**4.6 - MESIF**

MESIF has an additional state to MESI the Forward state-F.

“The F state designates a copy of data from which further copies can be made” [PATENT 1].

The F state is quite different than the O state of MOESI, a line holding data in the F state, like the O state is in charge of forwarding its data on to requesting nodes however unlike the O state, data stored in F is coherent with memory. There are 2 possible methods by which the F state can forward the data onto a requesting Node.

1. Transfer the data to the requesting node in the F state and go to the S state.
2. Transfer the data to the requesting node in the S state and stay in the F state.

The former implementation will be used in the following explanation and in the animation, as examples and explanations in the patent files describing the MESIF protocol use this assumption.

The F state removes the concern of multiple data copies being received from a request. Data stored in the F state is coherent with main memory when an F state cache line is being replaced a flush back to main memory is not required.

Broadcasts probes are sent from the source node in MESIF opposed to the home node, coherence in the MESIF protocol is enforced by every node in the system. Conflicts in the MESIF protocol unlike the MOESI protocol, which serialises requests on the home node, are dealt with via conflict messaging, with the conflicts being resolved by the home node.

The F state also helps enforce serialisation, when receiving multiple requests for a cache line it will respond to the first request and queue subsequent requests until the first request has finished.

The benefits of MESIF also present problems, when a line enters the M state the protocol does not have the features of an O state so a Modified line cannot be shared over multiple caches at the same time, unless the line is first flushed to main memory.

*States*

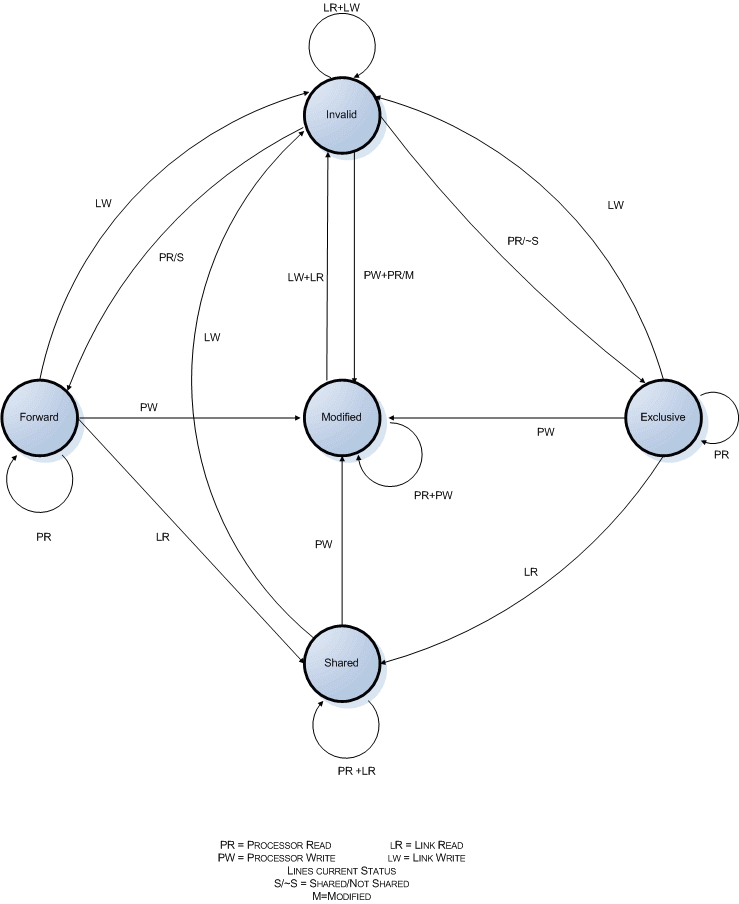
Modified – The most recent copy of this memory location is present in this cache only; the copy in memory is out of date or stale. The CPU in possession of this cache line may write new data to this line.

Exclusive – This is the only up to date copy of the data outside of memory and it is coherent with memory.

Shared – The most recent copy of this memory location is present in this cache and may be present in other caches in the Shared or Forward state. The copy in memory is up to date.

Invalid – This copy of the data is invalid.

Forward – The most recent copy of this memory location is present in this cache and may be present in other caches in the shared state. The copy in memory is up to date. Only one cache can hold a copy of the data in Forward at a time.

**Fig 4.16** MESIF transitions

*Protocol messages*

Intel provided more information than AMD on their protocols messaging so a more detailed description of the messaging involved in the MESIF protocol is possible.

