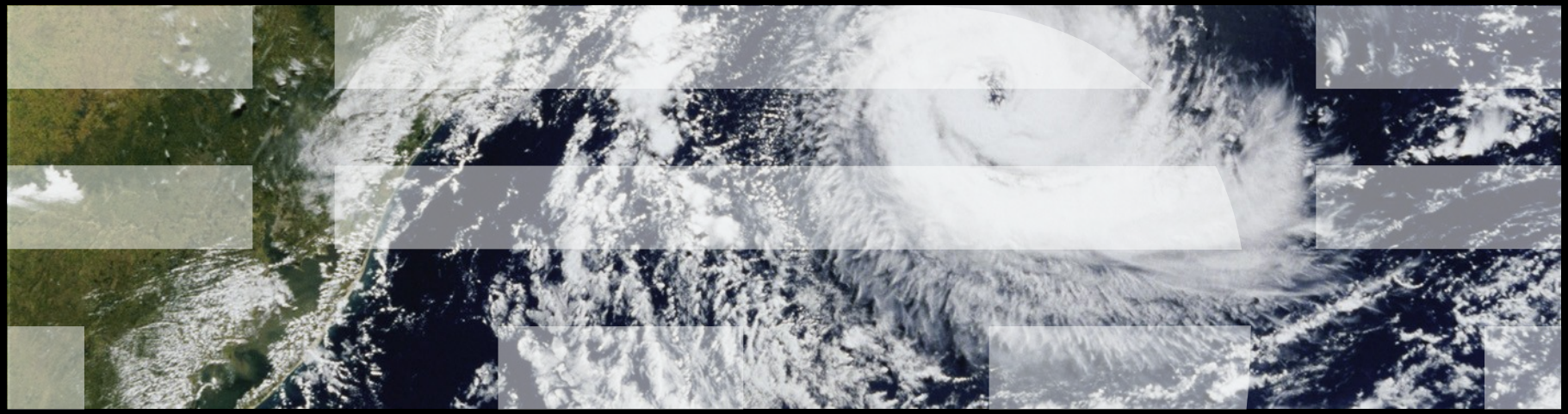
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center
Member, IBM Academy of Technology
PSU CS410/CS510 Advanced Topics in Concurrency, February 20, 2018

Linux-Kernel Memory Ordering: Help Arrives At Last!

Joint work with Jade Alglave, Luc Maranget, Andrea Parri, and Alan Stern



Overview

- Who cares about memory models?
- But memory-barrier.txt is incomplete!
- Project history
- Cat-language example: single-variable SC
- Current status and demo
- Not all communications relations are created equal
- Rough rules of thumb

Who Cares About Memory Models?

Example “Litmus Test”: Can This Happen?

Thread 0:

```
WRITE_ONCE(*x0, 1);
r1 = READ_ONCE(x1);
```

Thread 1:

```
WRITE_ONCE(*x1, 1);
r1 = READ_ONCE(x2);
```

Thread 2:

```
WRITE_ONCE(*x2, 1);
r1 = READ_ONCE(x0);
```

“Exists” Clause

```
(0:r1=0 /\ 1:r1=0 /\ 2:r1=0)
```

litmus/manual/extra/sb+o-o+o-o.litmus
(from https://github.com/paulmckrcu/litmus)

Example “Litmus Test”: All CPUs Can Reorder Earlier Writes With Later Reads of Different Variables, So ...

Thread 0:

```
WRITE_ONCE(*x0, 1);
r1 = READ_ONCE(x1);
```

Thread 1:

```
WRITE_ONCE(*x1, 1);
r1 = READ_ONCE(x2);
```

Thread 2:

```
WRITE_ONCE(*x2, 1);
r1 = READ_ONCE(x0);
```

“Exists” Clause

```
(0:r1=0 /\ 1:r1=0 /\ 2:r1=0)
```

litmus/manual/extra/sb+o-o+o-o.litmus
(from https://github.com/paulmckrcu/litmus)

Example “Litmus Test”: … This Can Happen!!!

Thread 0:

```
r1 = READ_ONCE(x1);
WRITE_ONCE(*x0, 1);
```

Thread 1:

```
r1 = READ_ONCE(x2);
WRITE_ONCE(*x1, 1);
```

Thread 2:

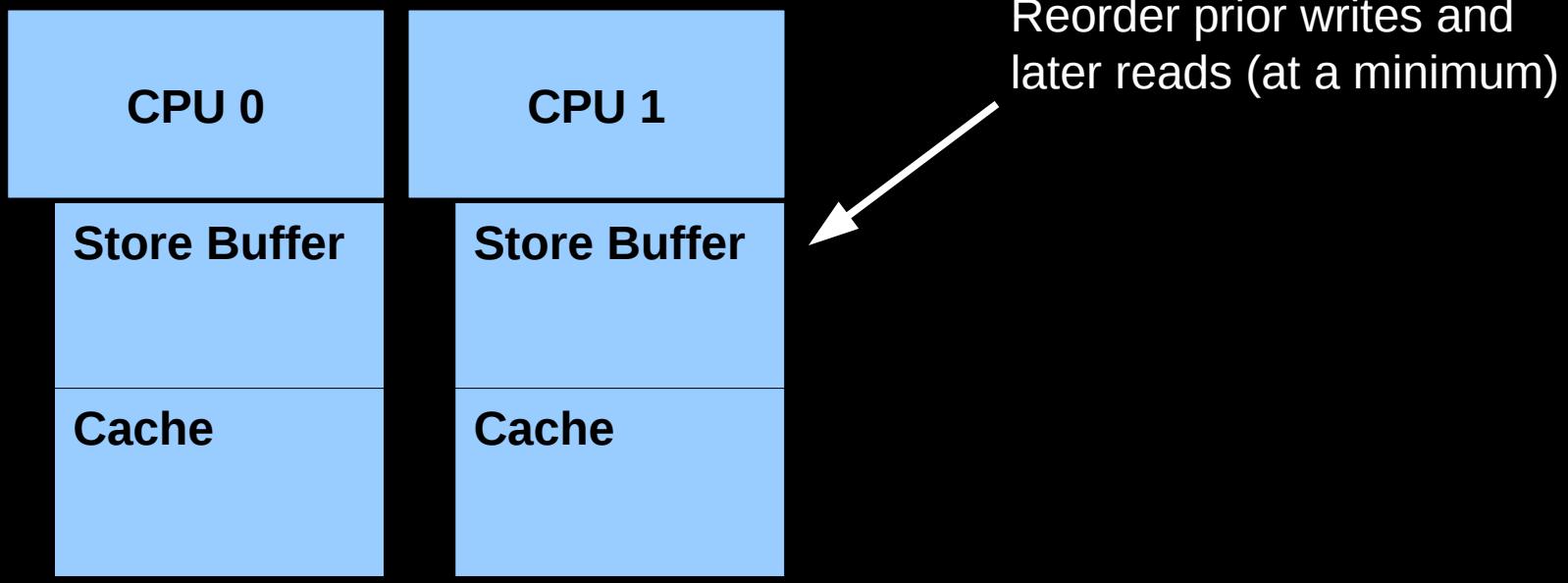
```
r1 = READ_ONCE(x0);
WRITE_ONCE(*x2, 1);
```

“Exists” Clause

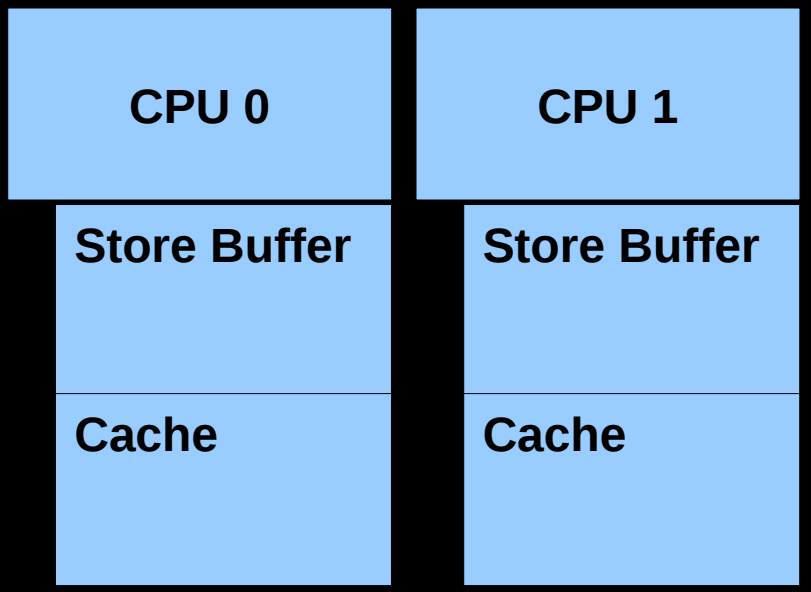
```
(0:r1=0 /\ 1:r1=0 /\ 2:r1=0)
```

litmus/manual/extra/sb+o-o+o-o.litmus
(from https://github.com/paulmckrcu/litmus)

This Differs From Sequential Consistency Due To Hardware and Compiler Performance Optimizations



This Differs From Sequential Consistency Due To Hardware and Compiler Performance Optimizations



Reorder prior writes and later reads (at a minimum)

Compiler optimizations result in more profound reorderings

Viktor Vafeiadis, “Sequential consistency considered harmful”
http://materials.dagstuhl.de/files/17/17451/17451.ViktorVafeiadis.Slides.pdf

Another Example “Litmus Test”: Can This Happen?

Thread 0:

```
WRITE_ONCE(*u0, 3);
smp_store_release(x1, 1);
```

Thread 1:

```
r1 = smp_load_acquire(x1);
r2 = READ_ONCE(*v0);
```

Thread 2:

```
WRITE_ONCE(*v0, 1);
smp_mb();
r2 = READ_ONCE(*u0);
```

“Exists” Clause

```
(1:r2=0 /\ 2:r2=0 /\ 1:r1=1)
```

litmus/auto/C-LB-GWR+R-A.litmus

Who Cares About Memory Models, and If So, Why???

- Hoped-for benefits of a Linux-kernel memory model
  - Memory-ordering education tool
  - Core-concurrent-code design aid
  - Ease porting to new hardware and new toolchains
  - Basis for additional concurrency code-analysis tooling
    - For example, CBMC and Nidhugg (CBMC now part of rcutorture)
- Likely drawbacks of a Linux-kernel memory model
  - Extremely limited size: Handful of processes with handful of code
    - Analyze concurrency core of algorithm
    - Maybe someday automatically identifying this core
    - Perhaps even automatically stitch together multiple analyses (dream on!)
  - Limited types of operations (no function call, structures, call_rcu(), …)
    - Can emulate some of these
    - We expect that tools will become more capable over time
    - (More on this on a later slide)

But memory-barrier.txt is Incomplete!

But memory-barrier.txt is Incomplete!

- The memory-barriers.txt file defined the kernel's memory model
  - Albeit quite incompletely
- The Linux kernel has left many corner cases unexplored
  - David, Peter, Will, and I added cases as requested: Organic growth
  - The Linux-kernel memory model must provide a complete definition
- Guiding principles:
  - Strength preferred to weakness
  - Simplicity preferred to complexity
  - Support existing non-buggy Linux-kernel code (later slide)
  - Be compatible with hardware supported by the Linux kernel (later slide)
  - Support future hardware, within reason
  - Be compatible with C11, where prudent and reasonable (later slide)
  - Expose questions and areas of uncertainty (later slide)
    - Which means not one but two memory models!

Support Existing Non-Buggy Linux-Kernel Code

- But there are some limitations in the model:
  - Compiler optimizations not modeled
  - Locking is missing spin_is_locked()
    - But likely need additional support in the underlying “herd” tool
  - No asynchronous RCU grace periods, but can emulate them:
    - Separate thread with release-acquire, grace period, and then callback code
  - Single access size, no partially overlapping accesses
    - But likely need additional support in 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
- Something about wanting the model to execute in finite time...

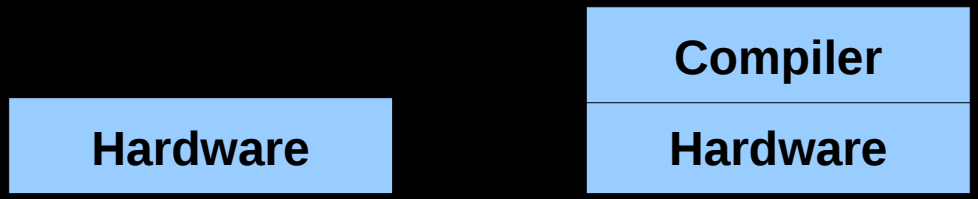
Be Compatible With HW Supported by Linux Kernel

- Model must be in some sense a least common denominator:
  - If a given system allows some behavior, the model must also do so
  - Note that the model can allow behavior forbidden by systems
- However, compiler & kernel code can mask HW weaknesses:
  - Alpha has memory barrier for smp_read_barrier_depends()
    - Now included in READ_ONCE() to avoid Alpha-specific core code
  - Itanium gcc emits ld.acq and st.rel for volatile loads and stores
- Key problem: How to know what does hardware do?
  - Check existing documentation
  - Consult HW architects, where available and responsive
  - Formal memory models, where available
  - Run experiments on real hardware

Be Compatible With HW Supported by Linux Kernel

- Model must be in some sense a least common denominator:
  - If a given system allows some behavior, the model must also do so
  - Note that the model can allow behavior forbidden by systems
- However, compiler & kernel code can mask HW weaknesses:
  - Alpha has memory barrier for smp_read_barrier_depends()
    - Now included in READ_ONCE() to avoid Alpha-specific core code
  - Itanium gcc emits ld.acq and st.rel for volatile loads and stores
- Key problem: How to know what does hardware do?
  - Check existing documentation
  - Consult HW architects, where available and responsive
  - Formal memory models, where available
  - Run experiments on real hardware
  - In the end, make our best guess!!! Expect changes over time...

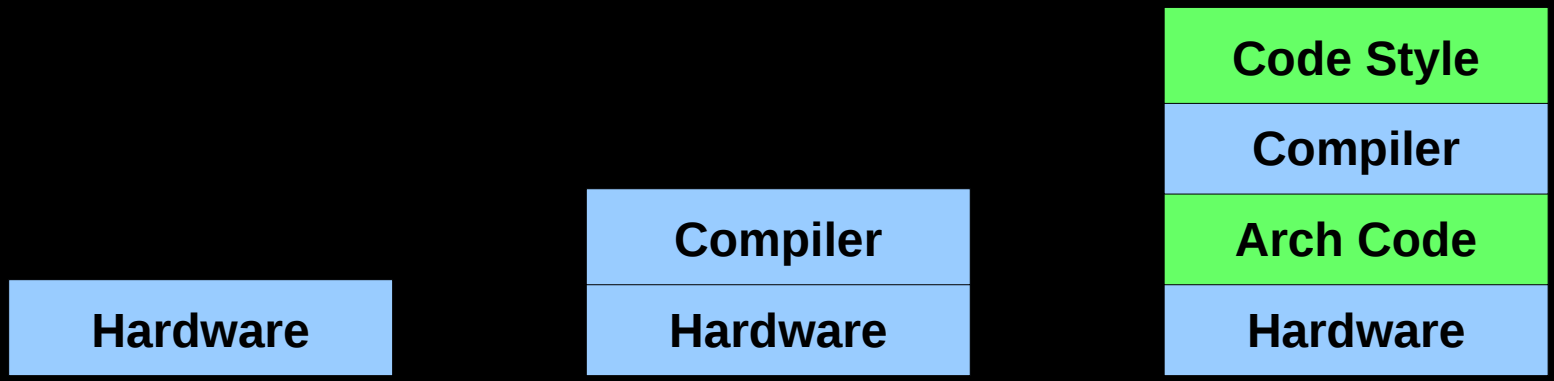
Progression of Memory Models



Hardware Memory Model

Language Memory Model

Progression of Memory Models



Be Compatible With C11, Where Reasonable

- smp_mb() stronger than C11 counterpart
- C11 lacks smp_rmb() and smp_wmb()
- Linux-kernel RMW atomics stronger than C11
- C11 doesn't have barrier-amplification primitives
  - smp_mb__before_atomic() and friends
- C11 doesn't have ubiquitous dependencies
  - Address, control, and data dependencies
  - But these dependencies help eliminate out-of-thin-air results!
- C11 doesn't have RCU grace periods
  - Though a proposal has been solicited and is in progress
- By default, support the Linux kernel's ordering needs

Project Pre-History

Project Prehistory

- 2005-present: C and C++ memory models
  - Working Draft, Standard for Programming Language C++
- 2009-present: x86, Power, and ARM memory models
  - http://www.cl.cam.ac.uk/~pes20/weakmemory/index.html
- 2014: Clear need for Linux-kernel memory model, but...
  - Legacy code, including unmarked shared accesses
  - Wide range of SMP systems, with varying degrees of documentation
  - High rate of change: Moving target!!!
- As a result, no takers

Project Prehistory

- 2005-present: C and C++ memory models
  - Working Draft, Standard for Programming Language C++
- 2009-present: x86, Power, and ARM memory models
  - http://www.cl.cam.ac.uk/~pes20/weakmemory/index.html
- 2014: Clear need for Linux-kernel memory model, but...
  - Legacy code, including unmarked shared accesses
  - Wide range of SMP systems, with varying degrees of documentation
  - High rate of change: Moving target!!!
- As a result, no takers
- Until early 2015

Our Founder

Our Founder

Jade Alglave, University College London and Microsoft Research

Founder's First Act: Adjust Requirements

- Strategy is what you are ***not*** going to do!

Founder's First Act: Adjust Requirements

- Strategy is what you are ***not*** going to do!
- New Requirements:
  - ~~Legacy code, including unmarked shared accesses~~
  - Wide range of SMP systems, with varying degrees of documentation
  - High rate of change: Moving target!!!

