# Explaining relaxed memory models with program transformations

Ori Lahav and Viktor Vafeiadis

Max Planck Institute for Software Systems (MPI-SWS)

FM, November 2016

# Relaxed memory models

- Sequential consistency (*aka "interleaving semantics"*) is the standard memory model for reasoning about concurrent programs.

- Modern hardware employs, e.g., local write buffers, hierarchies of caches, and speculative executions, that significantly improve performance, but invalidate SC in the presence of data races.

- To further improve performance, compilers perform concurrency-oblivious optimizations.

Relaxed memory models provide formal sound semantics for realistic high-performance concurrency.

# Litmus tests

## Store buffering (SB)

$$\begin{array}{l|l} x := 1; & y := 1; \\ a := y; \ // \, 0 & b := x; \ // \, 0 \end{array}$$

Allowed by x86-TSO, Power, ARM, C11 with non-SC accesses

## Load buffering (LB)

$$\begin{array}{l|l} a := x; \ // \, 1 & b := y; \ // \, 1 \\ y := 1; & x := 1; \end{array}$$

Allowed by Power, ARM, C11 with relaxed accesses

# Litmus tests

## Store buffering (SB)

$$\begin{array}{l|l} x := 1; & y := 1; \\ a := y; \ \#\, 0 & b := x; \ \#\, 0 \end{array}$$

Allowed by x86-TSO, Power, ARM, C11 with non-SC accesses

## Load buffering (LB)

$$\begin{array}{l|l} a := x; \ \#\, 1 & b := y; \ \#\, 1 \\ y := 1; & x := 1; \end{array}$$

Allowed by Power, ARM, C11 with relaxed accesses

Program transformations provide intuitive explanations.

# Litmus tests

## Store buffering (SB)

$$\begin{array}{l|l} x := 1; & y := 1; \\ a := y; \ \#\, 0 & b := x; \ \#\, 0 \end{array}$$

Allowed by x86-TSO, Power, ARM, C11 with non-SC accesses

## Load buffering (LB)

$$\begin{array}{l|l} a := x; \ \#\, 1 & b := y; \ \#\, 1 \\ y := 1; & x := 1; \end{array}$$

Allowed by Power, ARM, C11 with relaxed accesses

Program transformations provide intuitive explanations.

## Our goal

Formally reconcile relaxed memory models definitions with the transformations account.

# Litmus tests

## Store buffering (SB)

$$\begin{array}{l|l} x := 1; & y := 1; \\ a := y; \ /\!/ 0 & b := x; \ /\!/ 0 \end{array}$$

Allowed by x86-TSO, Power, ARM, C11 with non-SC accesses

## Load buffering (LB)

$$\begin{array}{l|l} a := x; \ /\!/ 1 & b := y; \ /\!/ 1 \\ y := 1; & x := 1; \end{array}$$

Allowed by Power, ARM, C11 with relaxed accesses

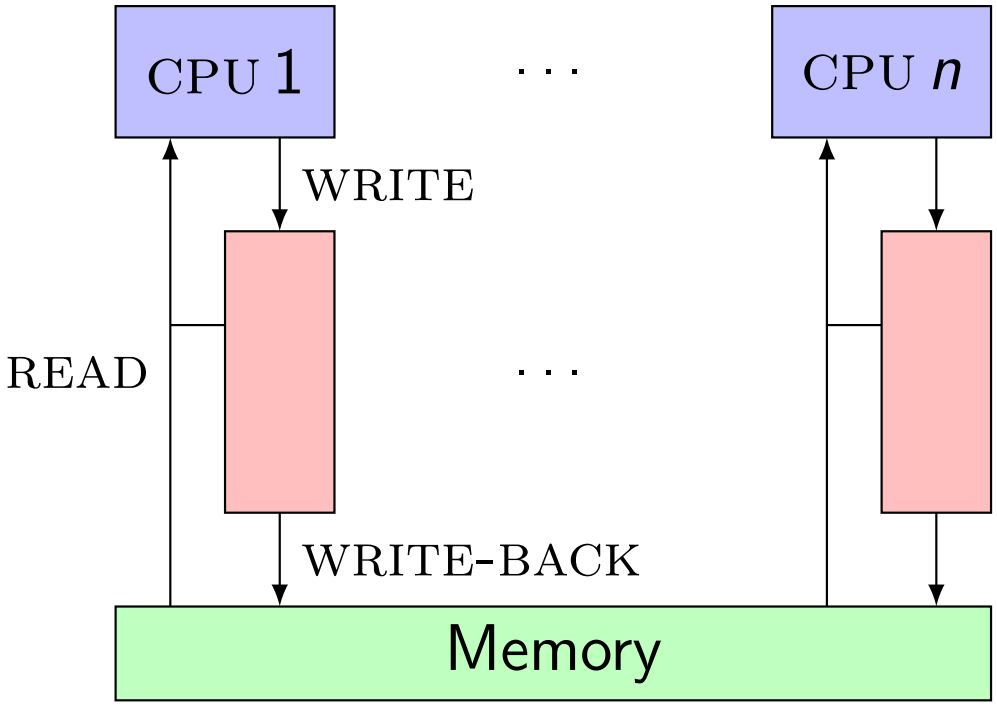
Program transformations provide intuitive explanations.

