University of Dublin

TRINITY COLLEGE

Cache Coherency
The MOESI and MESIF Protocols

Stephen Kirk
B.A.I. Engineering
Final Year Project May 2008
Supervisor: Jeremy Jones

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland

Declaration

I hereby certiy that this material is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my own work.

Signed: .....

Date: .....

Acknowledgements

I'd obviously like to thank Jeremy Jones for supervising my project and guiding me week in and week out in what to do.
I'd like to thank my parents for feeding a clothing me throught college, and finally Richard Carrol for listening to my presentation and giving me ideas on how to present the animation.

Abstract

Cache Coherence and the protocols that govern it are a core topic of the majority of Computer Engineering and Computer Science programmes worldwide. As computer systems are becoming increasingly complex with an escalating number of caches and CPUs new point-to-point(P2P) system architectures are being designed to take full advantage of these superior systems, and as such new cache coherence protocols are needed and being developed.

Recently AMD has released a new line of Opteron computers utilising HyperTransport p2p technology and the new MOESI Cache Coherence Protocol, Intel is to follow suit with the release of its Nehalem line of computers in the 4th quarter of 2008, Nehalem will use Intel's QuickPath Interconnect and the MESIF protocol. P2P systems are set to be the standard for the foreseeable future, and a central topic in both industry and academia.

Cache coherence protocols can be difficult to comprehend and follow as there are many actions occurring simultaneously. Standard teaching practices are left wanting in their description and demonstration of cache coherence, to remedy this interactive 2D animations where developed to describe these protocols [JONES 1].

This report aims to show the background, theory and technology underlying the MOESI and MESIF Cache Coherency protocols; it also aims to show the development of animations used to demonstrate said protocols.

Contents:

1 - Introduction

1.1 – Aims

This project aims to present interactive scalable 2D vector based technical animations of the MOESI and MESIF cache coherency protocols as implemented on a multiprocessor system using a point-to-point (p2p) architecture for the purpose of educating students and engineers.
This paper will inspect the following areas.

1.2 - Chapters

State of the Art Review
Chapter 2 contrasts and compares p2p architecture with its predecessor the Front Side Bus (FSB).

HyperTransport
Chapter 3 gives an in-depth look at the specifications and features of HyperTransport point-to-point technology.

Cache Coherency Protocols
Chapter 4 explains the basics of cache coherency, the basics of coherency are demonstrated using the MESI protocol. The theory and features of the MESIF and MOESI protocols are then described.

Vivio
The basic features of Vivio, the animation tool used in this project, will be discussed and how they are relevant to this project.

Animation: Design and Implementation
Chapter 6 shows the design of the animations and the basic classes of the programmes. It also gives an overview of the protocol logic implemented.

Evaluation
This final chapter discuss what was achieved and the problems encountered on this project.

2 - State Of the Art Review

2.1 – Introduction

This chapter concerns itself with point-to-point (p2p) technology and its role in computer architecture. A comparison between p2p bus architecture and its predecessor the front side bus (FSB) will be used to highlight the beneficial features of p2p technology and the reasons manufactures are switching to p2p interconnects. The core topic of comparison of these links is that of retrieving data from main memory and the performance of these architectures on multiprocessor systems. Technical specifications of either technology or that of cache coherence will not pursued in-depth in this chapter.

2.2 - Front Side Bus

The FSB is a bi-directional link that connects the CPU of a system to main memory and other components. Several CPUS and components may be connected to the same bus.

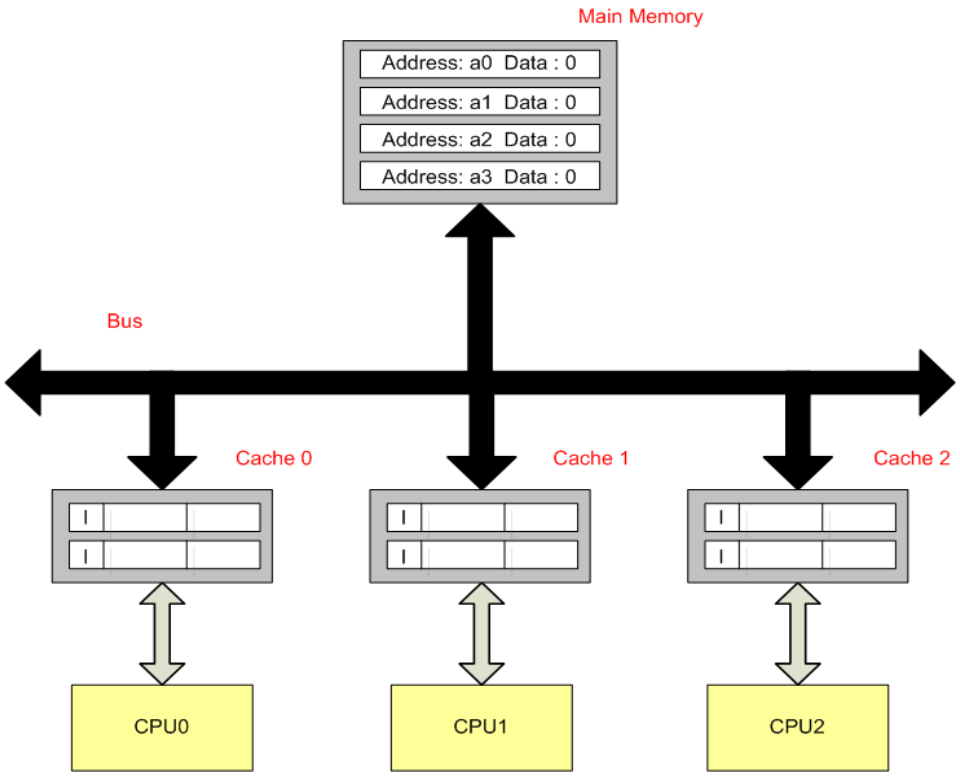


Fig 2.1 – Front Side Bus layout

The majority of computer applications require multiple memory accesses as they use more memory than the CPU's caches can store. The performance of a CPU is therefore highly dependent on the speed of its FSB in retrieving data from main memory.

The major problem with the FSB is that it doesn't scale well to multiprocessor systems, on a single processor system the bus is a low cost flexible solution suited to the task, however as more processors are added onto the shared bus performance starts to decline. This is due to the nature of the bus, on a FSB only one process at a time can access the bus, this causes a serious speed

bottleneck when multiple processes are trying to access the bus in parallel, add to this the memory access time and a multiprocessor systems performance is seriously degraded. Attempts to reduce accesses to memory by increasing cache size and the number of caches have reduced the aggregate amount of memory accesses but still haven't resolved the problem.

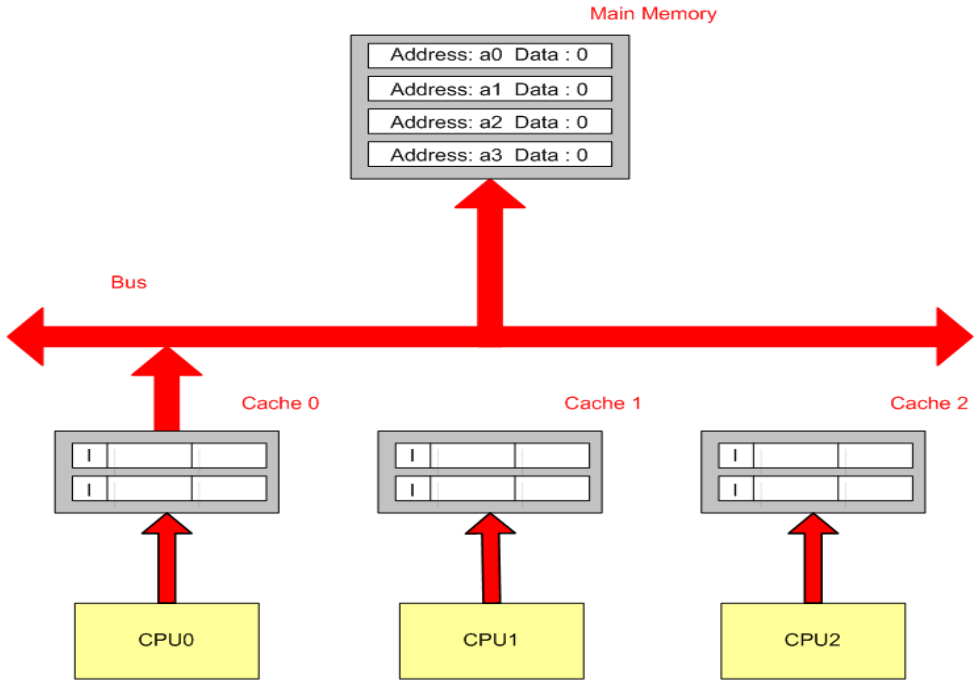


Fig 2.2 – Multiple bus accesses

As shown in **Fig 2.2** when multiple processors try to access the CPU at once it leads to a queue. If the access order of the CPUs to the bus where CPU0 followed by CPU1 and CPU2, it would result in CPU2 taking three times as long to perform a memory access. In essence there's a possibility of a memory access taking N times longer to complete (where N is the number or processors in a system).

2.3 - Point-to-Point

P2P links are high-speed bi-directional links connecting two components. Unlike a FSB a p2p link only connects two components.

Fig 2.3 shows the system architecture required for a p2p system. Every node in the system is connected by a p2p link. Nodes communicate with each other by passing messages, nodes not directly connected communicate via message forwarding as shown in **Fig 2.4**. A system node may consist of several components, in this case each node only contains one processor with its own local cache and memory. Memory in this architecture is split between the processors with a global addressing system; requests for memory not stored locally are forwarded onto another node.

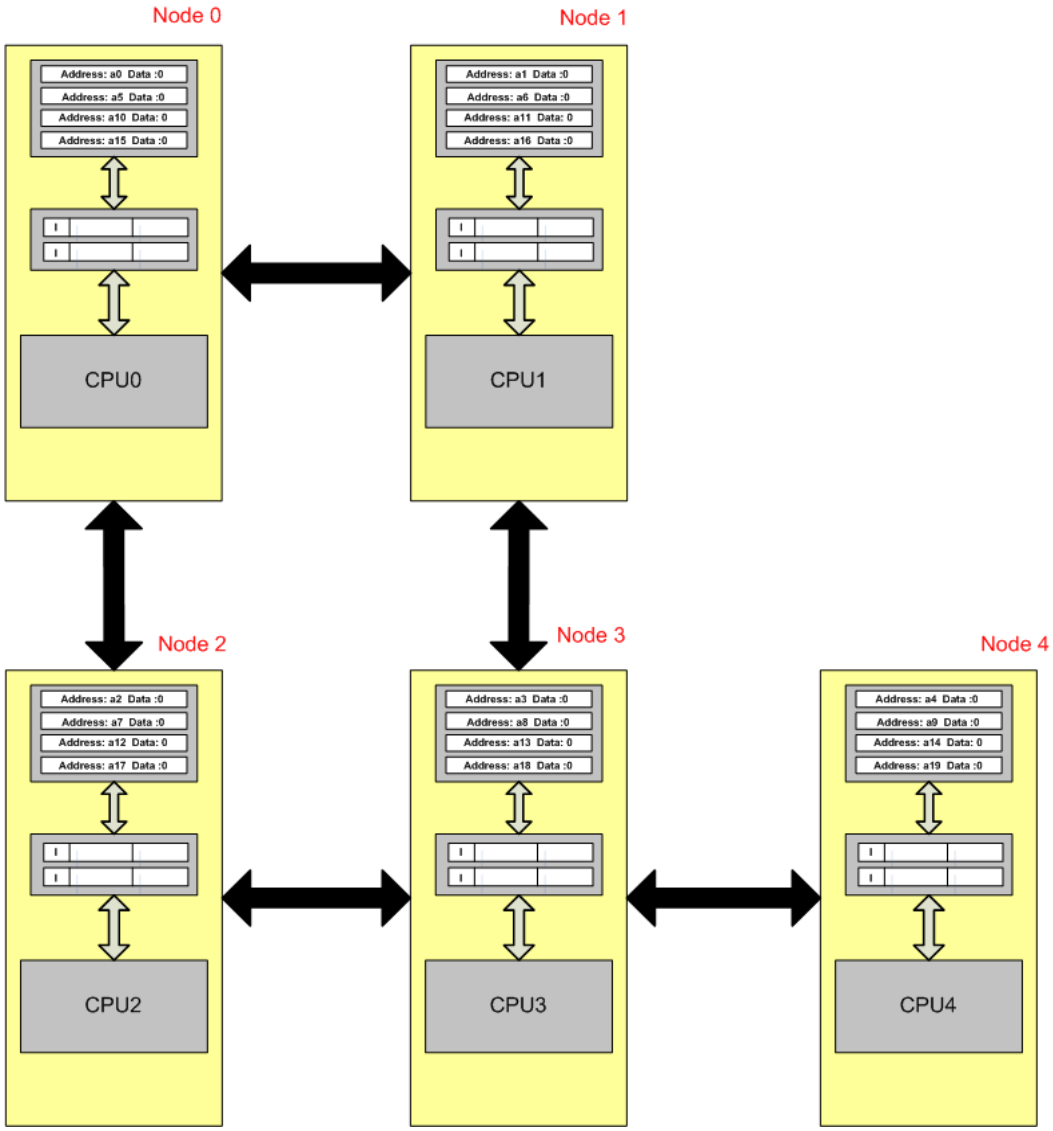


Fig 2.3 - Point-to-Point system architecture

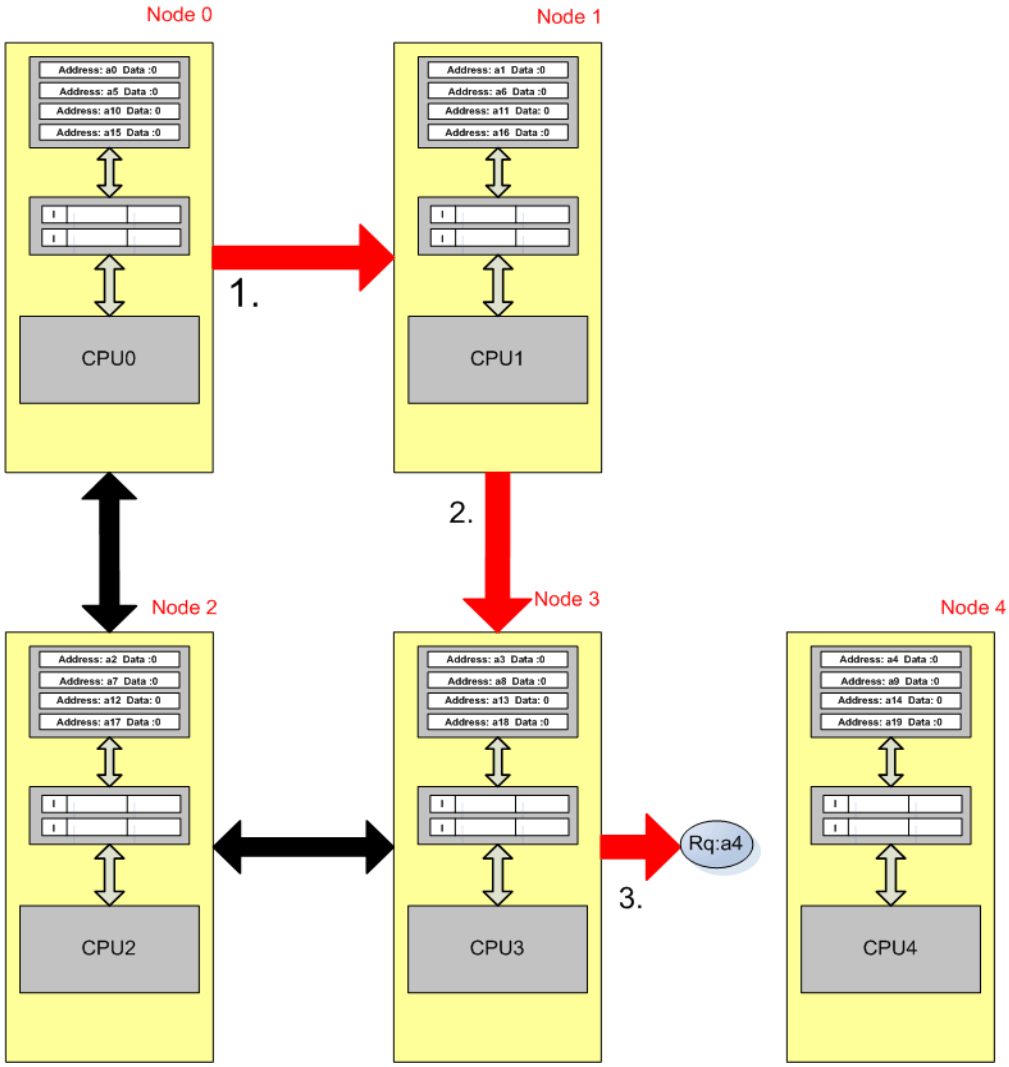


Fig 2.4 - Forward request from Node 0 to Node 4

A p2p systems links are bi-directional and provide separate paths for input and output operations allowing the CPU to transmit and receive data simultaneously. On a traditional FSB input and output operations where serialised and therefore much slower.
As shown in **Fig 2.5** p2p also allows all the CPUs to send requests in parallel allowing multiple simultaneous memory accesses, the process in **Fig 2.5** would take up to 4 times as long to execute on a FSB architecture.

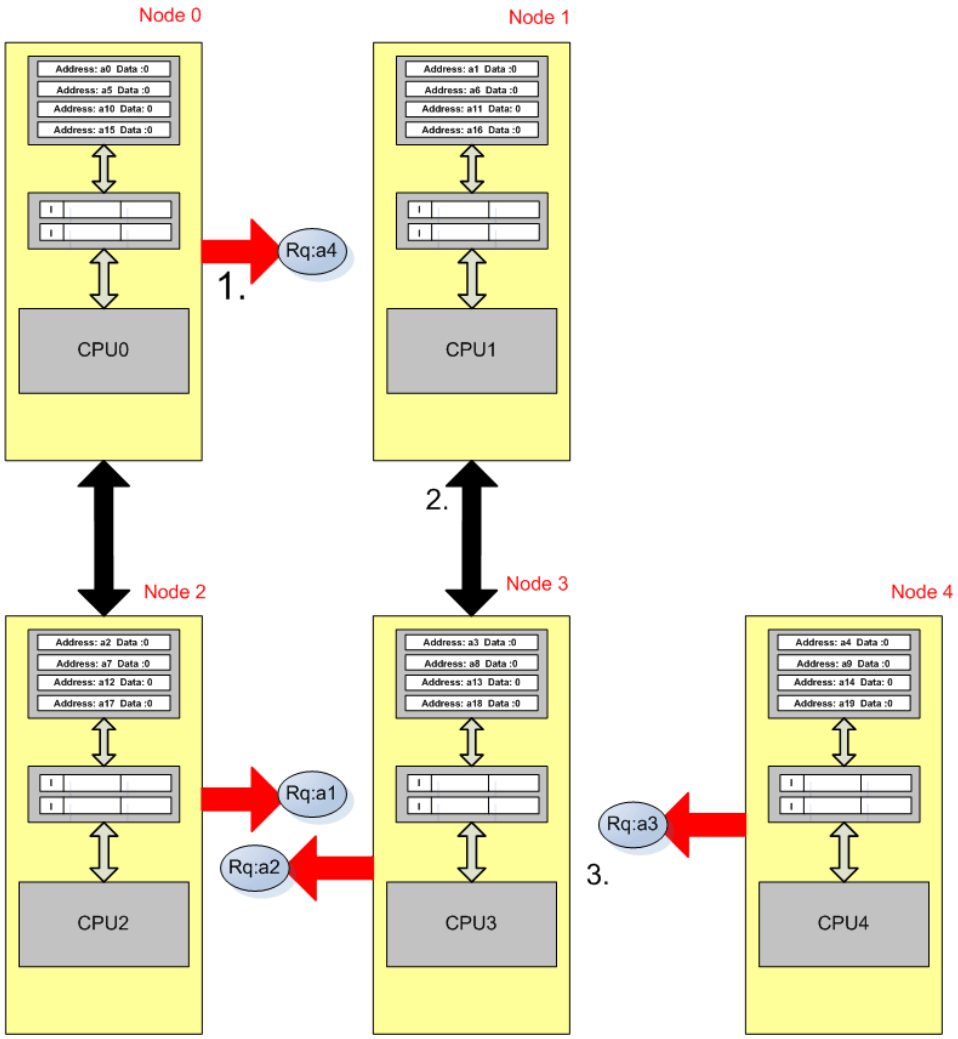


Fig 2.5 - Multiple Requests

2.4 Summary

The FSB while good in its time is an outdated technology for today's multiprocessor systems; the speed bottlenecks brought about by multiple accesses to a FSB erodes the gains from having multiple processors. The advent of the p2p buses make simultaneous memory accesses a reality, allowing greater parallelism in today's computers. The p2p bus is set to replace the FSB for the majority of future computers.

3. HyperTransport

3.1 - Introduction

This chapter looks at the newly developed HyperTransport (HT) p2p technology. Intel's QuickPath Interconnect is yet to be released, and as a full set of technical specifications is currently unavailable it is been overlooked.
The key features and specifications of HT will be shown along with the flexibility of the HT technology.

3.2 - Background

"HyperTransport technology is a high-speed, low latency, point-to-point link designed to increase the communication speed between integrated circuits in computers, servers, embedded systems, and networking and telecommunications equipment up to 48 times faster than some existing technologies."[AMD 1]

There are 3 main versions of HT, HT 1.0, 2.0 and 3.0. The specifications are defined and maintained by the HyperTransport Consortium

"The non-profit consortium publishes the specification, drives the development of future HyperTransport specifications and manages all specification issues. This enables the developer to confidently implement HyperTransport technology with the assurance that the resulting systems will be interoperable with other HyperTransport-based subsystems."[HYPER 1]

The founding members include AMD, SUN MICROSYSTEMS, APPLE COMPUTER, CISCO SYSTEMS, NVIDIA, ALLIANCE SEMICONDUCTOR, BROADCOM CORPORATION, PMC-Sierra and Transmeta lead the HT Consortium.

HT 3.0 is the most recent version of the HT technology and incorporates all the features of its previous incarnations giving it 100% backwards capability, so it will suffice to do a review of it alone.

3.3 – Specifications [HYPER 1]

- Maximum Clock Speed: 2.6 GHz
- Supported Clock Rates: *200 MHz, 400 MHz, 600 MHz, 800 MHz, 1 GHz, 1.2 GHz, 1.4 GHz, 1.8 GHz, 2.0 GHz, 2.4 GHz, 2.6 GHz*
- Maximum Aggregate Bandwidth (32-bit Links): 41.6GB/s

  (20.8 GB/s per Link)
- Utilises Packet Based Messaging

  Features:
  - PCI, PCI-X and PCI-Express Mapping

- AC Mode-capacitive coupling with ac/dc auto-sensing, auto-configuration
- HT Link Splitting 16 bit link as 2x8 bit Links.
- Hot plugging
- Dynamic Clock width/Link Adjustment
- Scalable Link Width 2 bits – 32 bits
- Asymmetric Link support (different link widths)
- Asynchronous link operation
- Dc Mode operation
- Priority Request interleaving
- Direct packets data streaming
- Error Retry

3.4 - Features

- Clock Rates

  HT3 supports all clock rates from the previous HT implementations this backward compatibility means for example that HT3 based CPUs can be installed on HT2 based motherboards.

- Low Latency

  The following features reduce HT latency (processing delays).

  - Forwarded Clock

    Unlike other serial links such as PCI-Express, which utilises clock encoding and decoding at both ends of the link (increasing the latency time), HT uses clock forwarding to transmit the clock onwards.

  - Low Packet Overhead

    Unlike PCI which requires up to 24 byte packet headers HT only has an 8 or 12 byte packet header depending if it's a read or write respectively. This is due to a less complex layer structure.

  - Priority Request Interleaving

    This enables a high priority request command to be inserted within a potentially long, lower priority data transfer that is already on the link.

  - Control Logic

    HT doesn't require intermediate control logic for direct processor-to-processor and processor-to-I/O connections [HYPER 2].

- PCI, PCI-X and PCI Express mapping
  Provides interfaces between HT and the PCI formats – making HT compatible with PCI devices. PCI is a computer expansion card interface format connecting sound, video and network cards to a motherboard. PCI was succeeded by PCI-X and then by PCI-Express. HT compatibility with these formats is a key feature as PCI is a standard for expansion card interfaces.

- AC operating mode
  Although not a feature to be implemented in processors-to-processor links, the AC mode is still important, making HT a more generic p2p technology. By switching to AC mode the link can achieve greater Signal Interconnect Distance, allowing HT to be implemented on larger physical systems. It automatically switches to the AC mode on the detection of Coupling Capacitors (which are used to connect two circuits allowing only the AC signal to pass through while the DC signal is blocked).

- Hot Plugging
  This feature is similar to plug-and-play; it allows the changing or adding of peripheral components that interact with the operating system while the system is active, while also informing the OS of the change. It allows HT devices to be installed and removed while the link is running. This is of particular use on systems where it's a nuisance or simply not feasible to restart when hardware needs to be installed or removed. A prime example where this feature will be utilized is on the server market where the system is constantly operational.

- Dynamic Link Clock/Width Adjustment
  This feature allows the CPU to dynamically change it clock rate and packet width. The aim of this is to reduce power consumption by lowering the clock rate or packet width if the CPU is running to fast for its current task. The CPU can change the clock rate to any of those supported by the link and it can reduce the bit size as low as needed.

3.5 - Summary

On top of its speed and bandwidth capabilities HT is a highly flexible product, which has the capability to replace the overlapping buses of previous systems into a single link. Its complete backwards capability with previous incarnations, compatibility with other technologies and the fact that it's a highly scalable product operable on large systems should make it a prevalent technology in the coming years and a generic solution for replacing buses on multiple systems.

4-Cache Coherency Protocols

4.1 - Introduction

This chapter covers cache coherency and the protocols that enforce it. The basics of cache coherency will be explained with the MESI protocol, MESI was designed for implementation on a FSB and is the basis of the MESIF and MOESI protocols, as such it's perfect to demonstrate cache coherence and the changes required in the crossover to p2p technology.

4.2 - Cache Coherency

Cache itself is small and fast memory linked to a CPU, it stores locations from main memory to reduce memory accesses for a CPU and to increase the CPUs overall speed.

On a system there may be multiple processors and caches. When multiple processors request a copy of the same data line to store on their caches it can lead to problems in keeping the data coherent throughout a system on all the caches.

Cache coherency refers to keeping data coherent throughout a system, and Cache Coherency protocols are the means by which the integrity of data is kept in the systems caches by storing it in specified states.

4.3 - MESI

MESI stands for the 4 states in which a memory location can be held in a cache, Modified, Exclusive, Shared and Invalid.

The Four States

Modified – The most recent copy of this memory location is present in this cache only; the copy in memory is out of date. 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 state. The copy in memory is up to date.

Invalid – This copy of the data is invalid.

System

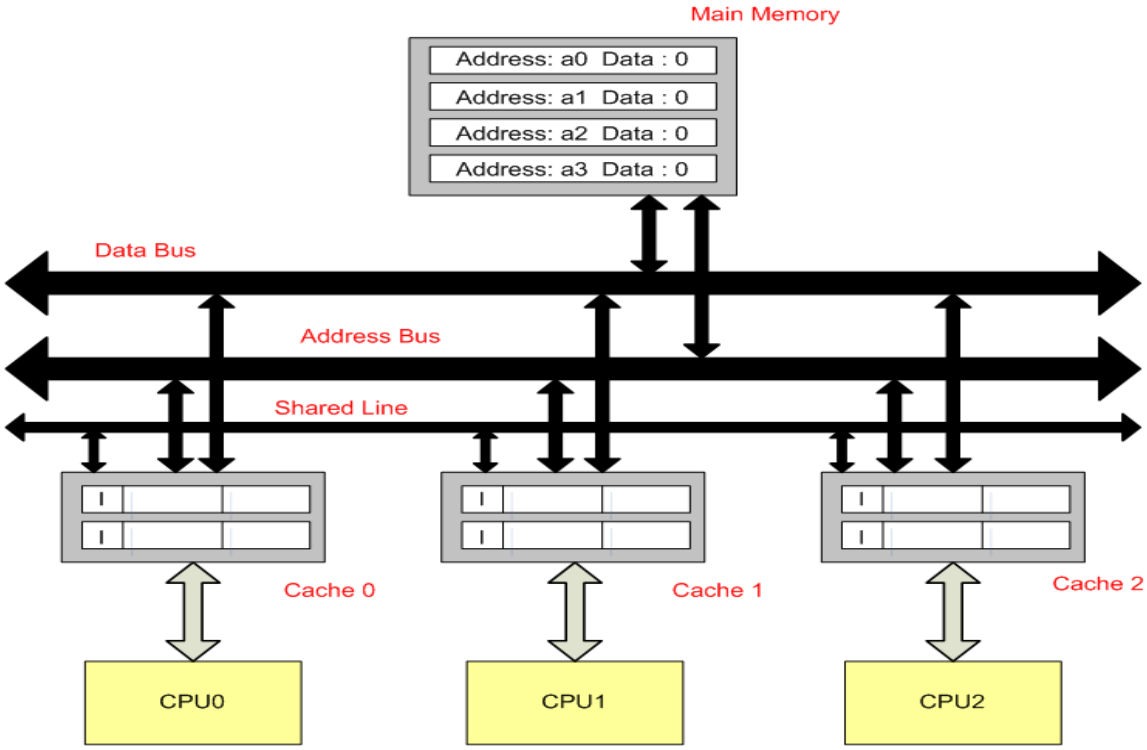


Fig 4.1 – Front Side Bus Architecture

As shown in **Fig 4.1** the protocol is being displayed on a 3-processor system, each processor has its own local cache and they are all connected to a single memory and each other via a Front Side Bus. All caches lines are initially in the Invalid state-I, meaning that they don't carry a valid copy of data, the caches are direct mapped meaning with all every location stored on cache line 0 and all odd stored on cache line 1.

In a bus-based system the caches are constantly snooping the bus. Snooping means a cache is checking the bus for accesses to addresses it has stored, this is so the cache can act appropriately if it detects another cache trying to read or write the data it has locally.

Requests for an address are put on the address bus, data is send on the data bus and other CPUs are informed if a data line is stored in a valid state on another cache by the shared line.

Demonstration

The following gives a run through the possible actions of a CPU and the resulting states a cache line is stored in as a result.

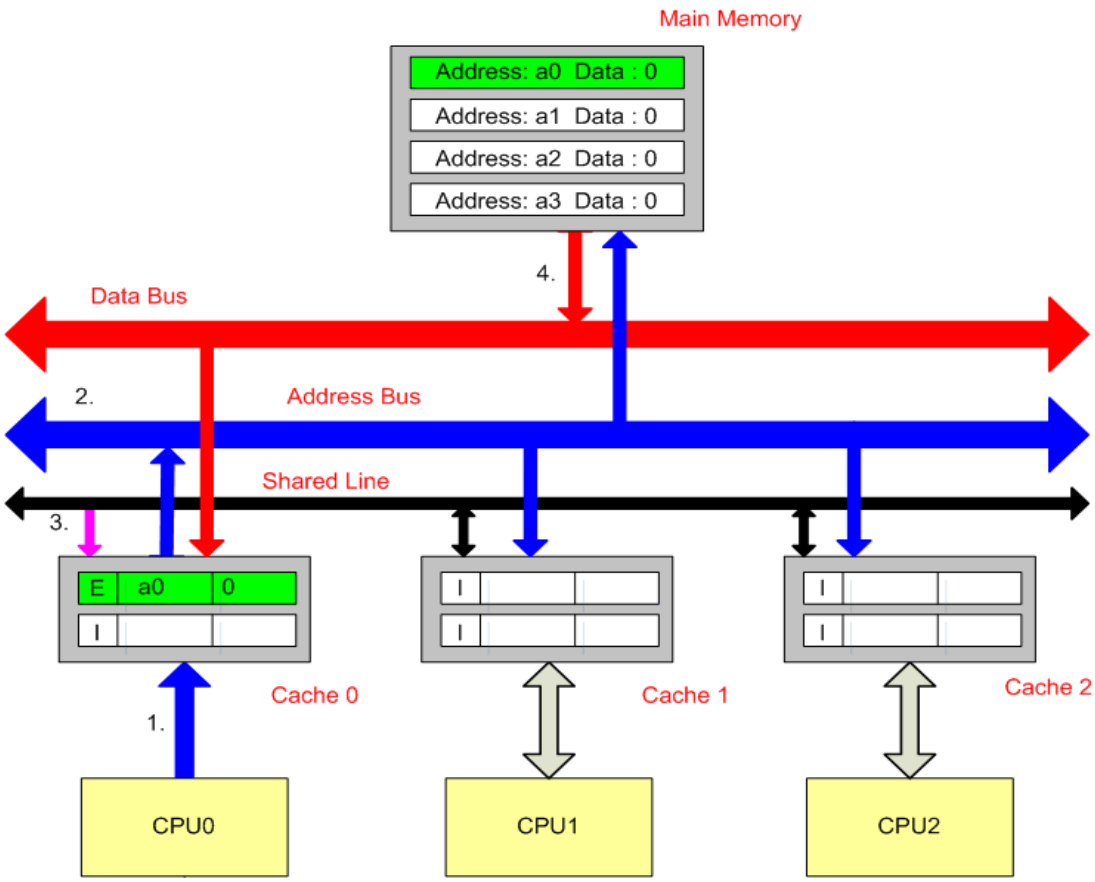


Fig 4.2 - CPU0 reads a0

CPU0 - Reads memory location a0.

1. CPU0 checks its local cache and doesn't find a valid copy of a0.
2. CPU0 sends a read request onto the address bus for a0.
3. No other CPU has a copy of it in its cache so none assert the shared line.
4. As main memory is not interrupted it pushes the data onto the data bus and CPU0 receives it. CPU0 stores the data in the Exclusive state-E, which means it has the only copy of a0 outside of memory and that its copy is coherent with memory.

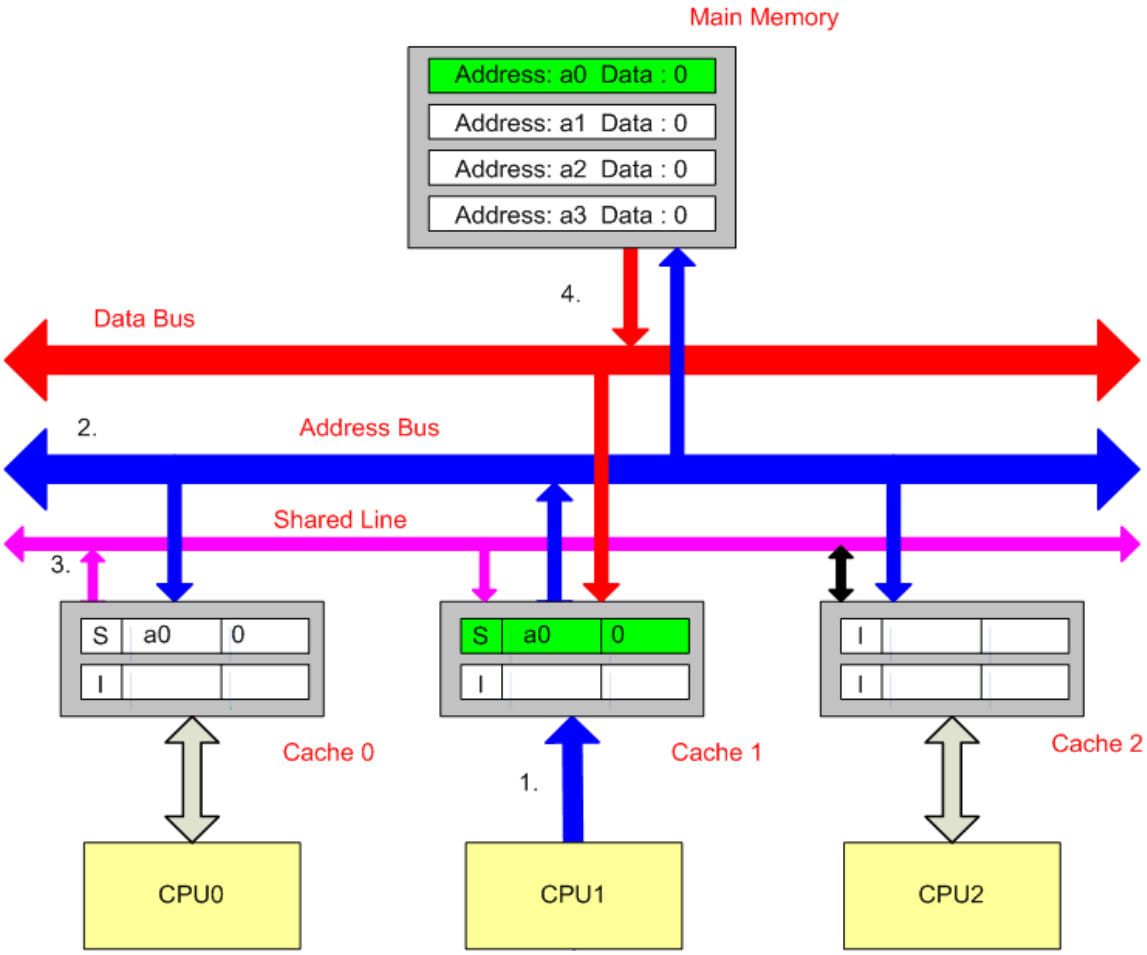


Fig 4.3 - CPU1 reads a0

CPU1 - Reads memory location a0.

