Objects in Oz

Martin Henz

Universität des Saarlandes
Saarbrücken, Germany

http://www.ps.uni-sb.de/~henz

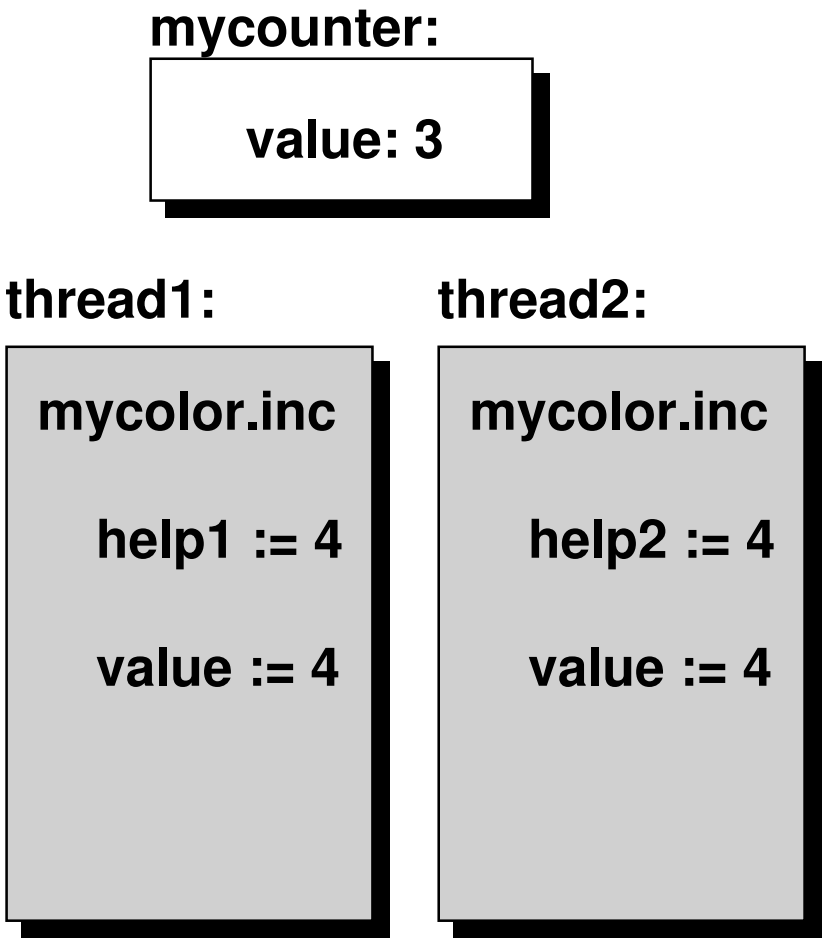
Overview

1. Some issues in concurrent objects
2. Concurrent constraint programming
3. Active and passive concurrent objects
4. Objects in Oz
5. Encapsulated Search
6. Application Scenario

1. Some issues in concurrent objects

Example 1: A Concurrent Counter

```
class counter
    attr value: 0
    meth inc
        help: int
        help := counter + 1
        value := help
    end
end
```



Concurrent Counter

Issue: Access control to shared data

Example 2: Buffer

writing thread

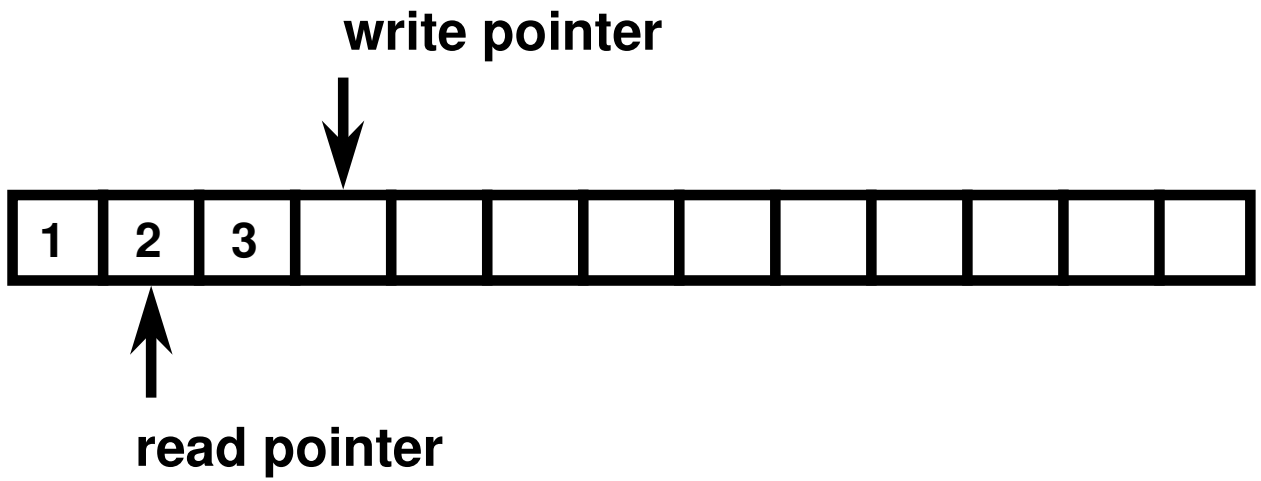
```
mybuffer.enq(1)

mybuffer.enq(2)

mybuffer.enq(3)
```

reading thread

```
mybuffer.enq(x)
```

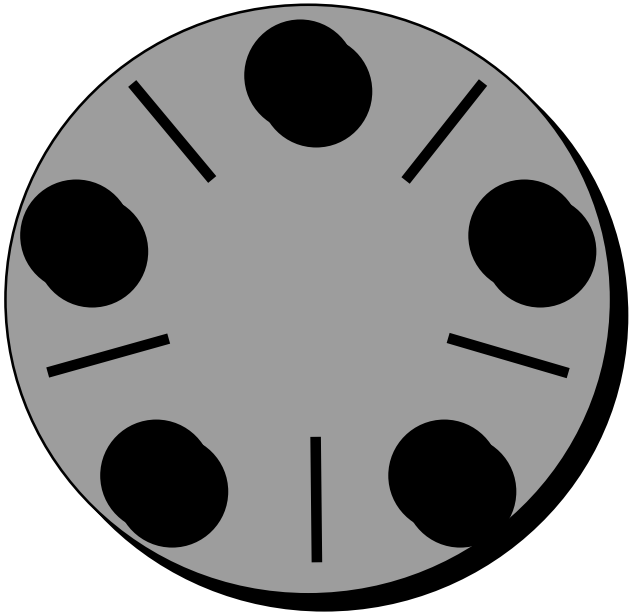


Buffer

Issue: flow control

read pointer may not catch up with write pointer

Example 3: Dining Philosophers



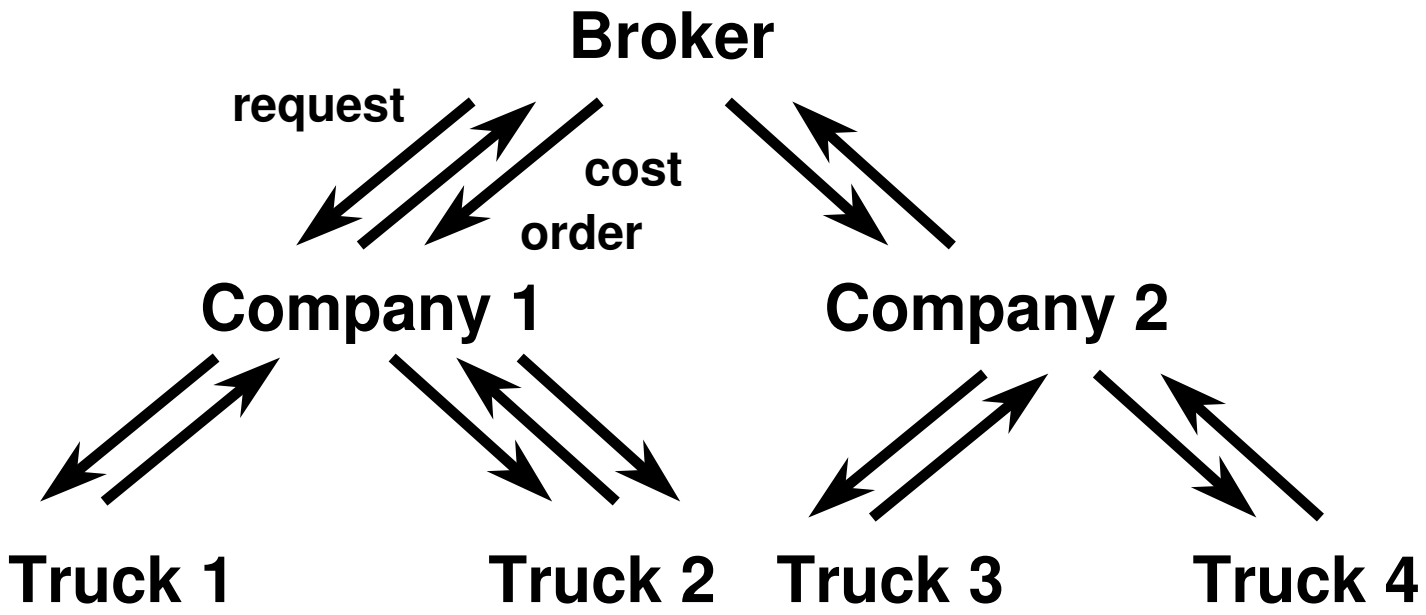
[Dijkstra 68]

Dining Philosophers

Issue: flow control

no neighboring philosophers
may be eating at the same time

Example 4: Freight Transportation



contract net protocol [Smith 80]
cooperative transportation scheduling
[Fischer, Müller, Pischel 94]

Freight Transportation

Issue: Flow control

companies and broker must
wait for all answers

Observation

Syncronization of concurrent objects comes in two distinct flavors:

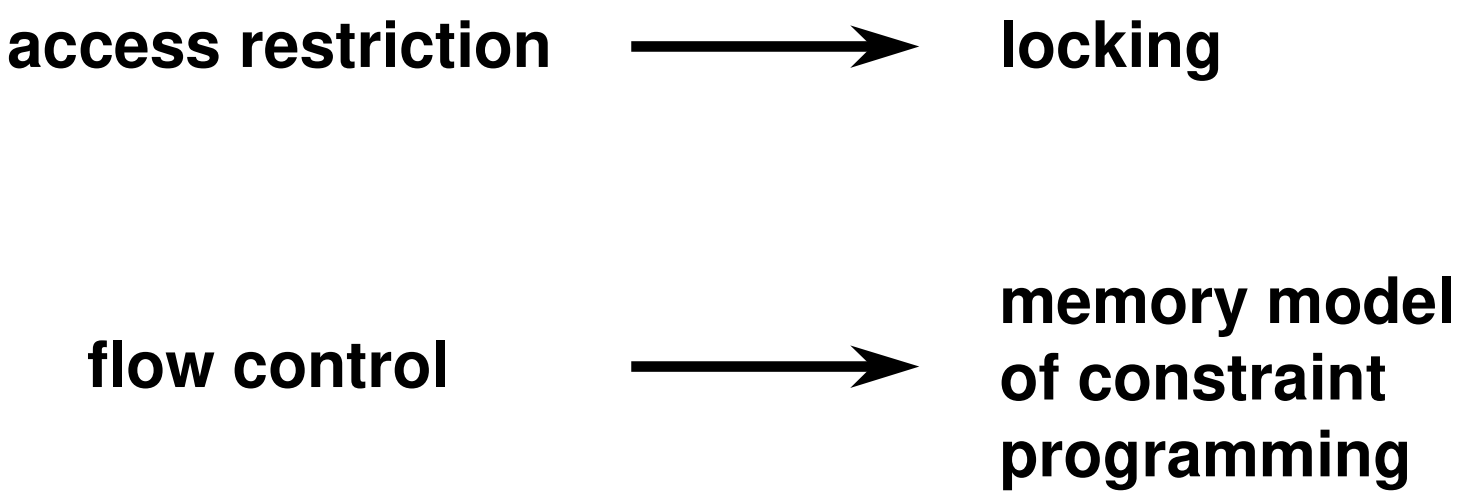
- **Access restriction to shared data**
- **Flow control**

Synchronization Code

methods are equipped with
special code to provide for
synchronization

ABCL [Yonezawa 86]
Actor languages [Lieberman 87]
Pool/T [America 87]
Demeter [Videira Lopez, Lieberherr 94]

