Paul E. McKenney, Meta Platforms Kernel Team
Puranjay Mohan, Kernel Developer at Amazon Web Services
Linux Plumbers Conference, eBPF Track, September 20, 2024

Instruction-level BPF memory model

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
- “Instruction-level BPF memory model”, LSF/MM/BPF 2024
  - https://lwn.net/Articles/976071/ (video: https://www.youtube.com/watch?v=QG-cLG9PekI)

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/slid memory-model-00
- “BPF Memory Model, Two Years On”, LPC
  - https://lpc.events/event/17/contribution
- “Instruction-Level BPF Memory Document
  - https://docs.google.com/d sKqkavUQyL2tThJXYWHS15qcbxIsFb0/edit?usp=sharing
- “Instruction-level BP model” MM/BPF 2024
  - https://lwn.net/Ar 71/ (video: https://www.youtube.com/watch?v=QG-cLG9PekI)

What more could possibly be needed???

What Formal Model & Tools???

What Formal Model & Tools???

- This model is upstream in the herdtools7 project
- To install and use:

```
git clone https://github.com/herd/herdtools7
cd herdtools7
# Follow instructions in INSTALL.md.
herd7 path/to/BPF/litmus/test
# Sample tests in catalogue/bpf/tests.
```

What Formal Model & Tools???

- Litmus tests in catalogue/bpf/tests:

```
CoRR+poonceonce+Once.litmus
CoRW+poonceonce+Once.litmus
CoWR+poonceonce+Once.litmus
CoWW+poonceonce.litmus
depencency_ordered_before.litmus
IRIW+fencembonceonces+OnceOnce.litmus
IRIW+poonceonces+OnceOnce.litmus
ISA2+poonceonces.litmus
LB+fcas-addr-once+once-scas.litmus
LB+fcas-ctrlcvg-once+once-scas.litmus
LB+fcas-ctrl-once+once-scas.litmus
LB+fcas-data-once+once-scas.litmus
LB+poonceonces.litmus
LockTwice.litmus
MP+fcas-addr-fcas+scas-scas.litmus
MP+fcas-ctrl-fcas+scas-scas.litmus
MP+fcas-data-fcas+scas-scas.litmus
MP+fcas-data-fcas+scas-scas-LKMM.litmus
MP+poonceonces.litmus
MP+pooncerelease+poacquireonce.litmus
R+fencembonceonces.litmus
R+poonceonces.litmus
S+atomiconce+data.litmus
SB+fence+fail_cmpxchg.litmus
SB+fencembonceonces.litmus
SB+fence+success_cmpxchg.litmus
SB+poonceonces.litmus
SB+rfionceonce-poonceonces.litmus
S+fence+addr.litmus
S+fence+ctrl-read.litmus
S+fence+ctrl-write.litmus
S+fence+data.litmus
S+onceatomic+data.litmus
S+poonceonces.litmus
WRC+poonceonces+Once.litmus
WRC+pooncerelease+fencermbonceonce+Once.litmus
W+RWC+poll+poaa+pola.litmus
X+addr-reads+corr-writes+data-rw.litmus
X-test-r2.litmus
```


UPDATED

What Formal Model & Tools???

- Litmus tests in catalogue/bpf/tests:

```
CoRR+poonceonce+Once.litmus
CoRW+poonceonce+Once.litmus
CoWR+poonceonce+Once.litmus
CoWW+poonceonce.litmus
depencency_ordered_before.litmus
IRIW+fencembonceonces+OnceOnce.litmus
IRIW+poonceonces+OnceOnce.litmus
ISA2+poonceonces.litmus
LB+fcas-addr-once+once-scas.litmus
LB+fcas-ctrlcvg-once+once-scas.litmus
LB+fcas-ctrl-once+once-scas.litmus
LB+fcas-data-once+once-scas.litmus
LB+poonceonces.litmus
LockTwice.litmus
MP+fcas-addr-fcas+sca
MP+fcas-ctrl-fcas+
MP+fcas-data-fcas
MP+fcas-data-fcas+
MP+poonceonces.litmus
MP+pooncerelease+poacquireonce.litmus
R+fencembonceonces.litmus
R+poonceonces.litmus
S+atomiconce+data.lit
SB+fence+fail_cmp
SB+fencembonce
SB+fence+s
SB+poon
SB+r
S
litmus
ite.litmus
.litmus
omic+data.litmus
nceonces.litmus
C+poonceonces+Once.litmus
WRC+pooncerelease+fencermbonceonce+Once.litmus
W+RWC+poll+poaa+pola.litmus
X+addr-reads+corr-writes+data-rw.litmus
X-test-r2.litmus
```

