Verification of Tree-Based Hierarchical Read-Copy Update in the Linux Kernel

Paul E. McKenney, IBM Linux Technology Center

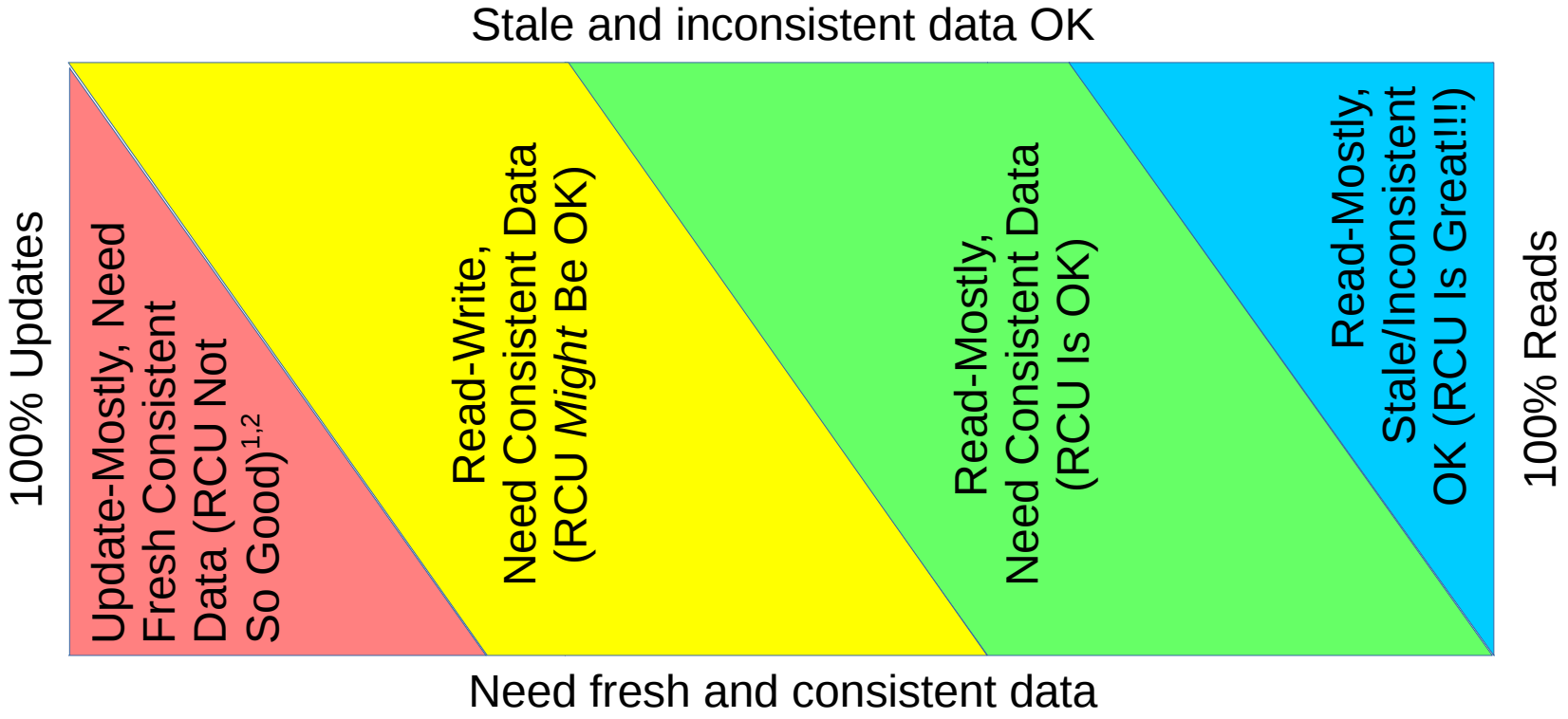
Joint work with Lihao Liang, Daniel Kroening, and Tom Melham, University of Oxford*

What Is RCU?

What Is RCU?

