Snoopy Protocol

Arvind
Computer Science and Artificial Intelligence Lab
M.I.T.

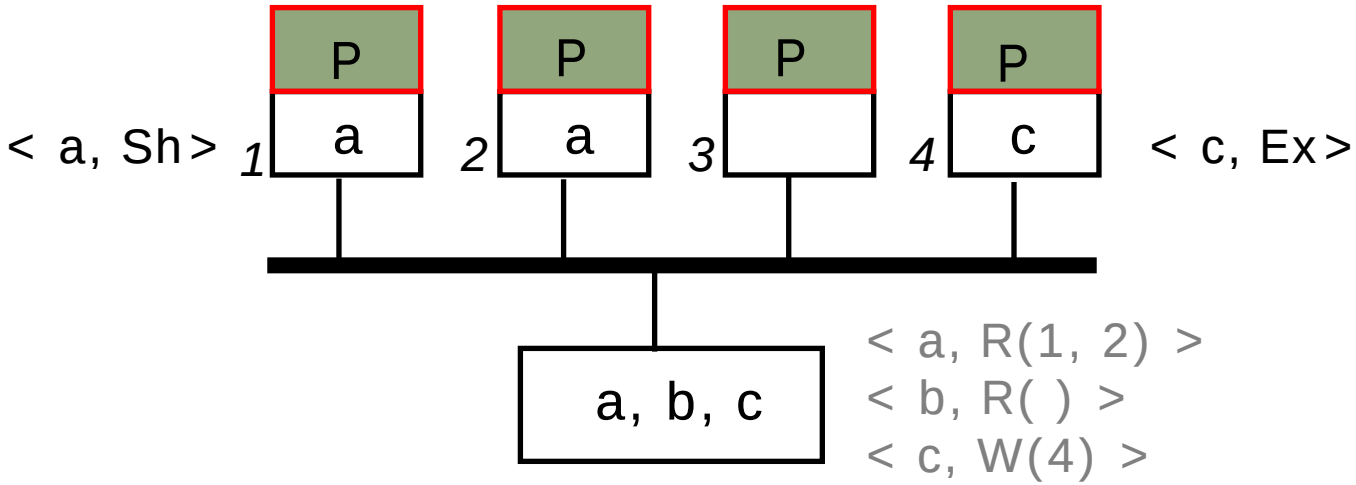
Based on the material prepared by
Arvind and Krste Asanovic

* Note: This lecture note is shorter than usual
in order to finish the material in the previous lecture.

Bus-Based Protocols:

One derived from the directory based protocol

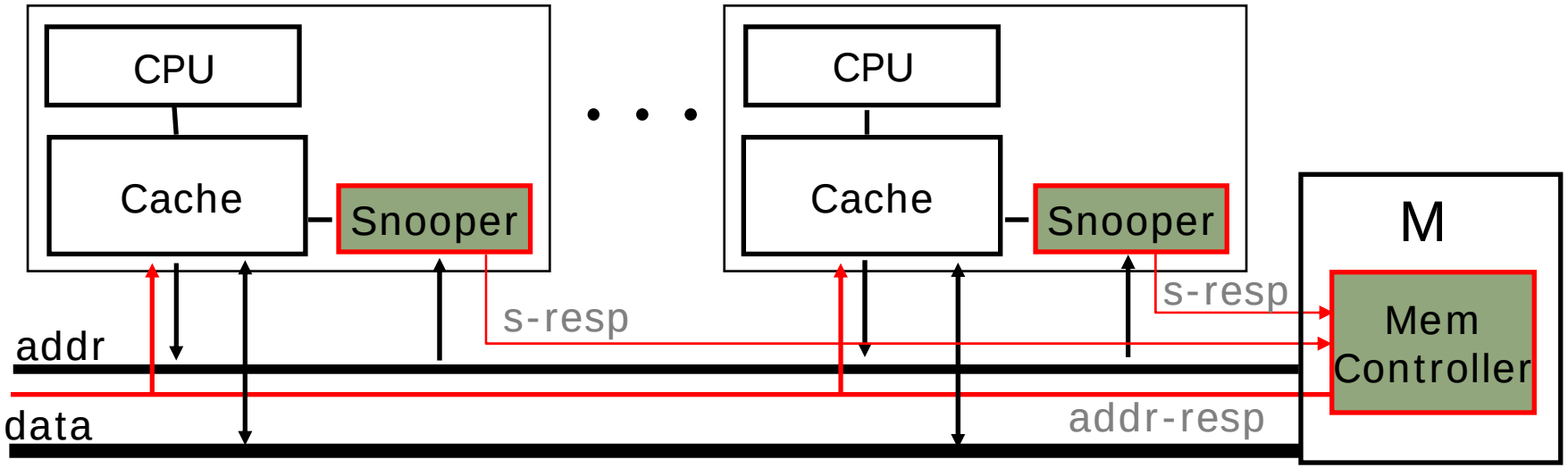
Bus based SMP's



- In a bus based system, it may be more efficient to broadcast the request directly to *all* caches and then collect their responses