## Our goal

Formally reconcile relaxed memory models definitions with the transformations account.

$$\text{TSO} \succ \text{C11-release/acquire} \succ \text{Power} \succ \text{ARM}$$

# Operational account for store buffering in x86-TSO




**Store buffering**

| | | |
|---|---|---|
| $x := 1;$ | $\parallel$ | $y := 1;$ |
| $a := y;$ // 0 | $\parallel$ | $b := x;$ // 0 |

**Store buffering + fences**

| | | |
|---|---|---|
| $x := 1;$ | $\parallel$ | $y := 1;$ |
| `fence;` | $\parallel$ | `fence;` |
| $a := y;$ // 0 | $\parallel$ | $b := x;$ // 0 |

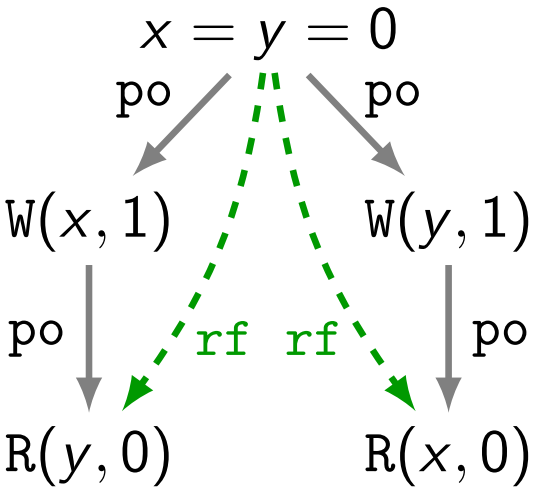
# SB and LB in axiomatic models

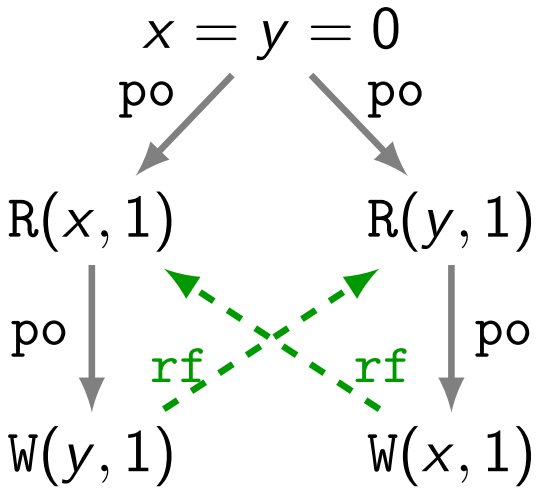
**Store buffering (SB)**

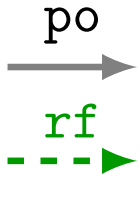
$x := 1;$ $a := y;$ // 0 $\quad\|\quad$ $y := 1;$ $b := x;$ // 0