Founder's First Act: Adjust Requirements

- Strategy is what you are ***not*** going to do!
- New Requirements:
  - ~~Legacy code, including unmarked shared accesses~~
  - Wide range of SMP systems, with varying degrees of documentation
  - High rate of change: Moving target!!!
- Adjustment advantage: Solution now feasible!
  - No longer need to model all possible compiler optimizations...
  - Optimizations not yet envisioned being the most difficult to model!!!
  - Jade expressed the model in the “cat” language
    - The “herd” tool uses the “cat” language to process concurrent code fragments, called “litmus tests” (example next slides)
    - Initially used a generic language called “LISA”, now C-like language
    - (See next few slides for a trivial example..)

Founder's Second Act: Create Prototype Model

- And to recruit a guy named Paul E. McKenney (Apr 2015):
  - Clarifications of less-than-rigorous memory-barriers.txt wording
  - Full RCU semantics: Easy one! 2+ decades RCU experience!!! Plus:
    - Jade has some RCU knowledge courtesy of ISO SC22 WG21 (C++)
    - “User-Level Implementations of Read-Copy Update”, 2012 IEEE TPDS
    - “Verifying Highly Concurrent Algorithms with Grace”, 2013 ESOP

Founder's Second Act: Create Prototype Model

- And to recruit a guy named Paul E. McKenney (Apr 2015):
  - Clarifications of less-than-rigorous memory-barriers.txt wording
  - Full RCU semantics: Easy one! 2+ decades RCU experience!!! Plus:
    - Jade has some RCU knowledge courtesy of ISO SC22 WG21 (C++)
    - “User-Level Implementations of Read-Copy Update”, 2012 IEEE TPDS
    - “Verifying Highly Concurrent Algorithms with Grace”, 2013 ESOP
- Initial overconfidence meets Jade Alglave memory-model acquisition process! (Dunning-Kruger effect in action!!!)
  - Linux kernel uses small fraction of RCU's capabilities
    - Often with good reason!
  - Large number of litmus tests, with text file to record outcomes
  - Followed up by polite but firm questions about why...
  - For but one example...

Example RCU Litmus Test: Trigger on Weak CPUs?

```
void P0(void)
{
  rcu_read_lock();
  r1 = READ_ONCE(y);
  WRITE_ONCE(x, 1);
  rcu_read_unlock();
}
```

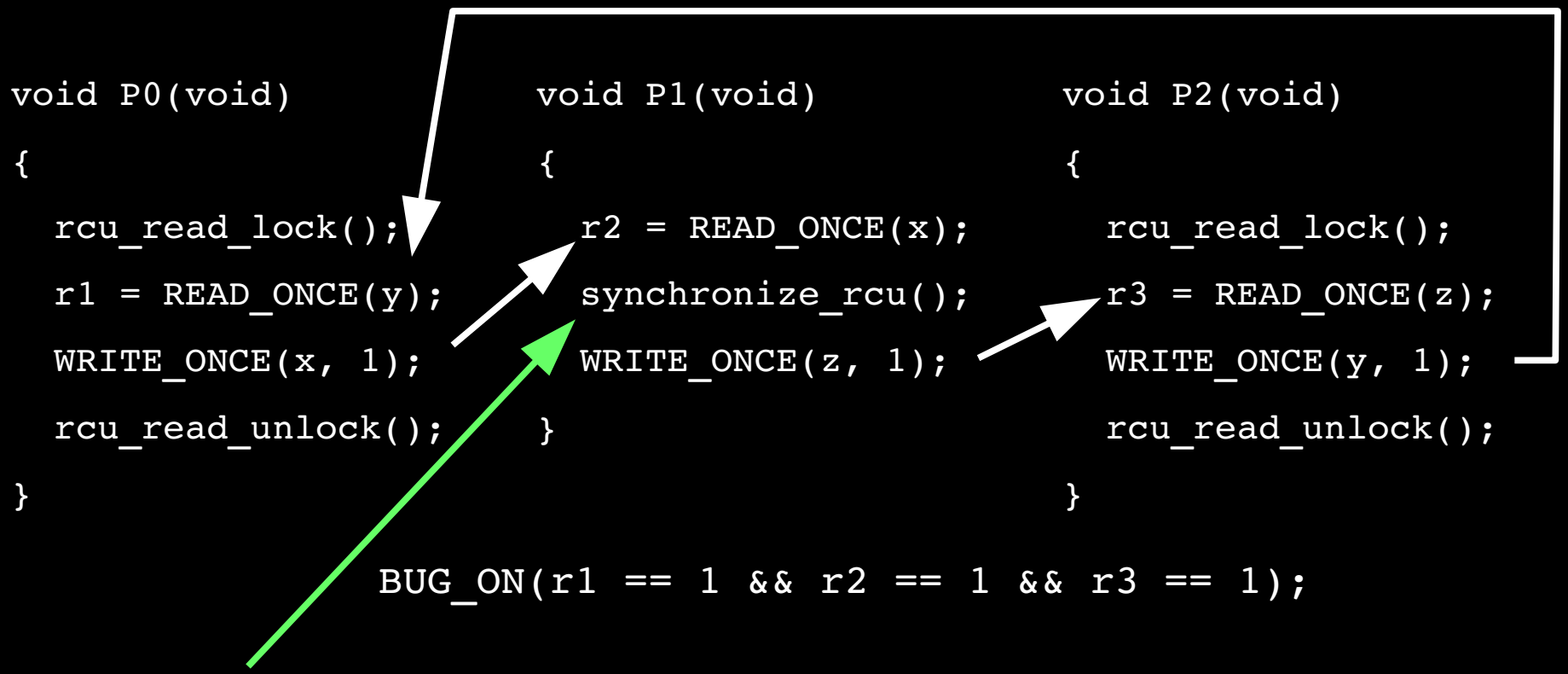
```
void P1(void)
{
  r2 = READ_ONCE(x);
  synchronize_rcu();
  WRITE_ONCE(z, 1);
}
```

```
void P2(void)
{
  rcu_read_lock();
  r3 = READ_ONCE(z);
  WRITE_ONCE(y, 1);
  rcu_read_unlock();
}
```

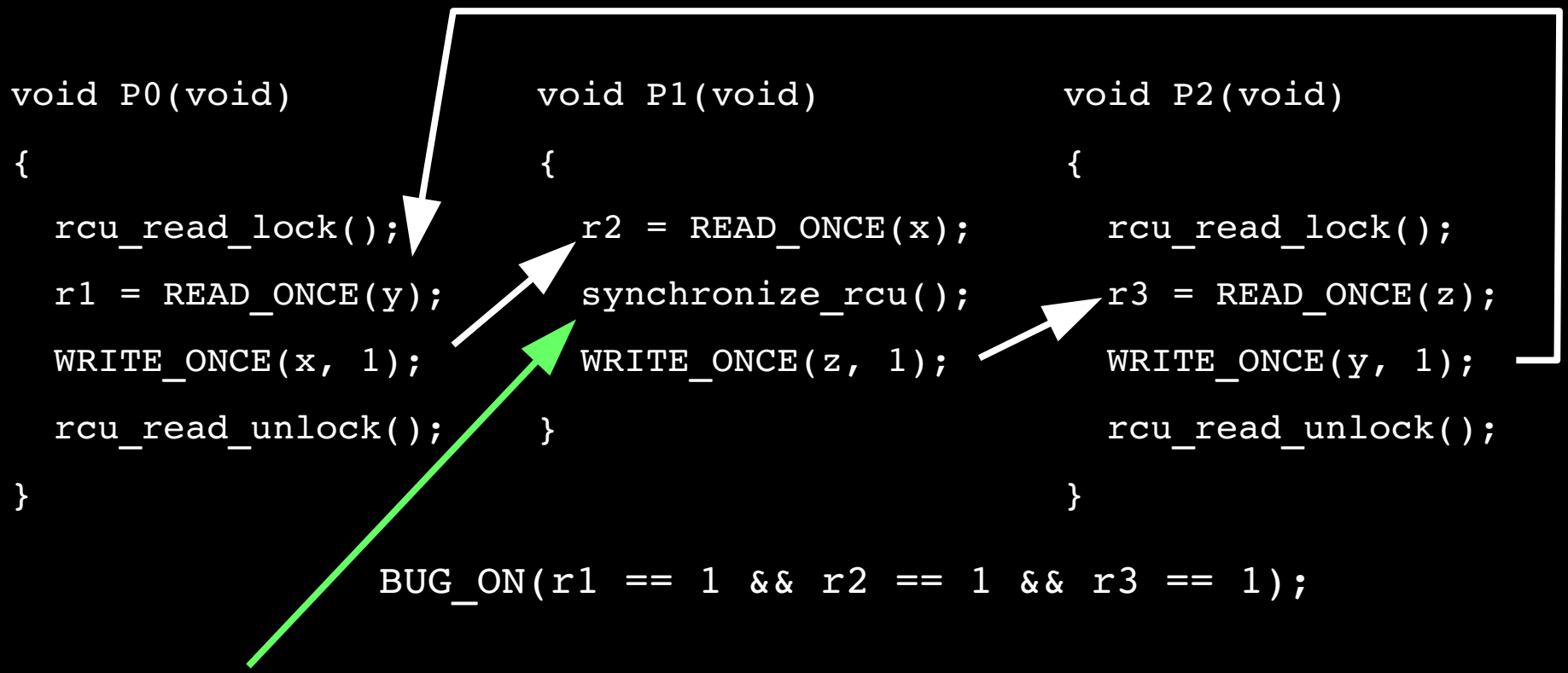
```
BUG_ON(r1 == 1 && r2 == 1 && r3 == 1);
```


C-RW-R+RW-G+RW-R.litmus

Example RCU Litmus Test: Trigger on Weak CPUs?

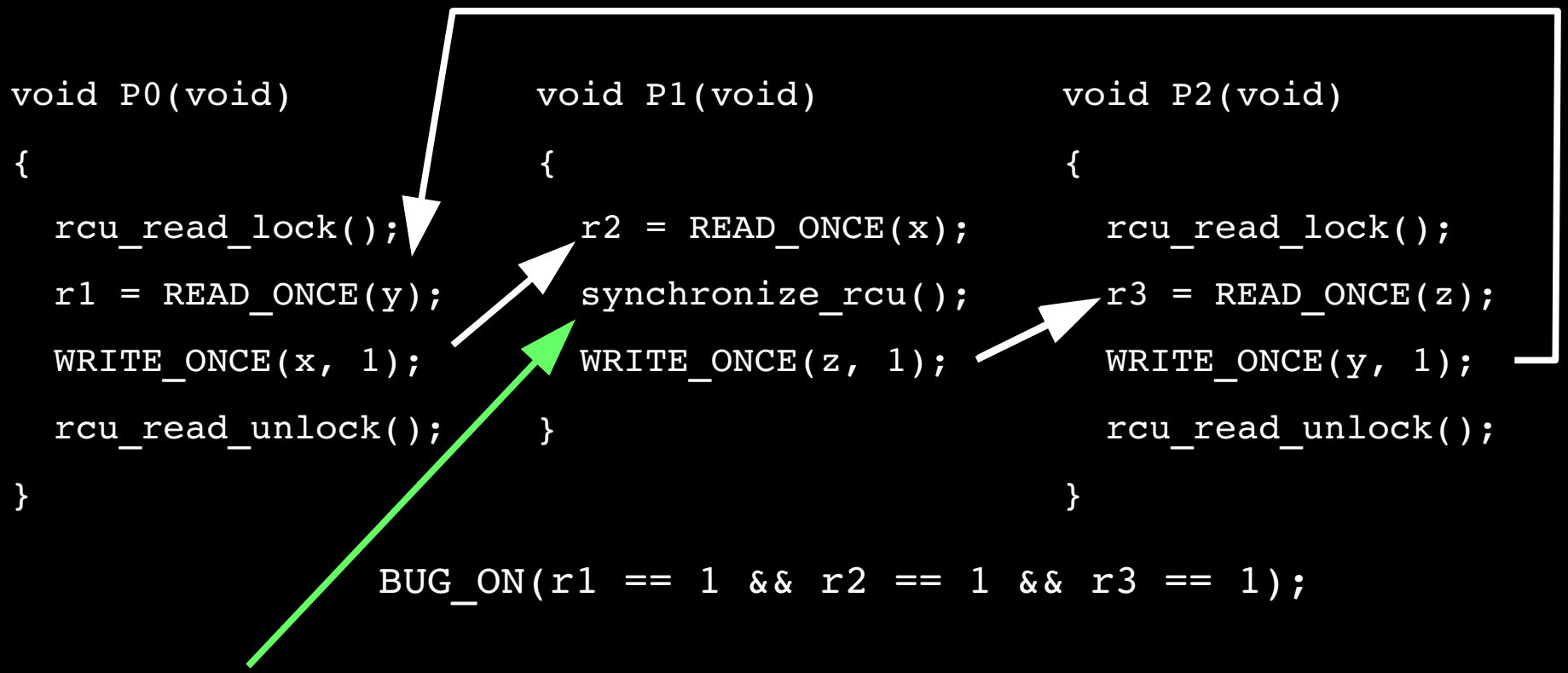


Example RCU Litmus Test: Trigger on Weak CPUs?



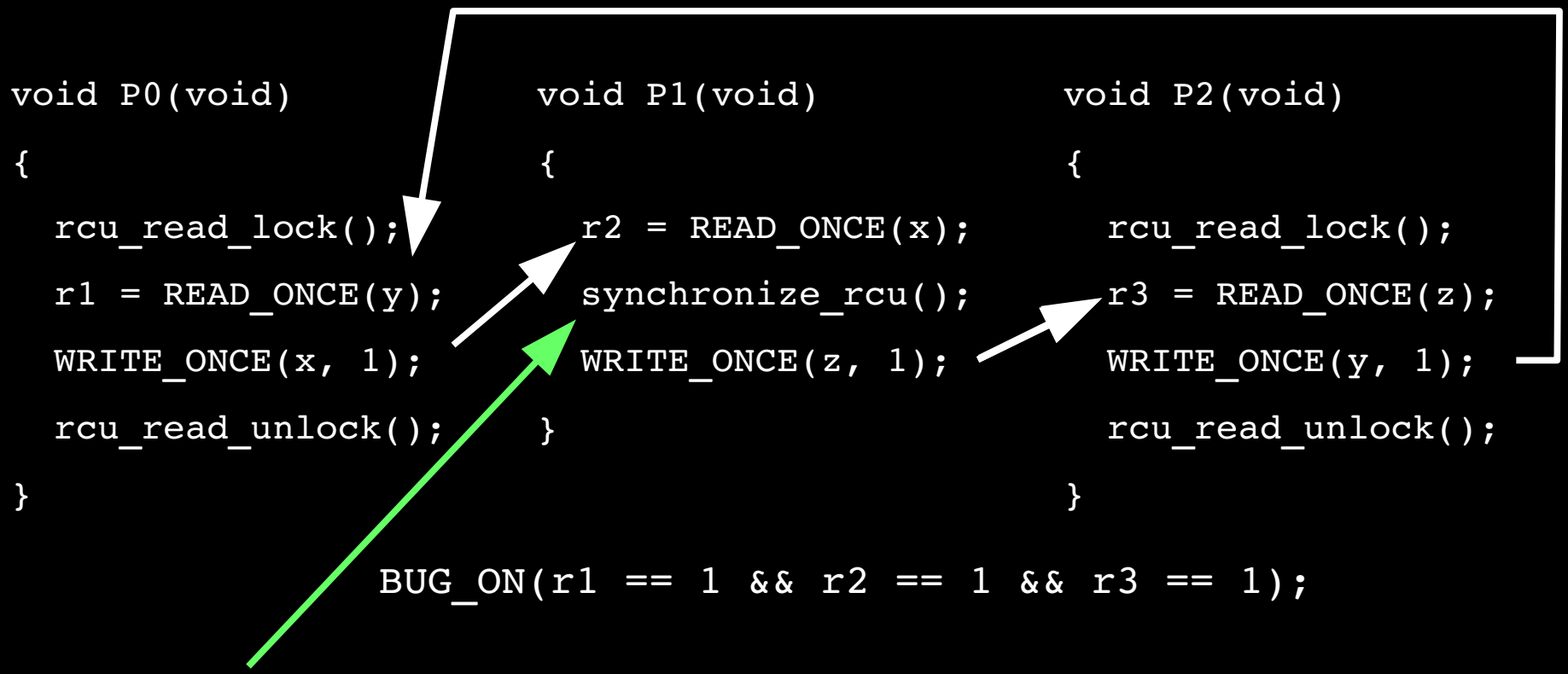
1. Any system doing this should have been strangled at birth
2. Reasonable systems really do this
3. There exist a great many unreasonable systems that really do this
4. A memory order is what I give to my hardware vendor!

Example RCU Litmus Test: Trigger on Weak CPUs?