Separation of Concerns



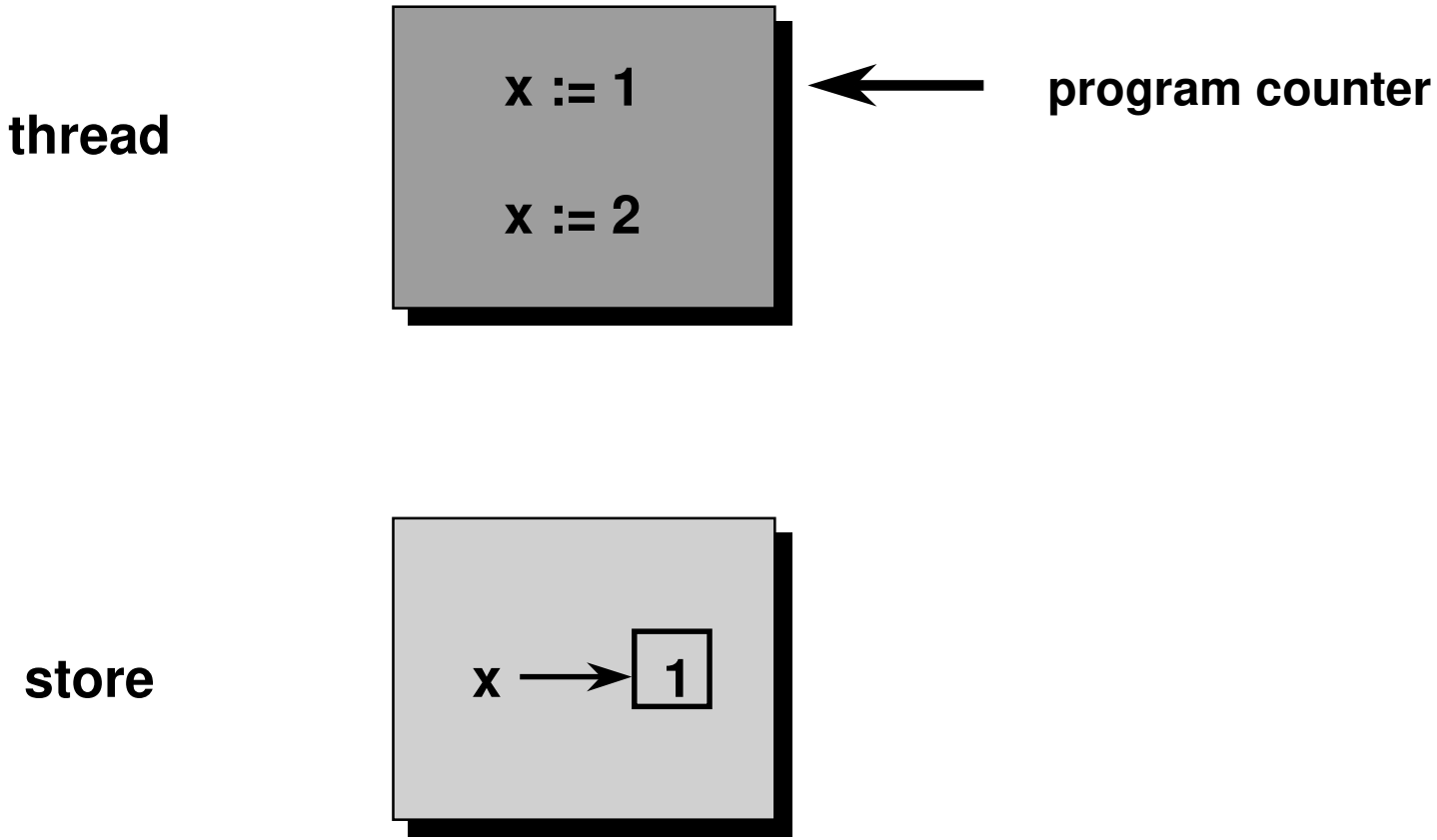
2. Concurrent Constraint Programming

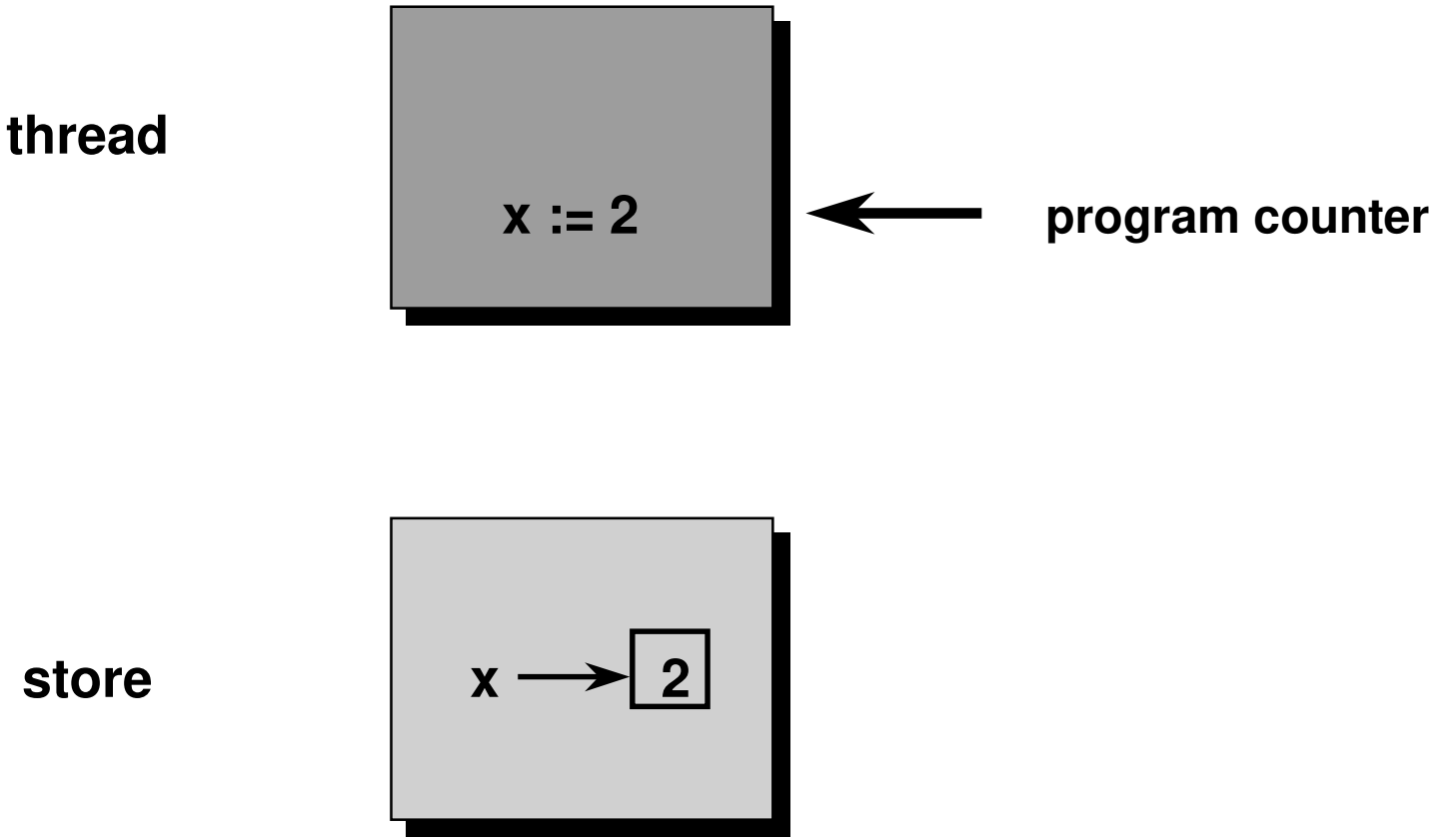
Memory Model of Constraint Programming

- based on monotonic increase of information
- synchronization on availability of information

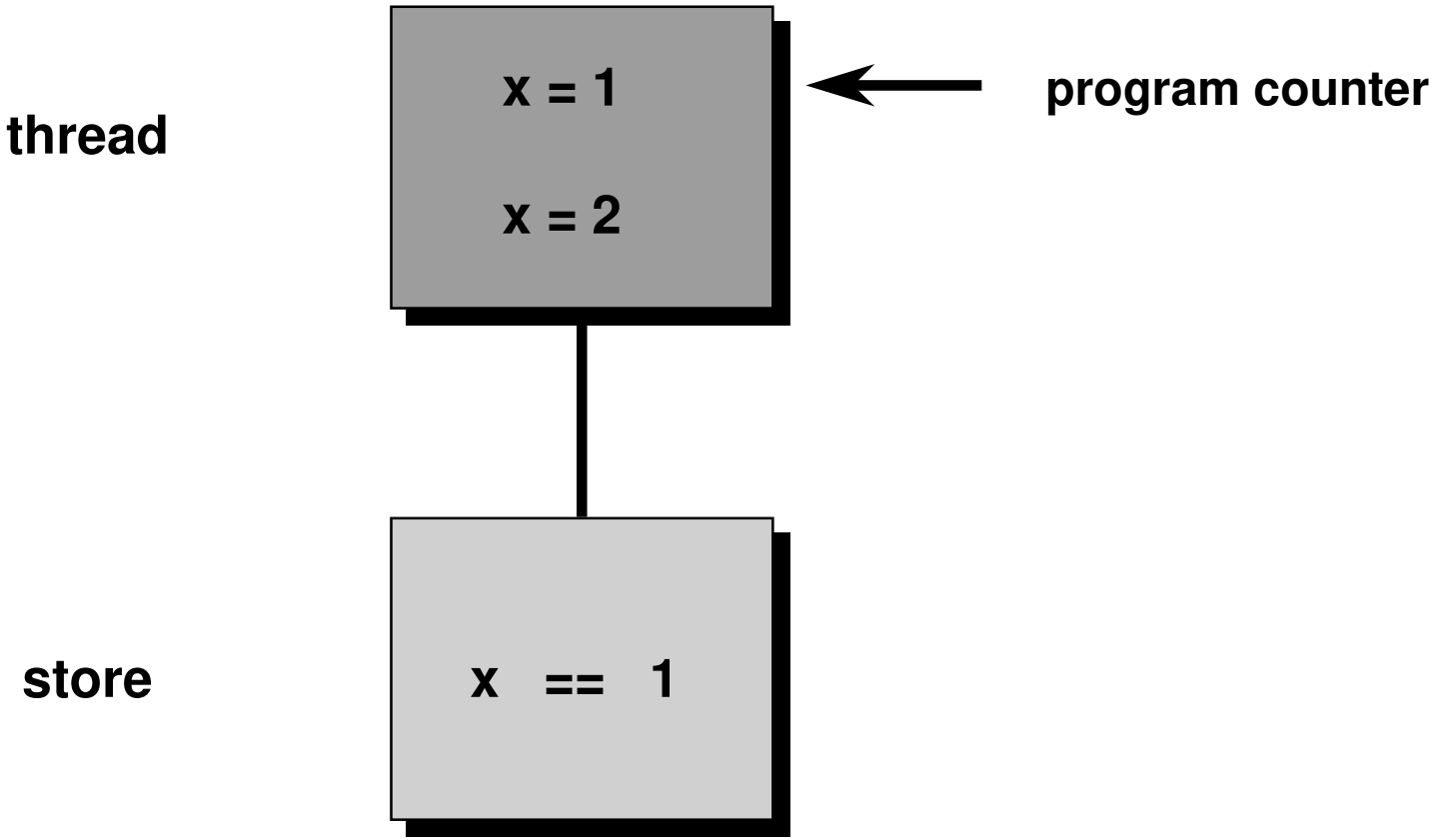
Constraint Logic Programming [Jaffar, Lassez 87]
Concurrent Constraint Programming [Saraswat 89]

