Verifying Programs In Weak Memory Models With Persistency

Prakash Saivasan
The Institute of Mathematical Sciences

Joint work with
Parosh Aziz Abdulla
Mohamed Faouzi Atig
Ahmed Bouajjani
K. Narayan Kumar

Verifying Programs In Weak Memory Models With Persistency

Verifying Programs In Weak Memory Models With Persistency

Outline

Verifying Programs In Weak Memory Models With Persistency

Outline

Verifying Programs In Weak Memory Models With Persistency

Outline

- Concurrent Programs

Verifying Programs In Weak Memory Models With Persistency

Outline

- Concurrent Programs

Verifying Programs In Weak Memory Models With Persistency

Outline

- Concurrent Programs
- Weak Memory models

Verifying Programs In Weak Memory Models With Persistency

Outline

- Concurrent Programs

- Weak Memory models

Verifying Programs In Weak Memory Models With Persistency

Outline

- Concurrent Programs
- Weak Memory models
- Persistency

Verifying Programs In Weak Memory Models With Persistency

Outline

- Concurrent Programs
- Weak Memory models
- Persistency

Verifying Programs In Weak Memory Models With Persistency

Outline

- Concurrent Programs
- Weak Memory models
- Persistency
- Verification

Verifying Programs In Weak Memory Models With Persistency

Outline

- Concurrent Programs
- Weak Memory models
- Persistency

Intel x86 model

- Verification

Verifying Programs In Weak Memory Models With Persistency

Outline

- Concurrent Programs
- Weak Memory models
- Persistency
- Verification

Intel x86 model

Extending Intel-x86 Consistency and Persistency

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

AZALEA RAAD, Imperial College London, United Kingdom
LUC MARANGET, Inria, France
VIKTOR VAFEIADIS, MPI-SWS, Germany

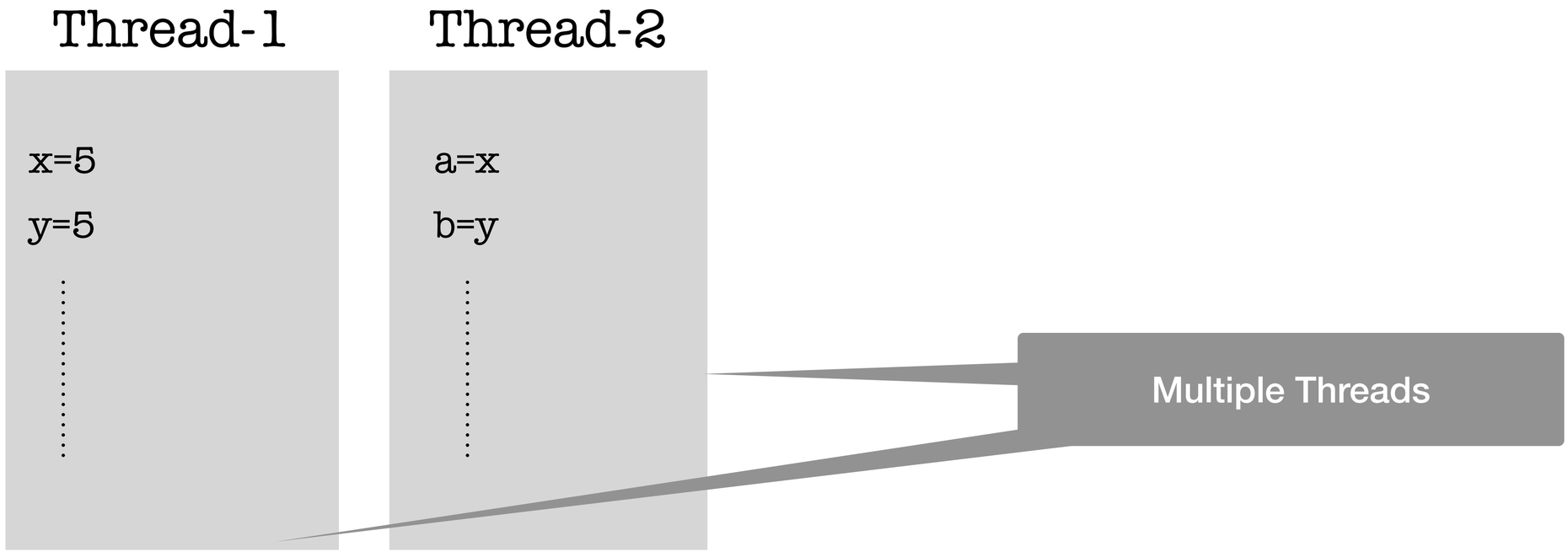
POPL 2022

CONCURRENT PROGRAMS

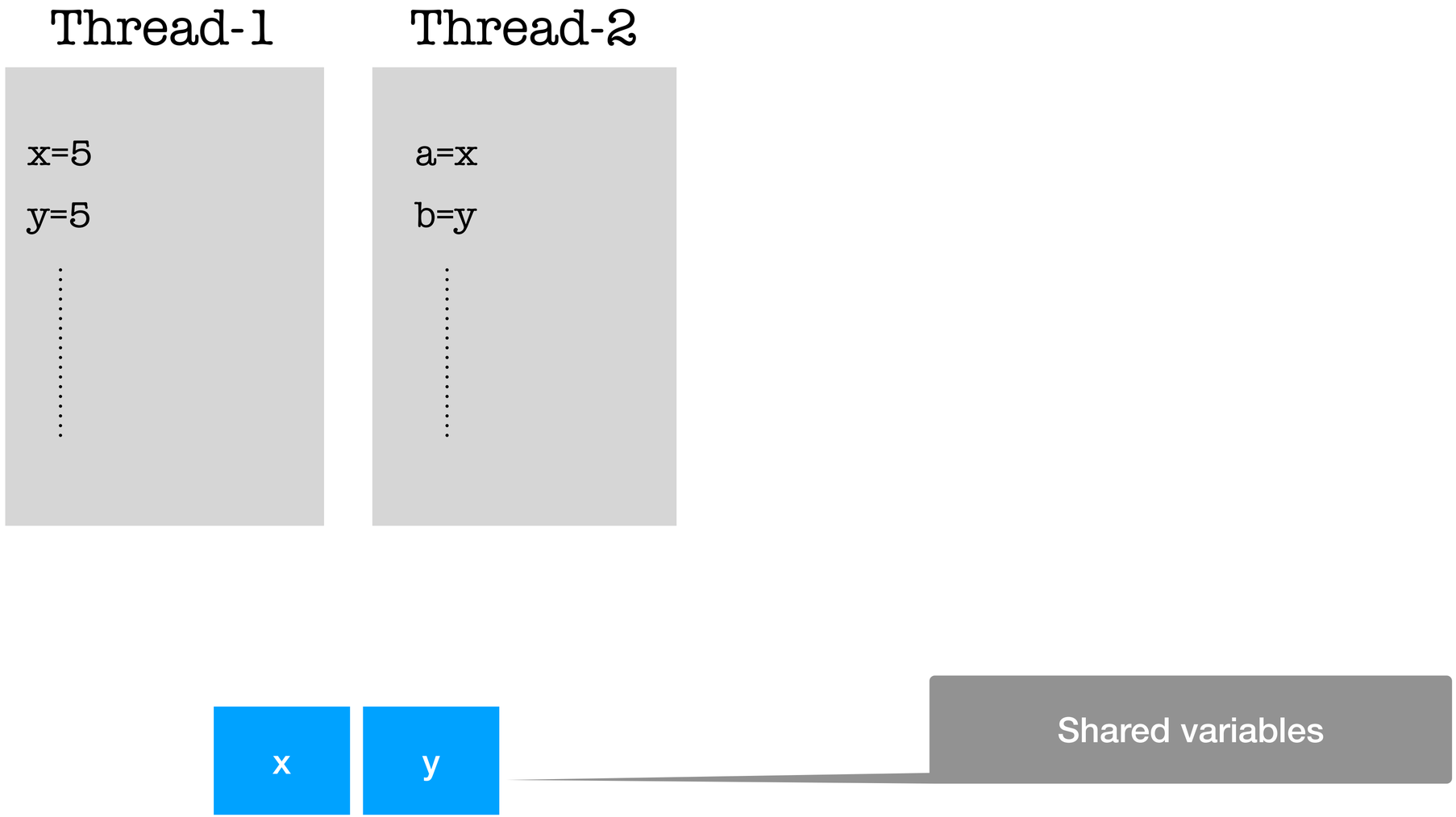
In the concurrent world, imperative is a wrong default - Tim Sweeny

Concurrent Programs

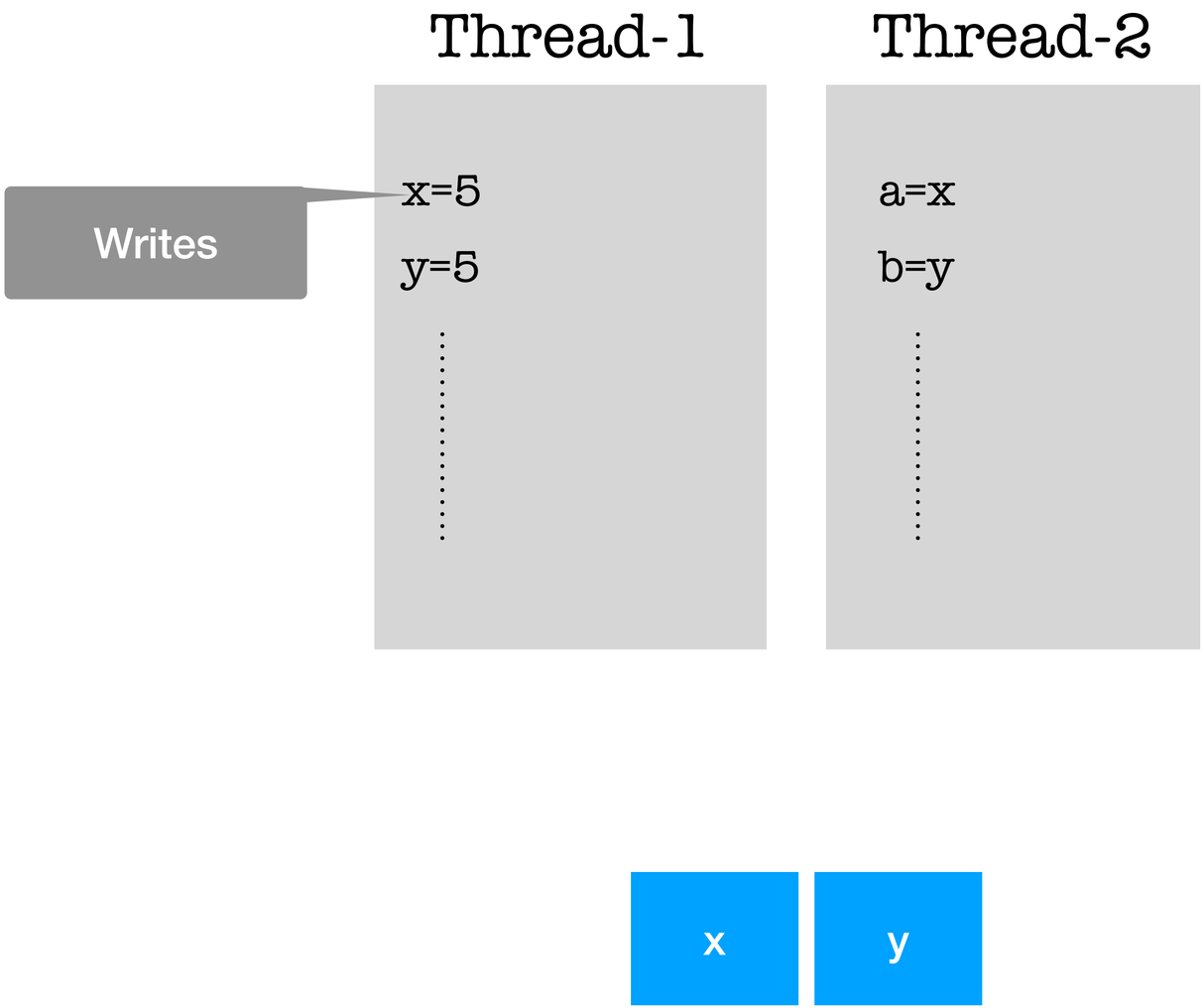
Concurrent Programs



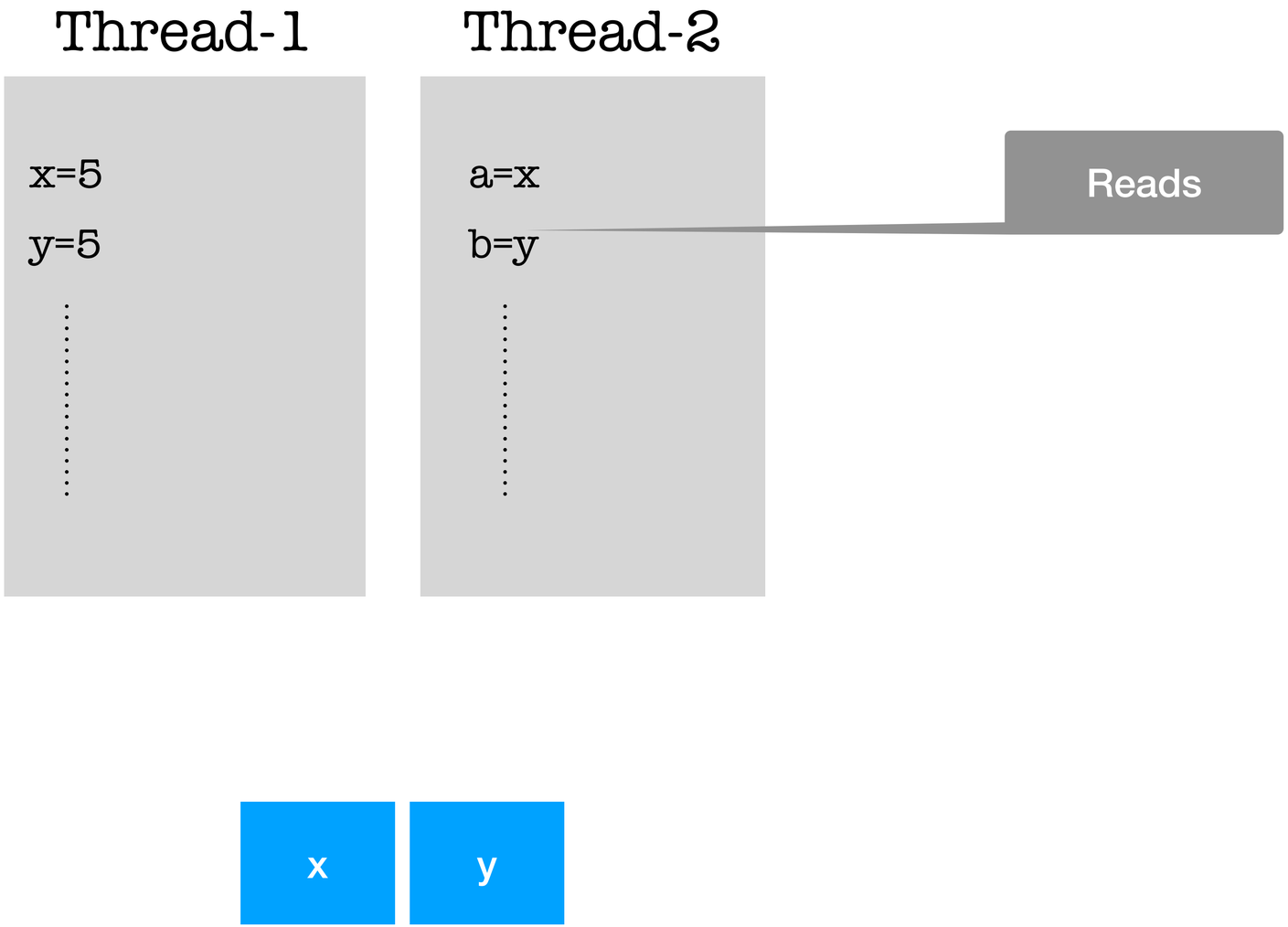
Concurrent Programs



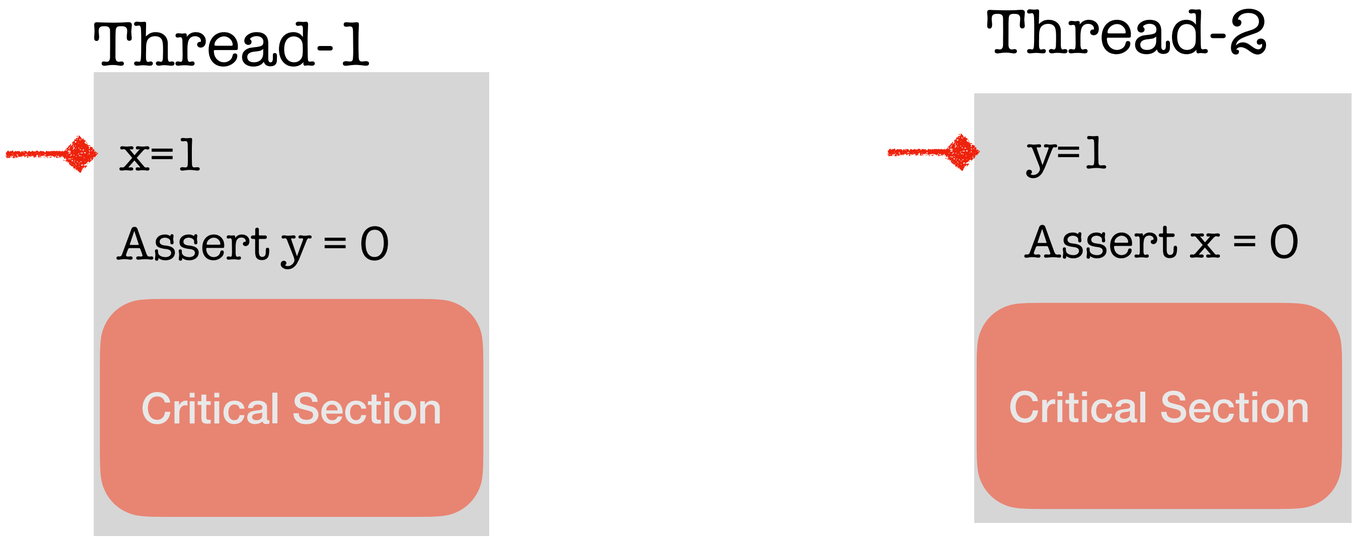
Concurrent Programs

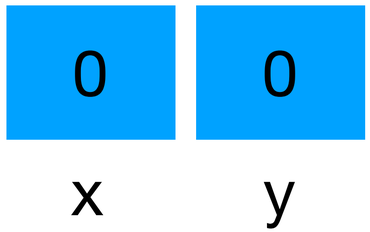


Concurrent Programs

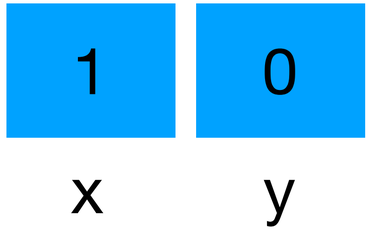


Dekker's Mutual Exclusion

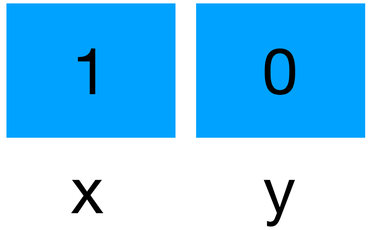




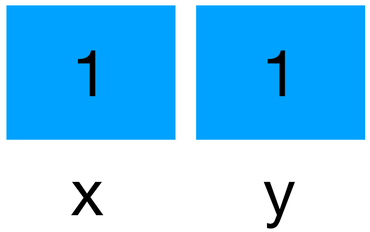
Dekker's Mutual Exclusion



Dekker's Mutual Exclusion



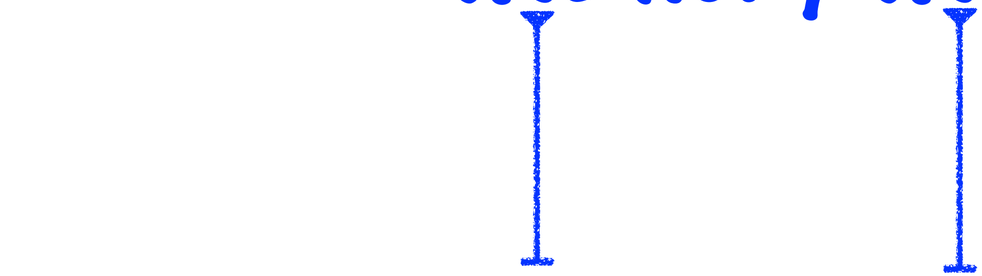
Dekker's Mutual Exclusion



WEAK MEMORY MODELS

don't communicate by sharing memory; share memory by communicating

Memory Models



Sequential Consistency

Weak Consistency

Memory Models

Sequential Consistency

Operations are atomic

Weak Consistency

Memory Models

Sequential Consistency

Operations are atomic

Weak Consistency

Operations can be re-ordered

Memory Models

Sequential Consistency

Operations are atomic

Weak Consistency

Operations can be re-ordered

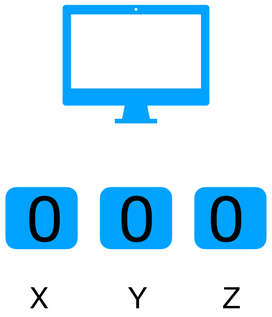
TSO, PSO, EX86

Sequential Consistency

Sequential Consistency

X Y Z

Sequential Consistency

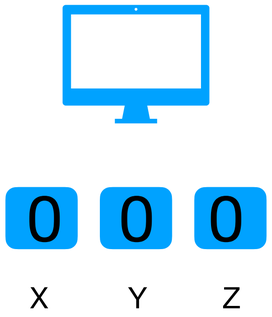


Instructions are sequential and immediate

Sequential Consistency

Instructions

Wr(x,1) Rd(y) RMW(z,0,2)



Instructions are sequential and immediate

Sequential Consistency

Instructions

Wr(x,1)

Rd(y)

RMW(z,0,2)

0 0 0

X Y Z

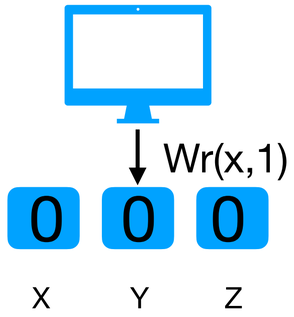
Sequential Consistency

Instructions

Wr(x,1)

Rd(y)

RMW(z,0,2)



Write modifies the value of a variable

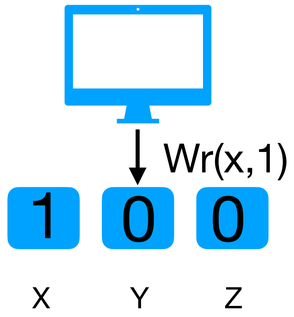
Sequential Consistency

Instructions

Wr(x,1)

Rd(y)

RMW(z,0,2)

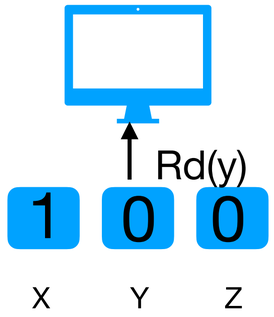


Write modifies the value of a variable

Sequential Consistency

Instructions

Wr(x,1) | Rd(y) | RMW(z,0,2)



Read fetches value of a variable

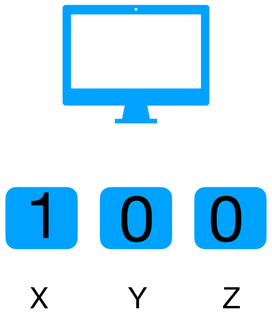
Sequential Consistency

Instructions

Wr(x,1)

Rd(y)

RMW(z,0,2)



Rmw tests and sets a variable

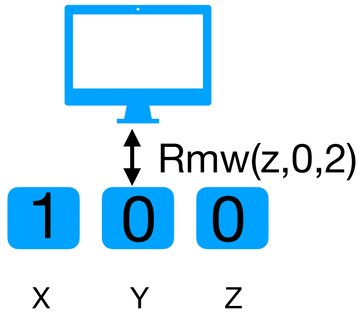
Sequential Consistency

Instructions

Wr(x,1)

Rd(y)

RMW(z,0,2)



Rmw tests and sets a variable

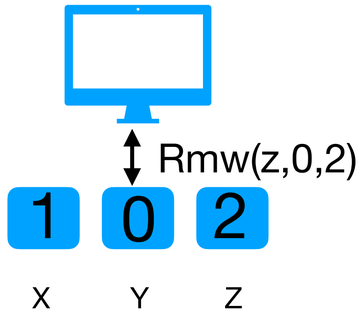
Sequential Consistency

Instructions

Wr(x,1)

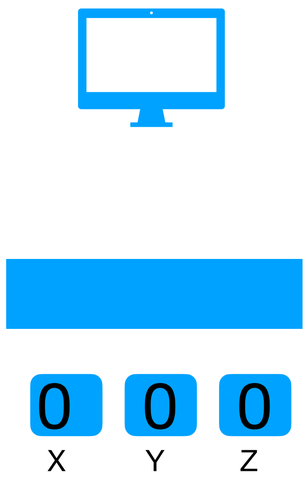
Rd(y)

RMW(z,0,2)

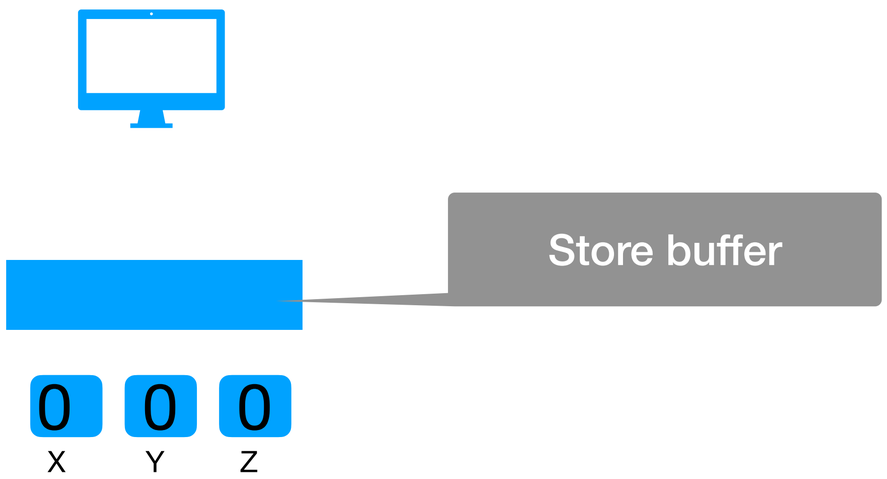


Rmw tests and sets a variable

Total Store Order



Total Store Order

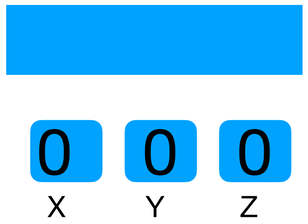


Writes are buffered, reads are immediate

Total Store Order

Instructions

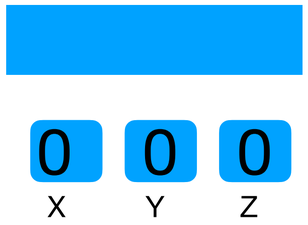
Wr(x,d) Rd(x) Mf RMW(x,b,d)



Total Store Order

Instructions

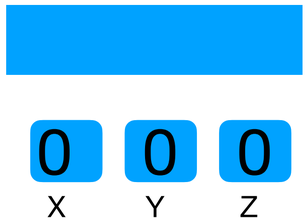
Wr(x,d) Rd(x) Mf RMW(x,b,d)



Total Store Order

Instructions

Wr(x,d) Rd(x) Mf RMW(x,b,d)

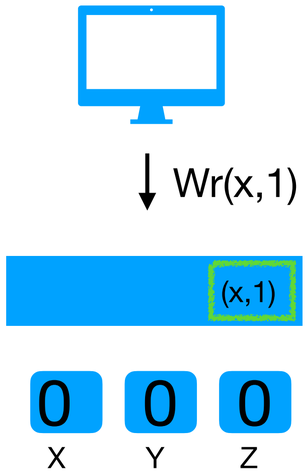


Writes are buffered

Total Store Order

Instructions

Wr(x,d) Rd(x) Mf RMW(x,b,d)



Writes are buffered

Total Store Order

Instructions

Wr(x,d) Rd(x) Mf RMW(x,b,d)

Wr(y,2)

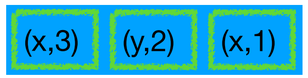
Writes are buffered

Total Store Order

Instructions

Wr(x,d) Rd(x) Mf RMW(x,b,d)

Wr(x,3)

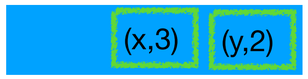


Writes are buffered

Total Store Order

Instructions

Wr(x,d) Rd(x) Mf RMW(x,b,d)

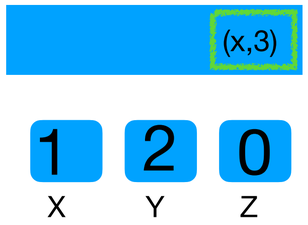


Propagated to memory non-deterministically

Total Store Order

Instructions

Wr(x,d) Rd(x) Mf RMW(x,b,d)

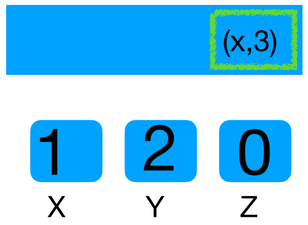


Propagated to memory non-deterministically

Total Store Order

Instructions

Wr(x,d) | Rd(x) | Mf | RMW(x,b,d)



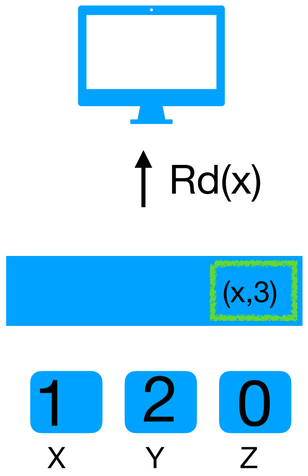
Reads are either from buffer or memory

In that order!!

