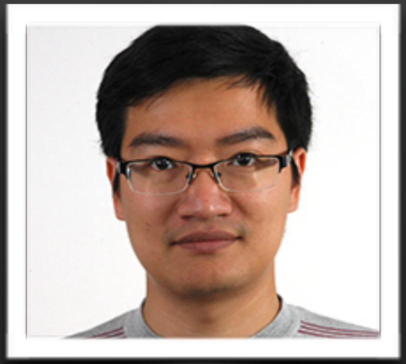
The Benefits of Duality in Verifying Concurrent Programs under TSO

Parosh Aziz Abdulla[1]

Mohamed Faouzi Atig[1]

Ahmed Bouajjani[2]



Tuan Phong Ngo[1]

[1]Uppsala University

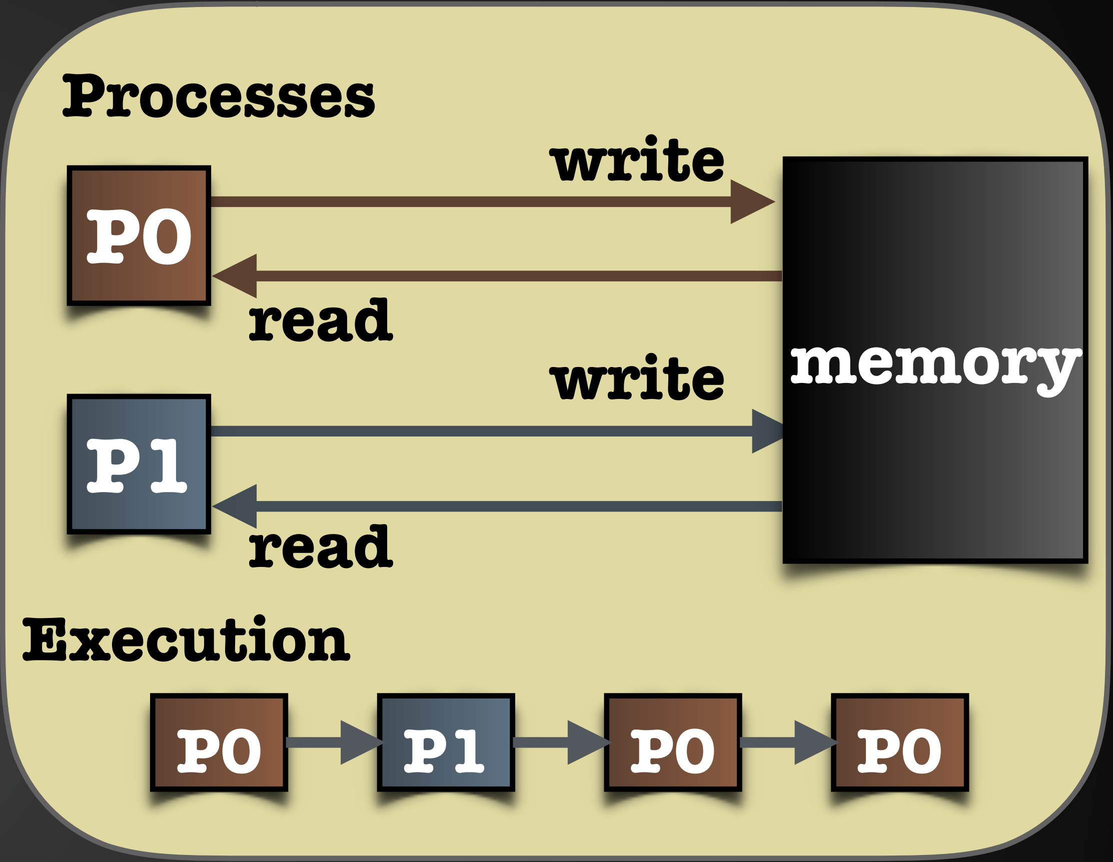
[2]IRIF, Université Paris Diderot & IUF

CONCUR 2016

Motivation

Sequential Consistency

- Processes (atomically) write to/read from shared memory
- Program order is persevered for each process
- Interleaving of the operations



Characteristics

😀 Simple and intuitive model

😞 Disallows many hardware/compiler optimizations

Weak Memory Models

Hardware Optimizations

- **Processors execute instructions out-of-order:**
  - 😀 **Better performance and energy**
  - 😞 **Non-intuitive behaviors: bugs**

Weak memory model: captures the semantics of out-of-order execution

Goal

- **Efficient verification technique for checking safety properties**

Outline

- Classical TSO (Total Store Order) semantics
- New semantics (Single-Buffer) allows:
  - applying well quasi-order framework
- New semantics (Dual-TSO) allows:
  - Efficient verification
  - Parameterized verification
- Verification under Dual-TSO
- Experimental Results
- Conclusions

