Cache coherency protocols (examples)

Examples of coherency protocols for cache memory are listed here. For simplicity, all "**miss**" Read and Write status transactions which obviously come from state "**I**" (or miss of Tag), in the diagrams are not shown. They are shown directly on the new state. Many of the following protocols have only historical value. At the moment the main protocols used are the R-MESI type / MESIF protocols and the HRT-ST-MESI (MOESI type) or a subset or an extension of these.

Contents

Cache coherency problem

In systems as Multiprocessor system, multi-core and NUMA system, where a dedicated cache for each *processor*, *core* or *node* is used, a consistency problem may occur when a same data is stored in more than one cache. This problem arises when a data is modified in one cache. This problem can be solved in two ways:

1. Invalidate all the copies on other caches (broadcast-invalidate)
2. Update all the copies on other caches (write-broadcasting), while the memory may be updated (write through) or not updated (write-back).

Note: Coherency generally applies only to data (as operands) and not to instructions (see Self-Modifying Code).

The schemes can be classified based on:

- Snoopy scheme vs Directory scheme and vs Shared caches
- Write through vs Write-back (ownership-based) protocol
- Update vs Invalidation protocol
- Intervention vs not Intervention
- Dirty-sharing vs not-dirty-sharing protocol (MOESI vs MESI)

Three approaches are adopted to maintain the coherency of data.

- **Bus watching or Snooping** – generally used for bus-based SMP – Symmetric Multiprocessor System/multi-core systems
- **Directory-based – Message-passing** – may be used in all systems but typically in NUMA system and in large multi-core systems
- **Shared cache** – generally used in multi-core systems

Snoopy coherency protocol

Protocol used in bus-based systems like a SMP systems

SMP – symmetric multiprocessor systems

> Systems operating under a single OS (Operating System) with two or more homogeneous processors and with a centralized shared Main Memory

Each processor has its own cache that acts as a bridge between processor and Main Memory. The connection is made using a System Bus or a Crossbar ("xbar") or a mix of two previously approach, bus for address and crossbar for Data (Data crossbar).[1][2][3]

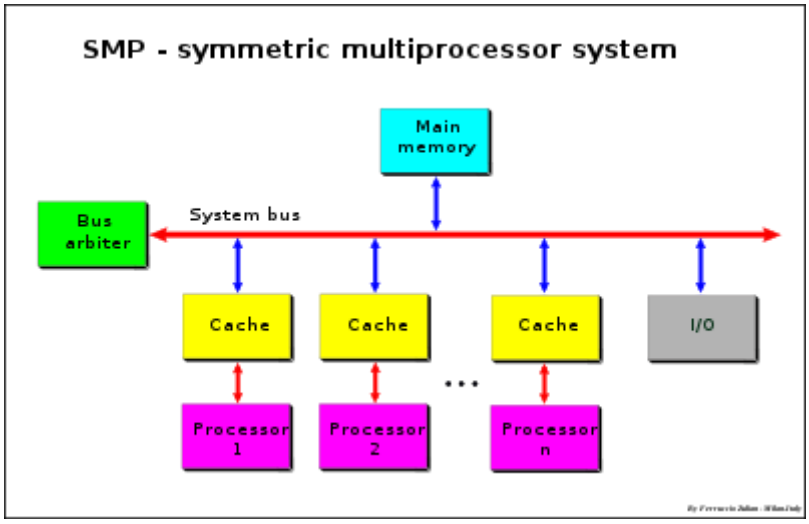
The bottleneck of these systems is the traffic and the Memory bandwidth. Bandwidth can be increasing by using large data bus path, data crossbar, memory interleaving (multi-bank parallel access) and *out of order* data transaction. The traffic can reduced by using a cache that acts as a *"filter"* versus the shared memory, that is the cache is an essential element for shared-memory in SMP systems.

In multiprocessor systems with separate caches that share a common memory, a same datum can be stored in more than one cache. A data consistency problem may occur when a data is modified in one cache only. The protocols to maintain the coherency for multiple processors are called **cache-coherency protocols**.

Usually in SMP the coherency is based on the ***"Bus watching"*** or ***"Snoopy"*** (after the Peanuts' character *Snoopy*) approach.
In a snooping system, all the caches monitor (or snoop) the bus transactions to intercept the data and determine if they have a copy on its cache.

Various cache-coherency protocols are used to maintain data coherency between caches.[4]

These protocols are generally classified based only on the cache states (from 3 to 5 and 7 or more) and the transactions between them, but this could create some confusion.



SMP – Symmetric Multiprocessor System

This definition is incomplete because it lacks important and essential information as the actions that these produce. These actions can be invoked from processor or bus (e.g. intervention, invalidation, broadcasting, updating, etc.). The type of actions are implementation depending. Protocols having the same states and the same transaction rules may be different, for instance protocol MESI with shared-intervention on unmodified data and MESI without intervention (see below). Protocols with different states may be practically the same protocol, for instance the 4-state MESI Illinois and 5-state MERSI (R-MESI) IBM / MESIF-Intel protocols are only a different implementation of the same functionality (see below).

The most common and popular protocols are considered the 4-state cache known as acronyms **MESI** and 5-state **MOESI**, this just only for easy pronunciation, terms derived from the name of the states used. Other protocols use the same states or un subset of these but with different implementations and often with different but equivalent terminology. With the term MESI or MOESI or a subset of these, generally it is referred to a class of protocols instead of a specific protocol.

Cache states

The states MESI and MOESI are often and more commonly called by different names.

- **M**=*Modified* or **D**=*Dirty* or **DE**=*Dirty-Exclusive* or **EM**=*Exclusive Modified*
  - modified in one cache only – write-back required at replacement.
  - data is stored only in one cache but the data in memory is not updated (invalid, not clean).
- **O**=*Owner* or **SD**=*Shared Dirty* or **SM**=*Shared Modified* or **T**=*Tagged*
  - modified, potentially shared, owned, write-back required at replacement.
  - data may be stored in more than a cache but the data in memory is not updated (invalid, not clean). Only one cache is the "owner", other caches are set "Valid" (S/V/SC). On bus read request, the data is supplied by the "owner" instead of the memory.
- **E**=*Exclusive* or **R**=*Reserved* or **VE**=*Valid-Exclusive* or **EC**=*Exclusive Clean* or **Me**=*Exclusive*
  - clean, in one cache only.

    - Data is stored only in one cache and *clean* in memory.
- **S**=*Shared* or **V**=*Valid* or **SC**=*Shared Clean*
    - shared or valid
    - Data potentially shared with other caches. The data can be clean or dirty. The term "clean" in SC is misleading because can be also dirty (see Dragon protocol).
- **I**=*Invalid*.
    - Cache line invalid. If the cache line is not present (no tag matching) it is considered equivalent to invalid, therefore invalid data means data present but invalid or not present in cache.

Special states:

- **F**=*Forward* or **R**=*Recent*
    - Additional states of MESI protocol
    - Last data read. It is a special "Valid" state that is the "Owner" for *non modified shared data*, used in some extended MESI protocols (MERSI or R-MESI IBM,[5][6] MESIF – Intel[7][8]). The R/F state is used to allow "intervention" when the value is clean but shared among many caches. This cache is responsible for intervention (***shared intervention***). On bus read request, the data is supplied by this cache instead of the memory. MERSI and MESIF are the same protocol with different terminology (**F** instead of **R**). Some time **R** is referred as "*shared last* " ($\mathbf{S_L}$).[9][10]
    - The state **R** = Recent is used not only in **MERSI** = **R-MESI** protocol but in several other protocols. This state can be used in combination with other states. For instance **RT-MESI**, **HR-MESI**, **HRT-MESI**, **HRT-ST-MESI**.[6][11][12] All protocols that use this state will be refereed as **R-MESI type**.
- **H**=*Hover* – **H-MESI** (additional state of MESI protocol)[11]
    - The Hover (**H**) state allows a cache line to maintain an address Tag in the directory even though the corresponding value in the cache entry is an invalid copy. If the correspondent value happens on the bus (address Tag matching) due a valid "Read" or "Write" operation, the entry is updated with a valid copy and its state is changed in **S**.
    - This state can be used in combination with other states. For instance **HR-MESI**, **HT-MESI**, **HRT-MESI**, **HRT-ST-MESI**.[6][11][12]

Various coherency protocols

	Protocols
SI protocol	Write Through
MSI protocol	Synapse protocol[4]
MEI protocol	IBM PowerPC 750,[13] MPC7400[6]
MES protocol	Firefly protocol[4]
MESI protocol	Pentium II,[14] PowerPC, Intel Harpertown (Xeon 5400)
MOSI protocol	Berkeley protocol[4]
MOESI protocol	AMD64,[15] MOESI,[16] T-MESI IBM[12]

Terminology used	
Illinois protocol	D-VE-S-I (= extended MESI)[4][17]
Write-once or Write-first	D-R-V-I (= MESI) [4][18][19]
Berkeley protocol	D-SD-V-I (= MOSI) [4]
Synapse protocol	D-V-I (= MSI) [4]
Firefly protocol	D-VE-S (= MES) DEC[4]
Dragon protocol	D-SD (SM ?)-SC-VE (= MOES) Xerox[4]
Bull HN ISI protocol	D-SD-R-V-I (= MOESI)[20]
MERSI (IBM) / MESIF (Intel) protocol	▪ R=Recent – IBM PowerPC G4, MPC7400[5][6] ▪ F=Forward – Intel[7] Intel Nehalem[21][22][23]
HRT-ST-MESI protocol	H=Hover, R=Recent,T=Tagged, ST=Shared-Tagged – IBM[11][12] – Note: The main terminologies are SD-D-R-V-I and MOESI and so they will be used both.
POWER4 IBM protocol	Mu-T-Me-M-S-S_L-I (L2 seven states)[9] ▪ **Mu**=*Unsolicited Modified – Modified Exclusive* – (**D**/**M**) (*) ▪ **T**=*Tagged – Modified Owner not Exclusive* (**SD**/**O**) ▪ **M**=*Modified Exclusive* – (**D**) ▪ **Me**=*Valid Exclusive* – (**R**/**E**) ▪ **S**=*Shared* – (**V**) ▪ **S_L**=*Shared Last – Sourced local* – (**Shared Owner Local**) ▪ **I**=*Invalid* – (**I**) (*) Special state – Asking for a reservation for load and store doubleword (for 64-bit implementations).

