The Best of Both Worlds: Trading Efficiency and Optimality in Fence Insertion for TSO

Parosh Aziz Abdulla

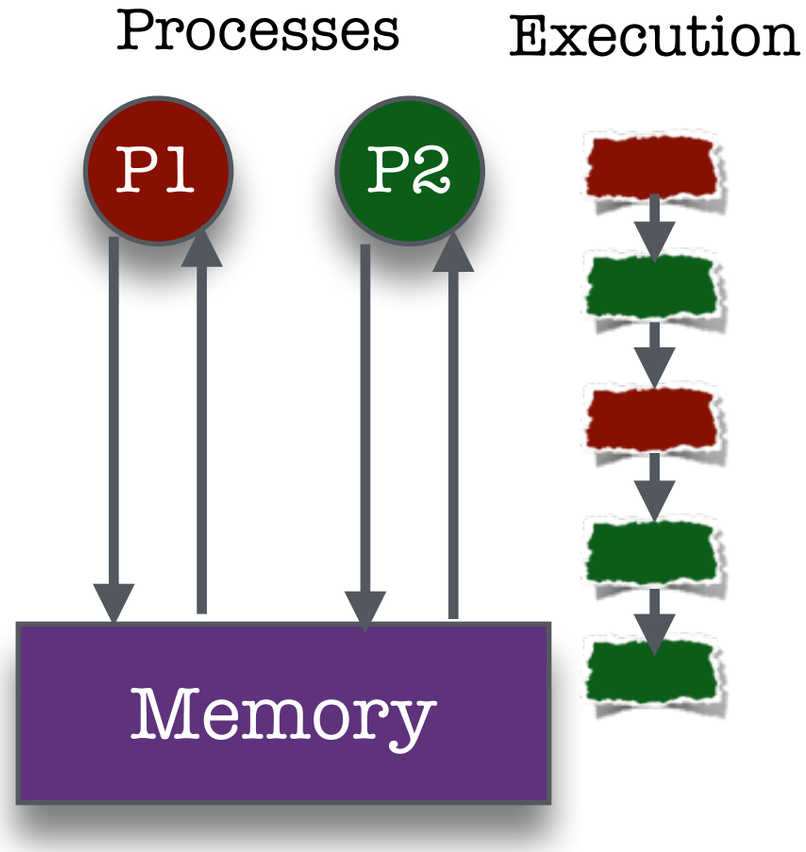
Mohamed Faouzi Atig

Tuan Phong Ngo

UPPSALA UNIVERSITET

Sequential Consistency

- Concurrent programs:
  - Processes write to and read from shared memory
  - Classical interleaving semantics (SC):
    - Shuffling atomic operations
    - Program order is preserved for each process



Dekker Mutual Exclusion Protocol

Init: x=y=0

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

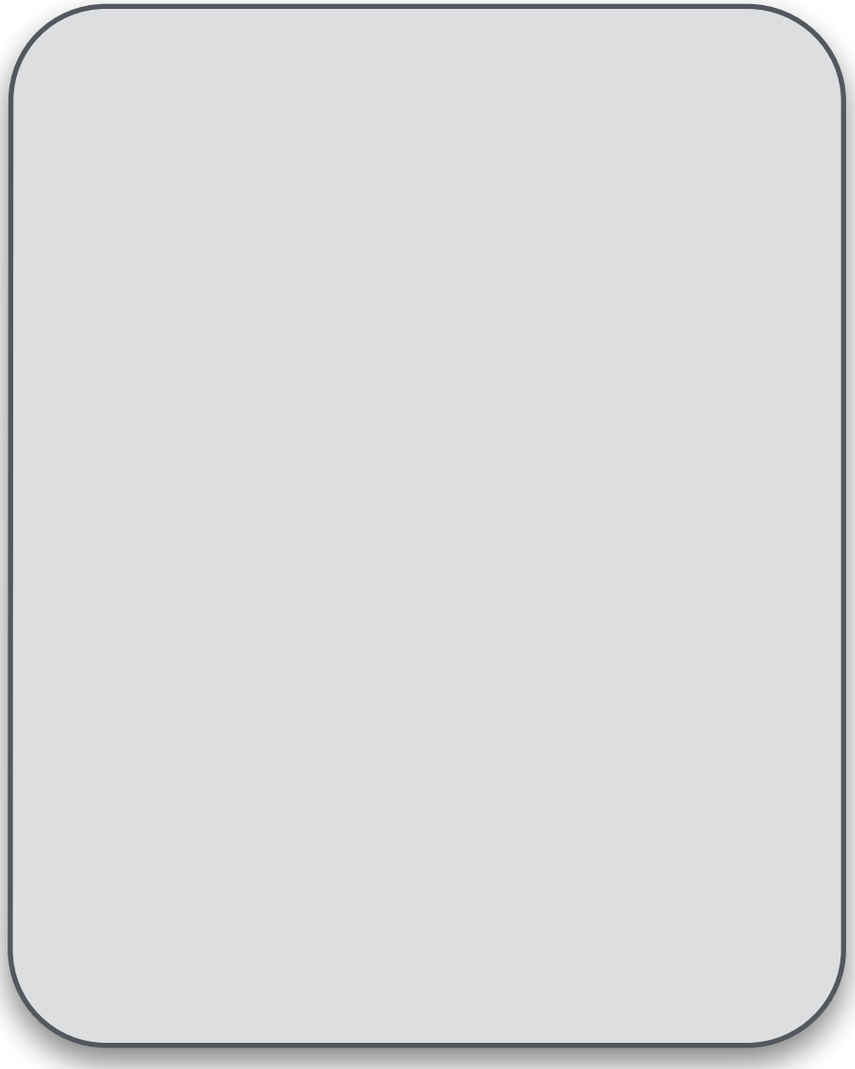
Process 1

```
1. y=1;
2. rx=x;
3. if (rx==0) {
4. //cs2;
5. }
```

Process 2

Specification S: not (cs1 && cs2)

An SC execution



x=0 y=0

Shared memory

Dekker Mutual Exclusion Protocol

Init: x=y=0

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

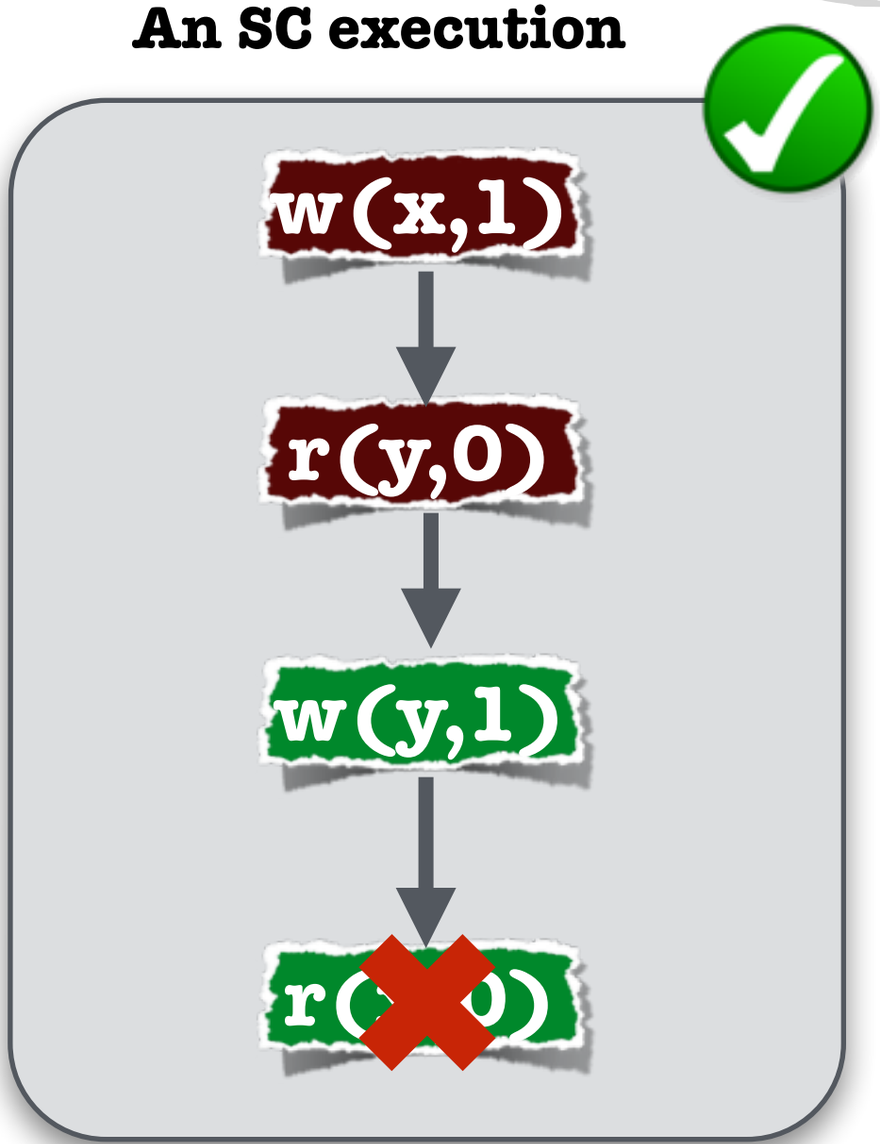
Process 1

```
1. y=1;
2. rx=x;
3. if (rx==0) {
4. //cs2;
5. }
```

Process 2

Specification S: not (cs1 && cs2)

An SC execution



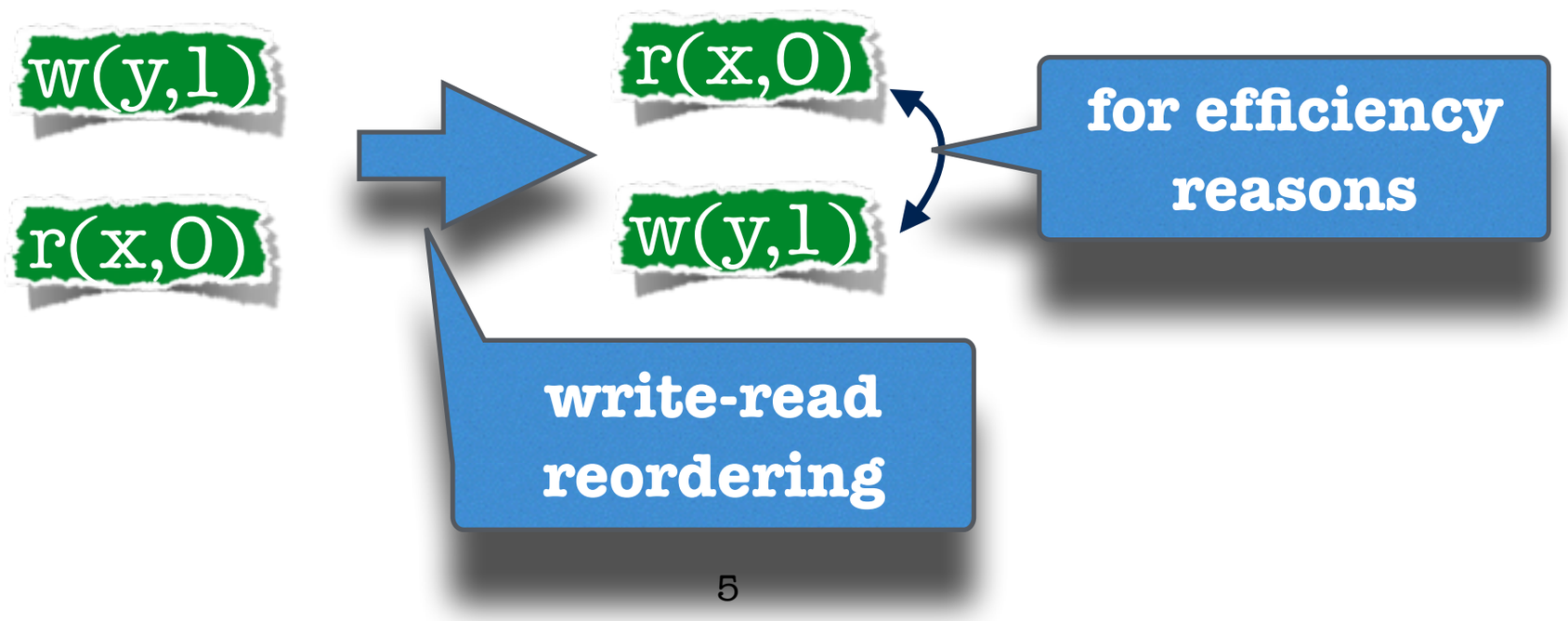
x=1 y=1

Shared memory

Weak Memory Models

- Modern processors and/or compilers:
  - Reorder instructions
  - Use caches and buffers

- Behaviors described by **weak memory models**:
  - Here: Total Store Ordering (TSO) memory model



Potential Bad Behaviours

Init: x=y=0

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

Process 1

```
1. y=1;
2. rx=x;
3. if (rx==0) {
4. //cs2;
5. }
```

Process 2

Specification S: not (cs1 && cs2)

A TSO execution

x=0 y=0

Shared memory

Potential Bad Behaviours

Init: x=y=0

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

Process 1

```
1. rx=x;
2. y=1;
3. if (rx==0) {
4. //cs2;
5. }
```

Process 2

Specification S: not (cs1 && cs2)

A TSO execution

x=0 y=0

Shared memory

Potential Bad Behaviours

Init: x=y=0

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

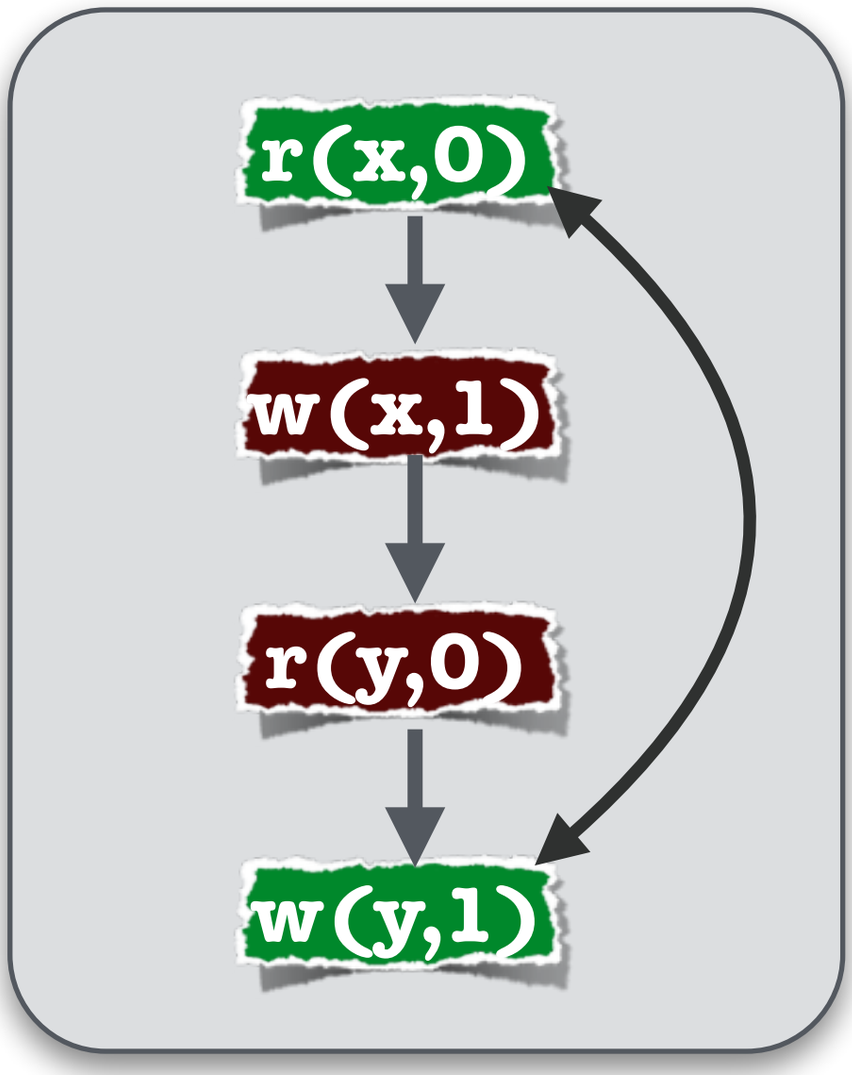
Process 1

```
1. rx=x;
2. y=1;
3. if (rx==0) {
4. //cs2;
5. }
```

Process 2

Specification S: not (cs1 && cs2)

A TSO execution



x=1 y=1

Shared memory

Potential Bad Behaviours

Fail under Intel Core2: observed 630/100.000 in one test [Sewell et al.]

Init: x=y=0

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

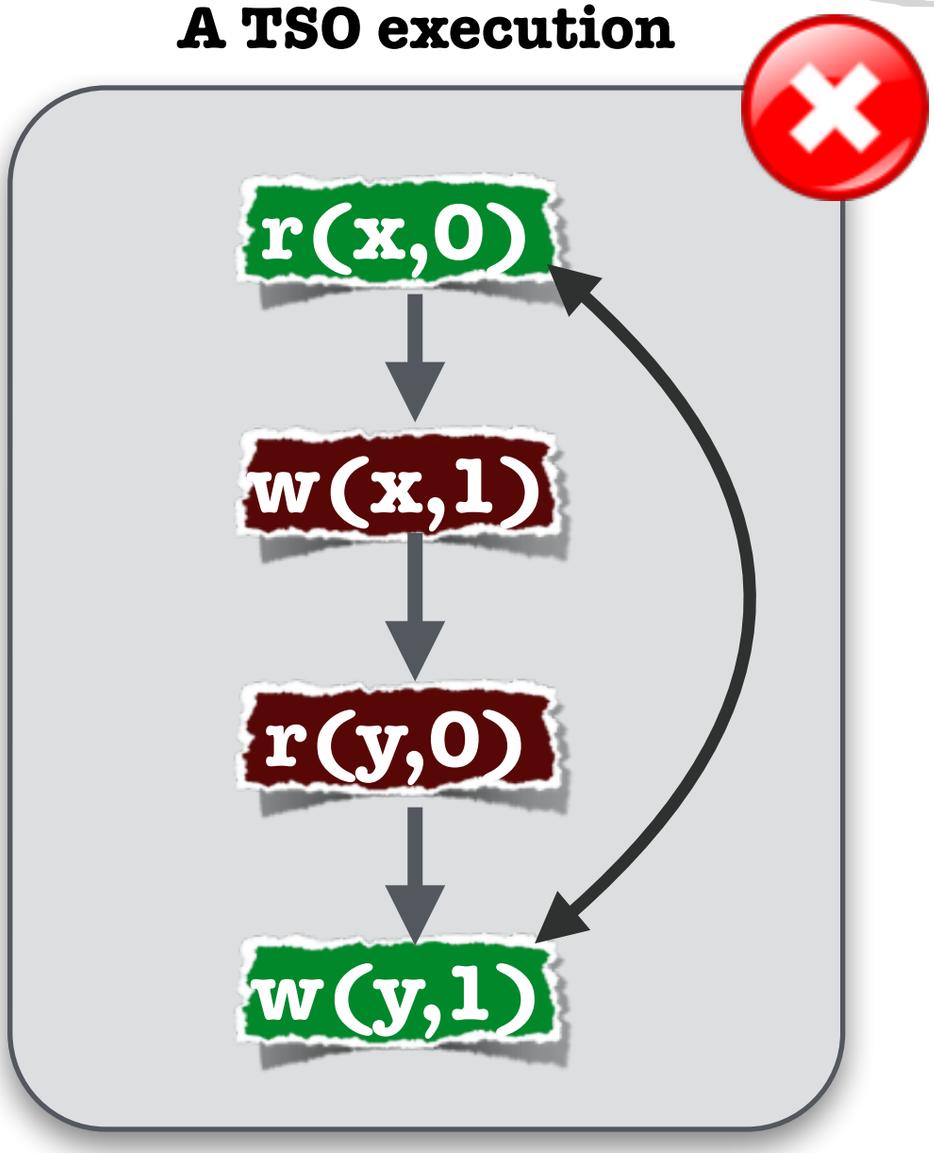
Process 1

```
1. rx=x;
2. y=1;
3. if (rx==0) {
4. //cs2;
5. }
```

Process 2

Specification S: not (cs1 && cs2)

A TSO execution

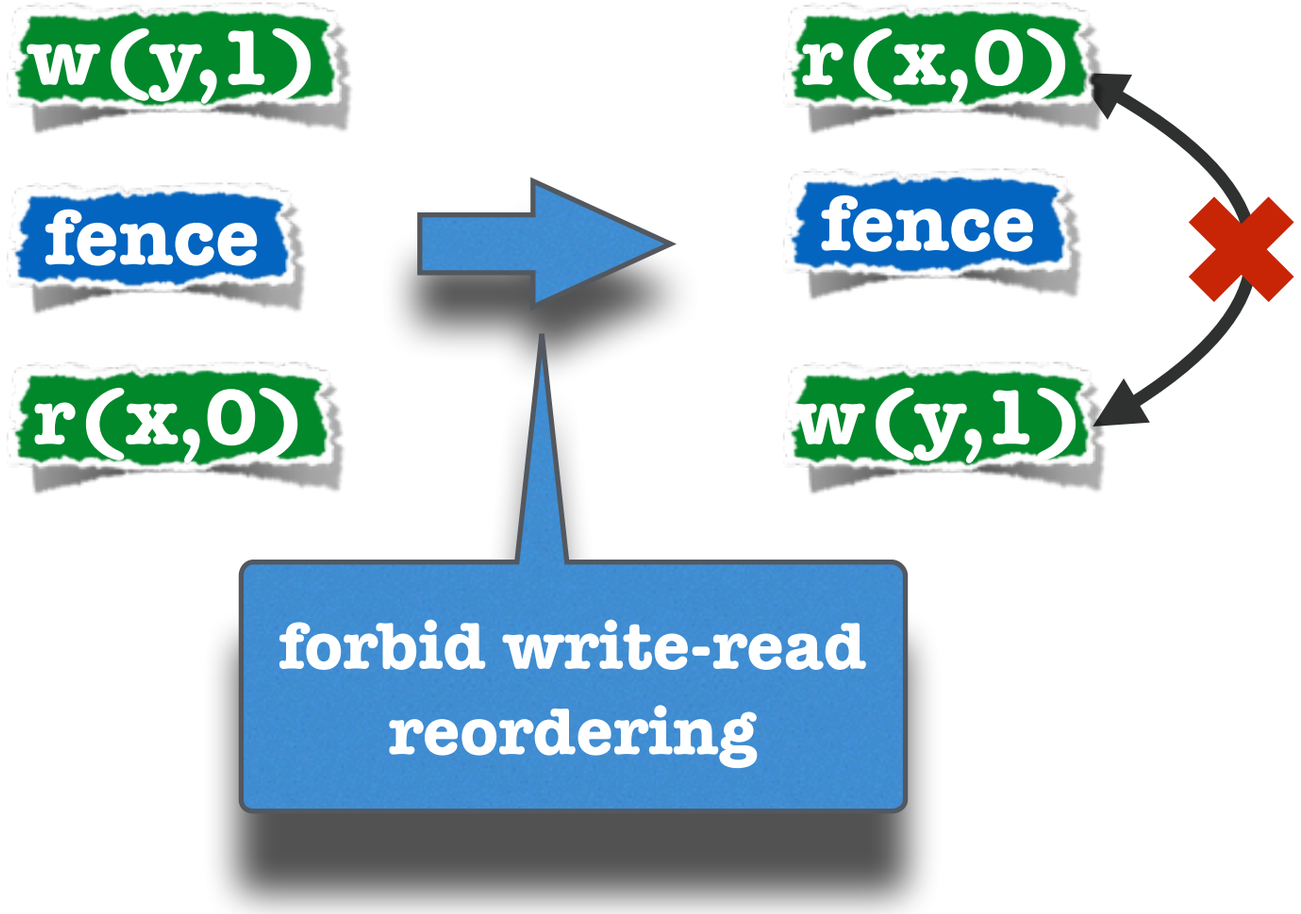


x=1 y=1

Shared memory

Correctness Problem

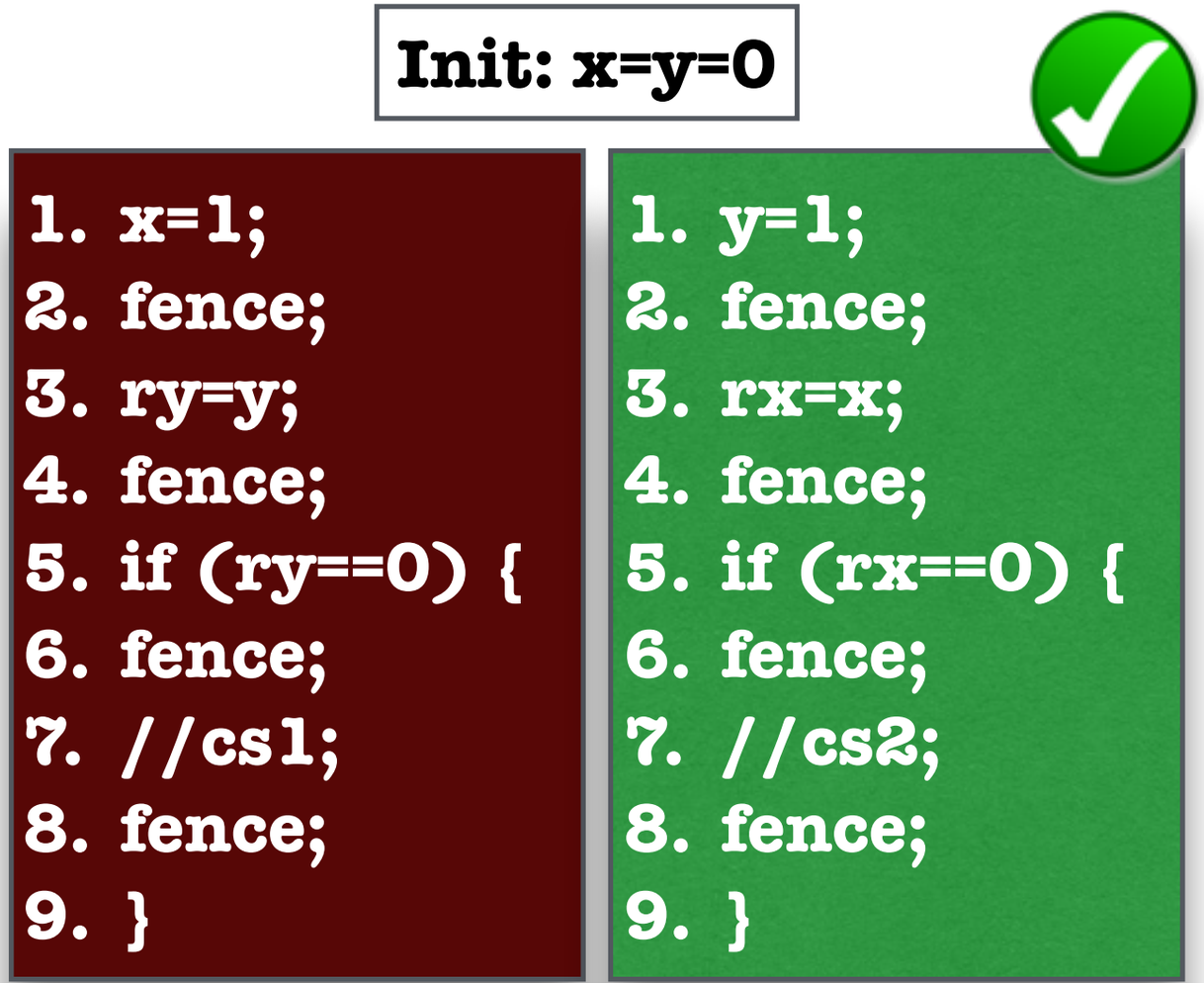
- How to make a program run correctly under TSO?
  - Insert memory **fences**



Naive Solution

- Insert a fence after each instruction

Init: x=y=0



Process 1	Process 2
1. x=1;	1. y=1;
2. fence;	2. fence;
3. ry=y;	3. rx=x;
4. fence;	4. fence;
5. if (ry==0) {	5. if (rx==0) {
6. fence;	6. fence;
7. //cs1;	7. //cs2;
8. fence;	8. fence;
9. }	9. }

Specification S: not (cs1 && cs2)

Naive Solution

- Insert a fence after each instruction

Disadvantage: Fences are expensive [Herlihy&Shavit]

- Normal instructions: **1-6 cycles**
- Fence instructions: **>100 cycles**

Init: x=y=0

Process 1	Process 2
1. x=1;	1. y=1;
2. fence;	2. fence;
3. ry=y;	3. rx=x;
4. fence;	4. fence;
5. if (ry==0) {	5. if (rx==0) {
6. fence;	6. fence;
7. //cs1;	7. //cs2;
8. fence;	8. fence;
9. }	9. }

Specification S: not (cs1 && cs2)

Optimal Solution

- Insert a **minimal number** of fences

Desired Properties:

- Automatic
- Efficient

Init: x=y=0

```
1. x=1;
2. fence;
3. ry=y;
4. if (ry==0) {
5. //cs1;
6. }
```

Process 1

```
1. y=1;
2. fence;
3. rx=x;
4. if (rx==0) {
5. //cs2;
6. }
```

Process 2

Specification S: not (cs1 && cs2)

Problem of Interest

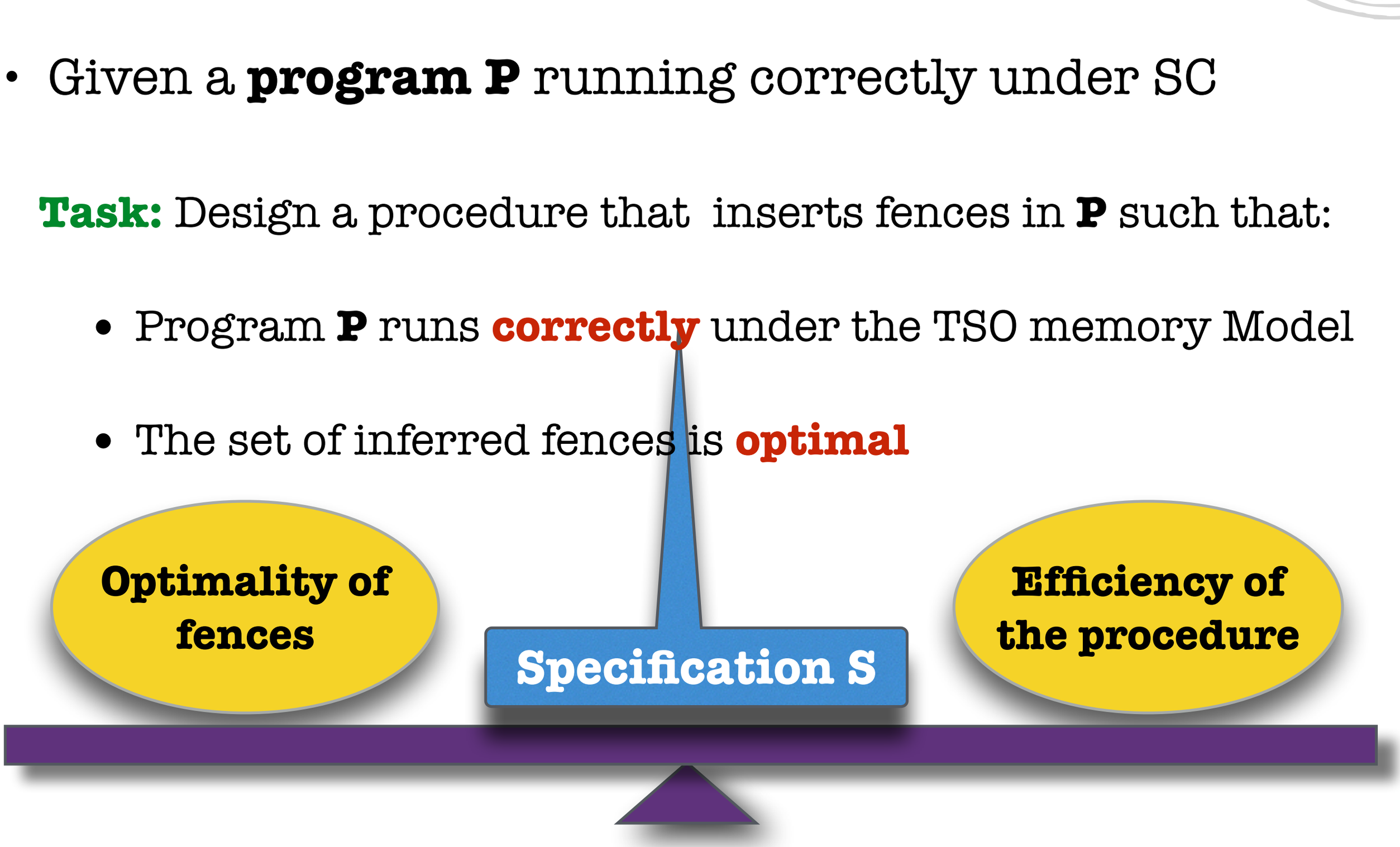
- Given a **program P** running correctly under SC

Problem of Interest

- Given a **program P** running correctly under SC

Task: Design a procedure that inserts fences in **P** such that:

- Program **P** runs **correctly** under the TSO memory Model
- The set of inferred fences is **optimal**

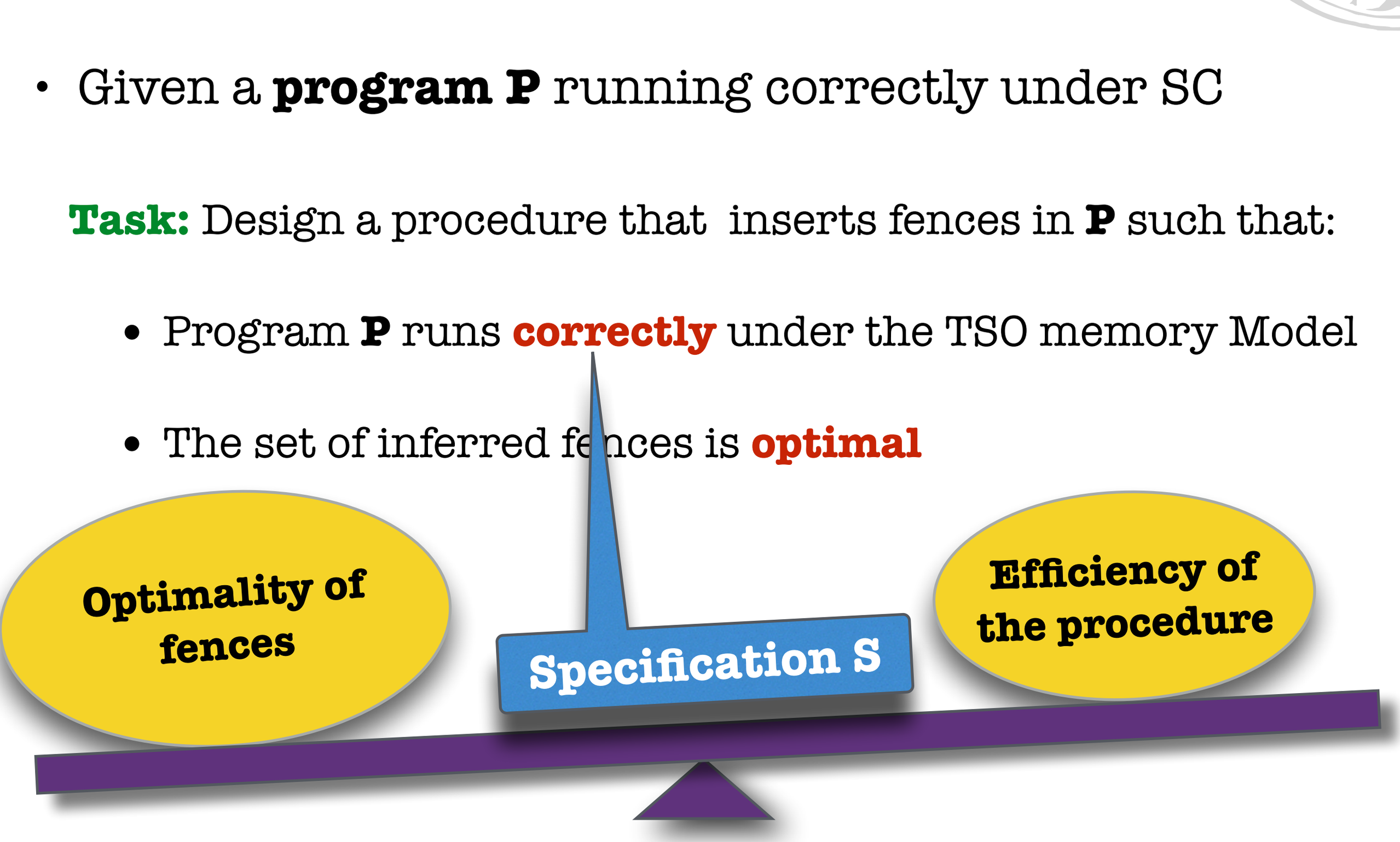


Problem of Interest

- Given a **program P** running correctly under SC

Task: Design a procedure that inserts fences in **P** such that:

- Program **P** runs **correctly** under the TSO memory Model
- The set of inferred fences is **optimal**

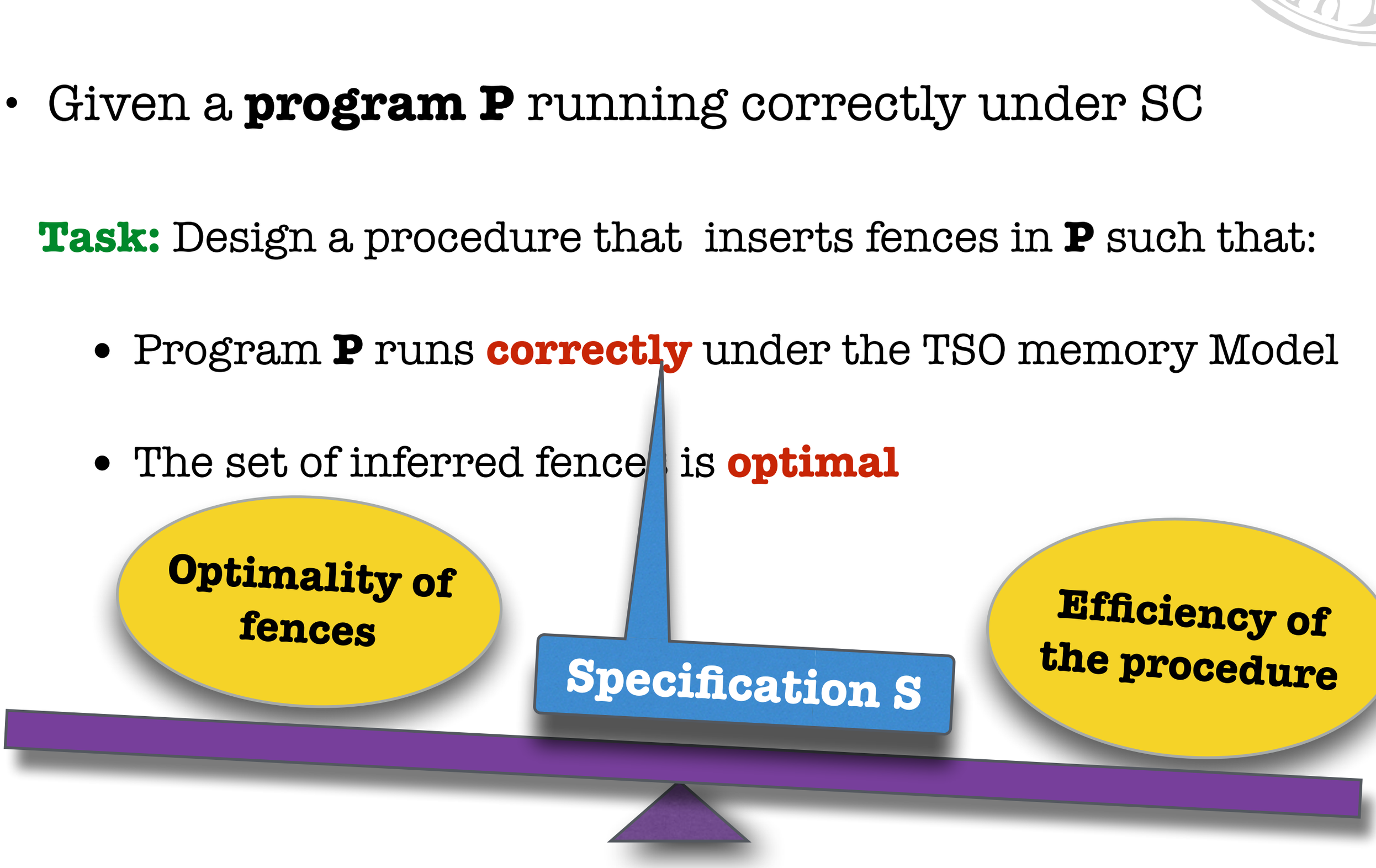


Problem of Interest

- Given a **program P** running correctly under SC

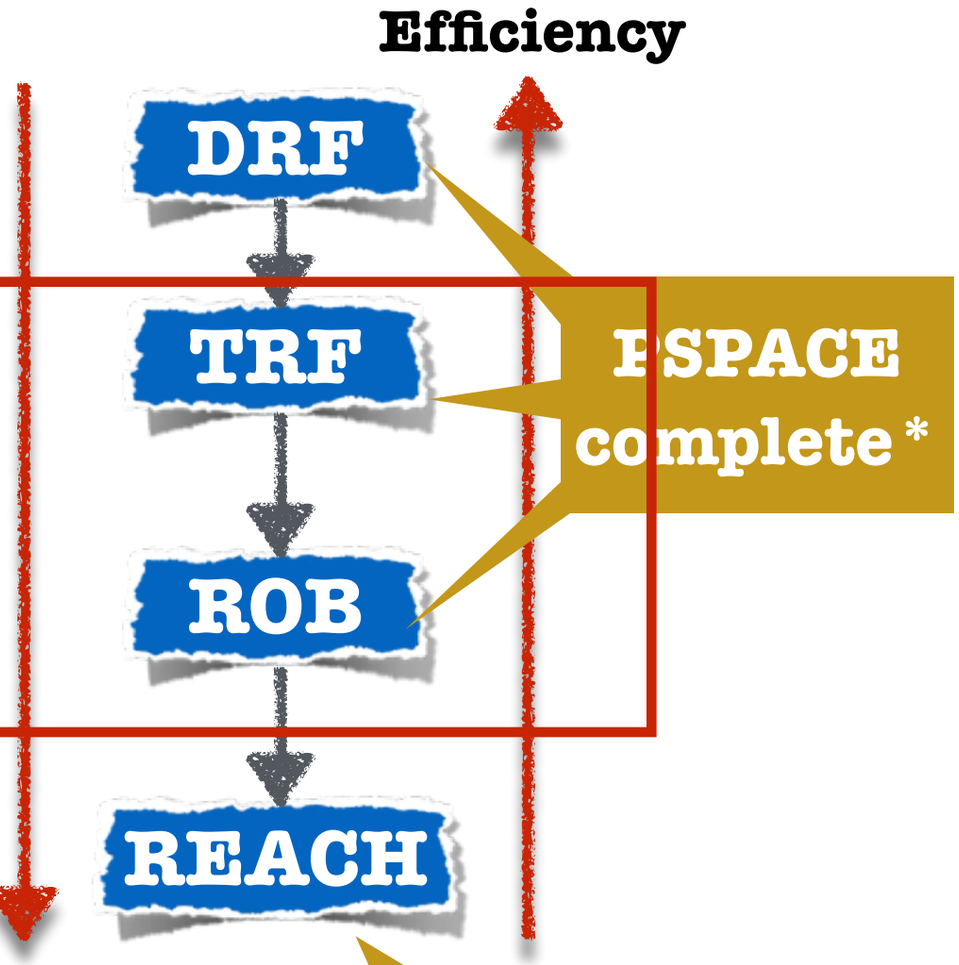
Task: Design a procedure that inserts fences in **P** such that:

- Program **P** runs **correctly** under the TSO memory Model
- The set of inferred fences is **optimal**



Specifications

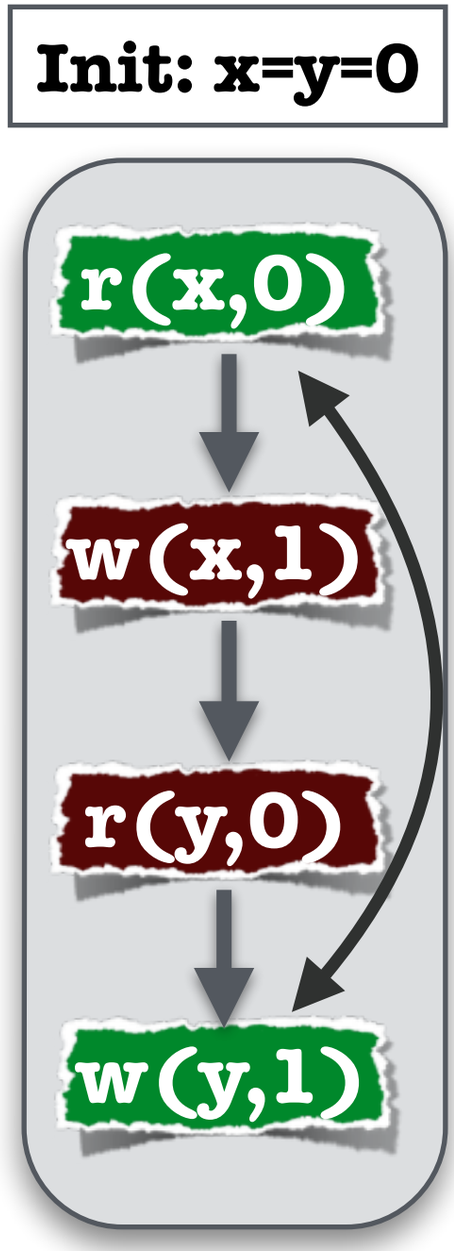
- **DRF**: data-race freedom, [Adve, Hill 1993]
- **TRF**: triangular-race freedom, [Owens]
- **ROB**: robustness, [Shasha&Snir], [Bouajjani et al.], [Alglave et al.]
- **REACH**: reachability, [Kuperstein et al.], [Abdulla et al.]