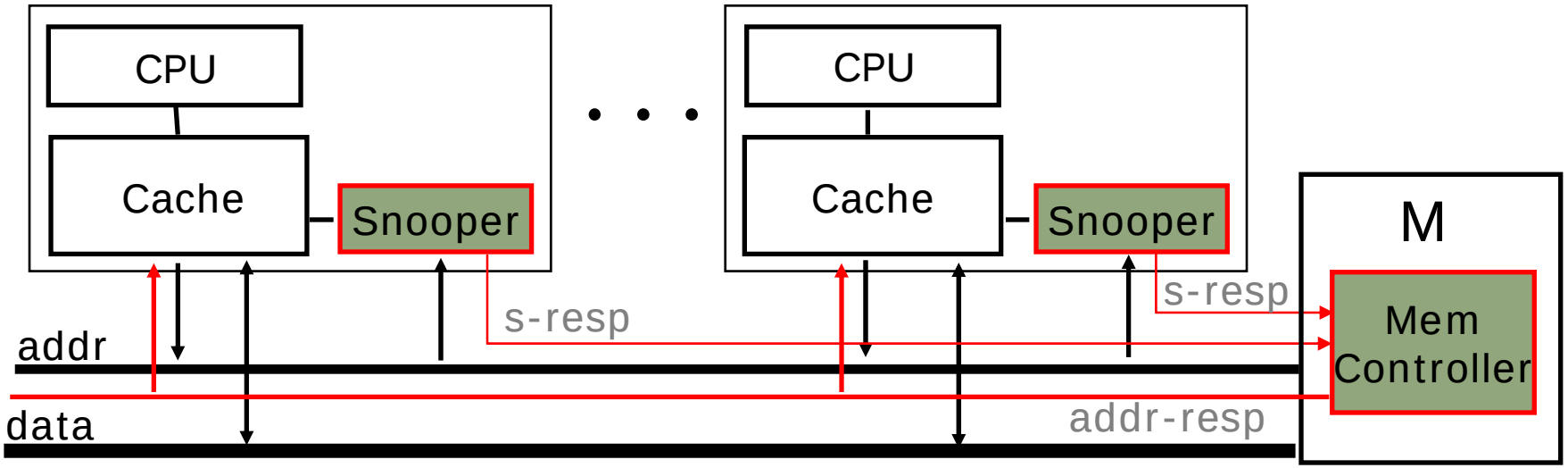
  $\Rightarrow$ *eliminates the need for home directory*