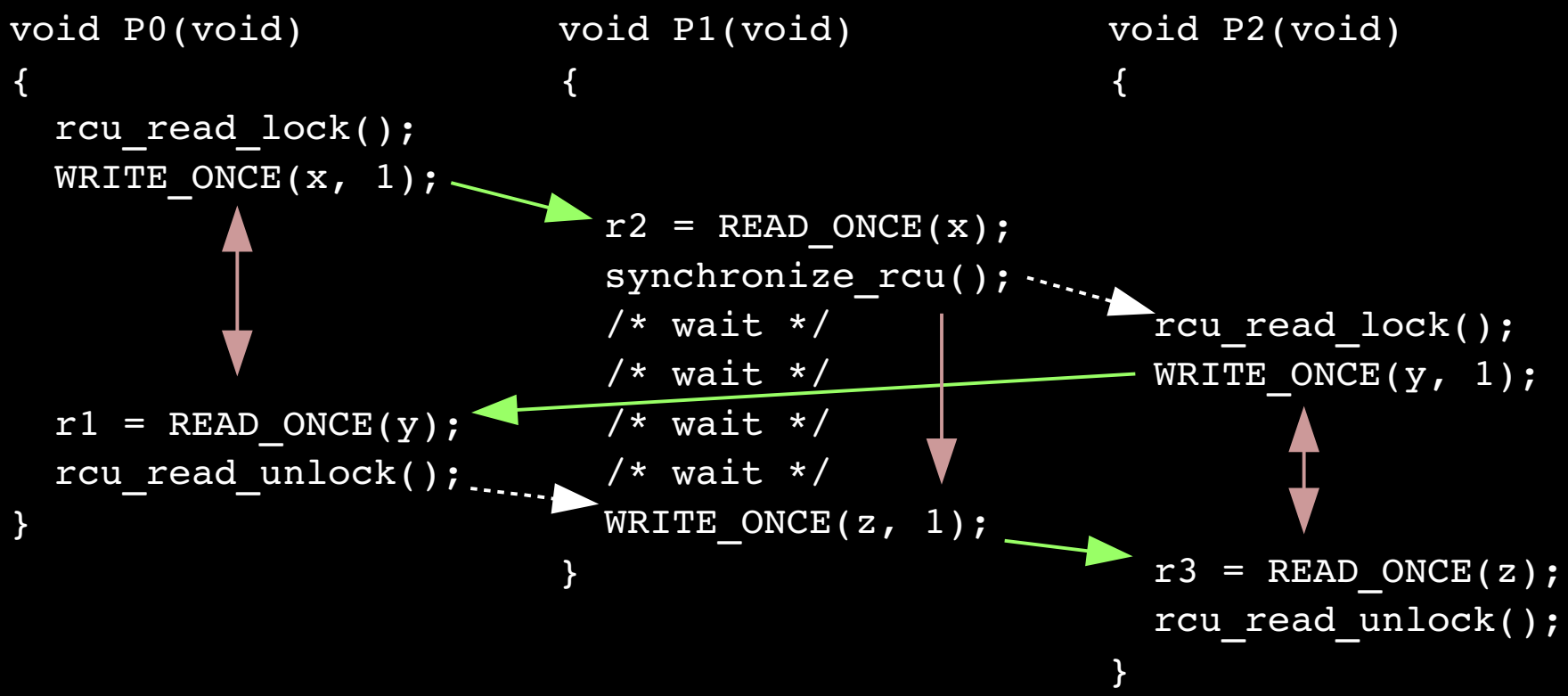
Litmus-test header comment: “Paul says allowed since mid-June”
No matter what you said, I agreed at some point in time!

Example RCU Litmus Test: Trigger on Weak CPUs?



Litmus-test header comment: “Paul says allowed since mid-June”
No matter what you said, I agreed at some point in time!
And this wasn't the only litmus test causing me problems!!!

RCU Litmus Test *Can* Trigger on Weak CPUs "This Cycle is Allowed"



But don't take my word for it...

The Tool Agrees (Given Late-2016 Memory Model)

```
$ herd7 -conf linux-kernel.cfg C-RW-R+RW-G+RW-R.litmus
Test auto/C-RW-R+RW-G+RW-R Allowed
States 8
0:r1=0; 1:r2=0; 2:r3=0;
0:r1=0; 1:r2=0; 2:r3=1;
0:r1=0; 1:r2=1; 2:r3=0;
0:r1=0; 1:r2=1; 2:r3=1;
0:r1=1; 1:r2=0; 2:r3=0;
0:r1=1; 1:r2=0; 2:r3=1;
0:r1=1; 1:r2=1; 2:r3=0;
0:r1=1; 1:r2=1; 2:r3=1;
Ok
Witnesses
Positive: 1 Negative: 7
Condition exists (0:r1=1 /\ 1:r2=1 /\ 2:r3=1)
Observation auto/C-RW-R+RW-G+RW-R Sometimes 1 7
Hash=0e5145d36c24bf7e57e9ef5f046716b8
```

Summer 2015 Rinse-Lather-Repeat Cycle

- The rinse-lather-repeat cycle:
  - Jade sends Paul litmus tests
    - RCU, non-RCU, combinations of RCU and non-RCU
  - Paul sends responses
  - Jade attempts to construct corresponding model
    - Which raises questions, which she passes along to Paul
    - Usually in the form of additional litmus tests
  - Paul realizes some responses are implementation-specific
  - Paul raises his level of abstraction, adjusts responses
- In a perfect world, Jack Slingwine and I would have fully defined RCU semantics back in the early 1990s
  - But you might have noticed that the world is imperfect!

At Summer's End...

- I create a writeup of RCU behavior
- This results in general rule:
  - If there are at least as many grace periods as read-side critical sections in a given cycle, then that cycle is forbidden
    - As in the earlier litmus test: Two critical sections, only one grace period
- Jade calls this “principled”
  - (Which is about as good as it gets for us Linux kernel hackers)
  - But she also says “difficult to represent as a formal memory model”
- However, summer is over, and Jade is out of time
  - She designates a successor

At Summer's End...

- I create a writeup of RCU behavior
- This results in general rule:
  - If there are at least as many grace periods as read-side critical sections in a given cycle, then that cycle is forbidden
    - As in the earlier litmus test: Two critical sections, only one grace period
- Jade calls this “principled”
  - (Which is about as good as it gets for us Linux kernel hackers)
  - But she also says “difficult to represent as a formal memory model”
- However, summer is over, and Jade is out of time
  - She designates a successor
- But first, Jade produced the first demonstration that a Linux-kernel memory model is feasible!!!
  - And forced me to a much better understanding of RCU!!!

Project Handoff: Jade's Successor

Luc Maranget, INRIA Paris (November 2015)

This Is Luc's First Exposure to RCU

This Is Luc's First Exposure to RCU

- It is my turn to use litmus tests as a form of communication
  - Sample tests that RCU should allow or forbid
    - Accompanied by detailed rationale for each
  - Series of RCU “implementations” in litmus-test language (AKA “LISA”)
    - With varying degrees of accuracy and solver overhead
    - Some of which require knowing the value loaded ***before*** the load
    - Which, surprisingly enough, is implementable in memory-model tools! “Prophecy variables”, they are called
  - Run Luc's models against litmus tests, return scorecard
    - With convergence, albeit slow convergence

This Is Luc's First Exposure to RCU

- It is my turn to use litmus tests as a form of communication
  - Sample tests that RCU should allow or forbid
    - Accompanied by detailed rationale for each
  - Series of RCU “implementations” in litmus-test language (AKA “LISA”)
    - With varying degrees of accuracy and solver overhead
    - Some of which require knowing the value loaded ***before*** the load
    - Which, surprisingly enough, is implementable in memory-model tools! “Prophecy variables”, they are called
  - Run Luc's models against litmus tests, return scorecard
    - With convergence, albeit slow convergence
- I try writing the RCU ordering rules myself
  - Luc: “I see what you are doing, but I don't like your coding style!”
  - Me: “Well, I am a kernel hacker, not a memory-ordering expert!”
  - Kernel-hacker evaluation of Luc's style: “Mutually assured recursion”
  - Luc's model of RCU also requires modifications to tooling

Luc's Model Passes Most Litmus Tests

- Luc: “I need you to break my model!”
  - Need automation: Scripts generate litmus tests and expected outcome
  - Currently at 2,722 automatically generated litmus tests to go with the 348 manually generated litmus tests
    - Which teaches me about mathematical “necklaces” and “bracelets”
  - Luc generated 1,879 more for good measure using the “diy” tool
  - Moral: Validation is critically important in theory as well as in practice
- But does the model match real hardware?
  - As represented by formal memory models?
  - As represented by real hardware implementations?
  - There will always be uncertainty: Provide two models, strong and weak
  - And who is going to run all the tests???
- But first: Luc produced first high-quality memory model for the Linux kernel that included a realistic RCU model!!!

Inject Hardware and Linux-Kernel Reality

Andrea Parri, Real-Time Systems Laboratory
Scuola Superiore Sant'Anna (January 2016)

Large Conversion Effort

- Created script to convert litmus test to Linux kernel module
  - And then ran the result on x86, ARM, and PowerPC
  - And on the actual hardware, just for good measure: Fun with types!!!
- Helped Luc add support for almost-C-language litmus tests
  - "r1 = READ_ONCE(x)" instead of LISA-code "r[once] r1 x"
- Contributed to the memory model itself
- Luc's infrastructure used to summarize results on the web
  - Compare results of different models, different hardware, and different litmus tests—extremely effective in driving memory-model evolution!

Model Comparison on the Web (Two Variants of RCU)

	RS2RS	SAMECRIT
LISA2Rt1G	Forbid	Allow
auto/RW-G+RW-R3	Forbid	Allow
auto/RW-G+RW-G+RW-R3	Forbid	Allow
auto/RW-G+RW-G+RW-G+RW-R3	Forbid	Allow
auto/RW-G+RW-G+RW-R3+RW-R3	Forbid	Allow
auto/RW-G+RW-R3+RW-G+RW-R3	Forbid	Allow

Summary of the differences for 2,000+ litmus tests!

Large Conversion Effort

- Results look pretty good, but are we just getting lucky???
  - Insufficient overlap between specialties!!!
  - Way too easy for us to talk past each other
    - Which would result in subtle flaws in the memory model
  - Need bridge between Linux-kernel RCU and formal memory models
- But first: Andrea developed and ran test infrastructure, plus contributed directly to the Linux-kernel memory model!!!

Bridging Between Linux Kernel and Formal Methods

Alan S. Stern, Rowland Institute at Harvard (February 2016)

Alan's Background

- Maintainer, Linux-kernel USB EHCI, OHCI, & UHCI drivers

A Bit More of Alan's Background

- Maintainer, Linux-kernel USB EHCI, OHCI, & UHCI drivers
- Education:
  - Harvard University, A.B. (Mathematics, summa cum laude), 1979
  - University of California, Berkeley, Ph.D. (Mathematics), 1984
- Selected Publications:
  - *NMR Data Processing*, Jeffrey C. Hoch and Alan S. Stern, Wiley-Liss, New York (1996).
  - "De novo Backbone and Sequence Design of an Idealized α/β-barrel Protein: Evidence of Stable Tertiary Structure", F. Offredi, F. Dubail, P. Kischel, K. Sarinski, A. S. Stern, C. Van de Weerdt, J. C. Hoch, C. Prosperi, J. M. Francois, S. L. Mayo, and J. A. Martial, J. Mol. Biol. 325, 163–174 (2003).
  - "User-Level Implementations of Read-Copy Update", Mathieu Desnoyers, Paul E. McKenney, Alan S. Stern, Michel R. Dagenais, and Jonathan Walpole, IEEE Trans. Par. Distr. Syst. 23, 375–382 (2012).

I Had Hoped That Alan Would Critique The Model

I Had Hoped That Alan Would Critique The Model Which He Did—By Rewriting It (Almost) From Scratch

Modeling RCU Read-Side Critical Sections

```
let matched = let rec
            unmatched-locks = Rcu-lock \ domain(matched)
        and unmatched-unlocks = Rcu-unlock \ range(matched)
        and unmatched = unmatched-locks | unmatched-unlocks
        and unmatched-po = [unmatched] ; po [unmatched]
        and unmatched-locks-to-unlocks =
                [unmatched-locks] ; po ; [unmatched-unlocks]
        and matched = matched | (unmatched-locks-to-unlocks \
                (unmatched-po ; unmatched-po))
        in matched
flag ~empty Rcu-lock \ domain(matched) as unbalanced-rcu-locking
flag ~empty Rcu-unlock \ range(matched) as unbalanced-rcu-locking
let crit = matched \ (po^-1 ; matched ; po^-1)
```

Handles multiple and nested critical sections
and also reports errors on mismatches!!!
And is an excellent example of “mutually assured recursion” design

Modeling RCU's Grace-Period Guarantee

```
let rscs = po ; crit^-1 ; po?
let link = hb* ; pb* ; prop
let gp-link = gp ; link
let rscs-link = rscs ; link
let rec rcu-path =
        gp-link |
        (gp-link ; rscs-link) |
        (rscs-link ; gp-link) |
        (rcu-path ; rcu-path) |
        (gp-link ; rcu-path ; rscs-link) |
        (rscs-link ; rcu-path ; gp-link)
irreflexive rcu-path as rcu
```

Handles arbitrary critical-section/grace-period combinations,
and also interfaces to remainder of memory model
And all of this in only 24 lines of code!!!

Small Example of Cat Language: Single-Variable SC

Small Example of Cat Language: Single-Variable SC

```
let com = rf | co | fr
let coherence-order = po-loc | com
acyclic coherence-order
```

- “rf” relation connects write to reads returning the value written: Causal!
- “co” relation connects pairs of writes to same variable
- “fr” relation connects reads to later writes to same variable (fr = rf^1 ; co)
- “po-loc” relation connects pairs of accesses to same variable within given thread
- Result: Aligned machine-sized accesses to given variable are globally ordered
- Note: Full memory model is about 200 lines of code!

Single-Variable SC Litmus Test

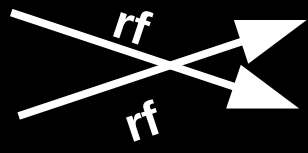
```
P0(void)                        P1(void)
{                               {
        WRITE_ONCE(x, 3);               r1 = READ_ONCE(x);
        WRITE_ONCE(x, 4);               r2 = READ_ONCE(x);
}                               }
```

```
BUG_ON(r1 == 4 && r2 == 3);
```

C-CO+o-o+o-o.litmus

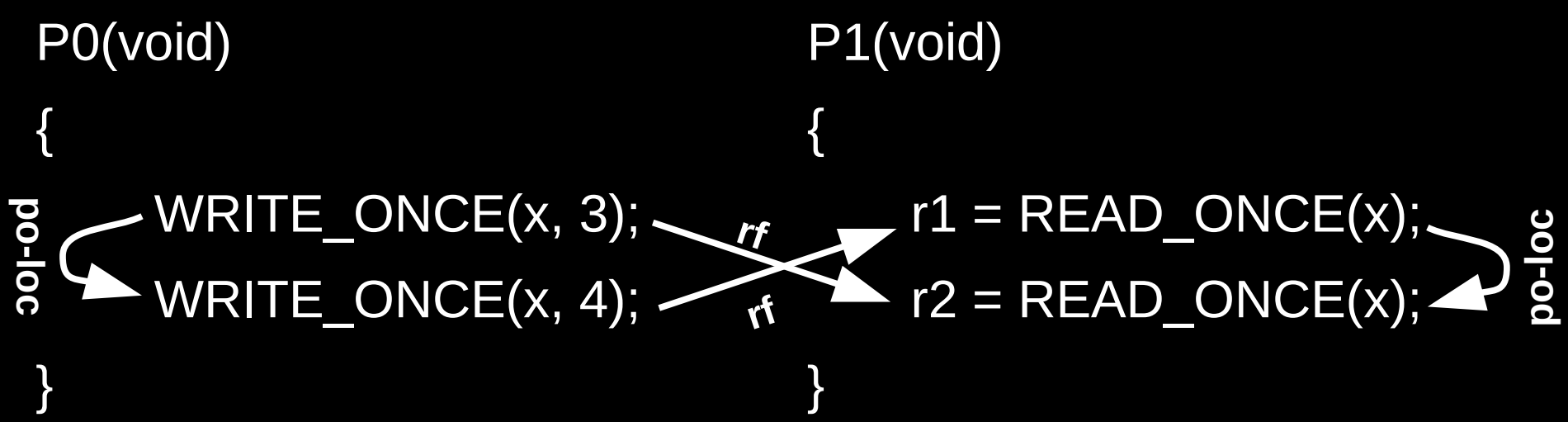
Single-Variable SC Litmus Test: rf Relationships

```
P0(void)                        P1(void)
{                               {
        WRITE_ONCE(x, 3);               r1 = READ_ONCE(x);
        WRITE_ONCE(x, 4);               r2 = READ_ONCE(x);
}                               }
```