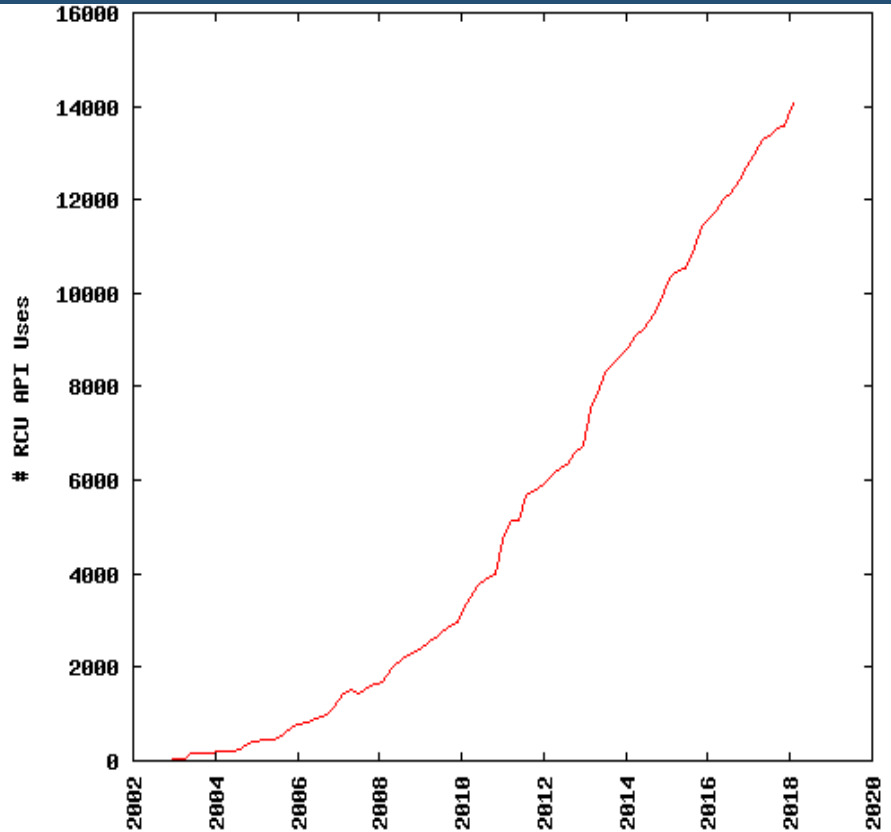
- **Synchronization primitive used in Linux kernel**
  - **Heavily used, and gaining use elsewhere as well**
- **Some implementations do read-only traversal of linked data structures using exactly the same sequence of machine instructions used in the absence of updates**
  - **Readers get excellent performance, scalability, …**
  - **Complex and highly concurrent implementation**
- **http://www.rdrop.com/users/paulmck/RCU/**

RCU Is Specialized: Area of Applicability



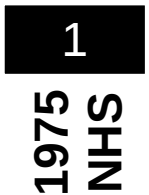
1. RCU provides ABA protection for update-friendly mechanisms
2. RCU provides bounded wait-free read-side primitives for real-time use

What Exactly Does “Heavily Used” Mean?



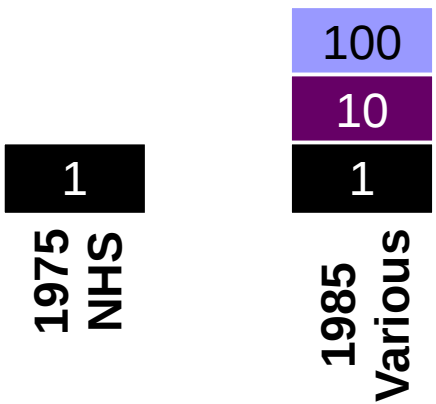
Isn't Making Software Work A Solved Problem?

Million-Year Bug: Once Per Million Years



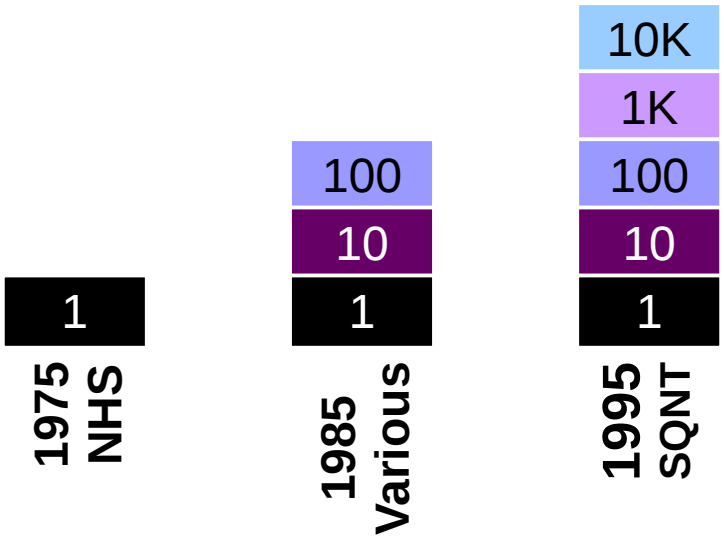
Isn't Making Software Work A Solved Problem?

