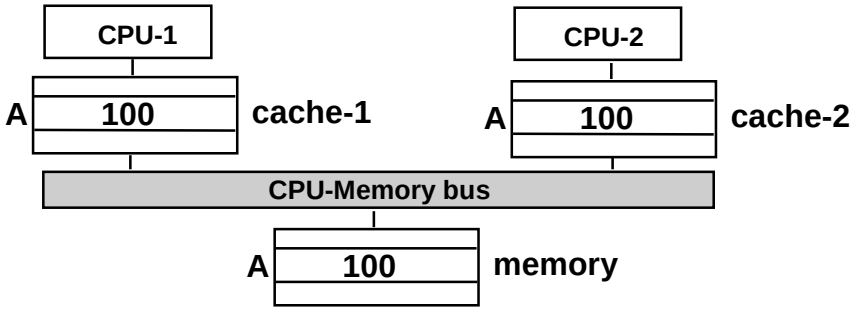
A Cache Coherence Protocol to Implement Sequential Consistency

Krste Asanovic
Laboratory for Computer Science
M.I.T.

http://www.csg.lcs.mit.edu/6.823

Memory Consistency in SMPs



Suppose CPU-1 updates A to 200.

write-back policy: **memory and cache-2 have stale values**

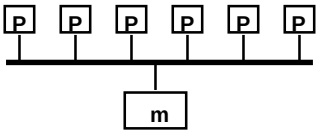
write-through policy: **cache-2 has a stale value**

Do these stale values matter?
What is the view of shared memory for programming?

Sequential Consistency, Relaxed memory models

Sequential Consistency

A Memory Model



" A system is *sequentially consistent* if 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 order specified by the program"

Leslie Lamport

Sequential Consistency =

arbitrary *order-preserving interleaving* of memory references of sequential programs

Caches & Sequential Consistency

Scenario 1

	prog T1	cache-1	memory	cache-2	prog T2

	ST X, 1
	ST Y,11
		X= 1
		Y=11
			X = 0
			Y =10
			X'=
			Y'=
				Y =
				Y'=
				X =
				X'=
					LD Y, R1
					ST Y', R1
					LD X, R2
					ST X',R2

• T1 is executed					
• T2 is executed		X= 1 Y=11	X = 0 Y =10 X'= Y'=	Y = 10 Y'= 10 X = 0 X'= 0	
• cache-1 writes back X & Y		X= 1 Y=11	X = 1 Y =11 X'= Y'=	Y = 10 Y'= 10 X = 0 X'= 0	
• cache-2 writes back X' & Y'		X= 1 Y=11	X = 1 Y =11 X'= 0 Y'=10	Y = 10 Y'= 10 X = 0 X'= 0	

assume a write-back cache

Caches & Sequential Consistency

Scenario 2

prog T1	cache-1	memory	cache-2	prog T2

ST X, 1
ST Y,11
	X= 1
	Y=11
		X = 0
		Y =10
		X'=
		Y'=
			Y =
			Y'=
			X =
			X'=
				LD Y, R1
				ST Y', R1
				LD X, R2
				ST X',R2

- **T1 is executed**

cache-1	memory	cache-2
X= 1 Y=11	X = 0 Y =11 X'= Y'=	Y = Y'= X = X'=

- **cache-1 writes back Y**

cache-1	memory	cache-2
X= 1 Y=11	X = 0 Y =11 X'= Y'=	Y = 11 Y'= 11 X = 0 X'= 0

- **T2 executed**

cache-1	memory	cache-2
X= 1 Y=11	X = 1 Y =11 X'= Y'=	Y = 11 Y'= 11 X = 0 X'= 0

- **cache-1 writes back X**

cache-1	memory	cache-2
X= 1 Y=11	X = 1 Y =11 X'= 0 Y'=11	Y = Y'= X = X'=

- **cache-2 writes back X' & Y'**

assume a write-back cache

Write-through Caches & Sequential Consistency

Scenario:

prog T1	cache-1	memory	cache-2	prog T2

ST X, 1
ST Y,11
	X= 0
	Y=10
		X = 0
		Y =10
		X'=
		Y'=
			Y =
			Y'=
			X = 0
			X'=
				LD Y, R1
				ST Y', R1
				LD X, R2
				ST X',R2

- **T1 executed**

cache-1	memory	cache-2
X= 1 Y=11	X = 1 Y =11 X'= Y'=	Y = Y'= X = 0 X'=

- **T2 executed**

cache-1	memory	cache-2
X= 1 Y=11	X = 1 Y =11 X'= 0 Y'= 11	Y = 11 Y'= 11 X = 0 X'= 0

Write-through caches don't preserve sequential consistency either

Maintaining Sequential Consistency

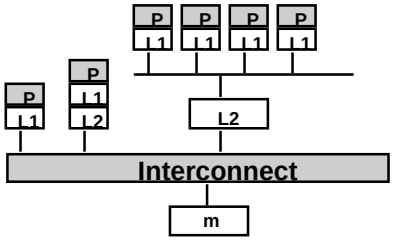
Multiple copies of a location in various caches can cause SC to breakdown.

