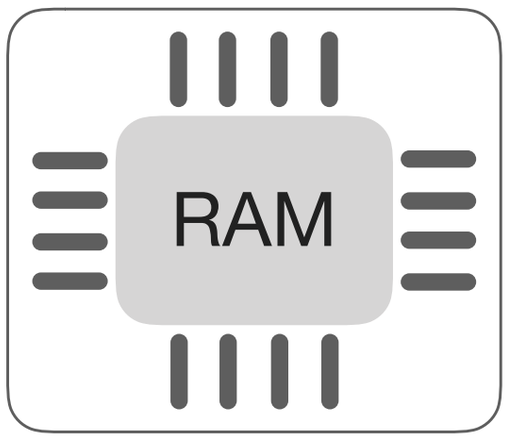
# Persistent Owicki-Gries Reasoning (POG)

**Azalea Raad**[1,2] Ori Lahav[3] Viktor Vafeiadis[1]

[1] Max Planck Institute for Software Systems (MPI-SWS)
[2] Imperial College London
[3] Tel Aviv University

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

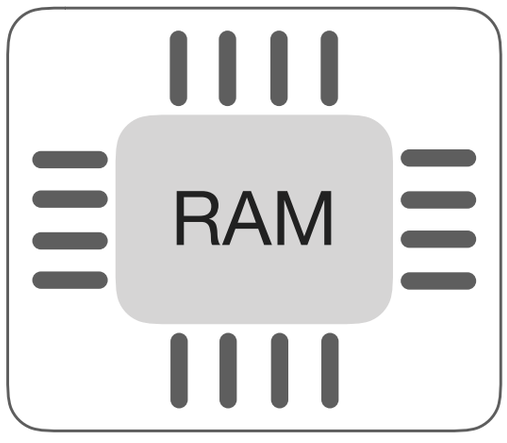
# Computer Storage




# Computer Storage

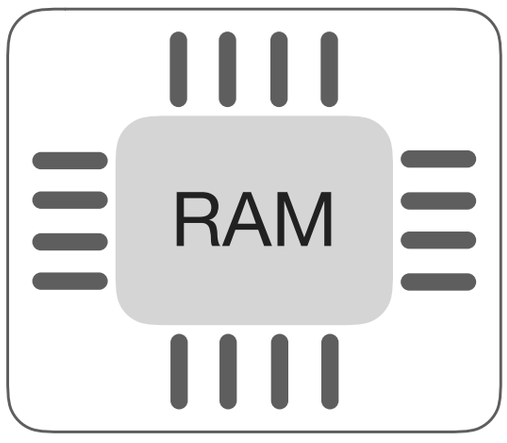
✓ *fast*

✗ *volatile*




# Computer Storage

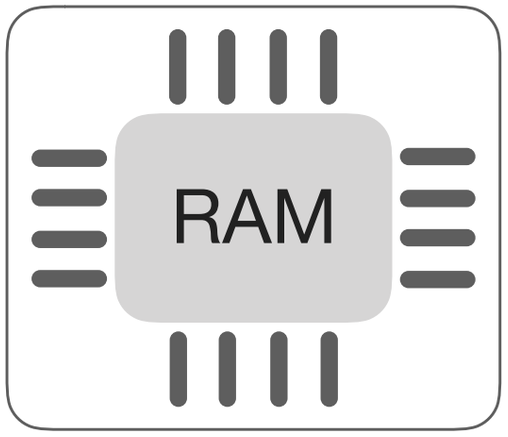
✓ *fast*
✗ *volatile*



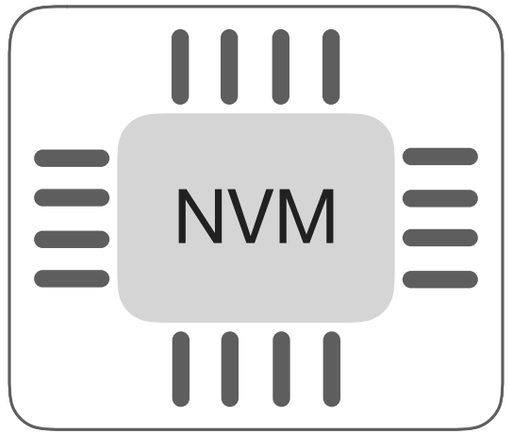


✗ *slow*
✓ *persistent*

# 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)

# Formal Persistency (Low-Level) ***Semantics***

Intel® Architecture Reference Manual

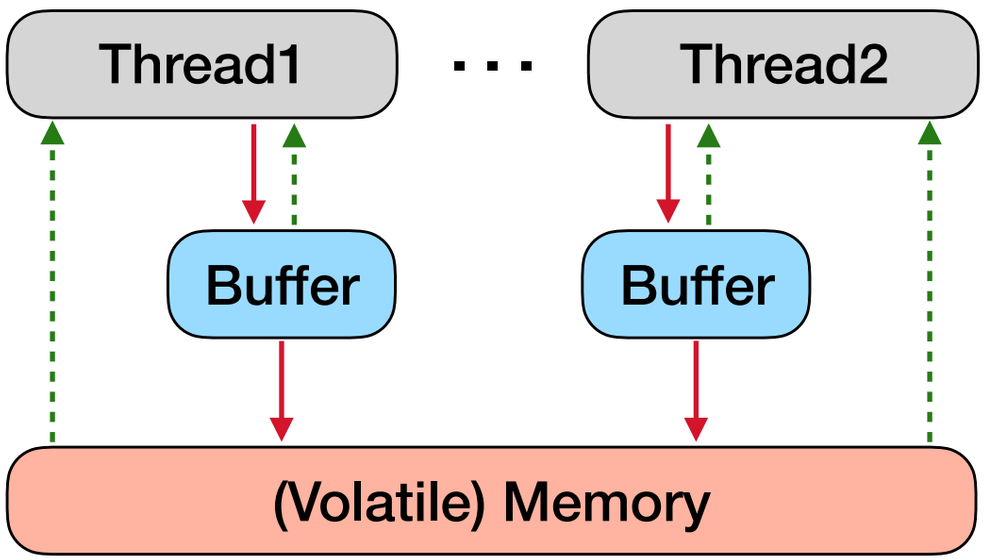

⇓

The ***Px86*** Model

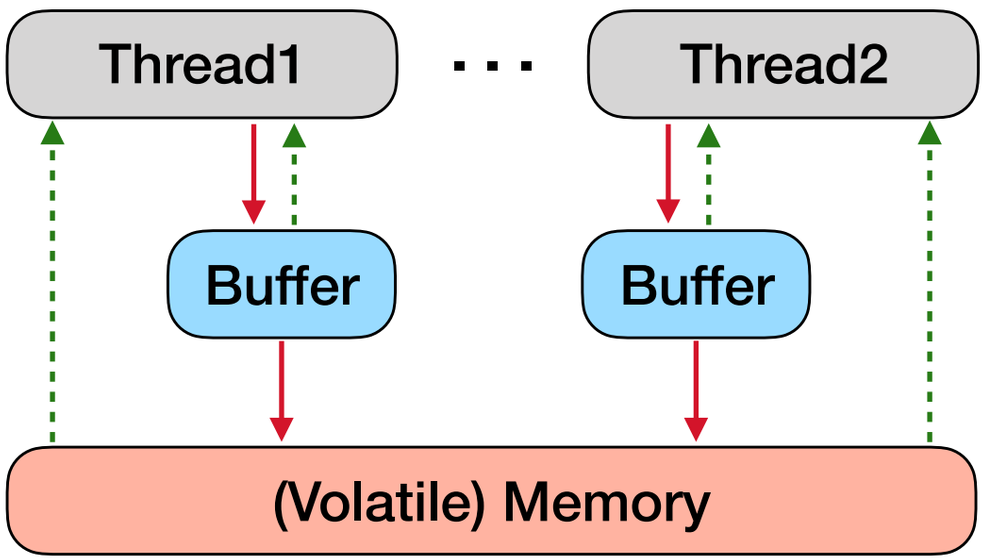
[Raad et al., POPL'20]

# x86: (Volatile) Concurrent Hardware Model (TSO)

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



# x86: (Volatile) Concurrent Hardware Model (TSO)

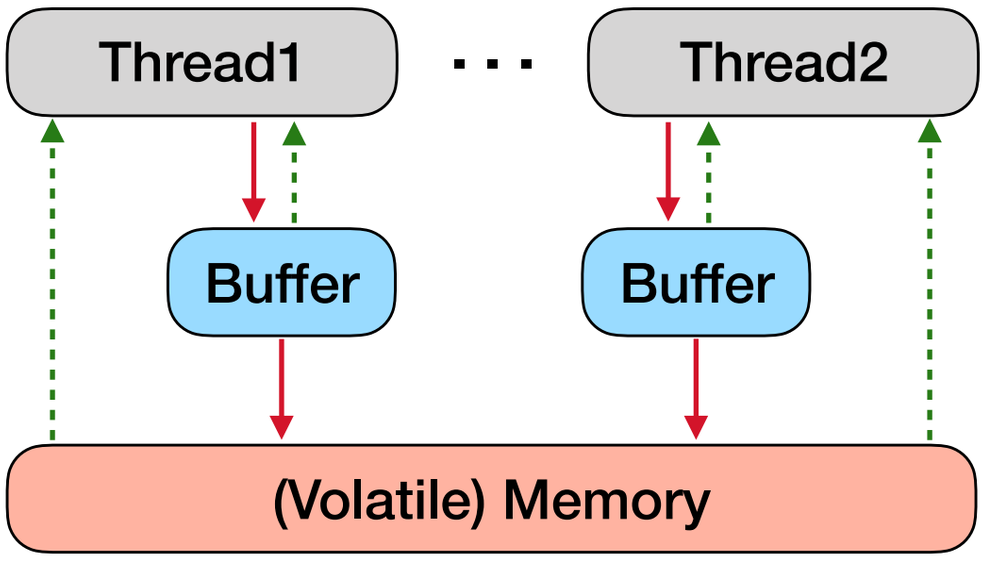



`x:=1` : adds `x:=1` to buffer

unbuffer* : buffer to memory (in FIFO order)

* at non-deterministic times

# x86: (Volatile) Concurrent Hardware Model (TSO)



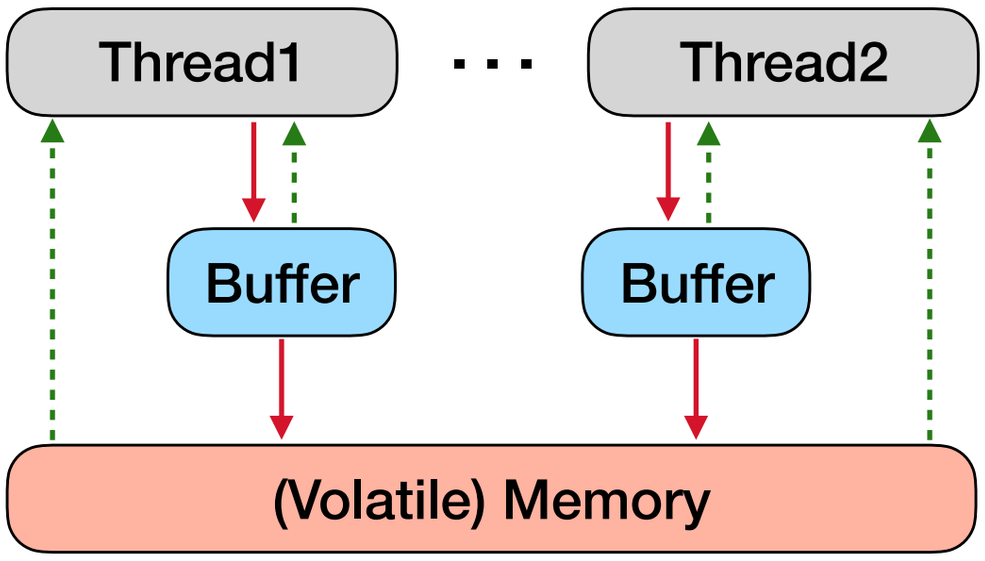

`x:=1` : adds `x:=1` to buffer

unbuffer* : buffer to memory (in FIFO order)

`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

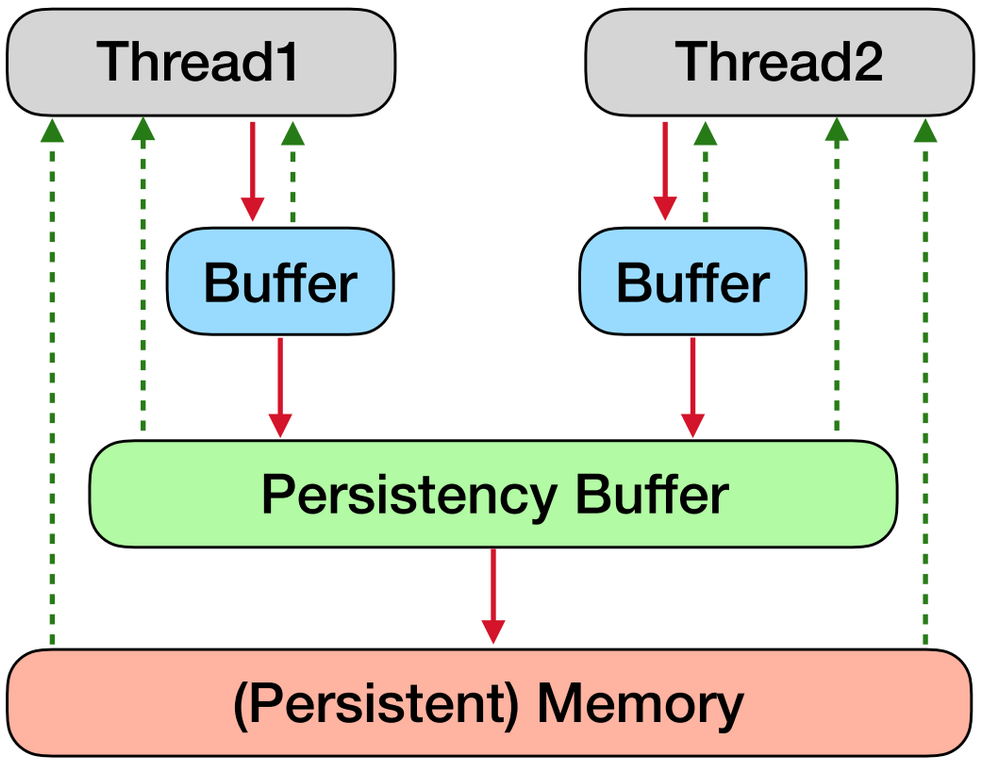
unbuffer* : buffer to memory (in FIFO order)

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

buffer and memory lost

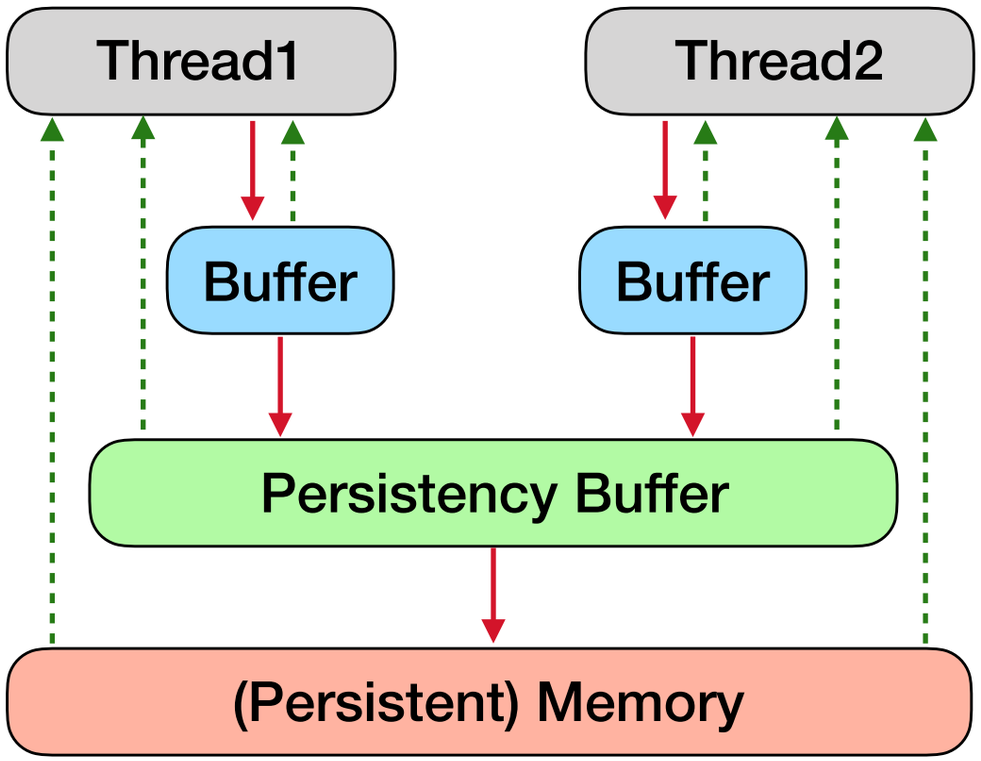
* at non-deterministic times

# Px86: (Persistent) Concurrent Hardware Model




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

# Px86: (Persistent) Concurrent Hardware Model

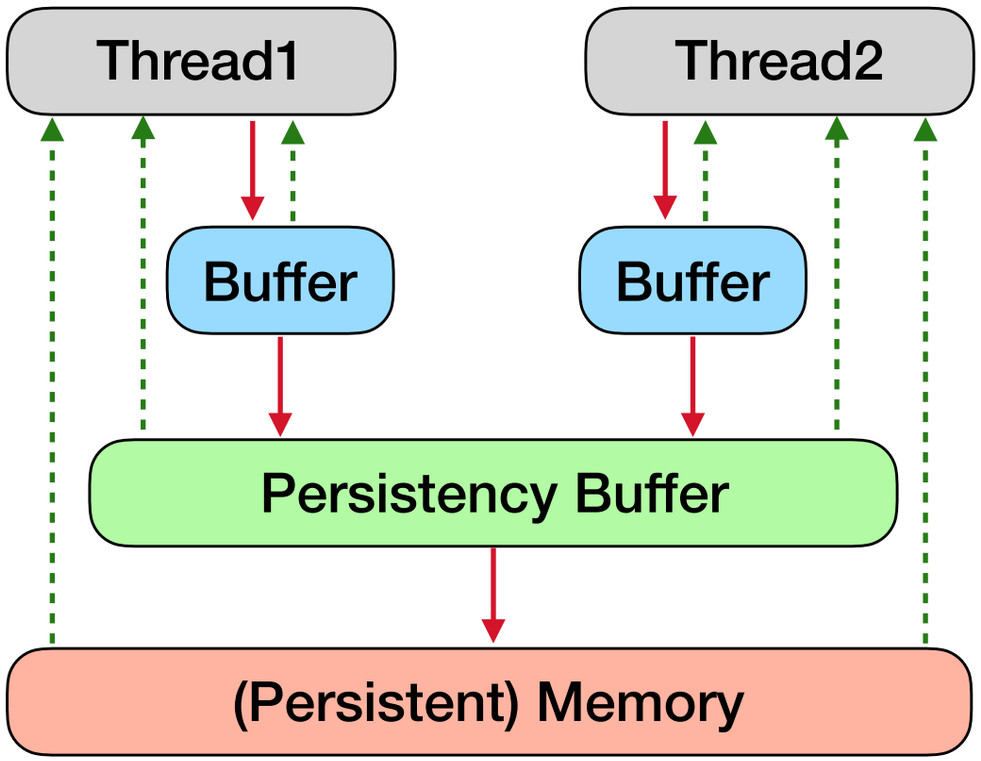



`x:=1` : adds `x:=1` to buffer

unbuffer* : buffer to pbuffer (in FIFO order)