Million-Year Bug: Once In Ten Millennia



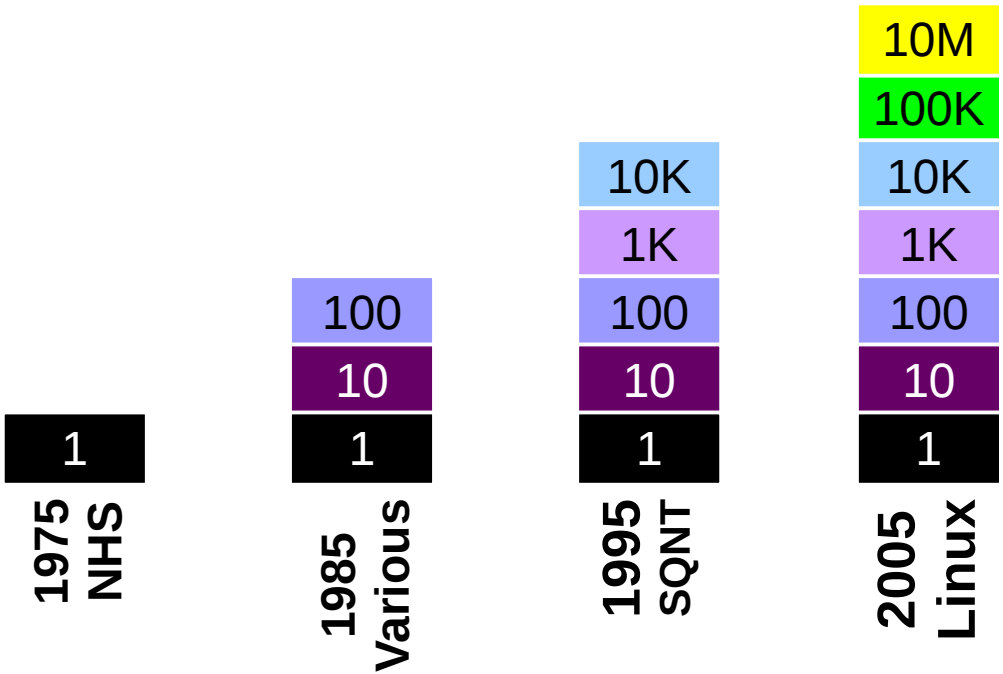
Isn't Making Software Work A Solved Problem?

Million-Year Bug: Once Per Century



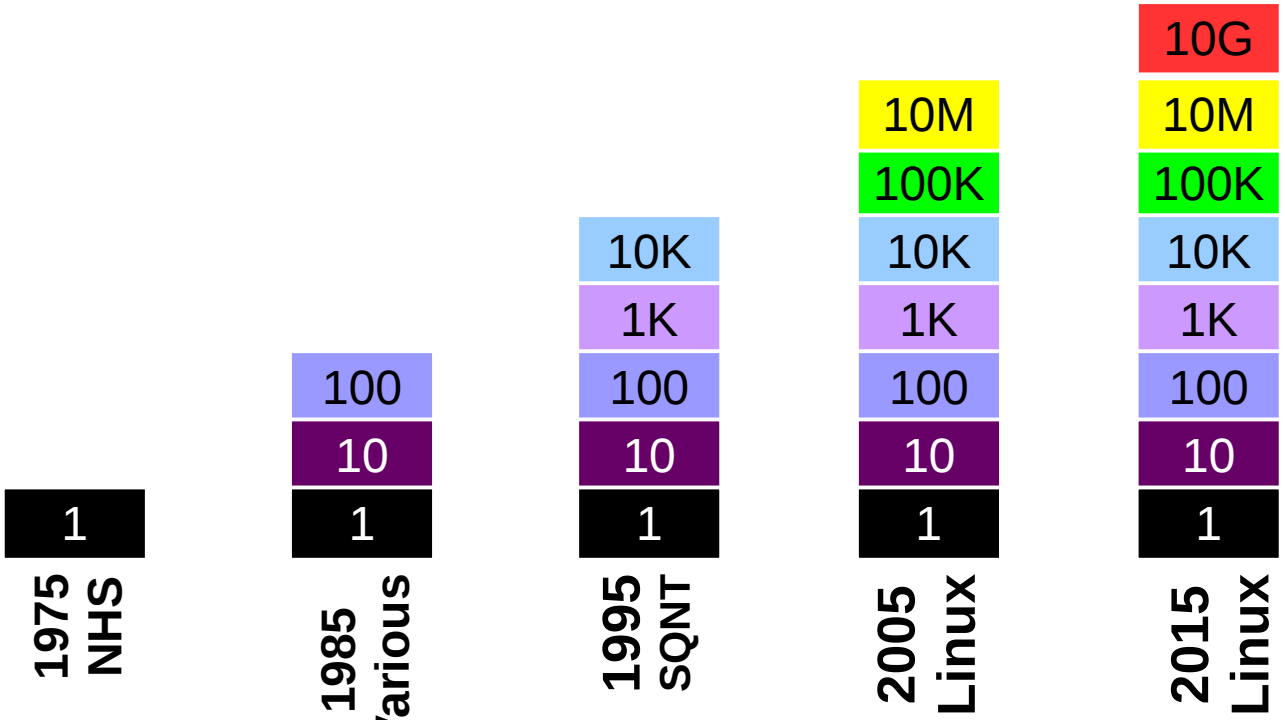
Isn't Making Software Work A Solved Problem?