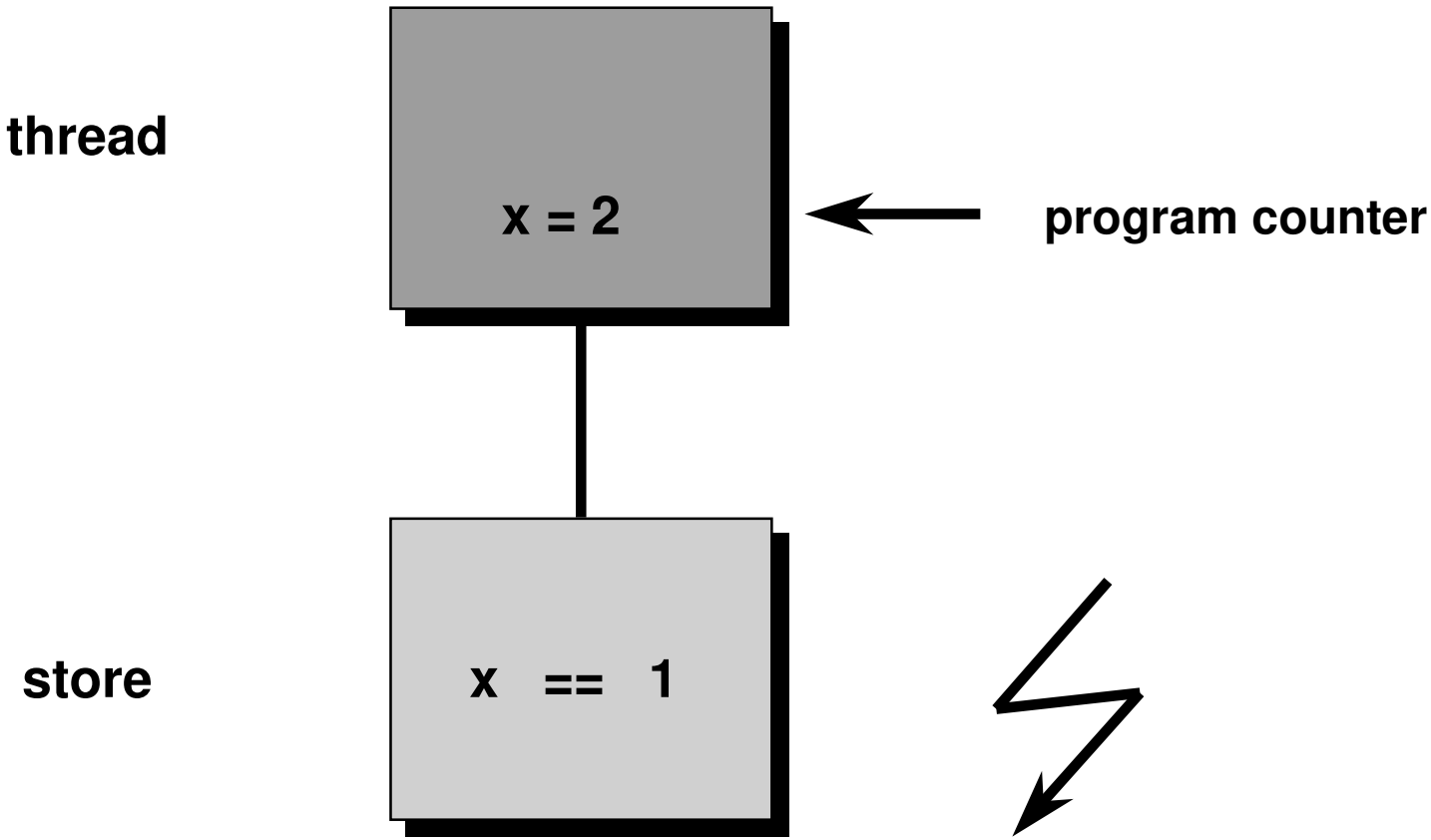
Random Access Memory




thread
x := 2
program counter
store
x
2

Constraint Store

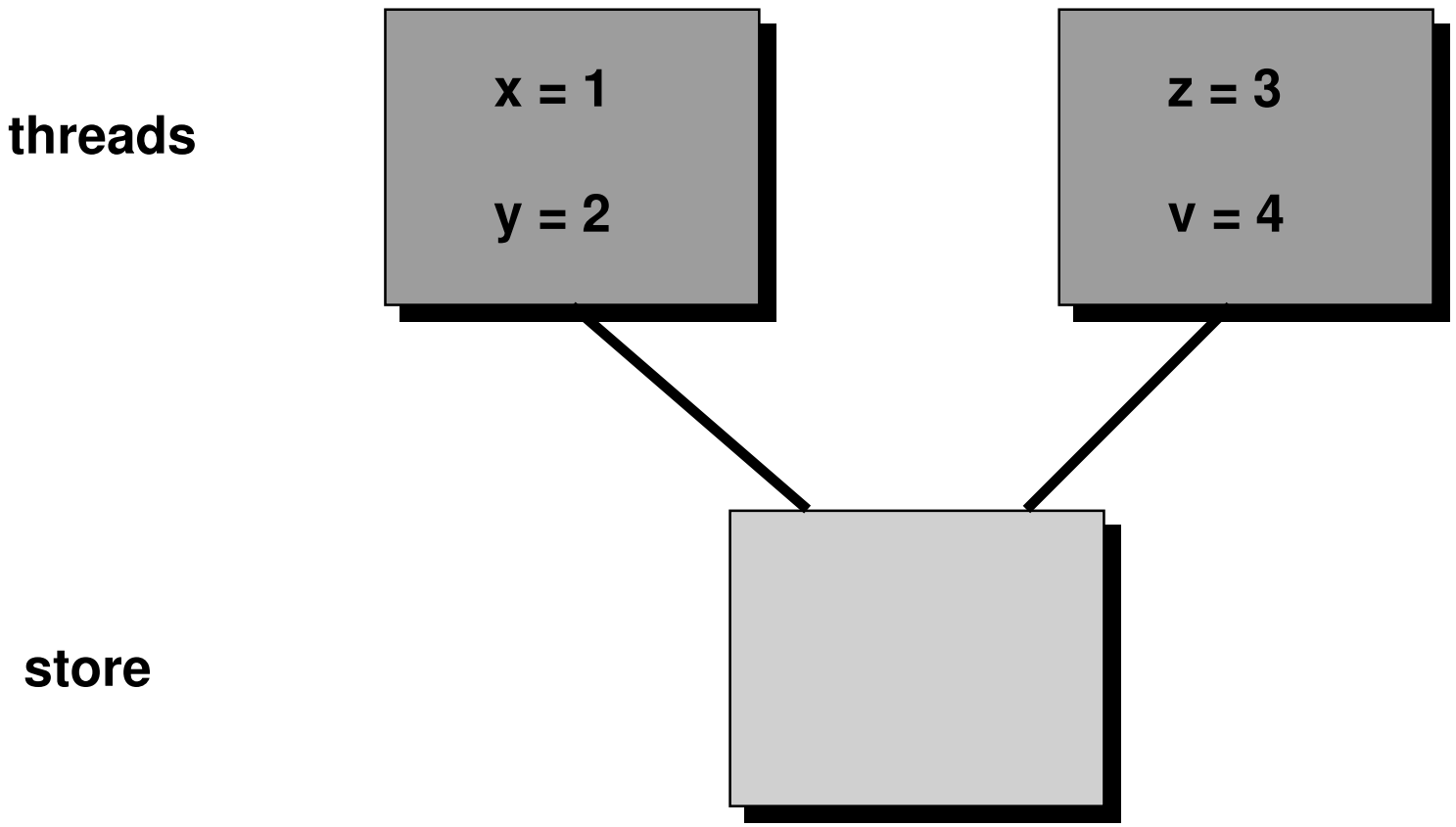


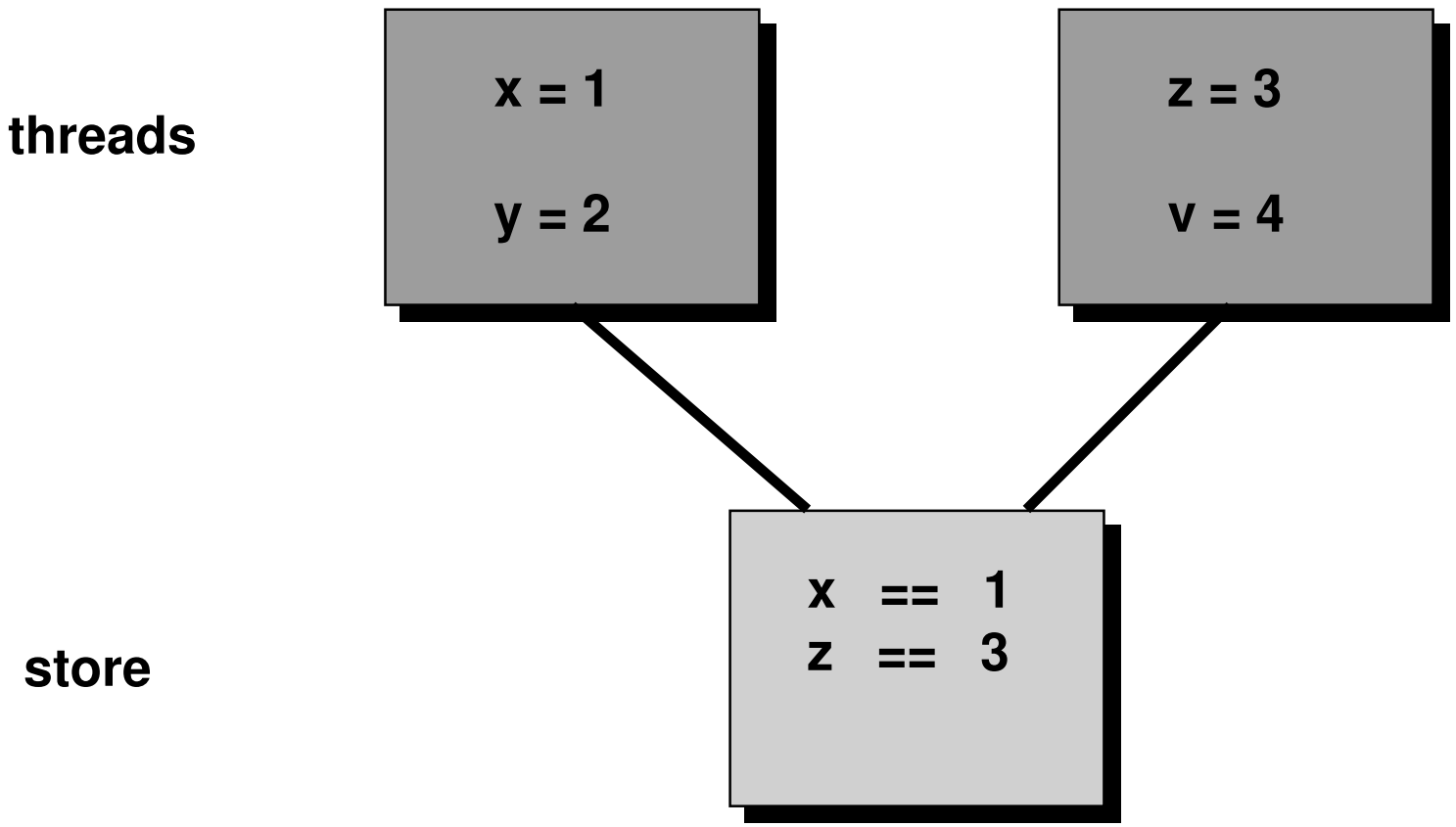

thread
x = 2
program counter
store
x == 1

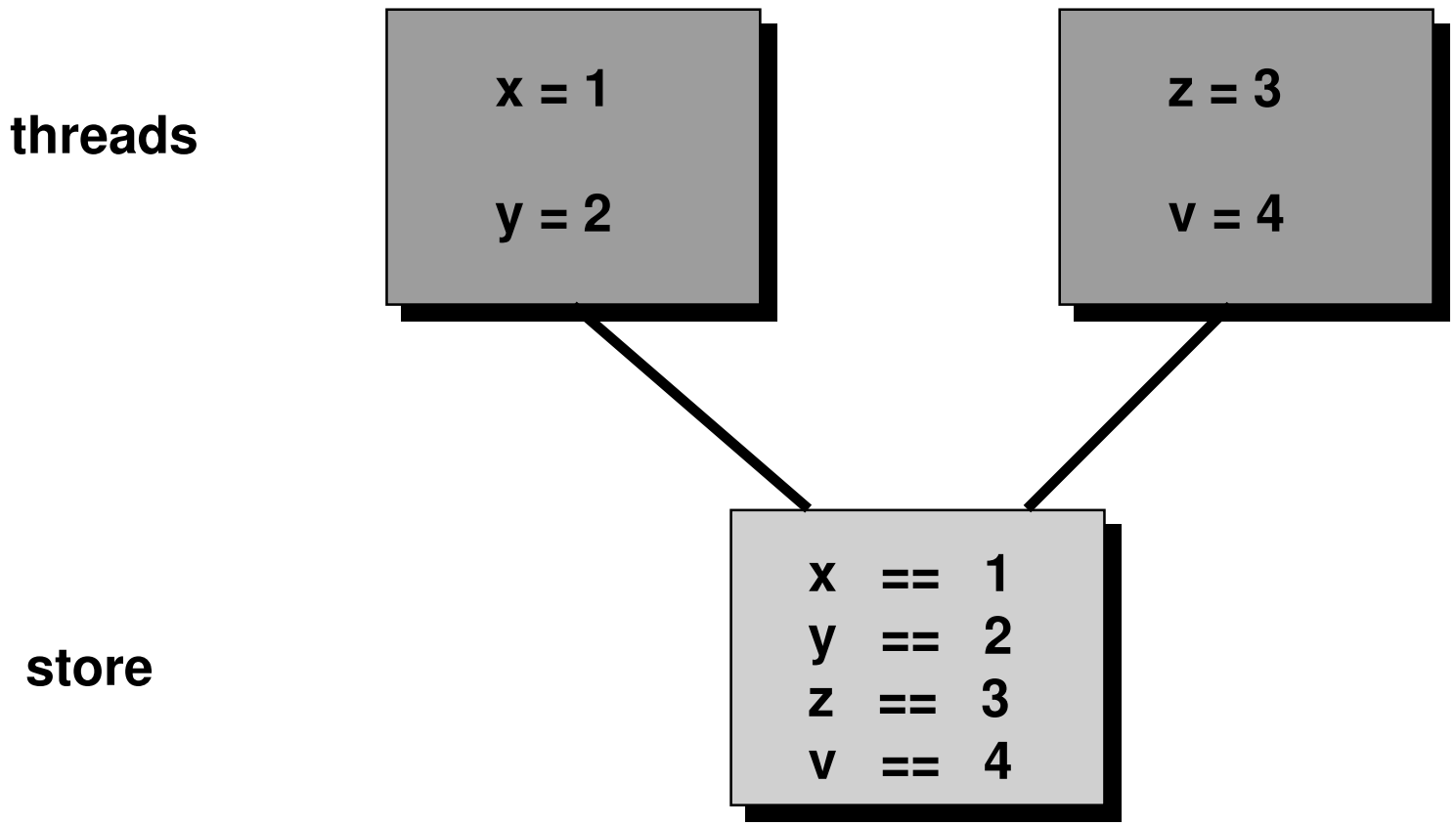
Constraint Store

- no update
