Обещающая компиляция в ARMv8

Антон Подкопаев[1] Ори Лахав[2] Виктор Вафеядис[2]

[1]СПбГУ, JetBrains Research, Россия

[2]Институт Макса Планка: Программные Системы, Германия

04.04.2017

Что нужно для быстрого ПО?

Что нужно для быстрого ПО?

- Хороший Алгоритм

Что нужно для быстрого ПО?

- Хороший Алгоритм
- Эффективный Компилятор

Что нужно для быстрого ПО?

- Хороший Алгоритм
- Эффективный Компилятор
- Производительный Процессор

Что нужно для быстрого ПО?

- Хороший Алгоритм
- Эффективный Компилятор
- Производительный Процессор

Что нужно для быстрого ПО?

- Хороший Алгоритм
- **?Эффективный?** Компилятор
- Производительный Процессор

Что нужно для быстрого ПО?

- Хороший Алгоритм
- **Оптимизирующий** Компилятор
- Производительный Процессор

Что нужно для быстрого ПО?

- Хороший Алгоритм
- **Оптимизирующий** Компилятор
- **?Производительный?** Процессор

Что нужно для быстрого ПО?

- Хороший Алгоритм
- **Оптимизирующий** Компилятор
- **Оптимизирующий** Процессор

$$[x] := 1;$$
$$a := [y]$$

Компилятор:
Независимые обращения.
Можно переупорядочить.

$$[x] := 1;$$
$$a := [y]$$

$$[x] := 1;$$
$$a := [y]$$

Процессор:

Независимые обращения.

Можно выполнить не по порядку.

$$[x] := 1;$$
$$a := [y]$$

Всегда ли корректны такие преобразования?

$$[x] := 1;$$
$$a := [y]$$

$$
\begin{array}{c}
[x] := 0; [y] := 0 \\
[x] := 1; \parallel [y] := 1; \\
a := [y] \parallel b := [x]
\end{array}
$$

$$
\begin{array}{c|c|c}
[x] := 0; [y] := 0 & & \\
\hline
[x] := 1; & \| & [y] := 1; \\
a := [y] & \| & b := [x]
\end{array}
$$

$$a = b = 0$$

$$[x] := 0; [y] := 0$$

A если переупорядочить?

$$a = b = 0$$

$$
\begin{array}{l}
[x] := 0; [y] := 0 \\
\left(\begin{array}{l|l}
[x] := 1; & [y] := 1; \\
a := [y] & b := [x]
\end{array}\right)
\end{array}
$$

$$a = b = 0$$

$$
\begin{array}{c||c}
[x] := 0; [y] := 0 & \\
a := [y]; & [y] := 1; \\
[x] := 1 & b := [x]
\end{array}
$$

$$a = b = 0$$

$$
\begin{array}{c|c}
[x] := 0; [y] := 0 & \\
a := [y]; & [y] := 1; \\
[x] := 1 & b := [x]
\end{array}
$$

$$a = b = 0$$

$$
\begin{array}{r}
[x] := 0; [y] := 0 \\
\left(\begin{array}{l|l}
[x] := 1; & [y] := 1; \\
a := [y] & b := [x]
\end{array}\right.
\end{array}
$$

$$a = b = 0$$

$$[x] := 0; [y] := 0$$

$$\left(\begin{array}{l|l} [x] := 1; & [y] := 1; \\ a := [y] & b := [x] \end{array}\right.$$

$$a = b = 0$$

Такое поведение наблюдается в реальности (например, GCC + x86)!

Модели памяти процессоров

Модели памяти ЯП

Модели памяти процессоров

- x86, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARM, [Flur et al., 2016]
- ...

Модели памяти ЯП

Модели памяти процессоров

- x86, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARM, [Flur et al., 2016]
- ...

Модели памяти ЯП

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

Модели памяти процессоров

- x86, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARM, [Flur et al., 2016]
- ...

Модели памяти ЯП

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

Имеют ряд существенных недостатков

Модели памяти процессоров

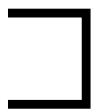
- x86, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARM, [Flur et al., 2016]
- ...

Модели памяти ЯП

- C/C++11, [Batty et al., 2011]
- Java, [Manson et al., 2005]
- “Обещающая” семантика, [Kang et al., 2017]

Модели памяти процессоров

- x86, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARM, [Flur et al., 2016]
- ...



Модели памяти ЯП

- C/C++11, [Batty et al., 2011]
- Java, [Manson et al., 2005]
- “Обещающая” семантика, [Kang et al., 2017]

Модели памяти процессоров

- x86, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARM, [Flur et al., 2016]
- ...

Корректность компиляции показана в [Kang et al., 2017]

Модели памяти ЯП

- C/C++11, [Batty et al., 2011]
- Java, [Manson et al., 2005]
- “Обещающая” семантика, [Kang et al., 2017]

Модели памяти процессоров

- x86, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARM, [Flur et al., 2016]
- ...

Корректность компиляции показана в [Kang et al., 2017]

Та же схема доказательства не подходит для ARM!

Модели памяти ЯП

- C/C++11, [Batty et al., 2011]
- Java, [Manson et al., 2005]
- “Обещающая” семантика, [Kang et al., 2017]

Модели памяти процессоров

- x86, [Owens et al., 2009]
- Power, [Alglave et al., 2014]
- ARM, [Flur et al., 2016]
- ...

Модели памяти ЯП

- C/C++11, [Batty et al., 2011]
- Java, [Manson et al., 2005]
- “Обещающая” семантика, [Kang et al., 2017]

Результаты данной работы

- Доказана корректность компиляции подмножества “обещающей” семантики [Kang et al., 2017] в модель ARMv8 [Flur et al., 2016]

Результаты данной работы

- Доказана корректность компиляции подмножества “обещающей” семантики [Kang et al., 2017] в моде

Расслабленные (relaxed) чтения и записи, высвобождающие (release) и преобретающие (acquire) барьеры памяти

Результаты данной работы

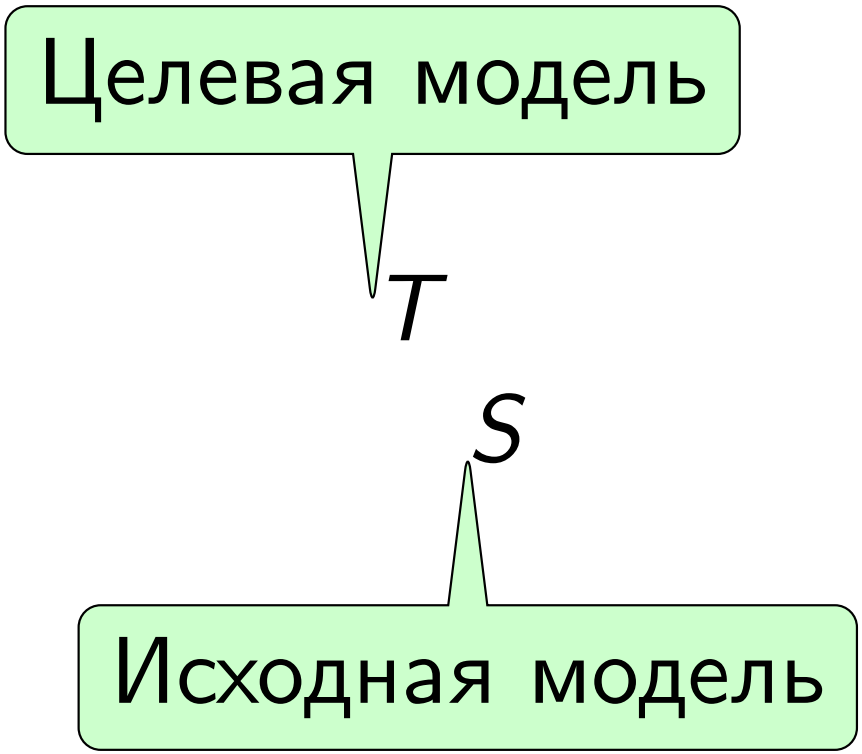
- Доказана корректность компиляции подмножества “обещающей” семантики [Kang et al., 2017] в модель ARMv8 [Flur et al., 2016]

- Формализована модель ARMv8 [Flur et al., 2016] и доказаны вспомогательные утверждения про неё

Корректность компиляции...

Корректность компиляции... Что это значит?

Корректность компиляции из S в T



Корректность компиляции из S в T

Исходная программа

$\forall Prog \in Syntax,$

T

$compile(Prog)$

S

Результат компиляции

Корректность компиляции из S в T

$$\forall Prog \in Syntax,$$
$$\{t_j\}_{j\in[1..k]} \underset{S}{-} T\text{-исполнение } compile(Prog).$$

Корректность компиляции из S в T

$\forall Prog \in Syntax,$

$\{t_j\}_{j \in [1..k]}$ – T-исполнение $compile(Prog)$.

$\exists \{s_i\}_{i \in [1..n]}$ – S-исполнение $Prog$,

Корректность компиляции из S в T

$\forall Prog \in Syntax,$

$\{t_j\}_{j \in [1..k]}$ – T-исполнение $compile(Prog)$.

$\exists \{s_i\}_{i \in [1..n]}$ – S-исполнение $Prog$,

$s_n \simeq t_k$.

Корректность компиляции из S в T

$\forall Prog \in Syntax,$

$\{t_j\}_{j \in [1..k]}$ – T-исполнение $compile(Prog)$.

$\exists \{s_i\}_{i \in [1..n]}$ – S-исполнение $Prog$,

$s_n \simeq t_k$.

e.g., финальное состояние памяти совпадает

Корректность компиляции из S в T

$\forall Prog \in Syntax,$

$\{t_j\}_{j \in [1..k]}$ – T-исполнение $compile(Prog)$.

$\exists \{s_i\}_{i \in [1..n]}$ – S-исполнение $Prog$,

$s_n \simeq t_k$.

Рассматриваем $compile = id$

Корректность компиляции из S в T

$\forall Prog \in Syntax,$

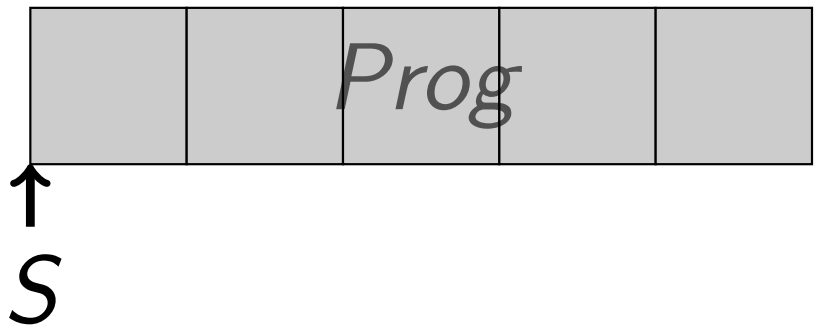
$\{t_j\}_{j \in [1..k]}$ – T-исполнение $Prog$.

$\exists \{s_i\}_{i \in [1..n]}$ – S-исполнение $Prog$,

$s_n \simeq t_k$.

Стандартная техника — симуляция

Пример симуляции



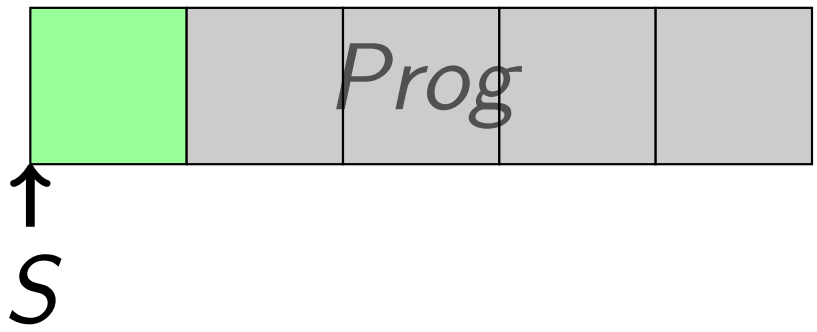
Инвариант выполняется

– выполнено T

S – исходная модель

T – целевая модель

Пример симуляции



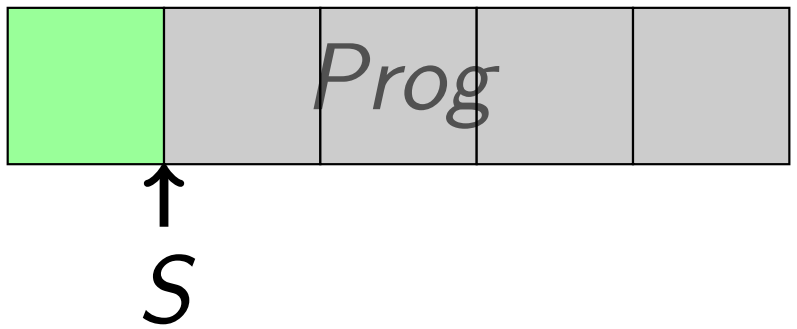
Инвариант не выполняется

– выполнено T

S – исходная модель

T – целевая модель

Пример симуляции



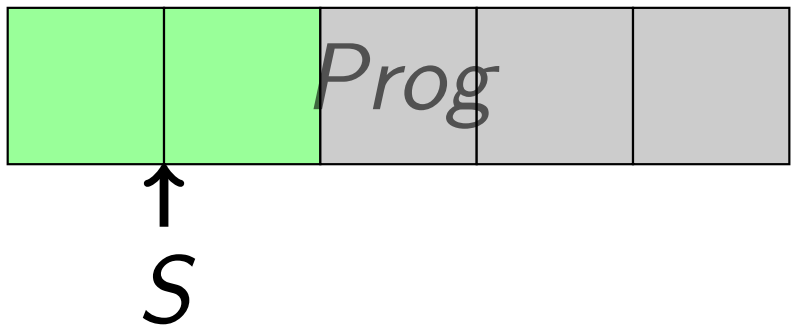
Инвариант выполняется

– выполнено T

S – исходная модель

T – целевая модель

Пример симуляции



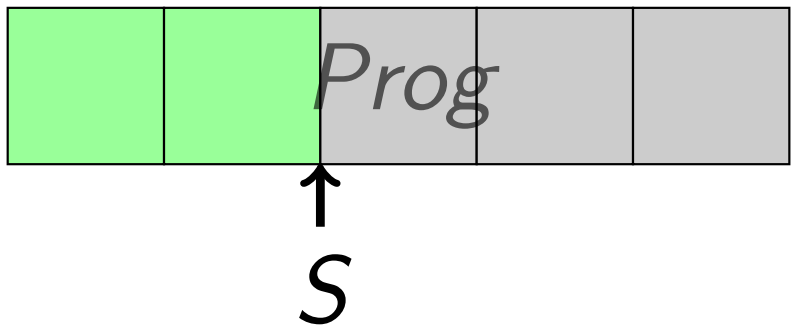
Инвариант не выполняется

– выполнено T

S – исходная модель

T – целевая модель

Пример симуляции



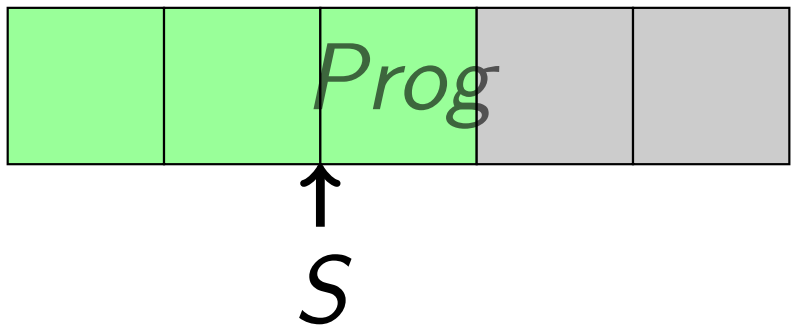
Инвариант выполняется

– выполнено T

S – исходная модель

T – целевая модель

Пример симуляции



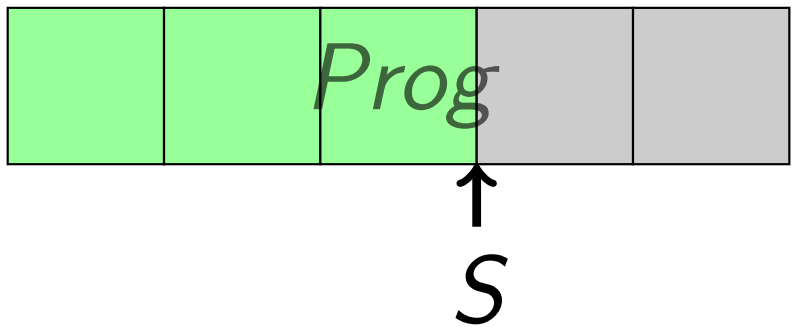
Инвариант не выполняется

– выполнено T

S – исходная модель

T – целевая модель

Пример симуляции



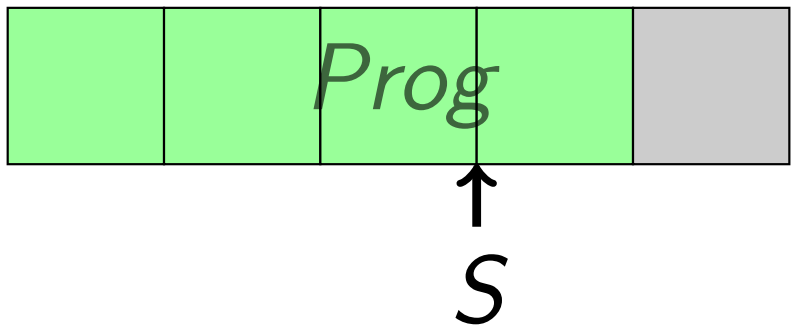
Инвариант выполняется

– выполнено T

S – исходная модель

T – целевая модель

Пример симуляции



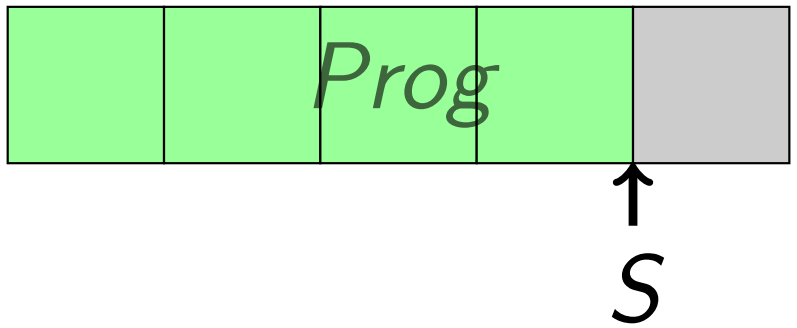
Инвариант не выполняется

– выполнено T

S – исходная модель

T – целевая модель

Пример симуляции



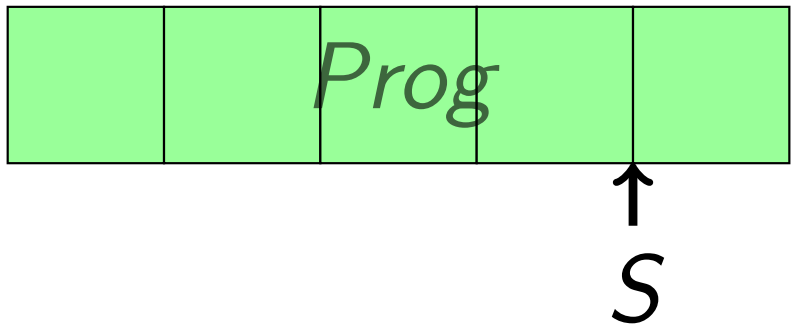
Инвариант выполняется

– выполнено T

S – исходная модель

T – целевая модель

Пример симуляции



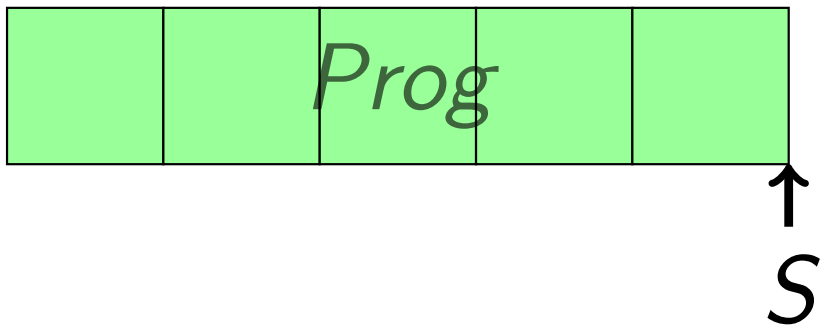
Инвариант не выполняется

– выполнено T

S – исходная модель

T – целевая модель

Пример симуляции



Инвариант выполняется

– выполнено T

S – исходная модель

T – целевая модель

...но симуляция напрямую не применина в нашем случае

...но симуляция напрямую не применина в нашем случае:

1. ARM выполняет инструкции не по порядку;
2. “Обещающая” семантика имеет больше явных ограничений.

План доказательства

План доказательства

1. Вводим промежуточную семантику ARM+τ

План доказательства

1. Вводим промежуточную семантику ARM+τ
   - обладает явными ограниченями, похожими на “обещающую” семантику;

План доказательства

1. Вводим промежуточную семантику ARM+τ
   - обладает явными ограниченями, похожими на “обещающую” семантику;
2. Доказываем бисимуляцию между ARM+τ и ARM;

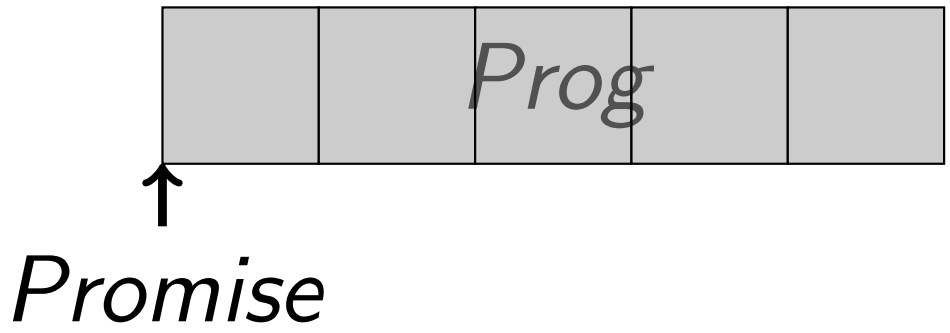
План доказательства

1. Вводим промежуточную семантику ARM+τ
   - обладает явными ограниченями, похожими на “обещающую” семантику;
2. Доказываем бисимуляцию между ARM+τ и ARM;
3. Показываем “запаздывающую” симуляцию ARM+τ “обещающей” семантикой.

