Verification of Concurrent Programs under Weak Memory

S. Krishna
IIT Bombay

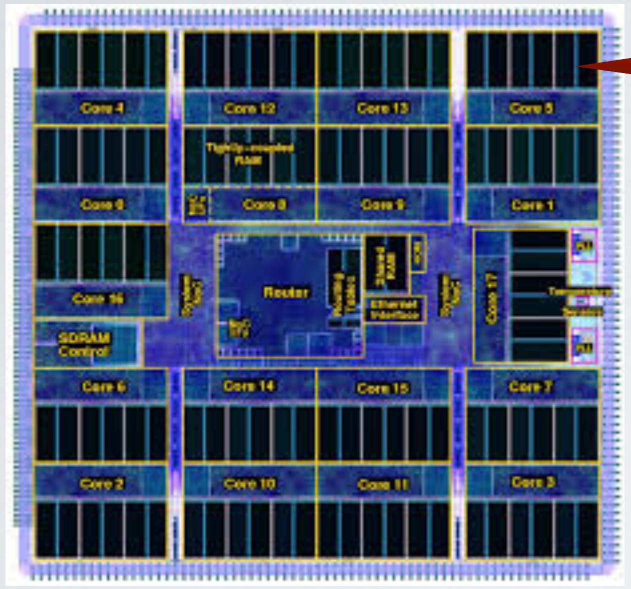
IFIP Working Group 2.2
Formal Description of Programming Concepts
September 24, 2025

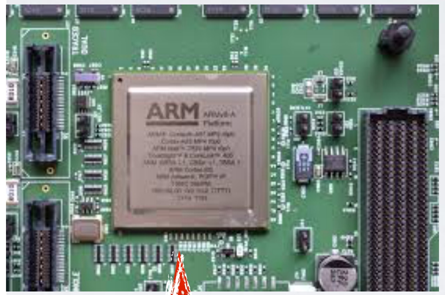
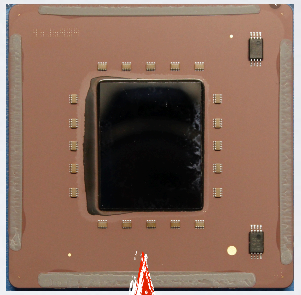
Concurrency

Concurrent systems are everywhere

motivation

Multicore architectures



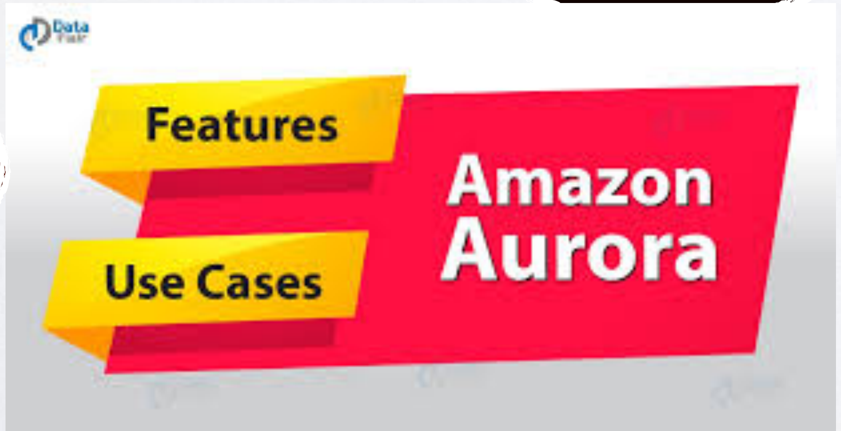
intel

ARM

IBM Power

Distributed databases

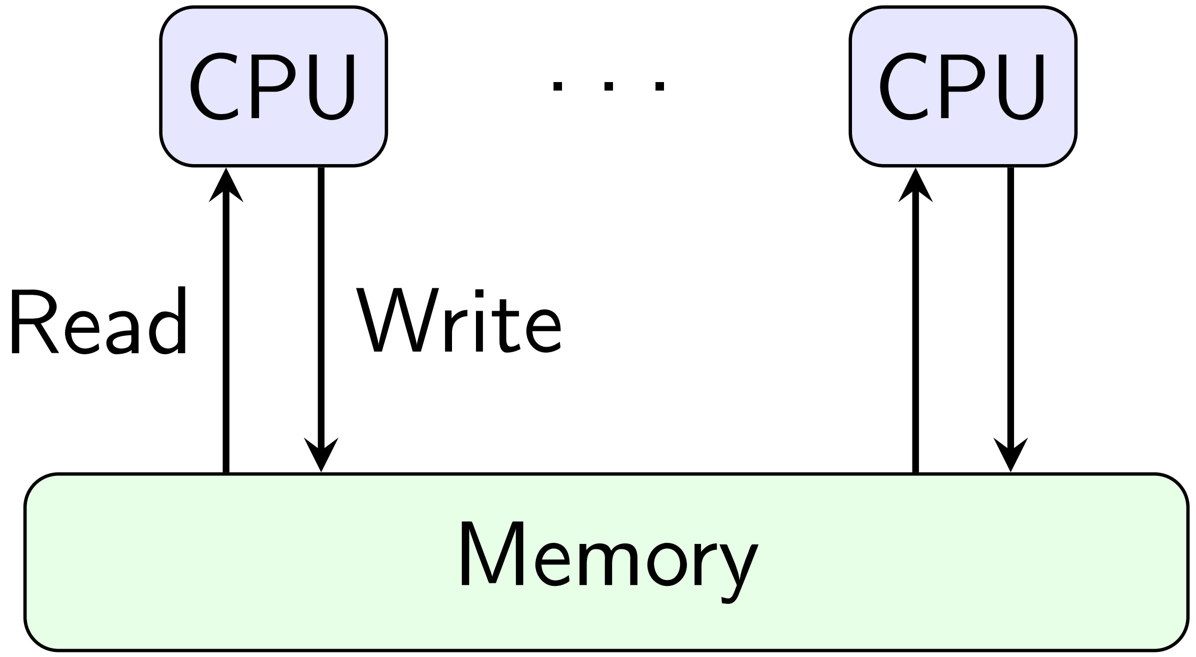
Facebook

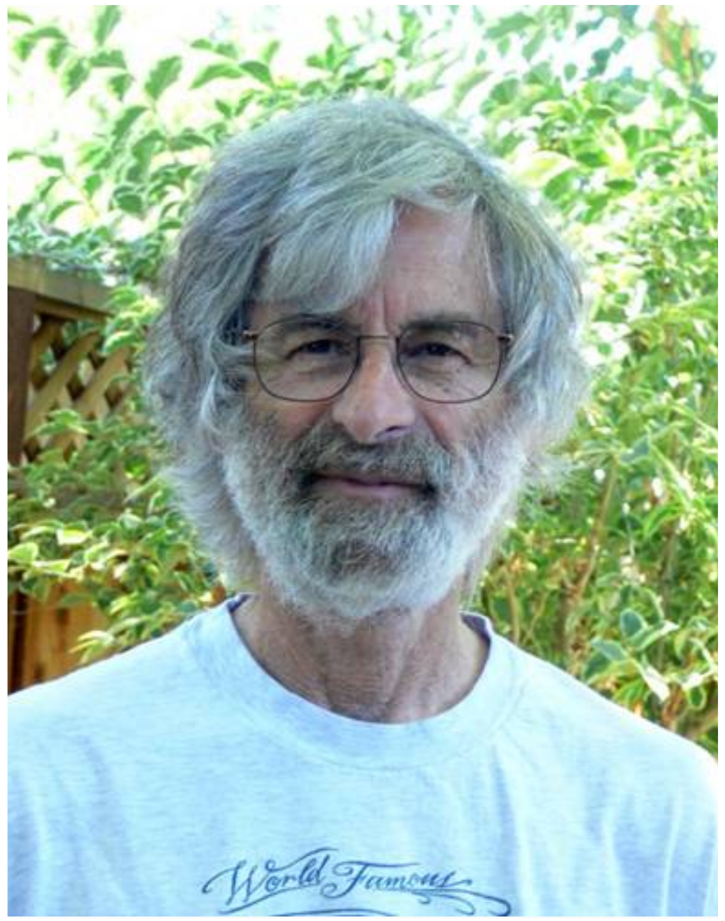


Programming languages

Concurrency with Modern C++
What every professional C++ programmer should know about concurrency
Rainer Grimm

Concurrent Programs (under SC)





Leslie Lamport
How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs, IEEE Trans. Computers, 1979.

