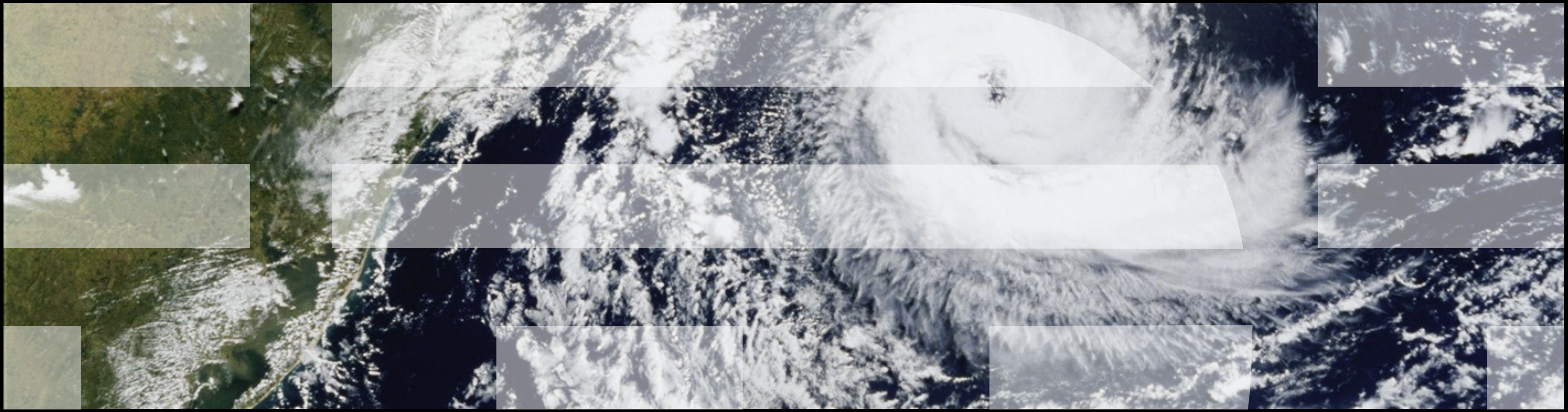
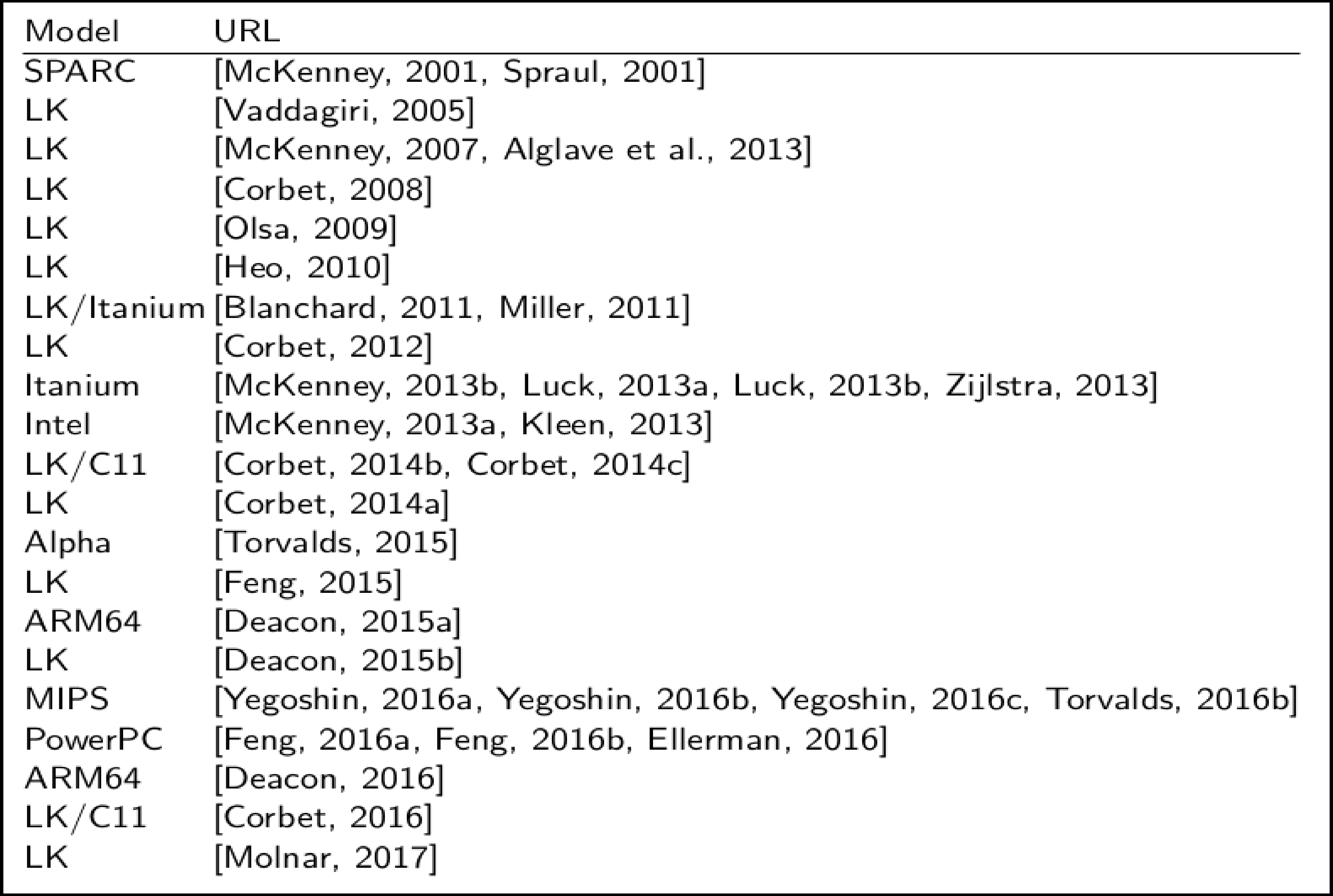
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center
Member, IBM Academy of Technology
SYSTOR Highlights Track, May 4, 2018

Frightening small children and disconcerting grown-ups: Concurrency in the Linux kernel

Jade Alglave (University College London & Microsoft Research); Luc Maranget (INRIA); Paul E. McKenney (IBM Corporation & Oregon State University); Andrea Parri (Scuola Superiore Sant'Anna); Alan Stern (Harvard University)



Concurrency In Linux Can Be a Contentious Topic



Model	URL
SPARC	[McKenney, 2001, Spraul, 2001]
LK	[Vaddagiri, 2005]
LK	[McKenney, 2007, Alglave et al., 2013]
LK	[Corbet, 2008]
LK	[Olsa, 2009]
LK	[Heo, 2010]
LK/Itanium	[Blanchard, 2011, Miller, 2011]
LK	[Corbet, 2012]
Itanium	[McKenney, 2013b, Luck, 2013a, Luck, 2013b, Zijlstra, 2013]
Intel	[McKenney, 2013a, Kleen, 2013]
LK/C11	[Corbet, 2014b, Corbet, 2014c]
LK	[Corbet, 2014a]
Alpha	[Torvalds, 2015]
LK	[Feng, 2015]
ARM64	[Deacon, 2015a]
LK	[Deacon, 2015b]
MIPS	[Yegoshin, 2016a, Yegoshin, 2016b, Yegoshin, 2016c, Torvalds, 2016b]
PowerPC	[Feng, 2016a, Feng, 2016b, Ellerman, 2016]
ARM64	[Deacon, 2016]
LK/C11	[Corbet, 2016]
LK	[Molnar, 2017]

“Still confusion situation all round”[sic] [Ziljstra, 2013]

Existing Documentation

- [Howells et al., 2017] lists what orderings are guaranteed;
- [Miller, 2017] summarises semantics of read-modify-writes;
- [McKenney, 2017a] documents ways of avoiding counterproductive optimisations.

But... [Gorman, 2013]

If Documentation/memory-barriers.txt could not be used to frighten small children before, it certainly can now.

Also [Howells et al., 2017]:

This document is not a specification; it is intentionally (for the sake of brevity) and unintentionally (due to being human) incomplete. [. . .] in case of any doubt (and there are many) please ask.

And Anyway [Torvalds, 2012]

With specs, there really *are* people who spend years discussing what the meaning of the word ”access” is or similar [. . .]. Combine that with a big spec that is 500+ pages in size and then try to apply that all to a project that is 15 million lines of code and sometimes *knowingly* has to do things that it simply knows are outside the spec [. . .]”

