

# Shared Memory Systems

**Randal E. Bryant**

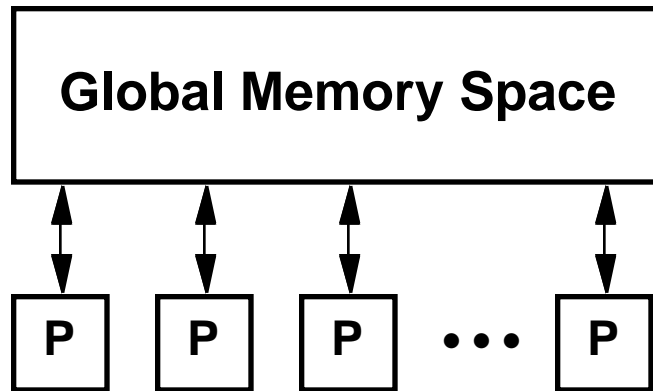
**CS 740**

**Nov. 19, 1997**

## **Topics**

- **Consistency Models**
- **Single Bus Systems**
  - Coherence based on snooping
- **Synchronization Programs**
- **Network-Based Systems**
  - Coherence based on directories

# Shared Memory Model



## Conceptual View

- **All processors access single memory**
  - Physical address space
  - Use virtual address mapping to partition among processes
- **If one processor updates location, then all will see it**
  - Memory consistency

# Bus-Based Realization

## Memory Bus

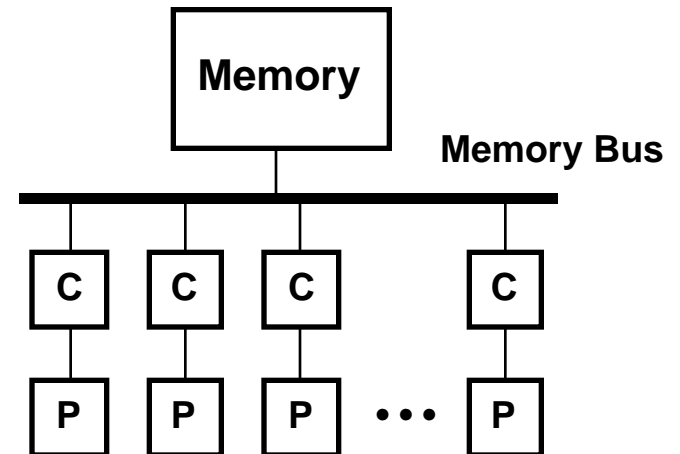
- Handles all accesses to shared memory

## Caches

- One per processor
- Allows local copies of heavily used data
- Must avoid stale data

## Considerations

- **Small step up from single processor system**
  - Support added to many microprocessor chips
- **Does not scale well**
  - Bus becomes bottleneck
  - Limited to ~16 processors



# Network-Based Realization

## Memory

- Partitioned Among Processors

## Network

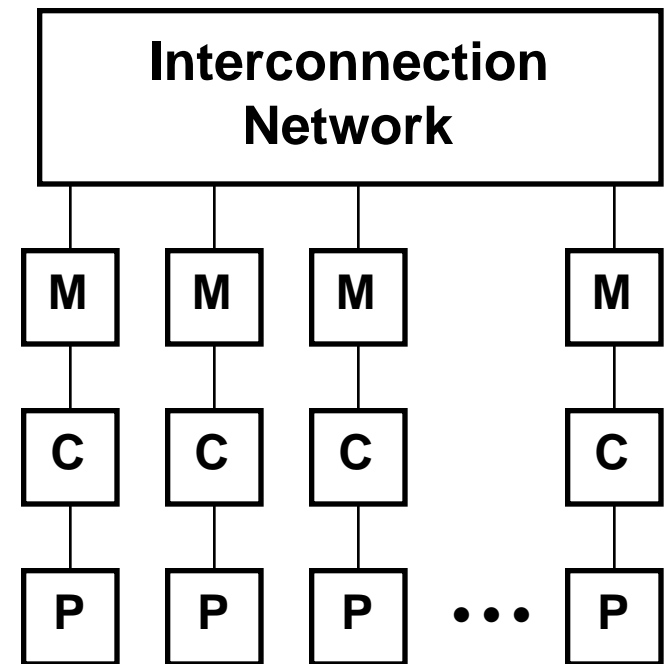
- Transmit messages to perform accesses to remote memories

## Caches

- Local copies of heavily used data
- Must avoid stale data
  - Harder than with bus-based system
  - Lots of things happening simultaneously

## Considerations

- Scales well
  - 1024 processor systems have been built
- Nonuniform memory access
  - 100's of cycles for remote access



# Memory Consistency

## Model

- Independent processes with access to shared variables
- No assumptions about relative timing of processes
  - Which starts first
  - Which runs fastest

## Sequential Consistency

- Each process executes its steps in program order
- Overall effect should match that of some interleaving of the individual process steps

Initially:  $x = y = 0$

### Process A

a1:  $x = 1$

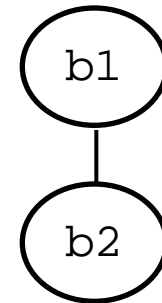
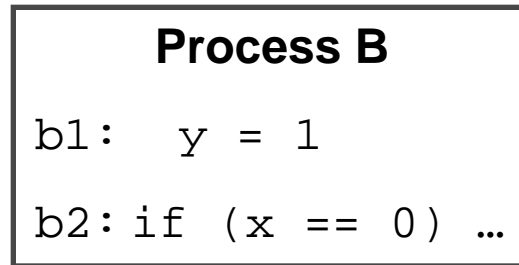
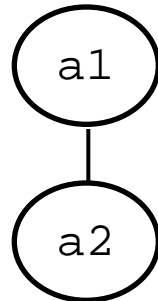
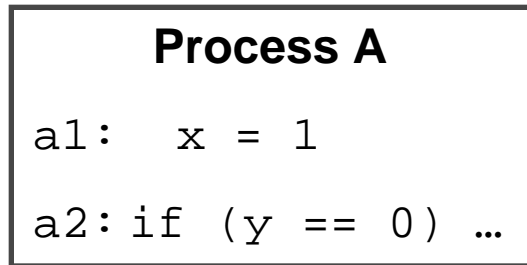
a2: `if (y == 0) ...`