Hardware support is required such that

- **only one processor at a time has write permission for a location**
- **no processor can load a stale copy of the location after a write**

$\Rightarrow$ ***cache coherence protocols***

A System with Multiple Caches



- **Modern systems often have hierarchical caches**
- **Each cache has exactly one parent but can have zero or more children**
- **Only a parent and its children can communicate directly**
- ***Inclusion property* is maintained between a parent and its children, i.e.,**

$$a \in L_i \quad \Rightarrow \quad a \in L_{i+1}$$

Cache Coherence Protocols for SC

write request:

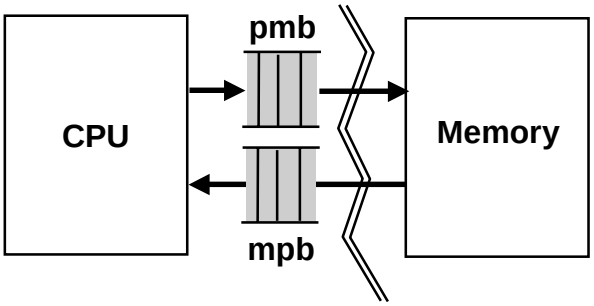
the address is *invalidated* or *updated* in all other caches before the write is performed

read request:

if a dirty copy is found in some cache, a write-back is performed before the memory is read

We will focus on Invalidation protocols

CPU-Memory Interface



Data memory will be modeled as an external interface

pmb: *processor-to-memory FIFO buffers*

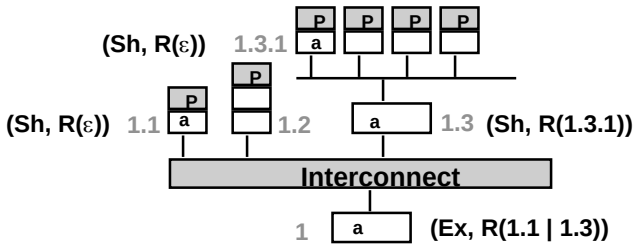
<t, Load(a)> or <t, Store(a,v)>

mpb: *memory-to-processor buffers (unordered)*

<t, v> or <t, ack>

Can use this interface for in-order or out-of-order CPUs

State Encoding



Each address in a cache keeps two types of state info

- ***sibling info:*** **does any of my siblings have a copy**
  - **Ex vs Sh**
- ***children info:*** **has this address been passed on to any of my children**
  - **W(id) vs R(dir)** ***where*** **dir = id_1| ...| id_n**

directory of children

R(ε) means uncached

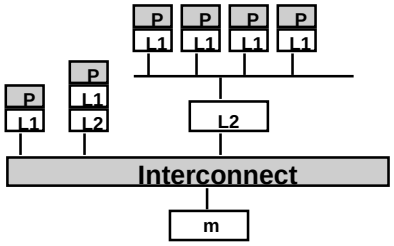
Cache State Implications

Sh ⇒ cache's siblings and descendents can only have Sh copies

Ex ⇒ each ancestor of the cache must be in Ex
⇒ *either* all children can have Sh copies
or **one child can have an Ex copy**

- **Once a parent gives an Ex copy to a child, the parent's data is considered stale**
- **A processor cannot overwrite data in Sh state in L1**

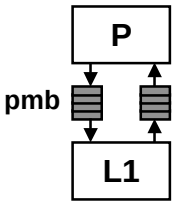
Initial State



- **All M's (caches) except the outermost M (home) are empty (ε)**
- **Initialize each cell of the outermost memory to be Cell(a,-,(Ex,R(ε)))**

address *value* *state*

Load & Store rules



- ***Load rule***

  <id, Cell(a,v,(cs,R(ε))) | m>, <t,Load(a)>;pmb, mpb
  → <id, Cell(a,v,(cs,R(ε))) | m>, pmb, mpb|<t,v>

