Lecture 15: Snoopy Coherence Protocols

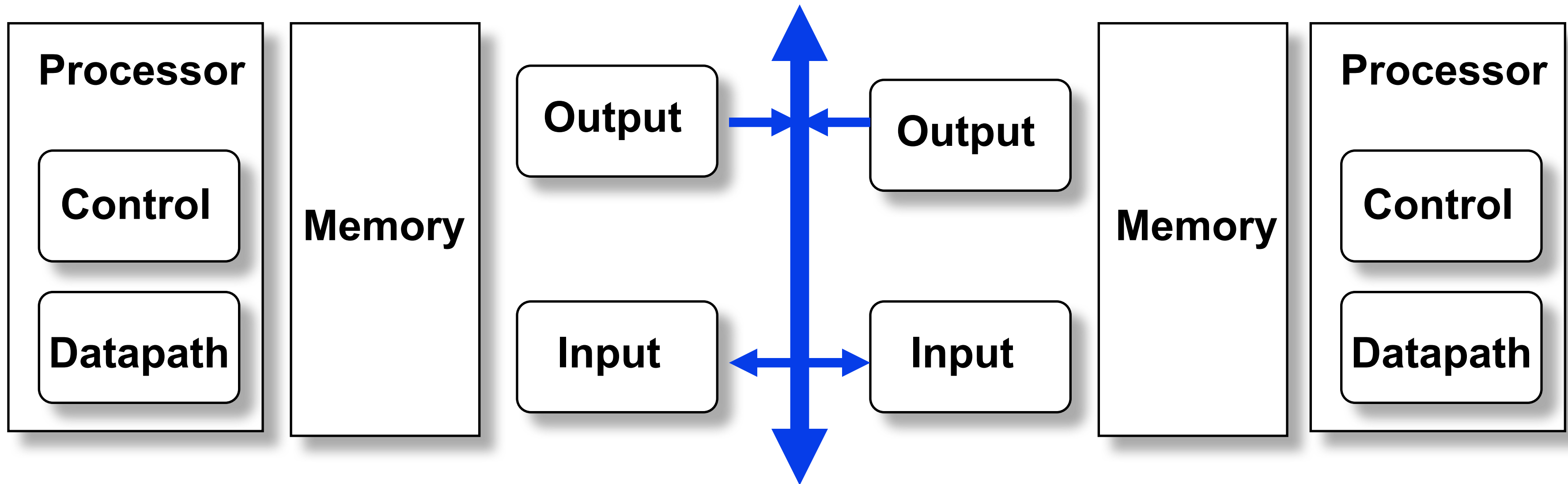
Vassilis Papaefstathiou

Iakovos Mavroidis

Computer Science Department

University of Crete

Where are We Now?



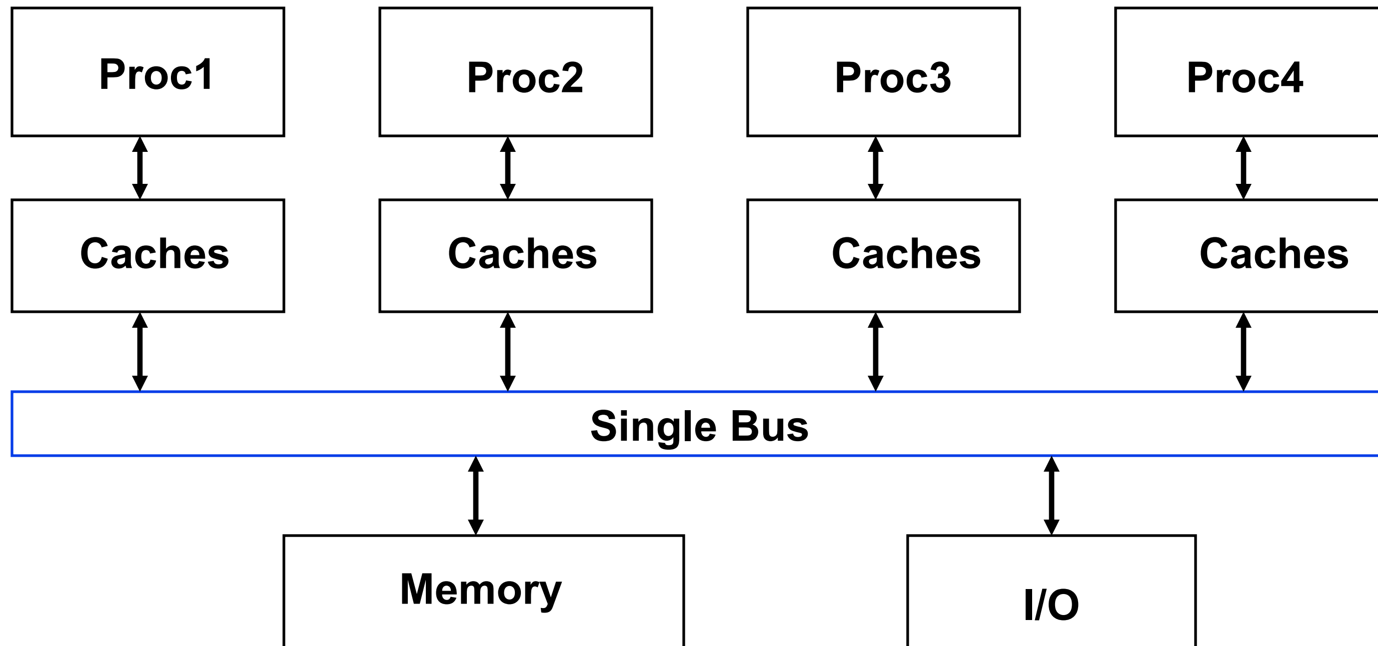
- Multiprocessor – multiple processors with a single shared address space
- Cluster – multiple computers (each with their own address space) connected over a local area network (LAN) functioning as a single system

Multiprocessor Basics

- Q1 – How do they share data?
- Q2 – How do they coordinate?
- Q3 – How scalable is the architecture? How many processors?