- Standard view of concurrency : threads communicate via a uniform shared memory
- Under SC, all threads see the same order of writes

Concurrent Programs (under Sequential Consistency)

Initially x = 0

th_1	$\|$	th_2
$x := 1$		$x := 2$
$a := x$		$b := x$

process

- **Threads**
  - **concurrent**
  - **local registers: a,b**
- **Shared variables: x,y**

Concurrent Programs (under SC)

Initially x = 0

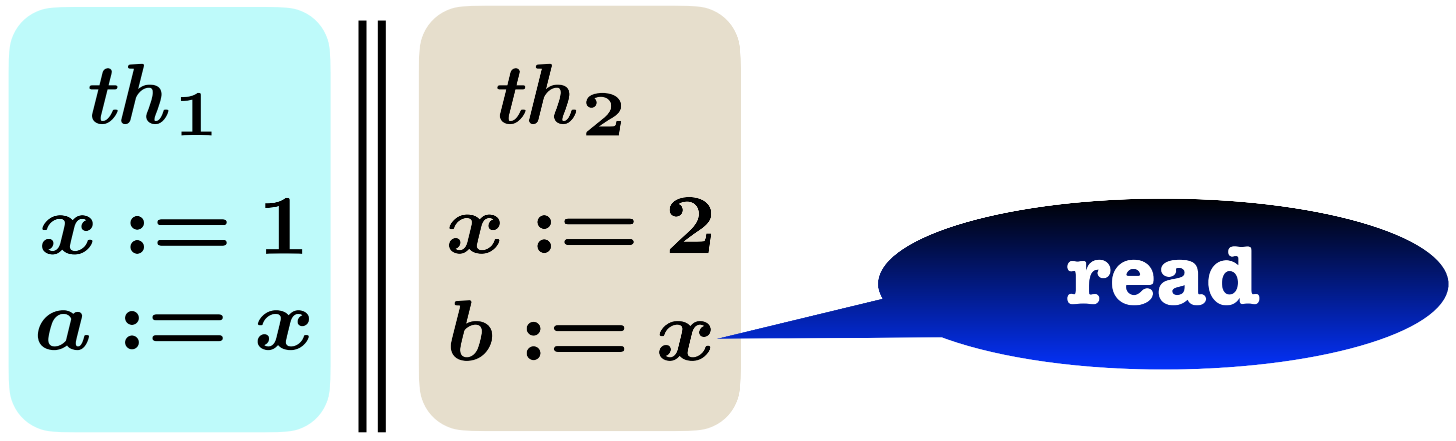
th_1	‖	th_2
$x := 1$		$x := 2$
$a := x$		$b := x$

write

- **Threads**
  - **concurrent**
  - **local registers: a,b**
- **Shared variables: x,y**

Concurrent Programs (under SC)

Initially x = 0



- **Threads**
  - **concurrent**
  - **local registers: a,b**
- **Shared variables: x,y**
- **Allow loops, conditionals**
- **Assignments involving local variables**

Concurrent Programs (under SC)

Initially x = 0

th_1	$\|$	th_2
$x := 1$		$x := 2$
$a := x$		$b := x$

$(a = 1) \wedge (b = 2)$?

assertion

π : execution

e_1 $x := 1$

e_2 $a := 1$

e_3 $x := 2$

e_4 $b := 2$

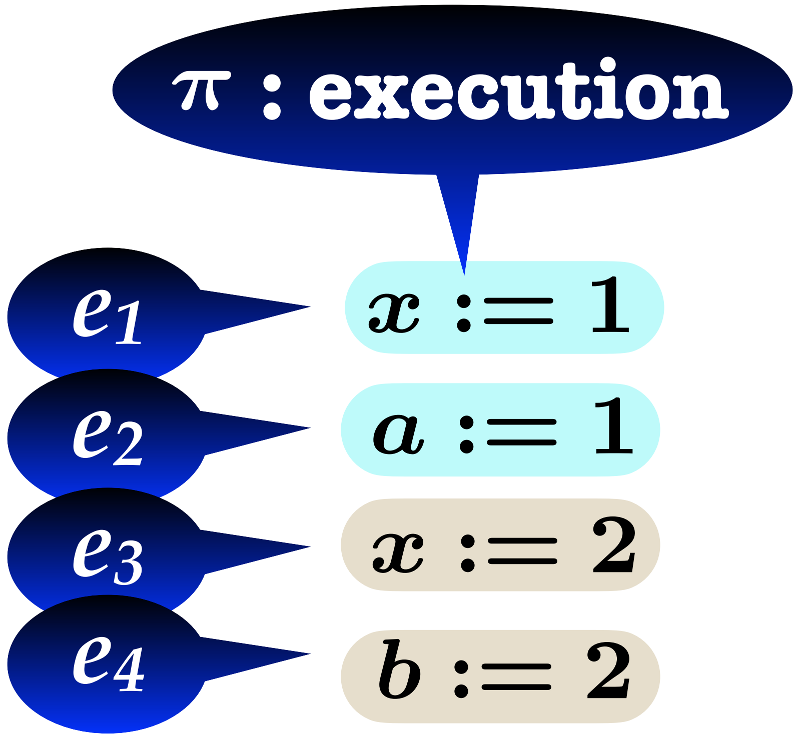
- **execute instructions of one thread in program-order**
- **interleave instructions of different threads**
- **read-from the last write**

- **Threads**
  - **concurrent**
  - **local registers: a,b**
- **Shared variables: x,y**
- **Allow loops, conditionals**
- **Assignments involving local variables**

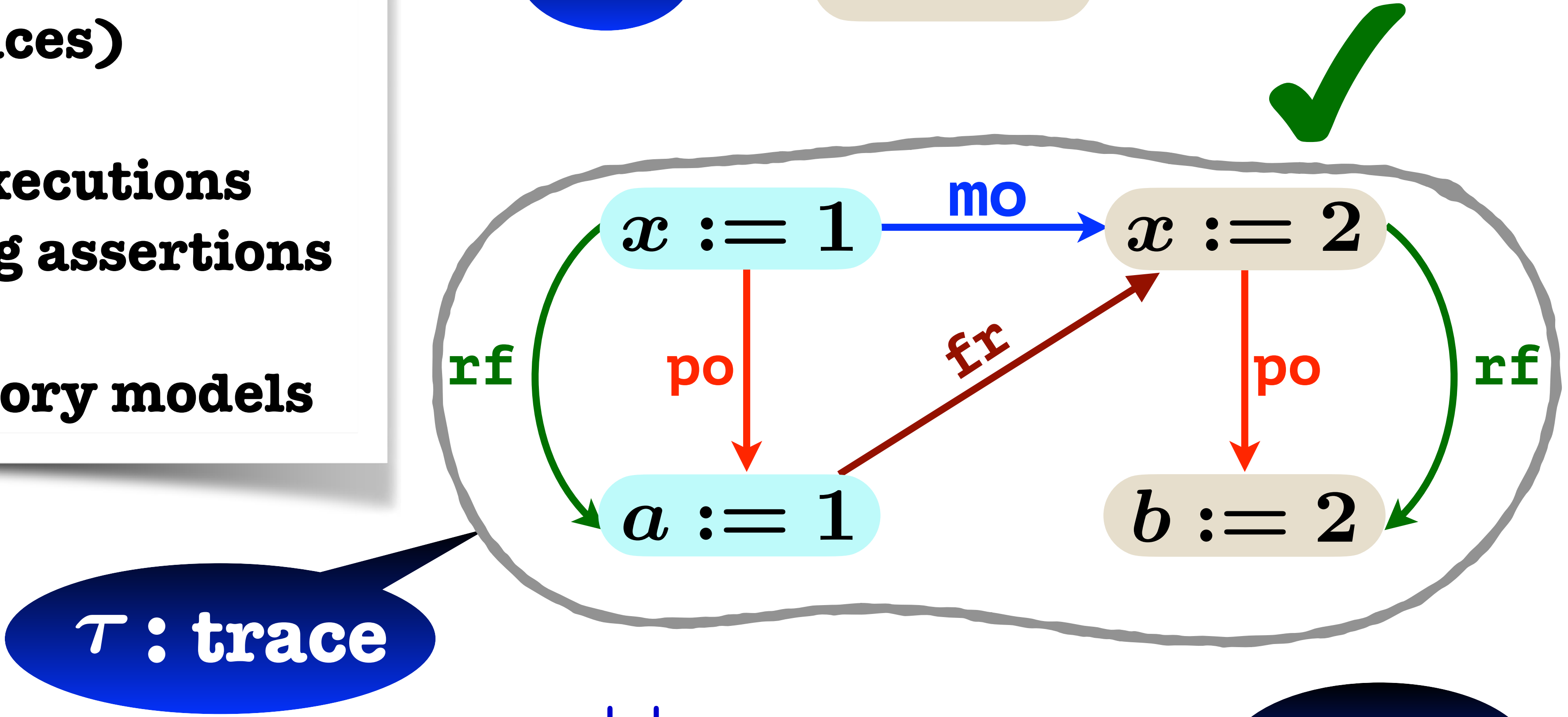
Traces (Sequential Consistency)

Initially x = 0

th_1	th_2
$x := 1$	$x := 2$
$a := x$	$b := x$



- **Execution Graphs (or Traces)**
  - **efficient:**
    - **more compact than executions**
    - **sufficient for checking assertions**
  - **abstract:**
    - **allows different memory models**



$$\begin{tikzcd}
x := 1 \arrow[r, "\mathsf{mo}"] \arrow[d, "\mathsf{po}"] \arrow[d, bend right, "\mathsf{rf}"'] & x := 2 \arrow[d, "\mathsf{po}"] \arrow[d, bend left, "\mathsf{rf}"] \\
a := 1 \arrow[ur, "\mathsf{fr}"] & b := 2
\end{tikzcd}$$

$\mathsf{mo} = \cup_{x \in X} \mathsf{mo}^x$ ← total

SC: $acyclic(\mathsf{po} \cup \mathsf{rf} \cup \mathsf{mo} \cup \mathsf{fr})$

Message Passing (under SC)

Init: x=y=0

```
1. x=1;
2. y=2;
```

Process 1

```
1. while(y=0)
   skip;
2. rx=x;
```

Process 2

Specification S: (rx=0) ✗

Store Buffer (under SC)

Init: x=y=0

```
1. x=1;
2. ry=y;
```

Process 1

```
1. y=1;
2. rx=x;
```

Process 2

Specification S: (rx=0 && ry=0) ✗

2+2W (under SC)