* at non-deterministic times

# Px86: (Persistent) Concurrent Hardware Model

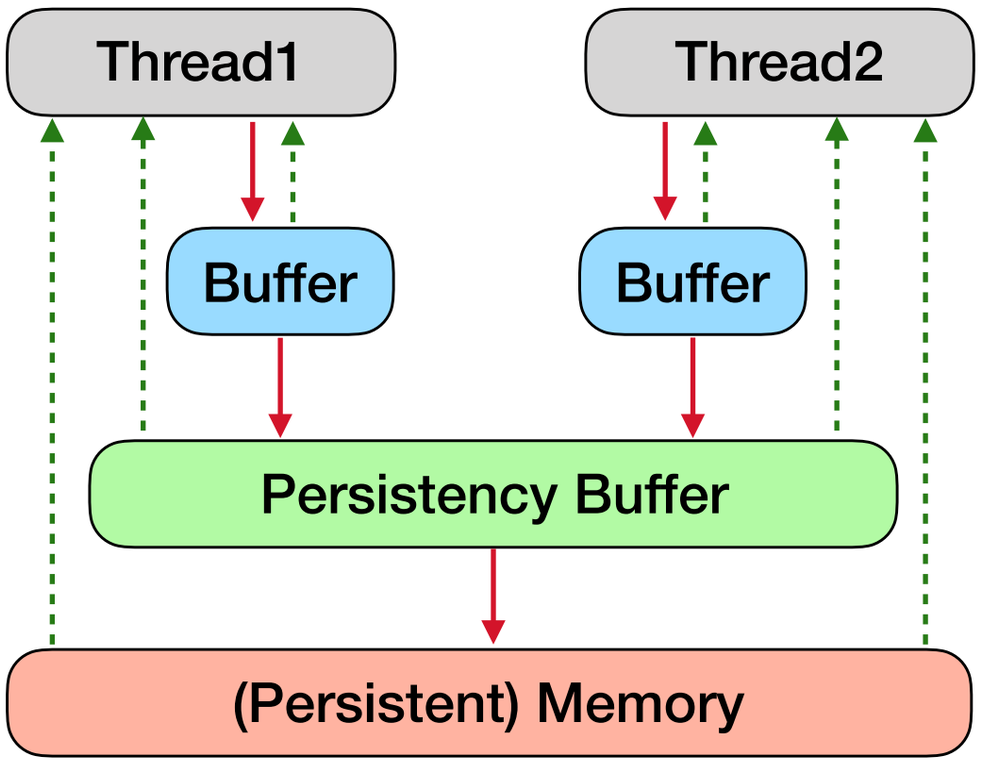



`x:=1` : adds `x:=1` to buffer

unbuffer* : buffer to pbuffer (in FIFO order)
unbuffer* : pbuffer to memory (in **FIFO-per-loc** order)

* at non-deterministic times

# Px86: (Persistent) Concurrent Hardware Model



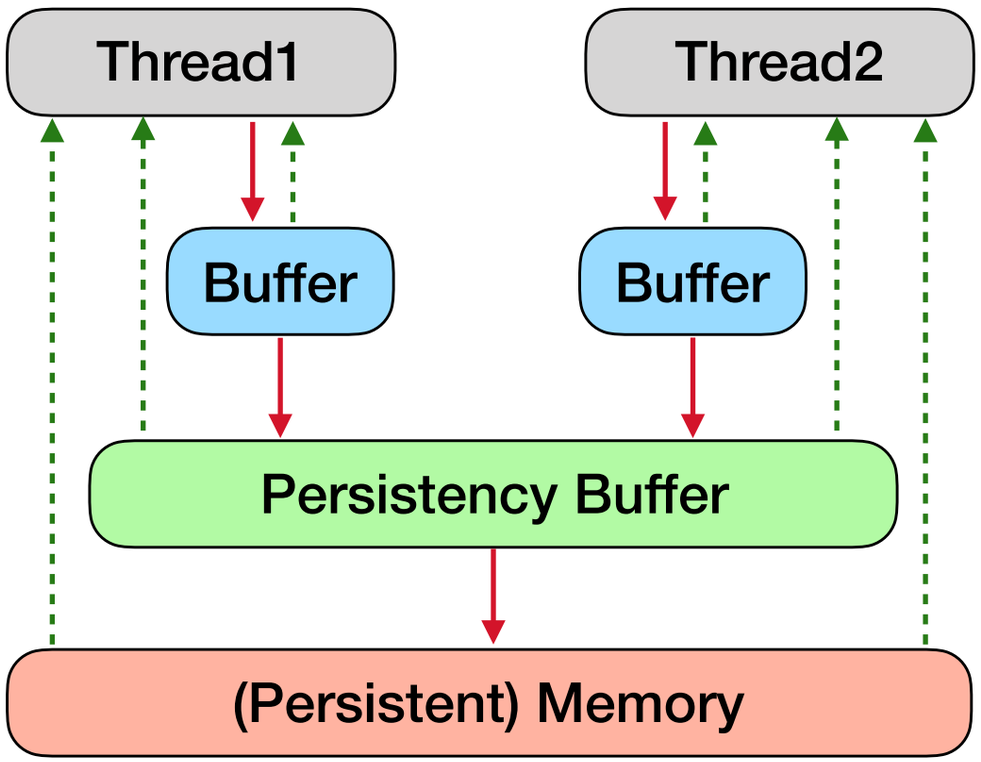

`x:=1` : adds `x:=1` to buffer

unbuffer* : buffer to pbuffer (in FIFO order)
unbuffer* : pbuffer to memory (in **FIFO-per-loc** order)

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

* at non-deterministic times

# Px86: (Persistent) Concurrent Hardware Model




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

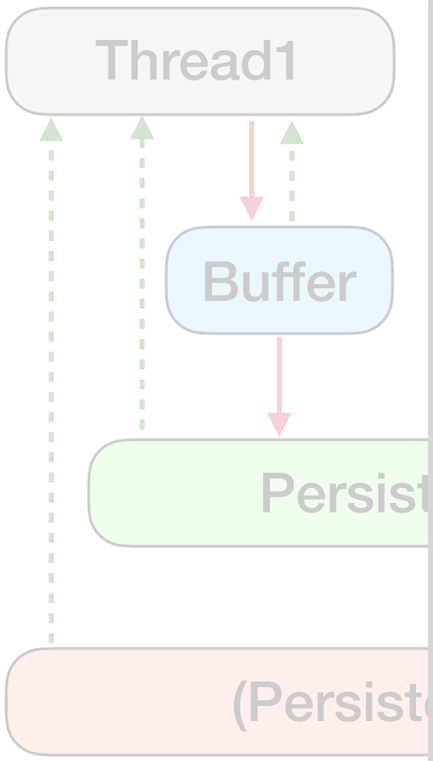
unbuffer* : buffer to pbuffer (in FIFO order)
unbuffer* : pbuffer to memory (in **FIFO-per-loc** order)

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

buffer and pbuffer lost

* at non-deterministic times

# Px86: (Persistent) Concurrent Hardware Model




## ***Problem***

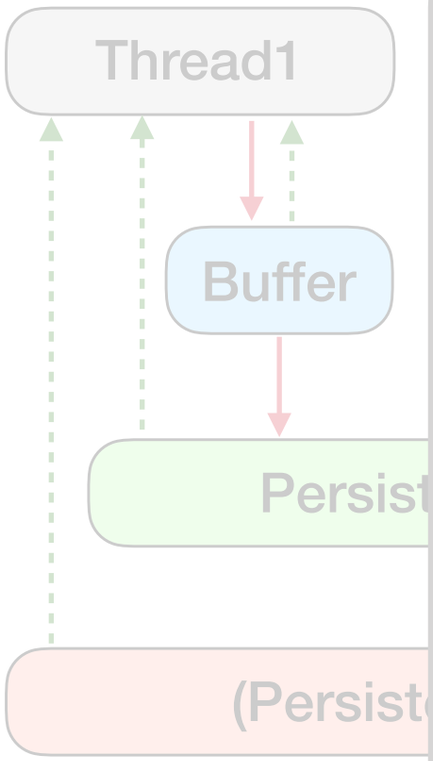
- **low-level**
- reasoning over program **executions**
- difficult to verify high-level invariants

er)
**per-loc** order)

est entry
ds latest entry

* at non-deterministic times

# Px86: (Persistent) Concurrent Hardware Model




er)
**per-loc** order)

est entry
ds latest entry

## ***Problem***

- **low-level**
- reasoning over program **executions**
- difficult to verify high-level invariants

## ***Solution***

- **high-level** reasoning via **program logics**
- reasoning over program **syntax**
- simpler to verify high-level invariants

* at non-deterministic times

# Towards a Persistent Program Logic

# Towards a Persistent Program Logic

- State of the art: ***no program logic for persistency***

# Towards a Persistent Program Logic

- State of the art: ***no program logic for persistency***
- Two possible avenues:
  - ❖ ***high-level (program logic) extension***
    - ➡ Take a program logic that is sound for x86 (TSO), e.g. OGRA
    - ➡ Extend it with persistency support for Px86

# Towards a Persistent Program Logic

- State of the art: ***no program logic for persistency***
- Two possible avenues:
  - ***high-level (program logic) extension***
    - ➡ Take a program logic that is sound for x86 (TSO), e.g. OGRA
    - ➡ Extend it with persistency support for Px86
  - ***low-level (semantic) extension***
    - ➡ Encode Px86 into x86 (TSO)
    - ➡ Use a program logic that is sound for TSO (e.g. OGRA)

# Towards a Persistent Program Logic

- State of the art: ***no program logic for persistency***
- Two possible avenues:
  - ❖ ***high-level (program logic) extension***
    - ➡ Take a program logic that is sound for x86 (TSO), e.g. OGRA
    - ➡ Extend it with persistency support for Px86
  - ❖ ***low-level (semantic) extension***
    - ➡ Encode Px86 into x86 (TSO)
    - ➡ Use a program logic that is sound for TSO (e.g. OGRA)

**Our Approach**

# Contributions

# Contributions

- ***lx86***: instrumented x86 semantics
  - ➡ reduces Px86 to x86 (TSO): ***removes pbuffer***
  - ➡ ***translation*** from Px86 to lx86
  - ➡ Several Challenges

# Contributions

- ***Ix86***: instrumented x86 semantics
  - ➡ reduces Px86 to x86 (TSO): ***removes pbuffer***
  - ➡ ***translation*** from Px86 to Ix86
  - ➡ Several Challenges

- ***POG***: the ***first*** program logic for persistency
  - ➡ built over Ix86
  - ➡ several ***examples*** of persistent reasoning

# Contributions

- ***Ix86***: instrumented x86 semantics
  - ➡ reduces Px86 to x86 (TSO): ***removes pbuffer***
  - ➡ ***translation*** from Px86 to Ix86
  - ➡ Several Challenges

← **This talk**

- ***POG***: the ***first*** program logic for persistency
  - ➡ built over Ix86
  - ➡ several ***examples*** of persistent reasoning

# Challenge #1: ***Weak Persistency***

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

# Challenge #1: ***Weak Persistency***

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

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

# Challenge #1: ***Weak Persistency***

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

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

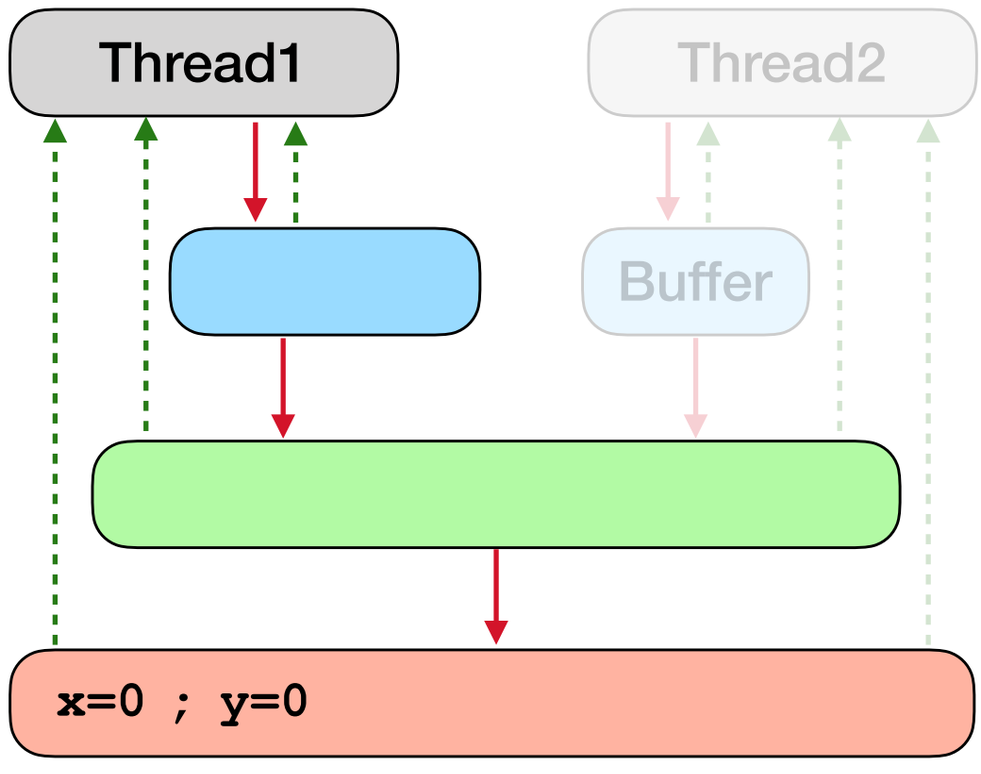
!! Writes may persist (be unbuffered from pbuffer) ***out of order***