<table>
<tr><td></td><td colspan="2"></td><td># of Proc</td></tr>
<tr><td rowspan="3">Communication model</td><td colspan="2">Message passing</td><td>8 to 2048</td></tr>
<tr><td rowspan="2">Shared address</td><td>NUMA</td><td>8 to 256</td></tr>
<tr><td>UMA</td><td>2 to 64</td></tr>
<tr><td rowspan="2">Physical connection</td><td colspan="2">Network</td><td>8 to 256</td></tr>
<tr><td colspan="2">Bus</td><td>2 to 36</td></tr>
</table>

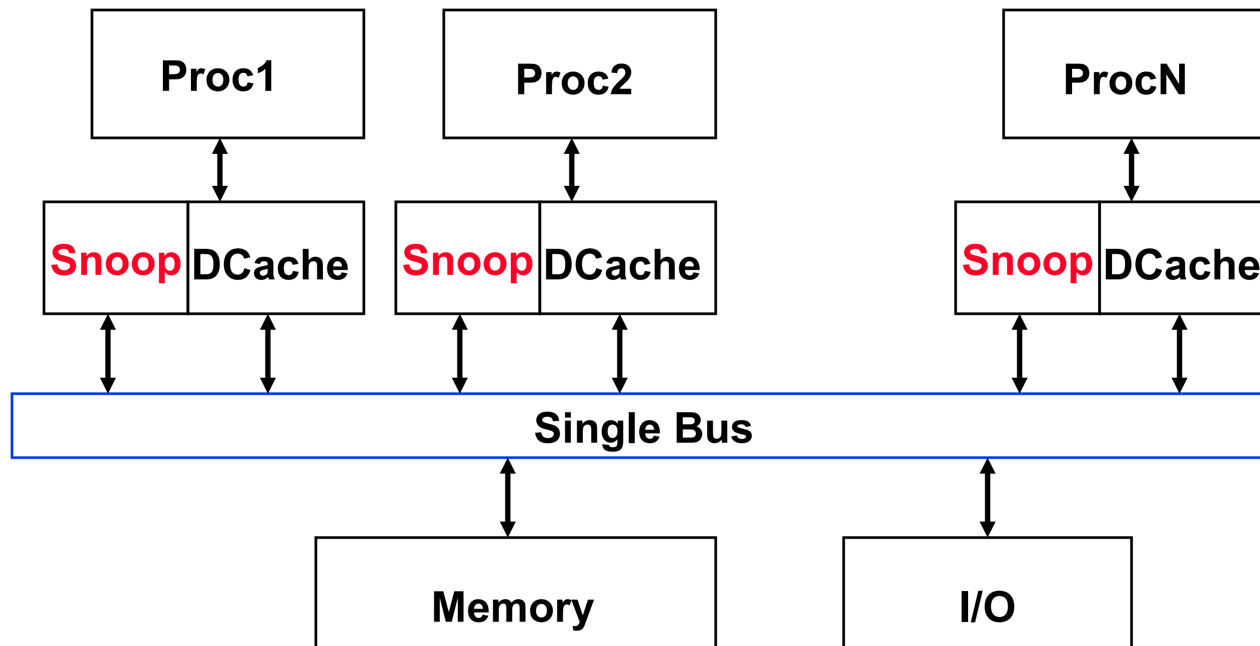
Single Bus (Shared Address UMA) Multi's



- Caches are used to reduce latency and to lower bus traffic
  - Write-back caches used to keep bus traffic at a minimum
- Must provide hardware to ensure that caches and memory are consistent (cache coherency)
- Must provide a hardware mechanism to support process synchronization

Multiprocessor Cache Coherency

- Cache coherency protocols
  - Bus snooping – cache controllers monitor shared bus traffic with duplicate address tag hardware (so they don't interfere with processor's access to the cache)