### Process B

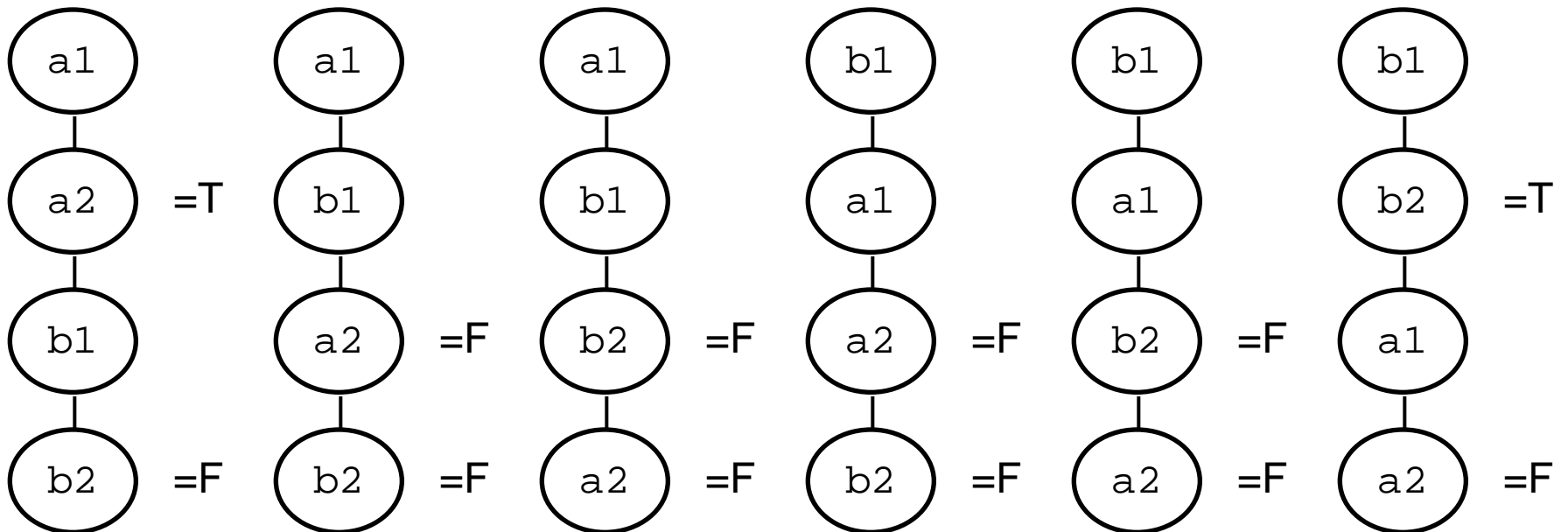
b1:  $y = 1$

b2: `if (x == 0) ...`

# Sequential Consistency Example

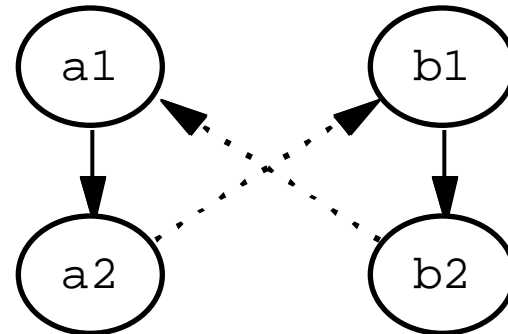


## Possible Interleavings



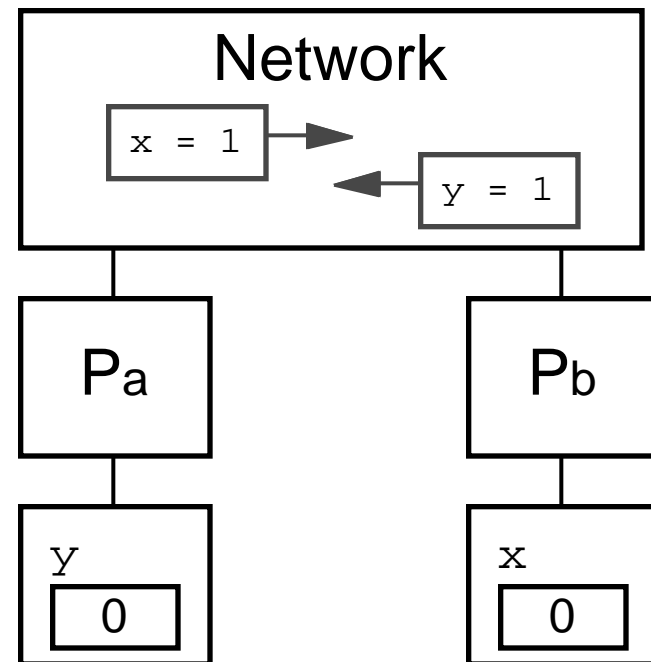
# Sequential Inconsistency

- **Cannot have both tests yield T**
  - b2 must precede a1
  - a2 must precede b1
  - Cannot satisfy these plus program order constraints



## Real Life Scenario

- **Process A**
  - Remote write x
  - Local read y
- **Process B**
  - Remote write y
  - Local read x
- **Could have both reads yield 0**



# Snoopy Bus-Based Consistency

## Caches

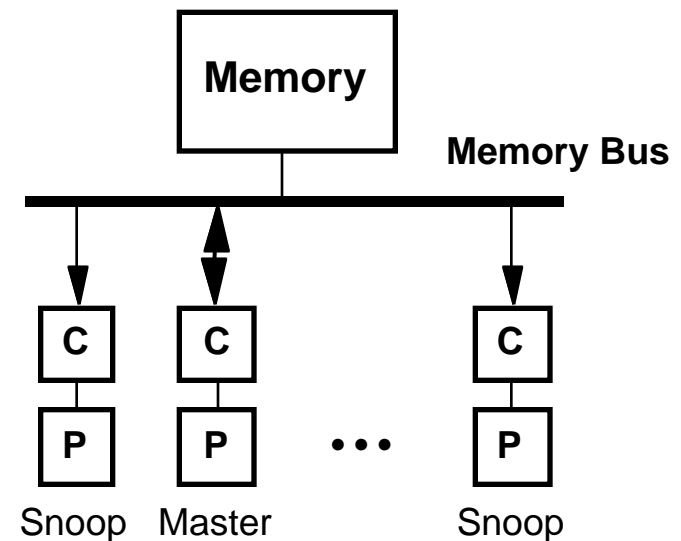
- **Write-back**
  - Minimize bus traffic
- **Monitor bus transactions when not master**

## Cached blocks

- **Clean block can have multiple, read-only copies**
- **To write, must obtain exclusive copy**
  - Marked as dirty

## Getting copy

- **Make bus request**
- **Memory replies if block clean**
- **Owning cache replies if dirty**



