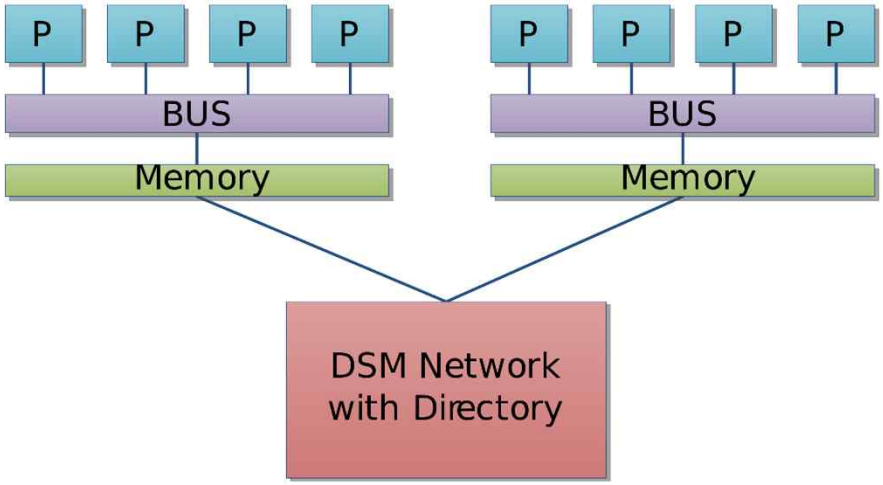
Multicore Computing
Lecture15 - Interconnect

남 범 석

bnam@skku.edu

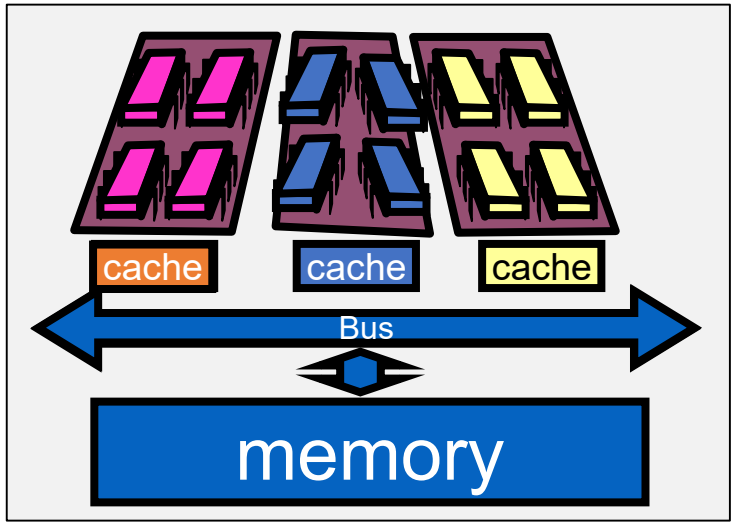
NUMA Architectures

- The address space is global to all processors, but memory is physically distributed

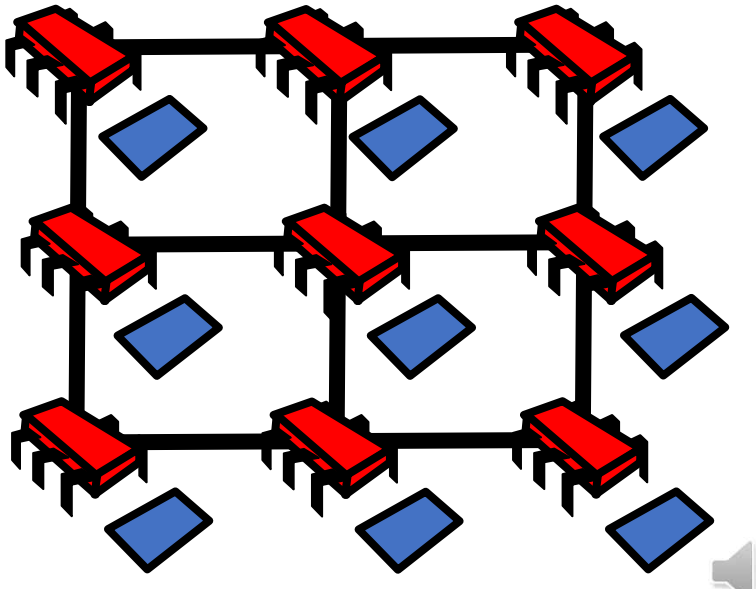


Interconnect

- Bus
  - Like a tiny Ethernet
  - Broadcast medium
  - Connects
    - Processors to memory
    - Processors to processors