The program exhibits the same observable behaviors under SC and TSO

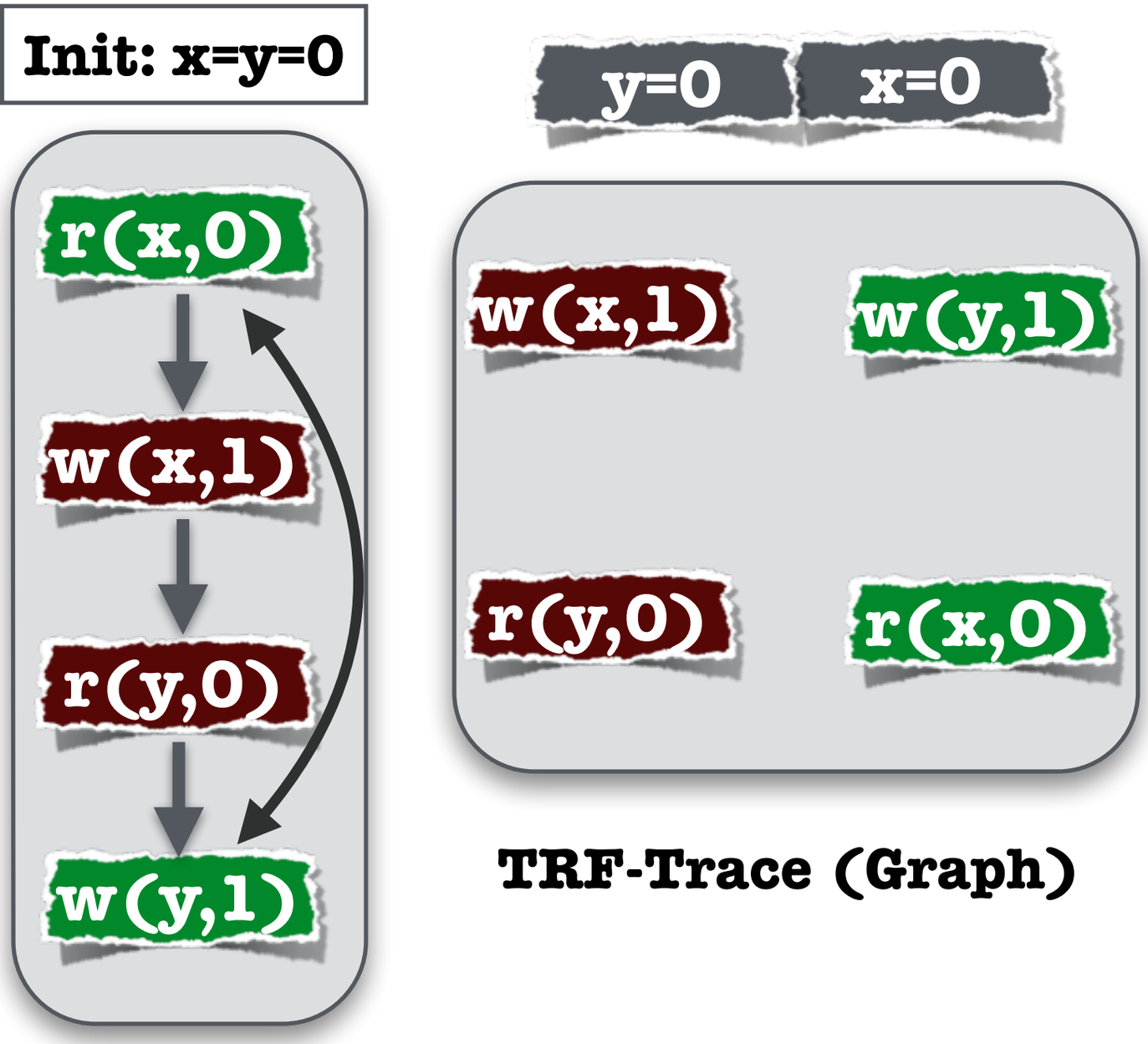
*** : for finite-state programs**

Observable Behaviors for **TRF**



An execution

Observable Behaviors for **TRF**

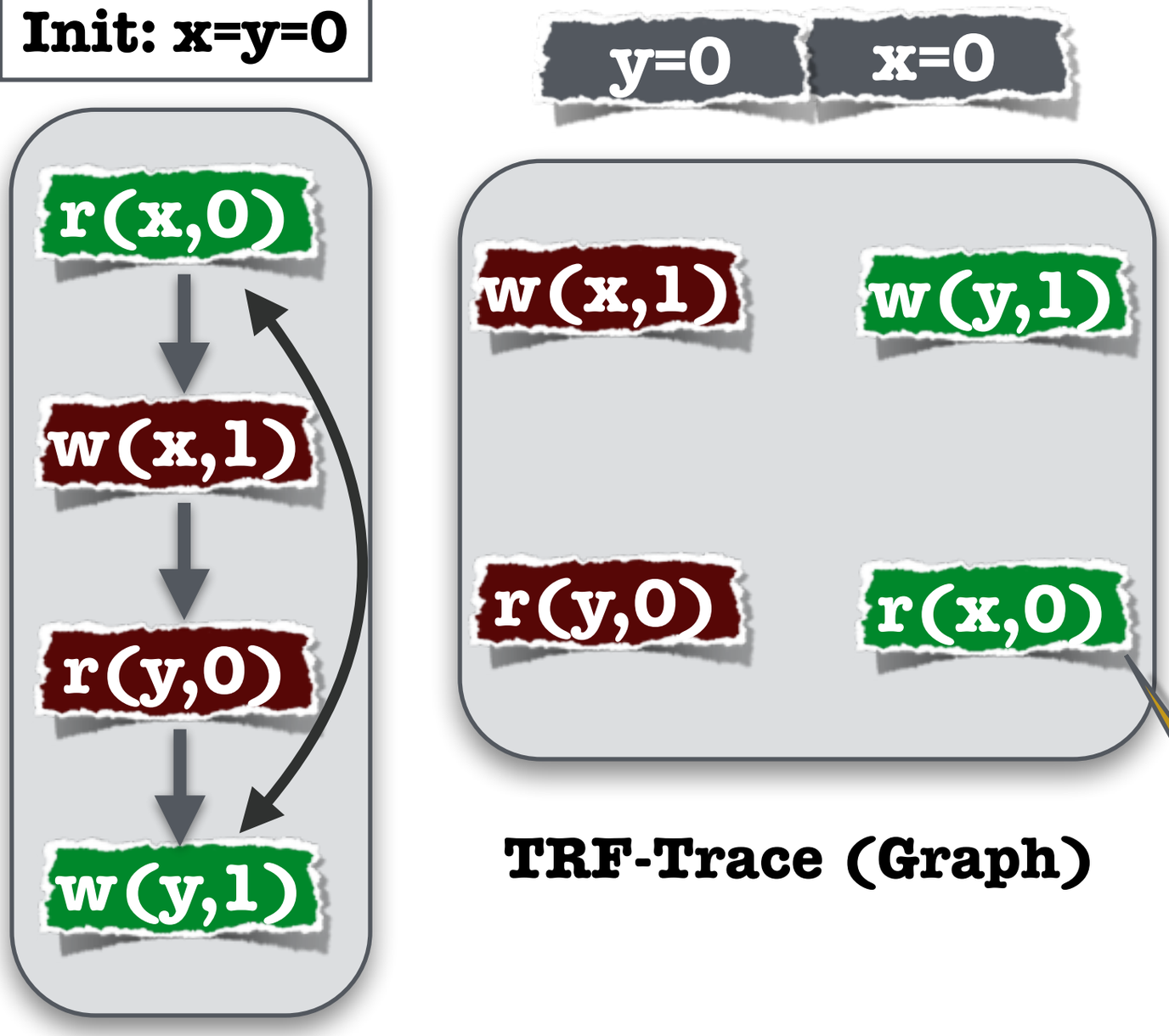


An execution

TRF-Trace (Graph)

Observable Behaviors for **TRF**

Init: x=y=0



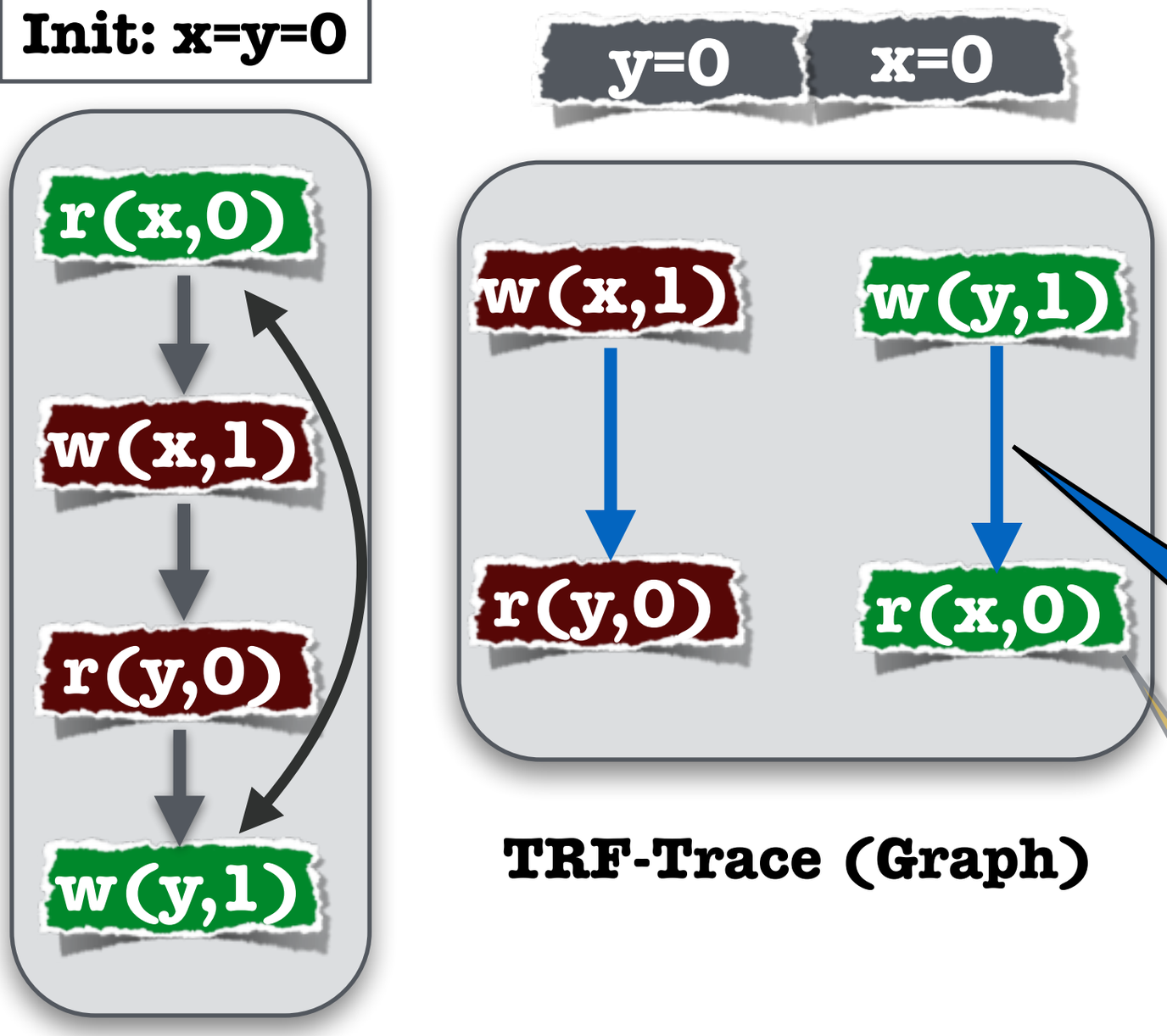
An execution

TRF-Trace (Graph)

Memory actions in the execution

Observable Behaviors for **TRF**

Init: x=y=0



An execution

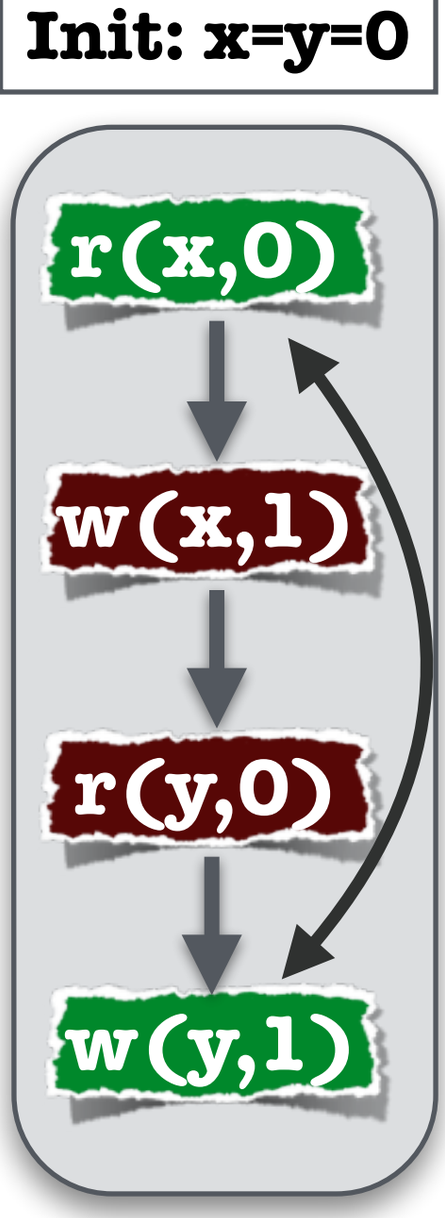
TRF-Trace (Graph)

po: Order of actions issued by one process

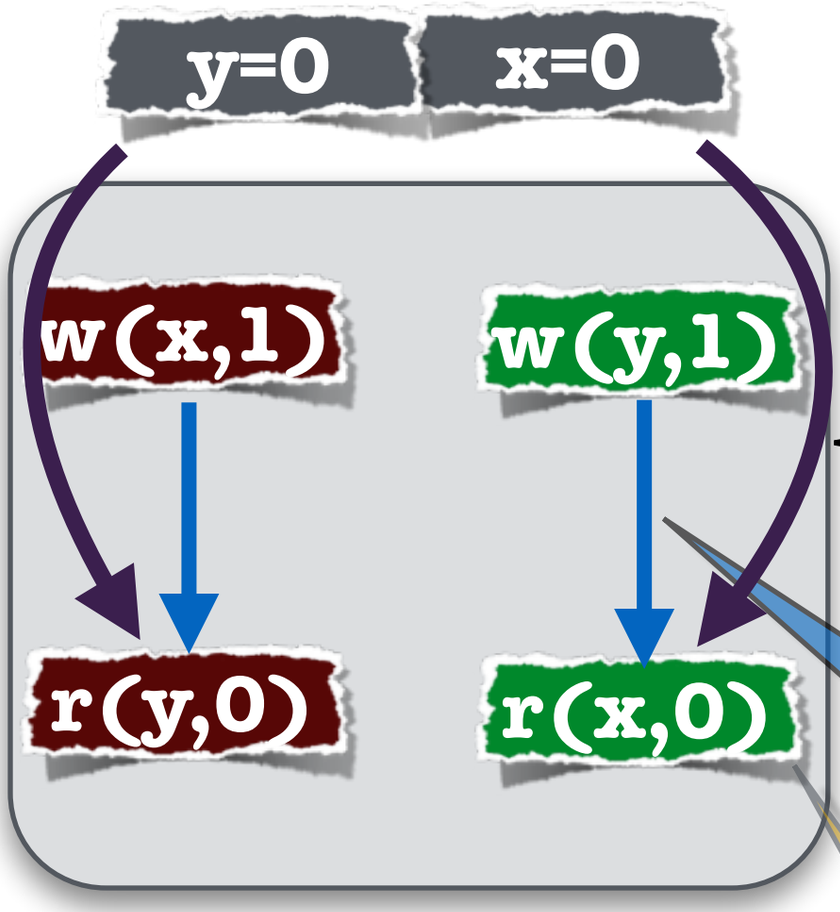
Memory actions in the execution

Observable Behaviors for **TRF**

Init: x=y=0



An execution



TRF-Trace (Graph)

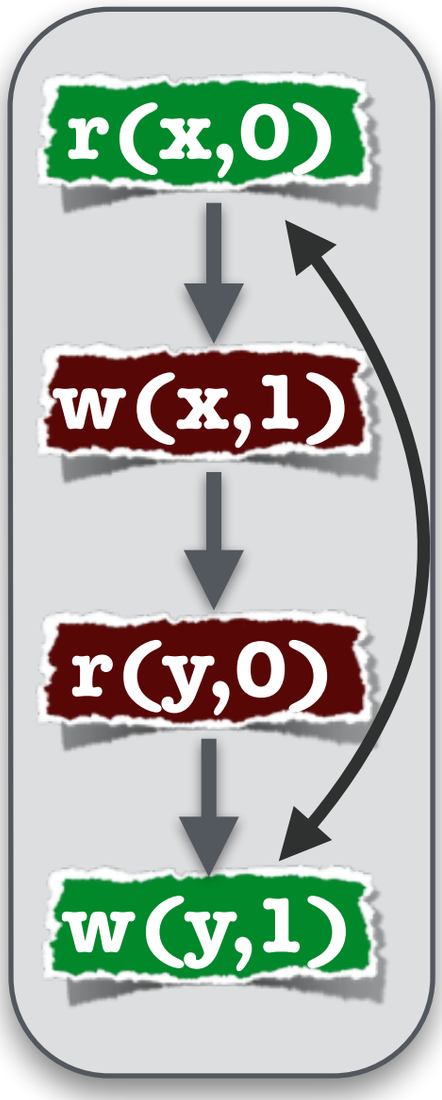
rdf: Write is source of a read

po: Order of actions issued by one process

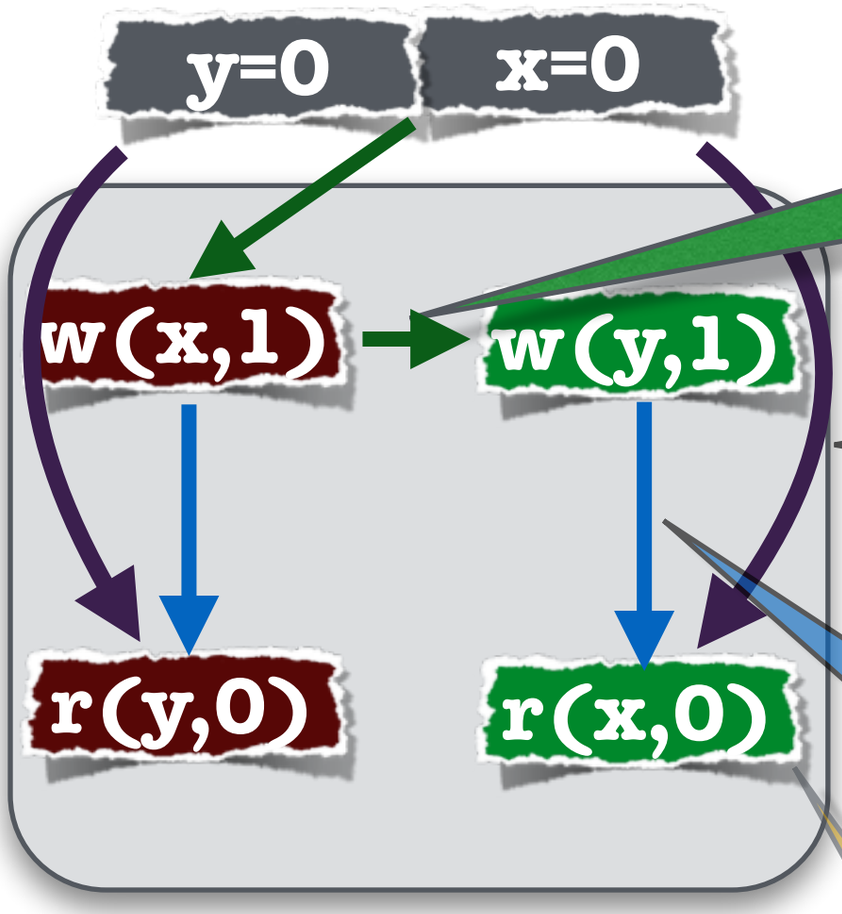
Memory actions in the execution

Observable Behaviors for **TRF**

Init: x=y=0



An execution



TRF-Trace (Graph)

=

{po,rdf,tst}

tst: Order of writes

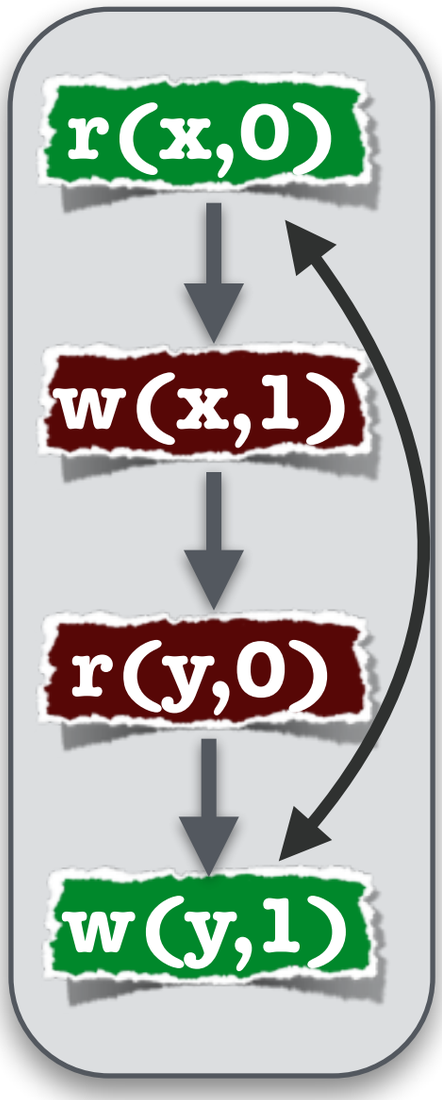
rdf: Write is source of a read

po: Order of actions issued by one process

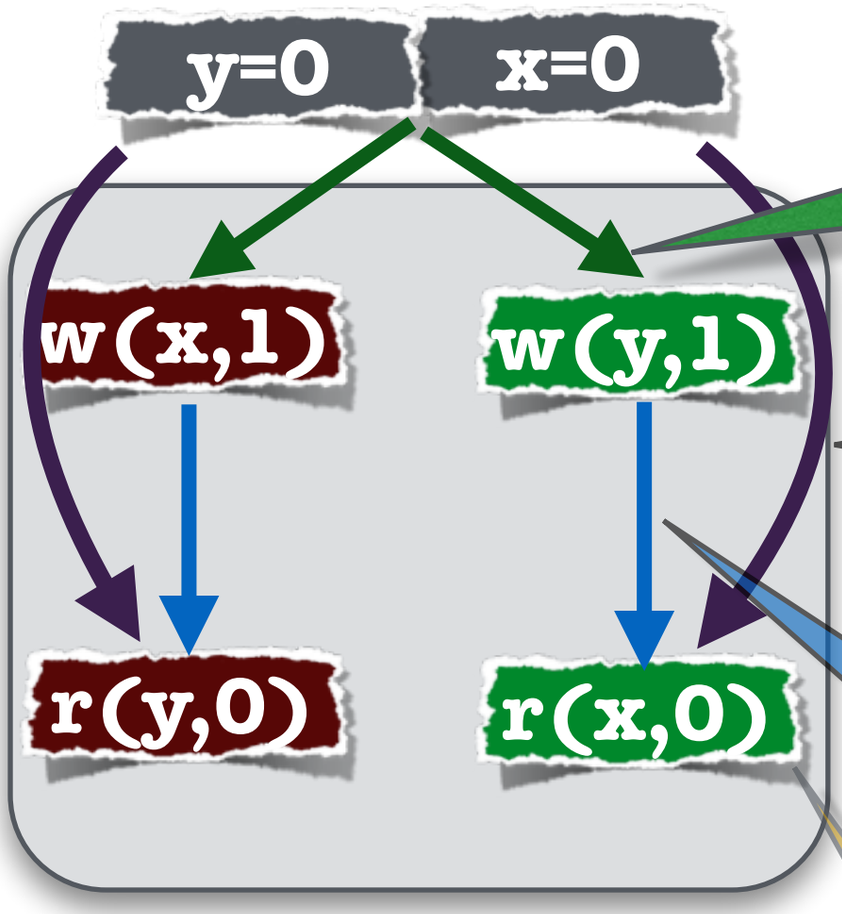
Memory actions in the execution

Observable Behaviors for **ROB**

Init: x=y=0



An execution



ROB-Trace (Graph)

