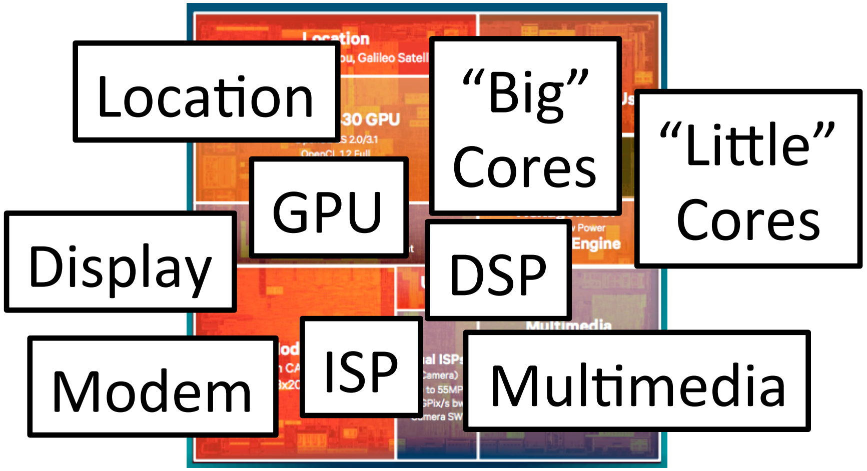
“Armour of Sir James Scudamore (1595)”, Todd Hoogerland, CC-BY-SA-3.0
http://commons.wikimedia.org/wiki/File:Scudamorearmor2.jpg

ArMOR: Defending Against Memory Consistency Model Mismatches in Heterogeneous Architectures

Daniel Lustig, Caroline Trippel, Michael Pellauer, and Margaret Martonosi

Motivation: MCMs Are Still Difficult!

- Are memory consistency models (MCMs) a solved problem? Not entirely!
  - Proper MCM specification: only partially solved
  - MCM-aware compilation: only partially solved
  - Cross-MCM dynamic binary translation: previously unsolved!

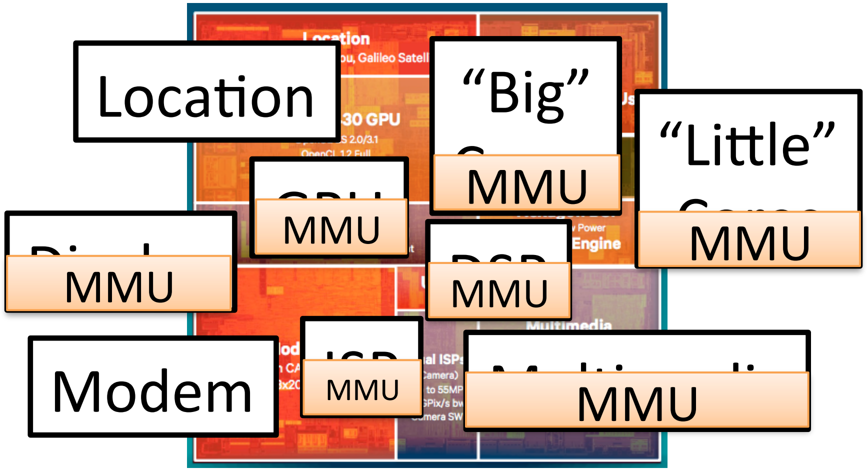


Qualcomm Snapdragon 810

- With the emergence of architecturally heterogeneous (and hence MCM-heterogeneous) systems, the problems are only going to get worse!

Motivation: MCMs Are Still Difficult!

- Are memory consistency models (MCMs) a solved problem? Not entirely!
  - Proper MCM specification: only partially solved
  - MCM-aware compilation: only partially solved
  - Cross-MCM dynamic binary translation: previously unsolved!



Qualcomm Snapdragon 810

- With the emergence of architecturally heterogeneous (and hence MCM-heterogeneous) systems, the problems are only going to get worse!

ArMOR Overview

- Goal: Take the guesswork out of specifying and analyzing memory consistency models (MCMs) within compilers, emulators, etc.
- Contributions:
  1. MOST: precise, portable, general-purpose **MCM specification format**
  2. MOST **analysis/manipulation** methodology that enables flexible compilation/translation/etc.
  3. Case study: automatic generation of **inter-MCM dynamic binary translation** modules ("shims")

Outline

- Why are MCMs inherently too complicated for simplified specifications like these?

Total Store Ordering (TSO)

A \ B	Ld	St
Ld	✓	✓
St	—	✓

- ArMOR solution: MOSTs
  - Memory Ordering Specification Tables

A \ B	Ld. Same Addr.	Ld. Diff. Addr.	St.
Load	✓	✓	✓
Store	✓$_L$	—	✓$_S$

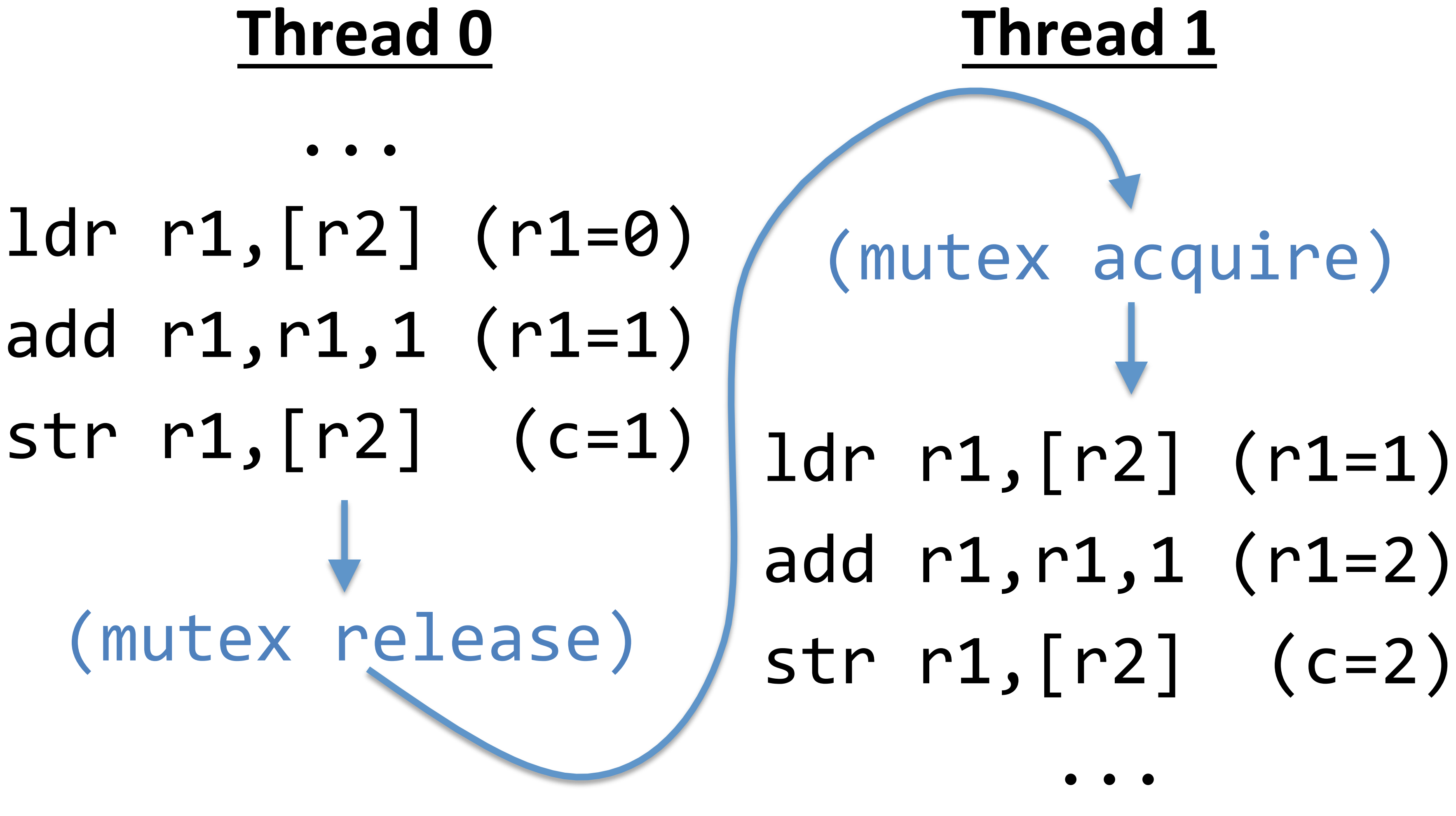
- Case study: cross-MCM dynamic binary translation