**Load buffering (LB)**

$a := x;$ // 1 $y := 1;$ $\quad\|\quad$ $b := y;$ // 1 $x := 1;$




program order

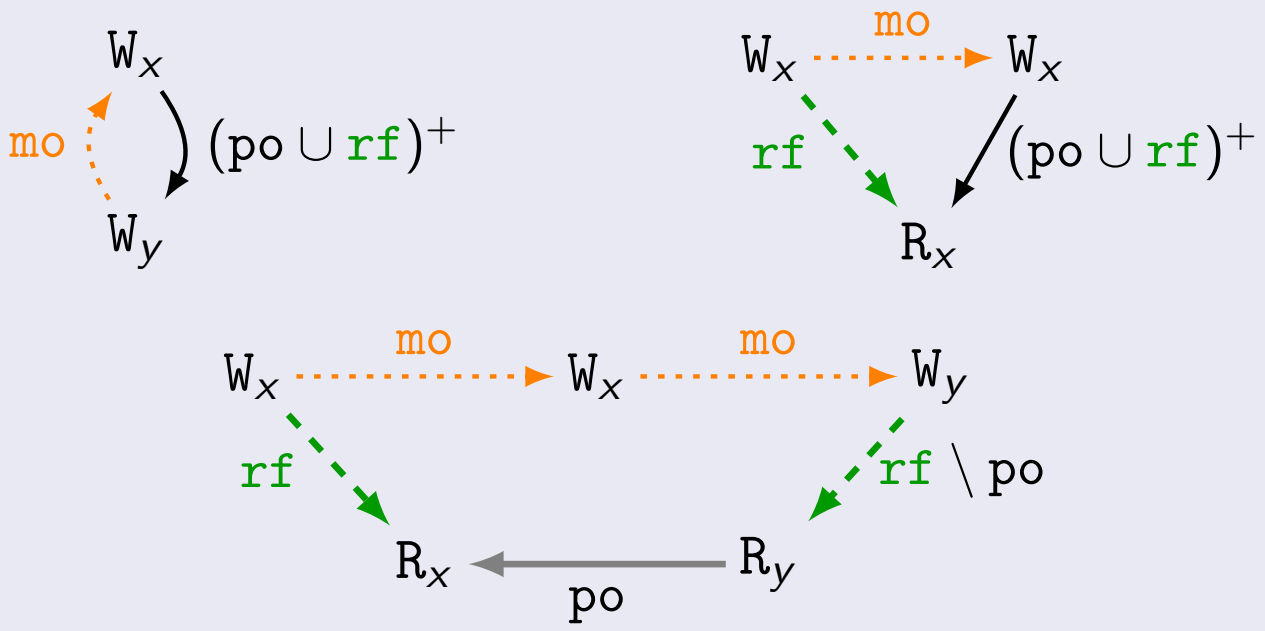
reads-from

# Axiomatic x86-TSO model

## Definition

An execution is TSO-*consistent* if:

- $\mathrm{po} \cup \mathtt{rf}$ is acyclic
- there exists a total ordering $\mathtt{mo}$ of all write events, such that the none of the following occurs:

