arm Research

The Semantics of Transactions and Weak Memory

in x86, Power, ARM, and C++

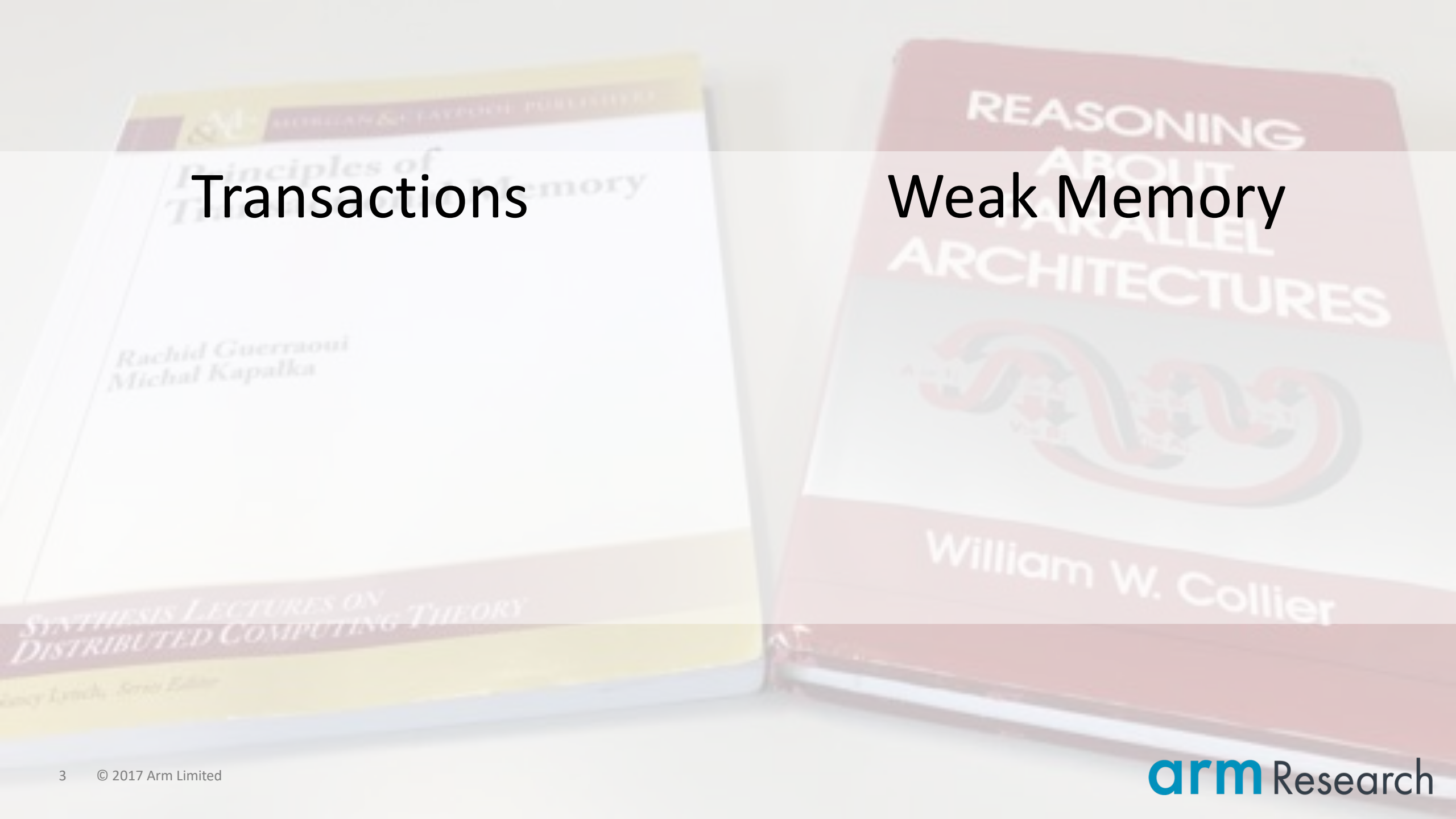
Nathan Chong, Tyler Sorensen, John Wickerson

June 2018

Tyler Sorensen

John Wickerson

arm Research



Transactions

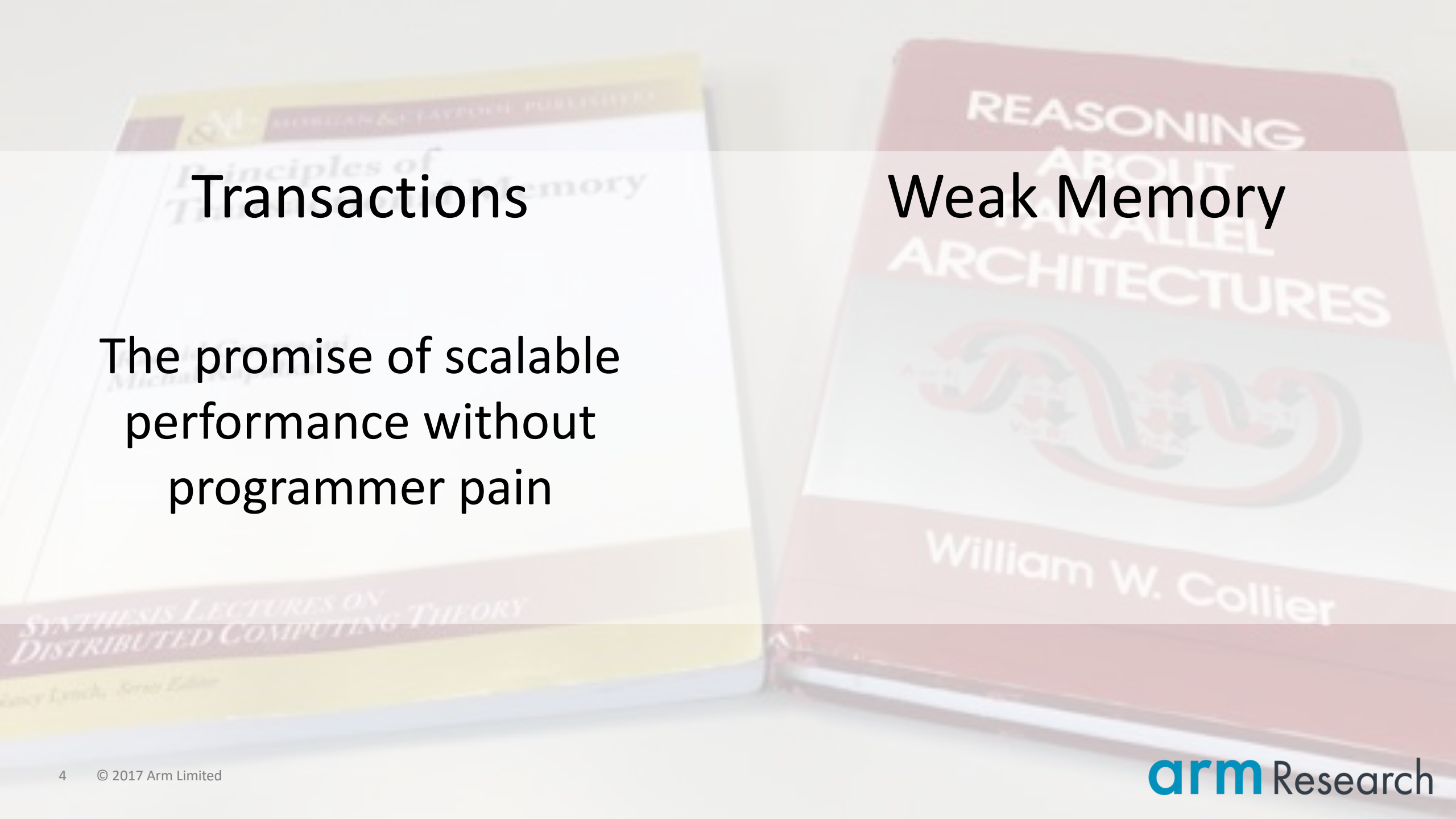
Weak Memory

arm Research

Transactions

The promise of scalable performance without programmer pain

Weak Memory



arm Research

Transactions

The promise of scalable performance without programmer pain

Weak Memory

Making sense of micro-architecture that breaks programmer intuition

Contributions

Clarify interplay between transactions and weak memory

for x86, Power, Armv8, and C++
using axiomatic semantics and automated tool support

Resulting in the discovery of

Unsoundness of lock elision wrt an Armv8 spinlock impl. (this talk)
Ambiguity in Power TM specification
Proposed simplification to C++ TM specification
… (*more in paper*)

TM-aware Memalloy [Wickerson et al., POPL 2017]

Axiomatic Armv8 model with TM

+

Lock elision

+

Armv8 spinlock impl.

TM-aware Memalloy [Wickerson et al., POPL 2017]

arm Research

Axiomatic Armv8 model with TM

+

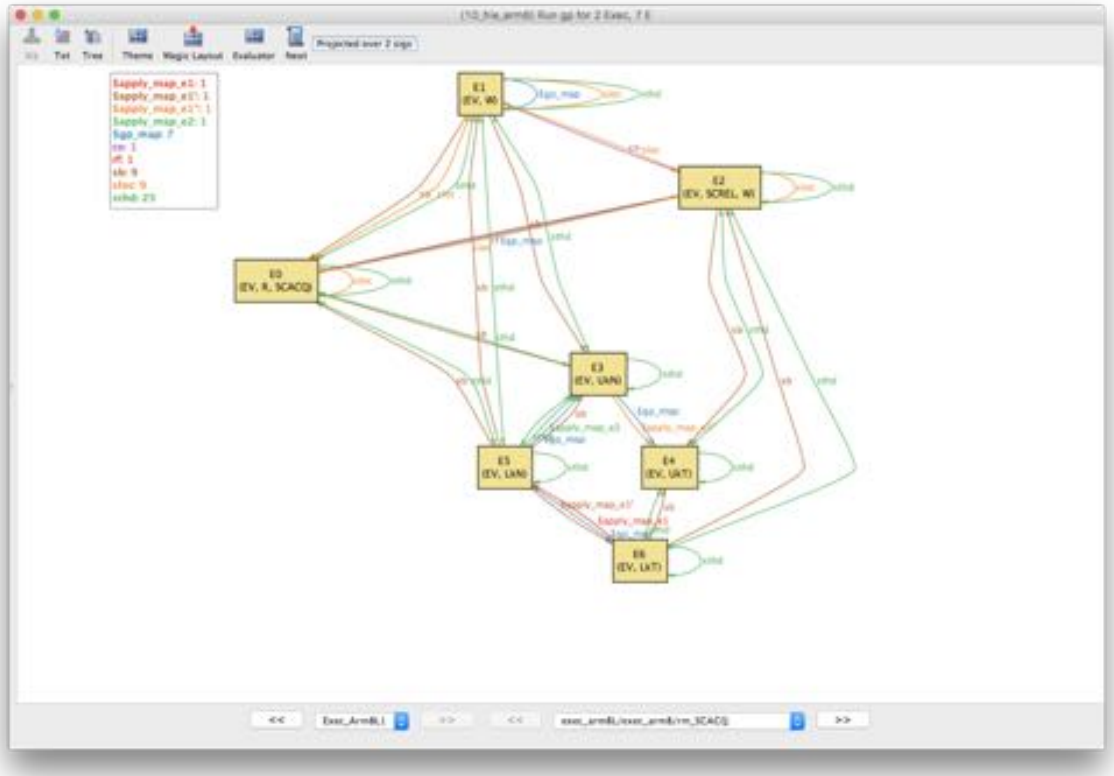
Lock elision

+

Armv8 spinlock impl.