What We Offer

- A formal consistency model
- Written in the cat language
- Thus executable within the herd tool

“[I]t is your kernel, so what is your preference?” [McKenney, 2016a]

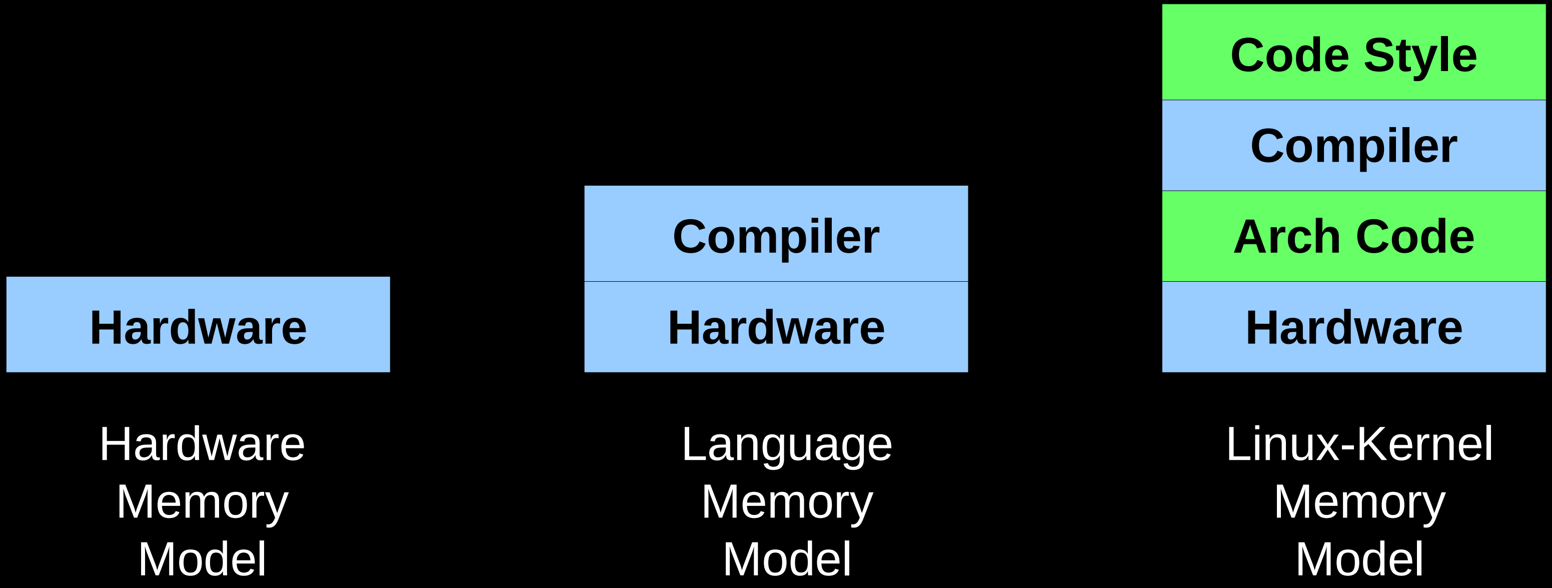
A Common Denominator of Hardware Models? [Torvalds, 2016a]

Weak memory ordering is [. . .] hard to think about [. . .] So the memory ordering rules should [. . .] absolutely be as tight as at all humanly possible, given real hardware constraints.

Not an Envelope for the Architectures it Supports? [Molnar, 2013]

it’s not true that Linux has to offer a barrier and locking model that panders to the weakest (and craziest!) memory ordering model amongst all the possible Linux platforms—theoretical or real metal. Instead what we want to do is to consciously, intelligently pick a sane, maintainable memory model and offer primitives for that—at least as far as generic code is concerned. Each architecture can map those primitives to the best of its abilities.

Not an Envelope for the Architectures it Supports? [Molnar, 2013]



The LK Should Have a Model of its Own [Torvalds, 2012]

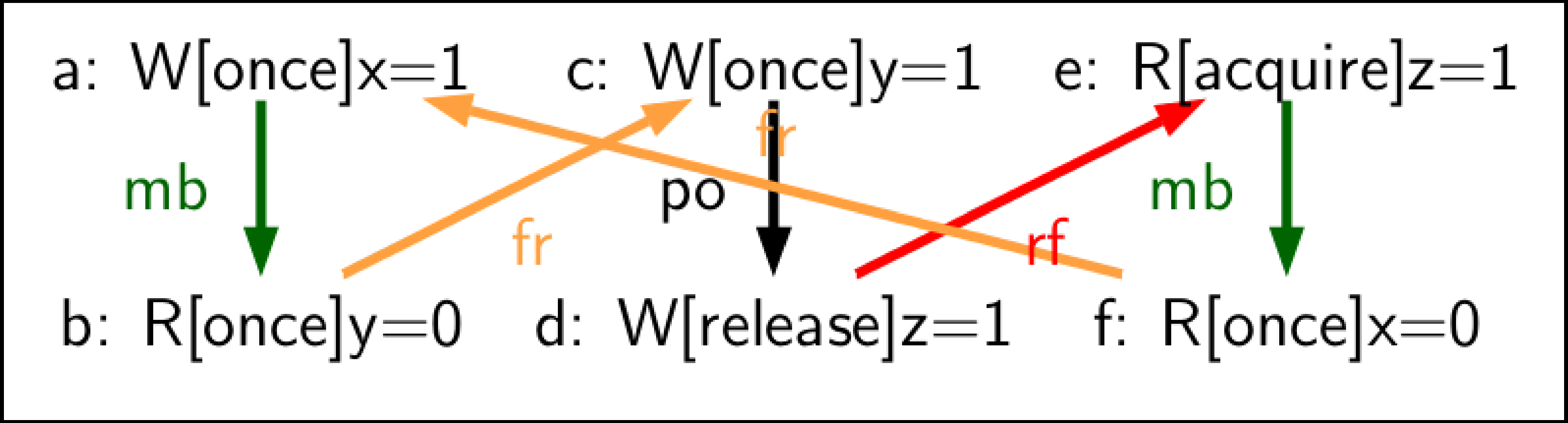
I do not believe for a second that we can actually use the C11 memory model in the kernel [. . .] We will continue to have to do things that are "outside the specs" [. . .] with models that C11 simply doesn't cover.

Core Model

An Example From Peter Zijlstra

https://www.spinics.net/lists/kernel/msg2421883.html

An Example From Peter Zijlstra: Forbidden



(ISA2 from test6.pdf)

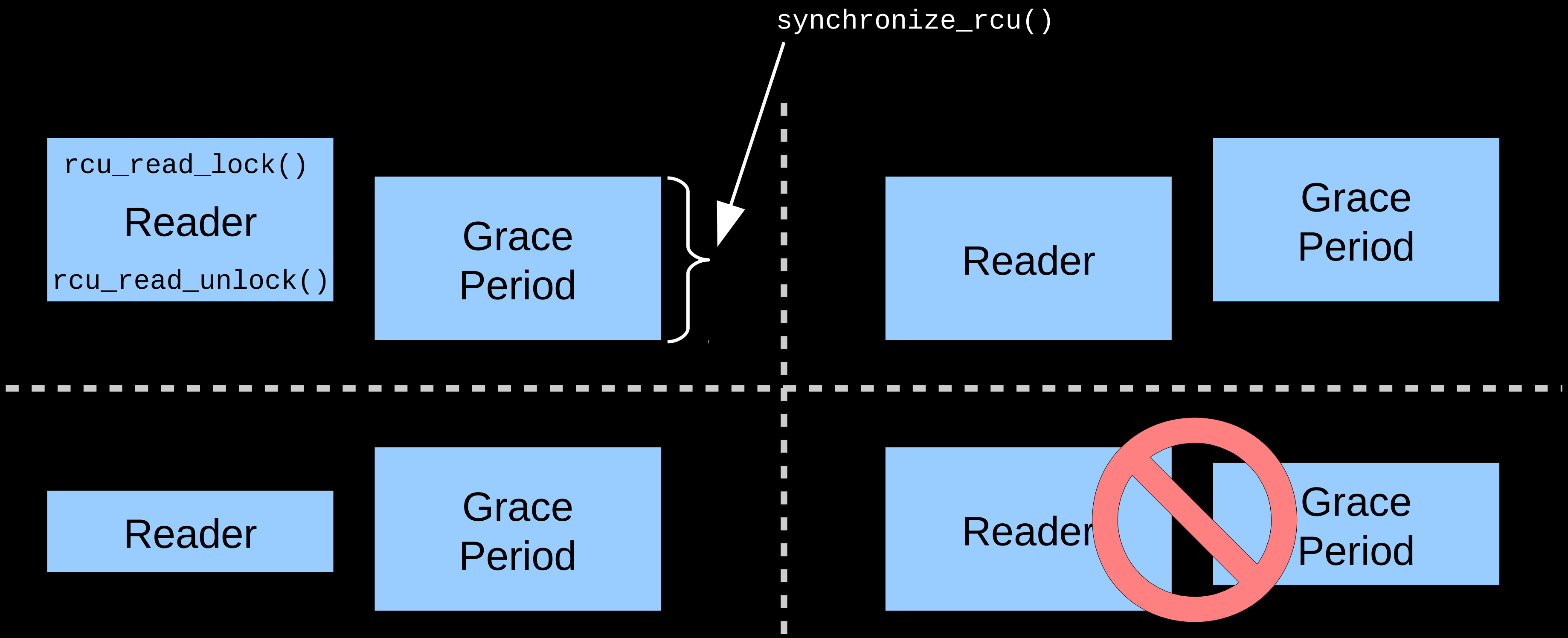
RCU

Fundamental Law of RCU [McKenney et al., 2013]

