Promising Compilation to ARMv8.3

Anton Podkopaev[1] Ori Lahav[2] Viktor Vafeiadis[3]

[1]St. Petersburg University, JetBrains Research, Russia

[2]Tel Aviv University, Israel

[3]Max Planck Institute for Software Systems, Germany

30.11.2017

Modern Compilers and CPUs are **Optimizing**

Modern Compilers and CPUs are **Optimizing**

Features

- reorderings
- cache
- buffers
- read-after-write elimination
- speculative execution
- fake dependency elimination
- ...

Modern Compilers and CPUs are **Optimizing**

Features

- reorderings
- cache
- buffers
- read-after-write elimination
- speculative execution
- fake dependency elimination
- ...

} Correct

Modern Compilers and CPUs are **Optimizing**

Features

- reorderings
- cache
- buffers
- read-after-write elimination
- speculative execution
- fake dependency elimination
- ...

} Correct for **one** thread

Modern Compilers and CPUs are **Optimizing**

Features

- reorderings
- cache
- buffers
- read-after-write elimination
- speculative execution
- fake dependency elimination
- ...

} Correct for **one** thread

Lead to strange concurrent behaviors

Modern Compilers and CPUs are **Optimizing**

Features

- reorderings
- cache
- buffers
- read-after-write elimination
- speculative execution
- fake dependency elimination
- ...

} Correct for **one** thread

Lead to **weak** concurrent behaviors

Weak Behavior: Load Buffering

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$

Weak Behavior: Load Buffering

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$

[x and y are initialized by 0]

Weak Behavior: Load Buffering

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$

[x and y are initialized by 0]

Final values $a = 1$, $b = 1$

Memory Model (MM) is semantics of concurrent system

Weak memory model allows weak behaviors

Most PLs and CPUs have weak MMs

Most PLs and CPUs have weak MMs, but only **racy** programs have weak behaviors

Most PLs and CPUs have

When to **care** about Weak MMs?

Most PLs and CPUs have

When to **care** about Weak MMs?
Writing/verifing lock-free code
(i.e., locks themselves)

(Weak) Memory Models in the Wild

For hardware

For programming languages

(Weak) Memory Models in the Wild

For hardware

- x86-TSO, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- $\mathrm{ARMv8\ POP}$, [Flur et al., 2016]
- $\mathrm{ARMv8.3}$, [Pulte et al., 2018]

For programming languages

(Weak) Memory Models in the Wild

For hardware

- x86-TSO, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- $\mathrm{ARMv8\ POP}$, [Flur et al., 2016]
- $\mathrm{ARMv8.3}$, [Pulte et al., 2018]

For programming languages

- C/C++11 MM, [Batty et al., 2011]
- Java MM, [Manson et al., 2005]

(Weak) Memory Models in the Wild

For hardware

- x86-TSO, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARMv8 POP, [Flur et al., 2016]
- ARMv8.3, [Pulte et al., 2018]

For programming languages

- C/C++11 MM, [Batty et al., 2011]
- Java MM, [Manson et al., 2005]

Have flaws:
out-of-thin-air values, unsoundness of optimizations, ...

(Weak) Memory Models in the Wild

For hardware

- x86-TSO, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARMv8 POP, [Flur et al., 2016]
- ARMv8.3, [Pulte et al., 2018]

For programming languages

- C/C++11 MM, [Batty et al., 2011]
- Java MM, [Manson et al., 2005]
- Proposed solution [Kang et al., 2017], Promise, for C/C++ and Java

(Weak) Memory Models in the Wild

For hardware

- x86-TSO, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARMv8 POP, [Flur et al., 2016]
- ARMv8.3, [Pulte et al., 2018]

Compilation correctness proved in [Kang et al., 2017] and [Podkopaev et al., 2017]

For programming languages

- C/C++11 MM, [Batty et al., 2011]
- Java MM, [Manson et al., 2005]
- Proposed solution [Kang et al., 2017], Promise, for C/C++ and Java

(Weak) Memory Models in the Wild

For hardware

- x86-TSO, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARMv8 POP, [Flur et al., 2016]
- ARMv8.3, [Pulte et al., 2018]

Compilation correctness proved in [Kang et al., 2017] and [Podkopaev et al., 2017]

Same ideas don't work for ARMv8.3!

For programming languages

- C/C++11 MM, [Batty et al., 2011]
- Java MM, [Manson et al., 2005]
- Proposed solution [Kang et al., 2017], Promise, for C/C++ and Java

(Weak) Memory Models in the Wild

For hardware

- x86-TSO, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARMv8 POP, [Flur et al., 2016]
- ARMv8.3, [Pulte et al., 2018]

For programming languages

- C/C++11 MM, [Batty et al., 2011]
- Java MM, [Manson et al., 2005]
- Proposed solution [Kang et al., 2017], Promise, for C/C++ and Java

Result

Proved compilation correctness from Promise subset to ARMv8.3

Result

Proved compilation correctness from
$\mathrm{Promise}$ subset to $\mathrm{ARMv8.3}$

relaxed loads and stores,
release and acquire fences,
no certification

Load Buffering

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$

[x and y are initialized by 0]

Final values $a = 1$, $b = 1$

Load Buffering in Promise?

