Demystifying Memory Models Across the Computing Stack

Yatin A. Manerkar, Caroline Trippel, Margaret Martonosi
Princeton University
ISCA 2019

While you wait:

1) Make sure you’ve got VirtualBox downloaded to your laptop:
   https://www.virtualbox.org/wiki/Downloads

2) Make sure you have Tutorial VM downloaded (or use one of the USB drives):
   VM Password: mcmsarefun

http://check.cs.princeton.edu/tutorial.html
Goals

▪ Reestablish the basics: Why Memory Consistency Models matter… more than ever!

▪ Give you concrete tools and techniques for broader MCM research

▪ Foster a broader community conversant and active in MCM issues

▪ Show connections outwards to other topics: Security, Distributed Systems, etc.

▪ Get you thinking about future research possibilities in this area
Our Approach Today

▪ Start from basic knowledge of Memory Consistency Models
  • Instruction at level of first-year graduate student
  • Will give background info.
  • If it’s too basic or too fast, say so.

▪ Variety is the spice of life... Intersperse:
  • Theory
  • Techniques
  • Tool specifics
  • Demos
What does this program print?

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
</table>
| 1 x = 1; | 3 if (y == 1)  
|          | print("Answer is:"); |
| 2 y = 1; | 4 if (x == 1)  
|          | print("42"); |
What does this program print?

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>① x = 1;</td>
<td>③ if (y == 1) print(&quot;Answer is: &quot;);</td>
</tr>
<tr>
<td>② y = 1;</td>
<td>④ if (x == 1) print(&quot;42&quot;);</td>
</tr>
</tbody>
</table>

Can it print “Answer is: 42”?
What does this program print?

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
</table>
| ① \( x = 1; \)   | ③ \( \text{if } \( y == 1 \) \)
|                   | \( \text{print("Answer is:");} \) |
| ② \( y = 1; \)   | ④ \( \text{if } \( x == 1 \) \)
|                   | \( \text{print("42"}; \) |

Can it print “Answer is: 42”?  Yes, eg: ①②③④
What does this program print?

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>① x = 1;</td>
<td>③ if (y == 1)</td>
</tr>
<tr>
<td></td>
<td>print(&quot;Answer is:&quot;);</td>
</tr>
<tr>
<td>② y = 1;</td>
<td>④ if (x == 1)</td>
</tr>
<tr>
<td></td>
<td>print(&quot;42&quot;);</td>
</tr>
</tbody>
</table>

Can it print "Answer is: 42"? Yes, eg: ① ② ③ ④

How about just "42"?
What does this program print?

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>❶ $x = 1$;</td>
<td>❹ if ($x == 1$) print(&quot;42&quot;);</td>
</tr>
<tr>
<td>❷ $y = 1$;</td>
<td></td>
</tr>
<tr>
<td>❸ if ($y == 1$) print(&quot;Answer is:&quot;);</td>
<td></td>
</tr>
</tbody>
</table>

Can it print “Answer is: 42”? Yes, eg: ❶ ❷ ❸ ❹

How about just “42”? Yes, eg: ❶ ❹ ❸ ❷ ❱
What does this program print?

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. x = 1;</td>
<td>3. if (y == 1)</td>
</tr>
<tr>
<td></td>
<td>print(&quot;Answer is:&quot;);</td>
</tr>
<tr>
<td>2. y = 1;</td>
<td>4. if (x == 1)</td>
</tr>
<tr>
<td></td>
<td>print(&quot;42&quot;);</td>
</tr>
</tbody>
</table>

Can it print “Answer is: 42”? Yes, eg: 1 2 3 4
How about just “42”? Yes, eg: 1 3 4 2
Could it print nothing?
What does this program print?

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>❶ x = 1;</td>
<td>❸ if (y == 1)</td>
</tr>
<tr>
<td></td>
<td>print(&quot;Answer is:&quot;);</td>
</tr>
<tr>
<td>❷ y = 1;</td>
<td>❹ if (x == 1)</td>
</tr>
<tr>
<td></td>
<td>print(&quot;42&quot;);</td>
</tr>
</tbody>
</table>

Can it print "Answer is: 42"? Yes, eg: ❶ 2 3 4
How about just "42"? Yes, eg: 1 3 4 2
Could it print nothing? Yes, eg: 3 4 1 2
What does this program print?