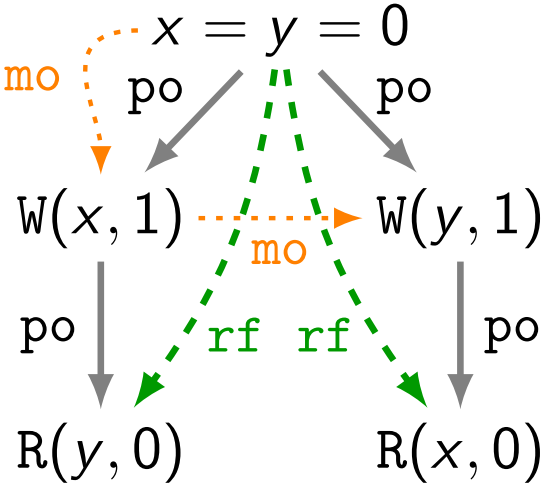



+ conditions on fences and RMWs

# SB and LB in TSO

## Store buffering (SB)

$x := 1;$ $a := y;$ // 0 $\quad\|\quad$ $y := 1;$ $b := x;$ // 0

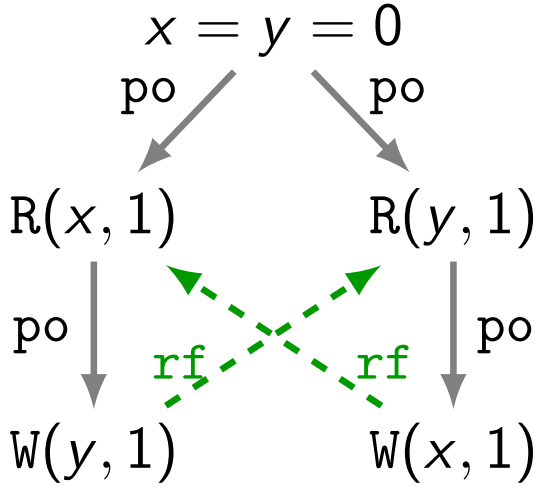



**allowed by TSO**

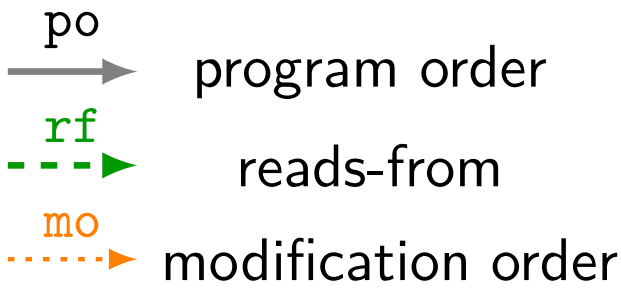
## Load buffering (LB)

$a := x;$ // 1 $\quad\|\quad$ $y := 1;$

$b := y;$ // 1 $\quad\|\quad$ $x := 1;$




**Forbidden by TSO**



# Sound optimizations under TSO

## Sound Transformations

$$P_{\text{source}} \rightsquigarrow P_{\text{target}} \implies [\![P_{\text{target}}]\!] \subseteq [\![P_{\text{source}}]\!]$$

## Write-read reordering

$$\begin{array}{l} x := 1; \\ a := y; \end{array} \rightsquigarrow \begin{array}{l} a := y; \\ x := 1; \end{array}$$

## Read-after-write elimination

$$\begin{array}{l} x := 1; \\ a := x; \end{array} \rightsquigarrow \begin{array}{l} x := 1; \\ a := 1; \end{array}$$

# Sound optimizations under TSO

## Sound Transformations

$$P_{\text{source}} \rightsquigarrow P_{\text{target}} \implies [\![P_{\text{target}}]\!] \subseteq [\![P_{\text{source}}]\!]$$

## Write-read reordering

$$\begin{array}{l} x := 1; \\ a := y; \end{array} \rightsquigarrow \begin{array}{l} a := y; \\ x := 1; \end{array}$$

## Read-after-write elimination

$$\begin{array}{l} x := 1; \\ a := x; \end{array} \rightsquigarrow \begin{array}{l} x := 1; \\ a := 1; \end{array}$$

## Definition

$G \rightsquigarrow_{\text{TSO}} G'$ if $G'$ is obtained from $G$ by one of the following:

$$\begin{array}{c} \mathtt{W}(x, v_x) \\ \text{po}\downarrow \\ \mathtt{R}(y, v_y) \end{array} \rightsquigarrow \begin{array}{c} \mathtt{R}(y, v_y) \\ \text{po}\downarrow \\ \mathtt{W}(x, v_x) \end{array} \qquad \begin{array}{c} \mathtt{W}(x, v) \\ \text{po}\downarrow \\ \mathtt{R}(x, v) \end{array} \rightsquigarrow \mathtt{W}(x, v)$$

## Theorem (Soundness of transformations)

*If $G \rightsquigarrow_{\text{TSO}} G'$ and $G'$ is TSO-consistent, so is $G$.*

# Alternative TSO characterisation

$$\text{TSO} = \text{SC} + \text{WR-reordering} + \text{RaW-elimination}$$

**Theorem**

*$G$ is TSO-consistent iff*
*there exists $G'$ such that $G \rightsquigarrow^*_{\text{TSO}} G'$ and $G'$ is SC-consistent.*

($\Leftarrow$) By soundness of transformations.

($\Rightarrow$) Assume that $G$ is TSO-consistent and not SC-consistent.
Show that some transformation is applicable and that it preserves TSO-consistency.

