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

RAM

HDD
Computer Storage

✓ fast
✗ volatile
Computer Storage

✓ fast
✗ volatile

✓ fast
✗ volatile

✗ slow
✓ persistent

RAM

HDD
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

```plaintext
// x = 0
x := 1
// x = 1
```
Q: Why *Formal* NVM Semantics?

Volatile memory

```plaintext
// 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?

**Volatile** memory

\[
\begin{align*}
  &// x = 0 \\
  x &:= 1 \\
  &// x = 1 \\
  &// no recovery \\
  &// x = 0
\end{align*}
\]

**Non-Volatile** memory

\[
\begin{align*}
  &// x = 0 \\
  x &:= 1 \\
  &// x = 1 \\
  &// recovery routine \\
  &// x = 1
\end{align*}
\]

A: Program *Verification*
Q: Why *Formal* NVM Semantics?

What about **Concurrency**?

```plaintext
// x = y = ... = 0
C_1 \parallel C_2 \parallel \ldots \parallel C_n
// ???

// recovery routine
// ???
```
Formal Semantic Models

Difficulty

Sequential (1940s)

time
Formal Semantic Models

Difficulty

Sequential (1940s)

SC (1979)

time
Formal Semantic Models

Difficulty

time

Sequential (1940s)

SC (1979)

WMC (1990s)
Weak Memory Consistency (WMC)

No total execution order ($to$) $\Rightarrow$

weak behaviour absent under SC, caused by:

- instruction **reordering** by compiler
- write propagation across **cache hierarchy**
WMC: Store Buffering

1. \(x := 1;\)
2. \(a := y\)
3. \(y := 1;\)
4. \(b := x\)

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
WMC: Store Buffering

1. \( x := 1; \)
2. \( a := y \)
3. \( y := 1; \)
4. \( b := x \)

\[
\begin{array}{cc}
0 & 1 \\
1 & 0 \\
1 & 1 \\
0 & 0 \\
\end{array}
\]

possible, due to reordering!
WMC: Store Buffering

possible, due to reordering!  

store buffering(SB)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

a:=y;  b:=x;
Weak Memory Consistency (WMC)

No total execution order (to) $\Rightarrow$

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 behaviour absent under SC, caused by:

- instruction reordering by compiler
- write propagation across cache hierarchy

**Consistency Model**

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

e.g. x86 (TSO), ARMv8, C11, Java
Formal Semantic Models

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>😞</td>
<td>(1940s)</td>
</tr>
<tr>
<td>😐</td>
<td>(1979)</td>
</tr>
<tr>
<td>😄</td>
<td>(1990s)</td>
</tr>
</tbody>
</table>

Sequential (1940s)  SC (1979)  WMC (1990s)
Formal Semantic Models

Difficulty

This Talk

Sequential (1940s)
SC (1979)
WMC (1990s)
WNVMC (2017)

time
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
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: Pxs6
   ✦ Warmup: *Sequential* Pxs6
   ✦ *Concurrent* Pxs6

2. A *formal* account Pxs6: *operational* semantics

3. A *formal* account Pxs6: *declarative* semantics

4. Other Low-level (hardware) persistency models

5. Further reading
1. An *intuitive* account of \( P_{x86} \)

Warmup: *Sequential* \( P_{x86} \)
Sequential Hardware

(CPU) Memory

(write) read
Sequential Hardware

\[ x := 1 \text{ : adds } x := 1 \text{ to memory} \]
Sequential Hardware

- **CPU**
  - READ: \( x := 1 \): adds \( x := 1 \) to memory
  - WRITE: \( a := x \): reads \( x \) from memory

(Volatile) Memory
Sequential Hardware

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

a := x : reads x from memory

memory lost
Sequential Hardware

CPU

(Volatile) Memory

x := 1 : adds \( x := 1 \) to memory

a := x : reads x from memory

memory lost

Persistence Buffer

CPU

(Persistent) Memory
Sequential Hardware

- **CPU**
- **(Volatile) Memory**
  - $x := 1$: adds $x := 1$ to memory
  - $a := x$: reads $x$ from memory
  - ⚡ memory lost
- **Persistence Buffer**
  - $x := 1$: adds $x := 1$ to p-buffer
- **CPU**
- **(Persistent) Memory**

Diagram: Connections between CPU, Memory, and Persistence Buffer.
**Sequential Hardware**

- **CPU**
  - $x := 1$ : adds $x := 1$ to memory
  - $a := x$ : reads $x$ from memory
  - Memory lost

- **(Volatile) Memory**

- **Persistence Buffer**
  - $x := 1$ : adds $x := 1$ to p-buffer
  - $a := x$ : if p-buffer contains $x$, reads latest entry; else reads from memory

- **(Persistent) Memory**
Sequential Hardware

- **CPU**
  - `x := 1` : adds `x := 1` to memory
  - `a := x` : reads `x` from memory
  - ⚠️ memory lost

- **(Volatile) Memory**

- **CPU**
  - `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

CPU

(Volatile) Memory

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

a:=x : reads x from memory

memory lost

CPU

Persistence Buffer

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

CPU

(Volatile) Memory

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

a:=x : reads x from memory

memory lost

CPU

Persistence Buffer

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* Persistes

```
// 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

generic *explicit persists?*
Explicit Persists: *Desiderata*

```plaintext
// 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 Persist: `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: \textbf{clwb}, \textbf{clflushopt}, \textbf{clflush}

- \textbf{clwb} and \textbf{clflushopt}: same \textbf{ordering} constraints
- \textbf{clwb} does not \textbf{invalidate} cache line
- \textbf{clflushopt} \textbf{invalidates} cache line
- \textbf{clflush}: strongest \textbf{ordering} constraints; \textbf{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`

```plaintext
// 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**

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

clflushopt x / clwb x;

y := 1;

// x=1; y=0; x=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*

```plaintext
// 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**

```c
// 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)

Thread1

... Thread2

Buffer

Buffer

(Volatile) Memory

\[ x := 1 : \text{adds } x := 1 \text{ to buffer} \]
x86: (Volatile) Concurrent Hardware Model (TSO)

\[ x := 1 \quad \text{: adds } x := 1 \text{ to buffer} \]

unbuffer* : buffer to memory

* at non-deterministic times
x86: (Volatile) Concurrent Hardware Model (TSO)

\[\text{x:=1} : \text{adds } \text{x:=1} \text{ to buffer}\]

\[\text{unbuffer* : buffer to memory}\]

\[\text{a:=x} : \text{if } \text{buffer contains } x, \text{reads latest entry else reads from memory}\]

* at non-deterministic times
x86: (Volatile) Concurrent Hardware Model (TSO)

Thread1  •••  Thread2

Buffer  Buffer

(Volatile) Memory

\[ x:=1 \] : adds \( x:=1 \) to buffer

unbuffer* : buffer to memory

\[ a:=x \] : if buffer contains \( x \), reads latest entry else reads from memory

\[ \text{buffer and memory lost} \]

* at non-deterministic times
Software WMC: Store Buffering

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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

possible, due to reordering!

store buffering (SB)

```plaintext
2 a := y;
1 x := 1
4 b := x;
3 y := 1
```
Hardware (Intel x86) WMC: Store Buffering
Hardware (Intel x86) WMC: Store Buffering
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

Thread1

\[ x = 1 \]

Thread2

\[ y := 1; \]
\[ b := x; \]

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

\[ x = 0; y = 0; \]

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

Thread1

\[ x = 1 \]

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

Thread2

\[ y = 1 \]

\[ y := 1; \]
\[ b := x; \]

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

Thread1

\[ x = 1 \]

\[ x := 1; \]
\[ a := y; // 0 \]

Thread2

\[ y = 1 \]

\[ y := 1; \]
\[ b := x; \]

Store Buffering (SB)
Hardware (Intel x86) WMC: Store Buffering

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

```
x = 1
y = 1
```

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

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

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

Store Buffering (SB)
Hardware (Intel x86) WMC: Store Buffering

Thread1

\[ x = 1 \]

Thread2

\[ y = 1 \]

\[ x = 0; \quad y = 0; \]

Thread1

\[ x := 1; \quad a := y; \quad // \quad 0 \]

Thread2

\[ y := 1; \quad b := x; \]

Store Buffering (SB)
Hardware (Intel x86) WMC: Store Buffering

Thread1
x := 1;
a := y; // 0

Thread2
y := 1;
b := x; // 0

Store Buffering (SB)
Hardware (Intel x86) WMC: Store Buffering

Thread1

\[ x := 1; \]
\[ a := y; \quad // \ 0 \]

Thread2

\[ y := 1; \]
\[ b := x; \quad // \ 0 \]

Store Buffering (SB)
Hardware (Intel x86) WMC: Store Buffering

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**

CPU

Persistence Buffer

(Persistent) Memory

**Concurrent, Volatile x86**

Thread1

Buffer

(Volatile) Memory

Thread2

Buffer

**Sequential, Persistent x86**

Thread1

Buffer

Persistence Buffer

(Persistent) Memory

Thread2

Buffer

(Volatile) Memory
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)