1. CPU1 checks its local cache and doesn't find a valid copy.
2. CPU1 sends the read request for the address onto the address bus.
3. Cache0 has a copy, it asserts the shared line and puts its copy into the shared state.
4. Even though CPU0 has a valid copy it does not interrupt, main memory puts the data on the bus. CPU1 stores the data in the Shared state-S; as CPU1 received the assert on the shared line previously it goes to S instead of E. Shared means that one or more caches may hold a copy of a0 and that they are all coherent with main memory.

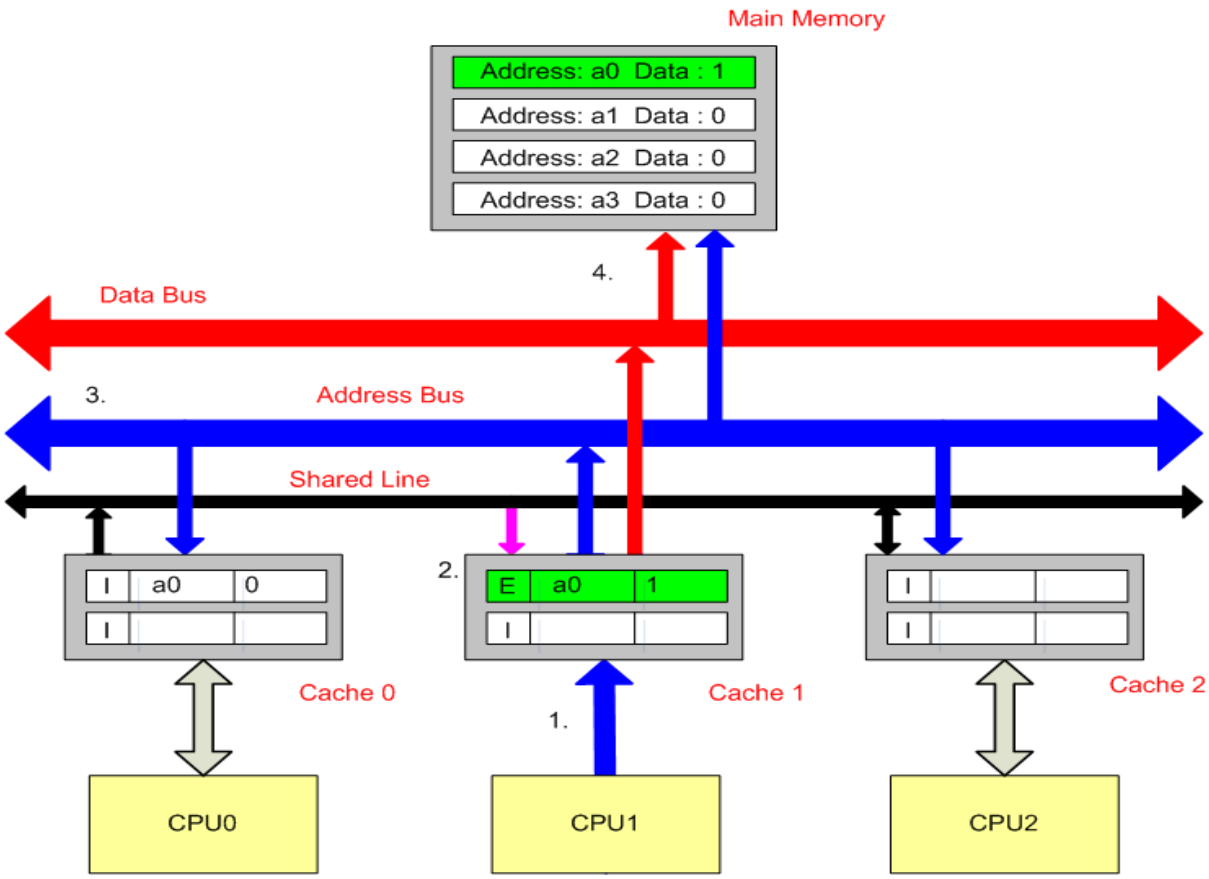


Fig 4.4 - CPU1 writes a0

CPU1 - Writes to memory location a0.

1. CPU1 checks its local cache and finds a valid copy in the shared state.
2. CPU1 now adds one to a0 and writes it going into the E state.
3. CPU1 sends an Invalidate message onto the address bus for a0, CPU0 puts its copy into the Invalid state-I.
4. At the same time CPU1 also flushes it copy of a0 to memory to keep it coherent.

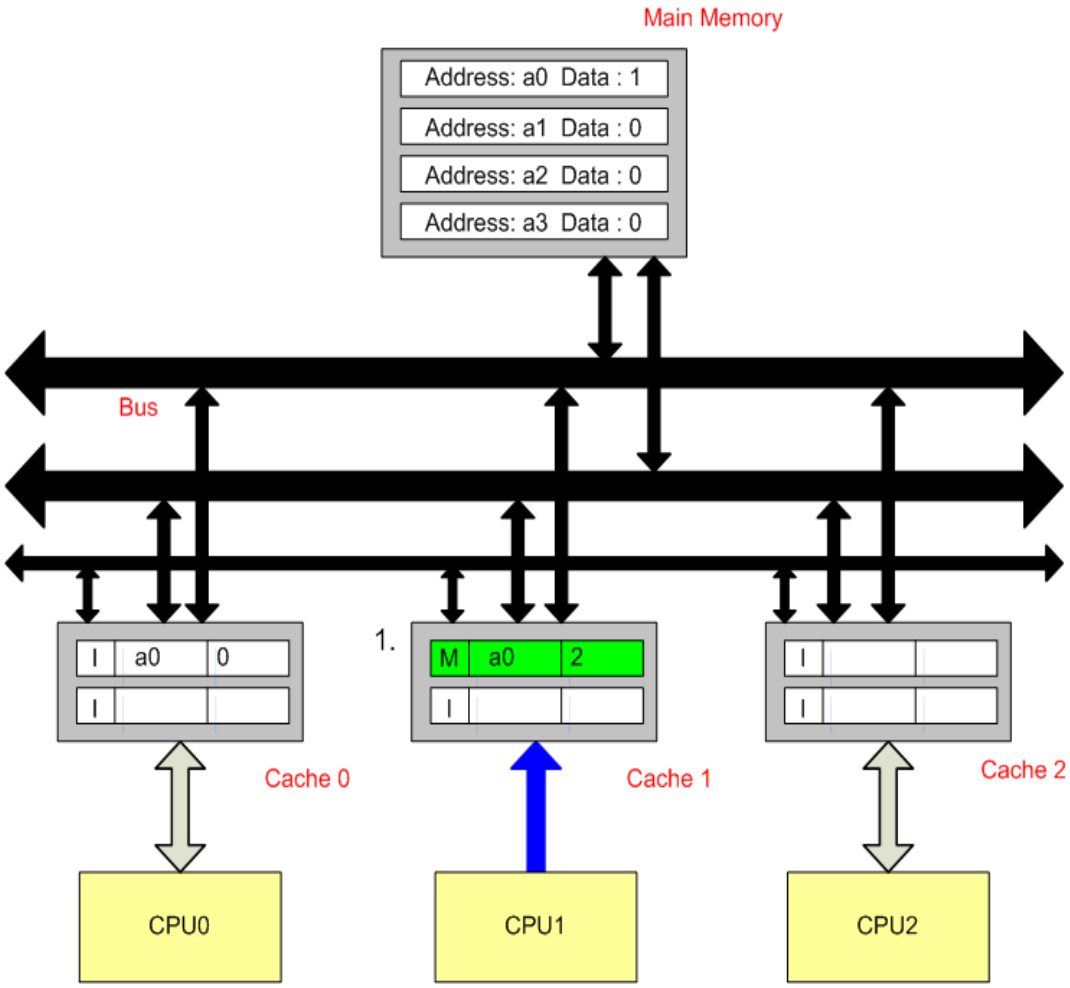


Fig 4.5 - CPU1 writes a0

CPU1 - Writes to memory location a0 again.

1. CPU1 checks its local cache and finds a valid copy in the E state. When a cache has the only valid copy of the data (i.e. the E state or M state) it can do a write straight away without accessing the bus.
   CPU1 writes to its copy of a0 and stores the data in the Modified state-M, which means it has the only valid copy in the entire system.

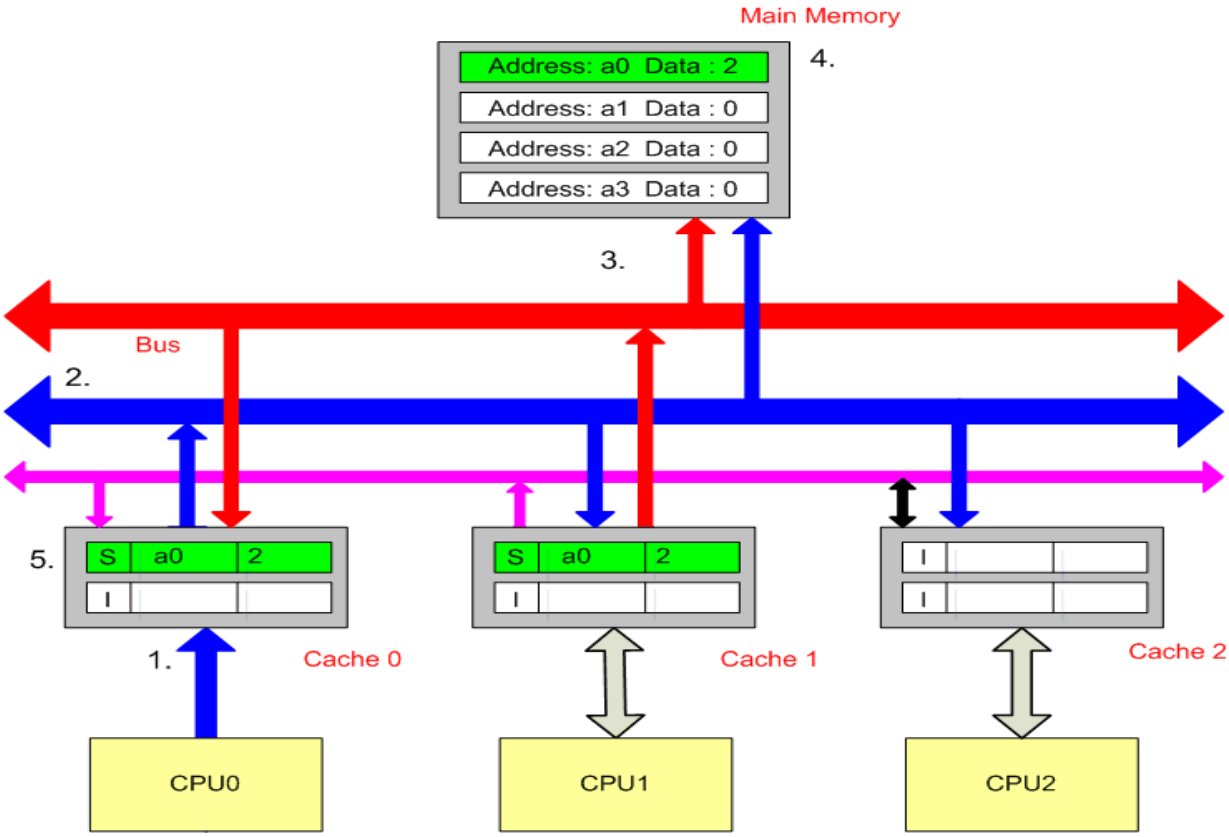


Fig 4.6 - CPU0 reads a0

CPU0 - Reads memory location a0.

1. CPU0 checks its local cache and doesn't find a valid copy.
2. CPU0 sends the read request for the address onto the address bus.
3. CPU1 is snooping the address and has a valid copy in the M state so it asserts the shared line, transfers its stored copy into the S state and puts its copy of the data onto the data bus.
4. CPU1 flushes a0 to memory to keep the data coherent now that it is transferring to the S state.
5. CPU0 reads the data from the data bus and goes into the S state as the shared line was asserted earlier.

Conflicts

Conflicts occur when 2 or more CPUs request an address at the same time. The FSB can only handle one request at a time, if two or more CPUs request the same data in parallel the requests will be automatically serialised, only one process at a time can access memory for data so conflicts can't occur.

4.4 – MOESI/MESIF Point-to-Point System

The virtual model used to demonstrate the MOESI and MESIF protocols consists of 5 nodes; each node in the system can communicate with the other 4 nodes via point-to-point links.
Every node in the system consists of a CPU with it own local

- Direct mapped 2 line cache-which means that all even address are stored in cache line 0 and all odd address lines are stored at cache line 1, addresses are assigned to a cache line by a simple modulus calculation done to the address.

  E.g. Address space 10 would go to location 0 as 10 mod 2=0.

- Main memory - memory in this system is based on a global address system with memory split evenly between the processors, if a CPU wishes to access memory on another processor it must request it.

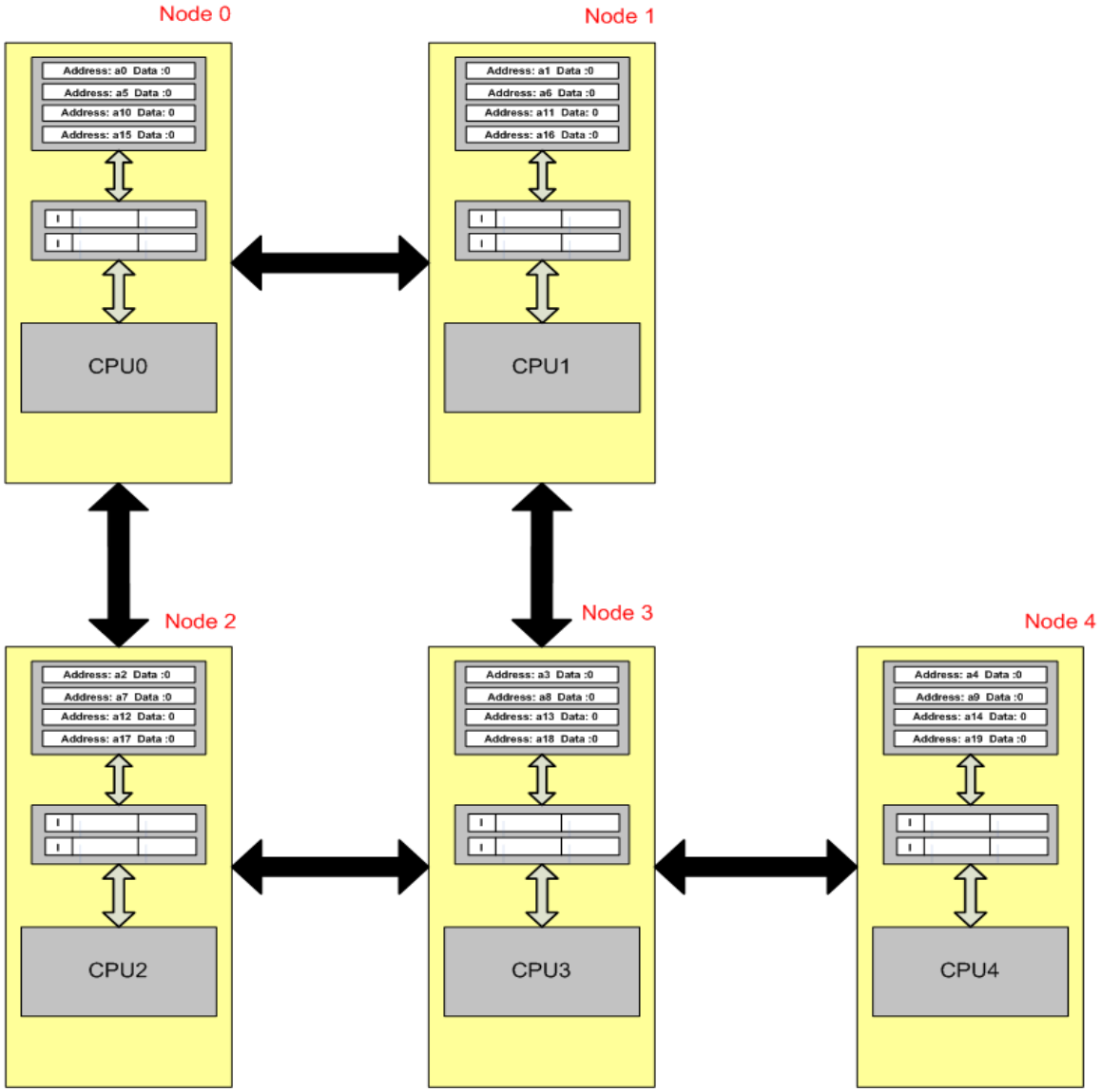


Fig 4.7 - p2p system

4.5 - MOESI

The main difference between MESI and MOESI is the Owned state-O. When a requested line is stored in the Modified state in the MESI protocol it must first update main memory if it wishes to transfer the data, the O state gets rid of this need. The O state is similar to a modified shared state, when a cache holds data in the M state and it receives a read request it changes to the O state, the data is then forwarded on to the requesting node, which stores the data in the Shared state.
The main benefit of the Owned state is that there is only a need to flush or transfer data to main memory when a line in the Owned state is being replaced.

States

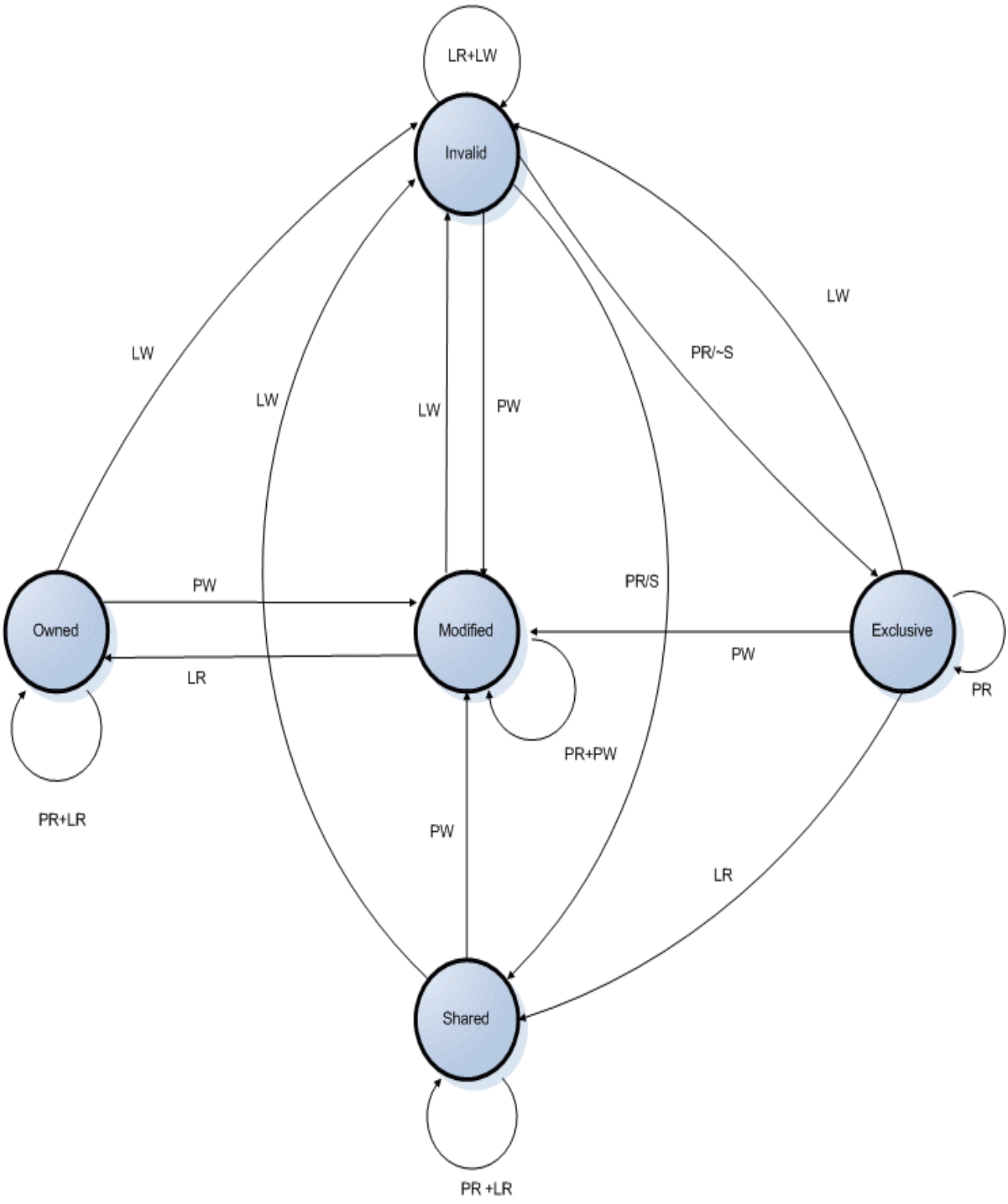
Modified – The most recent copy of this memory location is present in this cache only; the copy in memory is out of date or stale. The CPU in possession of this cache line may write new data to this line.

Owned – 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 out of date. Only one cache can hold a copy of the data in Owned at a time.

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 Owned state. The copy in memory is in date if the data is not cached on the system in the Owned state.

Invalid – This copy of the data is invalid.



PR = PROCESSOR READ LR = LINK READ
PW = PROCESSOR WRITE LW = LINK WRITE
S/~S = SHARED/NOT SHARED

Fig 4.8 - MOESI State transitions

Protocol Messages

There are 3 main messaging types split into virtual channels on the HyperTransport technology, requests, probes and responses, requests and responses are always from one node to another, whereas probes are broadcast to every node in the System.

Requests
- Read Request
- Non-posted Write Request
- Cache Block Commands
- Posted Write Request

Probes
- Broadcast Probe

Responses
- Read Response
- Probe Response
- Completion

Demonstration

The following demonstration shows the implementation of the MOESI protocol on a p2p system.

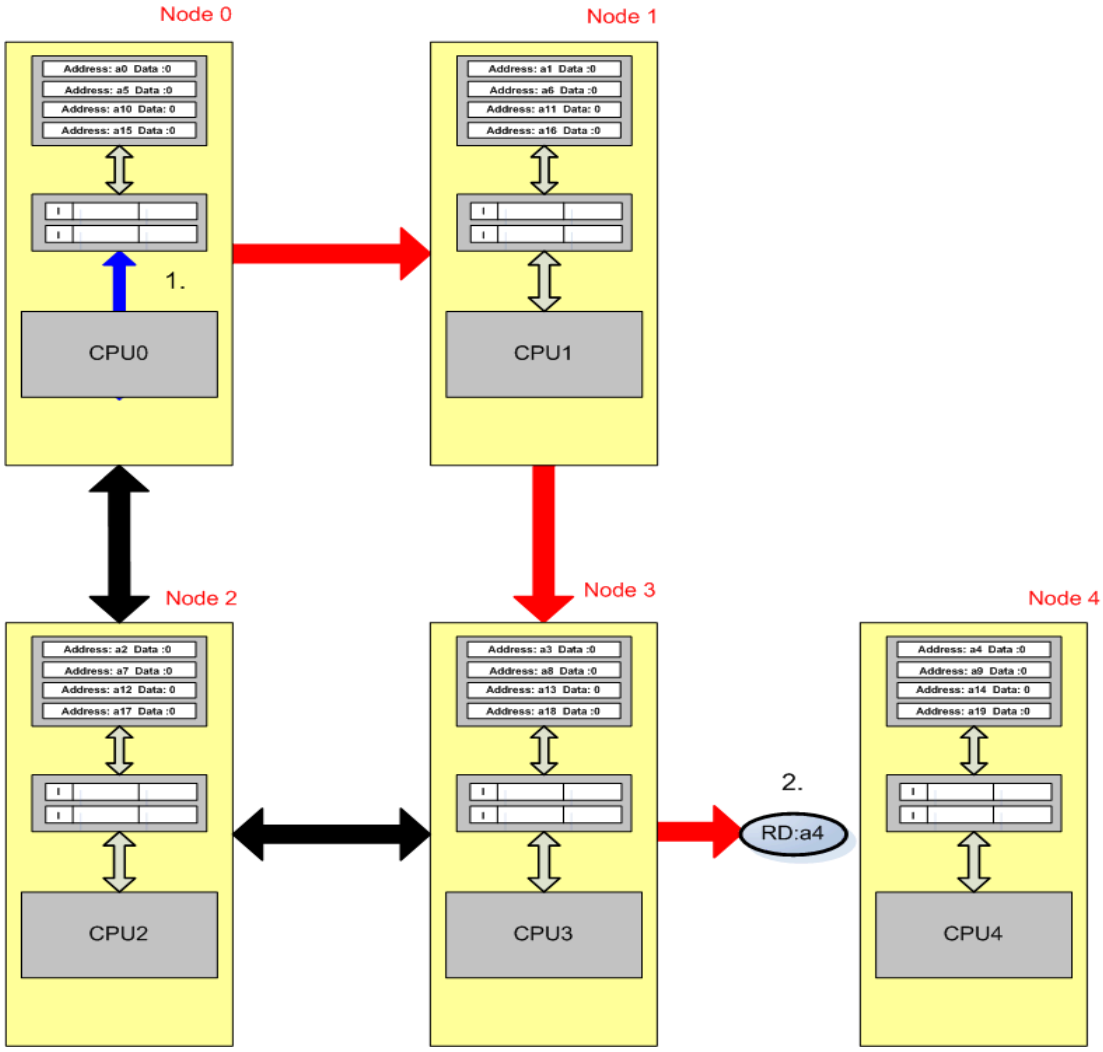


Fig 4.9 - CPU0 requests a4

CPU0 - Reads memory location a4.

1. CPU0 checks its local cache and doesn't find a valid copy; using the physical address it determines a4 is stored in Node 4.
2. CPU0 sends a read request to Node 4, the request is sent to Node 1 initially and the subsequent Nodes forward it onto Node 4 by checking the routing tables.

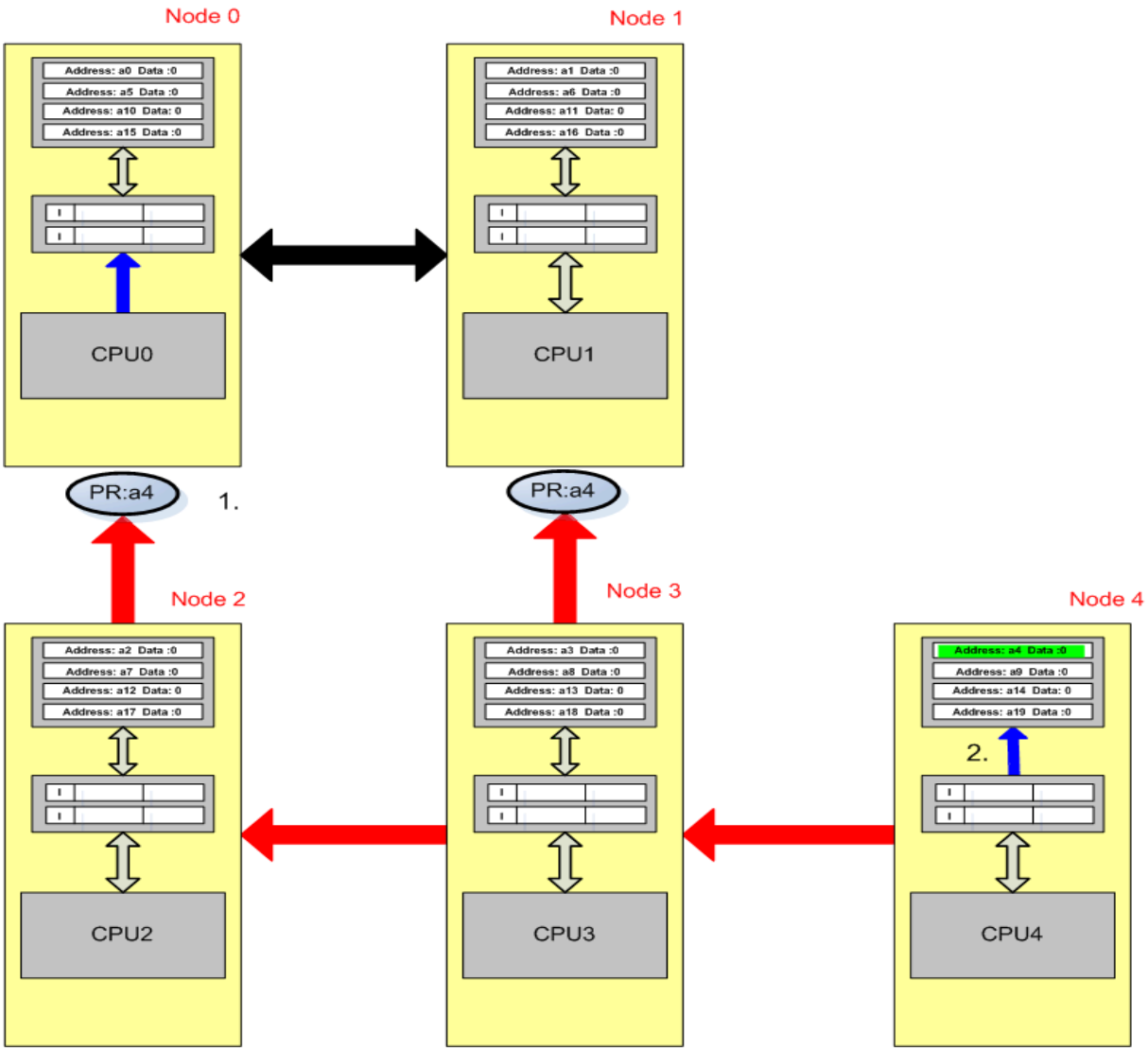


Fig 4.10 -CPU0 requests a4

CPU0 - Reads memory location a4.

1. On receiving a read request node 4 the home node broadcasts a probe for a4 to every node on the system including the source of the request Node 0 i.e. the source node.
   When it receives the request for a4 the home node will queue all other requests for access to a4 until node 0 is finished.
2. Node 4 simultaneously initiates a memory access for location a4 on it own local memory.

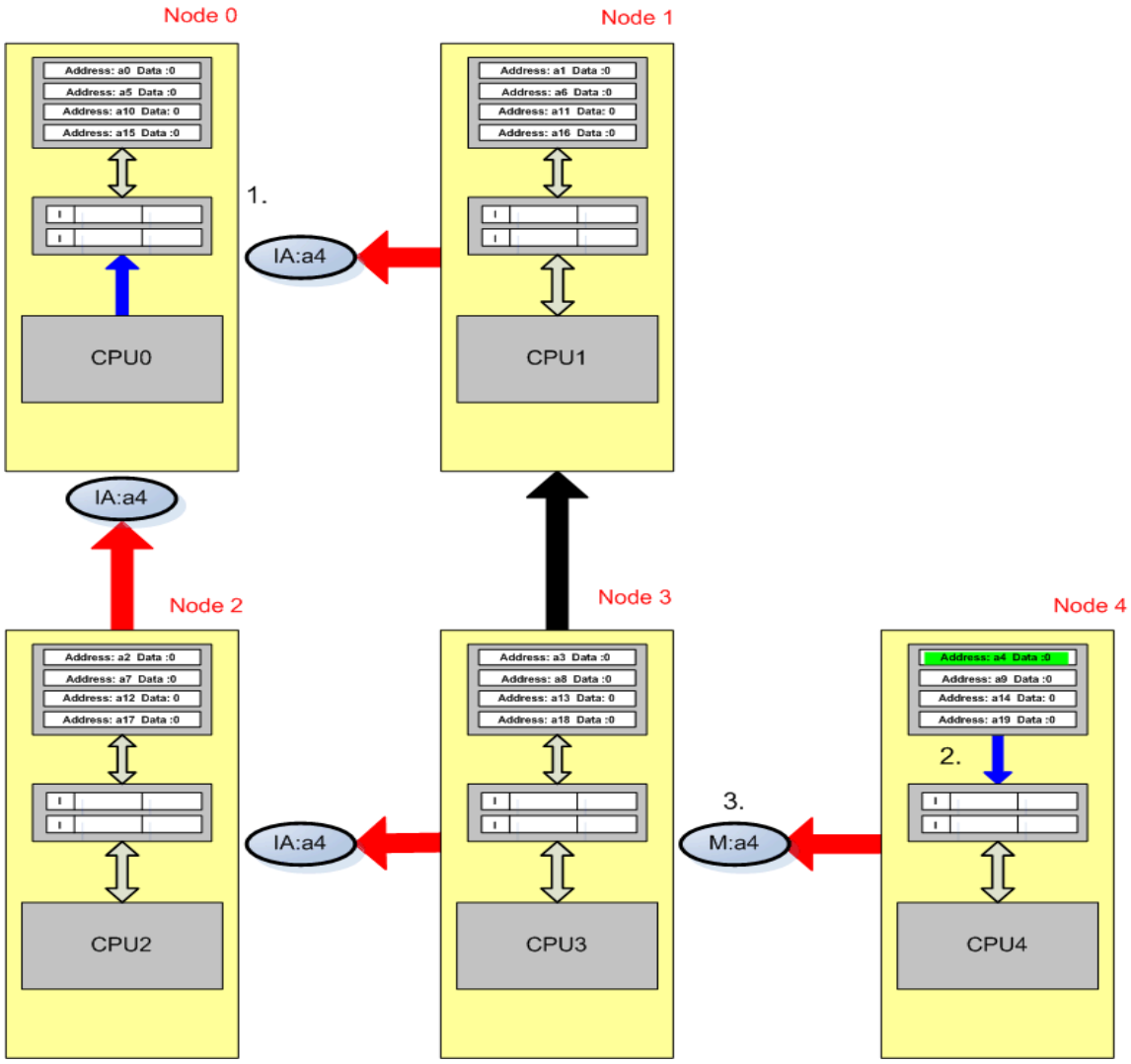


Fig 4.11 - Peer Nodes reply to source Node

CPU0 - Reads memory location a4.

1. Every node in the system on receiving the probe from the home node replies with an Invalid Acknowledge message (probe response), as none have a valid version of the data stored in their caches.
   A node will only reply with a read response if it has the data stored in Modified or Owned. The reason a node does not transfer the data if it has a copy stored in Exclusive or Shared is that a cache line stored in E or S is coherent with main memory, the source node will eventually receive the data from the home nodes memory access.
2. The source node must receive an acknowledgement from every node in the system (a node may consist of several components but all component responses are combined into one reply) and the data from memory before it continues.
   The home node initiates a memory access with every request it receives and this access cannot be cancelled, so even if the data is found on a peer nodes cache the source node must wait until it receives the copy from memory, which is a serious speed penalty on this protocol.

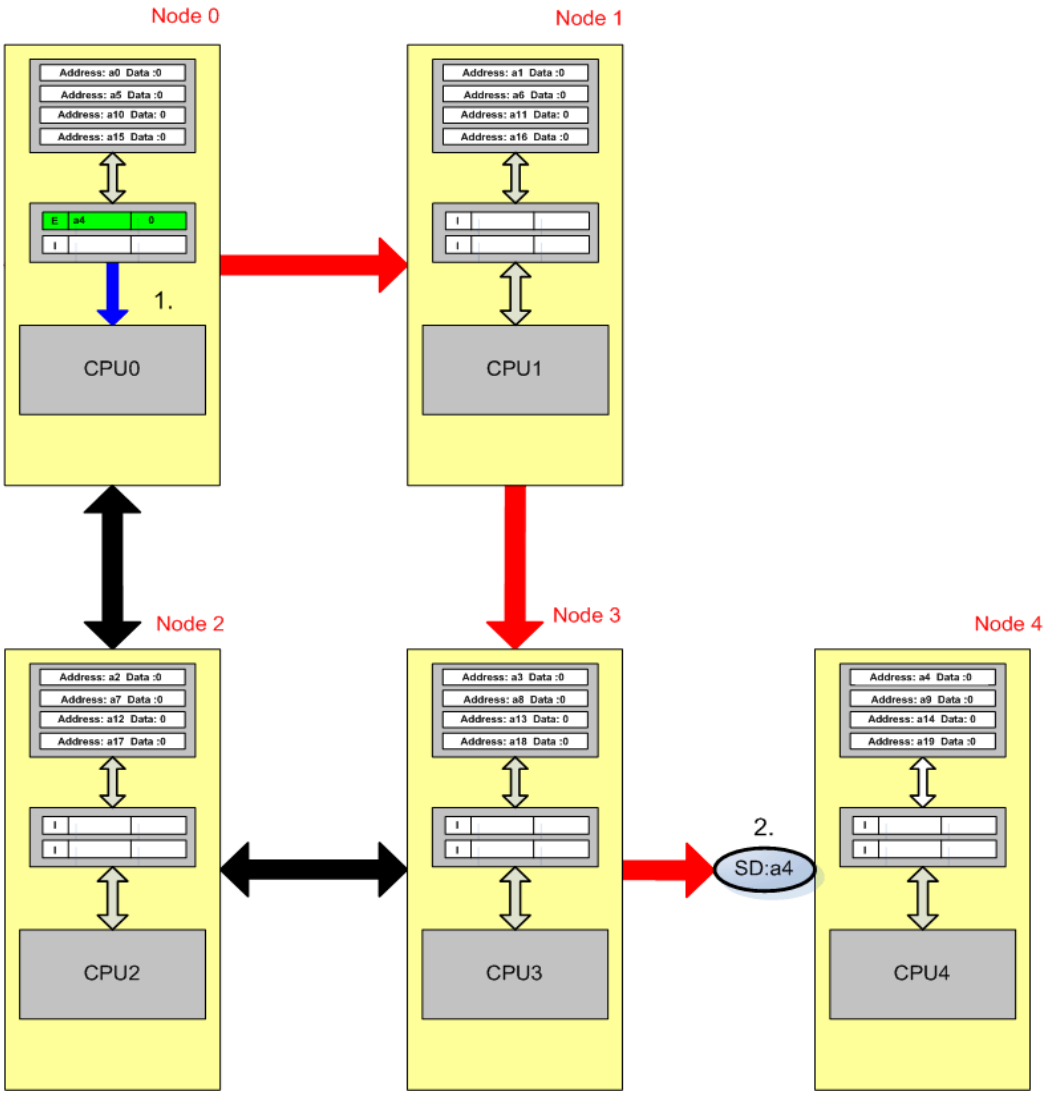


Fig 4.12 - CPU0 sends source done message

CPU0 - Reads memory location a4.