Load Buffering in Promise

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$

Memory: $[\langle x : 0@\mathbf{0}_\tau \rangle, \langle y : 0@\mathbf{0}_\tau \rangle]$

Final values $a = \_,\ b = \_$

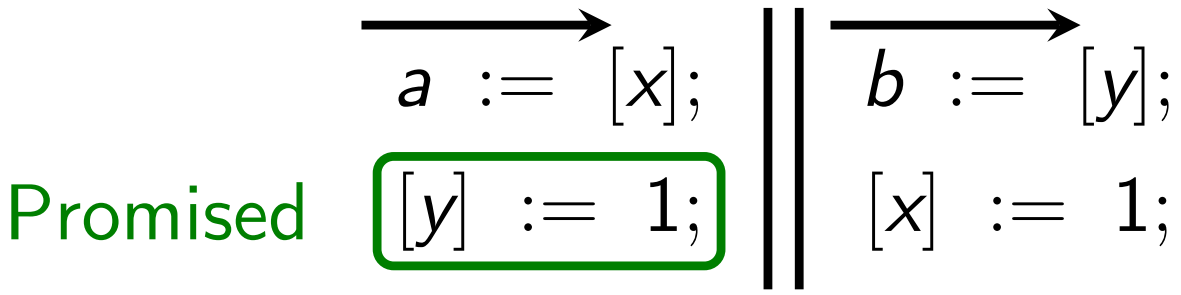
Load Buffering in Promise

$$\overrightarrow{a := [x];} \quad \Big\| \quad \overrightarrow{b := [y];}$$
$$[y] := 1; \quad \Big\| \quad [x] := 1;$$

Memory: $[\langle x : 0@\mathbf{0}_\tau \rangle, \langle y : 0@\mathbf{0}_\tau \rangle]$

Final values $a = \_$, $b = \_$

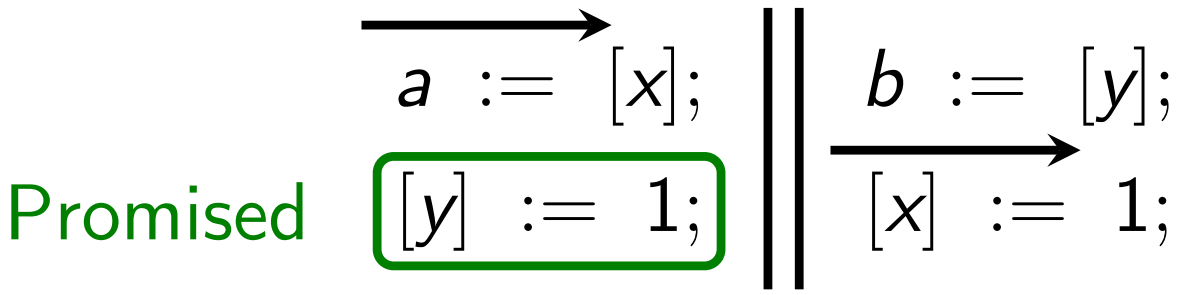
Load Buffering in Promise



Memory: $[\langle x : 0@\mathbf{0}_\tau\rangle, \langle y : 0@\mathbf{0}_\tau\rangle, \langle y : 1@\mathbf{1}_\tau\rangle]$

Final values $a = \_,\ b = \_$

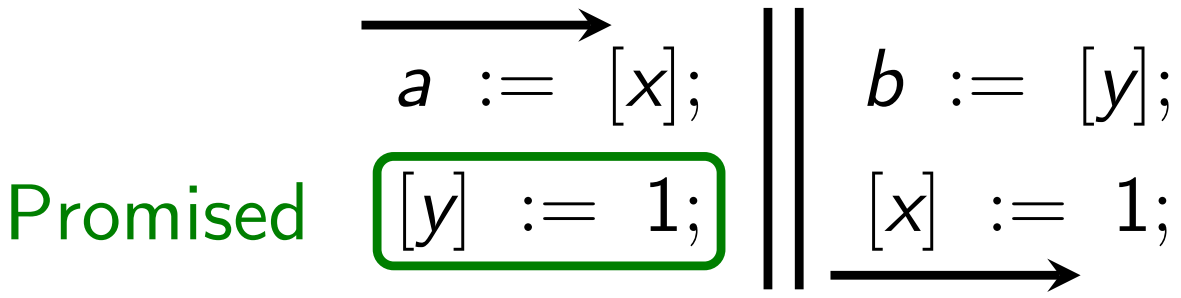
Load Buffering in Promise



Memory: $[\langle x : 0@\mathbf{0}_\tau\rangle, \langle y : 0@\mathbf{0}_\tau\rangle, \langle y : 1@\mathbf{1}_\tau\rangle]$

