Parametric Verification of Concurrent Programs under the TSO Weak Memory Model

Ahmed Bouajjani
Paris Diderot University

Based on joint work with

Parosh A. Abdulla Mohamed Faouzi Atig T. Phong Ngo
Uppsala University

Sebastian Burckhardt Madan Musuvathi
Microsoft Research

SynCoP+PV'17, Uppsala, April 22, 2017

Sequential Consistency

- Concurrent processes with Shared Memory
- Operations: Writes and Reads
- Computation of different processes are shuffled
- Program order is preserved for each process

Sequential Consistency

- Concurrent processes with Shared Memory
- Operations: Writes and Reads
- Computation of different processes are shuffled
- Program order is preserved for each process
- => Strong consistency:

  Operations are immediately visible to all processes

Sequential Consistency

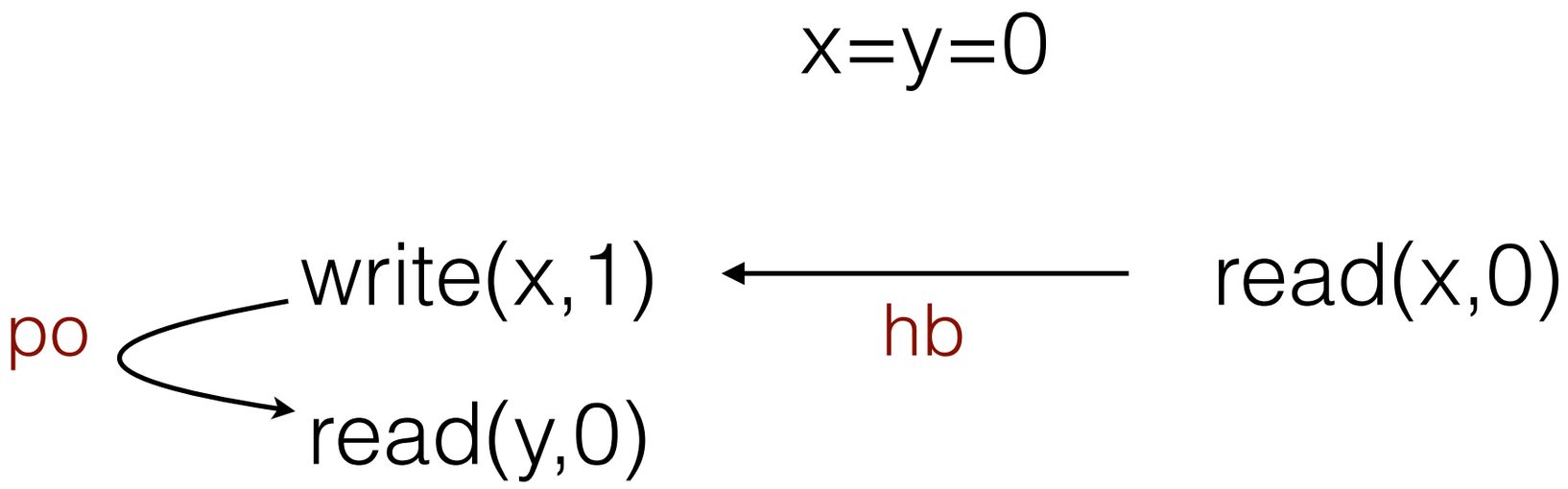
- Concurrent processes with Shared Memory
- Operations: Writes and Reads
- Computation of different processes are shuffled
- Program order is preserved for each process

- => Strong consistency:

  Operations are immediately visible to all processes

- Simple and Intuitive model
- Disallows many hardware/compiler optimisations

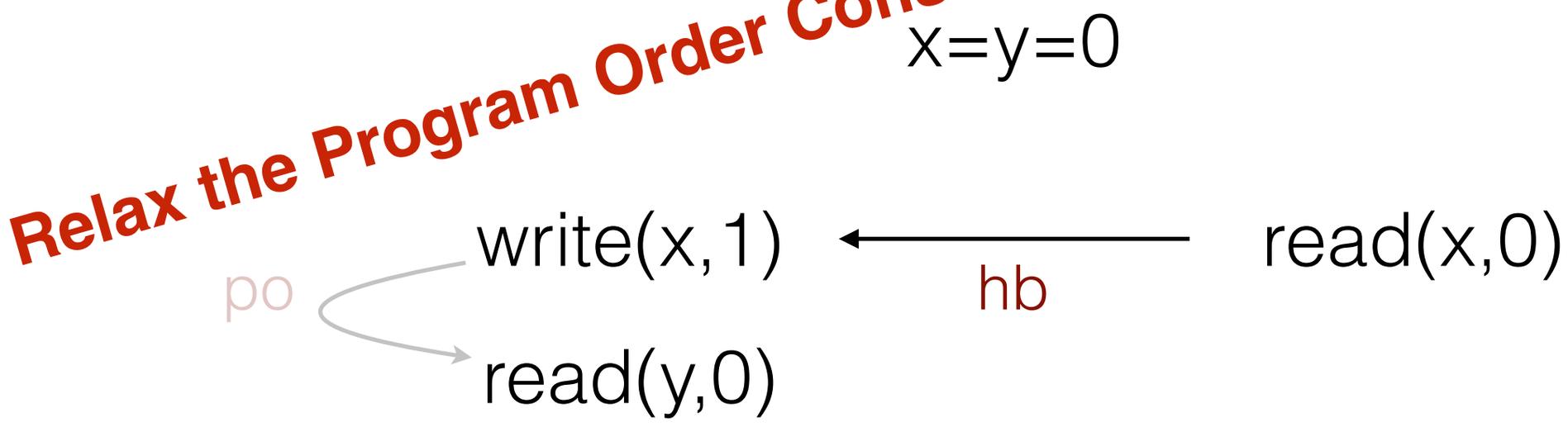
Weak Memory Models



SC read(x,0) write (x,1) read(y,0)

Weak Memory Models

Relax the Program Order Constraints



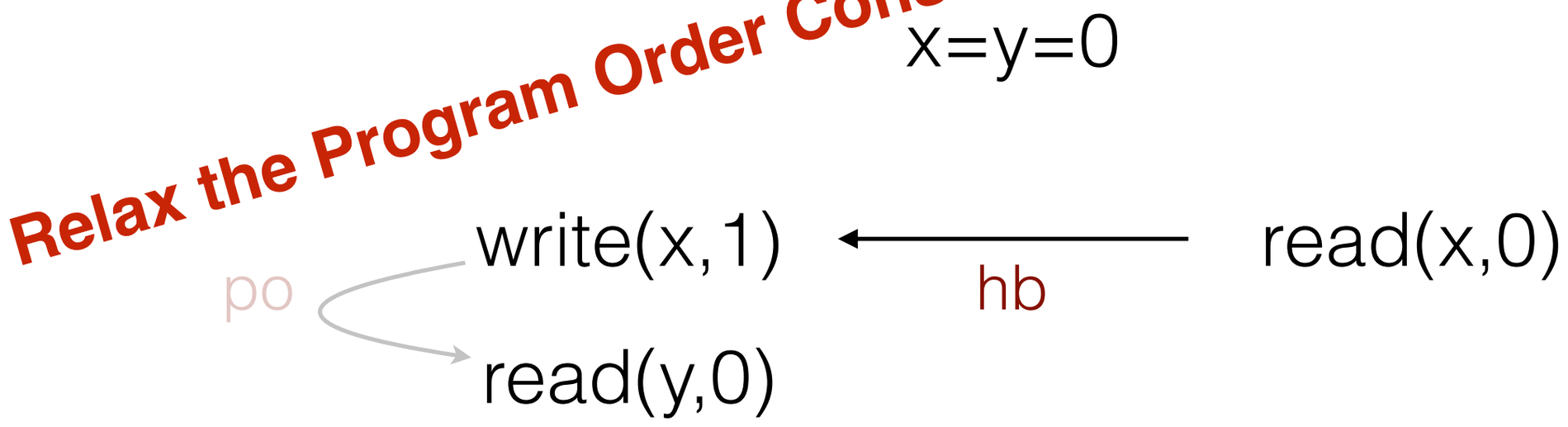
SC read(x,0) write (x,1) read(y,0)

Swap operations

TSO read(x,0) read(y,0) write (x,1)

Weak Memory Models

Relax the Program Order Constraints



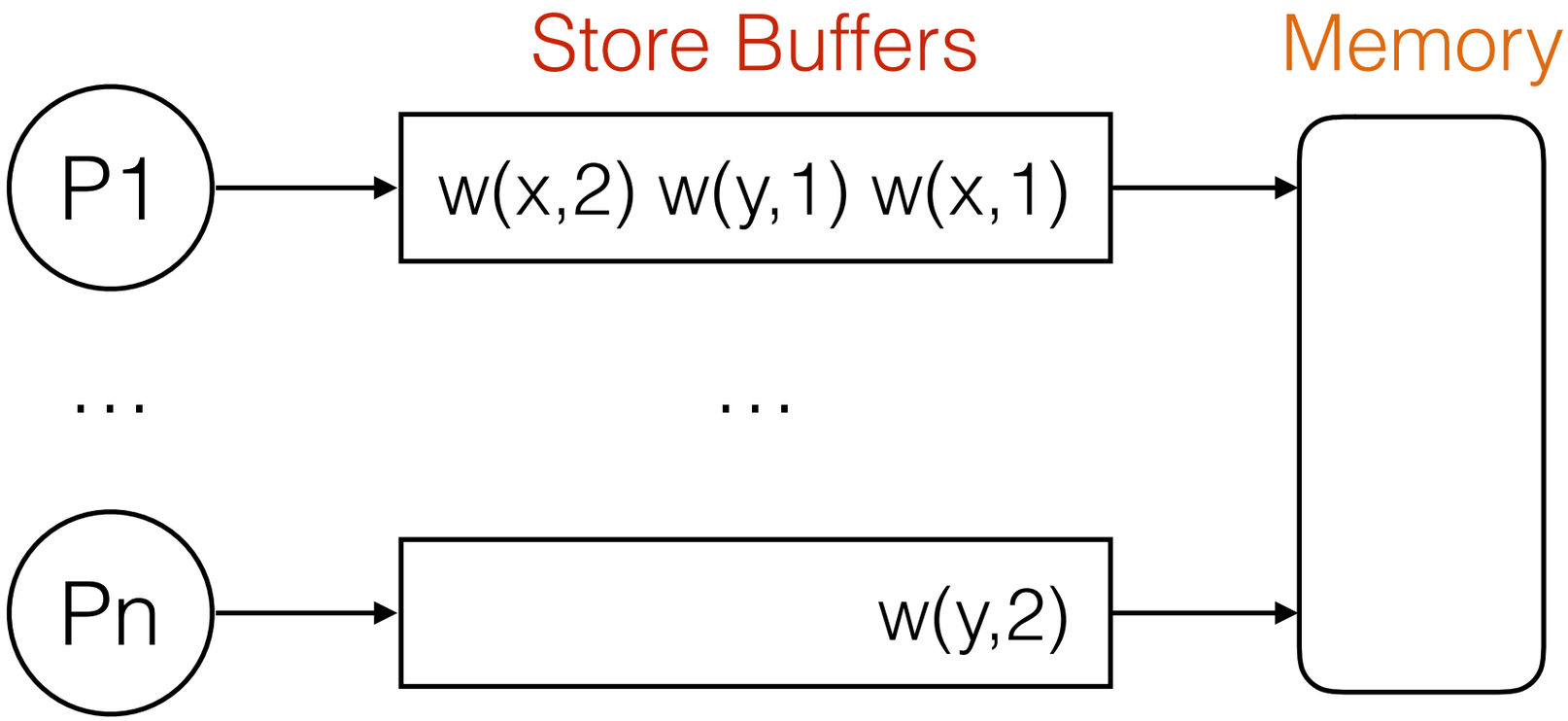
SC read(x,0) write (x,1) read(y,0)

TSO read(x,0) read(y,0) write (x,1)

Swap operations

Execute in parallel

Total Store Ordering



- writes are sent to **store buffers** (one per process)
- writes are committed to memory at any time
- reads are from
  - own store buffer if a value exists (last write to the variable)
  - otherwise from the memory
- fences executed when own buffer is empty

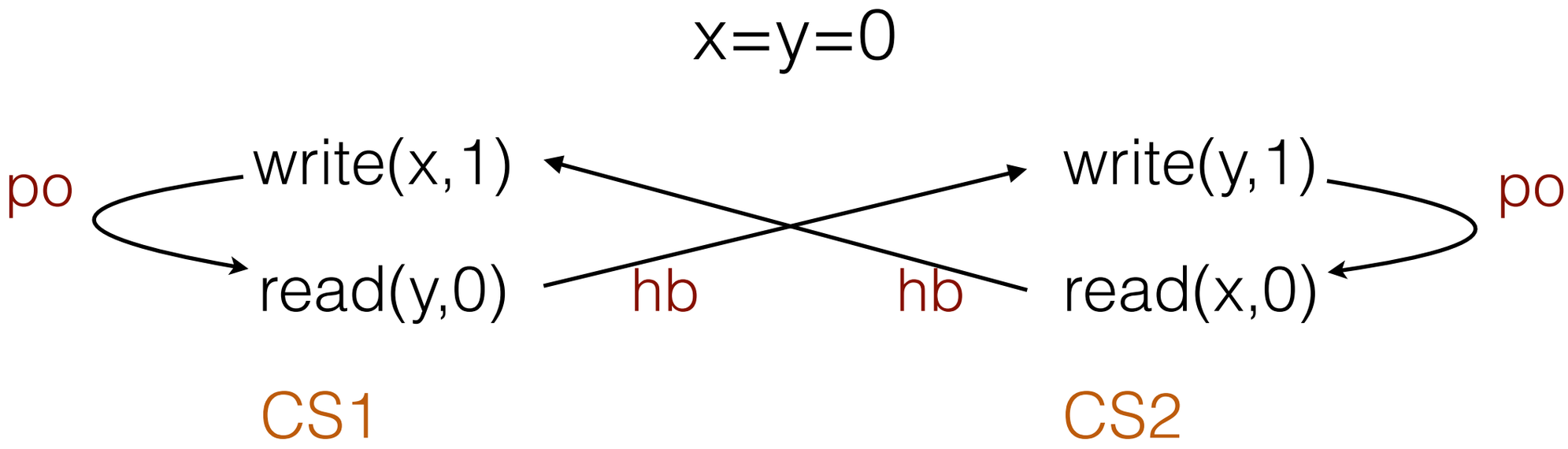
Non SC Behaviours

x=y=0

write(x,1)	write(y,1)
read(y,0)	read(x,0)
CS1	CS2

CS1 and CS2 ?