Induction metrics: $|\mathtt{po} \cap (\mathtt{W} \times \mathtt{R})|$

# Application: compilation correctness

**Compilation Correctness**

$$[\![compile(P)]\!]_{\text{target memory model}} \quad \subseteq \quad [\![P]\!]_{\text{source memory model}}$$

$$\text{TSO} = \text{SC} + \text{WR-reordering} + \text{RaW-elimination}$$

To prove $[\![compile(P)]\!]_{TSO} \quad \subseteq \quad [\![P]\!]_{C11}$, it remains to show:

- Compilation is correct for SC.
- WR-reorderings and RaW-eliminations correspond to C11-sound transformations in the source program.

# Explaining memory models with program transformations

TSO = SC + WR-reordering + RaW-elimination

C11 release/acquire = ?

Power = ?

ARM = ?

# C11 release/acquire

## Independent reads of independent writes (IRIW)

$$\begin{array}{l} a := x;\ \#1 \\ b := y;\ \#0 \end{array} \Big\| \; x := 1; \Big\| \; y := 1; \Big\| \begin{array}{l} c := y;\ \#1 \\ d := x;\ \#0 \end{array}$$

- This behavior is allowed by C11.
- No sound thread-local transformation can be applied.
- Sequentialization, $C_1 \parallel C_2 \rightsquigarrow C_1; C_2$, is applicable, and then the outcome is possible.

$$\begin{array}{l} a := x;\ \#1 \\ b := y;\ \#0 \end{array} \Big\| \; x := 1; \quad \rightsquigarrow \quad \begin{array}{l} x := 1; \\ a := x; \\ b := y; \end{array} \quad \rightsquigarrow \quad \begin{array}{l} x := 1; \\ a := 1; \\ b := y; \end{array} \quad \rightsquigarrow \quad \begin{array}{l} b := y; \\ x := 1; \\ a := 1; \end{array}$$

# C11 release/acquire counterexample

$$\text{RA} \prec \text{SC} + \text{WR-reordering} + \text{RaW-elimination} + \text{SEQ}$$

$$\begin{array}{l} y := 1; \\ x := 1; \\ a := x; \ // \, 3 \\ b := z; \ // \, 0 \end{array} \Bigg\| \; x := 3; \; \Bigg\| \begin{array}{l} z := 1; \\ x := 2; \\ c := x; \ // \, 3 \\ d := y; \ // \, 0 \end{array}$$

- This behavior is allowed by C11.
- No local transformations are possible.
- Sequentialisation rules out the outcome.

# POWER multiprocessor

- weaker than C11 release/acquire

$$\text{Power} \prec \text{SC} + \text{reorderings} + \text{eliminations} + \text{SEQ}$$

# Power axiomatic model

[Alglave et al.'14]

An execution $G$ is *Power-consistent* if the following hold:

1. $\texttt{mo}$ is a disjoint union of relations $\{\texttt{mo}_x\}_{x \in \text{Loc}}$, such that each relation $\texttt{mo}_x$ is a strict total order on $\texttt{W}_x\texttt{U}_x$.
2. $\texttt{hb}$ is acyclic. (*no-thin-air*)
3. $\texttt{po}|_x \cup \texttt{rf} \cup \texttt{fr} \cup \texttt{mo}$ is acyclic for every $x \in \text{Loc}$. (*SC-per-loc*)
4. $\texttt{fre}; \texttt{prop}; \texttt{hb}^*$ is irreflexive. (*observation*)
5. $\texttt{mo} \cup \texttt{prop}$ is acyclic. (*propagation*)
6. $\texttt{fr}; \texttt{mo}$ is irreflexive. (*atomicity*)
7. $\texttt{mo}; [\texttt{U}]; \texttt{po}; [\texttt{U}]$ is acyclic.

where:

- $\texttt{sync} = \texttt{po}; [\texttt{F}_{\texttt{sync}}]; \texttt{po}$ and $\texttt{lwsync} = \texttt{po}; [\texttt{F}_{\texttt{lwsync}}]; \texttt{po}$
- $\texttt{fence} = \texttt{sync} \cup ([\texttt{RU}]; \texttt{lwsync}; [\texttt{RWU}] \cup ([\texttt{W}]; \texttt{lwsync}; [\texttt{WU}]))$ (*fence order*)
- $\texttt{fr} = (\texttt{rf}^{-1}; \texttt{mo}) \setminus [\texttt{E}]$ (*read before*)
- $\texttt{rfe} = \texttt{rf} \setminus \texttt{po}$ and $\texttt{fre} = \texttt{fr} \setminus \texttt{po}$ (*external relations*)
- $\texttt{ppo} = \ldots$ (*preserved program order*)
- $\texttt{hb} = \texttt{ppo} \cup \texttt{fence} \cup \texttt{rfe}$ (*happens-before*)
- $\texttt{prop}_1 = [\texttt{WU}]; \texttt{rfe}^?; \texttt{fence}; \texttt{hb}^*; [\texttt{WU}]$
- $\texttt{prop}_2 = ((\texttt{mo} \cup \texttt{fr}) \setminus \texttt{po})^?; \texttt{rfe}^?; (\texttt{fence}; \texttt{hb}^*)^?; \texttt{sync}; \texttt{hb}^*$
- $\texttt{prop} = \texttt{prop}_1 \cup \texttt{prop}_2$ (*propagation relation*)

# Power axiomatic model

[Alglave et al.'14]

An execution $G$ is *Power-consistent* if the following hold:

1. $\texttt{mo}$ is a disjoint union of relations $\{\texttt{mo}_x\}_{x \in \text{Loc}}$, such that each relation $\texttt{mo}_x$ is a strict total order on $\texttt{W}_x\texttt{U}_x$.
2. $\texttt{hb}$ is acyclic. (*no-thin-air*)
3. $\texttt{po}|_x \cup \texttt{rf} \cup \texttt{fr} \cup \texttt{mo}$ is acyclic for every $x \in \text{Loc}$. (*SC-per-loc*)
4. $\texttt{fre}; \texttt{prop}; \texttt{hb}^*$ is irreflexive. (*observation*)
5. $\texttt{mo} \cup \texttt{prop}$ is acyclic. (*propagation*)
6. $\texttt{fr}; \texttt{mo}$ is irreflexive. (*atomicity*)

The model allows cycles in $\texttt{po} \cup \texttt{rf}$!

- The definition is not *prefix-closed*.
- *Speculation* is required to operationally construct executions.

- $\texttt{ppo} = \ldots$ (*preserved program order*)
- $\texttt{hb} = \texttt{ppo} \cup \texttt{fence} \cup \texttt{rfe}$ (*happens-before*)
- $\texttt{prop}_1 = [\texttt{WU}]; \texttt{rfe}^?; \texttt{fence}; \texttt{hb}^*; [\texttt{WU}]$
- $\texttt{prop}_2 = ((\texttt{mo} \cup \texttt{fr}) \setminus \texttt{po})^?; \texttt{rfe}^?; (\texttt{fence}; \texttt{hb}^*)^?; \texttt{sync}; \texttt{hb}^*$
- $\texttt{prop} = \texttt{prop}_1 \cup \texttt{prop}_2$ (*propagation relation*)

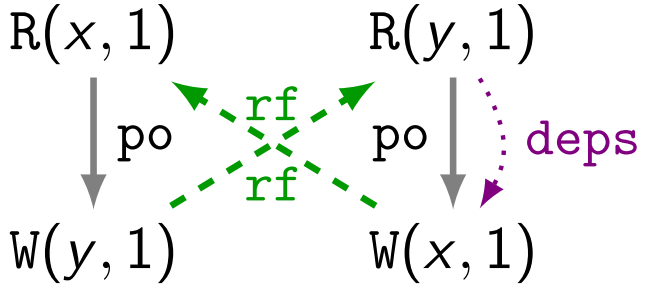
# Load buffering

The model allows cycles in $\mathtt{po} \cup \mathtt{rf}$!

- The definition is not *prefix-closed*.
- *Speculation* is required to operationally construct executions.