***Request Messages*** [PATENT 2]

Request messages are broadcasts and send to every Node on the System  
  
*Port Read Line (PRL):* This is a request for a copy of a data segment such as, for example, a cache line.   
*Port Read Invalidate Line (PRIL):* This is a request for a copy of a data segment where the provider node's copy of the data is invalidated. This message can also be referred to as a "request for ownership."   
*Port Write Line (PWL):* This message causes data (e.g., a modified cache line) to be written to memory. This message can also be referred to as a "dirty eviction."   
 **Response Messages** [PATENT 2]

The following messages are messages sent from Peer (i.e., Non-Home) nodes to the Requesting node in response to requests described above.   
 *Invalid State Acknowledgement (IACK):* This message is a response to a request (PRL, PRIL, PWL) when the node sending the response has an invalid copy of the requested data or no copy of the requested data.   
*Shared State Acknowledgement (SACK):* This message is a response to a request when the node sending the response has a copy of the requested data in the Shared state.   
*Acknowledgement of Data Received (DACK):* This message acknowledges the receipt of requested data.   
*Conflict:* This message indicates that there is a copending request for the requested cache line.   
*Data & State:* This message provides the requested data as well as an indication of the state of the data in the Requesting node.   
  
**Messages to Home Node** [PATENT 2]

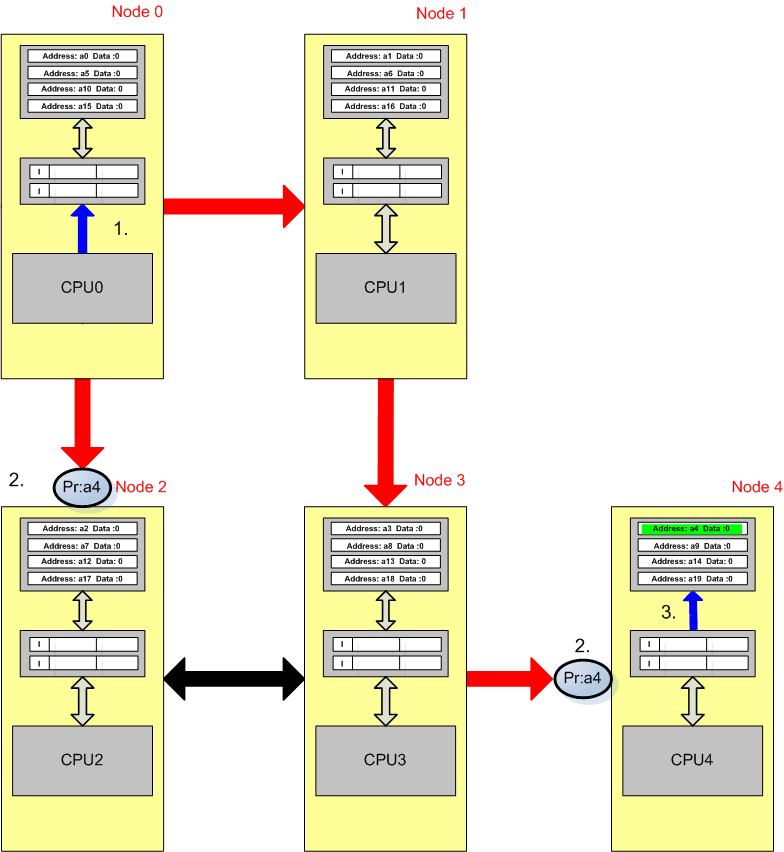
These messages are transmitted to the Home node by a Peer node.   
  
*Read (Conflicts):* This message requests data from the Home nodes and lists all conflicts, if any.   
*CNCL (Conflicts):* This message is sent to the Home node in response to a hit in a Peer node and lists all conflicts, if any. This message cancels the Home node's prefetch operation.   
*Data (Conflicts):* This message is used to write back data and lists all conflicts, if any.

**Messages From the Home Node** [PATENT 2]  
  
These messages are sent from the Home node to the Peer and/or Requesting nodes.   
  
*Data:* This message includes the requested data and can indicate the state of the data (M/E/F/S) to be used by the Requesting node.   
*Acknowledge (ACK):* This message indicates that the requested data has been sent to the Requesting node.

*Wait:* This message causes the receiving node to pause before sending further messages.   
*Transfer (XFR):* This message causes the receiving node to transfer data to the node indicated in the message.

*Demonstration*

The following demonstration shows the basics of the MESIF protocol and simple conflict resolution.

