Contents

Loosely Coupled (Multi-Computer) 1

Tightly Coupled (Multiprocessor) 1

Alternative Tightly Coupled Multiprocessor Organisation 1

Multiprocessor Cache Coherency 2

Write-Through Protocol 2

Transition Diagram 3

Write-Once Protocol 3

Transition Diagram 4

Firefly Protocol 4

Transition Diagram 7

MESI Cache Coherency Protocol 7

Transition Diagram 8

Loosely Coupled (Multi-Computer)

- Each CPU has its own memory, I/O facilities and OS
- CPUs **DO NOT** share *physical* memory
- Distributed systems…

Tightly Coupled (Multiprocessor)

- CPUs physically share memory and I/O
- Inter-processor communicate via shared memory
- Symmetric multiprocessing possible (ie. Single shared copy of operating system)
- Critical sections protected by locks/semaphores
- Processes can easily migrate between CPUs

Alternative Tightly Coupled Multiprocessor Organisation

- Shared 2nd level cache
- Cache coherency handled directly by CPUs

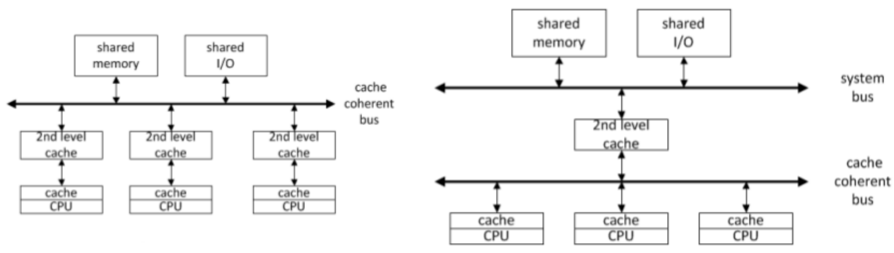

Figure 1: Alternative Multiprocessor Organisations

Multiprocessor Cache Coherency

- What's the problem? Must guarantee that a CPU always reads the most up to date value of a *memory location*

1. CPU 1 reads location X, copy of X stored in cache A
2. CPU B reads location X, copy of X stored in cache B
3. CPU A writes to location X X in cache and main memory updates (write-through)
4. CPU B reads X, **BUT** gets out of date (stale) copy from its cache B

- Solutions based on *bus watching* or *snoopy* caches
- *Snoopy* caches watch all bus transactions and update their internal state according to a pre-determined cache coherency protocol
- Cache coherency protocols given names
  - E.g. Dragon, **Firefly**, **MESI**, Berkeley, Illinois, **Write-Once** and **Write-Through**

Write-Through Protocol

- Key ideas
  - Uses write-through caches
  - When a cache observes a bus *write* to a memory location it has a copy of, it simply invalidates the cache line (a write-invalidate protocol)
  - The *next* time the CPU accesses the same memory location/cache line, the data *will* be fetched from memory (hence getting the most up to date value)

Transition Diagram

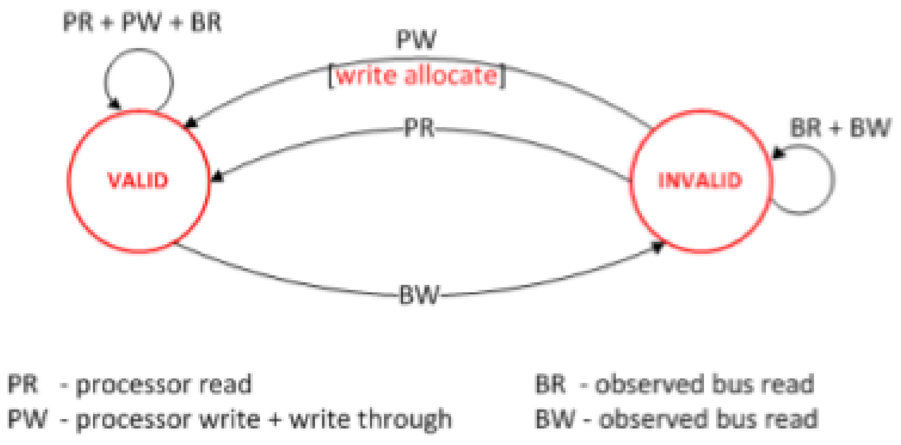

Figure 2: Transition Diagram

- Each cache line is in either the VALID or INVALID state (if an address is not in the cache, it is captured by the INVALID state)
- PW = processor write and write through to memory
- bus traffic = sum writes (write about 20% of memory accesses)
- Straightforward to implement and effective for small scale parallelism (< 6 CPUs)

Write-Once Protocol

- Uses write-deferred caches (to reduce bus traffic)
  - BUT still uses a write-invalidate protocol
- Each cache line can be in one of 4 states
  - **INVALID**
  - **VALID**: cache line in *one or more* caches, cache copies = memory
  - **RESERVED**: cache line in one cache ONLY, cache copy = memory
  - **DIRTY**: cache line in one cache ONLY and it is the ONLY up to date copy (memory copy out of date/stale)
- Key ideas
  - Write-through to VALID cache lines
  - Write back (local write) to RESERVED/DIRTY cache lines (as we know cache line in one cache ONLY)