Plus more than 100 more as of late Tuesday...

Example BPF Litmus Test

```
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 Corresponding 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
```

Load-Acquire and Store-Release

https://lore.kernel.org/bpf/20240729183246.4110549-1-yepeilin@google.com/T/

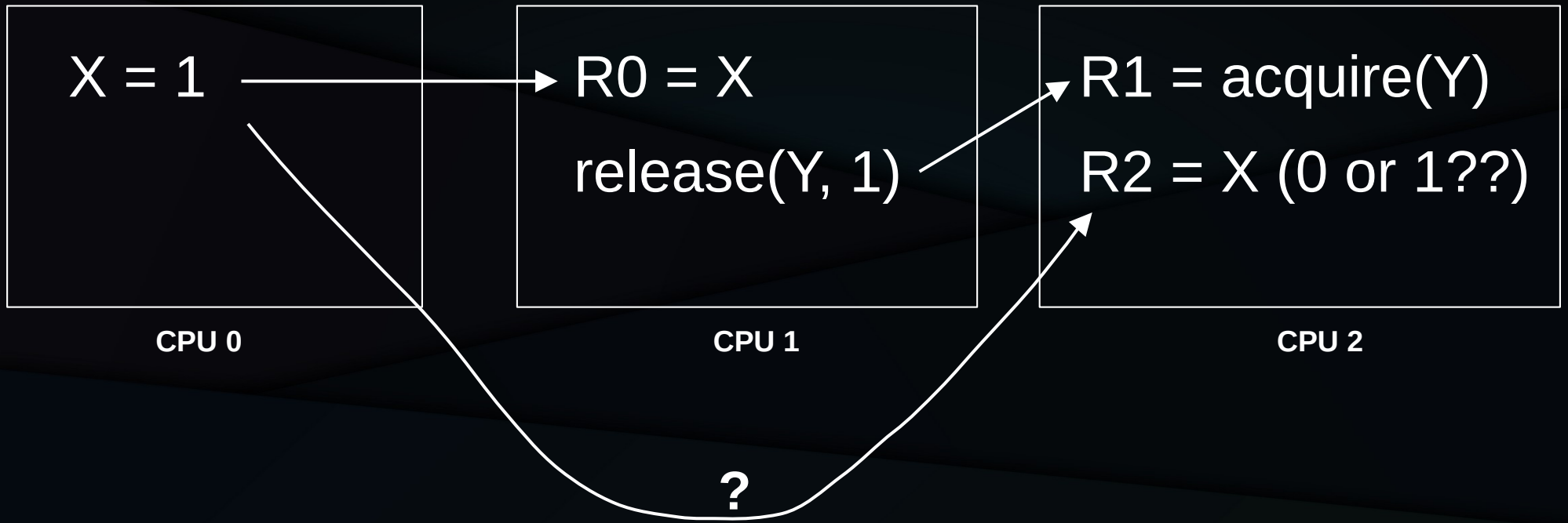
Load-Acquire and Store-Release

- Arbitrarily chose instruction formats

```
r0 = load_acquire((u32 *)(r2 + 0))
store_release((u32 *)(r2 + 0), r8)
```