Total Store Order

Instructions

Wr(x,d) | Rd(x) | Mf | RMW(x,b,d)

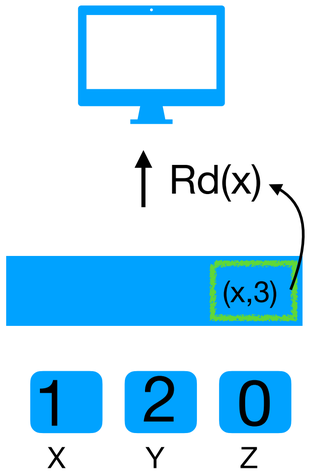


Reads are either from buffer or memory

Total Store Order

Instructions

Wr(x,d) | Rd(x) | Mf | RMW(x,b,d)

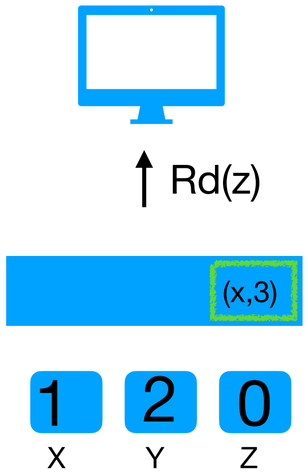


Reads are either from buffer or memory

Total Store Order

Instructions

Wr(x,d) | Rd(x) | Mf | RMW(x,b,d)

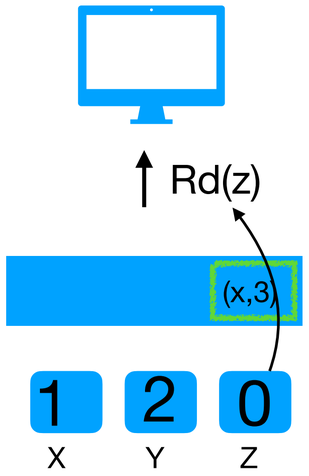


Reads are either from buffer or memory

Total Store Order

Instructions

Wr(x,d) | Rd(x) | Mf | RMW(x,b,d)

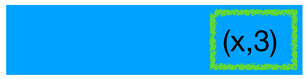


Reads are either from buffer or memory

Total Store Order

Instructions

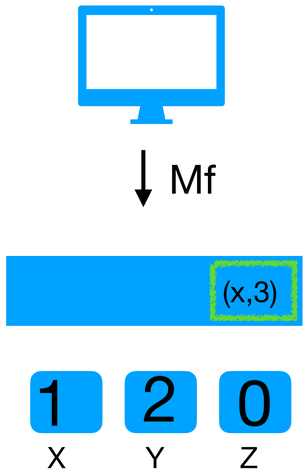
Wr(x,d) Rd(x) Mf RMW(x,b,d)



Total Store Order

Instructions

Wr(x,d) Rd(x) Mf RMW(x,b,d)

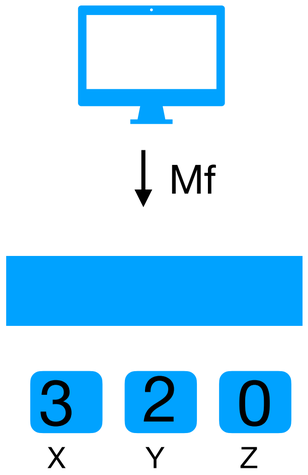


Memory fence ensures buffer is empty

Total Store Order

Instructions

Wr(x,d) Rd(x) Mf RMW(x,b,d)

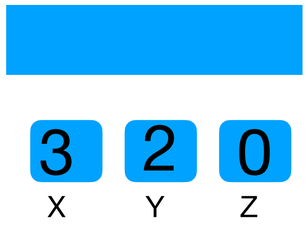


Memory fence ensures buffer is empty

Total Store Order

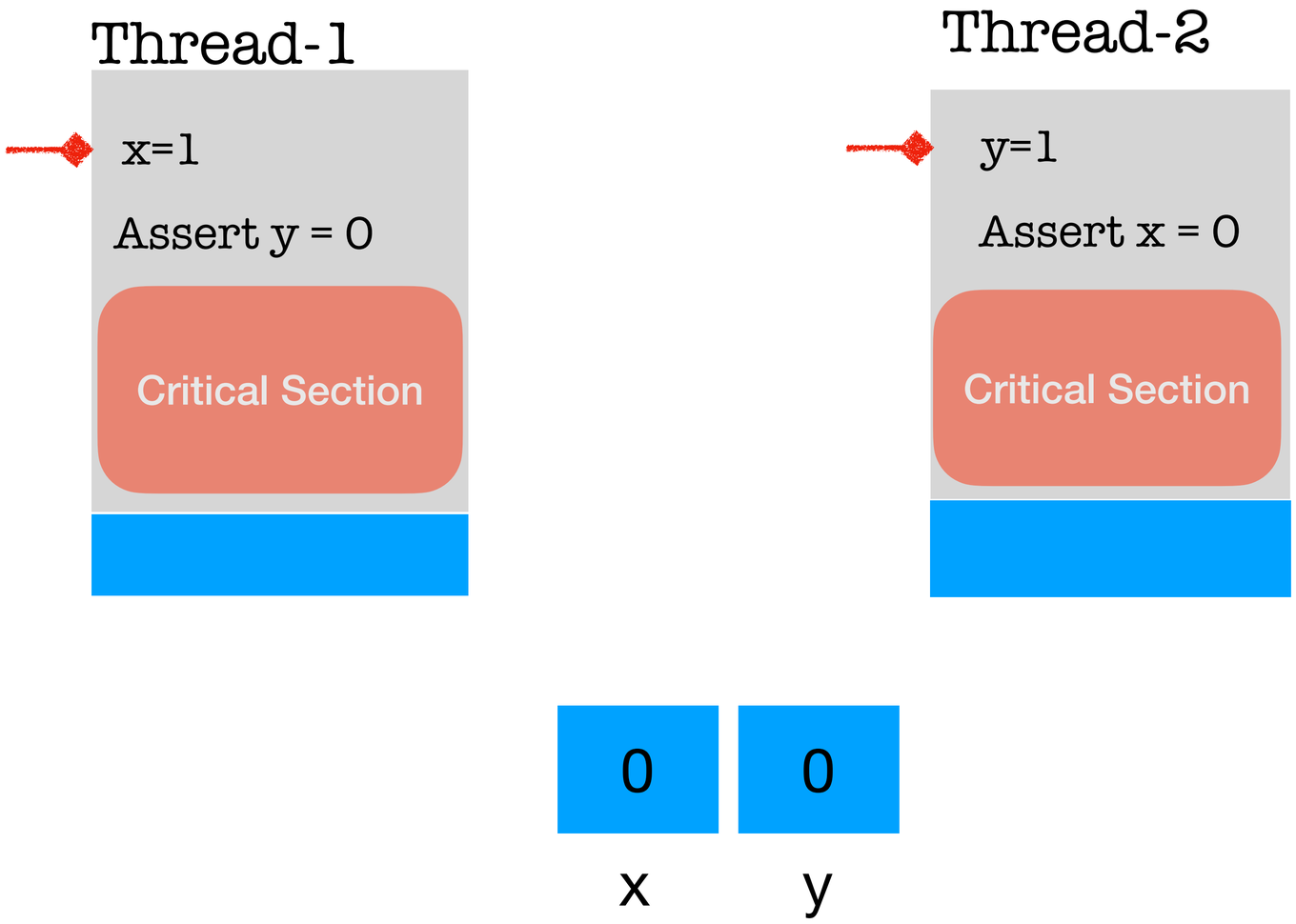
Instructions

Wr(x,d) Rd(x) Mf RMW(x,b,d)

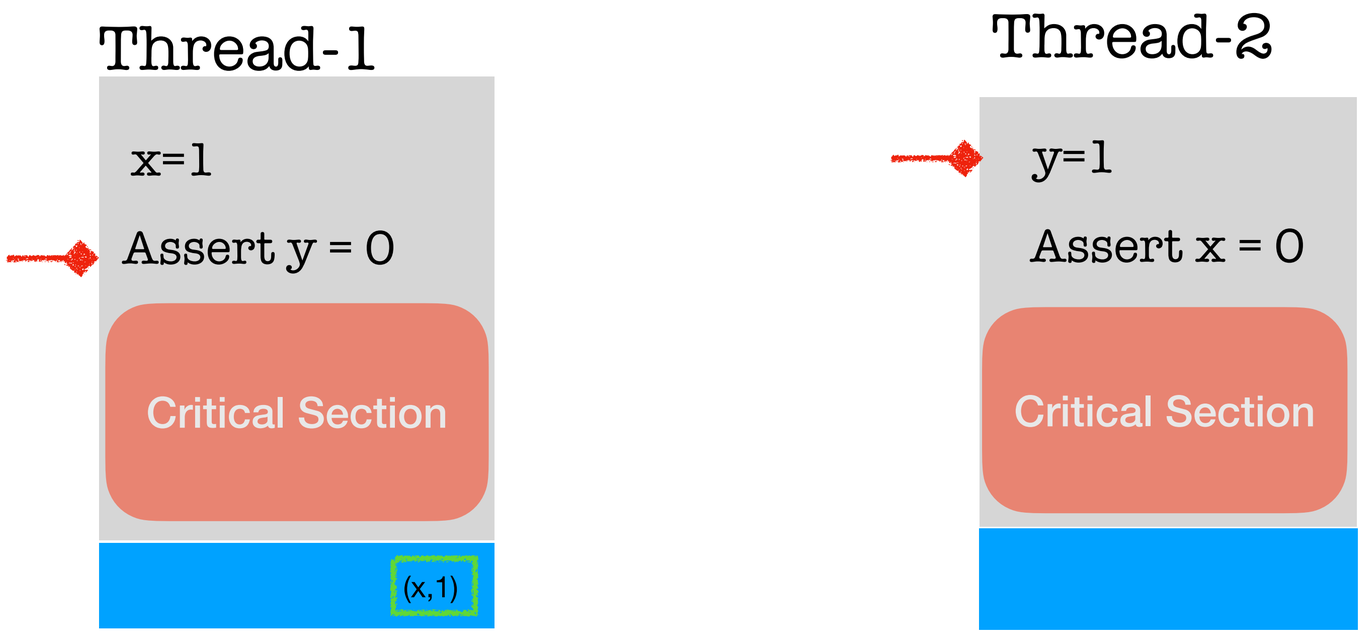


Memory fence ensures buffer is empty

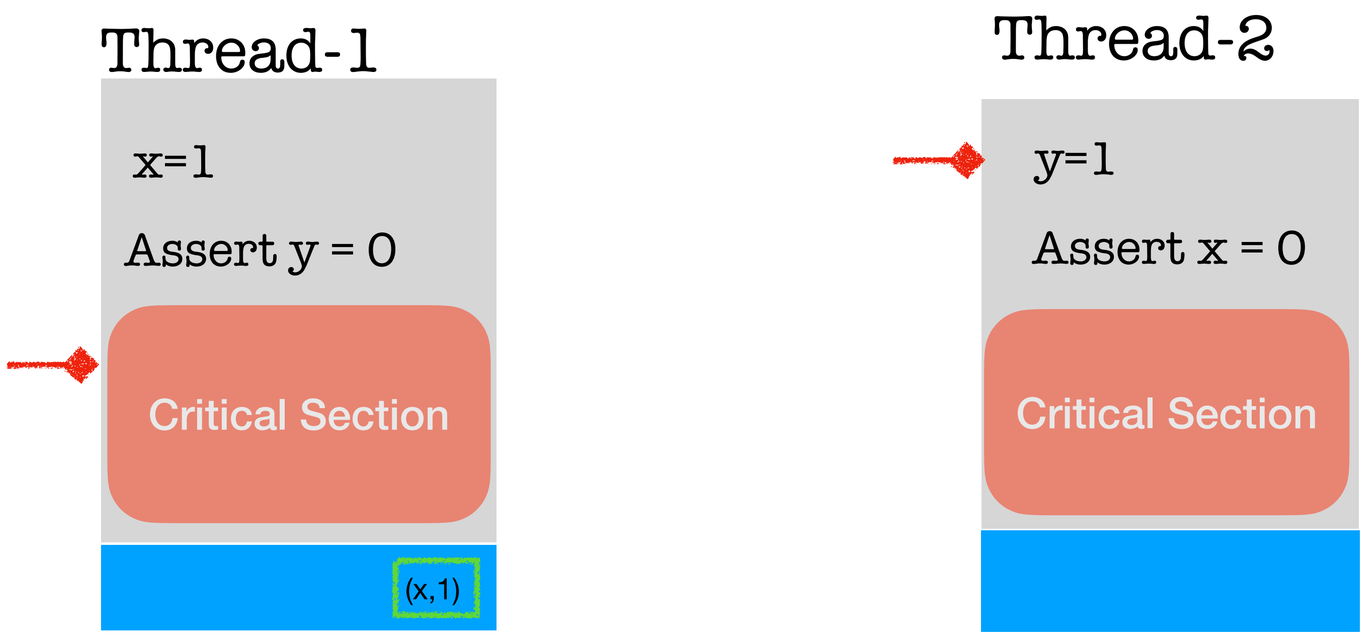
Dekker's Mutual Exclusion



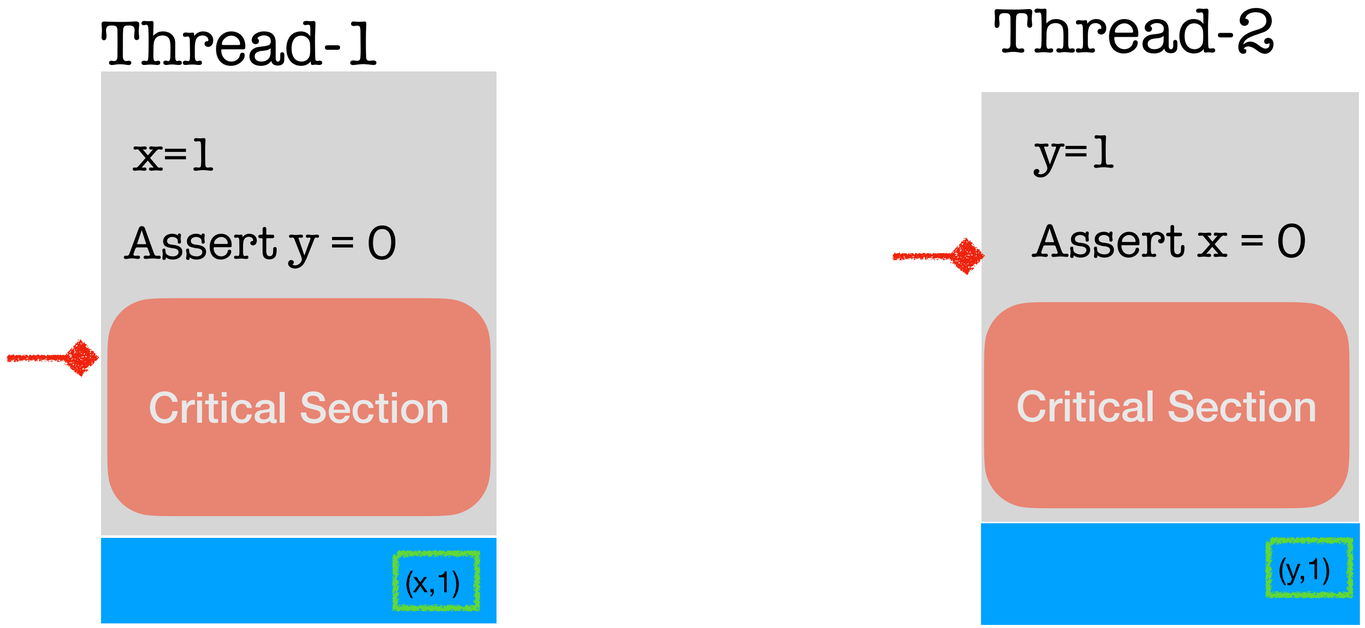
Dekker's Mutual Exclusion



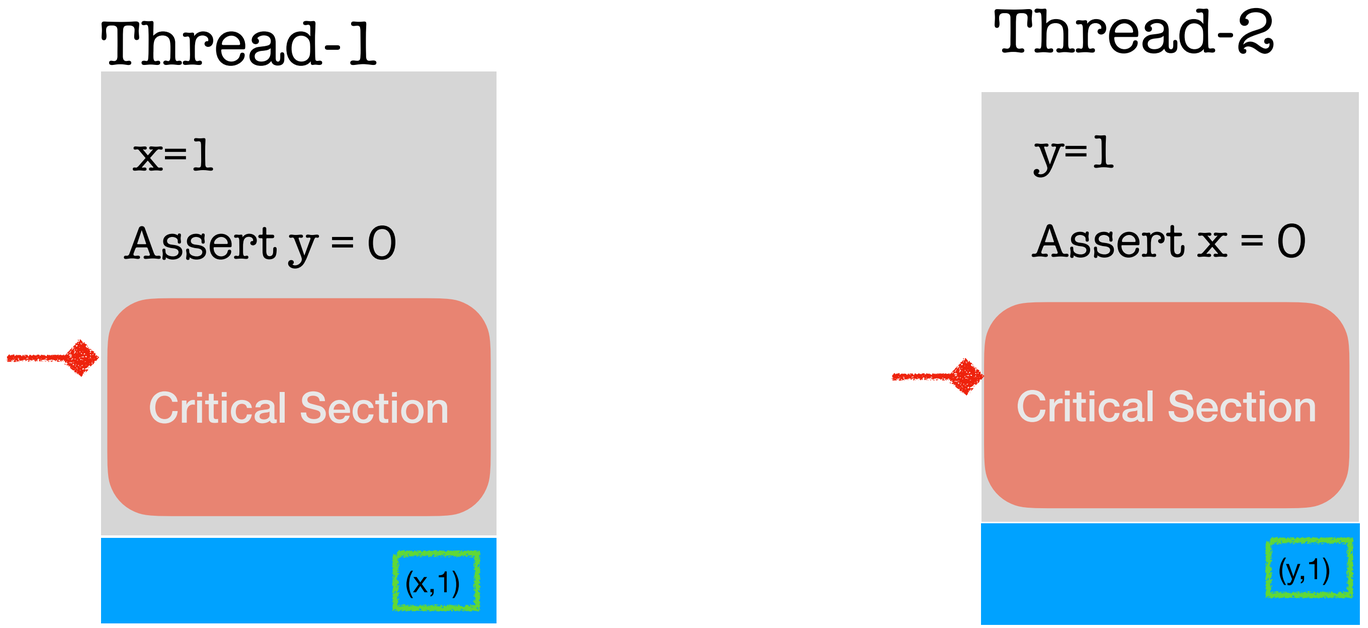
Dekker's Mutual Exclusion



Dekker's Mutual Exclusion



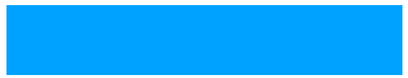
Dekker's Mutual Exclusion



Partial Store Order

Instructions

Wr(x,d) Sf Rd(x) Mf Rmw(x,b,d)



0 0

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)

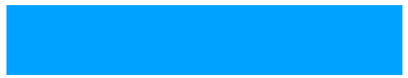
0 0

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)

Wr(x,1)



0 0

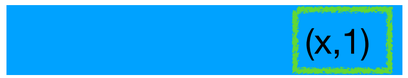
Writes are buffered

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)

Wr(x,1)



0 0

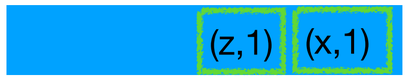
Writes are buffered

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)

Wr(z,1)



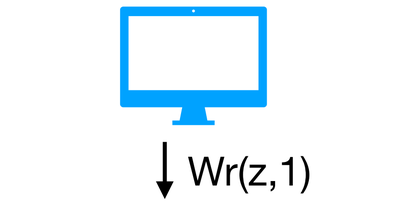
0 0

Writes are buffered

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)



0 0 0

X Y Z

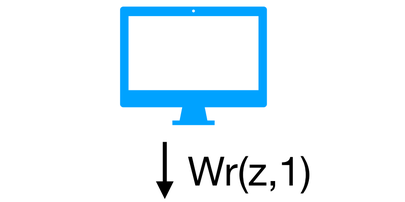
0 0

Propagated to memory non-deterministically

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)



0 0 0

X Y Z

0 0

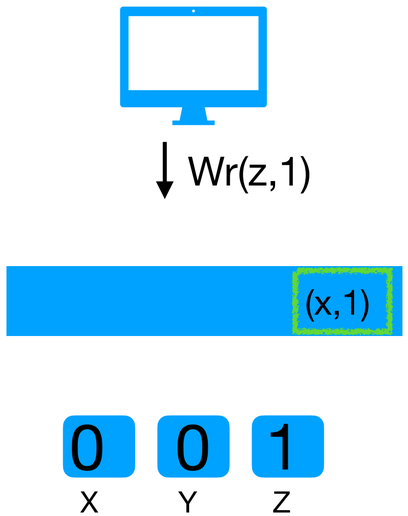
Propagated to memory non-deterministically

Reorders writes to different variables

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)



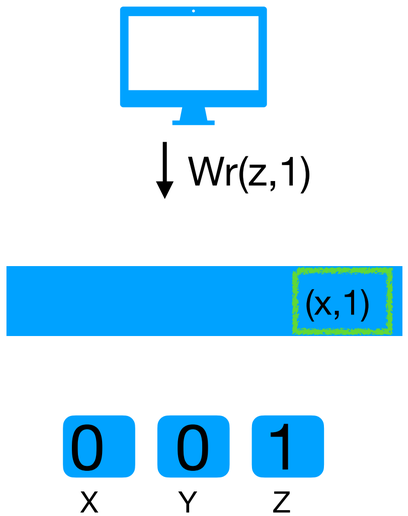
Propagated to memory non-deterministically

Reorders writes to different variables

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)

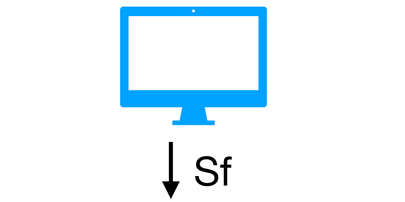


Store fence restricts re-ordering between writes

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)



0 0

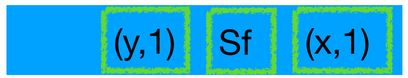
Store fence restricts re-ordering between writes

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)

Wr(y,1)



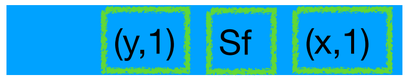
0 0

Store fence restricts re-ordering between writes

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)



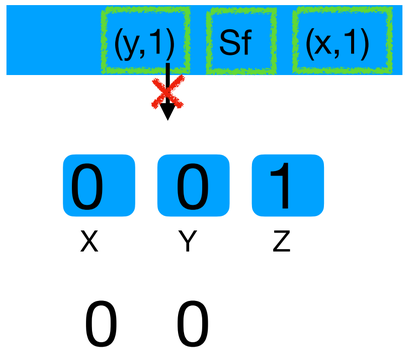
0 0

Store fence restricts re-ordering between writes

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)

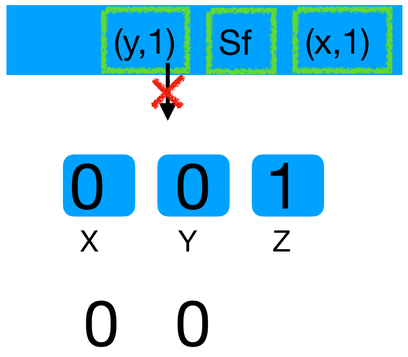


Store fence restricts re-ordering between writes

Partial Store Order

Instructions

Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)

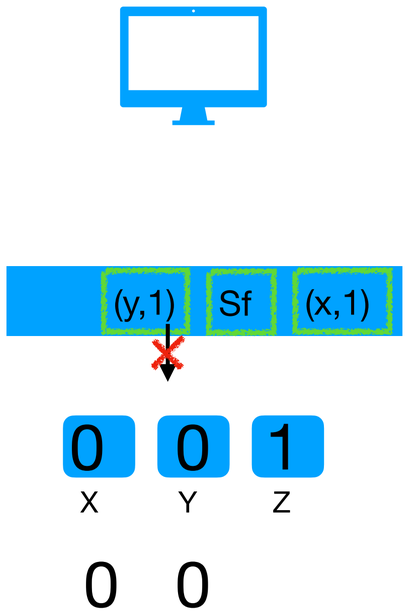


Store fence restricts re-ordering between writes

Partial Store Order

Instructions

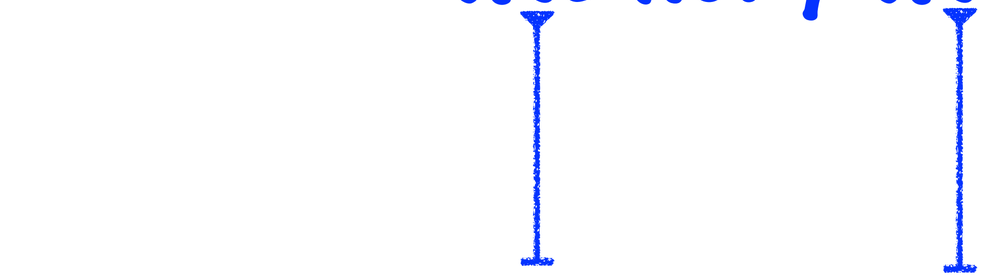
Wr(x,d) | Sf | Rd(x) | Mf | Rmw(x,b,d)



0 0

Store fence restricts re-ordering between writes

Memory Models



Sequential Consistency

Weak Consistency

Memory Models

Sequential Consistency

Atomic operations

Weak Consistency

Memory Models

Sequential Consistency

Atomic operations

Weak Consistency

Operations can be re-ordered

Memory Models

Sequential Consistency

Atomic operations

\+ Simple and intuitive

Weak Consistency

Operations can be re-ordered

Memory Models

Sequential Consistency

Atomic operations

+ Simple and intuitive

- Expensive

Weak Consistency

Operations can be re-ordered

Memory Models

Sequential Consistency

Atomic operations

+ Simple and intuitive

- Expensive

Weak Consistency

Operations can be re-ordered

+ Optimised for efficiency

Memory Models

Sequential Consistency

Atomic operations

+ Simple and intuitive

- Expensive

Weak Consistency

Operations can be re-ordered

+ Optimised for efficiency

- Complicated

Memory Models

Sequential Consistency

Atomic operations

+ Simple and intuitive

- Expensive

Weak Consistency

Operations can be re-ordered

+ Optimised for efficiency

- Complicated

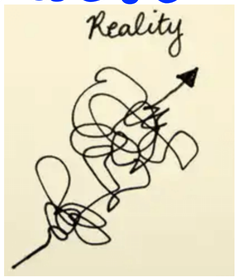
Memory Models

Sequential Consistency

Atomic operations

+ Simple and intuitive

- Expensive



Weak Consistency

Operations can be re-ordered

+ Optimised for efficiency

- Complicated

Extended x86 Explained

Extended x86 Explained

Instructions

Wb(x,d)

ntw(x,d)

Sf

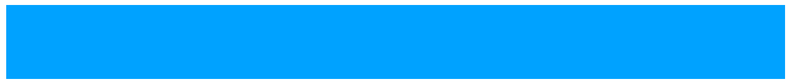
Fl(x)

Fo(x)

Rd(x)

Mf

Rmw(x,b,d)



Extended x86 Explained

Instructions

Wb(x,d)

ntw(x,d)

Sf

Fl(x)

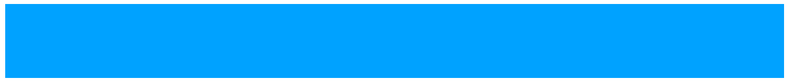
Fo(x)

Rd(x)

Mf

Rmw(x,b,d)

Writes are of type Ntw or Wb



Extended x86 Explained

Instructions

Wb(x,d)

ntw(x,d)

Sf

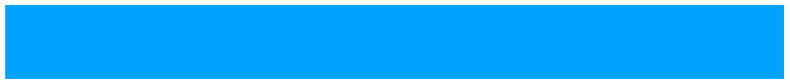
Fl(x)

Fo(x)

Rd(x)

Mf

Rmw(x,b,d)



Writes are of type Ntw or Wb

Original model has a complicated semantics with more kinds of writes.

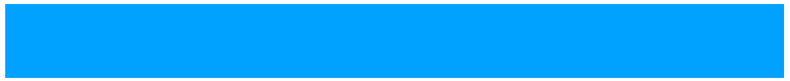
Extended x86 Explained

Instructions

Wb(x,d) ntw(x,d) Sf Fl(x) Fo(x)

Rd(x) Mf Rmw(x,b,d)

Writes are of type Ntw or Wb



Verification under Intel-x86 with Persistency

PAROSH ABDULLA, Uppsala University, Sweden
MOHAMED FAOUZI ATIG, Uppsala University, Sweden
AHMED BOUAJJANI, Université Paris Cité, France
K. NARAYAN KUMAR, Chennai Mathematical Institute and IRL ReLaX, India
PRAKASH SAIVASAN, Institute of Mathematical Sciences, HBNI and IRL ReLaX, India

For reachability, the two kinds of writes are sufficient

Extended x86 Explained

Instructions

Wb(x,d)

ntw(x,d)

Sf

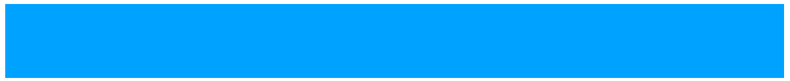
Fl(x)

Fo(x)

Rd(x)

Mf

Rmw(x,b,d)



Both types of writes are stored in buffer

Extended x86 Explained

Instructions

Wb(x,d)

ntw(x,d)

Sf

Fl(x)

Fo(x)

Rd(x)

Mf

Rmw(x,b,d)

0 0 0

X Y Z

Both types of writes are stored in buffer

Extended x86 Explained

Instructions

Wb(x,d) ntw(x,d) Sf Fl(x) Fo(x)

Rd(x) Mf Rmw(x,b,d)

Ntw writes re-order with writes of other variables

Extended x86 Explained

Instructions

Wb(x,d)

ntw(x,d)

Sf

Fl(x)

Fo(x)

Rd(x)

Mf

Rmw(x,b,d)

Wb writes do not re-order with each other

Extended x86 Explained

Instructions

Wb(x,d)

ntw(x,d)

Sf

Fl(x)

Fo(x)

Rd(x)

Mf

Rmw(x,b,d)

Wb writes do not re-order with each other

1 0 0

X Y Z

Extended x86 Explained

Instructions

Wb(x,d)

ntw(x,d)

Sf

Fl(x)

Fo(x)

Rd(x)

Mf

Rmw(x,b,d)

Wb writes do not re-order with each other

Extended x86 Explained

Instructions

Wb(x,d) ntw(x,d) Sf Fl(x) Fo(x)

Rd(x) Mf Rmw(x,b,d)

1 2 0

X Y Z

Fl, Sf disallows any re-orderings

Extended x86 Explained

Instructions

Wb(x,d) ntw(x,d) Sf Fl(x) Fo(x)

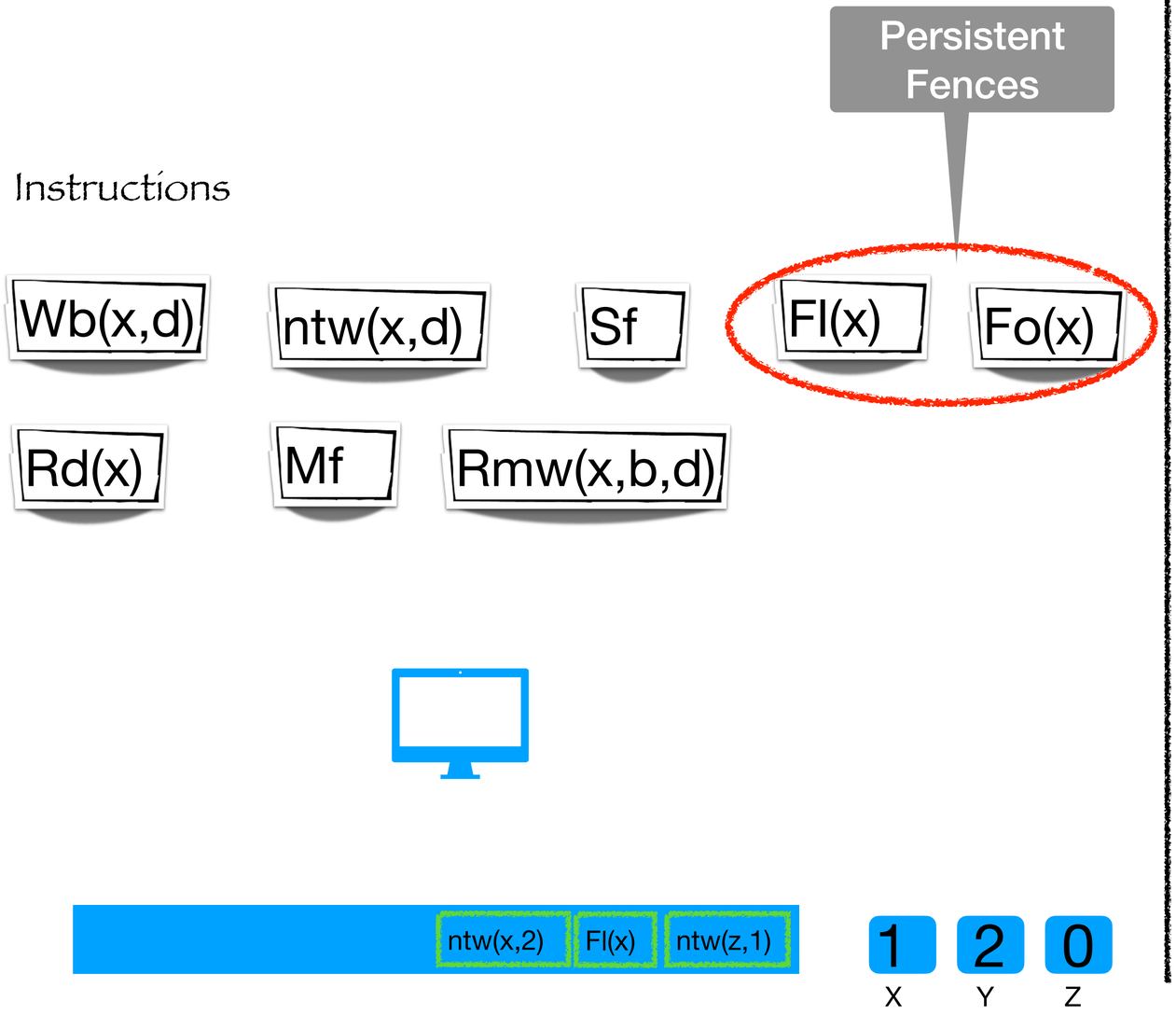
Rd(x) Mf Rmw(x,b,d)