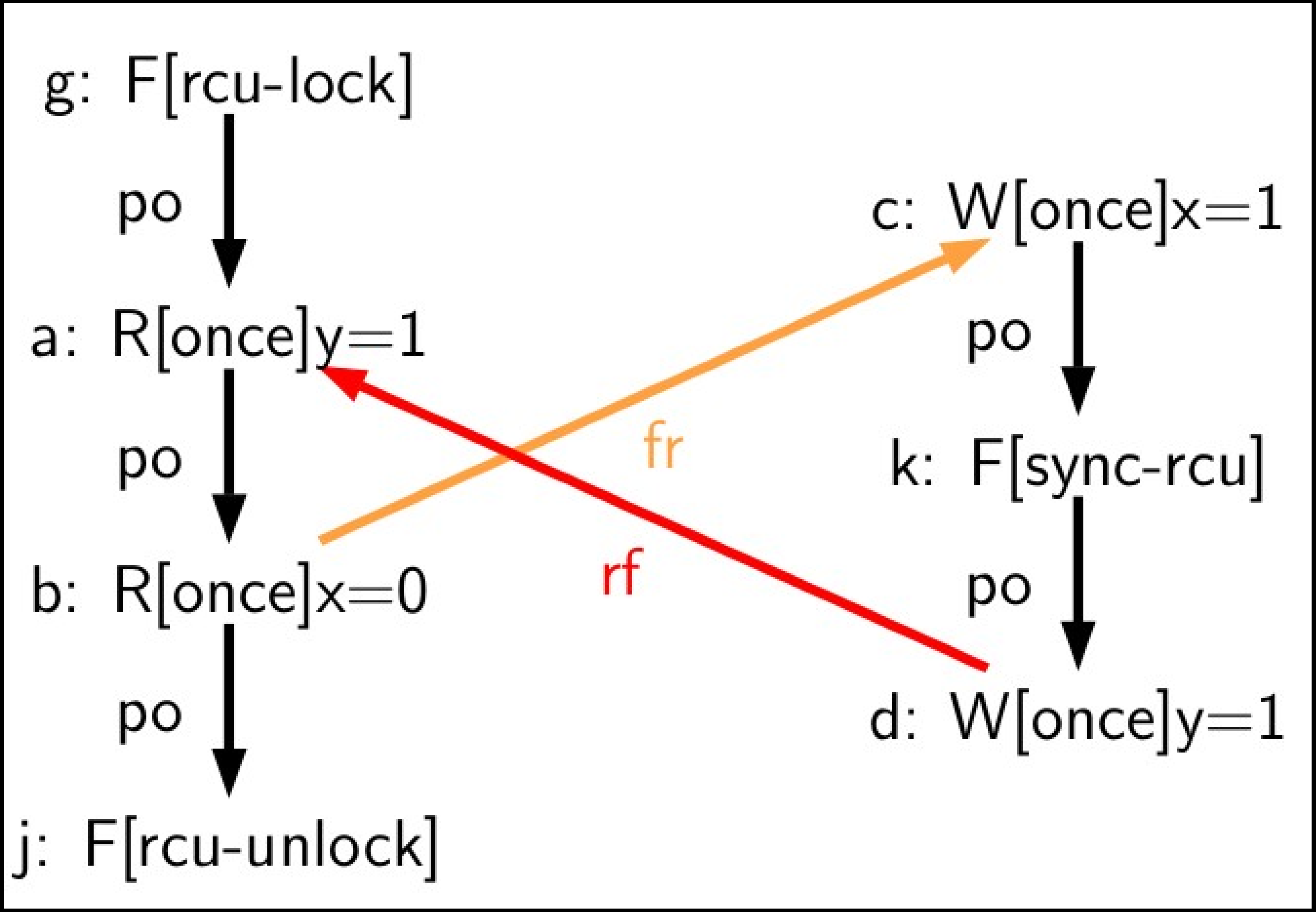
Read-side critical sections cannot span grace periods.

Fundamental Law of RCU [McKenney et al., 2013]