Thread1
Buffer
Persistence Buffer
Thread2
Buffer

(Persistent) Memory

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

NVM Semantics (Px86)
2. A *formal* account of Px86

*Operational* Semantics
Basic domains

\[ a \in \text{REG} \quad \text{Registers} \]
\[ v \in \text{VAL} \quad \text{Values} \]
\[ \tau \in \text{TId} \quad \text{Thread IDs} \]
Basic domains

\[ a \in \text{REG} \quad \text{Registers} \]
\[ v \in \text{VAL} \quad \text{Values} \]
\[ \tau \in \text{TID} \quad \text{Thread IDs} \]

Expressions and sequential commands

\[ \text{Exp} \ni e ::= v \mid a \mid e+e \mid \cdots \]
\[ \text{PCom} \ni c ::= \text{load} (x) \mid \text{store} (x, e) \mid \text{CAS} (x, e, e') \mid \text{FAA} (x, e) \]
\[ \quad \mid \text{mfence} \mid \text{sfence} \mid \text{flush}_{\text{opt}} x \mid \text{flush} x \]
\[ \text{Com} \ni C ::= e \mid c \mid \text{let } a := C \text{ in } C \]
\[ \quad \mid \text{if } (C) \text{ then } C \text{ else } C \mid \text{repeat } C \]
Px86 Programming Language

Basic domains

\[ a \in \text{REG} \quad \text{Registers} \]
\[ u \in \text{VAL} \quad \text{Values} \]
\[ \tau \in \text{TID} \quad \text{Thread IDs} \]

Expressions and sequential commands

\[ \text{Exp} \ni e ::= u \mid a \mid e + e \mid \cdots \]
\[ \text{PCom} \ni c ::= \text{load}(x) \mid \text{store}(x, e) \mid \text{CAS}(x, e, e') \mid \text{FAA}(x, e) \]
\[ \quad \mid \text{mfence} \mid \text{sfence} \mid \text{flush}_{\text{opt}} x \mid \text{flush} x \]
\[ \text{Com} \ni C ::= e \mid c \mid \text{let} a := C \text{ in } C \]
\[ \quad \mid \text{if } (C) \text{ then } C \text{ else } C \mid \text{repeat } C \]

RMW instructions

used for persistency
Px86 Programming Language

Basic domains

\[ a \in \text{REG} \quad \text{Registers} \]
\[ u \in \text{VAL} \quad \text{Values} \]
\[ \tau \in \text{TID} \quad \text{Thread IDs} \]

Expressions and sequential commands

\[ \text{Exp} \ni e ::= u \mid a \mid e+e \mid \cdots \]
\[ \text{PCom} \ni c ::= \text{load}(x) \mid \text{store}(x, e) \mid \text{CAS}(x, e, e') \mid \text{FAA}(x, e) \]
\[ \mid \text{mfence} \mid \text{sfence} \mid \text{flush}_{\text{opt}}(x) \mid \text{flush}(x) \]
\[ \text{Com} \ni C ::= e \mid c \mid \text{let} \ a:=C \ \text{in} \ C \]
\[ \mid \text{if} \ (C) \ \text{then} \ C \ \text{else} \ C \mid \text{repeat} \ C \]

Programs

\[ P \in \text{PROG} \triangleq \text{TID} \xrightarrow{\text{fin}} \text{COM} \]

\textit{RMW} instructions used for \textit{persistency}
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: $\text{COM} \xrightarrow{T_{\text{t}}: \text{LAB} \cup \{\varepsilon\}} \text{COM}$

$\text{LAB} \triangleq \{(R, x, v), (W, x, v), (U, x, v, v'), MF, SF, (FO, x), (FL, x) \mid x \in \text{Loc} \land v, v' \in \text{VAL}\}$
Px86 Program Transitions

**Thread transitions:** 
$\text{COM} \xrightarrow{\text{TId:LAB U } \{\varepsilon\}} \text{COM}$

$\text{LAB} \triangleq \{(R, x, v), (W, x, v), (U, x, v, v'), \text{MF}, \text{SF}, (\text{FO}, x), (\text{FL}, x) \mid x \in \text{Loc} \land v, v' \in \text{VAL}\}$

\[\text{load}(x) \xrightarrow{\tau:(R,x,v)} v\]  *(T-READ)*

\[\text{store}(x, v) \xrightarrow{\tau:(W,x,v)} v\]  *(T-WRITE)*
**Px86 Program Transitions**

**Thread transitions:** \( \text{COM} \xrightarrow{TID:LAB \cup \{\varepsilon\}} \text{COM} \)

\[ \text{LAB} \triangleq \{(R, x, v), (W, x, v), (U, x, v, v'), \text{MF}, \text{SF}, (F0, x), (FL, x) \mid x \in \text{Loc} \land v, v' \in \text{VAL}\} \]

- **(T-READ)**
  \[ \text{load}(x) \xrightarrow{\tau:(R,x,v)} v \]

- **(T-CAS1)**
  \[ \text{CAS}(x, v_1, v_2) \xrightarrow{\tau:(U,x,v_1,v_2)} 1 \]

- **(T-WRITE)**
  \[ \text{store}(x, v) \xrightarrow{\tau:(W,x,v)} v \]

- **(T-CAS0)**
  \[ v \neq v_1 \quad \text{CAS}(x, v_1, v_2) \xrightarrow{\tau:(R,x,v)} 0 \]
**Px86 Program Transitions**

Thread transitions: $\text{COM} \xrightarrow{\text{Tid: Lab} \cup \{ \varepsilon \}} \text{COM}$

$\text{Lab} \triangleq \{(R, x, v), (W, x, v), (U, x, v, v'), MF, SF, (F0, x), (FL, x) \mid x \in \text{Loc} \land v, v' \in \text{Val}\}$

- **(T-READ)**
  - $\text{load}(x) \xrightarrow{\tau:(R,x,v)} v$

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

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

- **(T-WRITE)**
  - $\text{store}(x, v) \xrightarrow{\tau:(W,x,v)} v$

- **(T-CAS0)**
  - $v \neq v_1 \xrightarrow{\tau:(R,x,v)} 0$
Px86 Program Transitions

\[ \text{Thread transitions: } \text{COM} \xrightarrow{T\text{Id}: \text{LAB} \cup \{\epsilon\}} \text{COM} \]

\[ \text{LAB} \triangleq \{(R, x, v), (W, x, v), (U, x, v, v'), \text{MF}, \text{SF}, (F0, x), (FL, x) \mid x \in \text{Loc} \land v, v' \in \text{VAL}\} \]

\[ \text{load}(x) \xrightarrow{T:\text{READ}} \text{v} \]

\[ \text{store}(x, v) \xrightarrow{T:\text{WRITE}} \text{v} \]

\[ \text{CAS}(x, v_1, v_2) \xrightarrow{T:\text{CAS1}} 1 \]

\[ \text{CAS}(x, v_1, v_2) \xrightarrow{T:\text{CAS0}} 0 \]

\[ \text{FAA}(x, v) \xrightarrow{T:\text{FAA}} v_0 \]

\[ \text{sfence} \xrightarrow{T:\text{SF}} 1 \]

\[ \text{mfence} \xrightarrow{T:\text{MF}} 1 \]
Px86 Program Transitions

**Thread transitions:** \( \text{COM} \xrightarrow{TId:Lab \cup \{\epsilon\}} \text{COM} \)

\( \text{Lab} \triangleq \{(R, x, v), (W, x, v), (U, x, v, v'), MF, SF, (F_0, x), (F_L, x) \mid x \in \text{Loc} \land v, v' \in \text{VAL}\} \)

- **(T-READ)**
  - \( \text{load}(x) \xrightarrow{\tau: (R, x, v)} v \)

- **(T-CAS1)**
  - \( \text{CAS}(x, v_1, v_2) \xrightarrow{\tau: (U, x, v_1, v_2)} 1 \)

- **(T-FAA)**
  - \( \text{FAA}(x, v) \xrightarrow{\tau: (U, x, v_0, v_0 + v)} v_0 \)

- **(T-CAS0)**
  - \( \text{CAS}(x, v_1, v_2) \xrightarrow{\tau: (R, x, v)} 0 \) if \( v \neq v_1 \)

- **(T-SF)**
  - \( \text{sfence} \xrightarrow{\tau: SF} 1 \)

- **(T-MF)**
  - \( \text{mfence} \xrightarrow{\tau: MF} 1 \)

- **(T-FL)**
  - \( \text{flush}(x) \xrightarrow{\tau: (F_L, x)} 1 \)

- **(T-FO)**
  - \( \text{flush}_{opt}(x) \xrightarrow{\tau: (F_0, x)} 1 \)
Thread transitions: $\text{COM} \xrightarrow{TID:LAB \cup \{\epsilon\}} \text{COM}$

$\text{LAB} \triangleq \{(R, x, v), (W, x, v), (U, x, v, v'), \text{MF}, \text{SF}, (F0, x), (FL, x) \mid x \in \text{Loc} \land v, v' \in \text{VAL}\}$

\[
\begin{align*}
C_1 & \xrightarrow{\tau:l} C_1' \quad \text{(T-Let1)} \\
\text{let } a := C_1 \text{ in } C_2 & \xrightarrow{\tau:l} \text{let } a := C_1' \text{ in } C_2 \\
\text{let } a := v \text{ in } C & \xrightarrow{\tau: \epsilon} C[v/a] \quad \text{(T-Let2)}
\end{align*}
\]
Px86 Program Transitions

