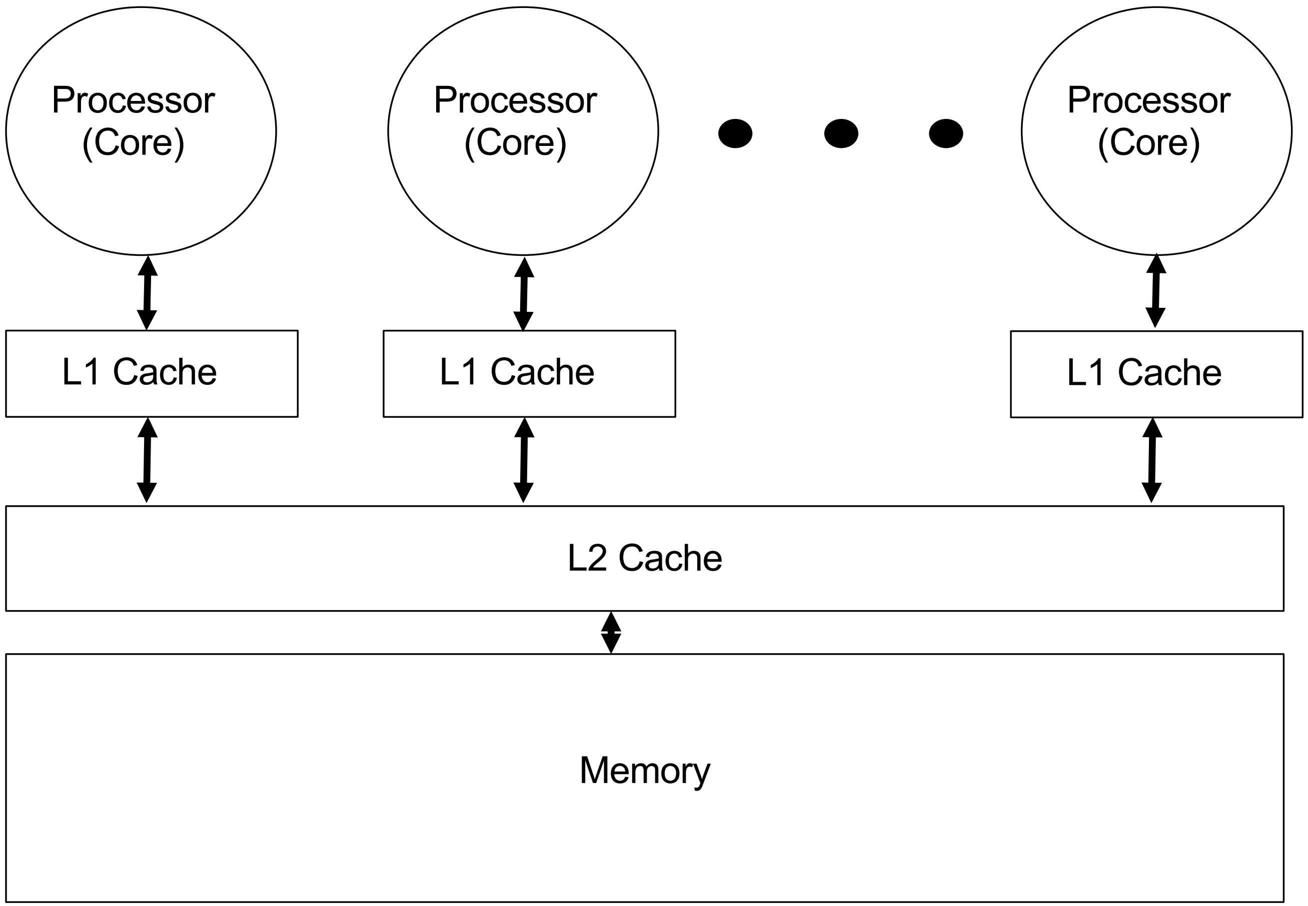
Lecture 6

Chip multiprocessors

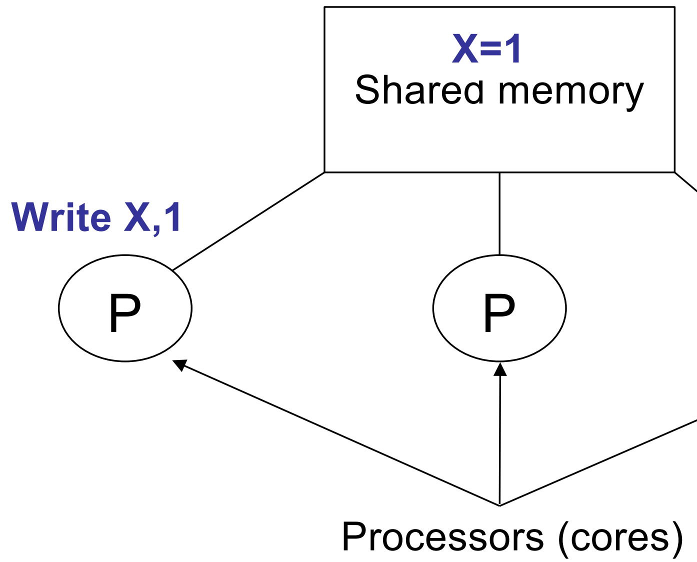
- **Multithreading techniques (Sections 5.2.1, 8.3)**
  ✓Interleaved (fine-grain) multithreading
  ✓Block (coarse-grain) multithreading
  ✓Simultaneous multithreading
- **Cache coherence solutions (Sections 5.4.1-5.4.3)**

Multicore
Programming Model

Architecture Model of a Multicore System

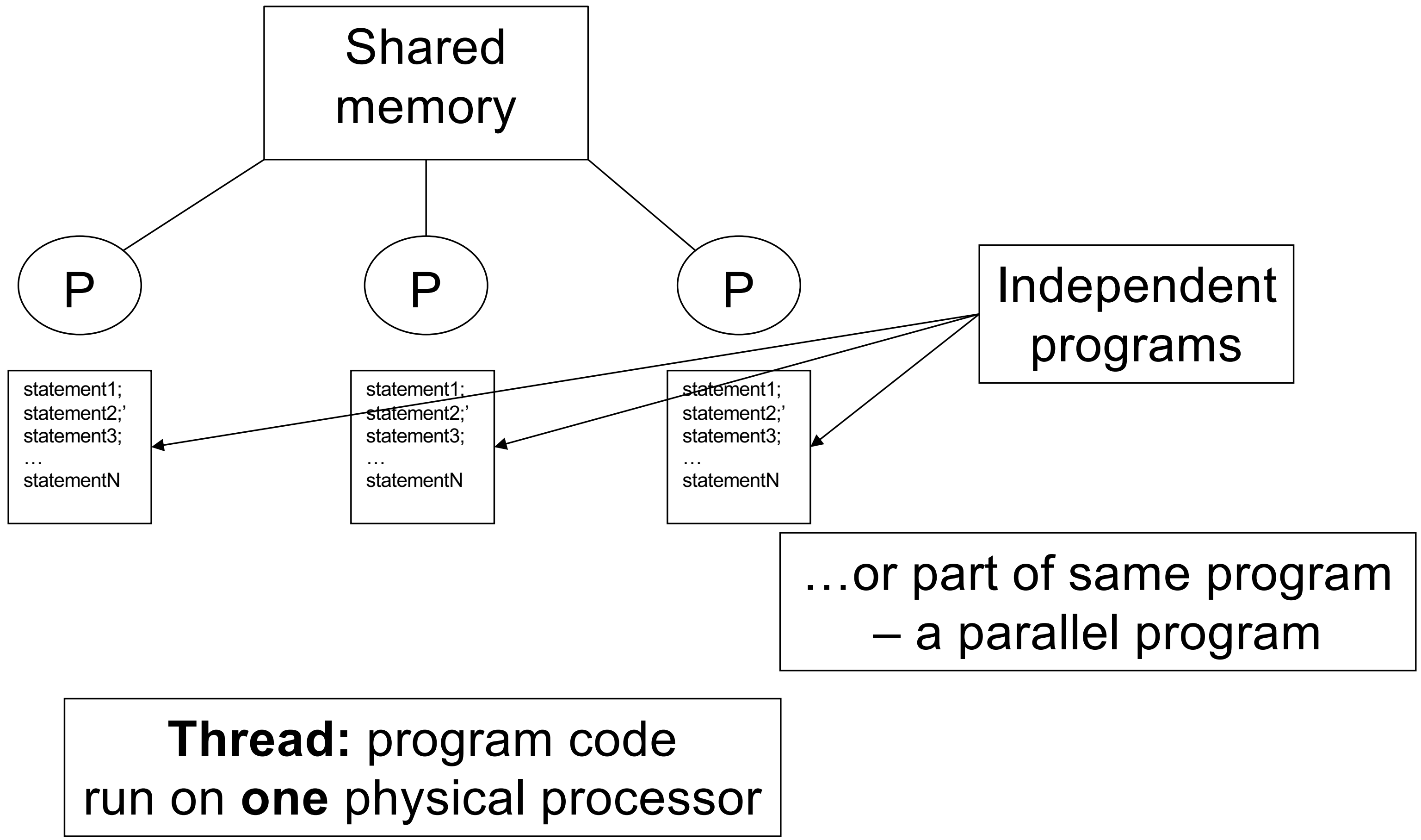


Programming Model of a Multicore System



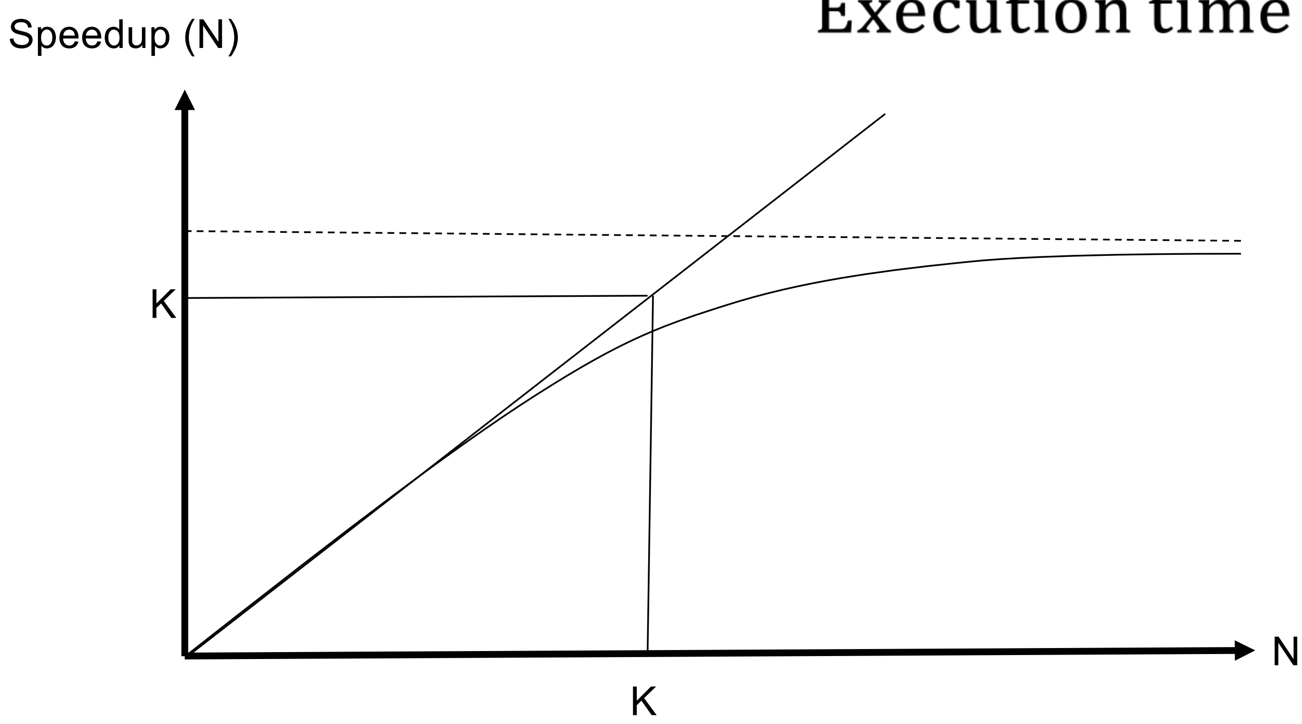
Discussion:
Make a parallel version of vector addition:
for(i=0;i<100;i++)
  A[i] = B[i] + C[i];
On 10 processors

Thread-Level Parallelism



Goal: Parallel Execution

$$\text{Speedup (N)} = \frac{\text{Execution time 1 thread}}{\text{Execution time N threads}}$$



SP < 1/f

- f – fraction of serial code
- See Amdahl's Law

Discussion:
What are the sources for sub-linear speedup?

Question:
Assume that 90% of a sequential program can be parallelized.

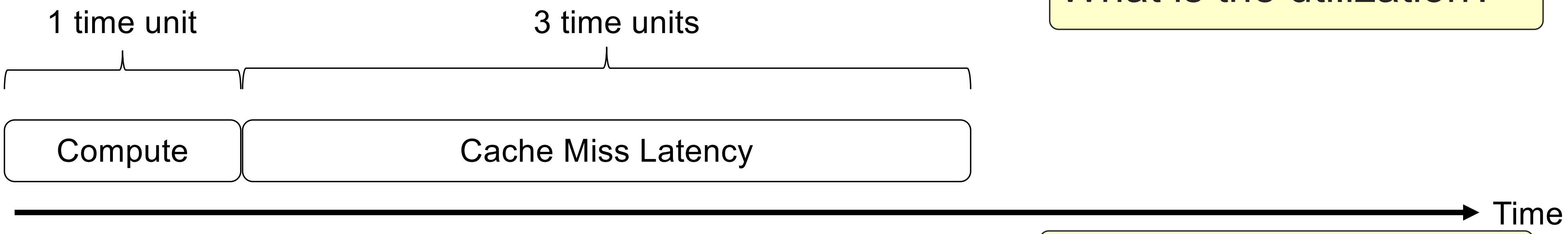
What is the maximum speedup we can get?

