Monitoring Sequential Consistency in Relaxed Memory Models

Jacob Burnim, Koushik Sen, Christos Stergiou

Sequential Consistency

- Lamport: "… the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in the sequence in the order specified by its program"
- Why is it important
  - What programmers assume
  - What programming languages guarantee for data-race free programs

Relaxed Memory Models

Initially x = y = 0

thread1:	thread2:
1: x = 1	3: y = 1
2: t1 = y	4: t2 = x

assert(not (t1 == 0 && t2 == 0))

- Relaxed memory model => increased concurrency and performance
- All SC executions satisfy the assertion
- Under relaxed memory models writes to x and y could be buffered until after the reads by t1 and t2

Problem

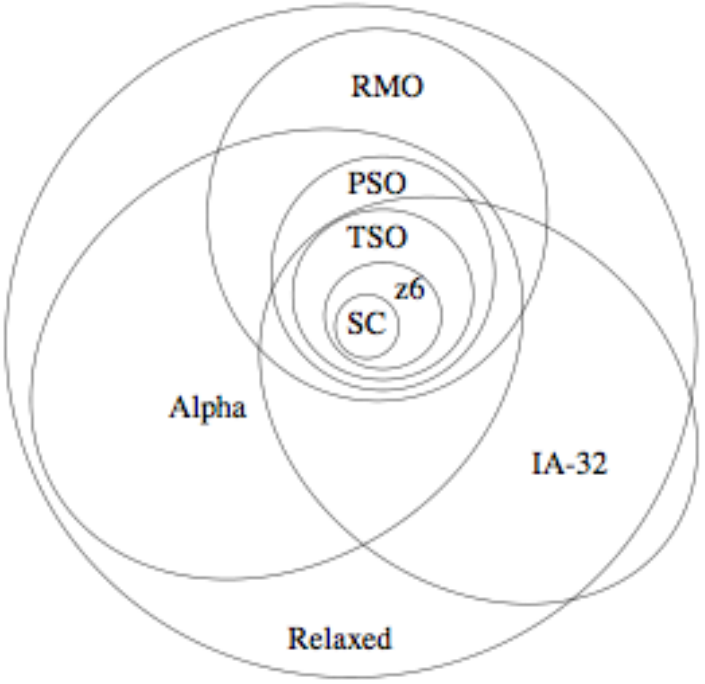
- Data races => No sequential consistency guarantee
  - Programs can exhibit unintuitive behaviors
- Software developers write programs with intentional data races
  - Highly concurrent libraries, lock-free data structures
  - Custom synchronization on certain frequent operations
- Traditional approaches for verification of concurrent programs do not scale
  - Model checking
  - Additional non-determinism from underlying memory model

Idea

- Programmers expect their software to be sequentially consistent
- Focus on violations of sequential consistency
- Monitoring: find all violations of sequential consistency by exploring only sequentially consistent executions
  - Combine with model checking
  - For each explored interleaving run monitoring algorithm

Relaxed Memory Models

- Common Relaxed Memory Models
  - TSO: Total Store Order
    x86 (approx)
  - PSO: Partial Store Order
    Power, ARM (approx)
- Good approximation of existing memory models
  - Bugs under TSO and PSO capture most bugs under modern architectures



Courtesy: Sebastian Burckhardt PhD Dissertation

Sober Monitoring

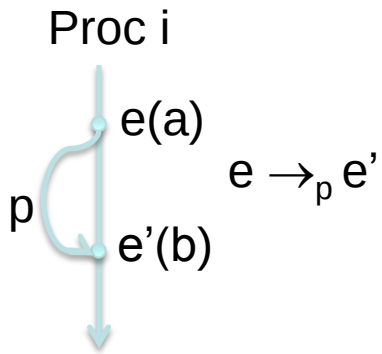
- Burckhardt et al. proposed monitoring for TSO relaxed memory model
- TSO : allows stores to be reordered past later loads, but maintains a total order over stores
- Based on axiomatic definition of TSO
- Has bug, their model turned out to be stricter that traditional TSO

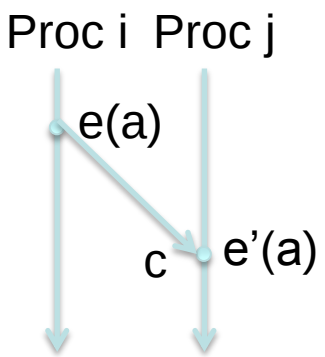
Our approach

- Provide monitor algorithms for TSO and PSO
- Based on operational semantics
  - Thus more intuitive
- Faster than previous approaches
- Sound and complete
- Re-execute violations under relaxed memory model
- Observe if violation is benign or leads to a bug
- Fixed problem in Sober approach

Axiomatic Definitions

- Relations on traces (= sets of loads & stores)
  - Program relation ($\rightarrow_p$): same thread events
  - Conflict relation ($\rightarrow_c$): racing events





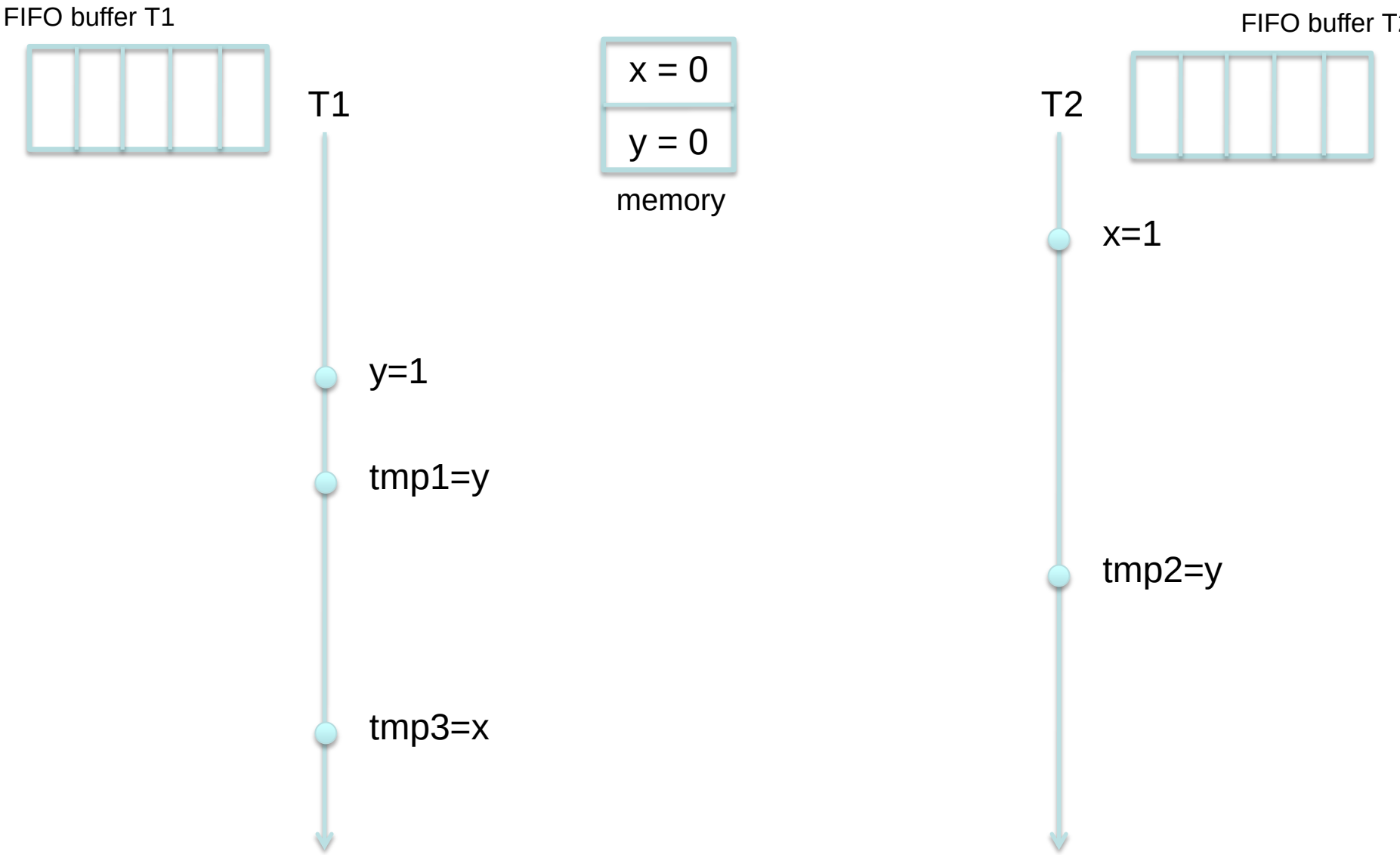
e = R(a) or W(a) and e' = W(a)

e = W(a) and e' = R(a) and e=src(e')

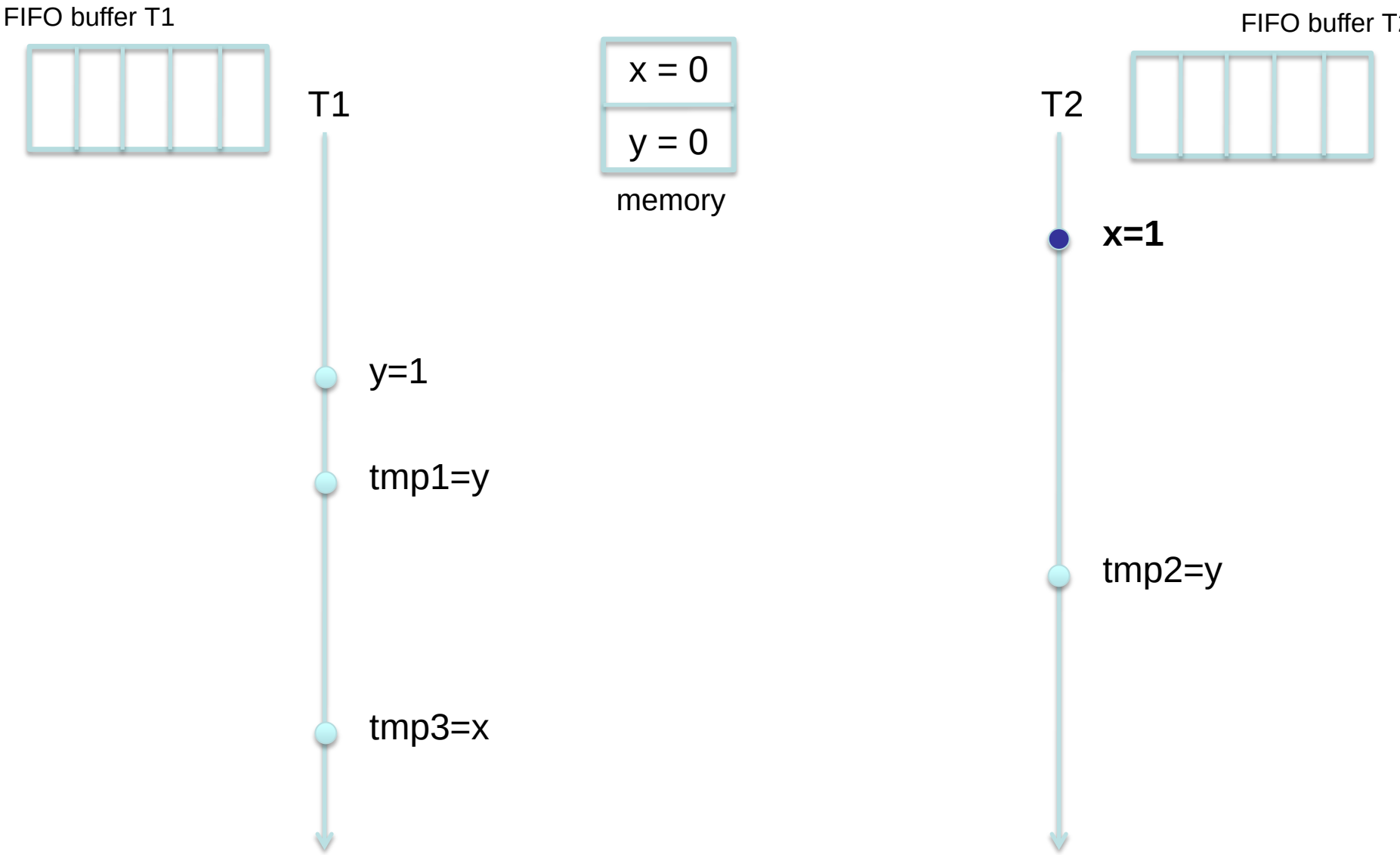
Axiomatic Definitions

- Happens-before relation: $\rightarrow_{hb} = (\rightarrow_c \cup \rightarrow_p)$
- A trace E is sequentially consistent iff relation $\rightarrow^*_{hb}$ is acyclic on the events of E

