Extending Intel-x86 Consistency and Persistency:

Formalising the Semantics of Intel-x86 Memory Types & Non-temporal Stores

Azalea Raad
Imperial College London

Luc Maranget
Inria Paris

Viktor Vafeiadis
MPI-SWS

POPL, 2022

azalea@imperial.ac.uk

SoundAndComplete.org

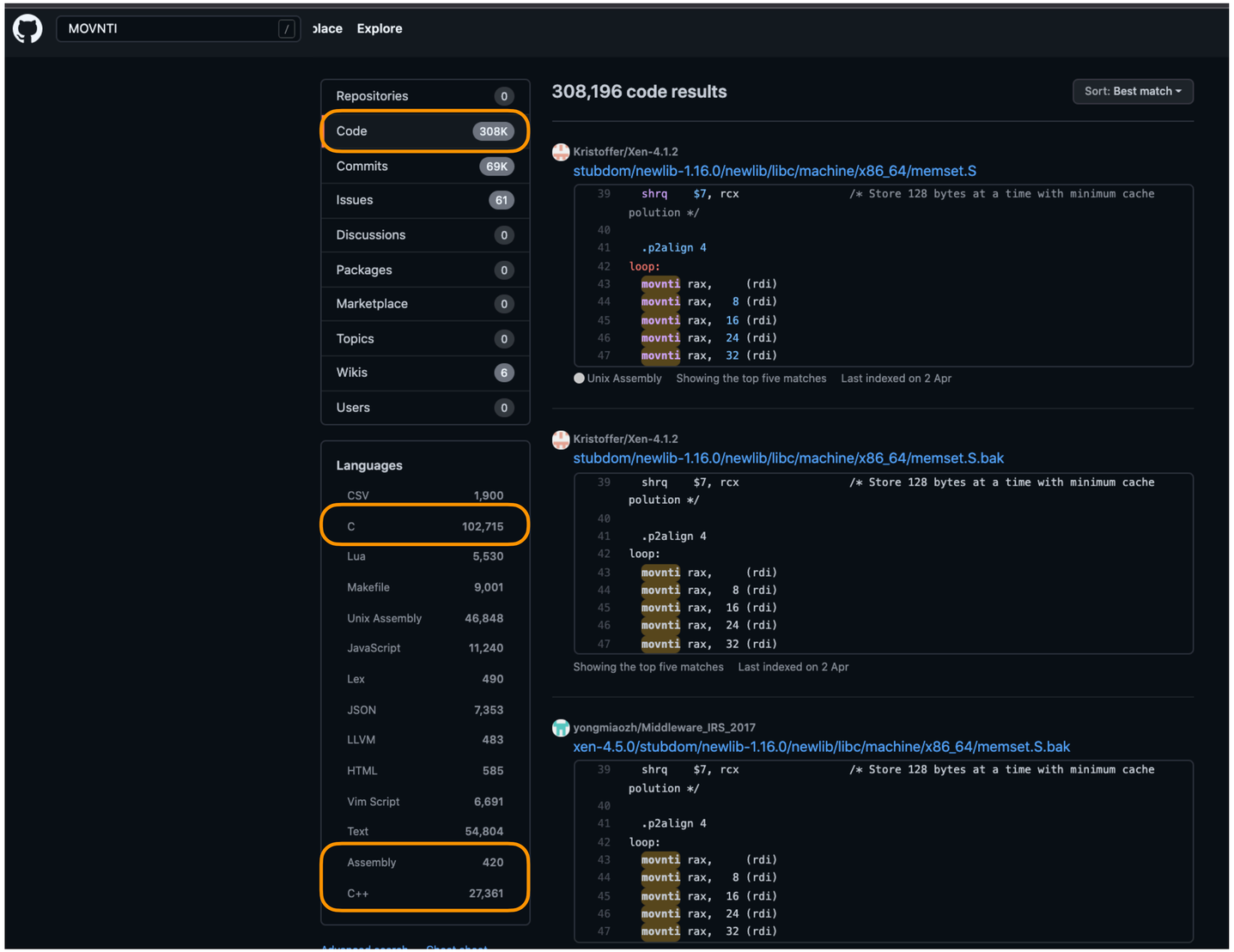
@azalearaad

Intel-x86 Non-temporal Stores

- Write ***directly to memory***, bypassing cache
- Avoids ***cache pollution***
- ***Ubiquitous*** (application-level use)

Intel-x86 Non-temporal Stores

- Write ***directly to memory***, bypassing cache
- Avoids ***cache pollution***
- ***Ubiquitous*** (application-level use)
  - ➡ 308K instances of MOVNTI on GitHub including in **C**, **C++** & **Assembly**



Intel-x86 Non-temporal Stores

- Write ***directly to memory***, bypassing cache
- Avoids ***cache pollution***
- ***Ubiquitous*** (application-level use)
  - 308K instances of MOVNTI on GitHub
    including in **C**, **C++** & **Assembly**
  - `memset` function in the **C runtime**
  - `memcpy` in **glibc**

Intel-x86 Non-temporal Stores

- Write ***directly to memory***, bypassing cache
- Avoids ***cache pollution***
- ***Ubiquitous*** (application-level use)
  - 308K instances of MOVNTI on GitHub
    including in **C**, **C++** & **Assembly**
  - `memset` function in the **C runtime**
  - `memcpy` in **glibc**
  - Large-scale projects: **PMDK** and **SPDK** to interface with ***NVM***
  - Large-scale projects: **DPDK** and **DML** to communicate with ***accelerators***

Intel-x86 Memory Types

- Also known as ***memory cacheability***[*]: UC, WC, WT, WB
- ***Non-cacheable*** types: bypass memory, access (read/write) memory directly
  - UC: Strong Uncacheable
  - WC: Write Combining
- ***Cacheable*** types: memory accesses go through the cache hierarchy
  - WB: Write Back
  - WT: Write Through

* There are two other memory types: WP and UC^{-}

Intel-x86 Memory Types

- Also known as ***memory cacheability***[*]: UC, WC, WT, WB
- ***Non-cacheable*** types: bypass memory, access (read/write) memory directly
  - UC: Strong Uncacheable
  - WC: Write Combining
- ***Cacheable*** types: memory accesses go through the cache hierarchy
  - WB: Write Back
  - WT: Write Through
- Use within ***system-level*** code
  - Linux Kernel: WC for frame buffer optimisation
  - Linux Kernel: UC for memory-mapped I/O
  - Interaction with non-cache-coherent DMA device drivers

* There are two other memory types: WP and UC^-

Intel-x86 Memory Types

❖ Also known as ***memory cacheability***[*] : UC, WC, WT, WB

❖ ***Non-cacheable*** types: bypass memory, access (read/write) memory directly

➡

➡

❖ ***Ca***

➡

➡

❖ Us

➡

➡ [illegible]

➡ Interaction with non-cache-coherent DMA device drivers

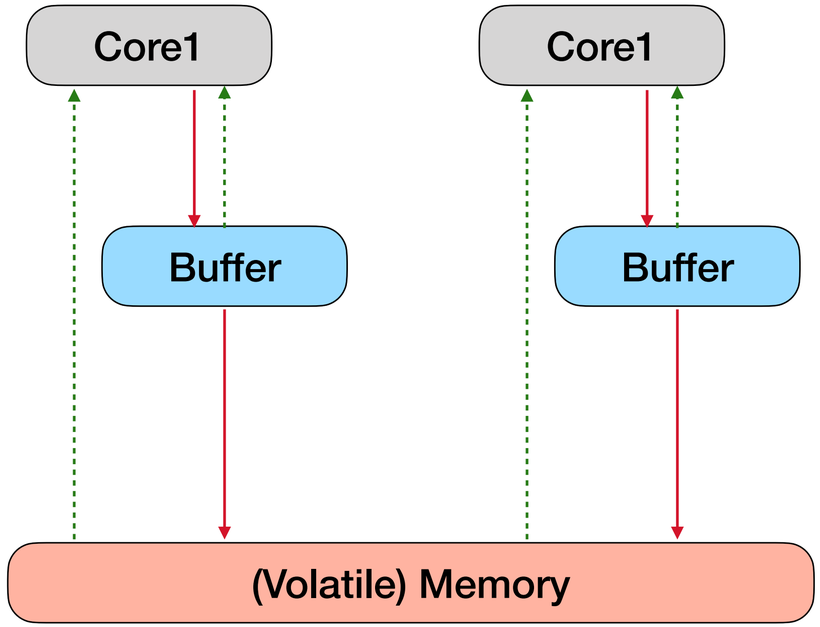
Ex86 (Extended x86):