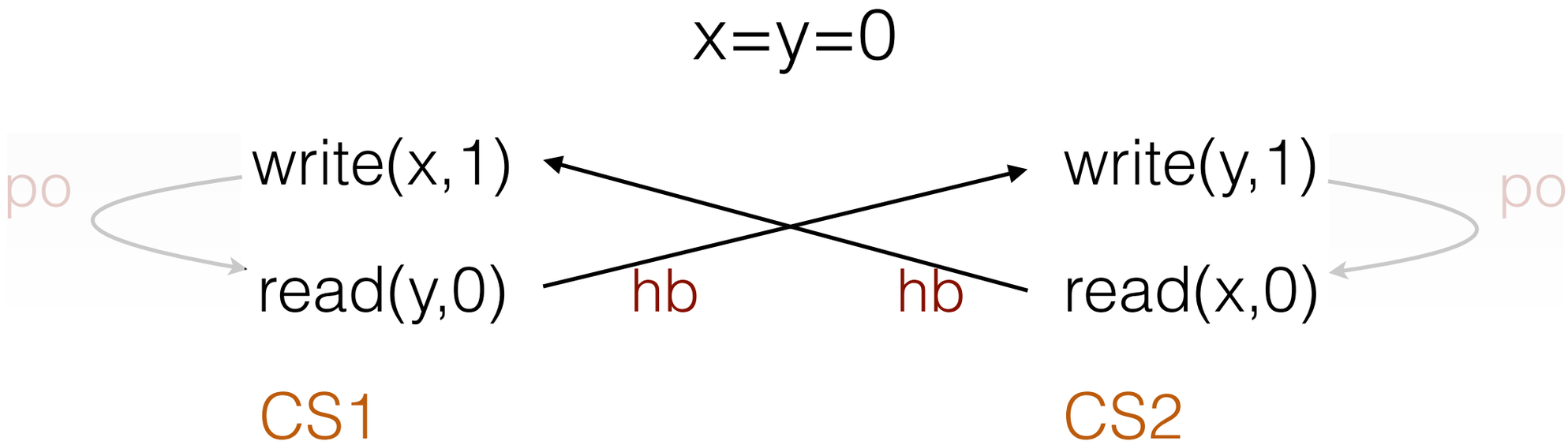
Non SC Behaviours



CS1 and CS2 ?

- **Impossible under SC**

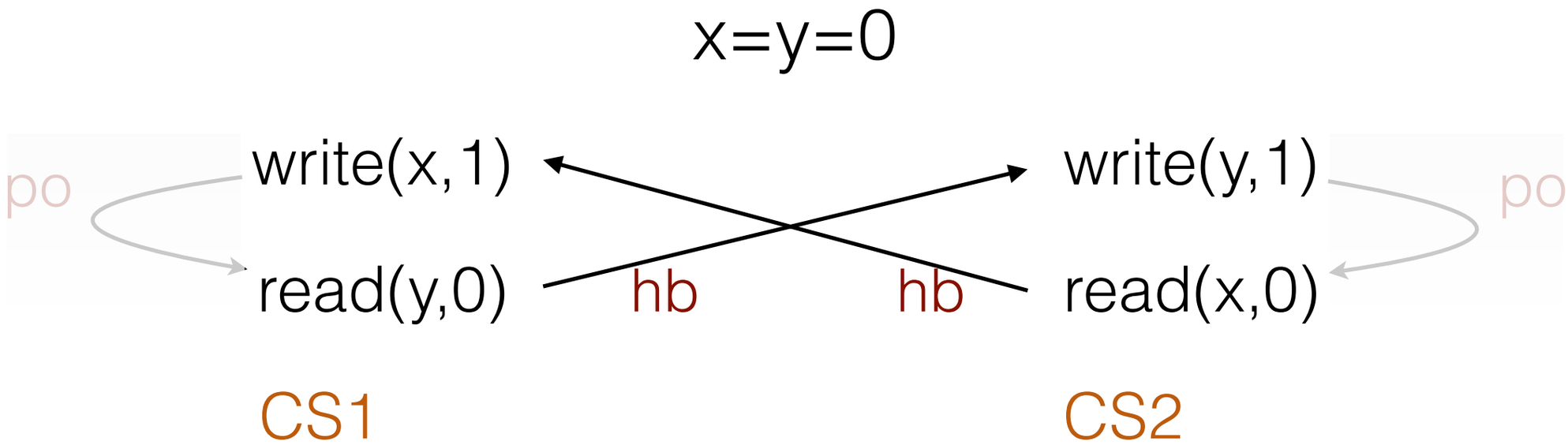
Non SC Behaviours



CS1 and CS2 ?

- **Impossible under SC**
- **Possible under TSO!**
  - writes are **delayed**: pending in store buffers
  - reads get old values in the memory (0's)

Non SC Behaviours



CS1 and CS2 ?

- **Impossible under SC**
- **Possible under TSO!**
  - writes are **delayed**: pending in store buffers
  - reads get old values in the memory (0's)
  - => po constraints are **relaxed**
  - => reads can **overtake** writes

TSO: Semantics

P1

> w(x,1)

r(y,0)

P2

> w(y,1)

r(x,0)

TSO: Semantics

P1

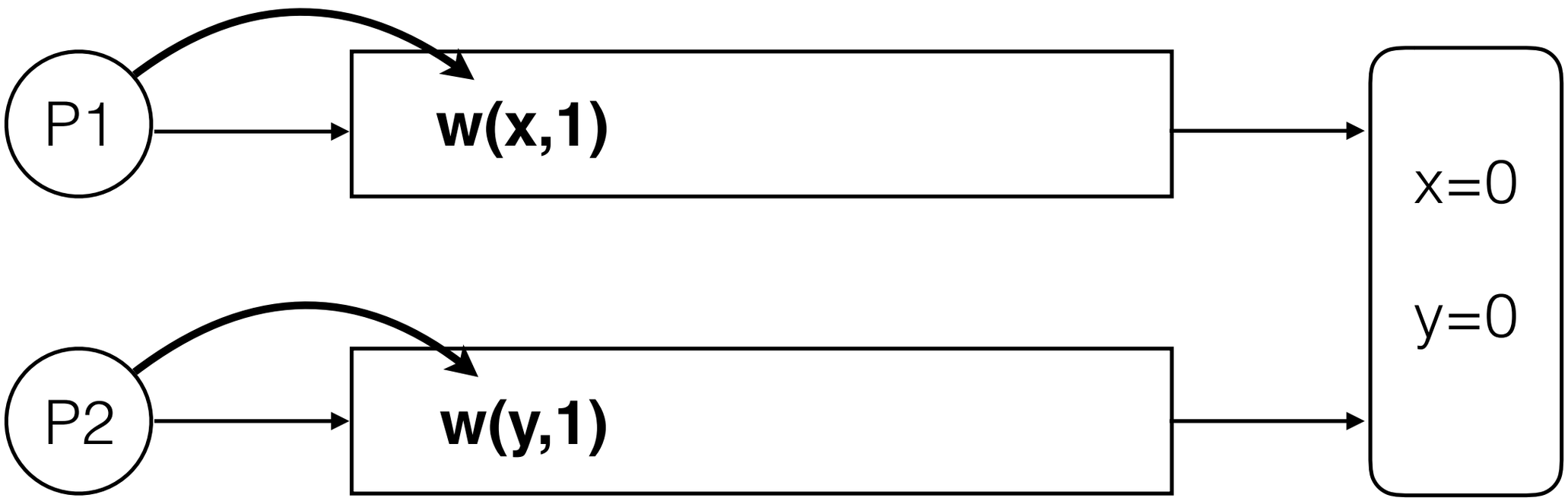
w(x,1)

r(y,0)

P2

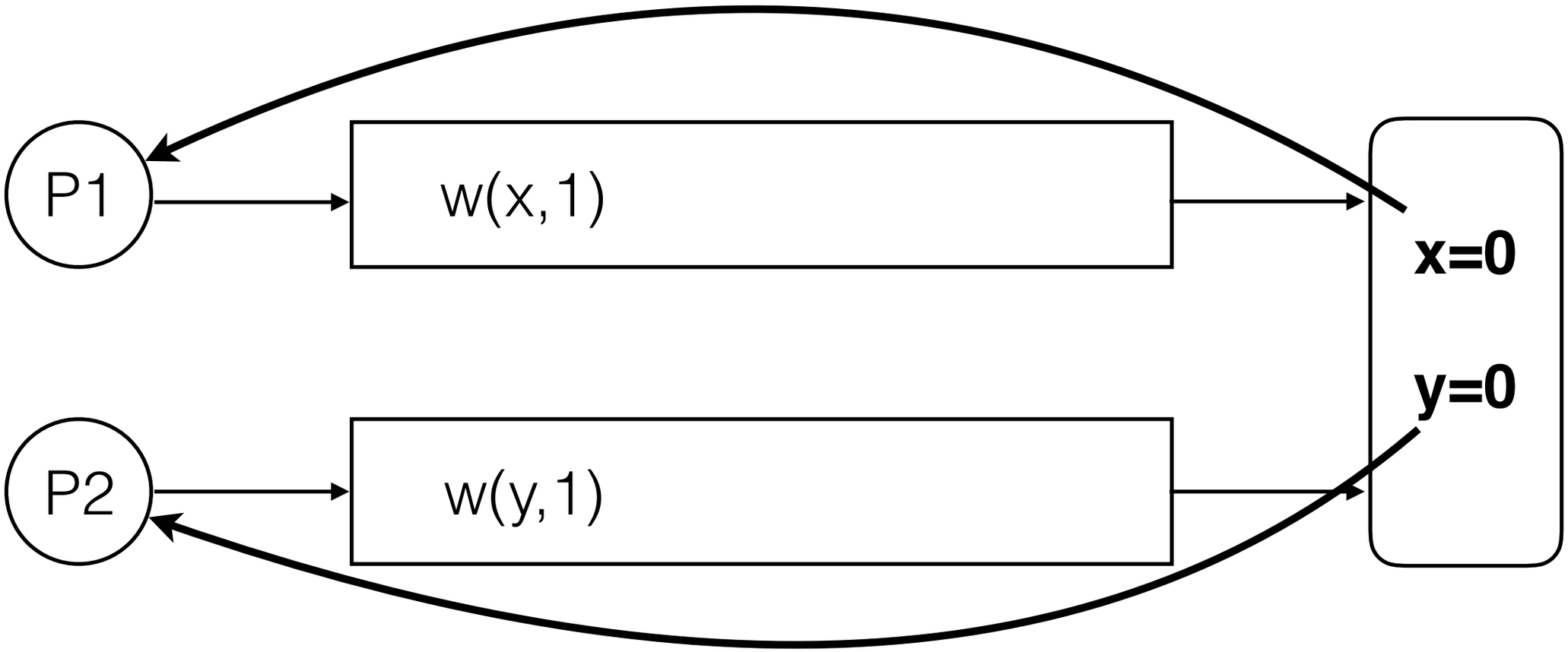
w(y,1)

r(x,0)

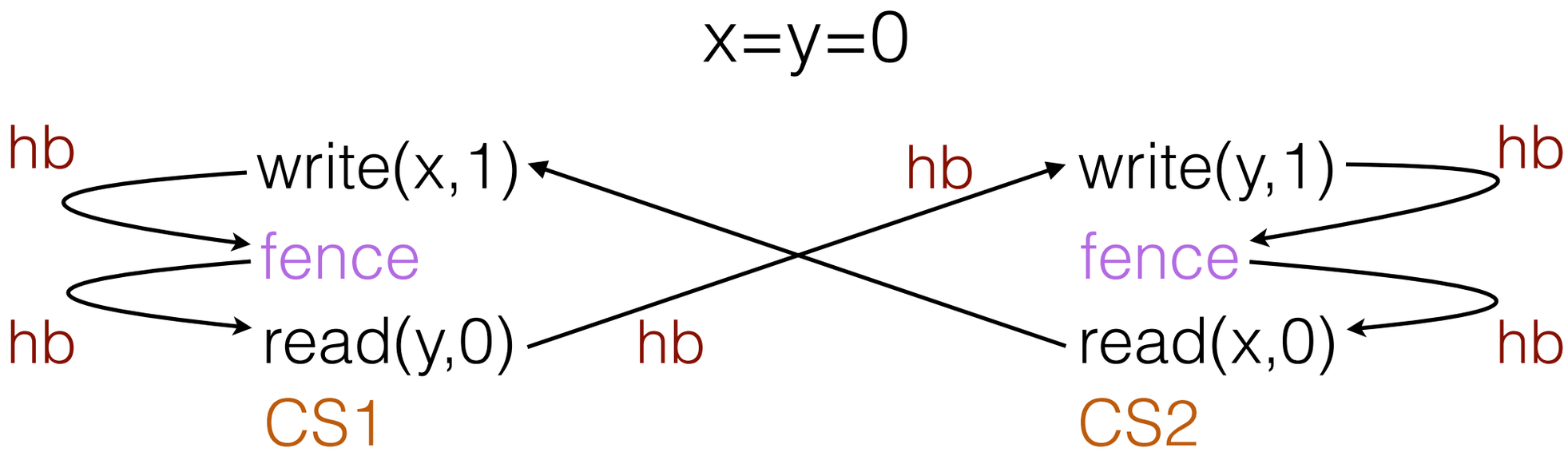


TSO: Semantics

P1	P2
w(x,1)	w(y,1)
> r(y,0)	> r(x,0)



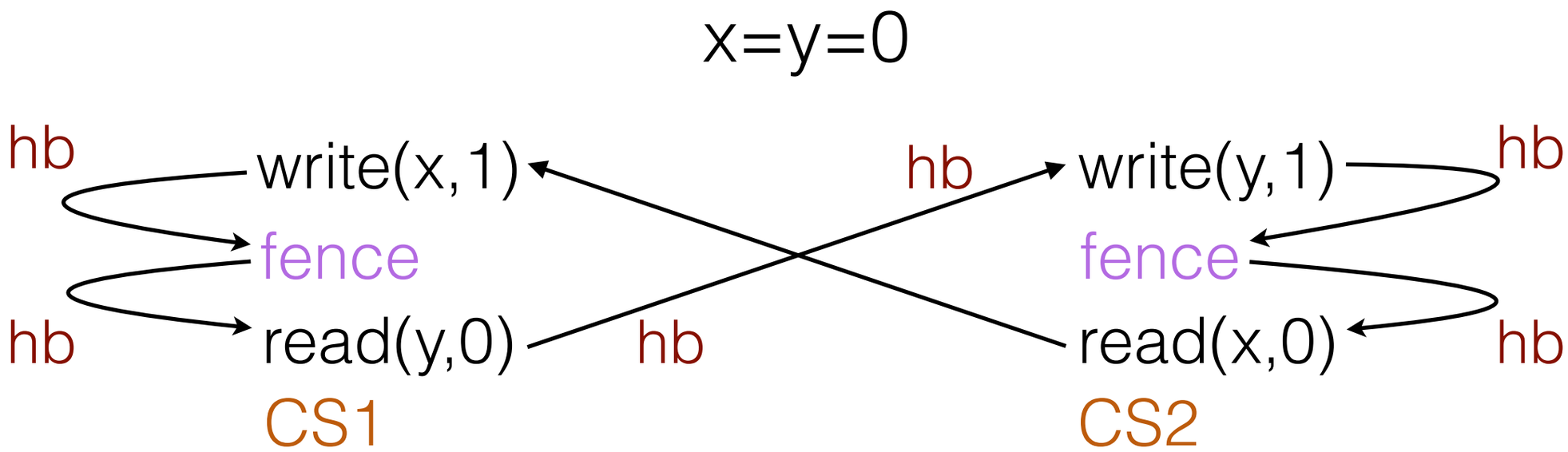
Avoiding Reordering: Fences



CS1 and CS2 ?