# Implementation Details

## Block Status

- **Maintained by each cache for each of its blocks**
- **Invalid**
  - Entry not valid
- **Clean**
  - Valid, read-only copy
  - Matches copy in main memory
- **Dirty**
  - Exclusive, writeable copy
  - Must write back to evict

## Bus Operations

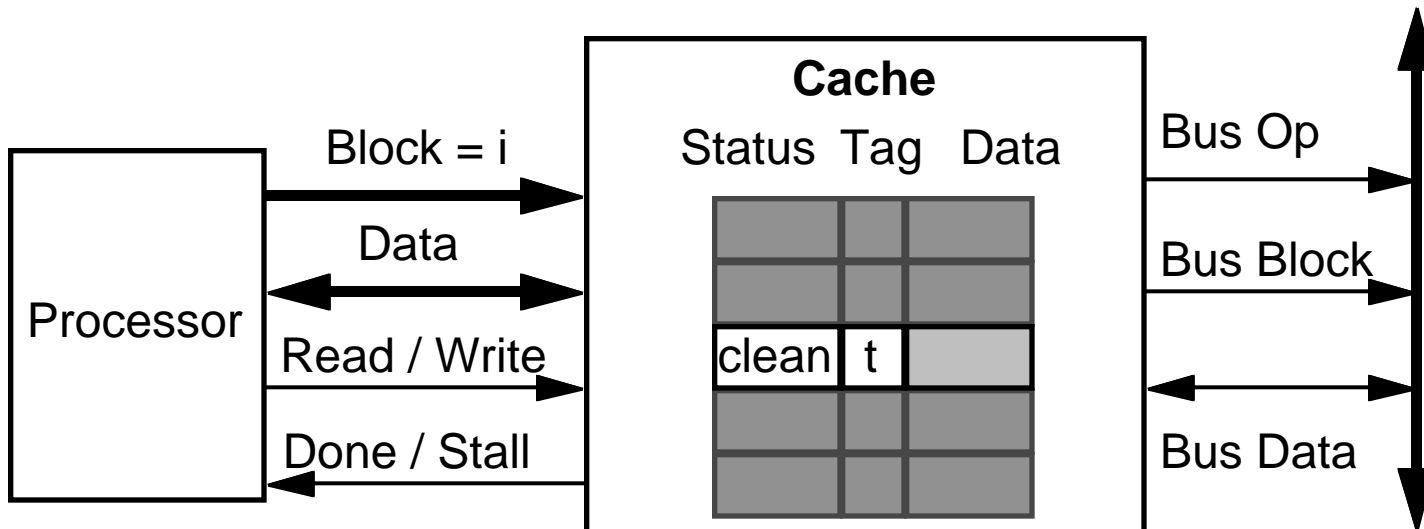
- **Read**
  - Get read-only copy
- **Invalidate**
  - Invalidate all other copies
  - Make local copy writeable
- **Write**
  - Write back dirty block
  - To make room for different block

# Performing Processor Operations

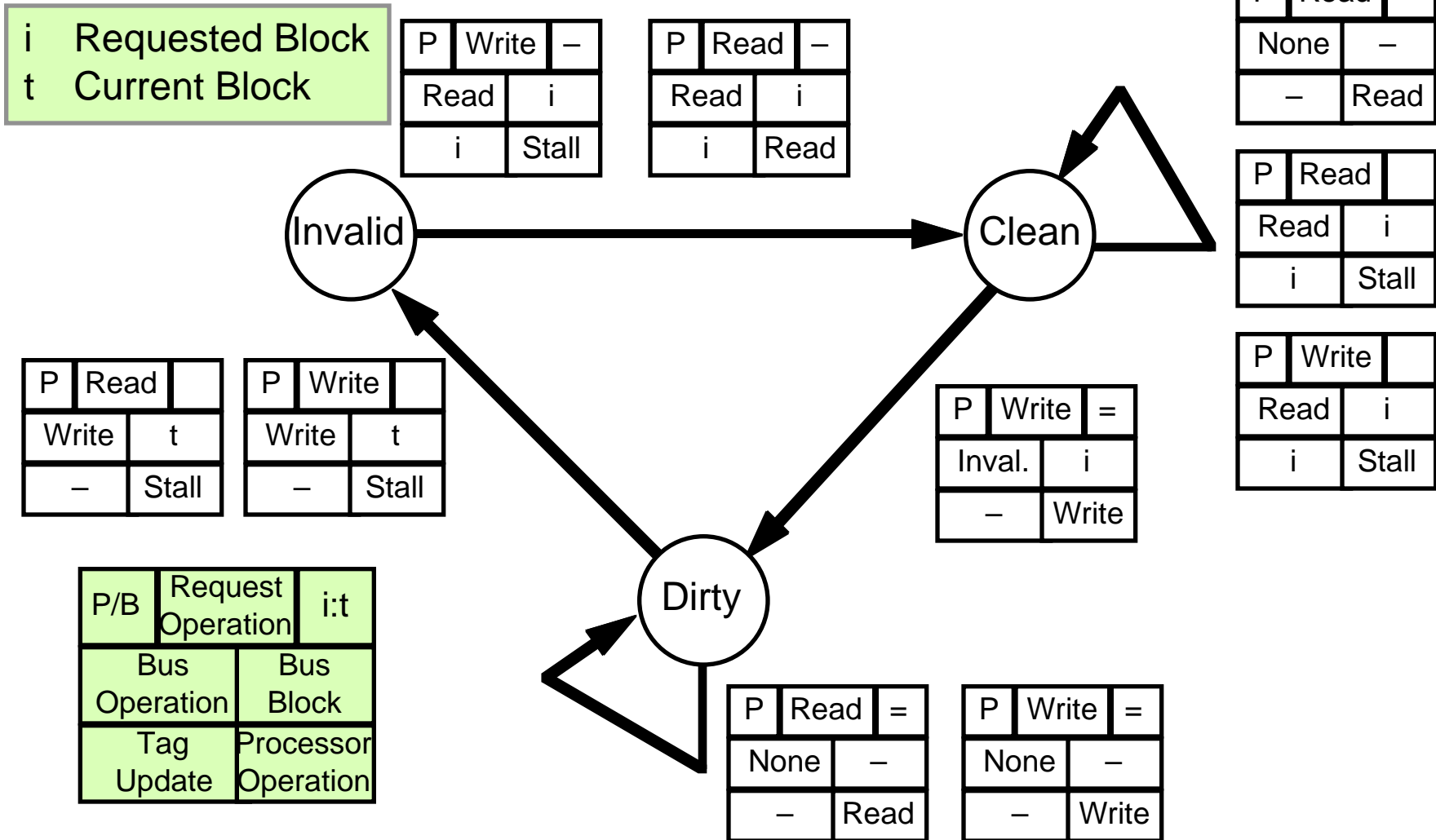
- **Processor requests cache to perform load or store**
  - On word in cache block  $i$
- **Cache line currently holds block  $t$** 
  - May or may not have  $i = t$
- **Cache can either:**
  - Perform operation using local copy
  - Issue bus request to get block
    - » Stall processor until block ready