TSO - Total Store Order

Widely Used

- Used by Sun SPARCv9
- Current formalization of Intel x86

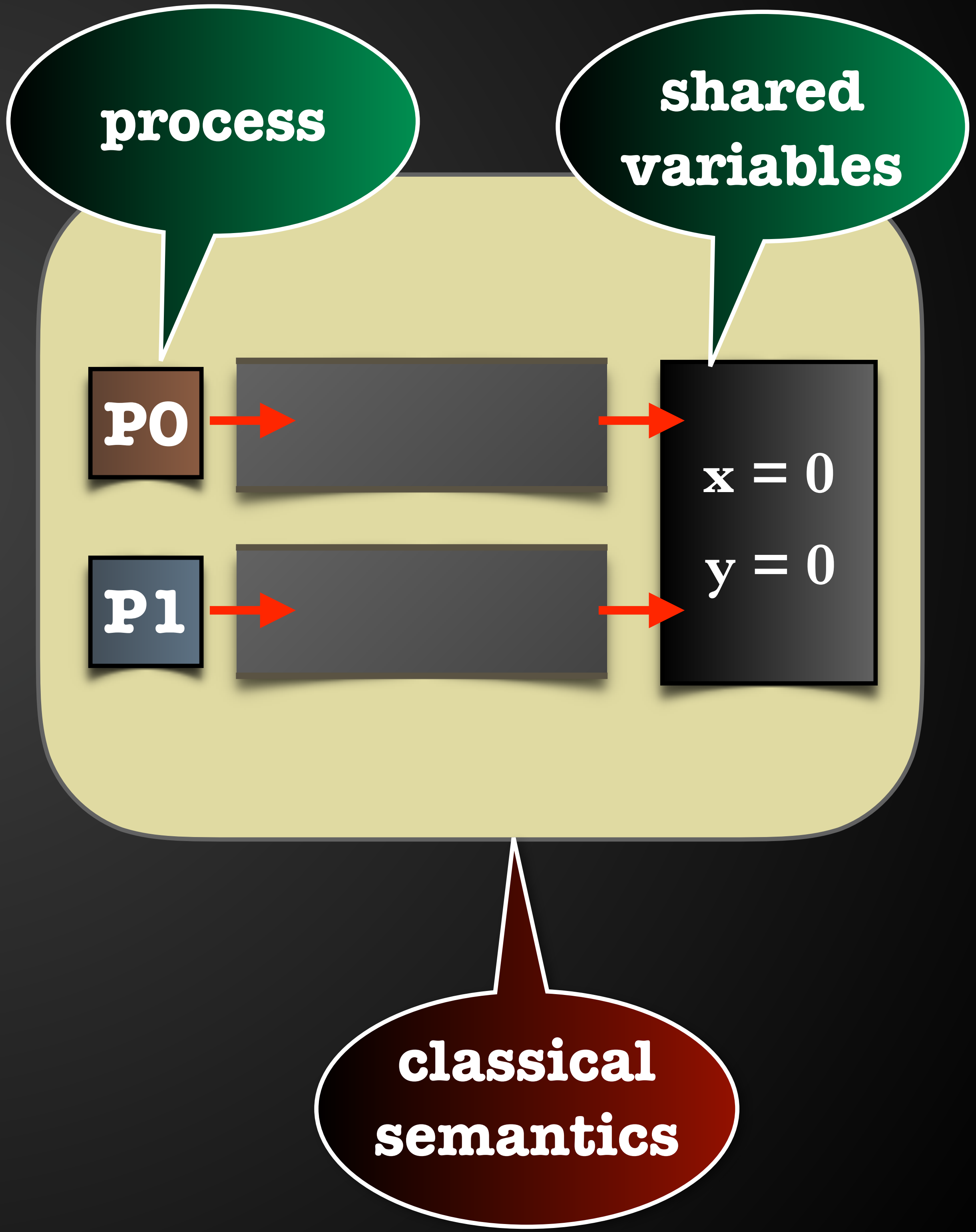
TSO - Total Store Order

Widely Used

- Used by Sun SPARCv9
- Current formalisation of Intel x86

Optimize Memory Access

- Memory writes are slow
- Introduce (perfect) store buffers



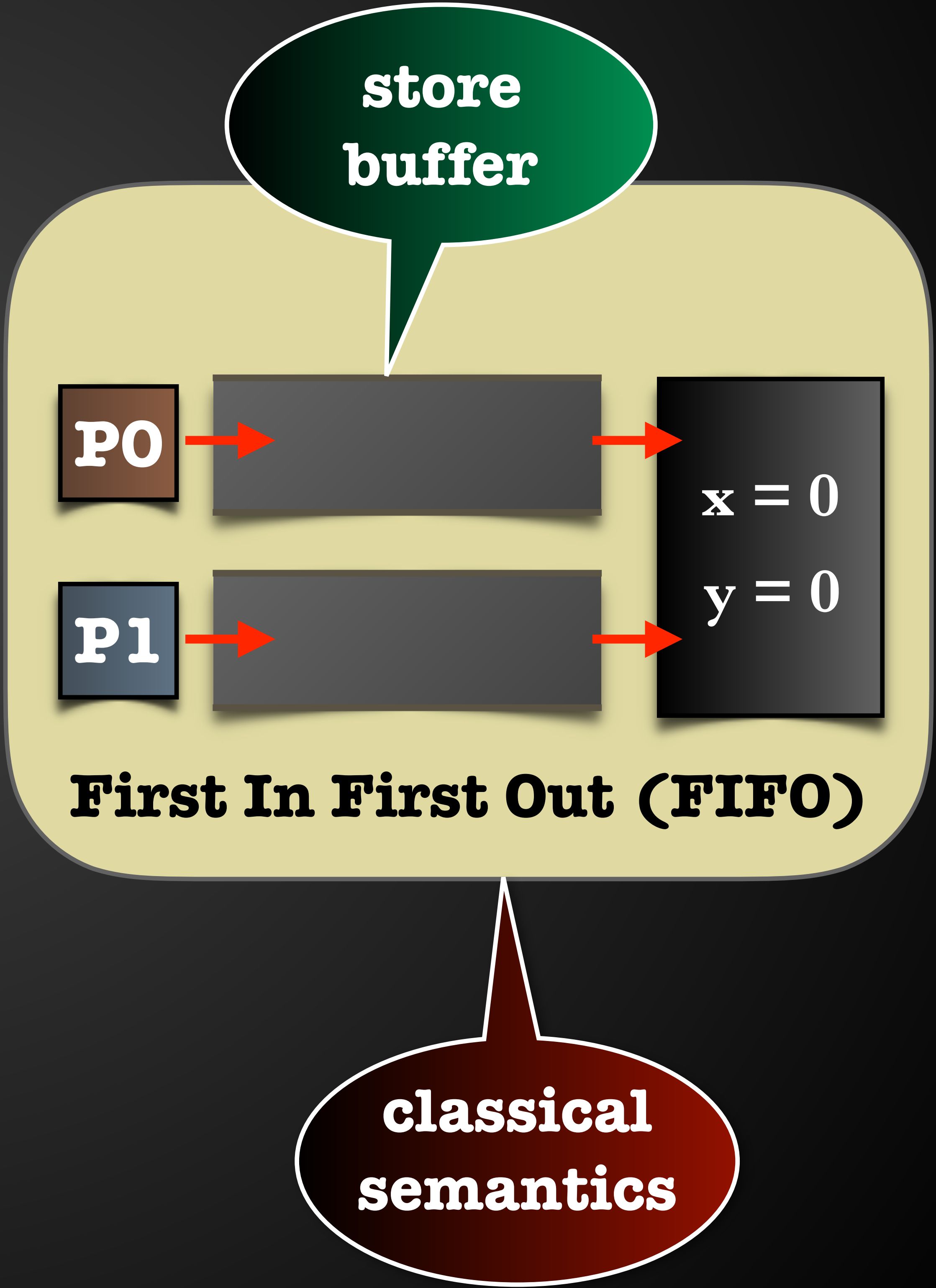
TSO - Total Store Order

Widely Used

- Used by Sun SPARCv9
- Current formalisation of Intel x86

Optimise Memory Access

- Memory writes are slow
- Introduce (perfect) store buffers



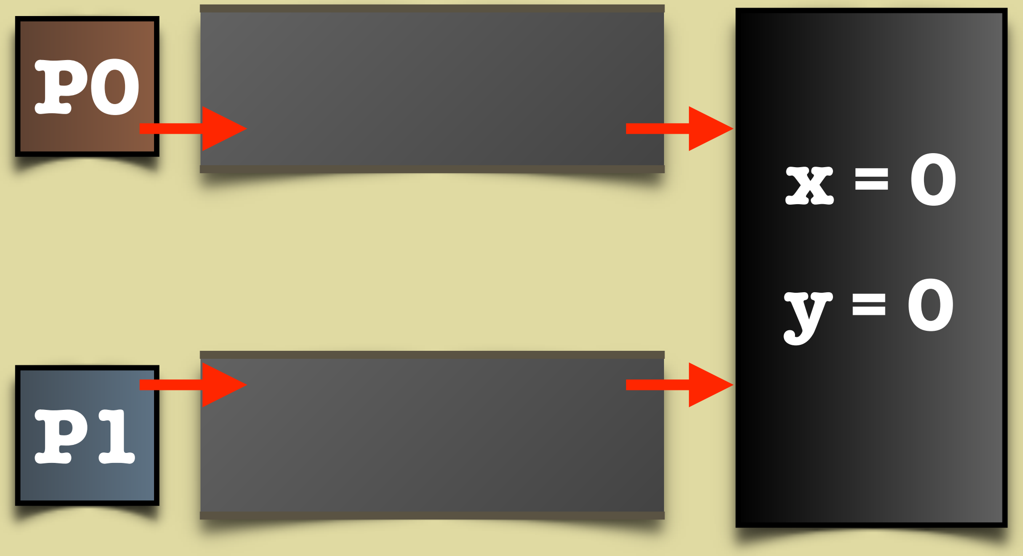
Classical TSO Semantics

P0: write: x = 1

P0: write: x = 2

P0: read: x = 2

P0: read: y = 0



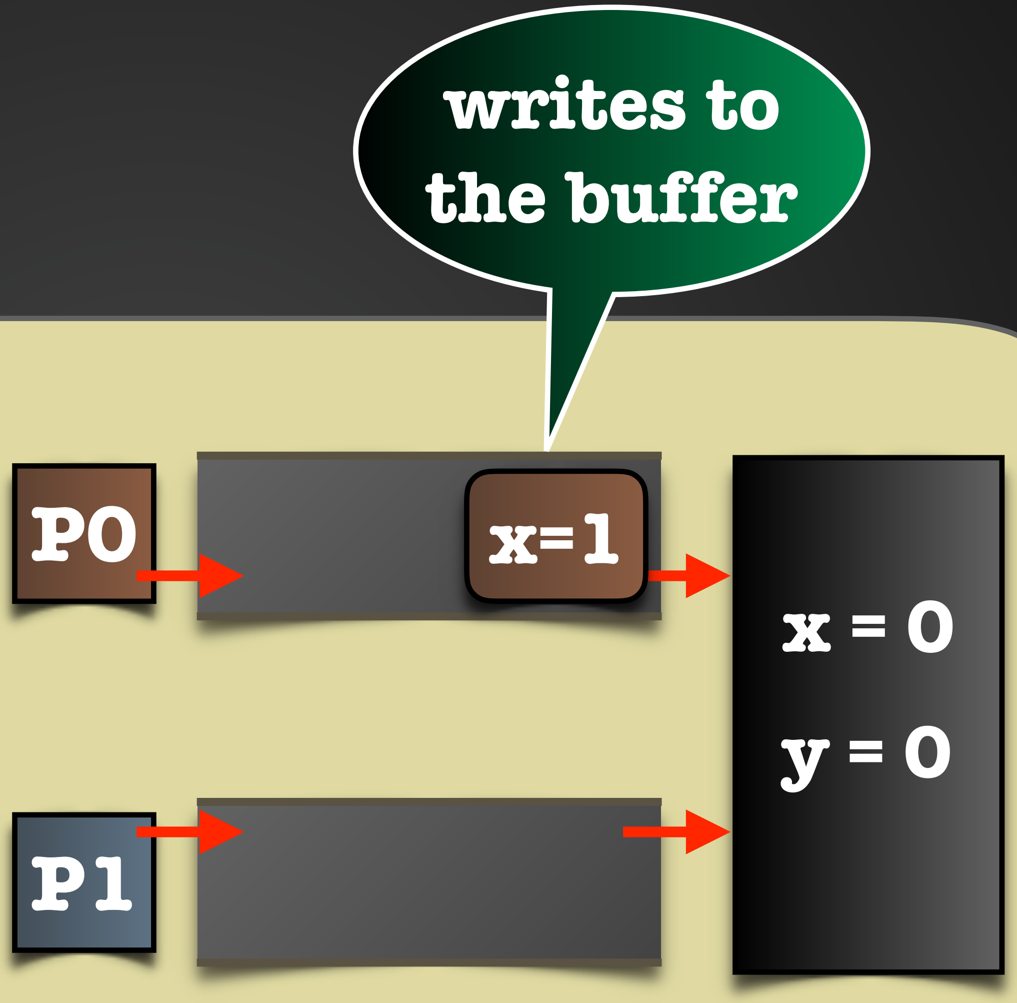
Classical TSO Semantics

P0: write: x = 1

P0: write: x = 2

P0: read: x = 2

P0: read: y = 0



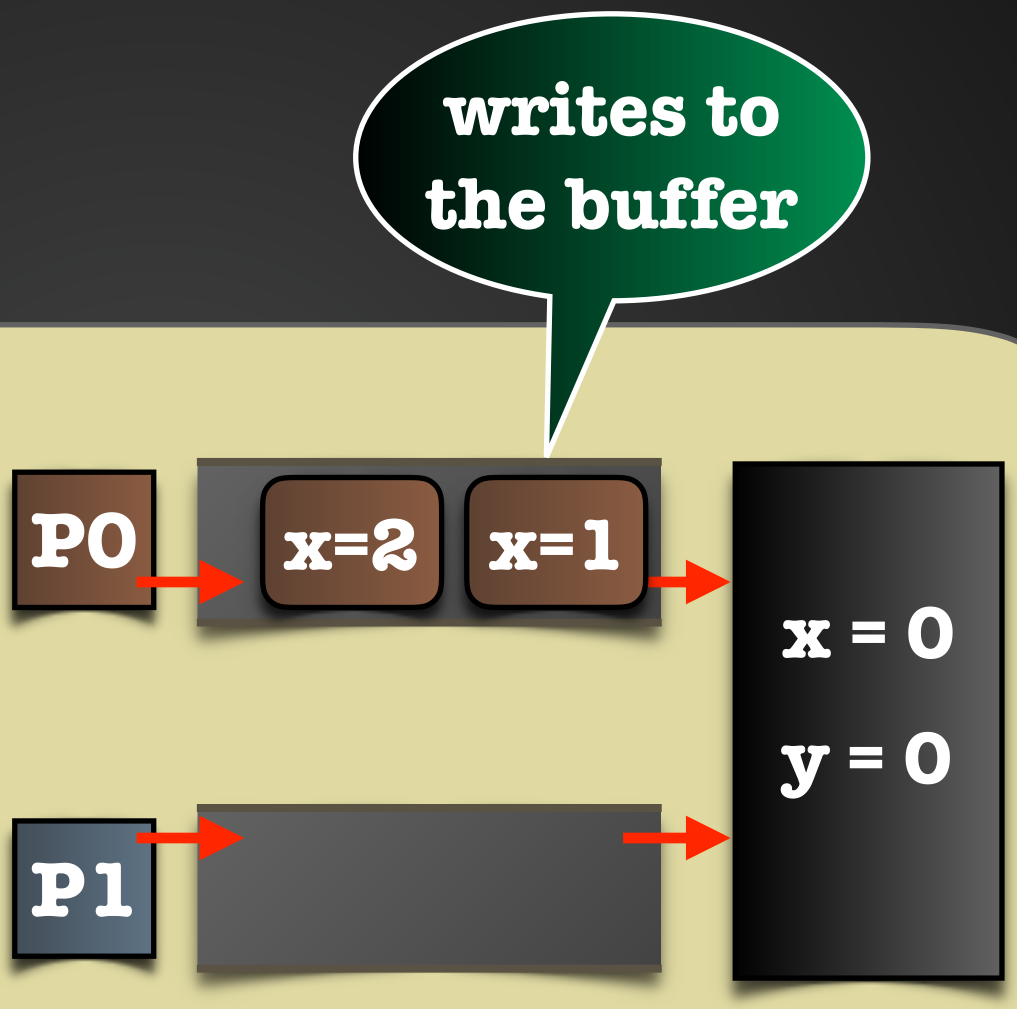
Classical TSO Semantics

P0: write: x = 1

P0: write: x = 2

P0: read: x = 2

P0: read: y = 0



writes to the buffer

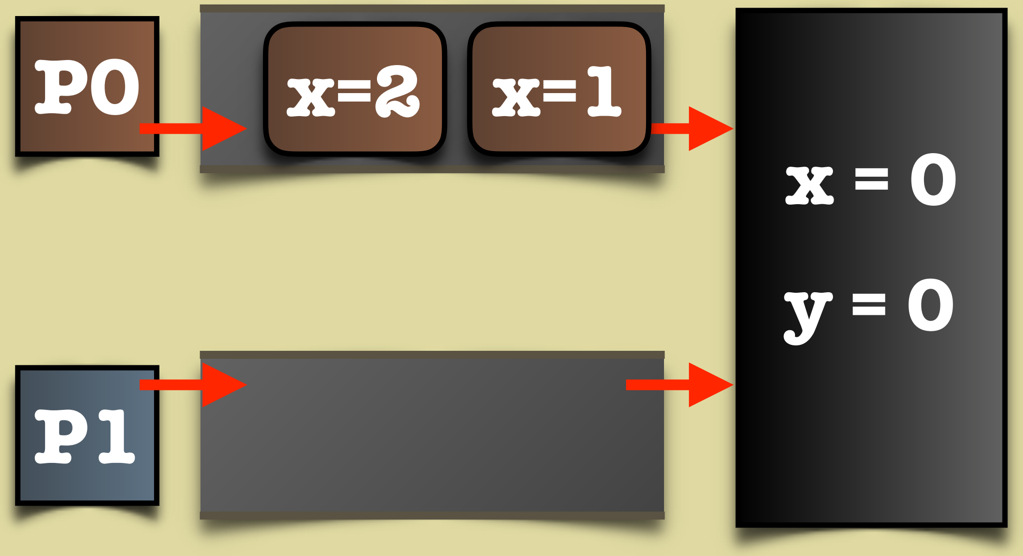
Classical TSO Semantics

P0: write: x = 1

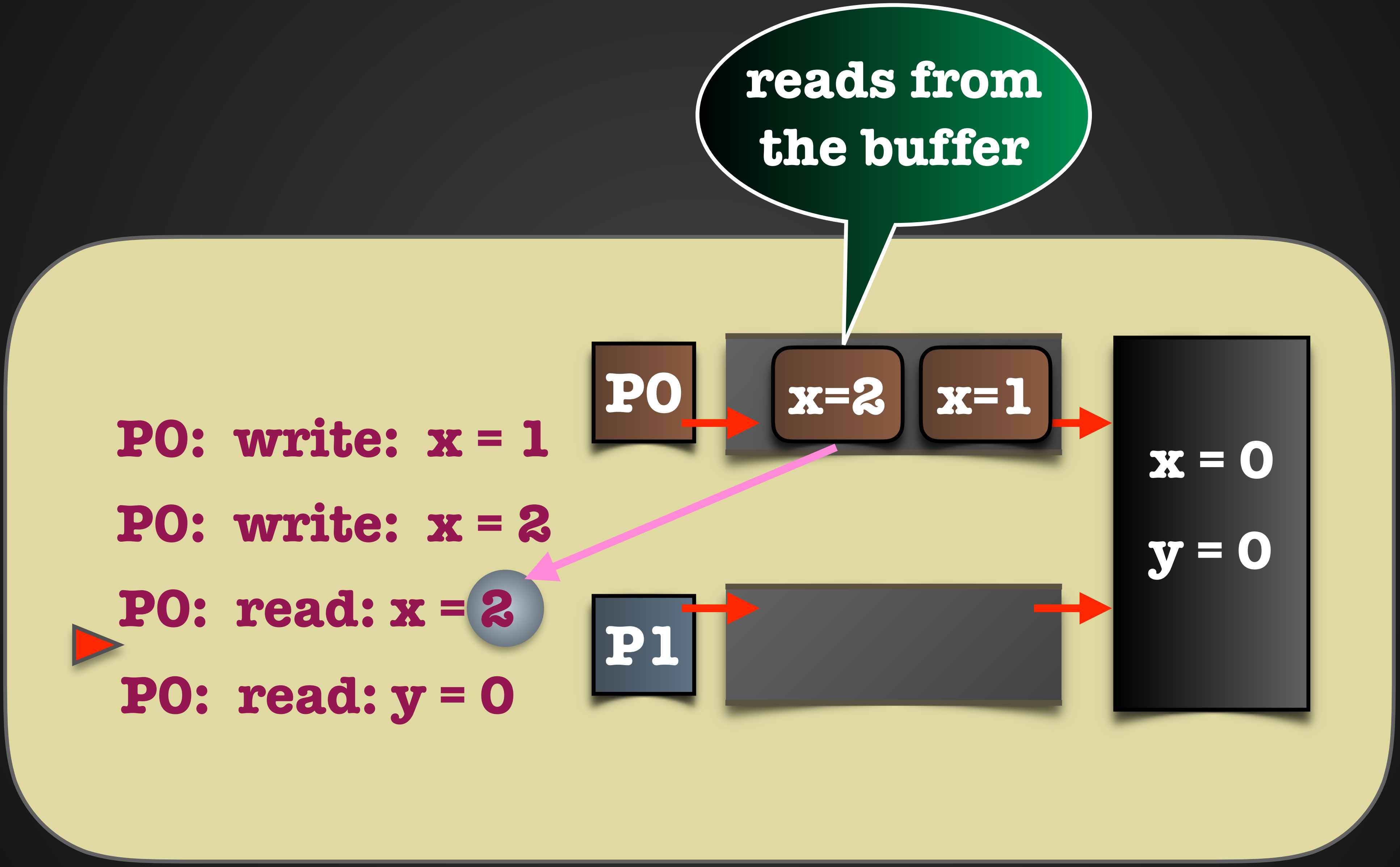
P0: write: x = 2

P0: read: x = 2

P0: read: y = 0



Classical TSO Semantics



reads from the buffer

P0: write: x = 1

P0: write: x = 2

P0: read: x = 2

P0: read: y = 0

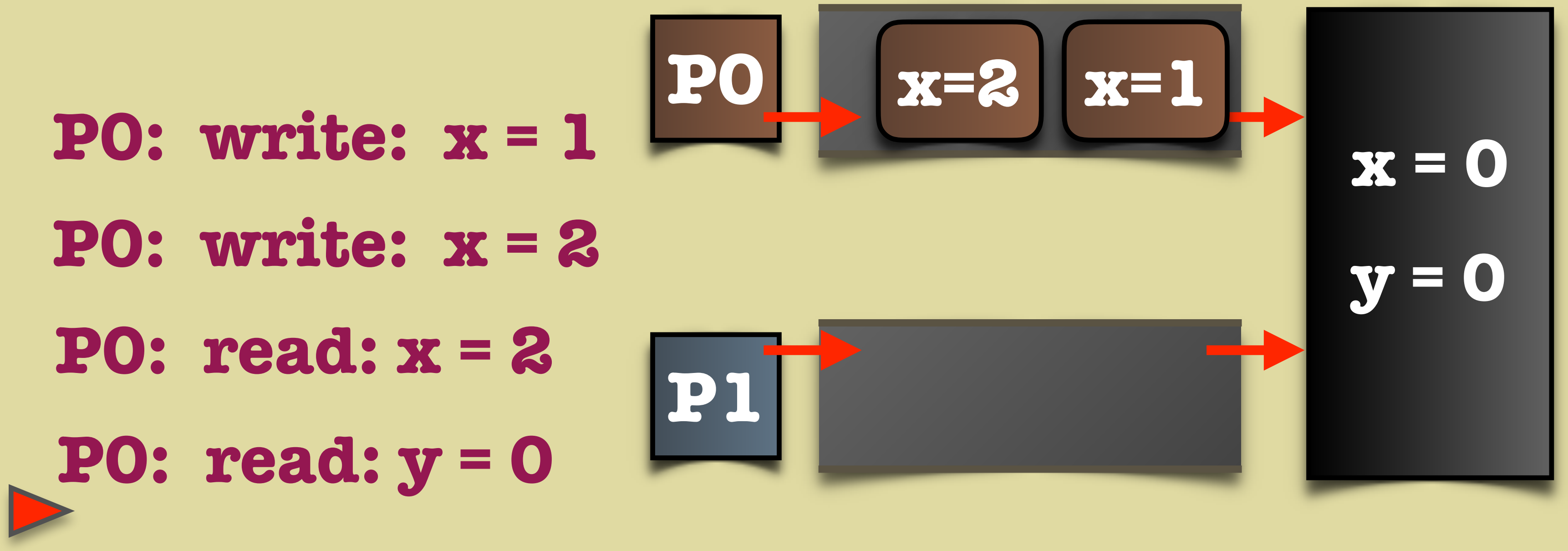
Classical TSO Semantics

P0: write: x = 1

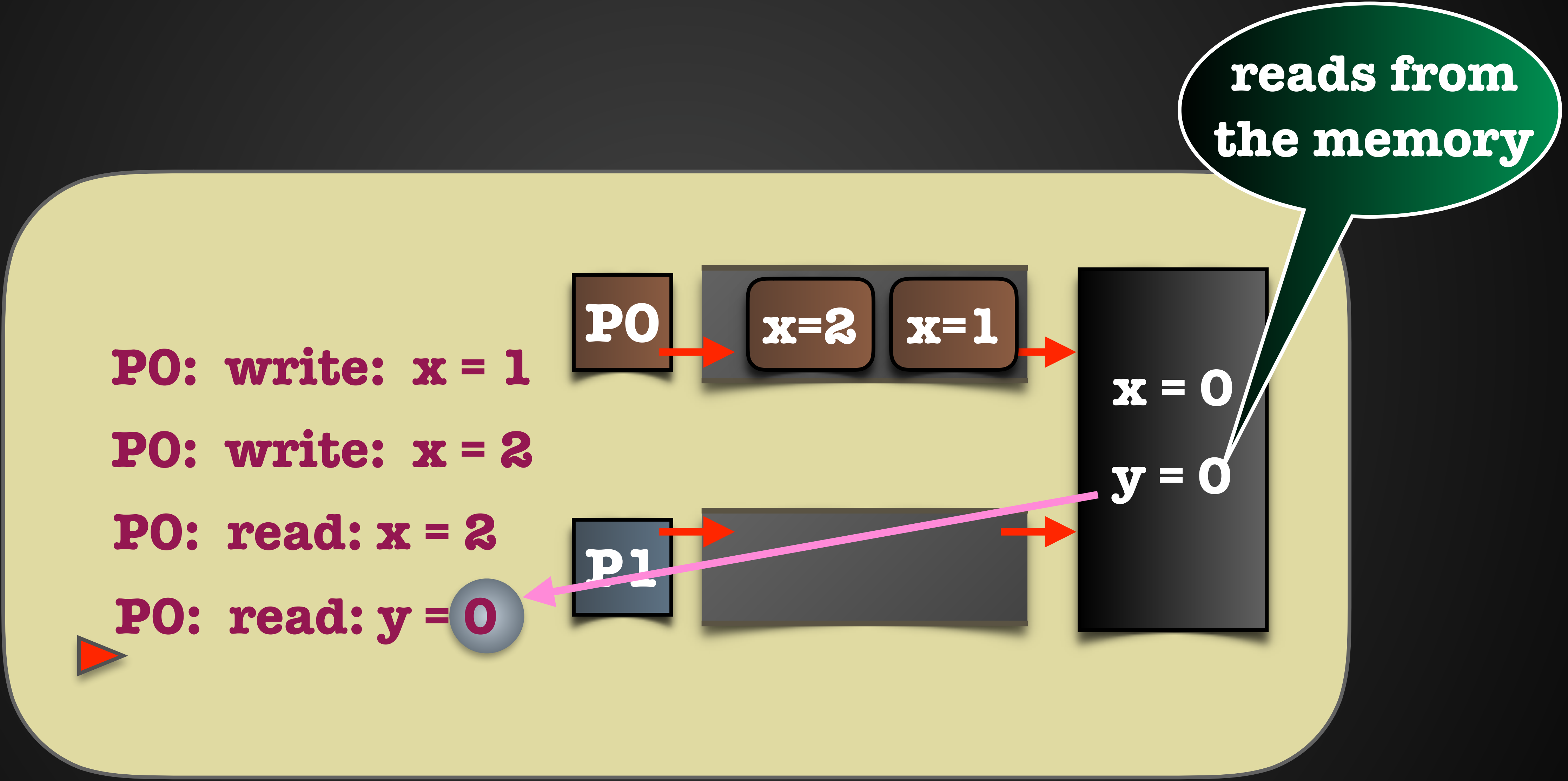
P0: write: x = 2

P0: read: x = 2

P0: read: y = 0



Classical TSO Semantics



P0: write: x = 1

P0: write: x = 2

P0: read: x = 2

P0: read: y = 0

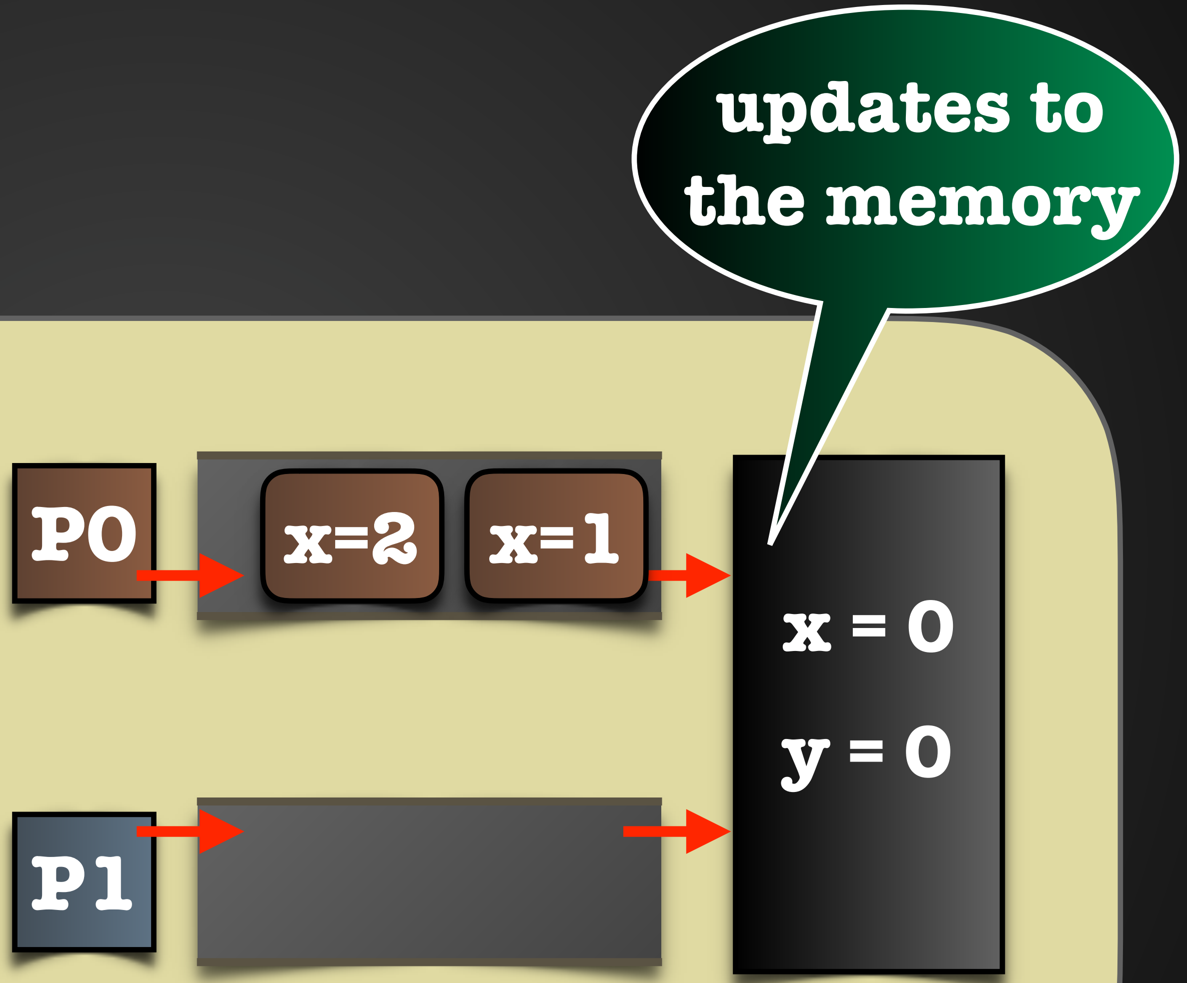
Classical TSO Semantics

PO: write: x = 1

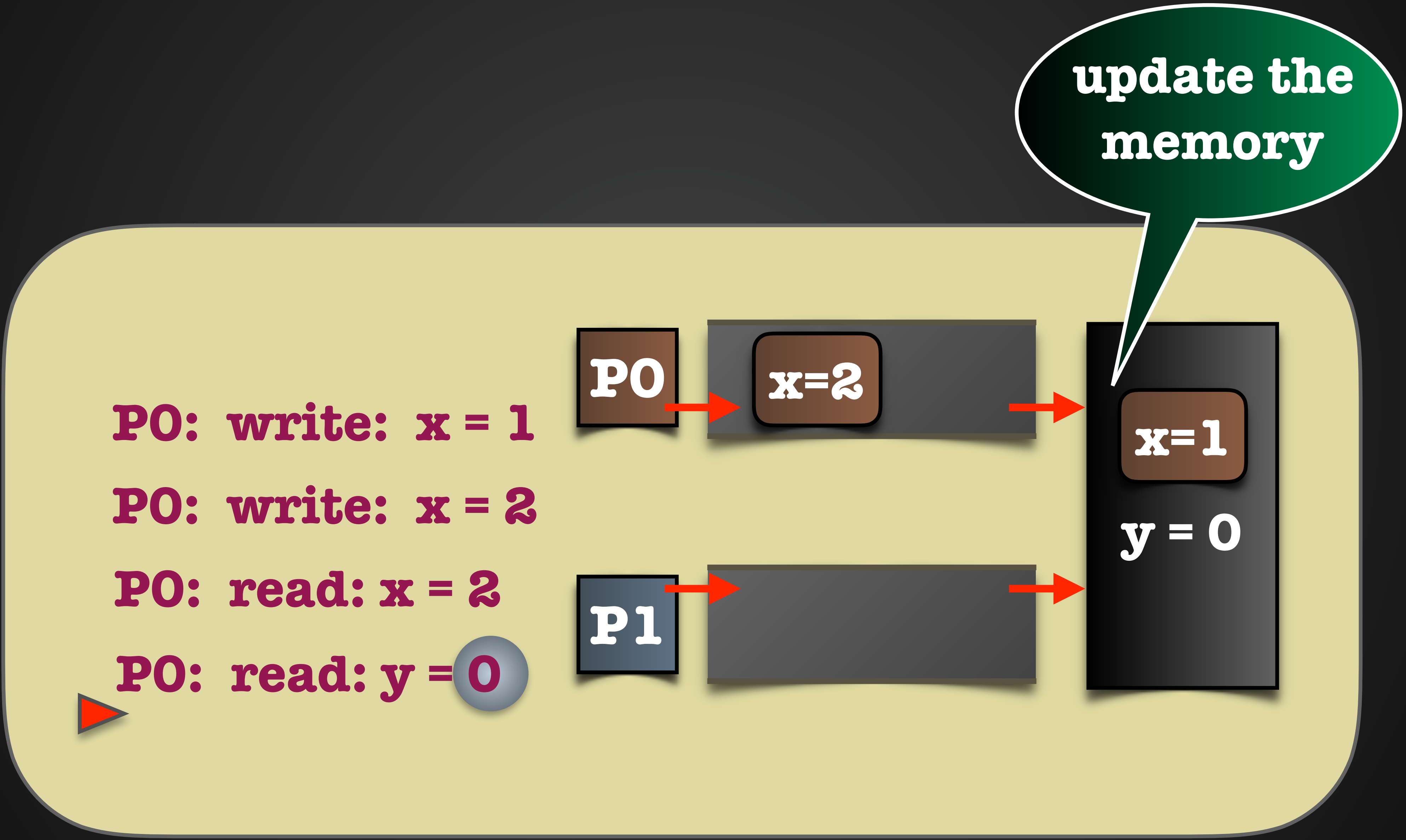
PO: write: x = 2

PO: read: x = 2

PO: read: y = 0



Classical TSO Semantics



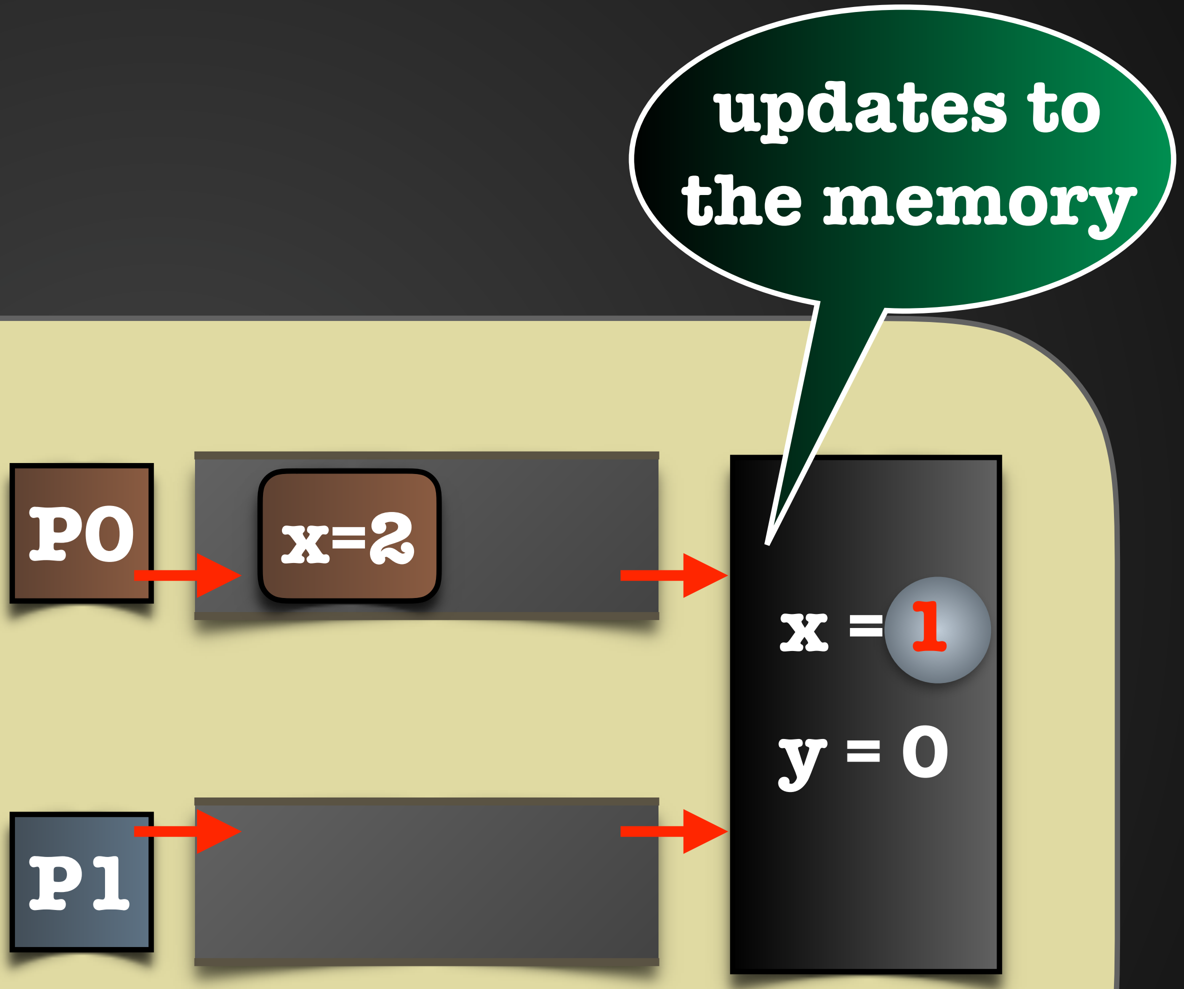
Classical TSO Semantics

P0: write: x = 1

P0: write: x = 2

P0: read: x = 2

P0: read: y = 0



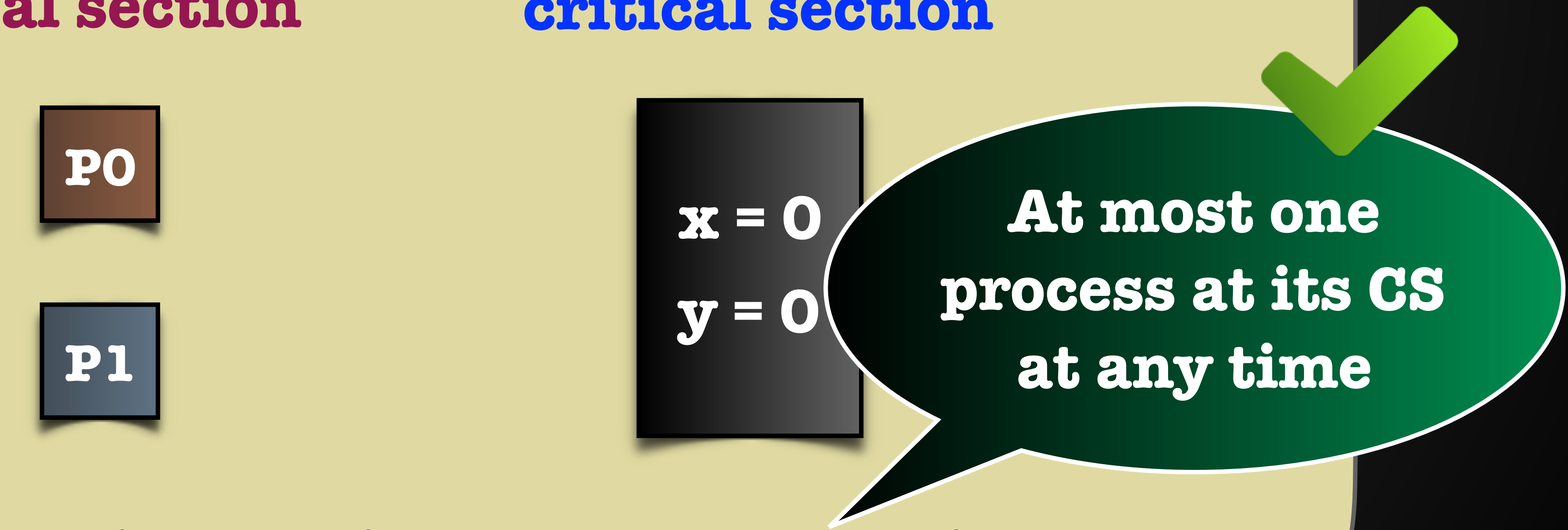
updates to the memory

Potentially Bad Behaviors - Dekker

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	**P1**
write: x = 1	**write: y = 1**
read: y = 0	**read: x = 0**
critical section	**critical section**



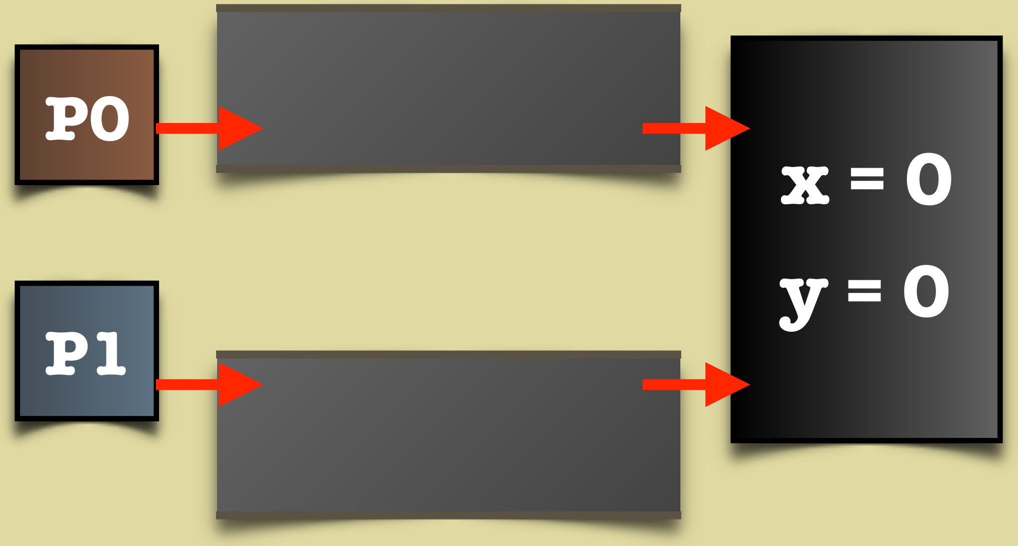
At most one process at its CS at any time

Sequential Consistency = Interleaving

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	**P1**
write: x = 1	**write: y = 1**
read: y = 0	**read: x = 0**
critical section	**critical section**

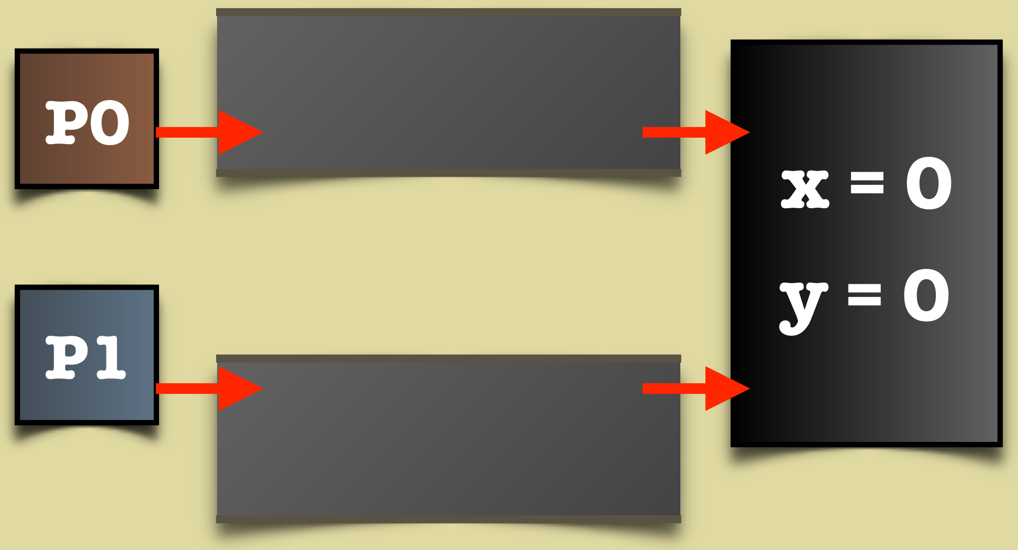


TSO

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	**P1**
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section



P0

P1

x = 0

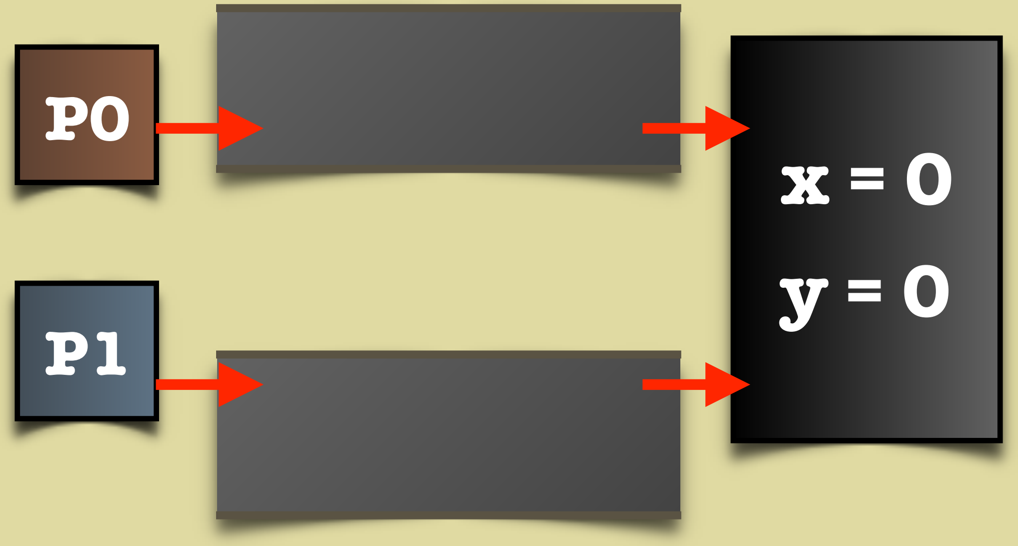
y = 0

TSO

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	P1
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section

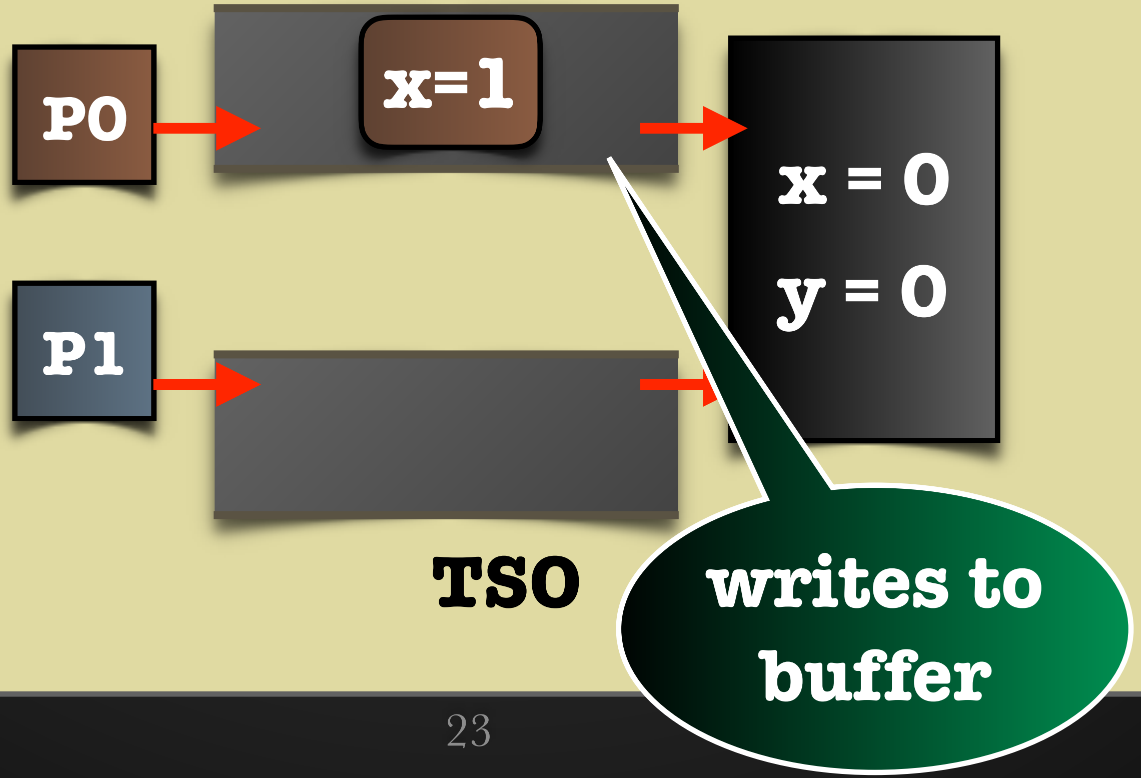


TSO

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

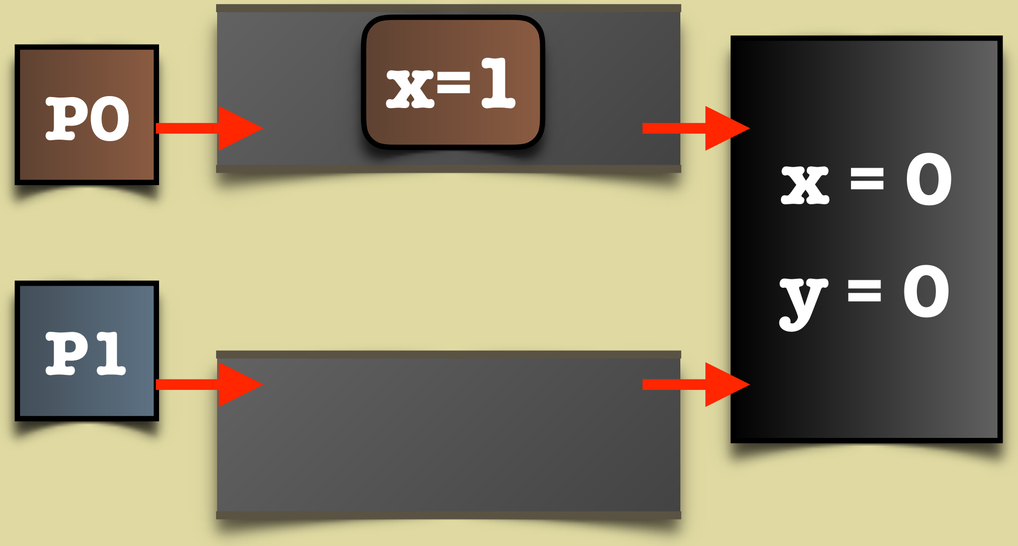
P0	**P1**
write: x = 1	**write: y = 1**
read: y = 0	**read: x = 0**
critical section	**critical section**



Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	**P1**
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section

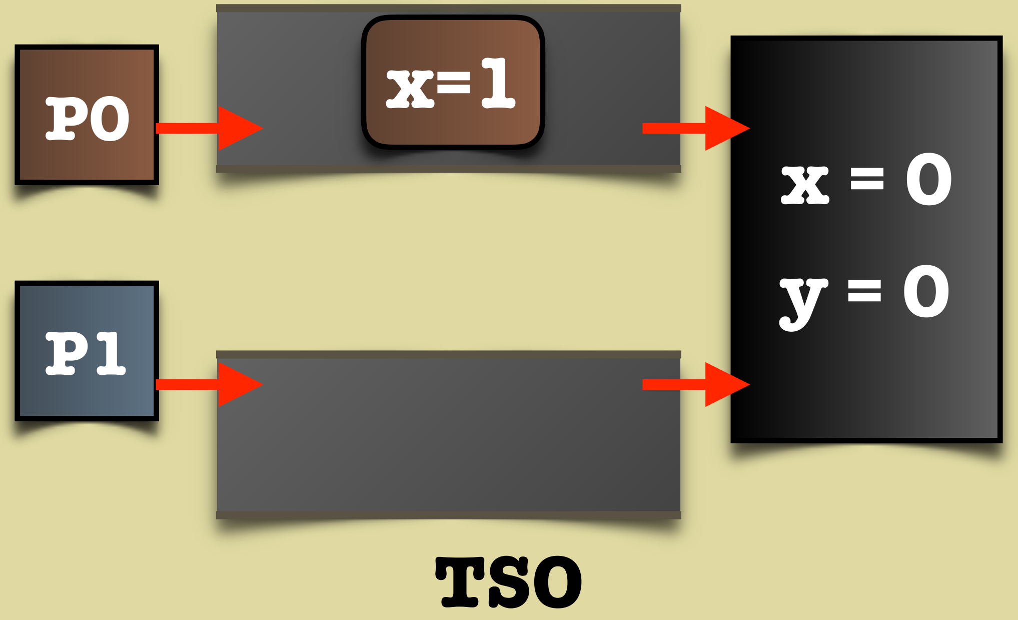


TSO

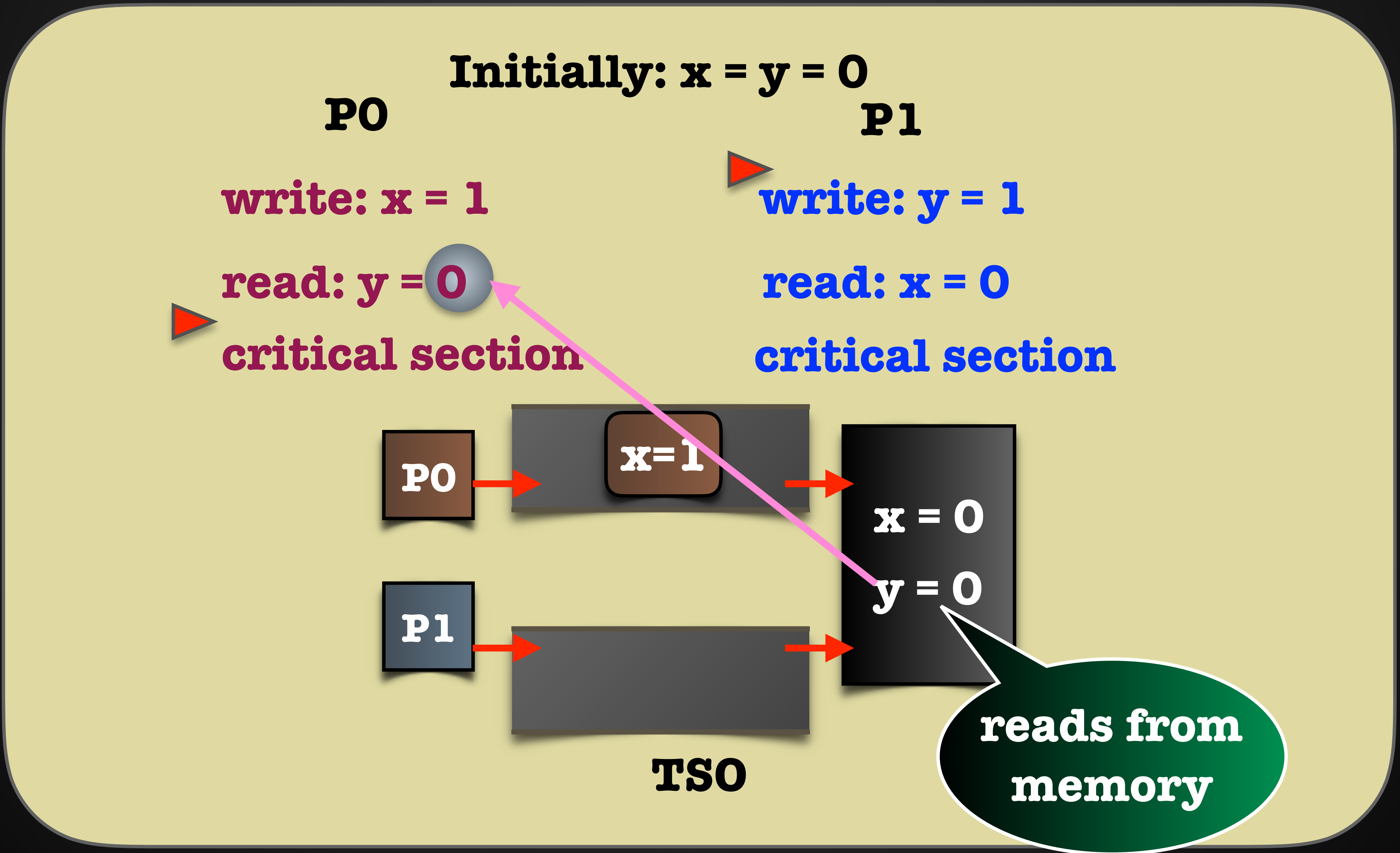
Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	P1
write: x = 1	▶ write: y = 1
read: y = 0	read: x = 0
▶ critical section	critical section



Potentially Bad Behaviours - Dekker



Initially: x = y = 0

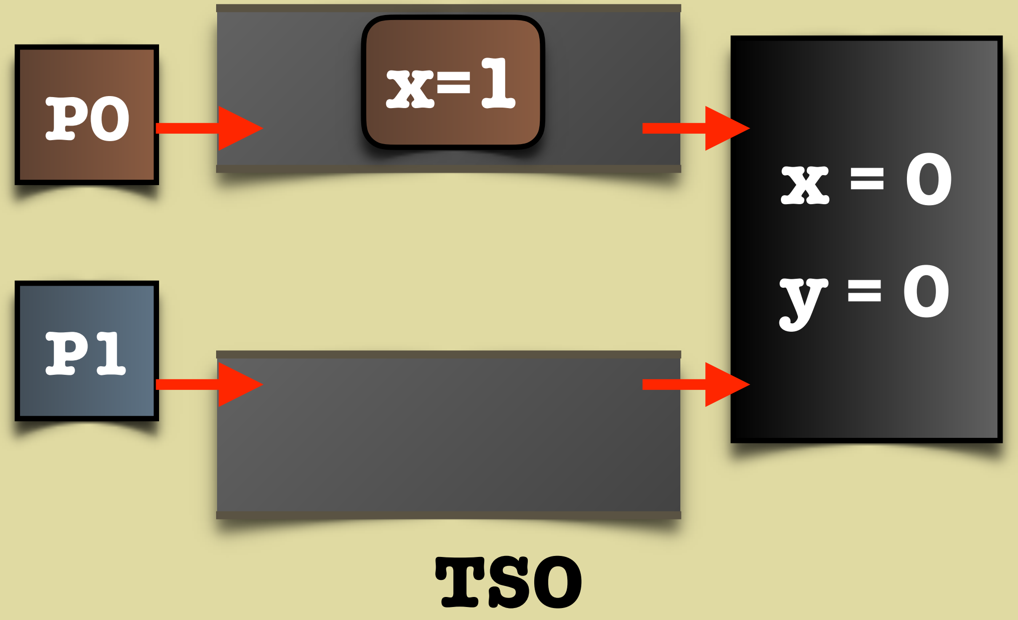
P0	P1
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	**P1**
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section

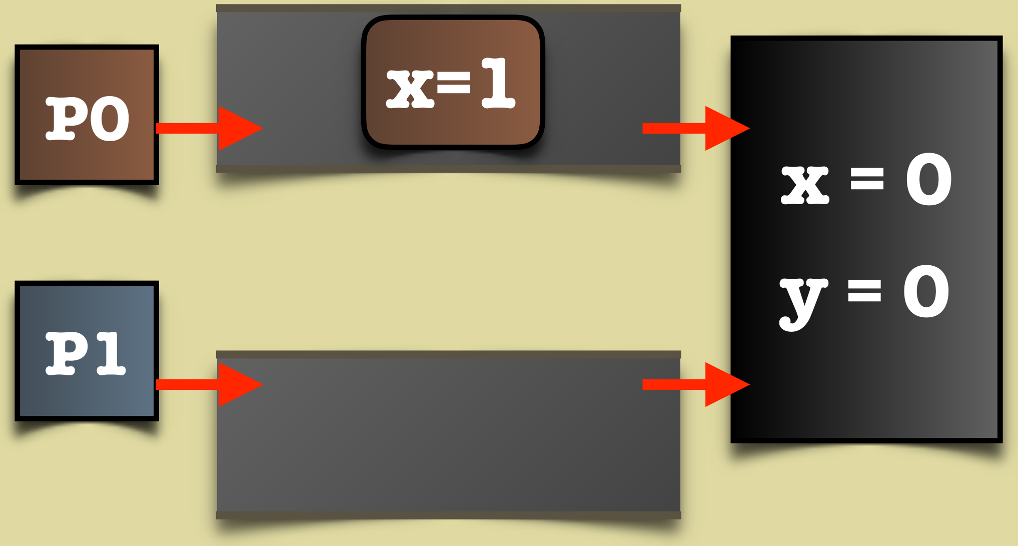
enters CS



Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	**P1**
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section



TSO

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0

write: x = 1

read: y = 0

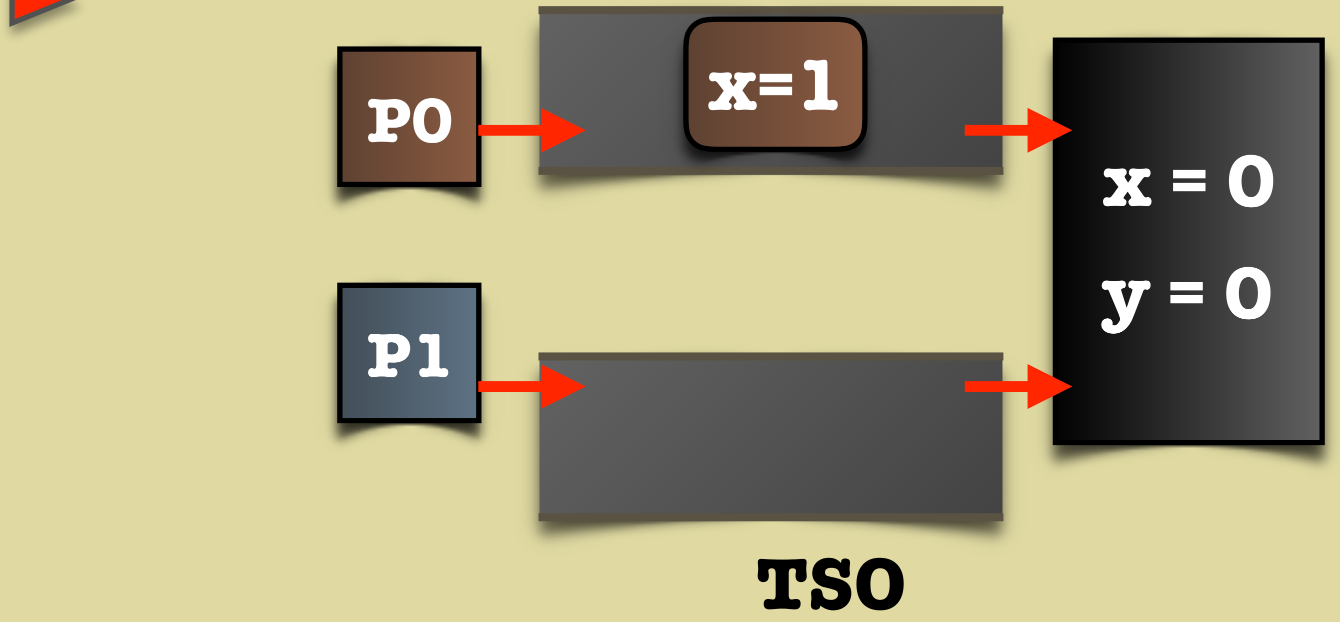
critical section

P1

write: y = 1

read: x = 0

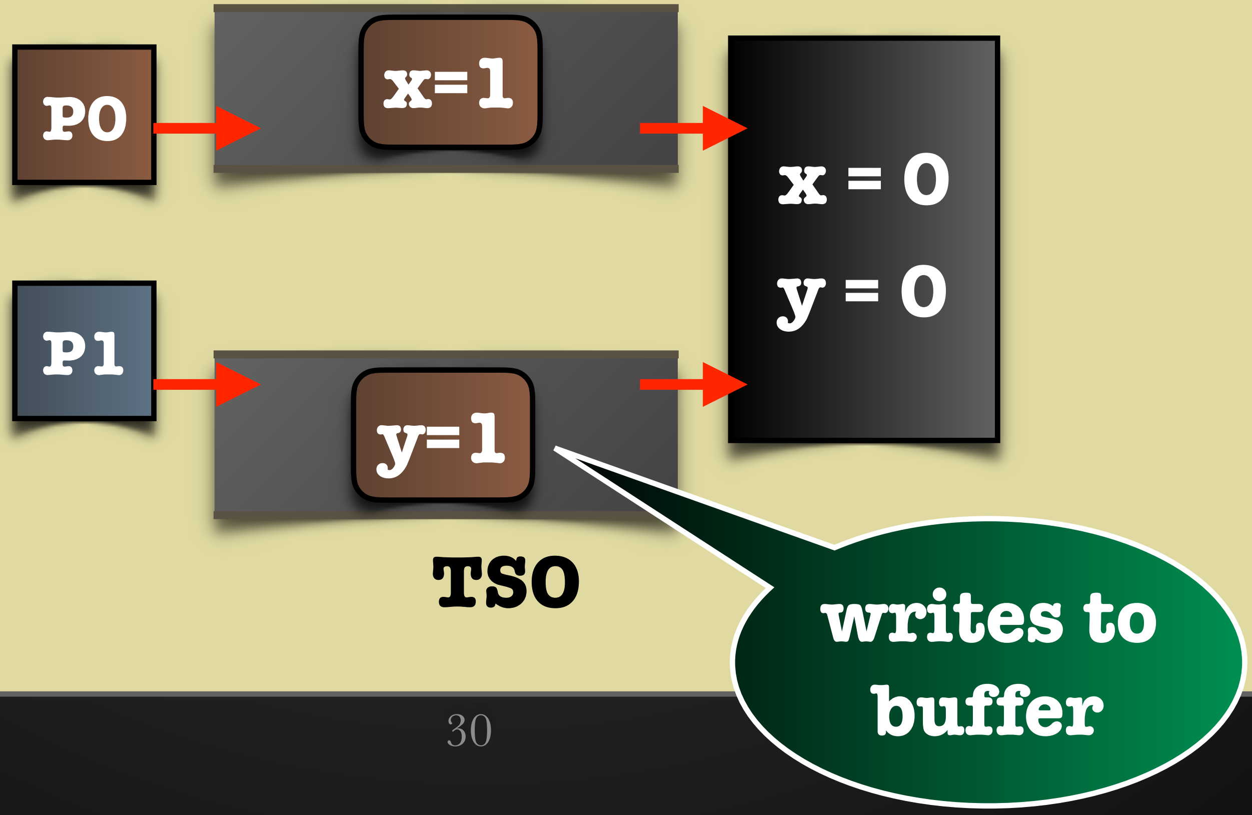
critical section



Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	**P1**
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section

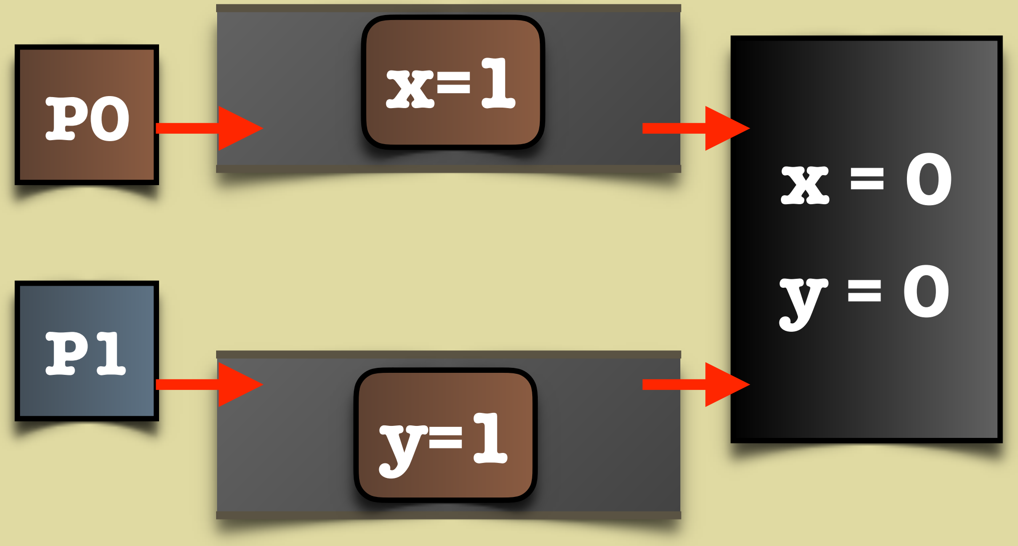


writes to buffer

Potentially Bad Behaviours - Dekker

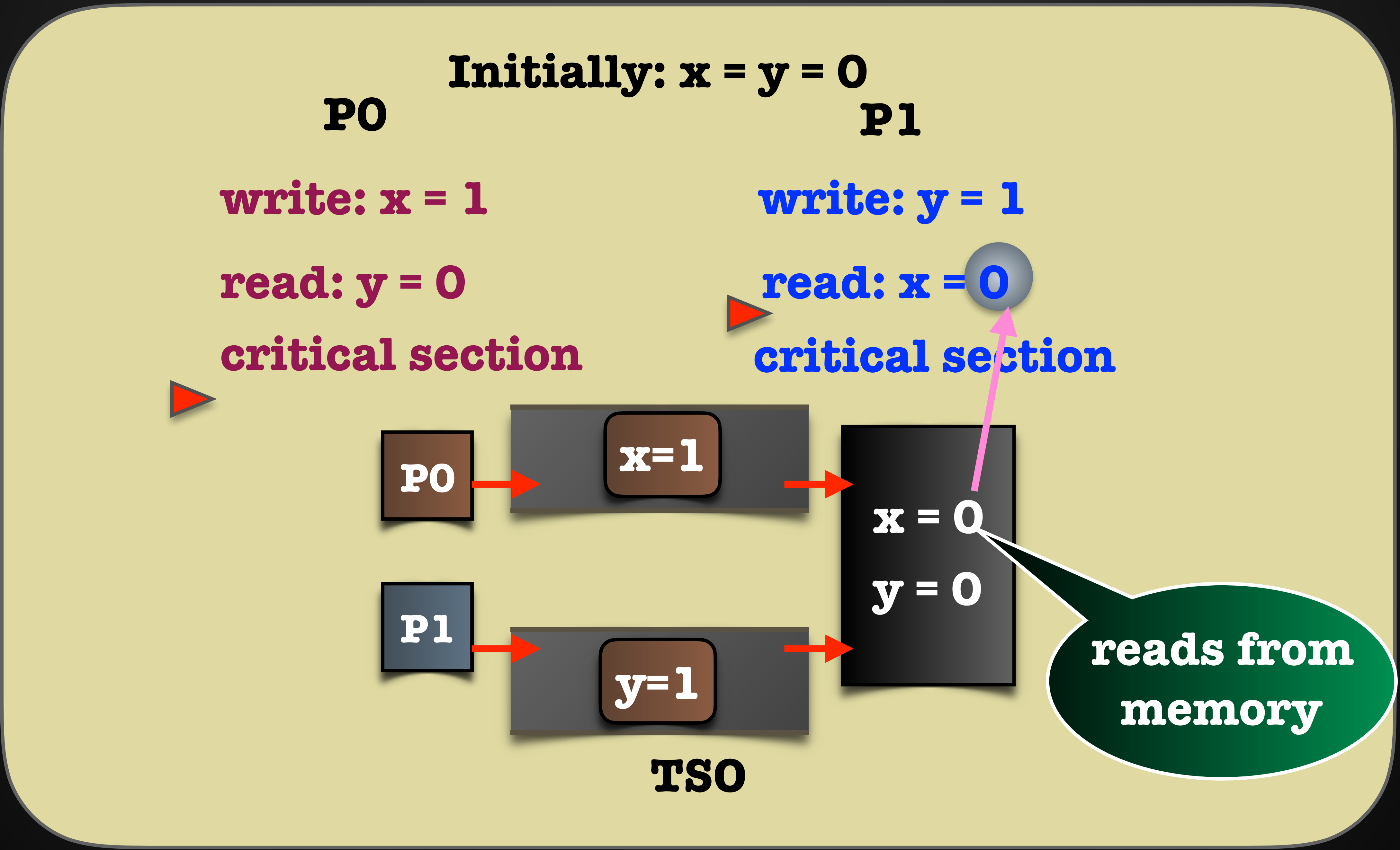
Initially: x = y = 0

P0	**P1**
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section



TSO

Potentially Bad Behaviours - Dekker



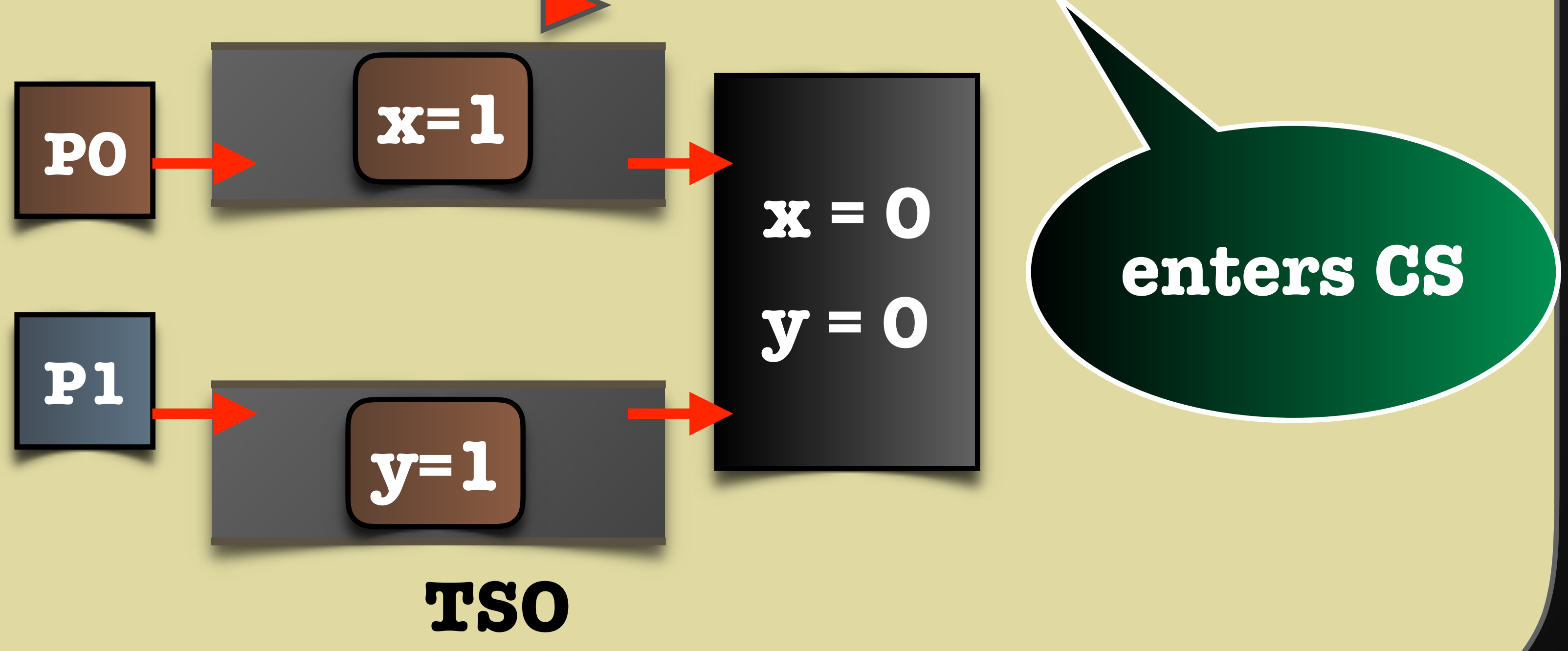
Initially: x = y = 0

P0	P1
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	**P1**
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section