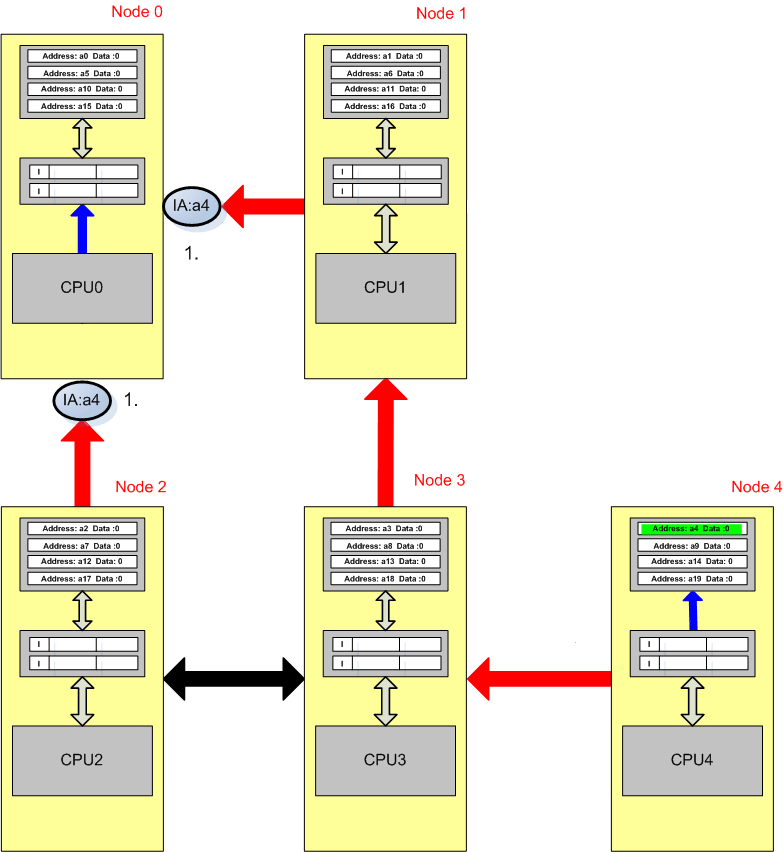
**Fig 4.17 -** Probe a4

CPU0 - Reads memory location a4.

1. CPU0 checks its local cache for a valid copy of a4.
2. Not finding a valid copy it sends a broadcast probe to every node on the system.

The source node will probe every Node even when the source node is the home node.

1. The home node on receiving the probe initiates a memory access to location a4.

**Fig 4.18** - Peer nodes reply to source node

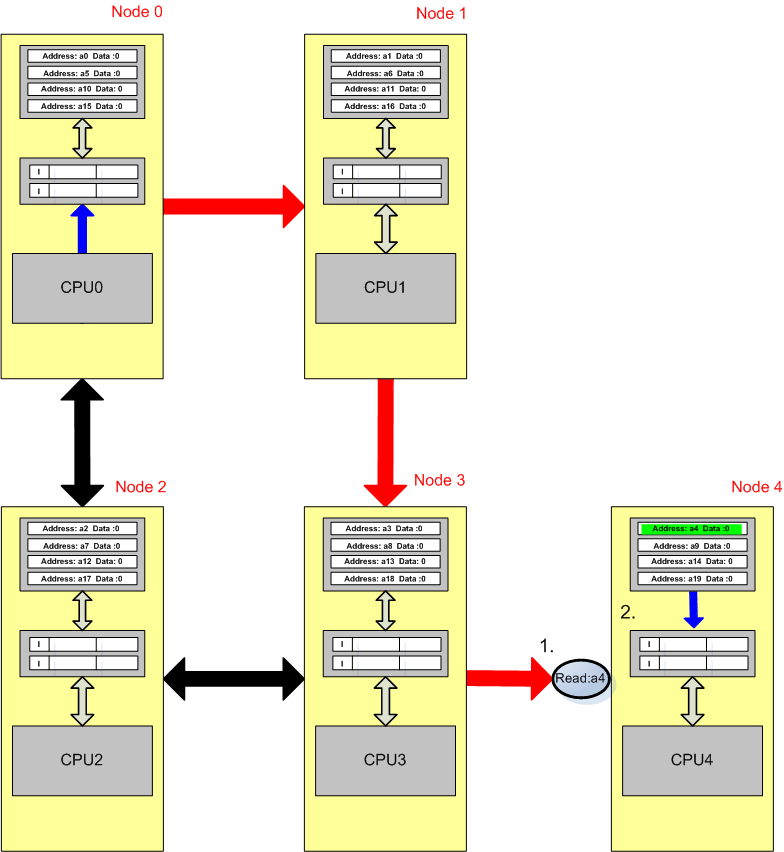
CPU0 – Reads memory location a4.

1. The source node now awaits all the acknowledgements from the peer nodes.

Unlike the MOESI protocol it does not wait on the data from the home nodes memory access.