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>① (x = 1);</td>
<td>③ (\text{if} \ (y == 1)) print(&quot;Answer is:&quot;);</td>
</tr>
<tr>
<td>② (y = 1);</td>
<td>④ (\text{if} \ (x == 1)) print(&quot;42&quot;);</td>
</tr>
</tbody>
</table>

Can it print “Answer is: 42”? Yes, eg: ① ② ③ ④

How about just “42”? Yes, eg: ① ③ ④ ②

Could it print nothing? Yes, eg: ③ ④ ① ②

These executions obey **Sequential Consistency (SC) [Lamport79]**, which requires that the results of the overall program correspond to some in-order interleaving of the statements from each individual thread.
How about “Answer is:”?
What does this program print?

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( x = 1; )</td>
<td>3 if ( y == 1 ) ( \text{print(&quot;Answer is:&quot;;} )</td>
</tr>
<tr>
<td>2 ( y = 1; )</td>
<td>4 if ( x == 1 ) ( \text{print(&quot;42&quot;);} )</td>
</tr>
</tbody>
</table>

How about “Answer is:”? It depends!

2 1 3 4
What does this program print?

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>① ( x = 1; )</td>
<td>③ if ( y == 1 ) print(&quot;Answer is:&quot;);</td>
</tr>
<tr>
<td>② ( y = 1; )</td>
<td>④ if ( x == 1 ) print(&quot;42&quot;);</td>
</tr>
</tbody>
</table>

How about “Answer is:”? It depends!

NO!
What does this program print?

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( x = 1; )</td>
<td>3 ( \text{if (} y == 1 \text{)} ) ( \text{print(&quot;Answer is:&quot;;);} )</td>
</tr>
<tr>
<td>2 ( y = 1; )</td>
<td>4 ( \text{if (} x == 1 \text{)} ) ( \text{print(&quot;42&quot;);} )</td>
</tr>
</tbody>
</table>

How about “Answer is:”? It depends!

NO! YES!

It depends! 2 1 3 4
What does this program print?

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1;</td>
<td>if (y == 1)</td>
</tr>
<tr>
<td></td>
<td>print(&quot;Answer is:&quot;);</td>
</tr>
<tr>
<td>y = 1;</td>
<td></td>
</tr>
<tr>
<td>if (x == 1)</td>
<td>print(&quot;42&quot;);</td>
</tr>
</tbody>
</table>

Why would we reorder memory operations?

How to specify what’s allowed and forbidden?

How do check that implementations match spec?

We’ll cover the answers today!
Why reorder memory operations?

Answer: Performance!

<table>
<thead>
<tr>
<th>Message Passing (mp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 0</td>
</tr>
<tr>
<td>x = 1;</td>
</tr>
<tr>
<td>y = 1;</td>
</tr>
<tr>
<td>Core 1</td>
</tr>
<tr>
<td>r1 = y;</td>
</tr>
<tr>
<td>r2 = x;</td>
</tr>
</tbody>
</table>

Can r1=1 and r2=0?
Why reorder memory operations?

**Answer: Performance!**

Can improve performance by sending both stores to memory in parallel

<table>
<thead>
<tr>
<th></th>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1; y = 1;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r1 = y; r2 = x;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Message Passing (mp)

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1;</td>
<td>r1 = y;</td>
</tr>
<tr>
<td>y = 1;</td>
<td>r2 = x;</td>
</tr>
</tbody>
</table>

Can r1=1 and r2=0?
Answer: Performance!

Message Passing (mp)

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1;</td>
<td>r1 = y;</td>
</tr>
<tr>
<td>y = 1;</td>
<td>r2 = x;</td>
</tr>
</tbody>
</table>

Can r1=1 and r2=0?

Store to y finishes quickly in cache
Why reorder memory operations?

Answer: Performance!

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = 1; )</td>
<td>( r_1 = y = 1;)</td>
</tr>
<tr>
<td>( y = 1; )</td>
<td>( r_2 = x; )</td>
</tr>
</tbody>
</table>

Can \( r_1 = 1 \) and \( r_2 = 0 \)?