Thread transitions: $\text{COM} \xrightarrow{T:\text{LAB} \cup \{\varepsilon\}} \text{COM}$

$\text{LAB} \triangleq \{(R, x, v), (W, x, v), (U, x, v, v'), \text{MF}, \text{SF}, (F0, x), (FL, x) \mid x \in \text{Loc} \land v, v' \in \text{Val}\}$

- $C_1 \xrightarrow{\tau:l} C'_1$ (T-Let1)
  
  $$\text{let } a := C_1 \text{ in } C_2 \xrightarrow{\tau:l} \text{let } a := C'_1 \text{ in } C_2$$

- $\quad$$

- $C \xrightarrow{\tau:l} C'$ (T-If1)

  $$\text{if } (C) \text{ then } C_1 \text{ else } C_2 \xrightarrow{\tau:l} \text{if } (C') \text{ then } C_1 \text{ else } C_2$$

  $$\begin{align*}
  v \neq 0 & \Rightarrow C = C_1 \\
  v = 0 & \Rightarrow C = C_2
  \end{align*}$$ (T-If2)

  $$\text{if } (v) \text{ then } C_1 \text{ else } C_2 \xrightarrow{\tau:e} C$$
Px86 Program Transitions

Thread transitions: COM \xrightarrow{T:\text{LAB} \cup \{\epsilon\}} \text{COM}

\text{LAB} \triangleq \{(R, x, v), (W, x, v), (U, x, v, v'), \text{MF}, \text{SF}, (F0, x), (FL, x) \mid x \in \text{Loc} \land v, v' \in \text{VAL}\}

\begin{align*}
C \xrightarrow{\tau:l} C' & \quad \text{(T-Let1)} \\
\text{let a:=C in C_2} & \xrightarrow{\tau:l} \text{let a:=C' in C_2} \\
\text{let a:=v in C} & \xrightarrow{\tau:\epsilon} C[v/a] \quad \text{(T-Let2)}
\end{align*}

\begin{align*}
C \xrightarrow{\tau:l} C' & \quad \text{(T-If1)} \\
\text{if (C) then C_1 else C_2} & \xrightarrow{\tau:l} \text{if (C') then C_1 else C_2} \\
\frac{v \neq 0}{v \neq 0} & \Rightarrow C=C_1 \quad \frac{v=0}{v=0} \Rightarrow C=C_2 \quad \text{(T-If2)} \\
\text{if (v) then C_1 else C_2} & \xrightarrow{\tau:\epsilon} C \\
\text{repeat C} & \xrightarrow{\tau:\epsilon} \text{if (C) then (repeat C) else 0} \quad \text{(T-Repeat)}
\end{align*}
Px86 Program Transitions

**Thread transitions:** $\text{TId}: \text{LAB} \uplus \{\epsilon\}$

$\text{COM} \xrightarrow{TId: \text{LAB} \uplus \{\epsilon\}} \text{COM}$

$\text{LAB} \triangleq \{(R, x, v), (W, x, v), (U, x, v, v'), MF, SF, (F0, x), (FL, x) \mid x \in \text{Loc} \land v, v' \in \text{VAL}\}$

**Program transitions:** $\text{PROG} \xrightarrow{TId: \text{LAB} \uplus \{\epsilon\}} \text{PROG}$

\[
\begin{align*}
P(\tau) & \xrightarrow{\tau: l} C \\
\text{P} & \xrightarrow{\tau: l} \text{P}[\tau \mapsto C] \\
\text{PROG} &
\end{align*}
\]
Px86 Storage System

Thread1

Buffer

Persistence Buffer

(Persistent) Memory

Thread2

Buffer
\[ M \in \text{Mem} \triangleq \text{Loc} \xrightarrow{\text{fin}} \text{Val} \]
\( M \in \text{Mem} \triangleq \text{Loc} \xrightarrow{\text{fin}} \text{Val} \)

\( B \in \text{BMap} \triangleq \text{TId} \xrightarrow{\text{fin}} \text{Buff} \)
buffer/unbuffer order: consistency model

→ execution reordering

\[ M \in \text{Mem} \triangleq \text{Loc} \overset{\text{fin}}{\rightarrow} \text{Val} \]

\[ B \in \text{BMap} \triangleq \text{TId} \overset{\text{fin}}{\rightarrow} \text{Buff} \]
buffer/unbuffer order: consistency model  
→ execution reordering

\[ M \in Mem \overset{\text{fin}}{\rightarrow} Val \]

\[ B \in BMap \overset{\text{fin}}{\rightarrow} \text{Buff} \]
buffer/unbuffer order: consistency model → execution reordering

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

(buffer/unbuffer order) execution reordering
→ delay writes/sfence/flushopt/flush execution in thread buffers

\[ M \in \text{Mem} \triangleq \text{Loc} \xrightarrow{\text{fin}} \text{Val} \]

\[ B \in \text{BMap} \triangleq \text{TId} \xrightarrow{\text{fin}} \text{Buff} \]

\[ b \in \text{Buff} \triangleq \text{SEQ} \left\{ (W, x, v), (FL, x), (FO, x), SF \mid x \in \text{Loc} \land v \in \text{Val} \right\} \]
### Px86 Storage System

**buffer/unbuffer order: consistency model**

→ **execution reordering**

#### Later in Program Order

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Read</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td>Write</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>sloc</td>
</tr>
<tr>
<td>C</td>
<td>RMW</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>D</td>
<td>mfence</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E</td>
<td>sfence</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td>flushopt</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>G</td>
<td>flush</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>sloc</td>
</tr>
</tbody>
</table>

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

- Reads reordered before writes/sfence/flushopt/flush → **delay** writes/sfence/flushopt/flush execution in thread buffers
- flushopt reordered w.r.t. writes/flushopt/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)

---

\[ M \in \text{Mem} \triangleq \text{Loc} \rightarrow \text{Val} \]

\[ B \in \text{BMap} \triangleq \text{TID} \rightarrow \text{Buff} \]

\[ b \in \text{Buff} \triangleq \text{SEQ} \left\{ (W, x, v), (FL, x), (FO, x), SF \mid x \in \text{Loc} \land v \in \text{Val} \right\} \]
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

\[ PB \in PBMAP \triangleq \text{Loc} \xrightarrow{\text{fin}} \text{PBuff} \]
\[ pb \in \text{PBuff} \triangleq \text{SEQ} \{ w(v) \mid v \in \text{VAL} \} \]
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**

\[
\begin{align*}
PB \in PBMAP & \triangleq \text{Loc} \xrightarrow{\text{fin}} \text{PBuff} \\
pb \in \text{PBuff} & \triangleq \text{SEQ} \left\langle \{w(v)\} \mid v \in \text{VAL} \right\rangle
\end{align*}
\]
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
- flushopt executed asynchronously → record & delay them in pbuff

buffer/unbuffer order: persistency model → persist reordering

\[ PB \in PBMAP \triangleq \text{Loc} \xrightarrow{\text{fin}} \text{PBuff} \]
\[ pb \in \text{PBuff} \triangleq \text{SEQ} \langle \{ w(v), f_0(\tau) \mid v \in \text{VAL} \land \tau \in \text{TId} \} \]
 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

- flush\_opt executed **asynchronously**
  - → record & delay them in pbuff

- flush\_opt + sfence/mfence/RMW = **persist sequence**
  - → ensure no flush\_opt in pbuff

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

\[
P_{B} \in \text{PBMAP} \triangleq \text{Loc} \xrightarrow{\text{fin}} \text{PBUFF}
\]

\[
p_{b} \in \text{PBUFF} \triangleq \text{SEQ} \langle \{w(v), f_{o}(\tau) \mid v \in \text{VAL} \land \tau \in \text{TId} \rangle
\]
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

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

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

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

\[\begin{align*}
PB & \in \text{PBMAP} \triangleq \text{Loc} \rightarrow \text{PBUFF} \\
pb & \in \text{PBUFF} \triangleq \text{SEQ} \langle \{w(v), fo(\tau) \mid v \in \text{VAL} \land \tau \in \text{TID} \} \rangle
\end{align*}\]

simplification due to Khyzha & Lahav [3]
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

- flush\textsubscript{opt} executed *asynchronously*
  - record & delay them in pbuff

- flush\textsubscript{opt} + sfence/mfence/RMW = *persist sequence*
  - ensure no flush\textsubscript{opt} in pbuff

Original model by Raad et al. [2]: one pbuff for all locations
Px86 Storage Transitions: Execution

<table>
<thead>
<tr>
<th>M ∈ Mem ∆ Loc → Val</th>
<th>PB ∈ PBMap ∆ Loc → PBuff</th>
<th>B ∈ BMap ∆ TId → Buff</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb ∈ PBuff ∆ SEQ {w(v), fo(τ)</td>
<td>v ∈ Val ∧ τ ∈ TId}</td>
<td>b ∈ Buff ∆ SEQ {(w, x, v), (FL, x), (FO, x), SF</td>
</tr>
</tbody>
</table>