**Load buffering (LB)**

$$\begin{array}{l} a := x; \ /\!/ 1 \\ y := 1; \end{array} \quad \Big\| \quad x := y;$$




**allowed by** Power

**Definition (Strong Power model)**

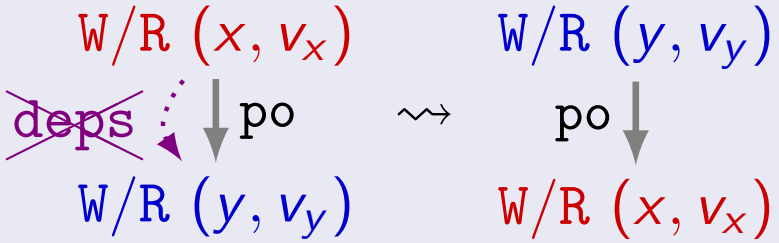
An execution is StrongPower-*consistent* if it is Power-consistent and $\mathtt{po} \cup \mathtt{rf}$ is acyclic.

# Reduction to StrongPower-consistency

$$\text{Power} = \text{StrongPower} + \text{reorderings}$$

## Definition

$G \rightsquigarrow_{\text{Power}} G'$ if $G'$ is obtained from $G$ by reordering two independent adjacent memory accesses to different locations:




## Theorem

*An execution $G$ is* Power*-consistent iff $G \rightsquigarrow^*_{\text{Power}} G'$ for some* StrongPower*-consistent execution $G'$.*

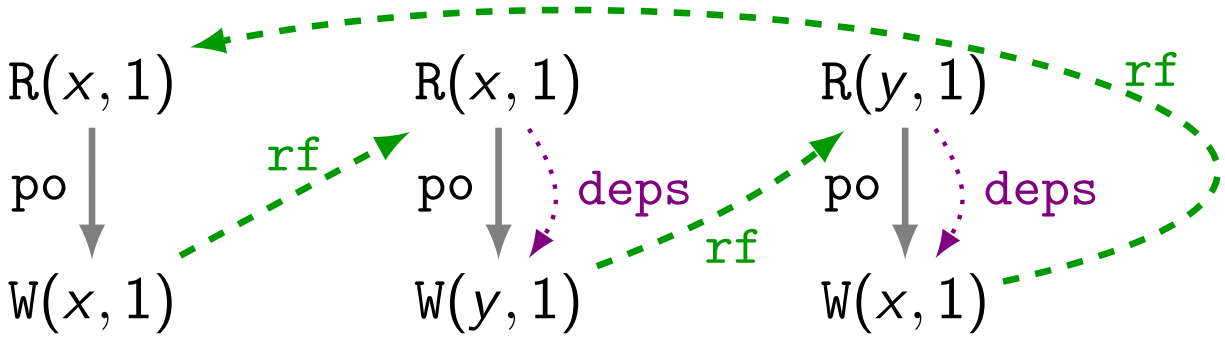
- Limitation: Power's `isync` fences are excluded

# A strange ARM behaviour

$$\text{ARM} \prec \text{StrongARM} + \text{reorderings} + \text{eliminations}$$

**ARM weak**

$$\begin{array}{l} a := x; \ /\!/ 1 \\ x := 1; \end{array} \quad \Big\| \quad y := x; \quad \Big\| \quad x := y;$$




**allowed by ARM**

- No local transformation can be applied.

# Summary

## High-level points

- Some memory models can be defined via transformations.
- But there is more to weak memory than transformations.

## Technical results

- TSO = SC + WR-reordering + RaW-elimination
- RA $\prec$ SC+ WR-reordering + RaW-elimination + SEQ
- Power = StrongPower + reorderings
- ARM $\prec$ StrongARM + reorderings + eliminations

## Application

- Simplify compilation correctness proofs

See `http://plv.mpi-sws.org/trns/` for more details and Coq proofs.

# Summary

## High-level points

- Some memory models can be defined via transformations.
- But there is more to weak memory than transformations.

## Technical results

- TSO $=$ SC $+$ WR-reordering $+$ RaW-elimination
- RA $\prec$ SC$+$ WR-reordering $+$ RaW-elimination $+$ SEQ
- Power $=$ StrongPower $+$ reorderings
- ARM $\prec$ StrongARM $+$ reorderings $+$ eliminations

## Application

- Simplify compilation correctness proofs

See `http://plv.mpi-sws.org/trns/` for more details and Coq proofs.

*Thank you!*