Init: x=y=0

```
1. x=1;
2. y=2;
3. a=y;
```

Process 1

```
1. y=1;
2. x=2;
3. b=x;
```

Process 2

Specification S: (a=1 && b=1) ✗

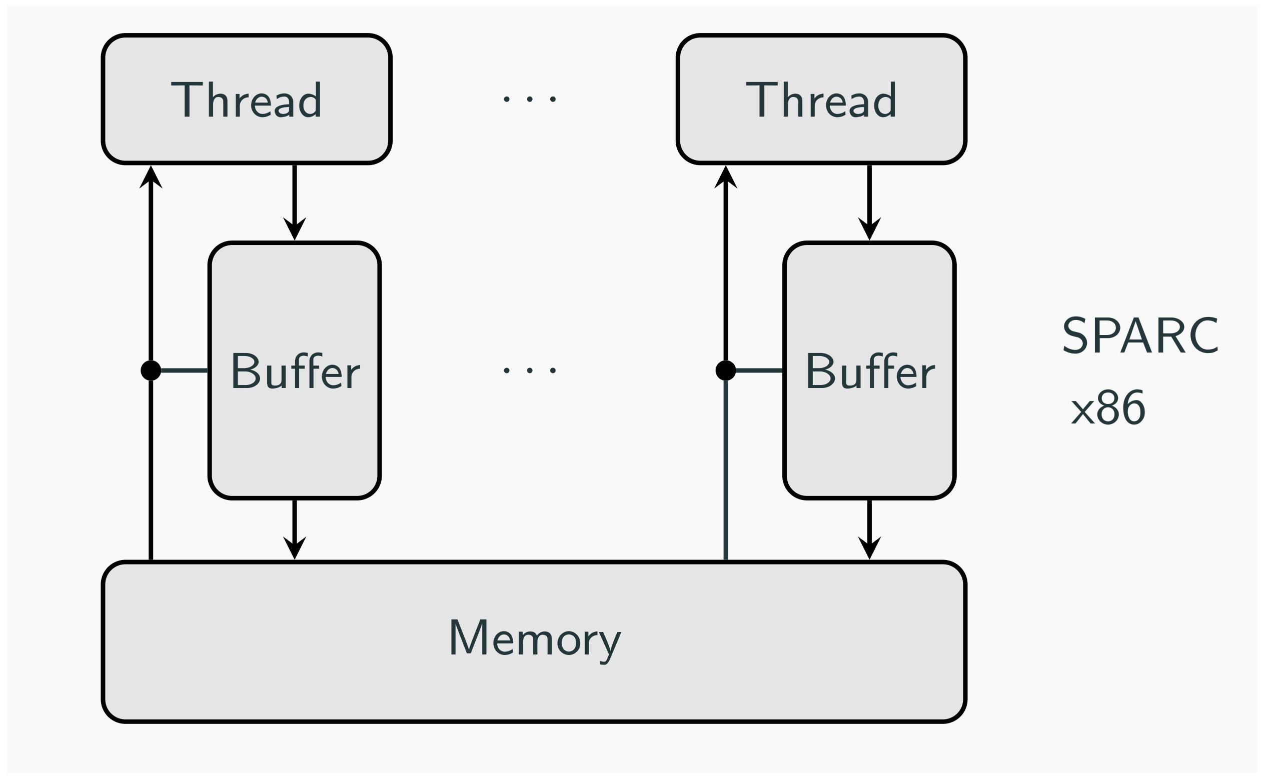
Total Store Order
(x-86 TSO)

[S. Owens, S. Sarkar, and P. Sewell. A better x86 memory model: x86-TSO (2009)]

[SPARC International, Inc. The SPARC Architecture Manual Version 9 (1994)]

[Intel. 2014. Intel 64 and IA-32 architectures software developer's manual]

The x-86 TSO Architecture



- Thread writes go to a thread owned buffer
- Buffers non-deterministically flush to memory
- A thread always reads from its own buffer, if possible
- Every SC behaviour is realizable under TSO
    - Flush each buffer immediately

Store Buffer (under TSO)

Init: x=y=0

Process 1

```
1. x=1;
2. ry=y;
```

Process 2

```
1. y=1;
2. rx=x;
```

Specification S: (rx=0 && ry=0) ✓

2+2W (under TSO)