Million-Year Bug: Once a Month



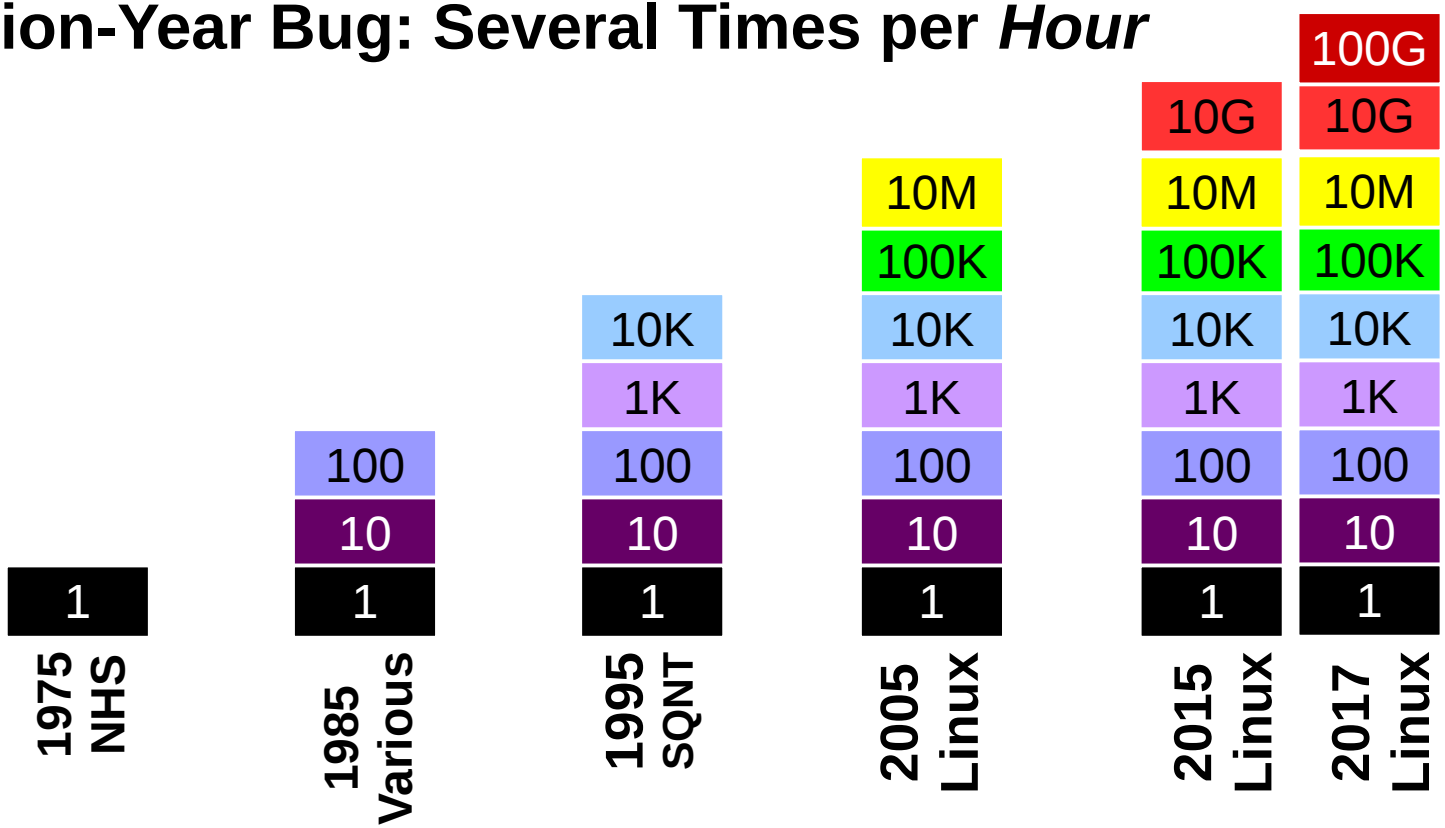
Isn't Making Software Work A Solved Problem?