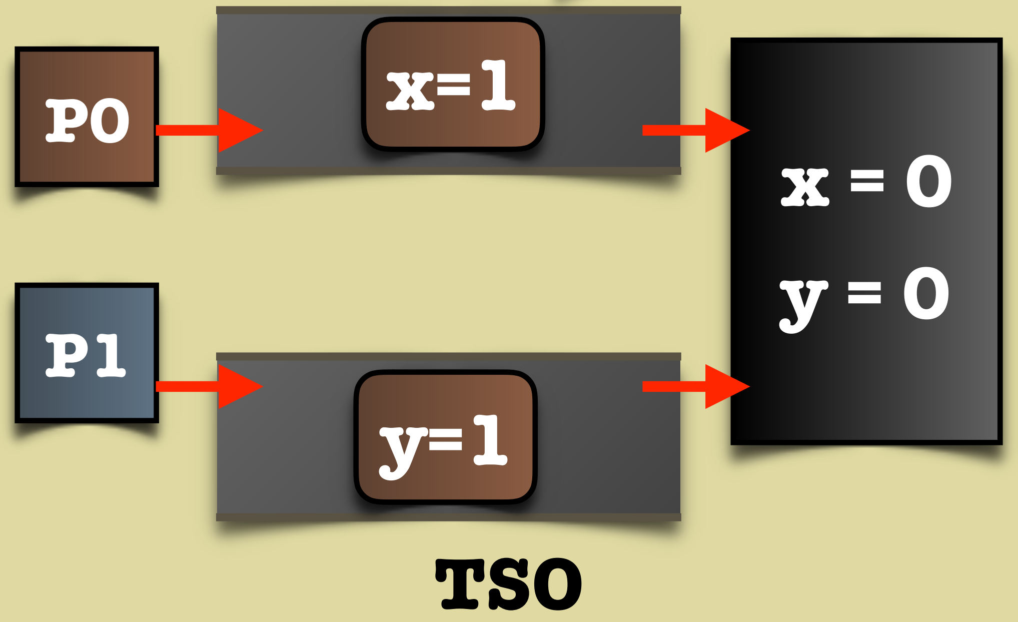


Potentially Bad Behaviours - Dekker

2 processes in CS at the same time

Initially: x = y = 0

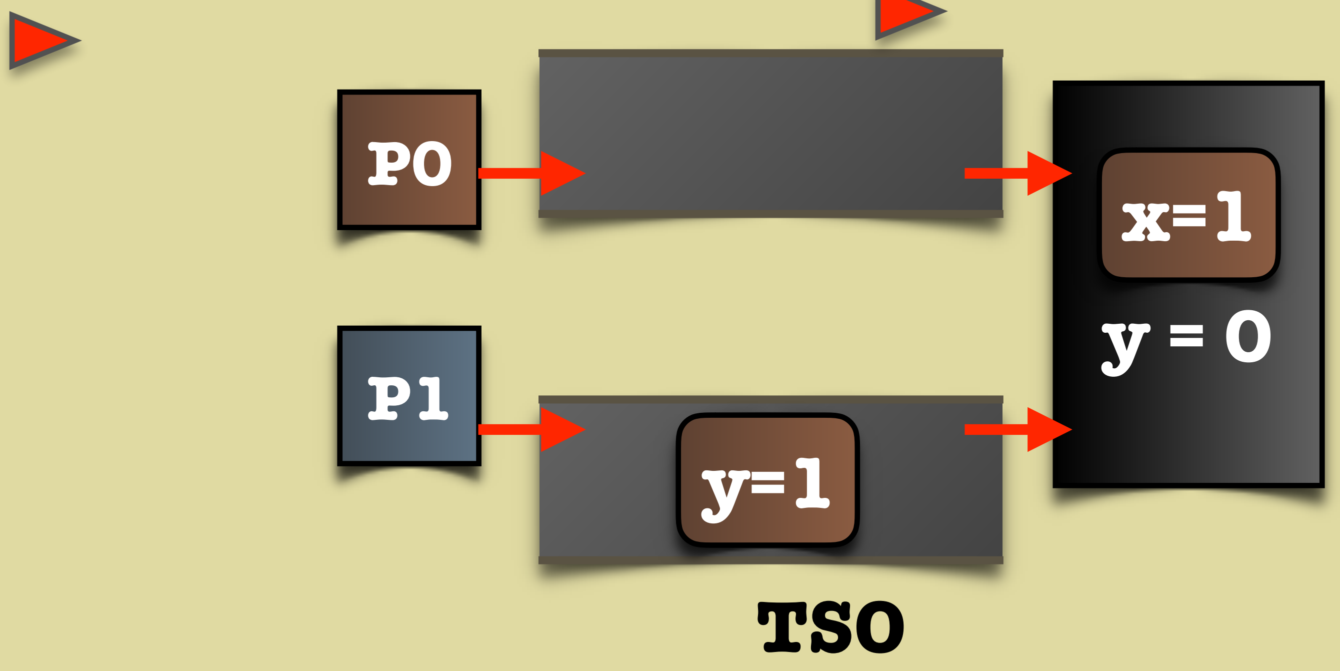
P0	**P1**
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section



Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	**P1**
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section



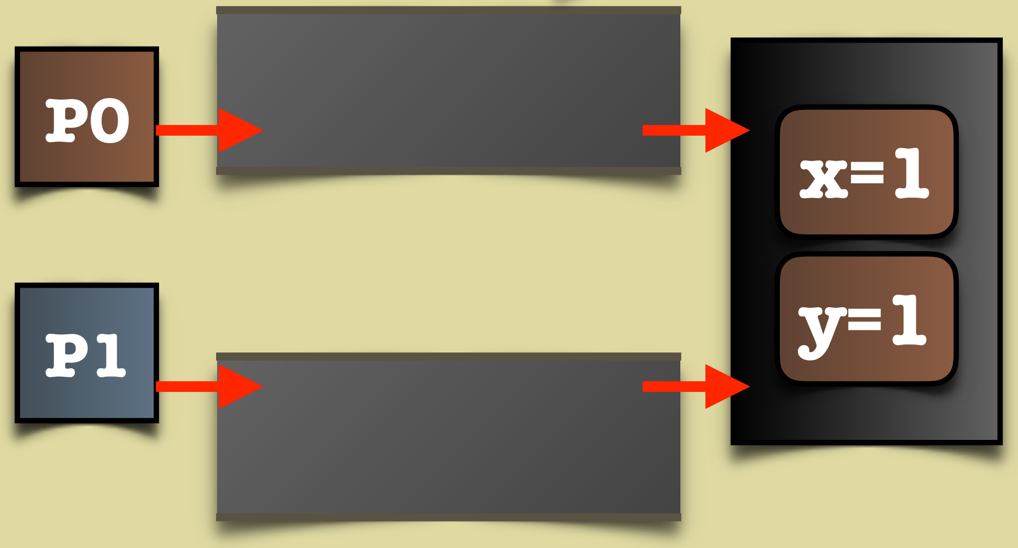
Potentially Bad Behaviours - Dekker

“read overtaking write”

“read overtaking write”

Initially: x = y = 0

P0	P1
write: x = 1	write: y = 1
read: y = 0	read: x = 0
critical section	critical section



TSO

Potentially Bad Behaviours - Dekker

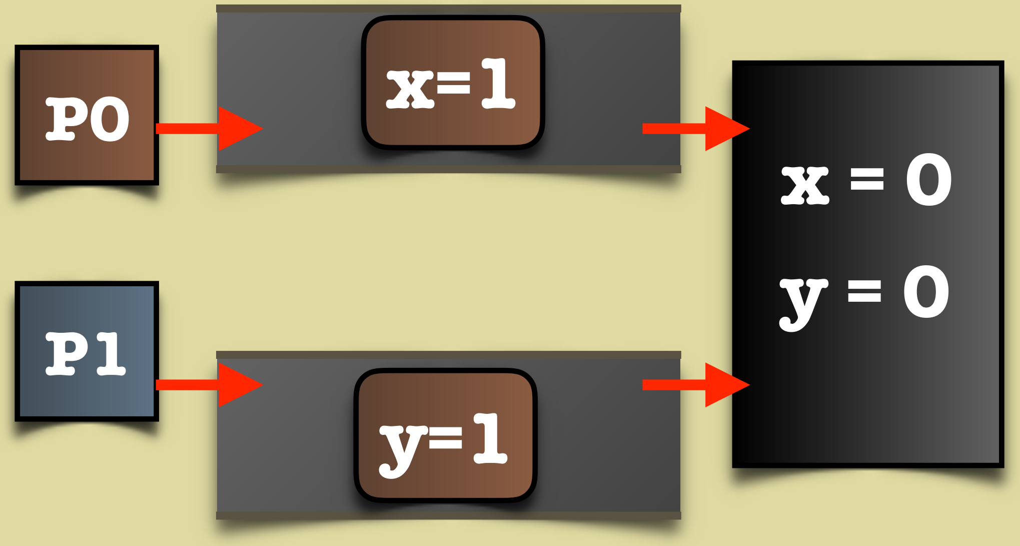
Initially: x = y = 0

P0	P1
write: x = 1	write: y = 1
mfence	mfence
read: y = 0	read: x = 0
critical section	critical section

fence instruction

flushes the buffer

prevents re-ordeirng



TSO

Potentially Bad Behaviours - Dekker

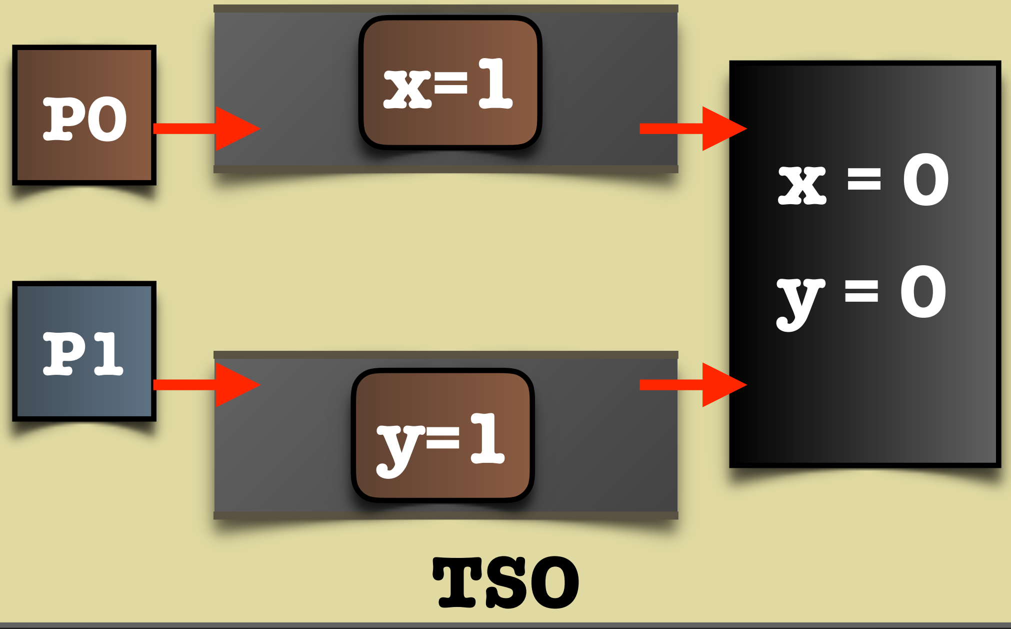
Initially: x = y = 0

P0

write: x = 1
▶ **mfence**
read: y = 0
critical section

P1

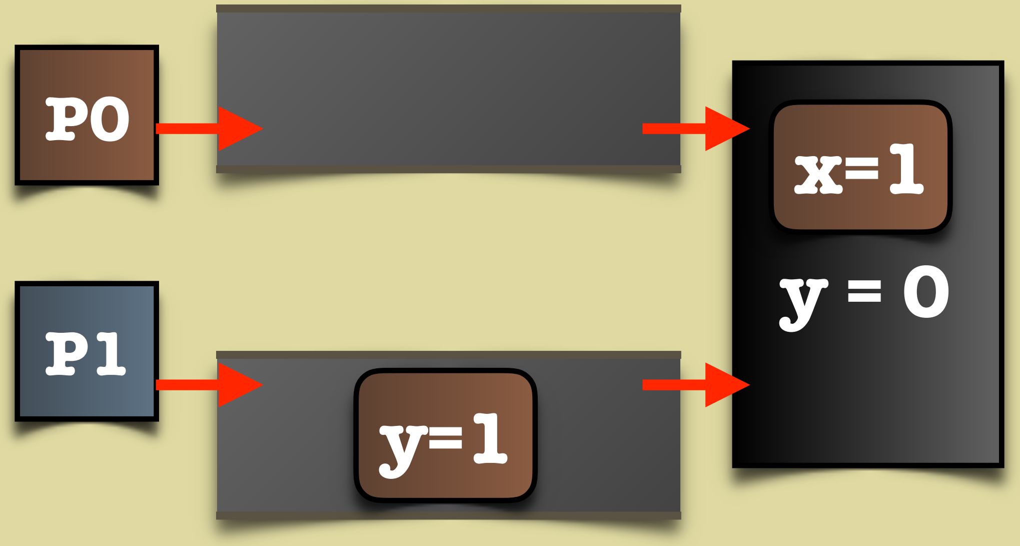
write: y = 1
▶ **mfence**
read: x = 0
critical section



Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	P1
write: x = 1	write: y = 1
<u>mfence</u>	<u>mfence</u>
read: y = 0	read: x = 0
critical section	critical section

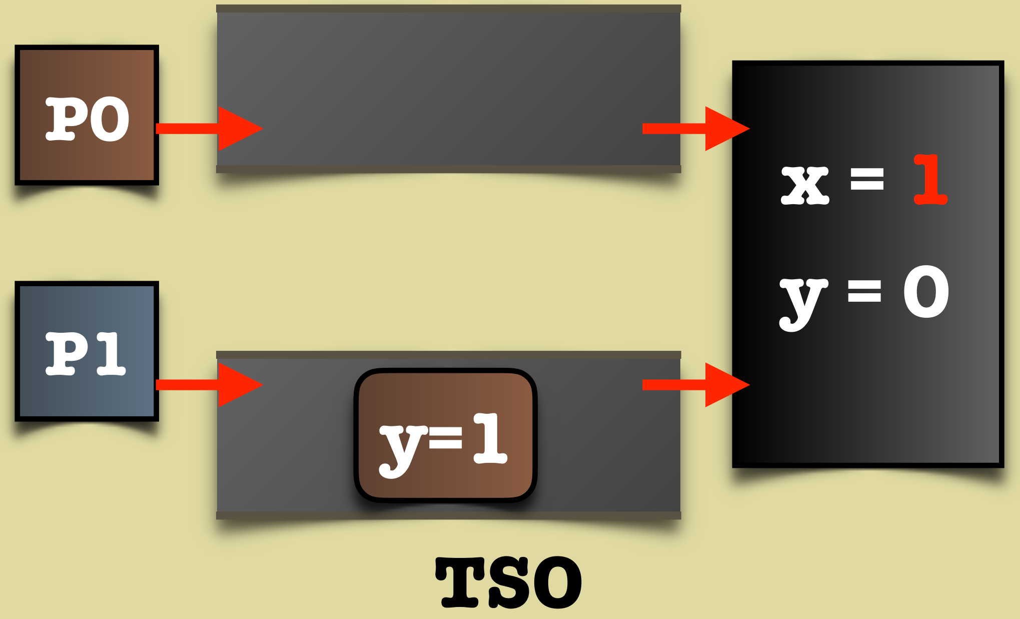


TSO

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	P1
write: x = 1	write: y = 1
mfence	mfence
read: y = 0	read: x = 0
critical section	critical section

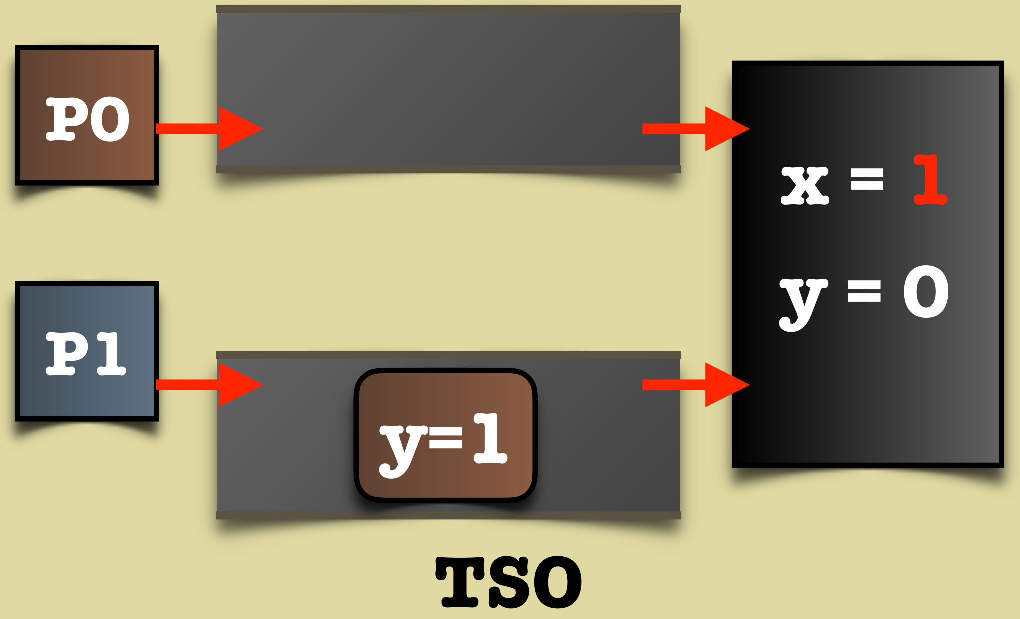


Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	P1
write: x = 1	write: y = 1
mfence	mfence
read: y = 0	read: x = 0
critical section	critical section

execute fence



Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0

write: x = 1

mfence

read: y = 0

critical section

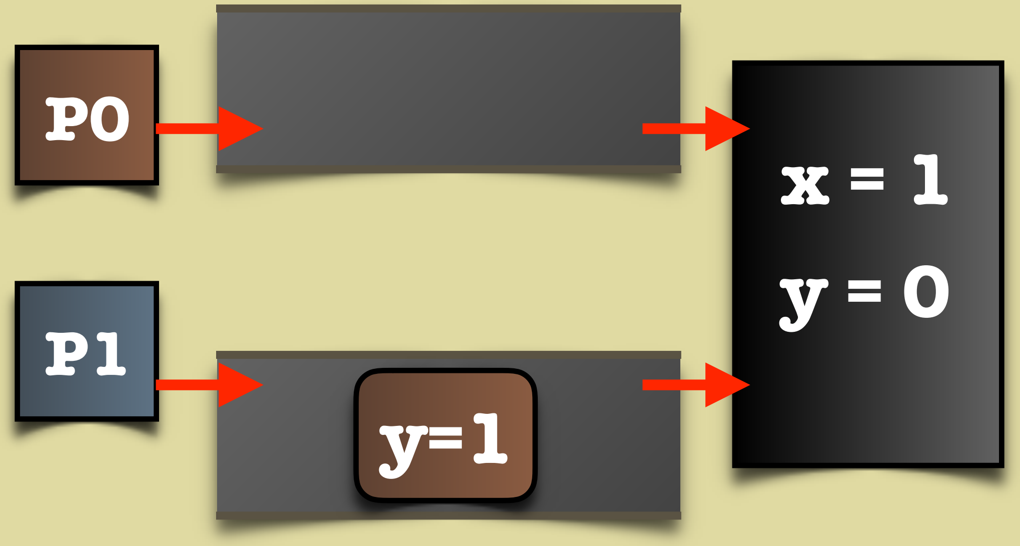
P1

write: y = 1

mfence

read: x = 0

critical section

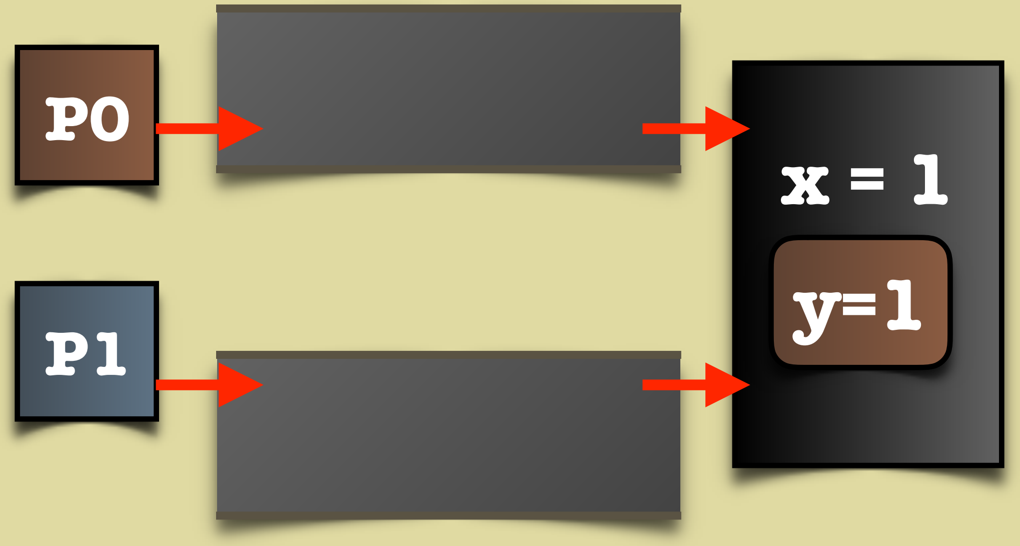


TSO

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	**P1**
write: x = 1	write: y = 1
mfence	mfence
read: y = 0	read: x = 0
critical section	critical section



TSO

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0

write: x = 1

mfence

read: y = 0

critical section

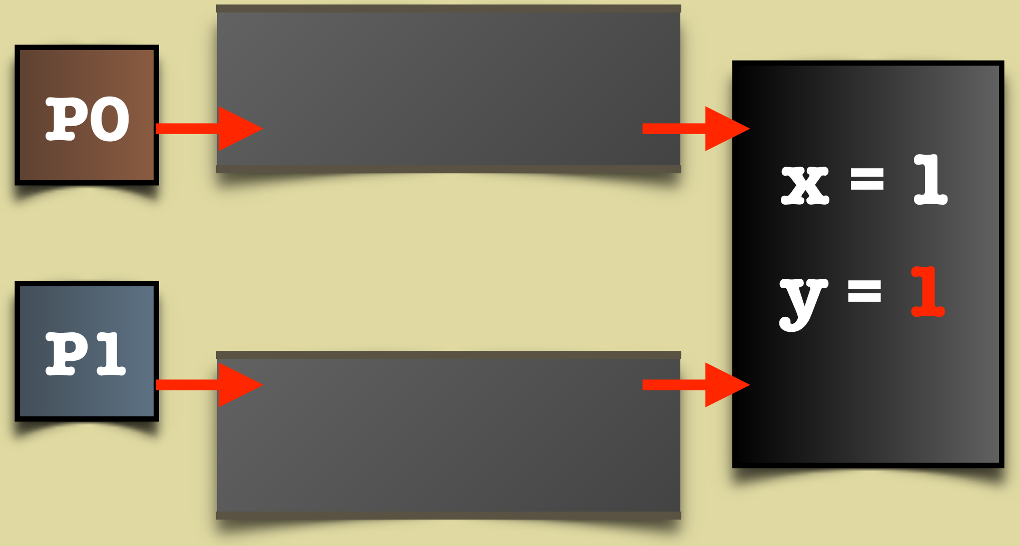
P1

write: y = 1

mfence

read: x = 0

critical section



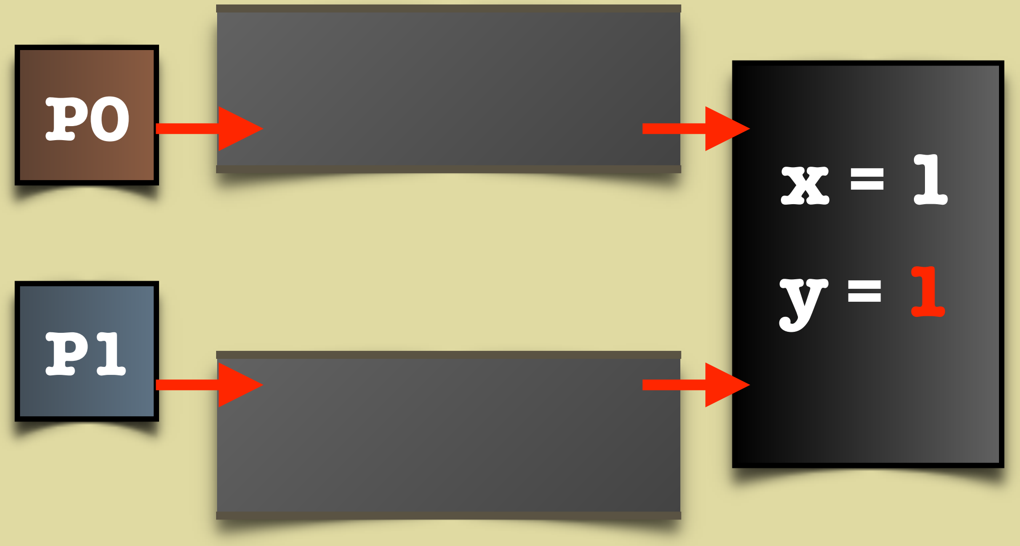
TSO

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	P1
write: x = 1	write: y = 1
mfence	mfence
read: y = 0	read: x = 0
critical section	critical section

execute fence

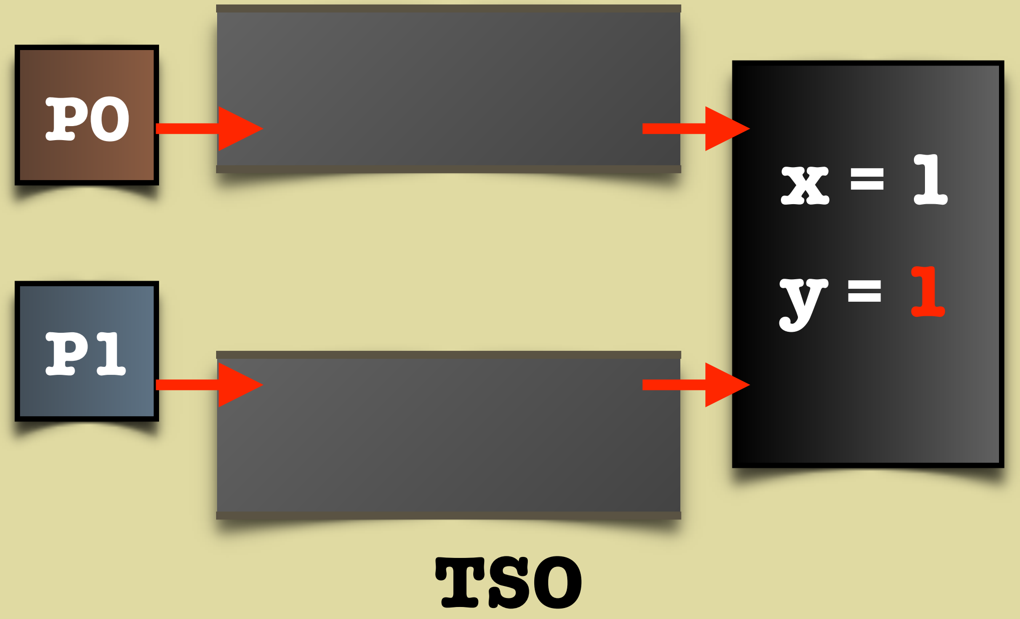


TSO

Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0	**P1**
write: x = 1	write: y = 1
mfence	mfence
read: y = 0	read: x = 0
critical section	critical section



Potentially Bad Behaviours - Dekker

Initially: x = y = 0

P0

write: x = 1

mfence

read: y = 0

critical section

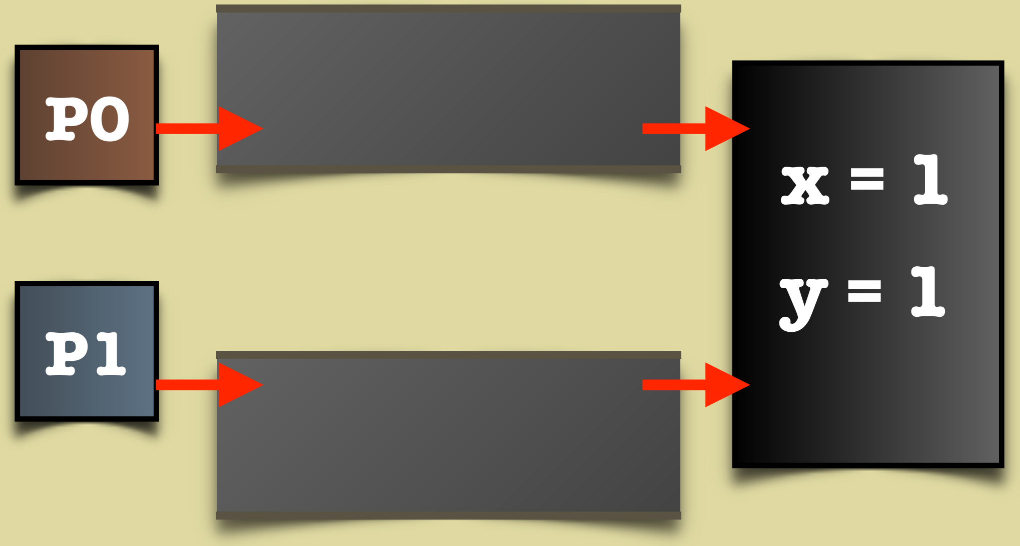
P1

write: y = 1

mfence

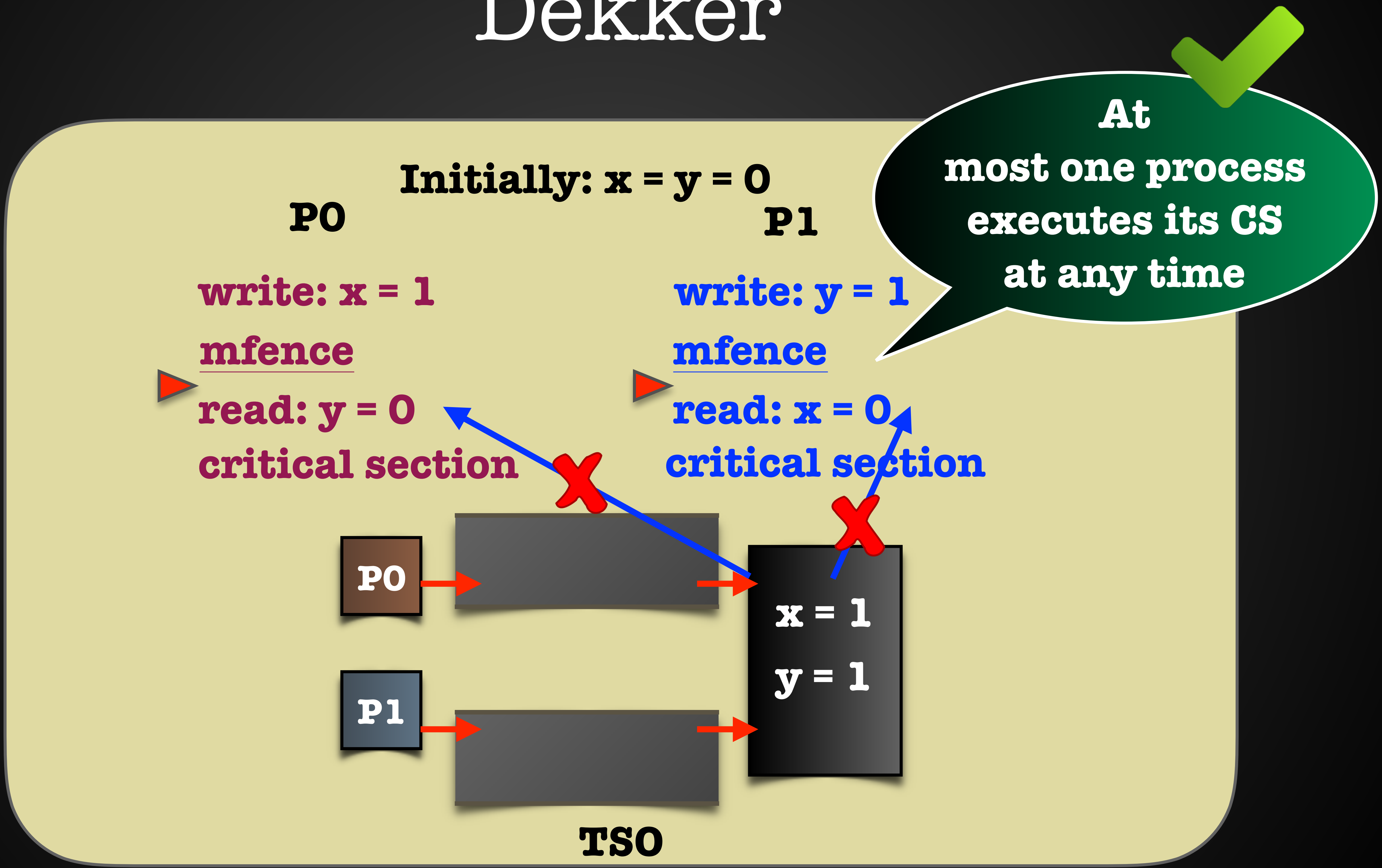
read: x = 0

critical section



TSO

Potentially Bad Behaviours - Dekker



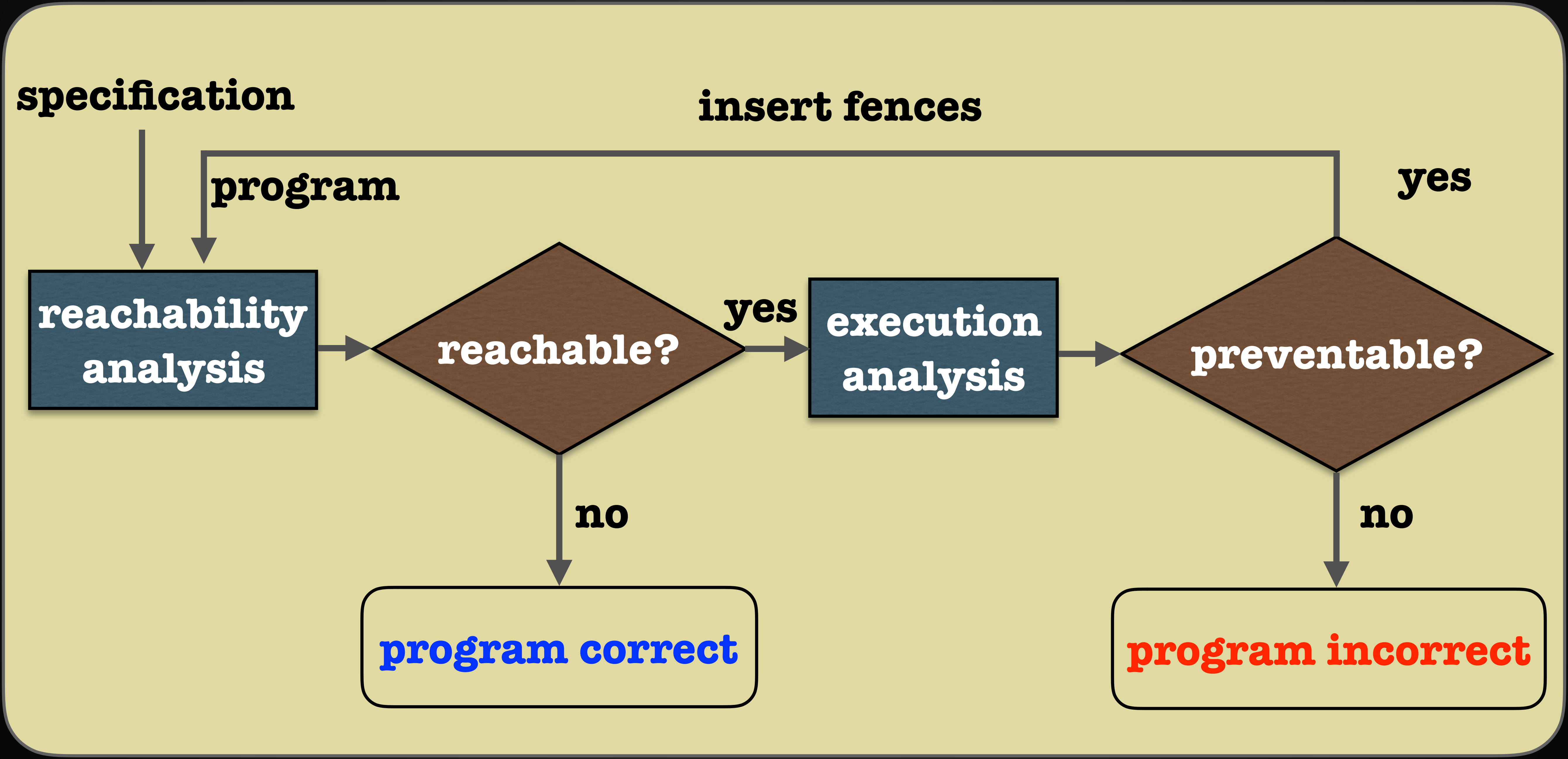
Initially: x = y = 0

P0	P1
write: x = 1	write: y = 1
mfence	mfence
read: y = 0	read: x = 0
critical section	critical section

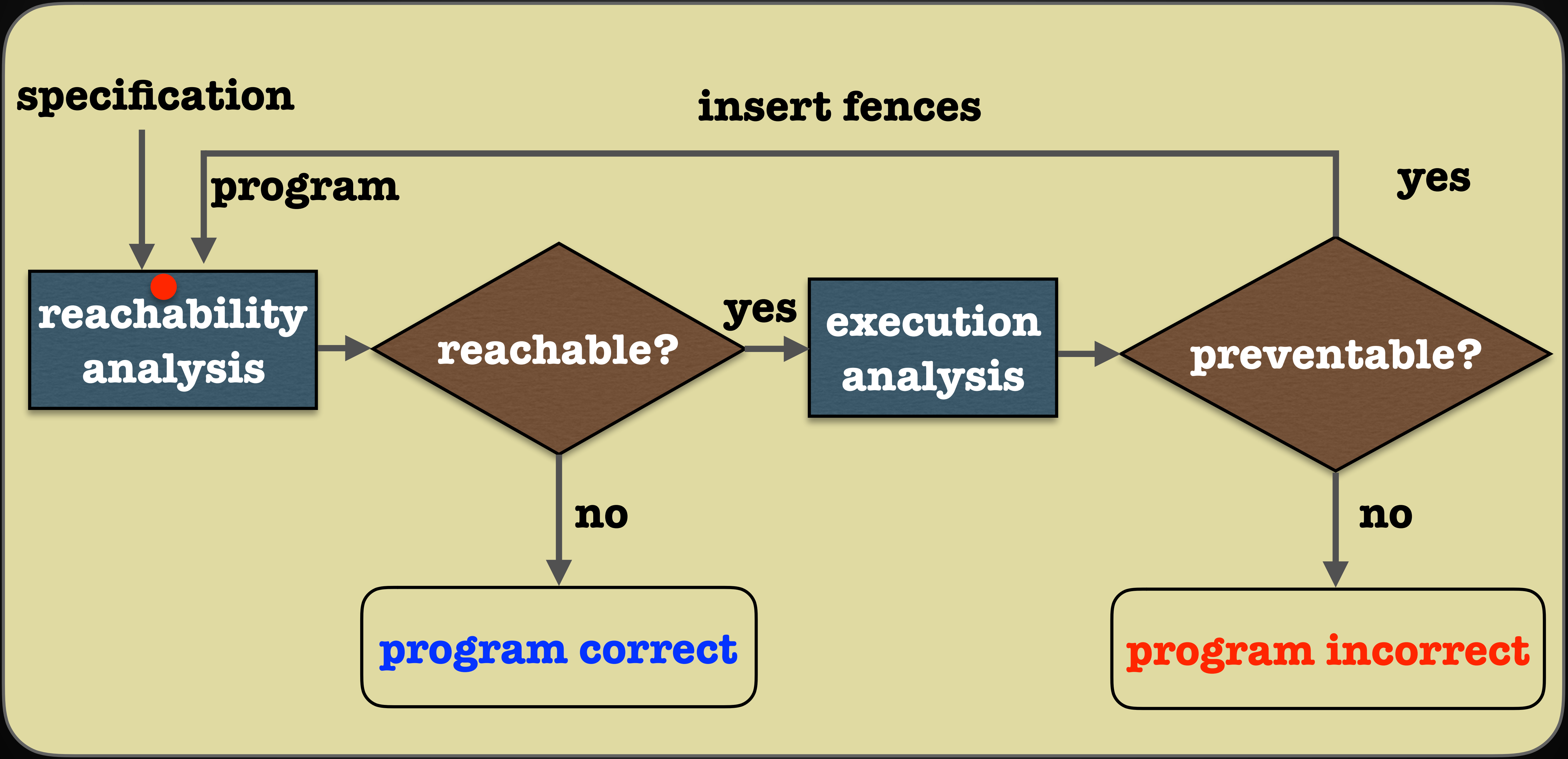
At most one process executes its CS at any time

Verification and Correction

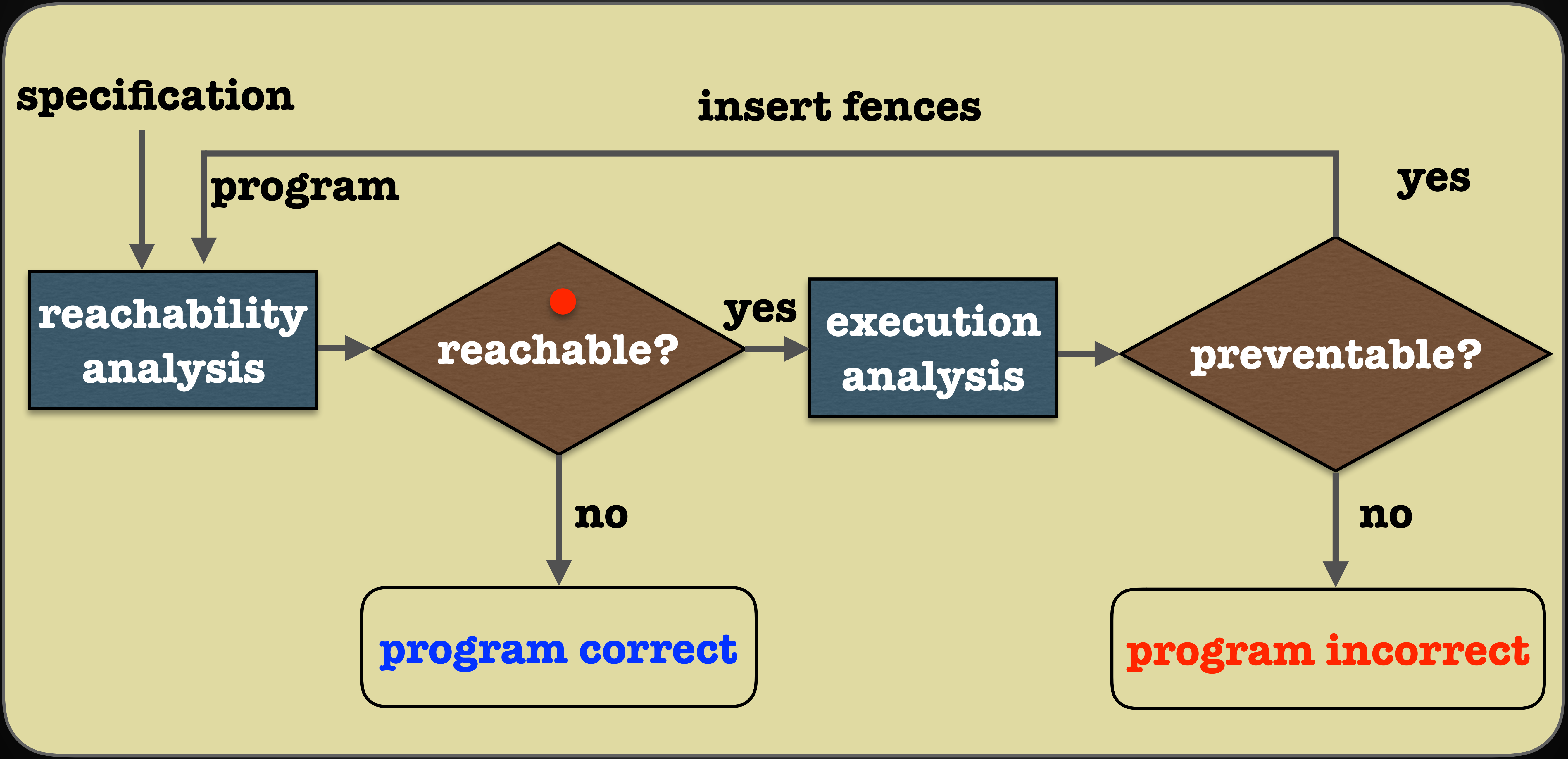
Verification and Correction



Verification and Correction



Verification and Correction



specification

insert fences

program

yes

reachability analysis

reachable?

yes

execution analysis

preventable?