1 2 0

X Y Z

Fo(x) cannot re-order with Sf and earlier writes to x

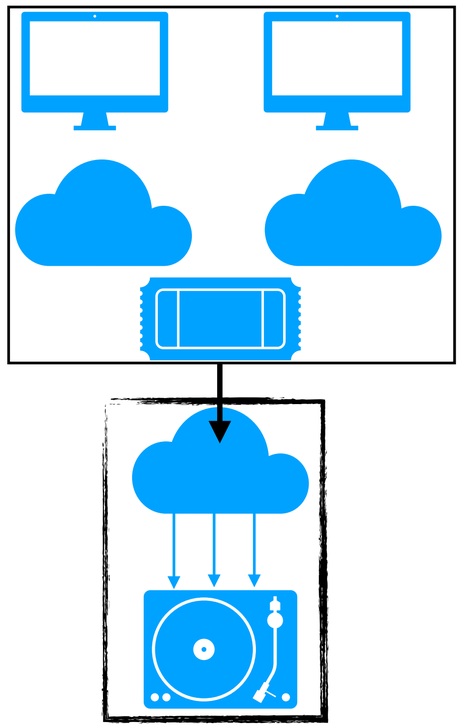
Extended x86 Explained



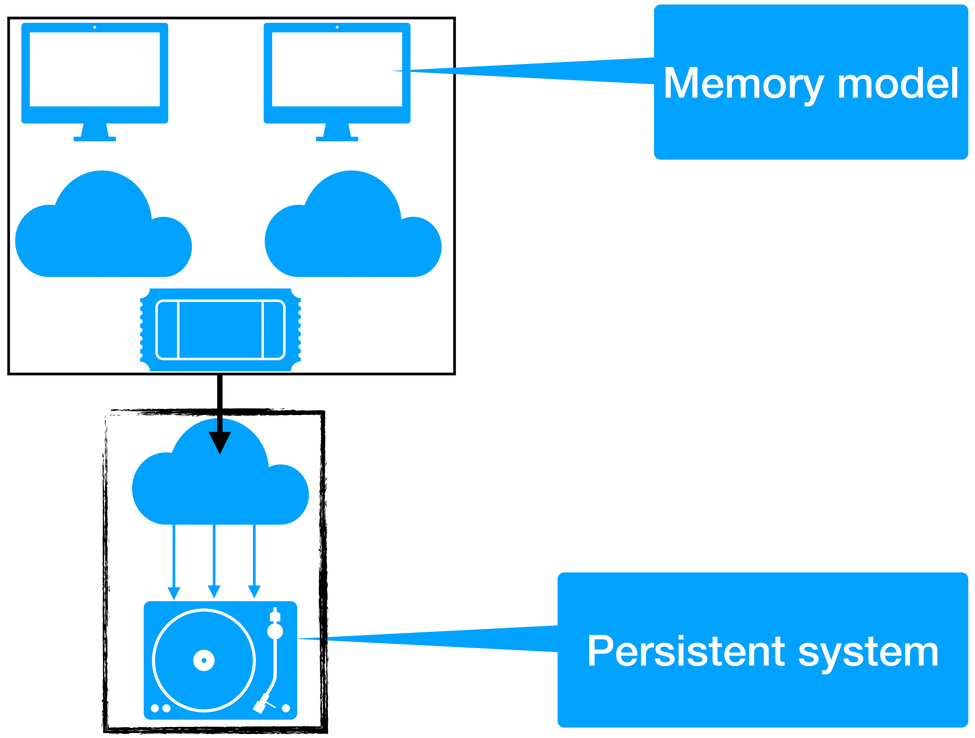
PERSISTENCY

Energy and persistence conquer all things - Benjamin Franklin

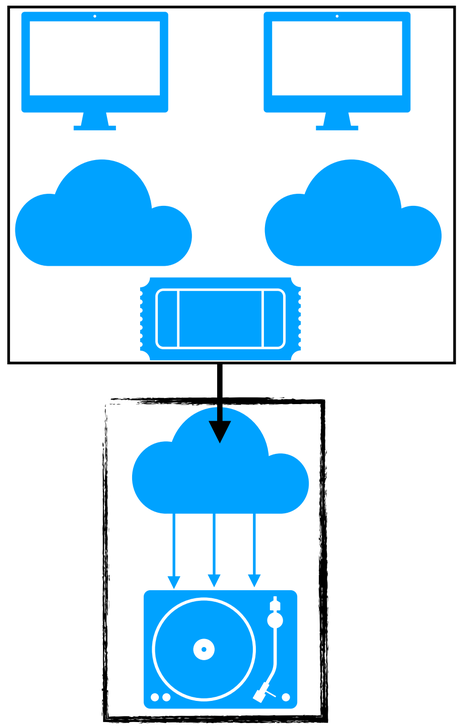
Concurrent Memory Systems + Persistency



Concurrent Memory Systems + Persistency

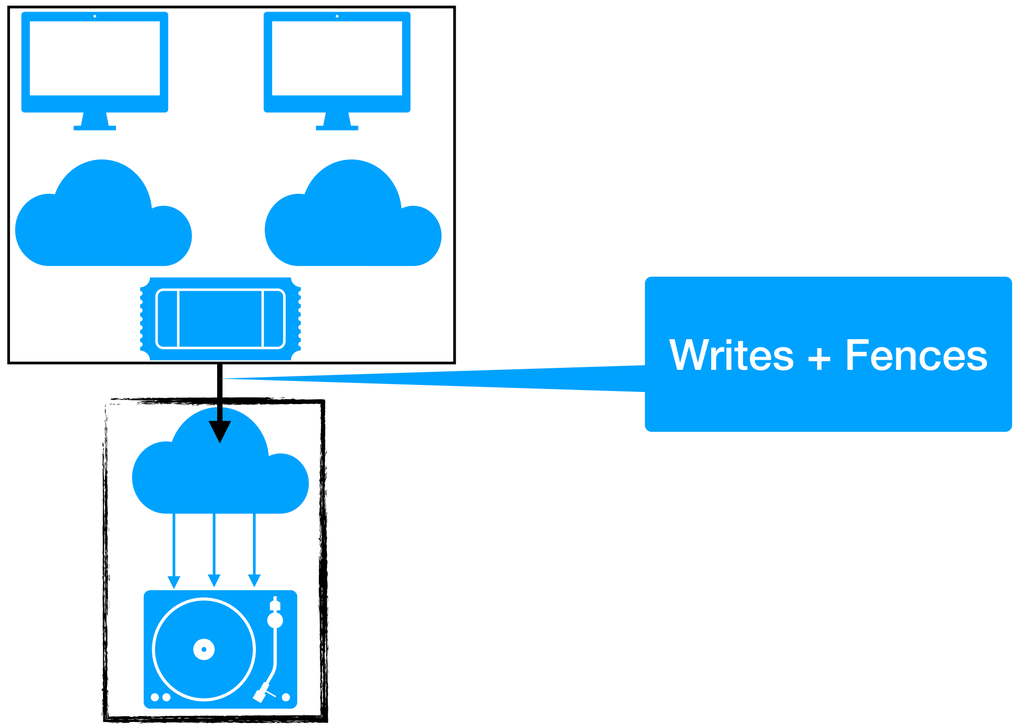


Concurrent Memory Systems + Persistency



Persistency

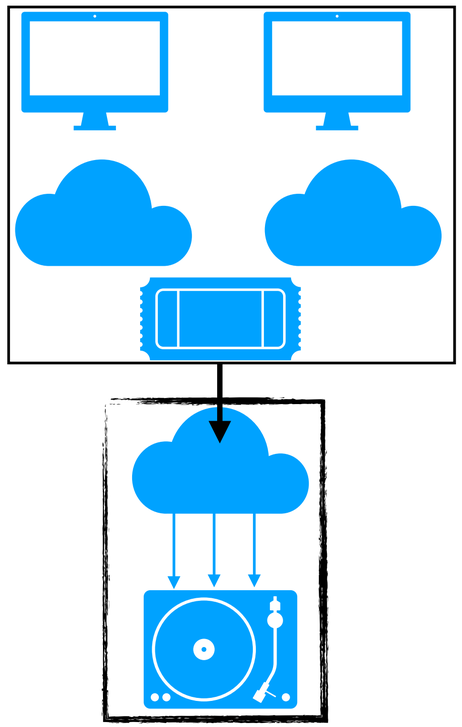
Concurrent Memory Systems + Persistency



Persistency

Archives writes, not necessarily in order

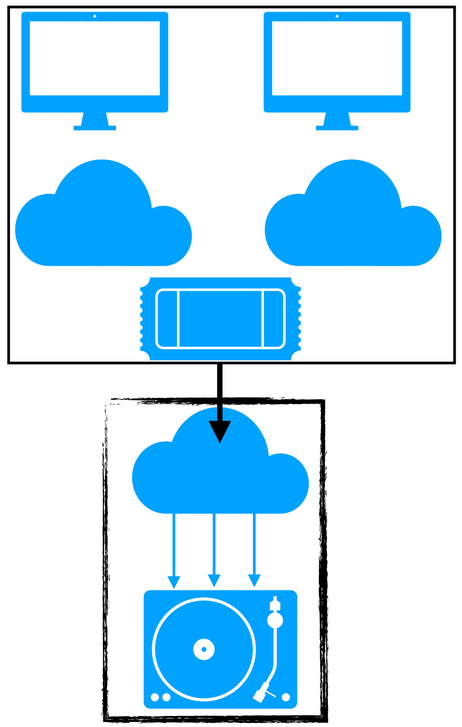
Concurrent Memory Systems + Persistency



Persistency

- Archives writes, not necessarily in order
- Re-orders incoming writes

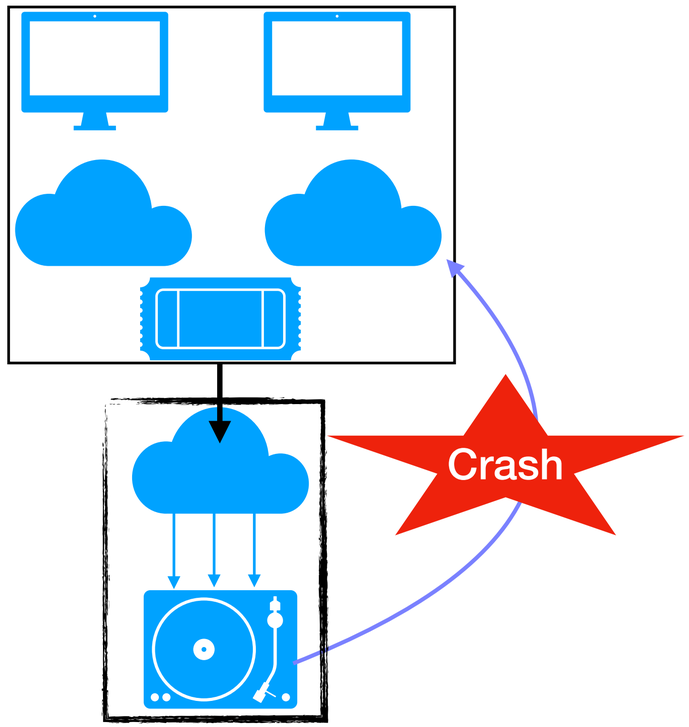
Concurrent Memory Systems + Persistency



Persistency

- Archives writes, not necessarily in order
- Re-orders incoming writes
- Fences can impose ordering

Concurrent Memory Systems + Persistency



Persistency

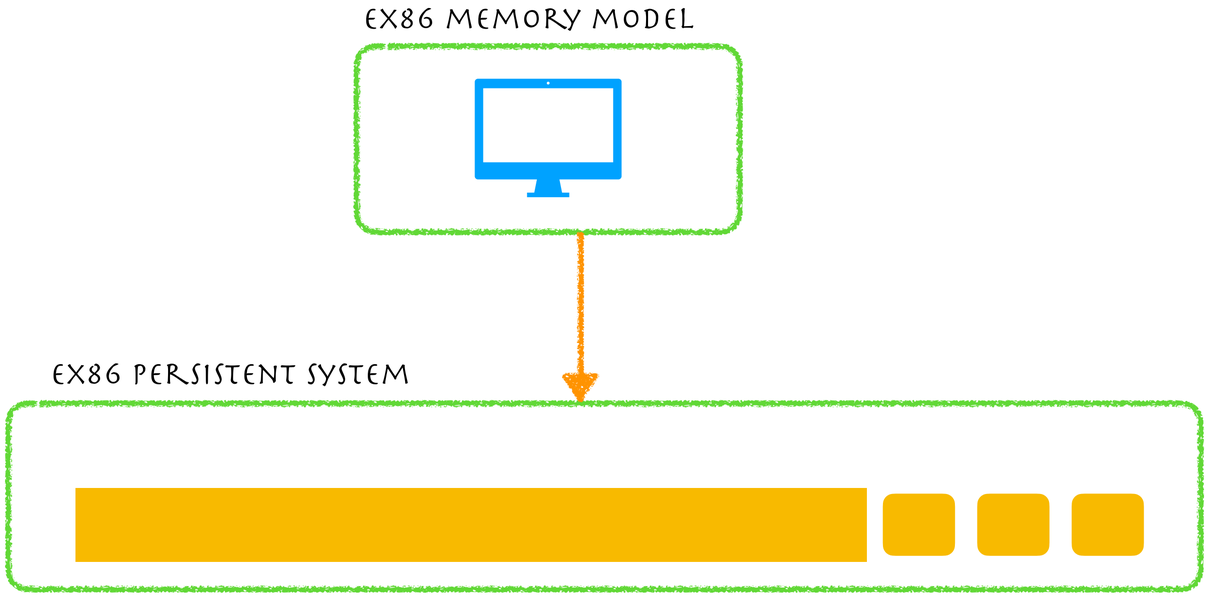
Archives writes, not necessarily in order

Re-orders incoming writes

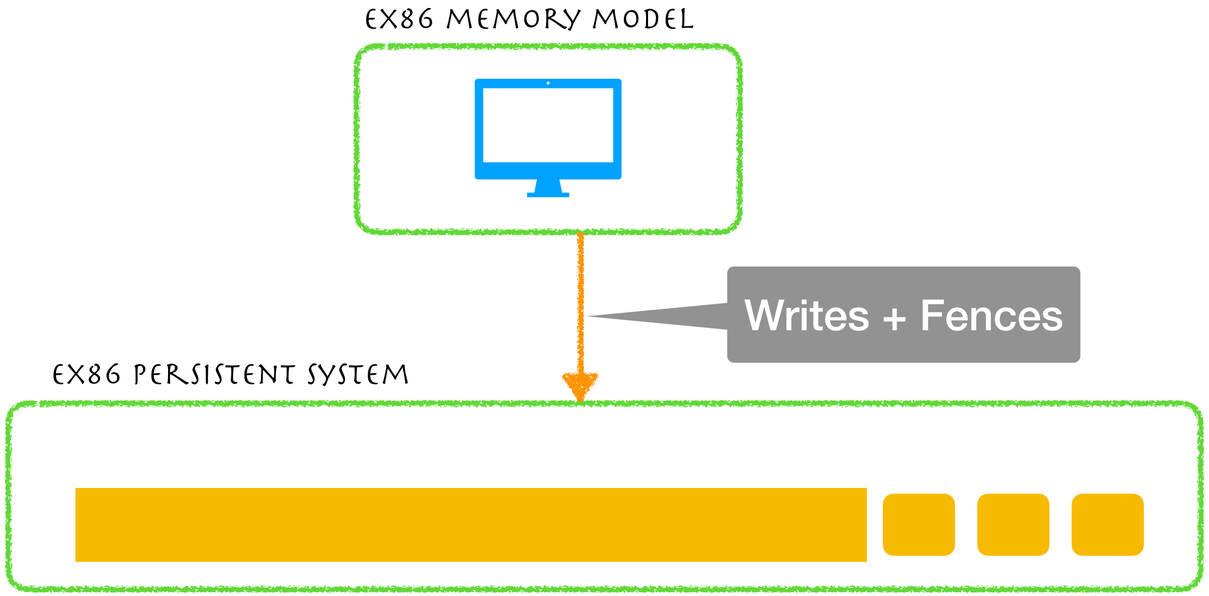
Fences can impose ordering

Useful in case of a crash

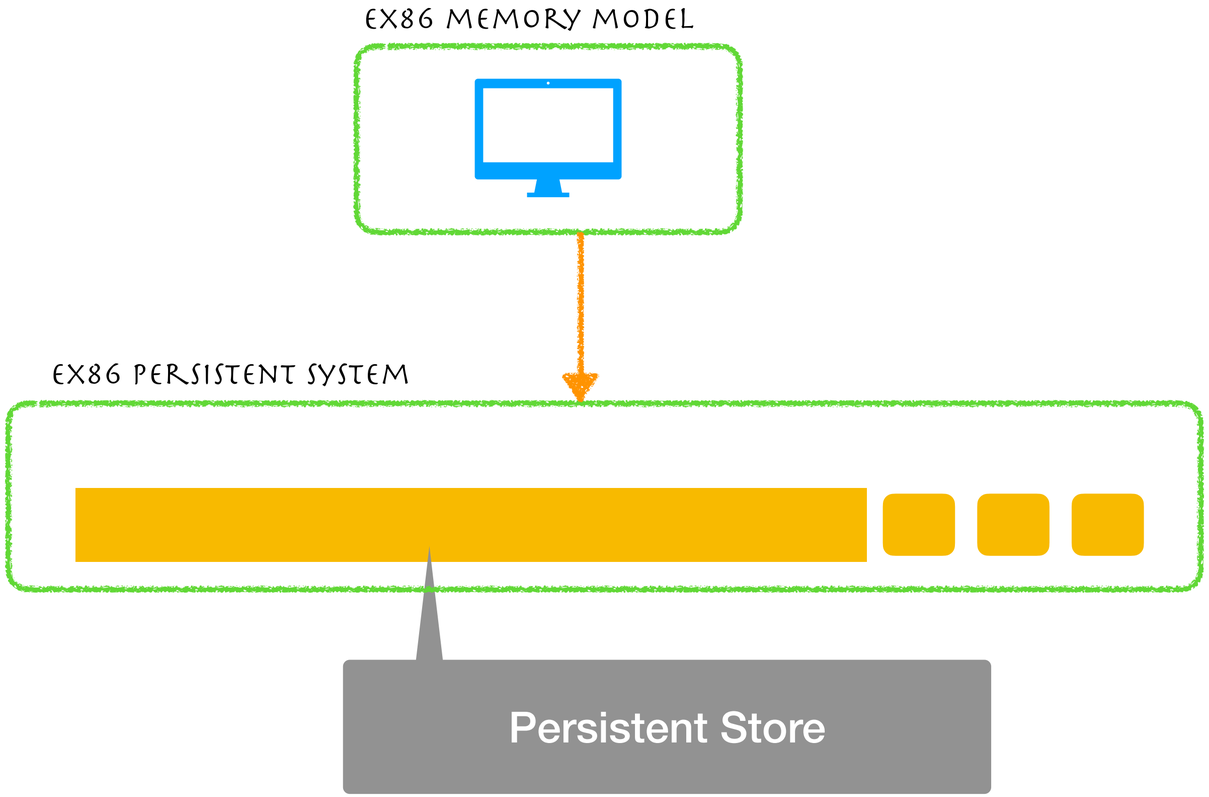
Extended x86 Persistent System



Extended x86 Persistent System



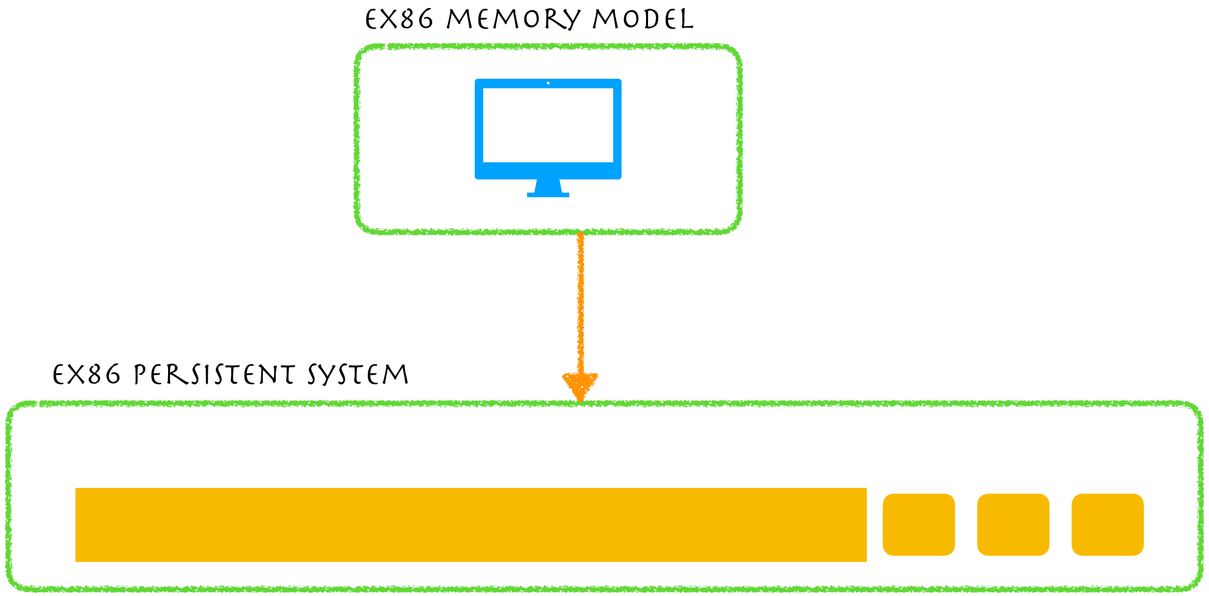
Extended x86 Persistent System



Extended x86 Persistent System

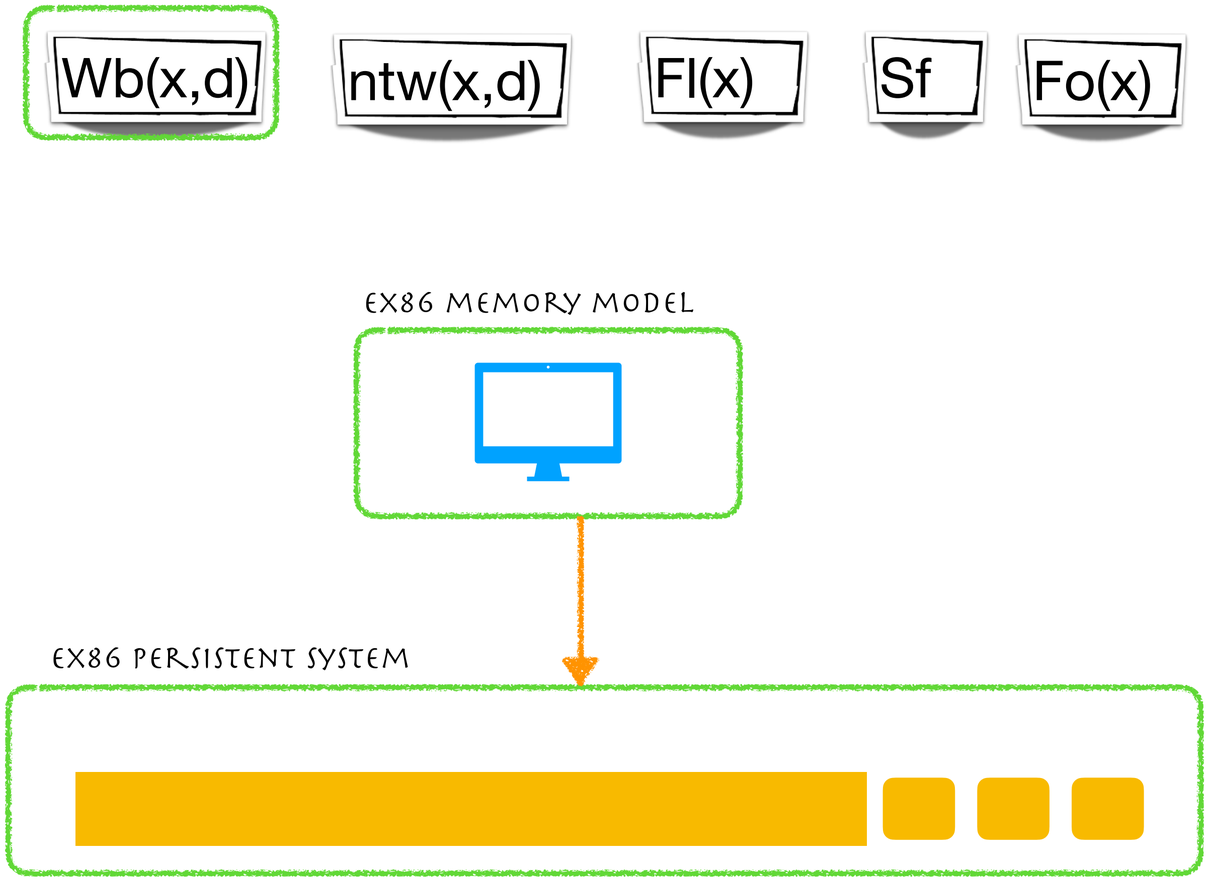
Instructions

Wb(x,d) ntw(x,d) Fl(x) Sf Fo(x)



Extended x86 Persistent System

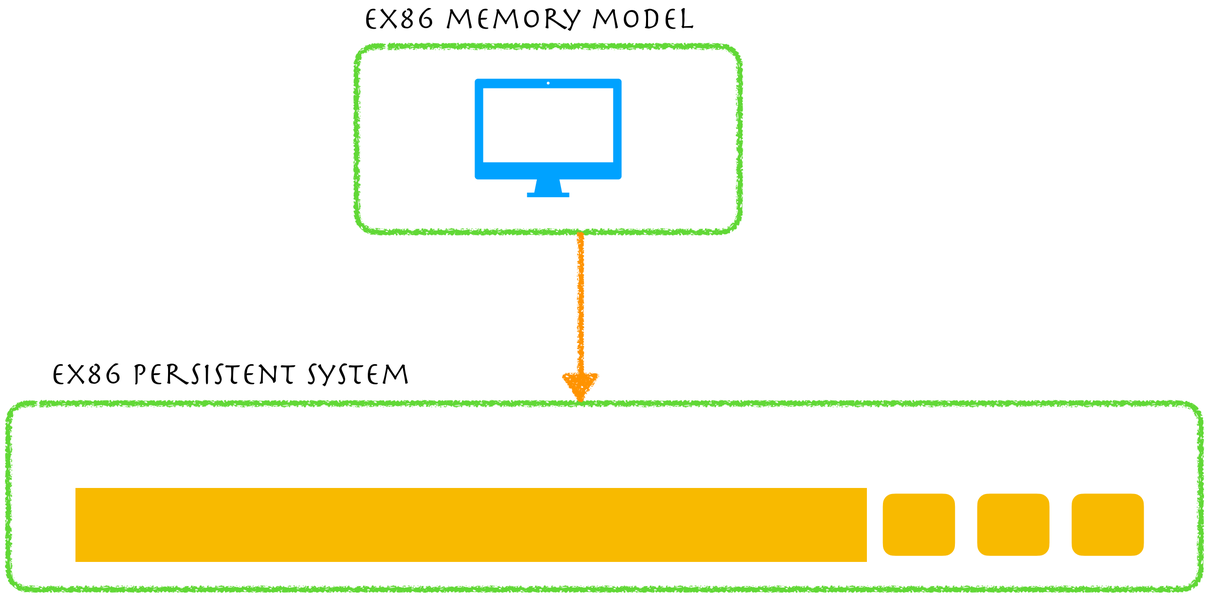
Instructions



Extended x86 Persistent System

Instructions

Wb(x,d) ntw(x,d) Fl(x) Sf Fo(x)

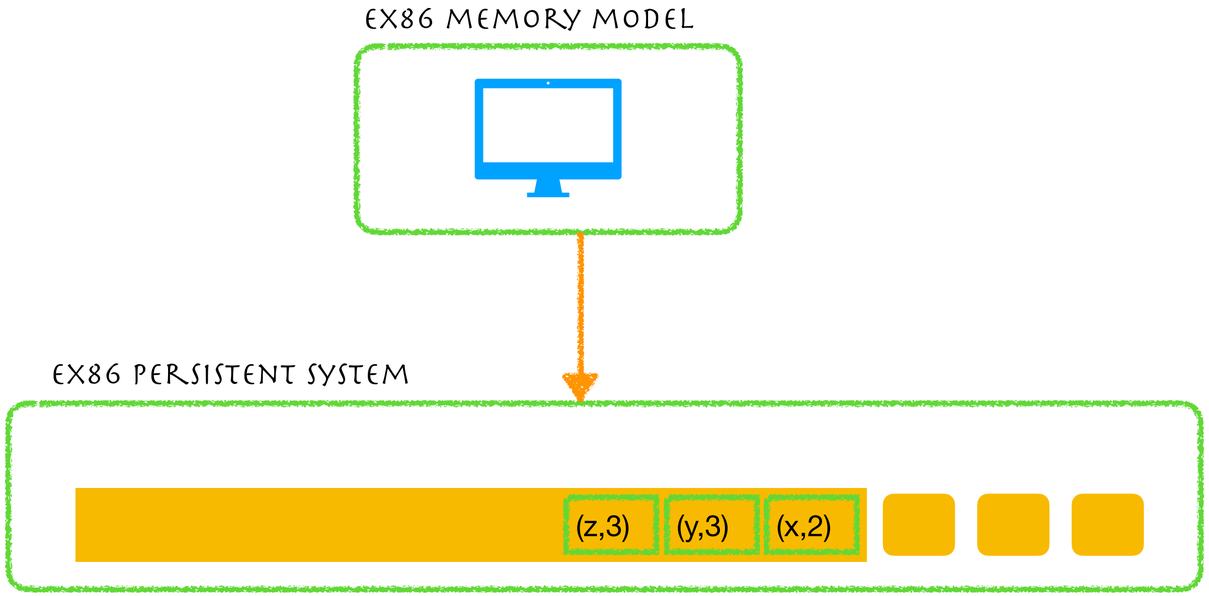


Wb writes are buffered

Extended x86 Persistent System

Instructions

Wb(x,d) ntw(x,d) Fl(x) Sf Fo(x)