Million-Year Bug: Several Times per *Day*



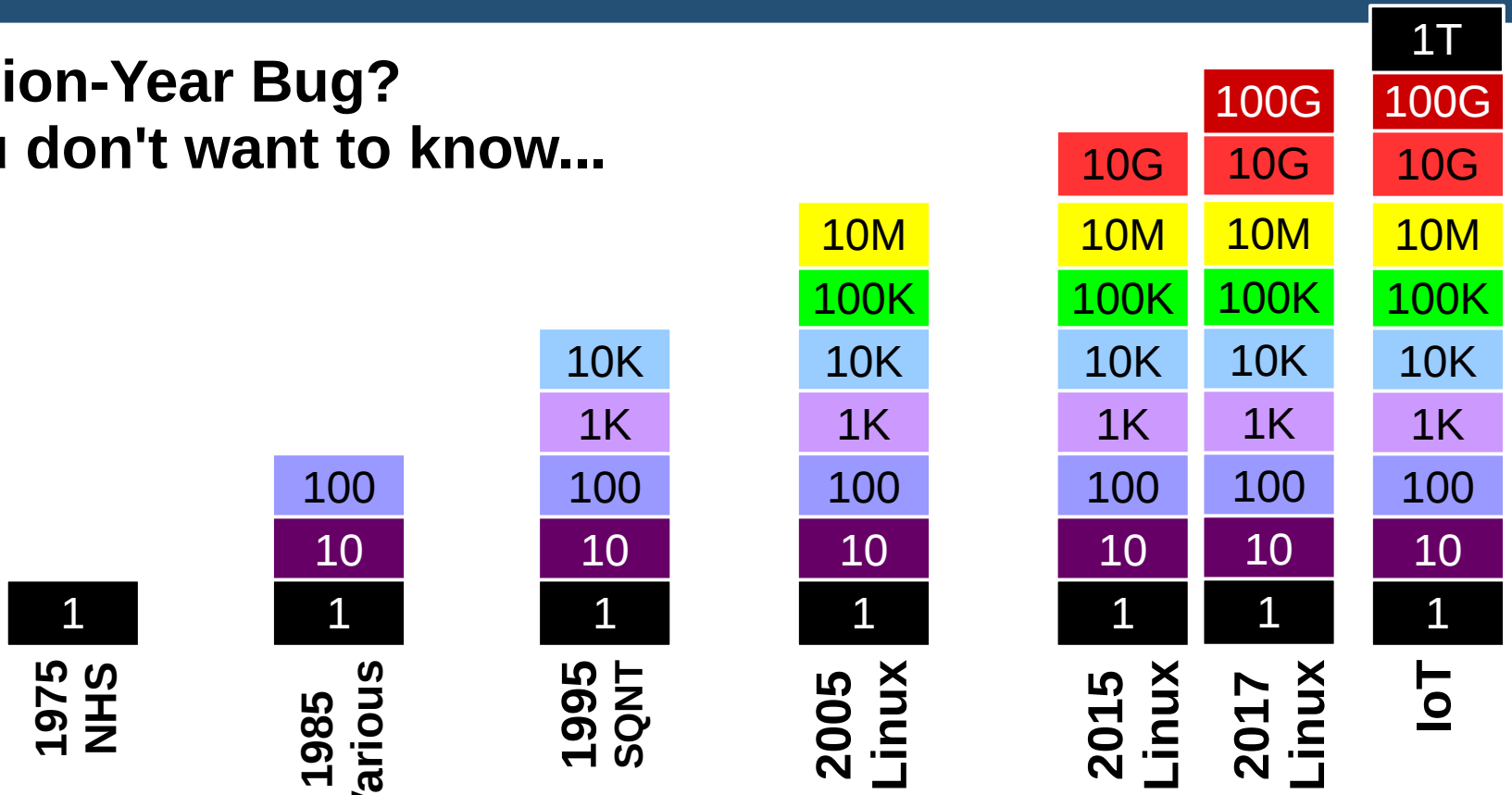
Isn't Making Software Work A Solved Problem?