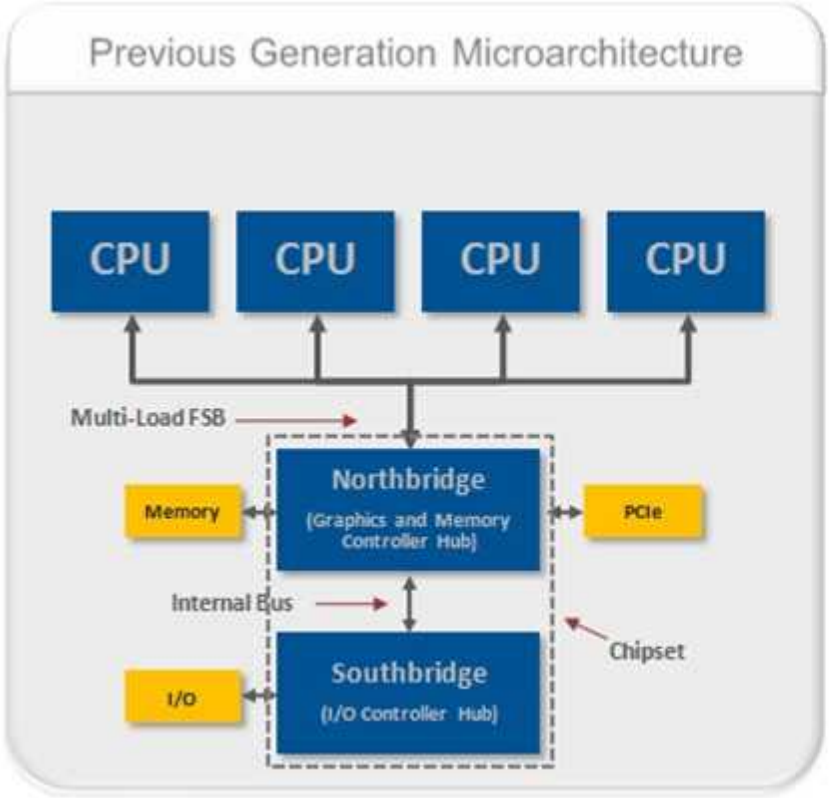


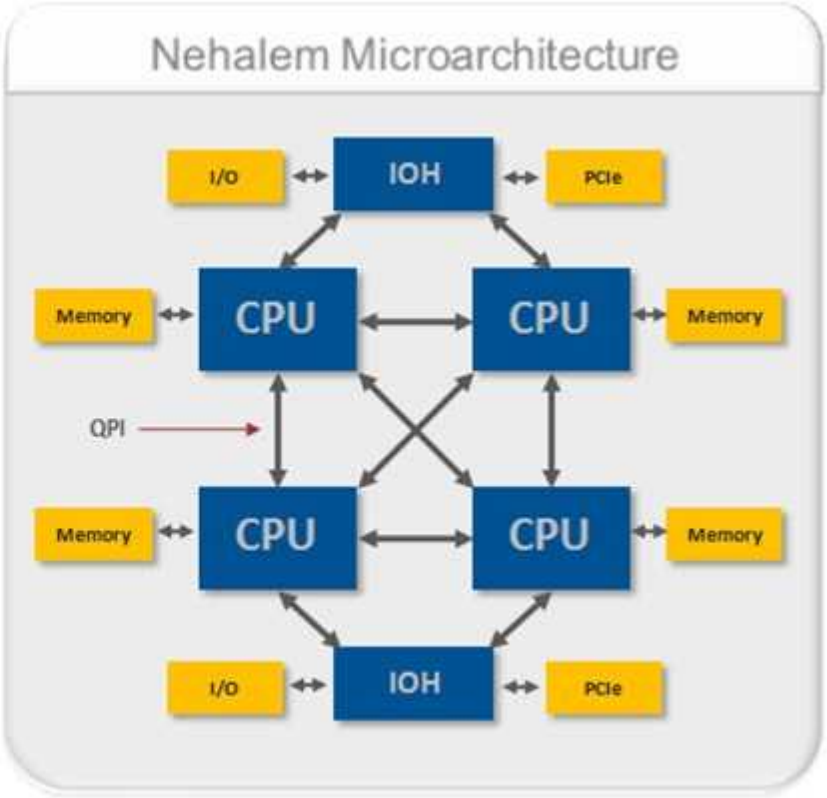
- Network
  - Tiny LAN
  - Mostly used on
  - large machines