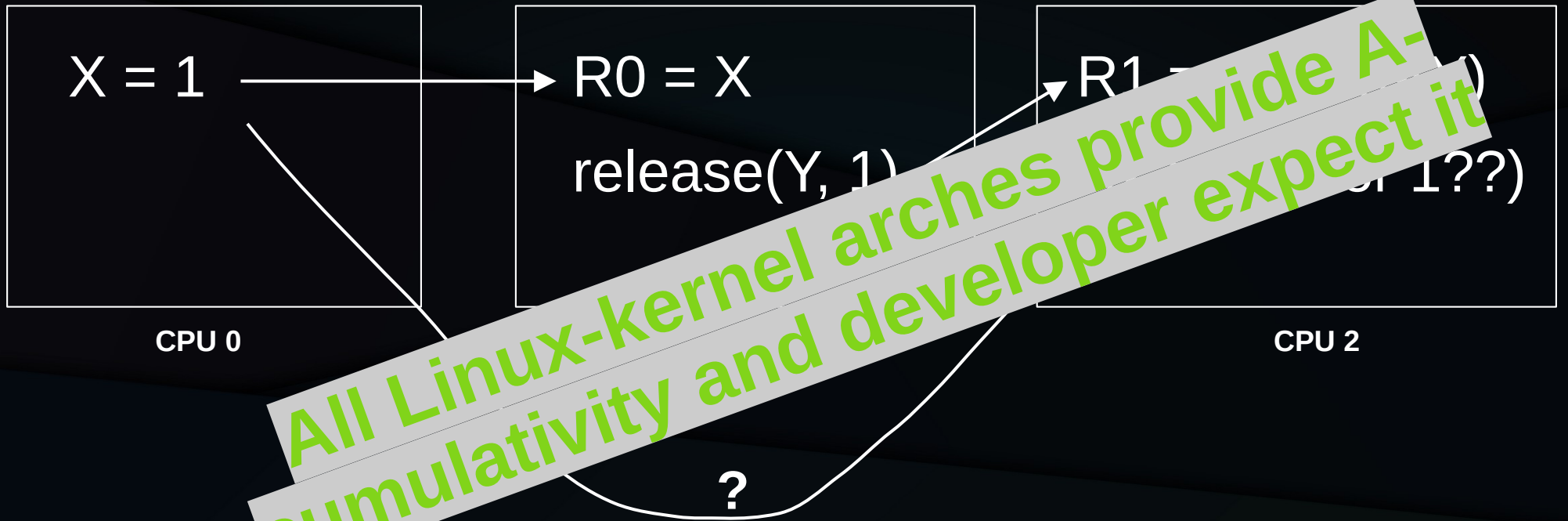
- Chose RCpc vs. RCsc (ARM64 ldapr vs. ldar)
  - ldapr (but not ldar) can be reordered with earlier stlr
  - ARM tried just ldar, performance forced ldapr
  - RCsc would force bad code on some architectures
- Made store_release A-commutative (see next slide)

What is A-Cumulativity???



If release stores are not A-cumulative, the final value of R2 can be zero!

What is A-Cumulativity???



If release stores are not A-cumulative, the final value of R2 can be zero!

What Did This Change Take?

```
git diff 4112e1ea..6315dd37 --stat herd lib/
 herd/BPFArch_herd.ml | 14 +++++++++++++-
 herd/BPFSem.ml       | 20 ++++++++++++++++++++
 herd/libdir/bpf.cat  | 11 +++++++++---
 lib/BPFBase.ml       | 23 +++++++++++++++++++++++--
 lib/BPFLexer.mll     |  2 ++
 lib/BPFParser.mly    | 12 ++++++++++++
 6 files changed, 76 insertions(+), 6 deletions(-)
```

Demo

Validation

Validation

- Convert existing LKMM tests to BPF!!!
  - https://github.com/paulmckrcu/litmus
  - Early days, and hopefully replaced by something a bit more formal at some point ;-)

Validation: Convert LKMM to BPF

- tools/memory-model/litmus-tests:
  - 35 Total
  - 22 Without RCU, SRCU, locking, and weak barriers
  - 20 Without "if" statements and smp_store_mb()
  - 20 Potentially convertable to BPF
  - 20 Compatible LKMM and BPF outcomes