Million-Year Bug: Several Times per *Hour*



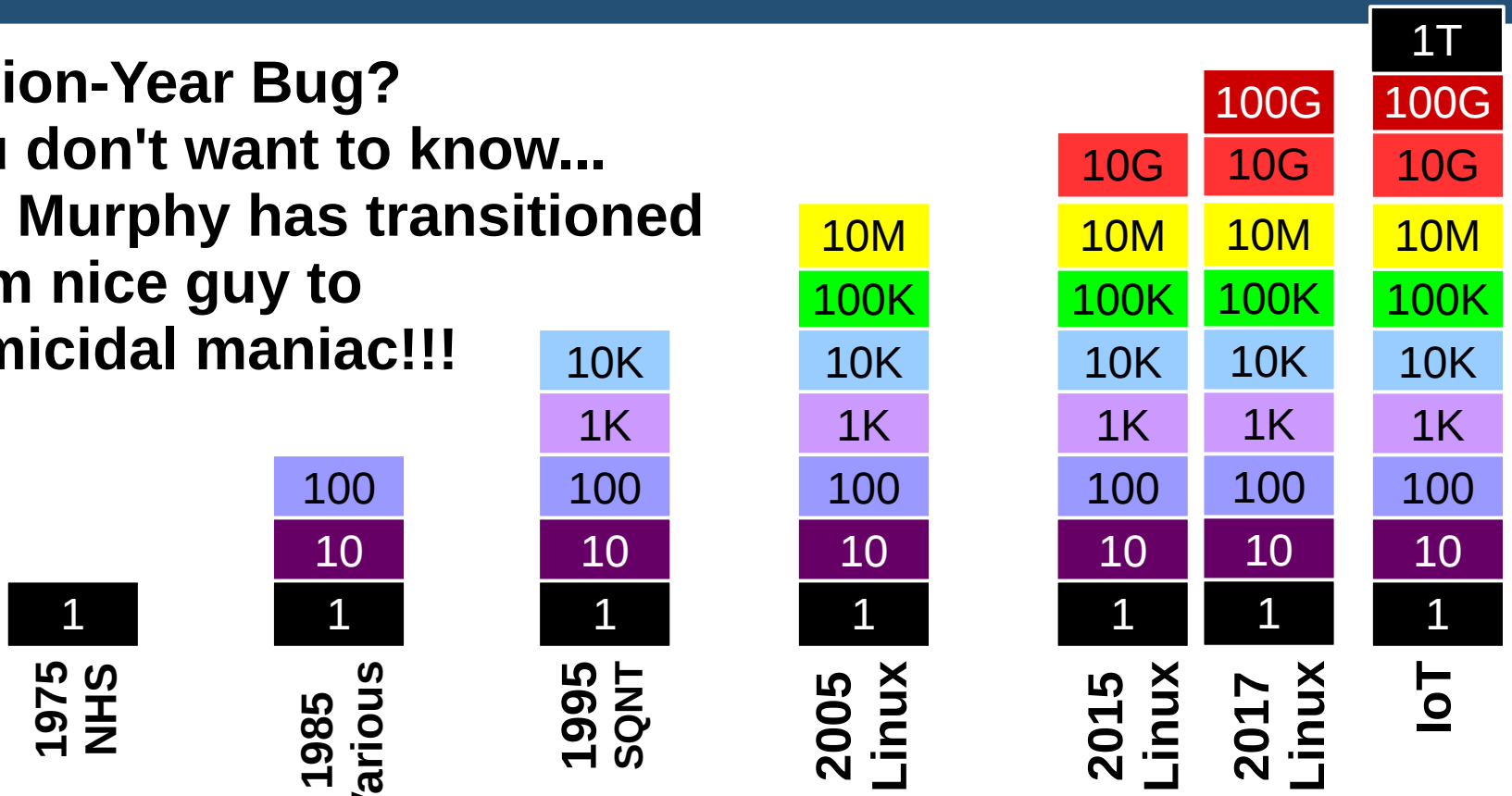
Isn't Making Software Work A Solved Problem?

Million-Year Bug?
You don't want to know...



Isn't Making Software Work A Solved Problem?

Million-Year Bug?
You don't want to know...
But Murphy has transitioned from nice guy to homicidal maniac!!!



Why Stress About Potential Low-Probability Bugs?

- Almost any bug can become a security exploit
  - Internet: Physical presence no longer required
  - Not restricted to software: Meltdown and Spectre
    - RCU is not the only thing with empirical spec!
- RCU is low level does not imply low risk
  - After all, Row Hammer hit DRAM!
- Might be a trillion IoT devices in the World
  - Translates to huge numbers of failures
  - Some of which might put the general public at risk
- RCU is well-contained test case for PoC

Why Not Try Formal Verification?

Why Not Try Formal Verification?

In Linux-Kernel RCU's Regression Tests...