Bus: A Broadcast Medium



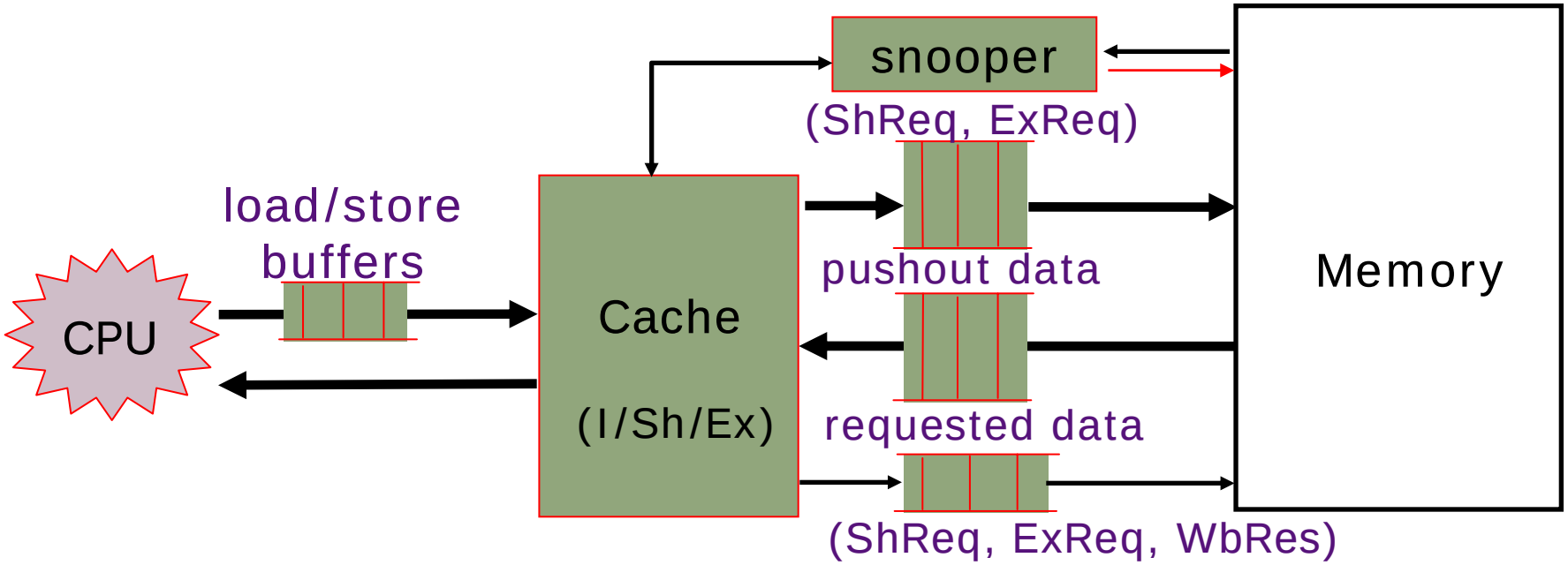
- *Address cycle:* two consecutive phases
  - *request phase:* a processor is selected to issue a request which is assigned a bus tag (i.e. the processor becomes the bus master
  - *response phase:* summary of responses from all the snoopers is returned to the requesting processor
- *Data cycle (if necessary):*
  - The data with its bus tag appear on the data bus
  - The bus tag is retired when the transaction terminates

Snooping on the Bus



- All snoopers listen to the bus requests (ShReq, ExReq, WbRes) of each processor
- A snooper interprets a ShReq as WbReq and ExReq as an InvReq or FlushReq (and ignores WbRes)
- Snooper's response:
  - *ok* means the processor is in the right state (either it does not have the requested data or has it in read only state).
  - *retry* means the processor state is not yet correct for the operation being requested.

Typical Processor-Memory Interface



- Distinct address cycle followed by zero or more data cycles
- In effect more than one request per processor can be on the bus at the same time ⇒ bus tags
- Snooper must respond immediately either with an *ok* or *retry*

Snooper's Input & Output

L1 & Snooper State

<cache, c2m, obt>

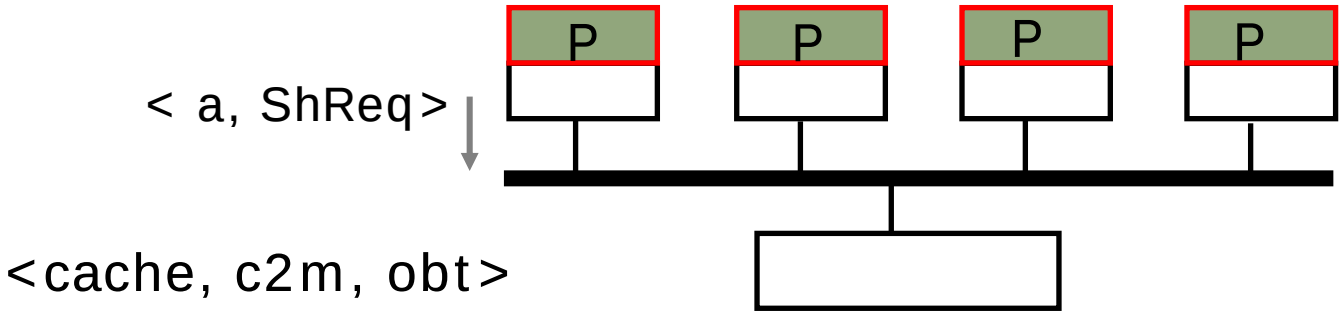
Outstanding bus transactions: a set of <btag, a>

<ShReq, a>
<ExReq, a>
<WbRes, a, v>

Needed to capture the data during a data cycle

- When L1 gets control of the bus, one message from c2m is assigned the tag and put on the bus
- <btag, WbRes, a, v> transactions only affect M
- <btag, ShReq, a> and <btag, ExReq, a> transactions are input to all other Snoopers
  - Each Snooper responds *ok* or *retry*
  - MC summarizes s-resp's into *unanimous-ok* or *retry*