Validation: Convert LKMM to BPF

- tools/memory-model/litmus-tests:
  - 35 Total
  - 22 Without RCU, SRCU, locking, and weak barriers
  - 20 Without "if", and smp_store_mb()
  - 20 Automatically convertable to BPF
  - 20 Compatible LKMM and BPF outcomes

Found one bug in the conversion script (fixed)

Validation

- https://github.com/paulmckrcu/litmus:
  - 5374 Total
  - 2493 Without RCU, SRCU, locking, and weak barriers
  - 2166 Without "if" statements and smp_store_mb()
  - 146 Potentially convertable to BPF
  - 133 Excluding casted/unmarked accesses and atomic RMW
  - 126 Compatible LKMM and BPF outcomes
  - 7 **With incompatible outcomes**

Validation

- https://github.com/paulmckrcu/litmus:
  - 5374 Total
  - 2493 Without RCU, SRCU, locking, ...
  - 2166 Without "if" statement...
  - 146 Potentially ...
  - 133 Ex... accesses and atomic RMW
  - 1... and BPF outcomes
  - ... **incompatible outcomes**

Found many script bugs and one memory-model bug (fix pending)

Validation

- https://github.com/paulmckrcu/litmus:
  - 5374 Total
  - 2493 Without R… SRCU, locking…
  - 2166 Without "if…
  - 146 Potentially…
  - 133 Ex… …ked ac… …d atomic RMW
  - 1… …MM and BPF outcomes
  - … **…ncompatible outcomes**

Found many … bugs and one memory-model b… (fix pending)

Fix was pending…

Validation

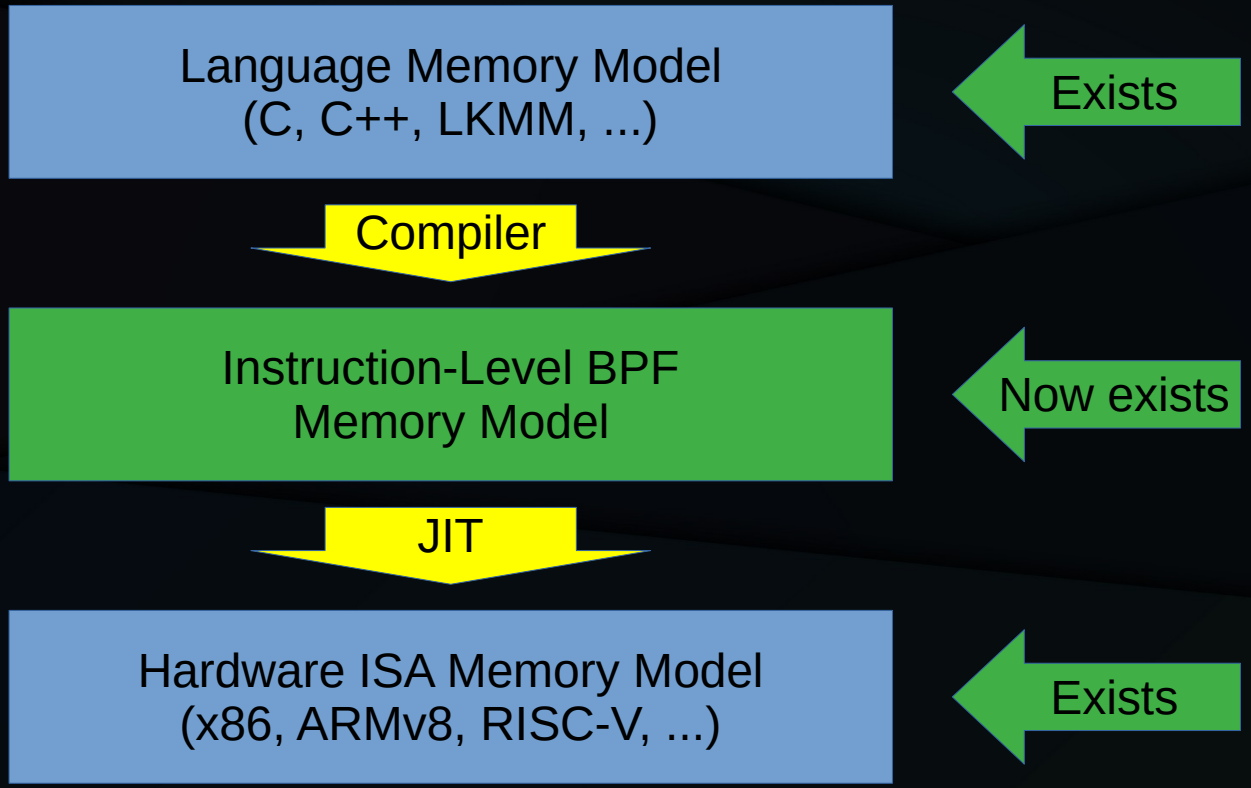
- https://github.com/paulmckrcu/litmus:
  - 5374 Total
  - 2493 Without RCU, SRCU, locking, and weak barriers
  - 2166 Without "if" statements and smp_store_mb()
  - 146 Potentially convertable to BPF
  - 133 Excluding casted/unmarked accesses and atomic RMW
  - 126 Compatible LKMM and BPF outcomes
  - 0 **With incompatible outcomes**