- monotonic increase of information

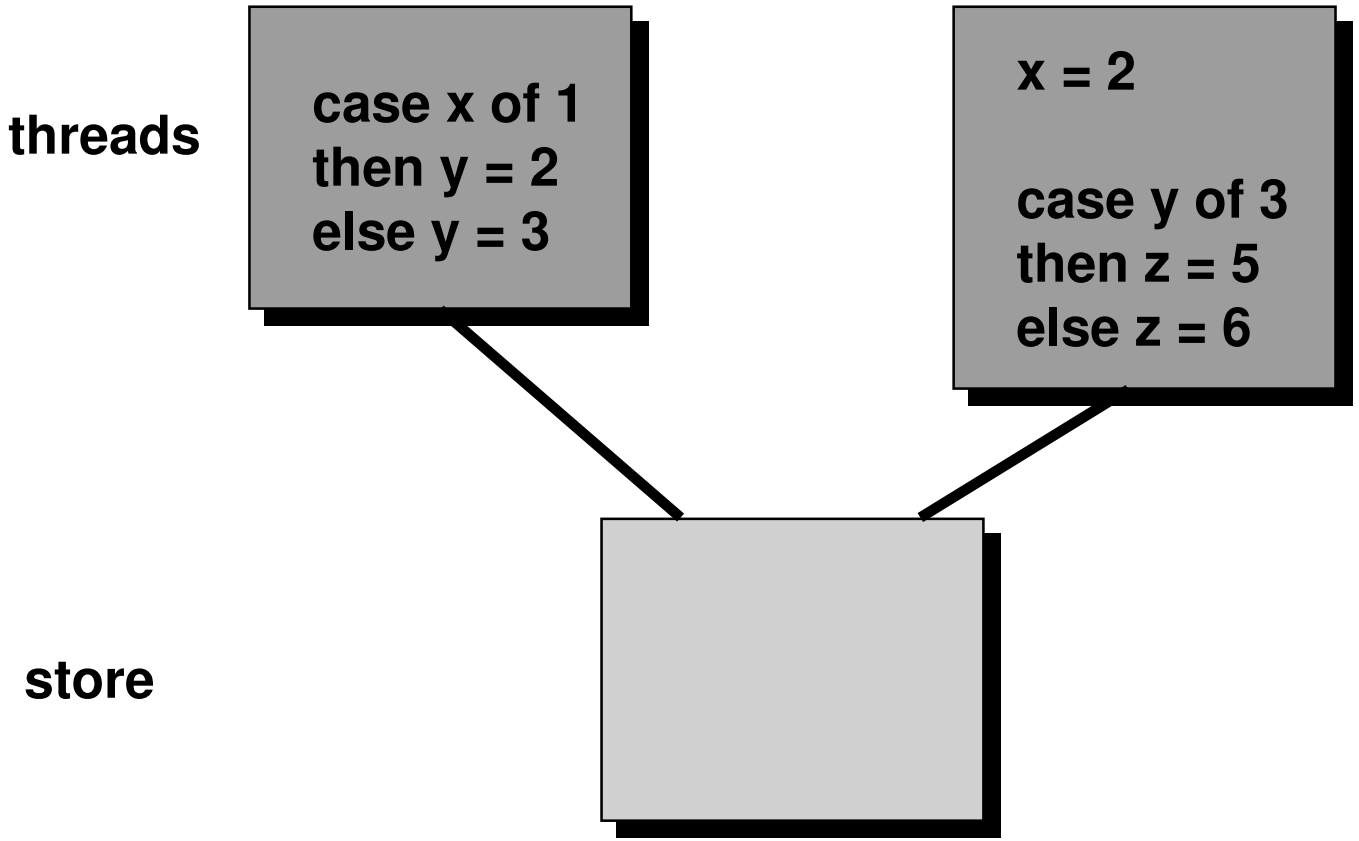
Threads

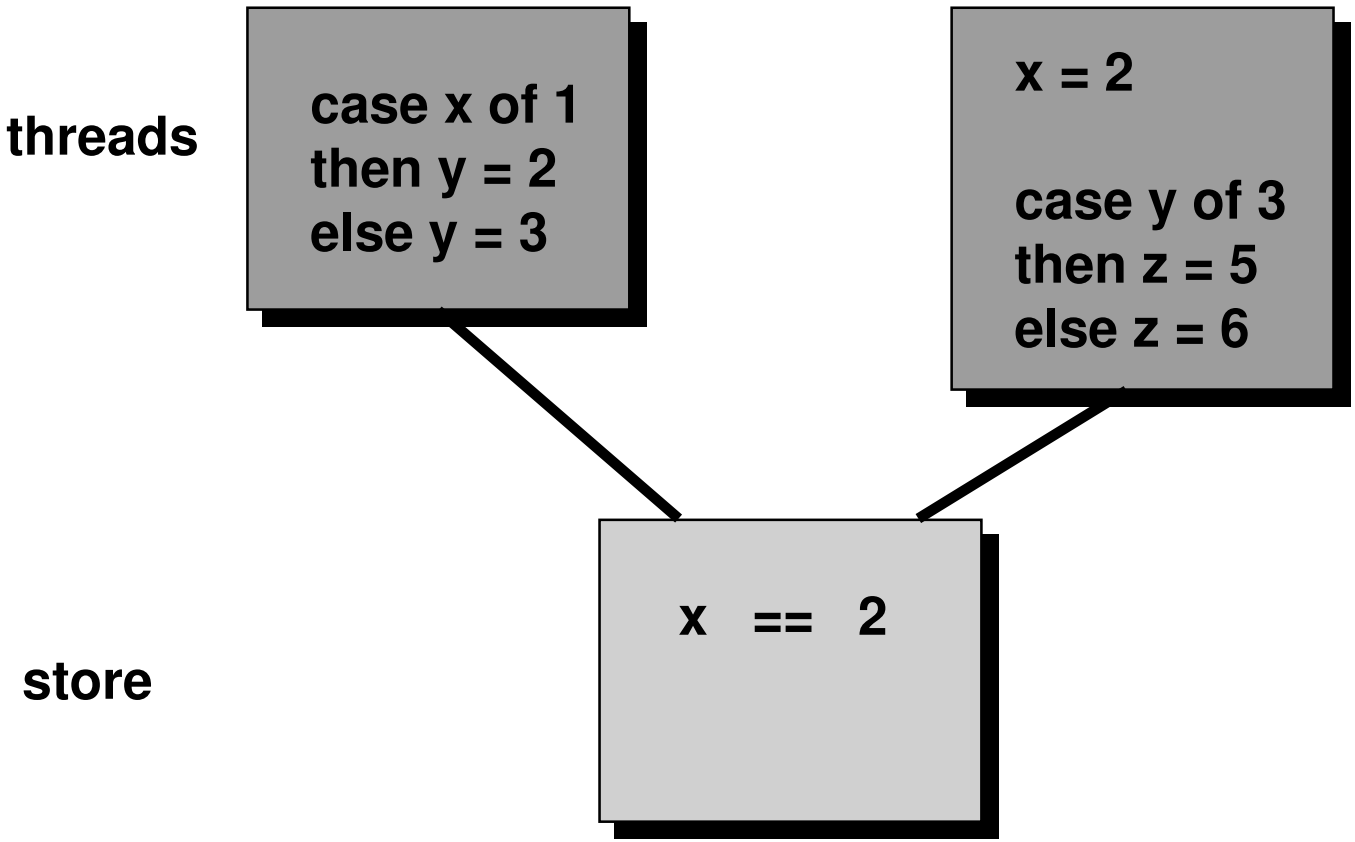


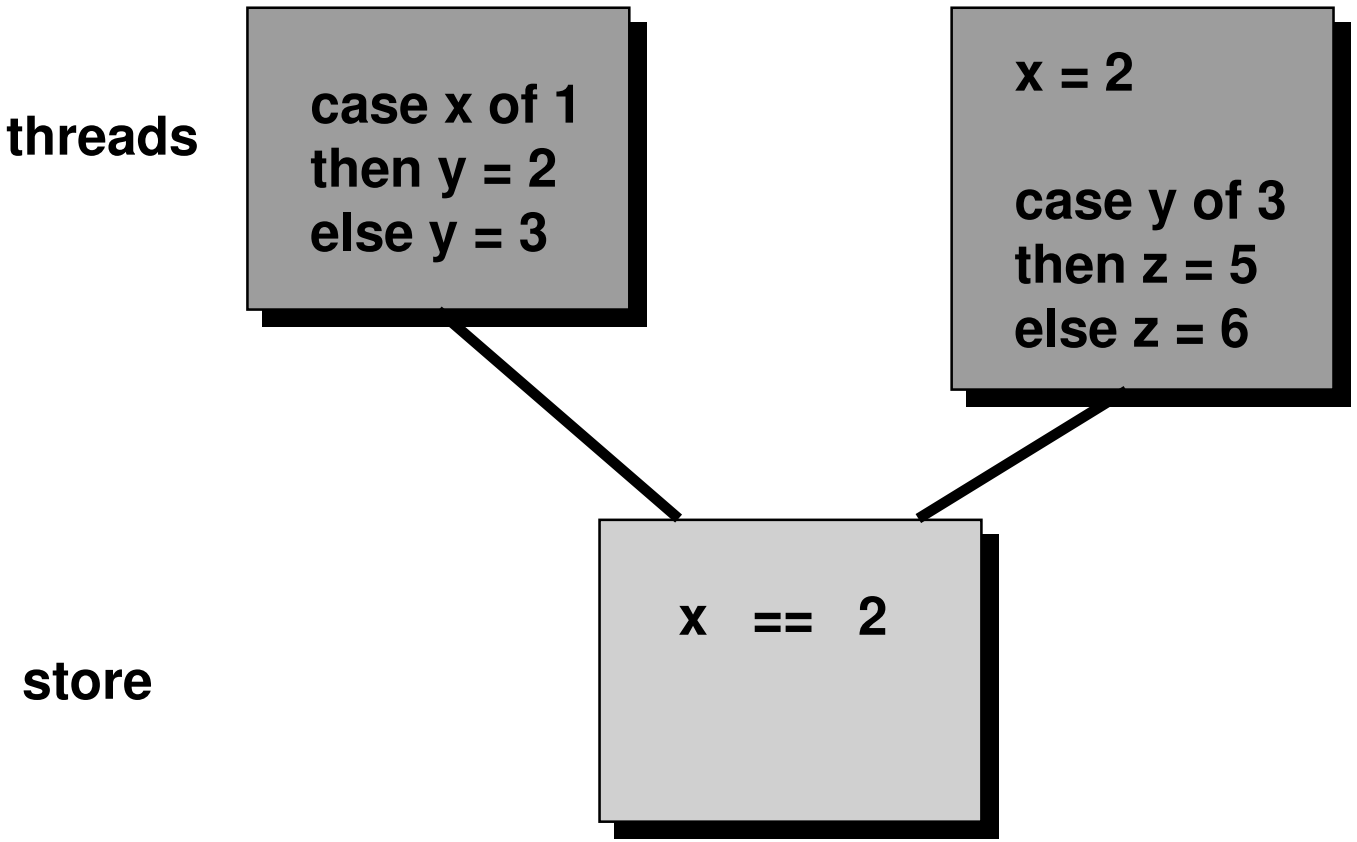

threads
x = 1
y = 2
z = 3
v = 4
store
x == 1
z == 3

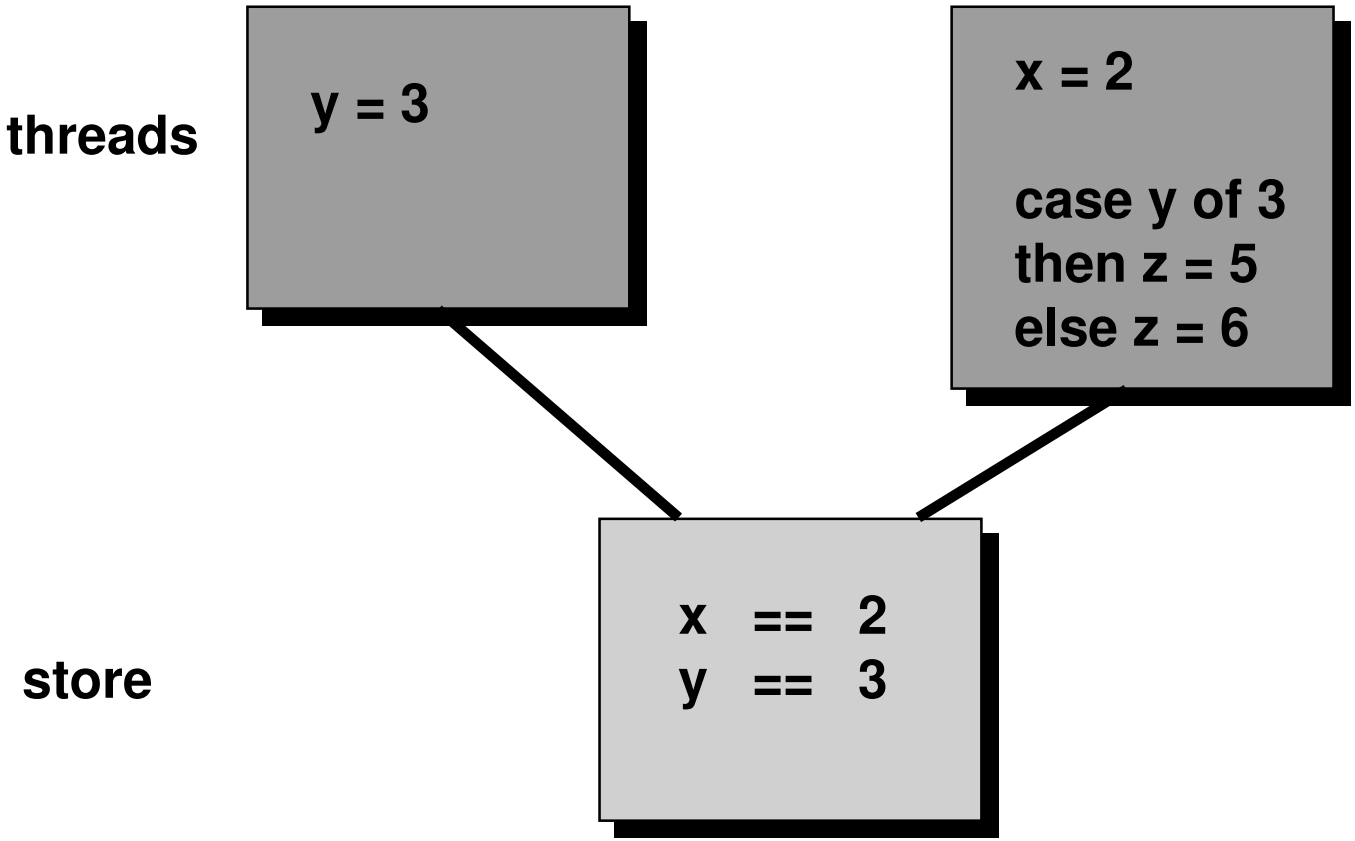

threads
x = 1
y = 2
z = 3
v = 4
store
x == 1
y == 2
z == 3
v == 4