- A fence forces flushing the store buffer
- => CS1 and CS2 becomes impossible

Avoiding Reordering: Fences

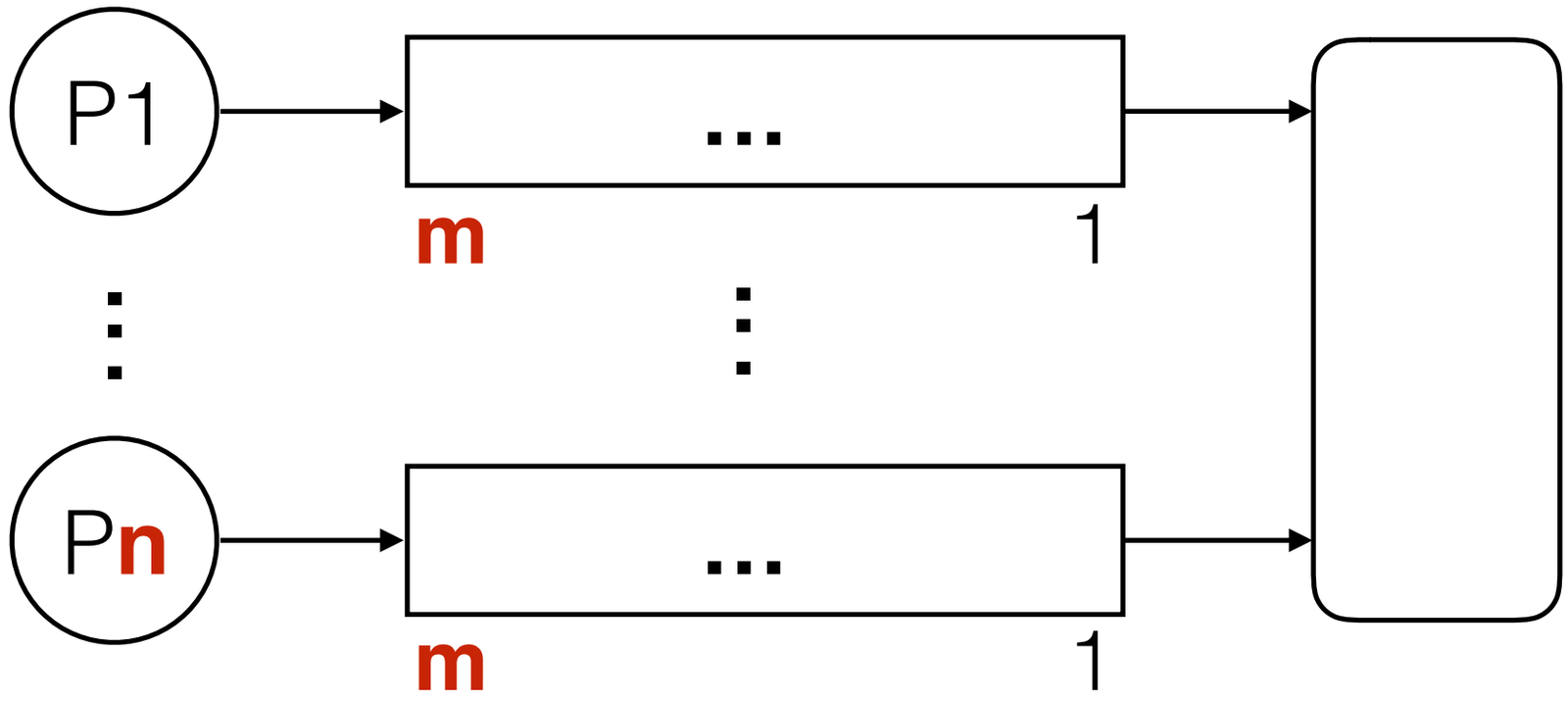


CS1 and CS2 ?

- A fence forces flushing the store buffer
- => CS1 and CS2 becomes impossible

SC can be enforced: fence after each write

Safety/Reachability Verification Problems



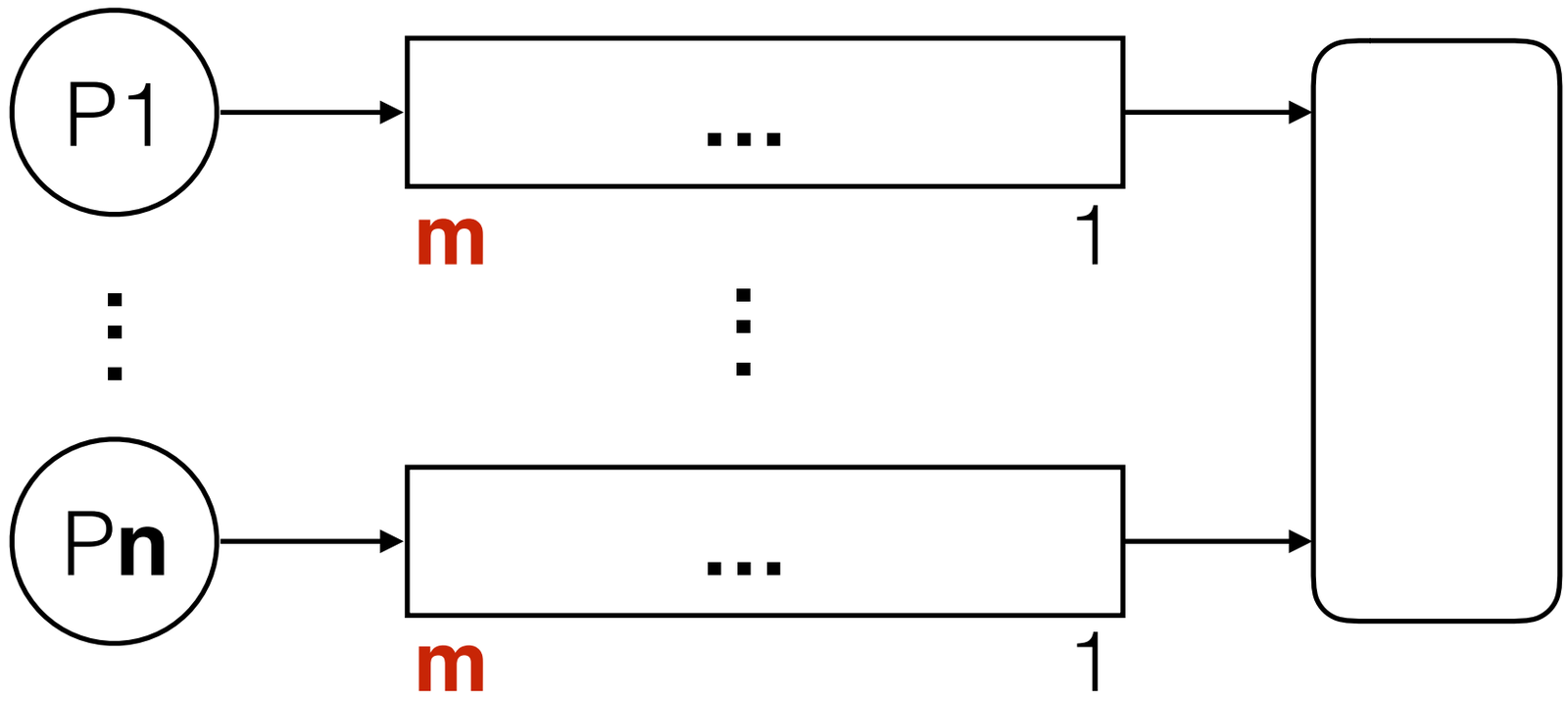
for every n, for every m,

[P1 || … || P**n** **]TSO(m)** **satisfies** **Always (Safe)**

there is n, there is m,

[P1 || … || P**n** **]TSO(m)** **satisfies** **Reachable (Not Safe)**

First step: Let us fix the number of processes



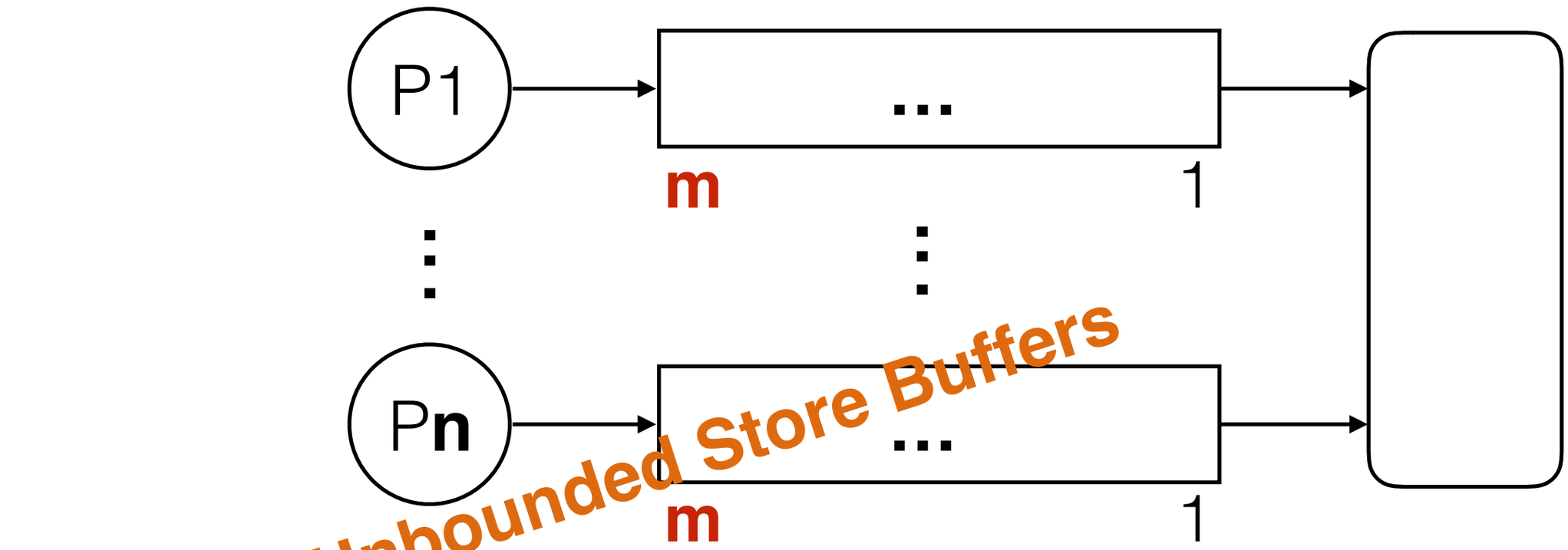
for every m,

[P1 || … || Pn]TSO(m) satisfies Always (Safe)

there is m,

[P1 || … || Pn]TSO(m) satisfies Reachable (Not Safe)

First step: Let us fix the number of processes



there is m,

[P1 || … || Pn]TSO(m) satisfies Reachable (Not Safe)

<=>

[P1 || … || Pn]TSO(∞) satisfies Reachable (Not Safe)

Reachability Problem for a given number of processes: Decidability, Complexity

Assume that processes are finite state

Under **SC**, the control state reachability problem is

- PSPACE-complete, for a fixed number of processes
- EXPSPACE-complete, for the parametric case

Reachability Problem for a given number of processes: Decidability, Complexity

Assume that processes are finite state

Under **SC**, the control state reachability problem is

- PSPACE-complete, for a fixed number of processes
- EXPSPACE-complete, for the parametric case

What about the TSO(∞) reachability?

store buffers are unbounded perfect FIFO queues!!

Reachability Problem for a given number of processes: Decidability, Complexity

Assume that processes are finite state

Under **SC**, the control state reachability problem is

- PSPACE-complete, for a fixed number of processes
- EXPSPACE-complete, for the parametric case

What about the TSO(∞) reachability?

store buffers are unbounded perfect FIFO queues!!

What about the parametric TSO(∞) reachability?

Reachability Problem for TSO programs: Results

- The TSO reachability problem is **decidable**

Reachability Problem for TSO programs: Results

- The TSO reachability problem is **decidable**
- … but it is **highly complex** (non primitive recursive)

Reduction to/from reachability in
lossy channel systems

[Atig, B., Burckhardt, Musuvathi, POPL'10]

Reachability Problem for TSO programs: Results

- The TSO reachability problem is **decidable**

- … but it is **highly complex** (non primitive recursive)

Reduction to/from reachability in
lossy channel systems

[Atig, B., Burckhardt, Musuvathi, POPL'10]

- The **parametric** TSO reachability problem is **decidable**

Reachability Problem for TSO programs: Results

- The TSO reachability problem is **decidable**
- … but it is **highly complex** (non primitive recursive)

Reduction to/from reachability in
lossy channel systems

[Atig, B., Burckhardt, Musuvathi, POPL'10]

- The **parametric** TSO reachability problem is **decidable**
  - A **dual semantics for TSO**
  - Monotonic system w.r.t. WQO
- **Simpler and more efficient reduction**

[Abdulla, Atig, B.,Ngo, CONCUR'16]

An example of TSO program

x=y=0

P1

> w(x,1)

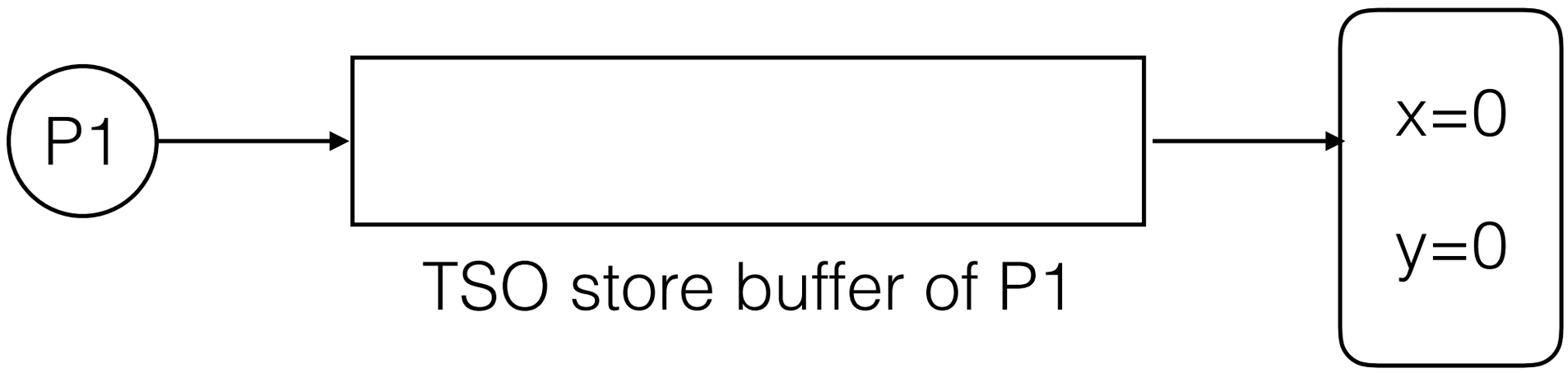
w(y,1)

w(x,2)