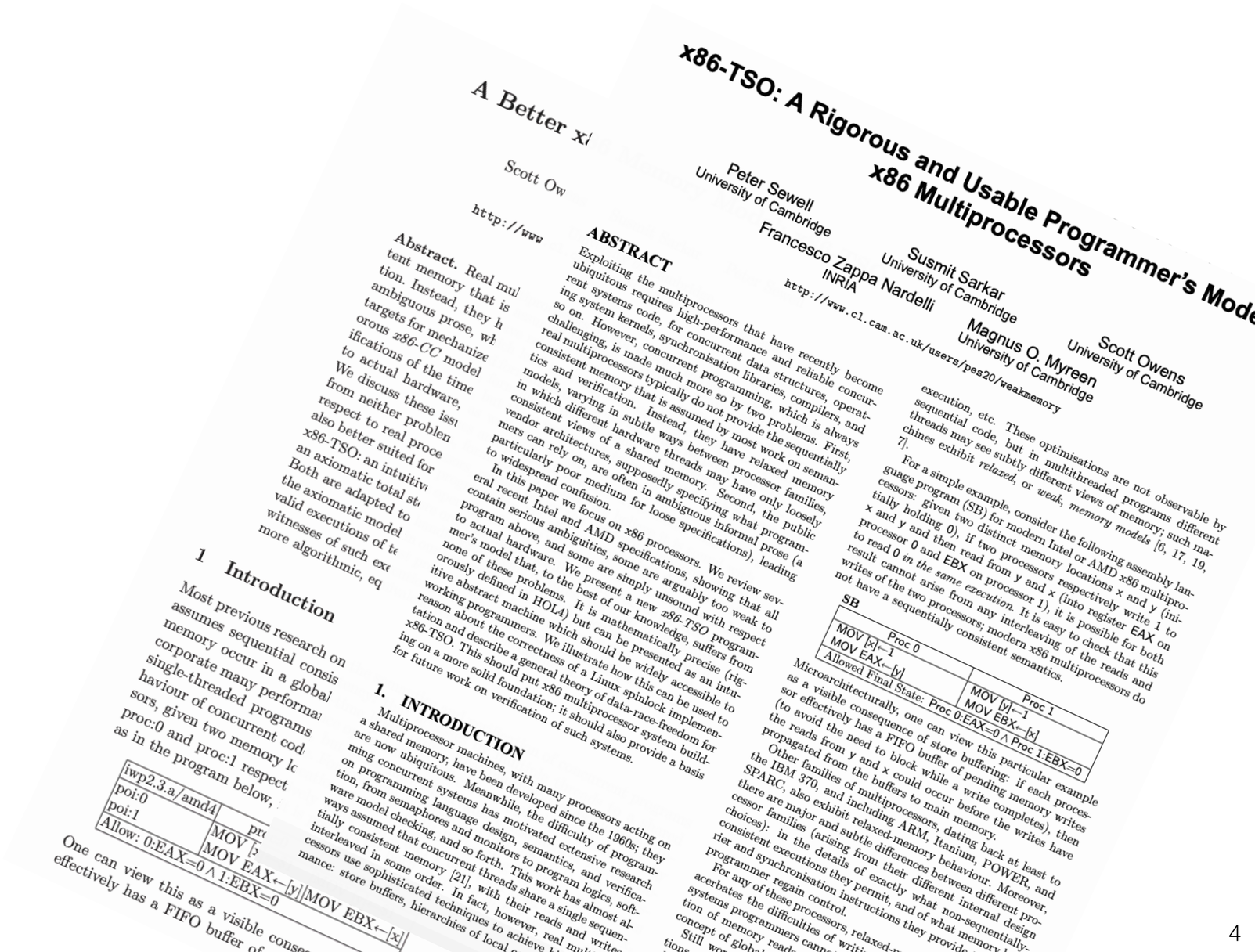
Formal **consistency** semantics of Intel-x86 architectures including
non-temporal stores & **memory types**

[*] There are two other memory types: WP and UC[-]

Ex86: Extended Intel-x86 ***Consistency*** Semantics

Isn't it just **TSO**?

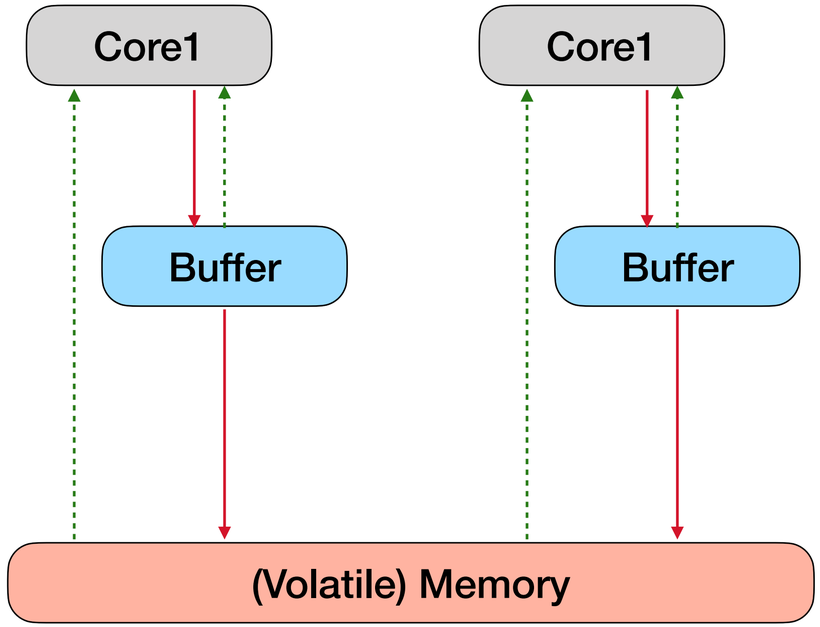


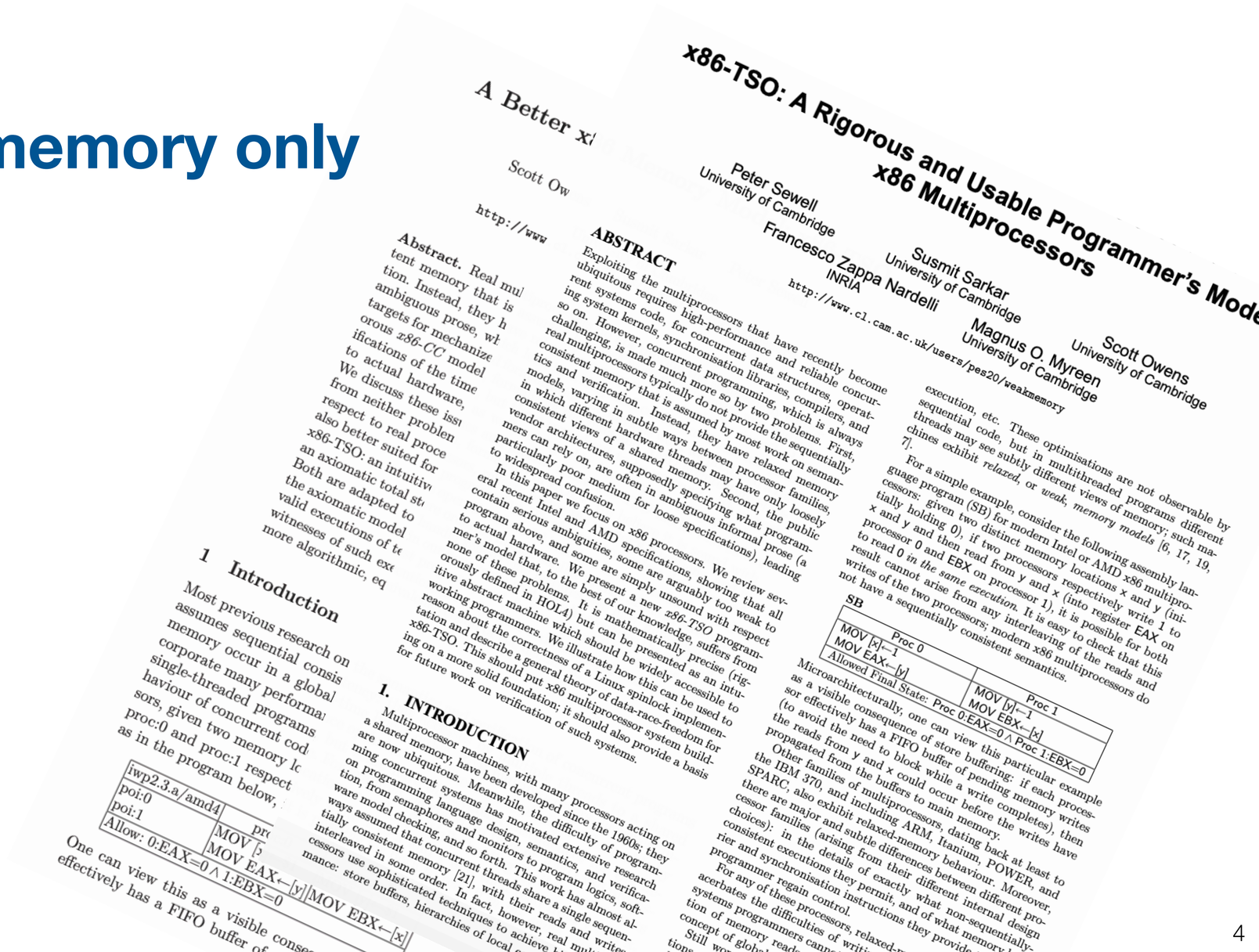


Ex86: Extended Intel-x86 ***Consistency*** Semantics

Isn't it just **TSO**?

