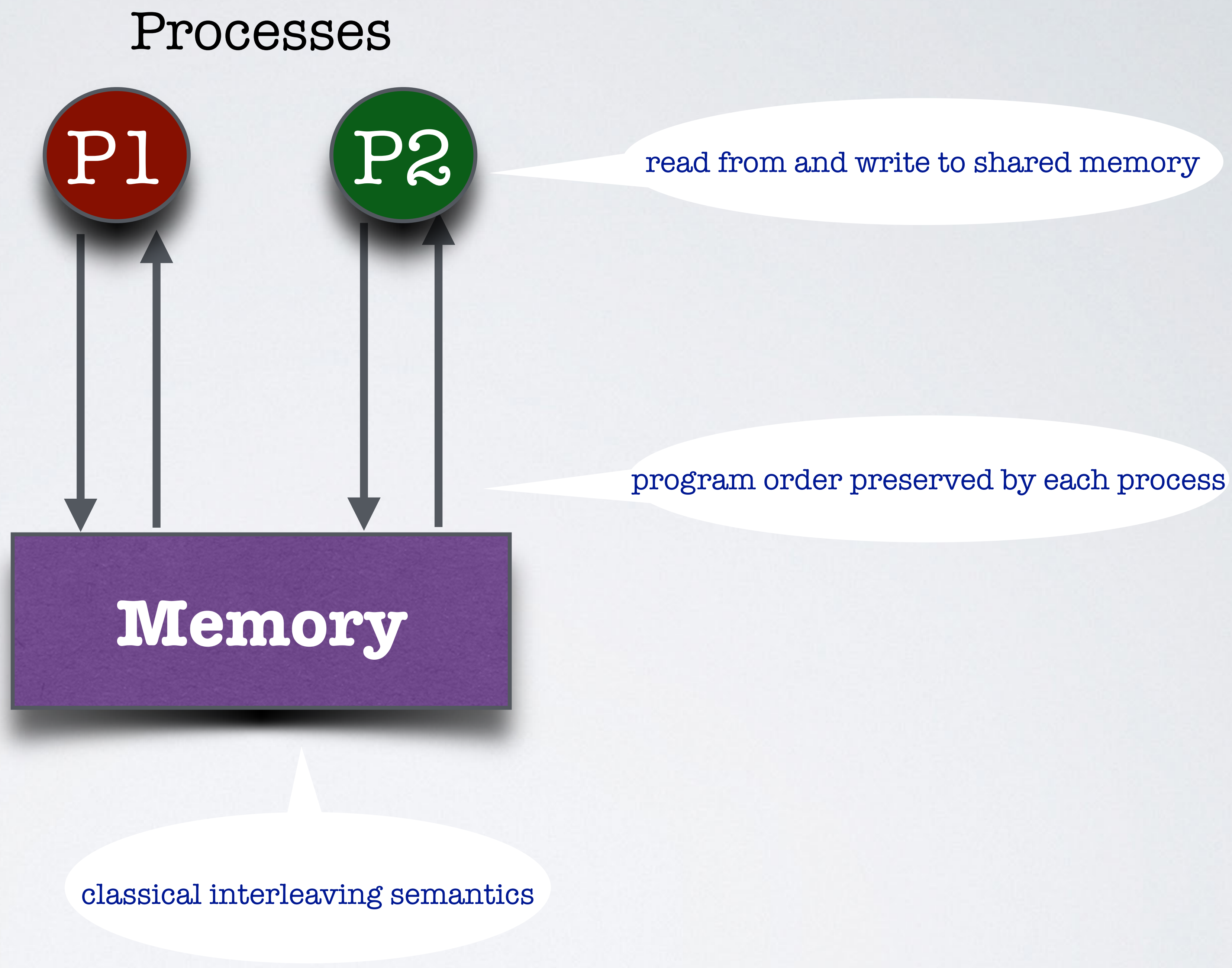
Verification of Concurrent Programs under Release Acquire

S. Krishna

Sequential Consistency


Processes
P1
P2
Memory
read from and write to shared memory
program order preserved by each process
classical interleaving semantics

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

An SC execution

x=0 y=0

Shared memory

Specification S: not (cs1 && cs2)

Dekker Mutual Exclusion Protocol

Init: x=y=0

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

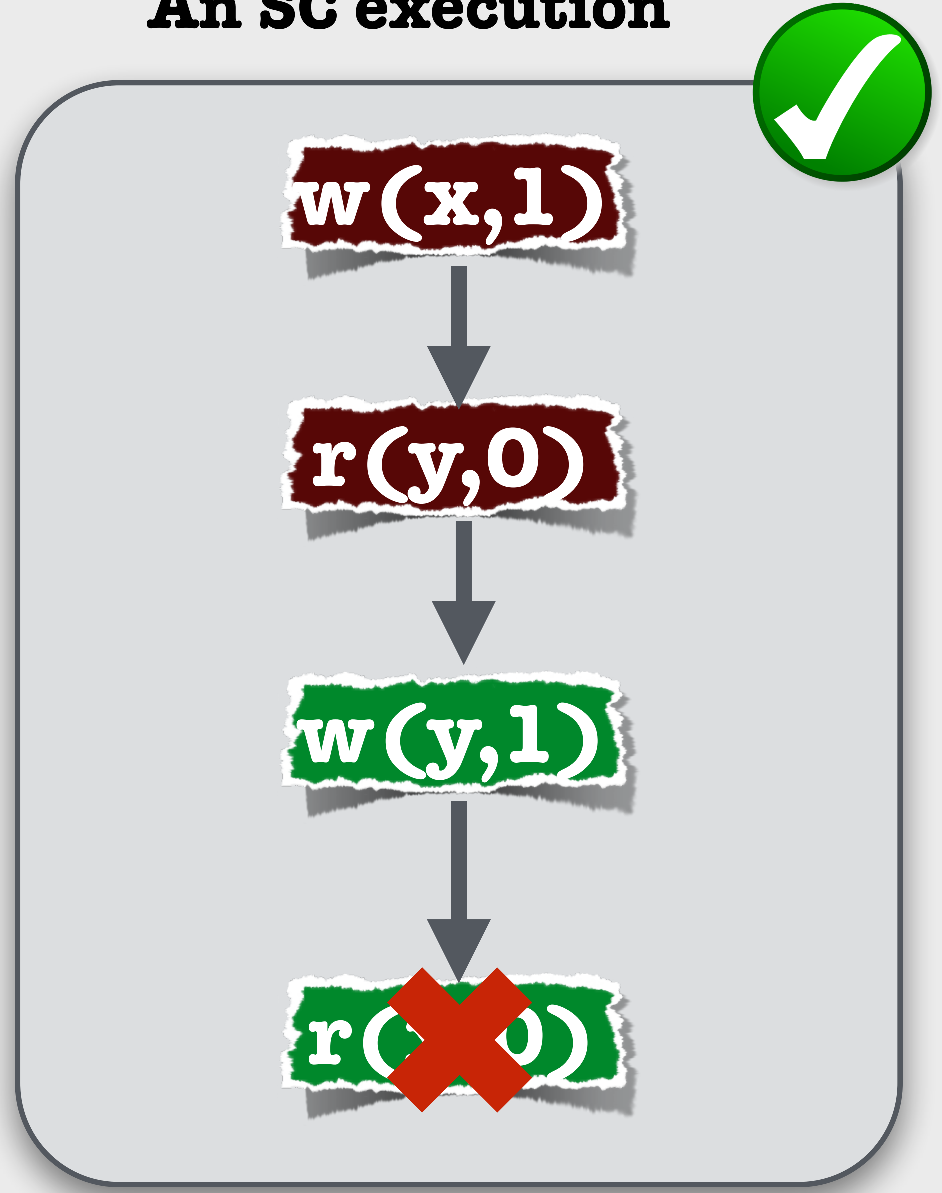
Process 1

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

Process 2

Specification S: not (cs1 && cs2)

An SC execution

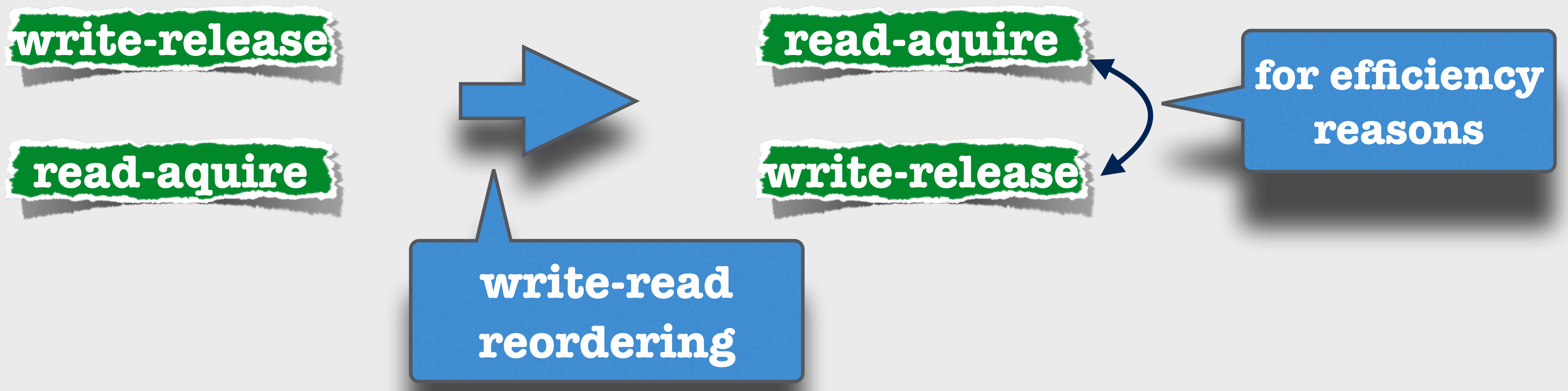


x=0 y=0

Shared memory

Weak Memory Models

- Modern processors and/or compilers:
  - Reorder instructions
  - Use caches and buffers
- Behaviors described by **weak memory models**:
  - The Release-Aquire fragment of C11



Potential Bad Behaviours

A RA execution

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

x=0   y=0

Shared memory

Specification S: not (cs1 && cs2)

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

A RA execution

x=0   y=0

Shared memory

Specification S: not (cs1 && cs2)

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

A RA execution

$$\begin{tikzcd}
r(x,0) \arrow[d] \arrow[ddd, bend left, leftrightarrow] \\
w(x,1) \arrow[d] \\
r(y,0) \arrow[d] \\
w(y,1)
\end{tikzcd}$$

x=0 y=0

Shared memory

Specification S: not (cs1 && cs2)

Potential Bad Behaviours

Init: x=y=0

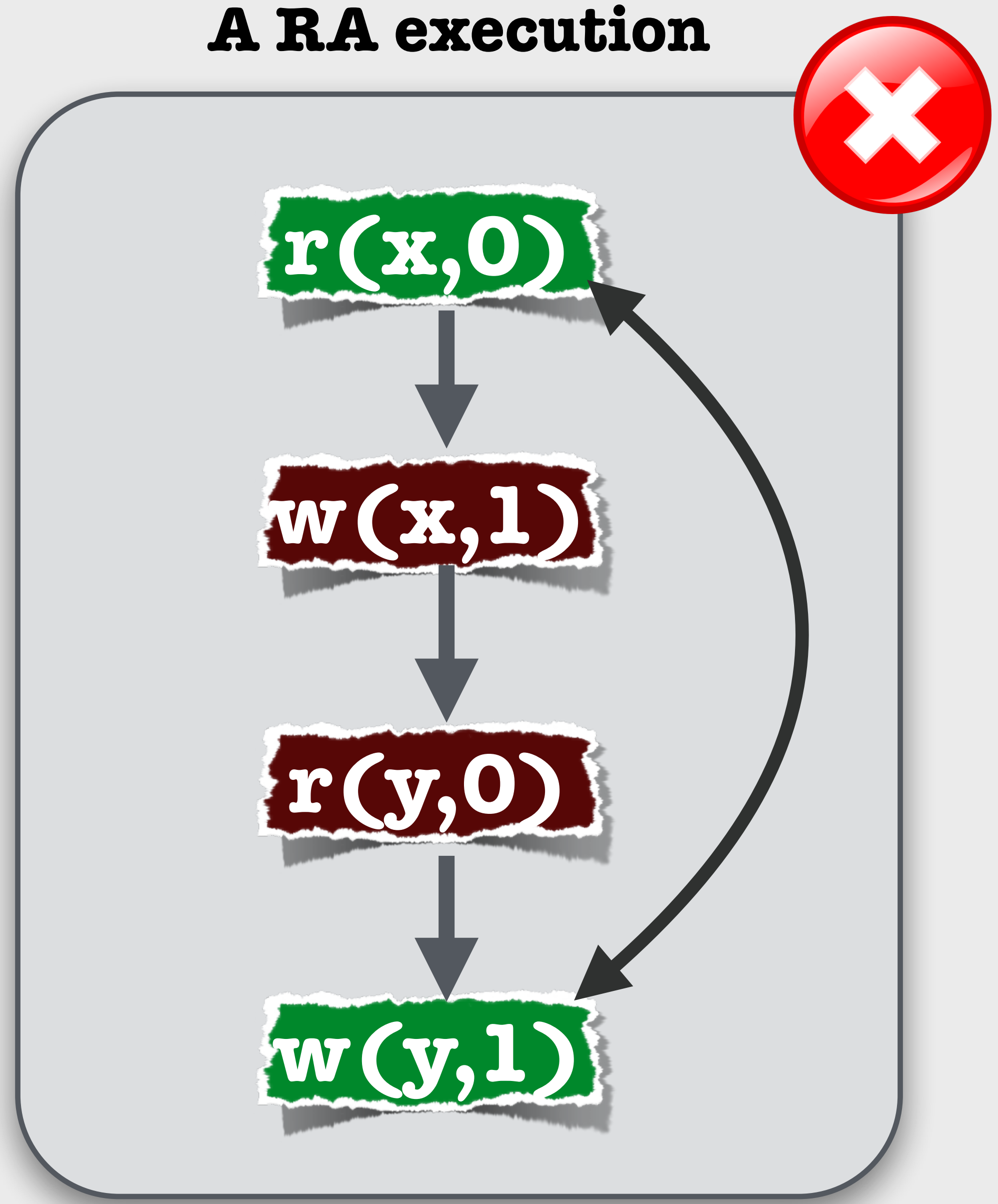
```
1. x=1;
2. ry=y;
3. if (ry==0) {
4. //cs1;
5. }
```

Process 1

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

Process 2

A RA execution



x=0 y=0

Shared memory

Specification S: not (cs1 && cs2)

Problem of Interest

Given a **program P** and a (control + memory) **state s**

- State Reachability Problem (Safety)

  Is **s** reachable in **P** under **RA**?

Decidability/ Complexity?

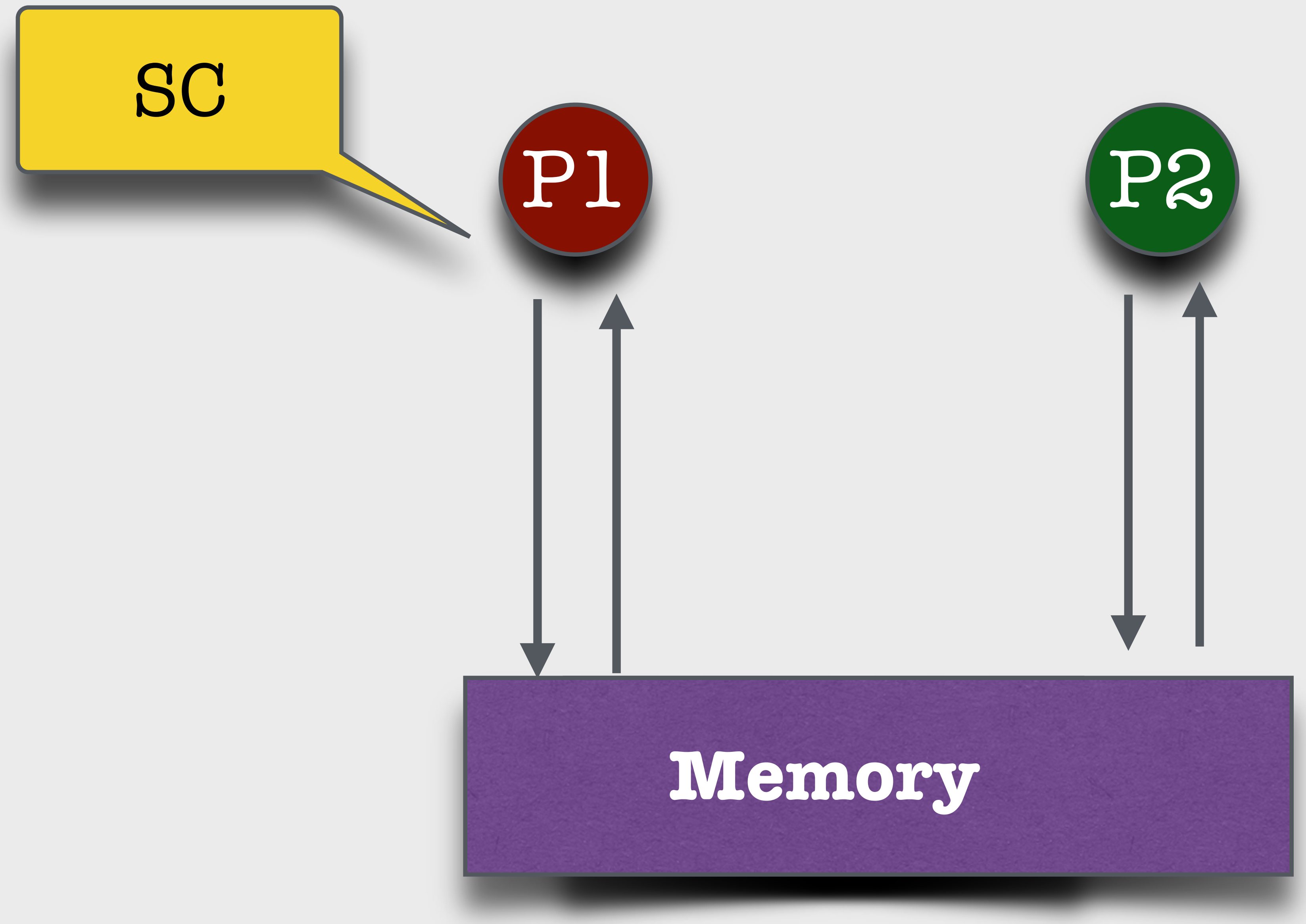
Each process is finite-state

- For **SC**, the reachability problem is PSPACE-complete
- Nontrivial for **RA** since the set of paths is nonregular

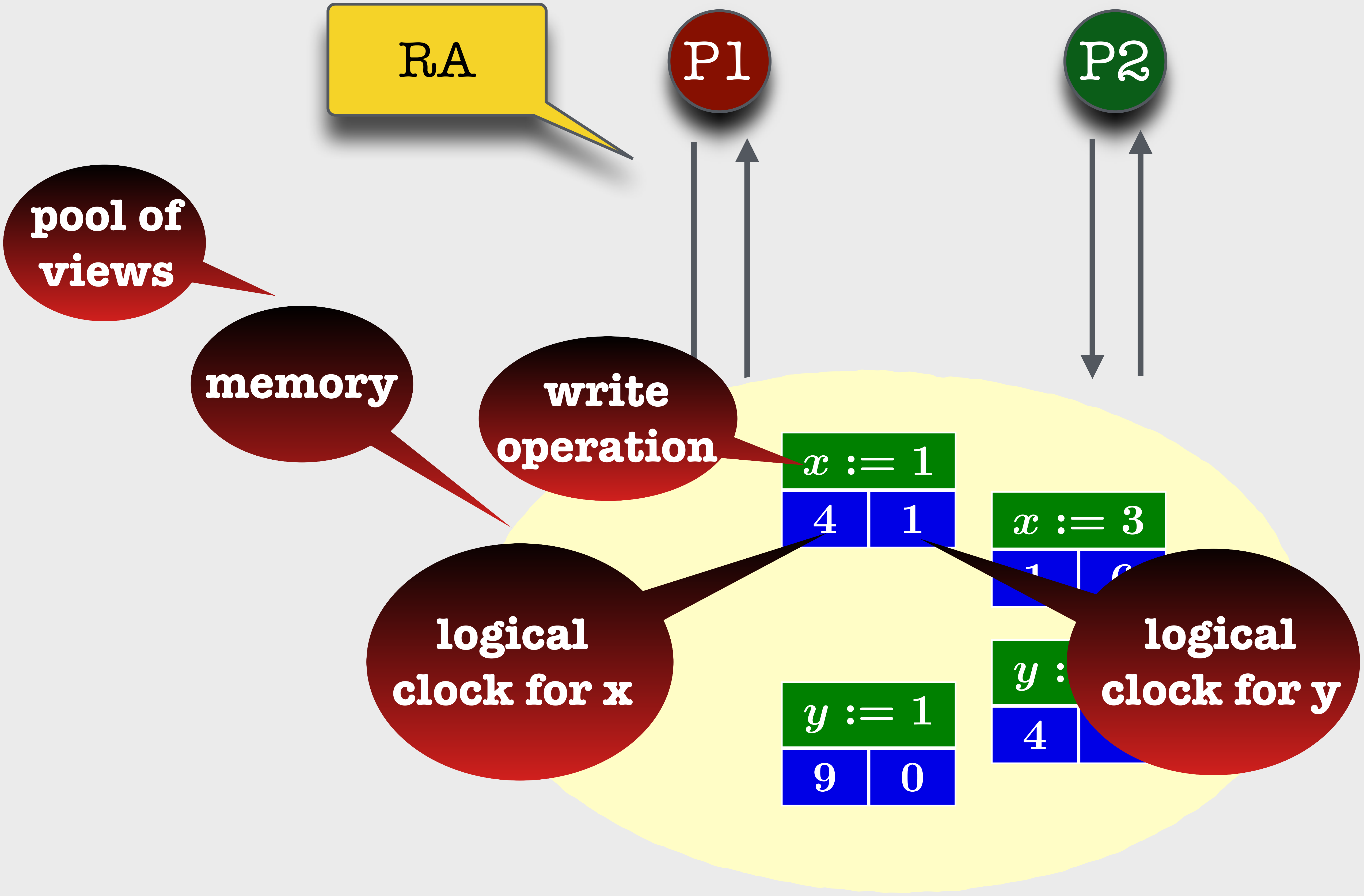
Operational Model for RA

[J. Kang et al. POPL 2017, A. Podkopaev et al. 2016, Arxiv]