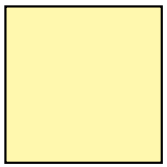
План доказательства

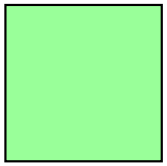
1. Вводим промежуточную семантику ARM+τ
   - обладает явными ограниченями, похожими на “обещающую” семантику;
2. Доказываем бисимуляцию между ARM+τ и ARM;
3. Показываем “запаздывающую” симуляцию ARM+τ “обещающей” семантикой.

“Запаздывающая” симуляция



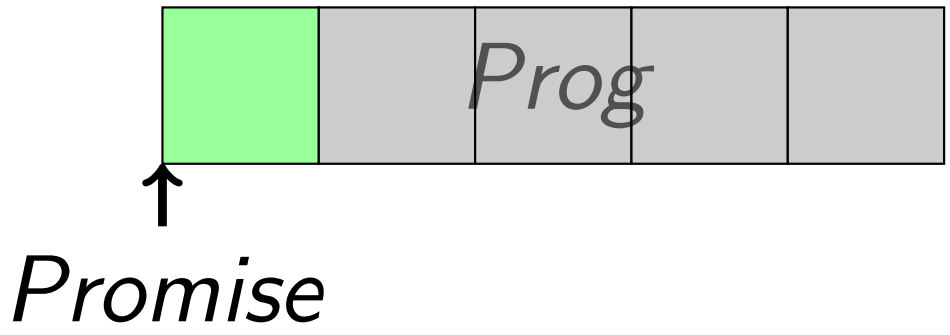
Инвариант: “обещающая” машина ждёт

 – частично выполнено *ARMv8*

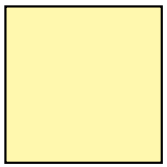
 – выполнено *ARMv8*

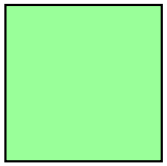
Promise – “обещающая” модель

“Запаздывающая” симуляция



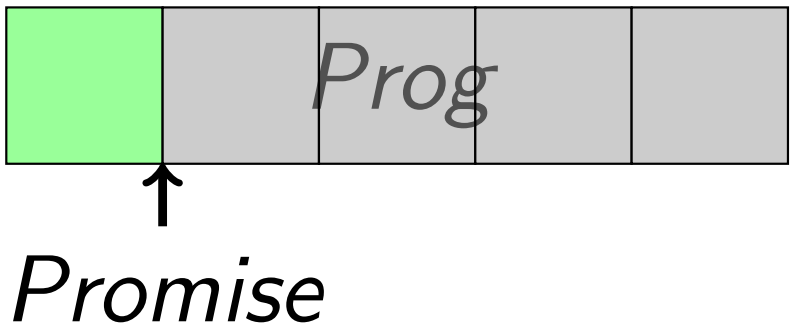
Инвариант: “обещающая” машина исполняется

 – частично выполнено *ARMv8*

 – выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция



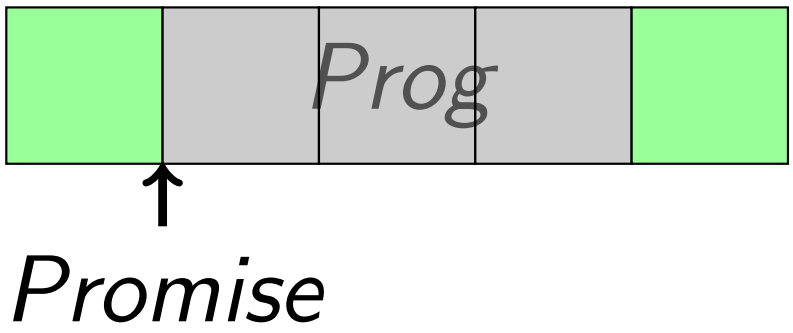
Инвариант: “обещающая” машина ждёт

– частично выполнено *ARMv8*

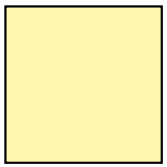
– выполнено *ARMv8*

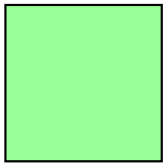
Promise – “обещающая” модель

“Запаздывающая” симуляция



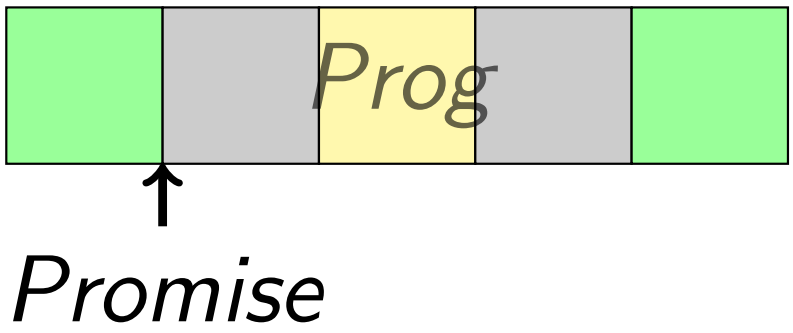
Инвариант: “обещающая” машина ждёт

 – частично выполнено *ARMv8*

 – выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция



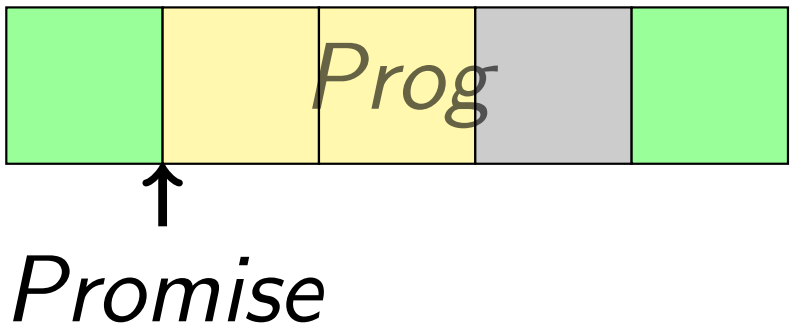
Инвариант: “обещающая” машина ждёт

– частично выполнено *ARMv8*

– выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция



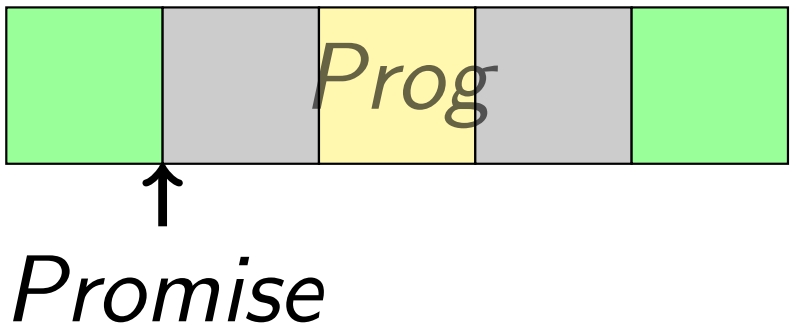
Инвариант: “обещающая” машина ждёт

– частично выполнено *ARMv8*

– выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция



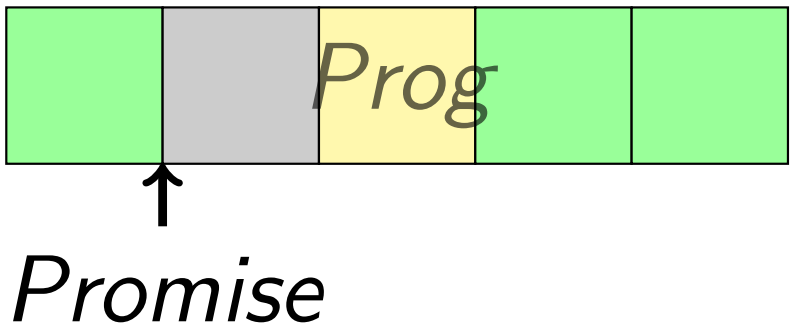
Инвариант: “обещающая” машина ждёт

– частично выполнено *ARMv8*

– выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция



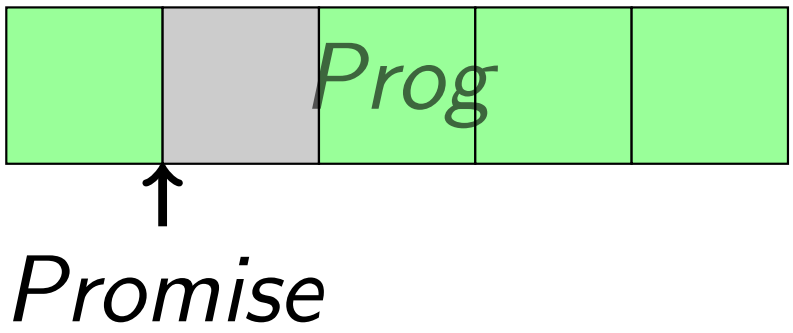
Инвариант: “обещающая” машина ждёт

– частично выполнено *ARMv8*

– выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция



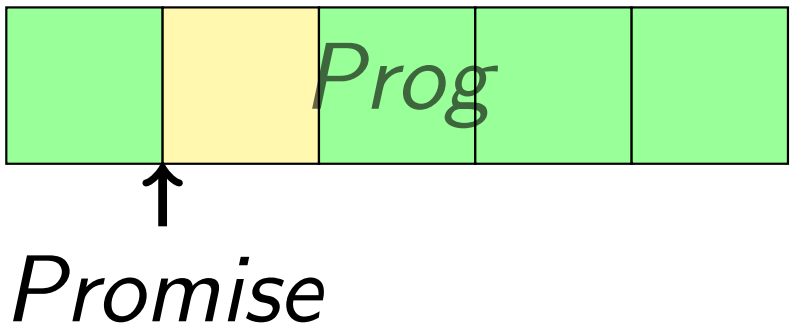
Инвариант: “обещающая” машина ждёт

– частично выполнено *ARMv8*

– выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция



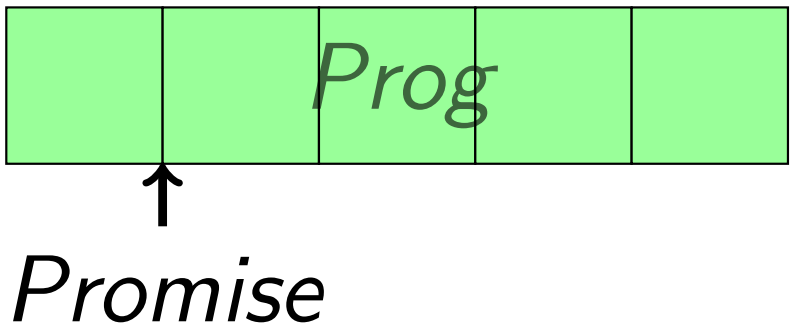
Инвариант: “обещающая” машина ждёт

– частично выполнено *ARMv8*

– выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция



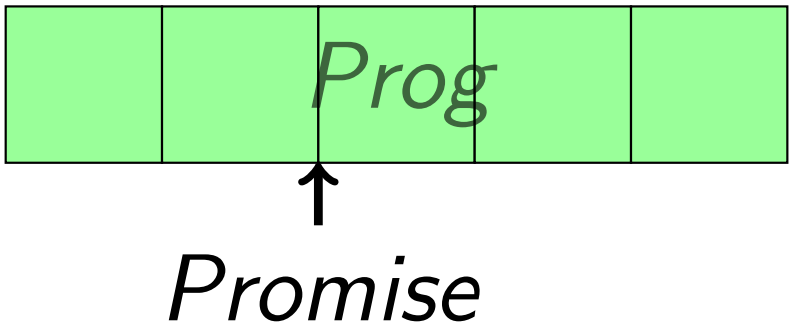
Инвариант: “обещающая” машина исполняется

– частично выполнено *ARMv8*

– выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция



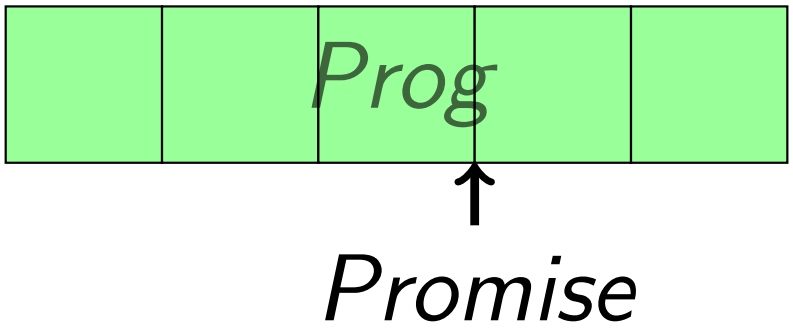
Инвариант: “обещающая” машина исполняется

– частично выполнено *ARMv8*

– выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция



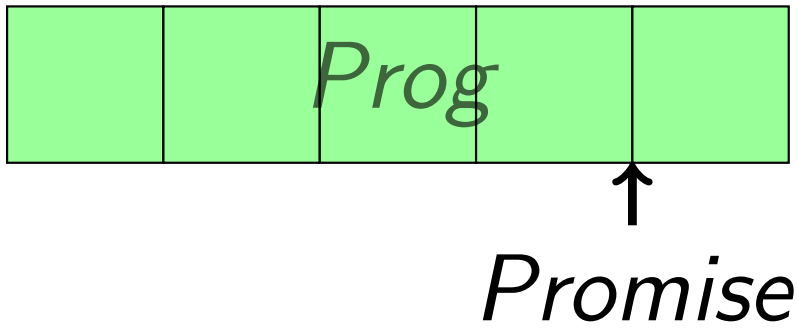
Инвариант: “обещающая” машина исполняется

– частично выполнено *ARMv8*

– выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция



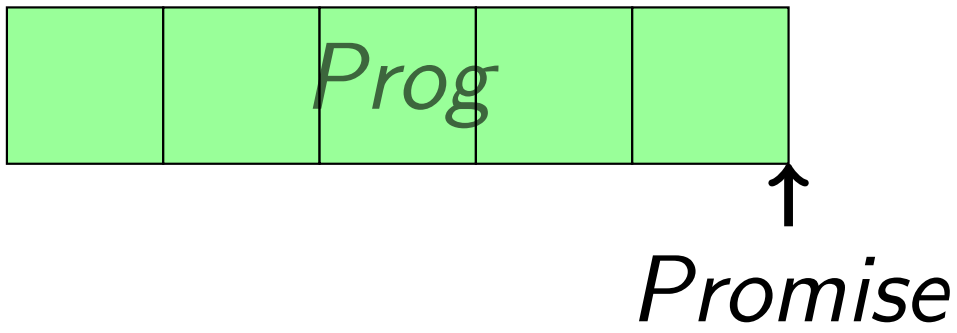
Инвариант: “обещающая” машина исполняется

– частично выполнено *ARMv8*

– выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция



Инвариант: “обещающая” машина ждёт

– частично выполнено *ARMv8*

– выполнено *ARMv8*

Promise – “обещающая” модель

“Запаздывающая” симуляция. Формально

Лемма 1:
$\forall (\mathbf{a}, \mathbf{p}) \in \mathcal{I}_{\text{pre}}, \exists \mathbf{p}', \mathbf{p} \xrightarrow[\text{Promise}]{} \mathbf{p}', (\mathbf{a}, \mathbf{p}') \in \mathcal{I}_{\text{pre}} \cup \mathcal{I}.$

Лемма 2:
$\forall (\mathbf{a}, \mathbf{p}) \in \mathcal{I}_{\text{pre}}, \exists n, \{\mathbf{p}_i\}_{i \in [0,n]}, \mathbf{p}_0 = \mathbf{p}, (\forall i < n, \mathbf{p}_i \xrightarrow[\text{Promise}]{} \mathbf{p}_{i+1}),$
$(\forall i < n, (\mathbf{a}, \mathbf{p}_i) \in \mathcal{I}_{\text{pre}}), (\mathbf{a}, \mathbf{p}_n) \in \mathcal{I}.$

Лемма 3:
$\forall (\mathbf{a}, \mathbf{p}) \in \mathcal{I},$
$(\forall \mathbf{a}', \mathbf{a} \xrightarrow[\text{ARM}+\tau]{\neg\ \textbf{Write commit}} \mathbf{a}' \Rightarrow (\mathbf{a}', \mathbf{p}) \in \mathcal{I}_{\text{pre}} \cup \mathcal{I}) \wedge$
$(\forall \mathbf{a}', \mathbf{a} \xrightarrow[\text{ARM}+\tau]{\textbf{Write commit}} \mathbf{a}' \Rightarrow \exists \mathbf{p}', \mathbf{p} \xrightarrow[\text{Promise}]{\textbf{Promise write}} \mathbf{p}', (\mathbf{a}', \mathbf{p}') \in \mathcal{I}_{\text{pre}} \cup \mathcal{I}).$

Лемма 4:
$\forall (\mathbf{a}, \mathbf{p}) \in \mathcal{I}, \mathbf{a}', \mathbf{a} \xrightarrow[\text{ARM}+\tau]{} \mathbf{a}' \Rightarrow$
$\exists \mathbf{p}', \mathbf{p} \xrightarrow[\text{Promise}]{}^{*} \mathbf{p}', (\mathbf{a}', \mathbf{p}') \in \mathcal{I}.$

Теорема:
$\forall \mathit{Prog}, \{\mathbf{a}_i\}_{i \in [0..n]},$
$\mathbf{a}^{\text{init}}(\mathit{compile}(\mathit{Prog})) = \mathbf{a}_0 \xrightarrow[\text{ARM}+\tau]{} \ldots \xrightarrow[\text{ARM}+\tau]{} \mathbf{a}_n, \text{Final}^{\text{ARM}+\tau}(\mathbf{a}_n),$
$\exists \{\mathbf{p}_i\}_{i \in [0..k]},$
$\mathbf{p}^{\text{init}}(\mathit{Prog}) = \mathbf{p}_0 \xrightarrow[\text{Promise}]{} \ldots \xrightarrow[\text{Promise}]{} \mathbf{p}_k, \text{Final}^{\text{Promise}}(\mathbf{p}_k),$
$\text{same-memory}(\mathbf{a}_n, \mathbf{p}_k).$

Promising Compilation to ARMv8

Anton Podkopaev[1], Ori Lahav[2], and Viktor Vafeiadis[2]

1 a.podkopaev@2009.spbu.ru, SPbU, JetBrains, Russia
2 {orilahav,viktor}@mpi-sws.org, MPI-SWS, Germany

Abstract

We prove the correctness of compilation of relaxed memory accesses and release-acquire fences from the "promising" semantics of Kang et al. [8] to the ARMv8 POP machine of Flur et al. [5]. The proof is highly non-trivial, because both the ARMv8 and the promising semantics provide some extremely weak consistency guarantees for normal memory accesses; however, they do so in rather different ways. Being the first formal proof about the ARMv8 POP model, in the process we also had to formalize the semantics of Flur et al. Our proof of compilation correctness to ARMv8 strengthens the results of the Kang et al., who only proved correctness of compilation to x86-TSO and Power, which are much simpler in comparison to ARMv8.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases ARM, Compilation Correctness, Weak Memory Model

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

One of the major unresolved topics in the semantics of programming languages has to do with giving semantics to concurrent shared-memory programs. While it is well understood that such programs cannot follow the naive paradigm of *sequential consistency* (SC) [11], it is not completely clear what the right semantics of such concurrent programs should be.

At the level of machine code, the semantics varies a lot depending on the hardware architecture, which is only loosely specified by the vendor manuals. In the last decade, academic researchers have produced formal models for the mainstream hardware architectures (e.g., x86-TSO [17], Power [16, 1], ARMv8 [5]) by engaging in discussions with hardware architects and subjecting existing hardware implementations to extensive tests.

In this paper, we will focus on the ARMv8 model due to Flur et al. [5], which is the most recent and arguably the most advanced such hardware memory model. Operational in nature, it models many low-level hardware features that affect the execution of concurrent

Планы на будущее

- Поддержка остальных конструкций “обещающей” семантики

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

- Механизация доказательства в Coq

Планы на будущее

- Поддержка остальных конструкций “обещающей” семантики

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

- Механизация доказательства в Coq

Спасибо!

Ссылки 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.


Lahav, O. and Vafeiadis, V. (2016).
Explaining Relaxed Memory Models with Program Transformations.
In *FM 2016*. Springer.


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

Ссылки II

Owens, S., Sarkar, S., and Sewell, P. (2009).
A better x86 memory model: x86-TSO.
In *TPHOLs*, volume 5674 of *LNCS*, pages 391–407. Springer.

The first step is a **formal semantics**

PL semantics should

PL semantics should

- allow efficient implementation;

PL semantics should

- allow efficient implementation;
  (x86, Power, ARM)

PL semantics should

- allow efficient implementation;
  (x86, Power, ARM)
- validate compiler optimizations;

PL semantics should

- allow efficient implementation;
  (x86, Power, ARM)
- validate compiler optimizations;
  (merging, rearranging, etc)

PL semantics should

- allow efficient implementation;
  (x86, Power, ARM)
- validate compiler optimizations;
  (merging, rearranging, etc)
- allow high-level reasoning.

	Eff. Impl.	Comp. Opt.	H.-l. Reasoning
Lamport's SC	✗	✗	✓
C/C++11 MM	✓	✓	✗
Java MM	✓	✗	✓

	Eff. Impl.	Comp. Opt.	H.-l. Reasoning
Lamport's SC	✗	✗	✓
C/C++11 MM	✓	✓	✗
Java MM	✓	✗	✓
A promising semantics			
[Kang et al., 2017]	✓	✓	✓

The Promise machine [Kang et al., 2017] is **proved** to be **compilable** to x86 and Power.

The Promise machine [Kang et al., 2017] is **proved** to be **compilable** to x86 and Power.

Sketch of the proof:

The Promise machine [Kang et al., 2017] is **proved** to be **compilable** to x86 and Power.

Sketch of the proof:

- x86 = SC + transformations,
  Power = "StrongPower" + transformations
  [Lahav and Vafeiadis, 2016];

The Promise machine [Kang et al., 2017] is **proved** to be **compilable** to x86 and Power.

Sketch of the proof:

- x86 = SC + transformations,
  Power = "StrongPower" + transformations
  [Lahav and Vafeiadis, 2016];
- the transformations are proved to be **sound** in the Promise machine;

The Promise machine [Kang et al., 2017] is **proved** to be **compilable** to x86 and Power.

Sketch of the proof:

- x86 = SC + transformations,
  Power = "StrongPower" + transformations
  [Lahav and Vafeiadis, 2016];
- the transformations are proved to be **sound** in the Promise machine;
- For every program, SC behaviors $\subset$ "StrongPower" behaviors $\subset$ behaviors of an axiomatic promise-free version of the Promise machine.