Bus Snooping Protocols

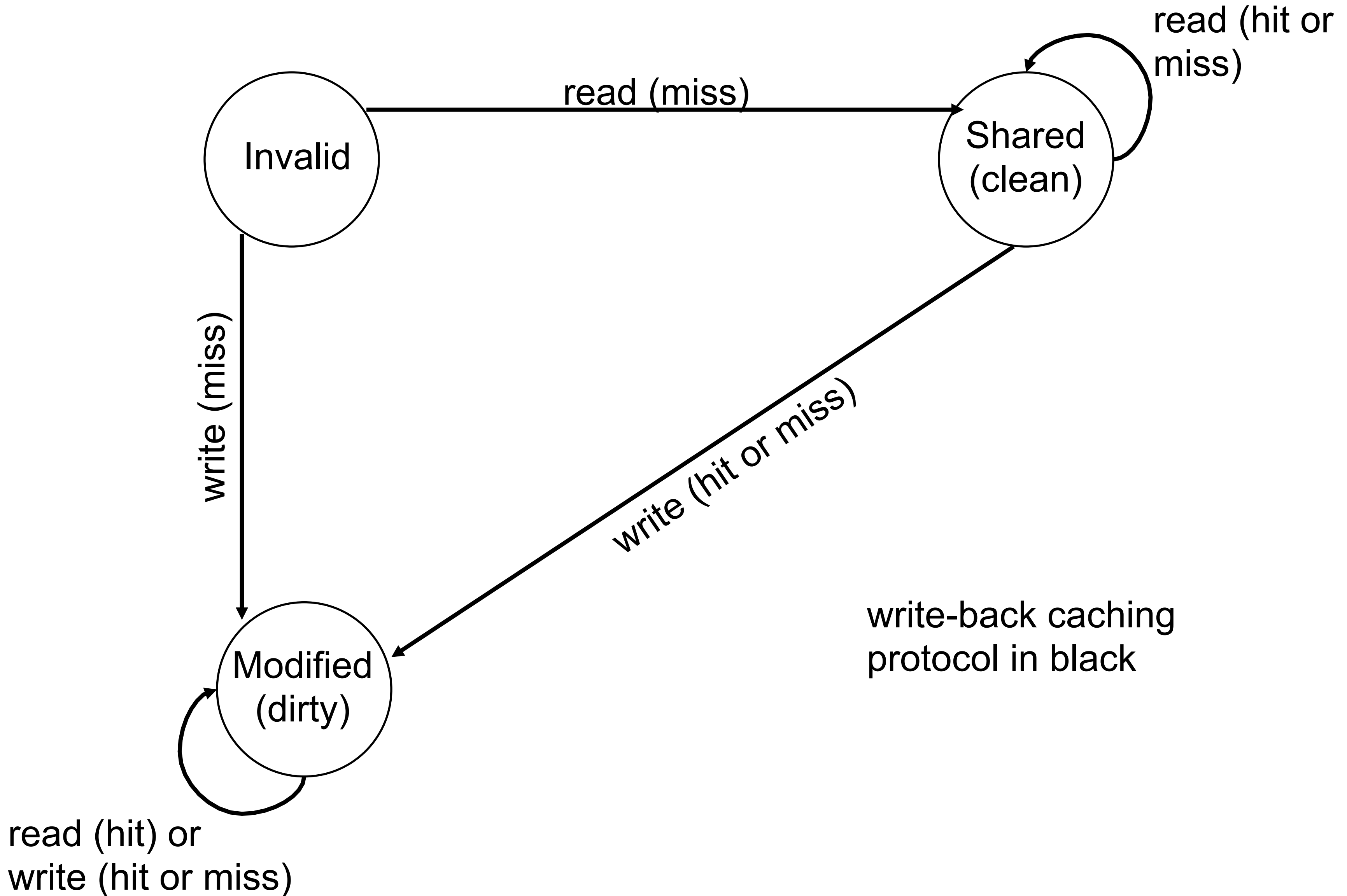
- Multiple copies are not a problem when reading
- Processor must have exclusive access to write a word
  - What happens if two processors try to write to the same shared data word in the same clock cycle? The bus arbiter decides which processor gets the bus first (and this will be the processor with the *first* exclusive access). Then the second processor will get exclusive access. Thus, bus arbitration forces sequential behavior.
  - This sequential consistency is the most conservative of the memory consistency models. With it, the result of any execution is the same as if the accesses of each processor were kept in order and the accesses among different processors were interleaved.
- All other processors sharing that data must be informed of writes

Handling Writes

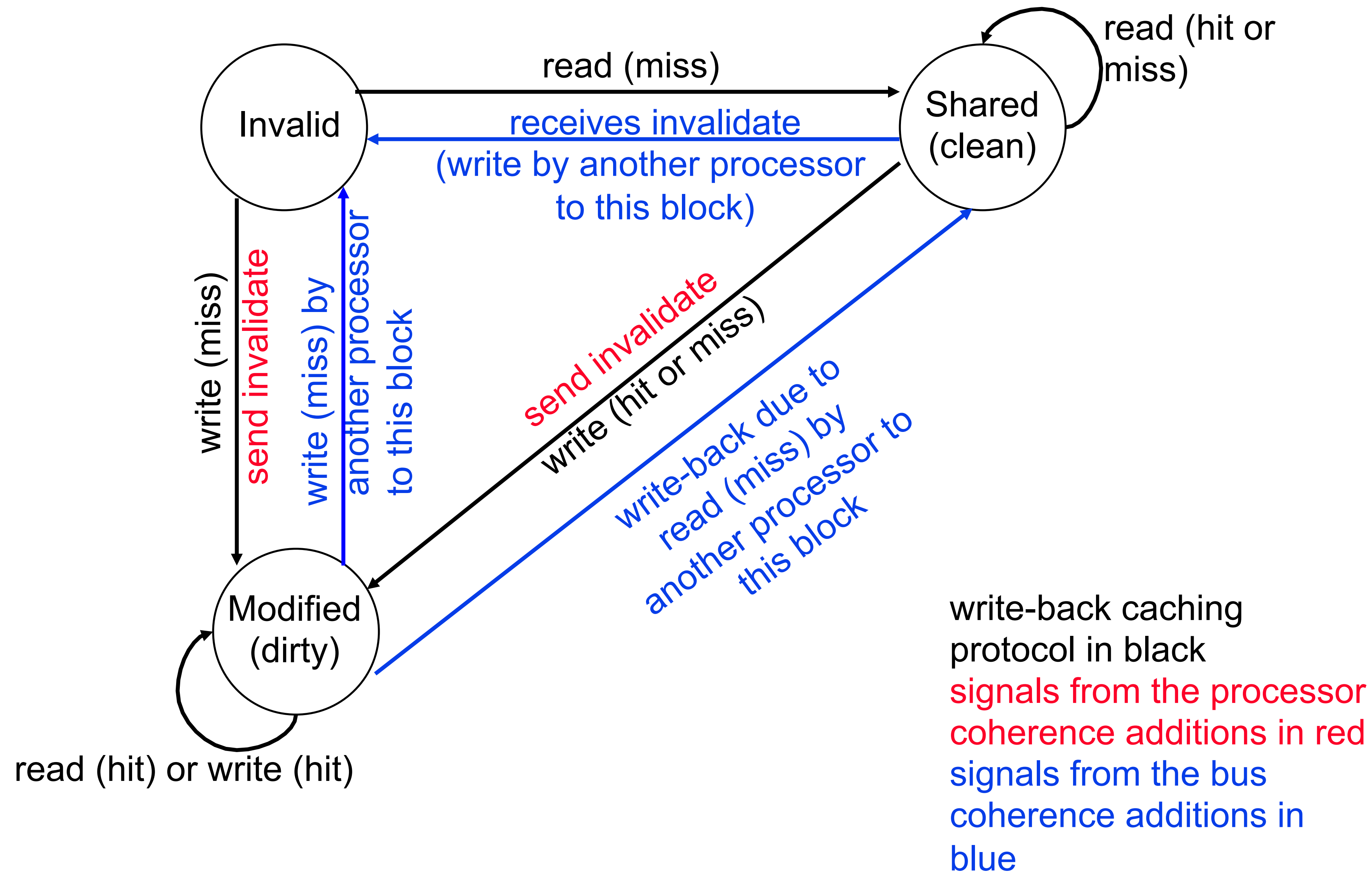
Ensuring that all other processors sharing data are informed of writes can be handled two ways:

1. Write-update (write-broadcast) – writing processor broadcasts new data over the bus, all copies are updated
   - All writes go to the bus → higher bus traffic
   - Since new values appear in caches sooner, can reduce latency
2. Write-invalidate – writing processor issues invalidation signal on bus, cache snoops check to see if they have a copy of the data, if so they invalidate their cache block containing the word (this allows multiple readers but only one writer)
   - Uses the bus only on the first write → lower bus traffic, so better use of bus bandwidth

A Write-Invalidate CC Protocol

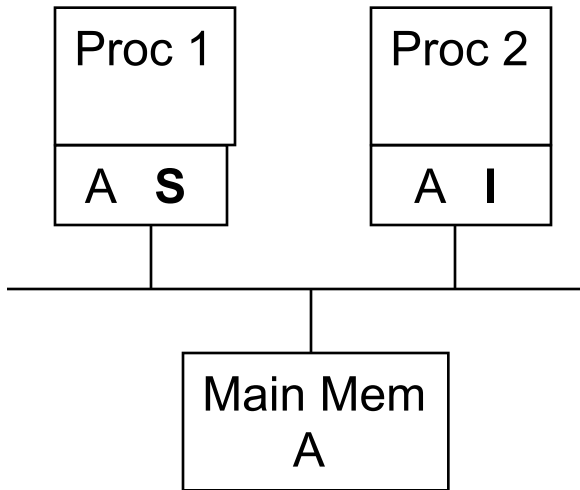


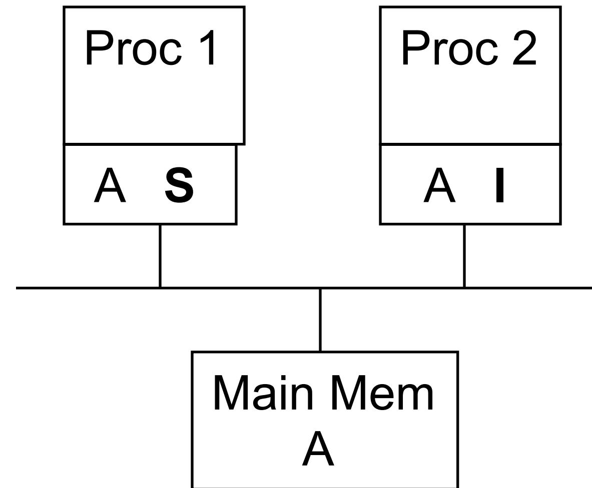
A Write-Invalidate CC Protocol

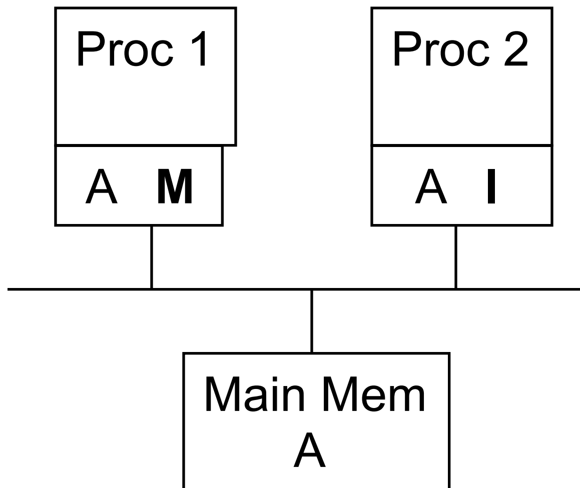


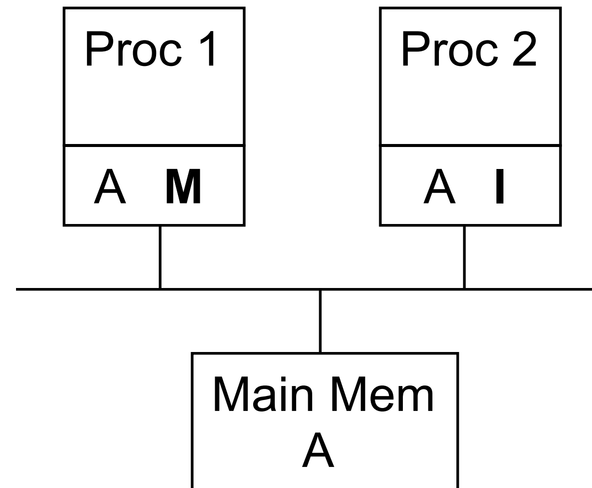
Write-Invalidate CC Examples

- I = invalid (many), S = shared (many), M = modified (only one)