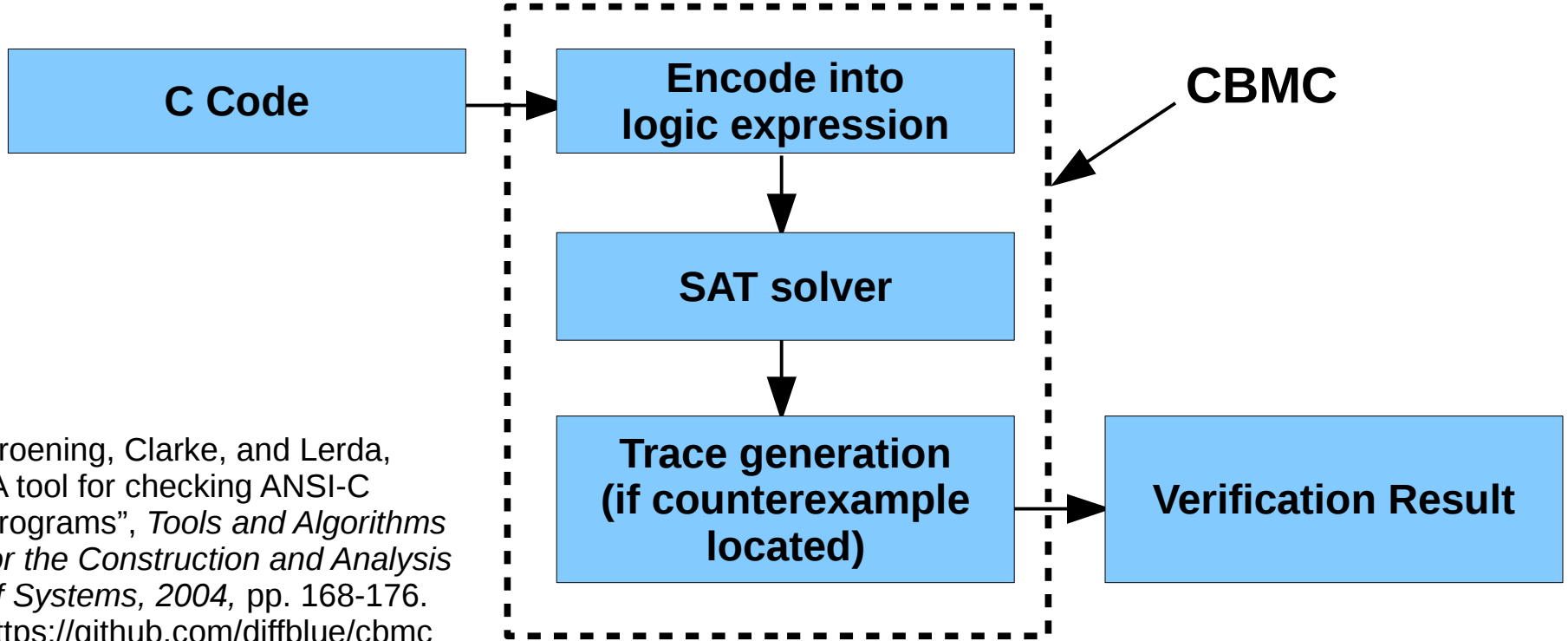
Formal Verification & Regression Tests: Requirements

- **Either automatic or no translation**
- **Correctly handle environment: memory model!**
- **Reasonable memory and CPU overhead**
- **Map back to lines of code containing bugs**
- **Main input: source code under test**
- **Find relevant bugs**

Scorecard for Linux-Kernel C Code

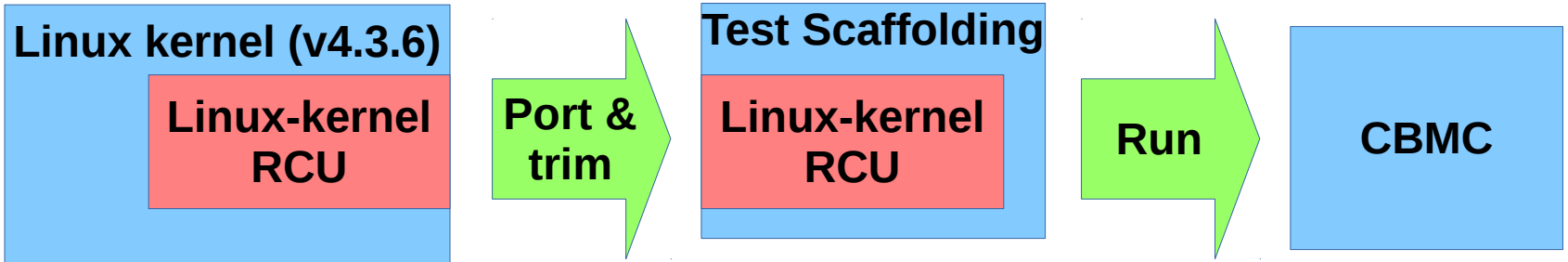
	Promela	PPCMEM	Herd	CBMC	Test
(1) Automated					
(2) Handle environment	(MM)		(MM)	(MM)	
(3) Low overhead				SAT?	
(4) Map to source code					
(5) Modest input					
(6) Relevant bugs	???	???	???	???	
Paul McKenney's first use	1993	2011	2014	2015	1973