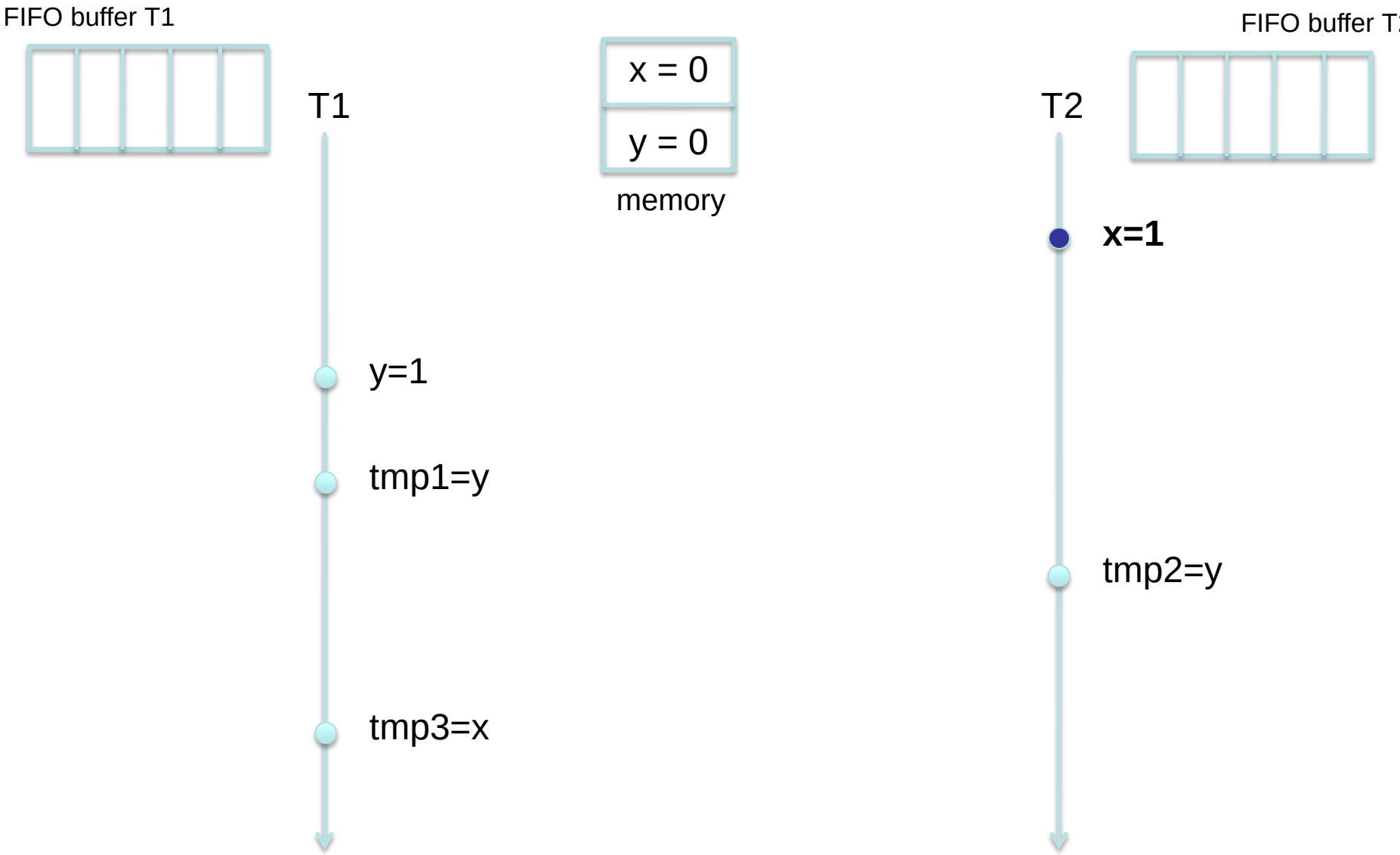
TSO Operational Definition



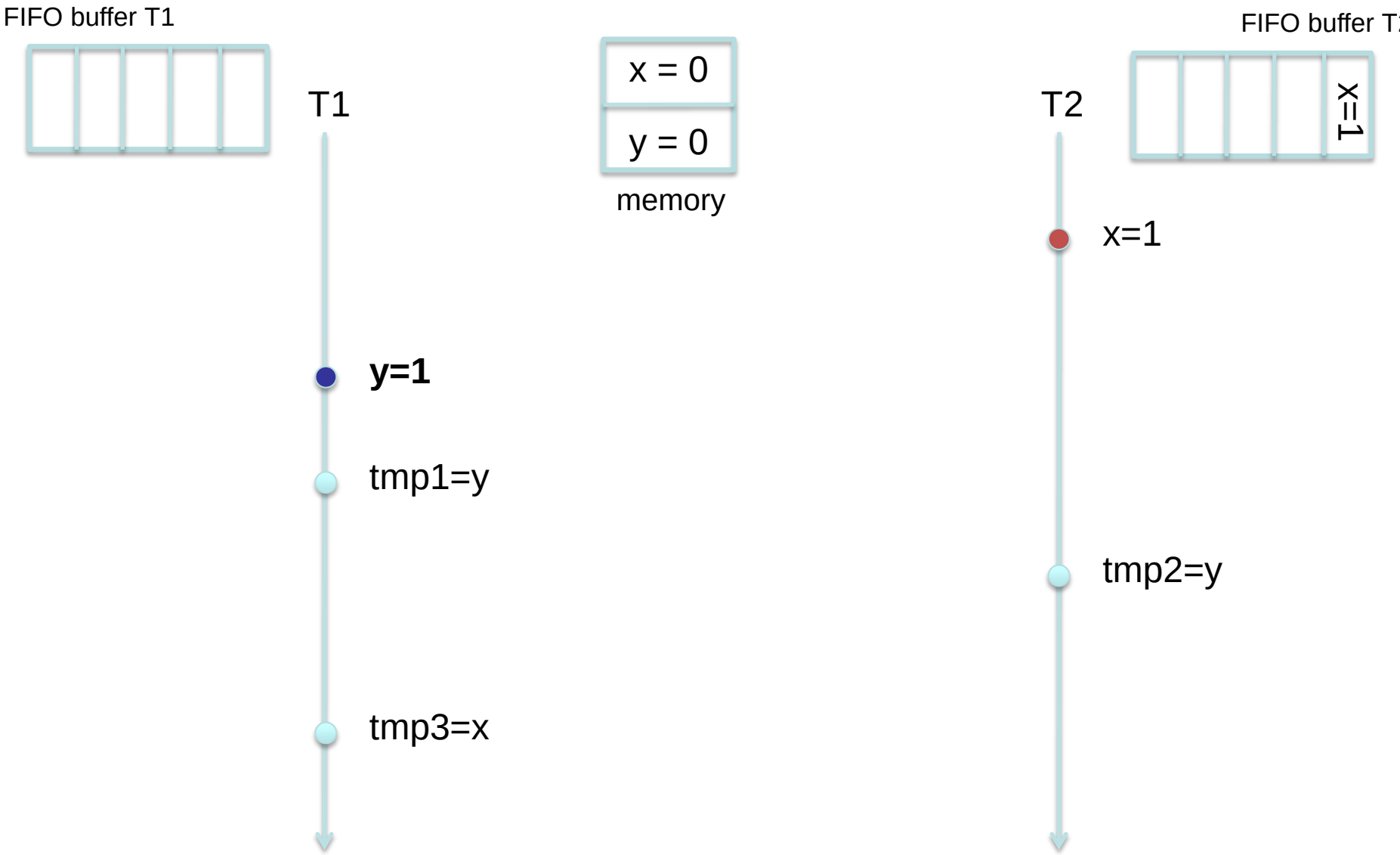
TSO Operational Definition



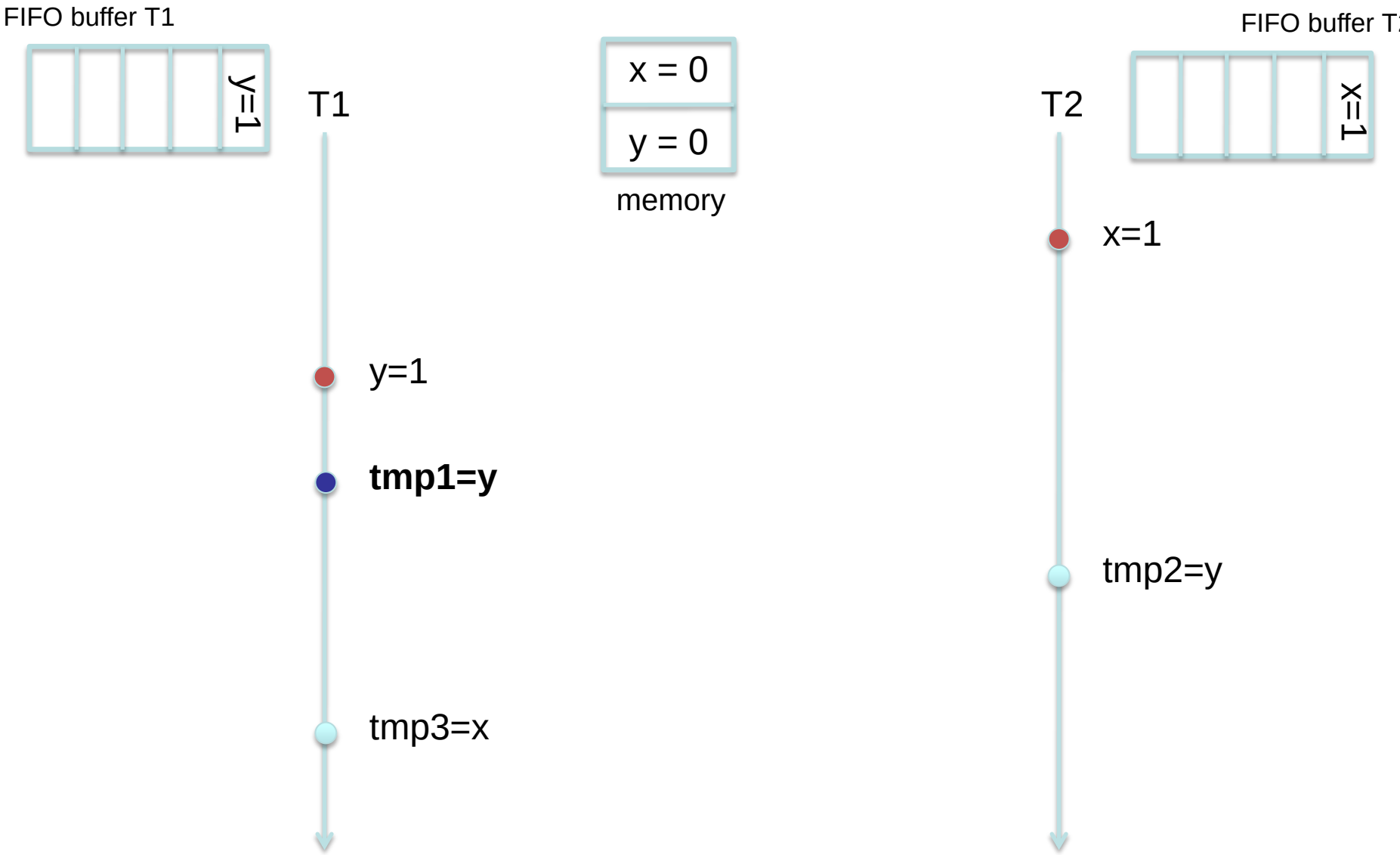
TSO Operational Definition



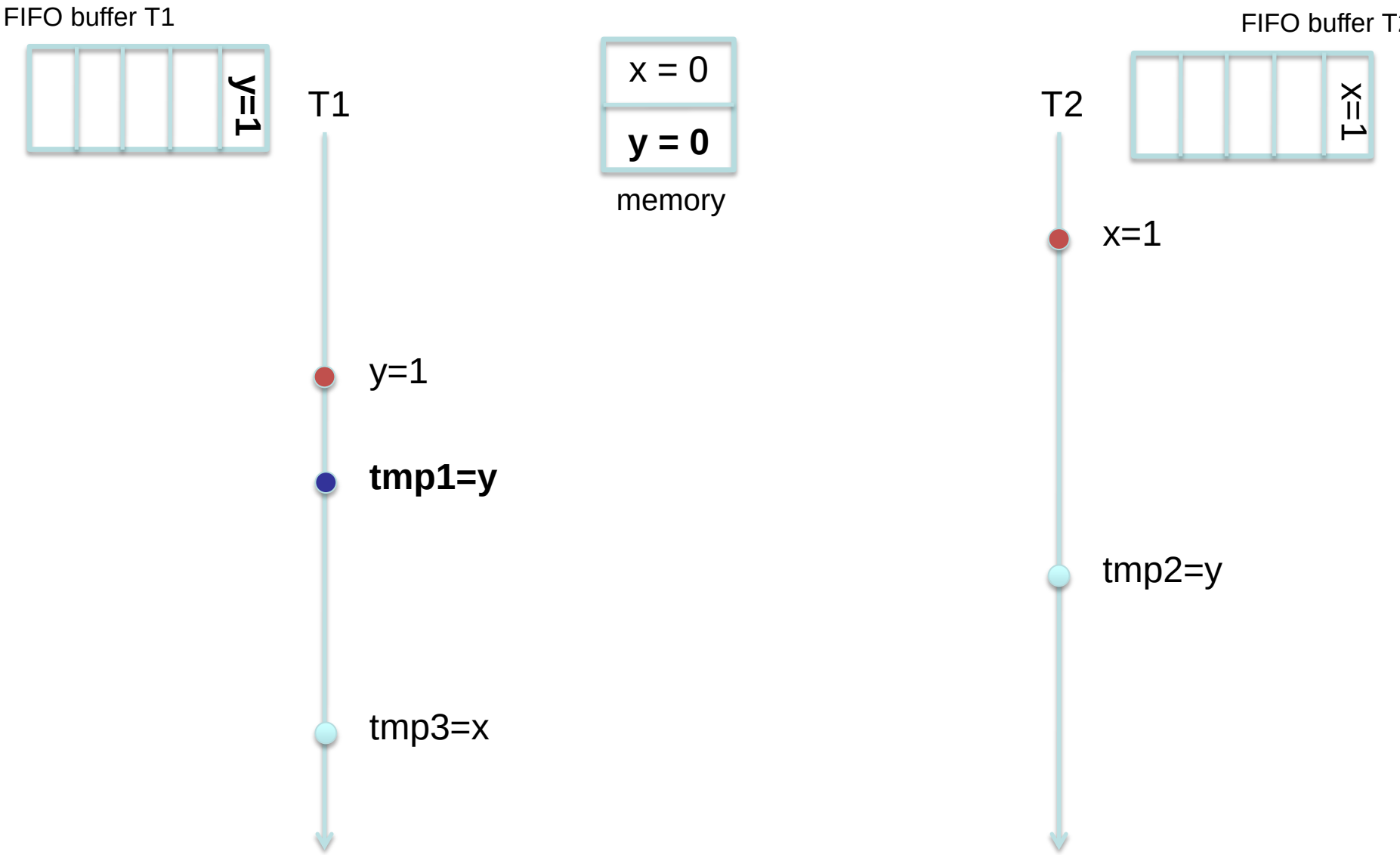
TSO Operational Definition



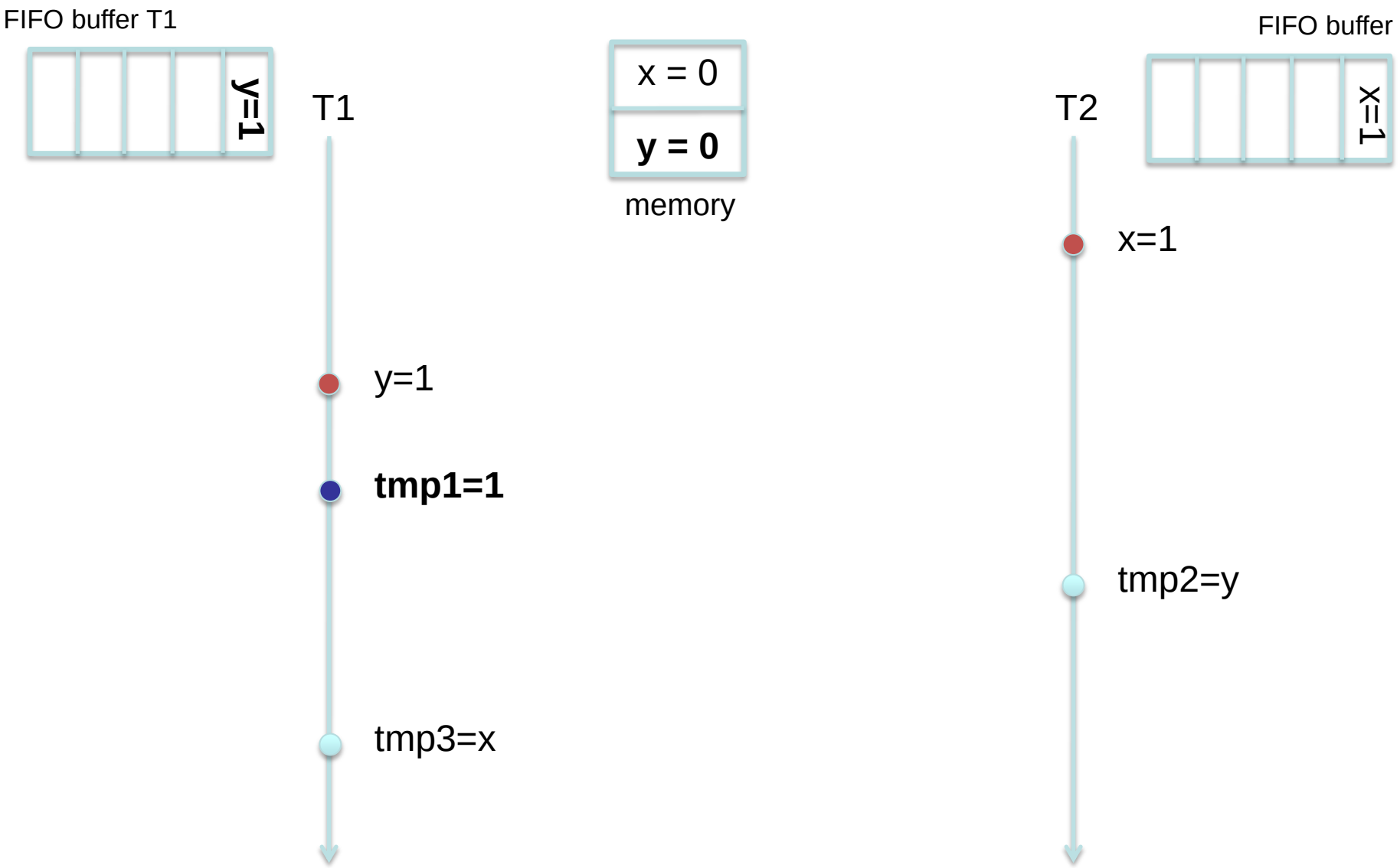
TSO Operational Definition



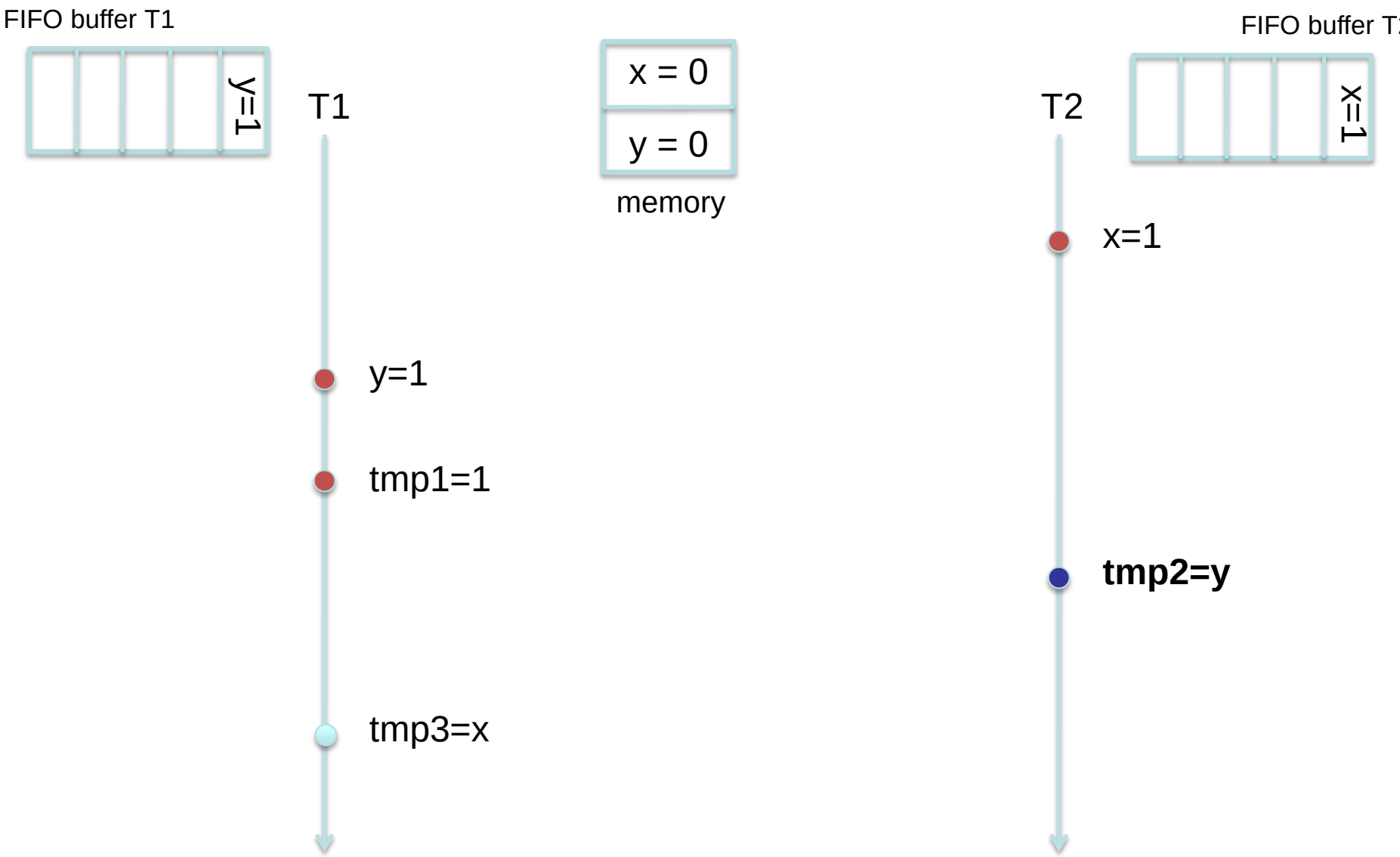
TSO Operational Definition



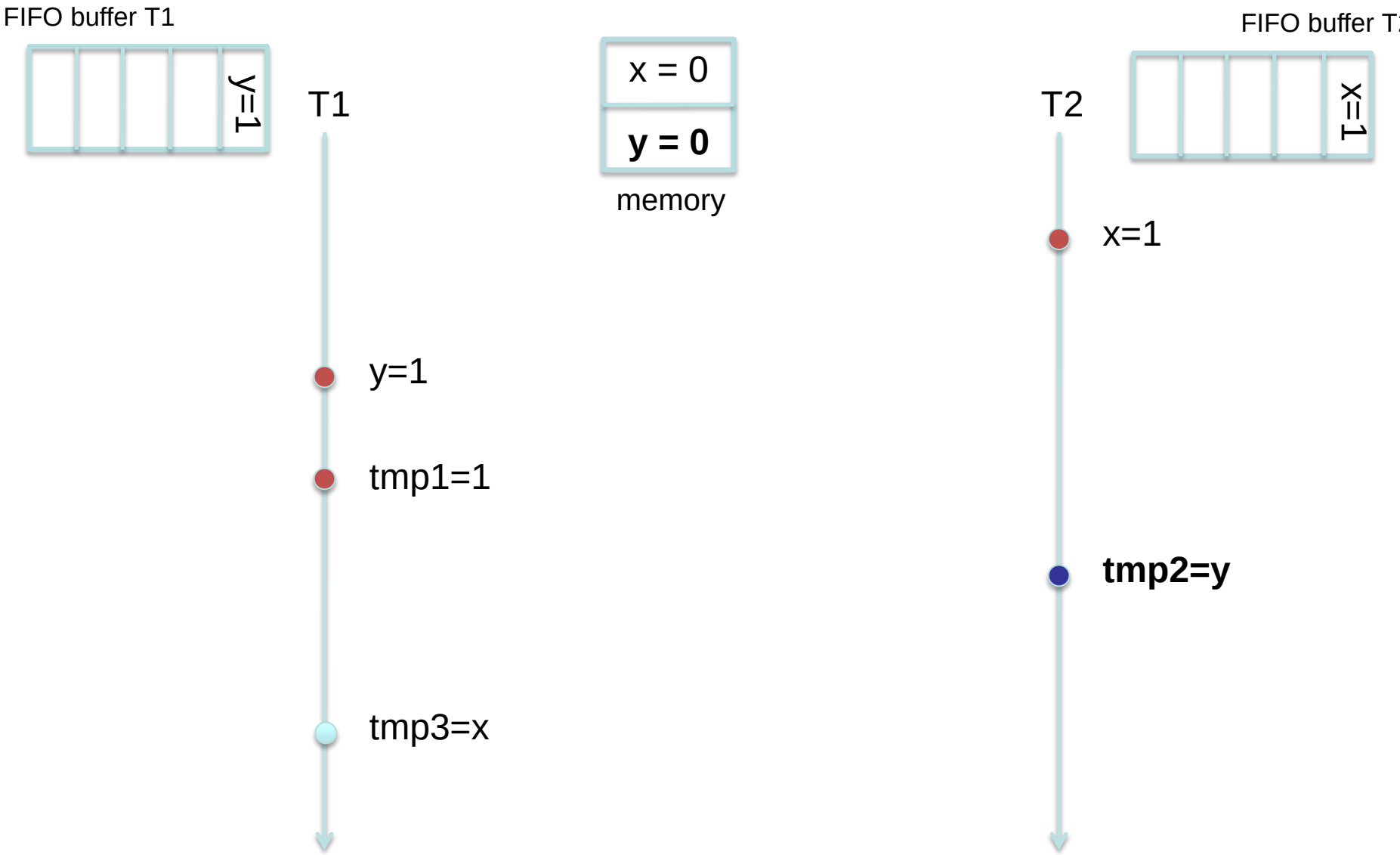
TSO Operational Definition



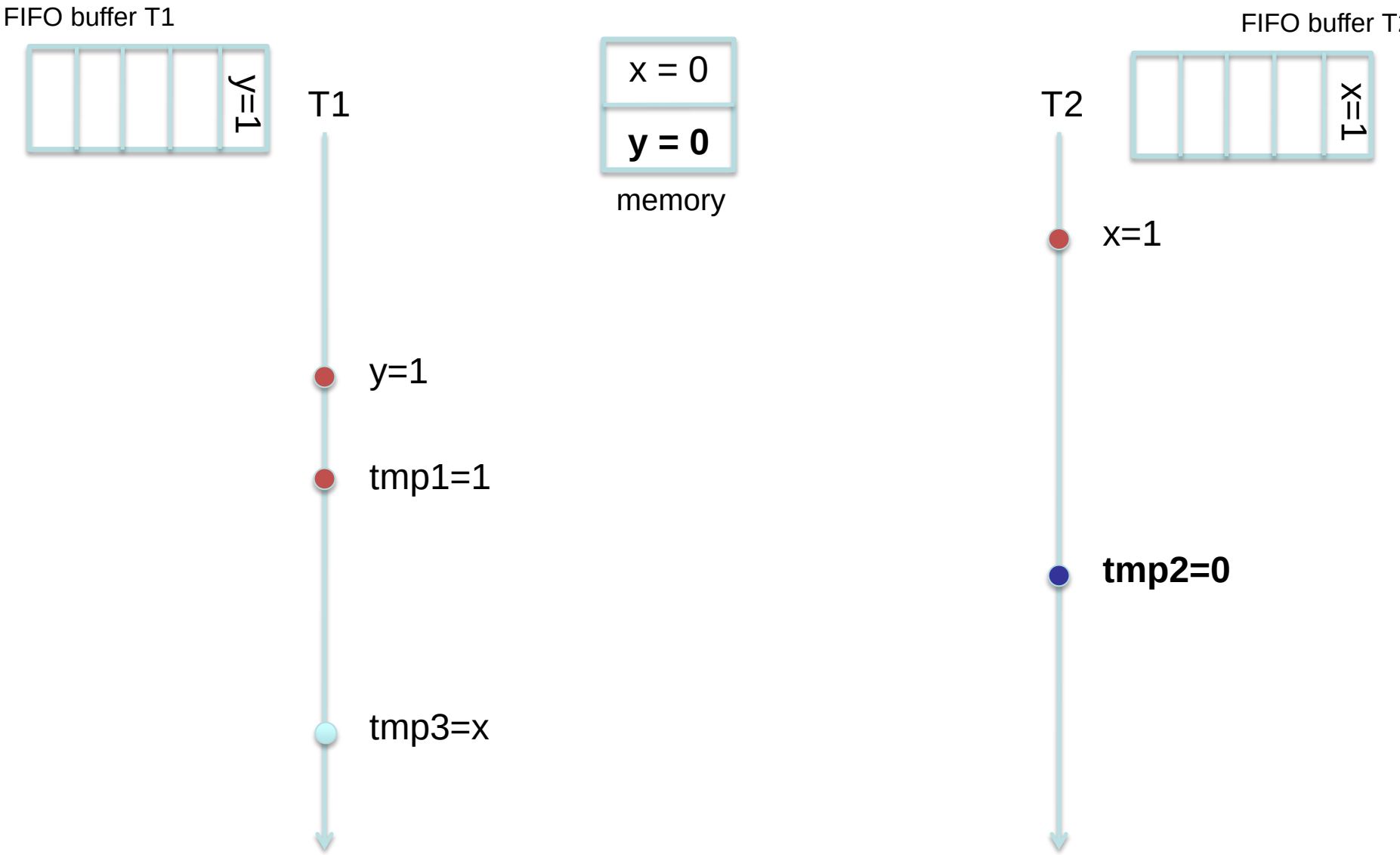
