Paul E. McKenney, Meta Platforms Kernel Team

LSF/MM/BPF, May 13, 2024

Instruction-level BPF memory model

© 2024 Meta Platforms

Paul E. McKenney, Meta Platforms Kernel Team
Puranjay Mohan, Kernel Developer
LSF/MM/BPF, May 13, 2024

Instruction-level BPF memory model

© 2024 Meta Platforms

History

- “Towards a BPF Memory Model”, LPC 2021
  - https://lpc.events/event/11/contributions/941/
- Kangrejos 2023 Hallway Track (with Jose Marchesi)
- “Instruction-Level BPF Memory Model”, IETF 118
  - https://datatracker.ietf.org/doc/agenda-118-bpf/
  - https://datatracker.ietf.org/meeting/118/materials/slides-118-bpf-bpf-memory-model-00
- “BPF Memory Model, Two Years On”, LPC 2023
  - https://lpc.events/event/17/contributions/1580/
- “Instruction-Level BPF Memory Model”, living Google Document
  - https://docs.google.com/document/d/1TaSEfWfLnRUi5KqkavUQyL2tThJXYWHS15qcbxIsFb0/edit?usp=sharing

History

- “Towards a BPF Memory Model”, LPC 2021
  - https://lpc.events/event/11/contributions/941/
- Kangrejos 2023 Hallway Track (with Jose Marchesi)
- “Instruction-Level BPF Memory Model”, IETF 118
  - https://datatracker.ietf.org/doc/agenda-118-bpf/
  - https://datatracker.ietf.org/meeting/118/ma … pf-bpf-memory-model-00
- “BPF Memory Model, Two Years O…
  - https://lpc.events/event/17/co…
- “Instruction-Level BP… …ng Google Document
  - https://docs.goo… …ument/d… …aSEfWfLnRUi5KqkavUQyL2tThJXYWHS15qcbxIsFb0/edit?usp=sharing

What more could possibly be needed???

Where Is the BPF Memory Model?

- Overall direction set in 2021
- Informal instruction-level ordering in late 2023
- We still need:
  - Formal definition and tools
    - Including comparison against hardware models
  - An official IETF standard for BPF memory model

Review of Informal Model

Review of Informal Model

- BPF Atomic Instructions
- BPF Conditional Jump Instructions
- BPF Load instructions
- BPF Memory-Reference Instructions

BPF Atomic Instructions

- `BPF_XCHG`, `BPF_CMPXCHG`
- `BPF_ADD`, `BPF_OR`, `BPF_AND`, `BPF_XOR`
- `BPF_FETCH` with one of the above

BPF Atomic Instructions 1/3

- `BPF_XCHG` and `BPF_CMPXCHG` instructions are fully ordered
- All CPUs and tasks agree that all instructions preceding or following a given `BPF_XCHG` or `BPF_CMPXCHG` instruction are ordered before or after, respectively, that same instruction
  - Consistent with Linux-kernel `atomic_xchg()` and `atomic_cmpxchg()`, respectively
  - Alternatively, consistent with the following:
    - `smp_mb(); atomic_cmpxchg_relaxed(); smp_mb();`

BPF Atomic Instructions 2/3

- BPF_ADD, BPF_OR, BPF_AND, BPF_XOR instructions are unordered
- CPUs and JITs can reorder these instructions freely
  - Consistent with Linux-kernel atomic_add(), atomic_or(), atomic_and(), and atomic_xor() APIs

BPF Atomic Instructions 3/3

- When accompanied by BPF_FETCH, BPF_ADD, BPF_OR, BPF_AND, BPF_XOR instructions are fully ordered
- All CPUs and tasks agree that all instructions preceding or following a given instruction adorned with BPF_FETCH are ordered before or after, respectively, that same instruction
  - Consistent with Linux-kernel atomic_fetch_add(), atomic_fetch_or(), atomic_fetch_and(), and atomic_fetch_xor() APIs

BPF Conditional Jump Instructions

- Modifiers to BPF_JMP32 and BPF_JMP instructions:
  - BPF_JEQ, BPF_JGT, BPF_JGE, BPF_JSET, BPF_JNE, BPF_JSGT, BPF_JSGE, BPF_JLT, BPF_JLE, BPF_JSLT, and BPF_JSLE
- Unconditional jump instructions (BPF_JA, BPF_CALL, BPF_EXIT) provide no memory-ordering semantics

BPF Conditional Jump Instructions

- These modifiers to BPF_JMP32 and BPF_JMP instructions provide weak ordering:
  - `BPF_JEQ`, `BPF_JGT`, `BPF_JGE`, `BPF_JSET`, `BPF_JNE`, `BPF_JSGT`, `BPF_JSGE`, `BPF_JLT`, `BPF_JLE`, `BPF_JSLT`, and `BPF_JSLE`
- Too-smart JITs might need to be careful

BPF Conditional Jump Instructions

- This weak ordering applies when:
  - Either the src or `dst` registers depend on a prior load instruction (`BPF_LD` or `BPF_LDX`), ***and***
  - There is a store instruction (`BPF_ST` or `BPF_STX`) *before control flow converges*, ***and***
  - The restrictions outlined in the "`CONTROL DEPENDENCIES`" section of `Documentation/memory-barriers.txt` are faithfully followed
    - Compilers do not understand control dependencies, and happily break them.
    - Optimizations involving conditional-move instructions requires the "before control flow converges" restriction

BPF Conditional Jump Instructions

- This weak ordering applies when:
  - Either the src or `dst` registers depend on a prior load instruction (`BPF_LD` or `BPF_LDX`), ***and***
  - There is a store instruction (`BPF_ST` or `BPF_STX`) ~~***before control flow converges, and***~~ following the conditional jump instruction in program order
  - ~~**The restrictions outlined in the "`CONTROL DEPENDENCIES`" section of `Documentation/memory-barriers.txt` are faithfully followed**~~
    - ~~**Compilers do not understand control dependencies, and happily break them.**~~
    - ~~**Optimizations involving conditional-move instructions requires the "before control flow converges" restriction**~~

BPF Conditional Jump Instructions

- This weak ordering applies when:
  - Either the src or `dst` registers depend on a pri[illegible]D or BPF_LDX), ***and***
  - There is a store instruction (BP[illegible] ~~re control flow converges, and~~ follo[illegible] instruction in program order
  - ~~The restricti[illegible] NTROL DEPENDENCIES" section of Docu[illegible] arriers.txt are faithfully followed~~
    - ~~[illegible] derstand control dependencies, and happily break them.~~
    - ~~[illegible] ns involving conditional-move instructions requires the "before [illegible] ol flow converges" restriction~~

Running BPF assembly through an optimizing compiler requires some care

Sometimes Translation is Required

Sometimes Translation is Required

- Paul E. McKenney’s original:
  - “Running BPF assembly through an optimizing compiler requires some care”

Sometimes Translation is Required

- Paul E. McKenney’s original:
  - “Running BPF assembly through an optimizing compiler requires some care”
- Alexei Starovoitov’s translation:
  - “Don’t run BPF assembly through an optimizing compiler”

BPF Conditional Jump Instructions

- This weak ordering applies when:
  - Either … `dst` registers depend on a pri… D or BPF_LDX), …
  - There is a store i… ~~*…re control flow converges, and*~~ … instruction in program order
  - ~~**The restricti… NDENCIES" section of Docu… followed**~~
    - ~~**…nderstand control depe… and … them.**~~
    - ~~**…ns involving conditional-move instructions requires the… efore …ol flow converges" restriction**~~

LKMM control dependencies need some "goto" love...