What Problems Can MCMs Cause?

```
public class Counter {
  private int c = 0;

  public void increment() {
    synchronized(this) { c++; }
  }
}
```

Wrap “c++;” in a mutex to make it thread-safe

What if Mutexes Fail to Address MCMs?



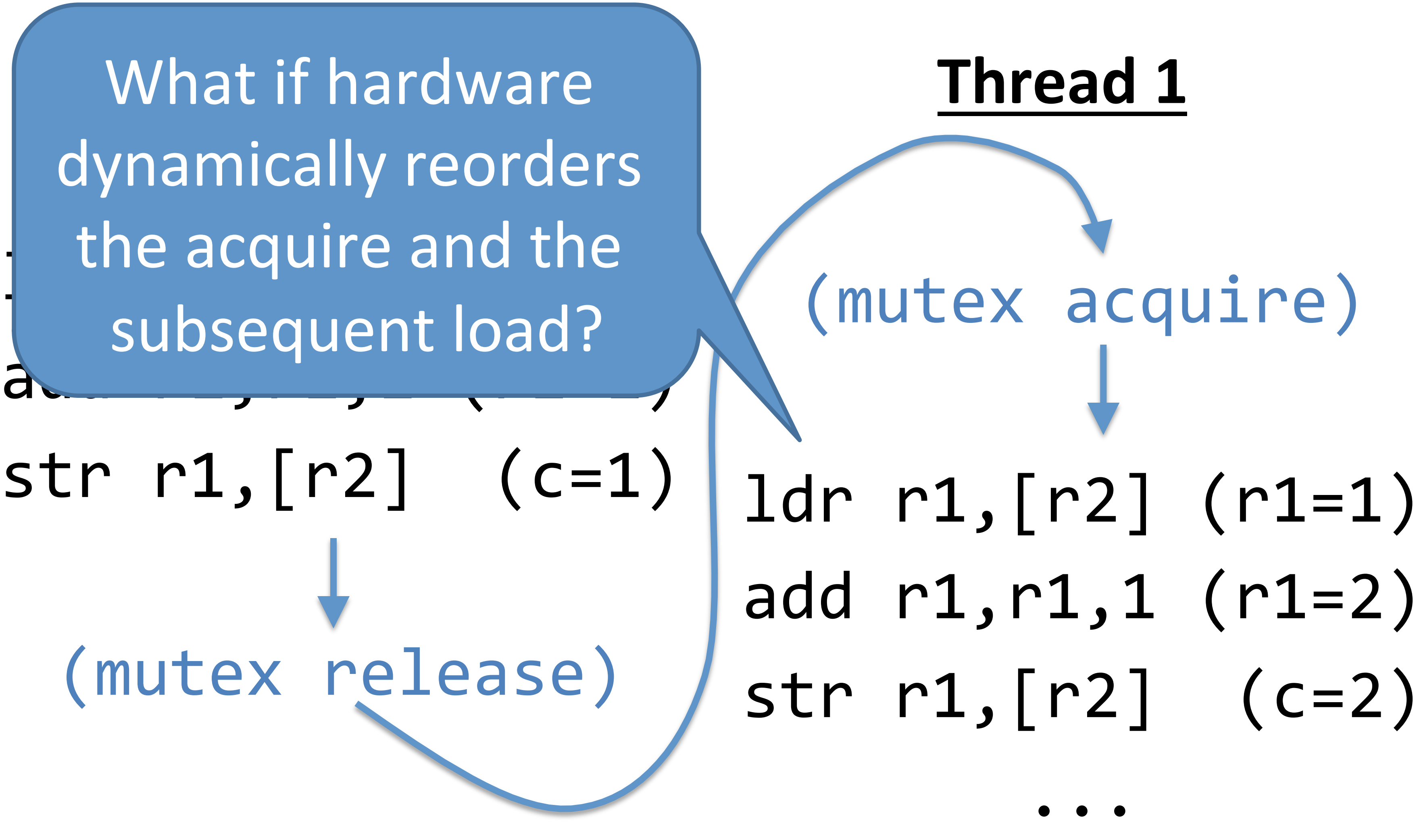
Thread 0

```
       ...
ldr r1,[r2] (r1=0)
add r1,r1,1 (r1=1)
str r1,[r2]  (c=1)

 (mutex release)
```

Thread 1

```
 (mutex acquire)

ldr r1,[r2] (r1=1)
add r1,r1,1 (r1=2)
str r1,[r2]  (c=2)
       ...
```

What if Mutexes Fail to Address MCMs?