Snoopy coherency operations

- Bus Transactions
- Data Characteristics
- Cache Operations

Bus transactions

The main operations are:

- *Write Through*
- *Write-Back*
- *Write Allocate*
- *Write-no-Allocate*
- *Cache Intervention*

    - *Shared Intervention*
    - *Dirty Intervention*
- *Invalidation*
- *Write-broadcast*
- *Intervention-broadcasting*

Write Through

- The cache line is updated both in cache and in MM or only in MM (*write no-allocate*).
- Simple to implement, high bandwidth consuming. It is better for single write.

Write-Back

- Data is written only in cache. Data is Write-Back to MM only when the data is replaced in cache or when required by other caches (see Write policy).
- It is better for multi-write on the same cache line.
- Intermediate solution: *Write Through* for the first write, *Write-Back* for the next (Write-once and Bull HN ISI[20] protocols).

Write Allocate

- On miss the data is read from the "owner" or from MM, then the data is written in cache (updating-partial write) (see Write policy).

Write-no-Allocate

- On miss the data is written only in MM without to involve the cache, or as in Bull HN ISI protocol, in the "owner" that is in **D** or **SD** cache (owner updating), if they are, else in MM.
- Write-no-Allocate usually is associated with Write Through.

- **Cache Intervention**

  (or shortly "*intervention* ")

  – **Shared Intervention** – shared-clean intervention (on unmodified data)

  – On *Read Miss* the data is supplied by the owner **E** or **R**/**F** or also **S** instead of the MM (see protocols Illinois , IBM R-MESI type and Intel MESIF).

  – **Dirty Intervention** (on modified data)

  – On *Read Miss* the data is supplied by the **M** (D) or **O** (SD) owner or by **E** (R) (*) instead of MM (e.g. MOESI protocol, RT-MESI, …).

  (*) – Not for **E** (R) in the original proposal MOESI protocol[16] and in some other implementations MOESI-Type.

  – "*Intervention* " is better compared to the "*not intervention* " because ***cache-to-cache*** transactions are much more faster than a MM access, and in addition it save memory bandwidth (memory traffic reduction). Extended MESI Illinois and R-MESI type / MESIF are therefore much better than the MOESI protocol (see MESI vs MOESI below)

- **Invalidation**

– On *Write Hit* with **S** (V) or **O** (SD) (shared) state, a bus transaction is sent to invalidate all the copies on the other caches (*Write-invalidate*).

- **Write-broadcast** (Write-update)

  – On *Write Hit* with **S** (V) or **O** (SD) (shared) state, a write is forward to other caches to update their copies (e.g. Intel Nehalem[22] Dragon protocol, Firefly (DEC).

  – Note – The updating operation on the other caches some time is called also ***Snarfing***. The caches snoop the bus and if there is a hit in a cache, this cache *snarfs* the data that transits on the bus and update its cache. Also the updating of the **H** in (H-MESI) state can be defined as *snarfing*. In the first case this happens in a write broadcast operation, on the second case both in read and write operations.

- **Intervention-broadcasting**

  – On an Intervention transaction, a cache with **H** state (H-MESI) updates its invalid copy with the value sent on the bus and its state is changed in **S**.[6]

- **Write invalidate vs broadcast**

  – Write Invalidate is better when multiple writes, typically partial write, are done by a processor before that the cache line is read by another processor.
  – Write-broadcast (updating) is better when there is a single producer and many consumers of data, but it is worse when a cache is filled with data that will be not next read again (bus traffic increasing, cache interference increasing).

  - Invalidation is the common solution.

Data characteristics

There are three characteristics of cached data:

- *Validity*
- *Exclusiveness*
- *Ownership*

- **Validity**

  – Any not invalid cache line, that is MOES / D-SD-R-V.

- **Exclusiveness**

  – Data valid only in one cache (data not shared) in **M** (D) or **E** (R) state, with MM not clean in case of **M** (D) and clean in case of **E** (R).

- **Ownership**

  – The cache that is responsible to supply the request data instead of a MM (Intervention) – Depending on the protocol, cache who must make the intervention can be **S**-**E**-**M** in MESI Illinois, or **R**/**F**-**E**-**M** in R-MESI type / MESIF or **M** (D) or **O** (SD) or also **E** (R) (*) in MOESI-type protocols, (e.g. AMD64,[16] Bull HN ISI[20] – see "Read Miss" operation below).

(*) – Implementation depending.

Note: Not to confuse the more restrictive "owner" definition in MOESI protocol with this more general definition.

Cache operations

The cache operations are:

- *Read Hit*
- *Read Miss*
- *Write Hit*
- *Write Miss*

- **Read Hit**

  – Data is read from cache. The state is unchanged

    – **Warning**: since this is an obvious operation, afterwards it will not be more considered, also in state transaction diagrams.

- **Read Miss**

  – The data read request is sent to the bus
  – There are several situations:

    - **Data stored only in MM**

      – The data is read from MM.
      – The cache is set **E** (R) or **S** (V)
      – **E** (R) if a special bus line ("*Shared line* ") is used to detect "*no data sharing* ". Used in all protocols having **E** (R) state except for Write-Once and Bull HN ISI protocols (see "Write Hit" below).

    - **Data stored in MM and in one or more caches in S** (V) **state or in R/F in R-MESI type / MESIF protocols**.

      – There are three situations:

    1. – Illinois protocol – a network priority is used to temporary and arbitrary assign the ownership to a **S** copy.
       - Data is supplied by the selected cache. Requesting cache is set **S** (**shared intervention** with MM clean).
    2. – R-MESI type / MESIF protocols – a copy is in **R**/**F** state (**shared owner**)
       – The data is supplied by the **R**/**F** cache. Sending cache is changed in **S** and the requesting cache is set **R**/**F** (in read miss the "ownership" is always taken by the last requesting cache) – **shared intervention**.
    3. – In all the other cases the data is supplied by the memory and the requesting cache is set **S** (V).

    - **Data stored in MM and only in one cache in E** (R) **state**.

1. – Data is supplied by a **E** (R) cache or by the MM, depending on the protocol.
   – From **E** (R) in extended MESI (e.g. Illinois, Pentium (R) II [14]), R-MESI type / MESIF and from same MOESI implementation (e.g. AMD64)
   – The requesting cache is set **S** (V), or **R**/**F** in R-MESI type / MESIF protocols and **E** (R) cache is changed in **S** (V) or in **I** in MEI protocol.
2. – In all the other cases the data is supplied by the MM.

- **Data modified in one or more caches with MM not clean**

  - **Protocol MOESI type – Data stored in M** (D) **or in O** (SD) **and the other caches in S** (V)

    – Data is sent to the requesting cache from the "owner" **M** (D) or **O** (SD). The requesting cache is set **S** (V) while **M** (D) is changed in **O** (SD).
    – The MM is not updated.

  - **Protocols MESI type and MEI – Data stored in M** (D) **and the other caches in S** (V) state

    – There are two solutions:

1. – Data is sent from the **M** (D) cache to the requesting cache and also to MM (e.g. Illinois, Pentium (R) II [14])
2. – The operation is made in two steps: the requesting transaction is stopped, the data is sent from the **M** (D) cache to MM then the wait transaction can proceed and the data is read from MM (e.g. MESI and MSI Synapse protocol).