P2

> r(x,2)

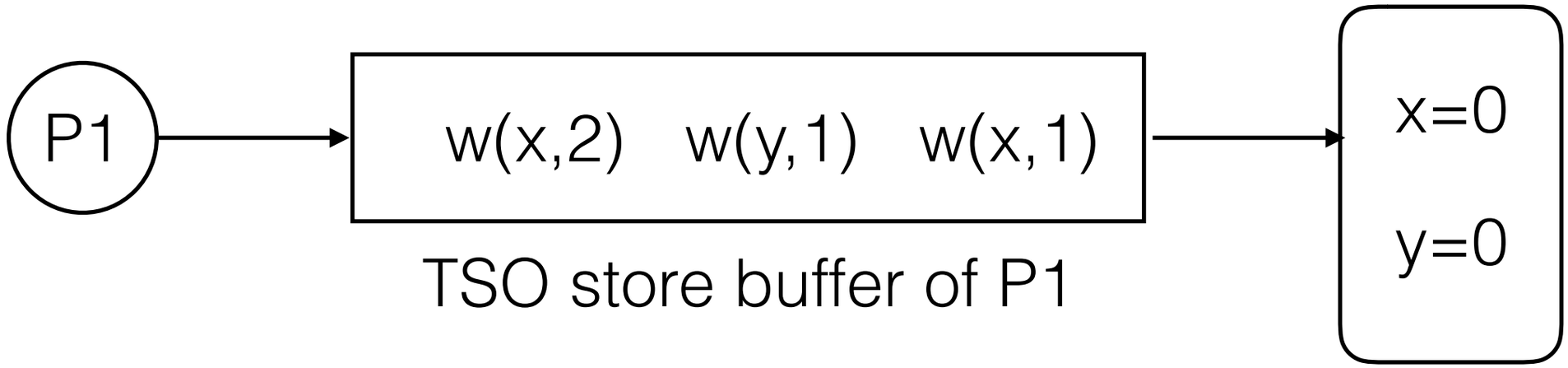
r(y,0)



An example of TSO program

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	

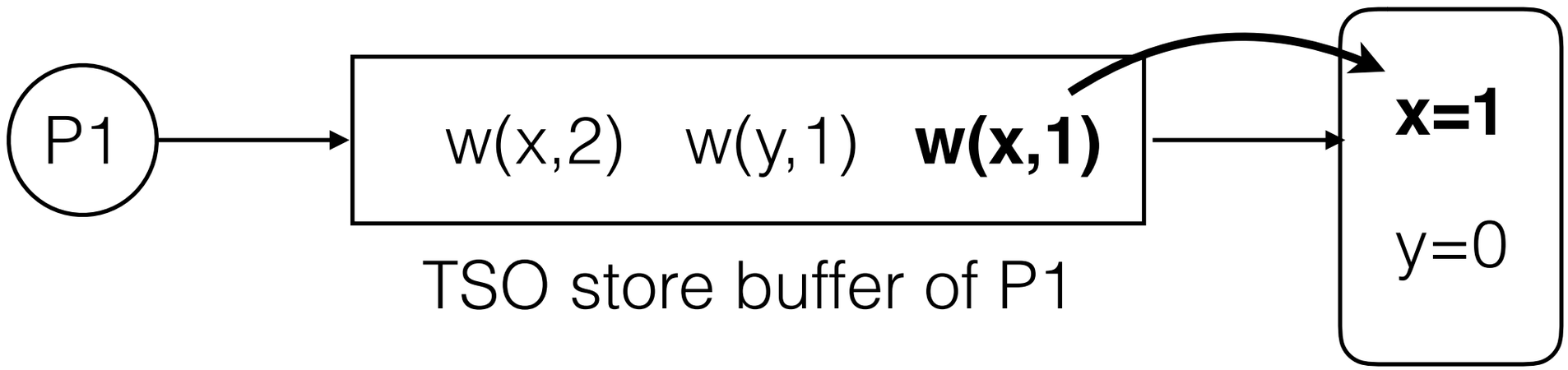


TSO store buffer of P1

An example of TSO program

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	

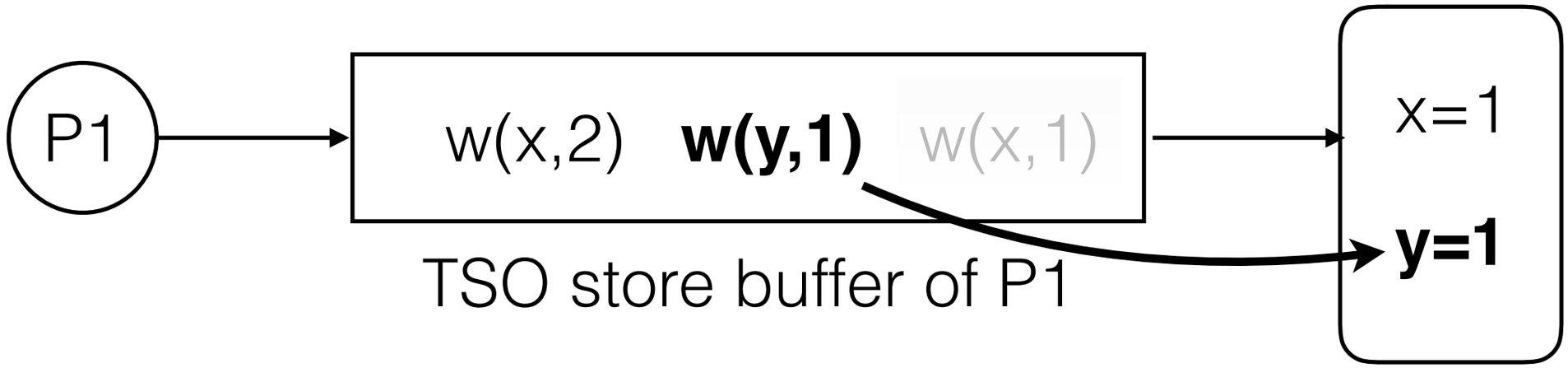


TSO store buffer of P1

An example of TSO program

x=y=0

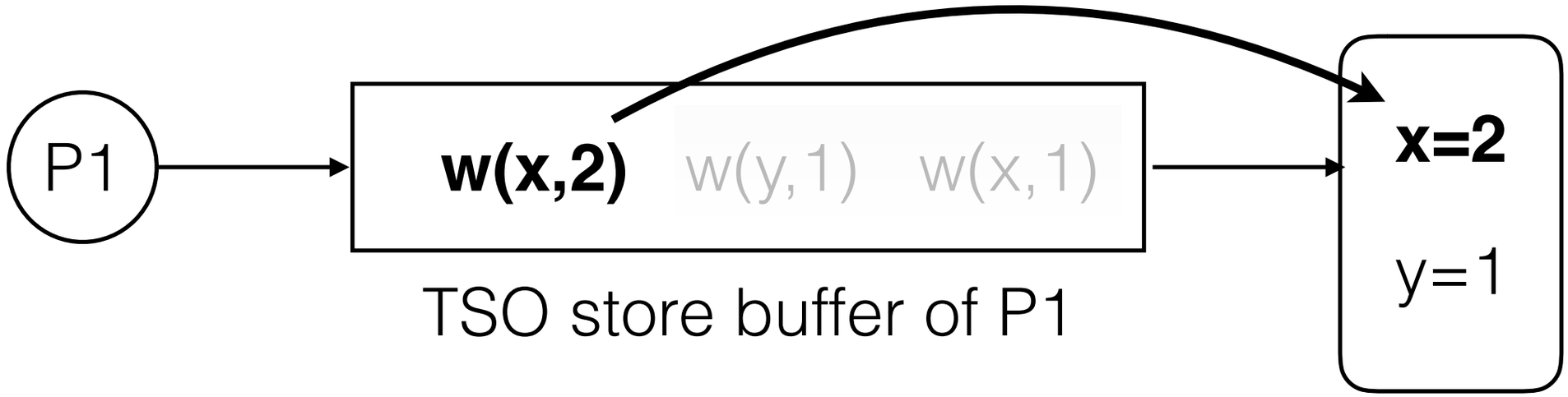
P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	



An example of TSO program

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	

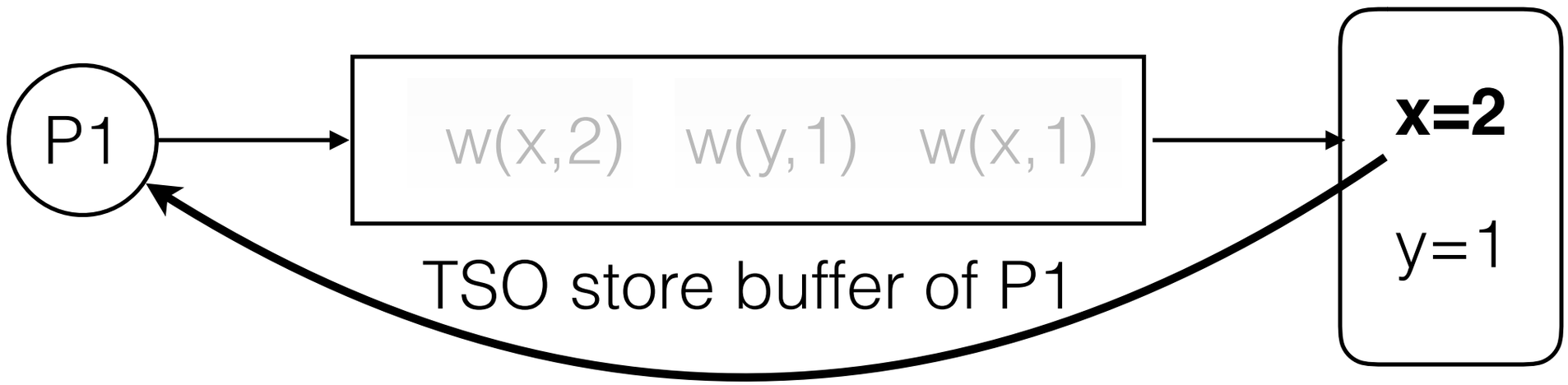


TSO store buffer of P1

An example of TSO program

x=y=0

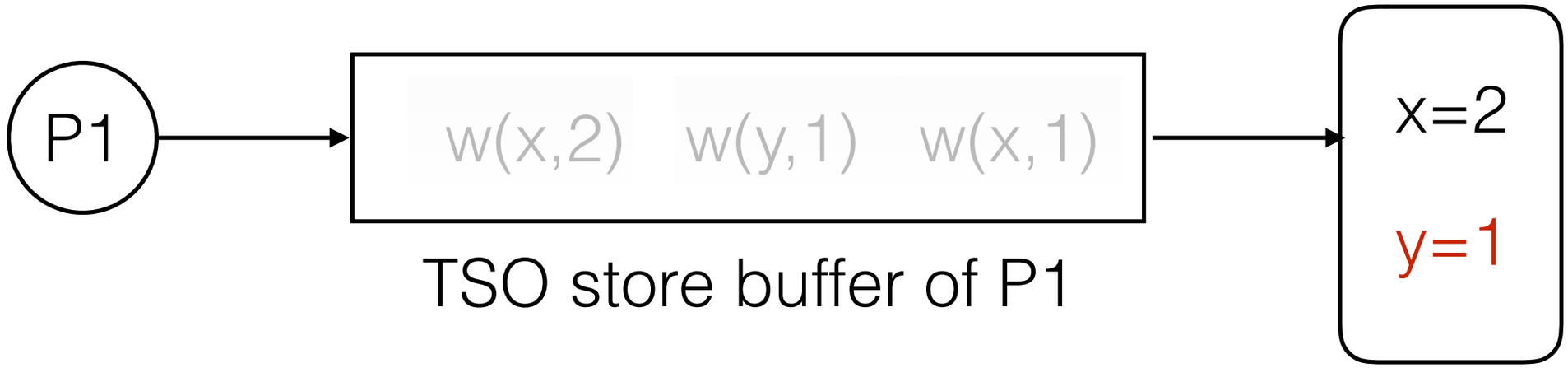
P1		P2
w(x,1)		**r(x,2)**
w(y,1)		r(y,0)
w(x,2)		



An example of TSO program

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	

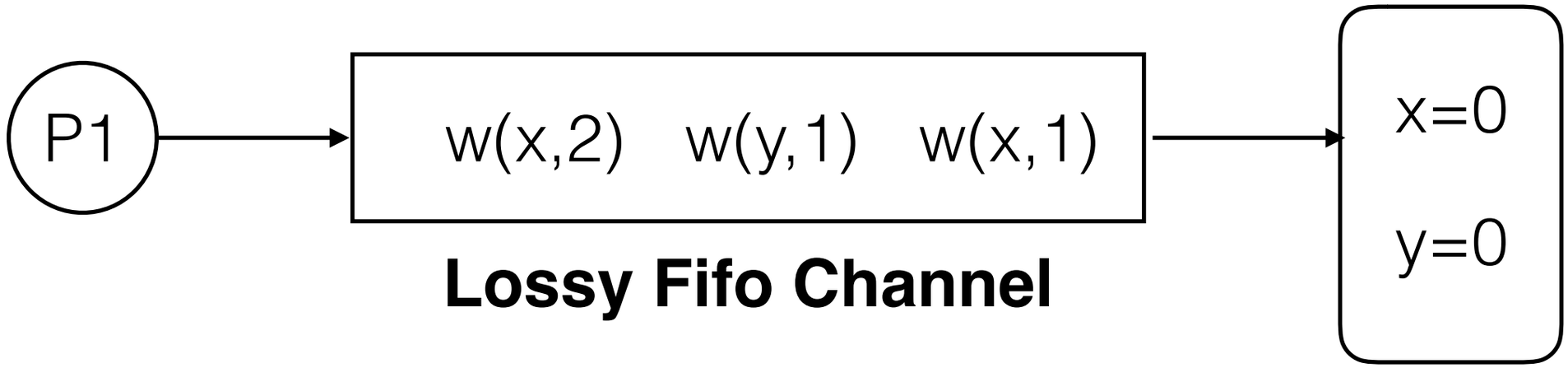


Deadlock under the TSO semantics

TSO Store Buffers —> Lossy Channels ?