TSO Operational Definition



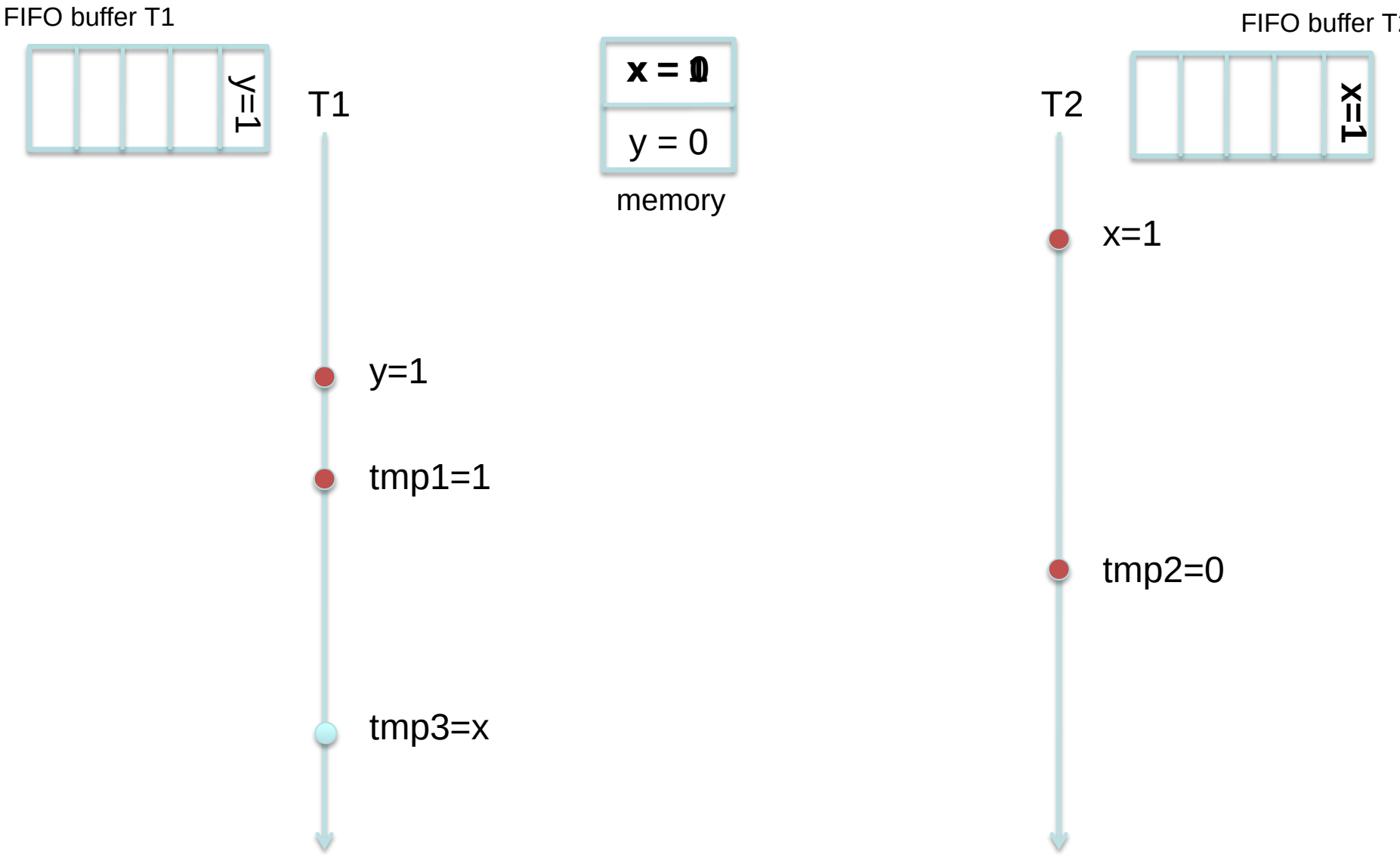
TSO Operational Definition



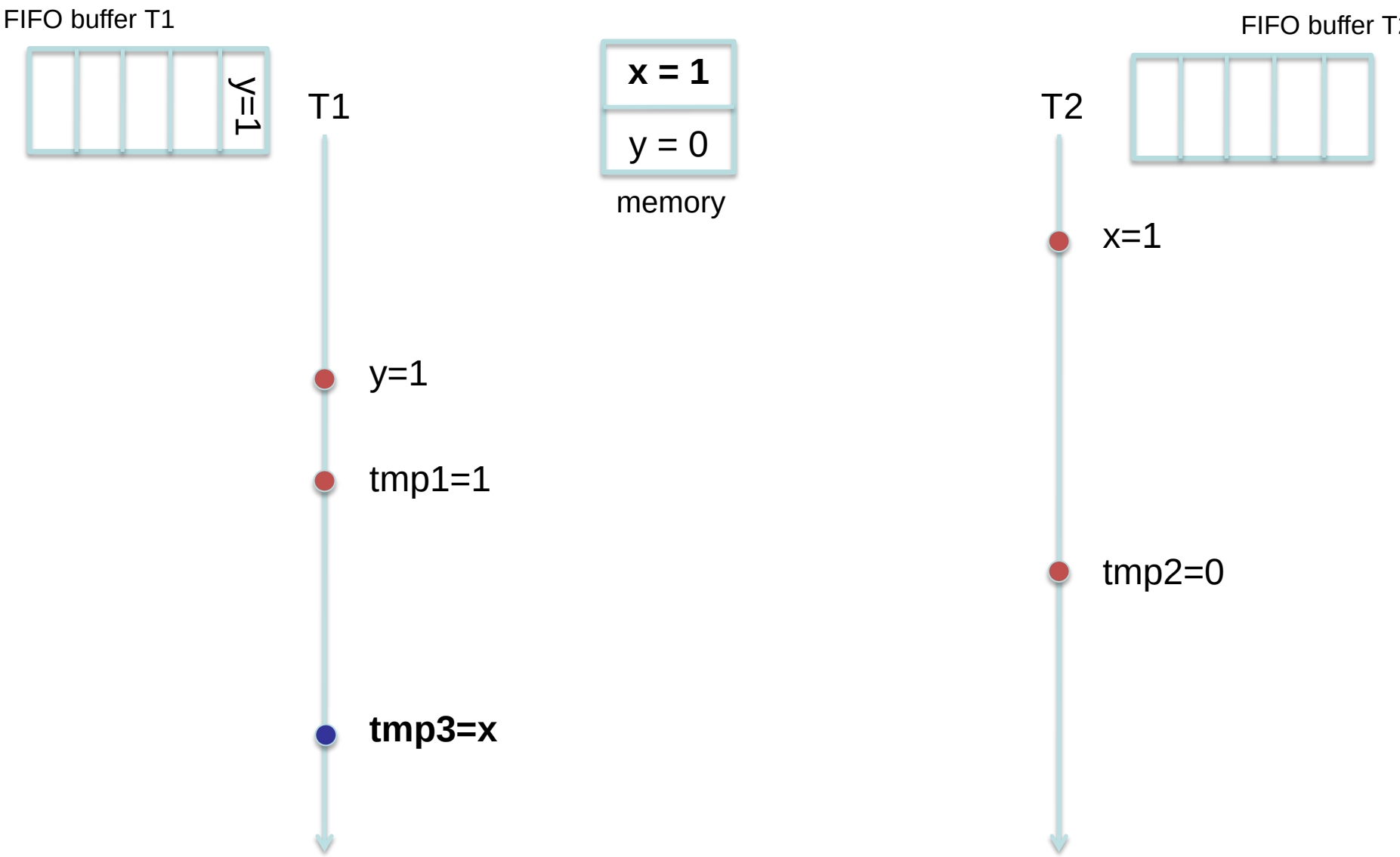
TSO Operational Definition



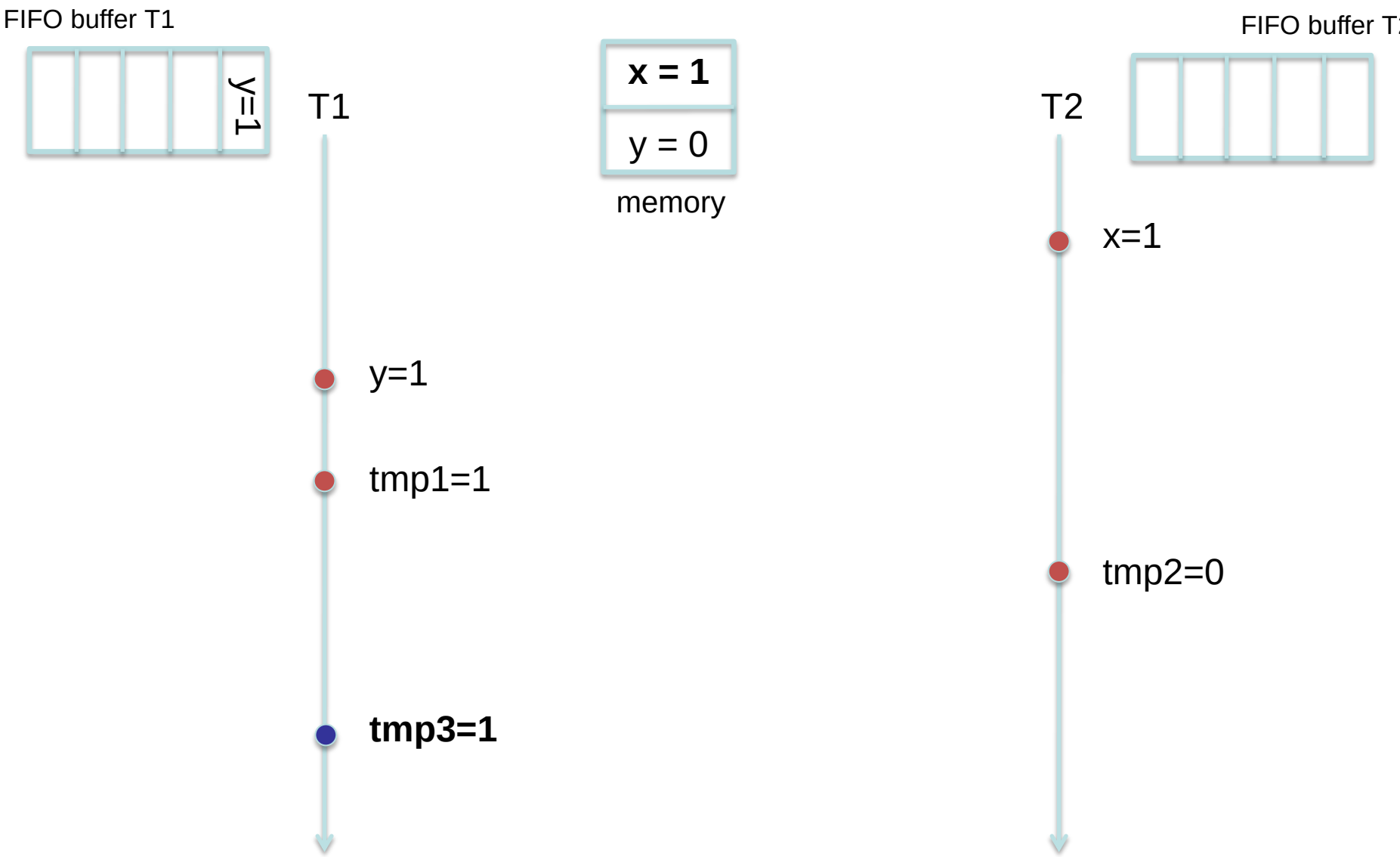
TSO Operational Definition



TSO Operational Definition



TSO Operational Definition



Monitor Algorithms

- Devise monitoring algorithms for TSO and PSO
- Use axiomatic definition of sequential consistency
  - Find cycles in happens-before relation
- Base algorithms on intuitive operational simulation instead of complex axiomatic semantics

Monitor Algorithms

- Model-check program and record trace of sequential consistent execution
- Iterate over trace and simulate relaxed memory model
  - thread local buffers and memory
- Delay a commit as long as possible
- Before each event check if a pending store could be committed after the event.
- If the event happens-after the store, report SC violation

Example

Initially: x = 0, y = 0

Thread1	Thread2
x=1	y=1
tmp1=y	tmp2=x

assert(not (tmp1==0 && tmp2==0))

Sequentially Consistent Execution Trace:

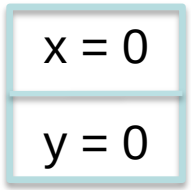
Example

simulated FIFO buffer T1

T1

x=1

tmp1=y



simulated memory

T2

y=1

tmp2=x

Simulated FIFO buffer T

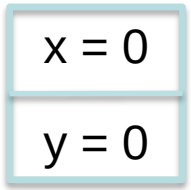
Example

simulated FIFO buffer T1

T1

x=1

tmp1=y



simulated memory

T2

y=1

tmp2=x

simulated FIFO buffer T

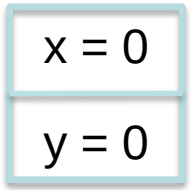
Example

simulated FIFO buffer T1

T1

x=1

tmp1=y



simulated memory

T2

y=1

tmp2=x

simulated FIFO buffer T

Example

simulated FIFO buffer T1

T1

x=1

tmp1=y

x = 0

y = 0

simulated memory

T2

y=1

tmp2=x

simulated FIFO buffer T

Example

simulated FIFO buffer T1

T1

x=1

tmp1=y

x = 0

y = 0

simulated memory

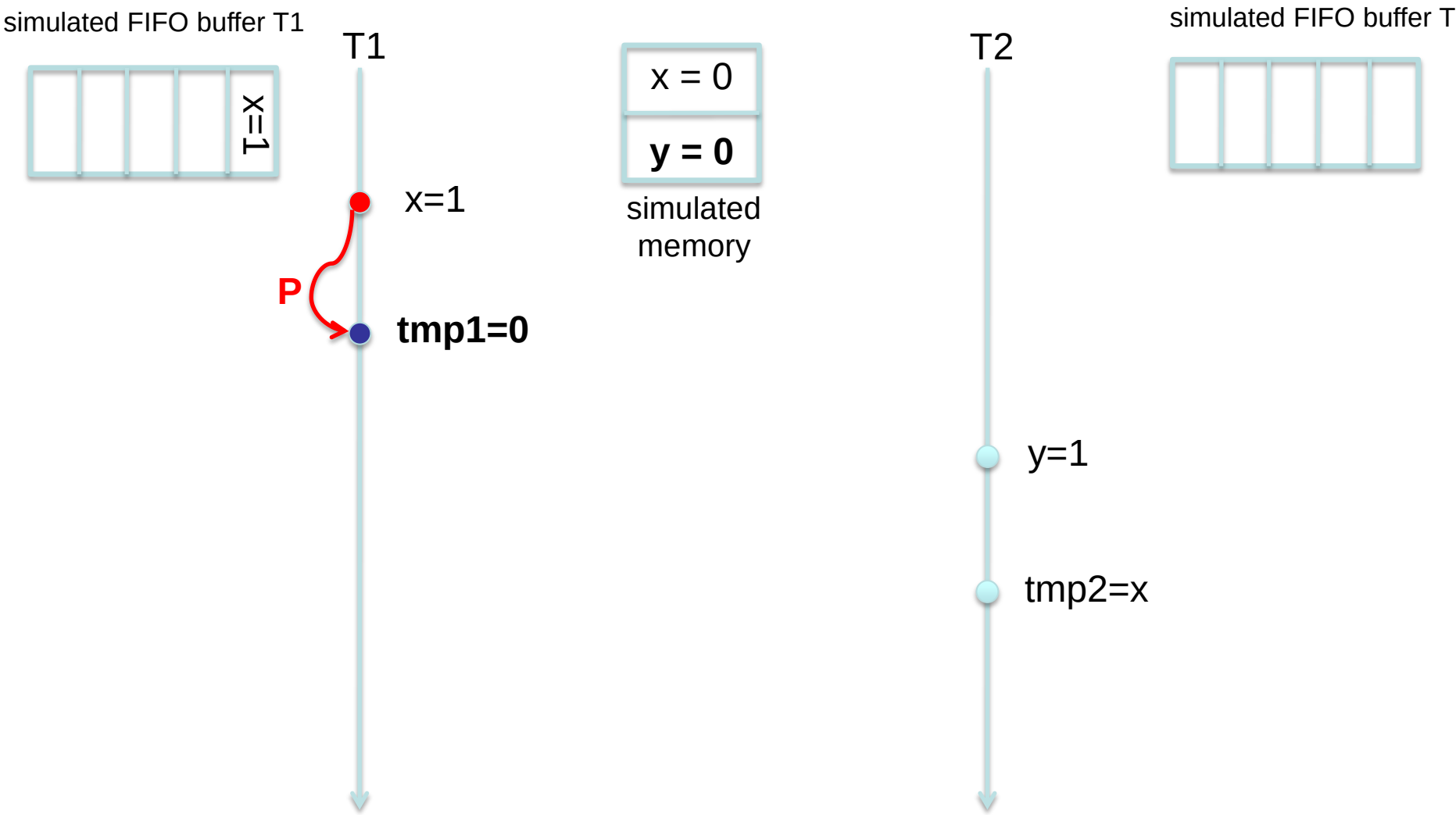
T2

y=1