# Challenge #1: ***Weak Persistency***

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

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

!! Writes may persist (be unbuffered from pbuffer) ***out of order***

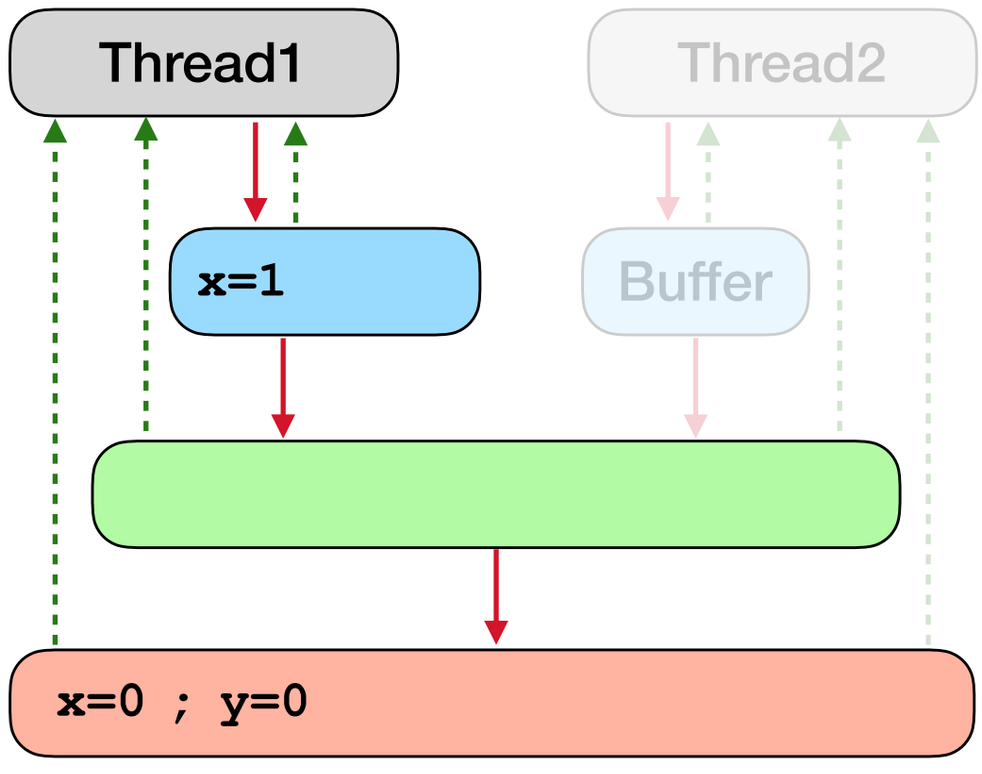


# Challenge #1: ***Weak Persistency***

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

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

!! Writes may persist (be unbuffered from pbuffer) ***out of order***

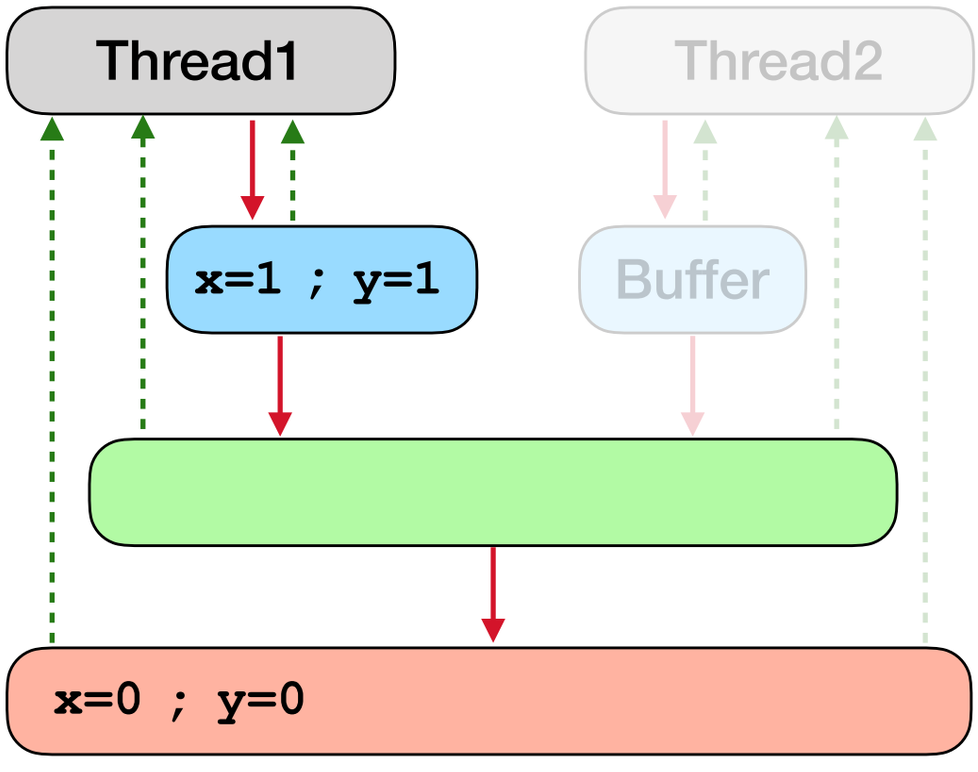


# Challenge #1: ***Weak Persistency***

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

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

!! Writes may persist (be unbuffered from pbuffer) ***out of order***

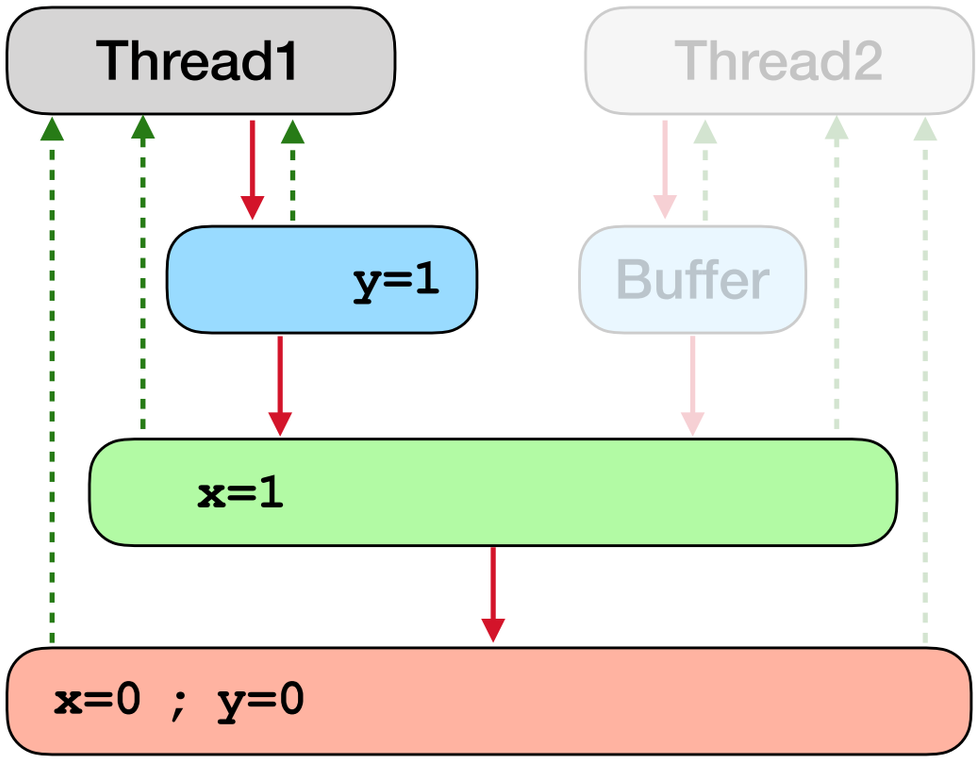


# Challenge #1: ***Weak Persistency***

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

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

!! Writes may persist (be unbuffered from pbuffer) ***out of order***

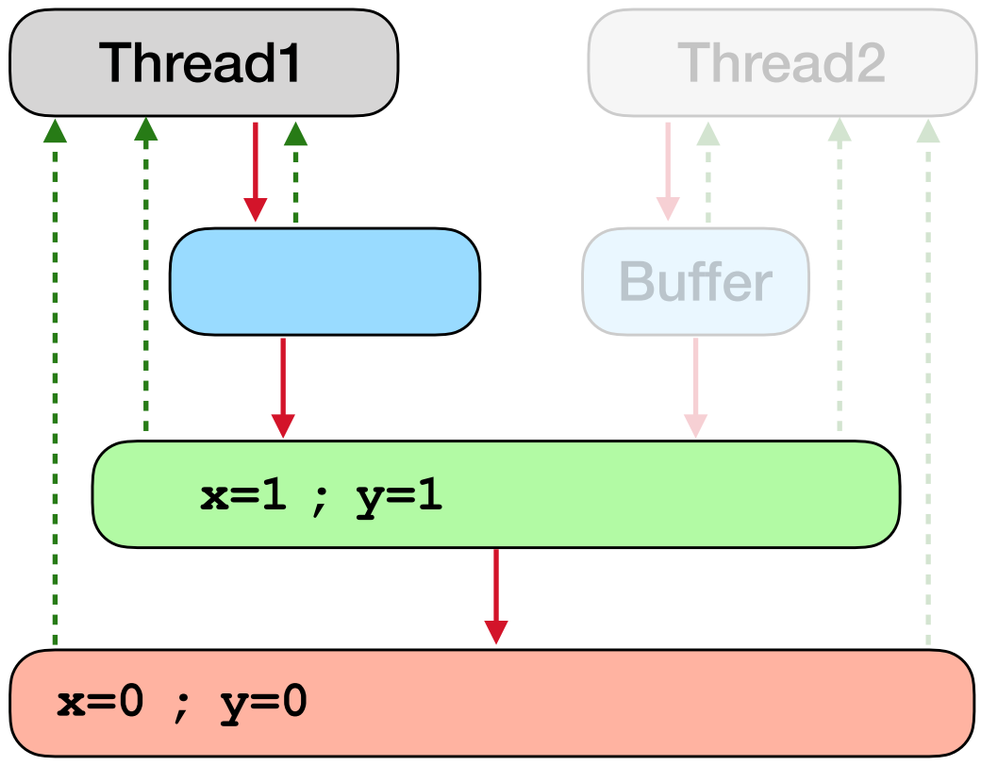


# Challenge #1: ***Weak Persistency***

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

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

!! Writes may persist (be unbuffered from pbuffer) ***out of order***

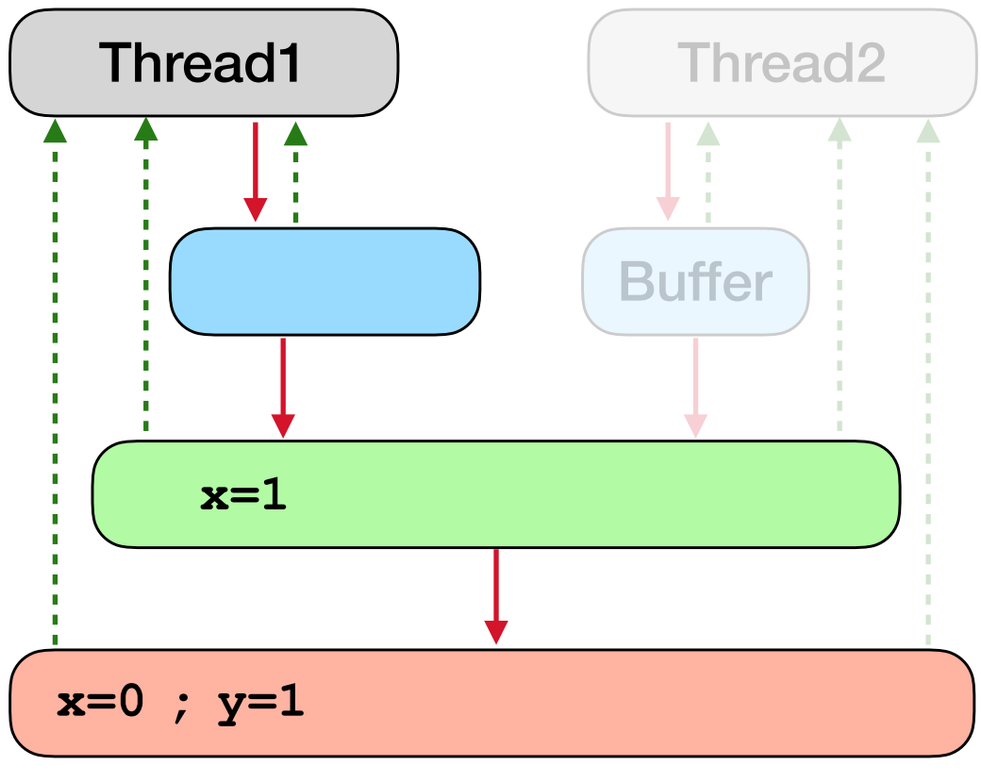


# Challenge #1: ***Weak Persistency***

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

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

!! Writes may persist (be unbuffered from pbuffer) ***out of order***

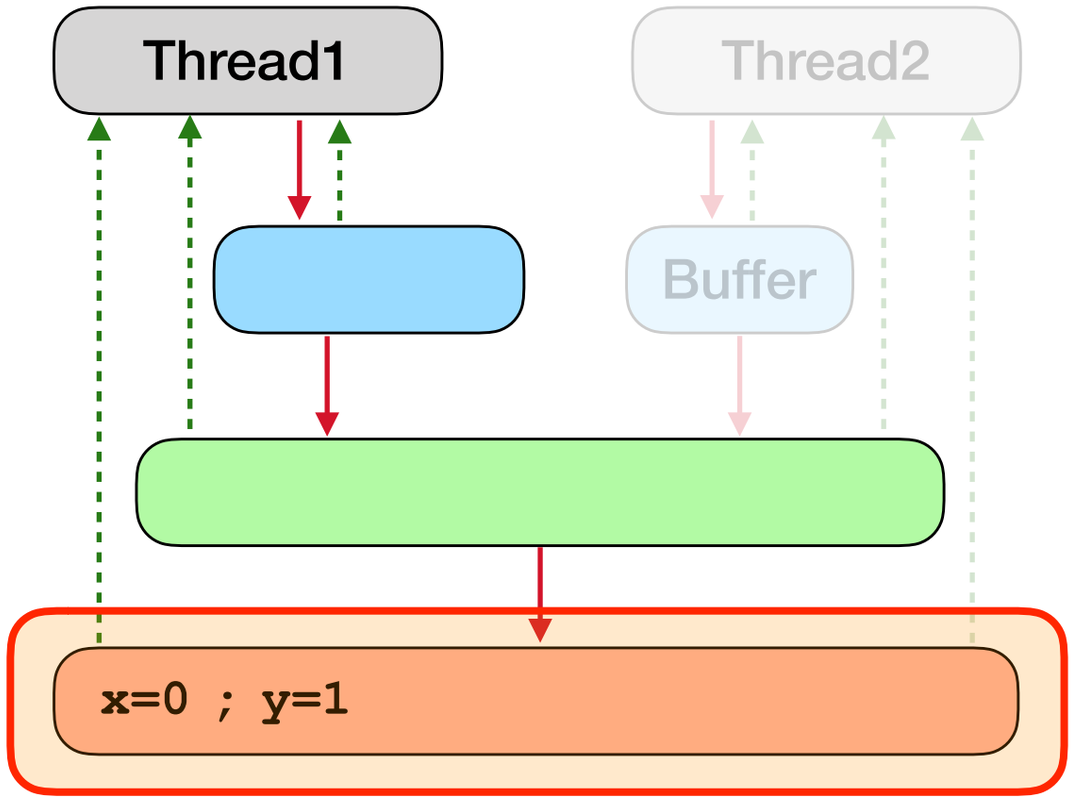


# Challenge #1: ***Weak Persistency***

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

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

!! Writes may persist (be unbuffered from pbuffer) ***out of order***

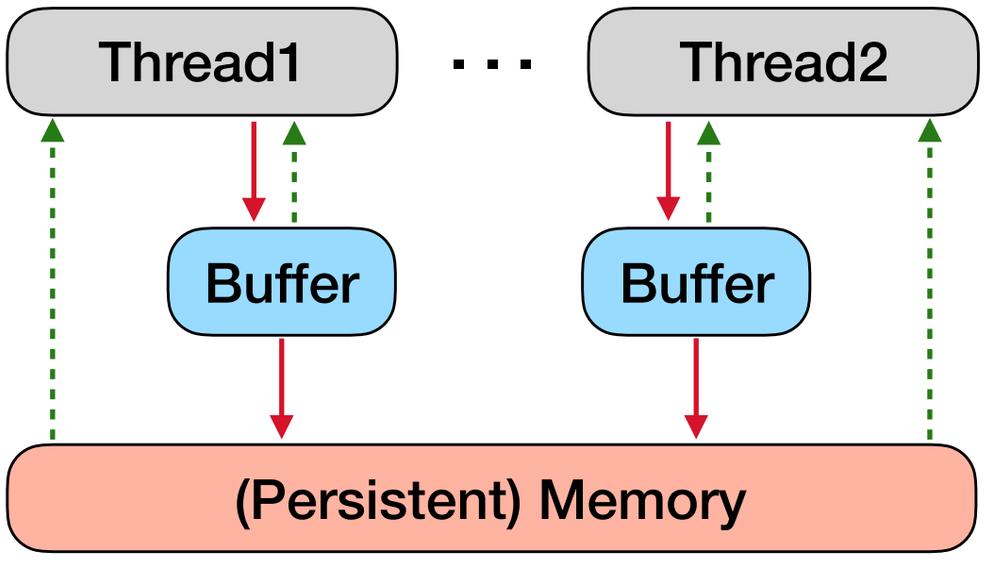


# Challenge #1: ***Weak Persistency***

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

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

!! Writes may persist (be unbuffered from pbuffer) ***out of order***



# Challenge #1: ***Weak Persistency***

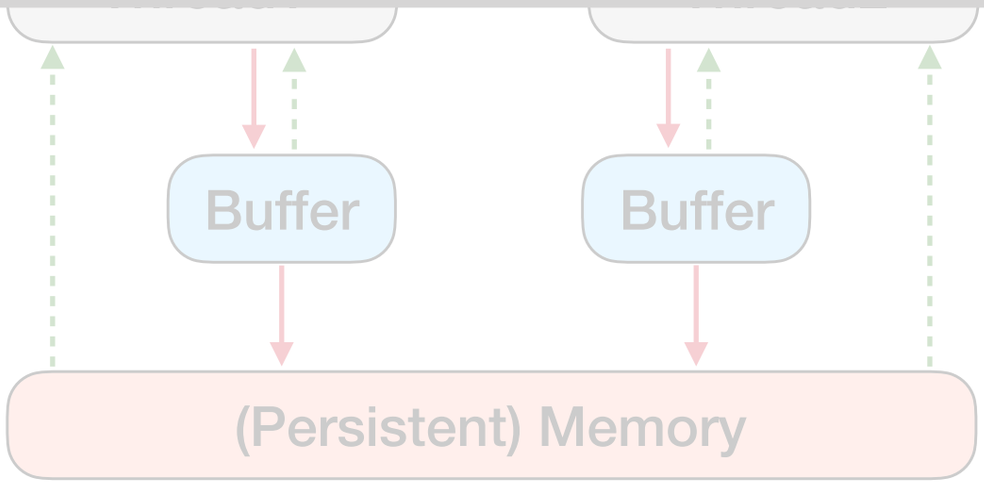
```
// x=y=0
x := 1;
y := 1;
```



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

!! Write … *order*

How to model this when we ***remove the*** pbuffer?