CBMC (Very) Rough Schematic



Kroening, Clarke, and Lerda, "A tool for checking ANSI-C Programs", *Tools and Algorithms for the Construction and Analysis of Systems, 2004,* pp. 168-176. https://github.com/diffblue/cbmc

Applying CBMC to Linux-Kernel RCU



- Reflects server-class RCU up to 16 CPUs with default config (32 or 64 CPUs w/non-default configs)
- Approximated interrupts & grace-period kthread w/hand-placed function calls, specified per-loop unrolling limits
- Modeled per-CPU variables with arrays
- Modeled locks with __CPROVER_atomic*()
- Memory consistency models: SC, TSO, and PSO
- Tested safety and a weak form of liveness

Healthy Skepticism

Healthy Skepticism

- **Formal verification of Linux-kernel RCU?**

Healthy Skepticism

- **Formal verification of Linux-kernel RCU?**
  - **Sure, I can also write printf(“VERIFIED\n”);**

Healthy Skepticism

- **Formal verification of Linux-kernel RCU?**
  - **Sure, I can also write printf(“VERIFIED\n”);**
- **I therefore maintain bug-injected RCU versions**
  - **https://paulmck.livejournal.com/46993.html**

Healthy Skepticism

- **Formal verification of Linux-kernel RCU?**
  - **Sure, I can also write printf(“VERIFIED\n”);**
- **I therefore maintain bug-injected RCU versions**
  - **https://paulmck.livejournal.com/46993.html**
- **How did CBMC do?**

Healthy Skepticism

- **Formal verification of Linux-kernel RCU?**
  - **Sure, I can also write printf(“VERIFIED\n”);**
- **I therefore maintain bug-injected RCU versions**
  - **https://paulmck.livejournal.com/46993.html**
- **How did CBMC do? Only 2 failures out of 30.**
  - **Interrupt over-approximation, memory exhaustion**
  - **Up to 90.4M SAT variables, 75GB, ~70 CPU hours**
    - **Ran on 64-bit 2.4GHz Xeon, 12 cores & 96GB memory**

Healthy Skepticism

- **Formal verification of Linux-kernel RCU?**
  - **Sure, I can also write printf(“VERIFIED\n”);**
- **I therefore maintain bug-injected RCU versions**
  - **https://paulmck.livejournal.com/46993.html**
- **How did CBMC do? Only 2 failures out of 30.**
  - **Interrupt over-approximation, memory exhaustion**
  - **Up to 90.4M SAT variables, 75GB, ~70 CPU hours**
    - **Ran on 64-bit 2.4GHz Xeon, 12 cores & 96GB memory**
- **But did not find new-to-me bugs!**

Summary and Challenges

Summary

- **Linux-kernel RCU robustness is important**
  - **Large installed base poses severe challenge**
- **First automated LK RCU formal verification**
  - **Two other teams have since done similar work**
- **Formal verification in regression tests: Almost**
  - **Future work: Find bugs I don't already know about!**
- **Nevertheless, this work demonstrates the nascent ability and potential of SAT-based formal-verification tools to handle real-world production-quality synchronization primitives**

Challenges/Limitations/Future Work

- **Better modeling of interrupts & kernel threads**
- **Model concurrent linked lists: call_rcu()**
- **Incorporate Linux-kernel memory model**
  - **And/or ARM, PowerPC, RISC-V, ...**
- **Forward progress: Detect hangs & deadlocks**
  - **Can already detect unconditional hangs/deadlocks**
- **Fully analyze unbounded looping**
  - **Or at least automatically derive unrolling bounds**
- **Larger programs: Automatic decomposition?**

Additional Challenges

- **Find bug in rcu_preempt_offline_tasks()**
  - **http://paulmck.livejournal.com/37782.html**
- **Find bug in RCU_NO_HZ_FULL_SYSIDLE**
  - **http://paulmck.livejournal.com/38016.html**
- **Find bug in RCU linked-list use cases**
  - **http://paulmck.livejournal.com/39793.html**
- **Verification Challenge 6**
  - **http://paulmck.livejournal.com/46993.html**
- **Find bugs in other popular open-source SW**

Legal Statement

- **This work represents the view of the authors and does not necessarily represent the view of IBM or University of Oxford.**
- **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?