Answer:
According to Amdahl's Law:
$SP = 1/(f – (1-f)/P)$
Hence, $SP<1/f = 1/(1-0.90) = 10$

Multithreading
(Sections 5.2.1, 8.3)

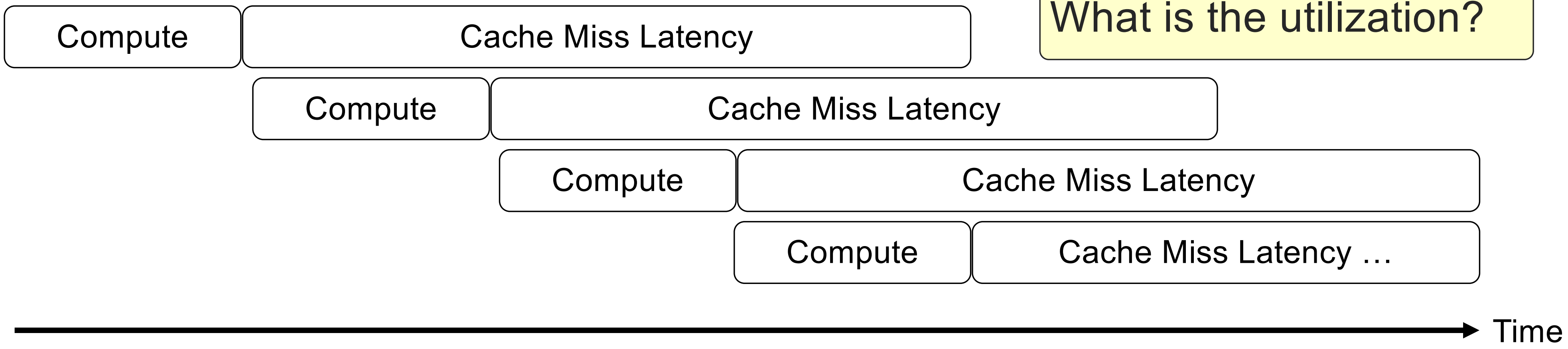
Motivation

Question:
What is the utilization?



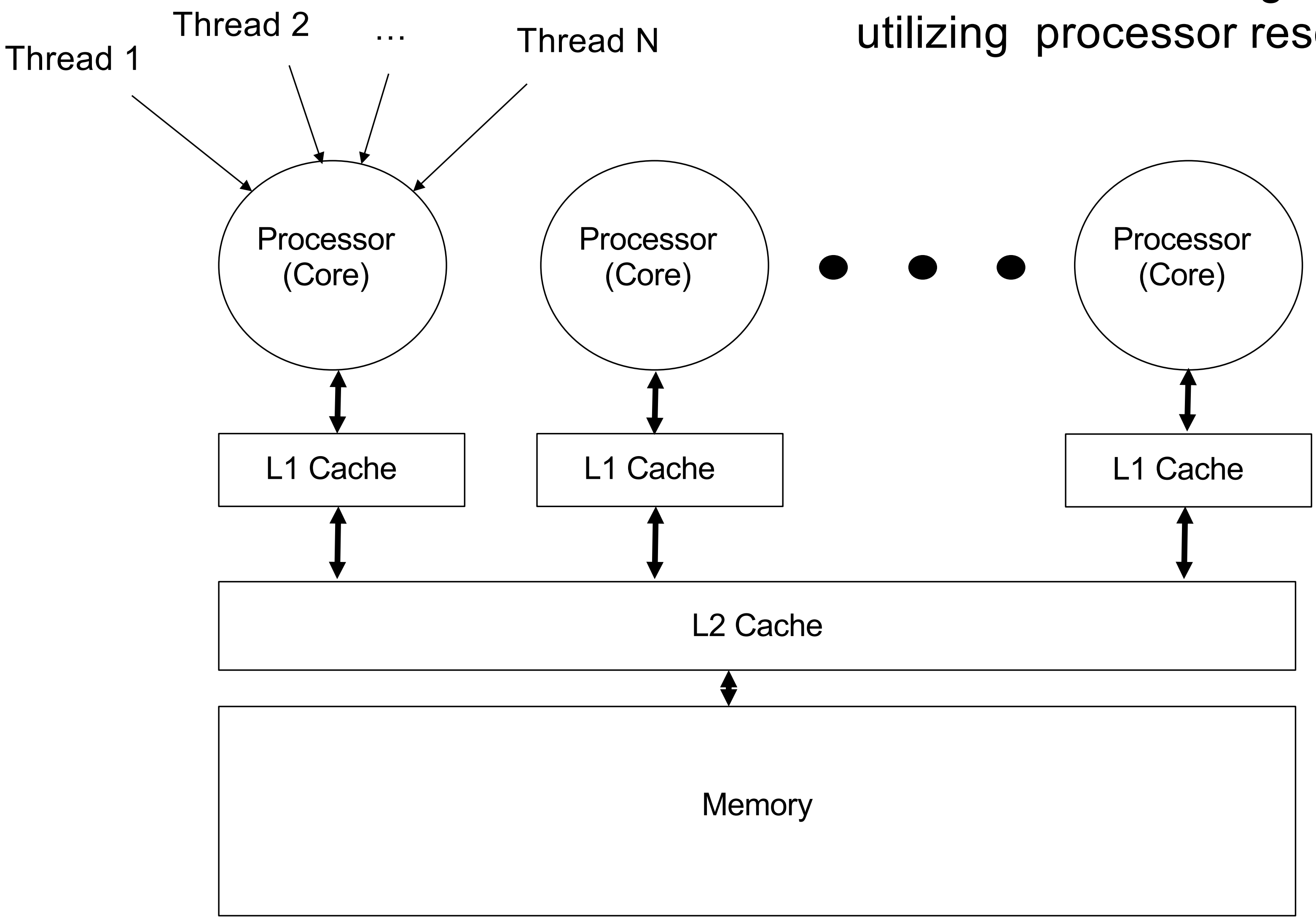
Utilization: 25%!

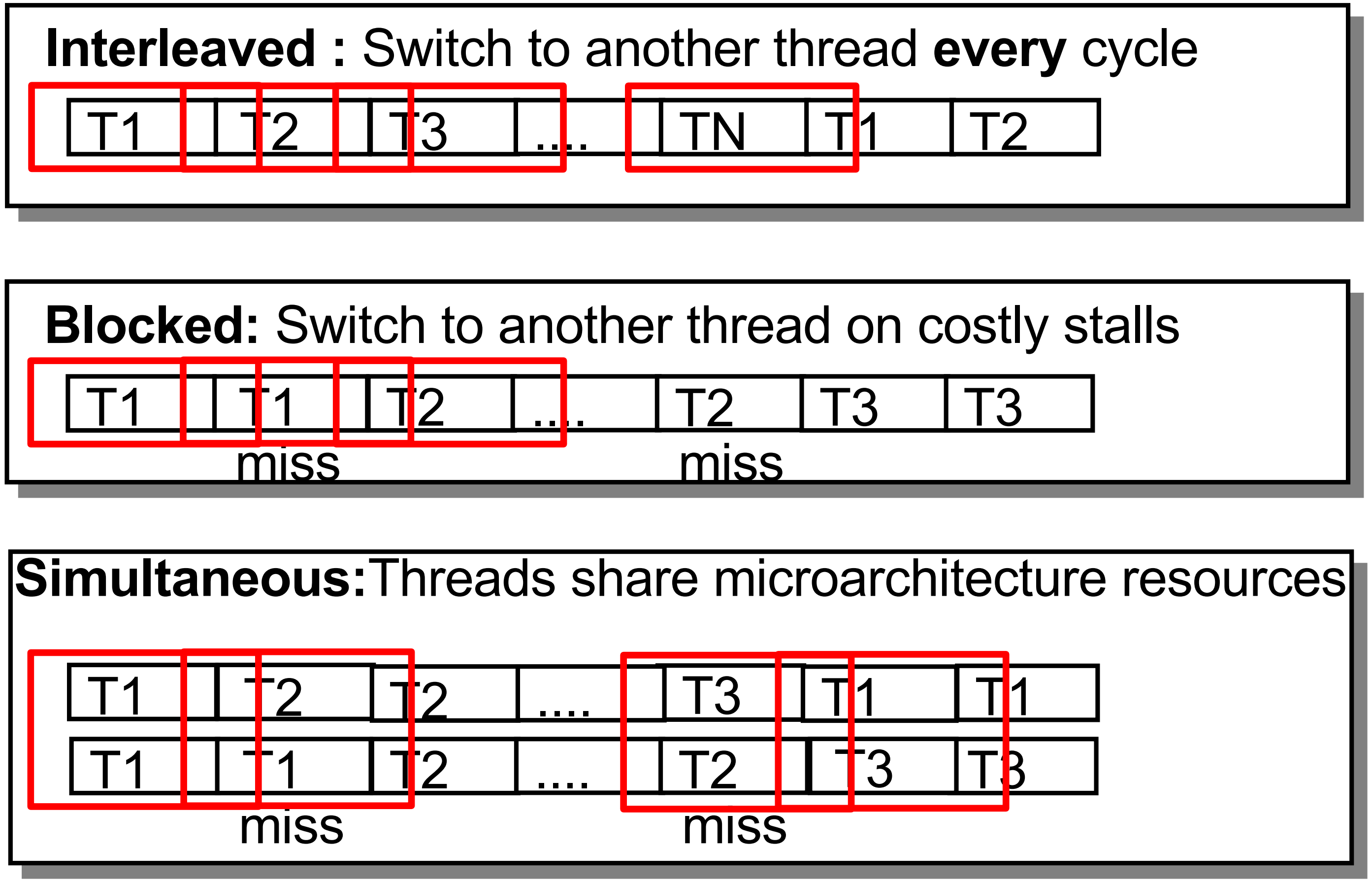
Question:
What is the utilization?



Four programs (threads) could overlap the latency
Utilization: 100%!

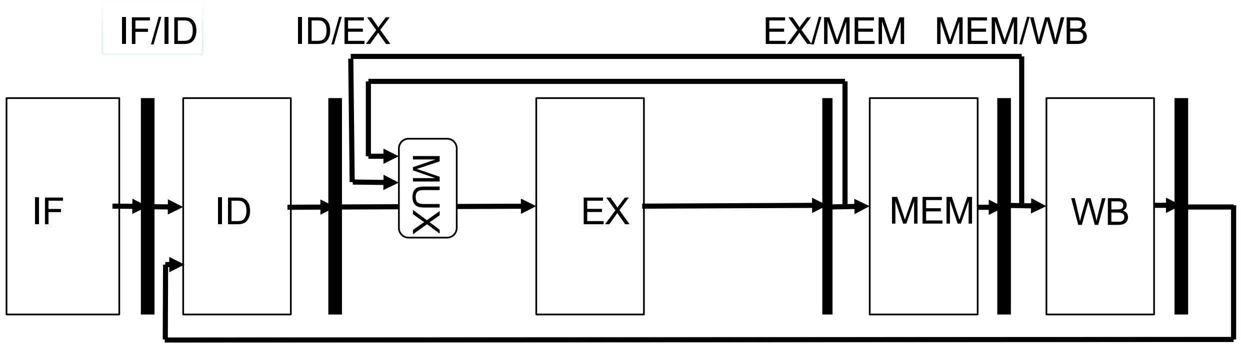
Multi-threading aims at
utilizing processor resources better



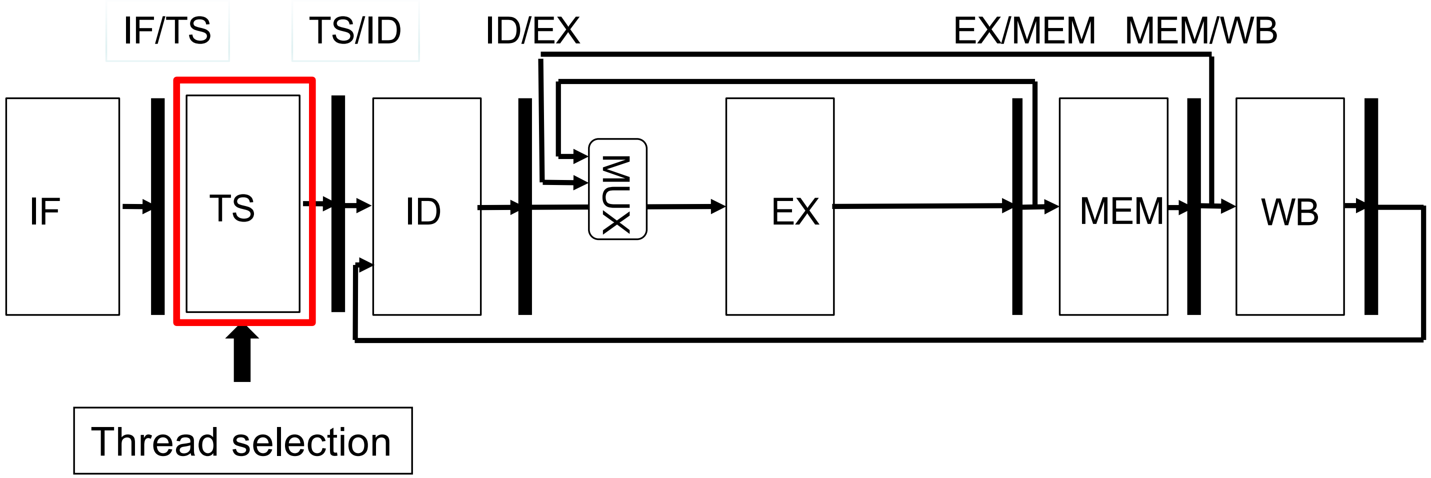


Interleaved : Switch to another thread every cycle
T1
T2
T3
....
TN
T1
T2
Blocked: Switch to another thread on costly stalls
T1
T1
T2
....
T2
T3
T3
miss
miss
Simultaneous:Threads share microarchitecture resources
T1
T2
T2
....
T3
T1
T1
T1
T1
T2
....
T2
T3
T3
miss
miss