Snooper's Response: *ShReq*



ShReq when input to a snooper acts like a WbReq

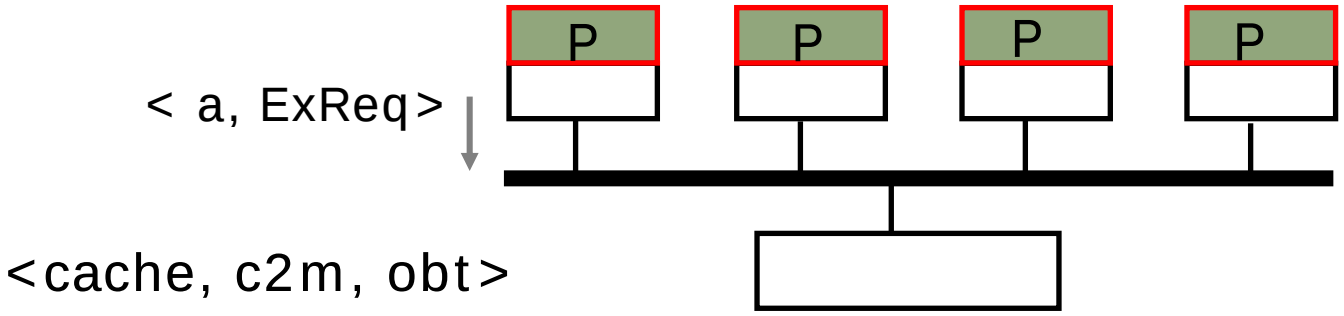
if a ∉ cache & <Wb, a, - > ∉ c2m
→ ok

if cache.state(a)==Sh & <Wb, a, - > ∉ c2m
→ ok

if cache.state(a)==Ex
→ retry; cache.setState(a, Sh); c2m.enq (Wb, a, v)

if <Wb, a, - > ∈ c2m
→ retry

Snooper's Response: *ExReq*



ExReq when input to a snooper acts like either a InvReq or FluShReq

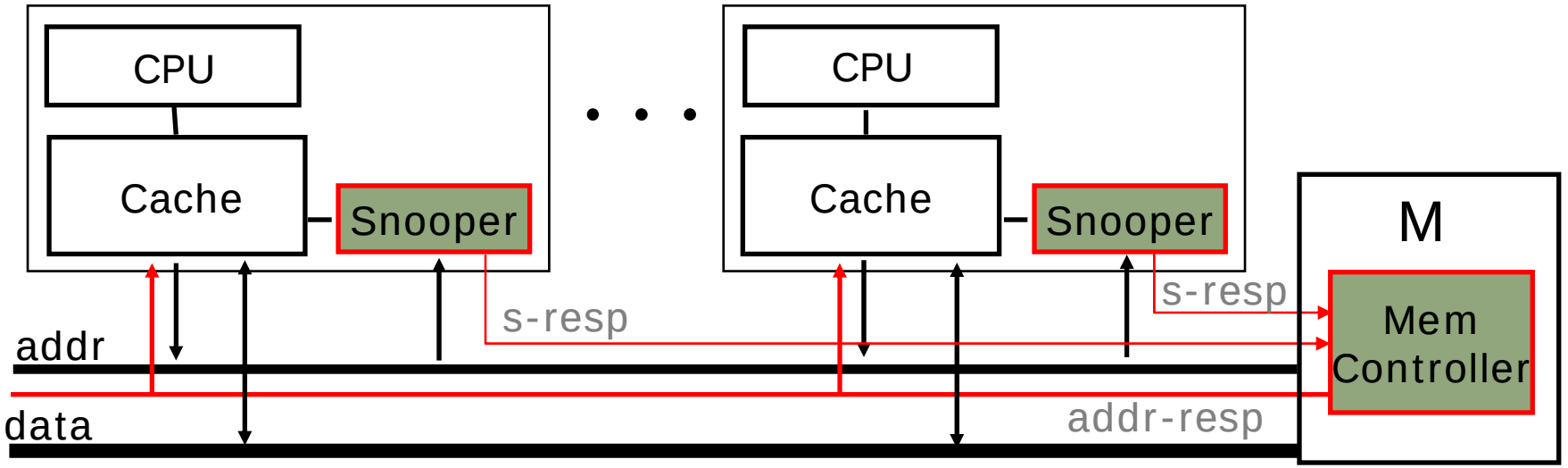
if a $\notin$ cache & <Wb, a, - > $\notin$ c2m
$\rightarrow$ ok