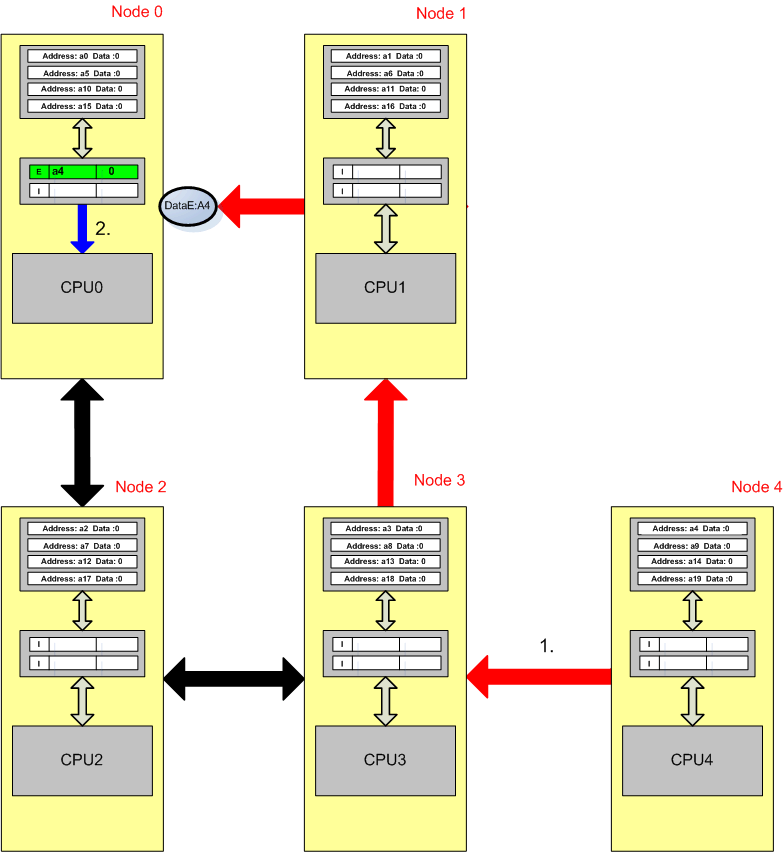
The Nodes all reply with invalid acknowledgements, as they do not have a cached copy of the data.

The nodes would only reply with the data if it were held in the M, F or E state. Not the shared state as the F line is in charge of forwarding data, if the F line on the system was replaced and left only S lines the data would be taken from memory, the only time a line is not coherent with memory is when it’s in the Invalid or Modified state.

**Fig 4.19** - Read Request

CPU0 - Reads memory location a4.

1. On receiving all its replies the source node sends a Read message confirming the read from memory, the home Node continues it access from memory and will reply when finished.

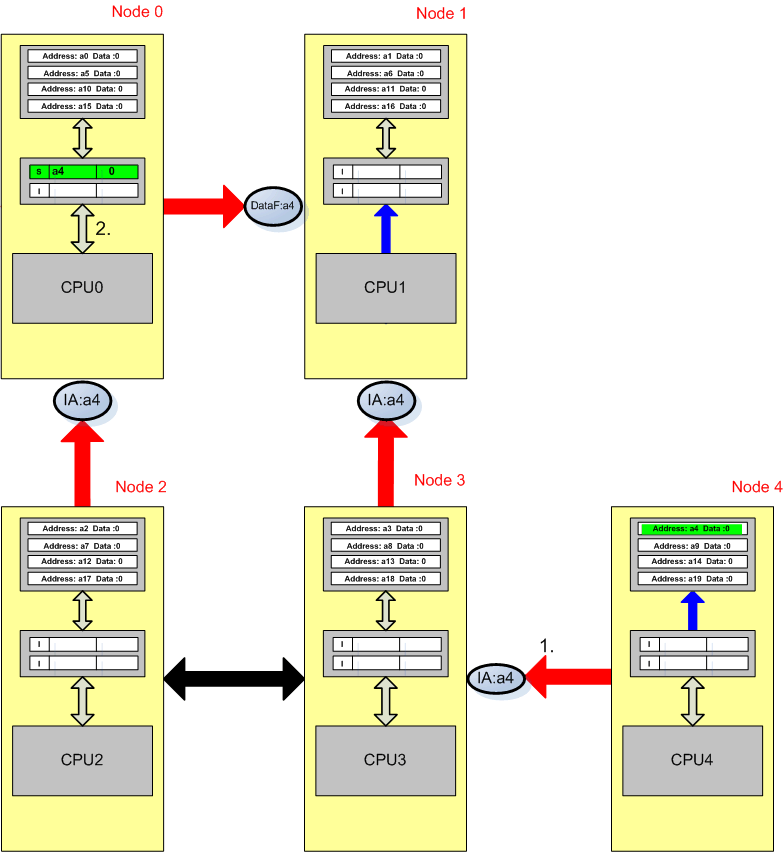
**Fig 4.20** - DataE message

CPU0 - Reads memory location a4.

1. The Home Node, node 4 sends the data in the Exclusive state in a DataE message, as no other nodes where found to have a copy of it on the system.
2. The cache stores it in exclusive and sends the data to its CPU.

CPU1 – Reads memory location a4.

CPU1 checks its local cache, no valid copy is found so it sends a broadcast probe around the system.