The Promise machine [Kang et al., 2017] is **proved** to be **compilable** to x86 and Power.

Sketch of the proof:

The proof scheme isn't applicable to **ARM**

$\subset$ behaviors of an axiomatic promise-free version of the Promise machine.

The Promise machine [Kang et al., 2017] is **proved** to be **compilable** to x86 and Power.

Sketch of the proof:

The proof scheme isn't applicable to **ARM**
and here is why:

$\subset$ behaviors of an axiomatic promise-free version of the Promise machine.

$$[x] := 0; [y] := 0$$

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

Allowed by ARMv8 [Flur et al., 2016]

$$[x] := 0; [y] := 0$$

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

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

The behavior cannot be explained by transformations over a strong enough model.

Compilation scheme

Promise	ARM
$[x]_{\mathsf{rlx}} := a$	$[x] := a$
$a := [x]_{\mathsf{rlx}}$	$a := [x]$
fence acq	dmb LD
fence rel	dmb SY

Compilation scheme

Promise	ARM
fence acq	dmb LD
fence rel	dmb SY

As the compilation scheme is bijection, we use **one** syntax in examples.

Example 1

$$
\begin{array}{c}
[x] := 0; [y] := 0 \\
\hline
[x] := 1; \;\|\; a := [y]; \\
[y] := 1 \;\|\; b := [x]
\end{array}
$$

Example 1

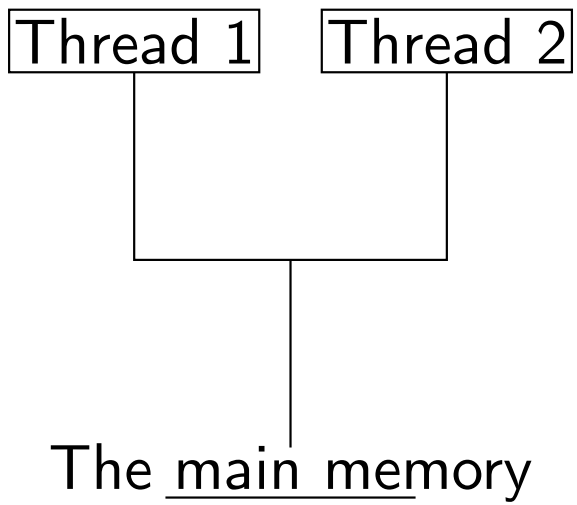
$$
\begin{array}{c}
[x] := 0; [y] := 0 \\
\hline
[x] := 1; \;\|\; a := [y]; \;\; /\!/\, 1 \\
[y] := 1 \;\|\; b := [x]
\end{array}
$$

Example 1

$$
\begin{array}{c}
[x] := 0; [y] := 0 \\
\hline
[x] := 1; \;\|\; a := [y]; \;\; /\!/\, 1 \\
[y] := 1 \;\|\; b := [x] \;\; /\!/\, 0
\end{array}
$$

Example 1

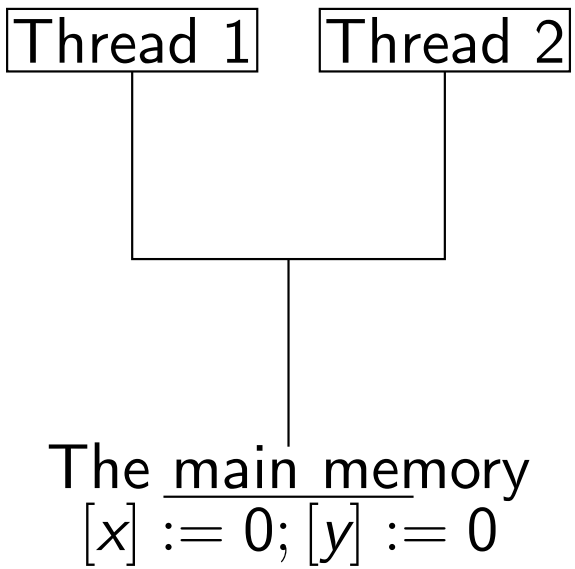
$$
\begin{array}{l}
[x] := 0; [y] := 0 \\
[x] := 1; \;\|\; a := [y]; \\
[y] := 1 \;\|\; b := [x]
\end{array}
$$



Example 1

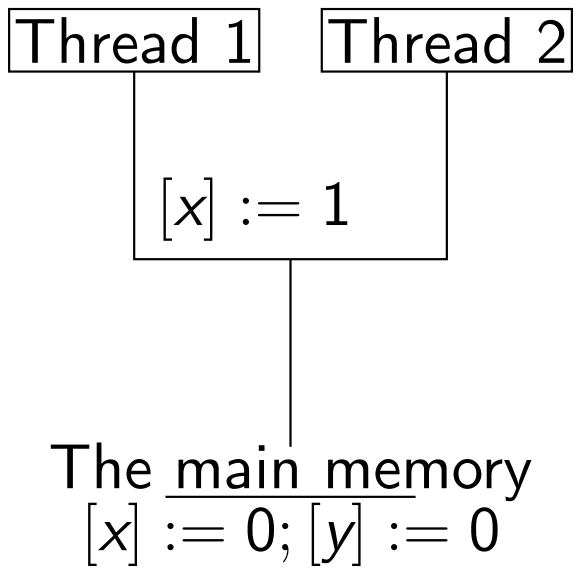
$$[x] := 0; [y] := 0$$

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



Example 1

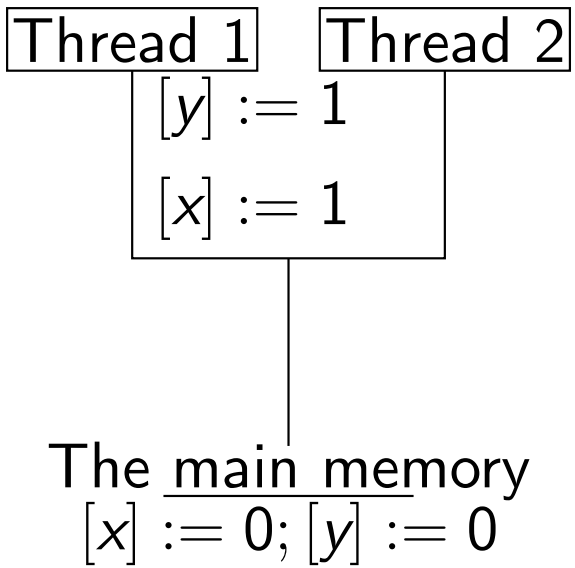
$$
\begin{array}{l||l}
[x] := 0; [y] := 0 & \\
[x] := 1; & a := [y]; \\
[y] := 1 & b := [x]
\end{array}
$$



Example 1

$$[x] := 0; [y] := 0$$

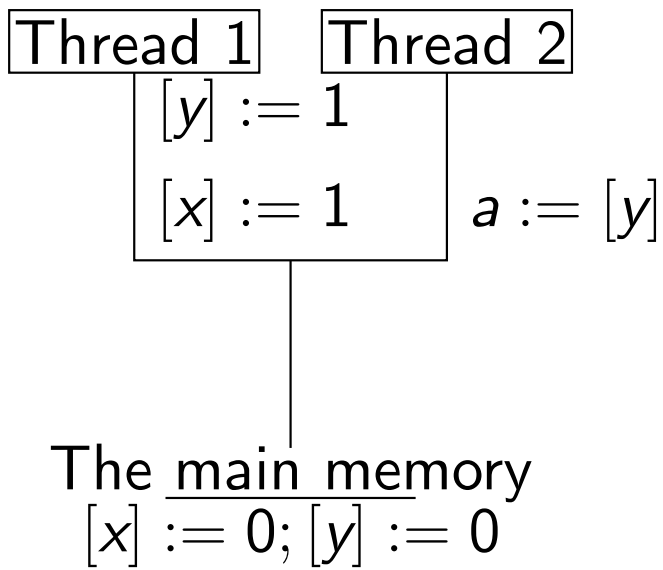
$$\begin{array}{l|l} [x] := 1; & a := [y]; \\ [y] := 1 & b := [x] \end{array}$$



Example 1

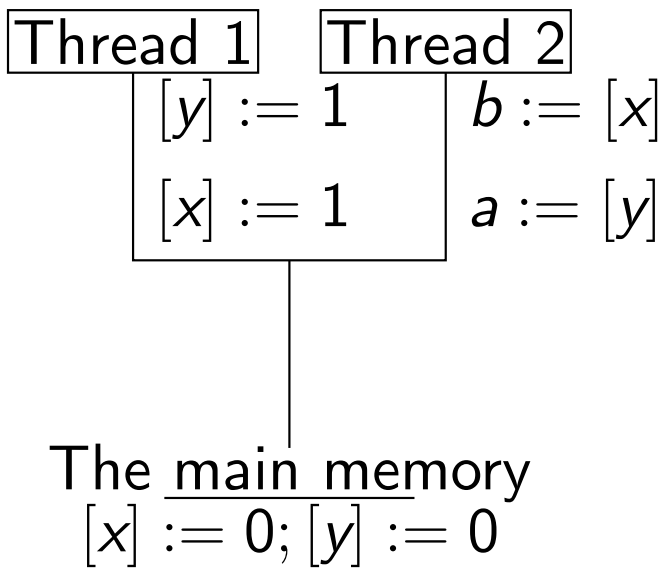
$$[x] := 0; [y] := 0$$

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



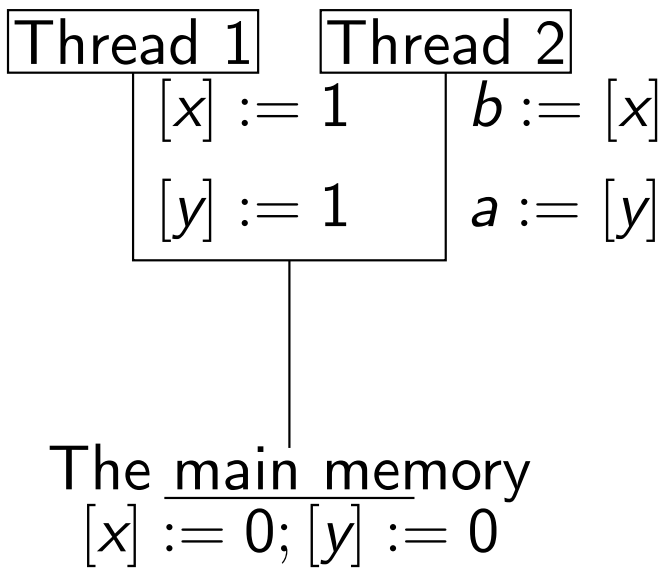
Example 1

$$
\begin{array}{l}
[x] := 0; [y] := 0 \\
\hline
[x] := 1; \;\|\; a := [y]; \\
[y] := 1 \;\|\; b := [x]
\end{array}
$$



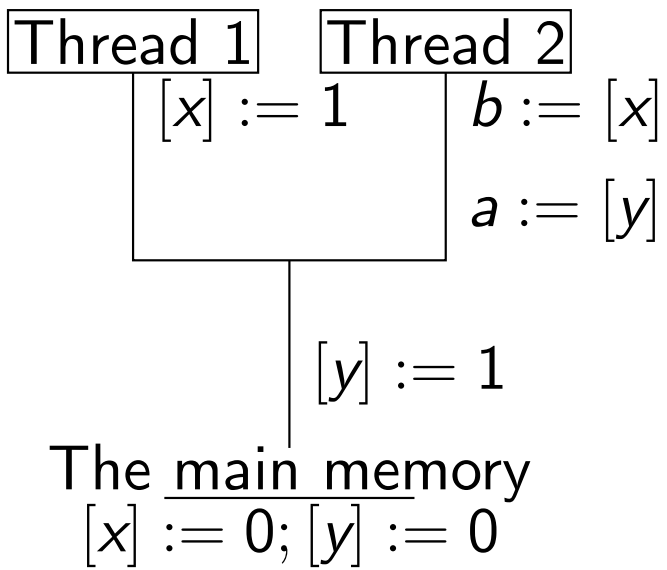
Example 1

$$
\begin{array}{l}
[x] := 0; [y] := 0 \\
\hline
[x] := 1; \;\|\; a := [y]; \\
[y] := 1 \;\|\; b := [x]
\end{array}
$$



Example 1

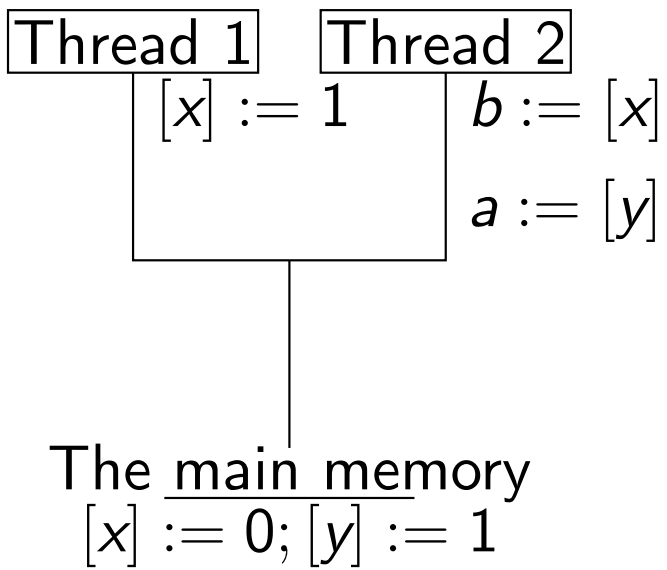
$$
\begin{array}{l}
[x] := 0; [y] := 0 \\
\hline
[x] := 1; \;\|\; a := [y]; \\
[y] := 1 \;\|\; b := [x]
\end{array}
$$



Example 1

$$[x] := 0; [y] := 0$$

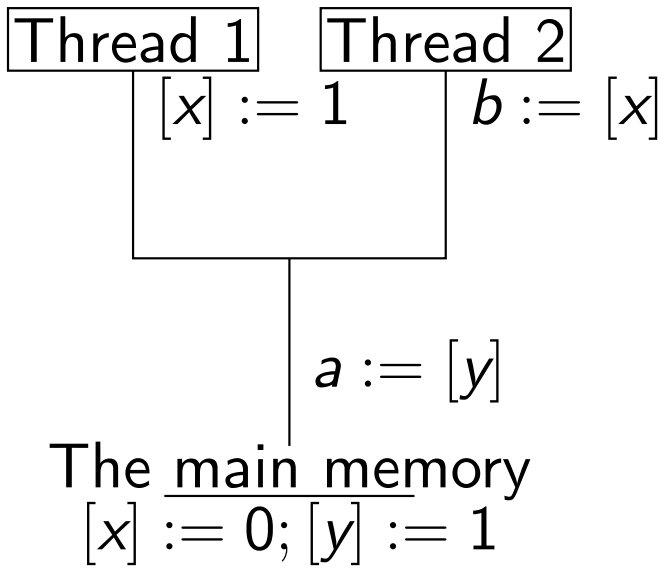
$$\begin{array}{l|l} [x] := 1; & a := [y]; \\ [y] := 1 & b := [x] \end{array}$$



Example 1

$$[x] := 0; [y] := 0$$

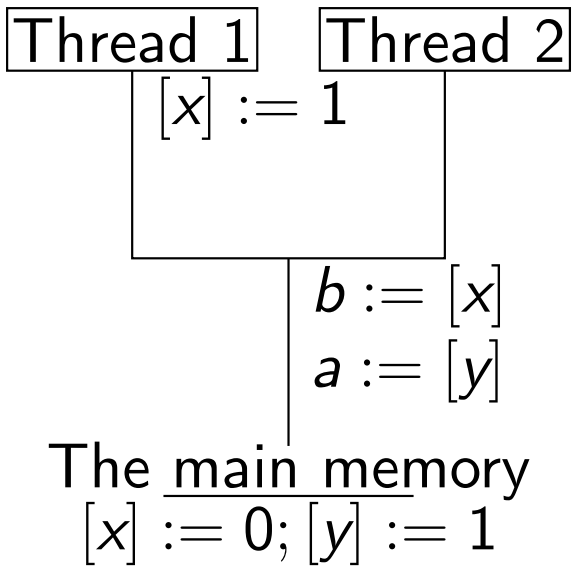
$$\begin{array}{l|l} [x] := 1; & a := [y]; \\ [y] := 1 & b := [x] \end{array}$$



Example 1

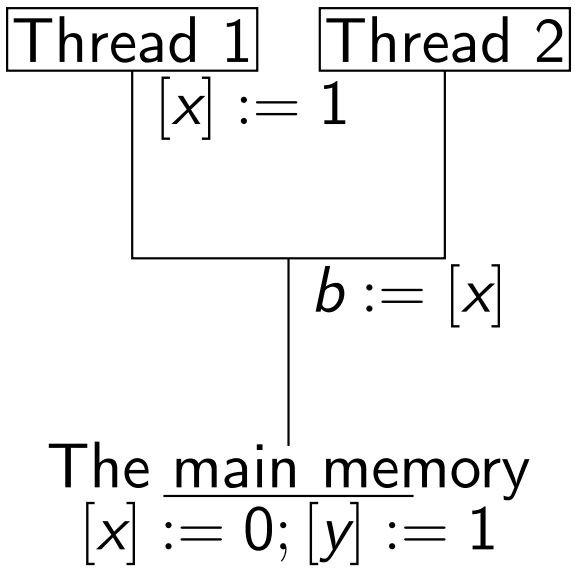
$$[x] := 0; [y] := 0$$

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



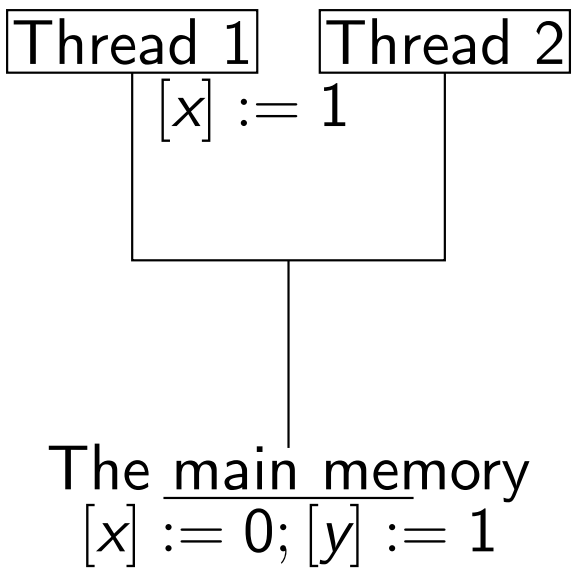
Example 1

$$
\begin{array}{l}
[x] := 0; [y] := 0 \\
\left.\begin{array}{l} [x] := 1; \\ [y] := 1 \end{array}\right. \;\Big\|\; \left.\begin{array}{l} a := [y]; \\ b := [x] \end{array}\right. \quad /\!/\ 1
\end{array}
$$



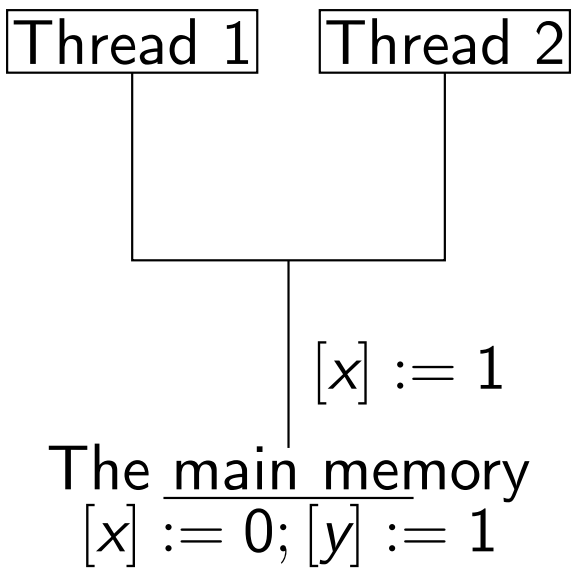
Example 1

$$
\begin{array}{l}
[x] := 0; [y] := 0 \\
\left.\begin{array}{l} [x] := 1; \\ [y] := 1 \end{array}\right. \;\Big\|\; \left.\begin{array}{l} a := [y]; \\ b := [x] \end{array}\right. \quad \begin{array}{l} /\!/\, 1 \\ /\!/\, 0 \end{array}
\end{array}
$$



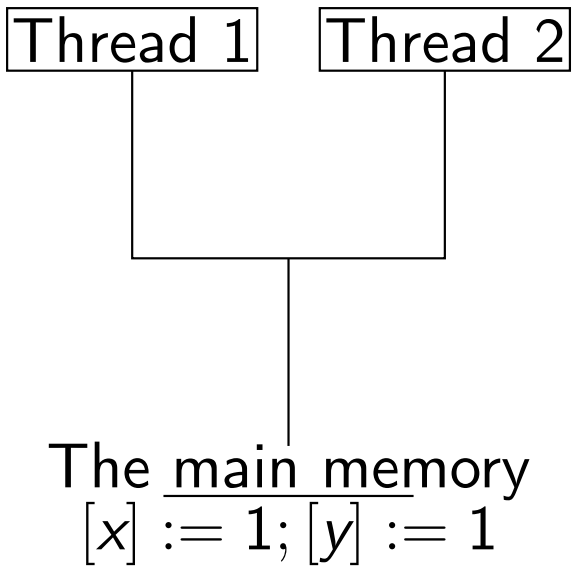
Example 1

$$
\begin{array}{l}
[x] := 0; [y] := 0 \\
\hline
[x] := 1; \;\|\; a := [y]; \quad // \; 1 \\
[y] := 1 \;\|\; b := [x] \quad // \; 0
\end{array}
$$



Example 1

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



Example 2

$$
\begin{array}{c}
[x] := 0; [y] := 0 \\
\hline
[x] := 1; \;\|\; a := [y]; \\
\text{dmb SY}; \;\|\; \text{dmb LD}; \\
[y] := 1 \;\|\; b := [x]
\end{array}
$$

