# 6.823 Computer System Architecture (2011F)
## 6823 Cache Coherence Protocol (directory-based)

## MSI: {M, S, I}

- M: Modifiable, Exclusive copy, cannot be present in any other cache
- S: Shared copy, can also be present in other caches
- I: Not present in this cache, can be present in other caches

M > S > I

## Cache states:

c.state(a) : sibling info – M|S|I
c.child[$c_k$](a) : child $c_k$ info – M|S|I
c.waitp(a) : Denotes if cache c is waiting for a response from its parent. If so, what type of response
- Nothing *means* not waiting
- Valid (M|S|I) *means* waiting for response to go to M or S or I as the case may be

c.waitc[$c_k$](a) : Denotes if cache c is waiting for a response from its child $c_k$. If so, what type of response
- Nothing | Valid (M|S|I)

## IsCompatible:

The states x, y of two sibling caches are compatible iff IsCompatible(x, y) is True where
IsCompatible(M, M) = False
IsCompatible(M, S) = False
IsCompatible(S, M) = False
All other cases = True

## Messages:

**Parent to Child:**

<c, m, M2C_Req, a, y>: Parent m requesting a child c to downgrade the state of location *a* to y

<c, m, M2C_Rep, a, x, y, data>: Parent m sending a notification to child c to upgrade the state of location *a* from x to y

**Child to Parent:**

<m, c, C2M_Req, a, y>: Child c requesting the parent m to upgrade the state of location *a* to y

<m, c, C2M_Rep, a, x, y, data>: Child c sending a notification to parent m saying it has downgraded the state of location *a* from x to y

# DataTransfer:

*Child to Parent data transfer*: Downgrade of a child from M to S and M to I requires that the dirty data for that location also gets transferred to the parent

DataTransfer(M, S) = True
DataTransfer(M, I) = True
DataTransfer(S, I) = False

*Parent to child data transfer*: Upgrade initiated by the parent usually requires a data transfer except in the case of S to M because the child already has the data

DataTransfer(I, S) = True
DataTransfer(I, M) = True
DataTransfer(S, M) = False

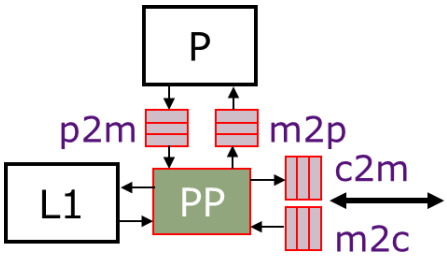
# Processor rules:

## Load-hit rule

inst is (Load a) & c.state(a) is S or M
→ p2m.deq;
m2p.enq(c.data(a));

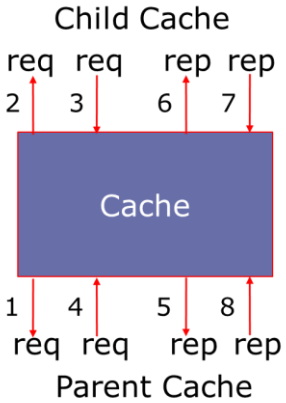
## Store-hit rule

inst is (Store a v) & c.state(a) is M
→ p2m.deq;
m2p.enq(Ack);
c.data(a):=v;




# Types of Actions:

A protocol specifies cache actions corresponding to each of these 8 requests and responses



## Sending Requests

### 1. Child sending Upgrade-to-y req

(c.state(a)<y) & (c.waitp(a)==Nothing)
→ c.waitp(a):=Valid y;
c2m.enq(<m, c, C2M_Req, a, y>);

### 2. Parent sending downgrade to y req

(m.child[i](a)>y) & (m.waitc[i](a)==Nothing)
→ m.waitc[i](a):=Valid y;
m2c.enq(<i, m, M2C_Req, a, y>);

## Dequeuing Requests

### 3. Child dequeuing Downgrade-to-y req

(m2c.msg=<c, m, M2C_Req, a, y>) & (c.state(a)≤y)
→ m2c.deq;

### 4. Parent dequeuing Upgrade-to-y req

(c2m.msg=<m, c, C2M_Req, a, y>) & (m.child[c](a)≥y)
→ c2m.deq;

## Sending Responses

### 5. Child sending Downgrade-from-x-to-y rep

(c.state(a)=x) & (y<x) & (∀i, c.child[i](a)≤y)
→ c.state(a):=y;
c2m.enq(<m, c, C2M_Rep, a, x, y,
(if DataTransfer(x,y) then c.data(a)
else _)>);

### 6. Parent sending Upgrade-from-x-to-y rep

(m.child[c](a)=x) & (y>x) & (m.state(a)≥y)
& (∀i≠c, IsCompatible(m.child[i](a),y))
→ m.child[c](a):=y;
m2c.enq(<c, m, M2C_Rep, a, x, y,
(if DataTransfer(x,y) then m.data(a)
else _)>);

## Receiving Responses

### 7. Child receiving Upgrade-from-x-to-y rep

m2c.msg=<c, m, M2C_Rep, a, x, y, data>
→ m2c.deq;
if c.state(a)==x then {
c.state(a):=y;
if DataTransfer(x,y) then c.data(a):=data;
if c.waitp(a) ≤ (Valid y)
then c.waitp(a):=Nothing;}

### 8. Parent receiving Downgrade-from-x-to-y rep

c2m.msg=<m, c, C2M_Rep, a, x, y, data>
→ c2m.deq;
m.child[c](a):=y;
if DataTransfer(x,y) then m.data(a):=data;
if m.waitc[c](a) ≥ (Valid y)
then m.waitc[c](a):=Nothing;