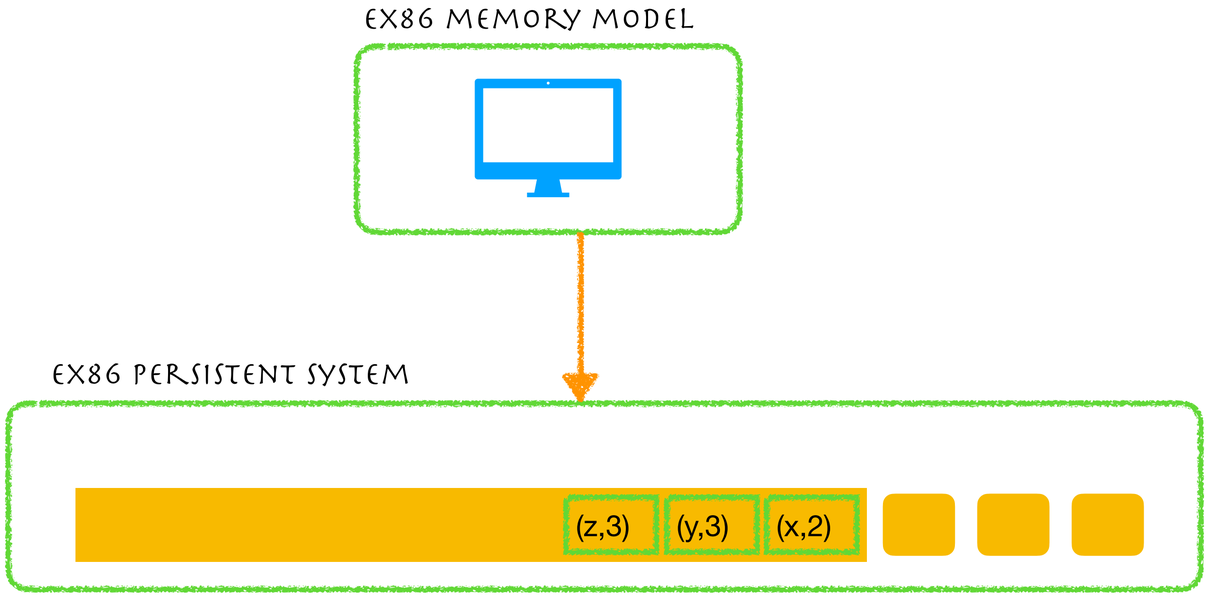


Wb writes are buffered

Extended x86 Persistent System

Instructions

Wb(x,d) ntw(x,d) Fl(x) Sf Fo(x)



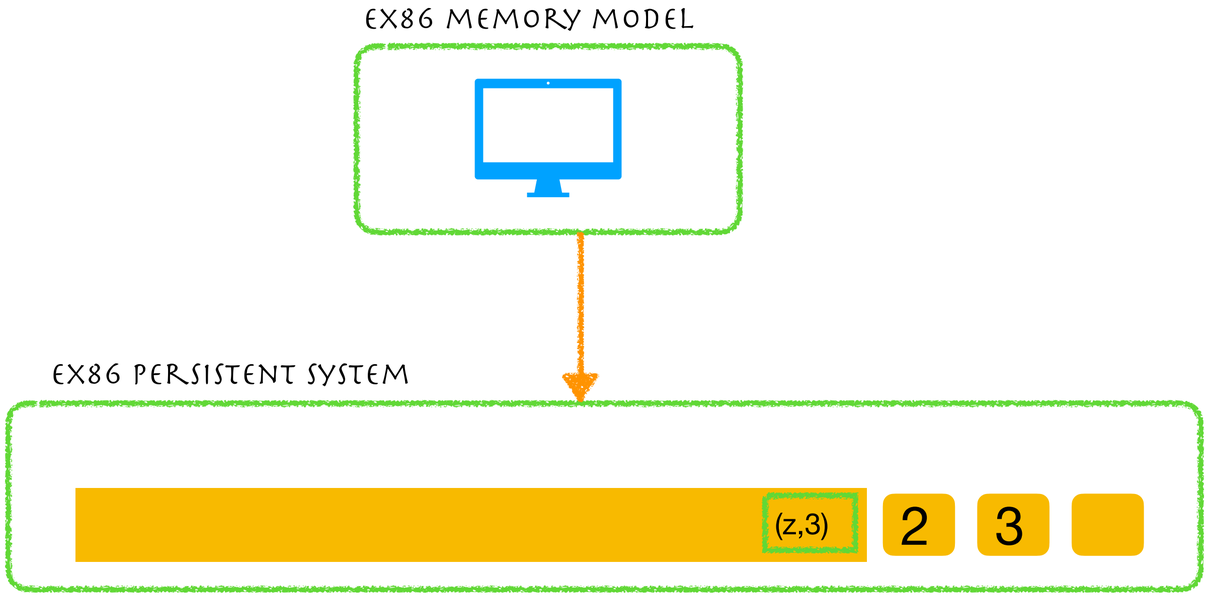
Propagated non-deterministically

Only per variable ordering

Extended x86 Persistent System

Instructions

Wb(x,d) ntw(x,d) Fl(x) Sf Fo(x)



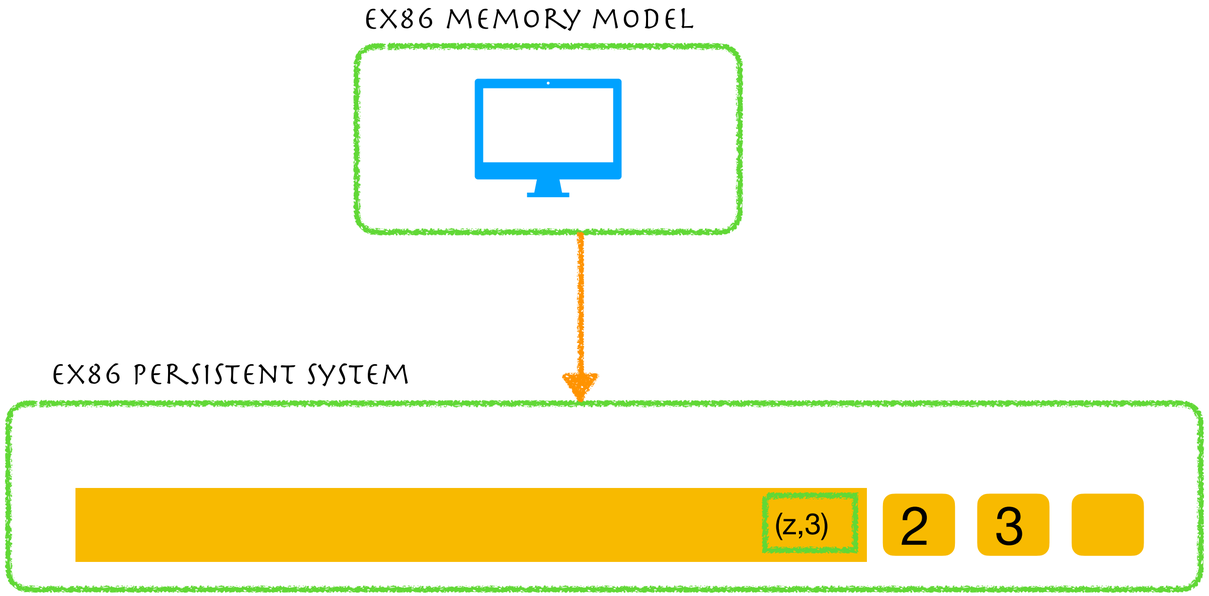
Propagated non-deterministically

Only per variable ordering

Extended x86 Persistent System

Instructions

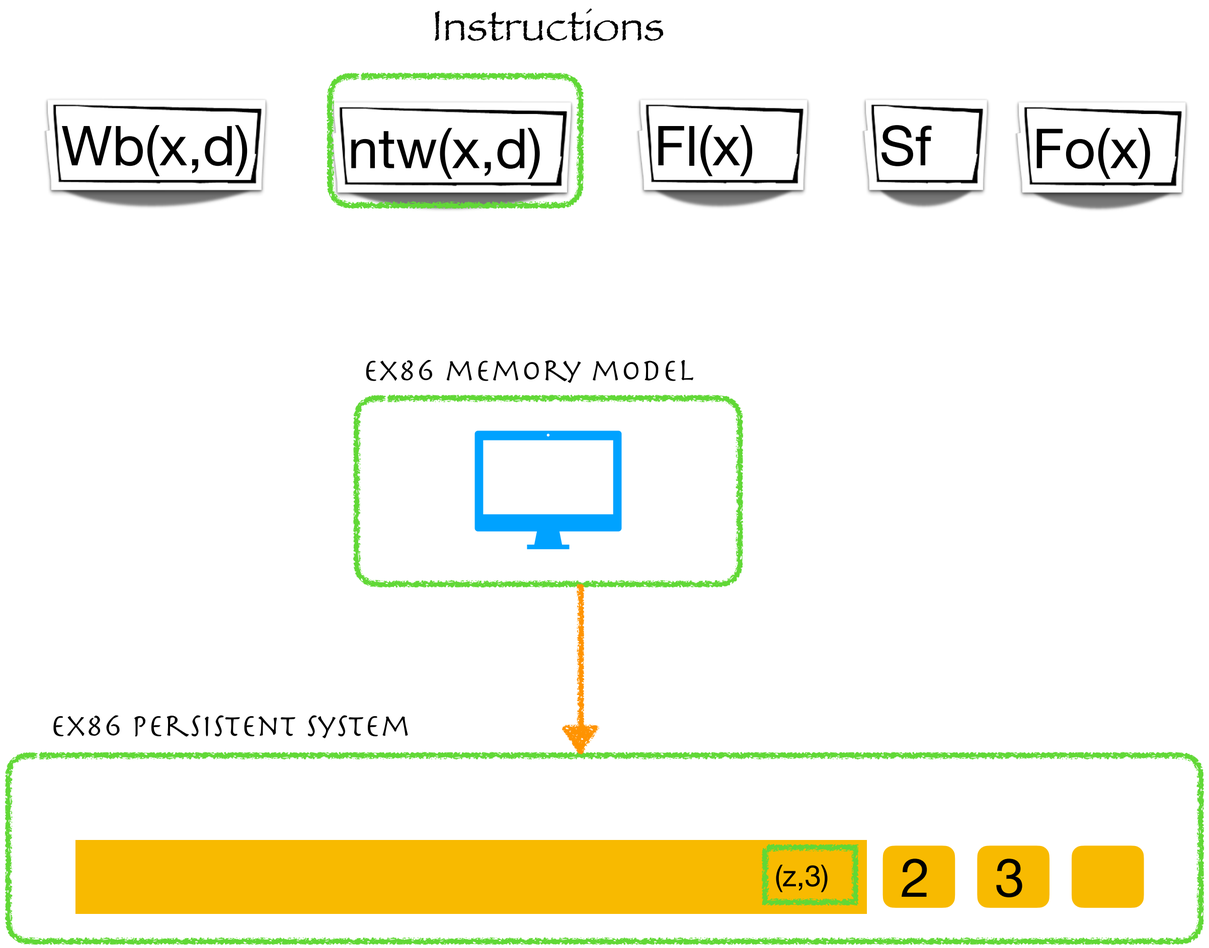
Wb(x,d) | ntw(x,d) | Fl(x) | Sf | Fo(x)



Propagated non-deterministically

Only per variable ordering

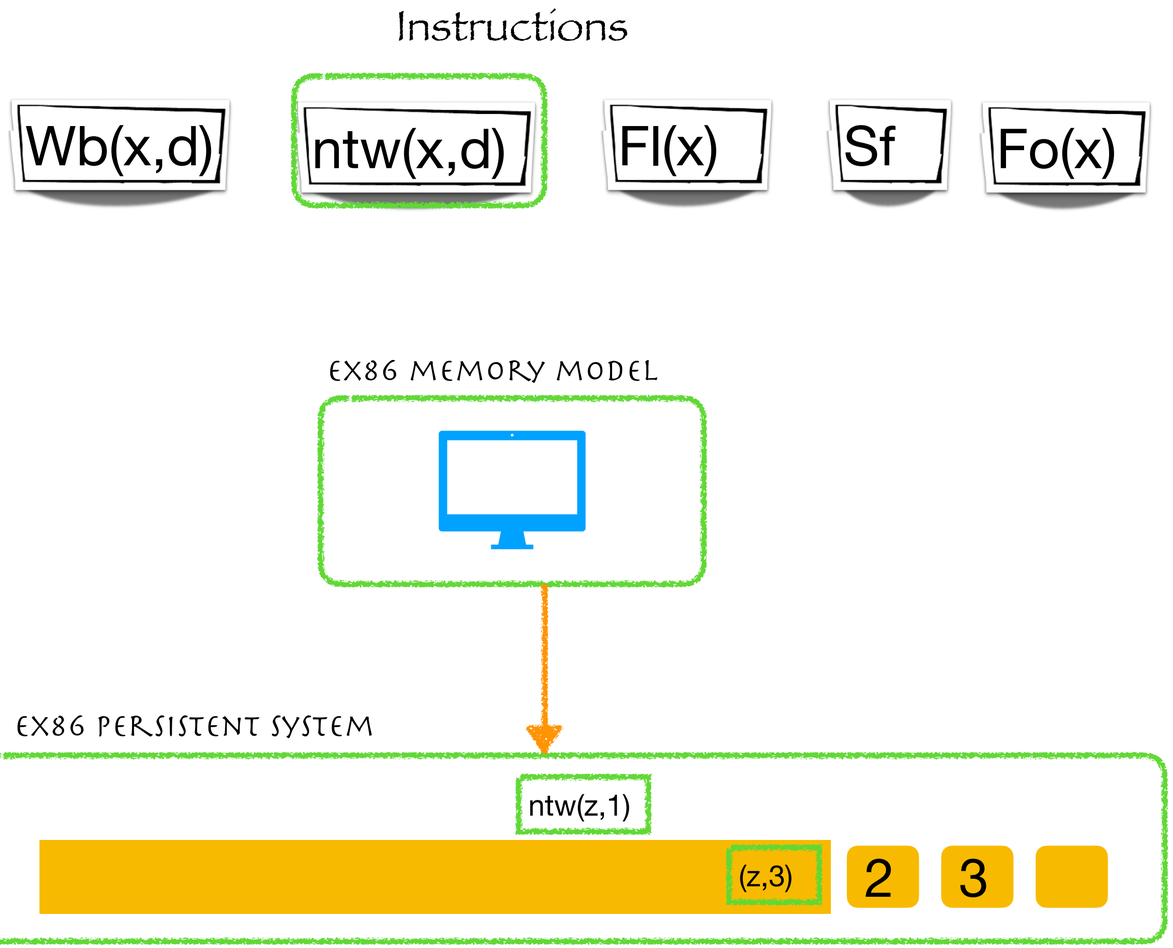
Extended x86 Persistent System



Ntw writes are propagated directly

Ensures no pending writes on the variable

Extended x86 Persistent System



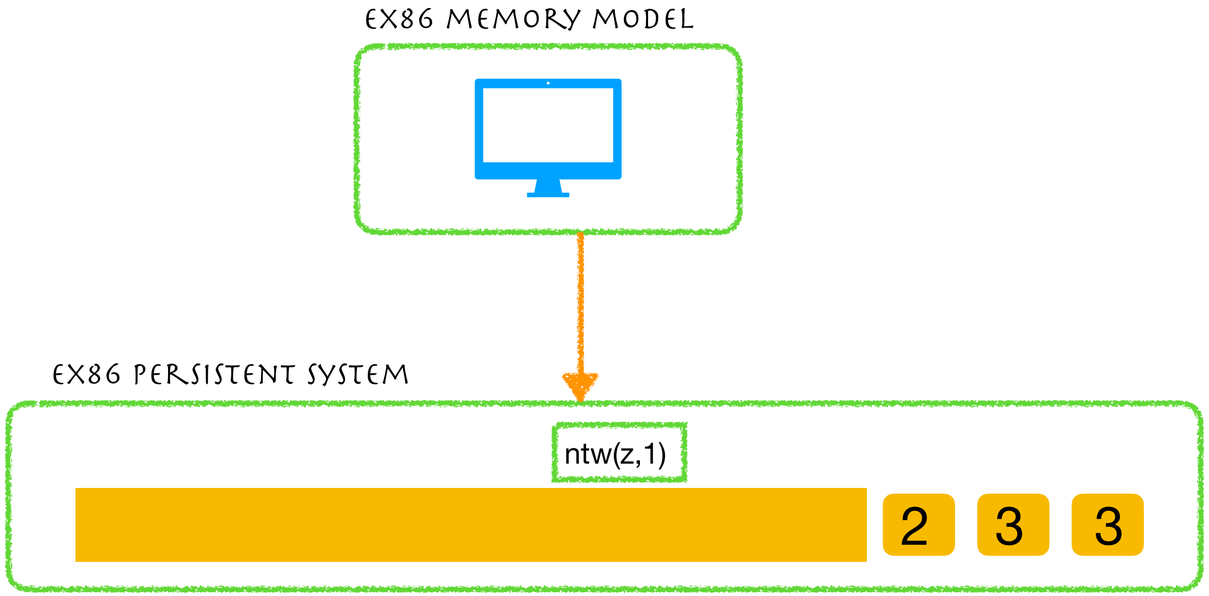
Ntw writes are propagated directly

Ensures no pending writes on the variable

Extended x86 Persistent System

Instructions

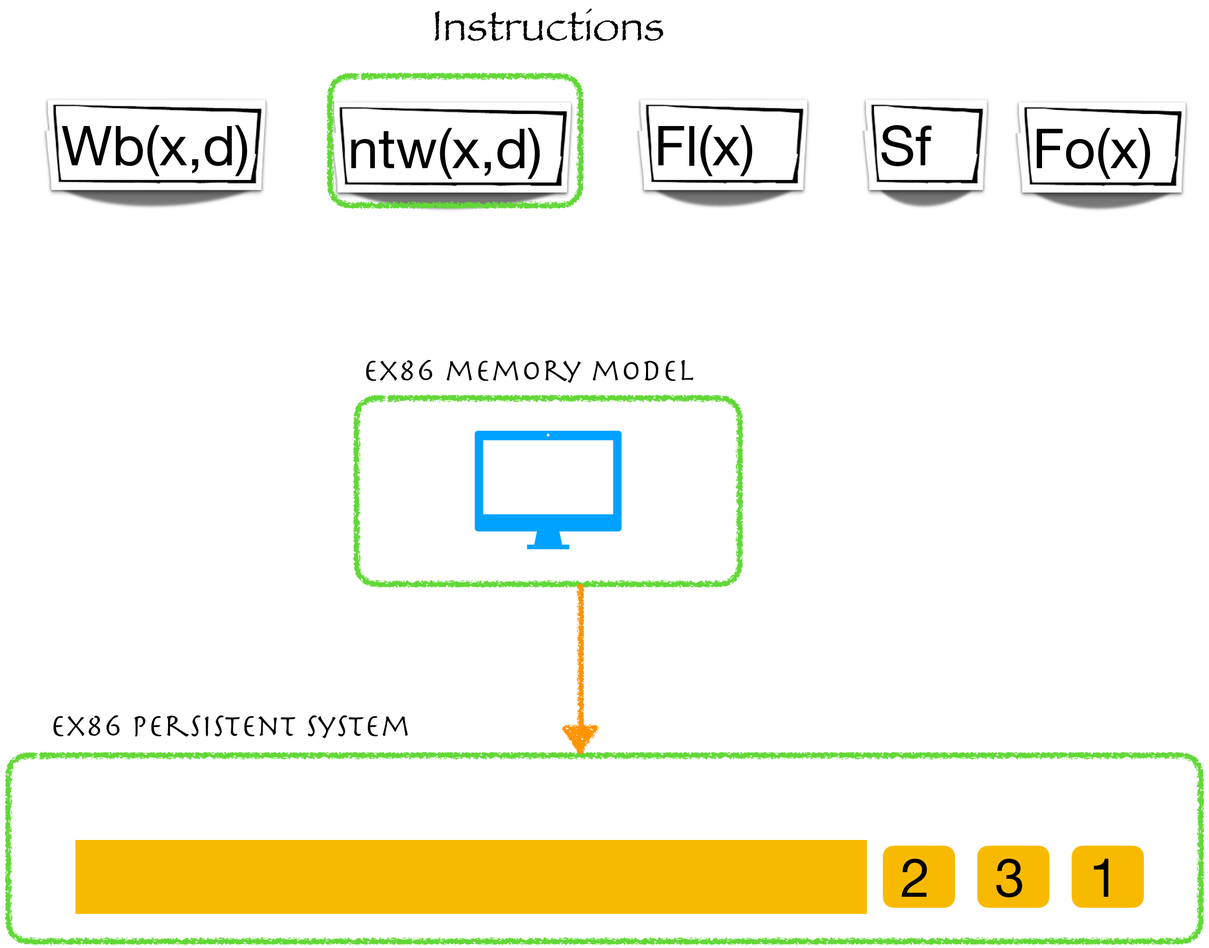
Wb(x,d) | ntw(x,d) | Fl(x) | Sf | Fo(x)



Ntw writes are propagated directly

Ensures no pending writes on the variable

Extended x86 Persistent System



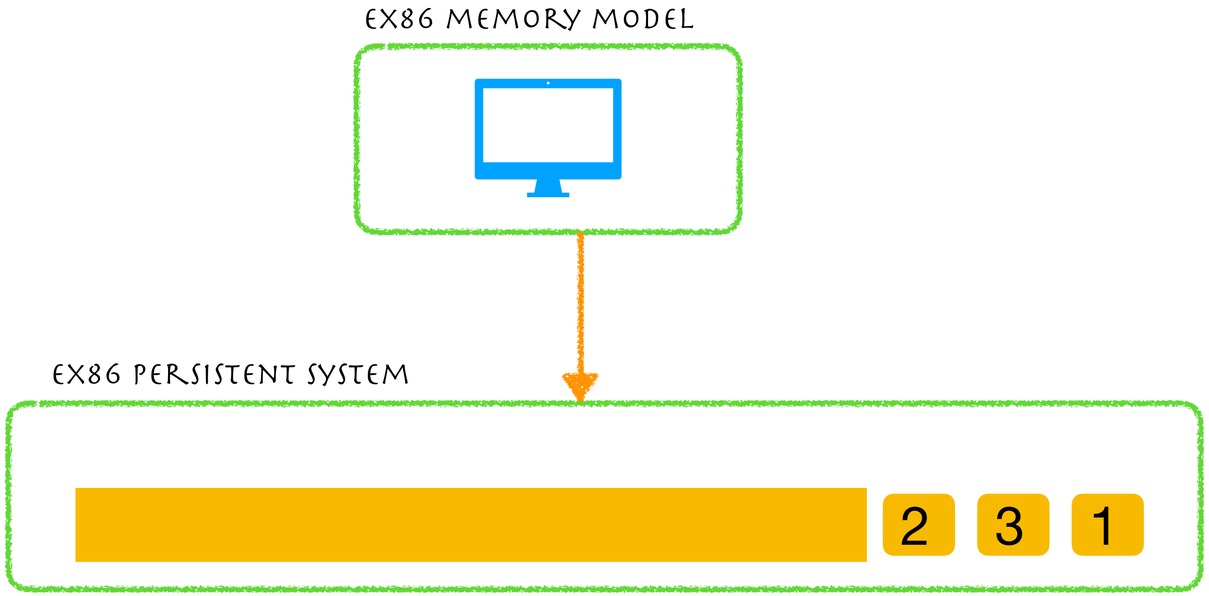
Ntw writes are propagated directly

Ensures no pending writes on the variable

Extended x86 Persistent System

Instructions

Wb(x,d) ntw(x,d) Fl(x) Sf Fo(x)



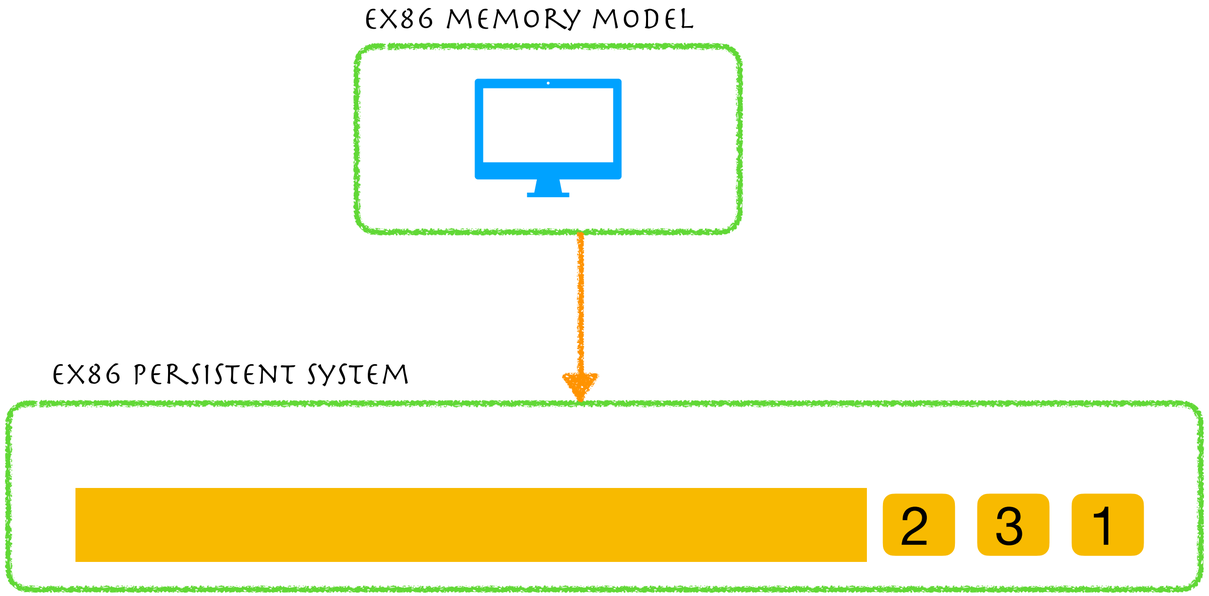
Ntw writes are propagated directly

Ensures no pending writes on the variable

Extended x86 Persistent System

Instructions

Wb(x,d) ntw(x,d) Fl(x) Sf Fo(x)

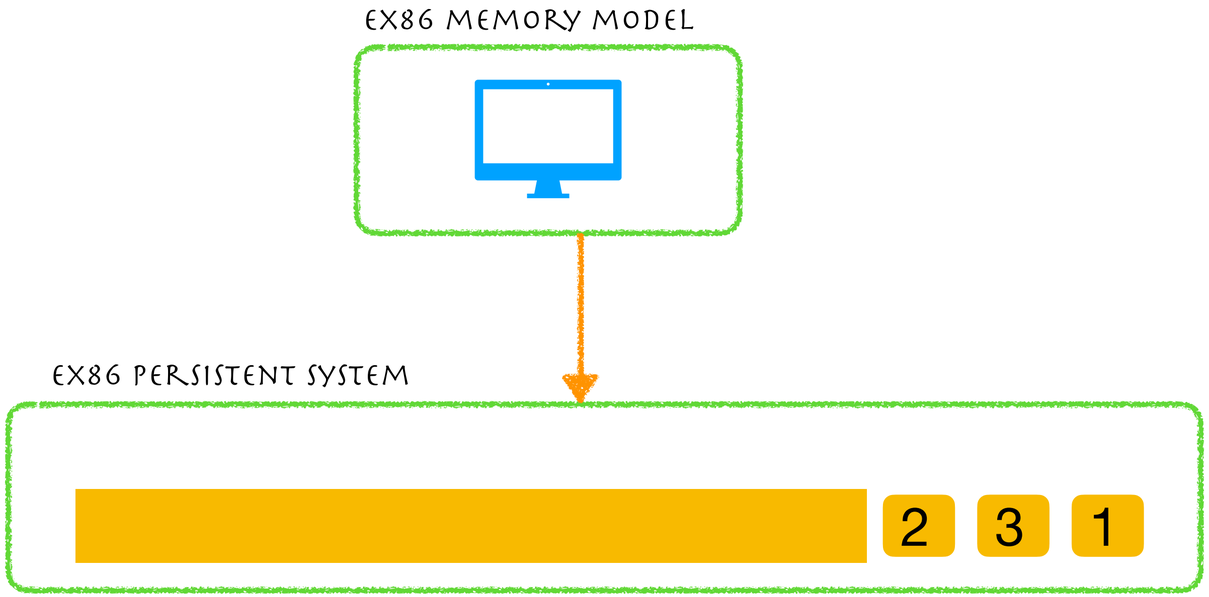


Fl ensures no pending writes on the variable

Extended x86 Persistent System

Instructions

Wb(x,d) ntw(x,d) Fl(x) Sf Fo(x)

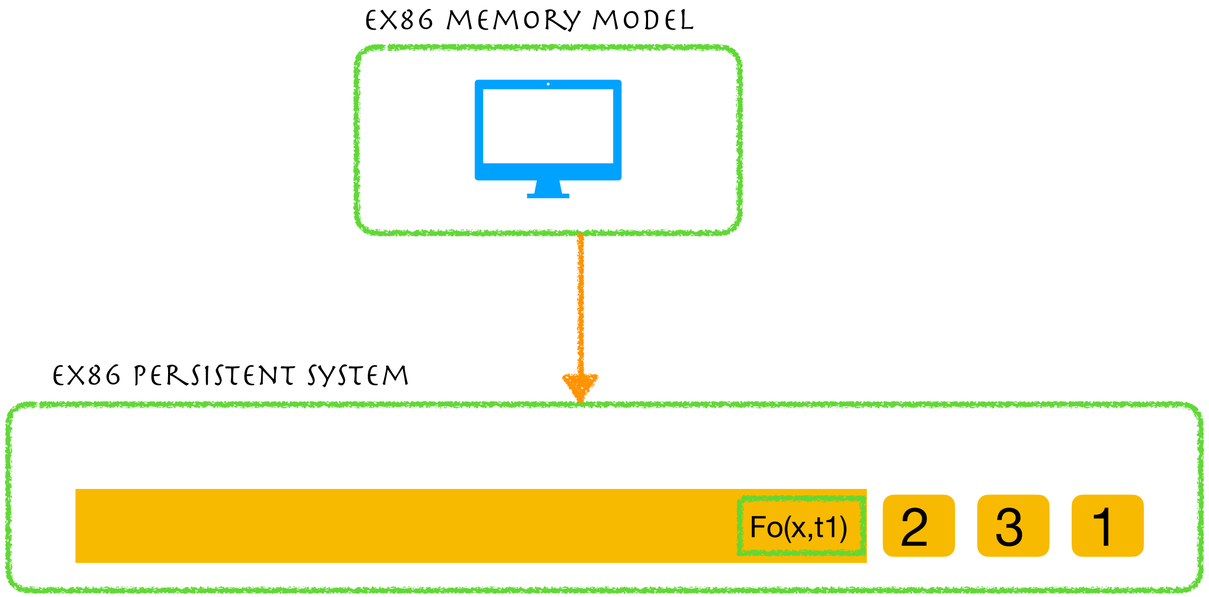


Fl ensures no pending writes on the variable

Extended x86 Persistent System

Instructions

Wb(x,d) ntw(x,d) Fl(x) Sf Fo(x)



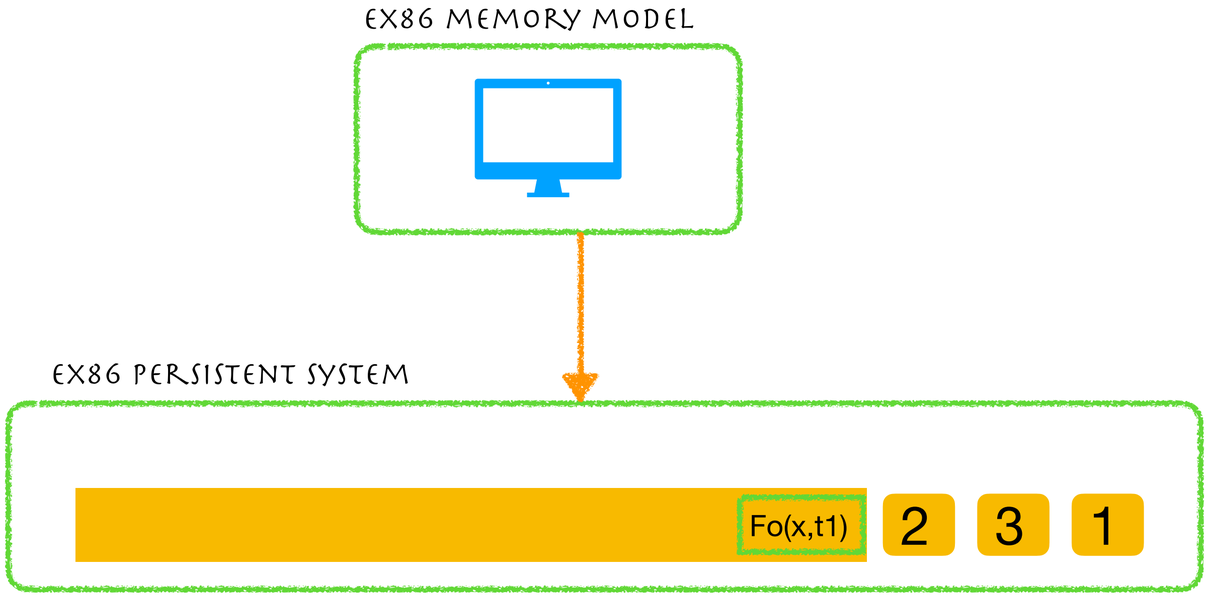
Fo is buffered with thread information

SF ensures no pending Fo of that thread

Extended x86 Persistent System

Instructions

Wb(x,d) ntw(x,d) Fl(x) Sf Fo(x)



