THE SEMANTICS OF TRANSACTIONS AND WEAK MEMORY IN X86, POWER, ARM, AND C++

Nathan Chong
Arm Ltd.

Tyler Sorensen
Imperial

John Wickerson
Imperial

UCL PPLV Seminar, Thursday 10 May 2018

OUTLINE

OUTLINE

- Weak memory

OUTLINE

- Weak memory
- Transactions

OUTLINE

- Weak memory
- Transactions
- Weak memory and transactions

OUTLINE

- Weak memory
- Transactions
- Weak memory and transactions
- Validating our models

OUTLINE

- Weak memory
- Transactions
- Weak memory and transactions
- Validating our models
- The problem with lock elision

OUTLINE

- Weak memory
- Transactions
- Weak memory and transactions
- Validating our models
- The problem with lock elision
- Related and future work

WEAK MEMORY

WEAK MEMORY

```
MOV [x] 1     ||  MOV [y] 1

MOV r0 [y]    ||  MOV r1 [x]
```

WEAK MEMORY

```
MOV [x] 1   ║ MOV [y] 1

MOV r0 [y]  ║ MOV r1 [x]
```

```
r0=1
r1=1
```

WEAK MEMORY

```
MOV [x] 1      ||  MOV [y] 1

MOV r0 [y]     ||  MOV r1 [x]
```

```
r0=1        r0=0
r1=1        r1=1
```

WEAK MEMORY

```
MOV [x] 1      ||  MOV [y] 1
MOV r0 [y]     ||  MOV r1 [x]
```

```
r0=1        r0=0        r0=1
r1=1        r1=1        r1=0
```

WEAK MEMORY

```
MOV [x] 1   || MOV [y] 1
MOV r0 [y]  || MOV r1 [x]
```

$$\underbrace{\begin{array}{ccc} \texttt{r0=1} & \texttt{r0=0} & \texttt{r0=1} \\ \texttt{r1=1} & \texttt{r1=1} & \texttt{r1=0} \end{array}}_{\text{SC}}$$

WEAK MEMORY

```
MOV [x] 1    ||  MOV [y] 1
MOV r0 [y]   ||  MOV r1 [x]
```

$$\underbrace{\begin{array}{ccc} r0=1 & r0=0 & r0=1 \\ r1=1 & r1=1 & r1=0 \end{array}}_{\text{SC}} \quad \begin{array}{c} r0=0 \\ r1=0 \end{array}$$

WEAK MEMORY

```
MOV [x] 1   ||  MOV [y] 1

MOV r0 [y]  ||  MOV r1 [x]
```

$$\underbrace{\underbrace{\begin{matrix}\texttt{r0=1}\\\texttt{r1=1}\end{matrix}\qquad\begin{matrix}\texttt{r0=0}\\\texttt{r1=1}\end{matrix}\qquad\begin{matrix}\texttt{r0=1}\\\texttt{r1=0}\end{matrix}}_{\text{SC}}\qquad\begin{matrix}\texttt{r0=0}\\\texttt{r1=0}\end{matrix}}_{\text{x86}}$$

WEAK MEMORY

```
MOV [x] 1   || MOV [y] 1

MOV r0 [y]  || MOV r1 [x]
```

$$\underbrace{\underbrace{\begin{array}{l}r0=1\\r1=1\end{array}\qquad\begin{array}{l}r0=0\\r1=1\end{array}\qquad\begin{array}{l}r0=1\\r1=0\end{array}}_{\text{SC}}\qquad\begin{array}{l}r0=0\\r1=0\end{array}}_{\text{x86}}$$

WEAK MEMORY

```
MOV [x] 1     ||  MOV [y] 1

MOV r0 [y]    ||  MOV r1 [x]
```

r0=1	r0=0	r0=1	r0=0
r1=1	r1=1	r1=0	r1=0

x86

K MEMORY

```
MOV [x] 1    ||  MOV [y] 1
MOV r0 [y]   ||  MOV r1 [x]
```

```
r0=1    r0=0    r0=1    r0=0
r1=1    r1=1    r1=0    r1=0
```

x86

AK MEMO

```
MOV [x] 1    ||  MOV [y] 1
MOV r0 [y]   ||  MOV r1 [x]
```

```
r0=1     r0=0     r0=1     r0=0
r1=1     r1=1     r1=0     r1=0
```

x86

```
MOV [x] 1     ||   OV [y] 1

MOV r0 [y]    ||  MOV r1 [x]
```

```
r0=1      r0=0      r0=1      r0=0
r1=1      r1=1      r1=0      r1=0
```

x86

```
MOV [x] 1
                ||  MOV
MOV r0 [y]
```

```
r0=1     r0=0     r0=
r1=1     r1=1     r1=0     r1=0
```

x86

ARM®

AMD

NVIDIA®

```
MOV [x] 1      ║ MOV
MOV r0 [y]     ║ MOV
```

THE C++ PROGRAMMING LANGUAGE

```
r0=1      r0=0      r0=
r1=1      r1=1                r1=0
```

IBM

OpenCL

intel®

ARM®

AMD

NVIDIA®

MOV [x] 1

[y]

MOV

Java™

THE C++ PROGRAMMING LANGUAGE

=0

r0=

r1=1

=1

r1=0

IBM

OpenCL

intel®

WEAK MEMORY IS HARD!

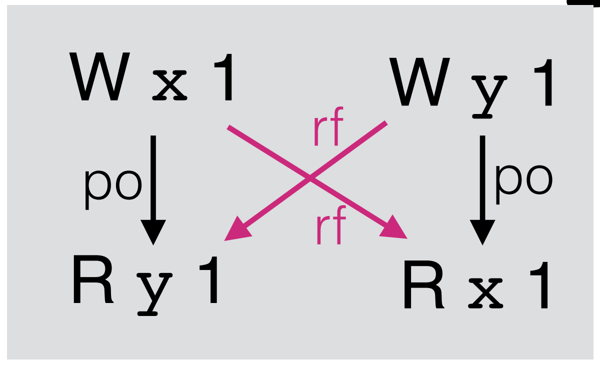
- x86 proved tricky to formalise correctly *[Sarkar et al., POPL'09; Owens et al., TPHOLs'09]*
- Bug found in deployed "Power 5" processors *[Alglave et al., CAV'10]*
- C++ specification did not guarantee its own key property *[Batty et al., POPL'11]*
- Routine compiler optimisations are invalid under Java and C++ memory models *[Sevcik, PLDI'11; Vafeiadis et al. POPL'15]*
- Behaviour of NVIDIA graphics processors contradicted NVIDIA's programming guide *[Alglave et al., ASPLOS'15]*

MODELLING WEAK MEMORY

`MOV [x] 1`	`MOV [y] 1`
`MOV r0 [y]`	`MOV r1 [x]`

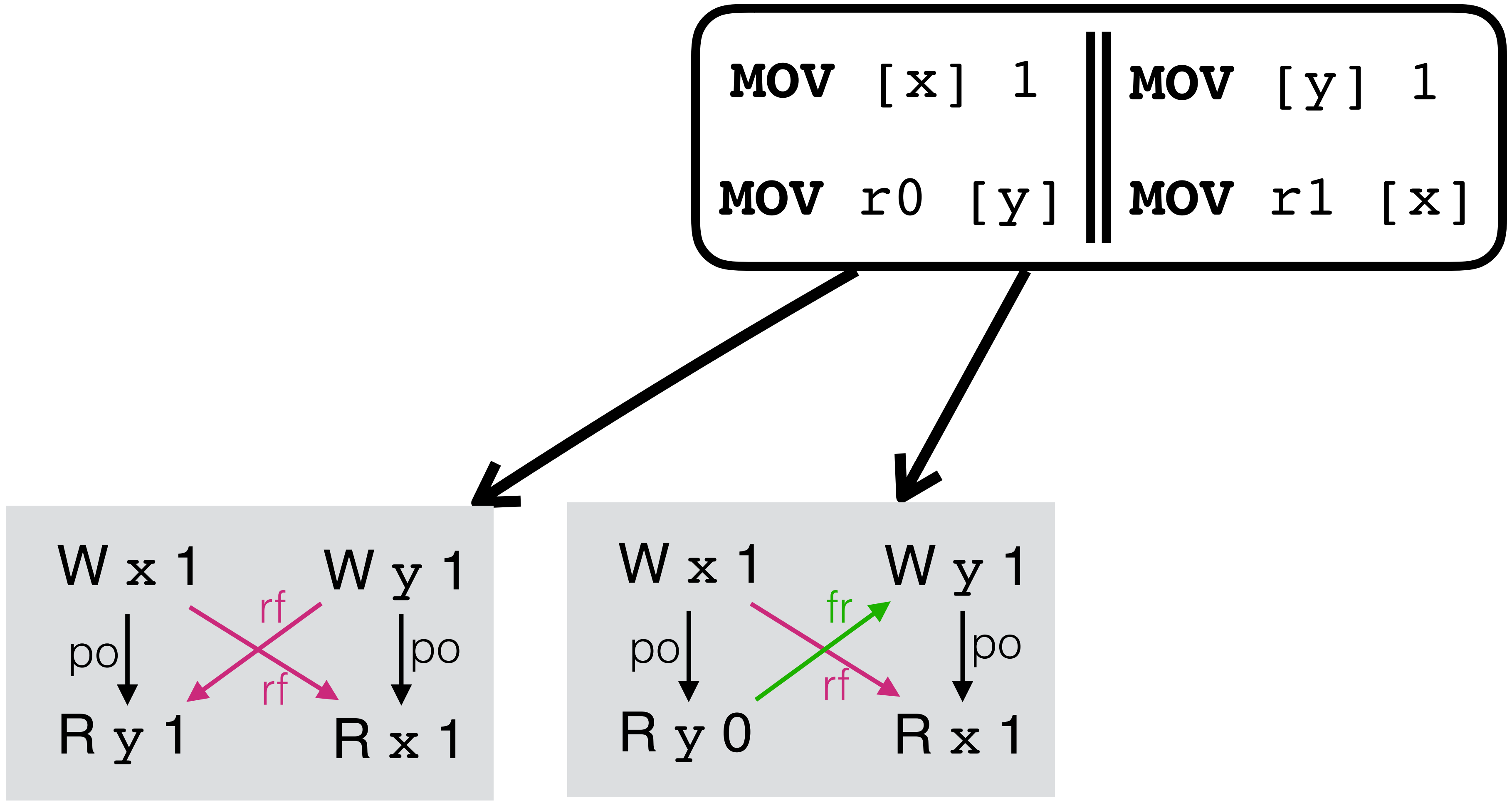
MODELLING WEAK MEMORY

```
MOV [x] 1    ||  MOV [y] 1
MOV r0 [y]   ||  MOV r1 [x]
```



r0=1 r1=1

MODELLING WEAK MEMORY

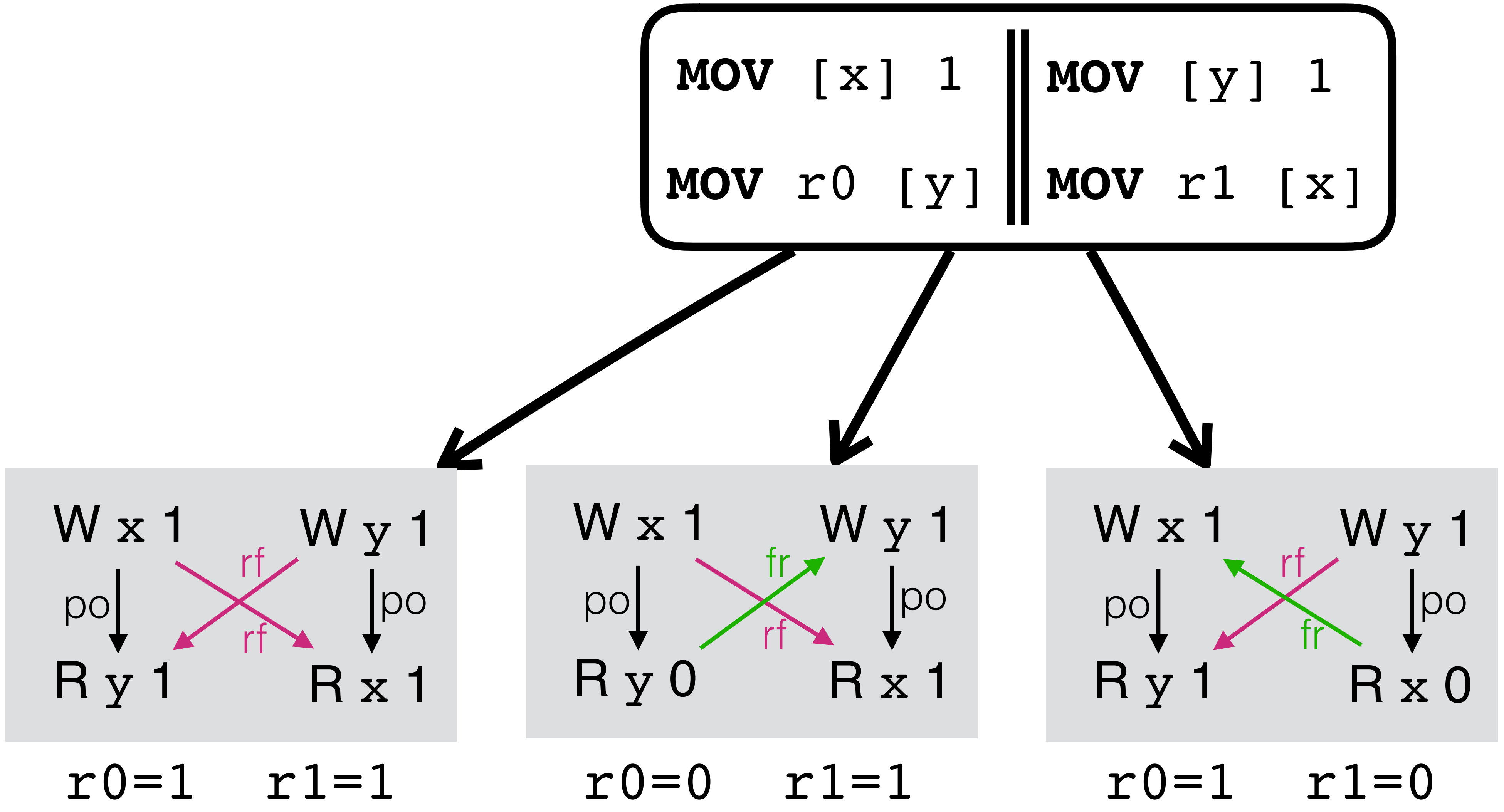


```
MOV [x] 1    ||  MOV [y] 1
MOV r0 [y]   ||  MOV r1 [x]
```

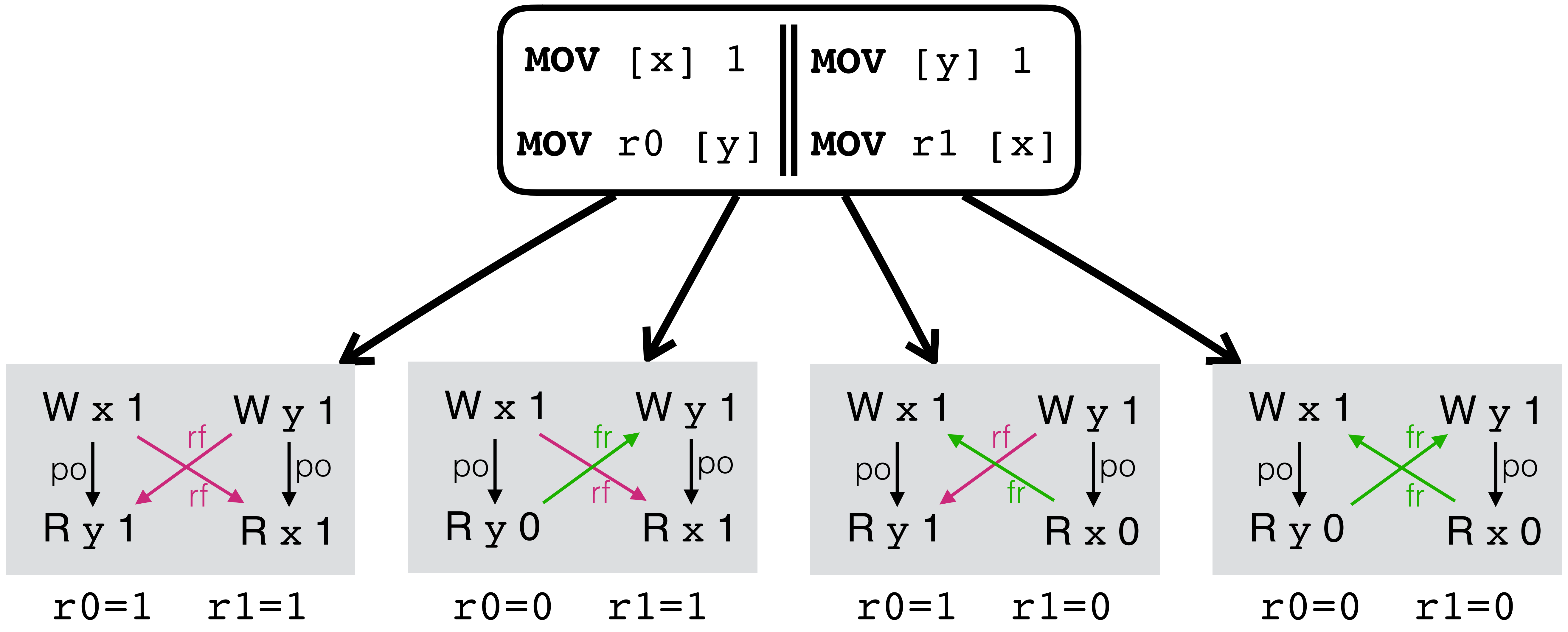
r0=1 r1=1

r0=0 r1=1

MODELLING WEAK MEMORY

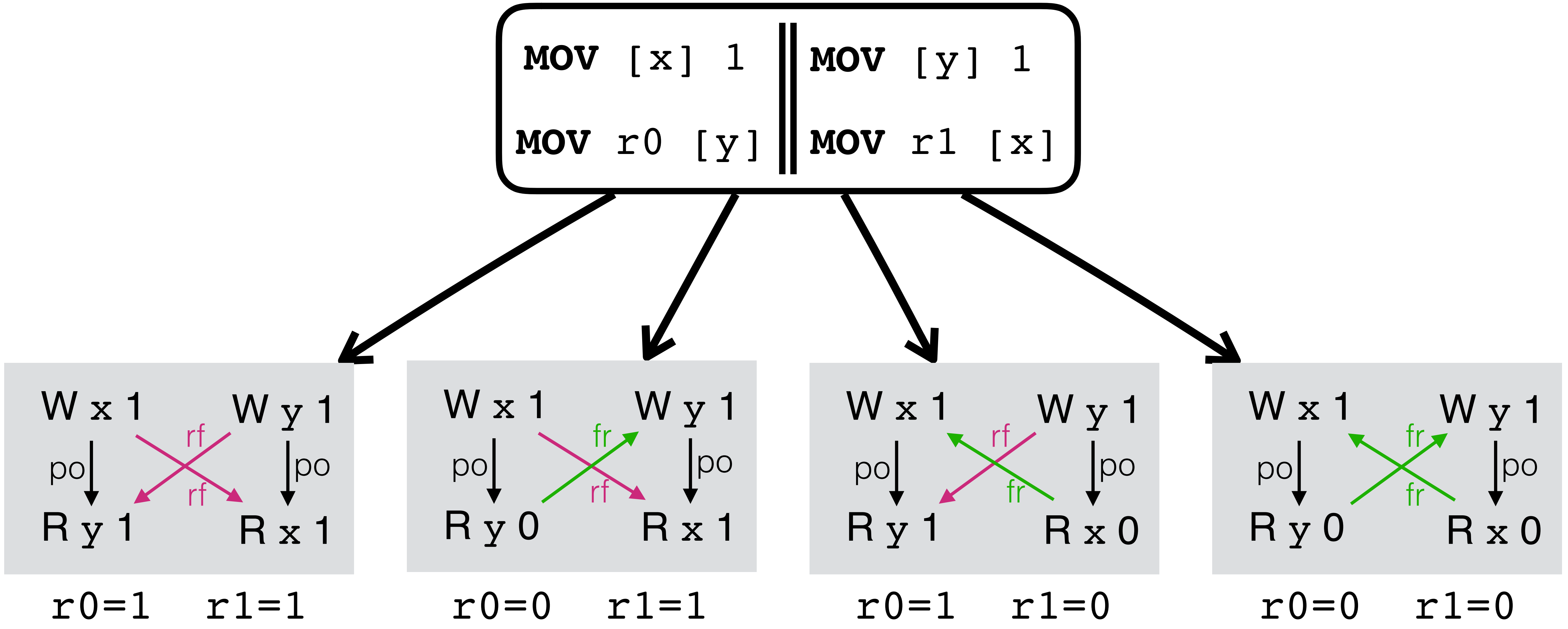


MODELLING WEAK MEMORY



```
MOV [x] 1    || MOV [y] 1
MOV r0 [y]   || MOV r1 [x]
```

MODELLING WEAK MEMORY

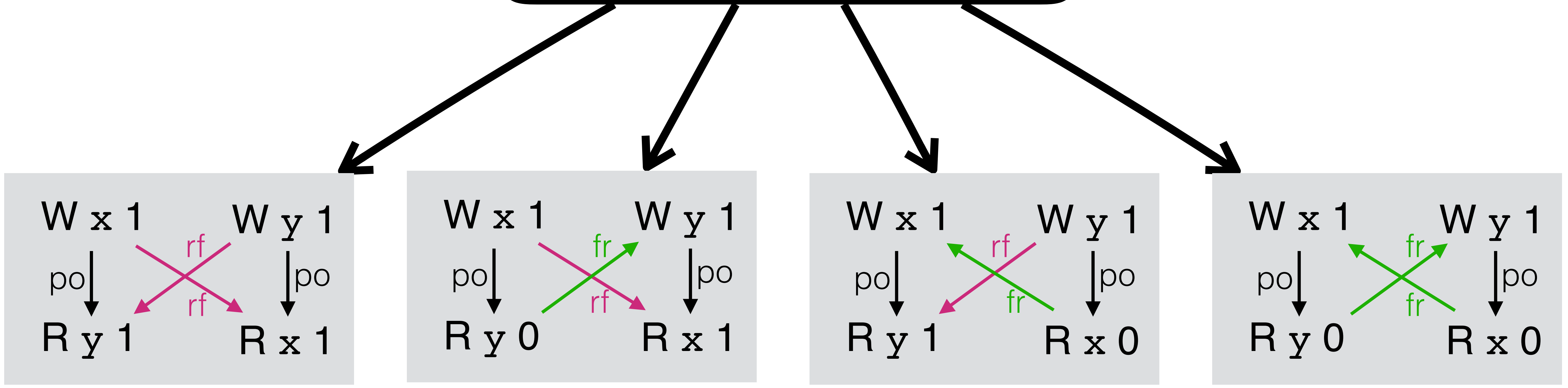


```
MOV [x] 1   || MOV [y] 1
MOV r0 [y]  || MOV r1 [x]
```