Init: x=y=0

```
1. x=1;
2. y=2;
3. a=y;
```

Process 1

```
1. y=1;
2. x=2;
3. b=x;
```

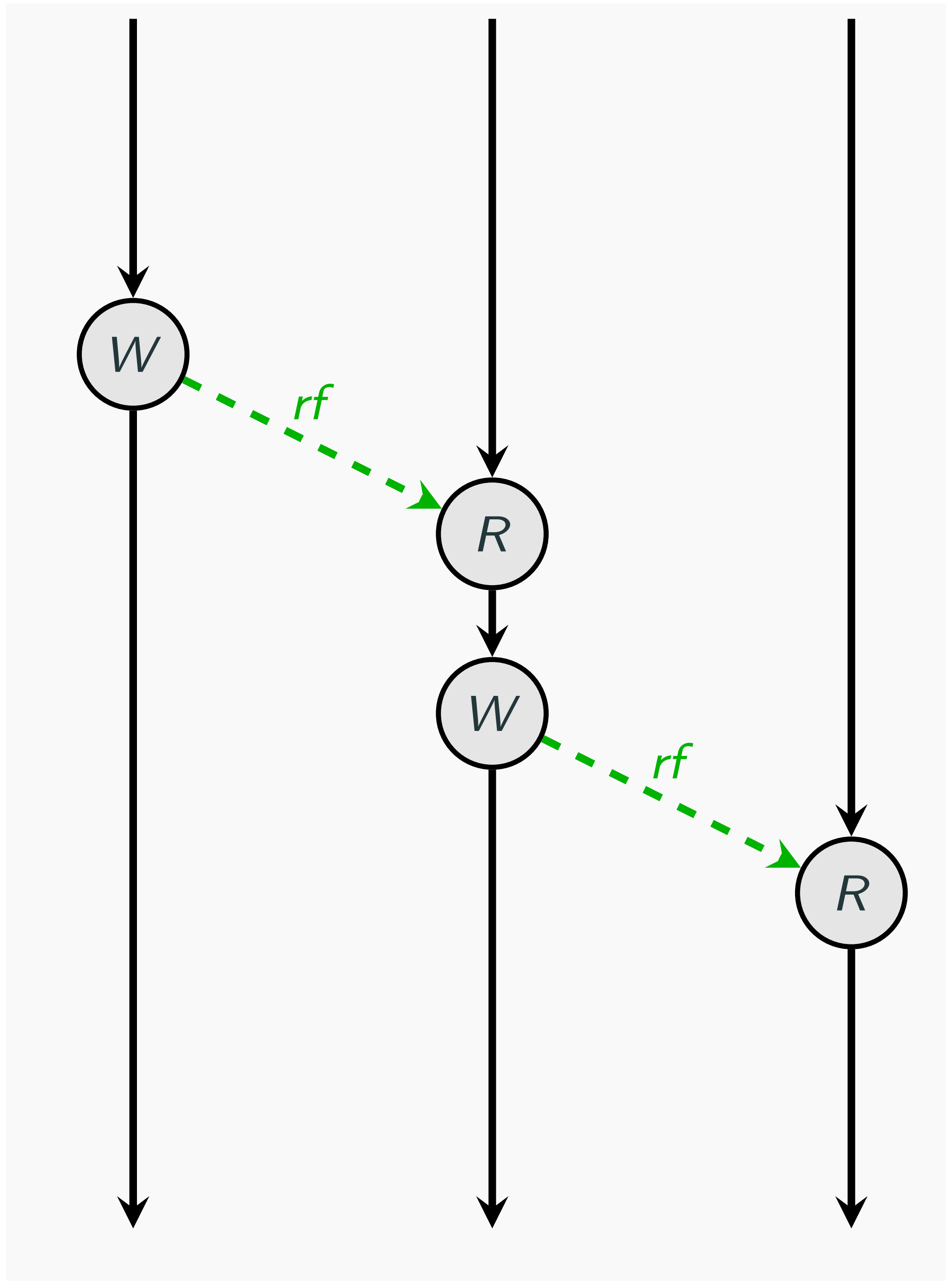
Process 2

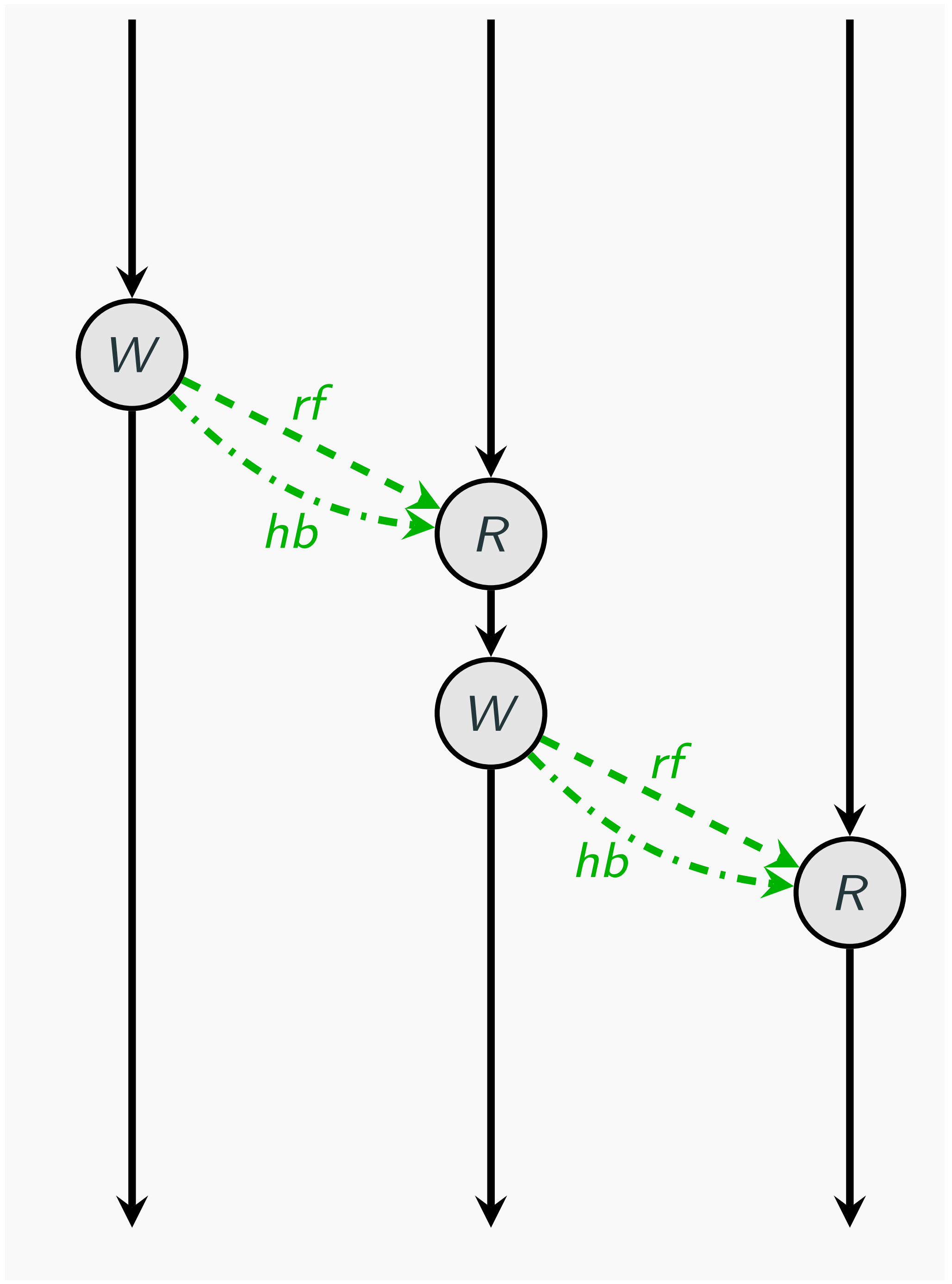
Specification S: (a=1 && b=1) ✗

Release-Acquire (RA)

MarkBatty et al, Mathematizing C++ concurrency, POPL 2011

Causality is a Partial Order

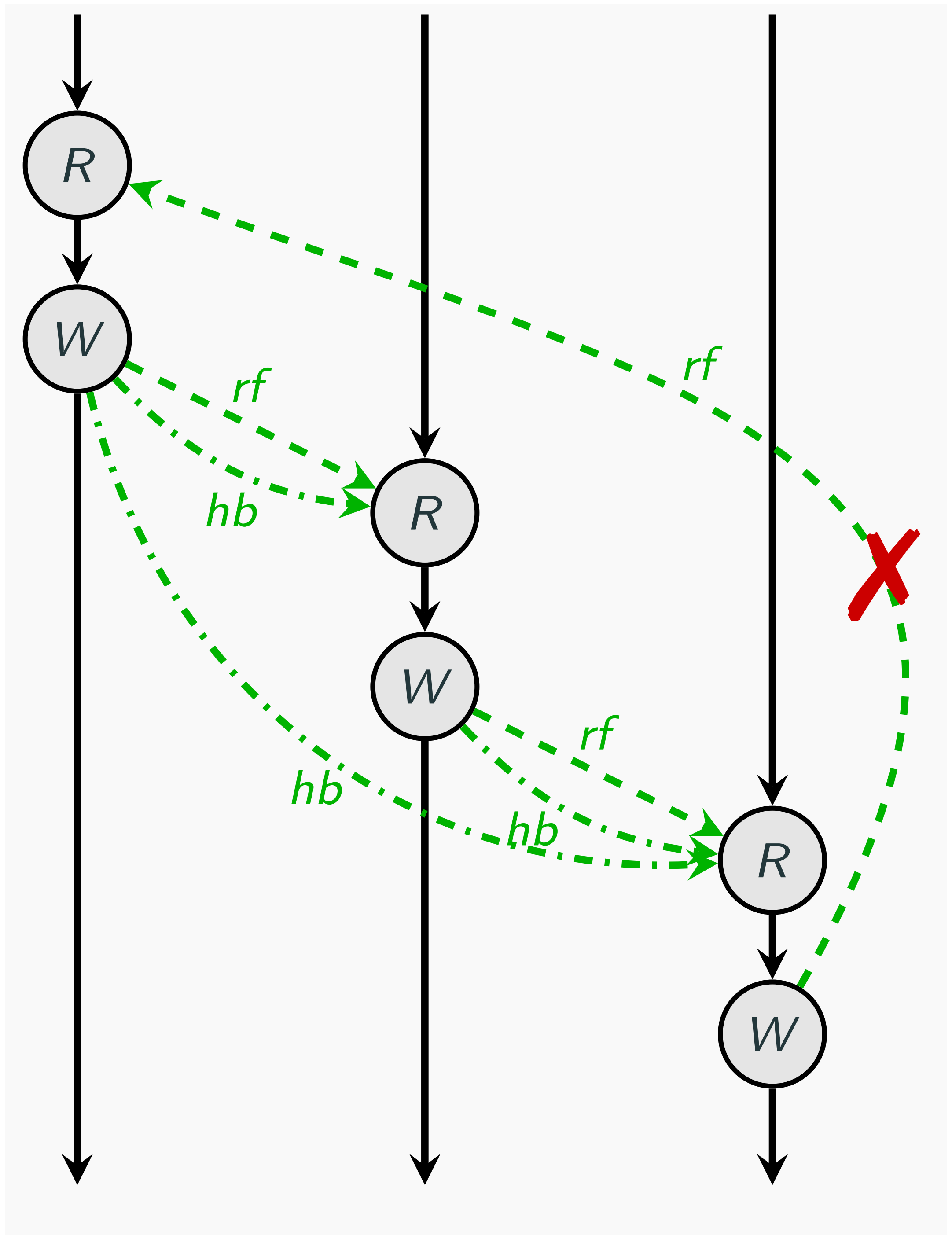




Happens-before (Causal Order)

$$hb \triangleq (po \cup rf)^+$$

Causality is a Partial Order

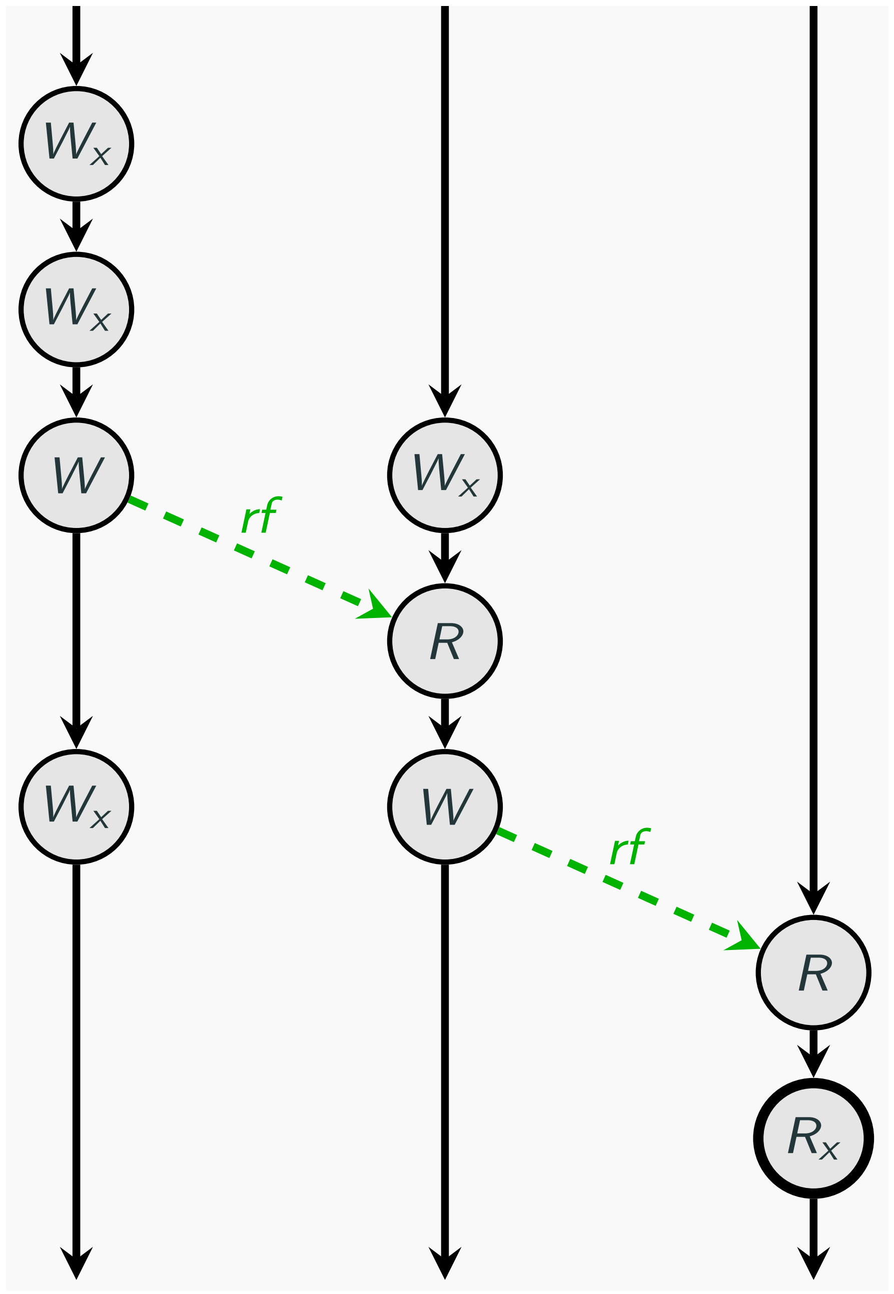


Happens-before (Causal Order)