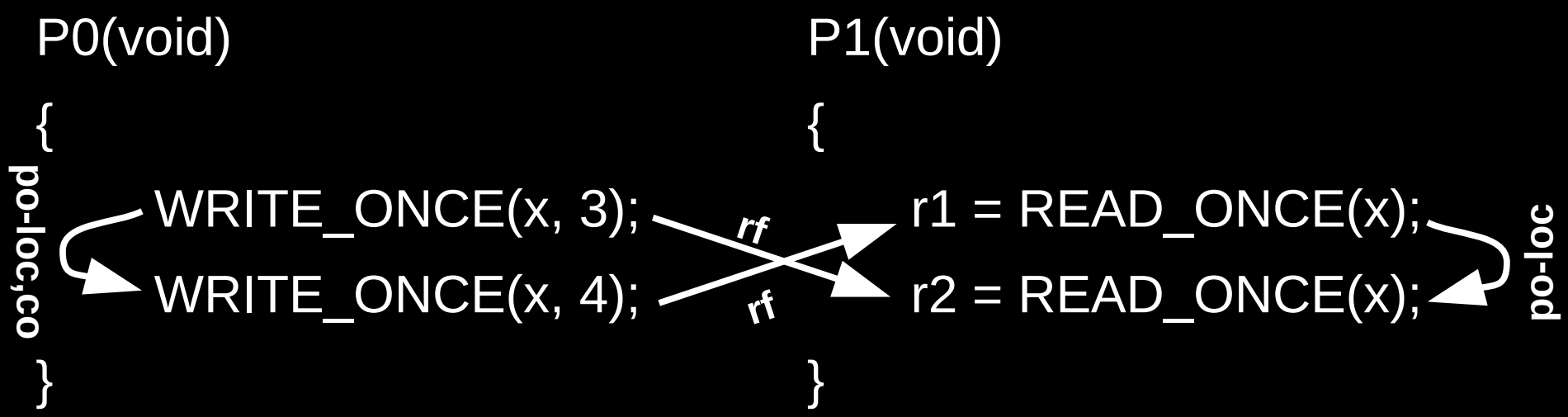

```
BUG_ON(r1 == 4 && r2 == 3);
```

Single-Variable SC Litmus Test: po-loc Relationships



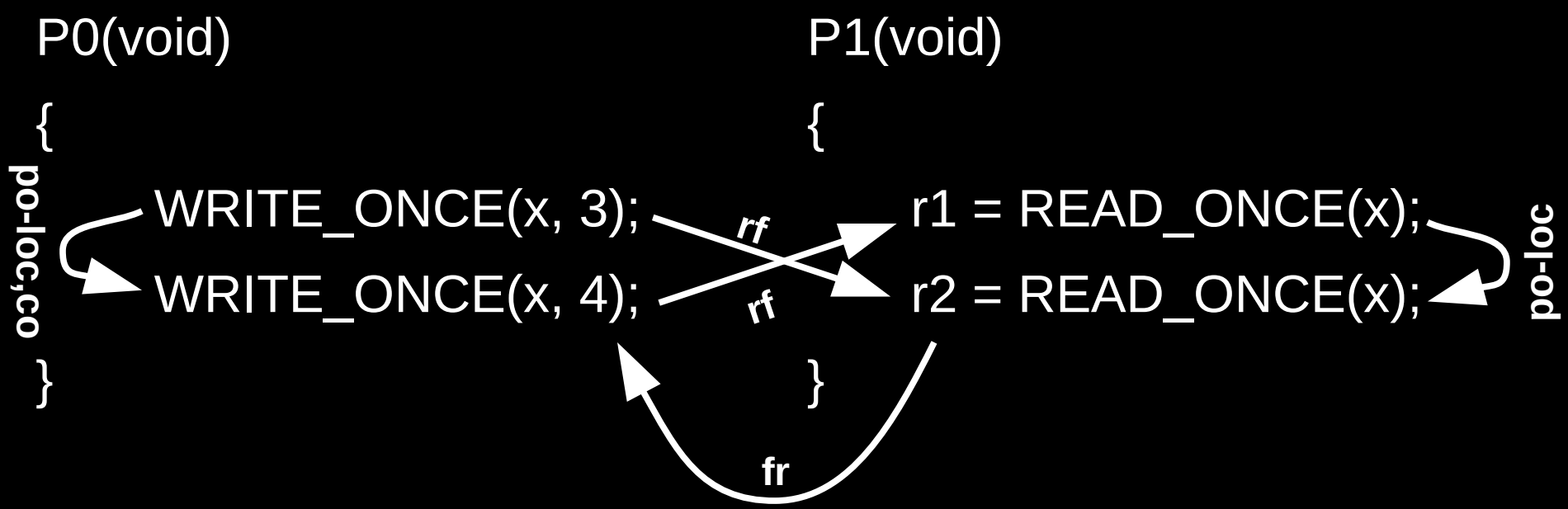

```
BUG_ON(r1 == 4 && r2 == 3);
```

Single-Variable SC Litmus Test: co Relationship



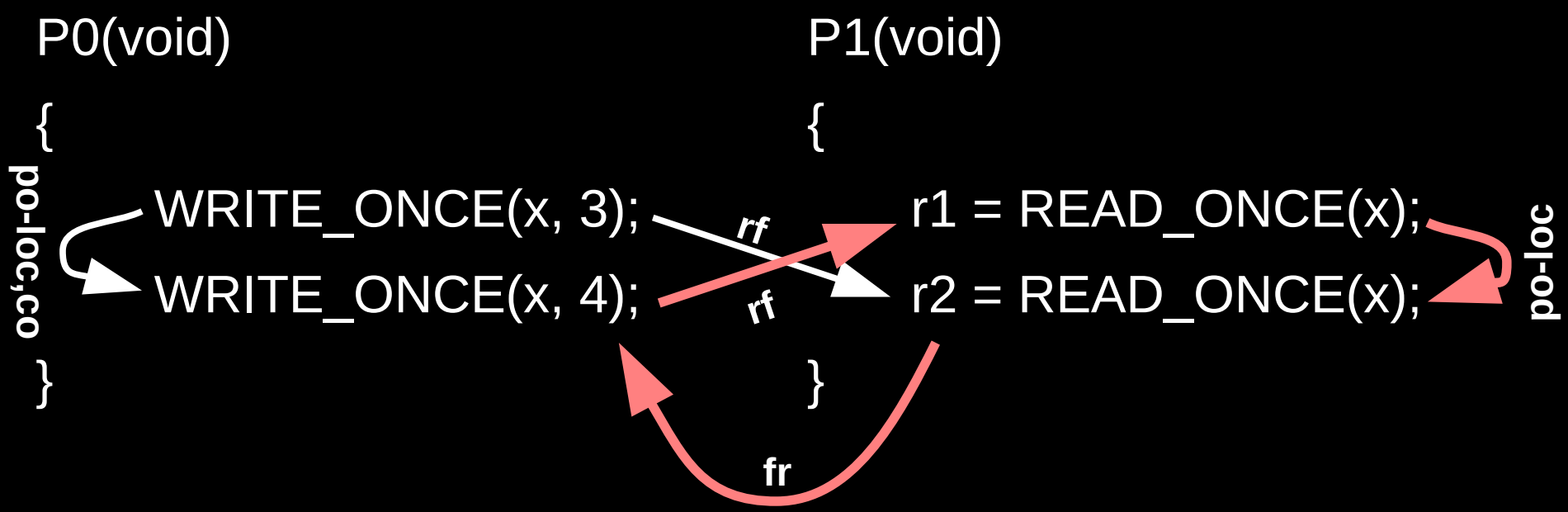

```
BUG_ON(r1 == 4 && r2 == 3);
```

Single-Variable SC Litmus Test: fr Relationships




```
BUG_ON(r1 == 4 && r2 == 3);
```

Single-Variable SC Litmus Test: Acyclic Check



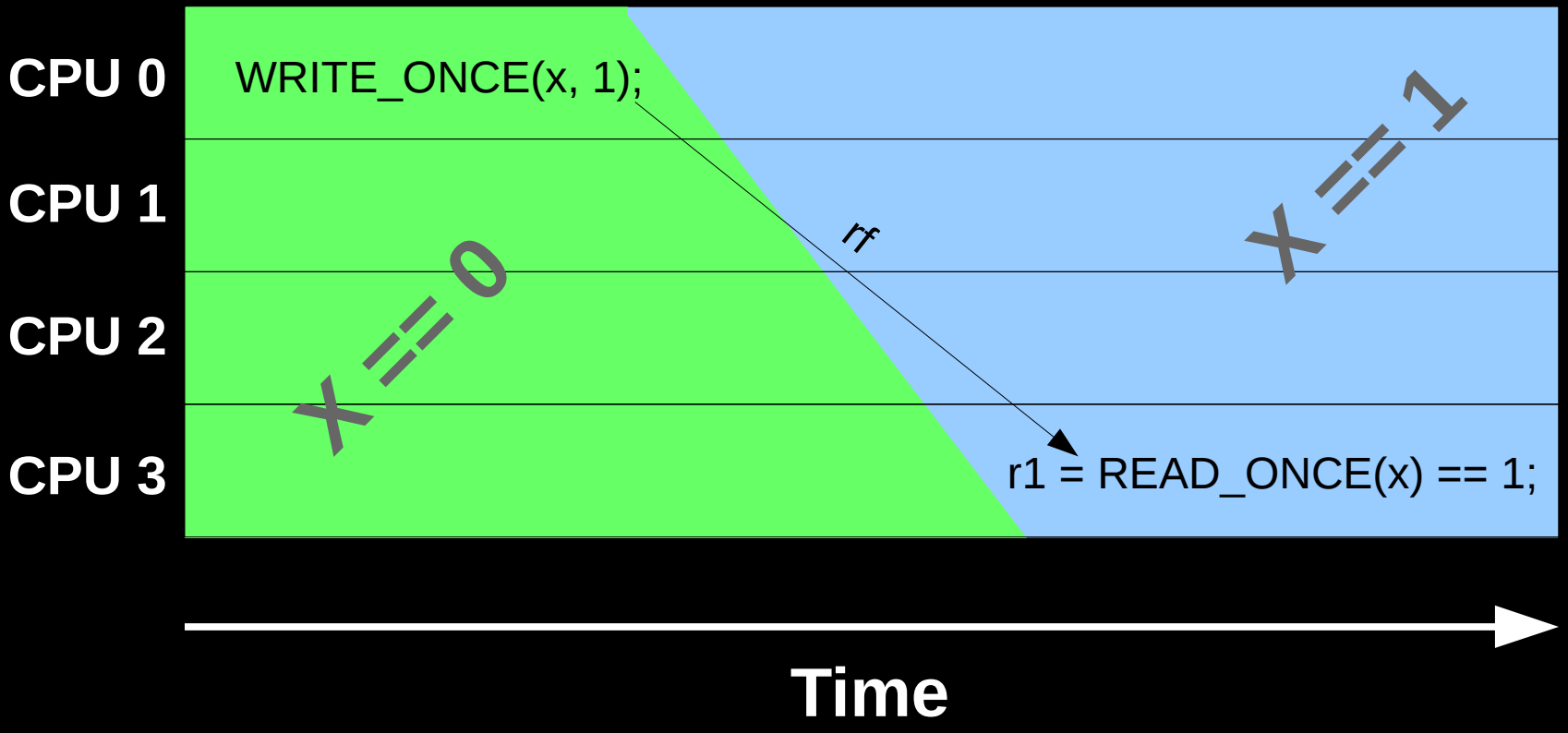

```
BUG_ON(r1 == 4 && r2 == 3);
```

Cycle, thus forbidden!

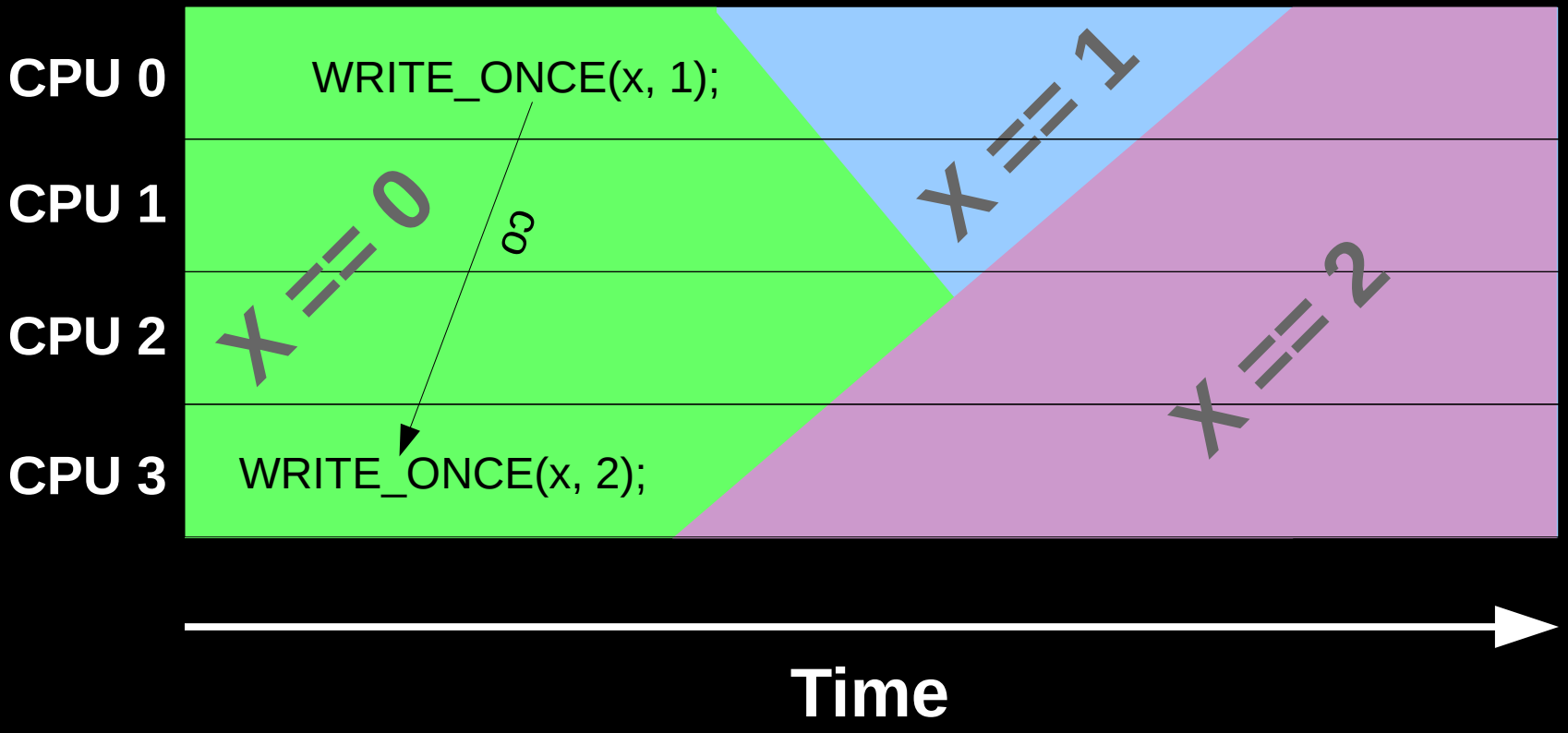
(Cycles are a generalization of memory-barrier pairing)

Not All Communications Relations Are Created Equal

Ordering vs. Time: The Reads-From (rf) Relation



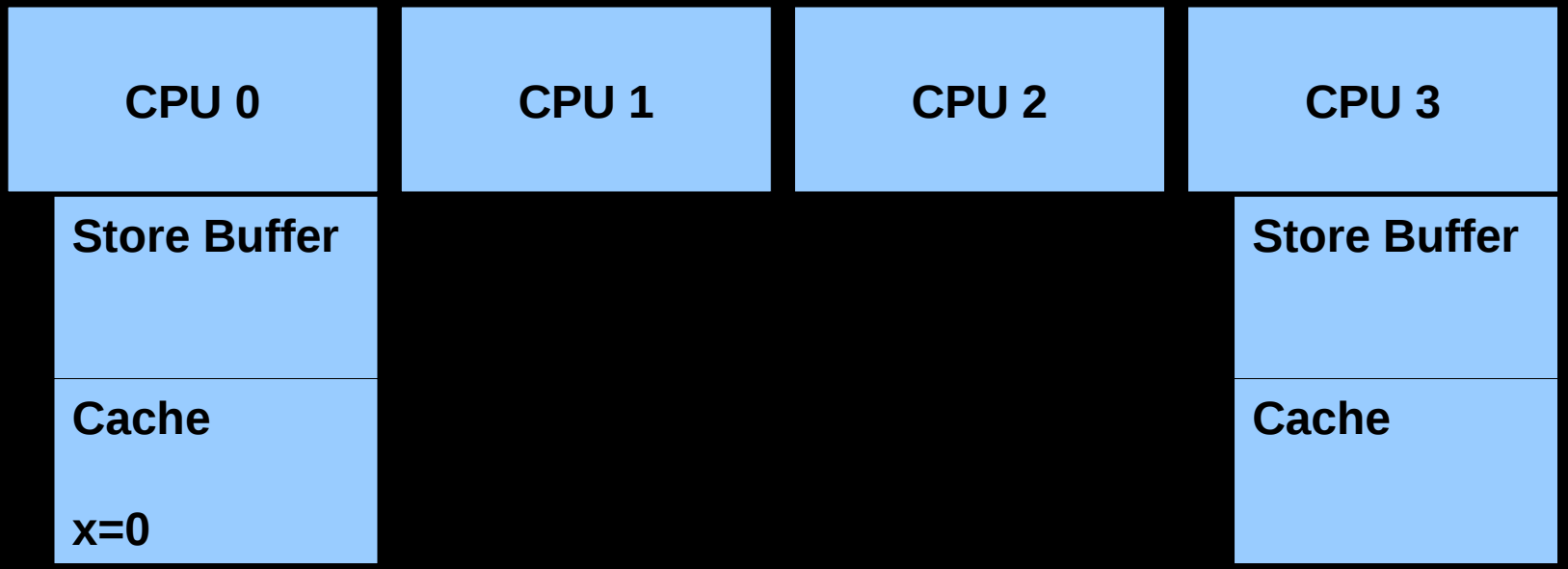
Ordering vs. Time: The Coherence (co) Relation Can Go Backwards In Time!