   – All cache are set **S** (V)

- **Write Hit**

– The data is written in cache
– There are several situations:

  - **Cache in S** (V) **or R/F or O** (SD) (sharing)

    – **Write invalidate**

      – Copy back

        – The data is written in cache and an invalidating transaction is sent to the bus to invalidate all the other caches
        – The cache is set **M** (D)

      – Write Through (Write-Once, Bull HN ISI)

        – Data is written in cache and in MM invalidating all the other caches. The cache is set **R** (E)

    – **Write broadcasting** (e.g. Firefly, Dragon)

      - The data is written in cache and a broadcasting transaction is sent to the bus to update all the other caches having a copy

– The cache is set **M** (D) if the "shared line" is off, otherwise is set **O** (SD). All the other copies are set **S** (V)

- **Cache in E** (R) **or M** (D) **state** (exclusiveness)

  – The write can take place locally without any other action. The state is set (or remains) **M** (D)

- **Write Miss**

  – **Write Allocate**

  – Read with Intent to Modified operation (RWITM)

  – Like a Read miss operation plus an invalidate command, then the cache is written (updated)
  – The requesting cache is set **M** (D), all the other caches are invalidated

  – Write broadcasting (e.g. Firefly, Dragon)

  – Like with a Read Miss. If "shared line" is "off" the data is written in cache and set **M** (D), otherwise like with a Write Hit – Write Broadcasting

  – **Write-no-Allocate**

  – The data is sent to MM, or like in Bull HN ISI protocol, only to the **D** (M) or **SD** (O) cache if they are, bypassing the cache.

Coherency protocols

– **warning** – For simplicity all Read and Write "miss" state transactions that obviously came from **I** state (or Tag miss), in the diagrams are not depicted. They are depicted directly on the new state.

– Note – Many of the following protocols have only historical value. At the present the main protocols used are R-MESI type / MESIF and HRT-ST-MES (MOESI type) or a subset of this.

MESI protocol

States MESI = D-R-V-I

– Use of a bus "shared line" to detect "shared" copy in the other caches

- **Processor operations**

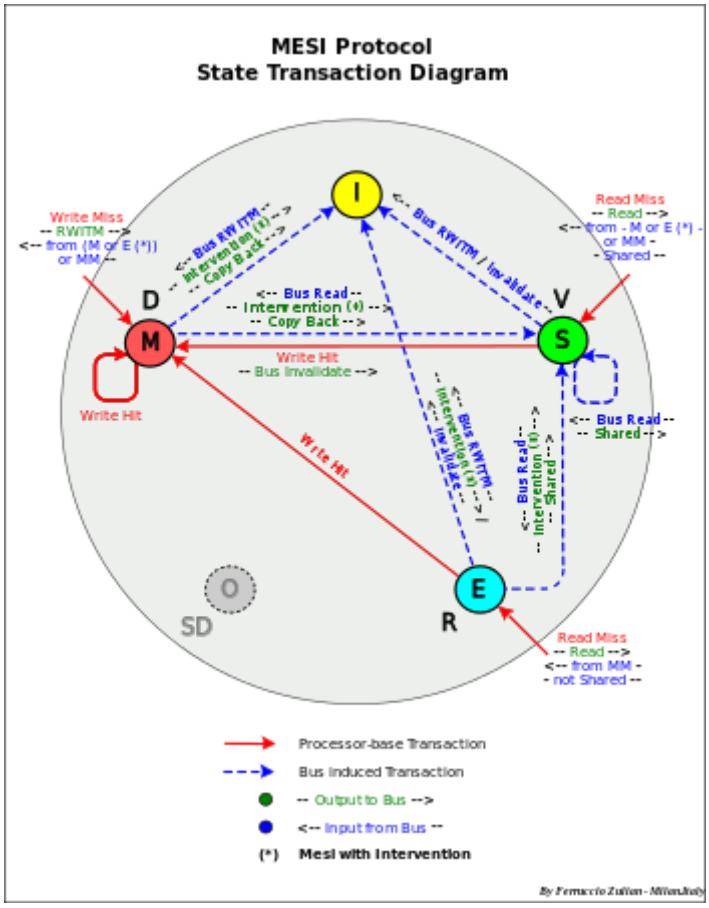
  - Read Miss

    There are two alternative implementations: standard MESI (not intervention) and extended MESI (with intervention)

    1 – MESI "no Intervention" (e.g. PowerPC 604 [24])

– If there is a **M** copy in a cache, the transaction is stopped and wait until the **M** cache updates the MM, then the transaction can continue and the data is read from MM. Both caches are set **S**

– else the data is read from MM. If the "shared line" is "on" the cache is set **S** else **E**



MESI Protocol – State Transaction Diagram

2 – MESI "Intervention" from **M** and **E** (e.g. Pentium (R) II [14])

– If there is a **M** or **E** copy (exclusiveness) in a cache, the data is supplied to the requesting cache from **M** or from **E** (Intervention). If the sending cache is **M** the data is also written at the same time in MM (copy back). All caches are set **S**

– else the data is read from MM. If the "shared line" is "on" the cache is set **S** else **E**

- Write Hit

  – If the cache is **M** or **E** (exclusiveness), the write can take place locally without any other action
  – else the data is written in cache and an invalidating transaction is sent to the bus to invalidate all the other caches
  – The cache is set **M**

- Write Miss

  – RWITM operation is sent to the bus. The operation is made in two step: a "Read Miss" with "invalidate" command to invalidate all the other caches, then like with a "Write Hit" with the state **M** (see Cache operation-Write Miss).

- **Bus transactions**

  - Bus Read

    – if **M** and "no Intervention" the data is sent to MM (Copy Back)

– if **M** and "Intervention" the data is sent to requesting cache and to MM (Copy Back)
– if **E** (*) and "Intervention" the data sent to requesting cache
– The state is changed (or remains) in **S**

- Bus Read – (RWITM)

  – As like with "Bus read"
  – The cache is set "Invalid" (**I**)

- Bus Invalidate Transaction

  The cache is set "Invalid" (**I**)

- **Operations**

  – *Write Allocate*
  – *Intervention*: from M – E (*)
  – *Write Invalidate*
  – *Copy-Back*: M replacement

  (*) – extended MESI

MOESI protocol

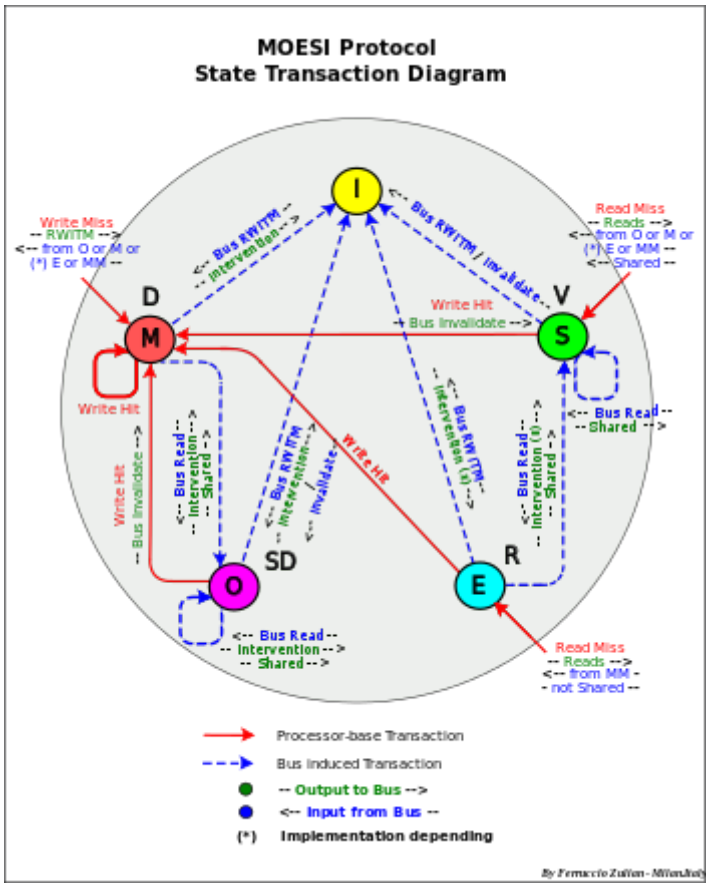
States MEOSI = D-R-SD-V-I = T-MESI IBM[12]

– Use of bus "shared line" to detect "shared" copy on the other caches

- **Processor operations**

  - Read Miss

    – If there is a **M** or **O** or **E** (*) copy in another cache the data is supplied by this cache (intervention). The requesting cache is set **S** , **M** is changed in **O** and **E** in **S**
    – else the data is read from MM.
    – If "shared line" is "on" the requesting cache is set **S** else **E**



MOESI Protocol – State Transaction Diagram

  - Write Hit

    – If the cache is **M** or **E** (exclusiveness), the write can take place locally without any other action

– else **O** or **S** (sharing) an "Invalidation" transaction is sent on the bus to invalidate all the other caches.
– The cache is set (or remains) **M**

- Write Miss

– A RWITM operation is sent to the bus
– Data is supplied from the "owner" or from MM as with Read Miss, then cache is written (updated)
– The cache is set **M** and all the other caches are set **I**

- **Bus transactions**

- Bus Read

– If the cache is **M** or **O** or **E** (*) the data is sent to requesting cache (intervention). If the cache is **E** the state is changed in **S**, else is set (or remains) **O**
– else the state is changed or remains in **S**

- Bus Read – (RWITM)

– If the cache is **M** or **O** or **E** (*) the data is sent to the bus (Intervention)
– The cache is set "Invalid" (**I**)

- Bus Invalidate Transaction

– The cache is set "Invalid" (**I**)