Interleaved Multithreading
Blocked Multithreading
Simultaneous Multithreading



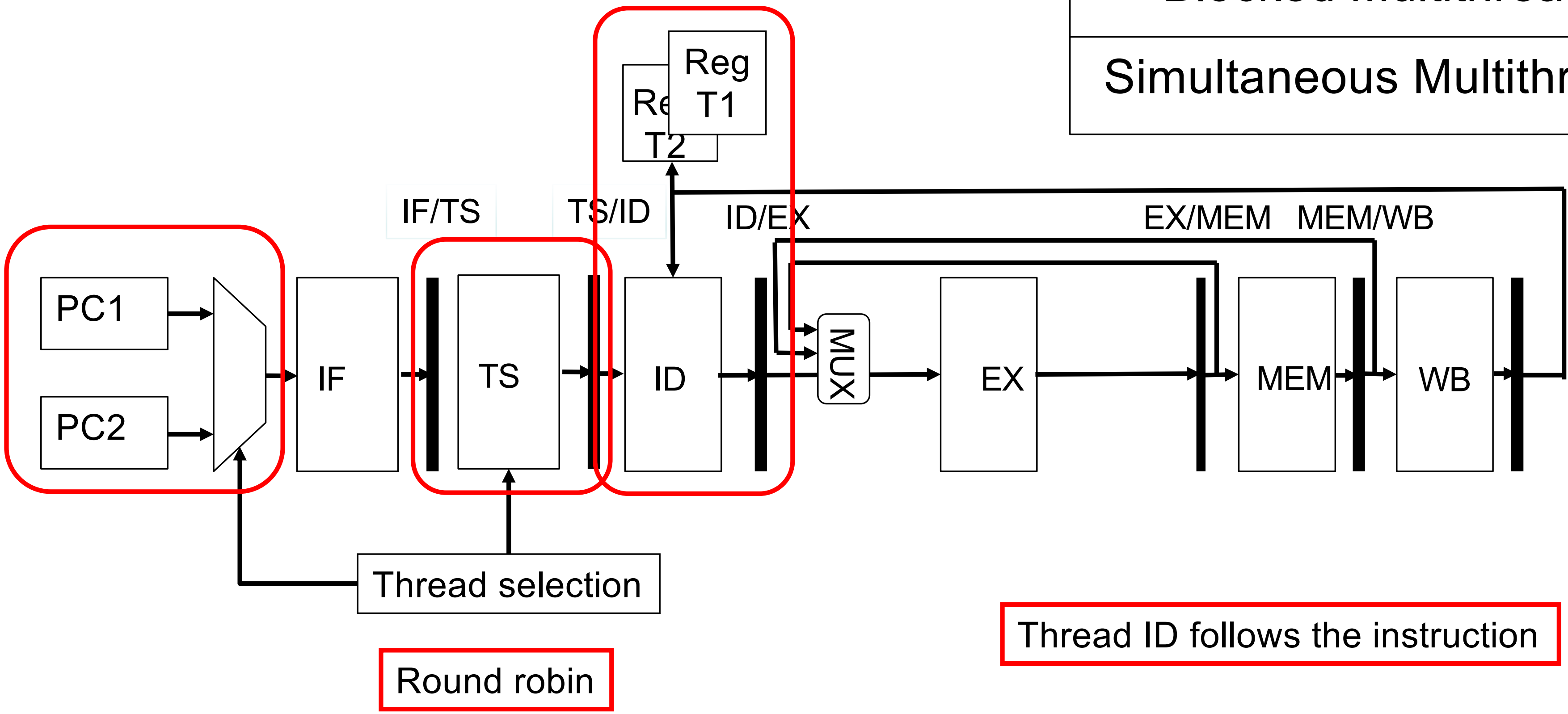
Interleaved Multithreading
Blocked Multithreading
Simultaneous Multithreading



Interleaved Multithreading

Blocked Multithreading

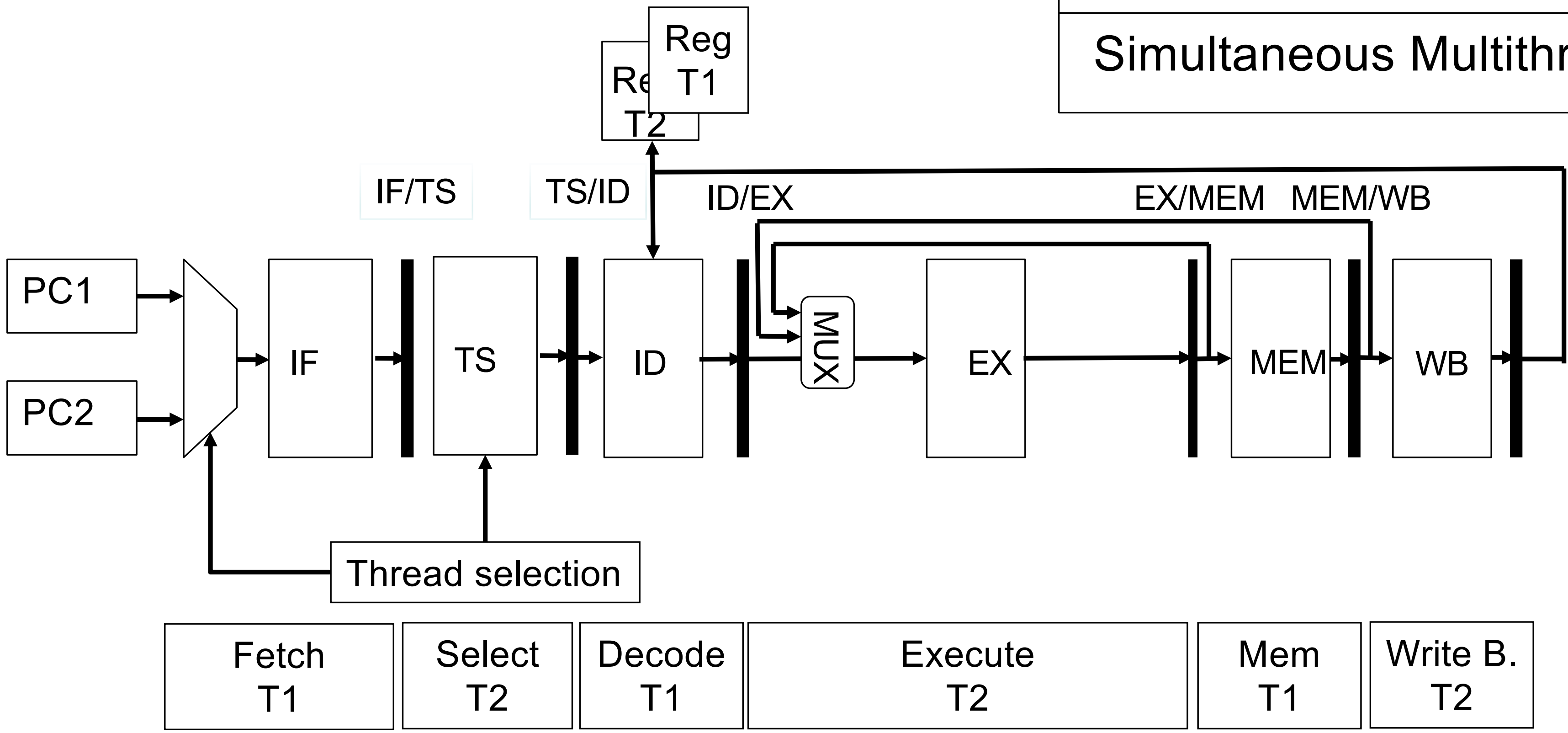
Simultaneous Multithreading



Interleaved Multithreading

Blocked Multithreading

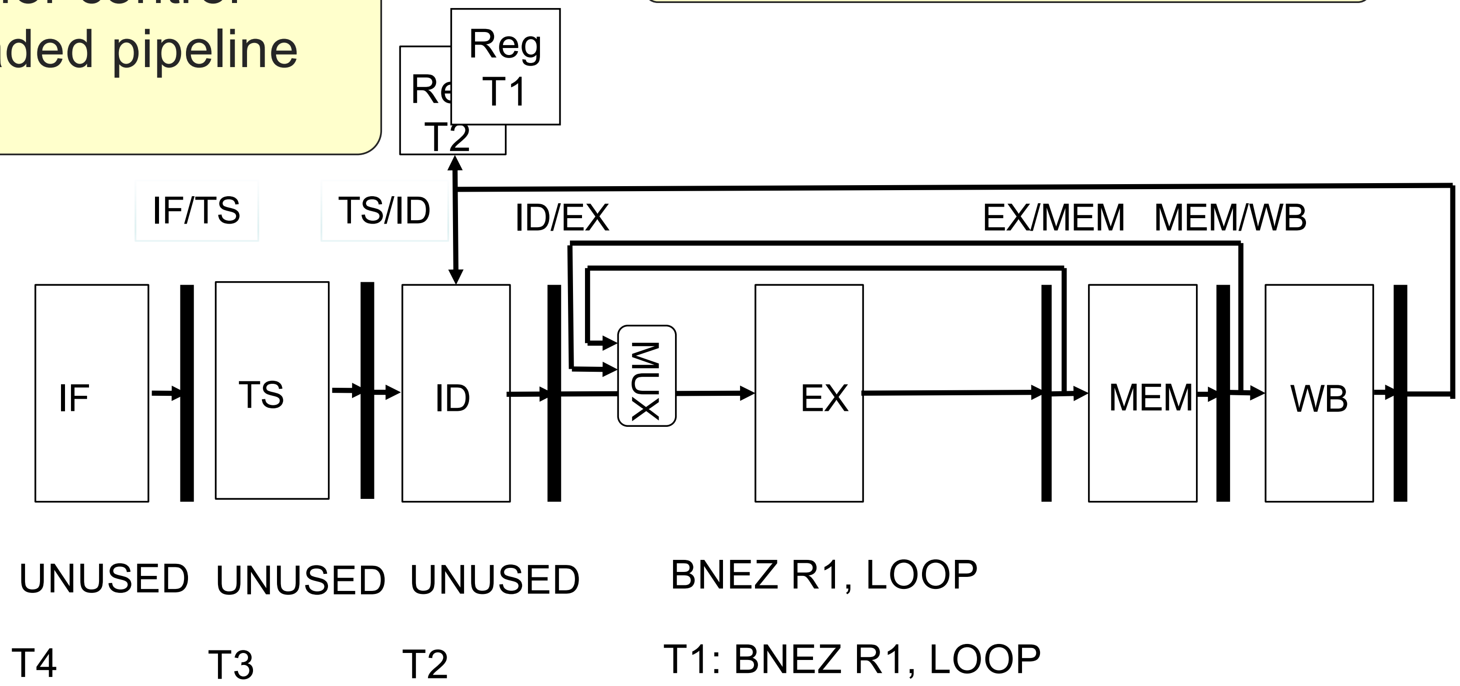
Simultaneous Multithreading



Hazards have less impact. Can hide long latencies

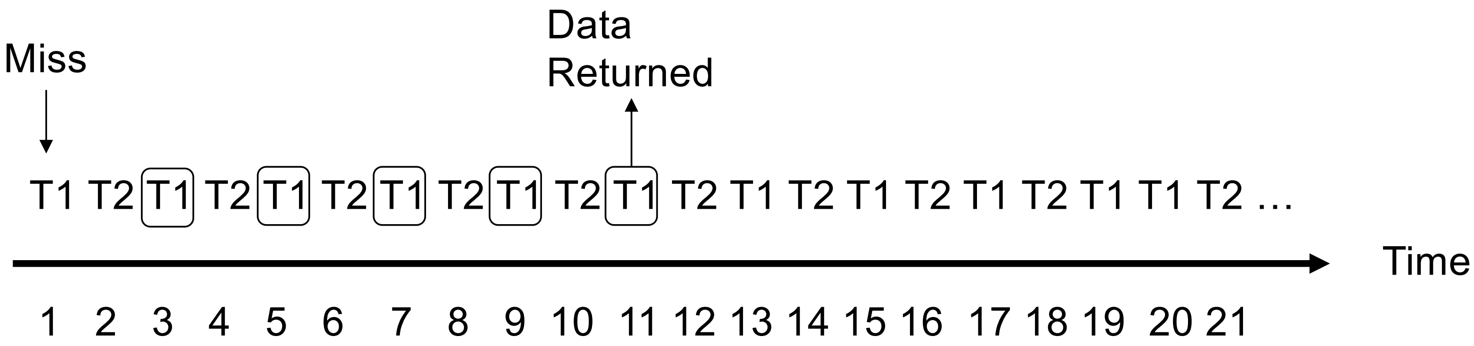
Question:
How many threads are needed to eliminate the cycles lost for control hazards in the multithreaded pipeline below?