JIT Complications

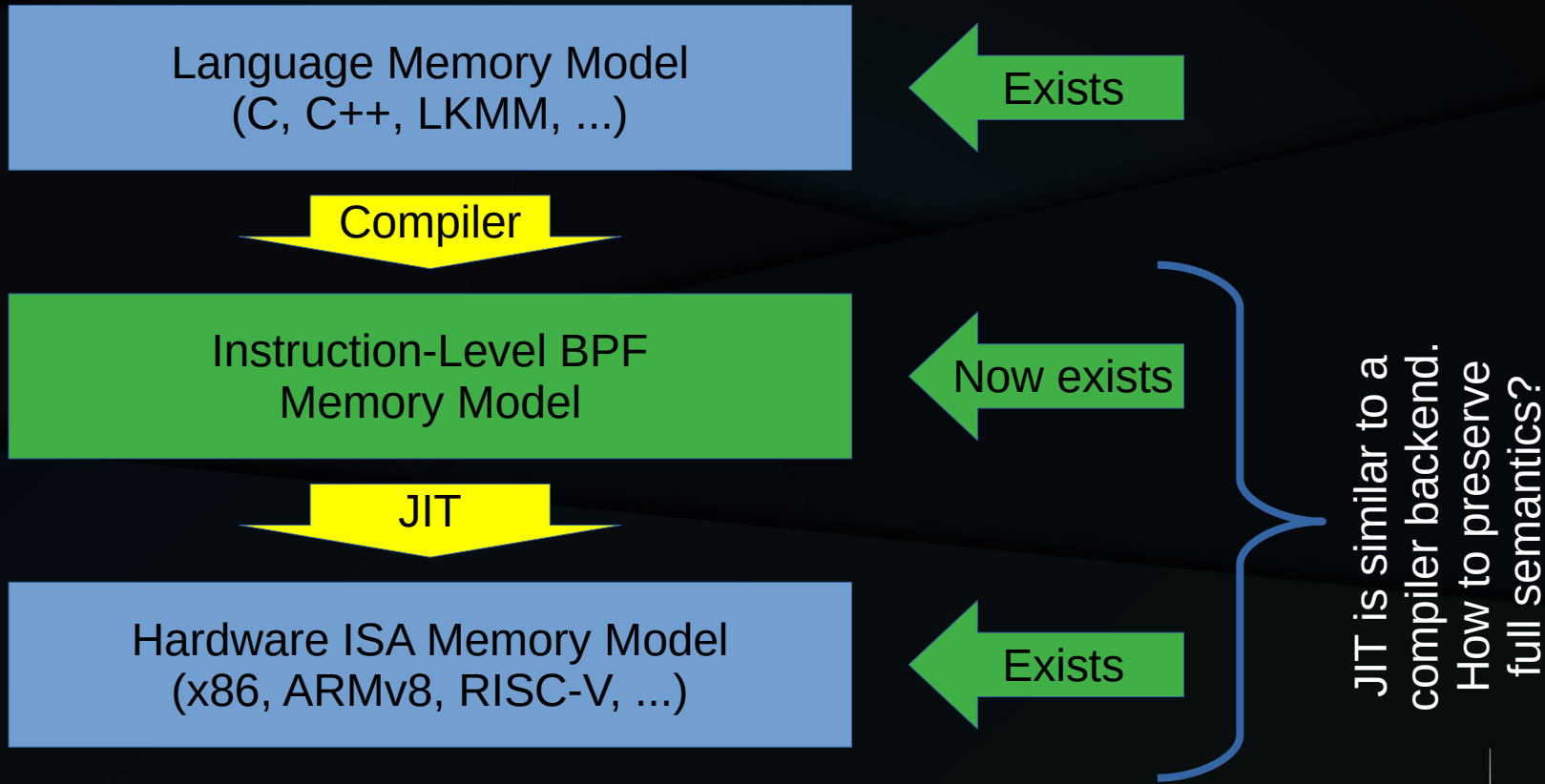
BPF Conditional Jump Instructions

- 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 Instructions To Other Instructions



BPF Instructions To Other Instructions



JIT Complications

- Register Mismatches
- ABI Calling Conventions
- Backend Optimizations

Register Mismatches

- BPF has R0-R10, real hardware has 16, 32, …
- Can map BPF R0-R10 to fixed HW registers
  - Usually gives up performance: spills/reloads
- If fewer HW registers, dynamically map
- Many JITs treat R0-R10 as C-language auto variables whose addresses have not been taken

ABI Calling Conventions

- BPF has calling conventions
- But so do hardware-assembly BPF helpers
- JIT might need to map calling conventions
- Fun when doing stack unwinding: shadow stack

Backend Optimizations

- Inlining complicates stack unwinding and optimizations
- Arithmetic optimizations
  - Multiplication by zero replaced by zero, discarding other operand and computations leading up to it
  - Subtracting an expression from itself is also cancelled
- Type-based inference
  - Range-based tracking of register values permits eliding of branch instructions

Optimizations Break Dependencies

Checking Dependencies

- The klitmus tool starts with an LKMM litmus test, then creates a kernel module that tests it
  - Can prove something happens, but cannot prove that something cannot happen
- Use klitmus-like tool translate JIT BPF assembly litmus tests to a kernel module, check for broken dependencies
  - Again, cannot prove breakage does not happen: Still useful