## Action Key

P/B	Request Operation	$i:t$
	Bus Operation	Bus Block
	Tag Update	Processor Operation



# Bus Master Actions

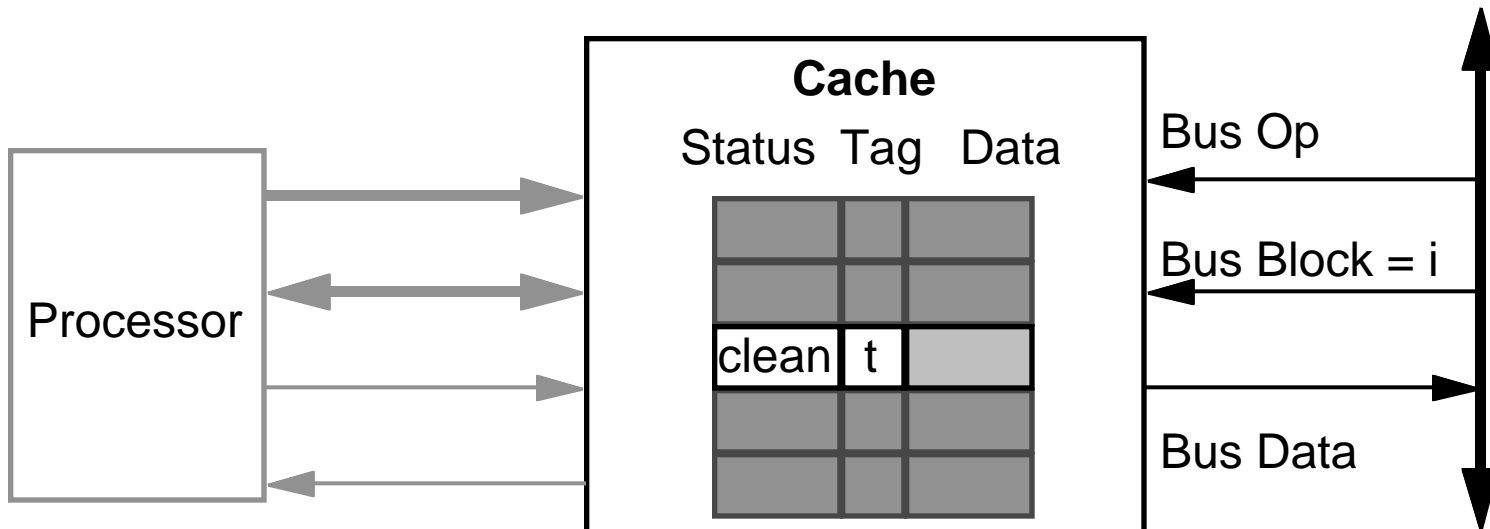


# Bus Monitoring

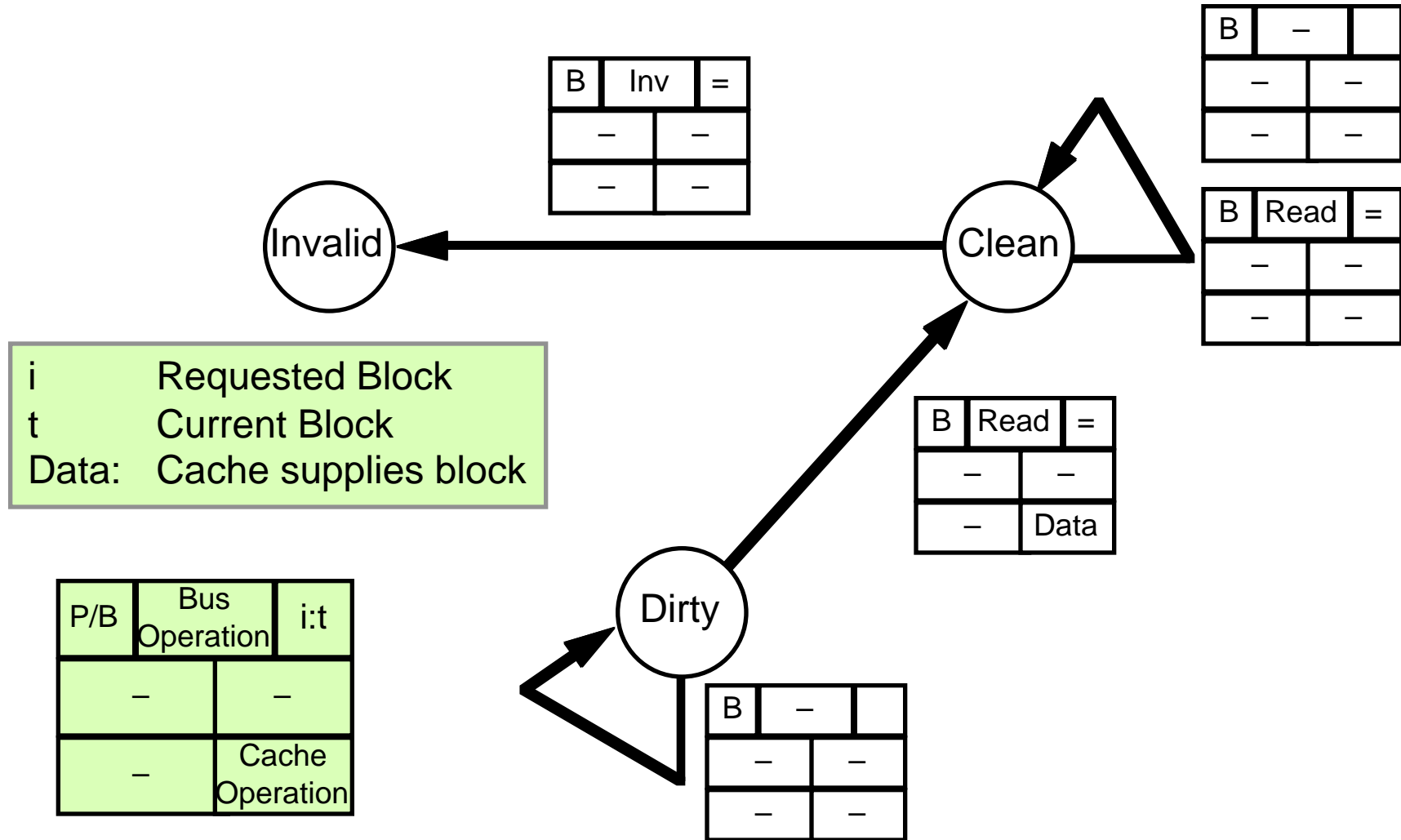
- **Cache monitors bus traffic when not master**
  - Looks for operations on blocks matching cache entries
- **Possible actions**
  - Invalidate entry
  - Allow sharing of exclusively held block
    - » Supply data on bus