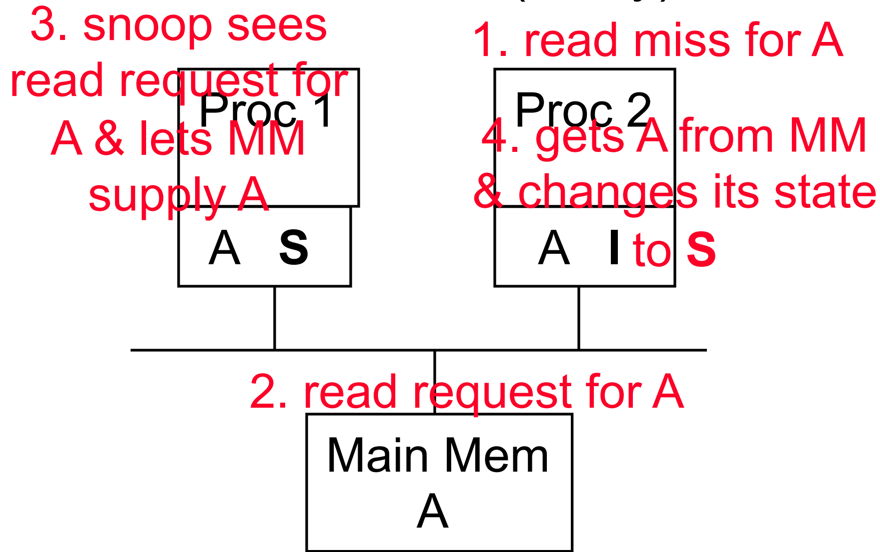


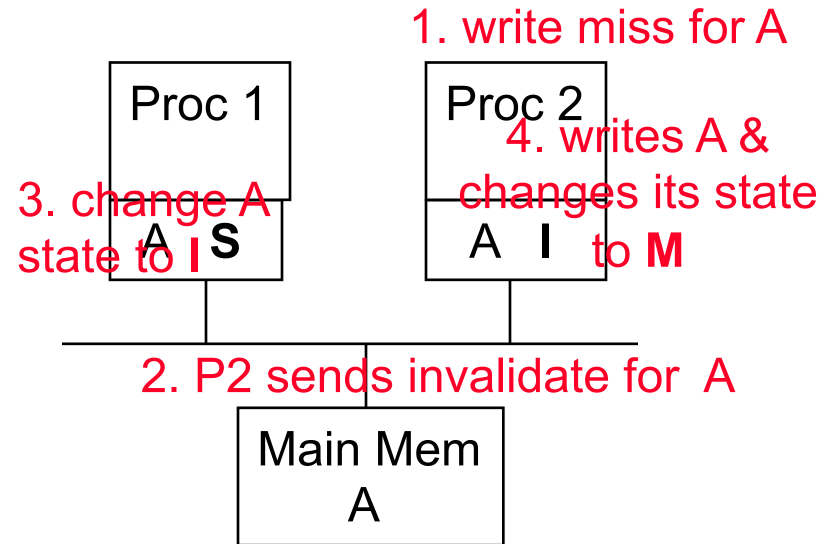


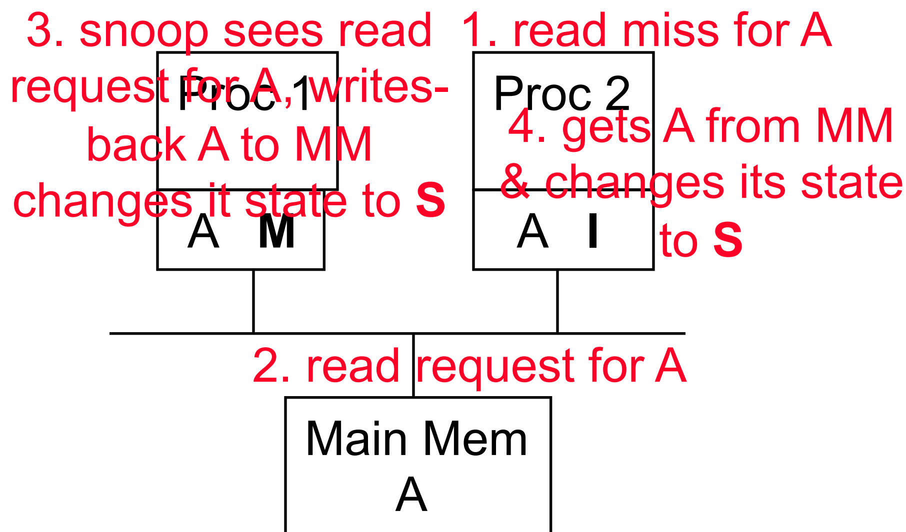


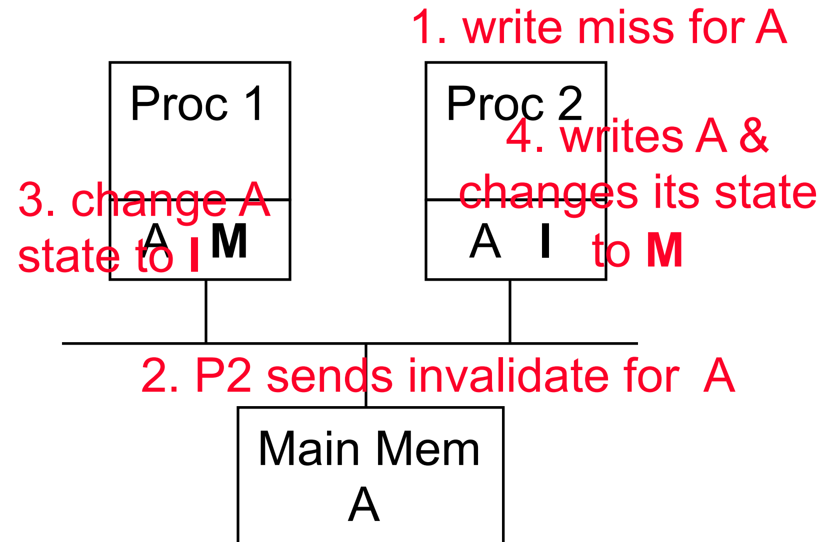
Write-Invalidate CC Examples

- I = invalid (many), S = shared (many), M = modified (only one)