Final values $a = \_$, $b = 1$

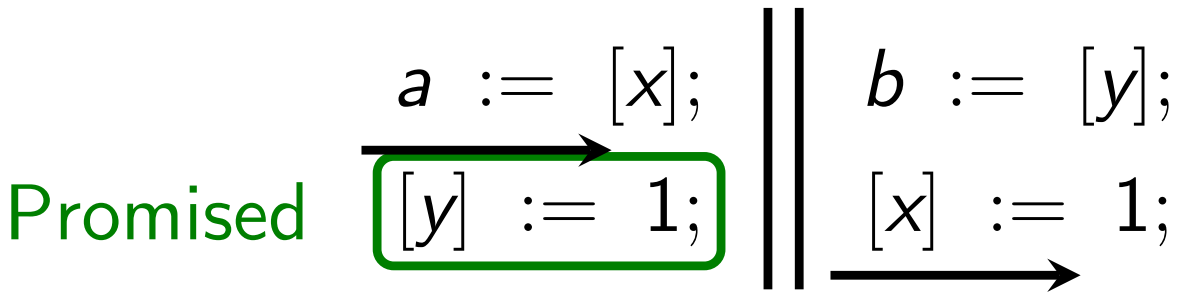
Load Buffering in Promise



Memory: $[\langle x : 0@\mathbf{0}_\tau\rangle, \langle y : 0@\mathbf{0}_\tau\rangle, \langle y : 1@\mathbf{1}_\tau\rangle, \langle x : 1@\mathbf{1}_\tau\rangle]$

Final values $a = \_$, $b = 1$

Load Buffering in Promise



Memory: $[\langle x : 0@\mathbf{0}_\tau\rangle, \langle y : 0@\mathbf{0}_\tau\rangle, \langle y : 1@\mathbf{1}_\tau\rangle, \langle x : 1@\mathbf{1}_\tau\rangle]$

Final values $a = 1$, $b = 1$

Load Buffering in Promise

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \quad \Big\| \quad \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$

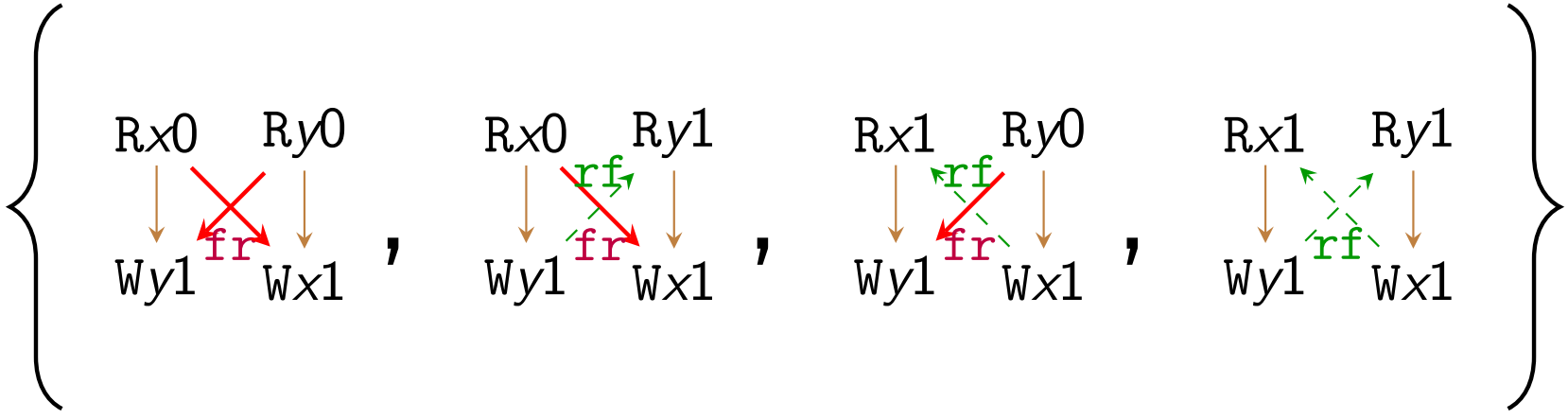
Memory: $[\langle x : 0@\mathbf{0}_\tau\rangle, \langle y : 0@\mathbf{0}_\tau\rangle, \langle y : 1@\mathbf{1}_\tau\rangle, \langle x : 1@\mathbf{1}_\tau\rangle]$

Final values $a = 1$, $b = 1$

Load Buffering in ARMv8.3?