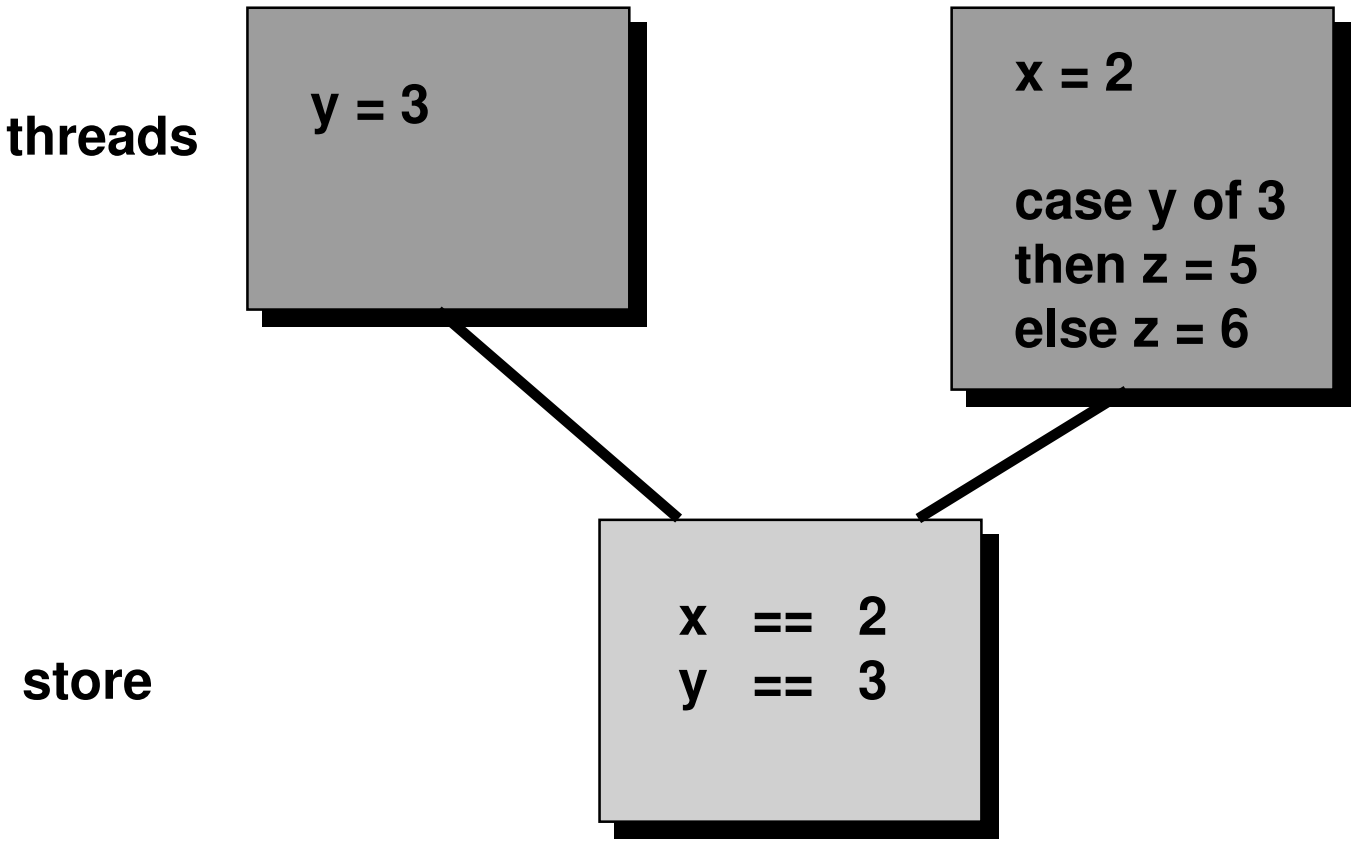
Flow Control through Constraint Store

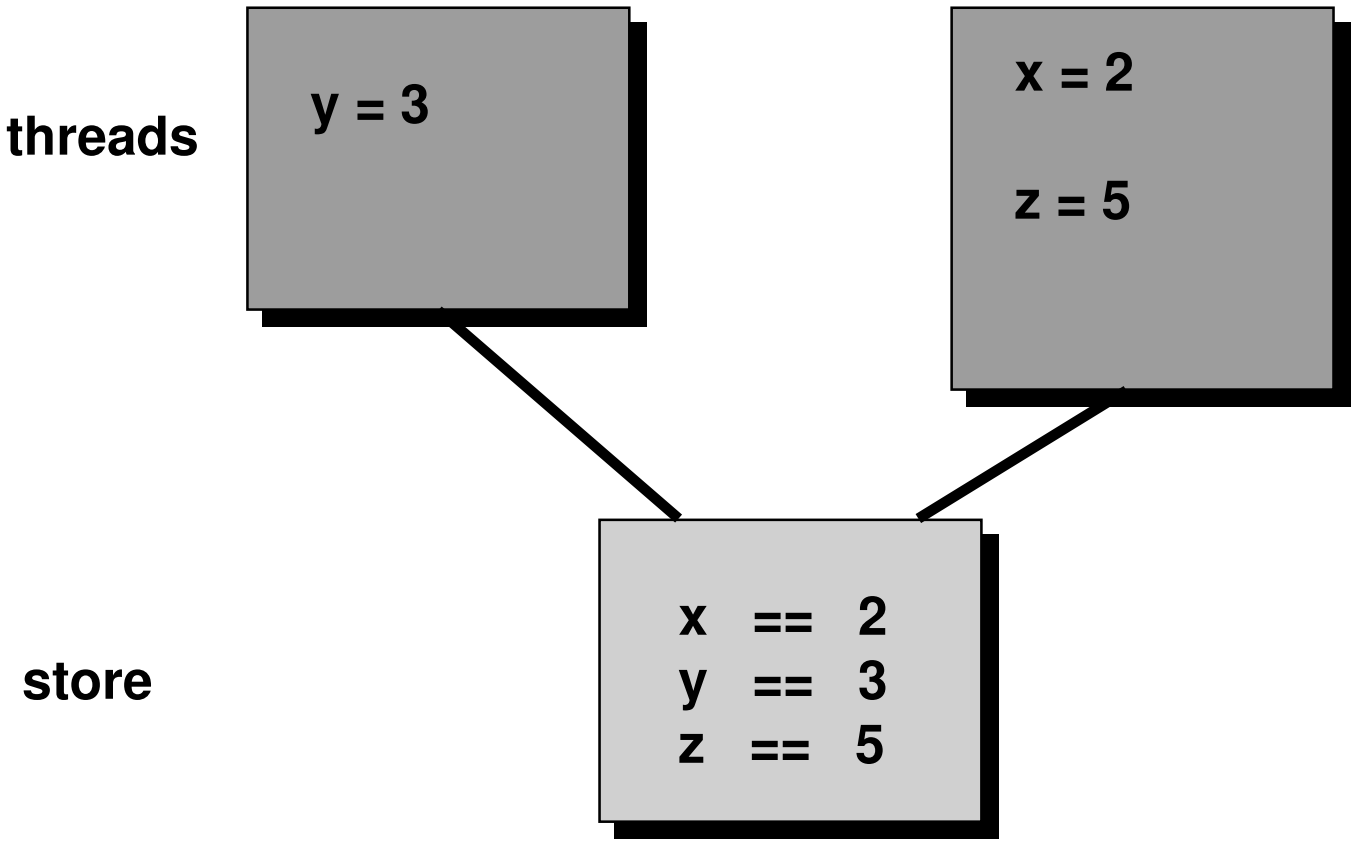



threads
case x of 1
then y = 2
else y = 3
x = 2
case y of 3
then z = 5
else z = 6
x == 2
store

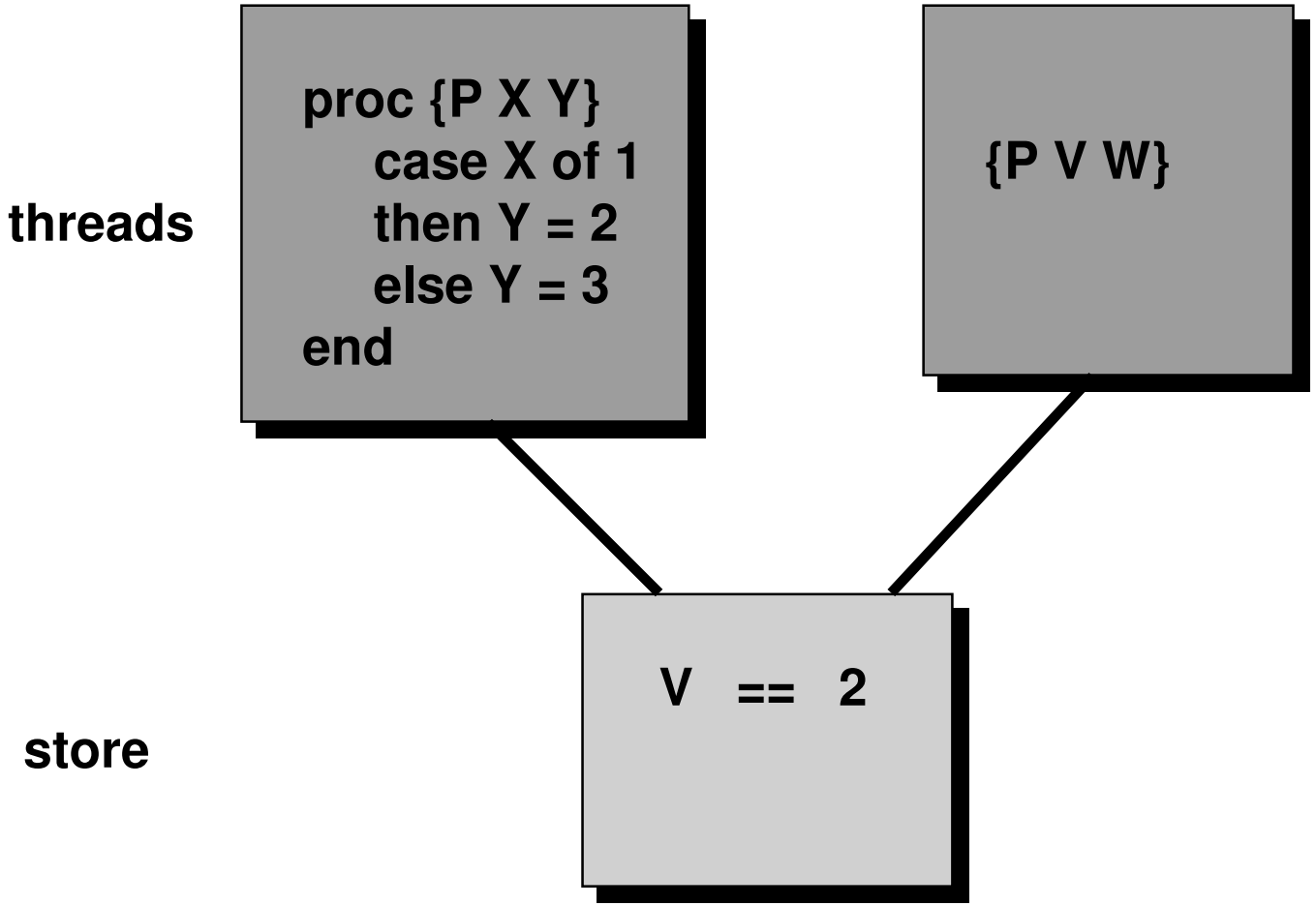

threads
case x of 1
then y = 2
else y = 3
x = 2
case y of 3
then z = 5
else z = 6
x == 2
store