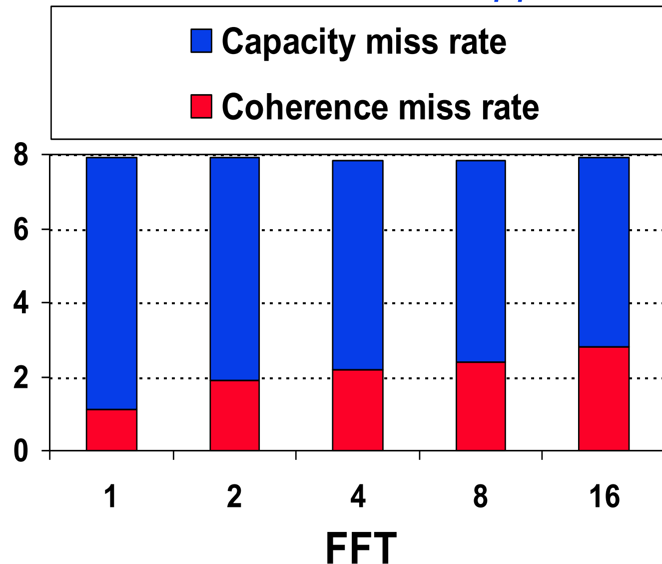


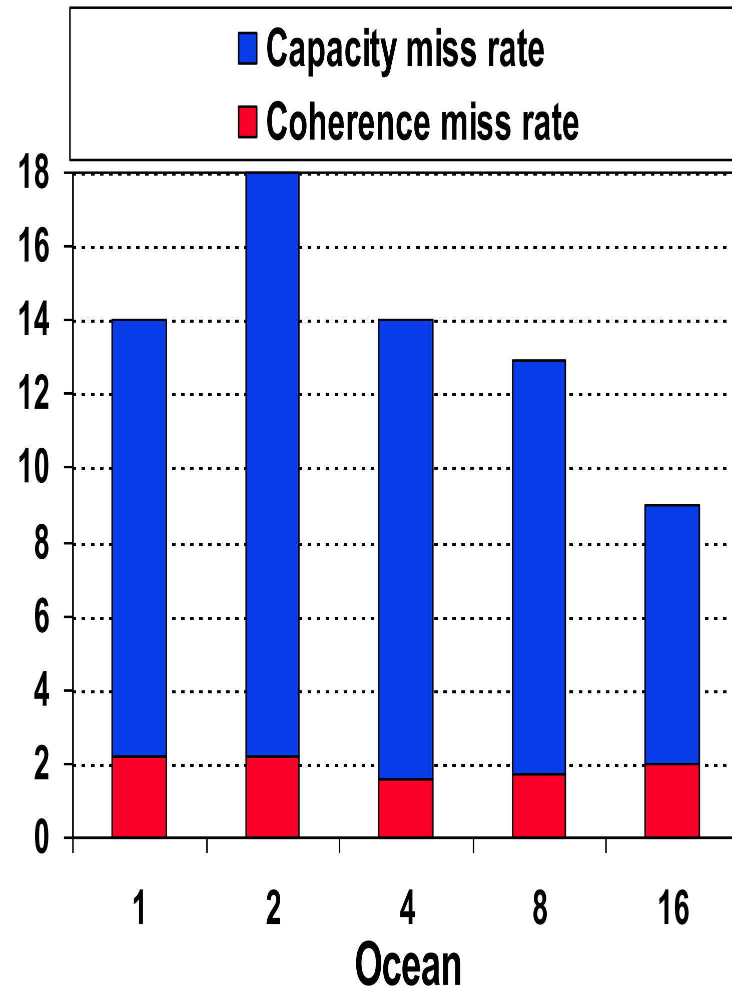
SMP Data Miss Rates

- Shared data has lower spatial and temporal locality
  - Share data misses often dominate cache behavior even though they may only be 10% to 40% of the data accesses

64KB 2-way set associative data cache with 32B blocks

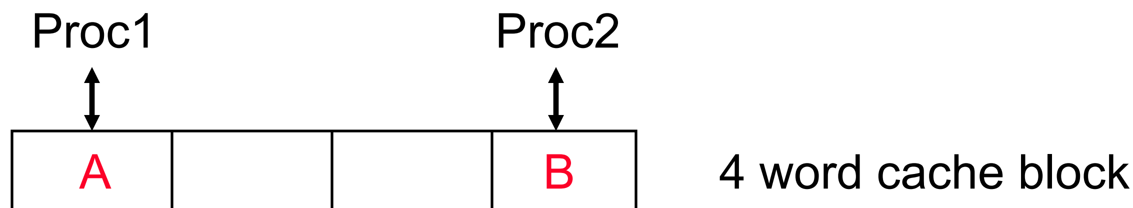
Hennessy & Patterson, *Computer Architecture: A Quantitative Approach*





Block Size Effects

- Writes to one word in a multi-word block mean
  - either the full block is invalidated (write-invalidate)
  - or the full block is exchanged between processors (write-update)
    - Alternatively, could broadcast *only* the written word
- Multi-word blocks can also result in false sharing: when two processors are writing to two different variables in the same cache block
  - With write-invalidate false sharing increases cache miss rates



- Compilers can help reduce false sharing by allocating highly correlated data to the same cache block

MESI Protocol (1)

