Beyond Weak Memory Consistency: The Challenges of Memory Persistency

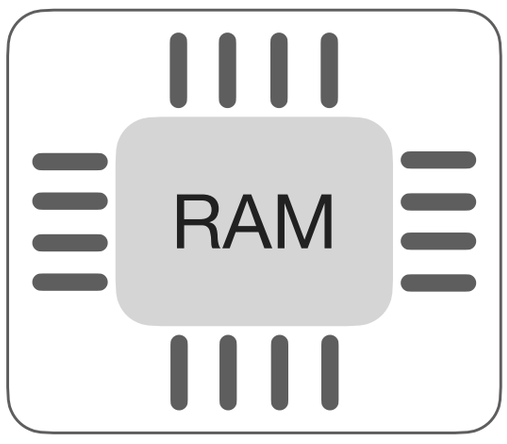
Part I: Low-Level Persistency Models

Azalea Raad
Imperial College London

Viktor Vafeiadis
MPI-SWS

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

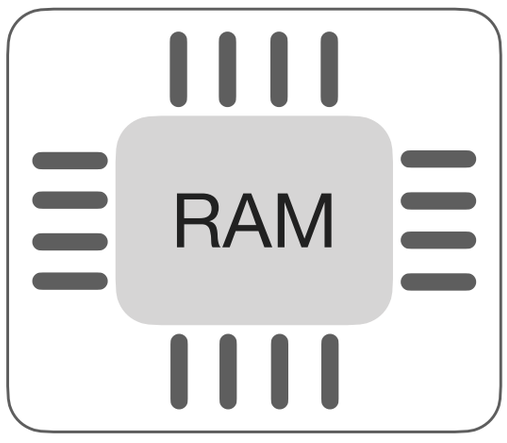
Computer Storage



Computer Storage

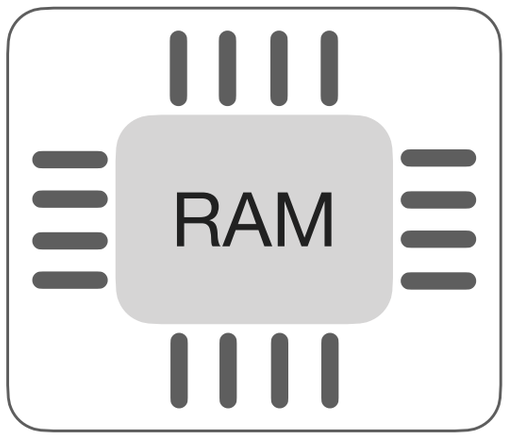
✓ *fast*

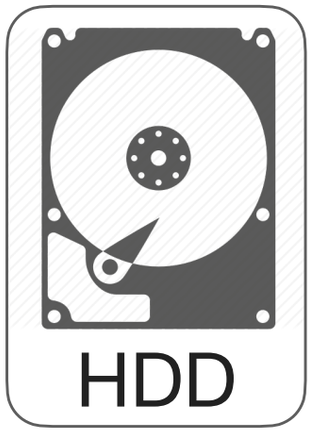
✗ *volatile*



Computer Storage

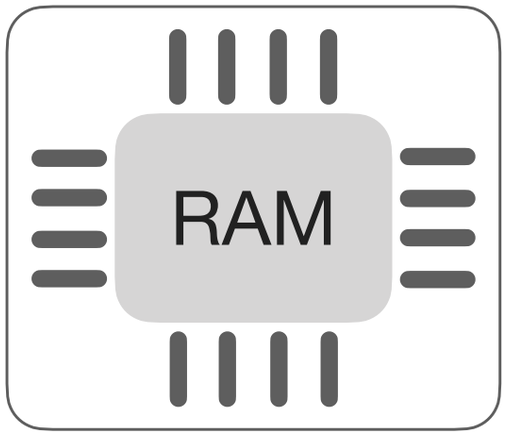
✓ *fast*
✗ *volatile*



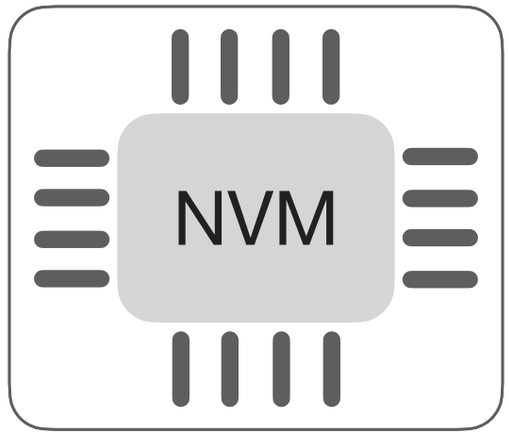


✗ *slow*
✓ *persistent*

What is Non-Volatile Memory (NVM)?



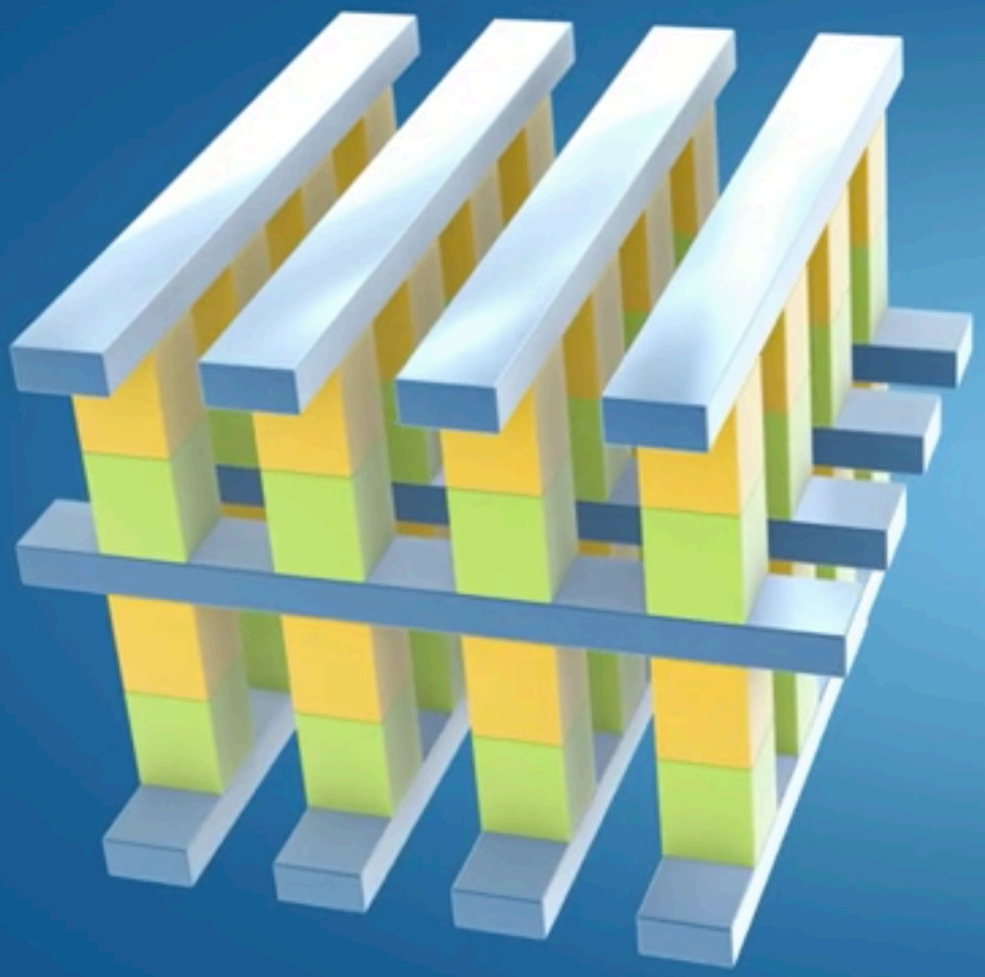
What is Non-Volatile Memory (NVM)?



NVM: Hybrid Storage + Memory

Best of both worlds:

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



INTEL® OPTANE™ TECHNOLOGY

FAST

DENSE

NON-VOLATILE

Q: Why ***Formal*** NVM Semantics?

Volatile memory

```
// x = 0
x := 1
// x = 1
```

Q: Why *Formal* NVM Semantics?

Volatile memory

```
// x = 0
x := 1
// x = 1
```

```
// no recovery
// x = 0
```

Q: Why *Formal* NVM Semantics?

Volatile memory

```
// x = 0
x := 1
// x = 1
```

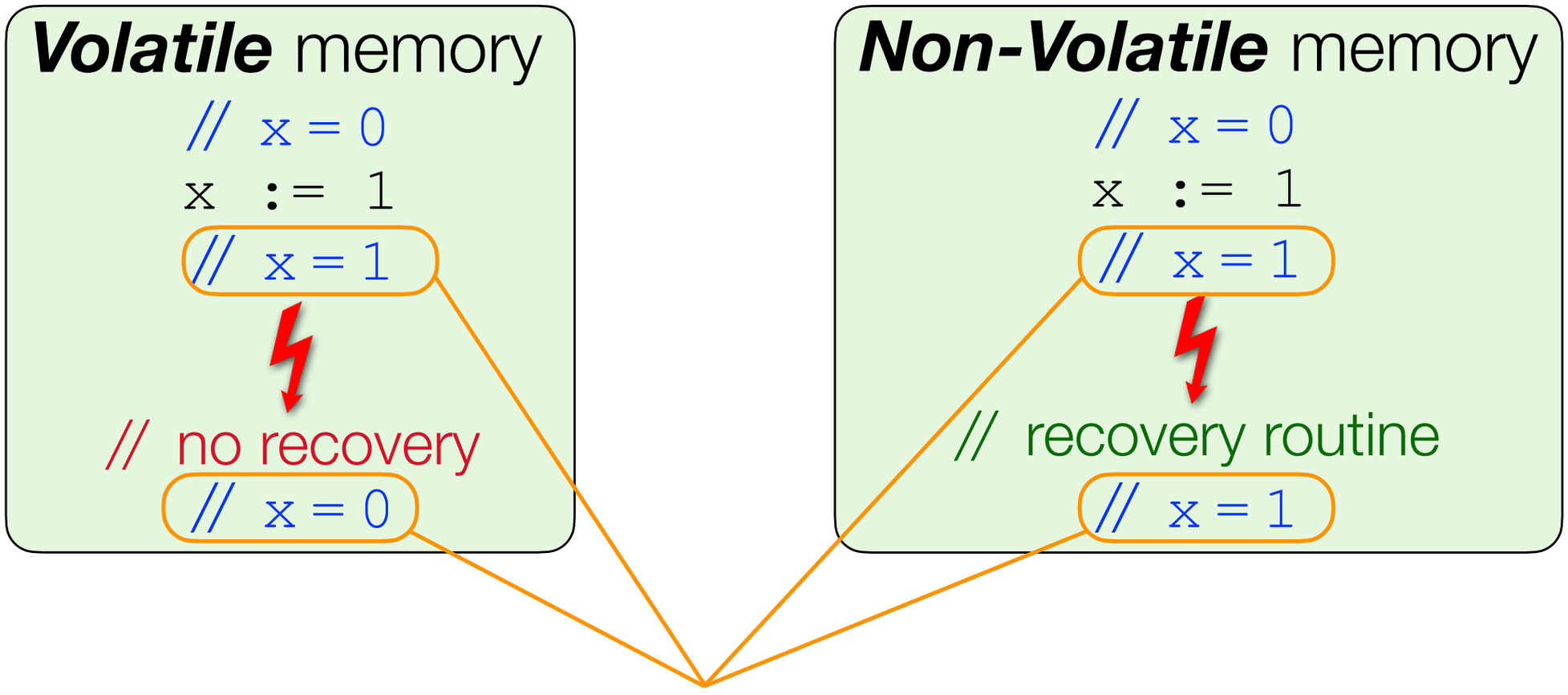
```
// no recovery
// x = 0
```

Non-Volatile memory

```
// x = 0
x := 1
// x = 1
```

```
// recovery routine
// x = 1
```

Q: Why ***Formal*** NVM Semantics?



A: Program ***Verification***

Q: Why ***Formal*** NVM Semantics?

What about ***Concurrency***?

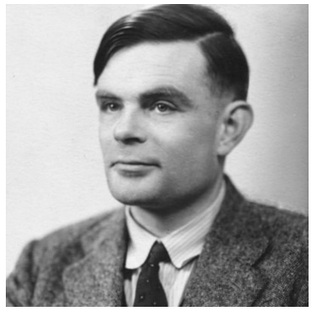
```
// x = y = ... = 0
C1 || C2 || ... || Cn
// ???
```

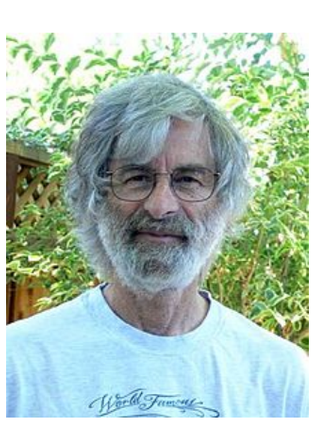
```
// recovery routine
// ???
```

Formal Semantic Models

Formal Semantic Models

Difficulty



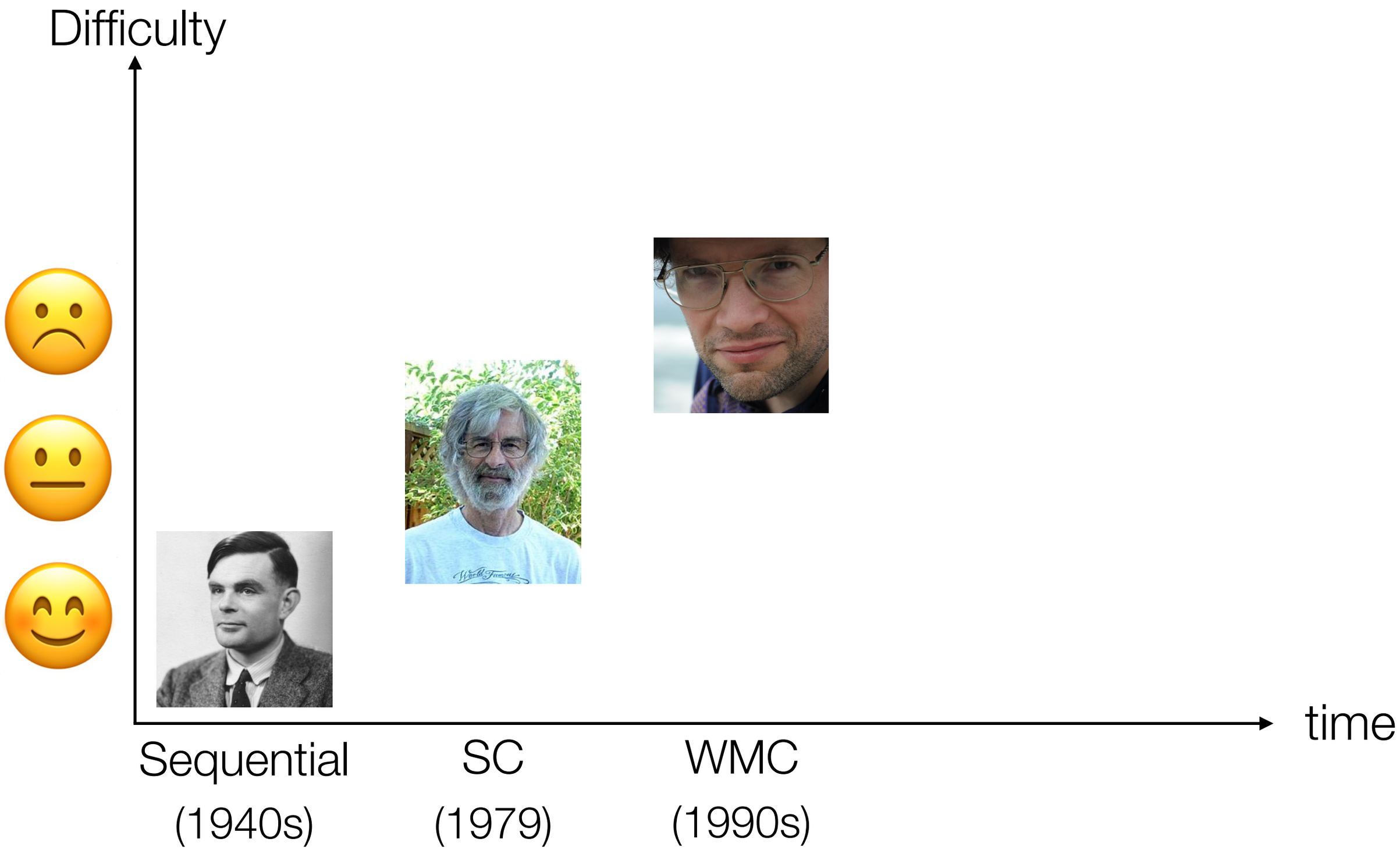


time

Sequential (1940s)

SC (1979)

Formal Semantic Models



Weak Memory Consistency (WMC)

No total execution order (*to*) ⇒

weak behaviour absent under SC, caused by:

- instruction ***reordering*** by compiler
- write propagation across ***cache hierarchy***

WMC: Store Buffering

```
1 x:=1;        ||   3 y:=1;
2 a:=y              4 b:=x
```

a	b
0	1
1	0
1	1

WMC: Store Buffering

```
1 x:=1;   ||   3 y:=1;
2 a:=y    ||   4 b:=x
```

a	b
0	1
1	0
1	1
0	0

possible, due to reordering!

```
2 a:=y;   ||   4 b:=x;
1 x:=1    ||   3 y:=1
```

WMC: Store Buffering

```
1 x:=1;   ||   3 y:=1;
2 a:=y    ||   4 b:=x
```

a	b
0	1
1	0
1	1
0	0

possible, due to reordering!

store buffering(SB)

```
2 a:=y;   ||   4 b:=x;
1 x:=1    ||   3 y:=1
```

Weak Memory Consistency (WMC)

No total execution order (*to*) ⇒

weak behaviour absent under SC, caused by:

- instruction ***reordering*** by compiler
- write propagation across ***cache hierarchy***

Weak Memory Consistency (WMC)

No total execution order (*to*) ⇒

weak be

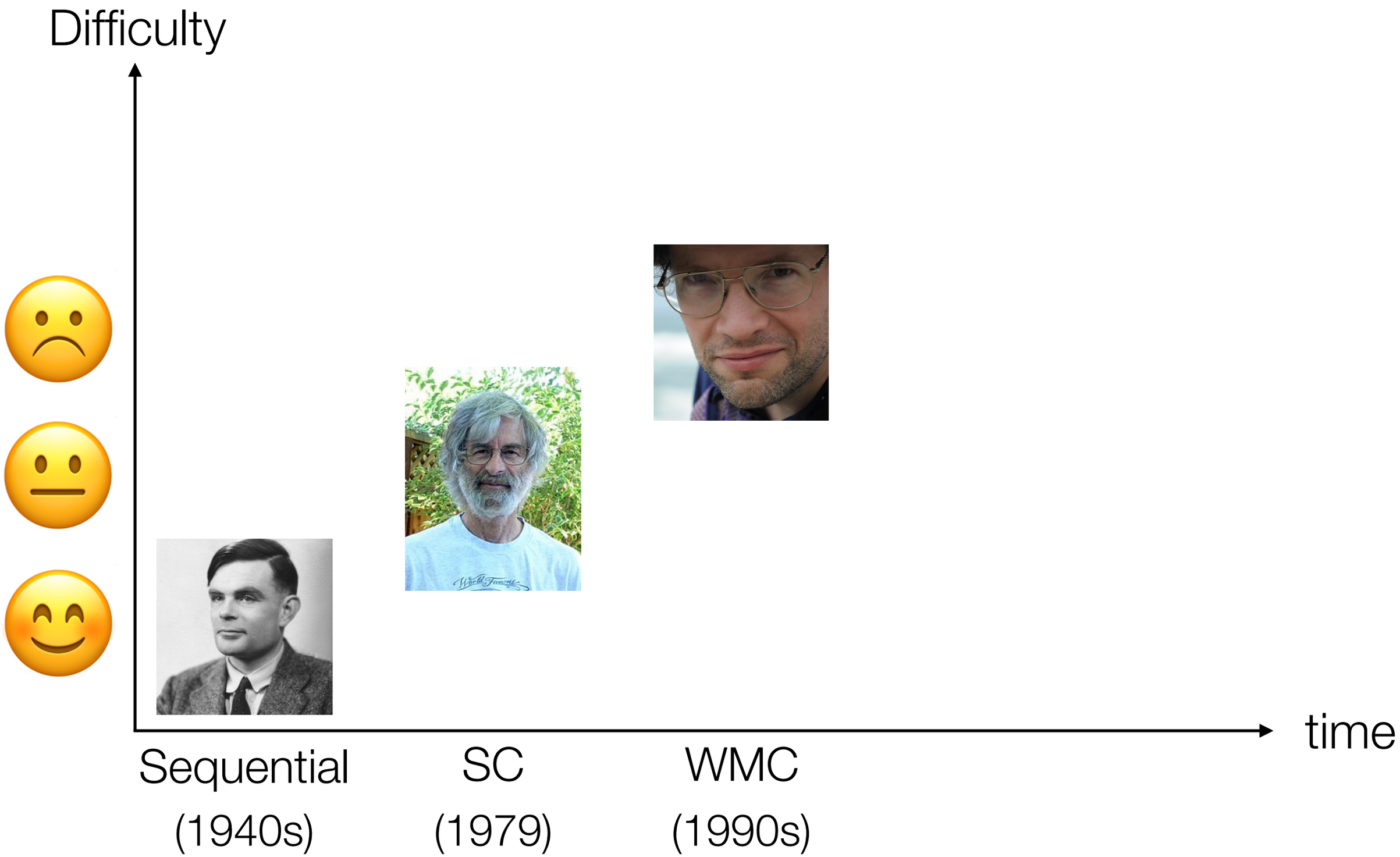
- instru
- write

Consistency Model

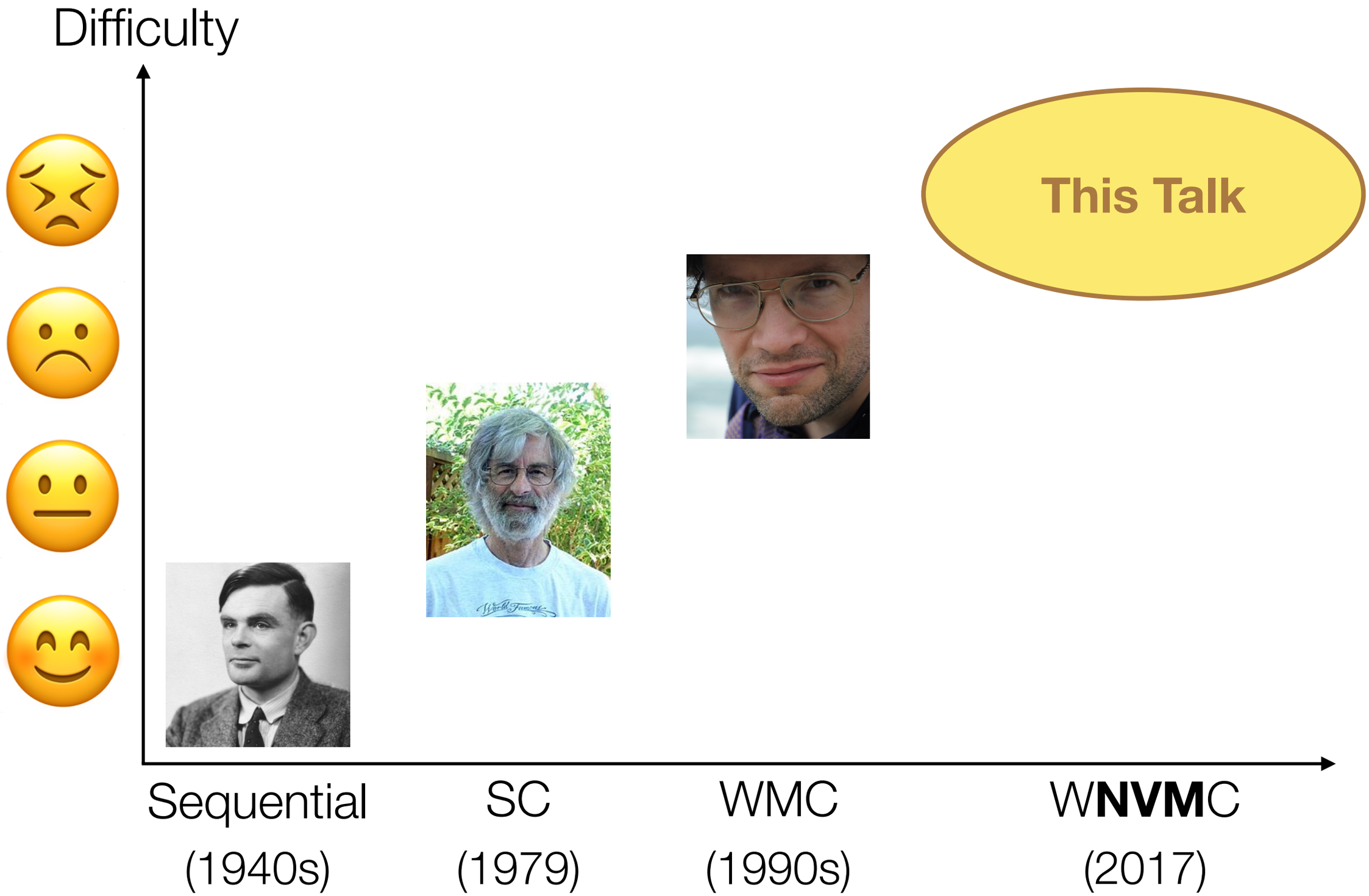
the ***order*** in which
writes are made visible
to other threads

e.g. x86 (TSO), ARMv8, C11, Java

Formal Semantic Models



Formal Semantic Models



What Can Go Wrong?

```
// x=y=0

x := 1;

y := 1;
```

```
// recovery routine
```

What Can Go Wrong?

```
// x=y=0

   x := 1;

   y := 1;
```

```
// recovery routine

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

What Can Go Wrong?

```
// x=y=0

x := 1;

y := 1;
```

```
// recovery routine

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

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

— ***asynchronous*** persists

What Can Go Wrong?

```
// x=y=0

x := 1;

y := 1;
```

```
// recovery routine

// x=y=1   OR   x=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***

— ***relaxed*** persists

What Can Go Wrong?

Consistency Model

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