What if hardware dynamically reorders the acquire and the subsequent load?

```
str r1,[r2]  (c=1)
```

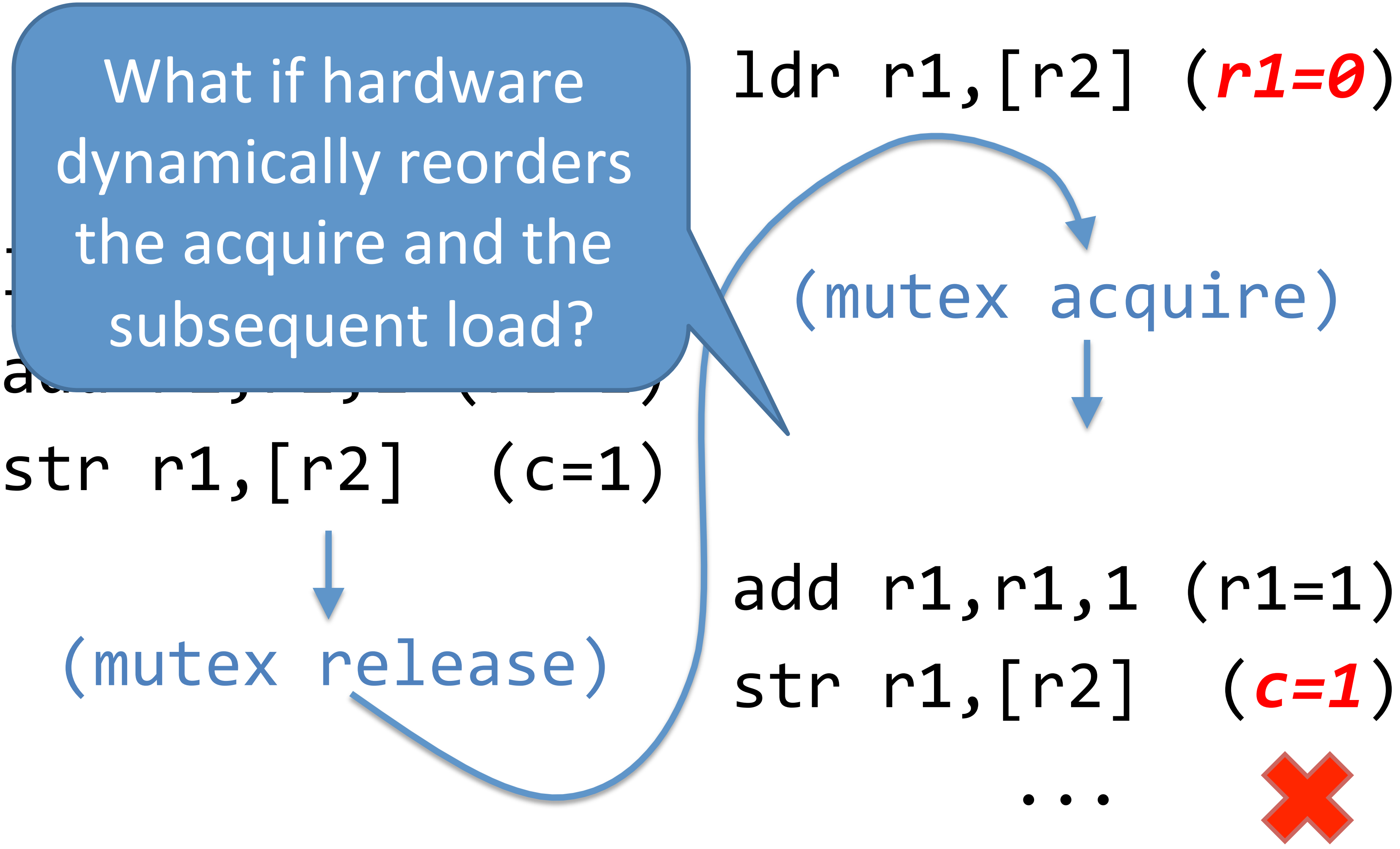
(mutex release)

Thread 1

(mutex acquire)

```
ldr r1,[r2] (r1=1)
add r1,r1,1 (r1=2)
str r1,[r2]  (c=2)
     ...
```

What if Mutexes Fail to Address MCMs?



What if hardware dynamically reorders the acquire and the subsequent load?

```
str r1,[r2]  (c=1)
```

(mutex release)

