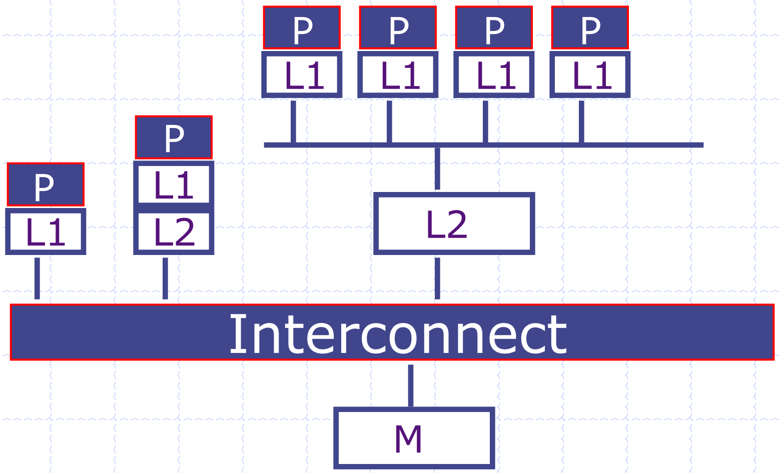
Constructive Computer Architecture

Cache Coherence

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

Shared Memory Systems

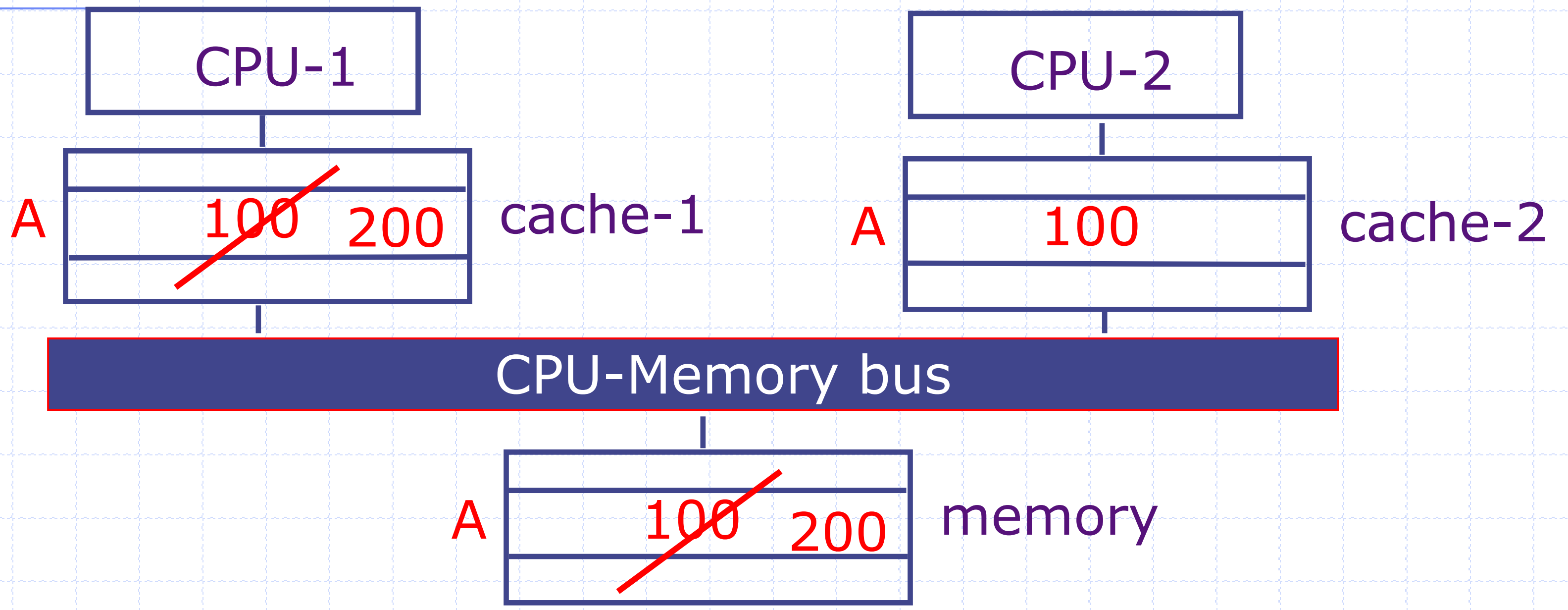


- Modern systems often have hierarchical caches
- Each cache has exactly one parent but can have zero or more children
- Logically only a parent and its children can communicate directly
- *Inclusion property* is maintained between a parent and its children, i.e.,

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

Because usually $L_{i+1} >> L_i$

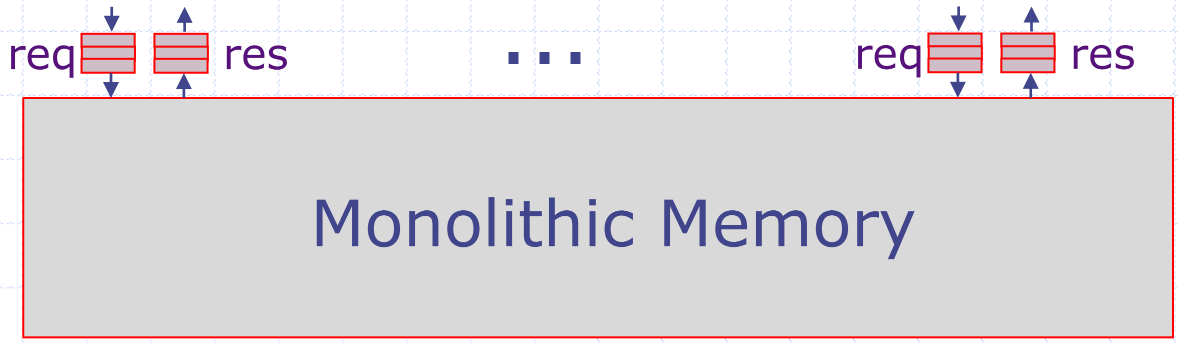
Cache-coherence problem



- Suppose CPU-1 updates A to 200.
  - *write-back:* memory and cache-2 have stale values
  - *write-through:* cache-2 has a stale value

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

Cache-Coherent Memory



- A monolithic or instantaneous memory processes one request at a time and responds to requests immediately
- A memory with hierarchy of caches is said to be *coherent or atomic*, if functionally it behaves like the monolithic memory

Maintaining Store Atomicity

- *Store atomicity* requires all processors to see writes occur in the same order
  - multiple copies of an address in various caches can cause this to be violated
- This property can be ensured if:
  - Only one cache at a time has the write permission for an address
  - No cache can have a stale copy of the data after a write to the address has been performed

$\Rightarrow$ *cache coherence protocols are used to implement store atomicity*

Cache Coherence Protocols

- Write request:
  - the address is *invalidated* in all other caches *before* the write is performed
- Read request:
  - if a dirty copy is found in some cache, that value must be used by doing a write-back and then reading the memory or forwarding that dirty value directly to the reader

Such protocols are called Invalidation-based

State needed to maintain Cache Coherence

- Use MSI encoding in caches where

  I *means* this cache does not contain the address

  S *means* this cache has the address but so may other caches; hence it can only be read