1. Node 0 on receiving all its replies finds that no other cache has a valid copy of the data; it uses the copy of data received from the Home Node and goes to the Exclusive state-E, it now has the only valid copy of data outside memory.
2. Node 0 has to let the Home Node know its finished so other processes can access the data, it sends a source done message to the Home Node allowing other process to access the data.

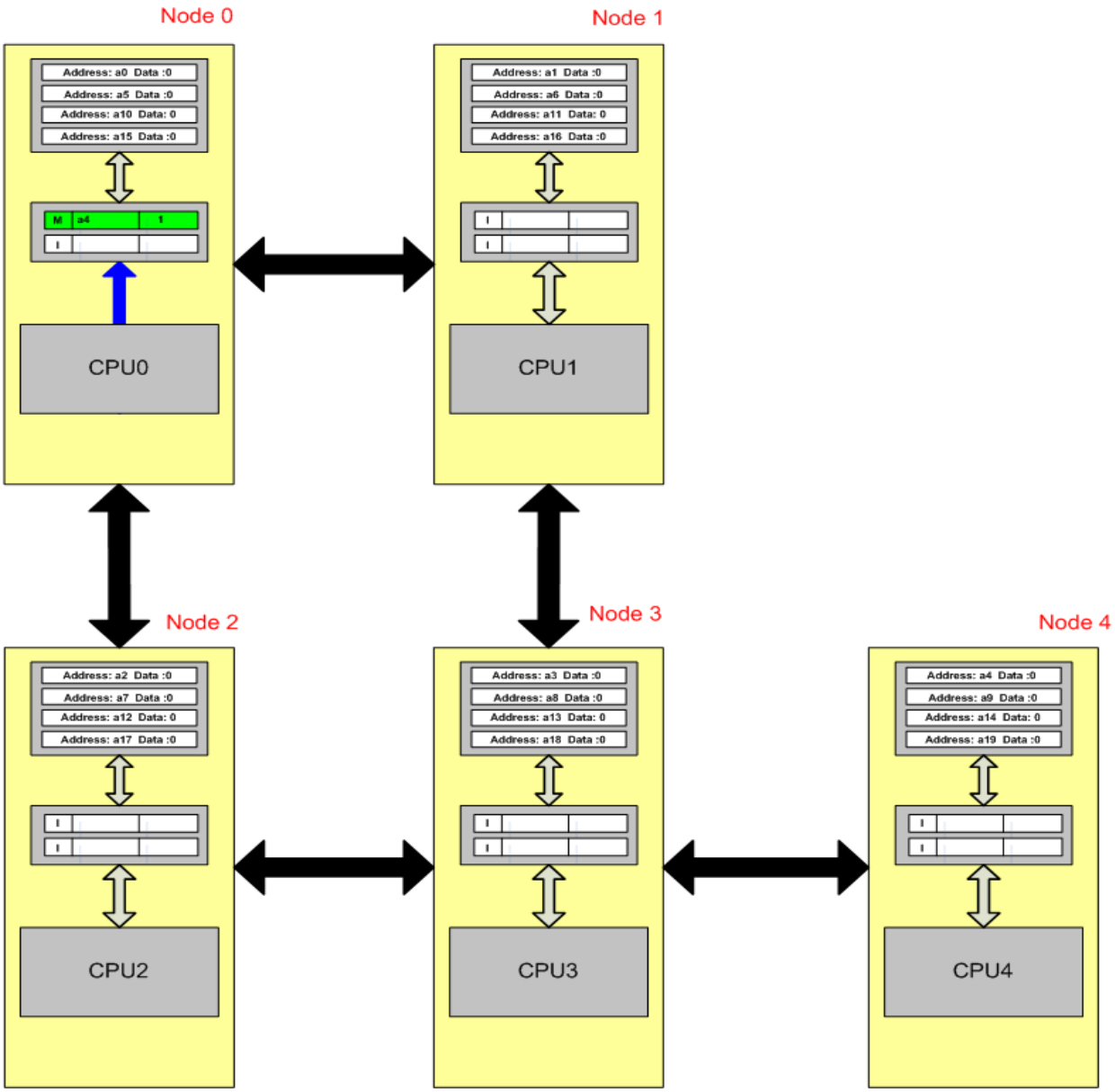


Fig 4.13 - Local write

CPU0 - Writes to memory location a4.

1. As CPU0 has a4 stored in exclusive it is able to do a straight write as no other node has a copy.

CPU1 - Reads memory location a4.

CPU1 checks its cache for a valid copy, not finding one its sends a read request to the home node.

The home node then sends out a probe broadcast for a4.

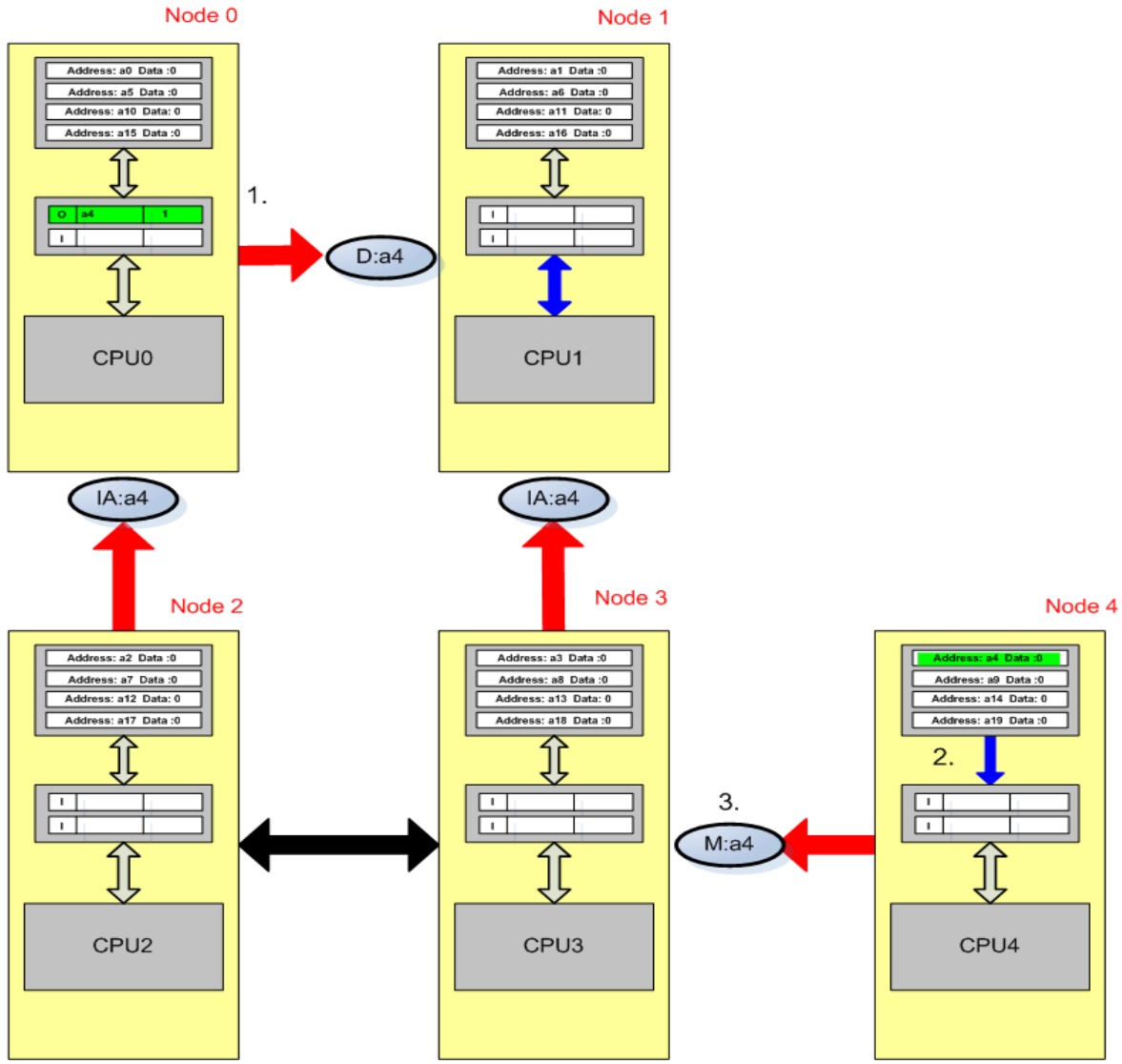


Fig 4.14 - Replies to probe broadcast

CPU1 - Reads memory location a4.

1. Node 0 on receiving the probe broadcast from the Home Node replies with the data (as it has it stored in the M state) to source node 1, on sending the data node 0 stores its copy of a4 in the Owned state as it is no longer holds the sole copy of the data.
   If this where the MESI protocol it would have to flush the data to the home nodes memory, the key benefit of the Owned state is that it reduces memory flushes, by allowing multiple caches to hold modified copies of data.
   A cache, which holds it in the O state, is in charge of supplying the data to any further requesting Nodes.
2. The Home Node must complete its memory access.
3. The Home Node on finishing its access sends the data to the source node, the data from memory will be discarded as a more recent version was found on node 0.

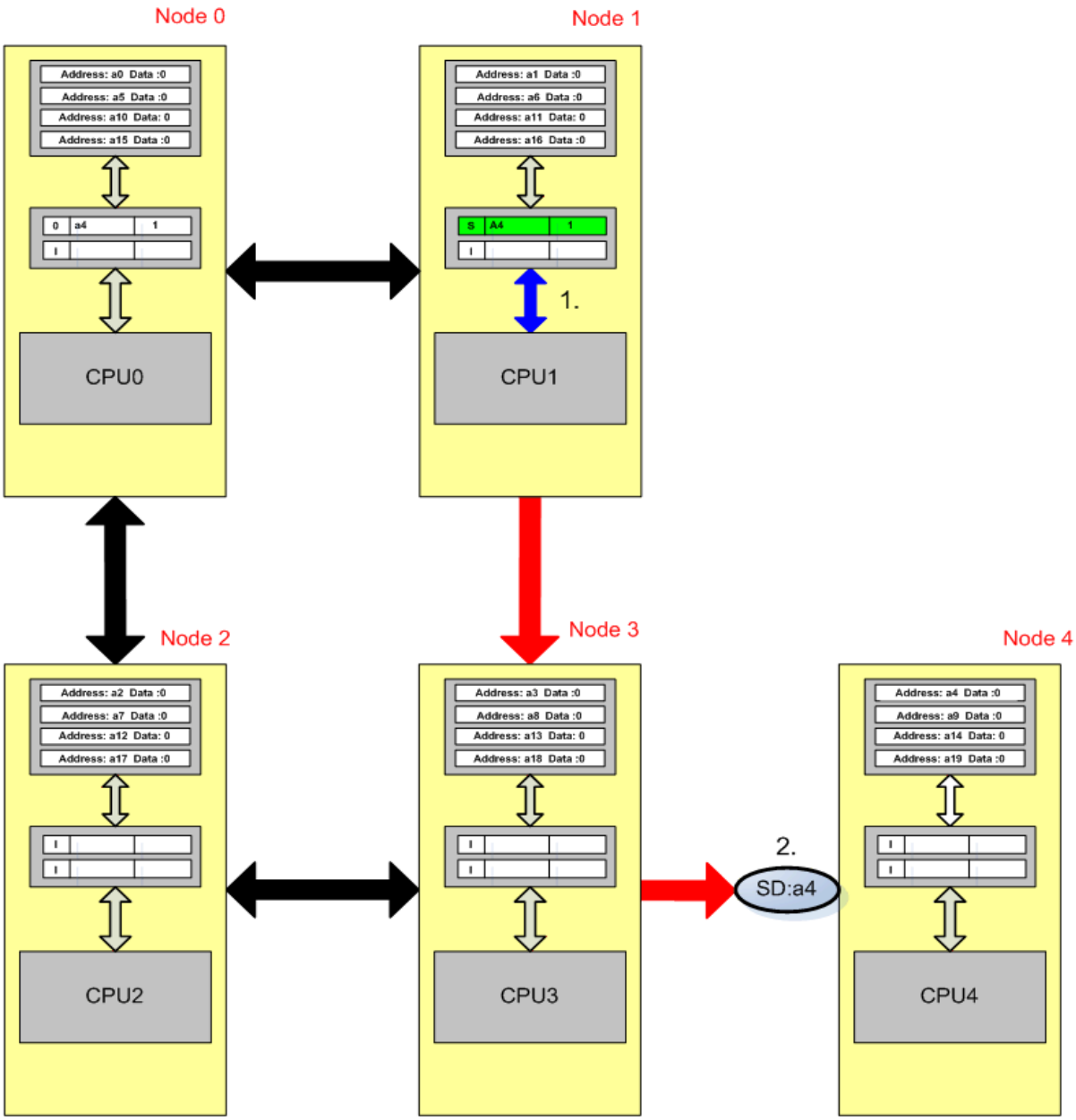


Fig 4.15 Source Done Message

CPU1 - Read memory location a4.

1. On receiving all its responses Node1 takes the value it received from Node 0 and goes to the S state.
2. The source Node sends a source done message to the Home Node to release the lock on a4.

Conflicts

Conflict resolution in the MOESI protocol uses the same principal as the MESI protocol and serialises requests, when two nodes request the same address the home node queues up the requests and deals with them one at a time. There is no way they can intrude on each other as they are completely serialised.

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

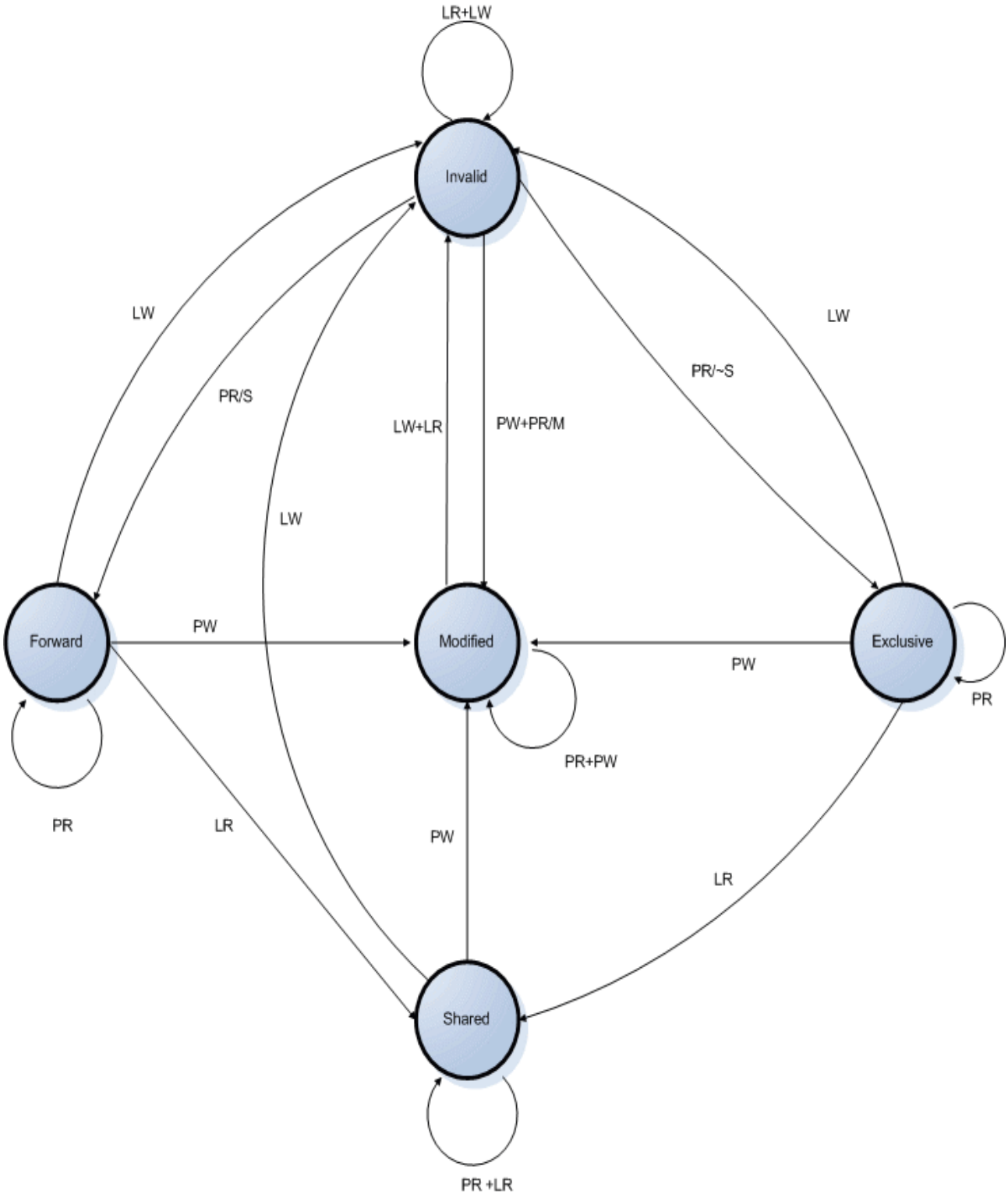
Modified – The most recent copy of this memory location is present in this cache only; the copy in memory is out of date or stale. The CPU in possession of this cache line may write new data to this line.

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.



PR = PROCESSOR READ
PW = PROCESSOR WRITE
LR = LINK READ
LW = LINK WRITE
LINES CURRENT STATUS
S/~S = SHARED/NOT SHARED
M=MODIFIED

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.

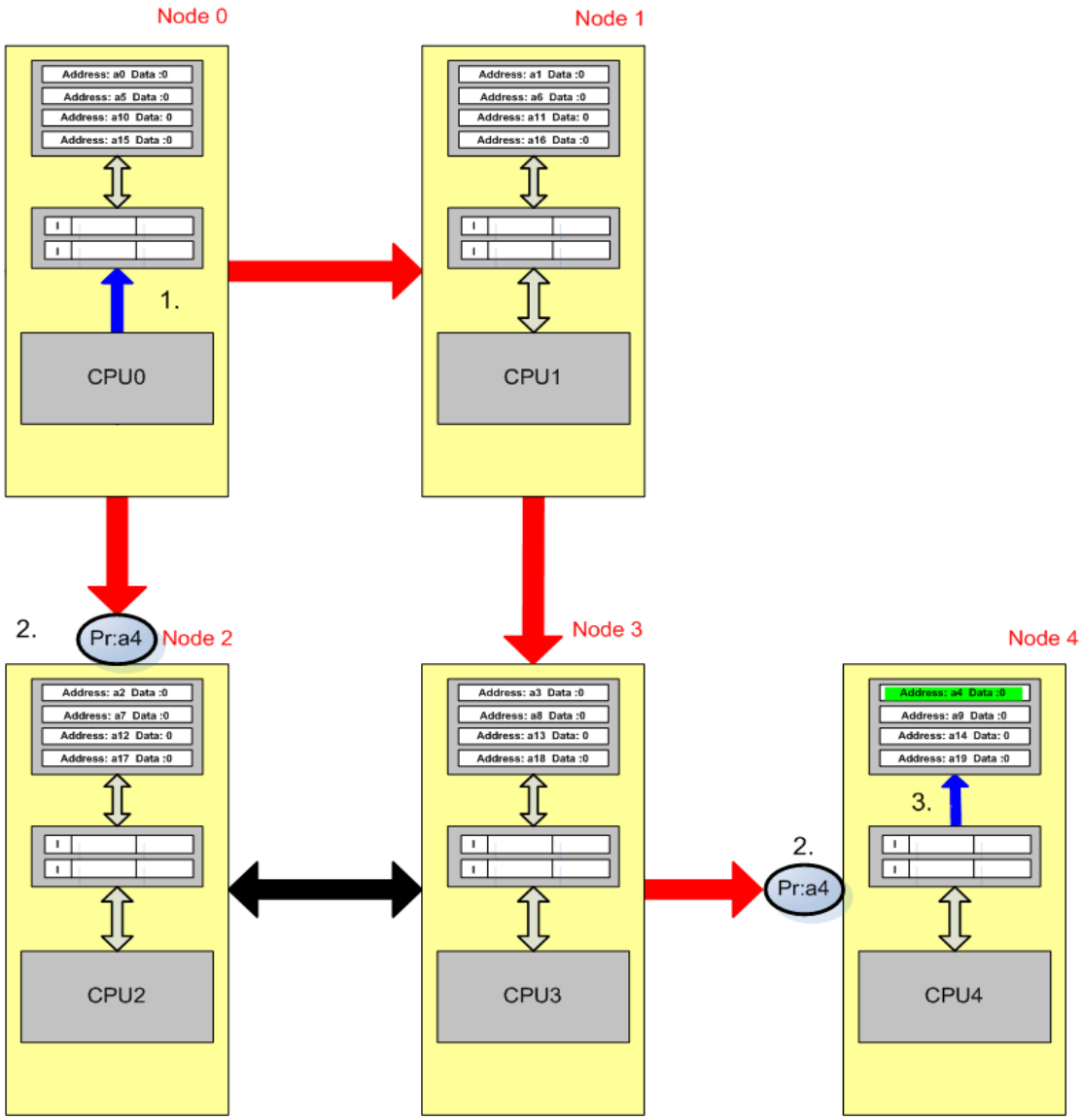


Fig 4.17 - Probe a4

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

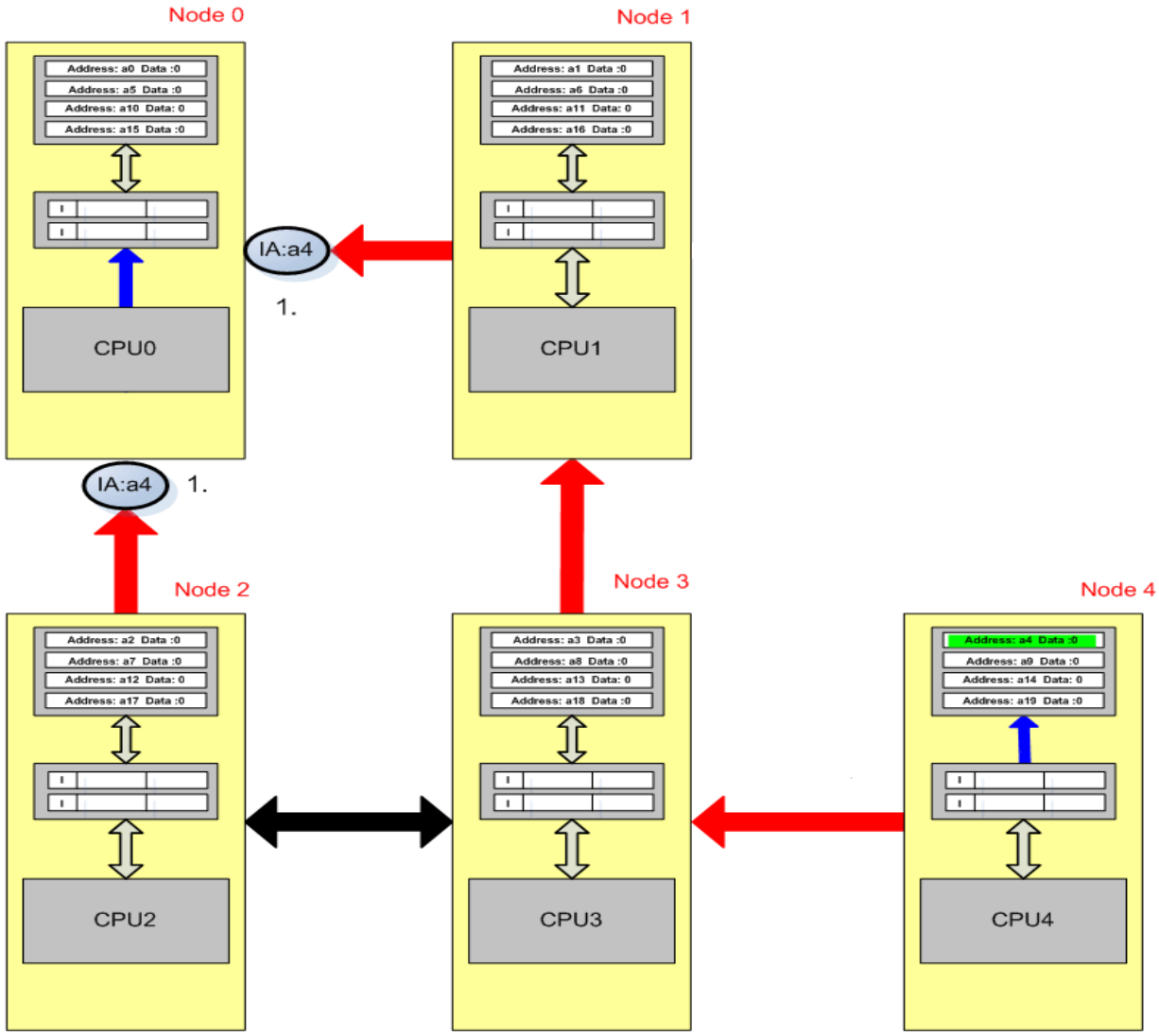


Fig 4.18 - Peer nodes reply to source node

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.

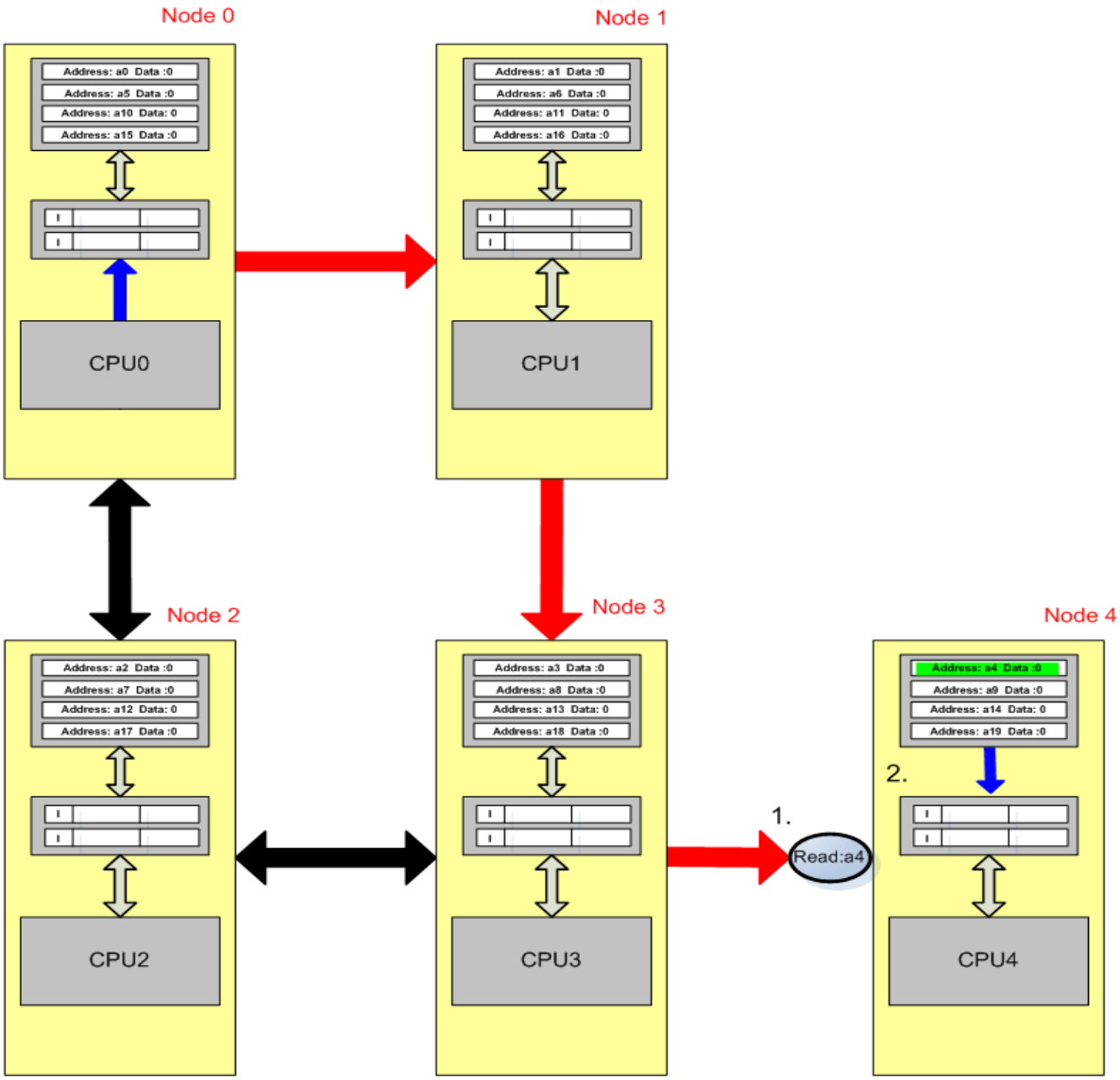


Fig 4.19 - Read Request

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.

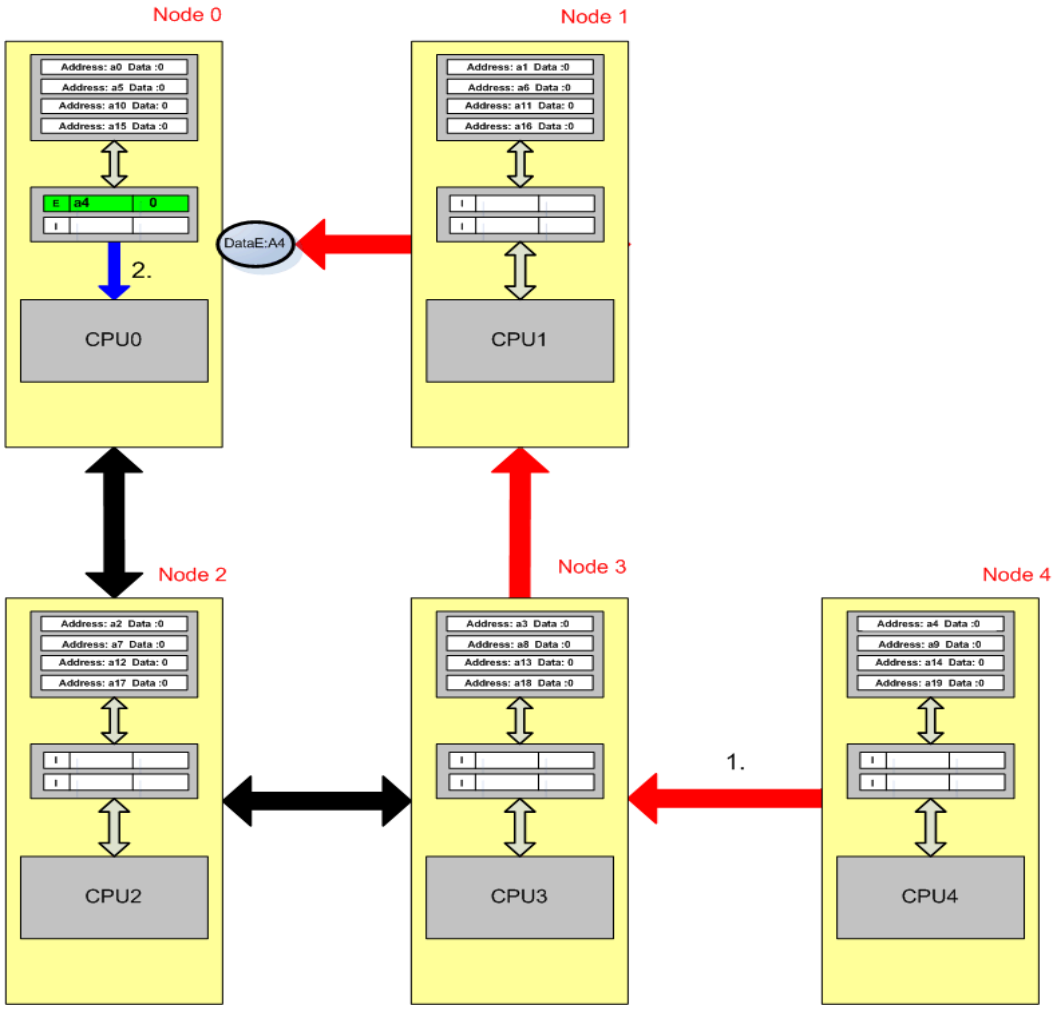


Fig 4.20 - DataE message

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.

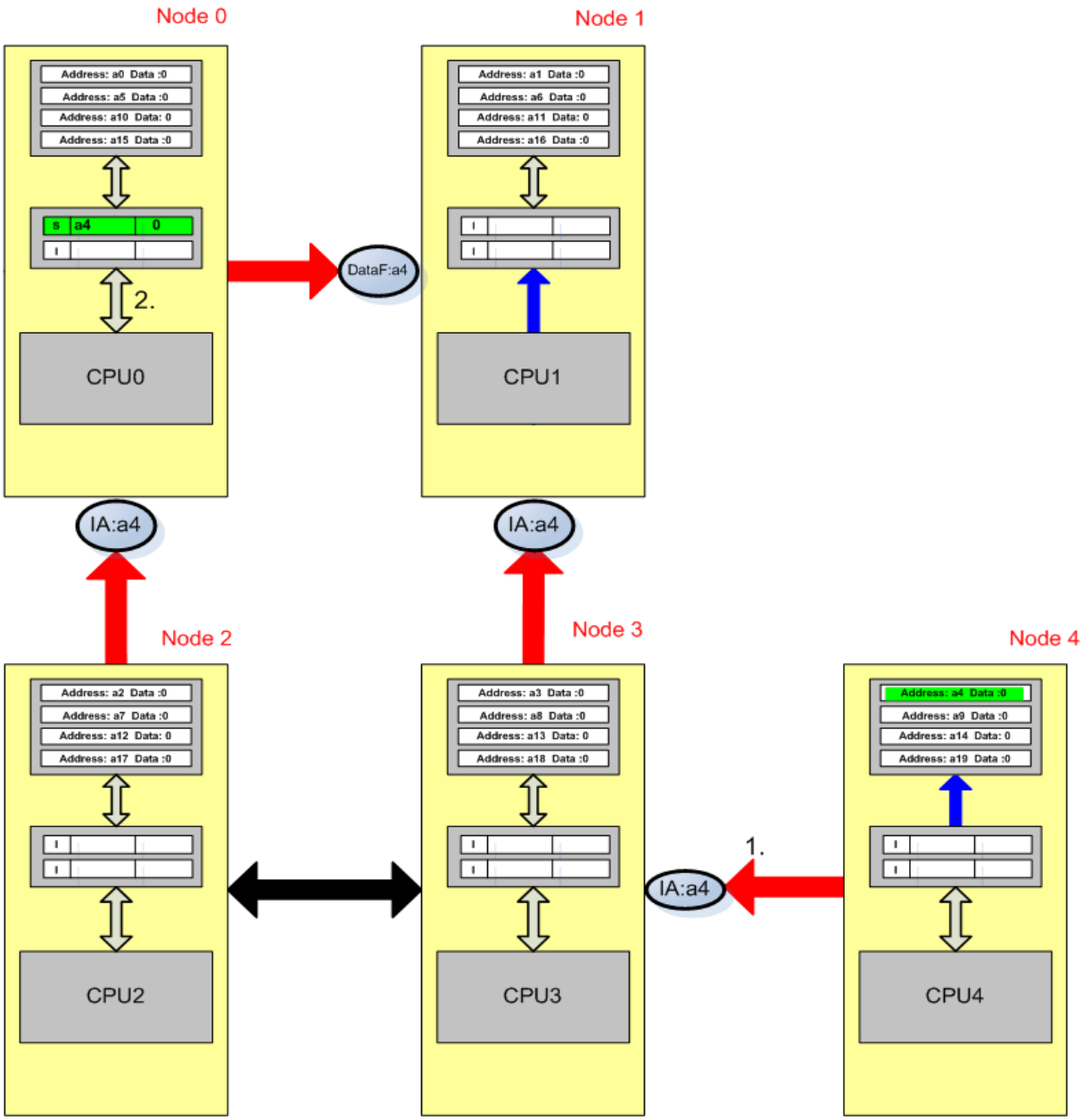


Fig 4.21 - Probe responses

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.