Example 2

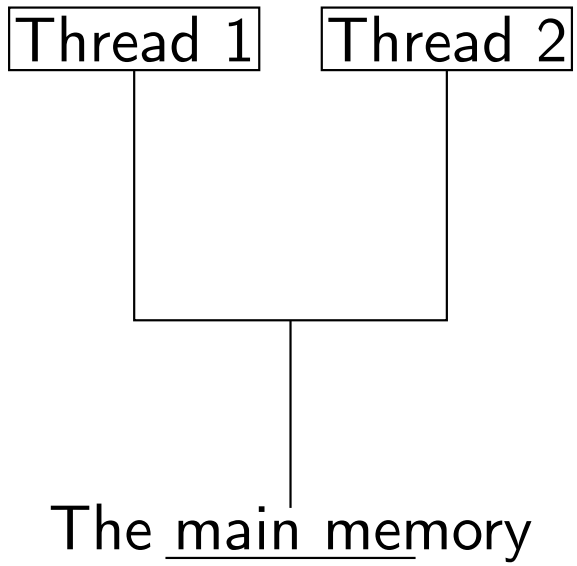
$$
\begin{array}{c}
[x] := 0; [y] := 0 \\
\hline
[x] := 1; \;\|\; a := [y]; \;\; /\!/\, 1 \\
\text{dmb SY}; \;\|\; \text{dmb LD}; \\
[y] := 1 \;\|\; b := [x]
\end{array}
$$

Example 2

$$
\begin{array}{c}
[x] := 0; [y] := 0 \\
\hline
[x] := 1; \;\|\; a := [y]; \;\; /\!/\, 1 \\
\text{dmb SY}; \;\|\; \text{dmb LD}; \\
[y] := 1 \;\|\; b := [x] \;\; /\!/\, 0
\end{array}
$$

Example 2

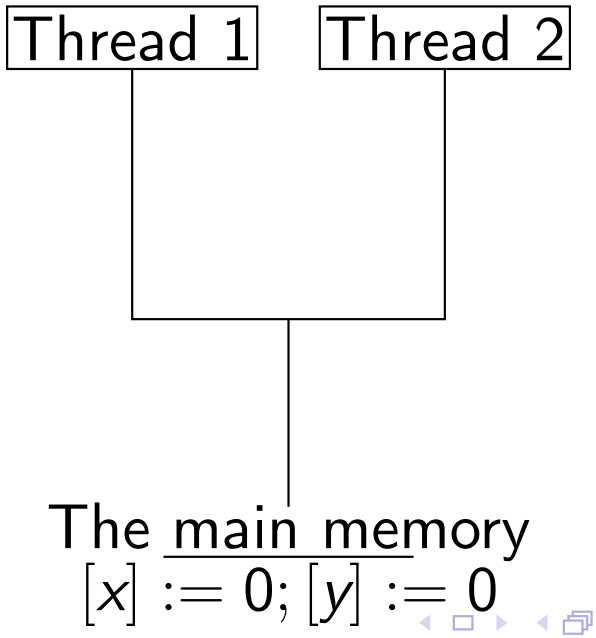
$$
\begin{array}{l}
[x] := 0; [y] := 0 \\
\hline
[x] := 1; \;\|\; a := [y]; \\
\text{dmb SY}; \;\|\; \text{dmb LD}; \\
[y] := 1 \;\|\; b := [x]
\end{array}
$$



Example 2

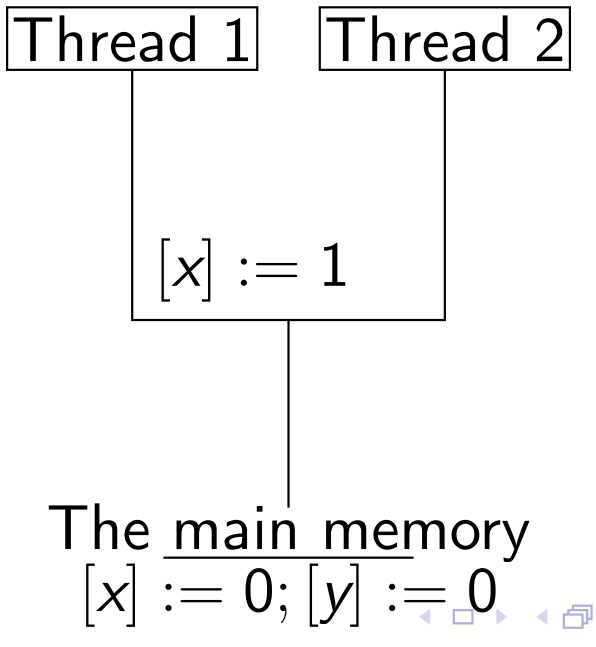
$[x] := 0; [y] := 0$

$[x] := 1;$	$a := [y];$
dmb SY;	dmb LD;
$[y] := 1$	$b := [x]$



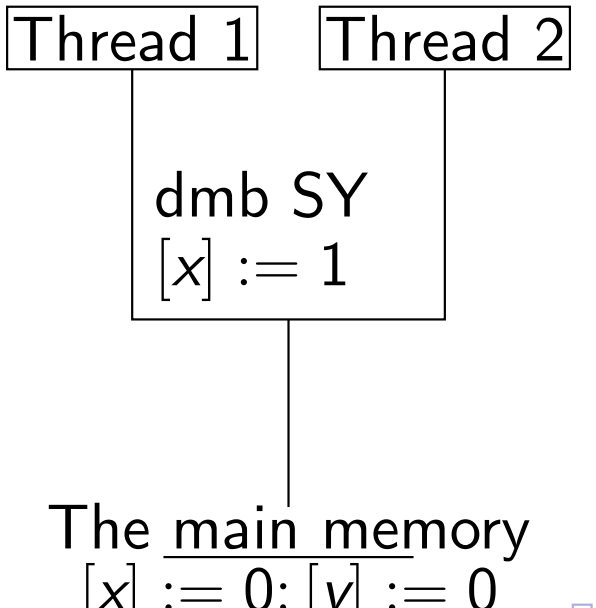
Example 2

$$
\begin{array}{l||l}
[x] := 0; [y] := 0 & \\
\hline
[x] := 1; & a := [y]; \\
\text{dmb SY}; & \text{dmb LD}; \\
[y] := 1 & b := [x]
\end{array}
$$



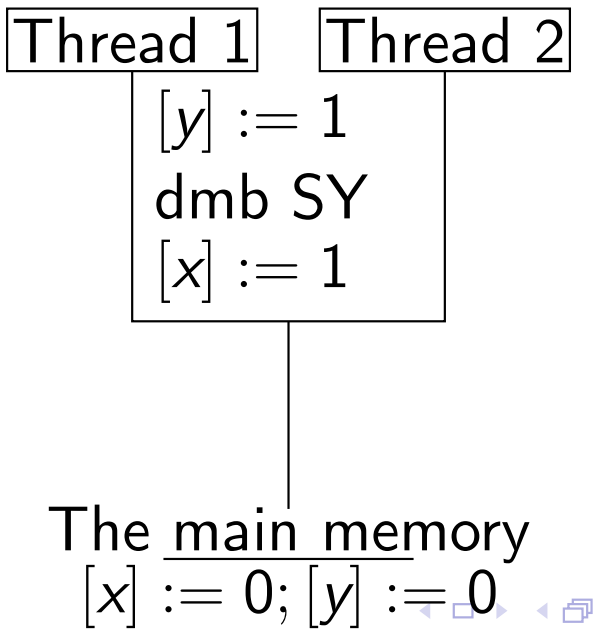
Example 2

$$
\begin{array}{l||l}
[x] := 0; [y] := 0 & \\
\hline
[x] := 1; & a := [y]; \\
\text{dmb SY}; & \text{dmb LD}; \\
[y] := 1 & b := [x]
\end{array}
$$



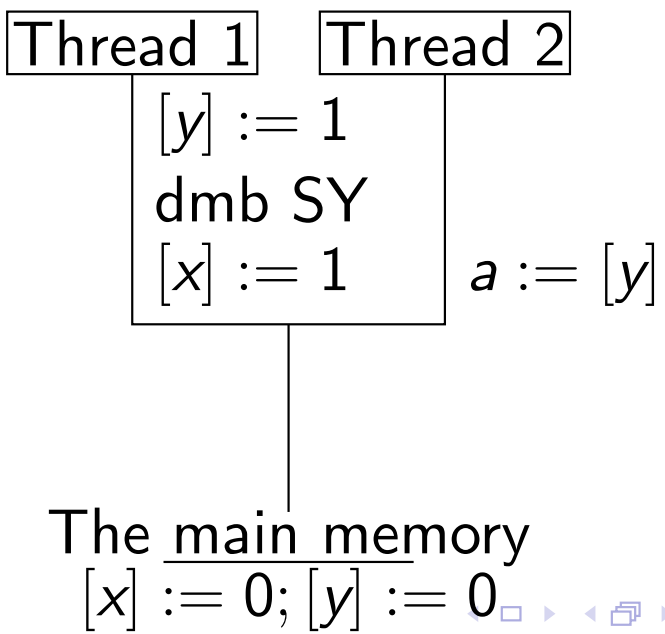
Example 2

$$
\begin{array}{l}
[x] := 0; [y] := 0 \\
\hline
[x] := 1; \quad \| \quad a := [y]; \\
\text{dmb SY}; \quad \| \quad \text{dmb LD}; \\
[y] := 1 \quad \| \quad b := [x]
\end{array}
$$

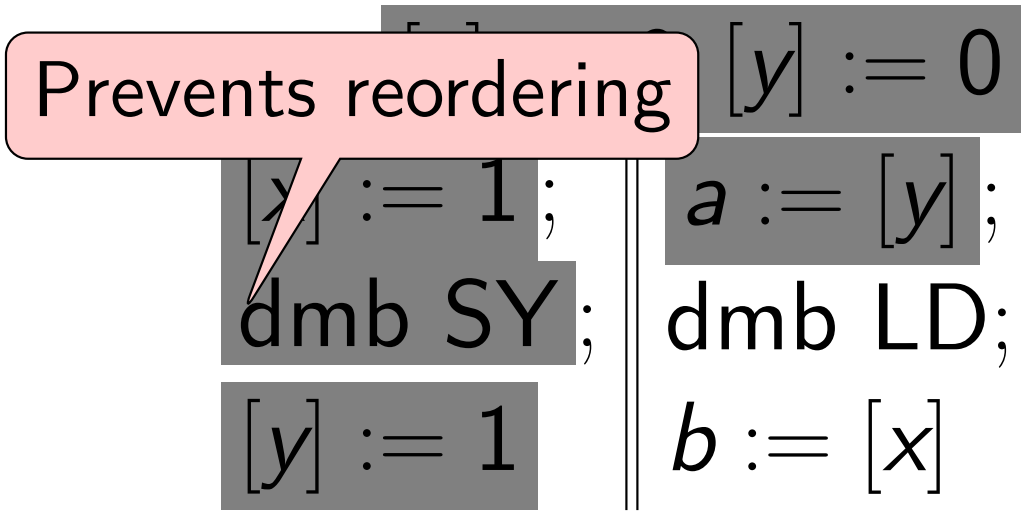


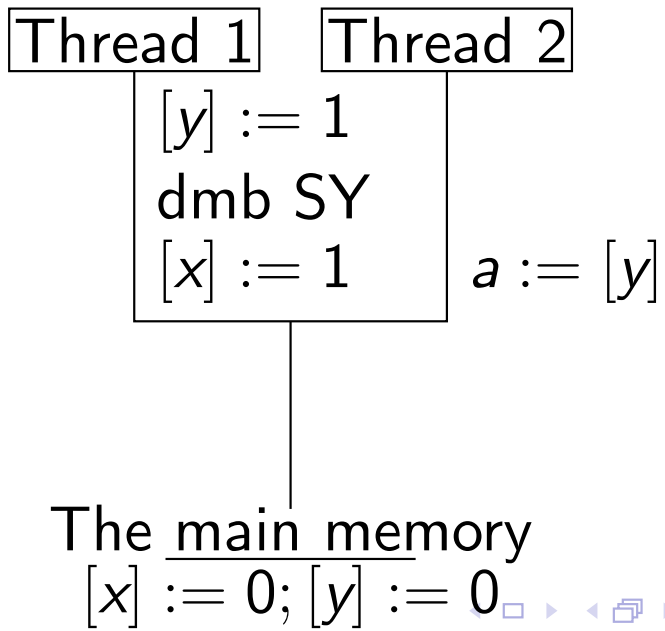
Example 2

$$
\begin{array}{l|l}
\multicolumn{2}{c}{[x] := 0; [y] := 0} \\
[x] := 1; & a := [y]; \\
\text{dmb SY}; & \text{dmb LD}; \\
[y] := 1 & b := [x]
\end{array}
$$

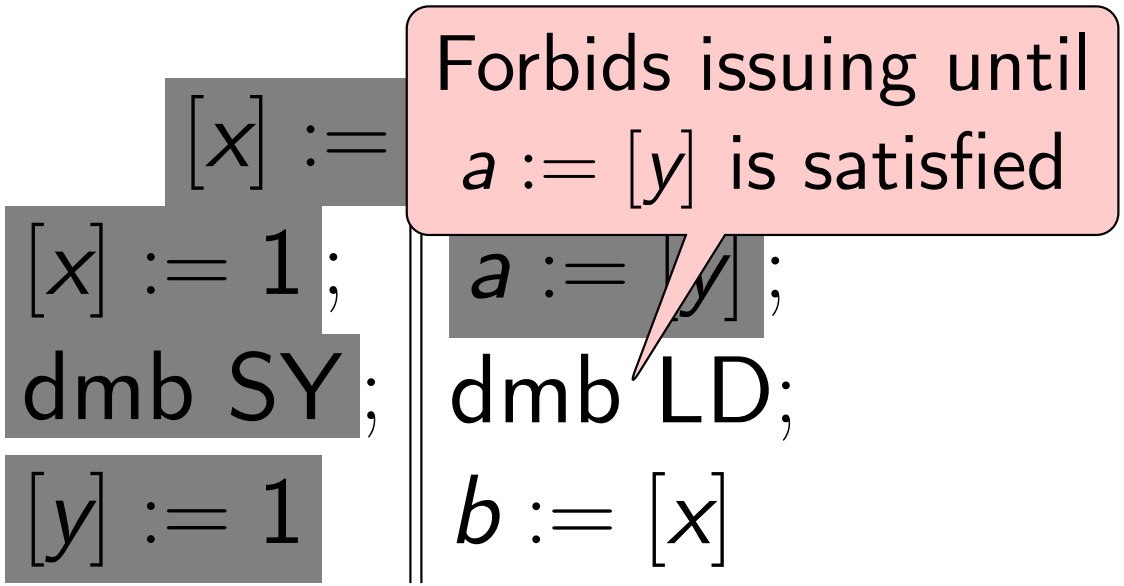


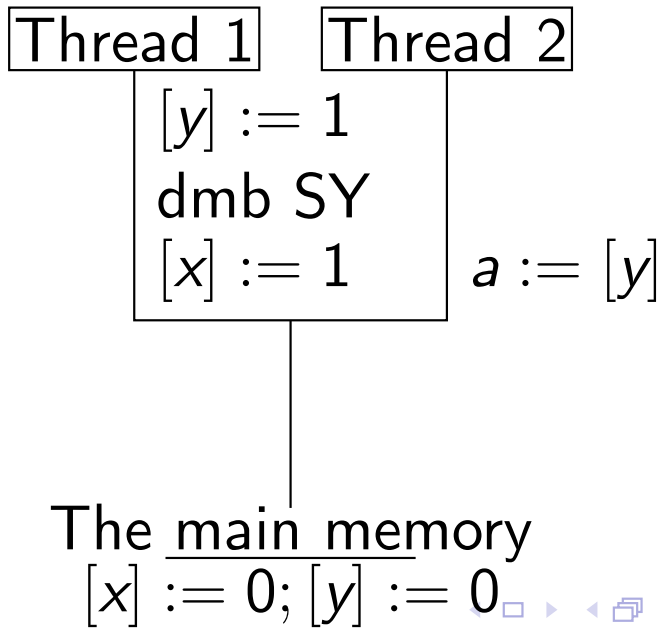
Example 2


Prevents reordering
[y] := 0
[x] := 1;
a := [y];
dmb SY;
dmb LD;
[y] := 1
b := [x]


Thread 1
Thread 2
[y] := 1
dmb SY
[x] := 1
a := [y]
The main memory
[x] := 0; [y] := 0

Example 2


Forbids issuing until
$a := [y]$ is satisfied
$[x] :=$
$[x] := 1$;
dmb SY;
$[y] := 1$
$a :=$ $[y]$;
dmb LD;
$b := [x]$


Thread 1
Thread 2
$[y] := 1$
dmb SY
$[x] := 1$
$a := [y]$
The main memory
$[x] := 0; [y] := 0$

Example 3

$$
\begin{array}{lll}
 & [x] := 0; [y] := 0 & \\
a := [x]; & \| \; b := [x]; & \| \; c := [y]; \\
[x] := 1 & \; [y] := b & \; [x] := c
\end{array}
$$

Example 3

$$
\begin{array}{lll}
 & [x] := 0; [y] := 0 & \\
a := [x]; \ // \ 1 \ \| & b := [x]; & \| \ c := [y]; \\
[x] := 1 & [y] := b & \| \ [x] := c
\end{array}
$$

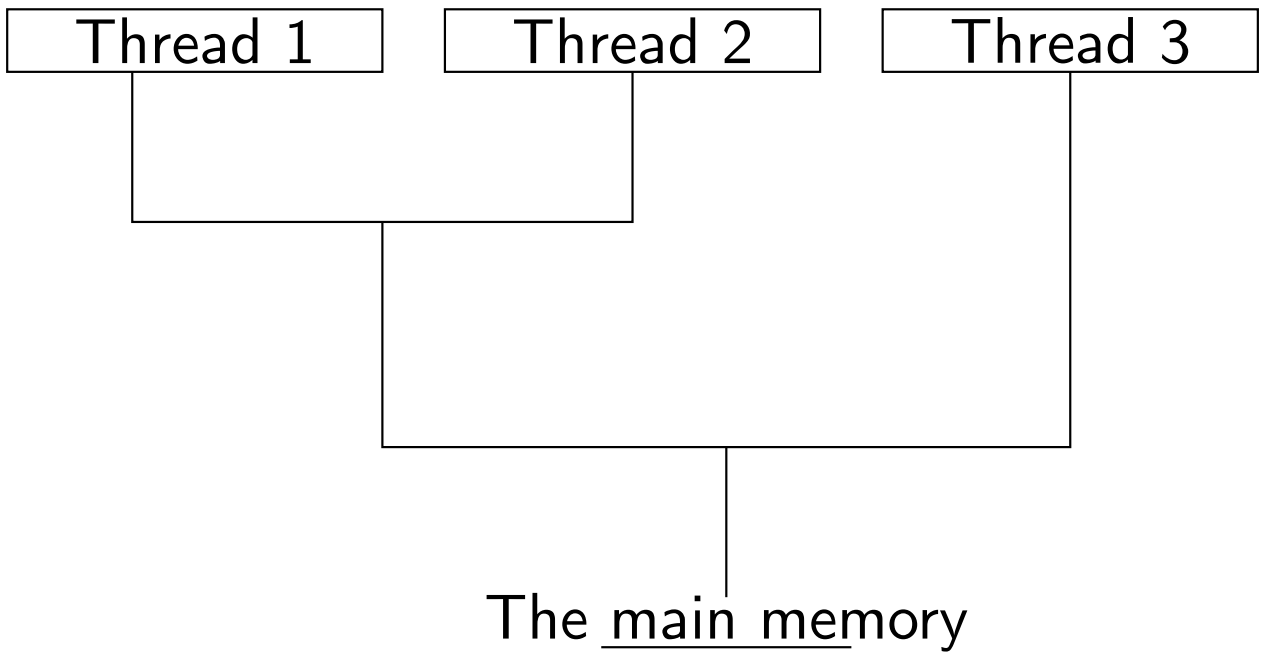
Example 3

$$
\begin{array}{lll}
 & [x] := 0; [y] := 0 & \\
a := [x]; & \| \; b := [x]; & \| \; c := [y]; \\
[x] := 1 & \; [y] := b & \; [x] := c
\end{array}
$$

Example 3

$$[x] := 0; [y] := 0$$

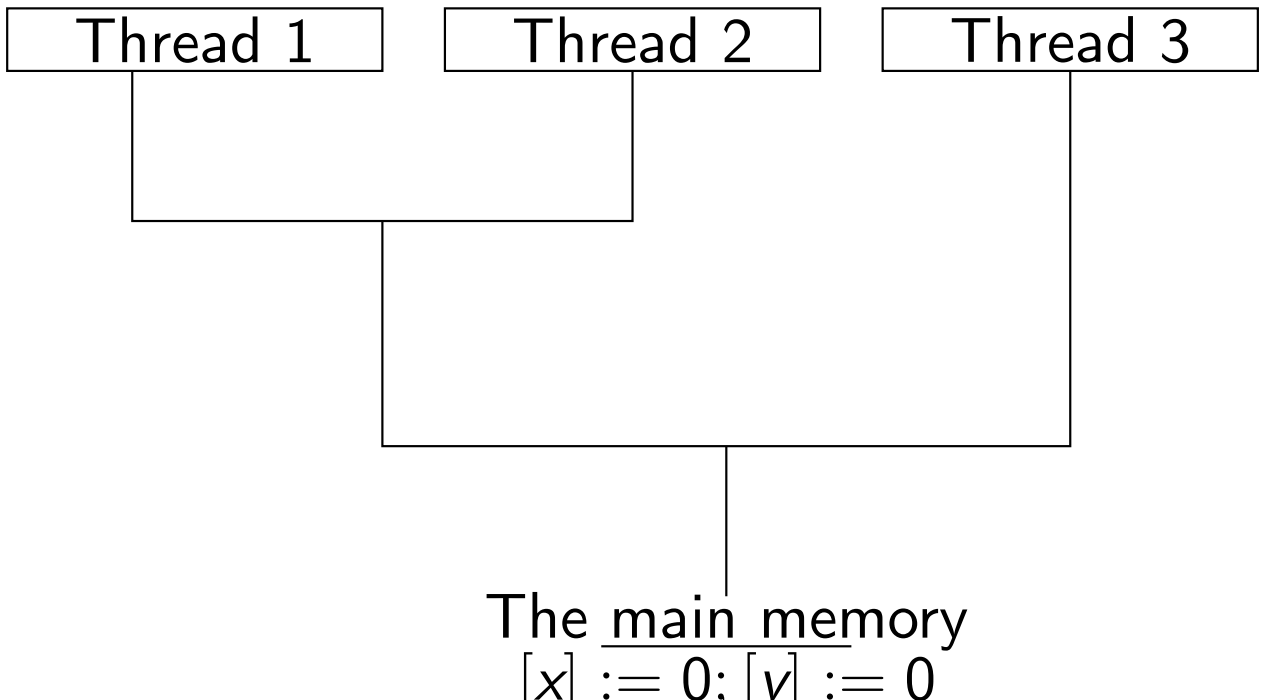
$$a := [x]; \quad [x] := 1 \quad \Big\| \quad b := [x]; \quad [y] := b \quad \Big\| \quad c := [y]; \quad [x] := c$$