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	

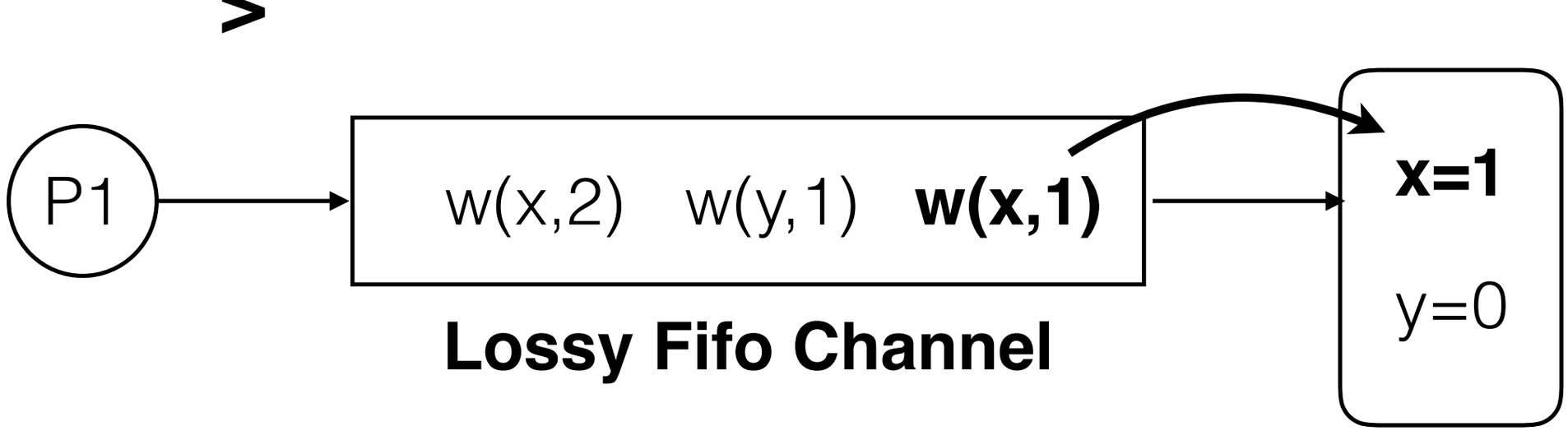


Lossy Fifo Channel

TSO Store Buffers —> Lossy Channels ?

x=y=0

P1	P2
w(x,1)	> r(x,2)
w(y,1)	r(y,0)
w(x,2)	
>	



Lossy Fifo Channel

TSO Store Buffers —> Lossy Channels ?

x=y=0

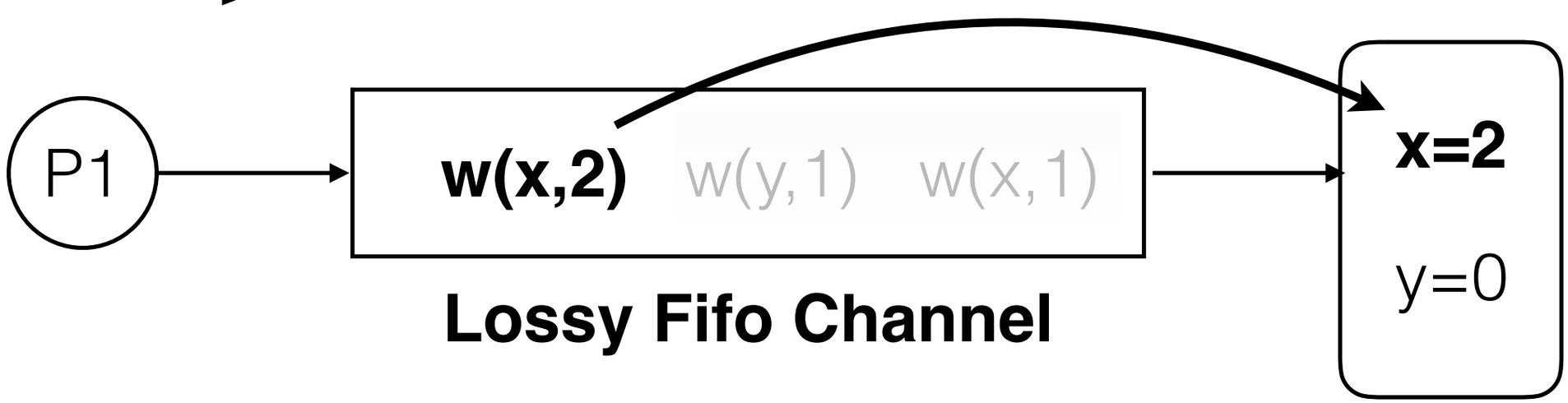
P1	P2
w(x,1)	> r(x,2)
w(y,1)	r(y,0)
w(x,2)	
>	

Lossy Fifo Channel

TSO Store Buffers —> Lossy Channels ?

x=y=0

P1	P2
w(x,1)	> r(x,2)
w(y,1)	r(y,0)
w(x,2)	



TSO Store Buffers —> Lossy Channels ?

x=y=0

P1

w(x,1)

w(y,1)

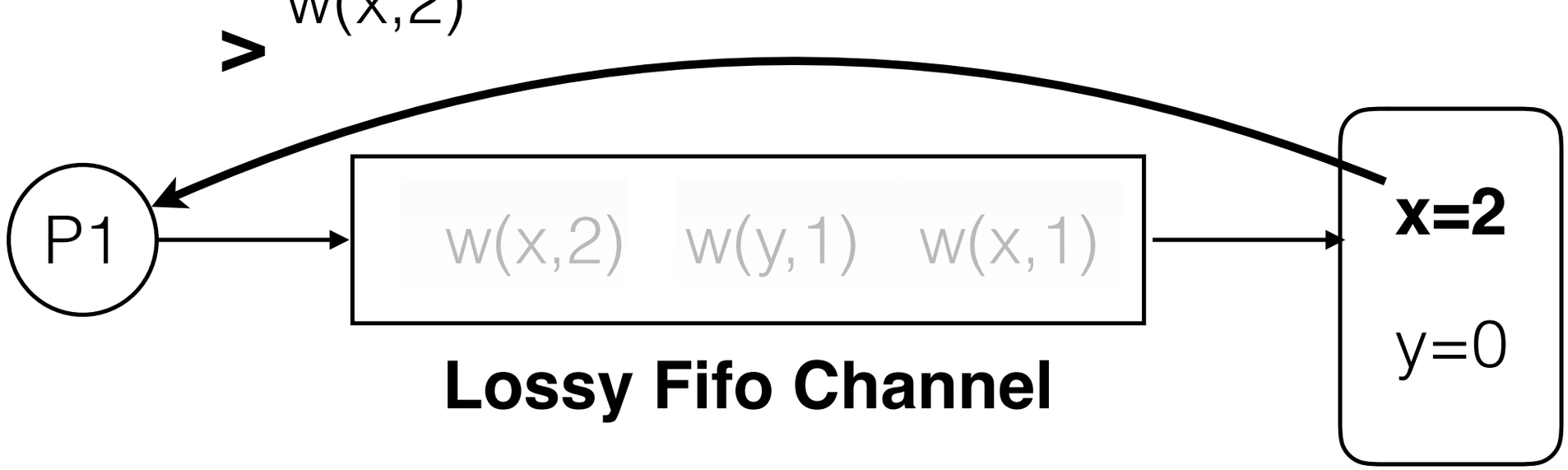
w(x,2)

>

P2

> **r(x,2)**

r(y,0)



Lossy Fifo Channel

TSO Store Buffers —> Lossy Channels ?

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	**r(y,0)**
w(x,2)	

Unsound simulation of TSO!

Store Memory Snapshots

x=y=0

P1

> w(x,1)

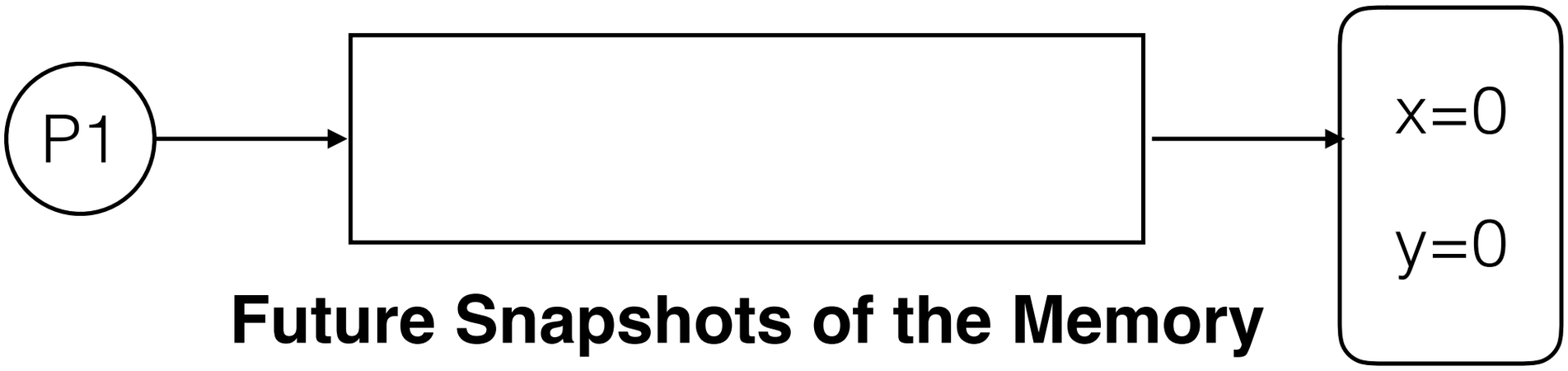
w(y,1)

w(x,2)

P2

> r(x,2)

r(y,0)



Future Snapshots of the Memory

Store Memory Snapshots

x=y=0

P1

w(x,1)

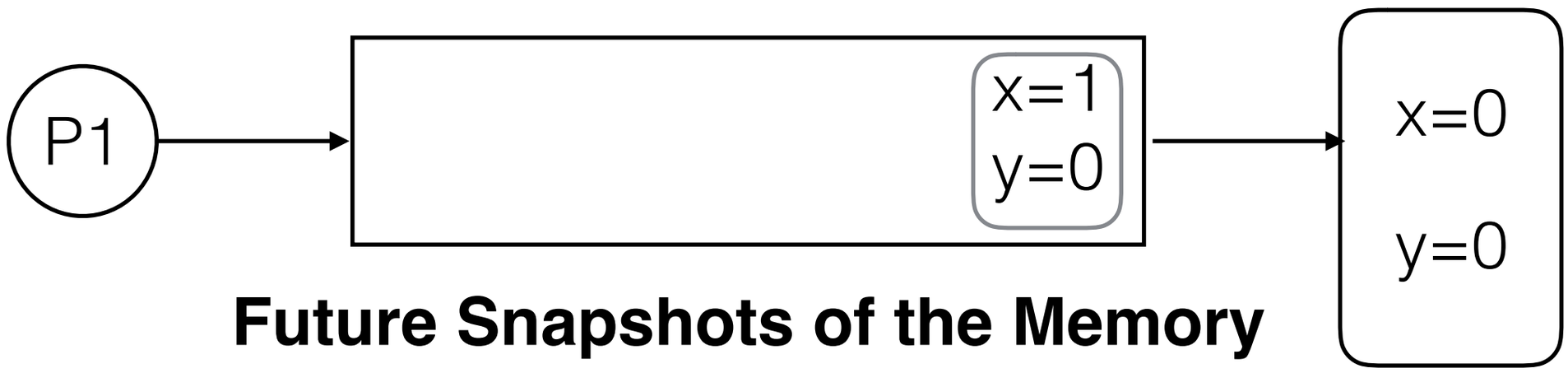
w(y,1)

w(x,2)

P2

r(x,2)

r(y,0)



Future Snapshots of the Memory

Store Memory Snapshots

x=y=0

P1

w(x,1)

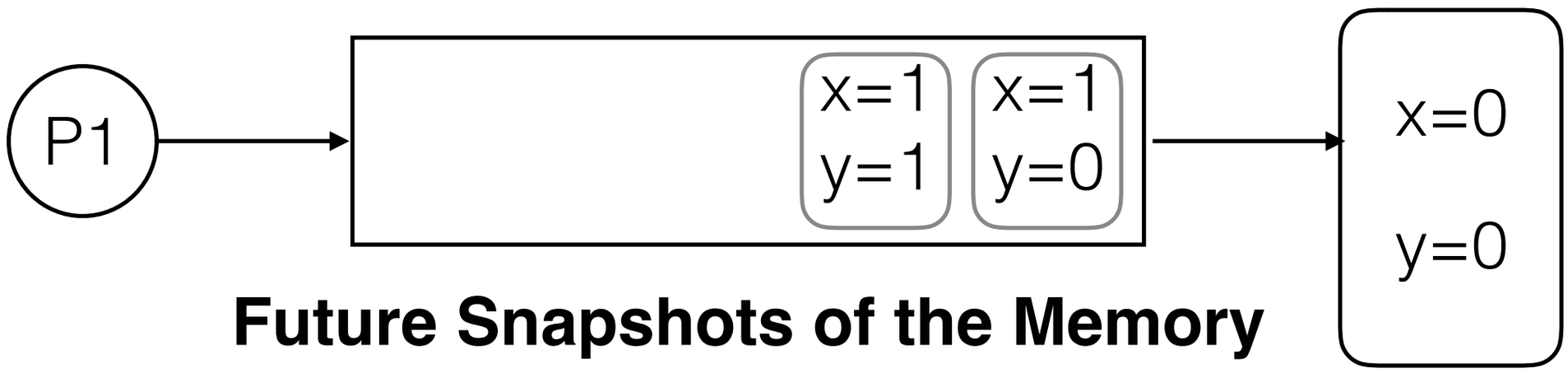
w(y,1)

w(x,2)

P2

r(x,2)

r(y,0)



Future Snapshots of the Memory

Store Memory Snapshots

x=y=0

P1

w(x,1)

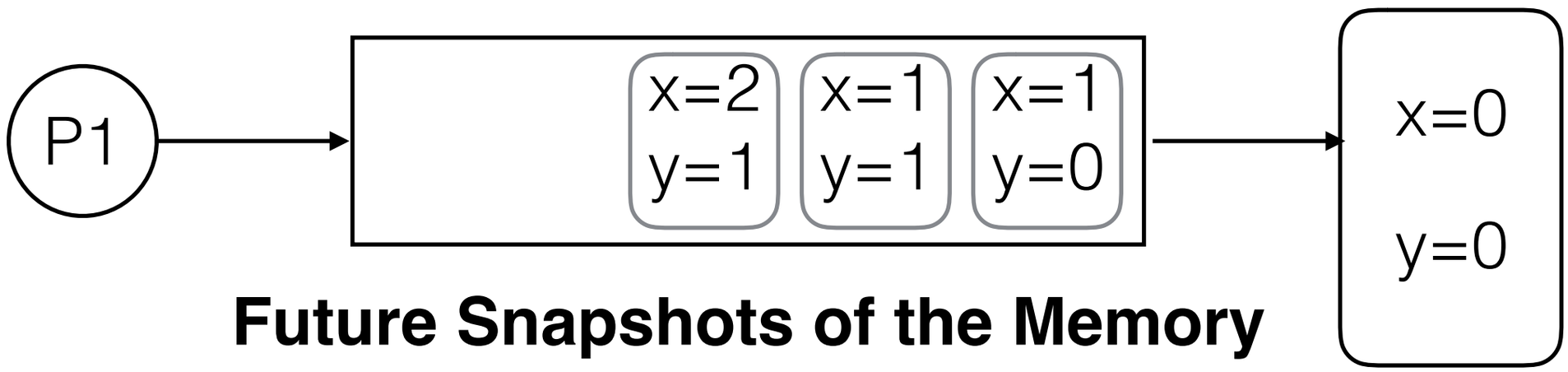
w(y,1)

w(x,2)