=

{po,rdf,st}

st: Order of writes per variable

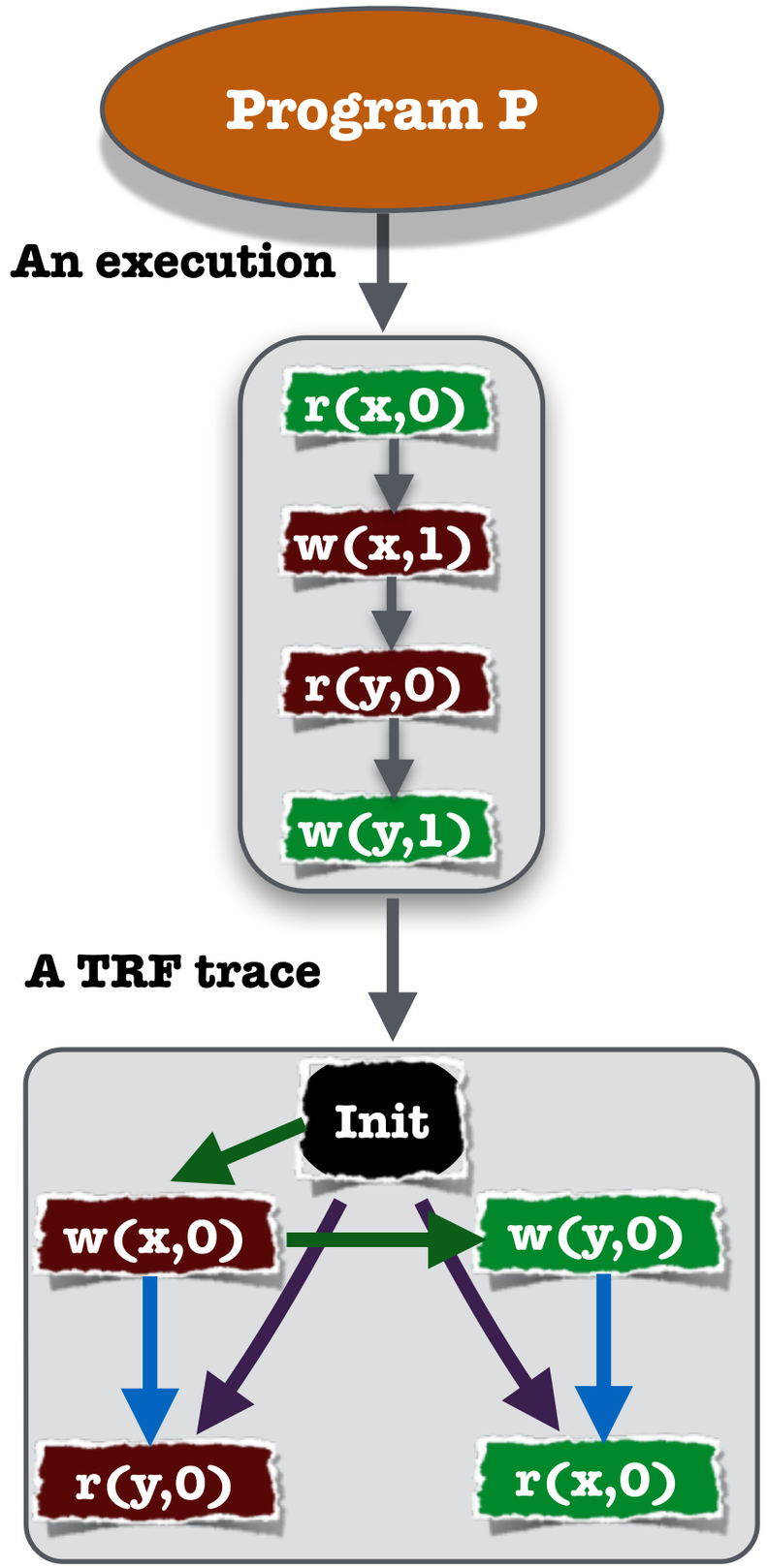
rdf: Write is source of a read

po: Order of actions issued by one process

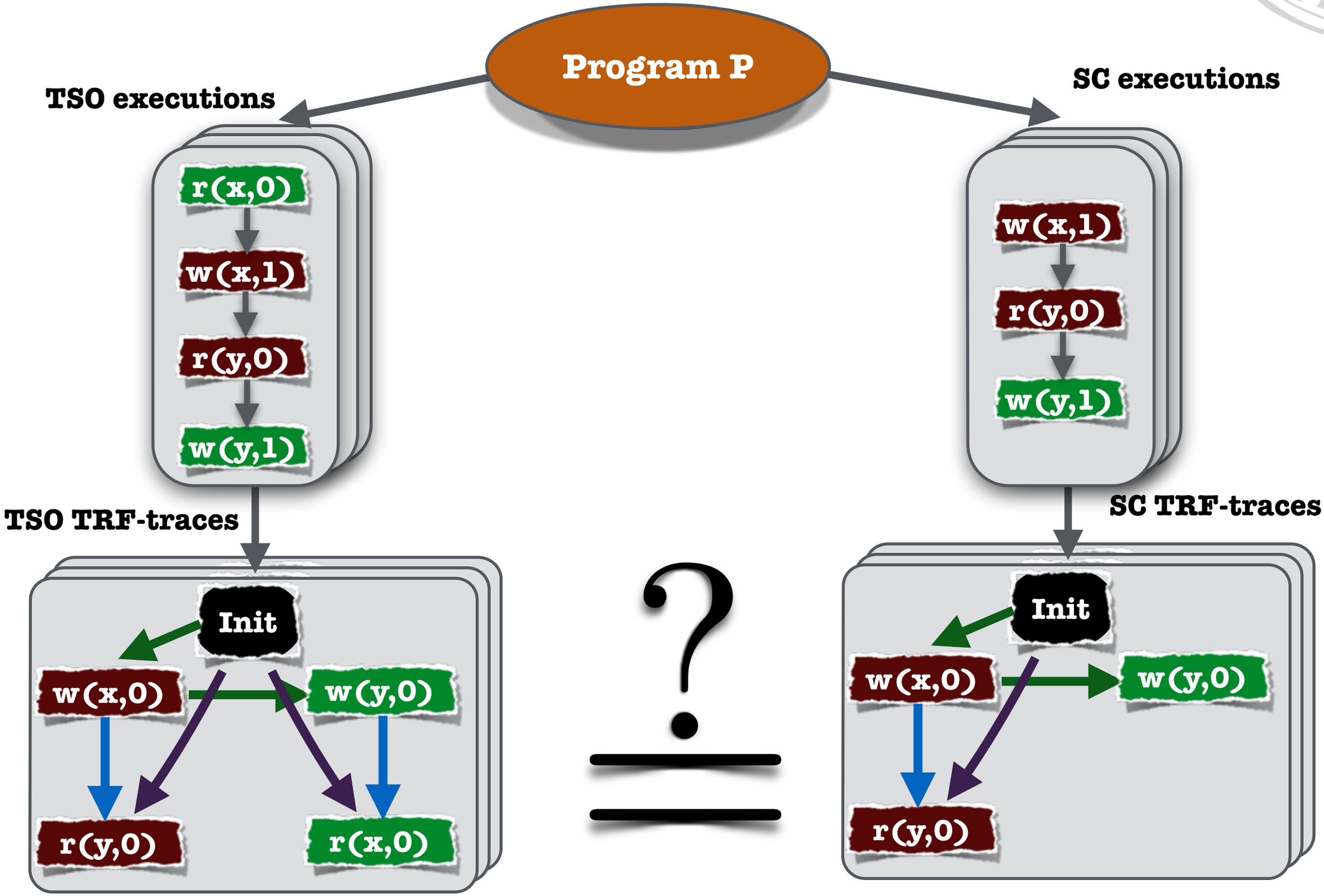
Memory actions in the execution

TRF Correctness Target

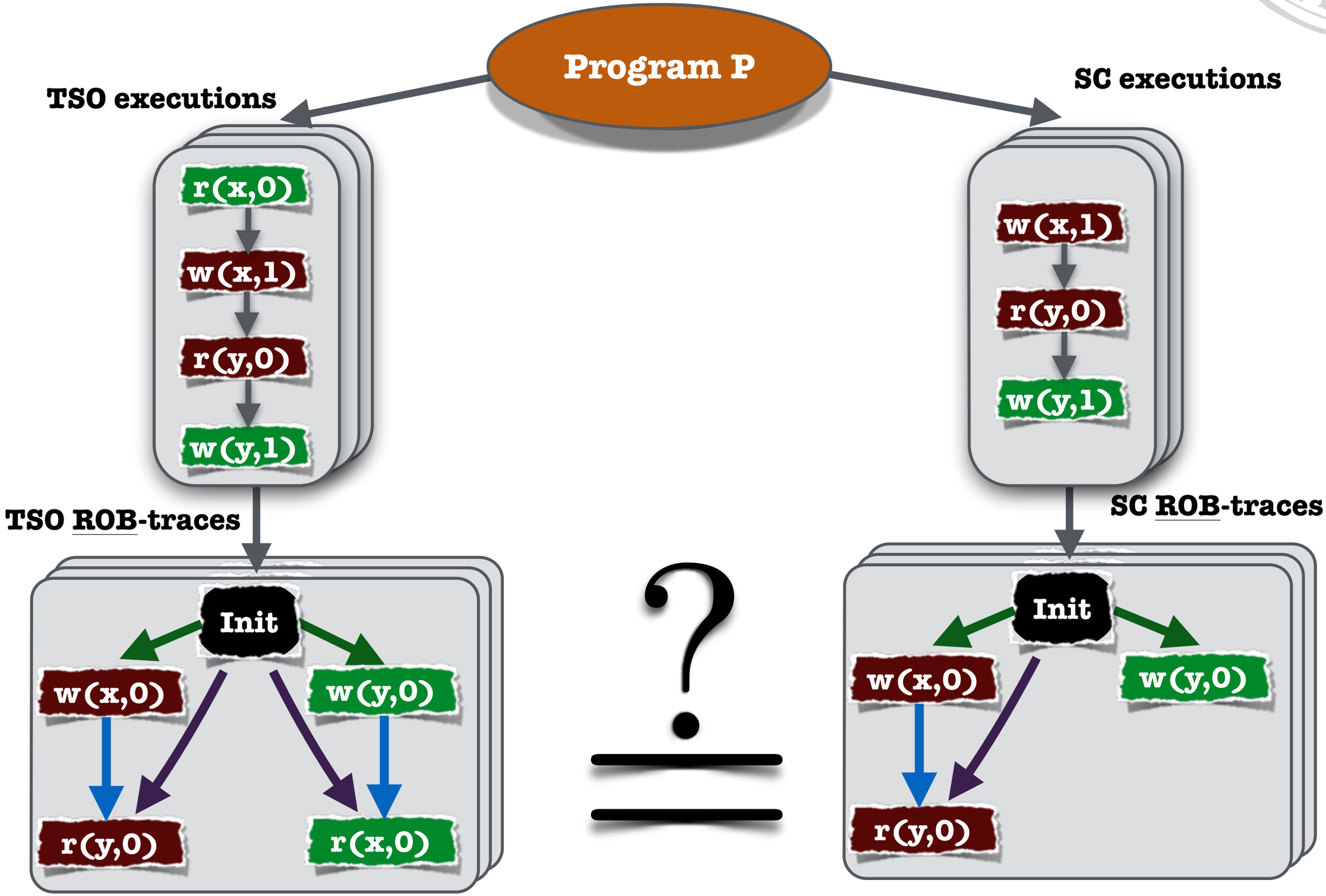
TRF Correctness Target



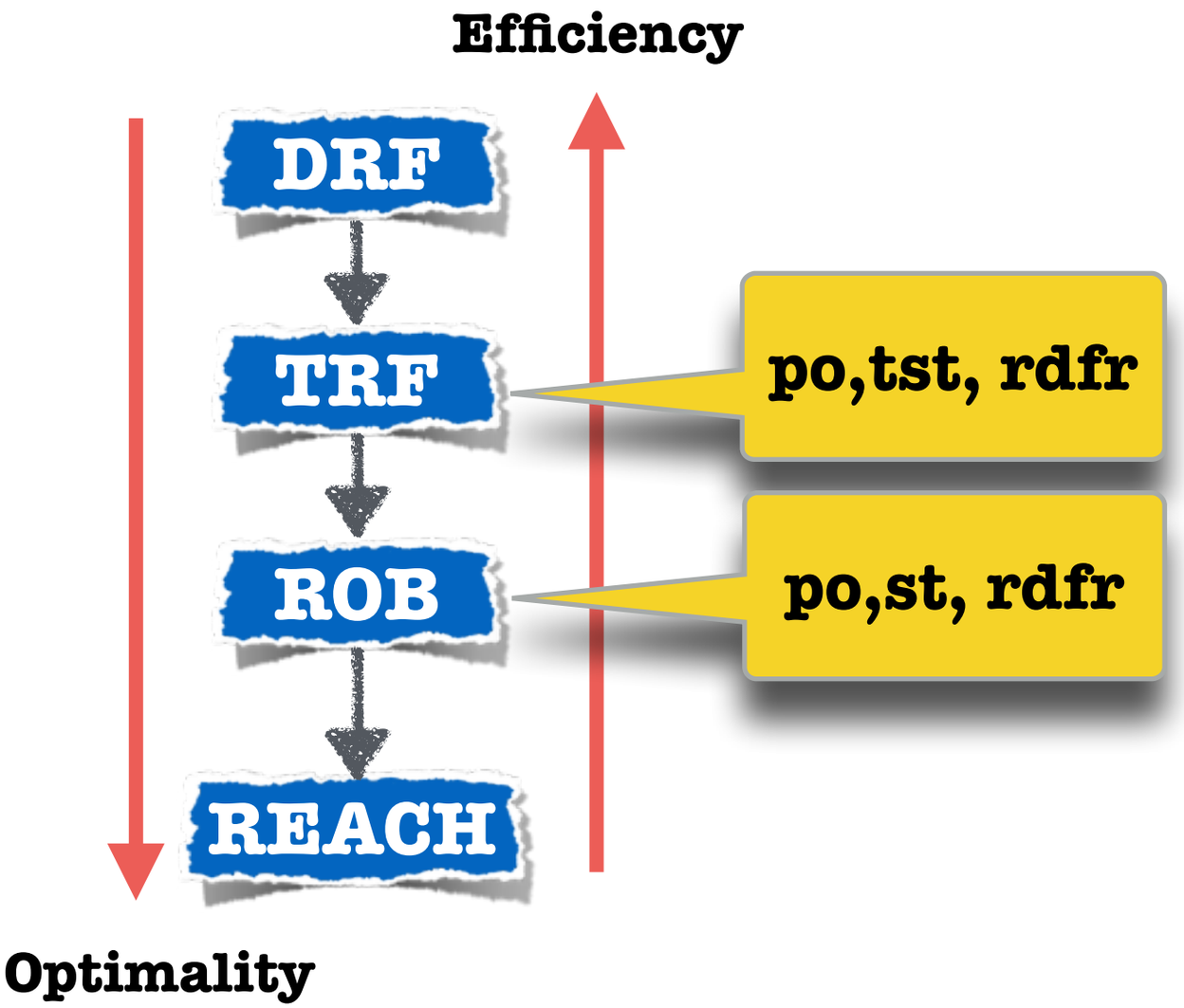
TRF Correctness Target



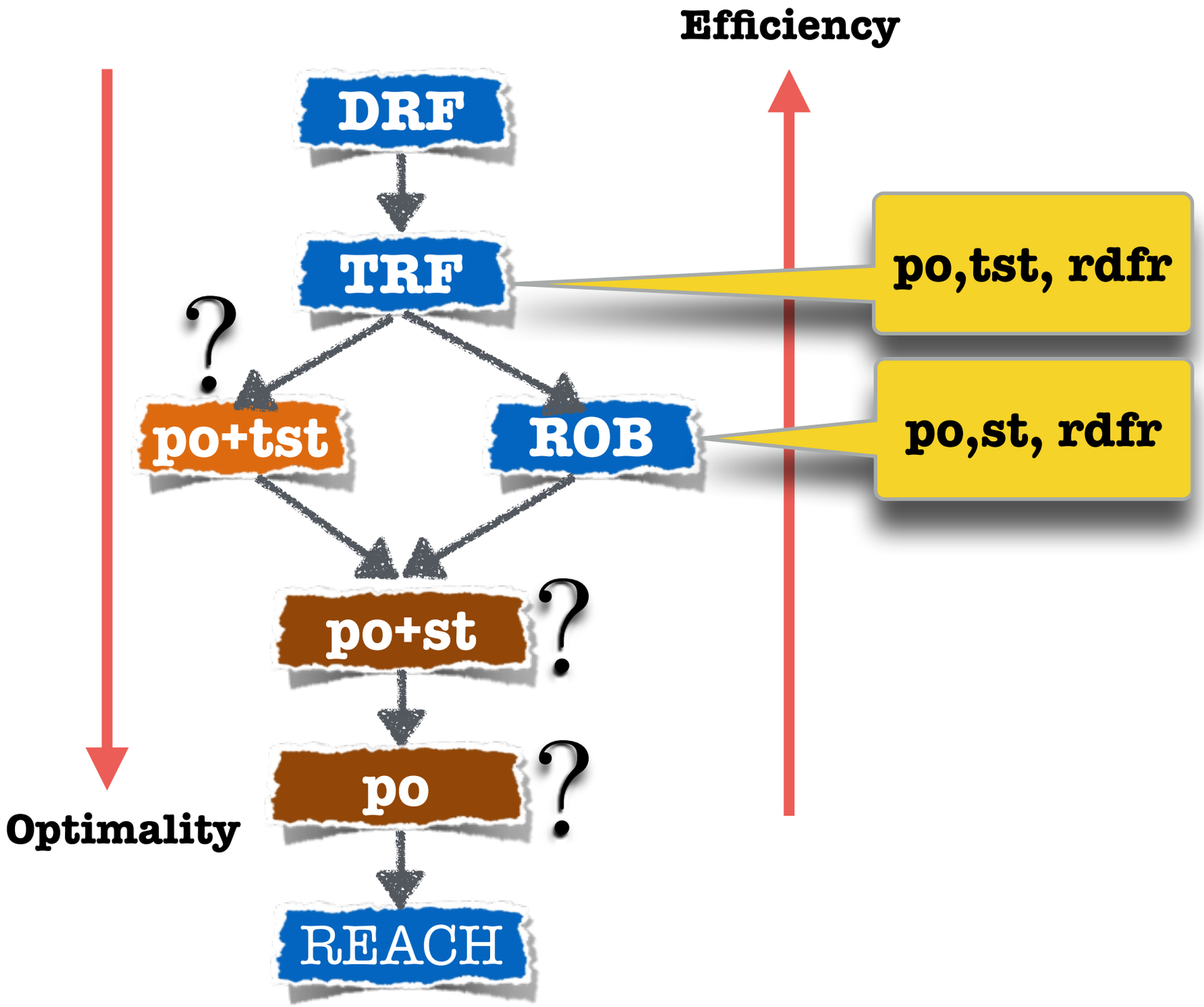
ROB Correctness Target



Specification Hierarchy



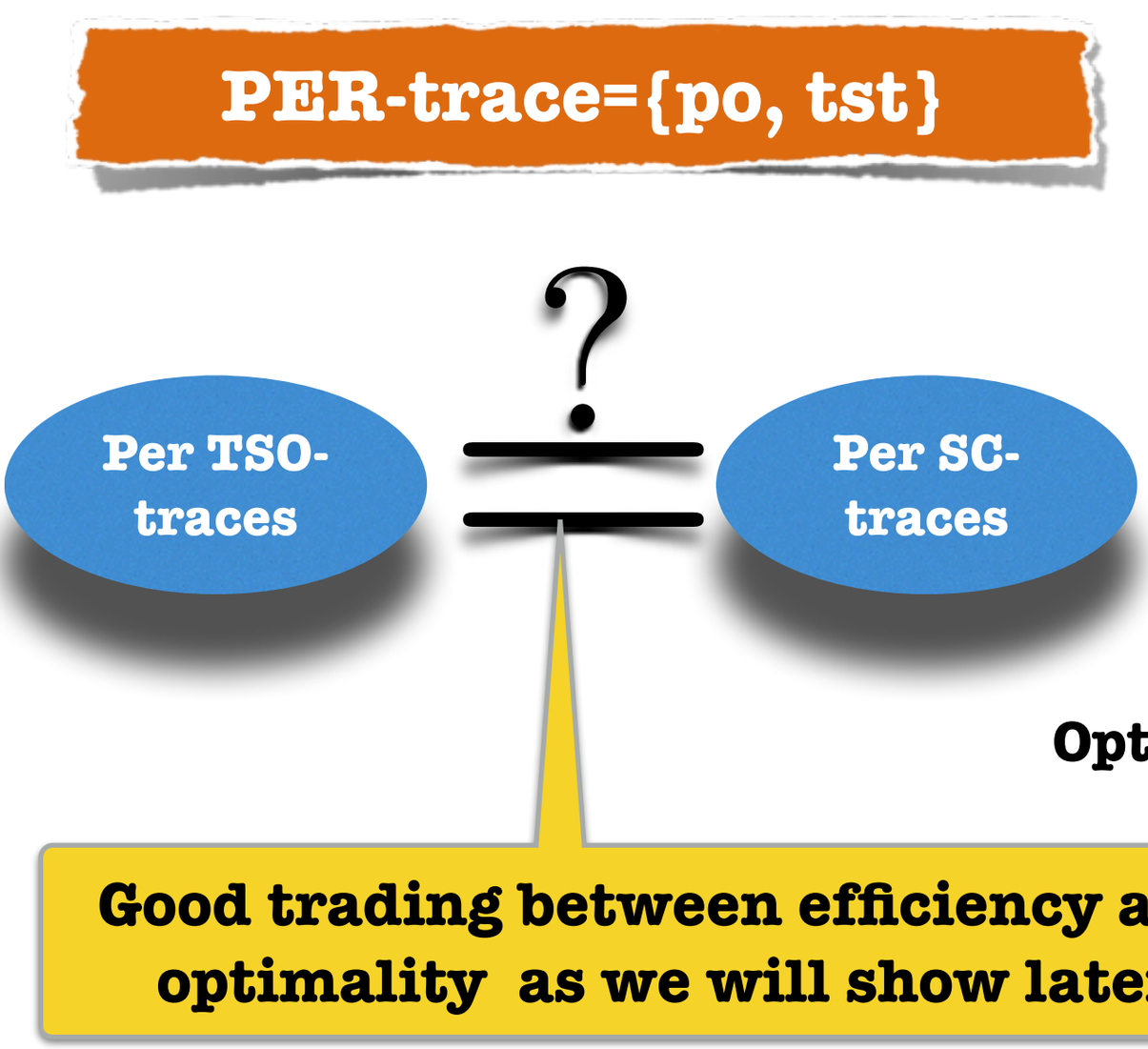
New Directions

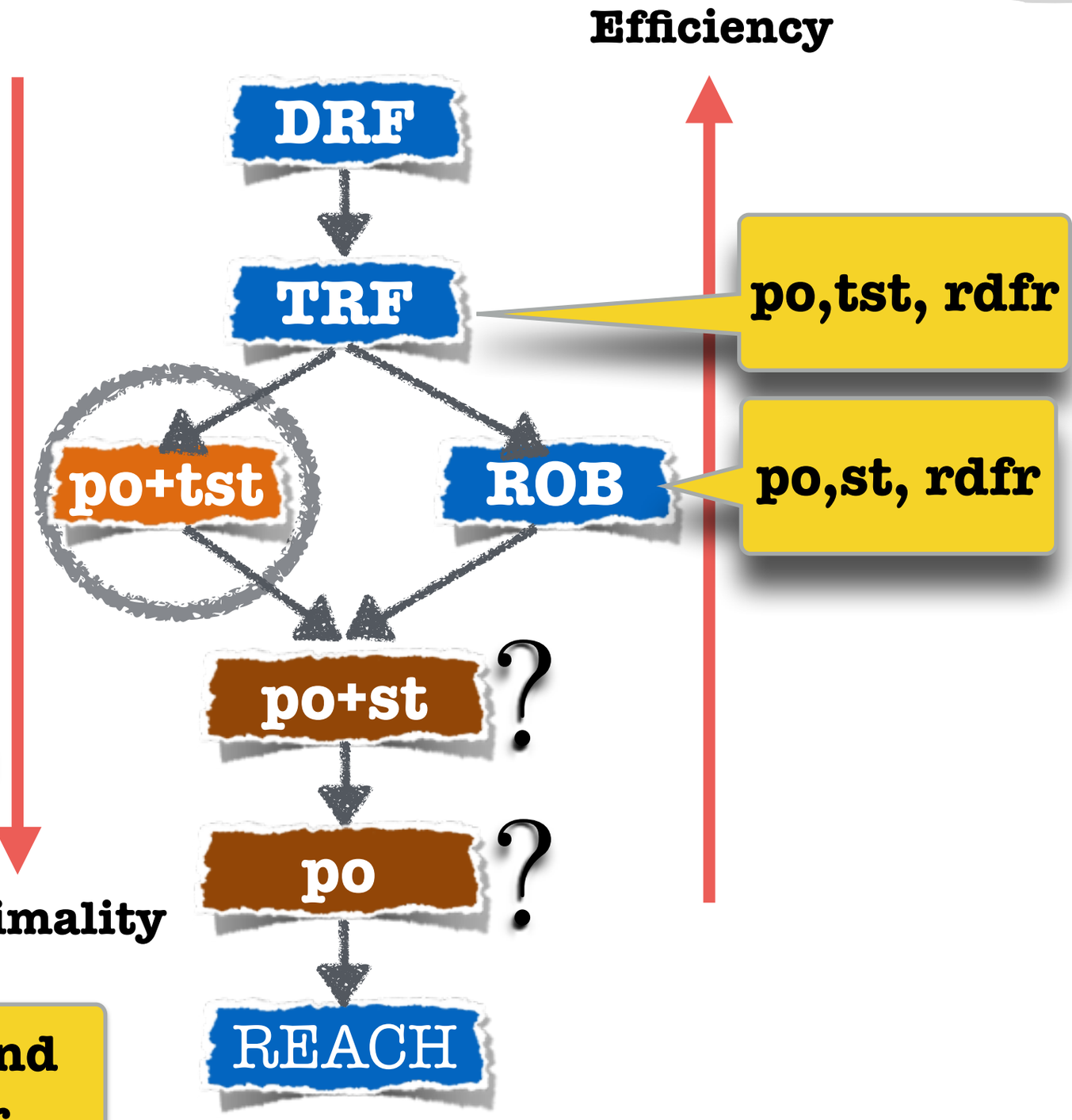


Persistence Specification

- **Goal:** study **(po, tst)** as stability specification

PER-trace={po, tst}





Our Contributions

- ✦ An Algorithm for Checking Trace-Persistence under TSO
- ✦ A Counter-Example Guided Fence Insertion Procedure
- ✦ Efficient Abstraction Techniques
- ✦ An Open Source Tool

Our Contributions

- ✦ An Algorithm for Checking Trace-Persistence under TSO
  - Reduce to SC reachability in an instrumented program
  - Source-to-source translation with linear overhead
  - Pspace-complete for finite-state programs

Checking Trace-Persistence

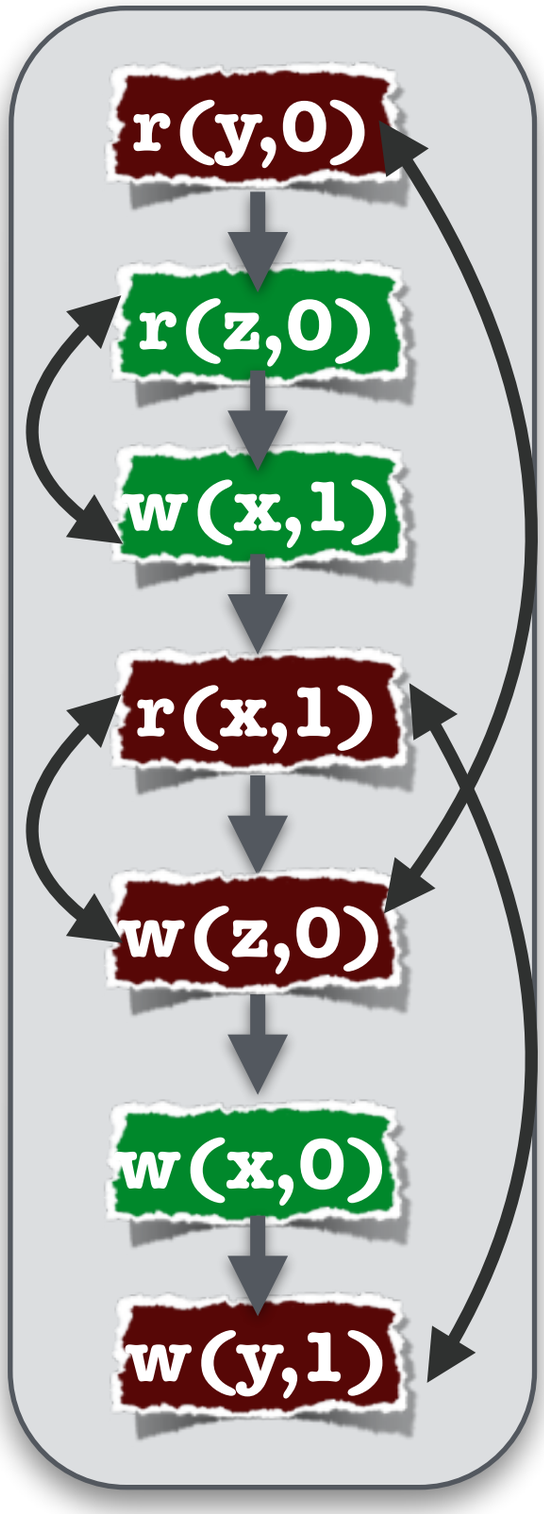
- Characterize the shape of minimal violations
- Check the existence of such minimal violations

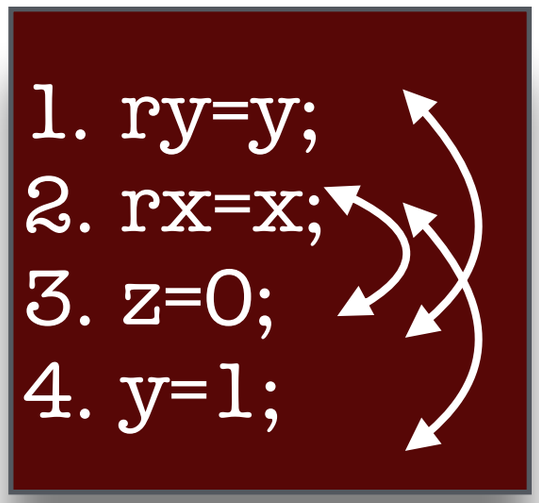
Minimal Violations

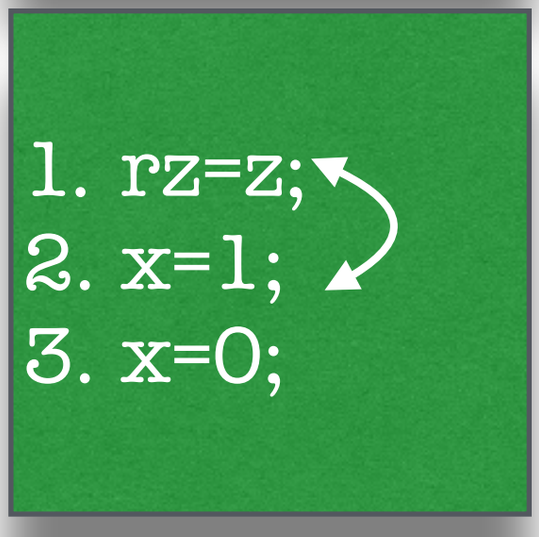
Init: x=y=z=0

- Minimal ⇒

  shortest violation



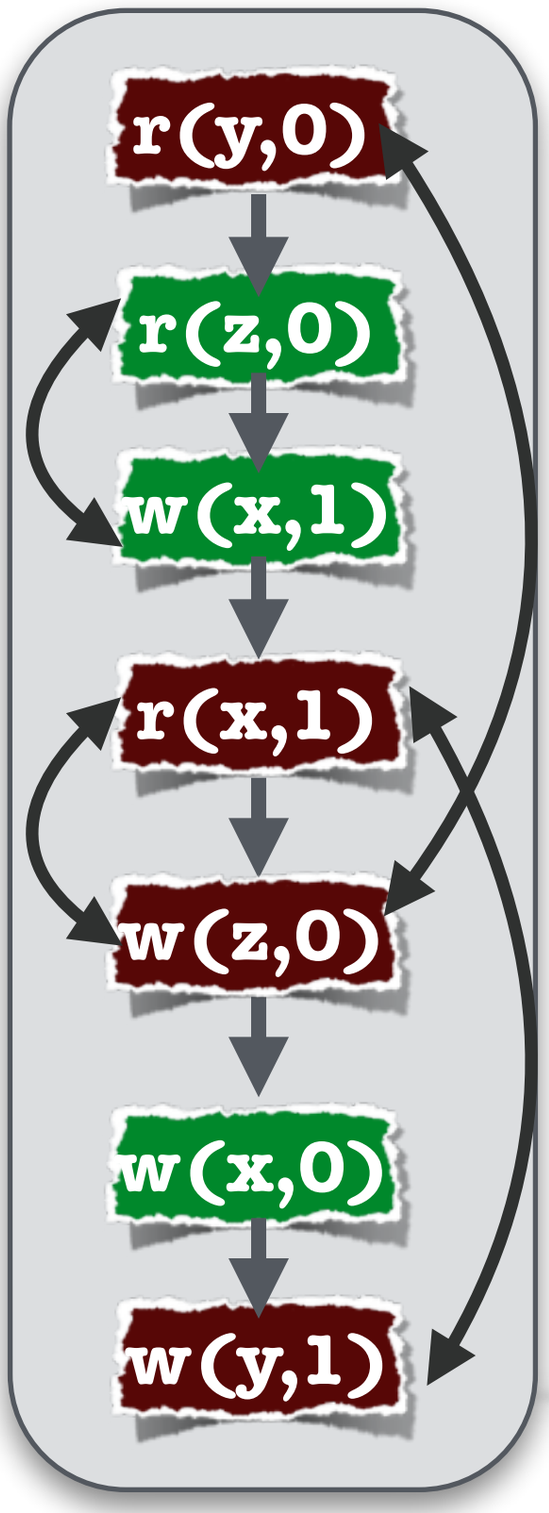




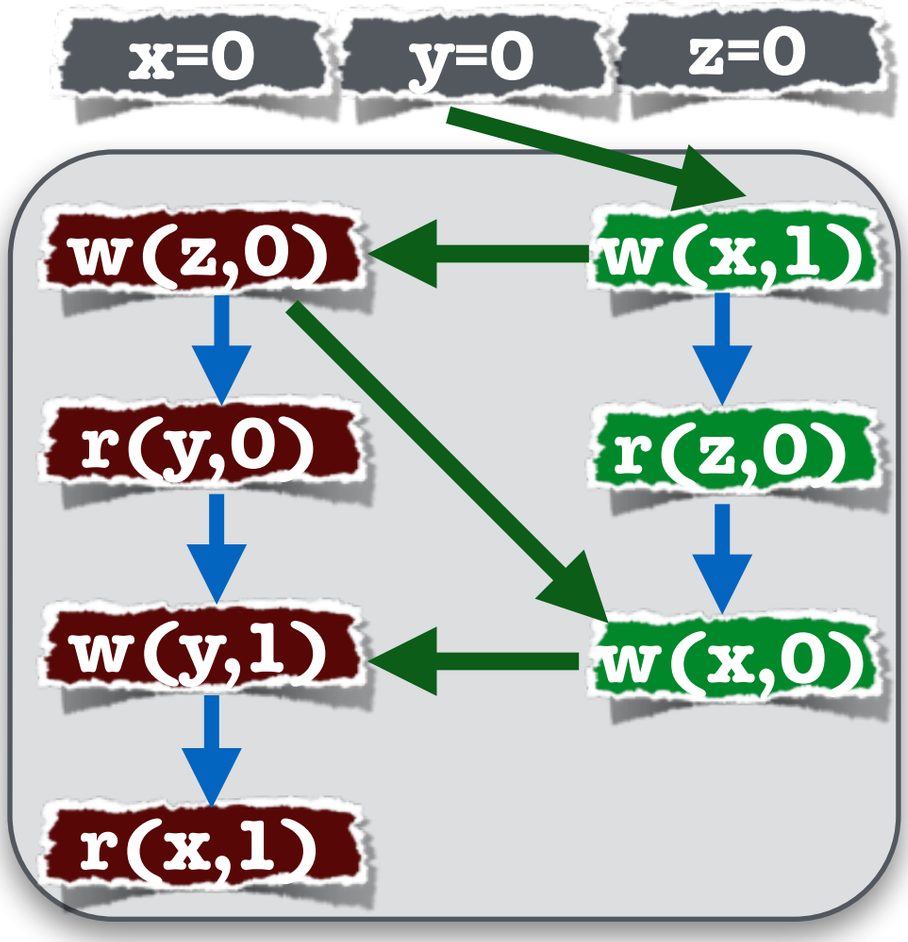
Minimal violation

Singly TSO Minimal Violations

Init: x=y=z=0



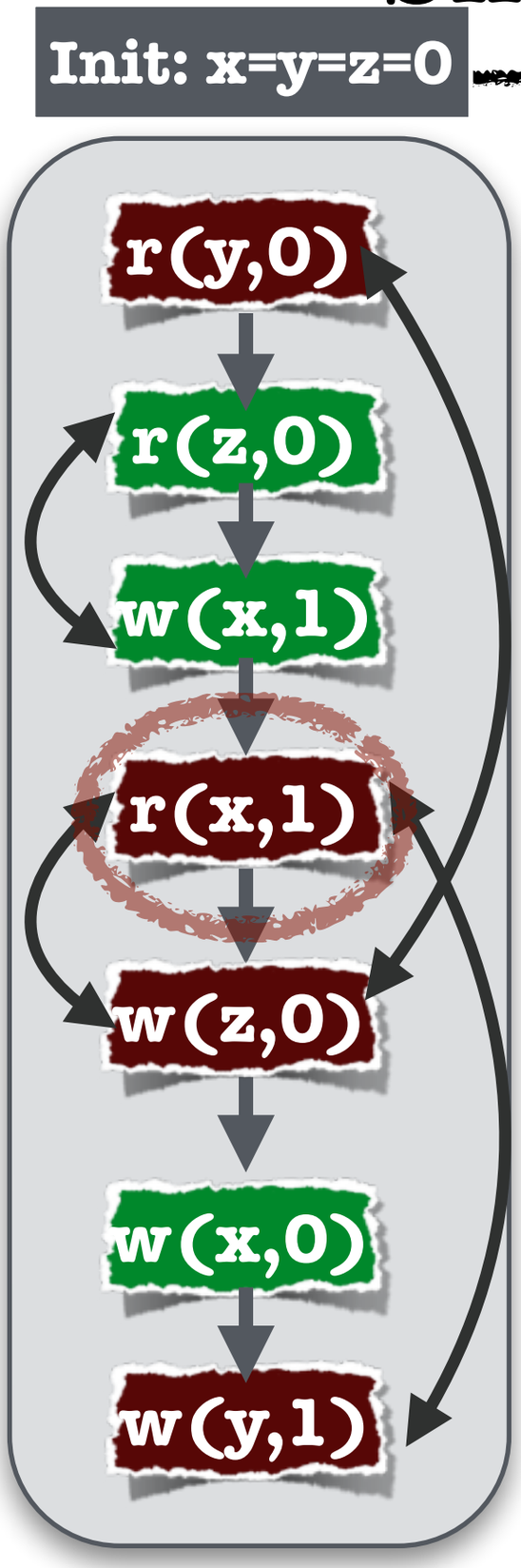
Minimal violation



PER-Trace

Theorem 1: We can construct a minimal violation where only one process (the pivot) uses rewriting

Singly TSO Minimal Violations

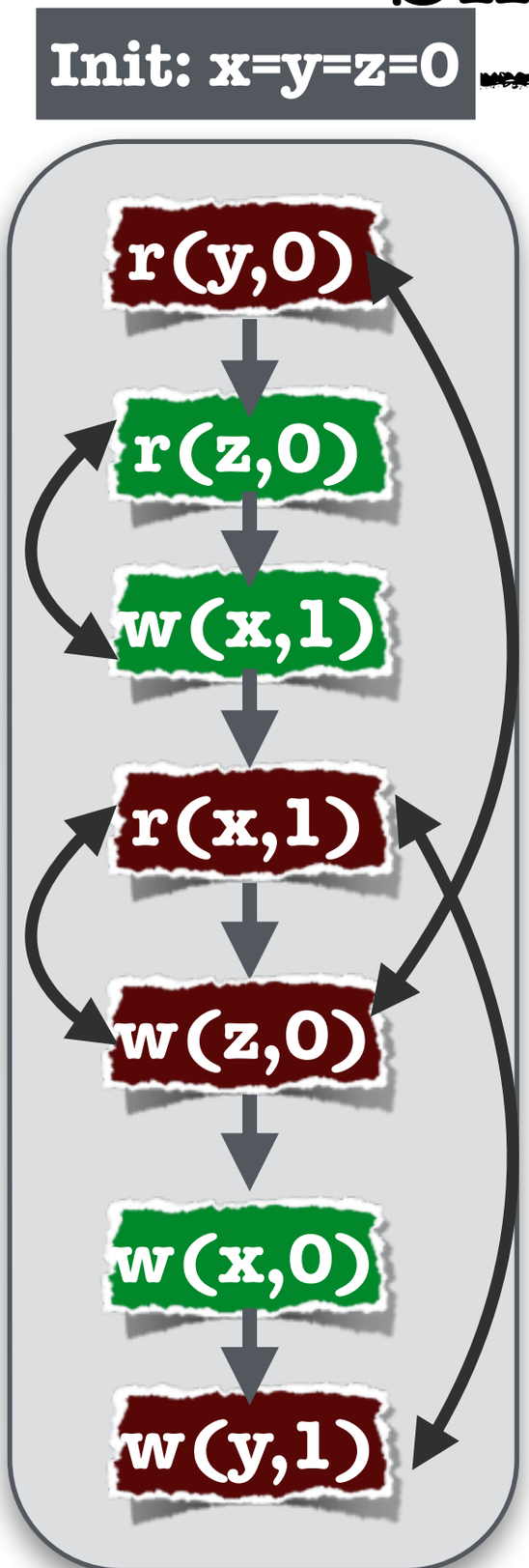


Minimal violation

Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**

Singly TSO Minimal Violations

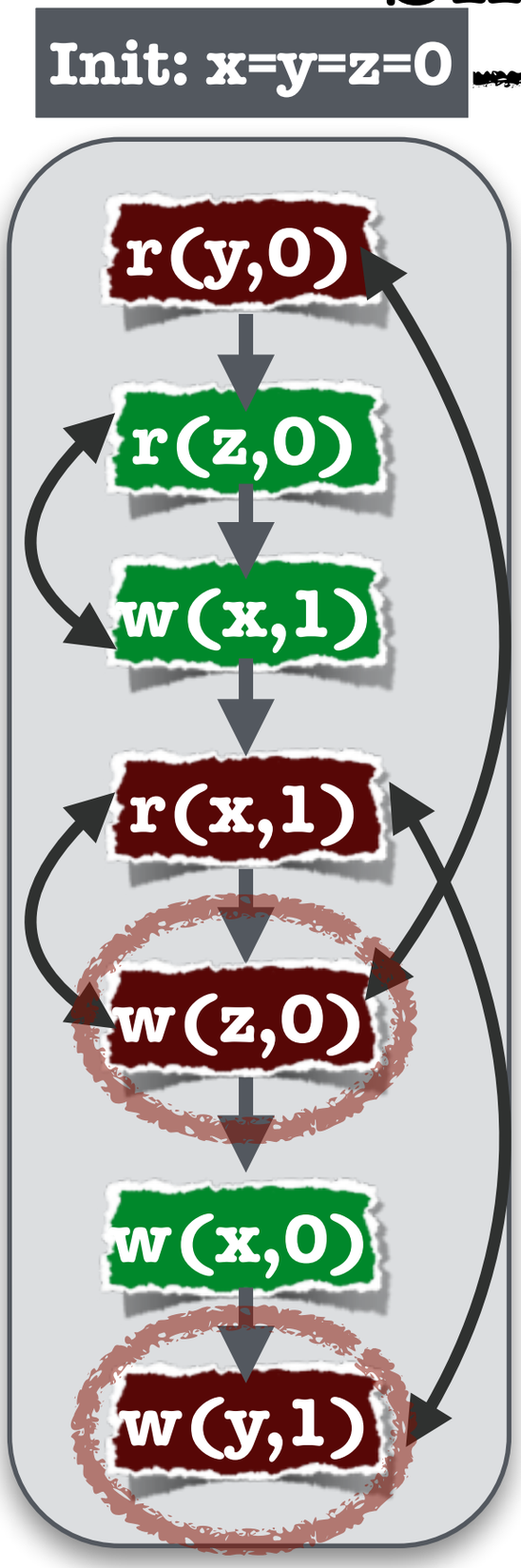


Minimal violation

Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**

Singly TSO Minimal Violations

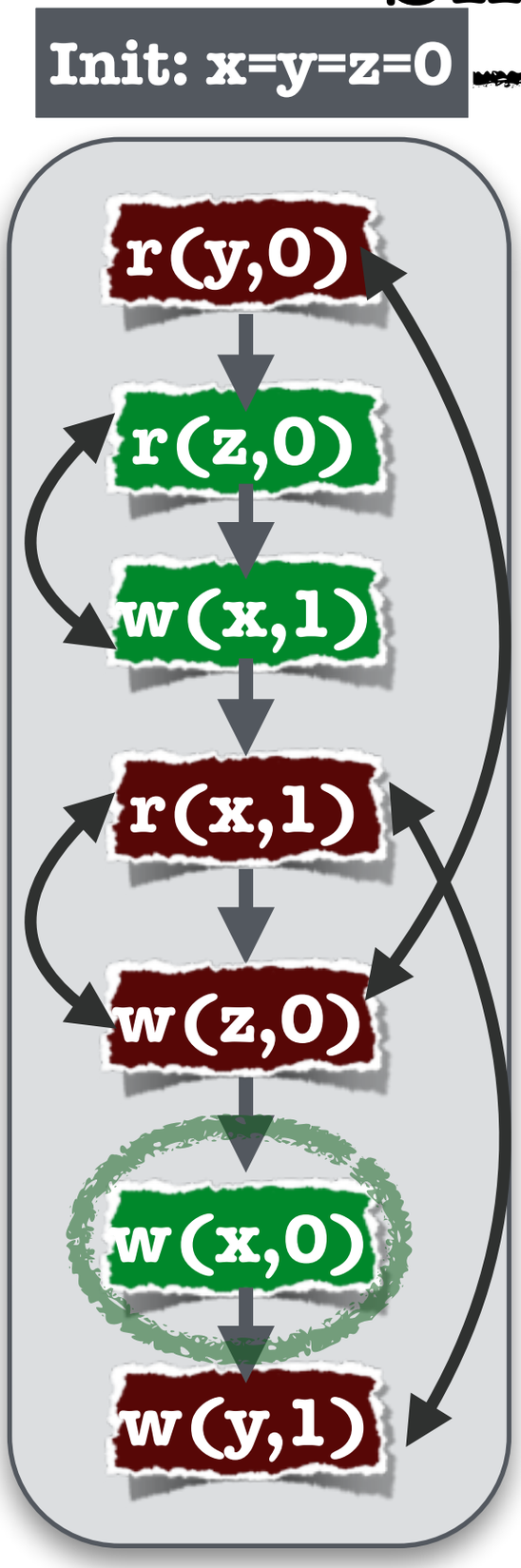


Minimal violation

Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**

Singly TSO Minimal Violations



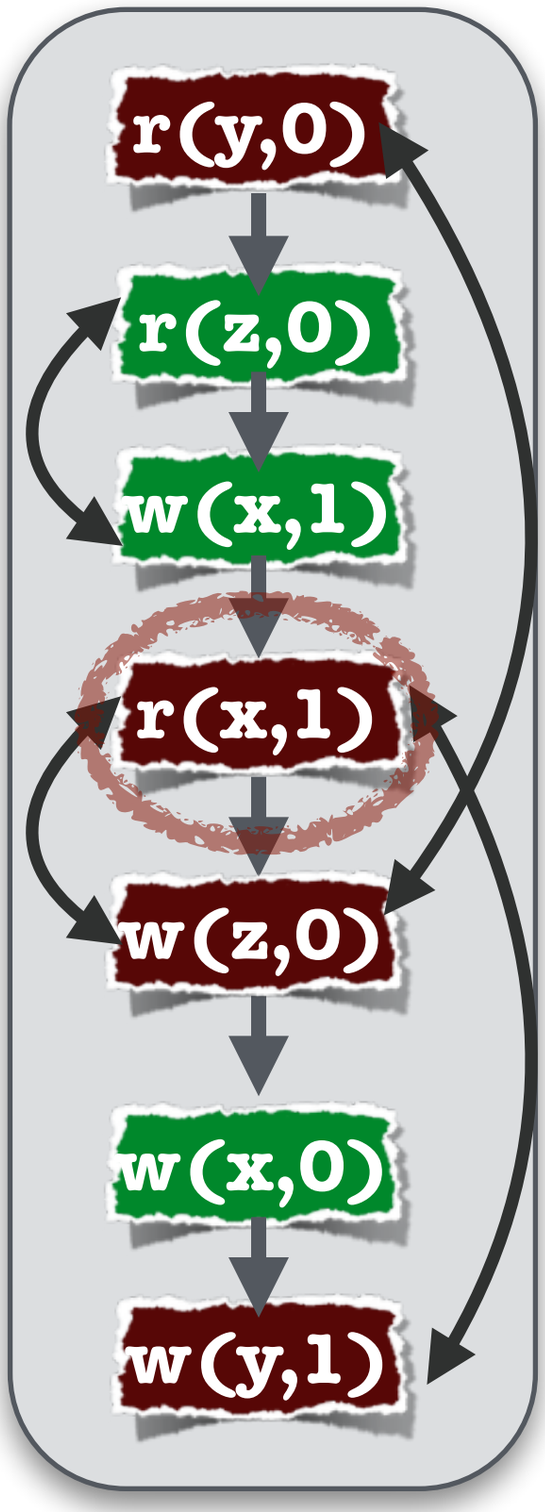
Minimal violation

Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**
  - ✦ **The last write on the variable of the read has different value**