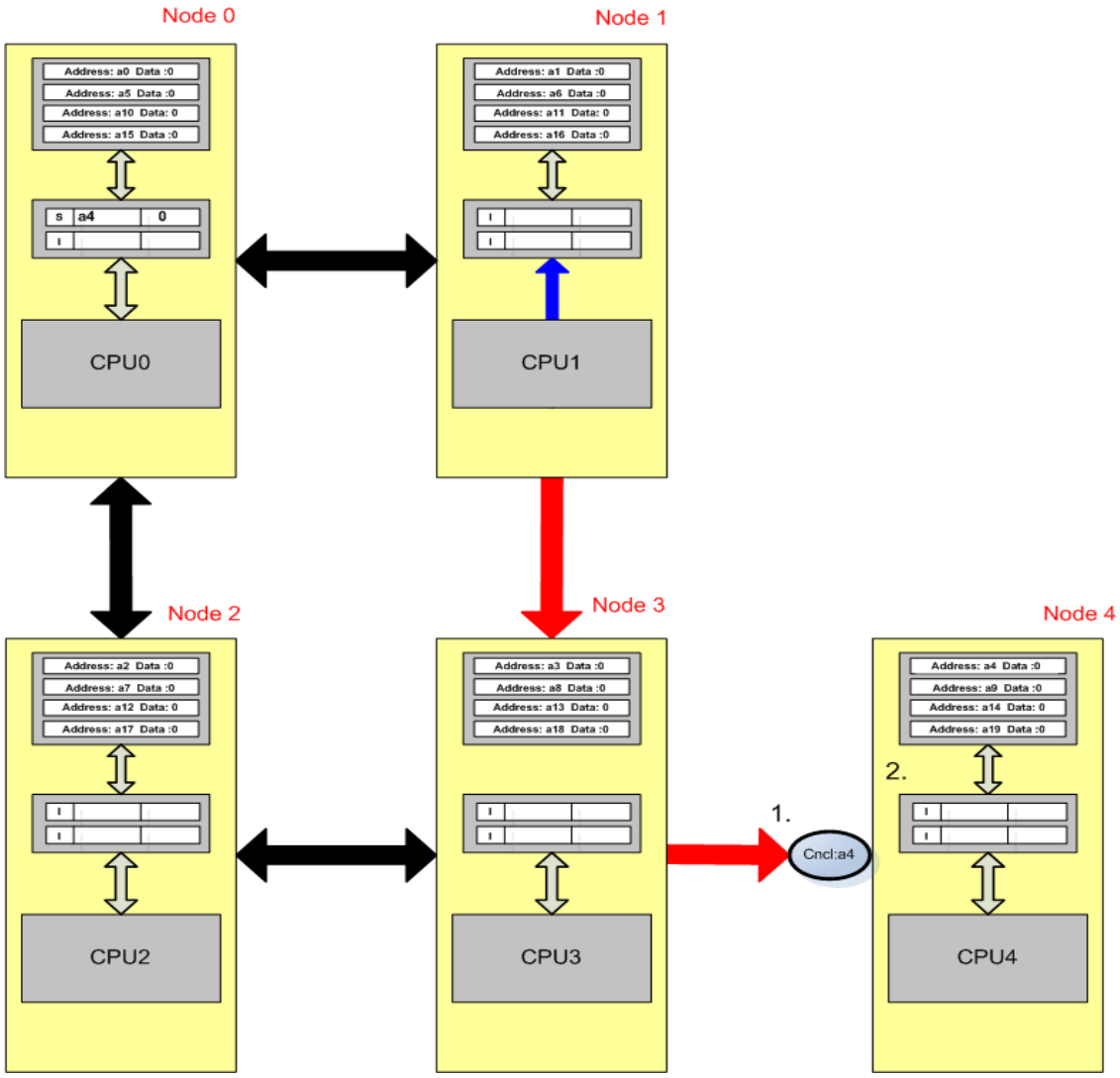


Fig 4.22 Cancel Message

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

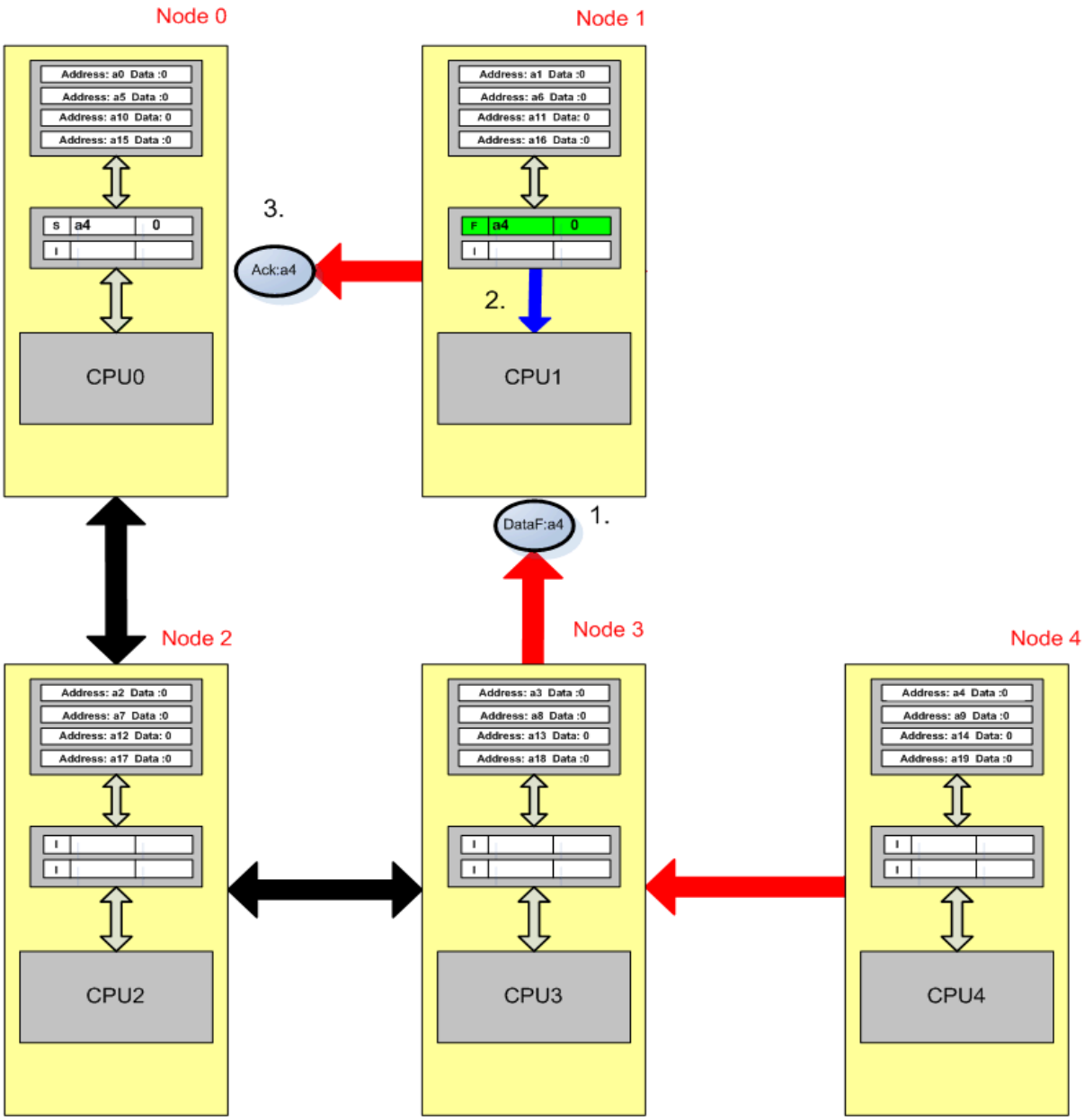


Fig 4.23 - DataF Confirmation

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.

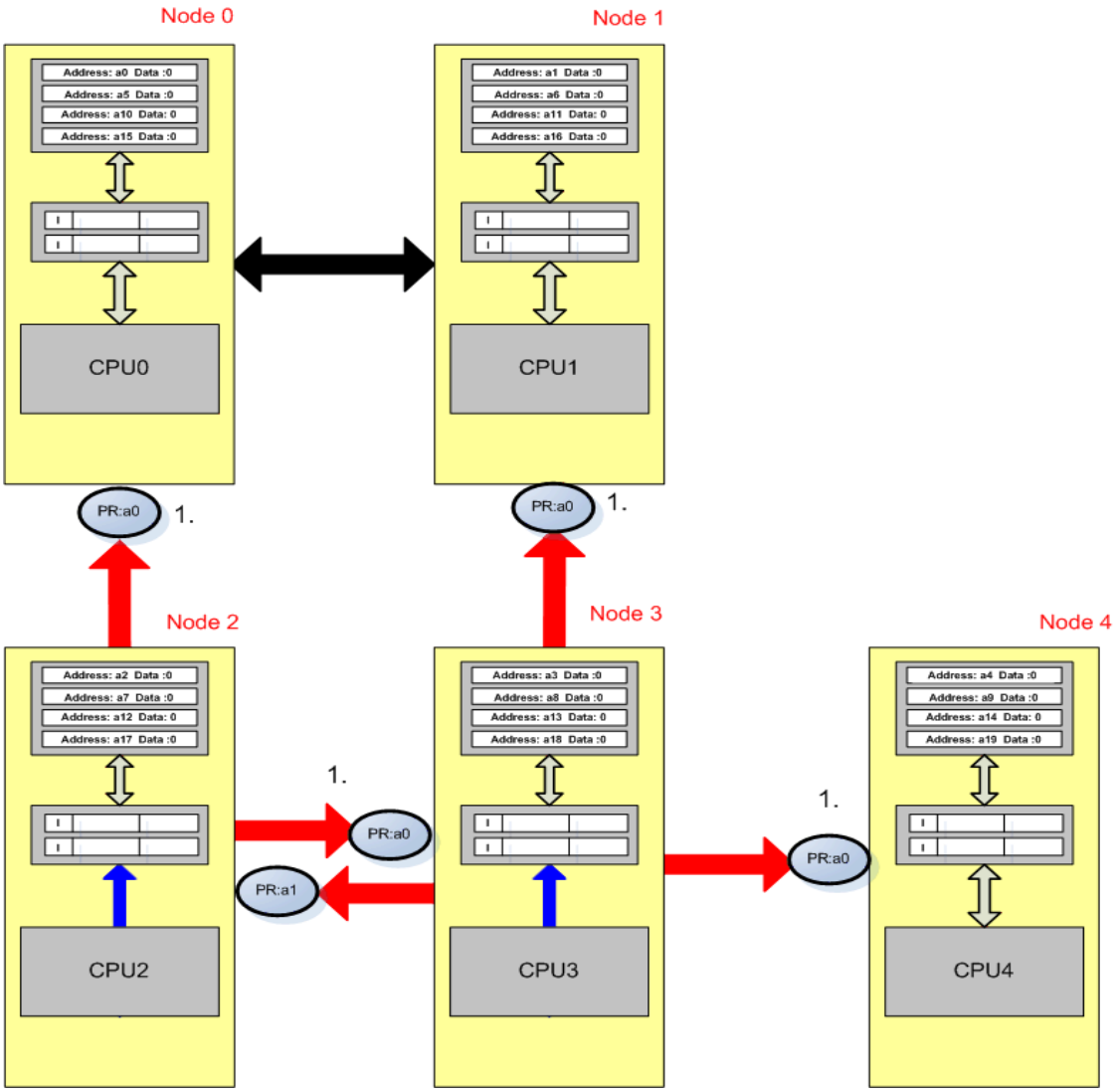


Fig 4.24 – CPU2 and CPU3 broadcast probes

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

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

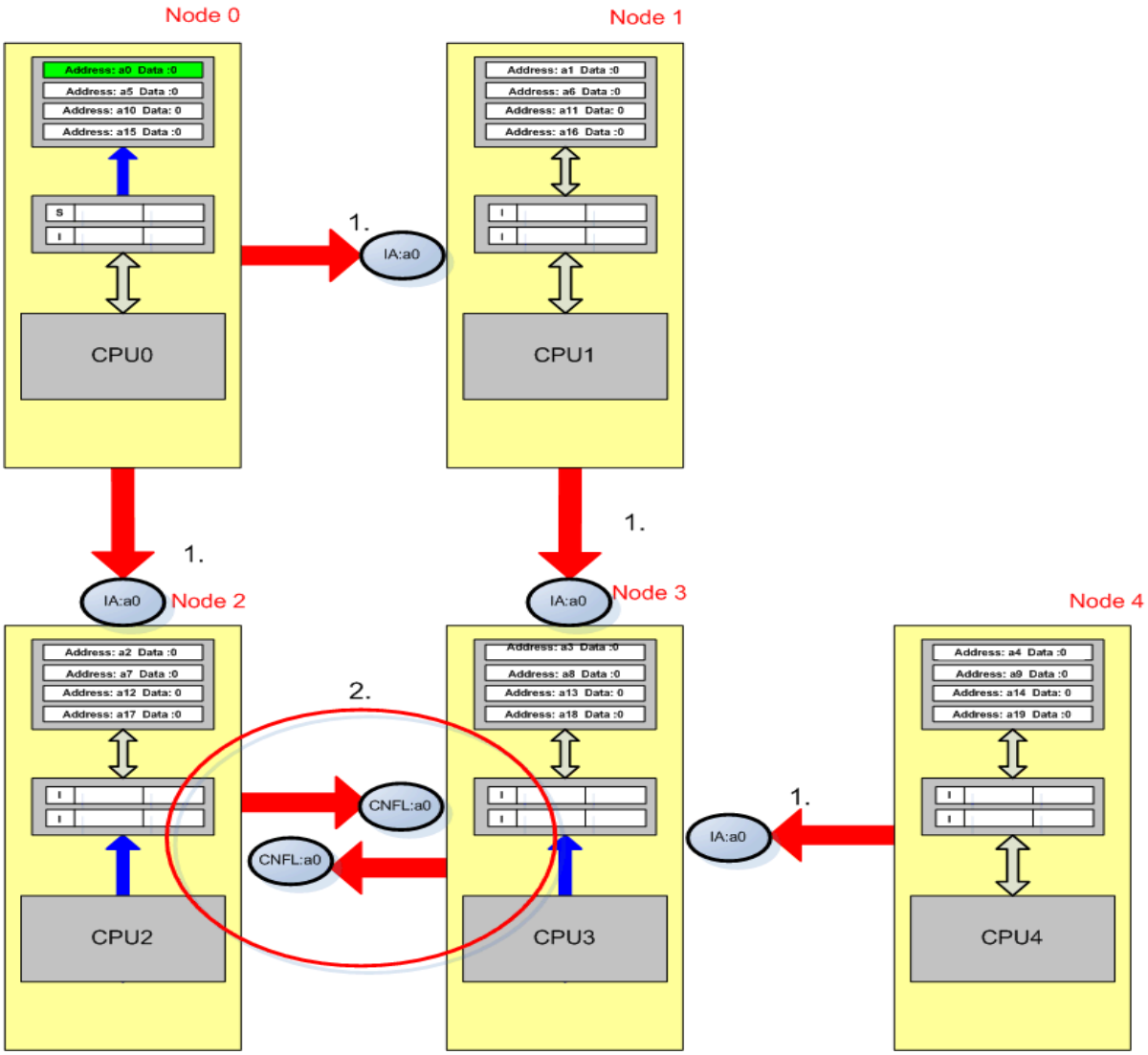


Fig 4.25 - Conflict replies

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

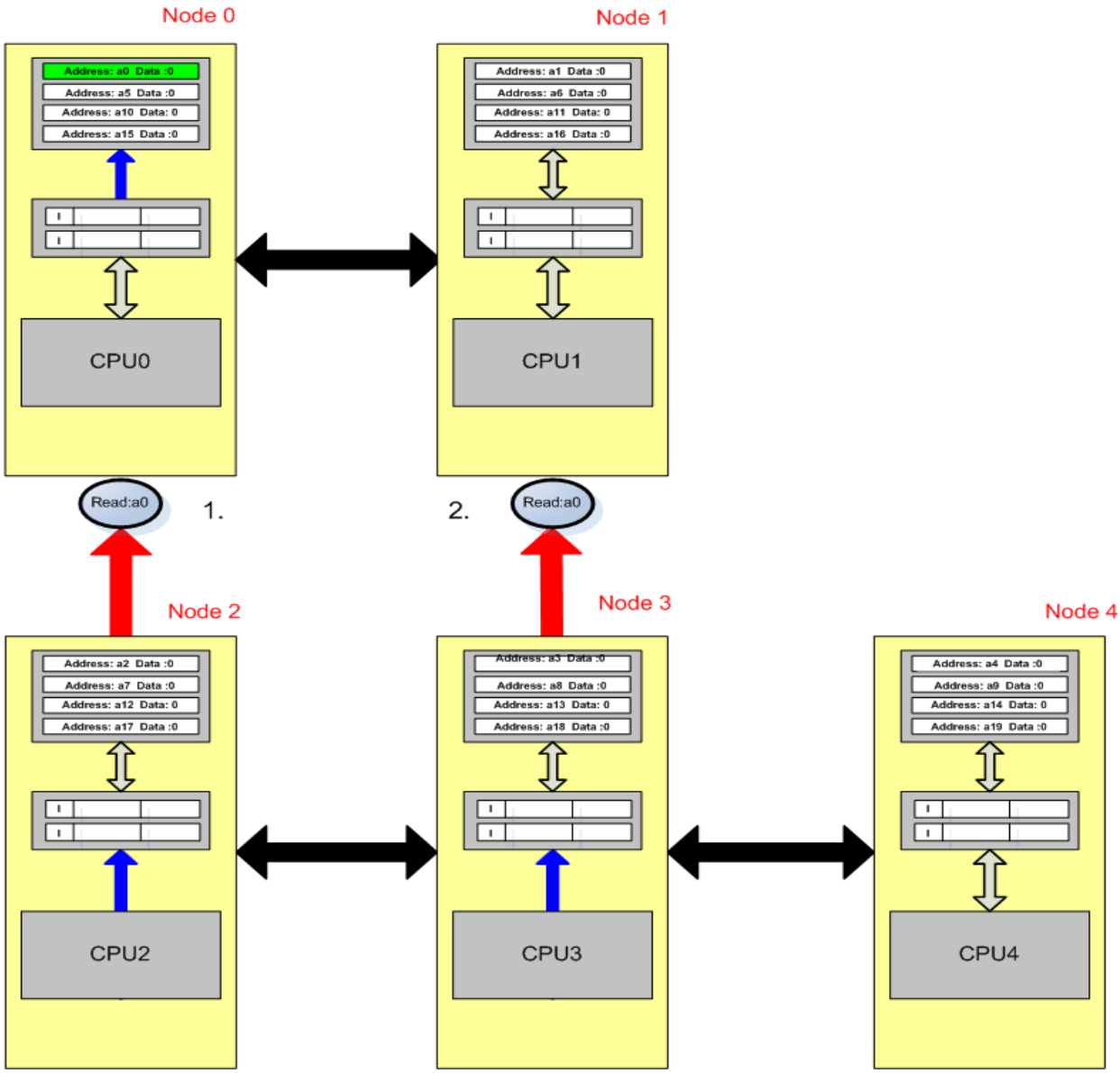


Fig 4.26 - Read Requests

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.
2. 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.

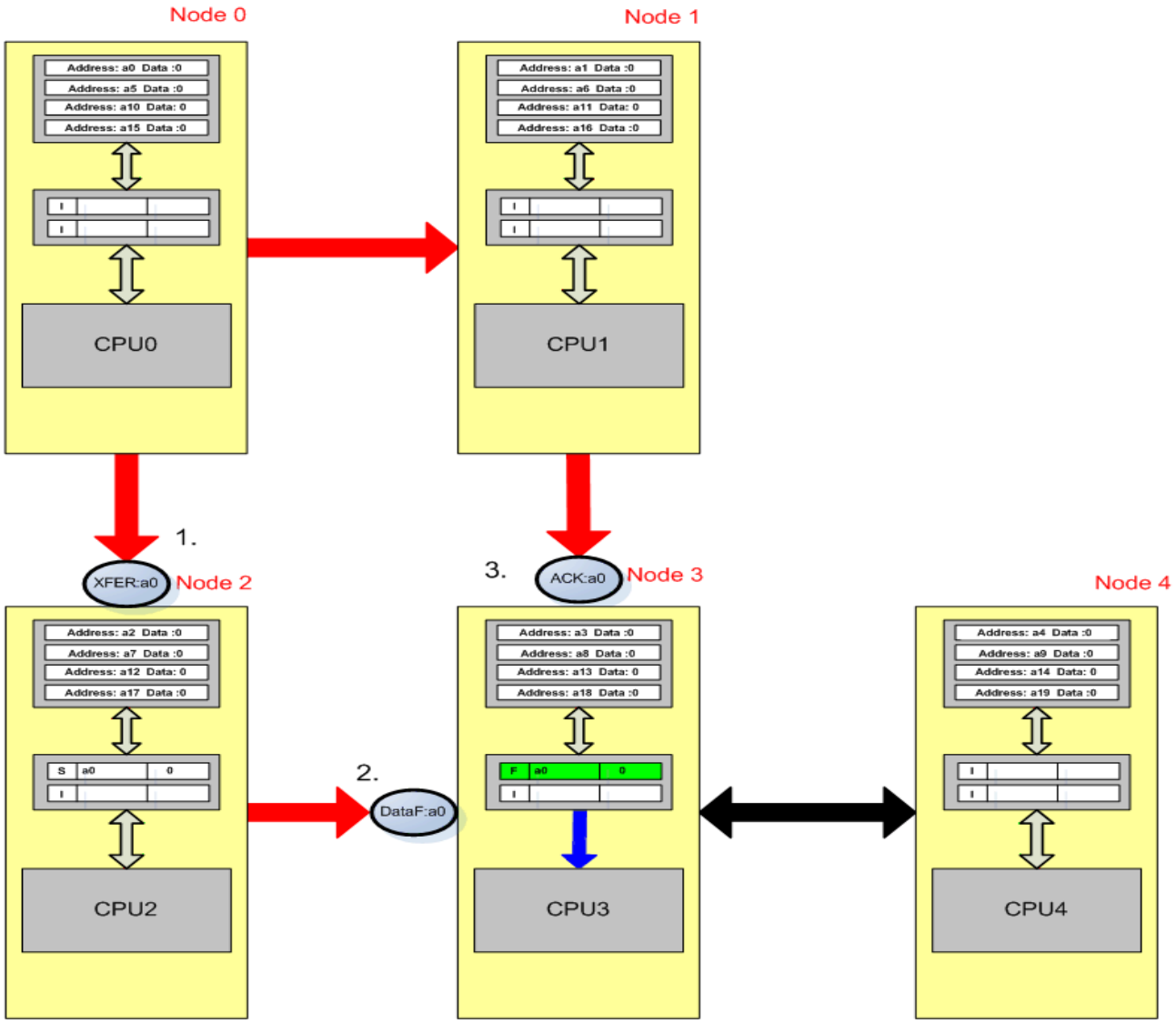


Fig 4.27 - Transfer Message and forward

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.
2. 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.
3. 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.

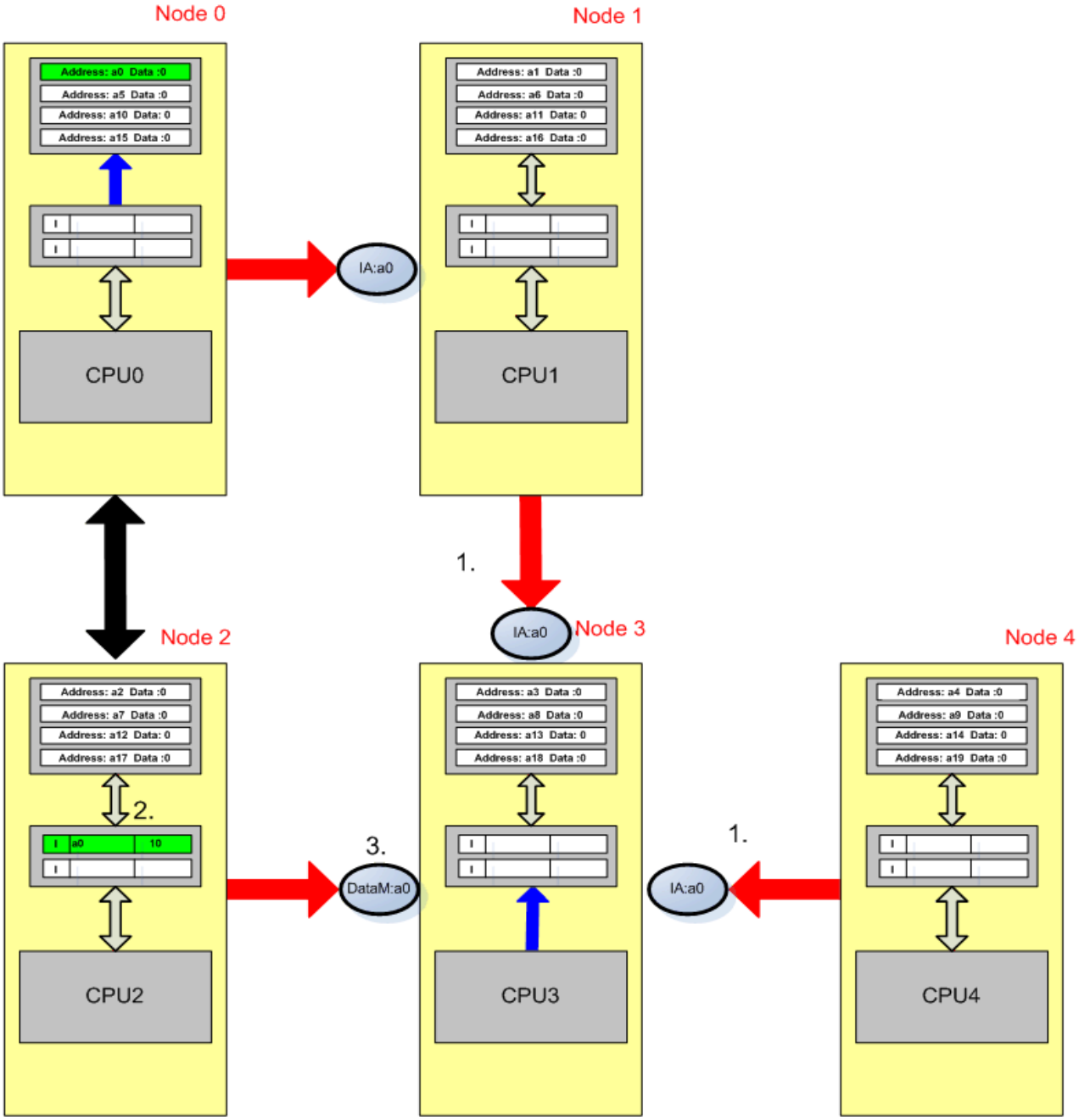


Fig 4.28 - Forward Modified Data

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.

4.7 - Summary

The more complex MESIF protocol seems to be the better of the two. Its capacity to cancel memory accesses when a cached value is found and it’s complicated but superior system of conflict resolution outweigh the advantages of the O states ability to share modified data , however the fact that Intel didn’t implement a similar Owned state feature is peculiar.

5-VIVIO

5.1-Introduction

This chapter is a brief summary of the animation tool Vivio which was used to develop the projects animations and why it's so suitable for the task.

5.2-Background

Vivio is the perfect tool for this project as it was originally made for illustrating cache coherency protocols. [JONES 1]
The reason these animations where needed in the first place was because cache coherence protocols can be complicated and difficult to grasp and Vivio provides many features that make it the optimal choice for these E-learning animations.

5.3 - Features

Vivio was designed with the following features in mind

1. Built-in support for smooth animation
2. Built-in support for animating multiple objects in parallel
3. Easy to install
4. Easy user control of animation playback
5. That the animations should scale with the window in which they are displayed
6. The animations should run at “sufficient” speed whether playing forwards or backwards.

[JONES 2]

- Playback

  The most beneficial aspect of Vivio animations is that they allow users to move backwards and forwards through an animation single stepping through every animation tick and jumping between key checkpoints in the animation.
  A user defined animation speed lets the user speed up and slow down the animation. When complex aspects are being animated its important the user can follow everything on screen so the ability to easily replay and scroll through an animation is Vivios greatest benefit.

- Interactive

  Vivio allows highly user Interactive animations to be developed. There are an extensive amount of states and actions to be explained in this animation, one single animation sequence would likely be boring and uninformative. Vivio allows the user to discover and focus on specific aspects on an animation themselves; engaging user interaction promotes both understanding and enthusiasm on the subject.

- Browser Capabilities

  Availability and ease of access is an important aspect for E-learning animations, Vivio animations are playable in online players and support Opera, Firefox and Internet Explorer. Animations are playable by installing a plug-in or an activeX controller in the case of IE. The user only has to take a few seconds to install the online player. Two websites are to be put online to showcase the animations.

- Rapid development

  Vivio takes charge of animation frame rates and the ability to jump backwards and forwards through the animation in the background, leaving the programmer to only worry about developing the animation itself.

6-Animation: Design and Implementation

6.1-Introduction

This chapter looks at the features, programming and structure of the two animations.
Both the animations utilize the same system topologies so the key difference between the animations is the logic of the protocol itself.
The main aim is to keep the animations clean and simple while being as informative as possible.

6.2 Design

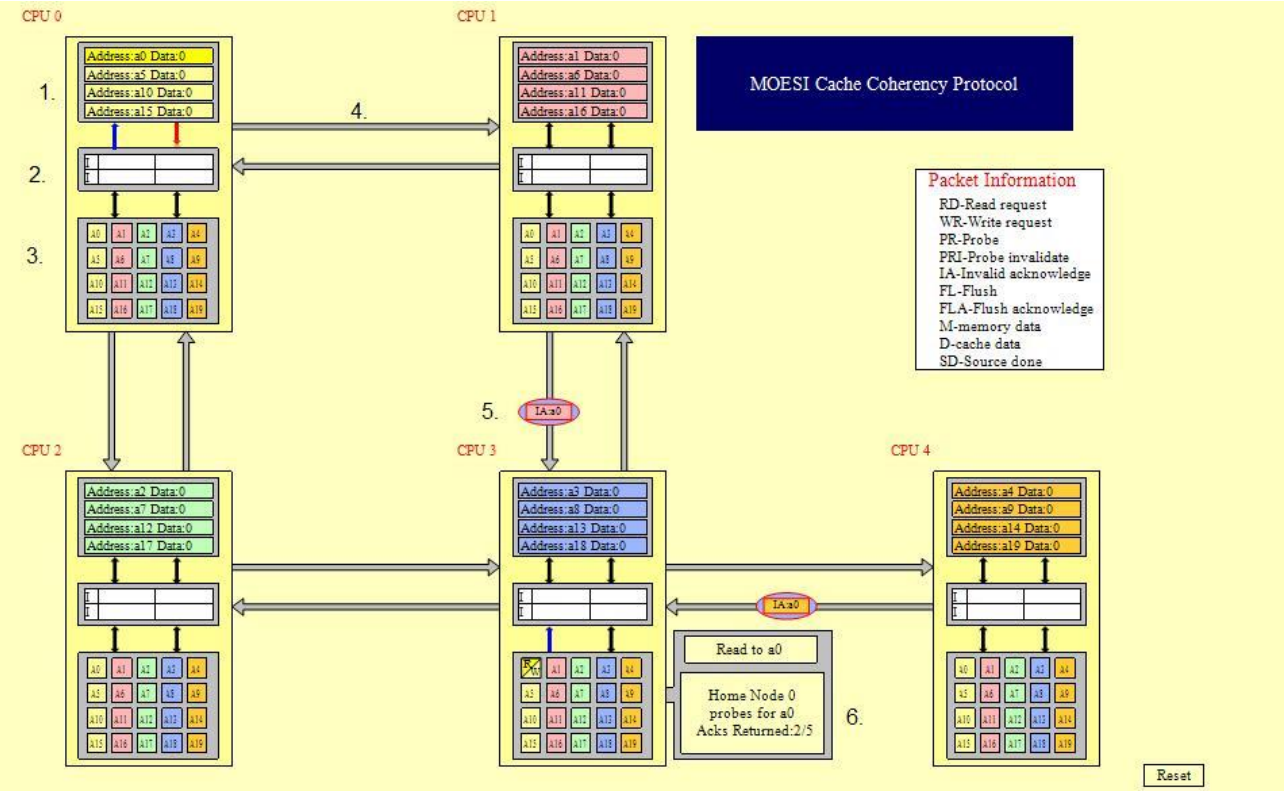


Fig 6.1 – MOESI animation

The animations are designed as shown in **Fig 6.1**; the protocols are being shown on a 5 Node multiprocessor system with each Node consisting of only one CPU with its own cache and memory.

The following numbers correspond to features in **Fig 6.1.**

1. Memory
The memory is split evenly between the Nodes and shared on a global addressing system, as there are 5 CPUs in the system a CPUs local memory holds every 5th memory location.
Each CPUs memory locations are colour coded to make accesses simpler to follow.

2. Cache
Each cache consists of two lines; the lines are direct mapped with the first space holding odd addresses and the second holding even addresses. All lines are initially implemented in the Invalid state.

3. CPU
The CPU is represented using rows of read and write buttons. Each button is colour coded to the memory of the node it is accessing. The user simulates reads and writes with button presses on each CPU.

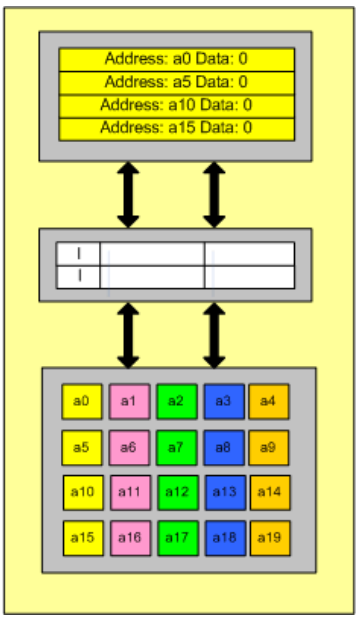


Fig 6.2 – System Node

In total every CPU has to be able to perform a total of 40 actions (20 reads and 20 writes). 40 buttons per node is hardly aesthetically pleasing, as such a single button is used to represent every address, with the user able to pick a read or write operation depending on what side is pressed.

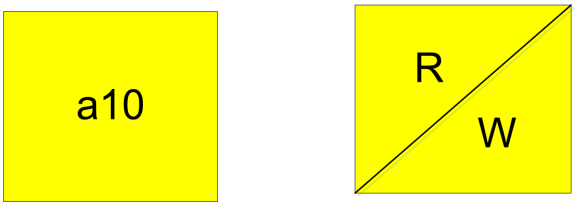


Fig 6.3 - Read Write buttons

When the user places the cursor over the button **Fig 6.3** it will change to show the read and write options, the portion the cursor is over is then highlighted.
If for example a read operation was selected the read portion of the button is kept highlighted until the operation has completed so the user can recall what choice they made.

4. Point-to-Point Links
The Nodes in the system are connected via point to point links, as p2p links are bidirectional separate arrows where used to show that two messages could be sent at once. Only one message can be send at a time on a line.

5. Packets
Messages in this system are sent via packets. As there can be up to 10 visible packets travelling through the system at a time with the potential of more queued messages, the user can find it hard to follow progress on screen and deduce which messages correlate to which process, to clarify a

packets source and destination colour coding was used.

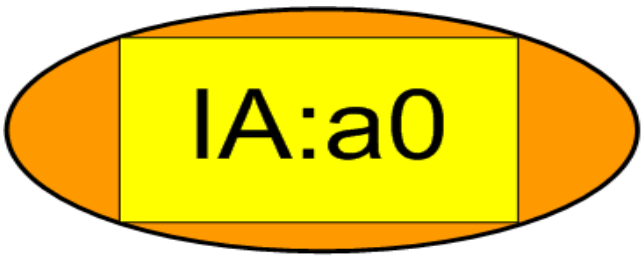


Fig 6.4 – Message Packet

The background colour of the ellipse in **Fig 6.4** gives the colour of the source node of the Request, the colour of the box signifies the node the current message is coming from, **Fig 6.4's** source node is the orange node requesting address space a0, and the yellow node is replying with an invalid acknowledge message.

6. Dialog Box

There's too much information to be solely contained within the transfer packets so a dialog box was decided upon to display extra information, putting the box inside the node itself made the animation to cramped so it was decided to hide the dialogue box behind the node and bring it out when required.

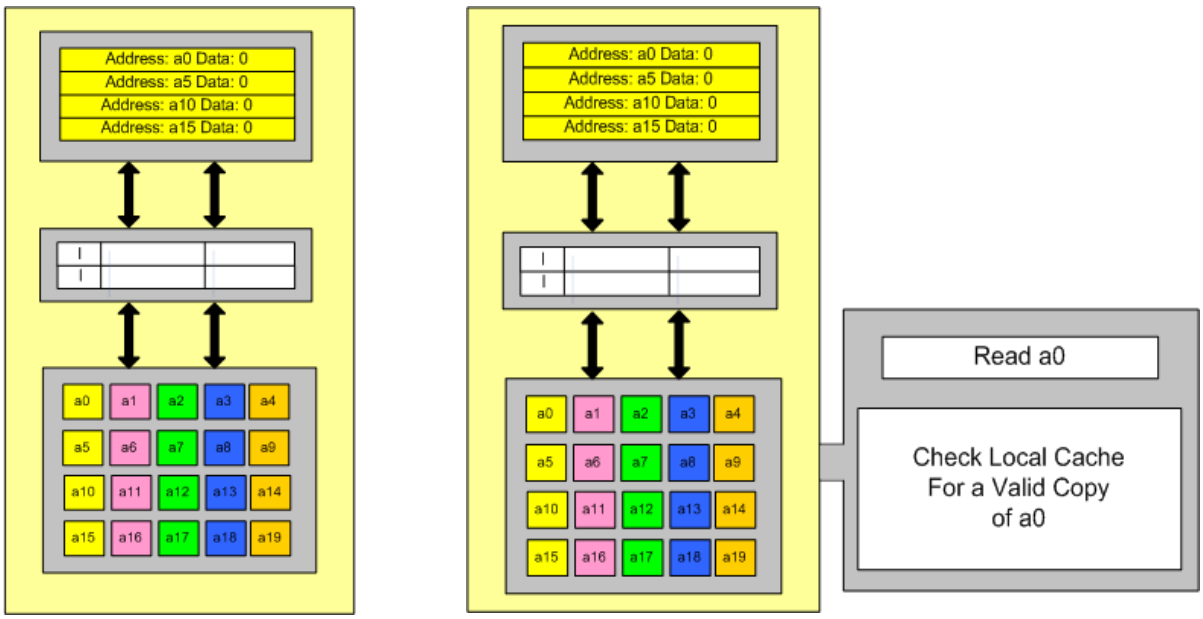


Fig 6.5 – Dialogue box

The box becomes visible when a read or write is initiated, the user with a simple click can hide it.

6.3 Implementation

The basic infrastructure of the two animations is similar with only subtle differences between them.