**Storage transitions:** \(\text{Mem} × \text{PBMap} × \text{BMap} \xrightarrow{\text{TId:Lab∪\{ε\}}} \text{Mem} × \text{PBMap} × \text{BMap}\)
Px86 Storage Transitions: Execution

\[ M \in \text{Mem} \triangleq \text{Loc} \rightarrow \text{Val} \quad \text{PB} \in \text{PBM} \triangleq \text{Loc} \rightarrow \text{PBUF} \quad B \in \text{BMAP} \triangleq \text{TId} \rightarrow \text{BUFF} \]

\[ pb \in \text{PBUF} \triangleq \text{SEQ} \langle \{w(v), f_0(\tau) \mid v \in \text{Val} \land \tau \in \text{TId}\} \quad b \in \text{BUFF} \triangleq \text{SEQ} \langle \{(w, x, v), (\text{FL}, x), (\text{FO}, x), \text{SF} \mid x \in \text{Loc} \land v \in \text{Val}\} \]

\[ \text{Storage transitions: } \text{MEM} \times \text{PBM} \times \text{BMAP} \xrightarrow{T\text{D:LAB}\cup \{e\}} \text{MEM} \times \text{PBM} \times \text{BMAP} \]

\[ \frac{B(\tau) = b}{\frac{M, PB, B}{\tau: (w, x, v)} \rightarrow M, PB, B[\tau \mapsto b.(w, x, v)]} \quad (\text{M-WRITE}) \]
Px86 Storage Transitions: Execution

\[ M \in \text{Mem} \overset{\text{fin}}{\triangleq} \text{Loc} \rightarrow \text{VAL} \quad \text{PB} \in \text{PBMap} \overset{\text{fin}}{\triangleq} \text{Loc} \rightarrow \text{PBuff} \quad \text{B} \in \text{BMap} \overset{\text{fin}}{\triangleq} \text{TId} \rightarrow \text{BUFF} \]

\[ \text{pb} \in \text{PBuff} \overset{\text{fin}}{\triangleq} \text{SEQ} \left\{ w(v), f_o(\tau) \mid v \in \text{VAL} \land \tau \in \text{TId} \right\} \quad \text{b} \in \text{BUFF} \overset{\text{fin}}{\triangleq} \text{SEQ} \left\{ (w, x, v), (\text{FL}, x), (\text{FO}, x), \text{SF} \mid x \in \text{Loc} \land v \in \text{VAL} \right\} \]

Storage transitions: \( \text{Mem} \times \text{PBMap} \times \text{BMap} \xrightarrow{\text{TId}\cup\{\epsilon\}} \text{Mem} \times \text{PBMap} \times \text{BMap} \)

\[
\begin{align*}
\text{B}(\tau) = b \\
\text{M, PB, B} \xrightarrow{\tau: (W, x, v)} \text{M, PB, B}[\tau \mapsto b.(W, x, v)]
\end{align*}
\]

\[
\text{(M-WRITE)}
\]

\[
\begin{align*}
\text{B}(\tau) = b \\
\text{M, PB, B} \xrightarrow{\tau: (\text{FL}, x)} \text{M, PB, B}[\tau \mapsto b.(\text{FL}, x)]
\end{align*}
\]

\[
\text{(M-FL)}
\]

\[
\begin{align*}
\text{B}(\tau) = b \\
\text{M, PB, B} \xrightarrow{\tau: (\text{FO}, x)} \text{M, PB, B}[\tau \mapsto b.(\text{FO}, x)]
\end{align*}
\]

\[
\text{(M-FO)}
\]

\[
\begin{align*}
\text{B}(\tau) = b \\
\text{M, PB, B} \xrightarrow{\tau: \text{SF}} \text{M, PB, B}[\tau \mapsto b.\text{SF}]
\end{align*}
\]

\[
\text{(M-SF)}
\]
Px86 Storage Transitions: Execution

\[
\begin{align*}
M \in \text{MEM} & \triangleq \text{Loc} \to \text{Val} \\
\text{PB} \in \text{PBMAP} & \triangleq \text{Loc} \to \text{PBUFF} \\
B \in \text{BMAP} & \triangleq \text{TID} \to \text{BUFF} \\
pb \in \text{PBUFF} & \triangleq \text{SEQ} \langle \{ w(v), f_0(\tau) \mid v \in \text{VAL} \land \tau \in \text{TID} \} \rangle \\
b \in \text{BUFF} & \triangleq \text{SEQ} \langle \{ (W, x, v), (FL, x), (FO, x), SF \mid x \in \text{LOC} \land v \in \text{VAL} \} \rangle
\end{align*}
\]

Storage transitions: \( \text{MEM} \times \text{PBMAP} \times \text{BMAP} \xrightarrow{\text{TID}: \text{LAB} \cup \{ \epsilon \}} \text{MEM} \times \text{PBMAP} \times \text{BMAP} \)

\[
\begin{align*}
\frac{B(\tau) = b}{M, PB, B} & \xrightarrow{\tau: (W, x, v)} M, PB, B[\tau \mapsto b.(W, x, v)] \quad \text{(M-WRITE)} \\
\frac{B(\tau) = b}{M, PB, B} & \xrightarrow{\tau: (FL, x)} M, PB, B[\tau \mapsto b.(FL, x)] \quad \text{(M-FL)} \\
\frac{B(\tau) = b}{M, PB, B} & \xrightarrow{\tau: (FO, x)} M, PB, B[\tau \mapsto b.(FO, x)] \quad \text{(M-FO)} \\
\frac{B(\tau) = b}{M, PB, B} & \xrightarrow{\tau: (R, x, v)} M, PB, B \quad \text{(M-READ)} \\
\frac{\text{rd}(M, PB, b, x) = v}{M, PB, B} & \xrightarrow{\tau: (R, x, v)} M, PB, B
\end{align*}
\]

\[
\text{rd}(M, PB, b, x) \triangleq \begin{cases} 
    v & \text{if } \exists S_1, S_2. \ b = S_1.(W, x, v).S_2 \land \forall v'. (W, x, v') \notin S_2 \\
    v & \text{else if } \exists S_1, S_2. \ PB(x) = S_1.w(v).S_2 \land \forall v'. w(v') \notin S_2 \\
    M(x) & \text{otherwise}
\end{cases}
\]
**Px86 Storage Transitions: Execution**

\[ M \in \text{Mem} \triangleq \text{Loc} \rightarrow \text{Val} \]

\[ PB \in \text{PBMap} \triangleq \text{Loc} \rightarrow \text{PBuffer} \]

\[ B \in \text{BMap} \triangleq \text{Tid} \rightarrow \text{Buffer} \]

\[ pB \in \text{PBuffer} \triangleq \text{Seq} \left\{ \{w(v), fo(\tau) \mid v \in \text{Val} \land \tau \in \text{Tid}\} \right\} \]

\[ b \in \text{Buffer} \triangleq \text{Seq} \left\{ \{(W, x, v), (FL, x), (FO, x), SF \mid x \in \text{Loc} \land v \in \text{Val}\} \right\} \]

\[ \text{Tid} : \text{Lab} \cup \{\epsilon\} \]

**Storage transitions:** \[ \text{Mem} \times \text{PBMap} \times \text{BMap} \xrightarrow{\text{Tid}: \text{Lab} \cup \{\epsilon\}} \text{Mem} \times \text{PBMap} \times \text{BMap} \]

\[ B(\tau) = b \quad \text{(M-WRITE)} \]

\[ M, PB, B \xrightarrow{\tau: (W, x, v)} M, PB, B[\tau \mapsto b.(W, x, v)] \]

\[ B(\tau) = b \quad \text{(M-FO)} \]

\[ M, PB, B \xrightarrow{\tau: (FO, x)} M, PB, B[\tau \mapsto b.(FO, x)] \]

\[ B(\tau) = b \quad \text{(M-READ)} \]

\[ M, PB, B \xrightarrow{\tau: (R, x, v)} M, PB, B \]

\[ B(\tau) = \varepsilon \quad \forall x. \text{fo}(\tau) \notin PB(x) \quad \text{(M-MF)} \]

\[ M, PB, B \xrightarrow{\tau: \text{MF}} M, PB, B \]

\[ \text{flush}_{\text{opt}} + \text{mfence/RMW} = \text{persist sequence} \]

\[ \rightarrow \text{ensure no } \text{flush}_{\text{opt}} \text{ in pbuff} \]

\[ \text{rd}(M, PB, b, x) \triangleq \begin{cases} v & \text{if } \exists S_1, S_2. b=S_1.(W, x, v).S_2 \land \forall v'. (W, x, v') \notin S_2 \\ v & \text{else if } \exists S_1, S_2. PB(x)=S_1.w(v).S_2 \land \forall v'. w(v') \notin S_2 \\ M(x) & \text{otherwise} \end{cases} \]
**Px86 Storage Transitions: Execution**

\[
\begin{align*}
M \in \text{Mem} & \triangleq \text{Loc} \rightarrow \text{Val} \\
PB \in \text{PBMAP} & \triangleq \text{Loc} \rightarrow \text{PBUFF} \\
B \in \text{BMAP} & \triangleq \text{TID} \rightarrow \text{BUFF} \\
pb \in \text{PBUFF} & \triangleq \text{SEQ} \left\{ w(v), fo(\tau) \mid v \in \text{Val} \land \tau \in \text{TID} \right\} \\
b \in \text{BUFF} & \triangleq \text{SEQ} \left\{ (w, x, v), (\text{FL}, x), (\text{FO}, x), \text{SF} \mid x \in \text{Loc} \land v \in \text{Val} \right\} \\
\text{Storage transitions: } & \text{Mem} \times \text{PBMAP} \times \text{BMAP} \\
\end{align*}
\]