$$hb \triangleq (po \cup rf)^{+}$$

must be acyclic

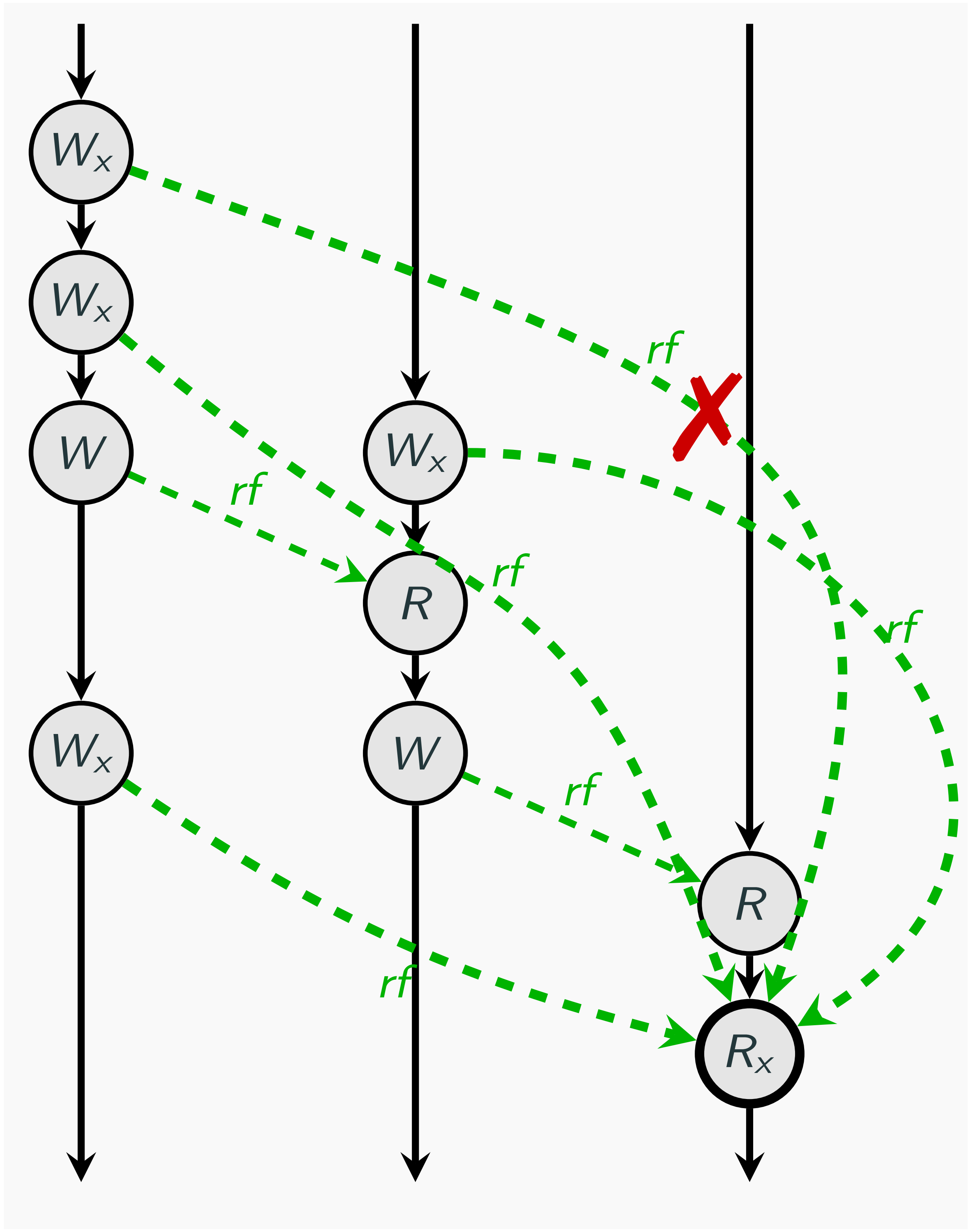
Read Maximal Writes



A read can only observe

- a new write, not known to it, or,
- a "most recent" write, known to it

Read Maximal Writes



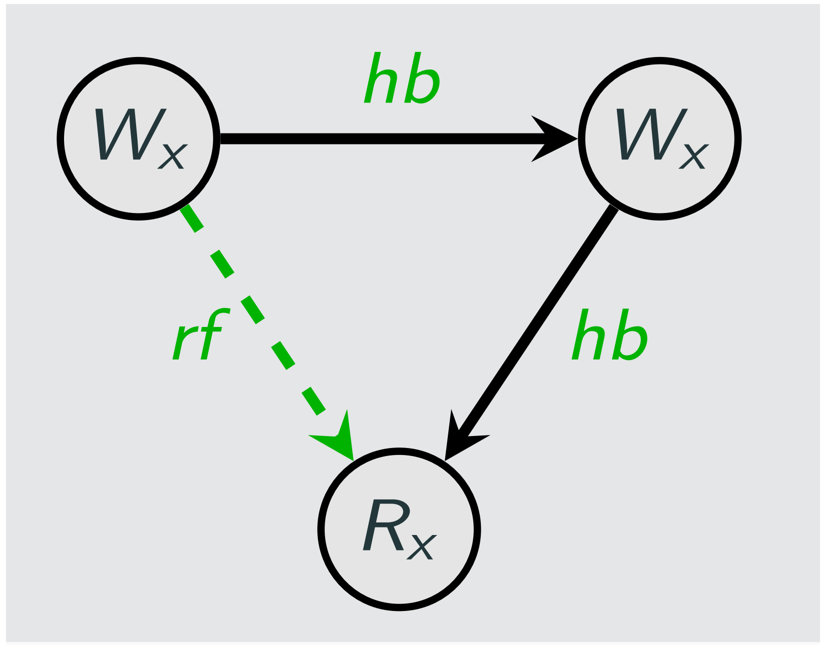
A read can only observe

- a new write, not known to it, or,
- a “most recent” write, known to it

Causal Consistency (CC)

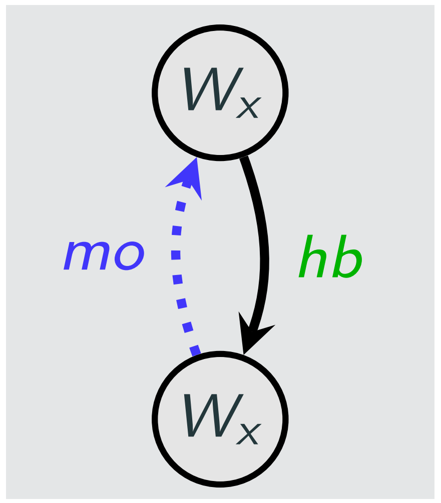
- Acyclic happens-before
- Weak read coherence

Violating Weak read coherence

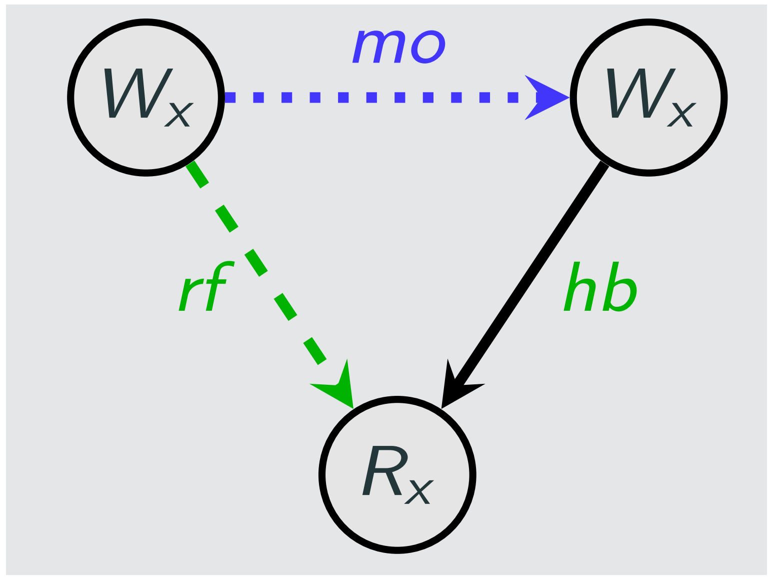


Release Acquire as Message Passing

- RA = CC + coherence
- Writes on the same location x are serialized using a modification order
- Write coherence : mo per variable cannot go against causality



- Read coherence : a read on x can only observe the maximal write on x



Store Buffer (under RA)

Init: x=y=0

```
1. x=1;
2. ry=y;
```

Process 1

```
1. y=1;
2. rx=x;
```

Process 2

Specification S: (rx=0 && ry=0) ✓

2+2W (under RA)

Init: x=y=0

```
1. x=1;
2. y=2;
3. a=y;
```

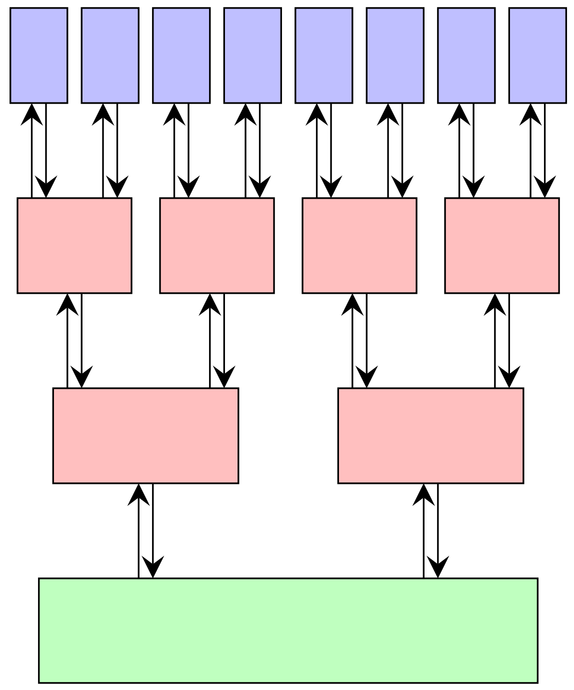
Process 1

```
1. y=1;
2. x=2;
3. b=x;
```