Message Passing (mp)

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = 1; )</td>
<td>( r_1 = y; )</td>
</tr>
<tr>
<td>( y = 1; )</td>
<td>( r_2 = x; )</td>
</tr>
</tbody>
</table>

Cache

| y: 1 |

Memory

| x: 0 |
Why reorder memory operations?

Answer: Performance!

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1; y = 1;</td>
<td>r1 = y = 1; r2 = x = 0;</td>
</tr>
</tbody>
</table>

Can r1=1 and r2=0?

Message Passing (mp):

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1; y = 1;</td>
<td>r1 = y; r2 = x;</td>
</tr>
</tbody>
</table>
Why reorder memory operations?

**Answer: Performance!**

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1; y = 1;</td>
<td>r1 = y; r2 = x;</td>
</tr>
</tbody>
</table>

Can \( r1 = 1 \) and \( r2 = 0 \)?

By the time store of \( x \) is complete, Core 1 has observed reordering!
Why reorder memory operations?

Answer: Performance!

Message Passing (mp)

<table>
<thead>
<tr>
<th></th>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>= 1;</td>
<td>r1 = y;</td>
</tr>
<tr>
<td>y</td>
<td>= 1;</td>
<td>r2 = x;</td>
</tr>
</tbody>
</table>

Can r1=1 and r2=0?

Fence/synchronization instructions can enforce order between memory operations where needed.
Compilers Reorder Memory Operations Too!

- Compiler optimizations can also result in weak memory behaviours
  - Example below: assume CPU performs instrs in order and 1 at a time

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>① x = 1;</td>
<td>④ r1 = y;</td>
</tr>
<tr>
<td>② y = 1;</td>
<td>⑤ r2 = x;</td>
</tr>
<tr>
<td>③ x = 2;</td>
<td></td>
</tr>
</tbody>
</table>

Can r1 = 1 and r2 = 0?
Compilers Reorder Memory Operations Too!

- Compiler optimizations can also result in weak memory behaviours
  - Example below: assume CPU performs instrs in order and 1 at a time

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x = 1;</td>
<td>4 r1 = y;</td>
</tr>
<tr>
<td>2 y = 1;</td>
<td>5 r2 = x;</td>
</tr>
<tr>
<td>3 x = 2;</td>
<td></td>
</tr>
</tbody>
</table>

Can r1 = 1 and r2 = 0?

Compiler may coalesce these 2 stores (since no same-thread reads of x in between)
Compilers Reorder Memory Operations Too!

- Compiler optimizations can also result in weak memory behaviours
  - Example below: assume CPU performs instrs in order and 1 at a time

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>② y = 1;</td>
<td>④ r1 = y;</td>
</tr>
<tr>
<td>③ x = 2;</td>
<td>⑤ r2 = x;</td>
</tr>
</tbody>
</table>

Can r1 = 1 and r2 = 0?
Compilers Reorder Memory Operations Too!

- Compiler optimizations can also result in weak memory behaviours
  - Example below: assume CPU performs instrs in order and 1 at a time

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>❶ x = 1;</td>
<td>❷ r1 = y;</td>
</tr>
<tr>
<td>❷ y = 1;</td>
<td>❺ r2 = x;</td>
</tr>
<tr>
<td>❸ x = 2;</td>
<td>❹ r1 = y;</td>
</tr>
</tbody>
</table>

Can r1 = 1 and r2 = 0?

Now ❷ ❹ ❺ ❸ gives r1 = 1 and r2 = 0!
Memory Consistency Models (MCMs)

- ISA instructions represent hardware operations (add, sub, ld, st, ...)
- MCMs similarly represent the orderings among hardware memory ops

Microarchitecture\(^1\)

\(^1\)Microarchitecture is a component-level (e.g. caches, pipeline stages, store buffers) model of the hardware.
Memory Consistency Models (MCMs)

- ISA instructions represent hardware operations (add, sub, ld, st, ...)
- MCMs similarly represent the orderings among hardware memory ops

Which compiler optimizations can I use?

Compiler

Microarchitecture

1Microarchitecture is a component-level (e.g. caches, pipeline stages, store buffers) model of the hardware.
Memory Consistency Models (MCMs)

- ISA instructions represent hardware operations (add, sub, ld, st, ...)
- MCMs similarly represent the orderings among hardware memory ops

Which compiler optimizations can I use?