**Case 1:**

\[
\begin{align*}
\frac{\text{B}(\tau) = b}{M, PB, B} \xrightarrow{\tau: (W, x, v)} M, PB, B[\tau \mapsto b.(W, x, v)] \\
\end{align*}
\]

**Case 2:**

\[
\begin{align*}
\frac{\text{B}(\tau) = b}{M, PB, B} \xrightarrow{\tau: (F0, x)} M, PB, B[\tau \mapsto b.(F0, x)] \\
\end{align*}
\]

**Case 3:**

\[
\begin{align*}
\frac{\text{B}(\tau) = b}{M, PB, B} \xrightarrow{\tau: (F0, x)} M, PB, B \\
\end{align*}
\]

**Case 4:**

\[
\begin{align*}
\frac{\text{B}(\tau) = \epsilon}{M, PB, B} \xrightarrow{\tau: \text{MF}} M, PB, B \\
\end{align*}
\]

**Case 5:**

\[
\begin{align*}
\frac{\text{rd}(M, PB, b, x) = v}{M, PB, B} \xrightarrow{\tau: (R, x, v)} M, PB, B \\
\end{align*}
\]

**Case 6:**

\[
\begin{align*}
\frac{\text{B}(\tau) = b}{M, PB, B} \xrightarrow{\tau: \text{MF}} M, PB, B \\
\end{align*}
\]

**Case 7:**

\[
\begin{align*}
\frac{\text{rd}(M, PB, \epsilon, x) = v_r}{M, PB, B} \xrightarrow{\tau: (U, x, v_r, v_w)} M, PB, B[\tau \mapsto pb.w(v_w)], B \\
\end{align*}
\]

\[
\text{flush}_\text{opt} + \text{mfence/RMW} = \text{persist sequence} \\
\rightarrow \text{ensure no flush}_\text{opt} \text{ in pbuff}
\]
Px86 Storage Transitions: Delayed Propagation

### Transition Rules

<table>
<thead>
<tr>
<th>Transition</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M \in \text{Mem} \triangleq \text{Loc} \rightarrow \text{Val} )</td>
<td>( PB \in \text{PMap} \triangleq \text{Loc} \rightarrow \text{PBuff} )</td>
</tr>
<tr>
<td>( \text{pb} \in \text{PBuff} \triangleq \text{SEQ} \left( { w(v), f_0(\tau) \mid v \in \text{Val} \land \tau \in \text{TId} } \right) )</td>
<td>( b \in \text{Buff} \triangleq \text{SEQ} \left( { (w, x, v), (\text{FL}, x), (\text{FO}, x), \text{SF} \mid x \in \text{Loc} \land v \in \text{Val} } \right) )</td>
</tr>
</tbody>
</table>

### Storage Transitions

\[
\text{Mem} \times \text{PMap} \times \text{BMap} \xrightarrow{\text{TId:Lab} \cup \{ e \}} \text{Mem} \times \text{PMap} \times \text{BMap}
\]
writes/flush/sfence ordered w.r.t. one another → their buffer/unbuffer orders agree (FIFO)

Persisting writes may be delayed → record & delay writes in pbuf
Px86 Storage Transitions: Delayed Propagation

writes/flush/sfence ordered w.r.t. one another $\rightarrow$ their buffer/unbuffer orders agree (FIFO)
Persisting writes may be delayed $\rightarrow$ record & delay writes in pbuff
flush executed synchronously $\rightarrow$ no need to delay/record them in pbuff
Para-86 Storage Transitions: Delayed Propagation

\[ M \in \text{MEM} \triangleq \text{Loc} \rightarrow \text{Val} \quad PB \in \text{PBMAP} \triangleq \text{Loc} \rightarrow \text{PBUFF} \quad B \in \text{BMAP} \triangleq \text{TID} \rightarrow \text{BUFF} \]

\[ \text{pb} \in \text{PBUFF} \triangleq \text{SEQ} \langle \{ w(v), \text{fo}(\tau) \mid v \in \text{Val} \land \tau \in \text{TID} \} \quad b \in \text{BUFF} \triangleq \text{SEQ} \langle \{ (W, x, v), (FL, x), (FO, x), SF \mid x \in \text{Loc} \land v \in \text{Val} \} \]

**Storage transitions:** \( \text{MEM} \times \text{PBMAP} \times \text{BMAP} \xrightarrow{\text{TID}: \text{LAB} \cup \{\epsilon\}} \text{MEM} \times \text{PBMAP} \times \text{BMAP} \)

\[
\begin{align*}
B(\tau) = (W, x, v).b' & \quad PB(x) = \text{pb} \\
M, PB, B \xrightarrow{\tau; \epsilon} M, PB[x \mapsto \text{pb}.w(v)], B[\tau \mapsto b'] & \quad (\text{M-PROP W})
\end{align*}
\]

\[
\begin{align*}
B(\tau) = (FL, x).b' & \quad PB(x) = \epsilon \\
M, PB, B \xrightarrow{\tau; \epsilon} M, PB, B[\tau \mapsto b'] & \quad (\text{M-PROP FL})
\end{align*}
\]

\[
\begin{align*}
B(\tau) = \text{SF}.b & \quad \forall x. \text{fo}(\tau) \notin PB(x) \\
M, PB, B \xrightarrow{\tau; \epsilon} M, PB, B[\tau \mapsto b] & \quad (\text{M-PROP SF})
\end{align*}
\]

- writes/flush/sfence ordered w.r.t. one another → their buffer/unbuffer orders agree (FIFO)
- Persisting writes may be **delayed** → record & delay writes in pbuf
- flush executed **synchronously** → no need to delay/record them in pbuf
- flush\text{opt} + sfence/mfence/RMW = **persist sequence** → ensure no flush\text{opt} in pbuf
### Px86 Storage Transitions: Delayed Propagation

**Storage transitions:** \( \text{MEM} \times \text{PBMAP} \times \text{BMAP} \xrightarrow{\text{TID}:\text{LAB} \cup \{\epsilon\}} \text{MEM} \times \text{PBMAP} \times \text{BMAP} \)

1. **(M-PROPFL)**
   
   \[
   \frac{B(\tau) = (F, x).b', PB(x) = \epsilon}{M, PB, B \xrightarrow{\tau: \epsilon} M, PB, B[\tau \mapsto b']} \]

2. **(M-PROPW)**
   
   \[
   \frac{B(\tau) = (W, x, v).b', PB(x) = \text{pb}}{M, PB, B \xrightarrow{\tau: \epsilon} M, PB[x \mapsto \text{pb}.w(v)], B[\tau \mapsto b']} \]

3. **(M-PROPSF)**
   
   \[
   \frac{B(\tau) = \text{SF}.b, \forall x. \text{fo}(\tau) \notin PB(x)}{M, PB, B \xrightarrow{\tau: \epsilon} M, PB, B[\tau \mapsto b]} \]

4. **(M-PROPFO)**
   
   \[
   \frac{B(\tau) = b_1.(F, 0, x).b_2, PB(x) = \text{pb}, \text{SF}, (W, x, -), (F, 0, x), (F, L, x) \notin b_1}{M, PB, B \xrightarrow{\tau: \epsilon} M, PB[x \mapsto \text{pb}.\text{fo}(\tau)], B[\tau \mapsto b_1.b_2]} \]

- **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
- **flush_{opt} + sfence/mfence/RMW = persist sequence** → ensure no flush_{opt} in pbuff
- **flush_{opt} executed asynchronously** → record & delay them in pbuff
- **flush_{opt} reordered w.r.t. writes/flush_{opt}/flush on diff. locations** → their buffer/unbuffer orders can disagree
Px86 Storage Transitions: Delayed Persists

\[ M \in \text{Mem} \triangleq \text{Loc} \rightarrow \text{Val} \quad \text{PB} \in \text{PBM} \triangleq \text{Loc} \rightarrow \text{PBuff} \quad B \in \text{BMap} \triangleq \text{TId} \rightarrow \text{Buff} \]

\[ \text{pb} \in \text{PBuff} \triangleq \text{Seq} \langle \{ w(v), f_o(\tau) \mid v \in \text{Val} \land \tau \in \text{TId} \} \rangle \quad b \in \text{Buff} \triangleq \text{Seq} \langle \{ (W, x, v), (FL, x), (F0, x), SF \mid x \in \text{Loc} \land v \in \text{Val} \} \rangle \]