Process 2

Specification S: (a=1 && b=1) ✓

Other Weak Memory Models



- Hardware
  - IBM POWER, ARM
- Programming language models
  - Fragments of C++ (RA, Strong RA(SRA), Weak RA(WRA)), Relaxed, RC 20

Verification Problems

Reachability (bounded data domain)

- **Under SC, decidable**
  - easy to see EXPTIME, but in PSPACE, [Kozen 1977]
- **x-86 TSO, Ackermann complete**

  [Atig et al., POPL 2010]
- **Release Acquire (RA)**
  - undecidable with atomic-read-write
  - Ackermann hard with just reads and writes
  - decidability open with just reads and writes

  [Abdulla et al., PLDI 2019]

Reachability (bounded data domain)

- **Undecidable under POWER**

  [Abdulla et al., NETYS 2020]

- **Weak and Strong RA, Ackermann-complete**

  [Boker and Lahav, PLDI 2020, TOPLAS 2022]

- **Undecidable under ARM**

  [can be shown similar to POWER]

Bounded Model Checking

- **Bounded context switching**
  - well studied under approximation for SC [Qadeer et al 2005]
- **x-86 TSO, POWER**
  - code-to-code translation to SC under bounded context switches [Atig et al, FPS 2014, Abdulla et al, TACAS 2017]
- **Bounded context switching does not help for RA**
  - View bounding under RA
  - code-to-code translation to SC under bounded context switches [Abdulla et al., PLDI 2019]

Bounded Model Checking

- **Bounded context switching**
    - state of the art tool Dartagnan
    - across weak memory models and GPU
    - adopts an SMT based approach

      [Ponce de Leon et al 2022, 2024]

won Gold and Silver medals at SVCOMP

Bounded Model Checking

- **Partial Order Reduction**
    - two interleavings are equivalent if one can be obtained from the other by swapping adjacent, independent actions.
    - Each such class is a Mazurkiewicz trace
    - In each equivalence class, POR explores at least one interleaving

Bounded Model Checking

- **Partial Order Reduction**
  - Dynamic Partial Order Reduction (DPOR) [Flanagan and Godefroid 2005]
  - Source DPOR : significant reduction in the number of explored interleavings than DPOR [Abdulla et al, POPL 2014, J.ACM 2017]
- **DPOR for Weak Memory : Trace Optimality**

Verification of Concurrent Programs

Initially $x = 0$

th_1	$\|$	th_2
$x := 1$		$x := 2$
$a := x$		$b := x$

$(a = 1) \wedge (b = 2)$?

Violation

Is there an execution witnessing the violation?

$$\mathit{Prog} \longrightarrow \boxed{\textbf{DPOR}} \longrightarrow$$

at least one execution per trace

$x := 1$	$x := 1$
$x := 2$	$x := 2$
$a := 2$	$b := 2$
$b := 2$	$a := 2$

$$\begin{tikzcd}
x := 1 \arrow[r, "\mathsf{mo}"] \arrow[d, "\mathsf{po}"'] & x := 2 \arrow[dl, "\mathsf{rf}"'] \arrow[d, "\mathsf{po}"] \arrow[d, bend left, "\mathsf{rf}"] \\
a := 2 & b := 2
\end{tikzcd}$$

$x := 2$	$x := 2$
$x := 1$	$x := 1$
$b := 1$	$a := 1$
$a := 1$	$b := 1$

$$\begin{tikzcd}
x := 1 \arrow[d, bend right, "\mathsf{rf}"'] \arrow[d, "\mathsf{po}"] \arrow[dr, "\mathsf{rf}"] & x := 2 \arrow[l, "\mathsf{mo}"'] \arrow[d, "\mathsf{po}"] \\
a := 1 & b := 1
\end{tikzcd}$$

Abdulla et al POPL'14

$x := 1$
$a := 1$
$x := 2$
$b := 2$

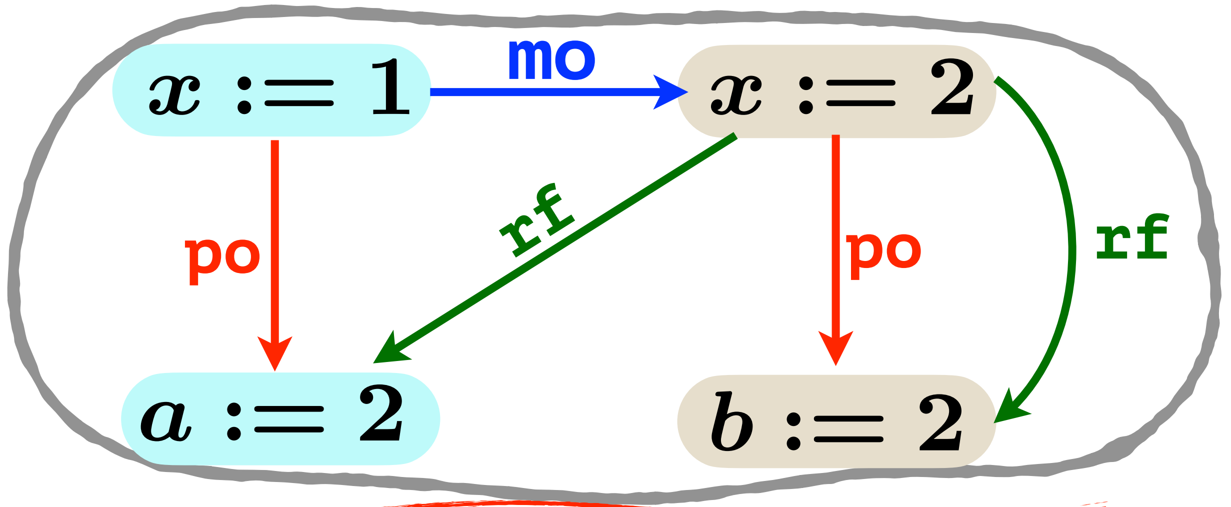
$$\begin{tikzcd}
x := 1 \arrow[d, bend right, "\mathsf{rf}"'] \arrow[d, "\mathsf{po}"] & x := 2 \arrow[l, "\mathsf{mo}"'] \arrow[d, "\mathsf{po}"] \arrow[d, bend left, "\mathsf{rf}"] \\
a := 1 & b := 2 \arrow[ul, "\mathsf{fr}"']
\end{tikzcd}$$

(optimal) DPOR

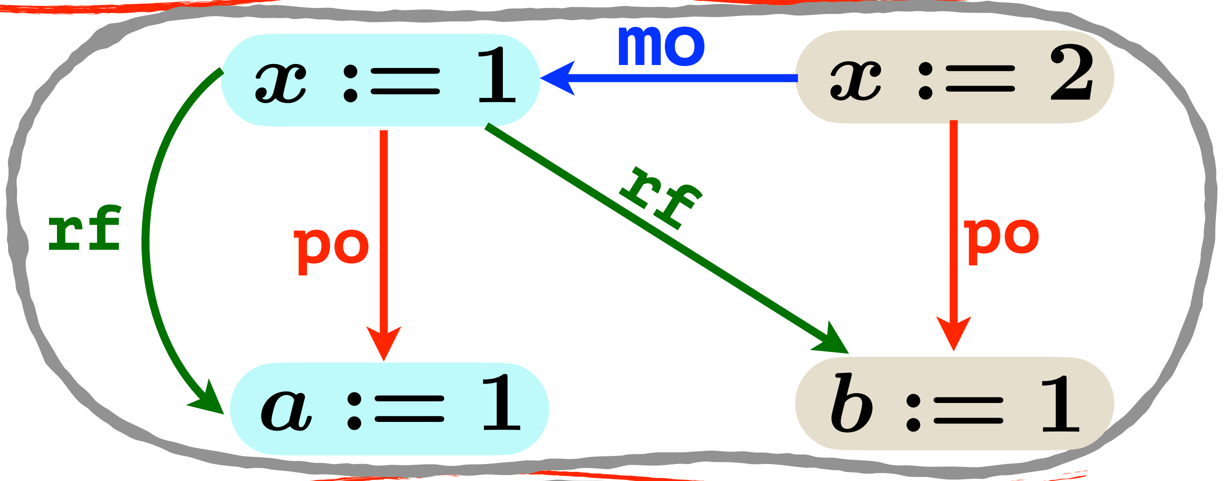
$x := 2$
$b := 2$
$x := 1$
$a := 1$

$$\begin{tikzcd}
x := 1 \arrow[r, "\mathsf{mo}"] \arrow[d, bend right, "\mathsf{rf}"'] \arrow[d, "\mathsf{po}"] & x := 2 \arrow[d, "\mathsf{po}"] \arrow[d, bend left, "\mathsf{rf}"] \\
a := 1 \arrow[ur, "\mathsf{fr}"] & b := 2
\end{tikzcd}$$