SC: ✓

MODELLING WEAK MEMORY

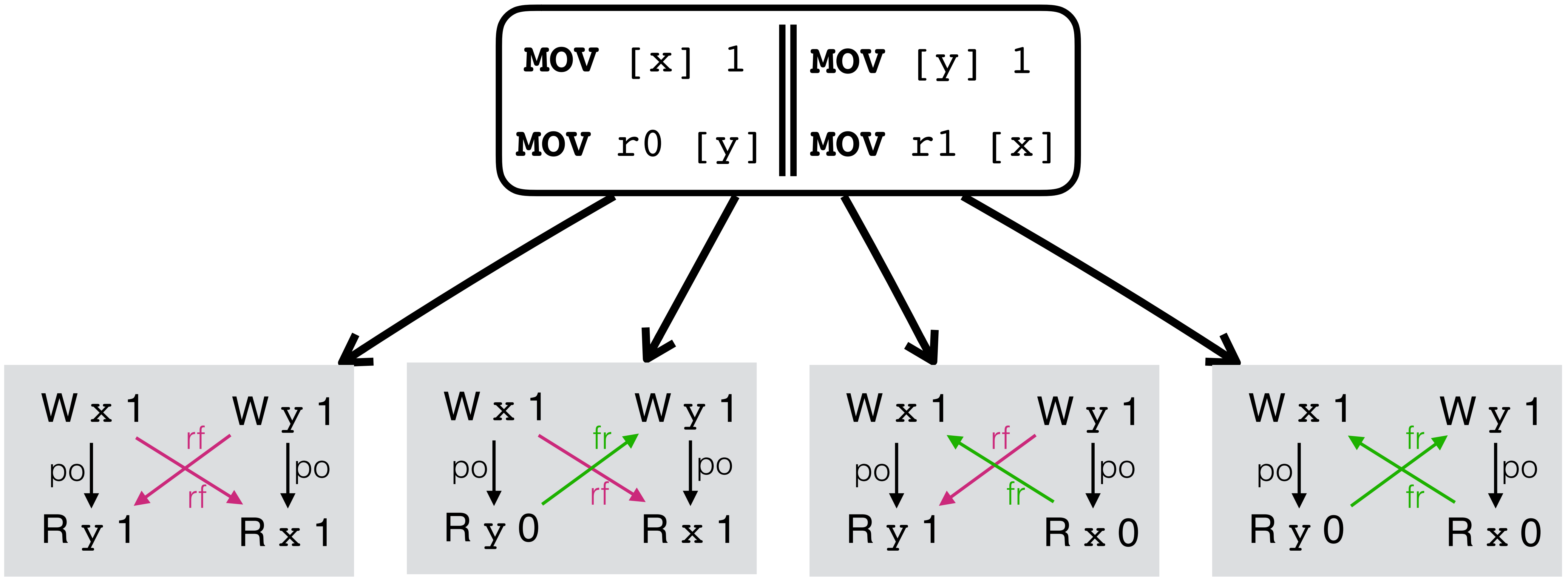
```
MOV [x] 1    ||  MOV [y] 1
MOV r0 [y]   ||  MOV r1 [x]
```



r0=1 r1=1 | r0=0 r1=1 | r0=1 r1=0 | r0=0 r1=0

SC: ✓ SC: ✓

MODELLING WEAK MEMORY



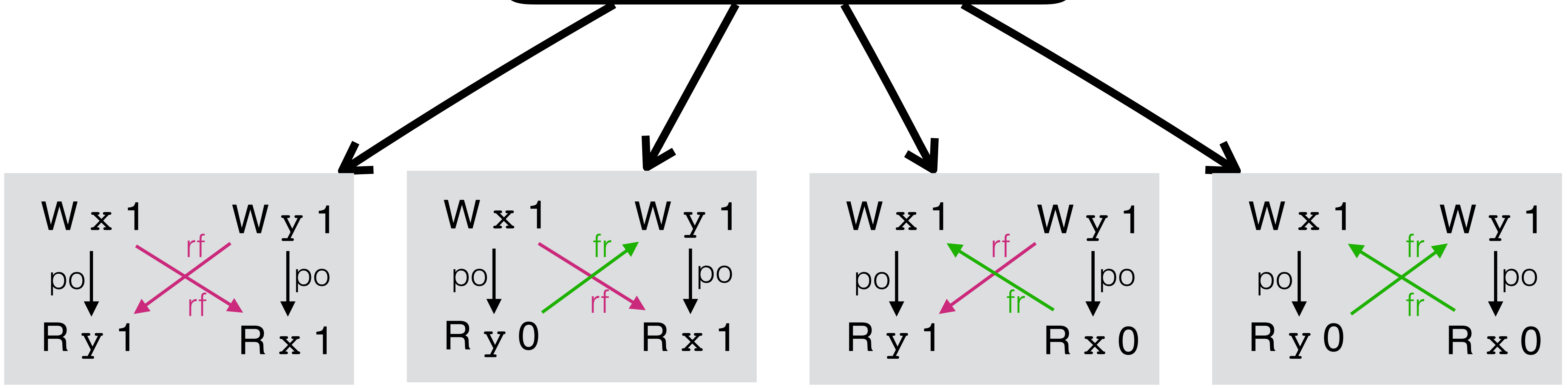
```
MOV [x] 1      ||  MOV [y] 1
MOV r0  [y]    ||  MOV r1  [x]
```

r0=1 r1=1 | r0=0 r1=1 | r0=1 r1=0 | r0=0 r1=0

SC: ✓ | SC: ✓ | SC: ✓

MODELLING WEAK MEMORY

```
MOV [x] 1    ||  MOV [y] 1
MOV r0 [y]   ||  MOV r1 [x]
```



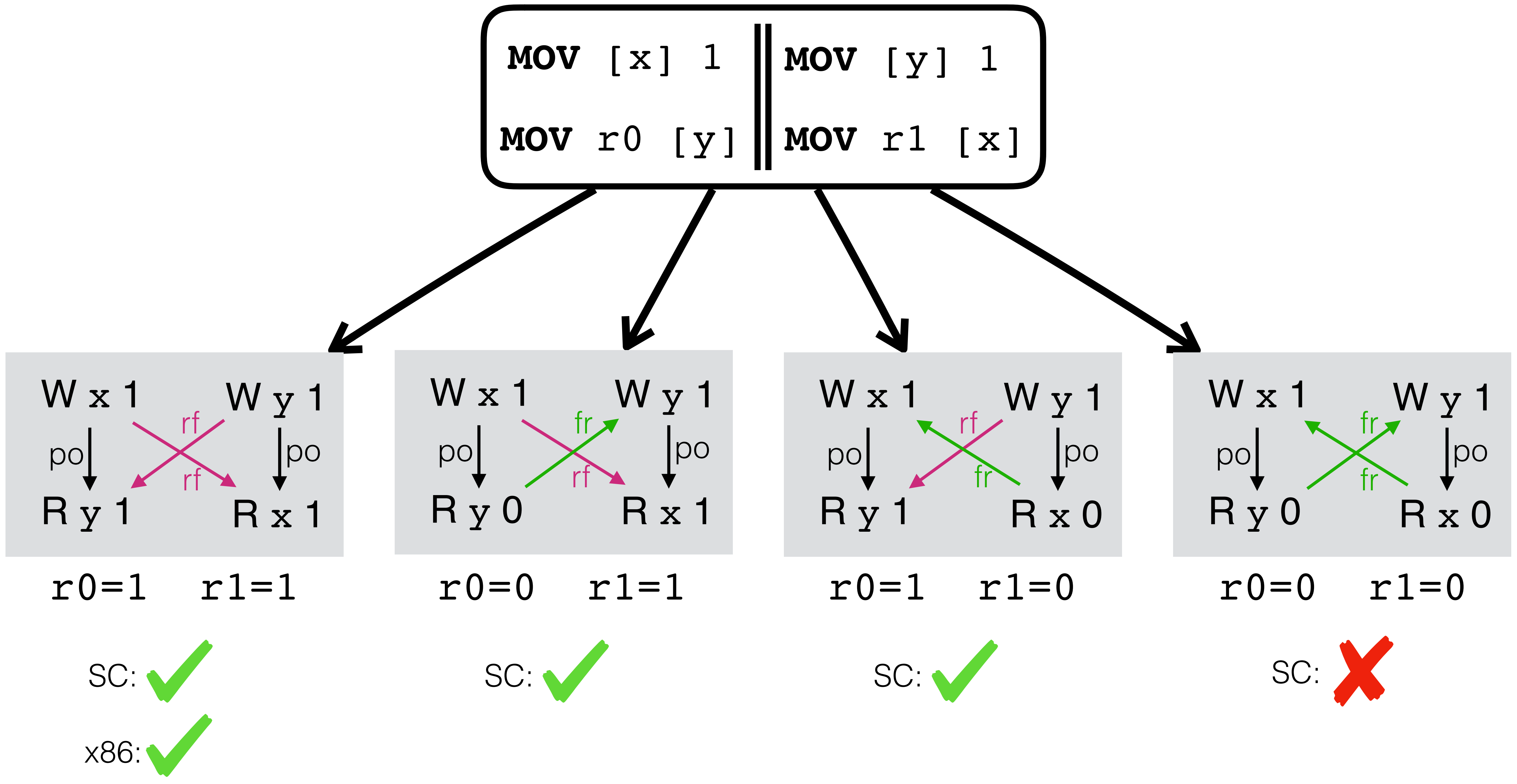
r0=1 r1=1 SC: ✓

r0=0 r1=1 SC: ✓

r0=1 r1=0 SC: ✓

r0=0 r1=0 SC: ✗

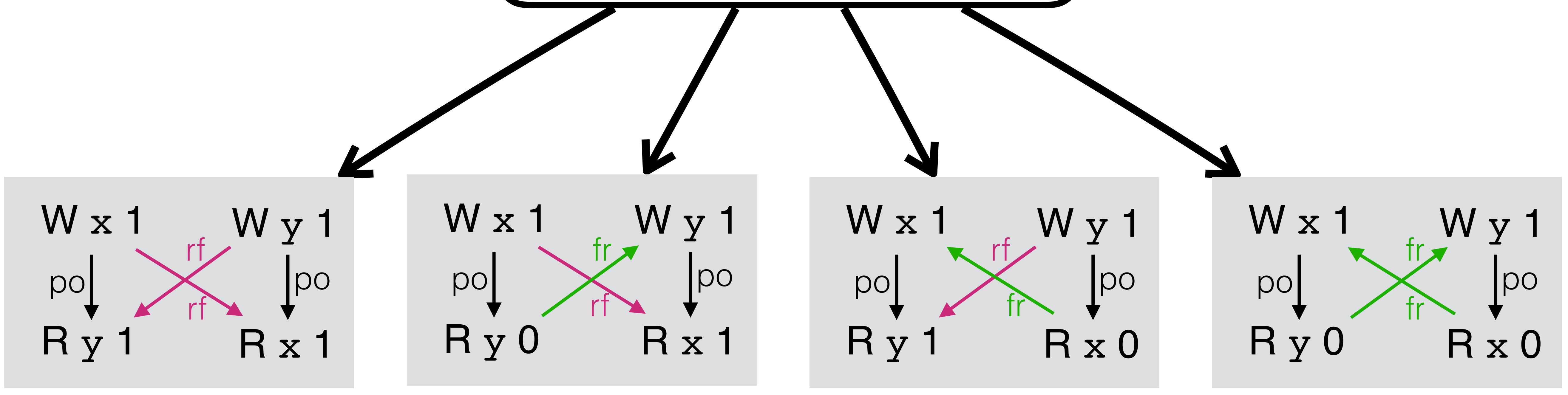
MODELLING WEAK MEMORY



```
MOV [x] 1    || MOV [y] 1
MOV r0 [y]   || MOV r1 [x]
```

MODELLING WEAK MEMORY

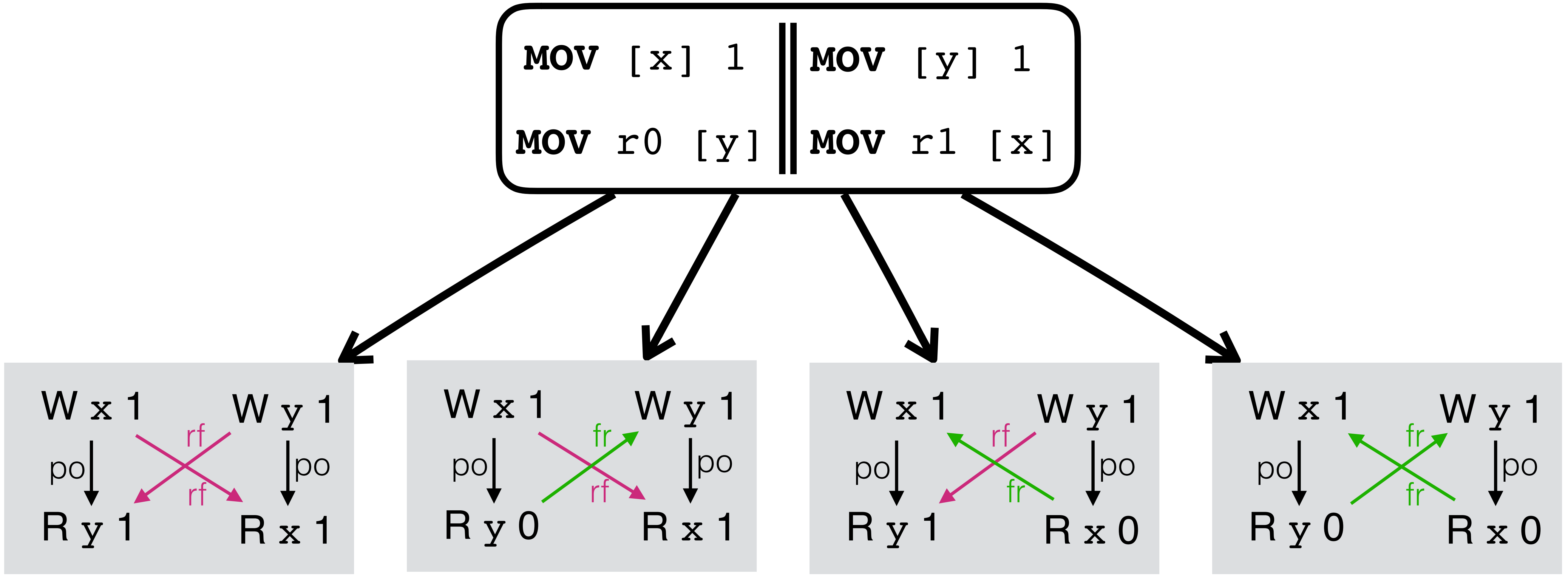
```
MOV [x] 1    || MOV [y] 1
MOV r0 [y]   || MOV r1 [x]
```



r0=1 r1=1	r0=0 r1=1	r0=1 r1=0	r0=0 r1=0
SC: ✓	SC: ✓	SC: ✓	SC: ✗
x86: ✓	x86: ✓		

MODELLING WEAK MEMORY

```
MOV [x] 1     ||  MOV [y] 1
MOV r0 [y]    ||  MOV r1 [x]
```



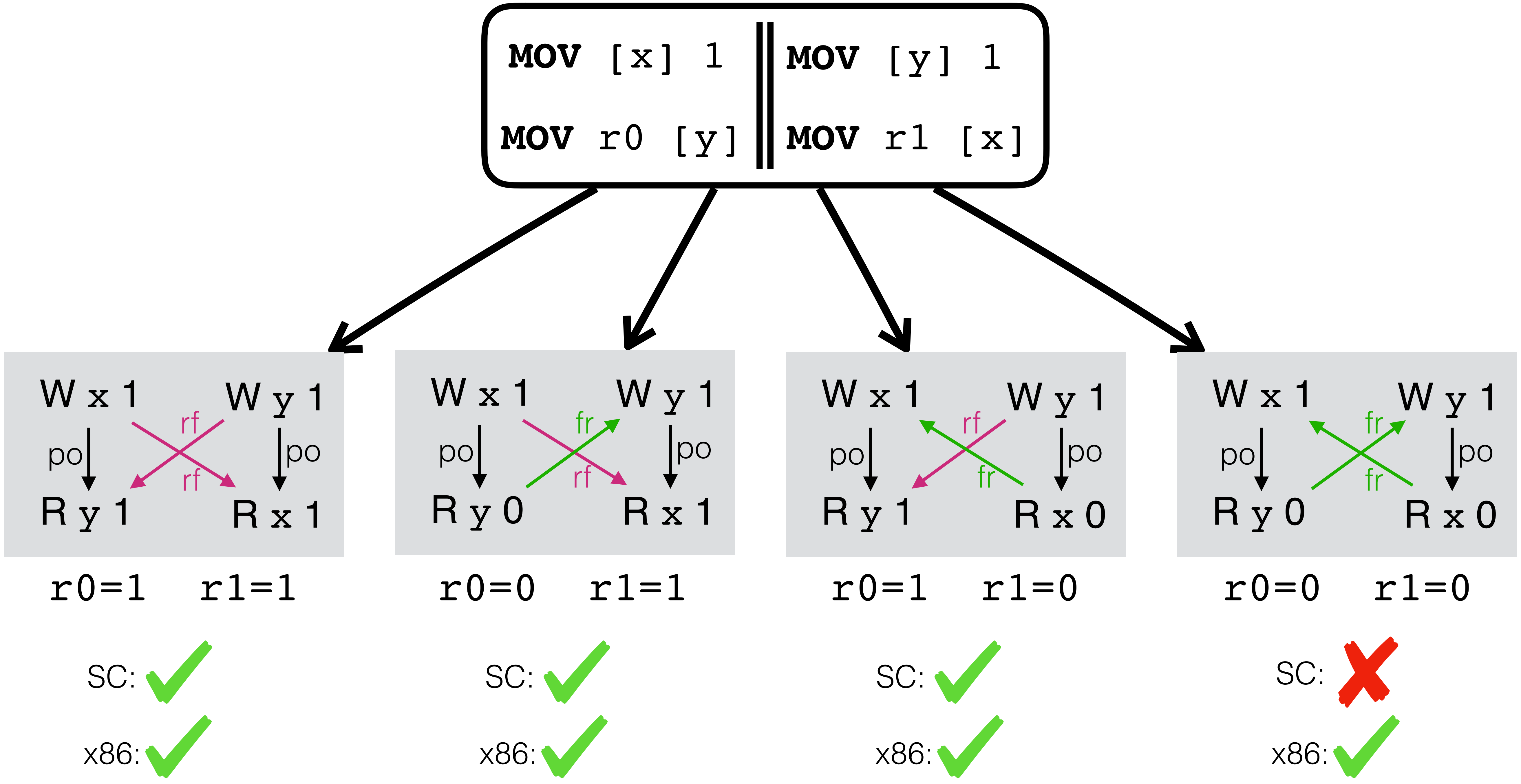
`r0=1  r1=1` — SC: ✓ x86: ✓

`r0=0  r1=1` — SC: ✓ x86: ✓

`r0=1  r1=0` — SC: ✓ x86: ✓

`r0=0  r1=0` — SC: ✗

MODELLING WEAK MEMORY

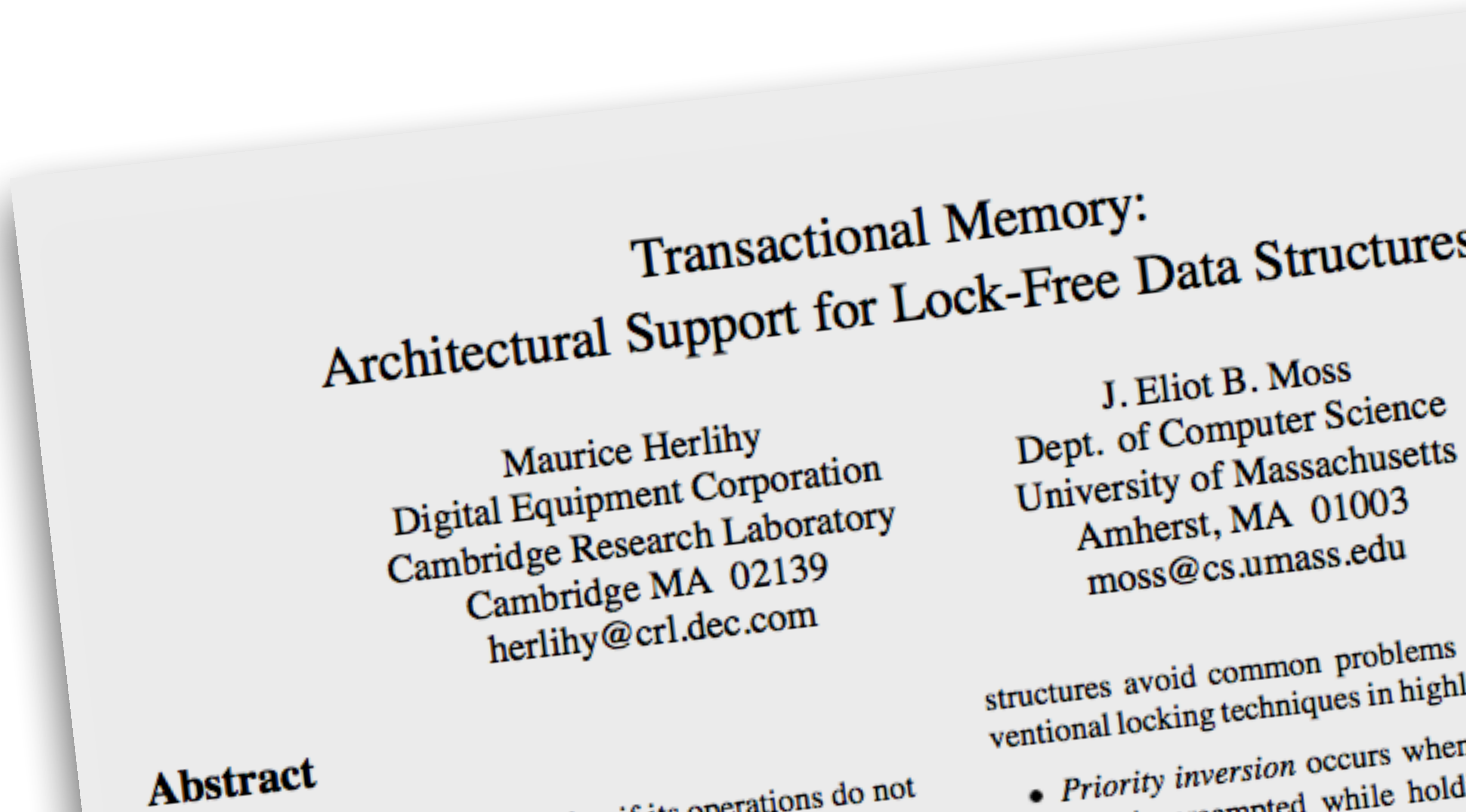


```
MOV [x] 1   || MOV [y] 1
MOV r0 [y]  || MOV r1 [x]
```