***Read-side critical sections cannot span grace periods*.**



RCU-MP: Forbidden



Validating The Model

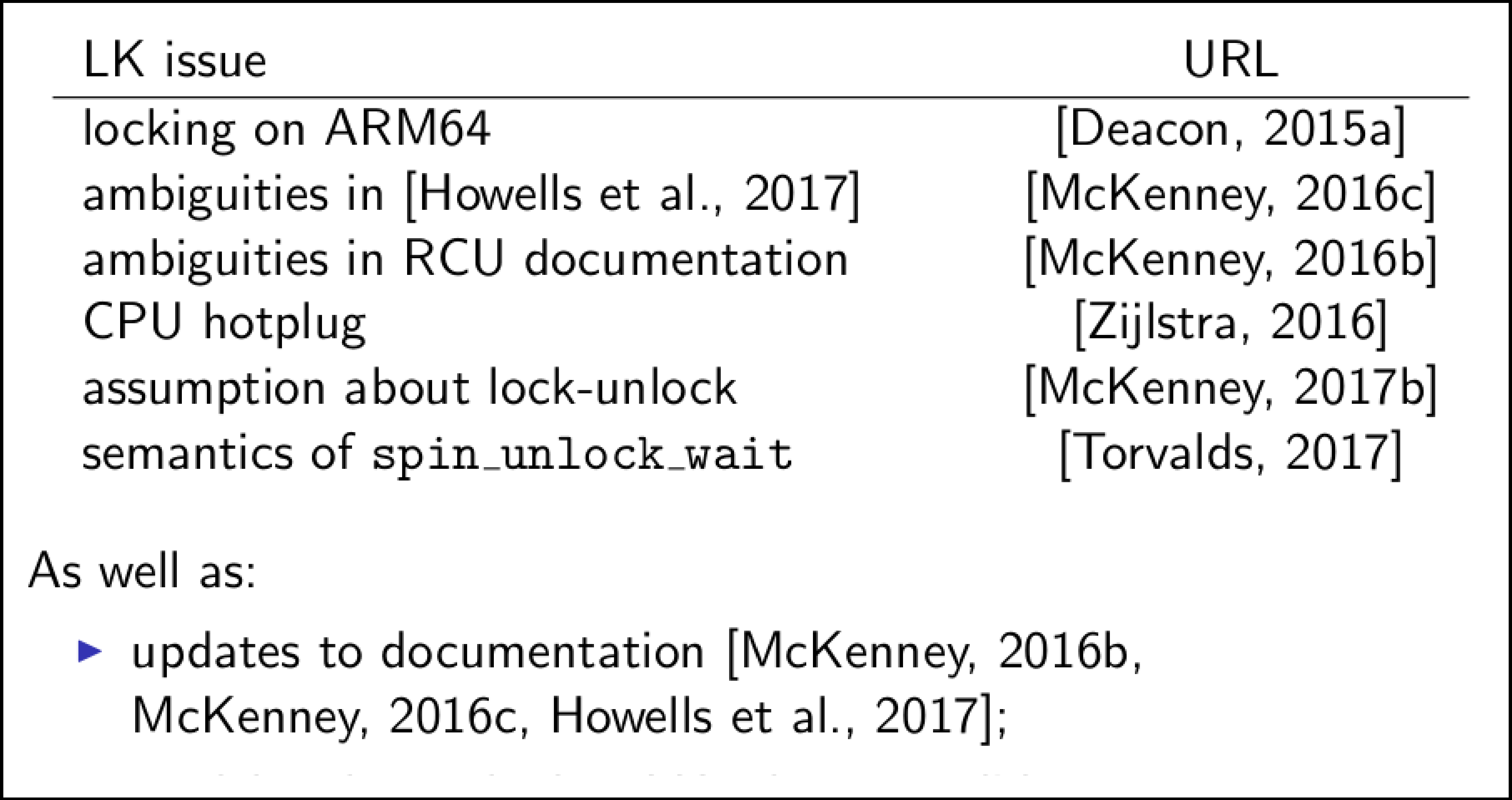
Experimentally

	Model	Power8	ARMv8	ARMv7	X86	C11
LB	Allow	0/15G	0/10G	0/3.0G	0/33G	Allow
LB+ctrl+mb	Forbid	0/17G	0/5.1G	0/4.4G	0/18G	Allow
WRC	Allow	741k/7.7G	13k/5.2G	0/1.6G	0/17G	Allow
WRC+wmb+acq	Allow	0/7.5G	0/4.6G	0/1.6G	0/16G	Forbid
WRC+po-rel+rmb	Forbid	0/7.7G	0/5.3G	0/1.6G	0/17G	Forbid
SB	Allow	4.4G/15G	2.4G/10G	429M/3.0G	765M/33G	Allow
SB+mbs	Forbid	0/15G	0/10G	0/3.0G	0/33G	Forbid
MP	Allow	57M/15G	104M/10G	3.0M/3.0G	0/33G	Allow
MP+wmb+rmb	Forbid	0/15G	0/9.4G	0/2.6G	0/33G	Forbid
PeterZ-No-Synchro	Allow	26M/4.6G	3.6M/900M	10k/380M	351k/7.2G	Allow
PeterZ	Forbid	0/8.7G	0/2.5G	0/2.2G	0/9.1G	Allow
RCU-deferred-free	Forbid	0/256M	0/576M	0/15M	0/25M	—
RCU-MP	Forbid	0/672M	0/336M	0/336M	0/336M	—
RWC	Allow	88M/11G	94M/4.8G	7.5M/1.6G	5.6M/17G	Allow
RWC+mbs	Forbid	0/8.7G	0/2.5G	0/2.2G	0/9.1G	Allow

Socially

Seven maintainers have agreed to chaperon our model.
https://www.spinics.net/lists/kernel/msg2421883.html

Issues That Our Work Helped Address



LK issue	URL
locking on ARM64	[Deacon, 2015a]
ambiguities in [Howells et al., 2017]	[McKenney, 2016c]
ambiguities in RCU documentation	[McKenney, 2016b]
CPU hotplug	[Zijlstra, 2016]
assumption about lock-unlock	[McKenney, 2017b]
semantics of `spin_unlock_wait`	[Torvalds, 2017]

As well as:

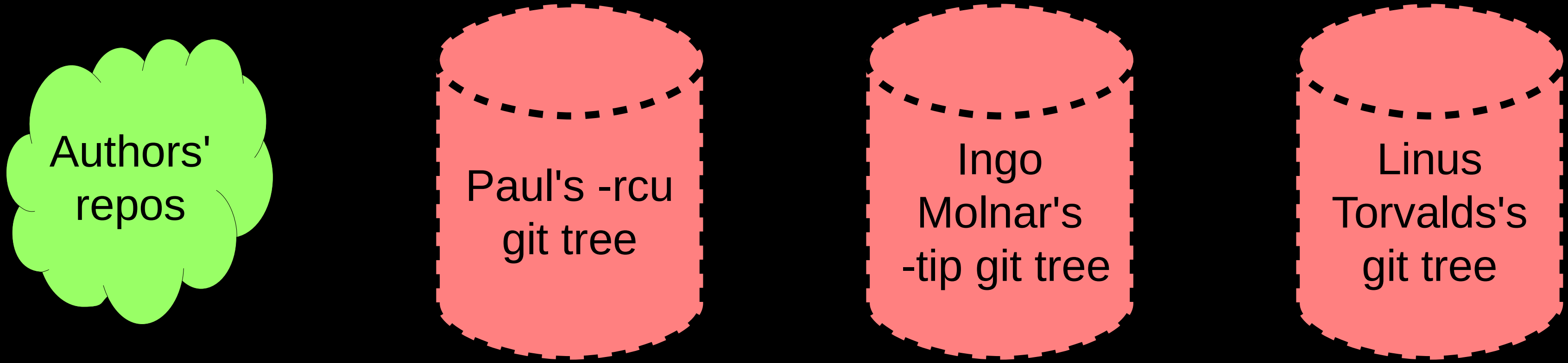
- updates to documentation [McKenney, 2016b, McKenney, 2016c, Howells et al., 2017];

Future Prospects

- Moving target as hardware, workloads, & code style changes
  - RISC-V interaction, fill out locking primitives, plain accesses, …
- Design/education tool, hoped-for use by verification tools
- If this is so great, why isn't it in the Linux kernel???

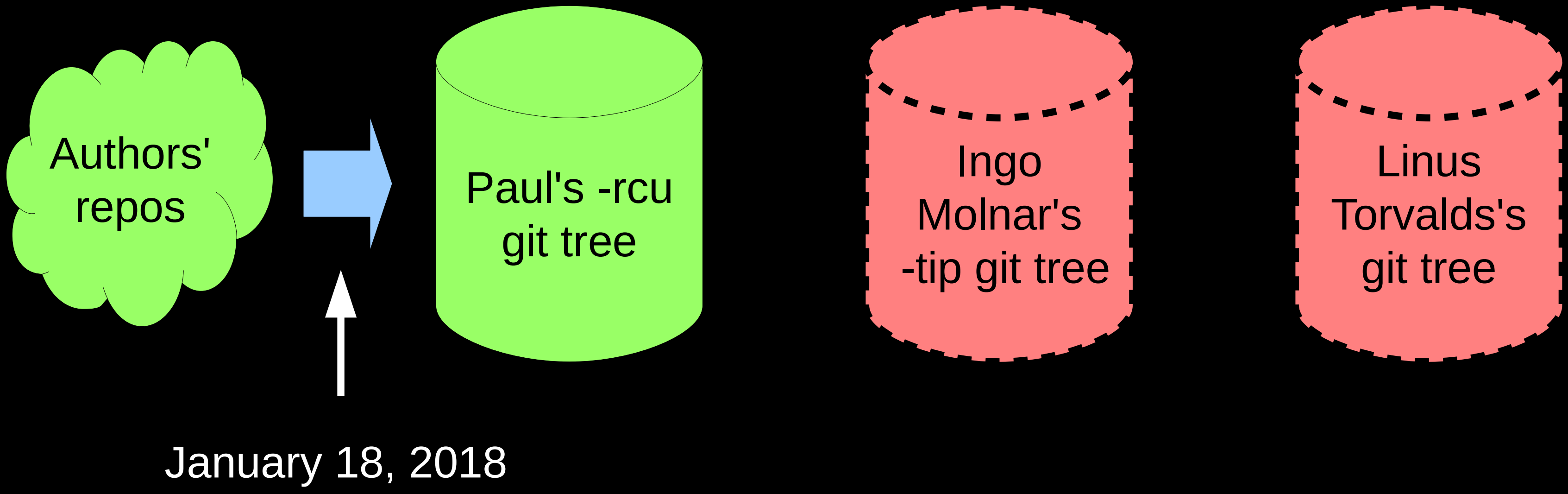
Future Prospects

- Moving target as hardware, workloads, & code style changes
  - RISC-V interaction, fill out locking primitives, plain accesses, ...
- Design/education tool, hoped-for use by verification tools
- If this is so great, why isn't it in the Linux kernel???



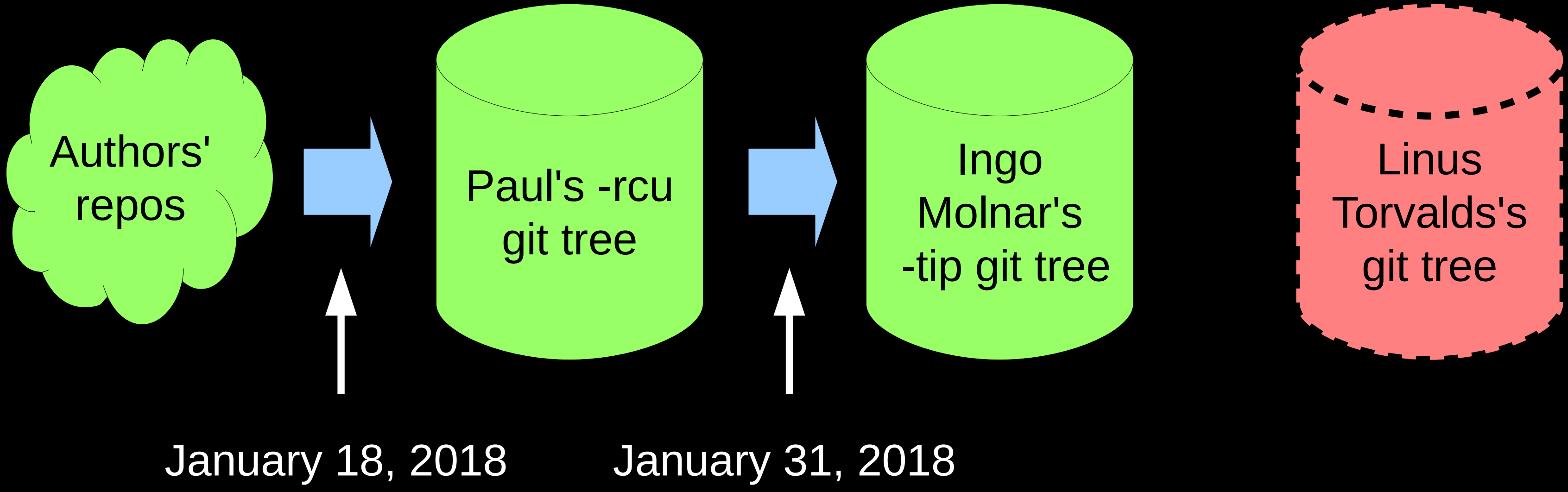
Future Prospects

- Moving target as hardware, workloads, & code style changes
  - RISC-V interaction, fill out locking primitives, plain accesses, ...
- Design/education tool, hoped-for use by verification tools
- If this is so great, why isn't it in the Linux kernel???