↓

TM-aware Memalloy [Wickerson et al., POPL 2017]



Counterexample

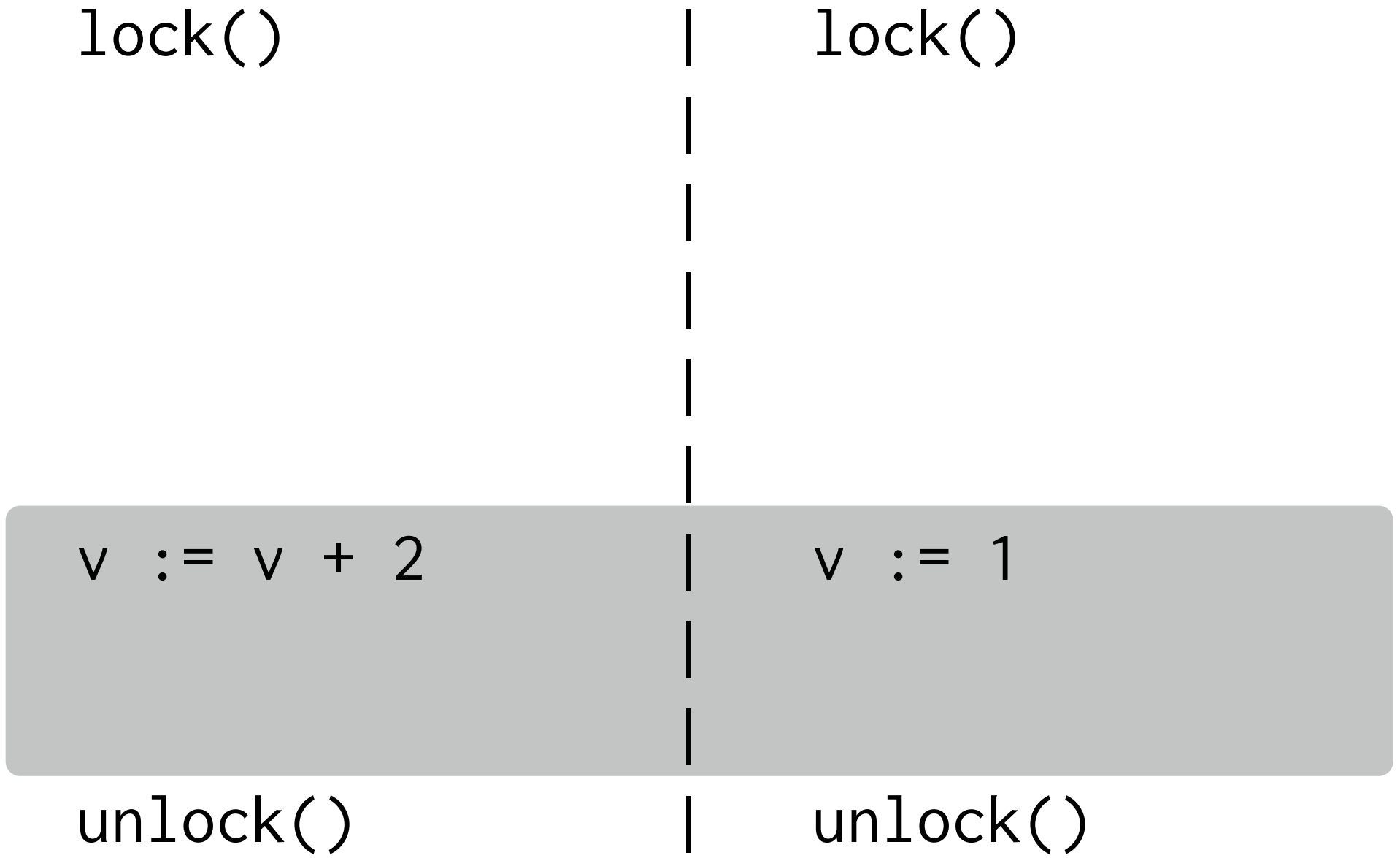
```
// Initially, v == 0

P0              |  P1
lock()          |  lock()
v := v + 2      |  v := 1
unlock()        |  unlock()

// Can v == 2?
// A violation of mutual exclusion
```

```
// Initially, v == 0

P0              |  P1
lock()          |  lock()
v := v + 2      |  v := 1
unlock()        |  unlock()

// Can v == 2?
// A violation of mutual exclusion
```

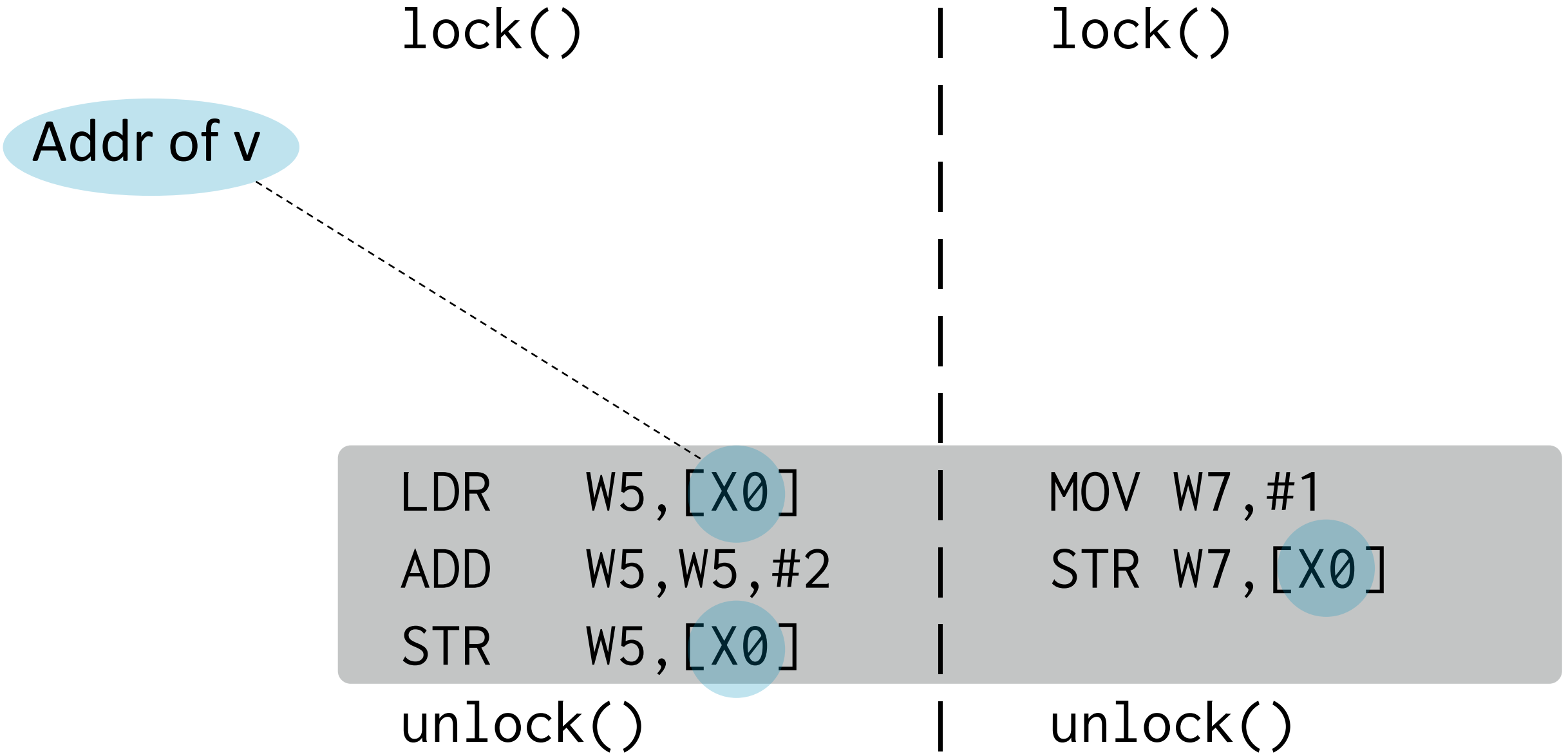
```
Serialise:
  // v == 0
  v := v + 2
  v := 1
  // v == 1
```

```
// Initially, v == 0

P0              |  P1
lock()          |  lock()
v := v + 2      |  v := 1
unlock()        |  unlock()

// Can v == 2?
// A violation of mutual exclusion
```

```
Serialise:
  // v == 0
  v := 1
  v := v + 2
  // v == 3
```



arm Research



arm Research

Lock elision [Rajwar and Goodman, MICRO 2001]