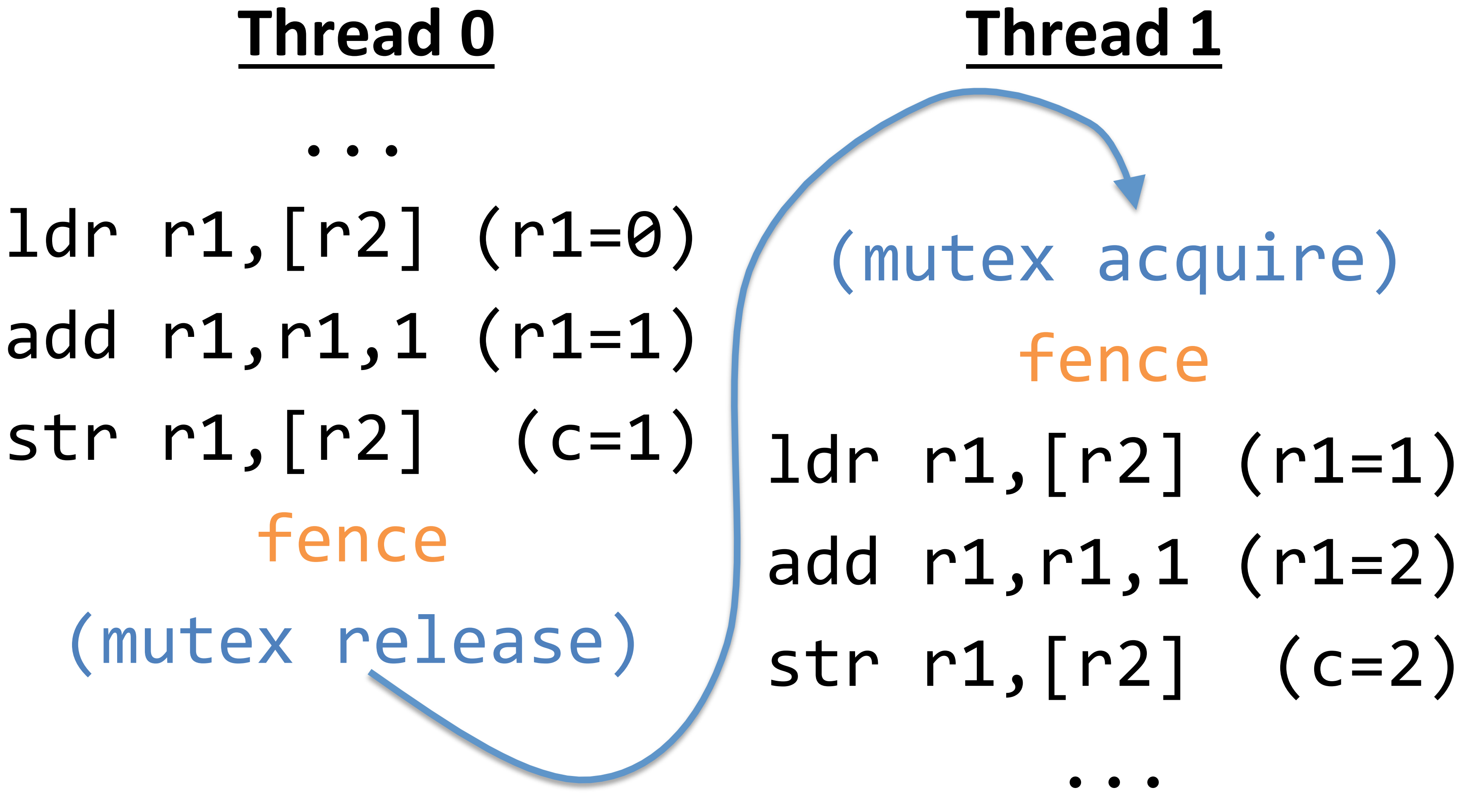
```
ldr r1,[r2] (r1=0)
```

(mutex acquire)

```
add r1,r1,1 (r1=1)
str r1,[r2]  (c=1)
...
```

✗

Mutexes Need Fences…But Which Ones?



Thread 0

```
      ...
ldr r1,[r2] (r1=0)
add r1,r1,1 (r1=1)
str r1,[r2]  (c=1)
     fence
 (mutex release)
```

Thread 1

```
 (mutex acquire)
      fence
ldr r1,[r2] (r1=1)
add r1,r1,1 (r1=2)
str r1,[r2]  (c=2)
       ...
```

Lustig et al., “ArMOR: Defending Against MCM Mismatches…”

Mutexes Need Fences…But Which Ones?

Need to choose fences with “any→store” and “load→any” semantics

```
ldr r1,[r2] (r1=0)
add r1,r1,1 (r1=1)
str r1,[r2]  (c=1)
 fence(any→store)
  (mutex release)
```

```
  (mutex acquire)
 fence(load→any)
ldr r1,[r2] (r1=1)
add r1,r1,1 (r1=2)
str r1,[r2]  (c=2)
        ...
```

(Source: Google ART Compiler)

The MCM Analysis Guessing Game

Requirements of *Acquire* Semantics:

fence(load→any)

B A	Ld	St
Ld	✔	✔
St	—	—

(Source: Google ART Compiler)

- Q: How to compile fence(load→any) for ARM?
- Consider two options:

```
Option 1:
dmb ishld
```

B A	Ld	St
Ld	✔	✔
St	—	—

```
Option 2:
dmb ish
```

A	Ld	St
Ld	✔	✔
St	✔	✔

Lustig et al., "ArMOR: Defending Against MCM Mismatches…"

The MCM Analysis Guessing Game

Requirements of *Acquire* Semantics:

fence(load→any)

B A	Ld	St
Ld	✔	✔
St	—	—

(Source: Google ART Compiler)

Although "`dmb ishld`" looks sufficient, it actually may be too weak! But why?

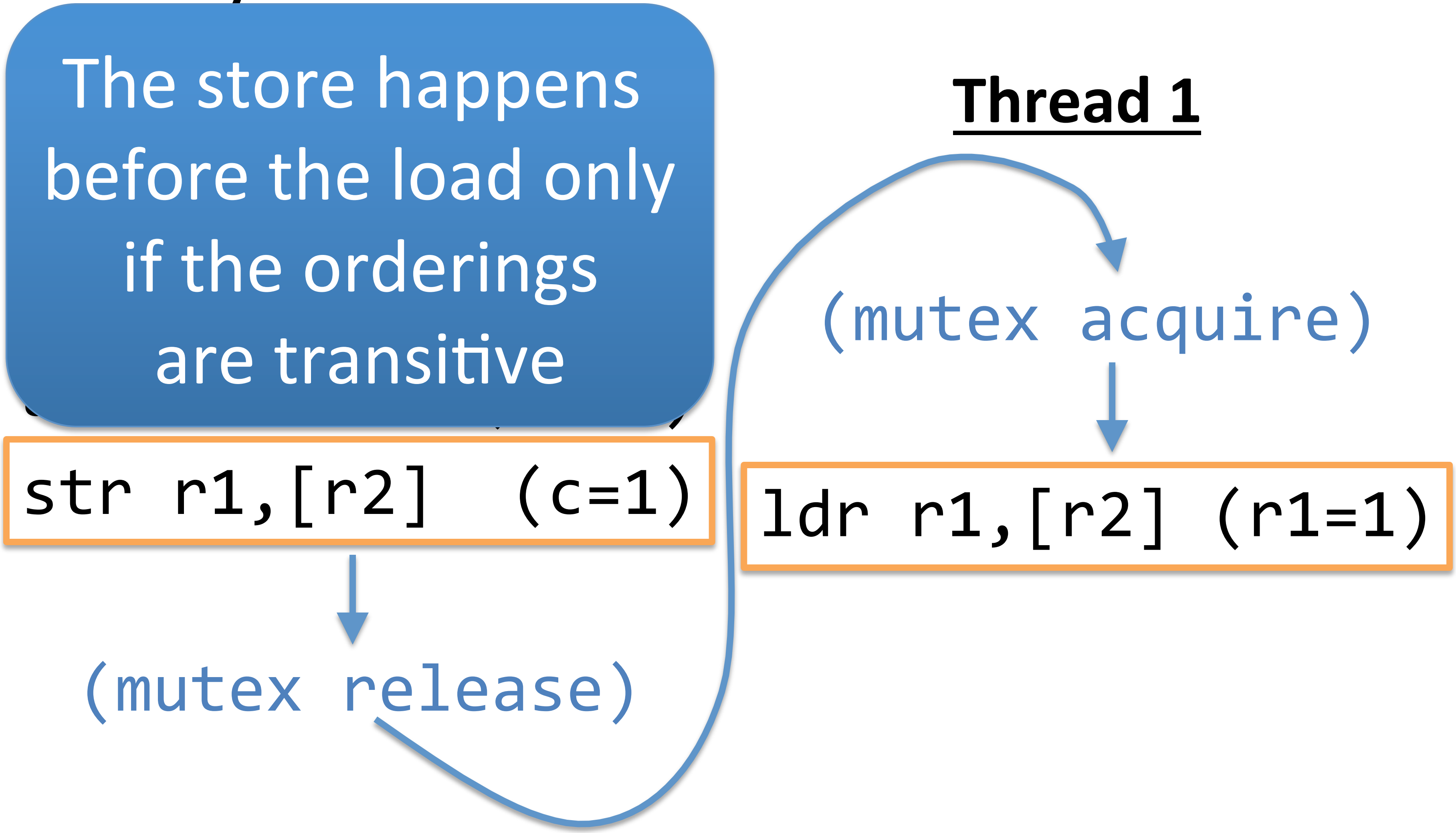
~~Option 1: `dmb ishld`~~

B A	Ld	St
Ld	✔	✔
St	—	—

Option 2: `dmb ish`

A	Ld	St
Ld	✔	✔
St	✔	✔

Why Low-Level MCM Details Matter



The store happens before the load only if the orderings are transitive

Thread 1

```
str r1,[r2]  (c=1)
```

(mutex release)