RA: High Level Description



RA: High Level Description

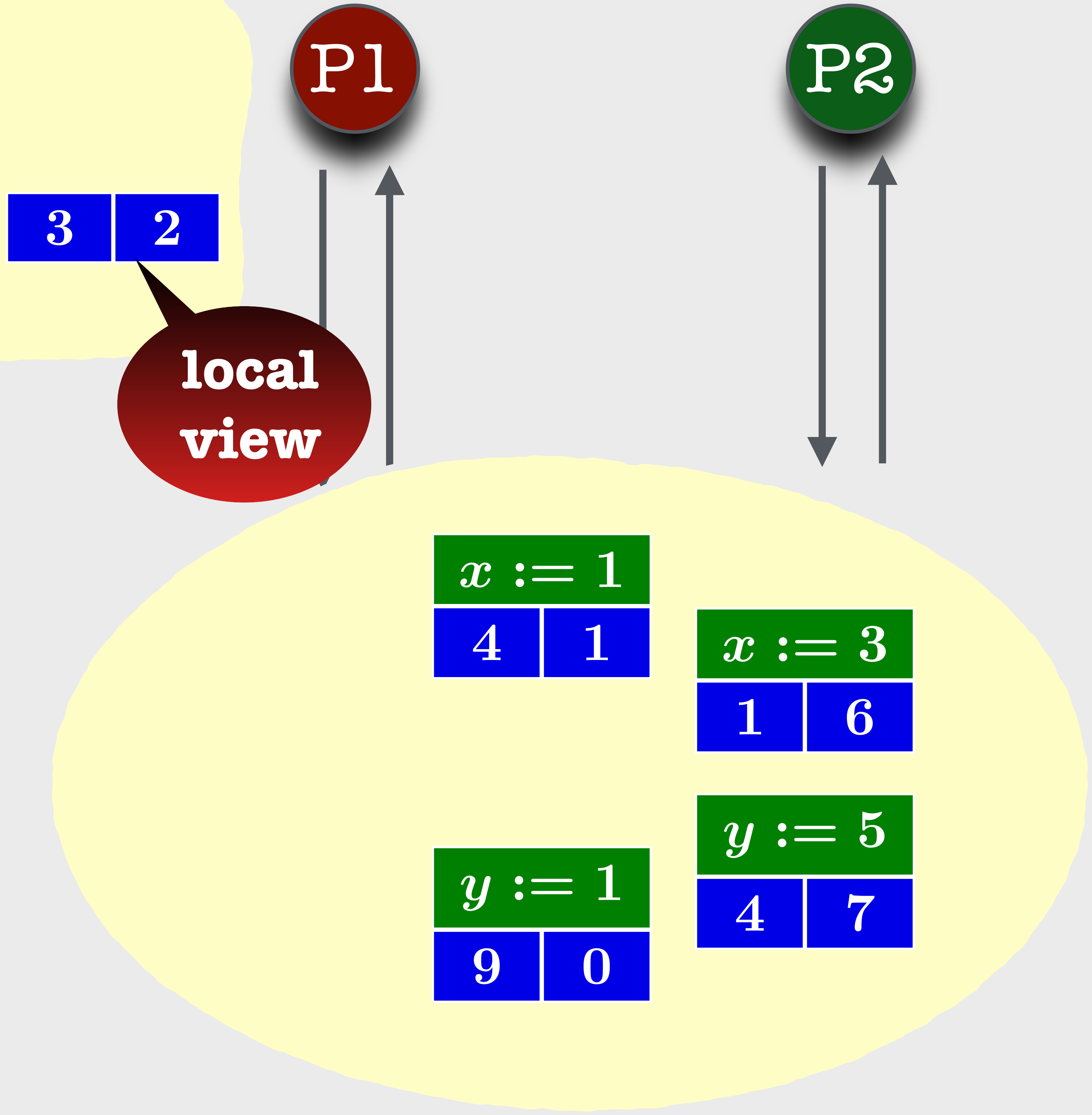


RA: High Level Description

P_1 : $a := x$

P_1 : $x := 2$

P_1 : $b := y$



RA: High Level Description

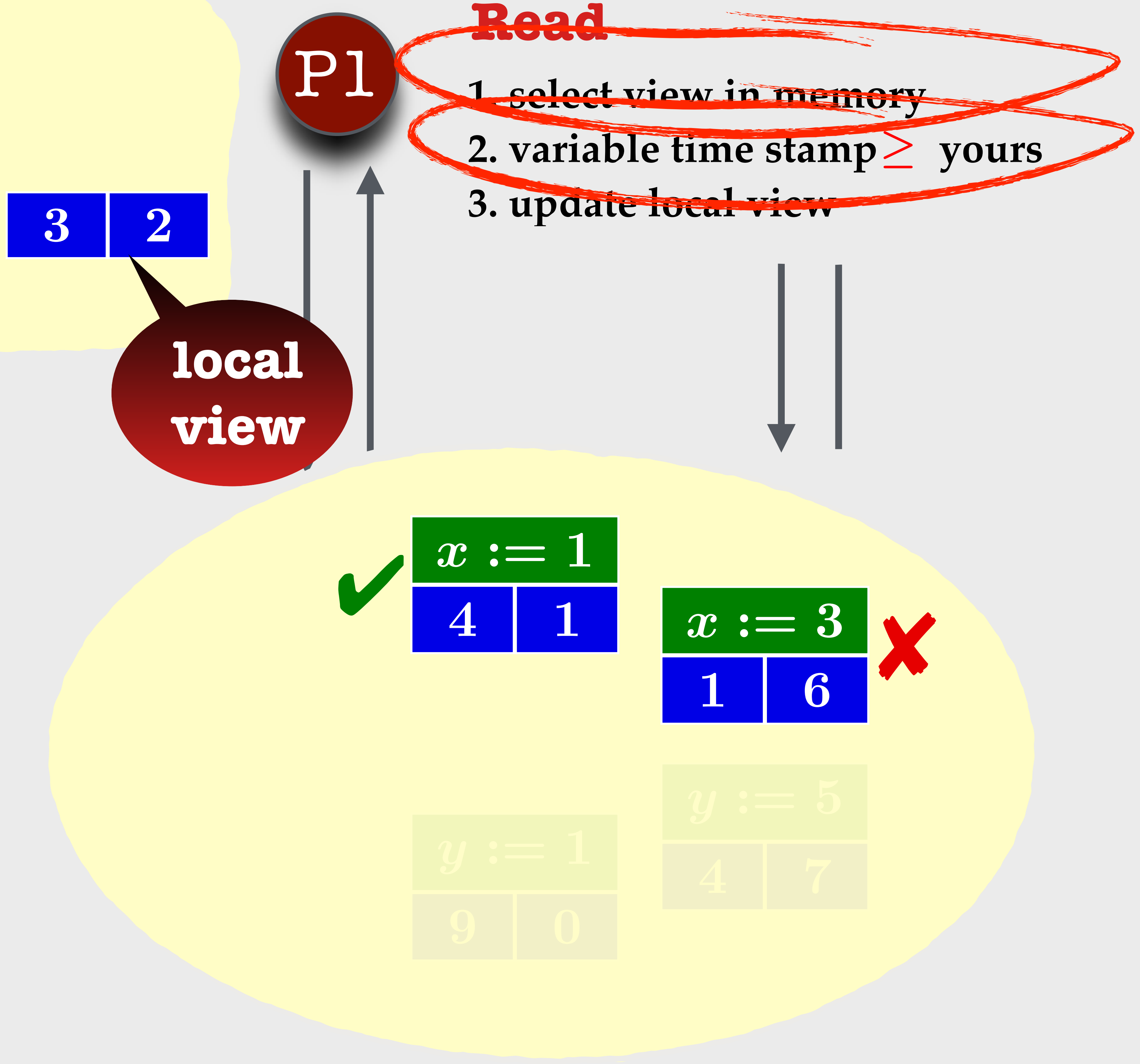
P_1 : $a := x$

P_1 : $x := 2$

P_1 : $b := y$

Read

1. select view in memory
2. variable time stamp $\geq$ yours
3. update local view



RA: High Level Description

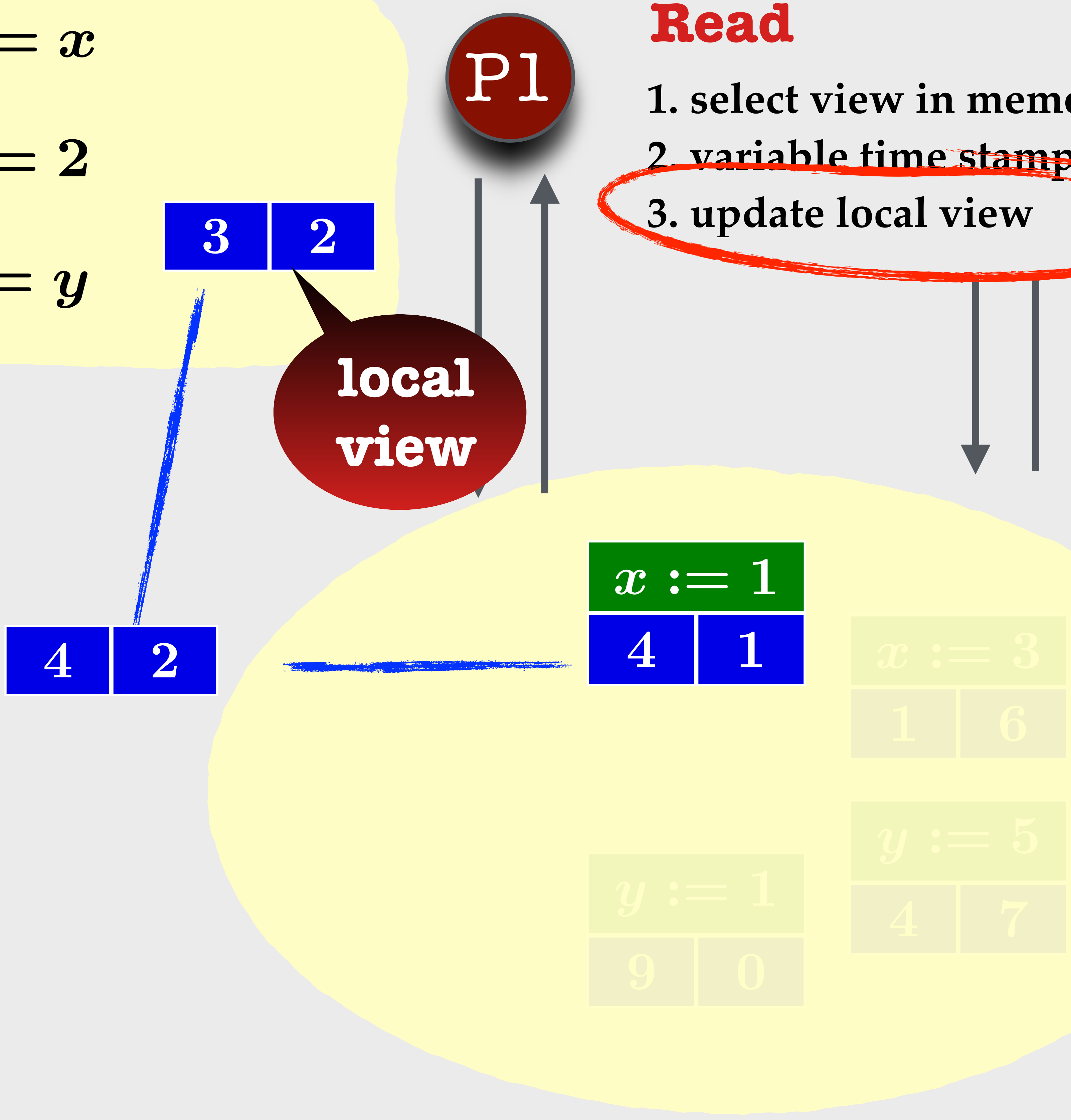
P_1 : $a := x$

P_1 : $x := 2$

P_1 : $b := y$

Read

1. select view in memory
2. variable time stamp $\geq$ yours
3. update local view



RA: High Level Description

P_1 : $a := x$

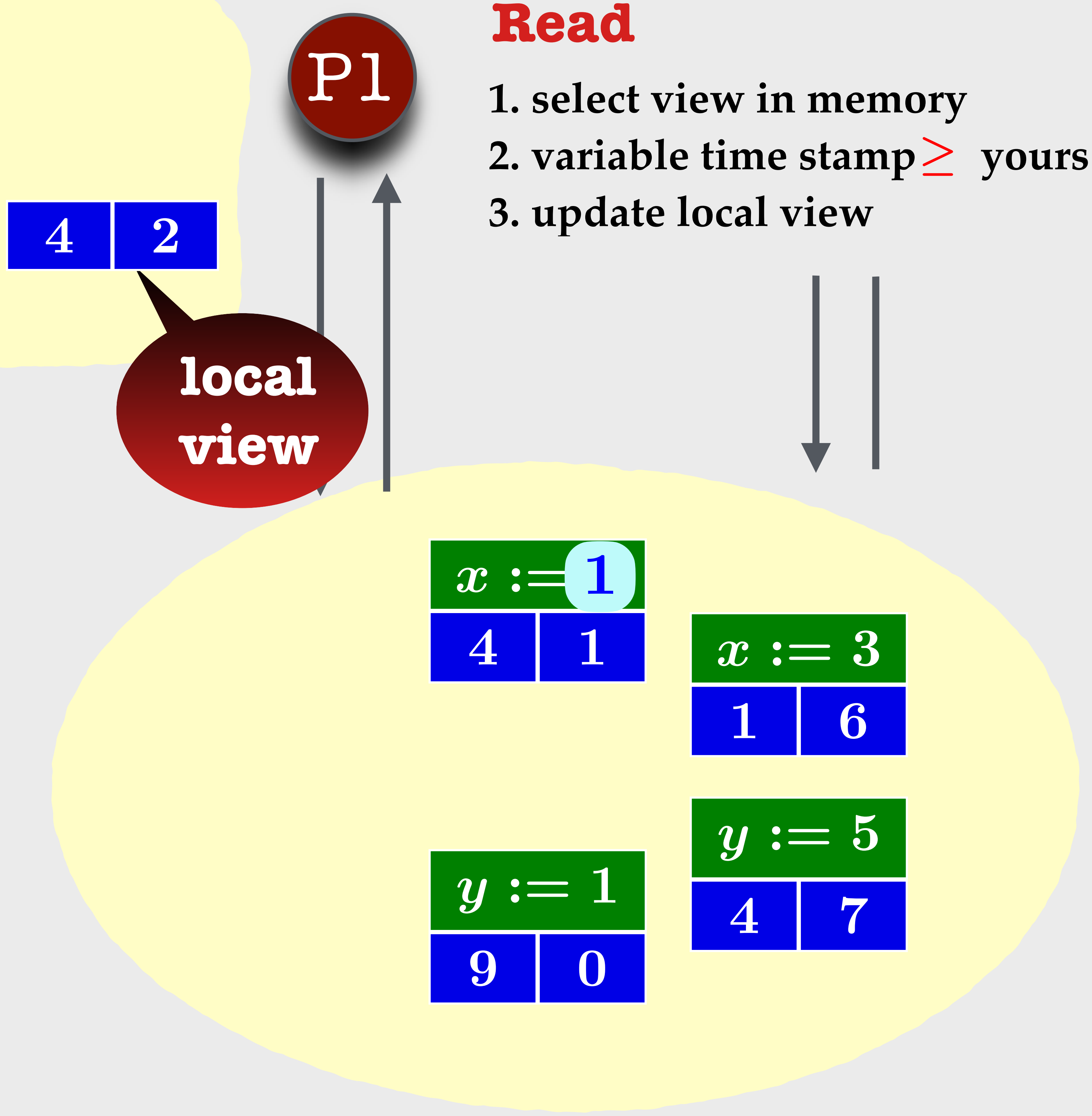
▶

P_1 : $x := 2$

P_1 : $b := y$

Read

1. select view in memory
2. variable time stamp $\geq$ yours
3. update local view



RA: High Level Description

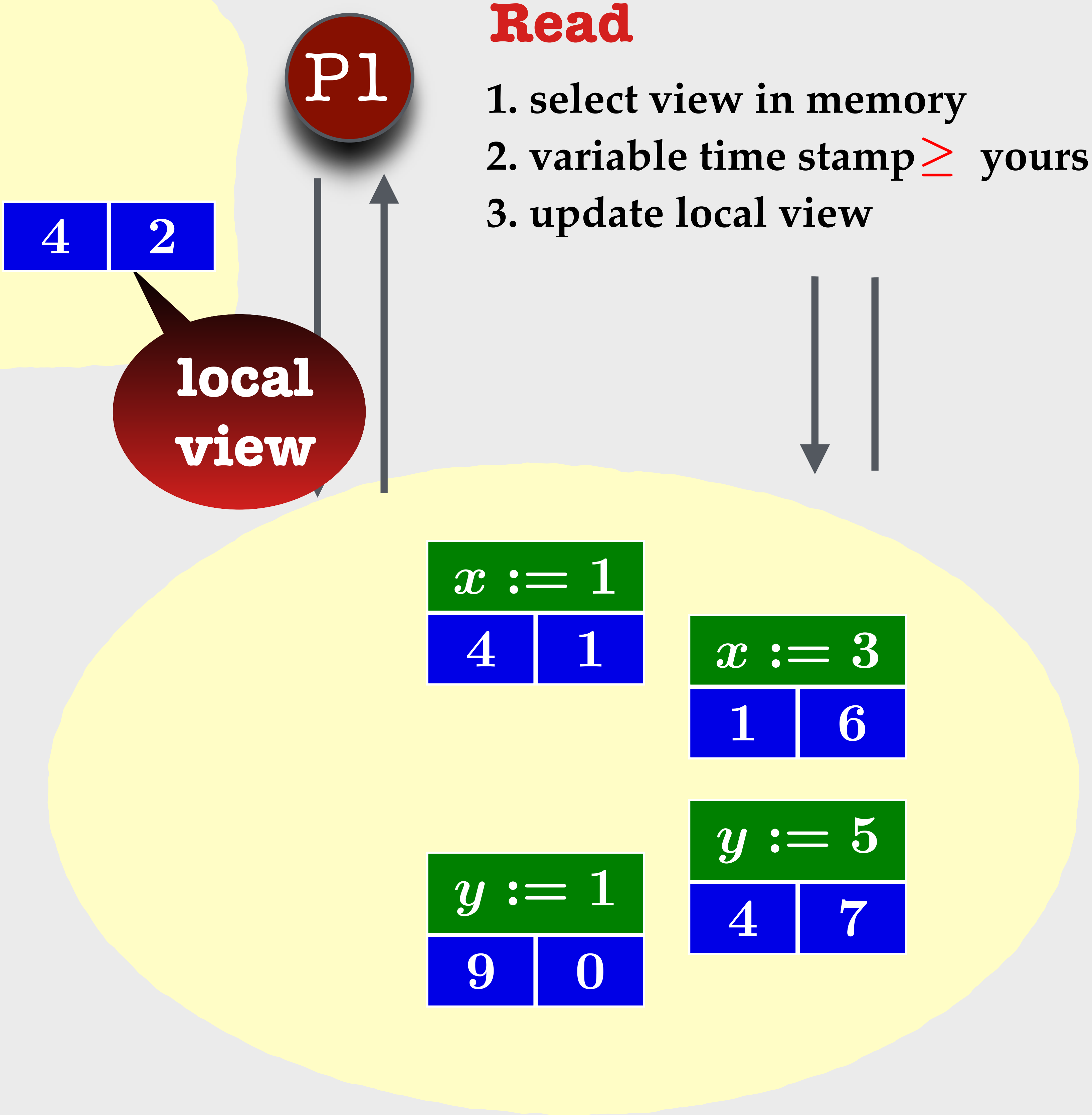
P_1 : $a := 1$

P_1 : $x := 2$

P_1 : $b := y$

Read

1. select view in memory
2. variable time stamp $\geq$ yours
3. update local view