Singly TSO Minimal Violations

Init: x=y=z=0

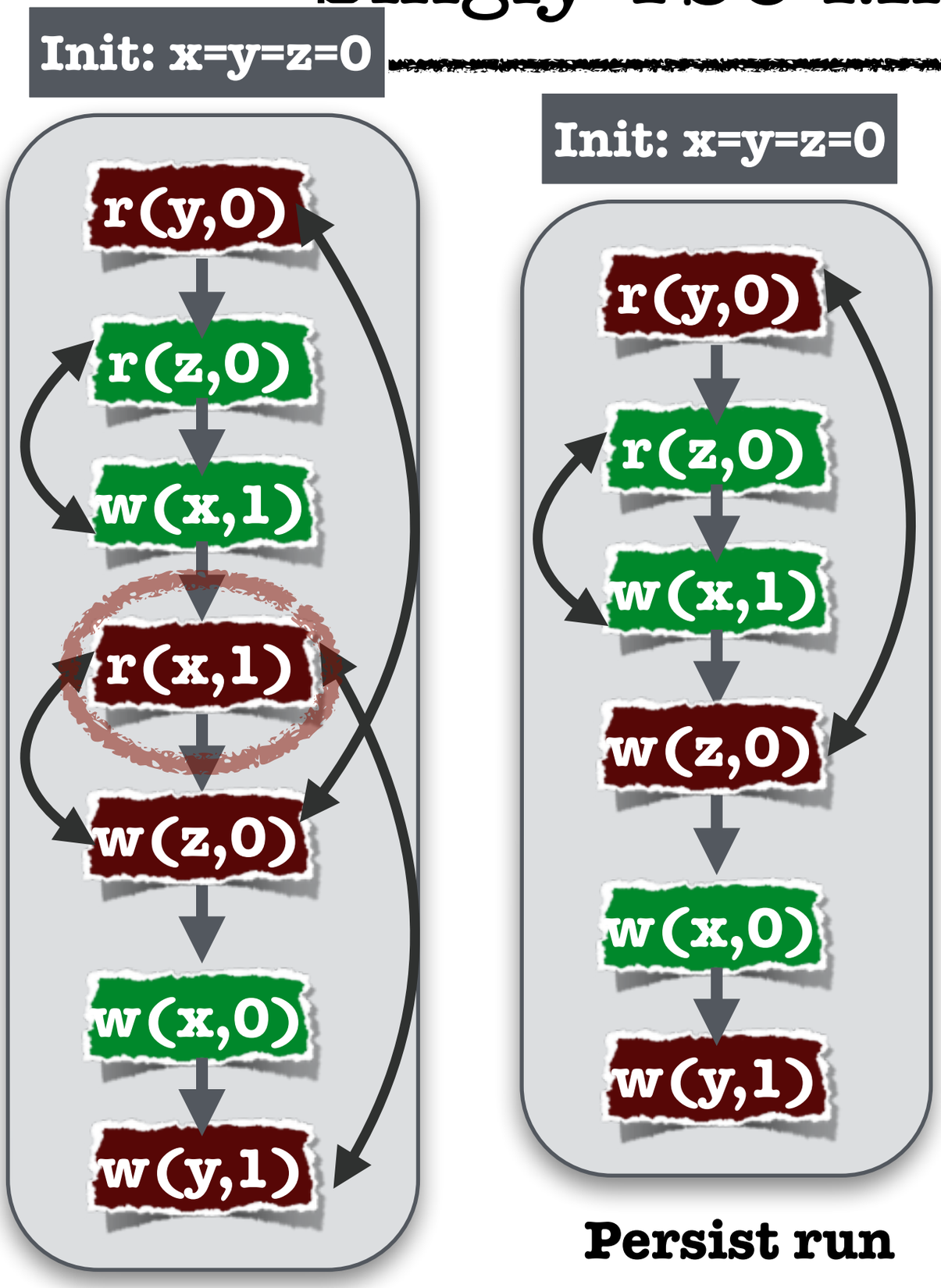


Minimal violation

Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**
  - ✦ **The last write on the variable of the read has different value**
- **Delete this read from the run**

Singly TSO Minimal Violations



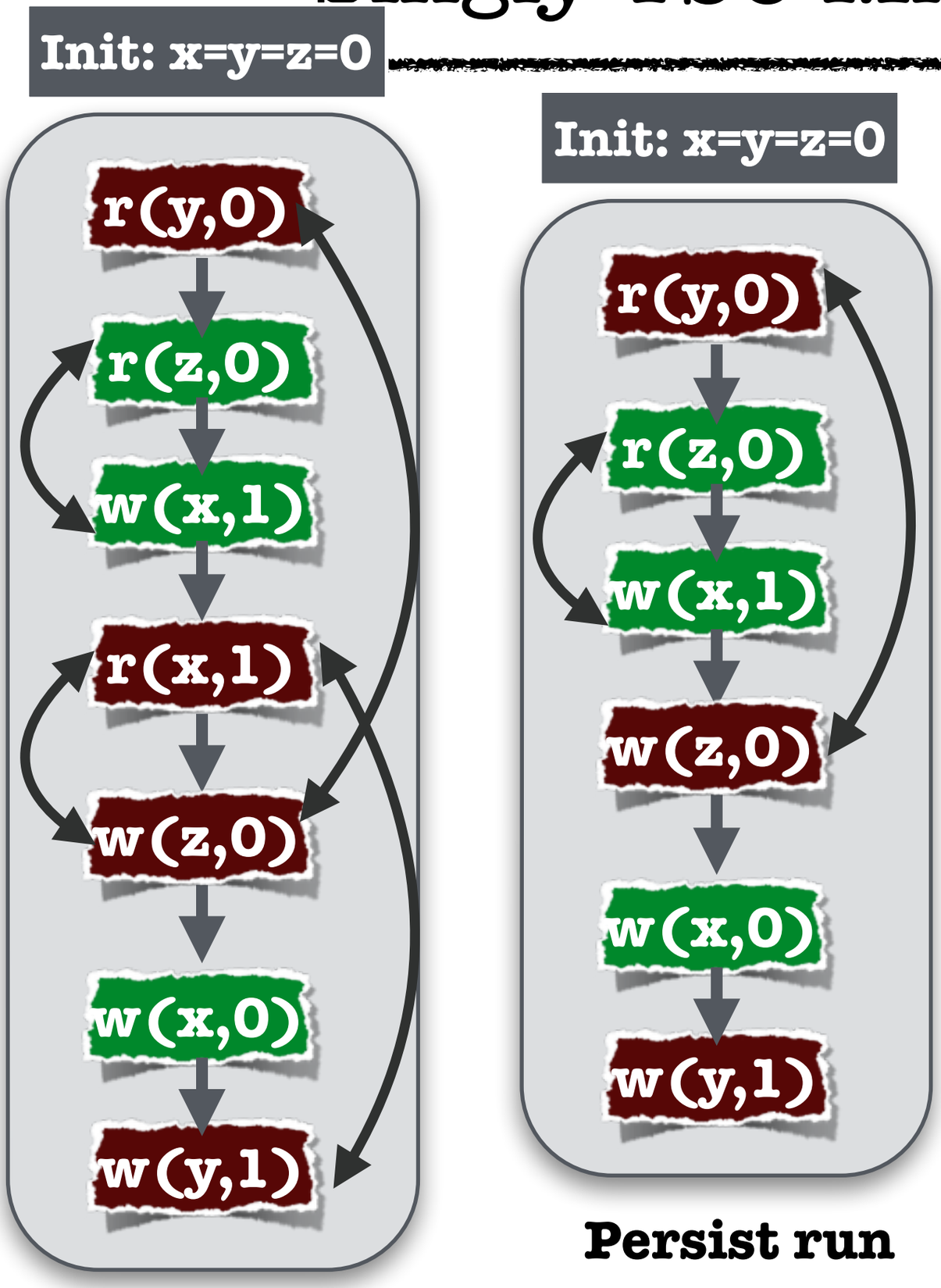
Minimal violation

Persist run

Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**
  - ✦ **The last write on the variable of the read has different value**
- **Delete this read from the run**

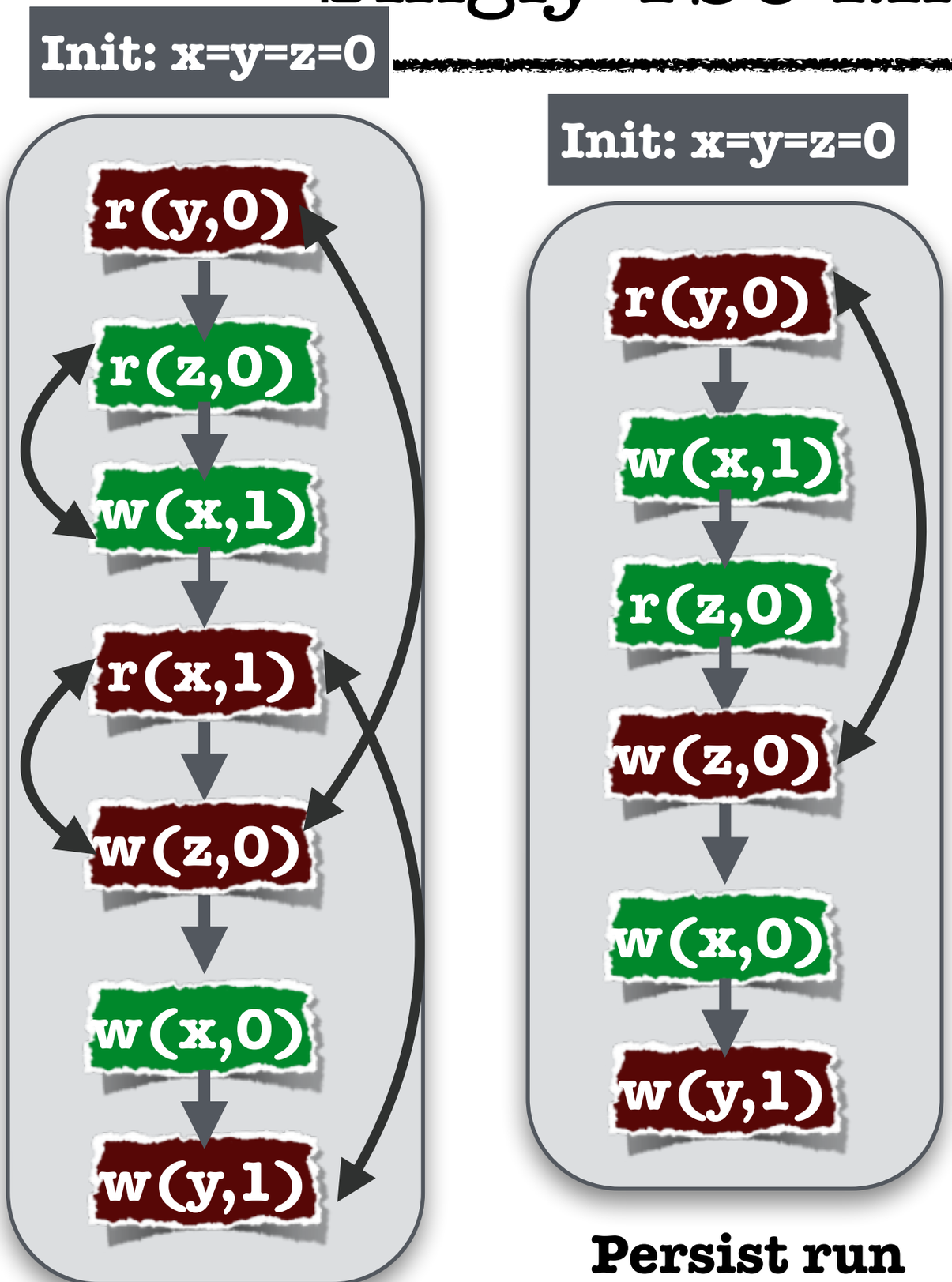
Singly TSO Minimal Violations



Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**
  - ✦ **The last write on the variable of the read has different value**
- **Delete this read from the run**
- **Construct the equivalent SC-run**

Singly TSO Minimal Violations



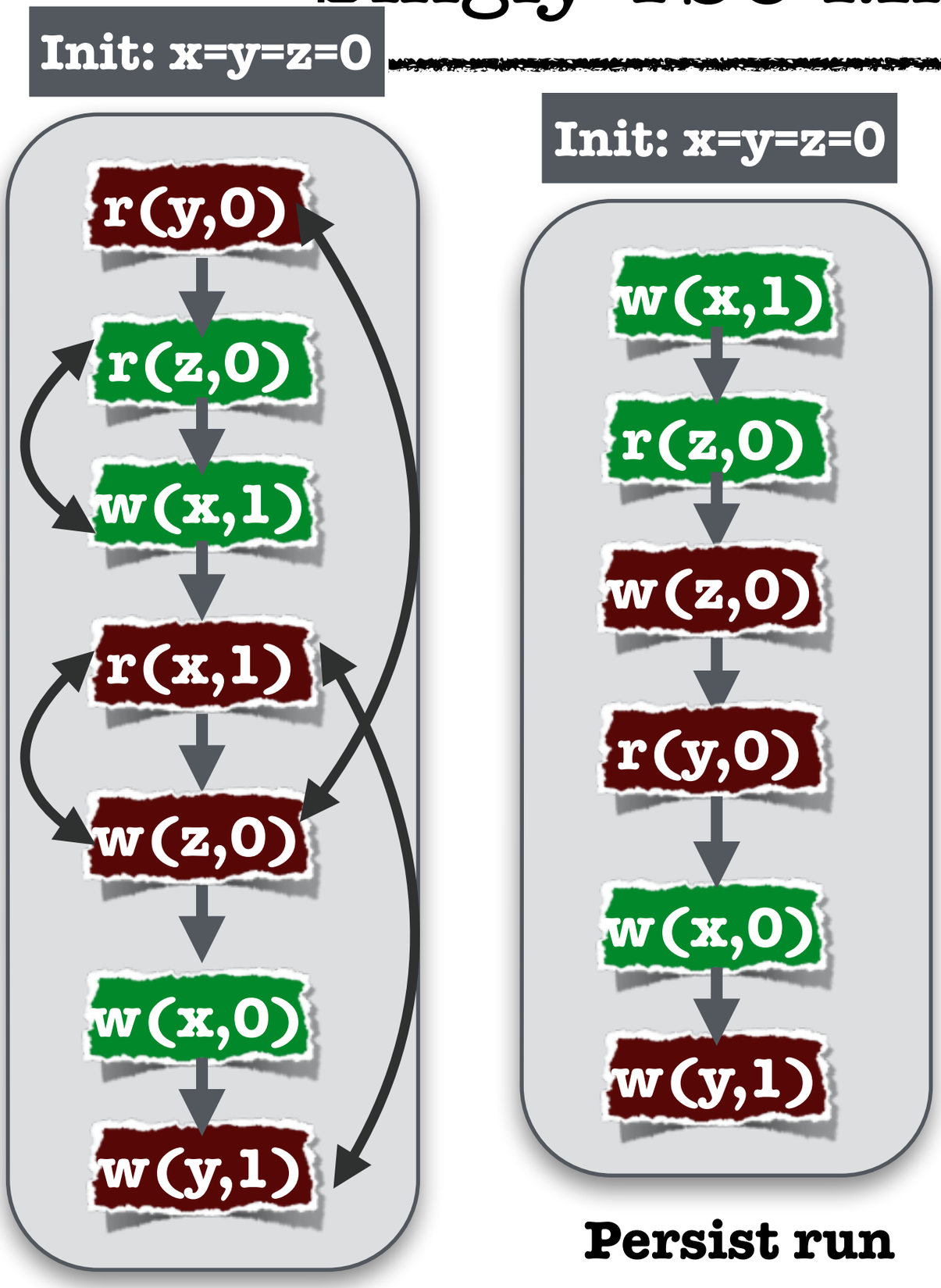
Minimal violation

Persist run

Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**
  - ✦ **The last write on the variable of the read has different value**
- **Delete this read from the run**
- **Construct the equivalent SC-run**

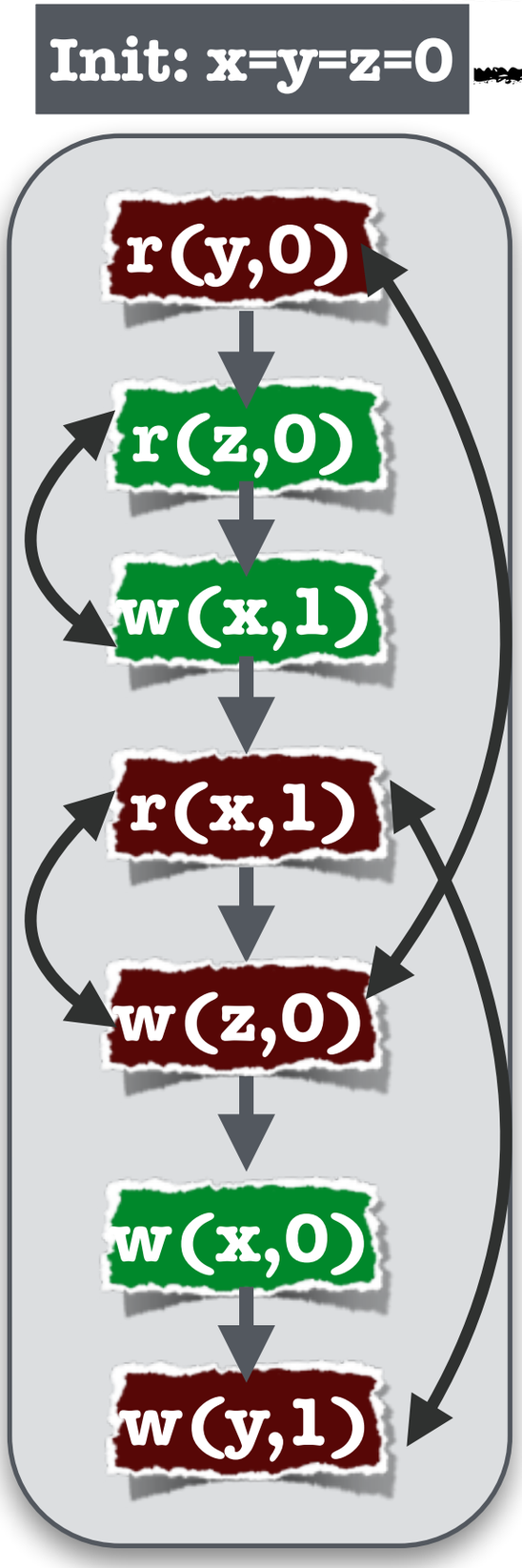
Singly TSO Minimal Violations



Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**
  - ✦ **The last write on the variable of the read has different value**
- **Delete this read from the run**
- **Construct the equivalent SC-run**

Singly TSO Minimal Violations



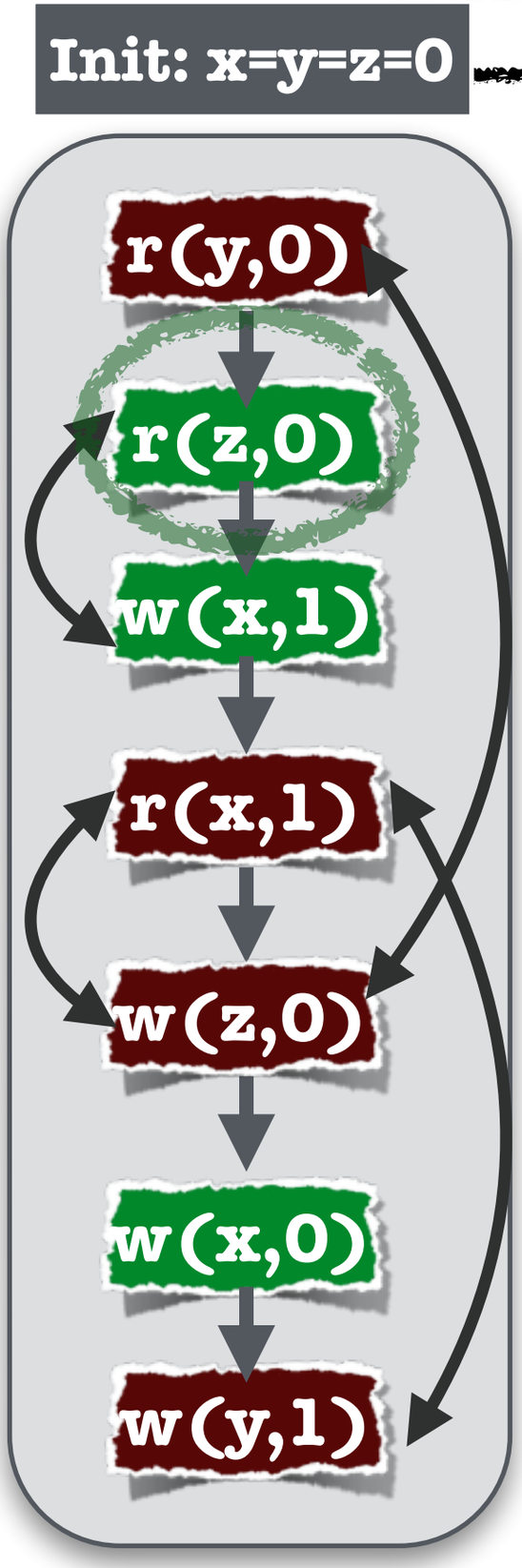
Minimal violation

Persist run

Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**
  - ✦ **The last write on the variable of the read has different value**
- **Delete this read from the run**
- **Construct the equivalent SC-run**
- **Replace the reads of other processes by the ones in the SC-run with same order**

Singly TSO Minimal Violations



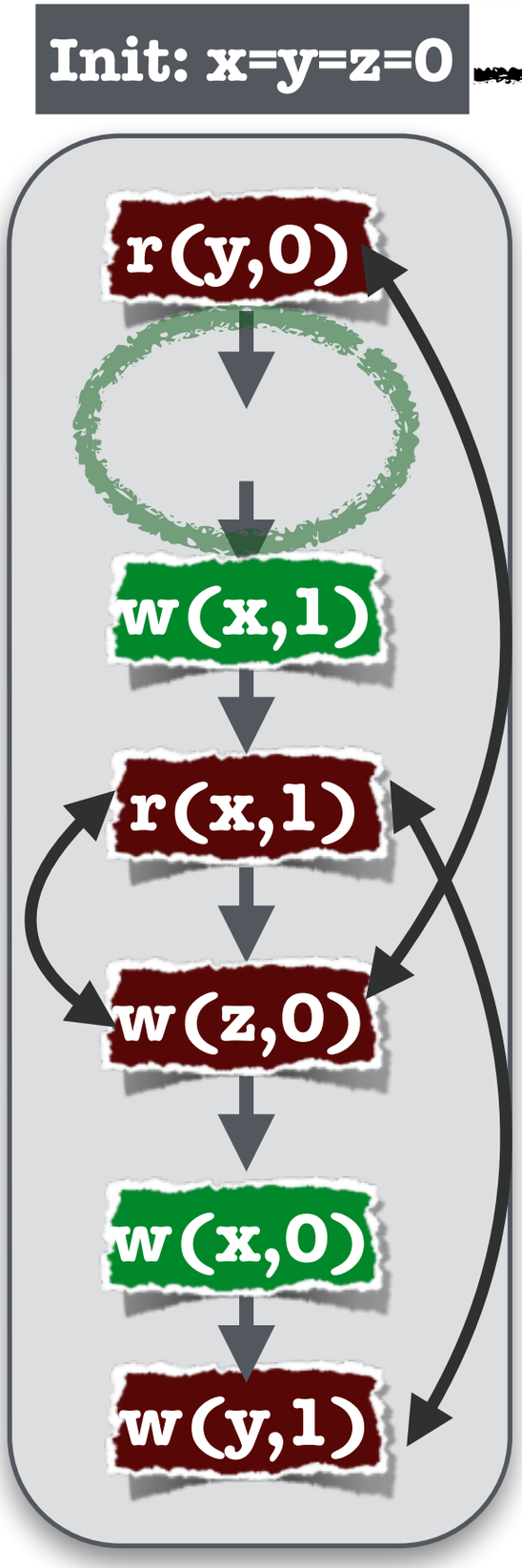
Minimal violation

Persist run

Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**
  - ✦ **The last write on the variable of the read has different value**
- **Delete this read from the run**
- **Construct the equivalent SC-run**
- **Replace the reads of other processes by the ones in the SC-run with same order**

Singly TSO Minimal Violations



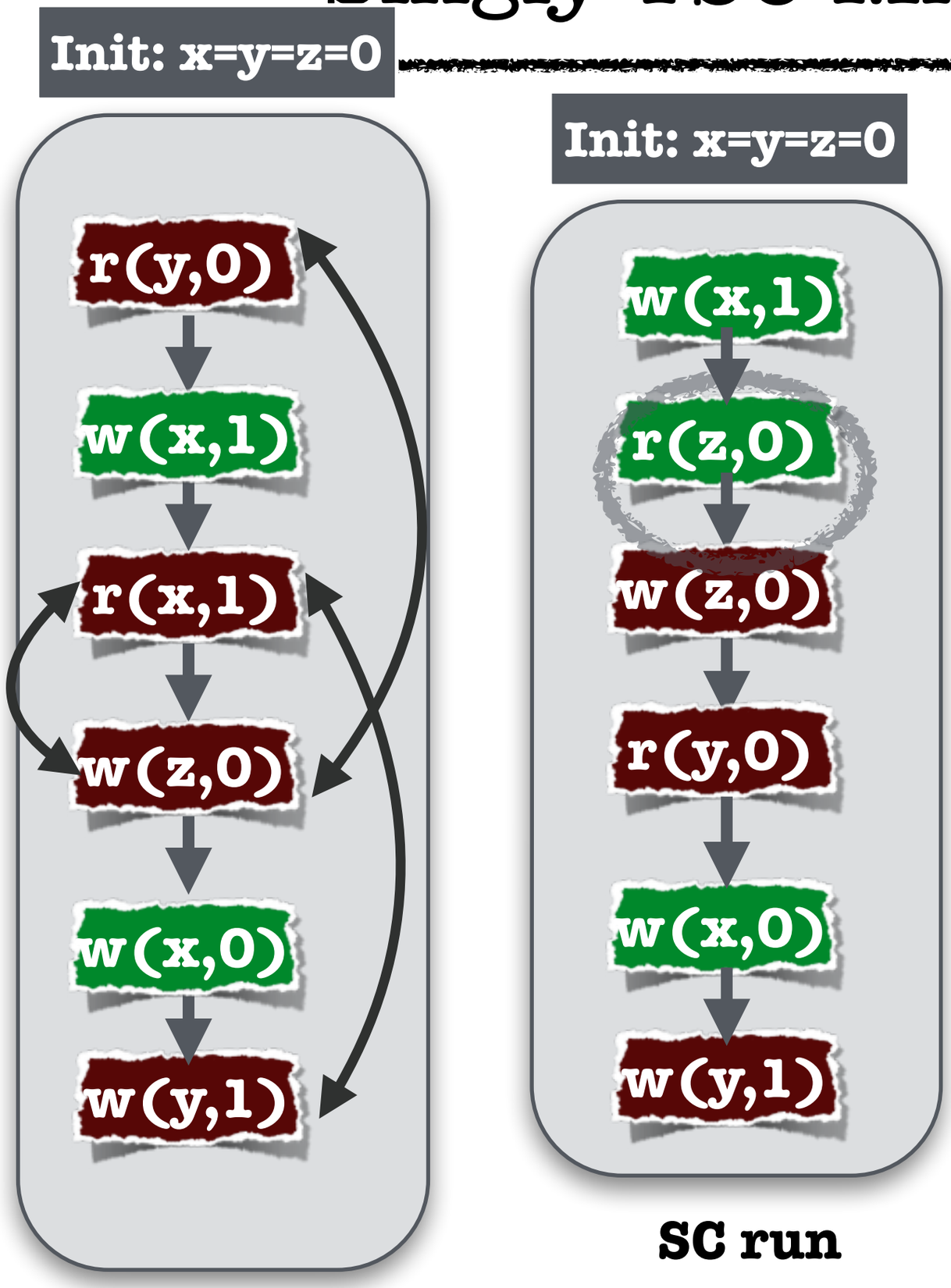
Minimal violation

SC run

Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**
  - ✦ **The last write on the variable of the read has different value**
- **Delete this read from the run**
- **Construct the equivalent SC-run**
- **Replace the reads of other processes by the ones in the SC-run with same order**

Singly TSO Minimal Violations



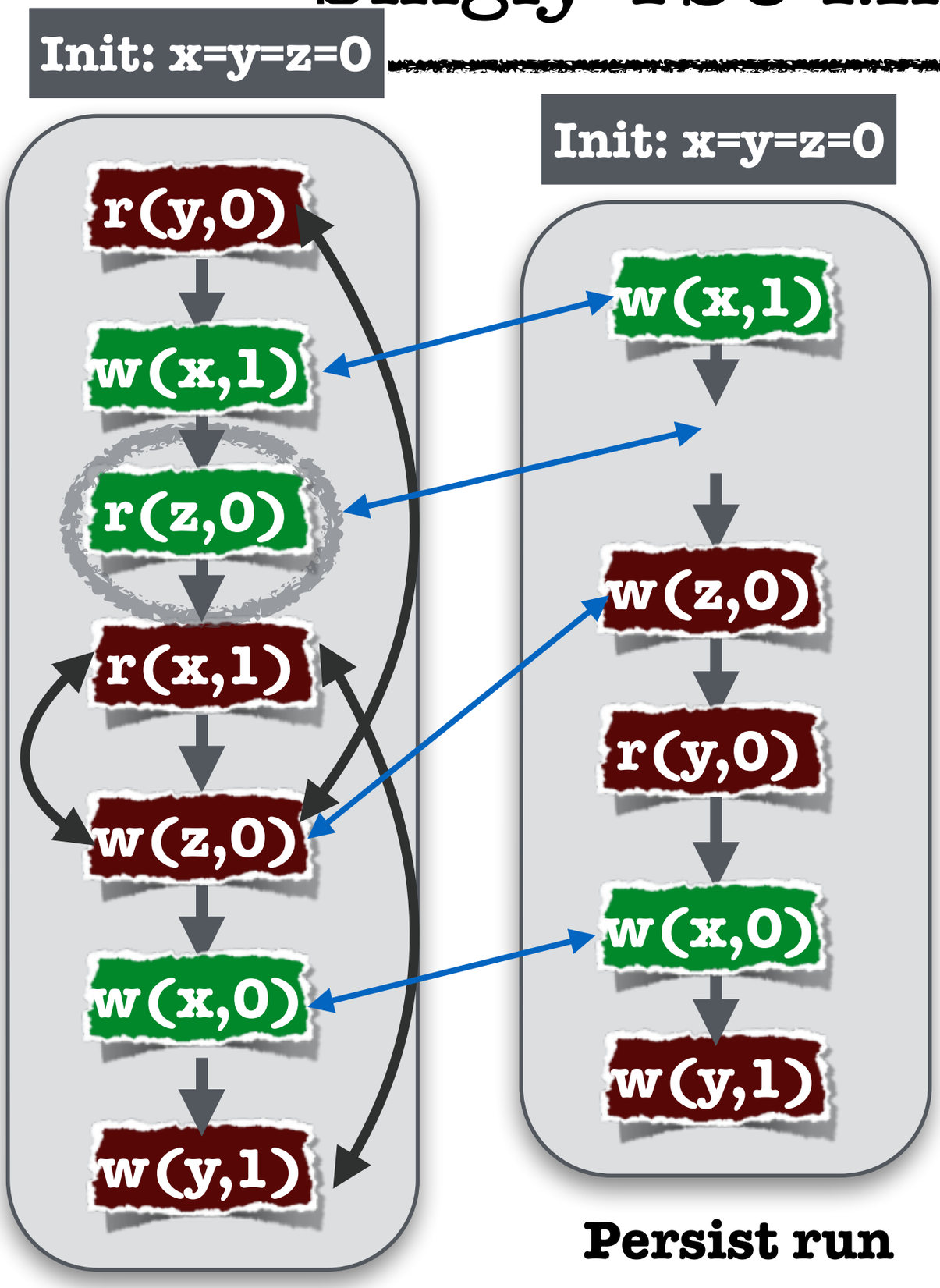
Minimal violation

SC run

Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**
  - ✦ **The last write on the variable of the read has different value**
- **Delete this read from the run**
- **Construct the equivalent SC-run**
- **Replace the reads of other processes by the ones in the SC-run with same order**

Singly TSO Minimal Violations



Persist run

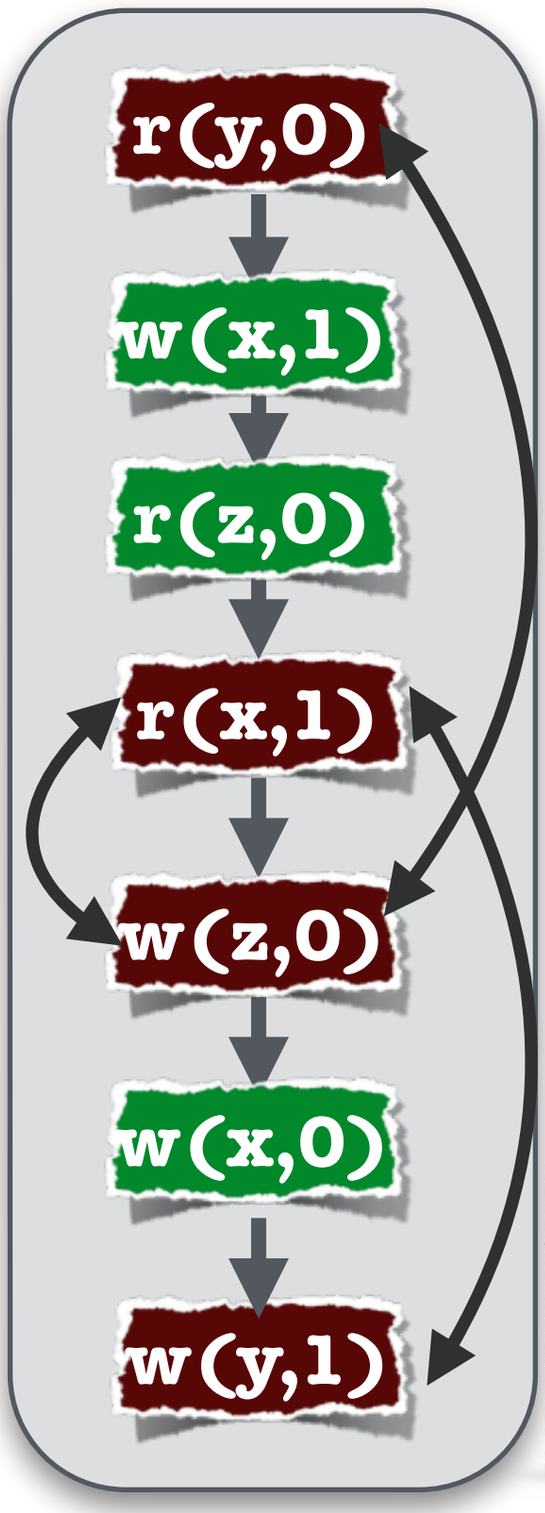
Singly TSO Minimal violation

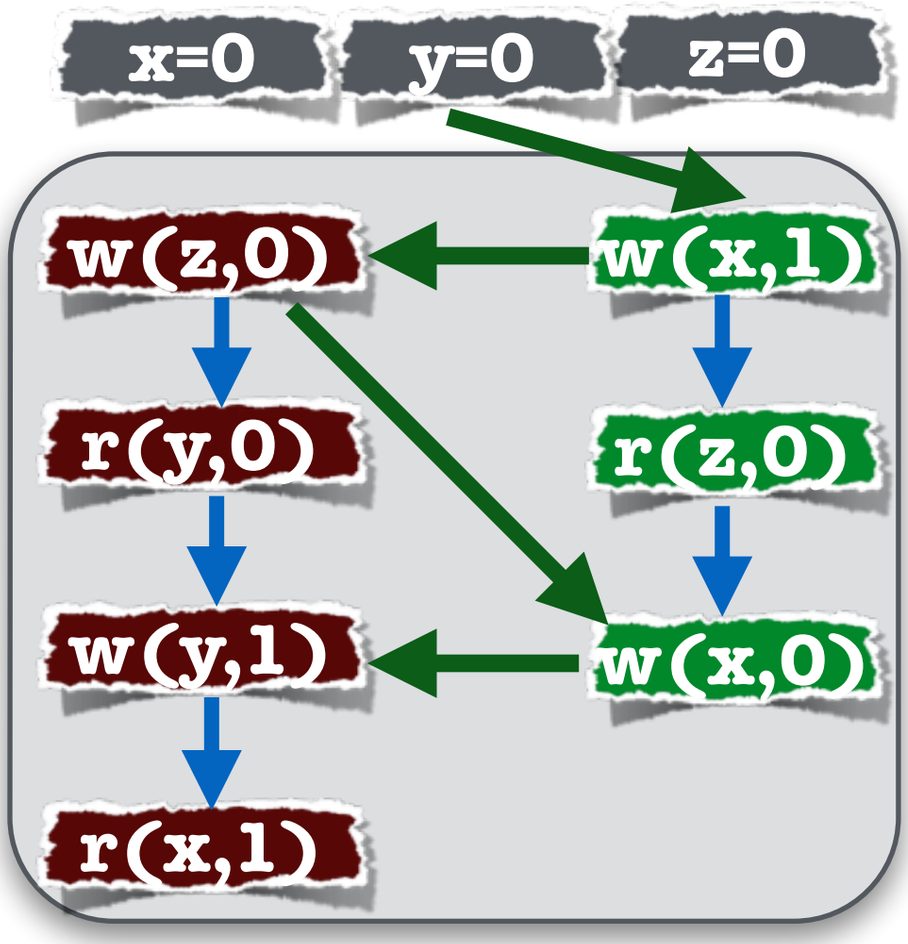
Proof sketch:

- **Find a pivot process s.t.:**
  - ✦ **This process ends with a read**
  - ✦ **This read delays some writes**
  - ✦ **The last write on the variable of the read has different value**
- **Delete this read from the run**
- **Construct the equivalent SC-run**
- **Replace the reads of other processes by the ones in the SC-run with same order**

Minimal Violations

Init: x=y=z=0



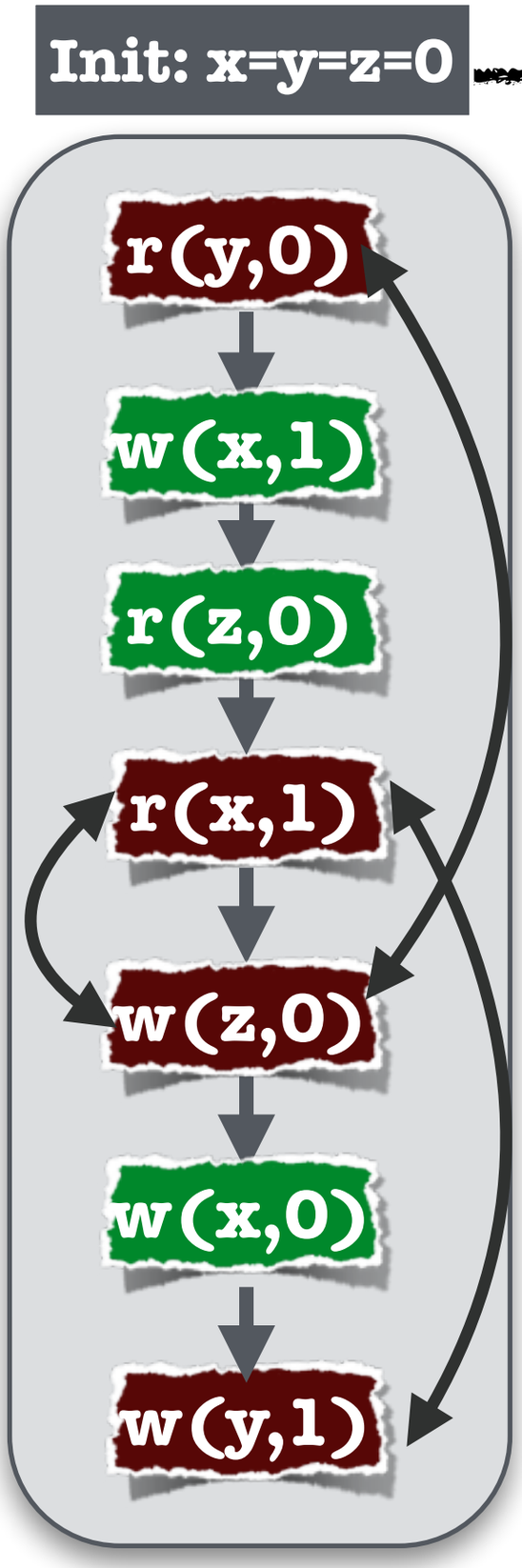


PER-Trace

Theorem 2: There is a singly TSO minimal violation where any overtaken write is also overtaken by the last read