Running B… …embly through an optimizing comp… …uires some care

Towards a Formal BPF Model

Goals of Formal BPF Memory Model

- Simple hardware-level model
- Consistent with LKMM
  - Prefer to avoid forbidding reorderings LKMM allows
- Low-overhead mappings to supported hardware
  - BPF should avoid forbidding reorderings allowed by ARMv8, PowerPC, RISC-V, x86, …
- Ability to grow as BPF instruction set grows

But Already Have Formal LKMM!!!

- Defines BPF limits of weakness
- Additional functionality can be excluded
  - Restrict the linux-kernel.def file (next slide)
- Just map from BPF assembly to LKMM C
  - Simple script!!!

Restrict `linux-kernel.defs` file

- Make a linux-bpf.defs file for use with the existing linux-kernel.{bell,cat} files:

```
xchg(X,V)  __xchg{mb}(X,V)
cmpxchg(X,V,W) __cmpxchg{mb}(X,V,W)
spin_lock(X) { __lock(X); }
spin_unlock(X) { __unlock(X); }
spin_trylock(X) __trylock(X)
spin_is_locked(X) __islocked(X)
atomic_add(V,X) { __atomic_op(X,+,V); }
atomic_and(V,X) { __atomic_op(X,&,V); }
atomic_or(V,X)  { __atomic_op(X,|,V); }
atomic_xor(V,X) { __atomic_op(X,^,V); }
atomic_fetch_add(V,X) __atomic_fetch_op{mb}(X,+,V)
atomic_fetch_and(V,X) __atomic_fetch_op{mb}(X,&,V)
atomic_fetch_or(V,X) __atomic_fetch_op{mb}(X,|,V)
atomic_fetch_xor(V,X) __atomic_fetch_op{mb}(X,^,V)
atomic_xchg(X,V) __xchg{mb}(X,V)
atomic_cmpxchg(X,V,W) __cmpxchg{mb}(X,V,W)
```

Restrict `linux-kernel.defs` File

- Make a linux-bpf.defs file for use with the existing linux-kernel.{bell,cat} files:

```
xchg(X,V)  __xchg{mb}(X,V)
cmpxchg(X,V,W) __cmpxchg{mb}(X,V,W)
spin_lock(X) { __lock(X); }
spin_unlock(X) { __unlock(X); }
spin_trylock(X) __trylock(X)
spin_is_locked(X) __islocked(X)
atomic_add(V,X) { __atomic_op(X,+,V); }
atomic_and(V,X) { __atomic_op(X,&,V); }
atomic_or(V,X)  { __atomic_op(X,|,V); }
atomic_xor(V,X) { __atomic_op(X,^,V); }
atomic_fetch_add(V,X) __atomic_fetch_op{
atomic_fetch_and(V,X) __atomic_fetc ,V)
atomic_fetch_or(V,X) __atomic_f ,|,V)
atomic_fetch_xor(V,X) __at {mb}(X,^,V)
atomic_xchg(X,V) __xch
atomic_cmpxchg(X,V,W) _ g{mb}(X,V,W)
```

Just deleted a bunch of lines!!!

Restrict `linux-kernel.defs` File

- Make a linux-bpf.defs file for use with the existing linux-kernel.{bell,cat} files:

```
xchg(X,V)  __xchg{mb}(X,V)
cmpxchg(X,V,W) __cmpxchg{mb}(X,[illegible]
spin_lock(X) { __lock(X); }
spin_unlock(X) { __unlock(X); }
spin_trylock(X) __trylock(X)
spin_is_locked(X) __islocked(X)
atomic_add(V,X) { __atomic_op(X,+,V); }
atomic_and(V,X) { __atomic_op(X,&,V); }
atomic_or(V,X)  { __atomic_op(X,|,V); }
atomic_xor(V,X) { __atomic_op(X,^,V); }
atomic_fetch_add(V,X) __atomic_fetch_op[illegible]
atomic_fetch_and(V,X) __atomic_fetc[illegible],V)
atomic_fetch_or(V,X) __atomic_f[illegible](X,|,V)
atomic_fetch_xor(V,X) __at[illegible]{mb}(X,^,V)
atomic_xchg(X,V) __xch[illegible]
atomic_cmpxchg(X,V,W) [illegible]g{mb}(X,V,W)
```

If only it were that easy...

Just deleted a [illegible]nch of lines!!!

Map From BPF to LKMM C Code

- The herd7 event structures are different for C code and assembly code (of any type)
- Assembly has constraints
  - For example, R0 is special for BPF_CMPXCHG
- Pitfalls converting branches into “if”/”while”

Map From BPF to LKMM C Code

- The herd7 event structures are diff[illegible]nt for C code and assembly code (of [illegible] type)
- Assembly has constrai[illegible]
  - For example, R0 i[illegible]ecial for BPF_CMPXCHG
- Pitfalls con[illegible]g branches into “if”/”while”

Need hardware memory model

Map From BPF to LKMM C Code

- The herd ... ent structures are diff... nt for C code and ass... bly code (of ... type)
- Assembly has con...
  - For example, R0 ... ecia... BPF_CMPXCHG
- Pitfalls conv... g branches int... f"/"while"

But LKMM useful debugging aid

Need hardware memory model

Map From BPF to LKMM C Code

- The her[...]ent structures are diff[...]nt for C code and ass[...]bly code (of [...] type)
- Assembly has [...]
  - For example, R0 [...] BPF_CMPXCHG
- Pitfalls con[...]g branches int[...]f"/"while"

But LKMM uses memory model

Need hardware debugging aid

Or vice versa!

Just Use Hardware Memory Model!

Just Use Hardware Memory Model!

- X86 is too strong
  - We don’t want BPF JITs to emit memory-barrier instructions after every conditional branch on ARMv8 and PowerPC
- PowerPC is not actively developed
  - Also larx/stcx instead of atomic instructions
- ARMv8?

ARMv8 Memory Model

- Actively developed and maintained
- Well designed (once you understand it!)
- Fully featured (e.g., mixed sizes)
  - Including load-acquire/store-release
- Includes irrelevant hardware features
- Stronger than PowerPC and 32-bit ARM

ARMv8 Memory Model

- Actively developed and maintained
- Well designed (once you u[illegible]stand it!)
- Fully featured (e.g., m[illegible]d sizes)
  - Including load-a[illegible]re/store-release
- Includes irre[illegible]vant hardware features
- Stronger than PowerPC and 32-bit ARM

Worth looking into

The ARMv8 Memory Model

How to Weaken ARMv8 As Needed?

- Review ARM's AArch64 Application Level Memory Model (ARM DDI 0487J.a ID042523), section B2.3 (31 pages)
  - Remove anything PowerPC cannot order
  - Remove other-multicopy atomicity
  - Other issues hidden by lack of BPF weak barriers

How to Weaken ARMv8 As Needed?