**Storage transitions:** \( \text{Mem} \times \text{PBM} \times \text{BMap} \xrightarrow{\text{TId}: \text{Lab} \cup \{ \epsilon \}} \text{Mem} \times \text{PBM} \times \text{BMap} \)

\[
\begin{align*}
\text{PB}(x) &= w(v).\text{pb} \\
\text{M, PB, B} &\xrightarrow{\tau: \epsilon} \text{M}[x \mapsto v], \text{PB}[x \mapsto \text{pb}], B \\
\text{(M-PERSISTW)}
\end{align*}
\]

\[
\begin{align*}
\text{PB}(x) &= f_o(\tau).\text{pb} \\
\text{M, PB, B} &\xrightarrow{\tau: \epsilon} \text{M}, \text{PB}[x \mapsto \text{pb}], B \\
\text{(M-PERSISTFO)}
\end{align*}
\]
Parsy86 Operational Semantics

Program transitions: \( \text{Prog} \xrightarrow{T_{\text{ID:LAB}} \cup \{\epsilon\}} \text{Prog} \)

Storage transitions: \( \text{Mem} \times \text{PBMAP} \times \text{BMAP} \xrightarrow{T_{\text{ID:LAB}} \cup \{\epsilon\}} \text{Mem} \times \text{PBMAP} \times \text{BMAP} \)
Px86 Operational Semantics

Program transitions: $\text{PROG} \xrightarrow{TID:LABU\{\epsilon\}} \text{PROG}$

Storage transitions: $\text{MEM} \times \text{PBMAP} \times \text{BMap} \xrightarrow{TID:LABU\{\epsilon\}} \text{MEM} \times \text{PBMAP} \times \text{BMap}$

Operational semantics: $\text{PROG} \times \text{MEM} \times \text{PBMAP} \times \text{BMap} \Rightarrow \text{PROG} \times \text{MEM} \times \text{PBMAP} \times \text{BMap}$
Px86 Operational Semantics

Program transitions: $\text{PROG} \xrightarrow{Tid:LabU\{\epsilon\}} \text{PROG}$

Storage transitions: $\text{MEM} \times \text{PBMAP} \times \text{BMAP} \xrightarrow{Tid:LabU\{\epsilon\}} \text{MEM} \times \text{PBMAP} \times \text{BMAP}$

Operational semantics: $\text{PROG} \times \text{MEM} \times \text{PBMAP} \times \text{BMAP} \Rightarrow \text{PROG} \times \text{MEM} \times \text{PBMAP} \times \text{BMAP}$

$\text{P} \xrightarrow{\tau:\epsilon} \text{P'}$  \hspace{1cm} \text{(SILENTP)}

$\text{P}, \text{M}, \text{PB}, \text{B} \Rightarrow \text{P'}, \text{M}, \text{PB}, \text{B}$
Px86 Operational Semantics

Program transitions: $\text{PROG} \xrightarrow{T\text{Id:LabU}\{\epsilon\}} \text{PROG}$

Storage transitions: $\text{MEM} \times \text{PBMAP} \times \text{BMap} \xrightarrow{T\text{Id:LabU}\{\epsilon\}} \text{MEM} \times \text{PBMAP} \times \text{BMap}$

Operational semantics: $\text{PROG} \times \text{MEM} \times \text{PBMAP} \times \text{BMap} \Rightarrow \text{PROG} \times \text{MEM} \times \text{PBMAP} \times \text{BMap}$

\[
\frac{P \xrightarrow{\tau : \epsilon} P'}{P, M, PB, B \Rightarrow P', M, PB, B} \quad (\text{SILENTP})
\]

\[
\frac{M, PB, B \xrightarrow{\tau : \epsilon} M', PB', B'}{P, M, PB, B \Rightarrow P, M', PB', B'} \quad (\text{SILENTS})
\]
Px86 Operational Semantics

Program transitions: $\mathbb{P}_{\text{prog}} \xrightarrow{\text{Tid} : \{ \epsilon \}} \mathbb{P}_{\text{prog}}$

Storage transitions: $\mathbb{M}_{\text{mem}} \times \mathbb{P}_{\text{bmap}} \times \mathbb{B}_{\text{bmap}} \xrightarrow{\text{Tid} : \{ \epsilon \}} \mathbb{M}_{\text{mem}} \times \mathbb{P}_{\text{bmap}} \times \mathbb{B}_{\text{bmap}}$

Operational semantics: $\mathbb{P}_{\text{prog}} \times \mathbb{M}_{\text{mem}} \times \mathbb{P}_{\text{bmap}} \times \mathbb{B}_{\text{bmap}} \Rightarrow \mathbb{P}_{\text{prog}} \times \mathbb{M}_{\text{mem}} \times \mathbb{P}_{\text{bmap}} \times \mathbb{B}_{\text{bmap}}$

- **Silent Program Transition**
  \[
  \frac{P \xrightarrow{\tau : \epsilon} P'}{P, M, PB, B \Rightarrow P', M, PB, B} \quad \text{(SilentP)}
  \]

- **Silent Storage Transition**
  \[
  \frac{M, PB, B \xrightarrow{\tau : \epsilon} M', PB', B'}{P, M, PB, B \Rightarrow P, M', PB', B'} \quad \text{(SilentS)}
  \]

- **Step Transition**
  \[
  \frac{P \xrightarrow{\tau : l} P' \quad M, PB, B \xrightarrow{\tau : l} M', PB', B'}{P, M, PB, B \Rightarrow P', M', PB', B'} \quad \text{(Step)}
  \]
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 \((n, \tau, l)\)
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 \((n, \tau, l)\)
  
  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: $< E, \text{po}, \text{rf}, \text{mo}>$
- $E$ is the set of **events** (graph nodes), including initialisation writes
  - each event if of the form $(n, \tau, l)$
  - $E$ is the set of events (graph nodes), including initialisation writes

```
1 x:=1;
2 a:=y //0
3 y:=1;
4 b:=x //0
```

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*)

```
1 x := 1;
2 a := y //0
3 y := 1;
4 b := x //0
```

[init]

```
1 W(x, 1)
2 R(y, 0)
3 W(y, 1)
4 R(x, 0)
```
Declarative Consistency Semantics (w/o Persistency)

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

```
1 x:=1;
2 a:=y //0
3 y:=1;
4 b:=x //0
```

```
[init]
1 W(x, 1)
2 R(y, 0)
3 W(y, 1)
4 R(x, 0)
```
Declarative Consistency Semantics (w/o Persistency)

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

```
Store Buffer

1. x := 1;
2. a := y // 0
3. y := 1;
4. b := x // 0

[init]
1. W(x, 1)
2. R(y, 0)
3. W(y, 1)
4. R(x, 0)
```
Declarative Consistency Semantics (w/o Persistency)

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

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

**Store Buffer**

```
[init]
1 W(x, 1)  3 W(y, 1)
2 R(y, 0)  4 R(x, 0)
```

```
1 x := 1;  3 y := 1;
2 a := y //0  4 b := x //0
```
Declarative Consistency Semantics (w/o Persistency)

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

---

**Example**

- **Store Buffer**
  - \(x := 1;\)
  - \(a := y;\)
  - \(y := 1;\)
  - \(b := x;\)

**Execution Graph**

- \(W(x, 1)\)
- \(R(y, 0)\)
- \(W(y, 1)\)
- \(R(x, 0)\)
- \([\text{init}]\)
Declarative Consistency Semantics (w/o Persistency)

- Represent program behaviours as a set of **consistent executions (graphs)**
- An execution graph is a tuple: $< E, \text{po}, \text{rf}, \text{mo} >$
- $E$ 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, $\text{rb} = \text{rf}^{-1} \cdot \text{mo}$, is the reads-before relation

```
[init]

1  x := 1;  3  y := 1;
2  a := y //0  4  b := x //0
```

```
1  W(x, 1)
2  R(y, 0)
3  W(y, 1)
4  R(x, 0)
```
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
What is a consistent execution?

- Depends on the (concurrency) memory model
- Intel-x86 consistency:

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

\(\text{Ri} : \text{internal (same-thread) subset of R} \)
\(\text{Re} : \text{external (diff.-thread) subset of R: R \setminus Ri} \)
What is a **consistent** execution?