Fo is buffered with thread information

SF ensures no pending Fo of that thread

Ensures prior writes by the thread are persisted

VERIFICATION

Crisis and deadlocks when they occur have at least this advantage.: they force us to think

Verifying Concurrent Systems

Verifying Concurrent Systems

Verifying Concurrent Systems

Correctness
Specification

Verifying Concurrent Systems

$\models$

Correctness Specification

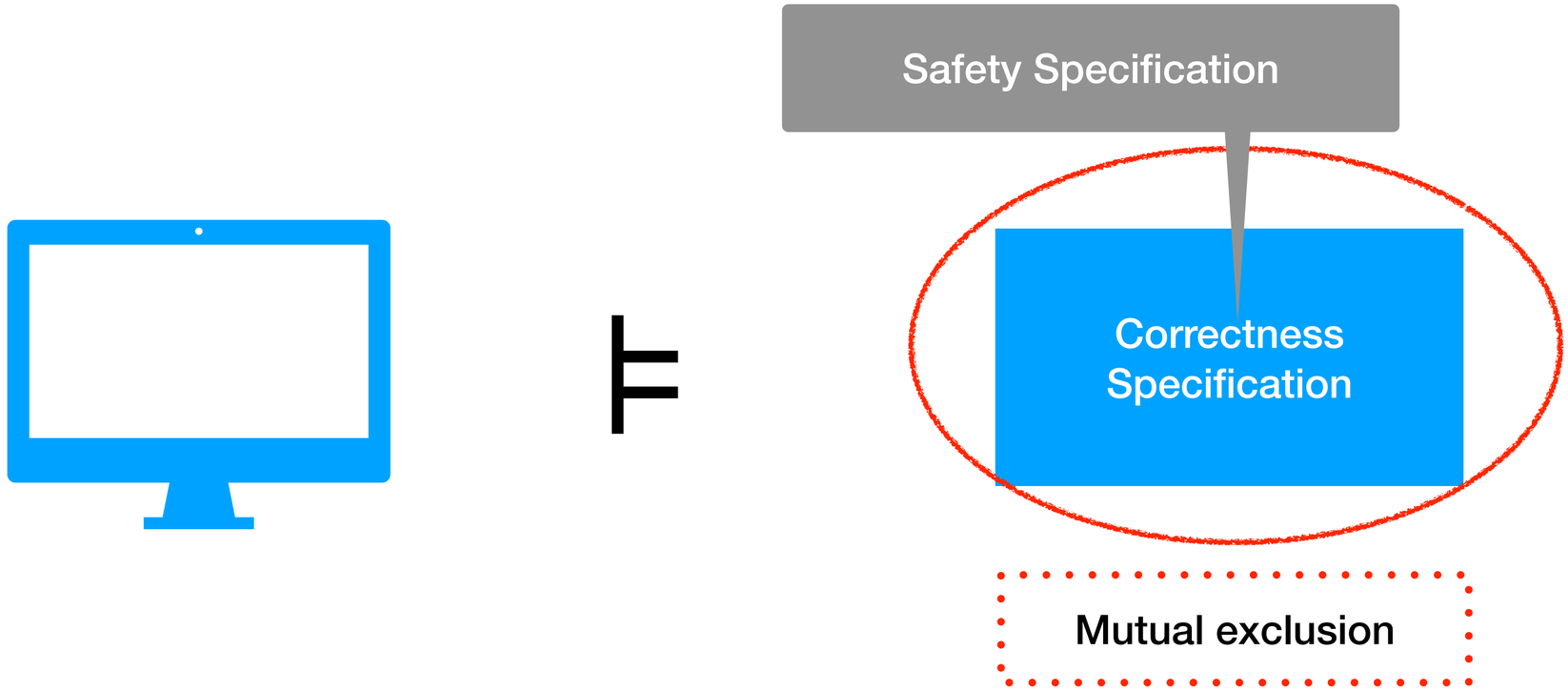
Verifying Concurrent Systems

$\models$

Correctness Specification

Mutual exclusion

Verifying Concurrent Systems



Verifying Concurrent Systems

$\models$

Correctness Specification

Mutual exclusion

Sequential Consistency	PSPACE-Complete
TSO	Non-primitive recursive
PSO	Non-primitive recursive

Verifying Concurrent Systems

$\models$

Correctness Specification

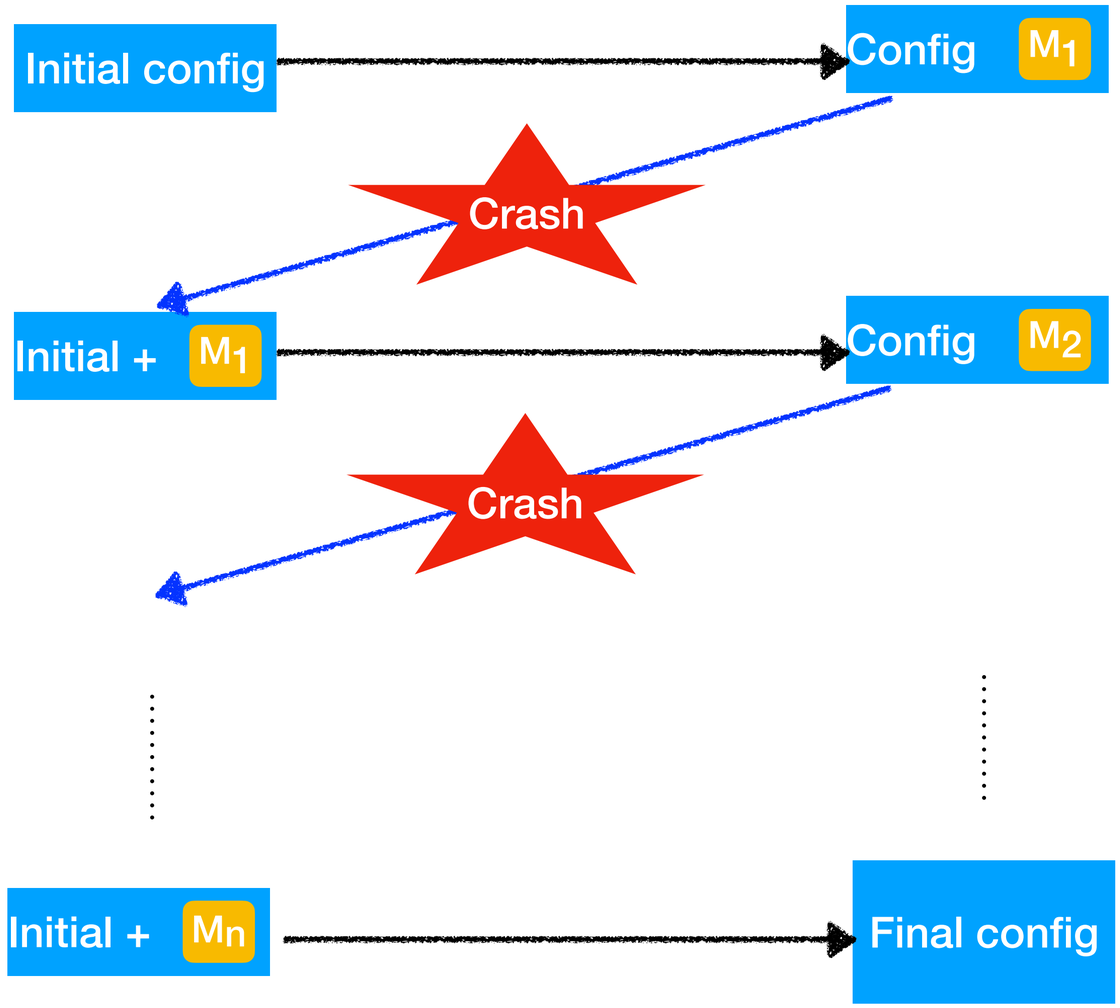
Mutual exclusion

Sequential Consistency	PSPACE-Complete
TSO	Non-primitive recursive
PSO	Non-primitive recursive

Theory of Well Structured Transition Systems

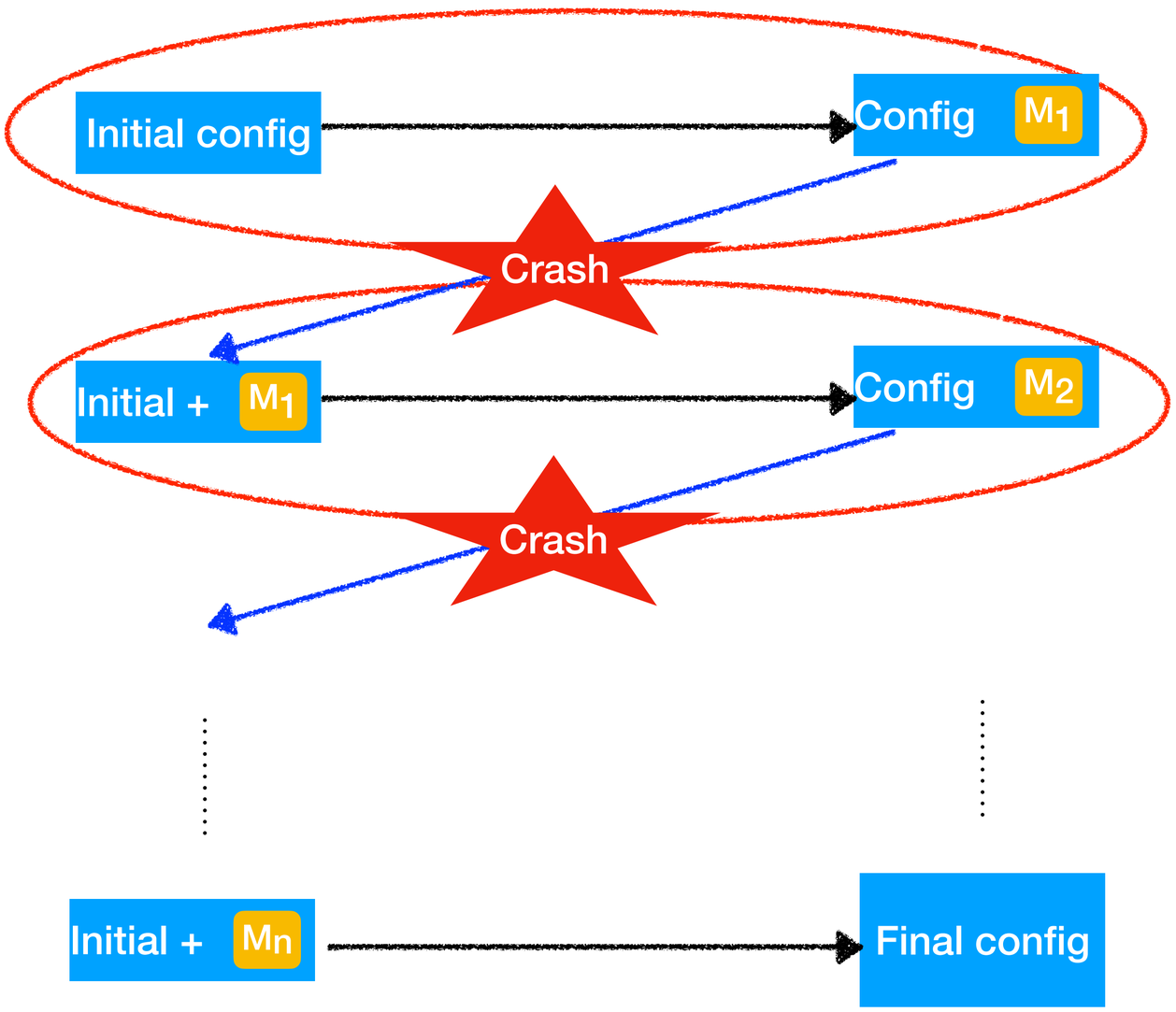
Persistent Reachability Problem

Persistent Reachability Problem



Persistent reachability: Whether programs can reach a program location in presence of crashes

Persistent Reachability Problem

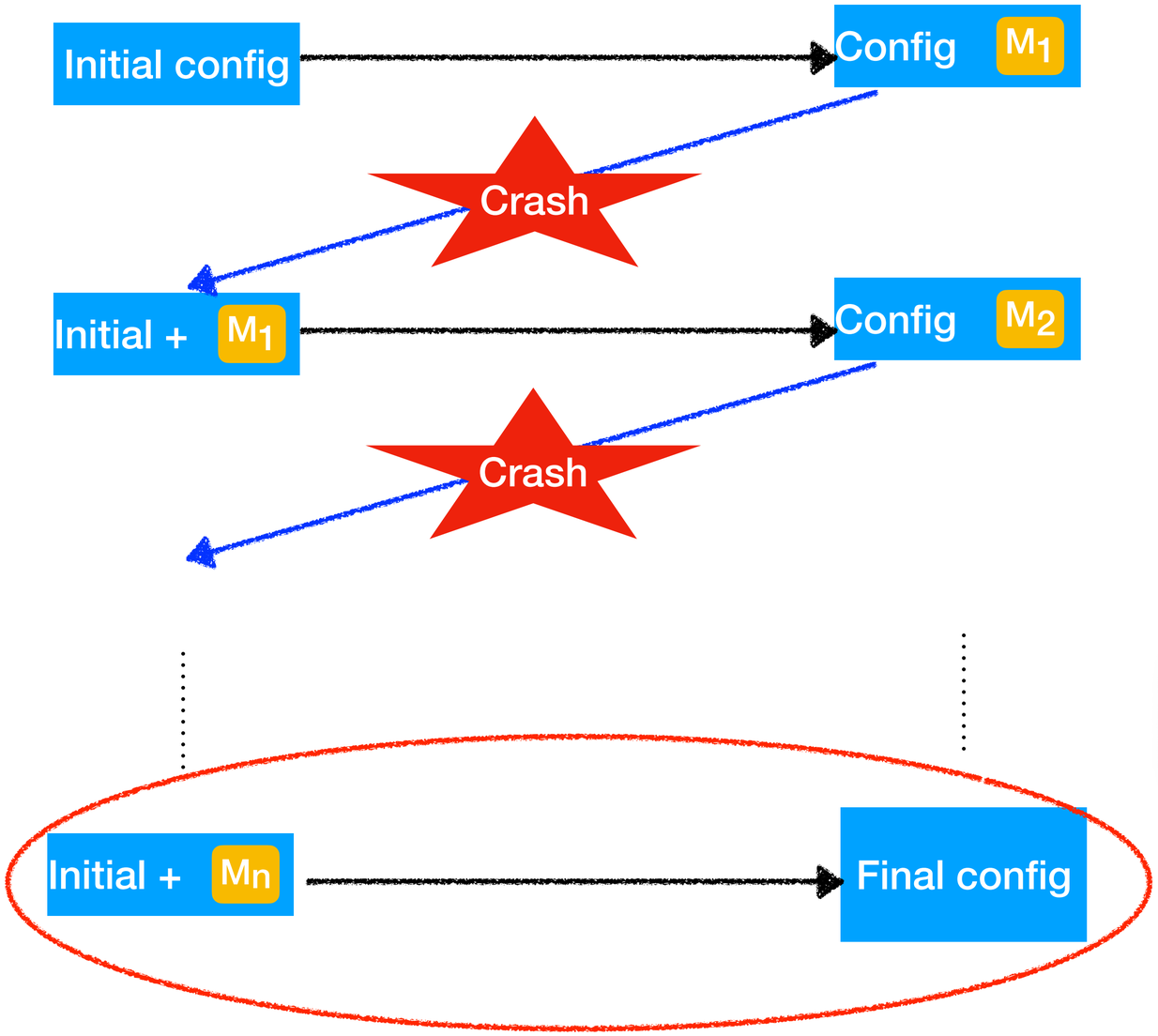


Persistent reachability: Whether programs can reach a program location in presence of crashes

Persistent memory

Whether a persistent memory can be reached

Persistent Reachability Problem



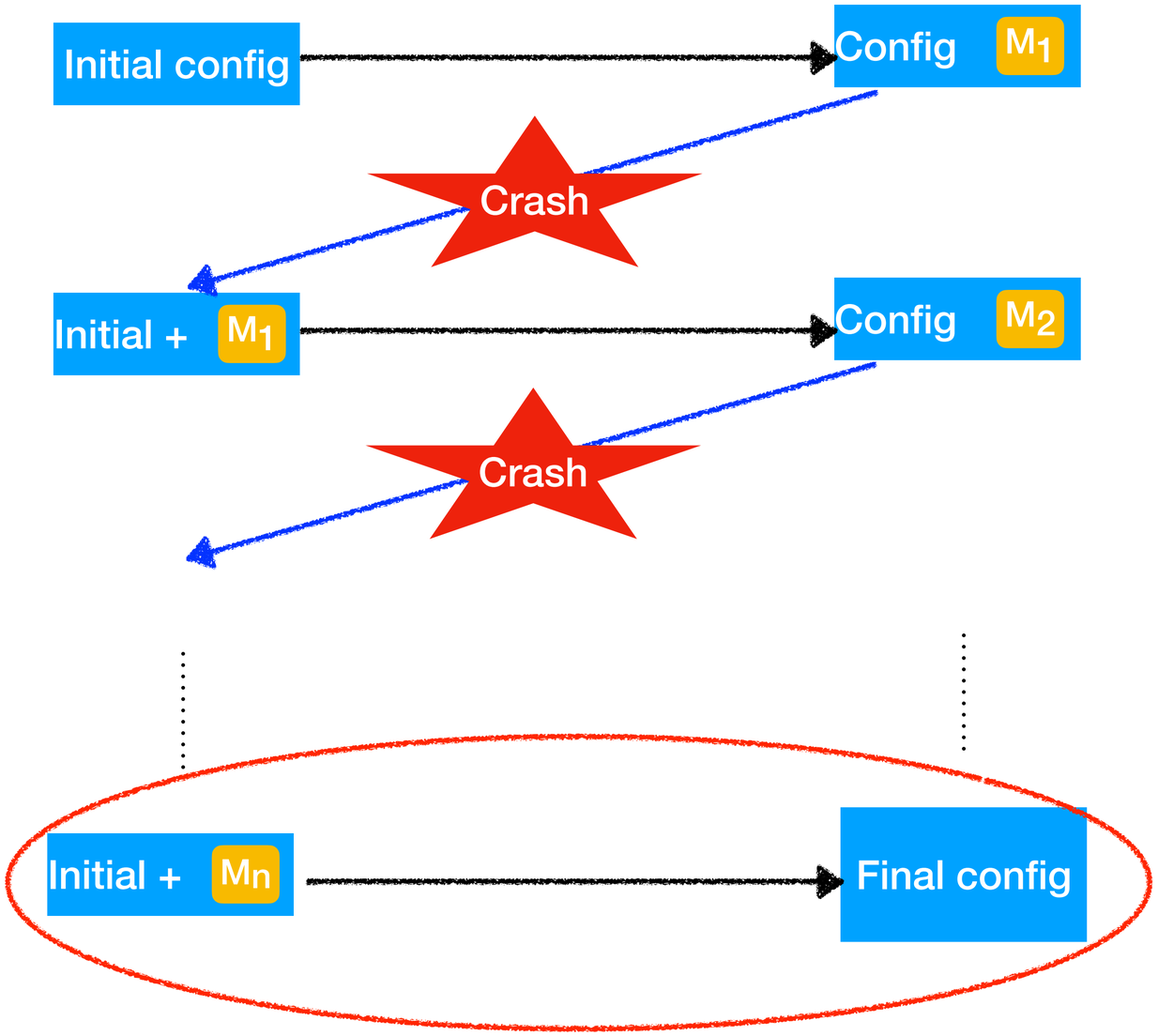
Persistent reachability: Whether programs can reach a program location in presence of crashes

Crash free reachability

Persistent memory

State reachability without crashes

Persistent Reachability Problem



Persistent reachability: Whether programs can reach a program location in presence of crashes

Crash free reachability + Persistent memory

VERIFYING EX86 WITH PERSISTENCY

All stable processes we shall predict. all unstable processes we shall control - John Von Neumann

VERIFYING EX86 WITH PERSISTENCY

All stable processes we shall predict, all unstable processes we shall control - John Von Neumann

- Persistent Memory Reachability

VERIFYING EX86 WITH PERSISTENCY

All stable processes we shall predict, all unstable processes we shall control - John Von Neumann

- Persistent Memory Reachability

- Crash Free Reachability

VERIFYING EX86 WITH PERSISTENCY

All stable processes we shall predict, all unstable processes we shall control - John Von Neumann

- Persistent Memory Reachability

- Crash Free Reachability

VERIFYING EX86 WITH PERSISTENCY

All stable processes we shall predict, all unstable processes we shall control - John Von Neumann

- Persistent Memory Reachability

- Crash Free Reachability

Verification under Intel-x86 with Persistency

PAROSH ABDULLA, Uppsala University, Sweden
MOHAMED FAOUZI ATIG, Uppsala University, Sweden
AHMED BOUAJJANI, Université Paris Cité, France
K. NARAYAN KUMAR, Chennai Mathematical Institute and IRL ReLaX, India
PRAKASH SAIVASAN, Institute of Mathematical Sciences, HBNI and IRL ReLaX, India

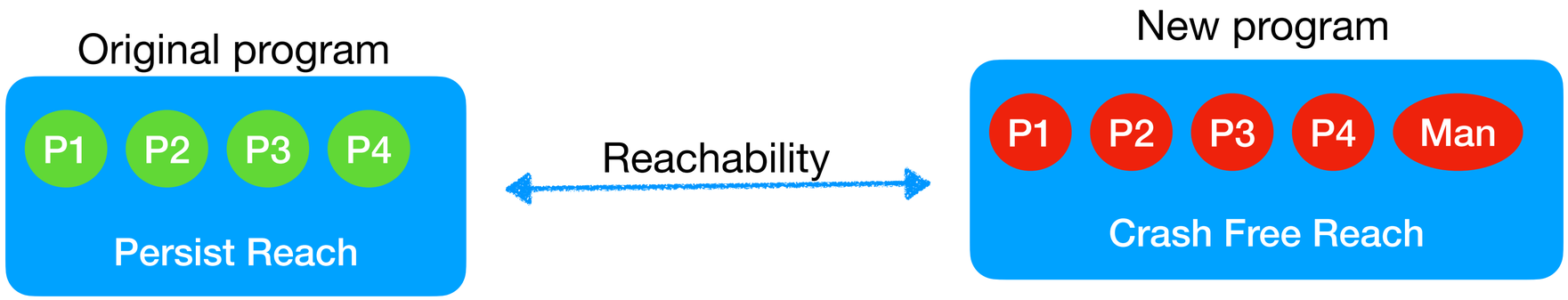
Persistent Memory Reachability

Persistent Memory Reachability

Persistent reachability problem reduces to crash free reachability in a new program

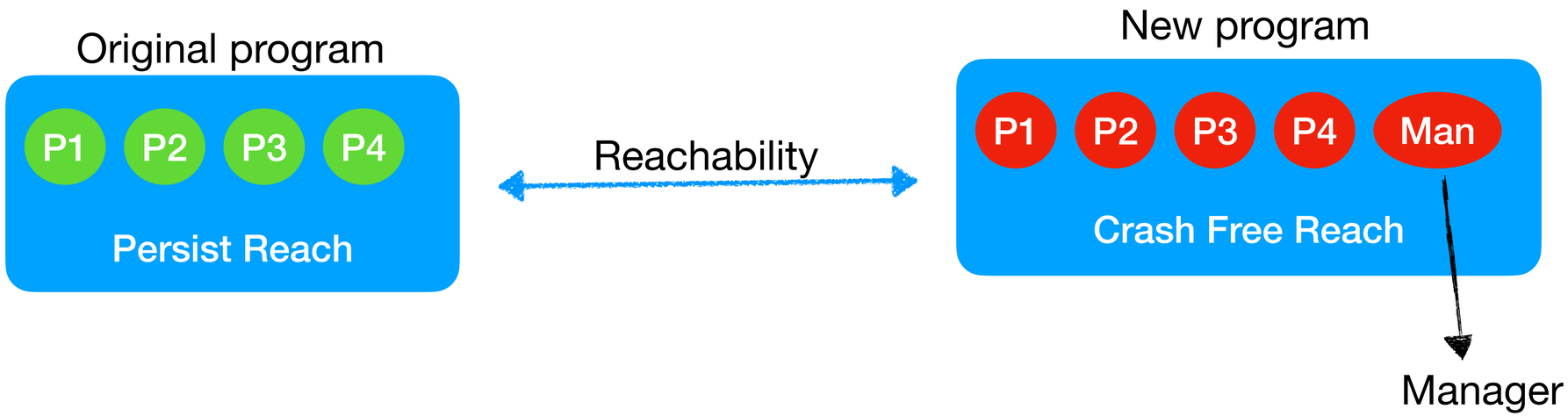
Persistent Memory Reachability

Persistent reachability problem reduces to crash free reachability in a new program



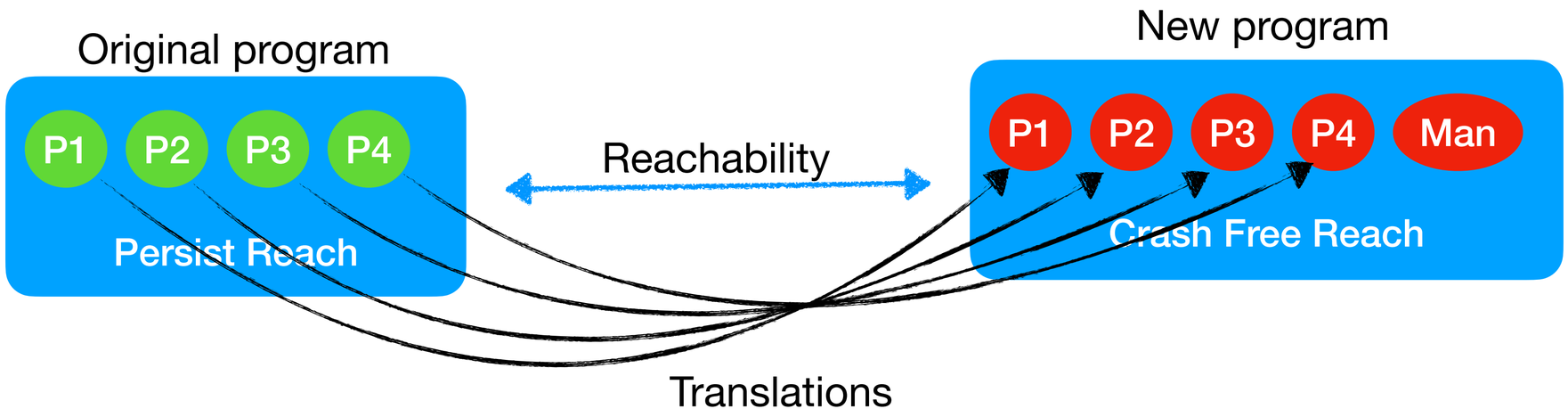
Persistent Memory Reachability

Persistent reachability problem reduces to crash free reachability in a new program



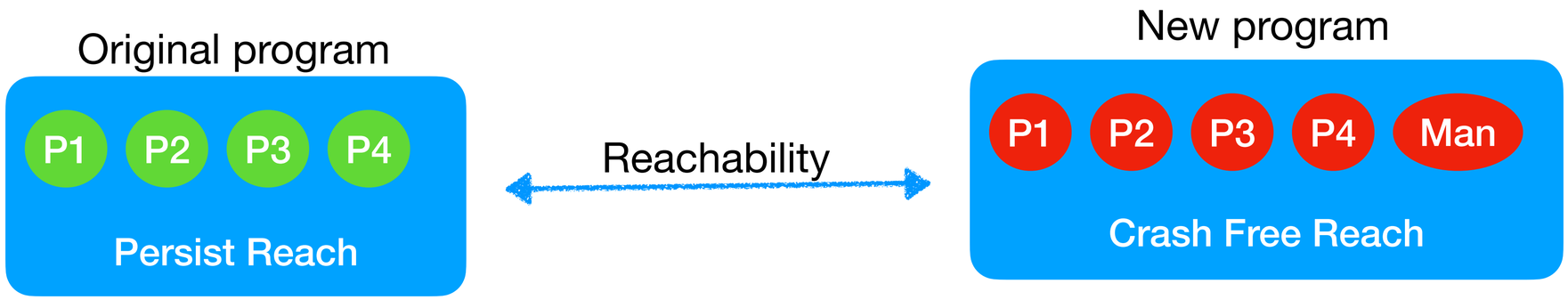
Persistent Memory Reachability

Persistent reachability problem reduces to crash free reachability in a new program



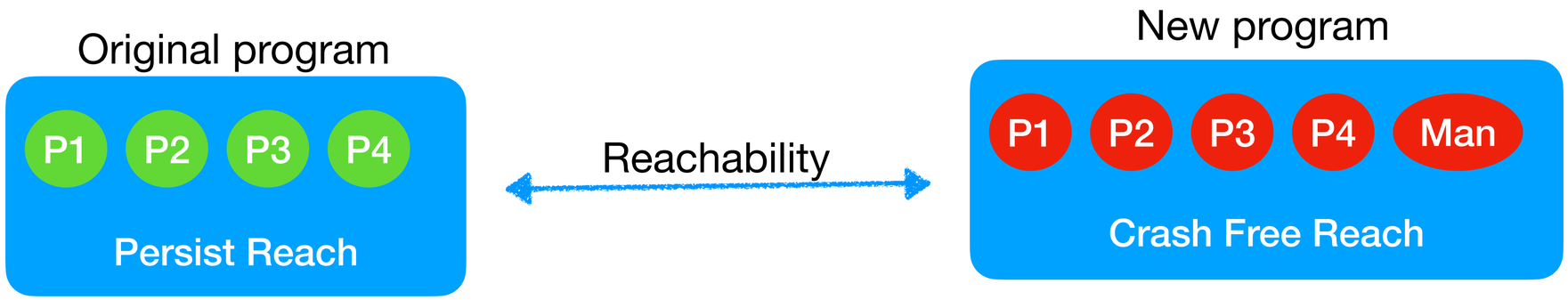
Persistent Memory Reachability

Persistent reachability problem reduces to crash free reachability in a new program



Persistent Memory Reachability

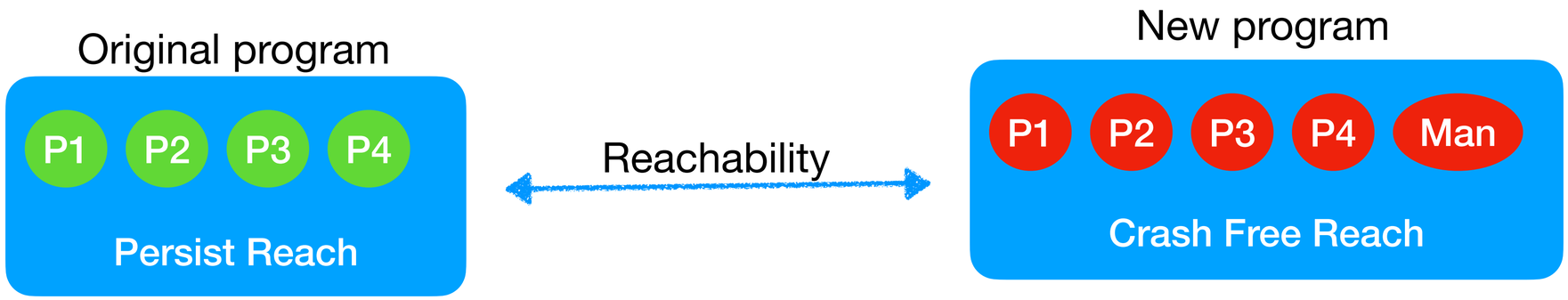
Persistent reachability problem reduces to crash free reachability in a new program



Translation employs a guess and verify technique

Persistent Memory Reachability

Persistent reachability problem reduces to crash free reachability in a new program