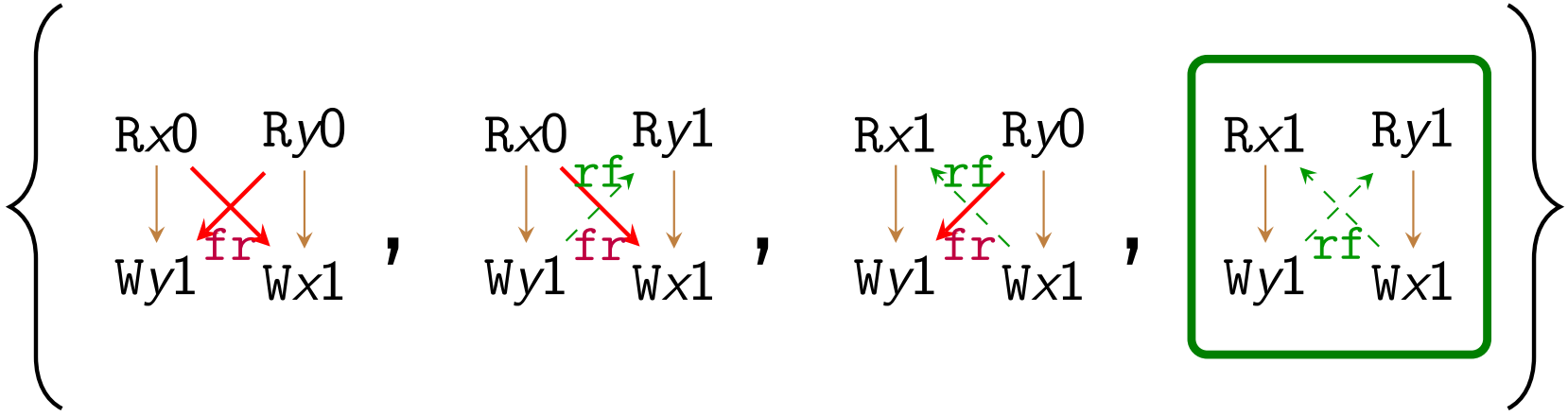
Load Buffering in ARMv8.3

$$\begin{array}{lll} a := [x]; & \Big\| & b := [y]; \\ [y] := 1; & \Big\| & [x] := 1; \end{array}$$



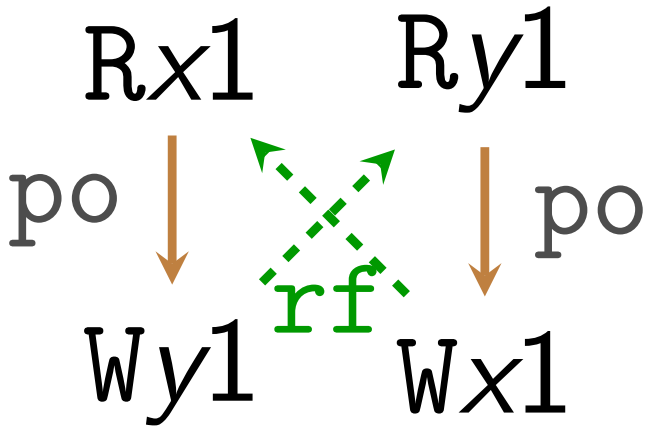
Load Buffering in ARMv8.3

$$a := [x]; \quad \Big\| \quad b := [y];$$
$$[y] := 1; \quad \Big\| \quad [x] := 1;$$



Load Buffering in ARMv8.3

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$



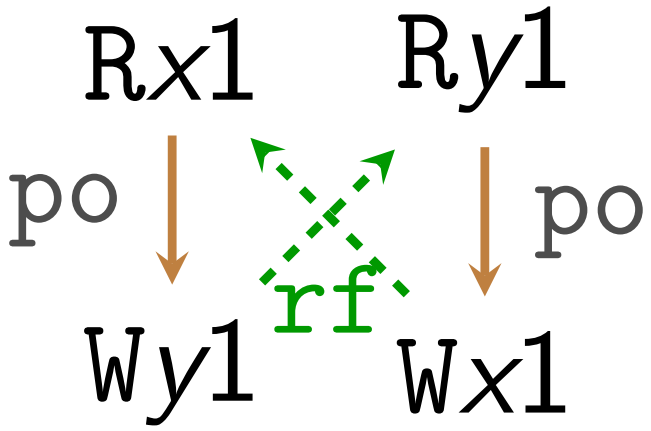
Load Buffering in ARMv8.3

$$\begin{array}{lll} a := [x]; & \Big\| & b := [y]; \\ [y] := 1; & \Big\| & [x] := 1; \end{array}$$

Axioms:

1. $\mathtt{po}\restriction_{\mathtt{loc}} \cup \mathtt{rf}$ is acyclic

...



How to prove correctness of compilation?

How to prove correctness of compilation?

Standard technique:
Simulation

Simulation works for operational semantics

Simulation works for operational semantics

How to **simulate graphs**?

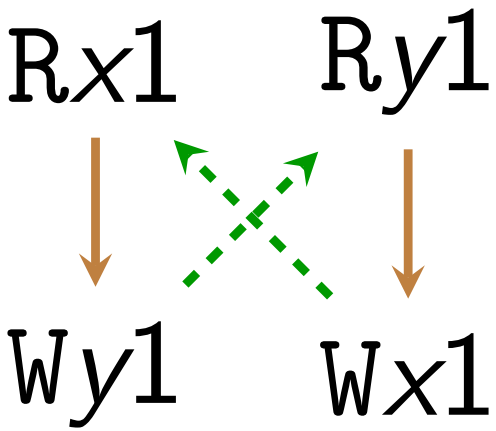
Simulation works for operational semantics

How to **simulate graphs**?

Traverse in proper order!