- Depends on the (concurrency) memory model
- Intel-x86 consistency:

\[
\text{rf}_i \cup \text{mo}_i \cup \text{rb}_i \subseteq \text{po} \quad \text{(Internal)}
\]

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

\[\text{R}_i : \text{internal (same-thread) subset of } \text{R} \]
\[\text{R}_{e} : \text{external (diff.-thread) subset of } \text{R} : \text{R} \setminus \text{R}_i\]
Consistent Executions (w/o Persistency)

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

\[
\text{rfi} \cup \text{moi} \cup \text{rbi} \subseteq \text{po} \\
\text{irreflexive}(\text{ob}) \quad \text{ob} = (\text{ppo} \cup \text{rfe} \cup \text{moe} \cup \text{rbe})^+ \\
\]

preserved program order: sloc or ✓ in table

\[
\begin{array}{cccccccc}
\text{A} & \text{Read} & ✓ & ✓ & ✓ & ✓ & ✓ & ✓ & ✓ \\
\text{B} & \text{Write} & X & ✓ & ✓ & ✓ & ✓ & ✓ & ✓ \\
\text{C} & \text{RMW} & ✓ & ✓ & ✓ & ✓ & ✓ & ✓ & ✓ \\
\text{D} & \text{mfence} & ✓ & ✓ & ✓ & ✓ & ✓ & ✓ & ✓ \\
\text{E} & \text{sfence} & X & ✓ & ✓ & ✓ & ✓ & ✓ & ✓ \\
\text{F} & \text{flush}_{\text{opt}} & X & X & ✓ & ✓ & ✓ & X & ✓ \\
\text{G} & \text{flush} & X & ✓ & ✓ & ✓ & ✓ & ✓ & ✓ \\
\end{array}
\]

\(\text{Ri} : \) internal (same-thread) subset of \(\text{R}\)  \\
\(\text{Re} : \) external (diff.-thread) subset of \(\text{R} : \text{R} \setminus \text{Ri}\)
What is a consistent execution?
- Depends on the (concurrency) memory model
- Intel-x86 consistency:

\[ \text{rfi} \cup \text{moi} \cup \text{rbi} \subseteq \text{po} \]

Irreflexive(\(ob\)) \(ob=(\text{ppo} \cup \text{rfe} \cup \text{moe} \cup \text{rbe})^+\)

Consistent Executions (w/o Persistency)

\[
\begin{array}{ll}
1 & x := 1; \\
2 & a := y // 0 \\
3 & y := 1; \\
4 & b := x // 0
\end{array}
\]

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

\[
\begin{array}{llllllll}
1 & \text{W}(x, 1) & 2 & \text{R}(y, 0) & 3 & \text{W}(y, 1) & 4 & \text{R}(x, 0)
\end{array}
\]
Consistent Executions (w/o Persistency)

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

\[
rfi \cup moi \cup rbi \subseteq po \\
\text{irreflexive}(ob) \quad \text{ob=}((\text{ppo}\cup rfe \cup moe \cup rbe)^+) \\
\]

preserved program order: sloc or ✓ in table

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 \textit{can reach} NVM — model-specific for Intel-x86: \( D = W \cup U \cup FL \cup FO \)

\[
\begin{align*}
1 & \quad x:=1; \\
2 & \quad clflushopt x; \\
3 & \quad sfence; \\
4 & \quad y:=1
\end{align*}
\]

\[
\begin{align*}
1 & \quad W(x, 1) \\
2 & \quad FO(x) \\
3 & \quad SF \\
4 & \quad W(y, 1)
\end{align*}
\]

: durable events (in \( D \))
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. $\text{init} \subseteq P$

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

```
1 W(x, 1)
2 FO(x)
3 SF
4 W(y, 1)
```

durable events (in $D$)
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. \(\text{init} \subseteq P\)

\[
\begin{align*}
1 & \quad x:=1; & \quad 1 & \quad W(x, 1) \\
2 & \quad \text{clflushopt } x; & \quad 2 & \quad \text{FO}(x) \\
3 & \quad \text{sfence}; & \quad 3 & \quad \text{SF} \\
4 & \quad y:=1 & \quad 4 & \quad W(y, 1)
\end{align*}
\]

: persisted event (in \(P\)
Declarative Consistency & Persistency Semantics

- Represent program behaviours as a set of consistent \& **persistent** executions
- An execution graph is a tuple: \( < E, \text{po}, \text{rf}, \text{mo}, P, \text{nvo} > \)
  - 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. \( \text{init} \subseteq P \)
  - \( \text{nvo} \subseteq D \times D \) is the **non-volatile-order**: a strict (partial) order on \( D \) that is downward-closed on \( P \):
    - if \( (e, e') \in \text{nvo} \) and \( e' \in P \), then \( e \in P \)

1  \( x := 1; \)
2  \( \text{clflushopt} x; \)
3  \( \text{sfence}; \)
4  \( y := 1 \)

1 \( W(x, 1) \)
2 \( \text{FO}(x) \)
3 \( \text{SF} \)
4 \( W(y, 1) \)

**durable events** (in \( D \))

1: persisted event (in \( P \))
Valid (Consistent & Persistent) Executions

Intel-x86 consistency:
\[
rfi \cup moi \cup rbi \subseteq po
\]
Irreflexive \((ob)\) \quad \text{ob}=((ppo \cup rfe \cup moe \cup rbe)^+)

Preserved program order (sloc or \(\checkmark\) in table)

<table>
<thead>
<tr>
<th>Earlier in Program Order</th>
<th>Later in Program Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Read</td>
</tr>
<tr>
<td>B</td>
<td>Write</td>
</tr>
<tr>
<td>C</td>
<td>RMW</td>
</tr>
<tr>
<td>D</td>
<td>mfence</td>
</tr>
<tr>
<td>E</td>
<td>sfence</td>
</tr>
<tr>
<td>F</td>
<td>flush(_{opt})</td>
</tr>
<tr>
<td>G</td>
<td>flush</td>
</tr>
</tbody>
</table>
Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:
  \[ rfi \cup moi \cup rb \subseteq po \]
  irreflexive(ob) \[ ob = (ppo \cup rfe \cup moe \cup rbe)^+ \]
  preserved program order (sloc or \( \checkmark \) in table)

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Read</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td>Write</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>sloc</td>
</tr>
<tr>
<td>C</td>
<td>RMW</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>D</td>
<td>mfence</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E</td>
<td>sfence</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td>flushopt</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>sloc</td>
</tr>
<tr>
<td>G</td>
<td>flush</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>sloc</td>
</tr>
</tbody>
</table>

1. \( x := 1; \)
2. \( \text{clflushopt } x; \)
3. \( \text{sfence}; \)
4. \( y := 1 \)

\( i \quad : \) persisted event (in \( P \))
Valid (Consistent & Persistent) Executions

- **Intel-x86 consistency:**
  \[
  rf_i \cup mo_i \cup rb_i \subseteq po
  \]
  
  irreflexive\(ob\) \(\Rightarrow \) \(ob=(ppo \cup rfe \cup moe \cup rbe)^+\)

  preserved program order (sloc or ✓ in table)

- **Intel-x86 persistency** (simplified):
  \[
  FL \cup dom(FO) \xrightarrow{po} SF \cup MF \cup U \subseteq P
  \]

---

1. \(x:=1;\)
2. \(\text{clflushopt } x;\)
3. \(\text{sfence};\)
4. \(y:=1\)

- persisted event (in \(P\))

---

Later in Program Order

<table>
<thead>
<tr>
<th></th>
<th>Read</th>
<th>Write</th>
<th>RMW</th>
<th>mfence</th>
<th>sfence</th>
<th>flushopt</th>
<th>flush</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>G</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Earlier in Program Order

- durable events (in \(D\))

1. \(W(x, 1)\)
2. \(FO(x)\)
3. \(SF\)
4. \(W(y, 1)\)

- persisted event (in \(P\))
Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:
  \[rf_i \cup moi \cup rb_i \subseteq po\]

- Intel-x86 irreflexive consistency:
  \[ob = (ppo \cup rfe \cup moe \cup rbe)^+\]

- Intel-x86 persistence (simplified):
  \[FL \cup \text{dom}(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P\]

- Preserved program order (sloc or ✓ in table)

**Example: 1**
- 1: \(x := 1\)
- 2: \(\text{clflushopt } x;\)
- 3: \(\text{sfence;}\)
- 4: \(y := 1\)

**Durable events (in D):**
- \(W(x, 1)\)
- \(FO(x)\)
- \(SF\)
- \(W(y, 1)\)

**Persisted event (in P):**
- \(W(x, 1)\)
Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:
  \[ \text{rfi} \cup \text{moi} \cup \text{rbi} \subseteq \text{po} \]
  irreflexive(ob) \( \text{ob} = (\text{ppo} \cup \text{rfe} \cup \text{moe} \cup \text{rbe})^+ \)
  preserved program order (sloc or \( \checkmark \) in table)

- Intel-x86 **persistence** (simplified):
  \[ \text{FL} \cup \text{dom(FO} \xrightarrow{\text{po}} \text{SF} \cup \text{MF} \cup \text{U}) \subseteq P \]
  strong persists
  persist sequences

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

1. \( W(x, 1) \)
2. \( \text{FO}(x) \)
3. \( \text{SF} \)
4. \( W(y, 1) \)

: durable events (in \( D \))

: persisted event (in \( P \))

<table>
<thead>
<tr>
<th></th>
<th>Read</th>
<th>Write</th>
<th>RMW</th>
<th>mfence</th>
<th>sfence</th>
<th>flushopt</th>
<th>flush</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>G</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Later in Program Order

Earlier in Program Order
Valid (Consistent & Persistent) Executions

**Intel-x86 consistency:**

\[
\text{rfi} \cup \text{moi} \cup \text{rbi} \subseteq \text{po}
\]

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

preserved program order (sloc or √ in table)

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

\[
\text{FL} \cup \text{dom}\left(\text{FO} \xrightarrow{\text{po}} \text{SF} \cup \text{MF} \cup \text{U}\right) \subseteq P
\]

\[
D \quad \text{ob} \cap \text{sloc} \quad D \subseteq \text{nvo}
\]