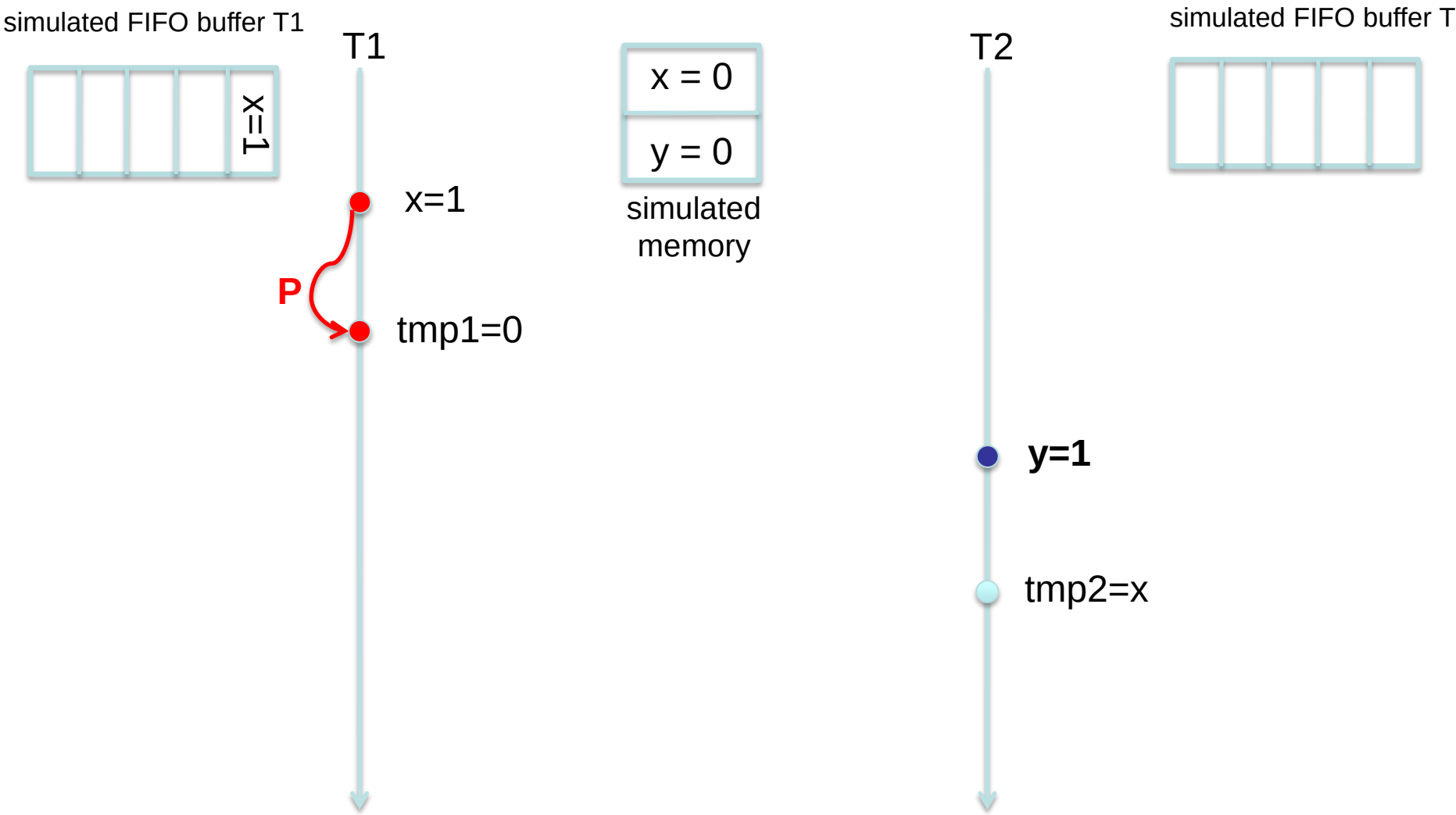
tmp2=x

simulated FIFO buffer T

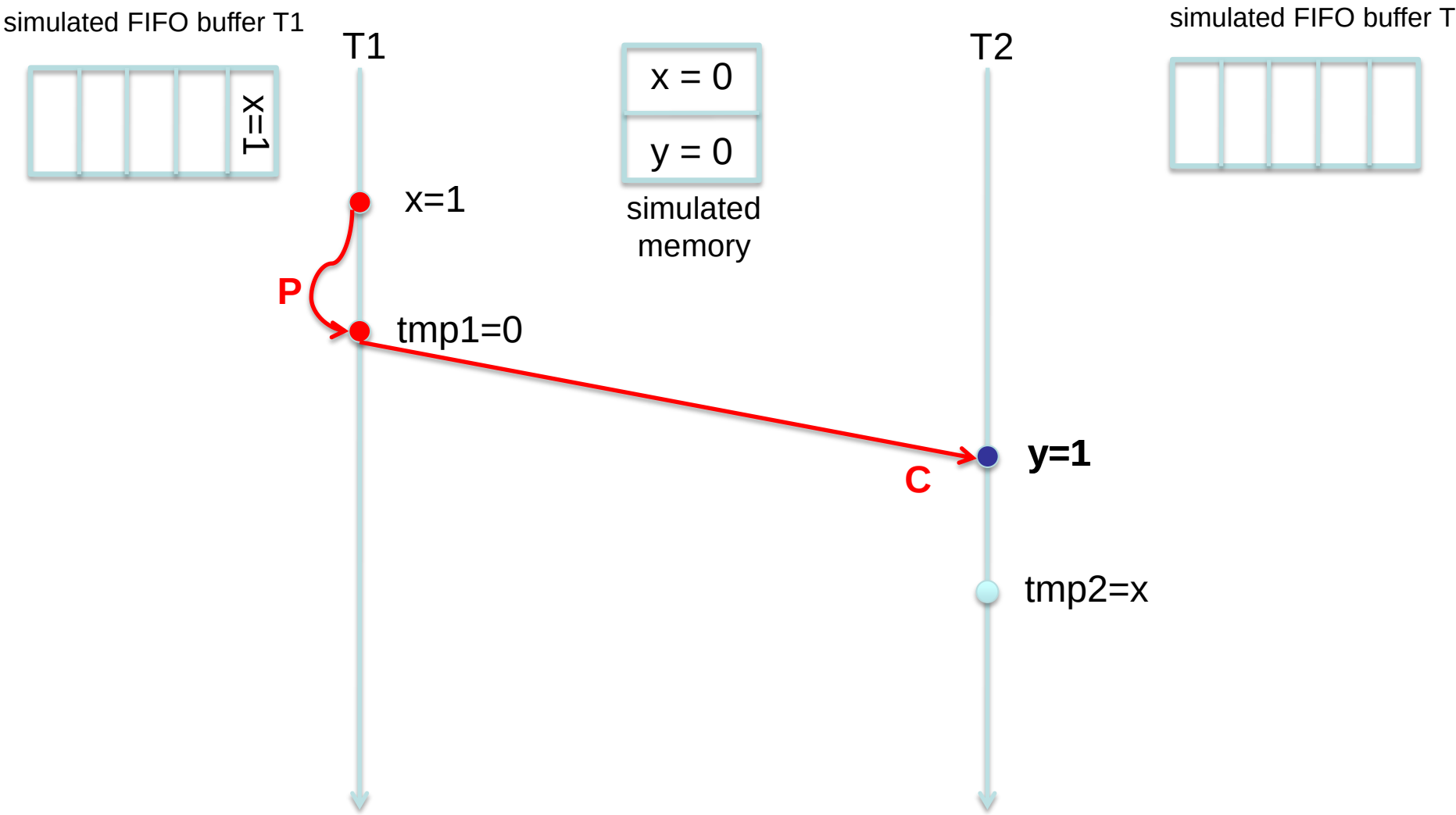
Example


simulated FIFO buffer T1
T1=x
T1
x=1
P
tmp1=0
x = 0
y = 0
simulated memory
T2
y=1
tmp2=x
simulated FIFO buffer T

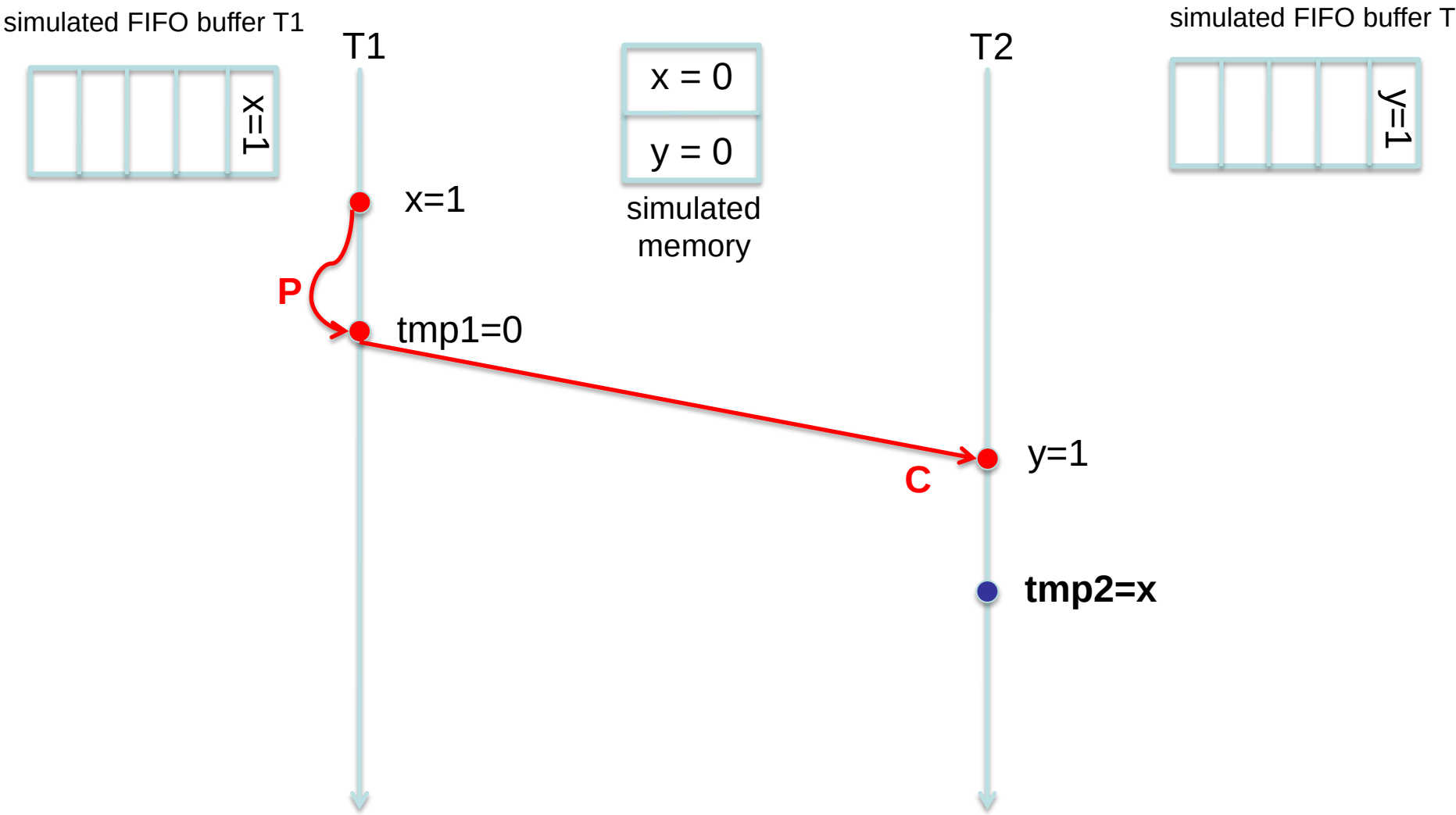
Example


simulated FIFO buffer T1
T1=x
T1
x=1
P
tmp1=0
x = 0
y = 0
simulated memory
T2
y=1
tmp2=x
simulated FIFO buffer T

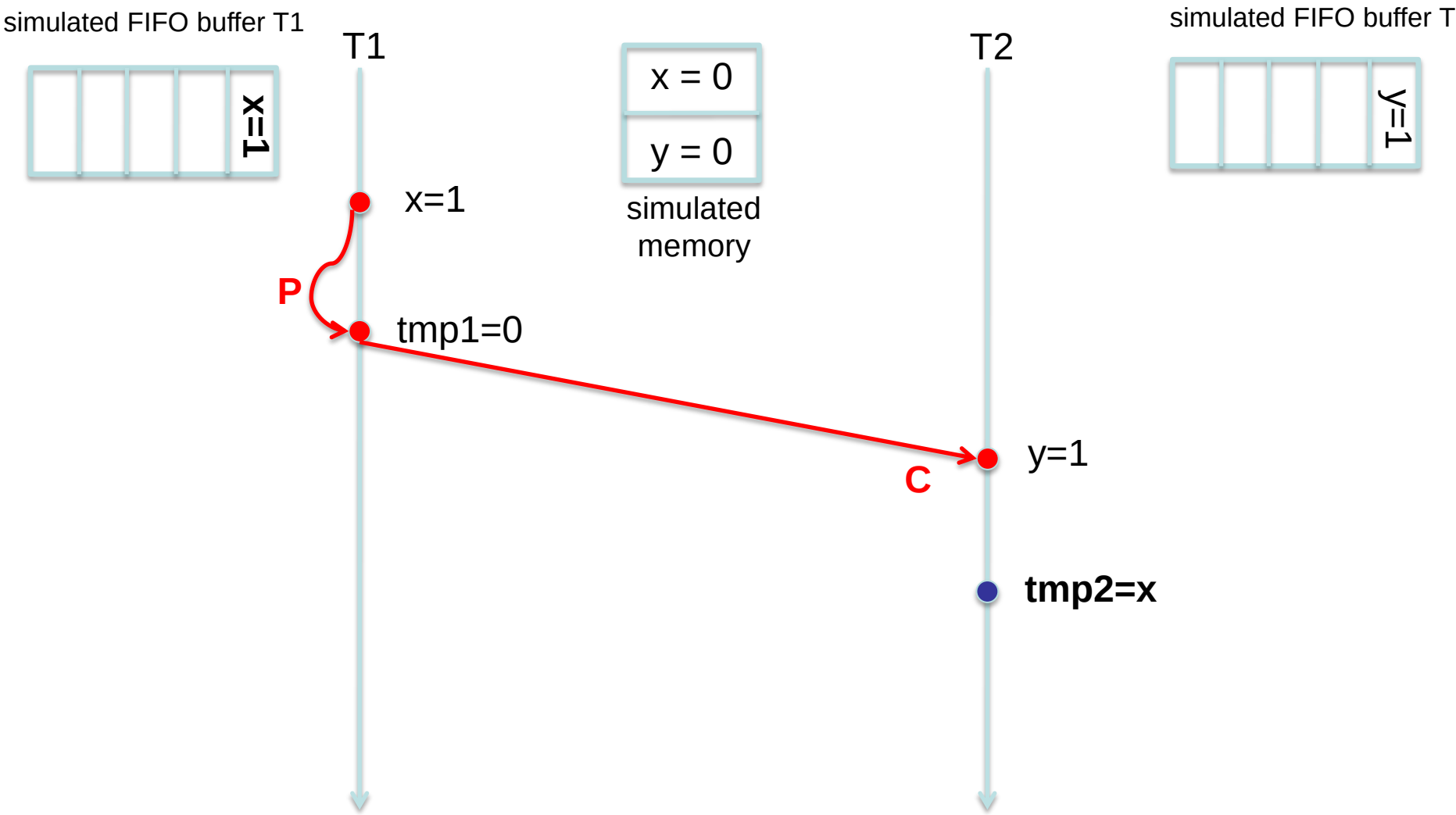
Example


simulated FIFO buffer T1
T1=x
T1
x = 0
y = 0
simulated memory
T2
simulated FIFO buffer T
x=1
P
tmp1=0
C
y=1
tmp2=x

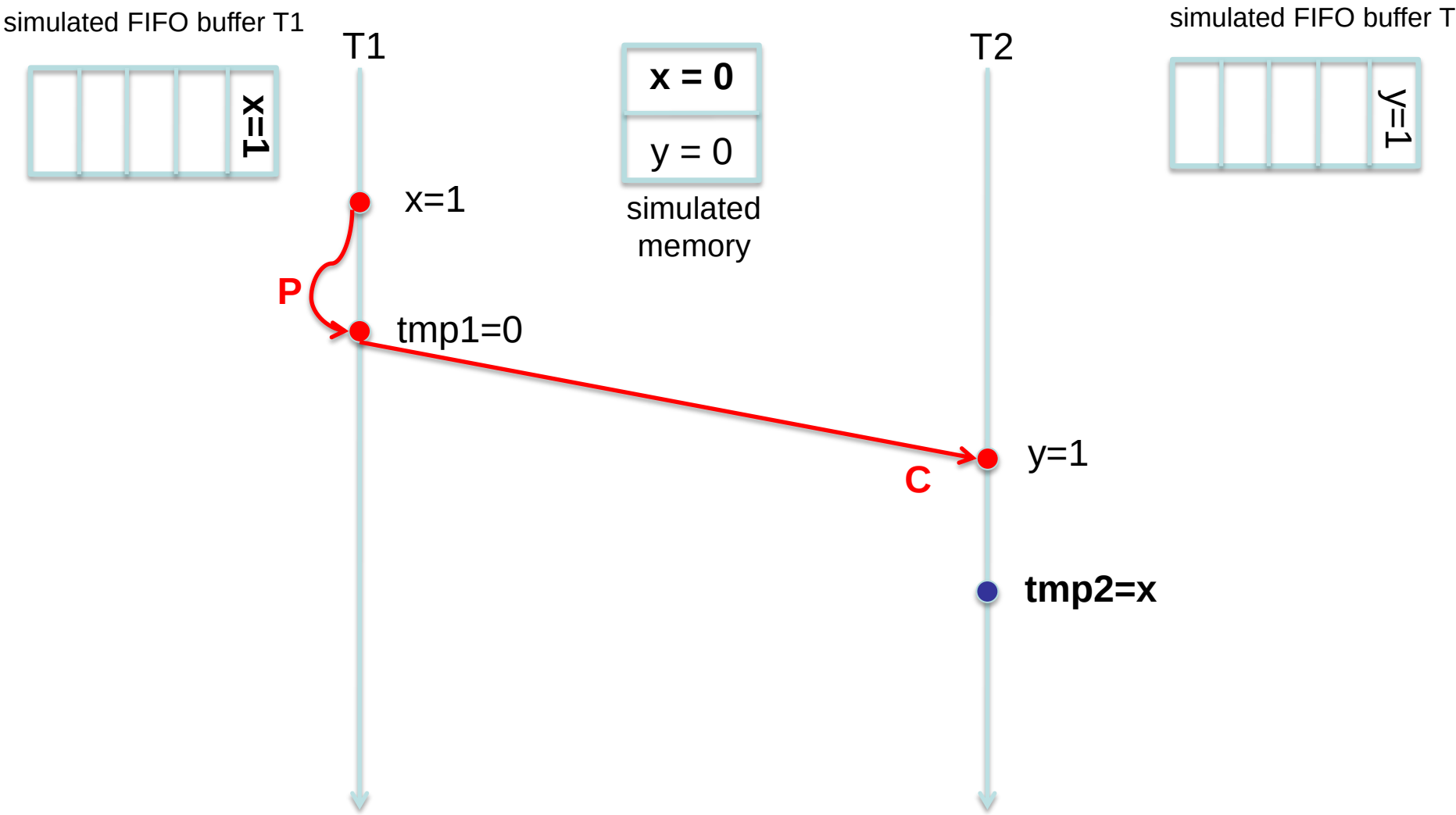
Example


simulated FIFO buffer T1
x=1
T1
x = 0
y = 0
simulated memory
T2
simulated FIFO buffer T
y=1
x=1
P
tmp1=0
C
y=1
tmp2=x