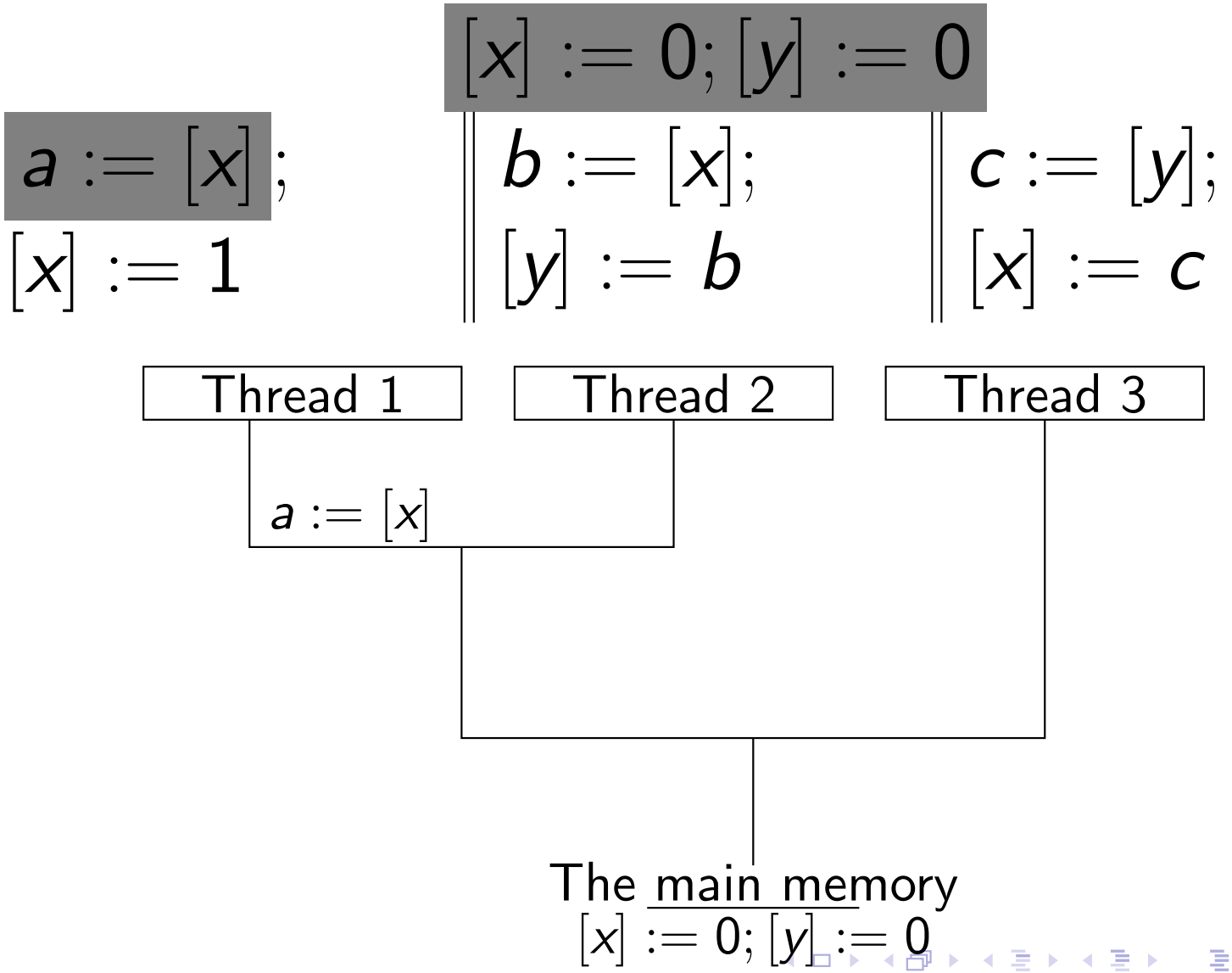
Example 3

$[x] := 0; [y] := 0$

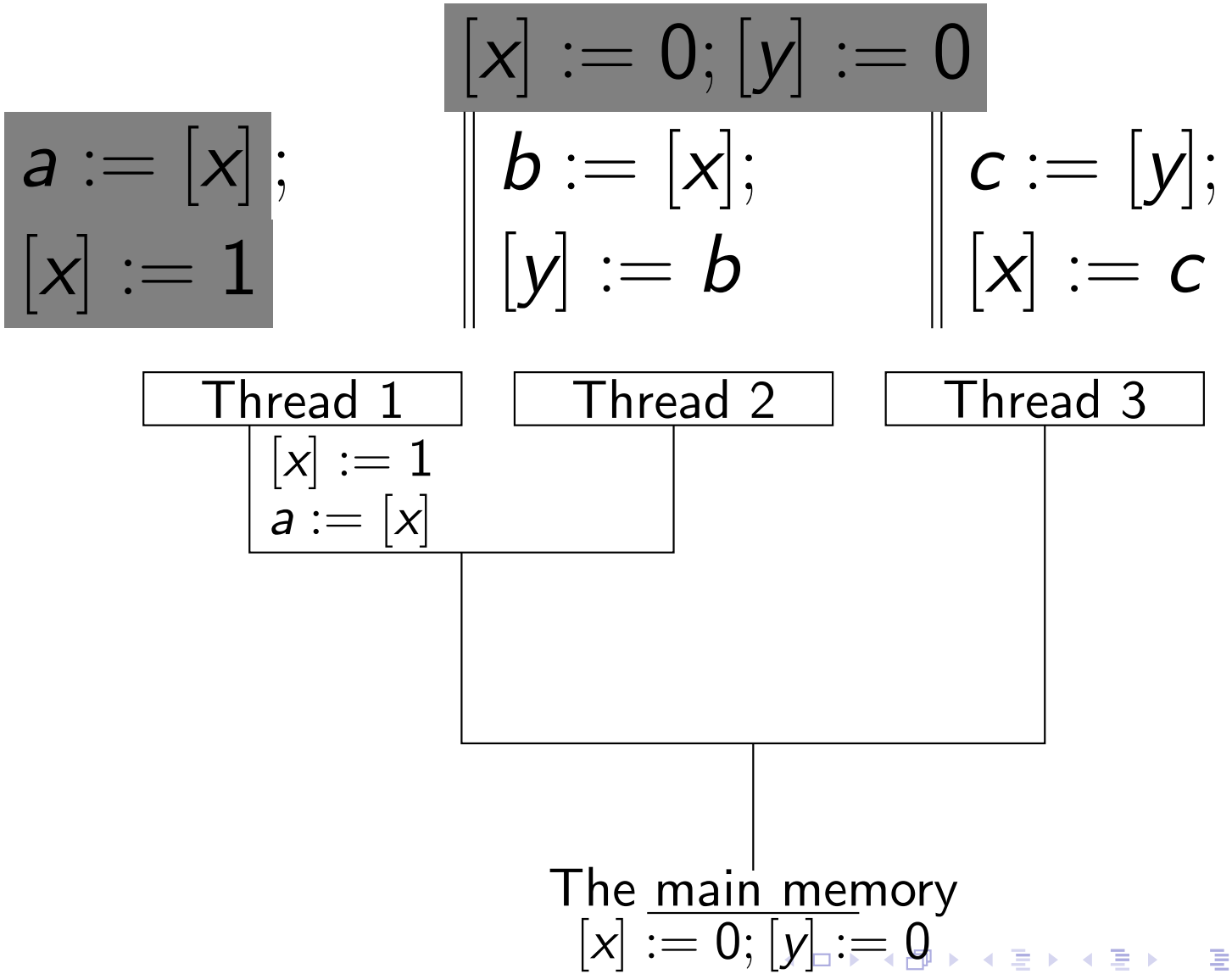
$a := [x];$ $[x] := 1$ || $b := [x];$ $[y] := b$ || $c := [y];$ $[x] := c$



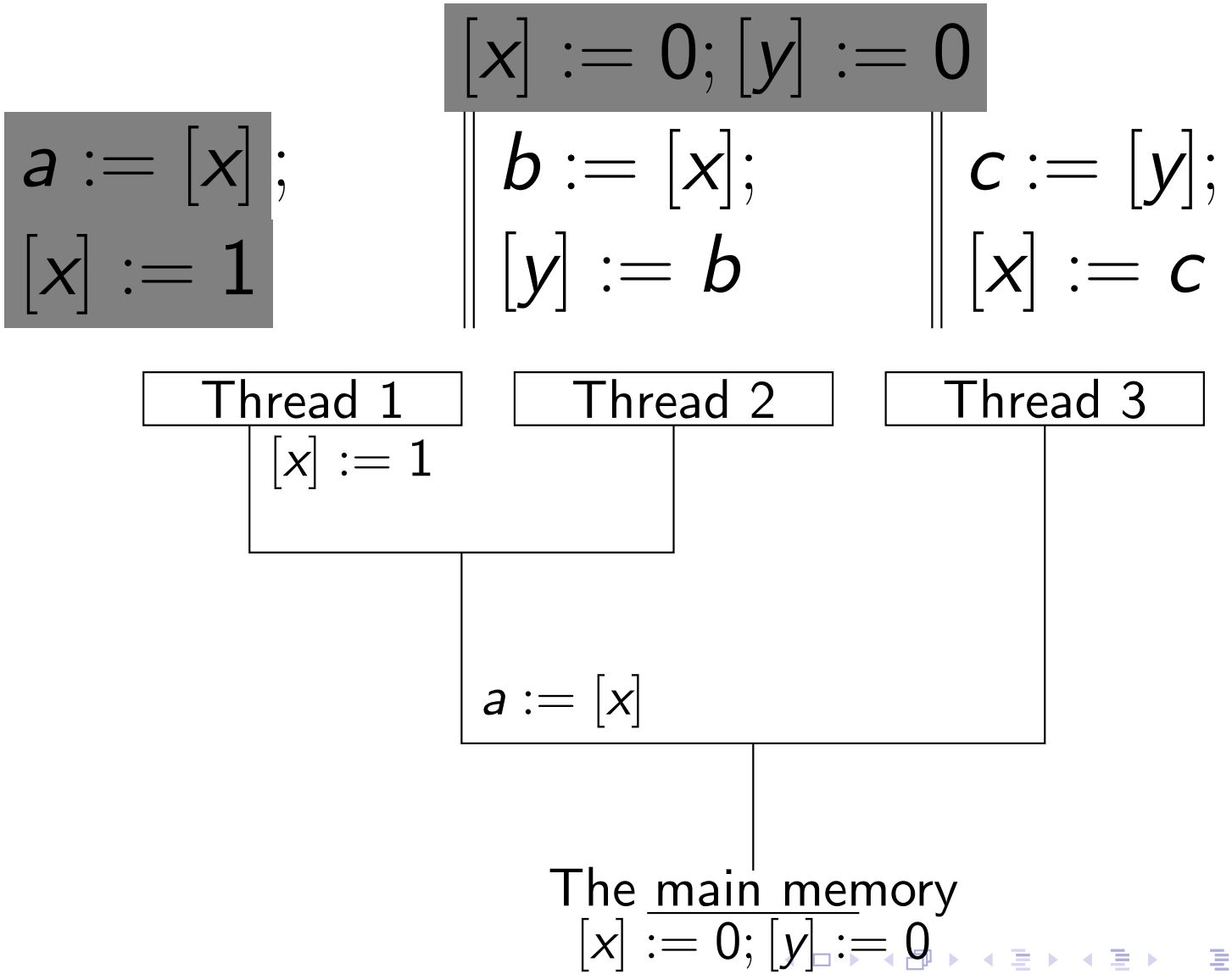
Example 3


[x] := 0; [y] := 0
a := [x];
[x] := 1
b := [x];
[y] := b
c := [y];
[x] := c
Thread 1
Thread 2
Thread 3
a := [x]
The main memory
[x] := 0; [y] := 0

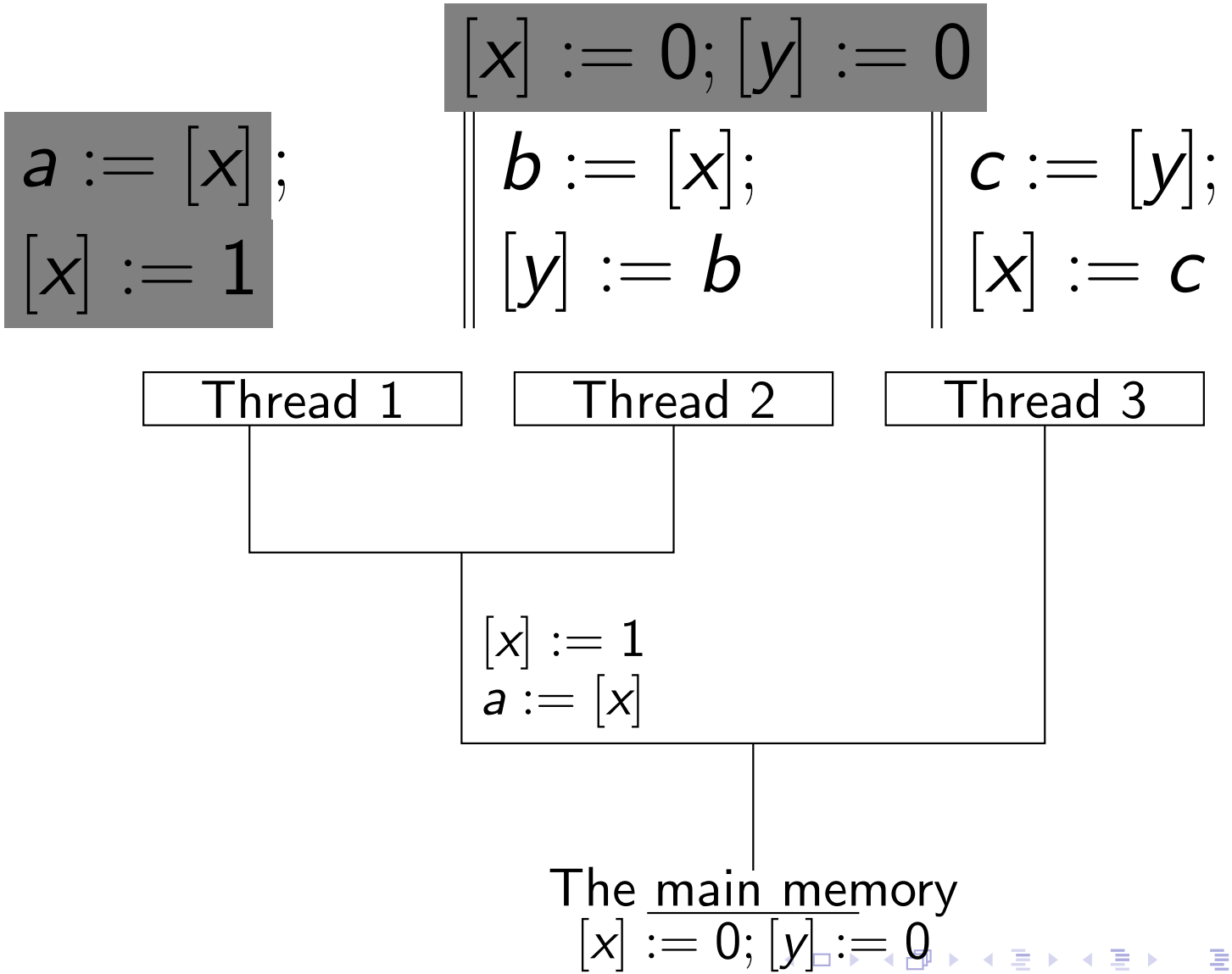
Example 3


[x] := 0; [y] := 0
a := [x];
[x] := 1
b := [x];
[y] := b
c := [y];
[x] := c
Thread 1
Thread 2
Thread 3
[x] := 1
a := [x]
The main memory
[x] := 0; [y] := 0

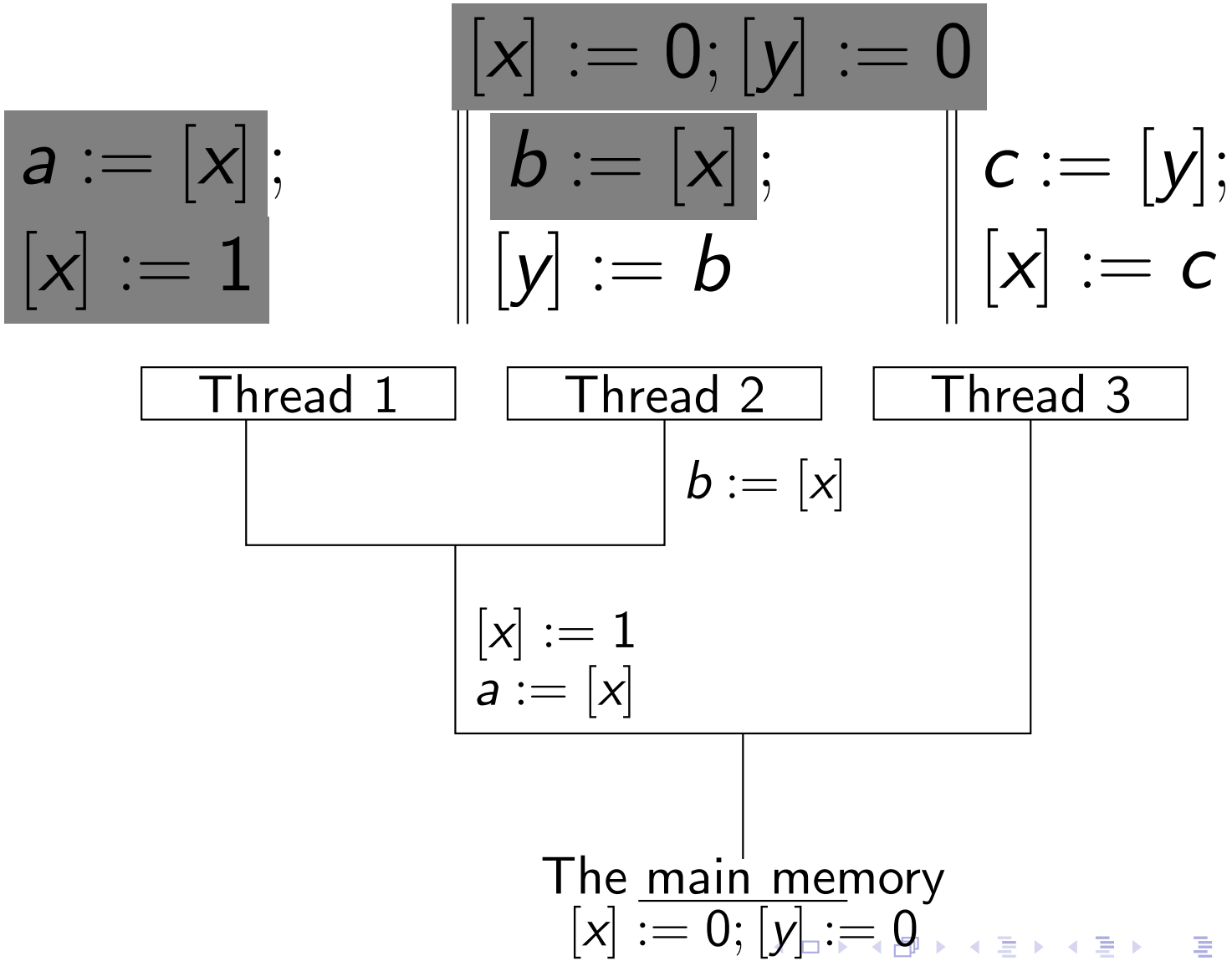
Example 3


[x] := 0; [y] := 0
a := [x];
[x] := 1
b := [x];
[y] := b
c := [y];
[x] := c
Thread 1
Thread 2
Thread 3
[x] := 1
a := [x]
The main memory
[x] := 0; [y] := 0

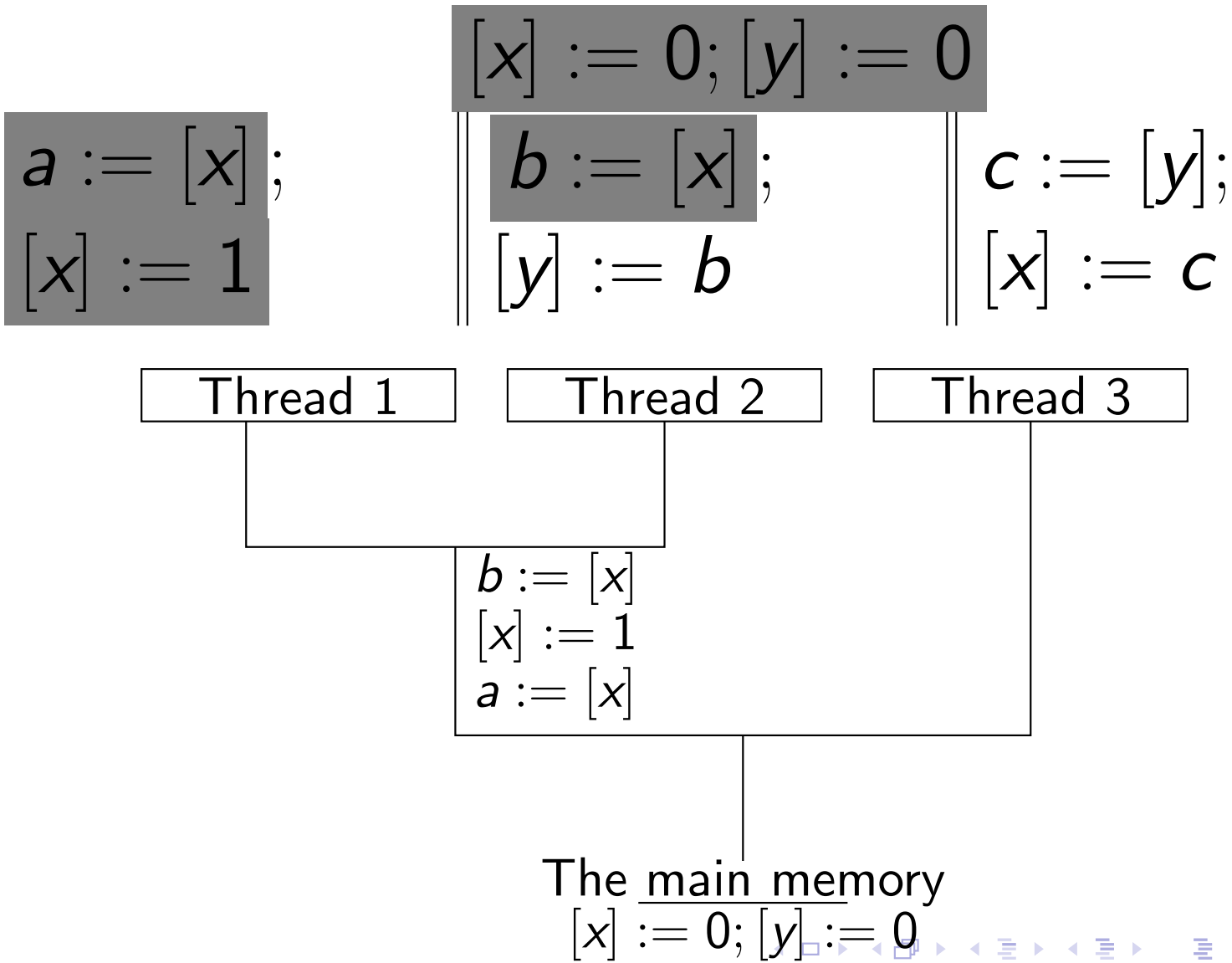
Example 3


[x] := 0; [y] := 0
a := [x]; [x] := 1
b := [x]; [y] := b
c := [y]; [x] := c
Thread 1
Thread 2
Thread 3
[x] := 1
a := [x]
The main memory
[x] := 0; [y] := 0