Transition Diagram

- When a memory location is read initially, it enters the cache in the VALID state
- A cache line in the VALID state changes to the RESERVED state when written for the first time (hence the name write-once)
- A write which causes a transition to the RESERVED state is written through to memory so that all other caches can observe the transaction and invalidate their copies if present (cache line now in one cache ONLY)
- Subsequent writes to a RESERVED cache line will use write-deferred cycles and the cache line marked as DIRTY as it is now the only up to date copy in the system (memory copy out of date)
- Caches must monitor bus for any reads or writes to its RESERVED and DIRTY cache lines
  - If a cache observes a read from one of its RESERVED cache lines, it simply changes its state to VALID
  - If a cache observes a read from one of its DIRTY cache lines, the cache must intervene (intervention cycle) and supply the data onto the bus to the requesting cache, the data is *also* written to memory and the state of both cache lines are changed to VALID
  - NB: behaviour on an observed write (DIRTY => INVALID)

Firefly Protocol

- Used in the explerimental Firefly DEC workstation
  - DEC = Digital Equipment Corporation
- Each cache line can be in one of 4 states:
  - ~Shared & ~Dirty (Exclusive & Clean)
  - ~Shared & Dirty (Exclusive and Dirty)
  - Shared & ~Dirty (Shared & Clean)
  - Shared & Dirty (Shared & Dirty)
- NB: these is NO INVALID state
- Key ideas
  - A cache *knows* if its cache lines are shared with other caches (may not actually be shared but that is OK)
  - When a cache line is read into the cache the *other* caches will assert a common SHARED bus line if they contain the same cache line
  - Writes to exclusive cache line are write-deferred
  - Writes to shared ache lines are *write-through* and the othe rcaches which contain the same cache lines are updated together with memory (write-update protocol)

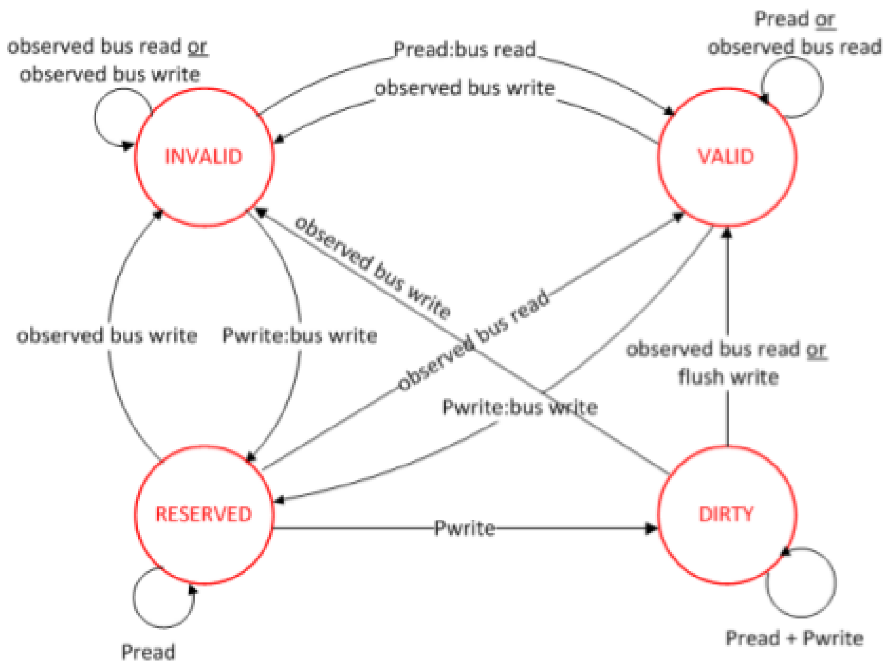

Figure 3: Transition Diagram

- When a cache line ceases to be shared, it needs an *extra* write-through cycle to find out that the cache line is no longer shared (SHARED signal will not be asserted)
- Sharing may be illusory e.g.
  * (i) processor may no longer be using a *shared* cache line
  * (ii) process migration between CPUs (sharing with an old copy of itself)