if cache.state(a)==Sh & <Wb, a, - > $\notin$ c2m
$\rightarrow$ ok ; cache.invalidate(a)

if cache.state(a)==Ex
$\rightarrow$ retry; cache.invalidate(a); c2m.enq (Wb, a, v)

if <Wb, a, - > $\in$ c2m
$\rightarrow$ retry

Memory Controller Response



Addr-Request	Addr-Response	Data
<tag,ShReq,a>	*retry*	
	u-ok	<tag,M[a]>
<tag,ExReq,a>	*retry*	
	u-ok	<tag,M[a]>
<tag,Wb,a>	*u-ok*	<tag,Wb,a,data>

data to be written in the memory

Effect of MC's Response on the Bus Master

Address Bus transaction <tag, a>

Unanimous-ok
  <a, type> = = c2m.first
    → c2m.deq
      obt.enq (tag, type, a)
Retry
  <a, type> = = c2m.first
    → c2m.deq
      c2m.enq (type, a)

Set up for the data cycle

randomization for retry

Data Bus transaction <tag, v>

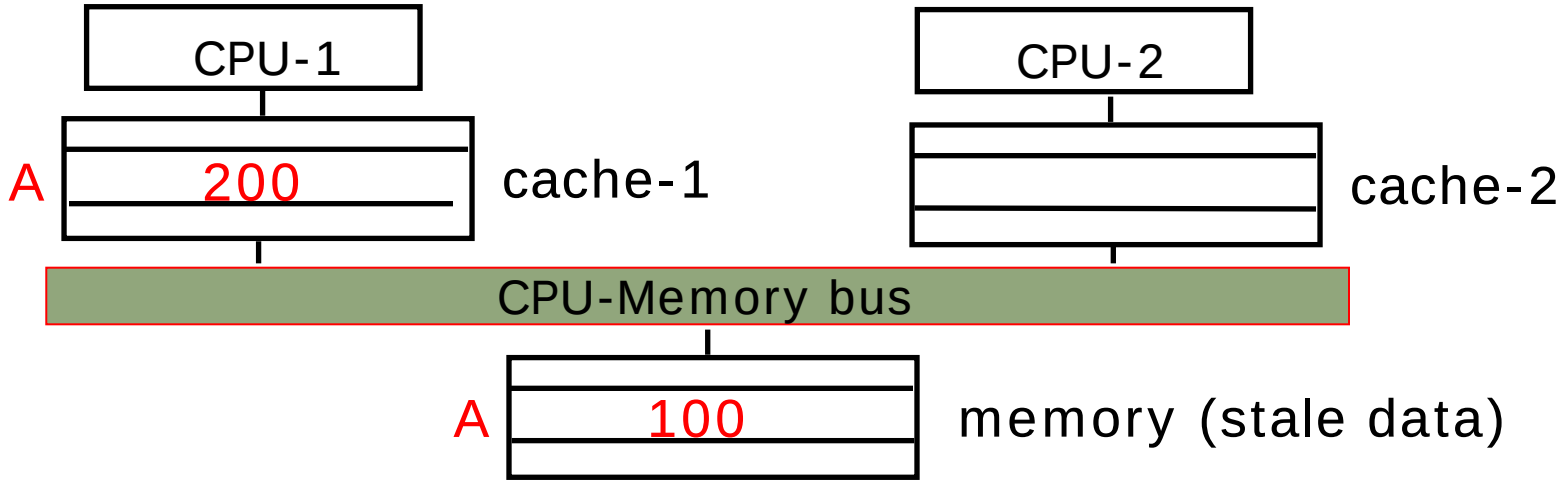
<tag, type, a > = = obt.first
  → cache.setState(a,type);
    cache.setData(a,v);
    obt.deq

type :: Sh | Ex

CSAIL

Bus Occupancy Issues
and
Synchronization Primitives

Intervention: an important optimization



On a cache miss, if the data is present in any other cache it is faster to supply the data to the requester cache from the cache that has it.

This is done in cooperation with the memory controller and by declaring one of the caches to be the “owner” of the address.