## Action Key

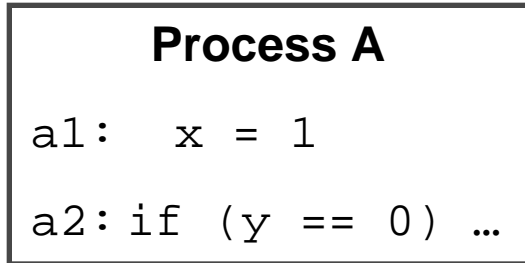
P/B	Bus Operation	i:t
–	–	–
–	–	Cache Operation



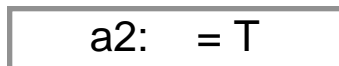
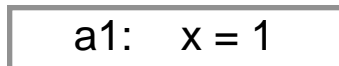
# Bus Snoop Actions



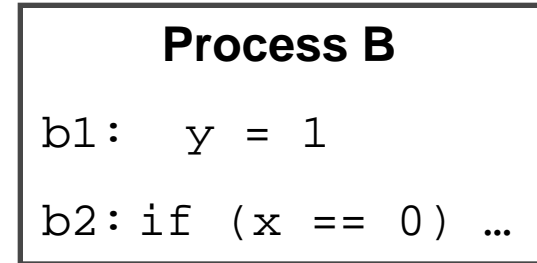
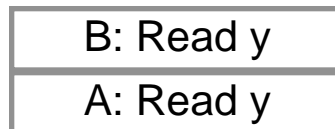
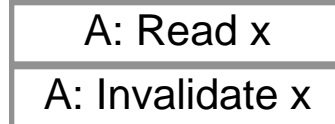
# Example 1



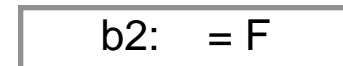
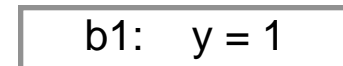
**A**



**Bus Transactions**



**B**



# Example 2

**Process A**  
a1: x = 1  
a2: if (y == 0) ...

**A**

a1: x = 1

a2: = F

**Process B**  
b1: y = 1  
b2: if (x == 0) ...

**B**

b1: y = 1

b2: = F

**Bus Transactions**

A: Read x  
A: Invalidate x

B: Read y  
B: Invalidate y

A: Read y

B: Read x

# Livelock Example

```
Process A  
a1:  y = 0
```

```
Process B  
b1:  while ((t=y) != 0)  
b2:    y = t+1
```

**A**

**Bus Transactions**

**B**

- 
- 
- 

Never gets  
chance to write

```
A: Read y  
B: Read y  
B: Invalidate y
```

```
b1:  t = y  
b2:  y = t+1
```

```
A: Read y  
B: Read y  
B: Invalidate y
```

```
b1:  t = y  
b2:  y = t+1
```