- Review ARM's AArch64 Application L… Memory Model (ARM DDI 0487… 23), section B2.3 (31 pages)
  - Remove anyth… not order
  - Rem… py atomicity
  - … hidden by lack of BPF weak barriers

The following slides review a couple of the more entertaining examples

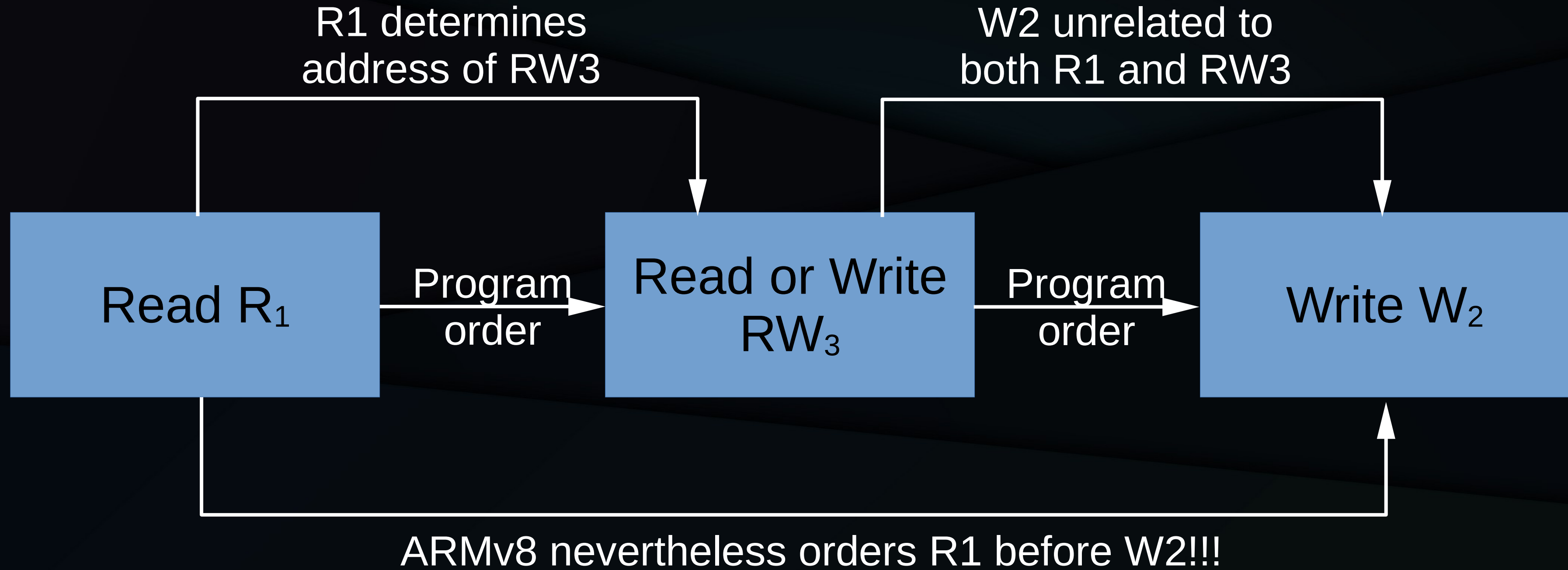
Example 1: Dependency Ordered Before?

Dependency Ordered Before?

Disturbing example from ARMv8:

- **Dependency-ordered-before**: A dependency creates externally-visible order between a Read Memory effect and another Memory effect generated by the same Observer. A Read Memory effect R_1 is Dependency-ordered-before a Read or Write Memory effect RW_2 from the same Observer if R_1 appears in program order before RW_2 and any of the following cases apply:
  - ...
  - RW_2 is a Write Memory effect W_2 that appears in program order after an Explicit Read or Write Memory effect RW_3 and there is an Address dependency from R_1 to RW_3.
  - ...

Dependency Ordered Before?



Dependency Ordered Before?

- Example Linux-kernel code fragment:

```
r1 = rcu_dereference(gp);
// r1 is a pointer to an array
WRITE_ONCE(r1[i].value, 42);
WRITE_ONCE(x, “This is a test”);

// Write to x ordered after read from gp???
```

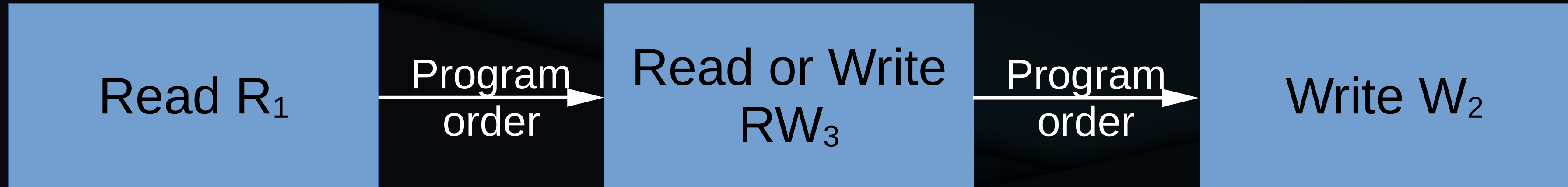
Dependency Ordered Before?

- Example Linux-kernel code fragme

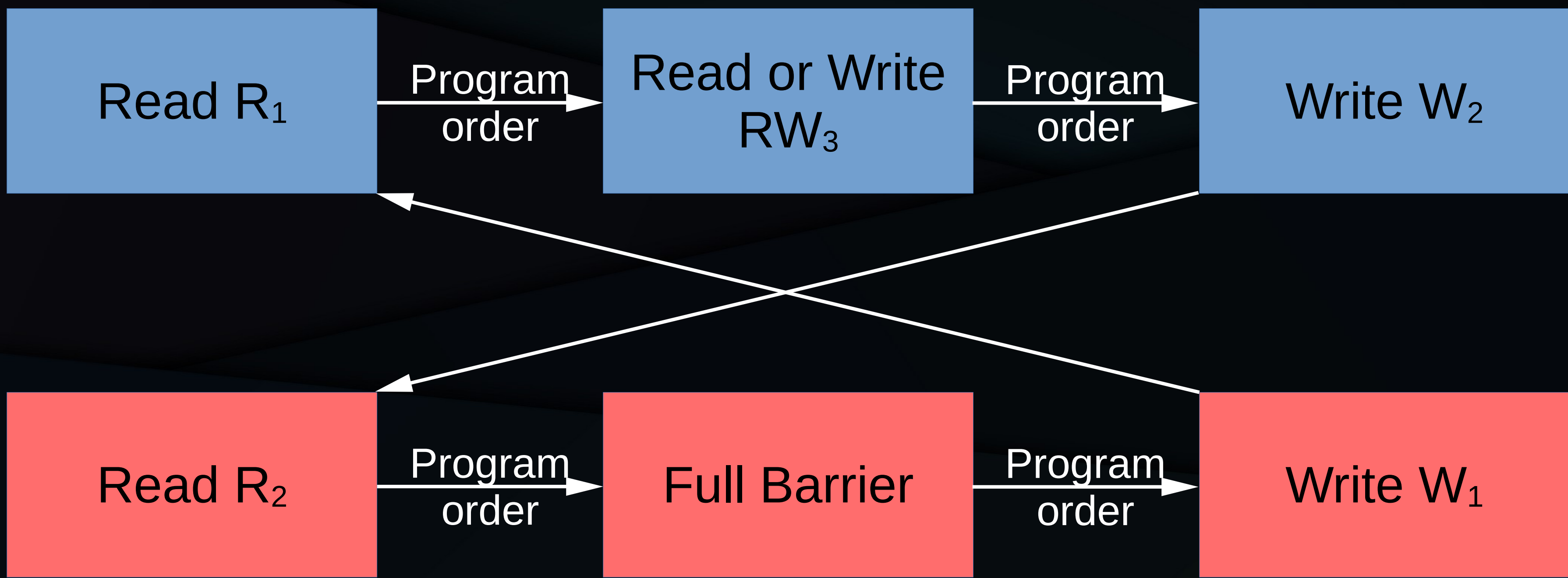
```
r1 = rcu_dereference(gp);
// r1 is a pointer to        ray
WRITE_ONCE(r1[i]       , 42);
WRITE_ONCE(x      is is a test”);
// Wri        x ordered after read from gp???
```

A question for a memory model!!!