Answer:
Four threads



Question:
How many cycles are unused if T1 experiences a cache miss that takes 10 cycles to serve?

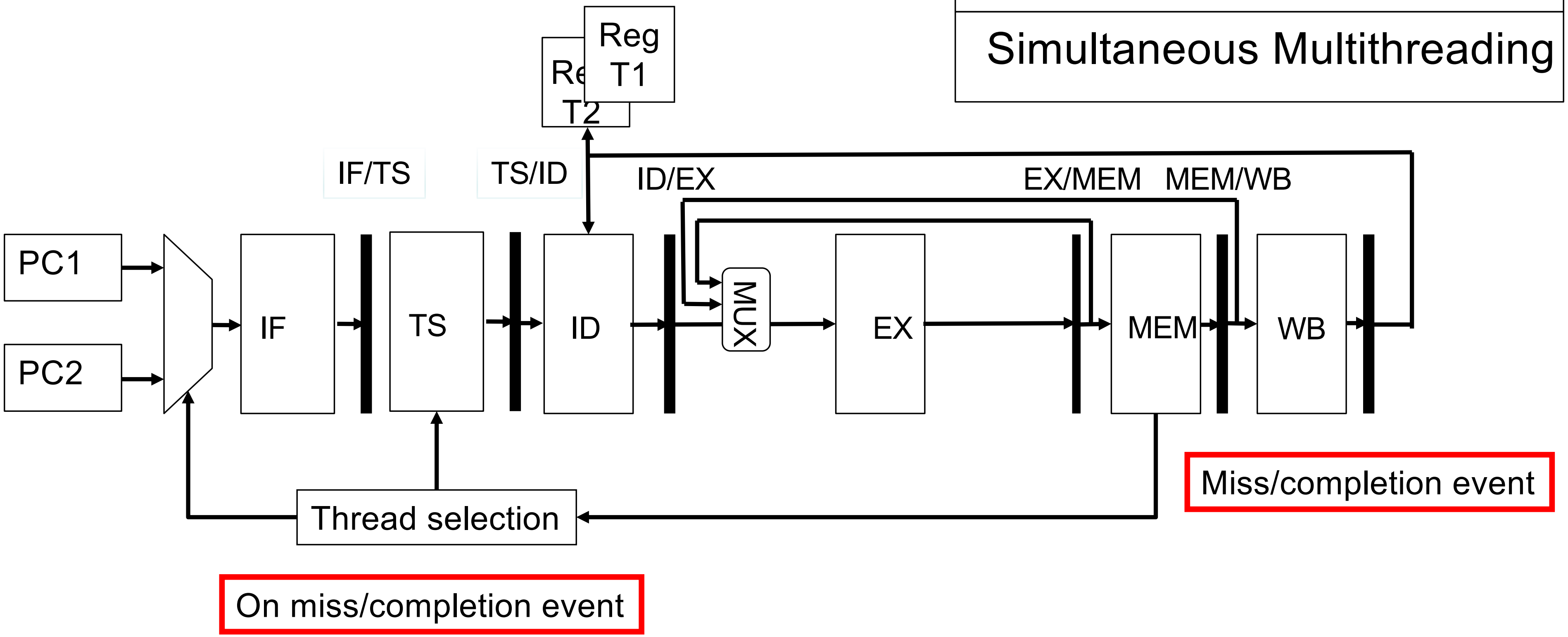
Answer:
5 cycles are unused. T2 can make use of half of the cycles.



Interleaved Multithreading

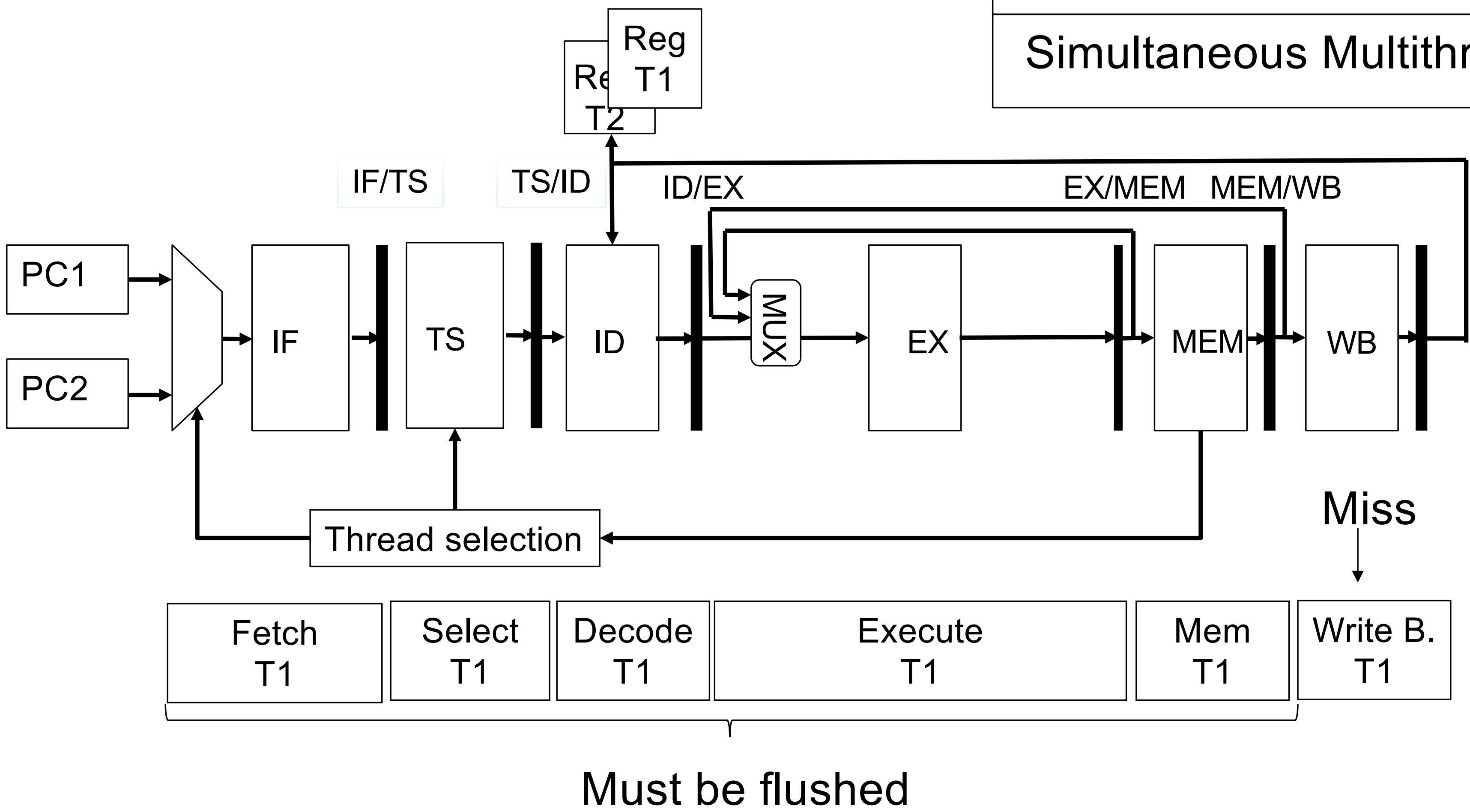
Blocked Multithreading

Simultaneous Multithreading



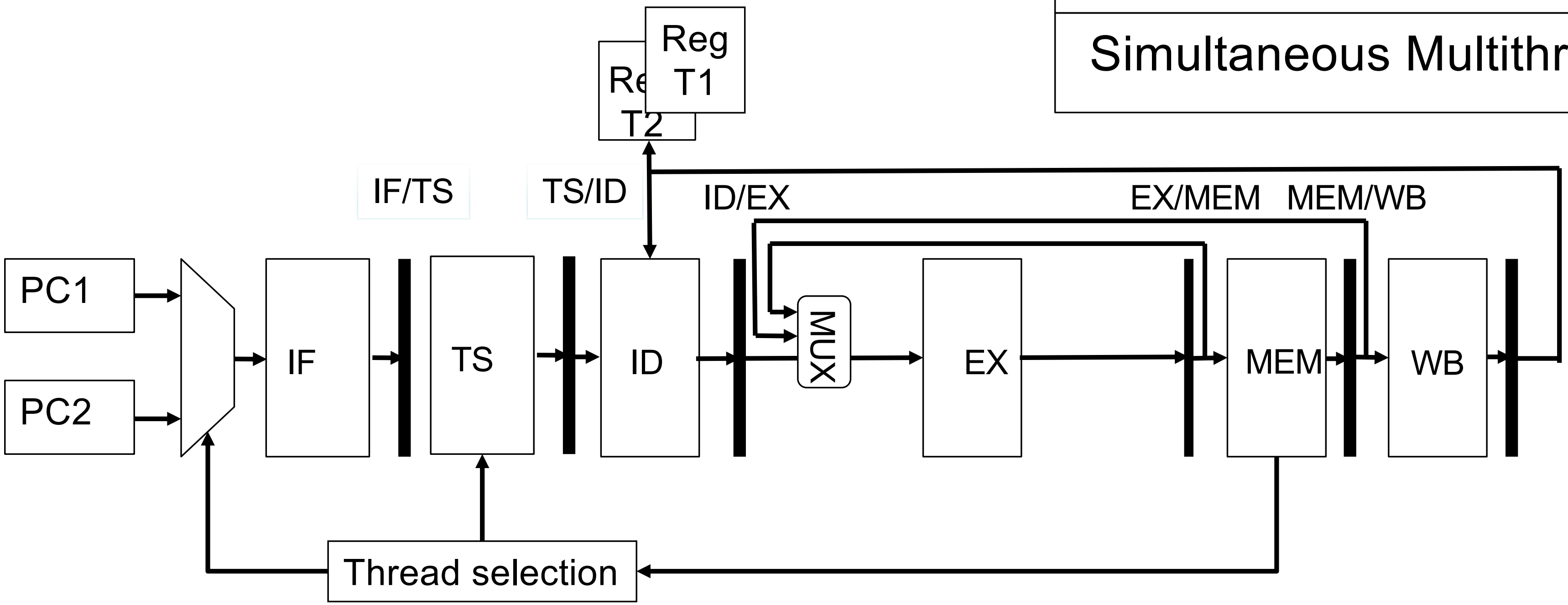
Cycle X

Interleaved Multithreading
Blocked Multithreading
Simultaneous Multithreading



Cycle X + 1

Interleaved Multithreading
Blocked Multithreading
Simultaneous Multithreading



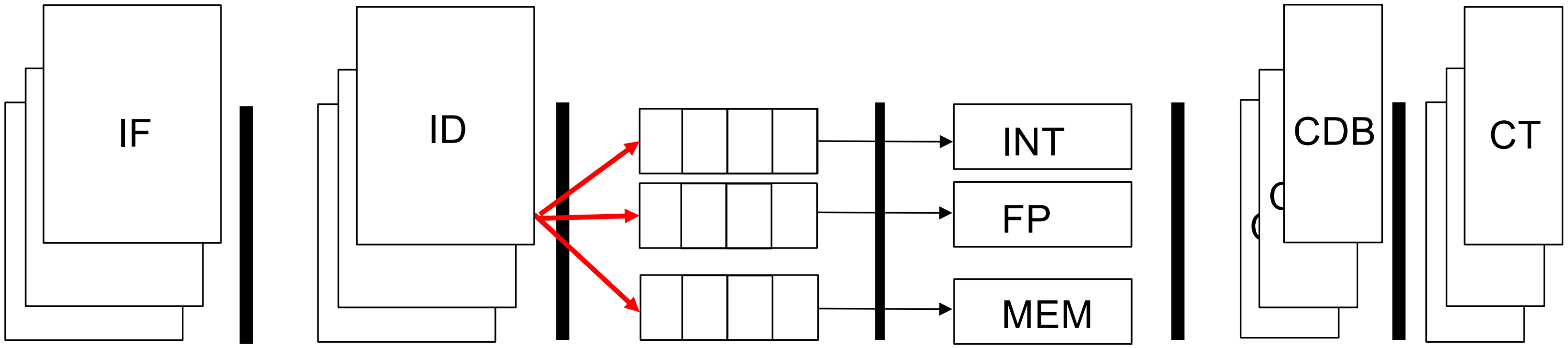
Fetch T2	Unused	Unused	Unused	Unused	Unused

Only effective for long-latency operations
Significant overhead in deep out-of-order pipelines