FSB vs QPI

- Front Side Bus
- QuickPath Interconnect

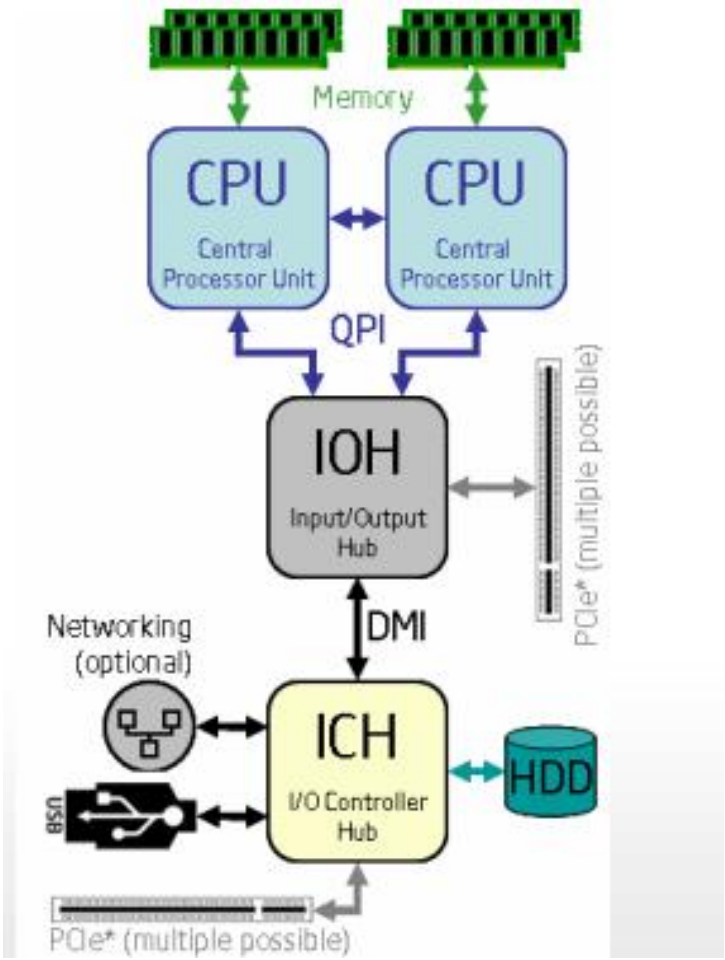


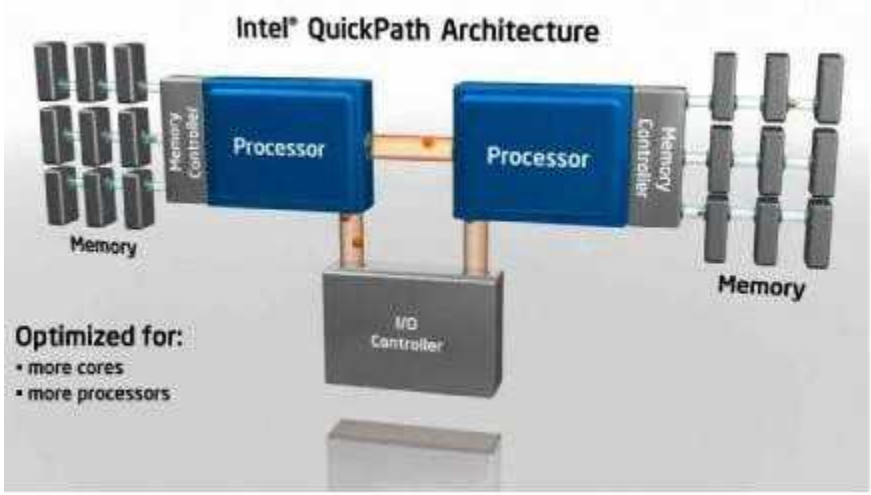


UMA = Uniform Memory Access
SMP = Symmetric MultiProcessor

NUMA = Non-Uniform Memory Access

Hypertransport & QuickPath Interconnect





- AMD HyperTransport (2003)
  Intel QuickPath Interconnect (2008)
- Memory controller now on the CPU
- Memory access is now **non-uniform**

Memory model

- Cache coherence tells us that memory will *eventually* be consistent
- The memory consistency policy tells us *when* this will happen
- Even if memory is consistent, changes don't propagate instantaneously
- These give rise to correctness issues involving program behavior

Memory Consistency Model

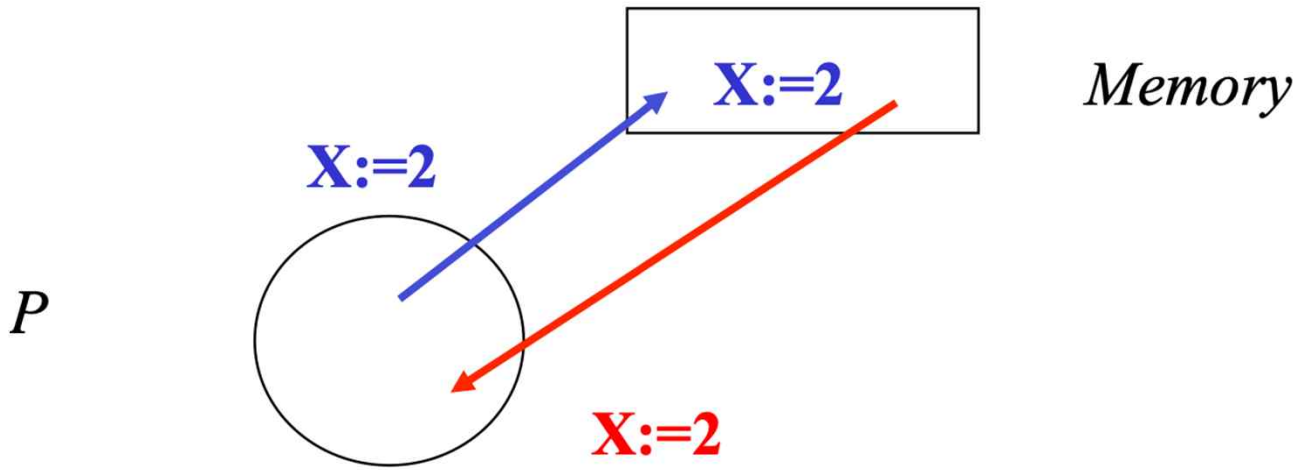
- The memory consistency model determines when a written value will be seen by a reader
- **Sequential Consistency** maintains a linear execution on a parallel architecture that is consistent with the sequential execution of some interleaved arrangement of the separate concurrent instruction streams
  - Expensive to implement
- **Relaxed consistency**
  - Enforce consistency only at well defined times
  - Useful in handling false sharing

Memory consistency

- A memory system is consistent if the following 3 conditions hold
  - Program order
  - Definition of a coherent view of memory
  - Serialization of writes

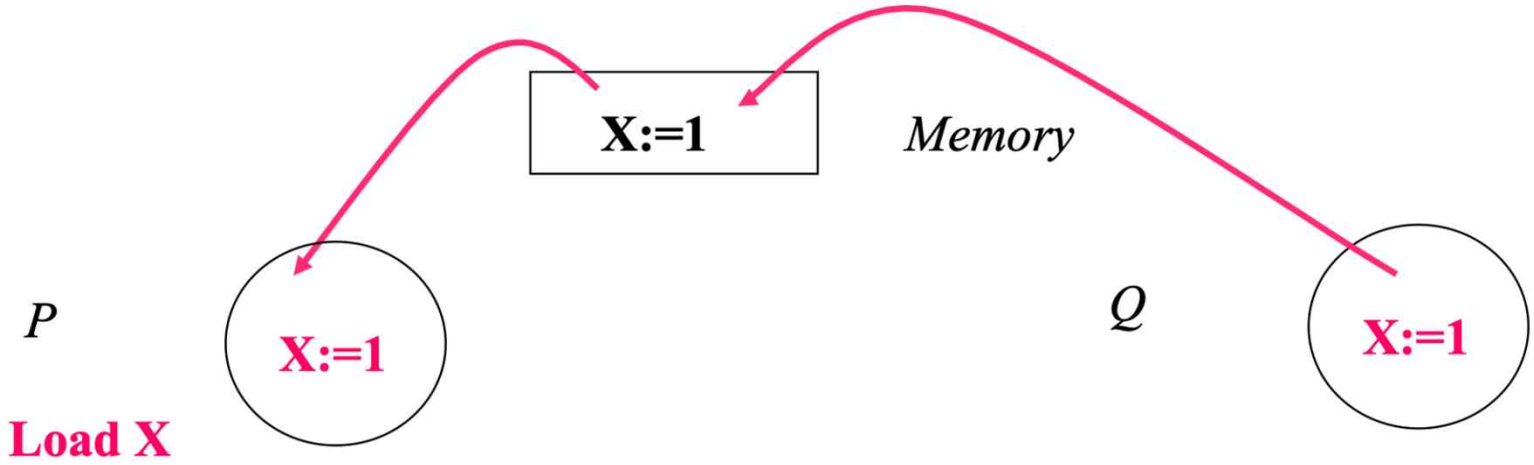
Program order

- If a processor writes and then reads the same location X, and there are no other intervening writes by other processors to X , then the read will always return the value previously written.