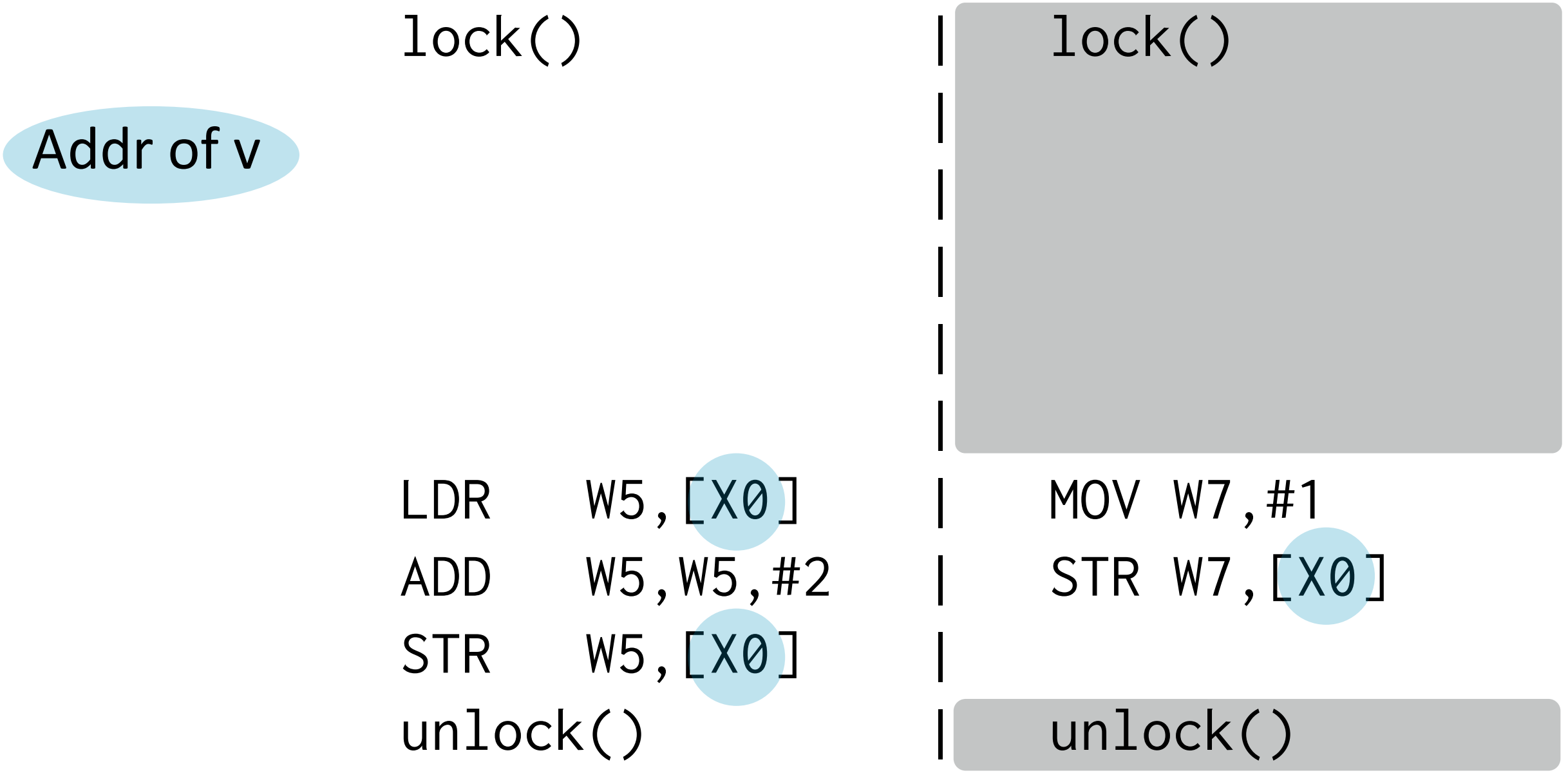
```
lock()
<crit>
unlock()
```

➔

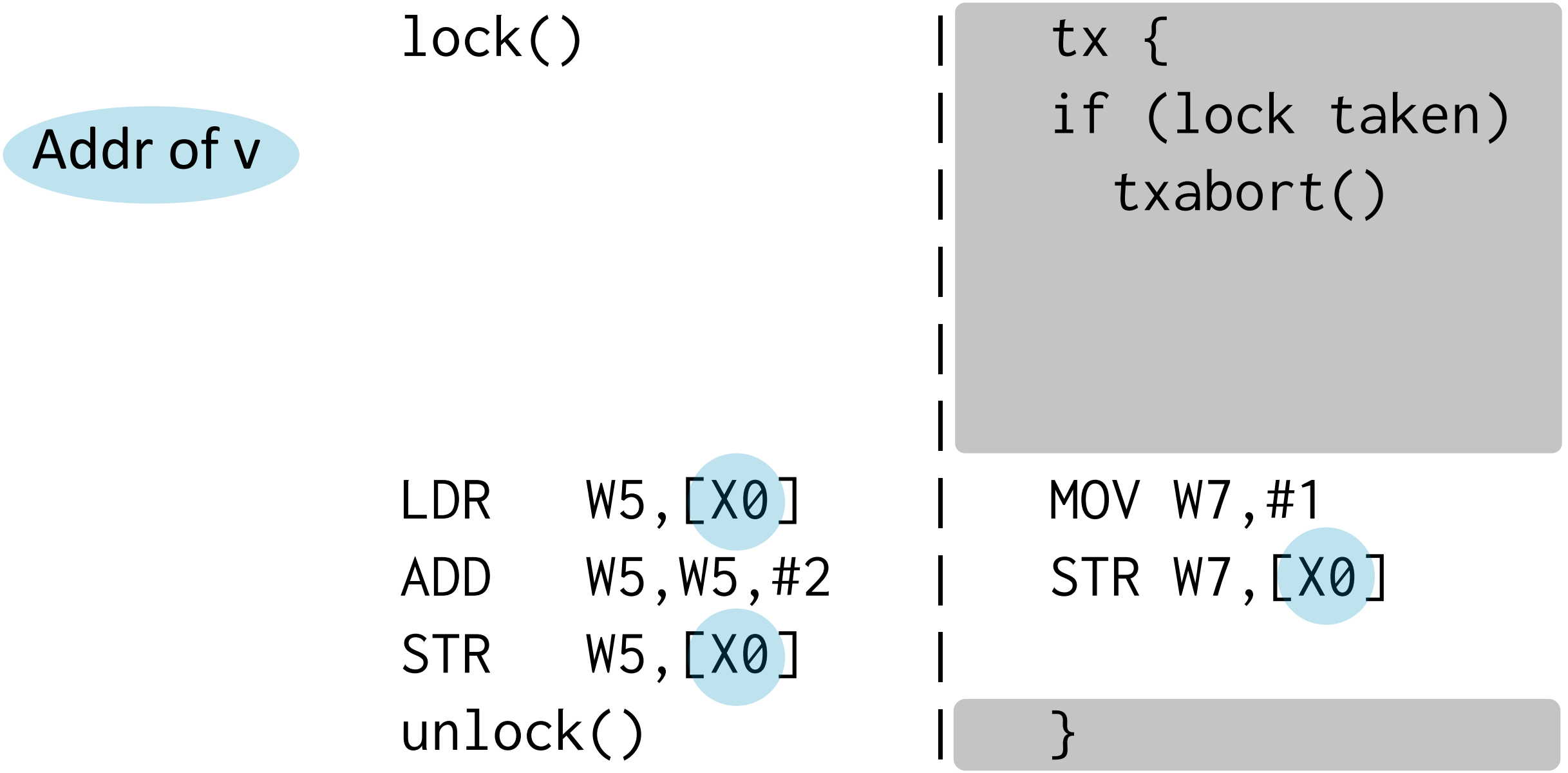
```
tx {
  if (lock taken) txabort()
  <crit>
}
```