False Sharing

A cache block contains more than one word

Cache-coherence is done at the block-level and not word-level

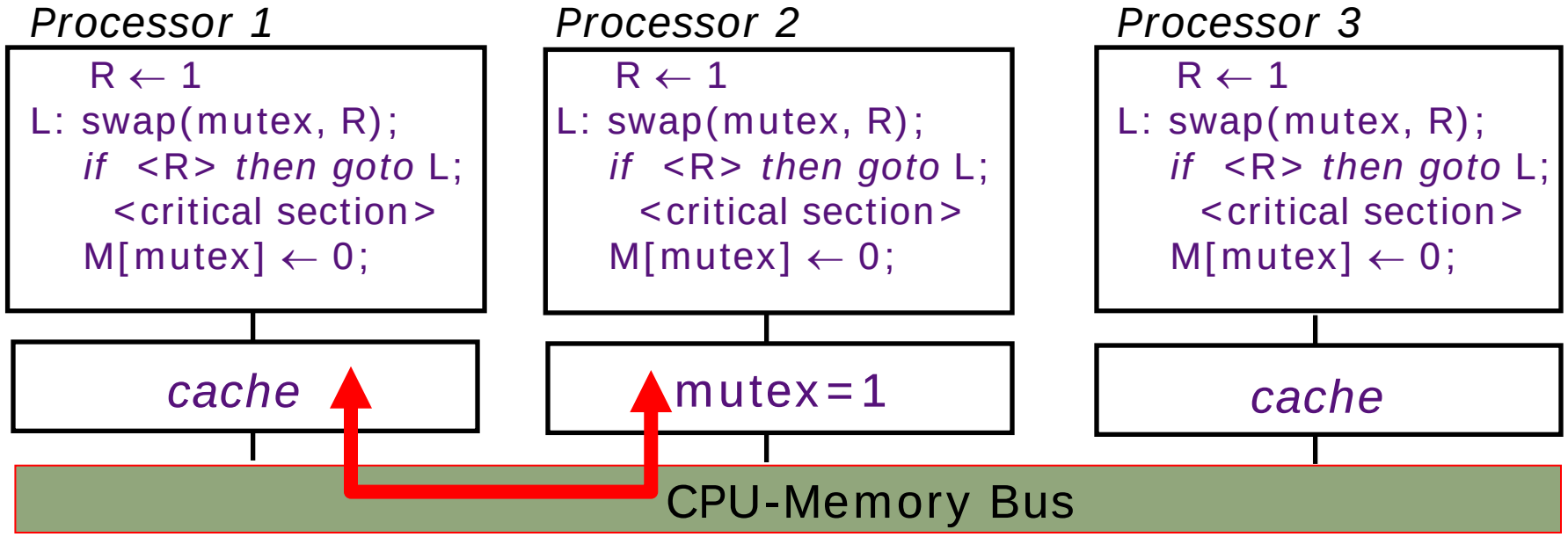
Suppose M_1 writes $word_i$ and M_2 writes $word_k$ and both words have the same block address.

What can happen?

The block will ping-pong between caches unnecessarily

Solutions: 1. Compiler can pack data differently
2. A dirty bit per word as opposed to per block

Synchronization and Caches:
Performance Issues



Cache-coherence protocols will cause mutex to *ping-pong* between P1's and P2's caches.

Ping-ponging can be reduced by first reading the mutex location *(non-atomically)* and executing a swap only if it is found to be zero.

Performance Related to Bus occupancy

In general, a *read-modify-write* instruction requires two memory (bus) operations without intervening memory operations by other processors

In a multiprocessor setting, bus needs to be locked for the entire duration of the atomic read and write operation

$\Rightarrow$ expensive for simple buses

$\Rightarrow$ *very expensive* for split-transaction buses

modern processors use

load-reserve

store-conditional

Load-reserve & Store-conditional

Special register(s) to hold reservation flag and address, and the outcome of store-conditional

```
Load-reserve(R, a):
    <flag, adr> ← <1, a>;
    R ← M[a];
```

```
Store-conditional(a, R):
    if <flag, adr> == <1, a>
    then  cancel other procs'
          reservation on a;
          M[a] ← <R>;
          status ← succeed;
    else  status ← fail;
```

If the snooper sees a store transaction to the address in the reserve register, the reserve bit is set to 0

- Several processors may reserve 'a' simultaneously
- These instructions are like ordinary loads and stores with respect to the bus traffic
- A store (-conditional) is performed only if the reserve bit is set to 1.

Performance:
Load-reserve & Store-conditional

The total number of memory (bus) transactions is not necessarily reduced, but splitting an atomic instruction into load-reserve & store-conditional:

- *increases bus utilization* (and reduces processor stall time), especially in split-transaction buses
- *reduces cache ping-pong effect* because processors trying to acquire a semaphore do not have to perform a store each time

Next Lecture

Beyond Sequential Consistency: Relaxed Memory Models