  M *means* only this cache has the address; hence it can be read and written

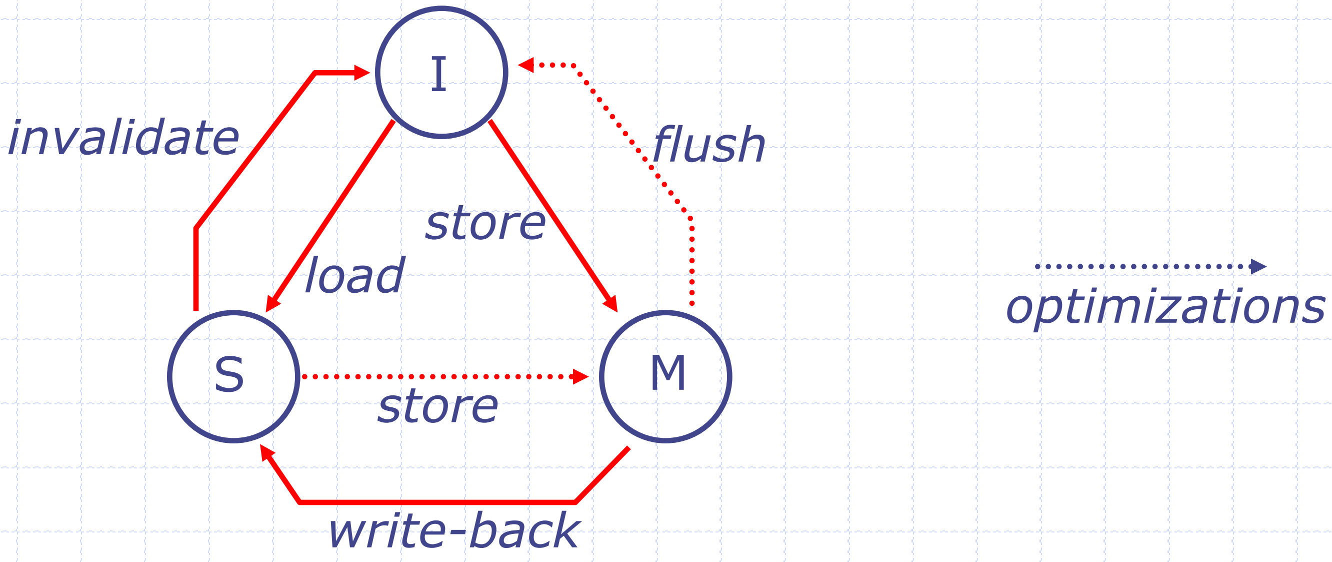
- The states M, S, I can be thought of as an order M > S > I
  - A transition from a lower state to a higher state is called an *Upgrade*
  - A transition from a higher state to a lower state is called a *Downgrade*

Sibling invariant and compatibility

- Sibling invariant:
  - Cache is in state M $\Rightarrow$ its siblings are in state I
  - That is, the sibling states are "compatible"

IsCompatible(M, M)	= False
IsCompatible(M, S)	= False
IsCompatible(S, M)	= False
All other cases	= True

Cache State Transitions



This state diagram is helpful as long as one remembers that each transition involves cooperation of other caches and the main memory to maintain the sibling invariants

Cache Actions

- On a read miss (i.e., Cache state is I):
  - In case some other cache has the address in state M then write back the dirty data to Memory
  - Read the value from Memory and set the state to S
- On a write miss (i.e., Cache state is I or S):
  - *Invalidate* the address in all other caches and in case some cache has the address in state M then write back the dirty data
  - Read the value from Memory if necessary and set the state to M

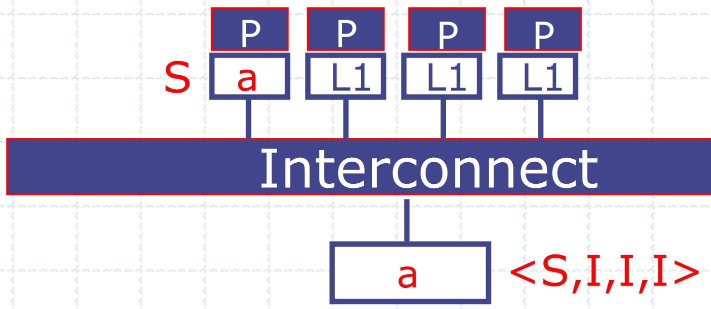
Misses cause Cache upgrade actions which in turn may cause further downgrades or upgrades on other caches