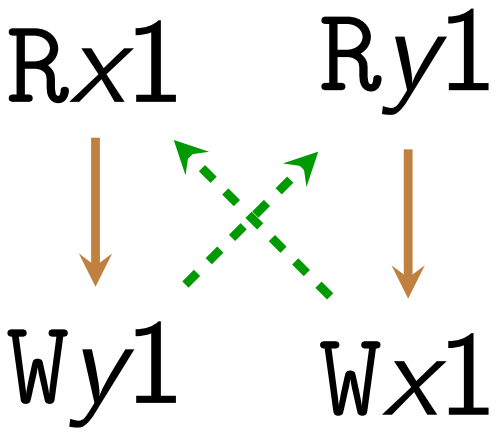
Traverse of ARMv8.3 execution

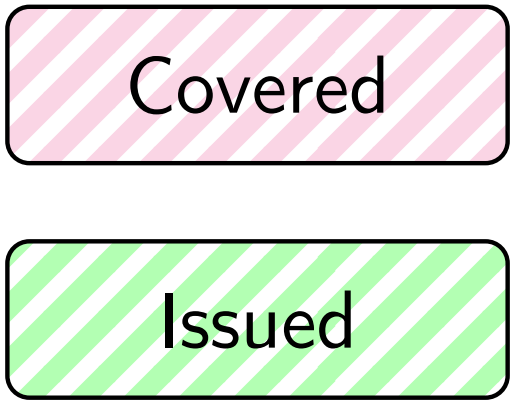
$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$



Traverse of ARMv8.3 execution

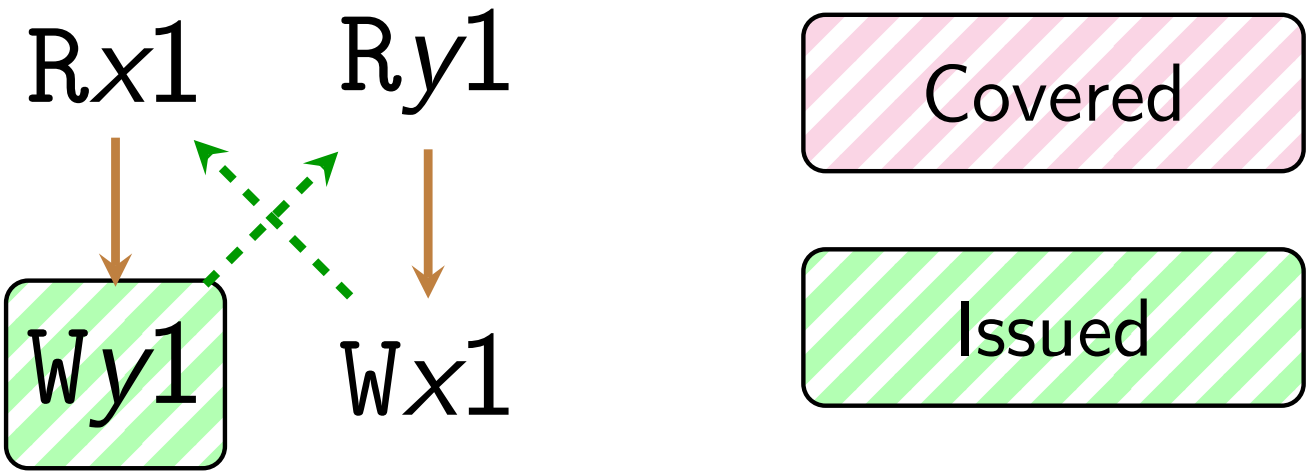
$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$





