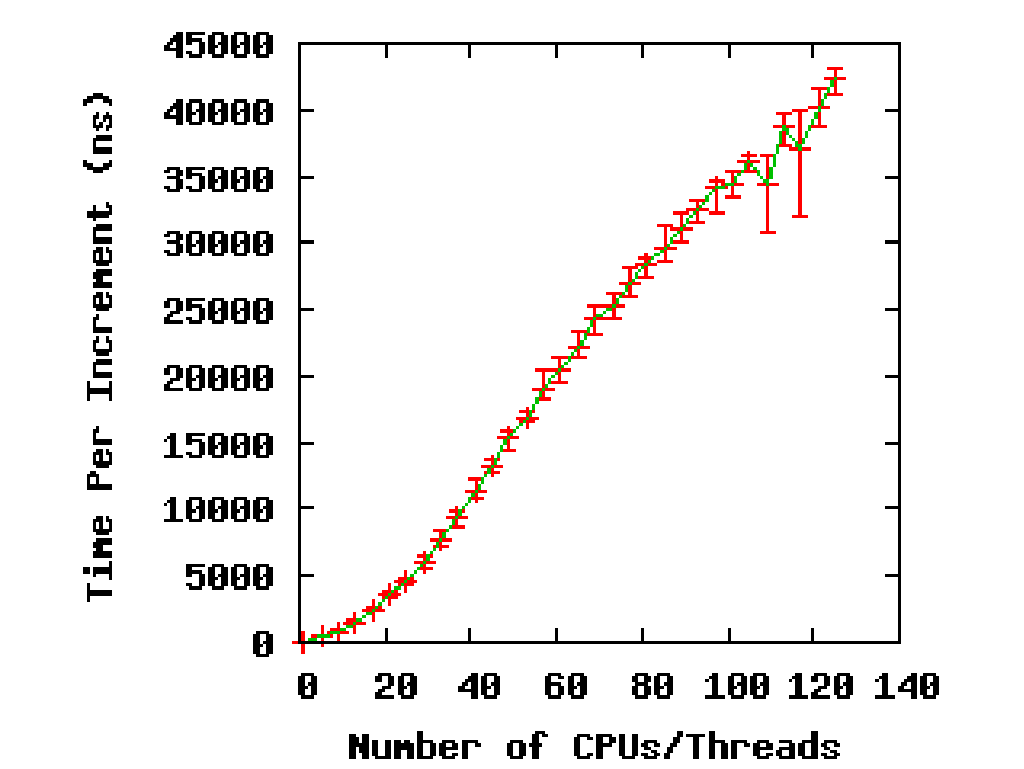
Performance of Synchronization Mechanisms

16-CPU 2.8GHz Intel X5550 (Nehalem) System

Operation	Cost (ns)	Ratio
Clock period	0.4	1
“Best-case” CAS	12.2	33.8
Best-case lock	25.6	71.2
Single cache miss	12.9	35.8
CAS cache miss	7.0	19.4
Single cache miss (off-core)	31.2	86.6
CAS cache miss (off-core)	31.2	86.5
Single cache miss (off-socket)	92.4	256.7
CAS cache miss (off-socket)	95.9	266.4

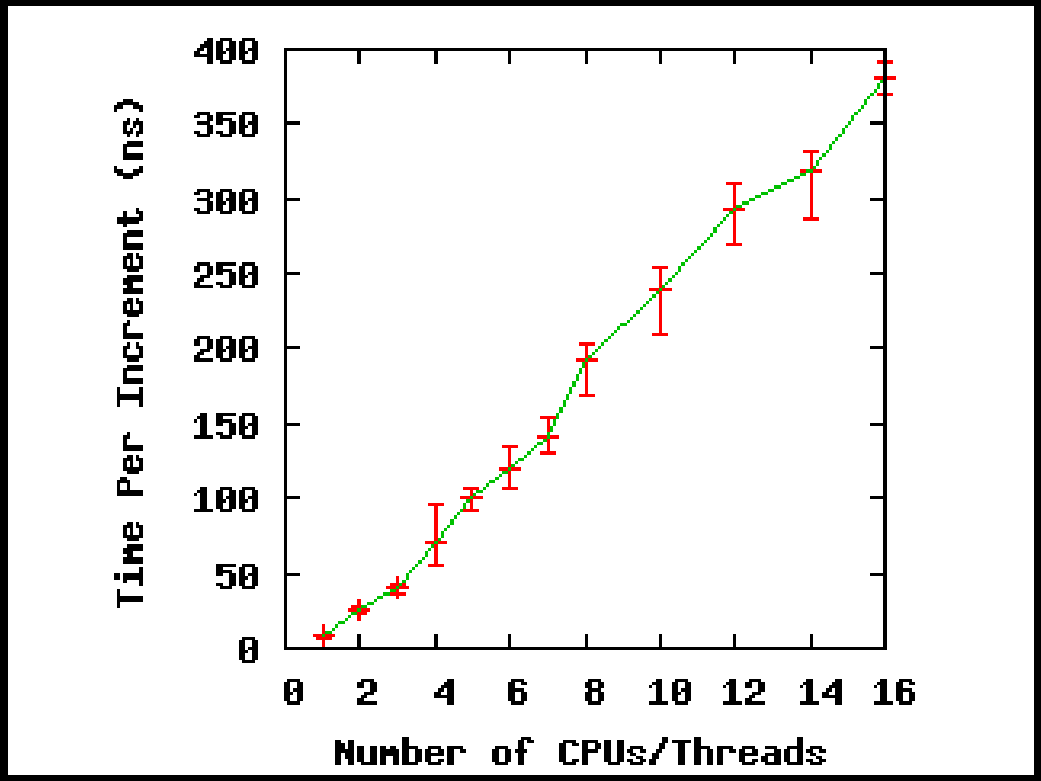
Maybe not such a big difference after all...
And these are best-case values!!! (Why?)