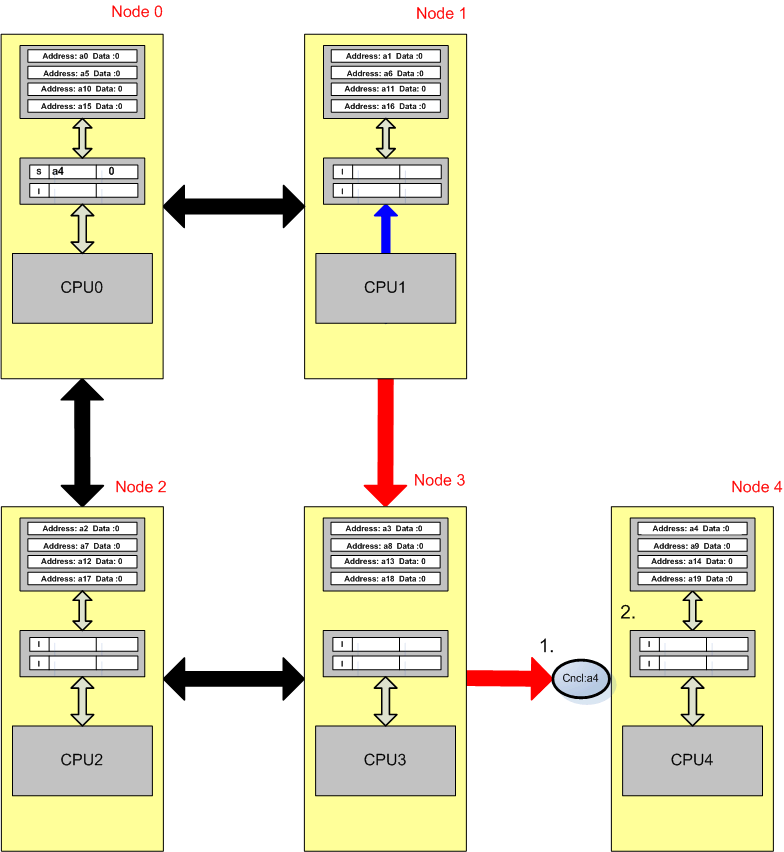
**Fig 4.21 -** Probe responses

CPU1 - Reads memory location a4.

1. All the nodes apart from node 0 reply with invalid acknowledge messages.
2. Node 0 as it has a copy of the data in the E state replies with a DataF message; on sending the data forwards in the F state it switches its copy to the S state.

An alternate method would have CPU0 switching to the F state and forwarding the data onwards in the S state to the Source Node.

On receiving the data from node 0 the source node can use the data; however it can not make the effects of this visible to the system until it has received an acknowledgement from the home node.

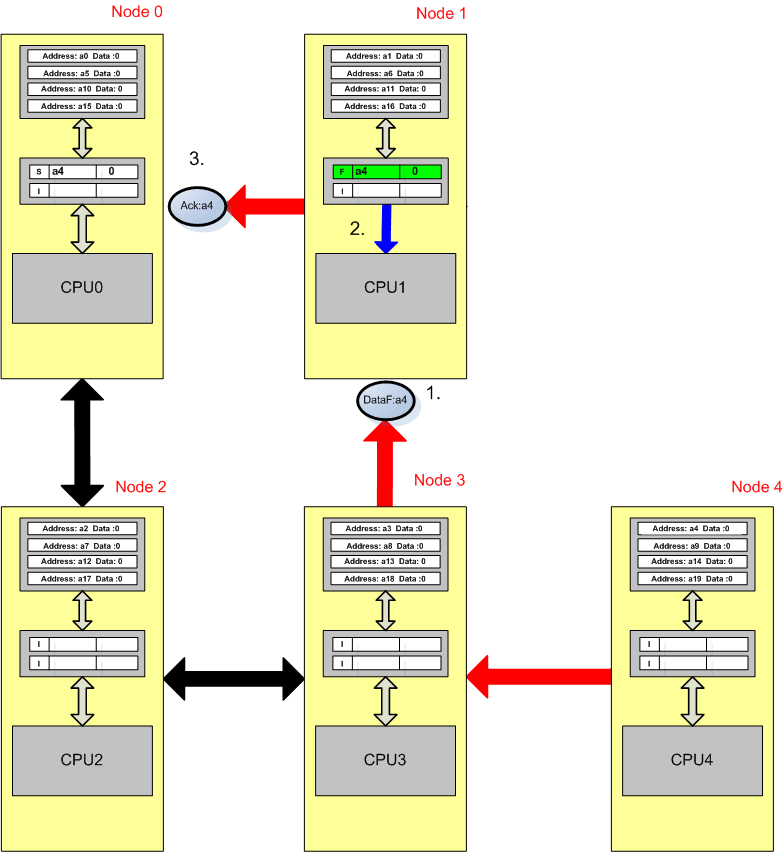
**Fig 4.22** Cancel Message

CPU1 - Reads memory location a4.

1. On receiving all it acknowledgements, the source node sends a cancel message to stop the home nodes memory access as it has already received a valid copy of the data.

The ability to cancel a memory access when there’s a cached value on the system gives the MESIF protocol a series edge over MOESI.

1. On receiving the cancel message the home node stops retrieving the data from memory.

**Fig 4.23** - DataF Confirmation

CPU1 - Reads memory location a4.