OUTLINE

- ~~Weak memory~~
- Transactions
- Weak memory and transactions
- Validating our models
- The problem with lock elision
- Related and future work

TRANSACTIONAL MEMORY



Transactional Memory:
Architectural Support for Lock-Free Data Structures

Maurice Herlihy
Digital Equipment Corporation
Cambridge Research Laboratory
Cambridge MA 02139
herlihy@crl.dec.com

J. Eliot B. Moss
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
moss@cs.umass.edu

Abstract

structures avoid common problems
ventional locking techniques in highl

• *Priority inversion* occurs wher

TRANSACTIONAL MEMORY

- X86:

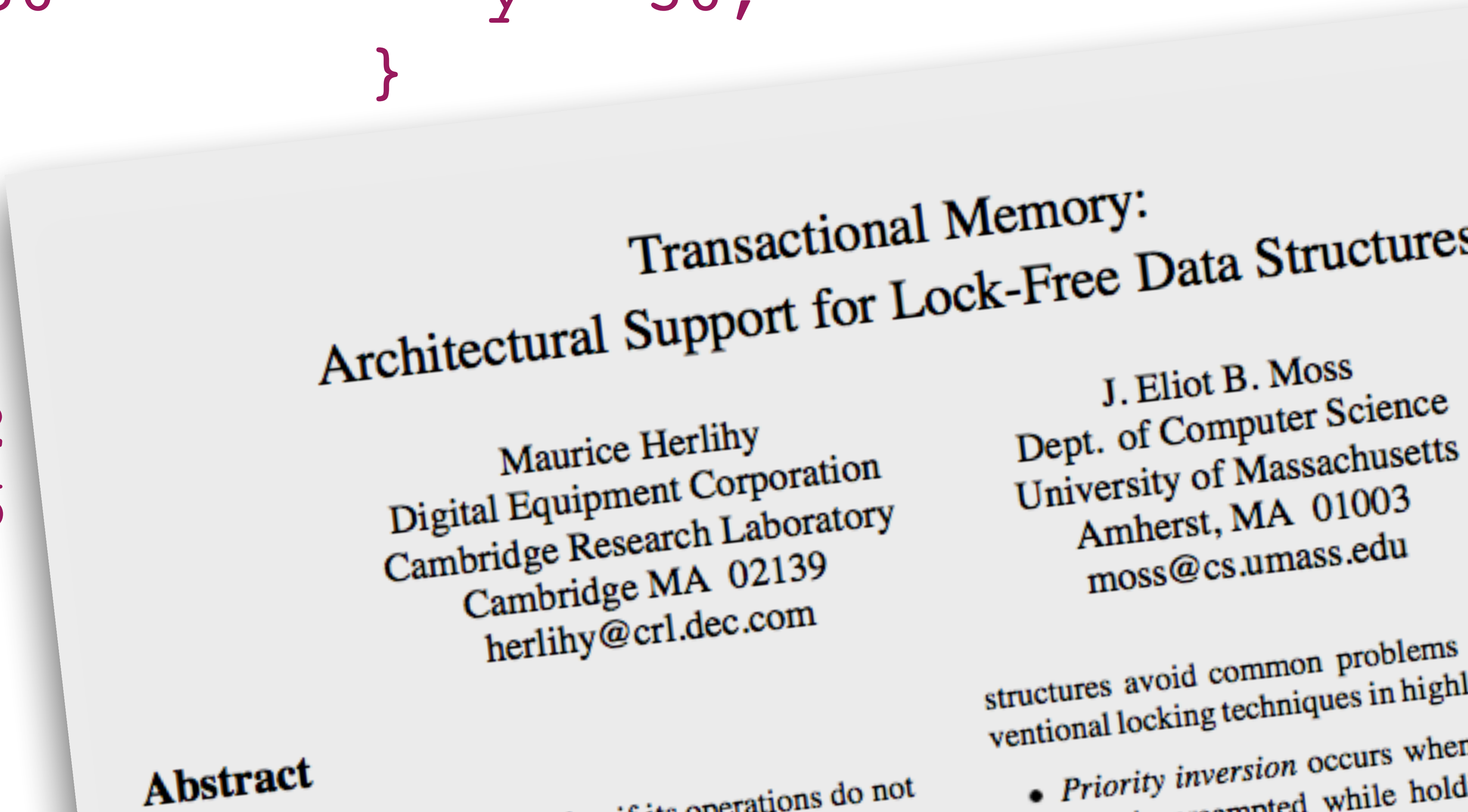
```
XBEGIN
MOV [x] 42
MOV [y] 36
XEND
```

- C++:

```
atomic {
  *x = 42;
  *y = 36;
}
```

- Power:

```
tbegin
stw x #42
stw y #36
tend
```



Transactional Memory:
Architectural Support for Lock-Free Data Structures

Maurice Herlihy
Digital Equipment Corporation
Cambridge Research Laboratory
Cambridge MA 02139
herlihy@crl.dec.com

J. Eliot B. Moss
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
moss@cs.umass.edu

Abstract

if its operations do not

structures avoid common problems
ventional locking techniques in highl

- *Priority inversion* occurs when
mpted while hold

OUTLINE

- ~~Weak memory~~
- ~~Transactions~~
- Weak memory and transactions
- Validating our models
- The problem with lock elision
- Related and future work

WEAK MEMORY + TM = ?

```
MOV [x] 1    ||  MOV [y] 1
MOV r0 [y]   ||  MOV r1 [x]
```

$$\underbrace{\underbrace{\begin{matrix}\texttt{r0=1}\\ \texttt{r1=1}\end{matrix} \qquad \begin{matrix}\texttt{r0=0}\\ \texttt{r1=1}\end{matrix} \qquad \begin{matrix}\texttt{r0=1}\\ \texttt{r1=0}\end{matrix}}_{\text{SC}} \qquad \begin{matrix}\texttt{r0=0}\\ \texttt{r1=0}\end{matrix}}_{\text{x86}}$$

WEAK MEMORY + TM = ?

```
XBEGIN            ||  XBEGIN
MOV  [x] 1        ||  MOV  [y] 1
MOV  r0  [y]      ||  MOV  r1  [x]
XEND              ||  XEND
```

$$\underbrace{\underbrace{\begin{matrix} \texttt{r0=1} & \texttt{r0=0} & \texttt{r0=1} \\ \texttt{r1=1} & \texttt{r1=1} & \texttt{r1=0} \end{matrix}}_{\text{SC}} \quad \begin{matrix} \texttt{r0=0} \\ \texttt{r1=0} \end{matrix}}_{\text{x86}}$$

WEAK MEMORY + TM = ?

```
XBEGIN          || XBEGIN
MOV  [x] 1      || MOV  [y] 1
MOV  r0  [y]    || MOV  r1  [x]
XEND            || XEND
```

$$\underbrace{\underbrace{\begin{array}{ccc} \texttt{r0=1} & \texttt{r0=0} & \texttt{r0=1} \\ \texttt{r1=1} & \texttt{r1=1} & \texttt{r1=0} \end{array}}_{\text{SC}} \begin{array}{c} \texttt{r0=0} \\ \texttt{r1=0} \end{array}}_{\text{x86}}$$

WEAK MEMORY + TM = ?

```
XBEGIN          ||  XBEGIN
MOV  [x] 1      ||  MOV  [y] 1
MOV  r0  [y]    ||  MOV  r1  [x]
XEND            ||  XEND
```

$$\underbrace{\underbrace{\begin{matrix}\texttt{r0=1}\\\texttt{r1=1}\end{matrix} \qquad \underbrace{\begin{matrix}\texttt{r0=0}\\\texttt{r1=1}\end{matrix} \qquad \begin{matrix}\texttt{r0=1}\\\texttt{r1=0}\end{matrix}}_{\text{transactional SC}}}_{\text{SC}} \qquad \begin{matrix}\texttt{r0=0}\\\texttt{r1=0}\end{matrix}}_{\text{x86}}$$

WEAK MEMORY + TM = ?

sequential
consistency

WEAK MEMORY + TM = ?

$$\begin{tikzcd}
\textbf{sequential consistency} & \\
& \textbf{weak consistency} \arrow[ul]
\end{tikzcd}$$

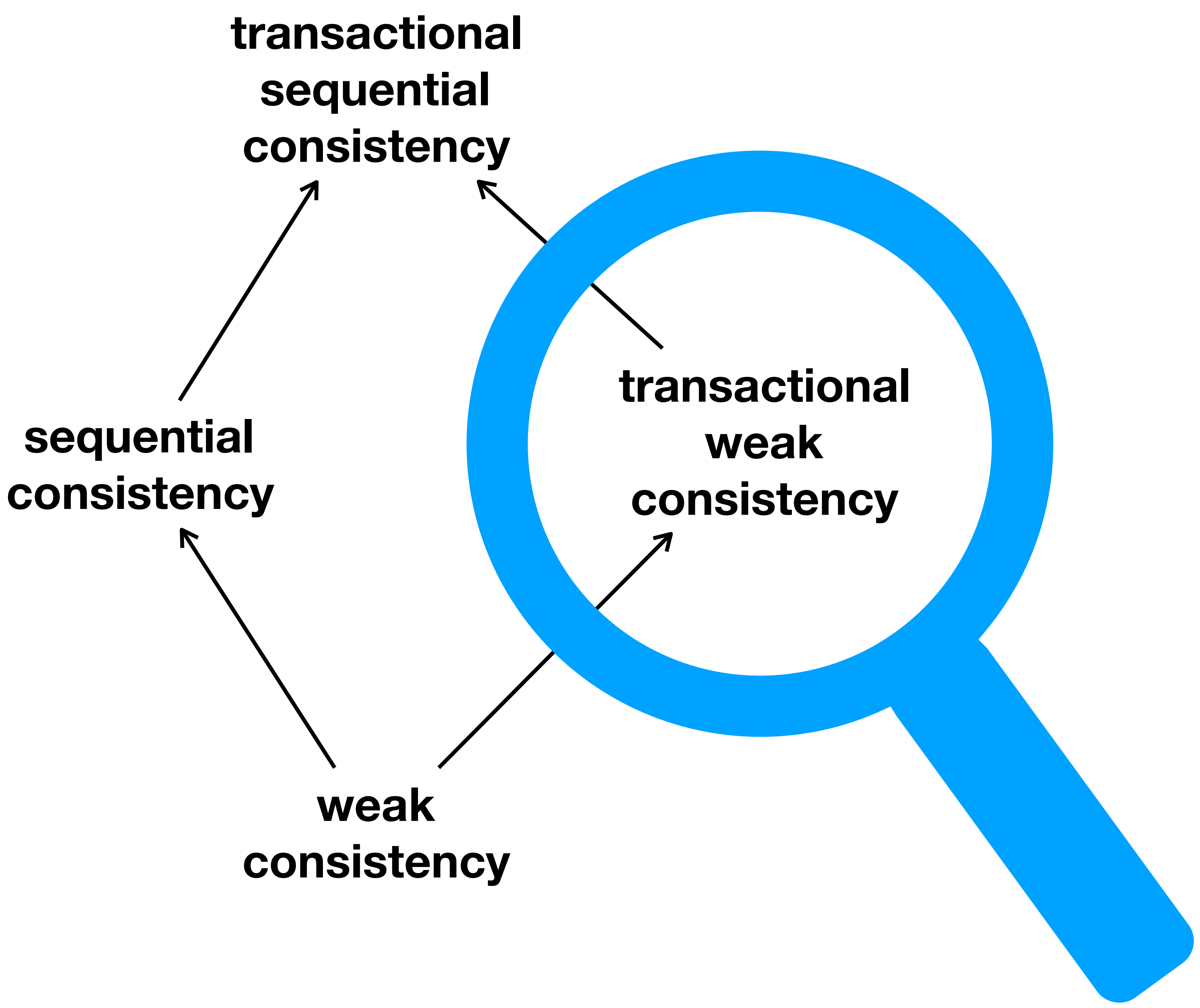
WEAK MEMORY + TM = ?

$$\begin{tikzcd}
 & \text{transactional sequential consistency} \\
\text{sequential consistency} \arrow[ur] & \\
 & \text{weak consistency} \arrow[ul]
\end{tikzcd}$$

WEAK MEMORY + TM = ?

$$\begin{tikzcd}
& \text{transactional sequential consistency} & \\
\text{sequential consistency} \arrow[ur] & & \text{transactional weak consistency} \arrow[ul] \\
& \text{weak consistency} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

WEAK MEMORY + TM = ?


transactional sequential consistency
sequential consistency
transactional weak consistency
weak consistency

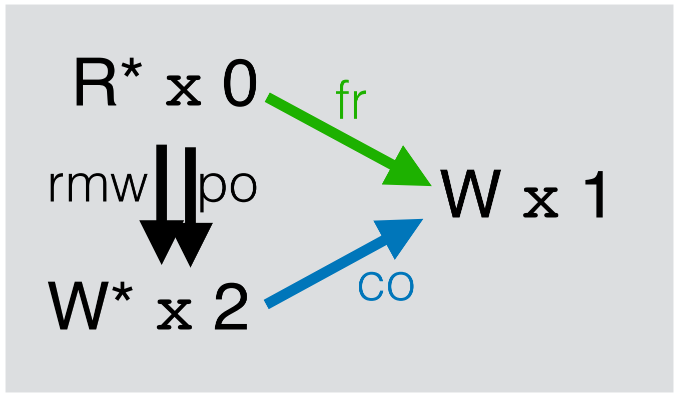
READ-MODIFY-WRITES

READ-MODIFY-WRITES

```
ldxr r1, [x]            ║ str #1, [x]
add  r1, r1, #2         ║
stxr r2, r1, [x]        ║
```

READ-MODIFY-WRITES

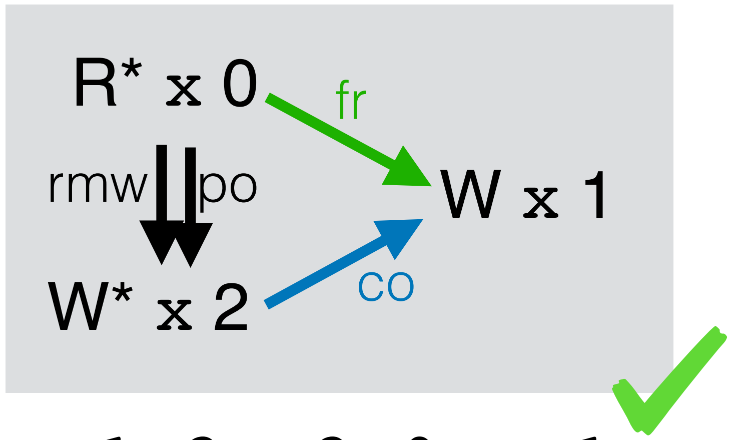
```
ldxr r1, [x]          ‖ str #1, [x]
add  r1, r1, #2       ‖
stxr r2, r1, [x]      ‖
```


$$\begin{tikzcd}
R^* \; x \; 0 \arrow[d, "rmw"', shift right] \arrow[d, "po", shift left] \arrow[r, "fr"] & W \; x \; 1 \\
W^* \; x \; 2 \arrow[ur, "co"'] &
\end{tikzcd}$$

```
r1=2 r2=0 x=1
```

READ-MODIFY-WRITES

```
ldxr r1, [x]          ‖ str #1, [x]
add  r1, r1, #2       ‖
stxr r2, r1, [x]      ‖
```



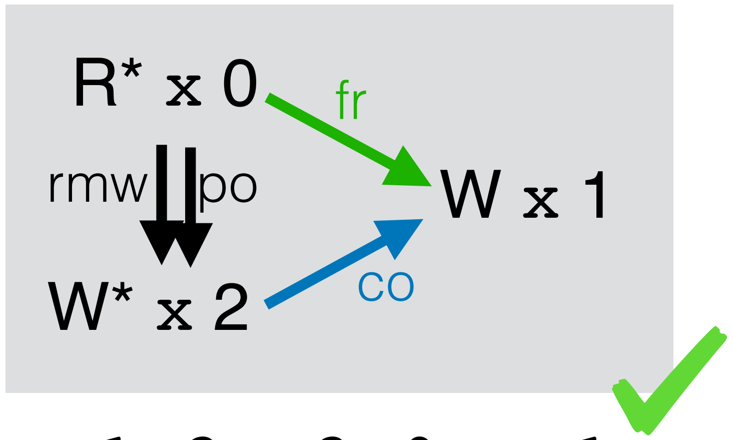
$$\begin{tikzcd}
R^* \; x \; 0 \arrow[d, "rmw"'] \arrow[d, shift left, "po"] \arrow[r, "fr"] & W \; x \; 1 \\
W^* \; x \; 2 \arrow[ur, "co"'] &
\end{tikzcd}$$

✓

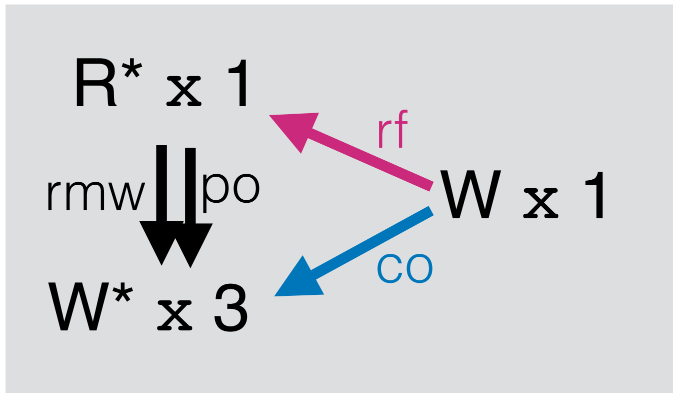
```
r1=2 r2=0 x=1
```

READ-MODIFY-WRITES

```
ldxr r1, [x]          || str #1, [x]
add  r1, r1, #2       ||
stxr r2, r1, [x]      ||
```



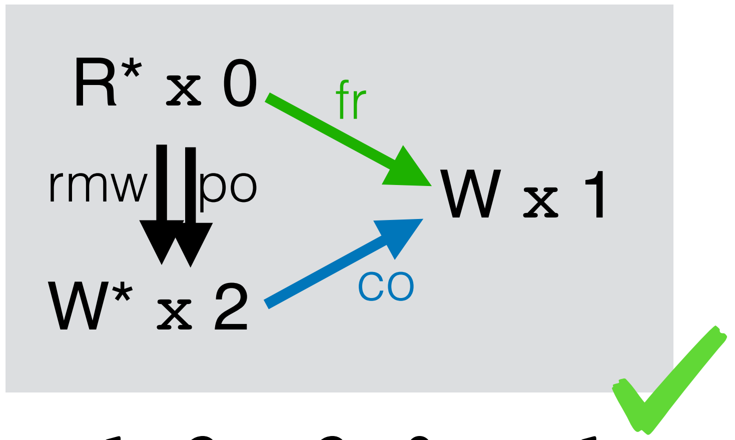
r1=2 r2=0 x=1



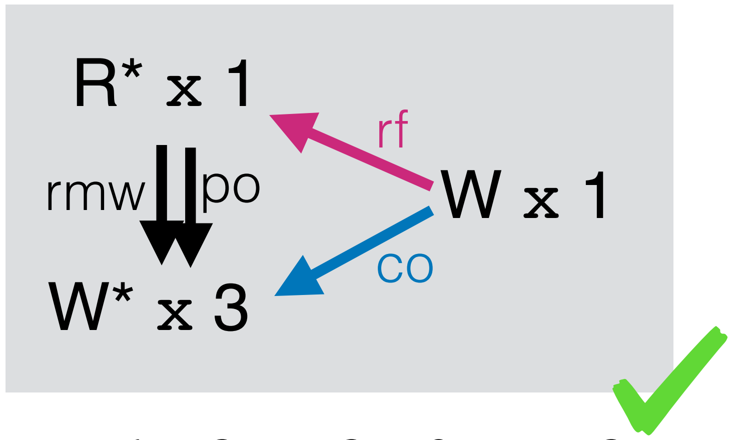
r1=3 r2=0 x=3

READ-MODIFY-WRITES

```
ldxr r1, [x]            ||  str #1, [x]
add  r1, r1, #2
stxr r2, r1, [x]
```