Add lock variable to read-set

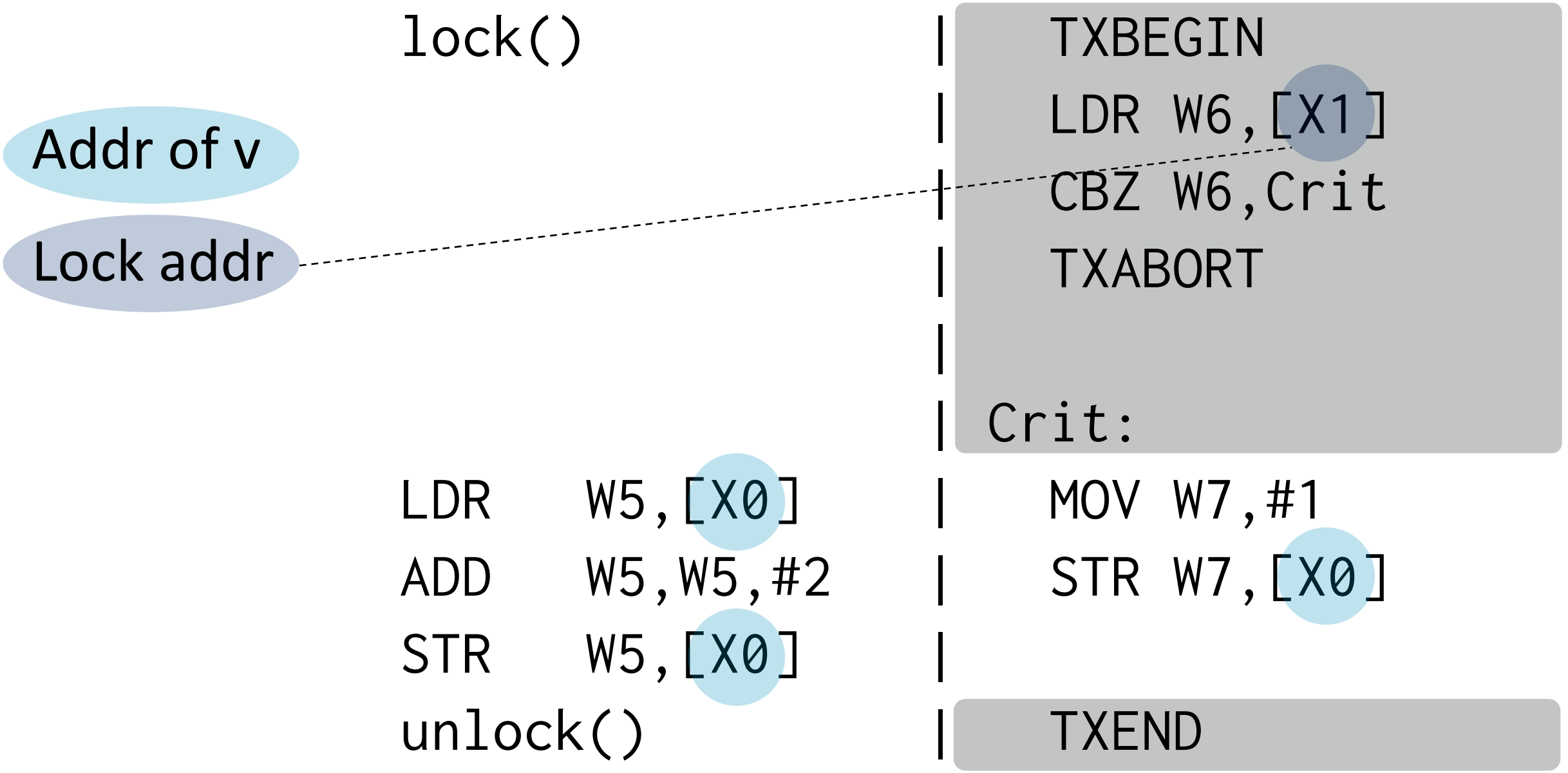
arm Research



arm Research



arm Research



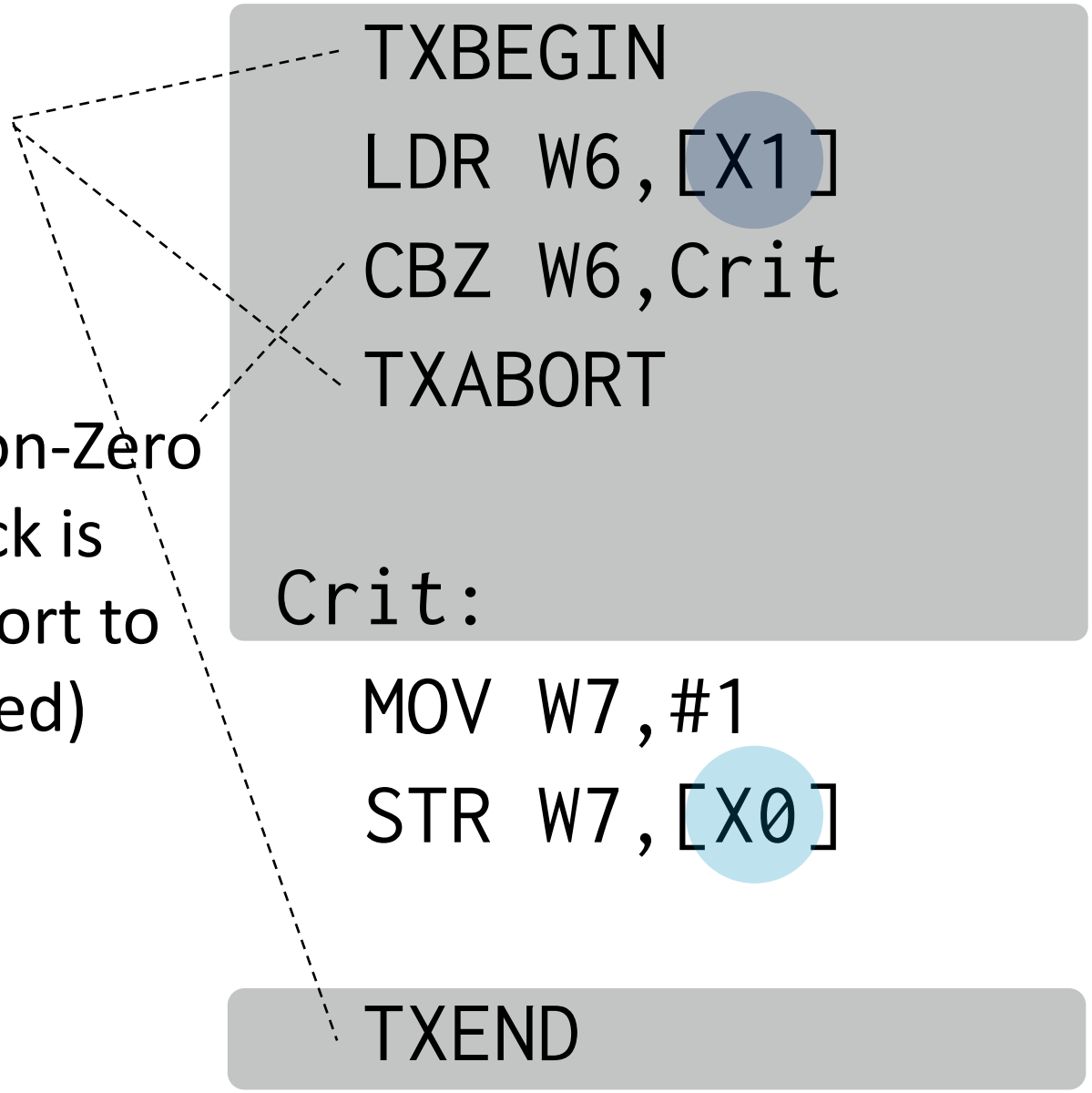
arm Research

Addr of v

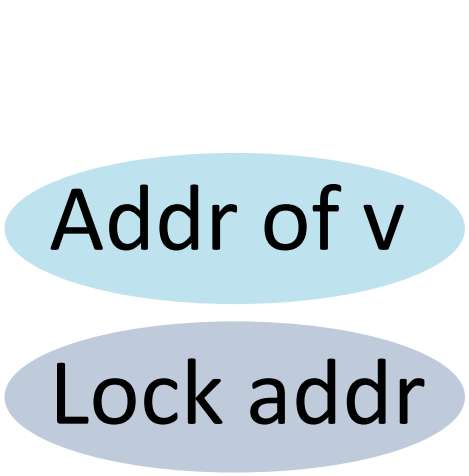
Lock addr

Hypothetical, but representative TM instructions

Compare-Branch-on-Zero
Jump to `Crit` if lock is free; otherwise abort to fail-handler (omitted)



arm Research




```
lock()                  |    TXBEGIN
                        |    LDR W6,[X1]
                        |    CBZ W6,Crit
                        |    TXABORT
                        |
                        | Crit:
LDR    W5,[X0]          |    MOV W7,#1
ADD    W5,W5,#2         |    STR W7,[X0]
STR    W5,[X0]          |
unlock()                |    TXEND
```


arm Research

Arm spinlock [Arm arch. reference manual, K9.3]

Arm spinlock [Arm arch. reference manual, K9.3]

Lock addr

```
lock()
<crit>
unlock()
```

➔

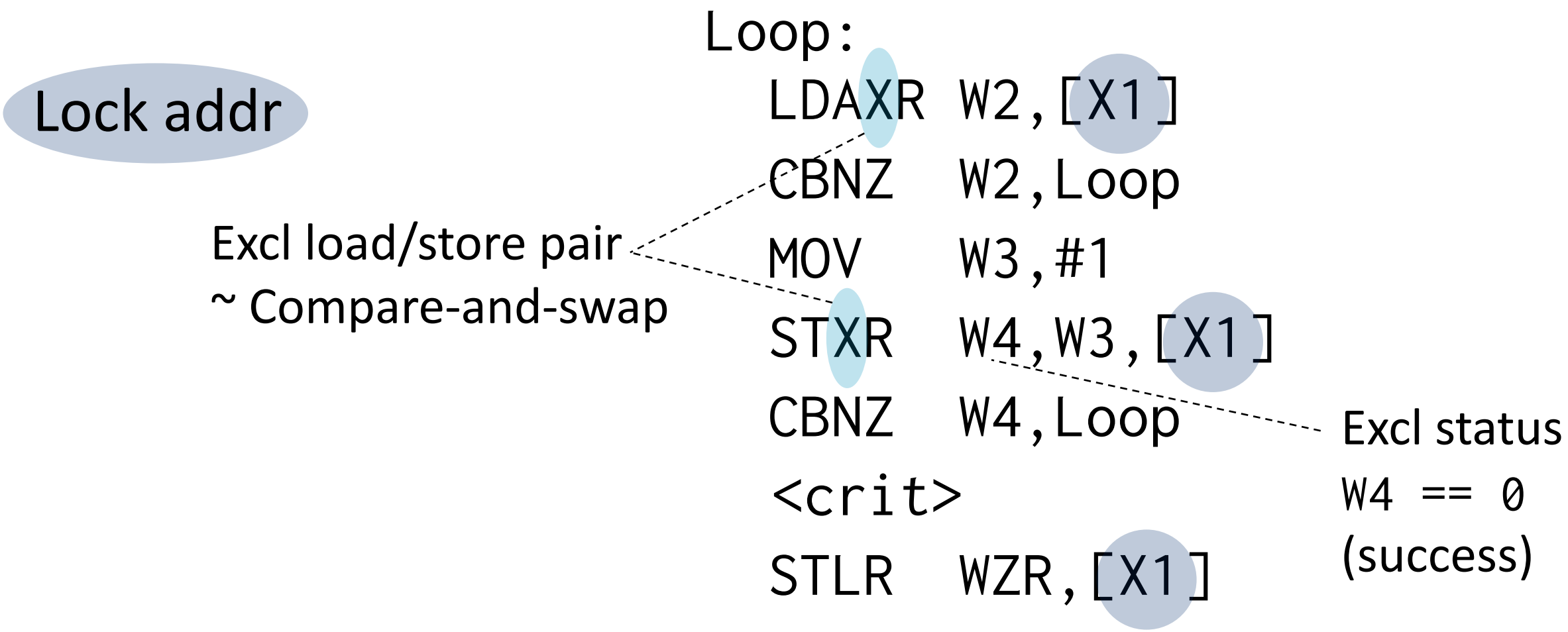
```
Loop:
    LDAXR  W2,[X1]
    CBNZ   W2,Loop
    MOV    W3,#1
    STXR   W4,W3,[X1]
    CBNZ   W4,Loop
    <crit>
    STLR   WZR,[X1]
```

Atomically update lock from free to taken

Unlock

arm Research

Arm spinlock [Arm arch. reference manual, K9.3]



arm Research

Arm spinlock [Arm arch. reference manual, K9.3]

Store-excl succeeds if it is the *immediate coherence successor* of the write read-from by the load-excl [Sarkar et al., PLDI 2012]

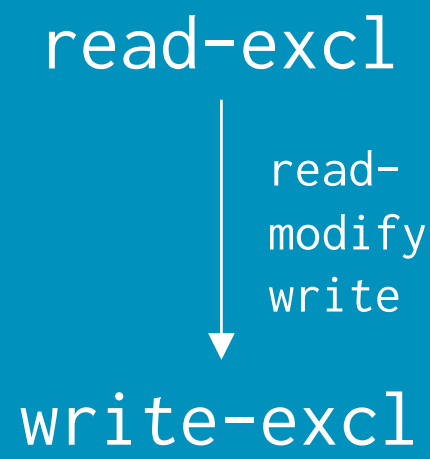
```
STLR   WZR,[X1]
```

(success)

arm Research

Arm spinlock [Arm arch. reference manual, K9.3]

Store-excl succeeds if it is the *immediate coherence successor* of the write read-from by the load-excl [Sarkar et al., PLDI 2012]



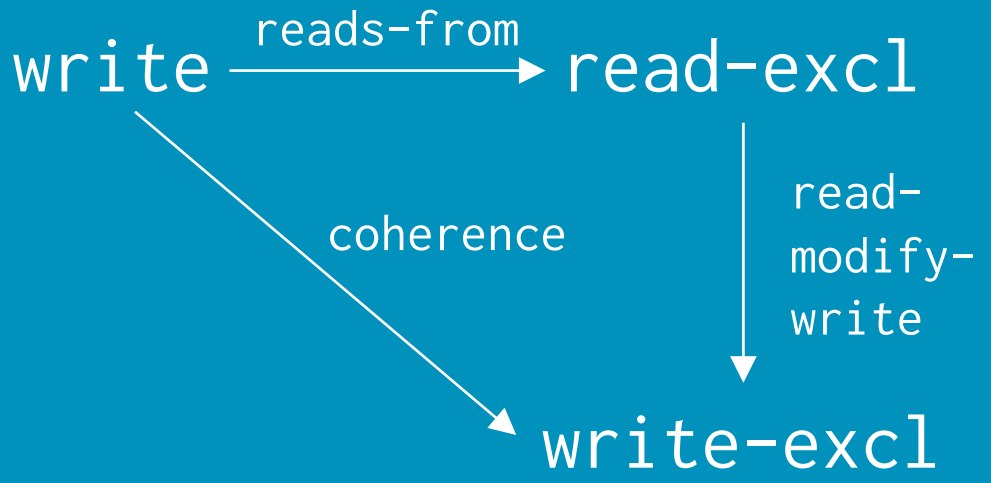
STLR WZR,[X1]

(success)

arm Research

Arm spinlock [Arm arch. reference manual, K9.3]

Store-excl succeeds if it is the *immediate coherence successor* of the write read-from by the load-excl [Sarkar et al., PLDI 2012]