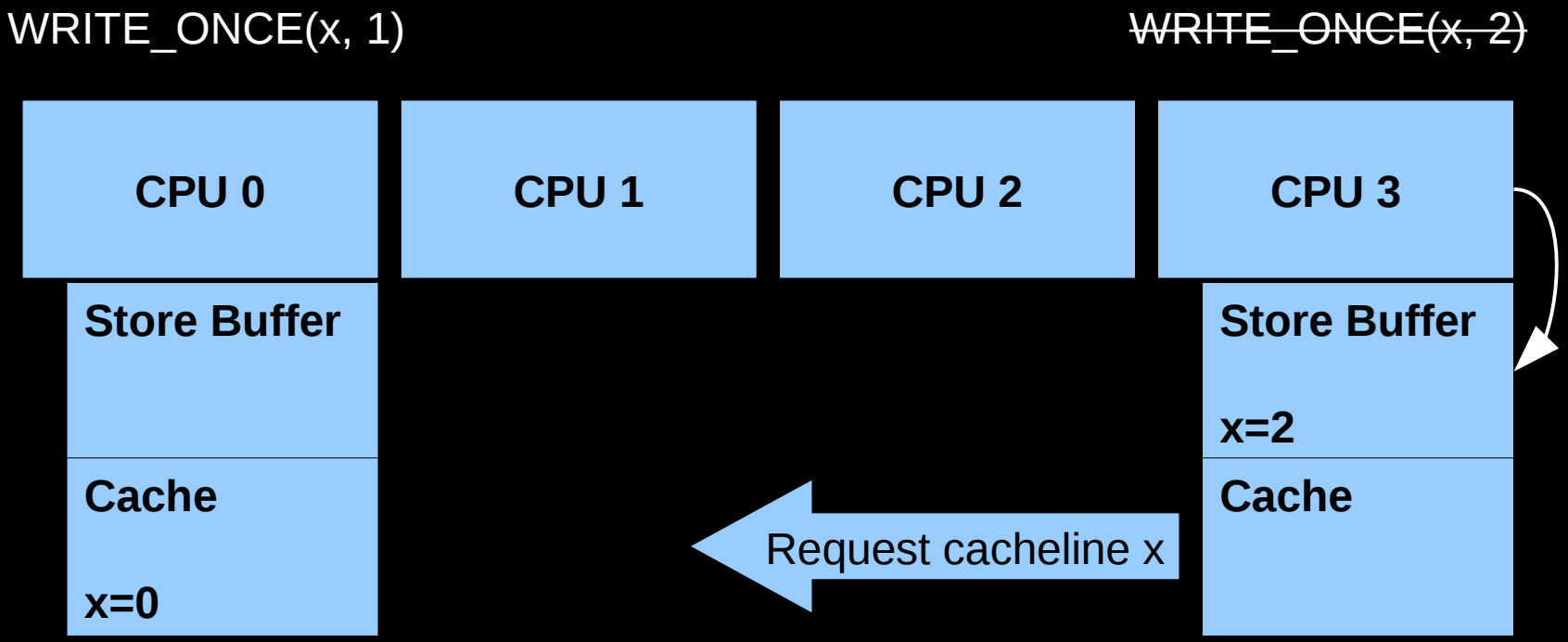
Ordering vs. Time: The Coherence (co) Relation Can Go Backwards In Time! How Can This Happen? (1/7)

WRITE_ONCE(x, 1)

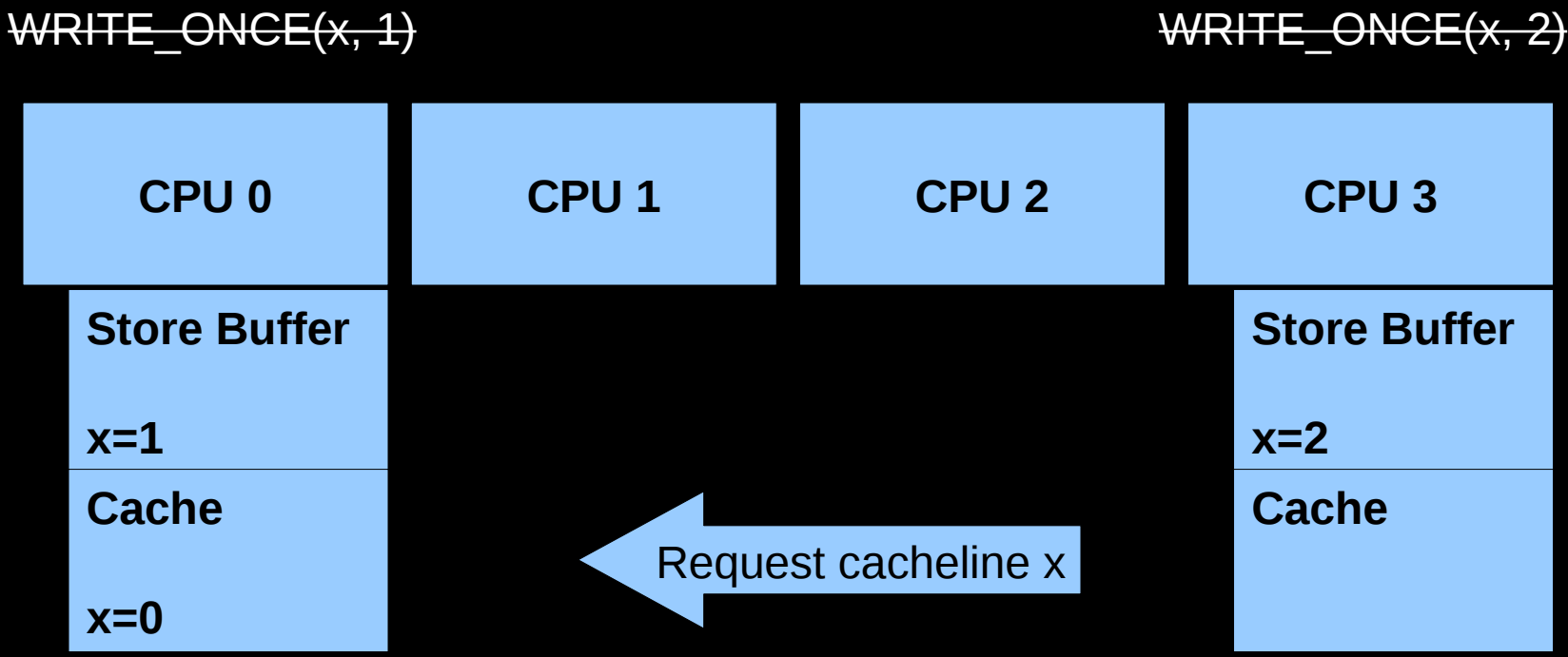
WRITE_ONCE(x, 2)



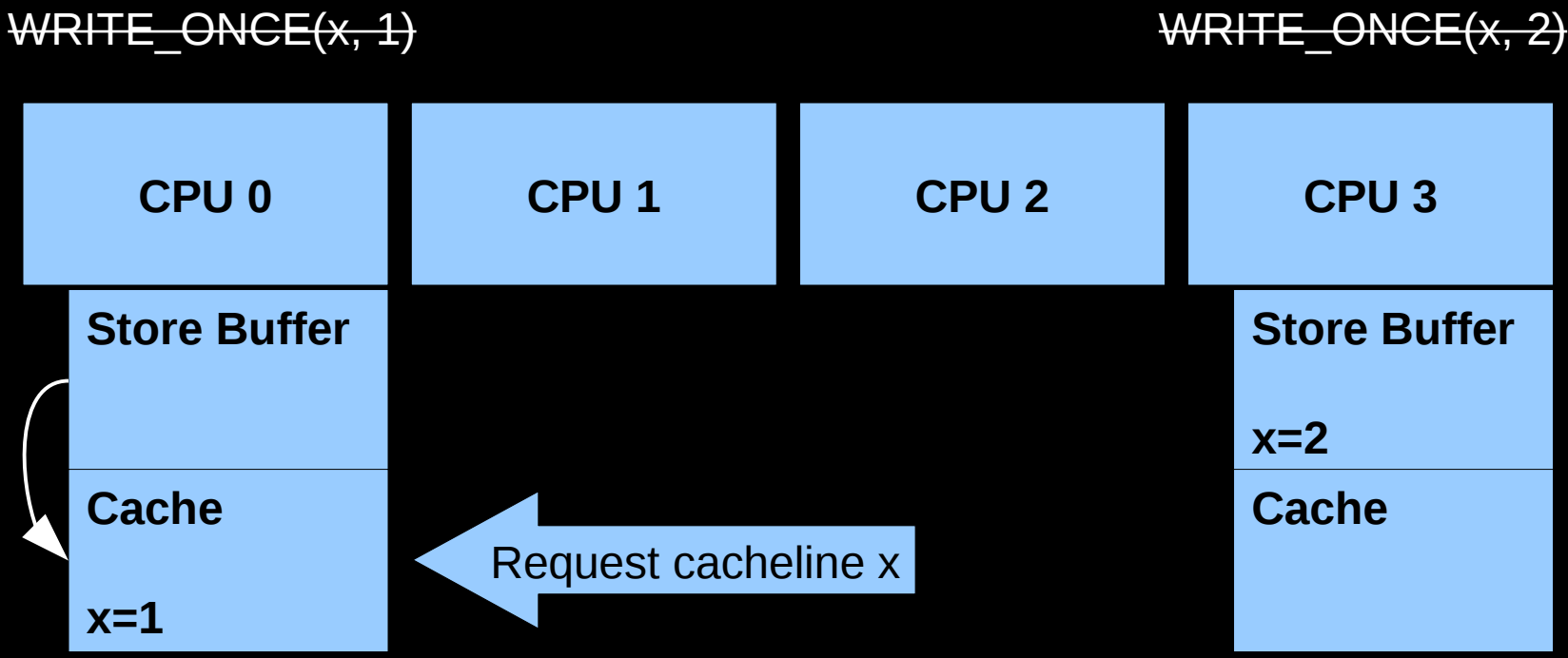
Ordering vs. Time: The Coherence (co) Relation Can Go Backwards In Time! How Can This Happen? (2/7)



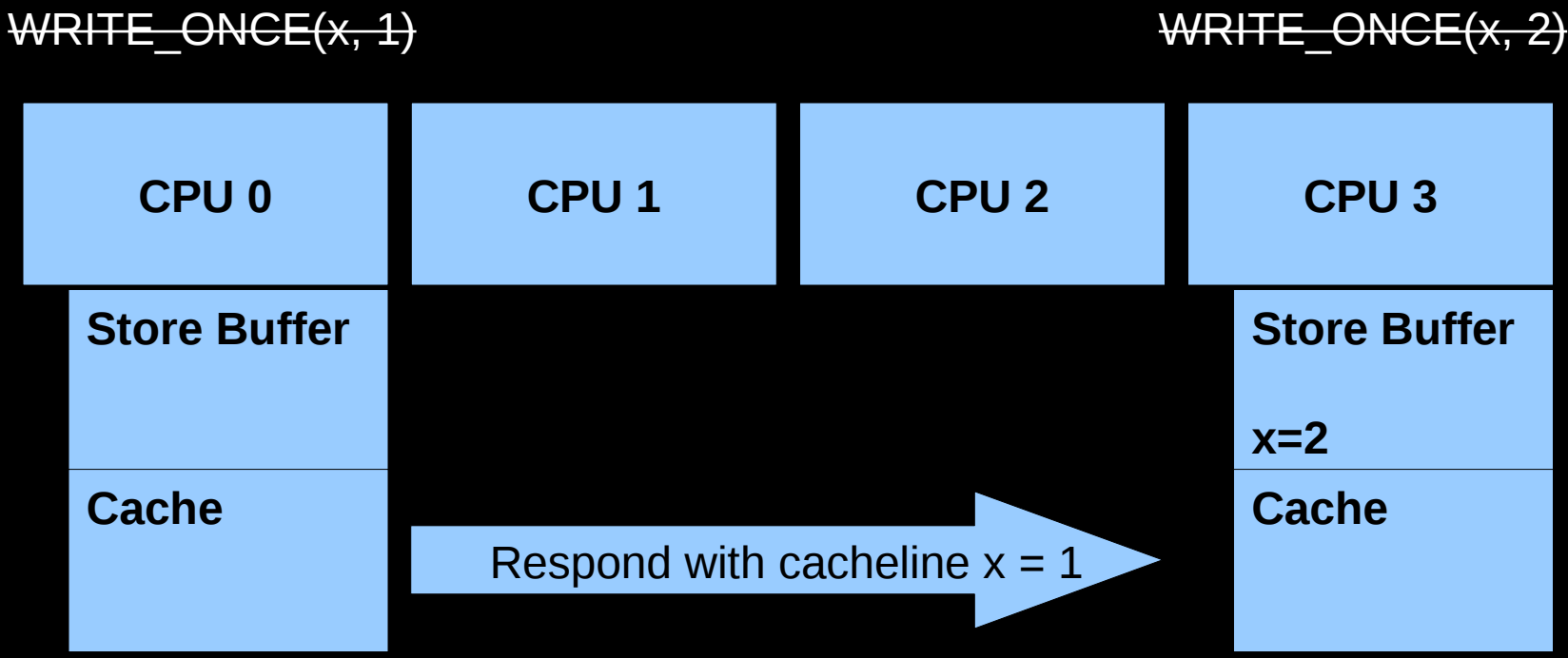
Ordering vs. Time: The Coherence (co) Relation Can Go Backwards In Time! How Can This Happen? (3/7)



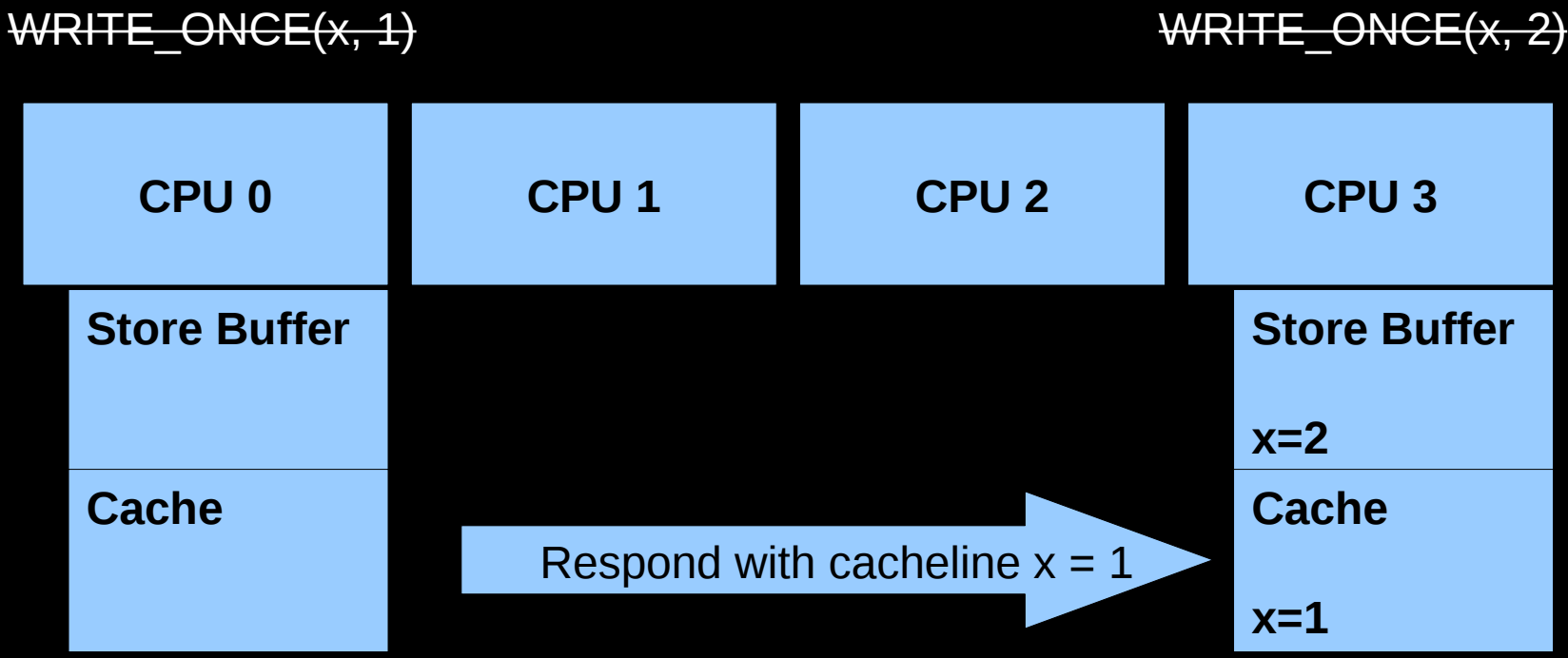
Ordering vs. Time: The Coherence (co) Relation Can Go Backwards In Time! How Can This Happen? (4/7)



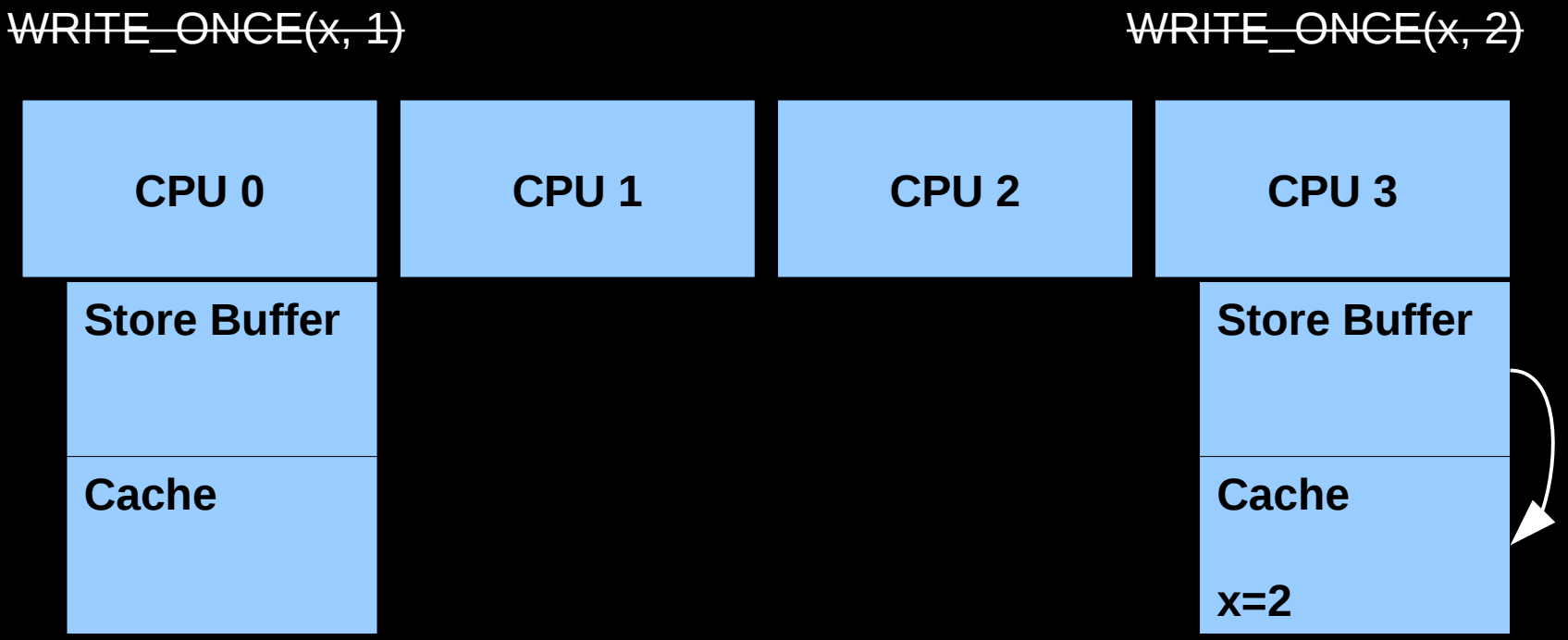
Ordering vs. Time: The Coherence (co) Relation Can Go Backwards In Time! How Can This Happen? (5/7)



Ordering vs. Time: The Coherence (co) Relation Can Go Backwards In Time! How Can This Happen? (6/7)