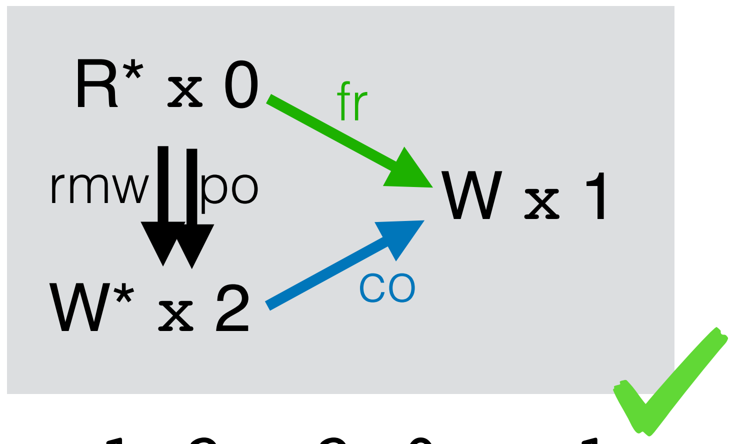
```
r1=2  r2=0  x=1
```



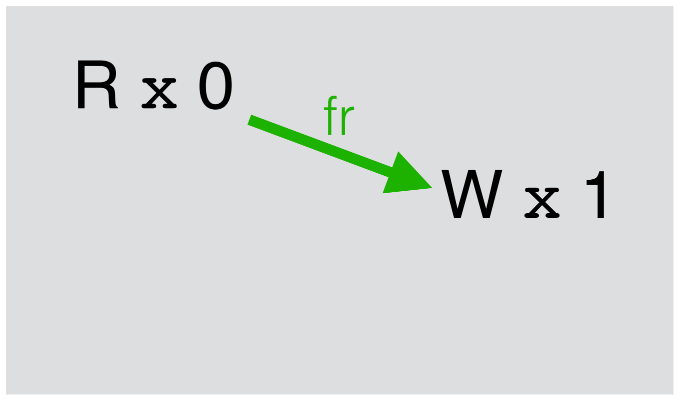
```
r1=3  r2=0  x=3
```

READ-MODIFY-WRITES

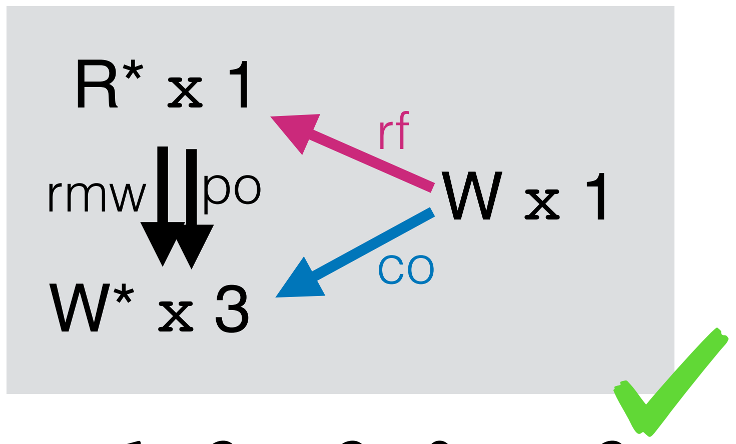
```
ldxr r1, [x]          ||  str #1, [x]
add  r1, r1, #2
stxr r2, r1, [x]
```



r1=2 r2=0 x=1



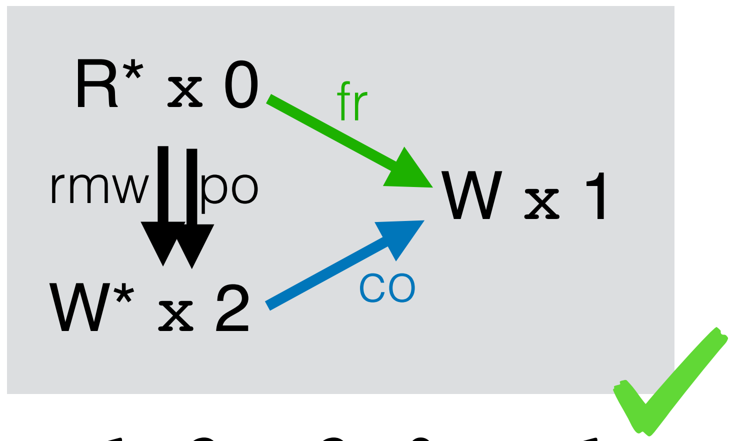
r1=2 r2=1 x=1



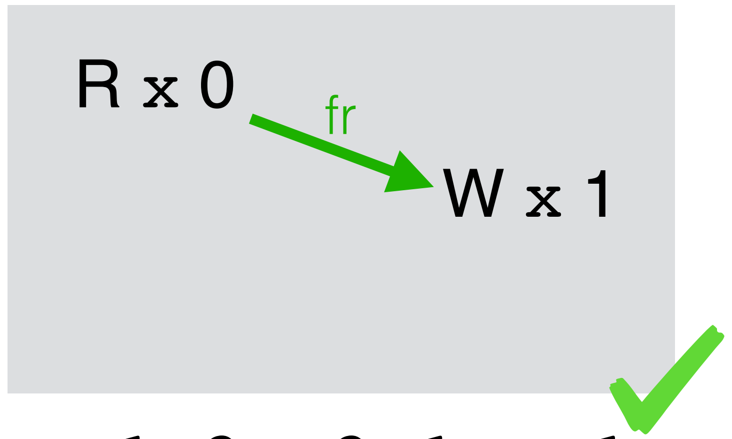
r1=3 r2=0 x=3

READ-MODIFY-WRITES

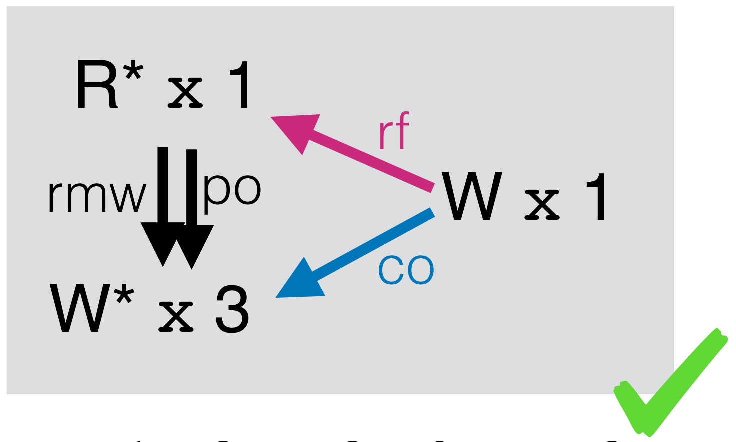
```
ldxr r1, [x]             ║ str #1, [x]
add  r1, r1, #2          ║
stxr r2, r1, [x]         ║
```


$$\begin{tikzcd} R^{*}\ x\ 0 \arrow[d, "rmw"', "po"] \arrow[r, "fr"] & W\ x\ 1 \\ W^{*}\ x\ 2 \arrow[ur, "co"'] & \end{tikzcd}$$
✓

r1=2 r2=0 x=1


$$\begin{tikzcd} R\ x\ 0 \arrow[r, "fr"] & W\ x\ 1 \end{tikzcd}$$
✓

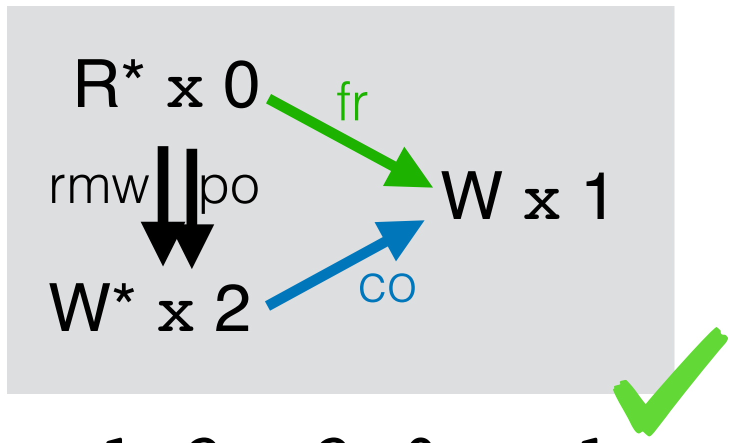
r1=2 r2=1 x=1


$$\begin{tikzcd} R^{*}\ x\ 1 \arrow[d, "rmw"', "po"] & \\ W^{*}\ x\ 3 & W\ x\ 1 \arrow[ul, "rf"'] \arrow[l, "co"] \end{tikzcd}$$
✓

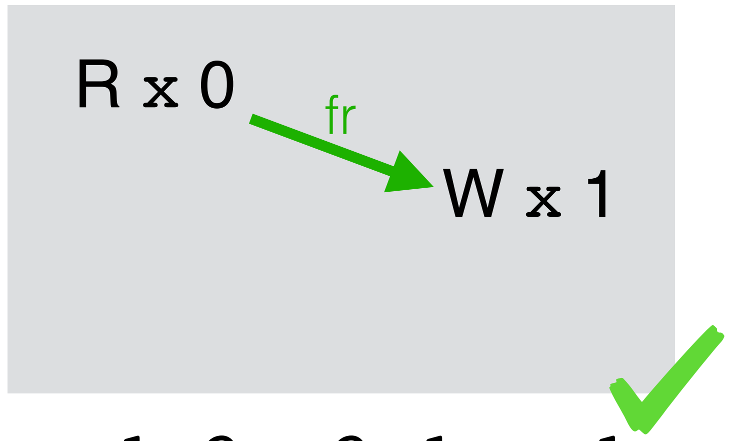
r1=3 r2=0 x=3

READ-MODIFY-WRITES

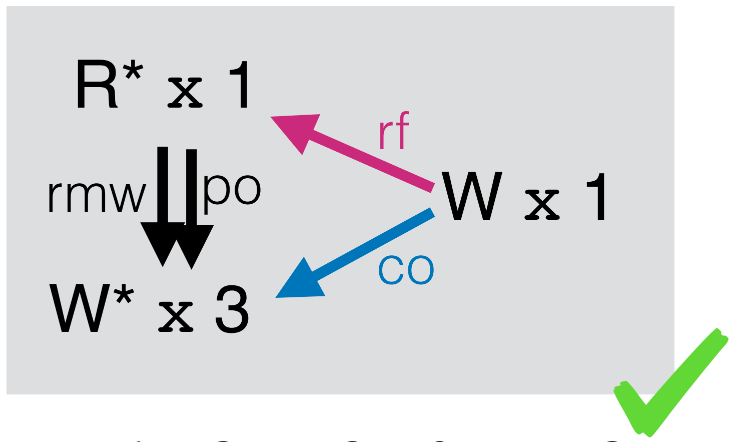
```
ldxr r1, [x]              ||  str #1, [x]
add  r1, r1, #2           ||
stxr r2, r1, [x]          ||
```



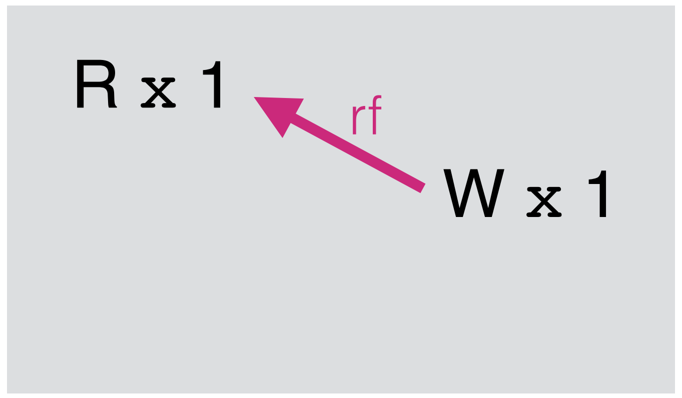
r1=2 r2=0 x=1



r1=2 r2=1 x=1



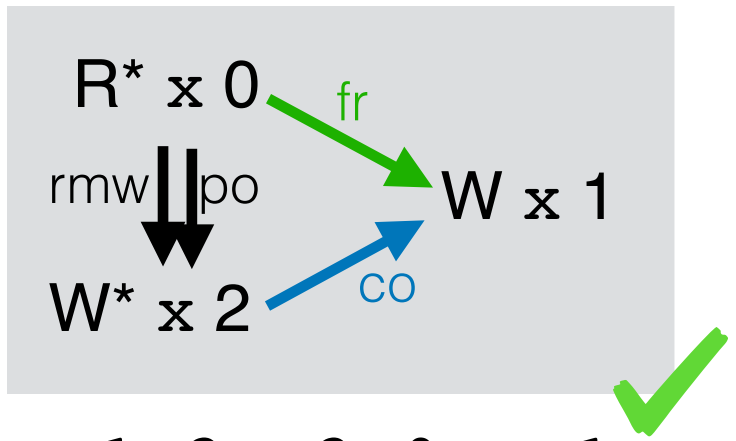
r1=3 r2=0 x=3



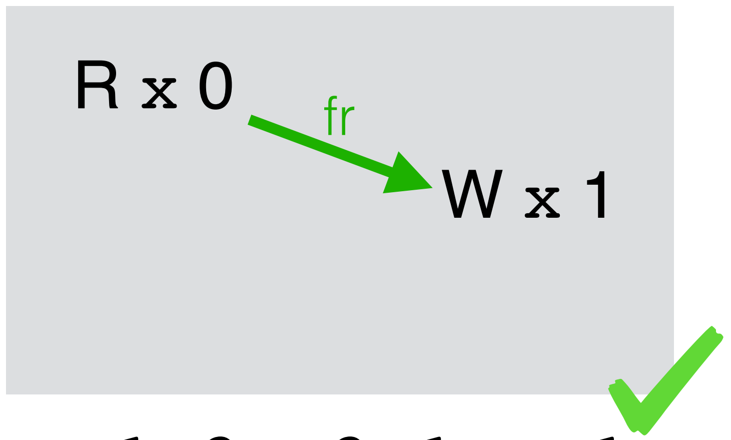
r1=3 r2=1 x=1

READ-MODIFY-WRITES

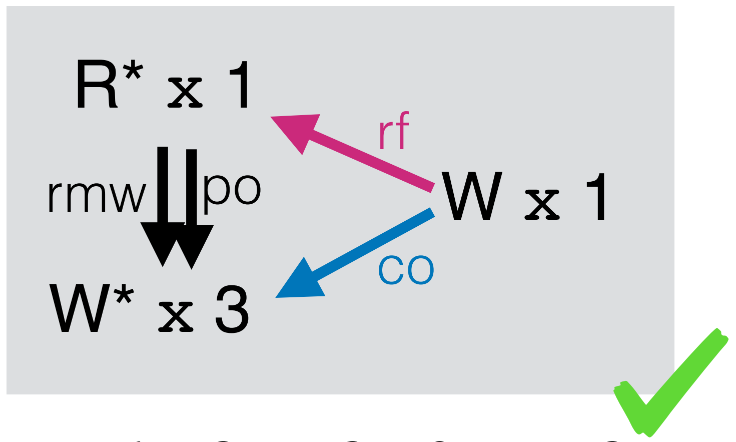
```
ldxr r1, [x]          ||  str #1, [x]
add  r1, r1, #2       ||
stxr r2, r1, [x]      ||
```



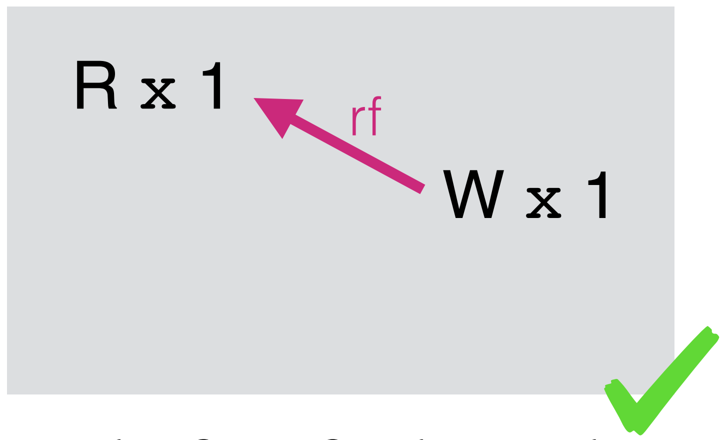
r1=2 r2=0 x=1



r1=2 r2=1 x=1



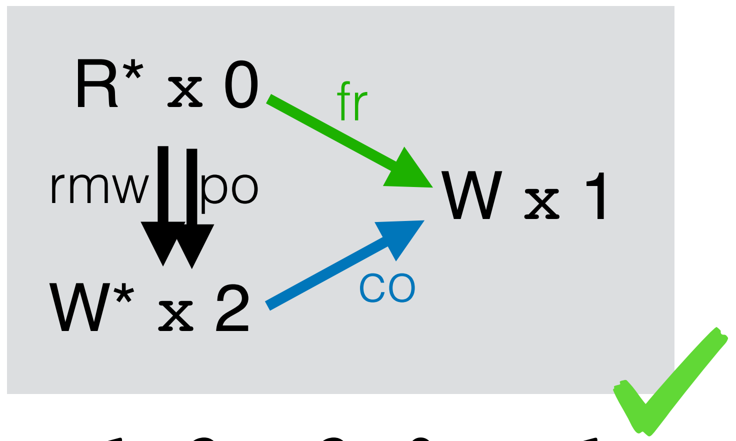
r1=3 r2=0 x=3



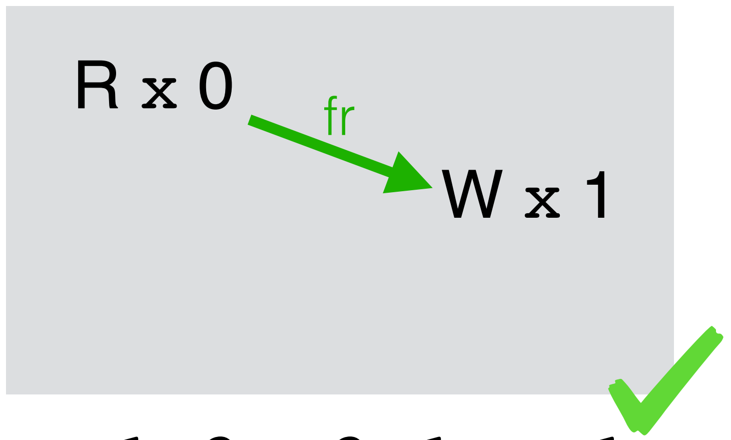
r1=3 r2=1 x=1

READ-MODIFY-WRITES

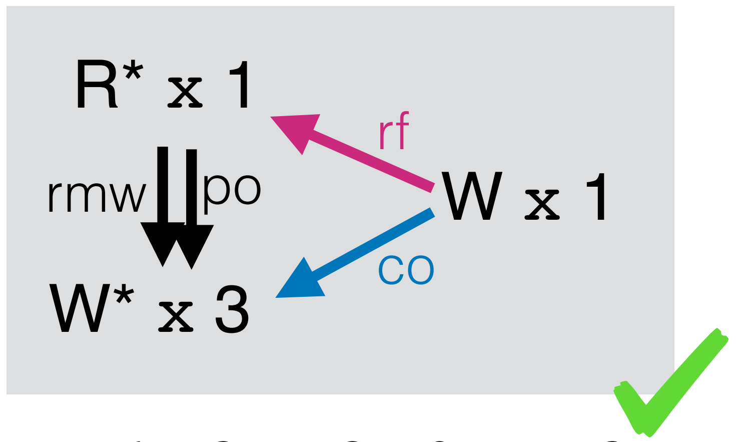
```
ldxr r1, [x]          ||  str #1, [x]
add  r1, r1, #2       ||
stxr r2, r1, [x]      ||
```