Definition of a coherent view of memory

- If a processor P reads from location X that was previously written by a processor Q , then the read will return the value previously written, if a sufficient amount of time has elapsed between the read and the write.



Serialization of writes

- If two processors write to the same location X, then other processors reading X will observe the same the sequence of values in the order written
- If 10 and then 20 is written into X, then no processor can read 20 and then 10

Cache Coherence in SMP

- Problem - multiple copies of same data in different caches
- Can result in an inconsistent view of memory
  - Write back policy can lead to inconsistency
  - Write through can also give problems unless caches monitor memory traffic

Hardware Solution to Cache Coherence

- Cache coherence hardware protocols
  - Dynamic recognition of potential problems
  - Run time solution
    - More efficient use of cache
  - Transparent to programmer / Compiler

- Implemented with:
  - Directory protocols
  - Snoopy protocols

Directory & Snoopy Protocols

- Directory Protocols
  - Effective in large scale systems with complex interconnection schemes
  - Collect and maintain information about copies of data in cache
    - Directory stored in main memory
  - Requests are checked against directory
    - Appropriate transfers are performed
  - Creates central bottleneck
- Snoopy Protocols
  - Suited to bus based multiprocessor
  - Distribute cache coherence responsibility among cache controllers
  - Cache recognizes that a line is shared
  - Updates announced to other caches
  - Increases bus traffic

- Write Update Protocol (Write Broadcast)
  - Multiple readers and writers
  - Updated word is distributed to all other processors
  - Multiple readers, one writer

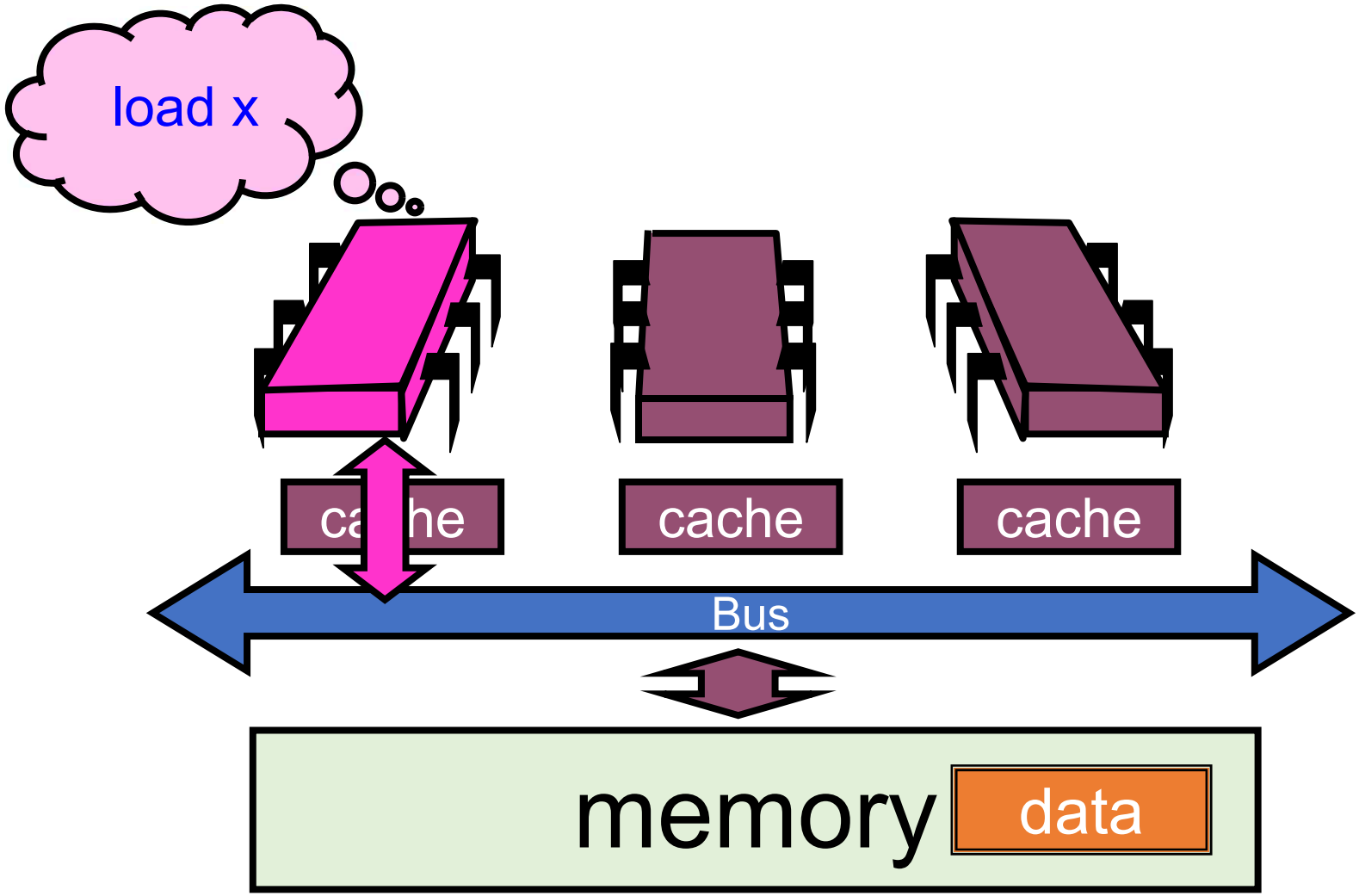
- Write Invalidate protocol (MESI)
  - When a write is required, all other caches of the line are invalidated
  - Writing processor then has exclusive (cheap) access until line is required by another processor

  - MESI Protocol - State of every line is marked as Modified, Exclusive, Shared or Invalid
    - two bits are included with each cache tag