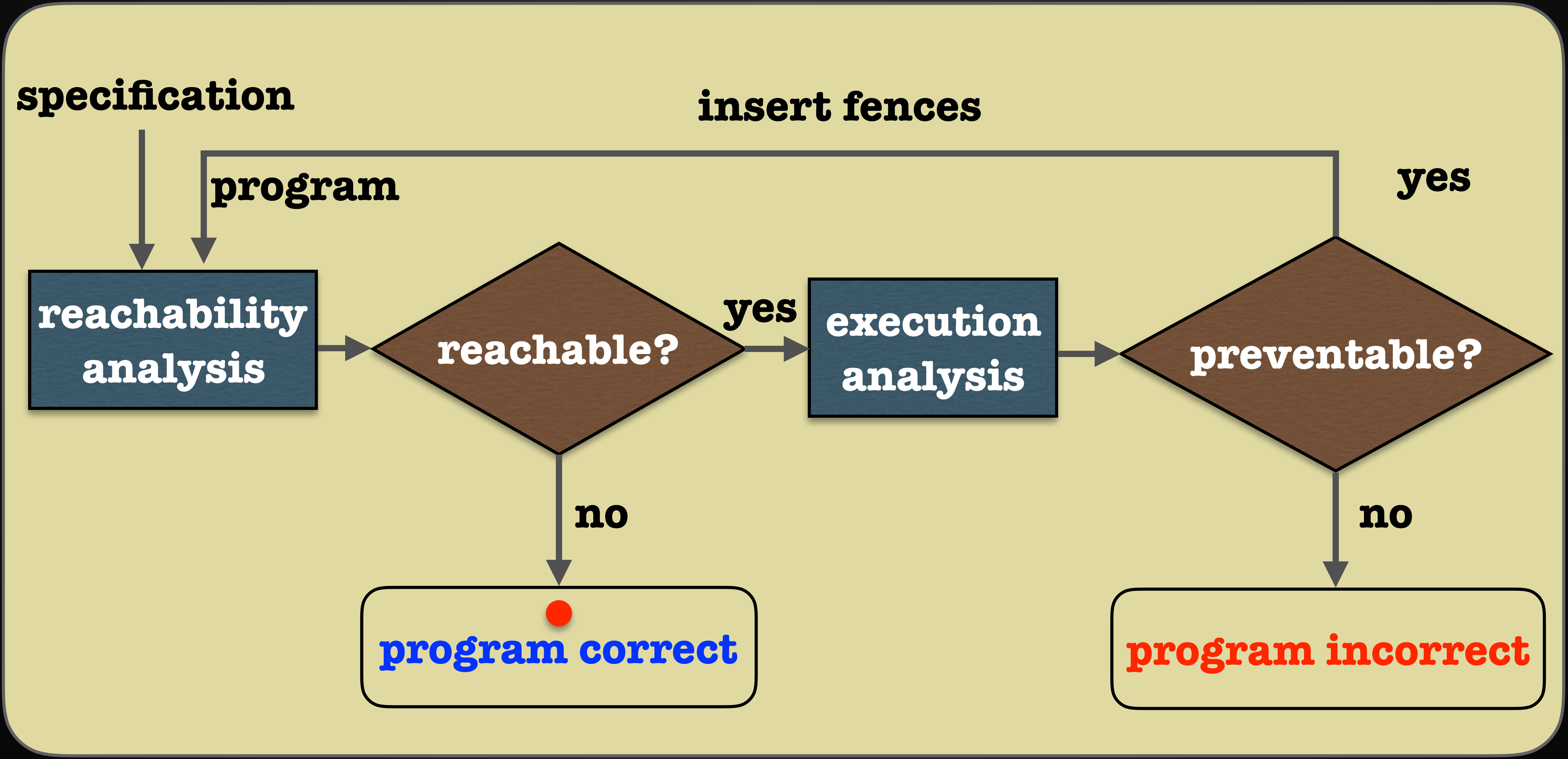
no

no

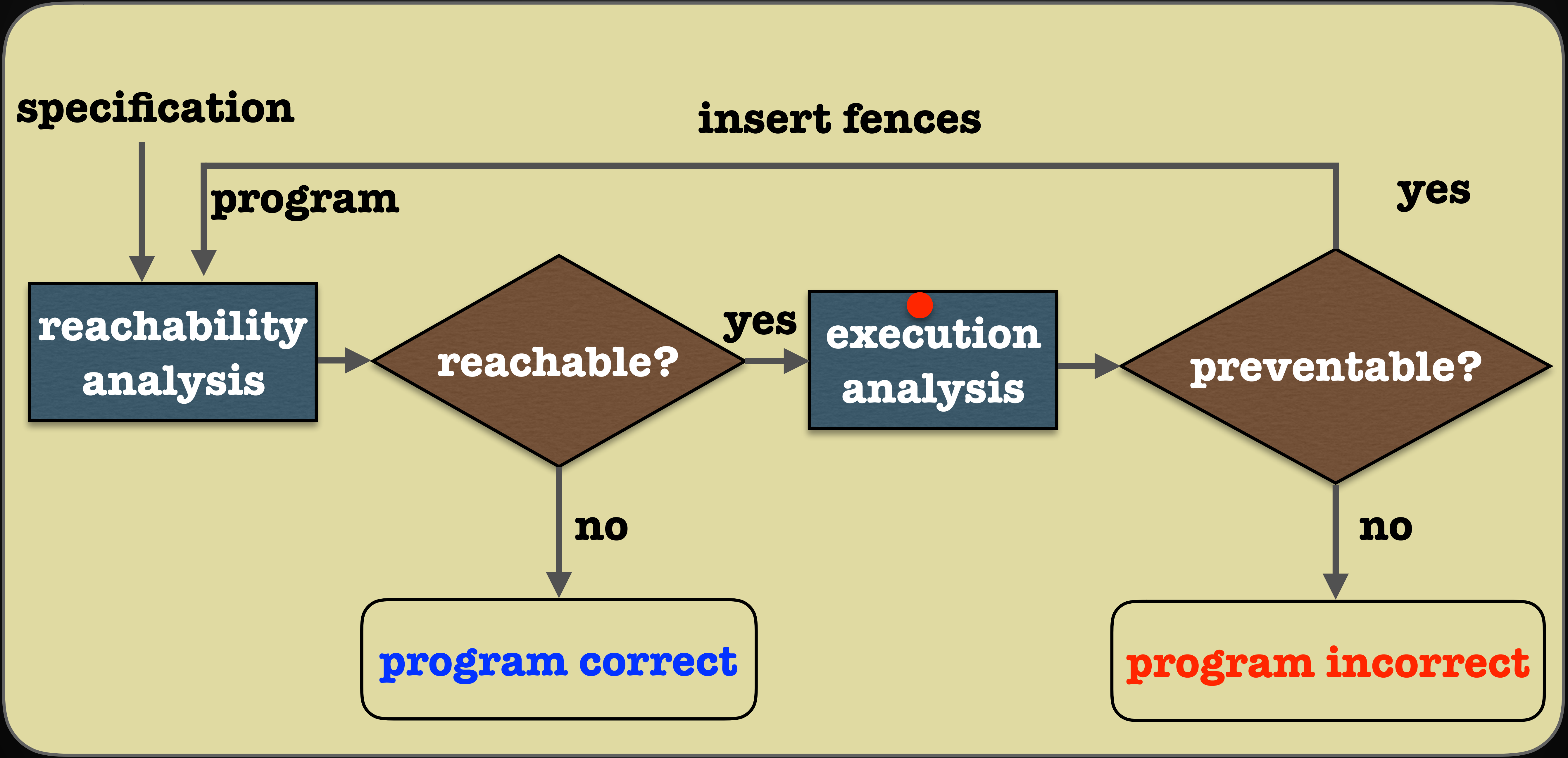
program correct

program incorrect

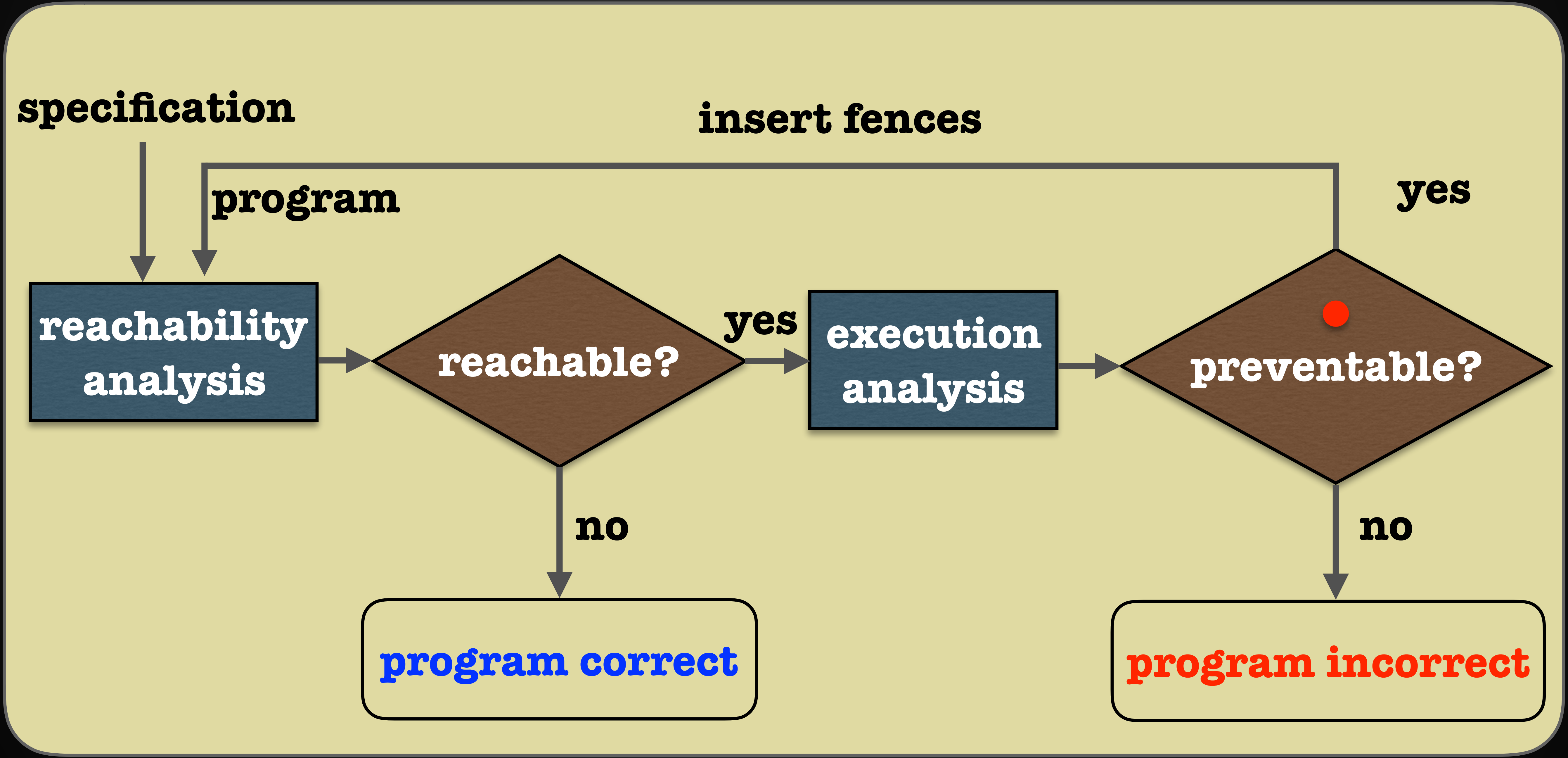
Verification and Correction



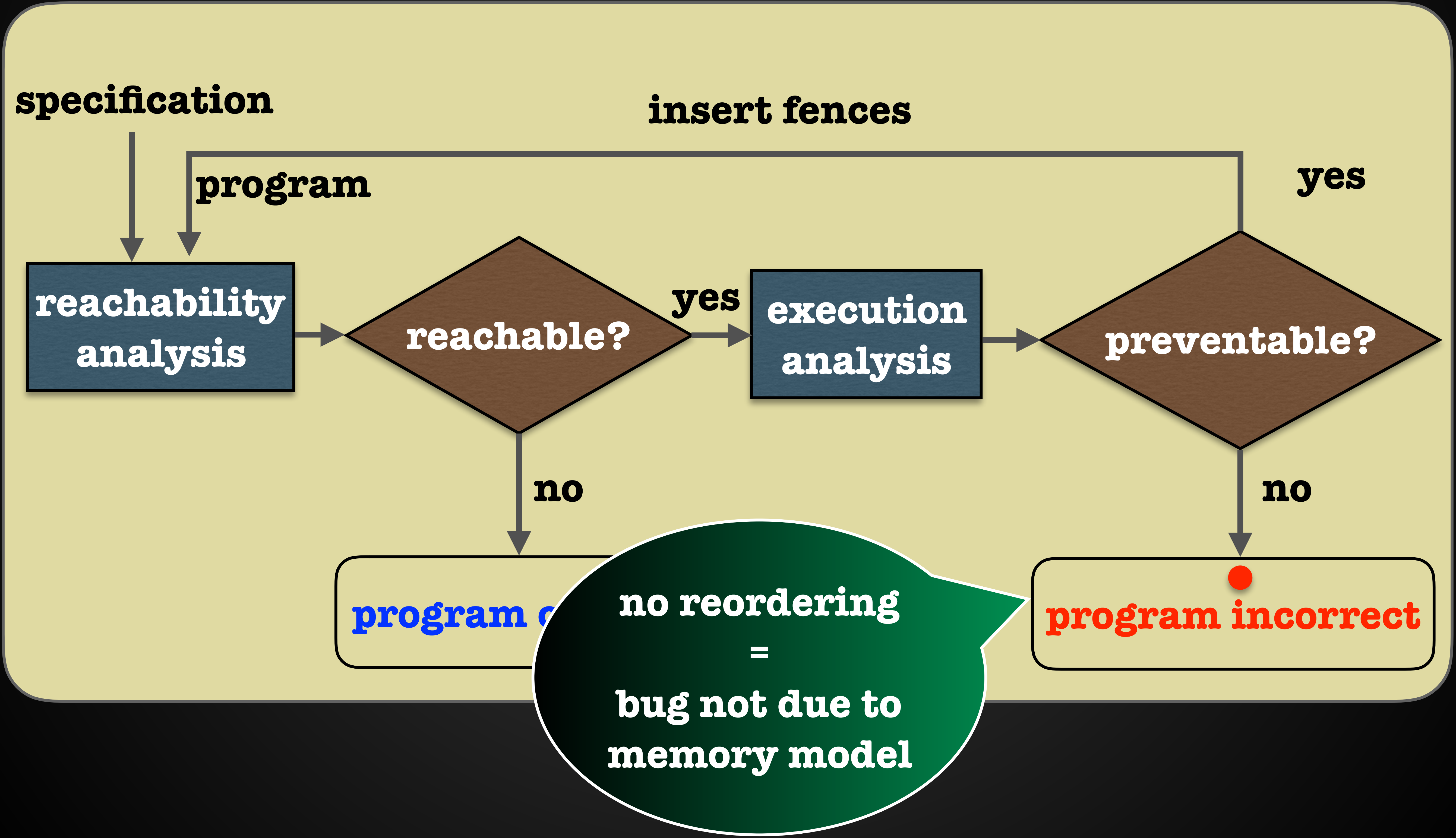
Verification and Correction



Verification and Correction

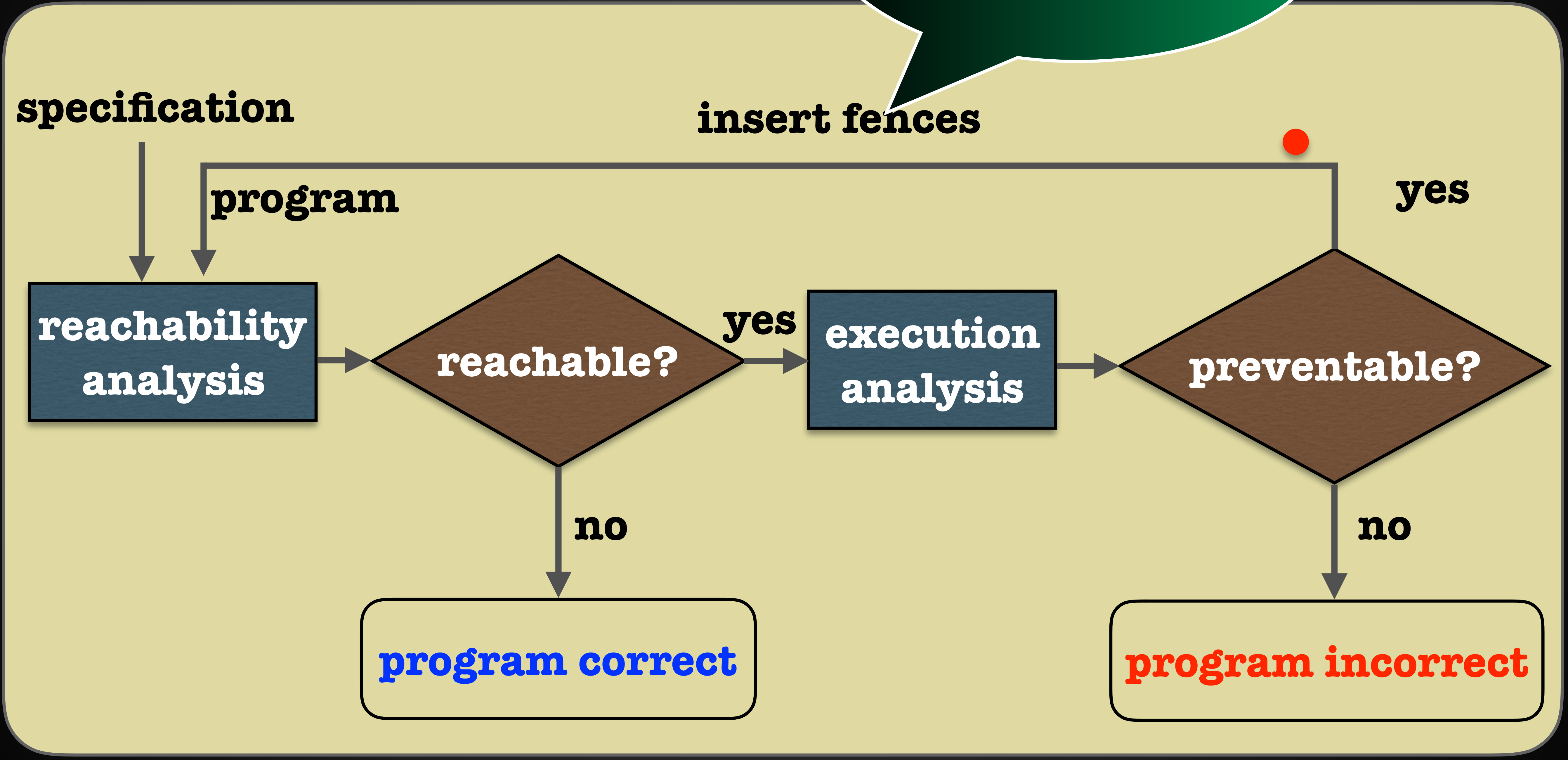


Verification and Correction



Verification an n

find reordering
and
prevent it



specification

program

insert fences

yes

reachability analysis

reachable?

yes

execution analysis

preventable?

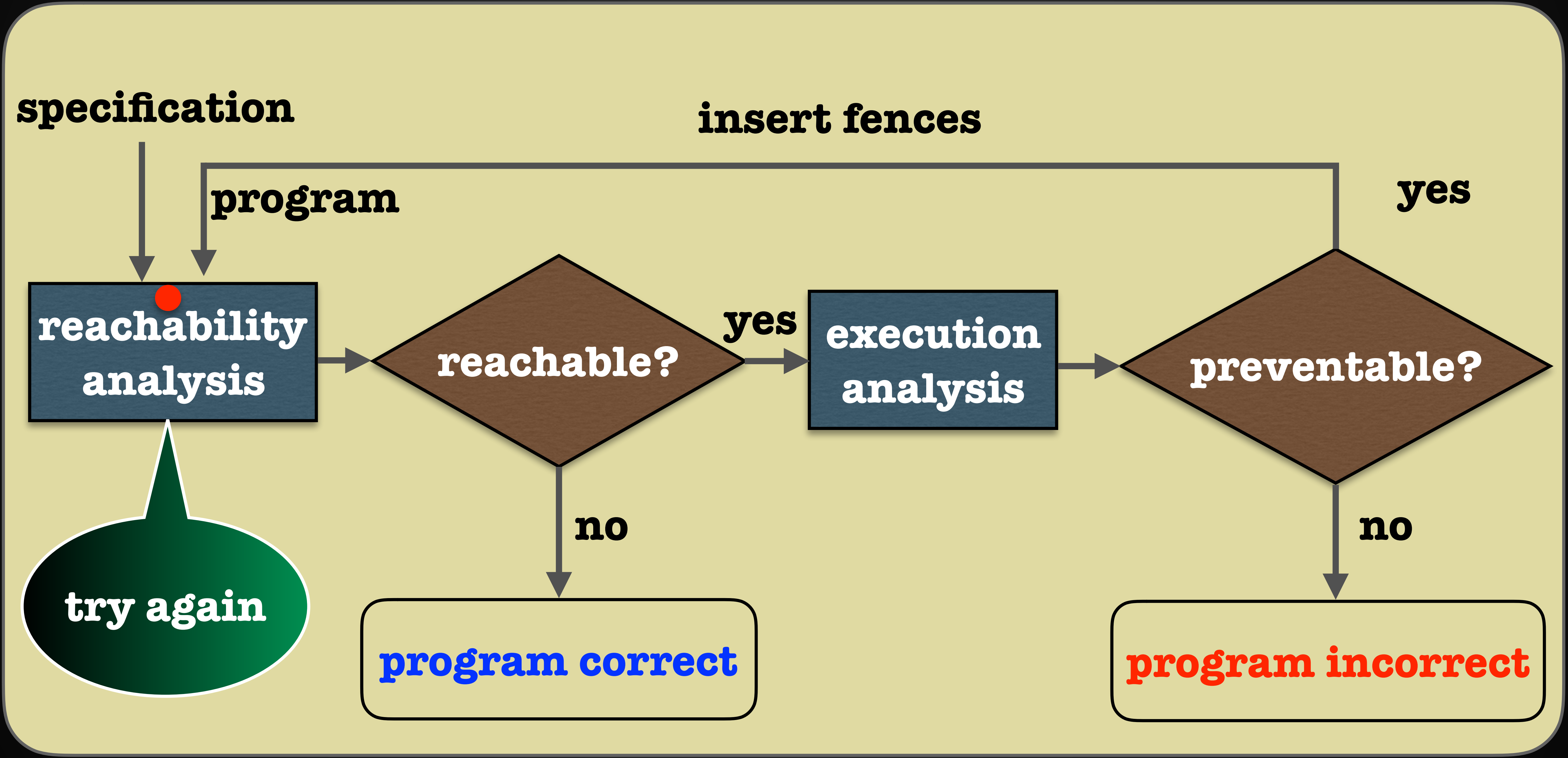
no

no

program correct

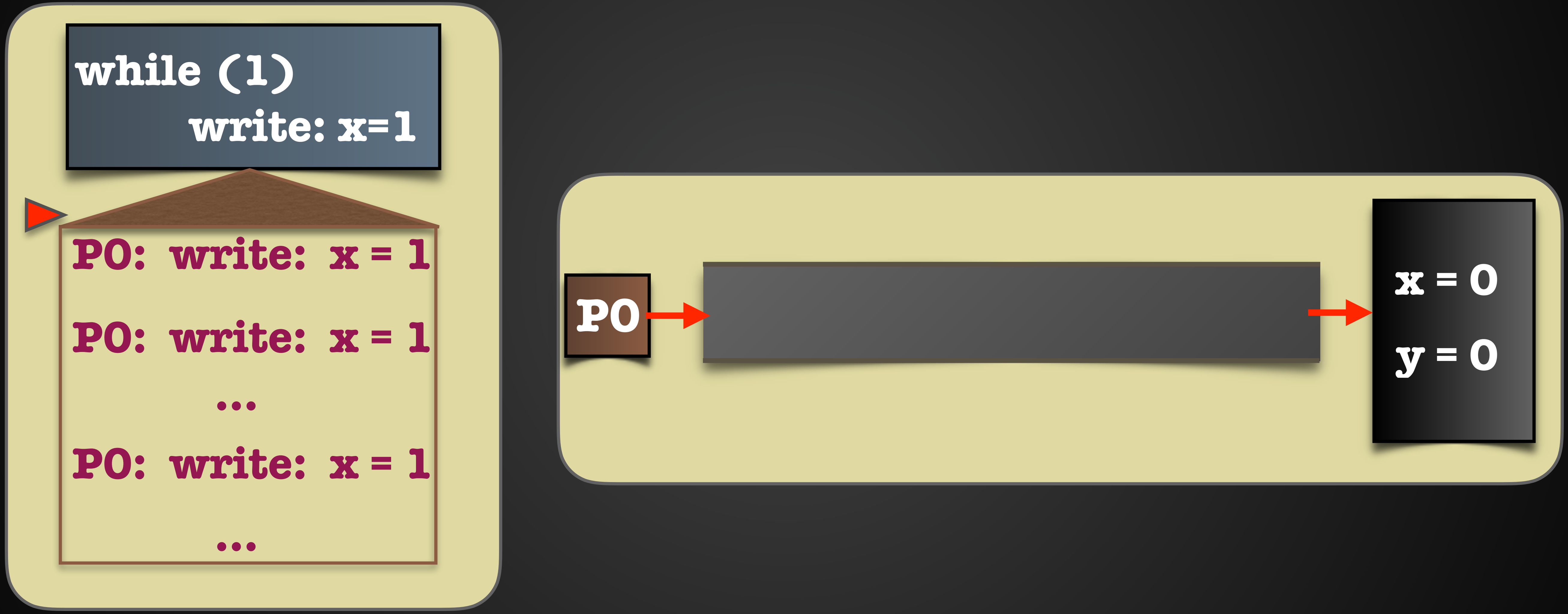
program incorrect

Verification and Correction



optimality = smallest set of fences needed for correctness

Verification under TSO is Difficult



```
while (1)
        write: x=1
```

PO: write: x = 1

PO: write: x = 1

...

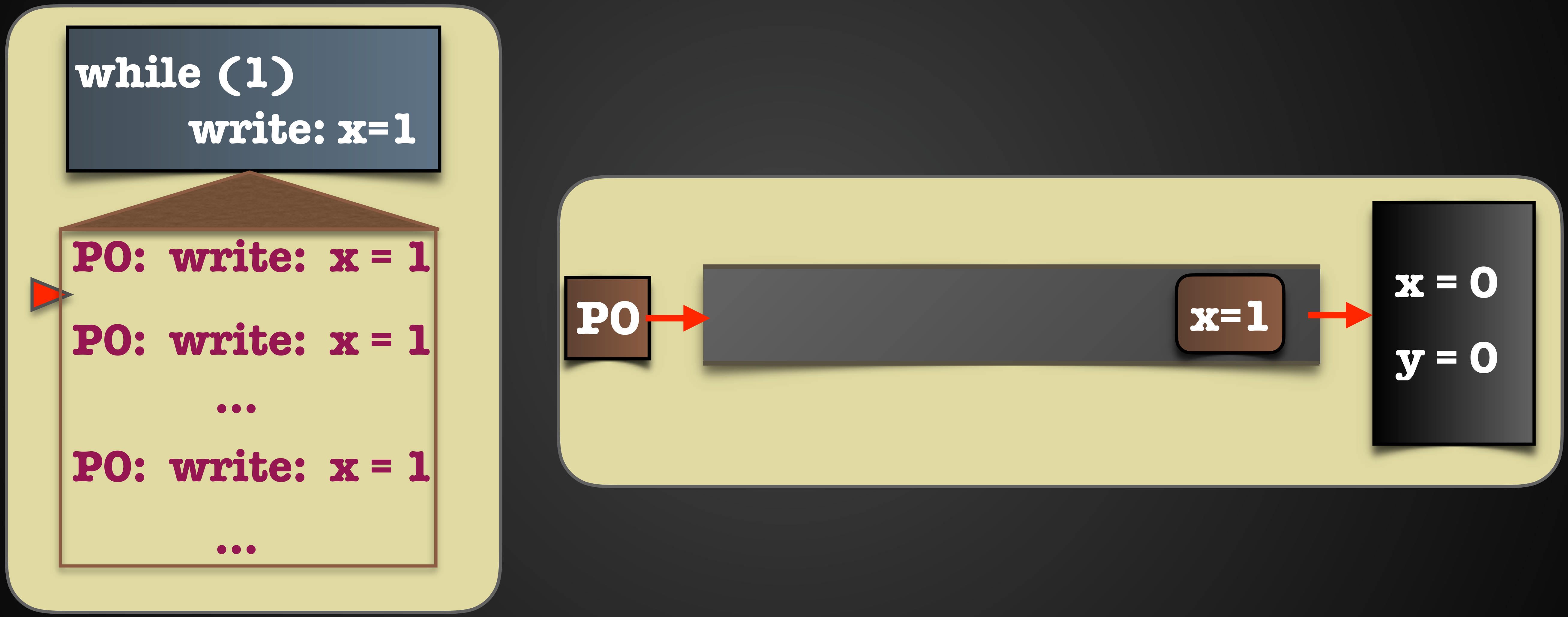
PO: write: x = 1

...

x = 0

y = 0

Verification under TSO is Difficult



```
while (1)
        write: x=1
```

P0: write: x = 1

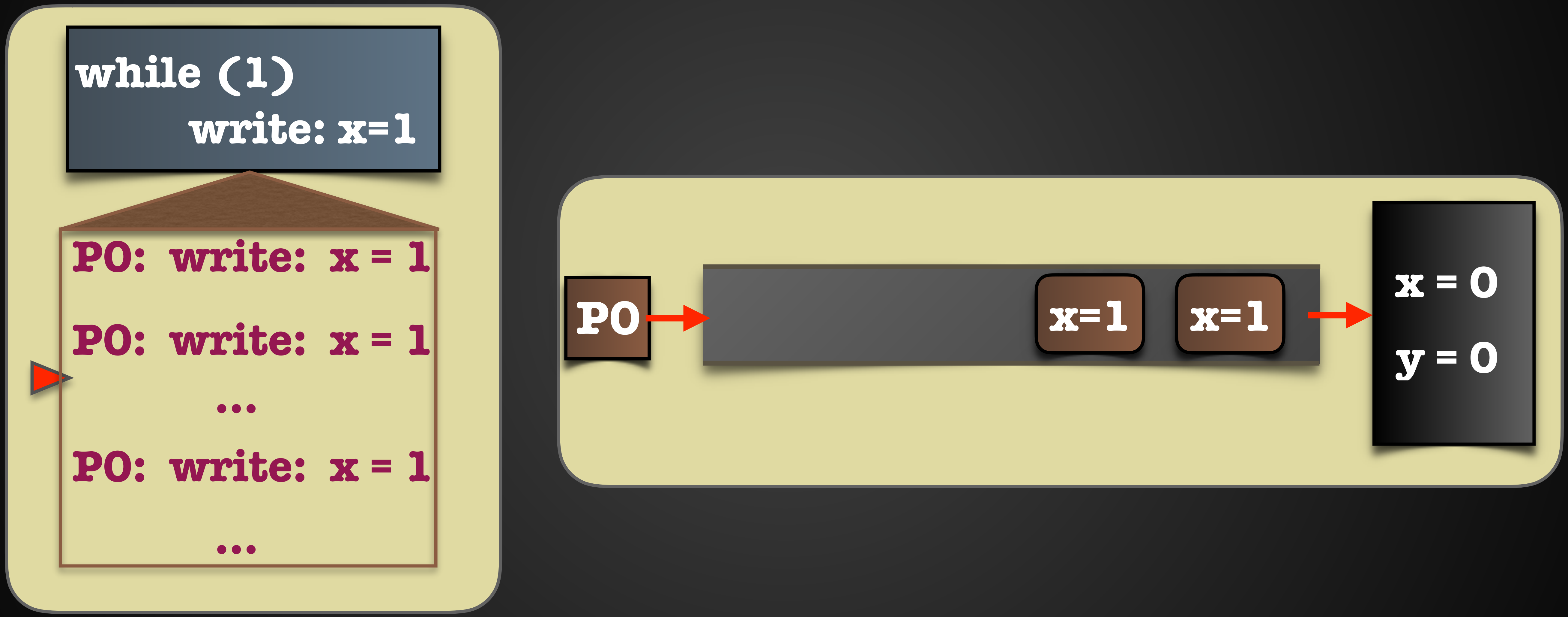
P0: write: x = 1

...

P0: write: x = 1

...

Verification under TSO is Difficult



```
while (1)
        write: x=1
```

PO: write: x = 1

PO: write: x = 1

...

PO: write: x = 1

...

PO

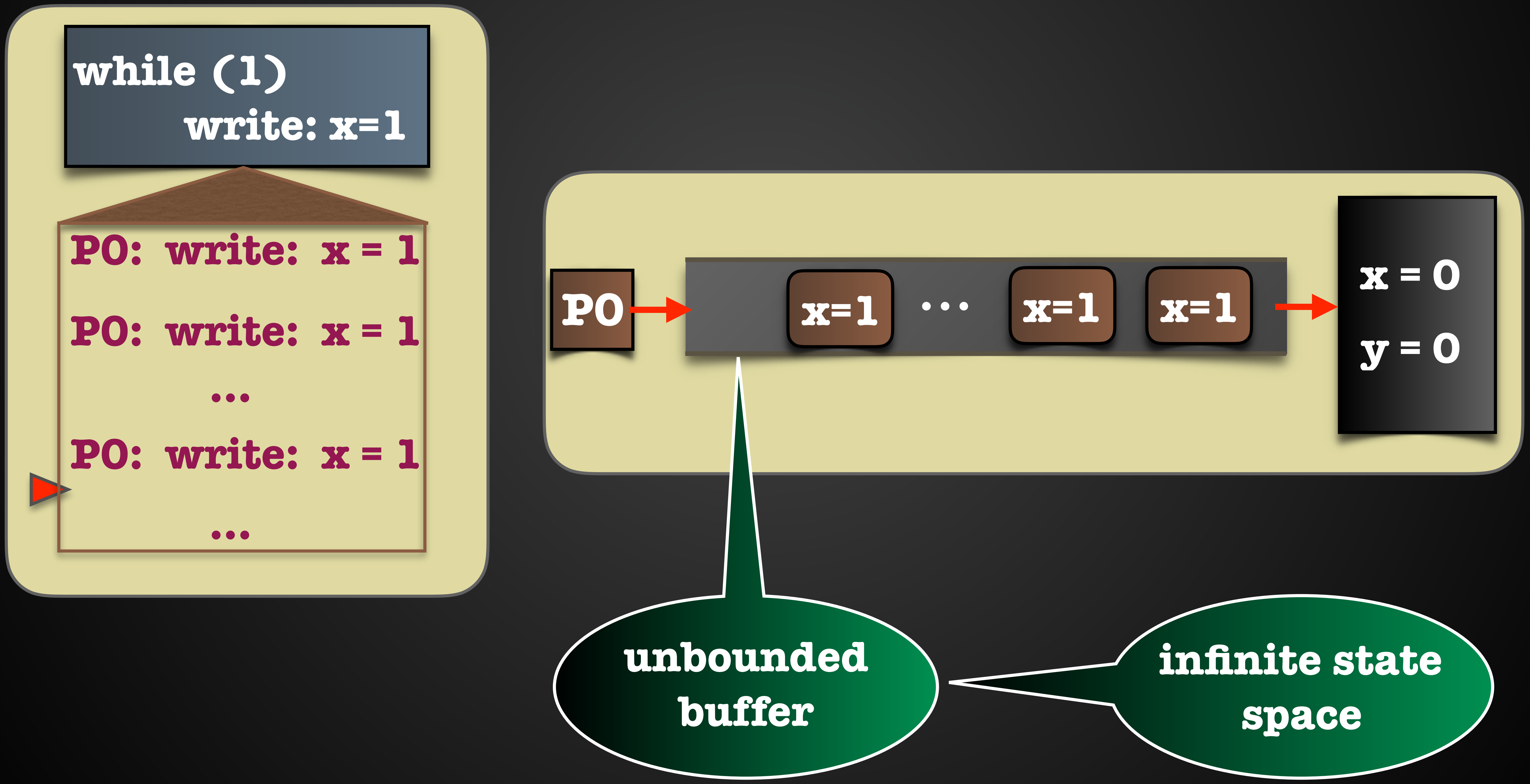
x=1

x=1

x = 0

y = 0

Verification under TSO is Difficult



```
while (1)
        write: x=1
```

PO: write: x = 1

PO: write: x = 1

...

PO: write: x = 1

...