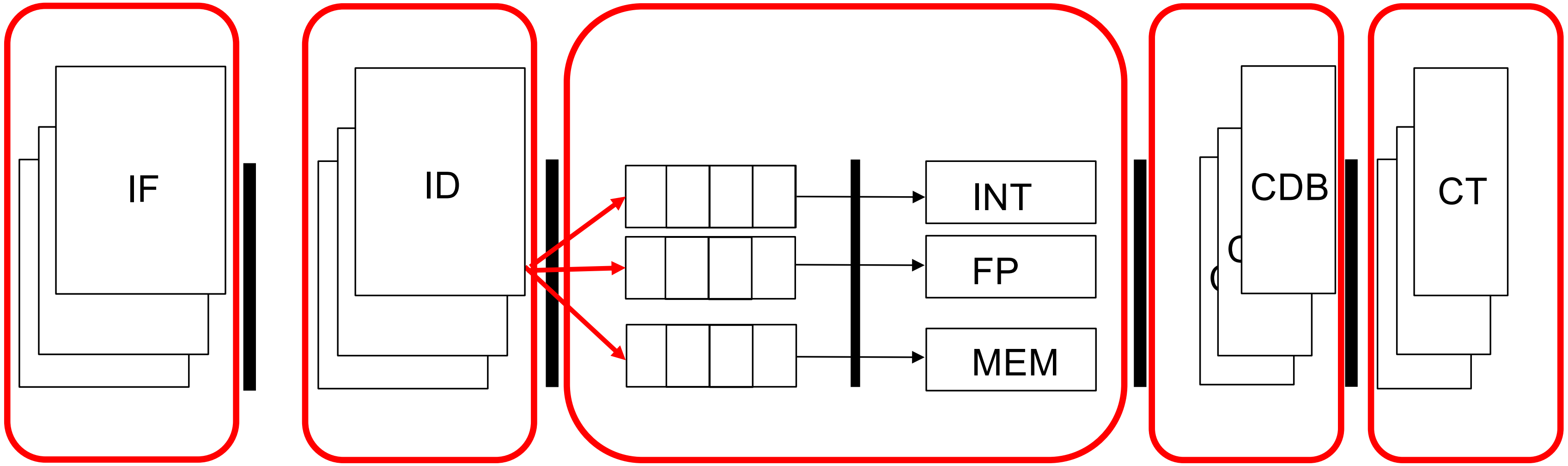
Interleaved Multithreading

Blocked Multithreading

Simultaneous Multithreading



Interleaved Multithreading
Blocked Multithreading
Simultaneous Multithreading



Cache Coherence Solutions
(Sections 5.4.1-5.4.3)

Latest Write to a Location: Single Processor

Notation:

- Processor *i* reads from location X: $R_i(X)$
- Processor *i* writes a value Y to location X: $W_i(X)$=Y

Example: Consider the following sequence of reads and writes to location X:

$W_i(X)=1 \longrightarrow R_i(X) \longrightarrow W_i(X)=2 \longrightarrow R_i(X) \longrightarrow W_i(X) = 3 \longrightarrow R_i(X)$

Question:
What value is returned by the last read request?

Answer:
The returned value is 3 because it is reflected **by the latest write** to location X.

Latest Write to a Location: Two Processors

Example: Consider the following sequence of reads and writes to location X from two processors 1 and 2:

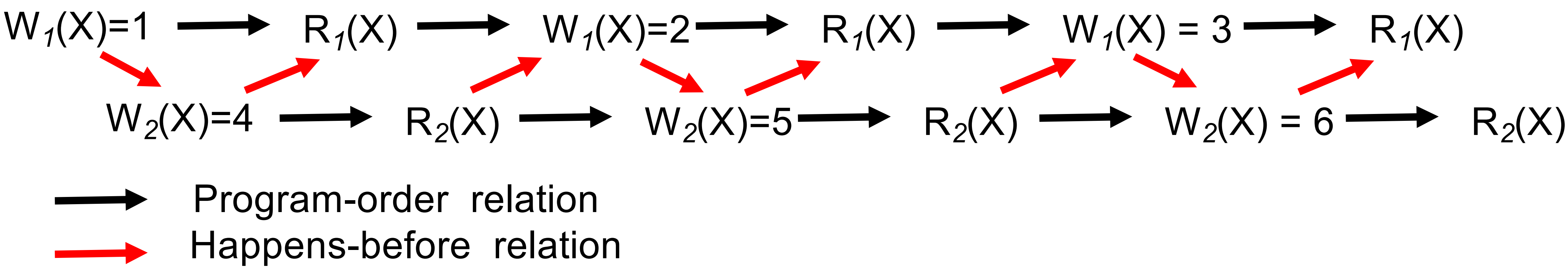
$$W_1(X)=1 \rightarrow R_1(X) \rightarrow W_1(X)=2 \rightarrow R_1(X) \rightarrow W_1(X) = 3 \rightarrow \boxed{R_1(X)}$$

$$W_2(X)=4 \rightarrow R_2(X) \rightarrow W_2(X)=5 \rightarrow R_2(X) \rightarrow W_2(X) = 6 \rightarrow R_2(X)$$

Question:
What value is returned by the last read request from processor 1?

Answer:
It all depends on the order by which writes arrive at memory!

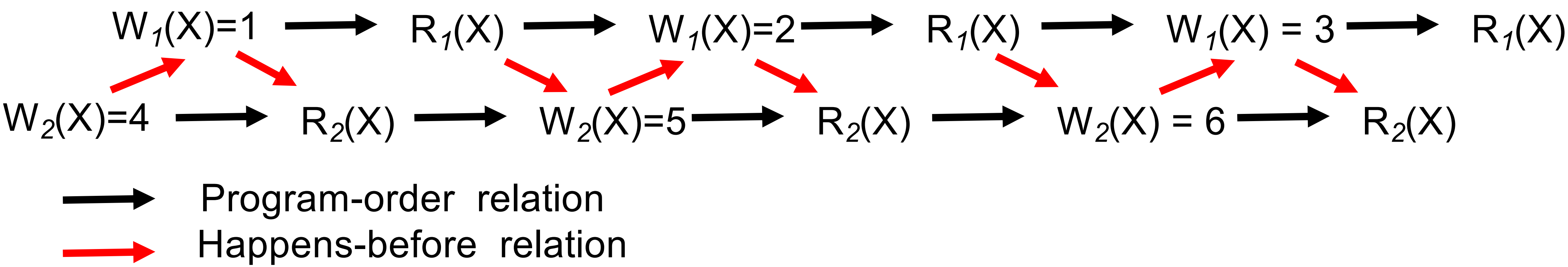
Latest Write to a Location: Two Processors – One Scenario



Question:
What value is returned by the last read request from processor 1?

Answer:
Now it's unambiguously 6 because the last write from Processor 2 is the last write.

Latest Write to a Location: Two Processors – Another Scenario

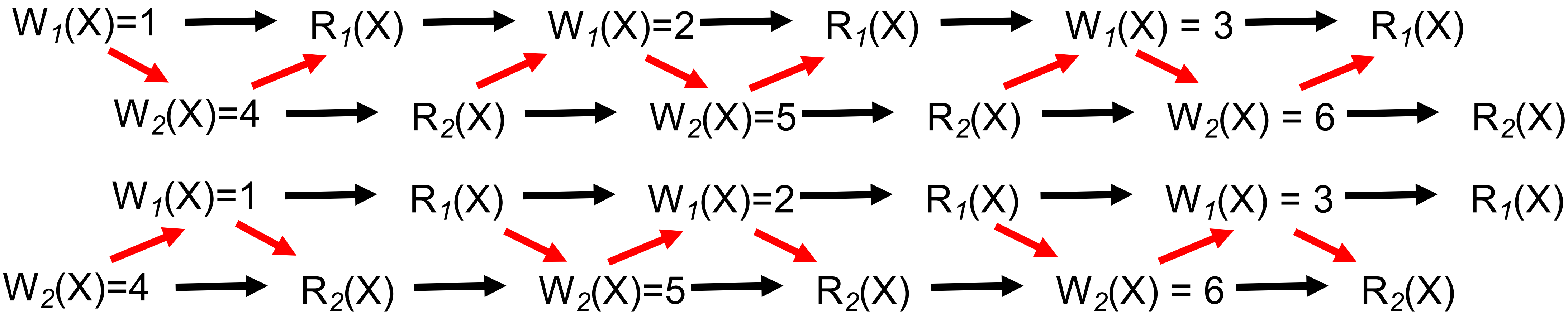


Question:
What value is returned by the last read request from processor 2?

Answer:
It's unambiguously 3 and the last write from Processor 1 is the last write.

Definition: Cache Coherence

Two executions:



For any execution (beyond the two shown) a memory system is coherent if:

- Memory operations form **one *serial order*** with respect to the ***same*** memory location
- Memory operations from each thread respect ***program order***
- The value returned by a read is the value of the latest write in the ***serial order***

Why Coherence is Important!

Assume A=0, initially

P1		**P2**	
A=1	$W_1(A)=1$	..=A	$R_{21}(A)$
A=2	$W_1(A)=2$	..=A	$R_{22}(A)$

Programmer expectation: ..

If first read from P2 returns a value X, second read must return a value >=X

Examples of **correct** (coherent) interleavings:

$W_1(A)=1 \rightarrow R_{21}(A) \rightarrow W_1(A)=2 \rightarrow R_{22}(A)$

$R_{21}(A) \rightarrow W_1(A)=1 \rightarrow W_1(A)=2 \rightarrow R_{22}(A)$

Examples of **incorrect** (incoherent) interleavings:

$W_1(A)=2 \rightarrow R_{21}(A) \rightarrow W_1(A)=1 \rightarrow R_{22}(A)$

$W_1(A)=1 \rightarrow R_{22}(A) \rightarrow W_1(A)=2 \rightarrow R_{21}(A)$

Assume A=0, initially

P1

A=1 $W_1(A)=1$

A=2 $W_1(A)=2$

P2

..=A $R_{21}(A)$

..=A $R_{22}(A)$

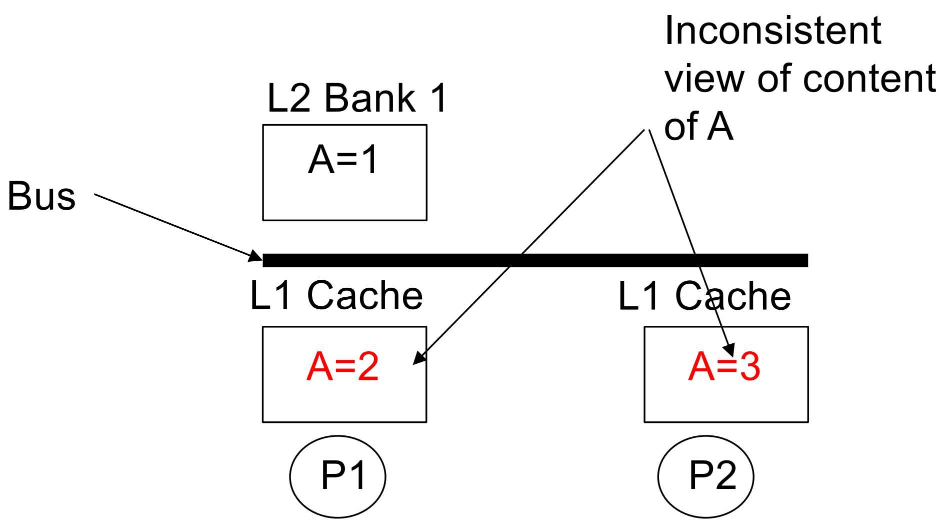
Question:
List all correct serial orders of the program and what R_{21} and R_{22} return in each case.

Answer:

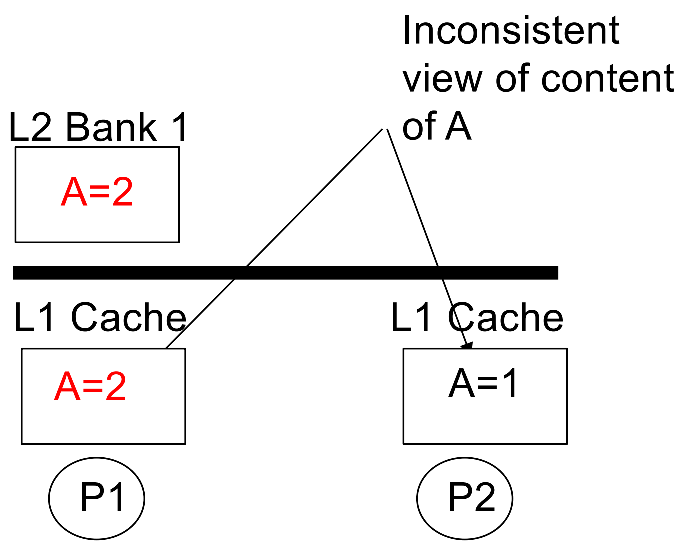
					$R_{21}(A)$	$R_{22}(A)$
S1	$R_{21}(A)$	$R_{22}(A)$	$W_1(A)=1$	$W_1(A)=2$	0	0
S2	$R_{21}(A)$	$W_1(A)=1$	$R_{22}(A)$	$W_1(A)=2$	0	1
S3	$R_{21}(A)$	$W_1(A)=1$	$W_1(A)=2$	$R_{22}(A)$	0	2
S4	$W_1(A)=1$	$R_{21}(A)$	$R_{22}(A)$	$W_1(A)=2$	1	1
S5	$W_1(A)=1$	$R_{21}(A)$	$W_1(A)=2$	$R_{22}(A)$	1	2
S6	$W_1(A)=1$	$W_1(A)=2$	$R_{21}(A)$	$R_{22}(A)$	2	2