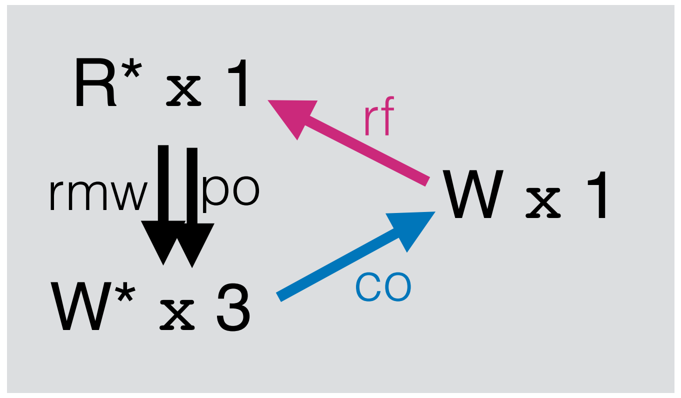
r1=2 r2=0 x=1



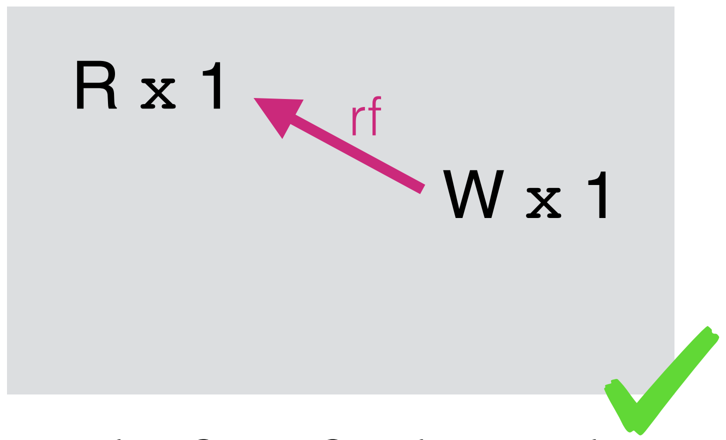
r1=2 r2=1 x=1



r1=3 r2=0 x=3



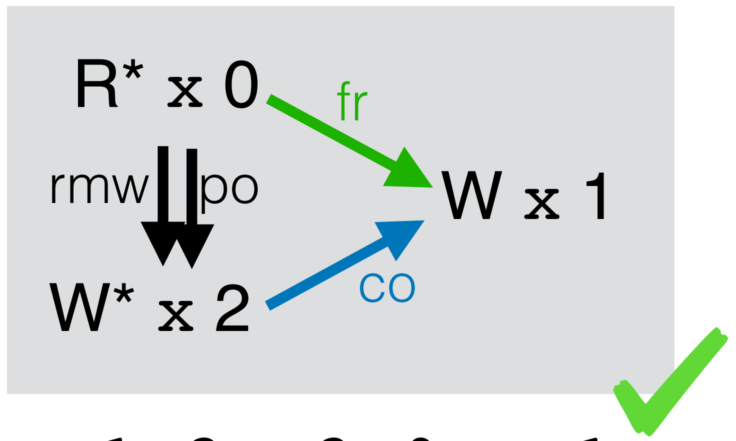
r1=3 r2=0 x=1



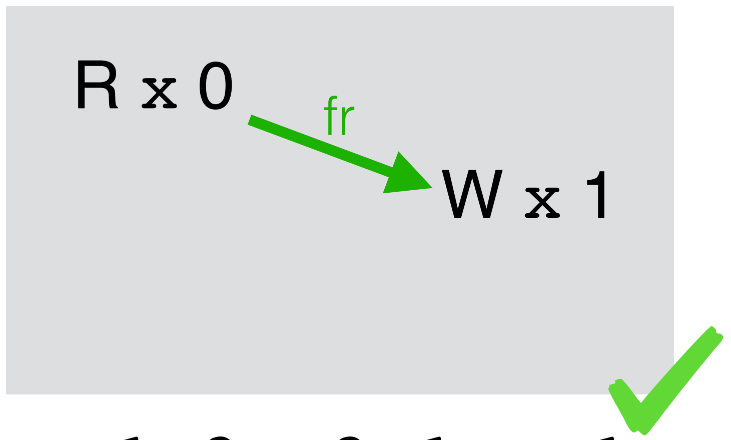
r1=3 r2=1 x=1

READ-MODIFY-WRITES

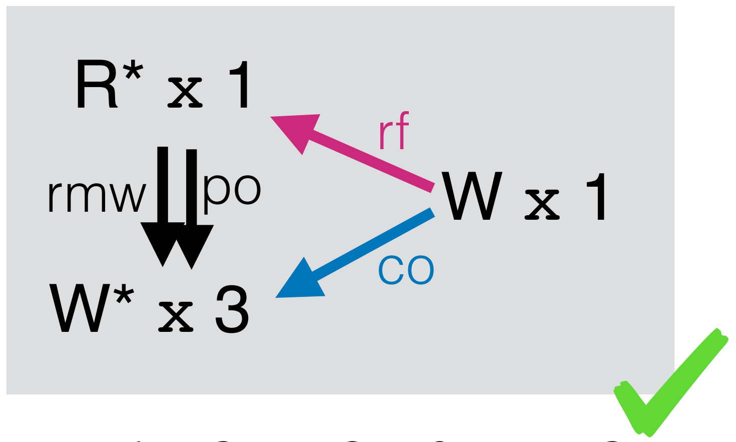
```
ldxr r1, [x]          ||  str #1, [x]
add  r1, r1, #2       ||
stxr r2, r1, [x]      ||
```



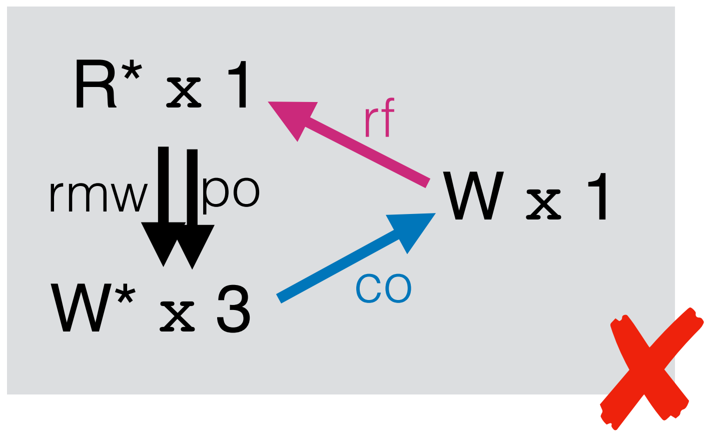
r1=2 r2=0 x=1



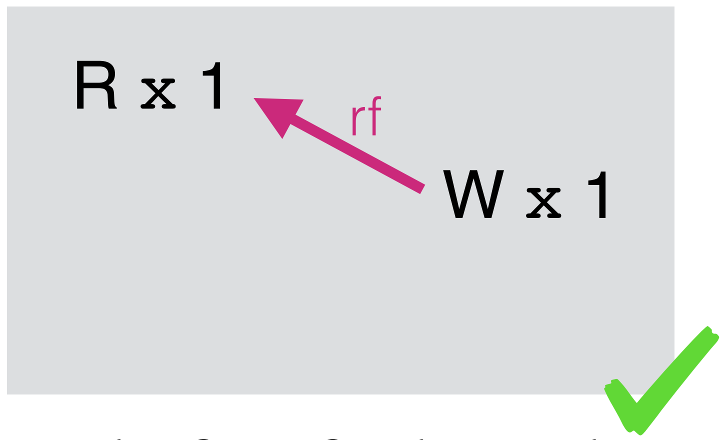
r1=2 r2=1 x=1



r1=3 r2=0 x=3



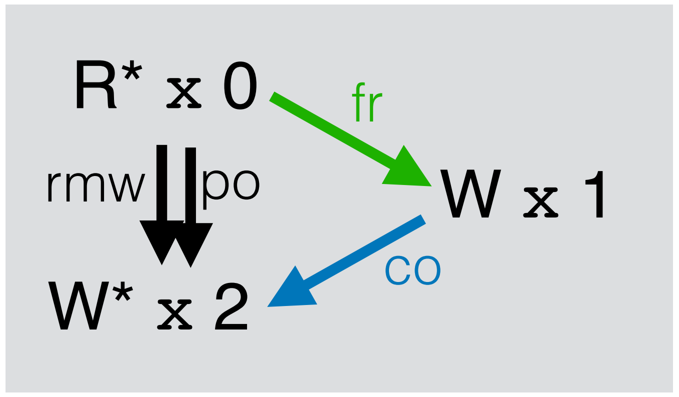
r1=3 r2=0 x=1



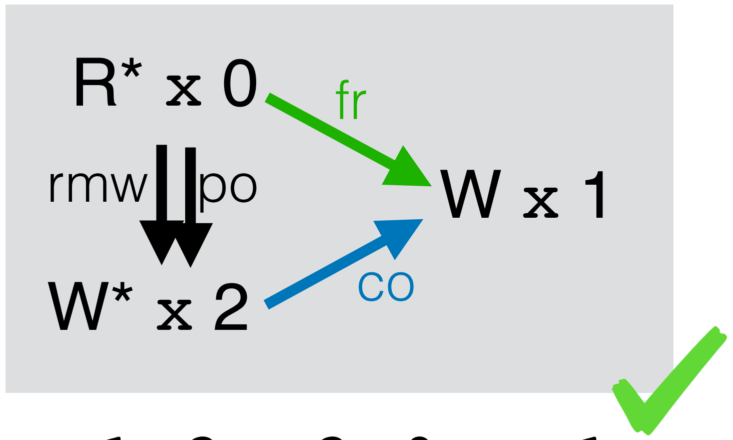
r1=3 r2=1 x=1

READ-MODIFY-WRITES

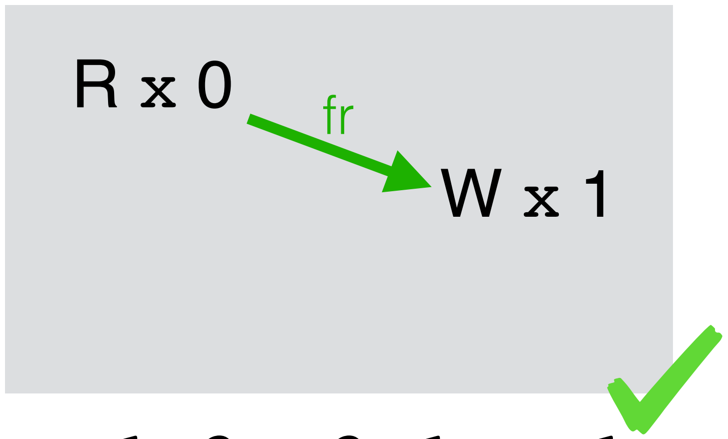
```
ldxr r1, [x]        ||  str #1, [x]
add  r1, r1, #2     ||
stxr r2, r1, [x]    ||
```



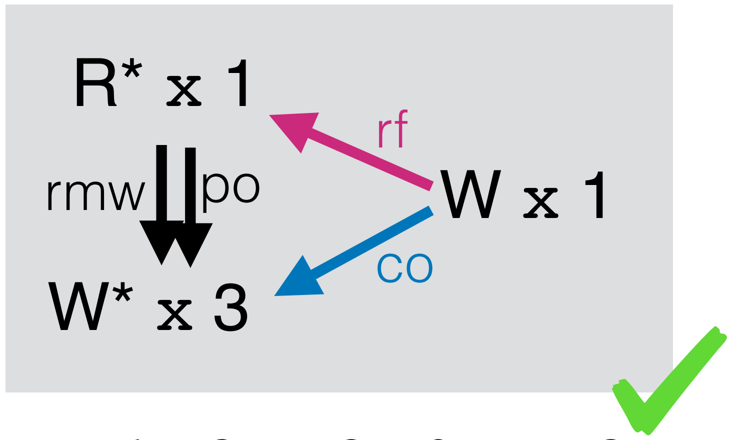
r1=2 r2=0 x=2



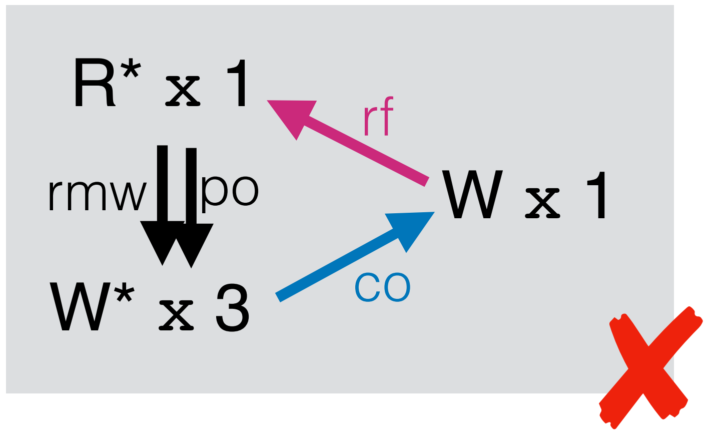
r1=2 r2=0 x=1



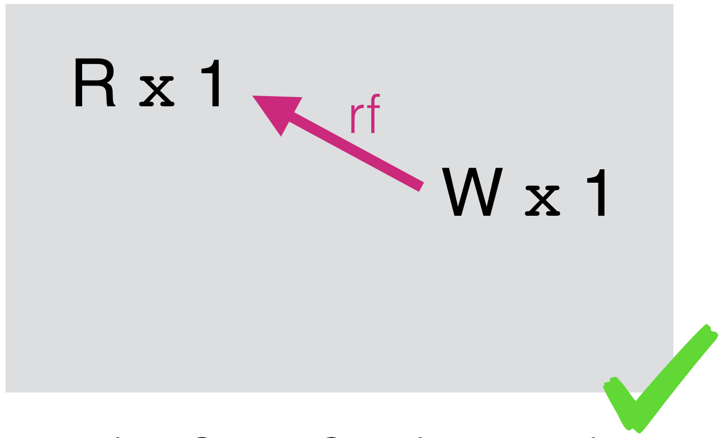
r1=2 r2=1 x=1



r1=3 r2=0 x=3



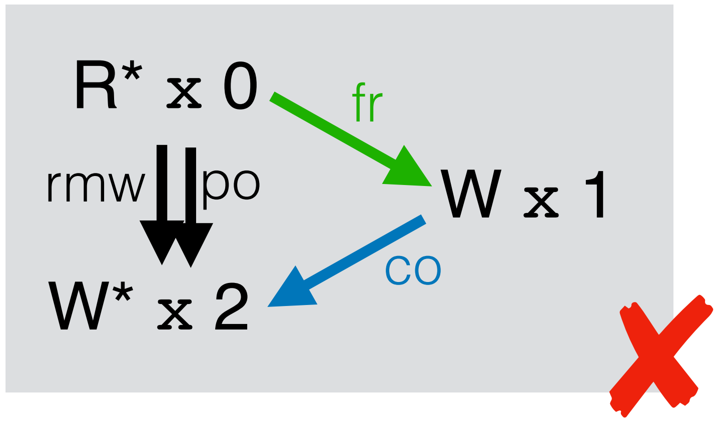
r1=3 r2=0 x=1



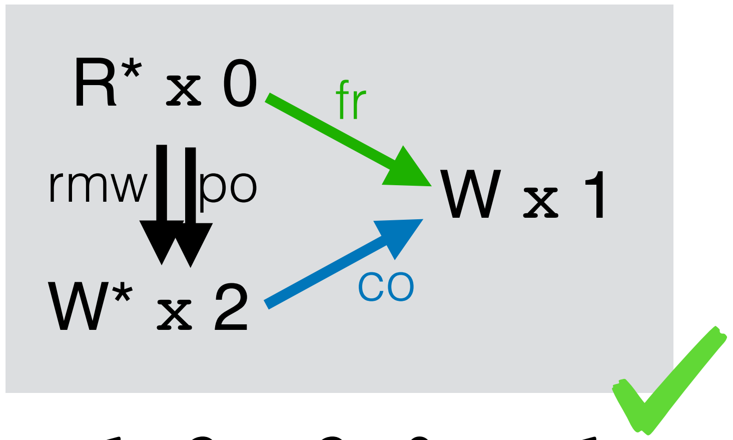
r1=3 r2=1 x=1

READ-MODIFY-WRITES

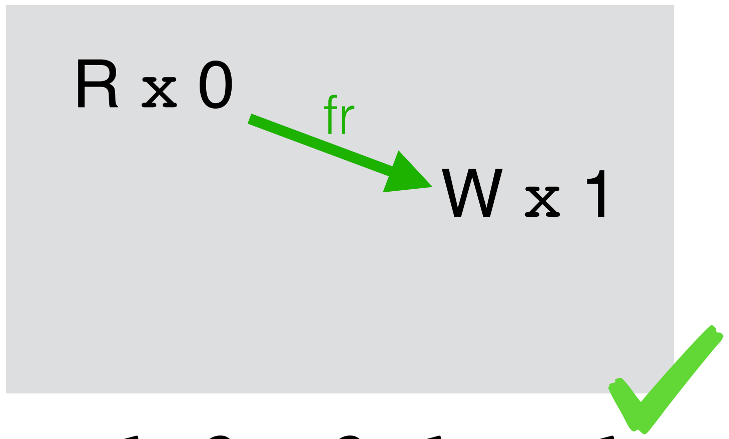
```
ldxr r1, [x]              ||  str #1, [x]
add  r1, r1, #2           ||
stxr r2, r1, [x]          ||
```



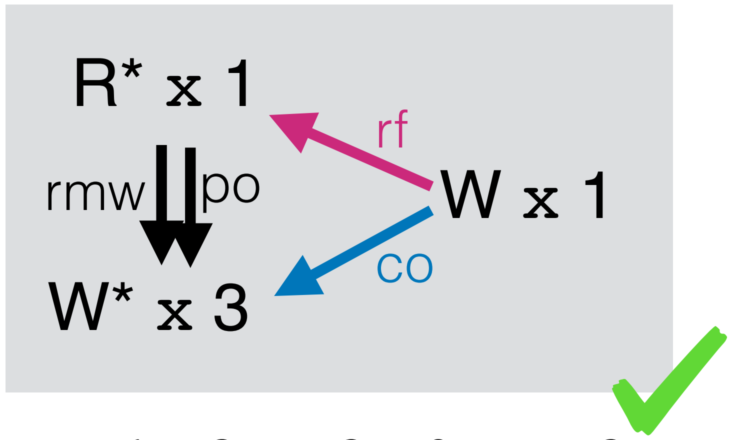
r1=2 r2=0 x=2



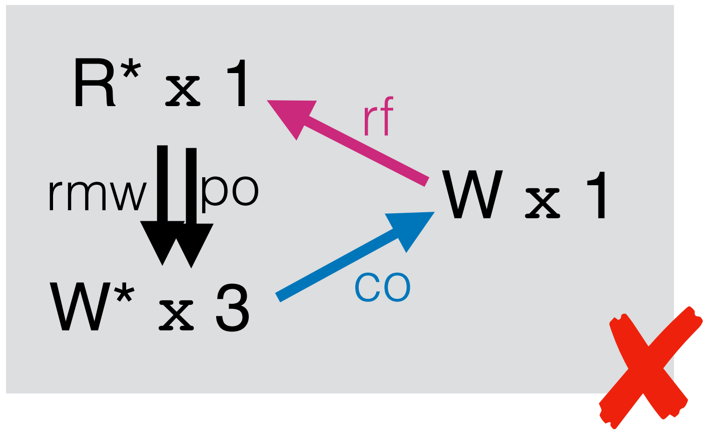
r1=2 r2=0 x=1



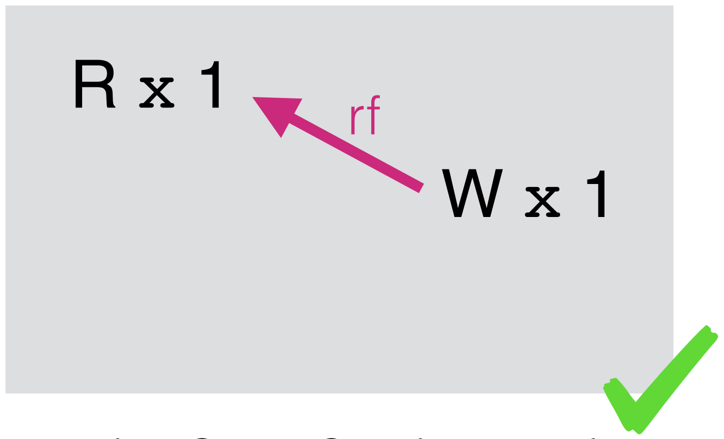
r1=2 r2=1 x=1



r1=3 r2=0 x=3

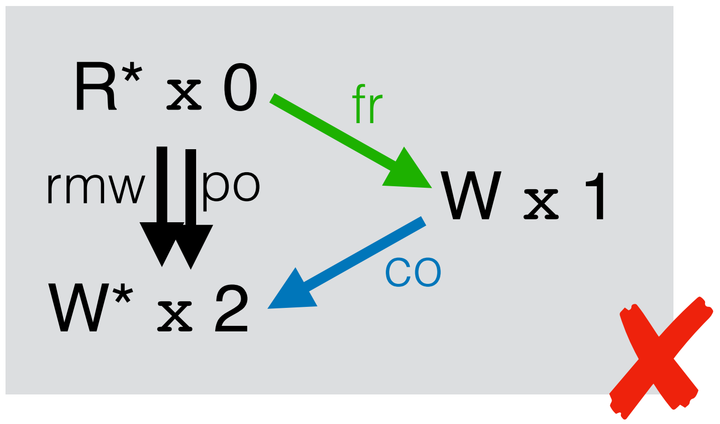


r1=3 r2=0 x=1



r1=3 r2=1 x=1

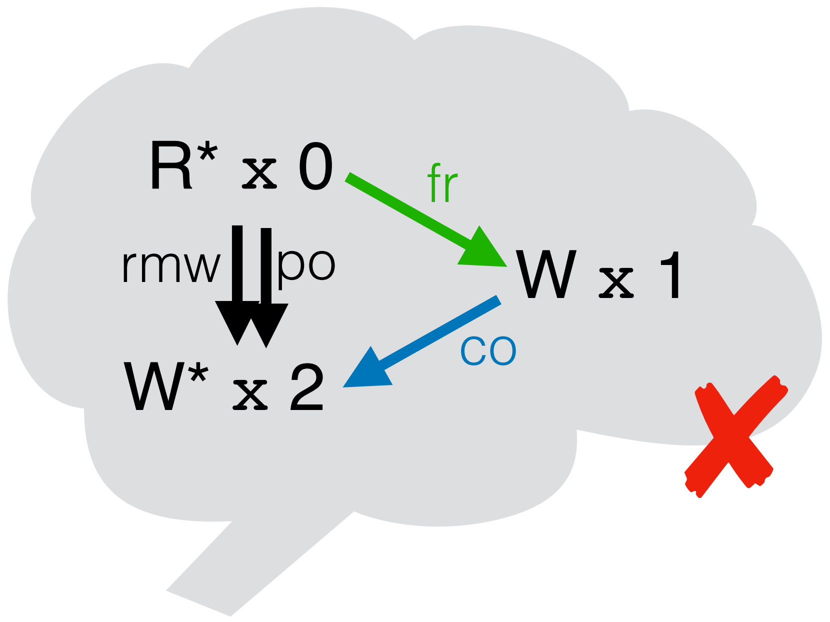
READ-MODIFY-WRITES


$$\begin{tikzcd}
R^* \; x \; 0 \arrow[d, Rightarrow, "rmw"'] \arrow[d, "po"] \arrow[dr, "fr"] & \\
 & W \; x \; 1 \arrow[dl, "co"] \\
W^* \; x \; 2 &
\end{tikzcd}$$