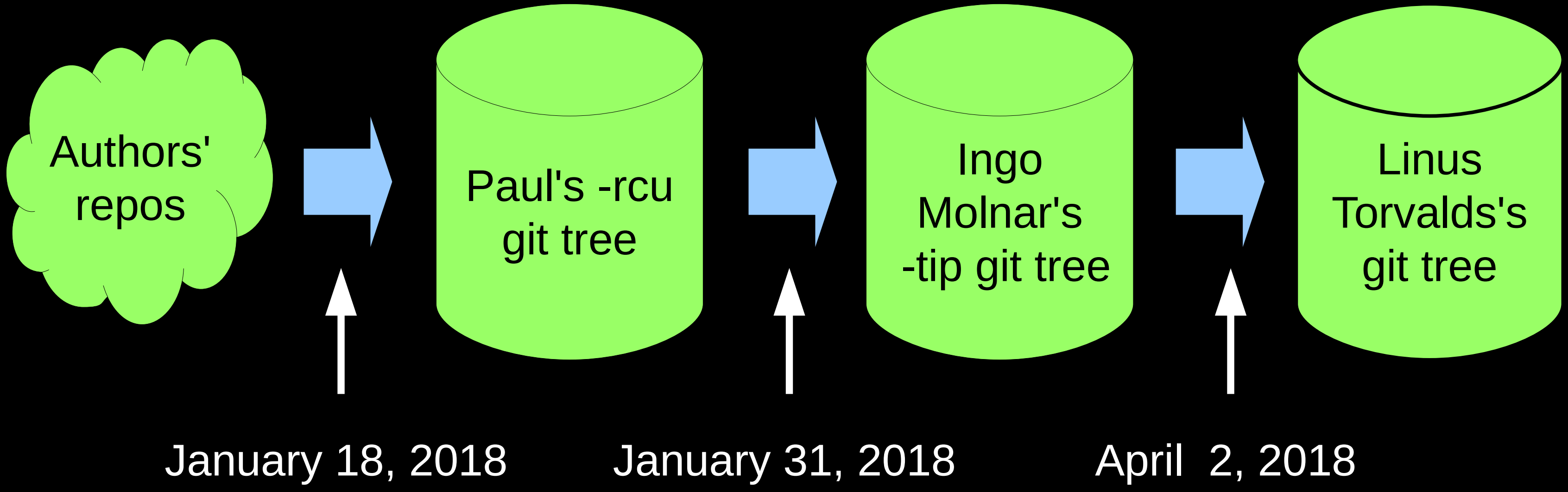
Future Prospects

- Moving target as hardware, workloads, & code style changes
  - RISC-V interaction, fill out locking primitives, plain accesses, ...
- Design/education tool, hoped-for use by verification tools
- If this is so great, why isn't it in the Linux kernel???



Future Prospects

- Moving target as hardware, workloads, & code style changes
  - RISC-V interaction, fill out locking primitives, plain accesses, …
- Design/education tool, hoped-for use by verification tools
- If this is so great, why isn't it in the Linux kernel???



Model Freely Available For Download and Use

- Download herd tool as part of diy toolset
  - http://diy.inria.fr/sources/index.html
- Build as described in INSTALL.txt
  - Need ocaml v4.02.0 or better: http://caml.inria.fr/download.en.html
    - Or install from your distro (easier and faster!)
- Run various litmus tests:
  - herd7 -conf linux-kernel.cfg litmus-tests/MP+polocks.litmus
  - herd7 -conf linux-kernel.cfg litmus-tests/R+poonceonces.litmus
  - herd7 -conf linux-kernel.cfg litmus-tests/R+poonceonces.litmus
- Other required files:
  - linux-kernel.def: Support pseudo-C code
  - linux-kernel.cfg: Specify kernel model
  - linux-kernel.bell: “Bell” file defining events and their relationships
  - linux-kernel.cat: “Cat” file defining actual memory model
  - litmus-tests/*.litmus: Litmus tests

git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git in tools/memory-model

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of University College London, Microsoft Research, Inria-Paris, IBM, Oregon State University, Scuola Superiore Sant'Anna, or Harvard University
- 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?

References (1/11)

- Alglave, J., Kroening, D., and Tautschnig, M. (2013).
  - Partial orders for efficient Bounded Model Checking of concurrent software.
  - In Computer Aided Verification (CAV), volume 8044 of LNCS, pages 141–157. Springer.
- Blanchard, A. (2011).
  - RE: [PATCH] smp call function many SMP race.
  - https://lkml.org/lkml/2011/1/11/489.
- Corbet, J. (2008).
  - The lockless page cache.
  - https://lwn.net/Articles/291826/.
- Corbet, J. (2012).
  - ACCESS ONCE().
  - https://lwn.net/Articles/508991/.
- Corbet, J. (2014a).
  - ACCESS ONCE() and compiler bugs.
  - https://lwn.net/Articles/624126/.

References (2/11)

- Corbet, J. (2014b).
  - C11 atomic variables and the kernel.
  - https://lwn.net/Articles/586838/.
- Corbet, J. (2014c).
  - C11 atomics part 2: “consume” semantics.
  - https://lwn.net/Articles/588300/.
- Corbet, J. (2016).
  - Time to move to C11 atomics?
  - https://lwn.net/Articles/691128/.
- Deacon, W. (2015a).
  - [PATCH] arm64: spinlock: serialise spin unlock wait against concurrent lockers.
  - https://marc.info/?l=linux-arm-kernel&m=144862480822027.
- Deacon, W. (2015b).
  - Re: [PATCH] arm64: spinlock: serialise spin unlock wait against concurrent lockers.
  - https://marc.info/?l=linux-arm-kernel&m=144898777124295.

References (3/11)

- Deacon, W. (2016).
  - [PATCH v2 1/3] arm64: spinlock: order spin {is locked, unlock wait} against local locks.
  - http://lists.infradead.org/pipermail/linux-arm-kernel/2016-June/434765.html.
- Desnoyers, M., McKenney, P. E., Stern, A. S., Dagenais, M. R., and Walpole, J. (2012).
  - User-level implementations of Read-Copy Update.
  - IEEE Trans. Parallel Distrib. Syst., 23(2):375–382.
- Ellerman, M. (2016).
  - [PATCH v3] powerpc: spinlock: Fix spin unlock wait().
  - https://marc.info/?l=linux-kernel&m=146521336230748&w=2.
- Feng, B. (2015).
  - Re: [PATCH 4/4] locking: Introduce smp cond acquire().
  - https://marc.info/?l=linux-kernel&m=144723482232285.

References (4/11)

- Feng, B. (2016a).
  - [PATCH v2] powerpc: spinlock: Fix spin unlock wait().
  - https://marc.info/?l=linux-kernel&m=146492558531292&w=2.
- Feng, B. (2016b).
  - [PATCH v4] powerpc: spinlock: Fix spin unlock wait().
  - https://marc.info/?l=linuxppc-embedded&m=146553051027169&w=2.
- Gorman, M. (2013).
  - LWN Quotes of the week, 2013-12-11.
  - http://lwn.net/Articles/575835/.
- Heo, T. (2010).
  - [PATCH 3/4] scheduler: replace migration thread with cpuhog.
  - https://marc.info/?l=linux-kernel&m=126806371630498.

References (5/11)

- Howells, D., McKenney, P. E., Deacon, W., and Zijlstra, P. (2017).
  - Linux kernel memory barriers.
  - https://www.kernel.org/doc/Documentation/memory-barriers.txt.
- Kleen, A. (2013).
  - Re: [patch v6 4/5] MCS lock: Barrier corrections.
  - https://marc.info/?l=linux-mm&m=138619237606428.
- Luck, T. (2013a).
  - RE: Does Itanium permit speculative stores?
  - https://marc.info/?l=linux-kernel&m=138427925823852.
- Luck, T. (2013b).
  - RE: Does Itanium permit speculative stores?
  - https://marc.info/?l=linux-kernel&m=138428203211477.