P2

r(x,2)

r(y,0)



Future Snapshots of the Memory

Store Memory Snapshots

x=y=0

P1

w(x,1)

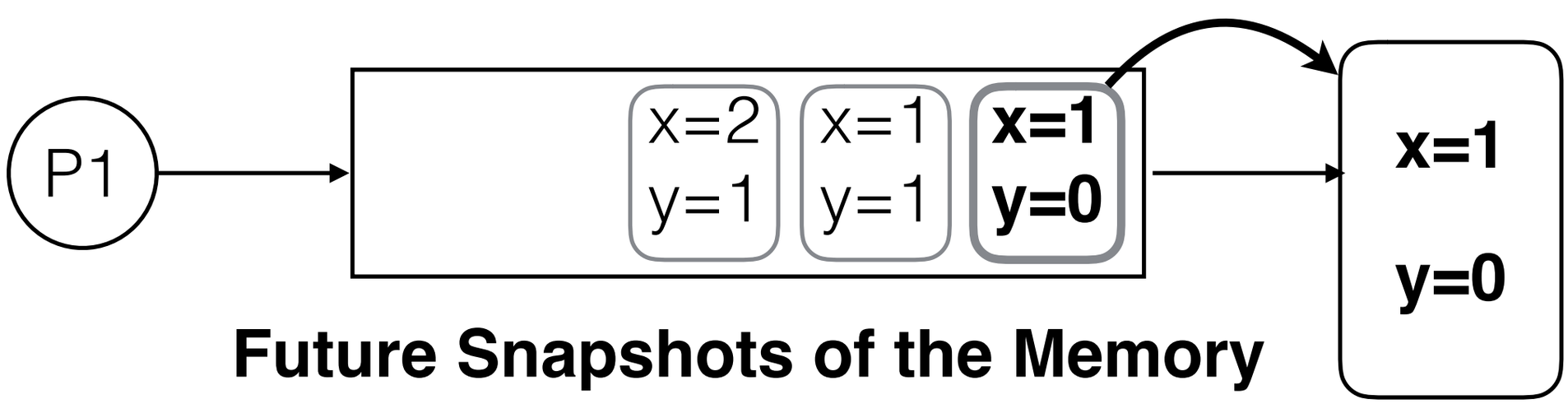
w(y,1)

w(x,2)

P2

r(x,2)

r(y,0)



Future Snapshots of the Memory

Store Memory Snapshots with Losses

x=y=0

P1

w(x,1)

w(y,1)

w(x,2)

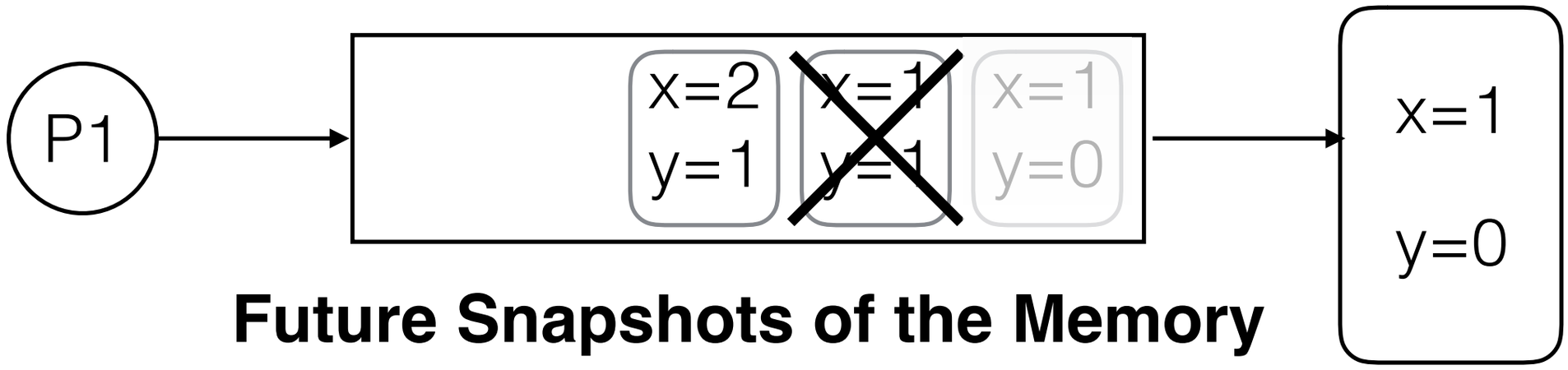
>

P2

>

r(x,2)

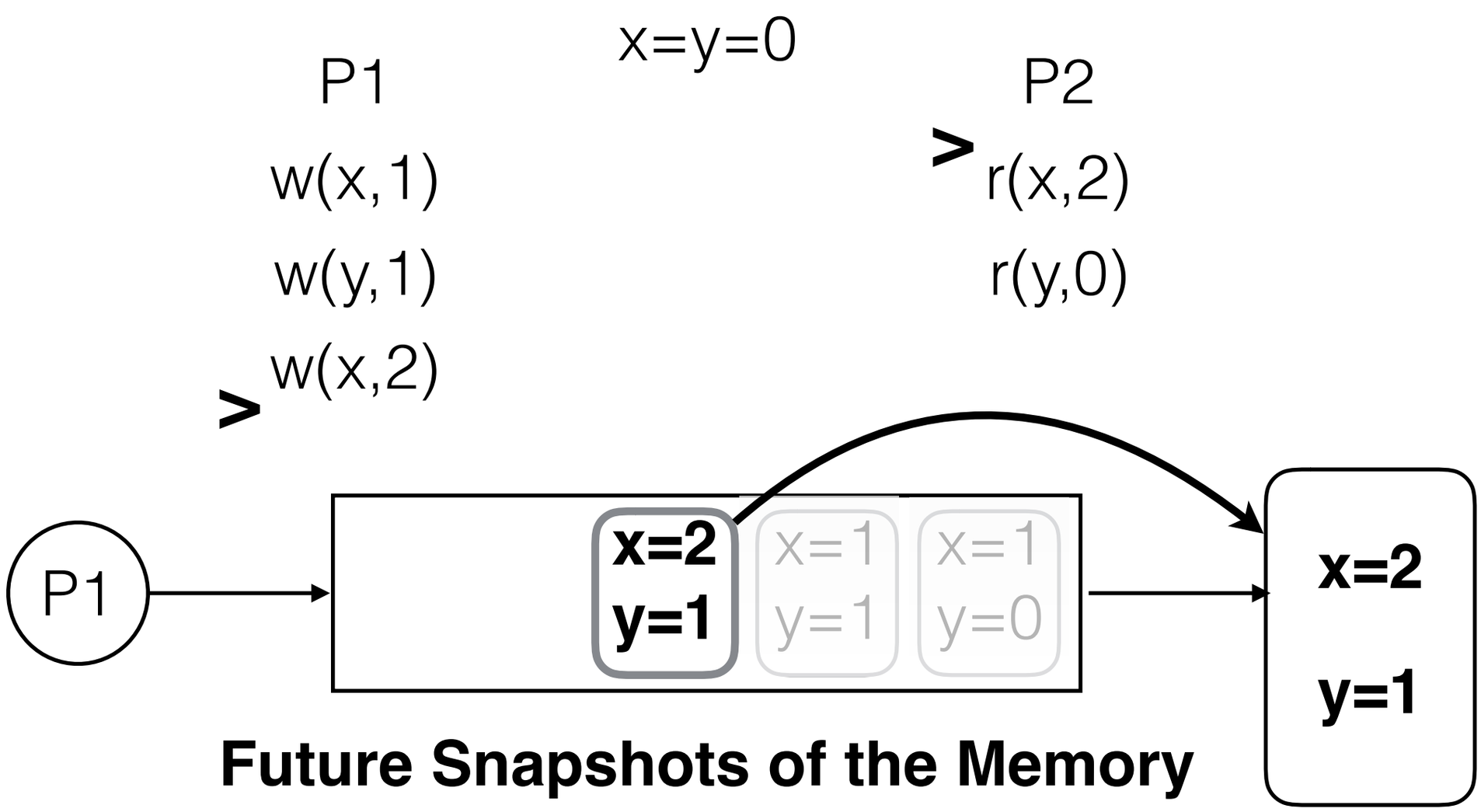
r(y,0)



Future Snapshots of the Memory

+ Lossyness

Store Memory Snapshots with Losses



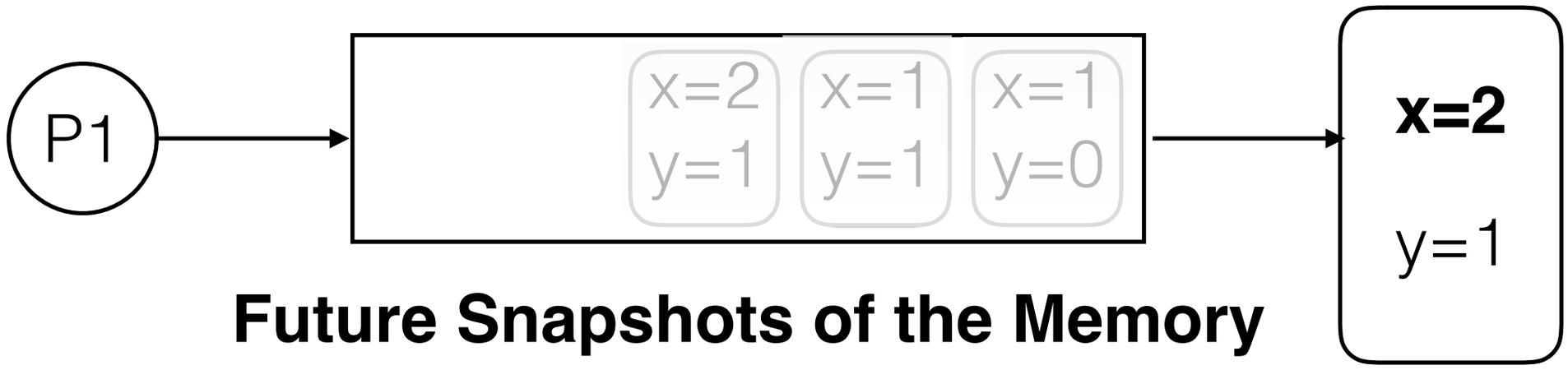
+ Lossyness

Store Memory Snapshots with Losses

x=y=0

P1	P2
w(x,1)	**r(x,2)**
w(y,1)	r(y,0)
w(x,2)	

> **>**



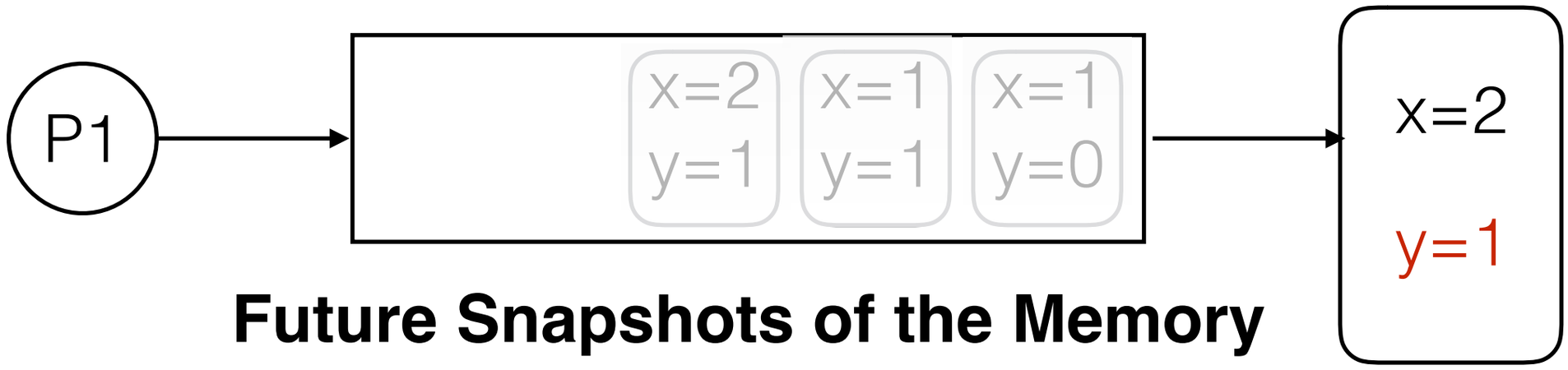
Future Snapshots of the Memory

+ Lossyness

Store Memory Snapshots with Losses

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	



Future Snapshots of the Memory

+ Lossyness

Valid Simulation of TSO

From TSO to Lossy Channel Systems

- 1-channel machine per process + composition

From TSO to Lossy Channel Systems

- 1-channel machine per process + composition

- **Each process:**
  - write: puts a new memory state at the tail of the channel
  - read: checks the channel, then the memory
  - memory update: moves the head of the channel to the memory

From TSO to Lossy Channel Systems

- 1-channel machine per process + composition

- **Each process:**
  - write: puts a new memory state at the tail of the channel
  - read: checks the channel, then the memory
  - memory update: moves the head of the channel to the memory

Problem: Interferences between processes ?
Processes must agree on the same order of memory updates