$x := 1$
$x := 2$
$a := 2$
$b := 2$

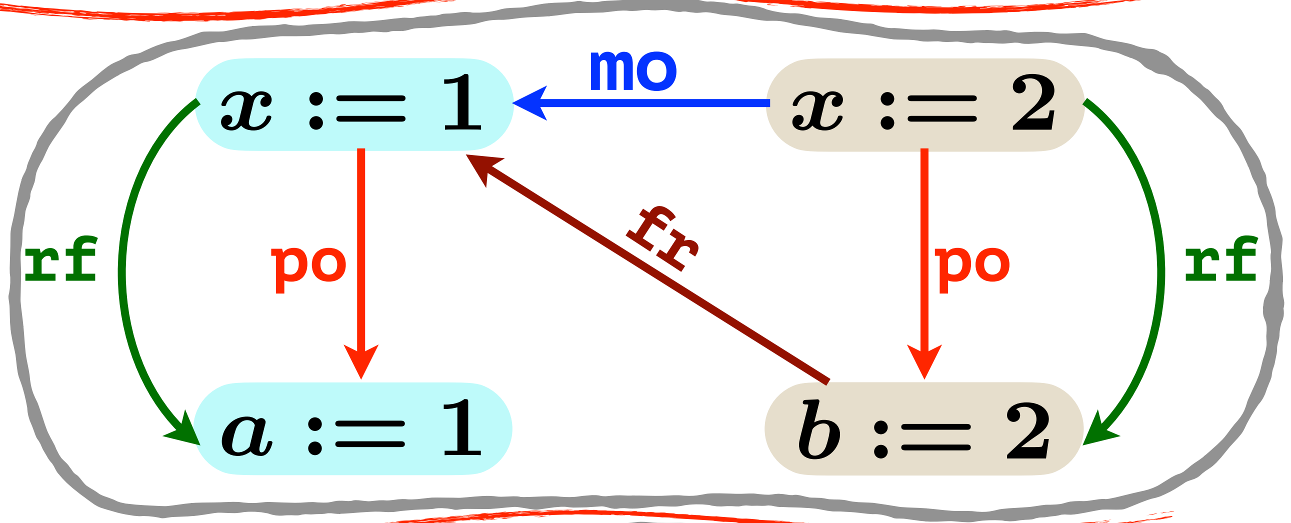


$x := 2$
$x := 1$
$b := 1$
$a := 1$



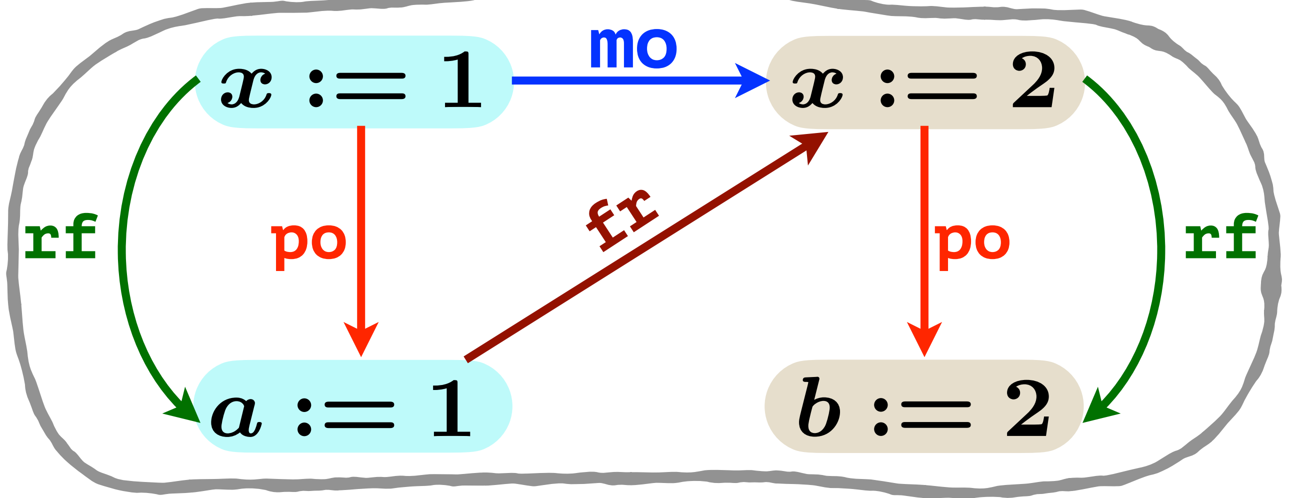
don't
need mo for RA

$x := 1$
$a := 1$
$x := 2$
$b := 2$



(super-
optimal)
DPOR

$x := 2$
$b := 2$
$x := 1$
$a := 1$



$x := 1$
$x := 2$
$a := 2$
$b := 2$

$$\begin{tikzcd}
x := 1 \arrow[r, "\text{mo}"] \arrow[d, "\text{po}"'] & x := 2 \arrow[dl, "\text{rf}"'] \arrow[d, "\text{po}"] \arrow[d, bend left, "\text{rf}"] \\
a := 2 & b := 2
\end{tikzcd}$$

$x := 2$
$x := 1$
$b := 1$
$a := 1$

$$\begin{tikzcd}
x := 1 \arrow[d, bend right, "\text{rf}"'] \arrow[d, "\text{po}"'] \arrow[dr, "\text{rf}"] & x := 2 \arrow[l, "\text{mo}"'] \arrow[d, "\text{po}"] \\
a := 1 & b := 1
\end{tikzcd}$$

$x := 1$
$a := 1$
$x := 2$
$b := 2$

$$\begin{tikzcd}
x := 1 \arrow[d, bend right, "\text{rf}"'] \arrow[d, "\text{po}"'] & x := 2 \arrow[l, "\text{mo}"'] \arrow[d, "\text{po}"] \arrow[d, bend left, "\text{rf}"] \\
a := 1 & b := 2 \arrow[ul, "\text{fr}"']
\end{tikzcd}$$

(super-optimal) DPOR

$x := 1$

$x := 2$

$a := 2$

$b := 2$

$$\begin{tikzcd}
x := 1 \arrow[d, "\mathtt{po}"'] & x := 2 \arrow[dl, "\mathtt{rf}"'] \arrow[d, "\mathtt{po}"] \arrow[d, bend left, "\mathtt{rf}"] \\
a := 2 & b := 2
\end{tikzcd}$$

$x := 2$

$x := 1$

$b := 1$

$a := 1$

$$\begin{tikzcd}
x := 1 \arrow[d, bend right, "\mathtt{rf}"'] \arrow[d, "\mathtt{po}"'] \arrow[dr, "\mathtt{rf}"] & x := 2 \arrow[d, "\mathtt{po}"] \\
a := 1 & b := 1
\end{tikzcd}$$

$x := 1$

$a := 1$

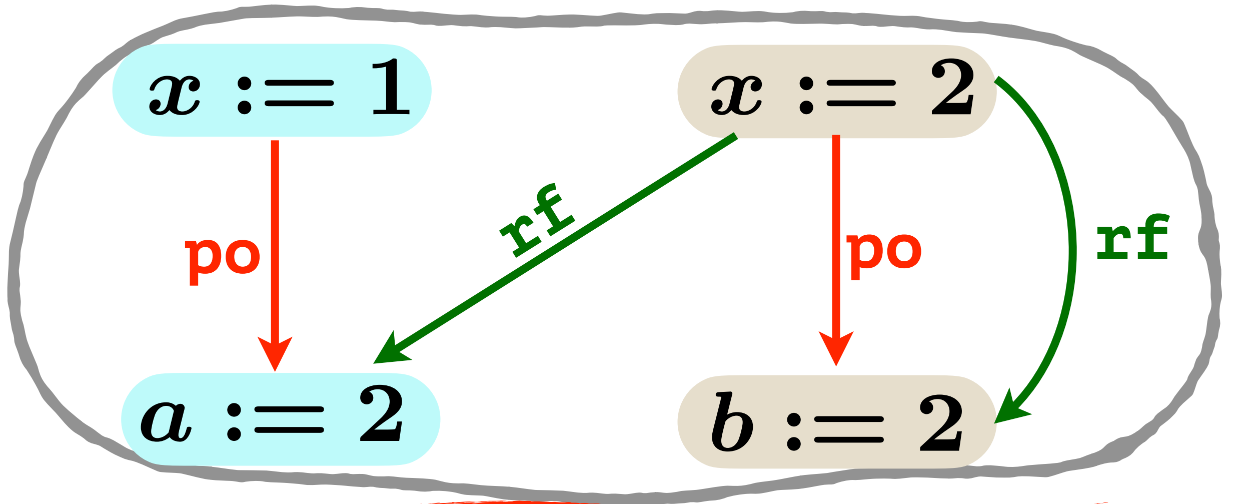
$x := 2$

$b := 2$

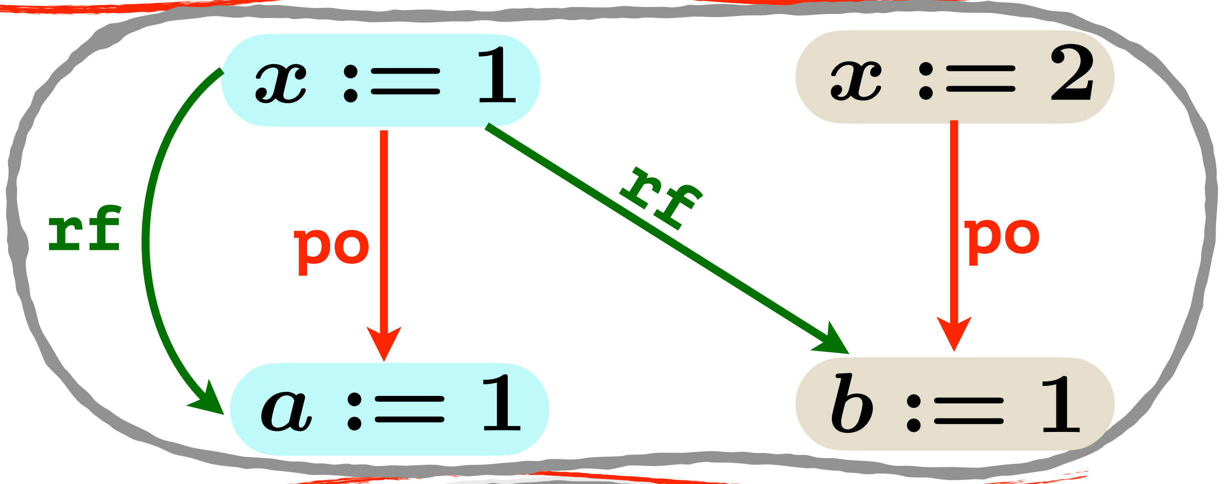
$$\begin{tikzcd}
x := 1 \arrow[d, bend right, "\mathtt{rf}"'] \arrow[d, "\mathtt{po}"'] & x := 2 \arrow[d, "\mathtt{po}"] \arrow[d, bend left, "\mathtt{rf}"] \\
a := 2 & b := 2
\end{tikzcd}$$