(mutex acquire)

```
ldr r1,[r2] (r1=1)
```

Why Low-Level MCM Details Matter

The store happens before the load only if the orderings are transitive

```
str r1,[r2]  (c=1)
fence(any→store)
(mutex release)
```

Thread 1

```
(mutex acquire)
fence(load→any)
ldr r1,[r2] (r1=1)
```

On ARM, fences must be "cumulative" to enforce transitivity

Why Low-Level MC

The store happens before the load only if the orderings are transitive

```
str r1,[r2]  (c=1)
fence(any→store)
(mutex release)
```

Need a ***cumulative*** fence which has “load→any” semantics

```
fence(load→any)
ldr r1,[r2] (r1=1)
```

On ARM, fences must be “cumulative” to enforce transitivity

ArMOR: Memory Ordering Specification Tables (MOSTs)

Requirements of *Acquire* Semantics:

fence(load→any)

B A	Ld	St
Ld	✔	✔
Cml.	✔	✔
St	—	—

Option 1: dmb ishld

B A	Ld	St
Ld	✔	✔
Cml.	?	?
St	—	—

Option 2: dmb ish

B A	Ld	St
Ld	✔	✔
Cml.	✔	✔
St	✔$_{S}$	✔$_{S}$

Key ArMOR insight: encode this kind of information directly into the specification tables

ArMOR: Memory Ordering

Hardware vendors are still responsible for providing correct and precise specifications!

fence(load→any)

A \ B	Ld	St
Ld	✓	✓
Cml.	✓	✓
St	—	—

Ld	✓	
Cml.	?	?
St	—	—

Ld	✓	✓
Cml.	✓	✓
St	✓ S	✓ S

Key ArMOR insight: encode this kind of information directly into the specification tables

ArMOR: Memory Ordering Specification Tables (MOSTs)

Requirements of *Acquire* Semantics: fence(load→any)

A \ B	Ld	St
Ld	✔	✔
Cml.	✔	✔
St	—	—

~~Option 1: dmb ishld~~

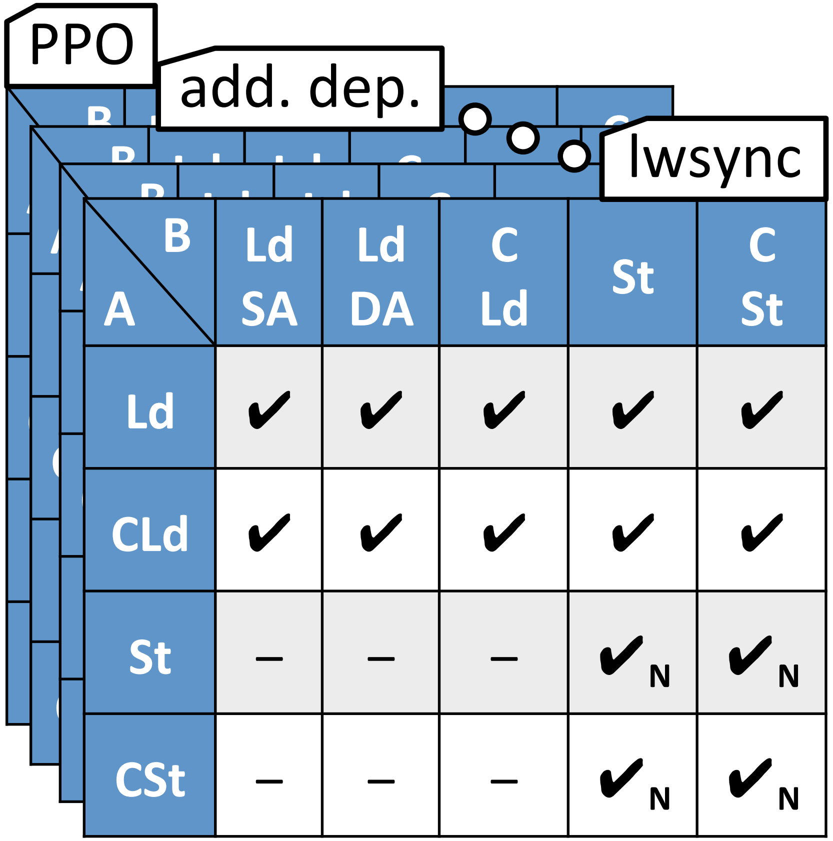
A \ B	Ld	St
Ld	✔	✔
Cml.	?	?
St	—	—

MOSTs make it clear why option 1 may be insufficient

Key ArMOR insight: encode this kind of information directly into the specification tables

Defining MCMs as Sets of MOSTs

- Each MCM is defined by its MOSTS:
  - PPO (default orderings)
  - Fences
  - Dependencies
- MOSTs enable automated, algorithmic analysis/comparison, ***even across MCMs***


PPO
add. dep.
lwsync

B / A	Ld SA	Ld DA	C Ld	St	C St
Ld	✔	✔	✔	✔	✔
CLd	✔	✔	✔	✔	✔
St	–	–	–	✔$_N$	✔$_N$
CSt	–	–	–	✔$_N$	✔$_N$

Power Memory Model

What are MOSTs Good For?

- To compile/map/JIT/translate/etc. from one MCM to another, need to be able to:
  - Compare MOSTs (<,≤,=,≠,≥,>)
  - Do MOST Arithmetic (+, -)
- ArMOR makes these analyses ***algorithmic***!

Seq. Cst.

A \ B	Ld. Same Addr.	Ld. Diff. Addr.	St.
Load	✔	✔	✔
Store	✔	✔	✔$_s$

−

zSeries

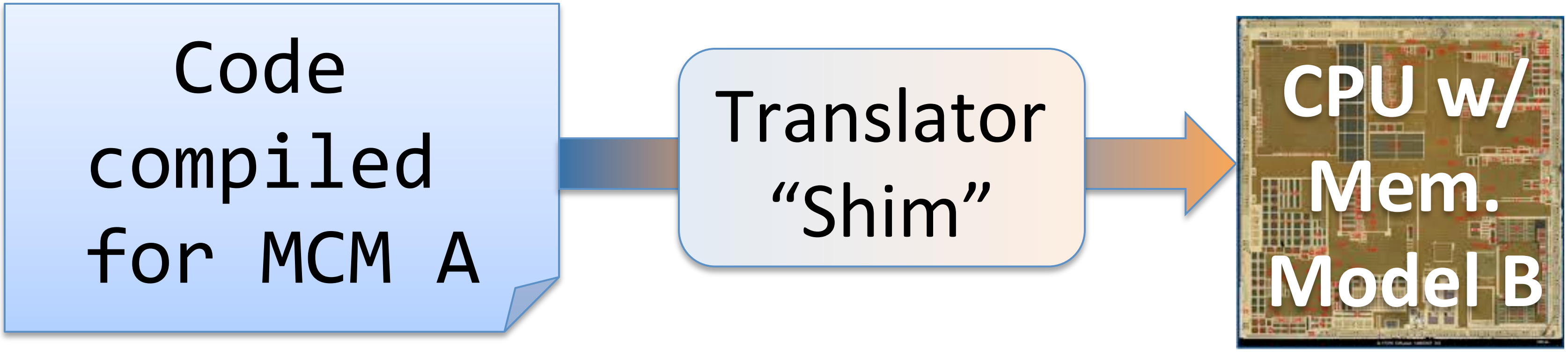
A \ B	Ld. Same Addr.	Ld. Diff. Addr.	St.
Load	✔	✔	✔
Store	✔	–	✔$_s$