From TSO to Lossy Channel Systems

- 1-channel machine per process + composition

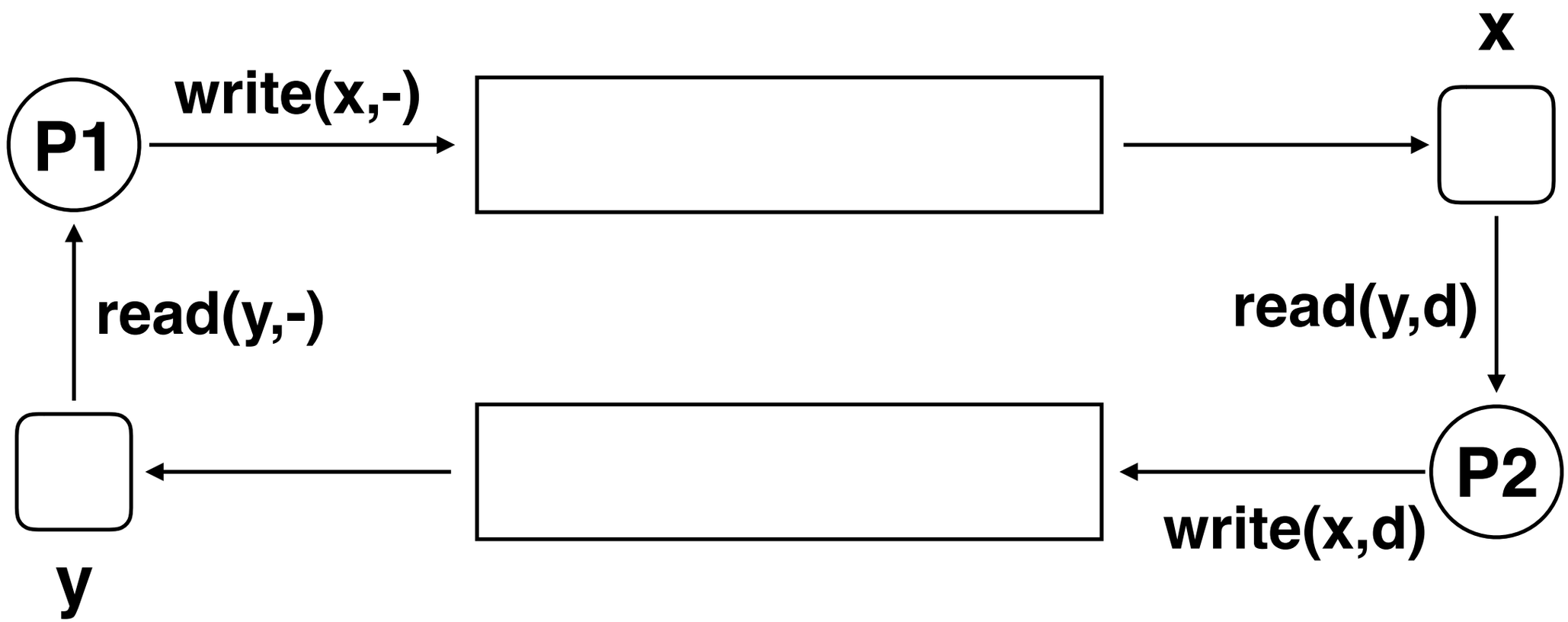
- **Each process:**
  - write: puts a new memory state at the tail of the channel
  - read: checks the channel, then the memory
  - memory update: moves the head of the channel to the memory

Problem: Interferences between processes ?
Processes must agree on the same order of memory updates

- **guesses writes by other processes**; put them in the channel

- **Validation of the guesses by composition:**
  - transitions are labelled by **write operations + process id**
  - machines are **synchronized** on these actions

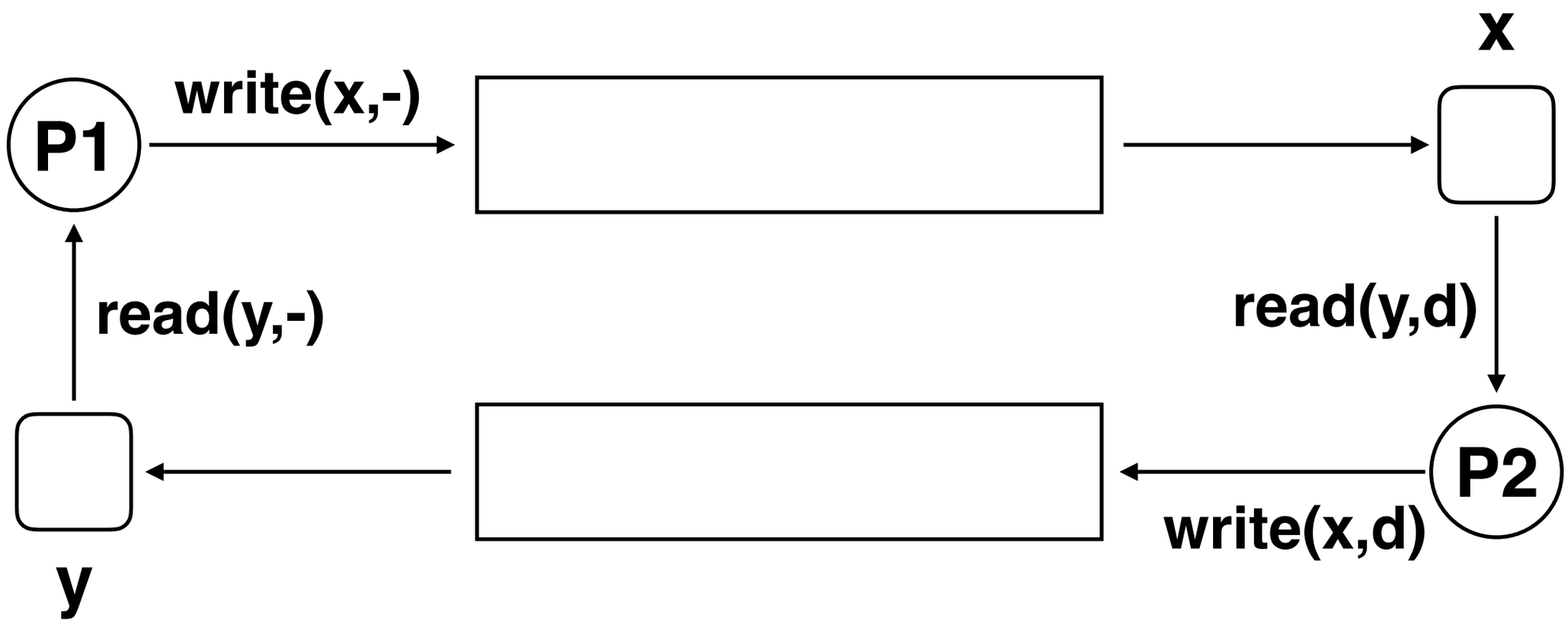
From Lossy Channel Systems to TSO programs



- **P1** simulates a LCS with one channel using **x** and **y**:
  - send(m) —> write(x, m)
  - receive(m) —> read(y,m)

- **P2** forwards values from **x** to **y**

From Lossy Channel Systems to TSO programs



- **P1** simulates a LCS with one channel using **x** and **y**:
  - send(m) —> write(x, m)
  - receive(m) —> read(y,m)
- **P2** forwards values from **x** to **y**

P2 can miss some values

Reachability for TSO programs

[Atig, B., Burckhardt, Musuvathi, 2010]

Thm: The control state reachability problem under TSO is reducible to the reachability problem in lossy channel systems, and vice-versa.

Reachability for TSO programs

[Atig, B., Burckhardt, Musuvathi, 2010]

Thm: The control state reachability problem under TSO is reducible to the reachability problem in lossy channel systems, and vice-versa.

Coro: The control state reachability problem under TSO is decidable, and it is non primitive recursive.

using [Abdulla & Jonsson1993, Abdulla et al. 1996,
Finkel & Schnoebelen 2001, Schnoebelen 2001]

Well …

The complexity is high!

Well …

The complexity is high!

… but this is **not the** main **problem**

Well …

The complexity is high!

… but this is **not the** main **problem**

The **proposed encoding** of TSO programs as LCS's

- **Is not practical**:

  it requires handling **memory snapshots**

- **Can not be extended to the parametric case**

  it manipulates **process id's**

Well …

The complexity is high!

… but this is **not the** main **problem**

The **proposed encoding** of TSO programs as LCS's

- **Is not practical**:

it requires handling **memory snapshots**

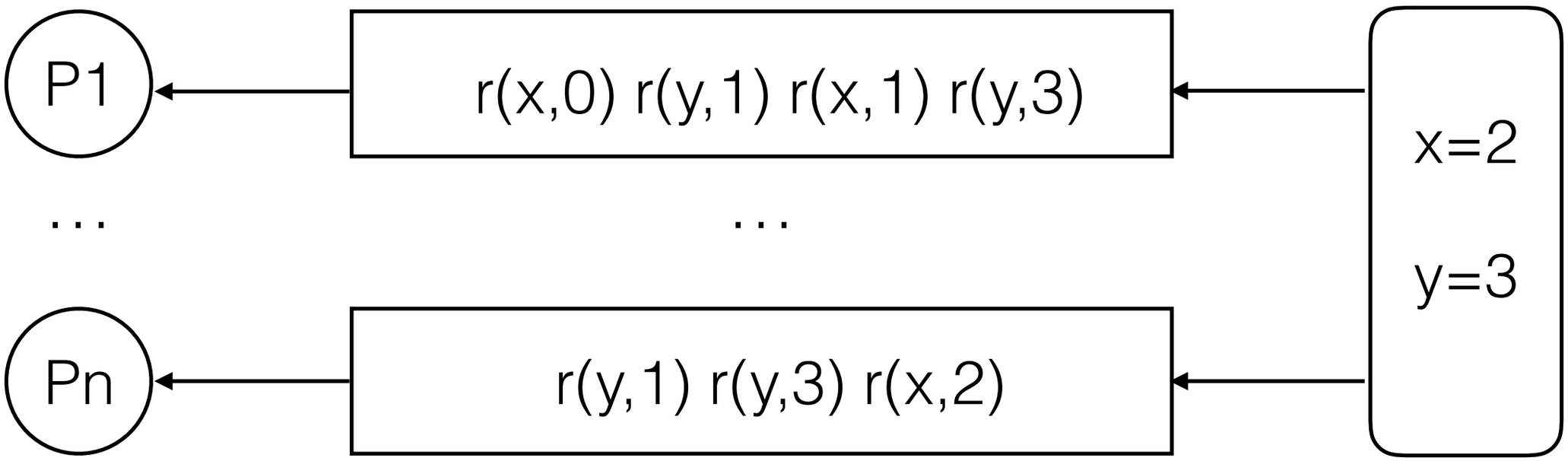
- **Can not be extended to the parametric case**

it manipulates **process id's**

=> We need to change our angle of view…

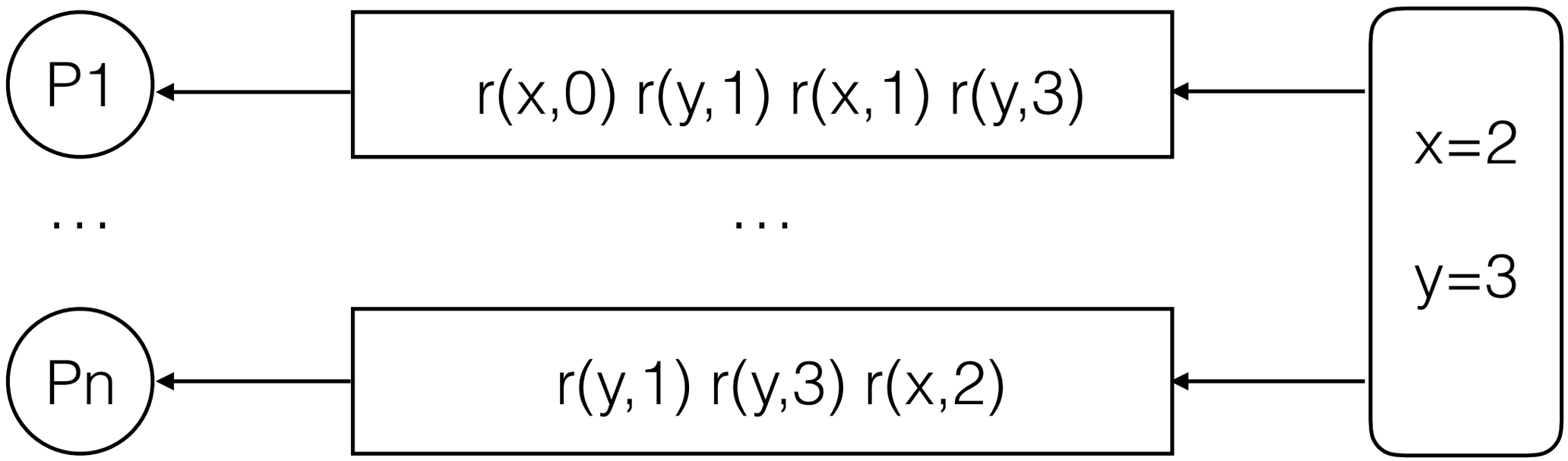
Dual TSO

- **Store Buffers —> Load Buffers**
- Writes **immediately update** the Memory
- Reads are **sent by the memory** to processes
- Reads **can be skipped** by processes (Load Buffers are **lossy**)



Dual TSO

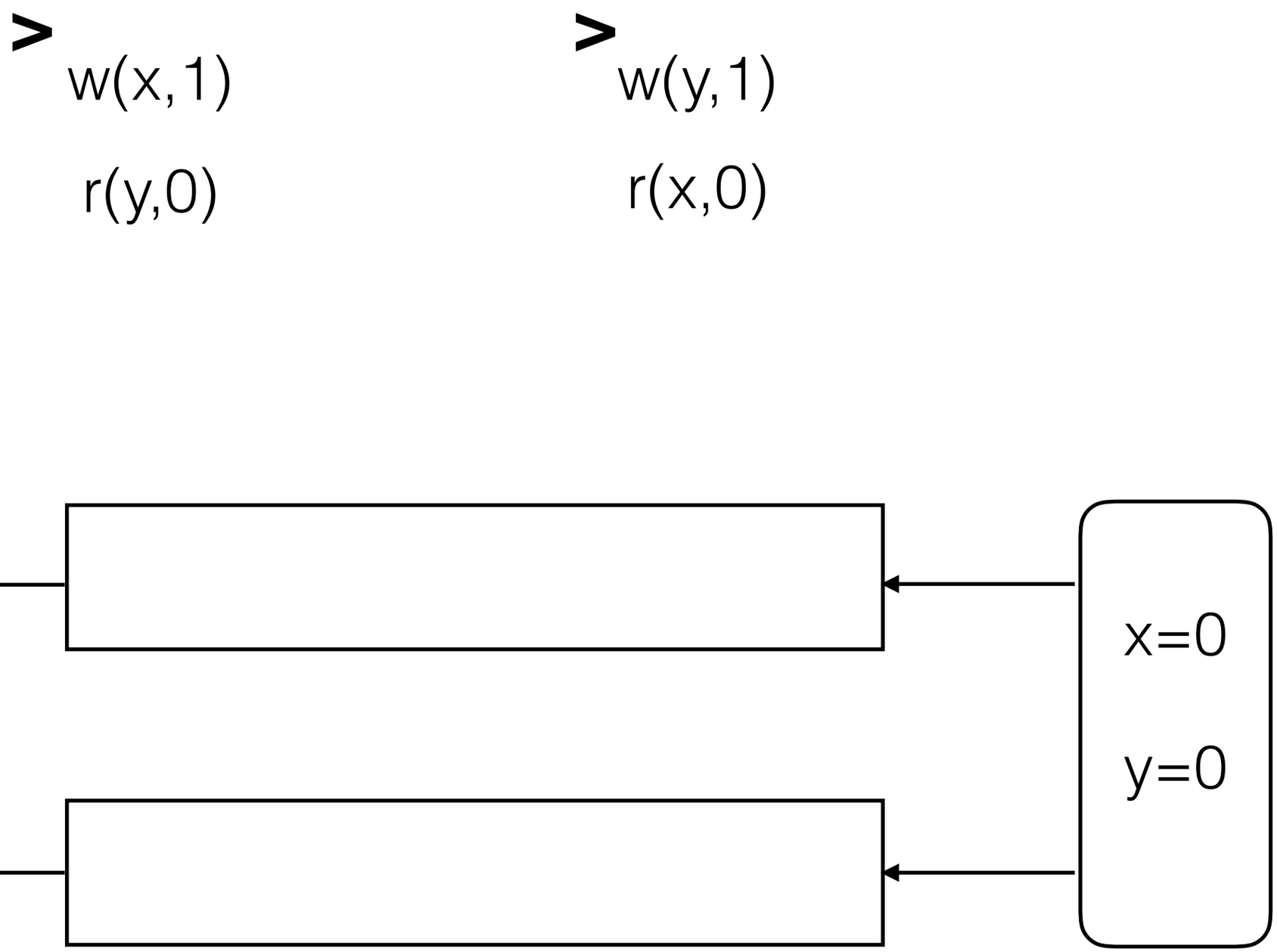
- **Store Buffers —> Load Buffers**
- Writes **immediately update** the Memory
- Reads are **sent by the memory** to processes
- Reads **can be skipped** by processes (Load Buffers are **lossy**)
- => **One** sequence of memory updates (order of writes)
- => Buffers contain **expected reads** by processes
- => Buffers represent a **“(sub)history”** of the memory updates