```
// x=
```

```
y=1
```

!! Ex

—

!! W

—

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

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

NVM Semantics

Consistency + Persistency Model

Outline

1. An *intuitive* account of Intel-x86 persistency: Px86
   - ✢ Warmup*:* ***Sequential*** Px86
   - ✢ ***Concurrent*** Px86

2. A *formal* account Px86: ***operational*** semantics

3. A *formal* account Px86: ***declarative*** semantics

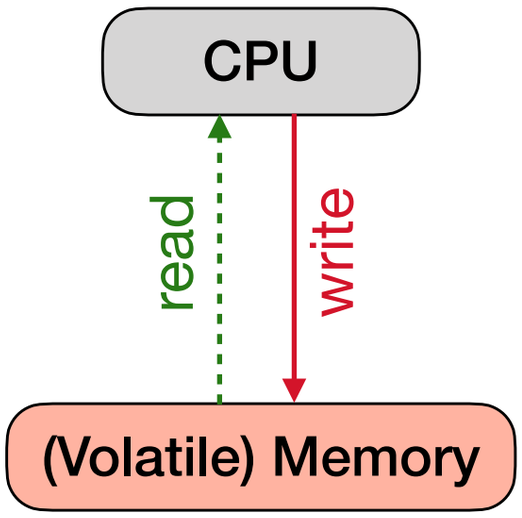
4. Other Low-level (hardware) persistency models

5. Further reading

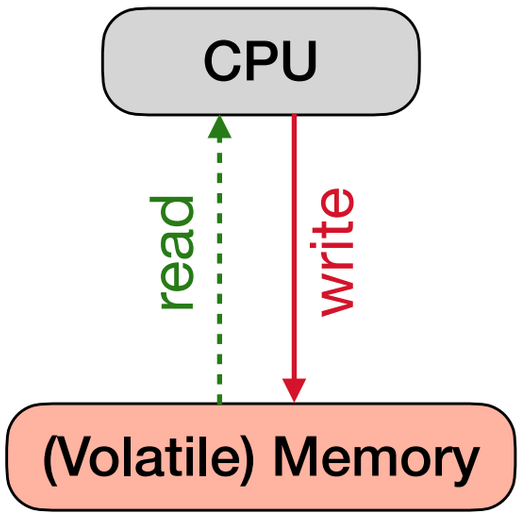
1. An *intuitive* account of Px86

Warmup: ***Sequential*** Px86

Sequential Hardware

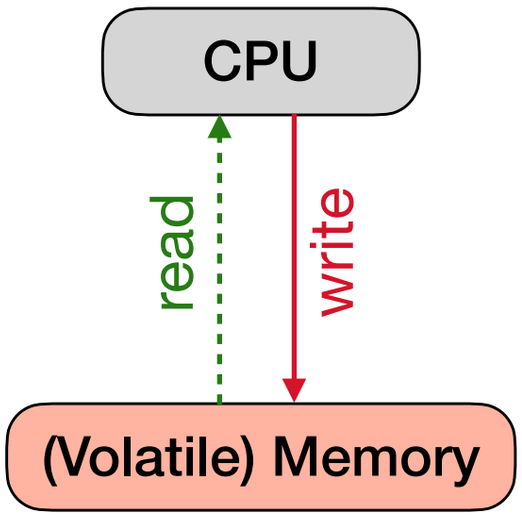


Sequential Hardware



`x:=1` : adds `x:=1` to memory

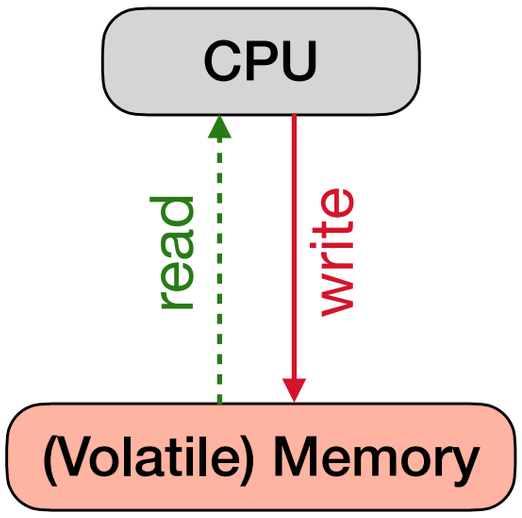
Sequential Hardware



`x:=1` : adds `x:=1` to memory

`a:=x` : reads `x` from memory

Sequential Hardware

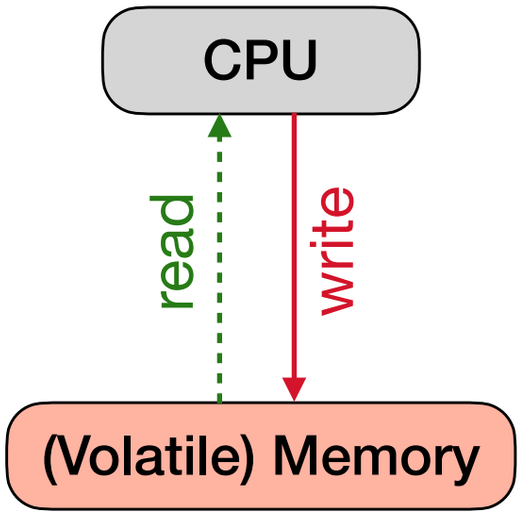


`x:=1` : adds `x:=1` to memory

`a:=x` : reads `x` from memory

memory lost

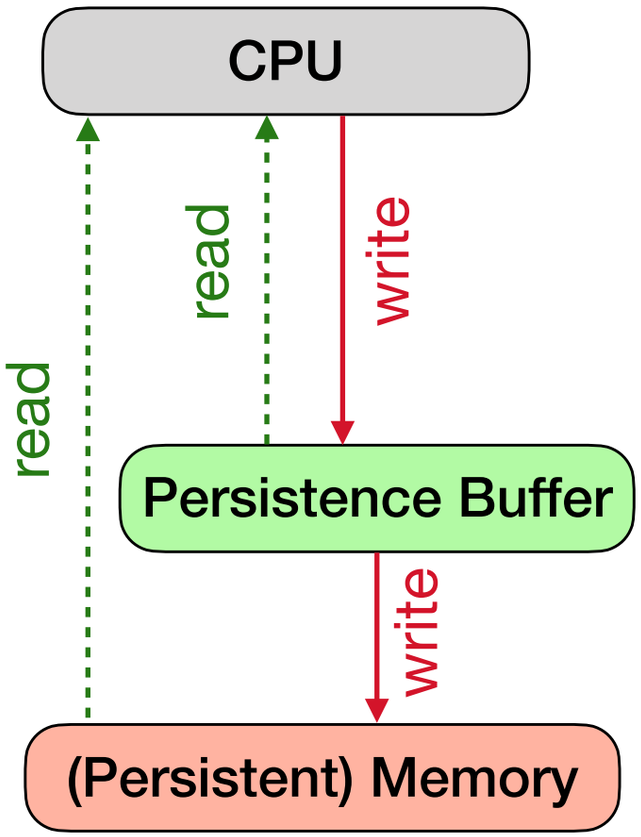
Sequential Hardware



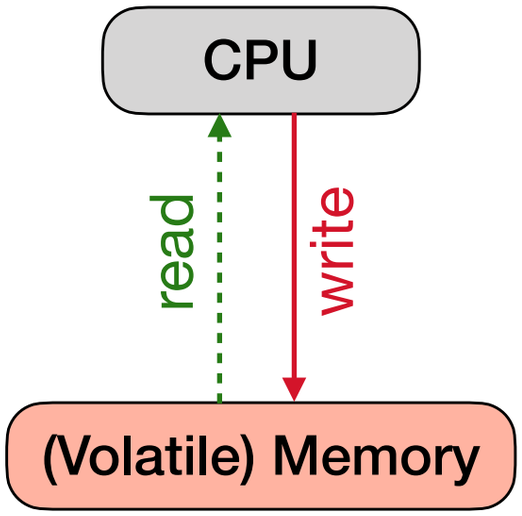
`x:=1` : adds `x:=1` to memory

`a:=x` : reads `x` from memory

memory lost



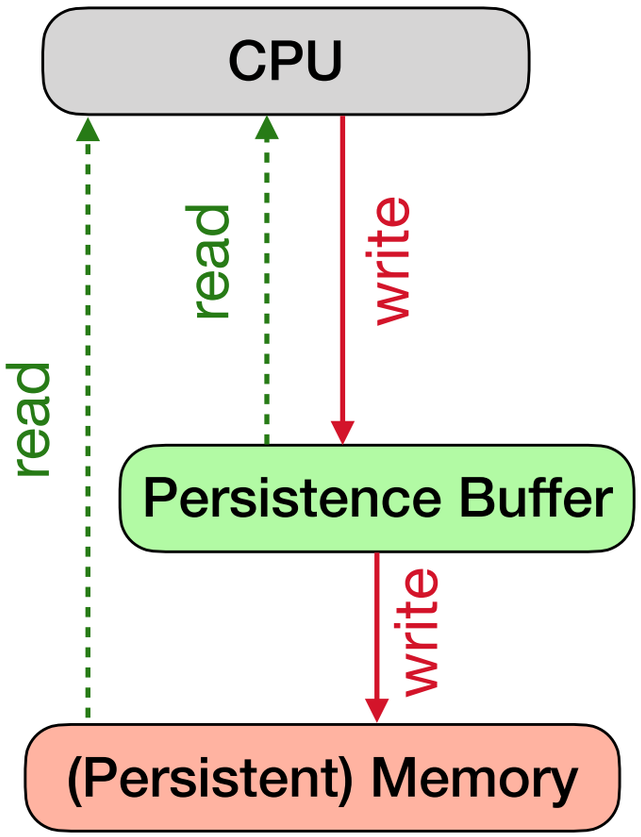
Sequential Hardware



`x:=1` : adds `x:=1` to memory

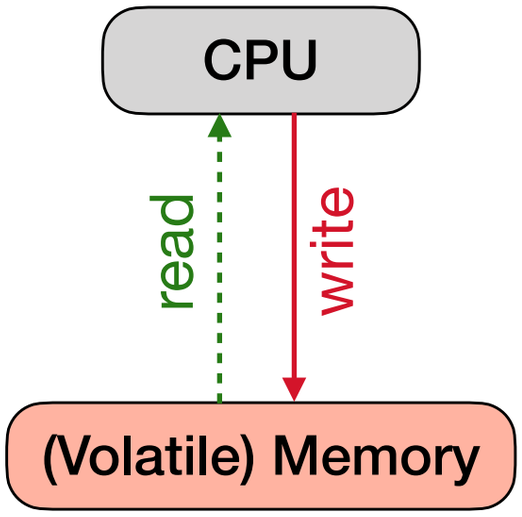
`a:=x` : reads `x` from memory

memory lost



`x:=1` : adds `x:=1` to p-buffer

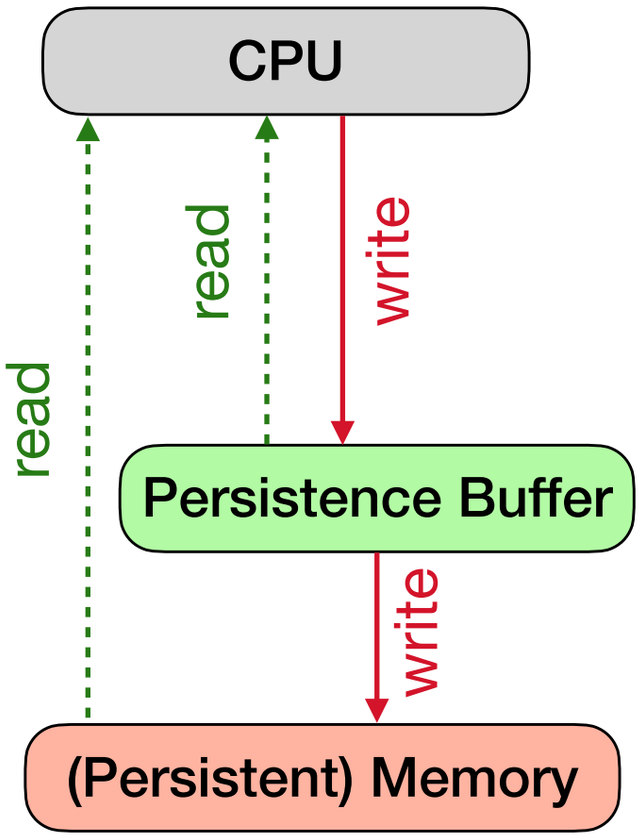
Sequential Hardware



`x:=1` : adds `x:=1` to memory

`a:=x` : reads `x` from memory

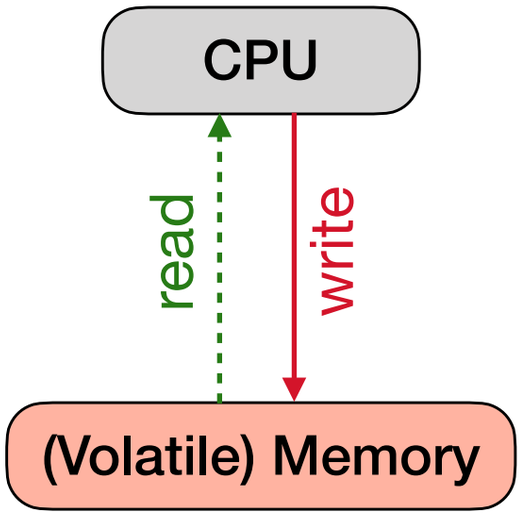
memory lost



`x:=1` : adds `x:=1` to p-buffer

`a:=x` : if p-buffer contains `x`, reads latest entry
else reads from memory

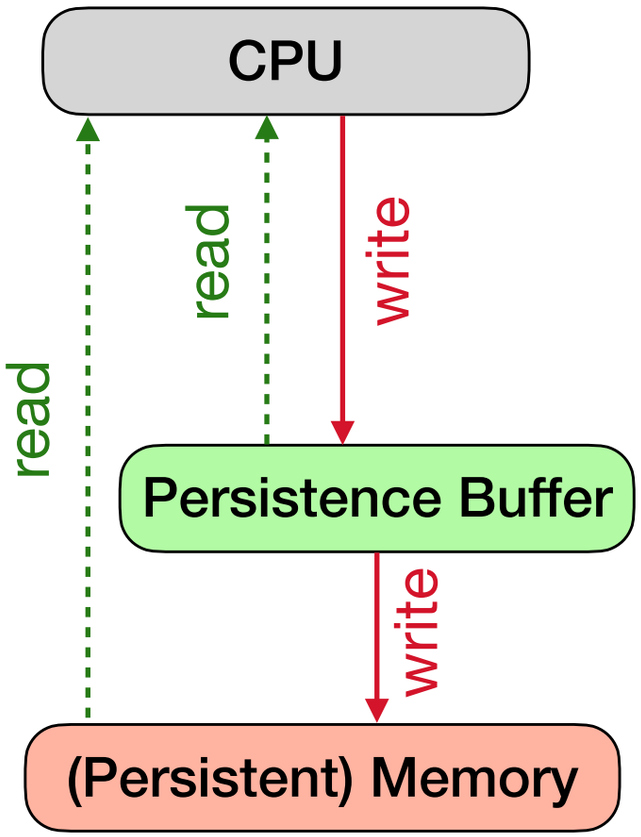
Sequential Hardware



`x:=1` : adds `x:=1` to memory

`a:=x` : reads `x` from memory

memory lost

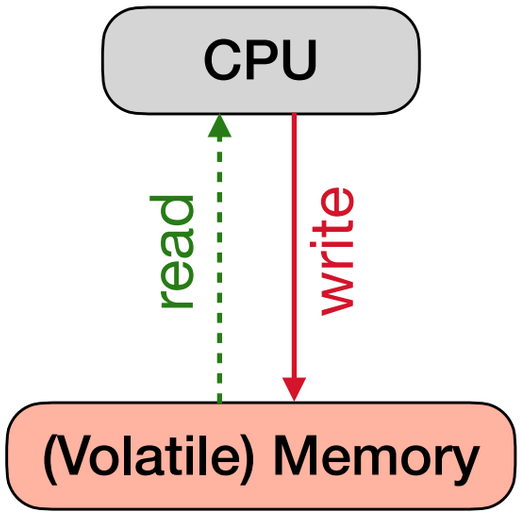


`x:=1` : adds `x:=1` to p-buffer

`a:=x` : if p-buffer contains `x`, reads latest entry
else reads from memory

p-buffer lost; memory ***retained***

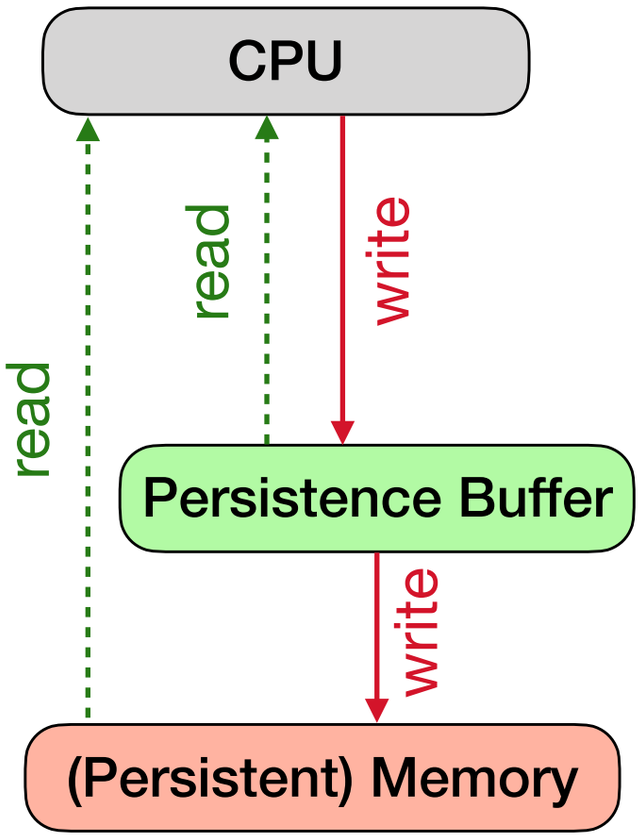
Sequential Hardware



`x:=1` : adds `x:=1` to memory

`a:=x` : reads `x` from memory

memory lost



`x:=1` : adds `x:=1` to p-buffer

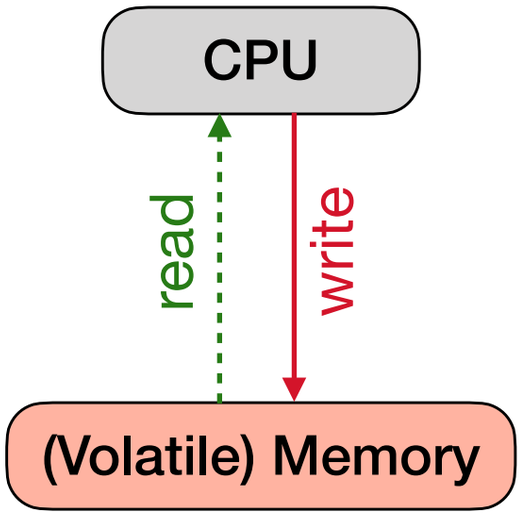
`a:=x` : if p-buffer contains `x`, reads latest entry
else reads from memory

p-buffer lost; memory ***retained***

unbuffer* : p-buffer to memory

* at non-deterministic times

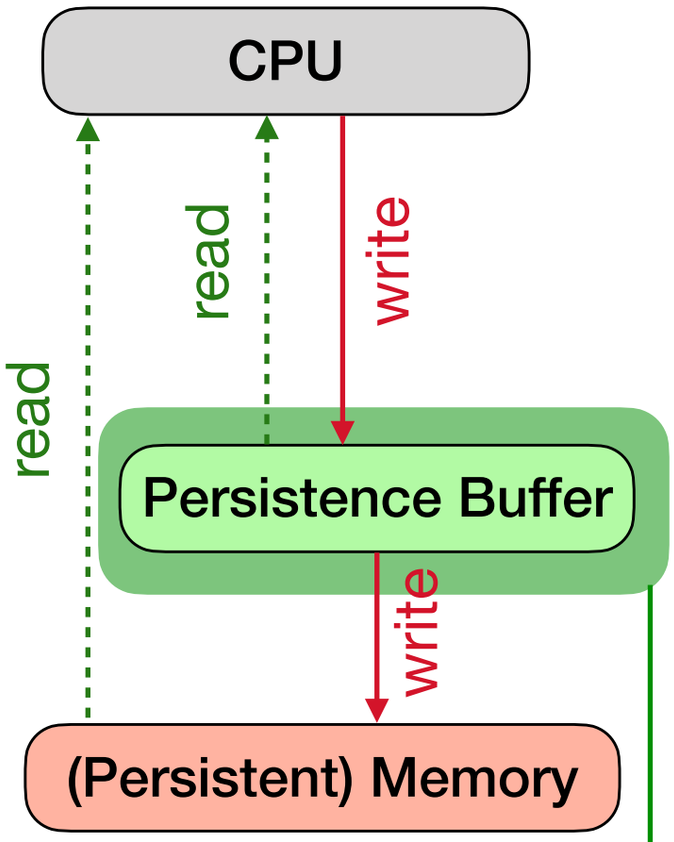
Sequential Hardware



`x:=1` : adds `x:=1` to memory

`a:=x` : reads `x` from memory

memory lost



`x:=1` : adds `x:=1` to p-buffer

`a:=x` : if p-buffer contains `x`, reads latest entry
else reads from memory

p-buffer lost; memory ***retained***

unbuffer* : p-buffer to memory

Unbuffered at ***non-deterministic*** points in time

Buffering & unbuffering orders may disagree

* at non-deterministic times

Handling ***Relaxed*** Persists

```
// x=0;y=0

x := 1;

y := 1;
```

```
// recovery routine

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

!! *out of order* *persists*

☛ ***explicit persists?***

Explicit Persists: ***Desiderata***

```
// x=0;y=0

    x := 1;
☛   persist x;
    y := 1;
```

```
// recovery routine

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

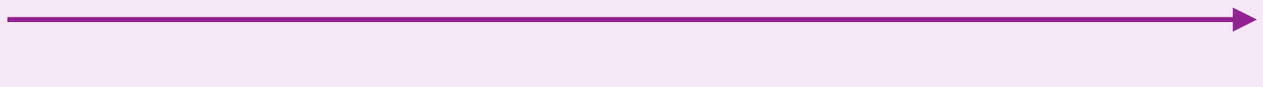
!! *out of order* *persists*

☛ ***explicit persists?***

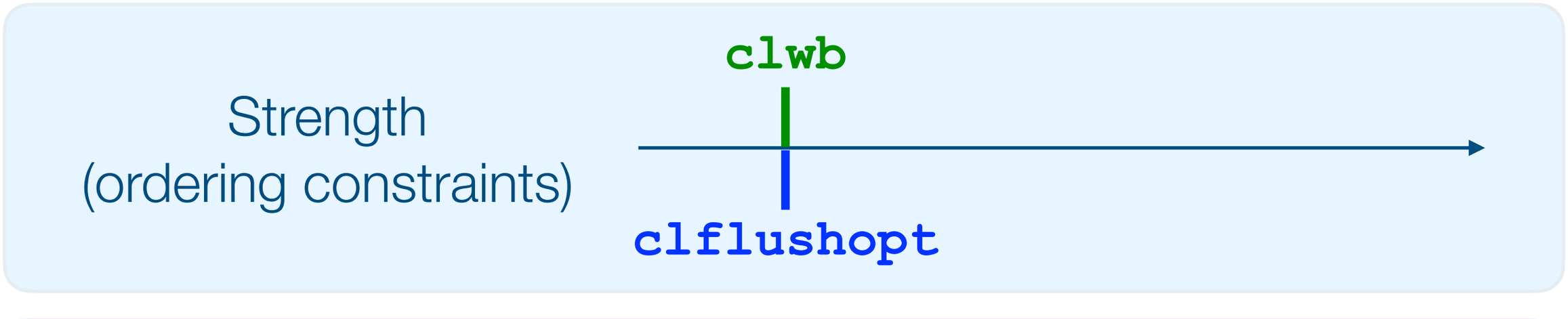
x86 Persists: `clwb`, `clflushopt`, `clflush`

Strength
(ordering constraints)

Performance

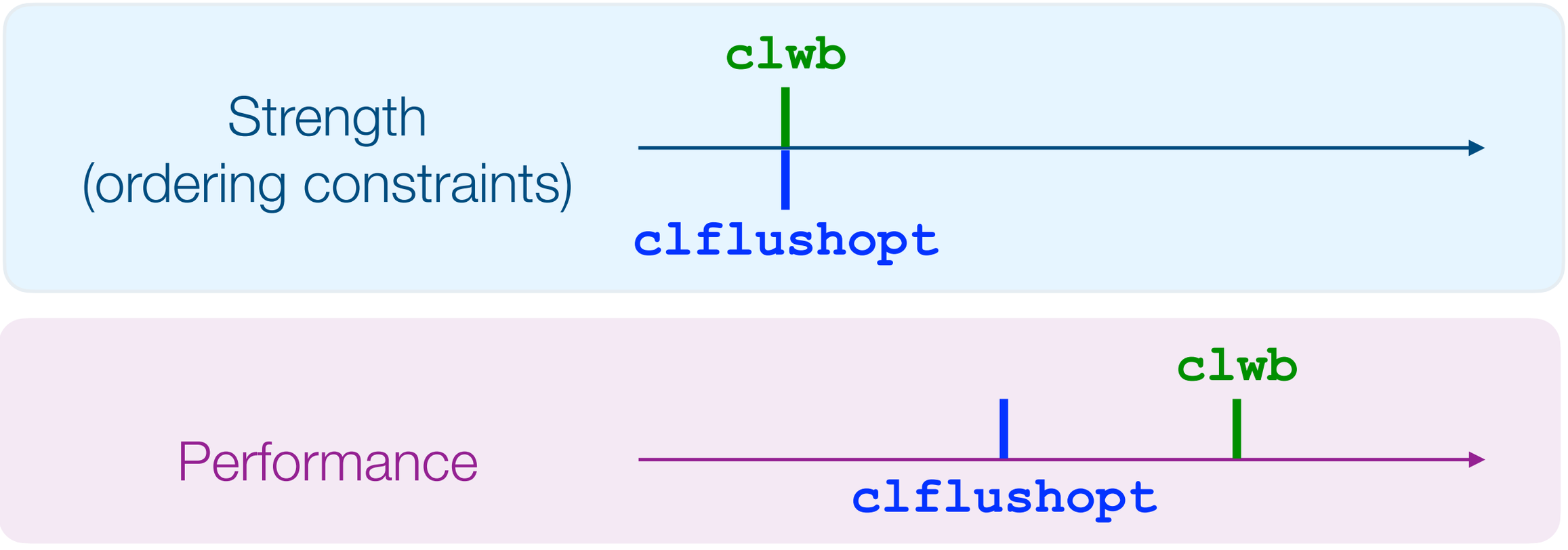


x86 Persists: `clwb`, `clflushopt`, `clflush`



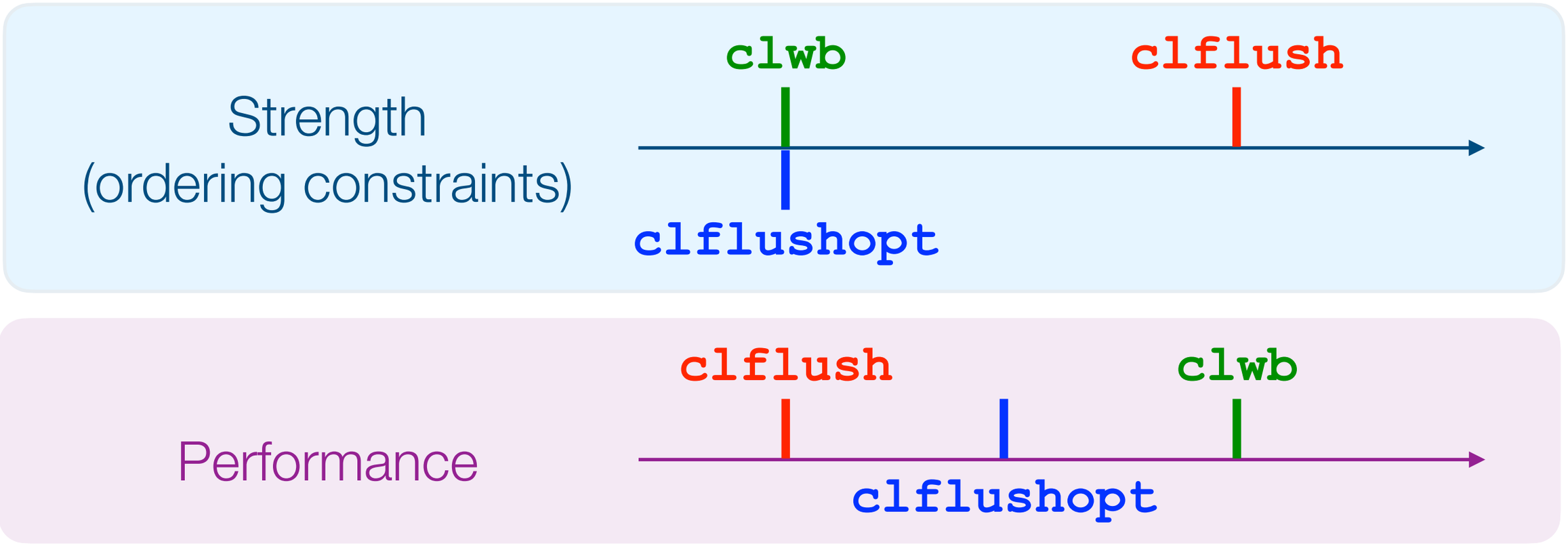
✤ `clwb` and `clflushopt`: **same ordering** constraints

x86 Persists: `clwb`, `clflushopt`, `clflush`



- `clwb` and `clflushopt`: **same ordering** constraints
- `clwb` **does not invalidate** cache line
- `clflushopt` **invalidates** cache line

x86 Persists: `clwb`, `clflushopt`, `clflush`



- `clwb` and `clflushopt`: **same ordering** constraints
- `clwb` **does not invalidate** cache line
- `clflushopt` **invalidates** cache line
- `clflush`: **strongest** ordering constraints; **invalidates** cache line

Strong (Synchronous) Explicit Persists: **clflush**

```
// x=0;y=0

  x := 1;
☛ clflush x;
  y := 1;
```

```
// recovery routine

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

Weak (Asynchronous) Explicit Persists: **`clflushopt`** & **`clwb`**

```
// x=0;y=0

    x := 1;
    clflushopt x / clwb x;
    y := 1;

// recovery routine

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

Weak (Asynchronous) Explicit Persists: **`clflushopt`** & **`clwb`**

```
// x=0;y=0

x := 1;
clflushopt x / clwb x;
y := 1;
```

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

```
=0;y=1
```

weak explicit persists of x86 are ***asynchronous*** and can themselves ***persist out of order*** !

Solution: *Persist Sequences*

```
// x=0;y=0

   x := 1;
   clflushopt x/clwb x;
   sfence/mfence/RMW;
   y := 1;
```

```
// recovery routine

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

Solution: *Persist Sequences*

```
// x=0;y=0

x := 1;
clflushopt x/clwb x;
sfence/mfence/RMW;
y := 1;
```

```
// recovery routine

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

✣ ***Waits*** until earlier writes on **x** are persisted

✓ ***synchronous*** persists

Solution: *Persist Sequences*

```
// x=0;y=0

x := 1;
clflushopt x/clwb x;
sfence/mfence/RMW;
y := 1;

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

- ***Waits*** until earlier writes on **x** are persisted
- ***Disallows reordering***

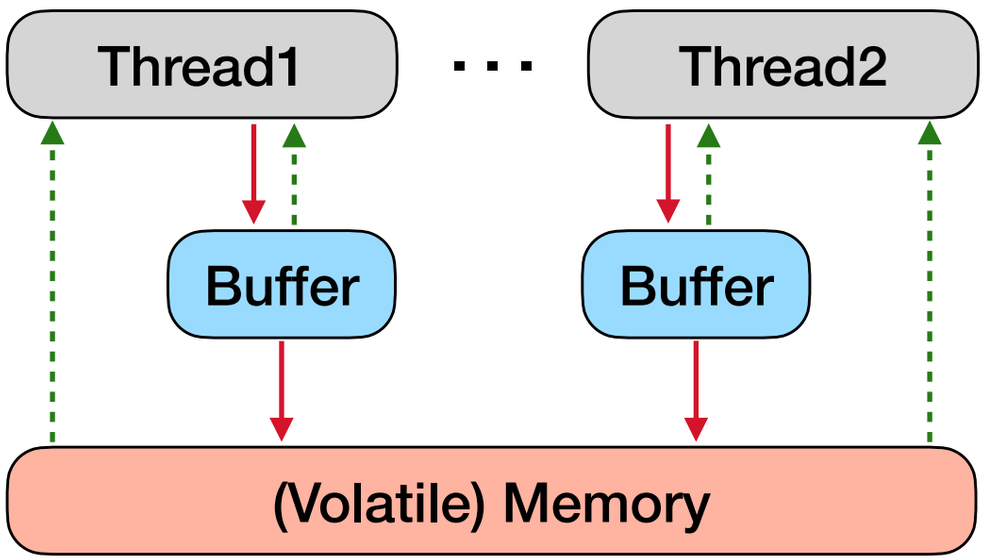
✓ ***synchronous*** persists

✓ ***no out of order*** persists

1. An *intuitive* account of Px86

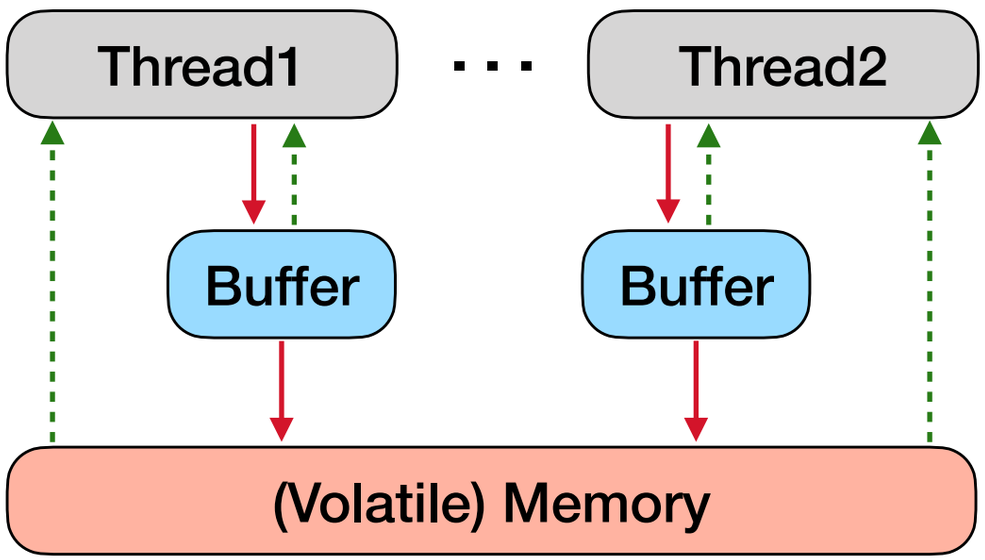
Concurrent Px86

x86: (Volatile) Concurrent Hardware Model (TSO)

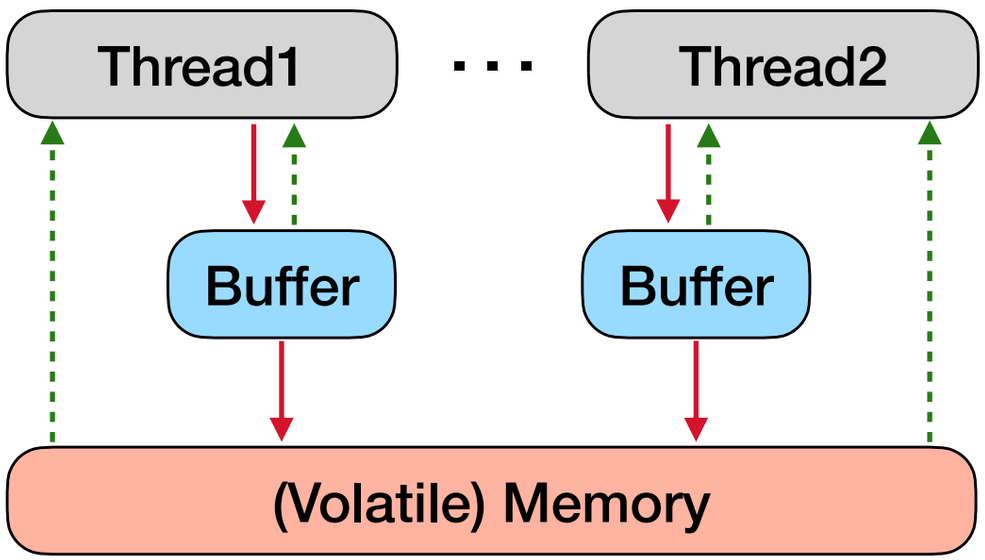


x86: (Volatile) Concurrent Hardware Model (TSO)

`x:=1` : adds `x:=1` to buffer



x86: (Volatile) Concurrent Hardware Model (TSO)

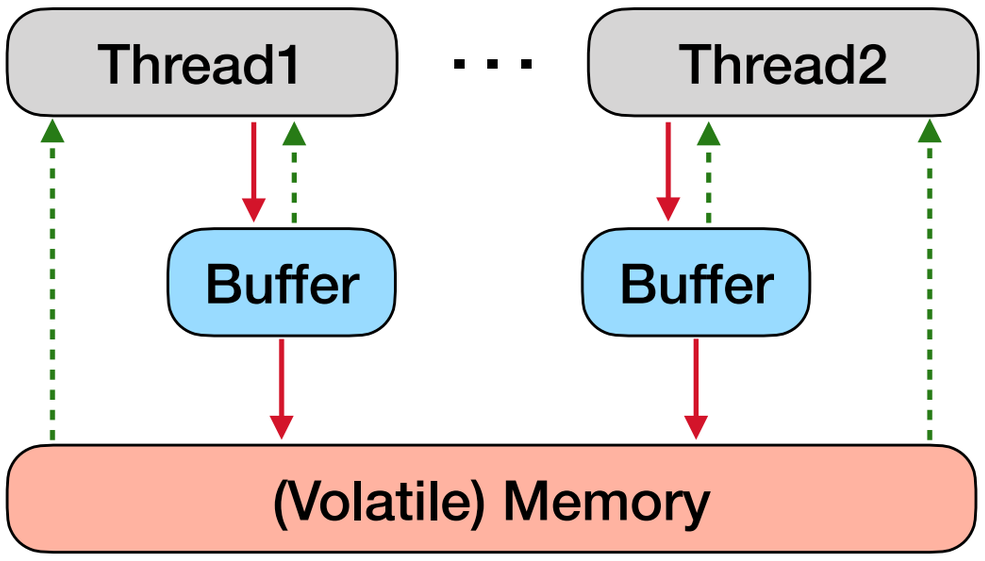


`x:=1` : adds `x:=1` to buffer

unbuffer* : buffer to memory

* at non-deterministic times

x86: (Volatile) Concurrent Hardware Model (TSO)



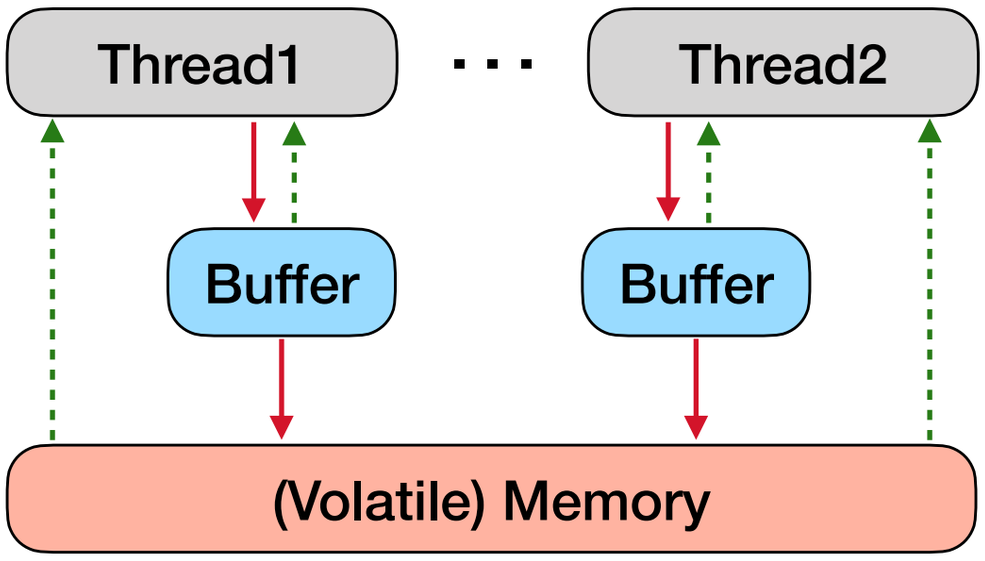
`x:=1` : adds `x:=1` to buffer

unbuffer* : buffer to memory

`a:=x` : if buffer contains `x`, reads latest entry
else reads from memory

* at non-deterministic times

x86: (Volatile) Concurrent Hardware Model (TSO)



`x:=1` : adds `x:=1` to buffer

unbuffer* : buffer to memory

`a:=x` : if buffer contains `x`, reads latest entry
else reads from memory

buffer and memory lost

* at non-deterministic times

Software WMC: Store Buffering

```
1 x:=1;   ||   3 y:=1;
2 a:=y    ||   4 b:=x
```

a	b
0	1
1	0
1	1
0	0

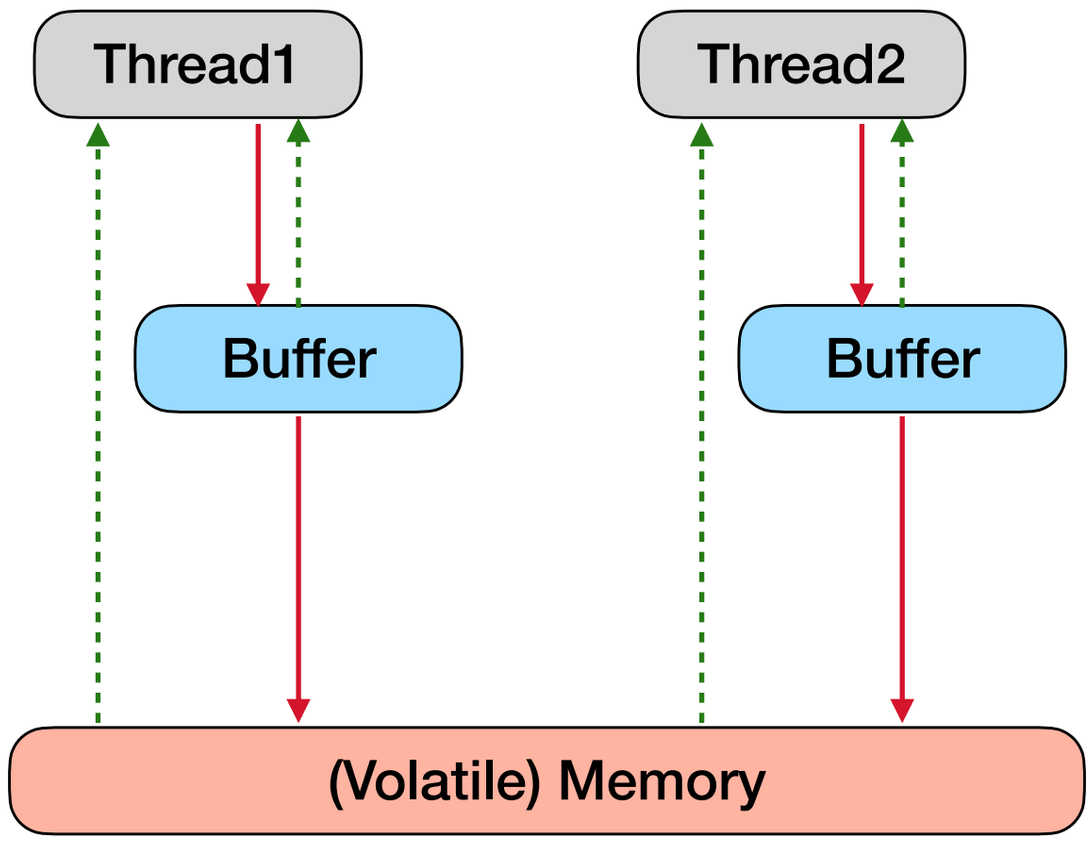
possible, due to reordering!

store buffering(SB)