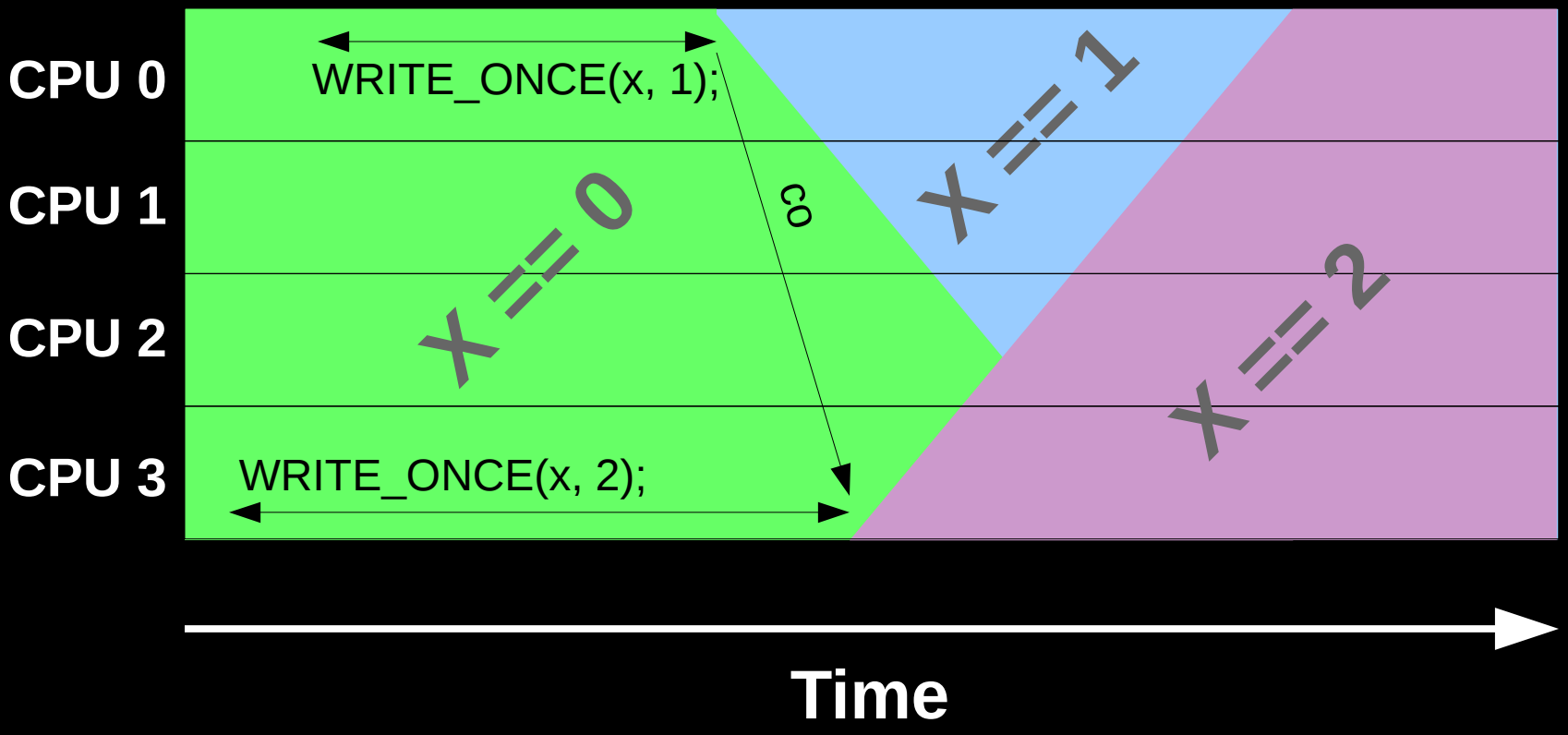


Ordering vs. Time: The Coherence (co) Relation Can Go Backwards In Time! How Can This Happen? (7/7)

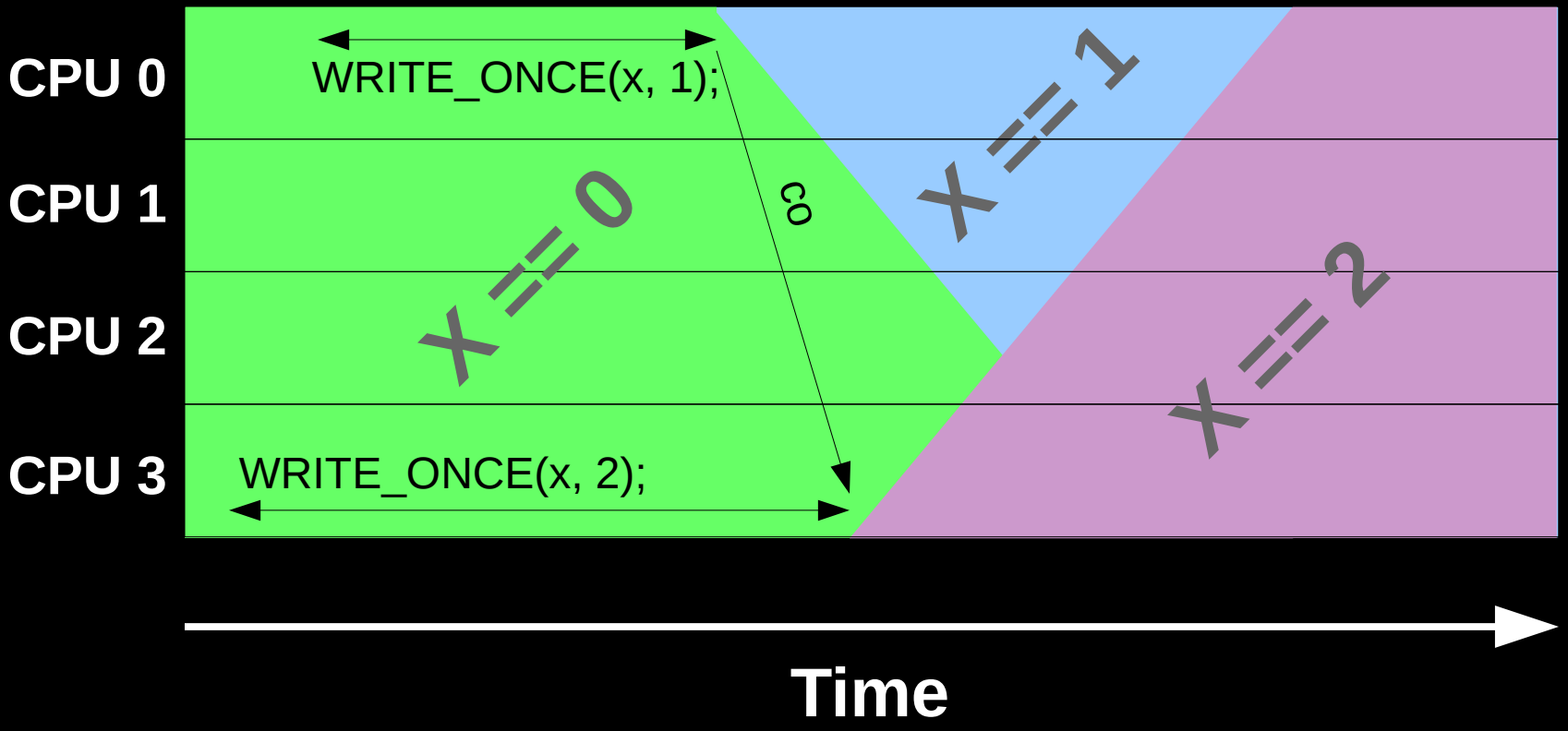


Writes are ***not*** instantaneous!

Ordering vs. Time: But the Coherence (co) Relation Goes *Forward* in Time Based on Cacheline!!!

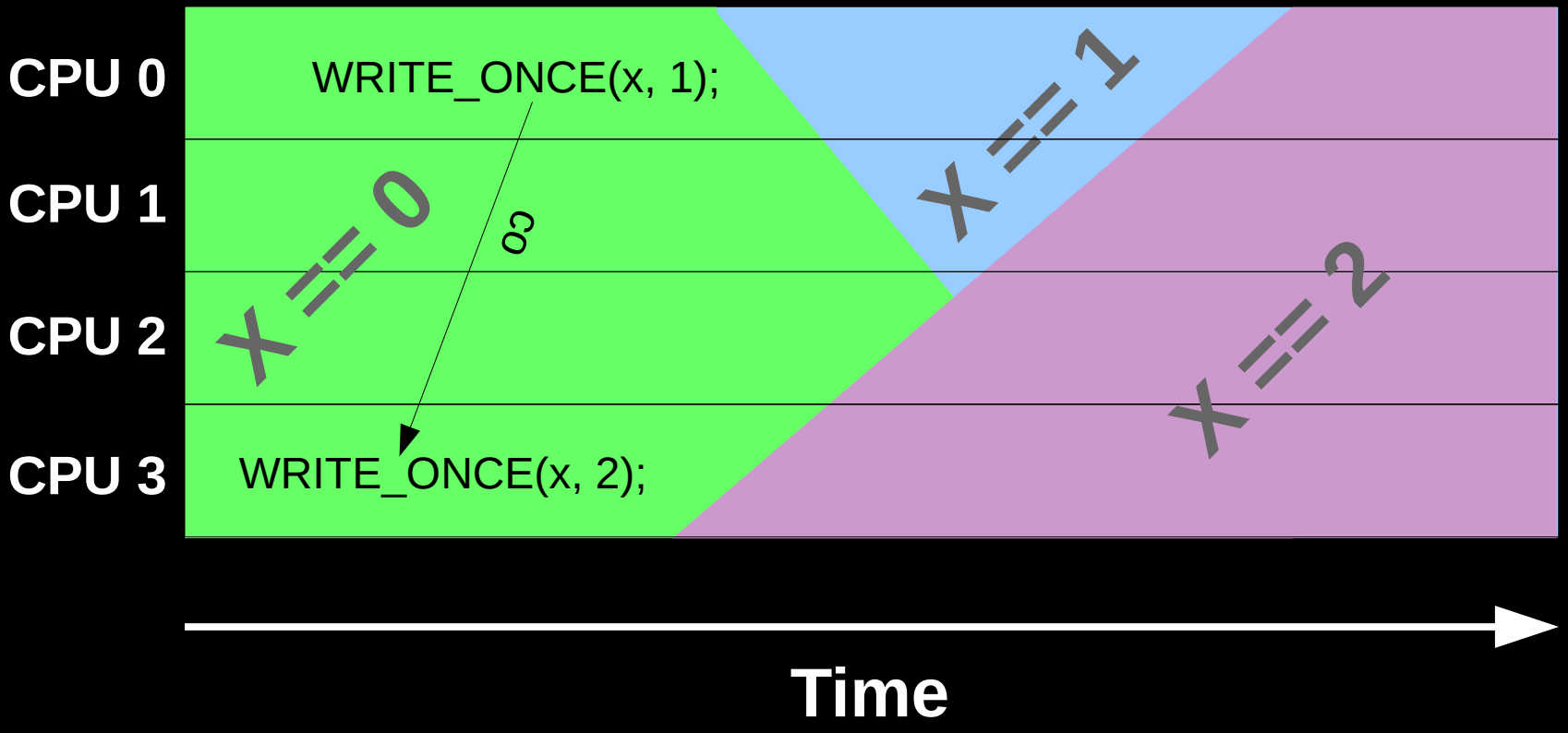


Ordering vs. Time: But the Coherence (co) Relation Goes *Forward* in Time Based on Cacheline!!!

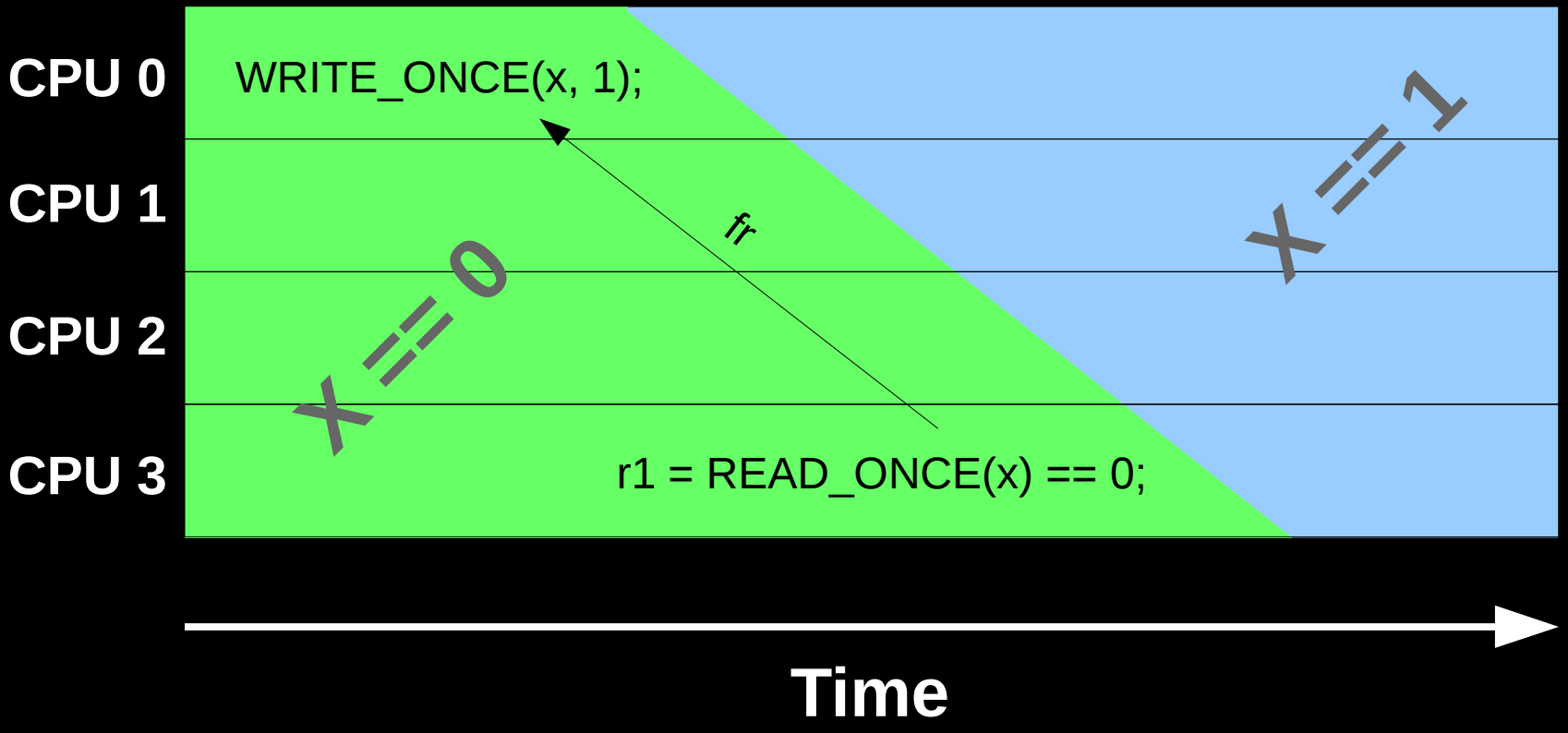


But cacheline movement is not directly visible to normal SW!

We Therefore Think in Terms of the Coherence (co) Relation Going Backwards In Time



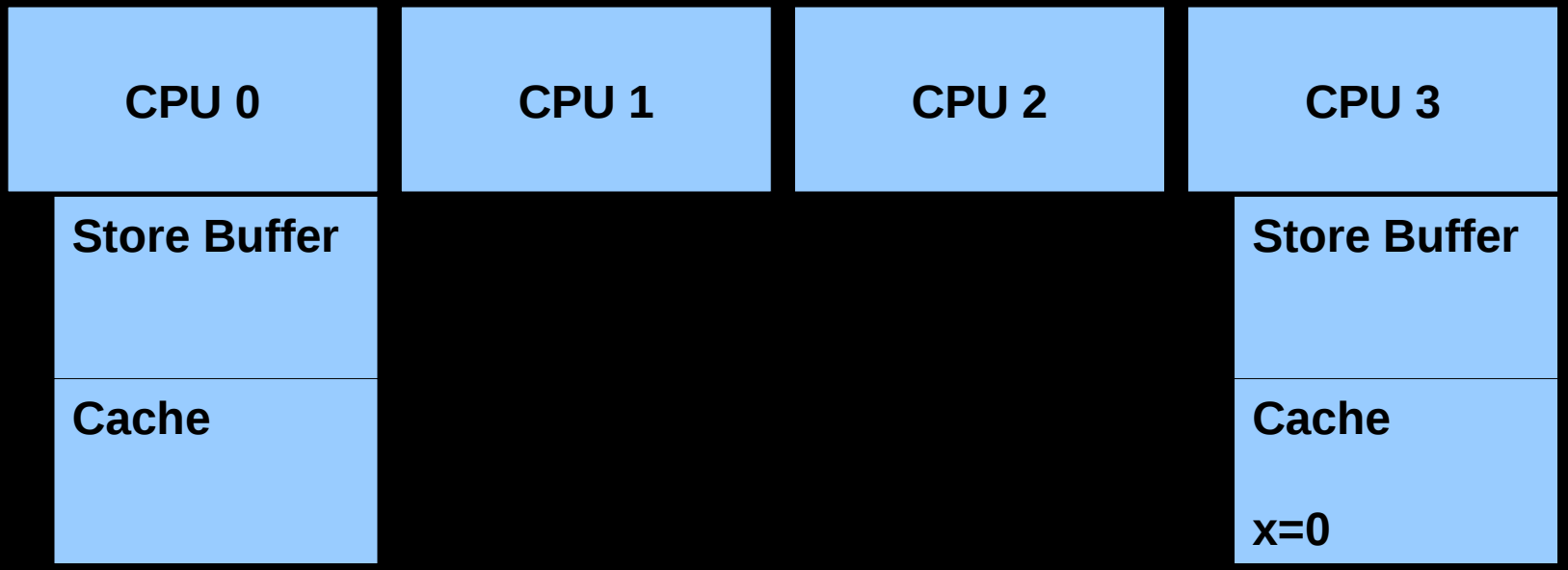
Ordering vs. Time: The From-Reads (fr) Relation Can Also Go Backwards In Time!