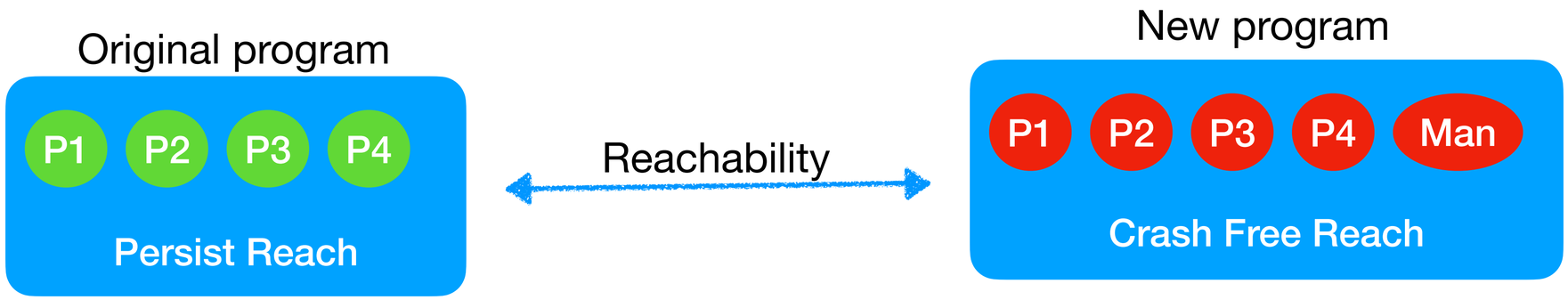


Translation employs a guess and verify technique

Guess the writes that will persist last

Persistent Memory Reachability

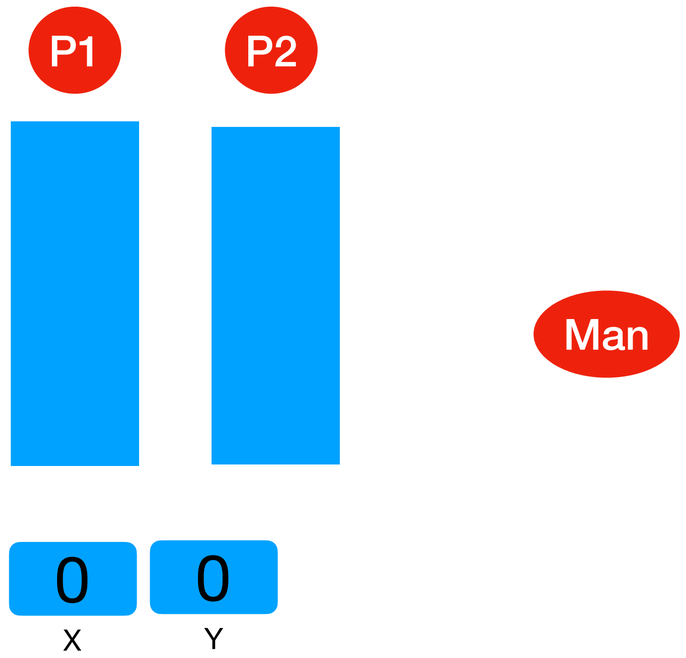
Persistent reachability problem reduces to crash free reachability in a new program



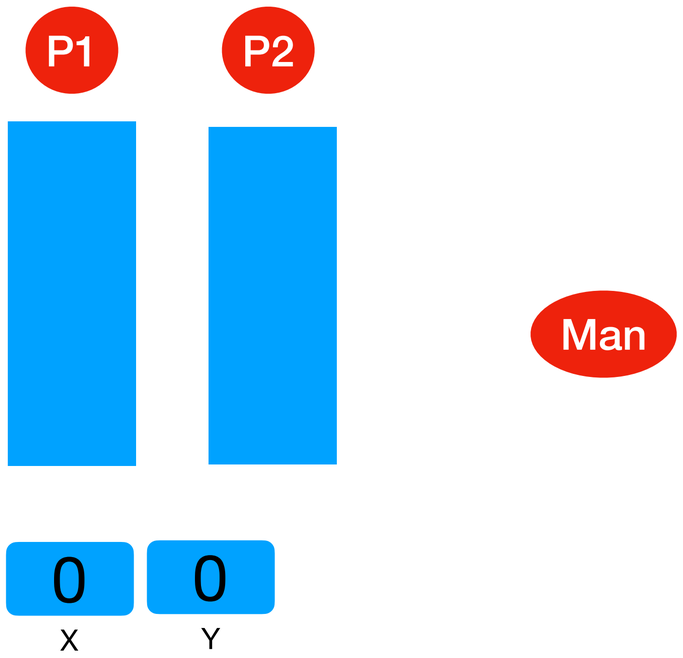
Translation employs a guess and verify technique

Ensure that the guessed writes are not overwritten

Persistent Memory Reachability



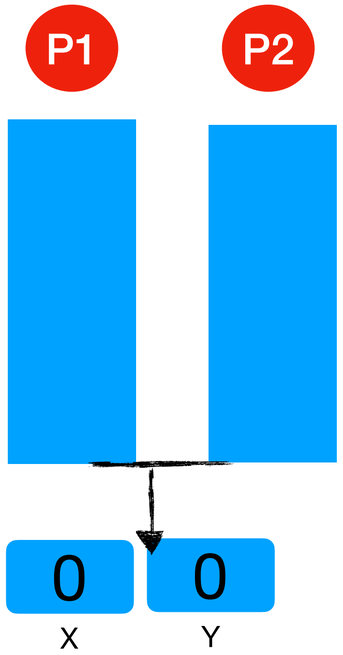
Persistent Memory Reachability



Man

Idea involves the manager speculating a write that will persist

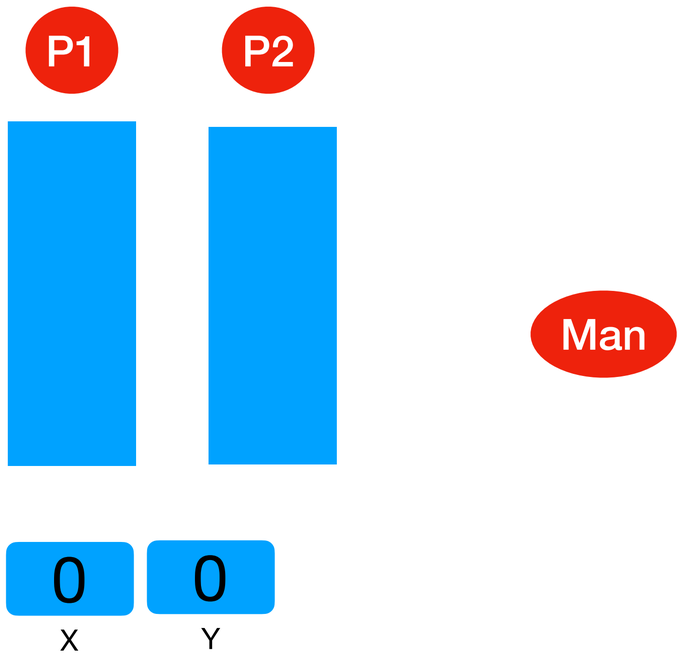
Persistent Memory Reachability



Man

Manager cannot observe all writes

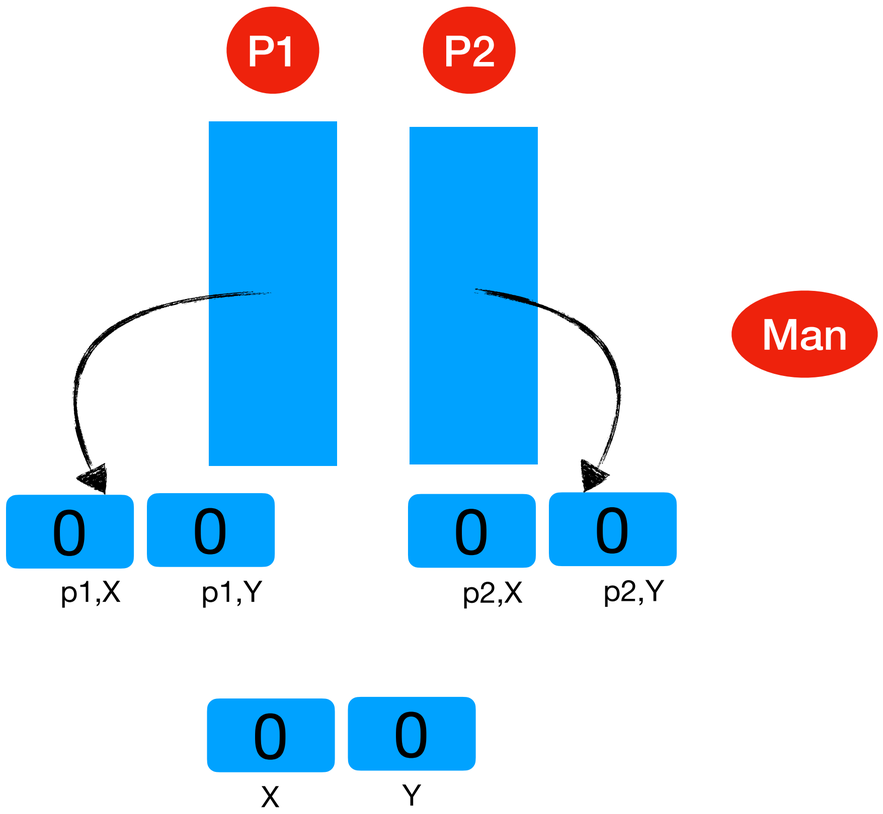
Persistent Memory Reachability



Man

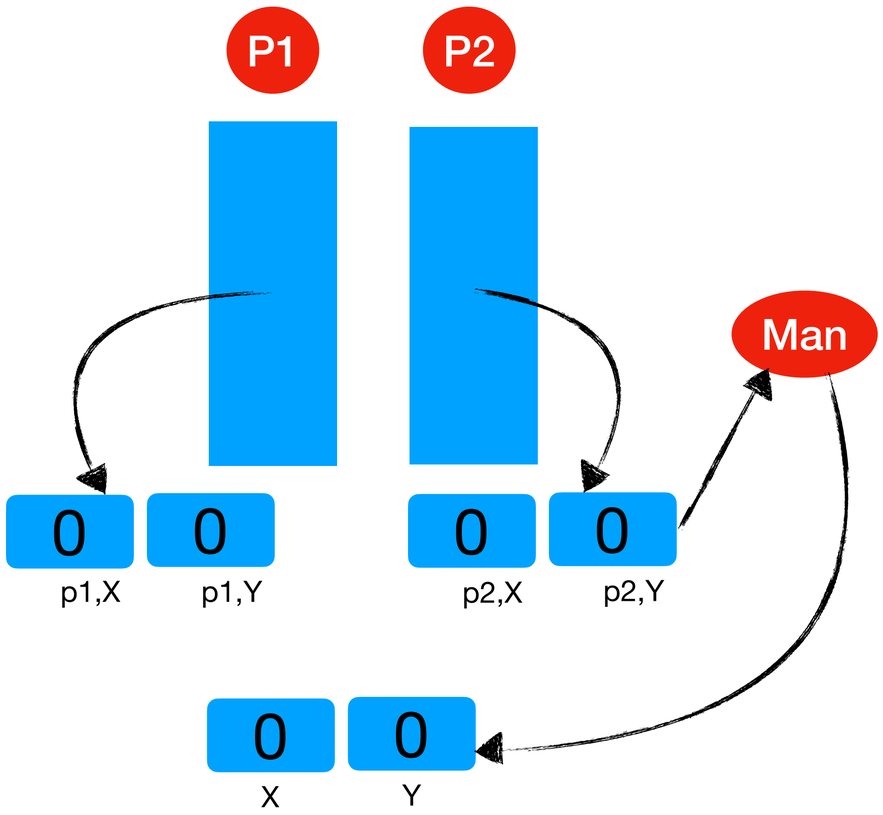
Introduce per process memory

Persistent Memory Reachability



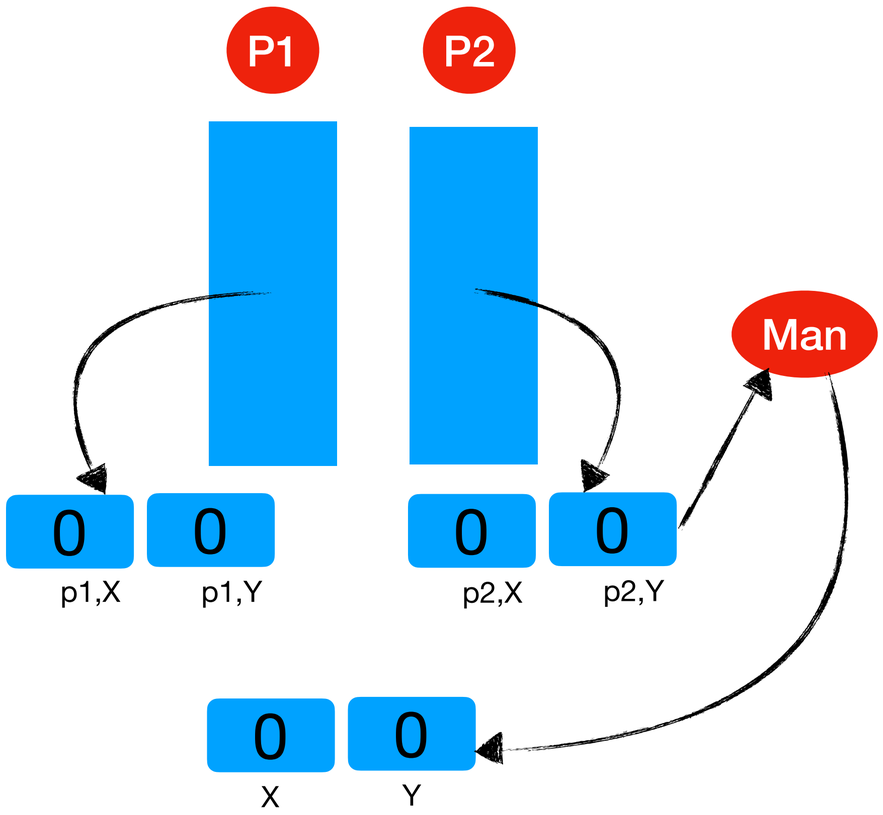
Threads write to their copy

Persistent Memory Reachability



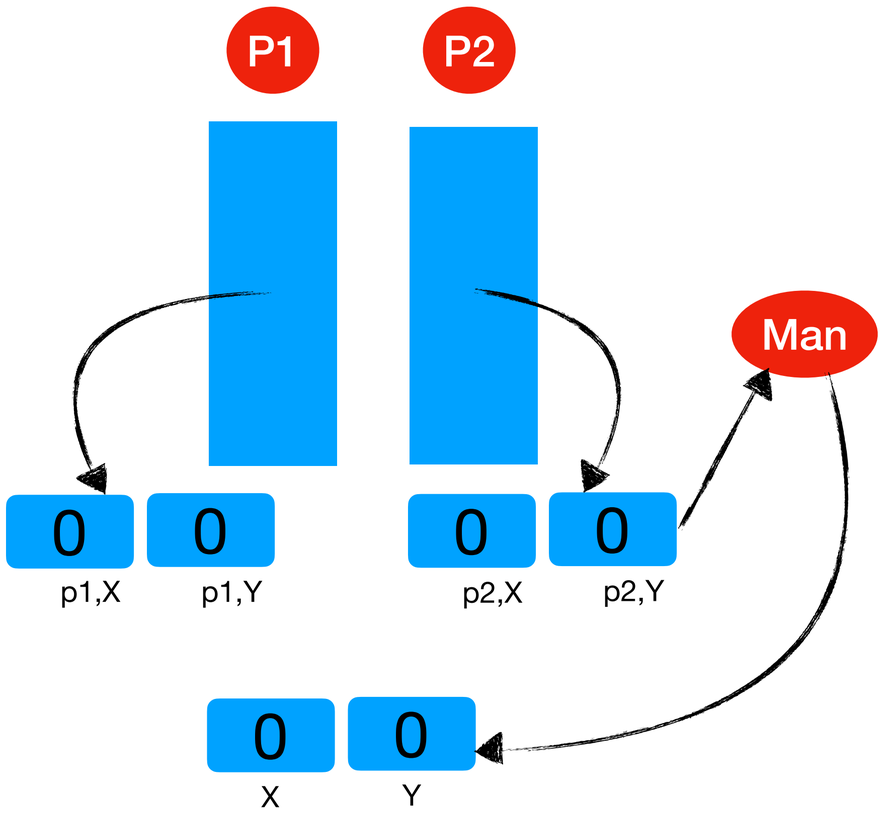
Manager transfers to the main memory

Persistent Memory Reachability



Needs to ensure update order is maintained

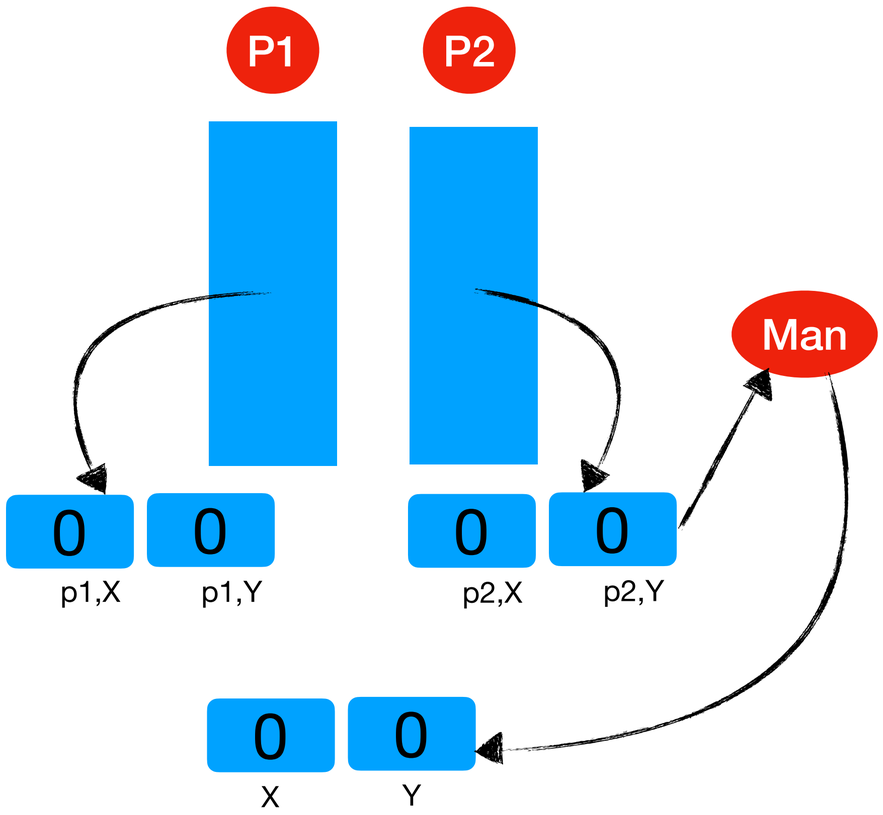
Persistent Memory Reachability



Needs to ensure update order is maintained

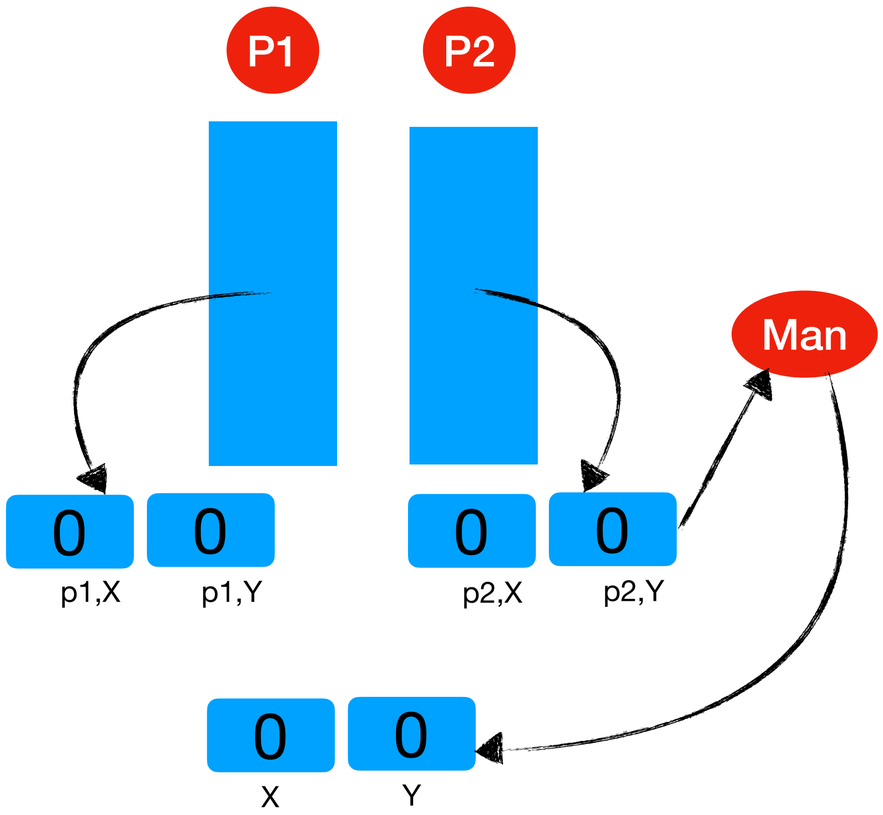
No more updates during a copy

Persistent Memory Reachability



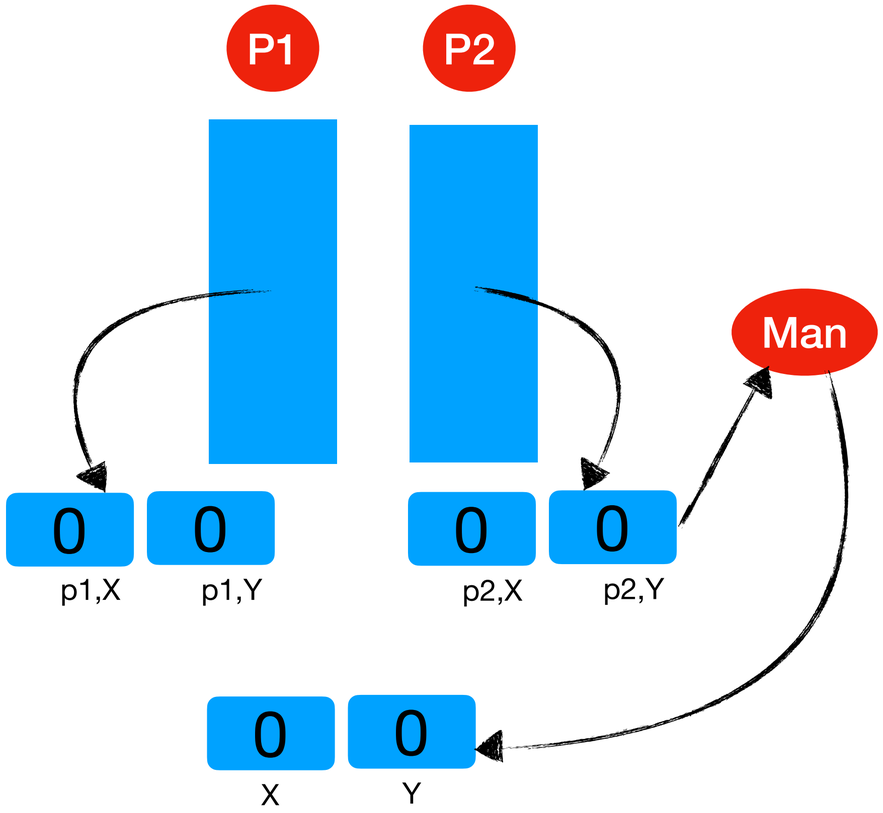
Reads now are from the copy or the main memory

Persistent Memory Reachability



Manager non-deterministically picks a write that will persist

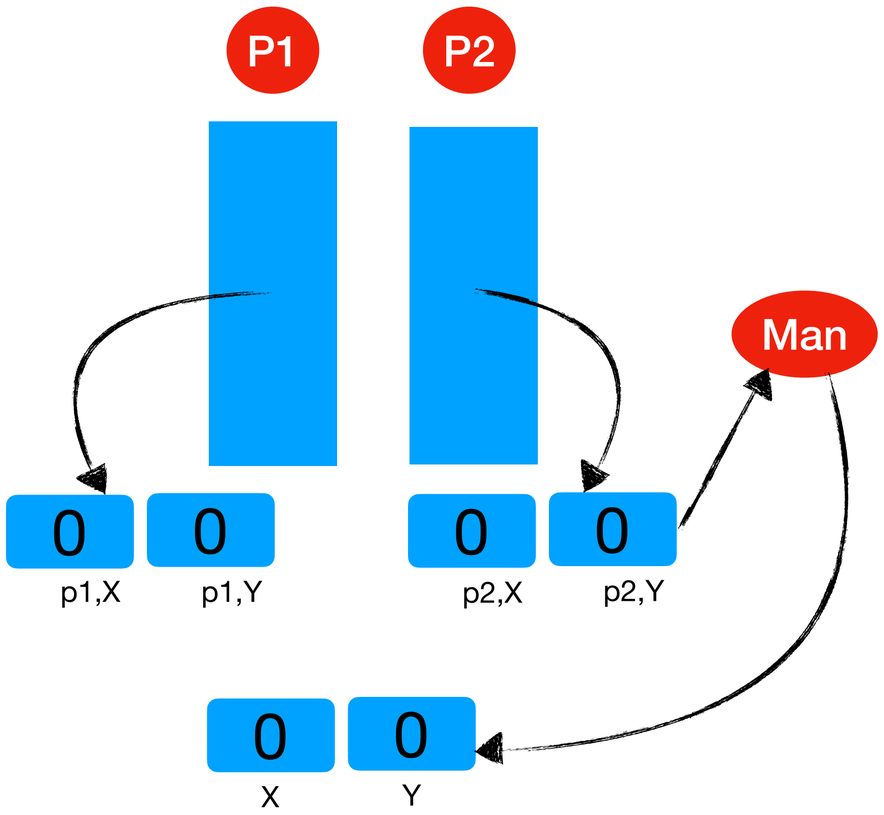
Persistent Memory Reachability



Manager non-deterministically picks a write that will persist

Needs to ensure the value is not over-written

Persistent Memory Reachability



Manager non-deterministically picks a write that will persist

Needs to ensure the value is not over-written

Frozen write

Spoilers

Spoilers

Frozen write is spoilt by certain bad patterns

Manager tries to avoid these spoilers / bad patterns

Spoilers

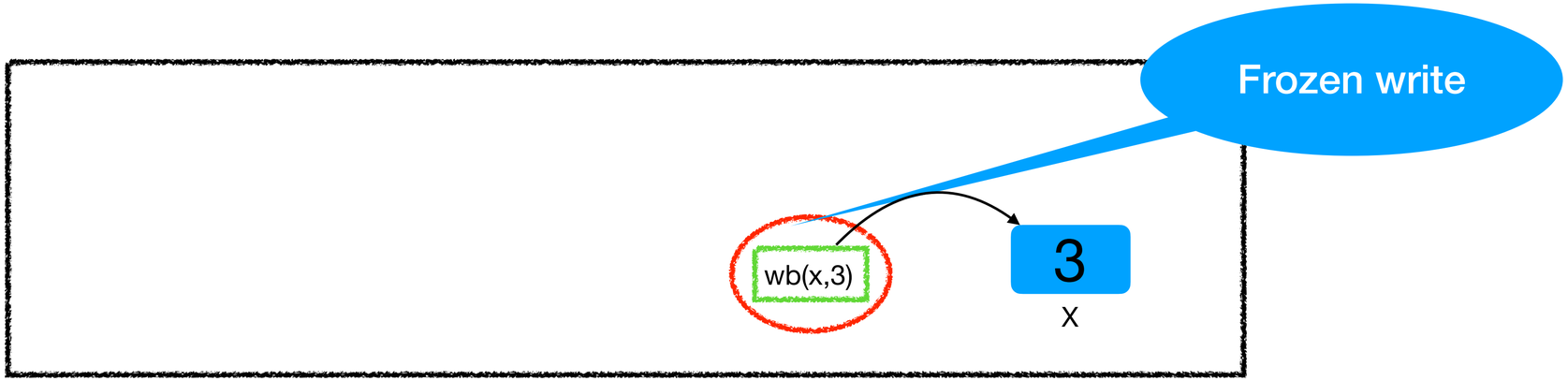
Frozen write is spoilt by certain bad patterns

Manager tries to avoid these spoilers / bad patterns

Spoilers

Frozen write is spoilt by certain bad patterns

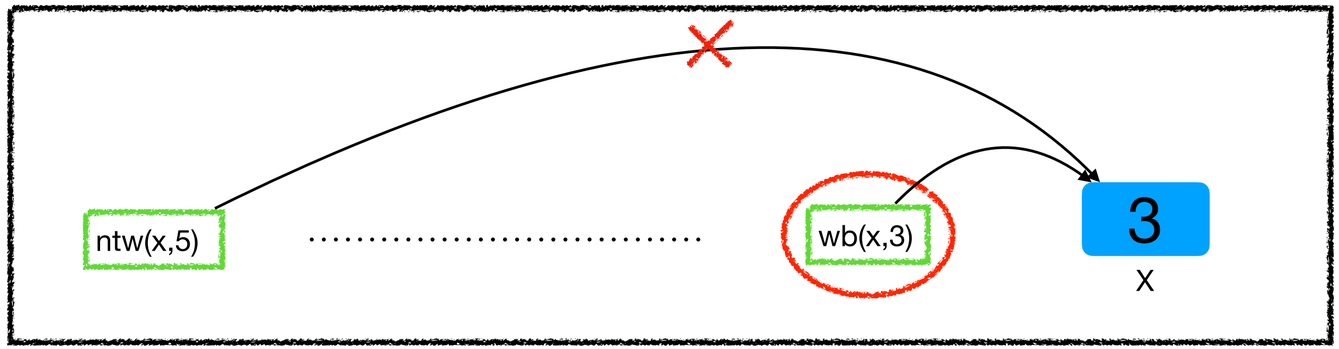
Manager tries to avoid these spoilers / bad patterns



Spoilers

Frozen write is spoilt by certain bad patterns

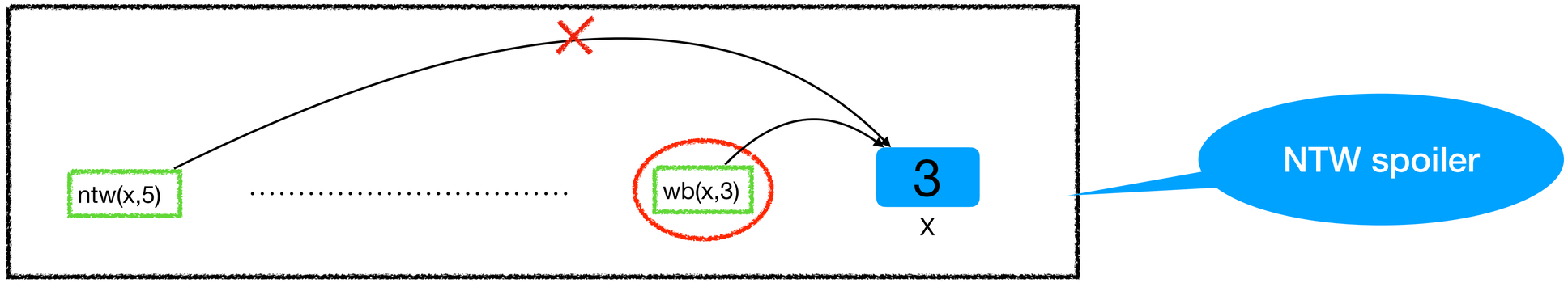
Manager tries to avoid these spoilers / bad patterns