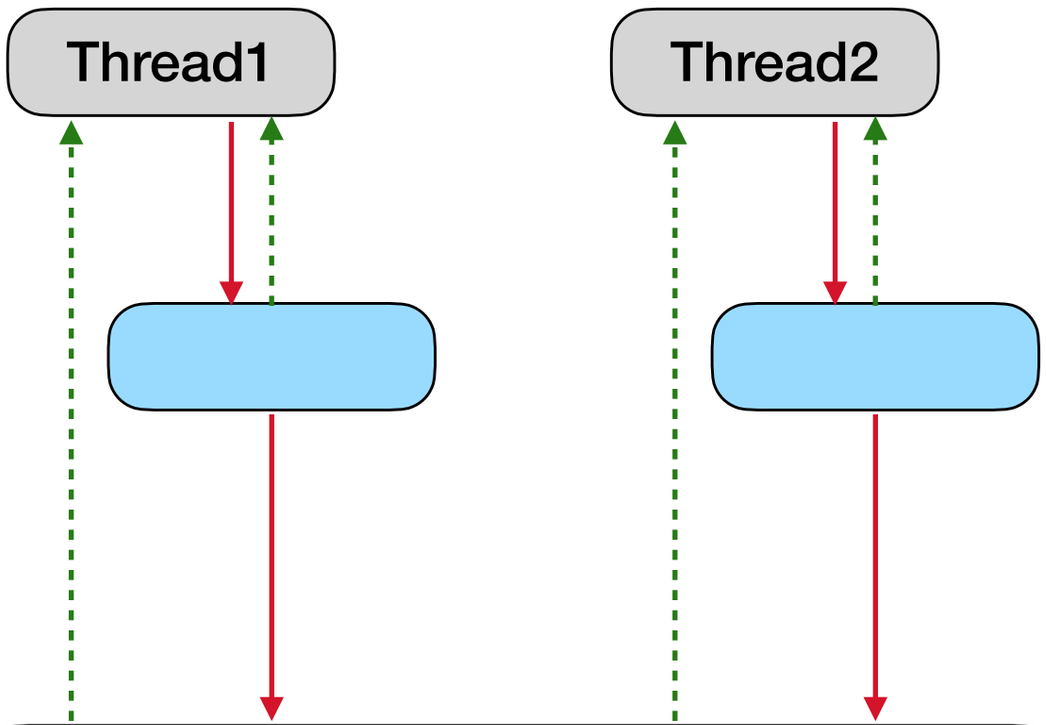
```
2 a:=y;   ||   4 b:=x;
1 x:=1    ||   3 y:=1
```

Hardware (Intel x86) WMC: Store Buffering

Hardware (Intel x86) WMC: Store Buffering



Hardware (Intel x86) WMC: Store Buffering



x = 0; y = 0;

Thread1

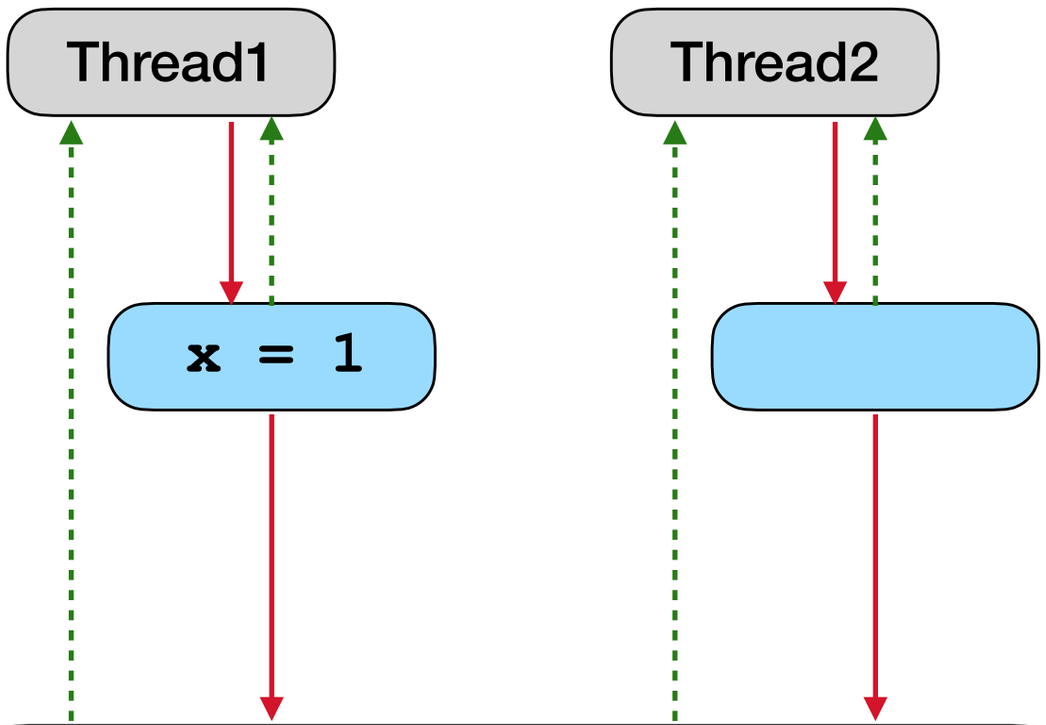
```
x := 1;
a := y;
```

Thread2

```
y := 1;
b := x;
```

Store Buffering (SB)

Hardware (Intel x86) WMC: Store Buffering

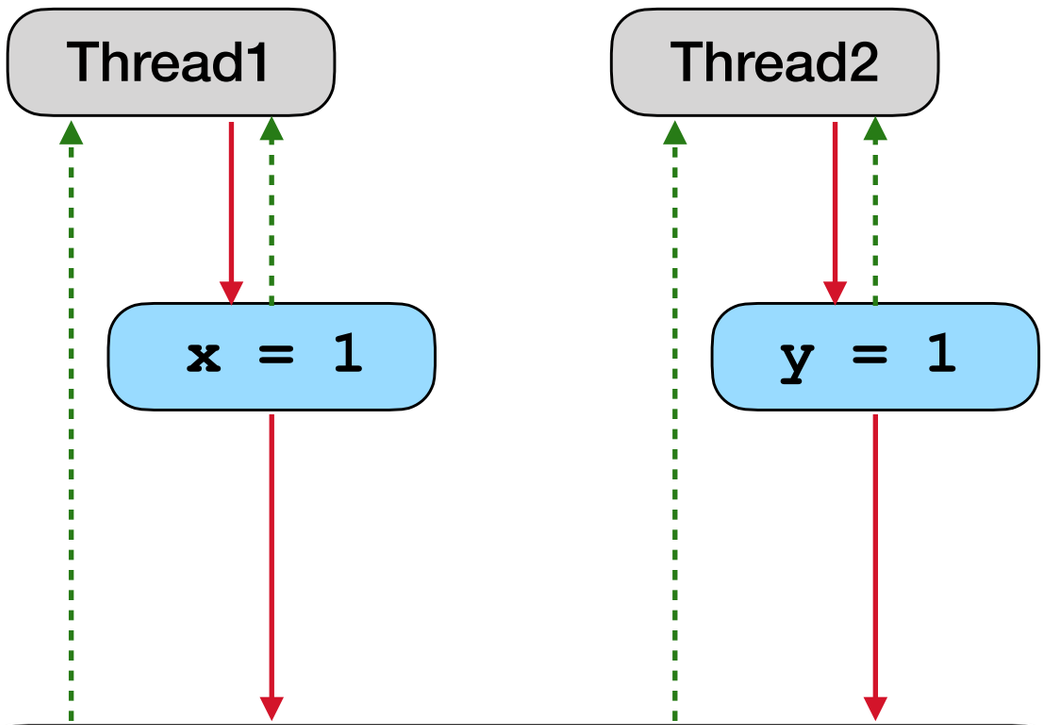


x = 0; y = 0;

Thread1	Thread2
x := 1;	☛ y := 1;
☛ a := y;	b := x;

Store Buffering (SB)

Hardware (Intel x86) WMC: Store Buffering

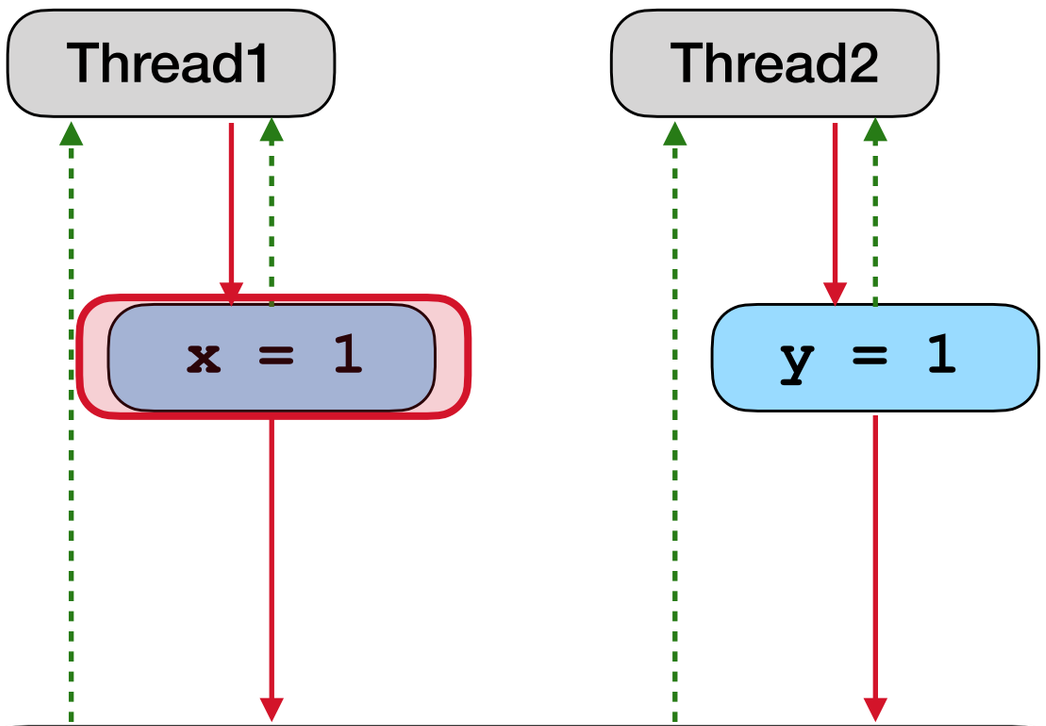


x = 0; y = 0;

Thread1	Thread2
x := 1;	y := 1;
☛ a := y;	☛ b := x;

Store Buffering (SB)

Hardware (Intel x86) WMC: Store Buffering

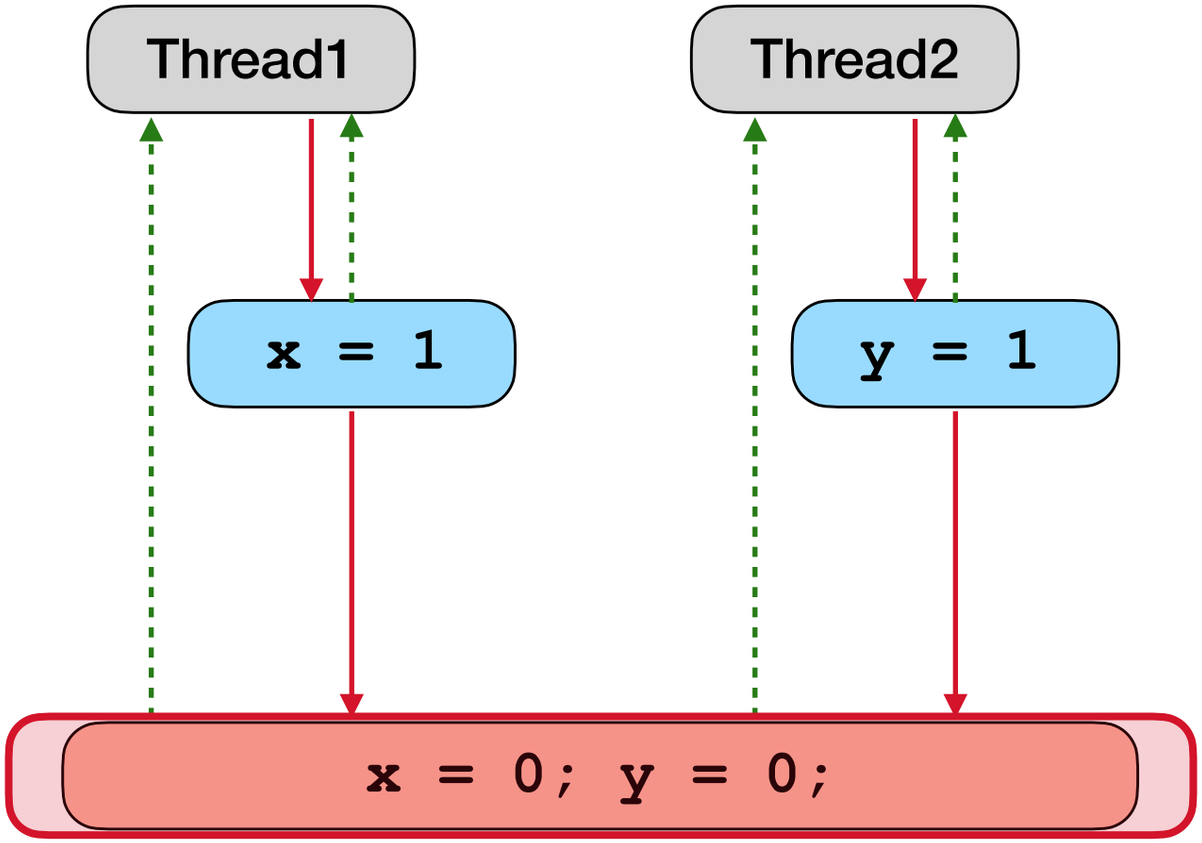


x = 0; y = 0;

Thread1	Thread2
x := 1;	y := 1;
☛ a := y;	☛ b := x;

Store Buffering (SB)

Hardware (Intel x86) WMC: Store Buffering



Thread1

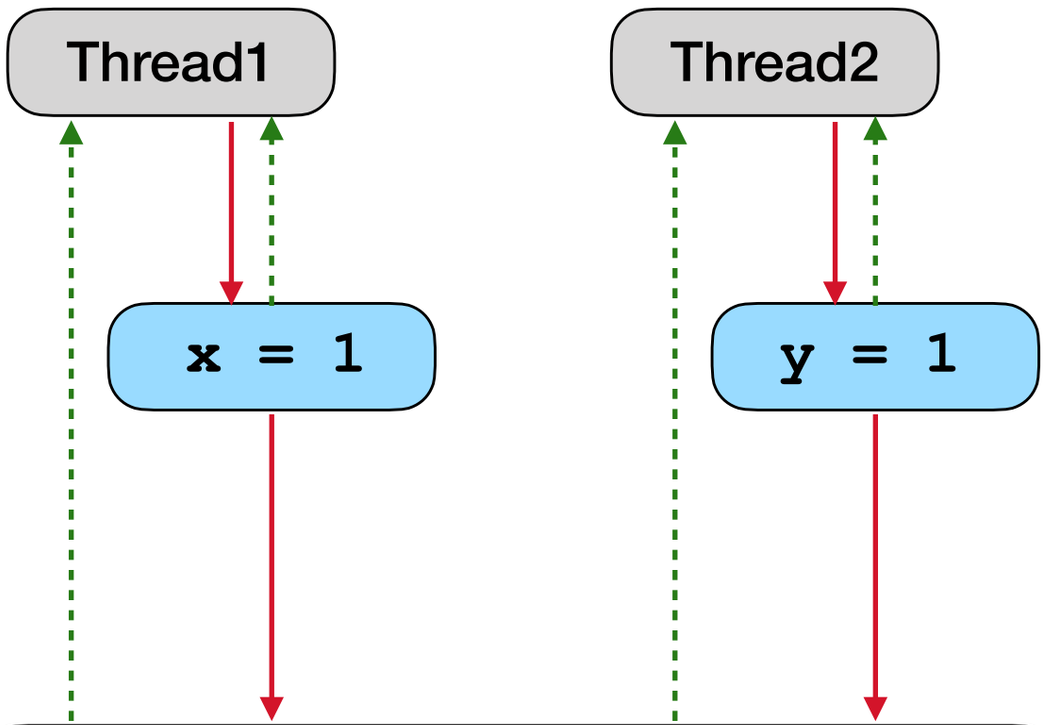
```
    x := 1;
    a := y;
```

Thread2

```
    y := 1;
    b := x;
```

Store Buffering (SB)

Hardware (Intel x86) WMC: Store Buffering



x = 0; y = 0;

Thread1

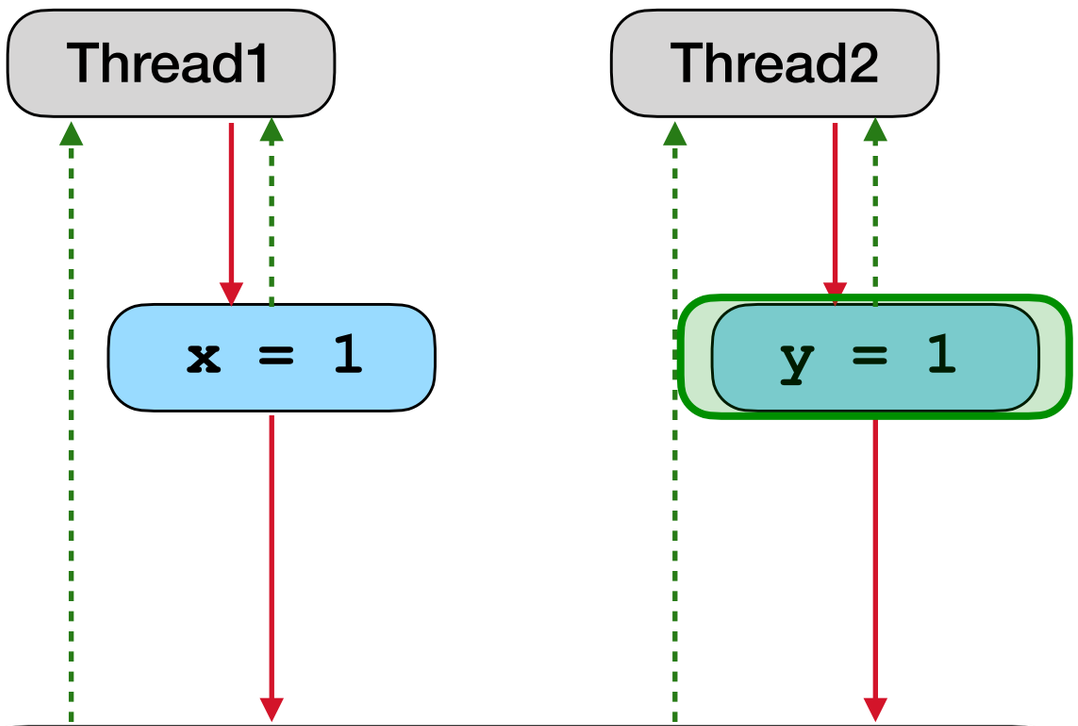
```
x := 1;
a := y; // 0
```

Thread2

```
y := 1;
b := x;
```

Store Buffering (SB)

Hardware (Intel x86) WMC: Store Buffering



x = 0; y = 0;

Thread1

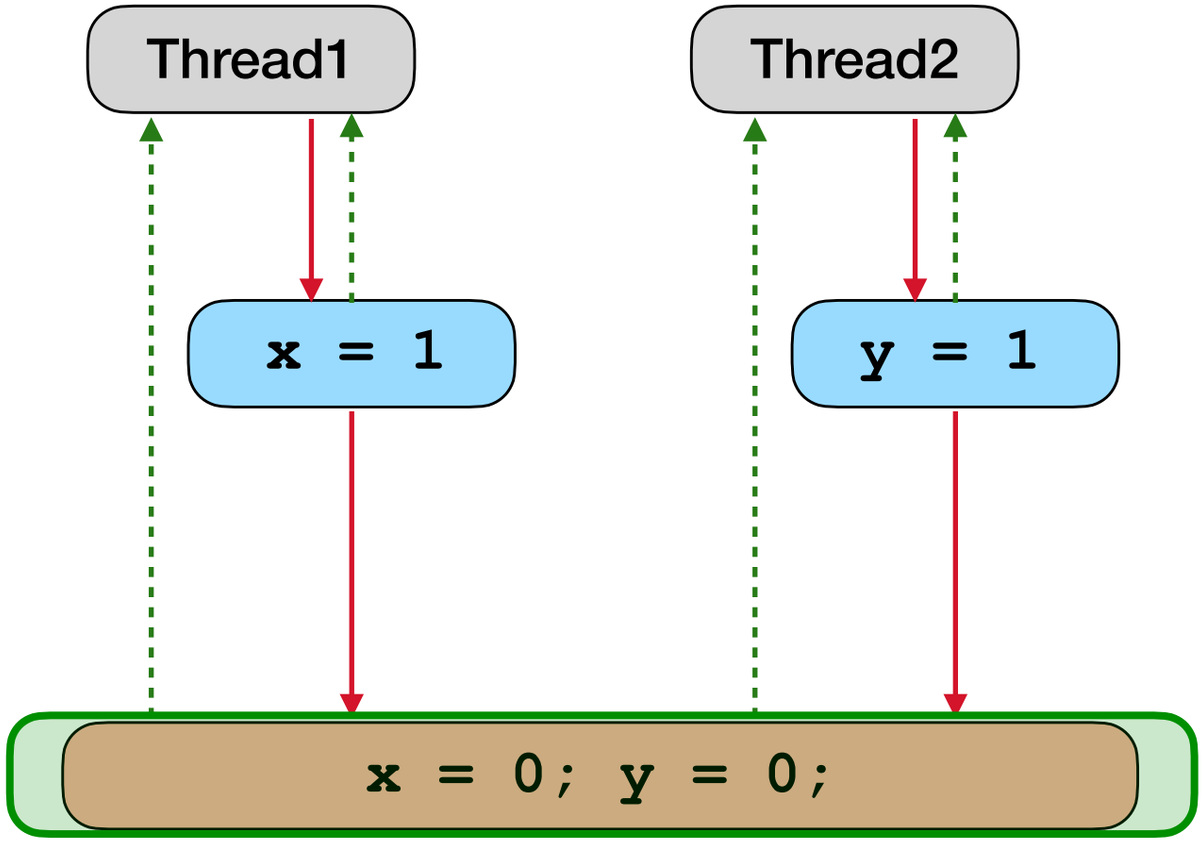
```
x := 1;
a := y; // 0
```

Thread2

```
y := 1;
b := x;
```

Store Buffering (SB)

Hardware (Intel x86) WMC: Store Buffering



Thread1

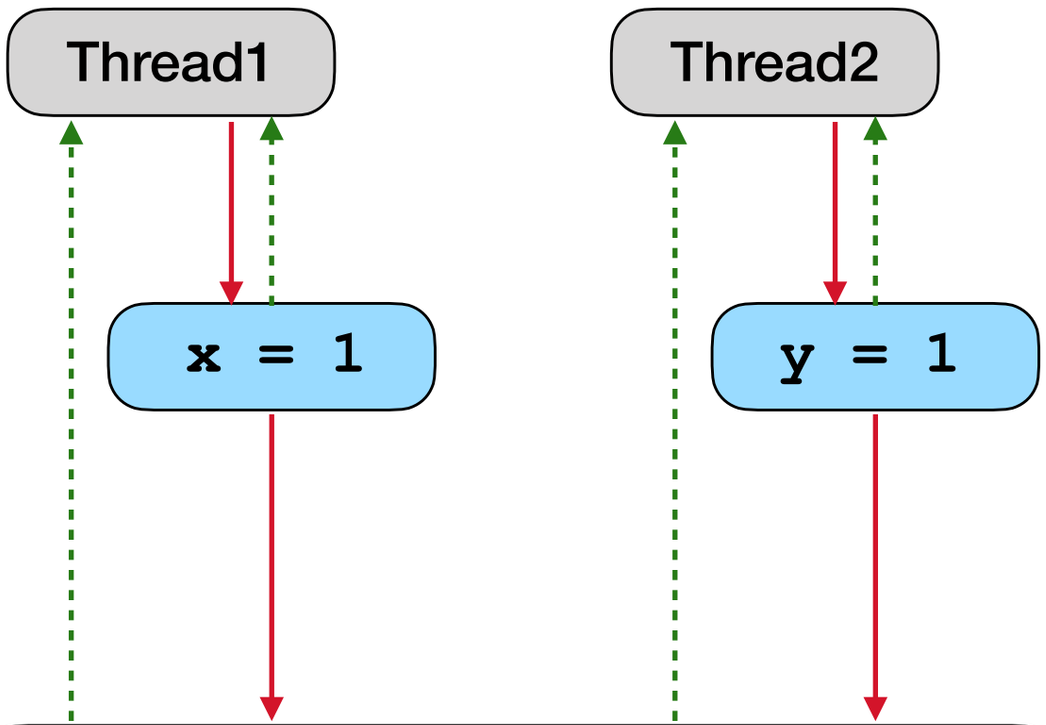
```
x := 1;
a := y; // 0
```

Thread2

```
y := 1;
b := x;
```

Store Buffering (SB)

Hardware (Intel x86) WMC: Store Buffering



x = 0; y = 0;

Thread1

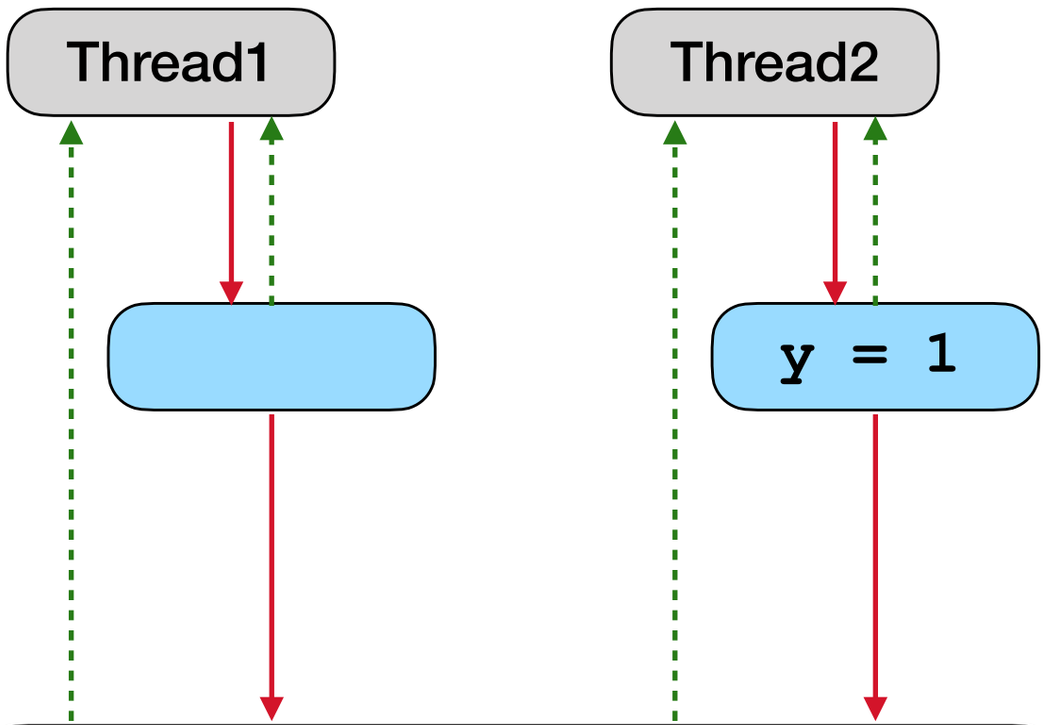
```
x := 1;
a := y; // 0
```

Thread2

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

Store Buffering (SB)

Hardware (Intel x86) WMC: Store Buffering



x = 1; y = 0;

Thread1

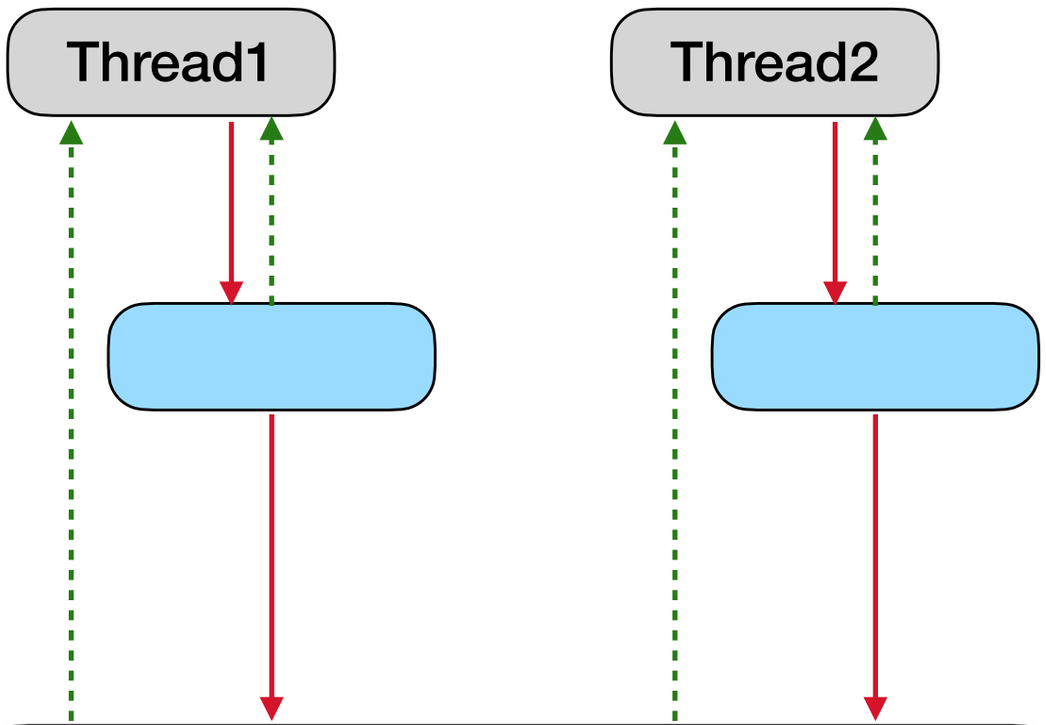
```
x := 1;
a := y; // 0
```

Thread2

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

Store Buffering (SB)

Hardware (Intel x86) WMC: Store Buffering



x = 1; y = 1;

Thread1

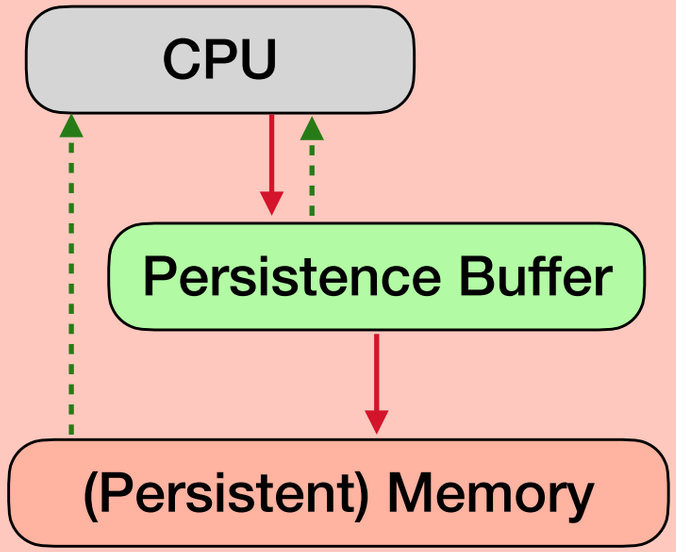
```
x := 1;
a := y; // 0
```

Thread2

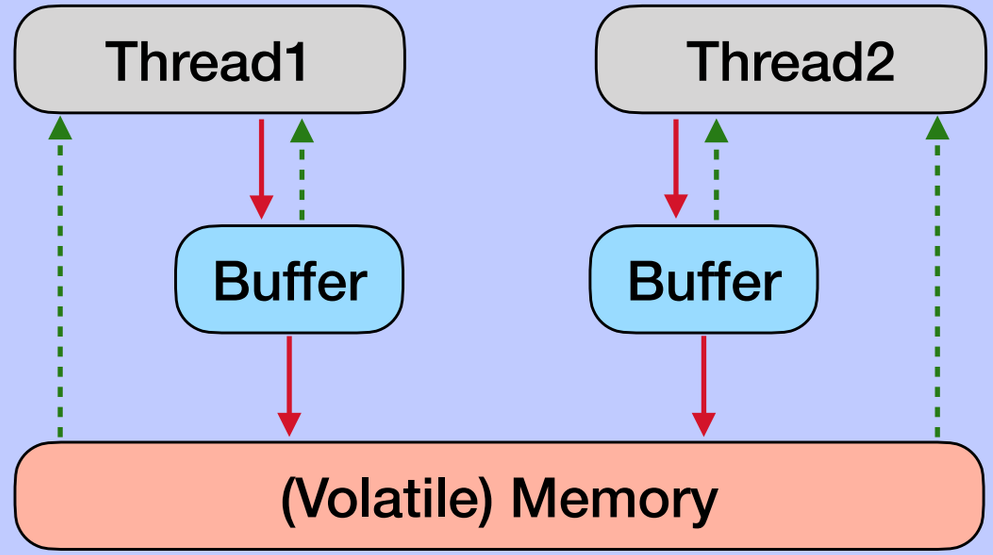
```
y := 1;
b := x; // 0
```

Store Buffering (SB)