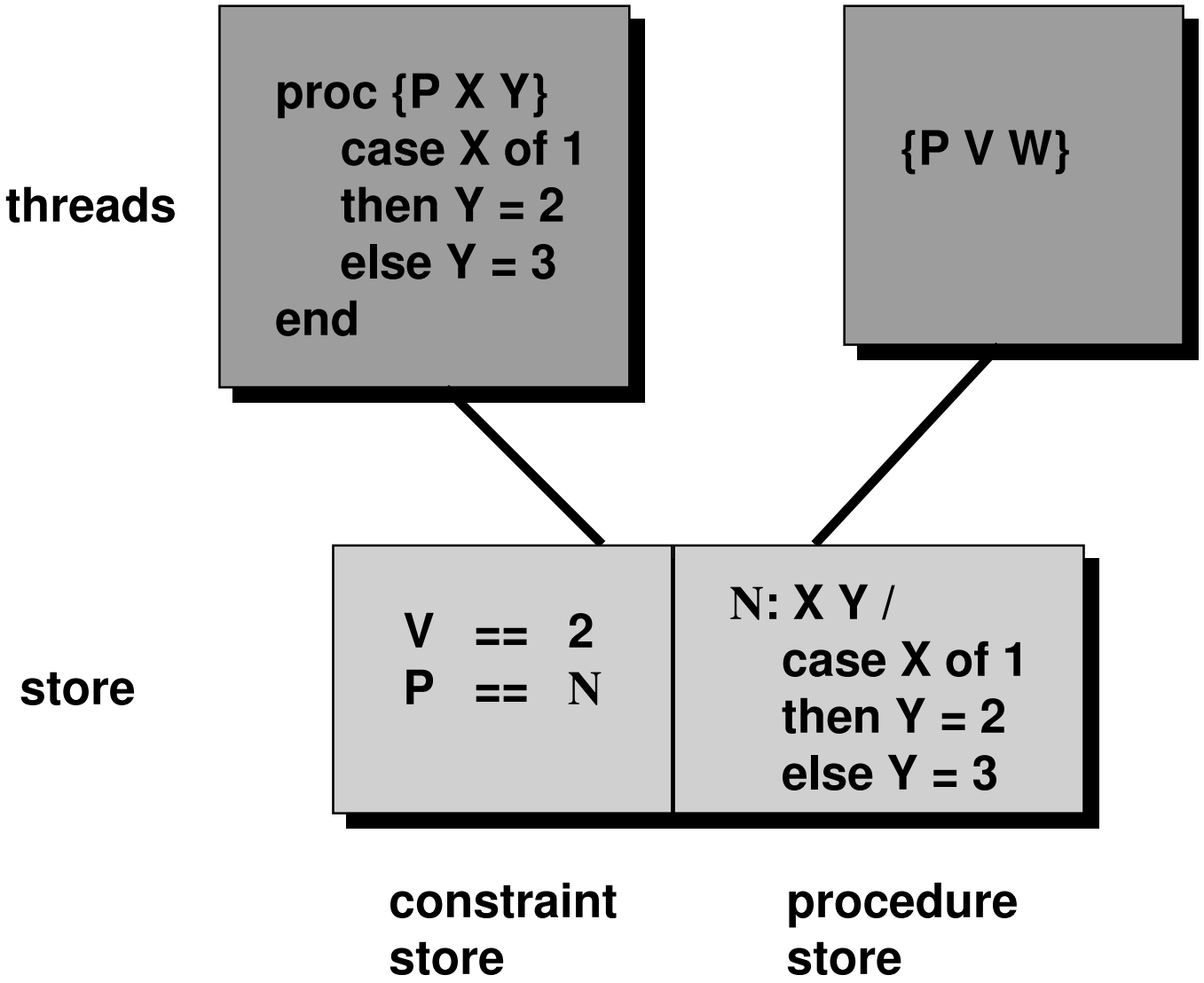

threads
y = 3
x = 2
case y of 3
then z = 5
else z = 6
store
x == 2
y == 3

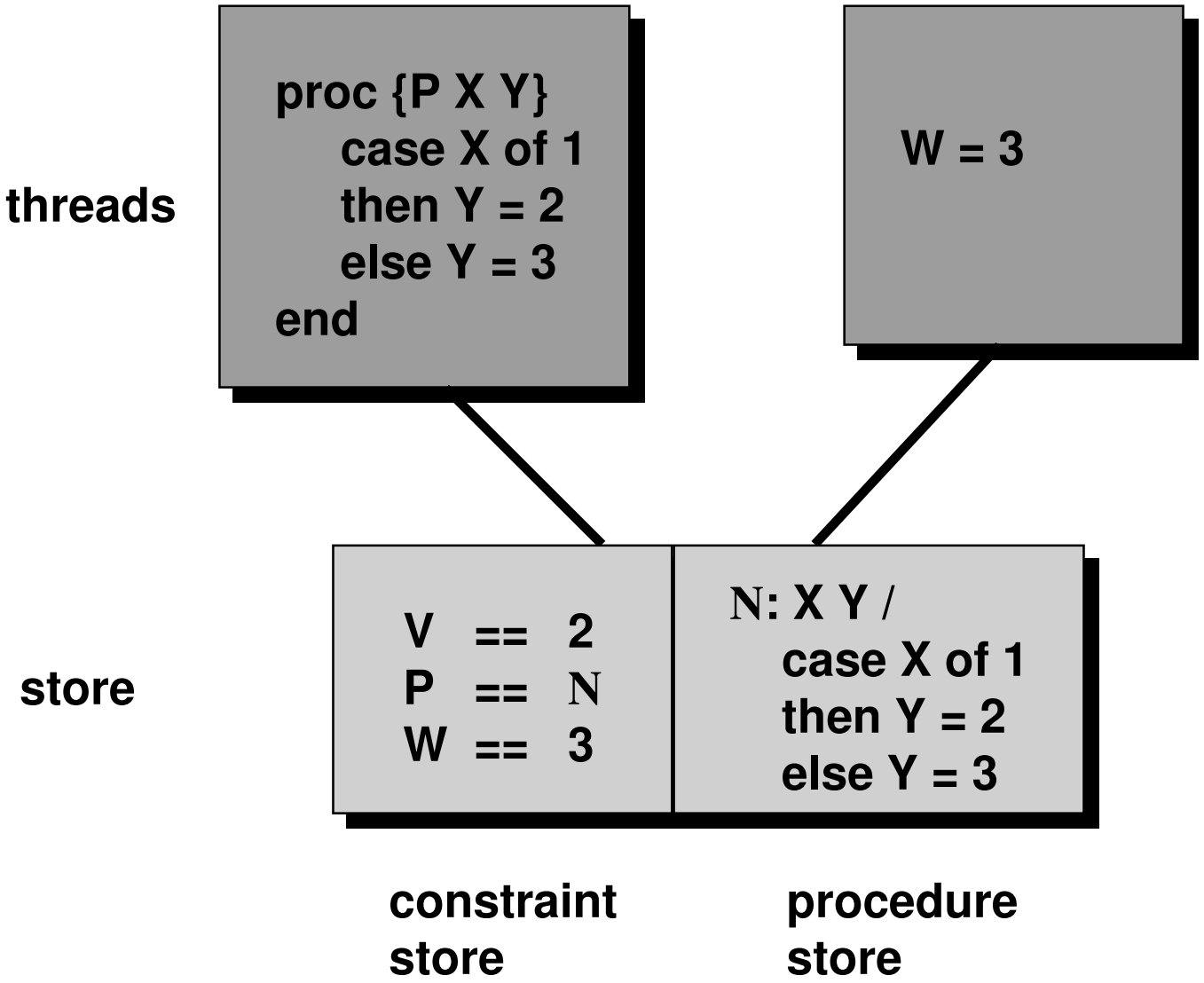

threads
y = 3
x = 2
case y of 3
then z = 5
else z = 6
store
x == 2
y == 3


threads
y = 3
x = 2
z = 5
store
x == 2
y == 3
z == 5

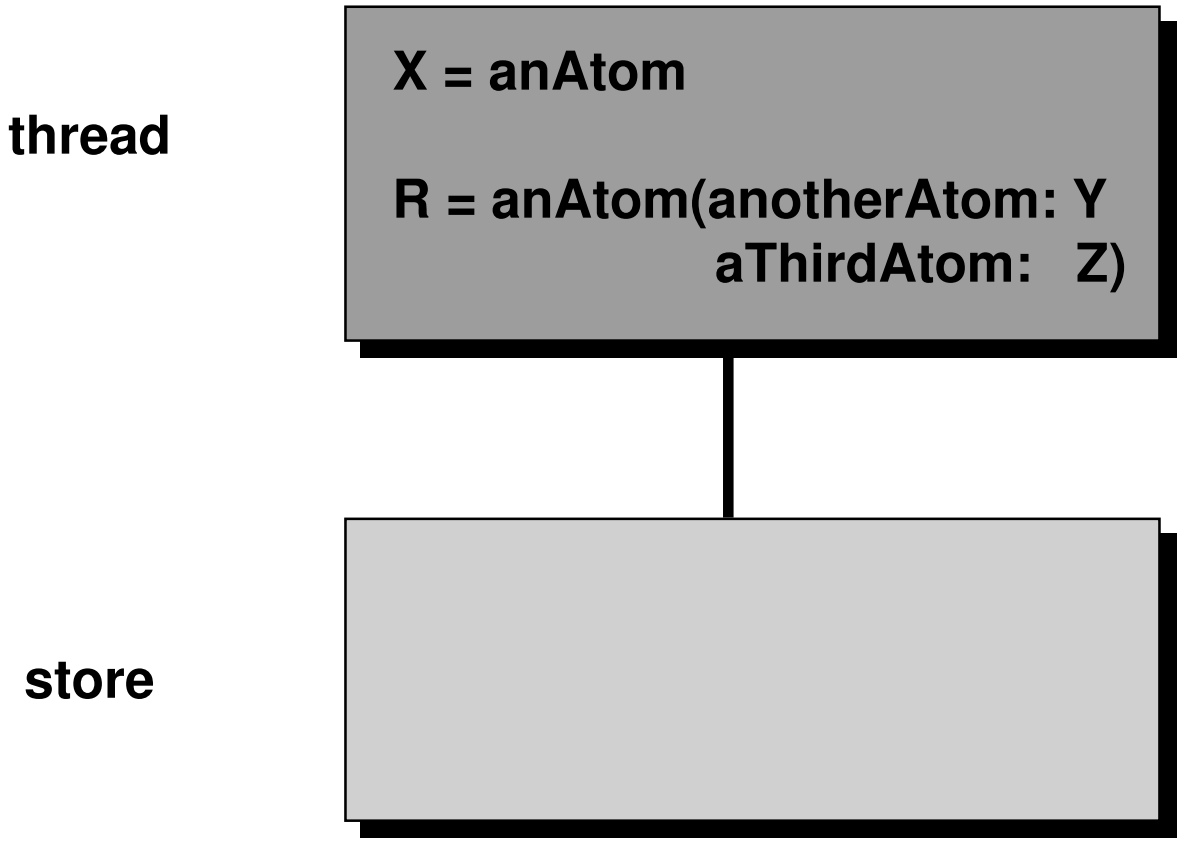
First-class Procedures

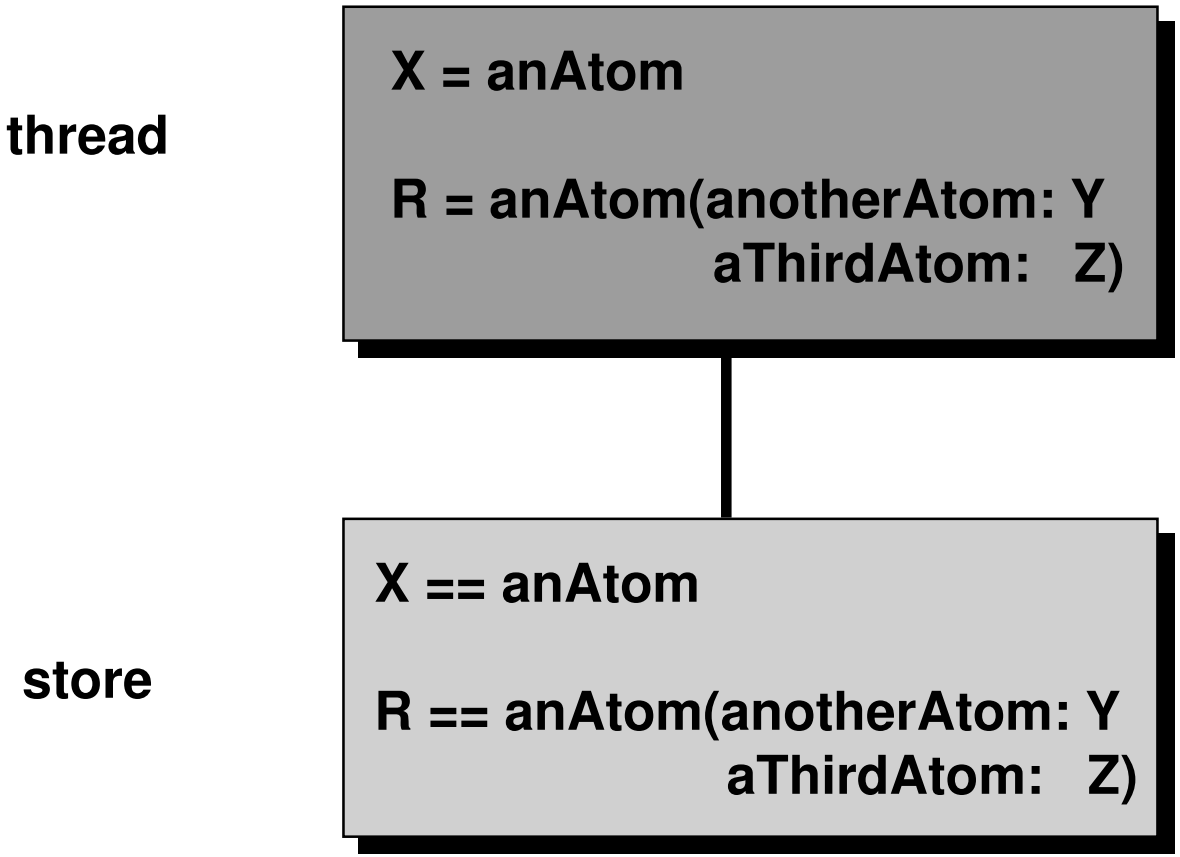



threads
proc {P X Y}
case X of 1
then Y = 2
else Y = 3
end
{P V W}
store
V == 2
P == N
N: X Y /
case X of 1
then Y = 2
else Y = 3
constraint store
procedure store


threads
proc {P X Y}
case X of 1
then Y = 2
else Y = 3
end
W = 3
store
V == 2
P == N
W == 3
N: X Y /
case X of 1
then Y = 2
else Y = 3
constraint store
procedure store

Atoms and Records

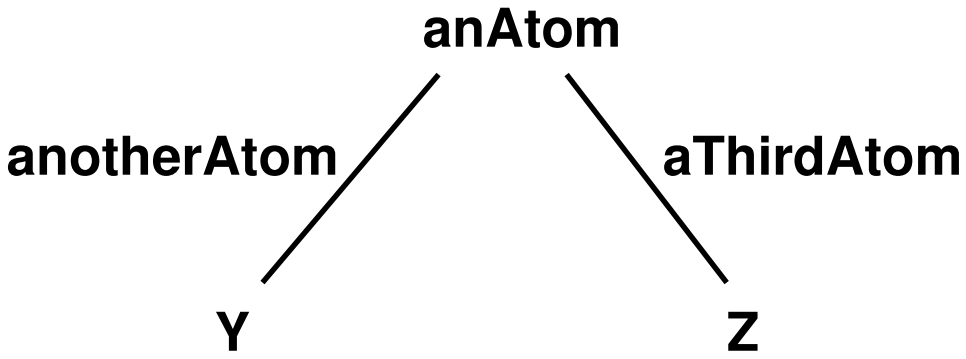



thread
X = anAtom
R = anAtom(anotherAtom: Y
aThirdAtom: Z)
store
X == anAtom
R == anAtom(anotherAtom: Y
aThirdAtom: Z)

Records are Labelled Feature Trees

```
R == anAtom(anotherAtom: Y
            aThirdAtom:  Z)
```