Performance of Synchronization Mechanisms



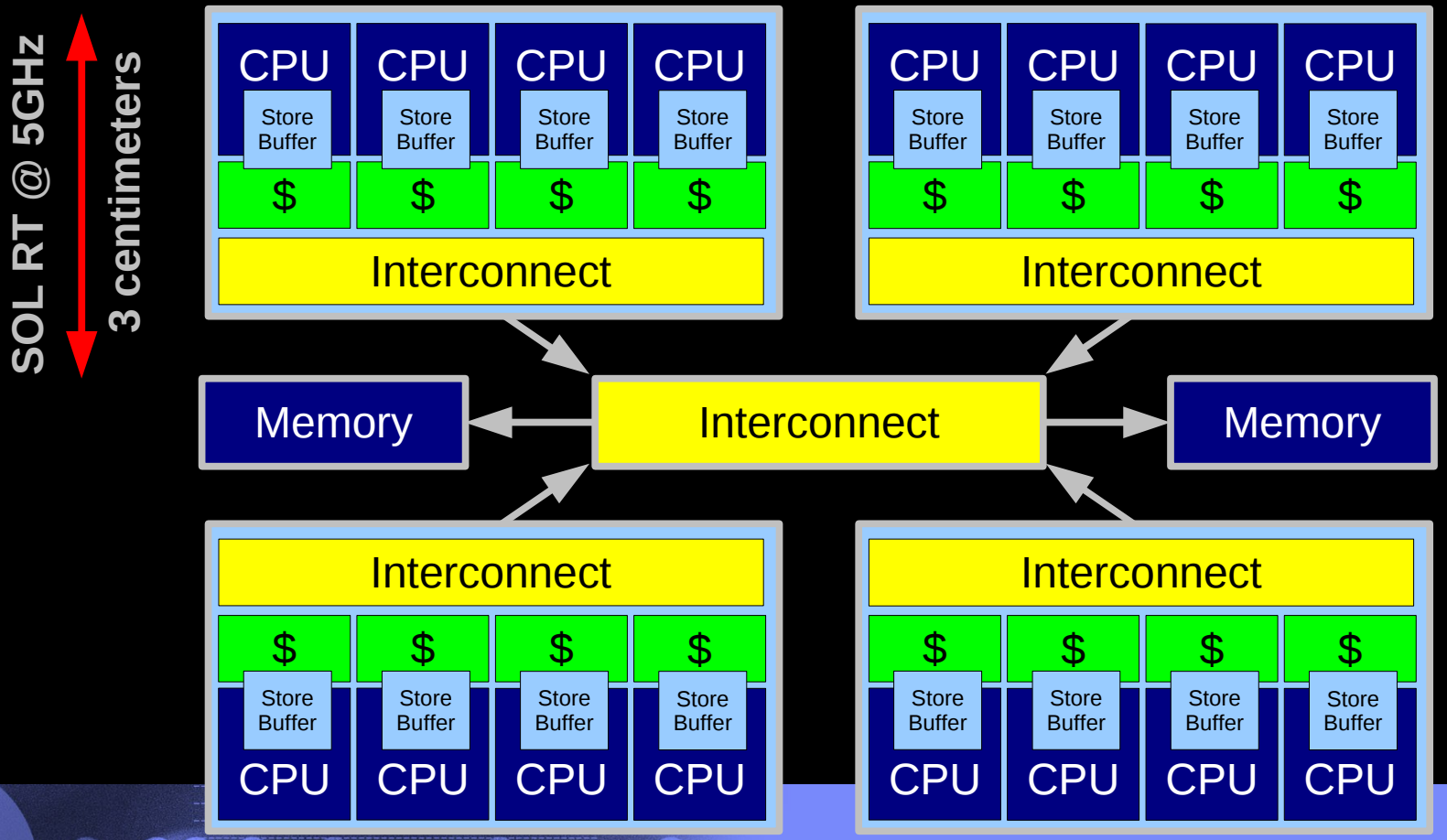
If you thought a *single* atomic operation was slow, try lots of them!!!
(Parallel atomic increment of single variable on 1.9GHz Power 5 system)