=

"SC – zSeries"

A \ B	Ld. Same Addr.	Ld. Diff. Addr.	St.
Load	–	–	–
Store	–	✔	–

Case Study: Dynamic MCM Translation



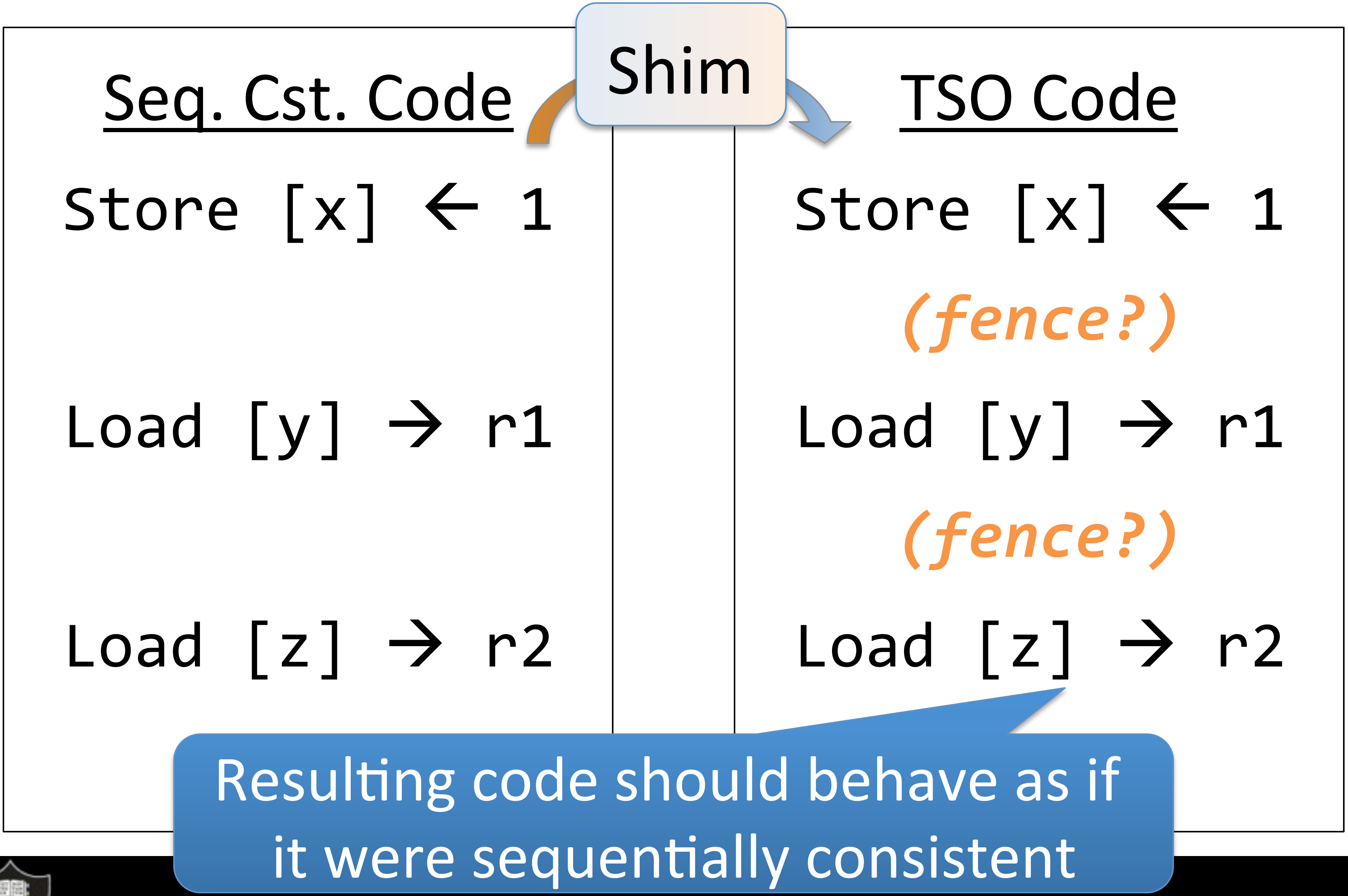
`Code compiled for MCM A`

Translator “Shim”

CPU w/ Mem. Model B

- Benefits: Performance/Energy/etc. [Venkat, ISCA’14]
- Challenges: differences in opcodes, memory layouts, calling conventions, etc. [DeVuyst, ASPLOS ’12]
- Most existing emulators/translators ***ignore MCMs*** and hence simply can’t do this today!