Nodes

Every node in the system is created by the CPU () class

```
class CPU(int x,int y,int CPUNo)
```

The class creates a node as shown in **Fig 6.5** at the (x, y) coordinates with a defined CPUNo.
The CPU () class consists of 5 subclasses

```
class Memory()
class Cache()
class DataLines(real xpos,real ypos)
class dialogBox()
class rwbuttons(real x, real y,real w,real h,int address)
```

Memory
The Memory () class creates a local memory of 4 spaces, it colour codes the memory locations and names them depending on the CPUNo.

Fig 6.6 – Memory

```
//Create 4 instances
for (int i = 0; i < 4; i++)
        //Stores the data at the 4 memory location
        mem[i] = 0;
        //create 4 rectangles to represent the memory locations
        memR[i]=Rectangle2(0, 0, blackpen,  brushsH[CPUNo], x+2*gap,y+2*gap+(h2+gap/2)*i, w2-gap*2, h2,blackbrush,0);
        //create the text to be displayed in the memory rectangles
        memRTxt[i]=Txt(0,HLEFT|VTOP, x+2*gap, y+2*gap+(h2+gap/2)*i, blackbrush, 0,"Address:a10 Data:10");
        //set text sixe
        memRTxt[i].setSize(w2*0.9,h2);
        //name the memory locations, name every 5th location depending on the CPUNo
        memRTxt[i].setTxt("Address:a%d Data:%d", CPUNo+(i*5), mem[i]);
end;
```

The class contains two functions

```
function highlight(int address, int flag)
function reset()
```

highlight () highlights a specified address location, reset () sets the address locations back to their default values.

Cache
The Cache () class create a 2 line direct mapped cache.

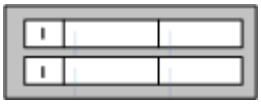


Fig 6.7 – Cache

The class also has a highlight () and reset () function similar to that of the memory () class. There are 3 main variables for each line in this class.

```
for (int i = 0; i < 2; i++)
        //stores the state the line is held in
        state[i]=INVALID;
        //contents of the line
        contents[i]=0;
        //address the line holds
        address[i]=-1;
end;
```

Data Lines

The DataLines() class creates the links connecting memory with cache and cache with the CPU in the animation.

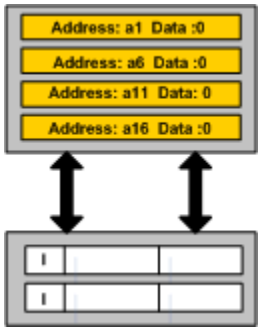


Fig 6.8 – Data Lines

The class contains 3 functions, on creating a DataLines() object two black arrows are created as shown in **Fig 6.8.**

```
function moveUp(int ticks, int wflag)
function moveDown(int ticks, int wflag)
function reset()
```

The moveUp() and moveDown() functions move the left arrow up and the right arrow down respectively, and the third function reset() sets the arrows back to their original state.

Dialog Box

The dialog box as shown in **Fig 6.5** is created by the dialogBox() class, its contains 4 functions

```
function showPBox(int ticks,int RW,int address)
function hidePBox(int ticks)
function setCurrentTask(string message)
function setTitle(string message)
```

The showPBox() function moves the dialog box out from behind the Node and sets the initial message, hidePBox() hides the dialogBox.
The setCurrentTask() and setTitle() functions changes the text in the dialog box.

The class contains 2 event handlers.

```
when processContainer.eventLB(int flags, real x, real y)
when processContainer.eventEE(int flags, real x, real y)
```

The dialog box reacts to 2 events a left mouse click event and an event enter event. A left click of the dialogue box calls the hidePBox() function and retracts the dialog box; entering the dialog box with the mouse highlights the dialog box.

Read/Write buttons
The rwbuttons() class creates a button as shown in **Fig 6.3.**
The class contains five event handlers

```
when background.eventEE(int flags, real x, real y)
when readButton.eventEE(int flags, real x, real y)
when writeButton.eventEE(int flags, real x, real y)
when readButton.eventLB(int flags, real x, real y)
when writeButton.eventLB(int flags, real x, real y)
```

The eventEE(event enter) functions deal with how the box responds to a user putting the cursor over the object.
The eventLB(left mouse button) functions cause a read or write to start depending on what side of the box is clicked.

Point-2-Point Links

The p2p links are created by the p2p() class

```
class p2p(int Ax,int Ay,int l, int CPUA, int CPUB,int orientation,int linkNo)
```

The class creates a bidirectional link at position (Ax, Ay) with a length l, the two CPUs a link connects are defined by CPUA and CPUB, the vertical or horizontal orientation is defined by the orientation variable. Every link is also given its own arbitrary identification number.
The p2p class contains two main functions,

```
function packetUp(int ticks,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message)
function packetDown(int ticks,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message)
```

which send a message packet up or down the links.

The following functions,

```
function linkPathUp(int linkNo,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message)
function linkPathDown(int linkNo,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message)
```

enforce serialisation on the links, a maximum of two packets at a time can be on a link with one travelling in either direction. Each link has a lock for both directions; a packet that has to be sent is put into a while loop until the link is free.

```
function linkPathUp(int linkNo,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message)

    //wait until the links lock is free
    while(linkLockUp[linkNo]==1)
            wait(1);
    end;
            //take possesion of the lock
            linkLockUp[linkNo]=1;
            //send the packet
            link[linkNo].packetUp(TICKS*2,wflag,brush1,brush2,address,messageLock,message);
            wait(TICKS*2);
            //release the lock
            linkLockUp[linkNo]=0;
```

```
end;
```

The linkPath() function is used to send packets from Node to Node it utilises the linkPathUp() and linkPathDown() functions to send packets along a predefined ordered route to simulate the routing tables of a system.

```
function linkPath(int CPUA,int CPUB,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message)
```

Heres an example of the logic of sending a message from CPU0 to any of the peer nodes in the system in the linkPath() function.

```
//if the source CPU is CPU0
if(CPUA==0)
        //if the destination CPU is CPU0,CPU3, or CPU4
        if(CPUB==1 || CPUB==3 || CPUB==4)

                if(CPUB==1)
                        lockMessage=messageLock;
                end;
                //send the message from CPU0 to CPU1
                linkPathUp(0,wflag,brush1,brush2,address,lockMessage,message);
                if(CPUB==3 || CPUB==4)
                        if(CPUB==3)
                                lockMessage=messageLock;
                        end;
                        //send the message from CPU1 to CPU3
                        linkPathUp(2,wflag,brush1,brush2,address,lockMessage,message);
                end;
                if(CPUB==4)
                        //send the message from CPU3 to CPU4
                        linkPathUp(4,wflag,brush1,brush2,address,messageLock,message);
                end;
        end;

        if(CPUB==2)
                //send the messagefrom CPU0 to CPU2
                linkPathUp(1,wflag,brush1,brush2,address,messageLock,message);
        end;

end;
```

The linkPath() function call is all that's required to send packets in this system.

Broadcast probes

Broadcast messages are simulated by the broadcast() function.

```
function broadcast(int CPUA,int returnCPU,int address,int invalidate,int messageLock,string message)
```

The broadcast() function initiates a broadcast message to every Node on the system. It sends a message to every node using the linkPath() function. A node on receiving a broadcast message enters the checkCache() function.

```
function checkCache(int CPUA,int CPUB,int address,int flag,int invalidate)
```

The checkCache() function will check for valid copies of the requested data in a node and reply with the appropriate message to the source node.

Queuing

In the MOESI protocol when the home node receives multiple requests for the same data line it

must queue up the requests and deal with them one at time, for this a queuing system was required. A FIFO (First In First Out) queue implementation was decided on as it's the simplest method, other approaches such as the prioritising of requests is outside the scope of these animations.

In the MESIF animation the same Queue () class was used in a different manner, MESIF does not have the need to queue requests in an ordered list. The Queue () class was used to store the conflicts the Home Node receives, an ordered list is not required for conflict resolution on the home node, however the functions in the Queue() class give desirable functions that a simple array could not.

The queues on both implementations are used to store the CPU number of a process waiting or in conflict and as there are only 5 CPUs the queue need only be of 5 spaces.

The Queue () class consists of the following functions

```
class Queue()

        // adds a process to the end of the Queue
        function addProcess(int CPUNo);

        //removes the current process from the Queue.
        function removeProcess();

        //checks if the process at the head of the queue is the current CPUNo
        int function checkProcess(int CPUNo)

        //emptys the queue, move end point back to 0 and marks the Queue as inactive.
        function emptyQueue()

        //checks for the process and if found deletes it from the queue
        function deleteProcess(int CPUNo)

        //returns the most recent process
        int function returnProcess()

        //returns wheter the queue currently has any waiting process'
        int function isQueueEmpty()
end;
```

6.4 Protocol Logic

MOESI

The logic of the protocol is called in the MOESI () function. The MOESI () function is called on the press of a read or write button. The following logic is based along the path of **Fig 6.9** below.
This path is assuming that the current requested line is uncached locally and that another line is not stored in the O or M states in the requested lines cache location.

1. The read/write request is initiated for a0.
2. Check if the CPU (source node) is already performing a process. If so cancel request.
3. Check for a valid copy of the data stored on the local cache.
4. Check for a cached M/O value at the requested addresses cache line location if a valid line is found flush it to main memory, otherwise continue.
5. Send a read or write request to the home node. Add process to the request queue.
6. Wait until its this CPUs turn on the home node.
7. The home node sends a probe broadcast to every node on the system and simultaneously initiates a memory access.
8. The source node waits until it received all responses.
9. Source node decides which state to change to on receiving all its responses.

Fig 6.9 also contains the logic of when to perform flushes and how to read and write cached data.

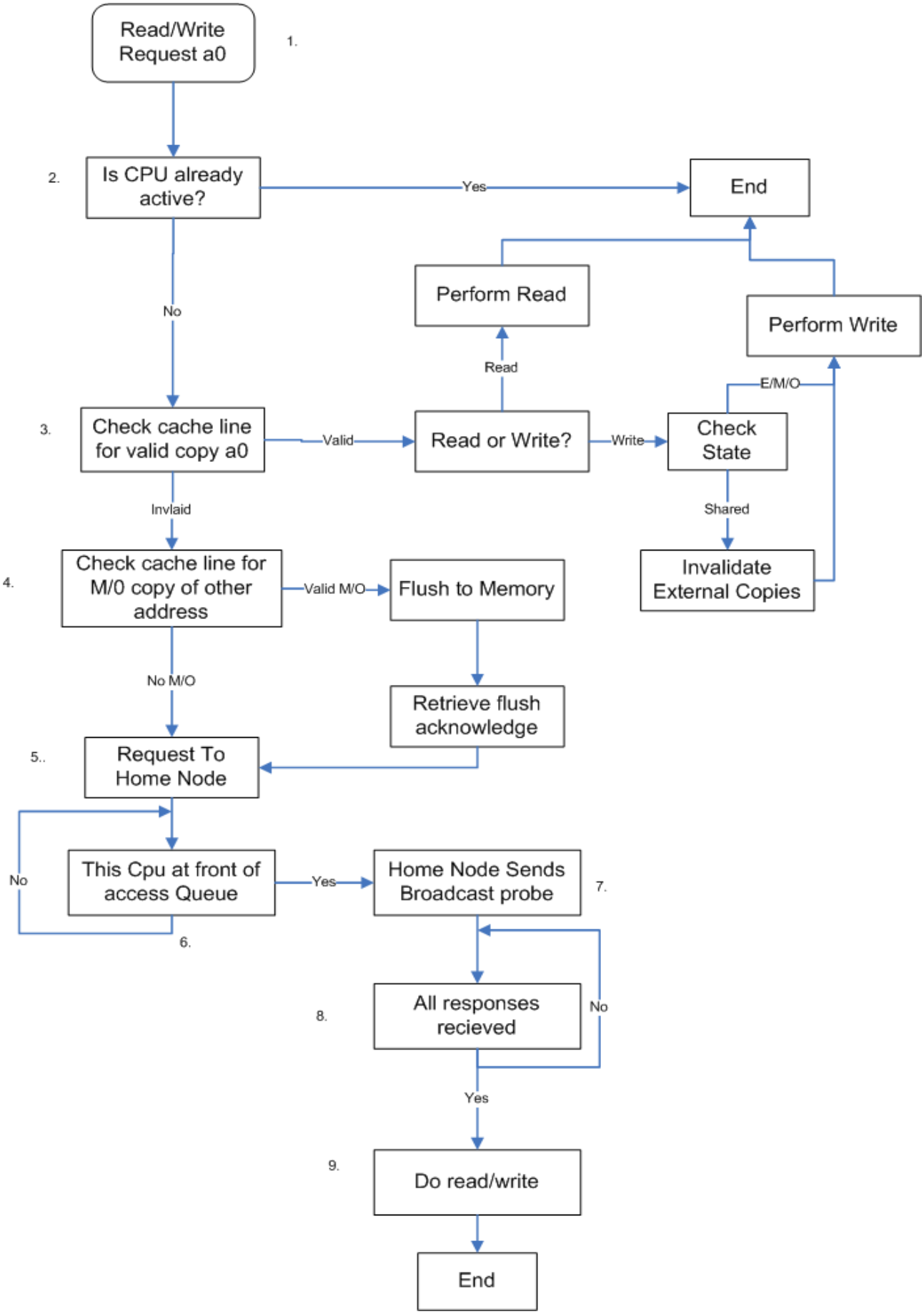


Fig 6.9 – MOESI Logic

MESIF

The logic of the protocol is contained in the MESIF () function. The MESIF () function is called on the press of a read or write button. The following logic is based along the path of **Fig 6.10** below. This path is assuming that the current requested line is uncached locally and that another line is not stored in the M state in the requested lines cache location.

1. The read/write request is initiated for a0.
2. Check if the CPU (source node) is already performing a process. If so cancel request.
3. Check for a valid copy of the data stored on the local cache.
4. Check for a cached M value at the same cache location of the requested address if a valid line is found flush it to main memory, otherwise continue.
5. Send a probe message to every node on the System.
6. Wait until all responses are received.
7. Send a read message to Home Node if there are no valid cached values on the system or a cancel message if a valid value was received.
8. Home Node replies with the data or a confirm message.
9. Perform read or write with the data.

Fig 6.10 also contains the logic of when to perform flushes and how to read and write cached data.

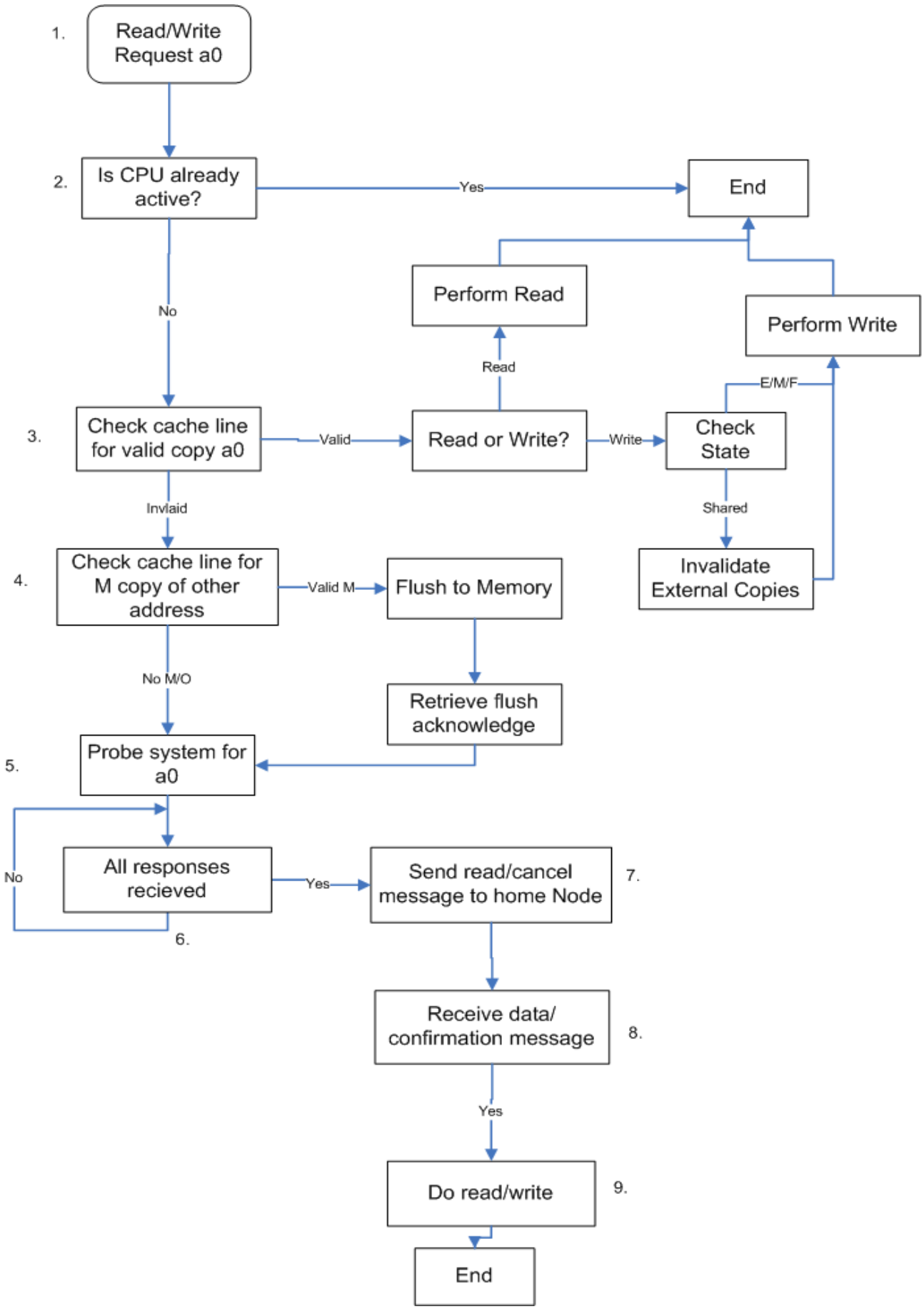


Fig 6.10 – MESIF Logic

7 – Conclusion

Logically the animations are as complete as possible with the information available. The majority of information on this project was garnished from patents. Intel's MESIF protocol was well documented with almost every conflict and eventuality described clearly and concisely, the only aspect still in question is the mechanism of flushing data to memory when a cache line is replaced. AMD however had a distinct lack of information available on many aspects of the MOESI protocol, and what information that was presented was quite often obscured and difficult to assimilate, this lack of information made several factors of the MOESI animation an educated guess, little information was available for write requests and none for parallel write requests to the same physical address. The assumptions made for the write requests are by no means the correct implementation, however they where picked as they went along the same logical mindset of other parts of the protocol. Although the protocols may not be 100% accurate it is hoped that when the protocols are published online, user feedback will hopefully remedy these gaps in the animations.

Aesthetically the animations are clean and simple to follow, development of the animations was an ease with the Vivio tool as shapes are easily created and animated around screen, the core amout of graphical objects implemented where native to Vivio with very few objects like the read/write buttons having to be manually created.

Researching and animating AMD and Intel's protocols have given me an in-depth understanding of cache coherence and the fine balancing act it is to find the most efficient implementation. The MESIF protocol in my opinion is the superior of the two as it can cancel memory accesses,however the exclusion of a similar O state to MOESI seems a peculiar decision on Intels behalf, the reasons for this absense are unclear perhaps the speed benefits where not deemed worth implementing.

8 – References

[JONES 1]

https://www.cs.tcd.ie/Jeremy.Jones/vivio/vivio.htm
A website containing numerous educational animations in Vivio, including a set of cache coherency animation

[JONES 2]

Vivio- A System for Creating Interactive Reversible E-Learning Animation for the WWW. Jeremy Jones.

[AMD]

http://www.amd.com/us-en/Processors/DevelopWithAMD/0,,30_2252_2353,00.html
A website containing background information on HyperTransport technology.

[HYPER 1]

http://www.hypertransport.org/tech/index.cfm
HyperTransport home page containing technical information on the technology.

[HYPER 2]

www.hypertransport.org/docs/wp/WhitePaper_HTC_WP02.pdf
A paper describing HyperTransports features.

[PATENT 1]

Forward state for use in cache coherency in a multiprocessor system, US 6,922,756 2005.
Herbert H. J. Hum, James R. Goodman.
http://www.patentstorm.us/patents/6922756-description.html

[PATENT 2]

Non-speculative distributed conflict resolution for a cache coherency protocol, US 6,954,829 2005.
Robert H. Beers, Herbert H. J. Hum, James R. Goodman.
http://www.freepatentsonline.com/6954829.html

[PATENT 3]

Hierarchical virtual model of a cache hierarchy in a multiprocessor system, US 7,111,128 2006.
Herbert H. J. Hum, James R. Goodman.
http://www.patentstorm.us/patents/7111128-description.html

[PATENT 4]

Hierarchical directories for cache coherency in a multiprocessor system, US 7,130,769 2006.
Herbert H. J. Hum, James R. Goodman.
http://www.patentstorm.us/patents/7130969-claims.html

[CONWAY]

The AMD Opteron Northbridge Architecture, IEEE Computer Society 2007.
Pat Conway, Bill Hughes.
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4287392