References (6/11)

- McKenney, P. E. (2007).
  - QRCU with lockless fastpath.
  - https://lwn.net/Articles/223752/.
- McKenney, P. E. (2013b).
  - Does Itanium permit speculative stores?
  - https://marc.info/?l=linux-kernel&m=138419150923282.
- McKenney, P. E. (2016b).
  - documentation: Present updated RCU guarantee.
  - https://patchwork.kernel.org/patch/9428001/.
- McKenney, P. E. (2016c).
  - documentation: Transitivity is not cumulativity.
  - http://www.spinics.net/lists/linux-tip-commits/msg32905.html.
- McKenney, P. E. (2017b).
  - srcu: Force full grace-period ordering.
  - https://patchwork.kernel.org/patch/9535987/

References (7/11)

- McKenney, P. E., Desnoyers, M., Jiangshan, L., and Triplett, J. (2013).
  - The RCU-barrier menagerie.
  - https://lwn.net/Articles/573497/.
- Miller, D. S. (2017).
  - Semantics and behavior of atomic and bitmask operations.
  - https://www.kernel.org/doc/core-api/atomic_ops.rst.
- Miller, M. (2011).
  - [PATCH 0/4 v3] smp call function many issues from review.
  - https://marc.info/?l=linux-kernel&m=130021726530804.
- Molnar, I. (2013).
  - Re: [patch v6 4/5] MCS lock: Barrier corrections.
  - https://marc.info/?l=linux-mm&m=138513336717990&w=2.

References (8/11)

- Molnar, I. (2017).
  - Re: [PATCH v2 0/9] remove spin unlock wait().
  - https://marc.info/?l=linux-kernel&m=149942365927828&w=2.
- Olsa, J. (2009).
  - [PATCHv5 2/2] memory barrier: adding smp mb after lock.
  - https://marc.info/?l=linux-netdev&m=124839648220382&w=2.
- Spraul, M. (2001).
  - Re: RFC: patch to allow lock-free traversal of lists with insertion.
  - http://lkml.iu.edu/hypermail/linux/kernel/0110.1/0410.html.
- Torvalds, L. (2012).
  - Re: Memory corruption due to word sharing.
  - https://gcc.gnu.org/ml/gcc/2012-02/msg00013.html
- .Torvalds, L. (2015).
  - Re: [patch 4/4] locking: Introduce smp cond acquire().
  - http://lkml.kernel.org/r/CA+55aFyXu5iFJfdm7o-RKUm_9a850iSzeM+whmtUAotkY0EvTw@mail.gmail.com.

References (9/11)

- Torvalds, L. (2016a).
  - Re: [rfc][patch] mips: Fix arch spin unlock().
  - https://lkml.org/lkml/2016/2/2/80.
- Torvalds, L. (2016b).
  - Re: [v3,11/41] mips: reuse asm-generic/barrier.h.
  - https://marc.info/?l=linux-kernel&m=145384764324700&w=2.
- Torvalds, L. (2017).
  - Re: [GIT PULL rcu/next] RCU commits for 4.13.
  - https://lkml.org/lkml/2017/6/27/1052.
- Vaddagiri, S. (2005).
  - [PATCH] Fix RCU race in access of nohz cpu mask.
  - http://lkml.iu.edu/hypermail/linux/kernel/0512.0/0976.html.

References (10/11)

- Yegoshin, L. (2016a).
  - Re: [v3,11/41] mips: reuse asm-generic/barrier.h.
  - https://marc.info/?l=linux-kernel&m=145263153305591&w=2.
- Yegoshin, L. (2016b).
  - Re: [v3,11/41] mips: reuse asm-generic/barrier.h.
  - https://marc.info/?l=linux-kernel&m=145280444229608&w=2.
- Yegoshin, L. (2016c).
  - Re: [v3,11/41] mips: reuse asm-generic/barrier.h.
  - https://marc.info/?l=linux-kernel&m=145280241129008&w=2.
- Zijlstra, P. (2013).
  - Re: Does Itanium permit speculative stores?
  - https://marc.info/?l=linux-kernel&m=138428080207125.

References (11/11)

- Zijlstra, P. (2016).
  - [tip:perf/urgent] perf/core: Fix sys perf event open() vs. hotplug.
  - https://www.spinics.net/lists/kernel/msg2421883.html.
- Ziljstra, P. (2013).
  - Re: [patch v6 4/5] MCS lock: Barrier corrections.
  - http://marc.info/?l=linux-mm&m=138514629508662&w=2.

Backup Slides

Overview

- What *have* we done? And why???
- Why not simply use the C++11 memory model?
- Why the Linux-kernel memory model (LKMM)?
- Synopsis of work, capabilities and limitations
- How does the memory model keep up with Linux kernel?
- Why two definitions of RCU?
- Model availability and use
- Future prospects: Acceptance into Linux kernel?

What *Have* We Done?
And Why???

First, Why???

- Linux kernel (LK) supports more than 20 CPU architectures
  - DEC Alpha, ARM, ARM64, IA64, MIPS, OpenRISC, PowerPC, RISC-V, S390, SPARC, x86, and 10+ more
- Concurrent LK code must run correctly everywhere!!!
- Hence, “Interesting” memory-ordering discussions on the Linux kernel mailing list (Section 1 of paper)
  - How is core Linux-kernel code supposed to behave?
  - How must Linux-kernel synchronization primitives be implemented?
  - Does it behave correctly on exotic hardware?
  - *What exactly can a Linux-kernel hacker get away with?*
    - And speaking as its main author, I am here to tell you that Documentation/memory-barriers.txt passed its sell-by date years ago...

So What Did We Do About All That???

Next, What???

- First formal memory model for the Linux kernel
  - Executable cat code, already used by Linux kernel hackers: Motivated removal of spin_unlock_wait() and DEC Alpha rework
    - Sections 2 and 3 of paper
- First memory model of any kind that includes RCU
  - Formulated fundamental law of RCU as well as the RCU axiom
    - Showed them to be equivalent (Section 4 of paper)
    - Verification tools can therefore use either fundamental law or RCU axiom
  - Showed that the userspace RCU library satisfies the fundamental law
    - (Section 6 of the paper)

Why Not Simply Use the C++11 Memory Model?

- The C++11 memory model lacks:
  - Memory fences that can restore sequential consistency (SC)
  - RMW atomic operations that can restore SC
  - Control, address, and data dependencies (memory_order_consume!)
    - LKMM therefore avoids out-of-thin-air (OOTA) accesses
  - Atomic operations on non-atomic variables (maybe atomic_ref)
  - Read-copy update (RCU)
    - Working on this... And only since 2014!
- David Howells tried porting Linux-kernel x86 to C11 atomics
  - Resulted in some success, but numerous bug reports
  - Maybe someday, but not there yet

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/p0124r4.html

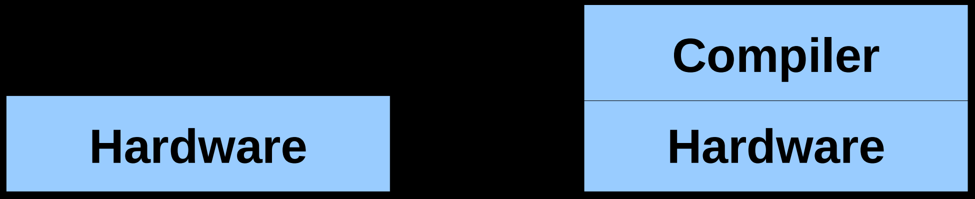
Why A Memory Model At All?

- Linux kernel supports more than 20 CPU architectures
  - DEC Alpha, ARM, ARM64, IA64, MIPS, OpenRISC, PowerPC, RISC-V, S390, SPARC, x86, and 10+ more
- Linux core kernel code must run correctly on all of them!!!
- Hence, “Interesting” memory-ordering discussions on the Linux kernel mailing list (Section 1 of paper)
  - How is core Linux-kernel code supposed to behave?
  - How must Linux-kernel synchronization primitives be implemented?
  - Does it behave correctly on exotic hardware?
  - *What exactly can a Linux-kernel hacker get away with?*
    - And Documentation/memory-barriers.txt has passed its sell-by date...