Px86: Persistent & Concurrent x86

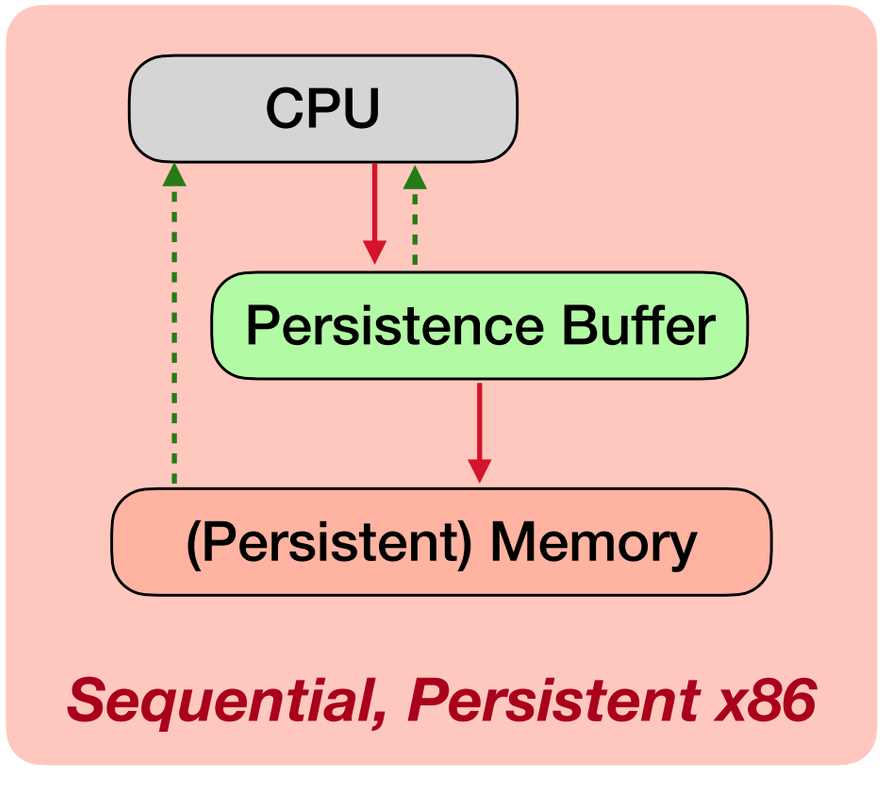


Sequential, Persistent x86

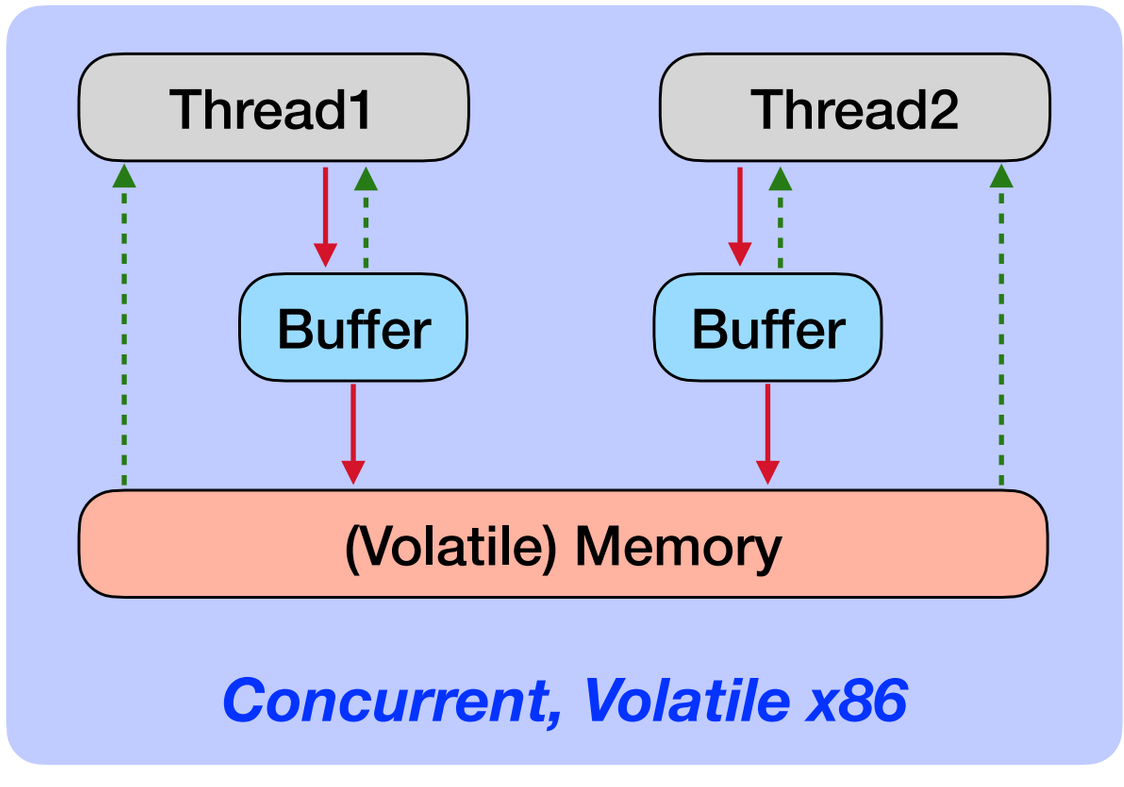


Concurrent, Volatile x86

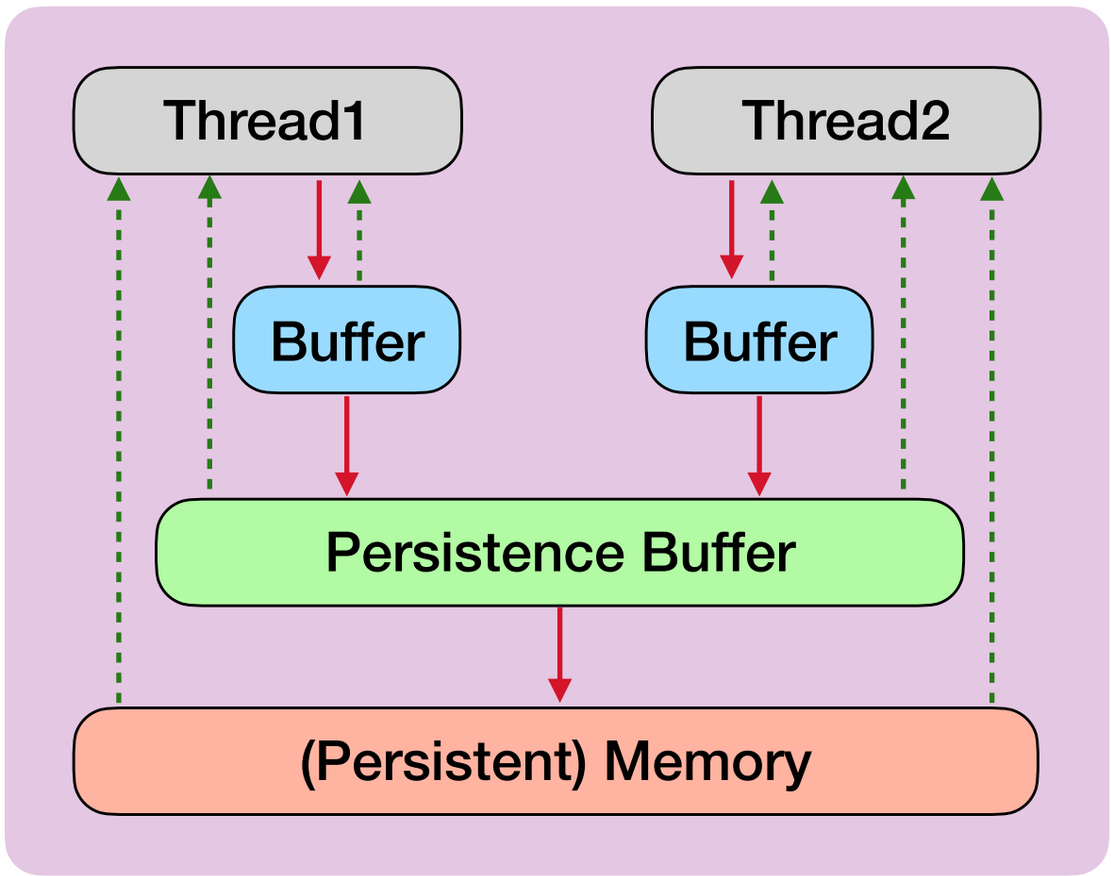
Px86: Persistent & Concurrent x86



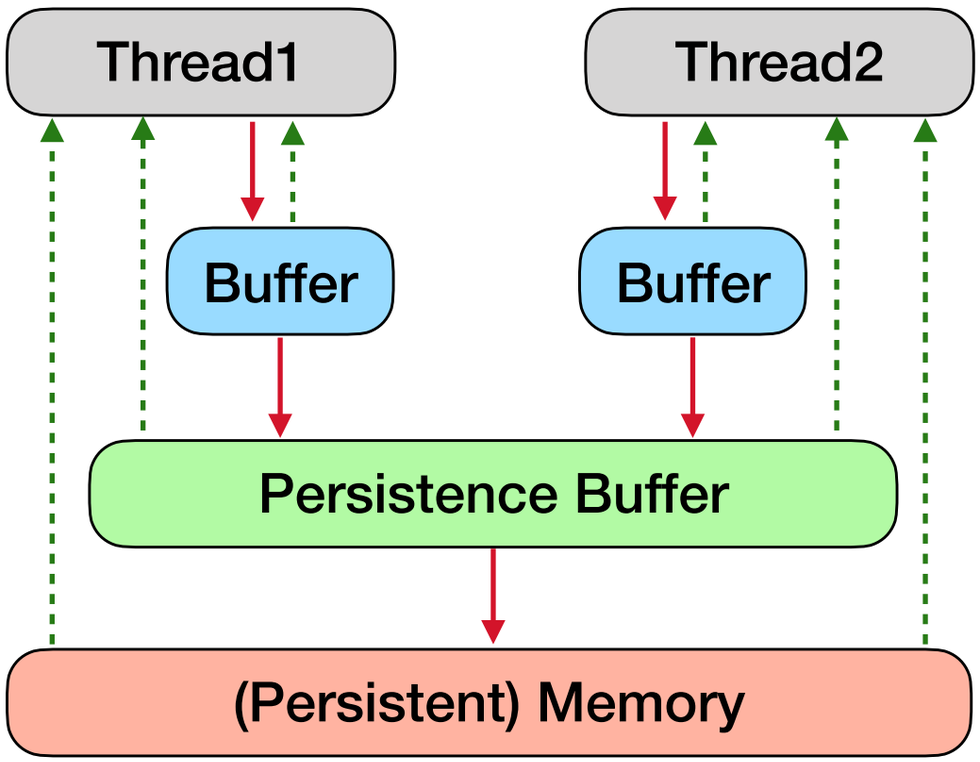
Sequential, Persistent x86



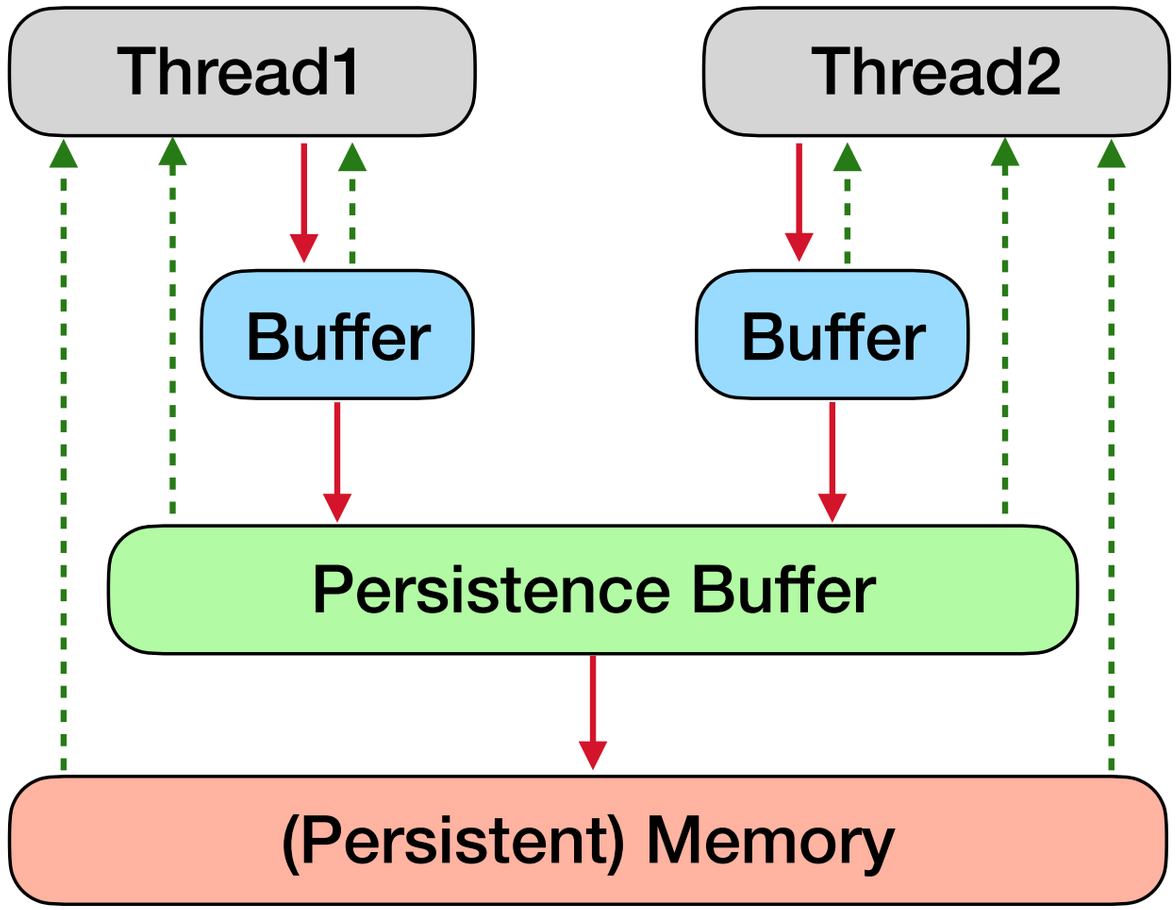
Concurrent, Volatile x86



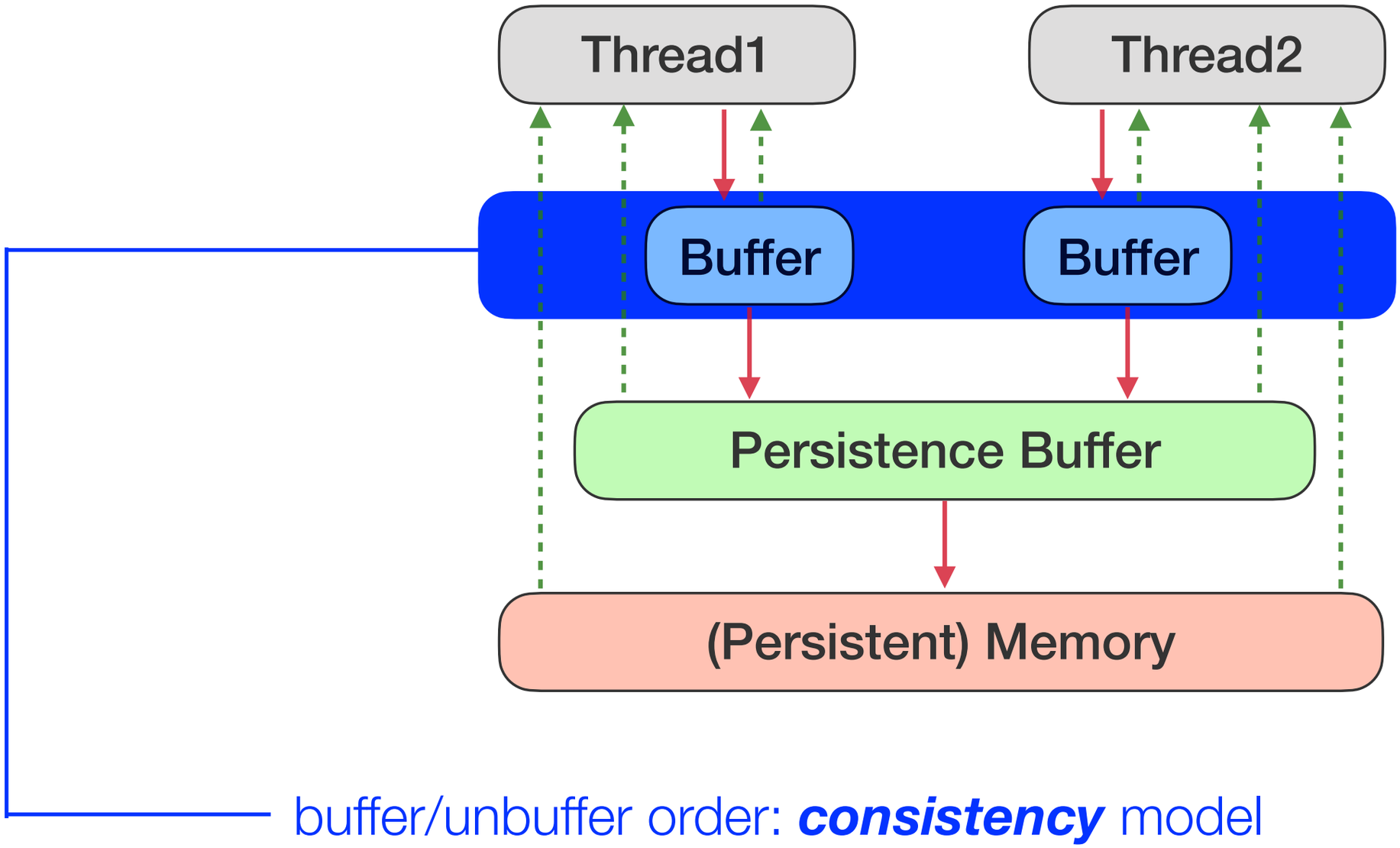
Persistent x86 (Px86)



Persistent x86 (Px86)

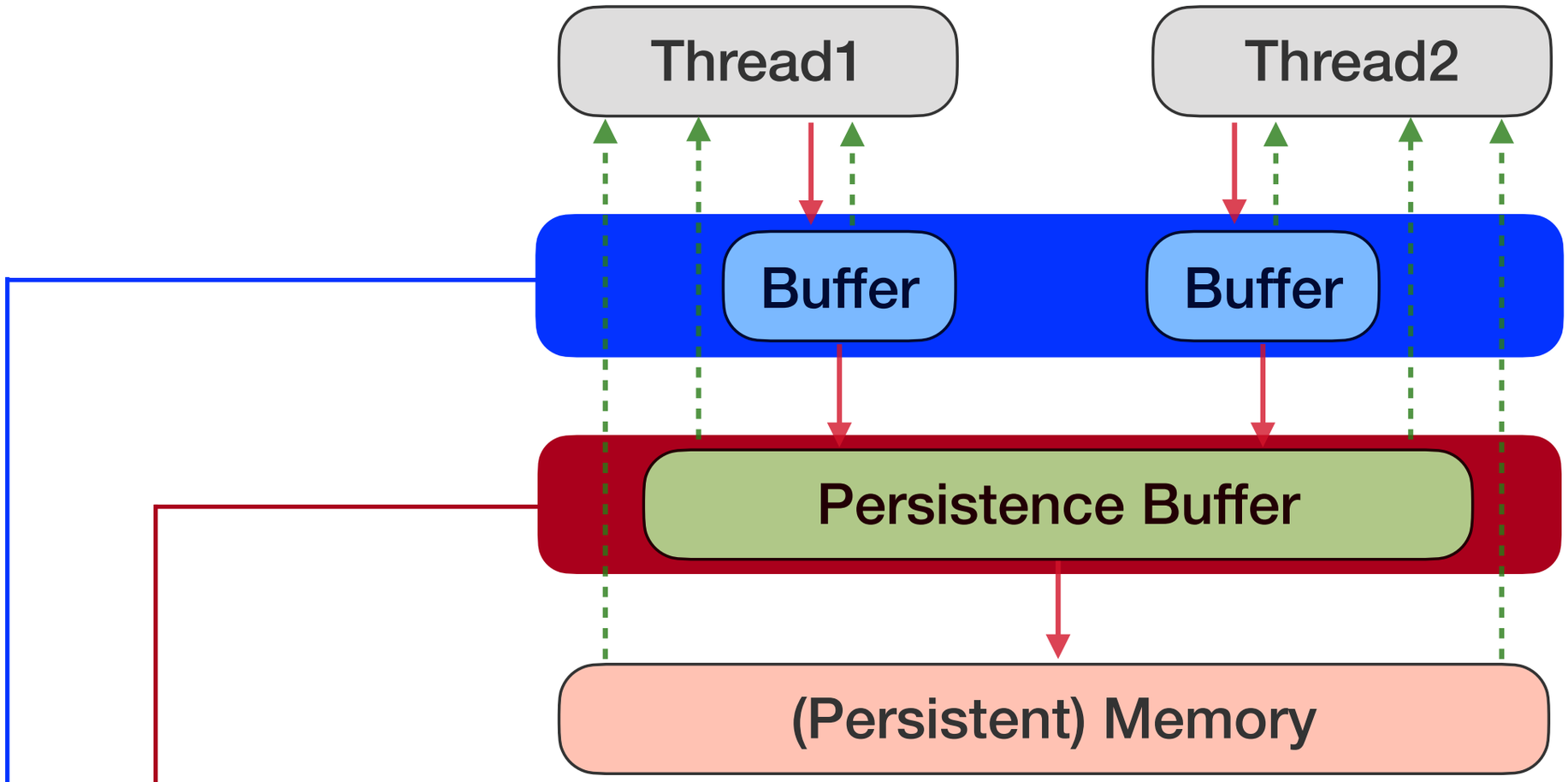


Persistent x86 (Px86)



buffer/unbuffer order: ***consistency*** model

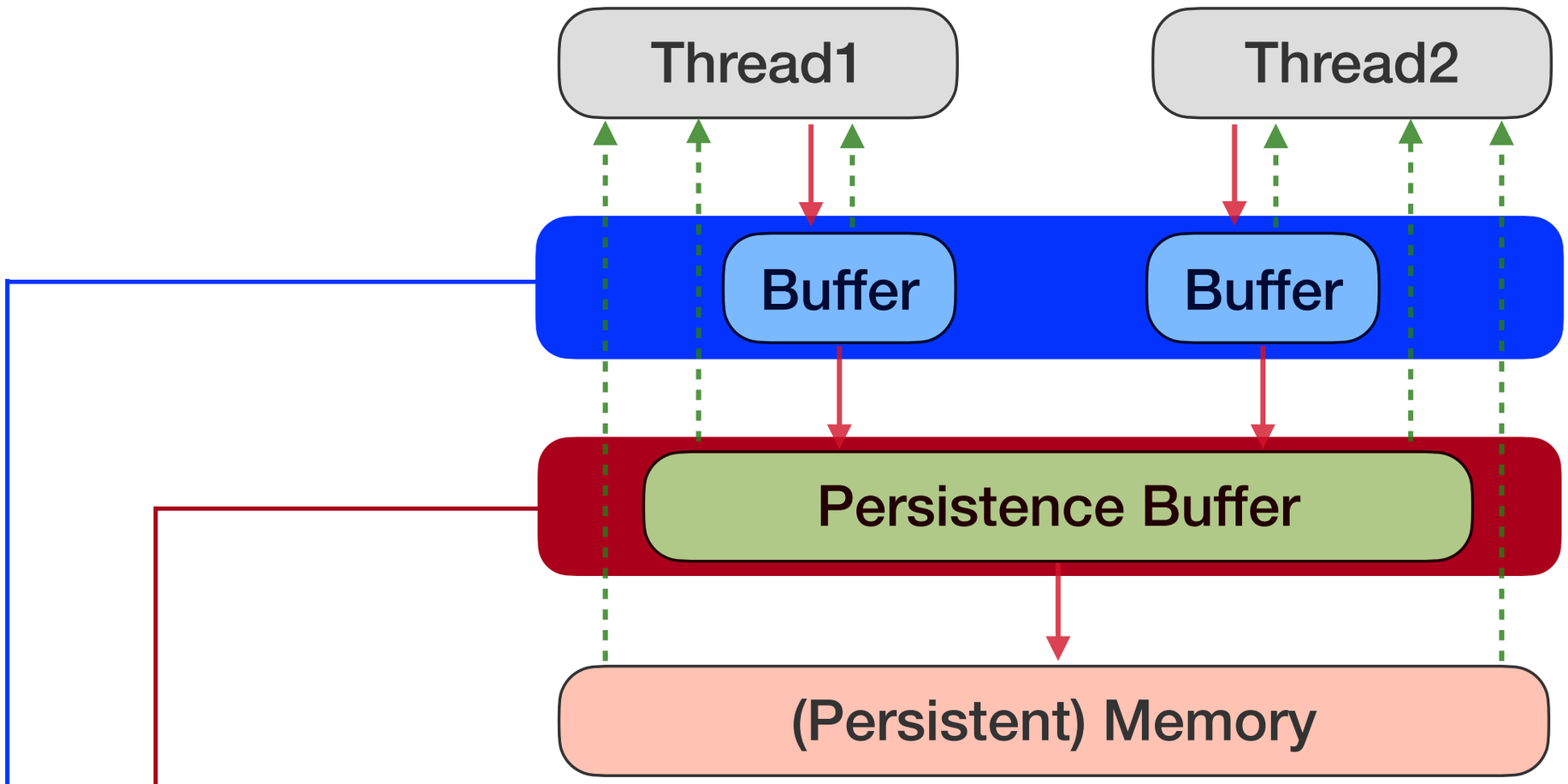
Persistent x86 (Px86)



buffer/unbuffer order: ***consistency*** model

buffer/unbuffer order: ***persistency*** model

Persistent x86 (Px86)



buffer/unbuffer order: ***consistency*** model

buffer/unbuffer order: ***persistency*** model

NVM Semantics (Px86)

2. A *formal* account of Px86

Operational Semantics

Px86 Programming Language

Basic domains

$\mathsf{a} \in \textsc{Reg}$ Registers

$v \in \textsc{Val}$ Values

$\tau \in \textsc{TId}$ Thread IDs

Px86 Programming Language

Basic domains

$\mathsf{a} \in \textsc{Reg}$ Registers

$v \in \textsc{Val}$ Values

$\tau \in \textsc{TId}$ Thread IDs

Expressions and sequential commands

$\textsc{Exp} \ni \mathsf{e} ::= v \mid \mathsf{a} \mid \mathsf{e}{+}\mathsf{e} \mid \cdots$

$\textsc{PCom} \ni \mathsf{c} ::= \mathbf{load}(x) \mid \mathbf{store}(x, \mathsf{e}) \mid \mathbf{CAS}(x, \mathsf{e}, \mathsf{e}') \mid \mathbf{FAA}(x, \mathsf{e})$
$\mid \mathbf{mfence} \mid \mathbf{sfence} \mid \mathbf{flush}_{\mathrm{opt}}\ x \mid \mathbf{flush}\ x$

$\textsc{Com} \ni \mathsf{C} ::= \mathsf{e} \mid \mathsf{c} \mid \mathbf{let}\ \mathsf{a} := \mathsf{C}\ \mathbf{in}\ \mathsf{C}$
$\mid \mathbf{if}\ (\mathsf{C})\ \mathbf{then}\ \mathsf{C}\ \mathbf{else}\ \mathsf{C} \mid \mathbf{repeat}\ \mathsf{C}$

RMW instructions

Px86 Programming Language

Basic domains

$\mathsf{a} \in \textsc{Reg}$ Registers

$v \in \textsc{Val}$ Values

$\tau \in \textsc{TId}$ Thread IDs

Expressions and sequential commands

$\textsc{Exp} \ni \mathsf{e} ::= v \mid \mathsf{a} \mid \mathsf{e}{+}\mathsf{e} \mid \cdots$

$\textsc{PCom} \ni \mathsf{c} ::= \mathbf{load}(x) \mid \mathbf{store}(x, \mathsf{e}) \mid \mathbf{CAS}(x, \mathsf{e}, \mathsf{e}') \mid \mathbf{FAA}(x, \mathsf{e})$
$\mid \mathbf{mfence} \mid \mathbf{sfence} \mid \mathbf{flush}_{\mathrm{opt}}\ x \mid \mathbf{flush}\ x$

$\textsc{Com} \ni \mathsf{C} ::= \mathsf{e} \mid \mathsf{c} \mid \mathbf{let}\ \mathsf{a}{:=}\mathsf{C}\ \mathbf{in}\ \mathsf{C}$
$\mid \mathbf{if}\ (\mathsf{C})\ \mathbf{then}\ \mathsf{C}\ \mathbf{else}\ \mathsf{C} \mid \mathbf{repeat}\ \mathsf{C}$

RMW instructions

used for ***persistency***

Px86 Programming Language

Basic domains

$\mathsf{a} \in \textsc{Reg}$ Registers

$v \in \textsc{Val}$ Values

$\tau \in \textsc{TId}$ Thread IDs

Expressions and sequential commands

$\textsc{Exp} \ni \mathsf{e} ::= v \mid \mathsf{a} \mid \mathsf{e}{+}\mathsf{e} \mid \cdots$

$\textsc{PCom} \ni \mathsf{c} ::= \mathbf{load}(x) \mid \mathbf{store}(x, \mathsf{e}) \mid \mathbf{CAS}(x, \mathsf{e}, \mathsf{e}') \mid \mathbf{FAA}(x, \mathsf{e})$
$\mid \mathbf{mfence} \mid \mathbf{sfence} \mid \mathbf{flush}_{\mathrm{opt}}\ x \mid \mathbf{flush}\ x$

$\textsc{Com} \ni \mathsf{C} ::= \mathsf{e} \mid \mathsf{c} \mid \mathbf{let}\ \mathsf{a}{:=}\mathsf{C}\ \mathbf{in}\ \mathsf{C}$
$\mid \mathbf{if}\ (\mathsf{C})\ \mathbf{then}\ \mathsf{C}\ \mathbf{else}\ \mathsf{C} \mid \mathbf{repeat}\ \mathsf{C}$

RMW instructions

used for ***persistency***

Programs

$\mathsf{P} \in \textsc{Prog} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Com}$

Px86 Operational Semantics

=

Px86 ***Program*** Transitions

+

Px86 ***Storage*** Transitions

➡ First formulated by Raad et al. [2]

➡ Later simplified by Khyzha and Lahav [3]

Px86 Program Transitions

Thread transitions: $\textsc{Com} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Com}$

$\textsc{Lab} \triangleq \{(\mathsf{R}, x, v), (\mathsf{W}, x, v), (\mathsf{U}, x, v, v'), \mathsf{MF}, \mathsf{SF}, (\mathsf{FO}, x), (\mathsf{FL}, x) \mid x \in \textsc{Loc} \wedge v, v' \in \textsc{Val}\}$

Px86 Program Transitions

Thread transitions: $\textsc{Com} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Com}$

$\textsc{Lab} \triangleq \{(\mathsf{R}, x, v), (\mathsf{W}, x, v), (\mathsf{U}, x, v, v'), \mathsf{MF}, \mathsf{SF}, (\mathsf{FO}, x), (\mathsf{FL}, x) \mid x \in \textsc{Loc} \wedge v, v' \in \textsc{Val}\}$

$$\frac{}{\mathbf{load}(x) \xrightarrow{\tau:(\mathsf{R},x,v)} v} \ (\textsc{T-Read})$$

$$\frac{}{\mathbf{store}(x, v) \xrightarrow{\tau:(\mathsf{W},x,v)} v} \ (\textsc{T-Write})$$

Px86 Program Transitions

Thread transitions: $\textsc{Com} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Com}$

$\textsc{Lab} \triangleq \{(\mathsf{R}, x, v), (\mathsf{W}, x, v), (\mathsf{U}, x, v, v'), \mathsf{MF}, \mathsf{SF}, (\mathsf{FO}, x), (\mathsf{FL}, x) \mid x \in \textsc{Loc} \wedge v, v' \in \textsc{Val}\}$

$$\frac{}{\mathbf{load}(x) \xrightarrow{\tau:(\mathsf{R},x,v)} v} \text{(T-Read)}$$

$$\frac{}{\mathbf{store}(x, v) \xrightarrow{\tau:(\mathsf{W},x,v)} v} \text{(T-Write)}$$

$$\frac{}{\mathbf{CAS}(x, v_1, v_2) \xrightarrow{\tau:(\mathsf{U},x,v_1,v_2)} 1} \text{(T-CAS1)}$$

$$\frac{v \neq v_1}{\mathbf{CAS}(x, v_1, v_2) \xrightarrow{\tau:(\mathsf{R},x,v)} 0} \text{(T-CAS0)}$$

Px86 Program Transitions

Thread transitions: $\textsc{Com} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Com}$

$\textsc{Lab} \triangleq \{(\mathsf{R}, x, v), (\mathsf{W}, x, v), (\mathsf{U}, x, v, v'), \mathsf{MF}, \mathsf{SF}, (\mathsf{FO}, x), (\mathsf{FL}, x) \mid x \in \textsc{Loc} \wedge v, v' \in \textsc{Val}\}$

$$\frac{}{\mathbf{load}(x) \xrightarrow{\tau:(\mathsf{R},x,v)} v}\ (\text{T-Read})$$

$$\frac{}{\mathbf{store}(x, v) \xrightarrow{\tau:(\mathsf{W},x,v)} v}\ (\text{T-Write})$$

$$\frac{}{\mathbf{CAS}(x, v_1, v_2) \xrightarrow{\tau:(\mathsf{U},x,v_1,v_2)} 1}\ (\text{T-CAS1})$$

$$\frac{v \neq v_1}{\mathbf{CAS}(x, v_1, v_2) \xrightarrow{\tau:(\mathsf{R},x,v)} 0}\ (\text{T-CAS0})$$

$$\frac{}{\mathbf{FAA}(x, v) \xrightarrow{\tau:(\mathsf{U},x,v_0,v_0+v)} v_0}\ (\text{T-FAA})$$

Px86 Program Transitions

Thread transitions: $\textsc{Com} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Com}$

$\textsc{Lab} \triangleq \{(\mathsf{R}, x, v), (\mathsf{W}, x, v), (\mathsf{U}, x, v, v'), \mathsf{MF}, \mathsf{SF}, (\mathsf{FO}, x), (\mathsf{FL}, x) \mid x \in \textsc{Loc} \wedge v, v' \in \textsc{Val}\}$

$$\frac{}{\mathbf{load}(x) \xrightarrow{\tau:(\mathsf{R},x,v)} v}\ (\text{T-Read})$$

$$\frac{}{\mathbf{store}(x, v) \xrightarrow{\tau:(\mathsf{W},x,v)} v}\ (\text{T-Write})$$

$$\frac{}{\mathbf{CAS}(x, v_1, v_2) \xrightarrow{\tau:(\mathsf{U},x,v_1,v_2)} 1}\ (\text{T-CAS1})$$

$$\frac{v \neq v_1}{\mathbf{CAS}(x, v_1, v_2) \xrightarrow{\tau:(\mathsf{R},x,v)} 0}\ (\text{T-CAS0})$$

$$\frac{}{\mathbf{FAA}(x, v) \xrightarrow{\tau:(\mathsf{U},x,v_0,v_0+v)} v_0}\ (\text{T-FAA})$$

$$\frac{}{\mathbf{sfence} \xrightarrow{\tau:\mathsf{SF}} 1}\ (\text{T-SF})$$

$$\frac{}{\mathbf{mfence} \xrightarrow{\tau:\mathsf{MF}} 1}\ (\text{T-MF})$$

Px86 Program Transitions

Thread transitions: $\textsc{Com} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Com}$

$\textsc{Lab} \triangleq \{(\mathsf{R}, x, v), (\mathsf{W}, x, v), (\mathsf{U}, x, v, v'), \mathsf{MF}, \mathsf{SF}, (\mathsf{FO}, x), (\mathsf{FL}, x) \mid x \in \textsc{Loc} \wedge v, v' \in \textsc{Val}\}$

$$\frac{}{\mathbf{load}(x) \xrightarrow{\tau:(\mathsf{R},x,v)} v}\ (\text{T-}\textsc{Read})$$

$$\frac{}{\mathbf{store}(x, v) \xrightarrow{\tau:(\mathsf{W},x,v)} v}\ (\text{T-}\textsc{Write})$$

$$\frac{}{\mathbf{CAS}(x, v_1, v_2) \xrightarrow{\tau:(\mathsf{U},x,v_1,v_2)} 1}\ (\text{T-CAS1})$$

$$\frac{v \neq v_1}{\mathbf{CAS}(x, v_1, v_2) \xrightarrow{\tau:(\mathsf{R},x,v)} 0}\ (\text{T-CAS0})$$

$$\frac{}{\mathbf{FAA}(x, v) \xrightarrow{\tau:(\mathsf{U},x,v_0,v_0+v)} v_0}\ (\text{T-FAA})$$

$$\frac{}{\mathbf{sfence} \xrightarrow{\tau:\mathsf{SF}} 1}\ (\text{T-SF})$$

$$\frac{}{\mathbf{mfence} \xrightarrow{\tau:\mathsf{MF}} 1}\ (\text{T-MF})$$

$$\frac{}{\mathbf{flush}\ x \xrightarrow{\tau:(\mathsf{FL},x)} 1}\ (\text{T-FL})$$

$$\frac{}{\mathbf{flush}_{\mathrm{opt}}\ x \xrightarrow{\tau:(\mathsf{FO},x)} 1}\ (\text{T-FO})$$

Px86 Program Transitions

Thread transitions: $\textsc{Com} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Com}$

$\textsc{Lab} \triangleq \{(\mathsf{R}, x, v), (\mathsf{W}, x, v), (\mathsf{U}, x, v, v'), \mathsf{MF}, \mathsf{SF}, (\mathsf{FO}, x), (\mathsf{FL}, x) \mid x \in \textsc{Loc} \wedge v, v' \in \textsc{Val}\}$

$$\frac{\mathrm{C}_1 \xrightarrow{\tau:l} \mathrm{C}_1'}{\textbf{let } \mathsf{a}{:=}\mathrm{C}_1 \textbf{ in } \mathrm{C}_2 \xrightarrow{\tau:l} \textbf{let } \mathsf{a}{:=}\mathrm{C}_1' \textbf{ in } \mathrm{C}_2} \ (\textsc{T-Let1})$$

$$\frac{}{\textbf{let } \mathsf{a}{:=}v \textbf{ in } \mathrm{C} \xrightarrow{\tau:\epsilon} \mathrm{C}[v/\mathsf{a}]} \ (\textsc{T-Let2})$$

Px86 Program Transitions

Thread transitions: $\textsc{Com} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Com}$

$\textsc{Lab} \triangleq \{(\mathsf{R}, x, v), (\mathsf{W}, x, v), (\mathsf{U}, x, v, v'), \mathsf{MF}, \mathsf{SF}, (\mathsf{FO}, x), (\mathsf{FL}, x) \mid x \in \textsc{Loc} \wedge v, v' \in \textsc{Val}\}$

$$\frac{\mathrm{C}_1 \xrightarrow{\tau:l} \mathrm{C}_1'}{\textbf{let } \mathsf{a}{:=}\mathrm{C}_1 \textbf{ in } \mathrm{C}_2 \xrightarrow{\tau:l} \textbf{let } \mathsf{a}{:=}\mathrm{C}_1' \textbf{ in } \mathrm{C}_2} \quad (\text{T-Let1})$$

$$\frac{}{\textbf{let } \mathsf{a}{:=}v \textbf{ in } \mathrm{C} \xrightarrow{\tau:\epsilon} \mathrm{C}[v/\mathsf{a}]} \quad (\text{T-Let2})$$

$$\frac{\mathrm{C} \xrightarrow{\tau:l} \mathrm{C}'}{\textbf{if } (\mathrm{C}) \textbf{ then } \mathrm{C}_1 \textbf{ else } \mathrm{C}_2 \xrightarrow{\tau:l} \textbf{if } (\mathrm{C}') \textbf{ then } \mathrm{C}_1 \textbf{ else } \mathrm{C}_2} \quad (\text{T-If1})$$

$$\frac{v{\neq}0 \Rightarrow \mathrm{C}{=}\mathrm{C}_1 \quad v{=}0 \Rightarrow \mathrm{C}{=}\mathrm{C}_2}{\textbf{if } (v) \textbf{ then } \mathrm{C}_1 \textbf{ else } \mathrm{C}_2 \xrightarrow{\tau:\epsilon} \mathrm{C}} \quad (\text{T-If2})$$