Performance of Synchronization Mechanisms



Same effect on a 16-CPU 2.8GHz Intel X5550 (Nehalem) system

System Hardware Structure

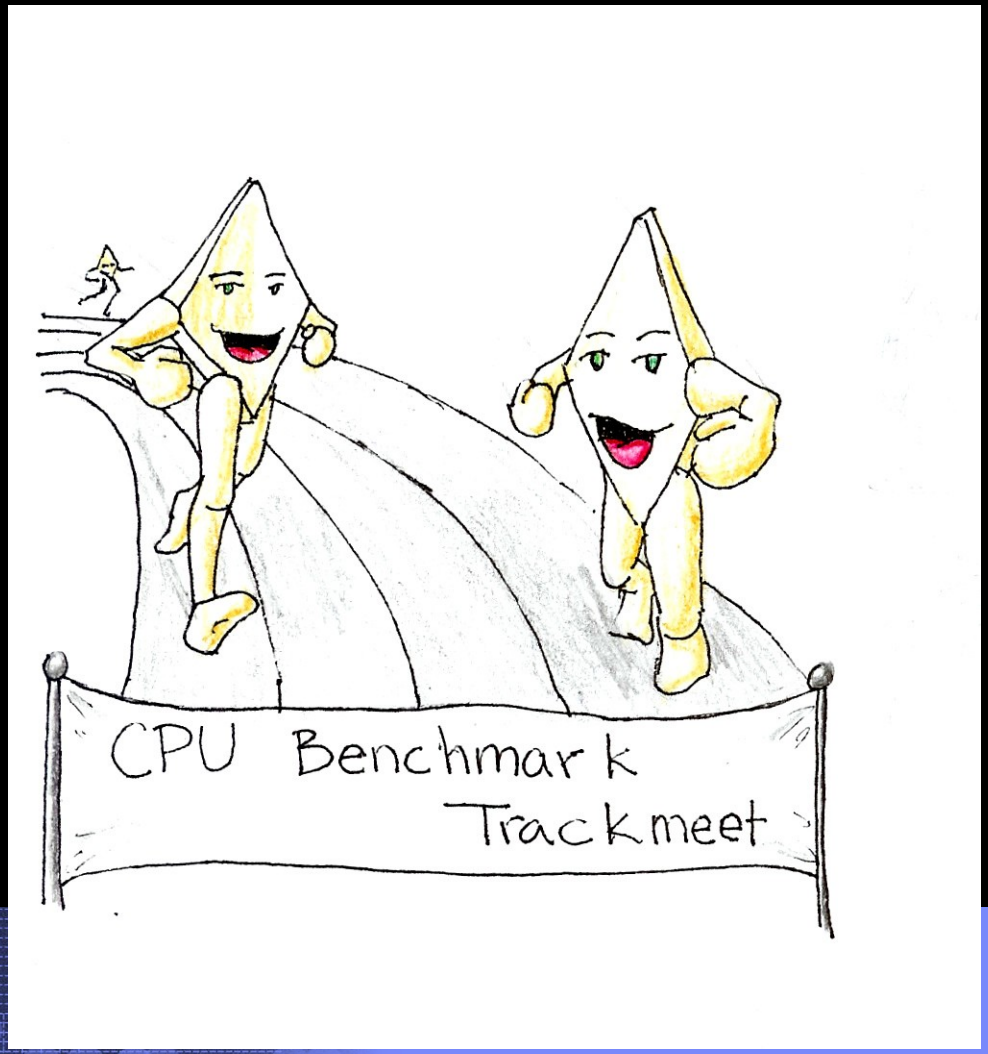


Electrons move at 0.03C to 0.3C in transistors and, so lots of waiting. 3D???