MSI protocol: some issues

- It never makes sense to have two outstanding requests for the same address from the same processor/cache
- It is possible to have multiple requests for the same address from different processors. Hence there is a need to arbitrate requests
- A cache needs to be able to evict an address in order to make room for a different address
  - Voluntary downgrade
- Memory system (higher-level cache) should be able to force a lower-level cache to downgrade
  - caches need to keep track of the state of their children’s caches

Directory State Encoding

Two-level (L1, M) system



- For each address in a cache, the directory keeps two types of info
  - c.state[a] (*sibling info):* do c's siblings have a copy of address *a*; M (means no), S (means maybe)
  - m.child[c_k][a] (*children info):* the state of child c_k for address *a*; At most one child can be in state M

Directory state encoding

wait states

- New states to deal with waiting for responses:
  - c.waitp[a] : Denotes if cache c is waiting for a response from its parent
    - No *means* not waiting
    - Yes (S|I) *means* waiting for a response to transition to state S or I, respectively
  - m.waitc[c_k][a] : Denotes if memory m is waiting for a response from its child c_k
    - No | Yes (M|S)
- Cache state in L1:

  <(M|S|I), (No | Yes(M|S))>
- Directory state in home memory (for each child):

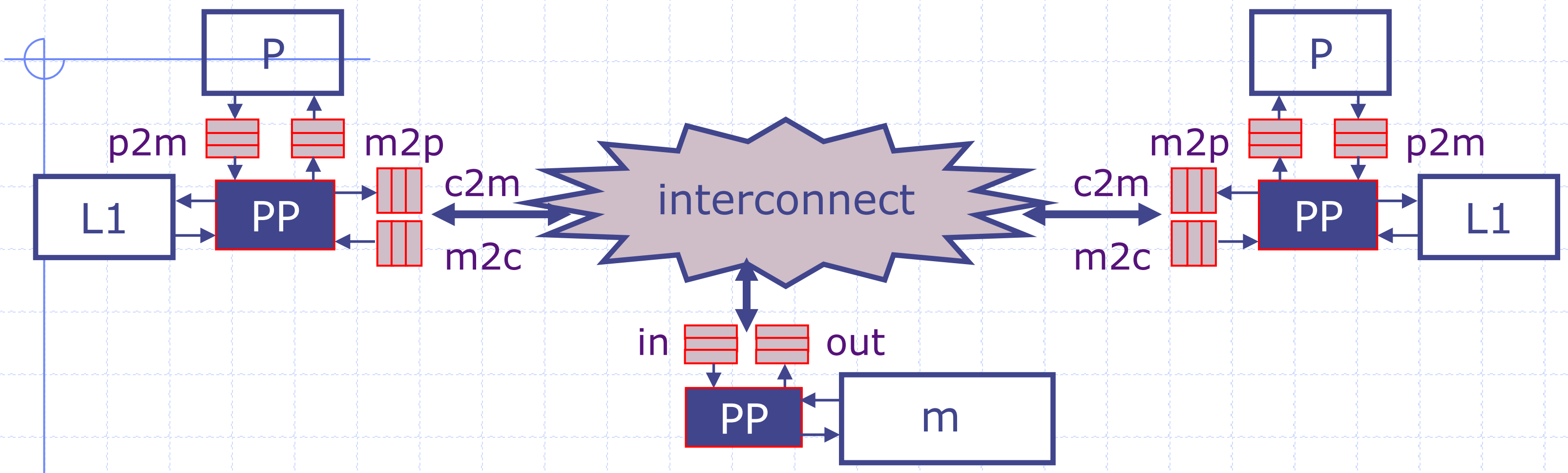
  <[(M|S|I), (No | Yes(S|I))]>

Child's state

Waiting for downgrade response

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*
- Messages in one src→dst path cannot block messages in another src→dst path