9 – Appendices

9.1 – MOESI.viv

```
//
// Vivio MOESI cache coherency protocol animation
//
// include files
//
#include "standard.vin"
#include "simpleButton.vin"
#include "imageButton.vin"

const LOGICALW = 3200;
const LOGICALH = 2100;

setViewport(0, 0, LOGICALW, LOGICALH, 1);


//
// fonts
//
smallFont = Font("Times New Roman", 14, 0);
hintFont = Font("Times New Roman", 16, 0);
titleFont =
 Font("Times New Roman", 24, 0);
//
// set background
//
offset=50;
bgbrush = SolidBrush(vellum);
setBgBrush(bgbrush);
navybrush = SolidBrush(rgb(0, 0, 102));
const int coloursH[5]={rgb(255, 255,  150),rgb(255, 183,  183),rgb(191, 255,  183),rgb(155, 182,  255),rgb(255, 204, 51)};
const int coloursS[5]={rgb(255, 255, 0),rgb(200, 0, 0), rgb(0, 255, 0),rgb(0, 0, 255),rgb(245, 184, 0)};
Brush brushsH[5];
Brush brushsS[5];
brushsH[0]= SolidBrush(coloursH[0]);
brushsH[1]= SolidBrush(coloursH[1]);
brushsH[2]= SolidBrush(coloursH[2]);
brushsH[3]= SolidBrush(coloursH[3]);
brushsH[4]= SolidBrush(coloursH[4]);
brushsS[0]= SolidBrush(coloursS[0]);
brushsS[1]= SolidBrush(coloursS[1]);
brushsS[2]= SolidBrush(coloursS[2]);
brushsS[3]= SolidBrush(coloursS[3]);
brushsS[4]= SolidBrush(coloursS[4]);
smallFont = Font("Times New Roman", 14, 0);
hintFont = Font("Times New Roman", 50, 0);
titleFont = Font("Times New Roman", 50, 0);

navybrush = SolidBrush(rgb(0, 0, 102));
title = Rectangle2(0, VCENTRE, 0, navybrush, LOGICALW-1500+offset*2, 100, 1000, 250, whitebrush, titleFont,"MOESI Cache Coherency Protocol");
title.setTxtOff(2, 1);
hintFont2 = Font("Times New Roman", 40, 0);


infoRect = Rectangle2(0, 0, blackpen, whitebrush,  LOGICALW-920+offset*2, 450, 500, 530);
//iNFORMATION BOX
str1 =  "Packet Information\n";
str += "RD-Read request\n";
str += "WR-Write request\n";
str += "PR-Probe\n";
str += "PRI-Probe invalidate\n";
str += "IA-Invalid acknowledge\n";
str += "FL-Flush\n";
str += "FLA-Flush acknowledge\n";
str += "M-memory data\n";
str += "D-cache data\n";
str += "SD-Source done\n";
hint = Txt(0, HLEFT | VTOP, LOGICALW-890+offset*2, 450, redbrush, hintFont, str1);
hint2 = Txt(0, HLEFT | VTOP, LOGICALW-860+offset*2, 520, blackbrush, hintFont2, str);

//A lock is needed for every memory location as multiple processors may access it at the same time
int memLock[20];
for(j=0;j<20;j++)
                memLock[j]=0;
end;

//
// title
//
navybrush = SolidBrush(rgb(0, 0, 102));
```

```
//
// title
//
navybrush = SolidBrush(rgb(0, 0, 102));

const int NCPU = 5;                                     // number of cpus
const int TICKS = 20;                   // animation speed
const int INVALID = 0;                  // M0ESI cache line states
const int SHARED = 1;
const int EXCLUSIVE = 2;
const int MODIFIED = 3;
const int OWNED = 4;
int ctrlPressed=0;

class CPU;
CPU cpu[5];
class p2p;
p2p link[5];
function checkCache(int cpuA,int cpuB,int address,int flag,int invalidate,int messageLock);
function linkPath(int cpuA,int cpuB,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message);
function linkPathDown(int linkNo,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message);
function linkPathUp(int linkNo,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message);
//The current process the system is on
int currentProcessNo=0;
int invalidateRestart[20];
class Queue()

        //Queue can store 5 operations
        int queue[5];
        int endPosition=0;
        int queueActive=0;

        for(int i=0;i<5;i++)
                queue[i]=-1;
        end;

        function addProcess(int cpuNo)
                queue[endPosition]=cpuNo;
                queueActive=1;
                endPosition++;
                if(endPosition==5)
                        endPosition--;
                end;
        end;
        function removeProcess()
                for(i=0;i<4;i++)
                        queue[i]=queue[i+1];
                end;
                endPosition--;
                if(endPosition==0)
                        queueActive=0;
                end;
                if(endPosition==-1)
                        endPosition=0;
                end;
        end;
        function deleteProcess(int cpuNo)
                for(i=0;i<endPosition;i++)
                        if(queue[i]==cpuNo)
                                for(j=i;j<endPosition;j++)
                                        queue[j]=queue[j+1];
                                end;
                                endPosition--;
                                if(endPosition==0)
                                        queueActive=0;
                                end;
                                if(endPosition==-1)
                                        endPosition=0;
                                end;
                        end;
                end;
        end;
        int function returnProcess()
                return queue[0];
        end;
end;
//need a queue for every memory space
Queue homeNodeQueue[20];
for(i=0;i<20;i++)
        homeNodeQueue[i]=Queue();
        invalidateRestart[i]=0;
end;

for(i=0;i<NCPU;i++)
 linkLockUp[i]=0;
```

```
 linkLockDown[i]=0;
 DRAMLock[i]=0;
end;

class CPU(int x,int y,int cpuNo)
		w=400;
		gap=15;
		h=700;
		h2=h/17;
		w2=w-(2*gap);
		int cacheLineFound=-1;
		int repliesReturned=0;
		int memoryAccessComplete=0;
		int cancelMemoryAccess=0;
		cpuLock=0;
		class Memory;
		class Cache;
		class DataLines;
		class dialogBox;
		Memory cpuMemory;
		Cache cpuCache;
		int dataLine=-1;
		DataLines cpuDataLines;
		DataLines cpuDataLines2;
		dialogBox cpudialogBox;
		int READ=0;
		int WRITE=1;
		int ReadOrWrite=READ;
		brushTemp= SolidBrush(rgb(255, 255, 150));
		r = Rectangle2(0, 0, blackpen, brushTemp, x-20, y, w+40, h+80);
		int invalidating=-1;
		int cpuRestart=0;
		int mesifUpdateMemory=0;
		int currentProcess=-1;
		Queue writeConflictQueue[2];
		int cQLock[2];
		cQLock[0]=0;
		cQLock[1]=0;
		writeConflictQueue[0]=Queue();
		writeConflictQueue[1]=Queue();
		conflictQueue[0]=Queue();
		conflictQueue[1]=Queue();
		int readCycle=-1;
		int probeCycle=-1;
		int readConflict=-1;
		int probeConflict=-1;
		int waitingOnHomeNodeAck[20];
		int mesifWrite=0;
		int conflict=0;
		int isCpuWriting=0;
		int lineModified=0;
		int localRorW[20];
		for(i=0;i<20;i++)
			localRorW[i]=0;
			waitingOnHomeNodeAck[i]=0;
		end;
		titleFont = Font("Times New Roman", 40, 0);
		t = Txt(0, HLEFT | VTOP, x-140, y-80, redbrush, titleFont, "CPU %d", cpuNo);

		class Memory()

			int mem[5];
			int stale[5];
			r = Rectangle2(0, 0, blackpen, gray192brush, x+gap, y+gap, w2, gap+h2*4.75);

			for (int i = 0; i < 4; i++)
				mem[i] = 0;
				stale[i] = 0;
				memR[i] = Rectangle2(0, 0, blackpen,  brushsH[cpuNo], x+2*gap, y+2*gap+(h2+gap/2)*i, w2-gap*2, h2, blackbrush, 0);
				memRTxt[i]=Txt(0,HLEFT|VTOP, x+2*gap, y+2*gap+(h2+gap/2)*i, blackbrush, 0,"Address:a10 Data:10");
				memRTxt[i].setSize(w2*0.9,h2);
				memRTxt[i].setTxt("Address:a%d Data:%d", cpuNo+(i*5), mem[i]);
			end;

			function highlight(int address, int flag)
				memR[address].setBrush((flag) ? brushsH[cpuNo] : brushsS[cpuNo]);
			end;

			function reset()
				for (int i = 0; i < 4; i++)
					mem[i] = 0;
					stale[i] = 0;
				end;
			end;
		end;
		class Cache()
			int state[2];
```

```
		int content[2];
		int address[2];

		r = Rectangle2(0, 0, blackpen, gray192brush, x+gap, y+h*0.421, w2, (h2+gap)*2);
		for (int i = 0; i < 2; i++)
			state[i]=INVALID;
			contents[i]=0;
			address[i]=-1;

			stateR[i] = Rectangle2(0, 0, blackpen, whitebrush, x+gap*2, y+h*0.421+gap+i*h2, w2*0.1, h2, blackbrush, 0);
			stateRTxt[i]=Txt(0,HLEFT|VTOP,  x+gap*2, y+h*0.421+gap+i*h2, blackbrush, 0,"M");
			stateRTxt[i].setSize(w2*0.1,h2);
			stateRTxt[i].setTxt("I");

			aR[i] = Rectangle2(0, 0, blackpen, whitebrush, x+gap*2+w2*0.1,y+h*0.421+gap+i*h2, w2*0.41, h2);
			aRTxt[i]=Txt(0,HLEFT|VTOP,  x+gap*2+w2*0.20,y+h*0.421+gap+i*h2, blackbrush, 0,"A10");
			aRTxt[i].setSize(w2*0.2,h2);
			aRTxt[i].setTxt("");

			 dR[i] = Rectangle2(0, 0, blackpen, whitebrush,  x+gap*2+w2*0.51,y+h*0.421+gap+i*h2, w2*0.41,h2);
			dRTxt[i]=Txt(0,HLEFT|VTOP,  x+gap*2+w2*0.60,y+h*0.421+gap+i*h2, blackbrush, 0,"80");
			dRTxt[i].setSize(w2*0.2,h2);
			dRTxt[i].setTxt("");
		end;
		function highlight(int cacheLine, int flag)
			stateR[cacheLine].setBrush((flag) ? whitebrush : greenbrush);
			aR[cacheLine].setBrush((flag) ? whitebrush : greenbrush);
			dR[cacheLine].setBrush((flag) ? whitebrush : greenbrush);
		end;
		function reset()
			for (int i = 0; i < 2; i++)
				state[i]=INVALID;
				contents[i]=0;
				address[i]=-1;
			end;
		end;
end;
class DataLines(real xpos,real ypos)
		bodyWidth=5;
		aWidth=12;
		l=60;

		arrow1 = Polygon(0, AAFILL | ABSOLUTE, 0, blackbrush, xpos,ypos, 0,0, -aWidth,-aWidth,
		-bodyWidth,-aWidth, -bodyWidth,-l, -aWidth,-l, 0,-l-aWidth, aWidth,-l, bodyWidth,-l, bodyWidth,-aWidth, aWidth,-aWidth );
		arrow2 = Polygon(0, AAFILL | ABSOLUTE, 0, blackbrush, xpos+h2*4,ypos, 0,0, -aWidth,-aWidth,
		-bodyWidth,-aWidth, -bodyWidth,-l, -aWidth,-l, 0,-l-aWidth, aWidth,-l, bodyWidth,-l, bodyWidth,-aWidth, aWidth,-aWidth );

		body1=Rectangle2(0, 0, 0, bluebrush, xpos-bodyWidth, ypos, 2*bodyWidth, -l);
		body2=Rectangle2(0, 0, 0, redbrush,  xpos-bodyWidth+h2*4,ypos-aWidth, 2*bodyWidth, -l);
		up= Polygon(0, AAFILL | ABSOLUTE, 0, bluebrush, xpos-aWidth,ypos, 0,0, aWidth,-10, 2*aWidth,0);
		down= Polygon(0, AAFILL | ABSOLUTE, 0, redbrush,  xpos+h2*4,ypos-l, 0,0, -aWidth,-10, aWidth,-10);

		up.setOpacity(0);
		down.setOpacity(0);
		body1.setOpacity(0);
		body2.setOpacity(0);

		function moveUp(int ticks, int wflag)

					// initial positions, size & opacity
					up.setPos(xpos-aWidth,ypos);
					body1.setPos(xpos-bodyWidth, ypos);
					body1.setSize(2*bodyWidth, 0);
					up.setOpacity(255);
					body1.setOpacity(255);

					// final positions, size & opacity
					up.setPos( xpos-aWidth,ypos-l, ticks, 1, 0);
					body1.setSize(2*bodyWidth, -l, ticks, 1, 0);
					arrow1.setOpacity(0, ticks, 1, wflag);

		end;

//
// animates down movement on vertical bus arrow
//
// may be called with ticks = 0
//
		function moveDown(int ticks, int wflag)
			//
			// initial positions, size & opacity
			//
			down.setPos( xpos+h2*4,ypos-l);
			body2.setPos (xpos-bodyWidth+h2*4,ypos-l-10);
			body2.setSize(2*bodyWidth, 0);
			down.setOpacity(255);
			body2.setOpacity(255);
```

```
                //
                // final positions, size & opacity
                //

                body2.setSize( 2*bodyWidth, l, ticks, 1, 0);
                down.setPos( xpos+h2*4,ypos, ticks, 1, 0);
                arrow2.setOpacity(0, ticks, 1, wflag);
        end;

        function reset()
                up.setOpacity(0);
                body1.setOpacity(0);
                down.setOpacity(0);
                body2.setOpacity(0);
                arrow1.setOpacity(255);
                arrow2.setOpacity(255);
        end;
end;

//Paths of broadcast message, needed as the the check
function cpu0path(int cpuA,int returnCpu,int address,int invalidate,int messageLock,string message)
                linkPath(0,1,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,1, address,0,invalidate,messageLock));
                linkPath(1,3,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,3, address,0,invalidate,messageLock));
                linkPath(3,4,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,4, address,0,invalidate,messageLock));
end;
function cpu1path1(int cpuA,int returnCpu,int address,int invalidate,int messageLock,string message)
                linkPath(1,3,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,3, address,0,invalidate,messageLock));
                linkPath(3,4,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,4, address,0,invalidate,messageLock));
end;
function cpu1path2(int cpuA,int returnCpu,int address,int invalidate,int messageLock,string message)
                linkPath(1,0,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,0, address,0,invalidate,messageLock));
                linkPath(0,2,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,2, address,0,invalidate,messageLock));
end;
function cpu2path1(int cpuA,int returnCpu,int address,int invalidate,int messageLock,string message)
                linkPath(2,3,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,3, address,0,invalidate,messageLock));
                linkPath(3,4,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,4, address,0,invalidate,messageLock));
end;
function cpu2path2(int cpuA,int returnCpu,int address,int invalidate,int messageLock,string message)
                linkPath(2,0,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,0, address,0,invalidate,messageLock));
                linkPath(0,1,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,1, address,0,invalidate,messageLock));

end;
function cpu3path1(int cpuA,int returnCpu,int address,int invalidate,int messageLock,string message)
                linkPath(3,1,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,1, address,0,invalidate,messageLock));
                linkPath(1,0,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,0, address,0,invalidate,messageLock));
end;
function cpu3path2(int cpuA,int returnCpu,int address,int invalidate,int messageLock,string message)
                linkPath(3,2,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,2, address,0,invalidate,messageLock));
end;
function cpu3path3(int cpuA,int returnCpu,int address,int invalidate,int messageLock,string message)
                linkPath(3,4,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                fork(checkCache(returnCpu,4, address,0,invalidate,messageLock));
end;
//sends broadcast probe to every node in the system
function broadcast(int cpuA,int returnCpu,int address,int invalidate,int messageLock,string message)
                if(cpuA==0)
                                fork(cpu0path(cpuA,returnCpu,address,invalidate,messageLock,message));
                                fork(checkCache(returnCpu,0, address,0,invalidate,messageLock));
                                linkPath(0,2,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                                fork(checkCache(returnCpu,2, address,0,invalidate,messageLock));
                end;
                if(cpuA==1)
                                fork(cpu1path1(cpuA,returnCpu,address,invalidate,messageLock,message));
                                fork(cpu1path2(cpuA,returnCpu,address,invalidate,messageLock,message));
                                fork(checkCache(returnCpu,1, address,0,invalidate,messageLock));
                end;
                if(cpuA==2)
                                fork(cpu2path1(cpuA,returnCpu,address,invalidate,messageLock,message));
                                fork(cpu2path2(cpuA,returnCpu,address,invalidate,messageLock,message));
                                fork(checkCache(returnCpu,2, address,0,invalidate,messageLock));
                end;
```

```
                if(cpuA==3)
                                fork(cpu3path1(cpuA,returnCpu,address,invalidate,messageLock,message));
                                fork(cpu3path2(cpuA,returnCpu,address,invalidate,messageLock,message));
                                fork(cpu3path3(cpuA,returnCpu,address,invalidate,messageLock,message));
                                fork(checkCache(returnCpu,3, address,0,invalidate,messageLock));
                end;

                if(cpuA==4)
                                linkPath(4,3,0,brushsH[returnCpu],brushsH[cpuA],address,messageLock,message);
                                fork(checkCache(returnCpu,4, address,0,invalidate,messageLock));
                                fork(cpu3path1(cpuA,returnCpu,address,invalidate,messageLock,message));
                                fork(cpu3path2(cpuA,returnCpu,address,invalidate,messageLock,message));
                                fork(checkCache(returnCpu,3, address,0,invalidate,messageLock));
                end;


        end;




class dialogBox()
        int x1=x-20;
        int x2=x1+440;
        int y1=y+570;
        int w=420;
        int h=150;
        int h2=40;
        int boxLock=0;
        int t1x=50;
        int t1y=y1-135;
        int t2x=40;
        int t2y=y1-40;
        f1 = Font("Times New Roman", 40, 0);
        f2 = Font("Times New Roman", 40, 0);

        processTxtBox1=Rectangle2(0,0, blackpen, vellumbrush, t1x+x1,t1y, w-70, h-80,blackbrush, f1, "");
        processTxtBox1.moveToBack();
        processTxtBox2=Rectangle2(0,0, blackpen, vellumbrush, t2x+x1,t2y, w-40, h+65,blackbrush, f2, "");
        processTxtBox2.moveToBack();
        processContainer= Polygon(0, 0, blackpen, gray192brush, x1,y1, 0,0, 20,0, 0,-h, w,0, 0,h*2+h2, -w,0, 0,-h, -20,0);
        processContainer.moveToBack();

        int ticksLocal=TICKS;
        int boxOut=0;


        function showPBox(int ticks,int RW,int address)
                if(RW==READ)
                        processTxtBox1.setTxt("Read to a%d",address);
                else
                        processTxtBox1.setTxt("Write to a%d",address);
                end;
                processContainer.setPos( x2,y1, ticks, 1, 0);
                processTxtBox1.setPos( x2+t1x,t1y, ticks, 1, 0);
                processTxtBox2.setPos( x2+t2x,t2y, ticks, 1, 1);
                boxOut=1;
        end;
        function hidePBox(int ticks)
                processContainer.setPos( x1,y1, ticks, 1, 0);
                processTxtBox1.setPos( x1+t1x,t1y, ticks, 1, 0);
                processTxtBox2.setPos( x1+t2x,t2y, ticks, 1, 1);
                boxOut=0;
        end;

        when processContainer.eventLB(int flags, real x, real y)
                if(boxOut==1 && flags & EB_LEFT==0)
                        boxOut=0;
                        start(1);
                        processContainer.setPos( x1,y1, ticksLocal, 1, 0);
                        processTxtBox1.setPos( x1+t1x,t1y, ticksLocal, 1, 0);
                        processTxtBox2.setPos( x1+t2x,t2y, ticksLocal, 1, 1);
                        processContainer.setBrush( gray192brush);
                        processTxtBox1.setBrush(vellumbrush);
                        processTxtBox2.setBrush(vellumbrush);
                end;
        end;

        when processContainer.eventEE(int flags, real x, real y)
                if(boxOut==1)
                        processContainer.setBrush(flags & EB_ENTER ?  gray128brush :  gray192brush);
                        processTxtBox1.setBrush(flags & EB_ENTER ?  gray192brush :  vellumbrush);
                        processTxtBox2.setBrush(flags & EB_ENTER ?  gray192brush :  vellumbrush);
                end;
        end;
```

```
		function setCurrentTask(string message)
			processTxtBox2.setTxt(message);
		end;

		function setTitle(string message)
			processTxtBox1.setTxt(message);
		end;
end;

function concurrentQueueRelease(int cpuA,int cpuB,int address,int doWrite,string mess)
		linkPath(cpuA,cpuB,0,brushsH[cpuA],brushsH[cpuA],address,-1,mess);
		if(doWrite==WRITE)
			invalidateRestart[address]=0;
		end;
		homeNodeQueue[address].removeProcess();
end;



//check memory in home node
function accessMemory(int address,Brush brush1,Brush brush2,int messageLock)
			cpu[address%NCPU].cpuDataLines.moveUp(TICKS*4, 1);
			cpu[address%NCPU].cpuMemory.highlight(address/4,0);
			cpu[address%NCPU].cpuDataLines.moveDown(TICKS*4, 1);
			message=format("M-%d:a",cpu[address%NCPU].cpuMemory.mem[address/4]);
			linkPath(address%NCPU,cpuNo,0,brush2,brush1,address,messageLock,message);
			repliesReturned++;
			message=format("Home Node %d\nprobes for a%d\n Acks Returned:%d/5", address%NCPU,address,repliesReturned-1);
			cpudialogBox.setCurrentTask(message);
end;

//The Moesi protocol logic
function MOESI(int address,int doWrite)
		//If the cpu is currently doing an operation
		if(cpuLock == 0)
			//start the animation
			start(1);
			repliesReturned=0;
			message=format("Check Local Cache\nfor a Valid copy\nof a%d",address);
			cpudialogBox.setCurrentTask(message);
			if(doWrite==WRITE)
				cpudialogBox.showPBox(20,WRITE,address);
			else
				cpudialogBox.showPBox(20,READ,address);
			end;
			invalidating=-1;
			cpuRestart=0;
			//start animating, lock the the cpu and move the data lines up
			cpuLock=1;
			dataLine=-1;
			cacheLineFound=-1;
			cpuDataLines2.moveUp(TICKS, 1);
			int read=0;
			//if the addressess is not on the cache or is invalid ie not available locally start the moesi protocol
			if(cpuCache.address[address%2]!=address || cpuCache.state[address%2]==INVALID)
				read=1;
				//if the cache currently holds a cache line at your requested location
				//in owned or exclusive you must first flush it to memory
				if(cpuCache.state[address%2]==MODIFIED || cpuCache.state[address%2]==OWNED )
					int flushAddr=cpuCache.address[address%2];
					int homeNode=flushAddr%NCPU;
					//add your request to flush the addr to the address queue
					//homeNodeQueue[flushAddr].addProcess(cpuNo);
				  //send your flush data to the home node
					message=format("Flush a%d to\nHome Node %d",flushAddr,homeNode);
					cpudialogBox.setCurrentTask(message);
					linkPath(cpuNo,homeNode,0,brushsH[cpuNo],brushsH[cpuNo],flushAddr,cpuNo,"FL:a");
					//move the data lines to the home node memory
					cpu[homeNode].cpuDataLines.moveUp(TICKS*4, 1);
					//store the data on the home node memory
					cpu[homeNode].cpuMemory.mem[flushAddr/4]=cpuCache.contents[address%2];
					cpu[homeNode].cpuMemory.memRTxt[flushAddr/4].setTxt("Address:a%d Data:%d",
					cpuCache.address[address%2], cpuCache.contents[address%2]);
					//highlight the data on the homenode memory
					cpu[homeNode].cpuMemory.highlight(address/4,0);
					cpu[homeNode].cpuDataLines.moveDown(TICKS*4, 1);
					//remove the process from the address queue
					homeNodeQueue[flushAddr].removeProcess();
					//return the flush acknowledgement to the source node
					message=format("Home Node %d\nreturns\nFlush Ack for a%d",homeNode,flushAddr);
					cpudialogBox.setCurrentTask(message);
		linnkPath(cpuCache.address[address%2]%NCPU,cpuNo,0,brushsH[cpuNo],brushsH[cpuCache.address[address%2]%NCPU],flushAddr,-
		1,"FLA:a");
					//set the data at the source node to invalid
					cpuCache.state[address%2]=INVALID;
					cpuCache.stateRTxt[address%2].setTxt("I");
```

```
		cpuCache.highlight(address%2,1);

end;
//this lock and the first while loop below stop the processor continuing until it has recieved all acks
//send the request to home node
if(doWrite==WRITE)
		message=format("Sending Write Request\nto Home Node %d", address%NCPU);
		cpudialogBox.setCurrentTask(message);
		linkPath(cpuNo,address%NCPU,0,brushsH[cpuNo],brushsH[cpuNo],address,cpuNo,"WR:a");
else
		message=format("Sending Read Request\nto Home Node %d", address%NCPU);
		cpudialogBox.setCurrentTask(message);
		linkPath(cpuNo,address%NCPU,0,brushsH[cpuNo],brushsH[cpuNo],address,cpuNo,"RD:a");
end;
		//this function checks the memory on the home node it moves the data line
		//and sends back a reply message

fork(accessMemory(address,brushsH[address%NCPU],brushsH[cpuNo],-1));
//send broadcast messages from the home node to all other nodes, the program cant continue until
//all the cache responses from the broadcast come back
message=format("Home Node %d\nprobes for a%d\n Acks Returned:0/5", address%NCPU,address);
cpudialogBox.setCurrentTask(message);
if(doWrite==WRITE)
		invalidating=address;
		invalidateRestart[address]=1;
		broadcast(address%NCPU,cpuNo,address,1,-1,"PRI:a");
else
		broadcast(address%NCPU,cpuNo,address,0,-1,"PR:a");
end;
//now wait until the reply from the memory comes
while(repliesReturned!=6)
		wait(1);
end;
//if  has remained -1 through all these function calls then no other cache has the
//data therefore go exclusive
if(cacheLineFound==-1)
		message=format("No Dirty Cache Lines,\nSend Source Done\nMessage");
		cpudialogBox.setCurrentTask(message);
		//take the data directly from the memory
		cpuCache.contents[address%2]=cpu[address%NCPU].cpuMemory.mem[address/4];
		cpuCache.state[address%2]=EXCLUSIVE;
		cpuCache.stateRTxt[address%2].setTxt("E");
else
		if(dataLine==-1)
				message=format("No Dirty Cache Lines,\nSend Source Done\nMessage");
				cpudialogBox.setCurrentTask(message);
				//take the data directly from the memory
				cpuCache.contents[address%2]=cpu[address%NCPU].cpuMemory.mem[address/4];
		else
//otherwise you must go to shared and take the data from an up to date cache
				message=format("Dirty Cache Line\nat Node %d,\nSend Source Done\n Message",dataLine);
				cpudialogBox.setCurrentTask(message);
				cpuCache.contents[address%2]=cpu[dataLine].cpuCache.contents[address%2];
		end;
				if(doWrite==WRITE)
						cpuCache.state[address%2]=EXCLUSIVE;
						cpuCache.stateRTxt[address%2].setTxt("E");
				else
						cpuCache.state[address%2]=SHARED;
						cpuCache.stateRTxt[address%2].setTxt("S");
				end;

end;
//set the address and data in the cache and highlight
cpuCache.address[address%2]=address;
cpuCache.aRTxt[address%2].setTxt("a%d",address);
cpuCache.dRTxt[address%2].setTxt("%d",cpuCache.contents[address%2]);
cpuCache.highlight(address%2,0);

//Send finished message to home cpu
//in case this is also a write and if its exclsuive it need to be added at the
//same time as the linkpath
//
//fork it later
//
if(doWrite==WRITE)
		cpuDataLines2.moveDown(TICKS, 1);
		cpuDataLines2.moveUp(TICKS, 1);
		cpuCache.contents[address%2]++;
		cpuCache.dRTxt[address%2].setTxt("%d",cpuCache.contents[address%2]);
		cpuCache.state[address%2]=MODIFIED;
		cpuCache.stateRTxt[address%2].setTxt("M");
end;
cpuDataLines2.moveDown(TICKS, 1);
linkPath(cpuNo,address%NCPU,0,brushsH[cpuNo],brushsH[cpuNo],address,-1,"SD:a");
if(doWrite==WRITE)
		invalidateRestart[address]=0;
```

```
                                        end;
                                        homeNodeQueue[address].removeProcess();
                                end;
                                //if you wish to write to the data ie add one to it
                                if(doWrite==WRITE && read==0)
                                        //variable to see if we had to add a process to the address queue
                                        processAdded=0;

                                        //you should have the data stored locally but you have to invalidate external copies
                                        //if its in shared or owned
                                        if(cpuCache.state[address%2]==SHARED || cpuCache.state[address%2]==OWNED)
                                                invalidating=address;
                                                //add the process to the address queue and set process added to 1
                                                processAdded=1;
                                        //sent to invalidate message to the home node
                                        //add 10 to the cpuNo for all invalidate messages
                                        linkPath(cpuNo,address%NCPU,0,brushsH[cpuNo],brushsH[cpuNo],address,cpuNo+10,"WR");
                                        if(invalidateRestart[address]==1)
                                                while(cpuRestart==0)
                                                        wait(1);
                                                end;
                                                cpuLock=0;
                                                MOESI(address,WRITE);
                                                return;
                                        else
                                                invalidateRestart[address]=1;
                                                        //send the invalidate message from the home node to all the caches

                                                        broadcast(address%NCPU,cpuNo,address,1,-1,"PRI:a");
                                                        while(repliesReturned!=5)
                                                                wait(1);
                                                        end;
                                                        invalidateRestart[address]=0;
                                                        invalidating=-1;
                                        end;
                                        //add one to the data and store it and set to modified
                                                cpuCache.contents[address%2]++;
                                                cpuCache.dRTxt[address%2].setTxt("%d",cpuCache.contents[address%2]);
                                                cpuCache.state[address%2]=MODIFIED;
                                                cpuCache.stateRTxt[address%2].setTxt("M");
                                                //if the data had to be flushed earlier ie a process was added to the memory queue,
                                                //send a done message to the home node and remove the process from the queue
                                                if(processAdded==1)
                                                        linkPath(cpuNo,address%NCPU,0,brushsH[cpuNo],brushsH[cpuNo],address,-1,"SD:a");
                                                        homeNodeQueue[address].removeProcess();
                                                end;
                                        else
                                                localRorW[address]=1;
                                                cpuDataLines2.moveDown(TICKS, 1);
                                                cpuDataLines2.moveUp(TICKS, 1);
                                                cpuCache.contents[address%2]++;
                                                cpuCache.dRTxt[address%2].setTxt("%d",cpuCache.contents[address%2]);
                                                cpuCache.state[address%2]=MODIFIED;
                                                cpuCache.stateRTxt[address%2].setTxt("M");
                                                localRorW[address]=0;
                                        end;

                        end;
                        cacheLineFound=-1;

                        cpudialogBox.setTitle("Last Action");
                        if(doWrite==WRITE)
                                message=format("Write a%d",address);
                                cpudialogBox.setCurrentTask(message);
                        else
                                message=format("Read a%d",address);
                                cpudialogBox.setCurrentTask(message);
                        end;
                        cpuLock = 0;
                        checkPoint();
        end;
end;

        class rwbuttons(real x, real y,real w,real h,int address)
                background=Rectangle2(0, 0, blackpen, brushsH[address%NCPU], x,y,w,h,blackbrush,0);
                buttonTxt=Txt(0, 0, x+w/2, y+h/2, blackbrush, 0, "A20");
                readButton=Polygon(0, AAFILL | ABSOLUTE, blackpen, brushsH[address%NCPU],  x,y,  0,0,  w,0,   0,h);
                writeButton=Polygon(0, AAFILL | ABSOLUTE, blackpen, brushsH[address%NCPU],  x,y+h,  0,0,  w,0,   w,-h);
                rtxt = Txt(0, HLEFT | VTOP, x, y, blackbrush, 0, "R");
                wtxt = Txt(0, HRIGHT | VBOTTOM, x+(w*1.05), y+(h*1.1), blackbrush, 0, "W");
                rtxt.setSize(w*0.66,h*0.66);
                wtxt.setSize(w*0.66,h*0.66);
                buttonTxt.setSize(w*0.8,h*0.8);
                buttonTxt.setTxt("A%d",address);
                 readButton.setOpacity(1);
                writeButton.setOpacity(1);
                wtxt.setOpacity(1);
```

```
		rtxt.setOpacity(1);
		int inUse;


		when background.eventEE(int flags, real x, real y)
			if(inUse==0)
			 readButton.setOpacity(flags & EB_ENTER ? 255 : 1);
			 writeButton.setOpacity(flags & EB_ENTER ? 255 : 1);
			 rtxt.setOpacity(flags & EB_ENTER ? 255 : 1);
			 wtxt.setOpacity(flags & EB_ENTER ? 255 : 1);
			end;

		end;
		when readButton.eventEE(int flags, real x, real y)
			if(inUse==0)
			 readButton.setBrush(flags & EB_ENTER ? brushsS[address%NCPU] :  brushsH[address%NCPU]);
			end;
		end;
		when writeButton.eventEE(int flags, real x, real y)
			if(inUse==0)
			 writeButton.setBrush(flags & EB_ENTER ? brushsS[address%NCPU] :  brushsH[address%NCPU]);
			end;
		end;

		when readButton.eventLB(int flags, real x, real y)
		if (flags & EB_LEFT==0 && inUse==0 )
					inUse=1;
					start(1);

					readButton.setOpacity(255);
					rtxt.setOpacity(255);
					writeButton.setOpacity(255);
					wtxt.setOpacity(255);
					readButton.setBrush( brushsS[address%NCPU]);
		MOESI(address,READ);
					readButton.setBrush( brushsH[address%NCPU]);
					readButton.setOpacity(1);
					rtxt.setOpacity(1);
					writeButton.setOpacity(1);
					wtxt.setOpacity(1);
					inUse=0;
		end;
		end;
		when writeButton.eventLB(int flags, real x, real y)
		if (flags & EB_LEFT==0 && inUse==0)
					inUse=1;
					while(EB_CTRL==1)
						wait(1);
					end;
					readButton.setOpacity(255);
					rtxt.setOpacity(255);
					writeButton.setOpacity(255);
					wtxt.setOpacity(255);
					writeButton.setBrush( brushsS[address%NCPU]);
		MOESI(address,WRITE);
					readButton.setBrush( brushsH[address%NCPU]);
					readButton.setOpacity(1);
					rtxt.setOpacity(1);
					writeButton.setOpacity(1);
					wtxt.setOpacity(1);
					inUse=0;
		end;
		end;


	end;




	cpuMemory= Memory();
	cpuCache=Cache();
	cpuDataLines=DataLines(x+w2*0.3,y+h*0.425);
	cpuDataLines2=DataLines(x+w2*0.3,y+h*0.685);
	cpudialogBox=dialogBox();

	r = Rectangle2(0, 0, blackpen, gray192brush,x+gap,y+h*0.684, w2, h*0.4);
	tempAddress=0;
	for(i=0;i<4;i++)
		for(int j=0;j<NCPU;j++)
			rwbuttons(x+gap*2.5+j*(h2*1.25+gap),y+h*0.684+gap+i*(h2*1.25+gap),h2*1.25,h2*1.25,tempAddress);
			tempAddress++;
		end;
	end;
end;
```

```
cpu[0]=CPU(50+offset*2,50+offset,0);
cpu[1]=CPU(1200+offset*2,50+offset,1);
cpu[2]=CPU(50+offset*2,1200+offset,2);
cpu[3]=CPU(1200+offset*2,1200+offset,3);
cpu[4]=CPU(2349+offset*2,1200+offset,4);

function checkCache(int cpuA,int cpuB,int address,int flag,int invalidate,int messageLock)

	while(cpu[cpuB].localRorW[address]==1)
		wait(1);
	end;

//if its not an invalidate probe
	if(invalidate==0)
		//if the line is not held in cpuB or is invalid do nothing
		if(cpu[cpuB].cpuCache.address[address%2]==address && cpu[cpuB].cpuCache.state[address%2]!=INVALID )
			//if its exclusive set it to shared
			if(cpu[cpuB].cpuCache.state[address%2]==EXCLUSIVE)
				cpu[cpuB].cpuCache.state[address%2]=SHARED;
				cpu[cpuB].cpuCache.stateRTxt[address%2].setTxt("S");
				cpu[cpuB].cpuCache.highlight(address%2,0);
			end;
			//if its modified set it to Owned
			if(cpu[cpuB].cpuCache.state[address%2]==MODIFIED)
				cpu[cpuB].cpuCache.state[address%2]=OWNED;
				cpu[cpuB].cpuCache.stateRTxt[address%2].setTxt("O");
				cpu[cpuB].cpuCache.highlight(address%2,0);
			end;
			//as its not invalid must let the source node know that its cant go to exclusive
			//for the purpose of simplicity im going copy any up to date copy of the cache line even though
			//in practice it will only take a copy from modified or owned
			cpu[cpuA].cacheLineFound=cpuB;

			//this protocol sends a read response only on modified or owned line and a probe reponse otherwise
			if(cpu[cpuB].cpuCache.state[address%2]==MODIFIED || cpu[cpuB].cpuCache.state[address%2]==OWNED)
				//cpu[cpuB].cpuCache.highlight(address%2, 0);
				cpu[cpuA].dataLine=cpuB;//cpu[cpuB].cpuCache.contents[address%2];
				cpu[cpuB].cpuCache.highlight(address%2,0);
				message=format("D-%d:a",cpu[cpuB].cpuCache.contents[address%2]);//cpu[cpuA].dataLine);
				linkPath(cpuB,cpuA,flag,brushsH[cpuB],brushsH[cpuB],address,messageLock,message);
				cpu[cpuA].repliesReturned++;
				message=format("Home Node %d\nprobes for a%d\n Acks Returned:%d/5",
				address%NCPU,address,cpu[cpuA].repliesReturned-1);
				cpu[cpuA].cpudialogBox.setCurrentTask(message);
				return;
			end;
		end;
		//if not modified or owned sent a probe response
		linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,messageLock,"IA:a");
		cpu[cpuA].repliesReturned++;
		message=format("Home Node %d\nprobes for a%d\n Acks Returned:%d/5", address%NCPU,address,cpu[cpuA].repliesReturned-1);
		cpu[cpuA].cpudialogBox.setCurrentTask(message);
	end;
	if(invalidate==1)
		int lineFound=0;
		if(cpuA!=cpuB && cpu[cpuB].cpuCache.address[address%2]==address && cpu[cpuB].cpuCache.state[address%2]!=INVALID )
			cpu[cpuA].cacheLineFound=cpuB;
			if(cpu[cpuB].cpuCache.state[address%2]==MODIFIED || cpu[cpuB].cpuCache.state[address%2]==OWNED)
				cpu[cpuA].dataLine=cpuB;//cpu[cpuB].cpuCache.contents[address%2];
				lineFound=1;
			end;
			cpu[cpuB].cpuCache.state[address%2]=INVALID;
			cpu[cpuB].cpuCache.stateRTxt[address%2].setTxt("I");
			cpu[cpuB].cpuCache.highlight(address%2, 1);
			if(invalidateRestart[address]==1 && cpu[cpuB].invalidating==address)
				cpu[cpuB].cpuRestart=1;
			end;
		end;
		if(lineFound==1)
			message=format("D-%d:",cpu[cpuB].cpuCache.contents[address%2]);//cpu[cpuA].dataLine);
			linkPath(cpuB,cpuA,flag,brushsH[cpuB],brushsH[cpuB],address,messageLock,message);
		else
			linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,messageLock,"IA:a");
		end;
			cpu[cpuA].repliesReturned++;
			message=format("Home Node %d\nprobes for a%d\n Acks Returned:%d/5", address%NCPU,address,cpu[cpuA].repliesReturned-1);
			cpu[cpuA].cpudialogBox.setCurrentTask(message);

	end;

end;

class p2p(int Ax,int Ay,int l, int cpuA, int cpuB,int orientation,int linkNo)

	int currentPacketUpNo=0;
	int currentPacketDownNo=0;
```

```
int waitingUp=0;
int waitingDown=0;
int packageW=120;
int packageH=40;
int arrowH=15;
int arrowW=30;
int xGap=200;
int yGap=105;
int boxOffsetX=20;
int boxOffsetY=15;

for(int i=0;i<5;i++)
	lock[i]=0;
end;
packetFont = Font("Times New Roman", 30, 0);
if(orientation==0)
	httUp = Polygon(0, AAFILL | ABSOLUTE, blackpen, gray192brush,  Ax,Ay-5, 0,0, l-arrowW,0,  l-arrowW,-arrowH,  l,arrowH/2,
	l-arrowW,arrowH*2, l-arrowW,arrowH, 0,arrowH );
	httDown = Polygon(0, AAFILL | ABSOLUTE, blackpen, gray192brush,  Ax,Ay+yGap-5, 0,arrowH/2, arrowW,-arrowH, arrowW,0, l,0, l,arrowH,
	arrowW,arrowH, arrowW,arrowH*2);
	for(i=0;i<5;i++)
		pUpB[i]=Ellipse(0, AAFILL, redpen, gray192brush, Ax, Ay+10,0,0, packageW+(boxOffsetX*2), packageH+(boxOffsetY*2));
		pUp[i] = Rectangle2(0, AAFILL, redpen, gray192brush, Ax+15,Ay,packageW,packageH,blackbrush,packetFont,0);
		pDownB[i]=Ellipse(0, AAFILL, redpen, gray192brush, Ax, Ay+50,0,0, packageW+(boxOffsetX*2), packageH+(boxOffsetY*2));
		pDown[i] = Rectangle2(0, AAFILL, redpen, gray192brush, Ax,Ay+50,packageW,packageH,blackbrush,packetFont,0);

		pUp[i].moveToBack();
		pDown[i].moveToBack();
		pUpB[i].moveToBack();
		pDownB[i].moveToBack();
		pUpB[i].setOpacity(0);
		pDownB[i].setOpacity(0);
		pUp[i].setOpacity(0);
		pDown[i].setOpacity(0);
	end;
else
	httUp = Polygon(0, AAFILL | ABSOLUTE, blackpen, gray192brush,  Ax+55,Ay, 0,0, arrowH,0, arrowH,l-arrowW, arrowH*2,l-arrowW,
	arrowH/2,l, -arrowH,l-arrowW, 0,l-arrowW);
	httDown = Polygon(0, AAFILL | ABSOLUTE, blackpen, gray192brush,  Ax+52+xGap,Ay, arrowH/2,0, arrowH*2,arrowW, arrowH,arrowW,
	arrowH,l, 0,l, 0,arrowW, -arrowH,arrowW );

	for(i=0;i<5;i++)
		pUpB[i]=Ellipse(0, AAFILL, redpen, gray192brush, Ax, Ay,0,0, packageW+(boxOffsetX*2), packageH+(boxOffsetY*2));
		pUp[i] = Rectangle2(0,AAFILL, redpen, gray192brush, Ax,Ay, packageW,packageH,blackbrush,packetFont,0);
		pDownB[i]=Ellipse(0, AAFILL,redpen, gray192brush, Ax+80, Ay,0,0 ,packageW+(boxOffsetX*2), packageH+(boxOffsetY*2));
		pDown[i] = Rectangle2(0, AAFILL, redpen, gray192brush, Ax+80,Ay, packageW,packageH,blackbrush,packetFont,0);

		pUp[i].moveToBack();
		pDown[i].moveToBack();
		pUpB[i].moveToBack();
		pDownB[i].moveToBack();
		pUpB[i].setOpacity(0);
		pDownB[i].setOpacity(0);
		pUp[i].setOpacity(0);
		pDown[i].setOpacity(0);
	end;
end;
	httUp.moveToBack();
	httDown.moveToBack();
function forkPackageUp(int packageNo,int xp,int yp,int ticks,int address,int messageLock)
	// pUp must wait until it finished before releasing the linkLock
	fork(pUp[packageNo].setPos(xp,yp,ticks , 1, 1));
	pUpB[packageNo].setPos(xp-boxOffsetX,yp-boxOffsetY,ticks , 1, 1);
	message=format("Waiting at\nHome Node %d,\nfor access to a%d",address%NCPU,address);
	if(messageLock>=10)
		cpu[messageLock-10].cpudialogBox.setCurrentTask(message);
	else
		cpu[messageLock].cpudialogBox.setCurrentTask(message);
	end;
	linkLockUp[linkNo]=0;
end;
function forkPackageDown(int packageNo,int xp,int yp,int ticks,int address,int messageLock)
	// pDown must wait until it finished before releasing the linkLock
	fork(pDownB[packageNo].setPos(xp-boxOffsetX,yp-boxOffsetY,ticks , 1, 1));
	pDown[packageNo].setPos(xp,yp,ticks , 1, 1);
	message=format("Waiting at\nHome Node %d,\nfor access to a%d",address%NCPU,address);
	if(messageLock>=10)
		cpu[messageLock-10].cpudialogBox.setCurrentTask(message);
	else
		cpu[messageLock].cpudialogBox.setCurrentTask(message);
	end;
	linkLockDown[linkNo]=0;
end;
function packetUp(int ticks,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message)
	pUpB[currentPacketUpNo].setOpacity(255);
	pUpB[currentPacketUpNo].setBrush(brush1);
	pUp[currentPacketUpNo].setOpacity(255);
```

```
pUp[currentPacketUpNo].setBrush(brush2);
tempMessage=format("%d",address);
tempMessage=message+tempMessage;
//pUp[currentPacketUpNo].setTxt("werr");
pUp[currentPacketUpNo].setTxt(tempMessage);
if(orientation==0)
	pUpB[currentPacketUpNo].setPos( Ax-packageW-2*boxOffsetX,Ay-packageH/2-boxOffsetY);
	pUp[currentPacketUpNo].setPos( Ax-packageW-boxOffsetX,Ay-packageH/2);
	if(messageLock==-1)
		fork(pUpB[currentPacketUpNo].setPos(Ax+l,Ay-packageH/2-boxOffsetY, ticks, 1, wflag));
		pUp[currentPacketUpNo].setPos(Ax+l+boxOffsetX, Ay-packageH/2, ticks, 1, wflag);
	else

	//add the process inside here and remove it in the main body
		homeNodeQueue[address].addProcess(messageLock);
		tempPackageNo=currentPacketUpNo;
		enterIf=0;
		if(homeNodeQueue[address].returnProcess()!=messageLock)
			enterIf=1;
			currentPacketUpNo++;
			if(currentPacketUpNo==5)
				currentPacketUpNo=0;
			end;
			waitingUp++;
			int xp=Ax+boxOffsetX+l-((packageW+2*boxOffsetX)*waitingUp);
			int yp=Ay-packageH/2;
			int tempWaiting=waitingUp;
			forkPackageUp(tempPackageNo,xp,yp ,ticks,address,messageLock);

		end;
		int exitLoop=0;
		while(homeNodeQueue[address].returnProcess()!=messageLock  && exitLoop==0)
			wait(1);

			if(tempWaiting>waitingUp)
				xp+=packageW+2*boxOffsetX;
				fork(pUpB[tempPackageNo].setPos(xp-boxOffsetX, yp-boxOffsetY, ticks, 1, wflag));
				pUp[tempPackageNo].setPos(xp, yp, ticks, 1, wflag);
			end;
			tempWaiting=waitingUp;
			if(invalidateRestart[address]==1 && messageLock>=10)
				exitLoop=1;
				homeNodeQueue[address].deleteProcess(messageLock);
			end;
		end;
		if(enterIf==1)
			waitingUp--;
		end;
		fork(pUpB[tempPackageNo].setPos(Ax+l+boxOffsetX, Ay-boxOffsetY-packageH/2, ticks, 1, wflag));
		pUp[tempPackageNo].setPos(Ax+l+boxOffsetX*2, Ay-packageH/2, ticks, 1, wflag);
	end;
else
	pUpB[currentPacketUpNo].setPos( Ax-boxOffsetX,Ay-packageH-boxOffsetY);
	pUp[currentPacketUpNo].setPos( Ax,Ay-packageH);
	if(messageLock==-1 )
		fork(pUpB[currentPacketUpNo].setPos(Ax-boxOffsetX, Ay+l, ticks, 1, wflag));
		pUp[currentPacketUpNo].setPos(Ax, Ay+l+boxOffsetY, ticks, 1, wflag);
	else
		homeNodeQueue[address].addProcess(messageLock);
		tempPackageNo=currentPacketUpNo;
		enterIf=0;
		if(homeNodeQueue[address].returnProcess()!=messageLock)
			enterIf=1;
			currentPacketUpNo++;
			if(currentPacketUpNo==5)
				currentPacketUpNo=0;
			end;
			waitingUp++;

			xp=Ax;
			yp=Ay+l+boxOffsetY-((packageH+boxOffsetY*2)*waitingUp);
			tempWaiting=waitingUp;
			forkPackageUp(tempPackageNo,xp,yp ,ticks,address,messageLock);
		end;
		exitLoop=0;
		while(homeNodeQueue[address].returnProcess()!=messageLock  && exitLoop==0)
			wait(1);
			if(tempWaiting>waitingUp)
				yp+=packageH+boxOffsetY*2;
				fork(pUpB[tempPackageNo].setPos(xp-boxOffsetX, yp-boxOffsetY, ticks, 1, wflag));
				pUp[tempPackageNo].setPos(xp, yp, ticks, 1, wflag);
			end;
			tempWaiting=waitingUp;
			if(invalidateRestart[address]==1 && messageLock>=10)
				exitLoop=1;
				homeNodeQueue[address].deleteProcess(messageLock);
			end;
```

```
                                        end;
                                        if(enterIf==1)
                                        waitingUp--;
                                        end;
                                        fork(pUpB[tempPackageNo].setPos(Ax-boxOffsetX, Ay+l+packageH, ticks, 1, wflag));
                                        pUp[tempPackageNo].setPos(Ax, Ay+l+packageH+boxOffsetY, ticks, 1, wflag);
                                end;
                        end;

end;

function packetDown(int ticks,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message)
                        pDownB[currentPacketDownNo].setOpacity(255);
                        pDownB[currentPacketDownNo].setBrush(brush1);
                        pDown[currentPacketDownNo].setOpacity(255);
                        pDown[currentPacketDownNo].setBrush(brush2);
                        tempMessage=format("%d",address);
                        tempMessage=message+tempMessage;
                        pDown[currentPacketDownNo].setTxt(tempMessage);
                        if(orientation==0)
                                pDownB[currentPacketDownNo].setPos( Ax+l+boxOffsetX,Ay-packageH/2+yGap-boxOffsetY);
                                pDown[currentPacketDownNo].setPos( Ax+l+2*boxOffsetX,Ay-packageH/2+yGap);
                                if(messageLock==-1)
                                        fork(pDownB[currentPacketDownNo].setPos(Ax-packageW-boxOffsetX*2, Ay-packageH/2+yGap-boxOffsetY, ticks, 1,
                                        wflag));
                                        pDown[currentPacketDownNo].setPos(Ax-packageW-boxOffsetX, Ay-packageH/2+yGap, ticks, 1, wflag);
                                else
                                        homeNodeQueue[address].addProcess(messageLock);
                                        int tempPackageNo=currentPacketDownNo;
                                        int enterIf=0;
                                        if(homeNodeQueue[address].returnProcess()!=messageLock)
                                                enterIf=1;
                                                currentPacketDownNo++;
                                                if(currentPacketDownNo==5)
                                                        currentPacketDownNo=0;
                                                end;
                                                waitingDown++;

                                                int xp=Ax-packageW-boxOffsetX+((packageW+boxOffsetX*2)*waitingDown);
                                                int yp=Ay-packageH/2+yGap;
                                                int tempWaiting=waitingDown;
                                                forkPackageDown(tempPackageNo,xp,yp ,ticks,address,messageLock);
                                        end;
                                        int exitLoop=0;
                                        while(homeNodeQueue[address].returnProcess()!=messageLock  && exitLoop==0)
                                                wait(1);
                                                if(tempWaiting>waitingDown)
                                                        xp-=packageW+boxOffsetX*2;
                                                        fork(pDownB[tempPackageNo].setPos(xp-boxOffsetX, yp-boxOffsetY, ticks, 1, wflag));
                                                        pDown[tempPackageNo].setPos(xp, yp, ticks, 1, wflag);
                                                end;
                                                tempWaiting=waitingDown;
                                                if(invalidateRestart[address]==1 && messageLock>=10)
                                                        exitLoop=1;
                                                        homeNodeQueue[address].deleteProcess(messageLock);
                                                end;
                                        end;
                                        if(enterIf==1)
                                                waitingDown--;
                                        end;
                                        fork(pDownB[tempPackageNo].setPos(Ax-packageW-boxOffsetX*2, Ay-packageH/2+yGap-boxOffsetY, ticks, 1,
                                        wflag));
                                        pDown[tempPackageNo].setPos(Ax-packageW-boxOffsetX, Ay-packageH/2+yGap, ticks, 1, wflag);

                                end;
                        else
                                pDownB[currentPacketDownNo].setPos( Ax+xGap-boxOffsetX,Ay+l);
                                pDown[currentPacketDownNo].setPos( Ax+xGap,Ay+l+boxOffsetY);
                                if(messageLock==-1 )
                                fork(pDownB[currentPacketDownNo].setPos(Ax+xGap-boxOffsetX, Ay-packageH-boxOffsetY*2, ticks, 1, wflag));
                                pDown[currentPacketDownNo].setPos(Ax+xGap, Ay-packageH-boxOffsetY, ticks, 1, wflag);
                                else
                                        homeNodeQueue[address].addProcess(messageLock);
                                        tempPackageNo=currentPacketDownNo;
                                        enterIf=0;
                                        if(homeNodeQueue[address].returnProcess()!=messageLock)
                                                enterIf=1;
                                                currentPacketDownNo++;
                                                if(currentPacketDownNo==5)
                                                        currentPacketDownNo=0;
                                                end;
                                                waitingDown++;

                                                xp=Ax+xGap;
                                                yp=Ay+((packageH+boxOffsetY*2)*waitingDown)-packageH-boxOffsetY;
                                                tempWaiting=waitingDown;
                                                forkPackageDown(tempPackageNo,xp,yp ,ticks,address,messageLock);
```

```
                                        end;
                                        exitLoop=0;
                                        while(homeNodeQueue[address].returnProcess()!=messageLock  && exitLoop==0)
                                                wait(1);
                                                if(tempWaiting>waitingDown)
                                                        yp-=packageH+boxOffsetY*2;
                                                        fork(pDownB[tempPackageNo].setPos(xp-boxOffsetX, yp-boxOffsetY, ticks, 1, wflag));
                                                        pDown[tempPackageNo].setPos(xp, yp, ticks, 1, wflag);
                                                end;
                                                tempWaiting=waitingDown;
                                                if(invalidateRestart[address]==1 && messageLock>=10)
                                                        exitLoop=1;
                                                        homeNodeQueue[address].deleteProcess(messageLock);
                                                end;
                                        end;
                                        if(enterIf==1)
                                                waitingDown--;
                                        end;
                                        fork(pDownB[tempPackageNo].setPos(Ax+xGap-boxOffsetX, Ay-packageH-boxOffsetY*2, ticks, 1, wflag));
                                        pDown[tempPackageNo].setPos(Ax+xGap, Ay-packageH-boxOffsetY, ticks, 1, wflag);
                                end;
                        end;
                end;


end;

link[0]=p2p(471+offset*2,285+offset,709     ,0,1  ,0 ,0);
link[1]=p2p(90+offset*2,828+offset,370     ,0,2  ,1 ,1);
link[2]=p2p(1250+offset*2,828+offset,370    ,1,3  ,1 ,2);
link[3]=p2p(471+offset*2,1450+offset,709    ,2,3  ,0 ,3);
link[4]=p2p(1619+offset*2,1450+offset,709   ,3,4  ,0 ,4);


function linkPathUp(int linkNo,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message)


        while(linkLockUp[linkNo]==1)
                wait(1);
        end;

                linkLockUp[linkNo]=1;
                link[linkNo].packetUp(TICKS*2,wflag,brush1,brush2,address,messageLock,message);
                wait(TICKS*2);
                linkLockUp[linkNo]=0;

end;

function linkPathDown(int linkNo,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message)

        while(linkLockDown[linkNo]==1)
                wait(1);
        end;
                linkLockDown[linkNo]=1;
                link[linkNo].packetDown(TICKS*2,wflag,brush1,brush2,address,messageLock,message);
                wait(TICKS*2);
                linkLockDown[linkNo]=0;
end;

function linkPath(int cpuA,int cpuB,int wflag,Brush brush1,Brush brush2,int address,int messageLock,string message)
        lockMessage=-1;

        if(cpuA!=cpuB)
                if(cpuA==0)

                        if(cpuB==1 || cpuB==3 || cpuB==4)
                                if(cpuB==1)
                                        lockMessage=messageLock;
                                end;
                                linkPathUp(0,wflag,brush1,brush2,address,lockMessage,message);
                                if(cpuB==3 || cpuB==4)
                                        if(cpuB==3)
                                                lockMessage=messageLock;
                                        end;

                                        linkPathUp(2,wflag,brush1,brush2,address,lockMessage,message);
                                end;
                                if(cpuB==4)
                                        linkPathUp(4,wflag,brush1,brush2,address,messageLock,message);
                                end;
                        end;

                        if(cpuB==2)
                                linkPathUp(1,wflag,brush1,brush2,address,messageLock,message);
                        end;

                end;
```

```
if(cpuA==1)

	if(cpuB==3 || cpuB==4 )
		if(cpuB==3)
			lockMessage=messageLock;
		end;
		linkPathUp(2,wflag,brush1,brush2,address,lockMessage,message);
		if(cpuB==4)
			linkPathUp(4,wflag,brush1,brush2,address,messageLock,message);
		end;

	end;

	if(cpuB==0 || cpuB==2)
		if(cpuB==0)
			lockMessage=messageLock;
		end;
		linkPathDown(0,wflag,brush1,brush2,address,lockMessage,message);
		if(cpuB==2)
			linkPathUp(1,wflag,brush1,brush2,address,messageLock,message);
		end;
	end;

end;


if(cpuA==2)

	if(cpuB==3 || cpuB==4)
		if(cpuB==3)
			lockMessage=messageLock;
		end;
		linkPathUp(3,wflag,brush1,brush2,address,lockMessage,message);
		if(cpuB==4)
			linkPathUp(4,wflag,brush1,brush2,address,messageLock,message);
		end;
	end;

	if(cpuB==0 || cpuB==1)
		if(cpuB==0)
			lockMessage=messageLock;
		end;
		linkPathDown(1,wflag,brush1,brush2,address,lockMessage,message);
		if(cpuB==1)
			linkPathUp(0,wflag,brush1,brush2,address,messageLock,message);
		end;
	end;

end;

if(cpuA==3)

	if(cpuB==1 || cpuB==0)
		if(cpuB==1)
			lockMessage=messageLock;
		end;
		linkPathDown(2,wflag,brush1,brush2,address,lockMessage,message);
		if(cpuB==0)
			linkPathDown(0,wflag,brush1,brush2,address,messageLock,message);
		end;

	end;
	if(cpuB==4)
		linkPathUp(4,wflag,brush1,brush2,address,messageLock,message);
	end;
	if(cpuB==2)
		linkPathDown(3,wflag,brush1,brush2,address,messageLock,message);
	end;

end;

if(cpuA==4)
	if(cpuB==3)
		lockMessage=messageLock;
	end;
	linkPathDown(4,wflag,brush1,brush2,address,lockMessage,message);
	if(cpuB==1 || cpuB==0 )
		if(cpuB==1)
			lockMessage=messageLock;
		end;
		linkPathDown(2,wflag,brush1,brush2,address,lockMessage,message);
		if(cpuB==0)
			linkPathDown(0,wflag,brush1,brush2,address,messageLock,message);
		end;

	end;
```

```
                                        if(cpuB==2)
                                                linkPathDown(3,wflag,brush1,brush2,address,messageLock,message);
                                        end;


                        end;
                else
                        int exitLoop=0;
                        if(messageLock!=-1)
                                homeNodeQueue[address].addProcess(messageLock);
                                while(homeNodeQueue[address].returnProcess()!=messageLock && exitLoop==0)
                                        wait(1);
                                        if(invalidateRestart[address]==1 && messageLock>=10)
                                                exitLoop=1;
                                                homeNodeQueue[address].deleteProcess(messageLock);
                                        end;
                                end;
                        end;

                end;

end;

resetFont =Font("Times New Roman", 40, 0);
resetButton = SimpleButton(LOGICALW-215, LOGICALH-75, 160, 60, bgbrush, gray192brush, blackbrush,resetFont , "Reset");
when resetButton.button.eventLB(int flags, real x, real y)
        if (flags & EB_LEFT == 0)

                reset();
        end;
end;
```