# Ix86 and Weak Persistency

Record ***two versions*** per Location $x$:

$x_v$ : volatile version

$x_p$ : persistent version

# Ix86 and Weak Persistency

Record ***two versions*** per Location $x$:

$x_v$ : volatile version

$x_p$ : persistent version

***Px86-to-Ix86 Translation:***

$x := 1$ ⇝ $x_v := 1$

# Ix86 and Weak Persistency

Record ***two versions*** per Location $x$:

$x_v$ : volatile version

$x_p$ : persistent version

***Px86-to-Ix86 Translation:***

| | | |
|---|---|---|
| $x:=1$ | ⇝ | $x_v:=1$ |
| $a:=x$ | ⇝ | $a:=x_v$ |

# Ix86 and Weak Persistency

Record ***two versions*** per Location $x$:

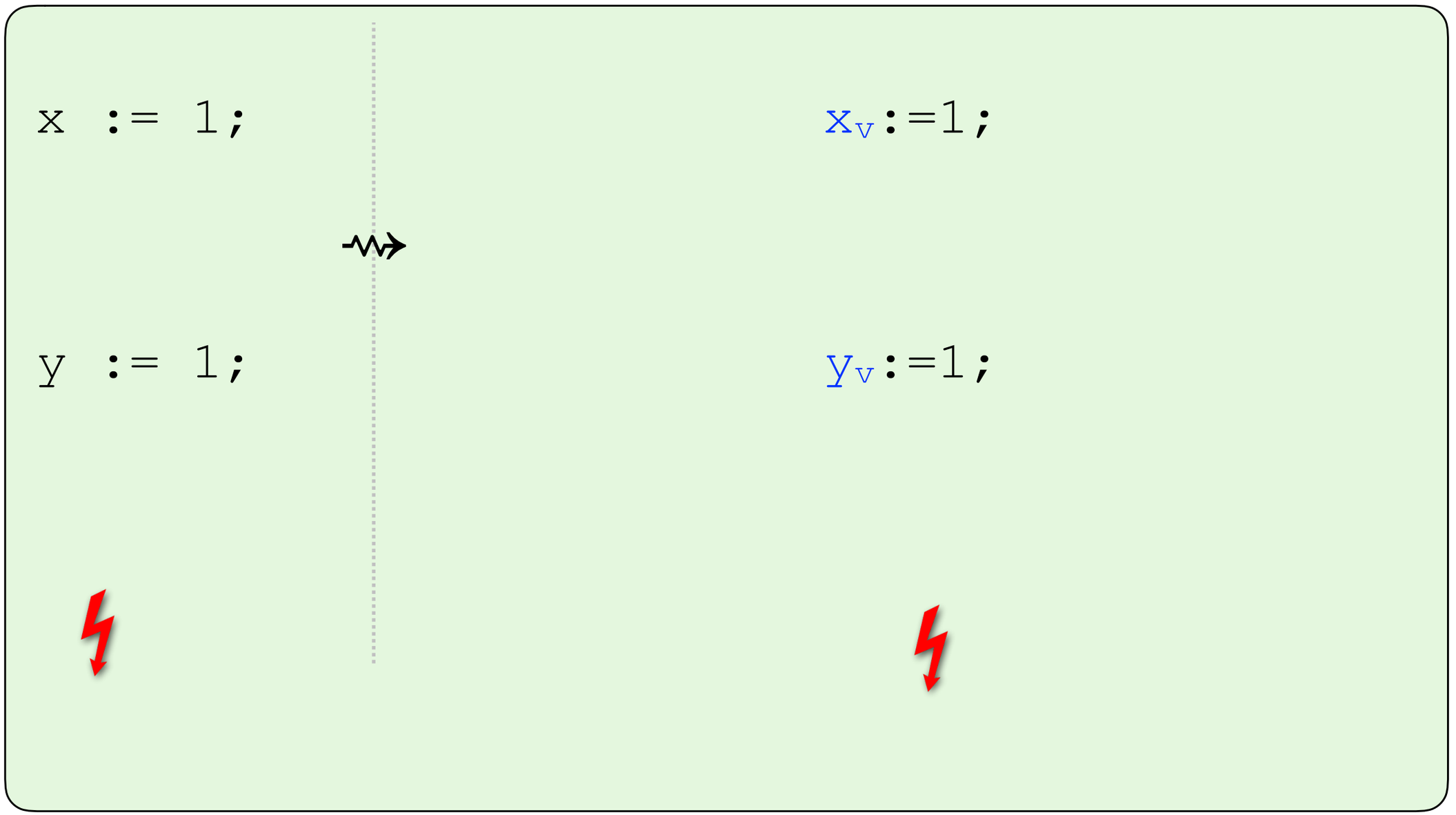
$x_v$ : volatile version

$x_p$ : persistent version

***Px86-to-Ix86 Translation:***

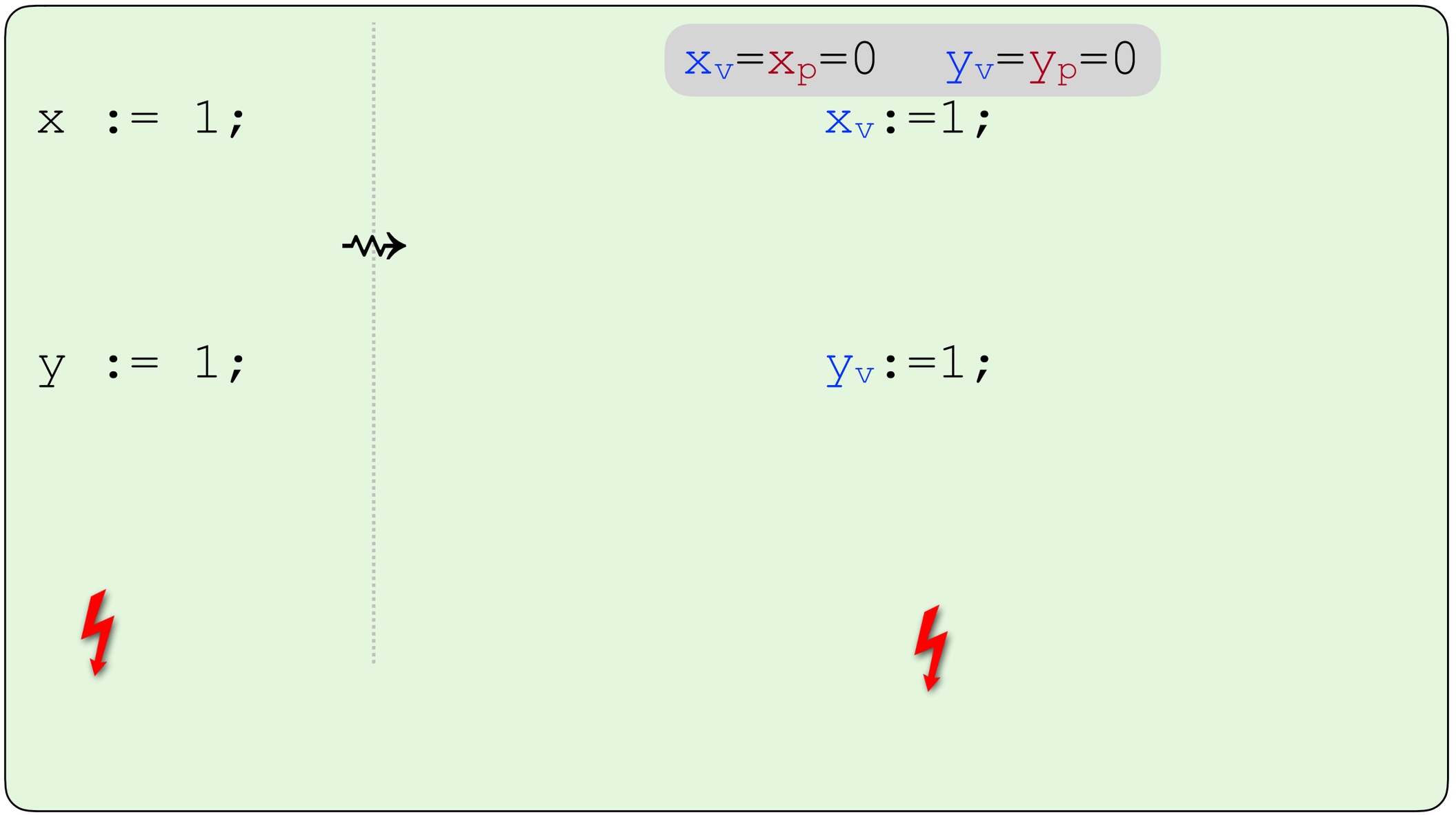
| | | |
|---|---|---|
| $x := 1$ | ⇝ | $x_v := 1$ |
| $a := x$ | ⇝ | $a := x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p := x_v$ |