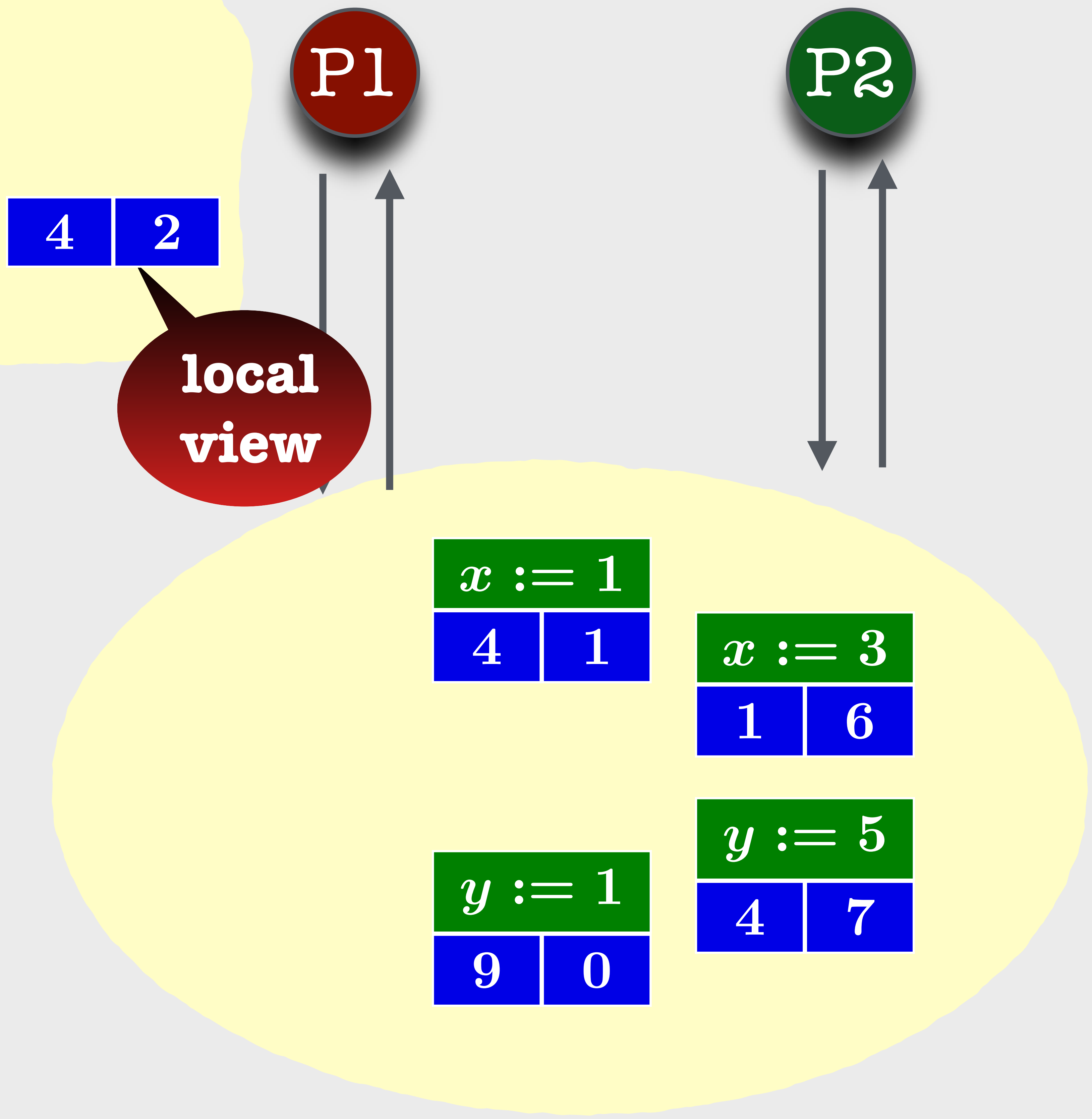


RA: High Level Description

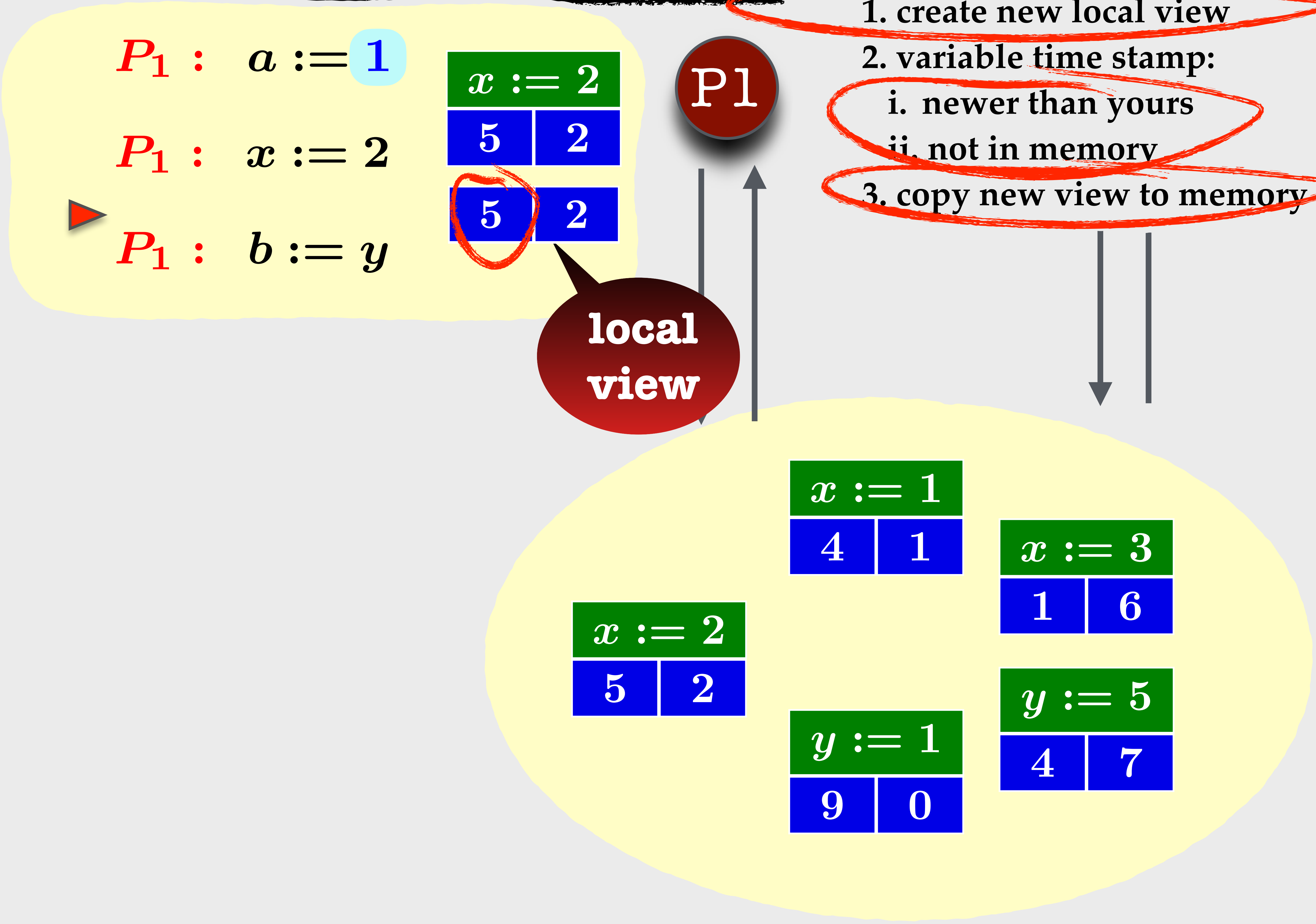
P_1 : $a := 1$

P_1 : $x := 2$

P_1 : $b := y$



RA: High Level Description



Write

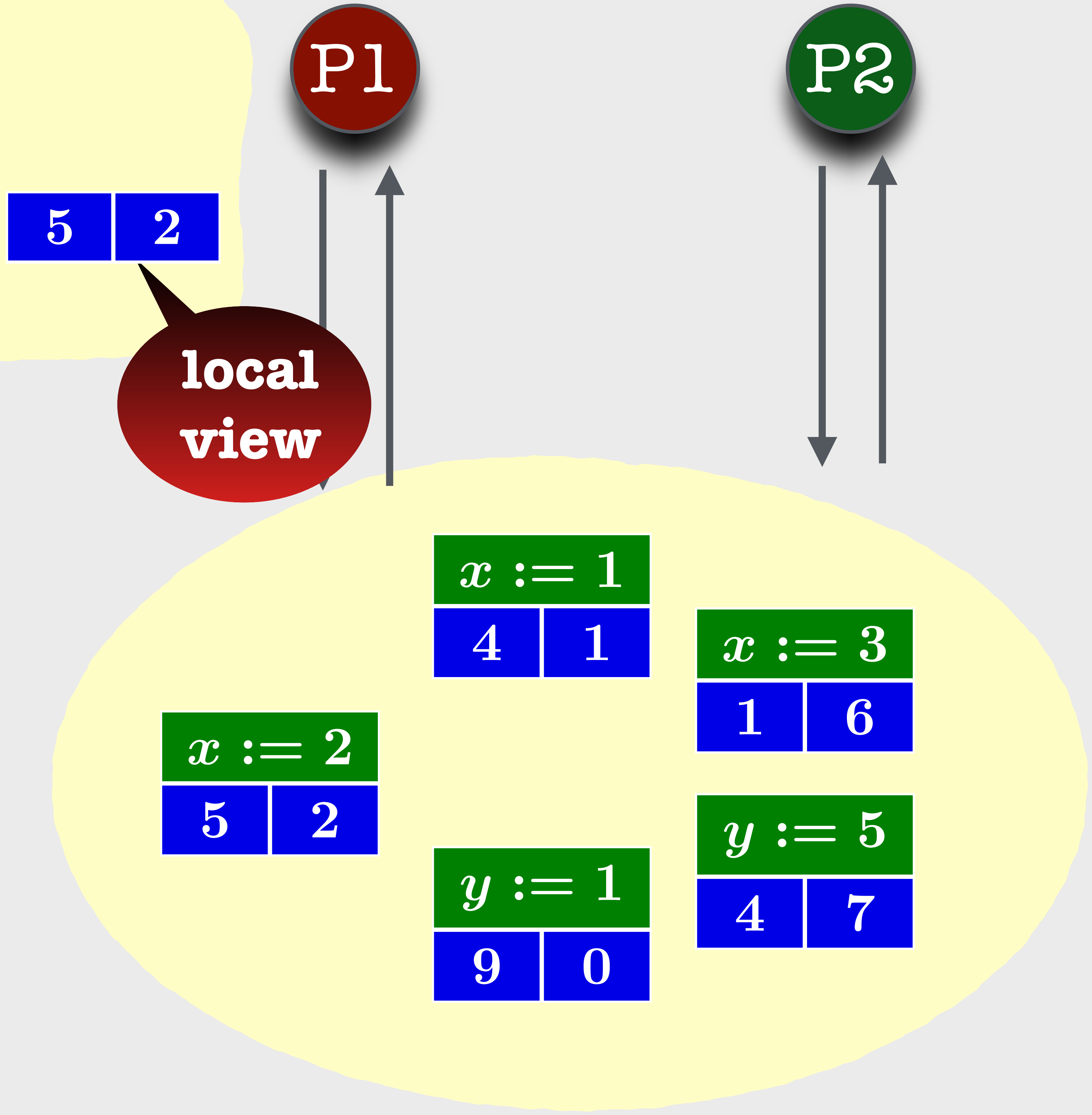
1. create new local view
2. variable time stamp:
   i. newer than yours
   ii. not in memory
3. copy new view to memory

RA: High Level Description

P_1 : $a := \boxed{1}$

P_1 : $x := 2$

P_1 : $b := y$



RA: High Level Description

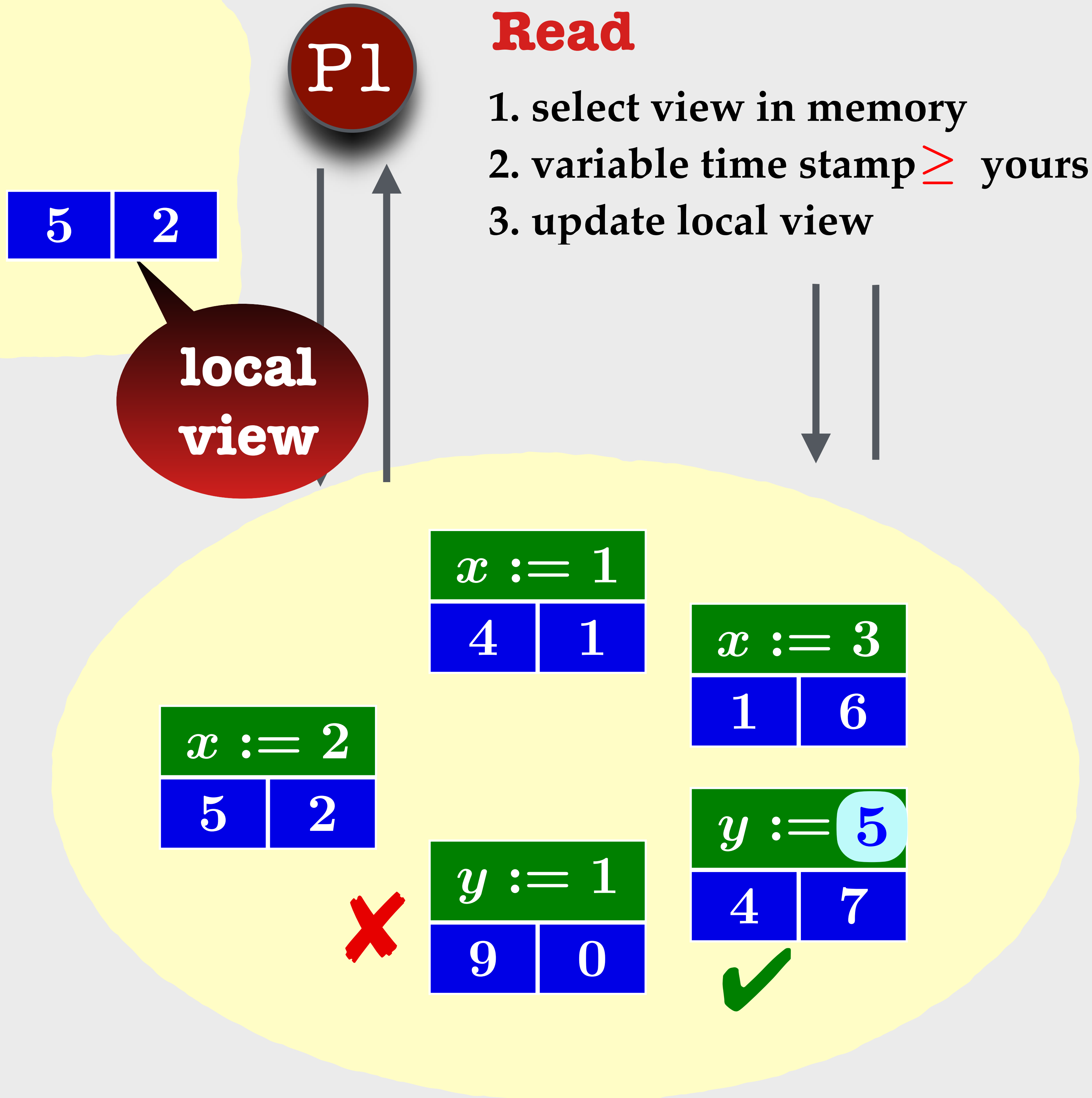
$P_1: \quad a := 1$

$P_1: \quad x := 2$

$P_1: \quad b := y$

Read

1. select view in memory
2. variable time stamp $\geq$ yours
3. update local view



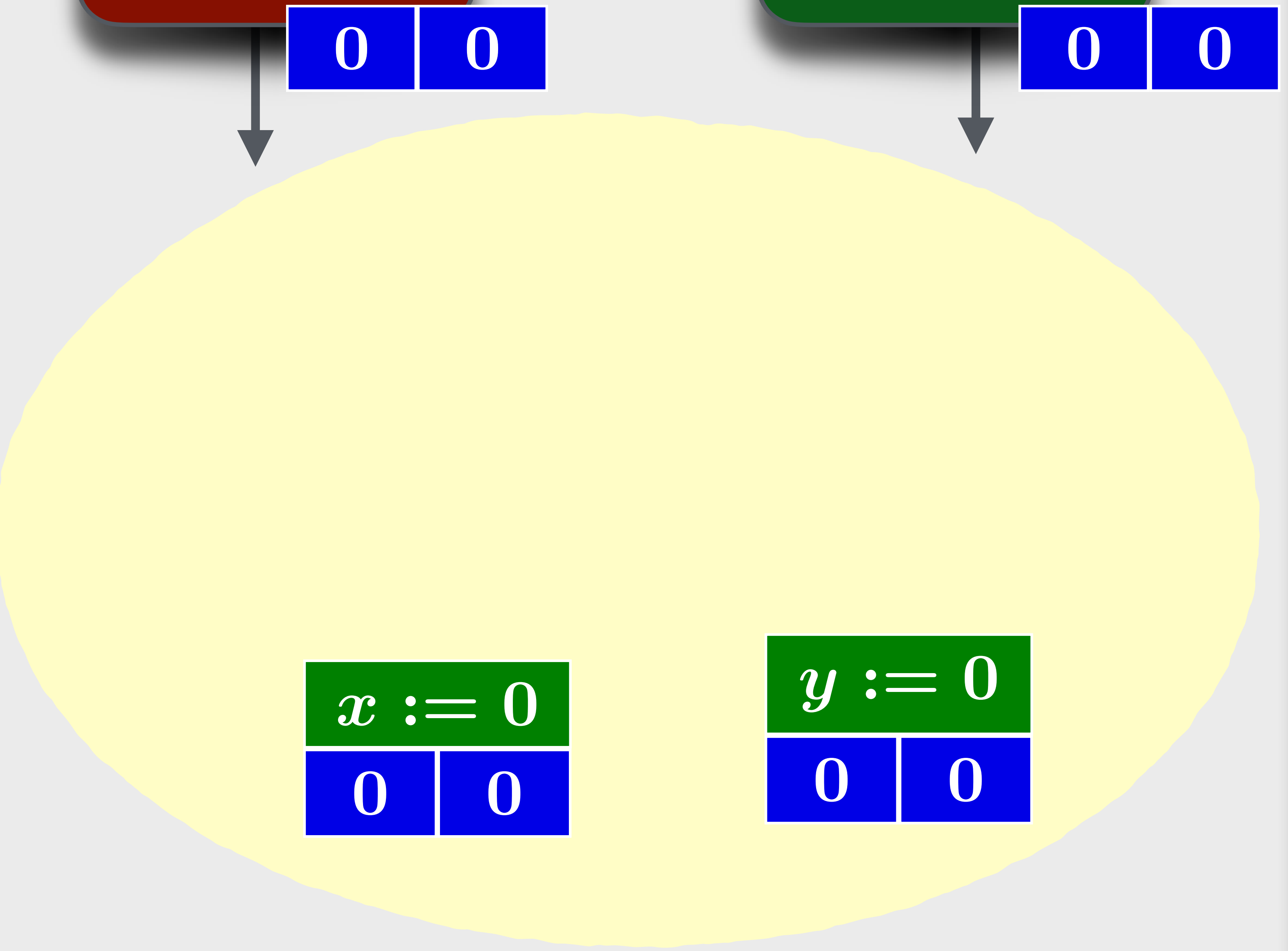
Register values: $r1=0 $r2=0

Process 1

```
1. $r1 = x;
2. y = 1;
3. $r3 = x;
```

Process 2

```
1. $r2 = y;
2. x = 1;
3. x = 2;
```



RA run

Reachable: $r1 =0, $r2 = 1 and $r3=2?