Spoilers

Frozen write is spoilt by certain bad patterns

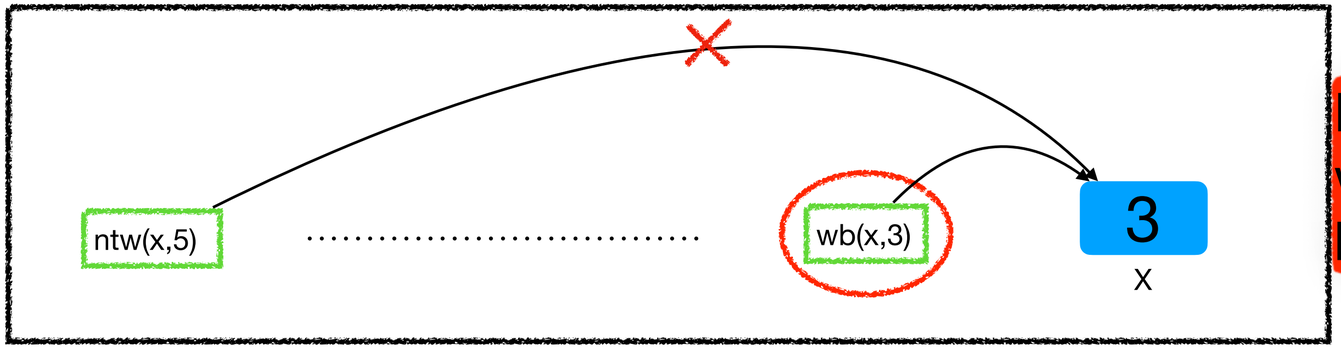
Manager tries to avoid these spoilers / bad patterns



Spoilers

Frozen write is spoilt by certain bad patterns

Manager tries to avoid these spoilers / bad patterns

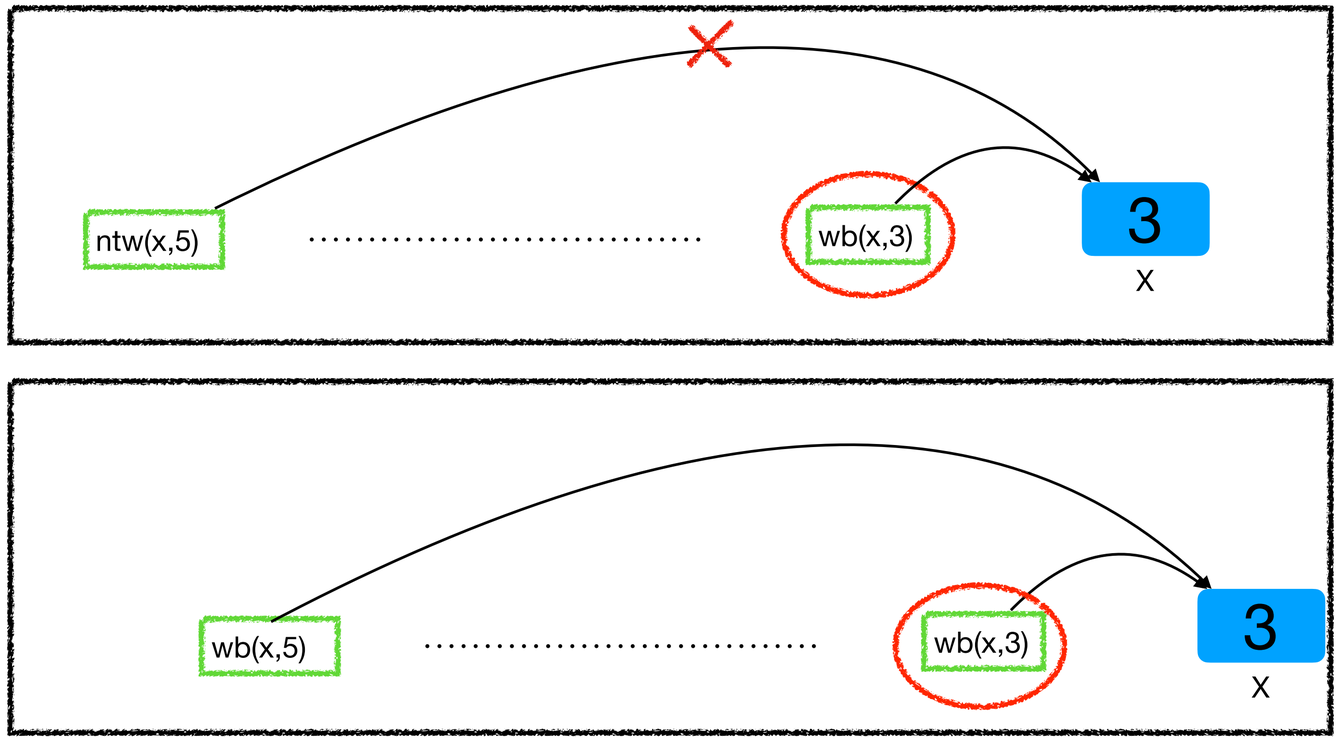


Manager can miss an ntw writes esp if it is followed by a wb write

Spoilers

Frozen write is spoilt by certain bad patterns

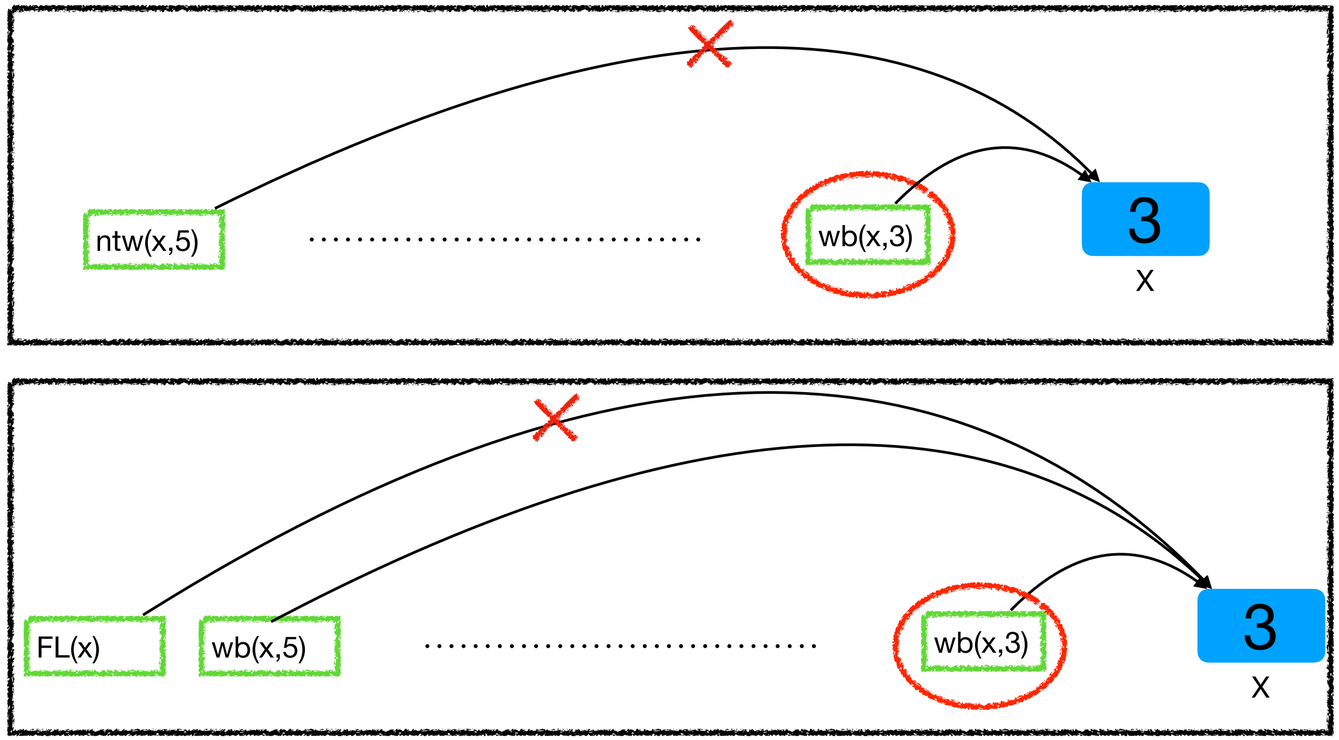
Manager tries to avoid these spoilers / bad patterns



Spoilers

Frozen write is spoilt by certain bad patterns

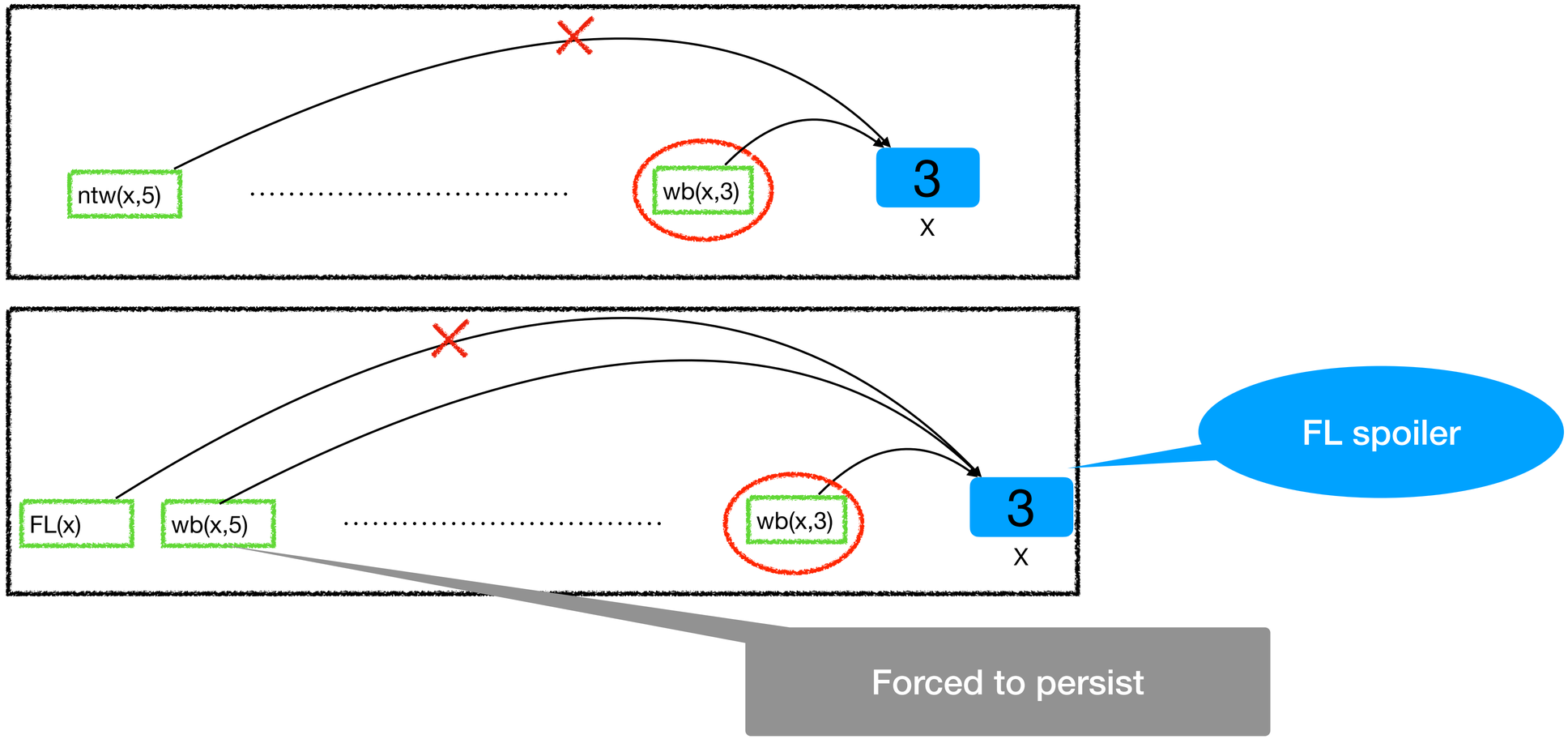
Manager tries to avoid these spoilers / bad patterns



Spoilers

Frozen write is spoilt by certain bad patterns

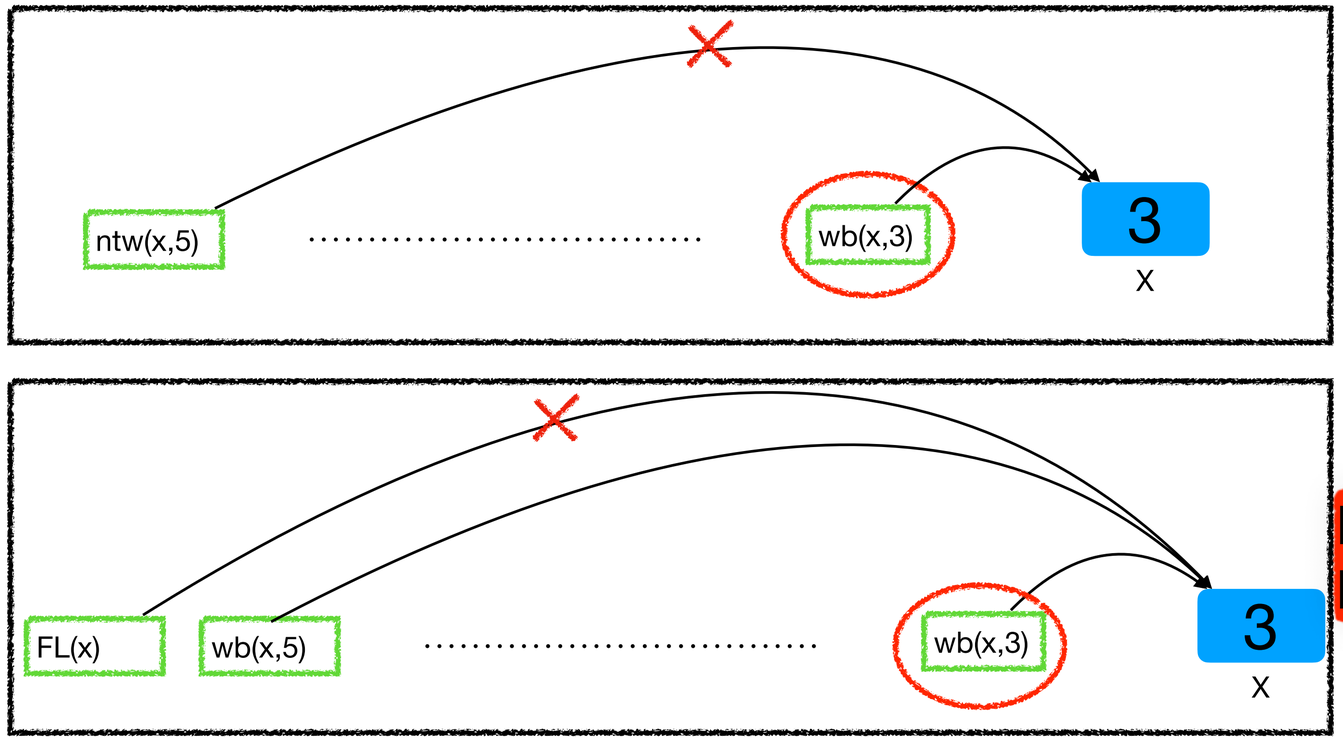
Manager tries to avoid these spoilers / bad patterns



Spoilers

Frozen write is spoilt by certain bad patterns

Manager tries to avoid these spoilers / bad patterns

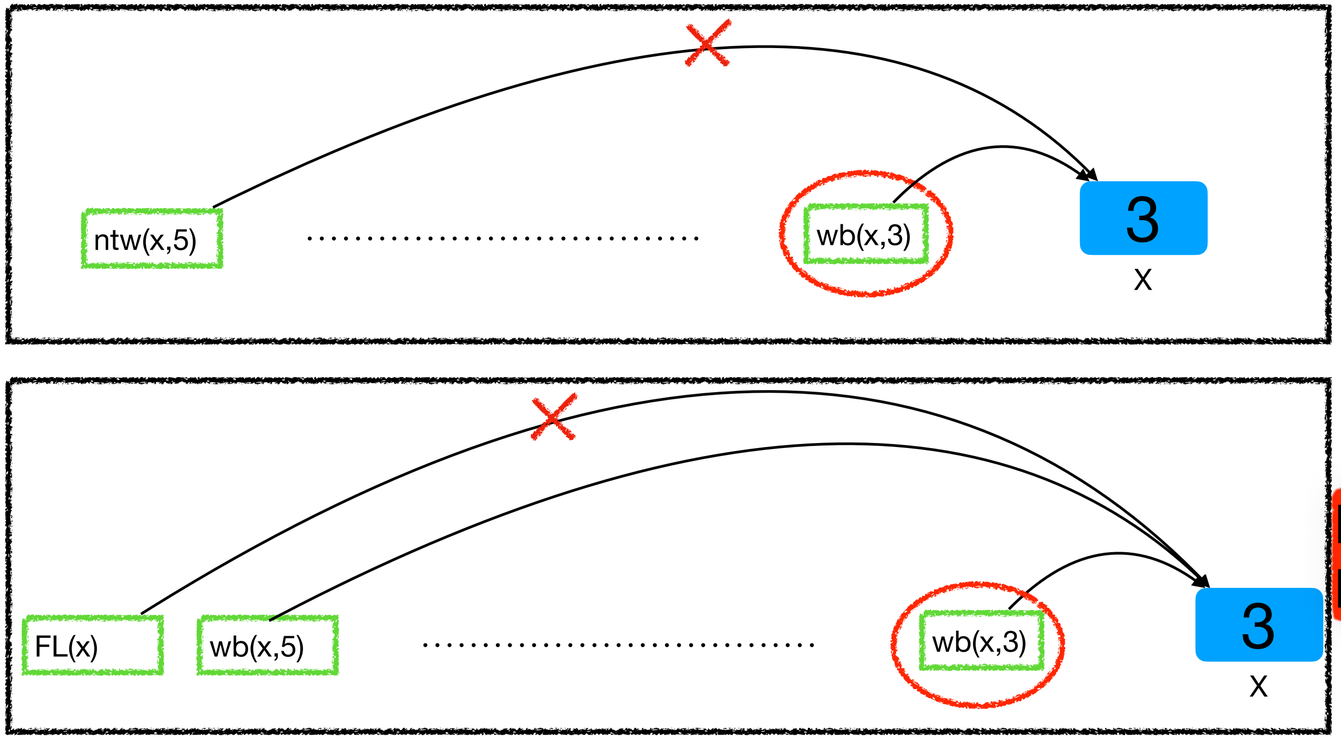


Manager cannot observe Fl operations

Spoilers

Frozen write is spoilt by certain bad patterns

Manager tries to avoid these spoilers / bad patterns



Manager cannot observe Fl operations

There are other spoilers involving Fo, we wont consider them

Detecting Spoilers

Detecting Spoilers

Threads

Detecting Spoilers

Threads

- Speculates the position of freeze for each variable
- Tracks potential spoilers to report to the manager

Detecting Spoilers

Threads

- Speculates the position of freeze for each variable
- Tracks potential spoilers to report to the manager

Manager

Detecting Spoilers

Threads

- Speculates the position of freeze for each variable
- Tracks potential spoilers to report to the manager

Manager

- Verifies the speculation of the threads
- Ensures that the potential spoilers are never seen

Detecting Spoilers

Threads

Difficult in presence of re-orderings

Speculates the position of freeze for each variable

Tracks potential spoilers to report to the manager

Manager

Verifies the speculation of the threads

Ensures that the potential spoilers are never seen

Detecting Spoilers

Threads

Difficult in presence of re-orderings

Speculates the position of freeze for each variable

Tracks potential spoilers to report to the manager

Manager

Verifies the speculation of the threads

Ensures that the potential spoilers are never seen

Persistent memory reachability reduces to crash free reachability

VERIFYING EX86 WITH PERSISTENCY

All stable processes we shall predict. all unstable processes we shall control - Benjamin Franklin

- Persistent Memory Reachability

- Crash Free Reachability

What About Crash-free Reachability?

What About Crash-free Reachability?

Crash-free reachability is undecidable

What About Crash-free Reachability?

Crash-free reachability is undecidable

Reduction from well known Post Correspondence Problem

What About Crash-free Reachability?

Crash-free reachability is undecidable

Reduction from well known Post Correspondence Problem

Crux of the reduction involves ability to implement alternating bit protocol

What About Crash-free Reachability?

Crash-free reachability is undecidable

Reduction from well known Post Correspondence Problem

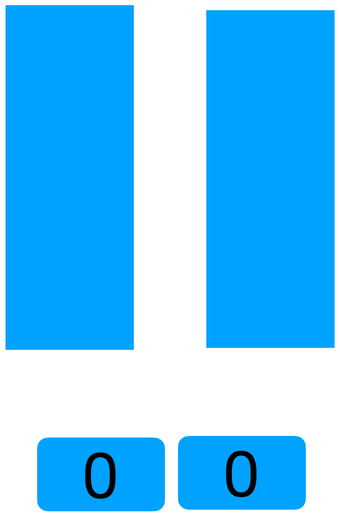
Crux of the reduction involves ability to implement alternating bit protocol

Thread 1:

```
1 repeat
2 |   Wb(x, 1);
3 |   Wb(y, 1);
4 until n times;
```

Thread 2:

```
1 repeat
2 |   assert(x = 0);
3 |   RMW(x, 1, 0);
4 |   assert(y = 0);
5 |   RMW(y, 1, 0);
6 |   assert(x = 0);
7 until n times;
8 ‘
```



What About Crash-free Reachability?

Crash-free reachability is undecidable

Reduction from well known Post Correspondence Problem

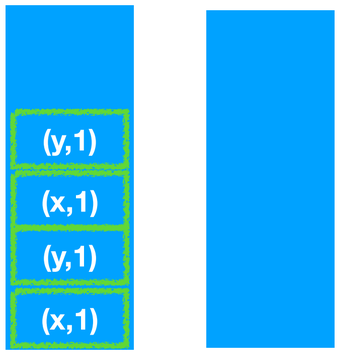
Crux of the reduction involves ability to implement alternating bit protocol

Thread 1:

```
1 repeat
2 |   Wb(x, 1);
3 |   Wb(y, 1);
4 until n times;
```

Thread 2:

```
1 repeat
2 |   assert(x = 0);
3 |   RMW(x, 1, 0);
4 |   assert(y = 0);
5 |   RMW(y, 1, 0);
6 |   assert(x = 0);
7 until n times;
8 ‘
```



0 0

What About Crash-free Reachability?

Crash-free reachability is undecidable

Reduction from well known Post Correspondence Problem

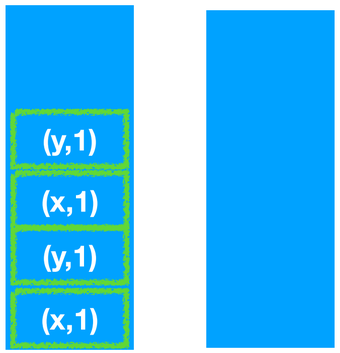
Crux of the reduction involves ability to implement alternating bit protocol

Thread 1:

```
1 repeat
2 |   Wb(x, 1);
3 |   Wb(y, 1);
4 until n times;
```

Thread 2:

```
1 repeat
2 |   assert(x = 0);
3 |   RMW(x, 1, 0);
4 |   assert(y = 0);
5 |   RMW(y, 1, 0);
6 |   assert(x = 0);
7 until n times;
8 ‘
```



0 0

What About Crash-free Reachability?

Crash-free reachability is undecidable

Reduction from well known Post Correspondence Problem

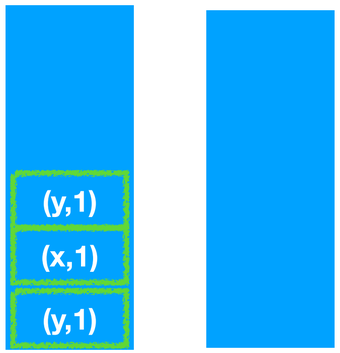
Crux of the reduction involves ability to implement alternating bit protocol

Thread 1:

```
1 repeat
2 |   Wb(x, 1);
3 |   Wb(y, 1);
4 until n times;
```

Thread 2:

```
1 repeat
2 |   assert(x = 0);
3 |   RMW(x, 1, 0);
4 |   assert(y = 0);
5 |   RMW(y, 1, 0);
6 |   assert(x = 0);
7 until n times;
8 ‘
```



What About Crash-free Reachability?

Crash-free reachability is undecidable

Reduction from well known Post Correspondence Problem

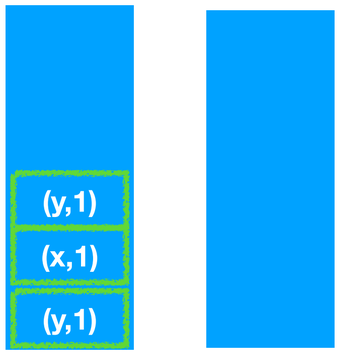
Crux of the reduction involves ability to implement alternating bit protocol

Thread 1:

```
1 repeat
2 |   Wb(x, 1);
3 |   Wb(y, 1);
4 until n times;
```

Thread 2:

```
1 repeat
2 |   assert(x = 0);
3 |   RMW(x, 1, 0);
4 |   assert(y = 0);
5 |   RMW(y, 1, 0);
6 |   assert(x = 0);
7 until n times;
8 ‘
```



What About Crash-free Reachability?

Crash-free reachability is undecidable

Reduction from well known Post Correspondence Problem

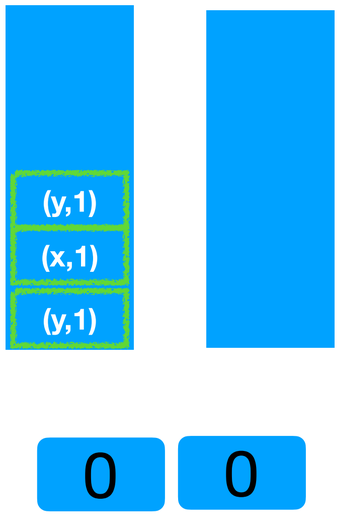
Crux of the reduction involves ability to implement alternating bit protocol

Thread 1:

```
1 repeat
2 |   Wb(x, 1);
3 |   Wb(y, 1);
4 until n times;
```

Thread 2:

```
1 repeat
2 |   assert(x = 0);
3 |   RMW(x, 1, 0);
4 |   assert(y = 0);
5 |   RMW(y, 1, 0);
6 |   assert(x = 0);
7 until n times;
8 ‘
```



What About Crash-free Reachability?

Crash-free reachability is undecidable

Reduction from well known Post Correspondence Problem

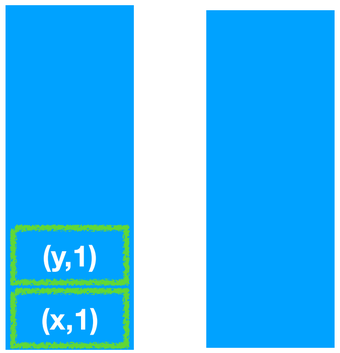
Crux of the reduction involves ability to implement alternating bit protocol

Thread 1:

```
1 repeat
2 |   Wb(x, 1);
3 |   Wb(y, 1);
4 until n times;
```

Thread 2:

```
1 repeat
2 |   assert(x = 0);
3 |   RMW(x, 1, 0);
4 |   assert(y = 0);
5 |   RMW(y, 1, 0);
6 |   assert(x = 0);
7 until n times;
8 ‘
```



0 1

What About Crash-free Reachability?

Crash-free reachability is undecidable

Reduction from well known Post Correspondence Problem

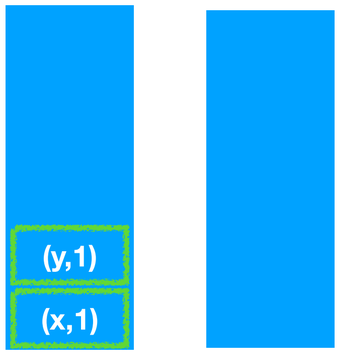
Crux of the reduction involves ability to implement alternating bit protocol

Thread 1:

```
1 repeat
2 |   Wb(x, 1);
3 |   Wb(y, 1);
4 until n times;
```

Thread 2:

```
1 repeat
2 |   assert(x = 0);
3 |   RMW(x, 1, 0);
4 |   assert(y = 0);
5 |   RMW(y, 1, 0);
6 |   assert(x = 0);
7 until n times;
8 '
```



What About Crash-free Reachability?