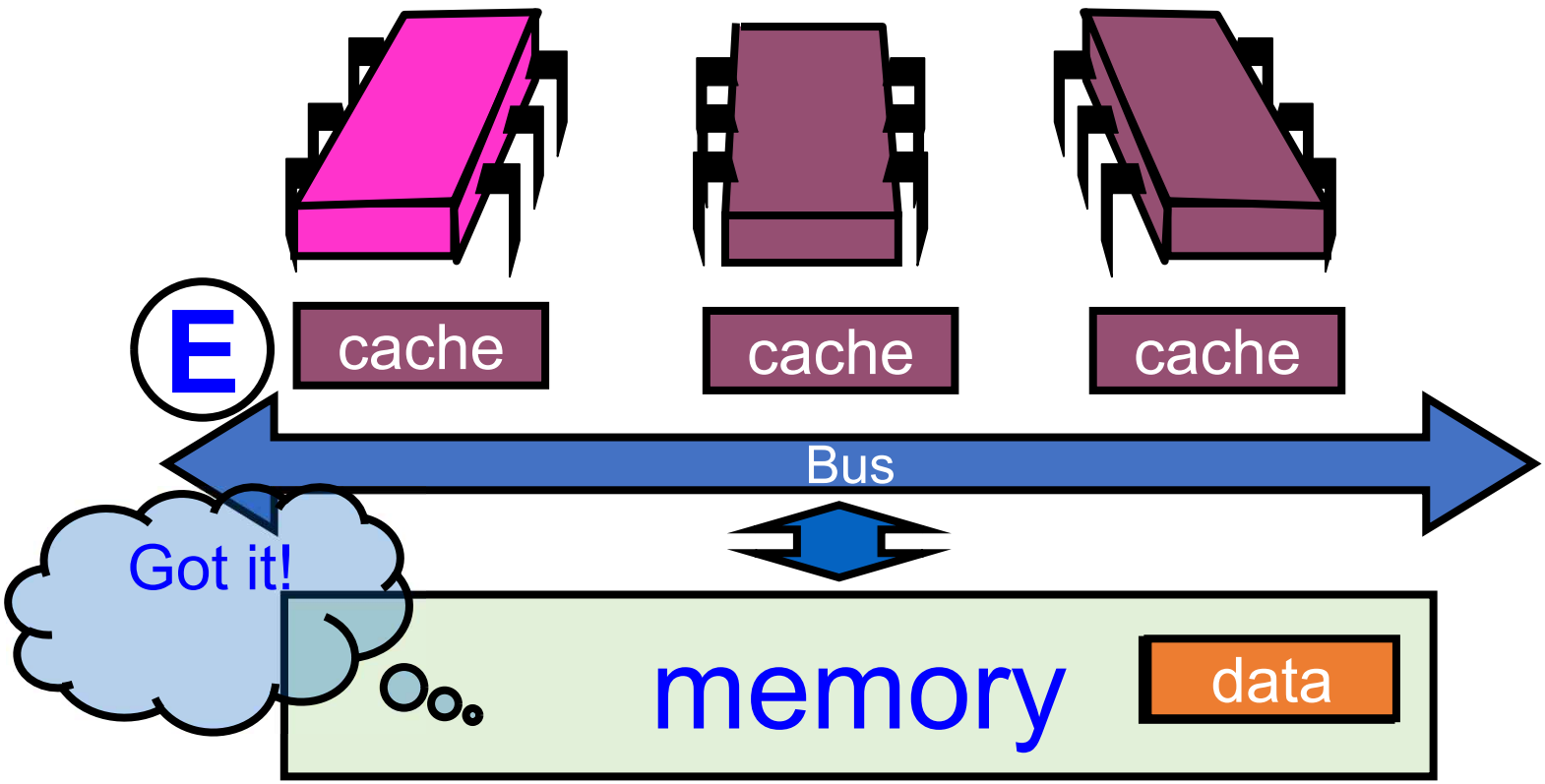
MESI

- Modified
  - Have modified cached data, must write back to memory
- Exclusive
  - Not modified, I have only copy
- Shared
  - Not modified, may be cached elsewhere
- Invalid
  - Cache contents not meaningful