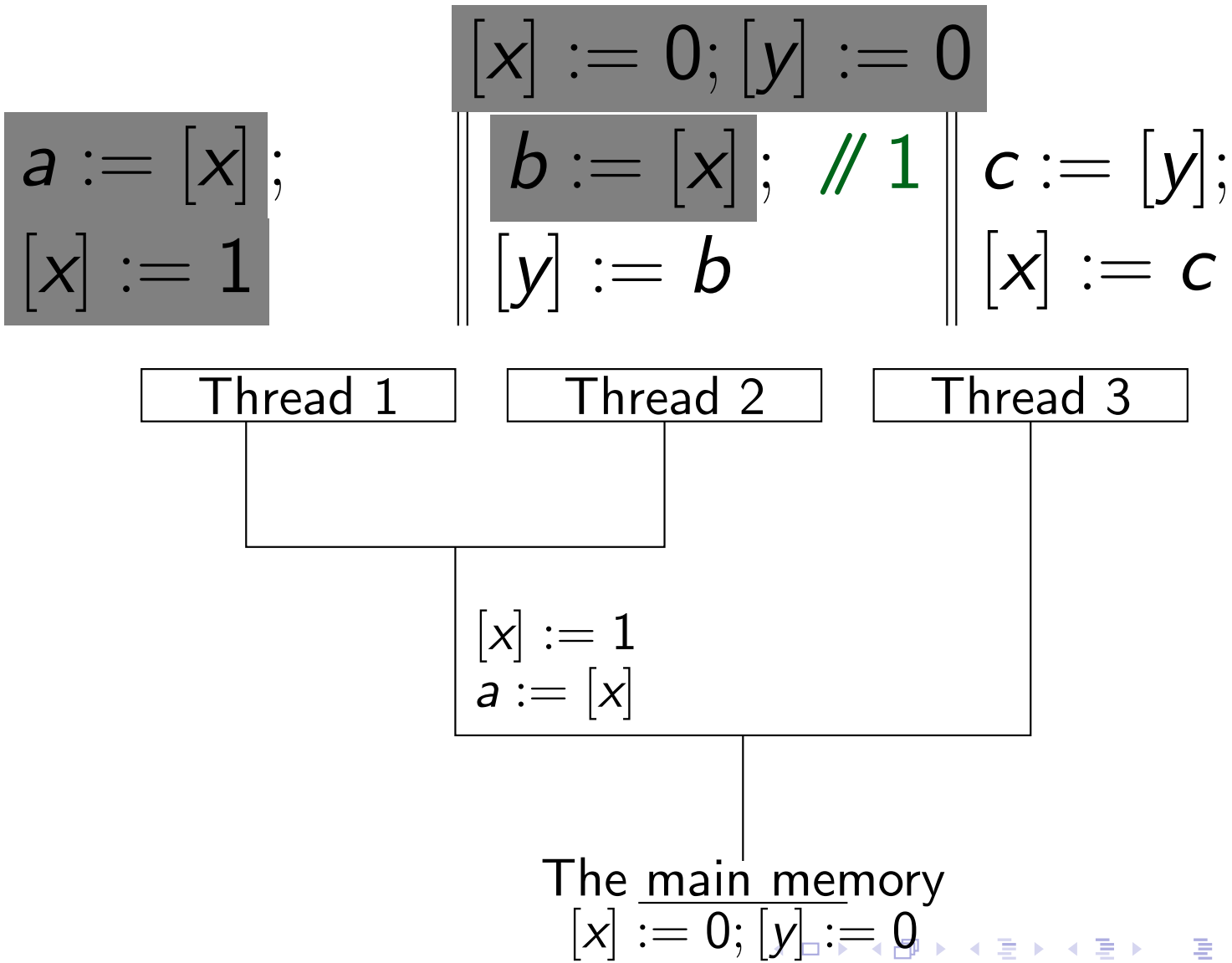
Example 3


[x] := 0; [y] := 0
a := [x]; [x] := 1
b := [x]; [y] := b
c := [y]; [x] := c
Thread 1
Thread 2
Thread 3
b := [x]
[x] := 1
a := [x]
The main memory
[x] := 0; [y] := 0

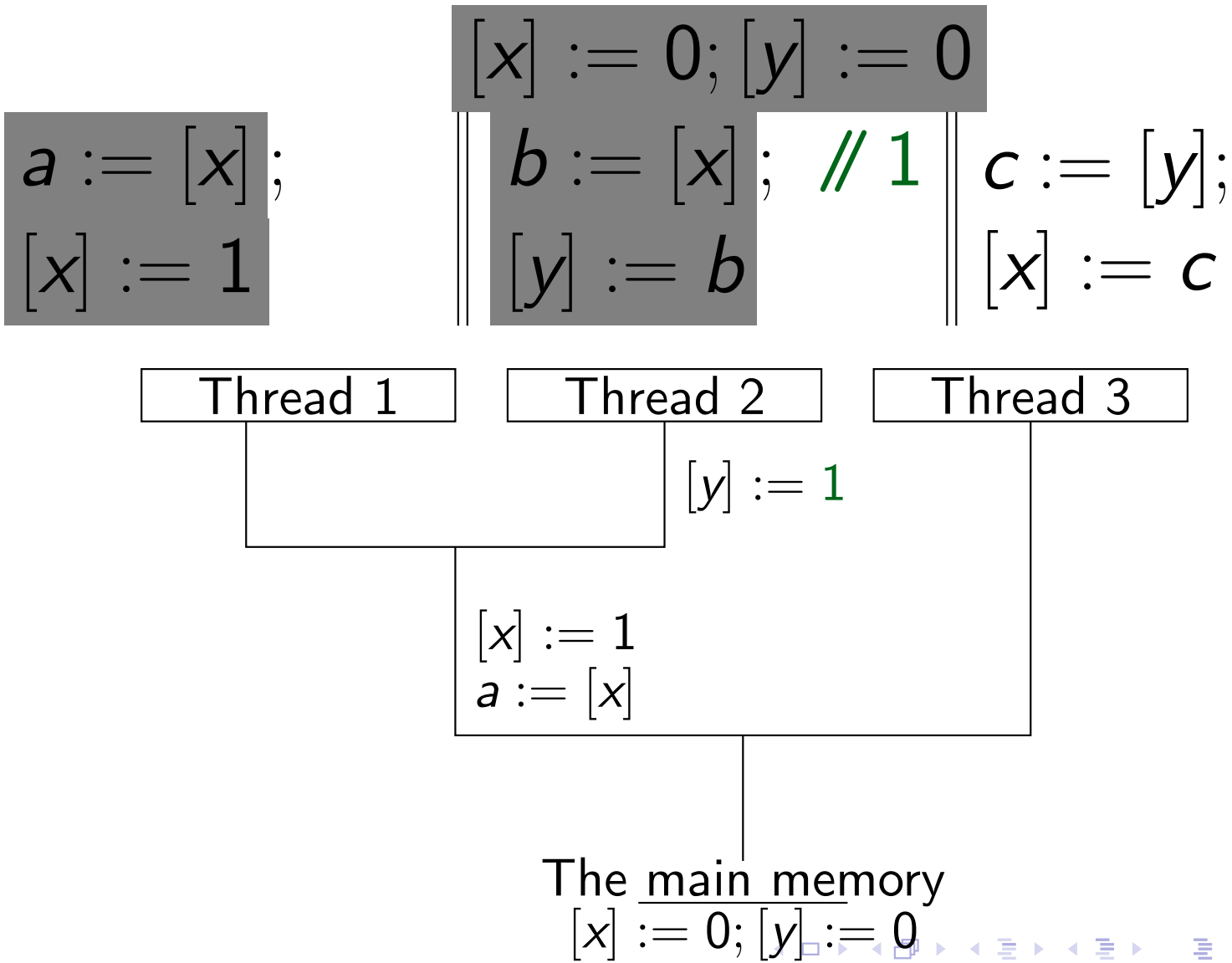
Example 3


[x] := 0; [y] := 0
a := [x];
[x] := 1
b := [x];
[y] := b
c := [y];
[x] := c
Thread 1
Thread 2
Thread 3
b := [x]
[x] := 1
a := [x]
The main memory
[x] := 0; [y] := 0

Example 3


[x] := 0; [y] := 0
a := [x];
[x] := 1
b := [x]; // 1
[y] := b
c := [y];
[x] := c
Thread 1
Thread 2
Thread 3
[x] := 1
a := [x]
The main memory
[x] := 0; [y] := 0

Example 3

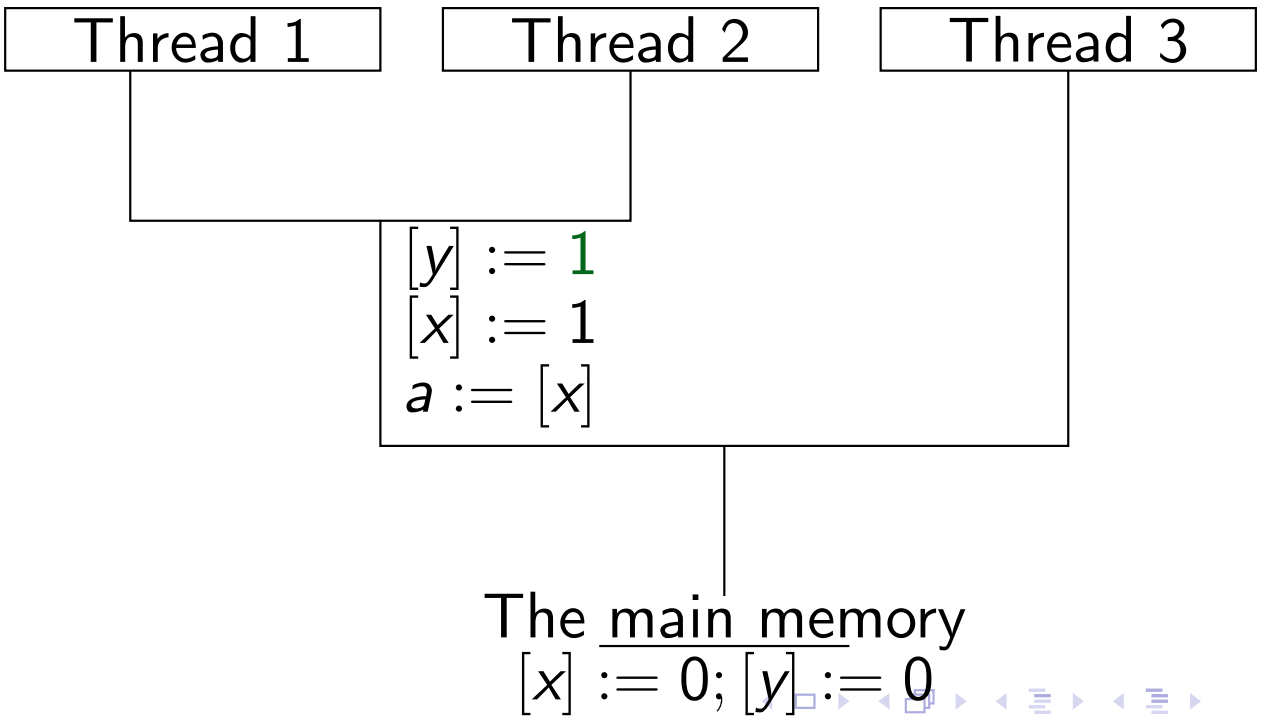

[x] := 0; [y] := 0
a := [x]; [x] := 1
b := [x]; // 1 [y] := b
c := [y]; [x] := c
Thread 1
Thread 2
Thread 3
[y] := 1
[x] := 1
a := [x]
The main memory
[x] := 0; [y] := 0

Example 3

$[x] := 0; [y] := 0$

$a := [x]; \quad \| \quad b := [x]; \; // \, 1 \quad \| \quad c := [y];$

$[x] := 1 \qquad\quad [y] := b \qquad\quad [x] := c$

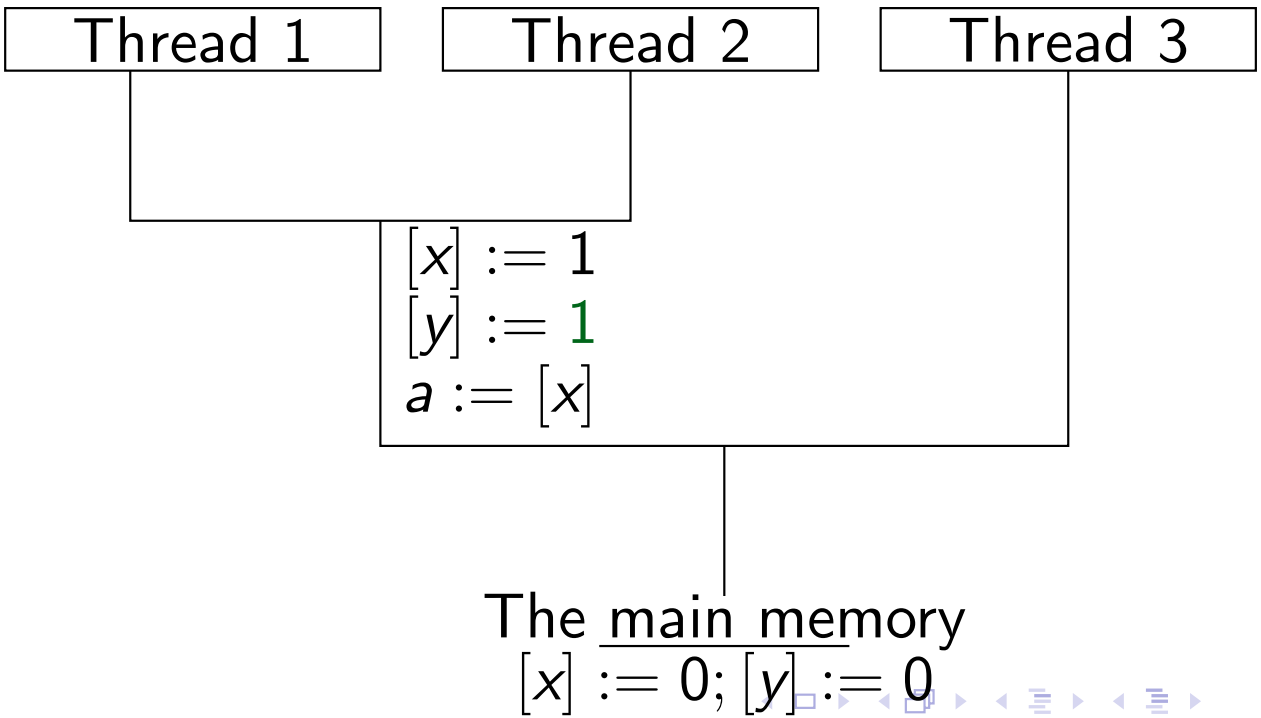


Example 3

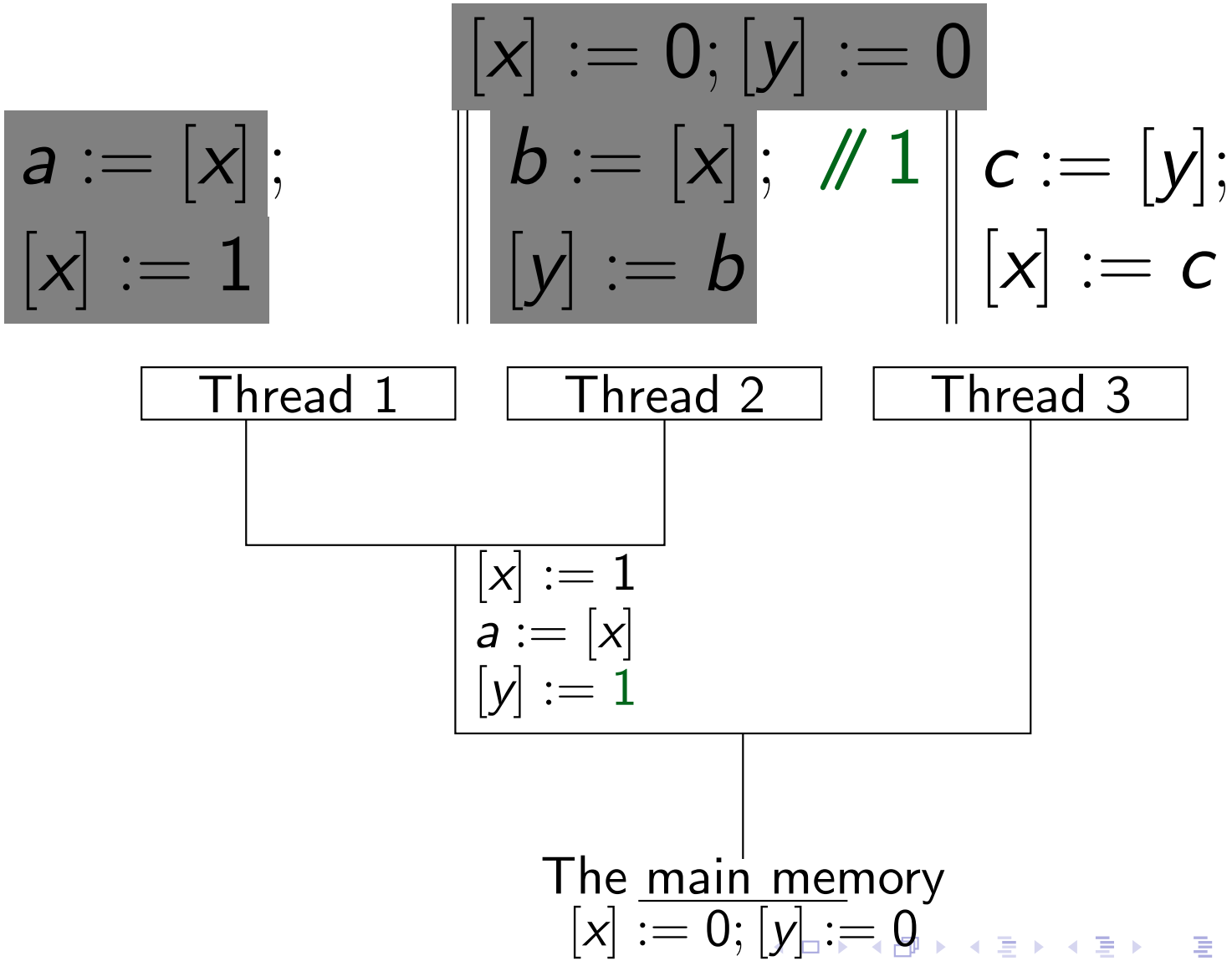
$[x] := 0; [y] := 0$

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

$[x] := 1 \qquad\quad [y] := b \qquad\quad [x] := c$



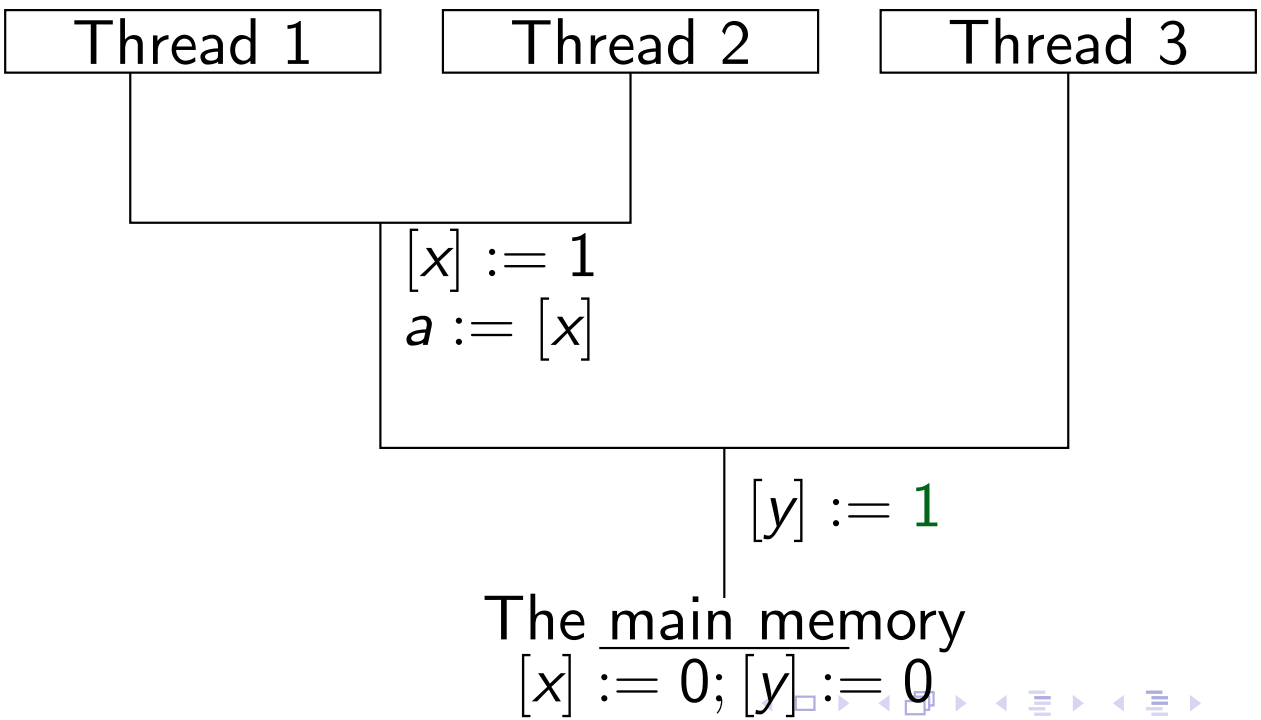
Example 3


[x] := 0; [y] := 0
a := [x]; [x] := 1
b := [x]; // 1
[y] := b
c := [y];
[x] := c
Thread 1
Thread 2
Thread 3
[x] := 1
a := [x]
[y] := 1
The main memory
[x] := 0; [y] := 0