Snoopy Cache
Coherence Protocols

The Cache Coherence Problem

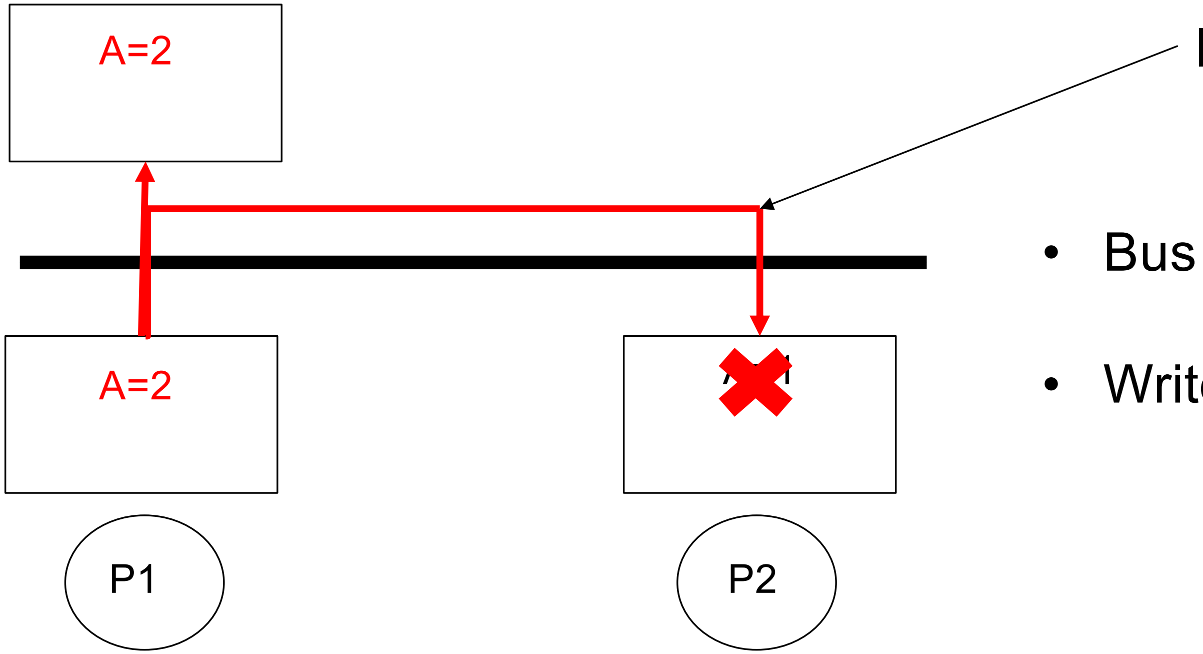


Write-back caches



Write-through caches

A Simple Snoopy Cache Protocol



Write-through caches

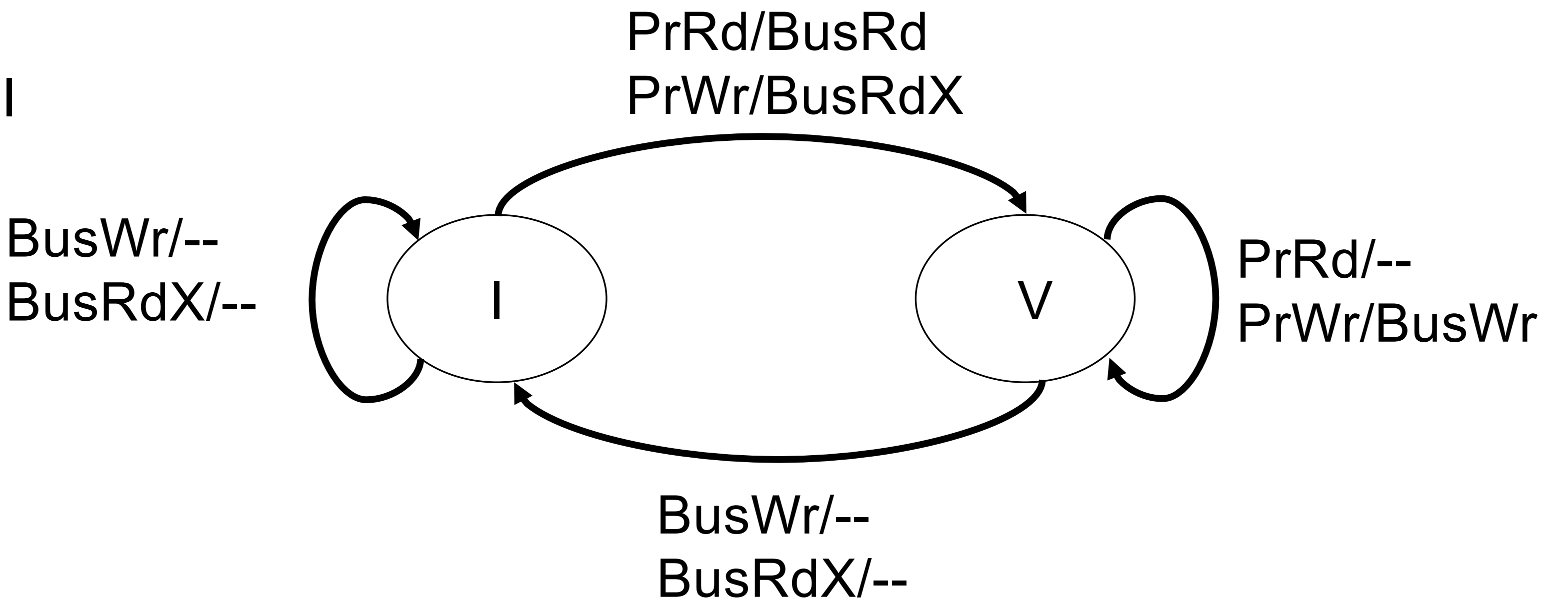
- Bus acts as serialization point
- Write atomicity is maintained

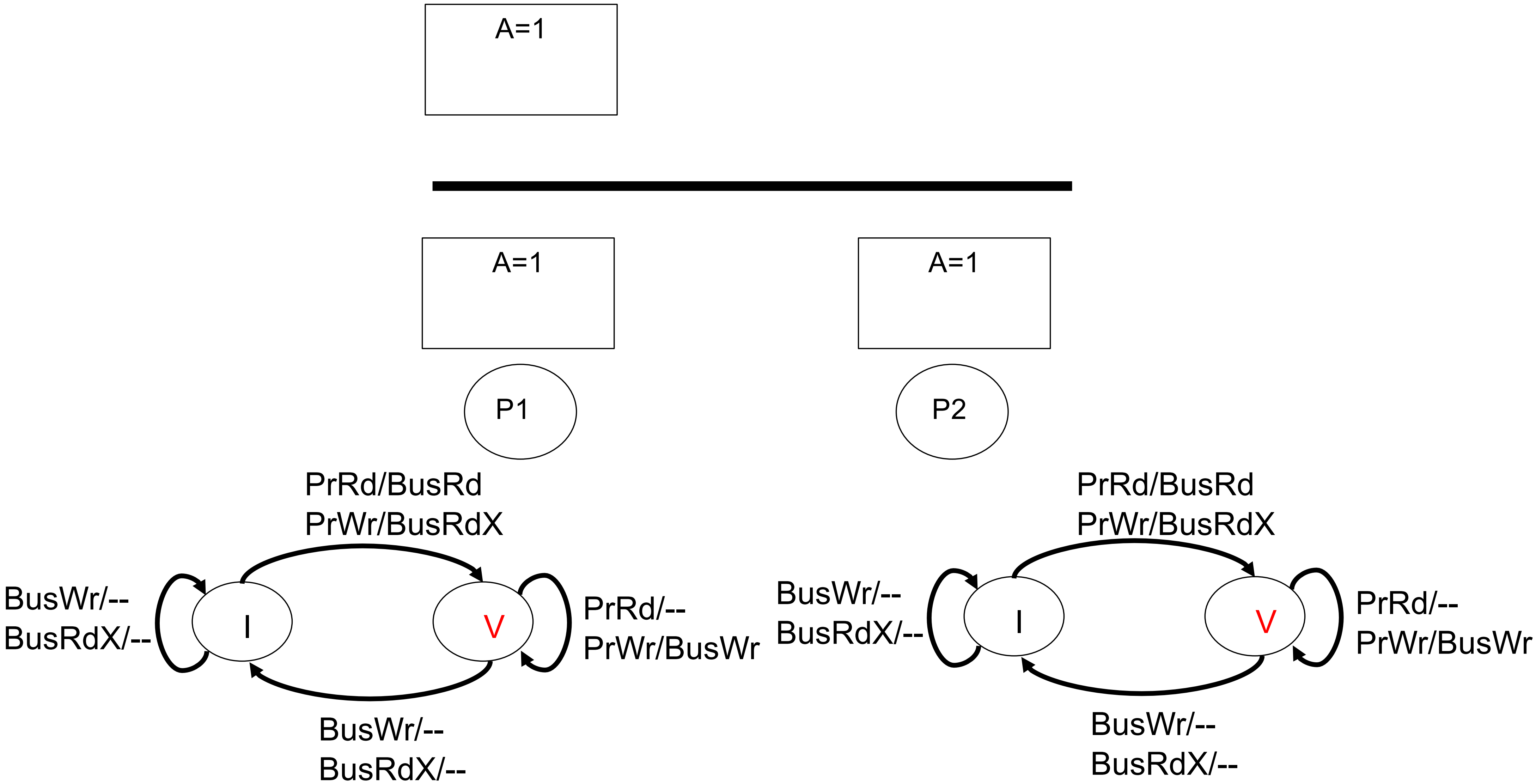
Implementing the Simple Cache Protocol

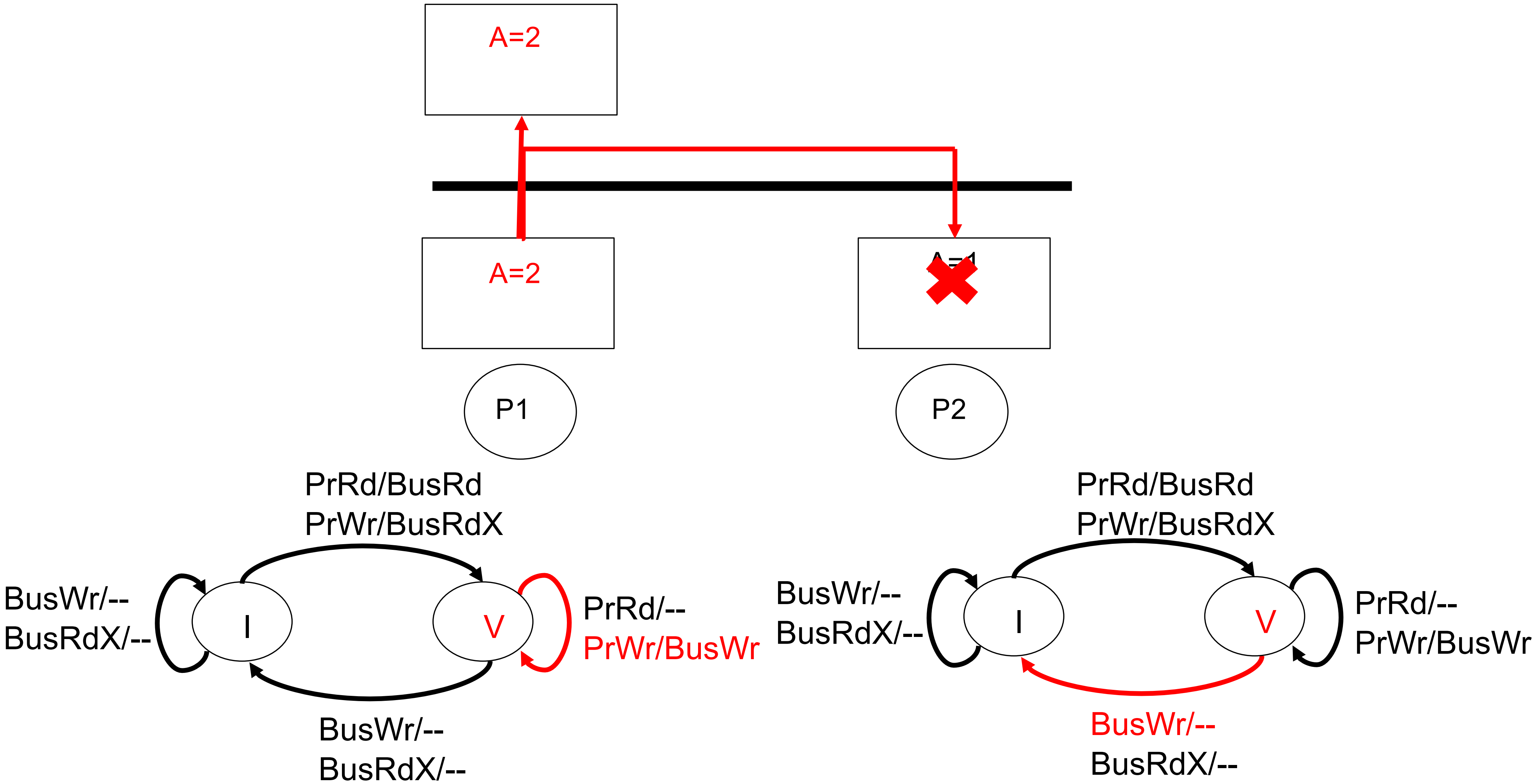
Processor-side requests:
- Processor read – PrRd
- Processor write – PrWr

Bus-side requests:
- Bus read – BusRd
- Bus read exclusive (inval.) - BusRdX
- Bus write (inval.) – BusWr