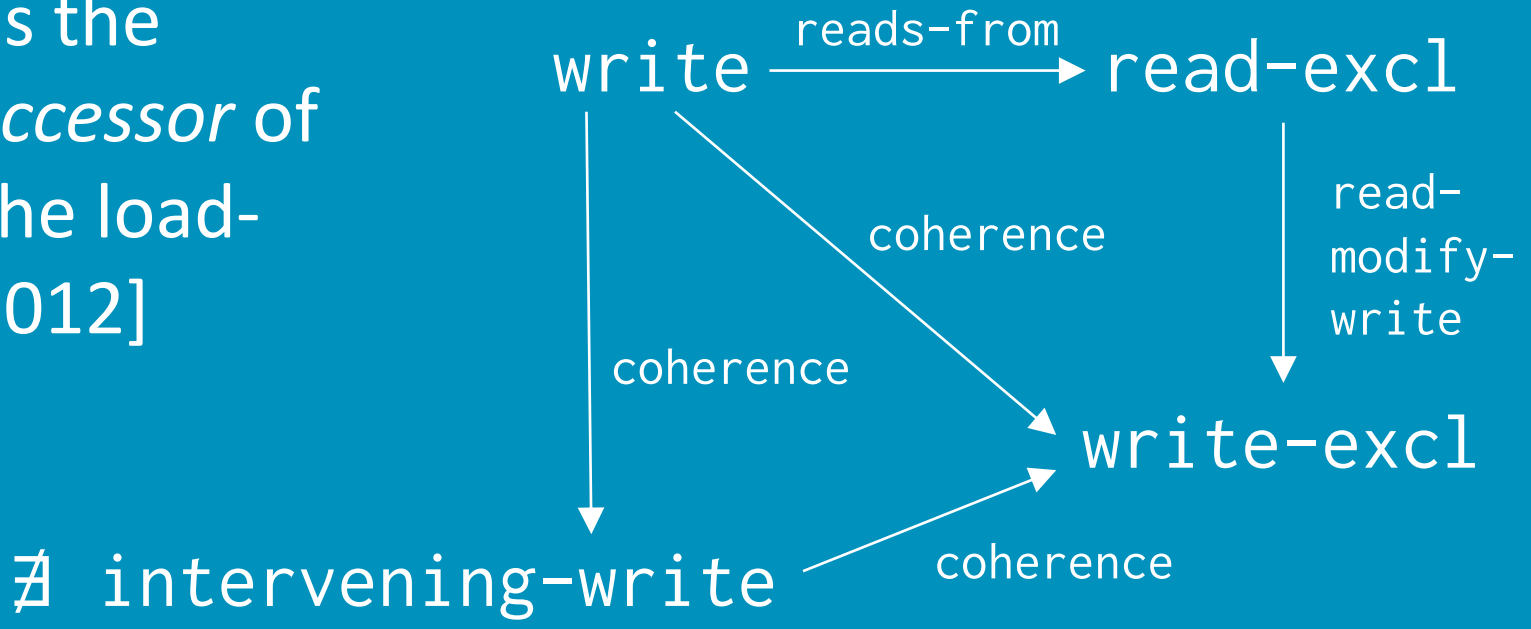
STLR WZR,[X1]

(success)

arm Research

Arm spinlock [Arm arch. reference manual, K9.3]

Store-excl succeeds if it is the *immediate coherence successor* of the write read-from by the load-excl [Sarkar et al., PLDI 2012]

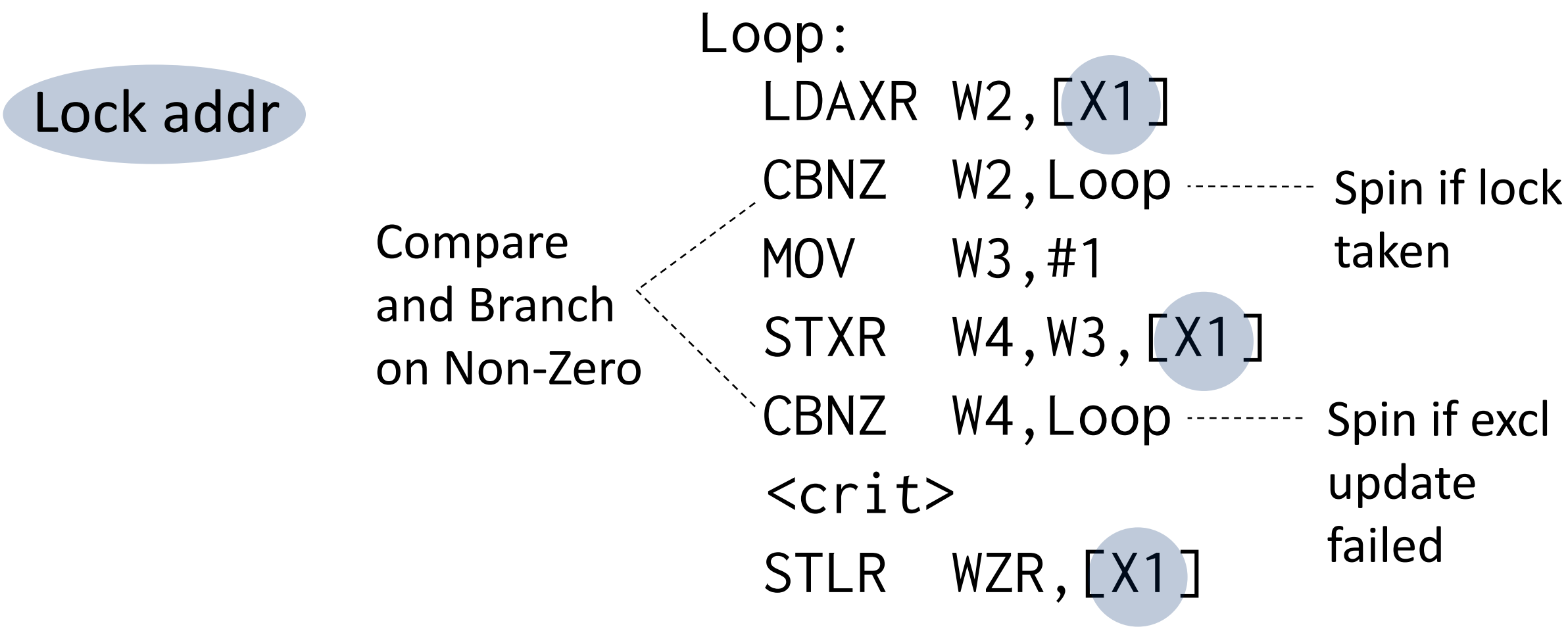


STLR WZR,[X1]

(success)

arm Research

Arm spinlock [Arm arch. reference manual, K9.3]



arm Research

Arm spinlock [Arm arch. reference manual, K9.3]

Lock addr

Unlock by writing 0; WZR = zero register

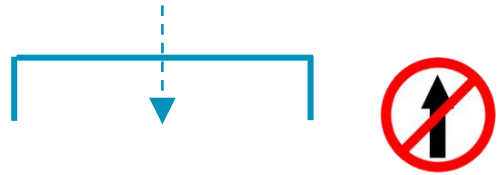
```
Loop:
    LDAXR  W2,[X1]
    CBNZ   W2,Loop
    MOV    W3,#1
    STXR   W4,W3,[X1]
    CBNZ   W4,Loop
    <crit>
    STLR   WZR,[X1]
```


arm Research

Arm spinlock [Arm arch. reference manual, K9.3]

Lock addr

```
Loop:
   LDAXR W2,[X1]
   CBNZ  W2,Loop
   MOV   W3,#1
   STXR  W4,W3,[X1]
   CBNZ  W4,Loop
   <crit>
   STLR  WZR,[X1]
```



Acquire/Release
~ “half barriers”
RCsc [Gharachorloo et al, ISCA 1990]

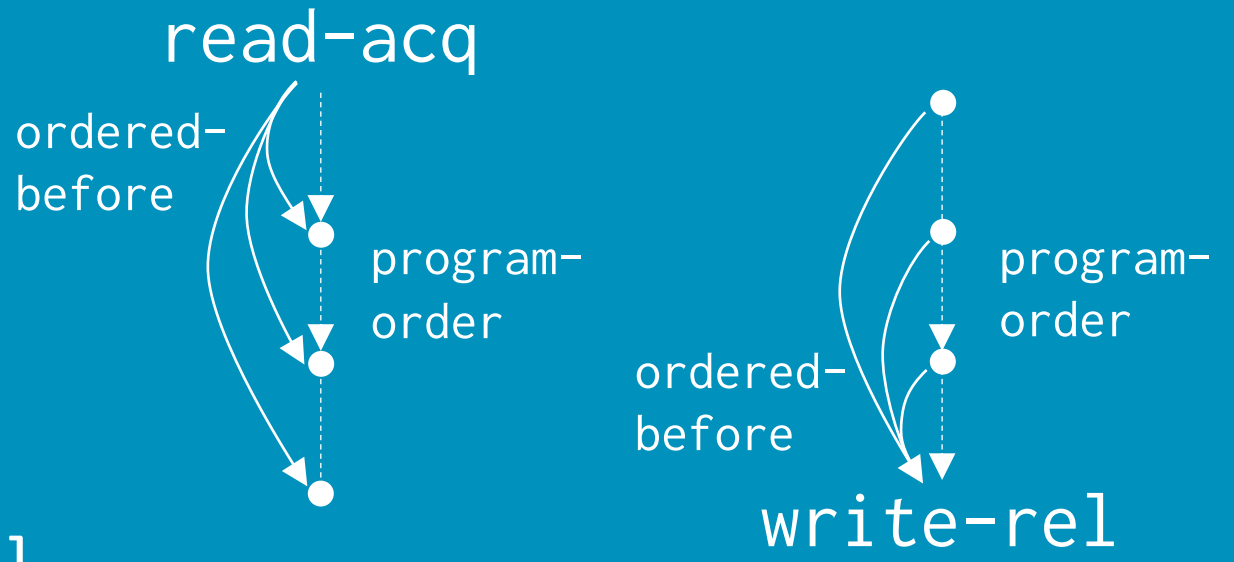
arm Research

Arm spinlock [Arm arch. reference manual, K9.3]

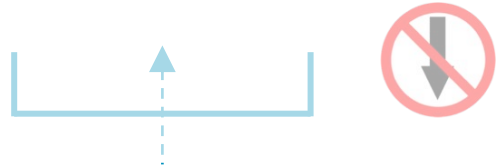
Read-acquire *ordered-before* any program-order successor

Any program-order predecessor *ordered-before* a write-release

[Arm arch. reference manual, B2.3]