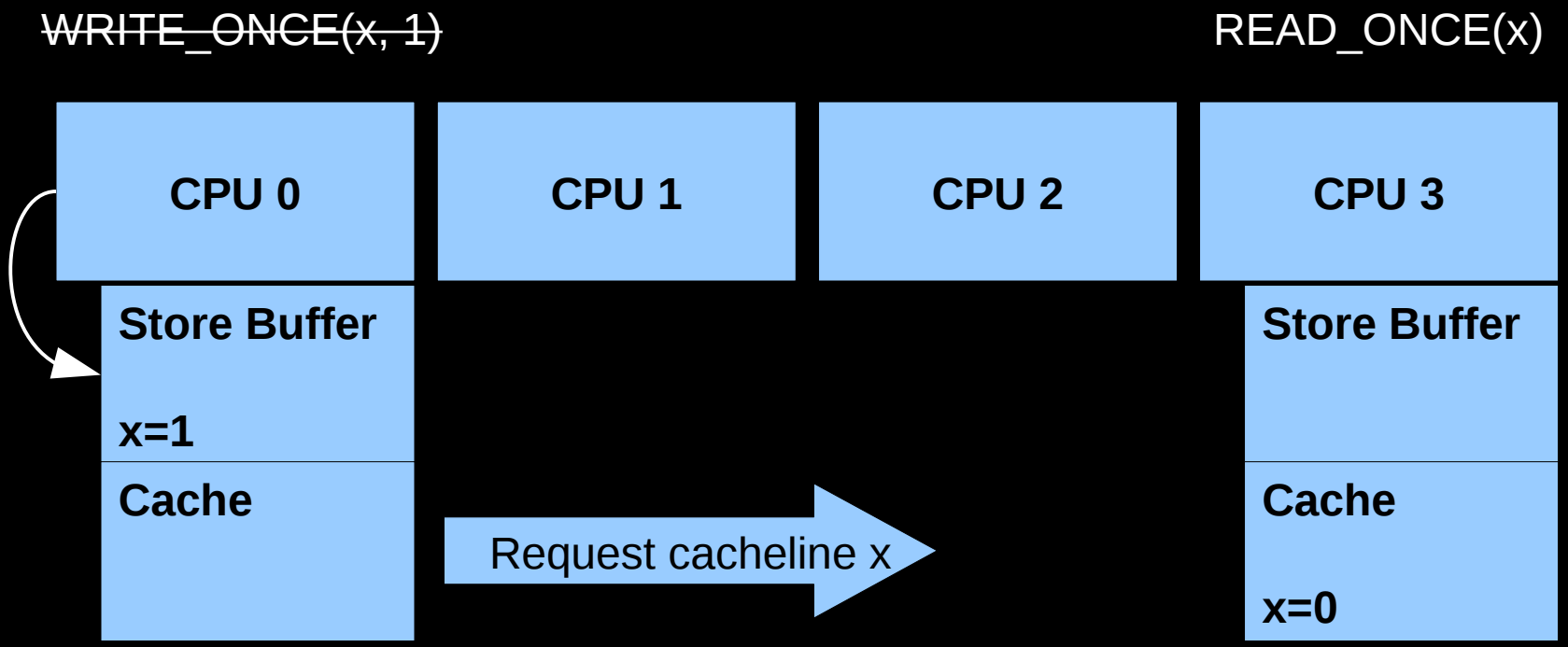
Ordering vs. Time: The From-Reads (fr) Relation Can Also Go Backwards In Time! (1/7)

WRITE_ONCE(x, 1)

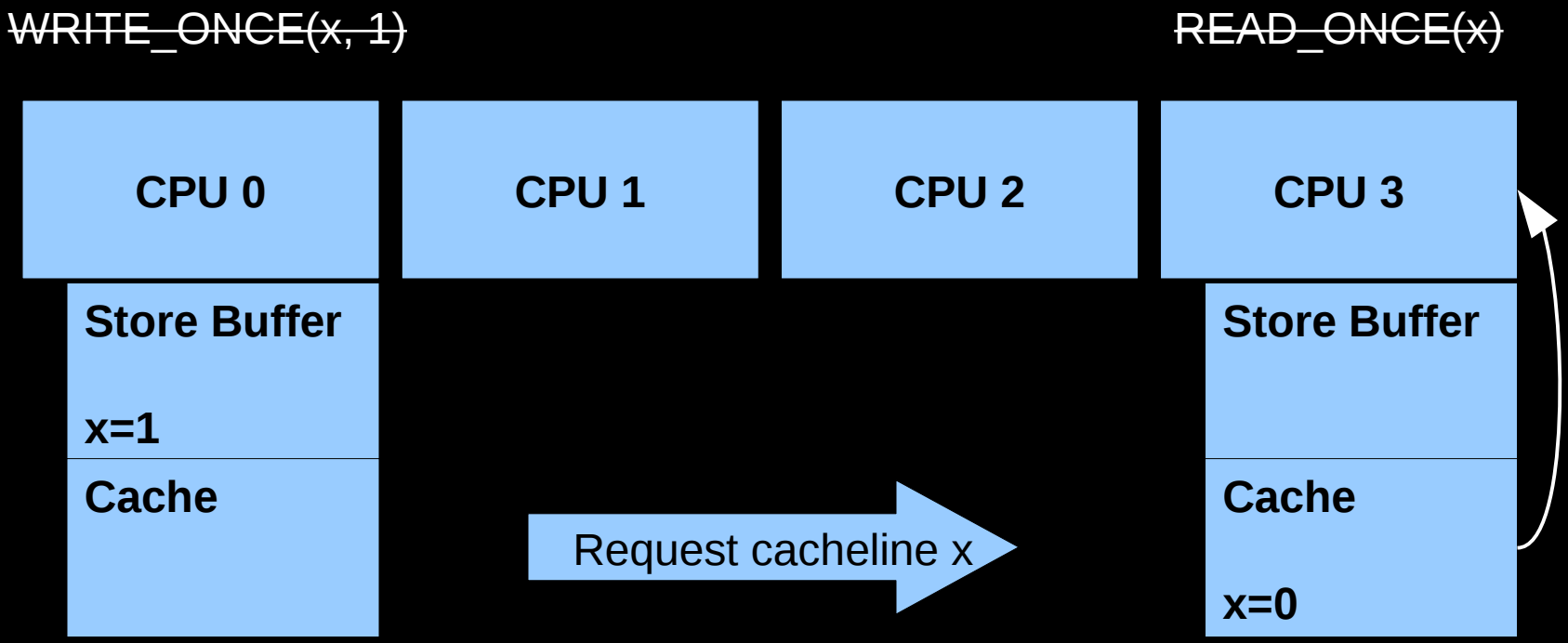
READ_ONCE(x)



Ordering vs. Time: The From-Reads (fr) Relation Can Also Go Backwards In Time! (2/7)

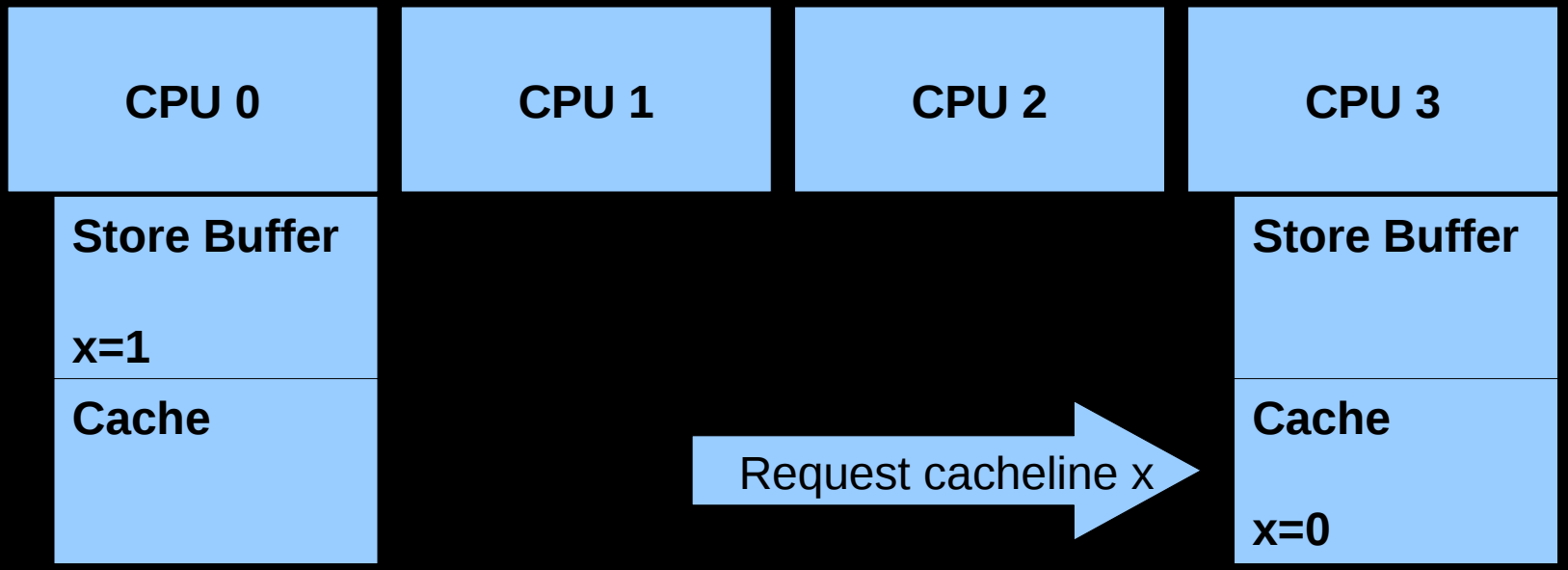


Ordering vs. Time: The From-Reads (fr) Relation Can Also Go Backwards In Time! (3/7)



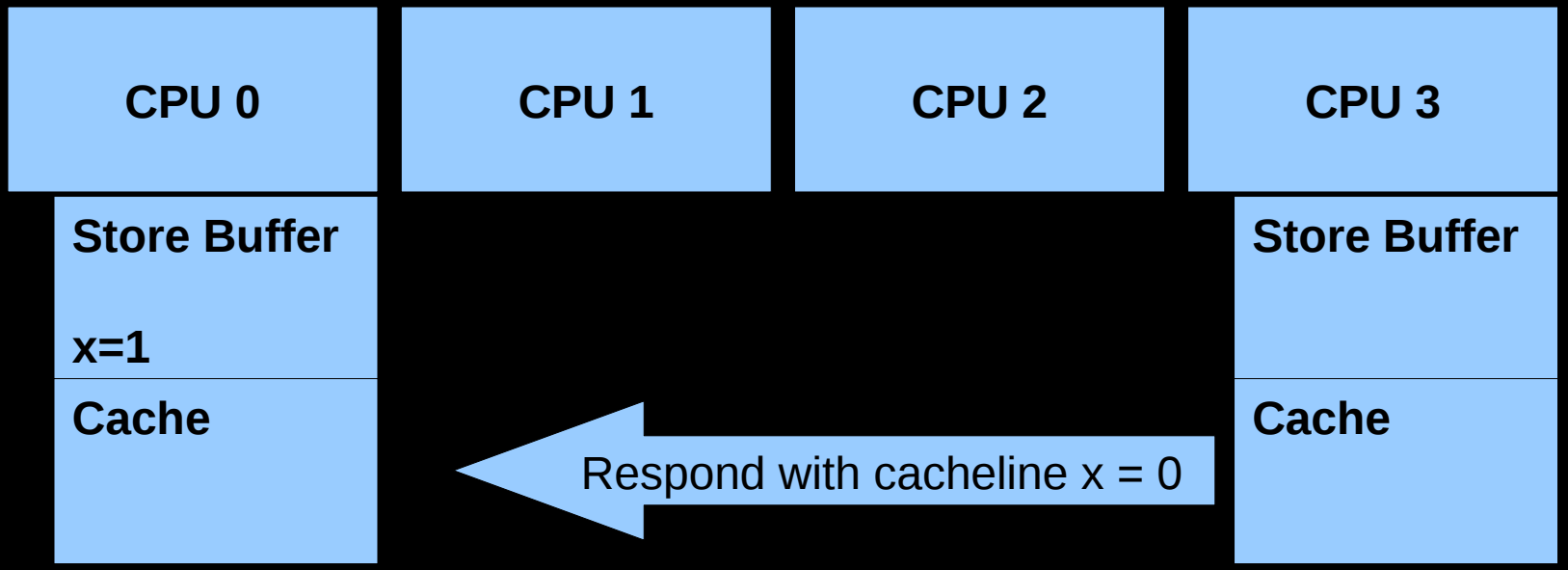
Ordering vs. Time: The From-Reads (fr) Relation Can Also Go Backwards In Time! (4/7)

~~WRITE_ONCE(x, 1)~~ ~~READ_ONCE(x)~~

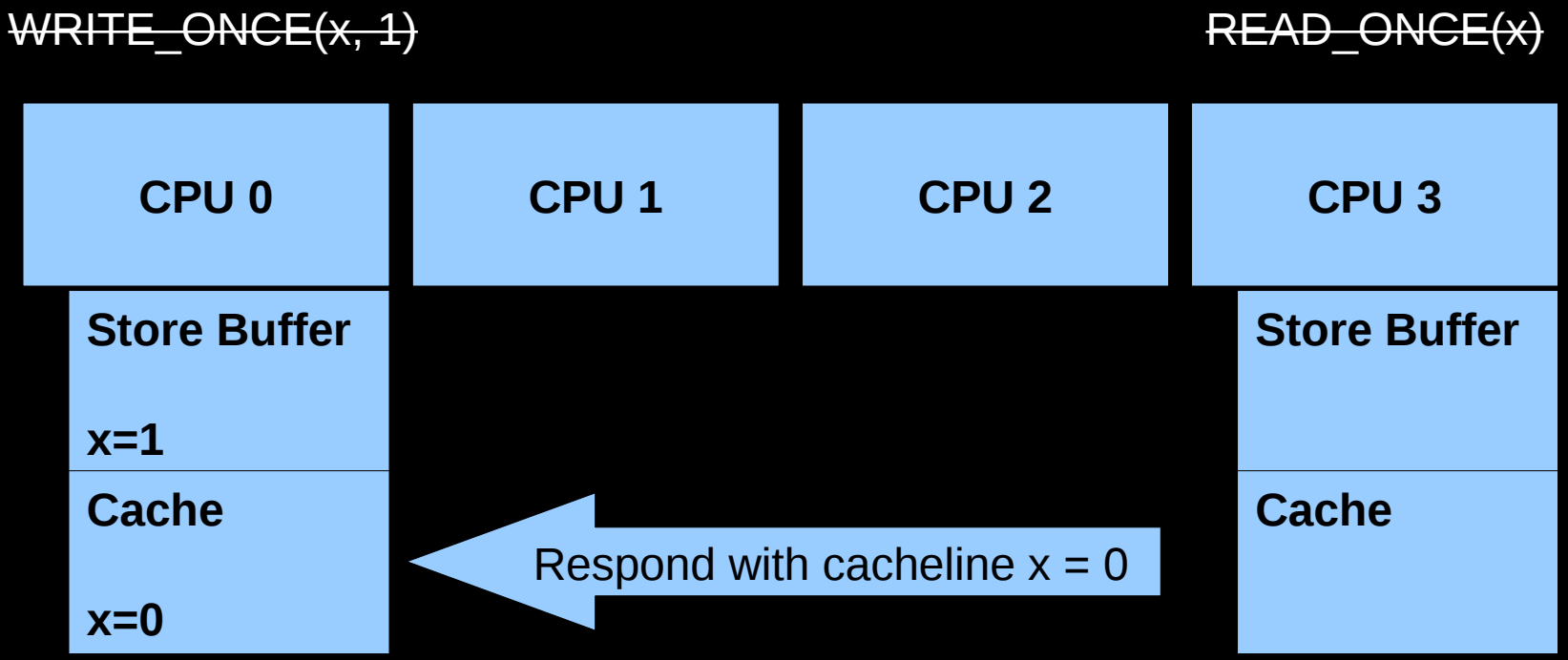


Ordering vs. Time: The From-Reads (fr) Relation Can Also Go Backwards In Time! (5/7)

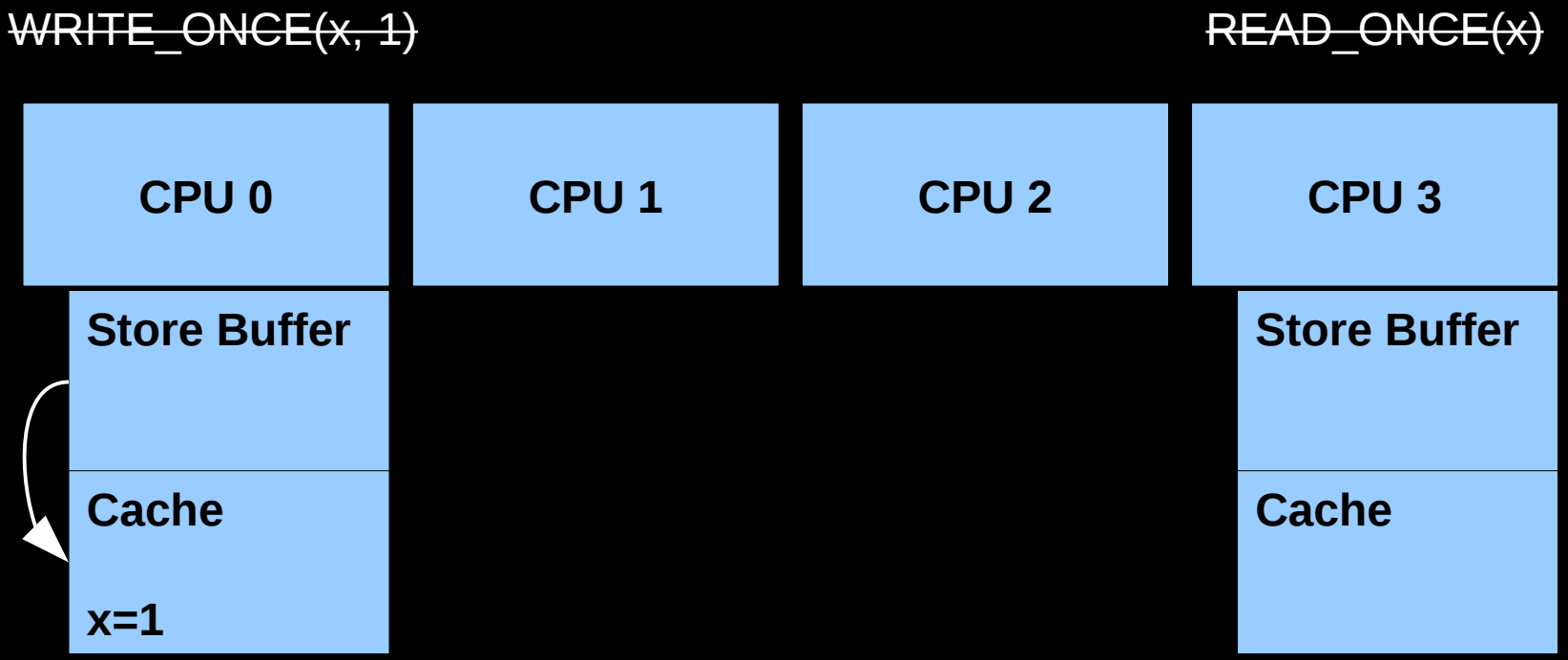
~~WRITE_ONCE(x, 1)~~ ~~READ_ONCE(x)~~



Ordering vs. Time: The From-Reads (fr) Relation Can Also Go Backwards In Time! (6/7)