Singly TSO Minimal violation

Violation Shape

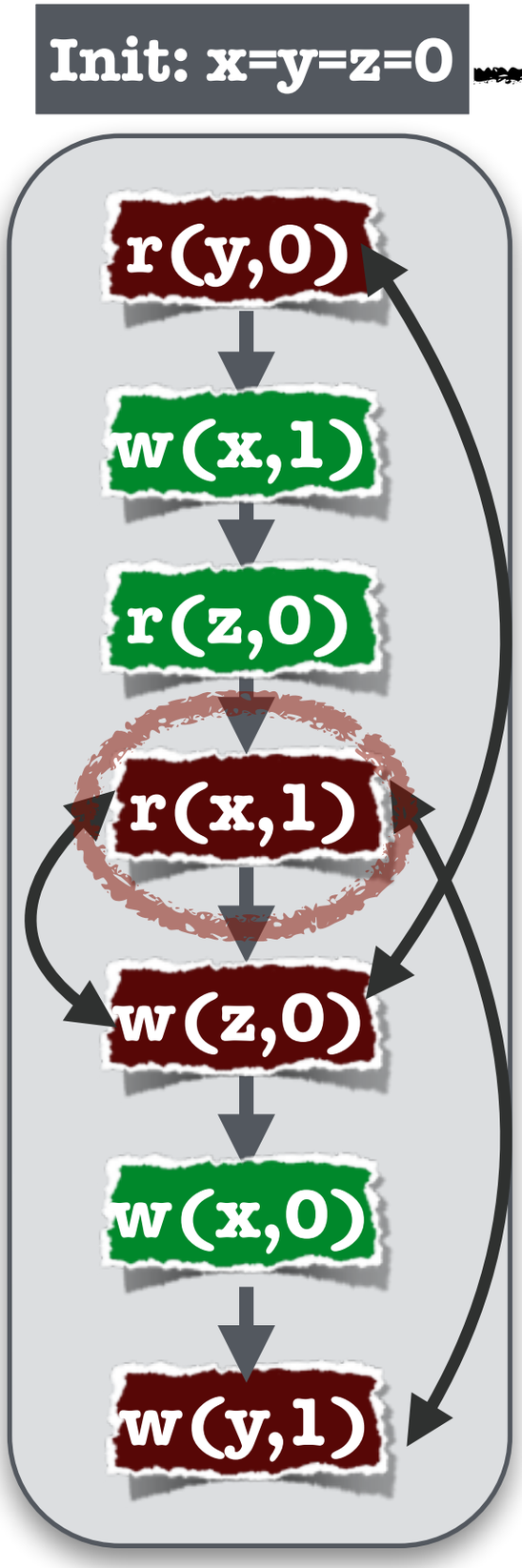


Proof sketch:

- **Remove all the actions of other processes occurring after the last read**

Singly TSO Minimal violation 1

Violation Shape

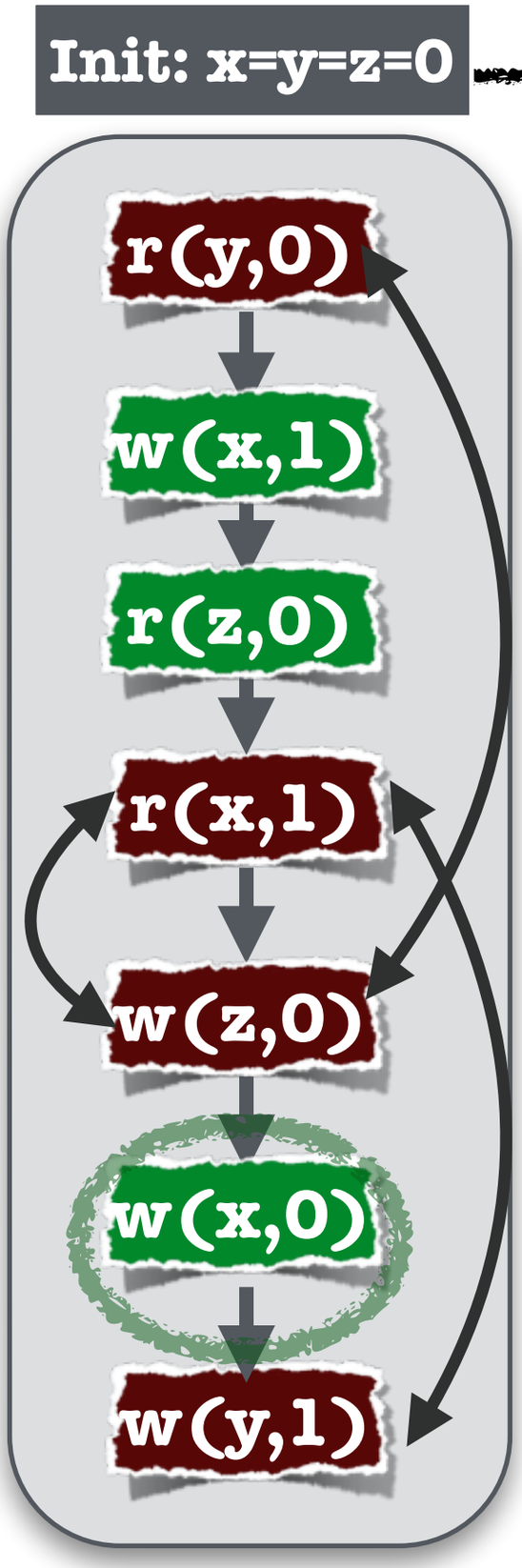


Proof sketch:

- **Remove all the actions of other processes occurring after the last read**

Singly TSO Minimal violation 1

Violation Shape

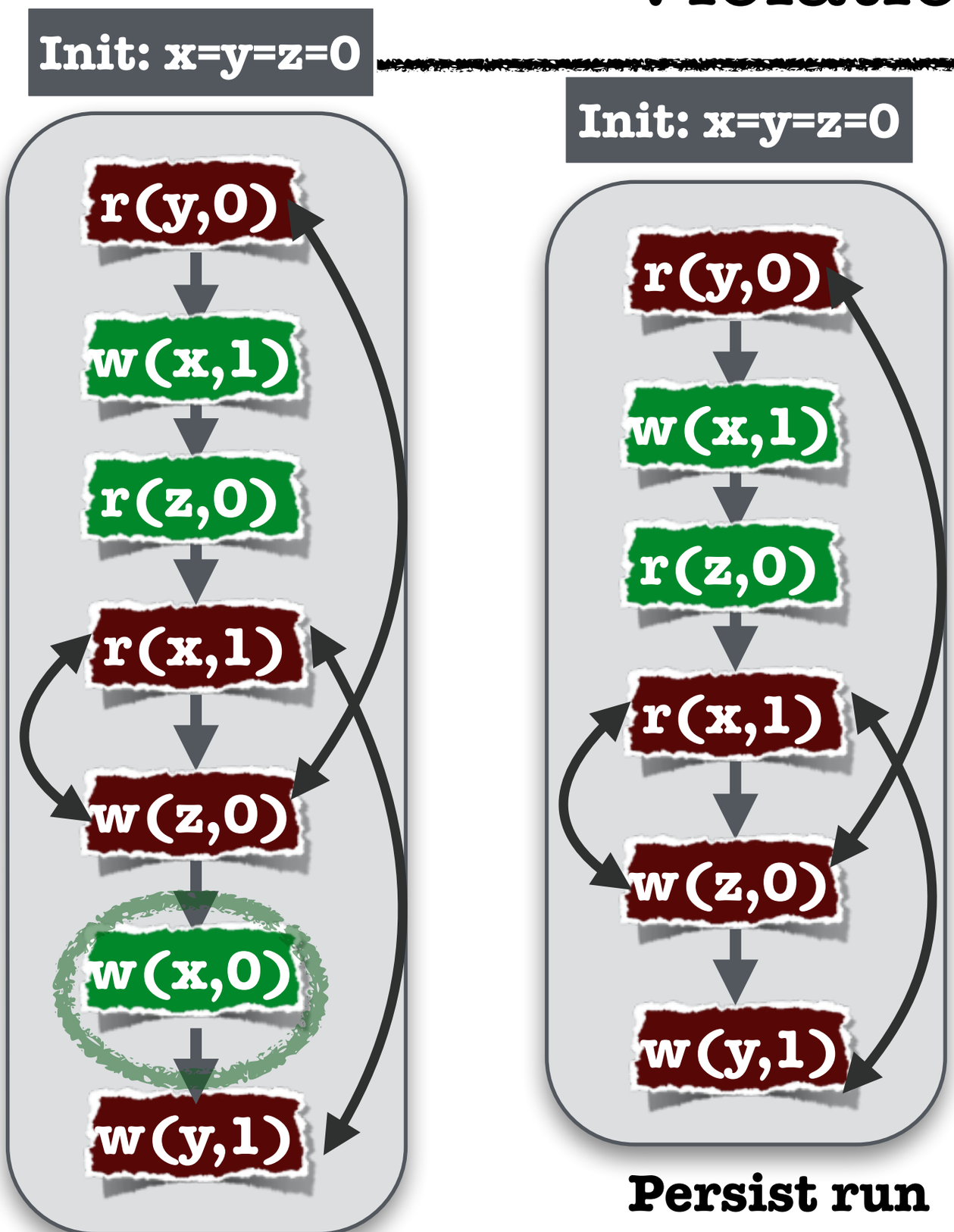


Proof sketch:

- **Remove all the actions of other processes occurring after the last read**

Singly TSO Minimal violation 1

Violation Shape



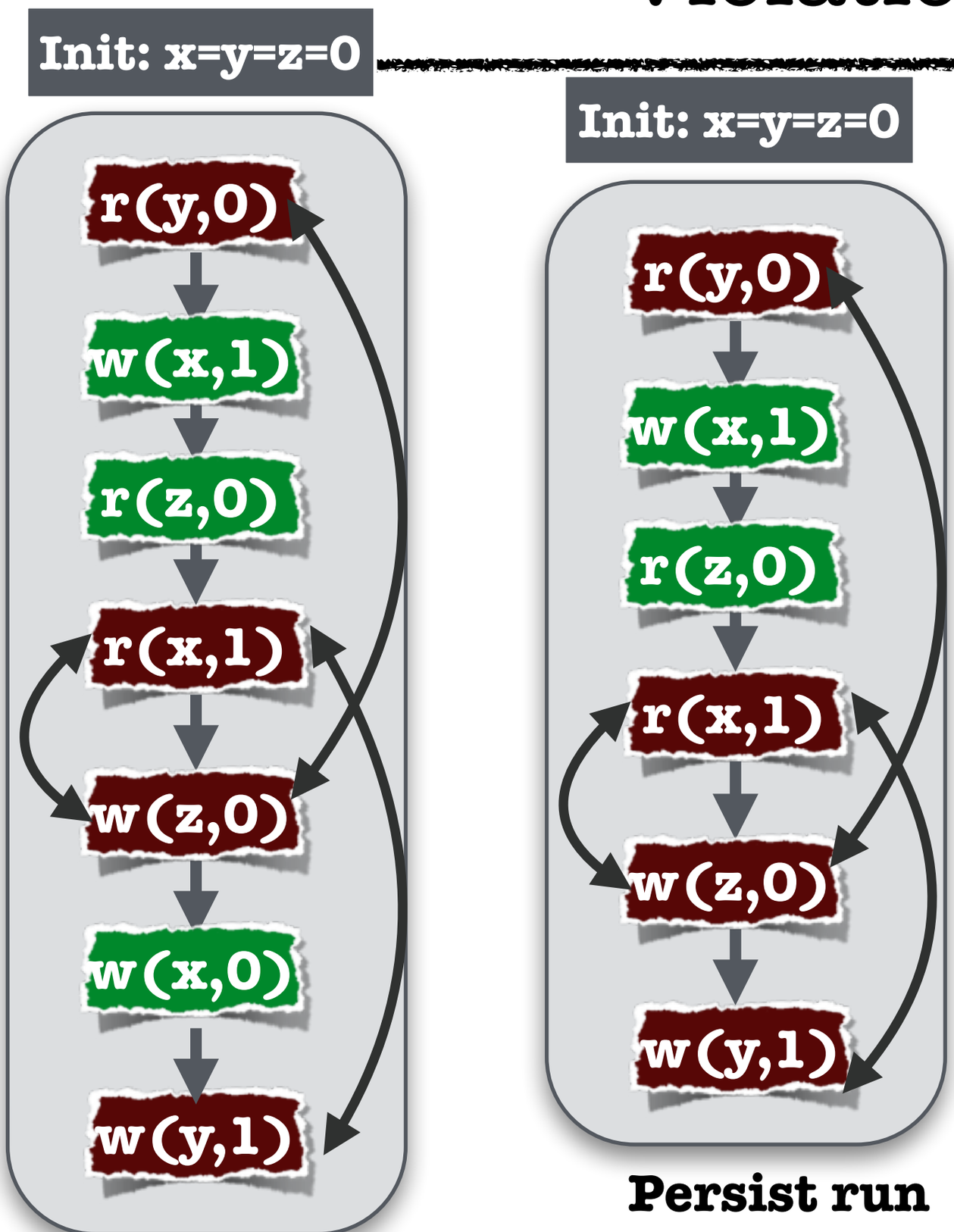
Persist run

Singly TSO Minimal violation 1

Proof sketch:

- **Remove all the actions of other processes occurring after the last read**

Violation Shape



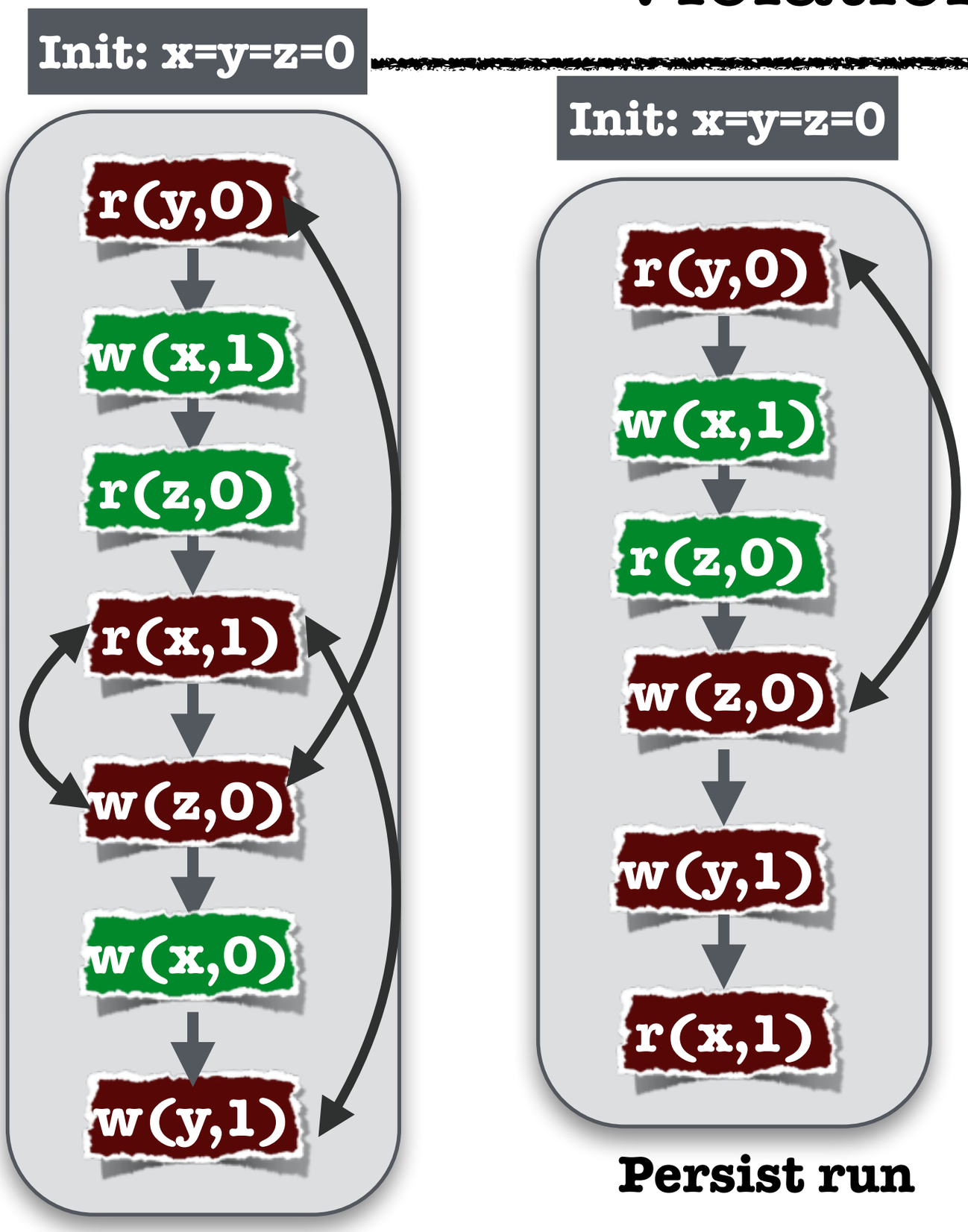
Singly TSO Minimal violation 1

Persist run

Proof sketch:

- **Remove all the actions of other processes occurring after the last read**
- **Construct the equivalent SC-run**

Violation Shape



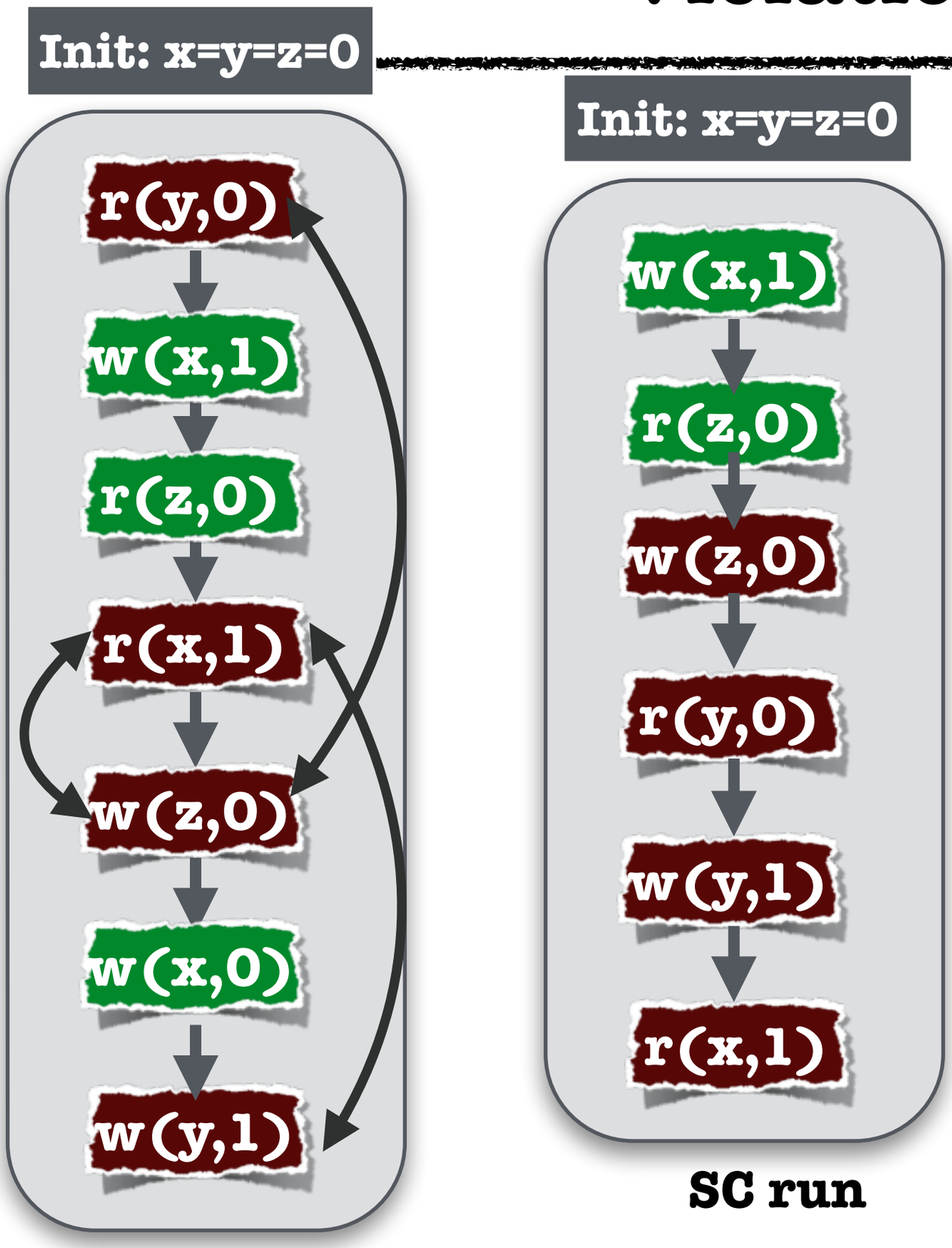
Singly TSO Minimal violation 1

Persist run

Proof sketch:

- **Remove all the actions of other processes occurring after the last read**
- **Construct the equivalent SC-run**

Violation Shape



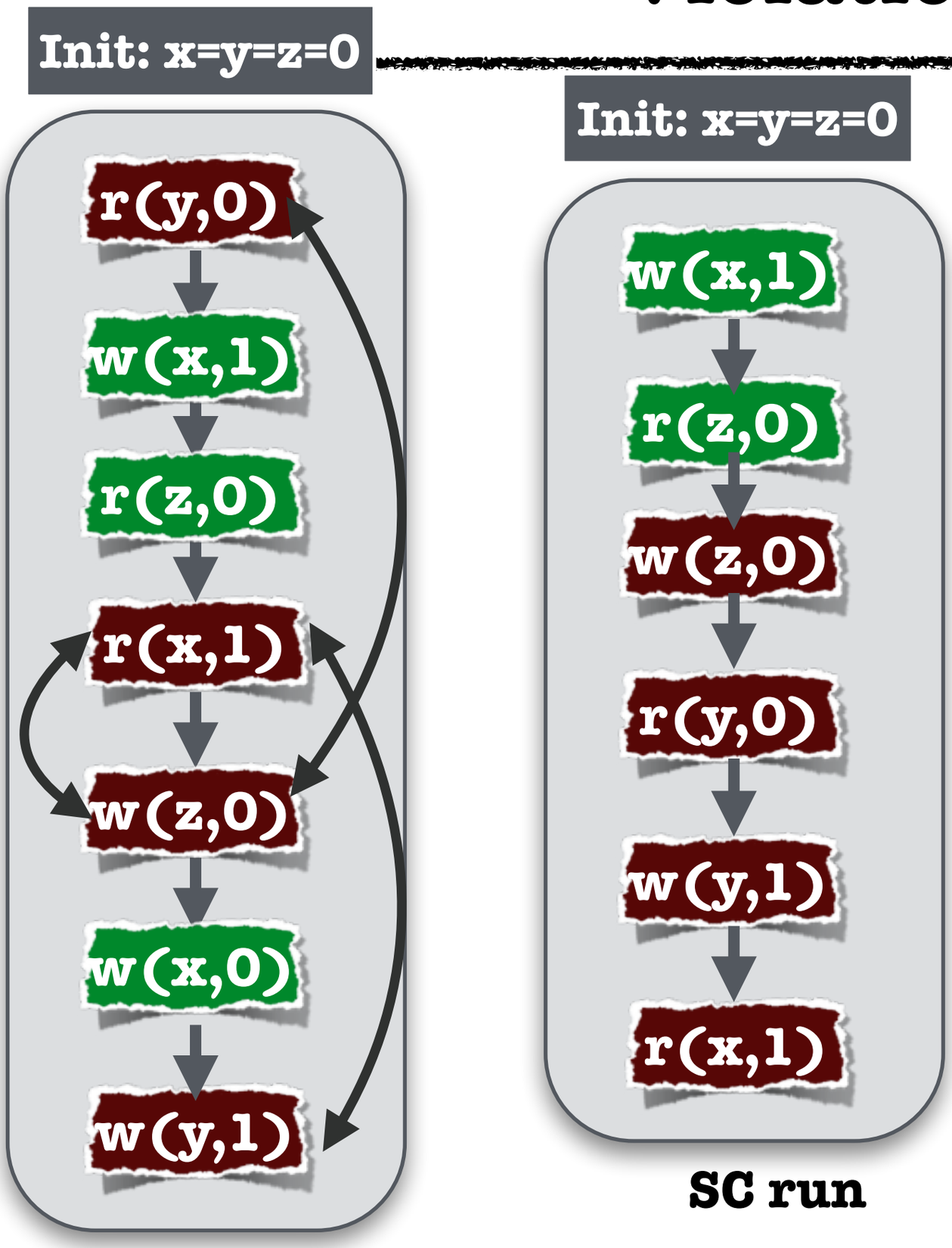
SC run

Singly TSO Minimal violation 1

Proof sketch:

- **Remove all the actions of other processes occurring after the last read**
- **Construct the equivalent SC-run**

Violation Shape



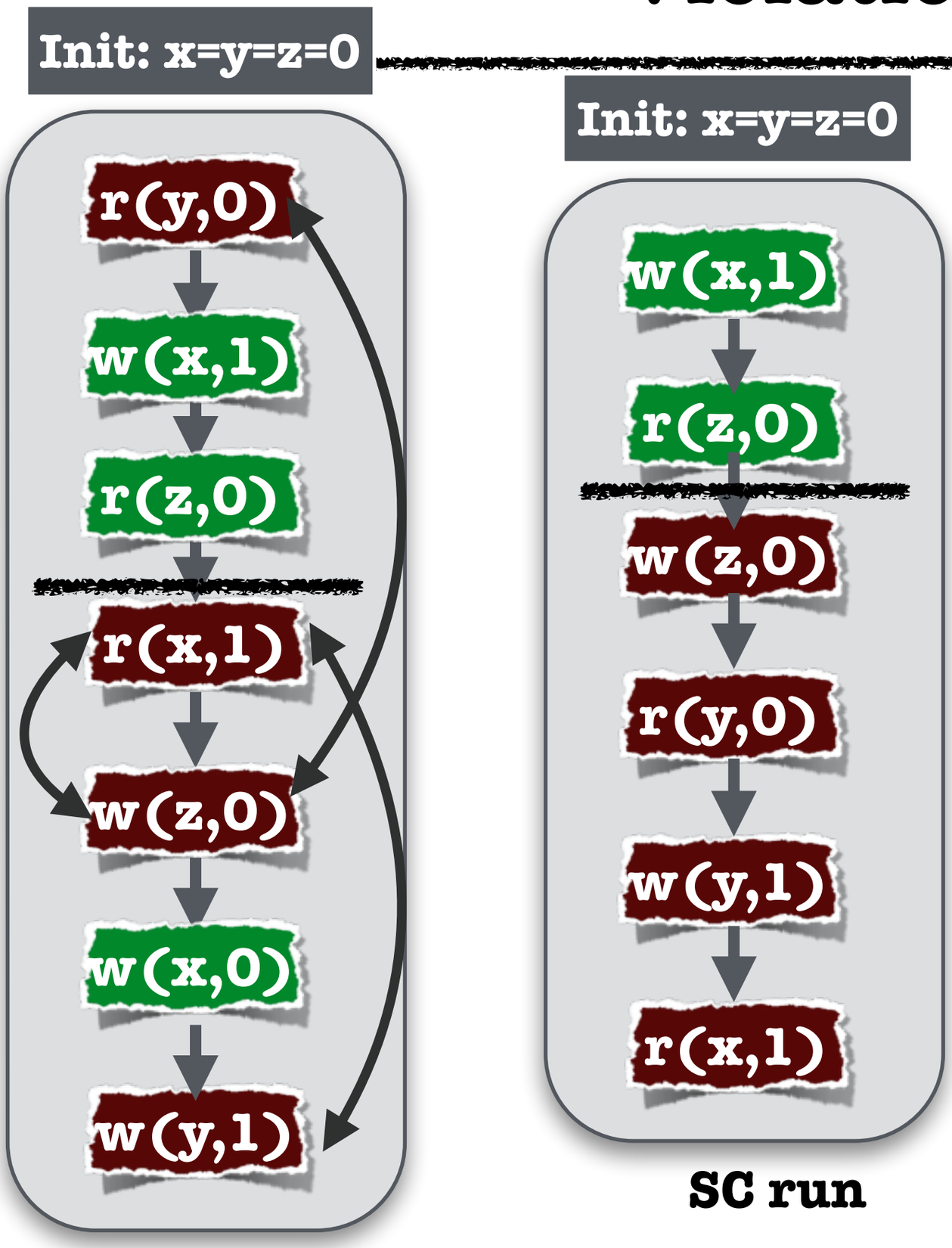
SC run

Singly TSO Minimal violation 1

Proof sketch:

- **Remove all the actions of other processes occurring after the last read**
- **Construct the equivalent SC-run**
- **Find bar points**

Violation Shape



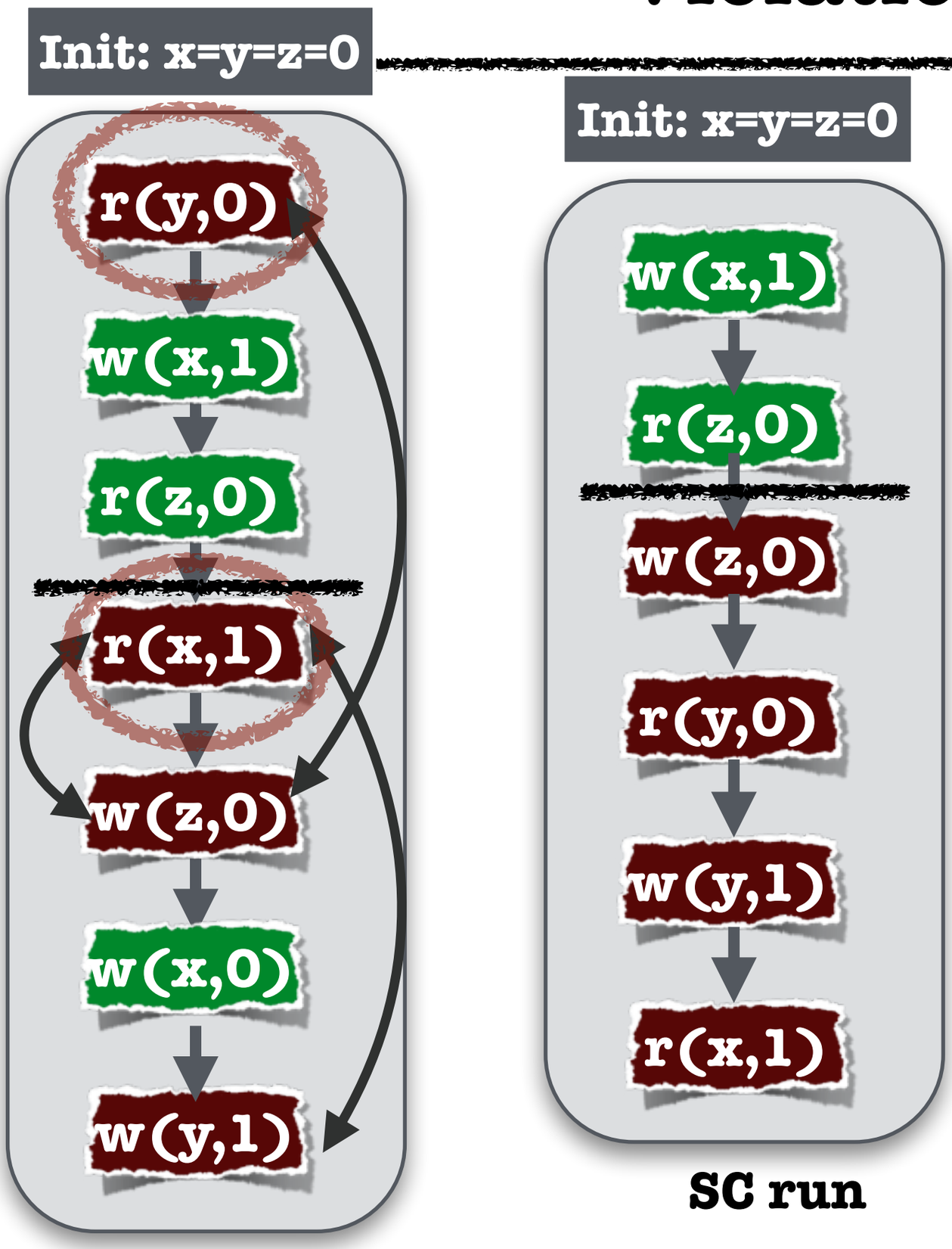
SC run

Singly TSO Minimal violation 1

Proof sketch:

- **Remove all the actions of other processes occurring after the last read**
- **Construct the equivalent SC-run**
- **Find bar points**

Violation Shape



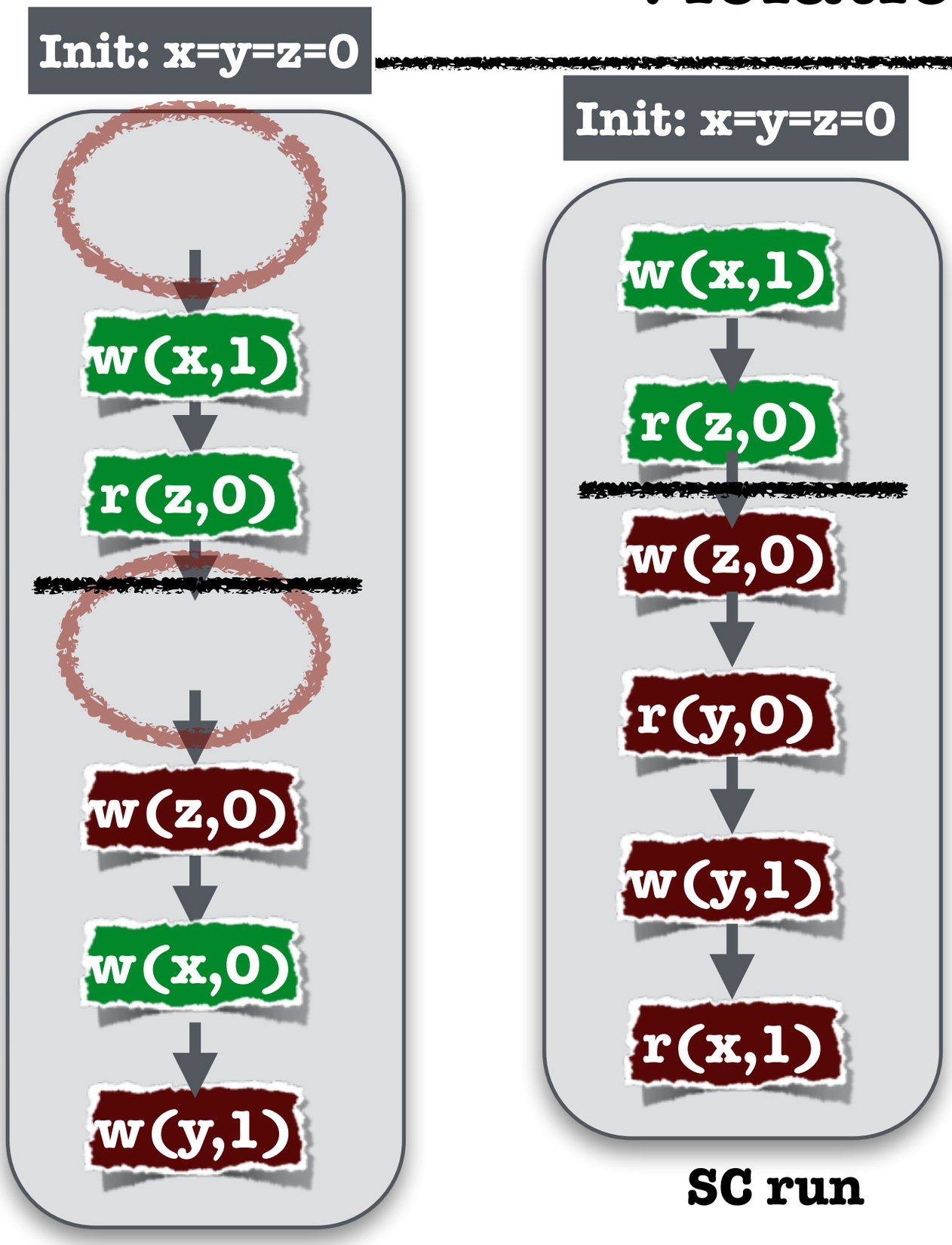
SC run

Singly TSO Minimal violation 1

Proof sketch:

- **Remove all the actions of other processes occurring after the last read**
- **Construct the equivalent SC-run**
- **Find bar points**
- **Delete all reads of the minimal**

Violation Shape

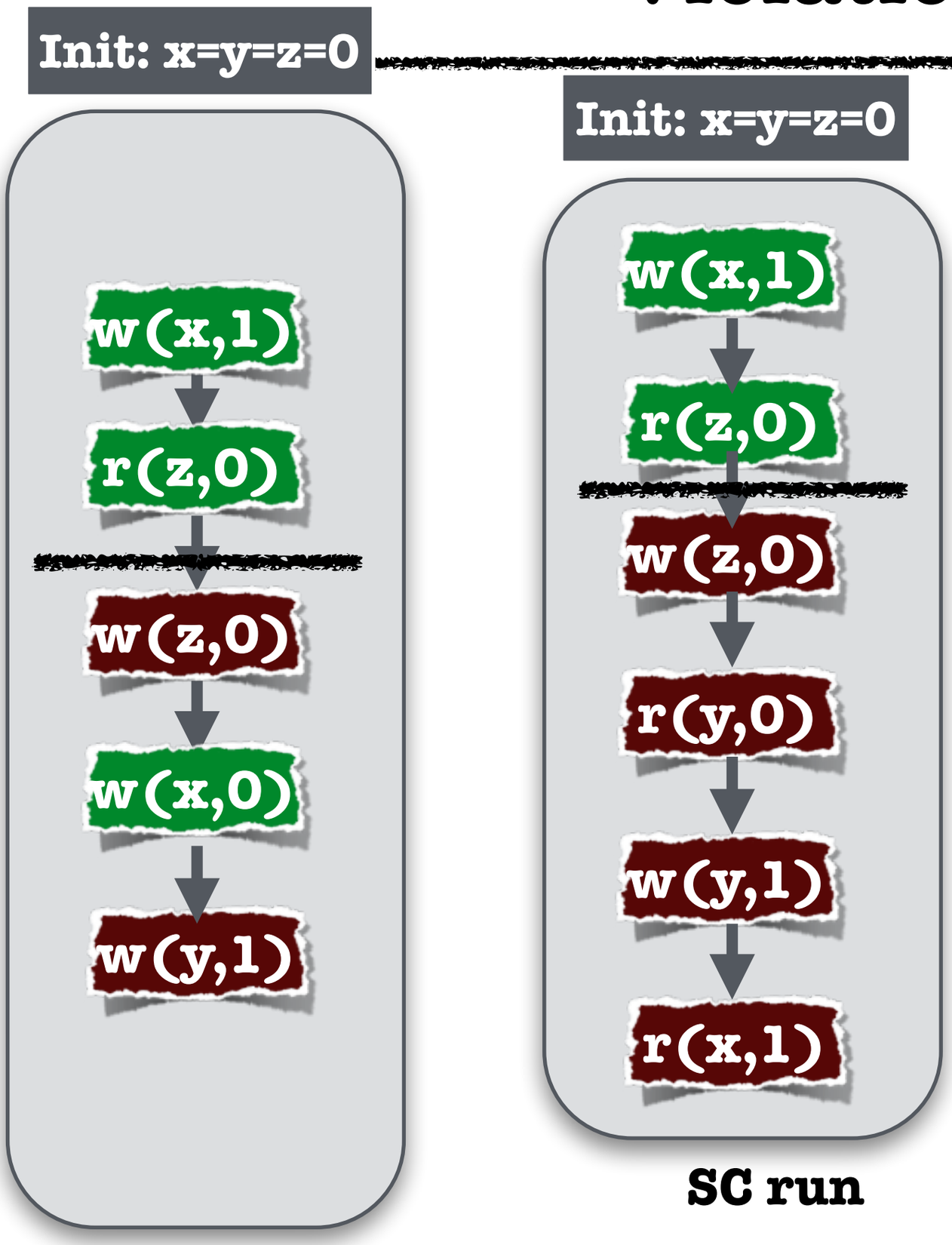


Singly TSO Minimal violation 1

Proof sketch:

- **Remove all the actions of other processes occurring after the last read**
- **Construct the equivalent SC-run**
- **Find bar points**
- **Delete all reads of the minimal**

Violation Shape

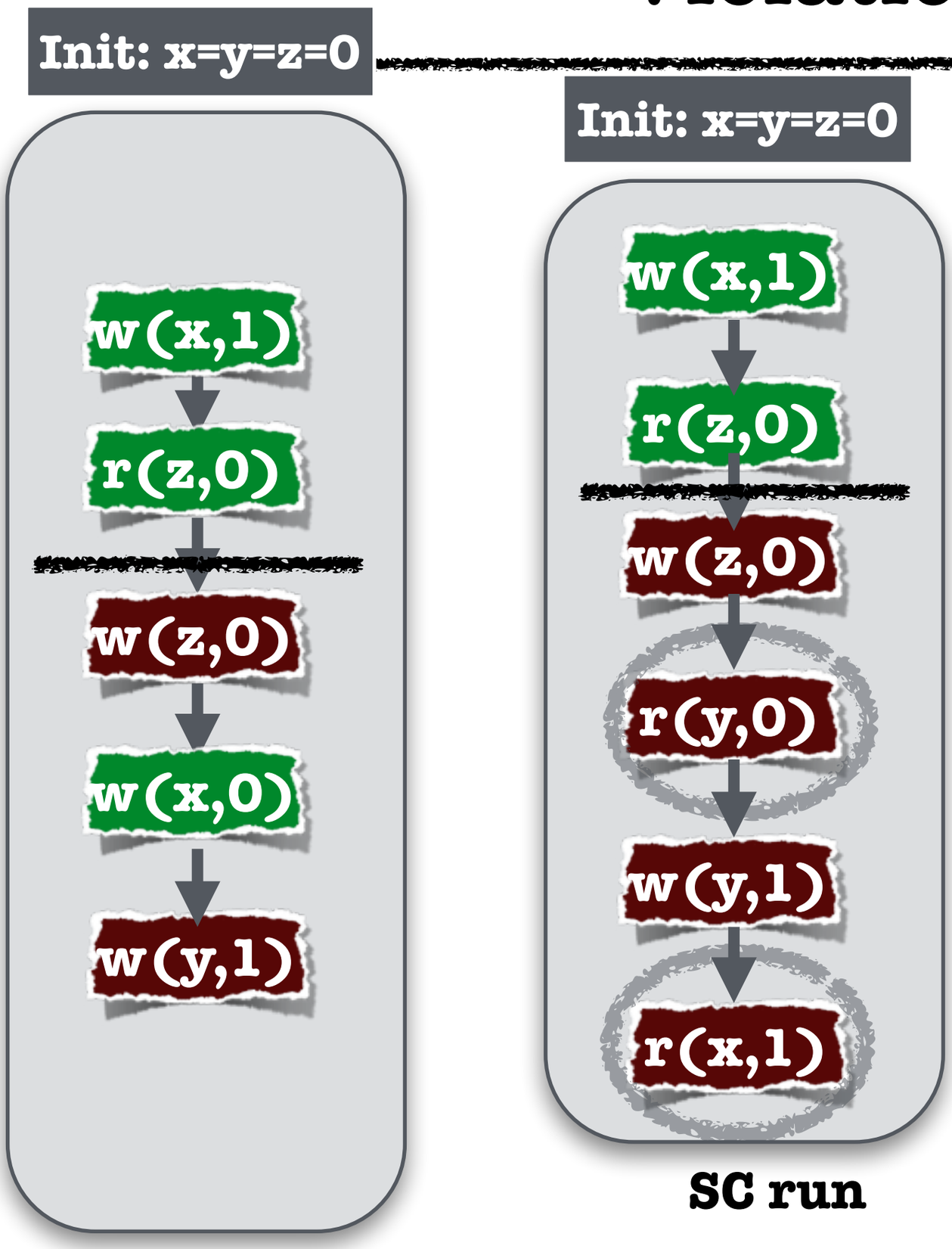


Singly TSO Minimal violation 1

Proof sketch:

- **Remove all the actions of other processes occurring after the last read**
- **Construct the equivalent SC-run**
- **Find bar points**
- **Delete all reads of the minimal**
- **Add the reads of SC run before bar point to the minimal with same order**

Violation Shape

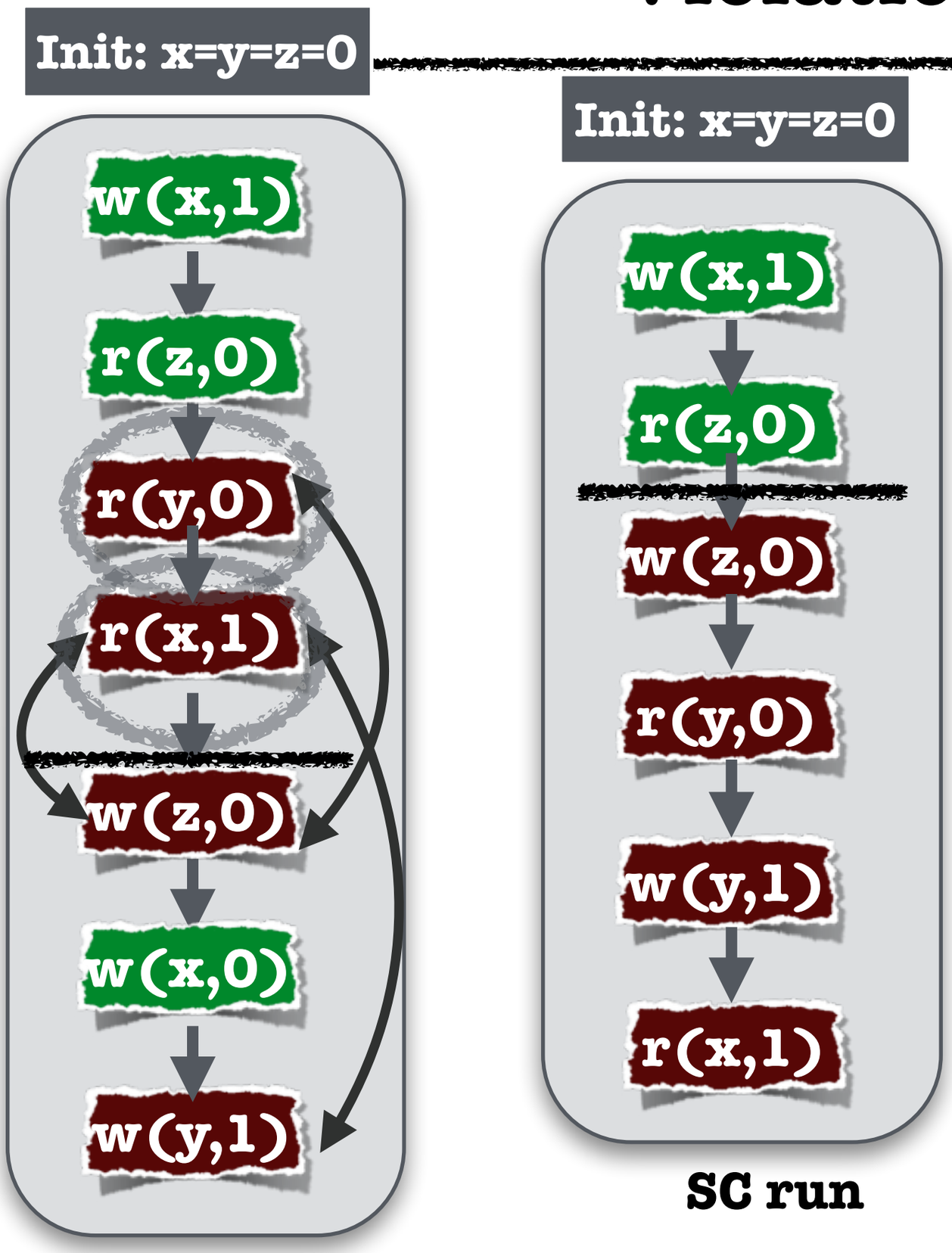


Singly TSO Minimal violation 1

Proof sketch:

- **Remove all the actions of other processes occurring after the last read**
- **Construct the equivalent SC-run**
- **Find bar points**
- **Delete all reads of the minimal**
- **Add the reads of SC run before bar point to the minimal with same order**
- **Add the reads of SC run after bar point to the minimal just before bar point**

Violation Shape

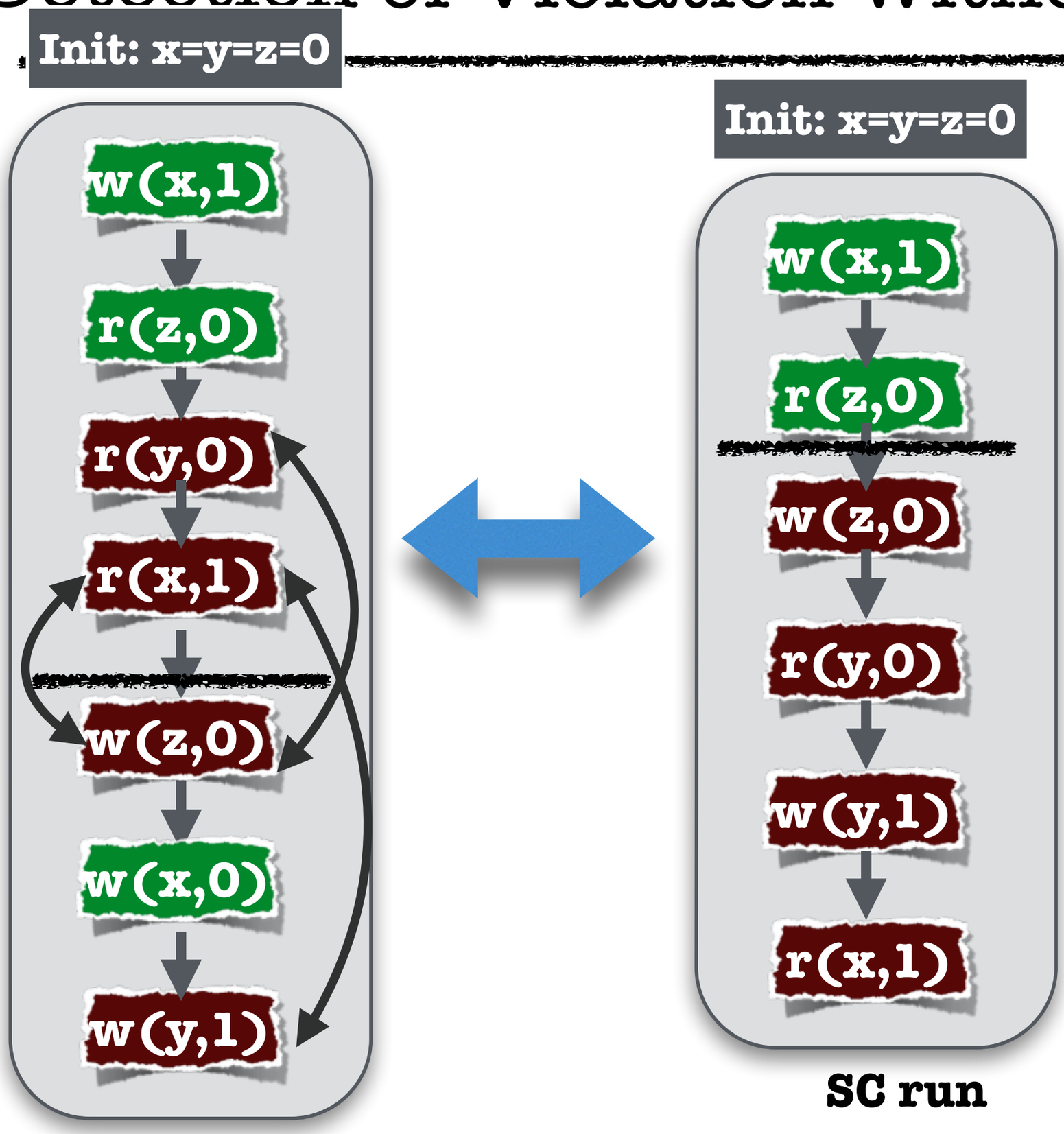


Proof sketch:

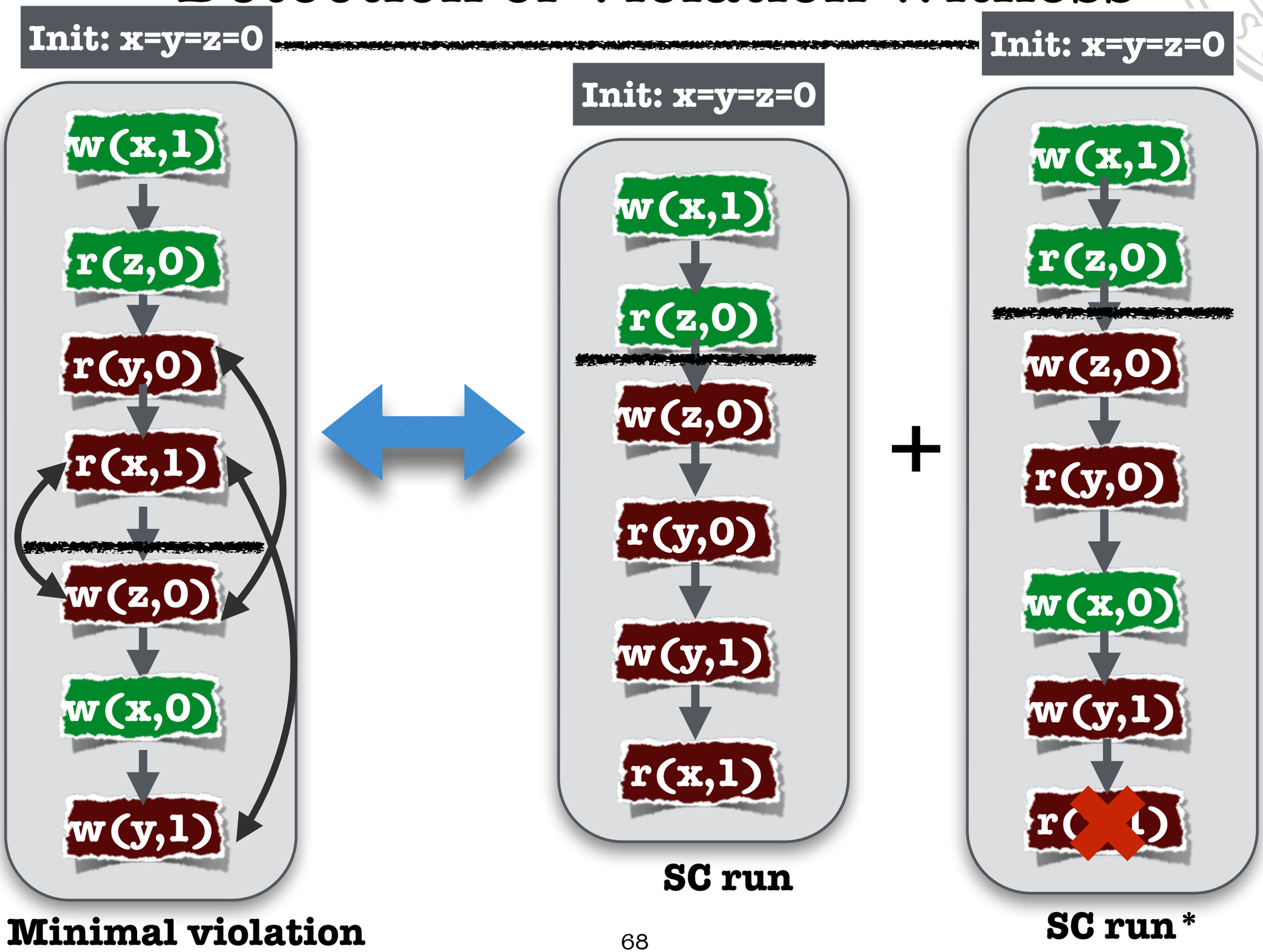
- **Remove all the actions of other processes occurring after the last read**
- **Construct the equivalent SC-run**
- **Find bar points**
- **Delete all reads of the minimal**
- **Add the reads of SC run before bar point to the minimal with same order**
- **Add the reads of SC run after bar point to the minimal just before bar point**

Singly TSO Minimal violation 2

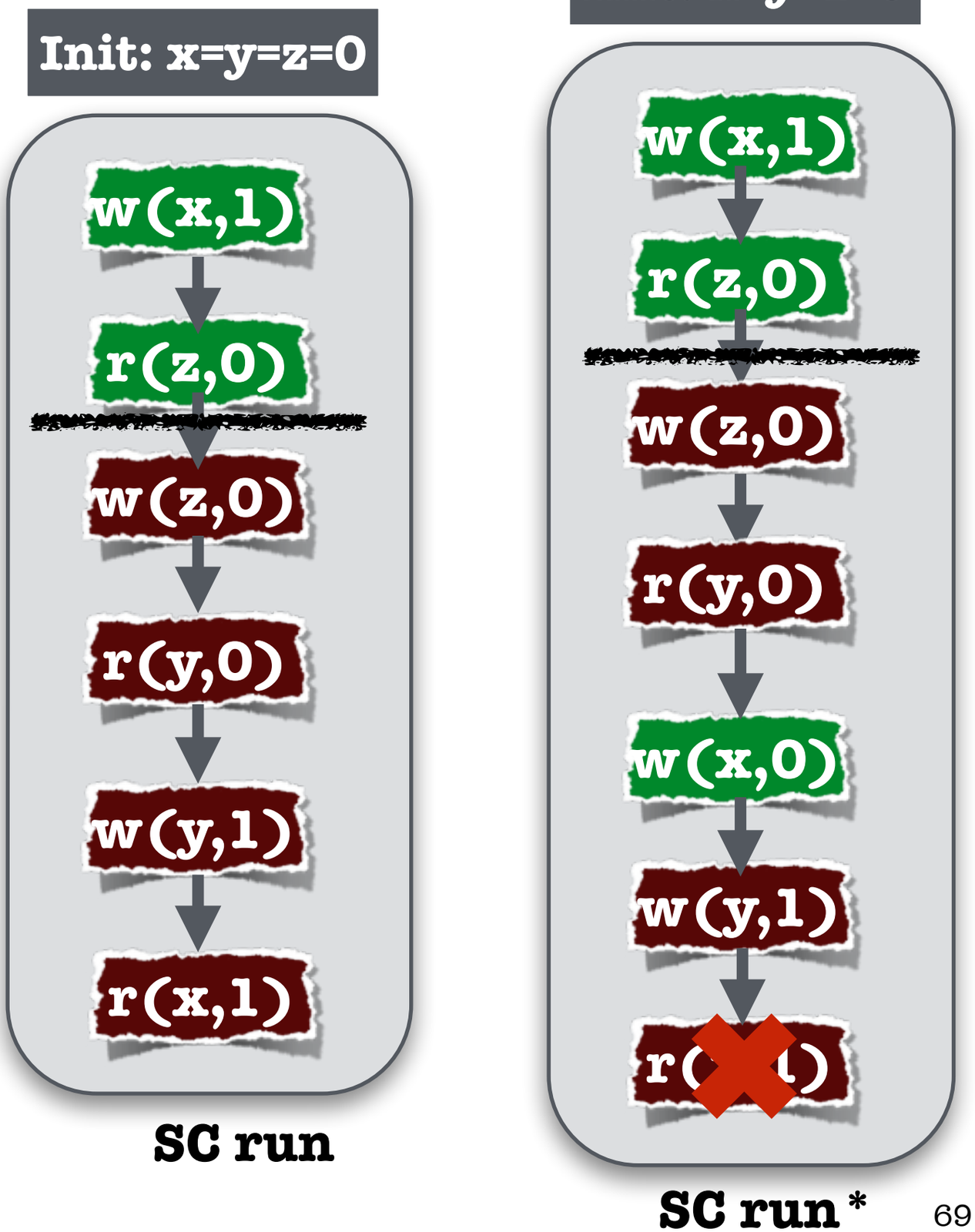
Detection of Violation Witness



Detection of Violation Witness

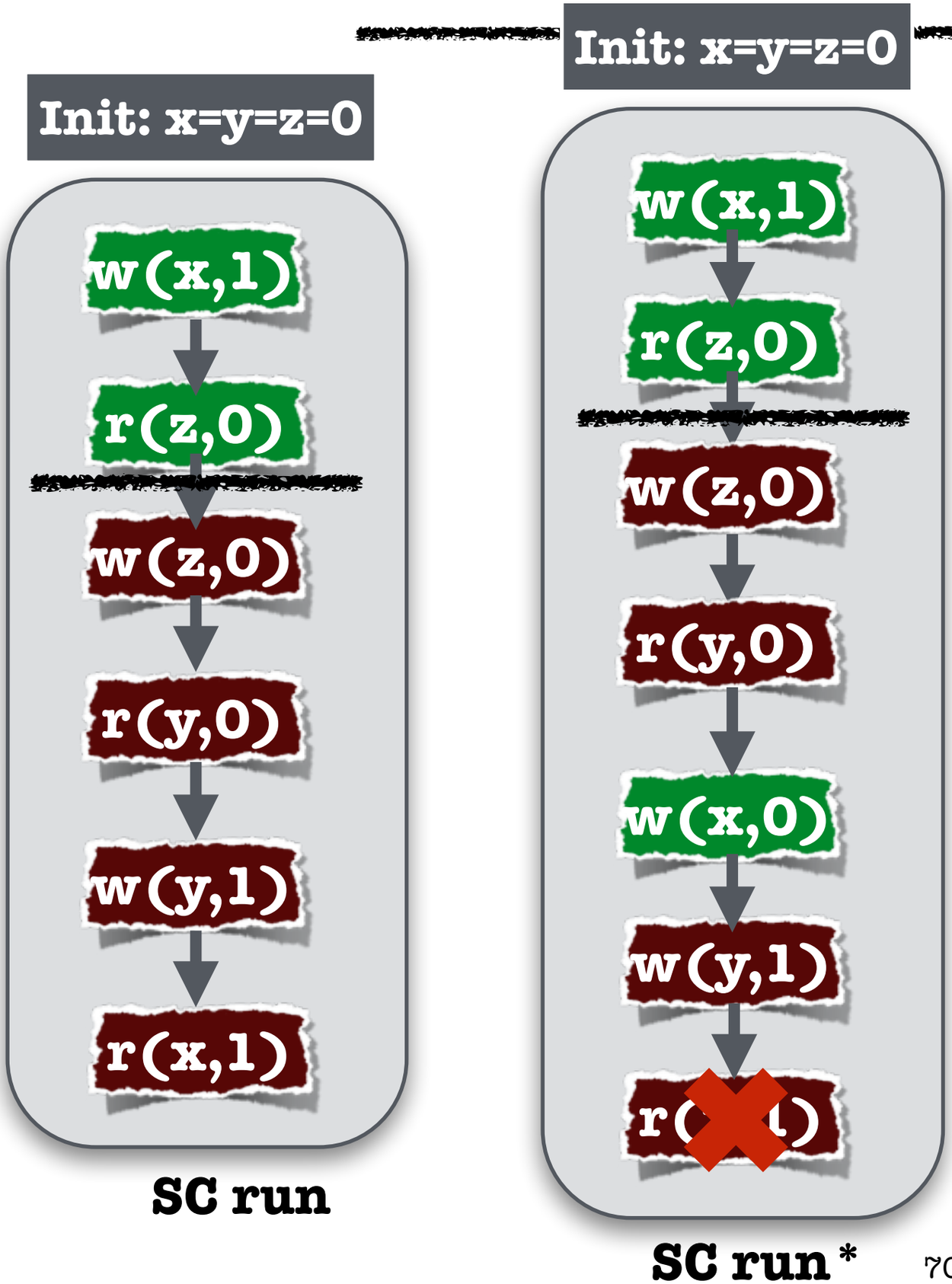


Code-to-Code Translation



Instrumentation:

Code-to-Code Translation

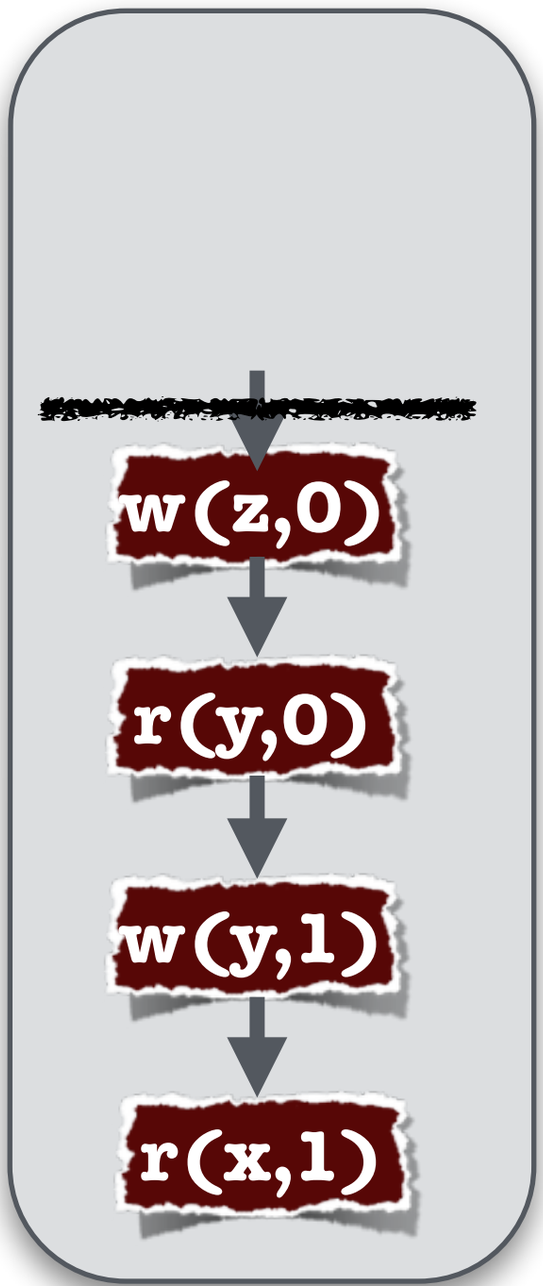


Instrumentation:

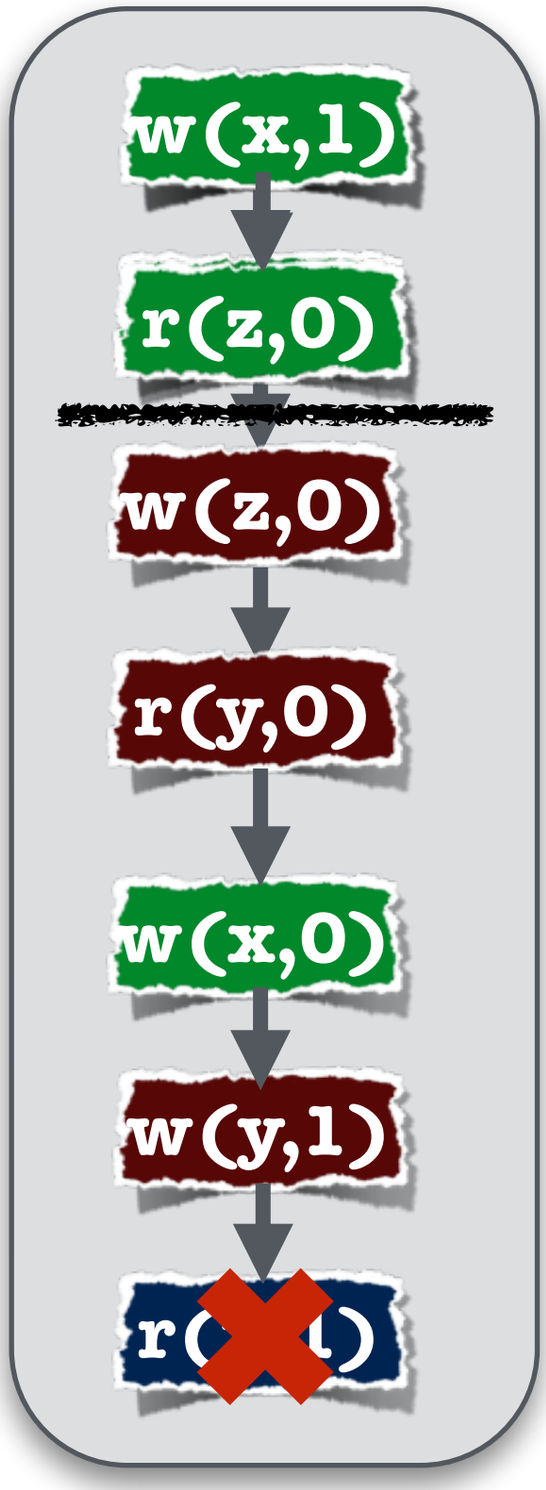
- **Merge the SC Prefix before reorderings**

Code-to-Code Translation

Init: x=y=z=0



SC run



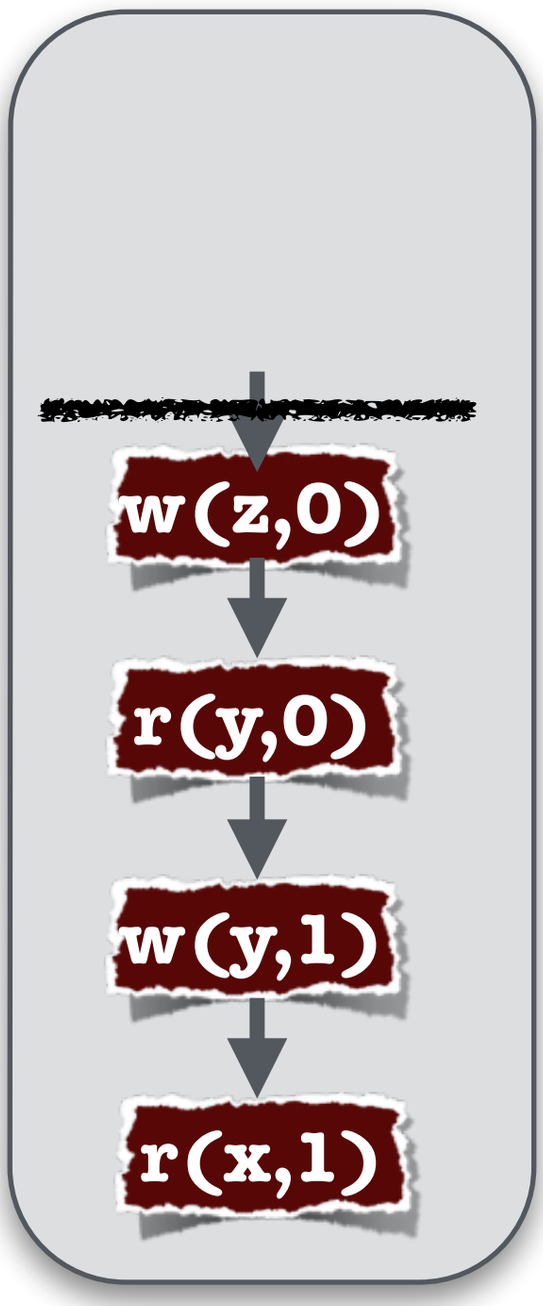
SC run *

Instrumentation:

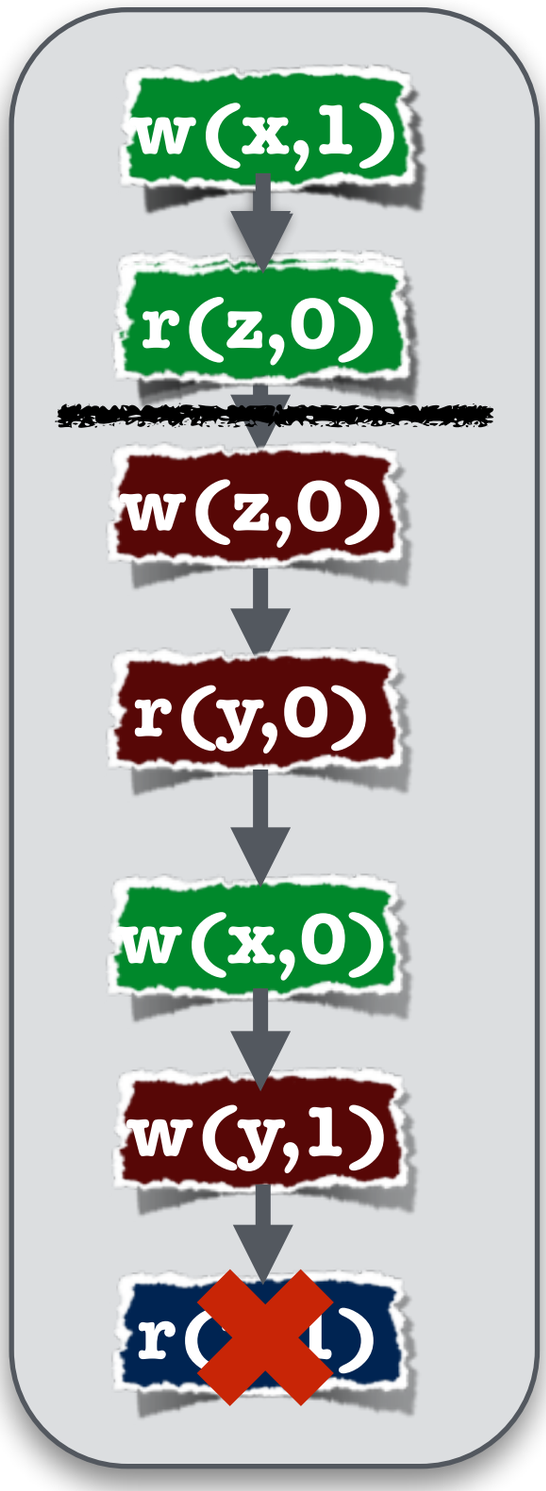
- **Merge the SC Prefix before reorderings**

Code-to-Code Translation

Init: x=y=z=0



SC run

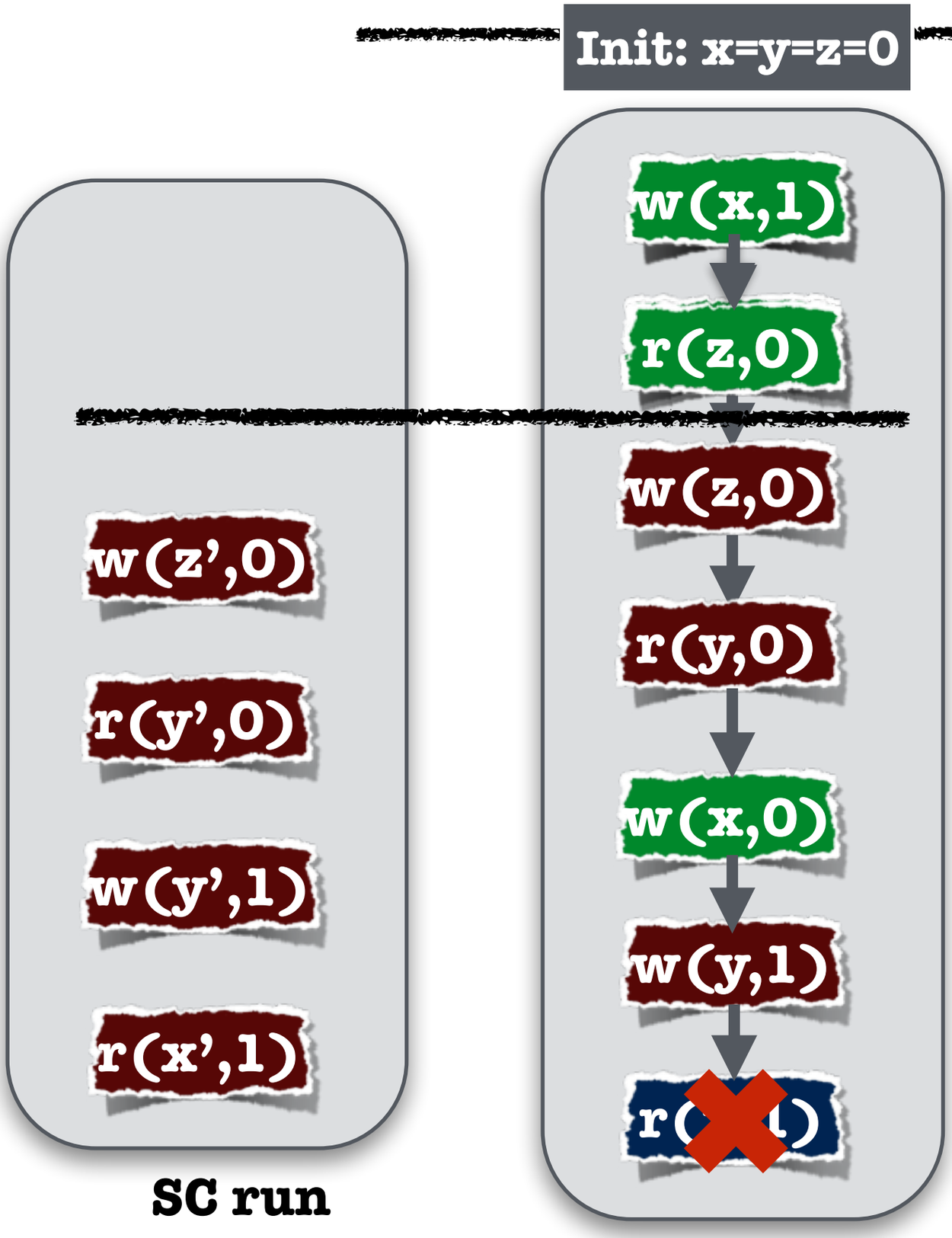


SC run *

Instrumentation:

- **Merge the SC Prefix before reorderings**
- **Make a snapshot of the memory**
- **Use this snapshot to validate the remaining actions of run**

