Constructive Computer Architecture

Cache Coherence

Arvind
Computer Science & Artificial Intelligence Lab.
Massachusetts Institute of Technology

Further issues

- Are these rules enough, i.e., complete?
- Effect of blocking vs non-blocking caches
- Communication systems and buffer requirements to avoid deadlocks

Are the rules exhaustive?

Parent rules

2. Parent to Child: Upgrade-to-y response

```
    (∀j, m.waitc[j][a]=No) & c2m.msg=<Req,c→m,a,y,-> & (∀i≠c,
    IsCompatible(m.child[i][a],y))
    → m2c.enq(<Resp, m→c, a, y,
                   (if (m.child[c][a]=I) then m.data[a] else -)>);
      m.child[c][a]:=y; c2m.deq;
```

What if the guard fails because
1.some child is not in compatible state?
or 2. some child is in wait state?

if condition 1 holds then rule 4 can be invoked

if condition 2 holds then rule 4 must have been invoked and the each child will eventually send a response

Is every rule necessary?

Consider rule 7 for cache

7. Child receiving downgrade-to-y request

```
    (m2c.msg=<Req, m→c, a, y, - >) & (c.state[a]≤y)
    → m2c.deq;
```

A downgrade request comes but the cache is already in the downgraded state

Can happen because of voluntary downgrade

8. Child to Parent: Downgrade-to-y response (vol)

```
    (c.waitp[a]=No) & (c.state[a]>y)
    → c2m.enq(<Resp, c->m, a, y,
                   (if (c.state[a]=M) then c.data[a] else - )>);
      c.state[a]:=y;
```

More rules?

- How about a voluntary upgrade rule from parent?

```
Parent to Child: Upgrade-to-S response (vol)
    (m.waitc[c][a]=No) & (m.cstate[c][a]=S)
    → m2c.enq(<Resp, m->c, a, M, -);
      m.cstate[c][a]:=M;
```

The child could have simultaneously evicted the line, in which case the parent eventually makes m.cstate[c][a] = I while the child makes its c.state[a] = M. This breaks our invariant

A cc protocol is like a Swiss watch, even the smallest change can easily (and usually does) introduce bugs

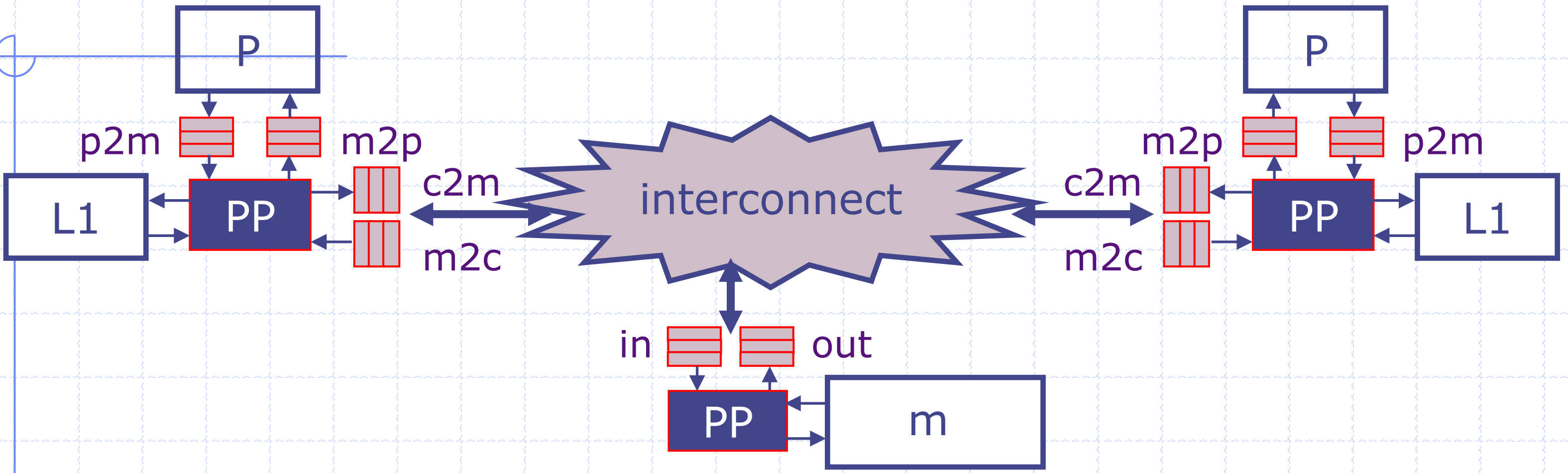
More rules?

- How about a “silent drop”

  8a. Child to Parent: Downgrade-S-to-I response (vol)
  (c.waitp[a]=No) & (c.state[a]=S)
  → ~~c2m.enq(<Resp, c->m, a, y,~~
  ~~(if (c.state[a]=M) then c.data[a] else -)>);~~
  c.state[a]:=I;

A Directory-based Protocol

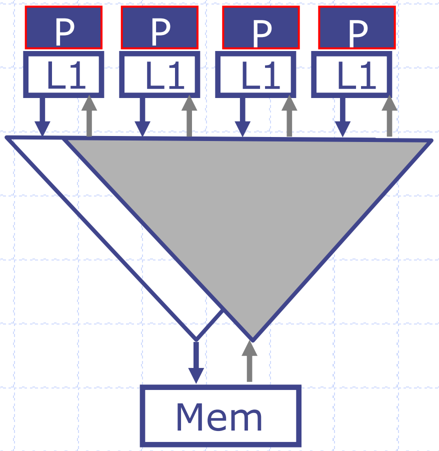
an abstract view



- Each cache has 2 pairs of queues
  - (c2m, m2c) to communicate with the memory
  - (p2m, m2p) to communicate with the processor
- Message format: <cmd, src→dst, a, s, data>

  Req/Resp address state
- FIFO message passing between each (src→dst) pair except a *Req cannot block a Resp*
- *Req* messages from p to m cannot block *Req* messages from m to p

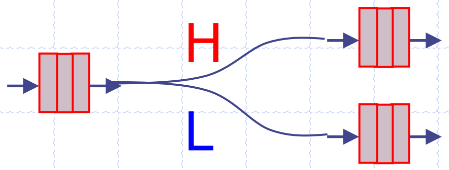
Communication Network



- Two virtual networks:
  - For requests and responses from cache to memory
  - For requests and responses from memory to caches
- Each network has H and L priority messages - a L message can never block an H message other than that messages are delivered in FIFO order

H and L Priority Messages

- At the memory, unprocessed request messages cannot block reply messages.
- H and L messages can share the same wires but must have separate queues



An L message can be processed only if H queue is empty

FIFO property of queues

- If FIFO property is not enforced, then the protocol can either deadlock or update with wrong data
- A deadlock scenario:

1. msg1: Child 1 requests (I -> M) upgrade
2. msg2: Parent responds to Child 1 with upgrade (I -> M)
3. msg3: Child 2 requests (I -> M) upgrade
4. msg4: Parent requests Child 1 (M -> I) downgrade
5. msg4 overtakes msg2
6. Child 1 sees msg4 and drops it
7. Parent never gets a response from Child 1 for msg4

Deadlocks due to buffer space

- A cache or memory always accepts a response, thus responses will always drain from the network
- From the children to the parent, two buffers are needed to implement the H-L priority. A child's req can be blocked and generate more requests
- From parent to all the children, just one buffer is needed for both requests and responses because a parent's req only generates responses