Compiler

How much can I buffer and reorder memory operations?

Microarchitecture

1 Microarchitecture is a component-level (e.g. caches, pipeline stages, store buffers) model of the hardware.
Memory Consistency Models (MCMs)

- ISA instructions represent hardware operations (add, sub, ld, st, ...)
- MCMs similarly represent the orderings among hardware memory ops

Which compiler optimizations can I use?

Compiler

ISA-Level MCM (x86, ARMv8, RISC-V, etc)

Microarchitecture

How much can I buffer and reorder memory operations?

1Microarchitecture is a component-level (e.g. caches, pipeline stages, store buffers) model of the hardware.
Memory Consistency Models (MCMs)

- ISA instructions represent hardware operations (add, sub, ld, st, ...)
- MCMs similarly represent the orderings among hardware memory ops

In a nutshell: MCMs specify what value will be returned when your program does a load!

Microarchitecture

ARMv8, RISC-V, etc)

How much can I buffer and reorder memory operations?

1Microarchitecture is a component-level (e.g. caches, pipeline stages, store buffers) model of the hardware.
Memory Consistency Models (MCMs)

Specify rules and guarantees about the ordering and visibility of accesses to shared memory [Sorin et al., 2011].
Memory Consistency Models (MCMs)

Specify rules and guarantees about the ordering and visibility of accesses to shared memory [Sorin et al., 2011].
Memory Consistency Models (MCMs)

Specify rules and guarantees about the ordering and visibility of accesses to shared memory [Sorin et al., 2011].
Memory Consistency Models (MCMs)

Specify rules and guarantees about the ordering and visibility of accesses to shared memory [Sorin et al., 2011].
How are MCMs specified?

- Natural language?
  - E.g. Sequential Consistency [Lamport 1979]

  “The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.”

- What about more complicated models?
How are MCMs specified?

- Excerpt from the ARMv8 manual (memory model section):

  Architecturally well-formed
  
  An architecturally well-formed execution must satisfy both of the following requirements:

  **Internal visibility requirement**
  
  For a read or a write RW₁ that appears in program order before a read or a write RW₂ to the same Location, the internal visibility requirement requires that exactly one of the following statements is true:

  - RW₂ is a write W₂ that is Coherence-after RW₁.
  - RW₁ is a write W₁ and RW₂ is a read R₂ such that either:
    - R₂ Reads-from W₁.
    - R₂ Reads-from another write that is Coherence-after W₁.
  - RW₁ and RW₂ are both reads R₁ and R₂ such that R₁ Reads-from a write W₃ and either:
    - R₂ Reads-from W₃.
    - R₂ Reads-from another write that is Coherence-after W₃.

  ———— Note ————

  If a Memory effect M₁ from an Observer appears in program order before a Memory effect M₂ from the same Observer, then M₁ will be seen to occur before M₂ by that Observer.
MCM Specifications Using Relations

- ISA-level MCMs defined using relational patterns [Shasha and Snir TOPLAS 1988]
- ISA-level executions are graphs
  - nodes: instructions, edges: ISA-level relations
- Eg: SC is \( \text{acyclic}(po \cup co \cup rf \cup fr) \)

Message passing (mp) litmus test

<table>
<thead>
<tr>
<th></th>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
<td>( x = 1; )</td>
<td>( (i3) r1 = y; )</td>
</tr>
<tr>
<td>(i2)</td>
<td>( y = 1; )</td>
<td>( (i4) r2 = x; )</td>
</tr>
<tr>
<td>SC Forbids:</td>
<td>( r1 = 1, r2 = 0 )</td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- po = Program order
- co = Coherence order
- rf = Reads-from
- fr = From-reads
MCM Specifications Using Relations

- ISA-level MCMs defined using relational patterns [Shasha and Snir TOPLAS 1988]
- ISA-level executions are graphs
  - **nodes**: instructions, **edges**: ISA-level relations
- Eg: SC is \( \text{acyclic} (po \cup co \cup rf \cup fr) \)