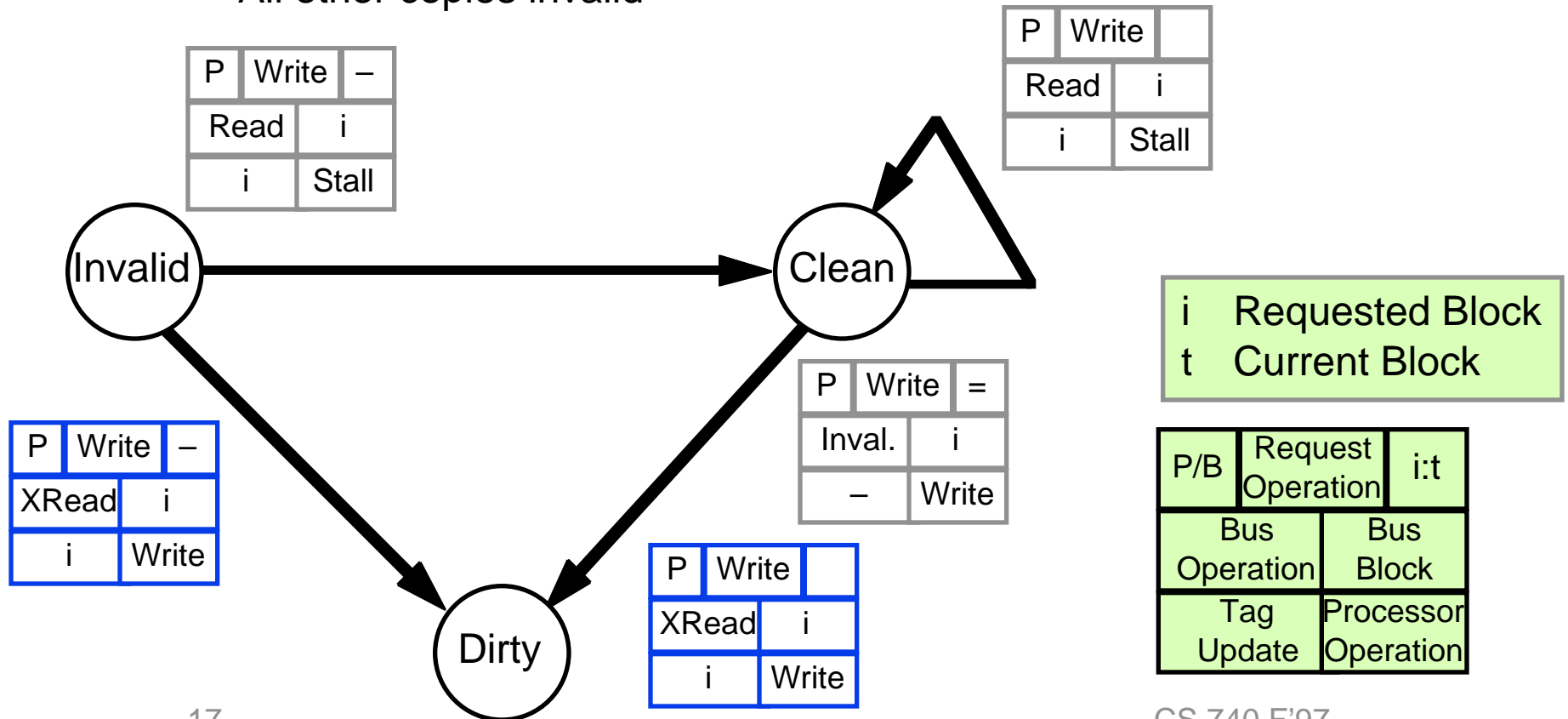
```
A: Read y  
B: Read y
```

- 
- 
-

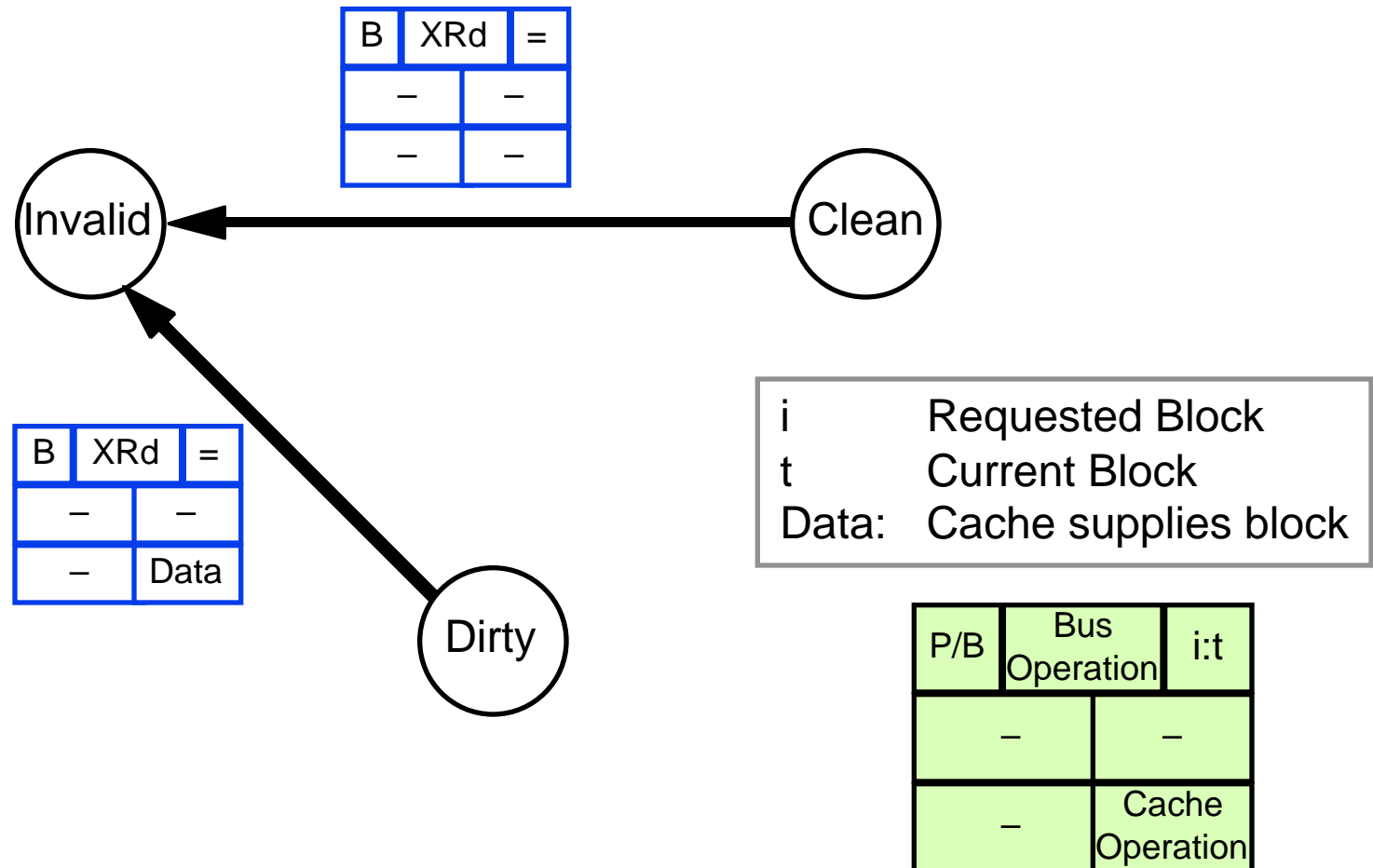
# Livelock Solution

## New Bus Operation

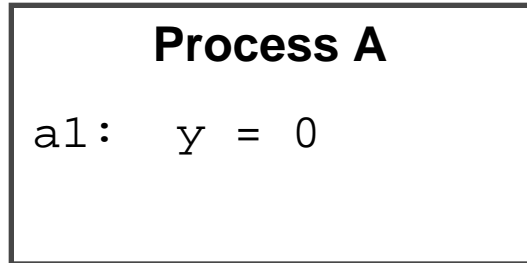
- XRead
  - Combines Read + Invalidate as atomic transaction
    - » Get copy of data
    - » All other copies invalid



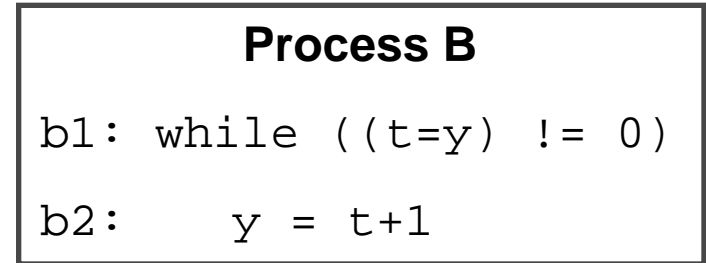
# Livelock Solution (Cont.)