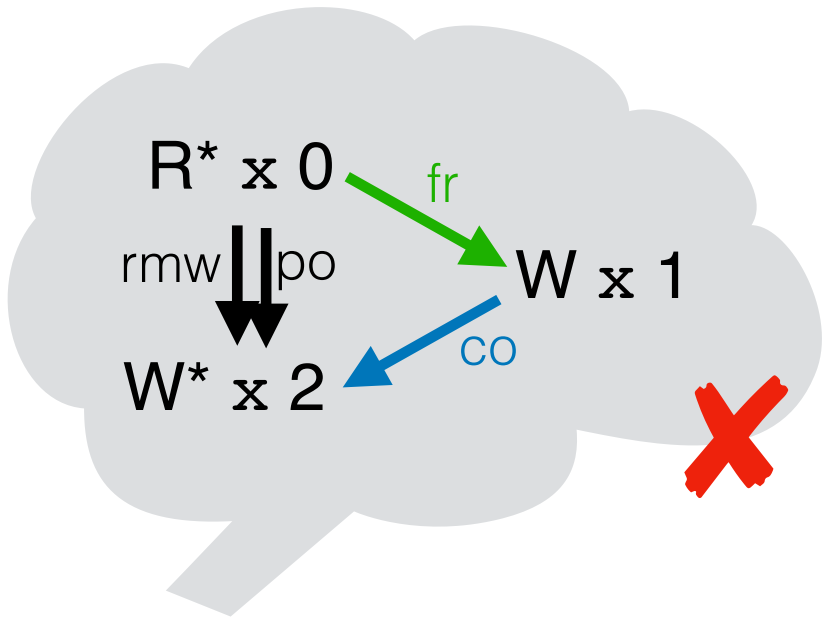
✗

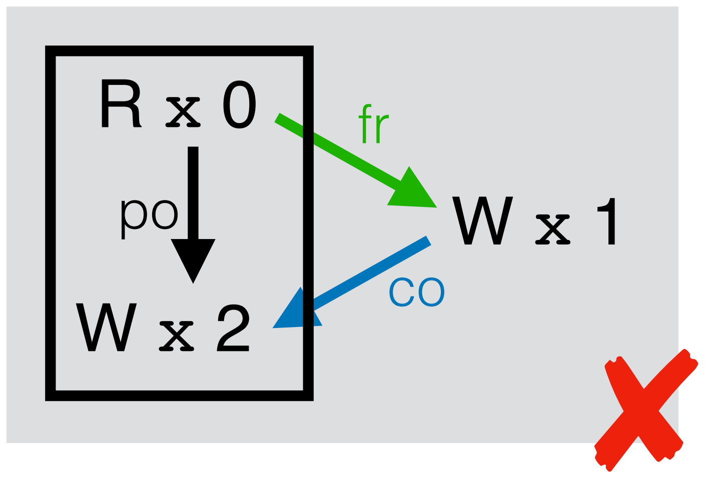
`r1=2 r2=0 x=2`

AXIOMS FOR TRANSACTIONS

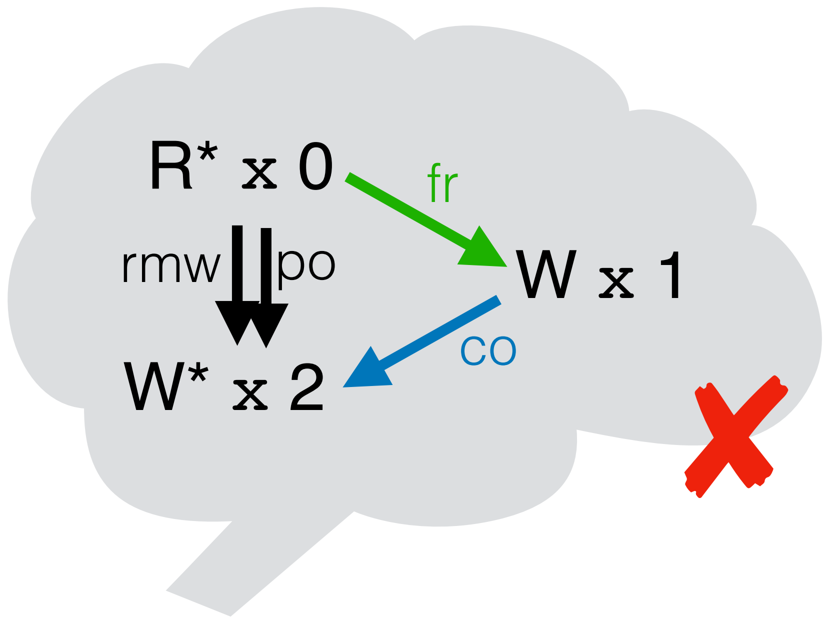


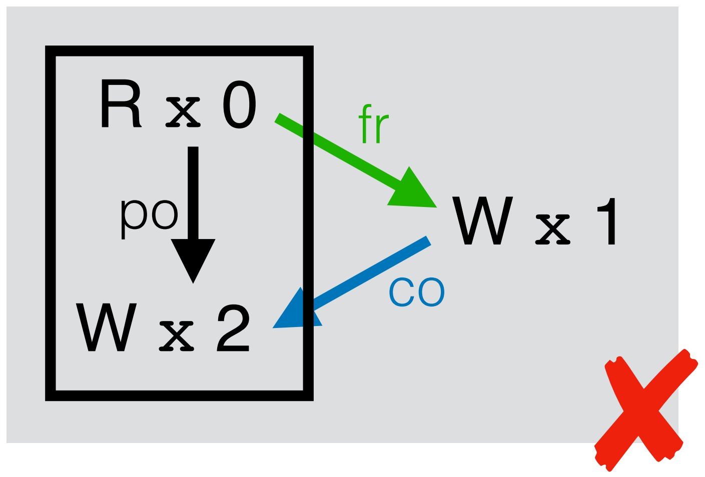
AXIOMS FOR TRANSACTIONS

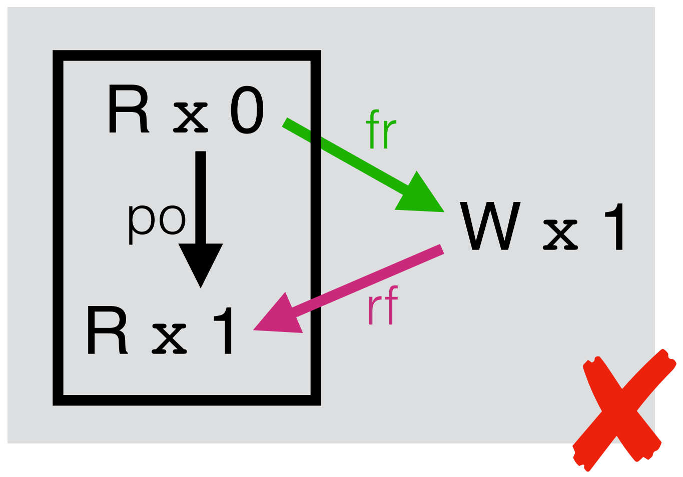




AXIOMS FOR TRANSACTIONS







AXIOMS FOR TRANSACTIONS

$$\begin{tikzcd}
R^* \text{ x } 0 \arrow[dr, "fr"] \arrow[dd, shift right, "rmw"'] \arrow[dd, shift left, "po"] & \\
 & W \text{ x } 1 \arrow[dl, "co"] \\
W^* \text{ x } 2 &
\end{tikzcd}$$

✗

$$\begin{tikzcd}
\boxed{R \text{ x } 0} \arrow[dr, "fr"] \arrow[dd, "po"'] & \\
 & W \text{ x } 1 \arrow[dl, "co"] \\
W \text{ x } 2 &
\end{tikzcd}$$

✗

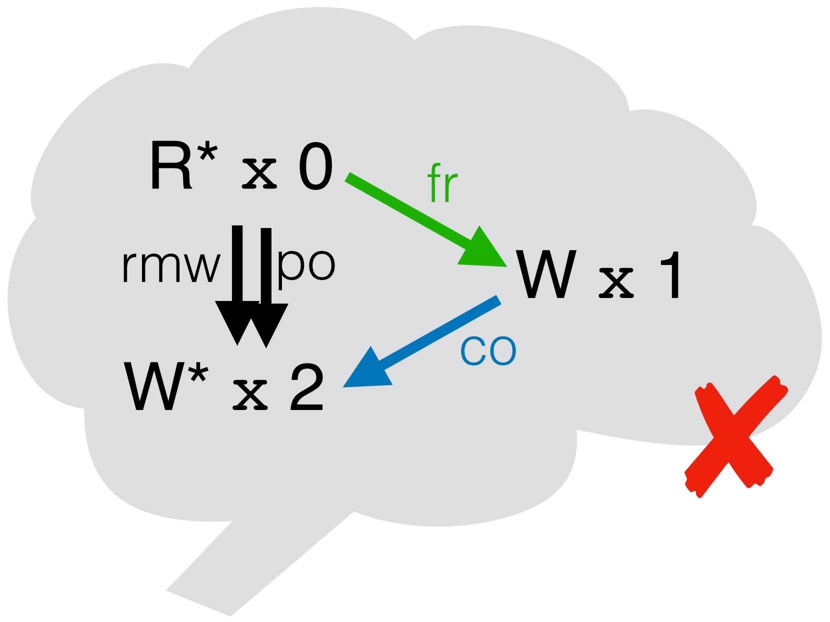
$$\begin{tikzcd}
R \text{ x } 0 \arrow[dr, "fr"] \arrow[dd, "po"'] & \\
 & W \text{ x } 1 \arrow[dl, "rf"] \\
R \text{ x } 1 &
\end{tikzcd}$$

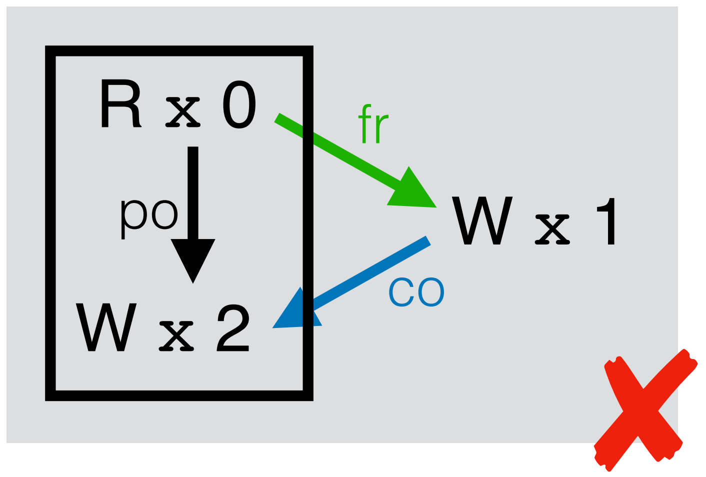
✗

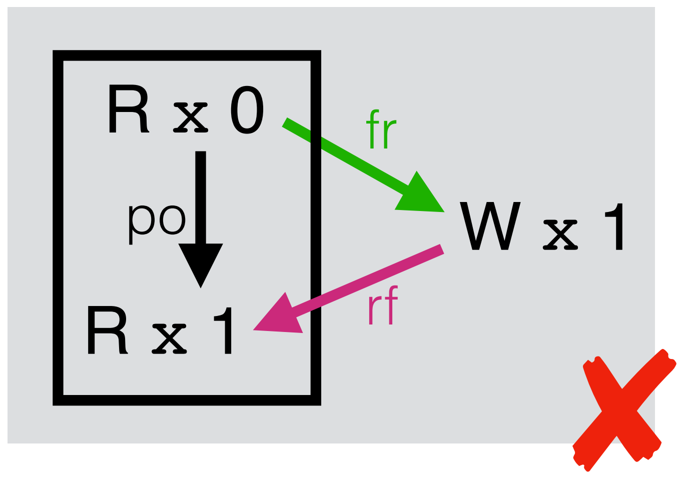
$$\begin{tikzcd}
W \text{ x } 1 \arrow[dr, "rf"] \arrow[dd, "po"'] & \\
 & R \text{ x } 1 \arrow[dl, "fr"] \\
W \text{ x } 2 &
\end{tikzcd}$$

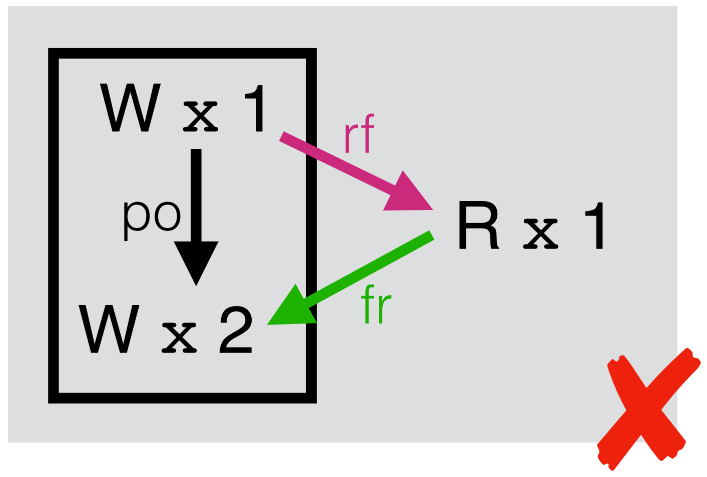
✗

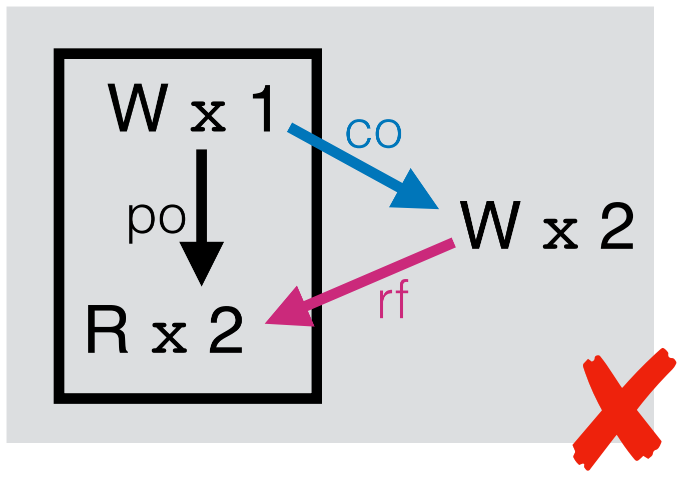
AXIOMS FOR TRANSACTIONS











X86 TRANSACTIONS

$$\textbf{acyclic}(po_{\text{loc}} \cup com) \qquad \text{(Coherence)}$$

$$\textbf{empty}(rmw \cap (fr_{\text{e}} \mathbin{;} co_{\text{e}})) \qquad \text{(RMWIsol)}$$

$$\textbf{acyclic}(hb) \qquad \text{(Order)}$$

$$\text{where } ppo = ((W \times W) \cup (R \times W) \cup (R \times R)) \cap po$$

$$tfence = po \cap ((\neg stxn \mathbin{;} stxn) \cup (stxn \mathbin{;} \neg stxn))$$

$$L = \text{domain}(rmw) \cup \text{range}(rmw)$$

$$implied = [L] \mathbin{;} po \cup po \mathbin{;} [L] \cup tfence$$

$$hb = mfence \cup ppo \cup implied \cup rf_{\text{e}} \cup fr \cup co$$

$$\textbf{acyclic}(\text{stronglift}(com, stxn)) \qquad \text{(StrongIsol)}$$

$$\textbf{acyclic}(\text{stronglift}(hb, stxn)) \qquad \text{(TxnOrder)}$$

ARM TRANSACTIONS

$$\textbf{acyclic}(po_{\mathrm{loc}} \cup com) \qquad (\textsc{Coherence})$$

$$\textbf{acyclic}(ob) \qquad (\textsc{Order})$$

$$\begin{aligned}
\text{where } dob &= \textit{(order imposed by dependencies, elided)} \\
aob &= \textit{(order imposed by atomic RMWs, elided)} \\
bob &= \textit{(order imposed by barriers, elided)} \\
tfence &= po \cap ((\neg stxn \,;\, stxn) \cup (stxn \,;\, \neg stxn)) \\
ob &= com_{\mathrm{e}} \cup dob \cup aob \cup bob \cup tfence
\end{aligned}$$

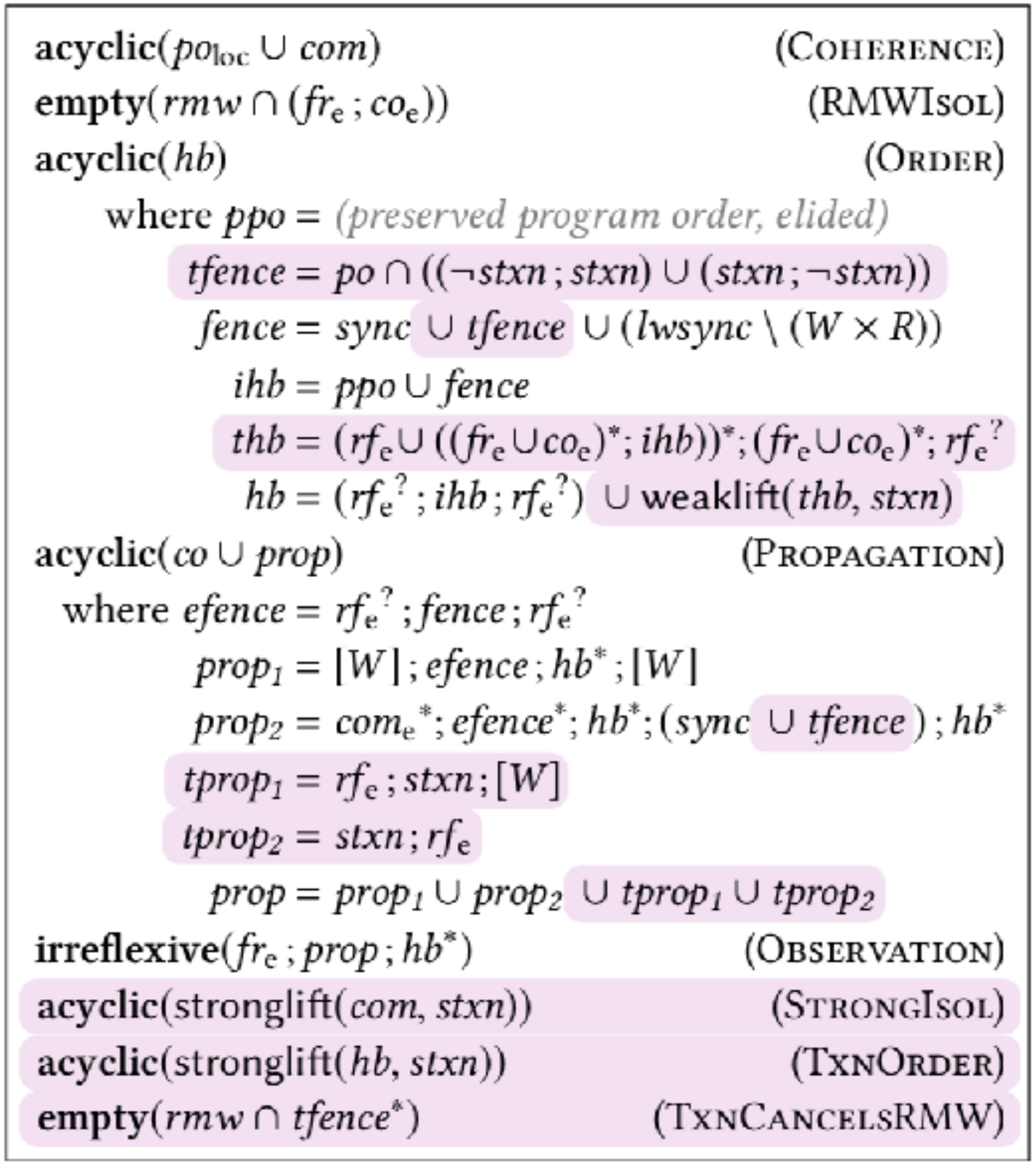
$$\textbf{empty}(rmw \cap (fr_{\mathrm{e}} \,;\, co_{\mathrm{e}})) \qquad (\textsc{RMWIsol})$$

$$\textbf{acyclic}(\mathrm{stronglift}(com, stxn)) \qquad (\textsc{StrongIsol})$$

$$\textbf{acyclic}(\mathrm{stronglift}(ob, stxn)) \qquad (\textsc{TxnOrder})$$

$$\textbf{empty}(rmw \cap tfence^{*}) \qquad (\textsc{TxnCancelsRMW})$$

POWER TRANSACTIONS



$\textbf{acyclic}(po_{\text{loc}} \cup com)$ (Coherence)

$\textbf{empty}(rmw \cap (fr_e ; co_e))$ (RMWIsol)

$\textbf{acyclic}(hb)$ (Order)

where $ppo =$ *(preserved program order, elided)*

$tfence = po \cap ((\neg stxn ; stxn) \cup (stxn ; \neg stxn))$

$fence = sync \cup tfence \cup (lwsync \setminus (W \times R))$

$ihb = ppo \cup fence$

$thb = (rf_e \cup ((fr_e \cup co_e)^* ; ihb))^* ; (fr_e \cup co_e)^* ; rf_e{}^?$

$hb = (rf_e{}^? ; ihb ; rf_e{}^?) \cup \text{weaklift}(thb, stxn)$

$\textbf{acyclic}(co \cup prop)$ (Propagation)

where $efence = rf_e{}^? ; fence ; rf_e{}^?$

$prop_1 = [W] ; efence ; hb^* ; [W]$

$prop_2 = com_e{}^* ; efence^* ; hb^* ; (sync \cup tfence) ; hb^*$

$tprop_1 = rf_e ; stxn ; [W]$

$tprop_2 = stxn ; rf_e$

$prop = prop_1 \cup prop_2 \cup tprop_1 \cup tprop_2$

$\textbf{irreflexive}(fr_e ; prop ; hb^*)$ (Observation)

$\textbf{acyclic}(\text{stronglift}(com, stxn))$ (StrongIsol)