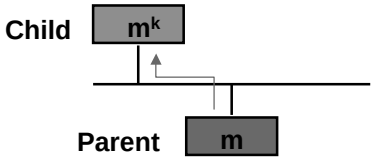
cache id *cache contents (array of cells)*

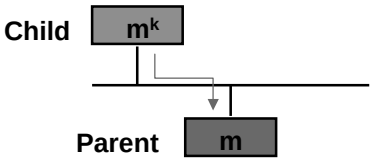
- ***Store rule***

  <id, Cell(a,-,(Ex,R(ε))) | m>, <t,Store(a,v)>;pmb, mpb
  → <id, Cell(a,v,(Ex,R(ε))) | m>, pmb, mpb|<t,Ack>

"|" represents unordered concatentation

Data Propagation Between Caches

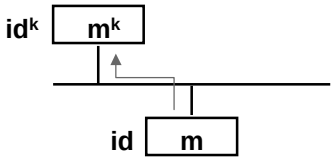




Caching rules
- ***R-caching rule***
- ***W-caching rule***

De-caching rules
- ***Write-back rule***
- ***Invalidate rule***

Caching Rules: *Parent to Child*



- ***R-caching rule***

  $\langle id, Cell(a,v,(cs,R(dir))) \mid m\rangle, \langle id^k, m^k\rangle$ *if* $id^k \notin dir$

  $\rightarrow \langle id, Cell(a,v,(cs,R(id^k \mid dir))) \mid m\rangle,$

  $\langle id^k, Cell(a,v,(Sh,R(\varepsilon))) \mid m^k\rangle$

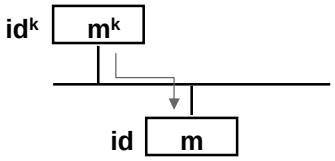
- ***W-caching rule***

  $\langle id, Cell(a,v,(Ex,R(\varepsilon))) \mid m\rangle, \langle id^k, m^k\rangle$

  $\rightarrow \langle id, Cell(a,v,(Ex,W(id^k))) \mid m\rangle,$

  $\langle id^k, Cell(a,v,(Ex,R(\varepsilon))) \mid m^k\rangle$

De-caching Rules: *Child to Parent*



- ***Writeback rule***

  < id, Cell(a,-,(Ex,W(id^k))) | m >,
  < id^k, Cell(a,v,(Ex,R(dir))) | m^k >

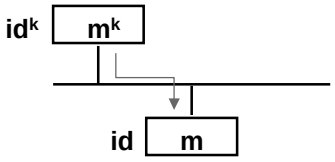
  → < id, Cell(a,v,(Ex,R(id^k))) | m >,
  <id^k, Cell(a,v,(Sh,R(dir))) | m^k >

- ***Invalidate rule***

  < id, Cell(a,v,(cs,R(id^k|dir))) | m >,
  < id^k, Cell(a,v,(Sh,R(ε))) | m^k >

  → < id, Cell(a,v,(cs,R(dir))) | m>,
  < id^k, m^k>

Local Rules



Some rules require observing and changing the state of two caches simultaneously (atomically), e.g.,

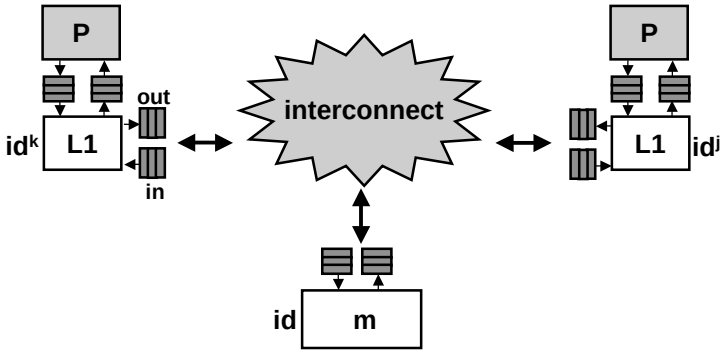
- ***Writeback rule***

  < id, Cell(a,-,(Ex,W(id^k))) | m >,
  < id^k, Cell(a,v,(Ex,R(dir))) | m^k >

  → < id, Cell(a,v,(Ex,R(id^k))) | m >,
  <id^k, Cell(a,v,(Sh,R(dir))) | m^k >

Usually not possible, especially if the caches are separated by a network

DSM and Messages



- ***Provide each M with in and out queue:*** **< id, m, in, out >**
- ***FIFO messages passing between each (src,dest) pair***
- ***Introduce request and reply messages:***