PO

x=1 ... x=1 x=1

x = 0

y = 0

unbounded buffer

infinite state space

Verification under TSO is Difficult

Existing Methods

- Under approximation

  😞 miss bugs: under-fencing

- Over approximation

  😞 spurious bugs: over-fencing

- Exact verification techniques

  😀 find real bugs iff they exist: optimal fencing

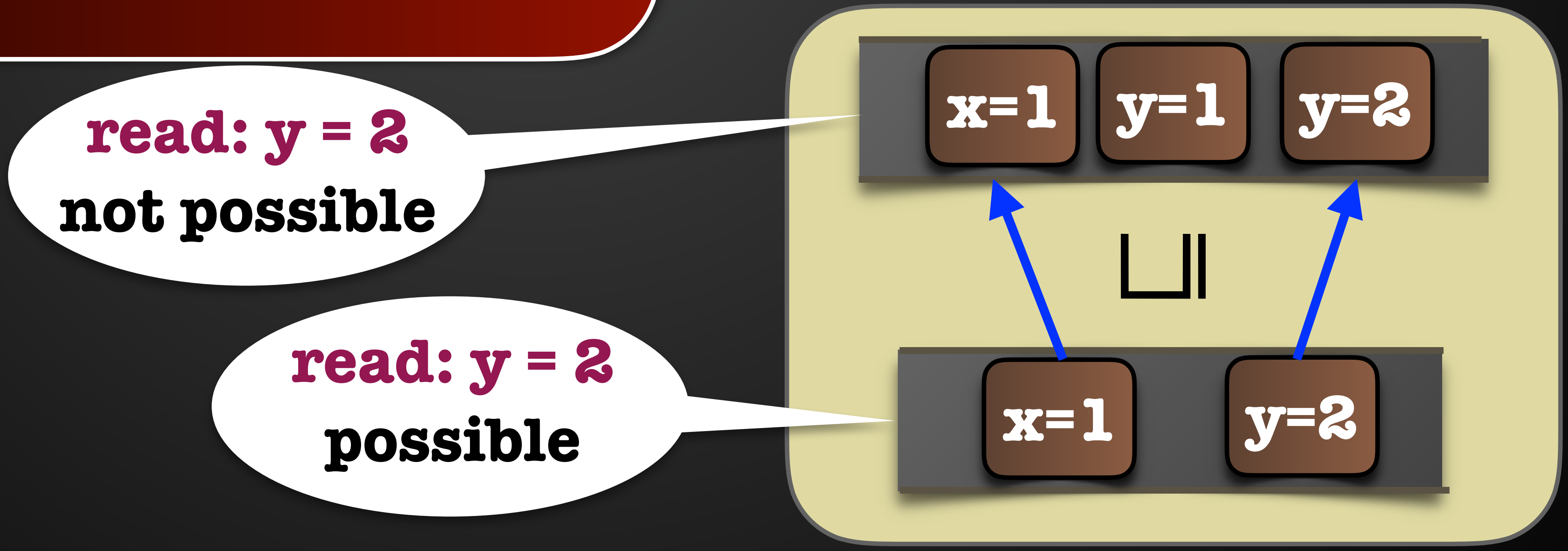
Exact Verification Techniques

Well-Quasi Ordering (WQO) Framework

- ordering on state space:
  - Well-quasi ordering
  - Monotonic transition system

WQO for TSO

- Sub-word ordering on store buffers:
  - monotone?



Exact Verification Techniques

Well-Quasi Ordering (WQO) Framework

- **ordering on state space:**
  - **W**
  - **M**

Monotonicity

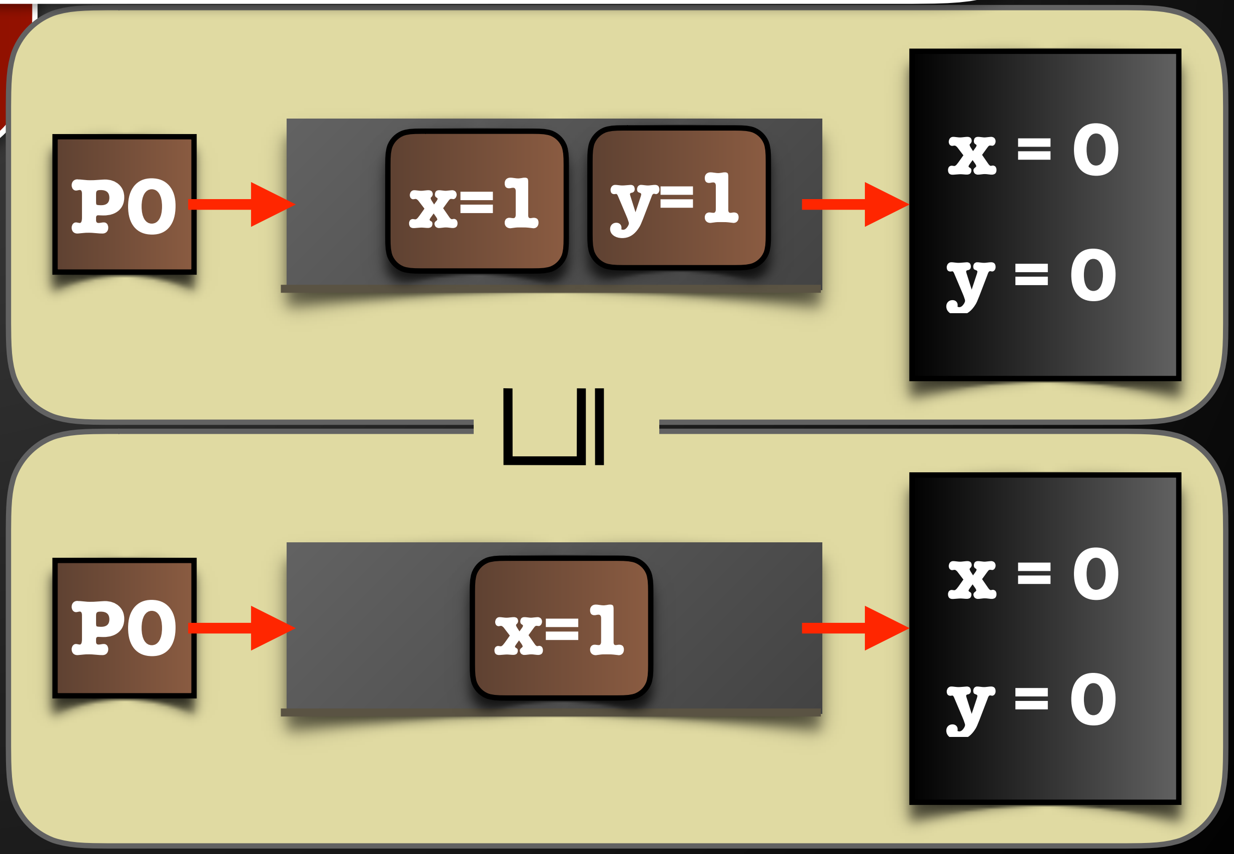
s_1 s_2

$\sqsubseteq$

s_3 s_4

WQO fo

- **Sub-word orderi**
  - **monotone?**



PO

x=1

y=1

x = 0

y = 0

$\sqsubseteq$

PO

x=1

x = 0

y = 0

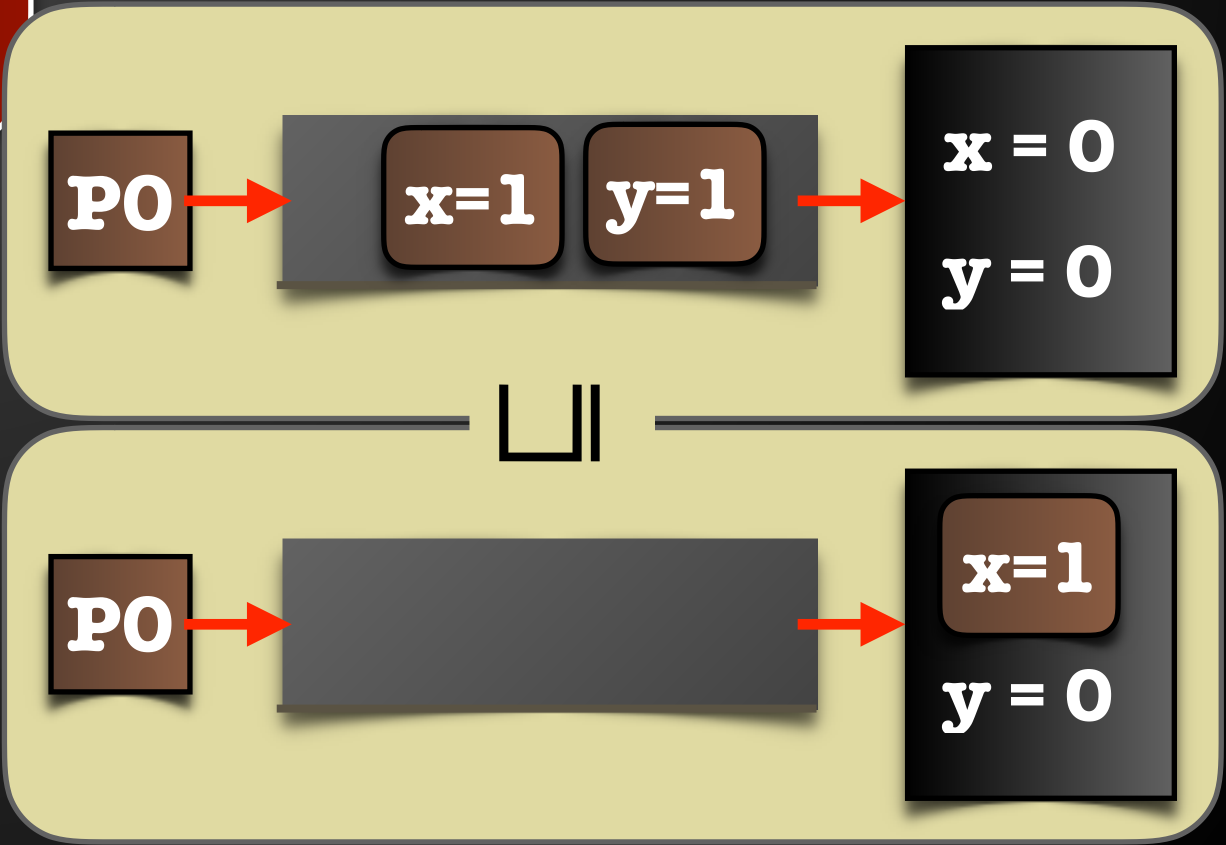
Exact Verification Techniques

Well-Quasi Ordering (WQO) Framework

- ordering on state space:
  - Well-quasi ordering
  - Monotonic transition system

WQO for TSO

- Sub-word ordering on store buffers:
  - monotone?



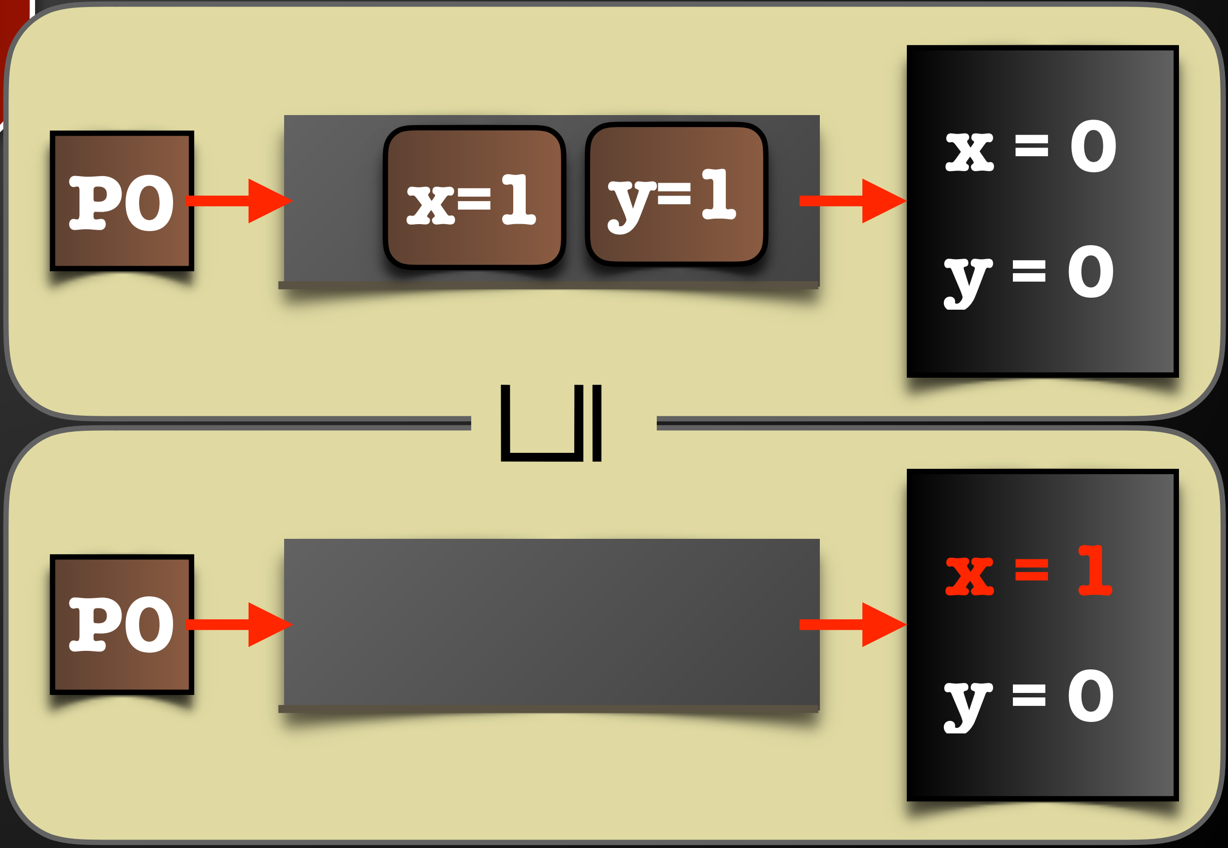
Exact Verification Techniques

Well-Quasi Ordering (WQO) Framework

- ordering on state space:
  - **Well-quasi ordering**
  - **Monotonic** transition system

WQO for TSO

- Sub-word ordering on store buffers:
  - **monotone?**



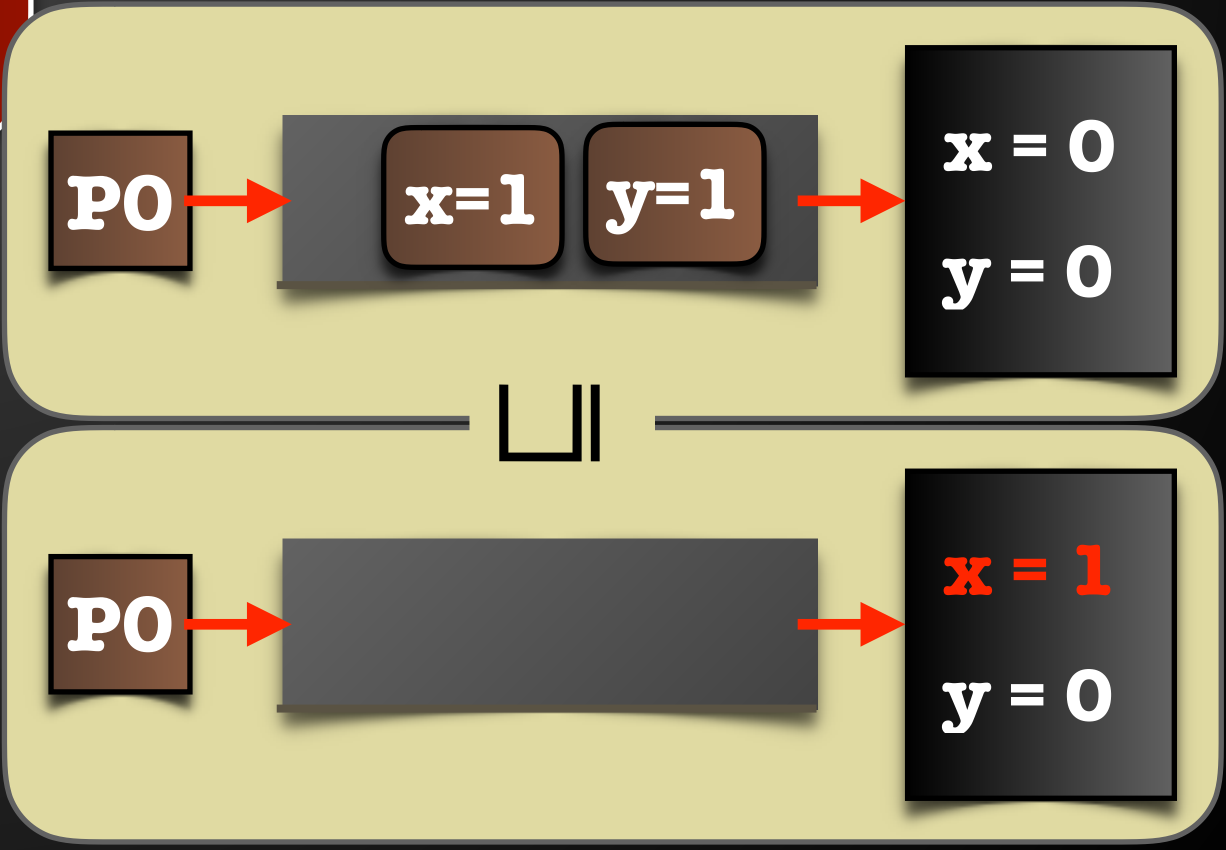
Exact Verification Techniques

Well-Quasi Ordering (WQO) Framework

- ordering on state space:
  - Well-quasi ordering
  - Monotonic transition system

WQO for TSO

- Sub-word ordering on store buffers:
  - monotone? NO!



Exact Verification Techniques

Well-Quasi Ordering (WQO) Framework

- ordering on state space:
  - Well-quasi ordering
  - Monotonic transition system

WQO for TSO

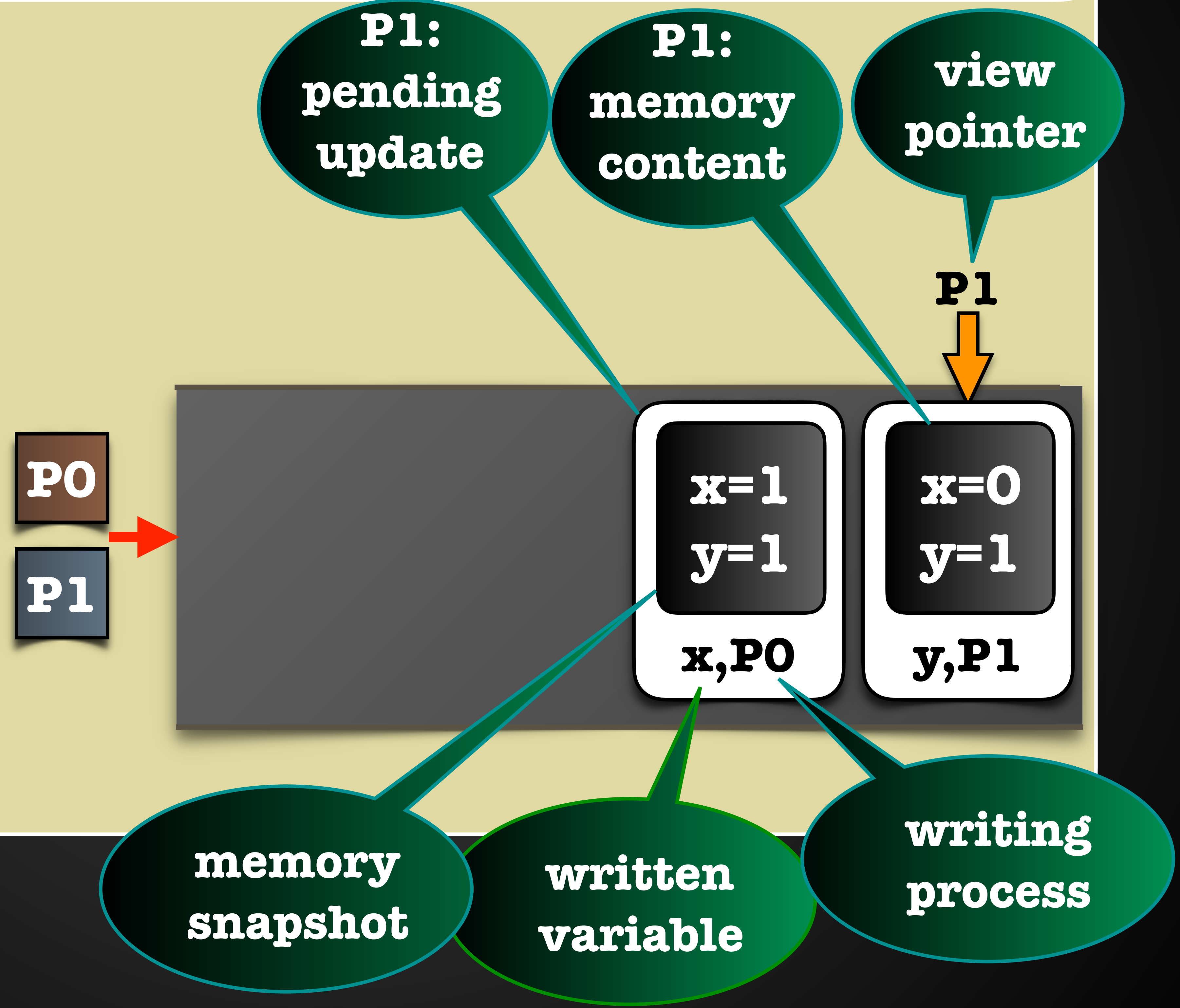
- Sub-word ordering on store buffers?
  - Not monotone!
- WQO cannot be applied easily to TSO

Semantics 2: Single Buffer Model [TACAS'12+13]

P0: write: x = 2

P1: write: y = 3

...

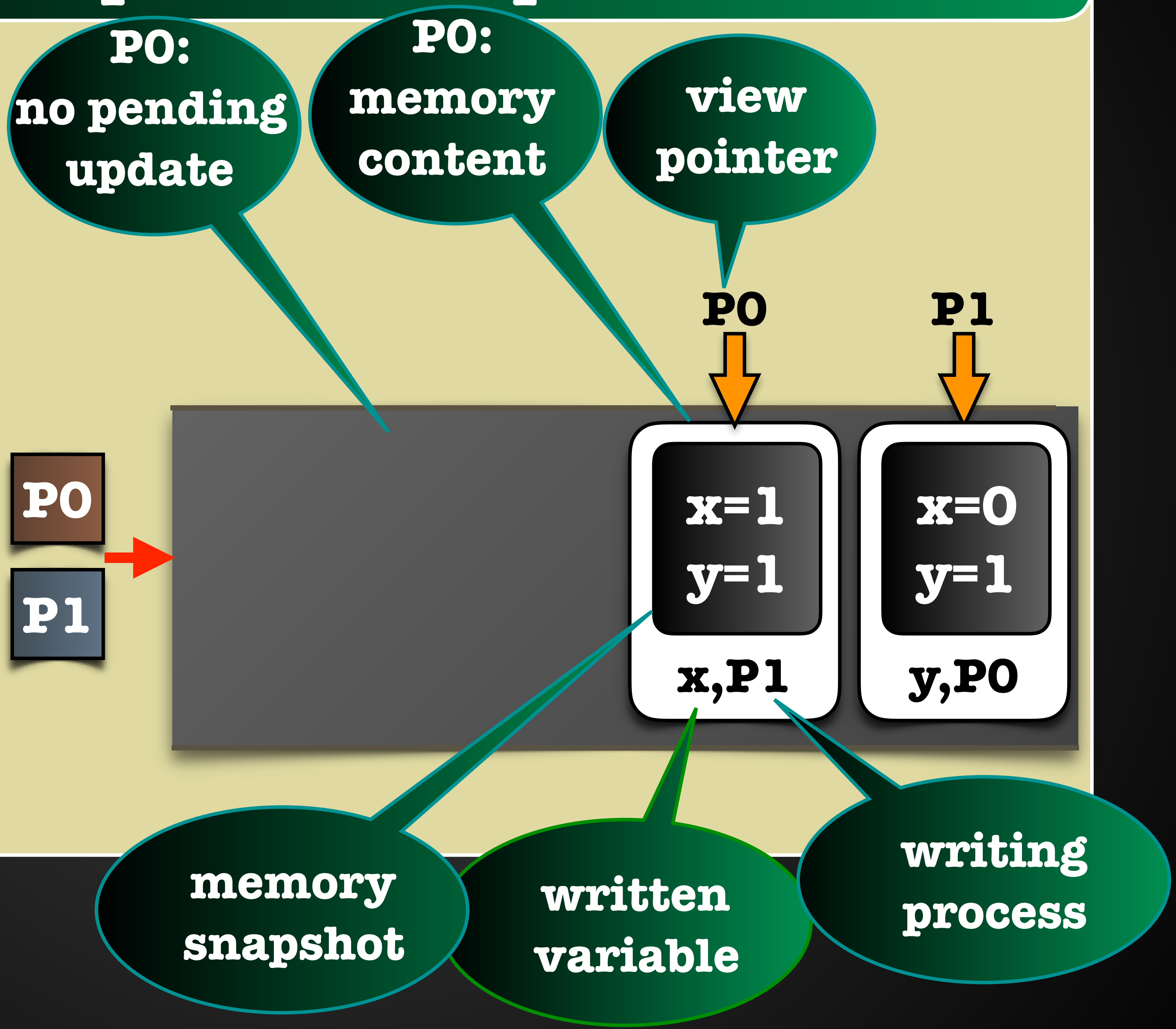


Semantics 2: Single Buffer Model [TACAS'12+13]

P0: write: x = 2

P1: write: y = 3

...

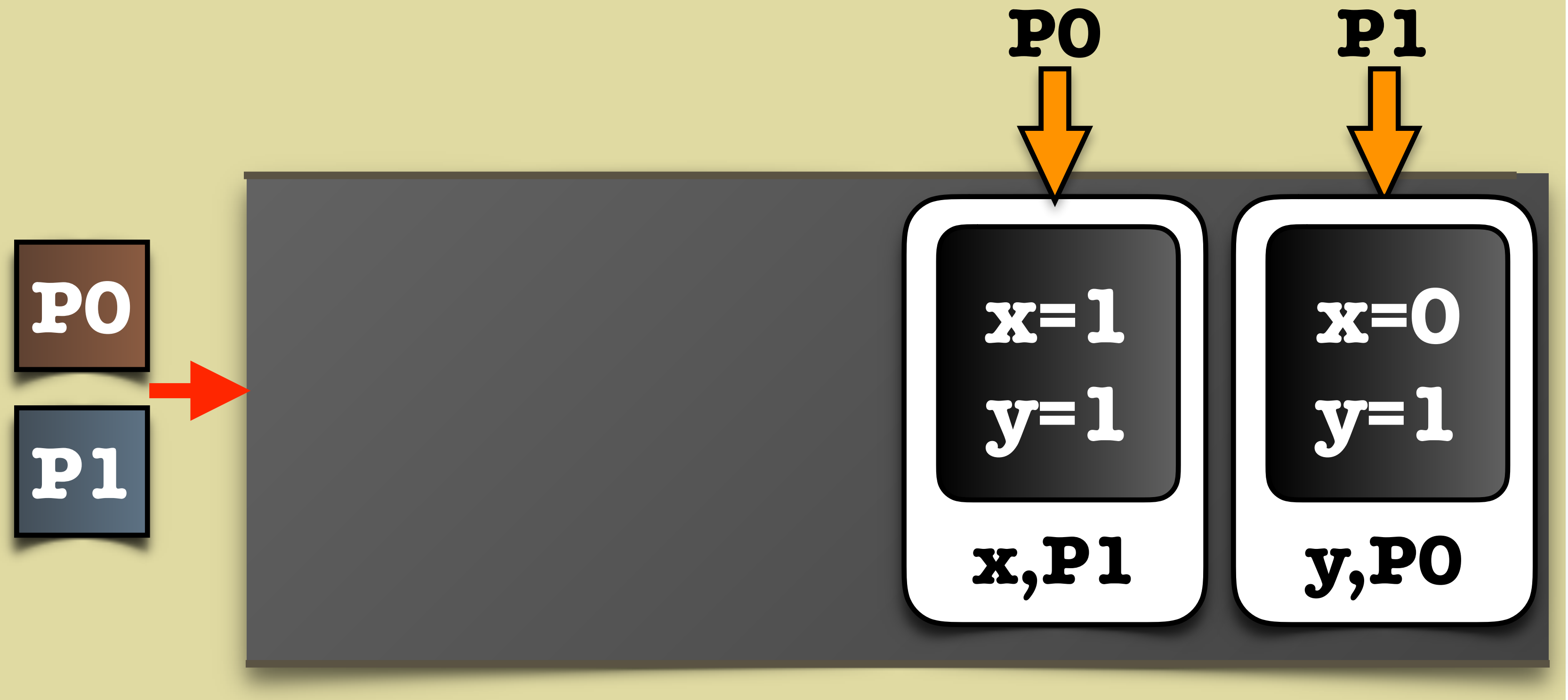


Semantics 2: Single Buffer Model [TACAS'12+13]

P0: write: x = 2

P1: write: y = 3

...

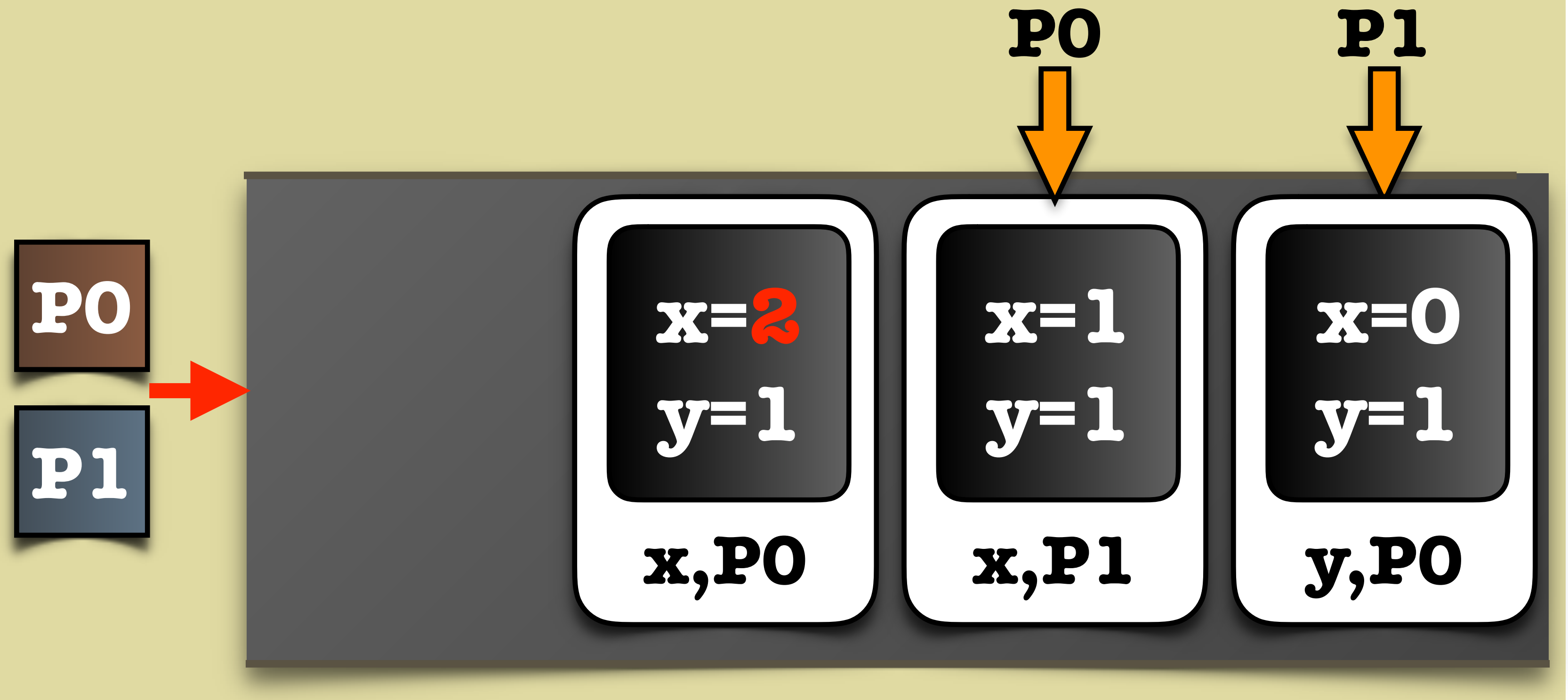


Semantics 2: Single Buffer Model [TACAS'12+13]

P0: write: x = 2

P1: write: y = 3

...



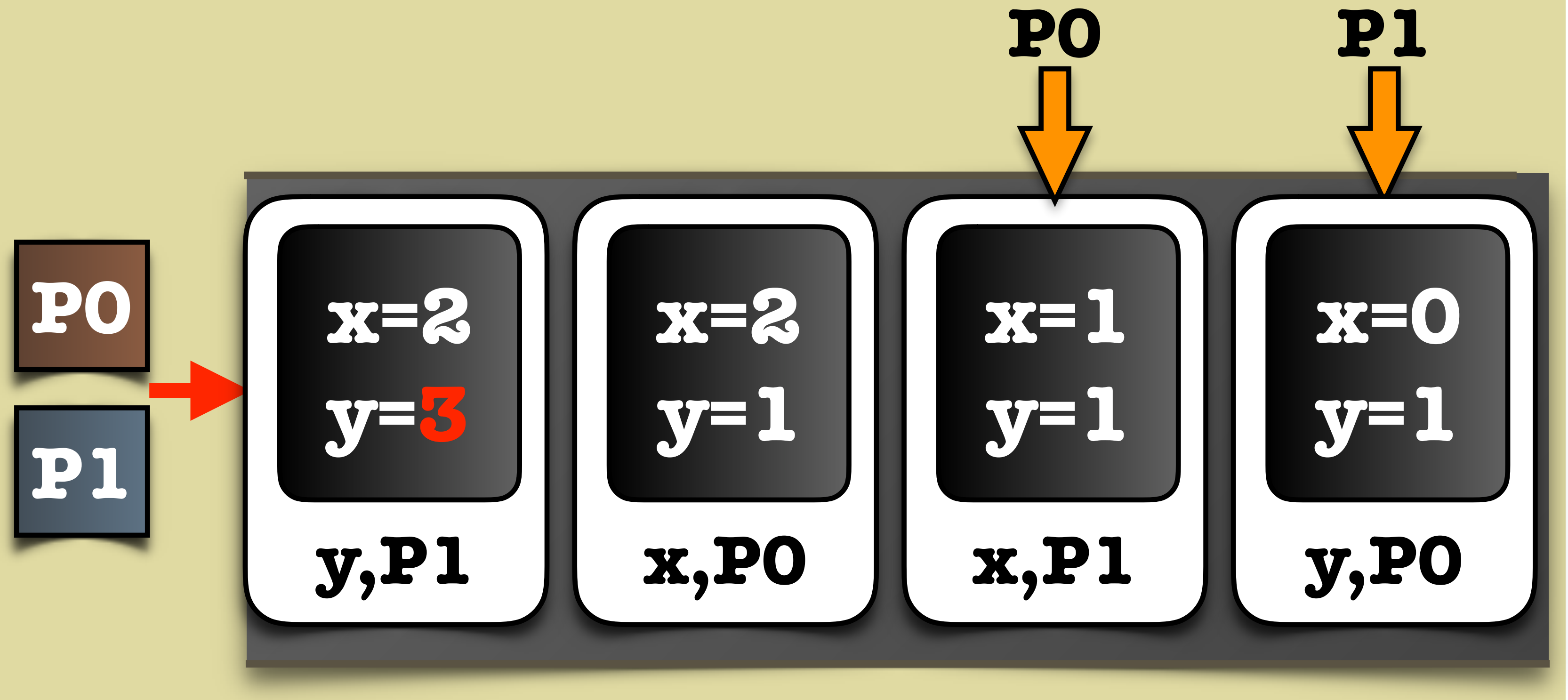
Semantics 2: Single Buffer Model [TACAS'12+13]

▶

P0: write: x = 2

P1: write: y = 3

...

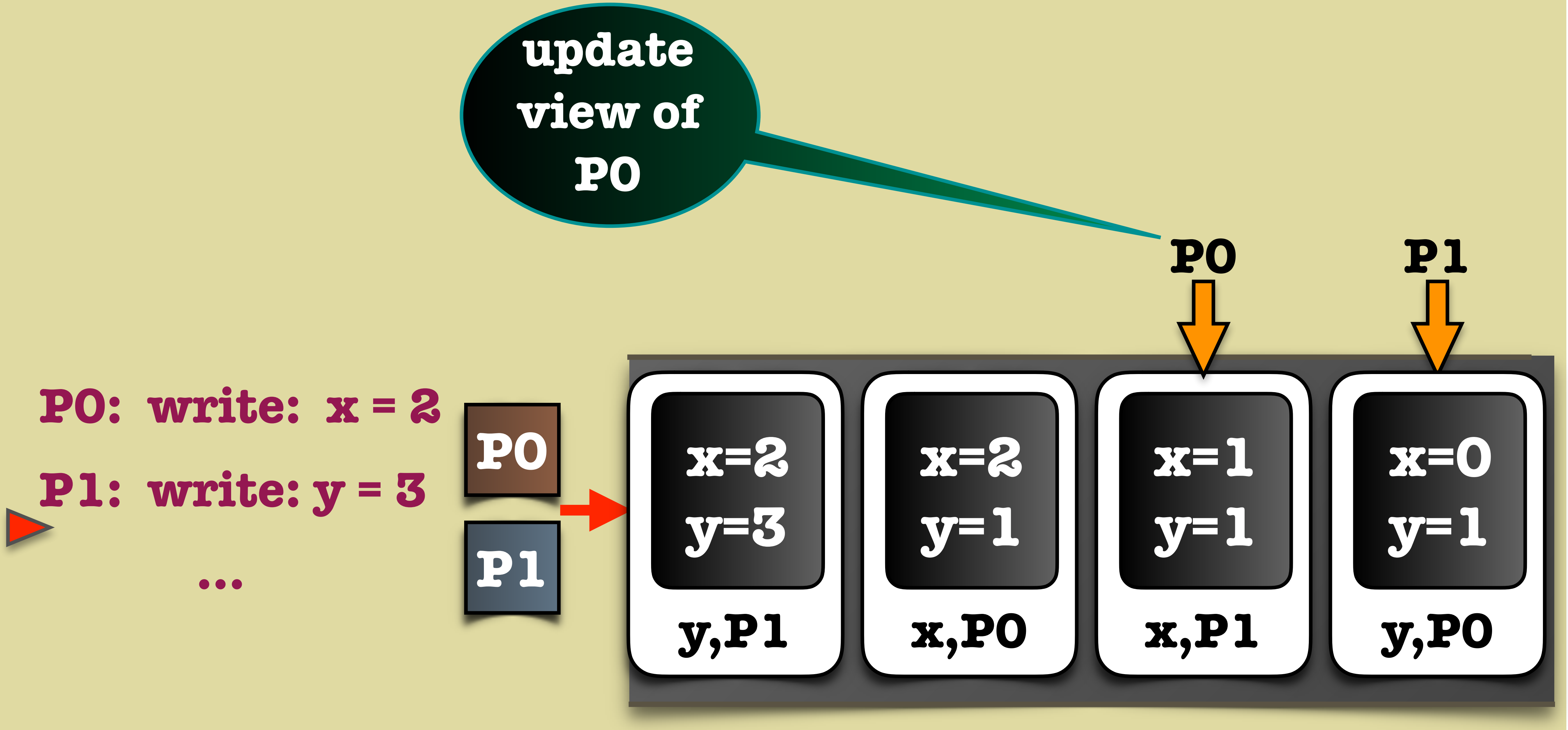


Semantics 2: Single Buffer Model [TACAS'12+13]

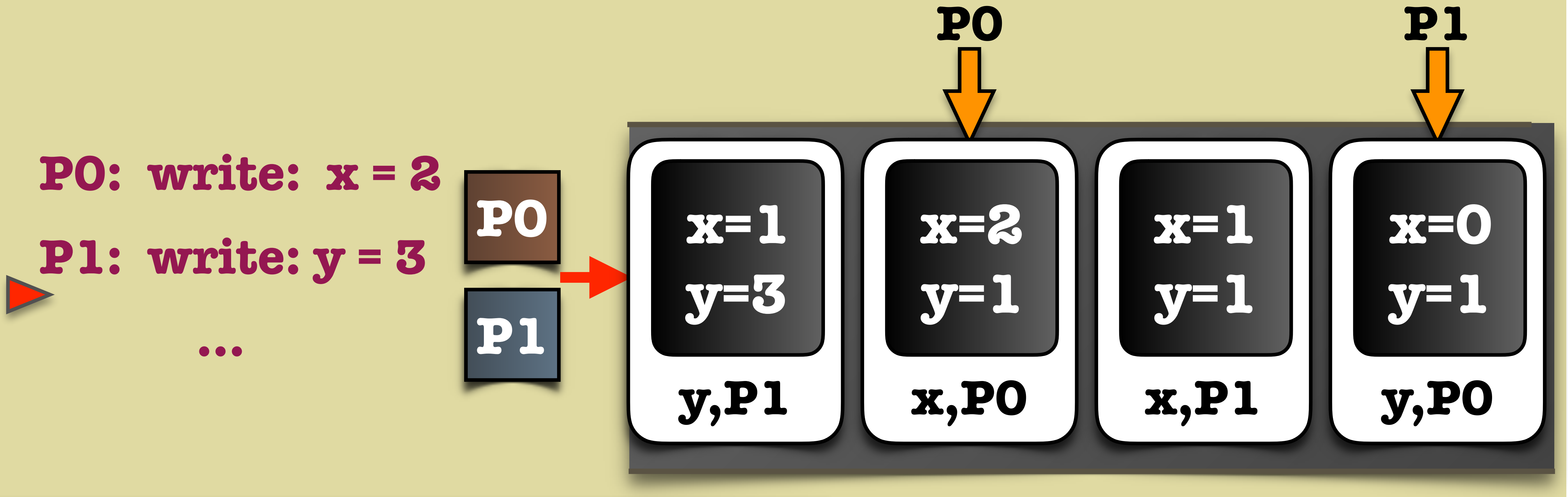
P0: write: x = 2

P1: write: y = 3

...