Transition Diagram

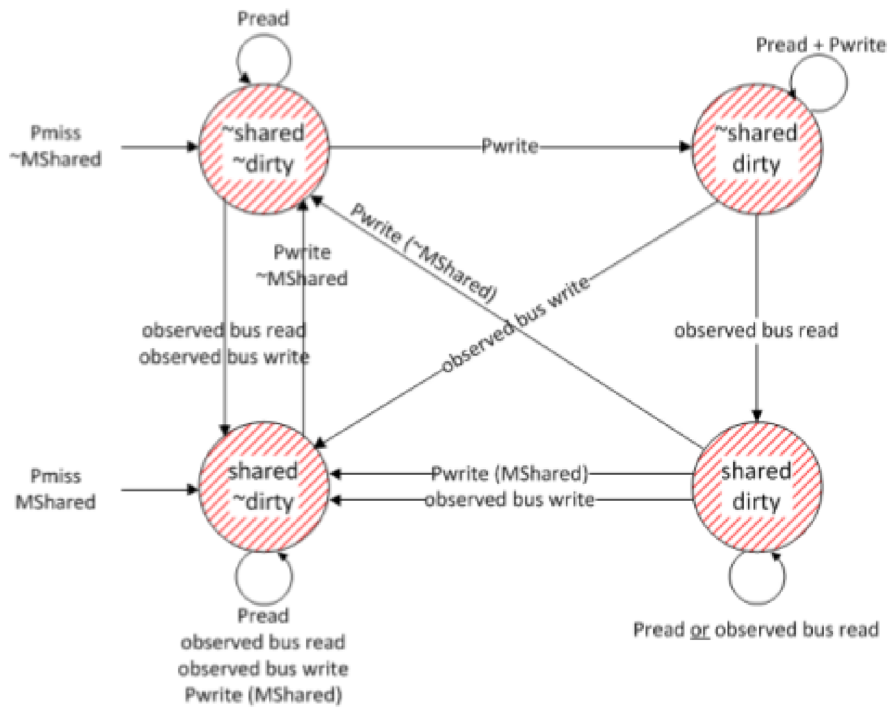

Figure 4: Transition Diagram

- As there is **NO** INVALID state
  - At reset the cache is placed in a *miss* mode and a bootstrap program gills cache with a sequence of addresses making it consistent with memory (VALID state)
  - During normal operation a location can be displaced if the cache line is needed to satisfy a miss, BUT the protocol never needs to invalidate cache line
- How is the Shared & Dirty state entered?
  - Asymmetrical behaviour with regard to updating memory (avoids changing memory cycle from a read cycle to a write cycle mid cycle)

MESI Cache Coherency Protocol

- Used by Intel CPUs

- Very similate to write-once BUT uses a *shared* bus signal (like Firefly) so cache line can enter cache and be placed directly in the shared or exclusive state
- A write-invalidate protocol
- MESI state compared with Write-Once states
  - *M*odified = dirty
  - *E*clusive = reserved
  - *S*hared = valid
  - *I*nvalid = invalid
- What is the difference between the MESI and Write-Once protocols?
  - Cache line may enter cache and be placed directly in the Exclusive (reserved) state
  - "Write-once" write-through cycles no longer necessary if cache line is Exclusive

Transition Diagram

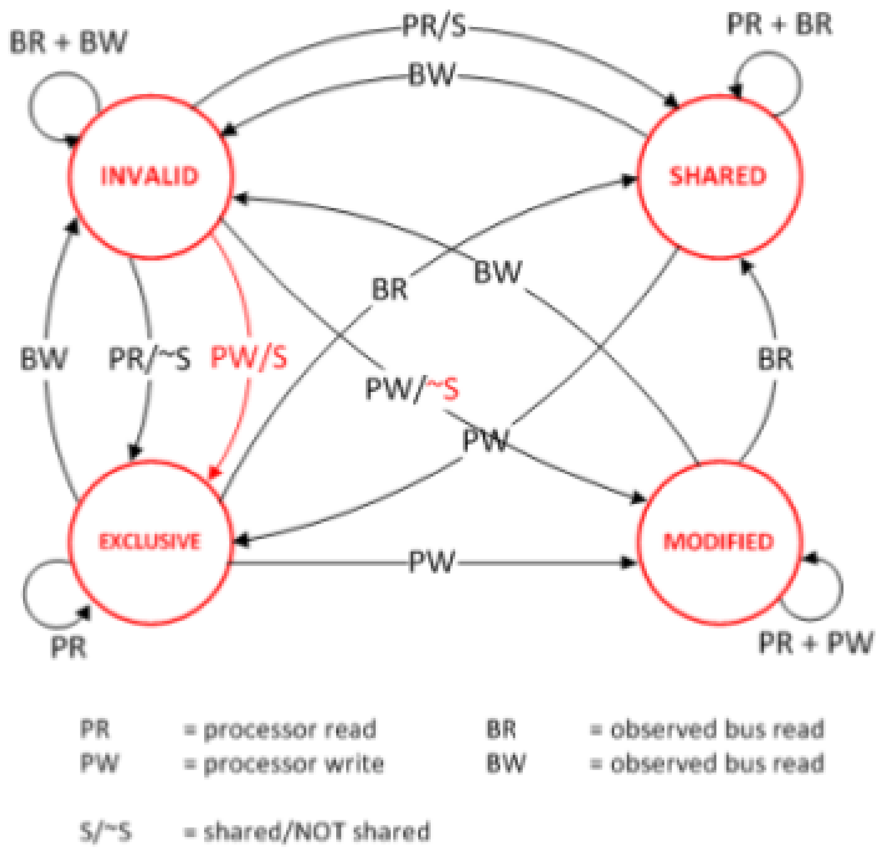

Figure 5: Transition Diagram