- **Operations**
- *Write Allocate*
- *Intervention*: from M-O-E (*)
- *Write Invalidate*
- *Copy-Back*: M-O replacement

– (*) implementation depending for **E**

Illinois protocol

States MESI = D-R-V-I[4]

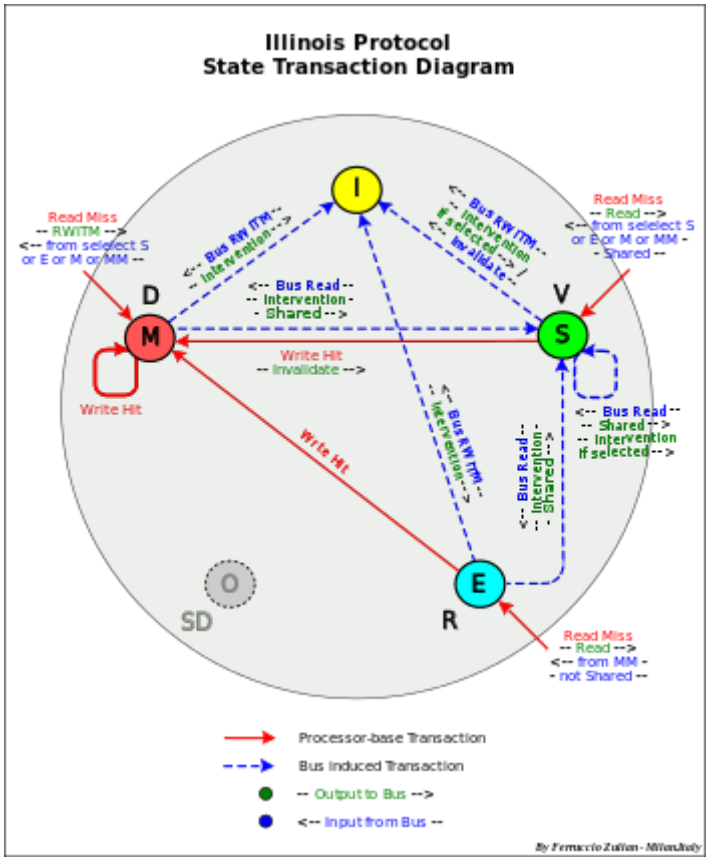
– Characteristics:

– It is an extension of MESI protocol
– Use of a network priority for **shared intervention** (intervention on shared data)
– Differences from MESI: in addition to **E** and **M**, intervention also from **S** (see Read Miss – point 1)

- **Operations**

- *Write Allocate*

- *Intervention*: from M-E-S
- *Write Invalidate*
- *Copy-Back*: M replacement



Illinois State Transaction Diagram

Write-once (or write-first) protocol

States D-R-V-I (MESI) [4][18][19]

– Characteristics:

– No use of "shared line" (protocol for standard or unmodifiable bus)
– Write Through on first Write Hit in state **V**, then Copy Back

- **Processor operations**
  - Read Miss

    – If there is a **D** copy in another cache, the data is supplied by this cache (intervention) and in the same time it is written also in MM (Copy-Back).
    – else the data is read from MM
    – all caches are set **V**

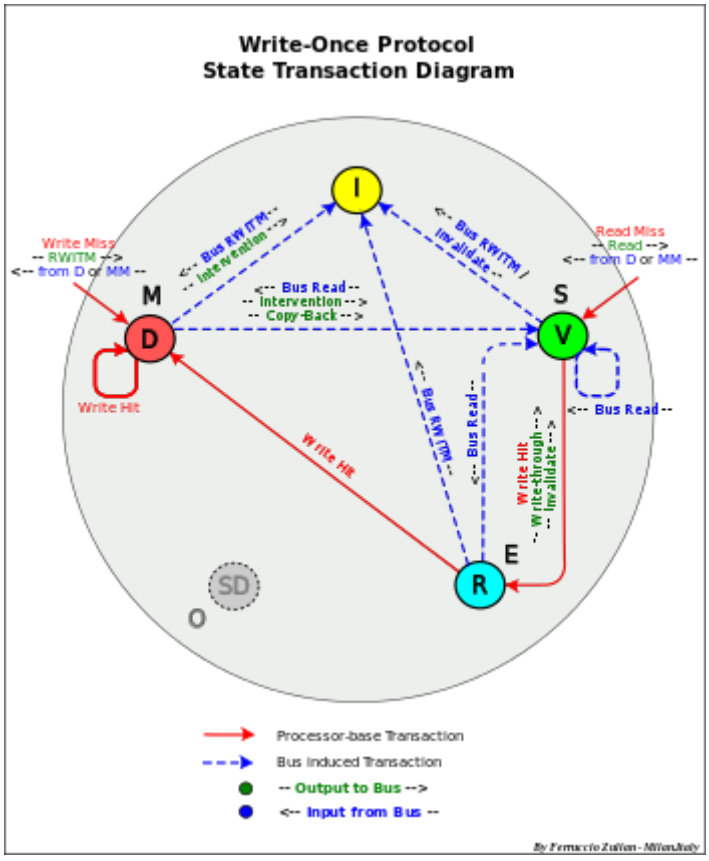
  - Write Hit

    – If the cache is **D** or **R** (exclusiveness), the write can take place locally without any other action and the state is set (or remains) **D**
    – else **V** (first Write Hit) the data is written in cache and in MM (Write Through) invalidating all the other caches (Write-Invalidate). – The cache is set **R**

- Write Miss

  – Like a Read Miss but with "invalidate" command (RWITM) plus a Write Hit in **D** state (updating). The cache is set **D** and all the other caches are set "Invalid" (**I**)

  – Note – Write Through is performed only in "Write Miss". It is point out that in this case a bus transaction in any case is needed to invalidate the other caches and therefore it can be taken advantage of this fact to update also the MM. In "Write Hit" instead no more transaction is needed so a "Write Through" it would become a useless operation in case that the cache were updated again.



Write-Once Protocol – State Transaction Diagram

- **Bus transactions**

  - Bus Read

    – If the cache is **D** the data is sent to requesting cache (intervention) and to MM (copy-back). The cache is set **V**
    – else the state is changed or remains in **V**

  - Bus Read – (RWITM)

    – If the cache is **D** the data is sent to the bus (Intervention)
    – The cache is set "Invalid" (**I**)

  - Bus Invalidate Transaction

    – The cache is set "Invalid" (**I**)

- **Operations**

- *Write Allocate*
- *Intervention*: from D
- *Write Through*: first write hit in **V** state
- *Write Invalidate*
- *Copy-Back*: D replacement

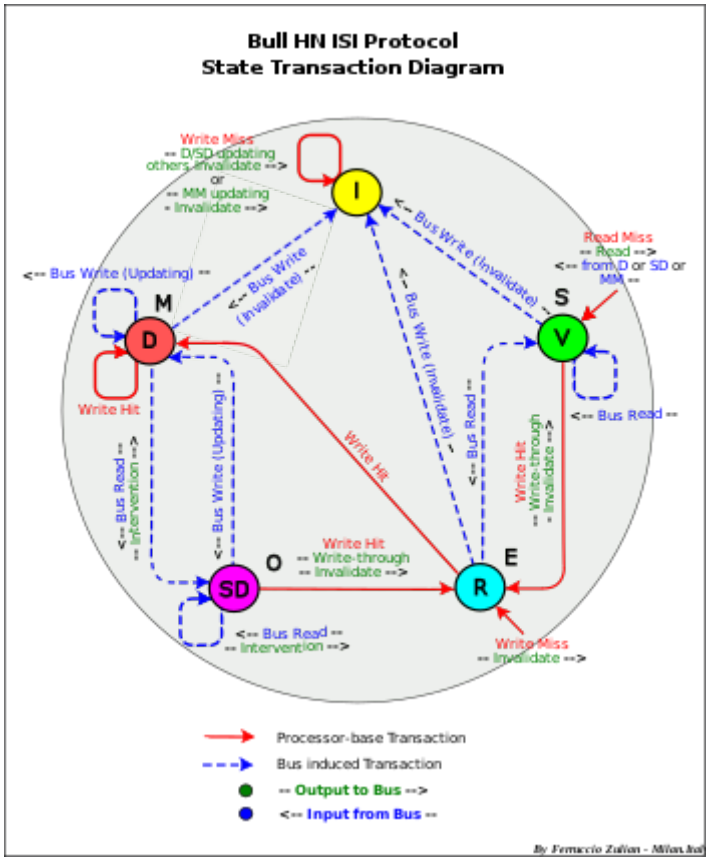
Bull HN ISI protocol

(Bull-Honeywell Italia)

States D-SD-R-V-I (MOESI)
Patented protocol (F. Zulian)[20]

– Characteristics:

– MOESI extension of the Write-Once protocol
- Write-no-allocate on miss with **D** or **SD** updating
- No use of RWITM
- No use of "shared line"



Bull HN ISI Protocol – State Transaction Diagram

- **Processor operations**
  - Read Miss

    - Like with MOESI with "Shared Line" "on" and intervention only from the "owner" **D** or **SD** but not from **R**

  - Write Hit

    - If the cache is **D** or **R**, like with MOESI, the write can take place locally without any other action. The cache is set (or remains) **D**
    - If **SD** or **V** (first write), like with Write-Once, the data is written in cache and in MM (Write Through) invalidating all the other caches (Write-Invalidate) – The cache is set **R**

  - Write Miss

    - The data is sent to the bus bypassing the cache (Write-no-allocate)
    - If there is an "owner" copy **D** or **SD**, the "owner" is updated (see Write-no-Allocate – owner updating) while the other caches are invalidated. The "owner" is set (or remains) **D**. The memory remains "dirty"
    - else the data is sent to MM invalidating all the other caches (Write-Invalidate)

- **Bus transactions**
  - Bus Read

    - Like with MOESI with intervention only from "owner" **D** or **SD**

  - Bus Read (Write Update / Write Invalidate)

    - If the cache is **D** or **SD**, the cache is updated, else is set "Invalid" (**I**)

- **Operations**
- *Write-no-allocate*: on miss
- *Write update*: on miss
- *Write Through*: for the first write, then copy back
- *Write Update / Write Invalidate*
- *Intervention*: from SD-D
- *Copy-Back*: D replacement or SD replacement with invalidate

Obs. - This is the only protocol that has O-E (SD-R) transactions and it is also the only one that makes use of the Write-no-allocated on miss.