TSO confirmed for **WB memory only**





Ex86: Extended Intel-x86 ***Consistency*** Semantics

Store buffering (SB)

Initially, $x = y = 0$

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

Message passing (MP)

Initially, $x = y = 0$

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

Ex86: Extended Intel-x86 ***Consistency*** Semantics

Store buffering (SB)

Initially, $x = y = 0$

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

Message passing (MP)

Initially, $x = y = 0$

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

SC ✗ ✗

Ex86: Extended Intel-x86 ***Consistency*** Semantics

Store buffering (SB)

Initially, $x = y = 0$

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

Message passing (MP)

Initially, $x = y = 0$

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

SC	✗	✗
TSO	✓	✗

Ex86: Extended Intel-x86 ***Consistency*** Semantics

Store buffering (SB)

Initially, $x = y = 0$

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

Message passing (MP)

Initially, $x = y = 0$

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

	SB	MP
SC	✗	✗
TSO/ WB, WT	✓	✗

WB, WT memory are subject to ***TSO*** consistency:

write-read reordering

WB and WT Memory Types

Table 11-2. Memory Types and Their Properties

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Write Through (WT)	Yes	No	Yes	Speculative Processor Ordering.
Write Back (WB)	Yes	Yes	Yes	Speculative Processor Ordering.

TSO

WB and WT Memory Types

Table 11-2. Memory Types and Their Properties

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Write Through (WT)	Yes	No	Yes	Speculative Processor Ordering.
Write Back (WB)	Yes	Yes	Yes	Speculative Processor Ordering.

TSO

applies **only** to **all-WB/ all-WT** accesses, **not mixed** accesses

WB and WT Memory Types

Table 11-2. Memory Types and Their Properties

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Write Through (WT)	Yes	No	Yes	Speculative Processor Ordering.
Write Back (WB)	Yes	Yes	Yes	Speculative Processor Ordering.

TSO

applies **only** to **all-WB/ all-WT** accesses, **not mixed** accesses

Write-through (WT) — Writes and reads to and from system memory are cached. Reads come from cache lines on cache hits; read misses cause cache fills. Speculative reads are allowed. All writes are written to a cache line (when possible) and through to system memory. When writing through to memory, invalid cache lines are never filled, and valid cache lines are either filled or invalidated. Write combining is allowed. This type of cache-control is appropriate for frame buffers or when there are devices on the system bus that access system memory, but do not perform snooping of memory accesses. It enforces coherency between caches in the processors and system memory.

Write-back (WB) — Writes and reads to and from system memory are cached. Reads come from cache lines on cache hits; read misses cause cache fills. Speculative reads are allowed. Write misses cause cache line fills (in processor families starting with the P6 family processors), and writes are performed entirely in the cache, when possible. Write combining is allowed. The write-back memory type reduces bus traffic by eliminating many unnecessary writes to system memory. Writes to a cache line are not immediately forwarded to system memory; instead, they are accumulated in the cache. The modified cache lines are written to system memory later, when a write-back operation is performed. Write-back operations are triggered when cache lines need to be deallocated, such as when new cache lines are being allocated in a cache that is already full. They also are triggered by the mechanisms used to maintain cache consistency. This type of cache-control provides the best performance, but it requires that all devices that access system memory on the system bus be able to snoop memory accesses to ensure system memory and cache coherency.

← The extent of WB/WT Specification in the Intel manual

Ex86: Extended Intel-x86 ***Consistency*** Semantics

	Store buffering (SB)	**Message passing (MP)**
	Initially, $x = y = 0$	Initially, $x = y = 0$
	$x := 1 \quad \| \quad y := 1$ $a := y \;\; // 0 \quad \| \quad b := x \;\; // 0$	$x := 1 \quad \| \quad a := y \;\; // 1$ $y := 1 \quad \| \quad b := x \;\; // 0$
SC	✗	✗
TSO/ WB, WT	✓	✗
UC	✗	✗

Ex86: Extended Intel-x86 ***Consistency*** Semantics

Store buffering (SB)

Initially, $x = y = 0$

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

Message passing (MP)

Initially, $x = y = 0$

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

	SB	MP
SC	✗	✗
TSO/ WB, WT	✓	✗
UC	✗	✗

UC memory is subject to **SC** consistency semantics:

no reordering

UC Memory Type

Table 11-2. Memory Types and Their Properties

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Strong Uncacheable (UC)	No	No	No	Strong Ordering ← SC

UC Memory Type

Table 11-2. Memory Types and Their Properties

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Strong Uncacheable (UC)	No	No	No	Strong Ordering ← SC

applies **only** to **all-UC** accesses, **not mixed** accesses

UC Memory Type

Table 11-2. Memory Types and Their Properties

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Strong Uncacheable (UC)	No	No	No	Strong Ordering ← SC

applies **only** to **all-UC** accesses, **not mixed** accesses

Strong Uncacheable (UC) —System memory locations are not cached. All reads and writes appear on the system bus and are executed in program order without reordering. No speculative memory accesses, page-table walks, or prefetches of speculated branch targets are made. This type of cache-control is useful for memory-mapped I/O devices. When used with normal RAM, it greatly reduces processor performance.

← The extent of UC Specification in the Intel manual

Ex86: Extended Intel-x86 ***Consistency*** Semantics

Store buffering (SB)

Initially, $x = y = 0$

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

Message passing (MP)

Initially, $x = y = 0$

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

	SB	MP
SC	✗	✗
TSO/ WB, WT	✓	✗
UC	✗	✗
WC	✗	✓

WC memory: **write-write reordering** on different locations

WC Memory Type

Table 11-2. Memory Types and Their Properties

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Write Combining (WC)	No	No	Yes	Weak Ordering. Available by programming MTRRs or by selecting it through the PAT.

write-write reordering on different locations

WC Memory Type

Table 11-2. Memory Types and Their Properties

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Write Combining (WC)	No	No	Yes	Weak Ordering. Available by programming MTRRs or by selecting it through the PAT.

write-write reordering on different locations

applies **only** to **all-WC** accesses, **not mixed** accesses

WC Memory Type

Table 11-2. Memory Types and Their Properties

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Write Combining (WC)	No	No	Yes	Weak Ordering. Available by programming MTRRs or by selecting it through the PAT.

write-write reordering on different locations

applies **only** to **all-WC** accesses, **not mixed** accesses

Write Combining (WC) — System memory locations are not cached (as with uncacheable memory) and coherency is not enforced by the processor's bus coherency protocol. Speculative reads are allowed. Writes may be delayed and combined in the write combining buffer (WC buffer) to reduce memory accesses. If the WC buffer is partially filled, the writes may be delayed until the next occurrence of a serializing event; such as an SFENCE or MFENCE instruction, CPUID or other serializing instruction, a read or write to uncached memory, an interrupt occurrence, or an execution of a LOCK instruction (including one with an XACQUIRE or XRELEASE prefix). In addition, an execution of the XEND instruction (to end a transactional region) evicts any writes that were buffered before the corresponding execution of the XBEGIN instruction (to begin the transactional region) before evicting any writes that were performed inside the transactional region.

This type of cache-control is appropriate for video frame buffers, where the order of writes is unimportant as long as the writes update memory so they can be seen on the graphics display. See Section 11.3.1, "Buffering of Write Combining Memory Locations," for more information about caching the WC memory type. This memory type is available in the Pentium Pro and Pentium II processors by programming the MTRRs; or in processor families starting from the Pentium III processors by programming the MTRRs or by selecting it through the PAT.

← The extent of WC Specification in the Intel manual

What about ***Non-temporal Stores***?

Intel Manual: Non-temporal Stores

These SSE and SSE2 non-temporal store instructions minimize cache pollutions by treating the memory being accessed as the write combining (WC) type.

Using the WC semantics, the store transaction will be weakly ordered, meaning that the data may not be written to memory in program order,

Intel Manual: Non-temporal Stores

These SSE and SSE2 non-temporal store instructions minimize cache pollutions by treating the memory being accessed as the write combining (WC) type.

Using the WC semantics, the store transaction will be weakly ordered, meaning that the data may not be written to memory in program order,

According to the Intel manual:

Non-temporal stores have the same semantics as ***WC memory***

Intel Manual: Non-temporal Stores

These SSE and SSE2 non-temporal store instructions minimize cache pollutions by treating the memory being accessed as the write combining (WC) type.