# Ix86 and Weak Persistency



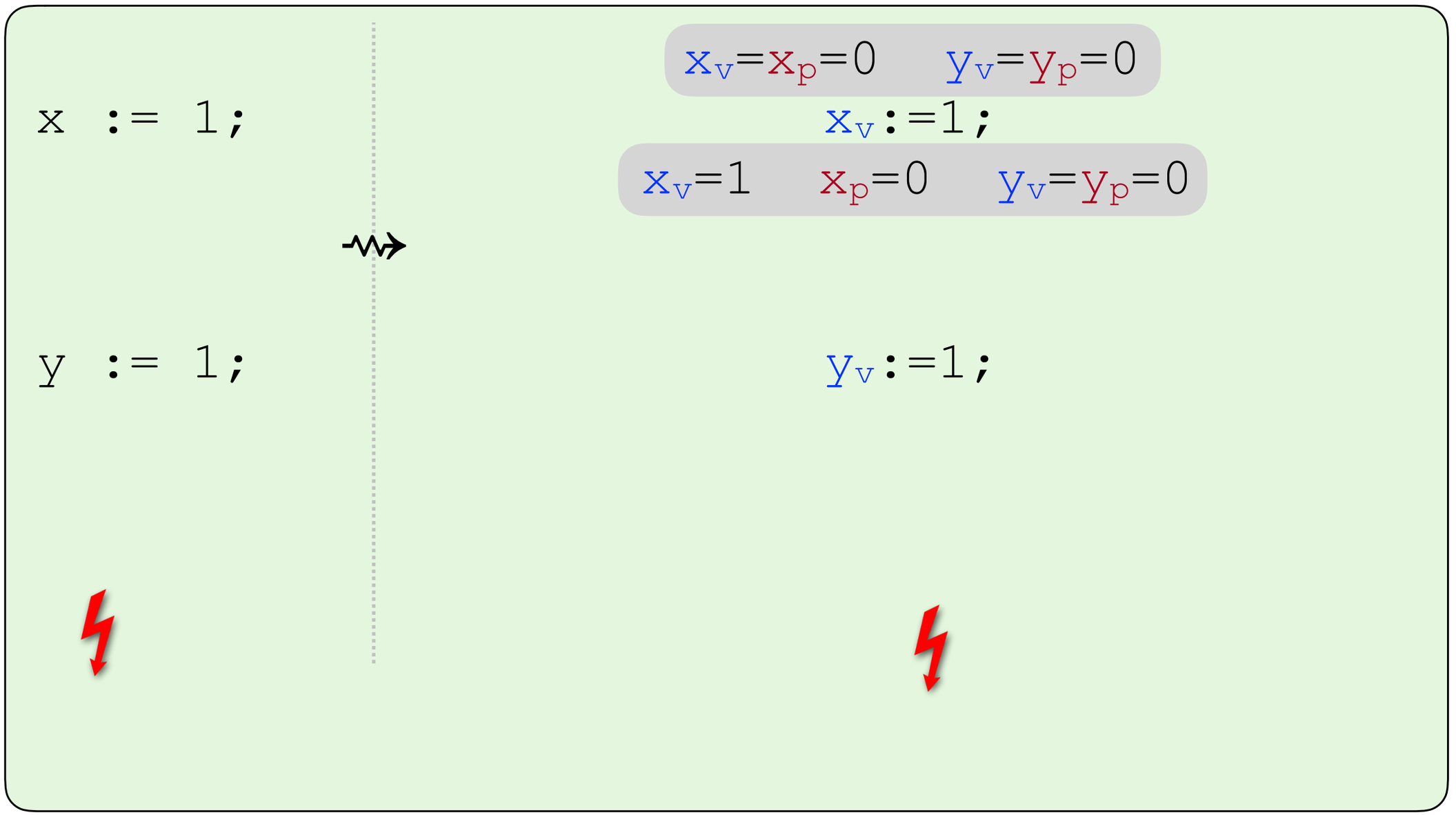

| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |

# Ix86 and Weak Persistency



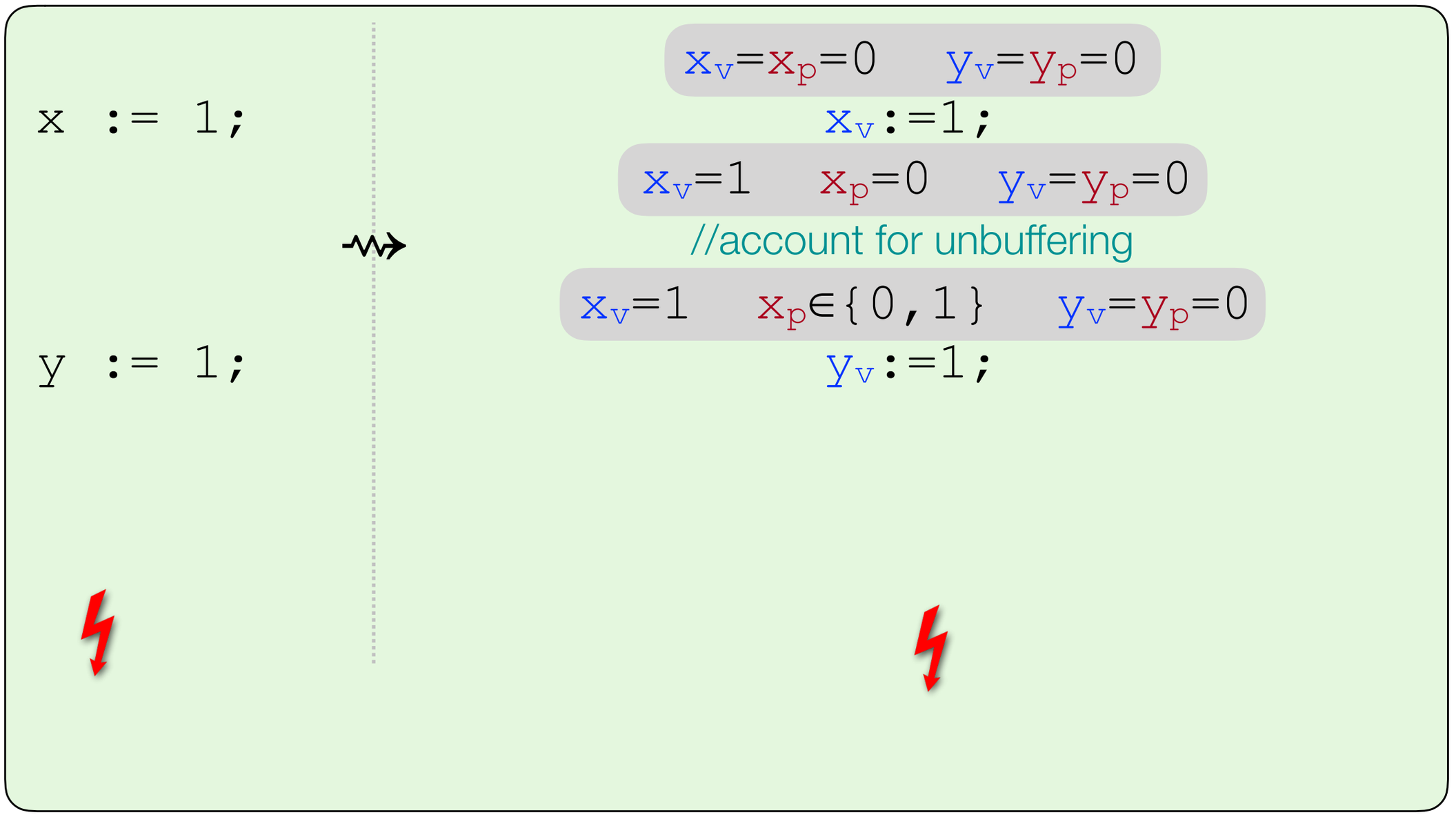

| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |

# Ix86 and Weak Persistency



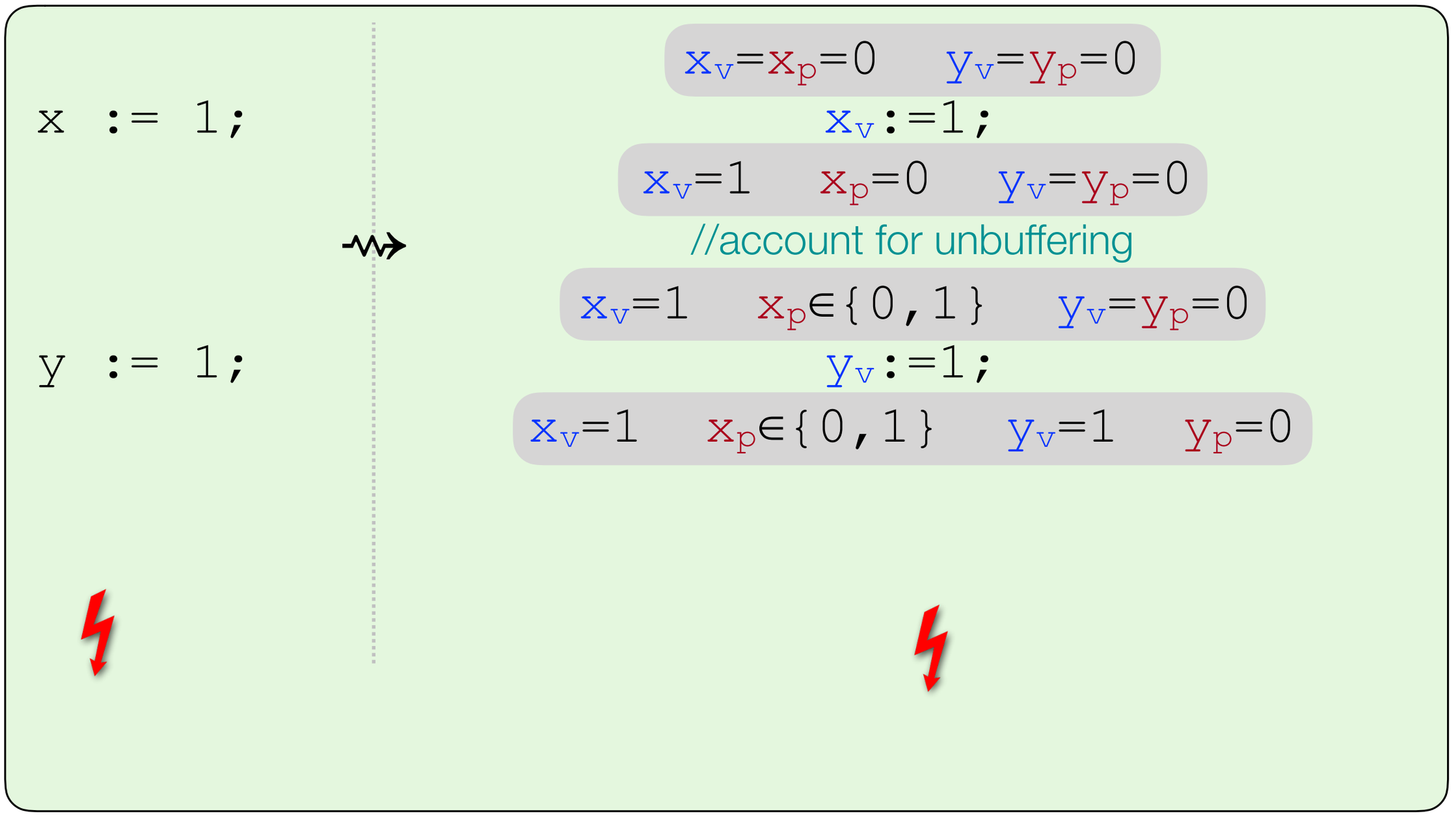

| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |

# Ix86 and Weak Persistency



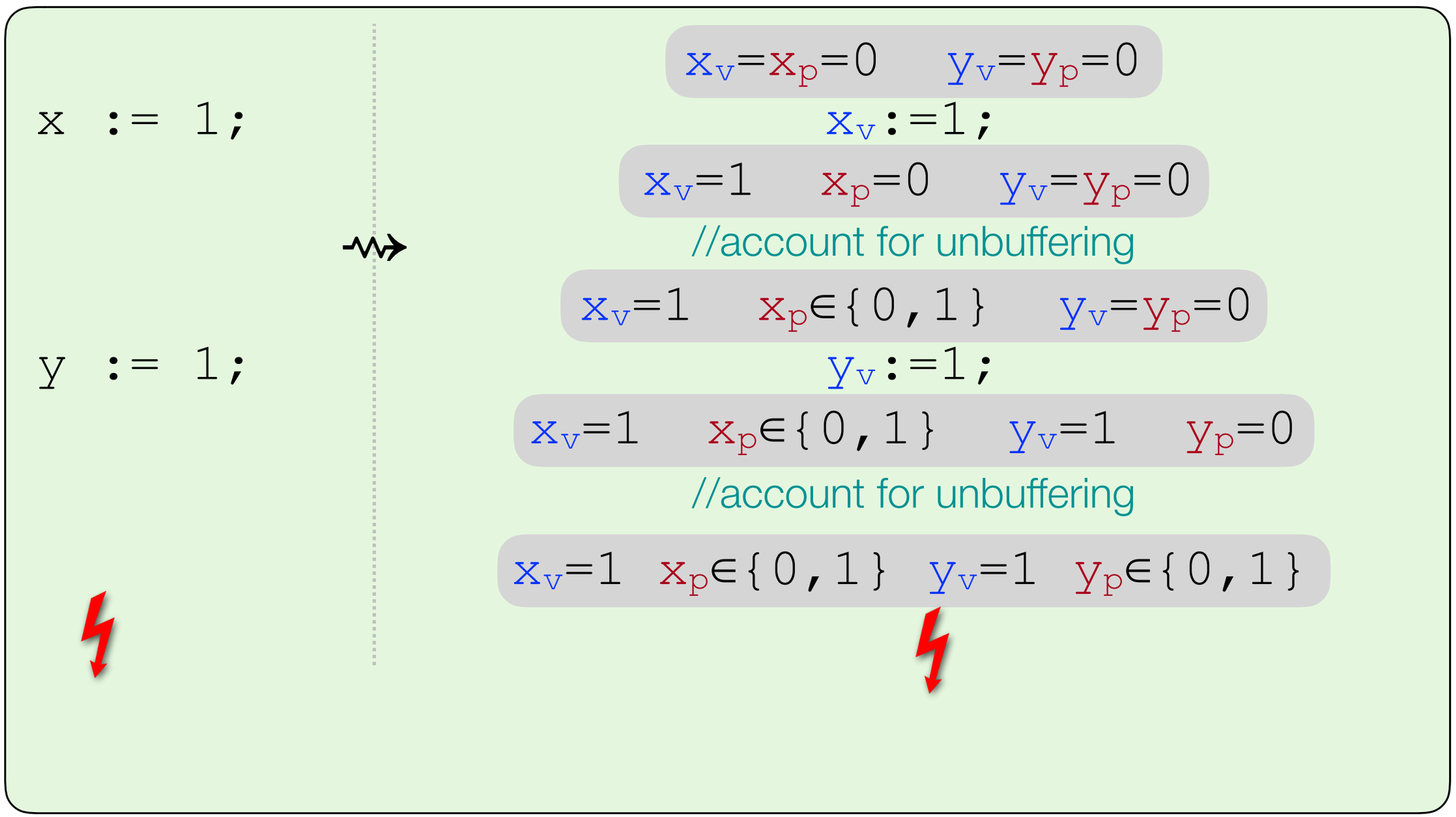

| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |

# Ix86 and Weak Persistency



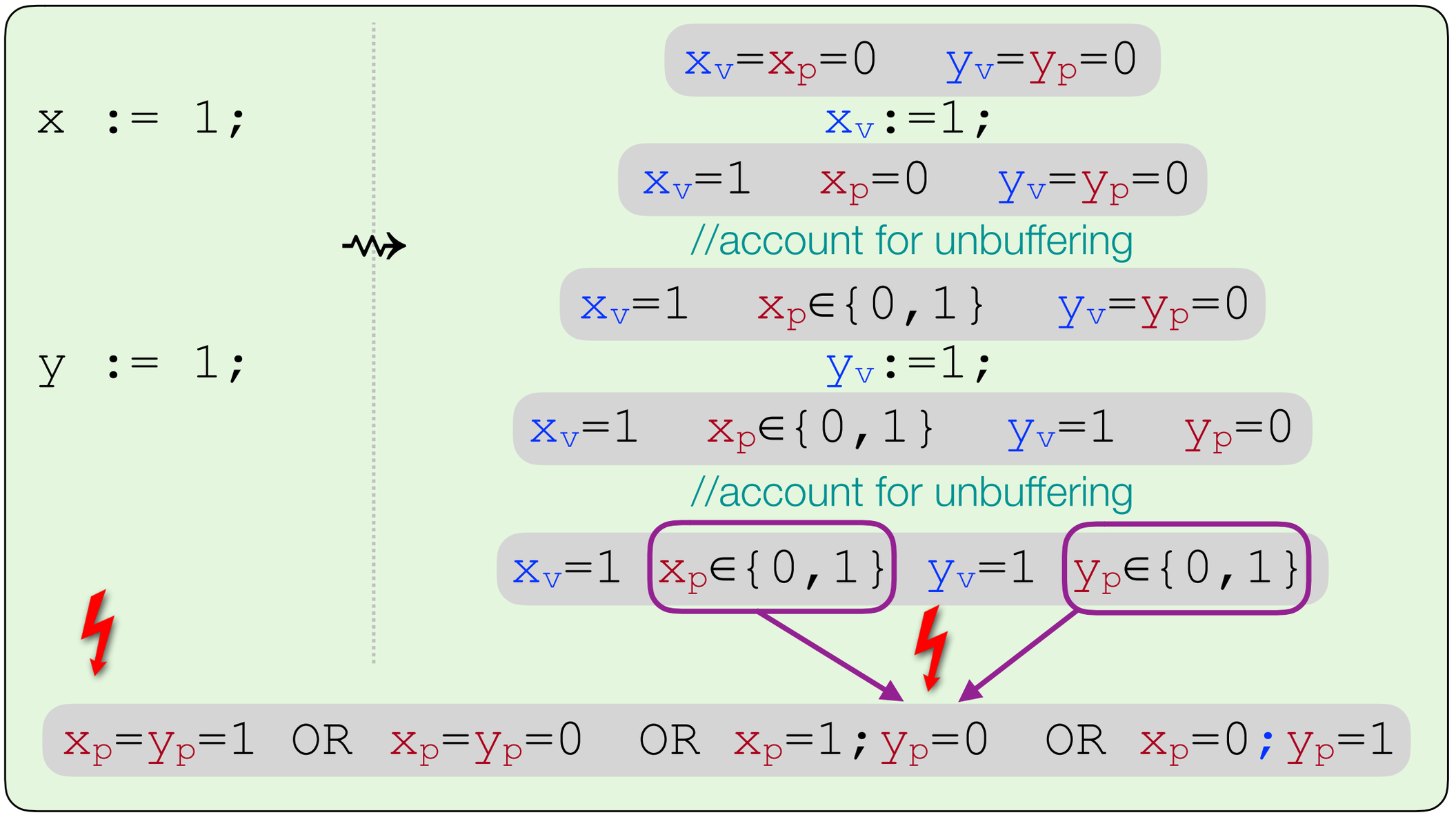

| | | |
|---|---|---|
| $x:=1$ | ⇝ | $x_v:=1$ |
| $a:=x$ | ⇝ | $a:=x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p:=x_v$ |

# Ix86 and Weak Persistency




| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |

# Ix86 and Weak Persistency




| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |

# Challenge #2: ***Asynchronous Persists***

```
// x=y=0

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

# Challenge #2: ***Asynchronous Persists***

```
// x=y=0

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

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

!! Explicit persists order writes in pbuffer

# Challenge #2: ***Asynchronous Persists***

```
// x=y=0

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

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

!! Explicit persists order writes in pbuffer

!! Explicit persists behave ***asynchronously***

— flush `x` does evict writes on `x` from the pbuffer

# Challenge #2: ***Asynchronous Persists***

```
// x=y=0

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

How to model this when we ***remove the*** pbuffer?

// x=y=1 OR x=1,y=0 OR x=0,y=1

!! Explicit persists order writes in pbuffer

!! Explicit persists behave ***asynchronously***

— flush x does evict writes on x from the pbuffer

# Ix86 and Asynchronous Persists — Naive Attempt

Record two versions per Location $x$:

$x_v$ : volatile version

$x_p$ : persistent version

***Px86-to-Ix86 Translation:***

| | | |
|---|---|---|
| $x := 1$ | ⇝ | $x_v := 1$ |
| $a := x$ | ⇝ | $a := x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p := x_v$ |

# Ix86 and Asynchronous Persists — Naive Attempt

Record two versions per Location `x`:

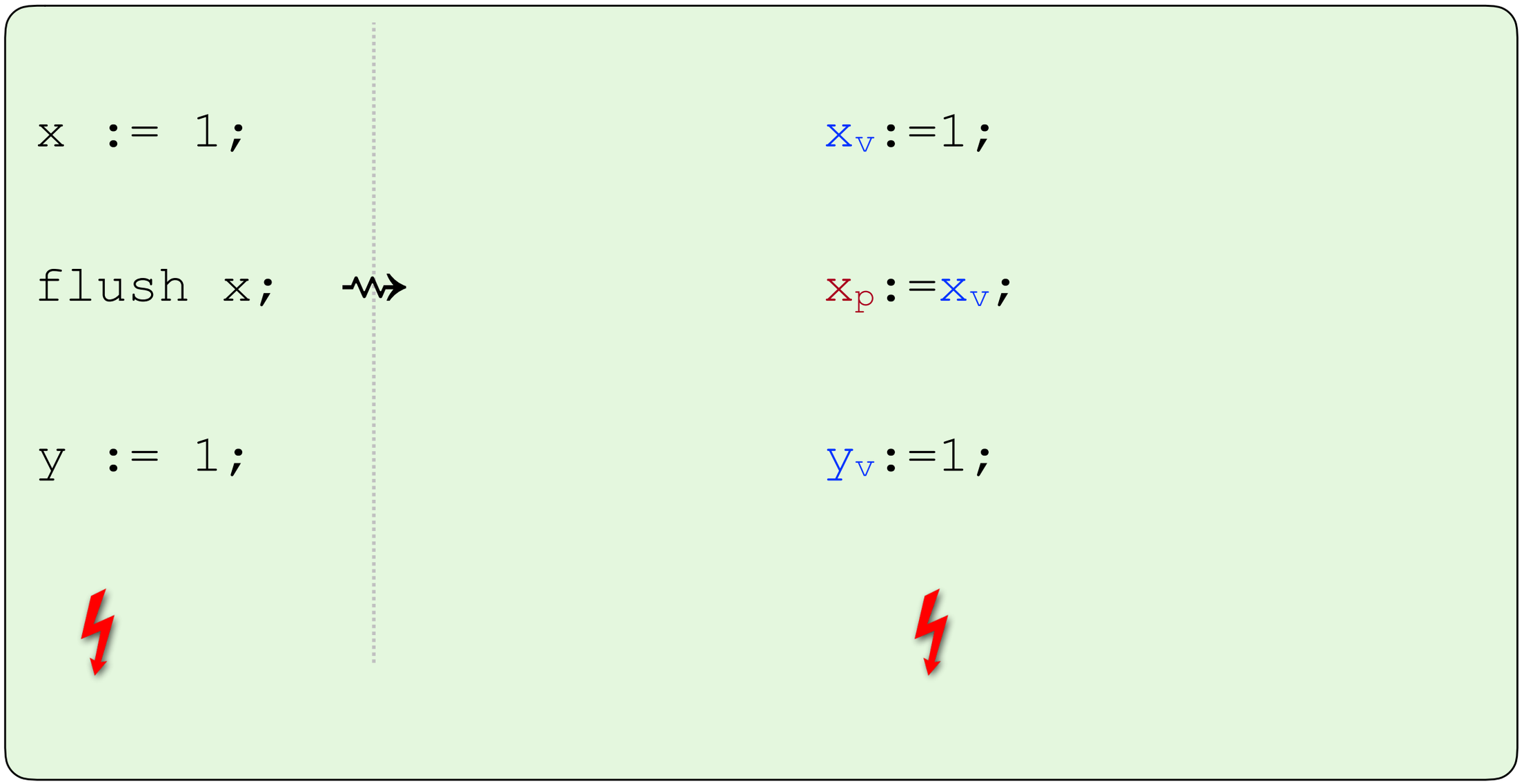
$x_v$ : volatile version

$x_p$ : persistent version

***Px86-to-Ix86 Translation:***

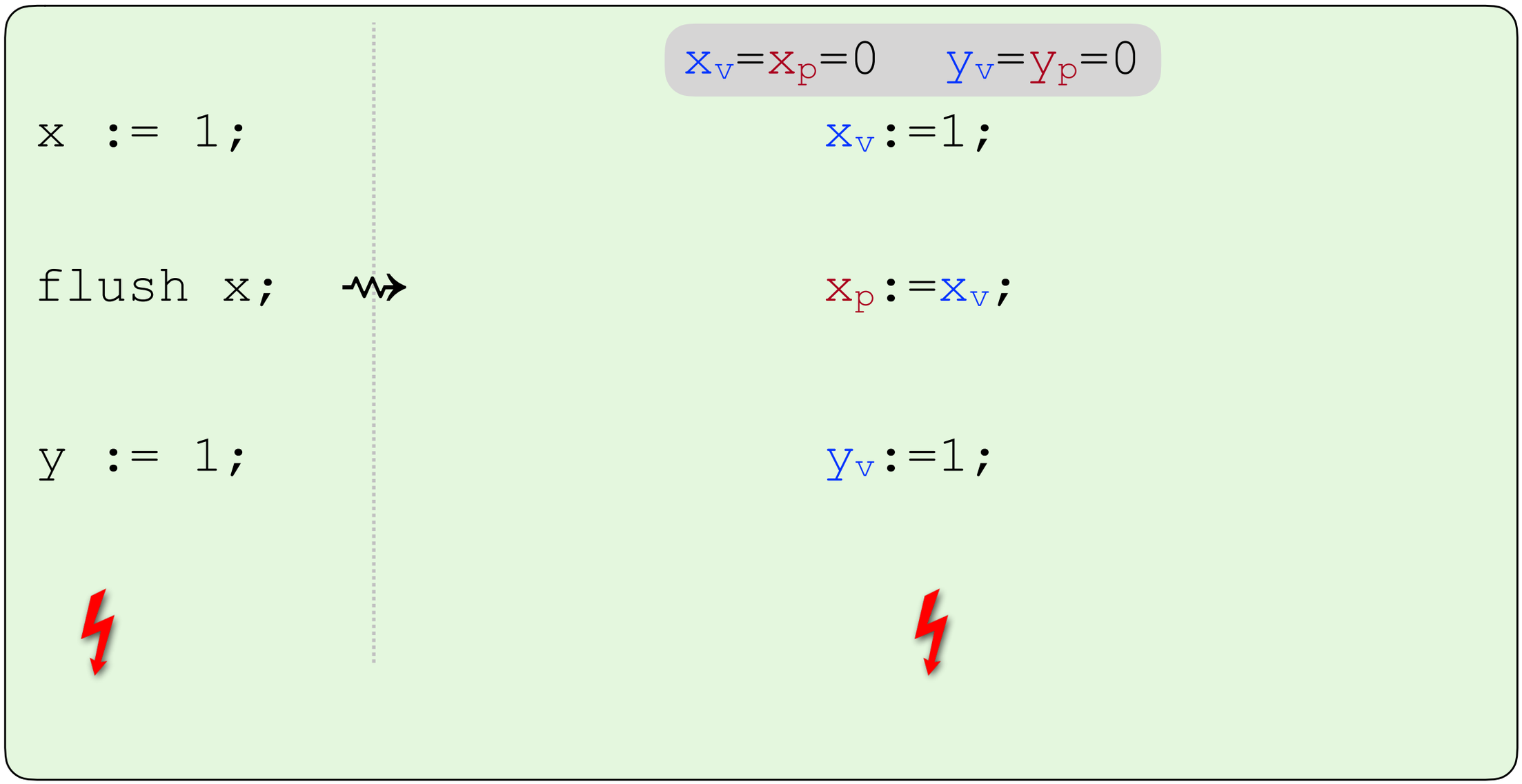
| | | |
|---|---|---|
| `x:=1` | ⇝ | $x_v$`:=1` |
| `a:=x` | ⇝ | `a:=`$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$`:=`$x_v$ |
| `flush x` | ⇝ | $x_p$`:=`$x_v$ |

# Ix86 and Asynchronous Persists — Naive Attempt



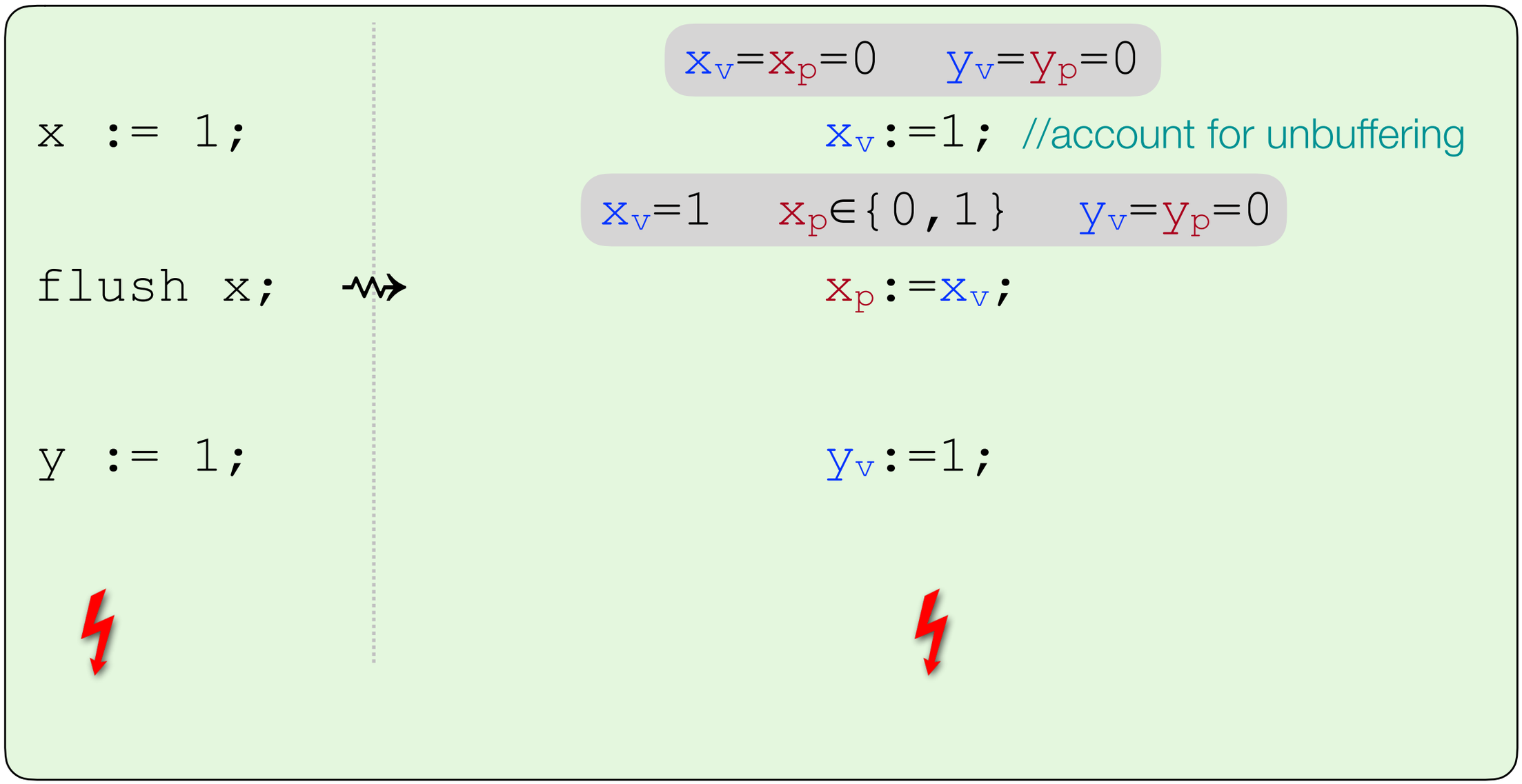

| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |
| flush x | ⇝ | $x_p$:=$x_v$ |