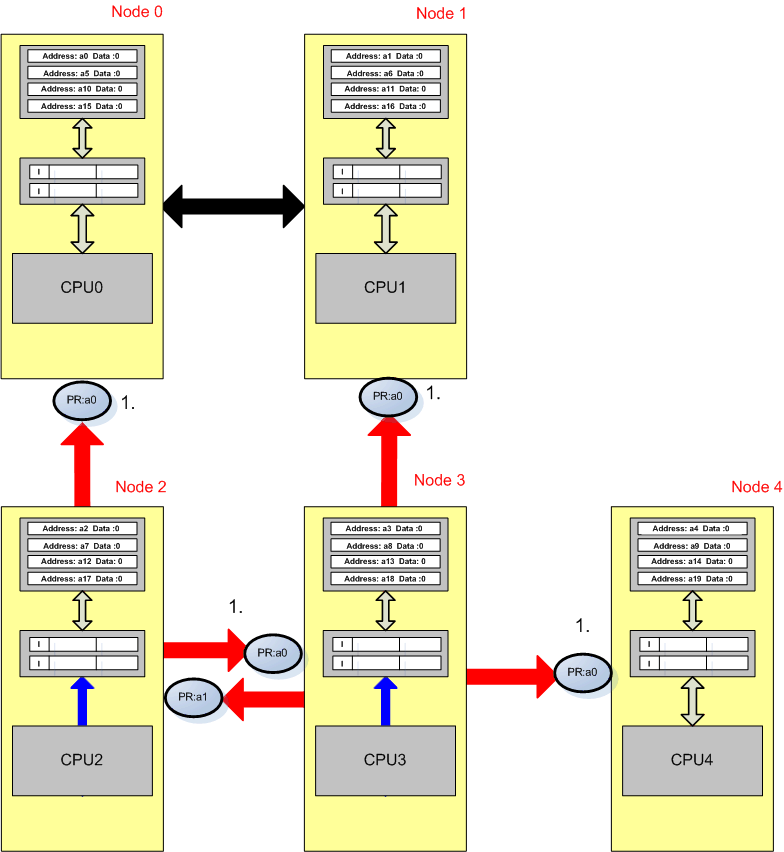
1. The Home Node confirms that the source Node can take the data in the F state.
2. The data is stored in the F state and the CPU can continue, it may already have used the data it received previously from node 0, on receiving the acknowledgement it can now transfer the data on to subsequent nodes.
3. On receiving it’s confirm message the source node forwards an acknowledge onto node 0. Node 0 had previously forwarded the data to the source node, any replies to subsequent requests sent to node 0 for a4 are queued until node 0 receives it acknowledgement.

An alternate method to the source node sending the acknowledge message is for the home node to send it. The source node was chosen to send the acknowledge message to make the animation simpler to follow as it reduced the amount of packets on screen at once.

*Conflict Resolution*

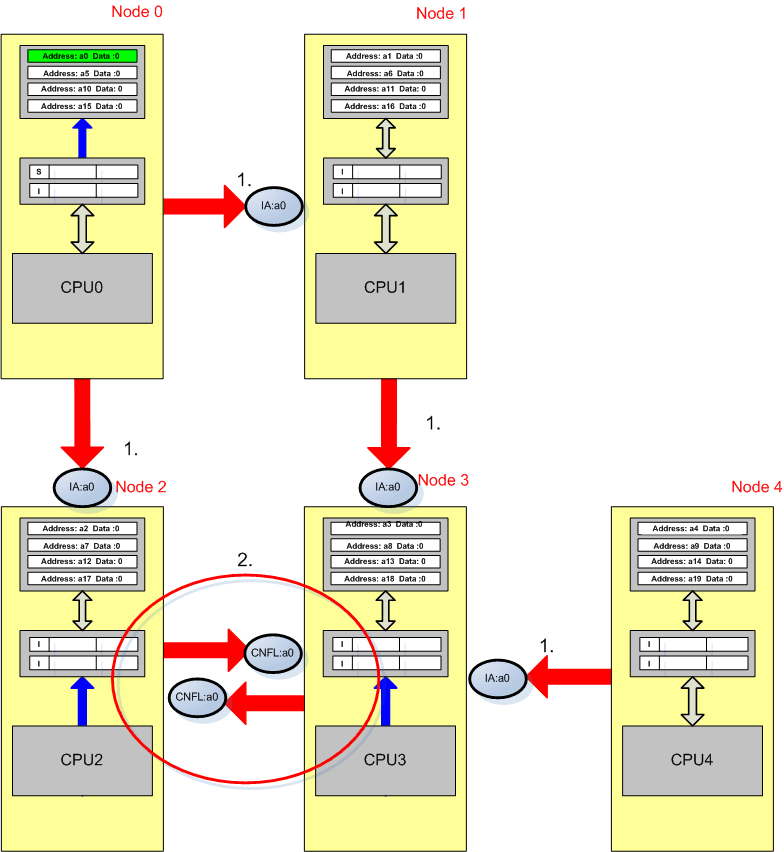
Conflicts are resolved using conflict messaging, if two CPUs are requesting the same address they reply with a conflict message to the other, these conflicts are then resolved at the home node.