RA run

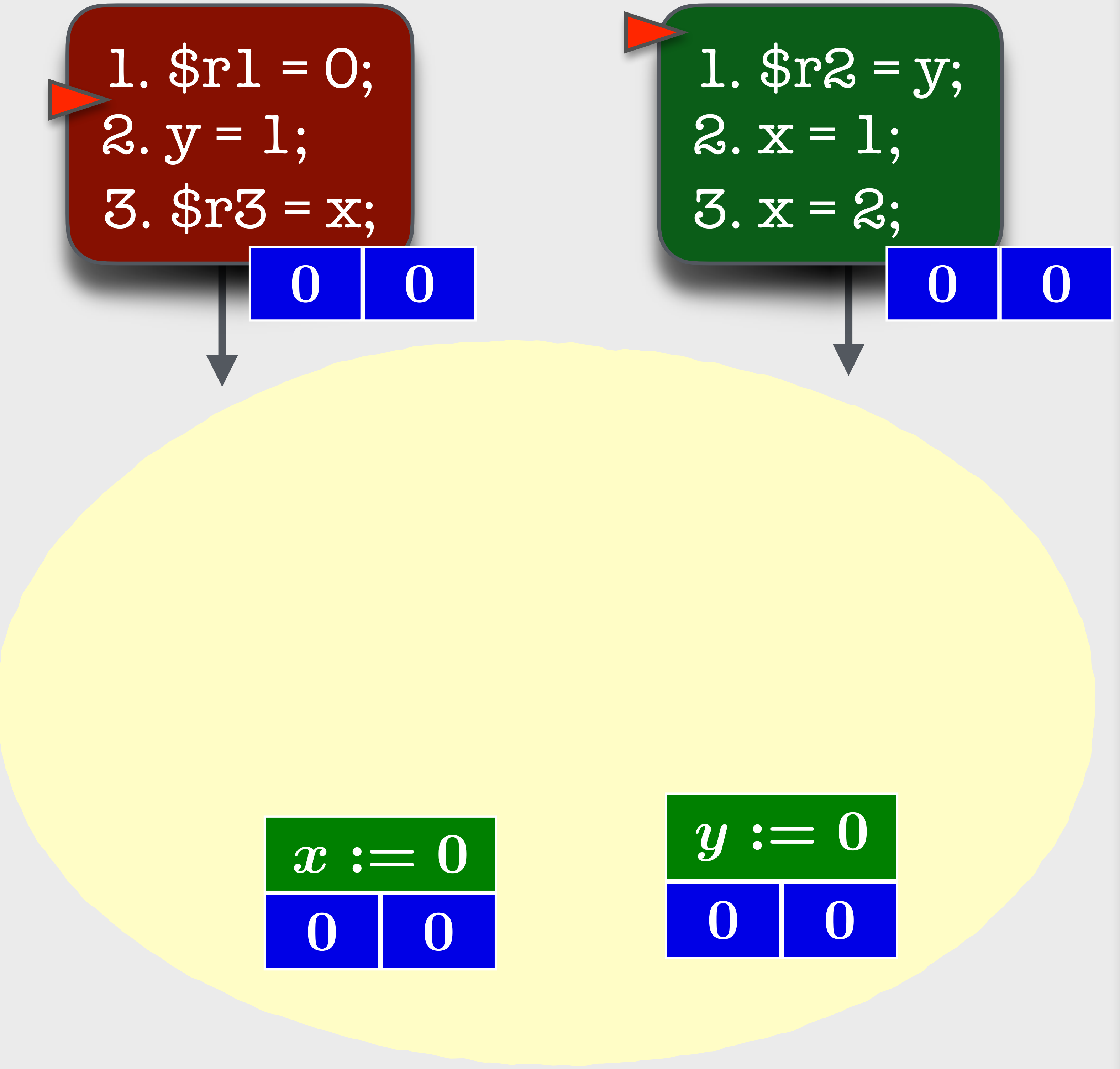
Register values: $r1=0 $r2=0

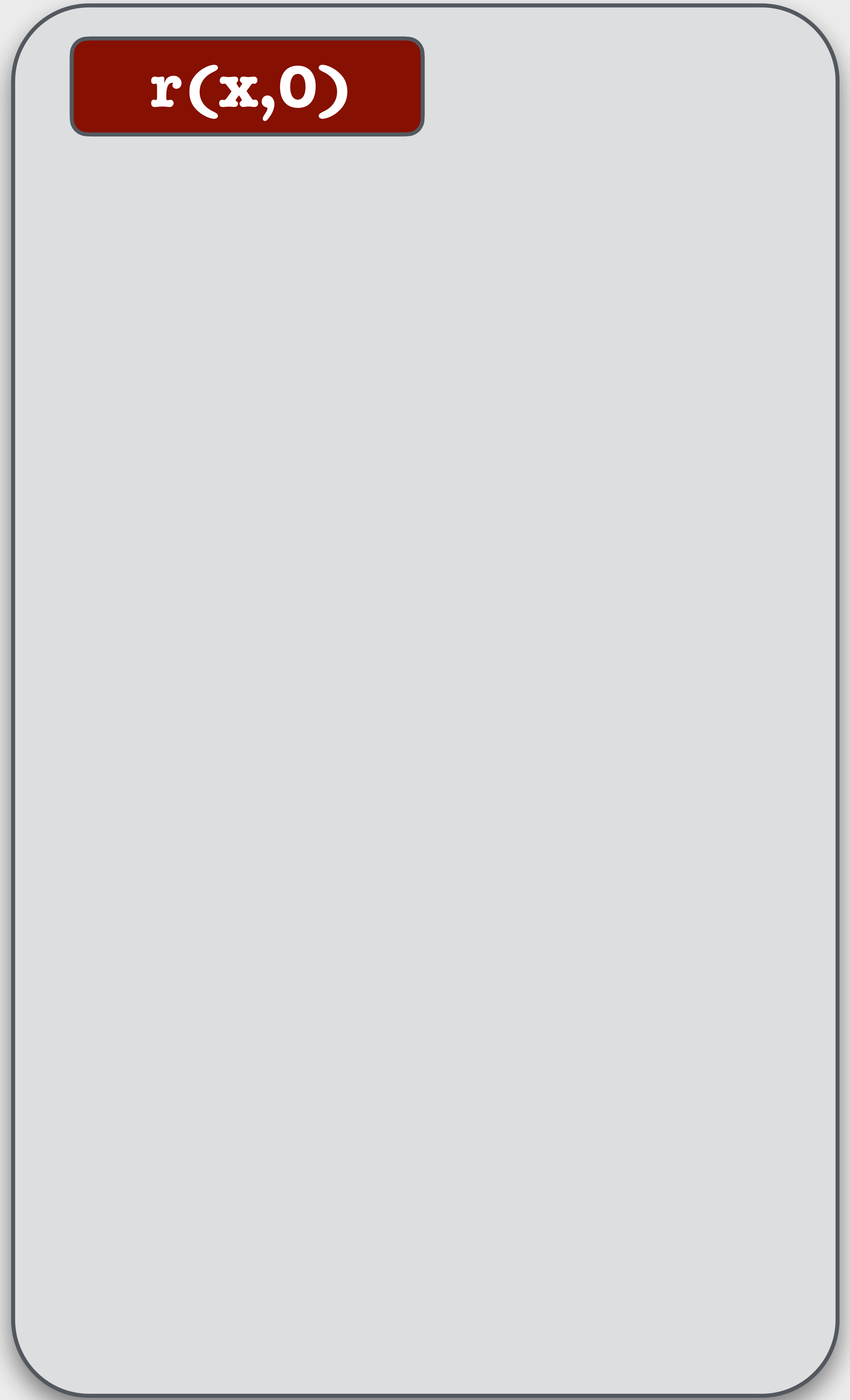
Process 1

```
1. $r1 = 0;
2. y = 1;
3. $r3 = x;
```

Process 2

```
1. $r2 = y;
2. x = 1;
3. x = 2;
```





Reachable: $r1 =0, $r2 = 1 and $r3=2?

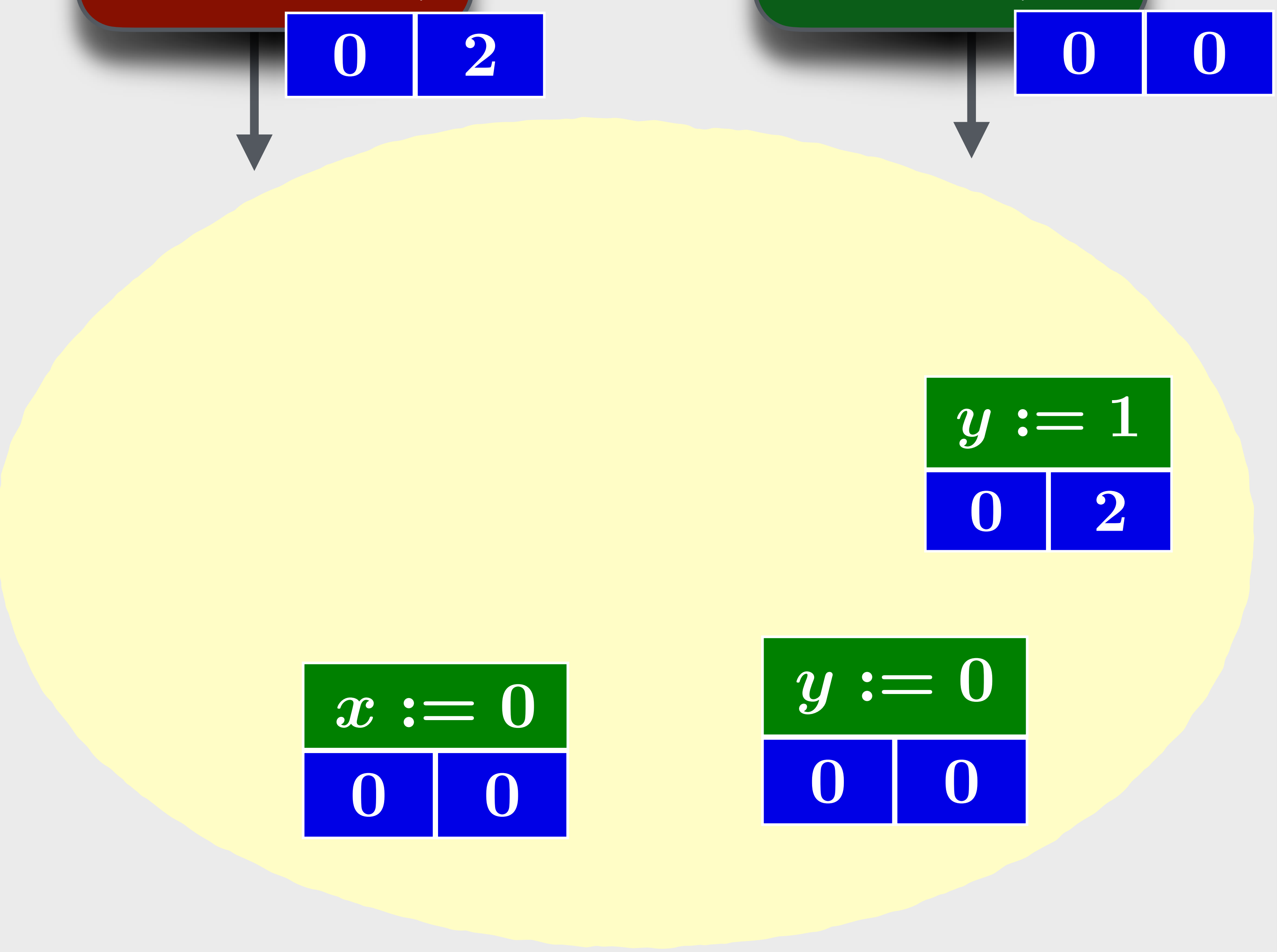
Register values: $r1=0 $r2=0

Process 1

1. $r1 = 0;
2. y = 1;
3. $r3 = x;

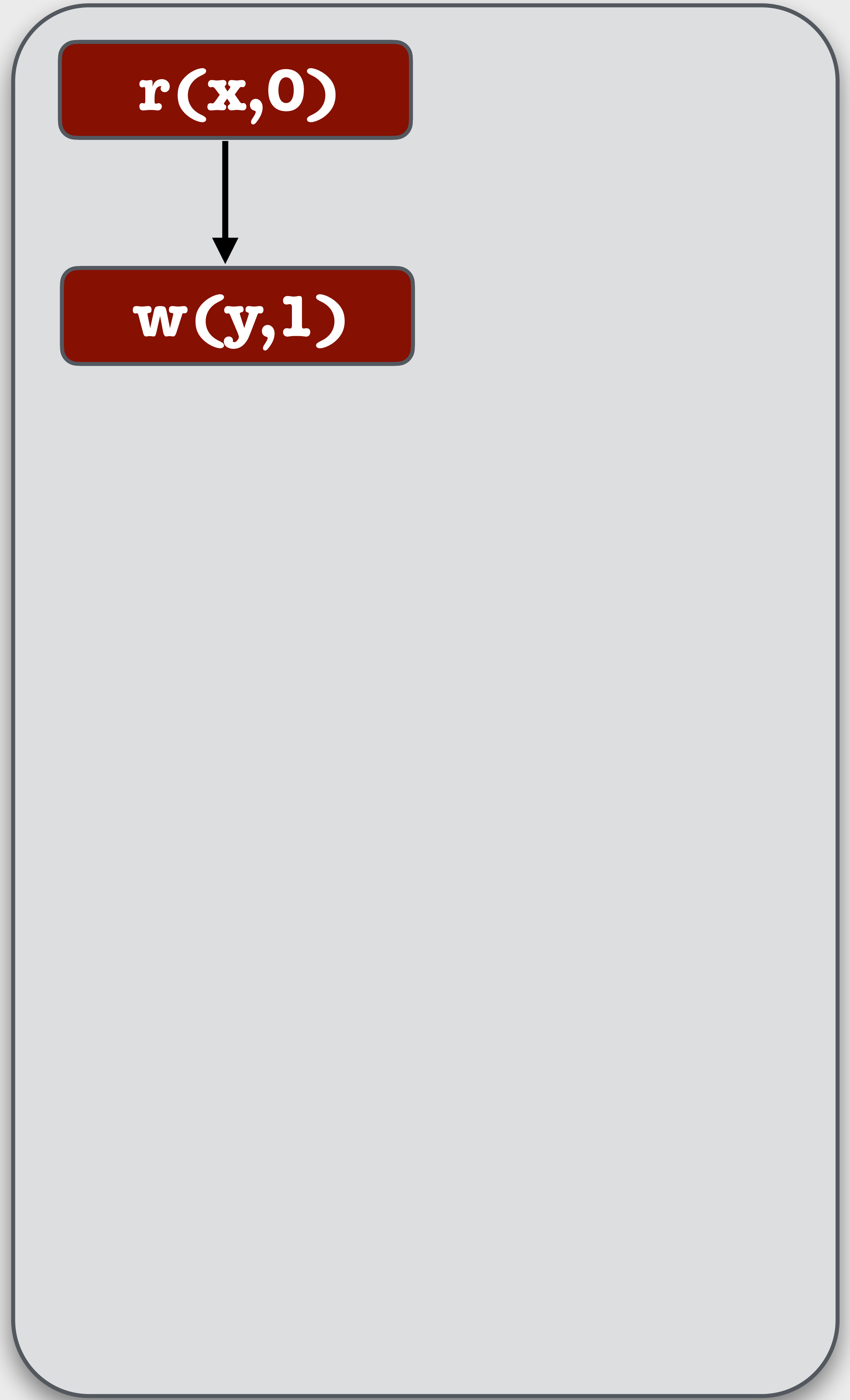
Process 2

1. $r2 = y;
2. x = 1;
3. x = 2;

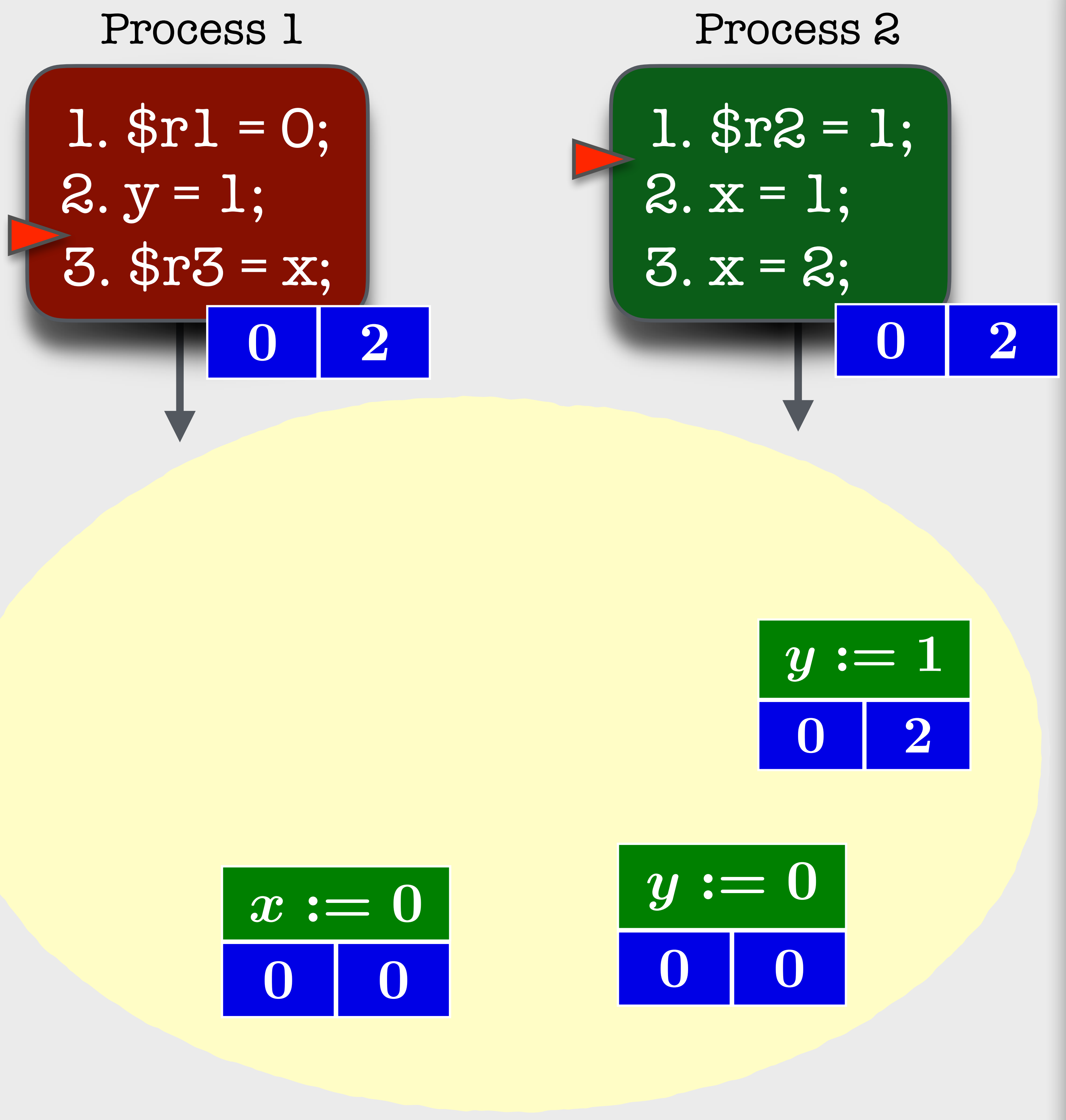


Reachable: $r1 =0, $r2 = 1 and $r3=2?

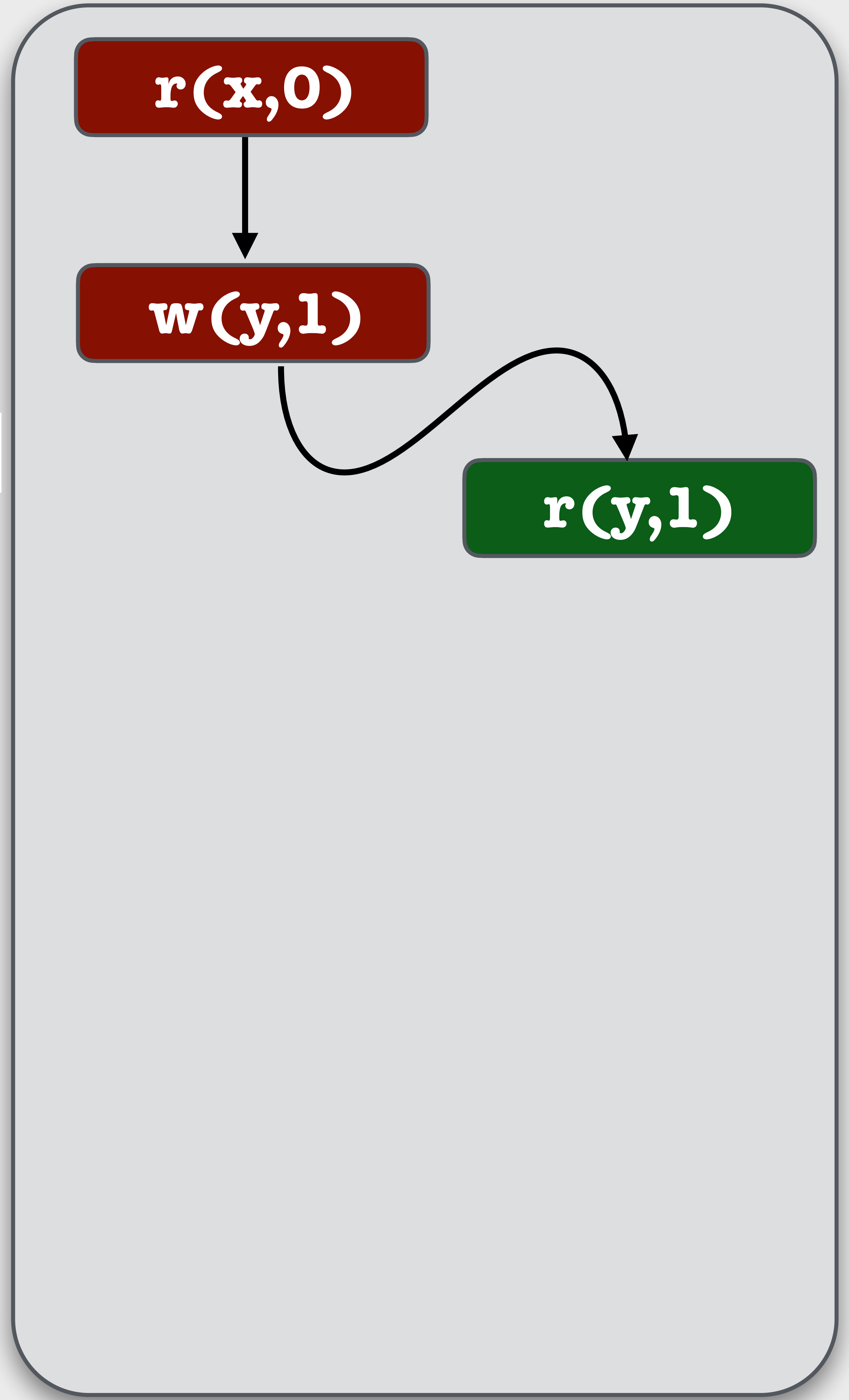
RA run



Register values: $r1=0 $r2=0



RA run



Reachable: $r1 =0, $r2 = 1 and $r3=2?