Synapse protocol

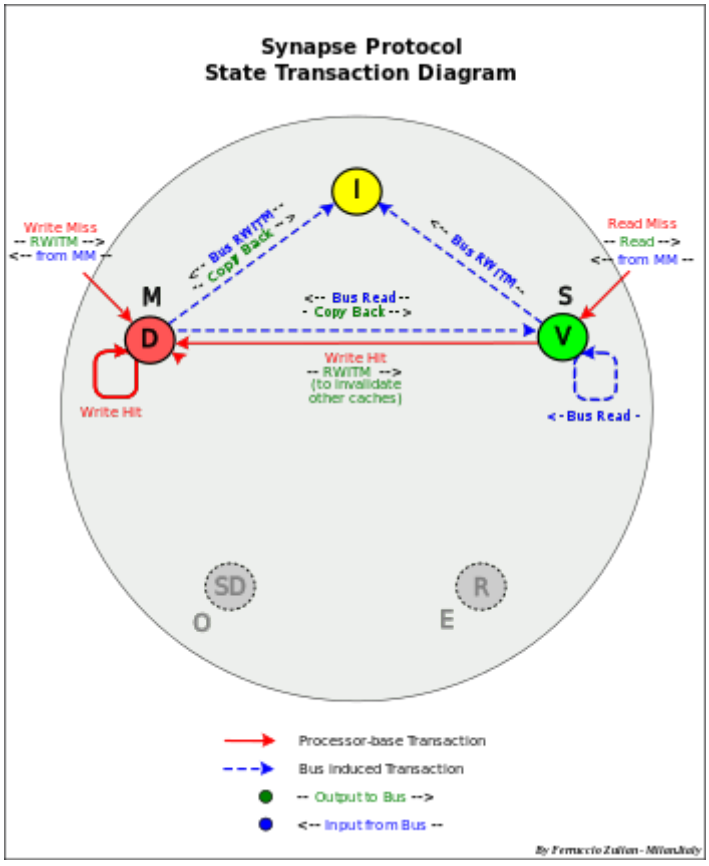
States D-V-I (MSI)[4]

- Characteristics:

- The characteristic of this protocol is ti have a single-bit tag with each cache line in MM, indicating that a cache have the line in **D** state.

- This bit prevents a possible race condition if the **D** cache does not respond quickly enough to inhibit the MM from responding before being updating.

- The data comes always from MM
- No use of "shared line"



Synapse Protocol – State Transaction Diagram

- **Processor operations**

  - Read Miss

    - If there is a **D** copy in another cache, the read transaction is rejected (no acknowledgement). The **D** copy is written back to MM and changes its state in **V**, then the requesting cache resends a new read transaction and the data is read from MM.
    - else the data is read from MM.
    - The cache is set **V**

  - Write Hit

    - If the cache is **D** , the write can take place locally without any other action.

- else **V**, like with Read Miss does, including a data transfer from memory with in addition an invalidate command (RWITM). This is done only to invalidate the other **V** copies because this protocol does not support an invalidation transaction.
- The cache is set **D**. All the other caches copy are set "Invalid" (**I**)

- Write Miss (RWITM)

  - Like with Read Miss, but with invalidate command. The cache line comes from MM, then the cache is written (updated). The cache is set **D**. All the other caches are set "Invalid" (**I**).

- **Bus transactions**

  - Bus Read

    - If the cache is **D**, the data is sent to MM (Copy Back). The cache is set **V**
    - else the state remains in **V**

  - Bus Read (RWITM)

    - If the cache is **D** the data is sent to MM (Copy Back)
    - The cache (**D** or **V**) is set "Invalid" (**I**)

- **Operations**
- *Write Allocate*
- *Intervention*: no intervention
- *Write Invalidate*: (RWITM)
- *No Invalidate transaction*
- *Copy-Back*: D replacement

Berkeley protocol

States D-SD-V-I (MOSI)[4]

- Characteristics:
- As with MOESI without **E** state
- No use of "shared line"

- **Processor operations**

  - Read Miss

    - The data is supplied by the "owner", that is from **D** or from **SD** else from MM. **D** is changed in **SD**
    - The cache is set **V**

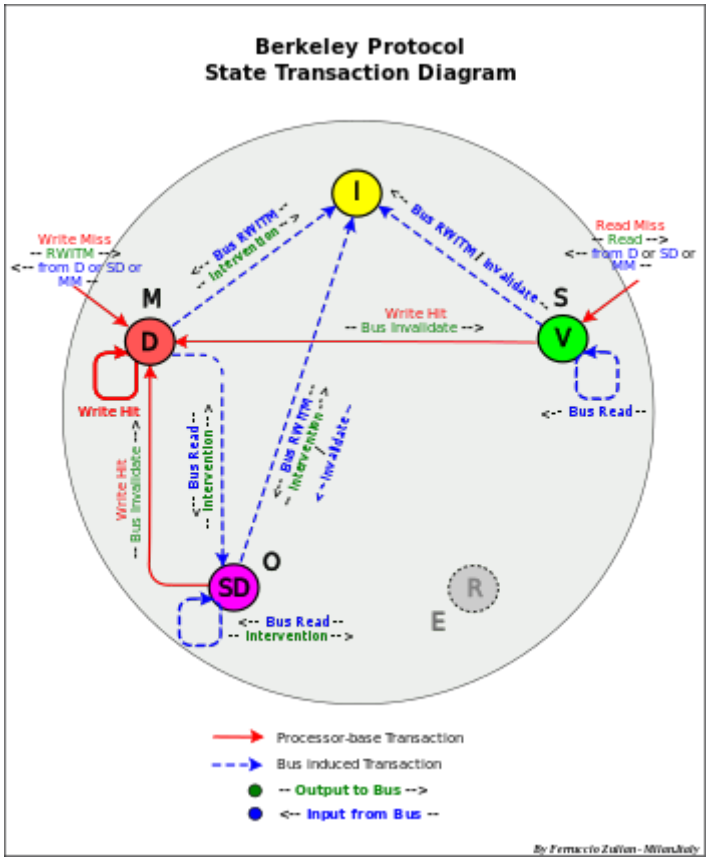
  - Write Hit

    - If the cache is **D** (exclusiveness), the write can take place locally without any other action

- else (**SD** or **V**), an "Invalidation" transaction is sent on the bus to invalidate the other caches.
- The cache is set (or remains) **D**

- Write Miss

- RWITM operation is sent to the bus
- Like with Read Miss, the data comes from the "owner", **D** or **SD** or from MM, then the cache is updated
- The cache is set **D**. all the other caches are set **I**



Berkeley Protocol – State Transaction Diagram

- **Bus transactions**

  - Bus Read

    - If the cache is **D** or **SD** the data is sent to requesting cache (intervention). The cache is set (or remains) in **SD**
    - else the cache remains in **V**

  - Bus Read – (RWITM)

    - If the cache is **D** or **SD** the data is sent to the bus (Intervention)
    - The cache is set "Invalid" (**I**)

  - Bus Invalidate Transaction

    - The cache is set "Invalid" (**I**)