The following diagrams detail a basic conflict resolution between two nodes, there are a multitude of different conflict scenarios but they are all along the basis of this explanation.

**Fig 4.24** – CPU2 and CPU3 broadcast probes

CPU2 and CPU3 request to read memory location a0 in parallel.

1. The two nodes send broadcast probes to every node on the System.

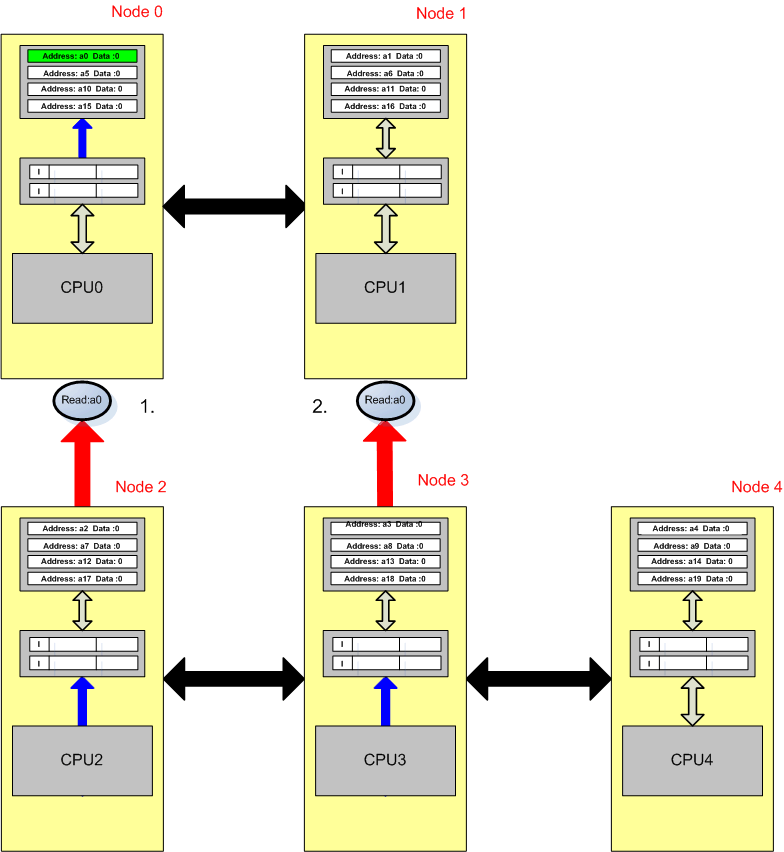
**Fig 4.25** - Conflict replies

CPU2 and CPU3 request to read memory location a0 in parallel.

1. No Node in the system has a valid copy of the data they all reply with an Invalid Acknowledgement messages.
2. As node 2 and 3 are currently trying to retrieve a0, on receiving a probe they reply with a conflict message (CNFL), informing the requesting node that it is trying to access the data also.

If either node 2 or 3 where requesting the data for a write instead of a read the reply would be an conflict invalidate (CNFLI) message to inform the requesting node that its performing a write.

It’s vital that the read and write conflicts are differentiated to maintain coherency in the system

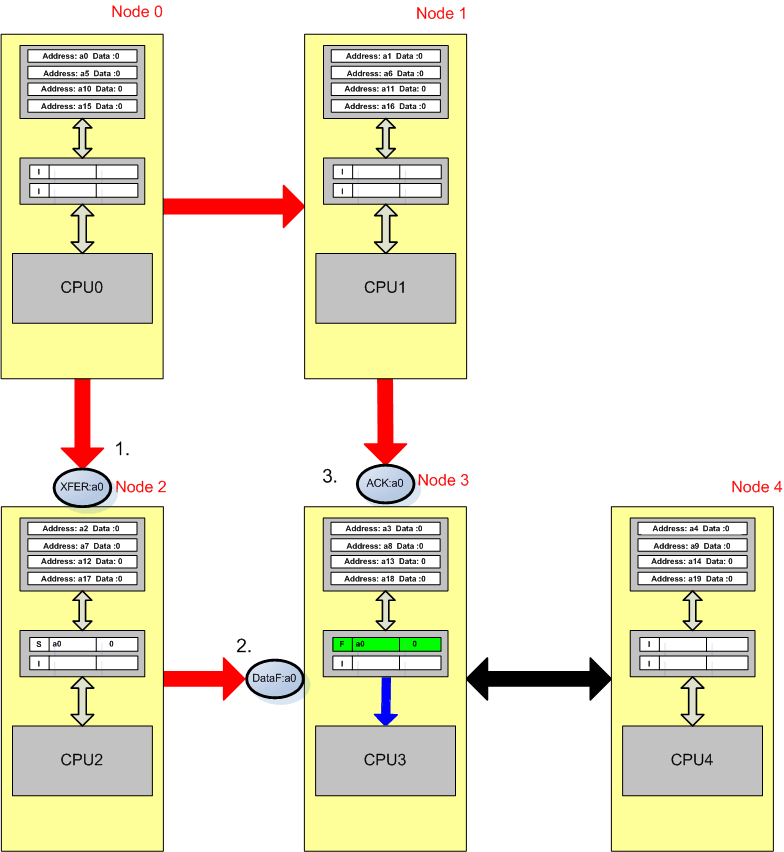
**Fig 4.26 -** Read Requests

CPU2 and CPU3 request to read memory location a0 in parallel.