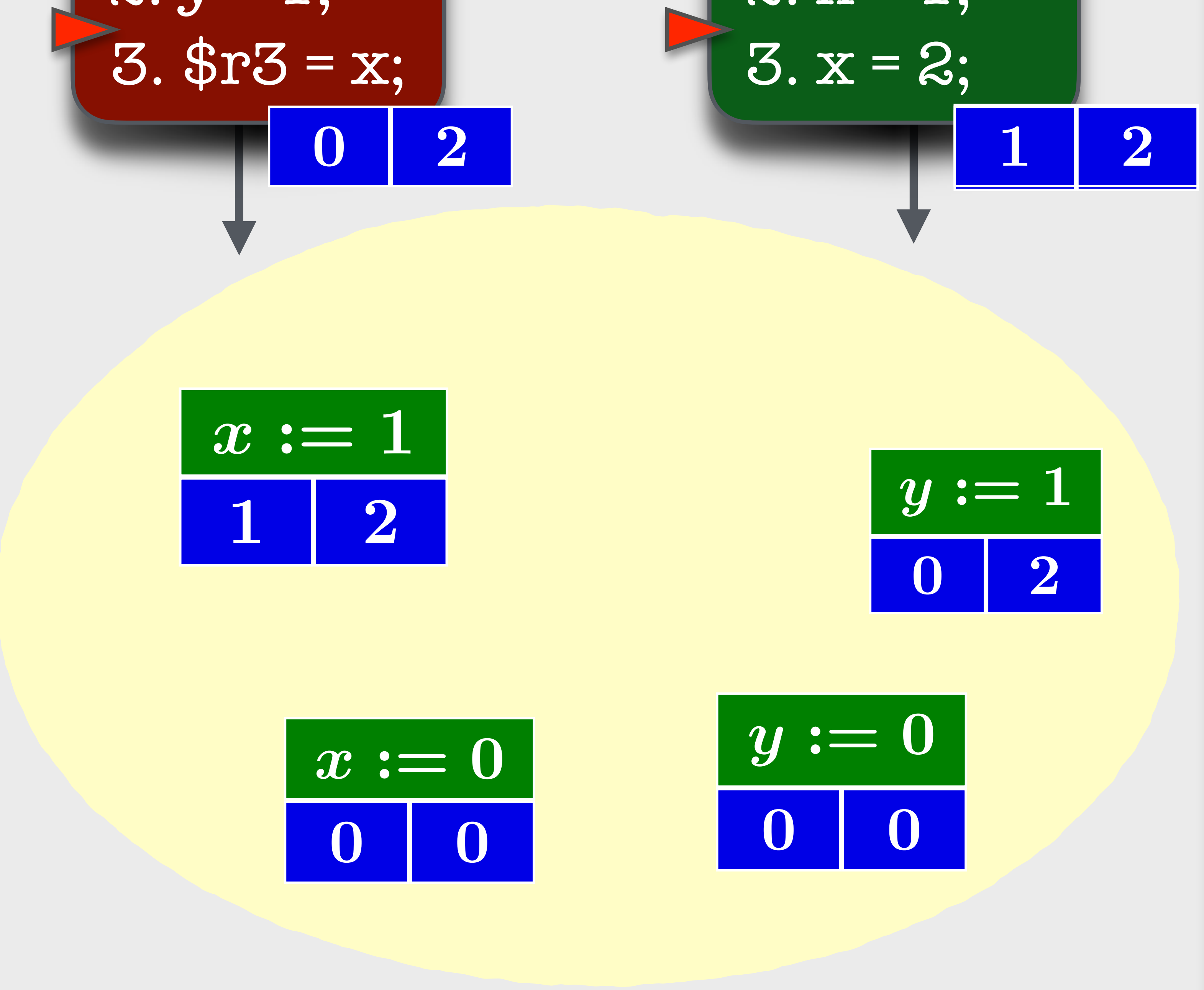
Register values: \$r1=0 \$r2=0

Process 1

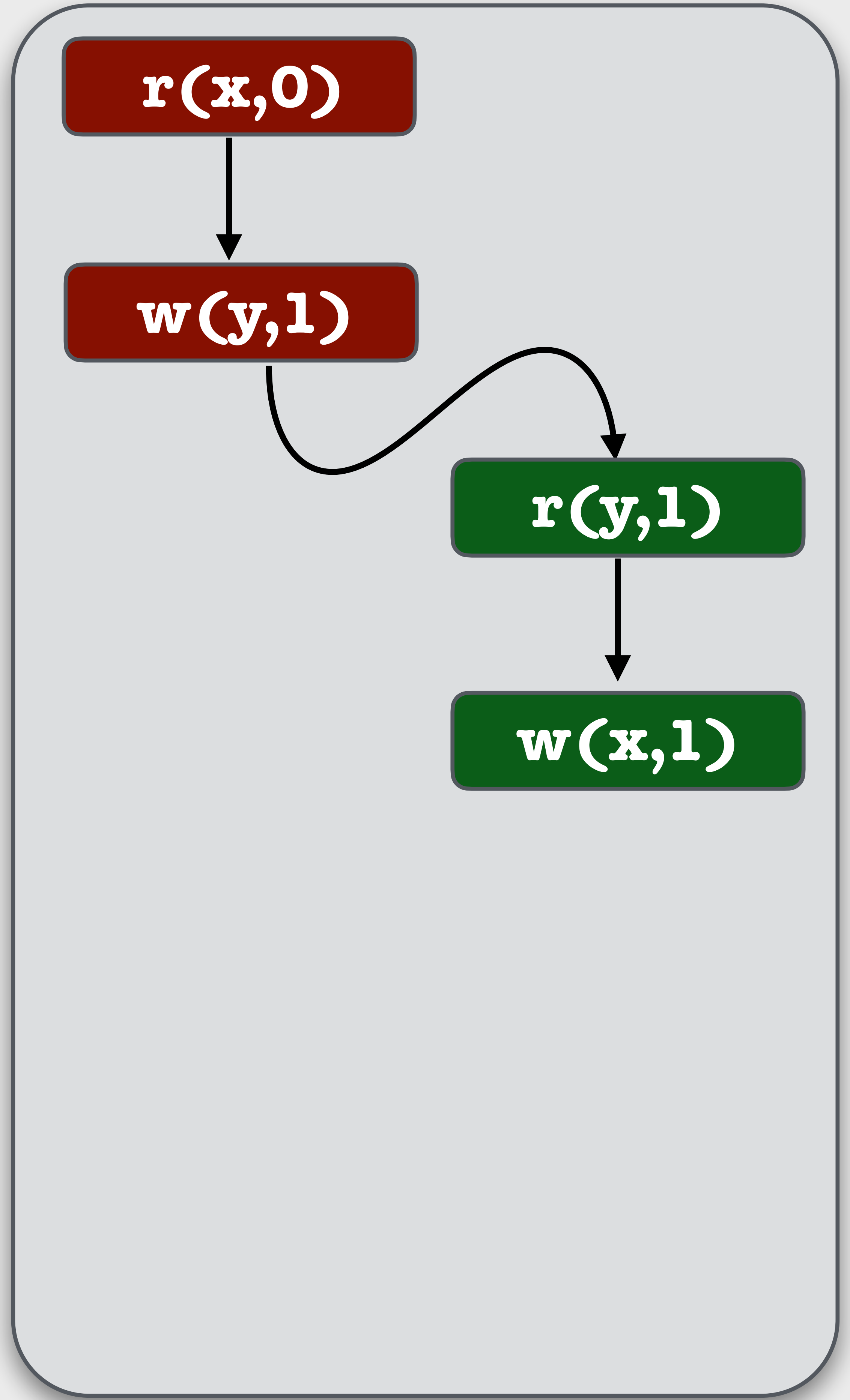
```
1. $r1 = 0;
2. y = 1;
3. $r3 = x;
```

Process 2

```
1. $r2 = 1;
2. x = 1;
3. x = 2;
```

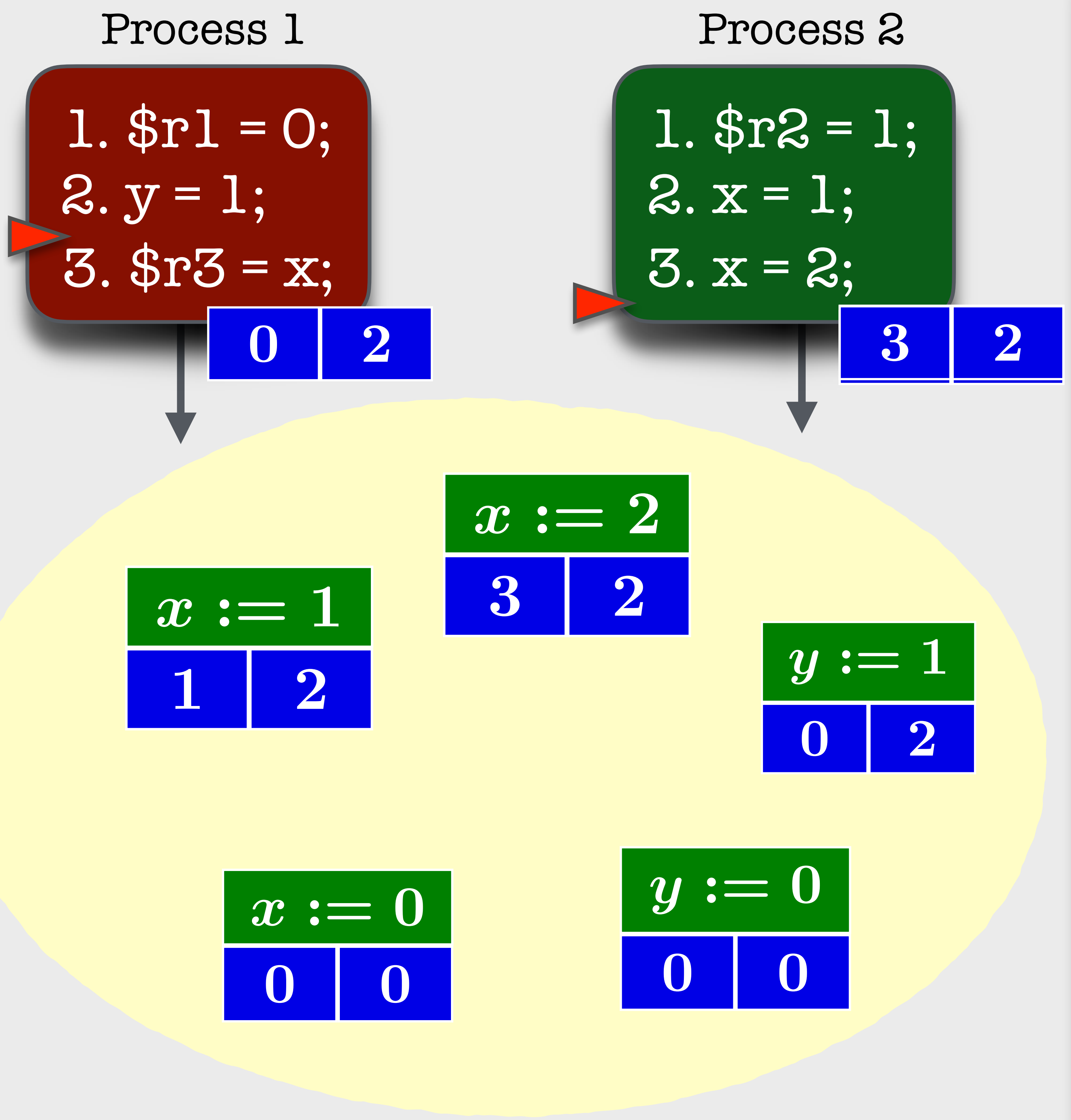


RA run



Reachable: \$r1 =0, \$r2 = 1 and \$r3=2?

Register values: $r1=0 $r2=0



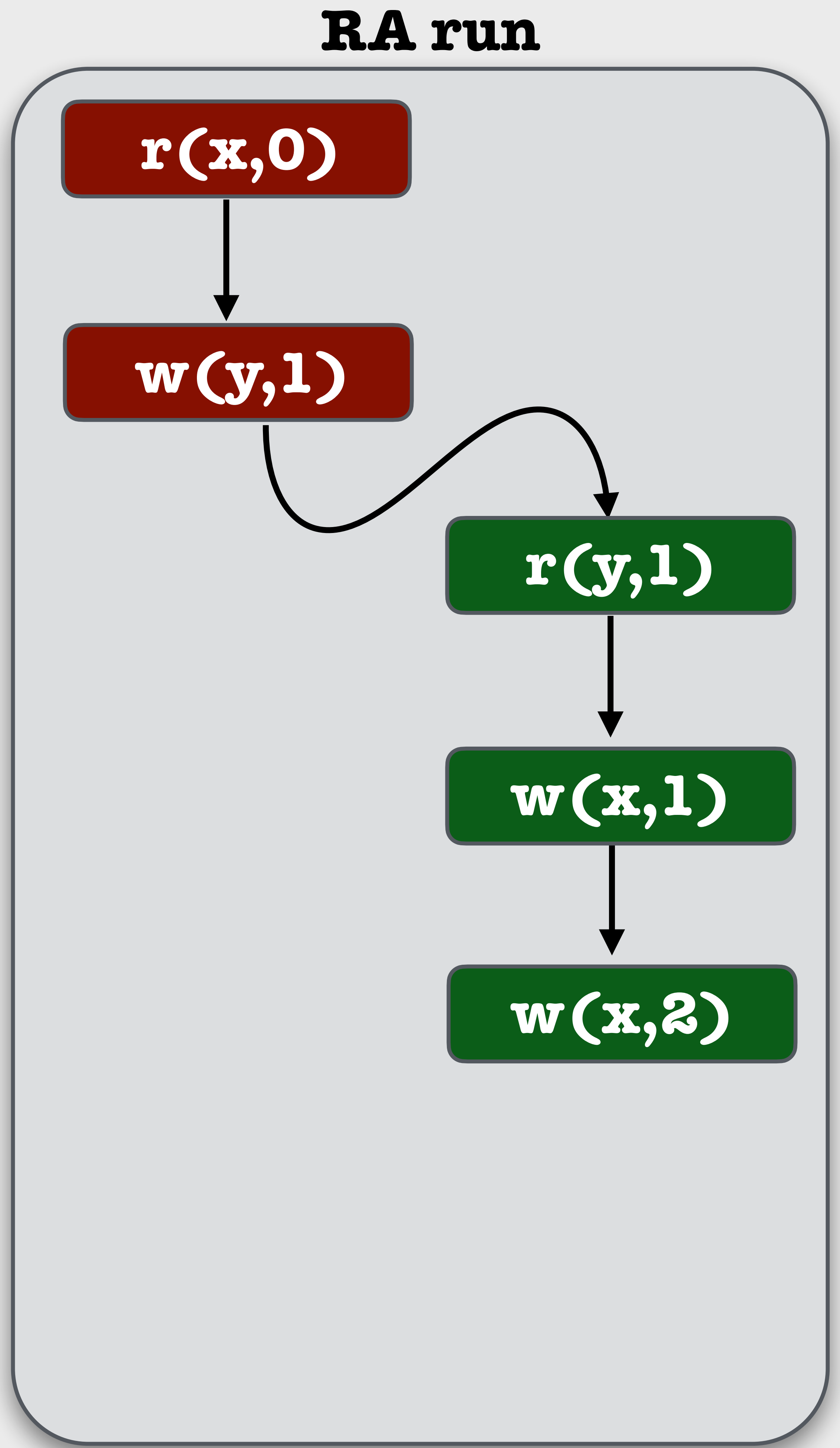
Process 1

```
1. $r1 = 0;
2. y = 1;
3. $r3 = x;
```

Process 2

```
1. $r2 = 1;
2. x = 1;
3. x = 2;
```

Reachable: $r1 =0, $r2 = 1 and $r3=2?



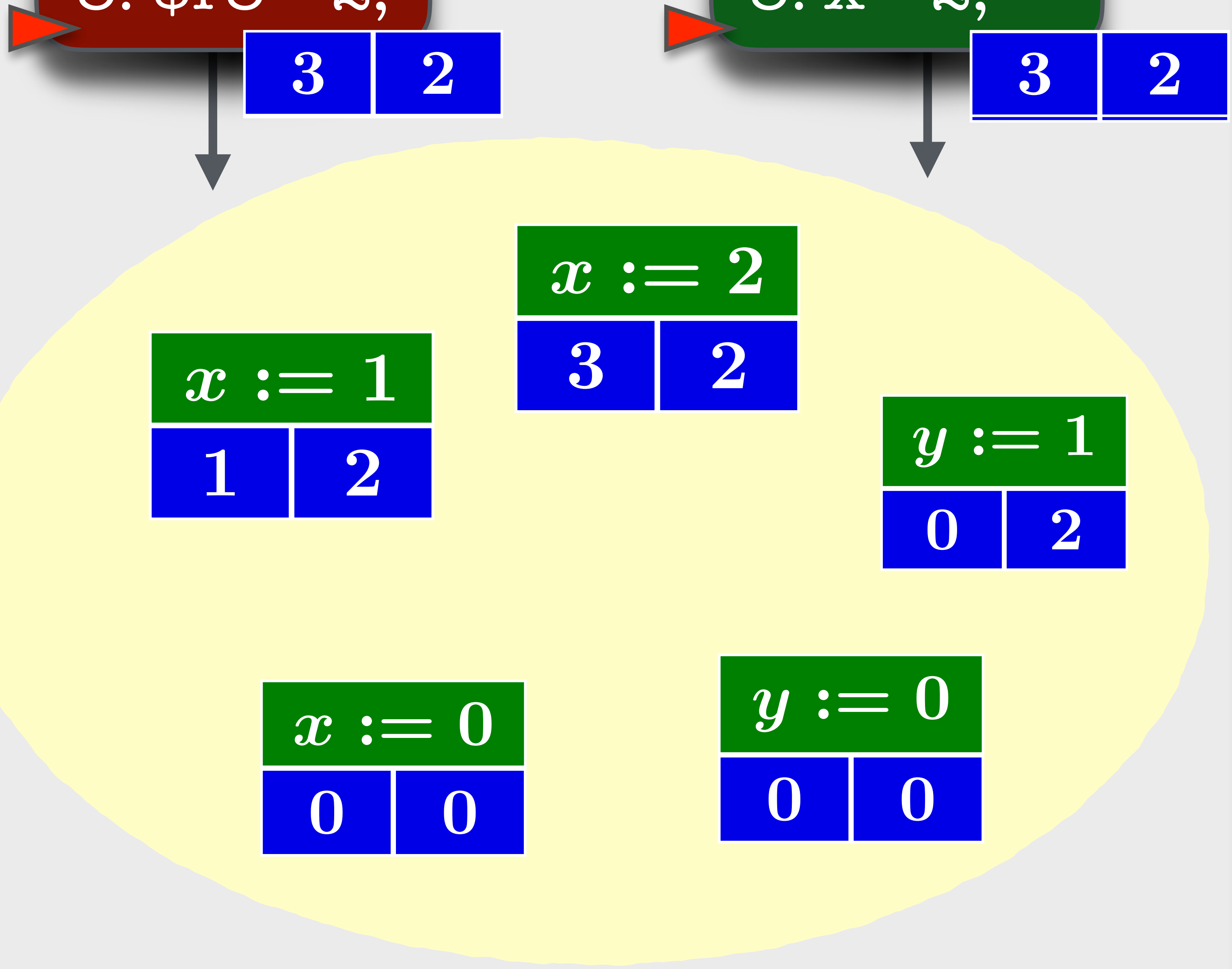
Register values: \$r1=0 \$r2=0

Process 1

```
1. $r1 = 0;
2. y = 1;
3. $r3 = 2;
```

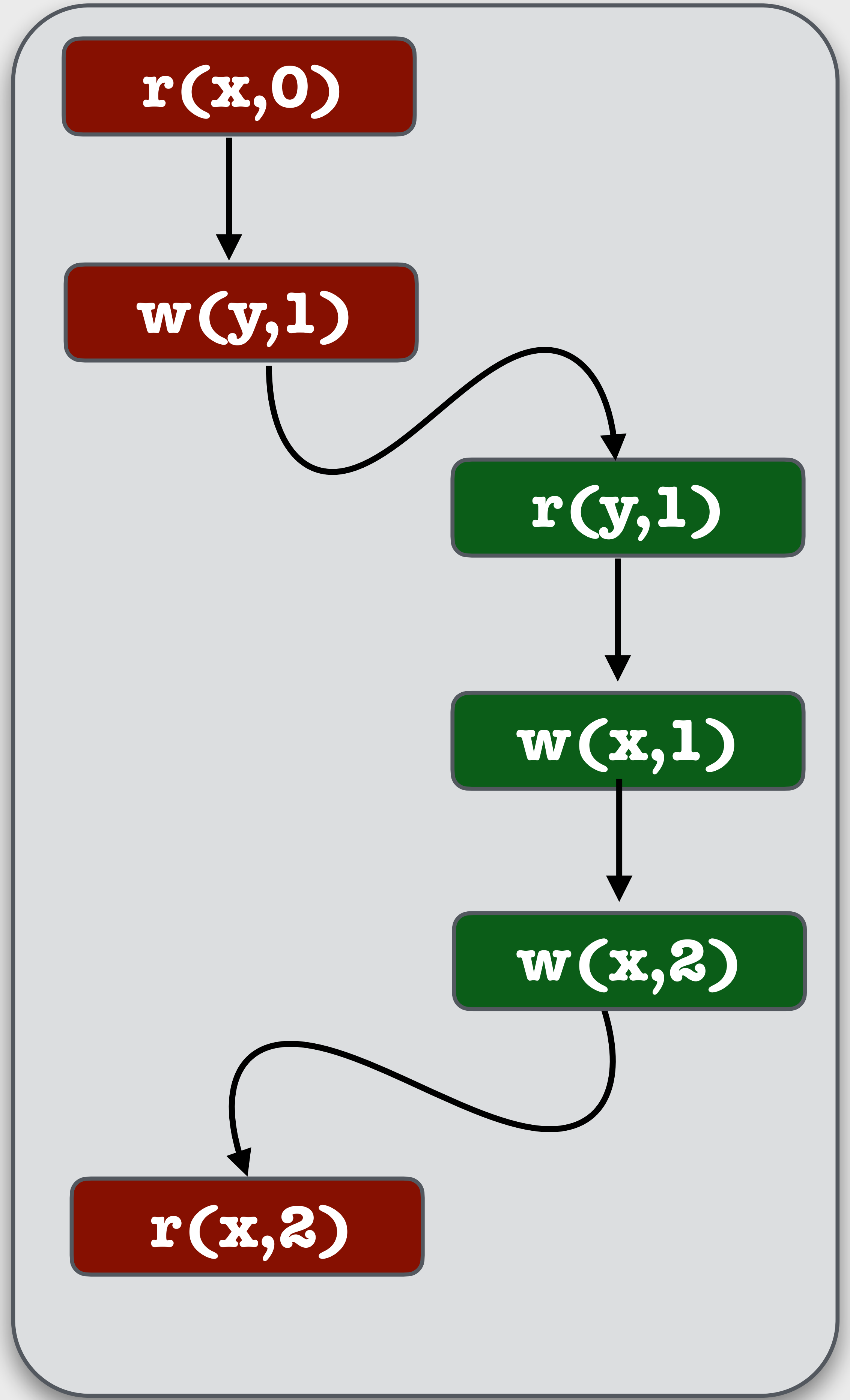
Process 2

```
1. $r2 = 1;
2. x = 1;
3. x = 2;
```

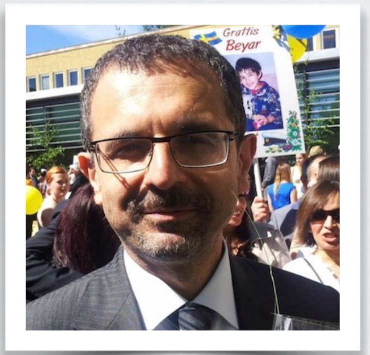


Reachable: \$r1 =0, \$r2 = 1 and \$r3=2?