- **Operations**

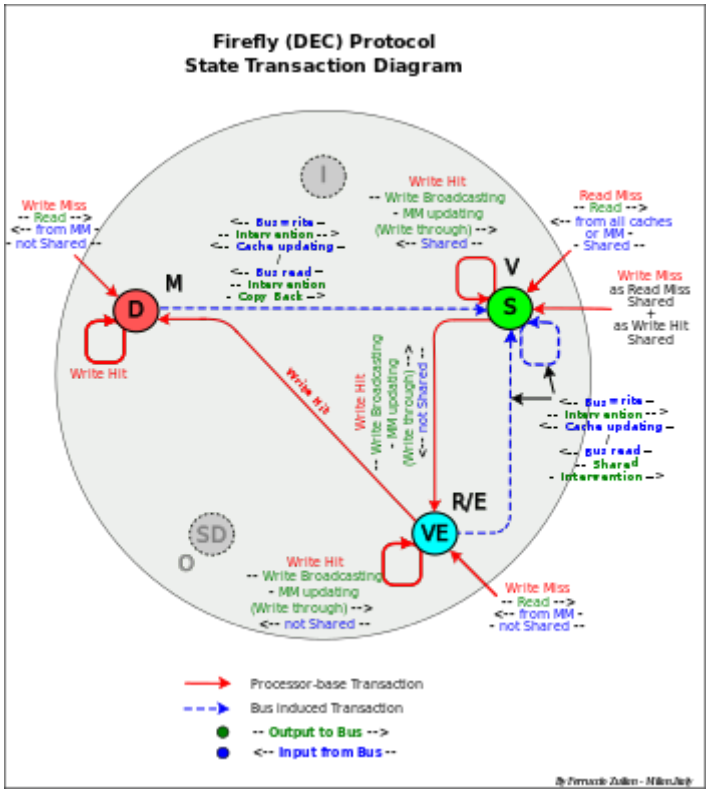
- *Write Allocate*
- *Intervention*: from D-SD
- *Write Invalidate*
- *Copy-Back*: D-SD replacement

Firefly (DEC) protocol

States D-VE-S (MES)[4]

- Characteristics:

- No "Invalid" state
- "Write-broadcasting"+"Write Through"
- Use of "shared line"
- "Write-broadcasting" avoid the necessity of "Invalid" state
- Simultaneous intervention from all caches (shared and dirty intervention – on not modified that modified data)
- This protocol requires a synchronous bus



Firefly Protocol – State Transaction Diagram

- **Processor operations**
  - Read Miss

    - Any other cache is the "owner", that is all the other caches with a copy supplied simultaneously the date on the bus (simultaneous intervention – the bus timing is fixed so that they all respond in the same cycle), otherwise the data is supplied from MM.
    - If there is a cache **D**, the data is sent simultaneously also to MM (Copy Back)
    - If there are copies in the other caches, the "Shared line" is set "on"
    - If "Shared line" is "on" all the caches are set **S** else the requesting cache is set **VE**.

  - Write Hit

    - If the cache is **D** or **VE** (exclusiveness), the write can take place locally without any other action and the cache is set **D**
    - else **S**, a "Write-broadcasting" is sent to the bus to update all the other caches and the MM (Write Through)

      - If there is a copy in another cache, the "Shared line" is set "on". If "Shared line" is set "off" the cache is set **VE** else all caches are set **S**

  - Write Miss

    - The operation is made in two steps. Read Miss then Write Hit.
    - If the data comes from a cache (Shared Line "on") a "Write-broadcasting" is sent to the bus to update all the other caches and the MM (Write Through). All the caches are set **S**
    - else the cache is set **D**

- **Bus transactions**
  - Bus Read

    - If hit (**D** or **VE** or **S**) the data is sent to the bus (intervention) and in case of **D** the data is written also in MM. The cache is set **S**

- Bus Read

  - If hit (**D** or **VE** or **S**) the data is sent to the bus (Intervention).

  - All the caches are set **S**

- Write Broadcasting

  - The cache is updated with new data. The state remains **S**

- **Operations**
- *Write Allocate*
- *Intervention*: from D-VE-S (from all "valid" caches)
- *Write-broadcasting – Write through*
- *Copy-Back*: D replacement and on any transaction with a cache D

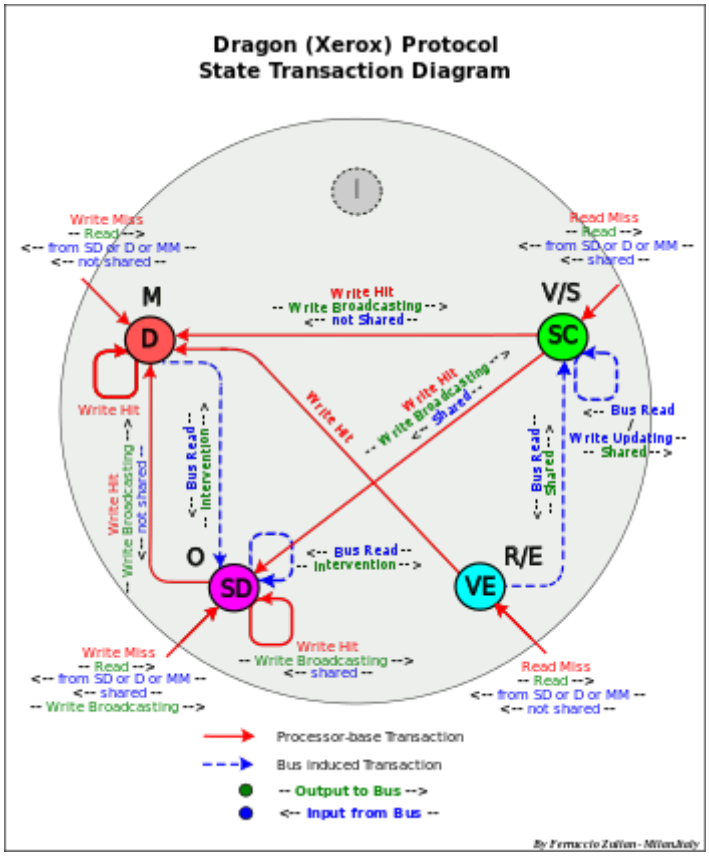
Dragon (Xerox) protocol

States D-SD-VE-SC (MOES)[4]

Note – the state **SC**, despite of the term "clean", can be "clean" or "dirty" as the **S** state of the other protocols. **SC** and **S** are equivalents

- Characteristics:

  - No "Invalid" state
  - "Write-broadcasting" (no "Write Through")
  - Use of "shared line"
  - "Write-broadcasting" avoid the necessity of "Invalid" state



Dragon Protocol – State Transaction Diagram

- **Processor operations**

  - Read Miss

    - The data is supplied by the "owner", that is from **D** or from **SD** else from MM. **D** is changed in **SD**
    - If "shared line" is "on" the cache is set **SC** else **VE**

  - Write Hit

    - If the cache is **D** or **VE** (exclusiveness), the write can take place locally without any other action. The cache is set (or remains) **D**

      - else **SD** or **SC** (sharing) the data is written in cache and a "Write-broadcasting" is sent to the bus to update all the other caches – The MM

is not updated (no Write through)
- If there is a copy in another cache, the "Shared line" is set "on"
- If the "Shared Line" is "on" the cache is set **SD**, else **D**. All the other caches possible copy are set **SC**

- Write Miss

- Like with Read Miss, the data comes from the "owner", **D** or **SD** or from MM, then the cache is updated
- If there is a copy in another cache, the "Shared line" is set "on".
- If the "Shared Line" is "on" the updated data is broadcast to the other caches and the state is set **SD**. All the other caches are set **SC**
- else the cache is **D**

- **Bus transactions**

- Bus Read

- If the cache is **D** or **SD** the data is sent to requesting cache (intervention). The cache is set (or remains) **SD**
- else the cache remains **SC**

- Bus Read

- If the cache is **D** or **SD** the data is sent to the bus (Intervention)

- The cache is set **SC**

- Write Broadcasting

- The cache is updated with new data. The cache remains **SC**

- **Operations**
- *Write Allocate*
- *Intervention*: from D-SD (but not from VE)
- *Write-broadcasting*
- *Copy-Back*: D-SD replacement

MERSI (IBM) / MESIF (Intel) protocol

States MERSI or R-MESI
States MESIF
Patented protocols – IBM (1997)[6] – Intel (2002)[8]

- MERSI and MESIF are the same identical protocol (only the name state is different ,**F** instead of **R**)

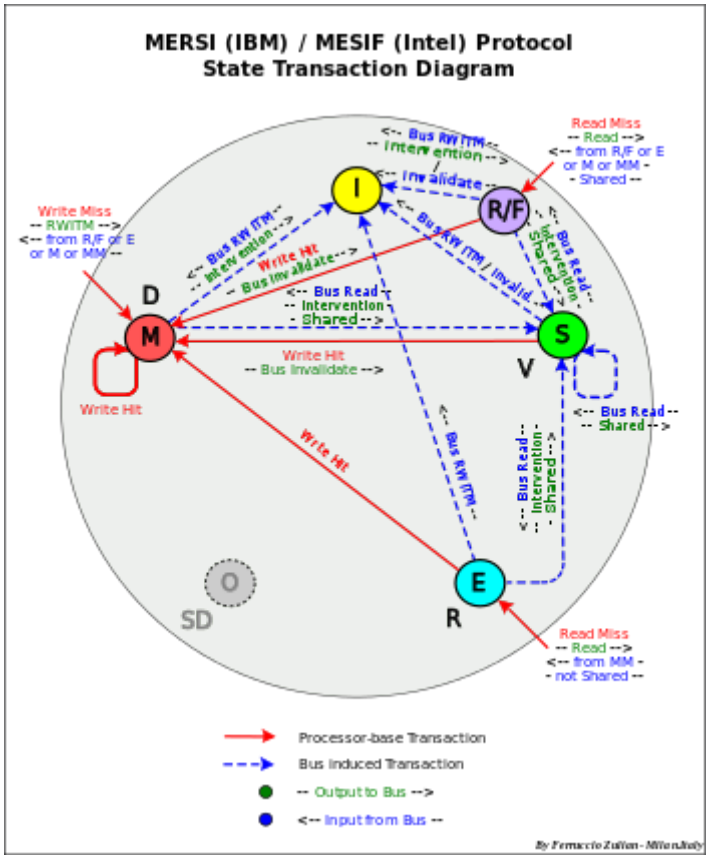
- Characteristics:

- The same functionality of Illinois protocol

- A new state **R** (Recent) / **F** (Forward) is the "*owner* " for "shared-clean" data (with MM updated).
- The "shared ownership" (on clean data) is not assigned by a network priority like with Illinois, but it is always assigned to the last cache with Read Miss, setting its state **R/F**
- The "ownership" is temporary loosed in case of **R/F** replacement. The "ownership" is reassigned again to the next Read Miss with caches "shared clean"

- Use of the "shared line"

- **Operations**
- *Write Allocate*
- *Intervention*: from M-E-R/F
- *Write Invalidate*
- *Copy-Back*: M replacement



MERSI – MESIF Protocol – State Transaction Diagram

MESI vs MOESI

MESI and MOESI are the most popular protocols

It is common opinion that MOESI is an extension of MESI protocol and therefore it is more sophisticate and more performant. This is true only if compared with standard MESI, that is MESI with "not sharing intervention". MESI with "sharing intervention", as ***MESI Illinois like or the equivalent 5-state protocols MERSI / MESIF , are much more performant than the MOESI protocol***.

In MOESI, cache-to-cache operations is made only on modified data. Instead in MESI Illinois type and MERSI / MESIF protocols, the cache-to-cache operations are always performed both with clean that with modified data. In case of modified data, the intervention is made by the "owner" M, but the ownership is not loosed because it is migrated in another cache (R/F cache in MERSI / MESIF or a selected cache as Illinois type). The only difference is that the MM must be updated. But also in MOESI this transaction should be done later in case of replacement, if no other modification occurs meanwhile. However this it is a smaller limit compared to the memory transactions due to the not-intervention, as in case of clean data for MOESI protocol. (see e.g. "Performance evaluation between MOESI (Shanghai) and MESIF Nehalem-EP"[21])

The most advance systems use only R-MESI / MESIF protocol or the more complete RT-MESI, HRT-ST-MESI and POWER4 IBM protocols that are an enhanced merging of MESI and MOESI protocols

Note: Cache-to-cache is an efficient approach in multiprocessor/multicore systems direct connected between them, but less in Remote cache as in NUMA systems where a standard MESI is preferable. Example in POWER4 IBM protocol "shared intervention" is made only "local" and not between remote module.

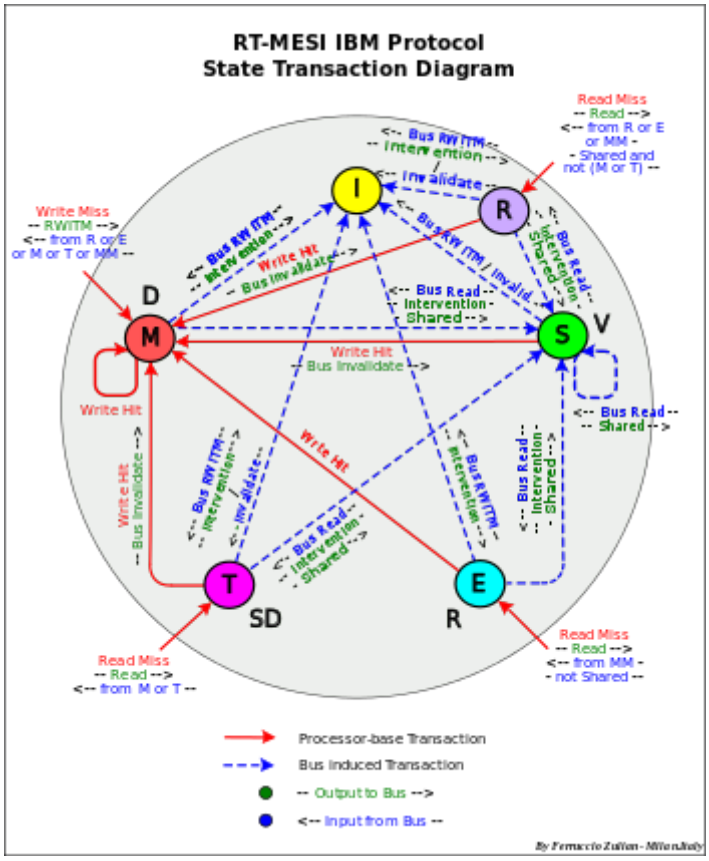
RT-MESI protocol

States RT-MESI
IBM patented protocol[11][12]

- Characteristics:

  - MESI and MOESI merging
  - Shared Intervention + Dirty Intervention (both on clean and dirty data)
  - Same functionality of R-MESI protocol with a new state **T**=Tagged, equivalent to **O** state
  - "Dirty-Owner" migration

  - The "owner" (both Shared or Dirty) is always the last requesting cache (the new "owner" (LRU) has less probability to be deallocated soon compared to the old one)
  - The "owners" are **T**, **M**, **E**, **R** (all except **S**)
  - Use of the "shared line"



RT-MESI Protocol – State Transaction Diagram

Processor operations

- Read Miss

  - If there is a **M** or **T** (dirty-ownership) copy in another cache, the data is supplied by this cache (dirty intervention). The requesting cache is set **T** and the previous **M** or **T** are changed in **S**
  - If there is a **E** or **R** (shared-ownership) copy in another cache, the data is supplied by this cache (shared intervention). The requesting data is set **R** and **E** or **R** are changed in **S**
  - else the data is read from MM and the cache is set **R**.

- Write Hit

  - If the cache is **M** or **E** (exclusiveness), the write can take place locally without any other action
  - else **T** or **R** or **S** (sharing) an "Invalidation" transaction is sent on the bus to invalidate all the other caches.
  - The cache is set (or remains) **M** and all the other caches are set **I**

- Write Miss

- RWITM operation is sent to the bus
- Data is supplied from the "owner" or from the MM as with Read Miss, then the data is written (updated) in cache.
- The cache is set **M** and all the other caches are set **I**

- **Bus transactions**
    - Bus Read

      - If the cache is **T** or **M** or **R** or **E** the data is sent to requesting cache (intervention).
      - The cache is set (or remains) in **S**

    - Bus Read – (RWITM)

      - If the cache is **T** or **M** or **R** or **E** the data is sent to requesting cache (intervention)
      - The cache is set "Invalid" (**I**)

    - Bus Invalidate Transaction

      - The cache is set "Invalid" (**I**)

- **Operations**
- *Write Allocate*
- *Intervention*: from T-M-R-E
- *Write Invalidate*
- *Copy-Back*: T-M replacement

RT-ST-MESI protocol

It is an improvement of RT-MESI protocol[12] and it is a subset of HRT-ST-MESI protocol[11]

S_T = Shared-Tagged

- Use of the "Shared-Tagged" state allows to maintain intervention after deallocation of a Tagged cache line

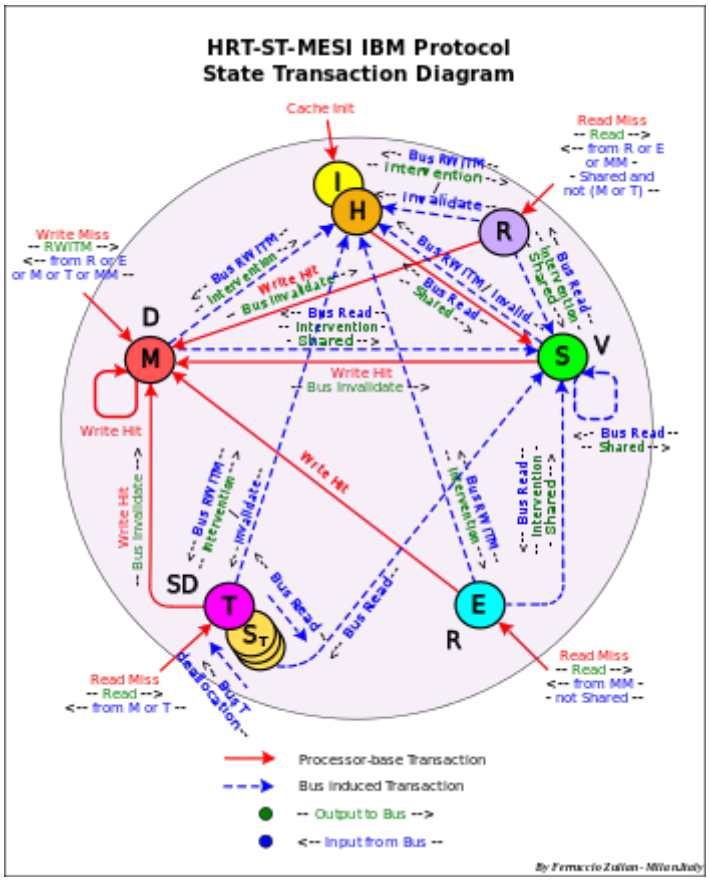
  - In case of **T** replacement (cache line deallocation), the data needs to be written back to MM and so to lose the "ownership". To avoid this, a new state **S_T** can be used. In Read Miss the previous **T** is set **S_T** instead of **S**. **S_T** is the candidate to replace the ownership in case of **T** deallocation. The **T** "Copy back" transaction is stopped (no MM updating) by the **S_T** cache that changes its state in **T**. In case of a new read from another cache, this last is set **T**, the previous **T** is changed in **S_T** and the previous **S_T** is changed in **S**.