$\textbf{acyclic}(\text{stronglift}(hb, stxn))$ (TxnOrder)

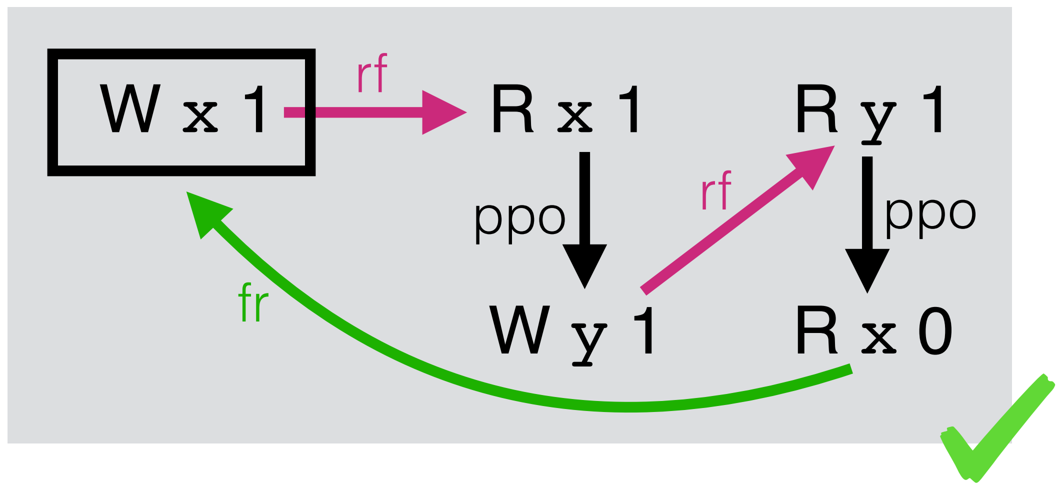
$\textbf{empty}(rmw \cap tfence^*)$ (TxnCancelsRMW)

POWER TRANSACTIONS

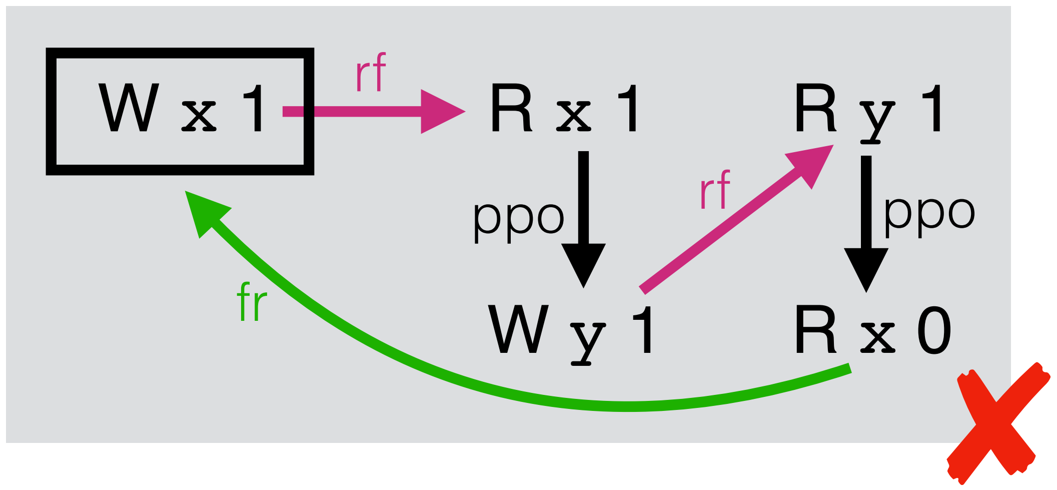
$$\begin{tikzcd}
\mathrm{W}\ \mathtt{x}\ 1 \arrow[r, "\mathrm{rf}"] & \mathrm{R}\ \mathtt{x}\ 1 \arrow[d, "\mathrm{ppo}"'] & \mathrm{R}\ \mathtt{y}\ 1 \arrow[d, "\mathrm{ppo}"] \\
 & \mathrm{W}\ \mathtt{y}\ 1 \arrow[ur, "\mathrm{rf}"] & \mathrm{R}\ \mathtt{x}\ 0 \arrow[ull, bend left, "\mathrm{fr}"]
\end{tikzcd}$$

✓

POWER TRANSACTIONS



POWER TRANSACTIONS

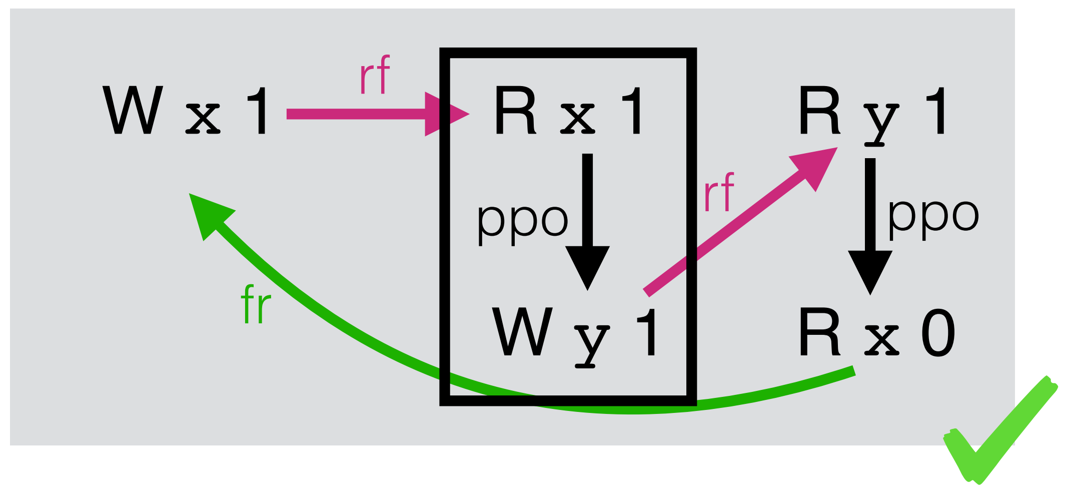


POWER TRANSACTIONS

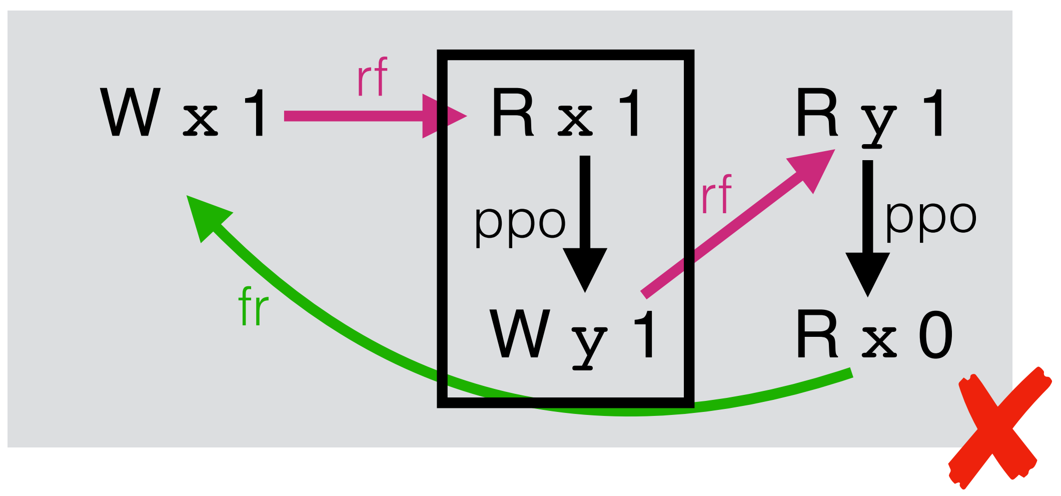
$$\begin{tikzcd}
\text{W x 1} \arrow[r, "\text{rf}"] & \text{R x 1} \arrow[d, "\text{ppo}"'] & \text{R y 1} \arrow[d, "\text{ppo}"] \\
 & \text{W y 1} \arrow[ur, "\text{rf}"] & \text{R x 0} \arrow[ull, bend left, "\text{fr}"]
\end{tikzcd}$$

✓

POWER TRANSACTIONS



POWER TRANSACTIONS



POWER TRANSACTIONS

$$\begin{tikzcd}
W\ x\ 1 \arrow[r, "rf"] & R\ x\ 1 \arrow[d, "ppo"'] & R\ y\ 1 \arrow[d, "ppo"] & W\ y\ 1 \arrow[l, "rf"'] \\
 & R\ y\ 0 \arrow[urr, bend right, "fr"'] & R\ x\ 0 \arrow[ull, bend left, "fr"] &
\end{tikzcd}$$

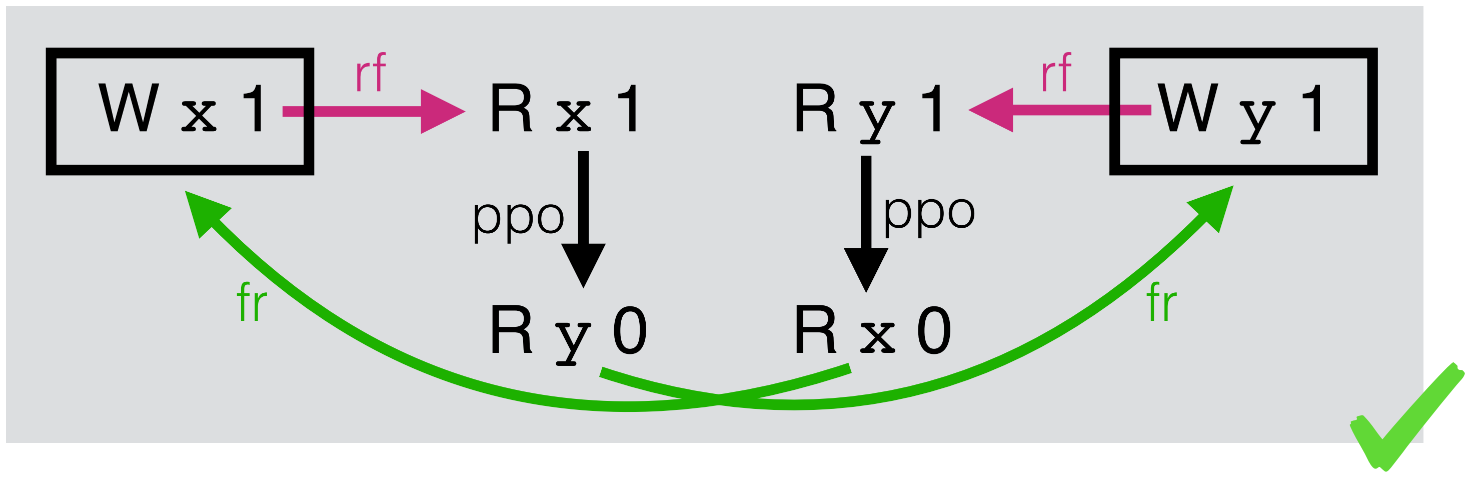
✓

POWER TRANSACTIONS

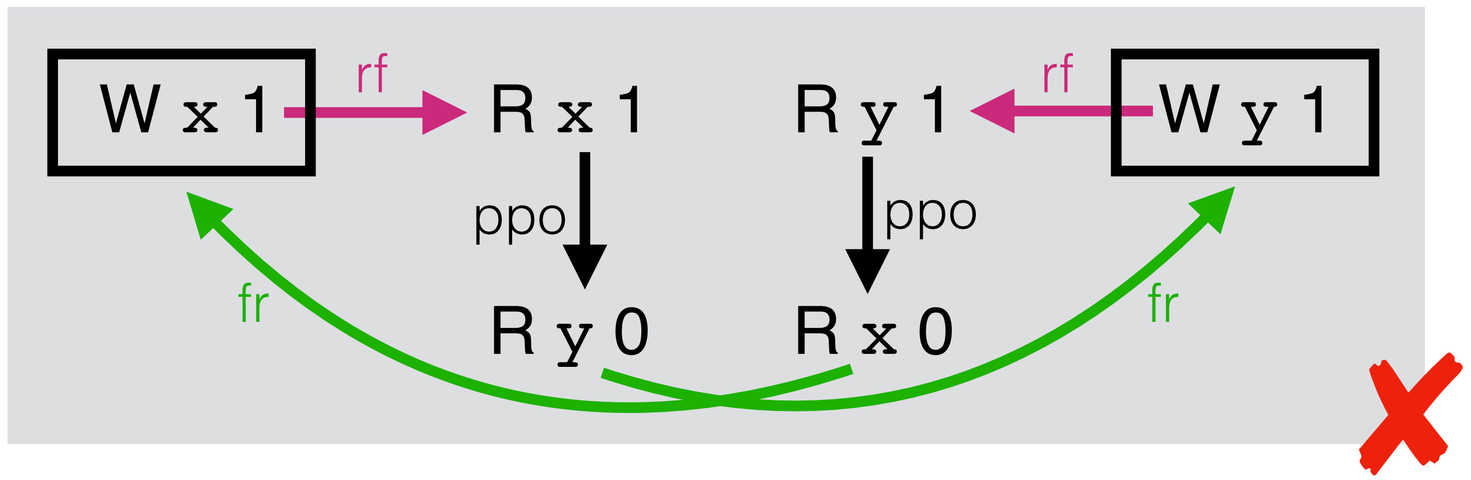
$$\begin{tikzcd}
\boxed{\mathrm{W}\ \mathtt{x}\ 1} \arrow[r, "rf"] & \mathrm{R}\ \mathtt{x}\ 1 \arrow[d, "ppo"'] & \mathrm{R}\ \mathtt{y}\ 1 \arrow[d, "ppo"] & \mathrm{W}\ \mathtt{y}\ 1 \arrow[l, "rf"'] \\
 & \mathrm{R}\ \mathtt{y}\ 0 \arrow[urr, bend right, "fr"'] & \mathrm{R}\ \mathtt{x}\ 0 \arrow[ull, bend left, "fr"] &
\end{tikzcd}$$

✓

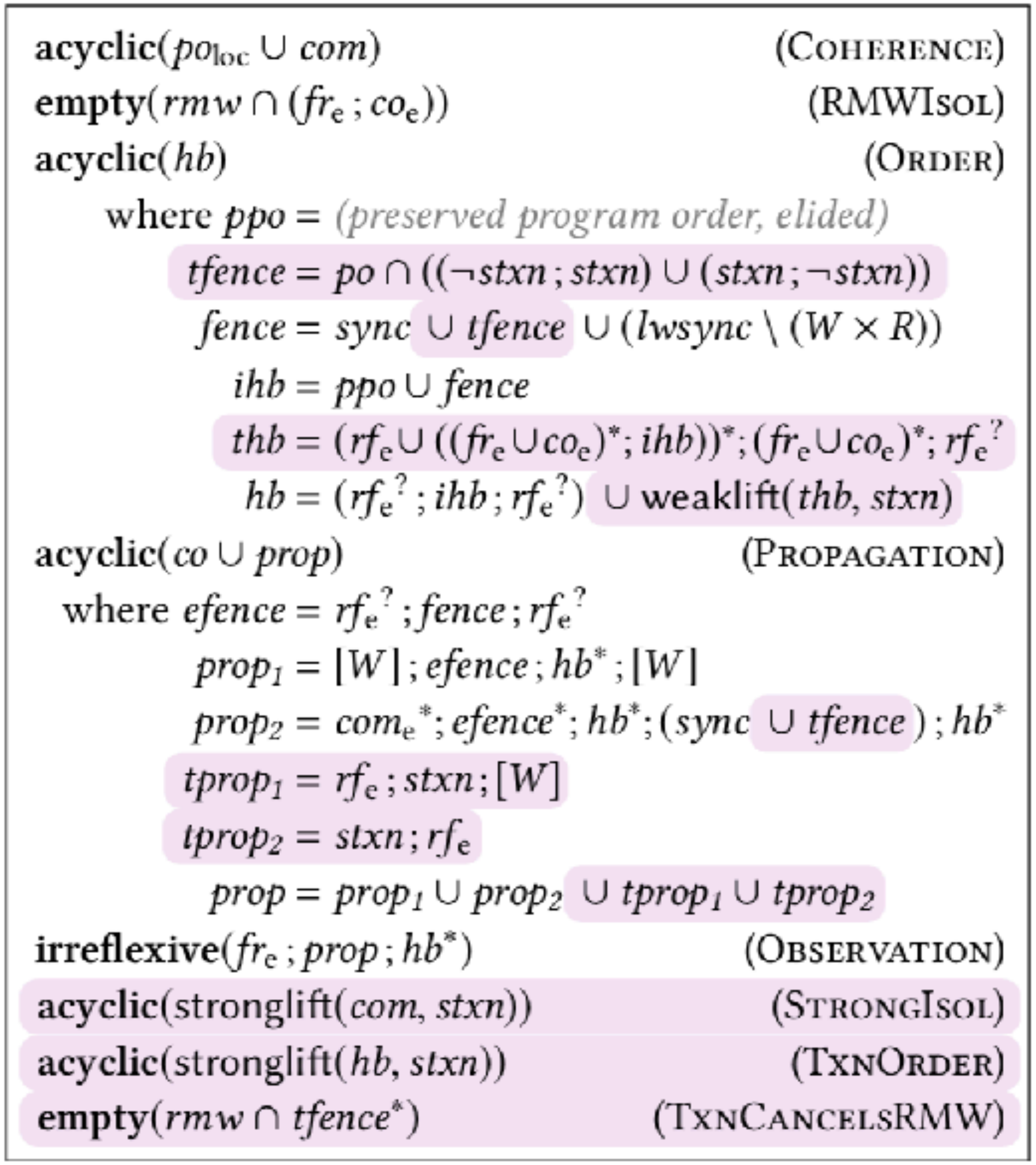
POWER TRANSACTIONS



POWER TRANSACTIONS



POWER TRANSACTIONS



acyclic($po_{\text{loc}} \cup com$) (COHERENCE)

empty($rmw \cap (fr_e ; co_e)$) (RMWISOL)

acyclic(hb) (ORDER)

where $ppo =$ *(preserved program order, elided)*

$tfence = po \cap ((\neg stxn ; stxn) \cup (stxn ; \neg stxn))$

$fence = sync \cup tfence \cup (lwsync \setminus (W \times R))$

$ihb = ppo \cup fence$

$thb = (rf_e \cup ((fr_e \cup co_e)^* ; ihb))^* ; (fr_e \cup co_e)^* ; rf_e^?$

$hb = (rf_e^? ; ihb ; rf_e^?) \cup \mathrm{weaklift}(thb, stxn)$

acyclic($co \cup prop$) (PROPAGATION)

where $efence = rf_e^? ; fence ; rf_e^?$

$prop_1 = [W] ; efence ; hb^* ; [W]$

$prop_2 = com_e^* ; efence^* ; hb^* ; (sync \cup tfence) ; hb^*$

$tprop_1 = rf_e ; stxn ; [W]$

$tprop_2 = stxn ; rf_e$

$prop = prop_1 \cup prop_2 \cup tprop_1 \cup tprop_2$

irreflexive($fr_e ; prop ; hb^*$) (OBSERVATION)

acyclic(stronglift($com, stxn$)) (STRONGISOL)

acyclic(stronglift($hb, stxn$)) (TXNORDER)

empty($rmw \cap tfence^*$) (TXNCANCELSRMW)

C++ TRANSACTIONS

$\textbf{irreflexive}(hb\,;com^*)$ (HbCom)

where $sw =$ *(synchronises-with, elided)*

$ecom = com \cup (co\,;rf)$

$tsw = \mathrm{weaklift}(ecom, stxn)$

$hb = (sw \cup tsw \cup po)^+$

$\textbf{empty}(rmw \cap (fr_e\,;co_e))$ (RMWIsol)

$\textbf{acyclic}(po \cup rf)$ (NoThinAir)

$\textbf{acyclic}(psc)$ (SeqCst)

where $psc =$ *(constraints on SC events, elided)*

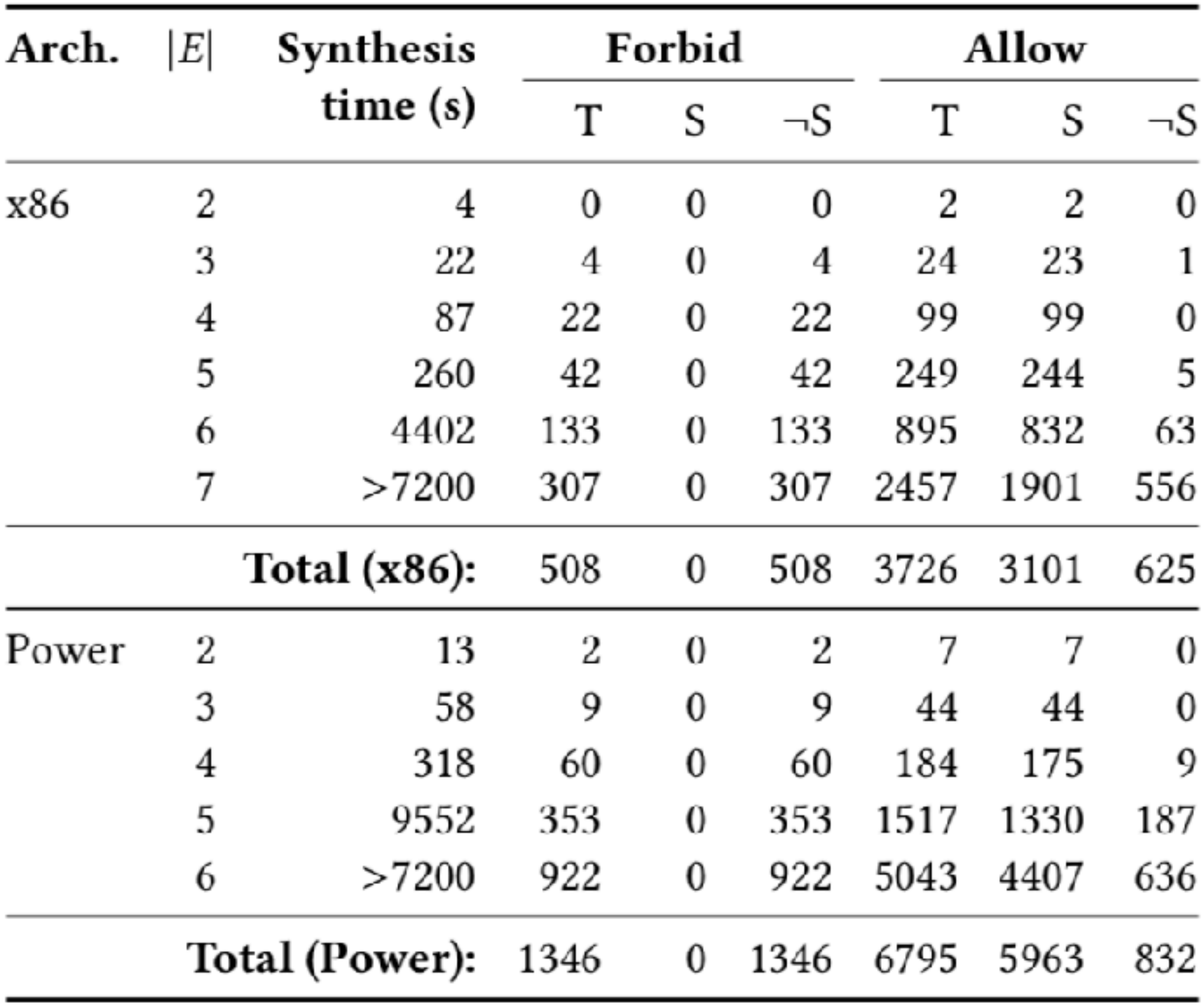
$\textbf{empty}(cnf \setminus Ato^2 \setminus (hb \cup hb^{-1}))$ (NoRace)

where $cnf = ((W \times W) \cup (R \times W) \cup (W \times R)) \cap sloc \setminus id$

OUTLINE

- ~~Weak memory~~
- ~~Transactions~~
- ~~Weak memory and transactions~~
- Validating our models
- The problem with lock elision
- Related and future work

MODEL VALIDATION



Arch.	|*E*|	Synthesis time (s)	Forbid			Allow		
			T	S	¬S	T	S	¬S
x86	2	4	0	0	0	2	2	0
	3	22	4	0	4	24	23	1
	4	87	22	0	22	99	99	0
	5	260	42	0	42	249	244	5
	6	4402	133	0	133	895	832	63
	7	>7200	307	0	307	2457	1901	556
		Total (x86):	508	0	508	3726	3101	625
Power	2	13	2	0	2	7	7	0
	3	58	9	0	9	44	44	0
	4	318	60	0	60	184	175	9
	5	9552	353	0	353	1517	1330	187
	6	>7200	922	0	922	5043	4407	636
		Total (Power):	1346	0	1346	6795	5963	832

MODEL VALIDATION

- Adding/coalescing/extending transactions should not introduce new behaviours.