Synopsis of Work

- First formal memory model for the Linux kernel
  - Executable cat code, already used by Linux kernel hackers: Motivated removal of spin_unlock_wait() and DEC Alpha rework
    - Sections 2 and 3 of paper
  - Tested on Power 8, ARMv8, ARMv7, and x86 hardware
    - Section 5 of paper plus supplementary materials (http://diy.inria.fr/linux/)
  - Avoids out-of-thin-air results! (For now, anyway...)
- First memory model of any kind that includes RCU
  - Formulated fundamental law of RCU as well as the RCU axiom
    - Showed them to be equivalent (Section 4 of paper)
    - Verification tools can therefore use either fundamental law or RCU axiom
  - Showed that the userspace RCU library satisfies the fundamental law
    - (Section 6 of the paper)

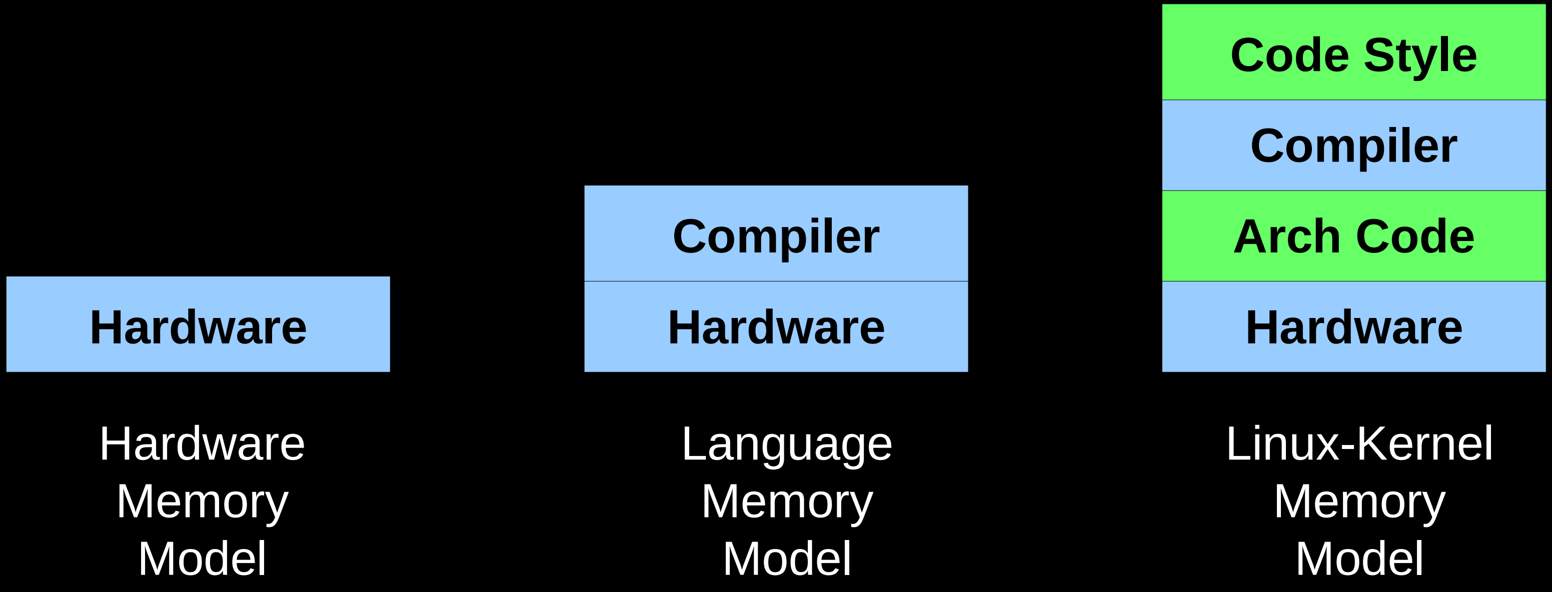
Progression of Memory Models



Hardware Memory Model

Language Memory Model

Progression of Memory Models



Current Model Capabilities ...

- READ_ONCE() and WRITE_ONCE()
- smp_store_release() and smp_load_acquire()
- rcu_assign_pointer() and rcu_dereference()
- rcu_read_lock(), rcu_read_unlock(), and synchronize_rcu()
  - Also synchronize_rcu_expedited(), but same as synchronize_rcu()
- smp_mb(), smp_rmb(), smp_wmb(), smp_mb__after spinlock(), and more
- Most atomic read-modify-write operations
- spin_lock(), spin_unlock(), and spin_trylock()

… And Limitations

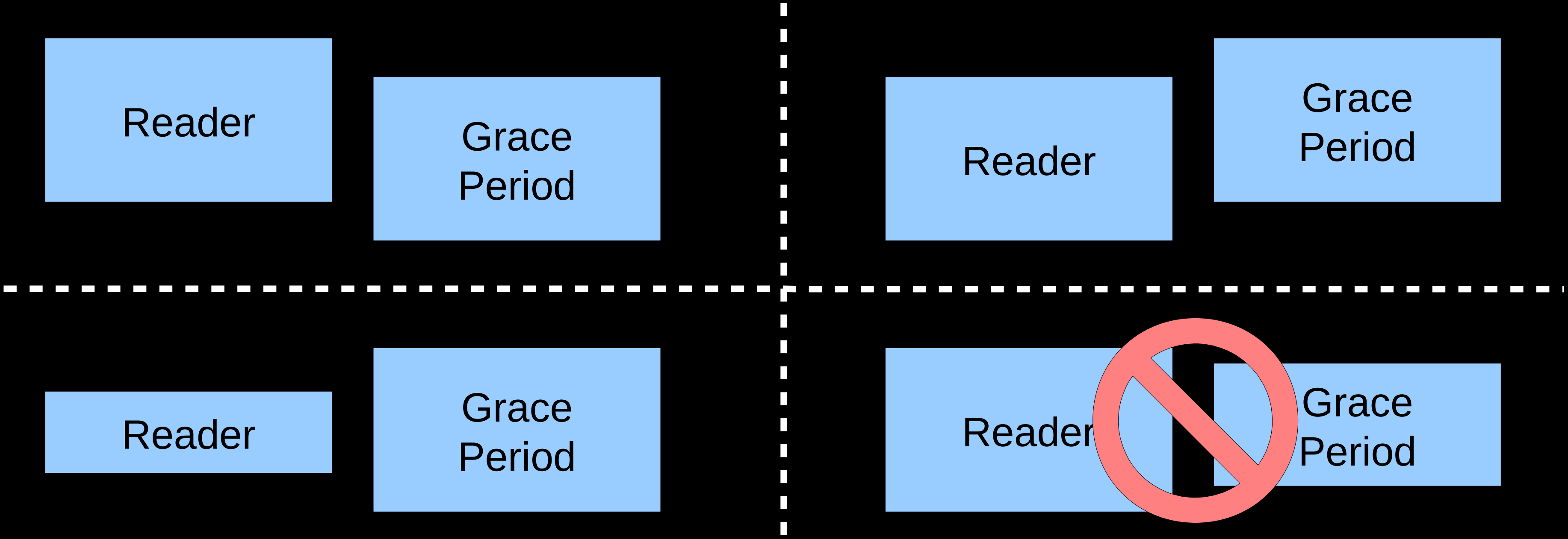
- There are some limitations in the model:
  - Compiler optimizations not modeled
  - Locking is missing spin_is_locked(), which may require changes to the underlying “herd” tool
  - No asynchronous RCU grace periods, emulate using a separate thread with release-acquire, grace period, and then callback code
  - Single access size, no partially overlapping accesses, which may require changes to the underlying “herd” tool
- And other limitations in the underlying “herd” tool:
  - No arrays or structs (but can do trivial linked lists)
  - No dynamic memory allocation
  - No interrupts, exceptions, I/O, or self-modifying code
  - No functions

How Does LKMM Keep Up With Linux Kernel?