R ==



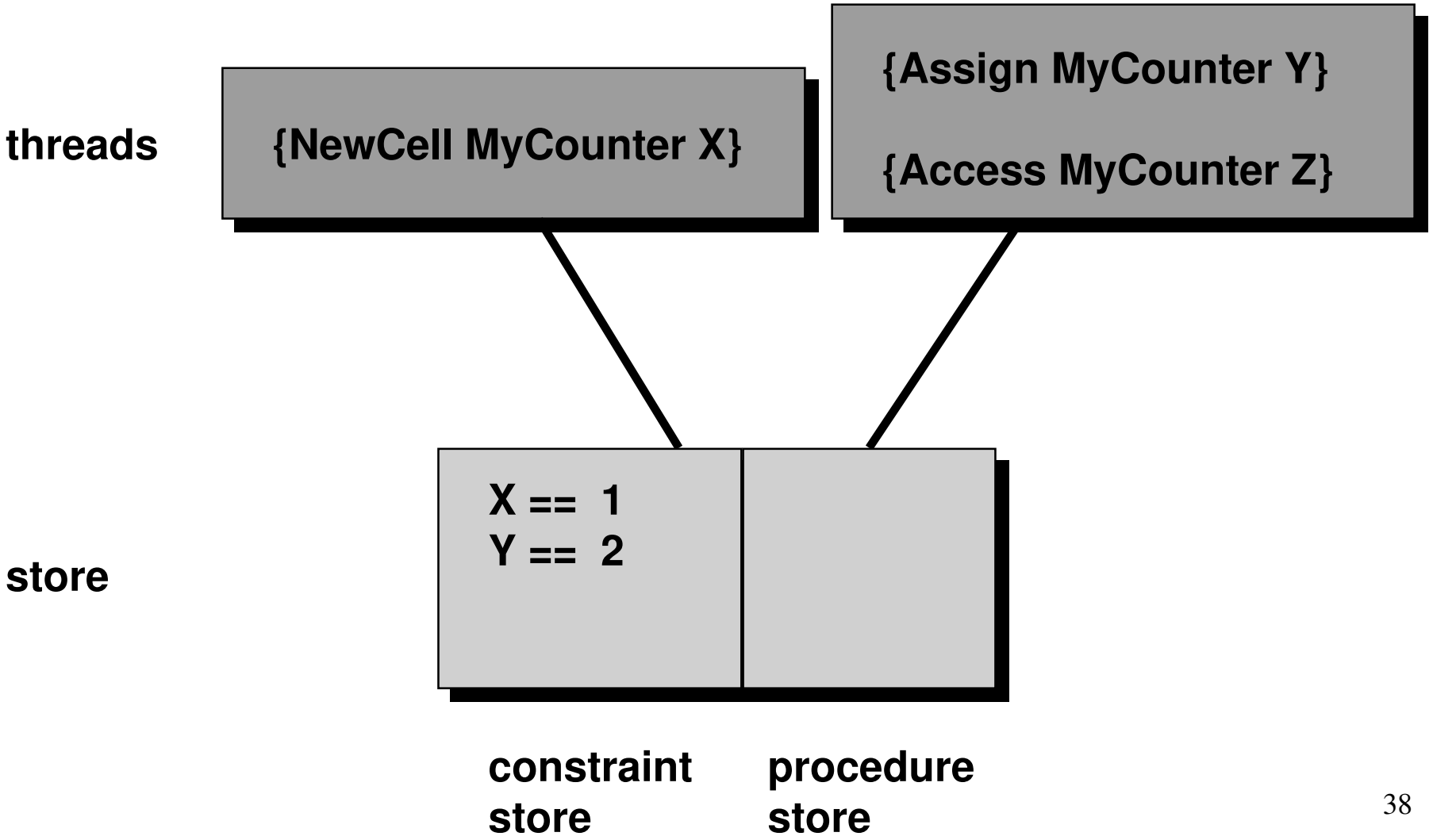
Records for Logic Programming [Smolka, Treinen 94]

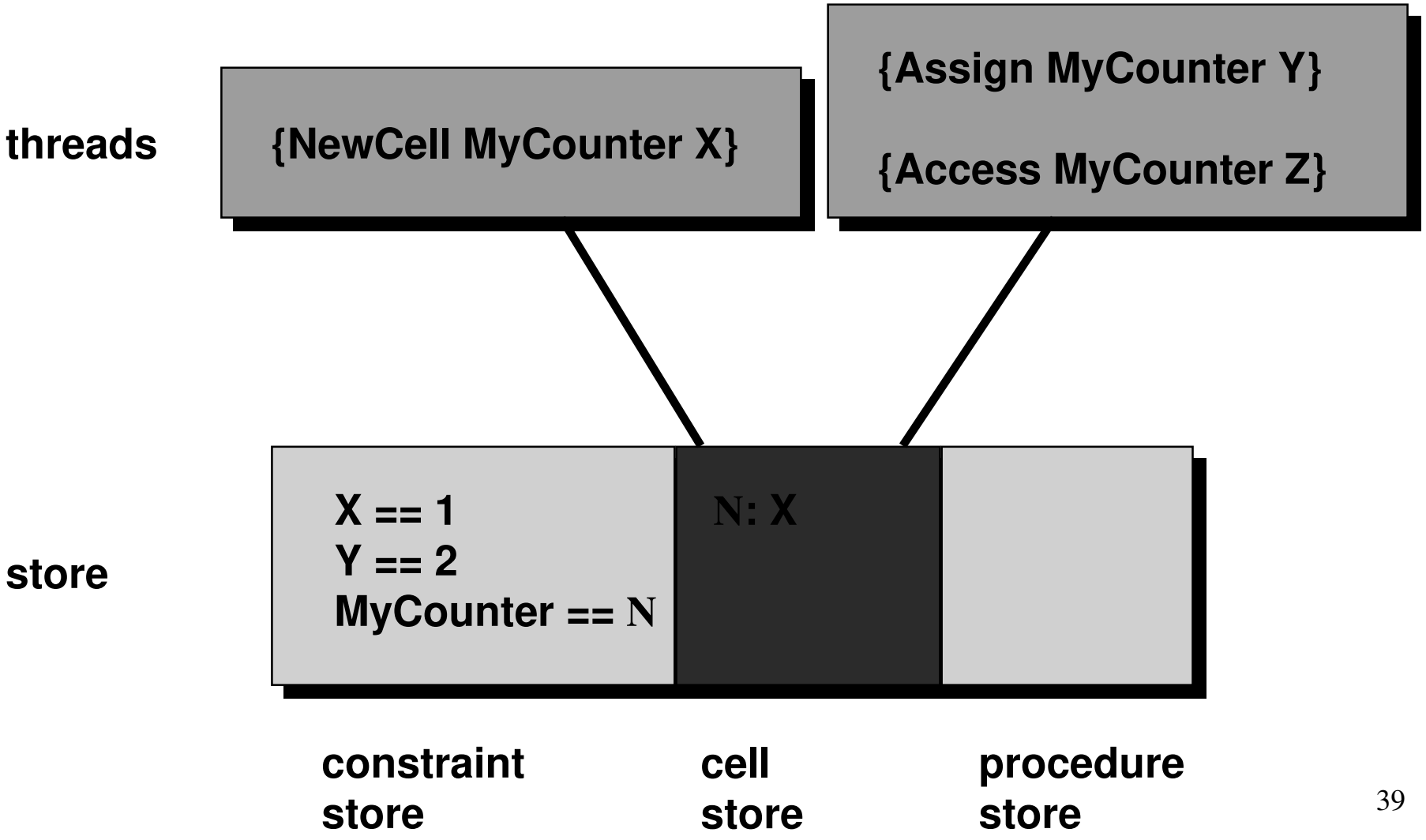
Lists as Records

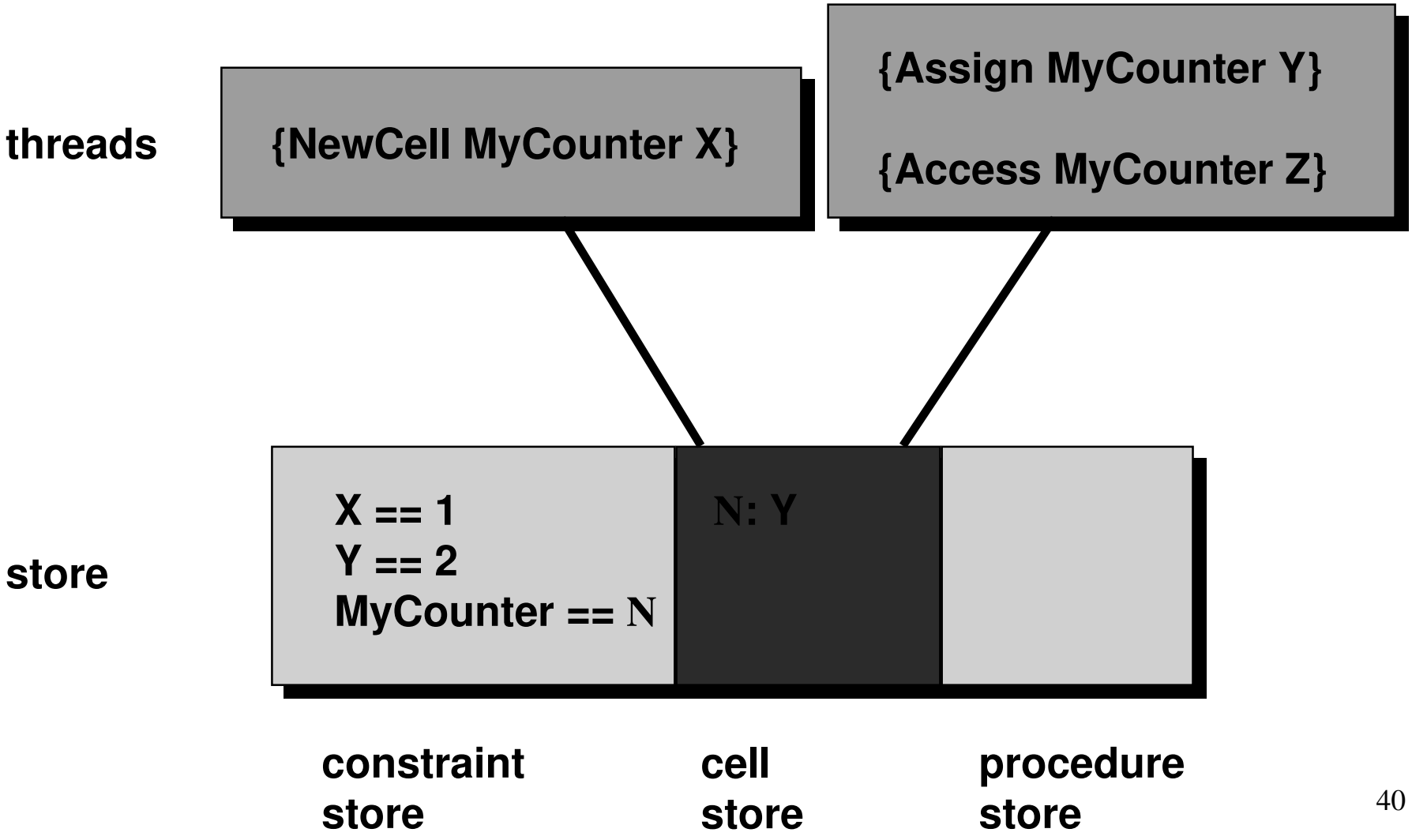
R == Y | Z | nil

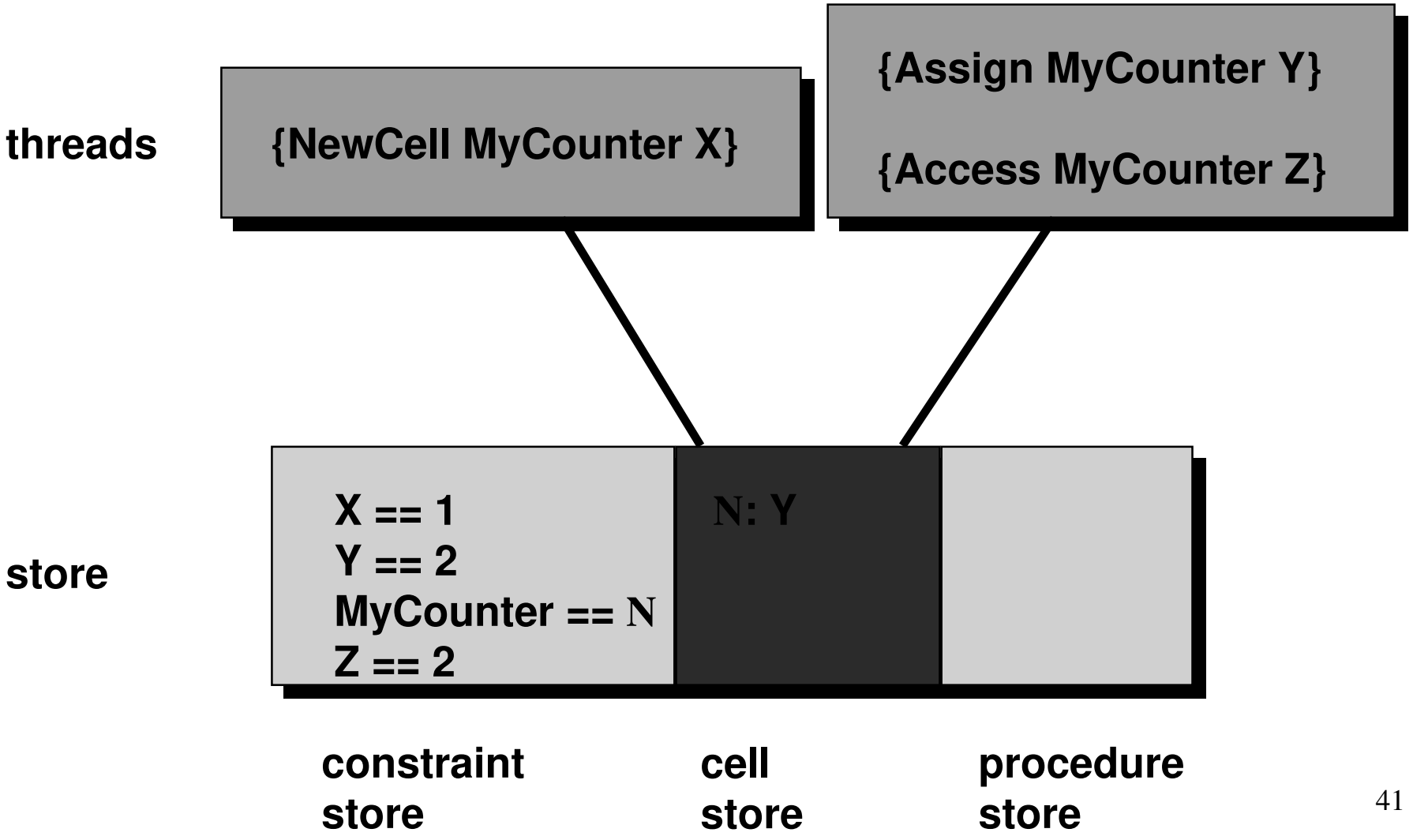
R == cons(1: Y 2: cons(1: Z 2: nil))