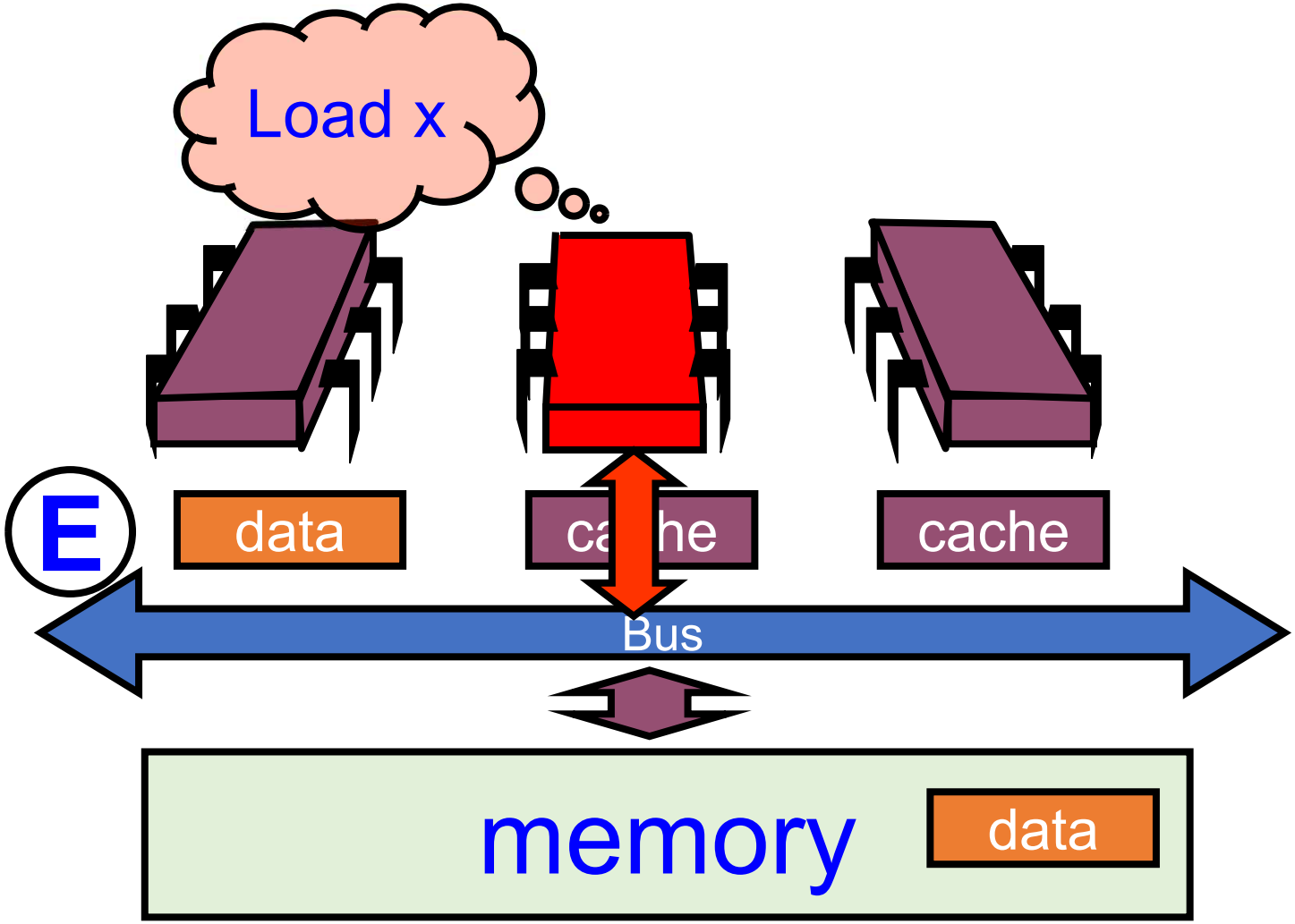
Processor Issues Load Request



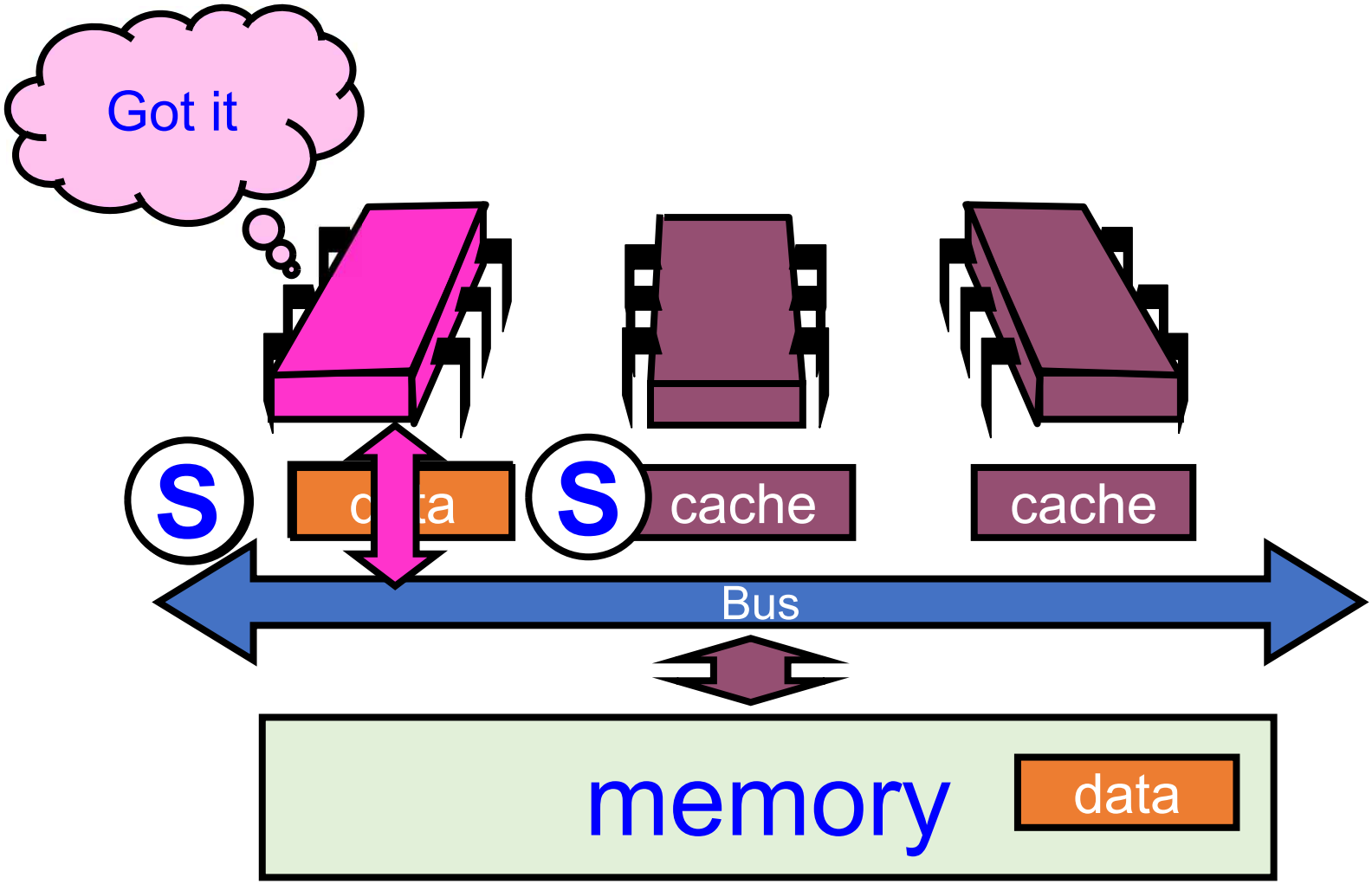
Memory Responds



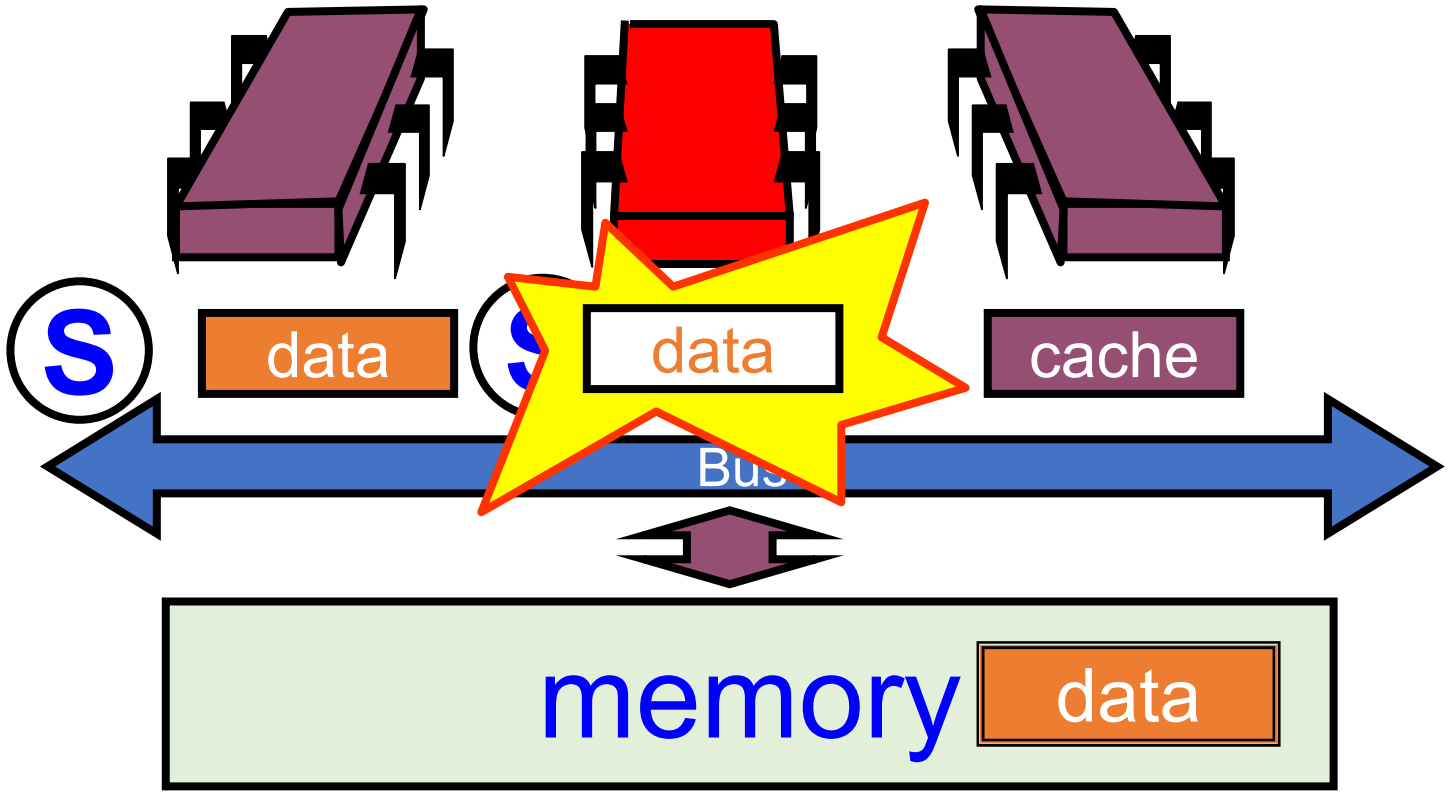
Processor Issues Load Request



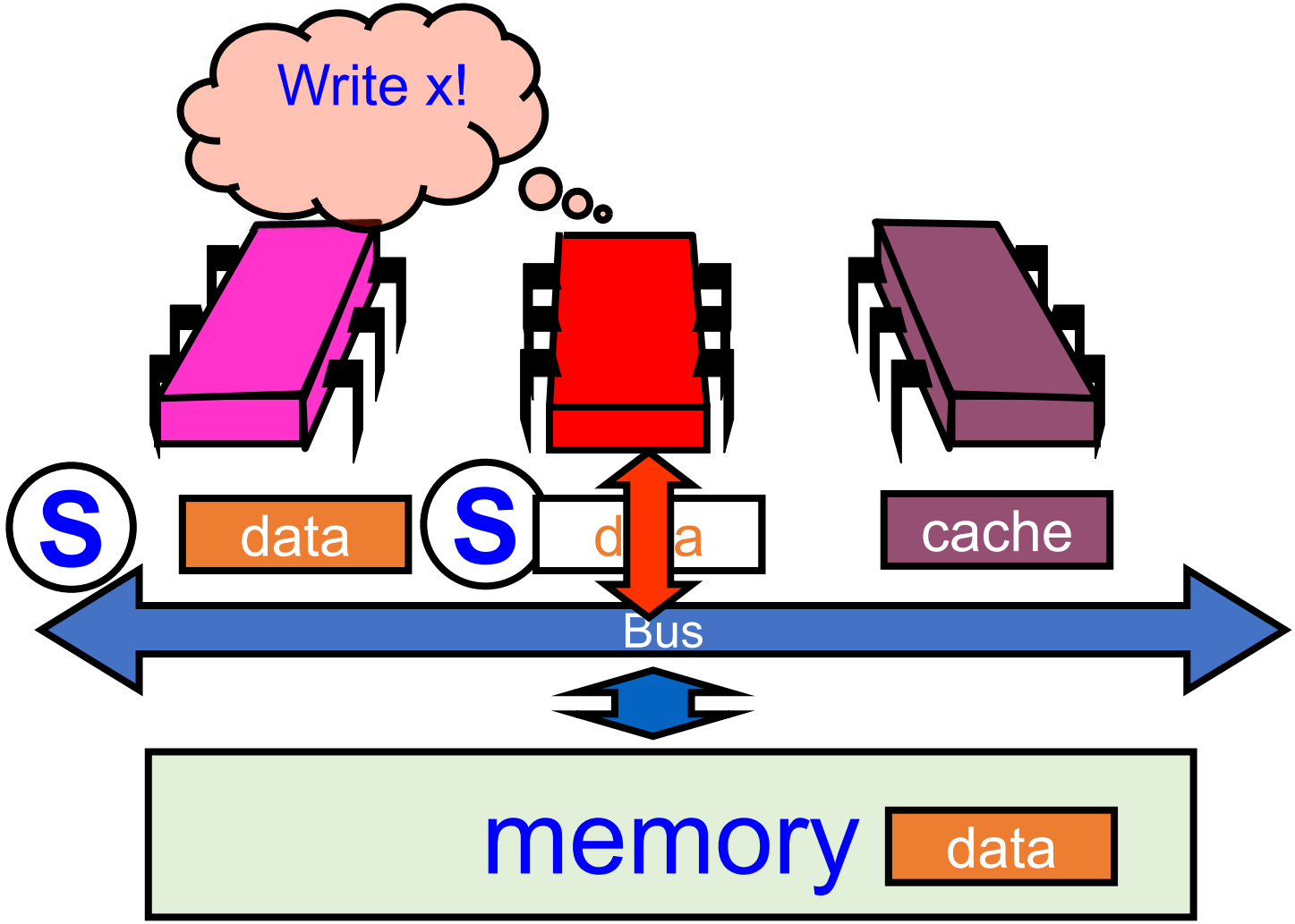
Other Processor Responds



Modify Cached Data



Write-Through Cache



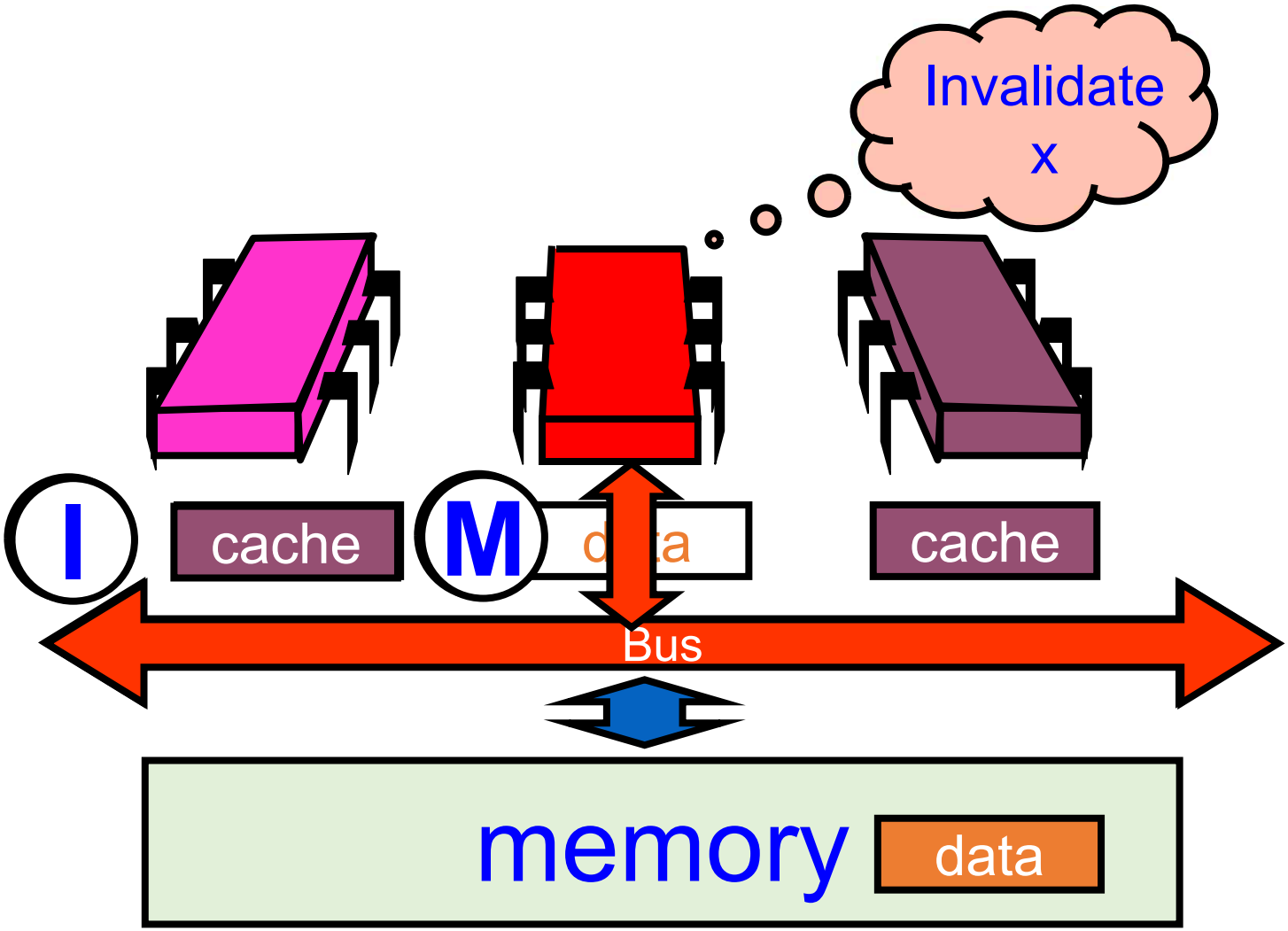
Write-Through Caches

- Immediately broadcast changes
- Good
  - Memory, caches always agree
  - More read hits, maybe
- Bad
  - Bus traffic on all writes
  - Most writes to unshared data
  - For example, loop indexes …

Write-Back Caches

- Accumulate changes in cache
- Write back when line evicted
  - Need the cache for something else
  - Another processor wants it

Invalidate



numactl – set memory affinity

- View the NUMA structure (on Linux):
- `numactl --hardware`
- numactl can also be used to control NUMA behavior

- `numactl [--interleave=nodes] [--preferred=node] [--membind=nodes] [--localalloc] <command> [args] ...`
- `numactl --show`
- `numactl –interleave=0,1 <command>`