Using the WC semantics, the store transaction will be weakly ordered, meaning that the data may not be written to memory in program order,

According to the Intel manual:

Non-temporal stores have the same semantics as ***WC memory***

But…

Ex86: Extended Intel-x86 ***Consistency*** Semantics

Store buffering (SB)

Initially, $x = y = 0$

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

Message passing (MP)

Initially, $x = y = 0$

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

	SB	MP
SC	✗	✗
TSO/ WB, WT	✓	✗
UC	✗	✗
WC	✗	✓
MOVNT	✓	✓

Ex86: Extended Intel-x86 ***Consistency*** Semantics

Store buffering (SB)

Initially, $x = y = 0$

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

Message passing (MP)

Initially, $x = y = 0$

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

	SB	MP
SC	✗	✗
TSO/ WB, WT	✓	✗
UC	✗	✗
WC	✗	✓
MOVNT	✓	✓

WC & NT stores have **different** semantics

Ex86: Extended Intel-x86 ***Consistency*** Semantics

Store buffering (SB)

Initially, $x = y = 0$

Message passing (MP)

Initially, $x = y = 0$

// 1

// 0

Solution

Validate the Ex86 Consistency Semantics!

TSO/ WB,

WC

MOVNT

✗

✓

⟵ WC & NT stores have **different** semantics

✓

✓

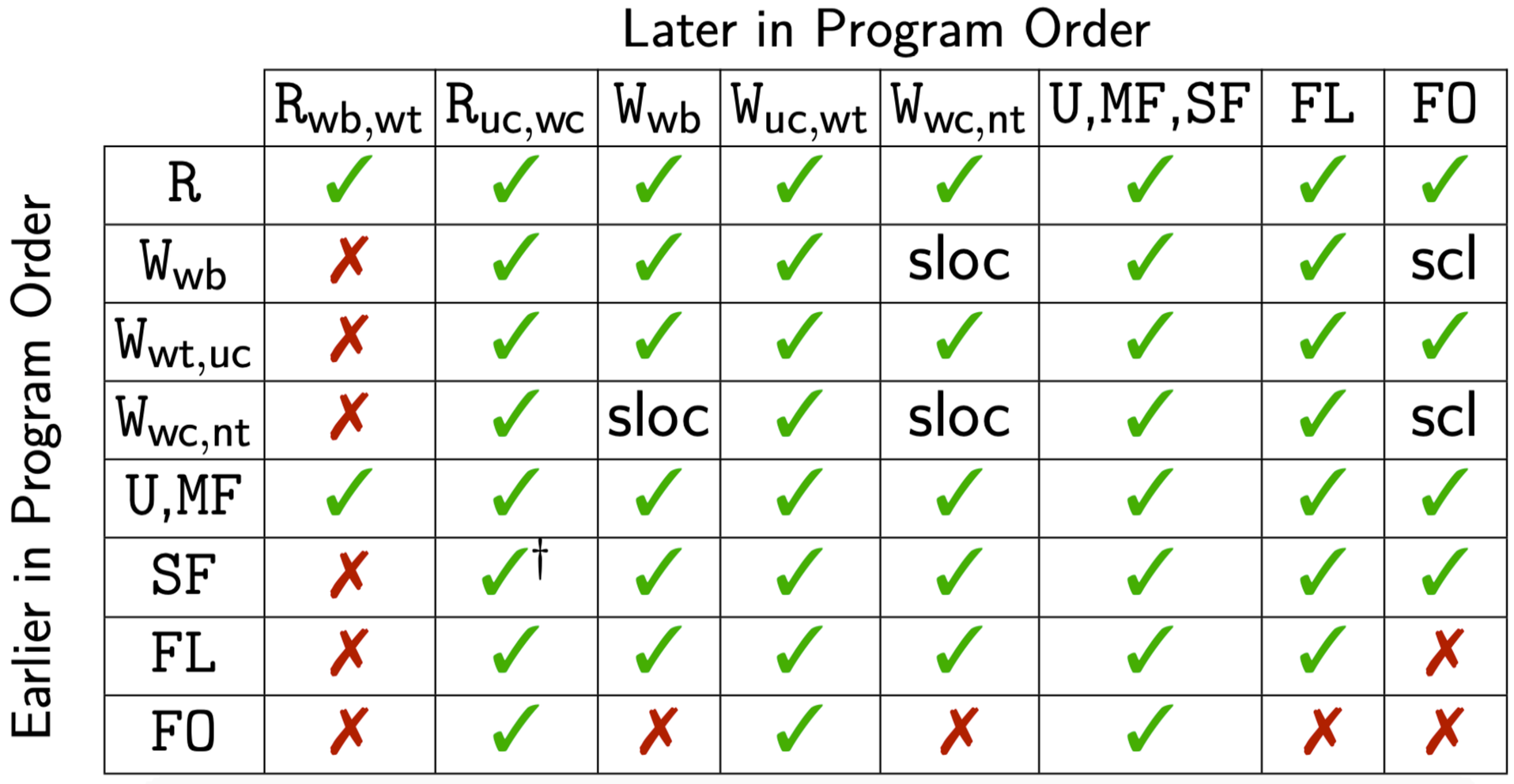
Ex86 Validation

- ***Validated*** Ex86 using the **diy** tool suite
- Extended the **klitmus** tool to allow for specifying memory types
- Built a test base of ***over 2200 tests***
- Ran tests on ***various Intel-x86 CPU implementations***
  - ➡ e.g. corel5, corel6 and Xeon
- Ran each test ***at least 6 x 10^8 times***; ran critical tests up to a few billion times

Ex86 Validation

- ***Validated*** Ex86 using the **diy** tool suite
- Extended the **klitmus** tool to allow for specifying memory types
- Built a test base of ***over 2200 tests***
- Ran tests on ***various Intel-x86 CPU implementations***
  - ➡ e.g. corel5, corel6 and Xeon
- Ran each test ***at least 6 x 10^8 times***; ran critical tests up to a few billion times
- For more details see: **`http://diy.inria.fr/x86-memtype`**

Ex86 Semantics: Preserved Ordering



Earlier \ Later	$R_{wb,wt}$	$R_{uc,wc}$	W_{wb}	$W_{uc,wt}$	$W_{wc,nt}$	U,MF,SF	FL	FO
R	✓	✓	✓	✓	✓	✓	✓	✓
W_{wb}	✗	✓	✓	✓	sloc	✓	✓	scl
$W_{wt,uc}$	✗	✓	✓	✓	✓	✓	✓	✓
$W_{wc,nt}$	✗	✓	sloc	✓	sloc	✓	✓	scl
U,MF	✓	✓	✓	✓	✓	✓	✓	✓
SF	✗	✓†	✓	✓	✓	✓	✓	✓
FL	✗	✓	✓	✓	✓	✓	✓	✗
FO	✗	✓	✗	✓	✗	✓	✗	✗