# Ix86 and Asynchronous Persists — Naive Attempt



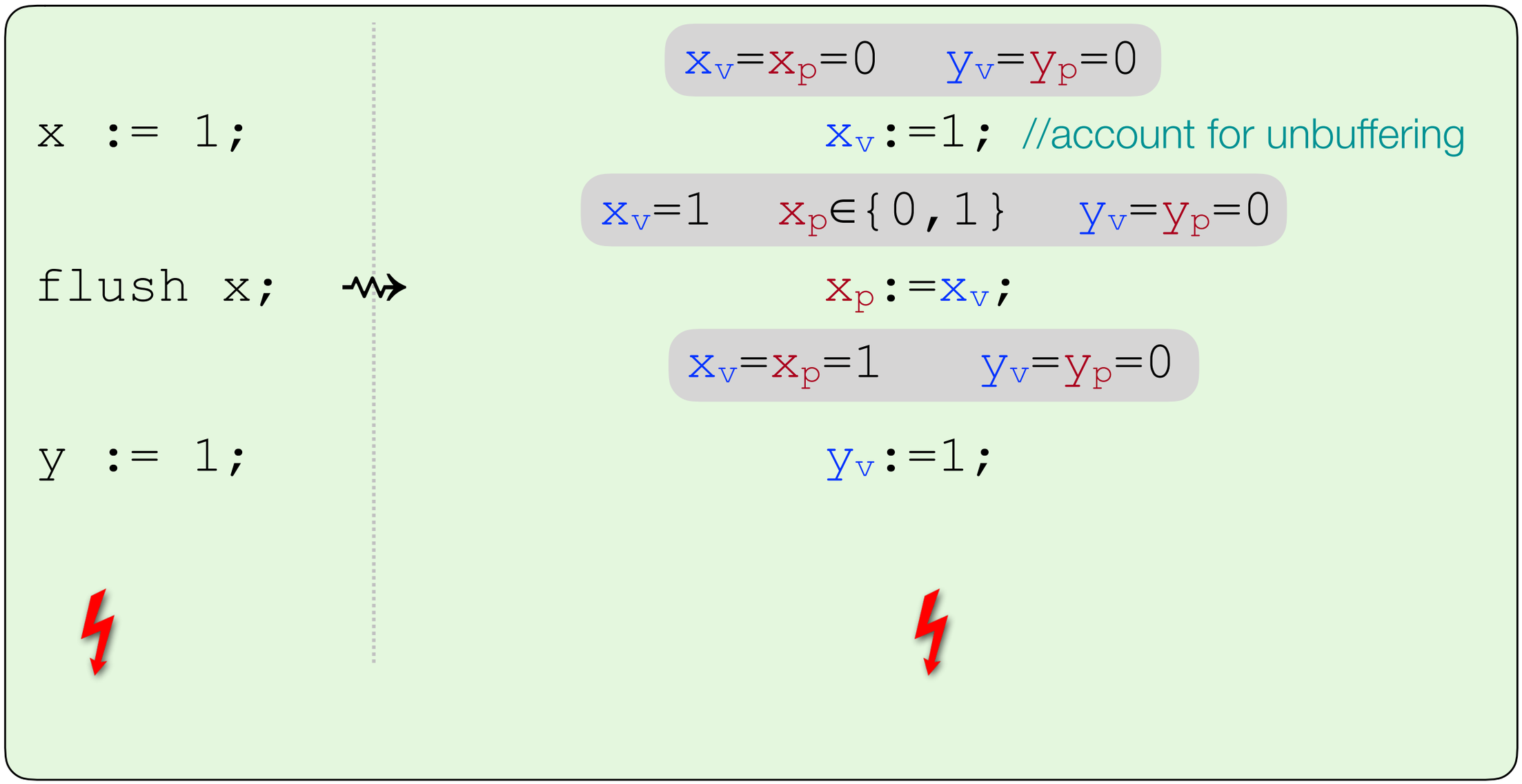

| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |
| flush x | ⇝ | $x_p$:=$x_v$ |

# Ix86 and Asynchronous Persists — Naive Attempt



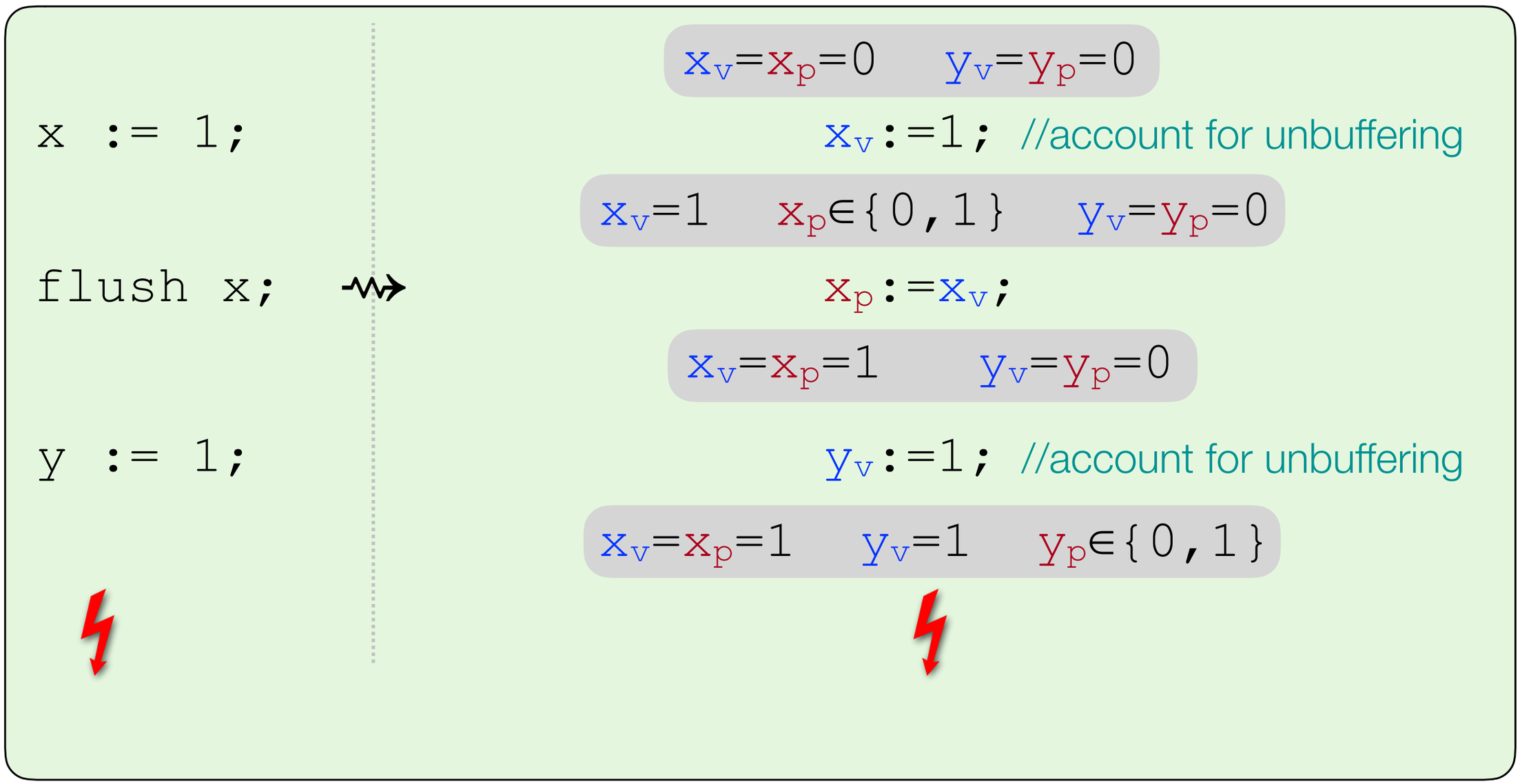

| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |
| flush x | ⇝ | $x_p$:=$x_v$ |

# Ix86 and Asynchronous Persists — Naive Attempt



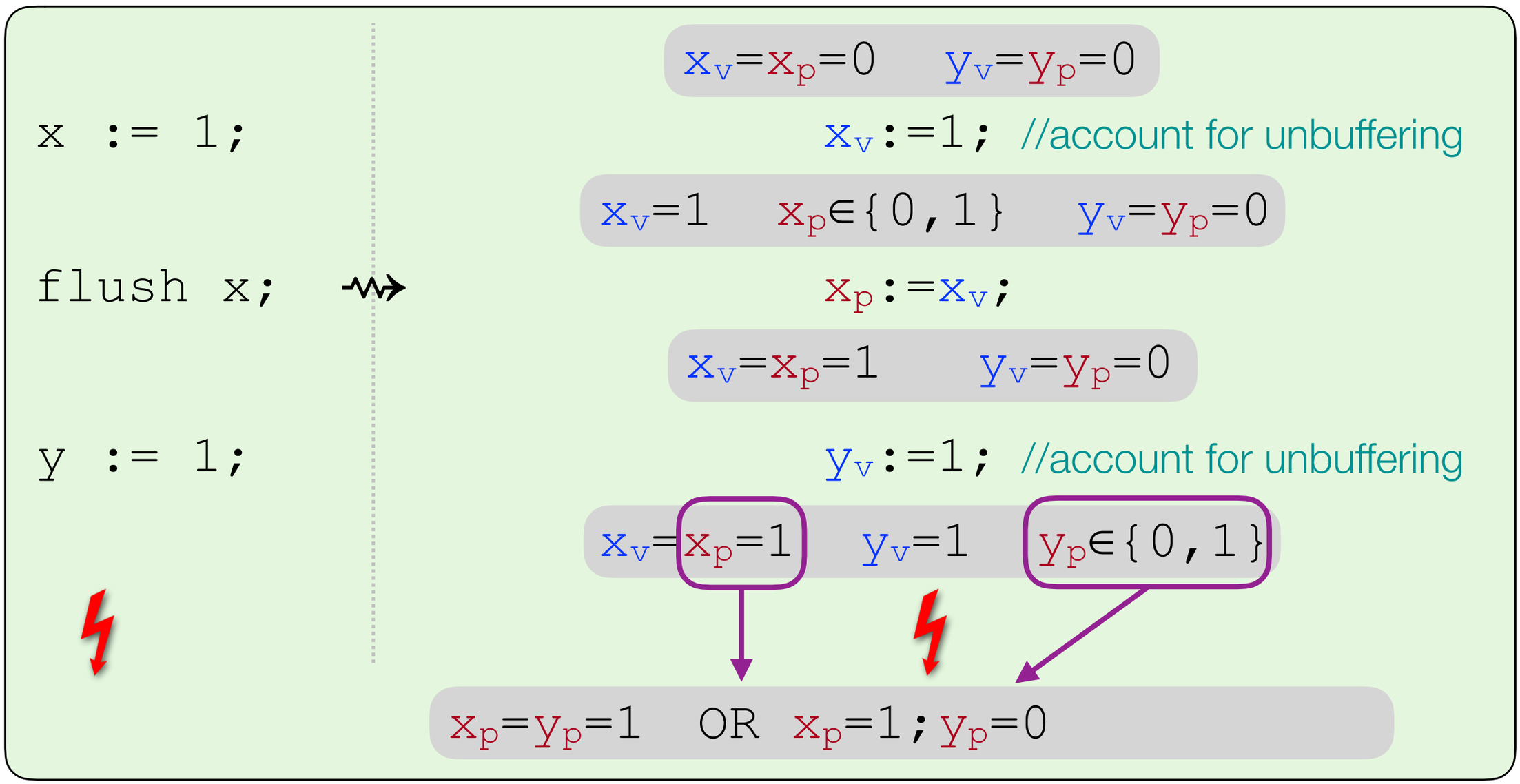

| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |
| flush x | ⇝ | $x_p$:=$x_v$ |

# Ix86 and Asynchronous Persists — Naive Attempt



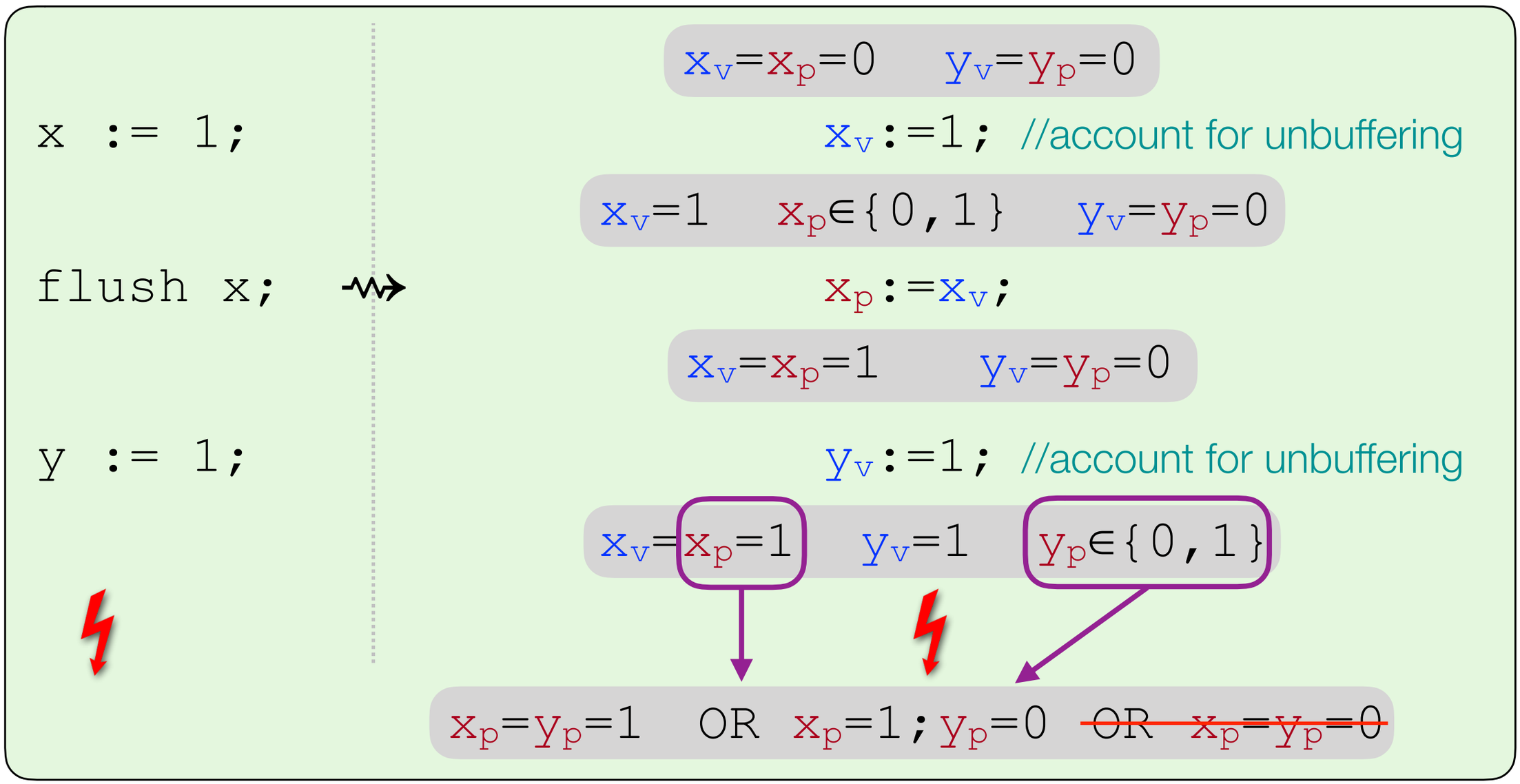

| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |
| flush x | ⇝ | $x_p$:=$x_v$ |

# Ix86 and Asynchronous Persists — Naive Attempt




| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |
| flush x | ⇝ | $x_p$:=$x_v$ |

# Ix86 and Asynchronous Persists — Naive Attempt




| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p$:=$x_v$ |
| flush x | ⇝ | $x_p$:=$x_v$ |

# Ix86 and Asynchronous Persists — Naive Attempt

$x_v{=}x_p{=}0 \quad y_v{=}y_p{=}0$

`x := 1;` → $x_v{:=}1$; //account for unbuffering

$x_v{=}1 \quad x_p{\in}\{0,1\} \quad y_v{=}y_p{=}0$

`flush x;` ⇝ $x_p{:=}x_v$;

$x_v{=}x_p{=}1 \quad y_v{=}y_p{=}0$

`y :=` … buffering

**Problem**

Models `flush` ***synchronously***!