Px86 Program Transitions

Thread transitions: $\textsc{Com} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Com}$

$\textsc{Lab} \triangleq \{(\mathsf{R}, x, v), (\mathsf{W}, x, v), (\mathsf{U}, x, v, v'), \mathsf{MF}, \mathsf{SF}, (\mathsf{FO}, x), (\mathsf{FL}, x) \mid x \in \textsc{Loc} \wedge v, v' \in \textsc{Val}\}$

$$\frac{\mathrm{C}_1 \xrightarrow{\tau:l} \mathrm{C}_1'}{\textbf{let } \mathsf{a}{:=}\mathrm{C}_1 \textbf{ in } \mathrm{C}_2 \xrightarrow{\tau:l} \textbf{let } \mathsf{a}{:=}\mathrm{C}_1' \textbf{ in } \mathrm{C}_2} \ (\textsc{T-Let1})$$

$$\frac{}{\textbf{let } \mathsf{a}{:=}v \textbf{ in } \mathrm{C} \xrightarrow{\tau:\epsilon} \mathrm{C}[v/\mathsf{a}]} \ (\textsc{T-Let2})$$

$$\frac{\mathrm{C} \xrightarrow{\tau:l} \mathrm{C}'}{\textbf{if } (\mathrm{C}) \textbf{ then } \mathrm{C}_1 \textbf{ else } \mathrm{C}_2 \xrightarrow{\tau:l} \textbf{if } (\mathrm{C}') \textbf{ then } \mathrm{C}_1 \textbf{ else } \mathrm{C}_2} \ (\textsc{T-If1})$$

$$\frac{v{\neq}0 \Rightarrow \mathrm{C}{=}\mathrm{C}_1 \quad v{=}0 \Rightarrow \mathrm{C}{=}\mathrm{C}_2}{\textbf{if } (v) \textbf{ then } \mathrm{C}_1 \textbf{ else } \mathrm{C}_2 \xrightarrow{\tau:\epsilon} \mathrm{C}} \ (\textsc{T-If2})$$

$$\frac{}{\textbf{repeat } \mathrm{C} \xrightarrow{\tau:\epsilon} \textbf{if } (\mathrm{C}) \textbf{ then } (\textbf{repeat } \mathrm{C}) \textbf{ else } 0} \ (\textsc{T-Repeat})$$

Px86 Program Transitions

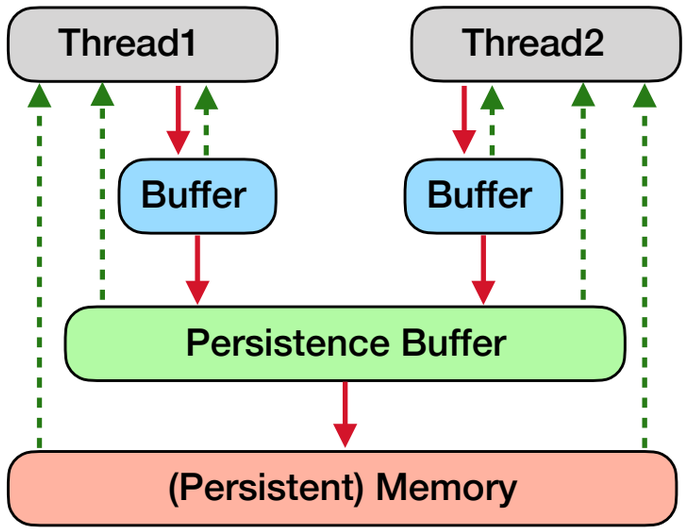
Thread transitions: $\textsc{Com} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Com}$

$\textsc{Lab} \triangleq \{(\mathsf{R}, x, v), (\mathsf{W}, x, v), (\mathsf{U}, x, v, v'), \mathsf{MF}, \mathsf{SF}, (\mathsf{FO}, x), (\mathsf{FL}, x) \mid x \in \textsc{Loc} \wedge v, v' \in \textsc{Val}\}$

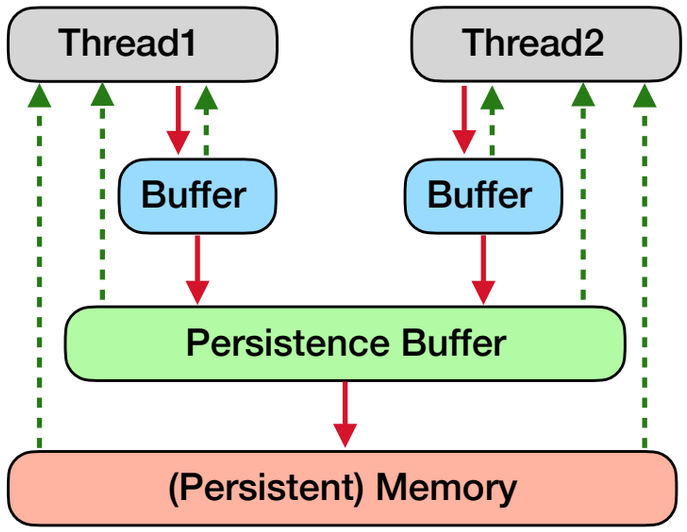
Program transitions: $\textsc{Prog} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Prog}$

$$\frac{\mathrm{P}(\tau) \xrightarrow{\tau:l} \mathrm{C}}{\mathrm{P} \xrightarrow{\tau:l} \mathrm{P}[\tau \mapsto \mathrm{C}]}\ (\textsc{Prog})$$

Px86 Storage System

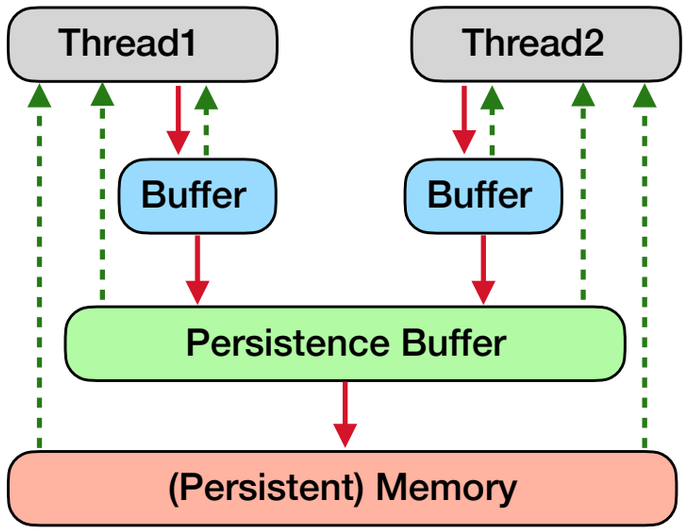


Px86 Storage System



$$M \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val}$$

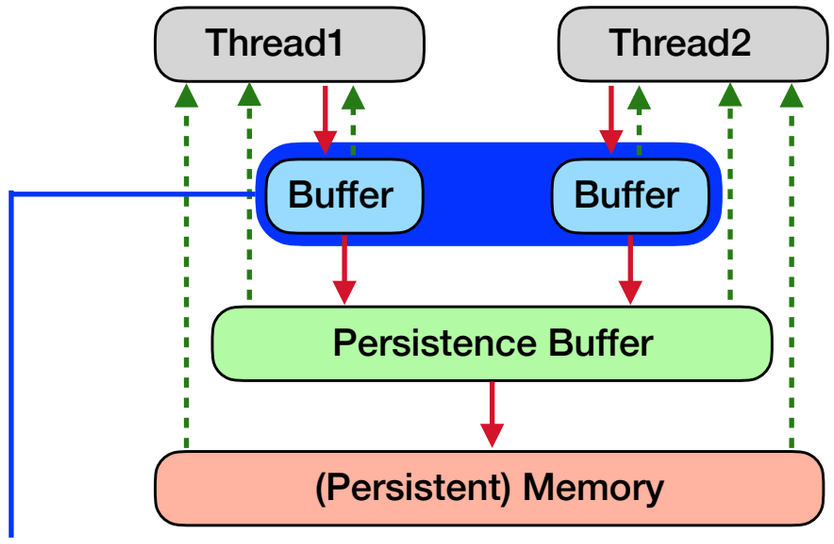
Px86 Storage System



$$M \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val}$$

$$B \in \textsc{BMap} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Buff}$$

Px86 Storage System



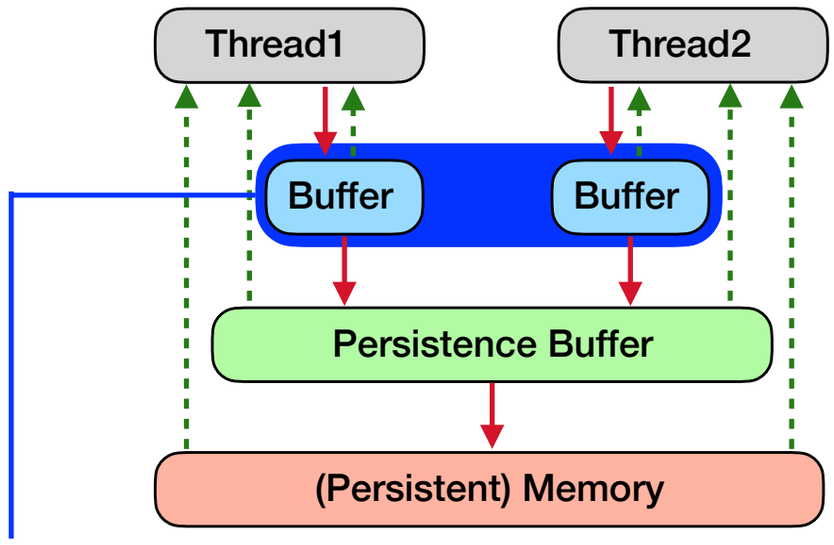
buffer/unbuffer order: consistency model

→ ***execution reordering***

$$M \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val}$$

$$B \in \textsc{BMap} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Buff}$$

Px86 Storage System



buffer/unbuffer order: consistency model

→ ***execution reordering***

Later in Program Order

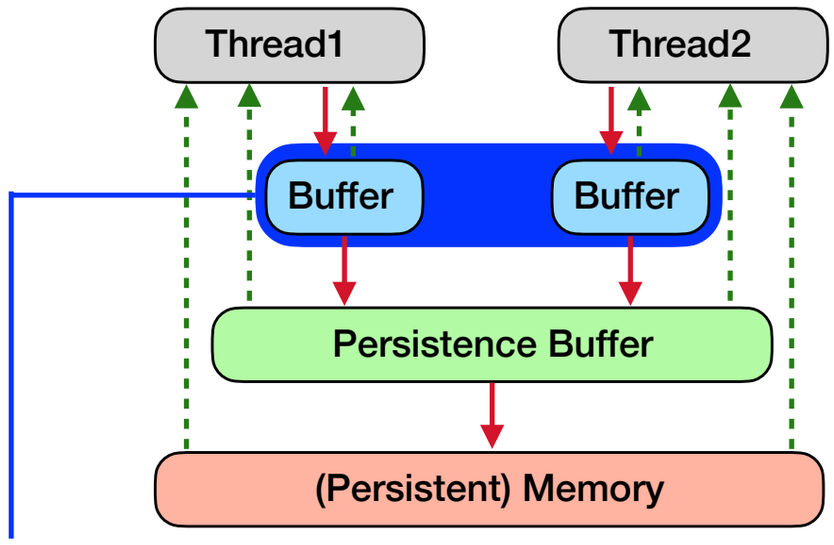
Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

Order preserved? ✓: yes ✗: no sloc : iff on the same location

$$M \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val}$$

$$B \in \textsc{BMap} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Buff}$$

Px86 Storage System



buffer/unbuffer order: consistency model

→ ***execution reordering***

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

Order preserved? ✓: yes ✗: no sloc : iff on the same location

✤ Reads reordered before writes/sfence/flush$_{opt}$/flush

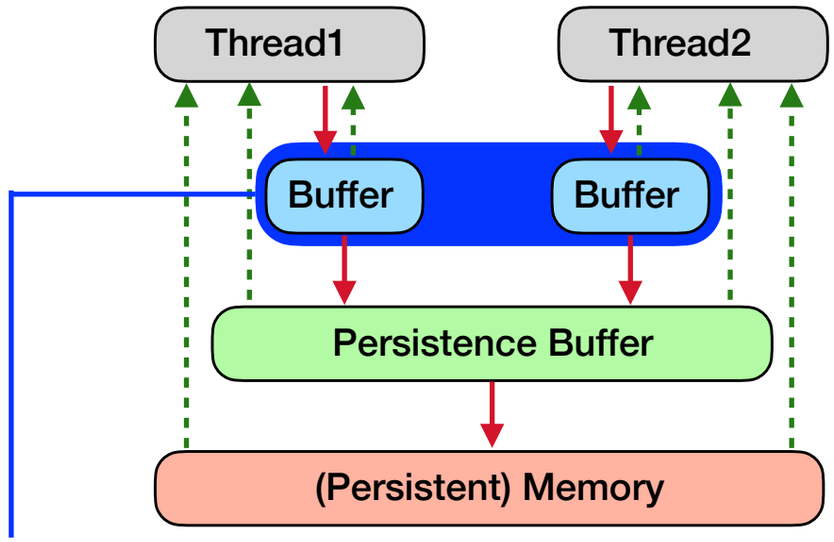
→ ***delay*** writes/sfence/flush$_{opt}$/flush execution in thread buffers

$$M \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val}$$

$$B \in \textsc{BMap} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$b \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathtt{W}, x, v), (\mathtt{FL}, x), (\mathtt{FO}, x), \mathtt{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Px86 Storage System



buffer/unbuffer order: consistency model

→ ***execution reordering***

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

Order preserved? ✓: yes ✗: no sloc : iff on the same location

✤ Reads reordered before writes/sfence/flush$_{opt}$/flush

→ ***delay*** writes/sfence/flush$_{opt}$/flush execution in thread buffers

✤ flush$_{opt}$ reordered w.r.t. writes/flush$_{opt}$/flush on diff. locations

→ their buffer/unbuffer orders (in thread buffers) can disagree

✤ writes/flush/sfence ordered w.r.t. one another

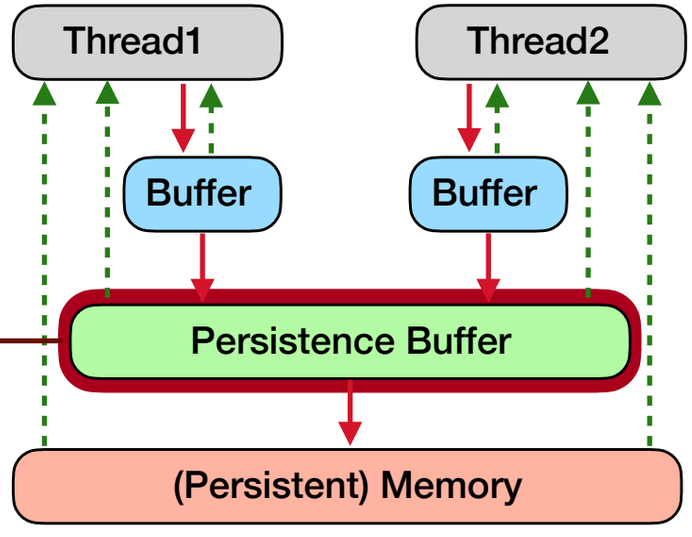
→ their buffer/unbuffer orders agree (FIFO)

$$M \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val}$$

$$B \in \textsc{BMap} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$b \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathtt{W}, x, v), (\mathtt{FL}, x), (\mathtt{FO}, x), \mathtt{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right\rangle$$

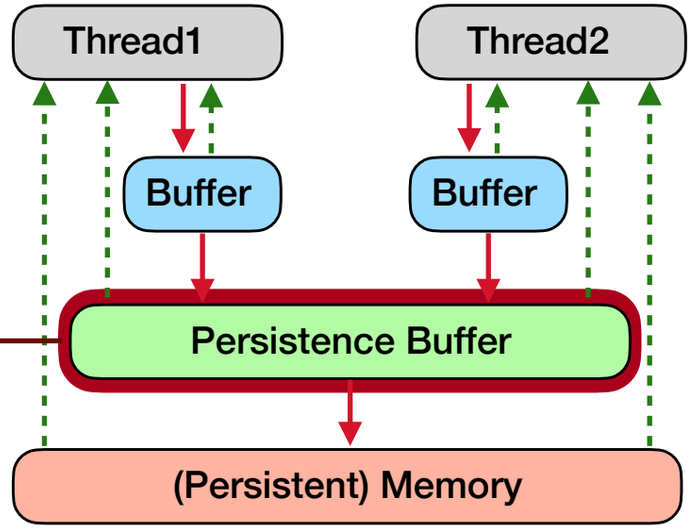
Px86 Storage System



buffer/unbuffer order: persistency model

→ ***persist reordering***

Px86 Storage System



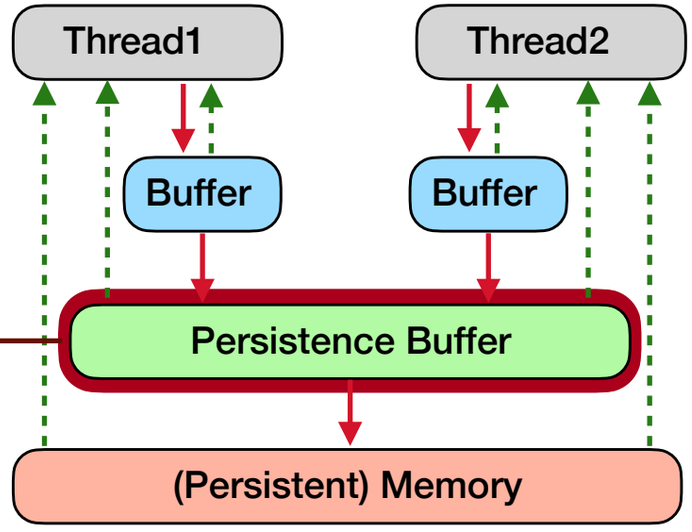
- Persisting writes may be ***delayed***
- Writes on different locations persist in different orders
  → ***per-location persist buffers***
  → record & delay writes in pbuff

buffer/unbuffer order: persistency model
→ ***persist reordering***

$$\mathrm{PB} \in \mathrm{PBMap} \triangleq \mathrm{Loc} \xrightarrow{\mathrm{fin}} \mathrm{PBuff}$$

$$\mathrm{pb} \in \mathrm{PBuff} \triangleq \mathrm{Seq}\left\langle \left\{ \mathrm{w}(v) \;\middle|\; v \in \mathrm{Val} \right\} \right.$$

Px86 Storage System



- Persisting writes may be ***delayed***
- Writes on different locations persist in different orders
  → ***per-location persist buffers***
  → record & delay writes in pbuff

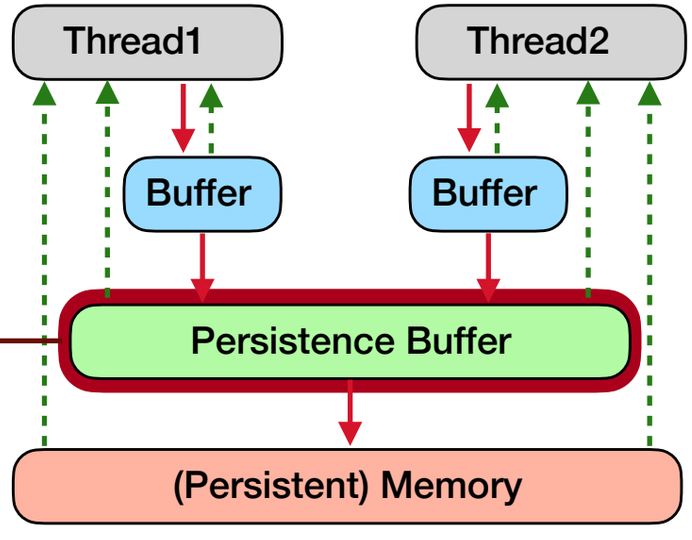
- flush executed ***synchronously***
  → no need to delay/record them in pbuff

buffer/unbuffer order: persistency model
→ ***persist reordering***

$$\mathrm{PB} \in \mathrm{PBMap} \triangleq \mathrm{Loc} \xrightarrow{\mathrm{fin}} \mathrm{PBuff}$$

$$\mathrm{pb} \in \mathrm{PBuff} \triangleq \mathrm{Seq}\left\langle \{\mathrm{w}(v) \mid v \in \mathrm{Val}\} \right.$$

Px86 Storage System



- Persisting writes may be ***delayed***
- Writes on different locations persist in different orders
  → ***per-location persist buffers***
  → record & delay writes in pbuff

- flush executed ***synchronously***
  → no need to delay/record them in pbuff

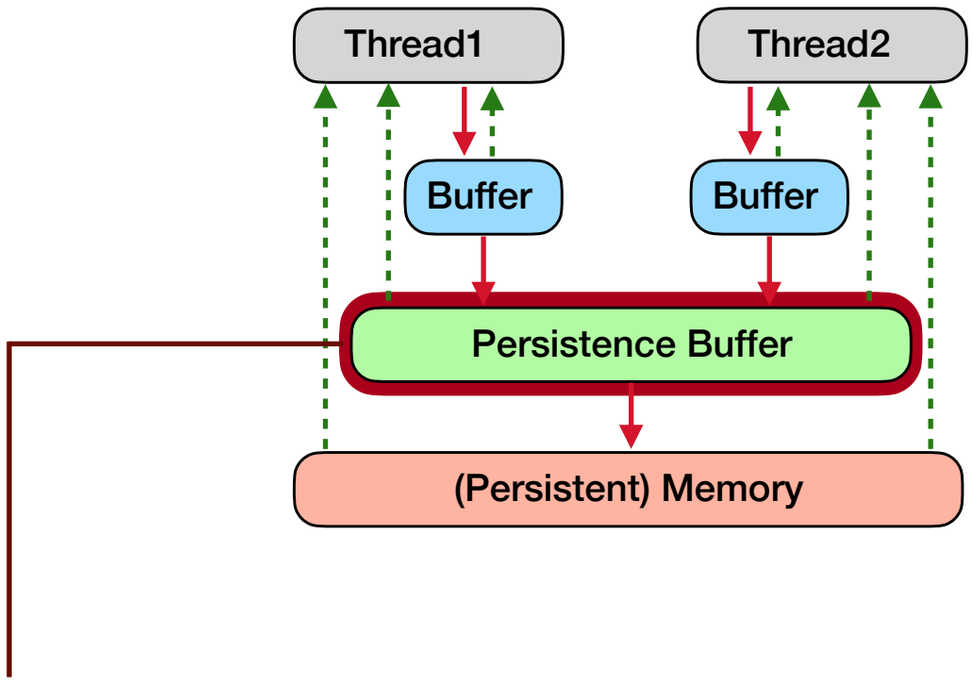
- $\text{flush}_{\text{opt}}$ executed ***asynchronously***
  → record & delay them in pbuff

buffer/unbuffer order: persistency model
→ ***persist reordering***

$$\text{PB} \in \text{PBMap} \triangleq \text{Loc} \xrightarrow{\text{fin}} \text{PBuff}$$

$$\text{pb} \in \text{PBuff} \triangleq \text{Seq}\left\langle \left\{ \text{w}(v), \text{fo}(\tau) \;\middle|\; v \in \text{Val} \wedge \tau \in \text{TId} \right\} \right\rangle$$

Px86 Storage System



buffer/unbuffer order: persistency model
→ ***persist reordering***

- Persisting writes may be ***delayed***
- Writes on different locations persist in different orders
  → ***per-location persist buffers***
  → record & delay writes in pbuff

- flush executed ***synchronously***
  → no need to delay/record them in pbuff

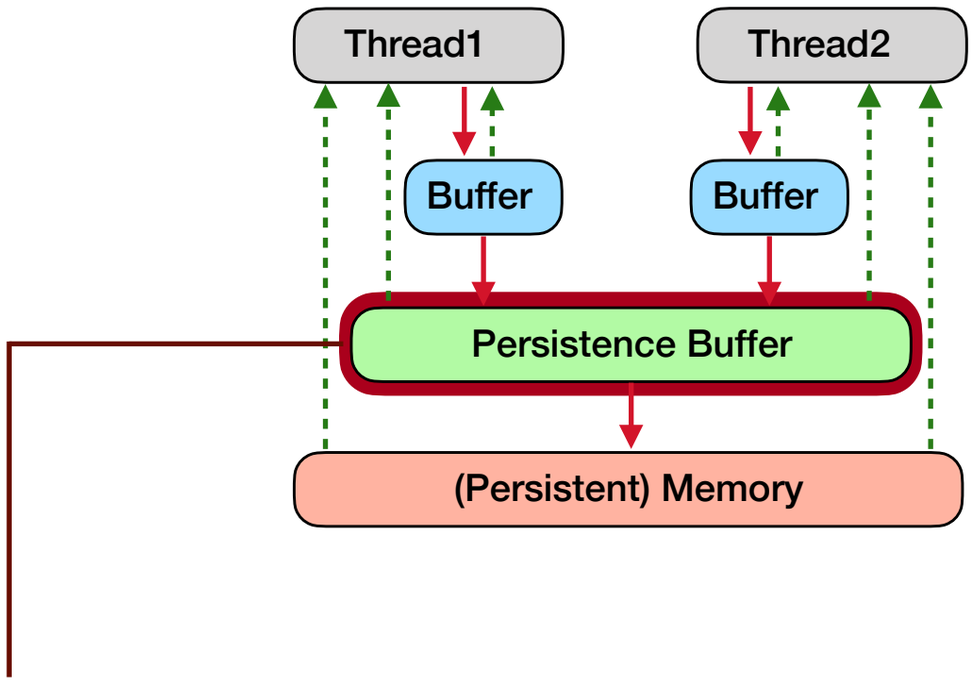
- $\text{flush}_{\text{opt}}$ executed ***asynchronously***
  → record & delay them in pbuff

- $\text{flush}_{\text{opt}}$ + sfence/mfence/RMW = ***persist sequence***
  → ensure no $\text{flush}_{\text{opt}}$ in pbuff

$$\text{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff}$$

$$\text{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \text{w}(v), \text{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{TId} \right\} \right\rangle$$

Px86 Storage System



buffer/unbuffer order: persistency model
→ ***persist reordering***

- Persisting writes may be ***delayed***
- Writes on different locations persist in different orders
  → ***per-location persist buffers***
  → record & delay writes in pbuff

- flush executed ***synchronously***
  → no need to delay/record them in pbuff

- $\text{flush}_{\text{opt}}$ executed ***asynchronously***
  → record & delay them in pbuff

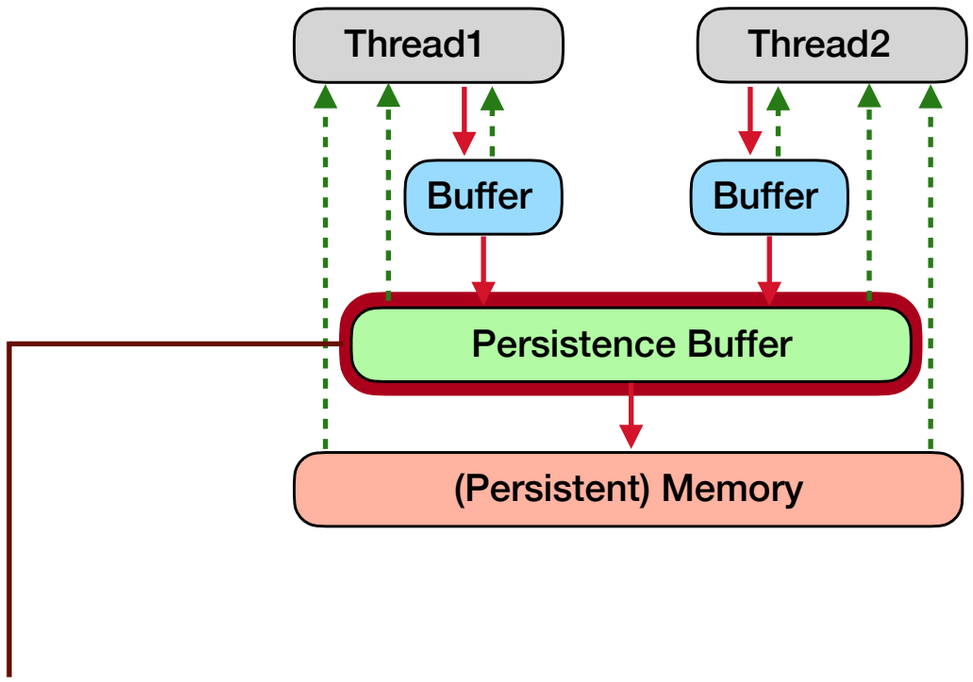
- $\text{flush}_{\text{opt}}$ + sfence/mfence/RMW = ***persist sequence***
  → ensure no $\text{flush}_{\text{opt}}$ in pbuff

$$\text{PB} \in \text{PBMap} \triangleq \text{Loc} \xrightarrow{\text{fin}} \text{PBuff}$$

$$\text{pb} \in \text{PBuff} \triangleq \text{Seq}\left\langle \left\{ \text{w}(v), \text{fo}(\tau) \;\middle|\; v \in \text{Val} \wedge \tau \in \text{TId} \right\} \right\rangle$$

simplification due to Khyzha & Lahav [3]

Px86 Storage System



buffer/unbuffer order: persistency model
→ ***persist reordering***

- Persisting writes may be ***delayed***
- Writes on different locations persist in different orders
  → ***per-location persist buffers***
  → record & delay writes in pbuff

- flush executed ***synchronously***
  → no need to delay/record them in pbuff

- $\text{flush}_{\text{opt}}$ executed ***asynchronously***
  → record & delay them in pbuff

- $\text{flush}_{\text{opt}}$ + sfence/mfence/RMW = ***persist sequence***
  → ensure no $\text{flush}_{\text{opt}}$ in pbuff

$$\text{PB} \in \text{PBMap} \triangleq \text{Loc} \xrightarrow{\text{fin}} \text{PBuff}$$

$$\text{pb} \in \text{PBuff} \triangleq \text{Seq}\left\langle \left\{ \text{w}(v), \text{fo}(\tau) \;\middle|\; v \in \text{Val} \wedge \tau \in \text{TId} \right\} \right\rangle$$

simplification due to Khyzha & Lahav [3]

➡ Original model by Raad et al. [2] : one pbuff for ***all*** locations

Px86 Storage Transitions: Execution

$$\mathsf{M} \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val} \qquad \mathsf{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff} \qquad \mathsf{B} \in \textsc{BMap} \triangleq \textsc{Tid} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$\mathsf{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \mathsf{w}(v), \mathsf{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{Tid} \right\} \right. \qquad \mathsf{b} \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathtt{W}, x, v), (\mathtt{FL}, x), (\mathtt{FO}, x), \mathtt{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{Tid}:\textsc{Lab} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

Px86 Storage Transitions: Execution

$$\mathrm{M} \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val} \qquad \mathrm{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff} \qquad \mathrm{B} \in \textsc{BMap} \triangleq \textsc{Tid} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$\mathrm{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \mathrm{w}(v), \mathrm{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{Tid} \right\} \right. \qquad \mathrm{b} \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathtt{W}, x, v), (\mathtt{FL}, x), (\mathtt{FO}, x), \mathtt{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{Tid}:\textsc{Lab} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathtt{W},x,v)} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.(\mathtt{W}, x, v)]} \text{(M-Write)}$$

Px86 Storage Transitions: Execution

$$\mathrm{M} \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val} \qquad \mathrm{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff} \qquad \mathrm{B} \in \textsc{BMap} \triangleq \textsc{Tid} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$\mathrm{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \mathsf{w}(v), \mathsf{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{Tid} \right\} \right. \quad \mathrm{b} \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathsf{W}, x, v), (\mathsf{FL}, x), (\mathsf{FO}, x), \mathsf{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{Tid}:\textsc{Lab} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathsf{W},x,v)} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.(\mathsf{W}, x, v)]} \text{(M-Write)}$$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathsf{FL},x)} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.(\mathsf{FL}, x)]} \text{(M-FL)}$$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathsf{FO},x)} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.(\mathsf{FO}, x)]} \text{(M-FO)}$$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\mathsf{SF}} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.\mathsf{SF}]} \text{(M-SF)}$$

Px86 Storage Transitions: Execution

$$\mathsf{M} \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val} \qquad \mathsf{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff} \qquad \mathsf{B} \in \textsc{BMap} \triangleq \textsc{Tid} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$\mathsf{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \mathsf{w}(v), \mathsf{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{Tid} \right\} \right. \qquad \mathsf{b} \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathsf{W}, x, v), (\mathsf{FL}, x), (\mathsf{FO}, x), \mathsf{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{Tid}:\textsc{Lab} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{\mathsf{B}(\tau)=\mathsf{b}}{\mathsf{M}, \mathsf{PB}, \mathsf{B} \xrightarrow{\tau:(\mathsf{W},x,v)} \mathsf{M}, \mathsf{PB}, \mathsf{B}[\tau \mapsto \mathsf{b}.(\mathsf{W}, x, v)]} \text{(M-Write)}$$

$$\frac{\mathsf{B}(\tau)=\mathsf{b}}{\mathsf{M}, \mathsf{PB}, \mathsf{B} \xrightarrow{\tau:(\mathsf{FL},x)} \mathsf{M}, \mathsf{PB}, \mathsf{B}[\tau \mapsto \mathsf{b}.(\mathsf{FL}, x)]} \text{(M-FL)}$$

$$\frac{\mathsf{B}(\tau)=\mathsf{b}}{\mathsf{M}, \mathsf{PB}, \mathsf{B} \xrightarrow{\tau:(\mathsf{FO},x)} \mathsf{M}, \mathsf{PB}, \mathsf{B}[\tau \mapsto \mathsf{b}.(\mathsf{FO}, x)]} \text{(M-FO)}$$

$$\frac{\mathsf{B}(\tau)=\mathsf{b}}{\mathsf{M}, \mathsf{PB}, \mathsf{B} \xrightarrow{\tau:\mathsf{SF}} \mathsf{M}, \mathsf{PB}, \mathsf{B}[\tau \mapsto \mathsf{b}.\mathsf{SF}]} \text{(M-SF)}$$

$$\frac{\mathsf{B}(\tau)=\mathsf{b} \quad \mathsf{rd}(\mathsf{M},\mathsf{PB},\mathsf{b},x)=v}{\mathsf{M}, \mathsf{PB}, \mathsf{B} \xrightarrow{\tau:(\mathsf{R},x,v)} \mathsf{M}, \mathsf{PB}, \mathsf{B}} \text{(M-Read)}$$

$$\mathsf{rd}(\mathsf{M}, \mathsf{PB}, \mathsf{b}, x) \triangleq \begin{cases} v & \text{if } \exists S_1, S_2.\ \mathsf{b}{=}S_1.(\mathsf{W}, x, v).S_2 \wedge \forall v'.\ (\mathsf{W}, x, v') \notin S_2 \\ v & \text{else if } \exists S_1, S_2.\ \mathsf{PB}(x){=}S_1.\mathsf{w}(v).S_2 \wedge \forall v'.\ \mathsf{w}(v') \notin S_2 \\ \mathsf{M}(x) & \text{otherwise} \end{cases}$$