Check Dependency Ordered Before



Check Dependency Ordered Before



PPC Dependency Ordered Before?

```
PPC ARMv8DOB5-PPC
{
0:r2=x; 0:r4=y; 0:r6=z;
1:r2=x; 1:r4=y; 1:r6=z;
}
 P0             | P1             ;
 li r1,1        | li r1,1        ;
 lwz r3,0(r2)   | lwz r3,0(r6)   ;
 xor r5,r3,r3   | sync           ;
 add r4,r5,r4   | stw r1,0(r2)   ;
 stw r1,0(r4)   |                ;
 stw r1,0(r6)   |                ;
locations [x;y;z]
exists (0:r3=1 /\ 1:r3=1)
```

PPC Dependency Ordered Before?

```
$ herd7 ARMv8DOB5-PPC.litmus
Test ARMv8DOB5-PPC Allowed
States 3
0:r3=0; 1:r3=0; [x]=1; [y]=1; [z]=1;
0:r3=0; 1:r3=1; [x]=1; [y]=1; [z]=1;
0:r3=1; 1:r3=0; [x]=1; [y]=1; [z]=1;
No
Witnesses
Positive: 0 Negative: 3
Condition exists (0:r3=1 /\ 1:r3=1)
Observation ARMv8DOB5-PPC Never 0 3
Time ARMv8DOB5-PPC 0.01
Hash=94585bab1e0261eb46eac418ba00b2f5
```

PPC Dependency Ordered Before?

```
$ herd7 ARMv8DOB5-PPC.litmus
Test ARMv8DOB5-PPC Allowed
States 3
0:r3=0; 1:r3=0; [x]=1; [y]=1; [z]=1;
0:r3=0; 1:r3=1; [x]=1; [y]=1; [z]
0:r3=1; 1:r3=0; [x]=1; [y]=1;
No
Witnesses
Positive: 0 Negativ
Condition exist 3=1 /\ 1:r3=1)
Observation ARMv DOB5-PPC Never 0 3
Time ARMv8DOB5-PPC 0.01
Hash=94585bab1e0261eb46eac418ba00b2f5
```

PowerPC orders this!!!

LKMM Dependency Ordered Before?

```
C ARMv8DOB5-LKMM

{
  x=y;
}

P0(int *u, int *x, int *y, int *z)
{
  int r1 = READ_ONCE(*x);
  WRITE_ONCE(*r1, 1);
  WRITE_ONCE(*z, 1);
}
```

```
P1(int *u, int *x, int *y, int *z)
{
        int r1 = READ_ONCE(*z);
        smp_mb();
        WRITE_ONCE(*x, u);
}

locations [x;y;z]
exists (0:r1=u /\ 1:r1=1)
```

LKMM Dependency Ordered Before?

```
$ herd7 -conf linux-kernel.cfg ARMv8DOB5-LKMM.litmus
Test ARMv8DOB5-LKMM Allowed
States 4
0:r1=u; 1:r1=0; [x]=u; [y]=0; [z]=1;
0:r1=u; 1:r1=1; [x]=u; [y]=0; [z]=1;
0:r1=y; 1:r1=0; [x]=u; [y]=1; [z]=1;
0:r1=y; 1:r1=1; [x]=u; [y]=1; [z]=1;
Ok
Witnesses
Positive: 1 Negative: 3
Condition exists (0:r1=u /\ 1:r1=1)
Observation ARMv8DOB5-LKMM Sometimes 1 3
Time ARMv8DOB5-LKMM 0.01
Hash=0bb0204225e06470ad922579dd92f14c
```

LKMM Dependency Ordered Before?

```
$ herd7 -conf linux-kernel.cfg ARMv8DOB5-LKMM.litmus
Test ARMv8DOB5-LKMM Allowed
States 4
0:r1=u; 1:r1=0; [x]=u; [y]=0; [z]=1;
0:r1=u; 1:r1=1; [x]=u; [y]=0; [z]=1;
0:r1=y; 1:r1=0; [x]=u; [y]=1; [z]=1;
0:r1=y; 1:r1=1; [x]=u; [y]=1; [z]=1;
Ok
Witnesses
Positive: 1 Negative: 3
Condition exists (0:r1=[illegible]=1)
Observation ARMv8DOB[illegible]ometimes 1 3
Time ARMv8DOB5-LK[illegible]
Hash=0bb0204225e06[illegible]ad922579dd92f14c
```

LKMM does not order this!!!

LKMM Dependency Ordered Before?

```
$ herd7 -conf linux-kernel.cfg ARMv8DOB5-LKMM.litmus
Test ARMv8DOB5-LKMM A    ed
States 4
0:r1=u; 1:r1=0; [x]=u;         ]=1;
0:r1=u; 1:r1=1; [x]=u; [y]=
0:r1=y; 1:r1=0; [x]=u; [y]=1;
0:r1=y; 1:r1=1; [x]=u; [y]=1; [z]=
Ok
Witnesses
Positive: 1 Negative: 3
Condition exists (0:r1=       =1)
Observation ARMv8DO        ometimes 1 3
Time ARMv8DOB5-LK
Hash=0bb0204225e0     ad922579dd92f14c
```

What should order this!!!

LKMM does BPF do???

LKMM Dependency Ordered Before?

```
C ARMv8DOB5-LKMM

{
  x=y;
}

P0(int *u, int *x, int *y, int *z)
{
  int r1 = READ_ONCE(*x);
  WRITE_ONCE(*r1, 1);
  WRITE_ONCE(*z, 1);
}
```

```
P1(int *u, int *x, int *y, int *z)
{
        int r1 = READ_ONCE(*z);
        smp_mb();
        WRITE_ONCE(*x, u);
}

locations [x;y;z]
exists (0:r1=u /\ 1:r1=1)
```

BPF Dependency Ordered Before?

- Why do ARMv8 and PowerPC order stores?

BPF Dependency Ordered Before?

- Why do ARMv8 and PowerPC order stores?
  - They order accesses to a single location

BPF Dependency Ordered Before?

- Why do ARMv8 and PowerPC order stores?
  - They order accesses to a single location
  - And before that load completes, the hardware has no idea whether the two stores are to the same location!!!

BPF Dependency Ordered Before?

- Why do ARMv8 and PowerPC order stores?
  - They order accesses to a single location
  - And before that load completes, the hardware has no idea whether the two stores are to the same location!!!
- Why doesn't LKMM order these stores???

BPF Dependency Ordered Before?

- Why do ARMv8 and PowerPC order stores?
  - They order accesses to a single location
  - And before that load completes, the hardware has no idea whether the two stores are to the same location!!!
- Why doesn't LKMM order these stores???
  - LKMM knows the full execution a priori

BPF Dependency Ordered Before?