$x_p{=}y_p{=}1$ OR $x_p{=}1; y_p{=}0$ ~~OR $x_p{=}y_p{=}0$~~

| | | |
|---|---|---|
| `x:=1` | ⇝ | $x_v{:=}1$ |
| `a:=x` | ⇝ | $a{:=}x_v$ |
| unbuffering (***non-det.*** times) | ⇝ | $x_p{:=}x_v$ |
| `flush x` | ⇝ | $x_p{:=}x_v$ |

# Ix86 and Asynchronous Persists

Record ***three versions*** per Location `x`:

$x_v$ : volatile version

$x_p$ : persistent version

$x_s$ : synchronous version
(for modelling `flush`)

# Ix86 and Asynchronous Persists

Record ***three versions*** per Location `x`:

$x_v$ : volatile version

$x_p$ : persistent version

$x_s$ : synchronous version
(for modelling `flush`)

***Px86-to-Ix86 Translation:***

`x:=1` ⇝ $x_v$`:=1`

`a:=x` ⇝ `a:=`$x_v$

# Ix86 and Asynchronous Persists

Record ***three versions*** per Location `x`:

$x_v$ : volatile version

$x_p$ : persistent version

$x_s$ : synchronous version
(for modelling `flush`)

***Px86-to-Ix86 Translation:***

| | | |
|---|---|---|
| `x:=1` | ⇝ | $x_v$`:=1` |
| `a:=x` | ⇝ | `a:=`$x_v$ |
| `flush x` | ⇝ | $x_s$`:=`$x_v$ |

# Ix86 and Asynchronous Persists

Record ***three versions*** per Location `x`:

$x_v$ : volatile version

$x_p$ : persistent version

$x_s$ : synchronous version
(for modelling `flush`)

***Px86-to-Ix86 Translation:***

| | | |
|---|---|---|
| `x:=1` | ⇝ | $x_v$`:=1` |
| `a:=x` | ⇝ | `a:=`$x_v$ |
| `flush x` | ⇝ | $x_s$`:=`$x_v$ |
| unbuffering (***non-det.*** times) (buffer→pbuffer) | ⇝ | $x_s$`:=`$x_v$ |

# Ix86 and Asynchronous Persists

Record ***three versions*** per Location `x`:

$x_v$ : volatile version

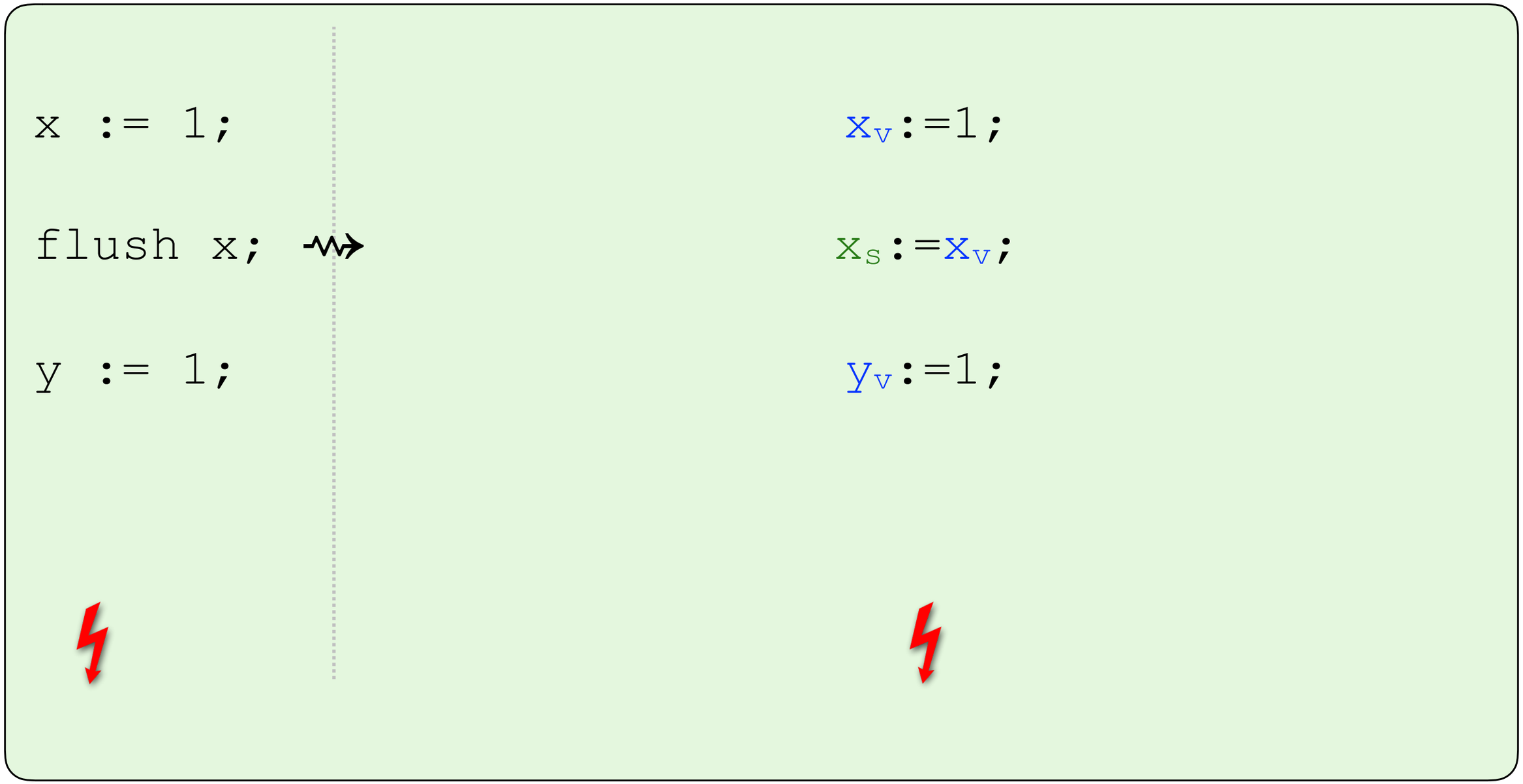
$x_p$ : persistent version

$x_s$ : synchronous version
(for modelling `flush`)

***Px86-to-Ix86 Translation:***

| | | |
|---|---|---|
| `x:=1` | ⇝ | $x_v$`:=1` |
| `a:=x` | ⇝ | `a:=`$x_v$ |
| `flush x` | ⇝ | $x_s$`:=`$x_v$ |
| unbuffering (***non-det.*** times) | | |
| (buffer→pbuffer) | ⇝ | $x_s$`:=`$x_v$ |
| (pbuffer→memory) | ⇝ | $\overline{x_p := x_s}$ |

# Ix86 and Asynchronous Persists




| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| flush x | ⇝ | $x_s$:=$x_v$ |
| unbuffering (***non-det*** times) | | |
| (buffer→pbuffer) | ⇝ | $x_s$:=$x_v$ |
| (pbuffer→memory) | ⇝ | $\overline{x_p := x_s}$ |

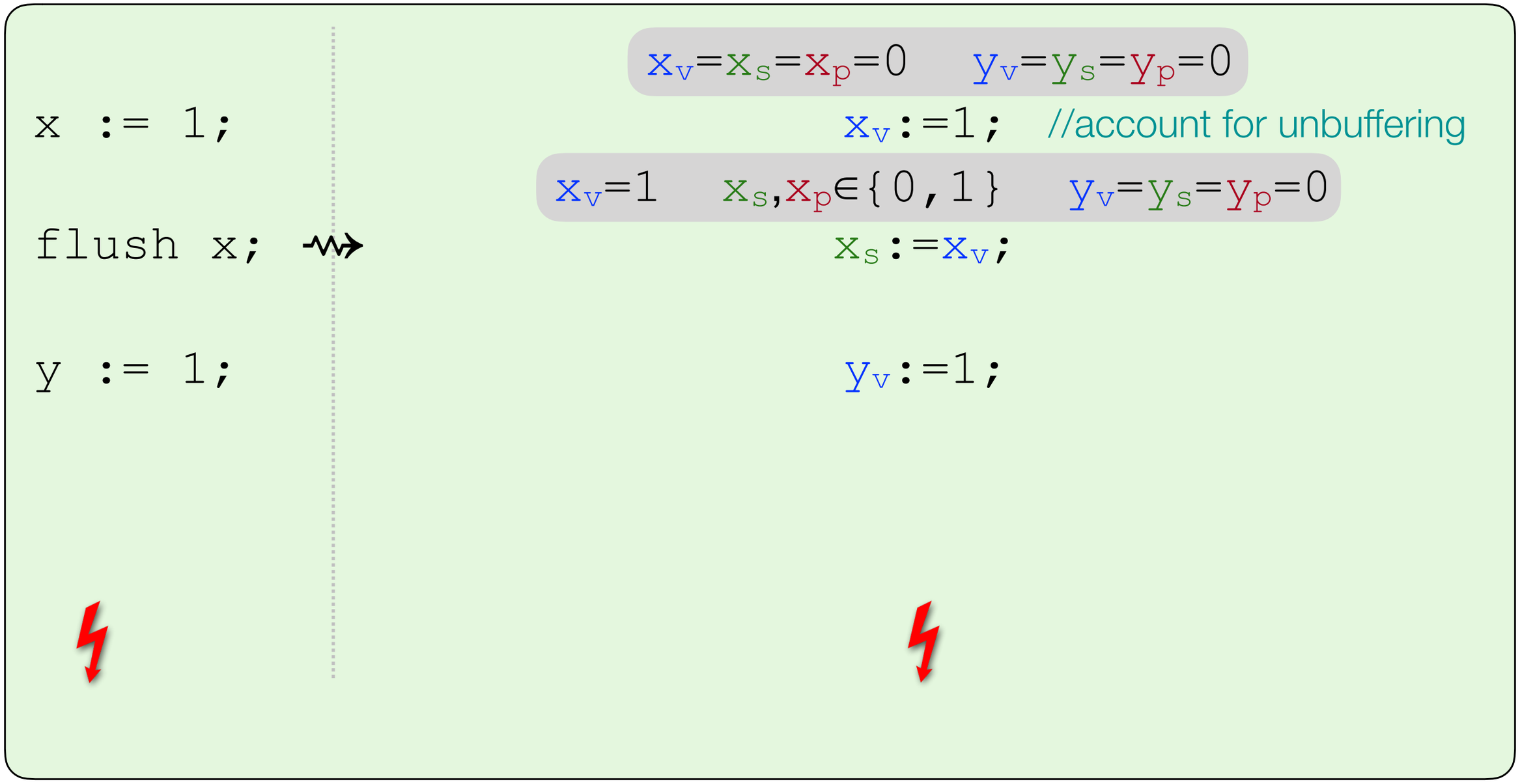
# Ix86 and Asynchronous Persists

$x_v=x_s=x_p=0 \quad y_v=y_s=y_p=0$

| | |
|---|---|
| `x := 1;` | $x_v:=1;$ |
| `flush x;` ⇝ | $x_s:=x_v;$ |
| `y := 1;` | $y_v:=1;$ |

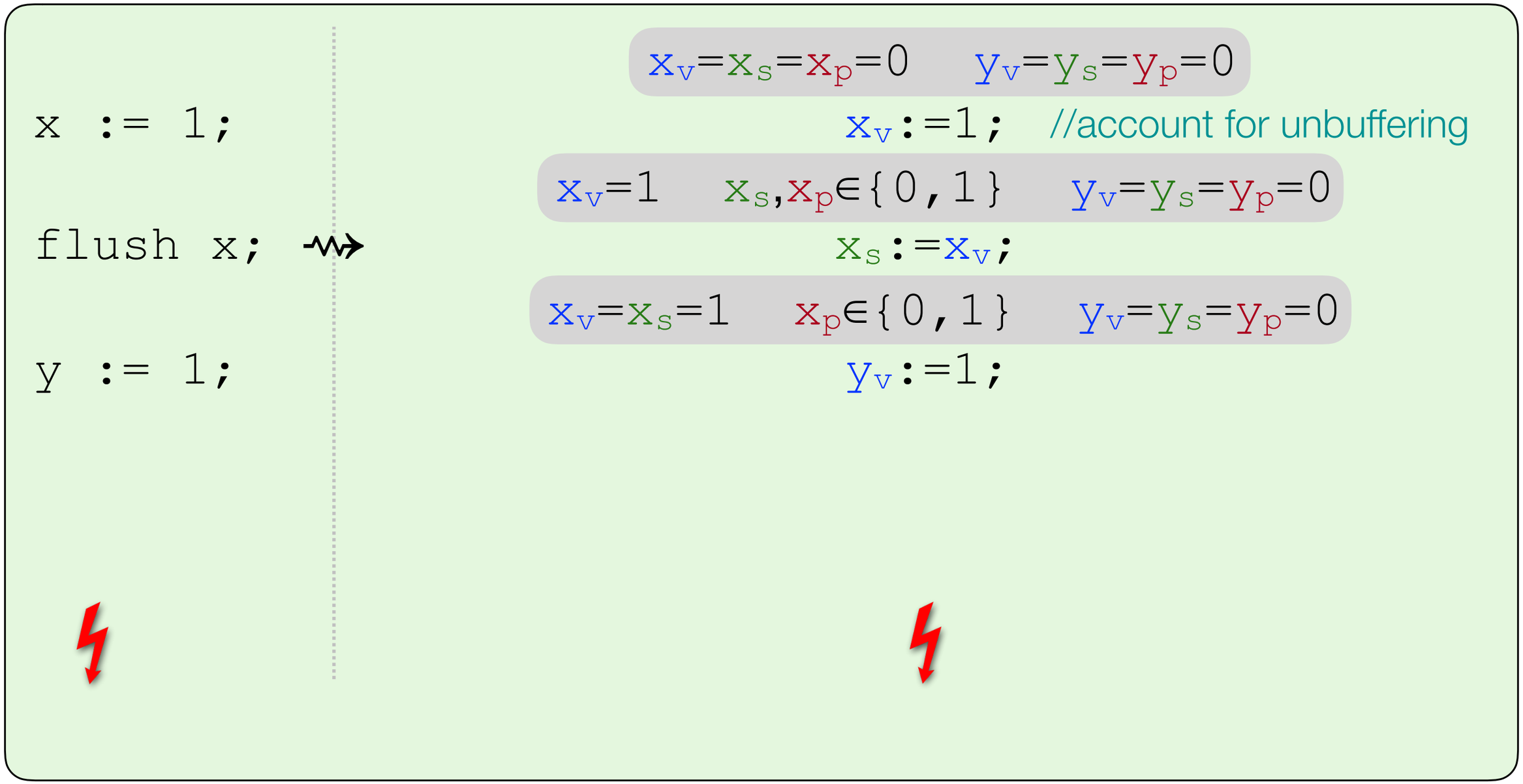
| | | |
|---|---|---|
| `x:=1` | ⇝ | $x_v:=1$ |
| `a:=x` | ⇝ | $a:=x_v$ |
| `flush x` | ⇝ | $x_s:=x_v$ |
| unbuffering (***non-det*** times) | | |
| (buffer→pbuffer) | ⇝ | $x_s:=x_v$ |
| (pbuffer→memory) | ⇝ | $\overline{x_p:=x_s}$ |

# Ix86 and Asynchronous Persists



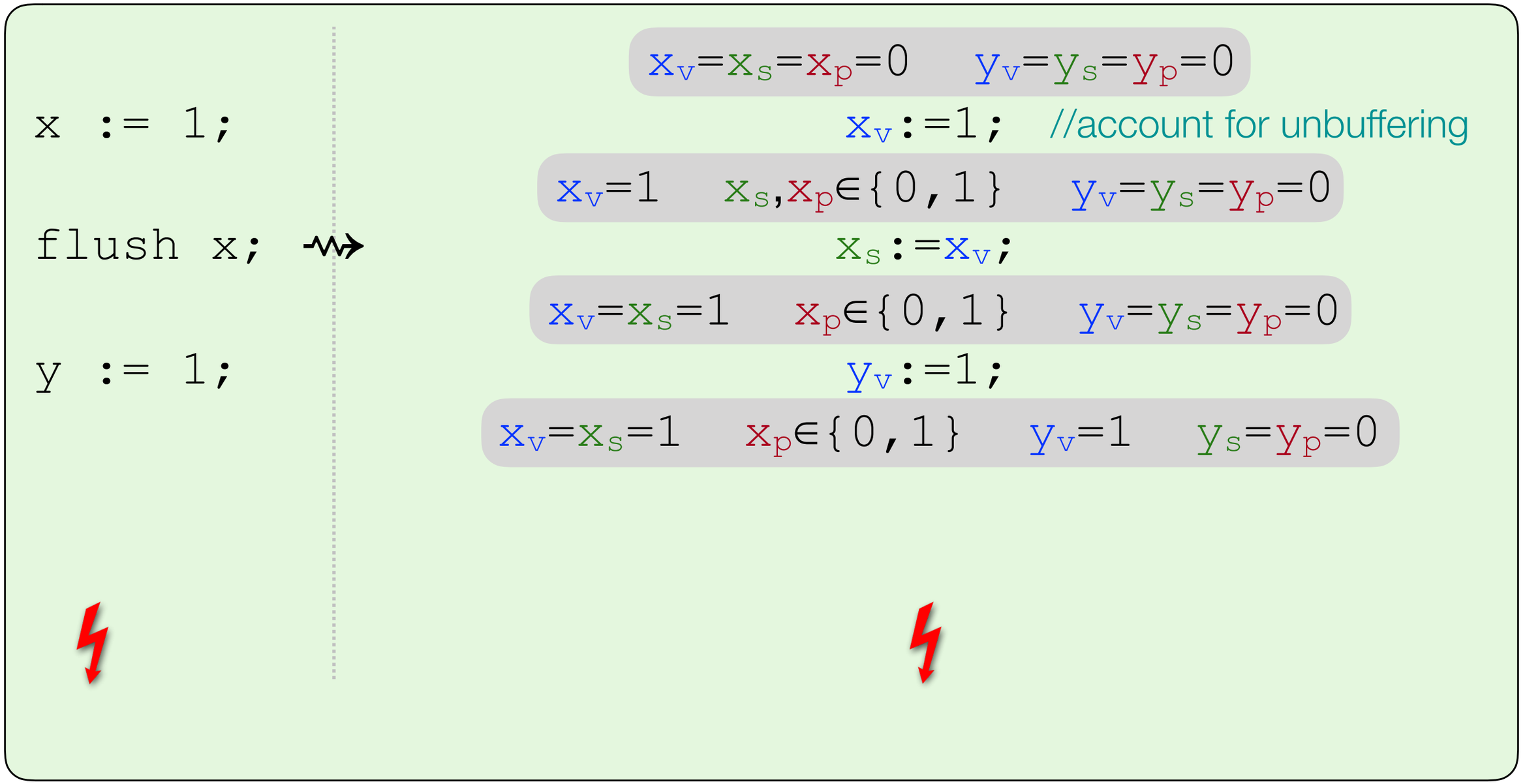

| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| flush x | ⇝ | $x_s$:=$x_v$ |
| unbuffering (***non-det*** times) | | |
| (buffer→pbuffer) | ⇝ | $x_s$:=$x_v$ |
| (pbuffer→memory) | ⇝ | $\overline{x_p:=x_s}$ |