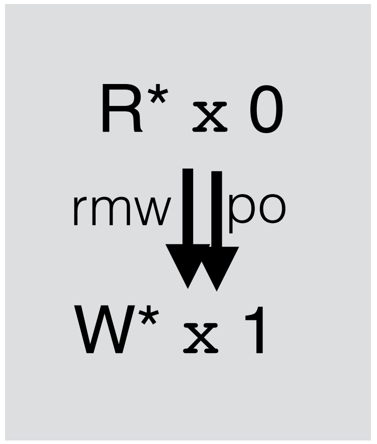
MODEL VALIDATION

- Adding/coalescing/extending transactions should not introduce new behaviours.
- Counterexample:

MODEL VALIDATION

- Adding/coalescing/extending transactions should not introduce new behaviours.

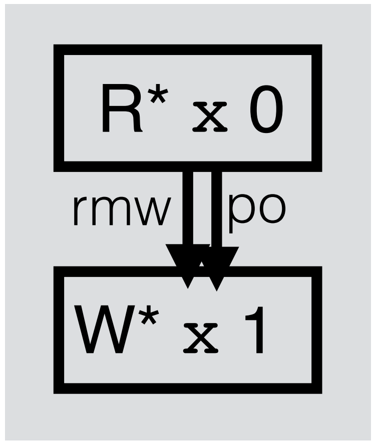
- Counterexample:



MODEL VALIDATION

- Adding/coalescing/extending transactions should not introduce new behaviours.

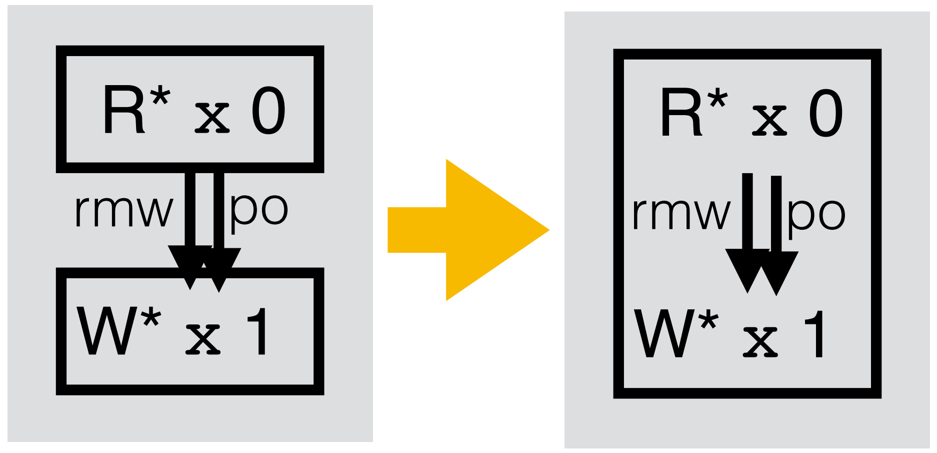
- Counterexample:



MODEL VALIDATION

- Adding/coalescing/extending transactions should not introduce new behaviours.

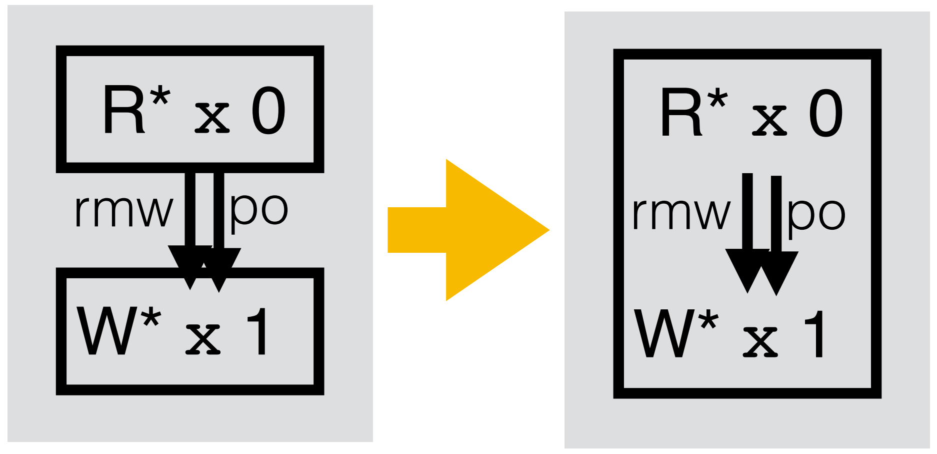
- Counterexample:



MODEL VALIDATION

- Adding/coalescing/extending transactions should not introduce new behaviours.

- Counterexample:



- C++ transactions compile soundly to x86/Power/Arm transactions via the usual mapping

OUTLINE

- ~~Weak memory~~
- ~~Transactions~~
- ~~Weak memory and transactions~~
- ~~Validating our models~~
- The problem with lock elision
- Related and future work

LOCK ELISION

x := x+2

```
lock()
ldr   W5,[X0]
add   W5,W5,#2
str   W5,[X0]
unlock()
```

```
lock()
mov W7,#1
str W7,[X0]
unlock()
```

x := 1

LOCK ELISION

```
              Loop:
              ldaxr W2,[X1]        txbegin
              cbnz  W2,Loop        ldr W6,[X1]
              mov   W3,#1          cbz W6,L1
              stxr  W4,W3,[X1]     txabort
              cbnz  W4,Loop        L1:
            { ldr   W5,[X0]        mov W7,#1    }
x := x+2    { add   W5,W5,#2       str W7,[X0]  }  x := 1
            { str   W5,[X0]        txend        }
              stlr  WZR,[X1]
```

LOCK ELISION

```
Loop:
ldaxr W2,[X1]
cbnz  W2,Loop
mov   W3,#1
stxr  W4,W3,[X1]
cbnz  W4,Loop
ldr   W5,[X0]
add   W5,W5,#2
str   W5,[X0]
stlr  WZR,[X1]
```

```
txbegin
ldr W6,[X1]
cbz W6,L1
txabort
L1:
mov W7,#1
str W7,[X0]
txend
```

LOCK ELISION

```
 Loop:
✓ldaxr W2,[X1]       txbegin
 cbnz  W2,Loop       ldr W6,[X1]
 mov   W3,#1         cbz W6,L1
 stxr  W4,W3,[X1]    txabort
 cbnz  W4,Loop       L1:
 ldr   W5,[X0]       mov W7,#1
 add   W5,W5,#2      str W7,[X0]
 str   W5,[X0]       txend
 stlr  WZR,[X1]
```

LOCK ELISION

```
 Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
 mov   W3,#1
 stxr  W4,W3,[X1]
 cbnz  W4,Loop
 ldr   W5,[X0]
 add   W5,W5,#2
 str   W5,[X0]
 stlr  WZR,[X1]
```

```
txbegin
ldr W6,[X1]
cbz W6,L1
txabort
L1:
mov W7,#1
str W7,[X0]
txend
```

LOCK ELISION

```
  Loop:
✓ldaxr W2,[X1]          txbegin
✓cbnz  W2,Loop          ldr W6,[X1]
 mov   W3,#1            cbz W6,L1
 stxr  W4,W3,[X1]       txabort
 cbnz  W4,Loop          L1:
✓ldr   W5,[X0]          mov W7,#1
 add   W5,W5,#2         str W7,[X0]
 str   W5,[X0]          txend
 stlr  WZR,[X1]
```

LOCK ELISION

```
 Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
 mov   W3,#1
 stxr  W4,W3,[X1]
 cbnz  W4,Loop
✓ldr   W5,[X0]
 add   W5,W5,#2
 str   W5,[X0]
 stlr  WZR,[X1]
```

```
✓txbegin
 ldr W6,[X1]
 cbz W6,L1
 txabort
 L1:
 mov W7,#1
 str W7,[X0]
 txend
```

LOCK ELISION

```
 Loop:
✓ldaxr W2,[X1]       ✓txbegin
✓cbnz  W2,Loop       ✓ldr W6,[X1]
 mov   W3,#1          cbz W6,L1
 stxr  W4,W3,[X1]     txabort
 cbnz  W4,Loop        L1:
✓ldr   W5,[X0]        mov W7,#1
 add   W5,W5,#2       str W7,[X0]
 str   W5,[X0]        txend
 stlr  WZR,[X1]
```

LOCK ELISION

```
 Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
 mov   W3,#1
 stxr  W4,W3,[X1]
 cbnz  W4,Loop
✓ldr   W5,[X0]
 add   W5,W5,#2
 str   W5,[X0]
 stlr  WZR,[X1]
```

```
✓txbegin
✓ldr W6,[X1]
✓cbz W6,L1
 txabort
 L1:
 mov W7,#1
 str W7,[X0]
 txend
```

LOCK ELISION

```
  Loop:
✓ ldaxr W2,[X1]
✓ cbnz  W2,Loop
  mov   W3,#1
  stxr  W4,W3,[X1]
  cbnz  W4,Loop
✓ ldr   W5,[X0]
  add   W5,W5,#2
  str   W5,[X0]
  stlr  WZR,[X1]
```

```
✓ txbegin
✓ ldr W6,[X1]
✓ cbz W6,L1
✓ txabort
  L1:
  mov W7,#1
  str W7,[X0]
  txend
```

LOCK ELISION

```
  Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
 mov   W3,#1
 stxr  W4,W3,[X1]
 cbnz  W4,Loop
✓ldr   W5,[X0]
 add   W5,W5,#2
 str   W5,[X0]
 stlr  WZR,[X1]
```

```
✓txbegin
✓ldr W6,[X1]
✓cbz W6,L1
✓txabort
✓L1:
 mov W7,#1
 str W7,[X0]
 txend
```

LOCK ELISION

```
  Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
 mov   W3,#1
 stxr  W4,W3,[X1]
 cbnz  W4,Loop
✓ldr   W5,[X0]
 add   W5,W5,#2
 str   W5,[X0]
 stlr  WZR,[X1]
```

```
✓txbegin
✓ldr W6,[X1]
✓cbz W6,L1
✓txabort
✓L1:
✓mov W7,#1
 str W7,[X0]
 txend
```

LOCK ELISION

```
 Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
 mov   W3,#1
 stxr  W4,W3,[X1]
 cbnz  W4,Loop
✓ldr   W5,[X0]
 add   W5,W5,#2
 str   W5,[X0]
 stlr  WZR,[X1]
```

```
✓txbegin
✓ldr W6,[X1]
✓cbz W6,L1
✓txabort
✓L1:
✓mov W7,#1
✓str W7,[X0]
 txend
```

LOCK ELISION

```
 Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
 mov   W3,#1
 stxr  W4,W3,[X1]
 cbnz  W4,Loop
✓ldr   W5,[X0]
 add   W5,W5,#2
 str   W5,[X0]
 stlr  WZR,[X1]
```

```
✓txbegin
✓ldr W6,[X1]
✓cbz W6,L1
✓txabort
✓L1:
✓mov W7,#1
✓str W7,[X0]
✓txend
```

LOCK ELISION

```
 Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
✓mov   W3,#1
 stxr  W4,W3,[X1]
 cbnz  W4,Loop
✓ldr   W5,[X0]
 add   W5,W5,#2
 str   W5,[X0]
 stlr  WZR,[X1]
```

```
✓txbegin
✓ldr W6,[X1]
✓cbz W6,L1
✓txabort
✓L1:
✓mov W7,#1
✓str W7,[X0]
✓txend
```

LOCK ELISION

```
  Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
✓mov   W3,#1
✓stxr  W4,W3,[X1]
 cbnz  W4,Loop
✓ldr   W5,[X0]
 add   W5,W5,#2
 str   W5,[X0]
 stlr  WZR,[X1]
```

```
✓txbegin
✓ldr W6,[X1]
✓cbz W6,L1
✓txabort
✓L1:
✓mov W7,#1
✓str W7,[X0]
✓txend
```

LOCK ELISION

```
  Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
✓mov   W3,#1
✓stxr  W4,W3,[X1]
✓cbnz  W4,Loop
✓ldr   W5,[X0]
 add   W5,W5,#2
 str   W5,[X0]
 stlr  WZR,[X1]
```

```
✓txbegin
✓ldr W6,[X1]
✓cbz W6,L1
✓txabort
✓L1:
✓mov W7,#1
✓str W7,[X0]
✓txend
```

LOCK ELISION

```
 Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
✓mov   W3,#1
✓stxr  W4,W3,[X1]
✓cbnz  W4,Loop
✓ldr   W5,[X0]
✓add   W5,W5,#2
 str   W5,[X0]
 stlr  WZR,[X1]
```

```
✓txbegin
✓ldr W6,[X1]
✓cbz W6,L1
✓txabort
✓L1:
✓mov W7,#1
✓str W7,[X0]
✓txend
```

LOCK ELISION

```
 Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
✓mov   W3,#1
✓stxr  W4,W3,[X1]
✓cbnz  W4,Loop
✓ldr   W5,[X0]
✓add   W5,W5,#2
✓str   W5,[X0]
 stlr  WZR,[X1]
```

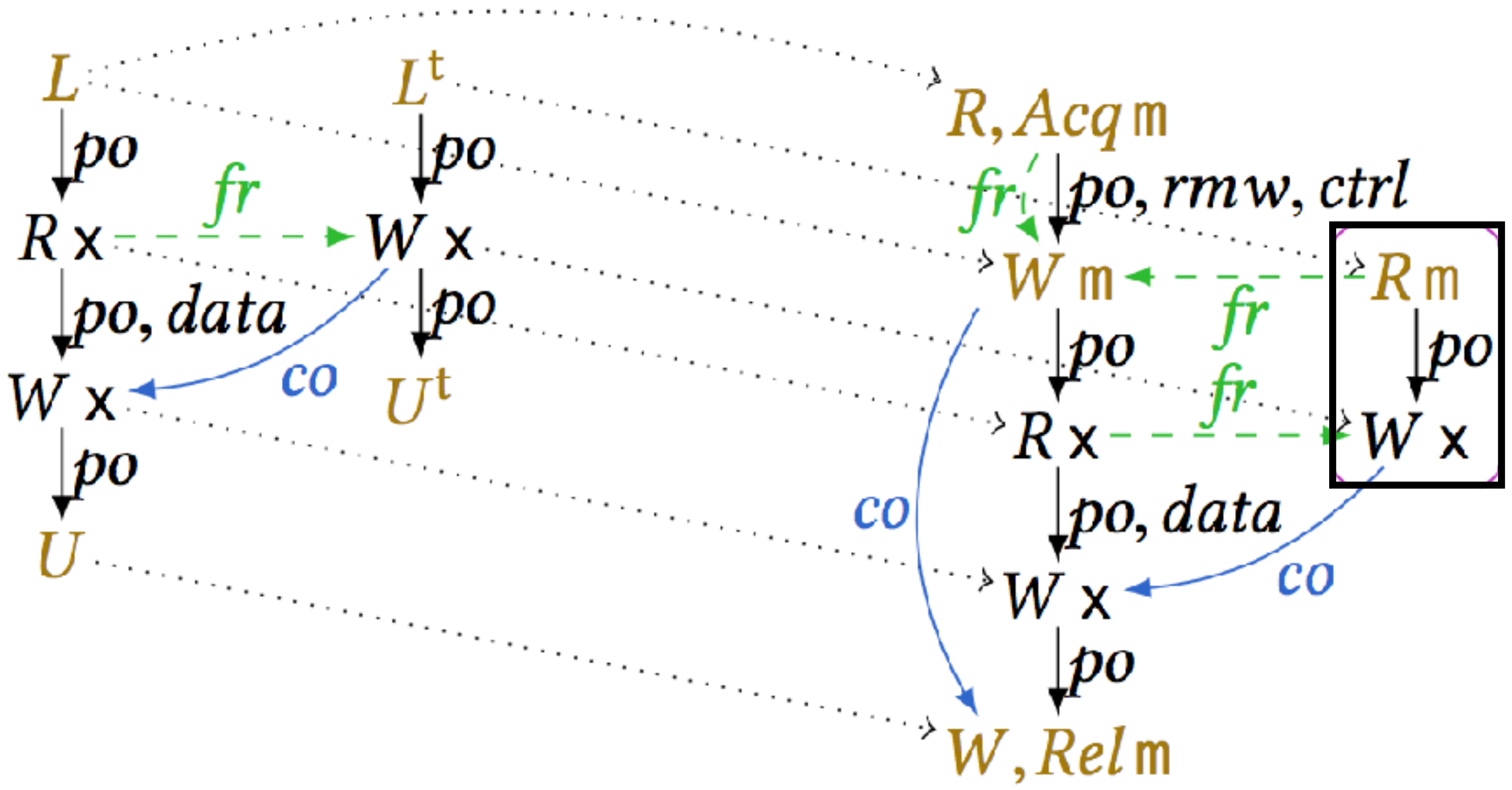
```
✓txbegin
✓ldr W6,[X1]
✓cbz W6,L1
✓txabort
✓L1:
✓mov W7,#1
✓str W7,[X0]
✓txend
```

LOCK ELISION

```
 Loop:
✓ldaxr W2,[X1]
✓cbnz  W2,Loop
✓mov   W3,#1
✓stxr  W4,W3,[X1]
✓cbnz  W4,Loop
✓ldr   W5,[X0]
✓add   W5,W5,#2
✓str   W5,[X0]
✓stlr  WZR,[X1]
```

```
✓txbegin
✓ldr W6,[X1]
✓cbz W6,L1
✓txabort
✓L1:
✓mov W7,#1
✓str W7,[X0]
✓txend
```

LOCK ELISION



OUTLINE

- ~~Weak memory~~
- ~~Transactions~~
- ~~Weak memory and transactions~~
- ~~Validating our models~~
- ~~The problem with lock elision~~
- Related and future work

RELATED WORK

- Dongol, Jagadeesan, and Riely (POPL '18):

  👎 atomicity only

  👎 not empirically validated

  👍 handle aborted transactions

  👎 establish metatheory

FUTURE DIRECTIONS

FUTURE DIRECTIONS

- Validate our model of C++ transactions

FUTURE DIRECTIONS

- Validate our model of C++ transactions
- Account for aborted/failed transactions

FUTURE DIRECTIONS

- Validate our model of C++ transactions
- Account for aborted/failed transactions
- Extend Power model to handle "rollback-only" transactions, and **`tsuspend`** and **`tresume`** instructions

FUTURE DIRECTIONS

- Validate our model of C++ transactions
- Account for aborted/failed transactions
- Extend Power model to handle "rollback-only" transactions, and **`tsuspend`** and **`tresume`** instructions
- Operational models

FUTURE DIRECTIONS

- Validate our model of C++ transactions
- Account for aborted/failed transactions
- Extend Power model to handle "rollback-only" transactions, and **`tsuspend`** and **`tresume`** instructions
- Operational models
- Verify code that implements or uses TM

THE SEMANTICS OF TRANSACTIONS AND WEAK MEMORY IN X86, POWER, ARM, AND C++

Nathan Chong
Arm Ltd.

Tyler Sorensen
Imperial

John Wickerson
Imperial

UCL PPLV Seminar, Thursday 10 May 2018