Cells: Random Access through the Back Door

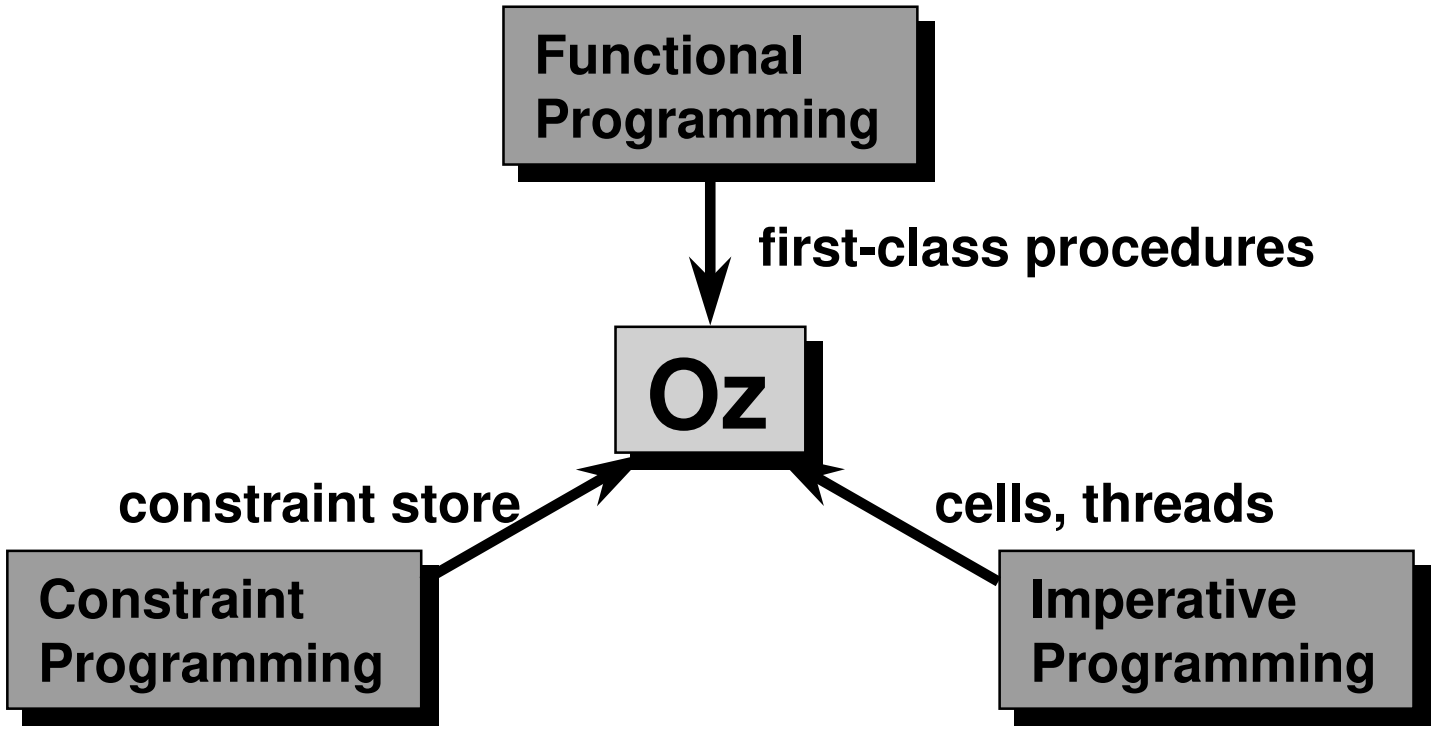



threads
{NewCell MyCounter X}
{Assign MyCounter Y}
{Access MyCounter Z}
store
X == 1
Y == 2
MyCounter == N
N: X
constraint store
cell store
procedure store


threads
{NewCell MyCounter X}
{Assign MyCounter Y}
{Access MyCounter Z}
store
X == 1
Y == 2
MyCounter == N
N: Y
constraint store
cell store
procedure store


threads
{NewCell MyCounter X}
{Assign MyCounter Y}
{Access MyCounter Z}
store
X == 1
Y == 2
MyCounter == N
Z == 2
N: Y
constraint store
cell store
procedure store

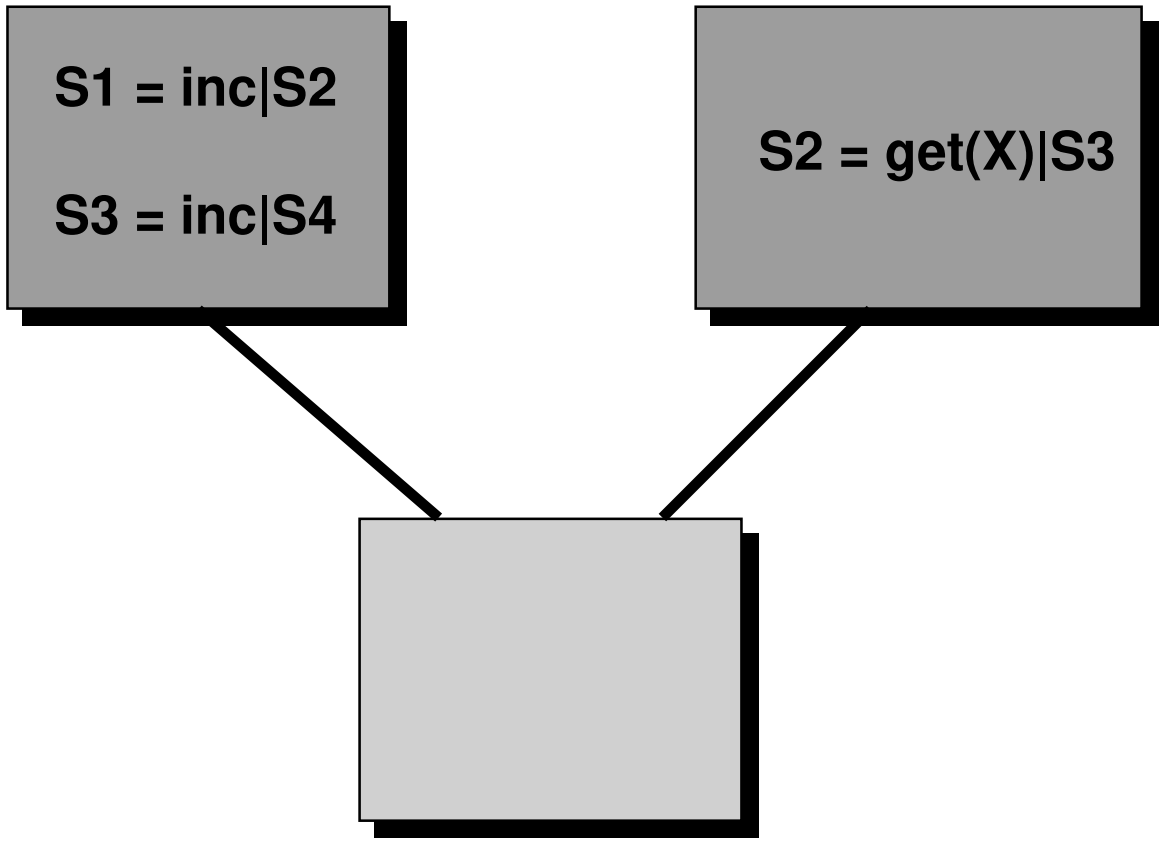
Concepts of Oz

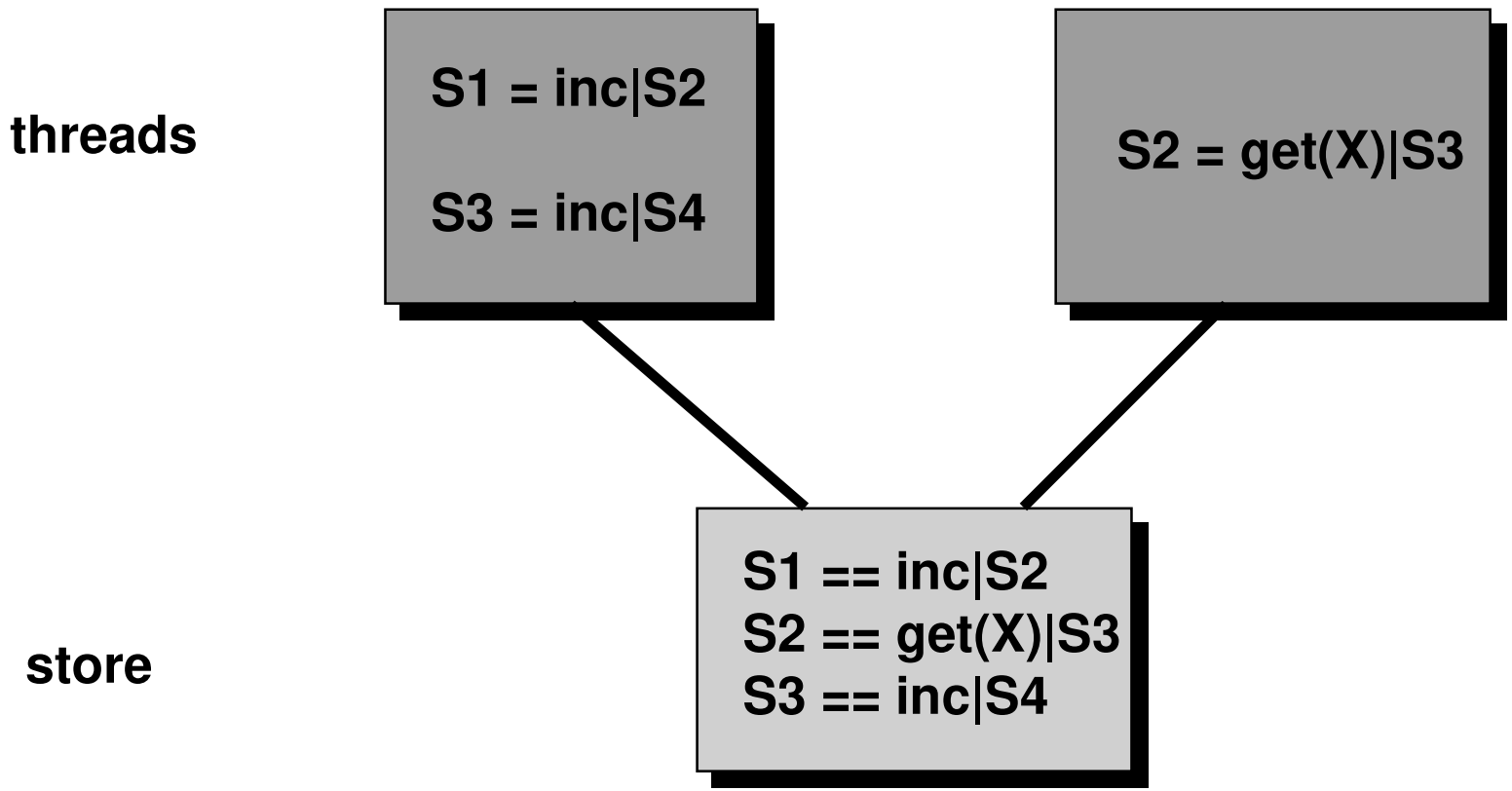


Streams

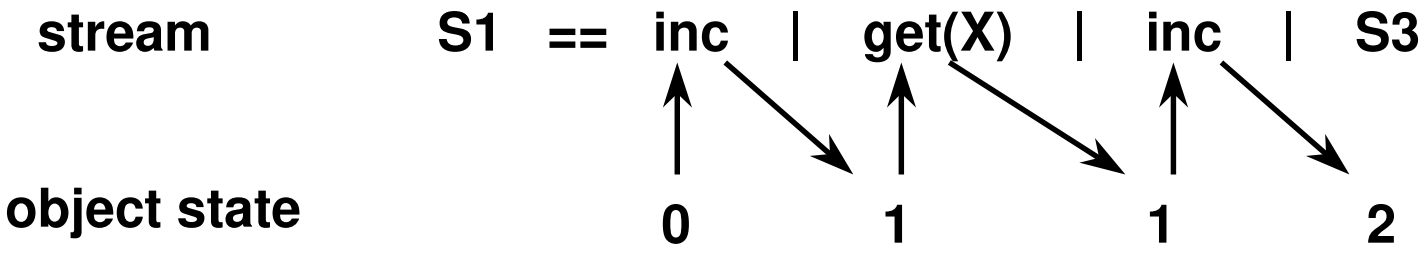
threads

store




threads
S1 = inc|S2
S3 = inc|S4
S2 = get(X)|S3
store
S1 == inc|S2
S2 == get(X)|S3
S3 == inc|S4

Stream-based Objects



```
proc {Counter S Value}
   case S
   of Message|Sr
   then   % compute
          % NewValue
          {Counter Sr NewValue}
end

{Counter S1 0}
```

Object-oriented Programming in Concurrent Prolog
[Shapiro, Takeuchi 83]