An additional improvement can be obtained using more than a S_T state, S_{T1}, S_{T2}, … S_{Tn}.

- In Read Miss, **T** is changed in **S_{T1}** and all the indices of the others **S_{Ti}** are increased by "1.

- In case of **T** deallocation, S_{T1} stops the "Copy Back" transaction, changes its state in **T** and all the indices of the others S_{Ti} are decrease by "1".
- In case of a deallocation, for instance S_{Tk}, the chain will be interrupted and all the S_{Ti} with index greater of "k" are automatically loosen in term of S_T and will be considered *de facto* only as simple **S** states also if they are set as S_T. All this because only S_{T1} intervenes to block and to replace itself with **T**. For instance if we have a situation type **T**, S_{T1}, S_{T3}, S_{T4} with S_{T2} replaced, if **T** will be replaced the new situation will be **T**, S_{T2}, S_{T3} without any S_{T1}.

HRT-ST-MESI protocol



HRT-ST-MESI Protocol – State Transaction Diagram

IBM patented full HRT-ST-MESI protocol[11][12]

- **I** state = Invalid Tag (*) – Invalid Data
- **H** state = Valid Tag – Invalid Data

- **I** state is set at the cache initialization and its state changes only after a processor Read or Write miss. After it will not return more in this state.

- **H** has the same functionality of **I** state but in addition with the ability to capture any bus transaction that match the Tag of the directory and to update the data cache.

- After the first utilization **I** is replaced by **H** in its functions

  - The main features are :

    - Write Back
    - Intervention both in sharing-clean and dirty data – from T-M-R-E
    - Reserve states of the Tagged (Shared-Tagged)
    - Invalid **H** state (Hover) auto-updating

(*) – Note: The Tag for definition is always valid, but until the first updating of the cache line it is considered invalid in order to avoid to update the cache also when this line has been not still required and used.

POWER4 IBM protocol

States M-T-Me-S-I -Mu-S_L = RT-MESI+Mu[9]

- Use of the "shared line"

- Used in multi-core/module systems – multi L2 cache [9]
- This protocol is equivalent to the RT-MESI protocol for system with multi L2 cache on multi-module systems

    - **S_L** - "Shared Last" equivalent to **R** on **RT-MESI**
    - **Me** - "Valid Exclusive" = **E**
    - **Mu** – unsolicited modified state

        - special state – asking for a reservation for load and store doubleword (for 64-bit implementations)

- "Shared intervention" from **S_L** is done only between L2 caches of the same module
- "Dirty intervention" from **T** is done only between L2 caches of the same module
- **Operations**

    - *Write Allocate*
    - *Intervention*: from M-T-VE-S_L = M-O-E-S_L
    - *Write Invalidate*
    - *Copy-Back*: M-T replacement

    - Note : T and S_L – Intervention only on the locale module

---

General considerations on the protocols

Under some conditions the most efficient and complete protocol turns out to be the HRT-ST-MESI protocol.

- Write Back
- Intervention both with dirty than shared-clean data
- Reserve states of the Tagged state (Shared-Tagged)
- Invalid **H** (Hover) state auto-updating

References

1. Multi-processor system with shared memory – http://www.freepatentsonline.com/5701413.html
2. Method for transferring data in a multiprocessor computer system with crossbar interconnecting unit – http://www.google.com/patents/EP0923032A1?cl=en
3. Specification and Verification of the PowerScale Bus Arbitration Protocol: An Industrial Experiment with LOTOS, Chap. 2, Pag. 4 – ftp://ftp.inrialpes.fr/pub/vasy/publications/cadp/Chehaibar-Garavel-et-al-96.pdf
4. , Archibald, J. and Baer, J. 1986 – Cache coherence protocols: evaluation using a multiprocessor simulation model. ACM Trans. Comput. Syst. 4, 4 (Sep. 1986), 273-298 – http://ctho.org/toread/forclass/18-742/3/p273-archibald.pdf.
5. MPC7400 RISC Microprocessor User's Manual – http://pccomponents.com/datasheets/MOT-MPC7400.PDF
6. Cache-coherency protocol with recently read state for data and instructions – IBM patent – http://www.google.com/patents/US5996049

7. An Introduction to the Intel® QuickPath Interconnect – http://www.intel.ie/content/dam/doc/white-paper/quick-path-interconnect-introduction-paper.pdf
8. Forward state for use in cache coherency in a multiprocessor system – Intel - https://www.google.com/patents/US6922756
9. "POWER4 System Microarchitecture", http://www.cc.gatech.edu/~bader/COURSES/UNM/ece637-Fall2003/papers/TDF02.pdf Archived (https://web.archive.org/web/20131107140531/http://www.cc.gatech.edu/~bader/COURSES/UNM/ece637-Fall2003/papers/TDF02.pdf) 2013-11-07 at the Wayback Machine
10. IBM PowerPC 476FP L2 Cache Core Databook – https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/8D5342097498C81A852575C50078D867/$file/
11. Cache Coherency Protocol Including an HR State – IBM patent – https://www.google.com/patents/US6275908
12. Cache Coherency Protocol with Tagged State for Modified Values – IBM patent – http://www.google.com/patents/US6334172
13. MPC750UM/D 12/2001 Rev. 1 MPC750 RISC Microprocessor Family User's Manual – http://www.freescale.com/files/32bit/doc/ref_manual/MPC750UM.pdf
14. Pentium Pro and Pentium II System Architecture pg. 160 – Di T. Shanley – https://books.google.com/books?id=MLJClvCYh34C&pg=PA160&lpg=PA160&dq=Pentium+protocol+cache&source=bl&ots=gaEvaTy&sig=QQEM1krp-H_0KUhi5Ti2bmxU2kU&hl=it&sa=X&ei=Zt6KT7icG8TE4gShzaDwCQ&ved=0CDcQ6AEwAg
15. AMD64 Technology – AMD64 Architecture Programmer's Manual Volume 2: System Programming – http://developer.amd.com/wordpress/media/2012/10/24593_APM_v21.pdf Archived (https://web.archive.org/web/20160303230732/http://developer.amd.com/wordpress/media/2012/10/24593_APM_v21.pdf) 2016-03-03 at the Wayback Machine
16. Sweazey, P., and Smith, A. J. A class of compatible cache consistency protocols and their support by the IEEE Futurebus. In *Proceedings of the 13th International Symposium on Computer Architecture*. IEEE. New York, 1986, pp. 414-423.) – http://pdf.aminer.org/000/419/524/a_class_of_compatible_cache_consistency_protocols_and_
17. Mark S. Papamarcos and Janak H. Patel. In ISCA '84: Proceedings of the 11th annual international symposium on Computer architecture, pages 348-354, New York, NY, USA, 1984. ACM – http://www.researchgate.net/publication/220771512_A_Low-Overhead_Coherence_Solution_for_Multiprocessors_with_Private_Cache_Memories/file/504
18. Using cache memory to reduce processor-memory traffic". Proceedings of the 10th annual international symposium on Computer architecture – ISCA '83. International Symposium on Computer Architecture: Stockholm, Sweden, June 13–17, 1983.pp. 127-128 – http://courses.cs.vt.edu/cs5204/fall11-kafura/Papers/TransactionalMemory/Goodman-SnoopyProtocol.pdf
19. Advanced Computer Architecture, 2E pg. 301 – Di Hwang – https://books.google.com/books?id=m4VFXr6qjroC&pg=PA301
20. Cache memory and related consistency protocol, Inventor Ferruccio Zulian, Bull HN ISI – http://www.google.com/patents/EP0396940B1?cl=en
21. Comparing Cache Architectures and Coherency Protocols on x86-64 Multicore SMP Systems – http://people.freebsd.org/~lstewart/articles/cache-performance-x86-2009.pdf
22. Cache Organization and Memory Management of the Intel Nehalem Computer Architecture http://gec.di.uminho.pt/Discip/MInf/cpd1011/PAC/material/nehalemPaper.pdf
23. David Kanter (2007-08-28), "The Common System Interface: Intel's Future Interconnect" (http://www.realworldtech.com/common-system-interface/5/), *Real World Tech*: 5, retrieved 2012-08-12

24. Optimizing the MESI Cache Coherence Protocol for Multithreaded Applications on Small Symmetric Multiprocessor Systems – http://tibrewala.net/papers/mesi98/ Archived (https://web.archive.org/web/20161022140749/http://tibrewala.net/papers/mesi98/) 2016-10-22 at the Wayback Machine

---