STLR WZR,[X1]



arm Research

arm Research

Addr of v

Lock addr

```
Loop:                          |     TXBEGIN
    LDAXR  W2,[X1]             |     LDR W6,[X1]
    CBNZ   W2,Loop             |     CBZ W6,Crit
    MOV    W3,#1               |     TXABORT
    STXR   W4,W3,[X1]          |
    CBNZ   W4,Loop             |  Crit:
    LDR    W5,[X0]             |     MOV W7,#1
    ADD    W5,W5,#2            |     STR W7,[X0]
    STR    W5,[X0]             |
    STLR   WZR,[X1]            |     TXEND
```


arm Research

```
lock()          |  lock()
v := v + 2      |  v := 1
unlock()        |  unlock()
```

A program combining
transactions and weak memory

Can v == 2 (violate mutual exclusion)?

v = 0

lock = 0

```
Loop:                    |    TXBEGIN
   LDAXR W2,[X1]         |    LDR W6,[X1]
   CBNZ  W2,Loop         |    CBZ W6,Crit
   MOV   W3,#1           |    TXABORT
   STXR  W4,W3,[X1]      |
   CBNZ  W4,Loop         | Crit:
   LDR   W5,[X0]         |    MOV W7,#1
   ADD   W5,W5,#2        |    STR W7,[X0]
   STR   W5,[X0]         |
   STLR  WZR,[X1]        |    TXEND
```


arm Research

v = 0

lock = 0

W2 = 0

```
Loop:                          |    TXBEGIN
1 LDAXR W2,[X1]                |    LDR W6,[X1]
  CBNZ  W2,Loop                |    CBZ W6,Crit
  MOV   W3,#1                  |    TXABORT
  STXR  W4,W3,[X1]             |
  CBNZ  W4,Loop                | Crit:
  LDR   W5,[X0]                |    MOV W7,#1
  ADD   W5,W5,#2               |    STR W7,[X0]
  STR   W5,[X0]                |
  STLR  WZR,[X1]               |    TXEND
```


arm Research

v = 0

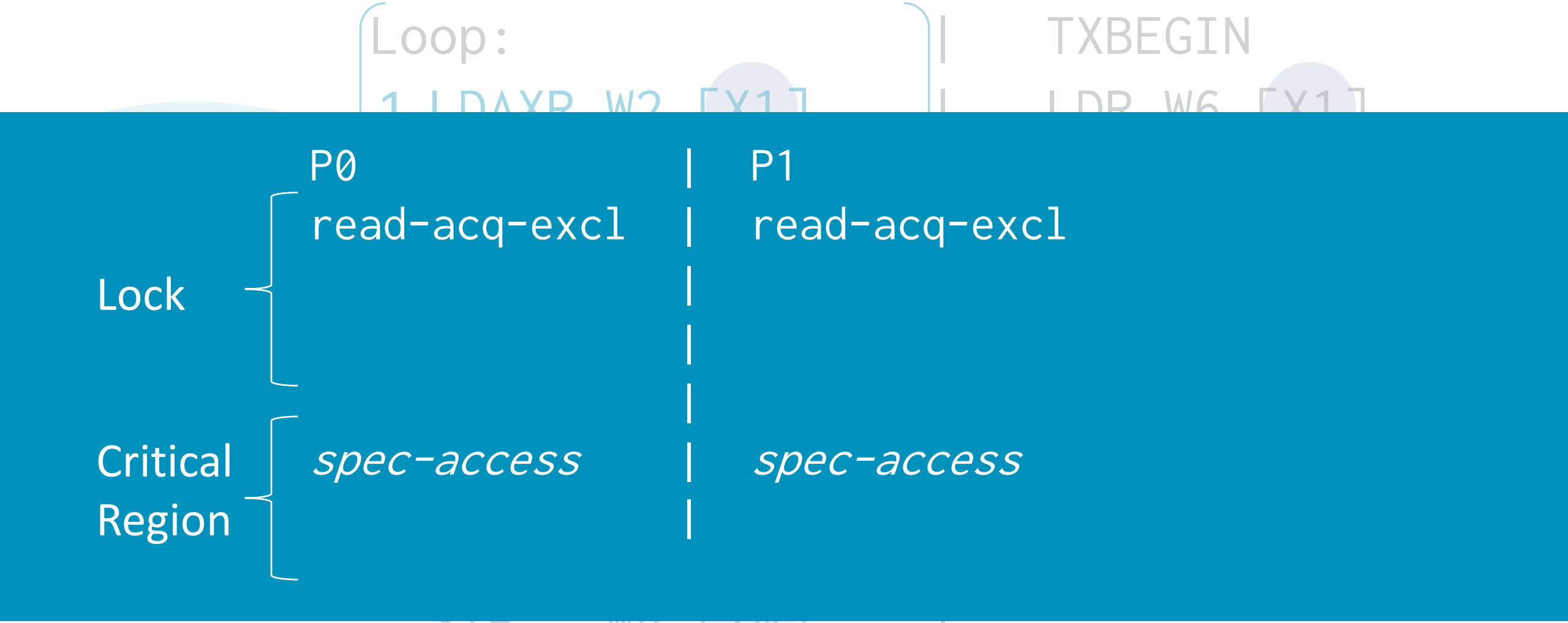
lock = 0

W2 = 0

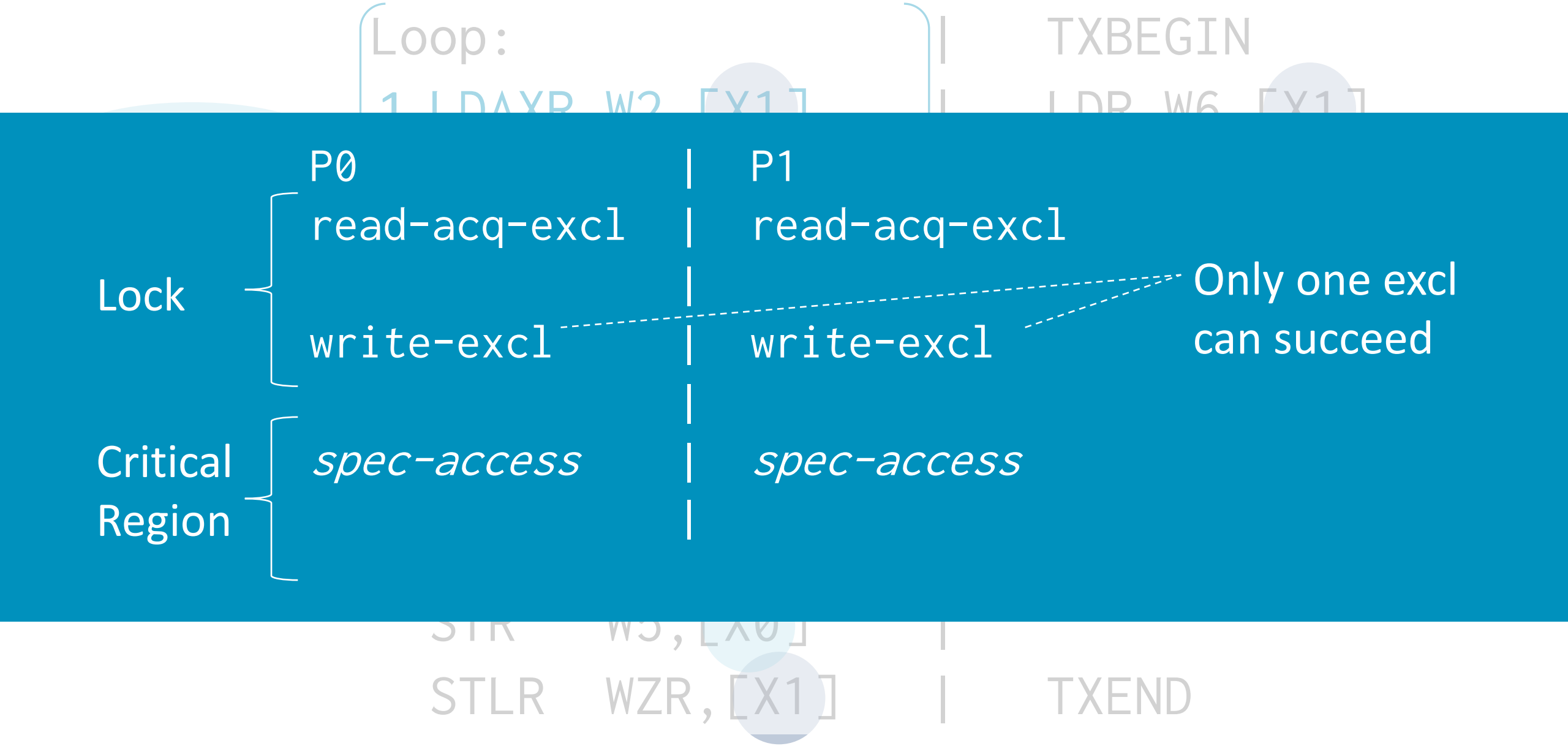
W5 = 0

```
Loop:                          |    TXBEGIN
1 LDAXR W2,[X1]                |    LDR W6,[X1]
  CBNZ  W2,Loop                |    CBZ W6,Crit
  MOV   W3,#1                  |    TXABORT
  STXR  W4,W3,[X1]             |
  CBNZ  W4,Loop                | Crit:
2 LDR   W5,[X0]                |    MOV W7,#1
  ADD   W5,W5,#2               |    STR W7,[X0]
  STR   W5,[X0]                |
  STLR  WZR,[X1]               |    TXEND
```


arm Research



arm Research



arm Research

It is “the [Arm] architecture’s intention to allow store exclusives to promise success/failure very early”