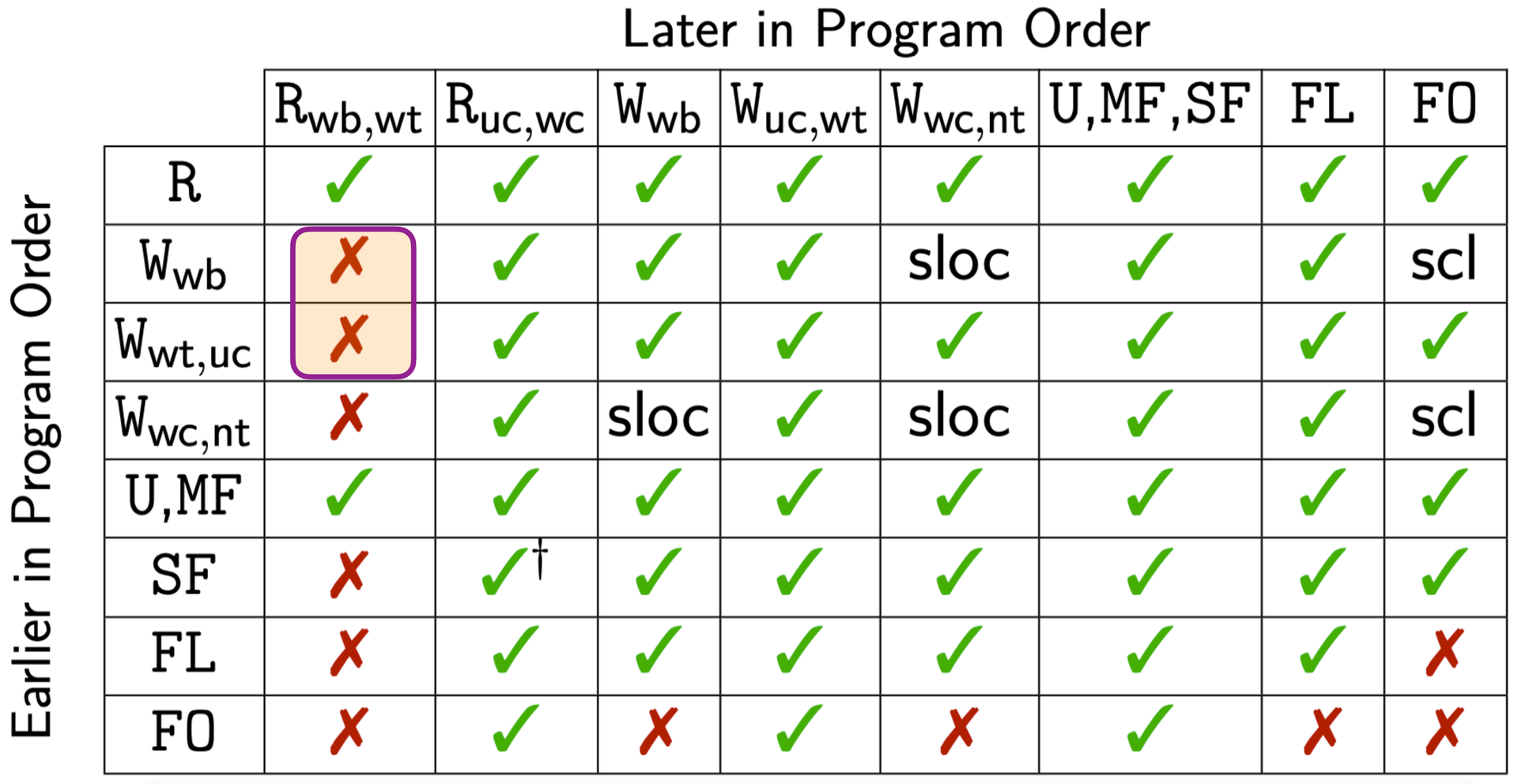
✓ Order preserved; may not be reordered

sloc: Order preserved iff on the same location

scl: Order preserved iff on the same cache line

✗ Order not preserved may be reordered

Ex86 Semantics: Preserved Ordering



Earlier in Program Order \ Later in Program Order	$R_{wb,wt}$	$R_{uc,wc}$	W_{wb}	$W_{uc,wt}$	$W_{wc,nt}$	U,MF,SF	FL	FO
R	✓	✓	✓	✓	✓	✓	✓	✓
W_{wb}	✗	✓	✓	✓	sloc	✓	✓	scl
$W_{wt,uc}$	✗	✓	✓	✓	✓	✓	✓	✓
$W_{wc,nt}$	✗	✓	sloc	✓	sloc	✓	✓	scl
U,MF	✓	✓	✓	✓	✓	✓	✓	✓
SF	✗	✓†	✓	✓	✓	✓	✓	✓
FL	✗	✓	✓	✓	✓	✓	✓	✗
FO	✗	✓	✗	✓	✗	✓	✗	✗

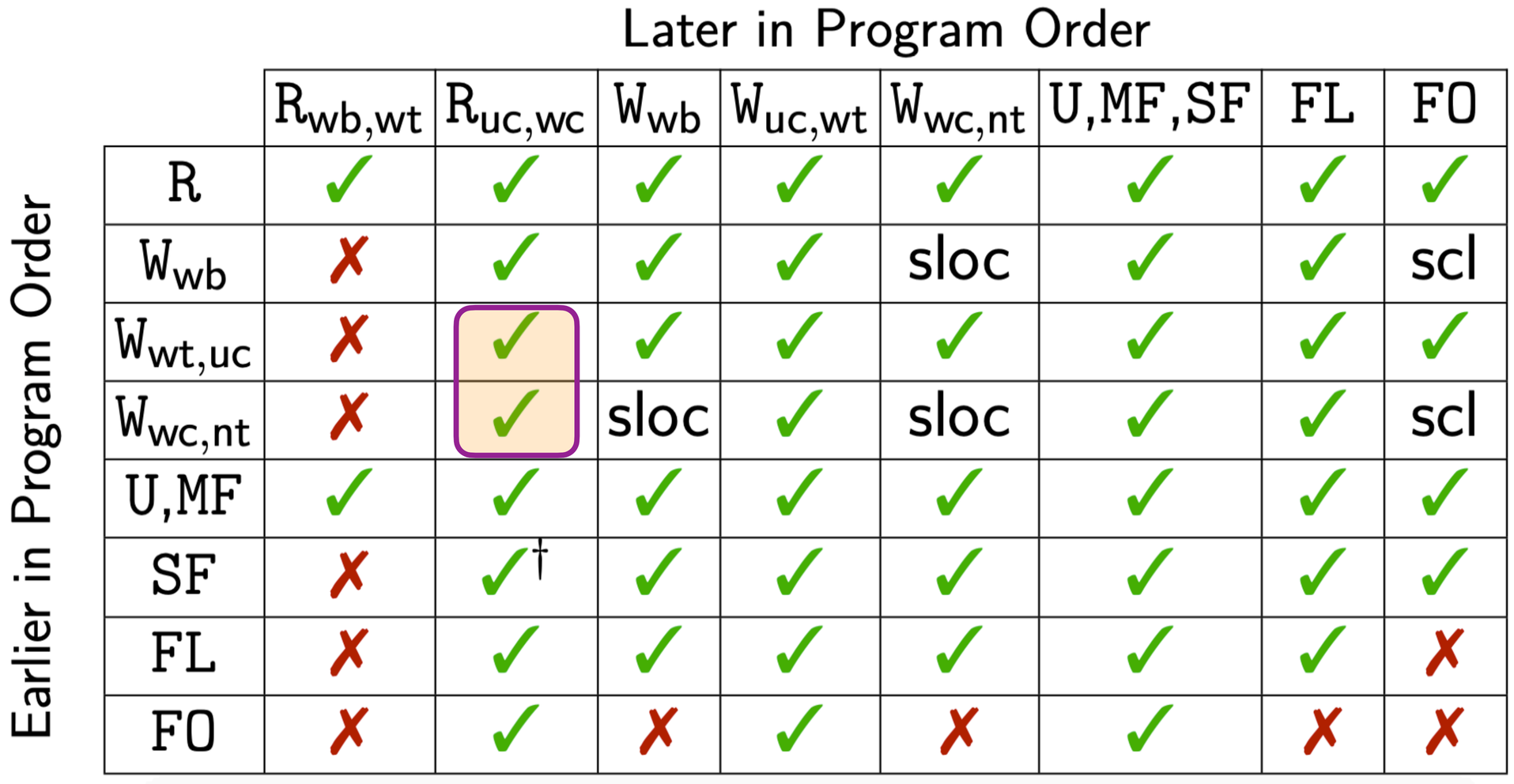
✓ Order preserved; may not be reordered

sloc: Order preserved iff on the same location

scl: Order preserved iff on the same cache line

✗ Order not preserved may be reordered

Ex86 Semantics: Preserved Ordering



Earlier in Program Order \ Later in Program Order	$R_{wb,wt}$	$R_{uc,wc}$	W_{wb}	$W_{uc,wt}$	$W_{wc,nt}$	U,MF,SF	FL	FO
R	✓	✓	✓	✓	✓	✓	✓	✓
W_{wb}	✗	✓	✓	✓	sloc	✓	✓	scl
$W_{wt,uc}$	✗	✓	✓	✓	✓	✓	✓	✓
$W_{wc,nt}$	✗	✓	sloc	✓	sloc	✓	✓	scl
U,MF	✓	✓	✓	✓	✓	✓	✓	✓
SF	✗	✓†	✓	✓	✓	✓	✓	✓
FL	✗	✓	✓	✓	✓	✓	✓	✗
FO	✗	✓	✗	✓	✗	✓	✗	✗