# Ix86 and Asynchronous Persists




| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| flush x | ⇝ | $x_s$:=$x_v$ |
| unbuffering (***non-det*** times) | | |
| (buffer→pbuffer) | ⇝ | $x_s$:=$x_v$ |
| (pbuffer→memory) | ⇝ | $\overline{x_p:=x_s}$ |

# Ix86 and Asynchronous Persists




| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v$:=1 |
| a:=x | ⇝ | a:=$x_v$ |
| flush x | ⇝ | $x_s$:=$x_v$ |
| unbuffering (***non-det*** times) | | |
| (buffer→pbuffer) | ⇝ | $x_s$:=$x_v$ |
| (pbuffer→memory) | ⇝ | $\overline{x_p:=x_s}$ |

# Ix86 and Asynchronous Persists

```
x := 1;

flush x;  ⇝

y := 1;
```

$x_v{=}x_s{=}x_p{=}0 \quad y_v{=}y_s{=}y_p{=}0$

$x_v{:=}1$; //account for unbuffering

$x_v{=}1 \quad x_s,x_p{\in}\{0,1\} \quad y_v{=}y_s{=}y_p{=}0$

$x_s{:=}x_v$;

$x_v{=}x_s{=}1 \quad x_p{\in}\{0,1\} \quad y_v{=}y_s{=}y_p{=}0$

$y_v{:=}1$;

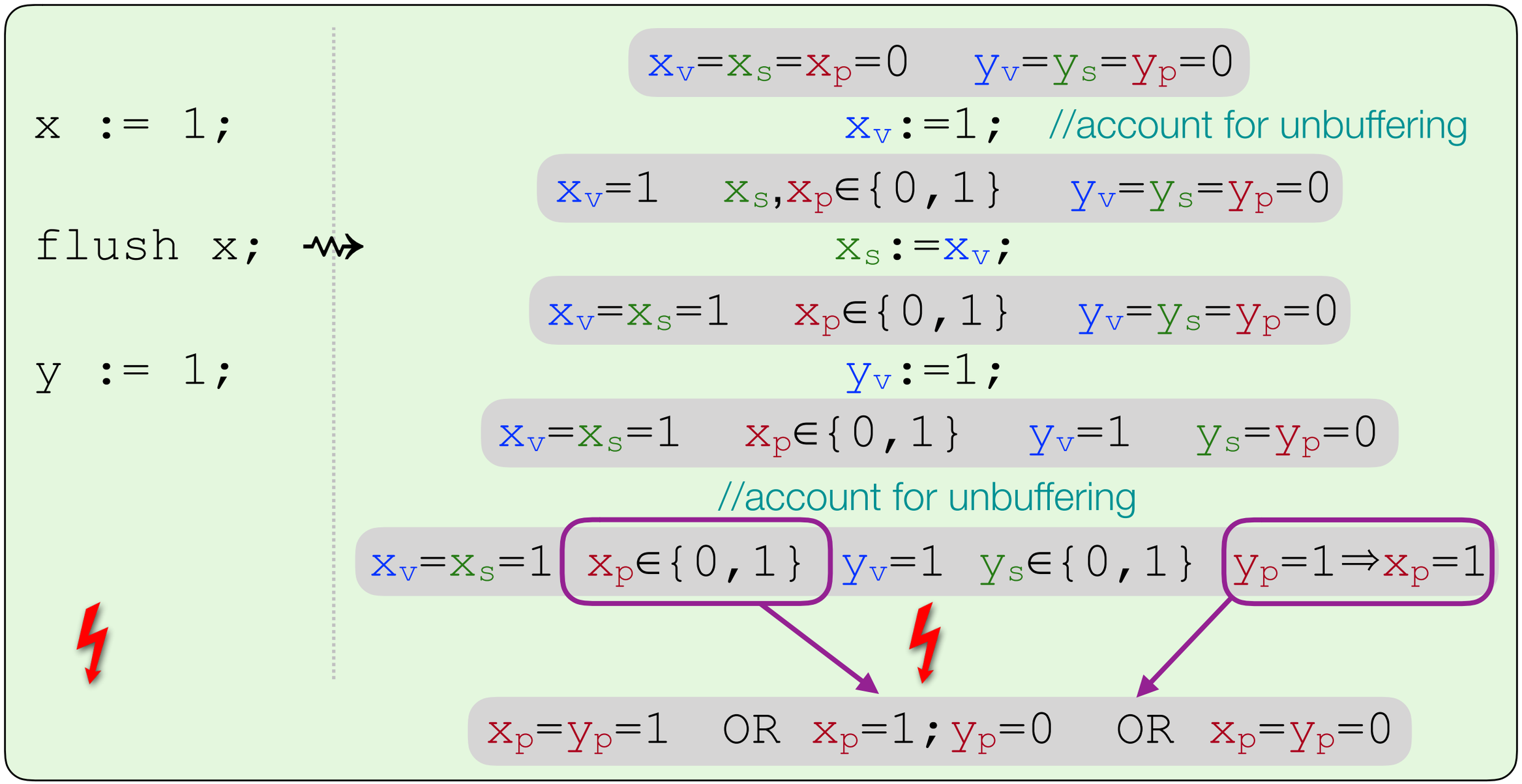
$x_v{=}x_s{=}1 \quad x_p{\in}\{0,1\} \quad y_v{=}1 \quad y_s{=}y_p{=}0$

//account for unbuffering

$x_v{=}x_s{=}1 \quad x_p{\in}\{0,1\} \quad y_v{=}1 \quad y_s{\in}\{0,1\} \quad y_p{=}1{\Rightarrow}x_p{=}1$

| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v{:=}1$ |
| a:=x | ⇝ | $a{:=}x_v$ |
| flush x | ⇝ | $x_s{:=}x_v$ |
| unbuffering (***non-det*** times) | | |
| (buffer→pbuffer) | ⇝ | $x_s{:=}x_v$ |
| (pbuffer→memory) | ⇝ | $\overline{x_p{:=}x_s}$ |

# Ix86 and Asynchronous Persists




| | | |
|---|---|---|
| x:=1 | ⇝ | $x_v:=1$ |
| a:=x | ⇝ | $a:=x_v$ |
| flush x | ⇝ | $x_s:=x_v$ |
| unbuffering (***non-det*** times) | | |
| (buffer→pbuffer) | ⇝ | $x_s:=x_v$ |
| (pbuffer→memory) | ⇝ | $\overline{x_p:=x_s}$ |

# Challenge #3: ***Weak*** Asynchronous Persists

```
// x=y=0

x := 1;
☛ flushopt x;
y := 1;
```

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

!! Weak explicit persists behave ***asynchronously***

# Challenge #3: ***Weak*** Asynchronous Persists

```
// x=y=0

x := 1;
y := 1;
☛ flushopt x;
```

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

!! Weak explicit persists behave ***asynchronously***

!! Weak explicit persists may be ***reordered***

# Challenge #3: ***Weak*** Asynchronous Persists

```
// x=y=0

x := 1;
y := 1;
☛ flushopt x;
```

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

!! Weak explicit persists behave ***asynchronously***

!! Weak explicit persists may be ***reordered***

# Challenge #3: ***Weak*** Asynchronous Persists

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

## ***Problem***

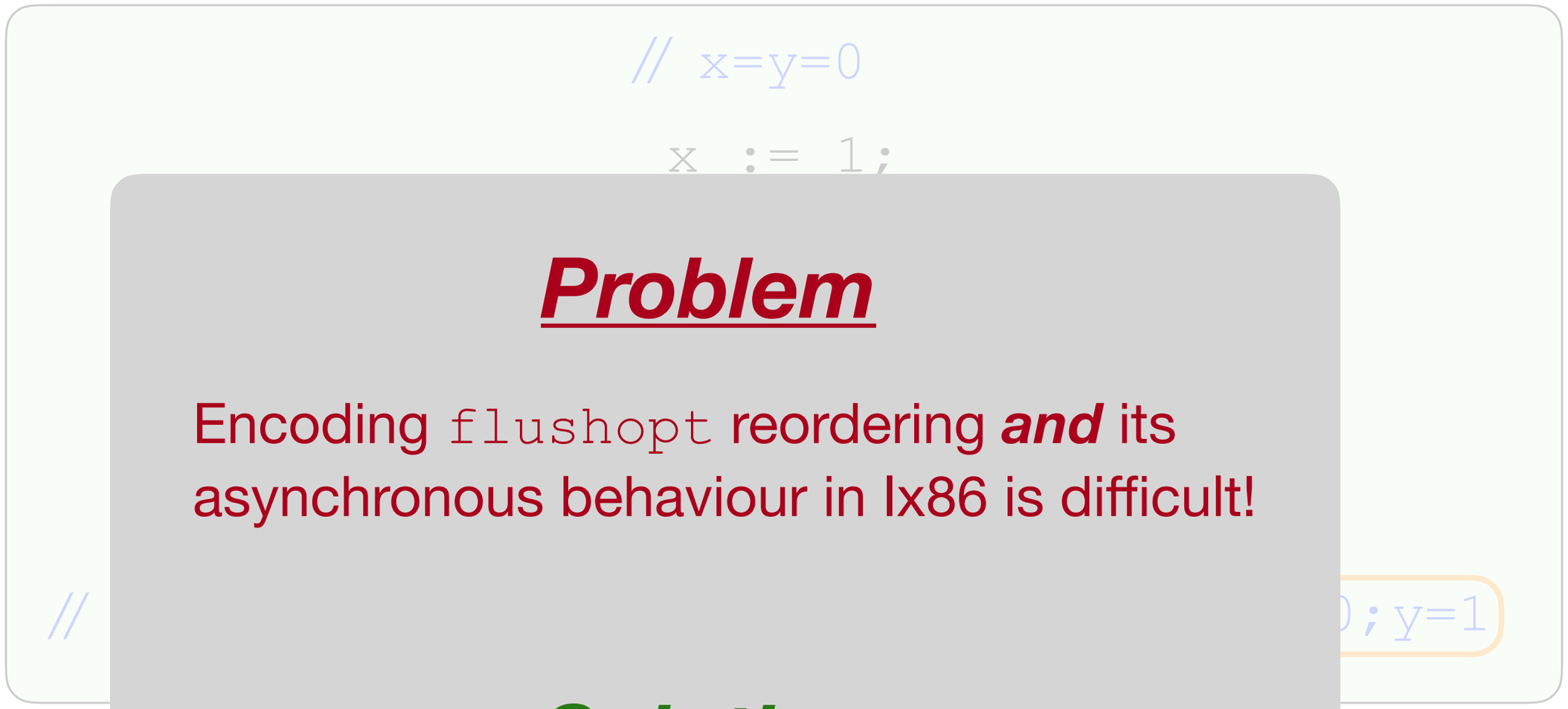
Encoding `flushopt` reordering ***and*** its asynchronous behaviour in Ix86 is difficult!

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

!! sly

!!

# Challenge #3: ***Weak*** Asynchronous Persists




## ***Problem***

Encoding `flushopt` reordering ***and*** its asynchronous behaviour in Ix86 is difficult!

## ***Solution***

Rewrite programs with `flushopt` to use `flush` instead!

# Eliminating `flushopt` Instructions

- **`flushopt`**
  - ➡ ***optimised*** variant of flush -- better performance
  - ➡ typically used in a particular ***programming pattern*** (epoch persistency)
  - ➡ ***transformation*** mechanism: rewrite programs with `flushopt`
    to ***equivalent*** ones with `flush`

# Conclusions

# Conclusions

- ***lx86***: instrumented x86 semantics
  - ➡ reduces Px86 to x86
  - ➡ ***translation*** from Px86 to lx86
  - ➡ addresses challenge #1 and challenge #2

# Conclusions

- ***Ix86***: instrumented x86 semantics
  - ➡ reduces Px86 to x86
  - ➡ ***translation*** from Px86 to Ix86
  - ➡ addresses challenge #1 and challenge #2
- ***Transformation Mechanism***
  - ➡ Convert programs with `flushopt` to ***equivalent*** ones with `flush`
  - ➡ addresses challenge #3

# Conclusions

- ***Ix86***: instrumented x86 semantics
  - ➡ reduces Px86 to x86
  - ➡ ***translation*** from Px86 to Ix86
  - ➡ addresses challenge #1 and challenge #2
- ***Transformation Mechanism***
  - ➡ Convert programs with `flushopt` to ***equivalent*** ones with `flush`
  - ➡ addresses challenge #3
- ***POG***: the ***first*** program logic for persistency
  - ➡ built over Ix86
  - ➡ several **examples** of persistent reasoning

# Conclusions

- ***Ix86***: instrumented x86 semantics
  - ➡ reduces Px86 to x86
  - ➡ ***translation*** from Px86 to Ix86
  - ➡ addresses challenge #1 and challenge #2
- ***Transformation Mechanism***
  - ➡ Convert programs with `flushopt` to ***equivalent*** ones with `flush`
  - ➡ addresses challenge #3
- ***POG***: the ***first*** program logic for persistency
  - ➡ built over Ix86
  - ➡ several **examples** of persistent reasoning

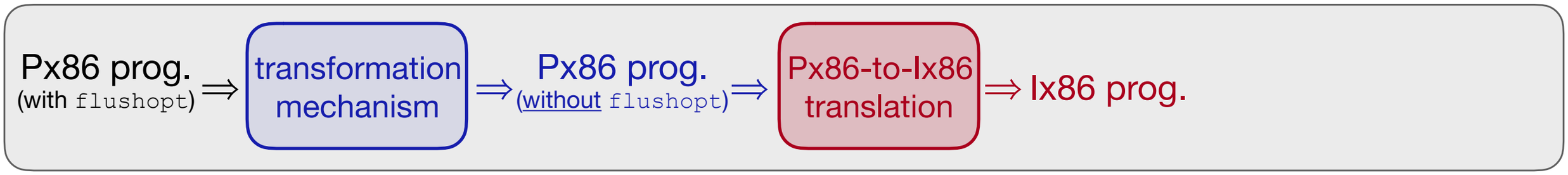
Px86 prog.
(with `flushopt`)

# Conclusions

- ***Ix86***: instrumented x86 semantics
  - ➡ reduces Px86 to x86
  - ➡ ***translation*** from Px86 to Ix86
  - ➡ addresses challenge #1 and challenge #2
- ***Transformation Mechanism***
  - ➡ Convert programs with `flushopt` to ***equivalent*** ones with `flush`
  - ➡ addresses challenge #3
- ***POG***: the ***first*** program logic for persistency
  - ➡ built over Ix86
  - ➡ several **examples** of persistent reasoning

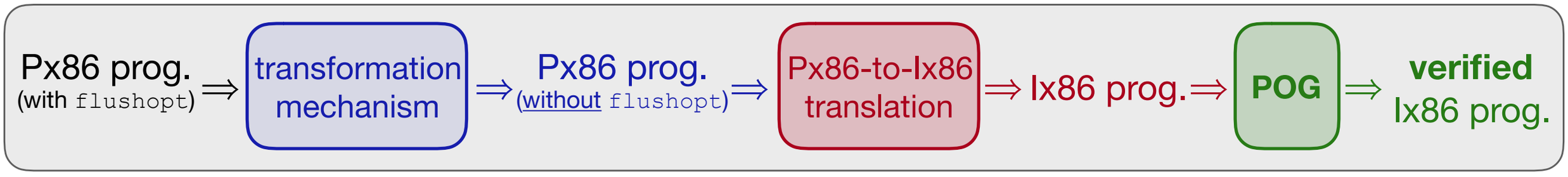
# Conclusions

- ***Ix86***: instrumented x86 semantics
  - ➡ reduces Px86 to x86
  - ➡ ***translation*** from Px86 to Ix86
  - ➡ addresses challenge #1 and challenge #2
- ***Transformation Mechanism***
  - ➡ Convert programs with `flushopt` to ***equivalent*** ones with `flush`
  - ➡ addresses challenge #3
- ***POG***: the ***first*** program logic for persistency
  - ➡ built over Ix86
  - ➡ several **examples** of persistent reasoning



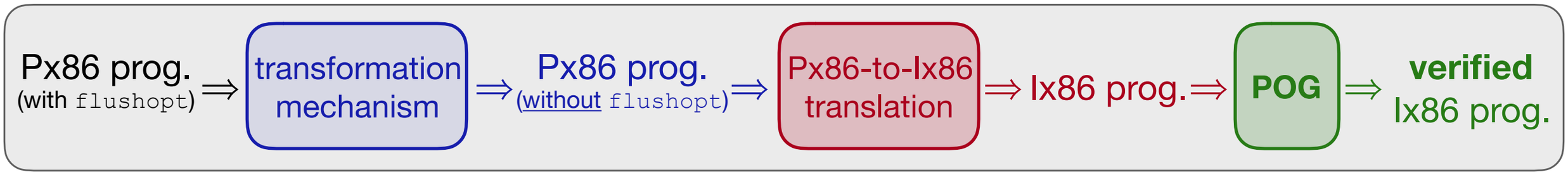
# Conclusions

- ***Ix86***: instrumented x86 semantics
  - ➡ reduces Px86 to x86
  - ➡ ***translation*** from Px86 to Ix86
  - ➡ addresses challenge #1 and challenge #2
- ***Transformation Mechanism***
  - ➡ Convert programs with `flushopt` to ***equivalent*** ones with `flush`
  - ➡ addresses challenge #3
- ***POG***: the ***first*** program logic for persistency
  - ➡ built over Ix86
  - ➡ several **examples** of persistent reasoning



# Conclusions

- ***Ix86***: instrumented x86 semantics
  - ➡ reduces Px86 to x86
  - ➡ ***translation*** from Px86 to Ix86
  - ➡ addresses challenge #1 and challenge #2
- ***Transformation Mechanism***
  - ➡ Convert programs with `flushopt` to ***equivalent*** ones with `flush`
  - ➡ addresses challenge #3
- ***POG***: the ***first*** program logic for persistency
  - ➡ built over Ix86
  - ➡ several **examples** of persistent reasoning




## Thank You for Listening!


azalea@imperial.ac.uk

SoundAndComplete.org

@azalearaad