Where Is the BPF Memory Model?

- Overall direction set in 2021
- Informal instruction-level ordering in late 2023
- Formal model and tools in early 2024
- Handle new load-acquire and store-release instructions in late 2024
  - Adjustments might be needed based on eventual instruction format and semantics
- Verification against LKMM in late 2024 (support for “if” statements still needed)
- Things known to still be left:
  - There might also be a full-barrier instruction
    - Currently emulated with no-operation value-returning atomic operations
  - Comparison of BPF MM against hardware models (klitmus-like tool TBD)

Summary

Summary

- BPF memory model now has:
  - Prototype load-acquire/store-release handling
  - Automated checking against LKMM
    - Other than some atomics and “if” statements

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 with base memory model
  - https://github.com/puranjaymohan/herdtools7.git with load-acquire/store-release 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?

Backup

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 prior load (BPF_LD or BPF_LDX), ***and***
  - There is a store instruction (BPF_ST…) ~~***converges, and*** following t~~…truction in program order
  - ~~**The restrictions … DEPENDENCIES" section of Documen… …ers.txt are faithfully followed**~~

~~**…and control dependencies, and happily break them.**~~

~~**…volving conditional-move instructions requires the "before con… converges" restriction**~~

Friends don't let friends run BPF assembly through an optimizing compiler