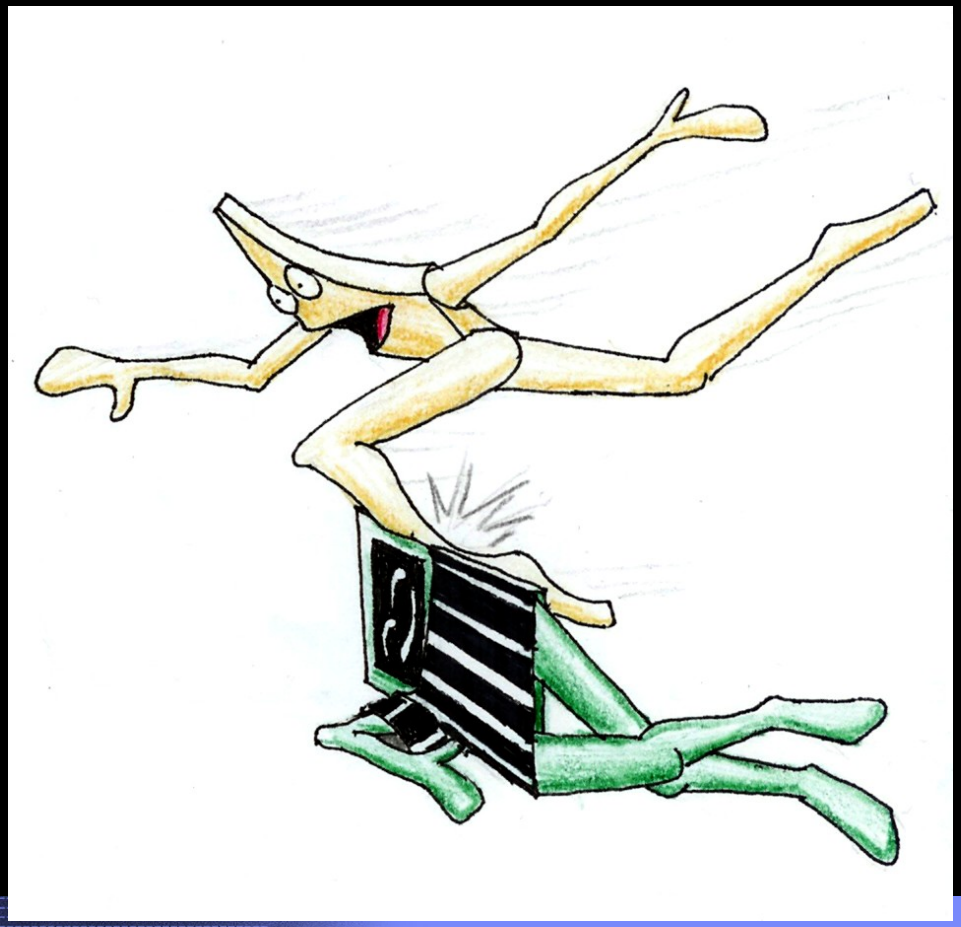
Visual Demonstration of Instruction Overhead