  $Msg(id_{src}, id_{dest}, cmd, H/L, a, v)$
- ***Low priority (L) msg cannot block high priority (H) msg***

Making the Rules *Local*

Each rule is replaced by two rules:

one for the sender and one for the receiver

For example, the Writeback rule splits into

Child's action

$< id^k, Cell(a,v,(Ex,R(dir)))|m^k, in^k, out^k >$

$\rightarrow$ $< id^k, Cell(a,v,(Sh,R(dir)))|m^k, in^k, out^k; msg(id^k,id,WbRep,H,a,v) >$

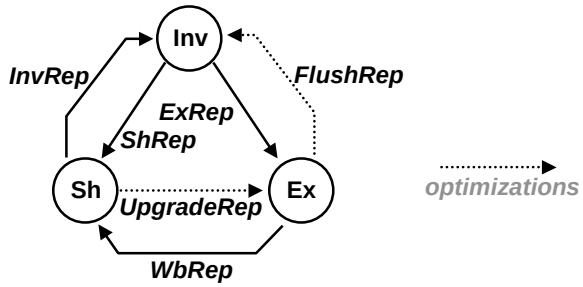
***where* id = parent(id^k)**

Parent's action

$< id, Cell(a,-,(Ex,W(id^k)))|m, msg(id^k,id,WbRep,H,a,v); in, out >$

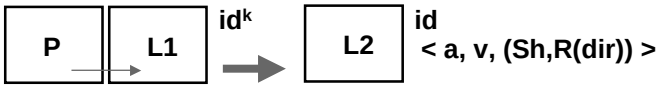
$\rightarrow$ $< id, Cell(a,v,(Ex,R(id^k)))|m, in, out >$

Cache State Transitions



What causes a state transition ?
or
When should a rule be applied ?

When to Apply a Rule



Consider the following rules:

- ***Load rule***

  <id, Cell(a,v, (cs,R(ε))) | m, in, out>, <t,Load(a)>;pmb, mpb
  → <id, Cell(a,v, (cs,R(ε))) | m, in, out>, pmb, mpb|<t,v>

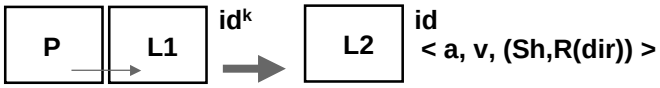
- ***R-caching rule for Sender (Parent)***

  <id, Cell(a,v,(cs,R(dir))) | m, in, out > ***if*** **$id^k \notin$ dir**
  → <id, Cell(a,v,(cs,R(id^k|dir))) | m, in, out;msg(id,id^k,ShRep,H,a,v) >

Suppose a is not in L1 but is present in L2 in the Sh state.

What should be done when P executes Load(a) ?

Issuing Requests



If a is not in L1, send a request (ShReq) to L2, and set the cache state to be transient (CachePending)

<id, m, in, out>, <t,Load(a)>;pmb, mpb *if* a∉ c
→ <id, Cell(a,-,CachePending)|m, in, out;msg(id,parent(id),ShReq,L,a)>,
<t,Load(a)>;pmb, mpb

The load instruction remains suspended

Protocol X2:

A Protocol for a system with two memory levels (L1 + M)

Xiaowei Shen

Simplified states:
Cache state: Sh and Ex
Memory state: R[dir] and W[id]

Load Rules

- ***Load-hit rule***

<id, Cell(a,v,Sh)|c, in, out>, <t,Load(a)>;pmb, mpb
→ <id, Cell(a,v,Sh)|c, in, out>, pmb, mpb|<t,v>

<id, Cell(a,v,Ex)|c, in, out>, <t,Load(a)>;pmb, mpb
→ <id, Cell(a,v,Ex)|c, in, out>, pmb, mpb|<t,v>

- ***Load-miss rule***

<id, c, in, out>, <t,Load(a)>;pmb, mpb *if* a$\notin$ c
→ <id, Cell(a,-,CachePending)|c, in, out;msg(id,Home,ShReq,L,a)>,
<t,Load(a)>;pmb, mpb

Store Rules

- ***Store-hit rule***

<id, Cell(a,-,Ex)|c, in, out>, <t,Store(a,v)>;pmb, mpb
→ <id, Cell(a,v,Ex)|c, in, out>, pmb, mpb|<t,Ack>