(super-optimal) DPOR

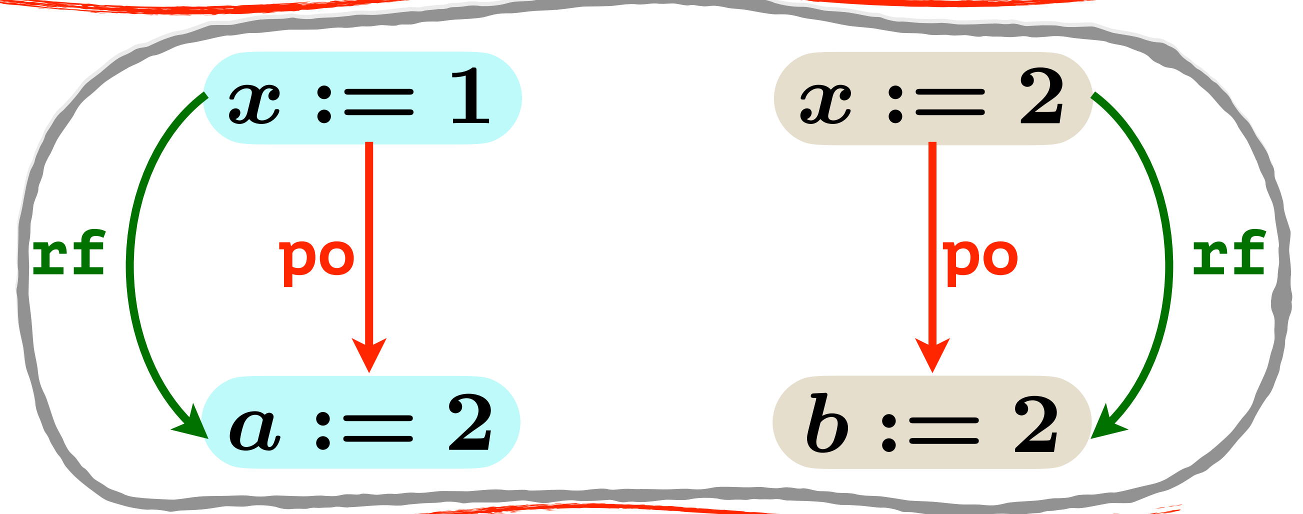
$x := 1$
$x := 2$
$a := 2$
$b := 2$

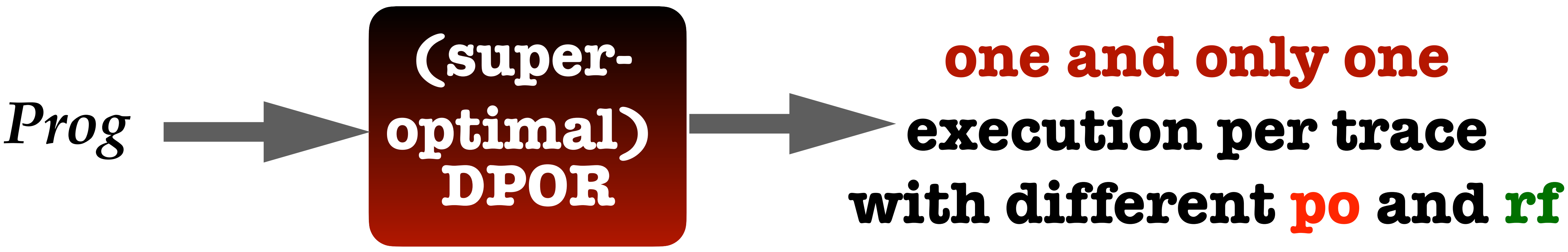


$x := 2$
$x := 1$
$b := 1$
$a := 1$



$x := 1$
$a := 1$
$x := 2$
$b := 2$





one and only one execution per trace with different po and rf

RA Versus SC

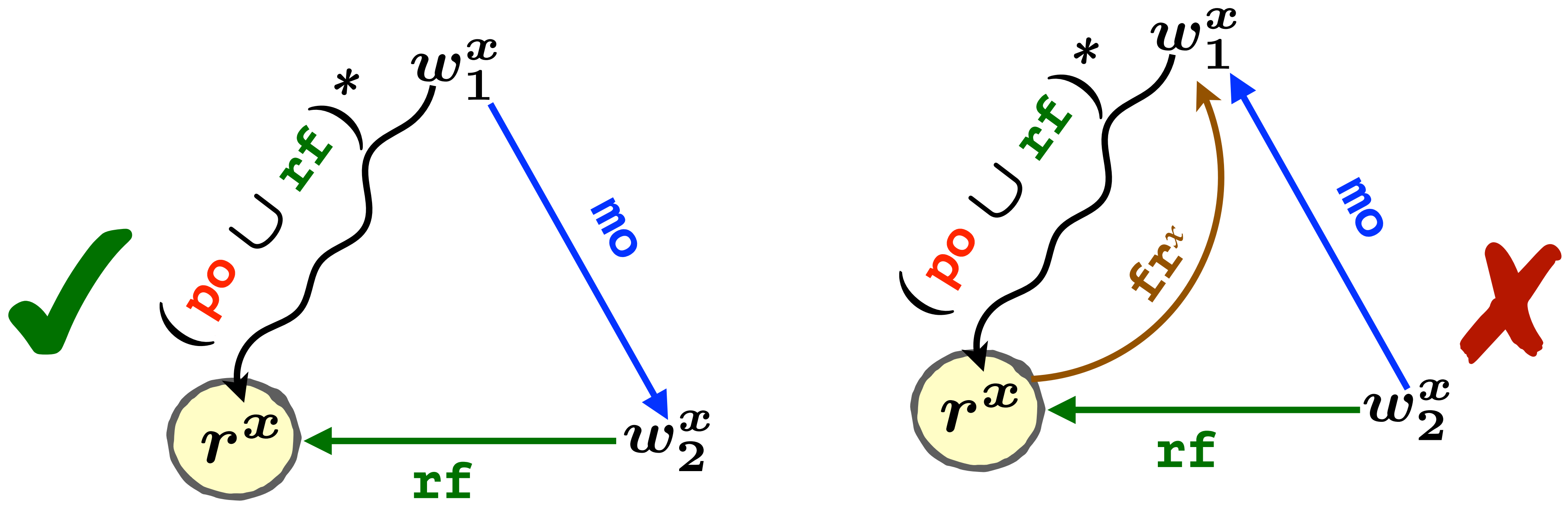
- **RA consistency is prefix determined**
  - for any po-maximal read r, consistency follows from the consistency of the trace without r and the hb-prefix of r, **not true for SC**

$$\begin{tikzcd}
w(x,0) \arrow[d, bend right=60, "mo"'] \arrow[rdd, "rf"] & w(y,0) \arrow[d, bend left=60, "mo"] \arrow[ldd] \\
w(x,1) \arrow[d] & w(y,1) \arrow[d] \\
r(y,0) & r(x,0)
\end{tikzcd}$$

  - Used for efficient exploration of consistent executions [Kokologiannakis et al, POPL 2017]
  - Model checker RCMC
  - Subsequent state-of-the-art model checker GenMC and Trust [Kokologiannakis et al, POPL 2022]

RA Versus SC

- **Saturation in RA**
  - Efficient consistency checking of reduced traces with just po and rf without mo


✓
w_1^x
$(\mathsf{po} \cup \mathsf{rf})^*$
mo
r^x
rf
w_2^x
✗
w_1^x
$(\mathsf{po} \cup \mathsf{rf})^*$
fr_x
mo
r^x
rf
w_2^x

  - Necessary mo edges are added on saturation. The partial mo can be extended to a total order preserving acyclicity
  - Allows efficient exploration of RA consistent reduced traces, tool Tracer

RA Versus SC

- **Saturation in RA**
  - Efficient consistency checking of reduced traces with just po and rf without mo
  - Consistency checking of reduced traces is NP-c for SC [Gibbons and Korach 1997]
  - This is poly time for RA [Abdulla et al OOPSLA 2018]

Complexity of Consistency Checking

In high level

Given an observed execution (i.e., an execution graph) G and a memory model MM, does G satisfy the axioms of MM?

- When G is fully specified $G = (E, po, rf, mo)$, this question is typically easy
  - Polynomial time for SC, TSO, CC, RA, C11 ...
- In practice, G is often specified only partially
- E.g., rf, mo are not observed at the program level
- **How hard is the problem then?**

Complexity of Consistency Checking

Execution Graph Consistency (given rf)

Given an execution graph $G = (E, po, rf)$ and a memory model MM, check if G is consistent wrt the axioms of MM.

Execution Graph Consistency (without rf)

Given an execution graph $G = (E, po)$ and a memory model MM, check if G is consistent wrt the axioms of MM.

Stateful Approaches

- Optimality has been key in stateless approaches
  - While the stateless approach keeps a tree of runs, all states are kept in stateful approaches
  - Stop exploration when a state is revisited

- **Recent work for SC** [Herbreteau et al, CONCUR 2025]
  - Guiding optimality principle like trace optimality?
  - Obtaining a minimal (within a polynomial factor) reduced transition system is NP-hard even when the underlying program is acyclic
  - Heuristics to improve efficiency of stateful methods

Some Other Directions

- Robustness between memory models

  [Margalit et al, POPL 2025]

- Hoare-style Program logics for Weak Memory

  [Lahav et al, ICALP 2015]

- Programs with Message passing

  [Enea et al, OOPSLA 2024]

- Event driven programs

  [Abdulla et al, ATVA 2023]

Thankyou