- Why do ARMv8 and PowerPC or… es?
  - They order accesses to a si…
  - And before that load c… e hardware has no idea whether th… s are to the same location!!!
- Why do… order these stores???
  - LKM… s the full execution a priori

BPF has no a priori knowledge, & thus should follow PPC & ARM

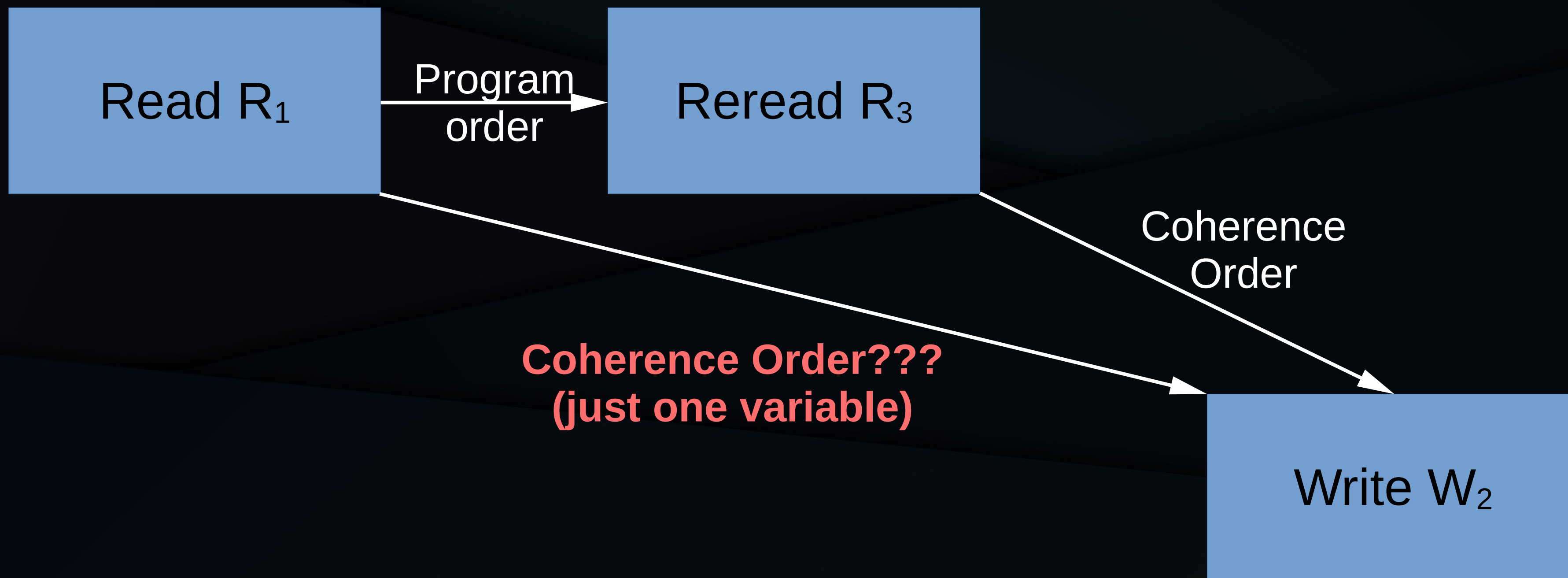
Example 2: Hazard Ordered Before?

Hazard Ordered Before?

Another disturbing example from ARMv8:

- **Hazard-ordered-before**: An Effect E_1 is Hazard-ordered-before an effect E_2 if all of the following apply:
  - E_1 is an Explicit Read Memory effect R_1.
  - R_1 appears in program-order before an Explicit Read Memory effect R_3
  - R_1 and R_3 access the same Location.
  - R_1 and E_2 are from different Observers.
  - R_3 is Coherence-before E_2.
  - E_2 is an Explicit Write Memory effect W_2.

Check Hazard Ordered Before???



Hazard Ordered Before?

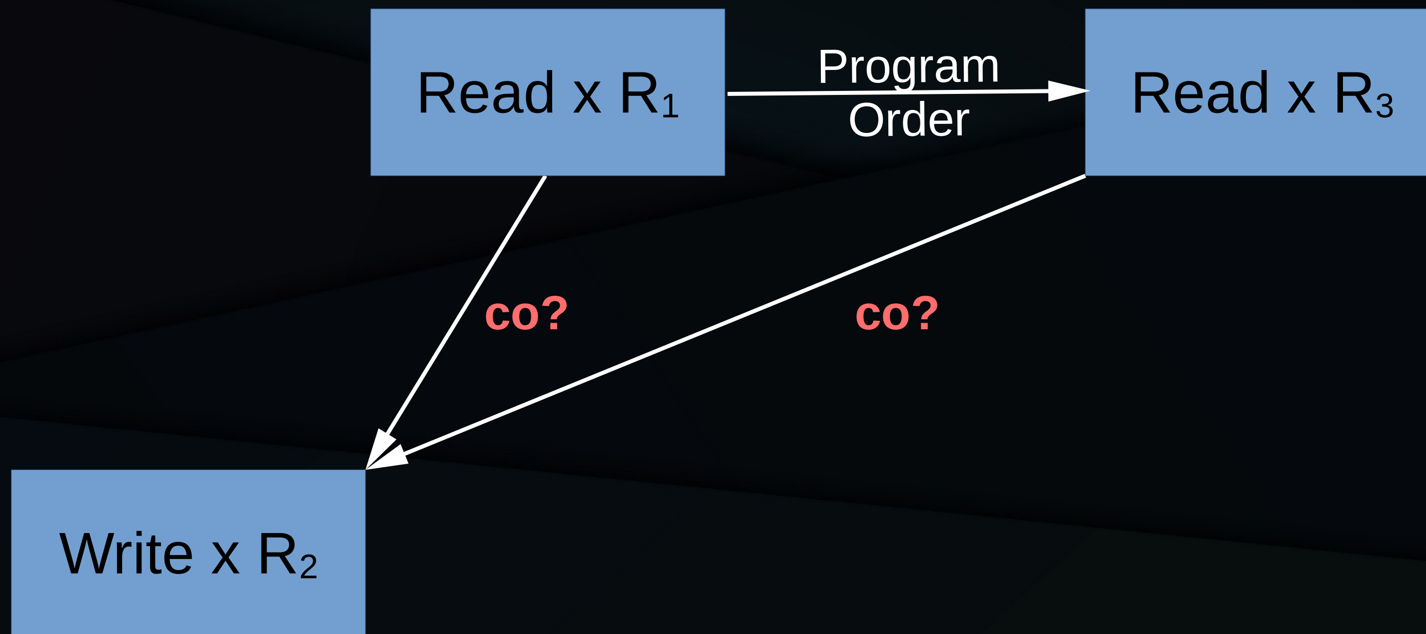
- Example Linux-kernel code fragment:
  - Thread 0:

```
r1 = READ_ONCE(x); // This cannot return 1!!!
r2 = READ_ONCE(x); // Returns 0
```