- ***Store-miss rule***

<id, c, in, out>, <t,Store(a,v)>;pmb, mpb *if* a $\notin$ c
→ <id, Cell(a,-,CachePending)|c, in, out;msg(id,Home,ExReq,L,a)>,
<t,Store(a,v)>;pmb, mpb

<id, Cell(a,-,Sh)|c, in, out>, <t,Store(a,v)>;pmb, mpb
→ <id, Cell(a,-,CachePending)|c, in, out;msg(id,Home,ExReq,L,a)>,
<t,Store(a,v)>;pmb, mpb

Processing ShReq Messages *(at Home)*

Uncached or Outstanding Shared Copies

<Cell(a,v,R(dir))|m, msg(id,Home,ShReq,L,a);in, out> ***if*** **id $\notin$ dir**

$\rightarrow$ <Cell(a,v,R(id|dir))|m, in, out;msg(Home,id,ShRep,H,a,v)>

Outstanding Exclusive Copy

<Cell(a,v,W(id'))|m, msg(id,Home,ShReq,L,a);in, out> ***if*** **id $\neq$ id'**

$\rightarrow$ <Cell(a,v,T_W(id'))|m, msg(id,Home,ShReq,L,a);in,
out;msg(Home,id',WbReq,H,a)>

Processing ExReq Messages *(at Home)*

Uncached

<Cell(a,v,R(ε))|m, msg(id,Home,ExReq,L,a);in, out>

$\rightarrow$ <Cell(a,v,W(id))|m, in, out;msg(Home,id,ExRep,H,a,v)>

Outstanding Shared Copies

<Cell(a,v,R(dir))|m, msg(id,Home,ExReq,L,a);in, out> ***if*** **dir $\neq \varepsilon$**

$\rightarrow$ <Cell(a,v,T_R(dir))|m, msg(id,Home,ExReq,L,a);in,
out;multicast(Home,dir-{id},InvReq,H,a)>

Outstanding Exclusive Copy

<Cell(a,v,W(id'))|m, msg(id,Home,ExReq,L,a);in, out> ***if*** **id $\neq$ id'**

$\rightarrow$ <Cell(a,v,T_W(id'))|m, msg(id,Home,ExReq,L,a);in,
out;msg(Home,id,FlushReq,H,a)>

Processing Reply Messages *(at cache)*

ShRep

<id, Cell(a,-,CachePending)|c, msg(Home,id,ShRep,H,a,v);in, out>
→ <id, Cell(a,v,Sh)|c, in, out>

ExRep

<id, Cell(a,-,CachePending)|c, msg(Home,id,ExRep,H,a,v);in, out>
→ <id, Cell(a,v,Ex)|c, in, out>

Processing Request Messages *(at cache)*

WbReq

<id, Cell(a,v,Ex)|c, msg(Home,id,WbReq,H,a);in, out>
→ <id, Cell(a,v,Sh)|c, in, out;msg(id,Home,WbRep,H,a,v)>

FlushReq

<id, Cell(a,v,Ex)|c, msg(Home,id,FlushReq,H,a);in, out>
→ <id, c, in, out;msg(id,Home,FlushRep,H,a,v)>

InvReq

<id, Cell(a,v,Sh)|c, msg(Home,id,InvReq,H,a);in, out>
→ <id, c, in, out;msg(id,Home,InvRep,H,a)>

<id, Cell(a,v,CachePending)|c, msg(Home,id,InvReq,H,a);in, out>
→ <id, Cell(a,v,CachePending)|c, in, out;msg(id,Home,InvRep,H,a)>

Processing Reply Messages *(at home)*

WbRep

<Cell(a,-,T_W(id))|m, msg(id,Home,WbRep,H,a,v);in, out>
$\rightarrow$ <Cell(a,v,R(id))|m, in, out>

FlushRep

<Cell(a,-,T_W(id))|m, msg(id,Home,FlushRep,H,a,v);in, out>
$\rightarrow$ <Cell(a,v,R(ε))|m, in, out>

InvRep

<Cell(a,v,T_R(id|dir))|m, msg(id,Home,InvRep,H,a);in, out> ***if*** **dir $\neq \varepsilon$**
$\rightarrow$ <Cell(a,v,T_R(dir))|m, in, out>

<Cell(a,v,T_R(id))|m, msg(id,Home,InvRep,H,a);in, out>
$\rightarrow$ <Cell(a,v,R(ε))|m, in, out>