9.2 – MESIF.viv

```
//
// Vivio MESIF cache coherency protocol animation

//
// include files
//
#include "standard.vin"
#include "simpleButton.vin"
#include "imageButton.vin"

const LOGICALW = 3200;
const LOGICALH = 2100;
ctrlRect = Rectangle2(0, 0, blackpen, whitebrush,  0, 0, 3200, 2100);
ctrlRect.setOpacity(1);
setViewport(0, 0, LOGICALW, LOGICALH, 1);


//
// fonts
//
smallFont = Font("Times New Roman", 14, 0);
hintFont = Font("Times New Roman", 16, 0);
titleFont =
 Font("Times New Roman", 24, 0);


//
// set background
//
bgbrush = SolidBrush(vellum);
setBgBrush(bgbrush);
navybrush = SolidBrush(rgb(0, 0, 102));

//colours of memory
const int coloursH[5]={rgb(255, 255,  150),rgb(255, 183,  183),rgb(191, 255,  183),rgb(155, 182,  255),rgb(255, 204, 51)};
const int coloursS[5]={rgb(255, 255, 0),rgb(200, 0, 0), rgb(0, 255, 0),rgb(0, 0, 255),rgb(245, 184, 0)};

//brushs for memory
Brush brushsH[5];
Brush brushsS[5];
brushsH[0]= SolidBrush(coloursH[0]);
brushsH[1]= SolidBrush(coloursH[1]);
brushsH[2]= SolidBrush(coloursH[2]);
brushsH[3]= SolidBrush(coloursH[3]);
brushsH[4]= SolidBrush(coloursH[4]);
brushsS[0]= SolidBrush(coloursS[0]);
brushsS[1]= SolidBrush(coloursS[1]);
brushsS[2]= SolidBrush(coloursS[2]);
```

```
brushsS[3]= SolidBrush(coloursS[3]);
brushsS[4]= SolidBrush(coloursS[4]);
smallFont = Font("Times New Roman", 14, 0);
hintFont = Font("Times New Roman", 50, 0);
titleFont = Font("Times New Roman", 50, 0);



//A lock is needed for every memory location as multiple processors may access it at the same time
int memLock[20];
for(j=0;j<20;j++)
		memLock[j]=0;
end;

//
// title
//
navybrush = SolidBrush(rgb(0, 0, 102));
//title = Rectangle2(0, VCENTRE, 0, navybrush, 1005, 0, LOGICALW/3, 30, whitebrush, titleFont, "MOESI Cache Coherency Protocol");
//title.setTxtOff(2, 1);
offset=50;

//
// title
//
infoRect = Rectangle2(0, 0, blackpen, whitebrush,  LOGICALW-920+offset*2, 450, 520, 600);
hintFont2 = Font("Times New Roman", 40, 0);
navybrush = SolidBrush(rgb(0, 0, 102));
title = Rectangle2(0, VCENTRE, 0, navybrush, LOGICALW-1500+offset*2, 100, 1000, 250, whitebrush, titleFont,"MESIF Cache Coherency Protocol");
title.setTxtOff(2, 1);

//information box
str1 =  "Packet Information\n";
str += "READ-Read from memory\n		request\n";
str += "CNCL-Cancel memory\n		access\n";
str += "ACK-acknowledge to\n		remove Lock\n";
str += "PR-Probe\n";
str += "PRI-Probe invalidate\n";
str += "FL-Flush\n";
str += "FLA-Flush acknowledge\n";
//str += "M-memory data\n";
str += "DataX-data in state X\n";

hint = Txt(0, HLEFT | VTOP, LOGICALW-890+offset*2, 450, redbrush, hintFont, str1);
hint2 = Txt(0, HLEFT | VTOP, LOGICALW-860+offset*2, 520, blackbrush, hintFont2, str);


const int NCPU = 5;					// number of cpus
const int TICKS = 20;			// animation speed
const int INVALID = 0;			// M0ESI cache line states
const int SHARED = 1;
const int EXCLUSIVE = 2;
const int MODIFIED = 3;
const int FORWARD = 4;


int currentProcessNo=0;
int DRAMLock[5];
int invalidateRestart[20];


class Queue()

		int endPosition=0;
		int queue[5];
		int queueActive=0;

		for(int i=0;i<5;i++)
			queue[i]=-1;
		end;

		function addProcess(int cpuNo)
			queue[endPosition]=cpuNo;
			queueActive=1;
			endPosition++;
			if(endPosition==5)
				endPosition--;
			end;
		end;
		function removeProcess()
			for(i=0;i<4;i++)
				queue[i]=queue[i+1];
			end;
			endPosition--;
			if(endPosition==0)
				queueActive=0;
			end;
```

```
                    if(endPosition==-1)
                        endPosition=0;
                    end;
            end;
            int function checkProcess(int cpuNo)
                    for(i=0;i<endPosition;i++)
                        if(queue[i]==cpuNo)
                            return 1;
                        end;
                    end;
                    return 0;
            end;

            function emptyQueue()
                    endPosition=0;
                    queueActive=0;
            end;

            function deleteProcess(int cpuNo)
                    for(i=0;i<endPosition;i++)
                        if(queue[i]==cpuNo)
                            for(j=i;j<endPosition;j++)
                                queue[j]=queue[j+1];
                            end;
                            endPosition--;
                            if(endPosition==0)
                                queueActive=0;
                            end;
                            if(endPosition==-1)
                                endPosition=0;
                            end;
                        end;
                    end;

            end;


            int function returnProcess()
                    return queue[0];
            end;

            int function isQueueEmpty()
                    return queueActive;
            end;

end;

//need a queue for every memory space
Queue homeNodeQueue[20];
Queue processCheck[20];
Queue homeNodeConflicts[20];
Queue writeQueue[20];
Queue finishedProcess[20];
int queueLock[20];
int mostRecent[20];
int homeNodeRequests[20];
for(i=0;i<20;i++)
            homeNodeQueue[i]=Queue();
            processCheck[i]=Queue();
            queueLock[i]=0;
            invalidateRestart[i]=0;
            mostRecent[i]=-1;
            writeQueue[i]=Queue();
            homeNodeRequests[i]=0;
            homeNode[i]=Queue();
            finishedProcess[i]=Queue();
            homeNodeConflicts[i]=Queue();
end;



function checkCache(int cpuA,int cpuB,int address,int flag,int invalidate);



for(i=0;i<NCPU;i++)
            linkLockUp[i]=0;
            linkLockDown[i]=0;
            ackToken[i]=0;
            DRAMLock[i]=0;
end;

//creates a bi direction link
class p2pLink(int Ax,int Ay,int l, int cpuA, int cpuB,int orientation,int linkNo)

            int currentPacketUpNo=0;
            int currentPacketDownNo=0;
```

```
int waitingUp=0;
int waitingDown=0;
int packageW=120;
int packageH=40;
int arrowH=15;
int arrowW=30;
int xGap=200;
int yGap=105;
int boxOffsetX=20;
int boxOffsetY=15;
for(int i=0;i<5;i++)
	lock[i]=0;
end;
packetFont = Font("Times New Roman", 30, 0);
//create a vertical or horizontal link
if(orientation==0)
	httUp = Polygon(0, AAFILL | ABSOLUTE, blackpen, gray192brush,  Ax,Ay-5, 0,0, l-arrowW,0,  l-arrowW,-arrowH,  l,arrowH/2,
	l-arrowW,arrowH*2, l-arrowW,arrowH, 0,arrowH );
	httDown = Polygon(0, AAFILL | ABSOLUTE, blackpen, gray192brush,  Ax,Ay+yGap-5, 0,arrowH/2, arrowW,-arrowH, arrowW,0, l,0, l,arrowH,
	arrowW,arrowH, arrowW,arrowH*2);
	for(i=0;i<5;i++)
		pUpB[i]=Ellipse(0, AAFILL, redpen, gray192brush, Ax, Ay+10,0,0, packageW+(boxOffsetX*2), packageH+(boxOffsetY*2));
		pUp[i] = Rectangle2(0, AAFILL, redpen, gray192brush, Ax+15,Ay,packageW,packageH,blackbrush,packetFont,0);
		pDownB[i]=Ellipse(0, AAFILL, redpen, gray192brush, Ax, Ay+50,0,0, packageW+(boxOffsetX*2), packageH+(boxOffsetY*2));
		pDown[i] = Rectangle2(0, AAFILL, redpen, gray192brush, Ax,Ay+50,packageW,packageH,blackbrush,packetFont,0);
		pUpB[i].setOpacity(0);
		pDownB[i].setOpacity(0);
		pUp[i].setOpacity(0);
		pDown[i].setOpacity(0);
	end;
else
	httUp = Polygon(0, AAFILL | ABSOLUTE, blackpen, gray192brush,  Ax+55,Ay, 0,0, arrowH,0, arrowH,l-arrowW, arrowH*2,l-arrowW,
	arrowH/2,l, -arrowH,l-arrowW, 0,l-arrowW);
	httDown = Polygon(0, AAFILL | ABSOLUTE, blackpen, gray192brush,  Ax+52+xGap,Ay, arrowH/2,0, arrowH*2,arrowW, arrowH,arrowW,
	arrowH,l, 0,l, 0,arrowW, -arrowH,arrowW );
	for(i=0;i<5;i++)
		pUpB[i]=Ellipse(0, AAFILL, redpen, gray192brush, Ax, Ay,0,0, packageW+(boxOffsetX*2), packageH+(boxOffsetY*2));
		pUp[i] = Rectangle2(0,AAFILL, redpen, gray192brush, Ax,Ay, packageW,packageH,blackbrush,packetFont,0);
		pDownB[i]=Ellipse(0, AAFILL,redpen, gray192brush, Ax+80, Ay,0,0 ,packageW+(boxOffsetX*2), packageH+(boxOffsetY*2));
		pDown[i] = Rectangle2(0, AAFILL, redpen, gray192brush, Ax+80,Ay, packageW,packageH,blackbrush,packetFont,0);
		pUpB[i].setOpacity(0);
		pDownB[i].setOpacity(0);
		pUp[i].setOpacity(0);
		pDown[i].setOpacity(0);
	end;
end;

//send packets up link
function packetUp(int ticks,int wflag,Brush brush1,Brush brush2,int address,string message)
	pUpB[currentPacketUpNo].setOpacity(255);
	pUpB[currentPacketUpNo].setBrush(brush1);
	pUp[currentPacketUpNo].setOpacity(255);
	pUp[currentPacketUpNo].setBrush(brush2);
	tempMessage=format("%d",address);
	tempMessage=message+tempMessage;

	pUp[currentPacketUpNo].setTxt(tempMessage);
	if(orientation==0)
		pUpB[currentPacketUpNo].setPos( Ax-packageW-2*boxOffsetX,Ay-packageH/2-boxOffsetY);
		pUp[currentPacketUpNo].setPos( Ax-packageW-boxOffsetX,Ay-packageH/2);
		fork(pUpB[currentPacketUpNo].setPos(Ax+l,Ay-packageH/2-boxOffsetY, ticks, 1, wflag));
		pUp[currentPacketUpNo].setPos(Ax+l+boxOffsetX, Ay-packageH/2, ticks, 1, wflag);
	else
		pUpB[currentPacketUpNo].setPos( Ax-boxOffsetX,Ay-packageH-boxOffsetY);
		pUp[currentPacketUpNo].setPos( Ax,Ay-packageH);
		fork(pUpB[currentPacketUpNo].setPos(Ax-boxOffsetX, Ay+l, ticks, 1, wflag));
		pUp[currentPacketUpNo].setPos(Ax, Ay+l+boxOffsetY, ticks, 1, wflag);
	end;

end;
//send packets down link
function packetDown(int ticks,int wflag,Brush brush1,Brush brush2,int address,string message)
	pDownB[currentPacketDownNo].setOpacity(255);
	pDownB[currentPacketDownNo].setBrush(brush1);
	pDown[currentPacketDownNo].setOpacity(255);
	pDown[currentPacketDownNo].setBrush(brush2);
	tempMessage=format("%d",address);
	tempMessage=message+tempMessage;
	pDown[currentPacketDownNo].setTxt(tempMessage);
	if(orientation==0)
		pDownB[currentPacketDownNo].setPos( Ax+l+boxOffsetX,Ay-packageH/2+yGap-boxOffsetY);
		pDown[currentPacketDownNo].setPos( Ax+l+2*boxOffsetX,Ay-packageH/2+yGap);
		fork(pDownB[currentPacketDownNo].setPos(Ax-packageW-boxOffsetX*2, Ay-packageH/2+yGap-boxOffsetY, ticks, 1, wflag));
		pDown[currentPacketDownNo].setPos(Ax-packageW-boxOffsetX, Ay-packageH/2+yGap, ticks, 1, wflag);
	else
		pDownB[currentPacketDownNo].setPos( Ax+xGap-boxOffsetX,Ay+l);
		pDown[currentPacketDownNo].setPos( Ax+xGap,Ay+l+boxOffsetY);
```

```
                                fork(pDownB[currentPacketDownNo].setPos(Ax+xGap-boxOffsetX, Ay-packageH-boxOffsetY*2, ticks, 1, wflag));
                                pDown[currentPacketDownNo].setPos(Ax+xGap, Ay-packageH-boxOffsetY, ticks, 1, wflag);

                        end;
                end;


end;


link[0]=p2pLink(471+offset*2,285+offset,709     ,0,1  ,0 ,0);
link[1]=p2pLink(90+offset*2,828+offset,370     ,0,2  ,1 ,1);
link[2]=p2pLink(1250+offset*2,828+offset,370    ,1,3  ,1 ,2);
link[3]=p2pLink(471+offset*2,1450+offset,709    ,2,3  ,0 ,3);
link[4]=p2pLink(1619+offset*2,1450+offset,709   ,3,4  ,0 ,4);

//send a packet up a link and ensure two packets dont travel up at once
function linkPathUp(int linkNo,int wflag,Brush brush1,Brush brush2,int address,string message)

        //only send a message when the link has possion of the lock
        while(linkLockUp[linkNo]==1)
                wait(1);
        end;

                linkLockUp[linkNo]=1;
                link[linkNo].packetUp(TICKS*2,wflag,brush1,brush2,address,message);
                wait(TICKS*2);
                linkLockUp[linkNo]=0;

end;

//send a packet down a link and ensure two packets dont travel up at once
function linkPathDown(int linkNo,int wflag,Brush brush1,Brush brush2,int address,string message)

        while(linkLockDown[linkNo]==1)
                wait(1);
        end;
                linkLockDown[linkNo]=1;
                link[linkNo].packetDown(TICKS*2,wflag,brush1,brush2,address,message);
                wait(TICKS*2);
                linkLockDown[linkNo]=0;
end;

//send messages along predefined paths
function linkPath(int cpuA,int cpuB,int wflag,Brush brush1,Brush brush2,int address,string message)


        if(cpuA!=cpuB)
                if(cpuA==0)

                        if(cpuB==1 || cpuB==3 || cpuB==4)
                                linkPathUp(0,wflag,brush1,brush2,address,message);
                                if(cpuB==3 || cpuB==4)
                                        linkPathUp(2,wflag,brush1,brush2,address,message);
                                end;
                                if(cpuB==4)
                                        linkPathUp(4,wflag,brush1,brush2,address,message);
                                end;
                        end;

                        if(cpuB==2)
                                linkPathUp(1,wflag,brush1,brush2,address,message);
                        end;

                end;

                if(cpuA==1)

                        if(cpuB==3 || cpuB==4 )
                                linkPathUp(2,wflag,brush1,brush2,address,message);
                                if(cpuB==4)
                                        linkPathUp(4,wflag,brush1,brush2,address,message);
                                end;

                        end;

                        if(cpuB==0 || cpuB==2)

                                linkPathDown(0,wflag,brush1,brush2,address,message);
                                if(cpuB==2)
                                        linkPathUp(1,wflag,brush1,brush2,address,message);
                                end;
                        end;

                end;
```

```
                        if(cpuA==2)

                                if(cpuB==3 || cpuB==4)

                                        linkPathUp(3,wflag,brush1,brush2,address,message);
                                        if(cpuB==4)
                                                linkPathUp(4,wflag,brush1,brush2,address,message);
                                        end;
                                end;

                                if(cpuB==0 || cpuB==1)

                                        linkPathDown(1,wflag,brush1,brush2,address,message);
                                        if(cpuB==1)
                                                linkPathUp(0,wflag,brush1,brush2,address,message);
                                        end;
                                end;

                        end;

                        if(cpuA==3)

                                if(cpuB==1 || cpuB==0)

                                        linkPathDown(2,wflag,brush1,brush2,address,message);
                                        if(cpuB==0)
                                                linkPathDown(0,wflag,brush1,brush2,address,message);
                                        end;

                                end;
                                if(cpuB==4)
                                        linkPathUp(4,wflag,brush1,brush2,address,message);
                                end;
                                if(cpuB==2)
                                        linkPathDown(3,wflag,brush1,brush2,address,message);
                                end;

                        end;

                        if(cpuA==4)

                                linkPathDown(4,wflag,brush1,brush2,address,message);
                                if(cpuB==1 || cpuB==0 )

                                        linkPathDown(2,wflag,brush1,brush2,address,message);
                                        if(cpuB==0)
                                                linkPathDown(0,wflag,brush1,brush2,address,message);
                                        end;

                                end;
                                if(cpuB==2)
                                        linkPathDown(3,wflag,brush1,brush2,address,message);
                                end;
                        end;
                else
                end;

end;


class CPU;
CPU cpu[5];

//create a node containg a cpu cache and memory
class CPU(int x,int y,int cpuNo)
                w=400;
                gap=15;
                h=700;
                h2=h/17;
                w2=w-(2*gap);
                int cacheLineFound=-1;
                int repliesReturned=0;
                int memoryAccessComplete=0;
                int cancelMemoryAccess=0;
                cpuLock=0;
                class Memory;
                class Cache;
                class DataLines;
                class dialogBox;
                tempQueue=Queue();
                Memory cpuMemory;
                Cache cpuCache;
                DataLines cpuDataLines;
                DataLines cpuDataLines2;
                dialogBox cpudialogBox;
                int READ=0;
                int WRITE=1;
```

```
int ReadOrWrite=READ;
brushTemp= SolidBrush(rgb(255, 255, 150));
r = Rectangle2(0, 0, blackpen, brushTemp, x-20, y, w+40, h+80);
int invalidating=-1;
int cpuRestart=0;
int mesifUpdateMemory=0;
int currentProcess=-1;
Queue writeConflictQueue[2];
int cQLock[2];
cQLock[0]=0;
cQLock[1]=0;
writeConflictQueue[0]=Queue();
writeConflictQueue[1]=Queue();
conflictQueue[0]=Queue();
conflictQueue[1]=Queue();
int readCycle=-1;
int probeCycle=-1;
int readConflict=-1;
int probeConflict=-1;
int waitingOnHomeNodeAck[20];
int mesifWrite=0;
int conflict=0;
int isCpuWriting=0;
int lineModified=0;
string conflicts="";
replyLock[20];
for(i=0;i<20;i++)
        replyLock[i]=0;
        waitingOnHomeNodeAck[i]=0;
end;

titleFont = Font("Times New Roman", 40, 0);
t = Txt(0, HLEFT | VTOP, x-140, y-80, redbrush, titleFont, "CPU %d", cpuNo);

class Memory()
                int mem[5];
                int stale[5];
                r = Rectangle2(0, 0, blackpen, gray192brush, x+gap, y+gap, w2, gap+h2*4.75);
                //Create 4 instances
                for (int i = 0; i < 4; i++)
                        //Stores the data at the 4 memory location
                        mem[i] = 0;
                        //create 4 rectangles to represent the memory locations
                        memR[i] = Rectangle2(0, 0, blackpen,  brushsH[cpuNo], x+2*gap, y+2*gap+(h2+gap/2)*i, w2-gap*2, h2, blackbrush,
                        0);
                        //create the text to be displayed in the memory rectangles
                        memRTxt[i]=Txt(0,HLEFT|VTOP, x+2*gap, y+2*gap+(h2+gap/2)*i, blackbrush, 0,"Address:a10 Data:10");
                        //set text sixe
                        memRTxt[i].setSize(w2*0.9,h2);
                        //name the memory locations, name every 5th location depending on the cpuNo
                        memRTxt[i].setTxt("Address:a%d Data:%d", cpuNo+(i*5), mem[i]);
                end;


                function highlight(int address, int flag)
                        memR[address].setBrush((flag) ? brushsH[cpuNo] : brushsS[cpuNo]);
                end;

                function reset()

                        for (int i = 0; i < 4; i++)
                                mem[i] = 0;
                                stale[i] = 0;
                        end;

                end;

end;


class Cache()
        int state[2];
        int content[2];
        int address[2];

        r = Rectangle2(0, 0, blackpen, gray192brush, x+gap, y+h*0.421, w2, (h2+gap)*2);
        //create 2 cache lines
        for (int i = 0; i < 2; i++)
                //stores the state the line is held in
                state[i]=INVALID;
                //contents of the line
                contents[i]=0;
                //address the line holds
                address[i]=-1;

                //creates the state box of the line
                stateR[i] = Rectangle2(0, 0, blackpen, whitebrush, x+gap*2, y+h*0.421+gap+i*h2, w2*0.1, h2, blackbrush, 0);
```

```
                stateRTxt[i]=Txt(0,HLEFT|VTOP,  x+gap*2, y+h*0.421+gap+i*h2, blackbrush, 0,"M");
                stateRTxt[i].setSize(w2*0.1,h2);
                stateRTxt[i].setTxt("I");
                //creates the address box
                aR[i] = Rectangle2(0, 0, blackpen, whitebrush, x+gap*2+w2*0.1,y+h*0.421+gap+i*h2, w2*0.41, h2);
                aRTxt[i]=Txt(0,HLEFT|VTOP,  x+gap*2+w2*0.20,y+h*0.421+gap+i*h2, blackbrush, 0,"A10");
                aRTxt[i].setSize(w2*0.2,h2);
                aRTxt[i].setTxt("");
                //creates the data box
          dR[i] = Rectangle2(0, 0, blackpen, whitebrush,  x+gap*2+w2*0.51,y+h*0.421+gap+i*h2, w2*0.41,h2);
                dRTxt[i]=Txt(0,HLEFT|VTOP,  x+gap*2+w2*0.60,y+h*0.421+gap+i*h2, blackbrush, 0,"80");
                dRTxt[i].setSize(w2*0.2,h2);
                dRTxt[i].setTxt("");
        end;
        //highlight a cache line
        function highlight(int cacheLine, int flag)
                        stateR[cacheLine].setBrush((flag) ? whitebrush : greenbrush);
                        aR[cacheLine].setBrush((flag) ? whitebrush : greenbrush);
                        dR[cacheLine].setBrush((flag) ? whitebrush : greenbrush);
        end;
        //reset cache
        function reset()

                for (int i = 0; i < 2; i++)
                        state[i]=INVALID;
                        contents[i]=0;
                        address[i]=-1;
                end;
        end;
end;
class DataLines(real xpos,real ypos)
        bodyWidth=5;
        aWidth=12;
        l=60;

        arrow1 = Polygon(0, AAFILL | ABSOLUTE, 0, blackbrush, xpos,ypos, 0,0, -aWidth,-aWidth,
        -bodyWidth,-aWidth, -bodyWidth,-l, -aWidth,-l, 0,-l-aWidth, aWidth,-l, bodyWidth,-l, bodyWidth,-aWidth, aWidth,-aWidth );
        arrow2 = Polygon(0, AAFILL | ABSOLUTE, 0, blackbrush, xpos+h2*4,ypos, 0,0, -aWidth,-aWidth,
        -bodyWidth,-aWidth, -bodyWidth,-l, -aWidth,-l, 0,-l-aWidth, aWidth,-l, bodyWidth,-l, bodyWidth,-aWidth, aWidth,-aWidth );

        body1=Rectangle2(0, 0, 0, bluebrush, xpos-bodyWidth, ypos, 2*bodyWidth, -l);
        body2=Rectangle2(0, 0, 0, redbrush,  xpos-bodyWidth+h2*4,ypos-aWidth, 2*bodyWidth, -l);
        up= Polygon(0, AAFILL | ABSOLUTE, 0, bluebrush, xpos-aWidth,ypos, 0,0, aWidth,-10, 2*aWidth,0);
        down= Polygon(0, AAFILL | ABSOLUTE, 0, redbrush,  xpos+h2*4,ypos-l, 0,0, -aWidth,-10, aWidth,-10);

        up.setOpacity(0);
        down.setOpacity(0);
        body1.setOpacity(0);
        body2.setOpacity(0);

        function moveUp(int ticks, int wflag)

                        // initial positions, size & opacity
                        up.setPos(xpos-aWidth,ypos);
                        body1.setPos(xpos-bodyWidth, ypos);
                        body1.setSize(2*bodyWidth, 0);
                        up.setOpacity(255);
                        body1.setOpacity(255);

                        // final positions, size & opacity
                        up.setPos( xpos-aWidth,ypos-l, ticks, 1, 0);
                        body1.setSize(2*bodyWidth, -l, ticks, 1, 0);
                        arrow1.setOpacity(0, ticks, 1, wflag);

        end;

//
// animates down movement on vertical bus arrow
//
// may be called with ticks = 0
//

        function moveDown(int ticks, int wflag)
                //
                // initial positions, size & opacity
                //
                down.setPos( xpos+h2*4,ypos-l);
                body2.setPos (xpos-bodyWidth+h2*4,ypos-l-10);
                body2.setSize(2*bodyWidth, 0);
                down.setOpacity(255);
                body2.setOpacity(255);

                //
                // final positions, size & opacity
                //

                body2.setSize( 2*bodyWidth, l, ticks, 1, 0);
```

```
		down.setPos( xpos+h2*4,ypos, ticks, 1, 0);
		arrow2.setOpacity(0, ticks, 1, wflag);

	end;

	function reset()


		up.setOpacity(0);
		body1.setOpacity(0);
		down.setOpacity(0);
		body2.setOpacity(0);
		arrow1.setOpacity(255);
		arrow2.setOpacity(255);


	end;


end;


	//paths for snooping must be called link this to insure that the cache is only checked on recieving a broadcast message
	function cpu0path(int cpuNo,int address,int invalidate,int flag,string message)
		linkPath(0,1,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
		fork(checkCache(cpuNo,1, address,flag,invalidate));
		linkPath(1,3,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
		fork(checkCache(cpuNo,3, address,flag,invalidate));
		linkPath(3,4,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
		fork(checkCache(cpuNo,4, address,flag,invalidate));
	end;
	function cpu1path(int cpuNo,int address,int invalidate,int flag,string message)
		linkPath(1,3,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
		fork(checkCache(cpuNo,3, address,flag,invalidate));
		linkPath(3,4,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
		fork(checkCache(cpuNo,4, address,flag,invalidate));
	end;
	function cpu2path(int cpuNo,int address,int invalidate,int flag,string message)
		linkPath(2,0,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
		fork(checkCache(cpuNo,0, address,flag,invalidate));
		linkPath(0,1,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
		fork(checkCache(cpuNo,1, address,flag,invalidate));
	end;
	function cpu3path1(int cpuNo,int address,int invalidate,int flag,string message)
		linkPath(3,1,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
		fork(checkCache(cpuNo,1, address,flag,invalidate));
		linkPath(1,0,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
		fork(checkCache(cpuNo,0, address,flag,invalidate));
	end;
	function cpu3path2(int cpuNo,int address,int invalidate,int flag,string message)
		linkPath(3,2,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
		fork(checkCache(cpuNo,2, address,flag,invalidate));
	end;
	function cpu3path3(int cpuNo,int address,int invalidate,int flag,string message)
		linkPath(3,4,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
		fork(checkCache(cpuNo,4, address,flag,invalidate));
	end;

	//sneds broadcast probe around system
	function broadcast(int cpuNo,int address,int invalidate,int flag,string message)

		if(cpuNo==0)
				fork(checkCache(cpuNo,0, address,flag,invalidate));
				fork(cpu0path(cpuNo,address,invalidate,flag,message));
				linkPath(0,2,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
				fork(checkCache(cpuNo,2, address,flag,invalidate));
		end;

		if(cpuNo==1)
				fork(checkCache(cpuNo,1, address,flag,invalidate));
				fork(cpu1path(cpuNo,address,invalidate,flag,message));
				linkPath(1,0,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
				fork(checkCache(cpuNo,0, address,flag,invalidate));
				linkPath(0,2,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
				fork(checkCache(cpuNo,2, address,flag,invalidate));
		end;

		if(cpuNo==2)
				fork(checkCache(cpuNo,2, address,flag,invalidate));
				fork(cpu2path(cpuNo,address,invalidate,flag,message));
				linkPath(2,3,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
				fork(checkCache(cpuNo,3, address,flag,invalidate));
				linkPath(3,4,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
				fork(checkCache(cpuNo,4, address,flag,invalidate));
		end;

		if(cpuNo==3)
```

```
                                        fork(checkCache(cpuNo,3, address,flag,invalidate));
                                        fork(cpu3path1(cpuNo,address,invalidate,flag,message));
                                        fork(cpu3path2(cpuNo,address,invalidate,flag,message));
                                        fork(cpu3path3(cpuNo,address,invalidate,flag,message));

                end;

                if(cpuNo==4)
                                        fork(checkCache(cpuNo,4, address,flag,invalidate));
                                        linkPath(4,3,0,brushsH[cpuNo],brushsH[cpuNo],address,message);
                                        fork(checkCache(cpuNo,3, address,flag,invalidate));
                                        fork(cpu3path1(cpuNo,address,invalidate,flag,message));
                                        fork(cpu3path2(cpuNo,address,invalidate,flag,message));
                end;


        end;


class dialogBox()

        int x1=x-20;
        int x2=x1+440;
        int y1=y+570;
        int w=420;
        int h=150;
        int h2=40;
        int boxLock=0;
        int t1x=50;
        int t1y=y1-135;
        int t2x=40;
        int t2y=y1-40;
        f1 = Font("Times New Roman", 40, 0);
        f2 = Font("Times New Roman", 40, 0);

        processTxtBox1=Rectangle2(0,0, blackpen, vellumbrush, t1x+x1,t1y, w-70, h-80,blackbrush, f1, "");
        processTxtBox1.moveToBack();
        processTxtBox2=Rectangle2(0,0, blackpen, vellumbrush, t2x+x1,t2y, w-40, h+65,blackbrush, f2, "");
        processTxtBox2.moveToBack();
        processContainer= Polygon(0, 0, blackpen, gray192brush, x1,y1, 0,0, 20,0, 0,-h, w,0, 0,h*2+h2, -w,0, 0,-h, -20,0);
        processContainer.moveToBack();

        int ticksLocal=TICKS;
        int boxOut=0;

        //animate box out
        function showPBox(int ticks,int RW,int address)

                if(RW==READ)
                        processTxtBox1.setTxt("Read to a%d",address);
                else
                        processTxtBox1.setTxt("Write to a%d",address);
                end;
                processContainer.setPos( x2,y1, ticks, 1, 0);
                processTxtBox1.setPos( x2+t1x,t1y, ticks, 1, 0);
                processTxtBox2.setPos( x2+t2x,t2y, ticks, 1, 1);
                boxOut=1;
        end;
        //animate box in
        function hidePBox(int ticks)
                processContainer.setPos( x1,y1, ticks, 1, 0);
                processTxtBox1.setPos( x1+t1x,t1y, ticks, 1, 0);
                processTxtBox2.setPos( x1+t2x,t2y, ticks, 1, 1);
                boxOut=0;
        end;

        when processContainer.eventLB(int flags, real x, real y)
                if(boxOut==1 && flags & EB_LEFT==0)
                        boxOut=0;
                        start(1);
                        processContainer.setPos( x1,y1, ticksLocal, 1, 0);
                        processTxtBox1.setPos( x1+t1x,t1y, ticksLocal, 1, 0);
                        processTxtBox2.setPos( x1+t2x,t2y, ticksLocal, 1, 1);
                        processContainer.setBrush( gray192brush);
                        processTxtBox1.setBrush(vellumbrush);
                        processTxtBox2.setBrush(vellumbrush);
                end;
        end;

        when processContainer.eventEE(int flags, real x, real y)
                if(boxOut==1)
                        processContainer.setBrush(flags & EB_ENTER ?  gray128brush :  gray192brush);
                        processTxtBox1.setBrush(flags & EB_ENTER ?  gray192brush :  vellumbrush);
                        processTxtBox2.setBrush(flags & EB_ENTER ?  gray192brush :  vellumbrush);
                end;
        end;
```

```
		function setCurrentTask(string message)
			processTxtBox2.setTxt(message);
		end;

		function setTitle(string message)
			processTxtBox1.setTxt(message);
		end;



end;

//these two function need to be forked thats why there not included in the main program.
function accessMemory(int cacheContents,int address)
					while(DRAMLock[address%NCPU]==1)
						wait(1);
					end;
					DRAMLock[address%NCPU]=1;
					cpu[address%NCPU].cpuDataLines.moveUp(TICKS*2, 1);
					cpu[address%NCPU].cpuMemory.highlight(address/4,0);
					cpu[address%NCPU].cpuMemory.mem[address/4]=cacheContents;
					cpu[address%NCPU].cpuMemory.memRTxt[address/4].setTxt("Address:a%d Data:%d",address, cacheContents);
					DRAMLock[address%NCPU]=0;
end;

function ackRelease(int cpuA,int cpuB,int address,string mess)
		linkPath(cpuA,cpuB,0,brushsH[cpuA],brushsH[cpuA],address,mess);
		cpu[cpuB].waitingOnHomeNodeAck[address]=0;
end;


///mesif logic
function MESIF(int address,int doWrite)



		//If the Cpu is not currently doing an operation
		if(cpuLock == 0 )
			cpuLock=1;
			//start animating, lock the the cpu and move the data lines up
			start(1);

			conflicts="";
			message=format("Check Local Cache\nfor a Valid copy\nof a%d",address);
			cpudialogBox.setCurrentTask(message);
			if(doWrite==WRITE)
				cpudialogBox.showPBox(20,WRITE,address);
			else
				cpudialogBox.showPBox(20,READ,address);
			end;

			//put message in dialog box
			cpuDataLines2.moveUp(TICKS, 1);
			message=format("Waiting\nfor Acknowlegement\n to Continue a%d",address);
			cpudialogBox.setCurrentTask(message);
			while(waitingOnHomeNodeAck[address]==1)
				wait(1);
			end;

			//this tells other cpus that broadcast this cpu that its currently writing
			//so it will be met with a conflict invalid reply
			if(doWrite==WRITE)
				isCpuWriting=1;
			else
				isCpuWriting=0;
			end;

			//If set to 1 the cpu is waiting on an acknowledement from the cpu
			//to continue, so this cpu will not return a reply until its at 0
			//waitingOnHomeNodeAck[address]=0;
			//this variable tells that thats a line returned from a broadcast is modified or not
			lineModified=0;
			//The cpu can be in a probe cycle or read cycle when its searching for its cache line.
			//Depending on the cycle its in, it will return different replies
			//If its in the probe or read cycle set the varibles to the addressess the cpu is
			//currently looking for
			probeCycle=-1;
			readCycle=-1;
			//If a cpu is trying to get the same cache line as when its broadcasted it will return a conflict
			//meesage, there are two different times it will send a conflict message,
			//when its broadcasting all the caches, or its sending a read message to the Home Node and awaiting the
			//returned cache line.
			//read conflict stores the cpu no that is currently reading
			//The probe conflict simply tells whether any other cpus where in their broadcast cycle
			readConflict=0;
			probeConflict=0;
			//This variable stores the cpu no. of the cpu which returned a valid copy of the cache line to
```

```
//this cpu on a broadcast
cacheLineFound=-1;