Armv8 flat operational model [Pulte et al., POPL 2018]

arm Research

v = 0

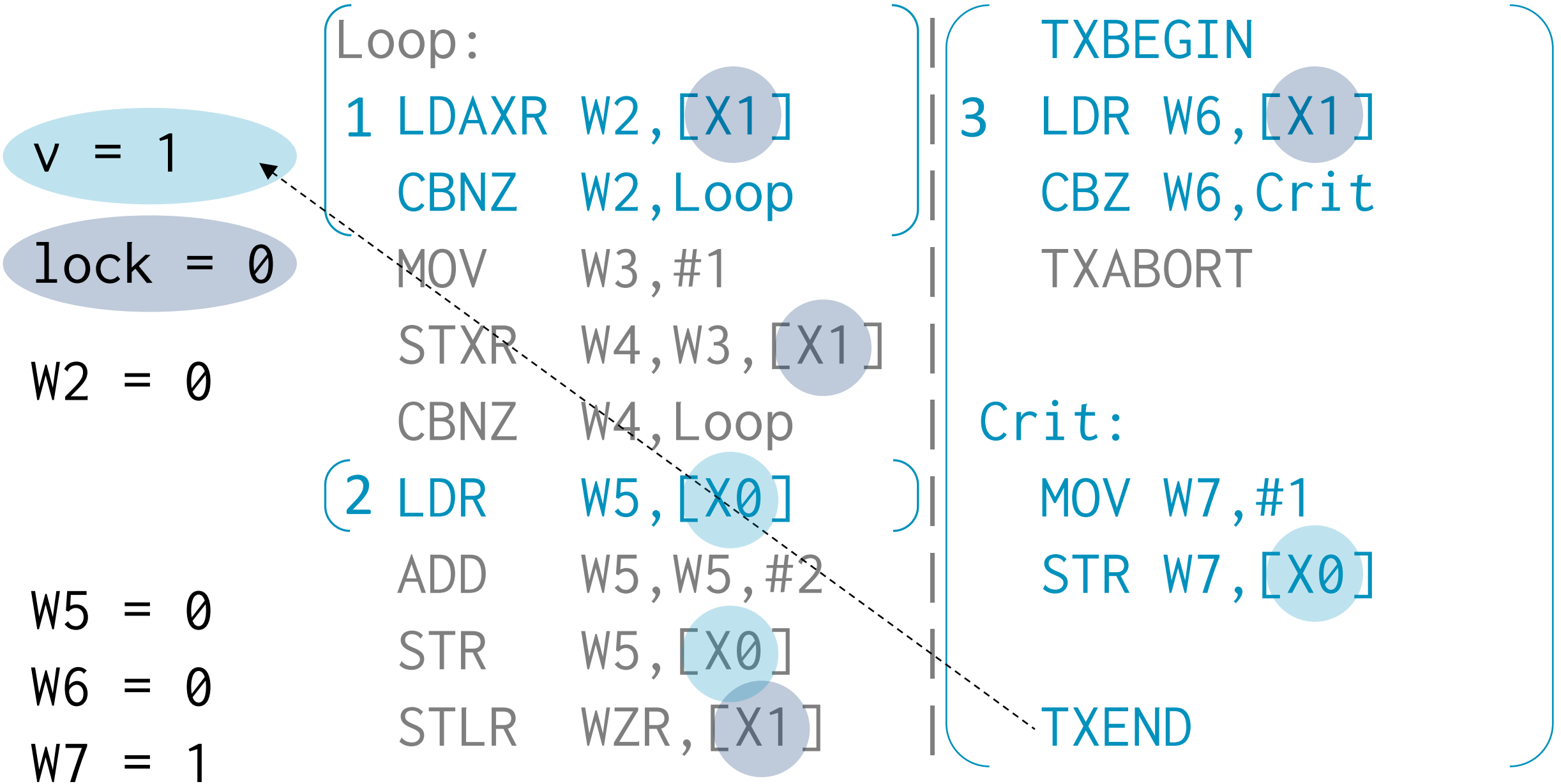
lock = 0

W2 = 0

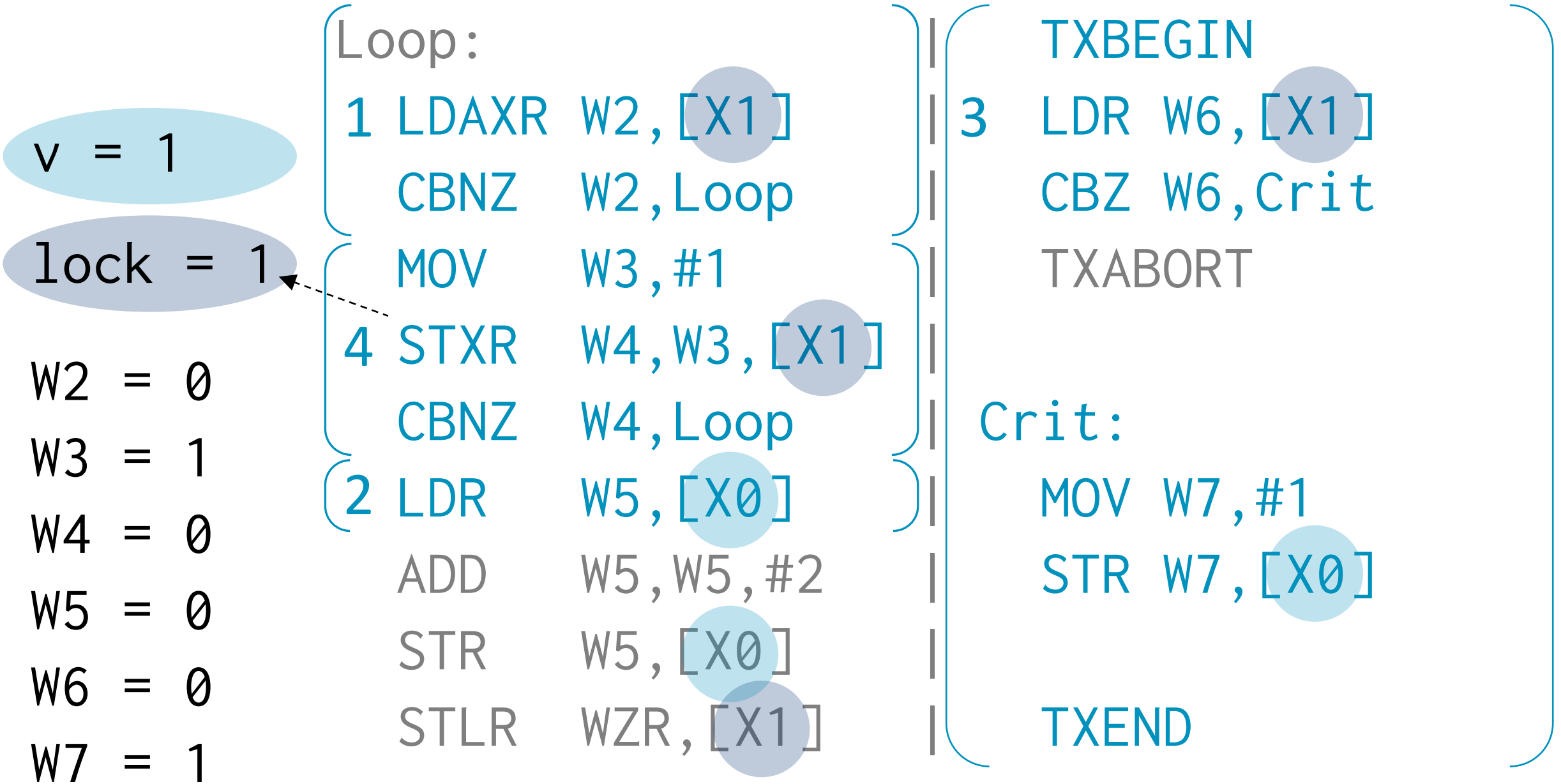
W5 = 0

```
Loop:                          |    TXBEGIN
1 LDAXR W2,[X1]                |    LDR W6,[X1]
  CBNZ  W2,Loop                |    CBZ W6,Crit
  MOV   W3,#1                  |    TXABORT
  STXR  W4,W3,[X1]             |
  CBNZ  W4,Loop                | Crit:
2 LDR   W5,[X0]                |    MOV W7,#1
  ADD   W5,W5,#2               |    STR W7,[X0]
  STR   W5,[X0]                |
  STLR  WZR,[X1]               |    TXEND
```