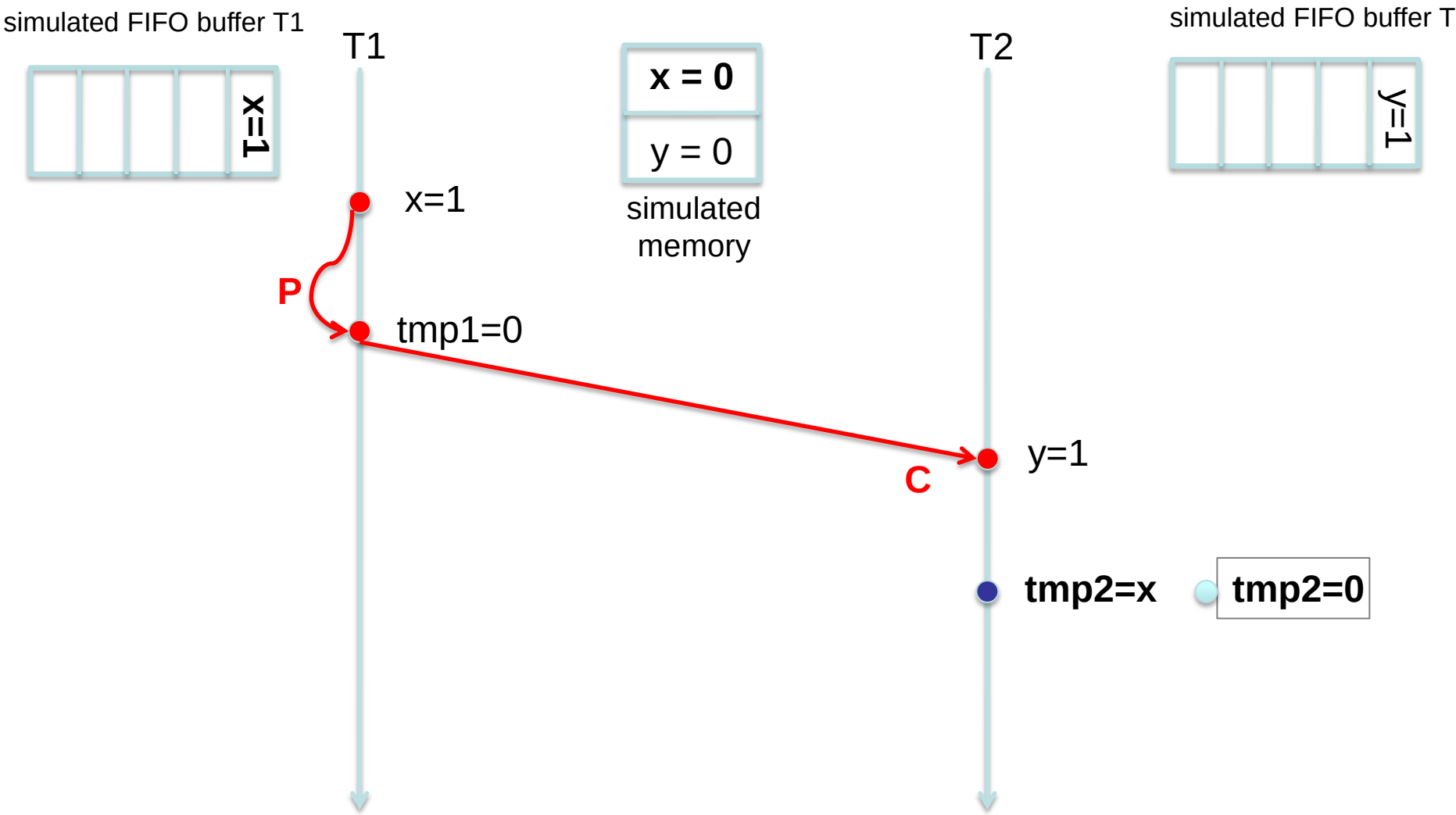
Example


simulated FIFO buffer T1
x=1
T1
x = 0
y = 0
simulated memory
T2
simulated FIFO buffer T
y=1
x=1
P
tmp1=0
C
y=1
tmp2=x

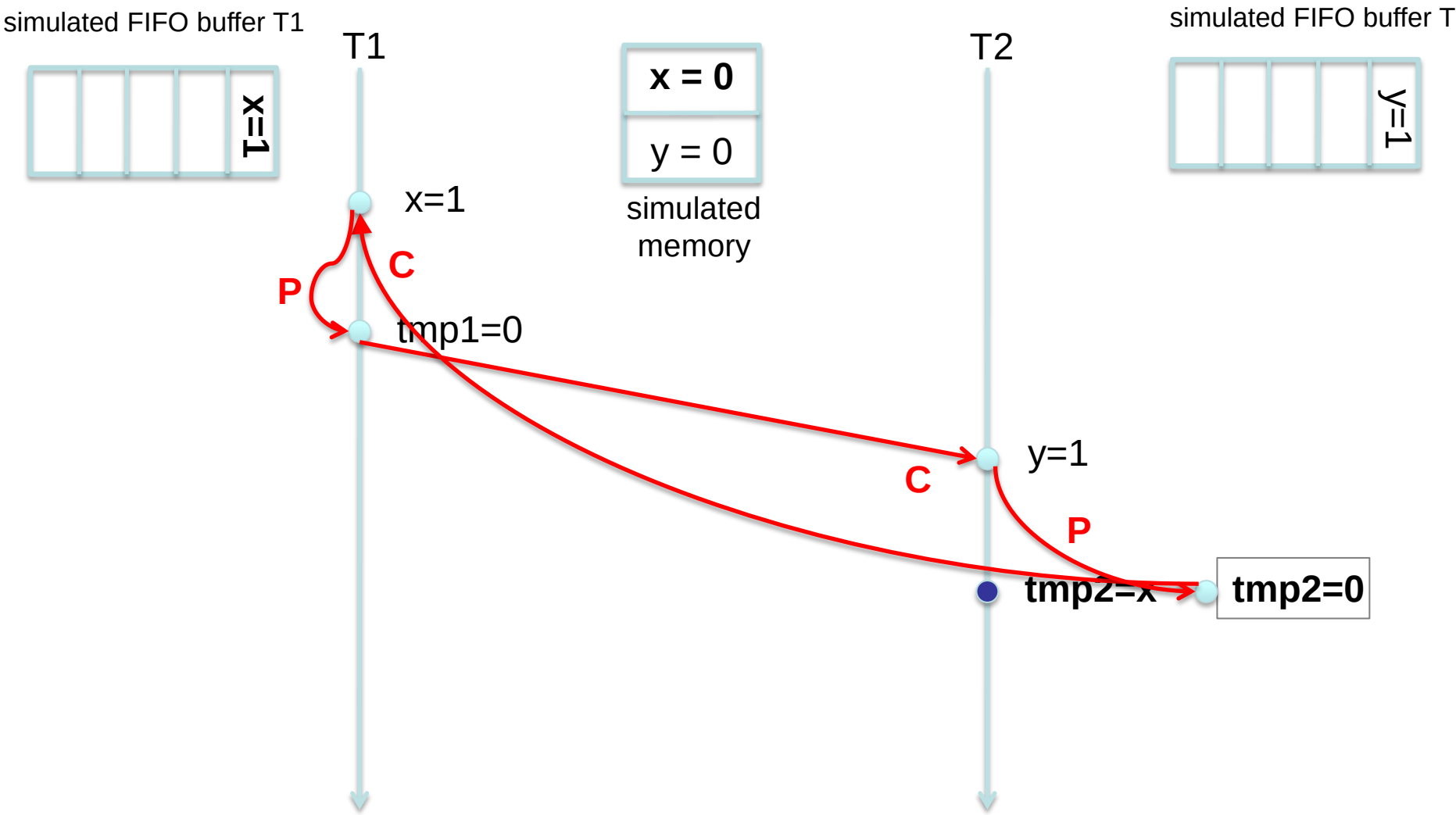
Example


simulated FIFO buffer T1
x=1
T1
x = 0
y = 0
simulated memory
T2
simulated FIFO buffer T
y=1
x=1
P
tmp1=0
C
y=1
tmp2=x

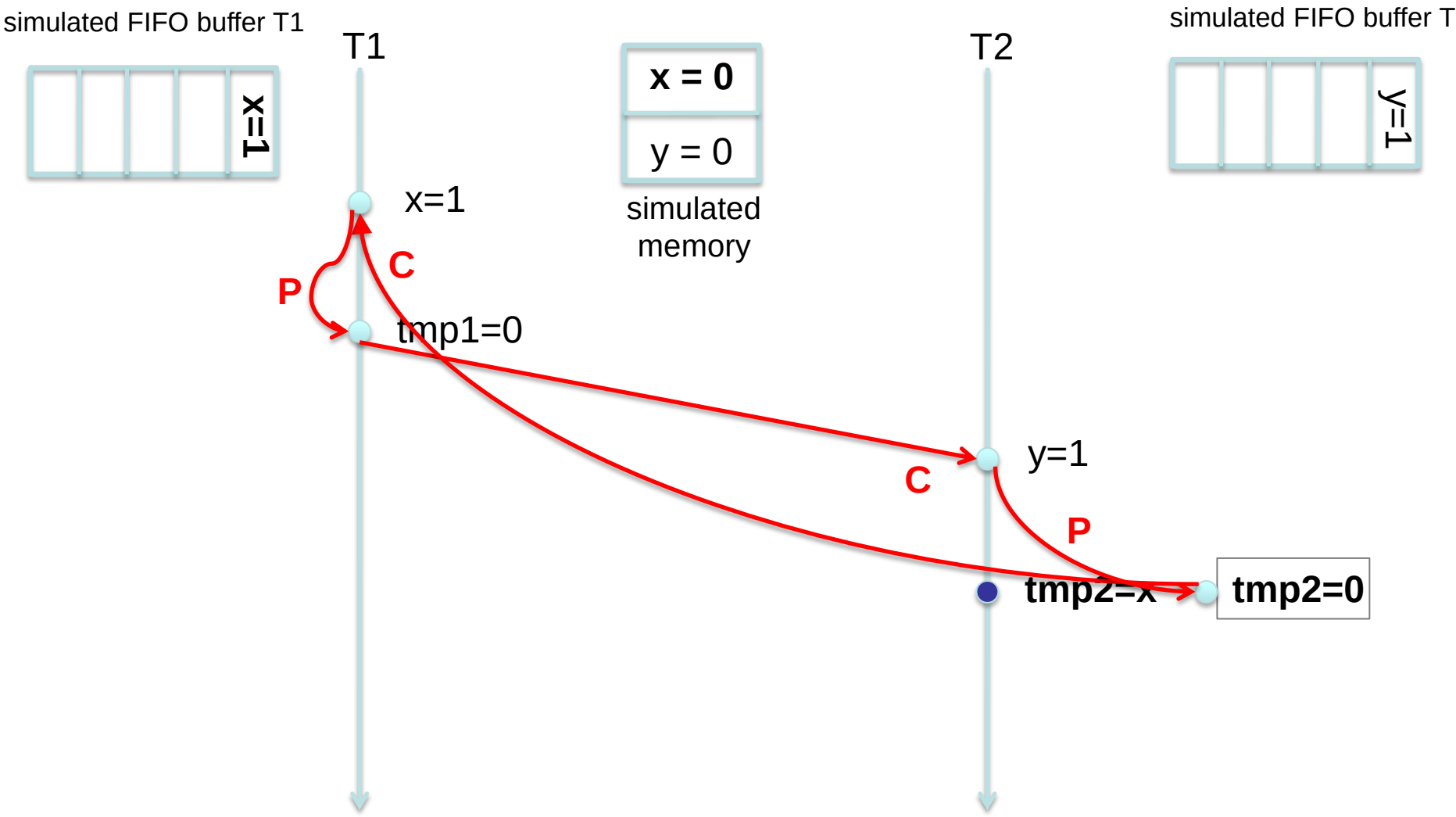
Example


simulated FIFO buffer T1
x=1
T1
x = 0
y = 0
simulated memory
T2
simulated FIFO buffer T
y=1
x=1
P
tmp1=0
C
y=1
tmp2=x
tmp2=0