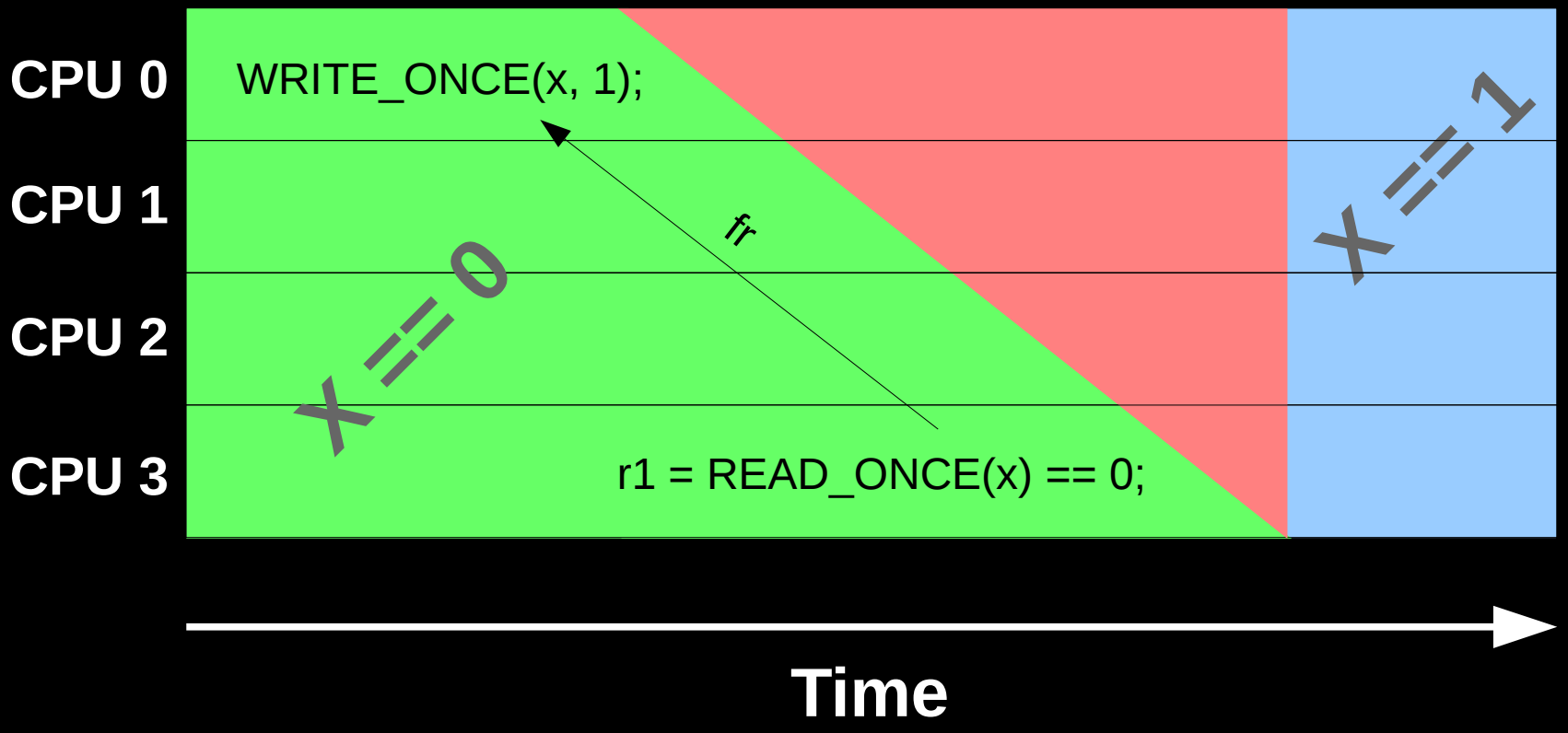
Ordering vs. Time: The From-Reads (fr) Relation Can Also Go Backwards In Time! (7/7)



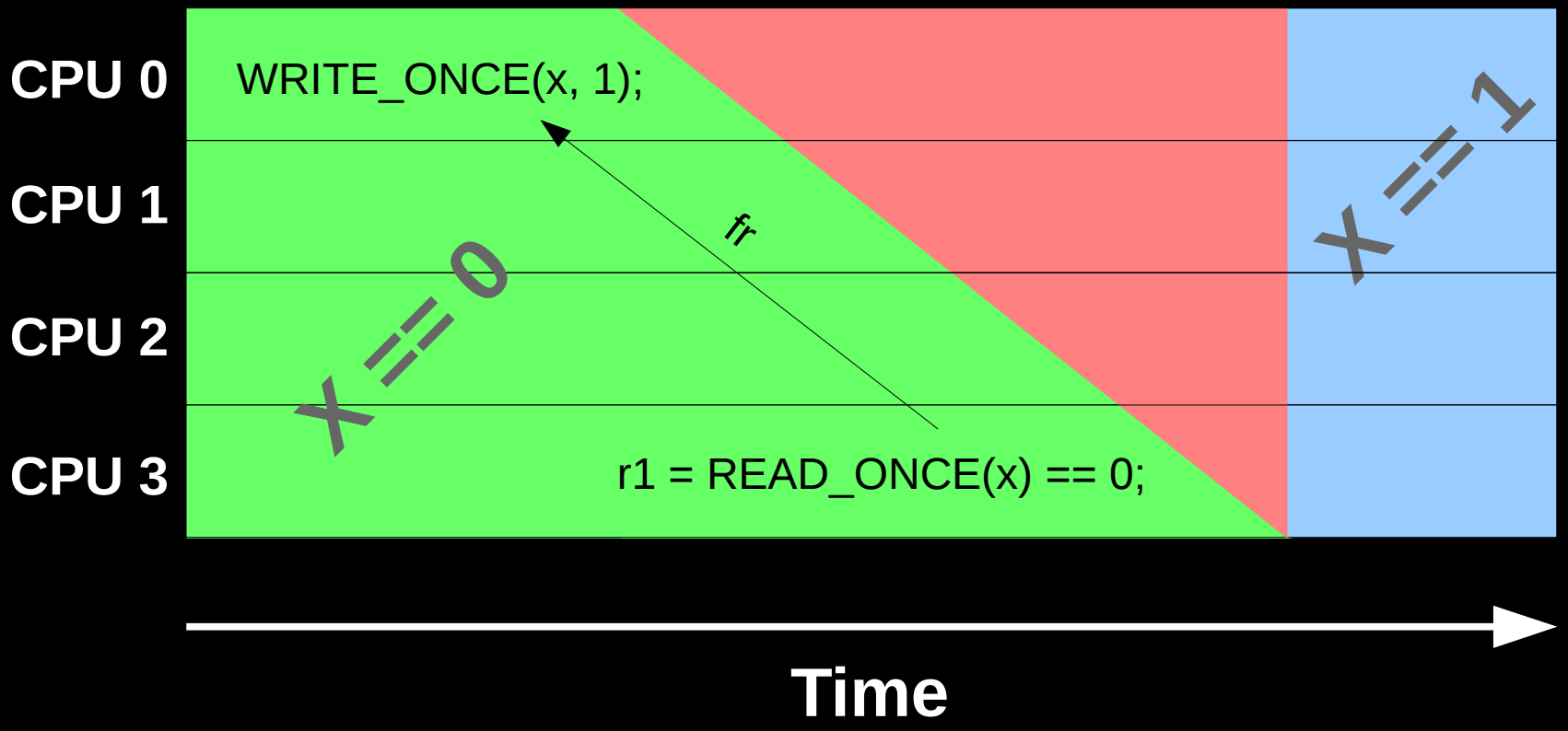
Again, writes are ***not*** instantaneous!

Can't HW Hide Non-Temporal Behavior From Users?

Can't HW Hide Non-Temporal Behavior From Users? Yes, But Not For Free (Many HW Tricks, Though)



Can't HW Hide Non-Temporal Behavior From Users? Yes, But Not For Free (Many HW Tricks, Though)



And some of these HW tricks have consequences: Meltdown & Spectre

Moral: More rf Links, Lighter-Weight Barriers!!!

A Hierarchy of Litmus Tests: Rough Rules of Thumb

- If all rf relations, can use dependencies or acquire
  - Some architecture might someday also require release, so careful!
- If only one relation is non-rf, can use release-acquire
  - Dependencies can ***sometimes*** be used instead of release-acquire
  - But be safe – actually run the model to find out exactly what works!!!
- If two or more relations are non-rf, strong barriers needed
  - ***At least*** one between each non-rf relation
  - But be safe – actually run the model to find out exactly what works!!!

But for full enlightenment, see memory models themselves:

- git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip.git
  - See branch “master” in directory tools/memory-model

A Hierarchy of Memory Ordering: Rough Overheads

- Read-write dependencies:
  - Free everywhere
- Read-read address dependencies:
  - Free other than on DEC Alpha
- Release/acquire chains and read-read control dependencies:
  - Lightweight: Compiler barrier on x86 and mainframe, special instructions on ARM, lightweight isync or lwsync barriers on PowerPC
- Restore sequential consistency:
  - Full memory barriers
    - Expensive pretty much everywhere
    - But usually affect performance more than scalability

How to Run Models

- Download herd tool as part of diy toolset
  - http://diy.inria.fr/sources/index.html
- Build as described in INSTALL.txt
  - Need ocaml v4.01.0 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 relationships
  - linux-kernel.cat: “Cat” file defining actual memory model
  - litmus-tests/*.litmus: Litmus tests

git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip.git branch master in tools/memory-model

A Hierarchy of Litmus Tests (1/3)

- All rf relations and dependencies
  - C-LB+ldref-o+o-ctrl-o+o-dep-o.litmus
- All rf relations but one dependency removed
  - C-LB+ldref-o+o-o+o-dep-o.litmus
- Message passing with read-to-read address dependency
  - C-MP+o-assign+o-dep-o.litmus (Alpha!)
- All rf relations, acquire load instead of one dependency
  - C-LB+ldref-o+acq-o+o-dep-o.litmus

From https://github.com/paulmckrcu/litmus directory manual/demo

A Hierarchy of Litmus Tests (2/3)

- All rf relations, but all dependencies replaced by acquires
  - C-LB+acq-o+acq-o+acq-o.litmus
- One co relation, the rest remain rf relations
  - C-WWC+o+acq-o+acq-o.litmus
- One co, rest remain rf, but with release-acquire
  - C-WWC+o+o-rel+acq-o.litmus
- One co, one fr, and only one remaining rf relation
  - C-Z6.0+o-rel+acq-o+o-mb-o.litmus
- One co, one fr, one rf, and full memory barriers
  - C-Z6.0+o-mb-o+acq-o+o-mb-o.litmus

From https://github.com/paulmckrcu/litmus directory manual/demo

A Hierarchy of Litmus Tests (3/3)

- One co, one fr, one rf, and all but one full memory barriers
  - C-3.SB+o-o+o-mb-o+o-mb-o.litmus
- One co, one fr, one rf, and all full memory barriers
  - C-3.SB+o-mb-o+o-mb-o+o-mb-o.litmus
- IRIW, but with release-acquire
  - C-IRIW+rel+rel+acq-o+acq-o.litmus
- Independent reads of independent writes (IRIW), full barriers
  - C-IRIW+o+o+o-mb-o+o-mb-o.litmus

From https://github.com/paulmckrcu/litmus directory manual/demo

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

- As noted earlier:
  - 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, but can emulate them 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

Summary

Summary

- We have automated much of memory-barriers.txt
  - And more precisely defined much in it!
  - Subject to change, but good set of guiding principles
- First realistic formal Linux-kernel memory model
- First realistic formal memory model including RCU
- Hoped-for benefits:
  - Memory-ordering education tool
  - Core-concurrent-code design aid
  - Ease porting to new hardware and new toolchains
  - Basis for additional concurrency code-analysis tooling
  - Satisfy those asking for it!!!

What Might The Future Hold?

What Might The Future Hold?

- Further validation and refinement of locking
- Modeling spin_is_locked()
  - But we handled spin_unlock_wait() by eliminating it from the kernel...
- Unmarked accesses (as in simple assignment statements)
  - And yes, Jade did exclude these from the initial prototype
  - But locking makes a carefully selected subset of them more plausible, especially in combination with a data-race detector
    - This combination holds future compiler optimizations harmless (we hope!)
    - Note that Jade's first model had neither locks nor data-race detectors
- Changes possibly needed for the RISC-V architecture
  - For example, weak locking primitives (unlock-lock and writes)

To Probe Deeper: Memory Models (1/2)

- “Simulating memory models with herd”, Alglave and Maranget (herd manual)
  - http://diy.inria.fr/tst/doc/herd.html
- “Herding cats: Modelling, Simulation, Testing, and Data-mining for Weak Memory”, Alglave et al.
  - http://www0.cs.ucl.ac.uk/staff/j.alglave/papers/toplas14.pdf
- Download page for herd: http://diy.inria.fr/herd/
- LWN article for herd: http://lwn.net/Articles/608550/ For PPCMEM: http://lwn.net/Articles/470681/
- Lots of Linux-kernel litmus tests: https://github.com/paulmckrcu/litmus
- “A better x86 memory model: x86-TSO”, Owens
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.153.6657&rep=rep1&type=pdf
- “Modelling the ARMv8 Architecture, Operationally: Concurrency and ISA”, Flur et al.
  - http://www.cl.cam.ac.uk/~pes20/popl16-armv8/top.pdf
- “Understanding POWER Multiprocessors”, Sarkar et al.
  - http://www.cl.cam.ac.uk/~pes20/ppc-supplemental/pldi105-sarkar.pdf
- “Synchronising C/C++ and POWER”, Sarkar et al.
  - http://www.cl.cam.ac.uk/~pes20/cppppc-supplemental/pldi010-sarkar.pdf

To Probe Deeper: Memory Models (2/2)

- “A Tutorial Introduction to the ARM and POWER Relaxed Memory Models”, Maranget et al.
  - http://www.cl.cam.ac.uk/~pes20/ppc-supplemental/test7.pdf
- RISC-V Memory Consistency Model Specification (DRAFT), Lustig (editor)
  - https://docs.google.com/viewer?a=v&pid=forums&srcid=MDQwMTcyODgwMjc3MjQxMjA0NzcBMDMwNDk4NTA2OTU5OTIyNTQ3MzcBd0x3TklFcERBd0FKATAuMgFncm91cHMucmlzY3Yub3JnAXYy&authuser=0
- “A Framework for the Investigation of Shared Memory Systems”,Bart Van Assche et al.
  - http://www.bartvanassche.be/publications/2000-csi.pdf
- Lots of relaxed-memory model information: http://www.cl.cam.ac.uk/~pes20/weakmemory/
- “Linux-Kernel Memory Model”, (informal) C++ working paper, McKenney et al.
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0124r2.html
- “A formal kernel memory-ordering model (parts 1 and 2)”, Alglave et al.
  - https://lwn.net/Articles/718628/ and https://lwn.net/Articles/720550/
- “Frightening small children and disconcerting grown-ups: Concurrency in the Linux kernel”, Alglave et al.
  - http://diy.inria.fr/linux/ (To appear in ASPLOS 2018)

To Probe Deeper: RCU

- Desnoyers et al.: “User-Level Implementations of Read-Copy Update”
  - http://www.rdrop.com/users/paulmck/RCU/urcu-main-accepted.2011.08.30a.pdf
  - http://www.computer.org/cms/Computer.org/dl/trans/td/2012/02/extras/ttd2012020375s.pdf
- McKenney et al.: “RCU Usage In the Linux Kernel: One Decade Later”
  - http://rdrop.com/users/paulmck/techreports/survey.2012.09.17a.pdf
  - http://rdrop.com/users/paulmck/techreports/RCUUsage.2013.02.24a.pdf
- McKenney: “Structured deferral: synchronization via procrastination”
  - http://doi.acm.org/10.1145/2483852.2483867
  - McKenney et al.: “User-space RCU” https://lwn.net/Articles/573424/
- McKenney et al: “User-space RCU”
  - https://lwn.net/Articles/573424/
- McKenney: “Requirements for RCU”
  - http://lwn.net/Articles/652156/ http://lwn.net/Articles/652677/ http://lwn.net/Articles/653326/
- McKenney: “Beyond the Issaquah Challenge: High-Performance Scalable Complex Updates”
  - http://www2.rdrop.com/users/paulmck/RCU/Updates.2016.09.19i.CPPCON.pdf
- McKenney, ed.: “Is Parallel Programming Hard, And, If So, What Can You Do About It?”
  - http://kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of their employers.
- 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?