Dual TSO: Semantics

P1

P2



Dual TSO: Semantics

P1

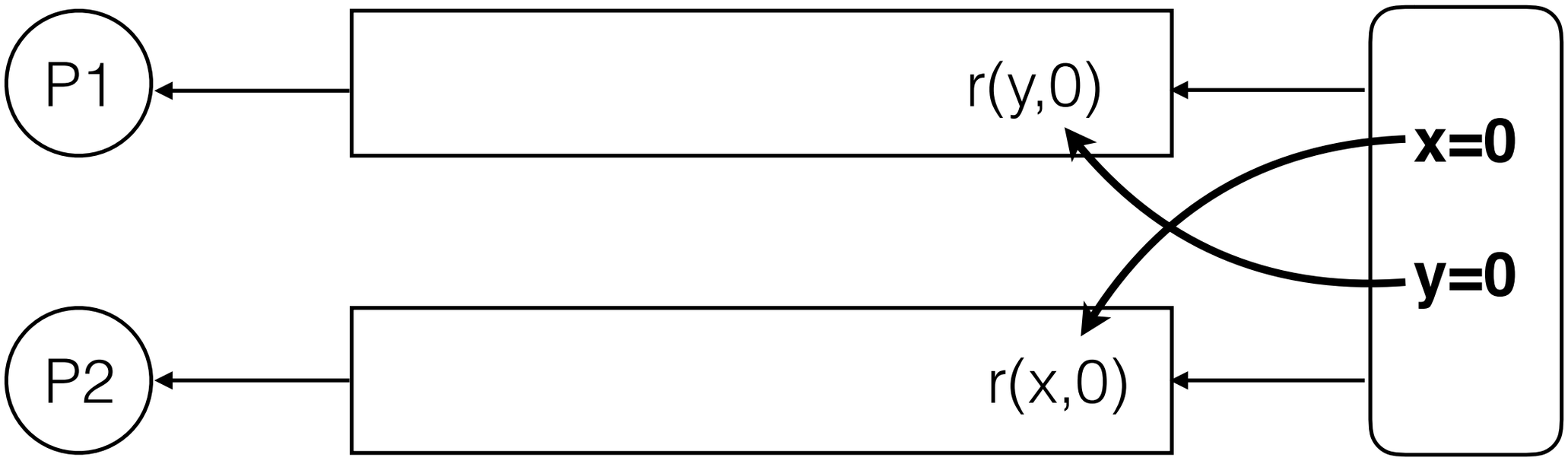
> w(x,1)

r(y,0)

P2

> w(y,1)

r(x,0)



Dual TSO: Semantics

P1

w(x,1)

>

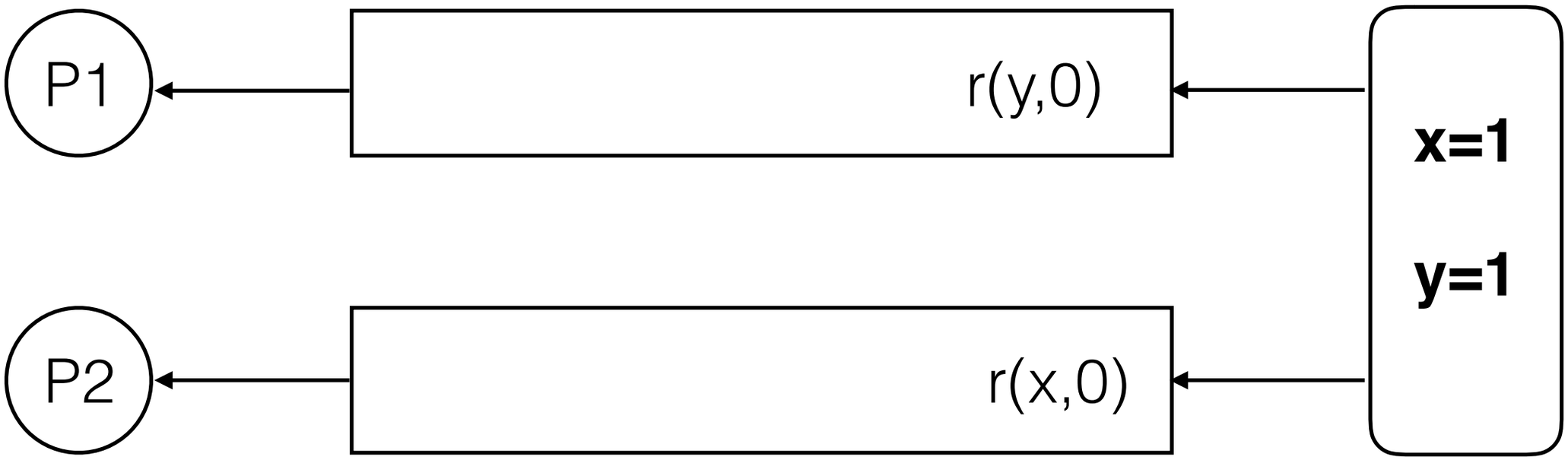
r(y,0)

P2

w(y,1)

>

r(x,0)



Dual TSO: Semantics

P1

w(x,1)

r(y,0)

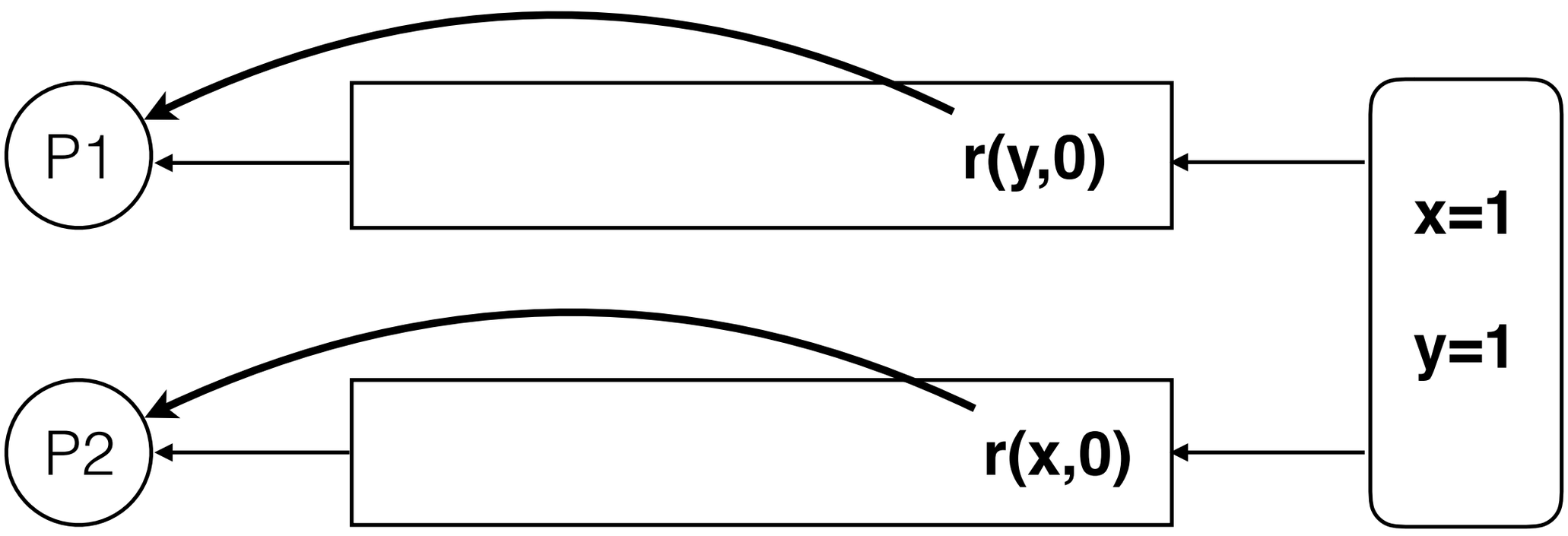
>

P2

w(y,1)

r(x,0)

>



Dual TSO ~ TSO

Thm: The Dual TSO semantics is equivalent to the TSO semantics with respect to the reachability problem.

Parametrized Verification

Given n≥1, a configuration of size n is:

- q1, …, qn, control states of P_1, …, P_n
- B1, …, Bn, contents of the load buffers of P_1, …, P_n
- Mem, the memory state

WQO ≼ between configurations of sizes n and m:

- same memory state
- exists an injective function h: [n] —> [m] s.t.
  - same control state, for each P_i and P'_h(i))
  - sub-word relation on load buffers, for each Pi and P'_h(i))

Thm: Parameterized Dual TSO systems are monotonic w.r.t. ≼

Comparing the two encodings

Dual TSO:

- No memory snapshot
- No reference to Process Id's
- Applicable to Parametric Verification
- More efficient verification algorithm

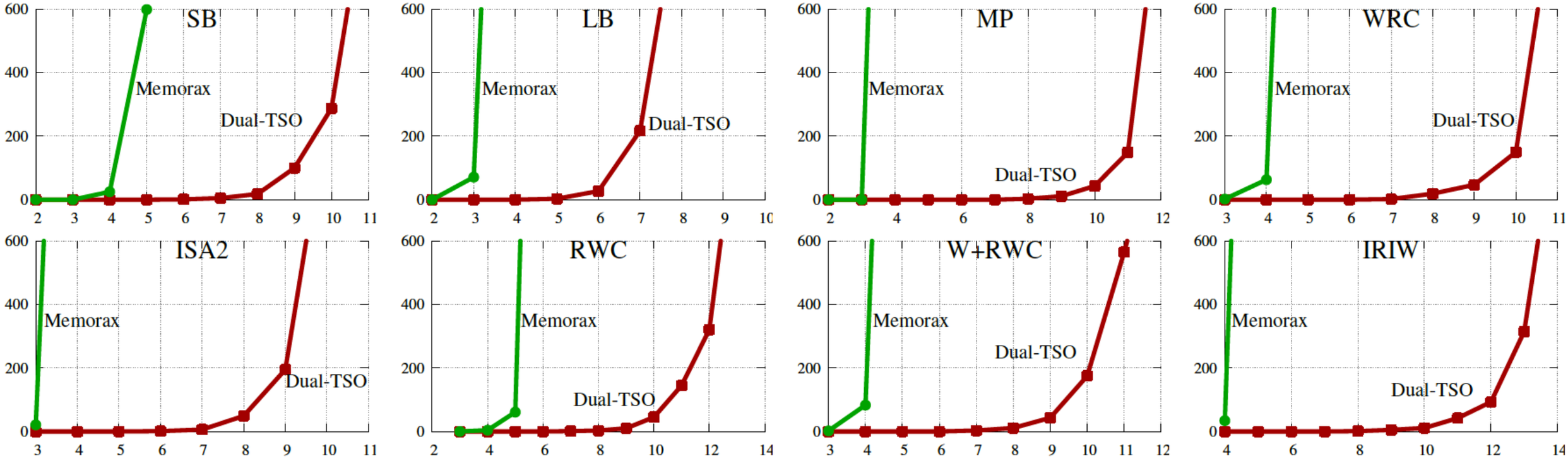
Experimental Results: Dual-TSO vs Memorax

Program	#P	Dual-TSO		Memorax	
		#T	#C	#T	#C
SB	5	0.3	10641	559.7	10515914
LB	3	0.0	2048	71.4	1499475
WRC	4	0.0	1507	63.3	1398393
ISA2	3	0.0	509	21.1	226519
RWC	5	0.1	4277	61.5	1196988
W+RWC	4	0.0	1713	83.6	1389009
IRIW	4	0.0	520	34.4	358057
Nbw_w_wr	2	0.0	222	10.7	200844
Sense_rev_bar	2	0.1	1704	0.8	20577
Dekker	2	0.1	5053	1.1	19788
Dekker_simple	2	0.0	98	0.0	595
Peterson	2	0.1	5442	5.2	90301
Peterson_loop	2	0.2	7632	5.6	100082
Szymanski	2	0.6	29018	1.0	26003
MP	4	0.0	883	TO	•
Ticket_spin_lock	3	0.9	18963	TO	•
Bakery	2	2.6	82050	TO	•
Dijkstra	2	0.2	8324	TO	•
Lamport_fast	3	17.7	292543	TO	•
Burns	4	124.3	2762578	TO	•

Experimental Results: Parameterised Case

Program	Dual-TSO	
	#T	#C
SB	0.0	147
LB	0.6	1028
MP	0.0	149
WRC	0.8	618
ISA2	4.3	1539
RWC	0.2	293
W+RWC	1.5	828
IRIW	4.6	648

Scalability: D-TSO vs Memorex



Conclusion

- Verification under WMM's is **hard**
- **Decidability for** (relatively strong) models such as **TSO**
- High complexity, but **practical approaches are possible**
- **Duality** —> **simple, general, and efficient** decision procedure

Conclusion

- Verification under WMM's is **hard**
- **Decidability for** (relatively strong) models such as **TSO**
- High complexity, but **practical approaches are possible**
- **Duality** —> **simple, general, and efficient** decision procedure

- Extension to other models ?
- Hardware/Programming Languages models ?

Conclusion

- Verification under WMM's is **hard**
- **Decidability for** (relatively strong) models such as **TSO**
- High complexity, but **practical approaches are possible**
- **Duality** —> **simple, general, and efficient** decision procedure

- Extension to other models ?
- Hardware/Programming Languages models ?
- Related to Consistency Criteria in concurrent/distributed syst.

Conclusion

- Verification under WMM's is **hard**
- **Decidability for** (relatively strong) models such as **TSO**
- High complexity, but **practical approaches are possible**
- **Duality** —> **simple, general, and efficient** decision procedure

- Extension to other models ?
- Hardware/Programming Languages models ?
- Related to Consistency Criteria in concurrent/distributed syst.
- Undecidability for more complex models (RMO, Power)
- Under/upper-approximate analyses are needed

E.g., context-bounded analysis for TSO

[Atig, B., Parlato, CAV 2011]

context-bounded analysis for Power

[Abdulla, Atig, B., Ngo, TACAS 2017]