Case Study: Dynamic MCM Translation



Shim

Seq. Cst. Code

```
Store [x] ← 1

Load [y] → r1

Load [z] → r2
```

TSO Code

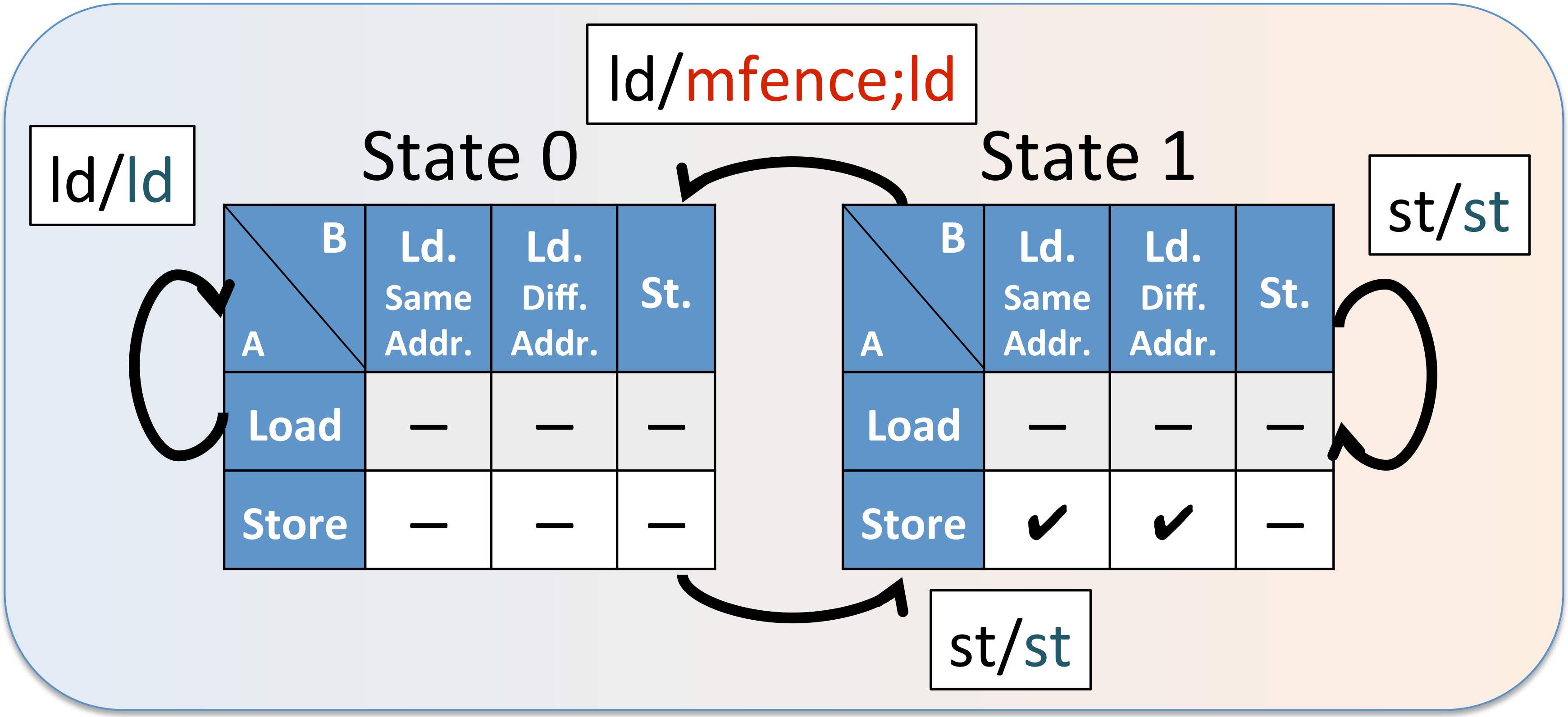
```
Store [x] ← 1
    (fence?)
Load [y] → r1
    (fence?)
Load [z] → r2
```

Resulting code should behave as if it were sequentially consistent

Lustig et al., "ArMOR: Defending Against MCM Mismatches…"

Case Study: Dynamic MCM Translation

ArMOR shim FSMs are ***automatically generated*** for ***any pairing of MCMs***



State 0

B A	Ld. Same Addr.	Ld. Diff. Addr.	St.
Load	—	—	—
Store	—	—	—

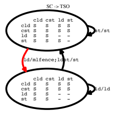
State 1

B A	Ld. Same Addr.	Ld. Diff. Addr.	St.
Load	—	—	—
Store	✔	✔	—

Lustig et al., “ArMOR: Defending Against MCM Mismatches…”

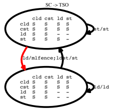
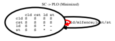
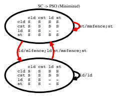
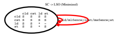
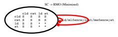
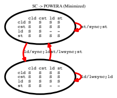
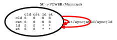
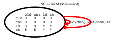
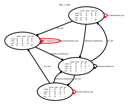
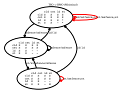
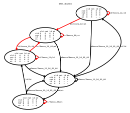
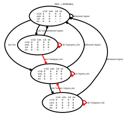
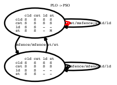
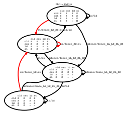
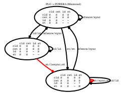
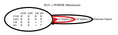
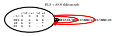
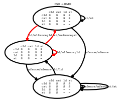
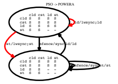
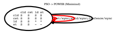
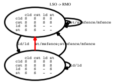
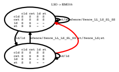
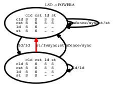
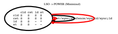
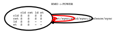
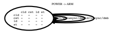
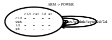
ArMOR Breadth of Applicability

- Translating from SC to TSO is just one example…

	TSO
SC	 SC -> TSO cld cst ld st cld S S S S cst S S S S ld S S - - st S S S - st/st ld/mlfence;ld st/st cld cst ld st cld S S S S cst S S S S ld S S - - st S S - - ld/ld

ArMOR Breadth of Applicability

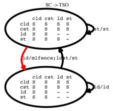
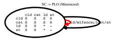
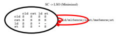
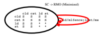
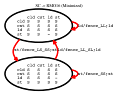
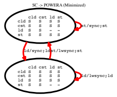
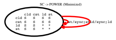
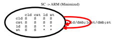
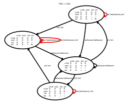
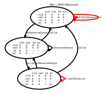
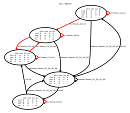
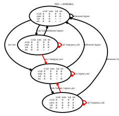
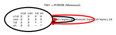
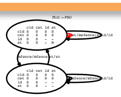
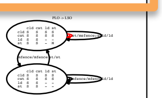
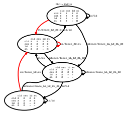
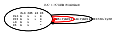
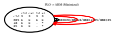
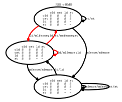
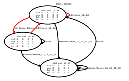
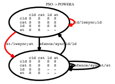
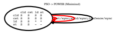
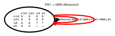
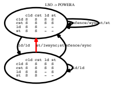
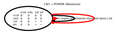
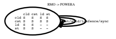
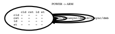
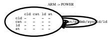
- ArMOR makes all of these scenarios possible!