---

Later in Program Order

<table>
<thead>
<tr>
<th></th>
<th>Read</th>
<th>Write</th>
<th>RMW</th>
<th>mfence</th>
<th>sfence</th>
<th>flushopt</th>
<th>flush</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>G</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Earlier in Program Order

\[
1 \quad \text{W(x, 1)}
\]

\[
2 \quad \text{FO(x)}
\]

\[
3 \quad \text{SF}
\]

\[
4 \quad \text{W(y, 1)}
\]

\[
\text{i} \quad \text{durable events (in D)}
\]

\[
\text{i} \quad \text{persisted event (in P)}
\]
Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:
  \[ \text{rfi} \cup \text{moi} \cup \text{rbi} \subseteq \text{po} \]
  irreflexive(ob) \[ \text{ob} = (\text{ppo} \cup \text{rfe} \cup \text{moe} \cup \text{rbe})^+ \]
  preserved program order (sloc or √ in table)

- Intel-x86 **persistence** (simplified):
  \[ \text{FL} \cup \text{dom(FO)} \stackrel{\text{po}}{\longrightarrow} \text{SF} \cup \text{MF} \cup \text{U} \subseteq P \]
  \[ D \text{ ob } \cap \text{sloc} \quad D \subseteq \text{nvo} \]

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

Durable events (in D)

Persisted event (in P)
Valid (Consistent & Persistent) Executions

- **Intel-x86 consistency:**
  \[ rfi \cup moi \cup rbi \subseteq po \]
  
  **Irreflexive** (ob) \[ ob = (ppo \cup rfe \cup moe \cup rbe)^+ \]
  
  Preserved program order (sloc or ✓ in table)

- **Intel-x86 persistency** (simplified):
  \[ FL \cup \text{dom}(FO \xrightarrow{po} SF \cup MF \cup U) \subseteq P \]
  
  \[ D \cap \text{sloc} \subseteq nvo \]
  
  \[ (FO \cup FL) \xrightarrow{ob} D \subseteq nvo \]

**Example Code**

1. \[ x:=1; \]
2. \[ clflushopt x; \]
3. \[ sfence; \]
4. \[ y:=1 \]

**Diagram**

- Node 1: \[ W(x, 1) \]
  - \[ ppo \subseteq ob \cap \text{sloc} \]
  - Durable events (in D)

- Node 2: \[ FO(x) \]
  - \[ SF \]

- Node 4: \[ W(y, 1) \]

**Symbol**

- \[ ✓ \]: Persisted event (in P)
**Valid (Consistent & Persistent) Executions**

- **Intel-x86 consistency:**
  
  $rfi \cup moi \cup rbi \subseteq po$

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

  preserved program order (sloc or $\checkmark$ in table)

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

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

  $D \quad ob \cap sloc \quad D \subseteq nvo$

  $(FO \cup FL) \quad ob \quad D \subseteq nvo$

1. $x := 1$
2. `clflushopt x`
3. `sfence`
4. $y := 1$

- $i$: persisted event (in $P$)

- Later in Program Order

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read</td>
<td>Write</td>
<td>RMW</td>
<td>mfence</td>
<td>sfence</td>
<td>flush</td>
<td>flushopt</td>
</tr>
<tr>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- Durable events (in $D$)

- Earlier in Program Order

- $W(x, 1)$

- $ppo \subseteq ob \cap sloc$

- `FO(x)`

- `SF`

- $W(y, 1)$
Valid (Consistent & Persistent) Executions

Intel-x86 consistency:
\[ \text{rfi} \cup \text{moi} \cup \text{rbi} \subseteq \text{po} \]
irreflexive(ob) \[ \text{ob} = (\text{ppo} \cup \text{rfe} \cup \text{moe} \cup \text{rbe})^+ \]
preserved program order (sloc or \checkmark in table)

Intel-x86 **persistency** (simplified):
\[ \text{FL} \cup \text{dom(FO}^{\text{po}} \rightarrow \text{SF} \cup \text{MF} \cup \text{U}) \subseteq P \]
\[ D \text{ ob} \cap \text{sloc} D \subseteq \text{nvo} \]
\[ (\text{FO} \cup \text{FL}) \text{ ob} D \subseteq \text{nvo} \]

1. \( x := 1; \)
2. \( \text{clflushopt } x; \)
3. \( \text{sfence}; \)
4. \( y := 1 \)

\( i \): persisted event (in \( P \))
Valid (Consistent & Persistent) Executions

- **Intel-x86 consistency:**
  \[
  \text{rfi} \cup \text{moi} \cup \text{rbi} \subseteq \text{po} \\
  \text{irreflexive}(\text{ob}) \quad \text{ob} = (\text{ppo} \cup \text{rfe} \cup \text{moe} \cup \text{rbe})^+ \\
  \text{preserved program order (sloc or \checkmark in table)}
  \]

- **Intel-x86 ** **persistency** **(simplified):**
  \[
  \text{FL} \cup \text{dom}(\text{FO} \xrightarrow{\text{po}} \text{SF} \cup \text{MF} \cup \text{U}) \subseteq P \\
  D \text{ob} \cap \text{sloc} \subseteq \text{nvo} \\
  (\text{FO} \cup \text{FL}) \xrightarrow{\text{ob}} D \subseteq \text{nvo}
  \]

\begin{itemize}
  \item 1 \textbf{x:=1;}
  \item 2 \textbf{clflushopt x;}
  \item 3 \textbf{sfence;}
  \item 4 \textbf{y:=1}
\end{itemize}

\textbf{i}: persisted event (in } P \text{)

Later in Program Order

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Write</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RMW</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>mfence</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>sfence</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>flushopt</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>flush</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Earlier in Program Order

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]

\[\]
Valid (Consistent & Persistent) Executions

- Intel-x86 consistency:
  \[ \text{rfi} \cup \text{moi} \cup \text{rbi} \subseteq \text{po} \]
- Irreflexive \( (\text{ob}) \):
  \[ \text{ob}=((\text{ppo} \cup \text{rfe} \cup \text{moe} \cup \text{rbe})^+) \]

- Preserved program order (sloc or ✓ in table)

- Intel-x86 **persistence** (simplified):
  \[ \text{FL} \cup \text{dom}(\text{FO} \xrightarrow{\text{po}} \text{SF} \cup \text{MF} \cup \text{U}) \subseteq P \]
  \[ D \cap \text{sloc} \subseteq \text{nvo} \]
  \[ (\text{FO} \cup \text{FL}) \cap \text{ob} \subseteq \text{nvo} \]

- **Persisted event** (in \( P \))

- **Durable events** (in \( D \))

Later in Program Order

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>G</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- **Write**
- **RMW**
- **mfence**
- **sfence**
- **flush**
- **flushopt**
- **sloc**

- **x:=1;**
- **clflushopt x;**
- **sfence;**
- **y:=1**

- **Persisted event** (in \( P \))
Valid (Consistent & Persistent) Executions

Intel-x86 consistency:

rfi ∪ moi ∪ rbi ⊆ po

irreflexive(ob)

ob = ((ppo ∪ rfe ∪ moe) ∪ rbe)'

preserved program order

(sloc or ✓ in table)

Intel-x86 persistence (simplified):

FL ∪ dom(FO po SF ∪ MF ∪ U) ⊆ P

D ob ∩ sloc D ⊆ nvo

(FO ∪ FL) ob D ⊆ nvo

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

: persisted event (in P)

Later in Program Order

<table>
<thead>
<tr>
<th></th>
<th>Read</th>
<th>Write</th>
<th>RMW</th>
<th>mfence</th>
<th>sfence</th>
<th>flushopt</th>
<th>flush</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ sloc</td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ x sloc</td>
<td>✓</td>
</tr>
<tr>
<td>G</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ sloc</td>
<td>✓</td>
</tr>
</tbody>
</table>

Durable events (in D)

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: \( \text{DC CVAP x} \) — analogous to \( \text{clflushopt x} \)
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 \( \text{ob} \) relation

- **ARMv8 Persistency**
  - Weak explicit persists: \texttt{DC CVAP x} — analogous to \texttt{clflushopt x}
  - Persist sequences: \texttt{DC CVAP x}; \texttt{DSB SY} — \texttt{DSB SY} is a strong fence (analogous to \texttt{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 \( \text{ob} \) relation

**ARMv8 Persistency**
- Weak explicit persists: \( \text{DC CVAP} \ x \) — analogous to \( \text{clflushopt} \ x \)
- Persist sequences: \( \text{DC CVAP} \ x; \text{DSB SY} \) — \( \text{DSB SY} \) is a strong fence (analogous to \( \text{mfence} \))
- 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: \texttt{DC\ CVAP\ x} — analogous to \texttt{clflushopt\ x}
- Persist sequences: \texttt{DC\ CVAP\ x}; \texttt{DSB\ SY} — \texttt{DSB\ SY} is a strong fence (analogous to \texttt{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)
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

Azalea Raad, John Wickerson, Gil Neiger, Viktor Vafeiadis

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

B. ARMv8 Persistency 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