Px86 Storage Transitions: Execution

$$\mathrm{M} \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val} \qquad \mathrm{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff} \qquad \mathrm{B} \in \textsc{BMap} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$\mathrm{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \mathtt{w}(v), \mathtt{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{TId} \right\} \right. \quad \left. \mathrm{b} \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathtt{W}, x, v), (\mathtt{FL}, x), (\mathtt{FO}, x), \mathtt{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{TId}:\textsc{Lab} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathtt{W},x,v)} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.(\mathtt{W}, x, v)]} \text{(M-Write)}$$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathtt{FL},x)} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.(\mathtt{FL}, x)]} \text{(M-FL)}$$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathtt{FO},x)} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.(\mathtt{FO}, x)]} \text{(M-FO)}$$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\mathtt{SF}} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.\mathtt{SF}]} \text{(M-SF)}$$

$$\frac{\mathrm{B}(\tau)=\mathrm{b} \quad \mathtt{rd}(\mathrm{M},\mathrm{PB},\mathrm{b},x)=v}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathtt{R},x,v)} \mathrm{M}, \mathrm{PB}, \mathrm{B}} \text{(M-Read)}$$

$$\frac{\mathrm{B}(\tau)=\epsilon \quad \forall x.\ \mathrm{fo}(\tau) \notin \mathrm{PB}(x)}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\mathtt{MF}} \mathrm{M}, \mathrm{PB}, \mathrm{B}} \text{(M-MF)}$$

$$\mathtt{rd}(\mathrm{M}, \mathrm{PB}, \mathrm{b}, x) \triangleq \begin{cases} v & \text{if } \exists S_1, S_2.\ \mathrm{b}=S_1.(\mathtt{W}, x, v).S_2 \wedge \forall v'.\ (\mathtt{W}, x, v') \notin S_2 \\ v & \text{else if } \exists S_1, S_2.\ \mathrm{PB}(x)=S_1.\mathtt{w}(v).S_2 \wedge \forall v'.\ \mathtt{w}(v') \notin S_2 \\ \mathrm{M}(x) & \text{otherwise} \end{cases}$$

flush$_{opt}$ + mfence/RMW = ***persist sequence***
→ ensure no flush$_{opt}$ in pbuff

Px86 Storage Transitions: Execution

$$\mathrm{M} \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val} \qquad \mathrm{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff} \qquad \mathrm{B} \in \textsc{BMap} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$\mathrm{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \mathtt{w}(v), \mathtt{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{TId} \right\} \right. \quad \left. \mathrm{b} \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathtt{W}, x, v), (\mathtt{FL}, x), (\mathtt{FO}, x), \mathtt{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{TId}:\textsc{Lab} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathtt{W},x,v)} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.(\mathtt{W}, x, v)]} \text{(M-Write)}$$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathtt{FL},x)} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.(\mathtt{FL}, x)]} \text{(M-FL)}$$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathtt{FO},x)} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.(\mathtt{FO}, x)]} \text{(M-FO)}$$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\mathtt{SF}} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}.\mathtt{SF}]} \text{(M-SF)}$$

$$\frac{\mathrm{B}(\tau)=\mathrm{b} \quad \mathtt{rd}(\mathrm{M},\mathrm{PB},\mathrm{b},x)=v}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathtt{R},x,v)} \mathrm{M}, \mathrm{PB}, \mathrm{B}} \text{(M-Read)}$$

$$\frac{\mathrm{B}(\tau)=\epsilon \quad \forall x.\ \mathtt{fo}(\tau) \notin \mathrm{PB}(x)}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\mathtt{MF}} \mathrm{M}, \mathrm{PB}, \mathrm{B}} \text{(M-MF)}$$

$$\frac{\mathrm{B}(\tau)=\epsilon \quad \mathtt{rd}(\mathrm{M}, \mathrm{PB}, \epsilon, x)=v_r \quad \mathrm{PB}(x) = \mathrm{pb} \quad \forall x.\ \mathtt{fo}(\tau) \notin \mathrm{PB}(x)}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:(\mathtt{U},x,v_r,v_w)} \mathrm{M}, \mathrm{PB}[x \mapsto \mathrm{pb}.\mathtt{w}(v_w)], \mathrm{B}} \text{(M-RMW)}$$

$$\mathtt{rd}(\mathrm{M}, \mathrm{PB}, \mathrm{b}, x) \triangleq \begin{cases} v & \text{if } \exists S_1, S_2.\ \mathrm{b}=S_1.(\mathtt{W}, x, v).S_2 \wedge \forall v'.\ (\mathtt{W}, x, v') \notin S_2 \\ v & \text{else if } \exists S_1, S_2.\ \mathrm{PB}(x)=S_1.\mathtt{w}(v).S_2 \wedge \forall v'.\ \mathtt{w}(v') \notin S_2 \\ \mathrm{M}(x) & \text{otherwise} \end{cases}$$

flush$_{opt}$ + mfence/RMW = ***persist sequence***
→ ensure no flush$_{opt}$ in pbuff

Px86 Storage Transitions: Delayed Propagation

$$\mathsf{M} \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val} \qquad \mathsf{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff} \qquad \mathsf{B} \in \textsc{BMap} \triangleq \textsc{Tid} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$\mathsf{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \mathsf{w}(v), \mathsf{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{Tid} \right\} \right. \qquad \mathsf{b} \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathtt{W}, x, v), (\mathtt{FL}, x), (\mathtt{FO}, x), \mathtt{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{Tid}:\textsc{Lab} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

Px86 Storage Transitions: Delayed Propagation

$$\mathsf{M} \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val} \qquad \mathsf{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff} \qquad \mathsf{B} \in \textsc{BMap} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$\mathsf{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \mathtt{w}(v), \mathtt{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{TId} \right\} \right. \qquad \mathsf{b} \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathtt{W}, x, v), (\mathtt{FL}, x), (\mathtt{FO}, x), \mathtt{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{TId}:\textsc{Lab} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{\mathsf{B}(\tau) = (\mathtt{W}, x, v).\mathsf{b}' \qquad \mathsf{PB}(x) = \mathsf{pb}}{\mathsf{M}, \mathsf{PB}, \mathsf{B} \xrightarrow{\tau:\epsilon} \mathsf{M}, \mathsf{PB}[x \mapsto \mathsf{pb}.\mathtt{w}(v)], \mathsf{B}[\tau \mapsto \mathsf{b}']} \; (\text{M-PropW})$$

- ✤ writes/flush/sfence ordered w.r.t. one another → their buffer/unbuffer orders agree (FIFO)
- ✤ Persisting writes may be ***delayed*** → record & delay writes in pbuff

Px86 Storage Transitions: Delayed Propagation

$$\mathrm{M} \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val} \qquad \mathrm{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff} \qquad \mathrm{B} \in \textsc{BMap} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$\mathrm{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \mathrm{w}(v), \mathrm{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{TId} \right\} \right. \qquad \mathrm{b} \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathtt{W}, x, v), (\mathtt{FL}, x), (\mathtt{FO}, x), \mathrm{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{TId}:\textsc{Lab} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{\mathrm{B}(\tau) = (\mathtt{W}, x, v).\mathrm{b}' \qquad \mathrm{PB}(x) = \mathrm{pb}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\epsilon} \mathrm{M}, \mathrm{PB}[x \mapsto \mathrm{pb}.\mathrm{w}(v)], \mathrm{B}[\tau \mapsto \mathrm{b}']} \text{(M-PropW)}$$

$$\frac{\mathrm{B}(\tau) = (\mathtt{FL}, x).\mathrm{b}' \qquad \mathrm{PB}(x) = \epsilon}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\epsilon} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}']} \text{(M-PropFL)}$$

- writes/flush/sfence ordered w.r.t. one another → their buffer/unbuffer orders agree (FIFO)
- Persisting writes may be ***delayed*** → record & delay writes in pbuff
- flush executed ***synchronously*** → no need to delay/record them in pbuff

Px86 Storage Transitions: Delayed Propagation

$$\mathrm{M} \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val} \qquad \mathrm{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff} \qquad \mathrm{B} \in \textsc{BMap} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$\mathrm{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \mathtt{w}(v), \mathtt{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{TId} \right\} \right. \quad \left. \mathrm{b} \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathtt{W}, x, v), (\mathtt{FL}, x), (\mathtt{FO}, x), \mathtt{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{TId}:\textsc{Lab} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{\mathrm{B}(\tau) = (\mathtt{W}, x, v).\mathrm{b}' \qquad \mathrm{PB}(x) = \mathrm{pb}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\epsilon} \mathrm{M}, \mathrm{PB}[x \mapsto \mathrm{pb}.\mathtt{w}(v)], \mathrm{B}[\tau \mapsto \mathrm{b}']} \text{(M-PropW)}$$

$$\frac{\mathrm{B}(\tau) = (\mathtt{FL}, x).\mathrm{b}' \qquad \mathrm{PB}(x) = \epsilon}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\epsilon} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}']} \text{(M-PropFL)}$$

$$\frac{\mathrm{B}(\tau) = \mathtt{SF}.\mathrm{b} \quad \forall x.\ \mathtt{fo}(\tau) \notin \mathrm{PB}(x)}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\epsilon} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}]} \text{(M-PropSF)}$$

- writes/flush/sfence ordered w.r.t. one another → their buffer/unbuffer orders agree (FIFO)
- Persisting writes may be ***delayed*** → record & delay writes in pbuff
- flush executed ***synchronously*** → no need to delay/record them in pbuff
- $\text{flush}_{\text{opt}}$ + sfence/mfence/RMW = ***persist sequence*** → ensure no $\text{flush}_{\text{opt}}$ in pbuff

Px86 Storage Transitions: Delayed Propagation

$$\mathrm{M} \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val} \qquad \mathrm{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff} \qquad \mathrm{B} \in \textsc{BMap} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$\mathrm{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \mathtt{w}(v), \mathtt{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{TId} \right\} \right. \quad \left. \mathrm{b} \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathtt{W}, x, v), (\mathtt{FL}, x), (\mathtt{FO}, x), \mathtt{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{TId}:\textsc{Lab} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{\mathrm{B}(\tau)=(\mathtt{W}, x, v).\mathrm{b}' \qquad \mathrm{PB}(x)=\mathrm{pb}}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\epsilon} \mathrm{M}, \mathrm{PB}[x \mapsto \mathrm{pb}.\mathtt{w}(v)], \mathrm{B}[\tau \mapsto \mathrm{b}']} \text{(M-PropW)}$$

$$\frac{\mathrm{B}(\tau)=(\mathtt{FL}, x).\mathrm{b}' \qquad \mathrm{PB}(x)=\epsilon}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\epsilon} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}']} \text{(M-PropFL)}$$

$$\frac{\mathrm{B}(\tau)=\mathtt{SF}.\mathrm{b} \quad \forall x.\ \mathtt{fo}(\tau) \notin \mathrm{PB}(x)}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\epsilon} \mathrm{M}, \mathrm{PB}, \mathrm{B}[\tau \mapsto \mathrm{b}]} \text{(M-PropSF)}$$

$$\frac{\mathrm{B}(\tau)=\mathrm{b}_1.(\mathtt{FO}, x).\mathrm{b}_2 \quad \mathrm{PB}(x)=\mathrm{pb} \quad \mathtt{SF}, (\mathtt{W}, x, -), (\mathtt{FO}, x), (\mathtt{FL}, x) \notin \mathrm{b}_1}{\mathrm{M}, \mathrm{PB}, \mathrm{B} \xrightarrow{\tau:\epsilon} \mathrm{M}, \mathrm{PB}[x \mapsto \mathrm{pb}.\mathtt{fo}(\tau)], \mathrm{B}[\tau \mapsto \mathrm{b}_1.\mathrm{b}_2]} \text{(M-PropFO)}$$

- writes/flush/sfence ordered w.r.t. one another → their buffer/unbuffer orders agree (FIFO)
- Persisting writes may be ***delayed*** → record & delay writes in pbuff
- flush executed ***synchronously*** → no need to delay/record them in pbuff
- $\text{flush}_{\text{opt}}$ + sfence/mfence/RMW = ***persist sequence*** → ensure no $\text{flush}_{\text{opt}}$ in pbuff
- $\text{flush}_{\text{opt}}$ executed ***asynchronously*** → record & delay them in pbuff
- $\text{flush}_{\text{opt}}$ reordered w.r.t. writes/$\text{flush}_{\text{opt}}$/flush on diff. locations → their buffer/unbuffer orders can disagree

Px86 Storage Transitions: Delayed Persists

$$\mathsf{M} \in \textsc{Mem} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{Val} \qquad \mathsf{PB} \in \textsc{PBMap} \triangleq \textsc{Loc} \xrightarrow{\text{fin}} \textsc{PBuff} \qquad \mathsf{B} \in \textsc{BMap} \triangleq \textsc{TId} \xrightarrow{\text{fin}} \textsc{Buff}$$

$$\mathsf{pb} \in \textsc{PBuff} \triangleq \textsc{Seq}\left\langle \left\{ \mathsf{w}(v), \mathsf{fo}(\tau) \;\middle|\; v \in \textsc{Val} \wedge \tau \in \textsc{TId} \right\} \right. \qquad \mathsf{b} \in \textsc{Buff} \triangleq \textsc{Seq}\left\langle \left\{ (\mathtt{W}, x, v), (\mathtt{FL}, x), (\mathtt{FO}, x), \mathtt{SF} \;\middle|\; x \in \textsc{Loc} \wedge v \in \textsc{Val} \right\} \right.$$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{TId}:\textsc{Lab} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{\mathsf{PB}(x) = \mathsf{w}(v).\mathsf{pb}}{\mathsf{M}, \mathsf{PB}, \mathsf{B} \xrightarrow{\tau:\epsilon} \mathsf{M}[x \mapsto v], \mathsf{PB}[x \mapsto \mathsf{pb}], \mathsf{B}} \text{ (M-PersistW)}$$

$$\frac{\mathsf{PB}(x) = \mathsf{fo}(\tau).\mathsf{pb}}{\mathsf{M}, \mathsf{PB}, \mathsf{B} \xrightarrow{\tau:\epsilon} \mathsf{M}, \mathsf{PB}[x \mapsto \mathsf{pb}], \mathsf{B}} \text{ (M-PersistFO)}$$

Px86 Operational Semantics

Program transitions: $\textsc{Prog} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Prog}$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

Px86 Operational Semantics

Program transitions: $\textsc{Prog} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Prog}$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

Operational semantics: $\textsc{Prog} \times \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \Rightarrow \textsc{Prog} \times \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

Px86 Operational Semantics

Program transitions: $\textsc{Prog} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Prog}$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

Operational semantics: $\textsc{Prog} \times \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \Rightarrow \textsc{Prog} \times \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{P \xrightarrow{\tau:\epsilon} P'}{P, M, PB, B \Rightarrow P', M, PB, B}\ (\textsc{SilentP})$$

Px86 Operational Semantics

Program transitions: $\textsc{Prog} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Prog}$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

Operational semantics: $\textsc{Prog} \times \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \Rightarrow \textsc{Prog} \times \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{P \xrightarrow{\tau:\epsilon} P'}{P, M, PB, B \Rightarrow P', M, PB, B}\ (\textsc{SilentP})$$

$$\frac{M, PB, B \xrightarrow{\tau:\epsilon} M', PB', B'}{P, M, PB, B \Rightarrow P, M', PB', B'}\ (\textsc{SilentS})$$

Px86 Operational Semantics

Program transitions: $\textsc{Prog} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Prog}$

Storage transitions: $\textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \xrightarrow{\textsc{Tid}:\textsc{Lab}\cup\{\epsilon\}} \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

Operational semantics: $\textsc{Prog} \times \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap} \Rightarrow \textsc{Prog} \times \textsc{Mem} \times \textsc{PBMap} \times \textsc{BMap}$

$$\frac{P \xrightarrow{\tau:\epsilon} P'}{P, M, PB, B \Rightarrow P', M, PB, B}\ (\textsc{SilentP})$$

$$\frac{M, PB, B \xrightarrow{\tau:\epsilon} M', PB', B'}{P, M, PB, B \Rightarrow P, M', PB', B'}\ (\textsc{SilentS})$$

$$\frac{P \xrightarrow{\tau:l} P' \qquad M, PB, B \xrightarrow{\tau:l} M', PB', B'}{P, M, PB, B \Rightarrow P', M', PB', B'}\ (\textsc{Step})$$

3. A *formal* account of Px86

Declarative Semantics

Declarative Consistency Semantics (w/o Persistency)

- Represent program behaviours as a set of ***consistent executions (graphs)***

Declarative Consistency Semantics (w/o Persistency)

- Represent program behaviours as a set of ***consistent executions (graphs)***
- An execution graph is a tuple: < *E*, po, rf, mo >
- *E* is the set of *events* (graph nodes), including initialisation writes

Declarative Consistency Semantics (w/o Persistency)

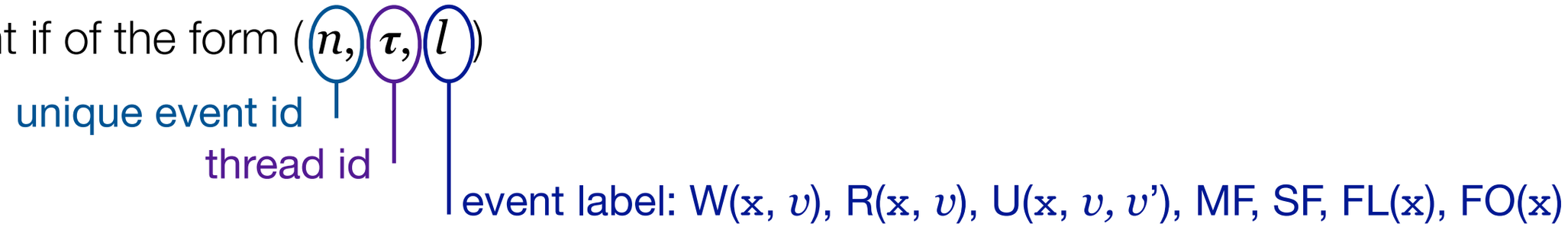
- Represent program behaviours as a set of ***consistent executions (graphs)***
- An execution graph is a tuple: < *E*, po, rf, mo >
- *E* is the set of *events* (graph nodes), including initialisation writes
  - each event if of the form (n, τ, l)

unique event id

thread id

Declarative Consistency Semantics (w/o Persistency)

- Represent program behaviours as a set of ***consistent executions (graphs)***
- An execution graph is a tuple: < *E*, po, rf, mo >
- *E* is the set of *events* (graph nodes), including initialisation writes
  - each event if of the form (n, τ, l)



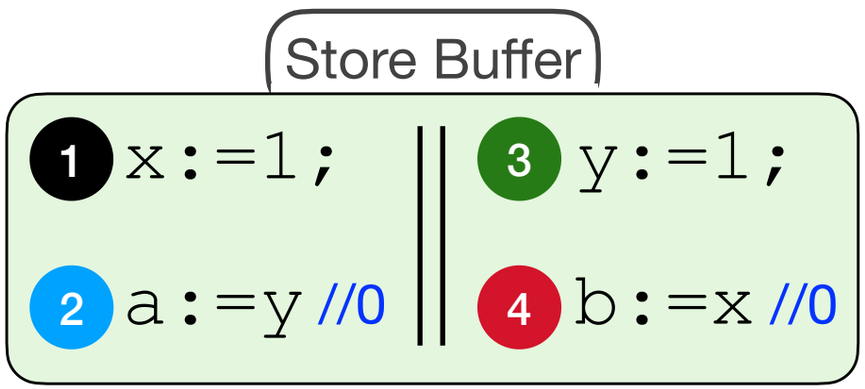
Declarative Consistency Semantics (w/o Persistency)

- Represent program behaviours as a set of ***consistent executions (graphs)***
- An execution graph is a tuple: < *E*, po, rf, mo >
- *E* is the set of *events* (graph nodes), including initialisation writes
  - each event if of the form (n, τ, l)

unique event id

thread id

event label: W(x, v), R(x, v), U(x, v, v'), MF, SF, FL(x), FO(x)



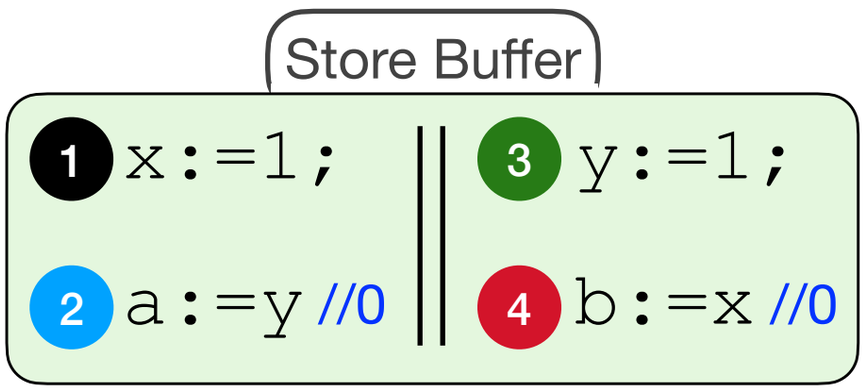
Declarative Consistency Semantics (w/o Persistency)

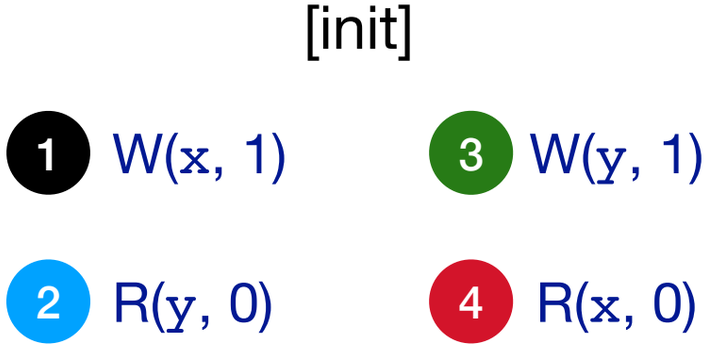
- Represent program behaviours as a set of ***consistent executions (graphs)***
- An execution graph is a tuple: < *E*, po, rf, mo >
- *E* is the set of *events* (graph nodes), including initialisation writes
  - each event if of the form (n, τ, l)

unique event id

thread id

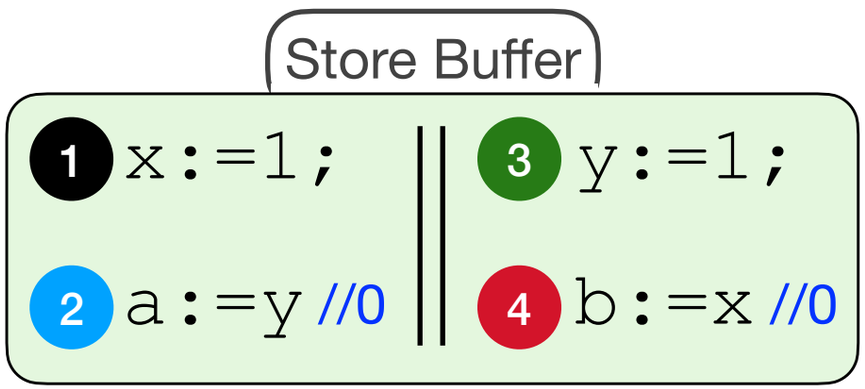
event label: W(x, v), R(x, v), U(x, v, v'), MF, SF, FL(x), FO(x)

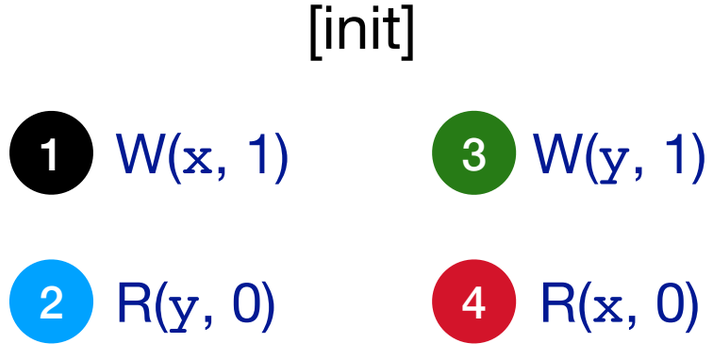




Declarative Consistency Semantics (w/o Persistency)

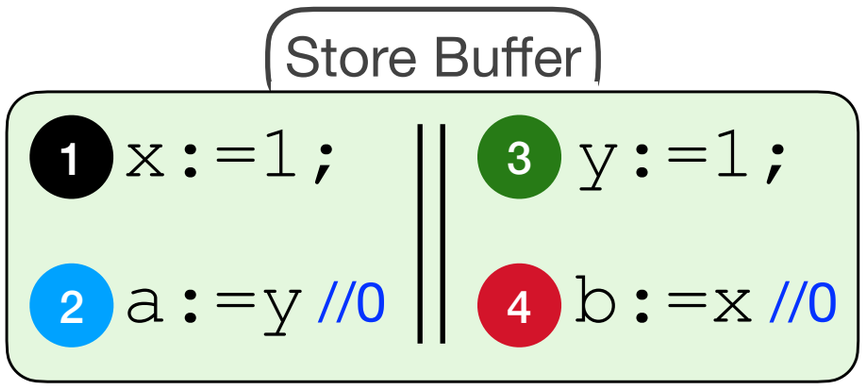
- Represent program behaviours as a set of ***consistent executions (graphs)***
- An execution graph is a tuple: < *E*, po, rf, mo >
- *E* is the set of *events* (graph nodes), including initialisation writes
- po, rf and mo are *relations* on events (graph edges)
  - po is the *program order*: strict total order on events of the same thread

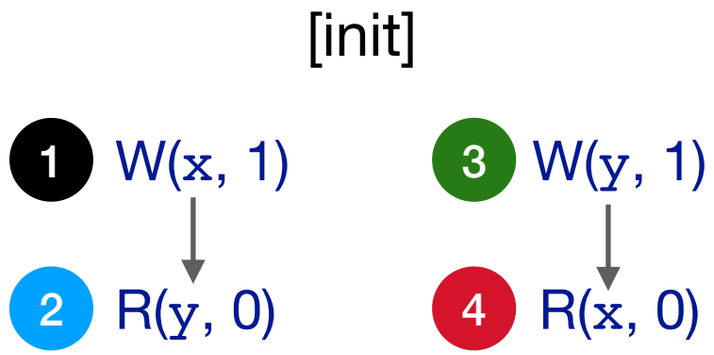




Declarative Consistency Semantics (w/o Persistency)

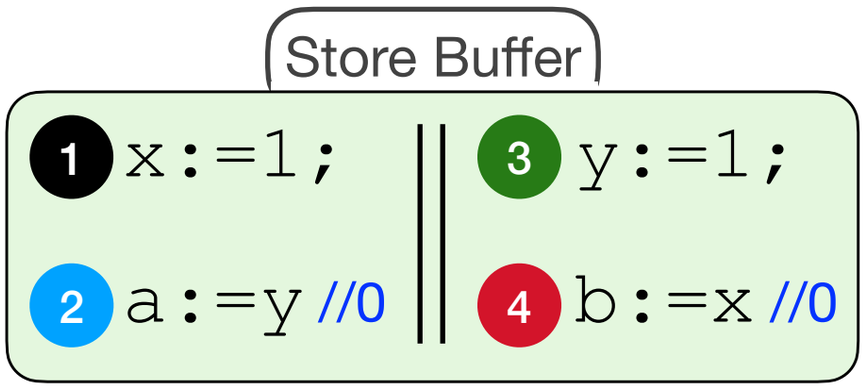
- Represent program behaviours as a set of ***consistent executions (graphs)***
- An execution graph is a tuple: < *E*, po, rf, mo >
- *E* is the set of *events* (graph nodes), including initialisation writes
- po, rf and mo are *relations* on events (graph edges)
  - po is the *program order*: strict total order on events of the same thread

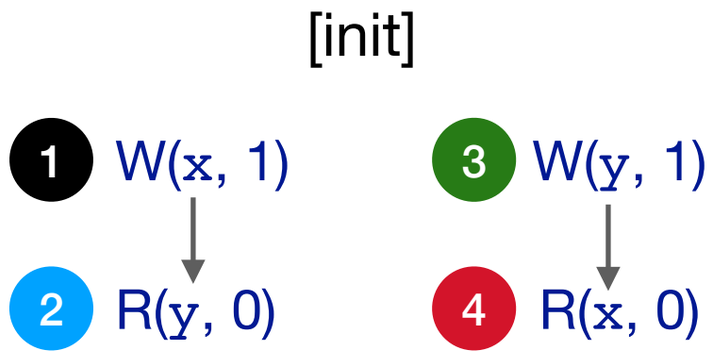




Declarative Consistency Semantics (w/o Persistency)

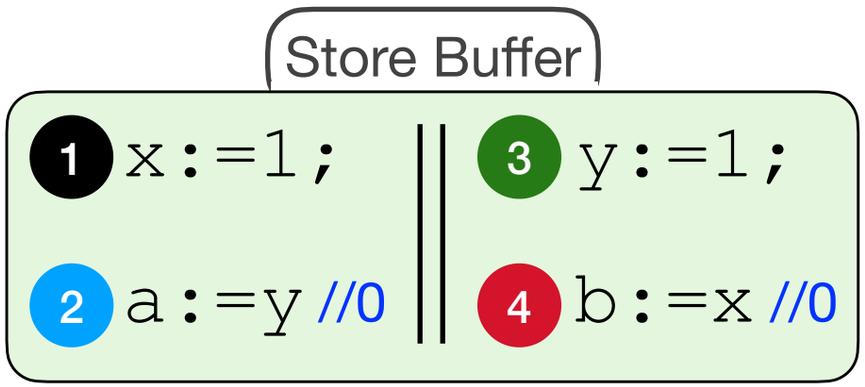
- Represent program behaviours as a set of ***consistent executions (graphs)***
- An execution graph is a tuple: < *E*, po, rf, mo >
- *E* is the set of *events* (graph nodes), including initialisation writes
- po, rf and mo are *relations* on events (graph edges)
  - po is the *program order*: strict total order on events of the same thread
  - rf is the *reads-from* relation: relating each read/update to exactly one write/update on the same location with the same value

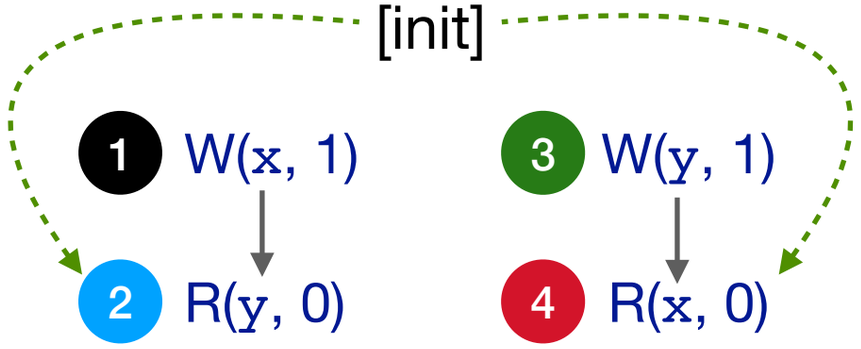




Declarative Consistency Semantics (w/o Persistency)

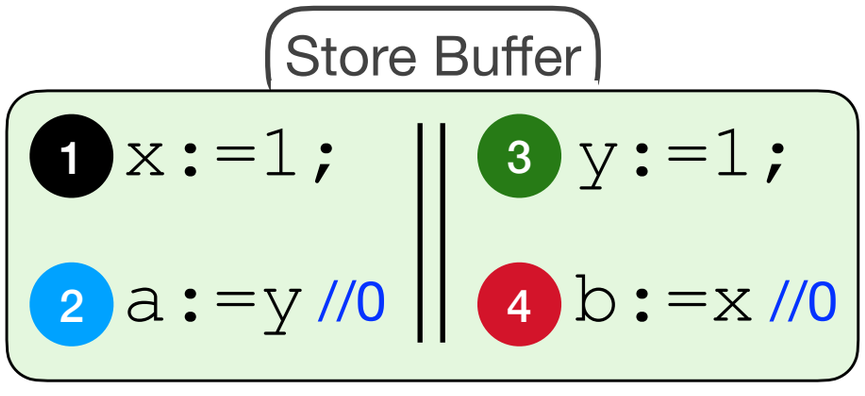
- Represent program behaviours as a set of ***consistent executions (graphs)***
- An execution graph is a tuple: < *E*, po, rf, mo >
- *E* is the set of *events* (graph nodes), including initialisation writes
- po, rf and mo are *relations* on events (graph edges)
  - po is the *program order*: strict total order on events of the same thread
  - rf is the *reads-from* relation: relating each read/update to exactly one write/update on the same location with the same value

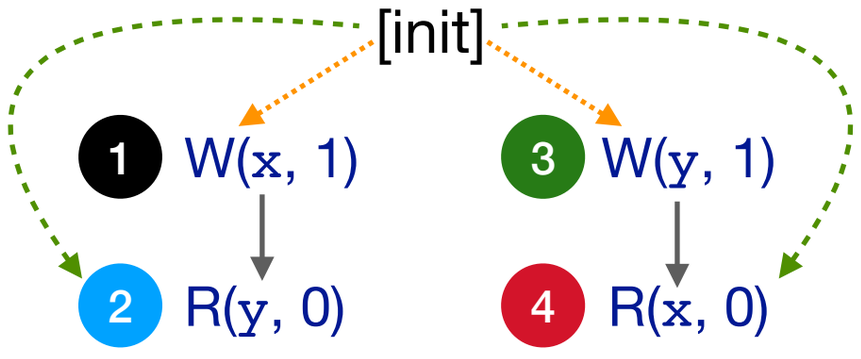




Declarative Consistency Semantics (w/o Persistency)

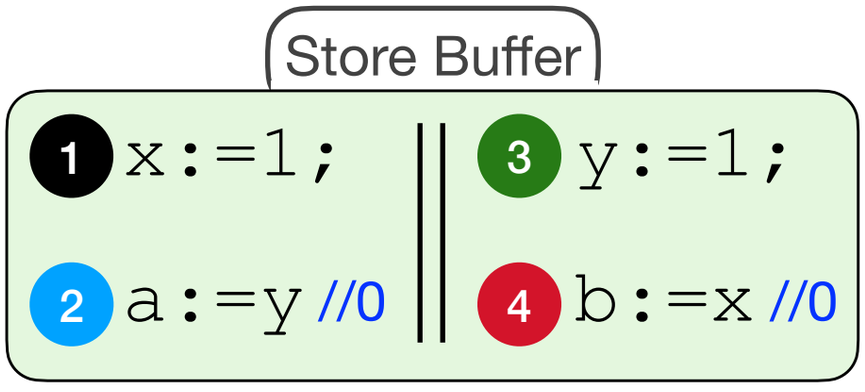
- Represent program behaviours as a set of ***consistent executions (graphs)***
- An execution graph is a tuple: < *E*, po, rf, mo >
- *E* is the set of *events* (graph nodes), including initialisation writes
- po, rf and mo are *relations* on events (graph edges)
  - po is the *program order*: strict total order on events of the same thread
  - rf is the *reads-from* relation: relating each read/update to exactly one write/update on the same location with the same value
  - mo is the *modification order*: strict total order on the writes/updates of the same location

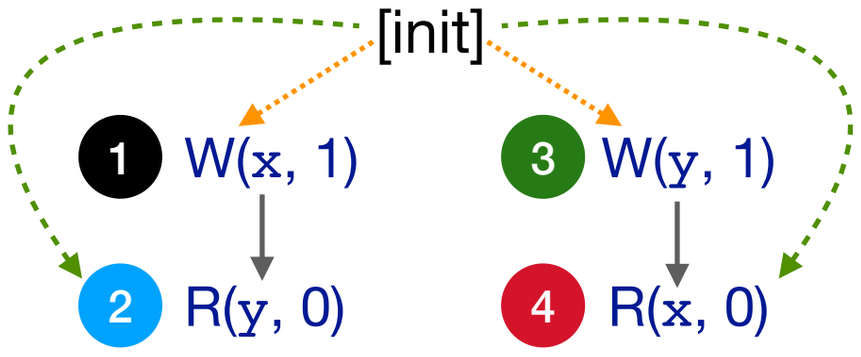




Declarative Consistency Semantics (w/o Persistency)

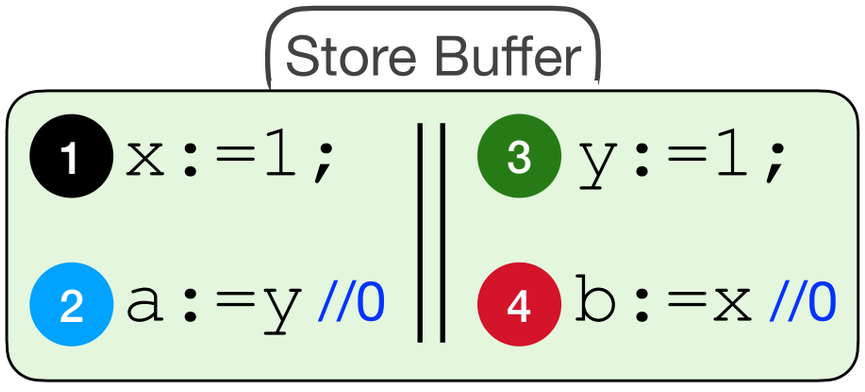
- Represent program behaviours as a set of ***consistent executions (graphs)***
- An execution graph is a tuple: < *E*, po, rf, mo >
- *E* is the set of *events* (graph nodes), including initialisation writes
- po, rf and mo are *relations* on events (graph edges)
  - po is the *program order*: strict total order on events of the same thread
  - rf is the *reads-from* relation: relating each read/update to exactly one write/update on the same location with the same value
  - mo is the *modification order*: strict total order on the writes/updates of the same location
  - Derived relation, rb = $\xrightarrow{rf^{-1}}\xrightarrow{mo}$, is the *reads-before* relation

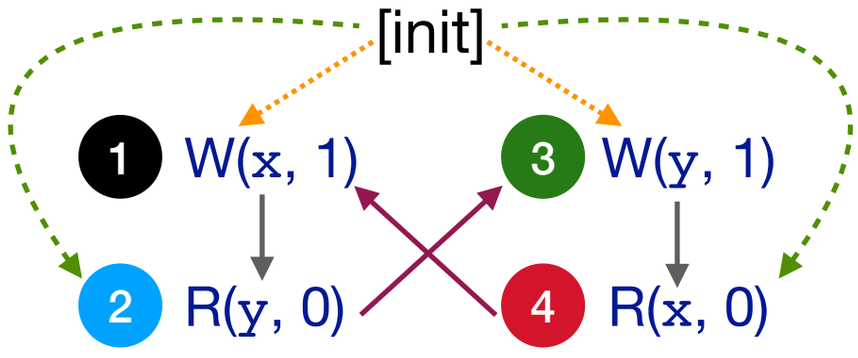




Declarative Consistency Semantics (w/o Persistency)

- Represent program behaviours as a set of ***consistent executions (graphs)***
- An execution graph is a tuple: < *E*, po, rf, mo >
- *E* is the set of *events* (graph nodes), including initialisation writes
- po, rf and mo are *relations* on events (graph edges)
  - po is the *program order*: strict total order on events of the same thread
  - rf is the *reads-from* relation: relating each read/update to exactly one write/update on the same location with the same value
  - mo is the *modification order*: strict total order on the writes/updates of the same location
  - Derived relation, rb = $\xrightarrow{rf^{-1}}\xrightarrow{mo}$, is the *reads-before* relation





Consistent Executions (w/o Persistency)

- What is a ***consistent*** execution?