Example

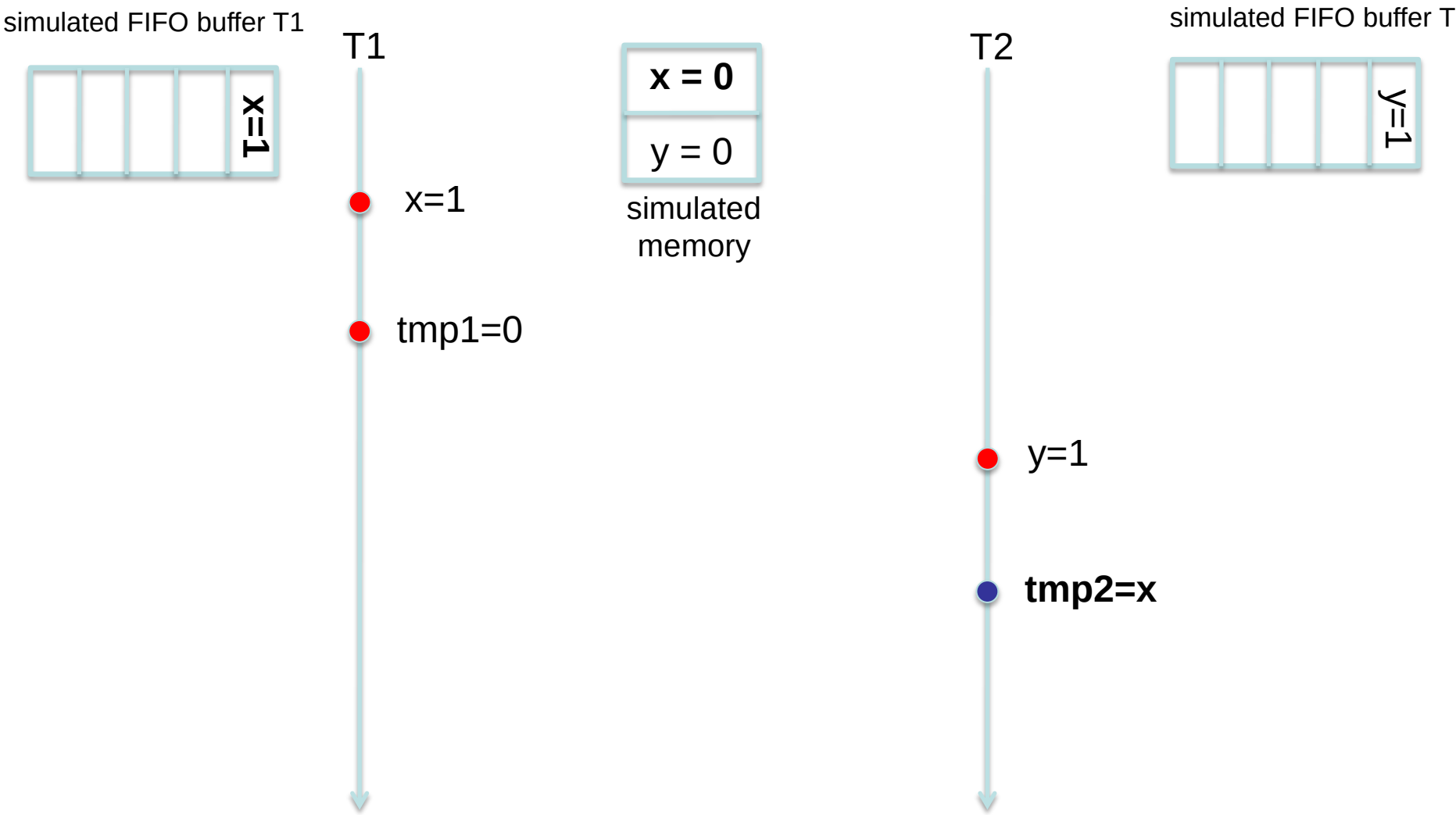

simulated FIFO buffer T1
x=1
T1
x = 0
y = 0
simulated memory
T2
simulated FIFO buffer T
y=1
x=1
P
C
tmp1=0
C
y=1
P
tmp2=x
tmp2=0

Example



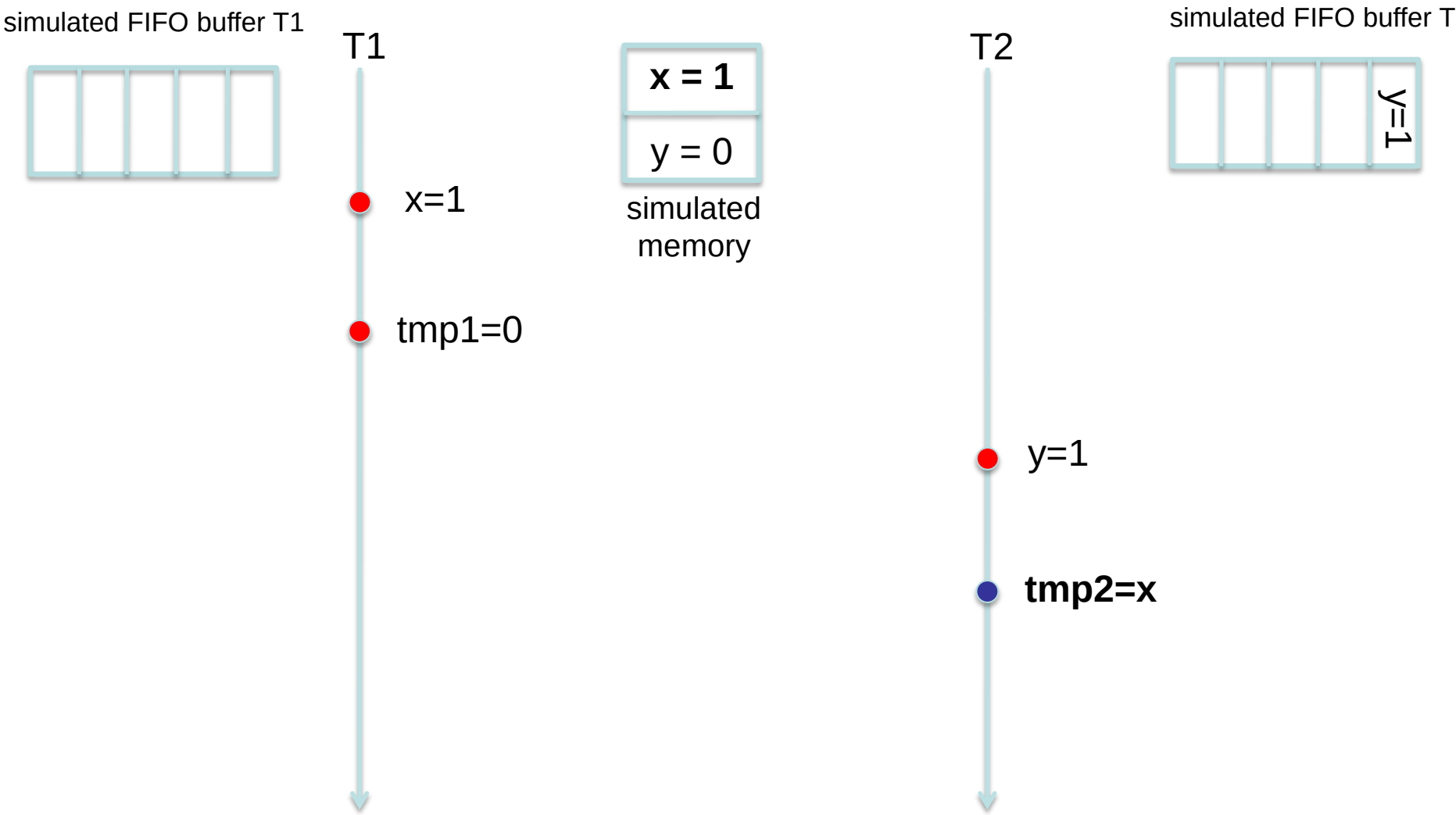
Record Sequential Consistency Violation

Example



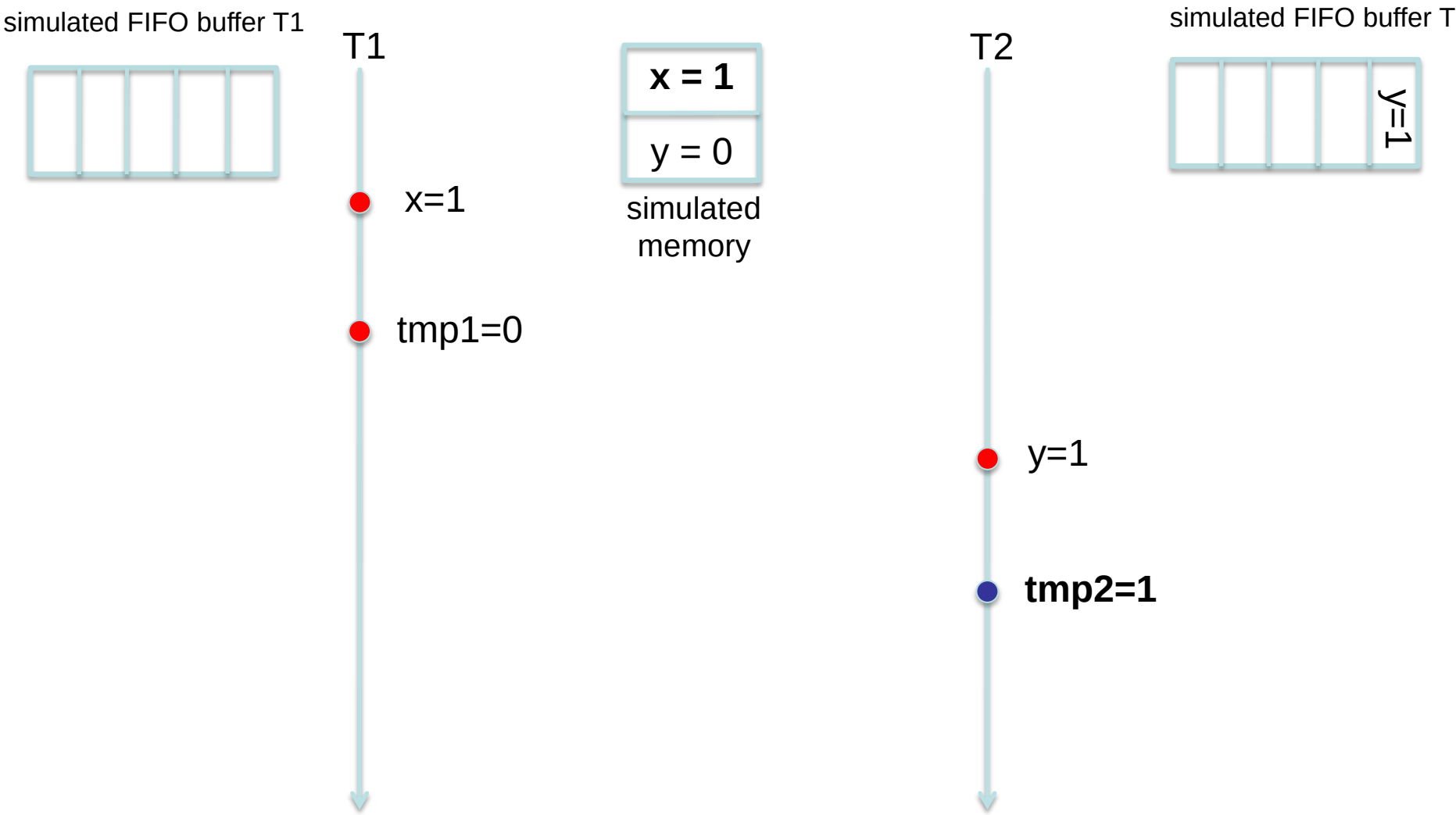
Record Sequential Consistency Violation

Example



Continue Sequential Consistent Execution

Example



Continue Sequential Consistent Execution

Advantages

- In a different execution, re-create violations by actually creating cycle and see if a bug is exposed
- Give back to the programmer actual TSO/PSO execution
- Easier to debug
- Sound
- Complete
- Simple
  - Complexity is pushed to the proofs of soundness and completeness
- Can run on any memory model with operational semantics

Evaluation

- We implemented the monitor algorithms for C programs
- We instrumented programs to capture load, store and compare-and-swap operations
- We used context-bounding model-checking to generate sequentially consistent traces
- We evaluated on seven benchmarks:
  - 2 mutual exclusion algorithms
  - 5 lock free data structures

Results

	LOC	# SC schedules	TSO cycles	TSO bugs	PSO cycles	PSO bugs
dekker	23	220	3	2	5	2
bakery	31	1434	3	1	4	1
msn	83	616	0	-	3	3
ms2	78	500	0	-	2	1
lazylist	155	1764	0	-	2	1
harris	121	802	0	-	4	2
snark	150	1208	0	-	4	0

Questions?