  - Thread 1:

```
WRITE_ONCE(x, 1);
```

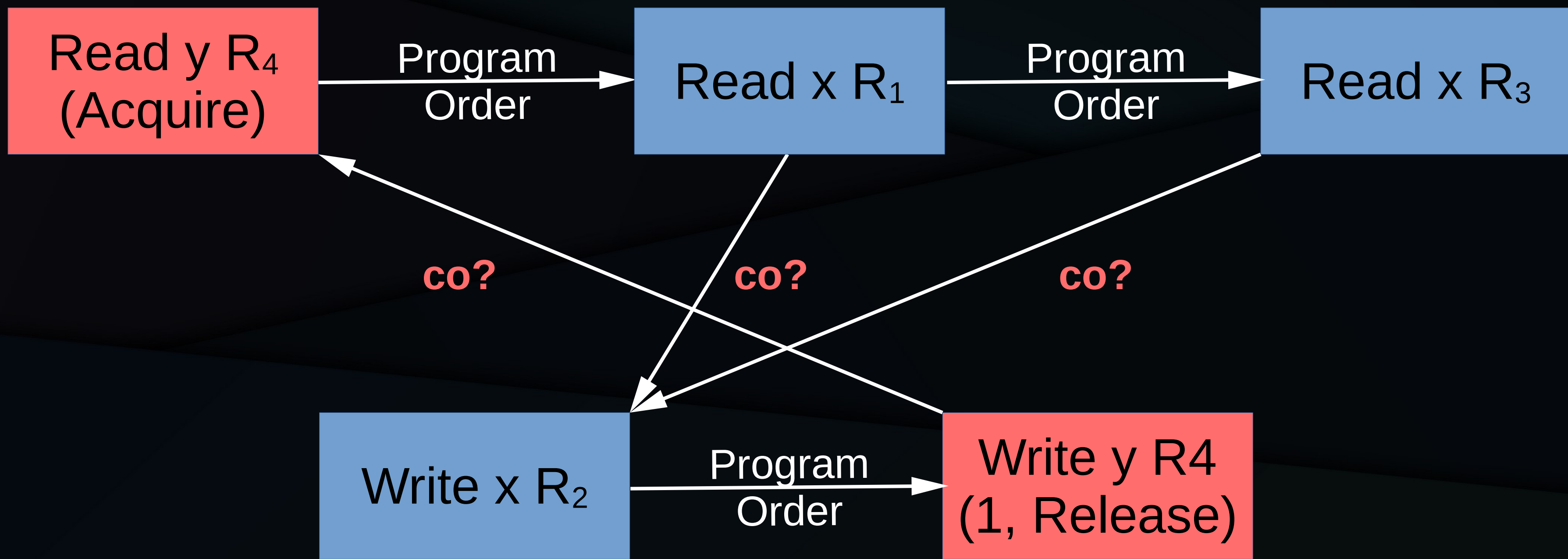
Check Hazard Ordered Before???



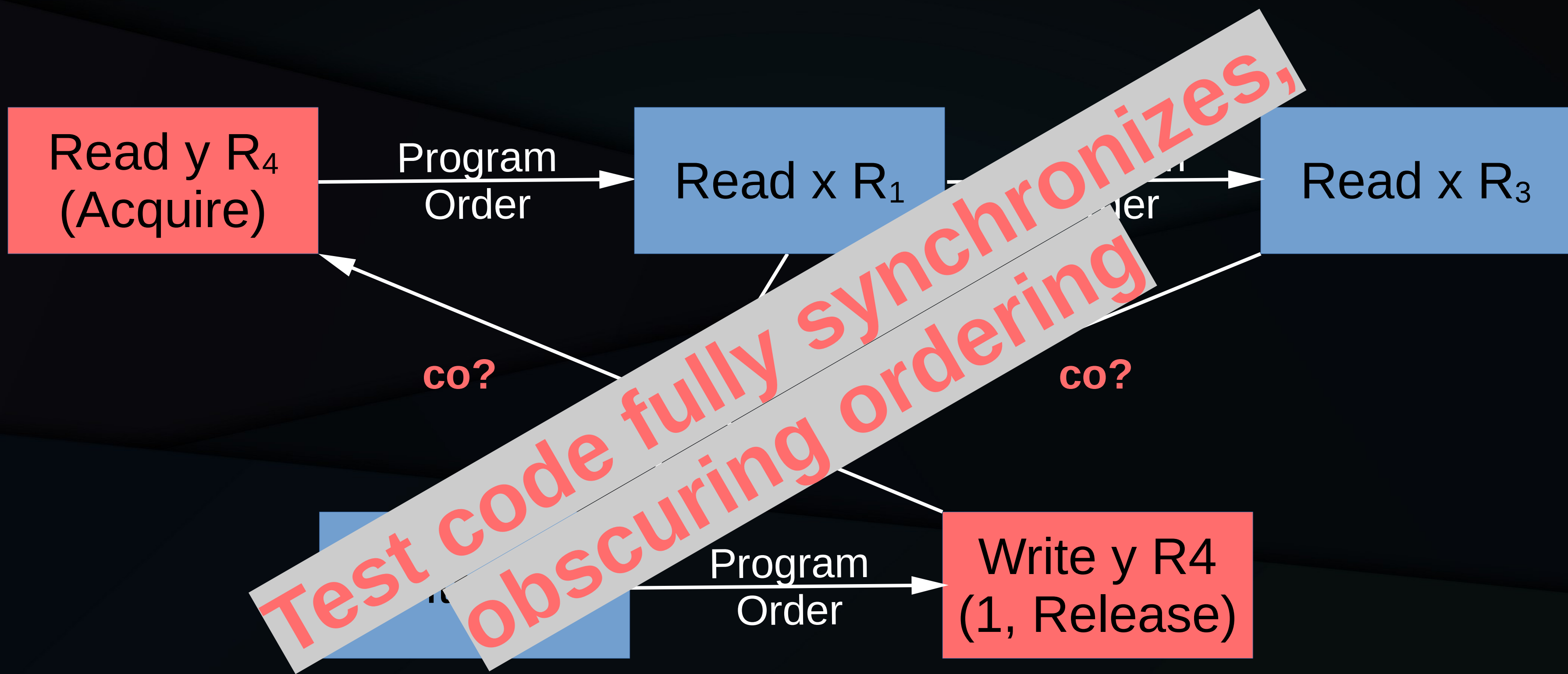
Check Hazard Ordered Before???

Clearly need something before reads and after write, but what?