Consistent Executions (w/o Persistency)

- What is a ***consistent*** execution?
  - Depends on the (concurrency) memory model

Consistent Executions (w/o Persistency)

- What is a ***consistent*** execution?
  - Depends on the (concurrency) memory model
  - Intel-x86 consistency:

$$rfi \cup moi \cup rbi \subseteq po \qquad \text{(Internal)}$$

Ri : internal (same-thread) subset of R
Re : external (diff.-thread) subset of R: R \ Ri

Consistent Executions (w/o Persistency)

- What is a ***consistent*** execution?
  - Depends on the (concurrency) memory model
  - Intel-x86 consistency:

$rfi \cup moi \cup rbi \subseteq po$ (Internal)

$irreflexive(ob) \qquad ob = (ppo \cup rfe \cup moe \cup rbe)^{+}$ (External)

Ri : internal (same-thread) subset of R
Re : external (diff.-thread) subset of R: R \ Ri

Consistent Executions (w/o Persistency)

- What is a ***consistent*** execution?
  - Depends on the (concurrency) memory model
  - Intel-x86 consistency:

$$rfi \cup moi \cup rbi \subseteq po \quad \text{(Internal)}$$

$$\text{irreflexive}(ob) \qquad ob = ((ppo \cup rfe \cup moe \cup rbe)^{+} \quad \text{(External)}$$

preserved program order: sloc or ✓ in table

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush$_{opt}$**	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush$_{opt}$**	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

Ri : internal (same-thread) subset of R

Re : external (diff.-thread) subset of R: R \ Ri

Consistent Executions (w/o Persistency)

- What is a ***consistent*** execution?
  - Depends on the (concurrency) memory model
  - Intel-x86 consistency:

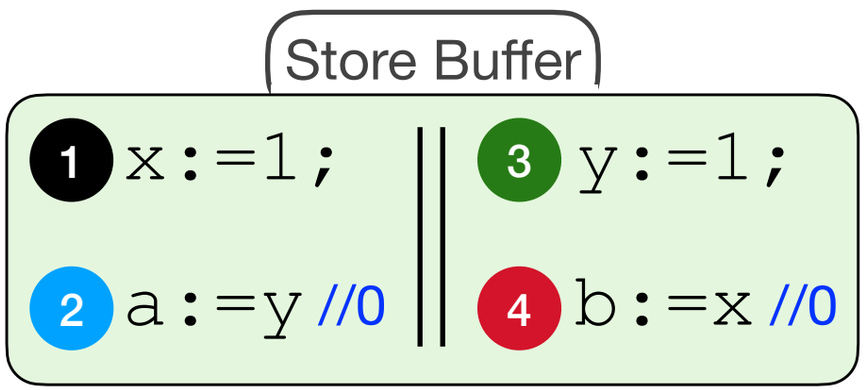
$$rfi \cup moi \cup rbi \subseteq po \quad \text{(Internal)}$$

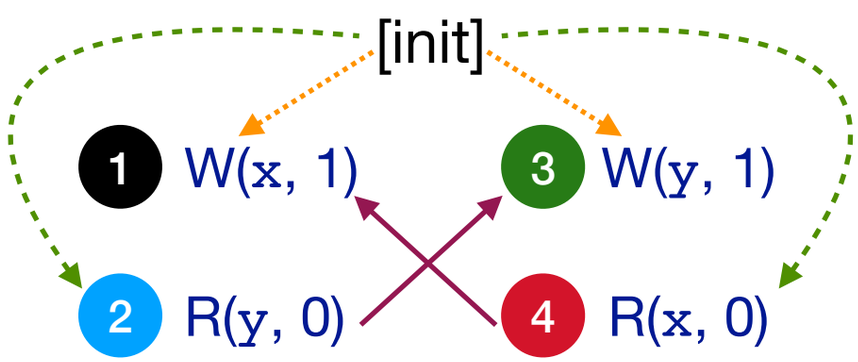
$$\text{irreflexive}(ob) \qquad ob = ((ppo \cup rfe \cup moe \cup rbe)^{+} \quad \text{(External)}$$

preserved program order: sloc or ✓ in table

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush$_{opt}$**	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush$_{opt}$**	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓





Ri : internal (same-thread) subset of R

Re : external (diff.-thread) subset of R: R \ Ri

Consistent Executions (w/o Persistency)

- What is a ***consistent*** execution?
  - Depends on the (concurrency) memory model
  - Intel-x86 consistency:

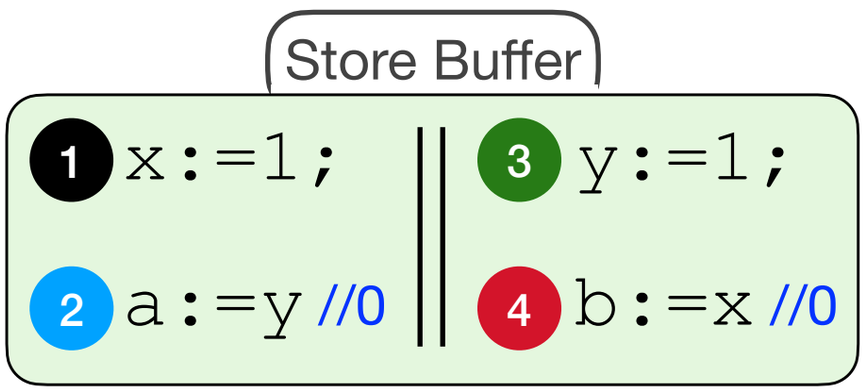
$$rfi \cup moi \cup rbi \subseteq po \quad \text{(Internal)}$$

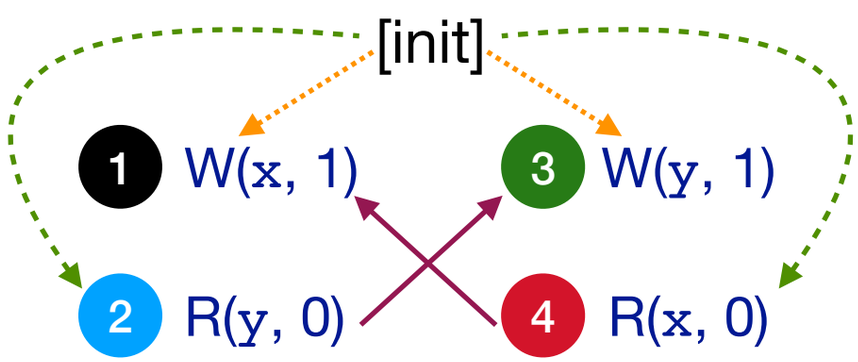
$$\text{irreflexive}(ob) \qquad ob = ((ppo \cup rfe \cup moe \cup rbe)^{+} \quad \text{(External)}$$

preserved program order: sloc or ✓ in table

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush$_{opt}$**	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush$_{opt}$**	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓





x86-consistent execution

Ri : internal (same-thread) subset of R

Re : external (diff.-thread) subset of R: R \ Ri

Declarative Consistency & Persistency Semantics

- Represent program behaviours as a set of consistent ***& persistent*** executions

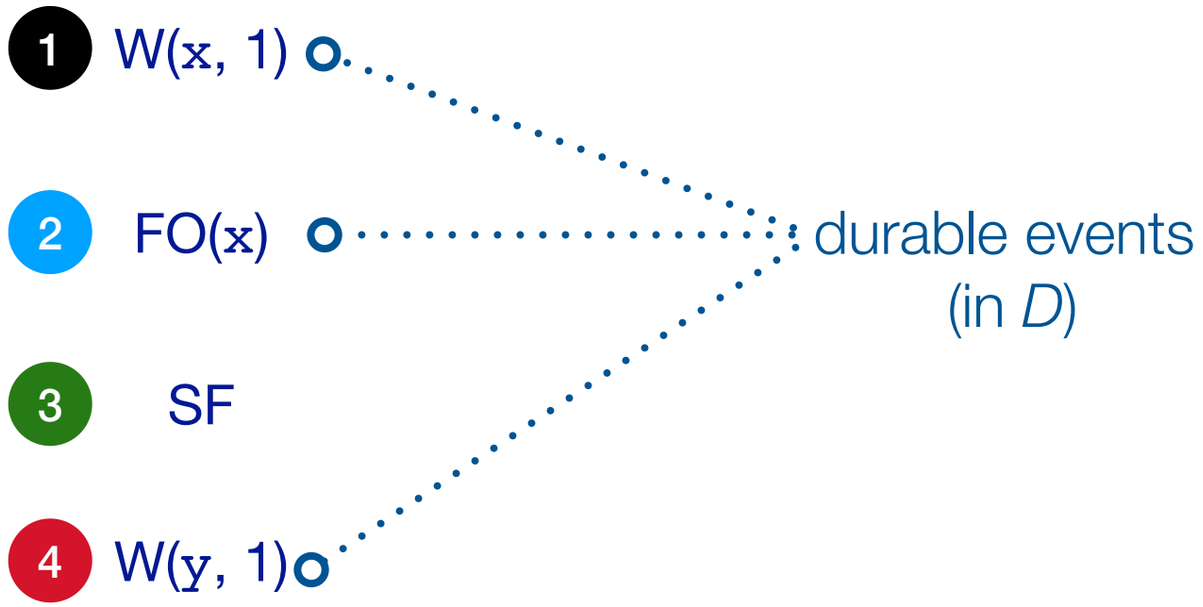
Declarative Consistency & Persistency Semantics

- Represent program behaviours as a set of consistent ***& persistent*** executions
- An execution graph is a tuple: < *E*, po, rf, mo, *P*, nvo >
  - Let $D \subseteq E$ be the set of *durable events*: events whose effect ***can reach*** NVM — model-specific for Intel-x86: $D = W \cup U \cup FL \cup FO$

Declarative Consistency & Persistency Semantics

- Represent program behaviours as a set of consistent ***& persistent*** executions
- An execution graph is a tuple: < *E*, po, rf, mo, *P*, nvo >
  - Let $D \subseteq E$ be the set of *durable events*: events whose effect ***can reach*** NVM — model-specific for Intel-x86: $D = W \cup U \cup FL \cup FO$

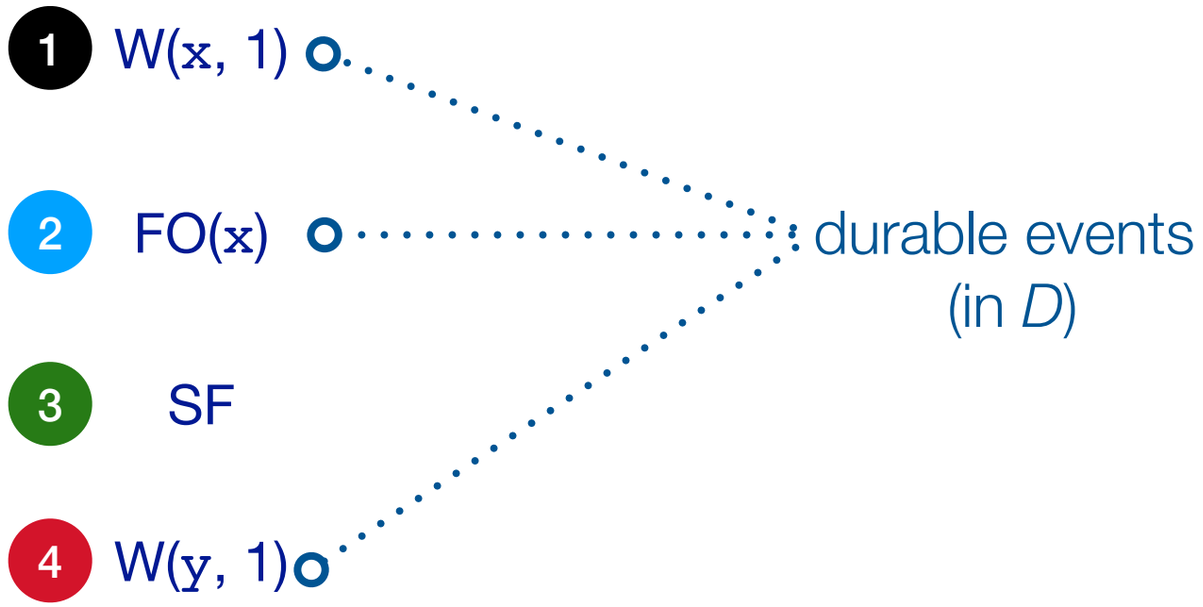
```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



Declarative Consistency & Persistency Semantics

- Represent program behaviours as a set of consistent ***& persistent*** executions
- An execution graph is a tuple: < *E*, po, rf, mo, *P*, nvo >
  - Let $D \subseteq E$ be the set of *durable events*: events whose effect ***can reach*** NVM — model-specific for Intel-x86: $D = W \cup U \cup FL \cup FO$
  - $P \subseteq D$ is the set of *persisted events*: events whose effect ***have reached*** NVM, s.t. init $\subseteq P$

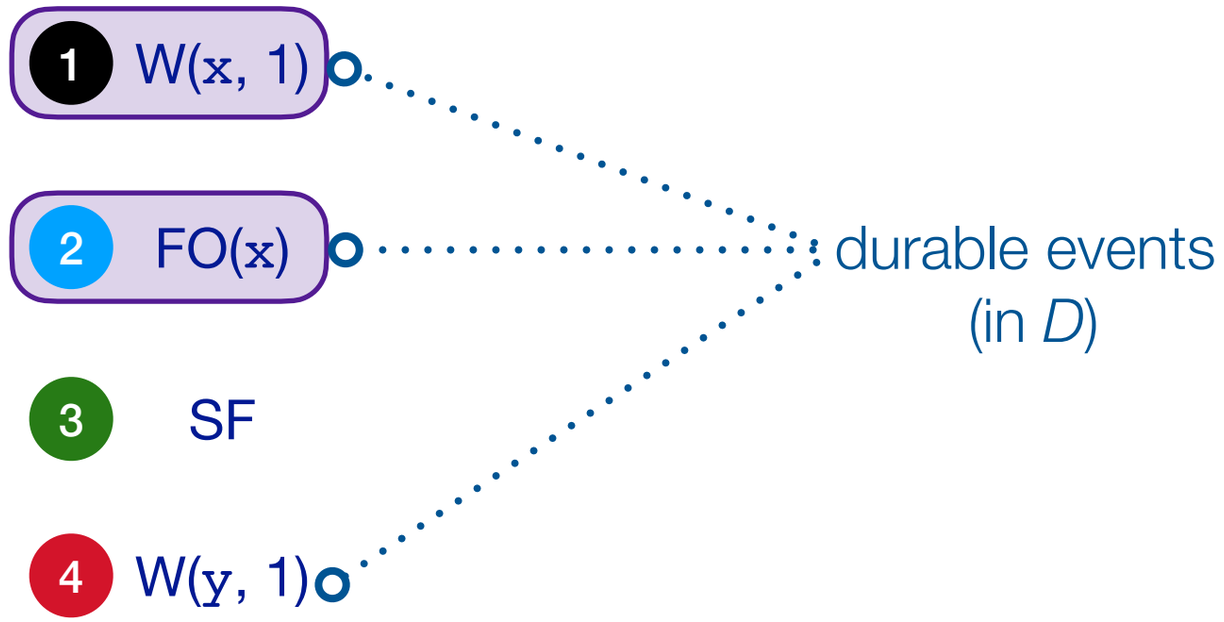
```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



Declarative Consistency & Persistency Semantics

- Represent program behaviours as a set of consistent ***& persistent*** executions
- An execution graph is a tuple: < *E*, po, rf, mo, *P*, nvo >
  - Let $D \subseteq E$ be the set of *durable events*: events whose effect ***can reach*** NVM — model-specific for Intel-x86: $D = W \cup U \cup FL \cup FO$
  - $P \subseteq D$ is the set of *persisted events*: events whose effect ***have reached*** NVM, s.t. init $\subseteq P$

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```

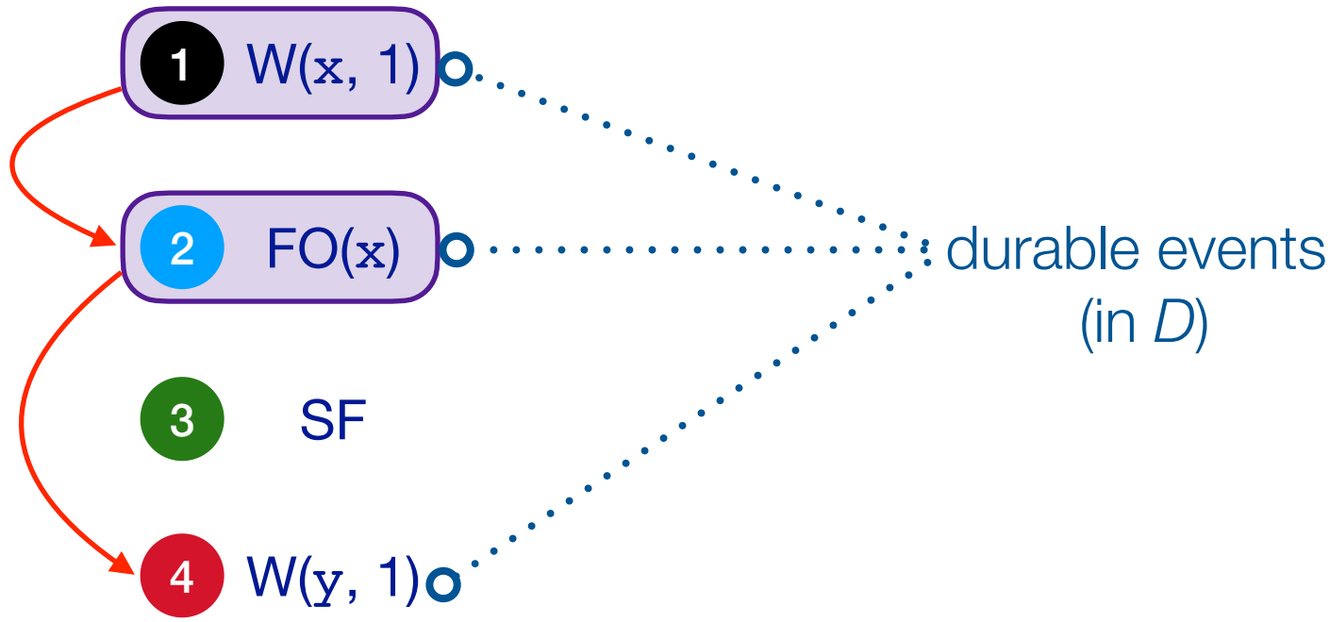


i : persisted event (in *P*)

Declarative Consistency & Persistency Semantics

- Represent program behaviours as a set of consistent ***& persistent*** executions
- An execution graph is a tuple: < *E*, po, rf, mo, *P*, nvo >
  - Let $D \subseteq E$ be the set of *durable events*: events whose effect ***can reach*** NVM — model-specific for Intel-x86: $D = W \cup U \cup FL \cup FO$
  - $P \subseteq D$ is the set of *persisted events*: events whose effect ***have reached*** NVM, s.t. init $\subseteq P$
  - nvo $\subseteq D \times D$ is the *non-volatile-order*: a strict (partial) order on *D* that is downward-closed on *P*: if (e, e') $\in$ nvo and e' $\in P$, then e $\in P$

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in *P*)

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $irreflexive(ob) \quad ob = (ppo \cup rfe \cup moe \cup rbe)^+$

  ppo: preserved program order (sloc or ✓ in table)

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $irreflexive(ob) \quad ob=((ppo \cup rfe \cup moe \cup rbe)^+$

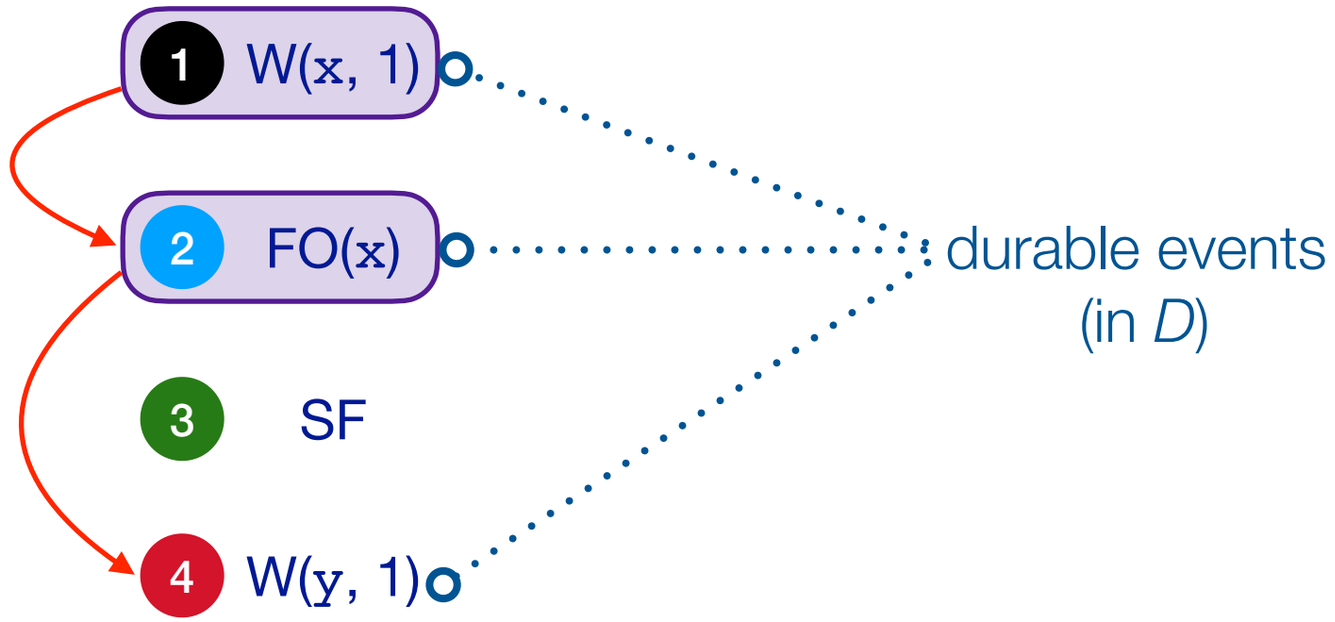
  ppo: preserved program order (sloc or ✓ in table)

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

- Intel-x86 ***persistency*** (simplified):

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in *P*)

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $irreflexive(ob) \quad ob = (ppo \cup rfe \cup moe \cup rbe)^+$

  ppo: preserved program order (sloc or ✓ in table)

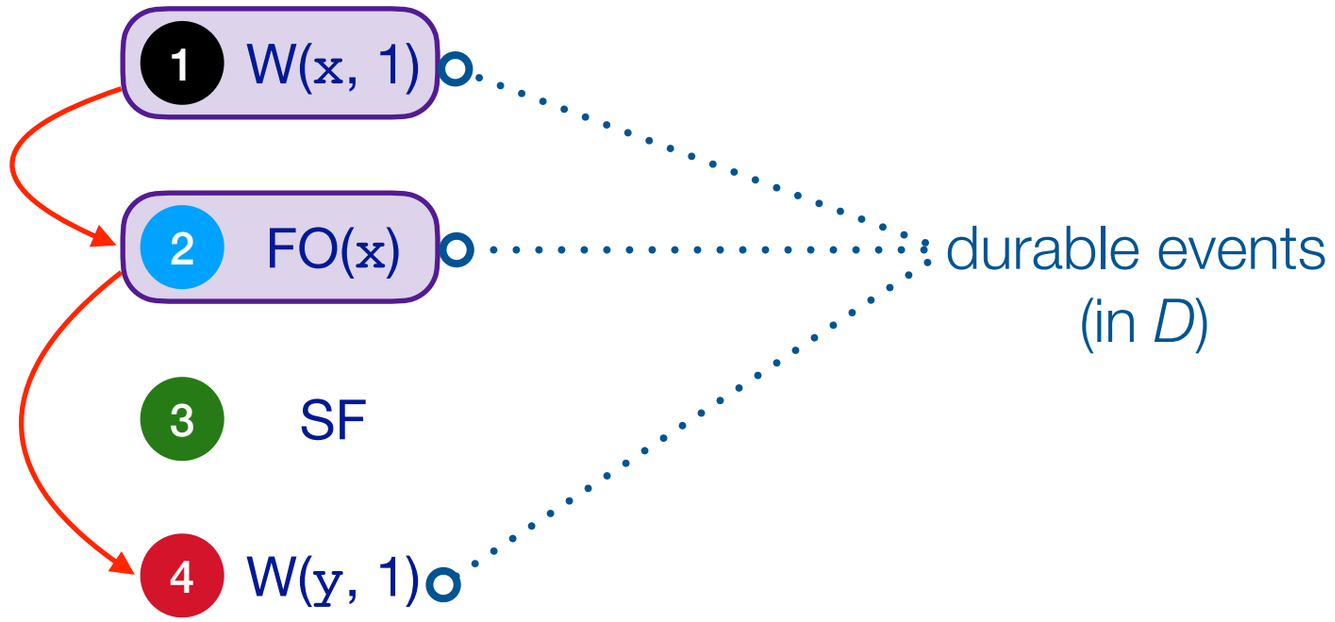
Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

- Intel-x86 ***persistency*** (simplified):

  $FL \cup dom(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in *P*)

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $irreflexive(ob) \quad ob = (ppo \cup rfe \cup moe \cup rbe)^+$

  ppo: preserved program order (sloc or ✓ in table)

Later in Program Order

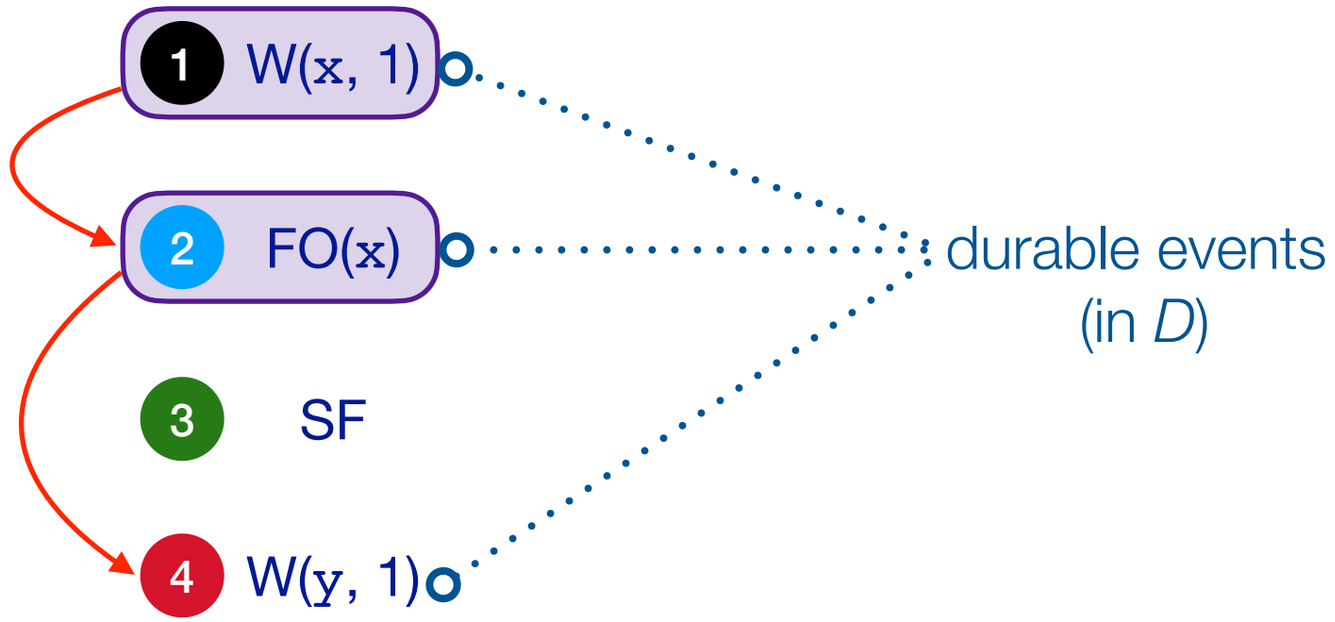
Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

- Intel-x86 ***persistency*** (simplified):

  $FL \cup dom(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$

  FL: strong persists

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in *P*)

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $irreflexive(ob) \quad ob = (ppo \cup rfe \cup moe \cup rbe)^+$

  ppo: preserved program order (sloc or ✓ in table)

Later in Program Order

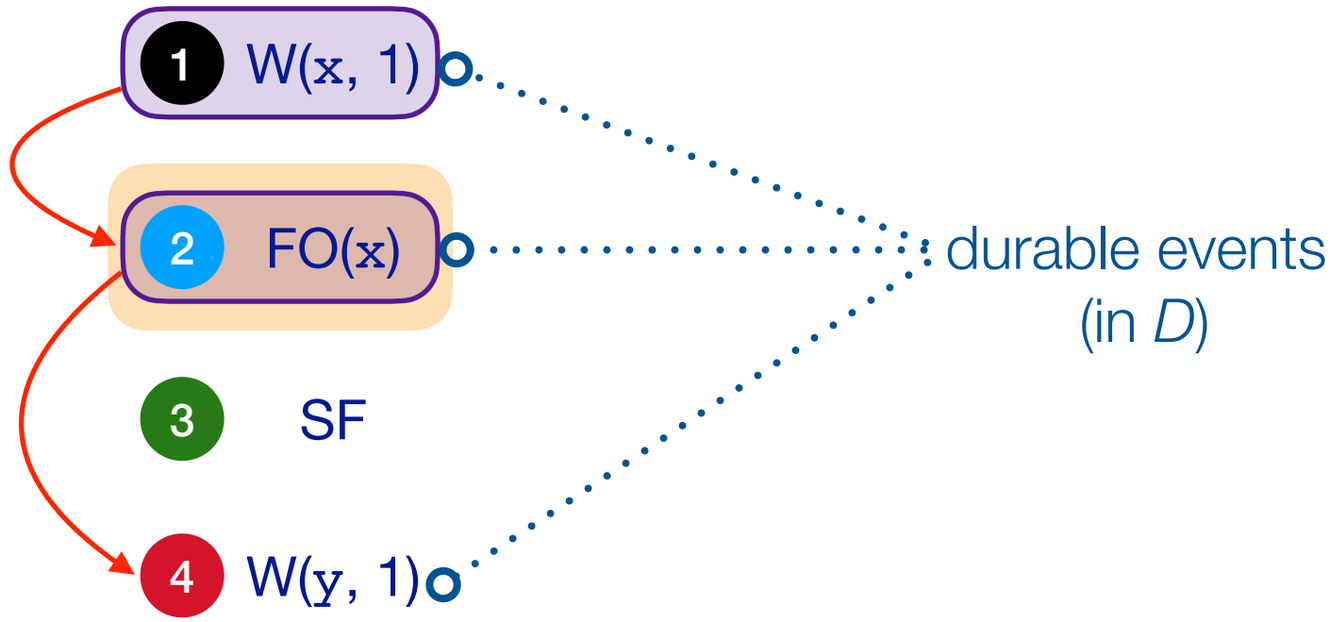
Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

- Intel-x86 ***persistency*** (simplified):

  $FL \cup dom(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$

  FL: strong persists; $FO \xrightarrow{po} SF \cup MF \cup U$: persist sequences

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in *P*)

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $irreflexive(ob)$ $\quad ob = (ppo \cup rfe \cup moe \cup rbe)^+$

  ppo: preserved program order (sloc or ✓ in table)

Later in Program Order

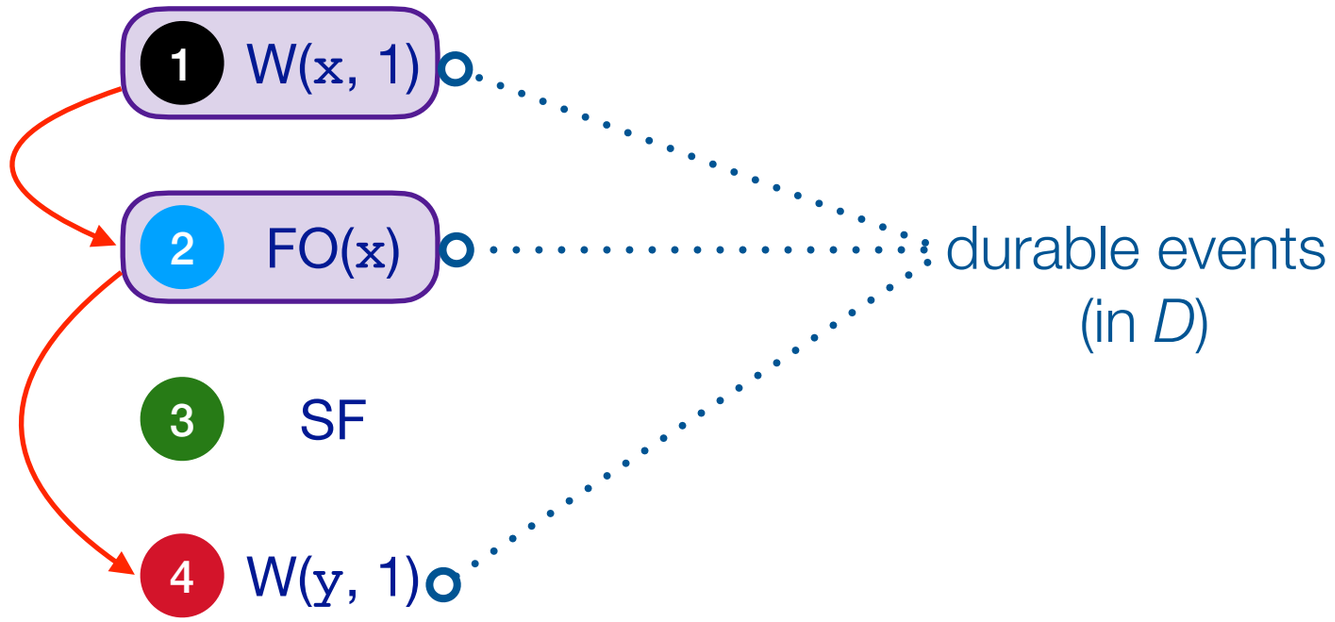
Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