RA run



(Non parameterized) Reachability under RA



PLDI 2019

Given a **program P** and a (control + memory) **state s**

- State Reachability Problem (Safety)

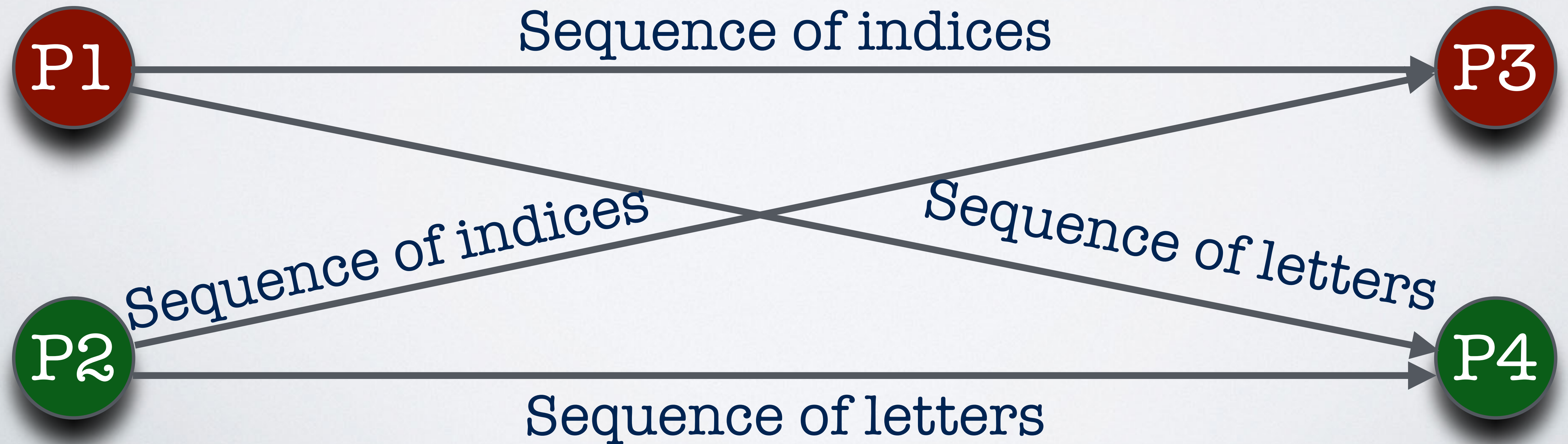
  Is **s** reachable in **P**?

The state reachability problem is undecidable for RA

Proof Idea:

Possible workaround?

- By reduction from the Post's correspondence Problem



Context-bounded Analysis (CBA)

- ✦ Efficient under-approximation technique for SC [Qadeer et al. 2005, Lal et al. 2009, Torre et al. 2009]
  - Several tools: CHESS, Corral, CSeq, etc.

The state reachability problem is still undecidable for **RA**
with a bounded number (3) of context switches
(context: only one "**active**" process)
P1 runs; P2 runs; P3 runs; P4 runs

Context-bounded Analysis (CBA)

- Efficient under-approximation technique for SC [Qadeer et al. 2005, Lal et al. 2009, Torre et al. 2009]
    - Several too

Need a different under approximation for RA

The state reachability problem is still undecidable for **RA**

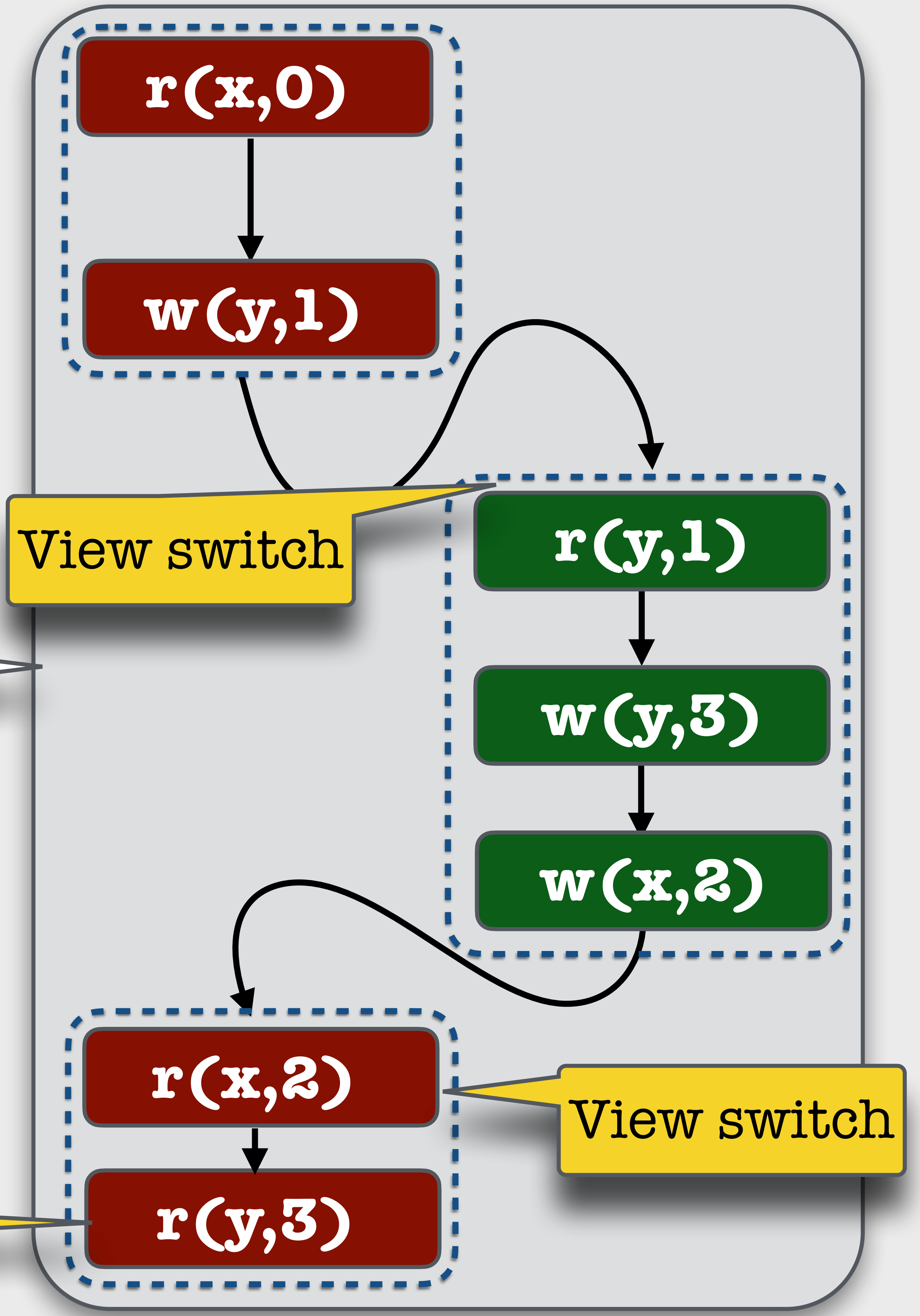
with a bounded number (3) of context switches

(context: only one "**active**" process)

P1 runs; P2 runs; P3 runs; P4 runs

View Switch

A view-switch happens when a process reads a value written by another process, and changes its view

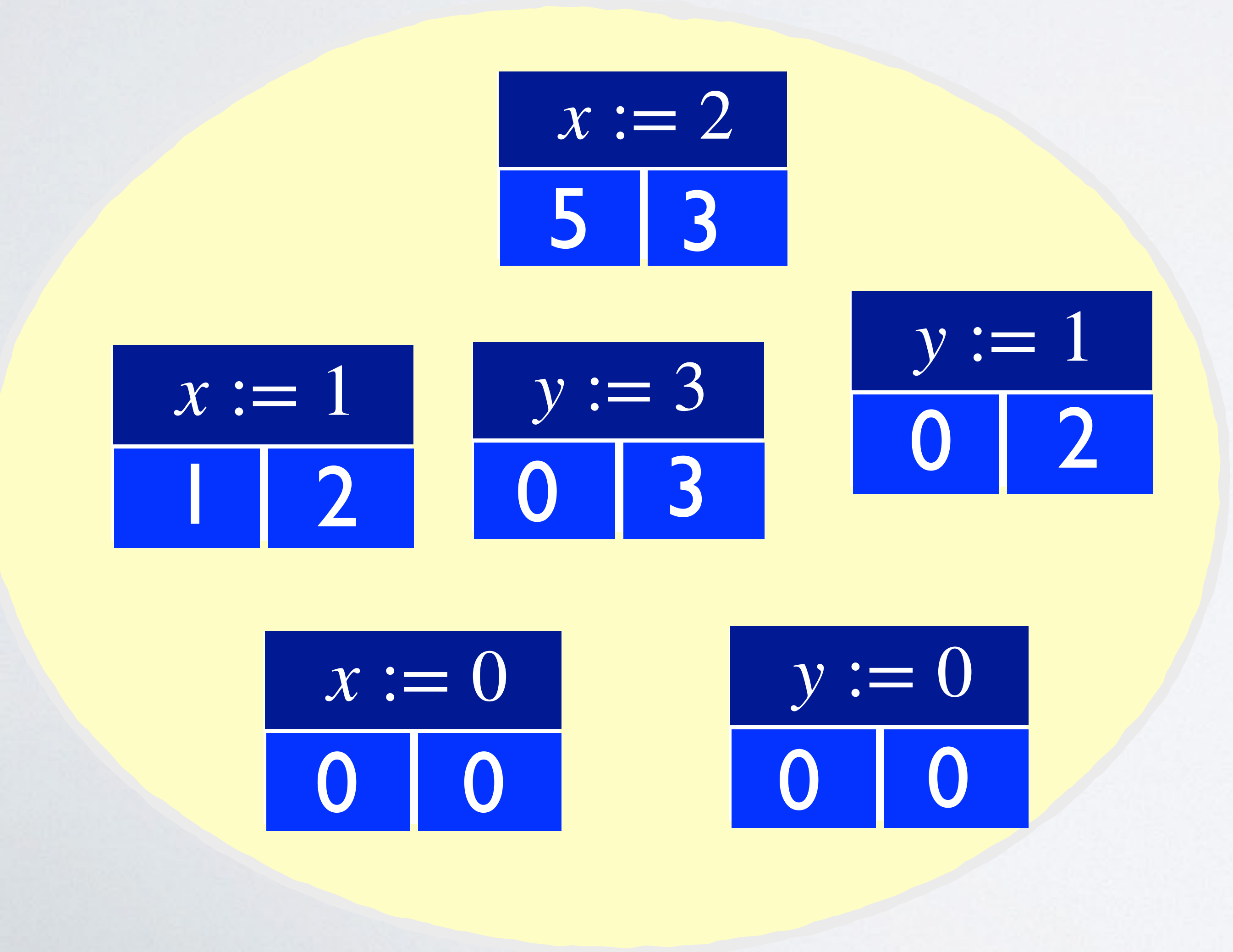


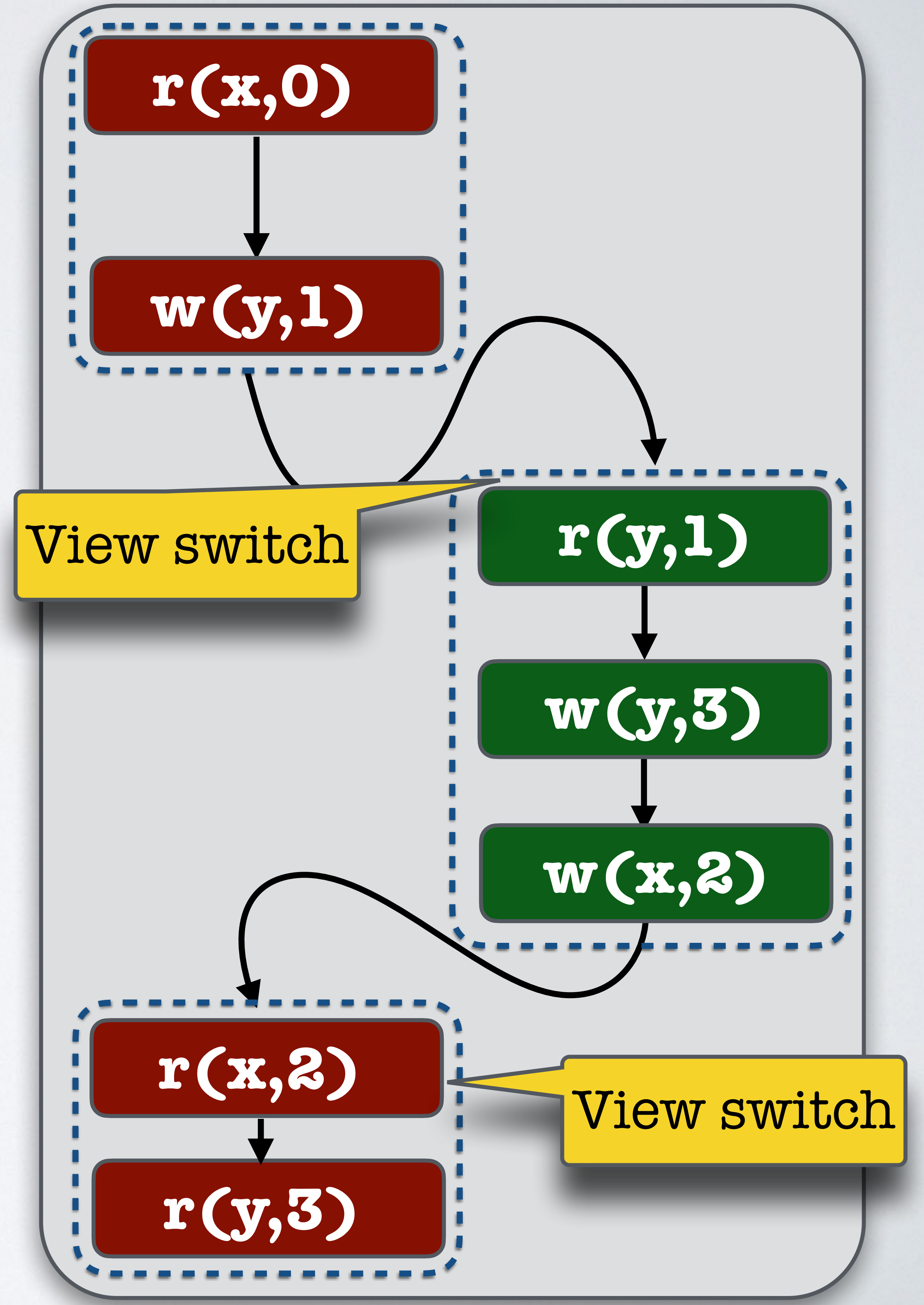
2-bounded run

No View switch

A view-switch happens when a process reads a value written by another process, and changes its view

Bounding the number of **essential** views in the memory

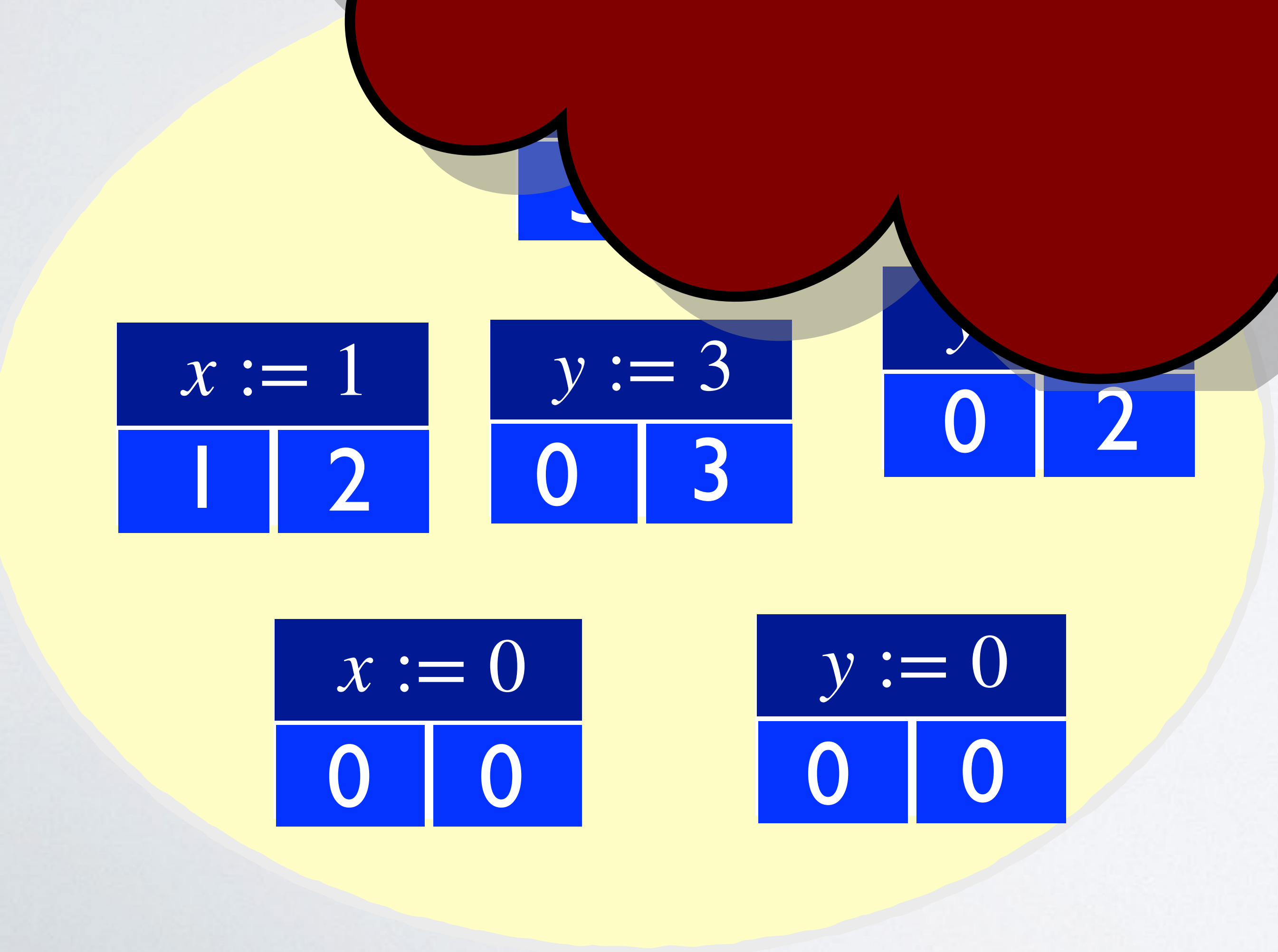


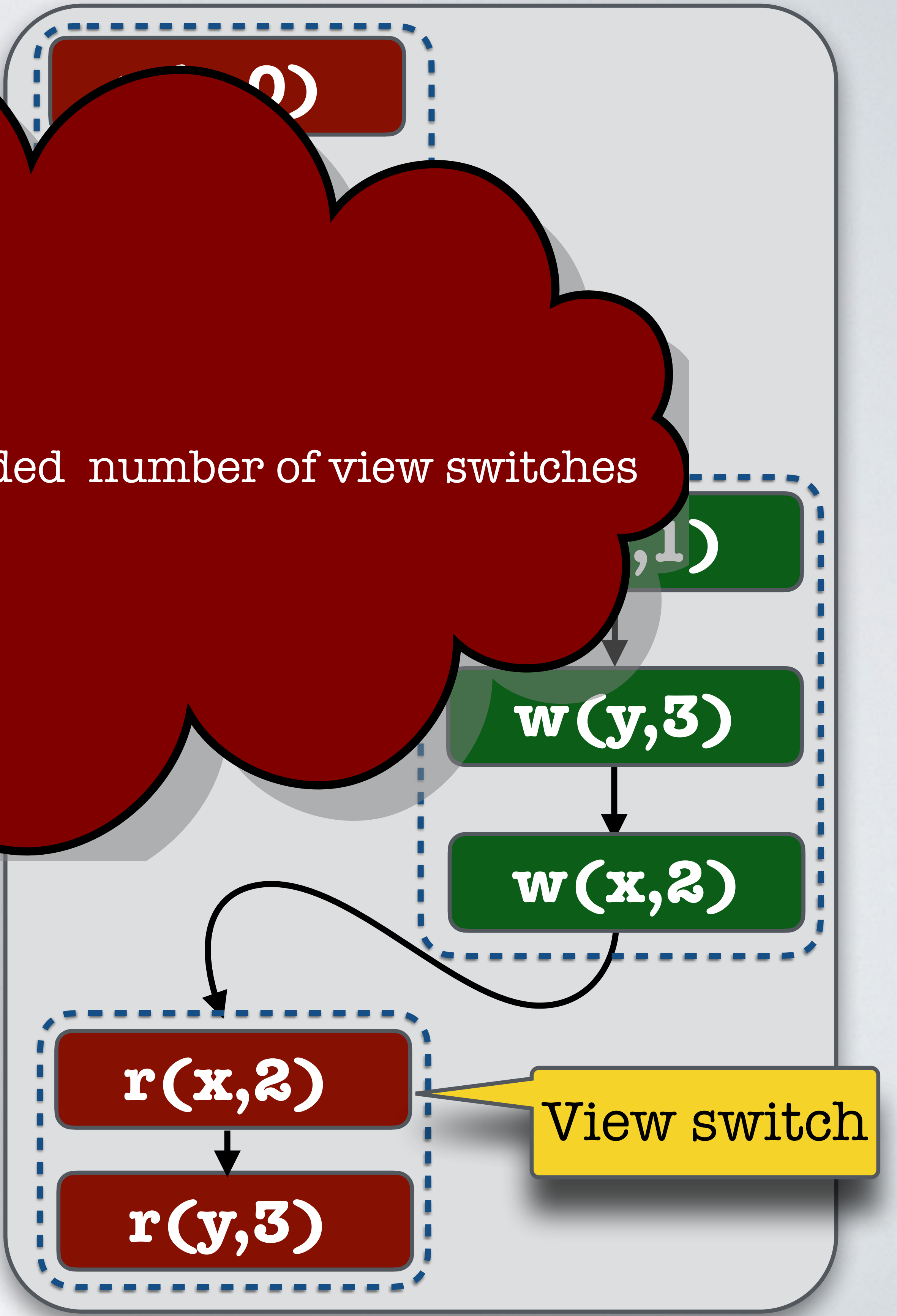


A view-switch happens when a process reads a value written by another process, and changes its view