**Legend:**
- \( po \) = Program order
- \( co \) = Coherence order
- \( rf \) = Reads-from
- \( fr \) = From-reads

**Message passing (mp) litmus test**

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) ( x = 1; )</td>
<td>(i3) ( r1 = y; )</td>
</tr>
<tr>
<td>(i2) ( y = 1; )</td>
<td>(i4) ( r2 = x; )</td>
</tr>
</tbody>
</table>

**SC Forbids:** \( r1 = 1, r2 = 0 \)
MCM Specifications Using Relations

- ISA-level MCMs defined using relational patterns [Shasha and Snir TOPLAS 1988]
- ISA-level executions are graphs
  - nodes: instructions, edges: ISA-level relations
- Eg: SC is $\text{acyclic}(\text{po} \cup \text{co} \cup \text{rf} \cup \text{fr})$

Message passing (mp) litmus test

<table>
<thead>
<tr>
<th></th>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
<td>$x = 1;$</td>
<td></td>
</tr>
<tr>
<td>(i2)</td>
<td>$y = 1;$</td>
<td></td>
</tr>
<tr>
<td>(i3)</td>
<td>$r1 = y;$</td>
<td>$r1 = y;$</td>
</tr>
<tr>
<td>(i4)</td>
<td>$r2 = x;$</td>
<td></td>
</tr>
</tbody>
</table>

SC Forbids: $r1 = 1$, $r2 = 0$

Legend:
- po = Program order
- co = Coherence order
- rf = Reads-from
- fr = From-reads
MCM Specifications Using Relations

- ISA-level MCMs defined using relational patterns [Shasha and Snir TOPLAS 1988]
- ISA-level executions are graphs
  - **nodes**: instructions, **edges**: ISA-level relations
- Eg: SC is \(\text{acyclic}(po \cup co \cup rf \cup fr)\)

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) (x = 1);</td>
<td>(i3) (r1 = y);</td>
</tr>
<tr>
<td>(i2) (y = 1);</td>
<td>(i4) (r2 = x);</td>
</tr>
</tbody>
</table>

**SC Forbids**: \(r1 = 1, r2 = 0\)

**Legend:**
- po = Program order
- co = Coherence order
- rf = Reads-from
- fr = From-reads

Message passing (mp) litmus test
MCM Specifications Using Relations

- ISA-level MCMs defined using relational patterns [Shasha and Snir TOPLAS 1988]
- ISA-level executions are graphs
  - **nodes:** instructions, **edges:** ISA-level relations
- Eg: SC is \( \text{acyclic}(\text{po} \cup \text{co} \cup \text{rf} \cup \text{fr}) \)
- Formal specifications of ISA + HLL MCMs
  - x86 [Owens et al. TPHOLS2009], ARM [Pulte et al. POPL2018], C11 [Batty et al. POPL 2011], ...
- Automated formal tools e.g. **herd** [Alglave et al. TOPLAS 2014]
  - Can formally analyse small test programs against these models

### Message passing (mp) litmus test

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) ( x = 1; )</td>
<td>(i3) ( r1 = y; )</td>
</tr>
<tr>
<td>(i2) ( y = 1; )</td>
<td>(i4) ( r2 = x; )</td>
</tr>
</tbody>
</table>

**SC Forbids:** \( r1 = 1, r2 = 0 \)

**Legend:**
- po = Program order
- co = Coherence order
- rf = Reads-from
- fr = From-reads
The Need for MCM Verification

- MCM specified at an interface between layers of the stack
- Upper layers target the MCM; lower layers must maintain it!

![Diagram]

- Upper layer (e.g. Compiler)
- Interface (e.g. ISA-Level MCM)
- Lower layer (e.g. Microarch.)
The Need for MCM Verification

- MCM specified at an interface between layers of the stack
- Upper layers target the MCM; lower layers must maintain it!

Targets MCM of lower layer

Upper layer (e.g. Compiler)

Interface (e.g. ISA-Level MCM)

Lower layer (e.g. Microarch.)
The Need for MCM Verification

- MCM specified at an interface between layers of the stack
- Upper layers target the MCM; lower layers must maintain it!

Targets MCM of lower layer

Upper layer (e.g. Compiler)

Interface (e.g. ISA-Level MCM)

Lower layer (e.g. Microarch.)