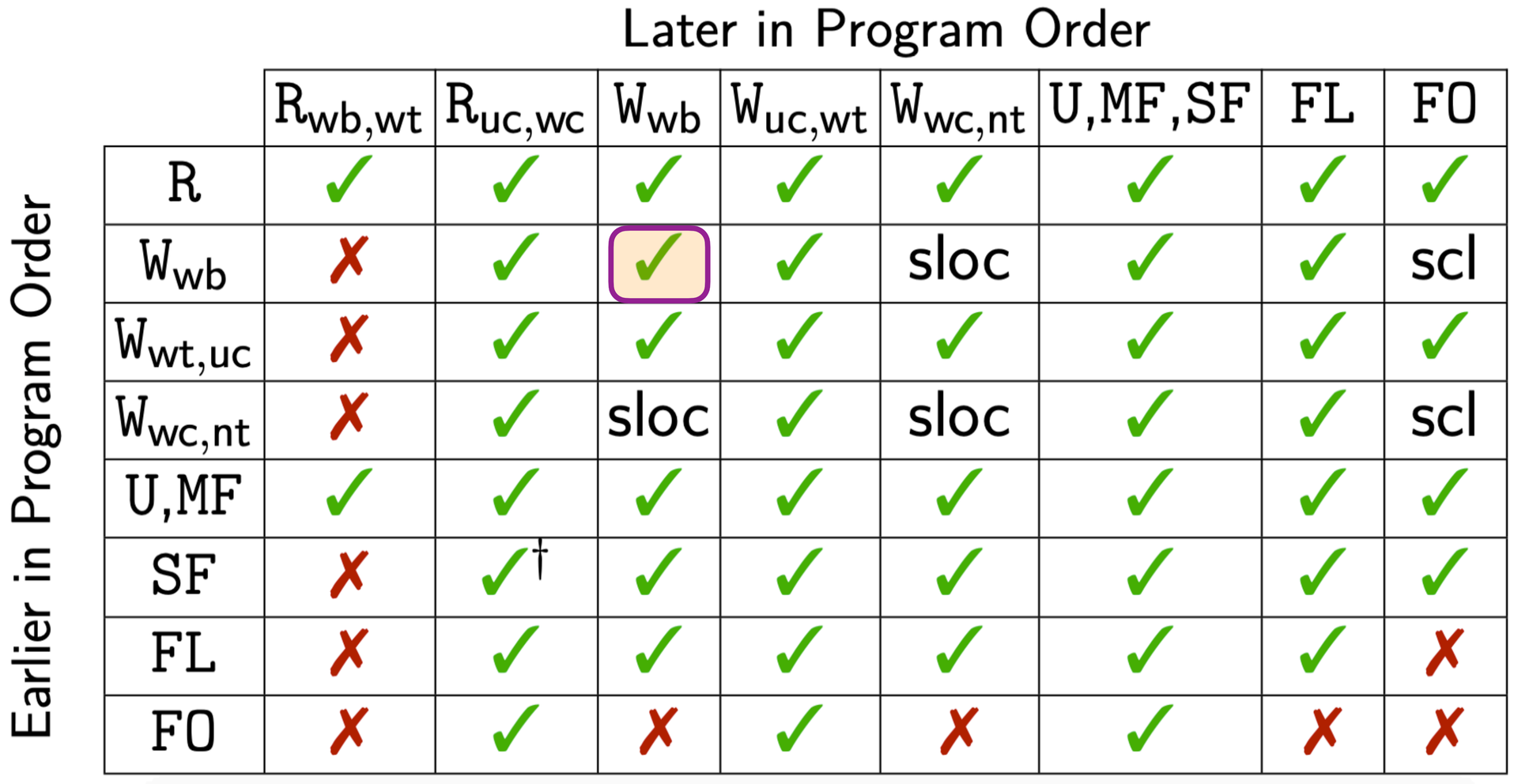
✓ Order preserved; may not be reordered

sloc: Order preserved iff on the same location

scl: Order preserved iff on the same cache line

✗ Order not preserved may be reordered

Ex86 Semantics: Preserved Ordering



Earlier in Program Order \ Later in Program Order	$R_{wb,wt}$	$R_{uc,wc}$	W_{wb}	$W_{uc,wt}$	$W_{wc,nt}$	U,MF,SF	FL	FO
R	✓	✓	✓	✓	✓	✓	✓	✓
W_{wb}	✗	✓	✓	✓	sloc	✓	✓	scl
$W_{wt,uc}$	✗	✓	✓	✓	✓	✓	✓	✓
$W_{wc,nt}$	✗	✓	sloc	✓	sloc	✓	✓	scl
U,MF	✓	✓	✓	✓	✓	✓	✓	✓
SF	✗	✓†	✓	✓	✓	✓	✓	✓
FL	✗	✓	✓	✓	✓	✓	✓	✗
FO	✗	✓	✗	✓	✗	✓	✗	✗

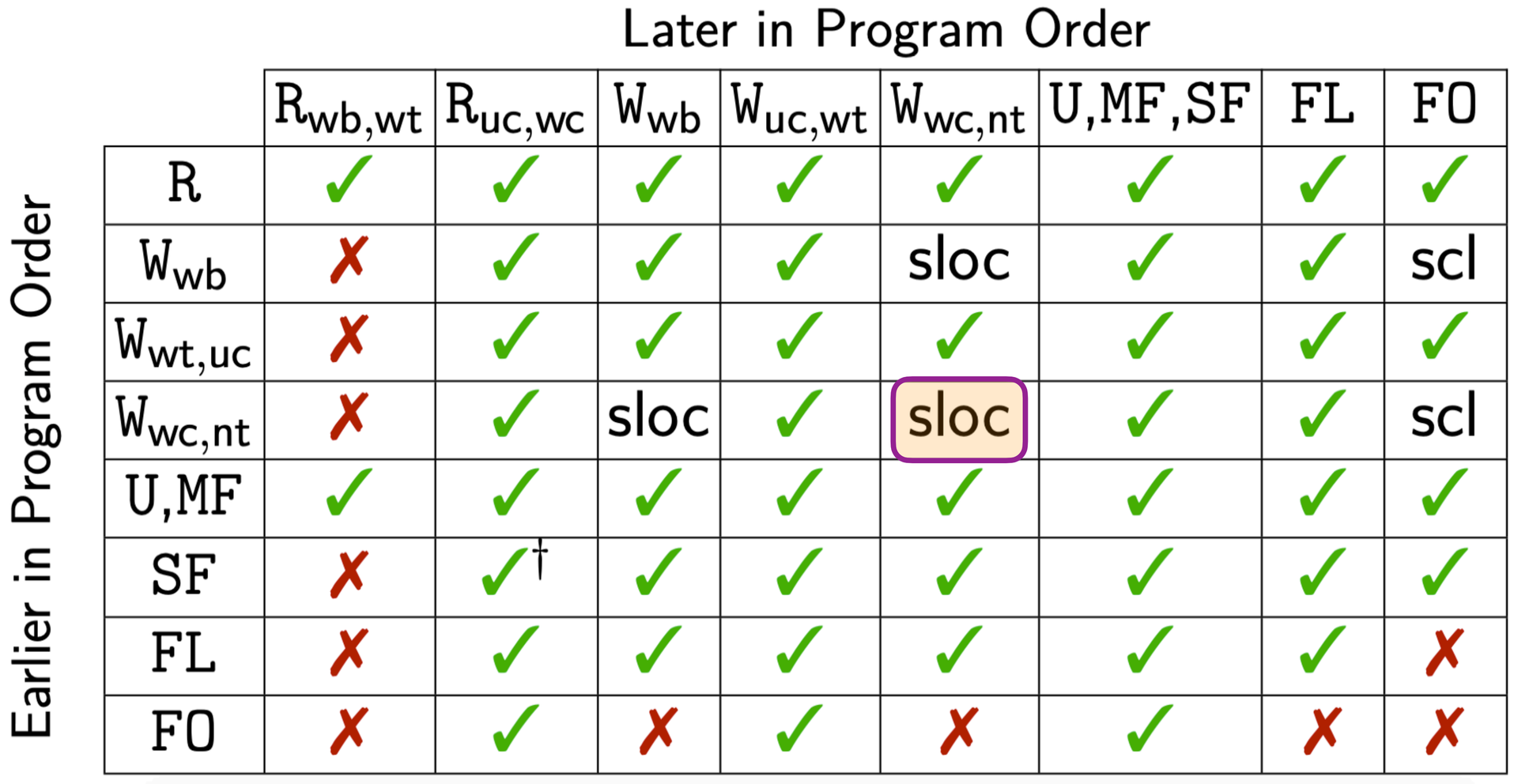
✓ Order preserved; may not be reordered

sloc: Order preserved iff on the same location

scl: Order preserved iff on the same cache line

✗ Order not preserved may be reordered

Ex86 Semantics: Preserved Ordering



	$R_{wb,wt}$	$R_{uc,wc}$	W_{wb}	$W_{uc,wt}$	$W_{wc,nt}$	U,MF,SF	FL	FO
R	✓	✓	✓	✓	✓	✓	✓	✓
W_{wb}	✗	✓	✓	✓	sloc	✓	✓	scl
$W_{wt,uc}$	✗	✓	✓	✓	✓	✓	✓	✓
$W_{wc,nt}$	✗	✓	sloc	✓	sloc	✓	✓	scl
U,MF	✓	✓	✓	✓	✓	✓	✓	✓
SF	✗	✓†	✓	✓	✓	✓	✓	✓
FL	✗	✓	✓	✓	✓	✓	✓	✗
FO	✗	✓	✗	✓	✗	✓	✗	✗