	TSO	PLO	PSO	LSO	RMO	RMO16	POWERA	POWER	ARM
SC									
TSO	-								
PLO	-	-							
PSO	-		-						
LSO	-	-	-	-					
RMO	-	-	-	-	-				
POWERA	-	-	-	-			-		
POWER	-	-	-	-	-	-	-	-	
ARM	-	-	-	-	-	-	-		-

Lustig et al., “ArMOR: Defending Against MCM Mismatches…”

ArMOR Breadth of Applicability

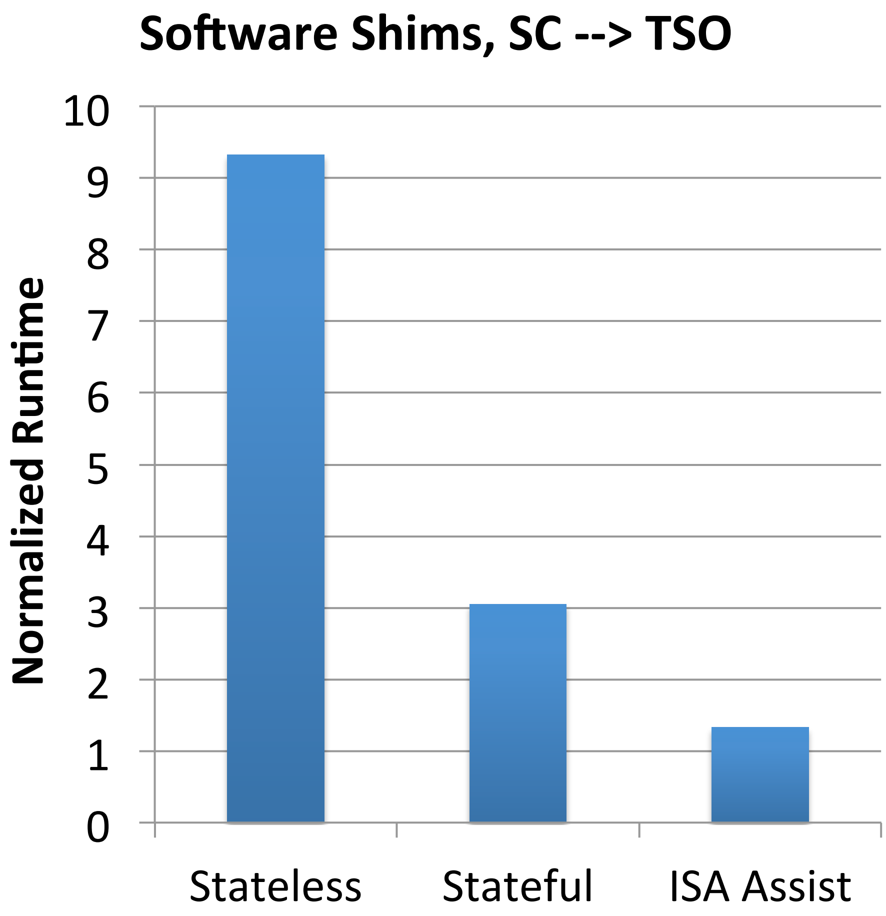
- ArMOR makes all of these scenarios possible!

	TSO	PLO	PSO	LSO	RMO	RMO16	POWERA	POWER	ARM
SC									
TSO	-								
PLO	-	-							
PSO	-		-						
LSO	-	-	-	-					
RMO	-	-	-	-	-				
POWERA	-	-	-	-			-		
POWER	-	-	-	-	-	-	-	-	
ARM	-	-	-	-	-	-	-		-

Lustig et al., “ArMOR: Defending Against MCM Mismatches…”

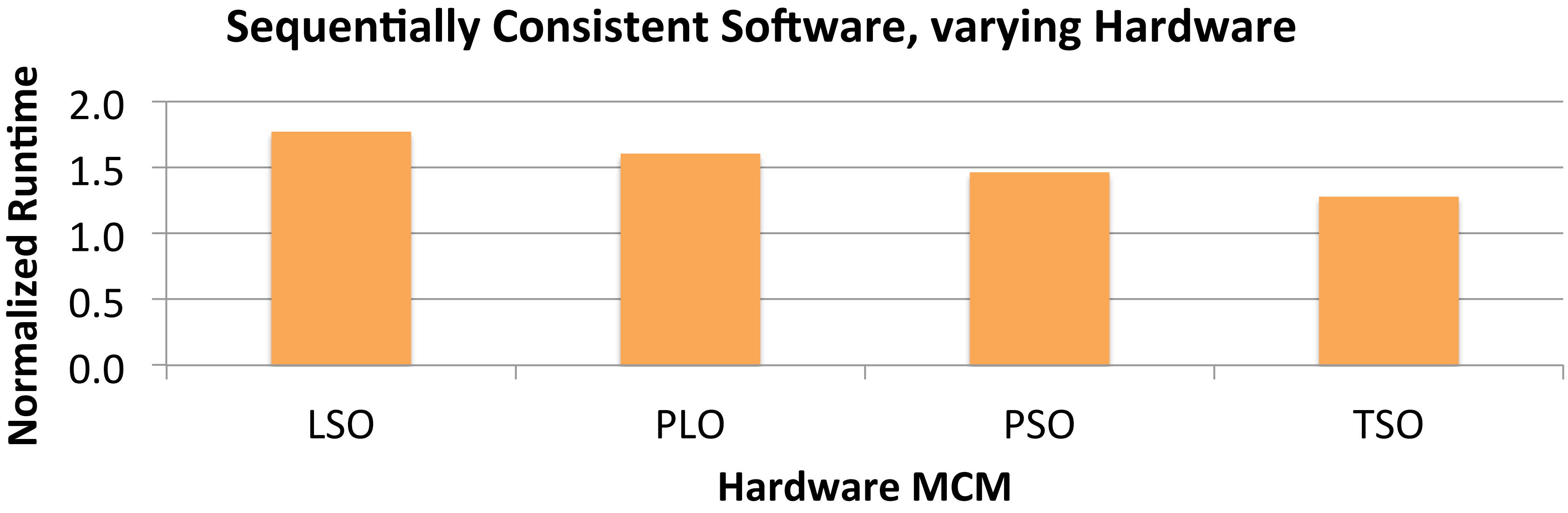
Software Shim Performance Analysis

- Implemented using Intel Pin
- Three shim designs:
  - Naïve/Stateless
  - Stateful
  - Stateful + ISA assist:
    - ignore thread-private and DRF accesses
- PARSEC benchmarks



Hardware Shim Performance Analysis

- Small FSMs placed into issue queue of gem5 simulator out-of-order pipeline
- Takeaway: low area/performance overhead, easily adapts to any SW/HW combination!



Conclusion

- MCMs are not a solved problem, but ArMOR is a major step towards improving the situation
- ArMOR's MOST framework enables systematic and algorithmic MCM analysis
- ArMOR adds precision to existing use cases (e.g., compiler analysis) and enables forward-looking use cases (e.g., inter-MCM translation)
- Easily adapts to any HW/SW combination or any use case, even those which don't yet exist!

ArMOR: Defending Against Memory Consistency Model Mismatches in Heterogeneous Architectures

Daniel Lustig, Caroline Trippel, Michael Pellauer, and Margaret Martonosi

Source code and 54 page gallery of MOSTs and shims: www.princeton.edu/~dlustig

“Armour of Sir James Scudamore (1595)”, Todd Hoogerland, CC-BY-SA-3.0
http://commons.wikimedia.org/wiki/File:Scudamorearmor2.jpg