- There are many variations on cache coherence protocols

- Another write-invalidate protocol used in the Pentium 4 (and many other micro's) is MESI with four states:
  - Modified – (same) only modified cache copy is up-to-date; memory copy and all other cache copies are out-of-date
  - Exclusive – only one copy of the shared data is allowed to be cached; memory has an up-to-date copy
    - Since there is only one copy of the block, write hits don't need to send invalidate signal
  - Shared – multiple copies of the shared data may be cached (i.e., data permitted to be cached with more than one processor); memory has an up-to-date copy
  - Invalid – same

MESI Protocol (2)

- Cache line changes state as a function of memory access events.
- Event may be either
  - Due to local processor activity (i.e. cache access)
  - Due to bus activity - as a result of snooping
- Cache line has its own state affected only if address matches

MESI Protocol (3)

- Operation can be described informally by looking at action in local processor
  - Read Hit
  - Read Miss
  - Write Hit
  - Write Miss
- More formally by state transition diagram

MESI Local Read Hit

- Line must be in one of MES
- This must be correct local value (if M it must have been modified locally)
- Simply return value
- No state change

MESI Local Read Miss (1)

- No other copy in caches
  - Processor makes bus request to memory
  - Value read to local cache, marked E
- One cache has E copy
  - Processor makes bus request to memory
  - Snooping cache puts copy value on the bus
  - Memory access is abandoned
  - Local processor caches value
  - Both lines set to S

MESI Local Read Miss (2)

- Several caches have S copy
  - Processor makes bus request to memory
  - One cache puts copy value on the bus (arbitrated)
  - Memory access is abandoned
  - Local processor caches value
  - Local copy set to S
  - Other copies remain S

MESI Local Read Miss (3)

- One cache has M copy
  - Processor makes bus request to memory
  - Snooping cache puts copy value on the bus
  - Memory access is abandoned
  - Local processor caches value
  - Local copy tagged S
  - **Source (M) value copied back to memory**
  - Source value M -> S

MESI Local Write Hit (1)

Line must be one of MES

- M
  - line is exclusive and already ‘dirty’
  - Update local cache value
  - no state change
- E
  - Update local cache value
  - State E -> M

MESI Local Write Hit (2)

- S
  - Processor broadcasts an invalidate on bus
  - Snooping processors with S copy change S->I
  - Local cache value is updated
  - Local state change S->M

MESI Local Write Miss (1)

Detailed action depends on copies in other processors

- No other copies
  - Value read from memory to local cache (?)
  - Value updated
  - Local copy state set to M

MESI Local Write Miss (2)

- Other copies, either one in state E or more in state S
  - Value read from memory to local cache - bus transaction marked RWITM (read with intent to modify)
  - Snooping processors see this and set their copy state to I
  - Local copy updated & state set to M

MESI Local Write Miss (3)

Another copy in state M

- Processor issues bus transaction marked RWITM
- Snooping processor sees this
  - Blocks RWITM request
  - Takes control of bus
  - Writes back its copy to memory
  - Sets its copy state to I

MESI Local Write Miss (4)

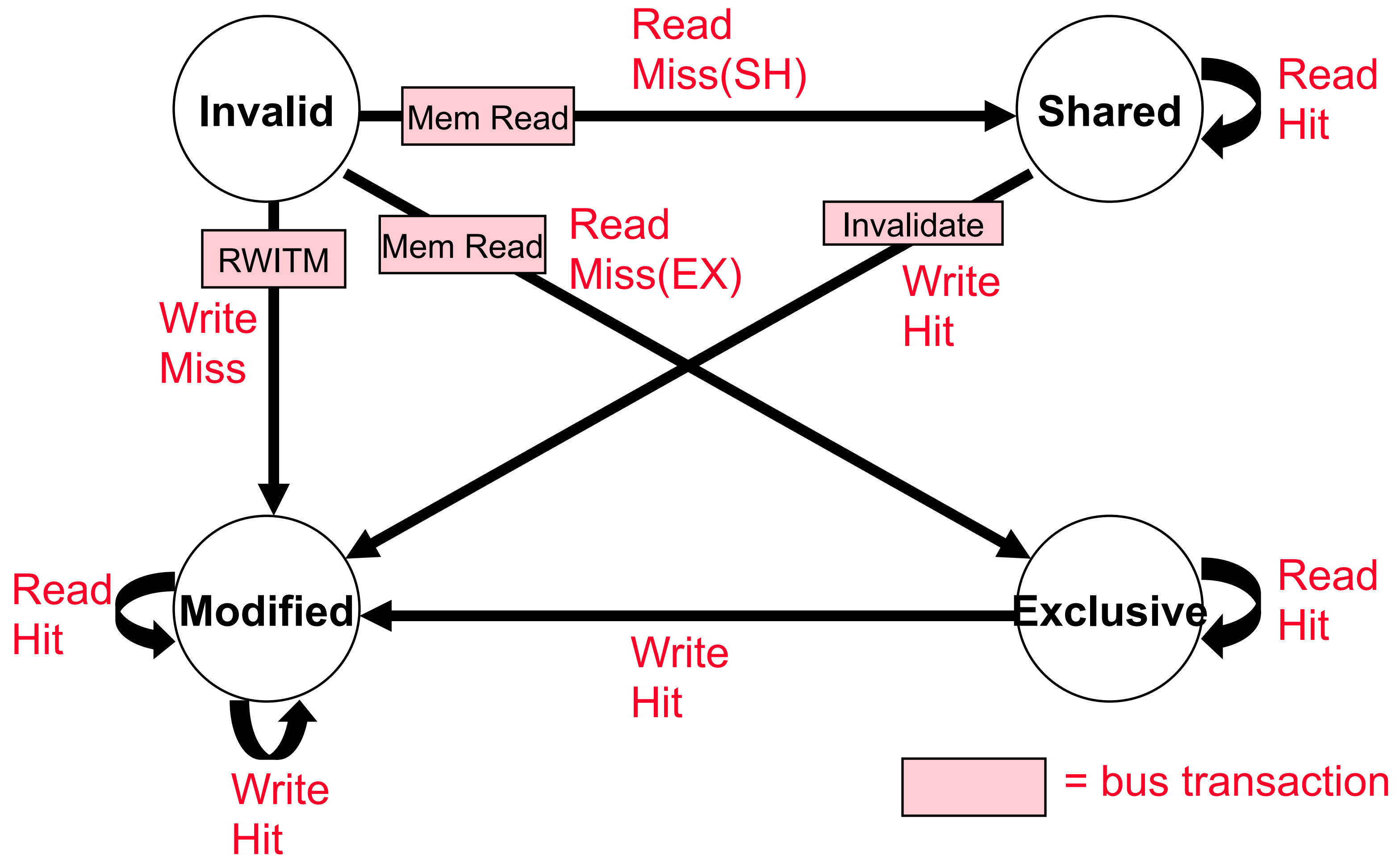
Another copy in state M (continued)