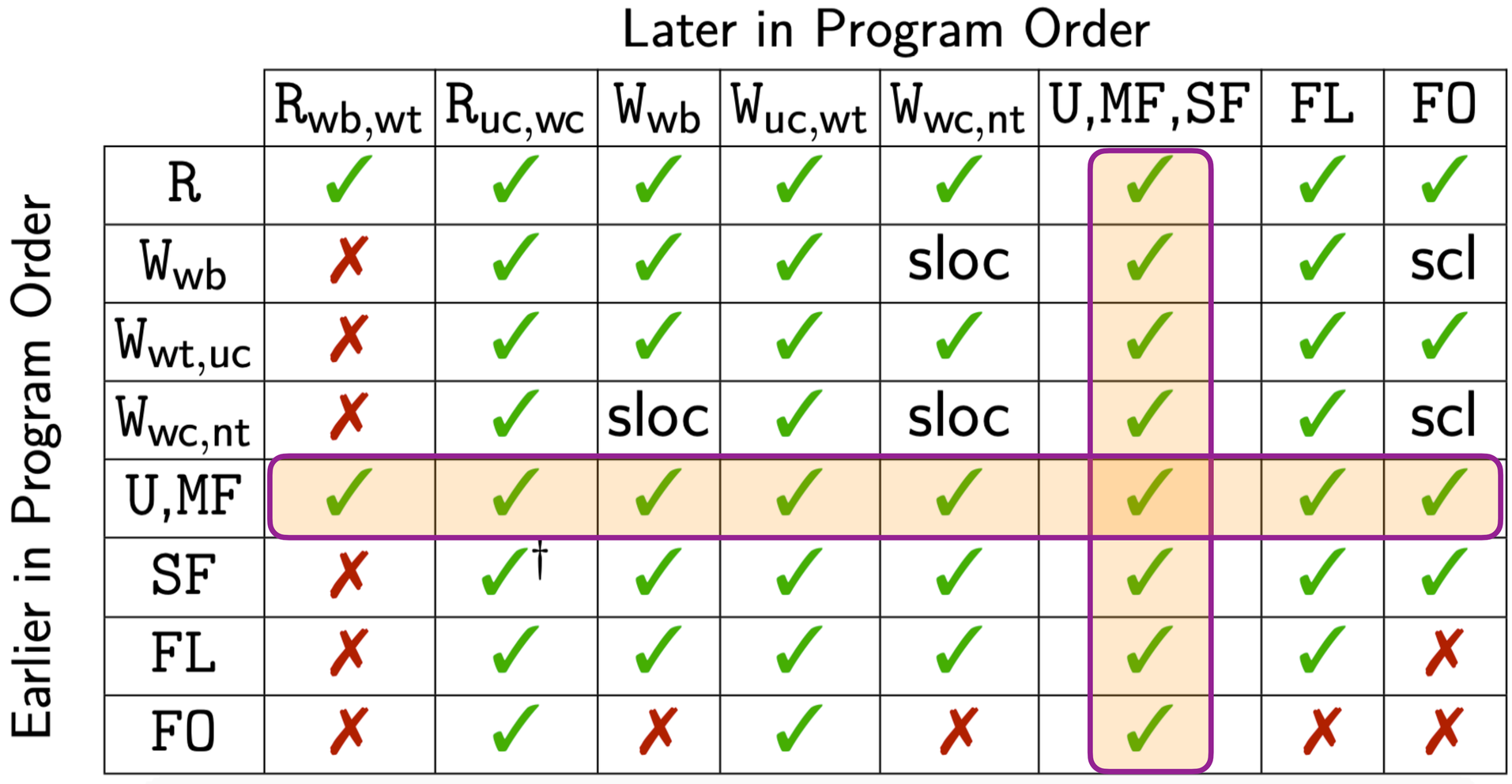
✓ Order preserved; may not be reordered

sloc: Order preserved iff on the same location

scl: Order preserved iff on the same cache line

✗ Order not preserved may be reordered

Ex86 Semantics: Preserved Ordering



Earlier in Program Order \ Later in Program Order	$R_{wb,wt}$	$R_{uc,wc}$	W_{wb}	$W_{uc,wt}$	$W_{wc,nt}$	U,MF,SF	FL	FO
R	✓	✓	✓	✓	✓	✓	✓	✓
W_{wb}	✗	✓	✓	✓	sloc	✓	✓	scl
$W_{wt,uc}$	✗	✓	✓	✓	✓	✓	✓	✓
$W_{wc,nt}$	✗	✓	sloc	✓	sloc	✓	✓	scl
U,MF	✓	✓	✓	✓	✓	✓	✓	✓
SF	✗	✓†	✓	✓	✓	✓	✓	✓
FL	✗	✓	✓	✓	✓	✓	✓	✗
FO	✗	✓	✗	✓	✗	✓	✗	✗

✓ Order preserved; may not be reordered

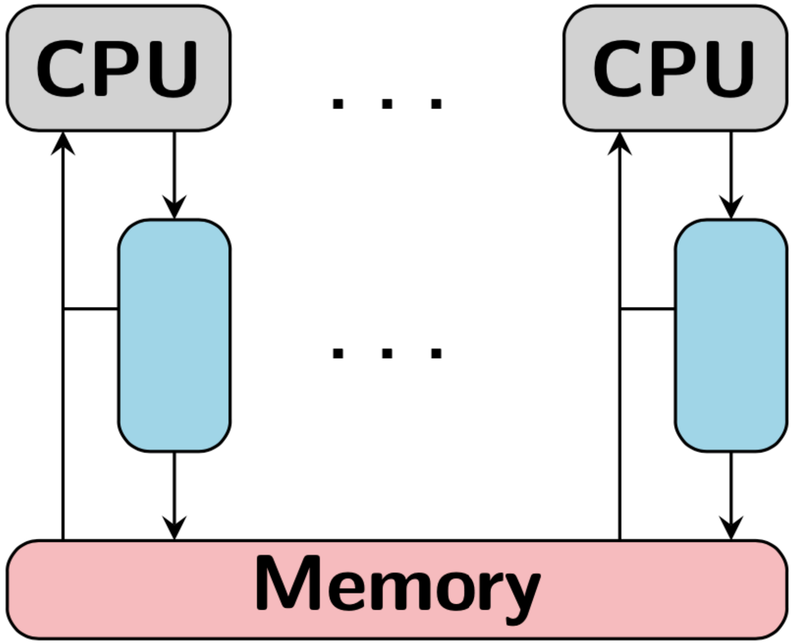
sloc: Order preserved iff on the same location

scl: Order preserved iff on the same cache line

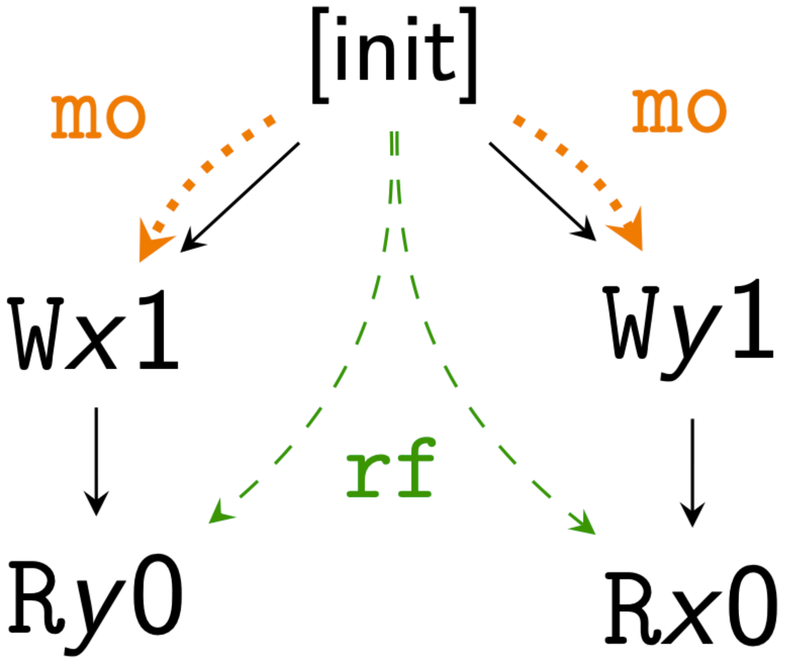
✗ Order not preserved may be reordered

Ex86 Semantics: Two ***Equivalent*** Models

Operational Ex86



Declarative Ex86

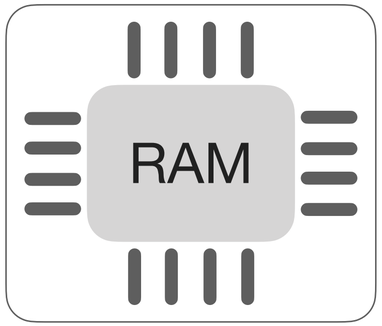


Proved the ***equivalence*** of the two models

What about Intel-x86 ***Persistency*** Semantics?

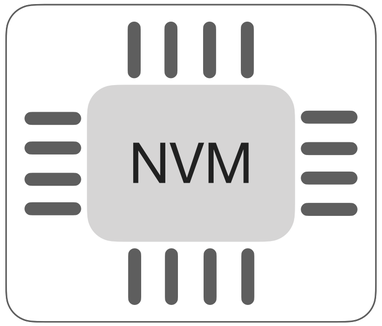
Computer Storage

✓ *fast*
✗ *volatile*



✗ *slow*
✓ *persistent*

What is Non-Volatile Memory (NVM)?