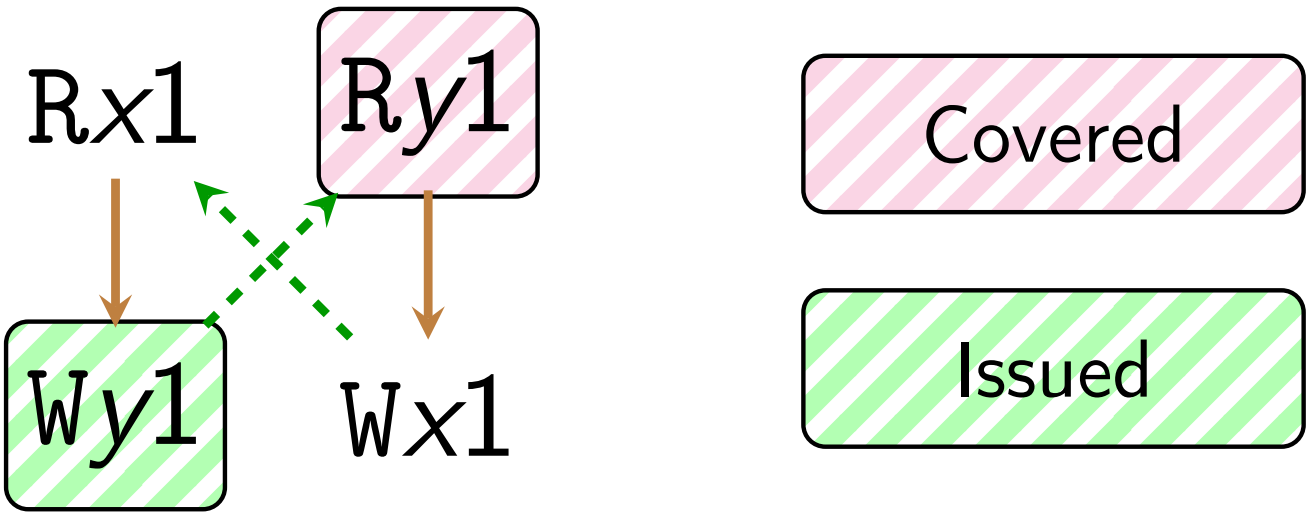
Traverse of ARMv8.3 execution

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$



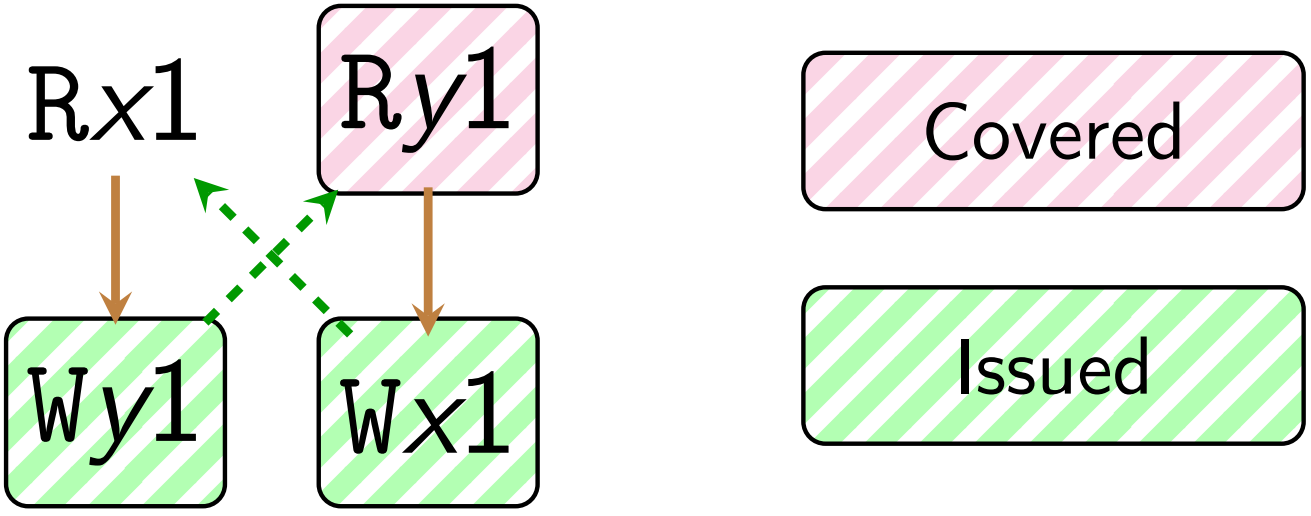
Traverse of ARMv8.3 execution

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$



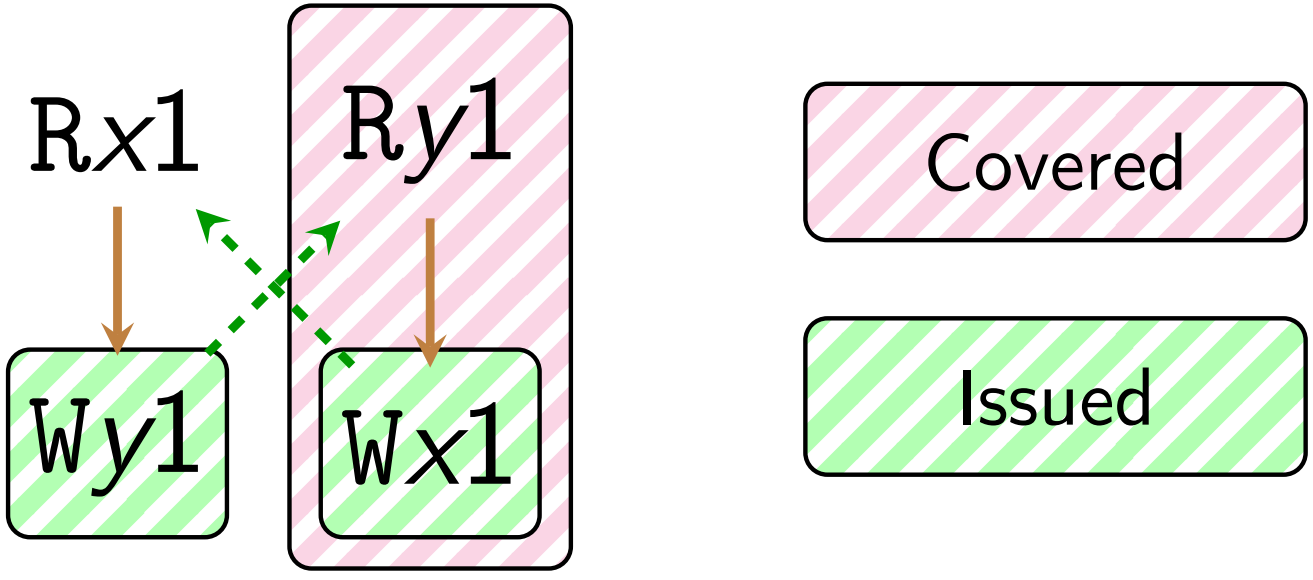
Traverse of ARMv8.3 execution

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$