- Original local processor re-issues RWITM request
- Is now simple no-copy case
  - Value read from memory to local cache
  - Local copy value updated
  - Local copy state set to M

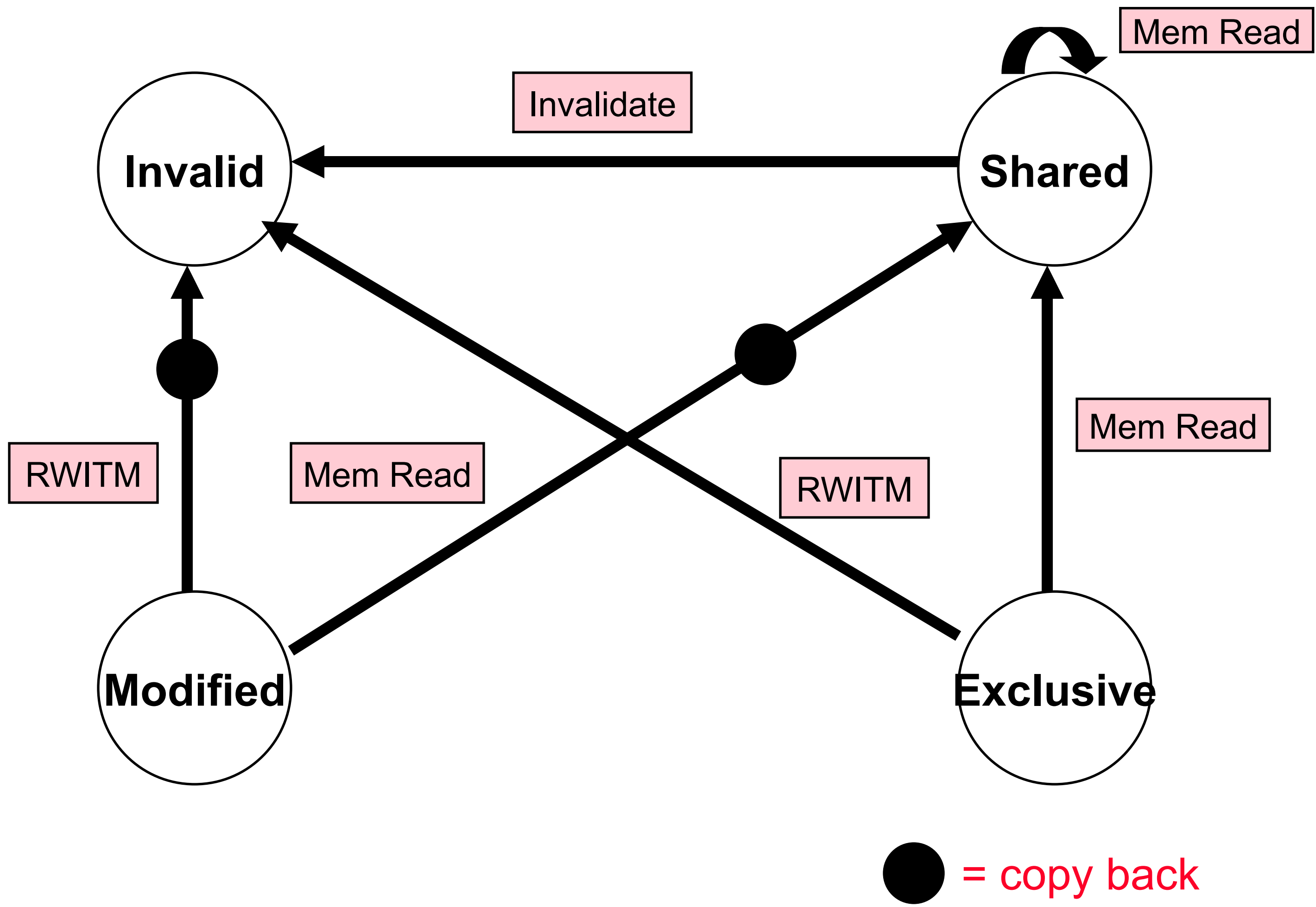
Putting it all together

- All of this information can be described compactly using a state transition diagram
- Diagram shows what happens to a cache line in a processor as a result of
  - memory accesses made by that processor (read hit/miss, write hit/miss)
  - memory accesses made by other processors that result in bus transactions observed by this snoopy cache (Mem read, RWITM,Invalidate)

MESI – locally initiated accesses



MESI – remotely initiated accesses



MESI notes

- There are minor variations (particularly to do with write miss)
- Normal ‘write back’ when cache line is evicted is done if line state is M
- Multi-level caches
  - If caches are inclusive, only the lowest level cache needs to snoop on the bus