NVM: Hybrid Storage + Memory

Best of both worlds:

✓ ***persistent*** (like HDD)
✓ ***fast***, ***random access*** (like RAM)

What Can Go Wrong?

```
// x=0;y=0
x := 1;
y := 1;
```

What Can Go Wrong?

```
// x=0;y=0
x := 1;
y := 1;
```

```
// x=1;y=1
```

What Can Go Wrong?

```
// x=0;y=0
x := 1;
y := 1;
```

```
// x=1;y=1 OR x=0;y=0
```

!! Execution continues ***ahead of persistence***

— ***asynchronous*** persists

What Can Go Wrong?

```
// x=0;y=0
x := 1;
y := 1;
```

```
// x=1;y=1 OR x=0;y=0 OR x=1;y=0
```

!! Execution continues ***ahead of persistence***

— ***asynchronous*** persists

What Can Go Wrong?

```
// x=0;y=0
x := 1;
y := 1;
```

// x=1;y=1 OR x=0;y=0 OR x=1;y=0 OR x=0;y=1

!! Execution continues ***ahead of persistence***

— ***asynchronous*** persists

!! Writes may persist ***out of order***

What Can Go Wrong?

Consistency Model

the ***order*** in which writes
are ***made visible*** to other threads

Persistency Model

the ***order*** in which writes
are ***persisted*** to NVM

Full Semantics

Consistency + Persistency Model

PEx86 (Persistent Extended x86):

Formal **consistency + Persistency** semantics of

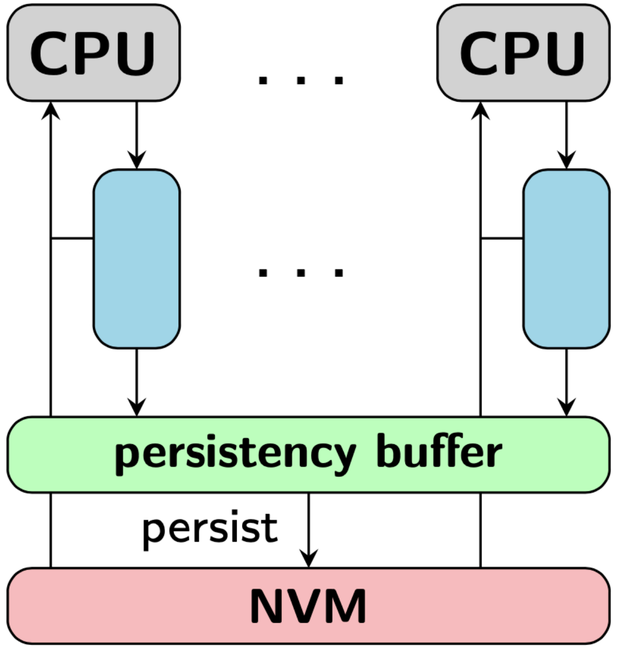
Intel-x86 architectures

including

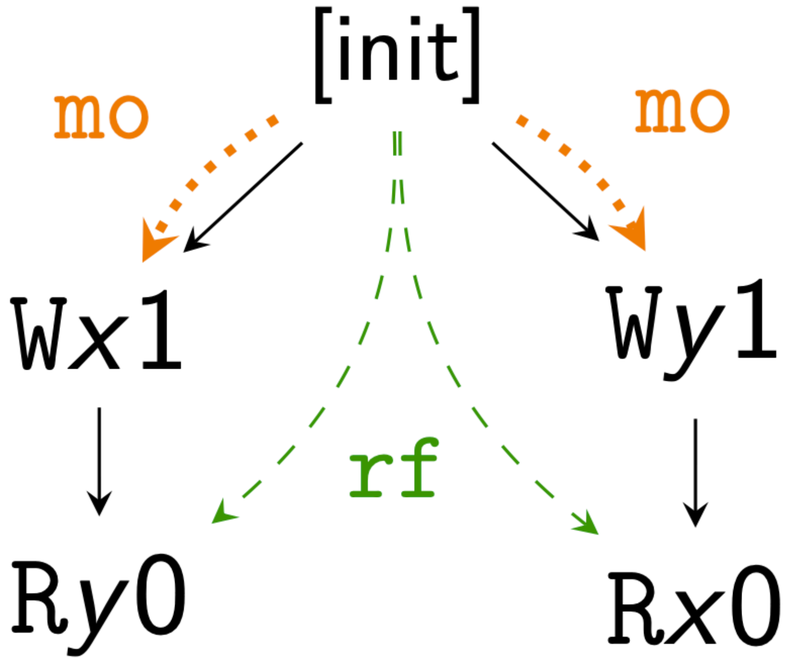
non-temporal stores & **memory types**

PEx86 Semantics: Two ***Equivalent*** Models

Operational PEx86



Declarative PEx86



Proved the ***equivalence*** of the two models

PEx86: Persistent Extended Intel-x86 Semantics

$x, y \in \mathrm{Loc}_{\mathrm{wb}}$	$x, x', y \in \mathrm{Loc}_{\mathrm{wb}}$	$x, x', y \in \mathrm{Loc}_{\mathrm{wb}}$	$x, x', y \in \mathrm{Loc}_{\mathrm{wb}}$	$x, x', y \in \mathrm{Loc}_{\mathrm{wb}}$
$x := 1$ $y := 1$	$x := 1$ **clflush** x' $y := 1$	$x := 1$ **clflushopt** x' $y := 1$	$x := 1$ **clflushopt** x' **xchg**$(y, 1)$	$x := 1$ **clflushopt** x'; **sfence** $y := 1$
rec: $x,y \in \{0,1\}$	rec: $y{=}1 \Rightarrow x{=}1$	rec: $x,y \in \{0,1\}$	rec: $y{=}1 \Rightarrow x{=}1$	rec: $y{=}1 \Rightarrow x{=}1$

$x \in \mathrm{Loc}_{\mathrm{uc} \cup \mathrm{wt}}$ $y \in \mathrm{Loc}$	$x \in \mathrm{Loc}_{\mathrm{wc}}$, $y \in \mathrm{Loc}_{\mathrm{wc} \cup \mathrm{wb}}$	$x \in \mathrm{Loc}_{\mathrm{wc}}$, $y \in \mathrm{Loc}_{\mathrm{uc} \cup \mathrm{wt}}$	$x \in \mathrm{Loc}_{\mathrm{wb} \cup \mathrm{wt} \cup \mathrm{wc}}$ $y \in \mathrm{Loc}_{\mathrm{uc} \cup \mathrm{wt}}$	$x \in \mathrm{Loc}_{\mathrm{wb} \cup \mathrm{wt} \cup \mathrm{wc}}$ $y \in \mathrm{Loc}_{\mathrm{wc} \cup \mathrm{wb}}$
$x := 1$ $y := 1$	$x := 1$ $y := 1$	$x := 1$ $y := 1$	$x := 1$ $x :=_{\mathrm{NT}} 2$ $y := 1$	$x := 1$ $x :=_{\mathrm{NT}} 2$ **sfence** $y := 1$
rec: $y{=}1 \Rightarrow x{=}1$	rec: $x,y \in \{0,1\}$	rec: $y{=}1 \Rightarrow x{=}1$	rec: $y{=}1 \Rightarrow x{=}2$	rec: $y{=}1 \Rightarrow x{=}2$

Conclusions

- Developed ***Ex86***: an extensive Intel-x86 ***consistency*** model
  - Memory types (WB, WT, WC, UC)
  - Non-temporal stores
- Formalised Ex86 both ***operationally*** & ***declaratively***, and proved them ***equivalent***
- ***Empirically validated*** Ex86 through extensive testing
- Developed ***PEx86***: an extensive Intel-x86 ***consistency*** and ***persistency*** model
  - Memory types (WB, WT, WC, UC)
  - Non-temporal stores
- Formalised PEx86 both ***operationally*** & ***declaratively***, and proved them ***equivalent***

Conclusions

- Developed ***Ex86***: an extensive Intel-x86 ***consistency*** model
  - Memory types (WB, WT, WC, UC)
  - Non-temporal stores
- Formalised Ex86 both ***operationally*** & ***declaratively***, and proved them ***equivalent***
- ***Empirically validated*** Ex86 through extensive testing
- Developed ***PEx86***: an extensive Intel-x86 ***consistency*** and ***persistency*** model
  - Memory types (WB, WT, WC, UC)
  - Non-temporal stores
- Formalised PEx86 both ***operationally*** & ***declaratively***, and proved them ***equivalent***
- Future work
  - Program logics
  - Model checking algorithms

Conclusions

- Developed ***Ex86***: an extensive Intel-x86 ***consistency*** model
  - Memory types (WB, WT, WC, UC)
  - Non-temporal stores
- Formalised Ex86 both ***operationally*** & ***declaratively***, and proved them ***equivalent***
- ***Empirically validated*** Ex86 through extensive testing
- Developed ***PEx86***: an extensive Intel-x86 ***consistency*** and ***persistency*** model
  - Memory types (WB, WT, WC, UC)
  - Non-temporal stores
- Formalised PEx86 both ***operationally*** & ***declaratively***, and proved them ***equivalent***
- Future work
  - Program logics
  - Model checking algorithms

Thank You for Listening!

azalea@imperial.ac.uk SoundAndComplete.org @azalearaad