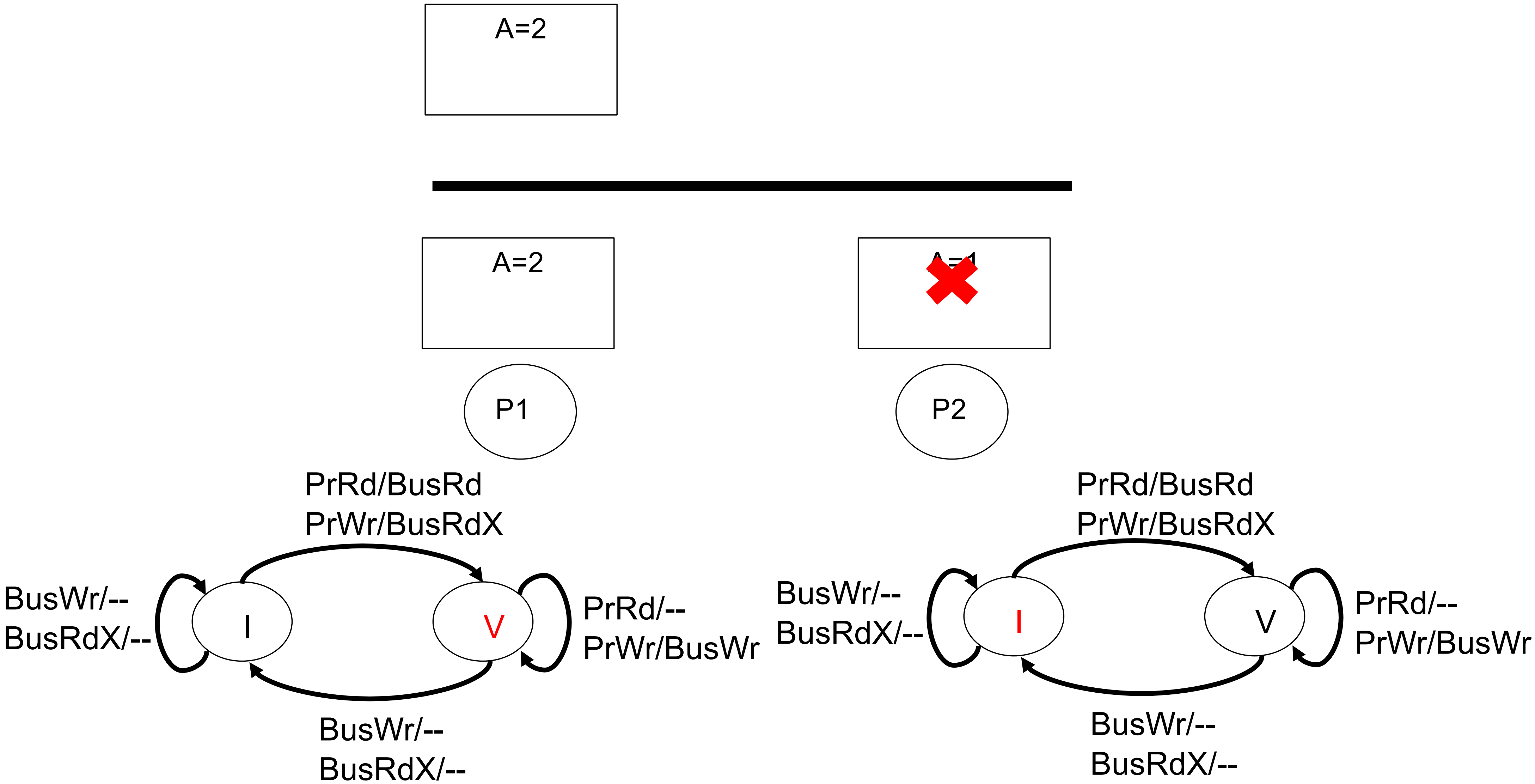
Cache states per block:
- Valid – V
- InValid – I

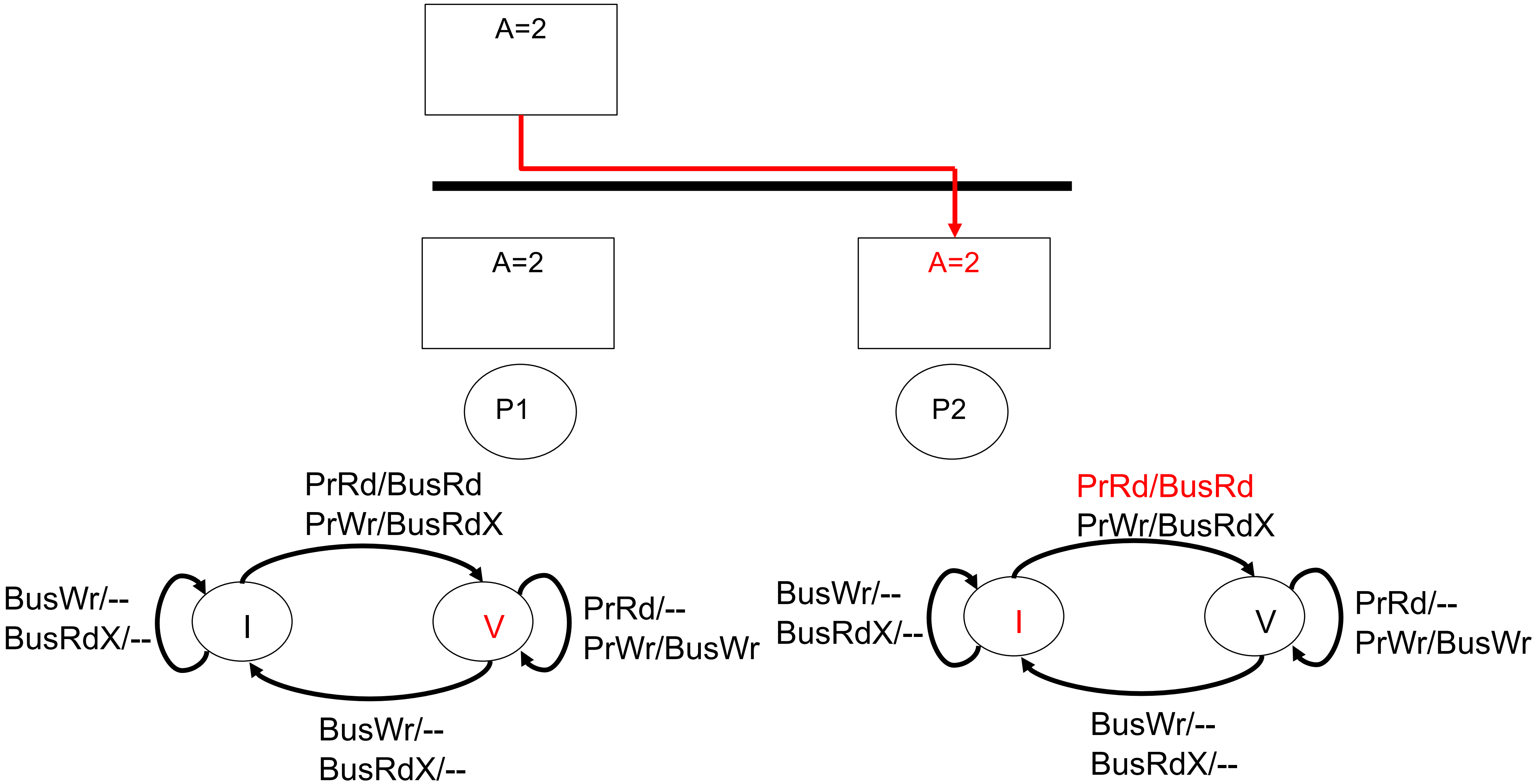


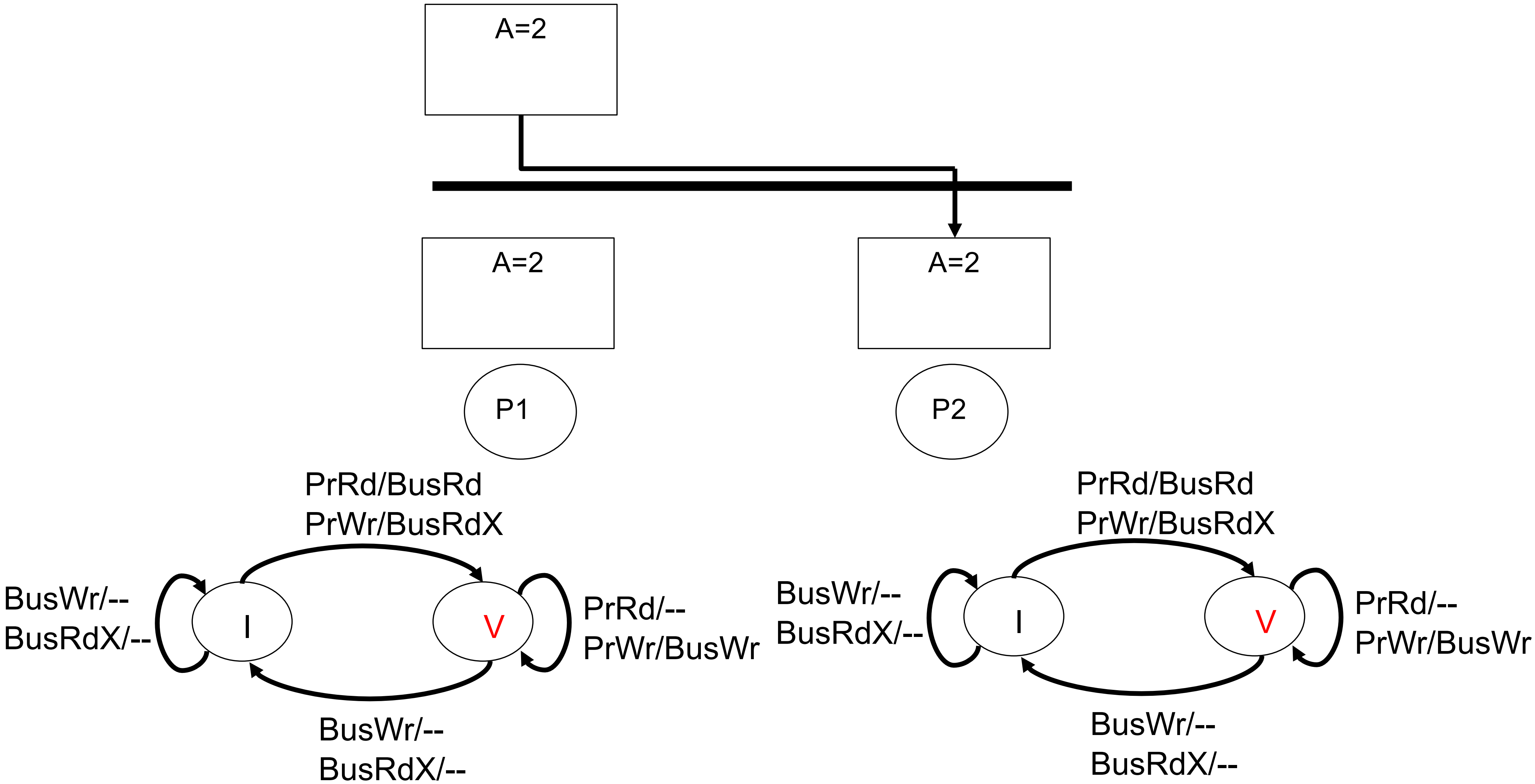

A=1
A=1
A=1
P1
P2
PrRd/BusRd
PrWr/BusRdX
BusWr/--
BusRdX/--
I
V
PrRd/--
PrWr/BusWr
BusWr/--
BusRdX/--
PrRd/BusRd
PrWr/BusRdX
BusWr/--
BusRdX/--
I
V
PrRd/--
PrWr/BusWr
BusWr/--
BusRdX/--



A=2
A=2
A=1
P1
P2
PrRd/BusRd
PrWr/BusRdX
BusWr/--
BusRdX/--
I
V
PrRd/--
PrWr/BusWr
BusWr/--
BusRdX/--
PrRd/BusRd
PrWr/BusRdX
BusWr/--
BusRdX/--
I
V
PrRd/--
PrWr/BusWr
BusWr/--
BusRdX/--

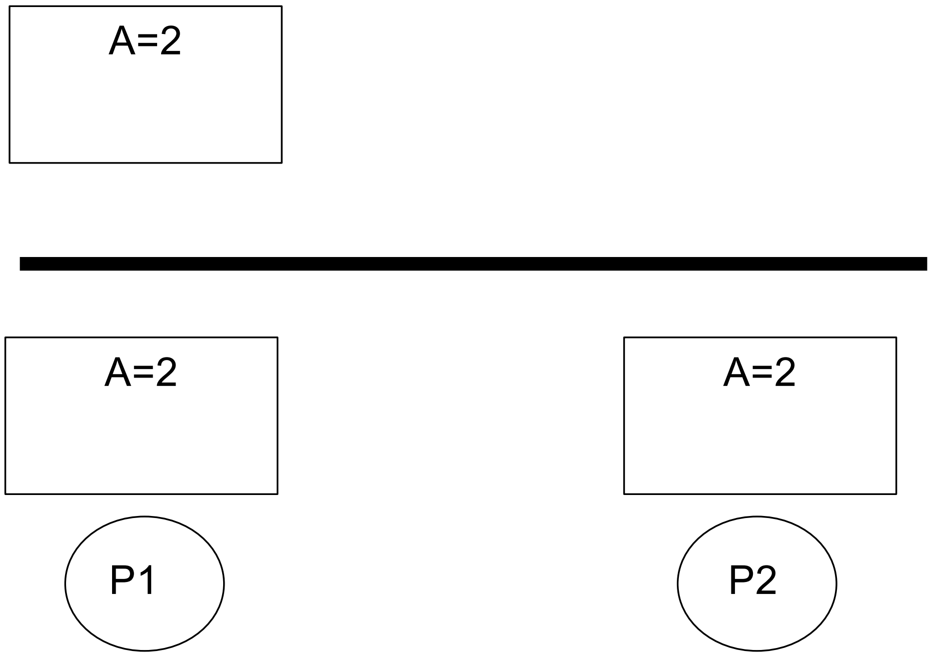

A=2
A=2
P1
P2
PrRd/BusRd
PrWr/BusRdX
BusWr/--
BusRdX/--
I
V
PrRd/--
PrWr/BusWr
BusWr/--
BusRdX/--
PrRd/BusRd
PrWr/BusRdX
BusWr/--
BusRdX/--
I
V
PrRd/--
PrWr/BusWr
BusWr/--
BusRdX/--