Must maintain MCM of interface!
The Need for MCM Verification

- MCM specified at an interface between layers of the stack
- Upper layers target the MCM; lower layers must maintain it!

Upper layer (e.g. Compiler)

Lower layer (e.g. Microarch.)

Must maintain MCM of interface!
The Check Suite: Tools For Verifying Memory Orderings and their Security Implications

High-Level Languages (HLL)  Compiler  OS  Architecture (ISA)  Microarchitecture  RTL (e.g. Verilog)

PipeCheck [Micro ‘14] [IEEE MICRO Top Picks]

For more info: check.cs.Princeton.edu
The Check Suite: Tools For Verifying Memory Orderings and their Security Implications

- High-Level Languages (HLL)
- Compiler
- Architecture (ISA)
- Microarchitecture
- RTL (e.g. Verilog)

- TriCheck [ASPLOS ‘17] [IEEE MICRO Top Picks]
- COATCheck [ASPLOS ‘16] [IEEE MICRO Top Picks]
- PipeCheck [Micro ‘14] [IEEE MICRO Top Picks]
- CCICheck [Micro ‘15] [Nominated for Best Paper Award]
- RTLCheck [Micro ‘17] [IEEE MICRO Top Picks Honorable Mention]

For more info: check.cs.Princeton.edu
The Check Suite: Tools For Verifying Memory Orderings and their Security Implications

High-Level Languages (HLL)  
Compiler  
Architecture (ISA)  
Microarchitecture  
RTL (e.g. Verilog)

TriCheck [ASPLOS ‘17] [IEEE MICRO Top Picks]  
COATCheck [ASPLOS ‘16] [IEEE MICRO Top Picks]  
PipeCheck [Micro ‘14] [IEEE MICRO Top Picks]  
CCICheck [Micro ‘15] [Nominated for Best Paper Award]  
RTLCheck [Micro ‘17] [IEEE MICRO Top Picks Honorable Mention]

Our Approach

• Axiomatic specifications -> Happens-before graphs
• Check Happens-Before Graphs via Efficient SMT solvers
  • Cyclic => A->B->C->A... Can’t happen
  • Acyclic => Scenario is observable

For more info: check.cs.Princeton.edu
The Check Suite: Tools For Verifying Memory Orderings and their Security Implications

Our Approach
- Axiomatic specifications -> Happens-before graphs
- Check Happens-Before Graphs via Efficient SMT solvers
  - Cyclic => A->B->C->A... Can’t happen
  - Acyclic => Scenario is observable

For more info: check.cs.Princeton.edu
The Check Suite: Tools For Verifying Memory Orderings and their Security Implications

Our Approach
- Axiomatic specifications -> Happens-before graphs
- Check Happens-Before Graphs via Efficient SMT solvers
  - Cyclic => A->B->C->A... Can’t happen
  - Acyclic => Scenario is observable

So far, tools have found bugs in:
- Widely-used Research simulator
- Cache coherence paper
- IBM XL C++ compiler (fixed in v13.1.5)
- In-design commercial processors
- RISC-V ISA specification
- Compiler mapping proofs
- C++ 11 mem model
- SpectrePrime, MeltdownPrime
In a nutshell, our tool philosophy…

▪ Automate specification, verification, and translation related to MCMs
▪ Comprehensive exploration of ordering possibilities
▪ Key Techniques: Happens-before Graphs and SMT solvers
▪ Initially: Litmus-test driven (small test programs, 4-8 instrs)
▪ Now: PipeProof demonstrates complete (i.e. all-program) analysis
Outline

- Overview, Motivation, and MCM Background (15 minutes) (mm)
- PipeCheck: Verifying Microarchitectural Implementations against ISA Specs (45 minutes)
  - Includes hands-on of using uSpec DSL for specifying axioms (30 minutes) (ym)
- PipeProof: Beyond Litmus Tests (45 minutes) (ym)
  - Includes hands-on of proving simple microarch. across all programs (25 minutes)
- Coffee Break. 11-11:20
- Up and Down the Stack
  - RTLCheck (15 minutes) (ym)
  - TriCheck (10 minutes) (ct)
- Looking forward: Other uses of tools and techniques
  - CheckMate for security (25 minutes) (ct)
- Conclusions and Bigger Picture (10 minutes)