# Prevented Livelock Example

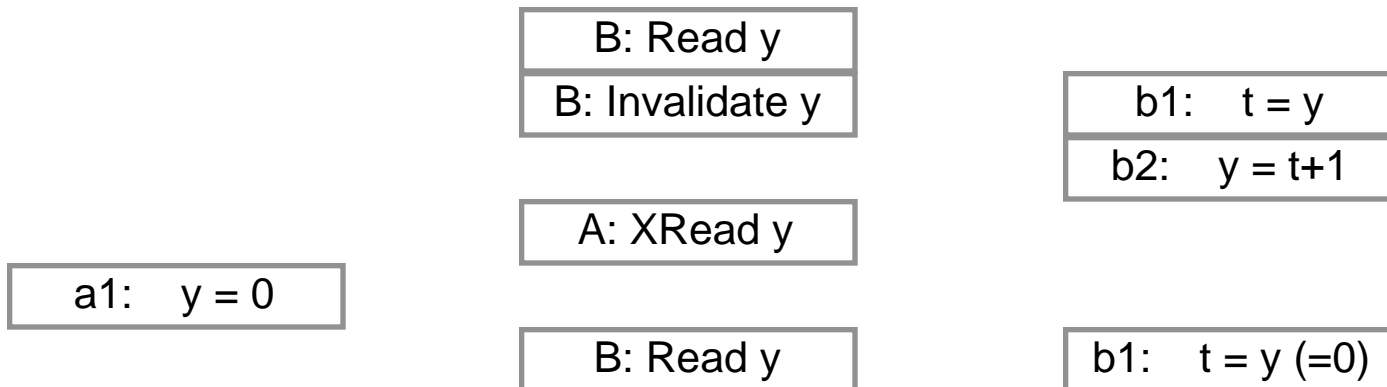


**A**



**B**

**Bus Transactions**



*Fair bus cannot exclude  
request indefinitely*

# Single Bus Machine Example

## SGI Challenge Series

- Up to 36 MIPS R4400 processors
- Up to 16 GB main memory

## Bus

- 256-bit wide data
- 40-bit wide address
- Data transferred at 1.22 GB / second
- Split transaction
  - Read request & Read response are separate bus transactions
  - Can use bus for other things while read outstanding
  - Complicates synchronization

## Performance

- 164 processor cycles to handle remote read
- Assuming no bus contention

# Program Synchronization

## Typical Scenario

- **Processes need access to shared resources**
  - Tables, I/O devices, program state
- **Use software locks to prevent simultaneous access**
  - Shared variables + synchronization conventions

## Naive Synchronization

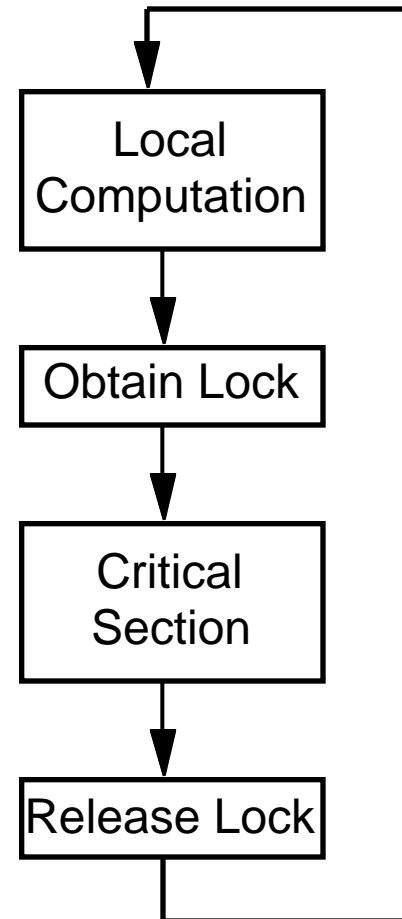
### Obtain #1

```
while (lock != 0)
    ;
lock = 1;
```

### Release

```
lock = 0;
```

Process



# Synchronization with Test-And-Set

## Test-And-Set

```
TST(x):  
    temp = x;  
    x = 1;  
    return temp;
```

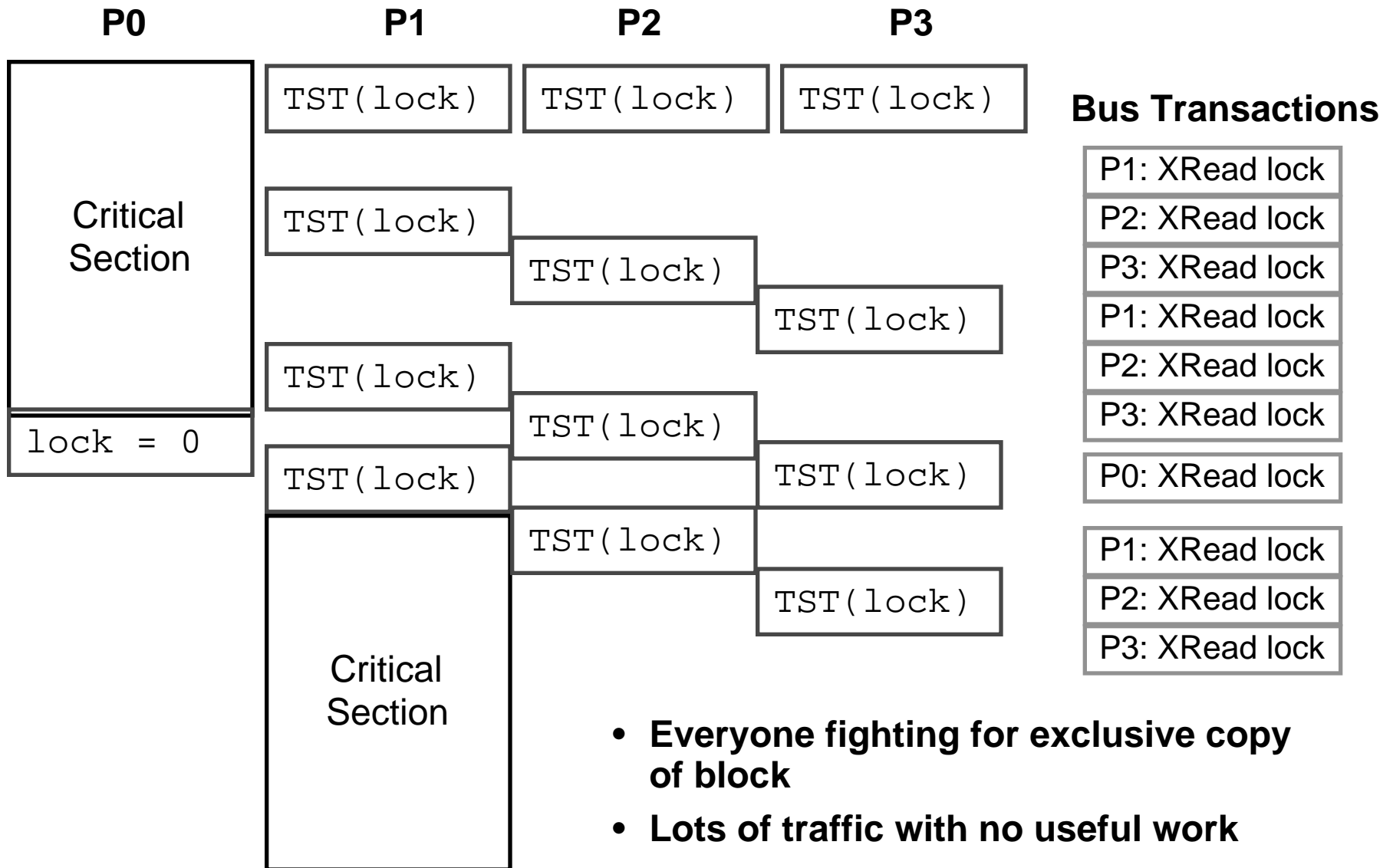
- **Atomically test value of variable and sets it to 1**
- **Could implement with cache protocol in manner similar to write**
  - Even write needs to read in block since only overwrites one word