Crash-free reachability is undecidable

Reduction from well known Post Correspondence Problem

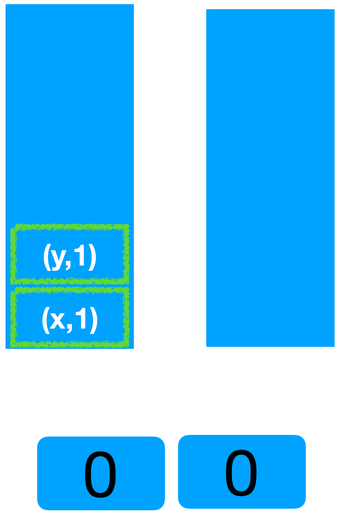
Crux of the reduction involves ability to implement alternating bit protocol

Thread 1:

```
1 repeat
2 |   Wb(x, 1);
3 |   Wb(y, 1);
4 until n times;
```

Thread 2:

```
1 repeat
2 |   assert(x = 0);
3 |   RMW(x, 1, 0);
4 |   assert(y = 0);
5 |   RMW(y, 1, 0);
6 |   assert(x = 0);
7 until n times;
8 ‘
```



Thread-2 reaches the end only if it did not miss any write

PCP: $U = \{u_1, \ldots u_\ell\} \quad V = \{v_1, \ldots v_\ell\}$

$\exists i_1 \ldots i_n : u_{i_1} \cdot u_{i_2} \ldots u_{i_n} = v_{i_1} \cdot v_{i_2} \ldots v_{i_n}$

PCP: $U = \{u_1, \ldots u_\ell\} \quad V = \{v_1, \ldots v_\ell\}$

$$\exists i_1 \ldots i_n : u_{i_1} \cdot u_{i_2} \ldots u_{i_n} = v_{i_1} \cdot v_{i_2} \ldots v_{i_n}$$

Crash-free reachability

Algorithm 1: PCPGen

```
1  Global Vars x, y, s, t
2  Local Vars i, j, flg := true
3  while ⋆ do
4  |   Let i ∈ [1, ℓ]
5  |   s :=ntw u_i
6  |   t :=ntw v_i
7  |   j := |u_i| + |v_i|
8  |   while j > 0 do
9  |   |   x := 1
10 |   |   y := 1
11 |   |   j = j − 1
12 x := #
```

Algorithm 2: PCPVerif

```
1  Global Vars x, y, s, t
2  Local Vars a, b
3  while (a ≠ #) do
4  |   rmw(x, 1, 0)
5  |   rmw(y, 0, 0)
6  |   Let b ∈ Σ
7  |   rmw(s, b, 0)
8  |   rmw(t, b, 0)
9  |   rmw(y, 1, 0)
10 |   rmw(x, 0, 0)
11 |   a := x
12 Halt
```

PCP: $U = \{u_1, \ldots u_\ell\} \quad V = \{v_1, \ldots v_\ell\}$

$$\exists i_1 \ldots i_n : u_{i_1} \cdot u_{i_2} \ldots u_{i_n} = v_{i_1} \cdot v_{i_2} \ldots v_{i_n}$$

Crash-free reachability

Algorithm 1: PCPGen

```
1  Global Vars x, y, s, t
2  Local Vars i, j, flg := true
3  while ⋆ do
4  |   Let i ∈ [1, ℓ]
5  |   s :=ntw u_i
6  |   t :=ntw v_i
7  |   j := |u_i| + |v_i|
8  |   while j > 0 do
9  |   |   x := 1
10 |   |   y := 1
11 |   |   j = j − 1
12 x := #
```

Algorithm 2: PCPVerif

```
1  Global Vars x, y, s, t
2  Local Vars a, b
3  while (a ≠ #) do
4  |   rmw(x, 1, 0)
5  |   rmw(y, 0, 0)
6  |   Let b ∈ Σ
7  |   rmw(s, b, 0)
8  |   rmw(t, b, 0)
9  |   rmw(y, 1, 0)
10 |   rmw(x, 0, 0)
11 |   a := x
12 Halt
```

PCPGen

PCP: $U = \{u_1, \ldots u_\ell\}$ $V = \{v_1, \ldots v_\ell\}$

$$\exists i_1 \ldots i_n : u_{i_1} \cdot u_{i_2} \ldots u_{i_n} = v_{i_1} \cdot v_{i_2} \ldots v_{i_n}$$

Crash-free reachability

Algorithm 1: PCPGen

```
1  Global Vars x, y, s, t
2  Local Vars i, j, flg := true
3  while ⋆ do
4  |  Let i ∈ [1, ℓ]
5  |  s :=ntw u_i
6  |  t :=ntw v_i
7  |  j := |u_i| + |v_i|
8  |  while j > 0 do
9  |  |  x := 1
10 |  |  y := 1
11 |  |  j = j − 1
12 x := #
```

Algorithm 2: PCPVerif

```
1  Global Vars x, y, s, t
2  Local Vars a, b
3  while (a ≠ #) do
4  |  rmw(x, 1, 0)
5  |  rmw(y, 0, 0)
6  |  Let b ∈ Σ
7  |  rmw(s, b, 0)
8  |  rmw(t, b, 0)
9  |  rmw(y, 1, 0)
10 |  rmw(x, 0, 0)
11 |  a := x
12 Halt
```

PCPGen

Pick an index

PCP: $U = \{u_1, \ldots u_\ell\} \quad V = \{v_1, \ldots v_\ell\}$

$$\exists i_1 \ldots i_n : u_{i_1} \cdot u_{i_2} \ldots u_{i_n} = v_{i_1} \cdot v_{i_2} \ldots v_{i_n}$$

Crash-free reachability

Algorithm 1: PCPGen

```
1  Global Vars x, y, s, t
2  Local Vars i, j, flg := true
3  while ⋆ do
4  |   Let i ∈ [1, ℓ]
5  |   s :=ntw u_i
6  |   t :=ntw v_i
7  |   j := |u_i| + |v_i|
8  |   while j > 0 do
9  |   |   x := 1
10 |   |   y := 1
11 |   |   j = j − 1
12 x := #
```

Algorithm 2: PCPVerif

```
1  Global Vars x, y, s, t
2  Local Vars a, b
3  while (a ≠ #) do
4  |   rmw(x, 1, 0)
5  |   rmw(y, 0, 0)
6  |   Let b ∈ Σ
7  |   rmw(s, b, 0)
8  |   rmw(t, b, 0)
9  |   rmw(y, 1, 0)
10 |   rmw(x, 0, 0)
11 |   a := x
12 Halt
```

PCPGen

Pick an index

Writes the corresponding words as ntw writes

PCP: $U = \{u_1, \ldots u_\ell\} \quad V = \{v_1, \ldots v_\ell\}$

$$\exists i_1 \ldots i_n : u_{i_1} \cdot u_{i_2} \ldots u_{i_n} = v_{i_1} \cdot v_{i_2} \ldots v_{i_n}$$

Crash-free reachability

Algorithm 1: PCPGen

```
1  Global Vars x, y, s, t
2  Local Vars i, j, flg := true
3  while ⋆ do
4      Let i ∈ [1, ℓ]
5      s :=ntw u_i
6      t :=ntw v_i
7      j := |u_i| + |v_i|
8      while j > 0 do
9          x := 1
10         y := 1
11         j = j − 1
12 x := #
```

Algorithm 2: PCPVerif

```
1  Global Vars x, y, s, t
2  Local Vars a, b
3  while (a ≠ #) do
4      rmw(x, 1, 0)
5      rmw(y, 0, 0)
6      Let b ∈ Σ
7      rmw(s, b, 0)
8      rmw(t, b, 0)
9      rmw(y, 1, 0)
10     rmw(x, 0, 0)
11     a := x
12 Halt
```

PCPGen

Pick an index

Writes the corresponding words as ntw writes

Encodes the size into the alt-bit protocol

PCP: $U = \{u_1, \ldots u_\ell\} \quad V = \{v_1, \ldots v_\ell\}$

$$\exists i_1 \ldots i_n : u_{i_1} \cdot u_{i_2} \ldots u_{i_n} = v_{i_1} \cdot v_{i_2} \ldots v_{i_n}$$

Crash-free reachability

Algorithm 1: PCPGen

```
1  Global Vars x, y, s, t
2  Local Vars i, j, flg := true
3  while ⋆ do
4  |   Let i ∈ [1, ℓ]
5  |   s :=ntw u_i
6  |   t :=ntw v_i
7  |   j := |u_i| + |v_i|
8  |   while j > 0 do
9  |   |   x := 1
10 |   |   y := 1
11 |   |   j = j − 1
12 x := #
```

Algorithm 2: PCPVerif

```
1  Global Vars x, y, s, t
2  Local Vars a, b
3  while (a ≠ #) do
4  |   rmw(x, 1, 0)
5  |   rmw(y, 0, 0)
6  |   Let b ∈ Σ
7  |   rmw(s, b, 0)
8  |   rmw(t, b, 0)
9  |   rmw(y, 1, 0)
10 |   rmw(x, 0, 0)
11 |   a := x
12 Halt
```

PCPGen

Pick an index

Writes the corresponding words as ntw writes

Encodes the size into the alt-bit protocol

PCP: $U = \{u_1, \ldots u_\ell\} \quad V = \{v_1, \ldots v_\ell\}$

$$\exists i_1 \ldots i_n : u_{i_1} \cdot u_{i_2} \ldots u_{i_n} = v_{i_1} \cdot v_{i_2} \ldots v_{i_n}$$

Crash-free reachability

Algorithm 1: PCPGen

```
1  Global Vars x, y, s, t
2  Local Vars i, j, flg := true
3  while ⋆ do
4  |   Let i ∈ [1, ℓ]
5  |   s :=ntw u_i
6  |   t :=ntw v_i
7  |   j := |u_i| + |v_i|
8  |   while j > 0 do
9  |   |   x := 1
10 |   |   y := 1
11 |   |   j = j − 1
12 x := #
```

Algorithm 2: PCPVerif

```
1  Global Vars x, y, s, t
2  Local Vars a, b
3  while (a ≠ #) do
4  |   rmw(x, 1, 0)
5  |   rmw(y, 0, 0)
6  |   Let b ∈ Σ
7  |   rmw(s, b, 0)
8  |   rmw(t, b, 0)
9  |   rmw(y, 1, 0)
10 |   rmw(x, 0, 0)
11 |   a := x
12 Halt
```

PCPGen

Pick an index

Writes the corresponding words as ntw writes

Encodes the size into the alt-bit protocol

PCPVerif

PCP: $U = \{u_1, \ldots u_\ell\} \quad V = \{v_1, \ldots v_\ell\}$

$$\exists i_1 \ldots i_n : u_{i_1} \cdot u_{i_2} \ldots u_{i_n} = v_{i_1} \cdot v_{i_2} \ldots v_{i_n}$$

Crash-free reachability

Algorithm 1: PCPGen

```
1  Global Vars x, y, s, t
2  Local Vars i, j, flg := true
3  while ⋆ do
4  |   Let i ∈ [1, ℓ]
5  |   s :=ntw u_i
6  |   t :=ntw v_i
7  |   j := |u_i| + |v_i|
8  |   while j > 0 do
9  |   |   x := 1
10 |   |   y := 1
11 |   |   j = j − 1
12 x := #
```

Algorithm 2: PCPVerif

```
1  Global Vars x, y, s, t
2  Local Vars a, b
3  while (a ≠ #) do
4  |   rmw(x, 1, 0)
5  |   rmw(y, 0, 0)
6  |   Let b ∈ Σ
7  |   rmw(s, b, 0)
8  |   rmw(t, b, 0)
9  |   rmw(y, 1, 0)
10 |   rmw(x, 0, 0)
11 |   a := x
12 Halt
```

PCPGen

Pick an index

Writes the corresponding words as ntw writes

Encodes the size into the alt-bit protocol

PCPVerif

Alt bit ensures no symbol is lost

PCP: $U = \{u_1, \ldots u_\ell\} \quad V = \{v_1, \ldots v_\ell\}$

$$\exists i_1 \ldots i_n : u_{i_1} \cdot u_{i_2} \ldots u_{i_n} = v_{i_1} \cdot v_{i_2} \ldots v_{i_n}$$

Crash-free reachability

Algorithm 1: PCPGen

```
1  Global Vars x, y, s, t
2  Local Vars i, j, flg := true
3  while ⋆ do
4  |   Let i ∈ [1, ℓ]
5  |   s :=ntw u_i
6  |   t :=ntw v_i
7  |   j := |u_i| + |v_i|
8  |   while j > 0 do
9  |   |   x := 1
10 |   |   y := 1
11 |   |   j = j − 1
12 x := #
```

Algorithm 2: PCPVerif

```
1  Global Vars x, y, s, t
2  Local Vars a, b
3  while (a ≠ #) do
4  |   rmw(x, 1, 0)
5  |   rmw(y, 0, 0)
6  |   Let b ∈ Σ
7  |   rmw(s, b, 0)
8  |   rmw(t, b, 0)
9  |   rmw(y, 1, 0)
10 |   rmw(x, 0, 0)
11 |   a := x
12 Halt
```

PCPGen

Pick an index

Writes the corresponding words as ntw writes

Encodes the size into the alt-bit protocol

PCPVerif

Alt bit ensures no symbol is lost

Verfies that the generated words are same

PCP: $U = \{u_1, \ldots u_\ell\} \quad V = \{v_1, \ldots v_\ell\}$

$$\exists i_1 \ldots i_n : u_{i_1} \cdot u_{i_2} \ldots u_{i_n} = v_{i_1} \cdot v_{i_2} \ldots v_{i_n}$$

Crash-free reachability

Algorithm 1: PCPGen

```
1  Global Vars x, y, s, t
2  Local Vars i, j, flg := true
3  while ⋆ do
4  |   Let i ∈ [1, ℓ]
5  |   s :=ntw u_i
6  |   t :=ntw v_i
7  |   j := |u_i| + |v_i|
8  |   while j > 0 do
9  |   |   x := 1
10 |   |   y := 1
11 |   |   j = j − 1
12 x := #
```

Algorithm 2: PCPVerif

```
1  Global Vars x, y, s, t
2  Local Vars a, b
3  while (a ≠ #) do
4  |   rmw(x, 1, 0)
5  |   rmw(y, 0, 0)
6  |   Let b ∈ Σ
7  |   rmw(s, b, 0)
8  |   rmw(t, b, 0)
9  |   rmw(y, 1, 0)
10 |   rmw(x, 0, 0)
11 |   a := x
12 Halt
```

PCPGen

Pick an index

Writes the corresponding words as ntw writes

Encodes the size into the alt-bit protocol

PCPVerif

Alt bit ensures no symbol is lost

Verfies that the generated words are same

Crash free reachability is undecidable

VERIFYING EX86 WITH PERSISTENCY

All stable processes we shall predict, all unstable processes we shall control - Benjamin Franklin

- Persistent Memory Reachability

- Crash Free Reachability

- Decidable Fragment

Alternation Bounded Reachability

Alternation Bounded Reachability

One source of undecidability is unbounded alternations between ntw and wb writes.

Alternation Bounded Reachability

K Alternation Bounded: An thread execution is k-alternation bounded if the thread alternates between wb and ntw writes at-most k times.

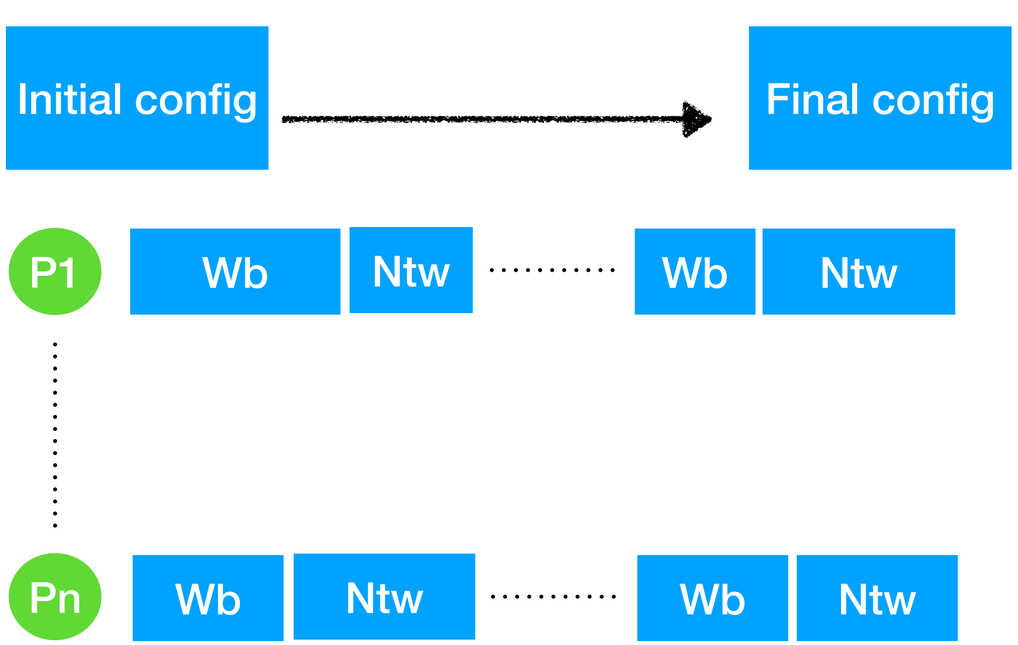
Alternation Bounded Reachability

K Alternation Bounded: An thread execution is k-alternation bounded if the thread alternates between wb and ntw writes at-most k times.

Alternation Bounded Reachability

K Alternation Bounded reachability asks if a final config can be reached by an execution in which every thread is k-alternation bounded

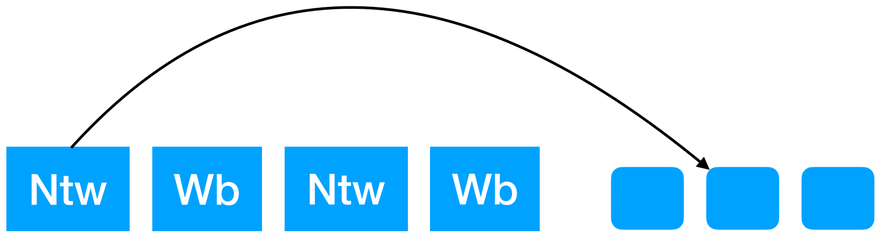
Alternation Bounded Reachability



K Alternation Bounded reachability asks if a final config can be reached by an execution in which every thread is k-alternation bounded

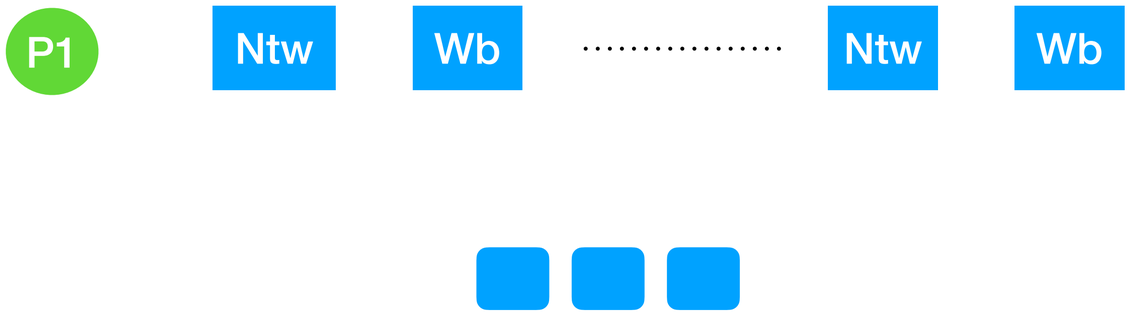
Decidability

Decidability



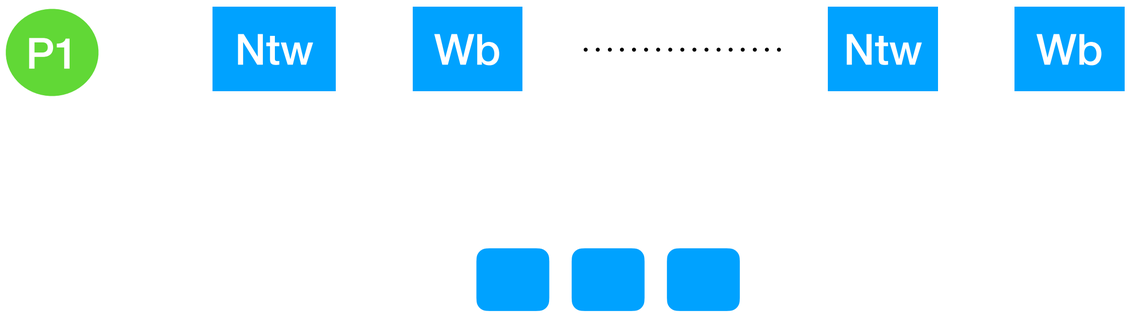
The writes between alternation blocks can re-order

Decidability



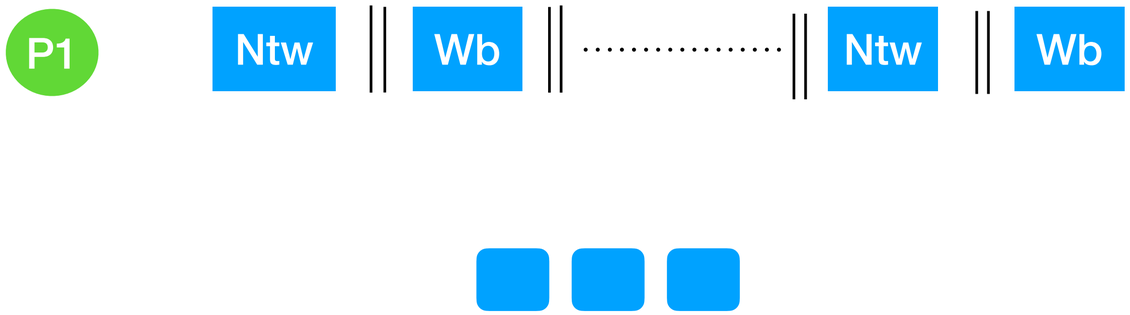
Execution within each block is like TSO or PSO

Decidability



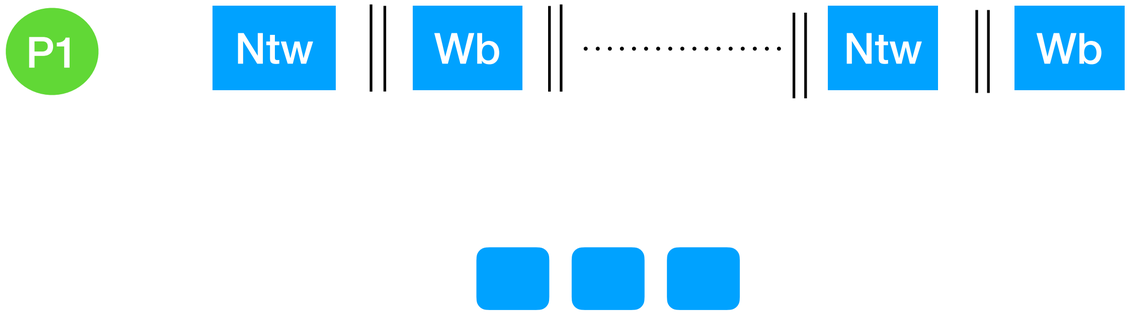
Decidability by reduction to reachability on PSO system

Decidability



Each alternation block is executed in parallel as a PSO thread

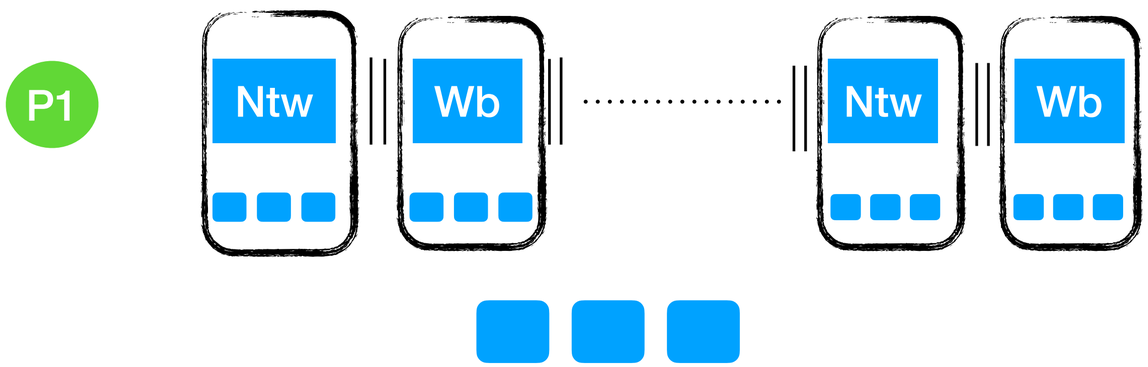
Decidability



Each alternation block is executed in parallel as a PSO thread

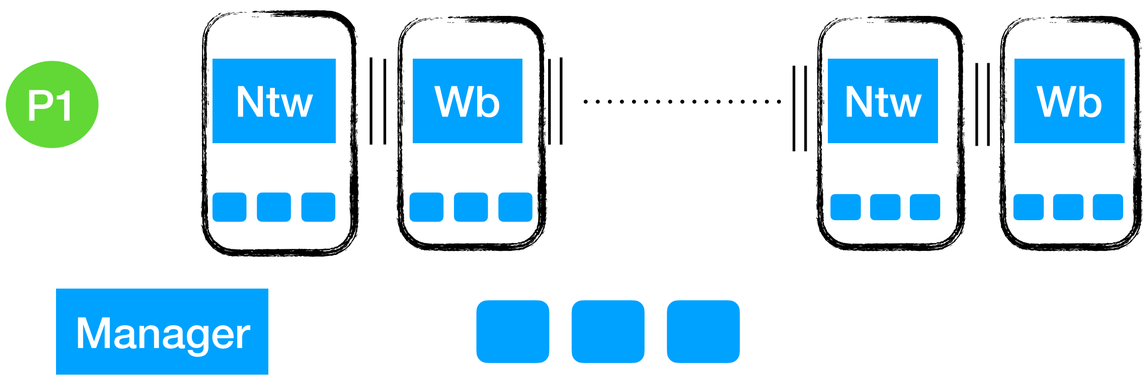
Later blocks depend on earlier blocks

Decidability



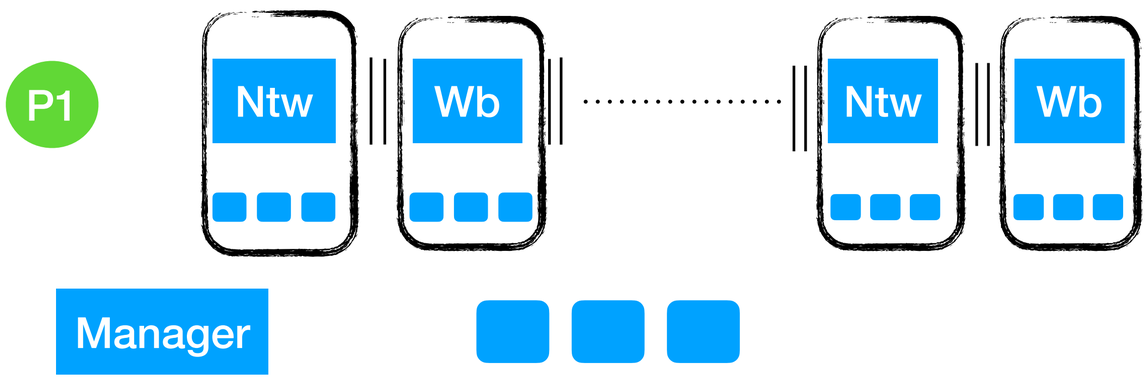
Memory is duplicated as per thread and per phase

Decidability



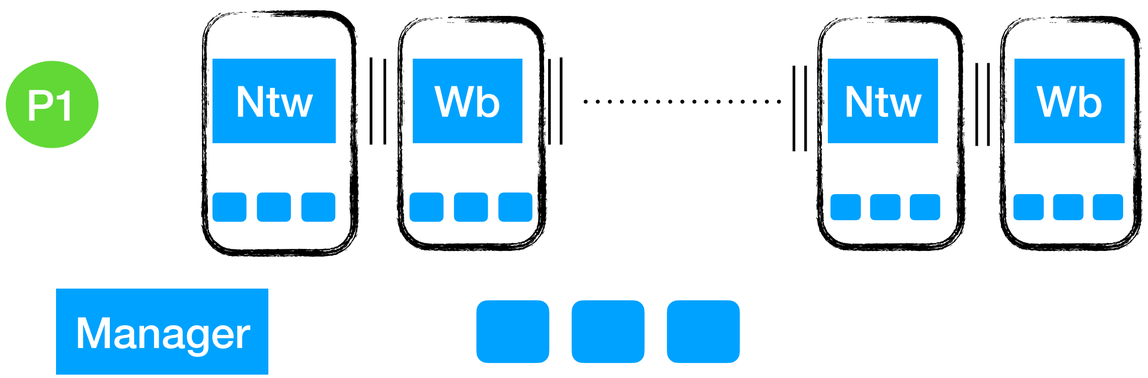
Memory is duplicated as per thread and per phase

Decidability



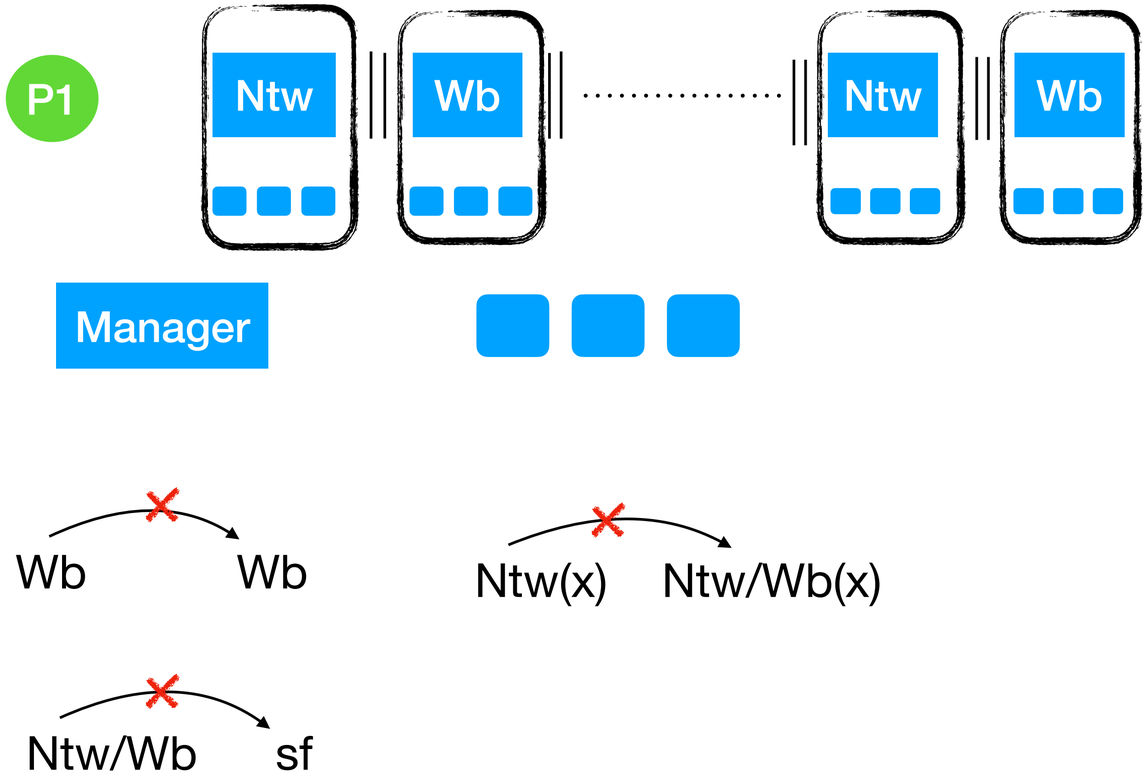
Manager moves the writes to main memory

Decidability



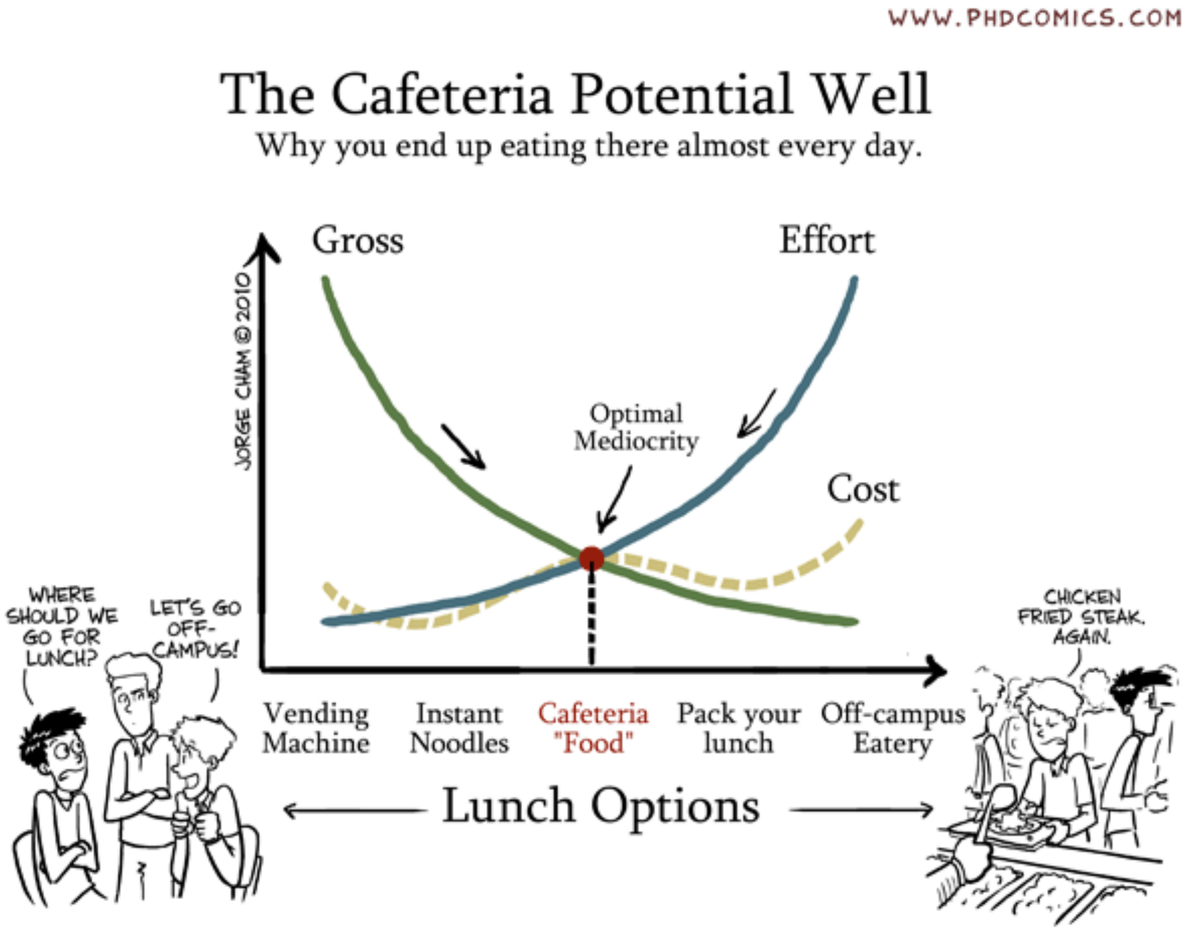
Manager ensures that the semantics is maintained

Decidability



Manager ensures that the semantics is maintained

Thank you


WWW.PHDCOMICS.COM
The Cafeteria Potential Well
Why you end up eating there almost every day.
JORGE CHAM © 2010
Gross
Effort
Optimal Mediocrity
Cost
WHERE SHOULD WE GO FOR LUNCH?
LET'S GO OFF-CAMPUS!
CHICKEN FRIED STEAK. AGAIN.
Vending Machine
Instant Noodles
Cafeteria "Food"
Pack your lunch
Off-campus Eatery
Lunch Options