Example 3

$[x] := 0; [y] := 0$

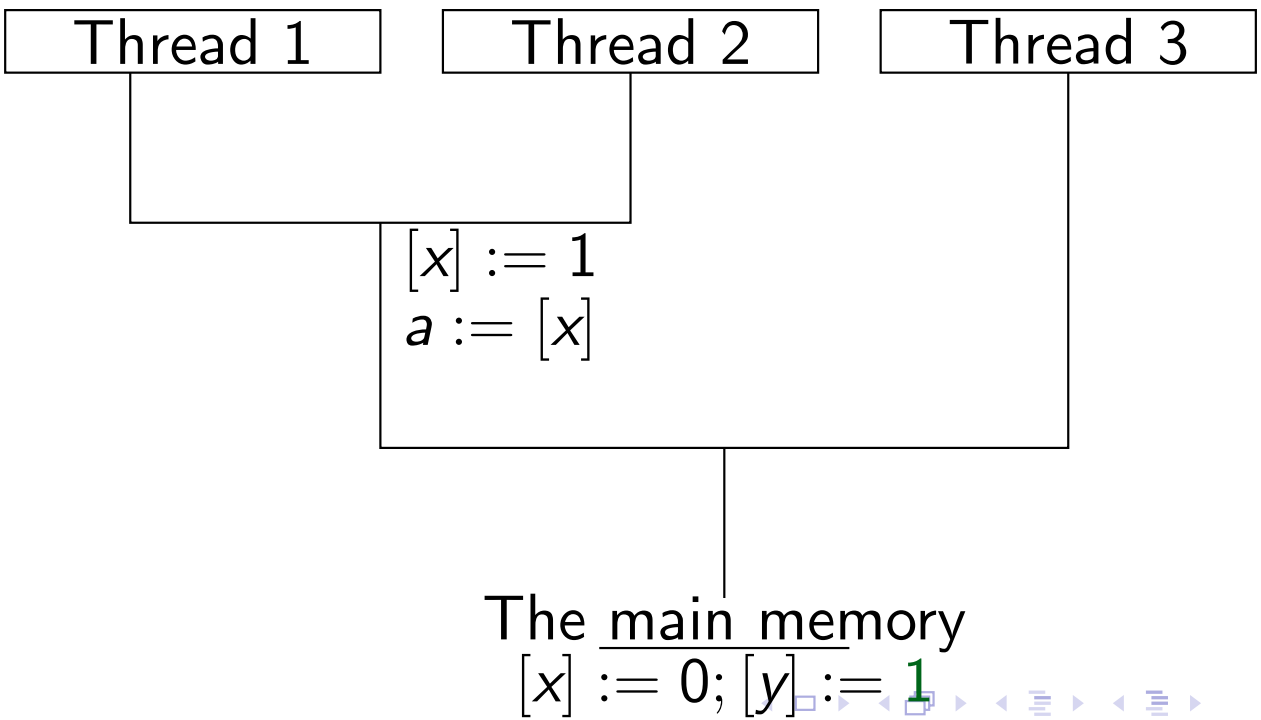
$a := [x]; \quad [x] := 1 \quad \| \quad b := [x]; \quad [y] := b \quad // 1 \quad \| \quad c := [y]; \quad [x] := c$



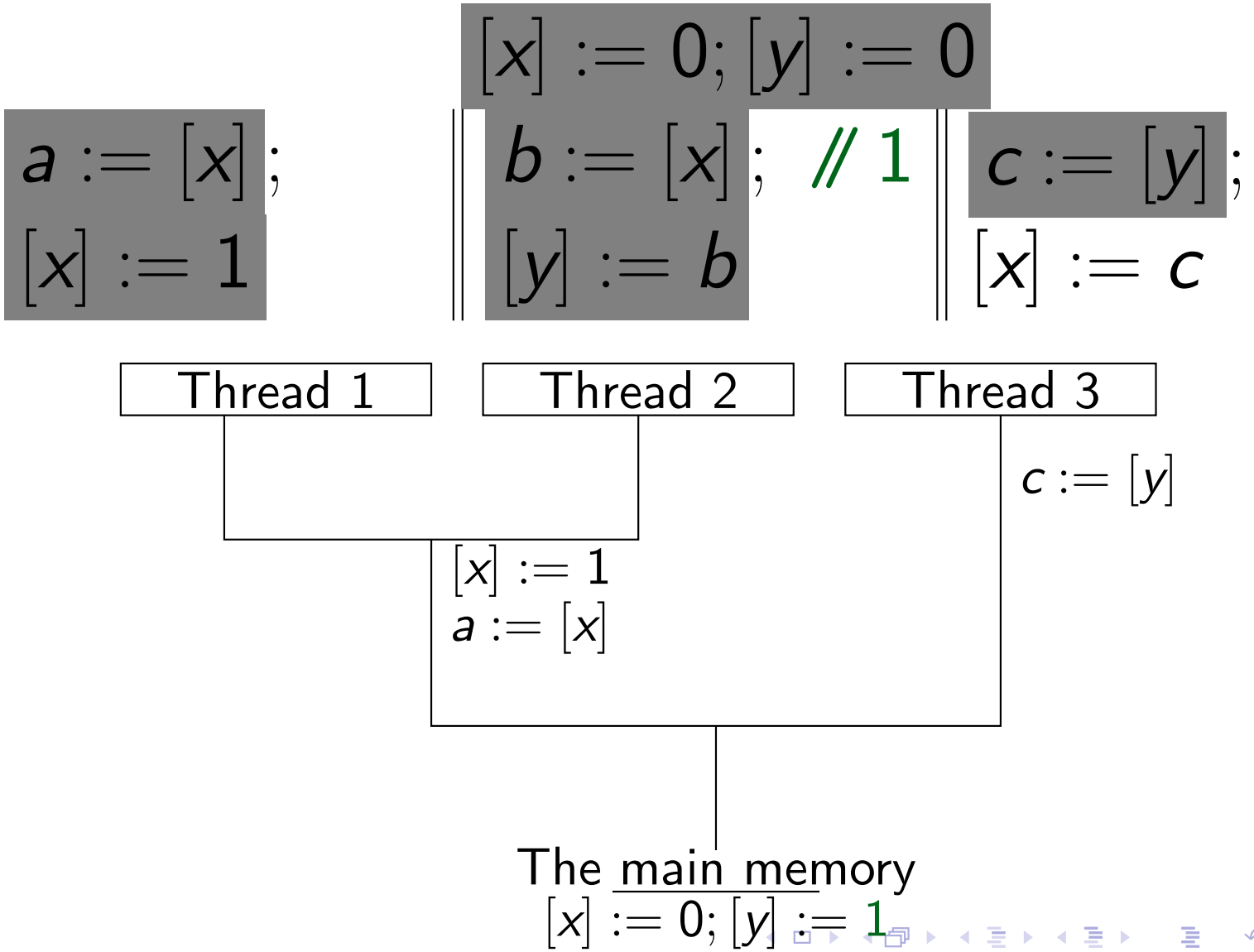
Example 3

$[x] := 0; [y] := 0$

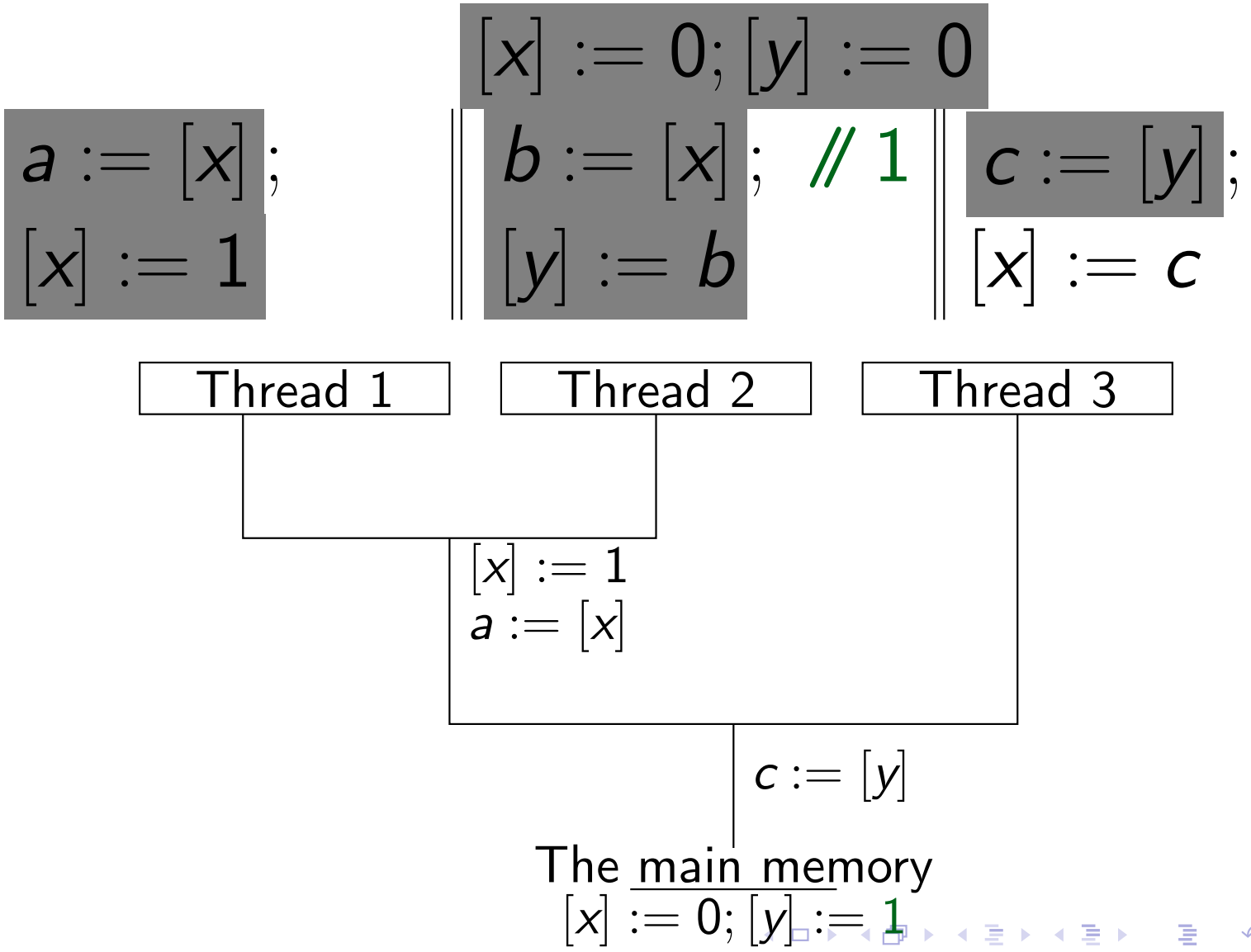
$a := [x]; \quad [x] := 1 \quad \| \quad b := [x]; \quad [y] := b \quad // 1 \quad \| \quad c := [y]; \quad [x] := c$



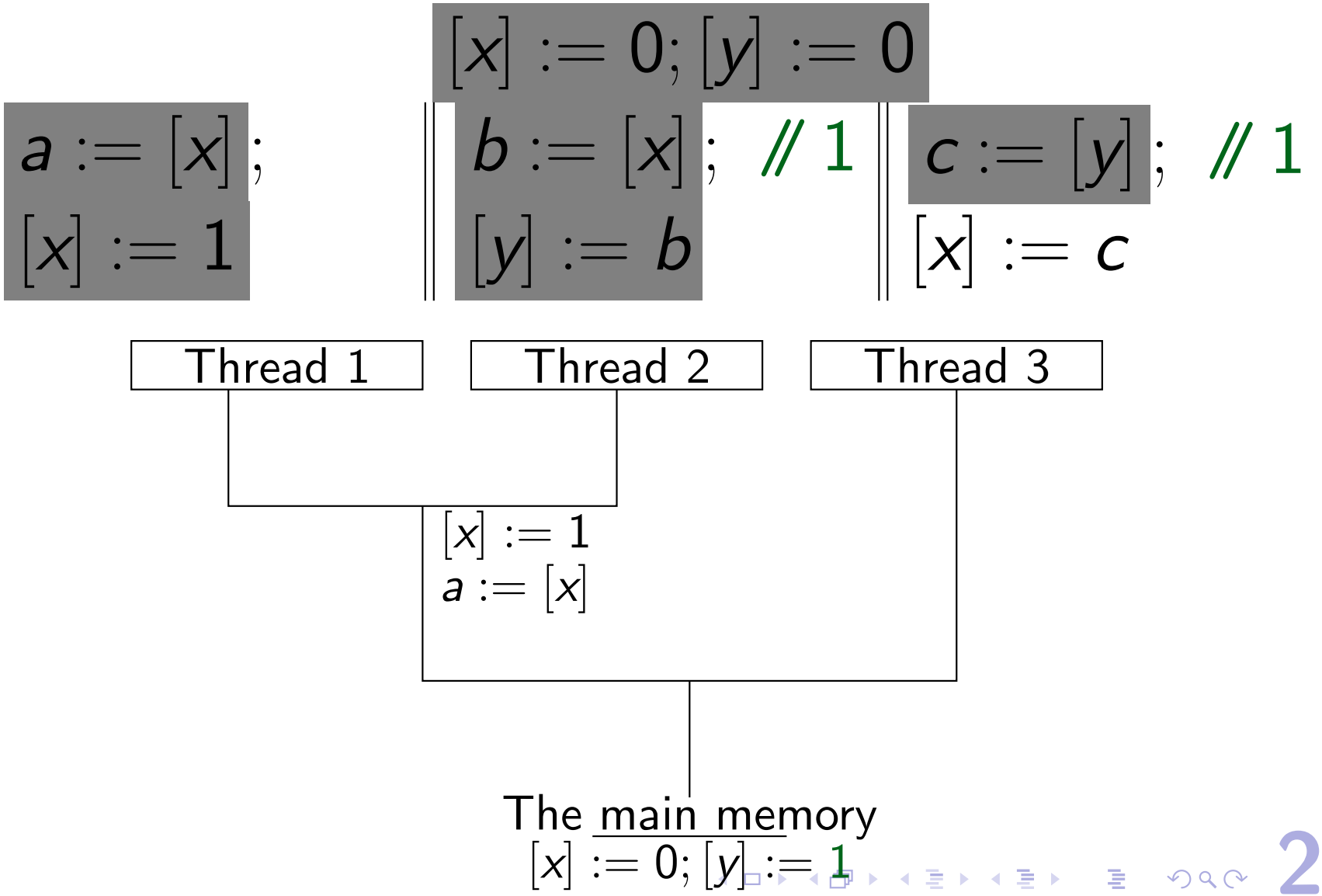
Example 3


[x] := 0; [y] := 0
a := [x];
[x] := 1
b := [x]; // 1
[y] := b
c := [y];
[x] := c
Thread 1
Thread 2
Thread 3
c := [y]
[x] := 1
a := [x]
The main memory
[x] := 0; [y] := 1

Example 3


[x] := 0; [y] := 0
a := [x];
[x] := 1
b := [x]; // 1
[y] := b
c := [y];
[x] := c
Thread 1
Thread 2
Thread 3
[x] := 1
a := [x]
c := [y]
The main memory
[x] := 0; [y] := 1

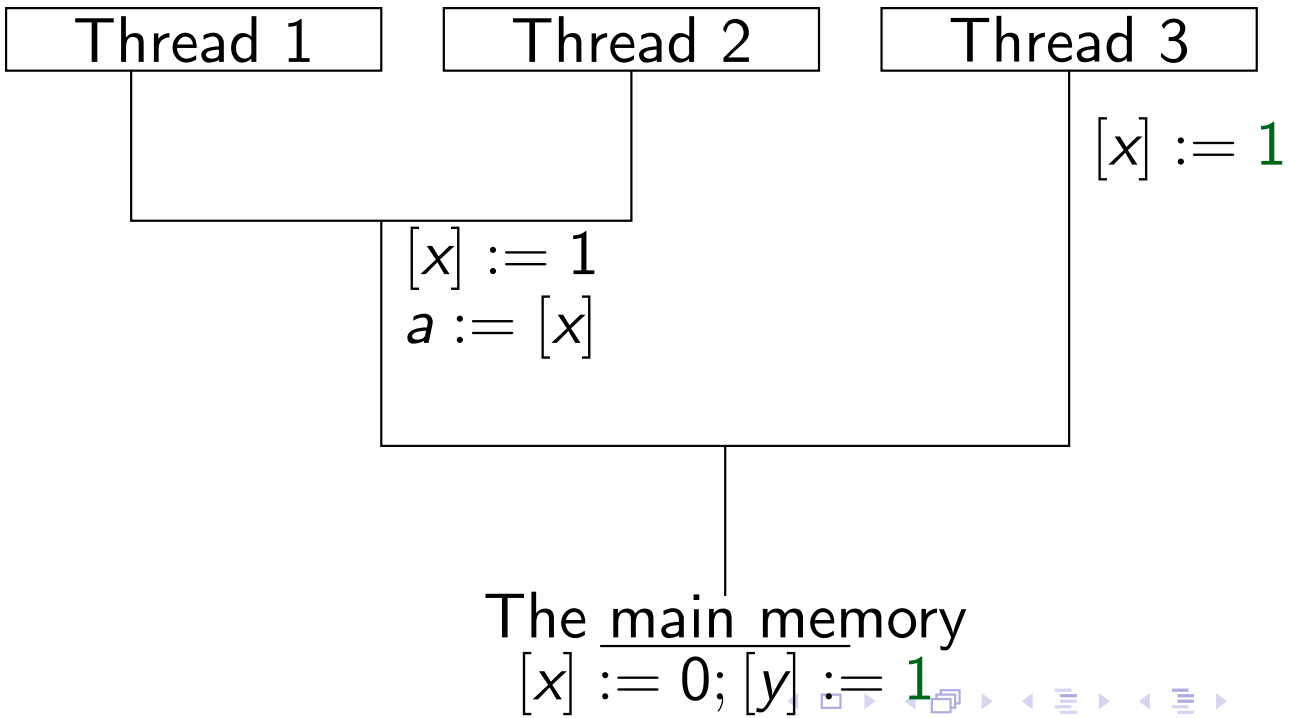
Example 3


[x] := 0; [y] := 0
a := [x]; [x] := 1
b := [x]; // 1 [y] := b
c := [y]; // 1 [x] := c
Thread 1
Thread 2
Thread 3
[x] := 1
a := [x]
The main memory
[x] := 0; [y] := 1

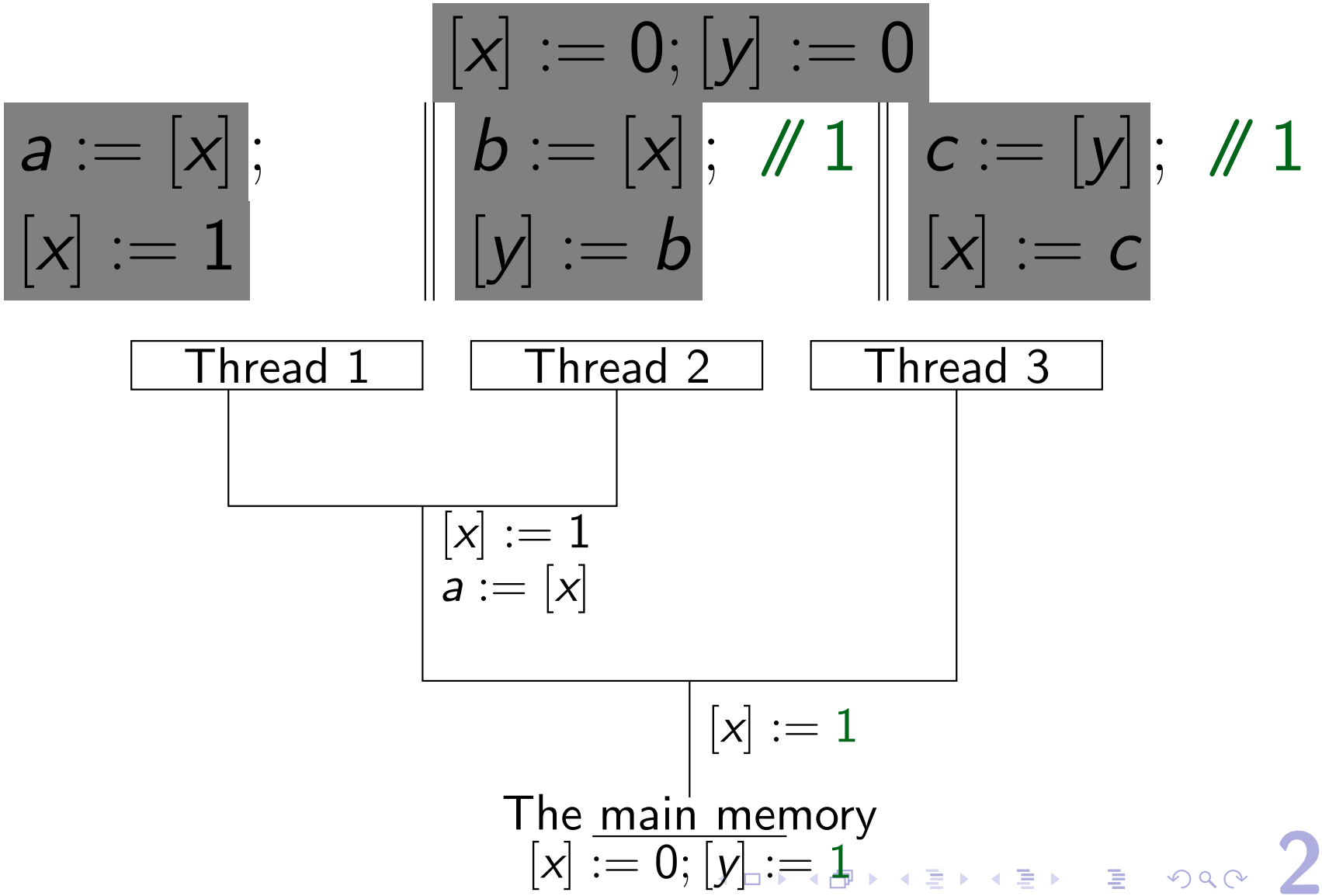
Example 3

$$[x] := 0; [y] := 0$$

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



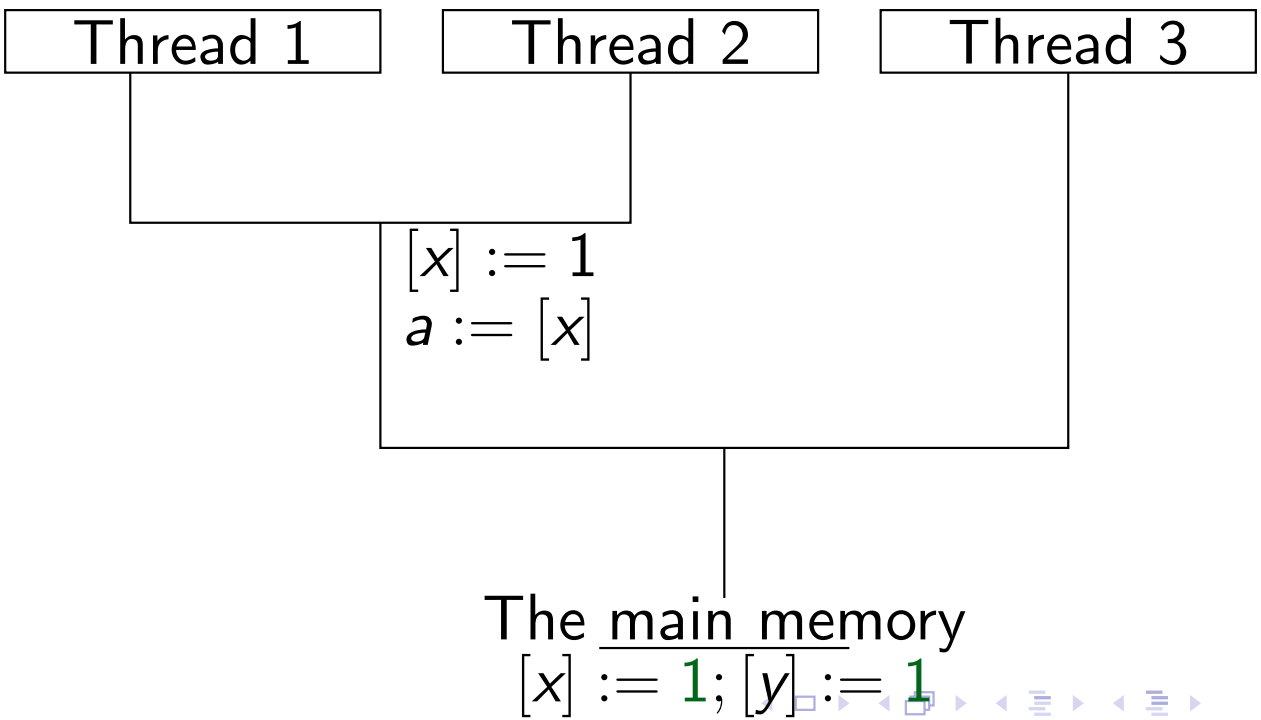
Example 3


[x] := 0; [y] := 0
a := [x]; [x] := 1
b := [x]; // 1
[y] := b
c := [y]; // 1
[x] := c
Thread 1
Thread 2
Thread 3
[x] := 1
a := [x]
[x] := 1
The main memory
[x] := 0; [y] := 1