Code-to-Code Translation

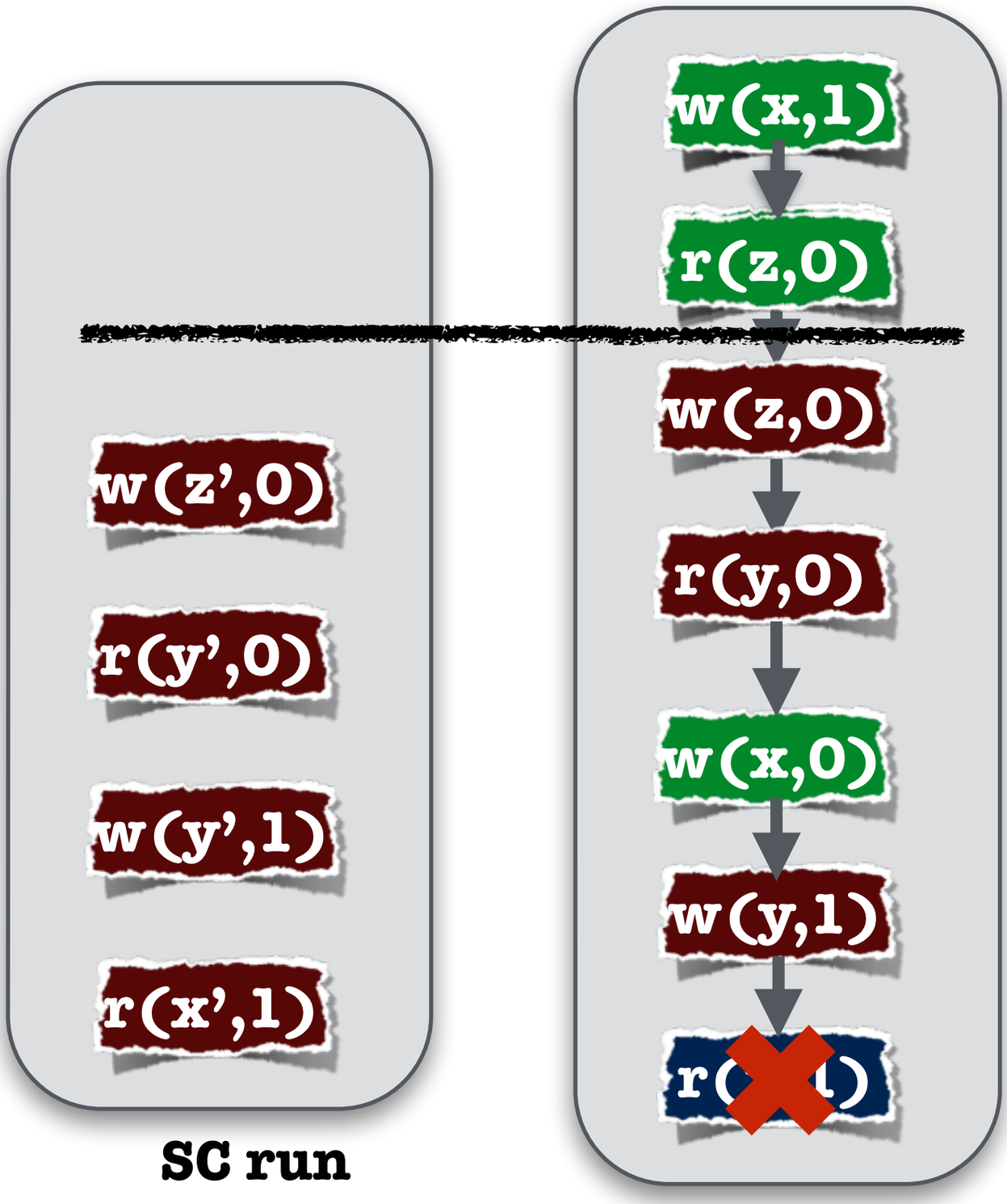


Instrumentation:

- **Merge the SC Prefix before reorderings**
- **Make a snapshot of the memory**
- **Use this snapshot to validate the remaining actions of run**

Code-to-Code Translation

Init: x=y=z=0

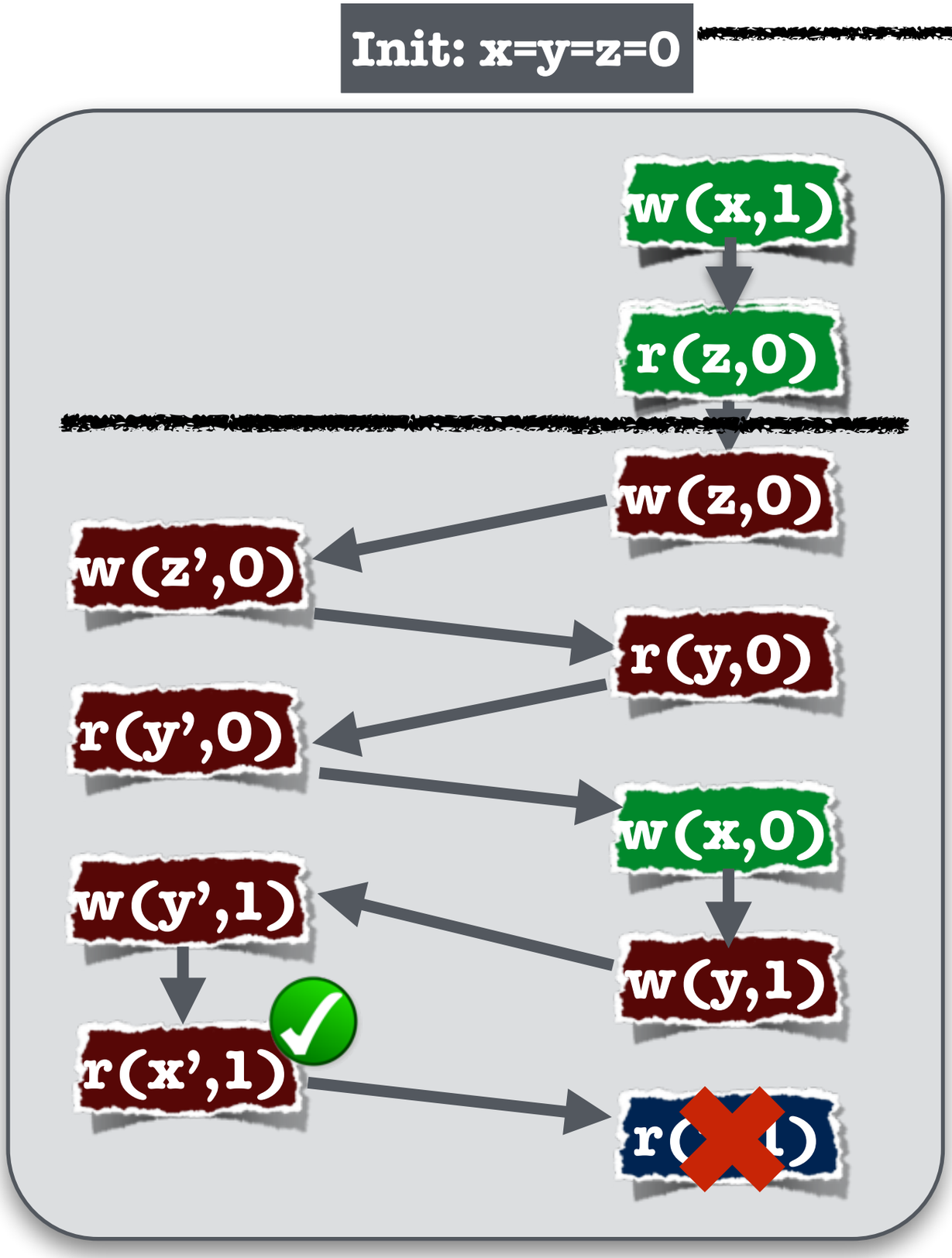


SC run **SC run ***

Instrumentation:

- **Merge the SC Prefix before reorderings**
- **Make a snapshot of the memory**
- **Use this snapshot to validate the remaining actions of run**
- **Synchronise the actions of the pivot process**

Code-to-Code Translation



SC run

Instrumentation:

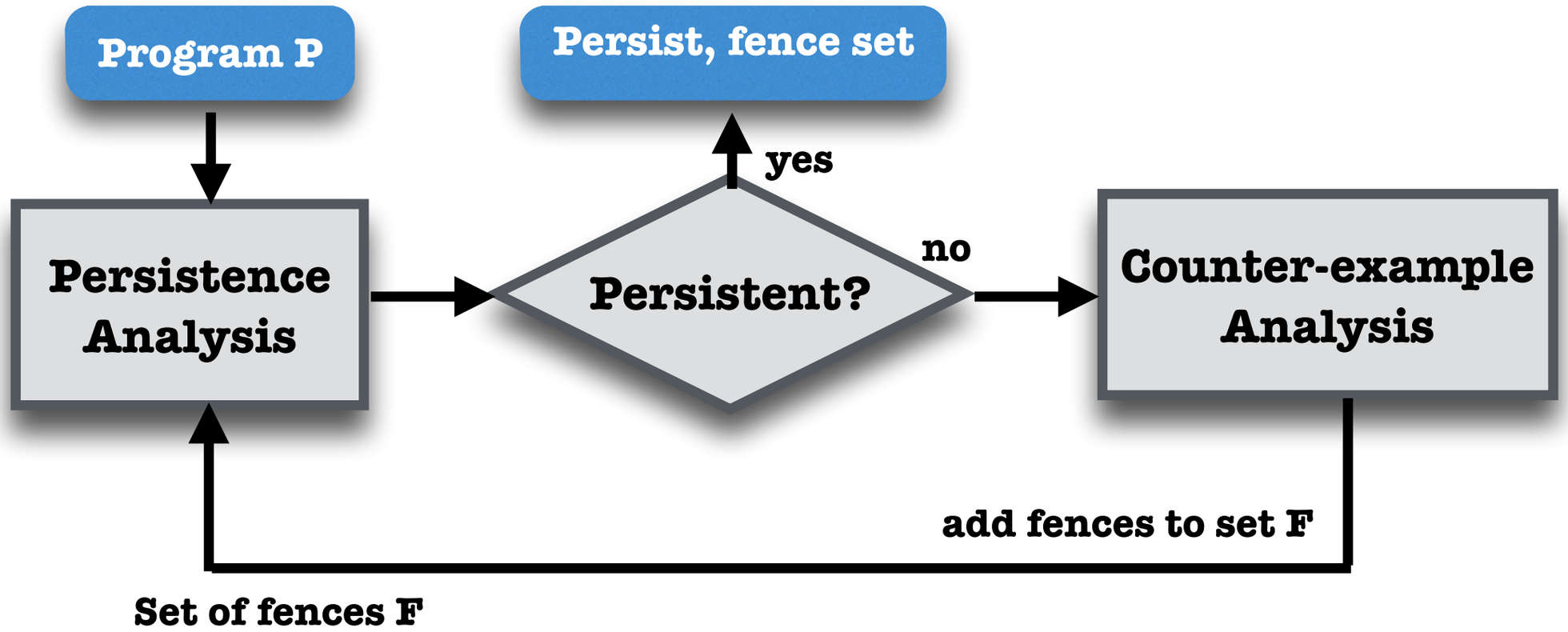
- Merge the SC Prefix before reorderings
- Make a snapshot of the memory
- Use this snapshot to validate the remaining actions of run
- Synchronise the actions of the pivot process

Checking Persistence: Complexity

✦ Checking Trace-Persistence under TSO is Pspace-complete

- finite data domain
- finite number of variables
- finite number of processes
- adding only 2 variables

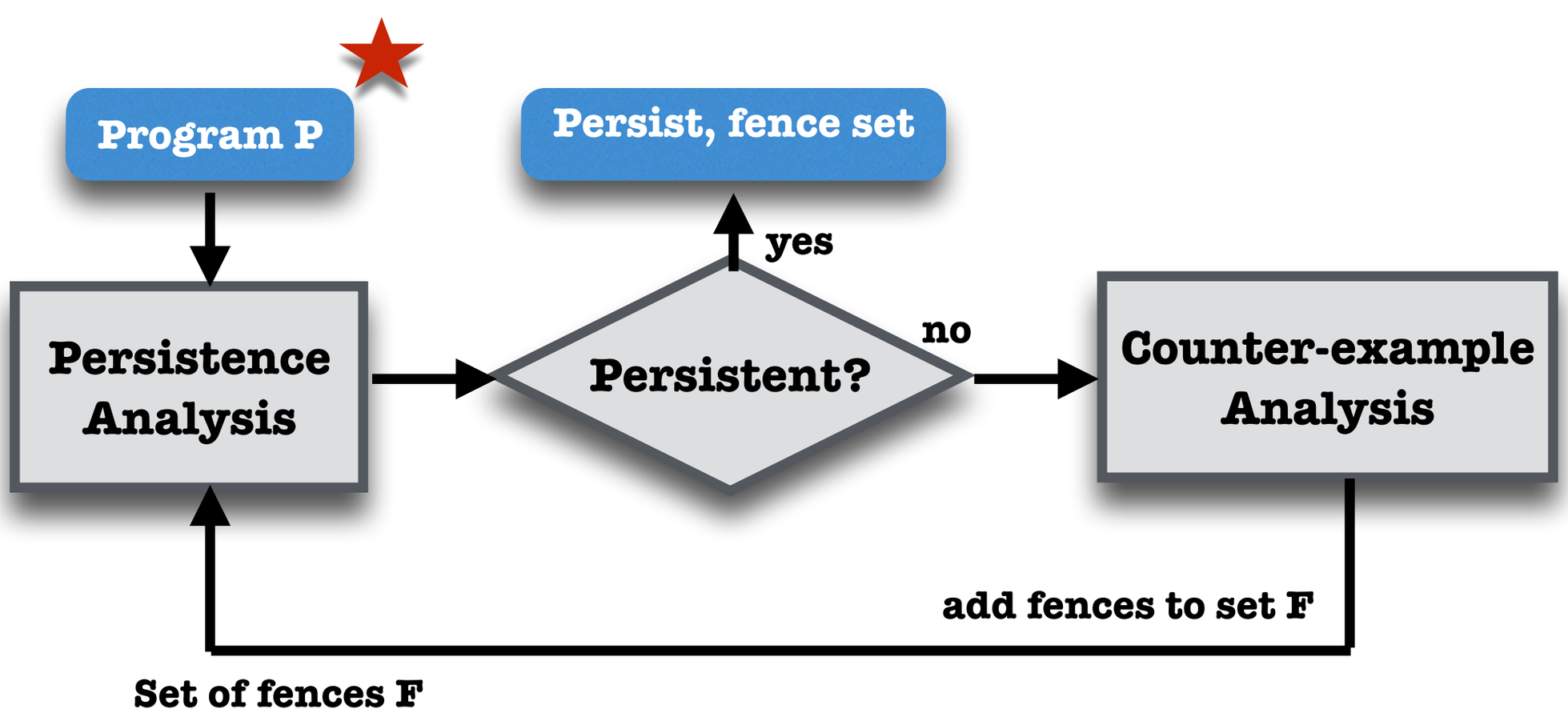
Counter-example Procedure



Counter-example Procedure

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

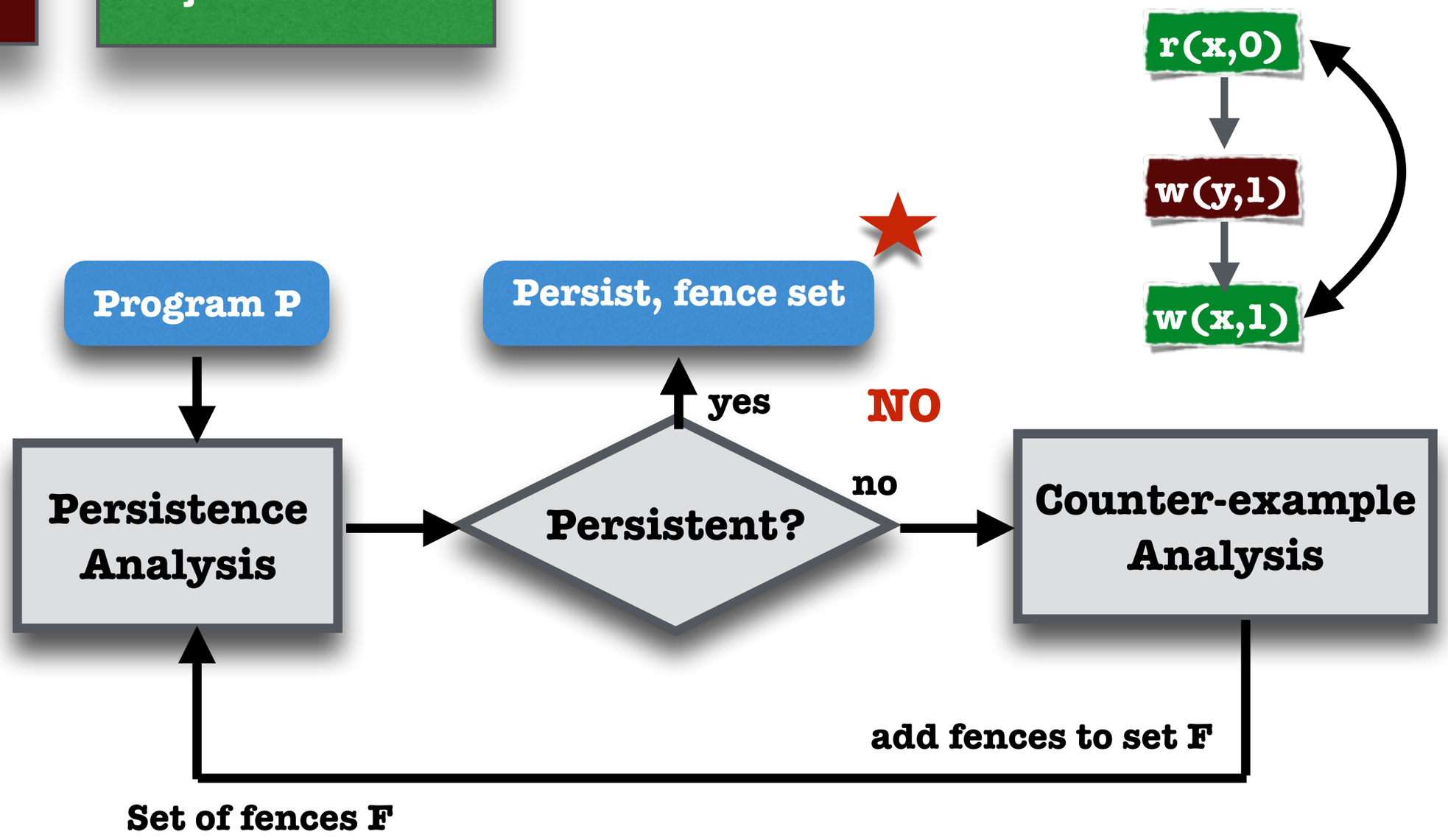
```
1. y=1;
2. rx=x;
3. if (rx==0) {
4. //cs2;
5. }
```



Counter-example Procedure

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

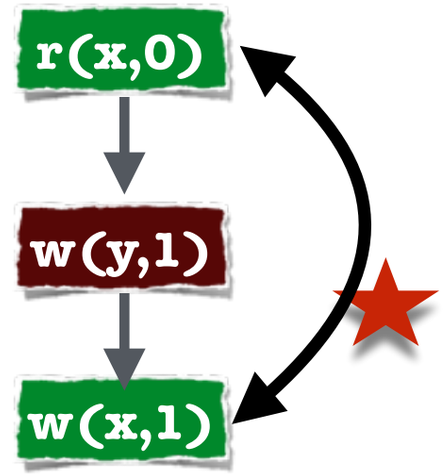
```
1. y=1;
2. rx=x;
3. if (rx==0) {
4. //cs2;
5. }
```

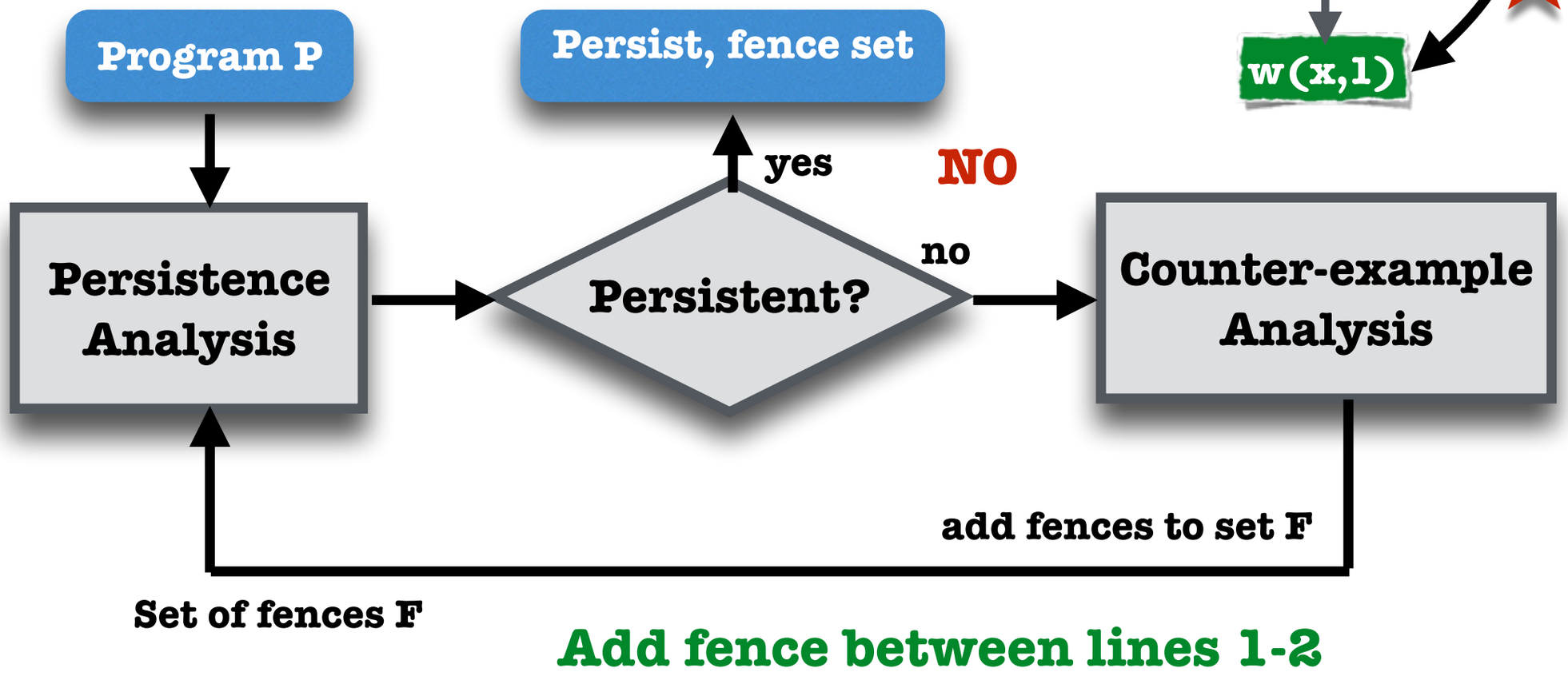


Counter-example Procedure

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

```
1. y=1;
2. rx=x;
3. if (rx==0) {
4. //cs2;
5. }
```



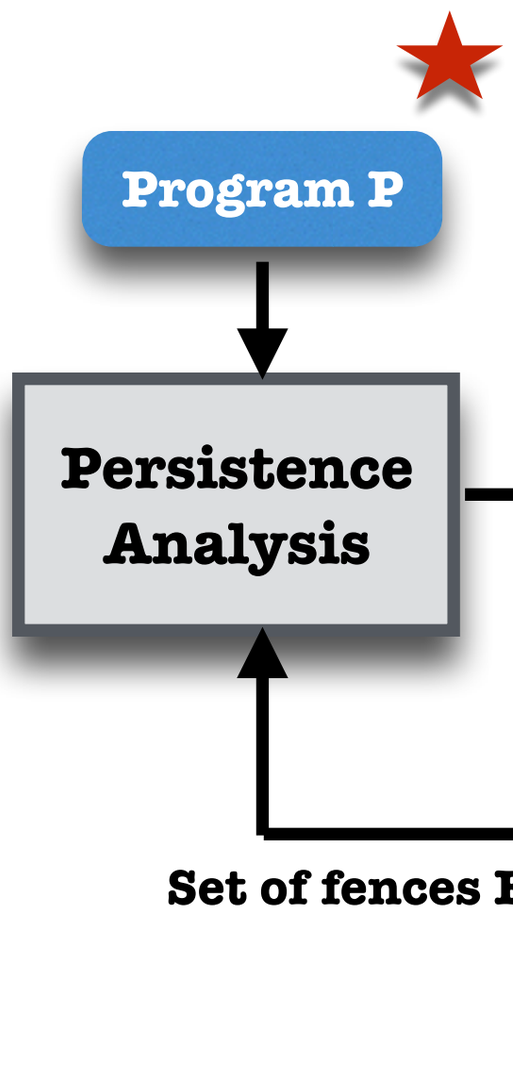


Add fence between lines 1-2

Counter-example Procedure

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

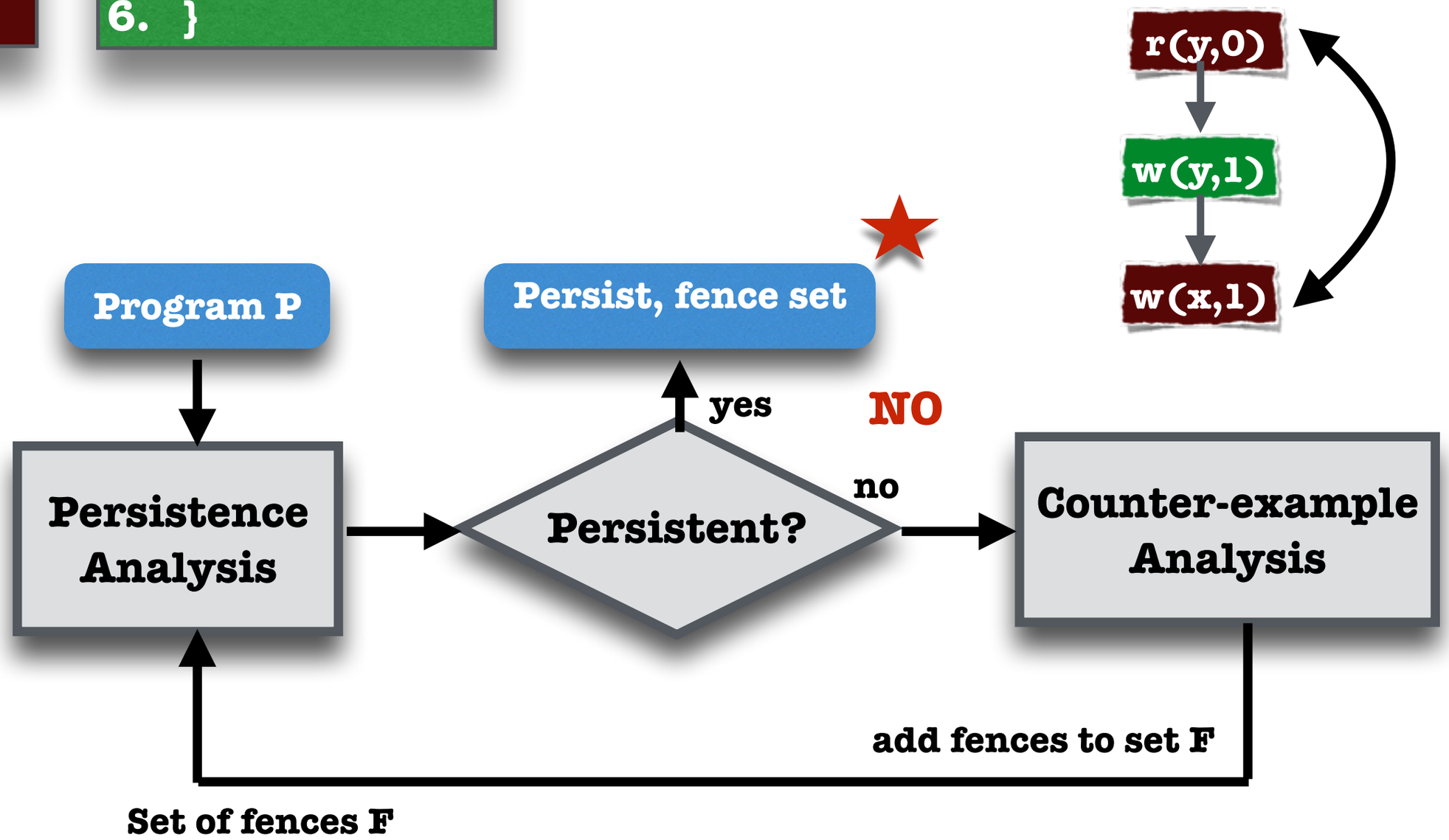
```
1. y=1;
2. fence
3. rx=x;
4. if (rx==0) {
5. //cs2;
6. }
```



Counter-example Procedure

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

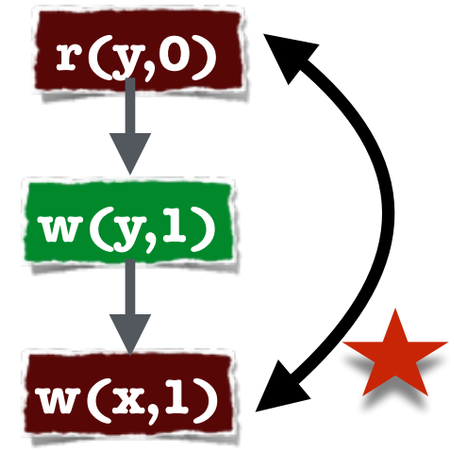
```
1. y=1;
2. fence
3. rx=x;
4. if (rx==0) {
5. //cs2;
6. }
```

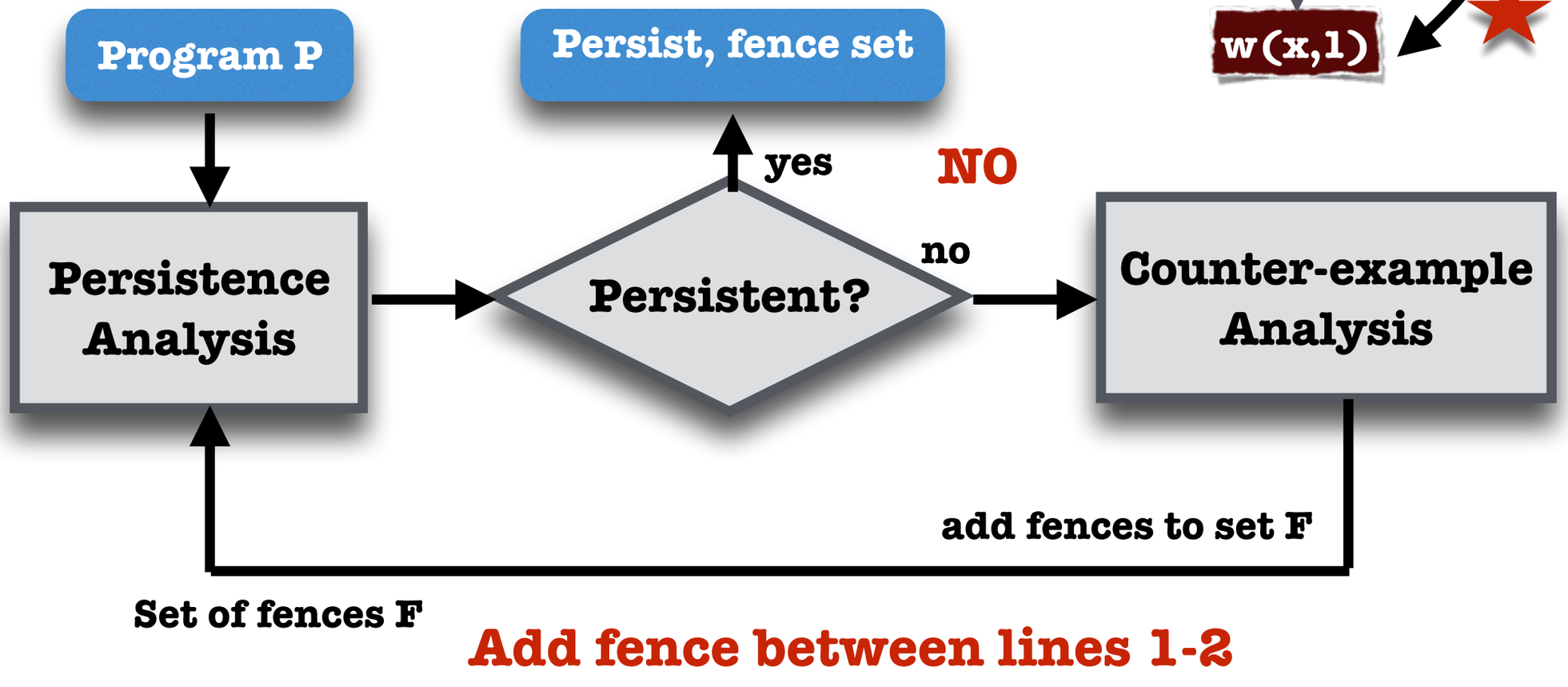


Counter-example Procedure

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

```
1. y=1;
2. fence
3. rx=x;
4. if (rx==0) {
5. //cs2;
6. }
```



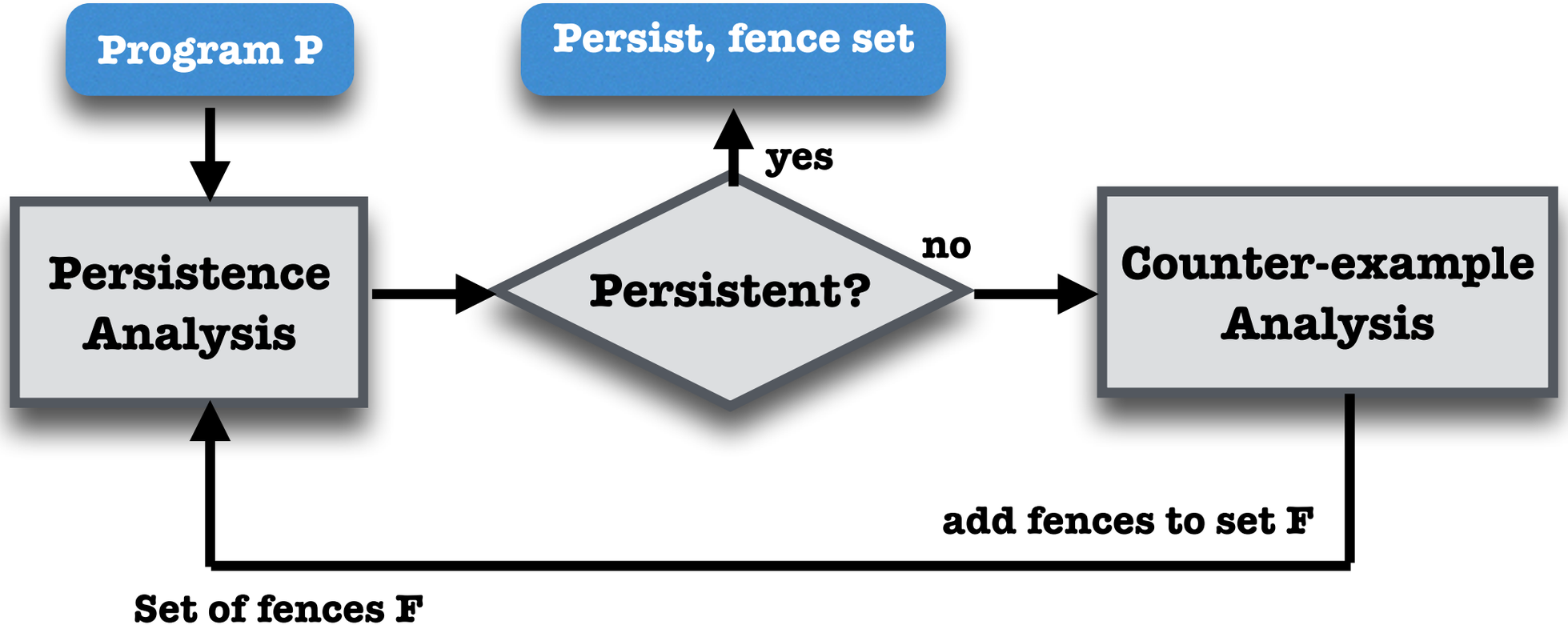


Add fence between lines 1-2

Counter-example Procedure

```
1. x=1;
2. fence
3. ry=y;
4. if (ry==0) {
5. //cs1;
6. }
```

```
1. y=1;
2. fence
3. rx=x;
4. if (rx==0) {
5. //cs2;
6. }
```

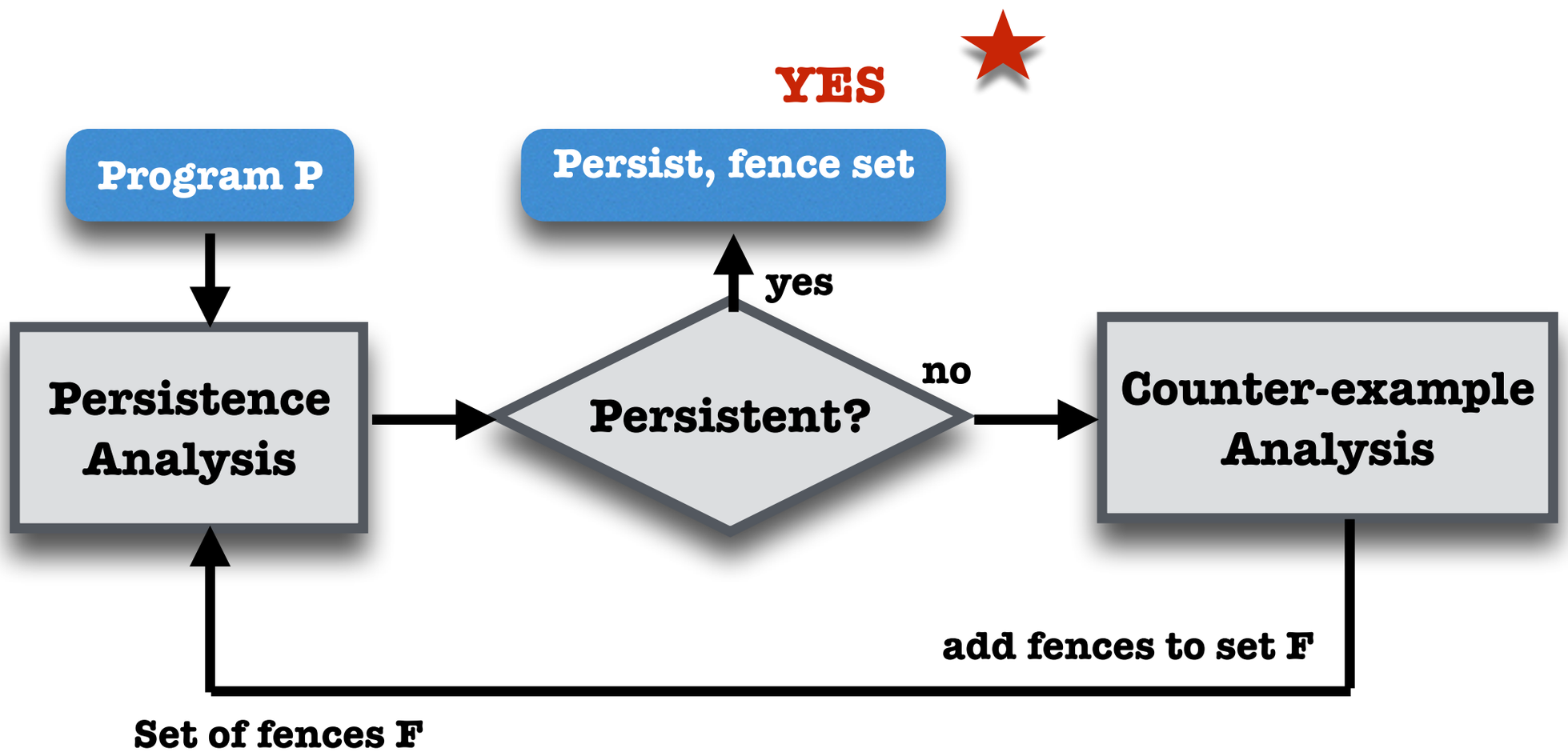


Counter-example Procedure

```
1. x=1;
2. fence
3. ry=y;
4. if (ry==0) {
5. //cs1;
6. }
```

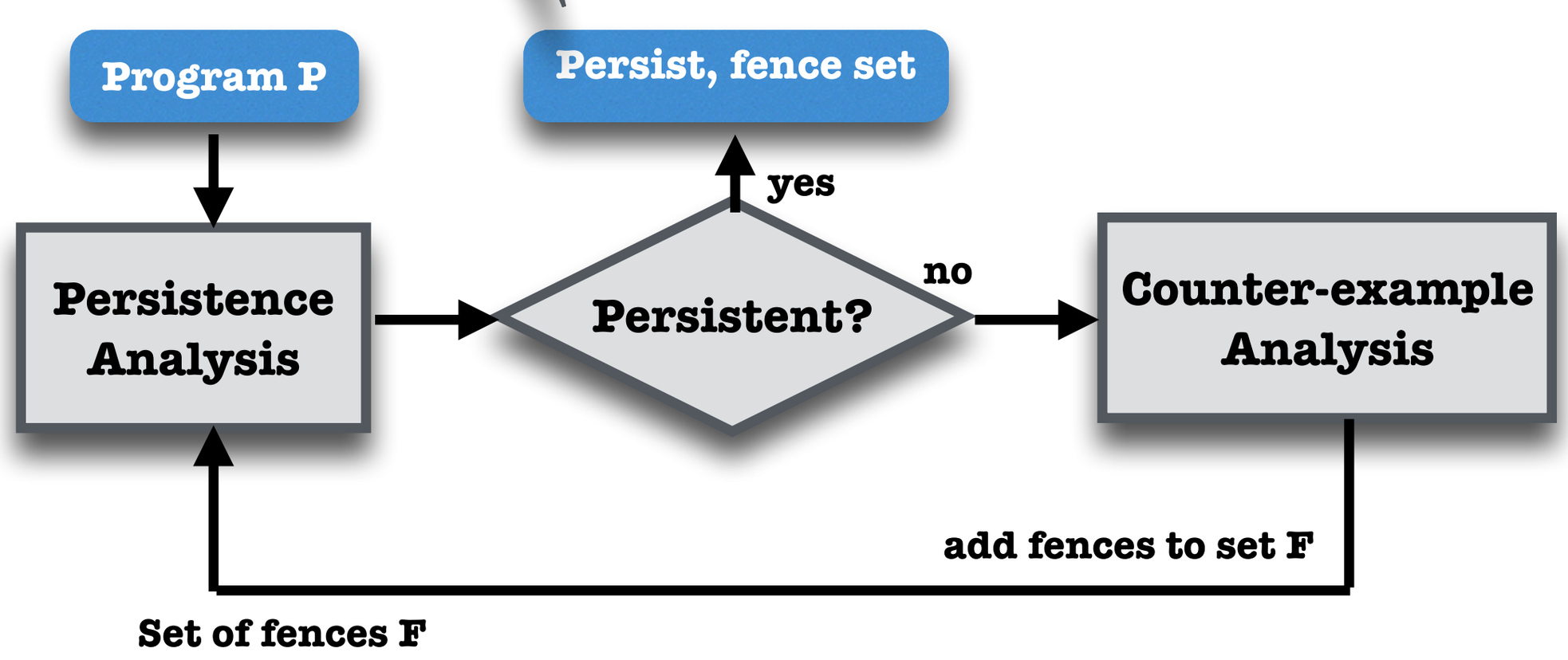
```
1. y=1;
2. fence
3. rx=x;
4. if (rx==0) {
5. //cs2;
6. }
```

YES



Counter-example Procedure

Optimality set: needs subset relations etc.