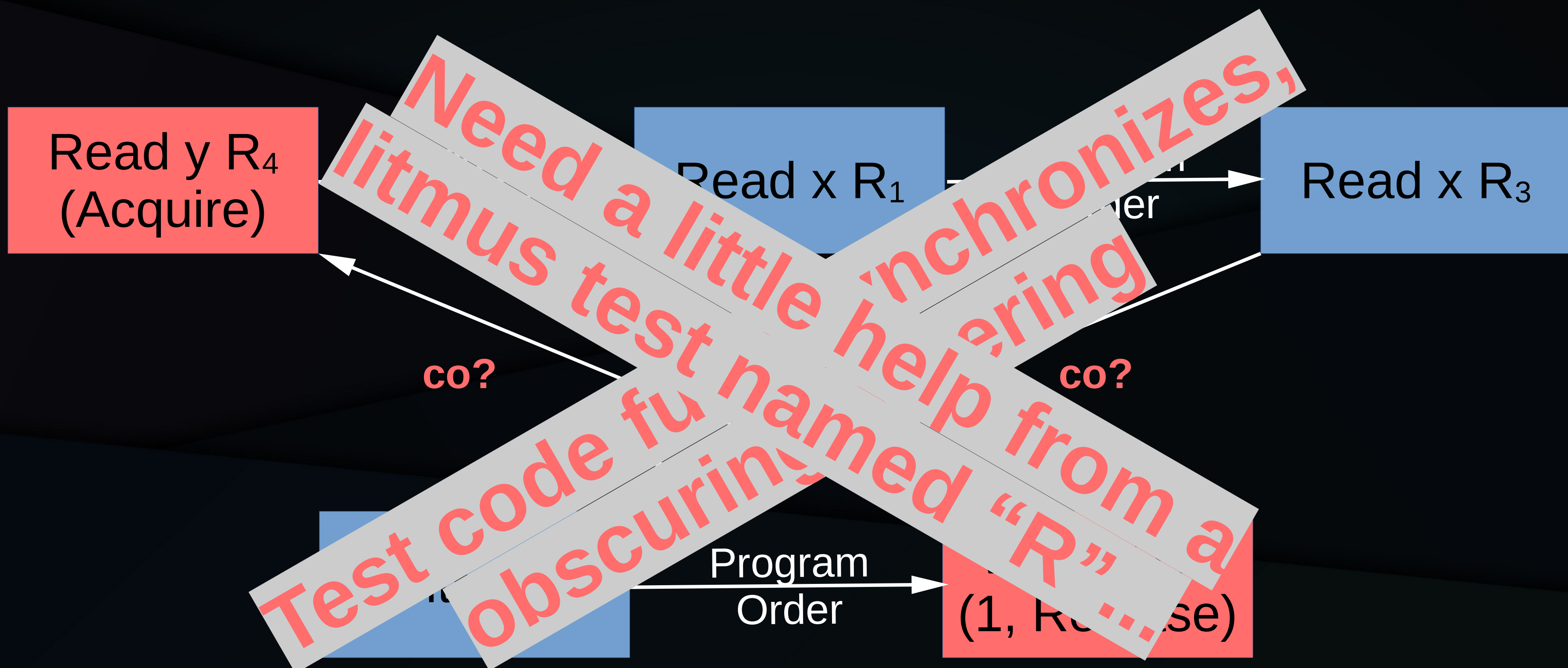
Use Light-Weight Reads-From Link?



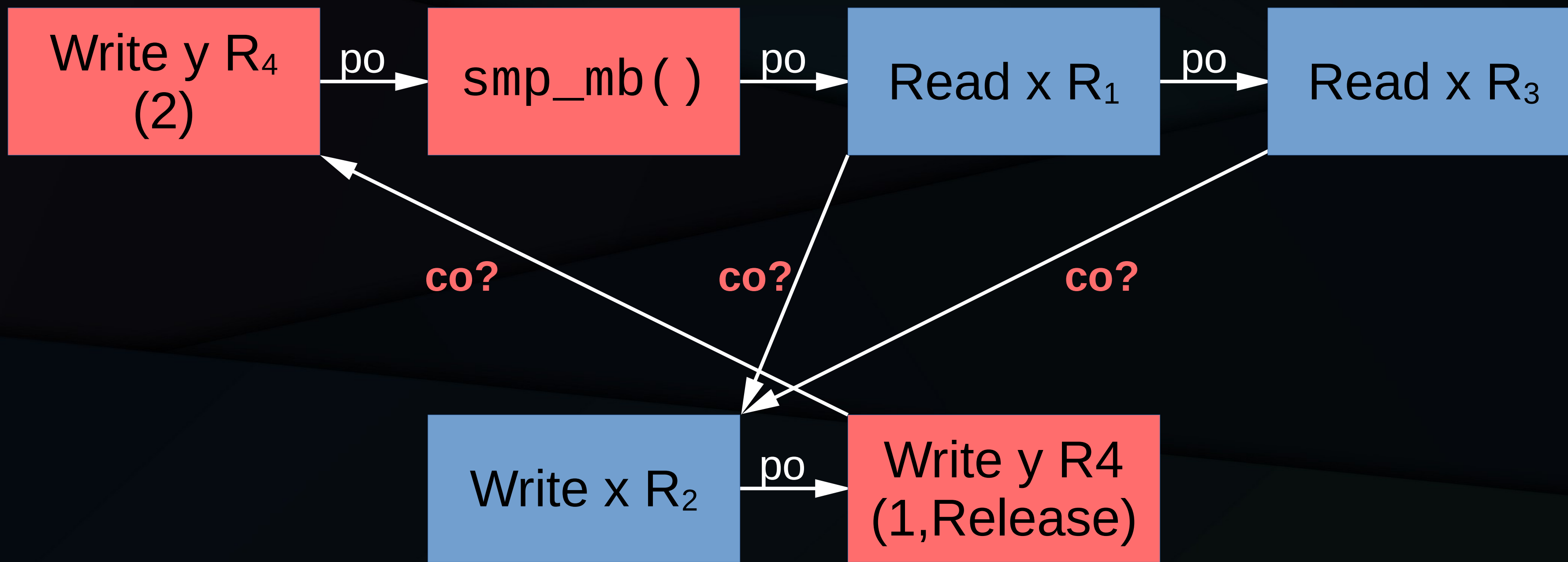
Use Light-Weight Reads-From Link?



Use Light-Weight Reads-From Link?



Check Hazard Ordered Before???



PPC Hazard Ordered Before?

```
PPC ARMv8HOB-PPC
{
0:r2=x; 0:r4=y;
1:r2=x; 1:r4=y;
}
 P0             | P1             ;
 li r1, 2       | li r1,1        ;
 stw r1,0(r2) | stw r1,0(r4) ;
 sync           | lwsync         ;
 lwz r3,0(r4) | stw r1,0(r2) ;
 lwz r5,0(r4) |                ;
locations [x;y]
exists (0:r3=0 /\ 0:r5=0 /\ x=2)
```

PPC Hazard Ordered Before?

```
$ herd7 ARMv8HOB-PPC.litmus
Test ARMv8HOB-PPC Allowed
States 6
0:r3=0; 0:r5=0; [x]=1; [y]=1;
0:r3=0; 0:r5=0; [x]=2; [y]=1;
0:r3=0; 0:r5=1; [x]=1; [y]=1;
0:r3=0; 0:r5=1; [x]=2; [y]=1;
0:r3=1; 0:r5=1; [x]=1; [y]=1;
0:r3=1; 0:r5=1; [x]=2; [y]=1;
Ok
Witnesses
Positive: 1 Negative: 5
Condition exists (0:r3=0 /\ 0:r5=0 /\ [x]=2)
Observation ARMv8HOB-PPC Sometimes 1 5
Time ARMv8HOB-PPC 0.01
Hash=a159a4cc43d21ddd63a9a1e230d266d1
```

PPC Hazard Ordered Before?

```
$ herd7 ARMv8HOB-PPC.litmus
Test ARMv8HOB-PPC Allowed
States 6
0:r3=0; 0:r5=0; [x]=1; [y]=1;
0:r3=0; 0:r5=0; [x]=2; [y]=1;
0:r3=0; 0:r5=1; [x]=1; [y]=1;
0:r3=0; 0:r5=1; [x]=2; [y]=1;
0:r3=1; 0:r5=1; [x]=1; [y]=1;
0:r3=1; 0:r5=1; [x]=2; [y]=1;
Ok
Witnesses
Positive: 1 Negativ[illegible]
Condition exists [illegible]0 /\ 0:r5=0 /\ [x]=2)
Observation A[illegible]-PPC Sometimes 1 5
Time ARMv8H[illegible] 0.01
Hash=a159a4d[illegible]43d21ddd63a9a1e230d266d1
```

PowerPC can make this happen!!!

How Does PPC Make This Happen?