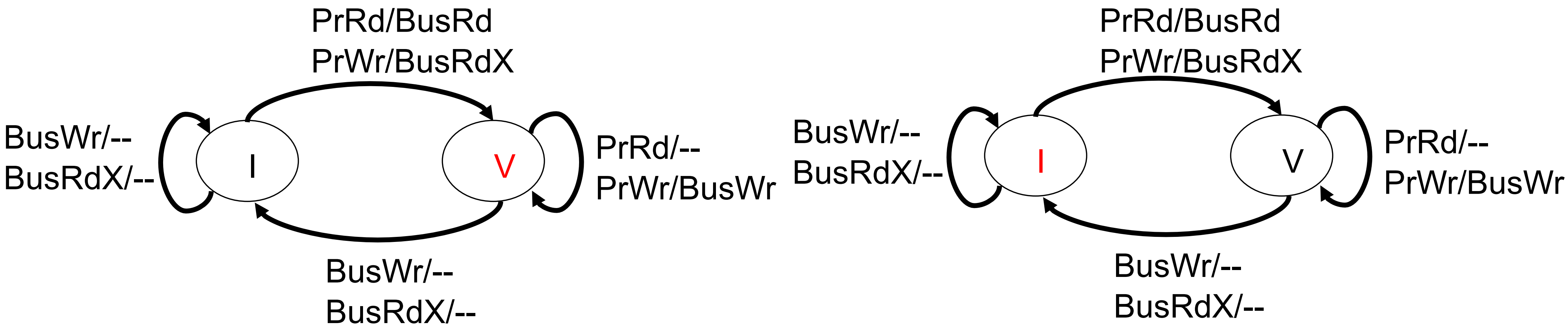

A=2
A=2
A=2
P1
P2
PrRd/BusRd
PrWr/BusRdX
BusWr/--
BusRdX/--
I
V
PrRd/--
PrWr/BusWr
BusWr/--
BusRdX/--
PrRd/BusRd
PrWr/BusRdX
BusWr/--
BusRdX/--
I
V
PrRd/--
PrWr/BusWr
BusWr/--
BusRdX/--

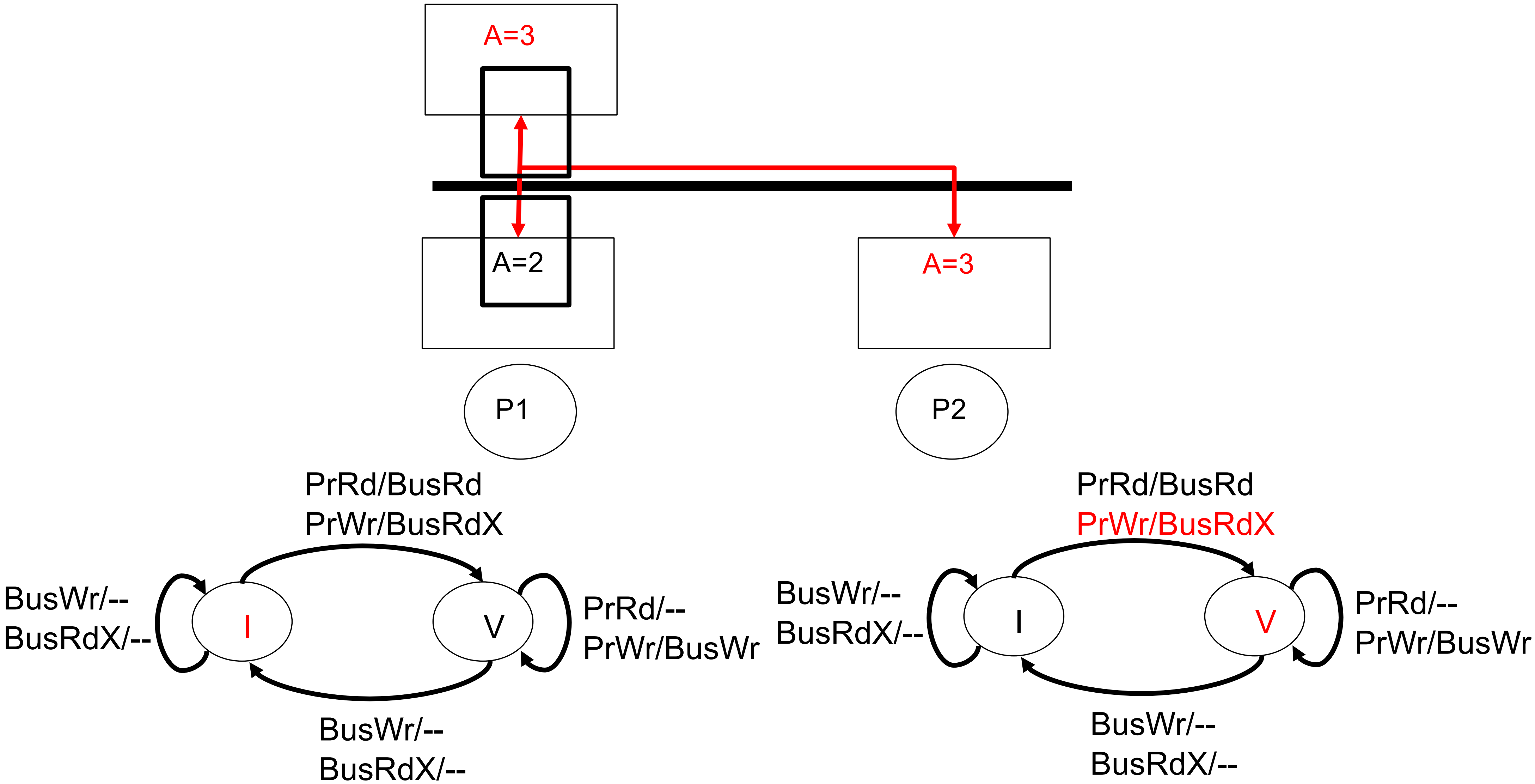


A=2
A=2
A=2
P1
P2
PrRd/BusRd
PrWr/BusRdX
BusWr/--
BusRdX/--
I
V
PrRd/--
PrWr/BusWr
BusWr/--
BusRdX/--
PrRd/BusRd
PrWr/BusRdX
BusWr/--
BusRdX/--
I
V
PrRd/--
PrWr/BusWr
BusWr/--
BusRdX/--

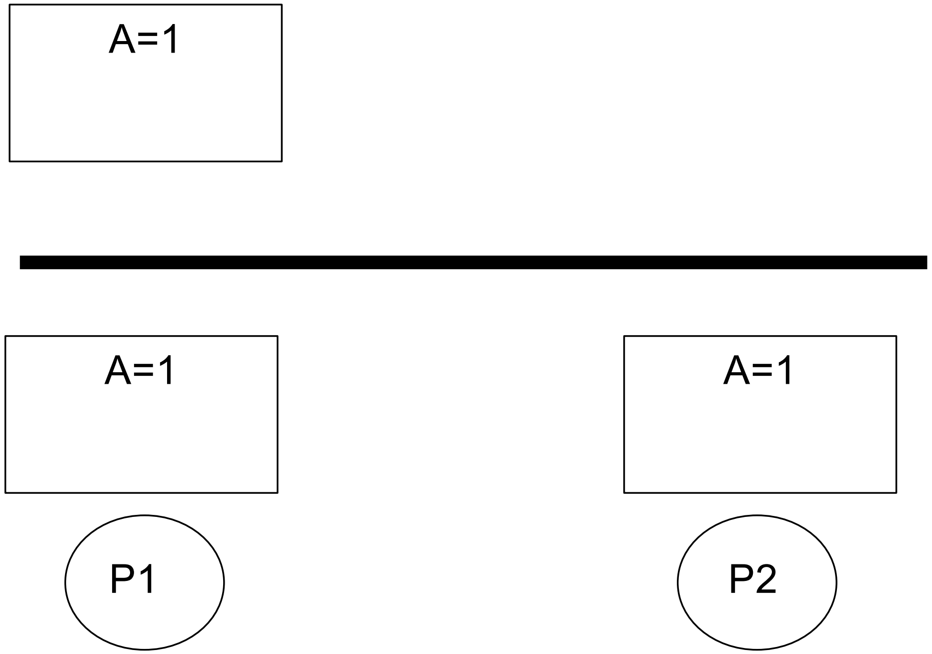
Question:
What transactions are launched in response to a write request issued by P2?






A=3
A=2
A=3
P1
P2
PrRd/BusRd
PrWr/BusRdX
BusWr/--
BusRdX/--
I
V
PrRd/--
PrWr/BusWr
BusWr/--
BusRdX/--
PrRd/BusRd
PrWr/BusRdX
BusWr/--
BusRdX/--
I
V
PrRd/--
PrWr/BusWr
BusWr/--
BusRdX/--

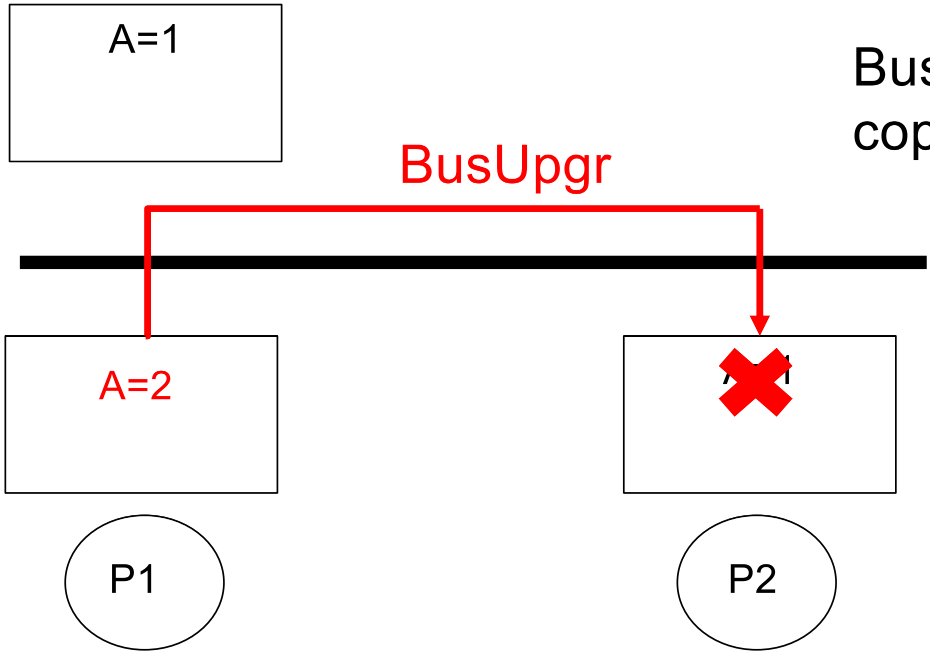
MSI – A Write-Back Cache-Protocol 1(3)



Write-back caches

- Writes carried out locally if block is **not** shared

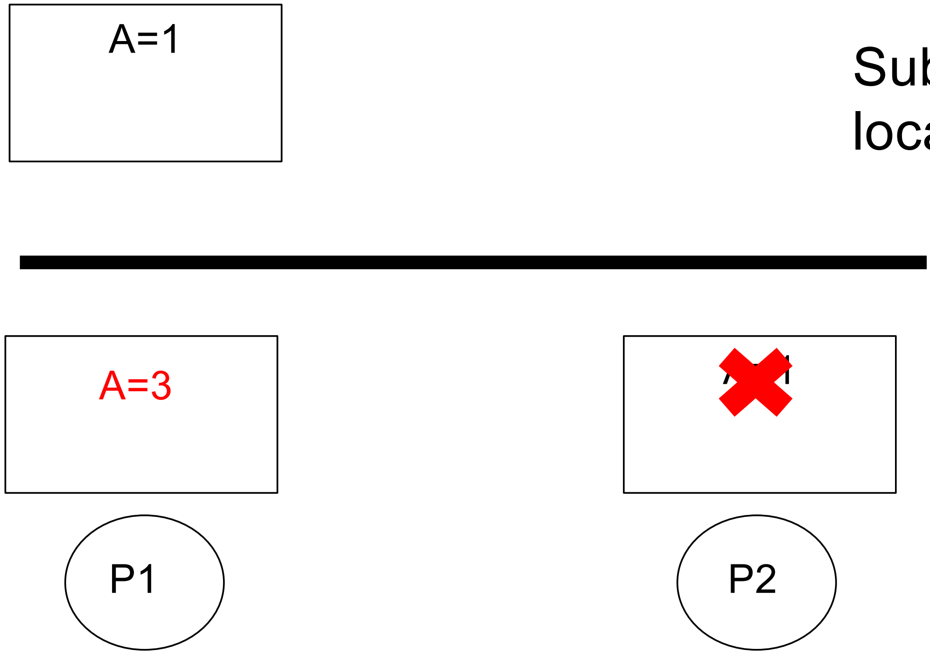
MSI – A Write-Back Cache-Protocol 2(3)



BusUpgrade request invalidates other copies, including memory

Write-back caches

MSI – A Write-Back Cache-Protocol 3(3)



Subsequent writes from P1 happen locally in P1's cache

Write-back caches

MSI – State Transition Diagram