The Bogroll Demonstration

CPU Performance: The Marketing Pitch

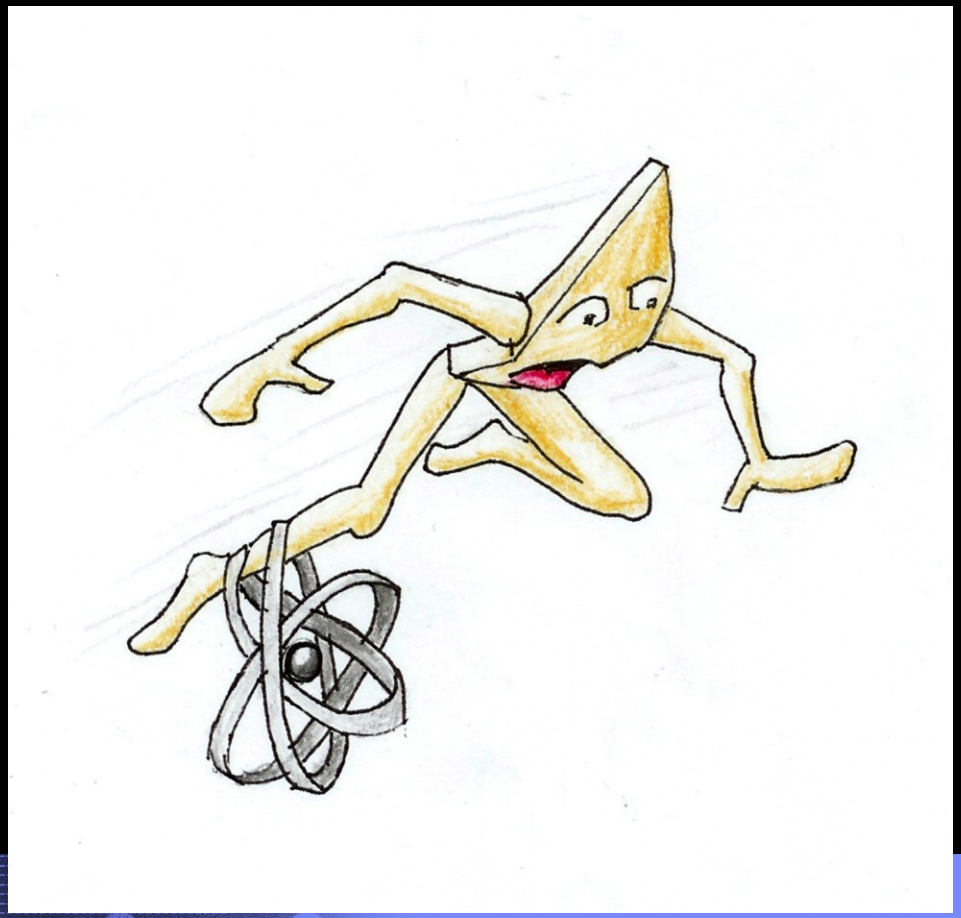


CPU Performance: Memory References



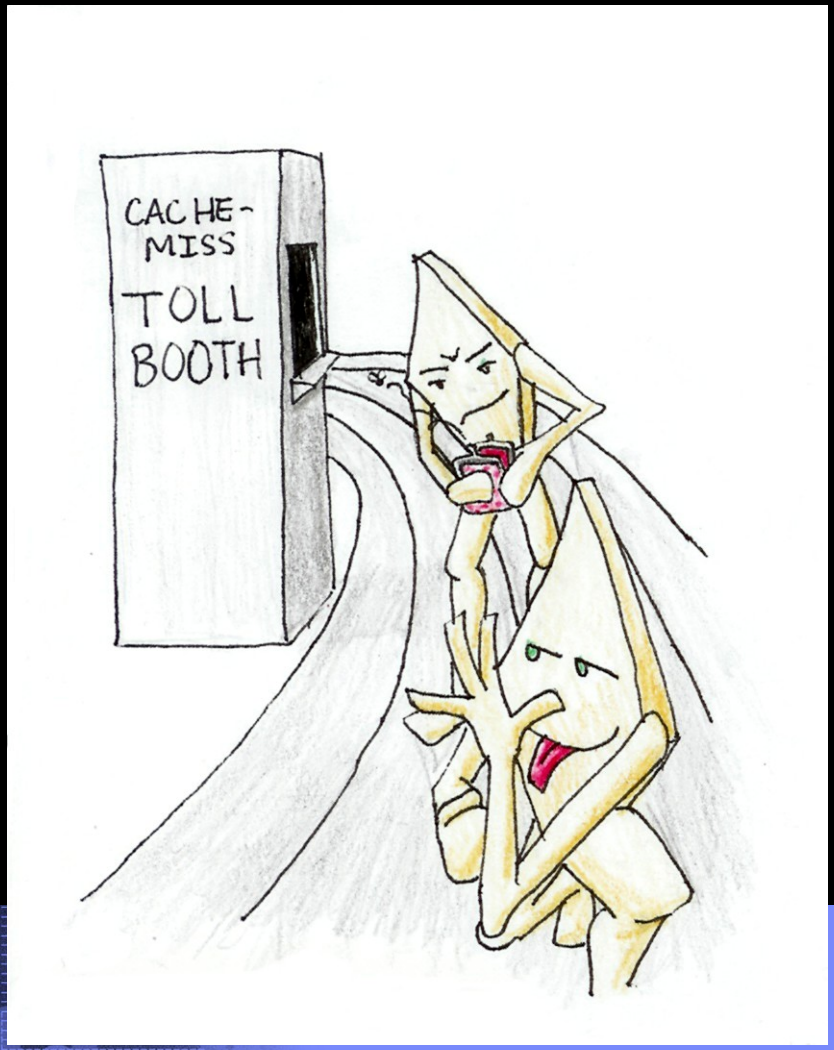
CPU Performance: Pipeline Flushes