- Stores are atemporal with weak barriers
  - Different PPC CPUs see stores in different orders
  - ARMv8 CPUs not doing stores agree on order
    - “Other-multicopy atomicity”, which ARMv8 does and PPC does not do
- Section 15.3.8 (“A Counter-Intuitive Case Study”) of perfbook* gives step-by-step analysis

* https://mirrors.edge.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html

How Does PPC Make This Happen?

- Stores are atemporal with weak bar
  - Different PPC CPUs see stores i erent orders
  - ARMv8 CPUs not doing st agree on order
    - “Other-multicopy ato , which ARMv8 does and PPC does not do
- Section 15 (A Counter-Intuitive Case Study”) of perf gives step-by-step analysis

But can ARMv8 make this happen?

* https://mirrors.edge.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html

ARMv8 Hazard Ordered Before?

```
AArch64 ARMv8HOB-AArch64
{
0:X2=x; 0:X4=y;
1:X2=x; 1:X4=y;
}
 P0             | P1             ;
 MOV X1,#1      | MOV X1,#1      ;
 STR X1,[X2]    | STR X1,[X4]    ;
 DMB SY         | STLR X1,[X2]   ;
 LDR X3,[X4]    |                ;
 LDR X5,[X4]    |                ;
locations [x;y]
exists (0:X3=0 /\ 0:X5=0 /\ x=2)
```

ARMv8 Hazard Ordered Before?

```
$ herd7 ARMv8HOB-AArch64.litmus
Test ARMv8HOB-AArch64 Allowed
States 3
0:X3=0; 0:X5=0; [x]=1; [y]=1;
0:X3=0; 0:X5=1; [x]=1; [y]=1;
0:X3=1; 0:X5=1; [x]=1; [y]=1;
No
Witnesses
Positive: 0 Negative: 4
Condition exists (0:X3=0 /\ 0:X5=0 /\ [x]=2)
Observation ARMv8HOB-AArch64 Never 0 4
Time ARMv8HOB-AArch64 0.02
Hash=58ec9f95130e00e226284e19adc7af48
```

ARMv8 Hazard Ordered Before?

```
$ herd7 ARMv8HOB-AArch64.litmus
Test ARMv8HOB-AArch64 Allowed
States 3
0:X3=0; 0:X5=0; [x]=1; [y]=1;
0:X3=0; 0:X5=1; [x]=1; [y]=1;
0:X3=1; 0:X5=1; [x]=1; [y]=1
No
Witnesses
Positive: 0 Negati
Condition exis    X3=0    0:X5=0 /\ [x]=2)
Observation    8HOB-AArch64 Never 0 4
Time ARMv8HO -AArch64 0.02
Hash=58ec9f95130e00e226284e19adc7af48
```

ARM really does not do this!

(Nor "R")

How Does PPC Make This Happen?

- Stores are atemporal with weak barri…
  - Different PPC CPUs see stores i… …erent orders
  - ARMv8 CPUs not doing st… …ee on order
    - “Other-multicopy ato… …RMv8 does and PPC does not do
- Section 15.3… …nter-Intuitive Case Study”) of perfb… g… step-by-step analysis

BPF does not do weak barriers

Not yet, anyway…

* https://mirrors.edge.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html

ARMv8 Lessons Learned

- Avoiding conditional-move instruction does not simplify things much
  - The cmpxchg instructions act similarly
- Avoiding other-multicopy atomicity simplifies things, but in complex ways
  - The devil is in the details, and I bet some devils still live
- Great complexity arises from some ARMv8 features:
  - MMU support (and faults), self-modifying code, cache-management instructions, MMIO accesses, shareability domains, limited-ordering regions, and vector instructions
- Weak barriers and weakly ordered instructions contribute some complexity
- Converging control flow a no-go for assembly languages (and gotos)

Why No BPF Assembly Language???

Your BPF Assembly Language!!!

```
BPF S+fence+data

{
int x=0; int y=10;
0:r0=x; 0:r1=y;
0:r5=tmp; (* only used for the atomic op in P0 to enforce ordering *)
1:r0=x; 1:r1=y;
}

P0                                          | P1                        ;
*(u32 *)(r0 + 0) = 2                        | r2 = *(u32 *)(r1 + 0)     ;
r6 = atomic_fetch_add((u64*)(r5 + 0), r6)  | *(u32 *)(r0 + 0) = r2     ;
*(u32 *)(r1 + 0) = 0                        |                           ;

exists (1:r2=0 /\ x=2)
```

And Your herd7 Output!!!

```
$ herd7 -model bpf_lkmm.cat S+fence+data.litmus
Test S+fence+data Allowed
States 3
1:r2=0; [x]=0;
1:r2=10; [x]=2;
1:r2=10; [x]=10;
No
Witnesses
Positive: 0 Negative: 3
Condition exists (1:r2=0 /\ [x]=2)
Observation S+fence+data Never 0 3
Time S+fence+data 0.00
Hash=a35dc5b17cde70582ebd0ea218dd3ba5
```

And Your herd7 Output!!!

```
$ herd7 -model bpf_lkmm.cat S+fence+data.litmus
Test S+fence+data Allowed
States 3
1:r2=0; [x]=0;
1:r2=10; [x]=2;
1:r2=10; [x]=10;
No
Witnesses
Positive: 0 Negat
Condition ex
Observatio
Time S+fenc
Hash=a35d    058    a218dd3ba5
```

Partial support for both syntax and LKMM-based memory model, courtesy of Puranjay

https://github.com/puranjaymohan/herdtools7.git herd/libdir/bpf_lkmm.cat

What is Next?

Goals of Formal BPF Memory Model

- Simple hardware-level model
- Consistent with LKMM
  - Prefer to avoid forbidding reorderings LKMM allows
- Low-overhead mappings to supported hardware
  - BPF should avoid forbidding reorderings allowed by ARMv8, PowerPC, RISC-V, x86, …
- Ability to grow as BPF instruction set grows

BPF Memory Model To-Do List

- Ordering for BPF branch instructions into herd7
- Hardware, Clang or GCC BPF mnemonics? (Poll!!!)
- Puranjay Mohan and Hernan Luis de Soto to ensure litmus-test compatibility
  - Puranjay working on herd7, Hernan on dartagnan
- Lots of memory-model validation
- Determine exact form of standard text
  - Base on LKMM, ARMv8, or something else?

BPF Memory Model Validation

- Check tools/memory-model/litmus-tests
  - In progress
- Check test6.pdf litmus tests [1]
- Check appropriate tests from github litmus [2]
- Verify BPF JIT ordering (in progress)

[1] https://www.cl.cam.ac.uk/~pes20/ppc-supplemental/test6.pdf
[2] https://github.com/paulmckrcu/litmus

Demo

Summary

Summary: BPF Memory Model

- Good progress, but much work remains
- Initial wording for standard text, subject to change
- Good frameworks for added BPF instructions, when and if
- We have a prototype of a BPF full-state-space-search formal-verification tool!!!

For More Information

- Linux-kernel BPF standards directory (includes instruction definitions)
  - Documentation/bpf/standardization
- The Herd toolsuite for memory-model verification and testing
  - https://github.com/herd/herdtools7
  - https://github.com/puranjaymohan/herdtools7.git with BPF prototype
- “Is Parallel Programming Hard, And, If So, What Can You Do About It?”
  - Chapter 12 (“Formal Verification”)
    - https://mirrors.edge.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html

Questions?