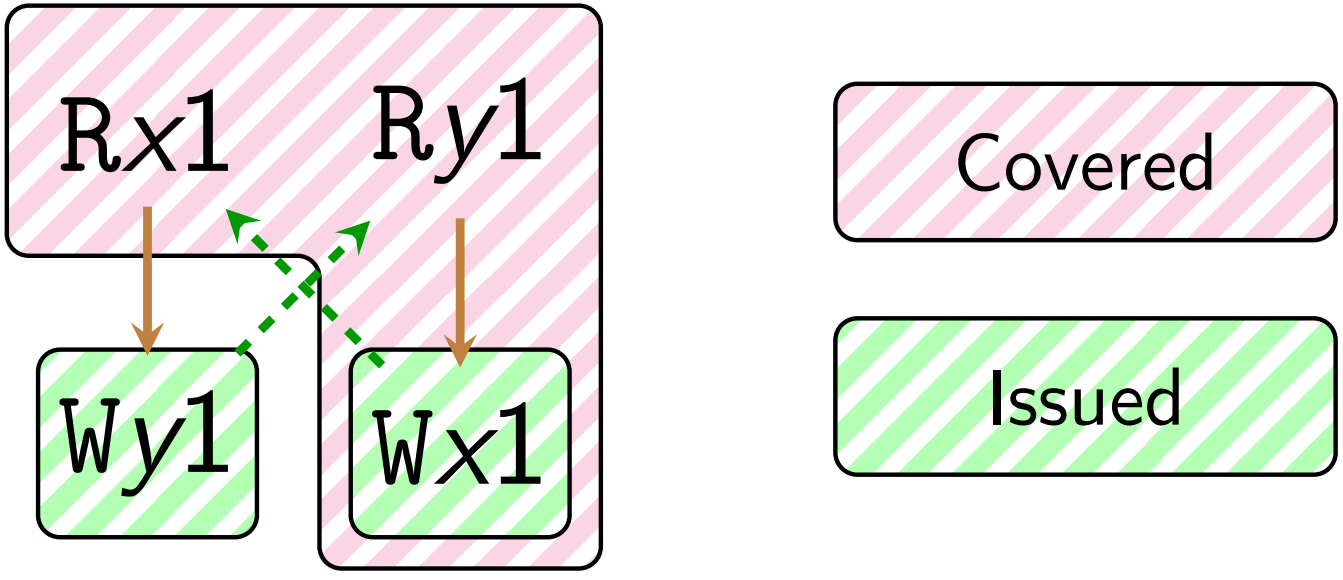
Traverse of ARMv8.3 execution

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$



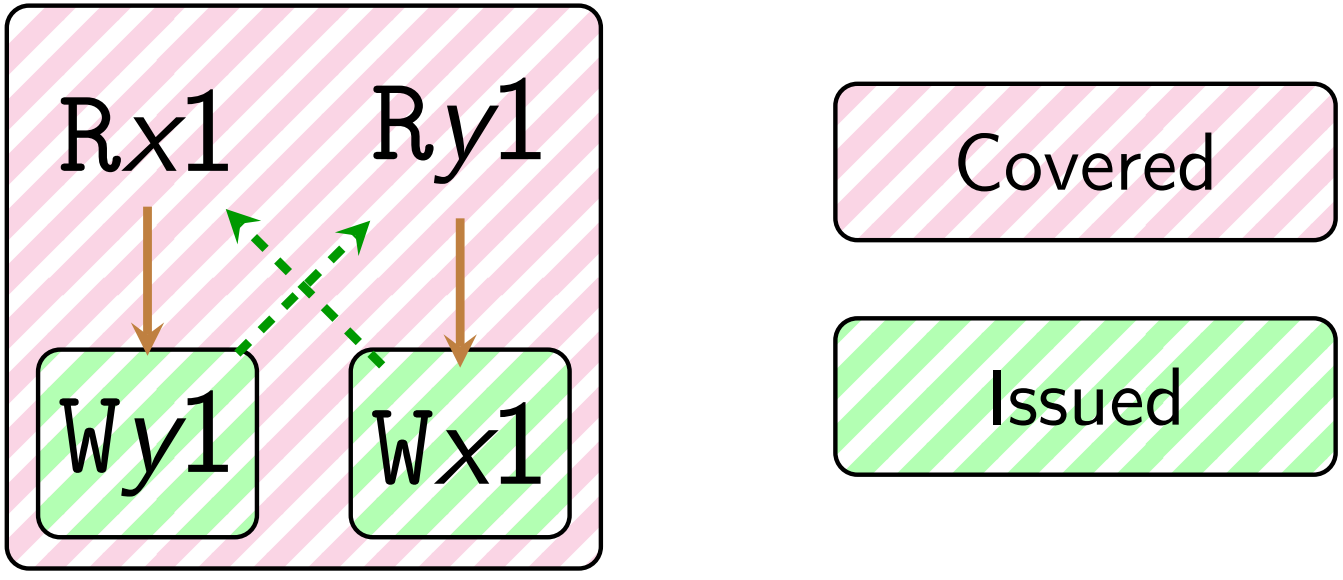
Traverse of ARMv8.3 execution

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$



Traverse of ARMv8.3 execution

$$\begin{array}{l} a := [x]; \\ [y] := 1; \end{array} \;\Big\|\; \begin{array}{l} b := [y]; \\ [x] := 1; \end{array}$$