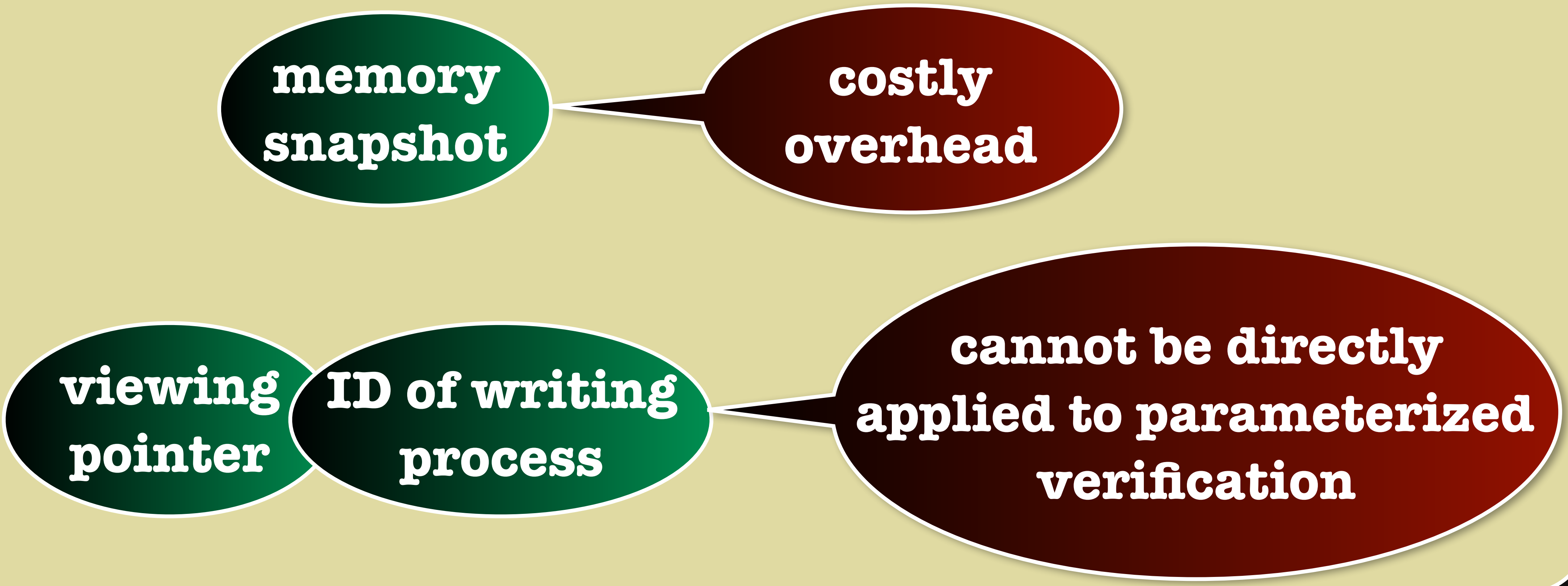
Semantics 2: Single Buffer Model [TACAS'12+13]



equivalent to classical TSO modulo reachability

Sub-word relation on the content of the single buffer is a **monotonic** WQO

Semantics 2: Single Buffer Model [TACAS'12+13]



memory snapshot

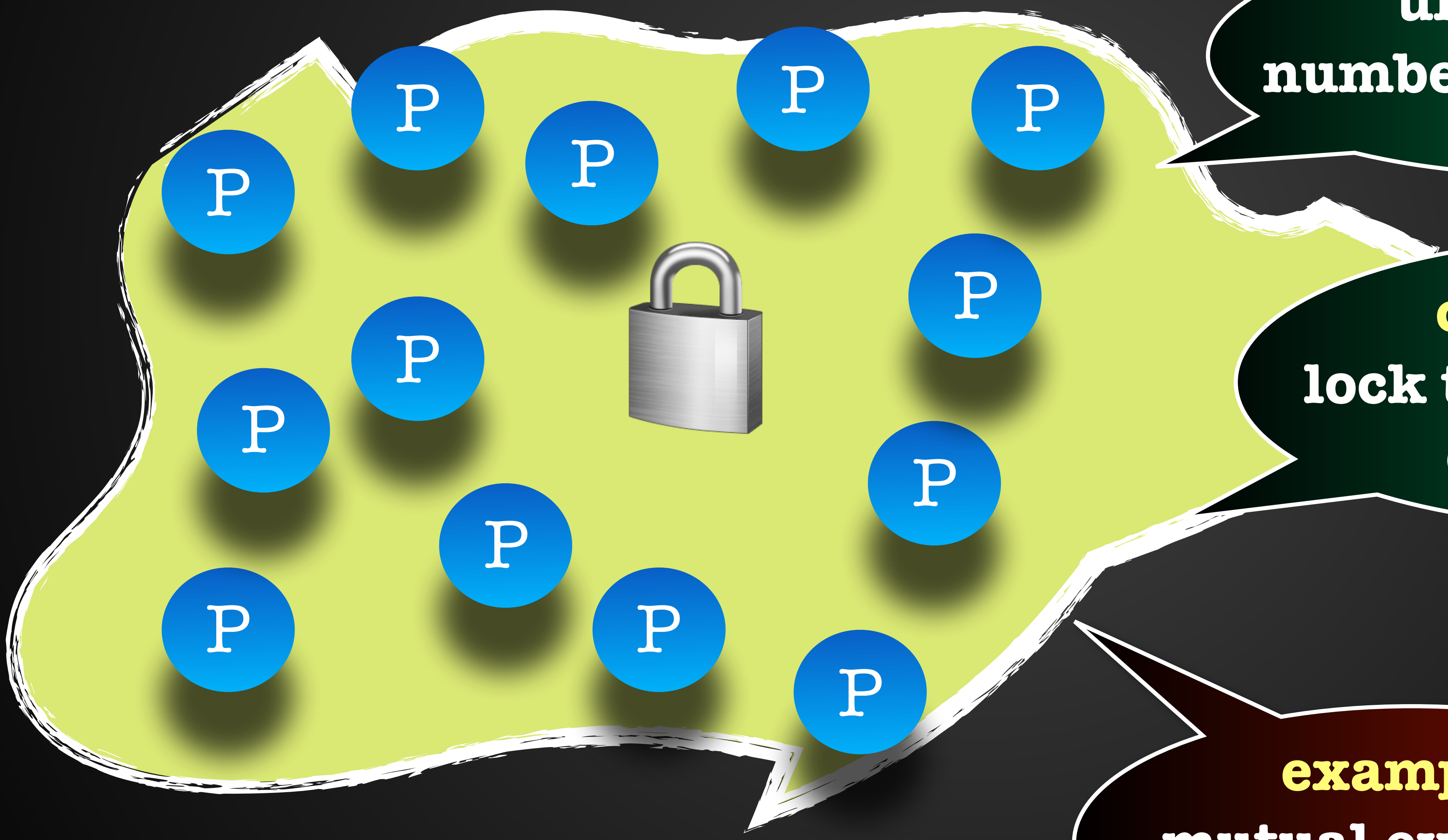
costly overhead

viewing pointer

ID of writing process

cannot be directly applied to parameterized verification

Parameterized Verification



unbounded number of processes

correctness: lock taken by at most one process

example: mutual exclusion protocols

Semantics 3: Dual-TSO

- Store buffers are replaced by load buffers
- Equivalent to classical TSO

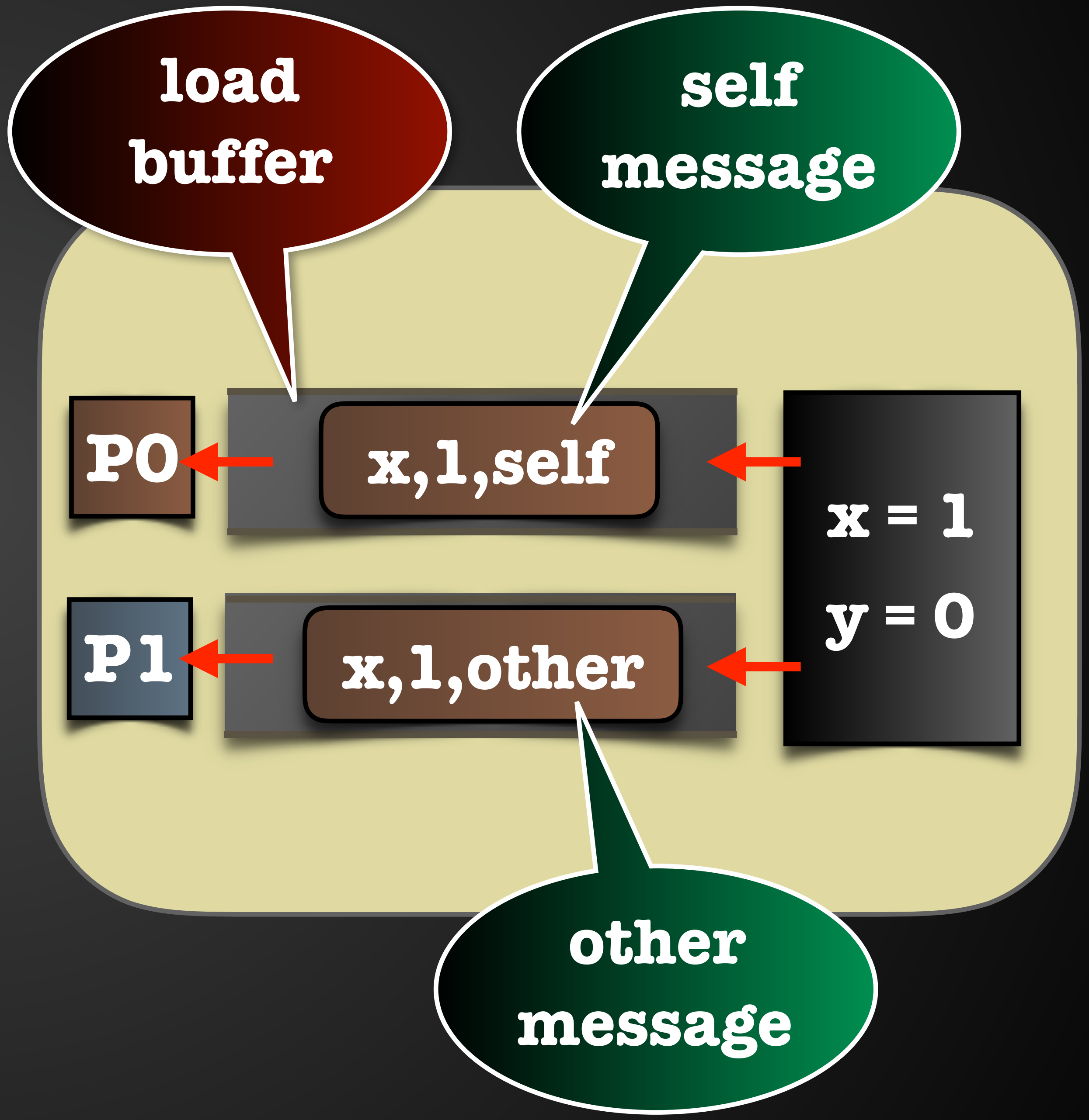
Exact Verification Technique

- Efficient analysis technique based on WQO
- Applicable to parameterized verification

Semantics 3: Dual-TSO

Store Buffers ☛ Load Buffers

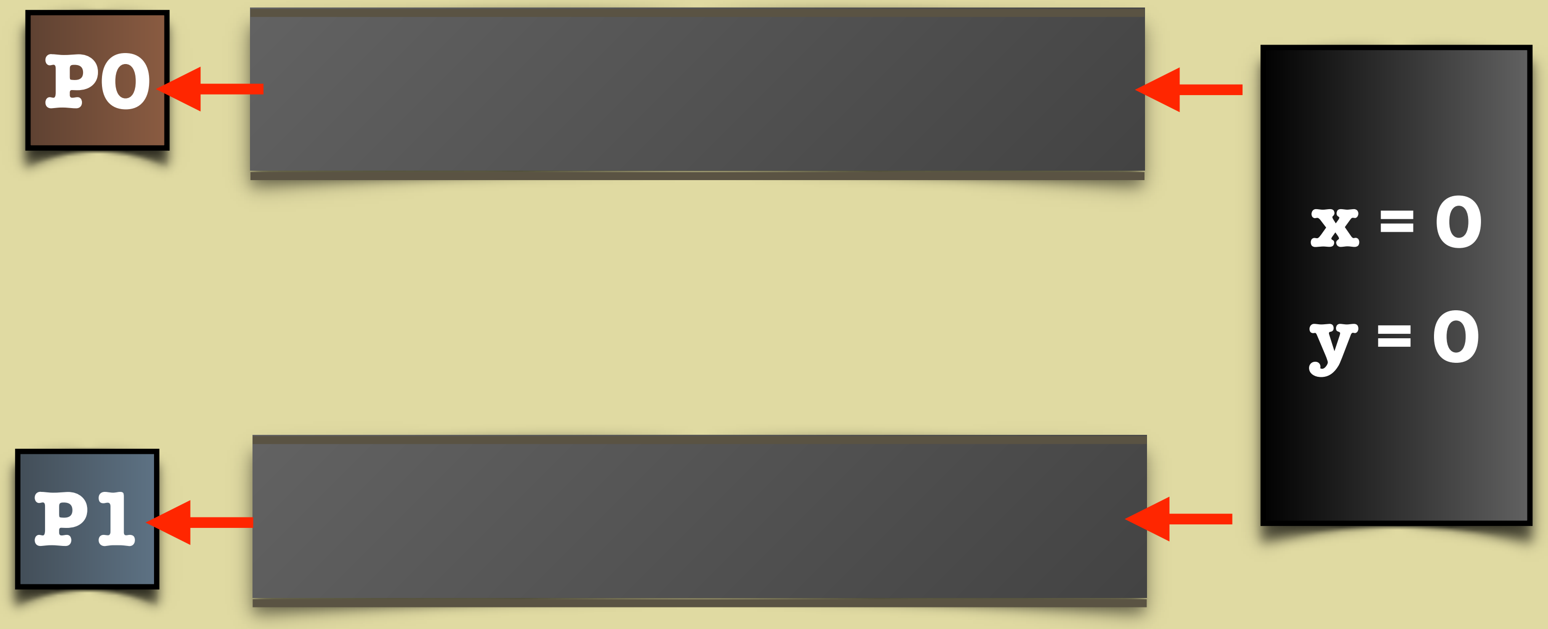
- Write operations immediately update the memory
- Load buffers contain expected read operations



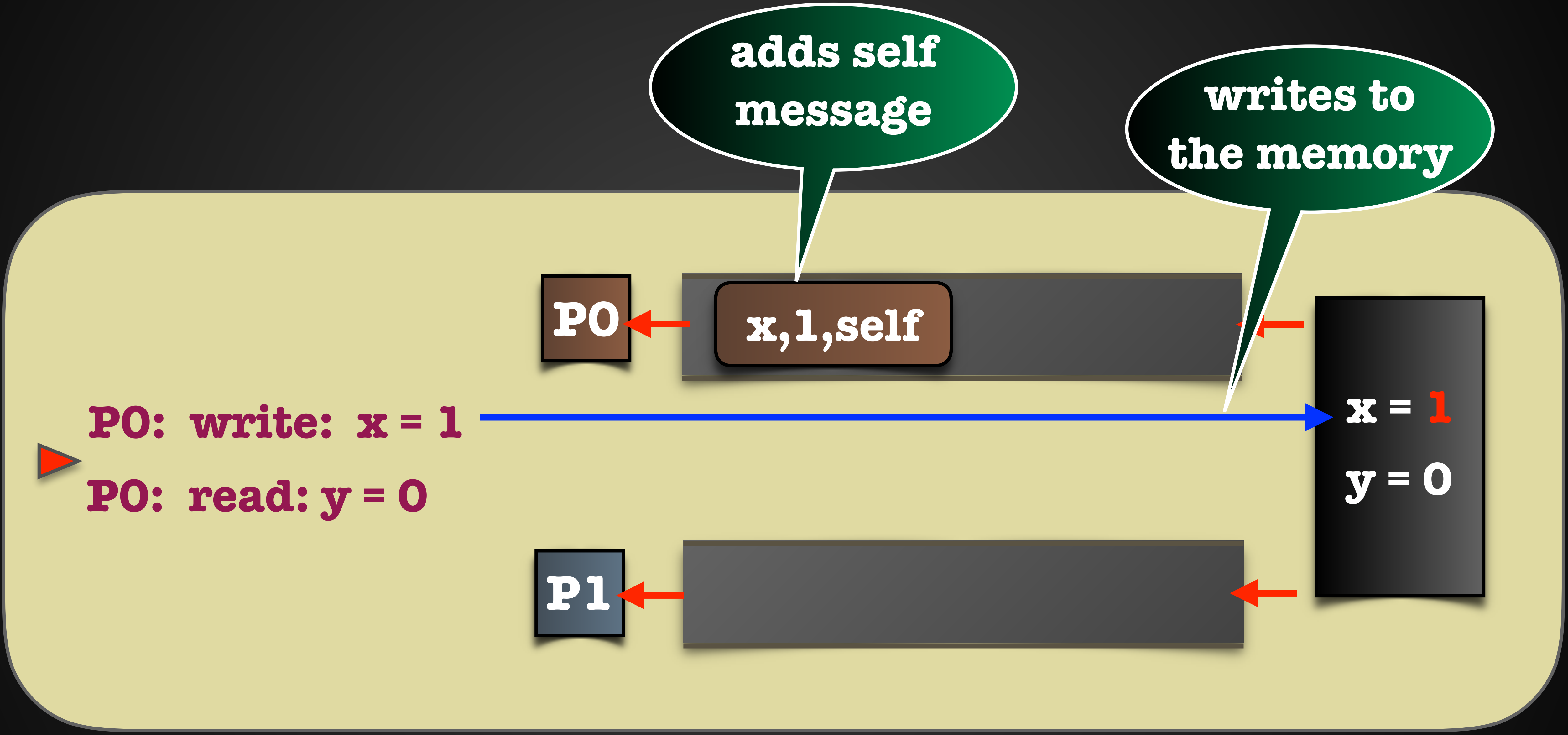
Semantics 3: Dual-TSO

P0: write: x = 1

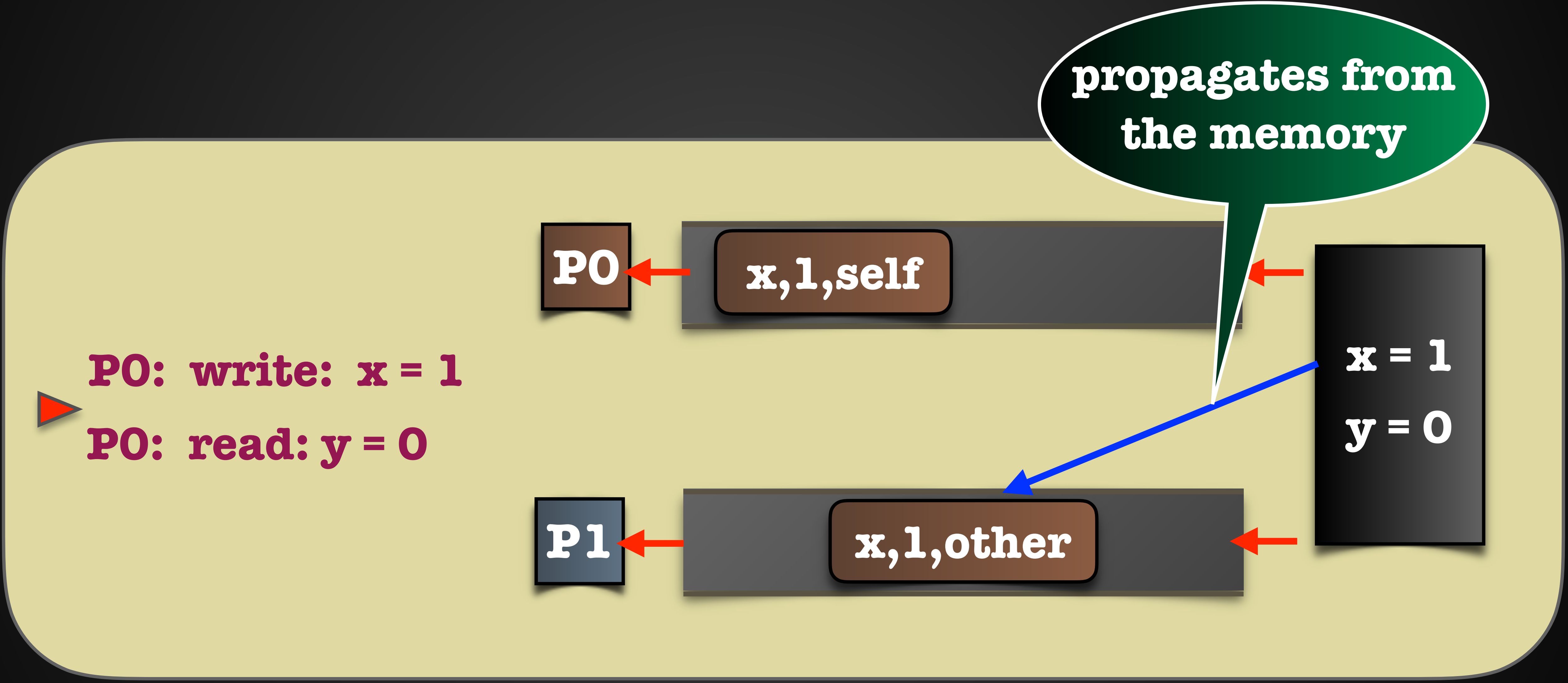
P0: read: y = 0



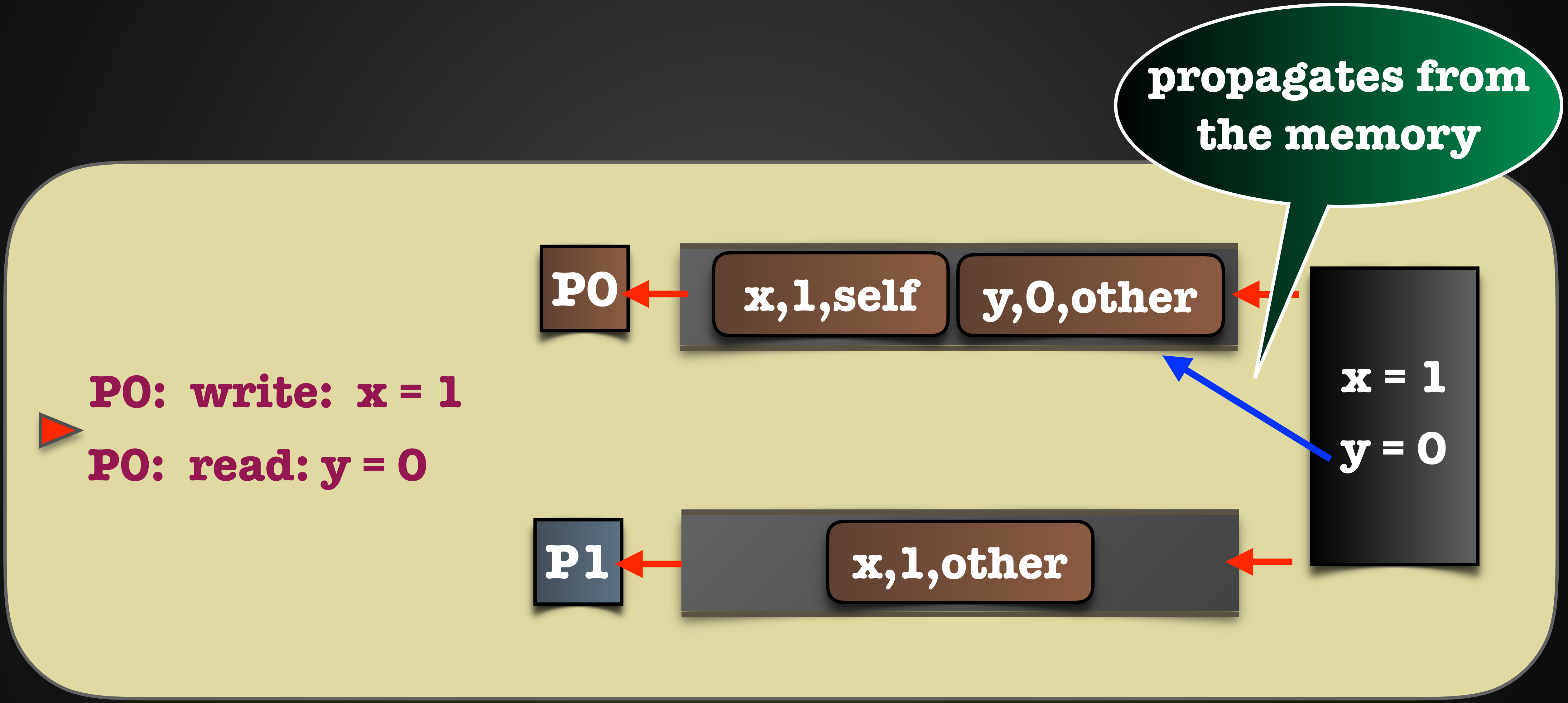
Semantics 3: Dual-TSO



Semantics 3: Dual-TSO



Semantics 3: Dual-TSO

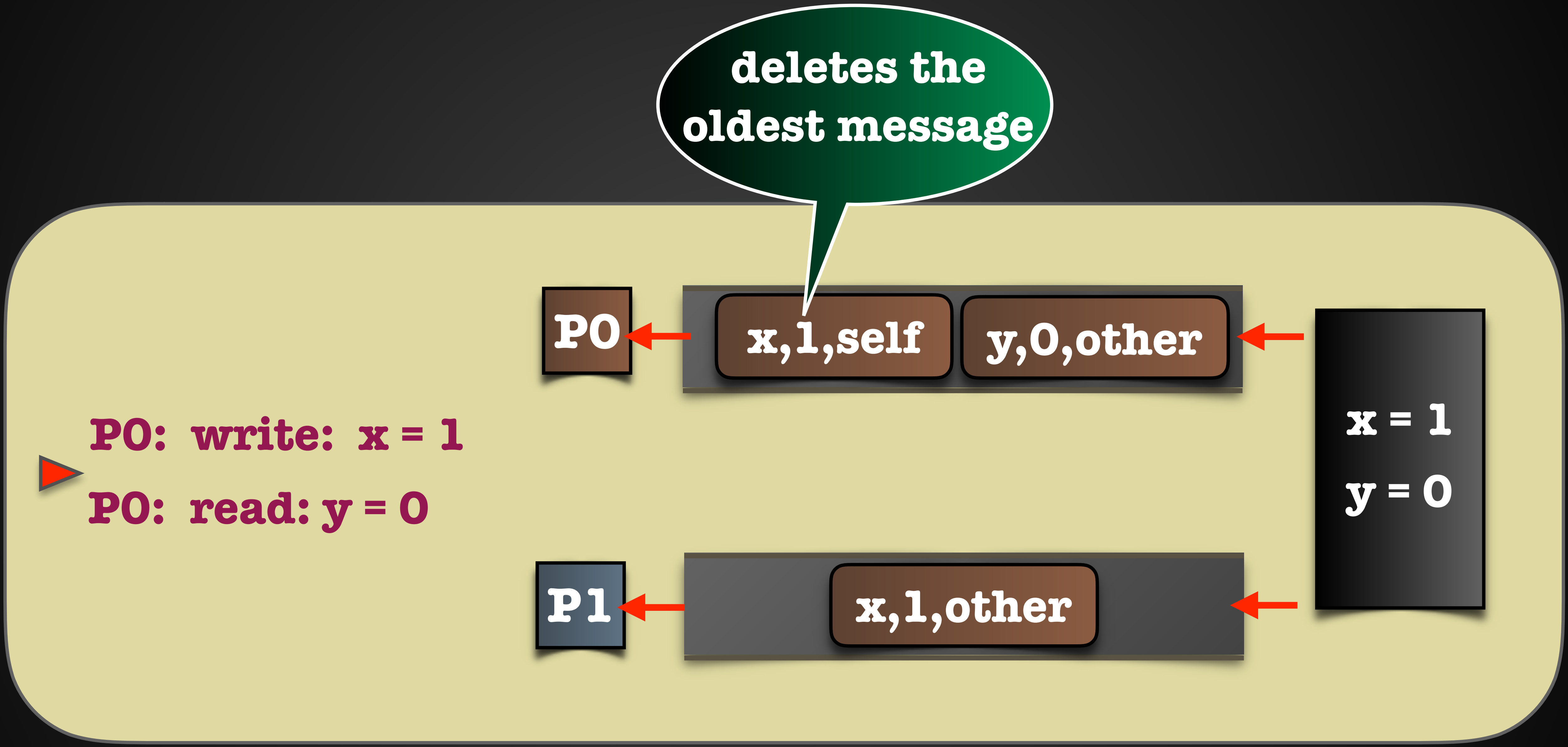


propagates from the memory

P0: write: x = 1

P0: read: y = 0

Semantics 3: Dual-TSO



deletes the oldest message

P0

x,1,self

y,0,other

x = 1

y = 0

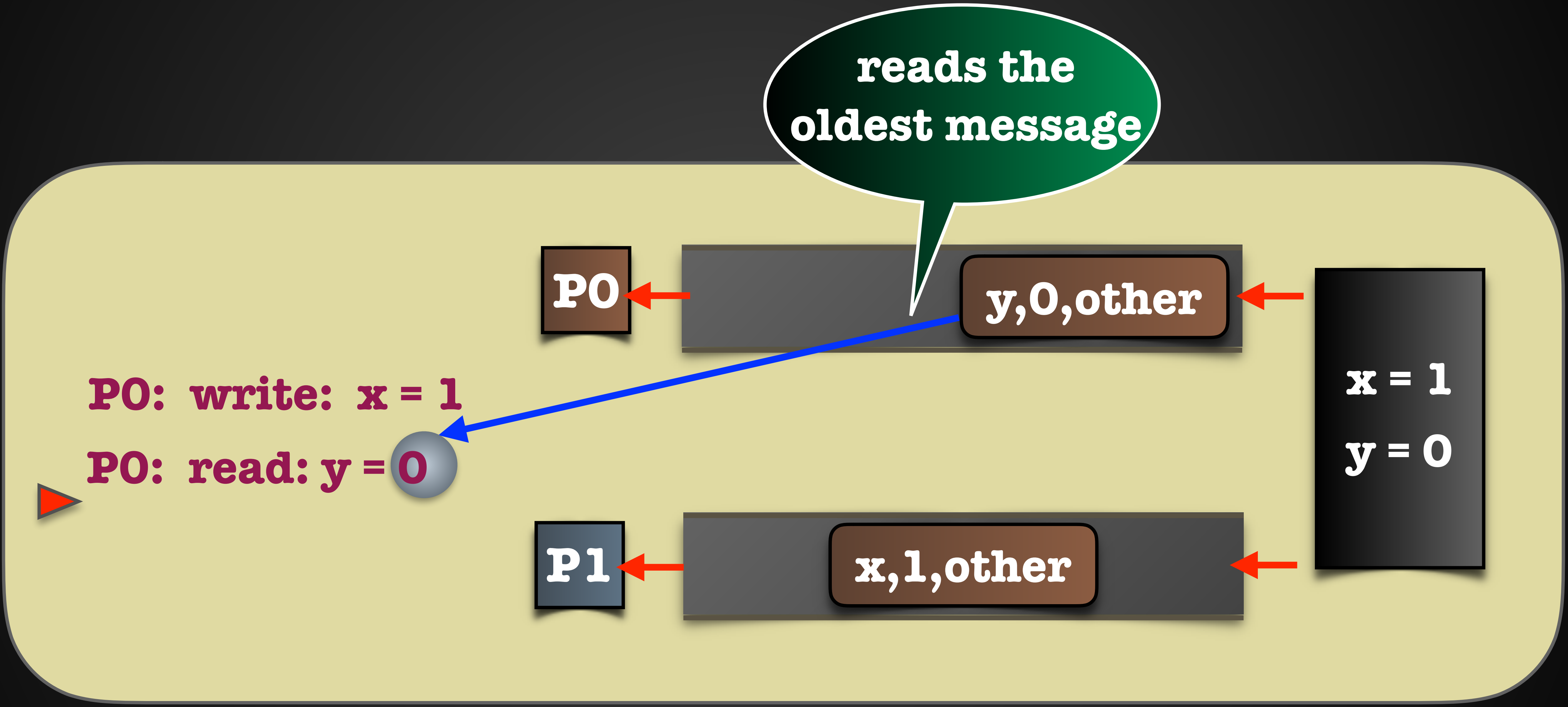
P0: write: x = 1

P0: read: y = 0

P1

x,1,other

Semantics 3: Dual-TSO



Semantics 3: Dual-TSO

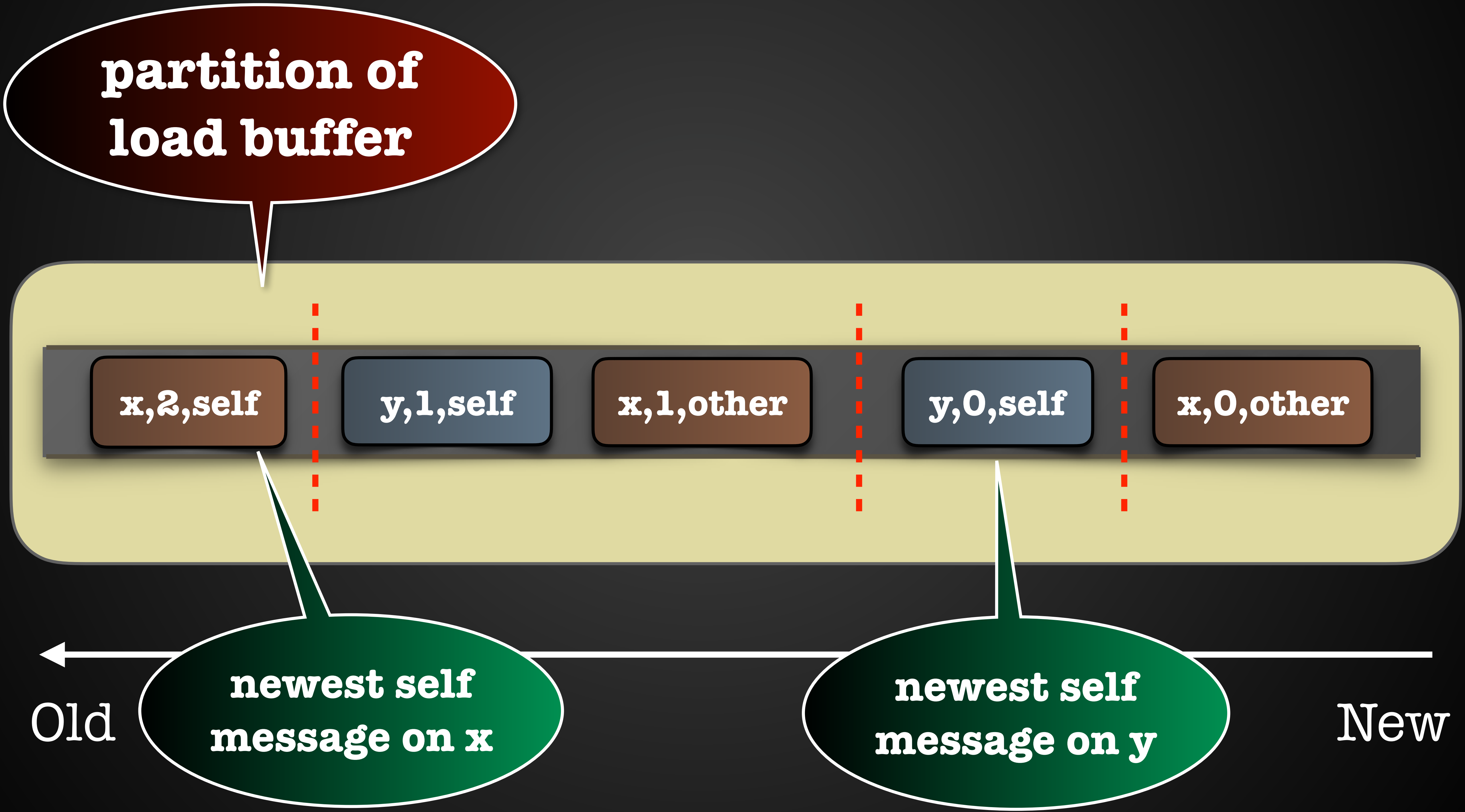
Theorem

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

Outline

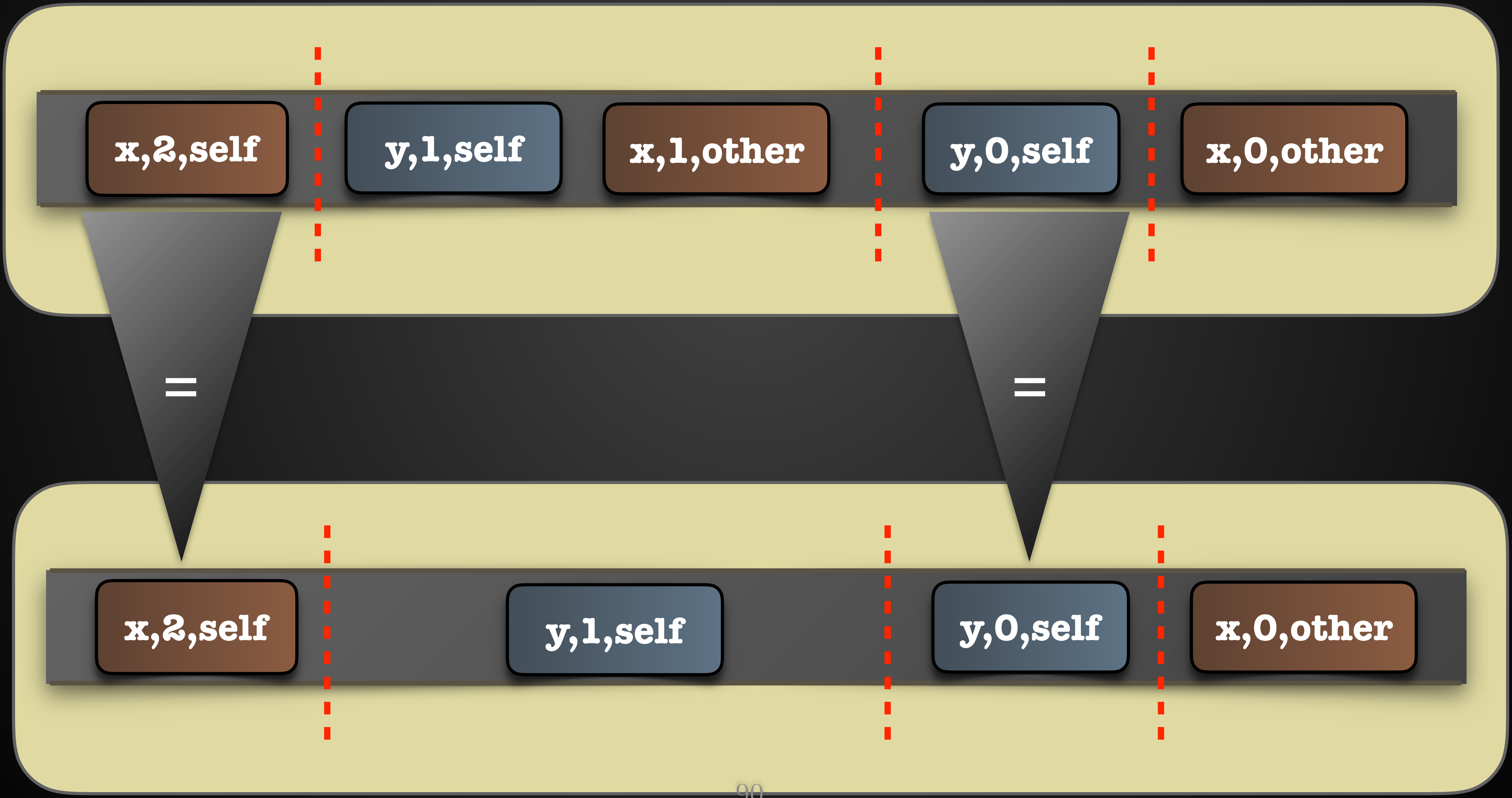
- Classical TSO semantics
- New semantics (Dual-TSO) allows:
  - Efficient verification
  - Parameterised verification
- Verification under Dual-TSO
- Experimental Results
- Conclusions