- Intel-x86 ***persistency*** (simplified):

  $FL \cup dom(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$

  $D \xrightarrow{ob \cap sloc} D \subseteq nvo$

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in *P*)

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $irreflexive(ob)$ $\quad ob = (ppo \cup rfe \cup moe \cup rbe)^+$

  ppo: preserved program order (sloc or ✓ in table)

Later in Program Order

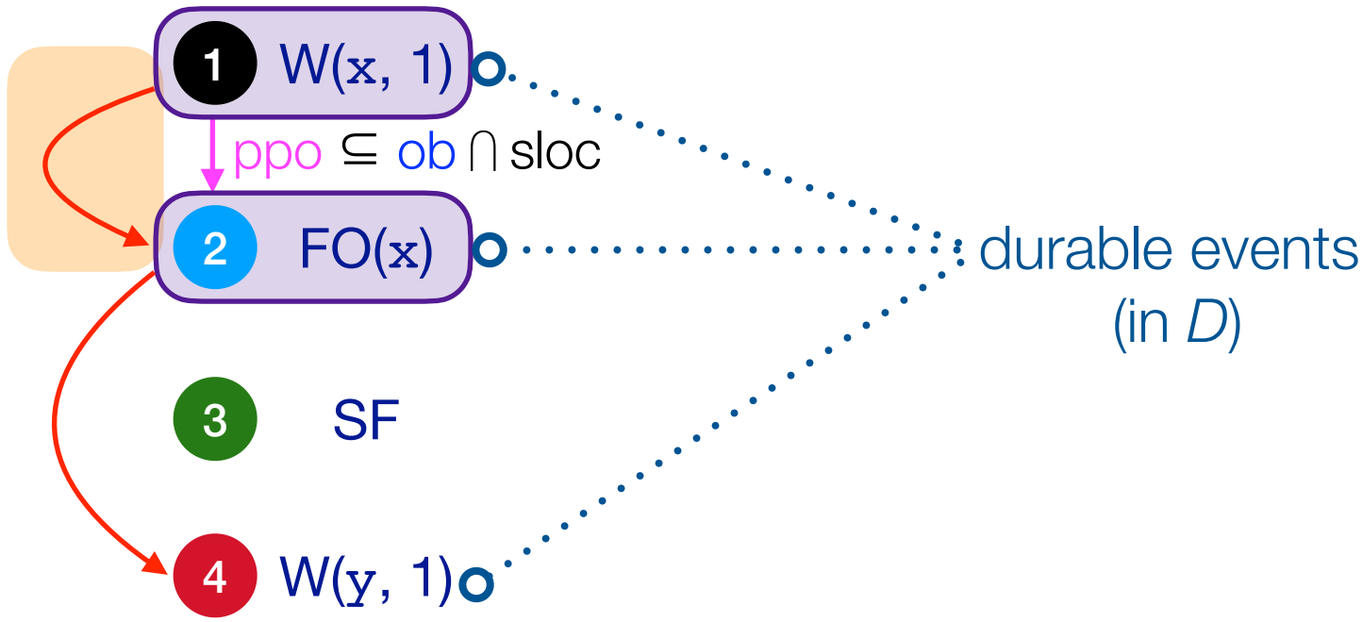
Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

- Intel-x86 ***persistency*** (simplified):

  $FL \cup dom(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$

  $D \xrightarrow{ob \cap sloc} D \subseteq nvo$

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in *P*)

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $irreflexive(ob)$ $\quad ob = (ppo \cup rfe \cup moe \cup rbe)^+$

  ppo: preserved program order (sloc or ✓ in table)

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

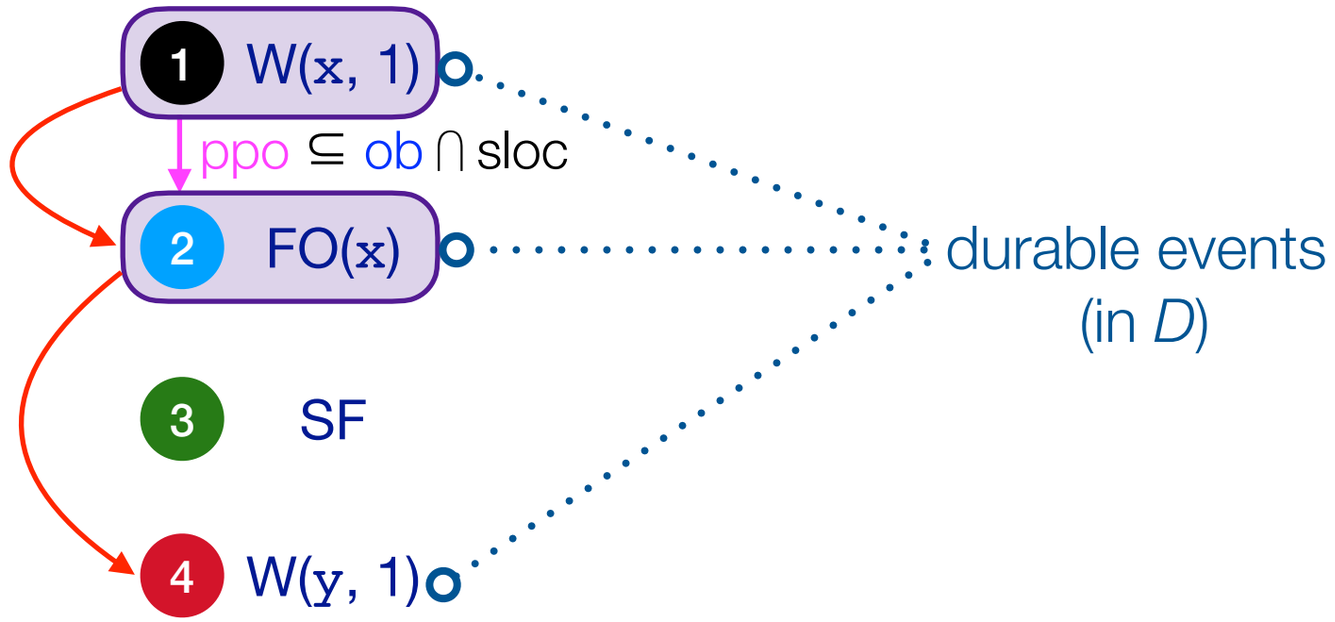
- Intel-x86 ***persistency*** (simplified):

  $FL \cup dom(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$

  $D \xrightarrow{ob \cap sloc} D \subseteq nvo$

  $(FO \cup FL) \xrightarrow{ob} D \subseteq nvo$

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in *P*)

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $irreflexive(ob)$ $\quad ob = (ppo \cup rfe \cup moe \cup rbe)^+$

  ppo: preserved program order (sloc or ✓ in table)

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

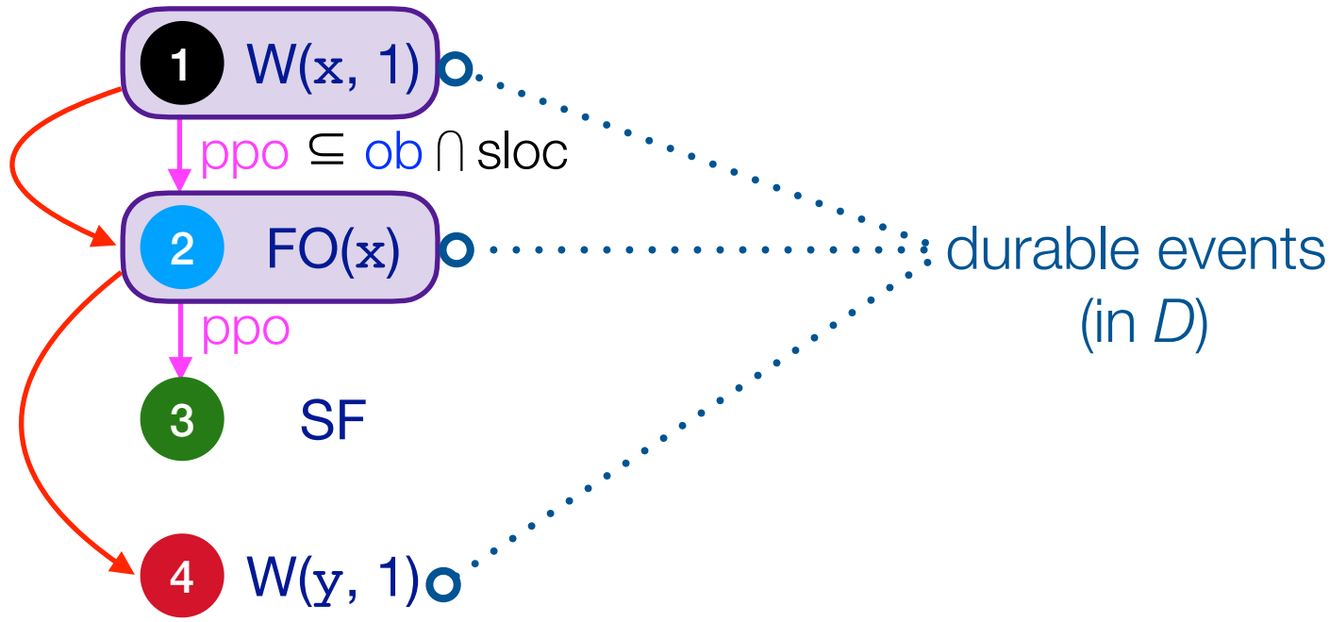
- Intel-x86 ***persistency*** (simplified):

  $FL \cup dom(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$

  $D \xrightarrow{ob \cap sloc} D \subseteq nvo$

  $(FO \cup FL) \xrightarrow{ob} D \subseteq nvo$

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in *P*)

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $irreflexive(ob)$ $\quad ob = (ppo \cup rfe \cup moe \cup rbe)^+$

  ppo: preserved program order (sloc or ✓ in table)

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

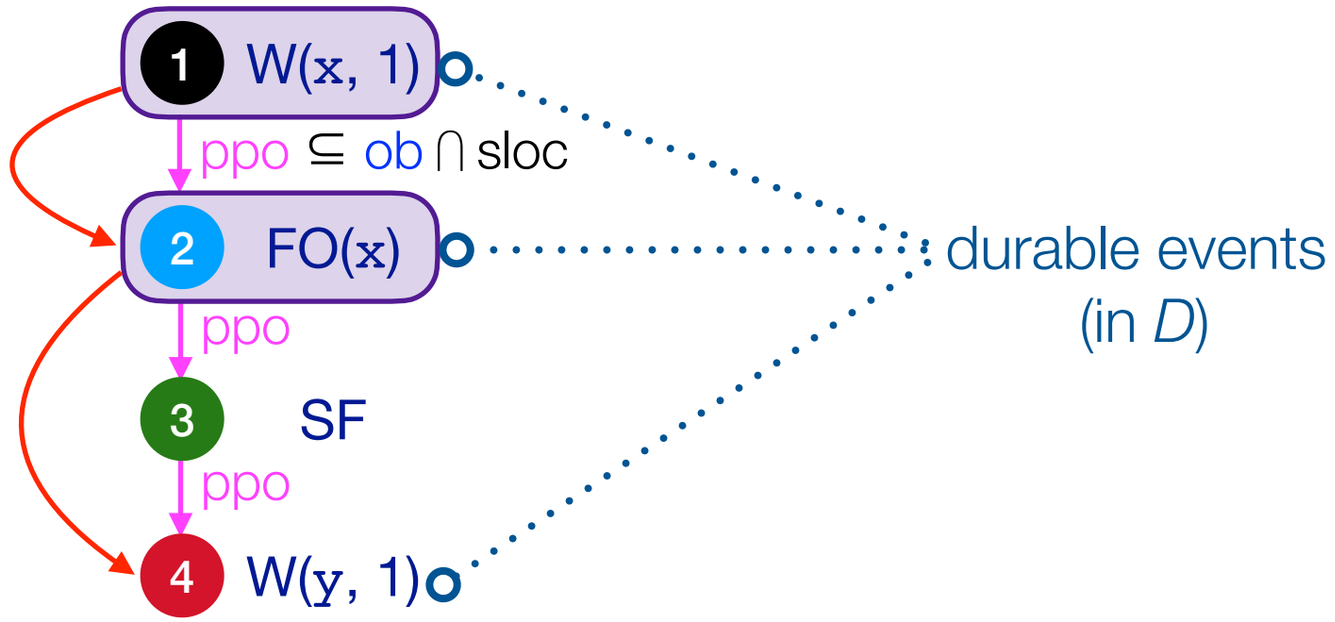
- Intel-x86 ***persistency*** (simplified):

  $FL \cup dom(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$

  $D \xrightarrow{ob \cap sloc} D \subseteq nvo$

  $(FO \cup FL) \xrightarrow{ob} D \subseteq nvo$

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in *P*)

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $\text{irreflexive}(ob) \qquad ob = (ppo \cup rfe \cup moe \cup rbe)^+$

  ppo: preserved program order (sloc or ✓ in table)

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

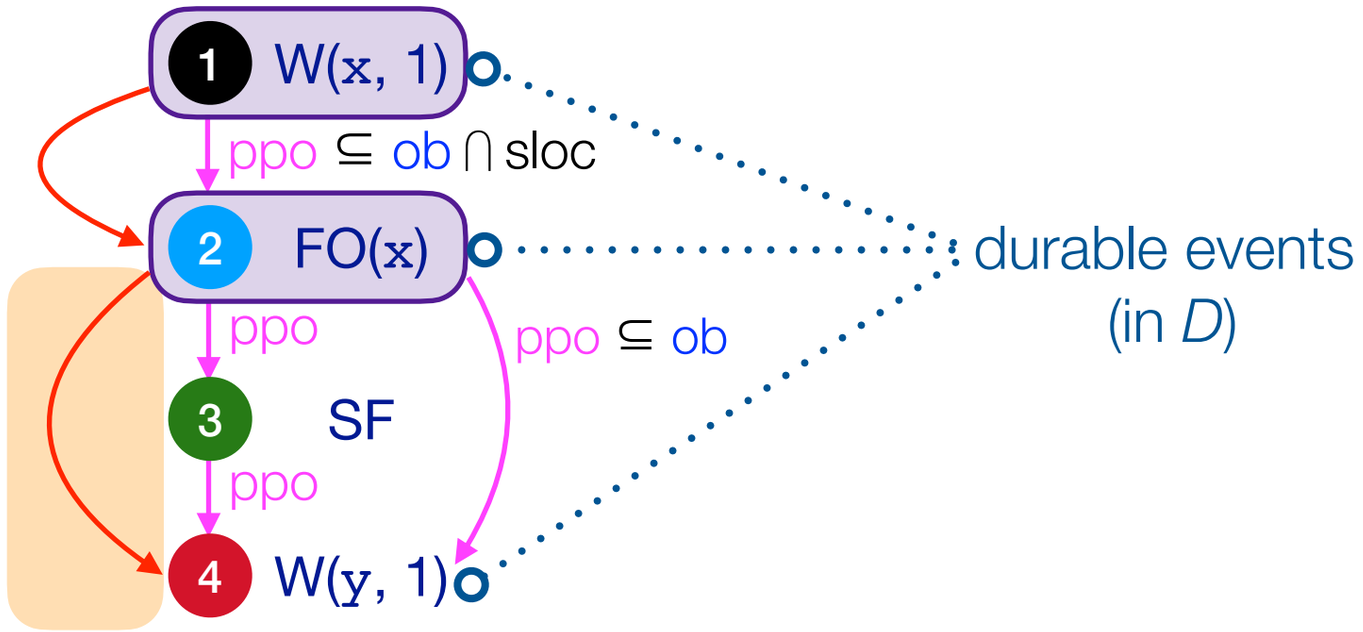
- Intel-x86 ***persistency*** (simplified):

  $FL \cup \text{dom}(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$

  $D \xrightarrow{ob \cap sloc} D \subseteq nvo$

  $(FO \cup FL) \xrightarrow{ob} D \subseteq nvo$

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in *P*)

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $irreflexive(ob)$ $\quad ob = (ppo \cup rfe \cup moe \cup rbe)^+$

  ppo: preserved program order (sloc or ✓ in table)

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

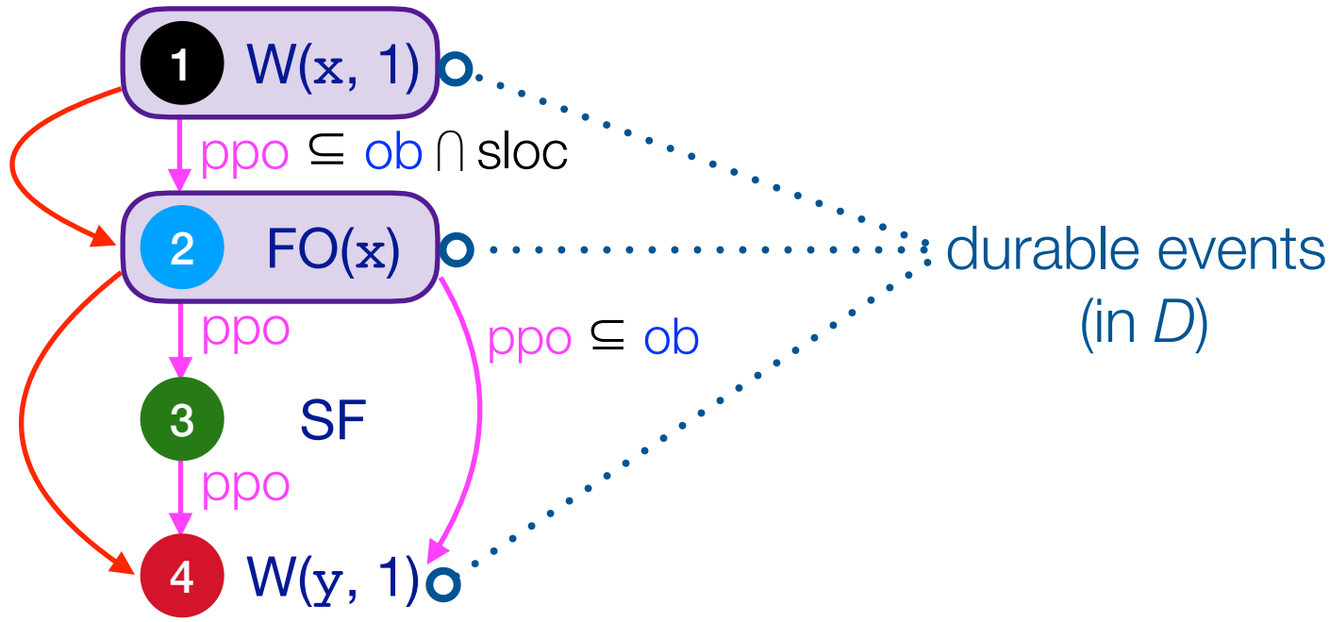
- Intel-x86 ***persistency*** (simplified):

  $FL \cup dom(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$

  $D \xrightarrow{ob \cap sloc} D \subseteq nvo$

  $(FO \cup FL) \xrightarrow{ob} D \subseteq nvo$

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in P)

Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:

  $rfi \cup moi \cup rbi \subseteq po$

  $irreflexive(ob)$ $\quad ob = (ppo \cup rfe \cup moe \cup rbe)^+$

  ppo: preserved program order (sloc or ✓ in table)

Later in Program Order

Earlier in Program Order		1 Read	2 Write	3 RMW	4 **mfence**	5 **sfence**	6 **flush**$_{opt}$	7 **flush**
A	Read	✓	✓	✓	✓	✓	✓	✓
B	Write	✗	✓	✓	✓	✓	sloc	✓
C	RMW	✓	✓	✓	✓	✓	✓	✓
D	**mfence**	✓	✓	✓	✓	✓	✓	✓
E	**sfence**	✗	✓	✓	✓	✓	✓	✓
F	**flush**$_{opt}$	✗	✗	✓	✓	✓	✗	sloc
G	**flush**	✗	✓	✓	✓	✓	sloc	✓

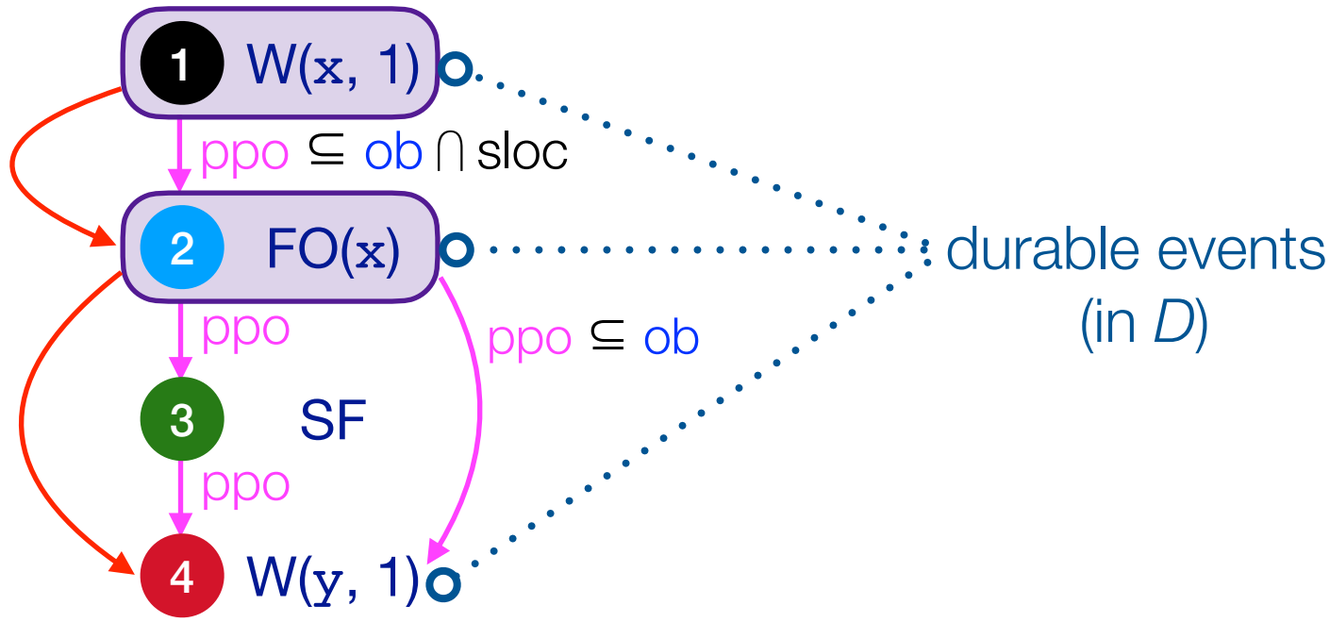
- Intel-x86 ***persistency*** (simplified):

  $FL \cup dom(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$

  $D \xrightarrow{ob \cap sloc} D \subseteq nvo$

  $(FO \cup FL) \xrightarrow{ob} D \subseteq nvo$

```
1 x:=1;
2 clflushopt x;
3 sfence;
4 y:=1
```



i : persisted event (in *P*)

x86-valid execution

4. Other hardware persistency models

ARMv8 Consistency & Persistency Models

- ARMv8 ***Consistency***
  - A complex model; much weaker than Intel-x86 consistency

ARMv8 Consistency & Persistency Models

- ARMv8 ***Consistency***
  - A complex model; much weaker than Intel-x86 consistency
  - Operational model: promising semantics by Kang et al.

ARMv8 Consistency & Persistency Models

- ARMv8 ***Consistency***
  - A complex model; much weaker than Intel-x86 consistency
  - Operational model: promising semantics by Kang et al.
  - Declarative model: by Pulte et al. — defines the ob relation

ARMv8 Consistency & Persistency Models

- ARMv8 ***Consistency***
  - A complex model; much weaker than Intel-x86 consistency
  - Operational model: promising semantics by Kang et al.
  - Declarative model: by Pulte et al. — defines the ob relation
- ARMv8 ***Persistency***
  - Weak explicit persists: `DC CVAP x` — analogous to `clflushopt x`

ARMv8 Consistency & Persistency Models

- ARMv8 ***Consistency***
  - A complex model; much weaker than Intel-x86 consistency
  - Operational model: promising semantics by Kang et al.
  - Declarative model: by Pulte et al. — defines the ob relation
- ARMv8 ***Persistency***
  - Weak explicit persists: `DC CVAP x` — analogous to `clflushopt x`
  - Persist sequences: `DC CVAP x; DSB SY` — `DSB SY` is a strong fence (analogous to `mfence`)

ARMv8 Consistency & Persistency Models

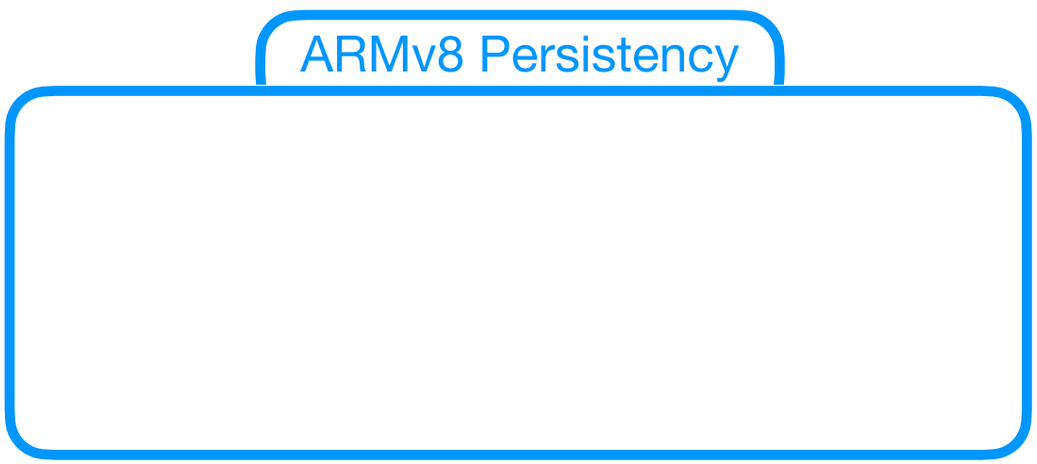
- ARMv8 ***Consistency***
  - A complex model; much weaker than Intel-x86 consistency
  - Operational model: promising semantics by Kang et al.
  - Declarative model: by Pulte et al. — defines the ob relation
- ARMv8 ***Persistency***
  - Weak explicit persists: `DC CVAP x` — analogous to `clflushopt x`
  - Persist sequences: `DC CVAP x; DSB SY` — `DSB SY` is a strong fence (analogous to `mfence`)
  - Operational model: an extension of promising semantics by Cho et al. [5]

ARMv8 Consistency & Persistency Models

- ARMv8 ***Consistency***
  - A complex model; much weaker than Intel-x86 consistency
  - Operational model: promising semantics by Kang et al.
  - Declarative model: by Pulte et al. — defines the ob relation
- ARMv8 ***Persistency***
  - Weak explicit persists: `DC CVAP x` — analogous to `clflushopt x`
  - Persist sequences: `DC CVAP x; DSB SY` — `DSB SY` is a strong fence (analogous to `mfence`)
  - Operational model: an extension of promising semantics by Cho et al. [5]
  - Declarative model: by Raad et al. [4], and Cho et al. [5] — (simplified)

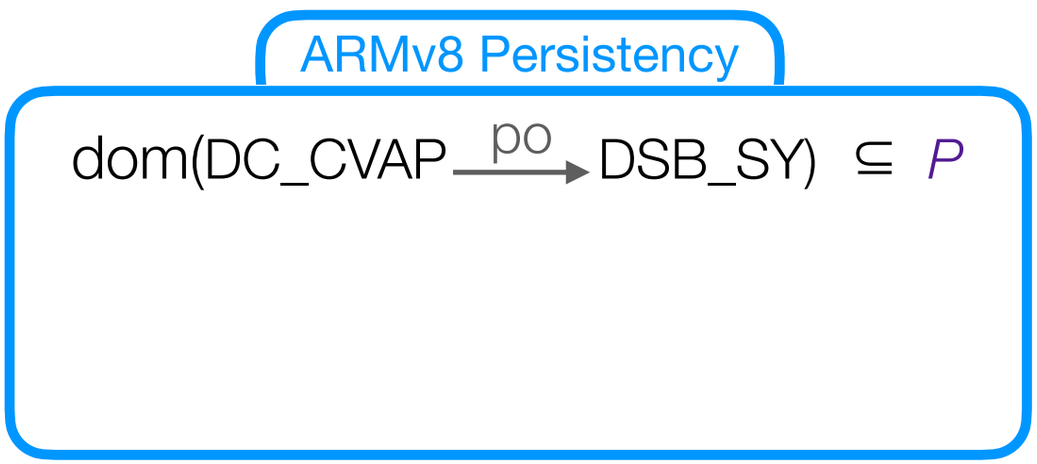
ARMv8 Consistency & Persistency Models

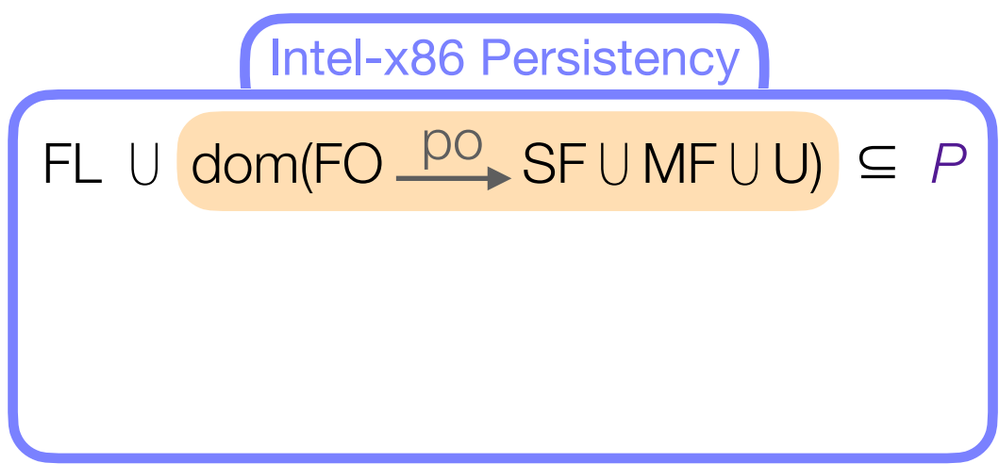
- ARMv8 ***Consistency***
  - A complex model; much weaker than Intel-x86 consistency
  - Operational model: promising semantics by Kang et al.
  - Declarative model: by Pulte et al. — defines the ob relation
- ARMv8 ***Persistency***
  - Weak explicit persists: `DC CVAP x` — analogous to `clflushopt x`
  - Persist sequences: `DC CVAP x; DSB SY` — `DSB SY` is a strong fence (analogous to `mfence`)
  - Operational model: an extension of promising semantics by Cho et al. [5]
  - Declarative model: by Raad et al. [4], and Cho et al. [5] — (simplified)



ARMv8 Consistency & Persistency Models

- ARMv8 ***Consistency***
  - A complex model; much weaker than Intel-x86 consistency
  - Operational model: promising semantics by Kang et al.
  - Declarative model: by Pulte et al. — defines the ob relation
- ARMv8 ***Persistency***
  - Weak explicit persists: `DC CVAP x` — analogous to `clflushopt x`
  - Persist sequences: `DC CVAP x; DSB SY` — `DSB SY` is a strong fence (analogous to `mfence`)
  - Operational model: an extension of promising semantics by Cho et al. [5]
  - Declarative model: by Raad et al. [4], and Cho et al. [5] — (simplified)





ARMv8 Consistency & Persistency Models

- ARMv8 ***Consistency***
  - A complex model; much weaker than Intel-x86 consistency
  - Operational model: promising semantics by Kang et al.
  - Declarative model: by Pulte et al. — defines the ob relation
- ARMv8 ***Persistency***
  - Weak explicit persists: `DC CVAP x` — analogous to `clflushopt x`
  - Persist sequences: `DC CVAP x; DSB SY` — `DSB SY` is a strong fence (analogous to `mfence`)
  - Operational model: an extension of promising semantics by Cho et al. [5]
  - Declarative model: by Raad et al. [4], and Cho et al. [5] — (simplified)

ARMv8 Persistency

$$dom(DC\_CVAP \xrightarrow{po} DSB\_SY) \subseteq P$$

$$D \xrightarrow{ob \cap sloc} D \subseteq nvo$$

Intel-x86 Persistency

$$FL \cup dom(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$$

$$D \xrightarrow{ob \cap sloc} D \subseteq nvo$$

ARMv8 Consistency & Persistency Models

- ARMv8 ***Consistency***
  - A complex model; much weaker than Intel-x86 consistency
  - Operational model: promising semantics by Kang et al.
  - Declarative model: by Pulte et al. — defines the ob relation
- ARMv8 ***Persistency***
  - Weak explicit persists: `DC CVAP x` — analogous to `clflushopt x`
  - Persist sequences: `DC CVAP x; DSB SY` — `DSB SY` is a strong fence (analogous to `mfence`)
  - Operational model: an extension of promising semantics by Cho et al. [5]
  - Declarative model: by Raad et al. [4], and Cho et al. [5] — (simplified)

ARMv8 Persistency

$$dom(DC\_CVAP \xrightarrow{po} DSB\_SY) \subseteq P$$

$$D \xrightarrow{ob \cap sloc} D \subseteq nvo$$

$$(DC\_CVAP) \xrightarrow{ob} D \subseteq nvo$$

Intel-x86 Persistency

$$FL \cup dom(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P$$

$$D \xrightarrow{ob \cap sloc} D \subseteq nvo$$

$$(FL \cup FO) \xrightarrow{ob} D \subseteq nvo$$

5. Further Reading

A. Intel-x86 / TSO Persistency Models

[1] *Persistence Semantics for Weak Memory: Integrating Epoch Persistency with the TSO Memory Model*
Azalea Raad, Viktor Vafeiadis

[2] *Persistency Semantics of the Intel-x86 Architecture*
Azalea Raad, John Wickerson, Gil Neiger, Viktor Vafeiadis

[3] *Taming x86-TSO Persistency*
Artem Khyzha, Ori Lahav

B. ARMv8 Persistency Models

[4] *Weak Persistency Semantics from the Ground Up: Formalising the Persistency Semantics of ARMv8 and Transactional Models*
Azalea Raad, John Wickerson, Viktor Vafeiadis

[5] *Revamping Hardware Persistency Models: View-Based and Axiomatic Persistency Models for Intel-x86 and Armv8*
Kyeongmin Cho, Sung-Hwan Lee, Azalea Raad, Jeehoon Kang

C. Persistent Verification

[6] *Linearizability of Persistent Memory Objects Under a Full-System-Crash Failure Model*
Joseph Izraelevitz, Hammurabi Mendes, Michael L. Scott

[7] *Persistent Owicki-Gries Reasoning: A Program Logic for Reasoning about Persistent Programs on Intel-x86*
Azalea Raad, Ori Lahav, Viktor Vafeiadis

[8] *Defining and Verifying Durable Opacity: Correctness for Persistent Software Transactional Memory*
Eleni Bila, Simon Doherty, Brijesh Dongol, John Derrick, Gerhard Schellhorn, Heike Wehrheim

[9] *PerSeVerE: Persistency Semantics for Verification under Ext4*
Michalis Kokologiannakis, Ilya Kaisin, Azalea Raad, Viktor Vafeiadis

[10] *Deciding reachability under persistent x86-TSO*
Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, Prakash Saivasan