Traversal formally

Cover step:

$$\frac{\ldots}{G \vdash \langle C, I \rangle \to \langle C \cup \{a\}, I \rangle}$$

Issue step:

$$\frac{\ldots}{G \vdash \langle C, I \rangle \to \langle C, I \cup \{w\} \rangle}$$

Traversal formally

Cover step:

$$\frac{\ldots}{G \vdash \langle C, I \rangle \to \langle C \cup \{a\}, I \rangle}$$

Mimics Promise requirements!

Issue step:

$$\frac{\ldots}{G \vdash \langle C, I \rangle \to \langle C, I \cup \{w\} \rangle}$$

Proof Structure

1. Prove $\mathrm{Promise}$ simulates traversal

Proof Structure

1. Prove $\mathrm{Promise}$ simulates traversal

2. Show completeness of traversal

Proof Structure

1. Prove $\mathrm{Promise}$ simulates traversal

2. Show completeness of traversal

$$\forall G.\ G \in \mathsf{Consistent}(\mathrm{ARMv8.3}) \Rightarrow \langle \emptyset, \emptyset \rangle \rightarrow^* \langle G.\mathsf{Events}, G.\mathsf{Writes} \rangle$$

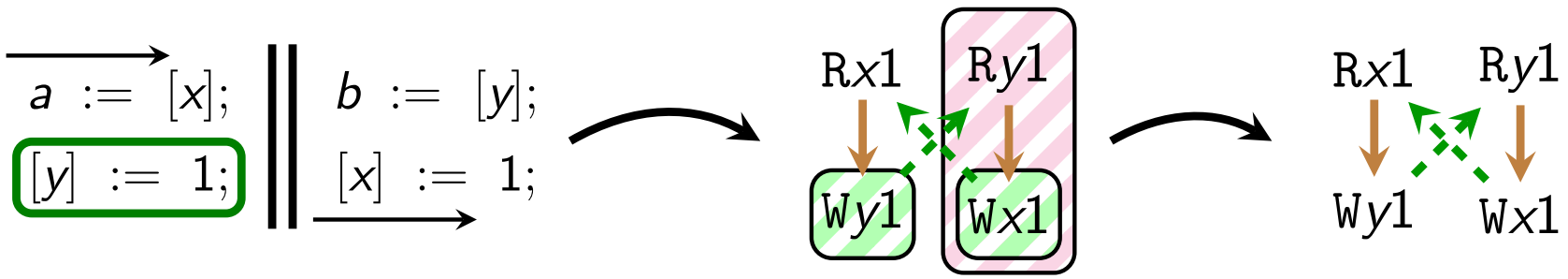
Future Work

- Cover other features of Promise

  (Read-Modify-Writes, Release/Acquire accesses, SC fences)

- Proof mechanization in Coq

Promising Compilation to ARMv8.3



`http://podkopaev.net/armpromise`

Thank you!

Links I


Alglave, J., Maranget, L., and Tautschnig, M. (2014).
Herding cats: Modelling, simulation, testing, and data mining for weak memory.
ACM Trans. Program. Lang. Syst., 36(2):7:1–7:74.


Batty, M., Owens, S., Sarkar, S., Sewell, P., and Weber, T. (2011).
Mathematizing C++ concurrency.
In *POPL 2011*, pages 55–66. ACM.


Flur, S., Gray, K. E., Pulte, C., Sarkar, S., Sezgin, A., Maranget, L., Deacon, W., and Sewell, P. (2016).
Modelling the ARMv8 architecture, operationally: Concurrency and ISA.
In *POPL 2016*, pages 608–621. ACM.


Kang, J., Hur, C.-K., Lahav, O., Vafeiadis, V., and Dreyer, D. (2017).
A promising semantics for relaxed-memory concurrency.
In *POPL 2017*. ACM.


Manson, J., Pugh, W., and Adve, S. V. (2005).
The Java memory model.
In *POPL 2005*, pages 378–391. ACM.


Owens, S., Sarkar, S., and Sewell, P. (2009).
A better x86 memory model: x86-TSO.
In *TPHOL 2009*, pages 391–407.

Links II

Podkopaev, A., Lahav, O., and Vafeiadis, V. (2017).
Promising compilation to ARMv8 POP.
In *ECOOP 2017*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Pulte, C., Flur, S., Deacon, W., French, J., Sarkar, S., and Sewell, P. (2018).
Simplifying ARM concurrency: Multicopy-atomic axiomatic and operational models for ARMv8.

Compilation Correctness

$$compile : S \rightarrow T$$
$$\forall Prog \in S.$$
$$[\![compile(Prog)]\!]_T \subseteq [\![Prog]\!]_S.$$