WQO under Dual-TSO



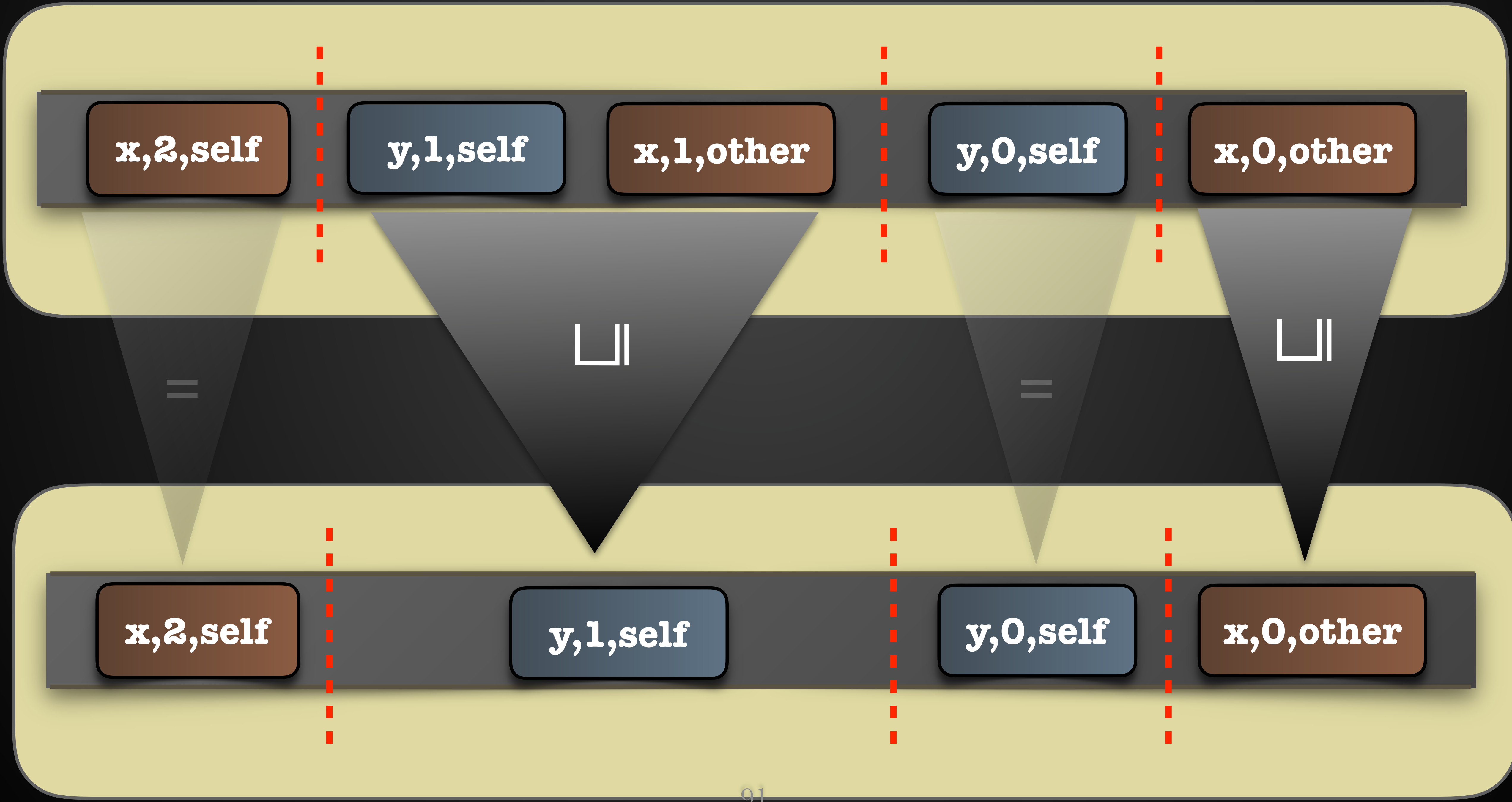
WQO under Dual-TSO

Extension of sub-word ordering



WQO under Dual-TSO

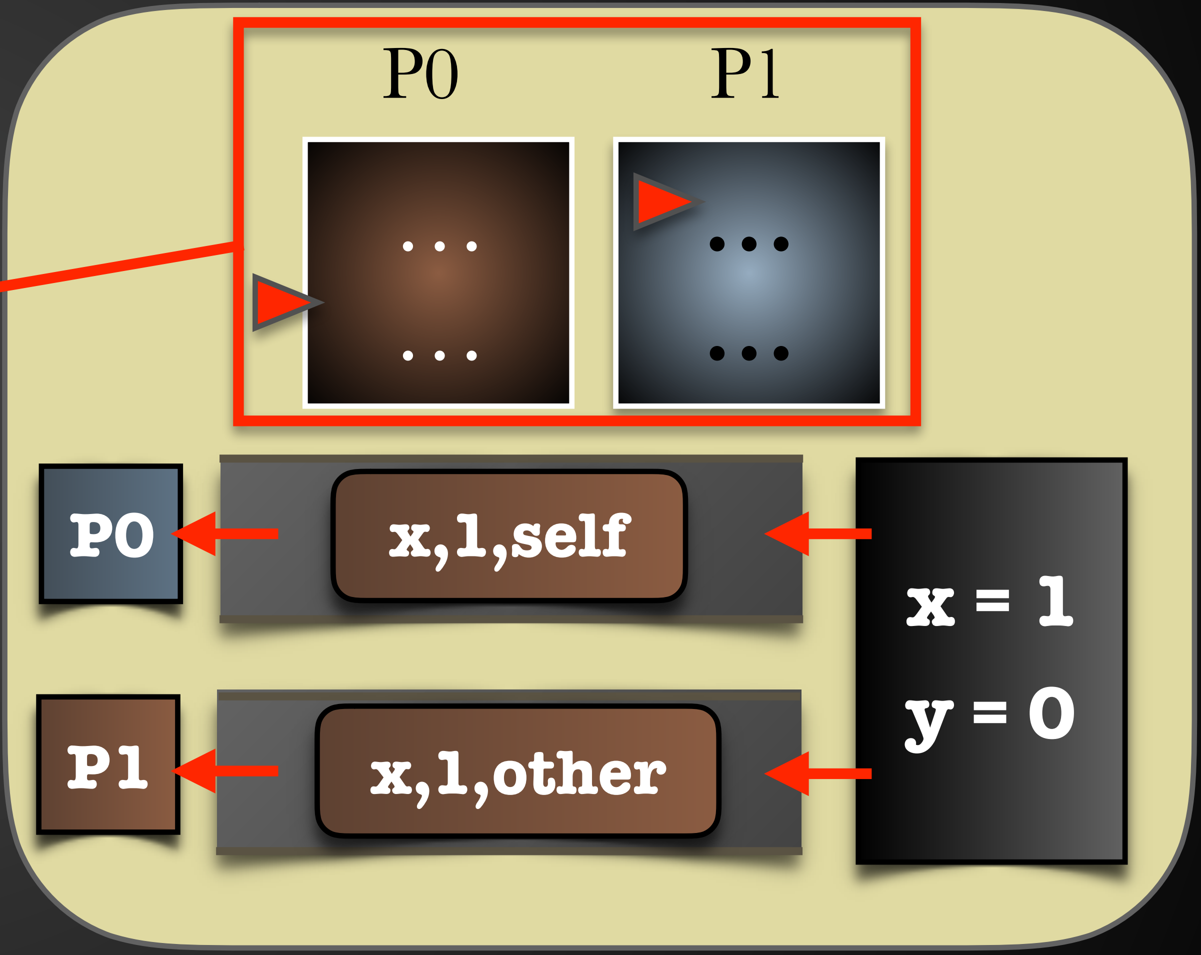
Extension of sub-word ordering



WQO under Dual-TSO

WQO for Dual-TSO

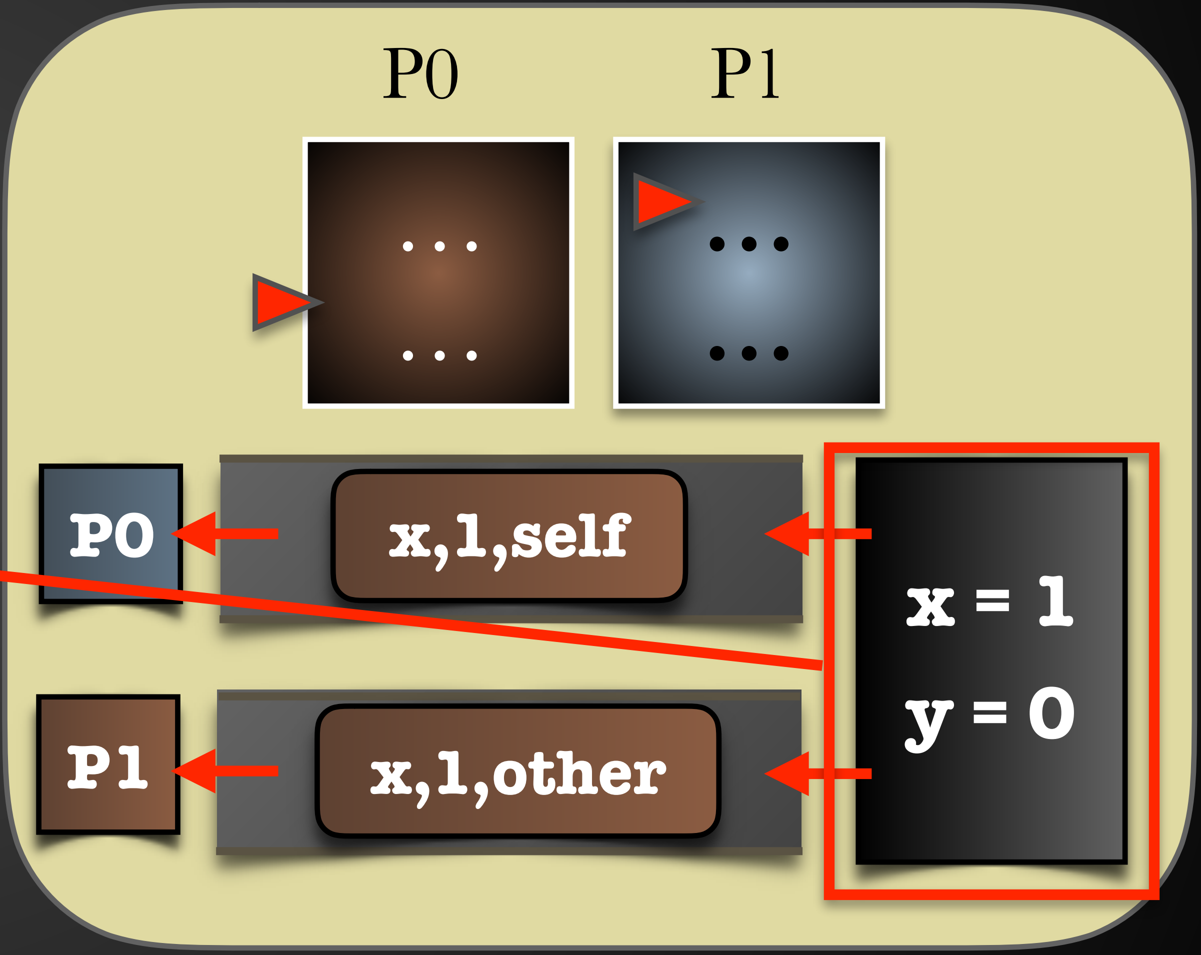
- Same local states of processes
- Same shared memory
- Sub-word relation on load buffers



WQO under Dual-TSO

WQO for Dual-TSO

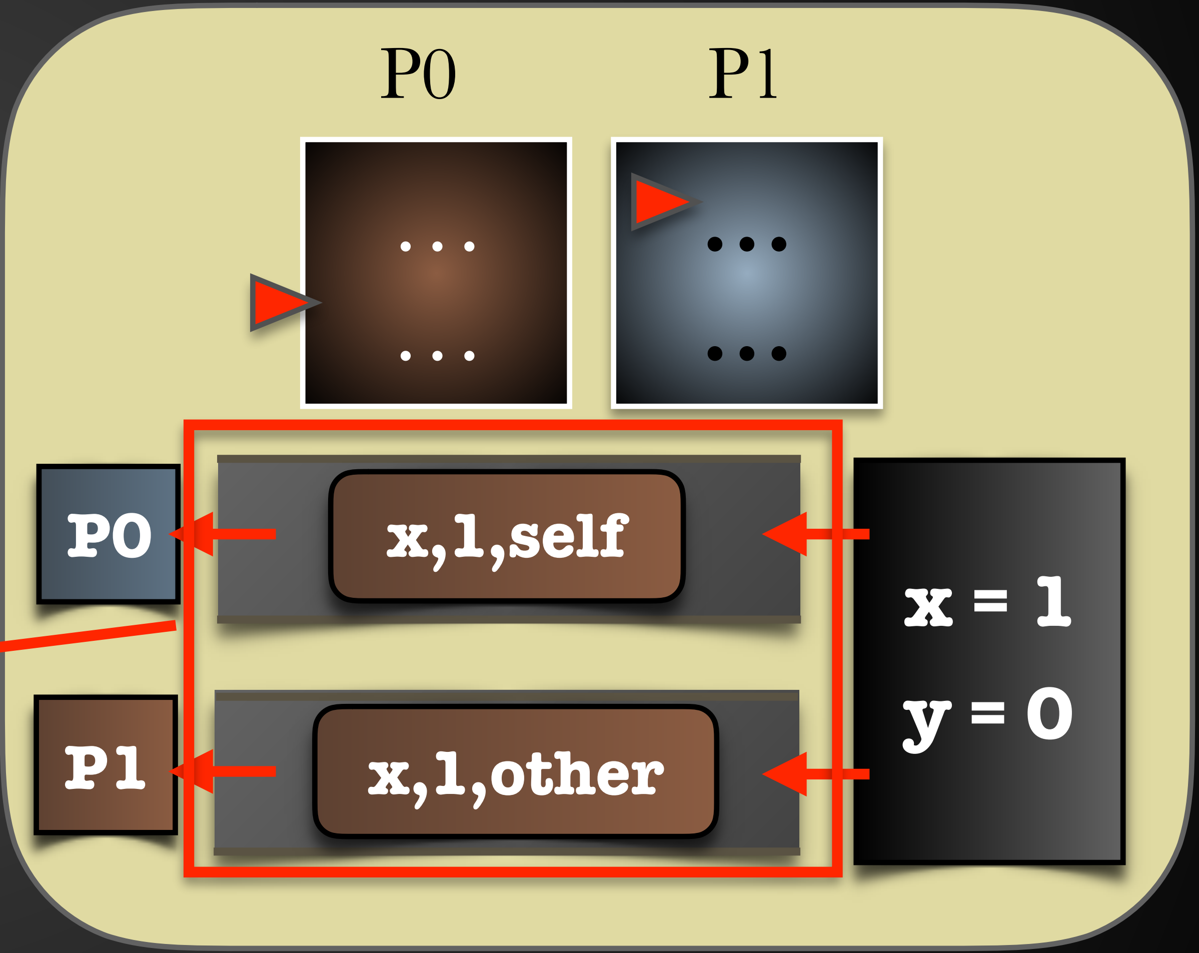
- Same local states of processes
- Same shared memory
- Sub-word relation on load buffers



WQO under Dual-TSO

WQO for Dual-TSO

- Same local states of processes
- Same shared memory
- Sub-word relation on load buffers



Dual-TSO vs Single Buffer

Dual-TSO	Single Buffer
NO memory snapshot	**Need** memory snapshot
No viewing pointer, ID of process	**Need** viewing pointers, IDs of processes
Several channels: one channel per process	**Only one** channel
have **read** ns	Buffers have **write** operations

efficient

can be applied to parameterised verification

Outline

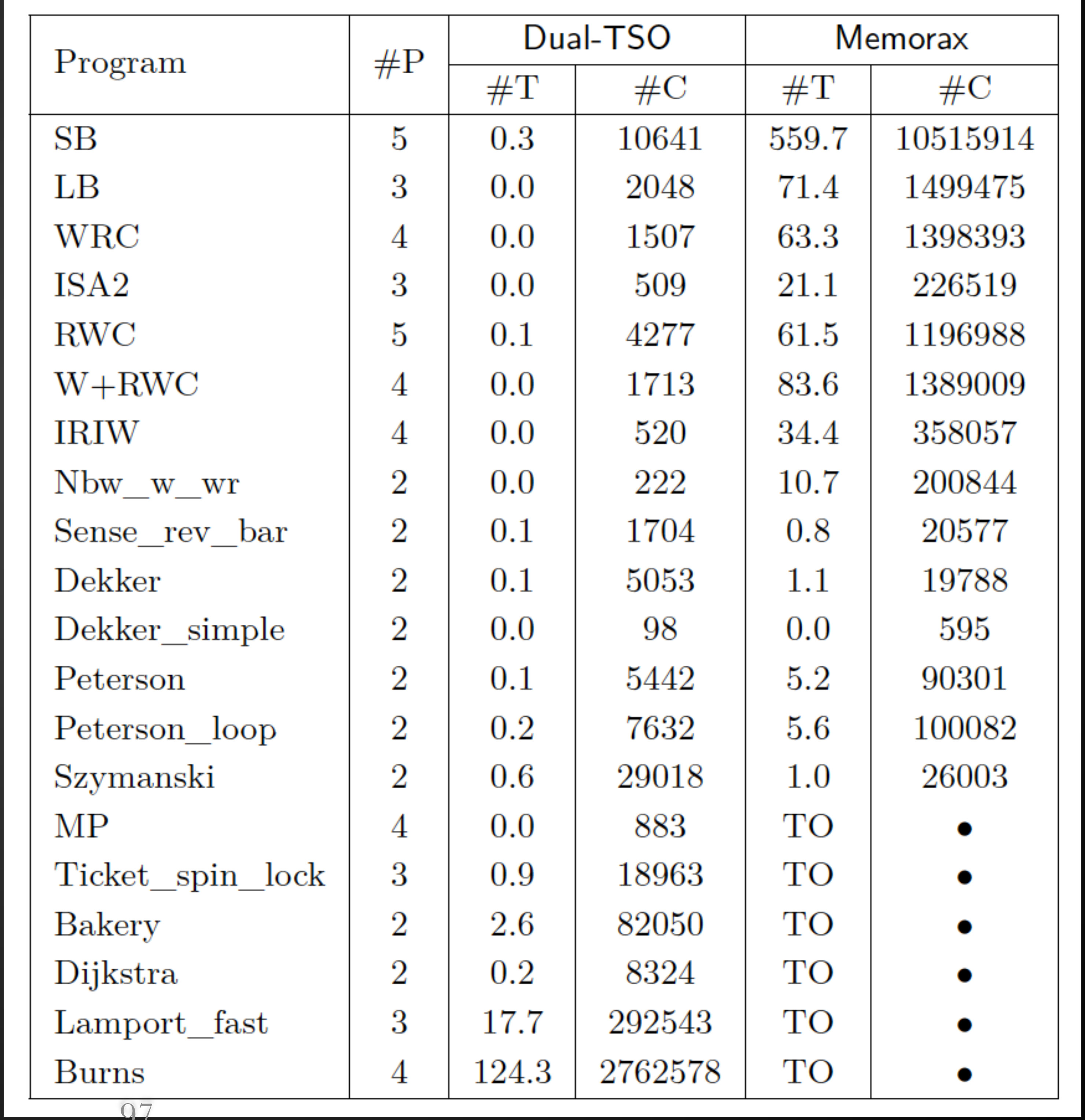
- Classical TSO semantics
- New semantics (Dual-TSO) allows:
  - Efficient verification
  - Parameterised verification
- Verification under Dual-TSO
- Experimental Results
- Conclusions

Experimental Results

Single buffer approach (exact method [TACAS12+13])

Dual-TSO vs Memorax

- **Running time**
- **Memory consumption**



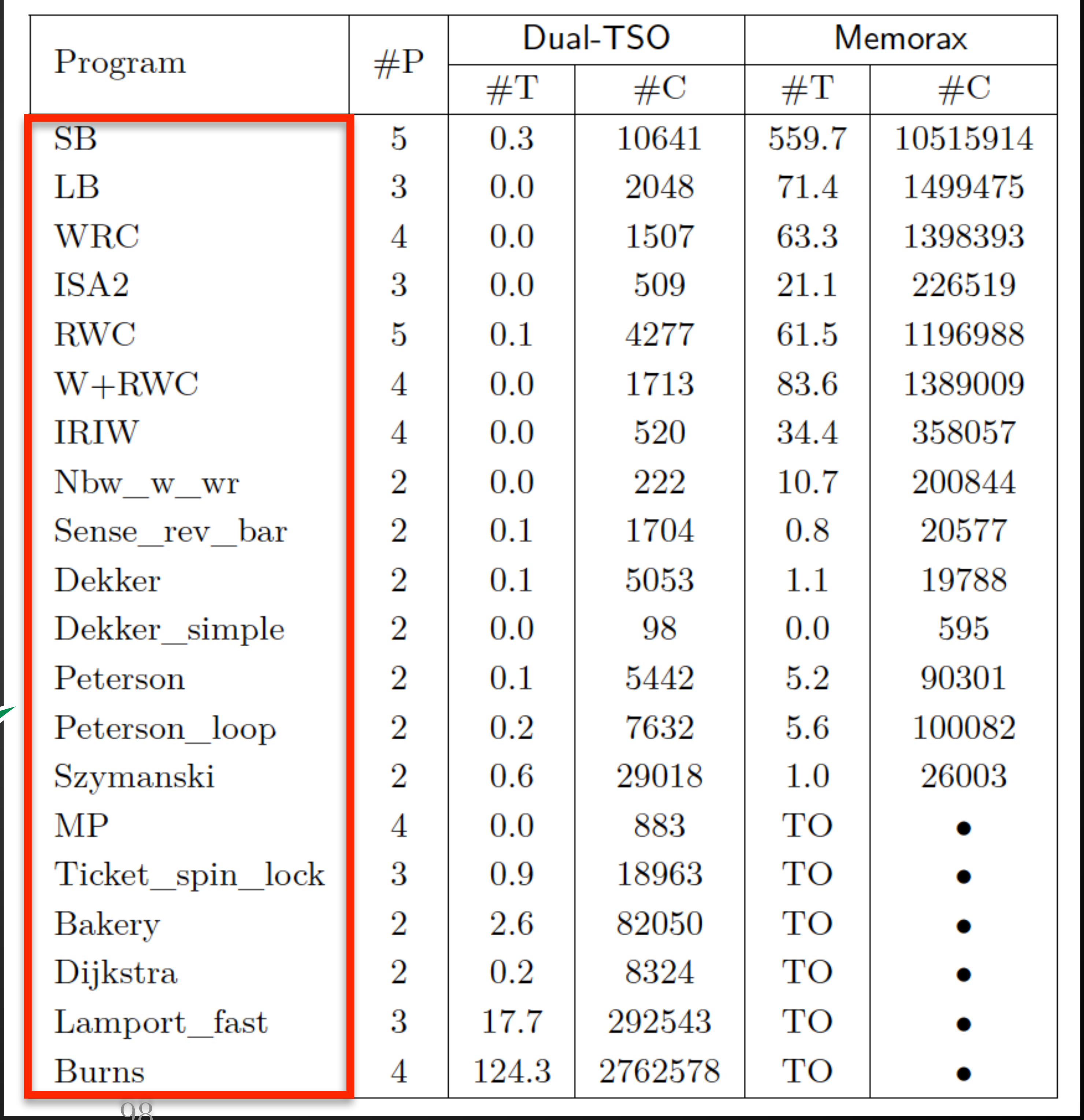
Program	#P	Dual-TSO		Memorax	
		#T	#C	#T	#C
SB	5	0.3	10641	559.7	10515914
LB	3	0.0	2048	71.4	1499475
WRC	4	0.0	1507	63.3	1398393
ISA2	3	0.0	509	21.1	226519
RWC	5	0.1	4277	61.5	1196988
W+RWC	4	0.0	1713	83.6	1389009
IRIW	4	0.0	520	34.4	358057
Nbw_w_wr	2	0.0	222	10.7	200844
Sense_rev_bar	2	0.1	1704	0.8	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	•

https://www.it.uu.se/katalog/tuang296/dual-tso

Experimental Results

Dual-TSO vs Memorax

- **Running time**
- **Memory consumption**



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	●

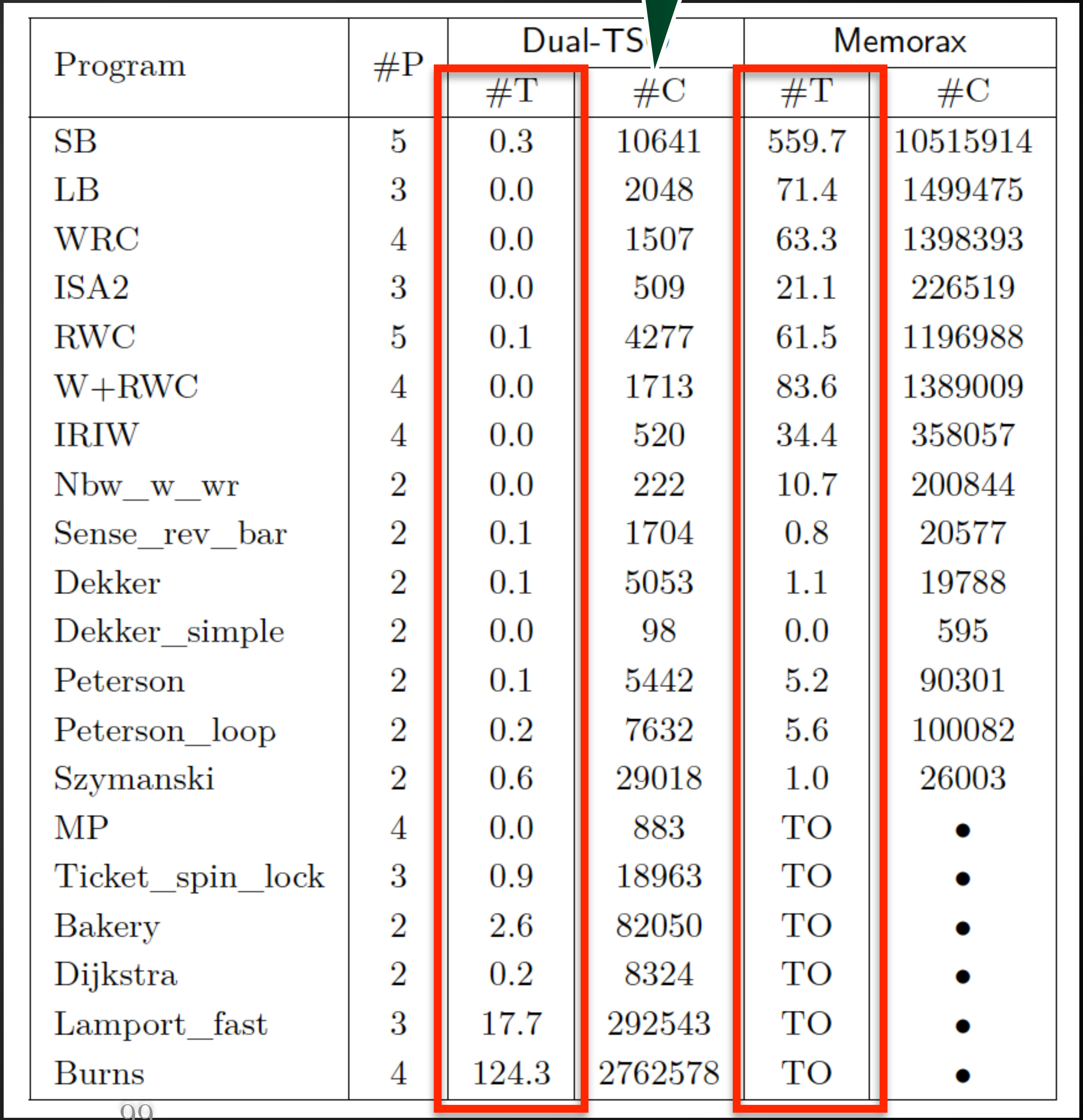
standard benchmarks: litmus tests and mutual algorithms

Experimental R

running time
in seconds

Dual-TSO vs Memorax

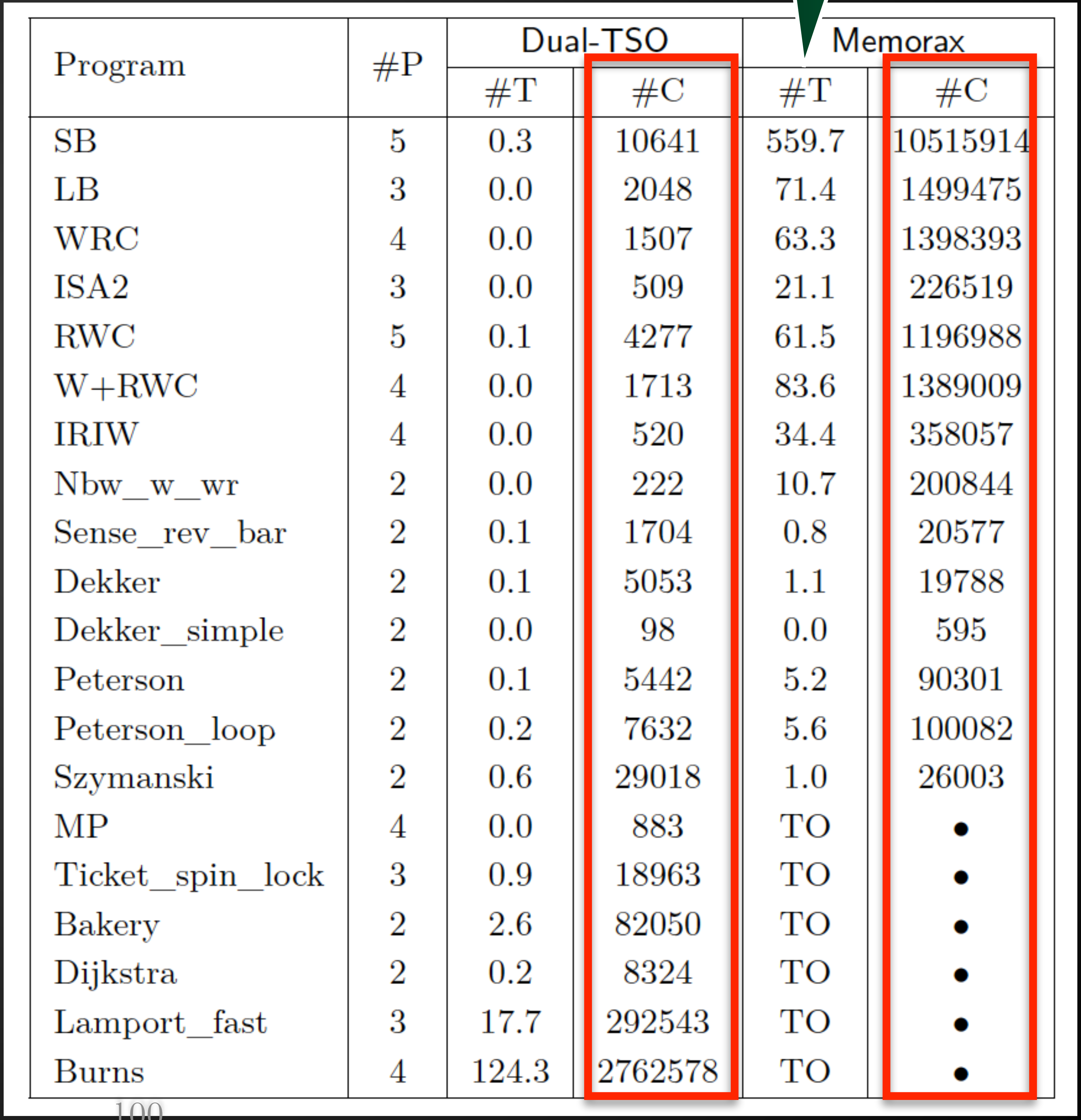
- **Running time**
- **Memory consumption**



Program	#P	Dual-TS		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 Res

generated configurations



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	•

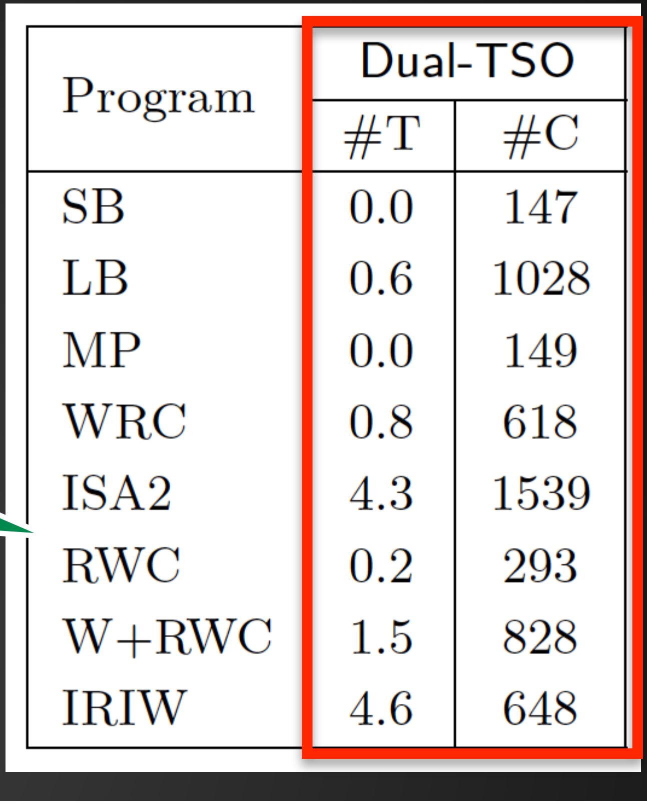
Dual-TSO vs Memorax

- **Running time**
- **Memory consumption**

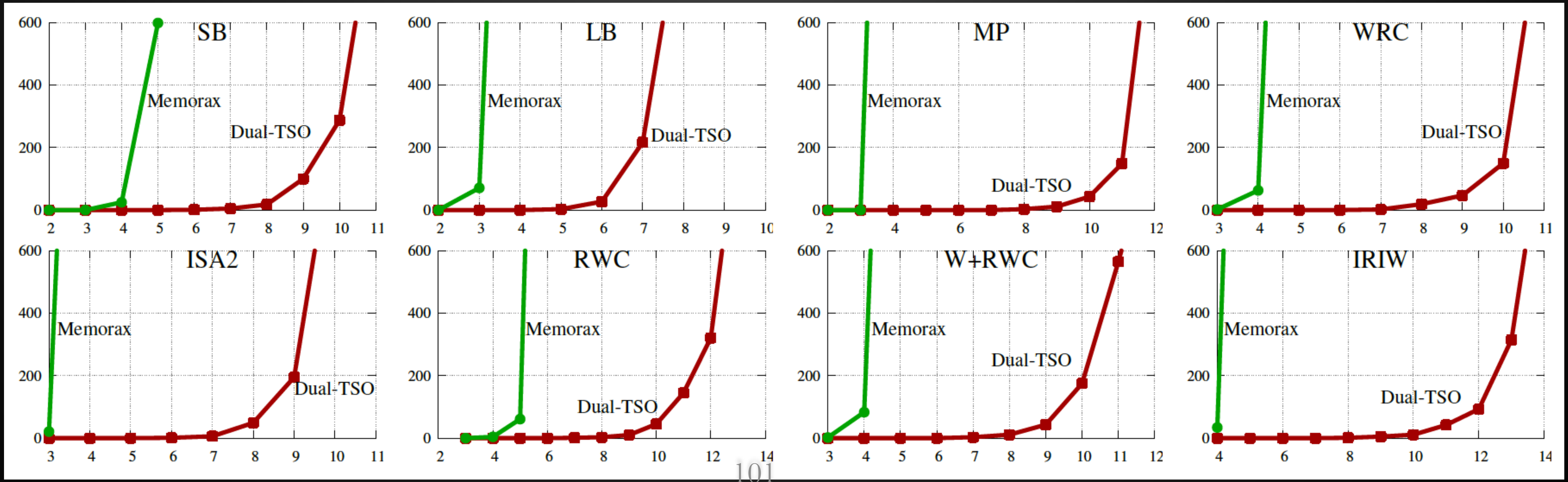
Dual-TSO is faster and uses less memory in most of examples

Experimental Results
Parameterised Cases

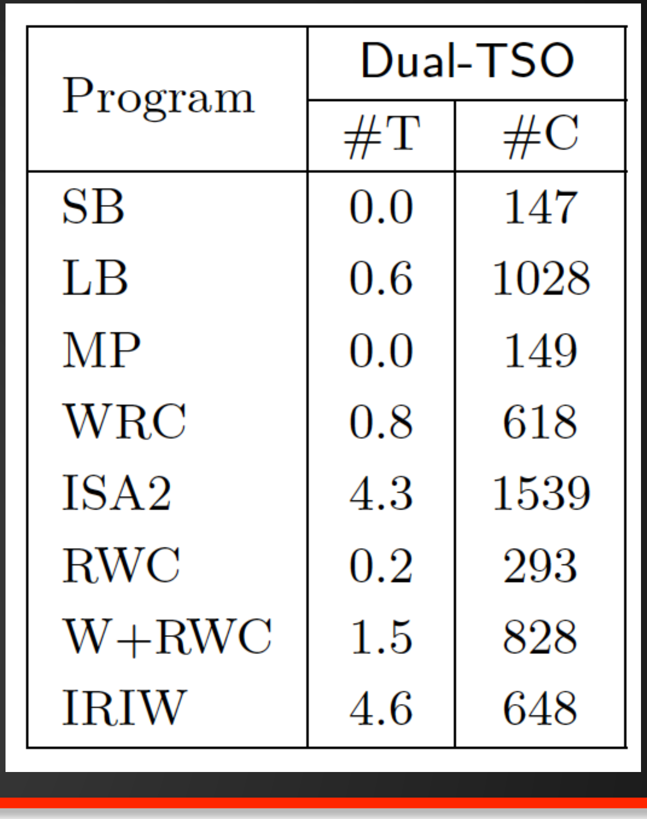
unbounded
number of processes



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

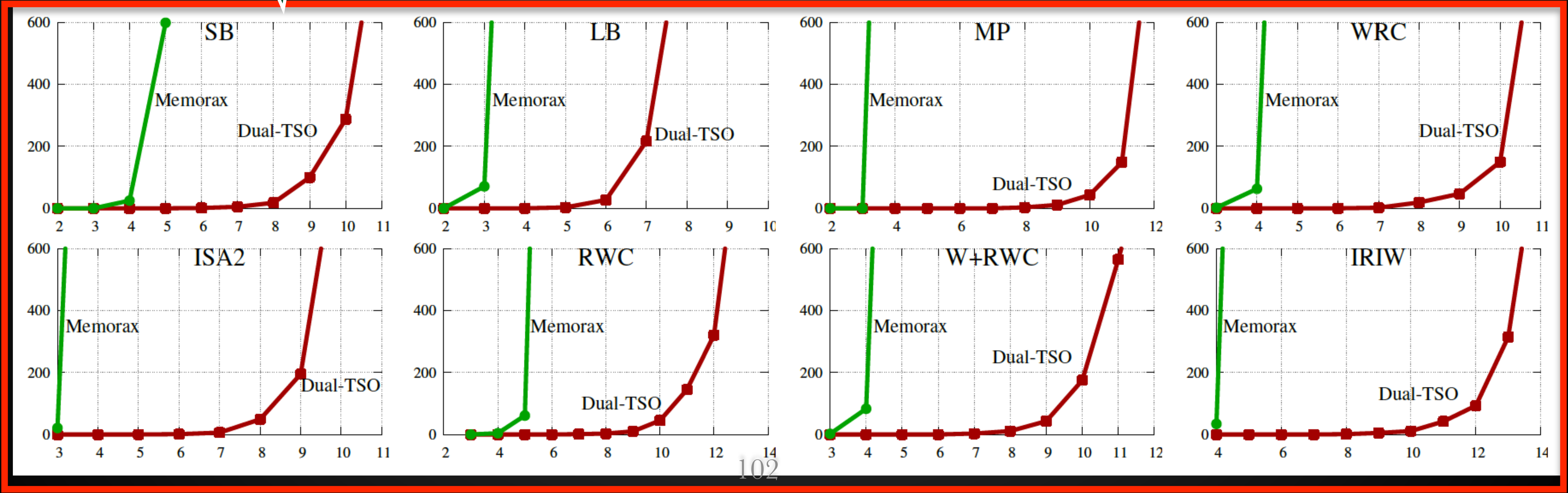


Experimental Results
Parameterised Cases



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

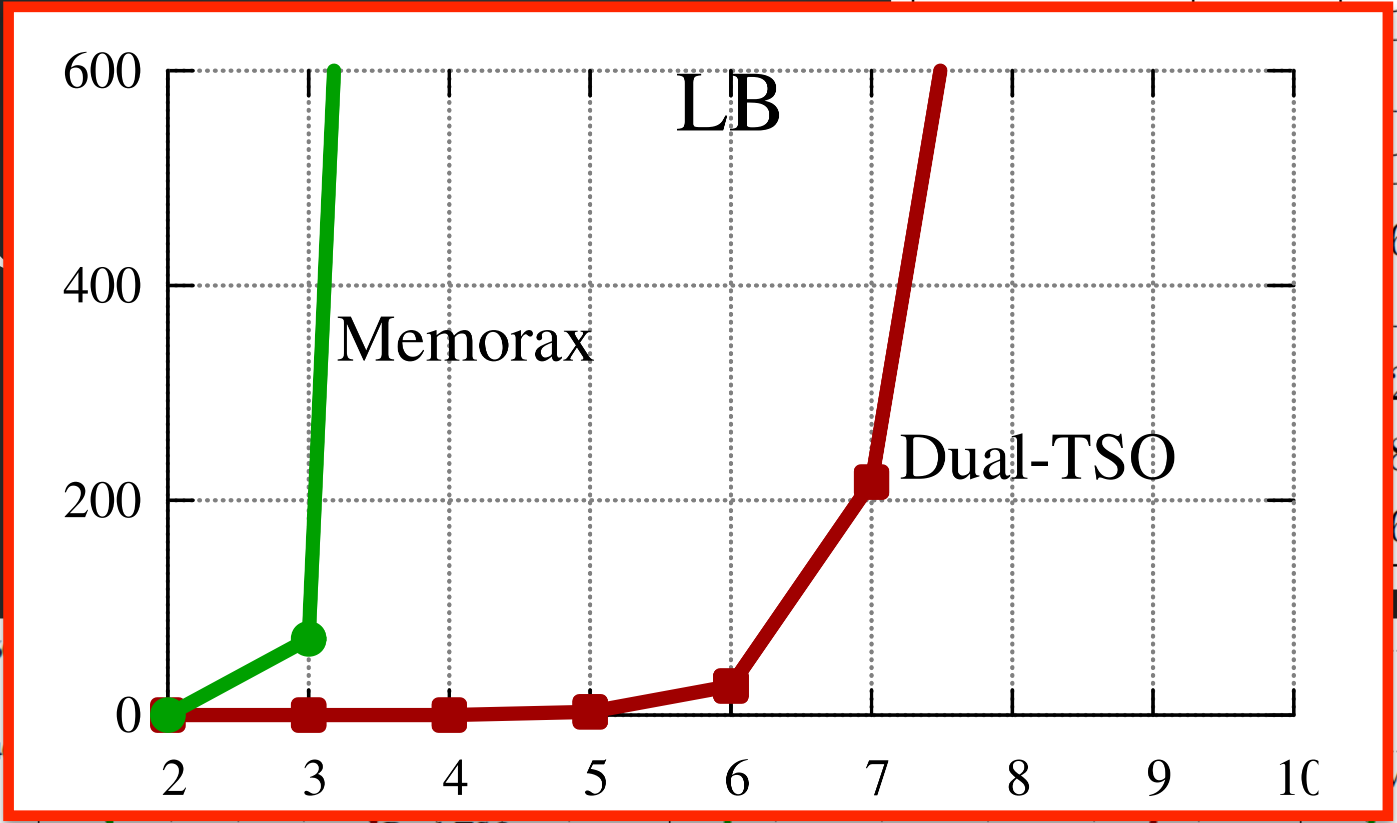
increasing the number of processes

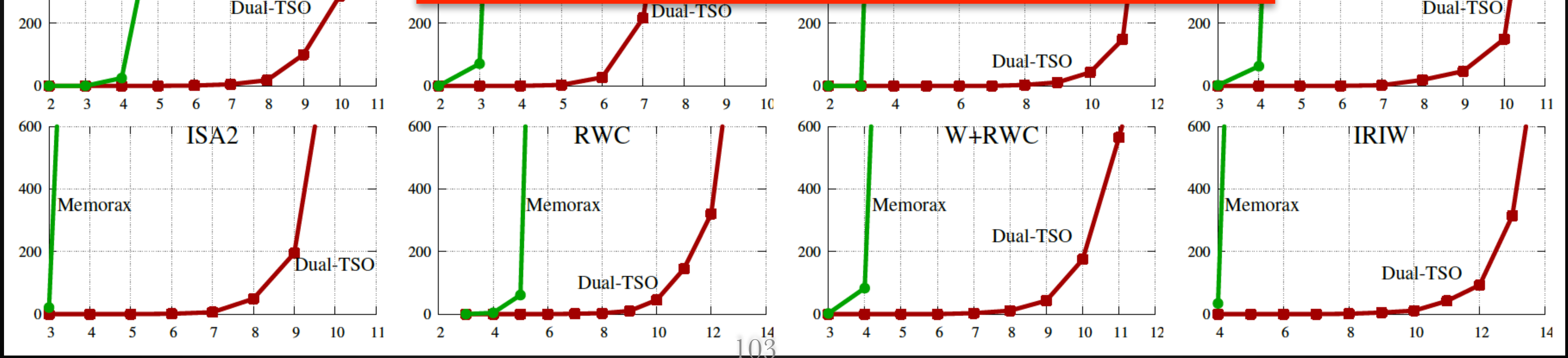


Experimental Results
Parameterised Cases

Dual-TSO is more scalable

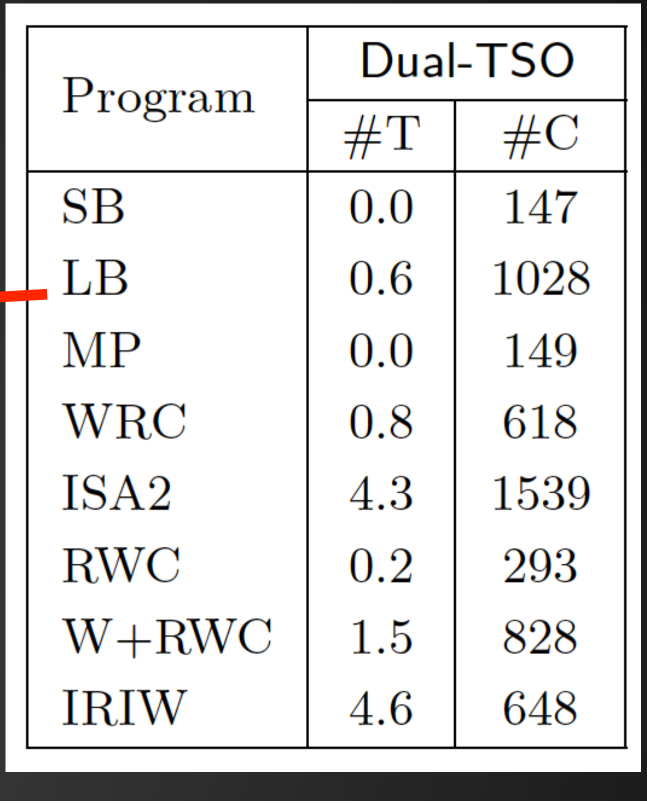
Program	Dual-TSO	
	#T	#C
		147
		028
		149
		618
		539
		293
		828
		648