## Modified Synchronization Code

**Obtain #2**

```
while (TST(lock))  
    ;
```

# Scenario with Obtain #2



# Revised Test-And-Set Synchronization

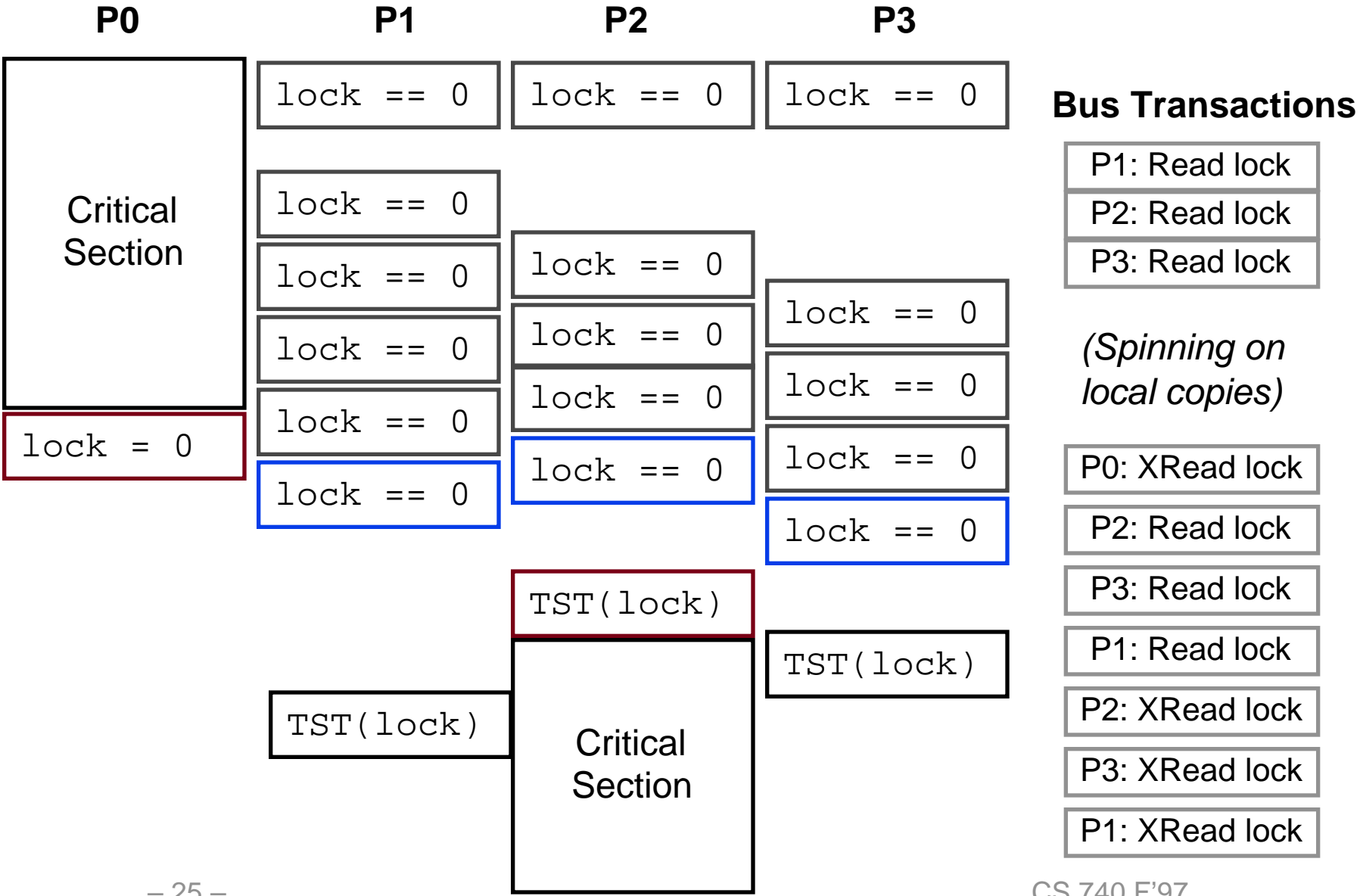
## Modified Synchronization Code

Obtain #3

```
do {  
    while (lock)  
        ;  
} while (TST(lock));
```

- Can used shared (read-only) copy of variable

# Scenario with Obtain #3



# Other Synchronization Primitives

## Atomic Exchange

```
exch Rd, D(Rb)
```

- Atomically swap register Rd and memory location Rb+D
- Implement TST(lock) as:

```
li r2, 1
```

```
exch r2, lock
```

**Obtain #4**

```
loop: ll r3, lock
      bne r3, 0, loop
      li r3, 1
      sc r3, lock
      beq r3, 0, loop
```

## Load-Linked / Store-Conditional

- Used in MIPS

```
ll Rt, D(Rb)
```

- Similar to load
- Set internal “link register” to effective (physical) address

```
sc Rt, D(Rb)
```

- Similar to store
- Abort if effective address matches link register, but this address modified since most recent load-linked
- Indicate success or failure by setting Rt to 1 or 0

# Network-Based Cache Coherency

## Home-Based Protocol

- Each block has “home”
  - Memory controller tracking its status
- Home maintains
  - Block status
  - Identity of copy holders
    - » 1 bit flag / processor

Memory Controller 4									
Block	Status	Copy Holders							
24	shared	0	1	0	1	0	1	0	1
25	remote	0	1	0	0	0	0	0	0
26	uncached	0	0	0	0	0	0	0	0

## Block Status Values

- **Shared**
  - 1 or more remote, read-only copies
- **Remote**
  - Writeable copy in remote cache
- **Uncached**
  - No remote copies

# Network-Based Consistency

## To Obtain Copy of Block

- Processor sends message to its home
- Home retrieves remote copy if status is remote
- Sends copy to requester
- If exclusive copy requested, send invalidate message to all other copy holders

## Tricky Details

- Lots of possible sources of deadlock & errors
- Don't have serialization of events imposed by bus
- Transactions only “seen” by sender & receiver