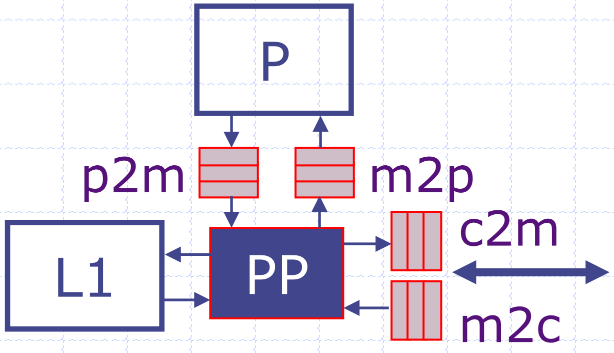
Processor Hit Rules

- Load-hit rule

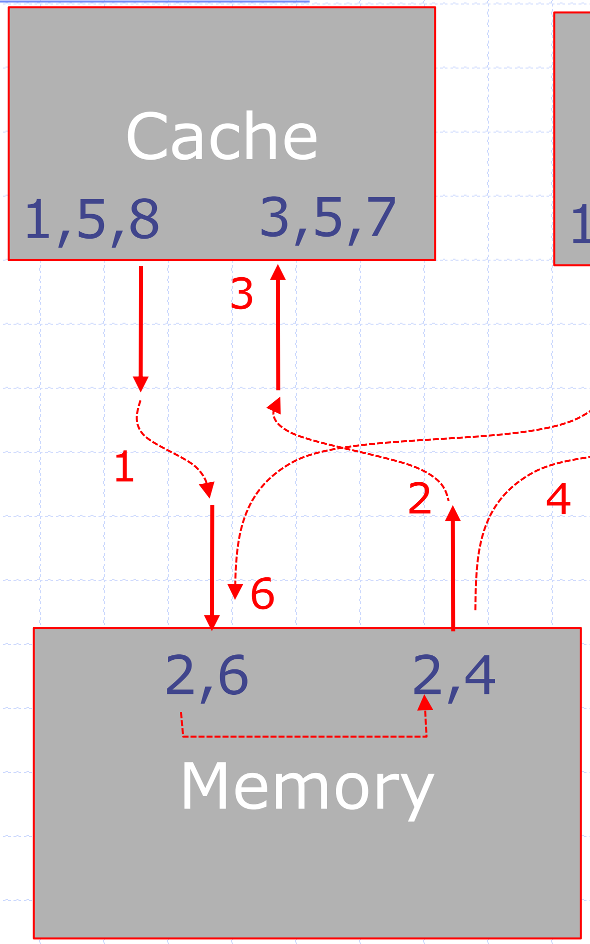
```
    p2m.msg=(Load a) &
    (c.state[a]>I)
→   p2m.deq;
    m2p.enq(c.data[a]);
```

- Store-hit rule

```
    p2m.msg=(Store a v) &
    c.state[a]=M
→   p2m.deq;
    m2p.enq(Ack);
    c.data[a]:=v;
```



Processing misses: Requests and Responses



1 Up-req send (cache)
2 Up-req proc, Up resp send (memory)
3 Up-resp proc (cache)
4 Dn-req send (memory)
5 Dn-req proc, Dn resp send (cache)
6 Dn-resp proc (memory)
7 Dn-req proc, drop (cache)
8 Voluntary Dn-resp (cache)

Invariants for a CC-protocol design

- Directory state is always a conservative estimate of a child's state
  - E.g., if directory thinks that a child cache is in S state then the cache has to be in either I or S state
- For every request there is a corresponding response, though sometimes a response may have been generated even before the request was processed
- Communication system has to ensure that
  - responses cannot be blocked by requests
  - a request cannot overtake a response for the same address
- At every merger point for requests, we will assume fair arbitration to avoid starvation