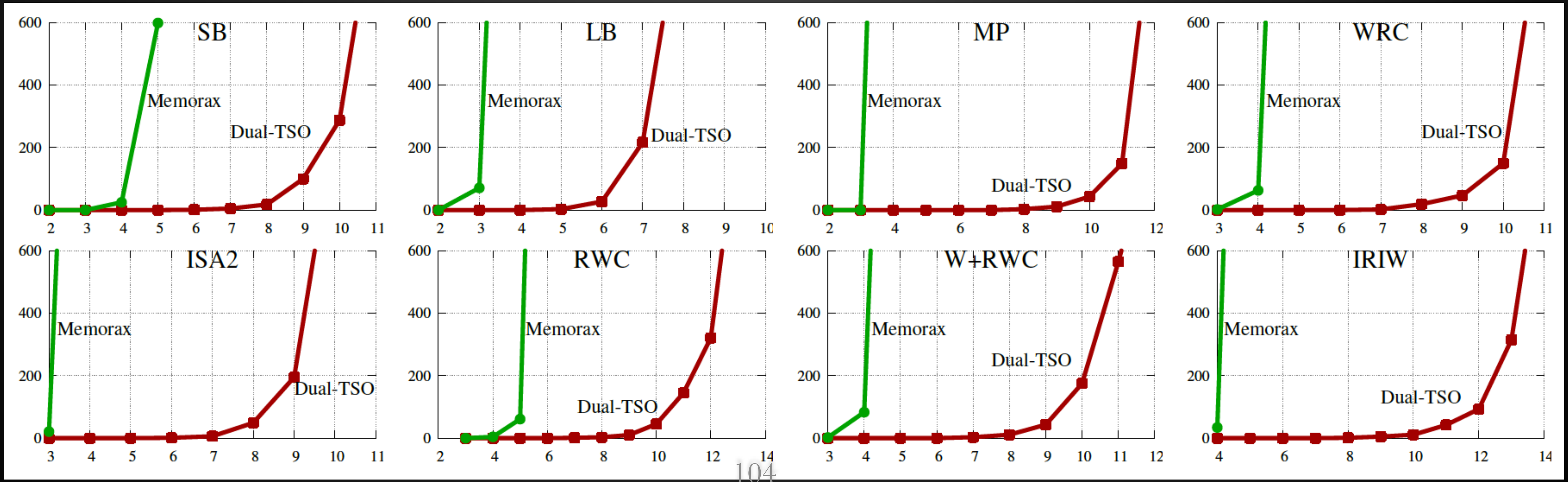


Experimental Results
Parameterised Cases

Dual-TSO is more efficient and scalable



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



Summary

Dual-TSO Model

- Exact (parameterised) reachability method:
  - Dual-TSO: Load buffers instead of store buffers
  - Using well quasi-ordering framework:
    - Efficient verification
    - Parameterized verification
- Prototype implementation

Future Work

Possible Extension

- Infinite data domain: predicate abstraction
- Apply to more memory models: e.g. PSO

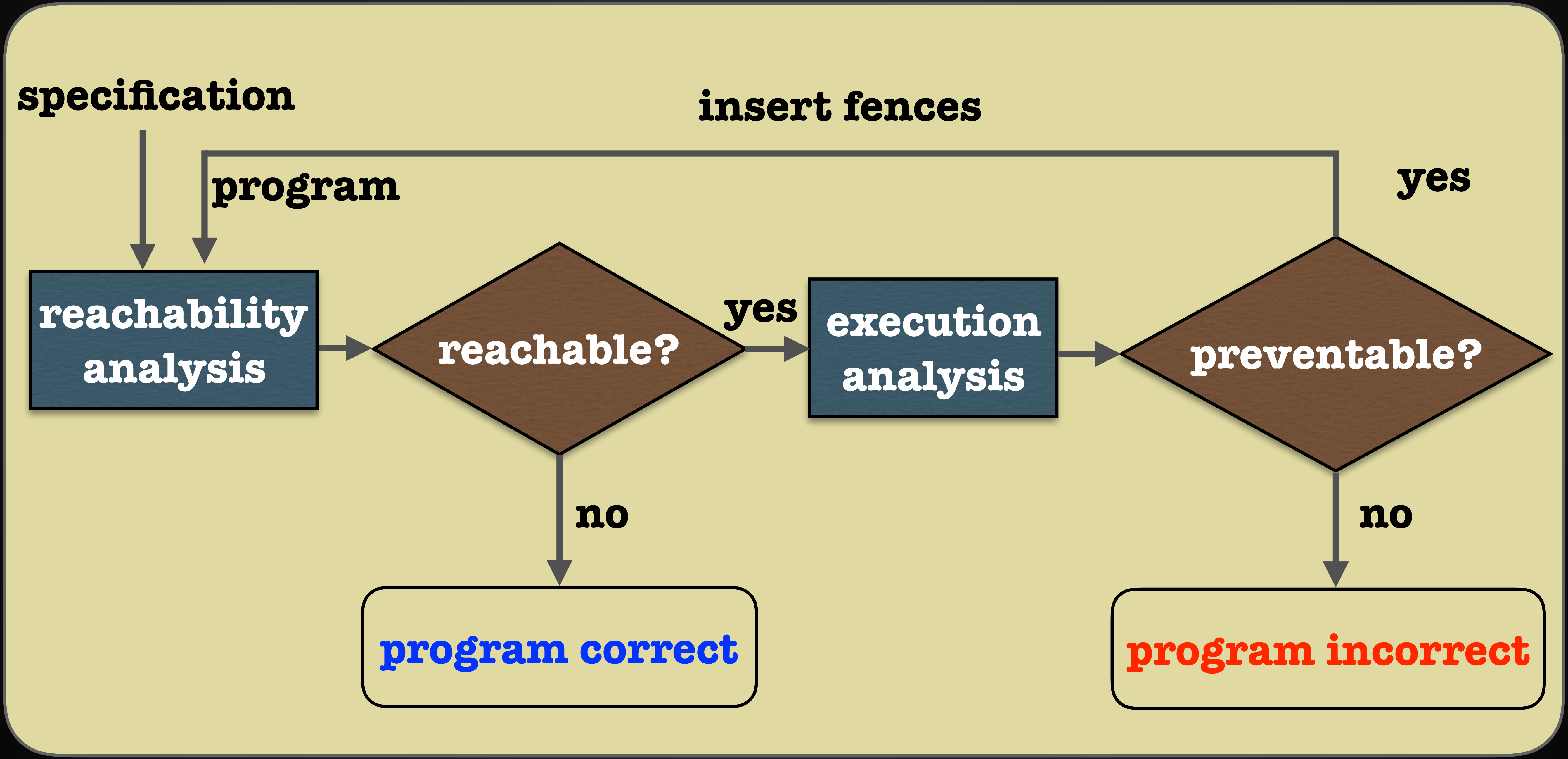
Thank you!

Question?

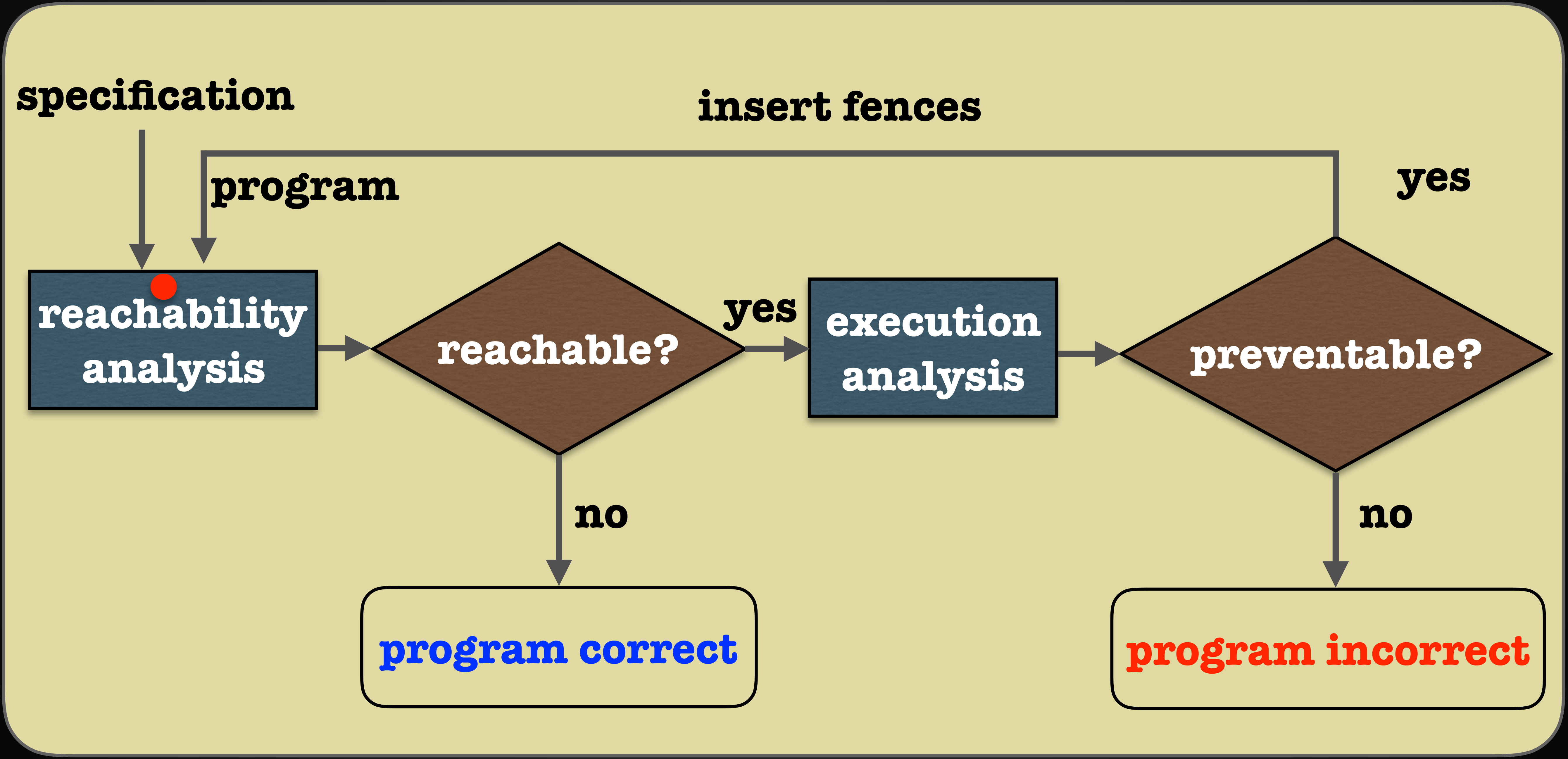
Appendix

Verification and Correction

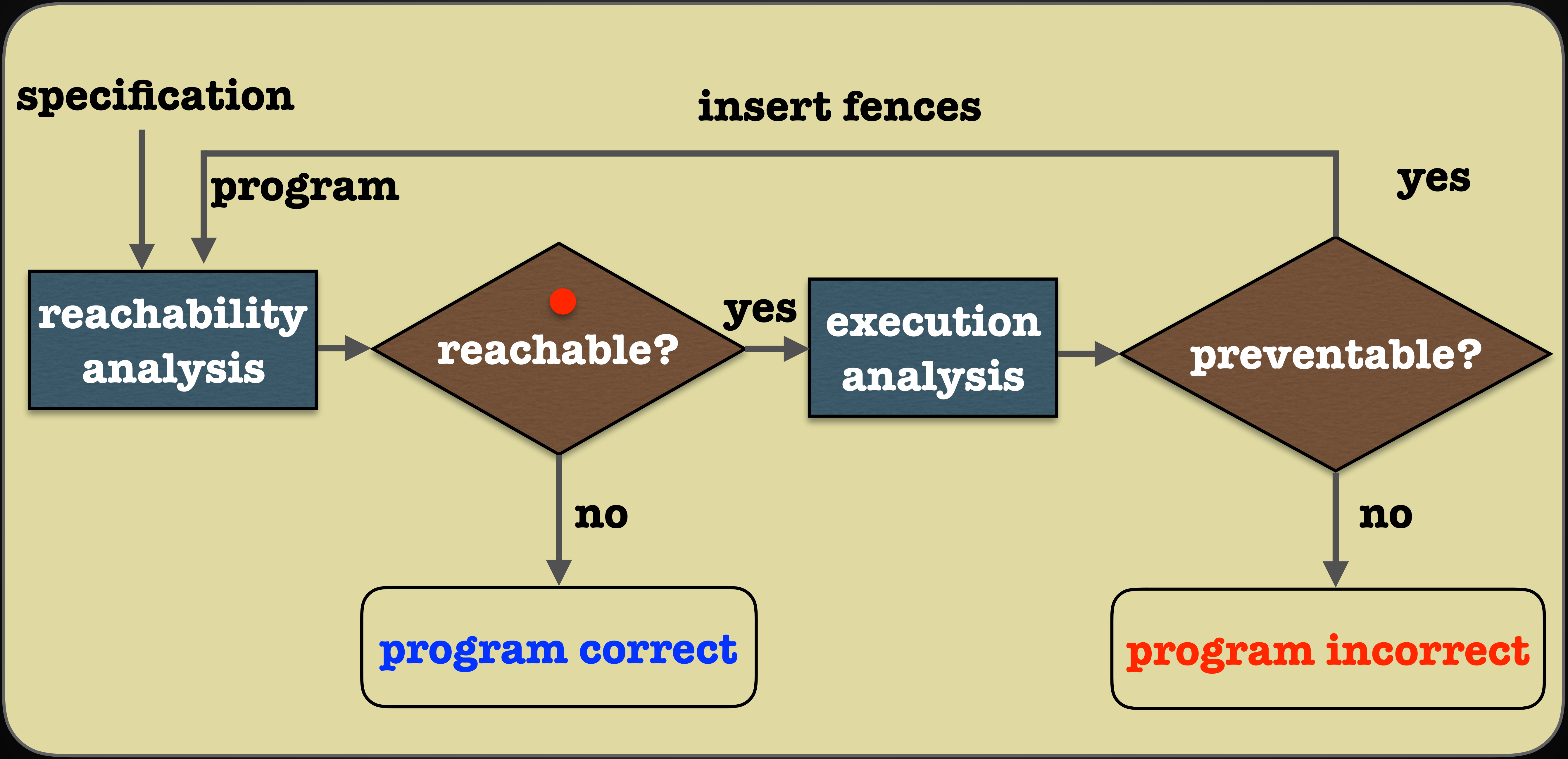
Verification and Correction



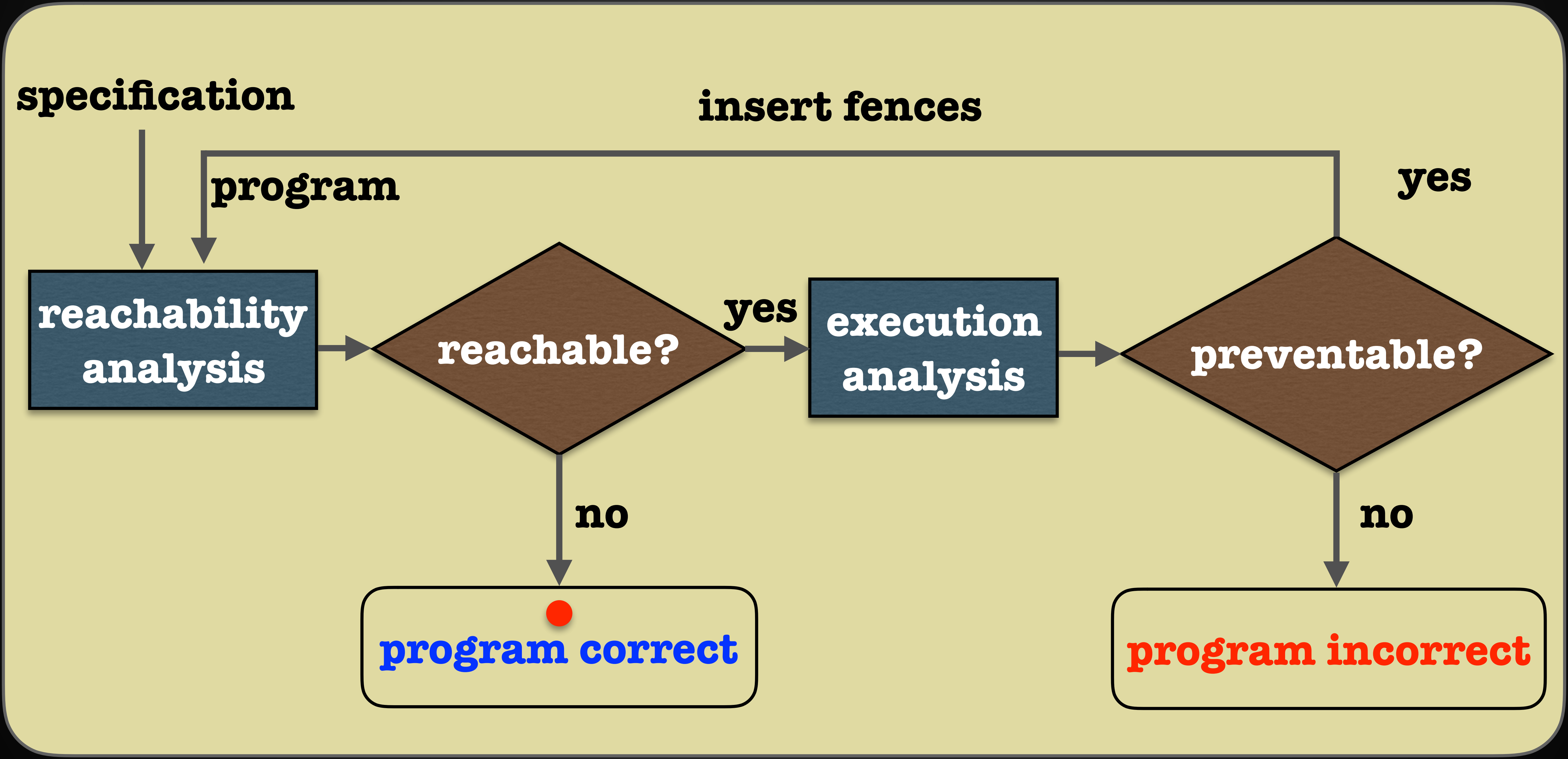
Verification and Correction



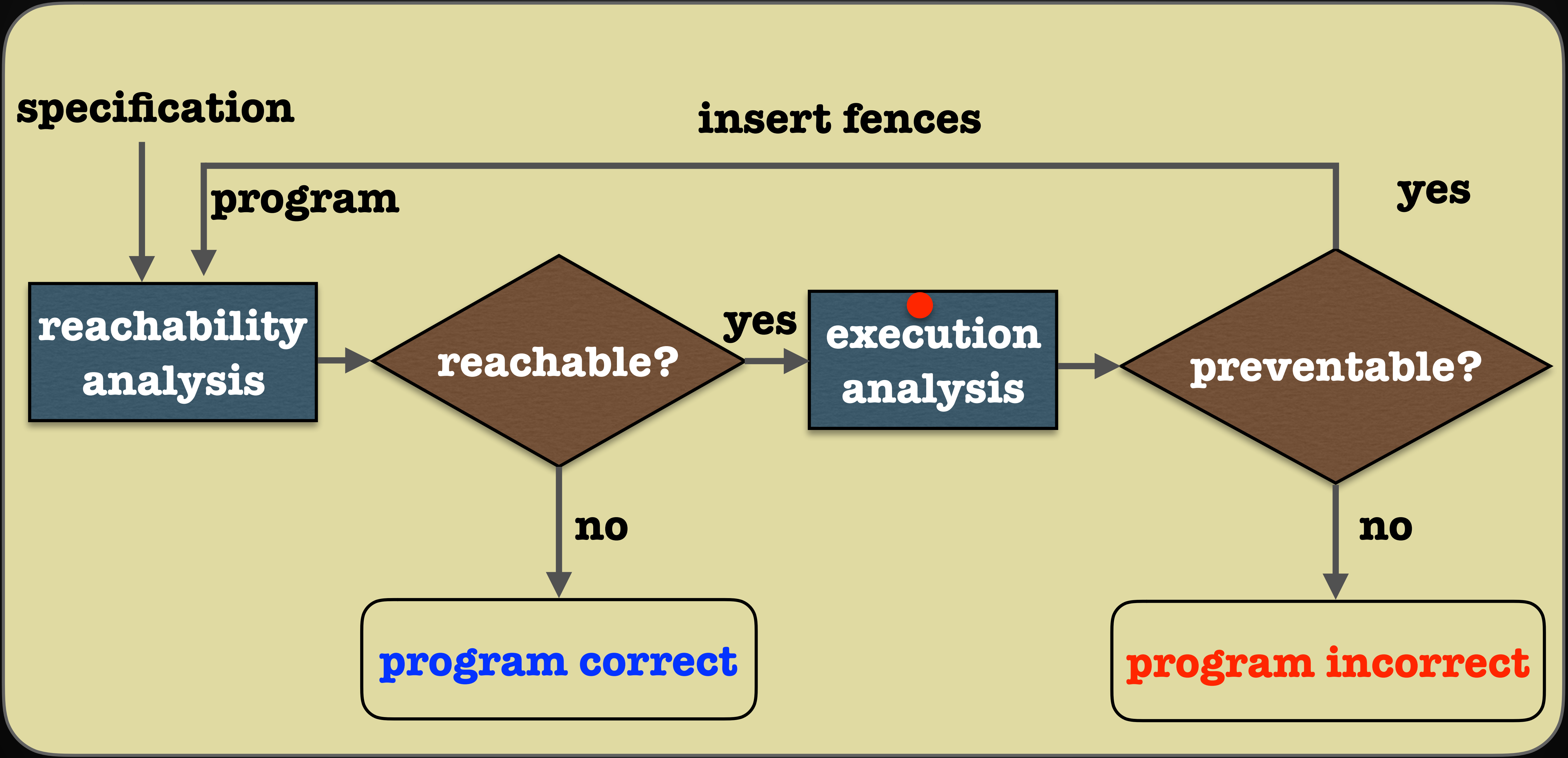
Verification and Correction



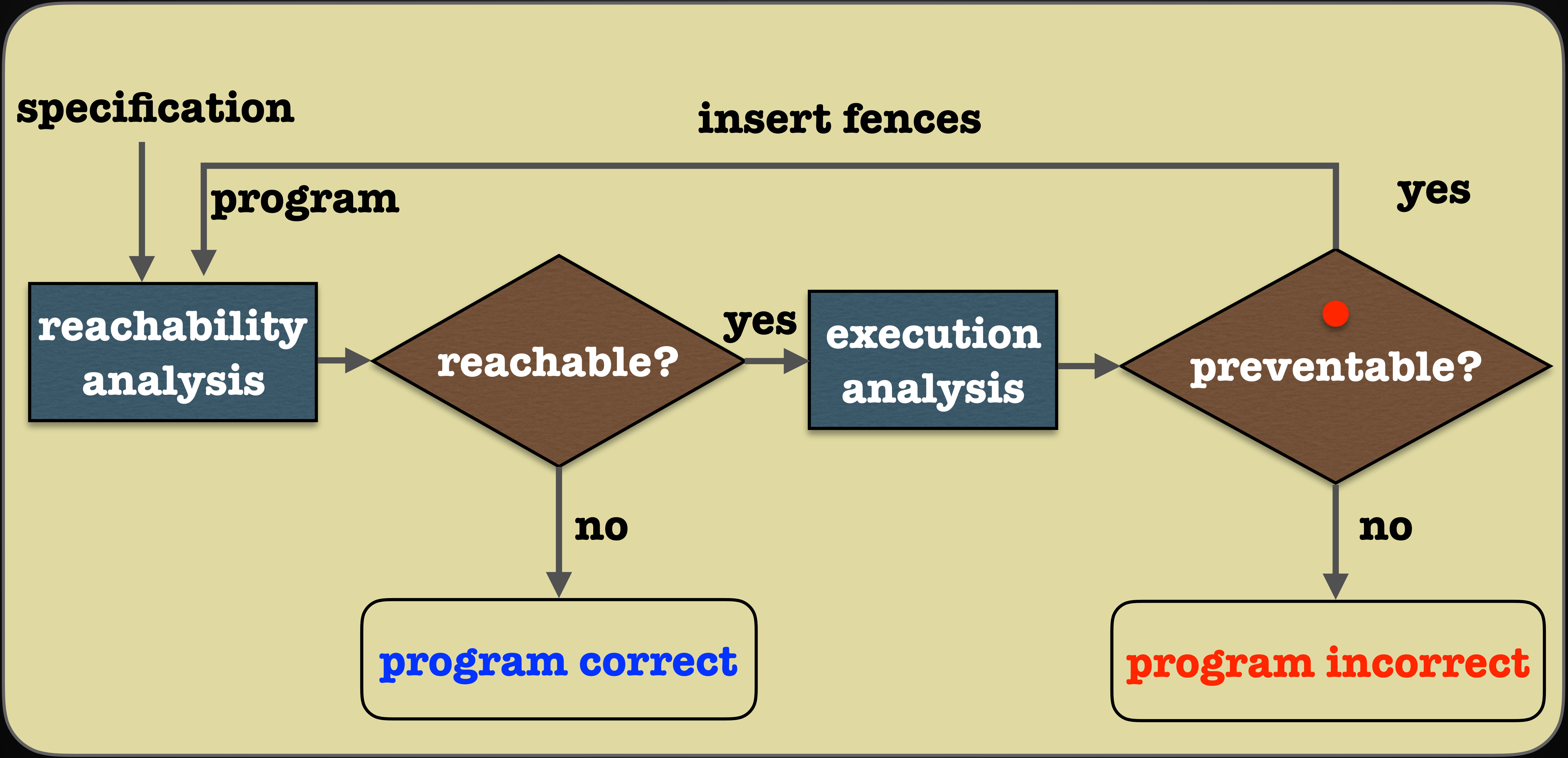
Verification and Correction



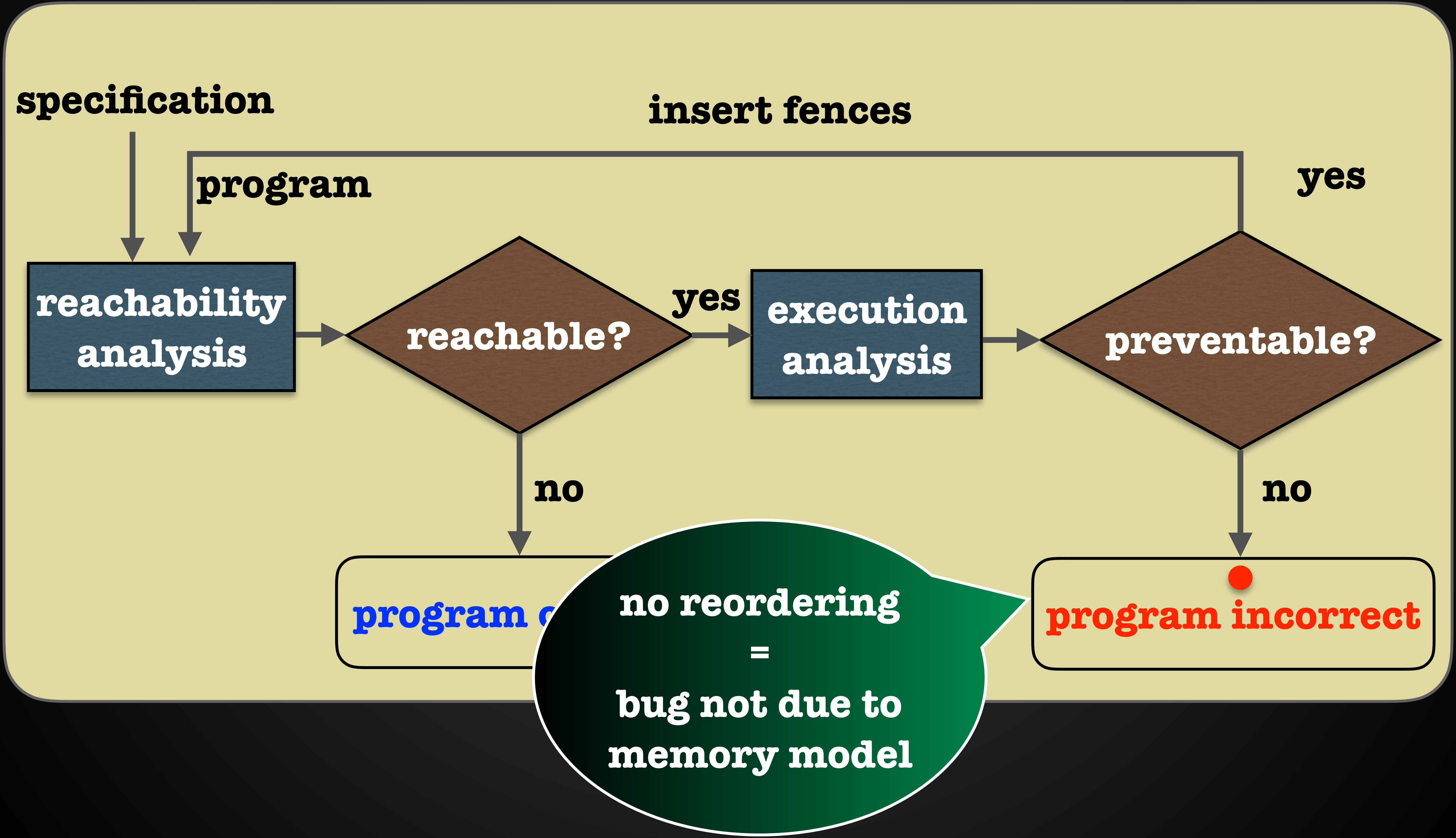
Verification and Correction



Verification and Correction

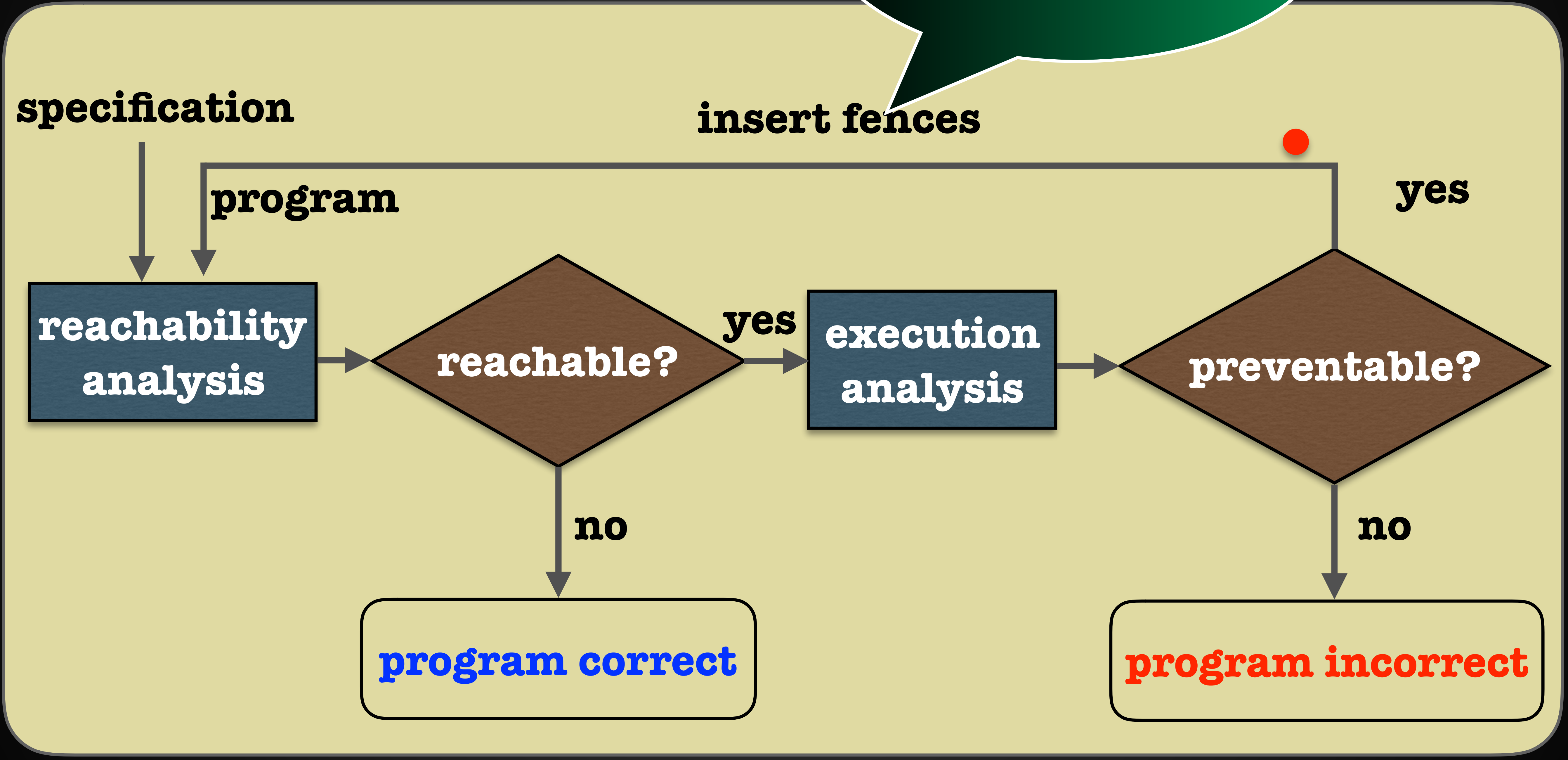


Verification and Correction

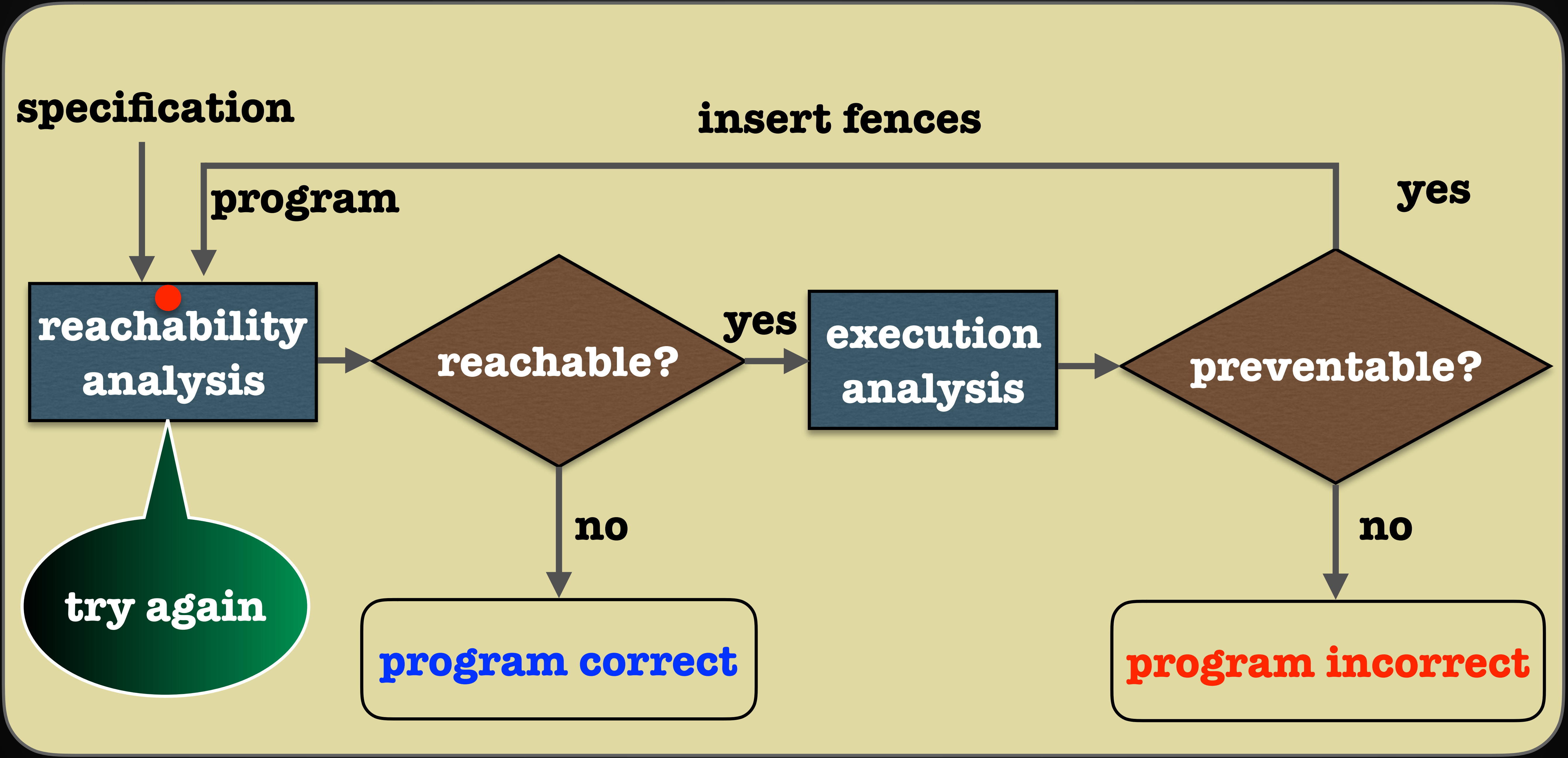


Verification an n

find reordering
and
prevent it



Verification and Correction



optimality = smallest set of fences needed for correctness