CPU Performance: Atomic Instructions



CPU Performance: Memory Barriers

CPU Performance: Cache Misses

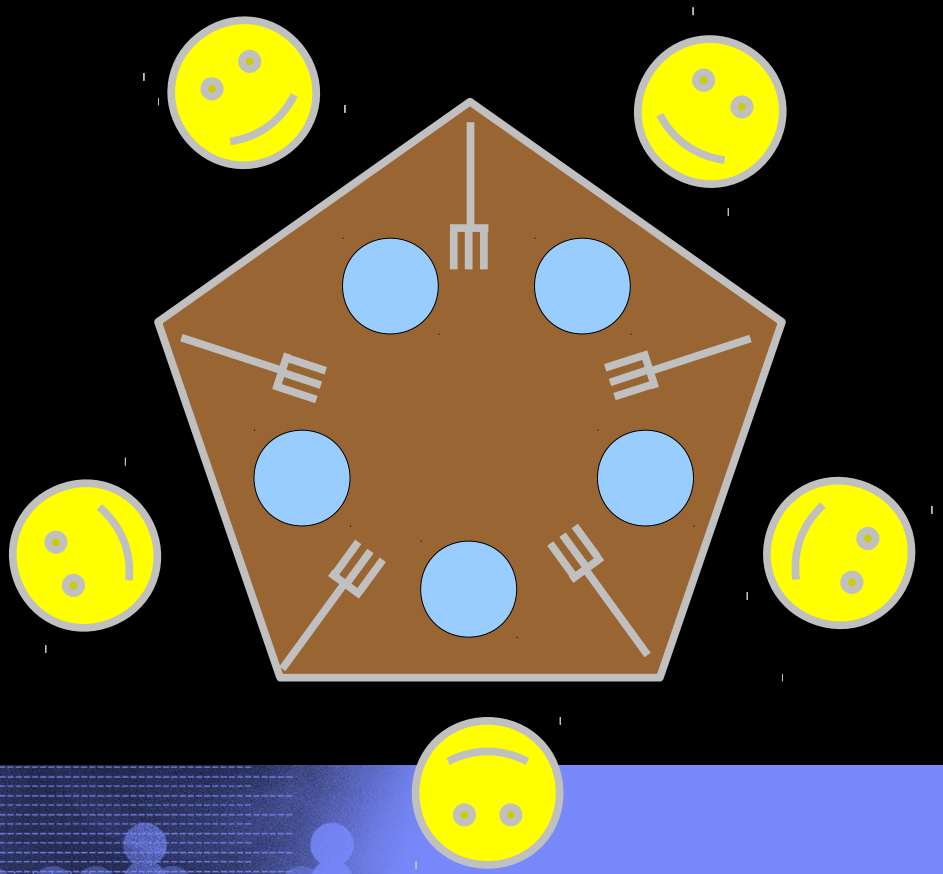


CPU Performance: I/O

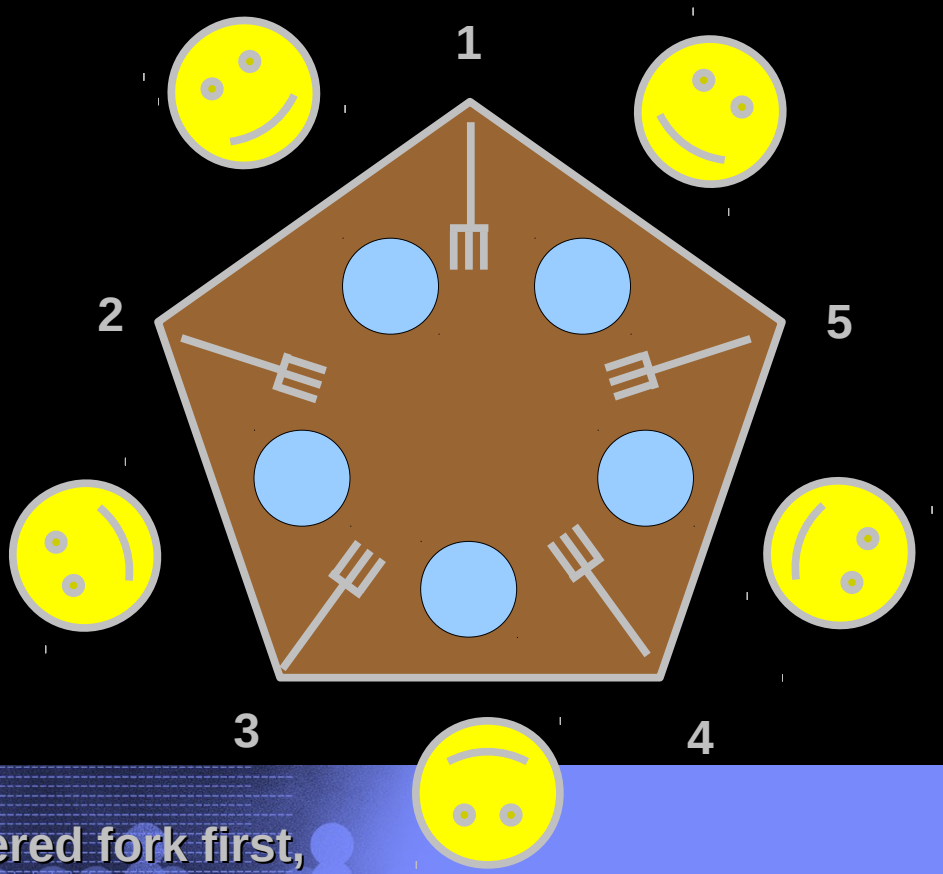
So We Need to Break Things Up...