Boundi
essential

The undecidability proof needed unbounded number of view switches





K-bounded Reachability Problem

Definition

Reachability problem restricted to K-bounded runs

Code-to-code translation

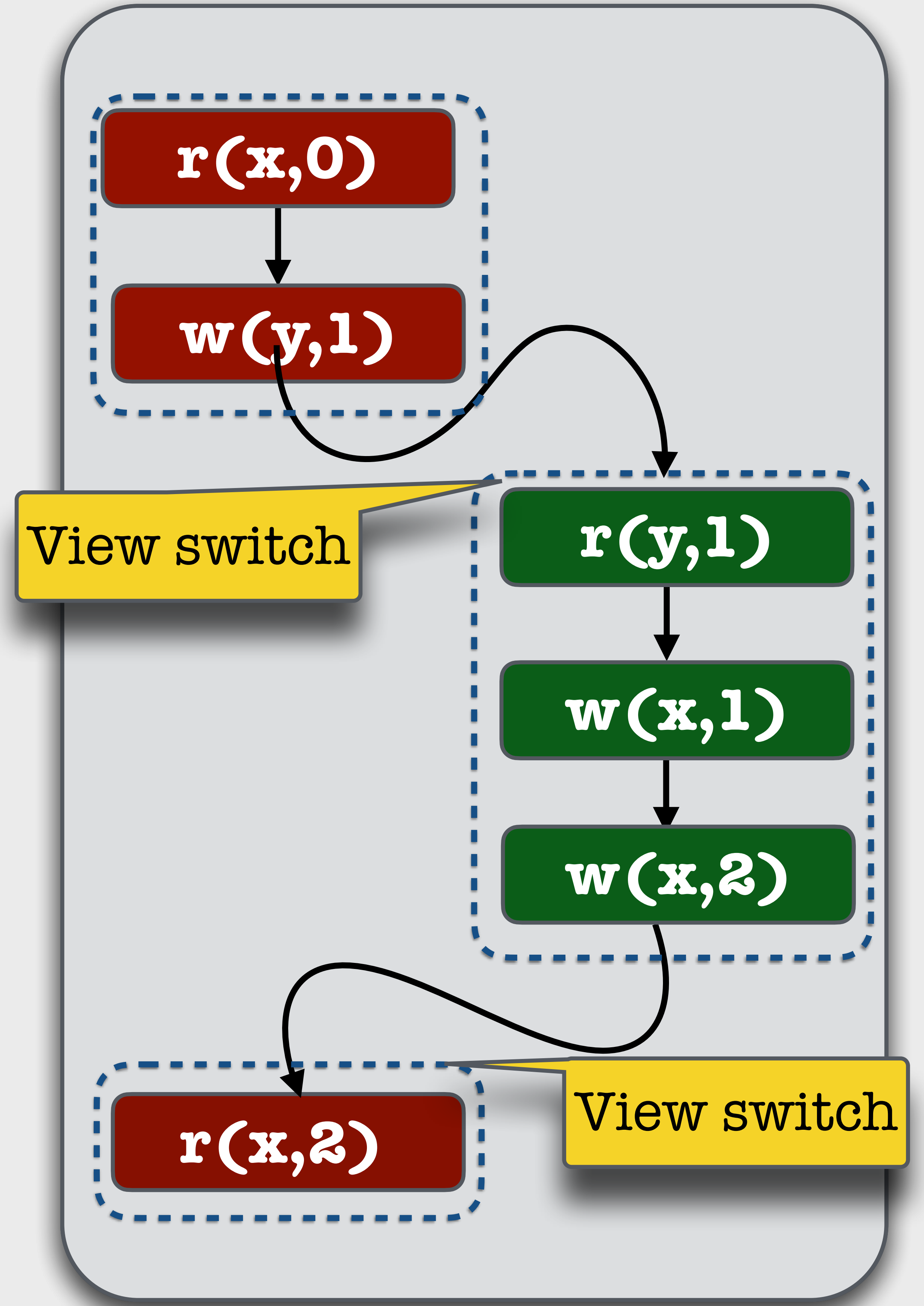
Theorem

The K-bounded reachability for RA is reducible to K+n bounded context reachability under SC

Corollary

The K-bounded reachability for RA is decidable for finite-state programs

Key Ideas



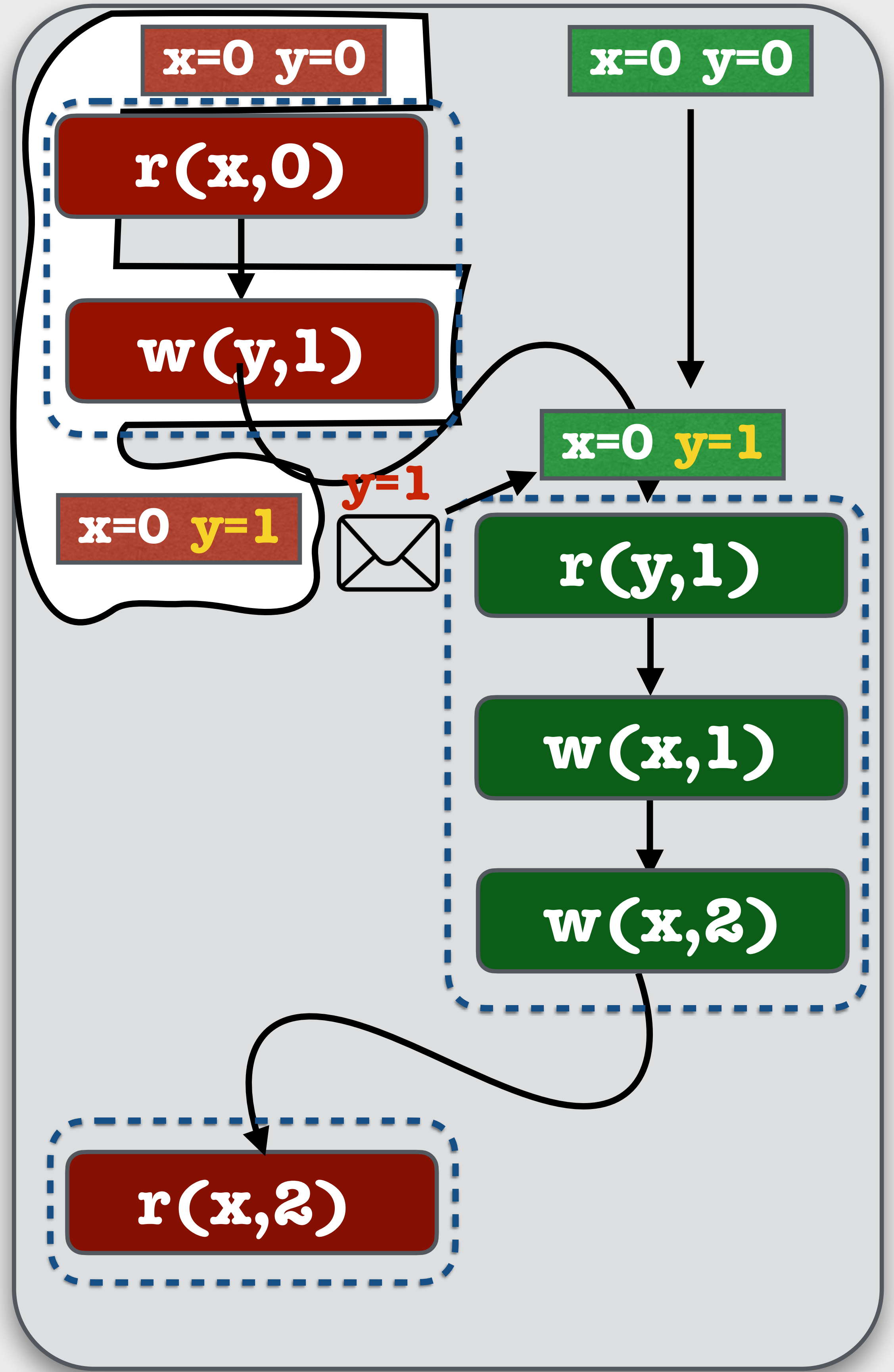
Two steps:

Locality

Validation

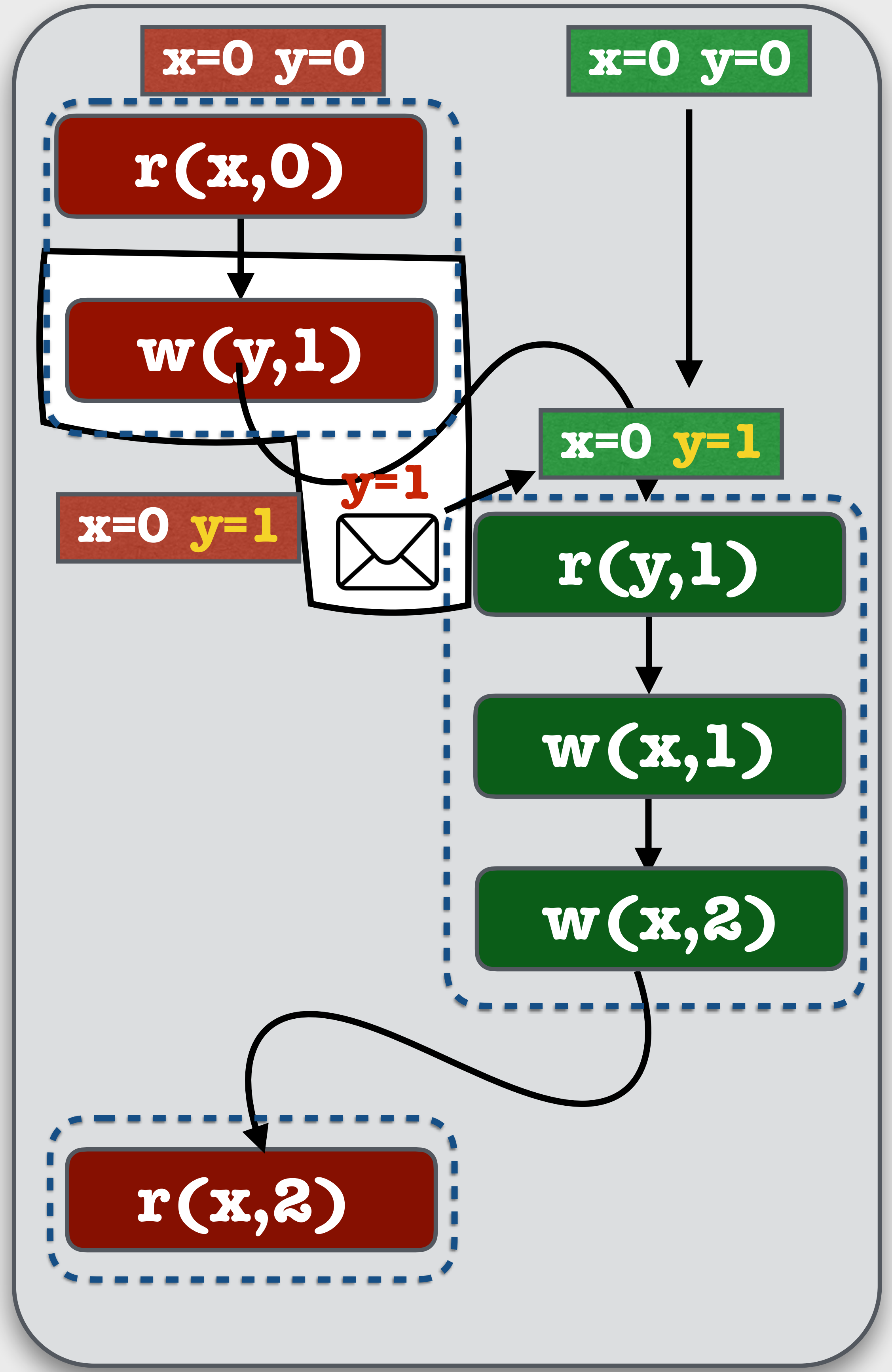
Locality

Validation

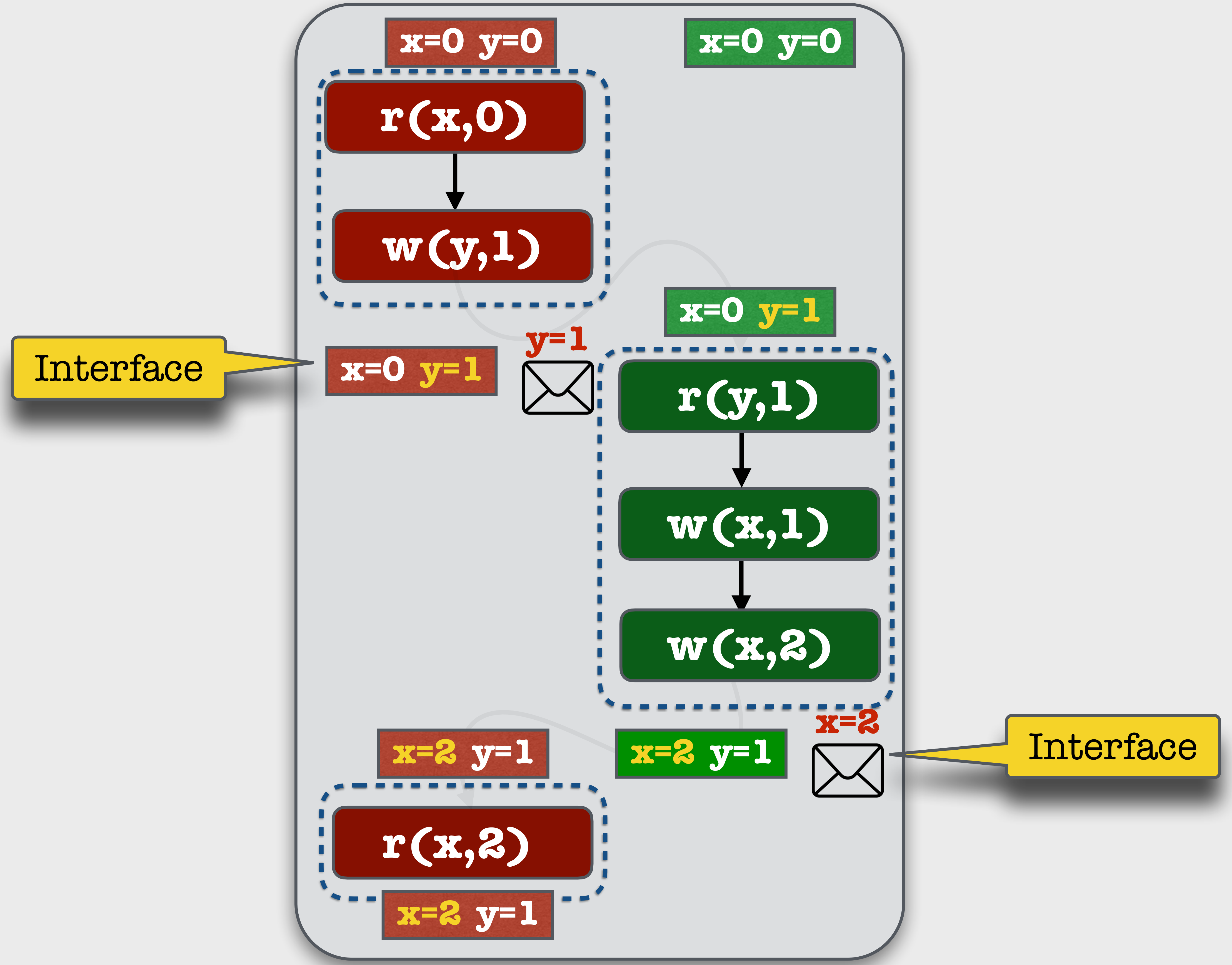


Locality

Validation

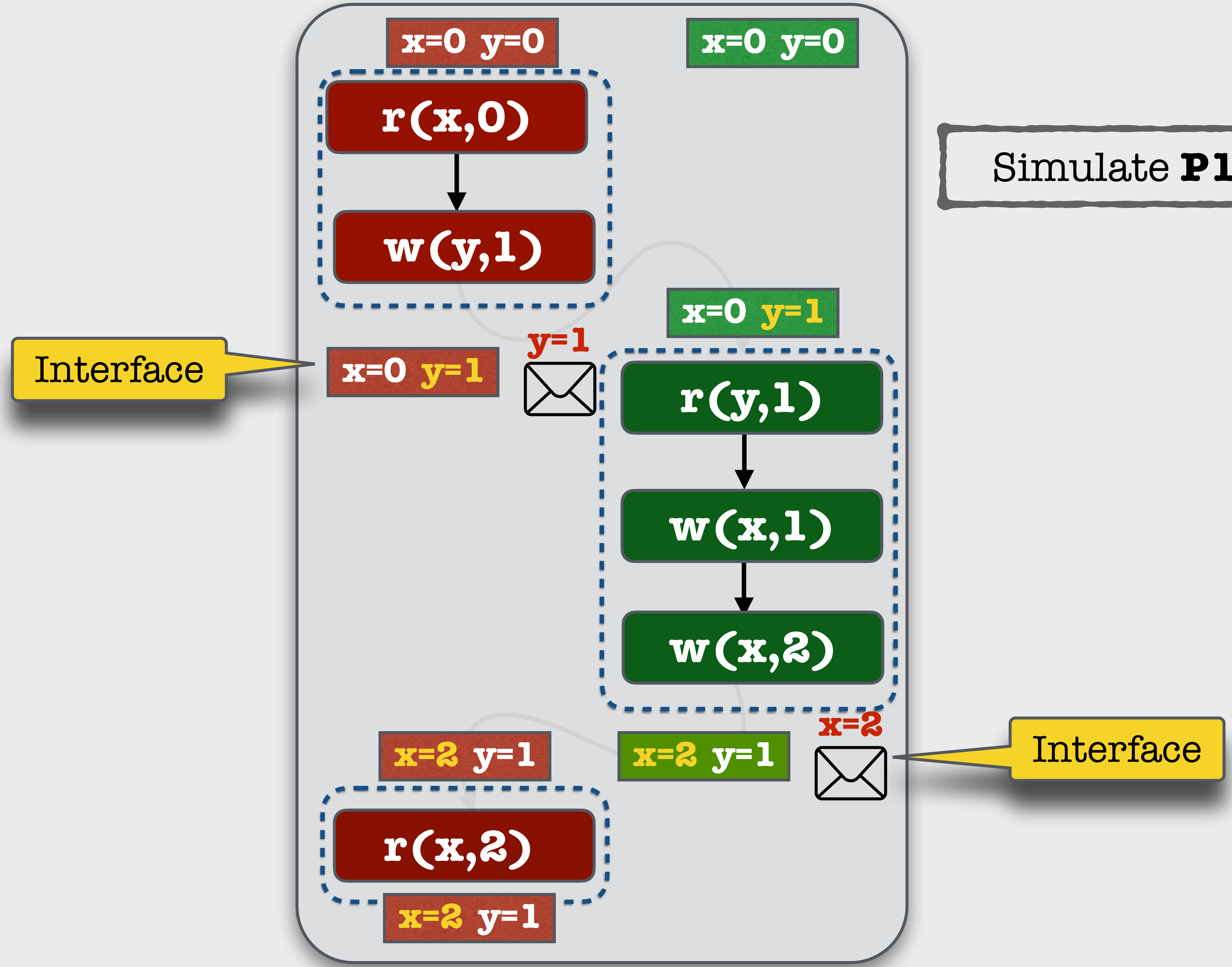


Locality Validation

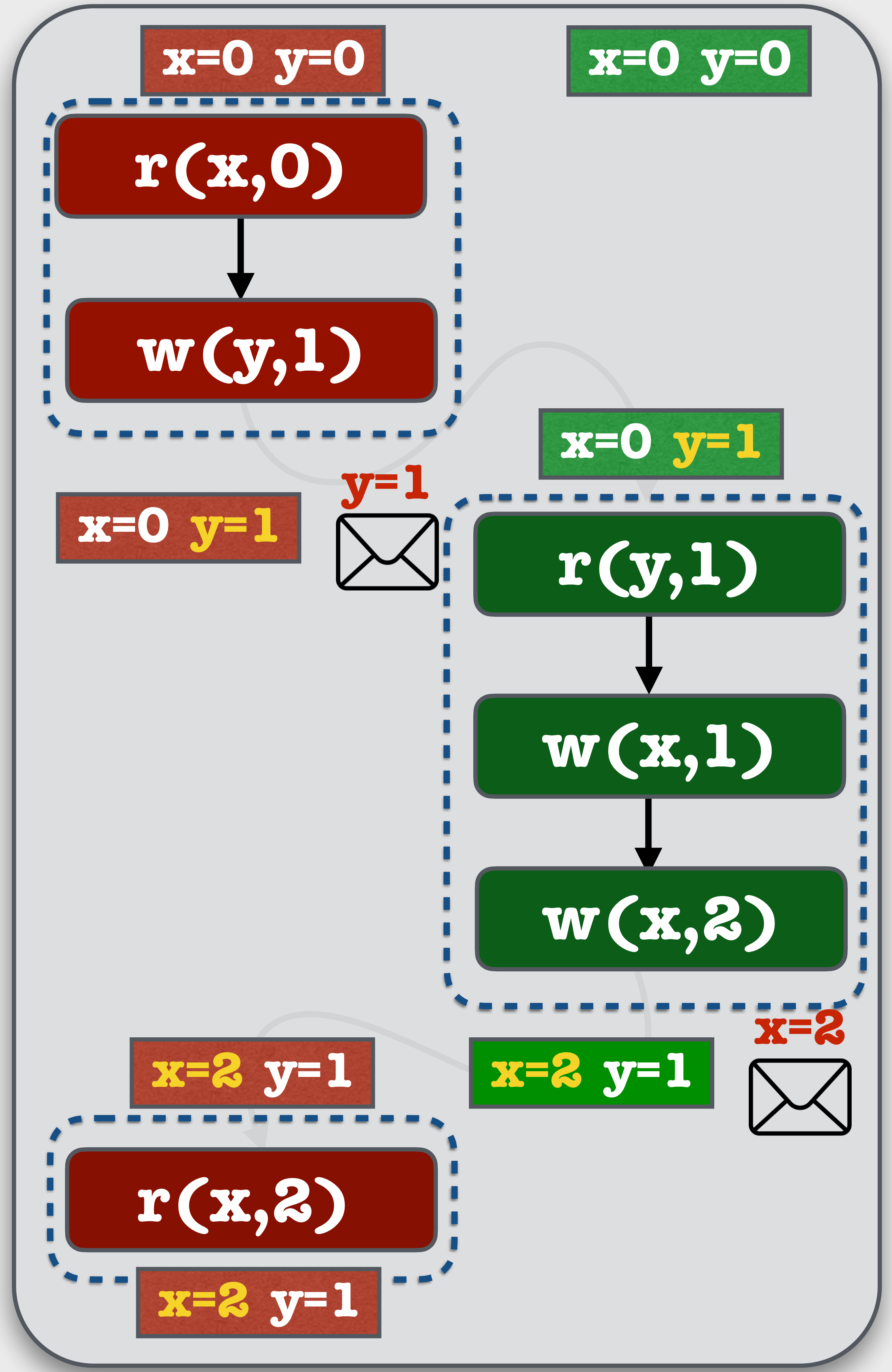


Locality Validation

Simulate **P1** under **SC**



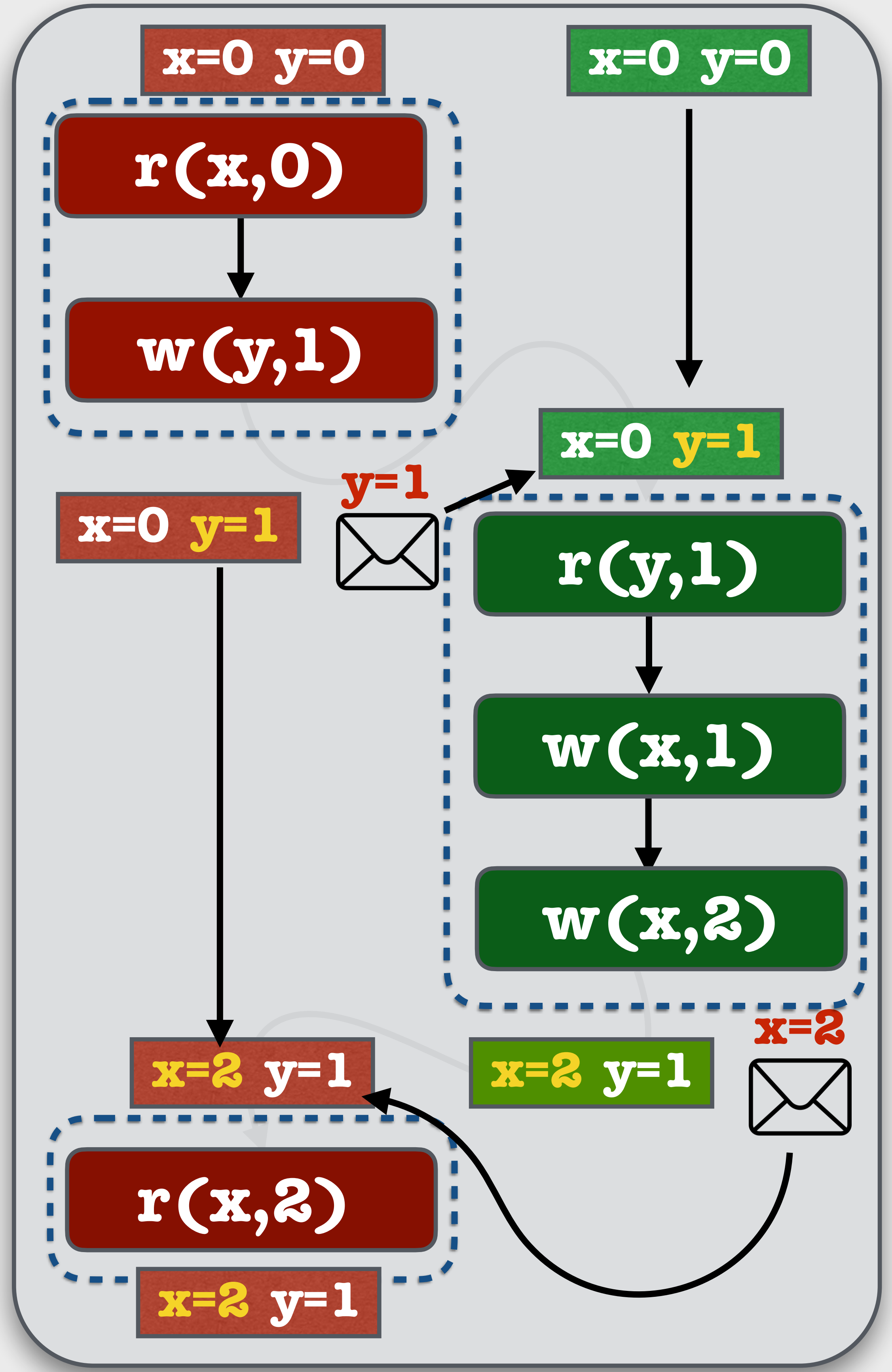
Locality Validation



Simulate **P2** under **SC**

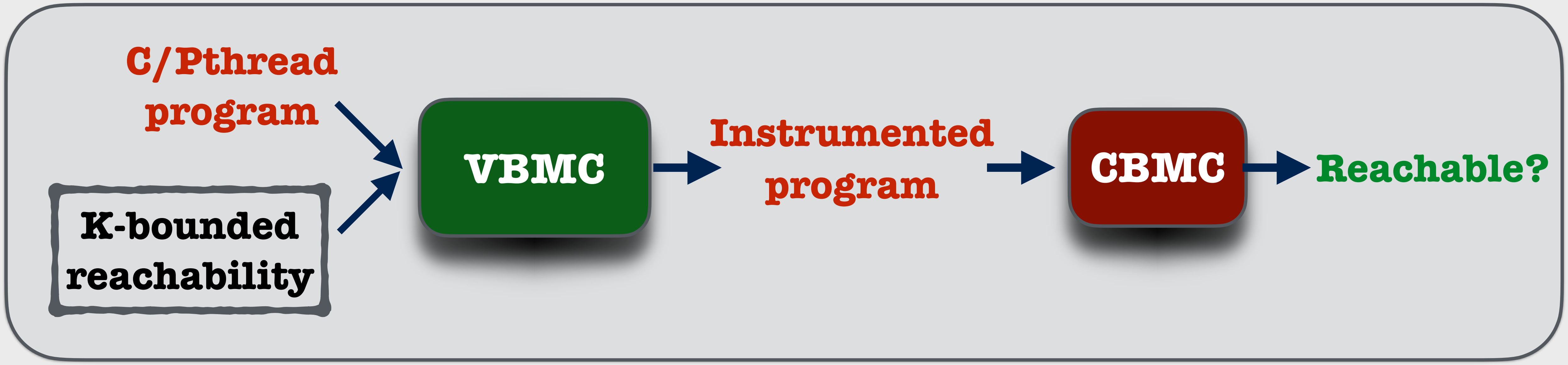
Locality

Validation



View Bounded Model Checker (VBMC)

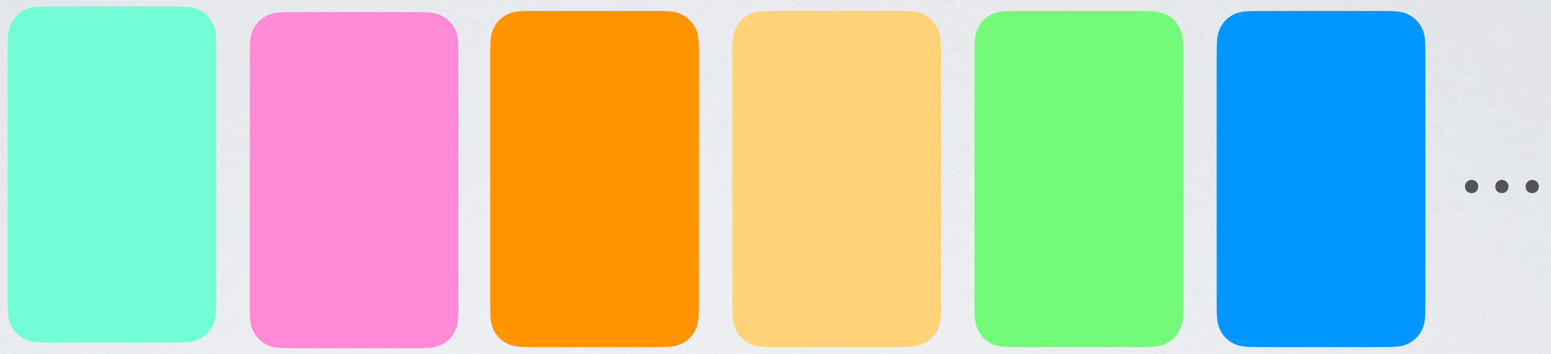
- Using CBMC as backend model checker



View Bounded Model Checker (VBMC)

- ✦ Tested with 4004 litmus tests [Sarkar et al. 2011]:
  - Same results as Herd [Alglave et al. 2014]
- ✦ Tested on concurrent benchmarks:
  - Few number of contexts sufficient for bug detection under RA
  - Catches isolated bugs faster than state of the art SMC tools Tracer, RCMC and CDSChecker

Parameterized Reachability



Inherent Undecidability

Arxiv. 2021

$x := 1$
$a := x$
$\vdots$
$\vdots$

$x := 1$
$a := x$
$\vdots$
$\vdots$

$\cdots\cdots$

$x := 1$
$a := x$
$\vdots$
$\vdots$

$\cdots\cdots$

Unboundedly many

Identical threads

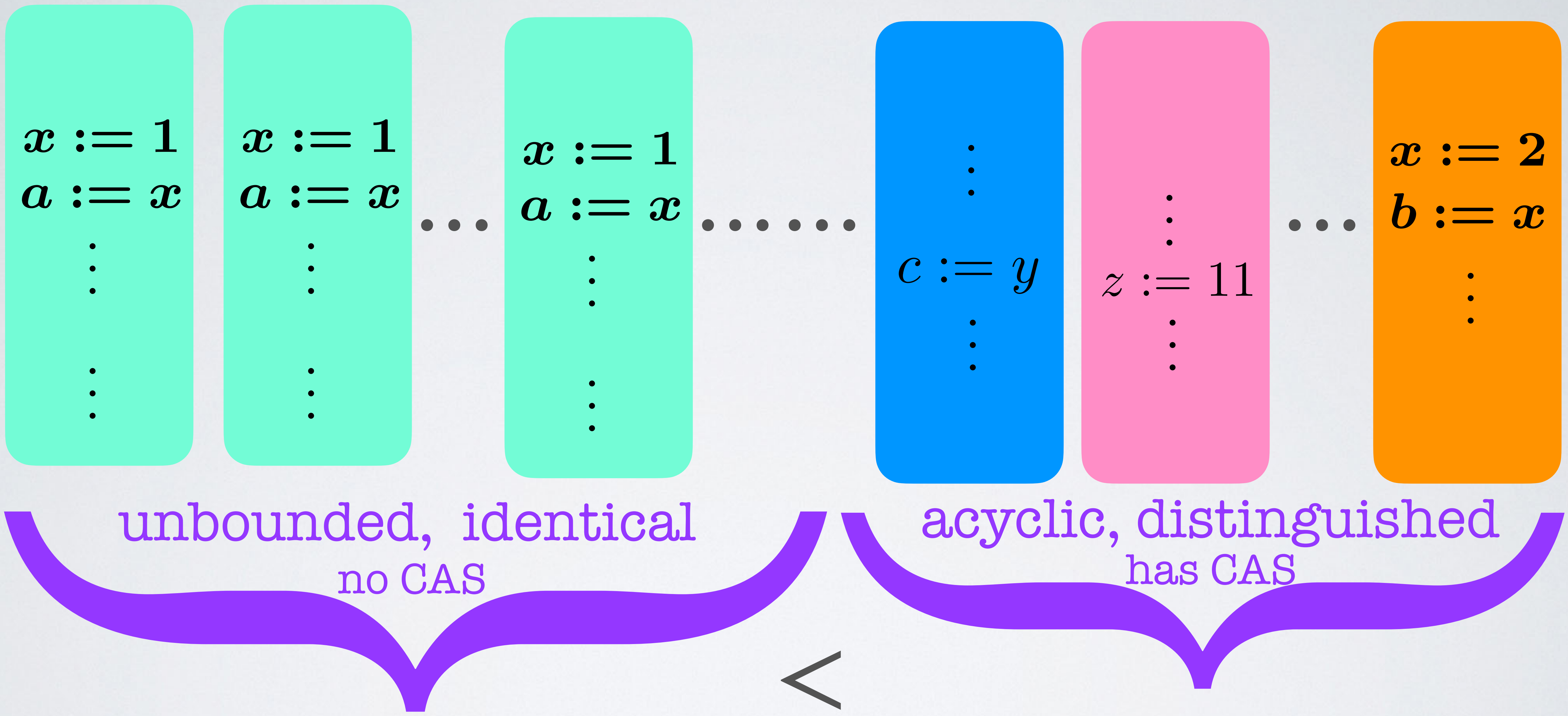
No CAS

Allowing CAS operations render state reachability undecidable for parameterized **RA,** even with acyclic, identical threads

Simulate the non parameterized setting of PLDI'19

QBF Sat

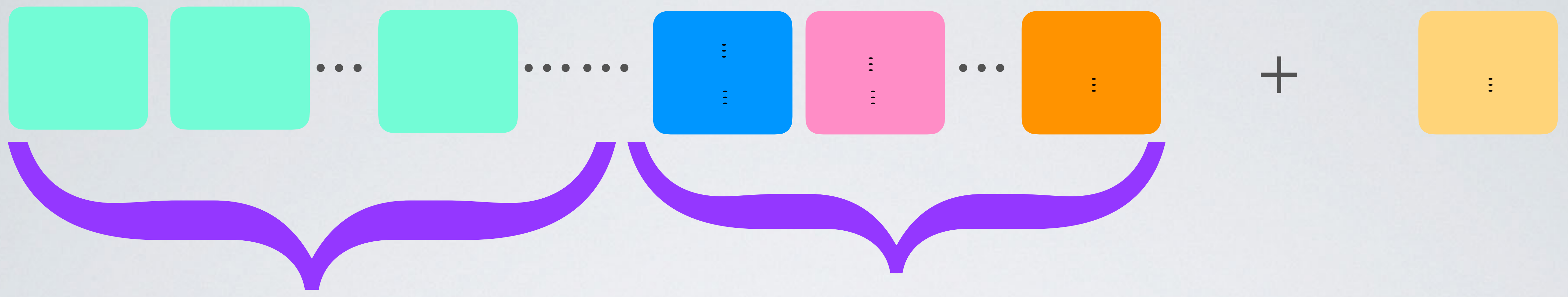
$$\leq$$



$$\leq$$

Query evaluation in linear Datalog

PSPACE completeness

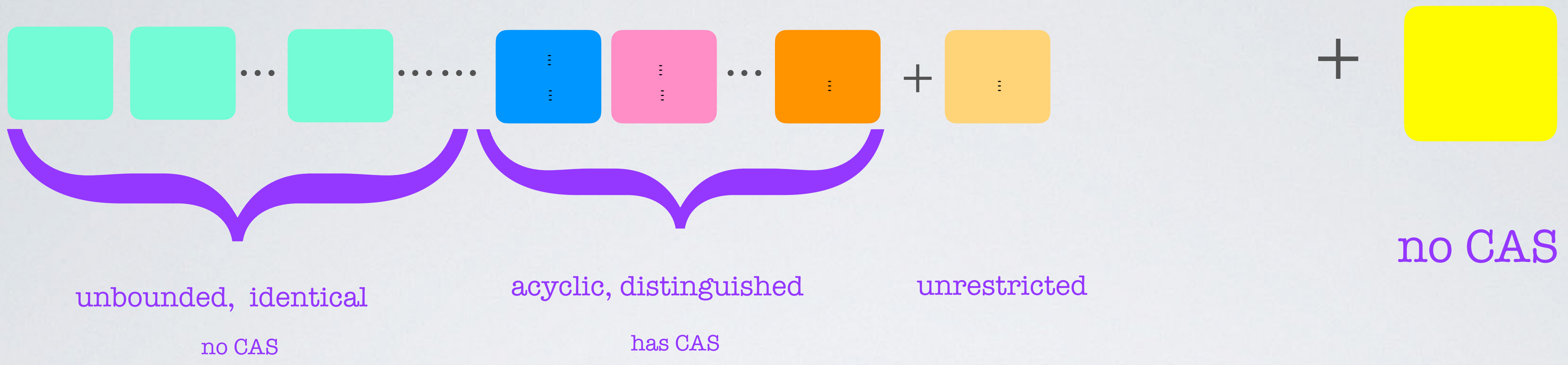


unbounded, identical
no CAS

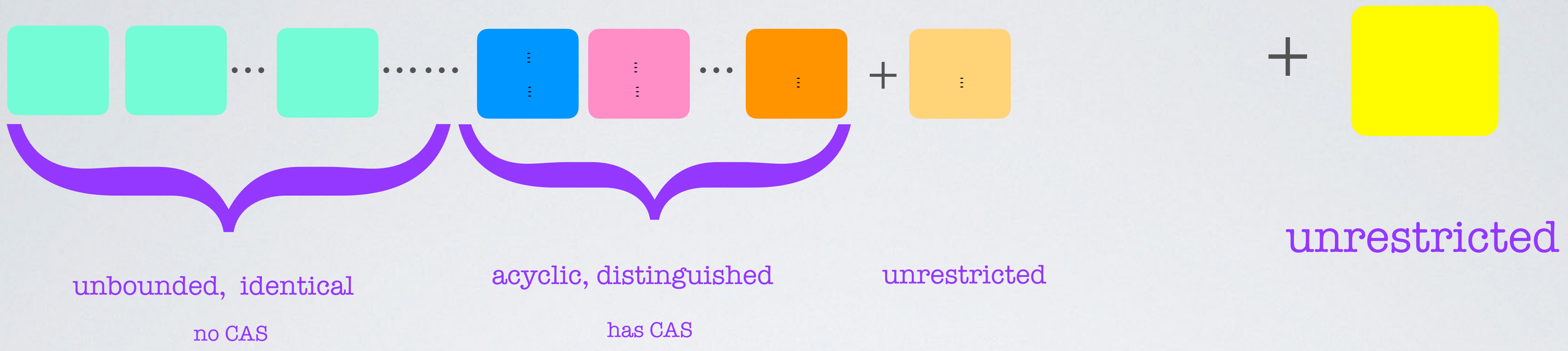
acyclic, distinguished
has CAS

unrestricted

NEXPTIME completeness



Non primitive recursive


unbounded, identical
no CAS
acyclic, distinguished
has CAS
unrestricted
unrestricted

Open

Thankyou