1. The read requests contain all the conflicts received from the broadcast probes.

Node twos request arrives first and it retrieves a valid copy of the data in the E state, the home node stores the conflicts until they are resolved.

1. The home node on receiving node threes read request treats it differently than node twos request as it has conflicts already stored for that address location. If the read request where dealt with normally CPU3 would receive the data in the E state and both CPU2 and CPU3 would have an exclusive copy of the line, this is where the conflict messages intervene.

**Fig 4.27** - Transfer Message and forward

CPU2 and CPU3 request to read memory location a0 in parallel.

1. On receiving the read request from node 3 the Home node sends a transfer message (XFER) to node 2, this is possible because due to the conflict messages the home node has stored where the most recent value of the data recedes.

As mentioned earlier if one of the CPUs was performing a write opposed to a read operation it would send a conflict invalidate message instead on a conflict message, this is integral to maintaining coherence.

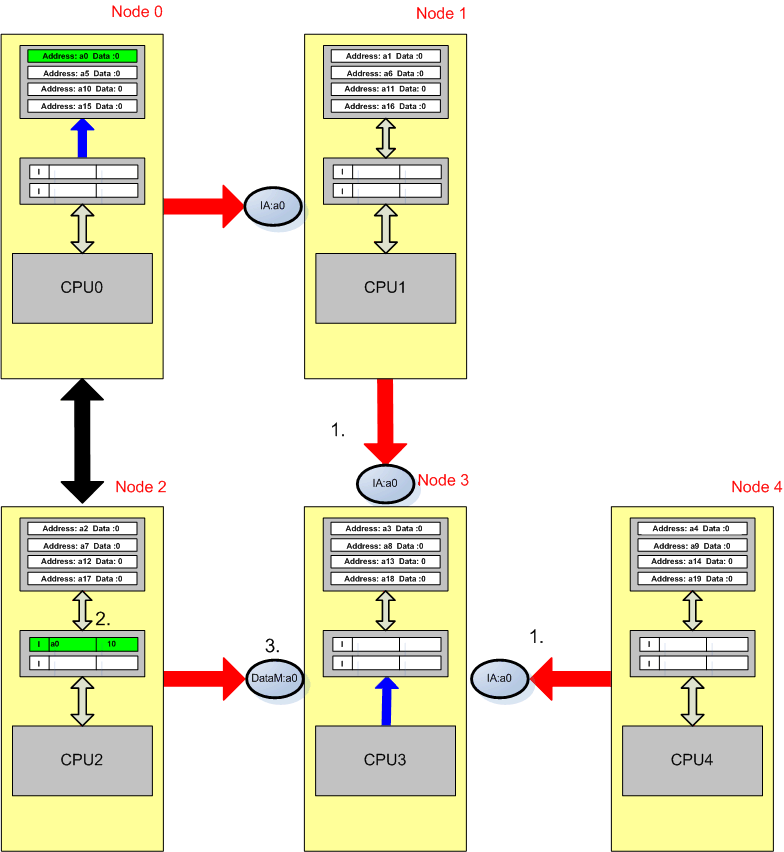
To maintain coherence a CPU before performing a write operation must invalidate external copies of the data it’s writing to. To enforce this invalidation when multiple processors request a line, the CNFLI message is used, the home node on seeing a CNFLI will in future send transfer invalidate messages (XFERI) instead of a XFER to ensure there are no out of date copies cached on the system.

The XFERI message informs the node to transfer its copy of the data to another node and then invalidate its copy.

1. On receiving the transfer message CPU2 changes the state of its cached a0 value from E to s and forwards a0 onto Node3 in a DataF message.

On receiving the copy of a0 in the F state cache 3 stores the line in F.

1. On sending the transfer message the home also sends an acknowledge message to Node 3 to inform that there are no further conflicts and to resume normal operation.

**Fig 4.28 -** Forward Modified Data

*Problems with the M state*

The MOESI protocols key advantage over MESIF is its owned state, which allows multiple cache copies of modified data on the system at once. From **FIG 4.28** CPU twos cache held a0 in the Modified state, on receiving a read request probe from node 3 it must invalidate its copy of the data and transfer it to Node3 in the Modified state, this is because there can be only one modified copy of a data line on the system. This can lead to a lot of wasted transfer cycles if it’s a commonly used location. A possible solution to this would be to flush a0 to main memory and then transfer it in the shared state allowing several copies on the system at once.