Exercise: Dining Philosophers Problem

Each philosopher requires two forks to eat.
Need to avoid starvation.



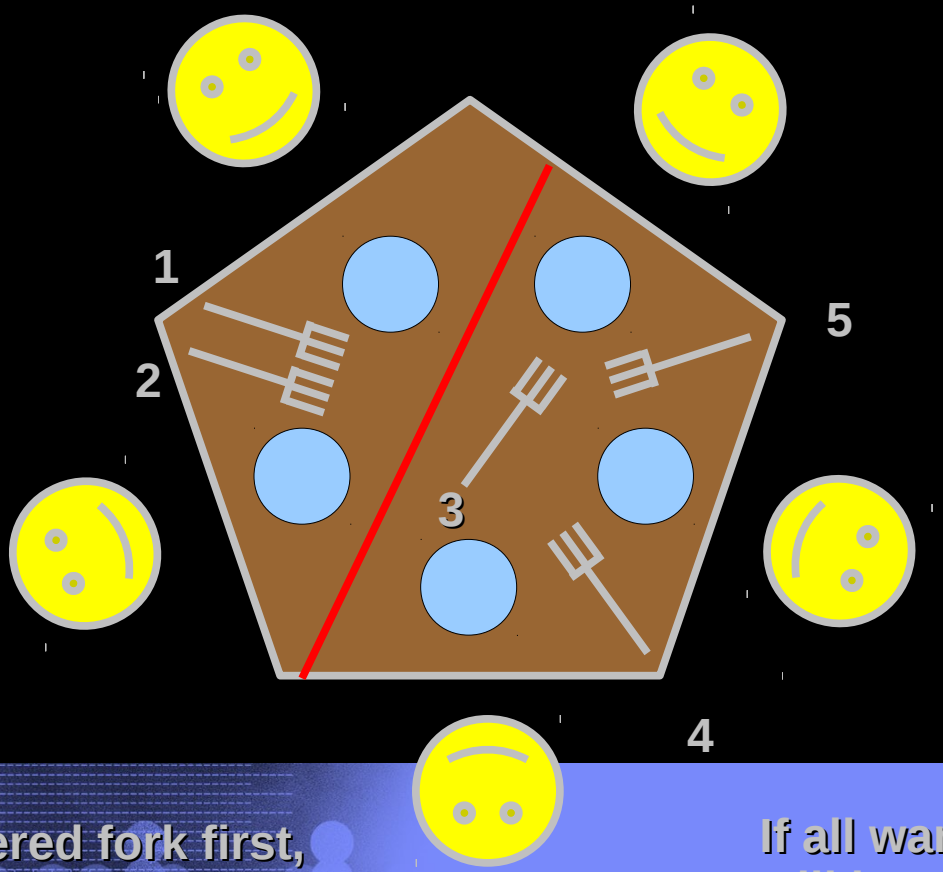
Exercise: Dining Philosophers Solution #1



Locking hierarchy.
Pick up low-numbered fork first, preventing deadlock.

Is this a good solution???

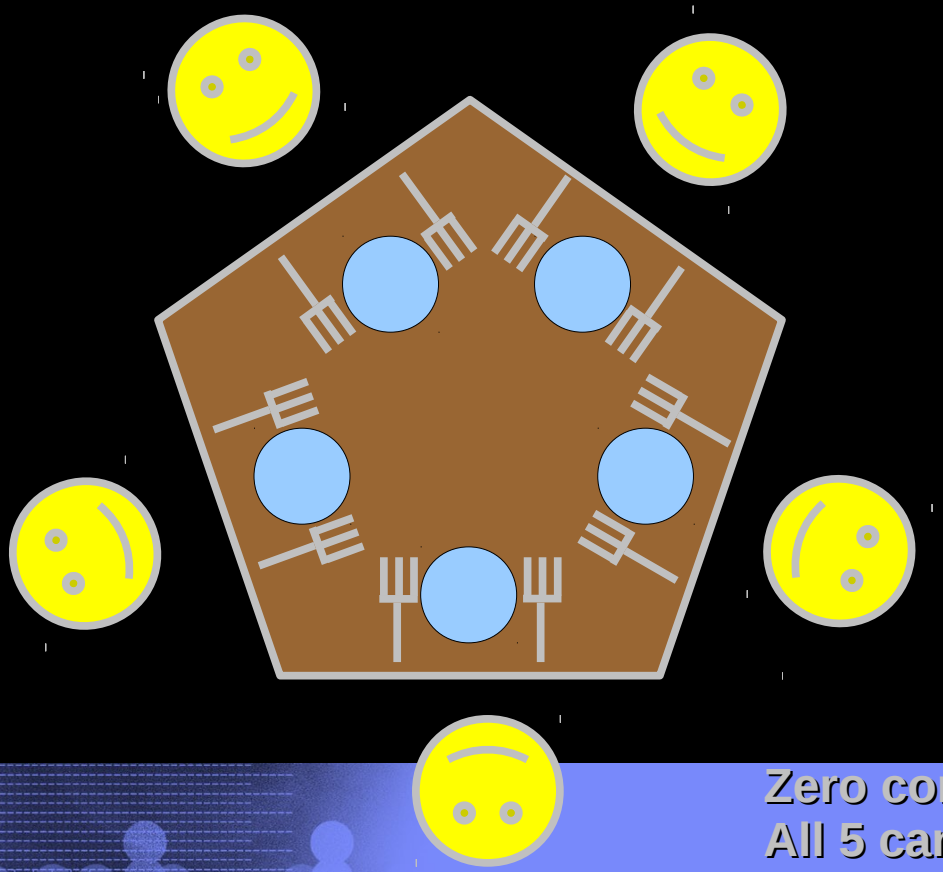
Exercise: Dining Philosophers Solution #2