Observation Abstraction

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. t=0;
5. }
```

Pivot Process

```
1. y=1;
2. rx=x;
3. if (rx==0) {
4. t=1;
5. }
```

Concrete

Observation Abstraction

Observe memory changes from:

- its own instructions
- writes of other processes that it can read

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. t=0;
5. }
```

```
1. y=1;
2. rx=x;
3. if (rx==0) {
4. t=1;
5. }
```

Pivot Process

Concrete

Abstract

Observe more sequence of memory views

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. t=0;
5. }
```

```
1. y=1;
```

Observation Abstraction

Persistence of the program w.r.t the pivot

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. t=0;
5. }
```

```
1. y=1;
2. rx=x;
3. if (rx==0) {
4. t=1;
5. }
```

Pivot Process

Concrete

Abstract

Persistence of the program w.r.t the pivot

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. t=0;
5. }
```

```
1. y=1;
```

Observation Abstraction

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. t=0;
5. }
```

```
1. y=1;
2. rx=x;
3. if (rx==0) {
4. t=1;
5. }
```

Pivot Process

Concrete

Efficiency without losing precision

Abstract

```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. t=0;
5. }
```

```
1. y=1;
```

Implemented prototype

- Open-source: Persist*
- Properties:
  - Using Spin as model-checker backend
  - Working for finite-state programs

*: https://www.it.uu.se/katalog/tuang296/persistence

Result without Abstraction

Program	#Process	Persist		Trencher		Musketeer		Memorax		Remmex	
		#Fence	#Time	**#Fence**	#Time	**#Fence**	#Time	**#Fence**	#Time	**#Fence**	#Time
SimDekker	4	**4**	4	**4**	163	**8**	1	•	OM	•	OM
Dekker	4	**8**	14	•	TO	**16**	783	•	OM	•	OM
Peterson	4	**4**	223	•	TO	•	TO	•	OM	•	OM
LamBak	3	**6**	104	•	TO	**18**	110	•	OM	**6**	372
LamBak	4	**8**	286	•	TO	•	TO	•	OM	•	OM
Dijkstra	4	**8**	30	•	TO	•	TO	•	OM	•	OM
Dc-Lock	6	**0**	7	**0**	139	**0**	1	•	TO	**0**	5
SpinLock	2	**0**	1	**0**	1	**0**	1	**0**	1	**0**	1
TSpinLock	2	**0**	1	**0**	1	**0**	1	•	OM	**0**	4
InlinePgsql	8	**0**	8	**0**	426	•	TO	•	OM	**0**	42
Burns	5	**9**	119	•	TO	•	TO	•	OM	•	TO
Szymaski	2	**8**	3	**8**	642	**11**	1	**3**	1	**3**	4
Szymaski	4	**16**	88	•	TO	•	TO	•	OM	•	TO
LamFast2	2	**8**	6	**4**	48	**15**	1	**4**	136	**4**	6
LamFast2	3	**12**	10	•	TO	•	TO	•	OM	**6**	108
CLH Lock	4	**4**	176	**4**	674	•	TO	•	OM	•	OM
Parker	6	**1**	76	**1**	425	**3**	1	•	OM	•	OM
Pgsql	6	**7**	26	**7**	260	**7**	1	•	OM	•	TO
AltenatingBit	2	**4**	2	**4**	5	**5**	1	**0**	4	**0**	3
IncSequence	6	**0**	21	**0**	194	**0**	1	•	OM	•	TO
TaskSchedule	10	**0**	5	**0**	418	**0**	1	•	TO	•	OM
NBW_W_WR	2	**0**	18	**1**	595	**6**	1	•	OM	•	TO
NBW_W_5R	6	**0**	3	**0**	514	**0**	1	•	OM	•	TO
SeqLock	4	**0**	63	**0**	364	**0**	1	•	OM	•	TO
write+r	5	**0**	2	**4**	7	**4**	1	•	OM	•	TO
r+detour	5	**0**	1	**3**	3	**3**	1	•	OM	•	TO
r+detours	5	**0**	1	**3**	1	**3**	1	•	OM	•	TO
write+r+co	6	**0**	7	**4**	38	**4**	1	•	OM	•	TO
sb+detours	6	**2**	3	**5**	3	**5**	1	•	OM	•	TO
sb+detour+co	6	**0**	1	**4**	3	**4**	1	•	OM	•	TO
Cilk WSQ	2	**2**	3	**2**	107	**3**	1	–	–	–	–
CL WSQ	2	**1**	2	**1**	796	**1**	1	–	–	–	–
FIFO iWSQ	2	**1**	2	**1**	354	**1**	1	–	–	–	–
LIFO iWSQ	2	**1**	7	**1**	558	**1**	1	–	–	–	–
Anchor iWSQ	2	**1**	2	**1**	20	**1**	1	–	–	–	–

Result without Abstraction

Program	#Process	Persist		Trencher		Musketeer		Memorax		Remmex	
		#Fence	#Time	**#Fence**	#Time	**#Fence**	#Time	**#Fence**	#Time	**#Fence**	#Time
SimDekker	4	**4**	4	**4**	163	**8**	1	•	OM	•	OM
Dekker	4	**8**	14	•	TO	**16**	783	•	OM	•	OM
Peterson	4	**4**	223	•	TO	•	TO	•	OM	•	OM
LamBak	3	**6**	104	•	TO	**18**	110	•	OM	**6**	372
LamBak	4	**8**	286	•	TO	•	TO	•	OM	•	OM
Dijkstra	4	**8**	30	•	TO	•	TO	•	OM	•	OM
Dc-Lock	6	**0**	7	**0**	139	**0**	1	•	TO	**0**	5
SpinLock	2	**0**	1	**0**	1	**0**	1	**0**	1	**0**	1
TSpinLock	2	**0**	1	**0**	1	**0**	1	•	OM	**0**	4
InlinePgsql	8	**0**	8	**0**	426	•	TO	•	OM	**0**	42
Burns	5	**9**	119	•	TO	•	TO	•	OM	•	TO
Szymaski	2	**8**	3	**8**	642	**11**	1	**3**	1	**3**	4
Szymaski	4	**16**	88	•	TO	•	TO	•	OM	•	TO
LamFast2	2	**8**	6	**4**	48	**15**	1	**4**	136	**4**	6
LamFast2	3	**12**	10	•	TO	•	TO	•	OM	**6**	108
CLH Lock	4	**4**	176	**4**	674	•	TO	•	OM	•	OM
Parker	6	**1**	76	**1**	425	**3**	1	•	OM	•	OM
Pgsql	6	**7**	26	**7**	260	**7**	1	•	OM	•	TO
AltenatingBit	2	**4**	2	**4**	5	**5**	1	**0**	4	**0**	3
IncSequence	6	**0**	21	**0**	194	**0**	1	•	OM	•	TO
TaskSchedule	10	**0**	5	**0**	418	**0**	1	•	TO	•	OM
NBW_W_WR	2	**0**	18	**1**	595	**6**	1	•	OM	•	TO
NBW_W_5R	6	**0**	3	**0**	514	**0**	1	•	OM	•	TO
SeqLock	4	**0**	63	**0**	364	**0**	1	•	OM	•	TO
write+r	5	**0**	2	**4**	7	**4**	1	•	OM	•	TO
r+detour	5	**0**	1	**3**	3	**3**	1	•	OM	•	TO
r+detours	5	**0**	1	**3**	1	**3**	1	•	OM	•	TO
write+r+co	6	**0**	7	**4**	38	**4**	1	•	OM	•	TO
sb+detours	6	**2**	3	**5**	3	**5**	1	•	OM	•	TO
sb+detour+co	6	**0**	1	**4**	3	**4**	1	•	OM	•	TO
Cilk WSQ	2	**2**	3	**2**	107	**3**	1	–	–	–	–
CL WSQ	2	**1**	2	**1**	796	**1**	1	–	–	–	–
FIFO iWSQ	2	**1**	2	**1**	354	**1**	1	–	–	–	–
LIFO iWSQ	2	**1**	7	**1**	558	**1**	1	–	–	–	–
Anchor iWSQ	2	**1**	2	**1**	20	**1**	1	–	–	–	–

Program	Memorax		Remmex	
	#Fence	#Time	**#Fence**	#Time
SimDekker	•	OM	•	OM
Dekker	•	OM	•	OM
Peterson	•	OM	•	OM
LamBak	•	OM	**6**	372
LamBak	•	OM	•	OM
Dijkstra	•	OM	•	OM
Dc-Lock	•	TO	**0**	5
SpinLock	**0**	1	**0**	1
TSpinLock	•	OM	**0**	4
InlinePgsql	•	OM	**0**	42
Burns	•	OM	•	TO
Szymaski	**3**	1	**3**	4
Szymaski	•	OM	•	TO
LamFast2	**4**	136	**4**	6
LamFast2	•	OM	**6**	108
CLH Lock	•	OM	•	OM
Parker	•	OM	•	OM
Pgsql	•	OM	•	TO
AltenatingBit	**0**	4	**0**	3
IncSequence	•	OM	•	TO
TaskSchedule	•	TO	•	OM
NBW_W_WR	•	OM	•	TO
NBW_W_5R	•	OM	•	TO
SeqLock	•	OM	•	TO
write+r	•	OM	•	TO
r+detour	•	OM	•	TO
r+detours	•	OM	•	TO
write+r+co	•	OM	•	TO
sb+detours	•	OM	•	TO
sb+detour+co	•	OM	•	TO
Cilk WSQ	–	–	–	–
CL WSQ	–	–	–	–
FIFO iWSQ	–	–	–	–
LIFO iWSQ	–	–	–	–
Anchor iWSQ	–	–	–	–

Memorax: Uppsala
Remmex: Liege

Program	Memorax		Remmex	
	#Fence	#Time	**#Fence**	#Time
SimDekke	•	OM	•	OM
Dekker	•	OM	•	OM
Peterson	•	OM	•	OM
LamBak	•	OM	**6**	372
LamBak	•	OM	•	OM
Dijkstra	•	OM	•	OM
Dc-Lock	•	TO	**0**	5
SpinLock	**0**	1	**0**	1
TSpinLoc	•	OM	**0**	4
InlinePgsq	•	OM	**0**	42
Burns	•	OM	•	TO
Szymaski	**3**	1	**3**	4
Szymaski	•	OM	•	TO
LamFast2	**4**	136	**4**	6
LamFast2	•	OM	**6**	108
CLH Lock	•	OM	•	OM
Parker	•	OM	•	OM
Pgsql	•	OM	•	TO
Altenating	**0**	4	**0**	3
IncSequen	•	OM	•	TO
TaskSched	•	TO	•	OM
NBW_W_	•	OM	•	TO
NBW_W_5	•	OM	•	TO
SeqLock	•	OM	•	TO
write+r	•	OM	•	TO
r+detour	•	OM	•	TO
r+detours	•	OM	•	TO
write+r+c	•	OM	•	TO
sb+detour	•	OM	•	TO
sb+detour	•	OM	•	TO
Cilk WSQ	–	–	–	–
CL WSQ	–	–	–	–
FIFO iWS	–	–	–	–
LIFO iWS	–	–	–	–
Anchor iW	–	–	–	–

Result without Abstraction

Program	#Process	Persist		Trencher		Musketeer		Memorax		Remmex	
		#Fence	#Time	**#Fence**	#Time	**#Fence**	#Time	**#Fence**	#Time	**#Fence**	#Time
SimDekker	4	**4**	4	**4**	163	**8**	1	•	OM	•	OM
Dekker	4	**8**	14	•	TO	**16**	783	•	OM	•	OM
Peterson	4	**4**	223	•	TO	•	TO	•	OM	•	OM
LamBak	3	**6**	104	•	TO	**18**	110	•	OM	**6**	372
LamBak	4	**8**	286	•	TO	•	TO	•	OM	•	OM
Dijkstra	4	**8**	30	•	TO	•	TO	•	OM	•	OM
Dc-Lock	6	**0**	7	**0**	139	**0**	1	•	TO	**0**	5
SpinLock	2	**0**	1	**0**	1	**0**	1	**0**	1	**0**	1
TSpinLock	2	**0**	1	**0**	1	**0**	1	•	OM	**0**	4
InlinePgsql	8	**0**	8	**0**	426	•	TO	•	OM	**0**	42
Burns	5	**9**	119	•	TO	•	TO	•	OM	•	TO
Szymaski	2	**8**	3	**8**	642	**11**	1	**3**	1	**3**	4
Szymaski	4	**16**	88	•	TO	•	TO	•	OM	•	TO
LamFast2	2	**8**	6	**4**	48	**15**	1	**4**	136	**4**	6
LamFast2	3	**12**	10	•	TO	•	TO	•	OM	**6**	108
CLH Lock	4	**4**	176	**4**	674	•	TO	•	OM	•	OM
Parker	6	**1**	76	**1**	425	**3**	1	•	OM	•	OM
Pgsql	6	**7**	26	**7**	260	**7**	1	•	OM	•	TO
AltenatingBit	2	**4**	2	**4**	5	**5**	1	**0**	4	**0**	3
IncSequence	6	**0**	21	**0**	194	**0**	1	•	OM	•	TO
TaskSchedule	10	**0**	5	**0**	418	**0**	1	•	TO	•	OM
NBW_W_WR	2	**0**	18	**1**	595	**6**	1	•	OM	•	TO
NBW_W_5R	6	**0**	3	**0**	514	**0**	1	•	OM	•	TO
SeqLock	4	**0**	63	**0**	364	**0**	1	•	OM	•	TO
write+r	5	**0**	2	**4**	7	**4**	1	•	OM	•	TO
r+detour	5	**0**	1	**3**	3	**3**	1	•	OM	•	TO
r+detours	5	**0**	1	**3**	1	**3**	1	•	OM	•	TO
write+r+co	6	**0**	7	**4**	38	**4**	1	•	OM	•	TO
sb+detours	6	**2**	3	**5**	3	**5**	1	•	OM	•	TO
sb+detour+co	6	**0**	1	**4**	3	**4**	1	•	OM	•	TO
Cilk WSQ	2	**2**	3	**2**	107	**3**	1	–	–	–	–
CL WSQ	2	**1**	2	**1**	796	**1**	1	–	–	–	–
FIFO iWSQ	2	**1**	2	**1**	354	**1**	1	–	–	–	–
LIFO iWSQ	2	**1**	7	**1**	558	**1**	1	–	–	–	–
Anchor iWSQ	2	**1**	2	**1**	20	**1**	1	–	–	–	–

Result without Abstraction

Trencher: Kaiserslautern
Musketeer: Offoxd

Program	#P	Persist		Trencher		Musketeer	
		#Fence	#Time	**#Fence**	#Time	**#Fence**	#Time
Dekker	4	**8**	14	•	TO	**16**	783
Peterson	4	**4**	223	•	TO	•	TO
LamBak	3	**6**	104	•	TO	**18**	110
LamBak	4	**8**	286	•	TO	•	TO
Dijkstra	4	**8**	30	•	TO	•	TO
Dc-Lock	6	**0**	7	**0**	139	**0**	1
SpinLock	2	**0**	1	**0**	1	**0**	1
TSpinLock	2	**0**	1	**0**	1	**0**	1
InlinePgsql	8	**0**	8	**0**	426	•	TO
Burns	5	**9**	119	•	TO	•	TO
Szymaski	2	**8**	3	**8**	642	**11**	1

Trencher uses Spin

Musketter uses static analysis

Result without Abstraction

Program	#Process	Persist		Trencher		Musketeer		Memorax		Remmex	
		#Fence	#Time	**#Fence**	#Time	**#Fence**	#Time	**#Fence**	#Time	**#Fence**	#Time
SimDekker	4	**4**	4	**4**	163	**8**	1	•	OM	•	OM
Dekker	4	**8**	14	•	TO	**16**	783	•	OM	•	OM
Peterson	4	**4**	223	•	TO	•	TO	•	OM	•	OM
LamBak	3	**6**	104	•	TO	**18**	110	•	OM	**6**	372
LamBak	4	**8**	286	•	TO	•	TO	•	OM	•	OM
Dijkstra	4	**8**	30	•	TO	•	TO	•	OM	•	OM
Dc-Lock	6	**0**	7	**0**	139	**0**	1	•	OM	•	TO
SpinLock	2	**0**	1	**0**	1	**0**	1	•	TO	•	TO
TSpinLock	2	**0**	1	**0**	1	**0**	1	•	TO	•	TO
InlinePgsql	8	**0**	8	**0**	426	•	TO	•	TO	•	TO
Burns	5	**9**	119	•	TO	•	TO	•	TO	•	TO
Szymaski	2	**8**	3	**8**	642	**11**	1	•	TO	•	TO
r+detour	5	**0**	1	**3**	3	**3**	1	•	OM	•	TO
r+detours	5	**0**	1	**3**	1	**3**	1	•	OM	•	TO
write+r+co	6	**0**	7	**4**	38	**4**	1	•	OM	•	TO
sb+detours	6	**2**	3	**5**	3	**5**	1	•	OM	•	TO
sb+detour+co	6	**0**	1	**4**	3	**4**	1	•	OM	•	TO
Cilk WSQ	2	**2**	3	**2**	107	**3**	1	–	–	–	–
CL WSQ	2	**1**	2	**1**	796	**1**	1	–	–	–	–
FIFO iWSQ	2	**1**	2	**1**	354	**1**	1	–	–	–	–
LIFO iWSQ	2	**1**	7	**1**	558	**1**	1	–	–	–	–
Anchor iWSQ	2	**1**	2	**1**	20	**1**	1	–	–	–	–

Result without Abstraction

Program	#Process	Persist		Trencher		Musketeer		Memorax		Remmex	
		#Fence	#Time	**#Fence**	#Time	**#Fence**	#Time	**#Fence**	#Time	**#Fence**	#Time
SimDekker	4	**4**	4	**4**	163	**8**	1	•	OM	•	OM
Dekker	4	**8**	14	•	TO	**16**	783	•	OM	•	OM
Peterson	4	**4**	223	•	TO	•	TO	•	OM	•	OM
LamBak	3	**6**	104	•	TO	**18**	110	•	OM	**6**	372
LamBak	4	**8**	286	•	TO	•	TO	•	OM	•	OM
Dijkstra	4	**8**	30	•	TO	•	TO	•	OM	•	OM
Dc-Lock	6	**0**	7	**0**	139	**0**	1	–	–	–	–
SpinLock	2	**0**	1	**0**	1	**0**	1	–	–	–	–
TSpinLock	2	**0**	1	**0**	1	**0**	1	–	–	–	–
InlinePgsql	8	**0**	8	**0**	426	•	TO	–	–	–	–
Burns	5	**9**	119	•	TO	•	TO	–	–	–	–
Szymaski	2	**8**	3	**8**	642	**11**	1	–	–	–	–
r+detour	5	**0**	1	**3**	3	**3**	1	•	OM	•	TO
r+detours	5	**0**	1	**3**	1	**3**	1	•	OM	•	TO
write+r+co	6	**0**	7	**4**	38	**4**	1	•	OM	•	TO
sb+detours	6	**2**	3	**5**	3	**5**	1	•	OM	•	TO
sb+detour+co	6	**0**	1	**4**	3	**4**	1	•	OM	•	TO
Cilk WSQ	2	**2**	3	**2**	107	**3**	1	–	–	–	–
CL WSQ	2	**1**	2	**1**	796	**1**	1	–	–	–	–
FIFO iWSQ	2	**1**	2	**1**	354	**1**	1	–	–	–	–
LIFO iWSQ	2	**1**	7	**1**	558	**1**	1	–	–	–	–
Anchor iWSQ	2	**1**	2	**1**	20	**1**	1	–	–	–	–

Result without Abstraction

Program	#Process	Persist		Trencher		Musketeer		Memorax		Remmex	
		#Fence	#Time	**#Fence**	#Time	**#Fence**	#Time	**#Fence**	#Time	**#Fence**	#Time
SimDekker	4	**4**	4	**4**	163	**8**	1	•	OM	•	OM
Dekker	4	**8**	14	•	TO	**16**	783	•	OM	•	OM
Peterson	4	**4**	223	•	TO	•	TO	•	OM	•	OM
LamBak	3	**6**	104	•	TO	**18**	110	•	OM	**6**	372
LamBak	4	**8**	286	•	TO	•	TO	•	OM	•	OM
Dijkstra	4	**8**	30	•	TO	•	TO	•	OM	•	OM
Dc-Lock	6	**0**	7	**0**	139	**0**	1	**0**	1	**0**	5
SpinLock	2	**0**	1	**0**	1	**0**	1	**0**	1	**0**	1
TSpinLock	2	**0**	1	**0**	1	**0**	1	**0**	1	**0**	3
InlinePgsql	8	**0**	8	**0**	426	•	TO	•	TO	•	TO
Burns	5	**9**	119	•	TO	•	TO	•	TO	•	TO
Szymaski	2	**8**	3	**8**	642	**11**	1	•	TO	•	TO
r+detour	5	**0**	1	**3**		**3**	1	•	OM	•	TO
r+detours	5	**0**	1	**3**		**3**	1	•	OM	•	TO
write+r+co	6	**0**	7	**4**		**4**	1	•	OM	•	TO
sb+detours	6	**2**	3	**5**		**5**	1	•	OM	•	TO
sb+detour+co	6									•	TO
Cilk WSQ	2									–	–
CL WSQ	2									–	–
FIFO iWSQ	2									–	–
LIFO iWSQ	2									–	–
Anchor iWSQ	2									–	–

Persist is a good trade off between optimality and efficiency

Result with Abstraction

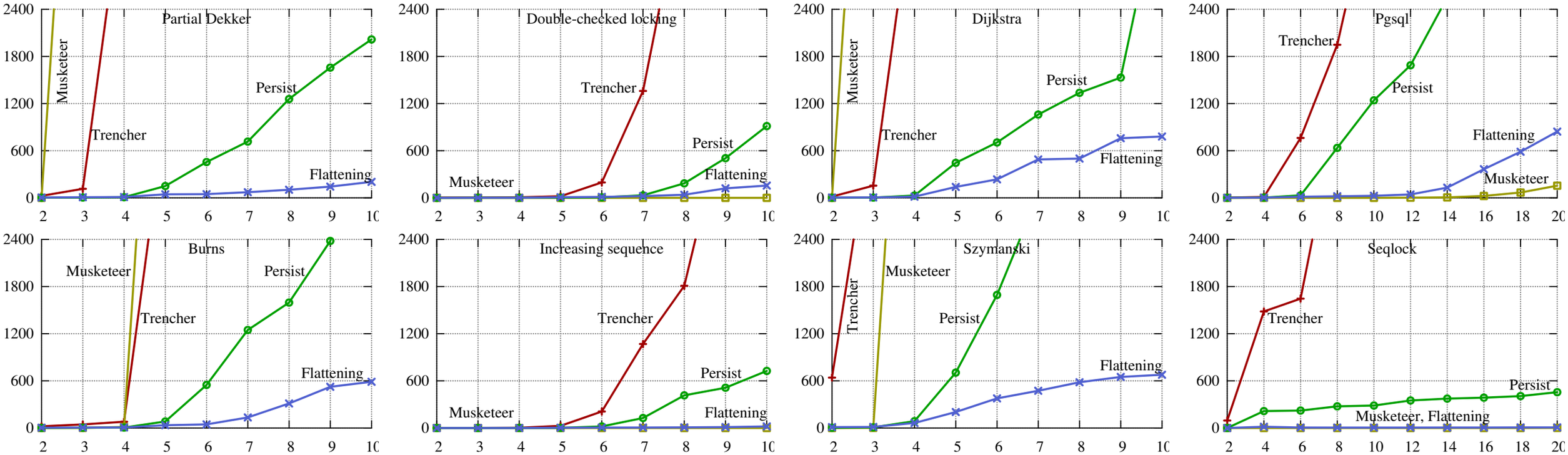


Figure 1: Persist with/without abstraction compared with Trencher and Musketeer. The x axis is number of processes, the y axis is running time in seconds.

Result with Abstraction

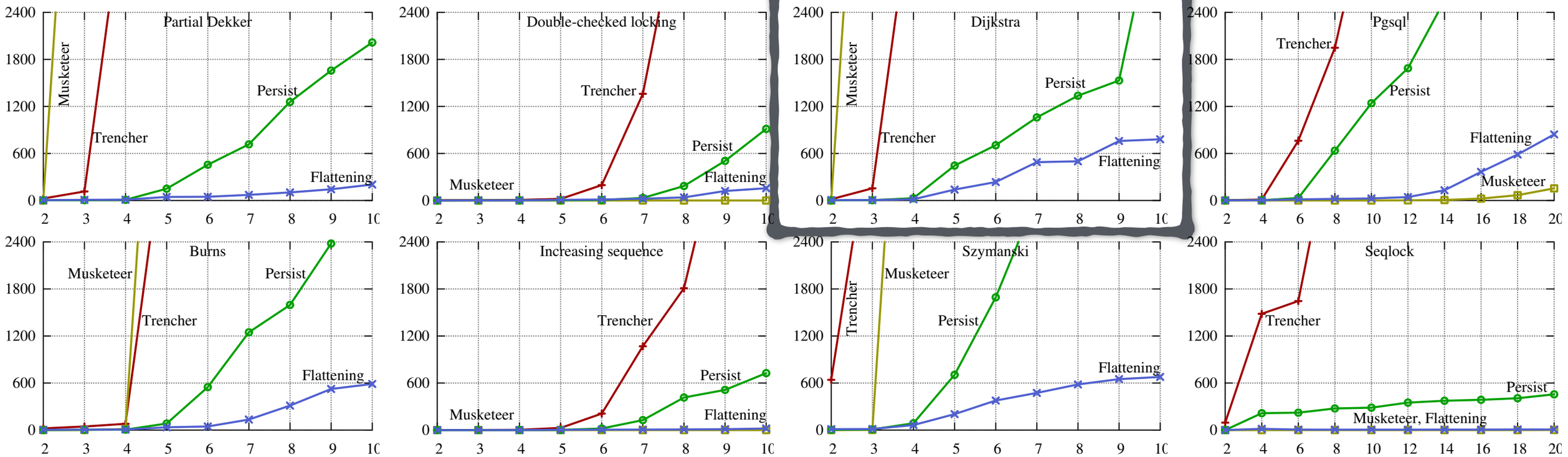


Figure 1: Persist with/without abstraction compared with Trencher and Musketeer. The x axis is number of processes, the y axis is running time in seconds.

Result with Abstraction

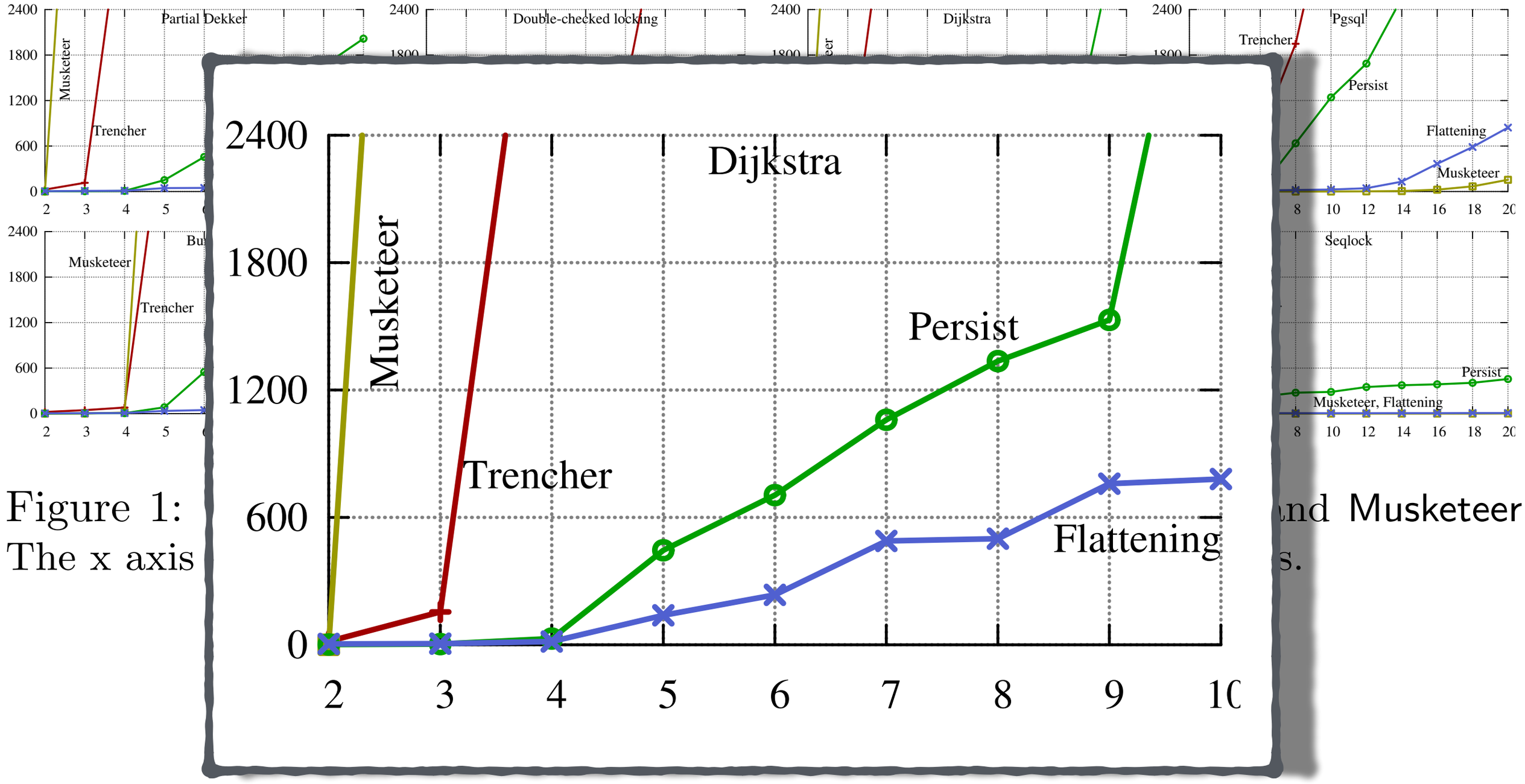


Figure 1: The x axis

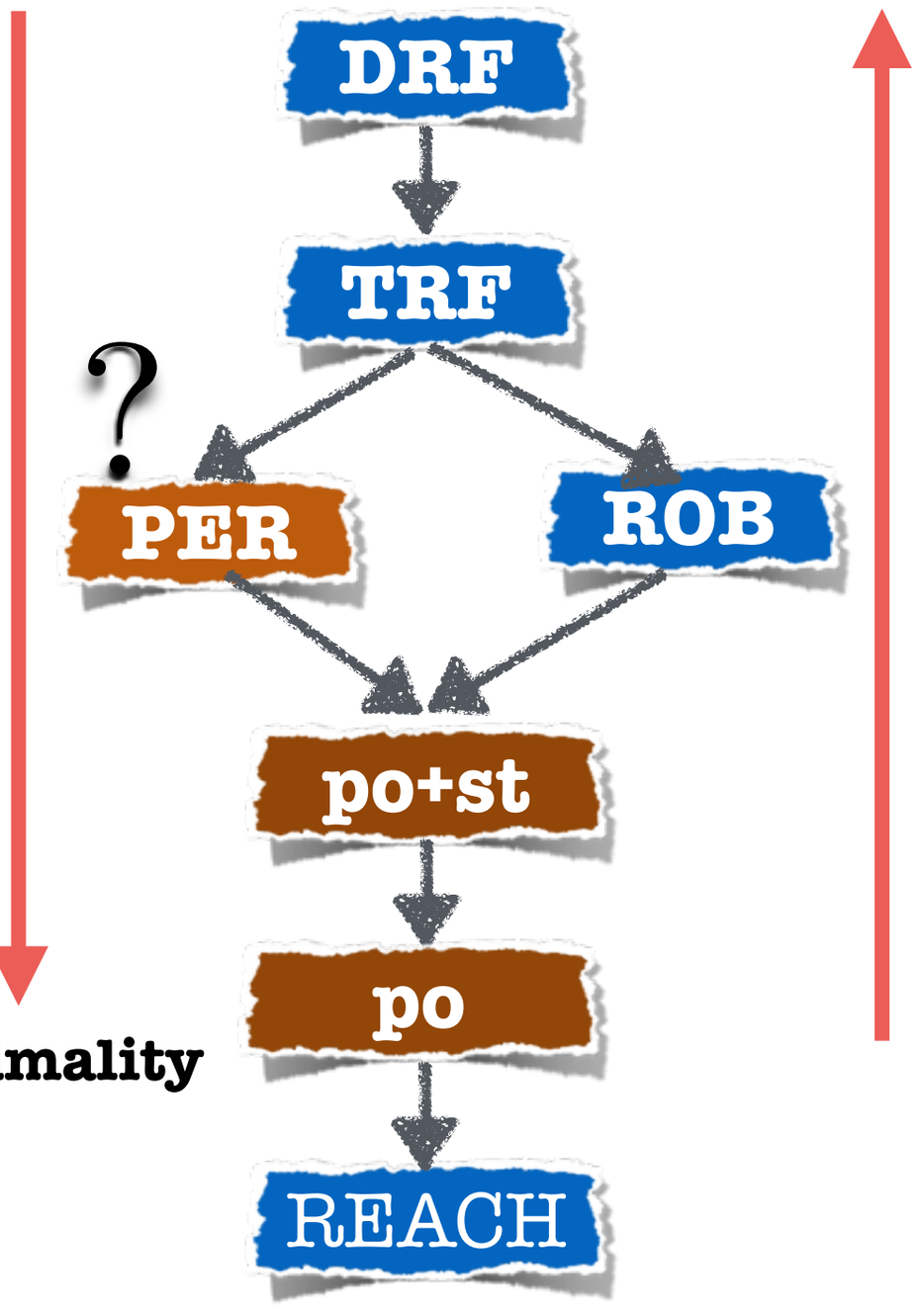
nd Musketeer.
s.

Conclusion

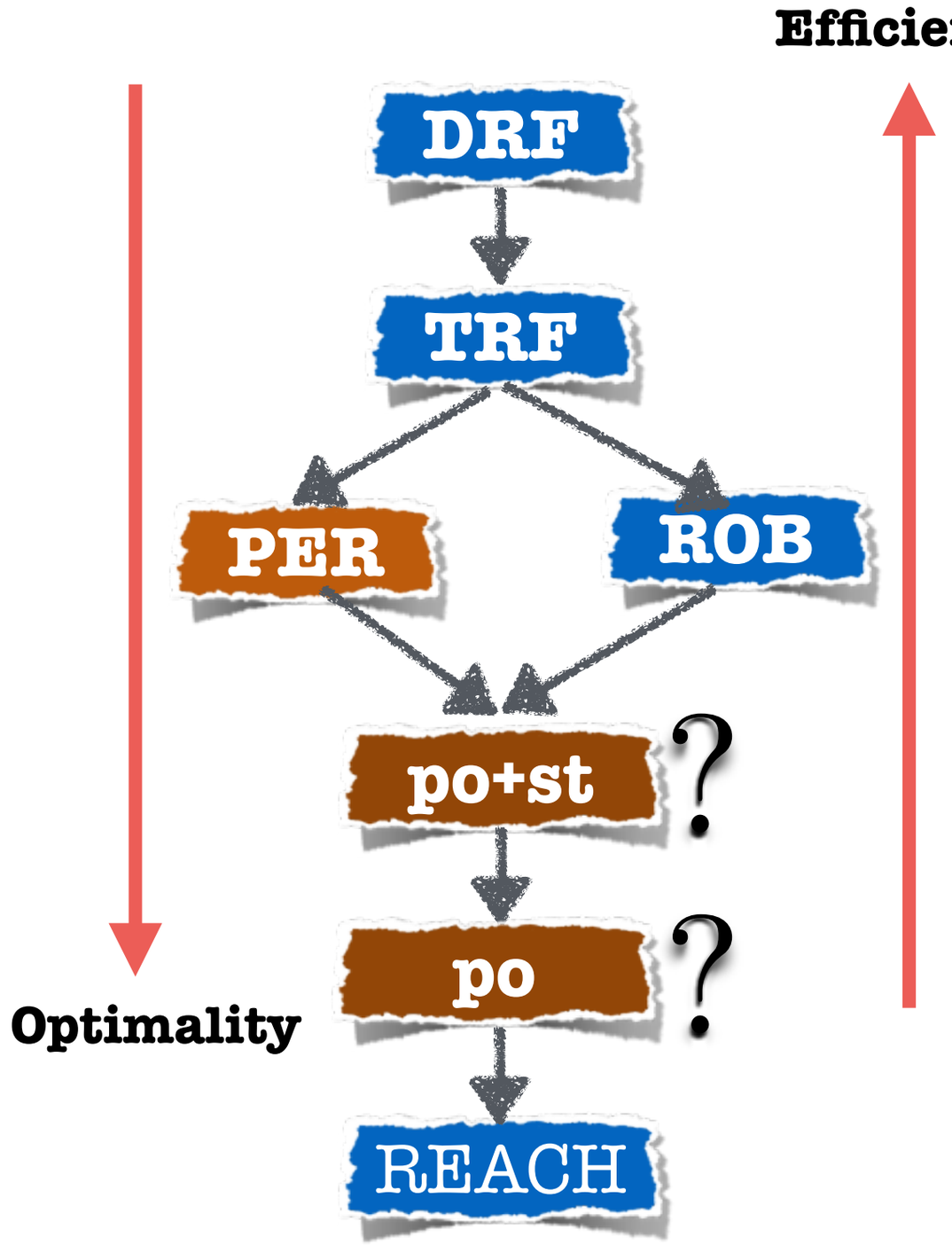
- ✦ New Correctness Criteria
- ✦ An Algorithm for Checking Trace-Persistence under TSO
- ✦ A Counter-Example Guided Fence Insertion Procedure
- ✦ Efficient Abstraction Techniques
- ✦ An Open Source Tool

Future Work

May not fit for PSO, ARM, Power



Future Work



Other models:
ARM, Power?

Thank you!