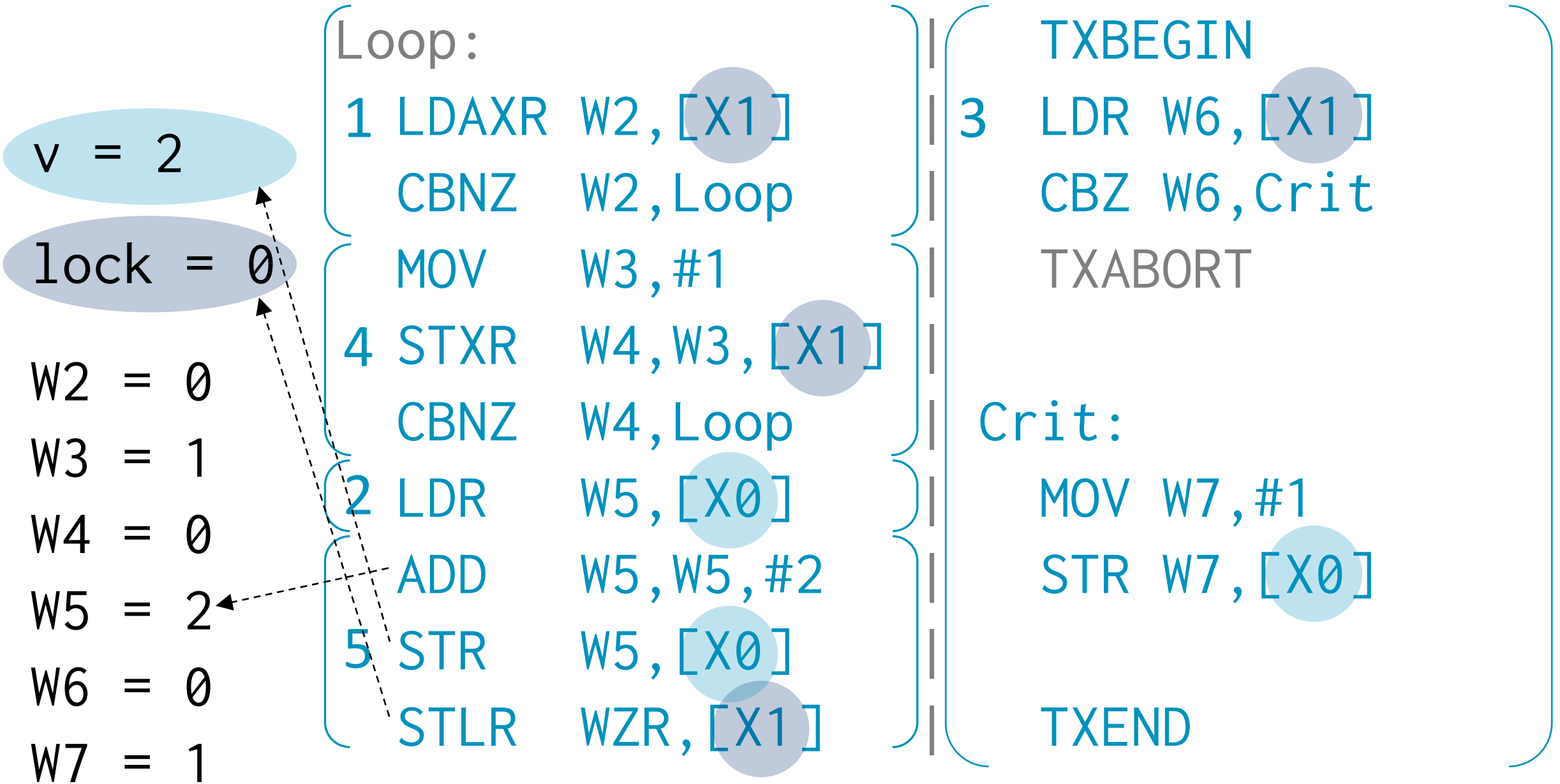

arm Research



arm Research



arm Research



arm Research

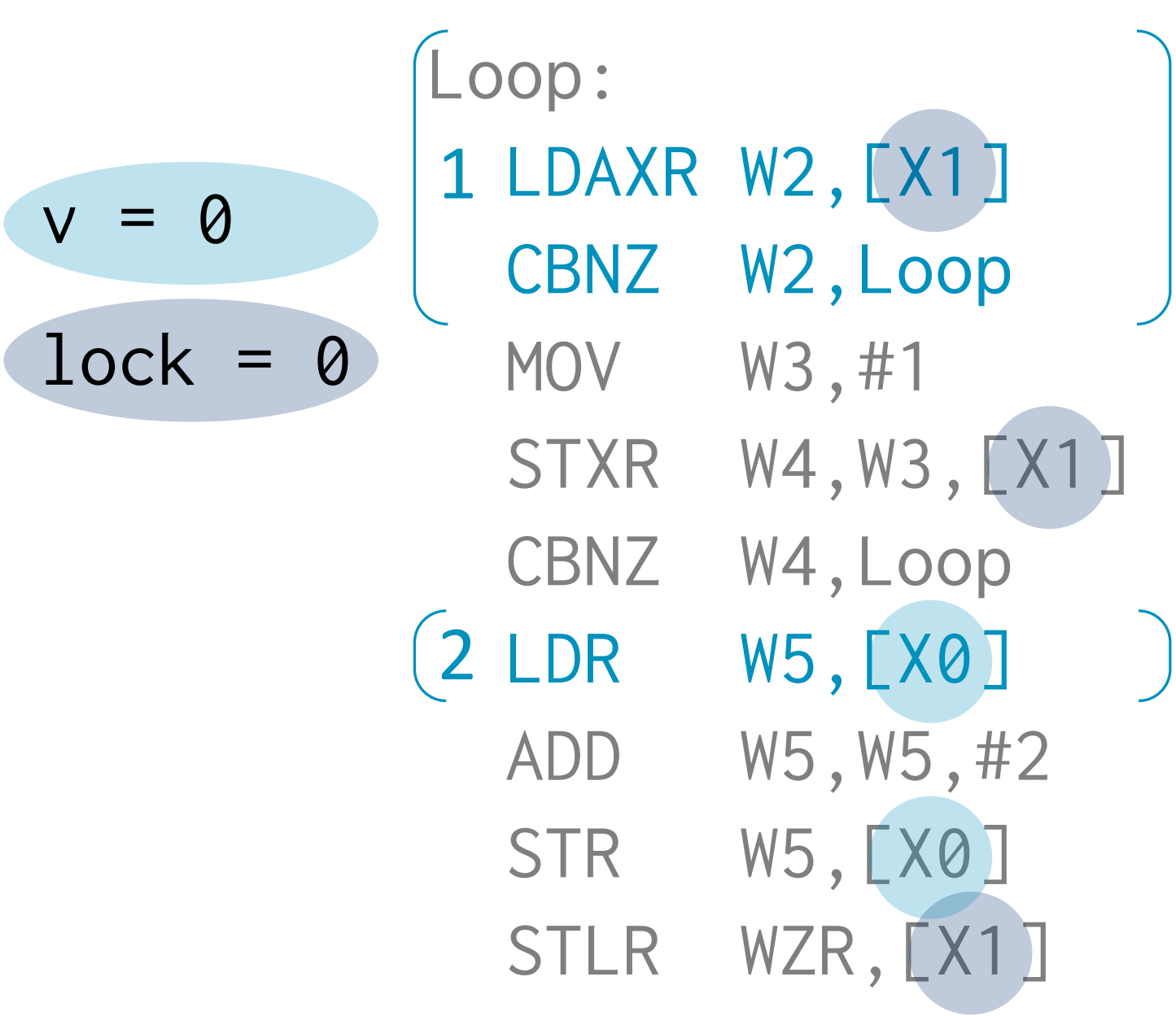
This Armv8 acquire-exclusive spinlock is safe, individually

Elided locks using only transactions are safe, individually

The combination is unsound: the characteristic that makes this spinlock safe (the lock variable must be written-to) is exactly the feature that lock elision takes advantage of

“[the] correctness [of lock elision] is guaranteed without any dependence on memory ordering”

[Rajwar and Goodman, MICRO 2001]



Vital assumption: lock acquisition *happens-before* any access of the critical region

arm Research

A seven(teen) year-old counterexample

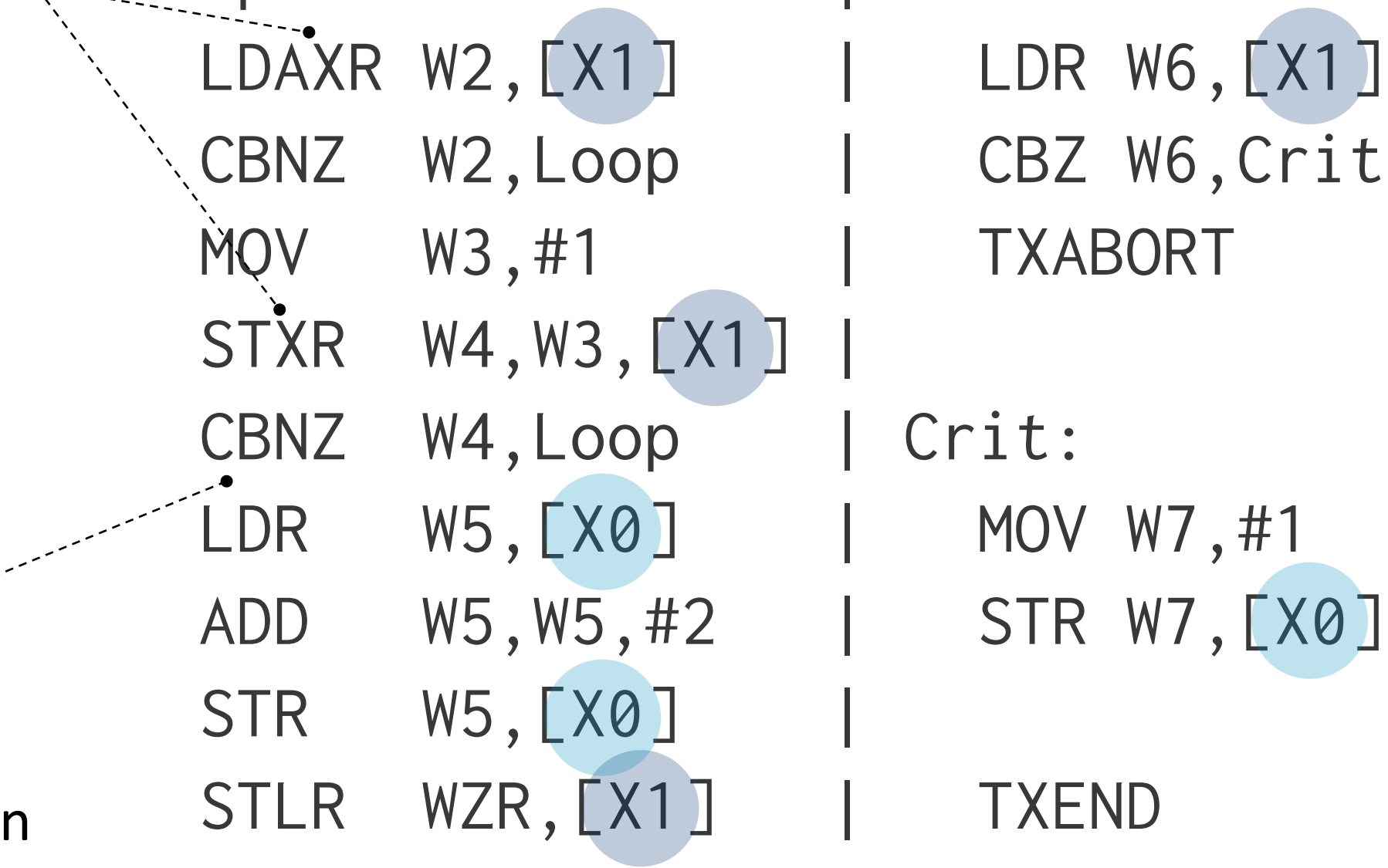
2001: Rajwar and Goodman introduce lock elision

2011: Acquire-release introduced to Armv8

2018: Lock elision counterexample

Replace excls with v8.1 AL (acq-rel) atomic

Insert DMB between lock() and critical region



arm Research

Key ideas and related work

Axiomatic framework and tools [Alglave et al., TOPLAS 2014]

Memalloy tool for automatically comparing memory models [Wickerson et al., POPL 2017]

Litmus test minimality [Lustig et al., ASPLOS 2017]

→ **Automated tool support for empirical testing and bounded verification**

"Transactions in Relaxed Memory Architectures", [Dongol et al., POPL 2018]

https://bit.ly/2xJvbcT

Our paper

TM extensions of x86, Power, Armv8, and C++ axiomatic memory models

Formal models backed by automated tooling for

- Synthesis of minimal tests for empirical testing
- Bounded verification of TM-related transformations and properties

Resulting in the discovery of

- Unsoundness of lock elision wrt an Armv8 spinlock impl.
- Ambiguity in Power TM specification
- Proposed simplification to C++ TM specification
- … (*more in paper*)

References

Alglave et al., “Herding Cats: Modelling, Simulation, Testing, and Data-mining for Weak Memory”, TOPLAS 2014

Arm, “ARMv8 Architecture Reference Manual”

Cain et al., “Robust Architectural Support for Transactional Memory in the Power Architecture”, ISCA 2013

Dongol et al., “Transactions in Relaxed Memory Architectures”, POPL 2018

Gharachorloo et al., “Memory consistency and event ordering in scalable shared-memory multiprocessors”, ISCA 1990

Lustig et al., “Automated Synthesis of Comprehensive Memory Model Litmus Test Suites”, ASPLOS 2017

Pulte et al., “Simplifying ARM Concurrency: Multicopy-atomic Axiomatic and Operational Models for ARMv8”, POPL 2018

Rajwar and Goodman, “Speculative Lock Elision: Enabling Highly Concurrent Multithreaded Execution”, MICRO 2001

Sarkar et al., “Synchronising C/C++ and POWER”, PLDI 2012

Wickerson et al., “Automatically Comparing Memory Consistency Models”, POPL 2017

Acknowledgements

We are grateful to Stephan Diestelhorst, Matt Horsnell, and Grigorios Magklis for extensive discussions of TM in general, and how it might interact with the Armv8 architecture memory model, to Nizamudheen Ahmed and Vishwanath HV for RTL testing, and to Peter Sewell for letting us access his Power machine.

We thank the following people for their insightful comments on various drafts of this work: Mark Batty, Andrea Cerone, George Constantinides, Stephen Dolan, Alastair Donaldson, Brijesh Dongol, Hugues Evrard, Shaked Flur, Graham Hazel, Radha Jagadeesan, Jan Kończak, Dominic Mulligan, Christopher Pulte, Alastair Reid, James Riely, the anonymous PLDI reviewers, and our shepherd, Julian Dolby.

This work was supported by an Imperial College Research Fellowship and the EPSRC (EP/K034448/1).

arm Research