Locking hierarchy.
Pick up low-numbered fork first, preventing deadlock.

If all want to eat, at least two will be able to do so.

Exercise: Dining Philosophers Solution #3



Zero contention.
All 5 can eat concurrently.
Excellent disease control.

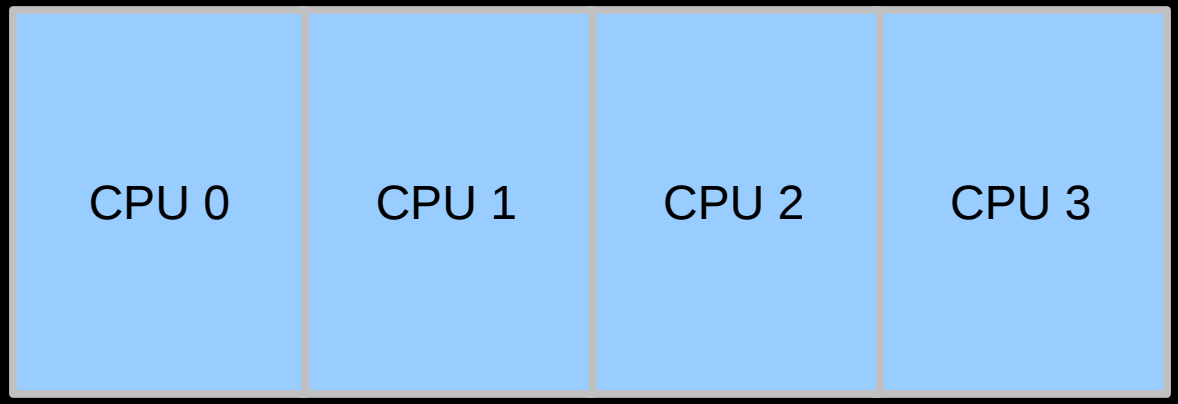
Exercise: Dining Philosophers Solutions

- **Objections to solution #2 and #3:**
  - **"You can't just change the rules like that!!!"**
    - No rule against moving or adding forks!!!
  - **"Dining Philosophers Problem valuable lock-hierarchy teaching tool – #3 just destroyed it!!!"**
    - Lock hierarchy is indeed very valuable and widely used, so the restriction "there can only be five forks positioned as shown" does indeed have its place, even if it didn't appear in this instance of the Dining Philosophers Problem.
    - But the lesson of transforming the problem into perfectly partitionable form is also very valuable, and given the wide availability of cheap multiprocessors, most desperately needed.
  - **"But what if each fork cost a million dollars?"**
    - Then we make the philosophers eat with their fingers... ☺

But What To Do...

- If you have a problem that does not partition nicely????

Embarrassingly Parallel



Per-CPU variables
Per-task variables
Per-device structures
...

If Cannot Fully Partition

- Use per-CPU/per-task caching
  - memory allocation, limit-aware counting
  - Reduce frequency of global interaction
- Use periodic update (e.g., load balancing)
  - Reduce frequency of global interaction
  - Give up some accuracy or responsiveness
- Perhaps random() is your friend
  - Coordination more expensive than it is worth

Overview

- What the #$I#@(&!!! is RCU-bh for???
- RCU status in mainline
- Breakage for performance and scalability

Questions?

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.
- This material is based upon work supported by the National Science Foundation under Grant No. CNS-0719851.
  - Joint work with Manish Gupta, Maged Michael, Phil Howard, Joshua Triplett, and Jonathan Walpole