//This variable stops the sequence from continuing until a memory access has been finished
memoryAccessComplete=0;
//This is used to stop a memory access when a cancel message is sent to the home node
cancelMemoryAccess=0;
//This stores all the replies recieved after a broadcast from the other cpus
repliesReturned=0;

//If theres a different cache line stored in the Modified State flush it to the Home Node
if(cpuCache.address[address%2]!=address && cpuCache.state[address%2]==MODIFIED)
		waitingOnHomeNodeAck[cpuCache.address[address%2]]=1;
		message=format("Flush a%d to\nHome Node %d",cpuCache.address[address%2],cpuCache.address[address
		%2]%NCPU);
		cpudialogBox.setCurrentTask(message);
		mostRecent[cpuCache.address[address%2]]=-1;
		linkPath(cpuNo,cpuCache.address[address%2]%NCPU,0,brushsH[cpuNo],brushsH[cpuNo],address,"FL:a");

		accessMemory(cpuCache.contents[address%2],cpuCache.address[address%2]);
		message=format("Home Node a%d\nresponds with a\nFlush\nAcknowledge",cpuCache.address[address%2]%NCPU);
		cpudialogBox.setCurrentTask(message);
		linkPath(cpuCache.address[address%2]%NCPU,cpuNo,0,brushsH[cpuNo],brushsH[cpuNo],address,"FA:a");
		cpuCache.state[address%2]=INVALID;
		cpuCache.stateRTxt[address%2].setTxt("I");
		waitingOnHomeNodeAck[cpuCache.address[address%2]]=0;
end;

//If the cache Line is not on the cache or is Invalid ie not available locally start the MESIF protocol
if(cpuCache.address[address%2]!=address || cpuCache.state[address%2]==INVALID ||
( cpuCache.address[address%2]==address && doWrite==WRITE && (cpuCache.state[address%2]==SHARED ||
cpuCache.state[address%2]==FORWARD)))

		//Set probe cycle to the address thats being searched for, it another cpu is also
		//probing it will know theres a conflict
		probeCycle=address;
		if(doWrite==WRITE )
			message=format("Send Probe Invalidate\nto all Nodes,\nAcks Returned 0/4", address%NCPU);
			cpudialogBox.setCurrentTask(message);
			//broadcast all the cpus
			broadcast(cpuNo,address,1,0,"PRI:a");
		else
			message=format("Send Probe\nto all Nodes,\nAcks Returned 0/4", address%NCPU);
			cpudialogBox.setCurrentTask(message);
			//broadcast all the cpus
			broadcast(cpuNo,address,0,0,"PR:a");
		end;


		//Wait for all the replies before continuing
		while(repliesReturned!=5)
			wait(1);
		end;
		//exit the probe cycle
		probeCycle=-1;

		//Enter read cycle
		readCycle=address;
		//If no valid cache Lines where found on the other cpus send a read message and
		//take the data from the home Node memory
		if(cacheLineFound==-1)
			message=format("No dirty lines,\nsend READ message\nto Home Node %d\nCpu
			Conflicts:"+conflicts,address%NCPU);
			cpudialogBox.setCurrentTask(message);
			linkPath(cpuNo,address%NCPU,0,brushsH[cpuNo],brushsH[cpuNo],address,"READ:a");
		else
		//If a cache line was found send a cancel message to the home node to stop accessing memory
			message=format("Dirty line found,\nsend CANCEL\nmessage\nto Home Node %d\nCpu
			Conflicts:"+conflicts,address%NCPU);
			cpudialogBox.setCurrentTask(message);
			linkPath(cpuNo,address%NCPU,0,brushsH[cpuNo],brushsH[cpuNo],address,"CNCL:a");
			cancelMemoryAccess=0;
		end;
		//Add the read or cancel message to the home nodes queue
		homeNodeQueue[address].addProcess(cpuNo);
		//your now waiting on a home node ack
		waitingOnHomeNodeAck[address]=1;


		//this deletes out of date conflicts
		tempQueue.emptyQueue();
		for(int i=0;i<finishedProcess[address].endPosition;i++)
			tempQueue.addProcess(finishedProcess[address].queue[i]);
		end;

		while(tempQueue.isQueueEmpty()==1)
			if(conflictQueue[address%2].checkProcess(tempQueue.returnProcess())==0)
				finishedProcess[address].deleteProcess(tempQueue.returnProcess());
```

```
		end;
		tempQueue.removeProcess();
end;


//The Home node searchs through all the conflicts in the read or cancel message and stores all unknown conflicts
while(conflictQueue[address%2].isQueueEmpty()==1)
		if(homeNodeConflicts[address].checkProcess(conflictQueue[address%2].returnProcess())==0
		&& finishedProcess[address].checkProcess(conflictQueue[address%2].returnProcess())==0)
				homeNodeConflicts[address].addProcess(conflictQueue[address%2].returnProcess());
				if(conflictQueue[address%2].returnProcess()>=10)
						writeQueue[address].addProcess(conflictQueue[address%2].returnProcess());
				end;

		end;
		conflictQueue[address%2].removeProcess();

end;

//check if the home node request is on the Queue
if(doWrite==WRITE)
		if(homeNodeConflicts[address].checkProcess(cpuNo+10)==0
		&& finishedProcess[address].checkProcess(cpuNo+10)==0)
				homeNodeConflicts[address].addProcess(cpuNo+10);
				writeQueue[address].addProcess(cpuNo+10);

		end;
else
		if(homeNodeConflicts[address].checkProcess(cpuNo)==0
		&& finishedProcess[address].checkProcess(cpuNo)==0)
				homeNodeConflicts[address].addProcess(cpuNo);
		end;
end;

//Wait until it this cpus turn on the home node
while(homeNodeQueue[address].returnProcess()!=cpuNo)
		wait(1);
end;
//Wait until the home node has finished its memory access, this will be quicker if a cancel message was sent
while(memoryAccessComplete==0)
		wait(1);
end;



//remove this process from the queues except the homeNodeQueue
if(doWrite==WRITE)
		writeQueue[address].deleteProcess(cpuNo+10);
		homeNodeConflicts[address].deleteProcess(cpuNo+10);
		finishedProcess[address].addProcess(cpuNo+10);
else
		homeNodeConflicts[address].deleteProcess(cpuNo);
		finishedProcess[address].addProcess(cpuNo);
end;
//If no valid cache lines where found
if(cacheLineFound==-1)

		//If there are no type of conflicts send the data from memory
		if(probeConflict==0 && readConflict==0&&mostRecent[address]==-1)

				//If there where no conflicts and the Queue is empty that means all cpus are finished with this
				//cache line
				//so all the queues can be emptied
				mostRecent[address]=cpuNo;
				homeNodeQueue[address].removeProcess();
				//Send the cache line in exclusive
				message=format("Home Node %d sends\na%d from memory\nin Exclusive",address
				%NCPU,address);
				cpudialogBox.setCurrentTask(message);
				linkPath(address%NCPU,cpuNo,0,brushsH[cpuNo],brushsH[address
				%NCPU],address,"DataE:a");

				//Enter the value on the source node
				cpuCache.contents[address%2]=cpu[address%NCPU].cpuMemory.mem[address/4];
				cpuCache.state[address%2]=EXCLUSIVE;
				cpuCache.stateRTxt[address%2].setTxt("E");
				cpuCache.address[address%2]=address;
				cpuCache.aRTxt[address%2].setTxt("a%d",address);
				cpuCache.dRTxt[address%2].setTxt("%d",cpuCache.contents[address%2]);
				cpuCache.highlight(address%2,0);

				if(doWrite==WRITE)
						//add one to the data and store it and set to modified
						cpuCache.contents[address%2]++;
						cpuCache.dRTxt[address%2].setTxt("%d",cpuCache.contents[address%2]);
						cpuCache.state[address%2]=MODIFIED;
						cpuCache.stateRTxt[address%2].setTxt("M");
```

```
                end;
                //the read cycle is over and you recieved your ack
                waitingOnHomeNodeAck[address]=0;
                readCycle=-1;
        else



                //add yourself to the queue, which tells who has the most recent copy
                //if you are the first process in with a conflict you get it in exclusive
                if(mostRecent[address]==-1)
                        mostRecent[address]=cpuNo;

                        homeNodeQueue[address].removeProcess();
                        message=format("Home Node %d sends\na%d from memory\nin
                        Exclusive",address%NCPU,address);
                        cpudialogBox.setCurrentTask(message);
                        linkPath(address%NCPU,cpuNo,0,brushsH[cpuNo],brushsH[address
                        %NCPU],address,"DataE:a");
                        cpuCache.state[address%2]=EXCLUSIVE;
                        cpuCache.stateRTxt[address%2].setTxt("E");
                        cpuCache.contents[address%2]=cpu[address
                        %NCPU].cpuMemory.mem[address/4];
                        cpuCache.address[address%2]=address;
                        cpuCache.aRTxt[address%2].setTxt("a%d",address);
                        cpuCache.dRTxt[address%2].setTxt("%d",cpuCache.contents[address%2]);
                        cpuCache.highlight(address%2,0);

                        //write to the data
                        if(doWrite==WRITE)
                                cpuDataLines2.moveDown(TICKS, 1);
                                cpuDataLines2.moveUp(TICKS, 1);
                                //add one to the data and store it and set to modified
                                cpuCache.contents[address%2]++;
                                cpuCache.dRTxt[address%2].setTxt("%d",cpuCache.contents[address
                                %2]);
                                cpuCache.state[address%2]=MODIFIED;
                                cpuCache.stateRTxt[address%2].setTxt("M");

                        end;
                        readCycle=-1;

                else

                        currentData=mostRecent[address];
                        mostRecent[address]=cpuNo;
                        homeNodeQueue[address].removeProcess();
                        //if there are no more conflicts send an ack to the source node
                        if(homeNodeConflicts[address].isQueueEmpty()==0)
                                fork(ackRelease(address%NCPU,cpuNo,address,"ACK:a"));
                        end;

                        //send a transfer message
                        if(doWrite==WRITE || writeQueue[address].isQueueEmpty()==1)
                                message=format("Home Node %d,\nsends a
                                Transfer\nInvalidate\nmessage to Node%d",address
                                %NCPU,currentData);
                                cpudialogBox.setCurrentTask(message);
                                linkPath(address
                                %NCPU,currentData,0,brushsH[cpuNo],brushsH[address
                                %NCPU],address,"XFERI:a");

                        else
                                message=format("Home Node %d,\nsends a Transfer\nmessage to
                                Node%d",address%NCPU,currentData);
                                cpudialogBox.setCurrentTask(message);
                                linkPath(address
                                %NCPU,currentData,0,brushsH[cpuNo],brushsH[address
                                %NCPU],address,"XFER:a");
                        end;
                        cpu[currentData].waitingOnHomeNodeAck[address]=0;

                        //wait until the data is recieved at the tranfer node
                        while(cpu[currentData].cpuCache.state[address%2]==SHARED ||
                                cpu[currentData].cpuCache.state[address%2]==INVALID ||
                                cpu[currentData].readCycle!=-1)
                                wait(1);
                        end;
                        goModified=0;
                        if(cpu[currentData].cpuCache.state[address%2]==MODIFIED)
                                goModified=1;
                        end;
                        if(doWrite==WRITE || goModified==1 ||writeQueue[address].isQueueEmpty()==1
                        )
                                cpu[currentData].cpuCache.state[address%2]=INVALID;
```

```
                                        cpu[currentData].cpuCache.stateRTxt[address%2].setTxt("I");
                                        cpu[currentData].cpuCache.stateRTxt[address%2].setTxt("I");
                                else
                                        cpu[currentData].cpuCache.state[address%2]=SHARED;
                                        cpu[currentData].cpuCache.stateRTxt[address%2].setTxt("S");
                                        cpu[currentData].cpuCache.stateRTxt[address%2].setTxt("S");
                                end;
                                //transfer the data
                                if(goModified==1)
                                        message=format("Node %d\nsends a%d\nin
                                        MODIFIED",currentData,address);
                                        cpudialogBox.setCurrentTask(message);
                        linkPath( currentData, cpuNo,0,brushsH[cpuNo],brushsH[currentData],address,"DataM:a");
                                        cpuCache.contents[address%2]= cpu [currentData].cpuCache. C
                                        ontents[ address%2];
                                        cpuCache.state[address%2]=MODIFIED;
                                        cpuCache.stateRTxt[address%2].setTxt("M");
                                else
                                        if(writeQueue[address].isQueueEmpty()==1||doWrite==WRITE)
                                                message=format("Node %d\nsends a%d\nin
                                                EXCLUSIVE" ,currentData,address);
                                                cpudialogBox.setCurrentTask(message);

                        linkPath( currentData,cpuNo,0,brushsH[cpuNo],brushsH[currentData],address,"DataE:a");
                                                cpuCache.contents[address%2]= cpu[currentData].
                                                cpuCache.contents[address%2];
                                                cpuCache.state[address%2]=EXCLUSIVE;
                                                cpuCache.stateRTxt[address%2].setTxt("E");
                                        else
                                                message=format("Node %d\nsends a%d\nin
                                                FORWARD",currentData,address);
                                                cpudialogBox.setCurrentTask(message);
                                                linkPath( currentData,cpuNo, 0, brushsH[ cpuNo],
                                                brushsH [currentData] address, "DataF:a");
                                                cpuCache.contents [address%2]= cpu [currentData]
                                                .cpuCache. contents[address%2];
                                                cpuCache.state[address%2]=FORWARD;
                                                cpuCache.stateRTxt[address%2].setTxt("F");
                                        end;
                                end;

                                cpuCache.address[address%2]=address;
                                cpuCache.aRTxt[address%2].setTxt("a%d",address);
                                cpuCache.dRTxt[address%2].setTxt("%d",cpuCache.contents[address%2]);
                                cpuCache.highlight(address%2,0);
                                if(doWrite==WRITE)
                                        //add one to the data and store it and set to modified
                                        cpuCache.contents[address%2]++;
                                        cpuCache.dRTxt[ address%2].
                                        setTxt("%d",cpuCache.contents[address%2]);
                                        cpuCache.state[address%2]=MODIFIED;
                                        cpuCache.stateRTxt[address%2].setTxt("M");
                                end;

                                readCycle=-1;

                        end;

                end;


        else

                mostRecent[address]=cpuNo;
                homeNodeQueue[address].removeProcess();
                if(doWrite==WRITE)
                        message=format( "Home Node %d\nverifies\nWrite Request\nfor
                        a%d",address%NCPU,address);
                        cpudialogBox.setCurrentTask(message);
                else
                        message=format("Home Node %d\nverifies\nRead Request\nfor
                        a%d",address%NCPU,address);
                        cpudialogBox.setCurrentTask(message);
                end;

                if(lineModified==1)

linkPath(address%NCPU,cpuNo,0,brushsH[cpuNo],brushsH[address%NCPU],address,"DataM:a");
                        cpuCache.contents[address%2]=cpu[cacheLineFound].cpuCache.contents[address%2];
                        cpuCache.state[address%2]=MODIFIED;
                        cpuCache.stateRTxt[address%2].setTxt("M");
                else
                        if(doWrite==WRITE)

linkPath(address%NCPU,cpuNo,0,brushsH[cpuNo],brushsH[address%NCPU],address,"DataE:a");

cpuCache.contents[address%2]=cpu[address%NCPU].cpuMemory.mem[address/4];
```

```
                                                cpuCache.state[address%2]=EXCLUSIVE;
                                                cpuCache.stateRTxt[address%2].setTxt("E");
                                        else

linkPath(address%NCPU,cpuNo,0,brushsH[cpuNo],brushsH[address%NCPU],address,"DataF:a");

cpuCache.contents[address%2]=cpu[address%NCPU].cpuMemory.mem[address/4];
                                                cpuCache.state[address%2]=FORWARD;
                                                cpuCache.stateRTxt[address%2].setTxt("F");
                                        end;
                                end;
                                cpuCache.address[address%2]=address;
                                cpuCache.aRTxt[address%2].setTxt("a%d",address);
                                cpuCache.dRTxt[address%2].setTxt("%d",cpuCache.contents[address%2]);
                                cpuCache.highlight(address%2,0);
                                message=format("Source Node %d\nverifies the changes\non Node %d",cacheLineFound);
                                cpudialogBox.setCurrentTask(message);
                                fork(ackRelease(cpuNo,cacheLineFound,address,"ACK:a"));
                                if(doWrite==WRITE)

                                        cpuDataLines2.moveDown(TICKS, 1);
                                        cpuDataLines2.moveUp(TICKS, 1);
                                        cpuCache.contents[address%2]++;
                                        cpuCache.dRTxt[address%2].setTxt("%d",cpuCache.contents[address%2]);
                                        cpuCache.state[address%2]=MODIFIED;
                                        cpuCache.stateRTxt[address%2].setTxt("M");

                                end;

                                waitingOnHomeNodeAck[address]=0;
                                readCycle=-1;

                        end;

                else
                        waitingOnHomeNodeAck[address]=1;
                        cpuCache.highlight(address%2,0);
                        cpuDataLines2.moveDown(TICKS, 1);
                        if(doWrite==WRITE)
                                if(cpuCache.state[address%2]==MODIFIED||cpuCache.state[address%2]==EXCLUSIVE)
                                        cpuDataLines2.moveUp(TICKS, 1);
                                        cpuCache.contents[address%2]++;
                                        cpuCache.dRTxt[address%2].setTxt("%d",cpuCache.contents[address%2]);
                                        cpuCache.state[address%2]=MODIFIED;
                                        cpuCache.stateRTxt[address%2].setTxt("M");

                                end;
                        end;
                        waitingOnHomeNodeAck[address]=0;
        end;
                while(waitingOnHomeNodeAck[address]==1|| readCycle!=-1)
                        wait(1);
                        message=format("Awaiting\nAcknowledge\nOr Transfer\nmessage");
                        cpudialogBox.setCurrentTask(message);
                end;
                readCycle=-1;
                readConflict=-1;
                probeCycle=-1;
                probeConflict=0;
                lineModified=0;
                //once finished send the data lines down, remove the cpuLock and add a checkpoint.

                cpudialogBox.setTitle("Last Action");
                if(doWrite==WRITE)
                        message=format("Write a%d",address);
                        cpudialogBox.setCurrentTask(message);
                else
                        message=format("Read a%d",address);
                        cpudialogBox.setCurrentTask(message);
                end;
                cpuLock = 0;
                checkPoint();


        end;



end;

class rwbuttons(real x, real y,real w,real h,int address)
        background=Rectangle2(0, 0, blackpen, brushsH[address%NCPU], x,y,w,h,blackbrush,0);
        buttonTxt=Txt(0, 0, x+w/2, y+h/2, blackbrush, 0, "A20");
        readButton=Polygon(0, AAFILL | ABSOLUTE, blackpen, brushsH[address%NCPU],  x,y,  0,0,  w,0,   0,h);
        writeButton=Polygon(0, AAFILL | ABSOLUTE, blackpen, brushsH[address%NCPU],  x,y+h,  0,0,  w,0,   w,-h);
        rtxt = Txt(0, HLEFT | VTOP, x, y, blackbrush, 0, "R");
        wtxt = Txt(0, HRIGHT | VBOTTOM, x+(w*1.05), y+(h*1.1), blackbrush, 0, "W");
```

```
		rtxt.setSize(w*0.66,h*0.66);
		wtxt.setSize(w*0.66,h*0.66);
		buttonTxt.setSize(w*0.8,h*0.8);
		buttonTxt.setTxt("A%d",address);
	readButton.setOpacity(1);
		writeButton.setOpacity(1);
		wtxt.setOpacity(1);
		rtxt.setOpacity(1);
		int inUse;


		when background.eventEE(int flags, real x, real y)
			if(inUse==0)
			 readButton.setOpacity(flags & EB_ENTER ? 255 : 0);
			 writeButton.setOpacity(flags & EB_ENTER ? 255 : 0);
			 rtxt.setOpacity(flags & EB_ENTER ? 255 : 0);
			 wtxt.setOpacity(flags & EB_ENTER ? 255 : 0);
			 end;
		end;

		when readButton.eventEE(int flags, real x, real y)
			if(inUse==0)
			 readButton.setBrush(flags & EB_ENTER ? brushsS[address%NCPU] :  brushsH[address%NCPU]);
			end;
		end;

		when writeButton.eventEE(int flags, real x, real y)
			if(inUse==0)
			 writeButton.setBrush(flags & EB_ENTER ? brushsS[address%NCPU] :  brushsH[address%NCPU]);
			end;
		end;

		when readButton.eventLB(int flags, real x, real y)
		if (flags & EB_LEFT==0 && inUse==0 )
					inUse=1;
					start(1);
					readButton.setOpacity(255);
					rtxt.setOpacity(255);
					writeButton.setOpacity(255);
					wtxt.setOpacity(255);
					readButton.setBrush( brushsS[address%NCPU]);
		MESIF(address,READ);
					readButton.setBrush( brushsH[address%NCPU]);
					readButton.setOpacity(1);
					rtxt.setOpacity(1);
					writeButton.setOpacity(1);
					wtxt.setOpacity(1);
					inUse=0;
		end;
		end;

		when writeButton.eventLB(int flags, real x, real y)
		if (flags & EB_LEFT==0 && inUse==0)
					inUse=1;
					while(EB_CTRL==1)
						wait(1);
					end;
					readButton.setOpacity(255);
					rtxt.setOpacity(255);
					writeButton.setOpacity(255);
					wtxt.setOpacity(255);
					writeButton.setBrush( brushsS[address%NCPU]);
		MESIF(address,WRITE);
					readButton.setBrush( brushsH[address%NCPU]);
					readButton.setOpacity(1);
					rtxt.setOpacity(1);
					writeButton.setOpacity(1);
					wtxt.setOpacity(1);
					inUse=0;
		end;
		end;


end;



cpuMemory= Memory();
cpuCache=Cache();
cpuDataLines=DataLines(x+w2*0.3,y+h*0.421);
cpuDataLines2=DataLines(x+w2*0.3,y+h*0.68);
cpudialogBox=dialogBox();

r = Rectangle2(0, 0, blackpen, gray192brush,x+gap,y+h*0.684, w2, h*0.4);
tempAddress=0;
```

```
		for(i=0;i<4;i++)
			for(int j=0;j<NCPU;j++)
				rwbuttons(x+gap*2.5+j*(h2*1.25+gap),y+h*0.684+gap+i*(h2*1.25+gap),h2*1.25,h2*1.25,tempAddress);
				tempAddress++;
			end;
		end;




end;



cpu[0]=CPU(50+offset*2,50+offset,0);
cpu[1]=CPU(1200+offset*2,50+offset,1);
cpu[2]=CPU(50+offset*2,1200+offset,2);
cpu[3]=CPU(1200+offset*2,1200+offset,3);
cpu[4]=CPU(2349+offset*2,1200+offset,4);




//if the address is stored in a cpus memory it accesses it, the access can be canceld at any time
function addressCheck(int address,int cpuA,int cpuB)

			if(address%NCPU==cpuB)
				while(DRAMLock[cpuB]==1)
					wait(1);
				end;
				DRAMLock[cpuB]=1;
				cpu[cpuB].cpuDataLines.moveUp(TICKS*2, 1);
				if(cpu[cpuA].cancelMemoryAccess==1)
					DRAMLock[cpuB]=0;
					return;
				end;
				cpu[cpuB].cpuMemory.highlight(address/4,0);
				if(cpu[cpuA].cancelMemoryAccess==1)
					DRAMLock[cpuB]=0;
					return;
				end;
				cpu[cpuB].cpuDataLines.moveDown(TICKS*2, 1);
				DRAMLock[cpuB]=0;

				cpu[cpuA].memoryAccessComplete=1;
			end;
end;


//checks a cache for a valid copy
function checkCache(int cpuA,int cpuB,int address,int flag,int invalidate)


		if(invalidate==0)
		//if cpuB is currently sending probes for this data reply with a conflict message
			if(cpu[cpuB].probeCycle==cpu[cpuA].probeCycle && cpu[cpuB].probeCycle!=-1 && cpu[cpuA].probeCycle!=-1 &&cpuA!=cpuB)


				if(cpu[cpuB].isCpuWriting==1)
					cpu[cpuA].conflictQueue[address%2].addProcess(cpuB+10);
					linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"CNFLI:a");

				else
					cpu[cpuA].conflictQueue[address%2].addProcess(cpuB);
					linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"CNFL:a");
				end;
				if(cpu[cpuA].conflicts=="")
					cpu[cpuA].conflicts=cpu[cpuA].conflicts+format("%d",cpuB);
				else
					cpu[cpuA].conflicts=cpu[cpuA].conflicts+format(",%d",cpuB);
				end;
				cpu[cpuA].probeConflict=1;

				if(address%NCPU==cpuB)
					cpu[cpuA].memoryAccessComplete=1;
				end;
				cpu[cpuA].repliesReturned++;
				message=format("Send Probe\nto all Nodes,\nAcks Returned %d/4", cpu[cpuA].repliesReturned-1);
				cpu[cpuA].cpudialogBox.setCurrentTask(message);
				return;
			end;
			//if cpuB has send its read message then halt the ack until it recieves a reply from the cpu
			if(cpu[cpuB].readCycle==cpu[cpuA].probeCycle && cpu[cpuB].readCycle!=-1 && cpu[cpuA].probeCycle!=-1 && cpuA!=cpuB)
				while(cpu[cpuB].readCycle==cpu[cpuA].probeCycle)
```

```
                        wait(1);
                end;
                if(cpu[cpuB].isCpuWriting==1)

                        linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"CNFLI:");
                else

                        linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"CNFL:a");
                end;
                if(cpu[cpuA].conflicts=="")
                        cpu[cpuA].conflicts=cpu[cpuA].conflicts+format("%d",cpuB);
                else
                        cpu[cpuA].conflicts=cpu[cpuA].conflicts+format(",%d",cpuB);
                end;
                cpu[cpuA].repliesReturned++;
          cpu[cpuA].readConflict=1;
                if(address%NCPU==cpuB)
                        cpu[cpuA].memoryAccessComplete=1;
                end;
                message=format("Send Probe\nto all Nodes,\nAcks Returned %d/4", cpu[cpuA].repliesReturned-1);
                cpu[cpuA].cpudialogBox.setCurrentTask(message);
                return;
        end;

        if(cpu[cpuB].cpuCache.address[address%2]==address   )

                while(cpu[cpuB].waitingOnHomeNodeAck[address]==1)
                        wait(1);
                end;

                if(cpu[cpuB].cpuCache.address[address%2]==address&&(
                        cpu[cpuB].cpuCache.state[address%2]==MODIFIED || cpu[cpuB].cpuCache.state[address%2]==EXCLUSIVE
                        ||cpu[cpuB].cpuCache.state[address%2]==FORWARD))
                        cpu[cpuA].mesifUpdateMemory=1;
                        if(cpu[cpuB].cpuCache.state[address%2]==MODIFIED)
                                cpu[cpuB].cpuCache.state[address%2]=INVALID;
                                cpu[cpuB].cpuCache.stateRTxt[address%2].setTxt("I");
                                cpu[cpuA].cacheLineFound=cpuB;
                                cpu[cpuB].cpuCache.highlight(address%2, 0);
                                cpu[cpuA].lineModified=1;
                                cpu[cpuB].waitingOnHomeNodeAck[address]=1;

                                linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"DataM:a");
                        else
                                cpu[cpuB].cpuCache.state[address%2]=SHARED;
                                cpu[cpuB].cpuCache.stateRTxt[address%2].setTxt("S");
                                cpu[cpuA].cacheLineFound=cpuB;
                                cpu[cpuB].cpuCache.highlight(address%2, 0);
                                cpu[cpuB].waitingOnHomeNodeAck[address]=1;
                                linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"DataF:a");
                        end;

                        cpu[cpuA].repliesReturned++;
                        if(address%NCPU==cpuB)
                                cpu[cpuA].memoryAccessComplete=1;
                        end;
                        message=format("Send Probe\nto all Nodes,\nAcks Returned %d/4", cpu[cpuA].repliesReturned-1);
                cpu[cpuA].cpudialogBox.setCurrentTask(message);
                        return;
                end;

        end;
        fork(addressCheck(address,cpuA,cpuB));

        linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"IA:a");

        cpu[cpuA].repliesReturned++;
        message=format("Send Probe\nto all Nodes,\nAcks Returned %d/4", cpu[cpuA].repliesReturned-1);
                cpu[cpuA].cpudialogBox.setCurrentTask(message);

end;

if(invalidate==1)
//if cpuB is currently sending probes for this data reply with a conflict message
        if(cpu[cpuB].probeCycle==cpu[cpuA].probeCycle && cpu[cpuB].probeCycle!=-1 && cpu[cpuA].probeCycle!=-1 &&cpuA!=cpuB)


                if(cpu[cpuB].isCpuWriting==1)
                        cpu[cpuA].conflictQueue[address%2].addProcess(cpuB+10);
                        linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"CNFLI:a");

                else
                        cpu[cpuA].conflictQueue[address%2].addProcess(cpuB);
                        linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"CNFL:a");
                end;
                if(cpu[cpuA].conflicts=="")
                        cpu[cpuA].conflicts=cpu[cpuA].conflicts+format("%d",cpuB);
```

```
		else
			cpu[cpuA].conflicts=cpu[cpuA].conflicts+format(",%d",cpuB);
		end;
		cpu[cpuA].probeConflict=1;

		if(address%NCPU==cpuB)
			cpu[cpuA].memoryAccessComplete=1;
		end;
		cpu[cpuA].repliesReturned++;
		message=format("Send Probe Invalidate\nto all Nodes,\nAcks Returned %d/4", cpu[cpuA].repliesReturned-1);
		cpu[cpuA].cpudialogBox.setCurrentTask(message);
		return;
end;
//if cpuB has send its read message then halt the ack until it recieves a reply from the cpu
if(cpu[cpuB].readCycle==cpu[cpuA].probeCycle && cpu[cpuB].readCycle!=-1 && cpu[cpuA].probeCycle!=-1 && cpuA!=cpuB)
		while(cpu[cpuB].readCycle==cpu[cpuA].probeCycle)
			wait(1);
		end;
		if(cpu[cpuB].isCpuWriting==1)
			linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"CNFLI:a");
		else

			linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"CNFL:a");
		end;
		if(cpu[cpuA].conflicts=="")
			cpu[cpuA].conflicts=cpu[cpuA].conflicts+format("%d",cpuB);
		else
			cpu[cpuA].conflicts=cpu[cpuA].conflicts+format(",%d",cpuB);
		end;
		cpu[cpuA].repliesReturned++;
  cpu[cpuA].readConflict=1;
		if(address%NCPU==cpuB)
			cpu[cpuA].memoryAccessComplete=1;
		end;
		message=format("Send Probe Invalidate\nto all Nodes,\nAcks Returned %d/4", cpu[cpuA].repliesReturned-1);
		cpu[cpuA].cpudialogBox.setCurrentTask(message);
		return;
end;

fork(addressCheck(address,cpuA,cpuB));
if(cpu[cpuB].cpuCache.address[address%2]==address  )

		while(cpu[cpuB].waitingOnHomeNodeAck[address]==1)
			wait(1);
		end;

		if(cpu[cpuB].cpuCache.address[address%2]==address&&(
			cpu[cpuB].cpuCache.state[address%2]==MODIFIED || cpu[cpuB].cpuCache.state[address%2]==EXCLUSIVE
			||cpu[cpuB].cpuCache.state[address%2]==FORWARD))


			if(cpu[cpuB].cpuCache.state[address%2]==MODIFIED)
				cpu[cpuA].lineModified=1;
				cpu[cpuA].mesifUpdateMemory=1;
			cpu[cpuB].cpuCache.state[address%2]=INVALID;
			cpu[cpuB].cpuCache.stateRTxt[address%2].setTxt("I");
			cpu[cpuA].cacheLineFound=cpuB;
			cpu[cpuB].cpuCache.highlight(address%2, 0);
			cpu[cpuB].waitingOnHomeNodeAck[address]=1;
				linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"DataM:a");
			else
				if(cpuA!=cpuB)
					cpu[cpuA].mesifUpdateMemory=1;
					cpu[cpuB].cpuCache.state[address%2]=INVALID;
					cpu[cpuB].cpuCache.stateRTxt[address%2].setTxt("I");
				end;
			cpu[cpuA].cacheLineFound=cpuB;
			cpu[cpuB].cpuCache.highlight(address%2, 0);
			cpu[cpuB].waitingOnHomeNodeAck[address]=1;
				linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"DataE:a");
			end;
			cpu[cpuA].repliesReturned++;
			message=format("Send Probe Invalidate\nto all Nodes,\nAcks Returned %d/4", cpu[cpuA].repliesReturned-1);
			cpu[cpuA].cpudialogBox.setCurrentTask(message);
			if(address%NCPU==cpuB)
				cpu[cpuA].memoryAccessComplete=1;
			end;
			return;
		end;
		if(cpuA!=cpuB)
		cpu[cpuB].cpuCache.state[address%2]=INVALID;
		cpu[cpuB].cpuCache.stateRTxt[address%2].setTxt("I");
		end;
end;
fork(addressCheck(address,cpuA,cpuB));
linkPath(cpuB,cpuA,flag,brushsH[cpuA],brushsH[cpuB],address,"IA:a");
cpu[cpuA].repliesReturned++;
```

```
                    message=format("Send Probe Invalidate\nto all Nodes,\nAcks Returned %d/4", cpu[cpuA].repliesReturned-1);
                    cpu[cpuA].cpudialogBox.setCurrentTask(message);
            end;


end;


resetFont =Font("Times New Roman", 40, 0);
resetButton = SimpleButton(LOGICALW-215, LOGICALH-75, 160, 60, bgbrush, gray192brush, blackbrush,resetFont , "Reset");
when resetButton.button.eventLB(int flags, real x, real y)
            if (flags & EB_LEFT == 0)

                    reset();
            end;
end;
```