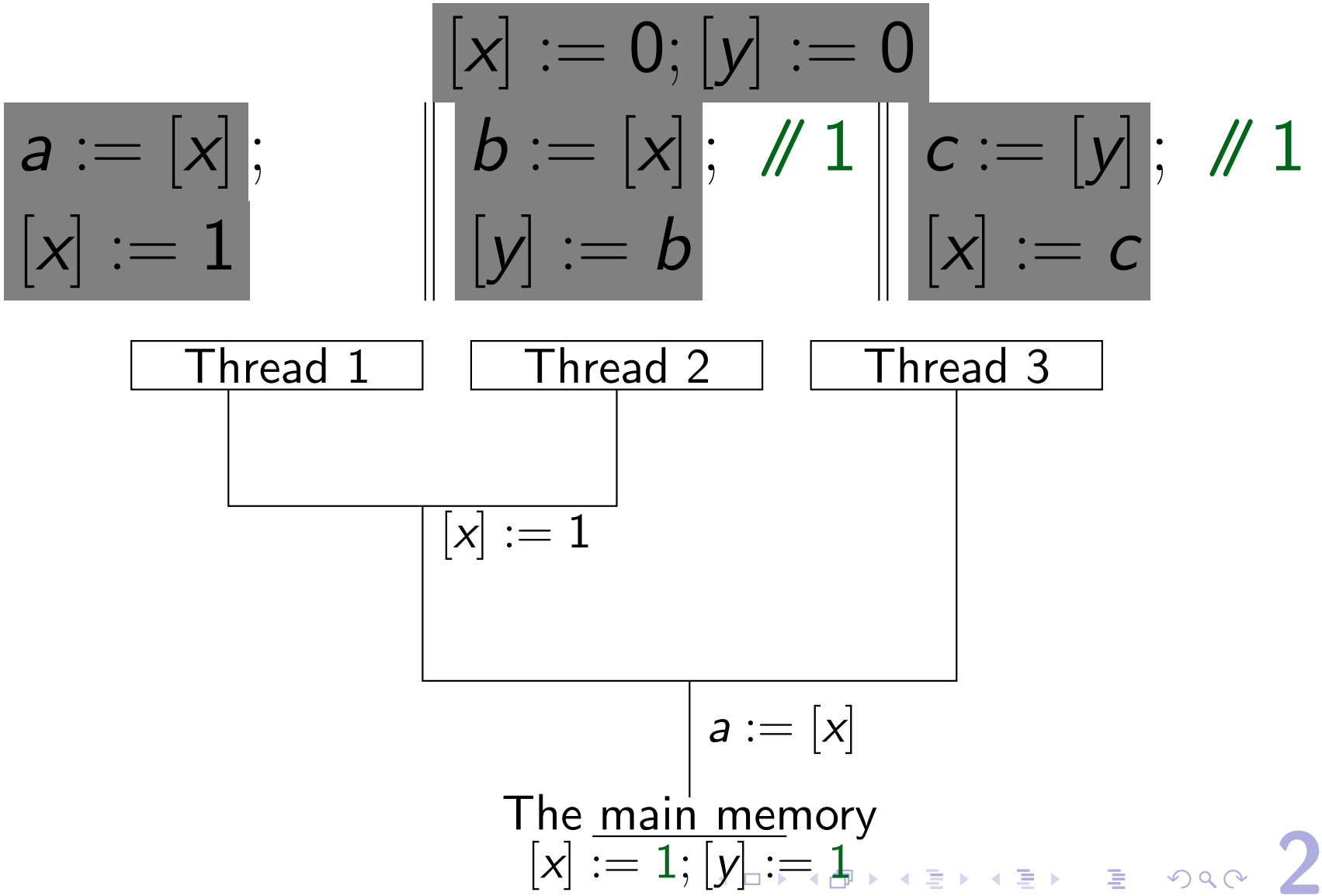
Example 3

$[x] := 0; [y] := 0$

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



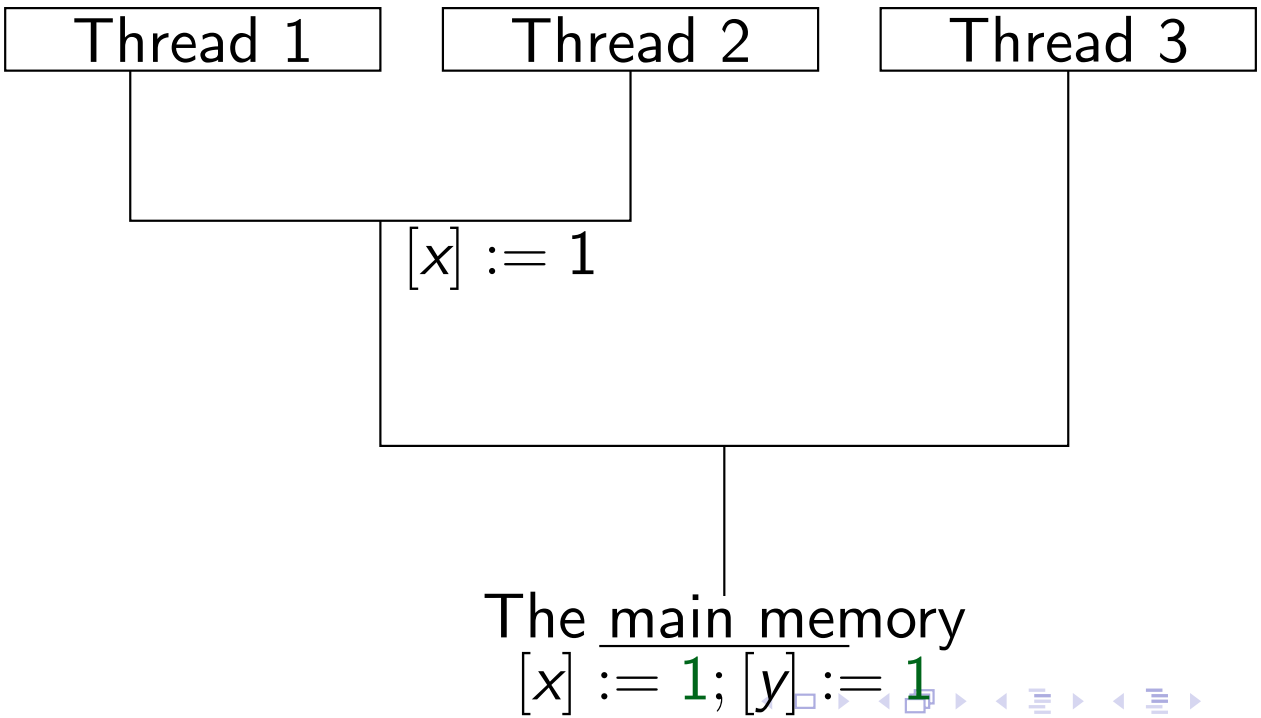
Example 3


[x] := 0; [y] := 0
a := [x]; [x] := 1
b := [x]; // 1 [y] := b
c := [y]; // 1 [x] := c
Thread 1
Thread 2
Thread 3
[x] := 1
a := [x]
The main memory
[x] := 1; [y] := 1

Example 3

$$[x] := 0; [y] := 0$$

$$a := [x]; \; // \, 1 \quad [x] := 1 \qquad \| \qquad b := [x]; \; // \, 1 \quad [y] := b \qquad \| \qquad c := [y]; \; // \, 1 \quad [x] := c$$

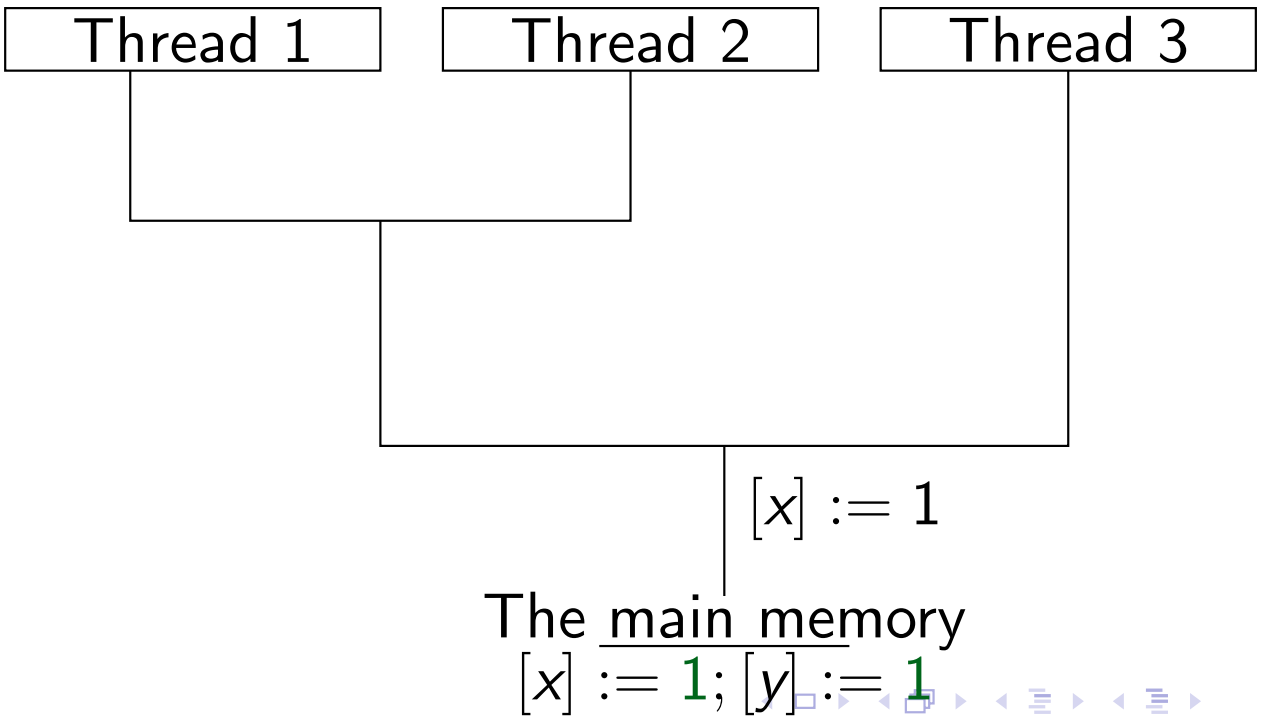


Example 3

$$[x] := 0; [y] := 0$$

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

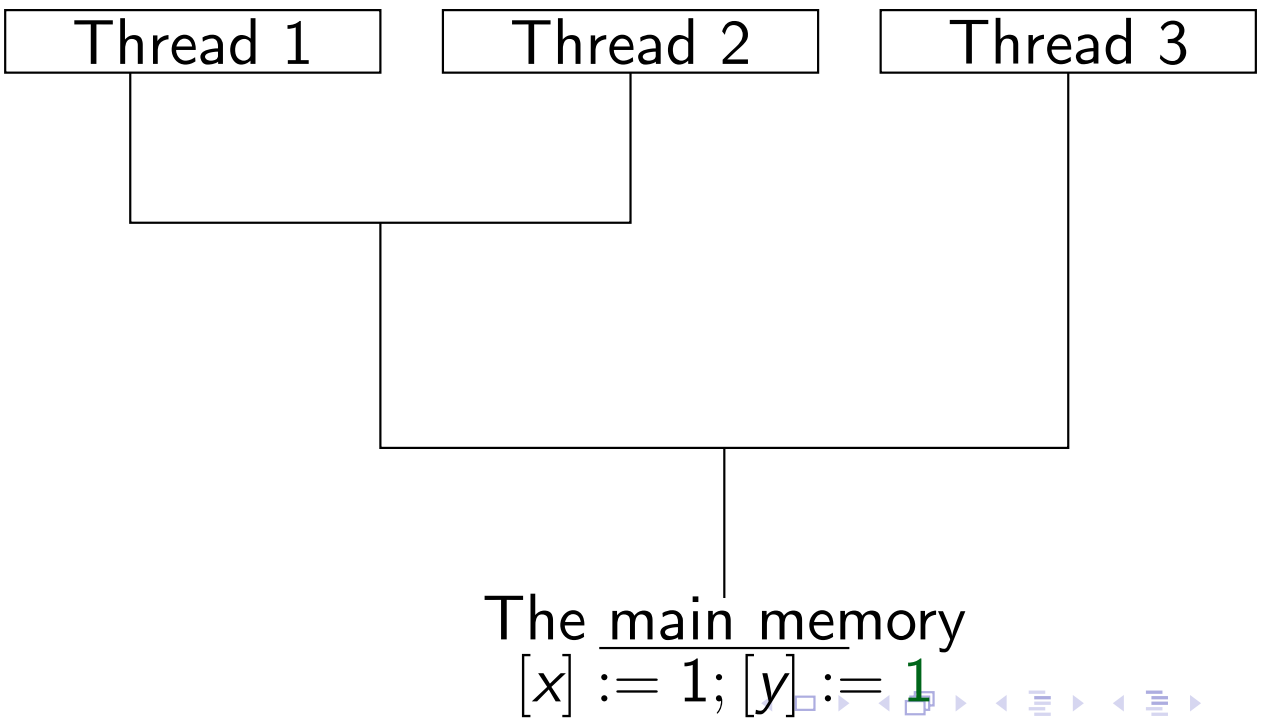
$$[x] := 1 \qquad\qquad [y] := b \qquad\qquad [x] := c$$



Example 3

$$[x] := 0; [y] := 0$$

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



Simulation

1. Introduce $SimInvariant : T_{State} \times S_{State}$;

Simulation

1. Introduce $SimInvariant : T_{State} \times S_{State}$;
2. Show that
   $\forall t, t' \in T_{State}, s \in S_{State}$.

Simulation

1. Introduce $SimInvariant : T_{State} \times S_{State}$;
2. Show that

   $\forall t, t' \in T_{State}, s \in S_{State}.$

   $correct(t), t \xrightarrow[T]{} t', SimInvariant(t, s),$

Simulation

1. Introduce $SimInvariant : T_{State} \times S_{State}$;
2. Show that

$$\forall t, t' \in T_{State}, s \in S_{State}.$$
$$correct(t), t \xrightarrow[T]{} t', SimInvariant(t, s),$$
$$\exists s' \in S_{State}. s \xrightarrow[S]{}^{*} s', SimInvariant(t', s').$$

The ARM+τ machine

- Add τ-map component to the ARM state;
- Modify **Write Commit** rule;
- Modify **Propagate** rule.

ARM+τ simulates ARM

$\forall Prog, \{\mathbf{s}_i\}_{i \in [0..n]}$,

ARM+τ simulates ARM

$$\forall \mathit{Prog}, \{\mathbf{s}_i\}_{i\in[0..n]},$$
$$\mathbf{s}^{\mathrm{init}}(\mathit{Prog}) = \mathbf{s}_0 \xrightarrow[\mathrm{ARM}]{} \ldots \xrightarrow[\mathrm{ARM}]{} \mathbf{s}_n, \mathsf{Final}^{\mathrm{ARM}}(\mathbf{s}_n),$$

ARM+τ simulates ARM

$$\forall Prog, \{\mathbf{s}_i\}_{i\in[0..n]},$$
$$\mathbf{s}^{\text{init}}(Prog) = \mathbf{s}_0 \xrightarrow[\text{ARM}]{} \ldots \xrightarrow[\text{ARM}]{} \mathbf{s}_n, \text{Final}^{\text{ARM}}(\mathbf{s}_n),$$
$$\exists \{\mathbf{a}_i | \mathbf{s}_i = \text{ARM}_{state}(\mathbf{a}_i)\}_{i\in[0..n]},$$
$$\mathbf{a}_0 \xrightarrow[\text{ARM}+\tau]{} \ldots \xrightarrow[\text{ARM}+\tau]{} \mathbf{a}_n.$$

ARM+τ simulates ARM

$$\forall Prog, \{\mathbf{s}_i\}_{i\in[0..n]},$$
$$\mathbf{s}^{\text{init}}(Prog) = \mathbf{s}_0 \xrightarrow[\text{ARM}]{} \ldots \xrightarrow[\text{ARM}]{} \mathbf{s}_n, \text{Final}^{\text{ARM}}(\mathbf{s}_n),$$
$$\exists \{\mathbf{a}_i | \mathbf{s}_i = \text{ARM}_{state}(\mathbf{a}_i)\}_{i\in[0..n]},$$
$$\mathbf{a}_0 \xrightarrow[\text{ARM}+\tau]{} \ldots \xrightarrow[\text{ARM}+\tau]{} \mathbf{a}_n.$$

Sketch of the proof:

ARM+τ simulates ARM

$$\forall Prog, \{\mathbf{s}_i\}_{i\in[0..n]},$$
$$\mathbf{s}^{\text{init}}(Prog) = \mathbf{s}_0 \xrightarrow[\text{ARM}]{} \ldots \xrightarrow[\text{ARM}]{} \mathbf{s}_n, \text{Final}^{\text{ARM}}(\mathbf{s}_n),$$
$$\exists \{\mathbf{a}_i | \mathbf{s}_i = \text{ARM}_{state}(\mathbf{a}_i)\}_{i\in[0..n]},$$
$$\mathbf{a}_0 \xrightarrow[\text{ARM}+\tau]{} \ldots \xrightarrow[\text{ARM}+\tau]{} \mathbf{a}_n.$$

Sketch of the proof:

- Construct an order on writes from $\mathbf{s}_n$;

ARM+τ simulates ARM

$$\forall Prog, \{\mathbf{s}_i\}_{i\in[0..n]},$$
$$\mathbf{s}^{\text{init}}(Prog) = \mathbf{s}_0 \xrightarrow[\text{ARM}]{} \ldots \xrightarrow[\text{ARM}]{} \mathbf{s}_n, \text{Final}^{\text{ARM}}(\mathbf{s}_n),$$
$$\exists \{\mathbf{a}_i | \mathbf{s}_i = \text{ARM}_{state}(\mathbf{a}_i)\}_{i\in[0..n]},$$
$$\mathbf{a}_0 \xrightarrow[\text{ARM}+\tau]{} \ldots \xrightarrow[\text{ARM}+\tau]{} \mathbf{a}_n.$$

Sketch of the proof:

- Construct an order on writes from $\mathbf{s}_n$;
- Show $\mathbf{s}_i$ doesn't contradict the order for all i;

ARM+τ simulates ARM

$$\forall \mathit{Prog}, \{\mathbf{s}_i\}_{i\in[0..n]},$$
$$\mathbf{s}^{\text{init}}(\mathit{Prog}) = \mathbf{s}_0 \xrightarrow[\text{ARM}]{} \ldots \xrightarrow[\text{ARM}]{} \mathbf{s}_n, \text{Final}^{\text{ARM}}(\mathbf{s}_n),$$
$$\exists \{\mathbf{a}_i | \mathbf{s}_i = \text{ARM}_{state}(\mathbf{a}_i)\}_{i\in[0..n]},$$
$$\mathbf{a}_0 \xrightarrow[\text{ARM}+\tau]{} \ldots \xrightarrow[\text{ARM}+\tau]{} \mathbf{a}_n.$$

Sketch of the proof:

- Construct an order on writes from $\mathbf{s}_n$;
- Show $\mathbf{s}_i$ doesn't contradict the order for all i;
- Show the order may coincide with τs in $\{\mathbf{a}_i\}_{i\in[0..n]}$.