- LKMM will require continued development and maintenance:
  - New CPU architectures will be added (most recently, RISC-V)
  - New synchronization primitives will be added
  - Old synchronization primitives will be removed (spin_unlock_wait())
  - New use cases will arise
- We expect LKMM rate of change to be similar to that of `Documentation/memory-barriers.txt`
  - Every few months, but sometimes in bursts
- Ten people signed up as maintainers, many having long experience with the Linux kernel and/or memory models

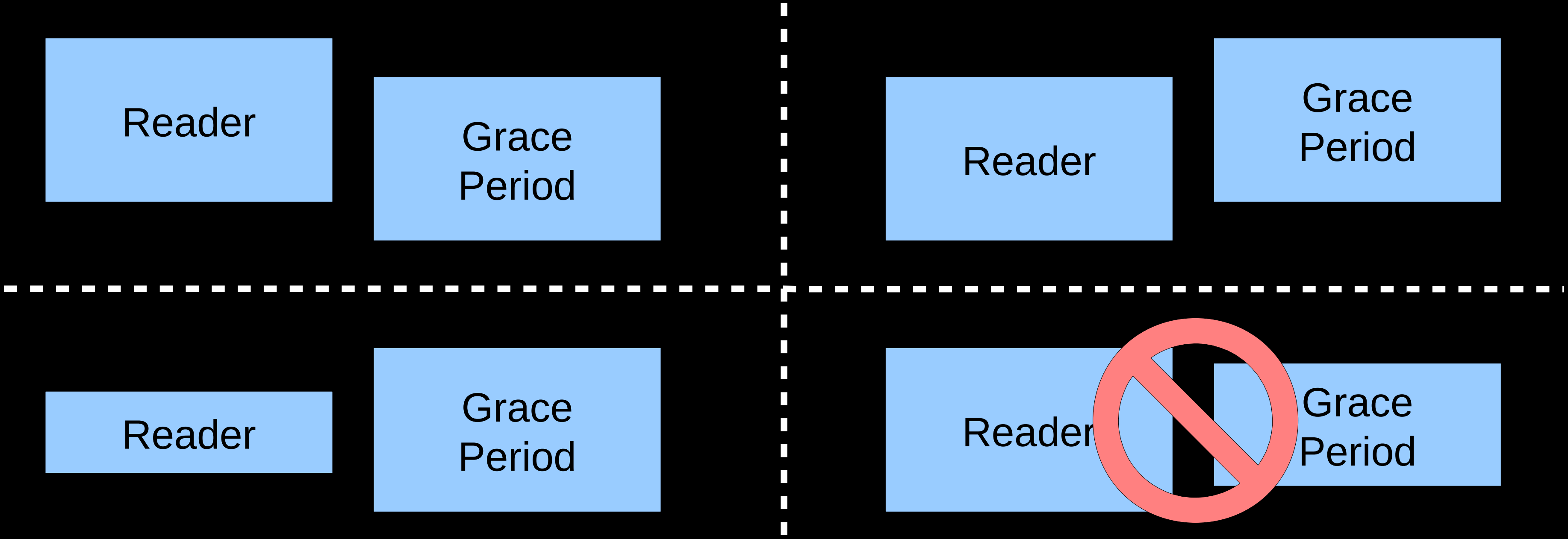
Why Two Definitions of RCU? (1) Fundamental Law

- Fundamental Law of RCU from Linux kernel and earlier:
  - RCU read-side critical section begins with rcu_read_lock() and ends with rcu_read_unlock()
  - Grace period waits for completion of all pre-existing critical sections



Why Two Definitions of RCU? (1) Fundamental Law

- Fundamental Law of RCU from Linux kernel and earlier:
  - RCU read-side critical section begins with rcu_read_lock() and ends with rcu_read_unlock()
  - Grace period waits for completion of all pre-existing critical sections



Great for developers and operational models, not so good for axiomatic models

Why Two Definitions of RCU? (2) RCU Axiom

- LKMM axiomatic model defined in terms of cycles
  - Must define behavior when all ordering in cycle is provided by RCU

Process 0:

```
rcu_read_lock();
WRITE_ONCE(x, 1);
r0 = READ_ONCE(y);
rcu_read_unlock();
```

Process 1:

```
rcu_read_lock();
WRITE_ONCE(y, 1);
r1 = READ_ONCE(z);
rcu_read_unlock();
```

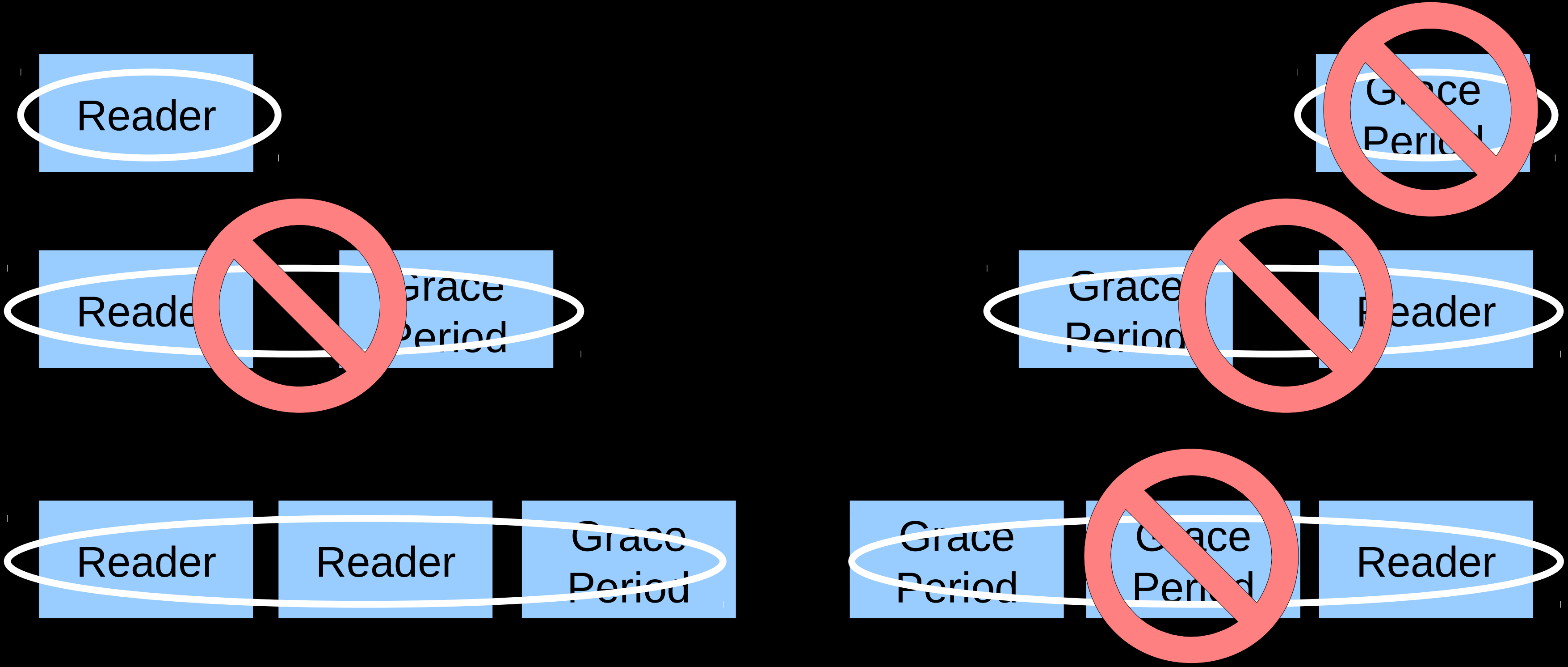
Process 2:

```
WRITE_ONCE(z, 1);
synchronize_rcu();
r2 = READ_ONCE(x);
```

r0 == 0 && r1 == 0 && r2 == 0?

Why Two Definitions of RCU? (2) RCU Axiom

- LKMM axiomatic model defined in terms of cycles
  - Must define behavior when all ordering in cycle is provided by RCU
  - Cycle forbidden if at least as many grace periods as critical sections



Why Two Definitions of RCU? (2) RCU Axiom

- LKMM axiomatic model defined in terms of cycles
  - Must define behavior when all ordering in cycle is provided by RCU
  - Cycle forbidden if at least as many grace periods as critical sections

Process 0:

```
rcu_read_lock();
WRITE_ONCE(x, 1);
r0 = READ_ONCE(y);
rcu_read_unlock();
```

Process 1:

```
rcu_read_lock();
WRITE_ONCE(y, 1);
r1 = READ_ONCE(z);
rcu_read_unlock();
```

Process 2:

```
WRITE_ONCE(z, 1);
synchronize_rcu();
r2 = READ_ONCE(x);
```

r0 == 0 && r1 == 0 && r2 == 0?

Yes, this can happen, two readers, only one grace period.

Why Two Definitions of RCU?

- We have proven the two definitions equivalent for the Linux-kernel memory model
- A given tool can therefore use whichever of the two definitions that works best for that tool

Have Linux Kernel Hackers Made Use of LKMM?

- 2016 Linux-kernel git commit contains an LKMM litmus test
  - https://www.spinics.net/lists/kernel/msg2421883.html
- LKMM has been used in discussion of RISC-V ordering
- It has also motivated:
  - Removal of spin_unlock_wait()
  - Greatly reducing use of smp_read_barrier_depends()
  - Helping to formulate RCU ordering requirements in C++ Standards Committee working paper
- Numerous Linux kernel hackers have installed LKMM

Future Prospects

- Moving target as hardware, workloads, & code style changes
  - RISC-V interaction, fill out locking primitives, plain accesses, …
- Design/education tool, hoped-for use by verification tools

Future Prospects

- Moving target as hardware, workloads, & code style changes
  - RISC-V interaction, fill out locking primitives, plain accesses, …
- Design/education tool, hoped-for use by verification tools
- If this is so great, why isn't it in the Linux kernel???