Advanced Computer Architectures

Multiprocessor systems and memory coherence problem

Czech Technical University in Prague, Faculty of Electrical Engineering
Slides authors: Michal Štepanovský, update Pavel Píša

Content

Multiprocessor systems

Change of meaning: Multiprocessor systems = system with multiple processors. Toady term processor can refer even to package/silicon with multiple cores.

Software point of view (how programmer seems the system):

- Shared memory systems - SMS. Single operating system (OS, single image), Standard: OpenMP, MPI (Message Passing Interface) can be used as well.
  - Advantages: easier programming and data sharing
- Distributed memory systems - DMS: communication and data exchange by **message passing**. The unique instance of OS on each node (processor, a group of processors - hybrid). Network protocols, RPC, Standard: MPI. Sometimes labeled as NoRMA (No Remote Memory Access)
  - Advantages: less HW required, easier scaling
  - Often speedup by Remote Direct Memory Access (RDMA), Remote Memory Access (RMA), i.e., for InfiniBand

Multiprocessor systems

Hardware point of view:

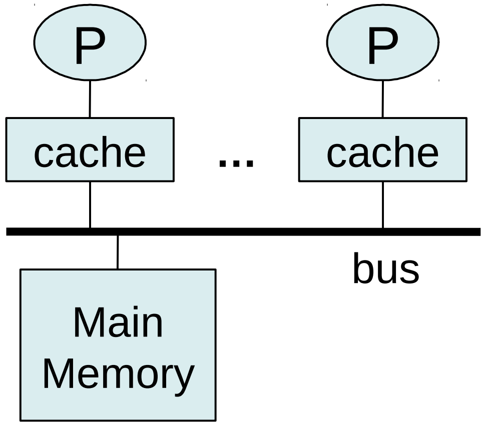
- Shared memory systems – single/common physical address-space – SMP: UMA
- Distributed memory system – memory physically distributed to multiple nodes, address-space private to the node (cluster) or global i.e., NUMA, then more or less hybrid memory organization

Definitions:

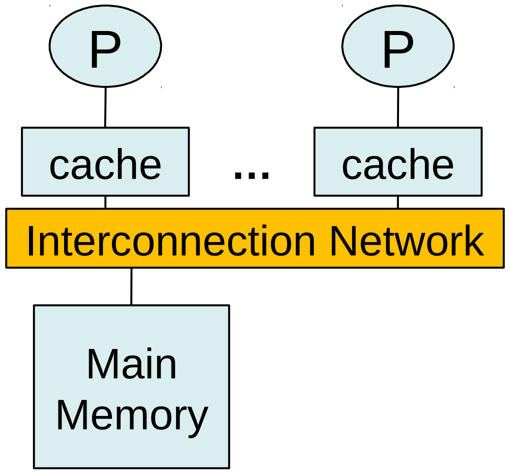
- **SMP** – Symmetric multiprocessor – processors are connected to the central common memory. Processors are **identical** and „access“ memory and the rest of the system same way (by same address, port, instructions).
- **UMA** – Uniform Memory Access – all memory ranges are accessed from different CPUs in the same time. UMA systems are the most often implemented as SMP today.
- **NUMA** – Non-Uniform Memory Access – memory access time depends on accessed location (address) and initiating processor/node. Faster node local, slower remote.
- **ccNUMA** – cache-coherent NUMA – coherence is guaranteed by HW resources
- **Cluster** – a group of cooperating computers with a fast interconnection network and SW which allows viewing the group as a single computing system.

Multiprocessor systems - examples

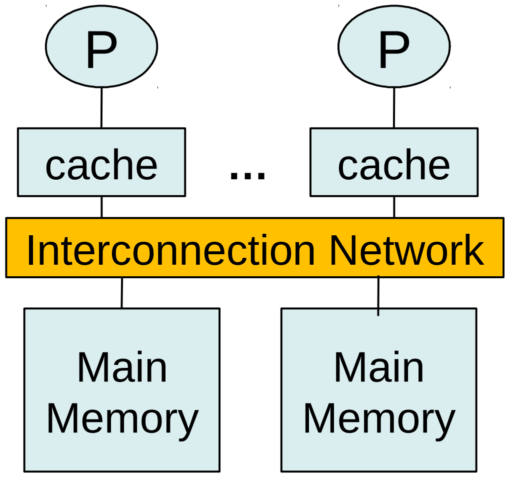
Traditional SMP, and UMA



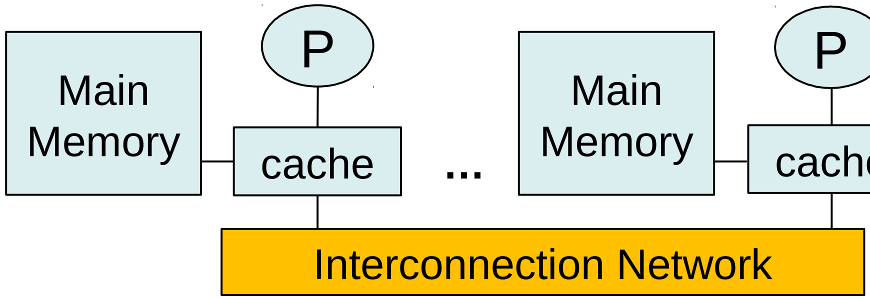
UMA



UMA



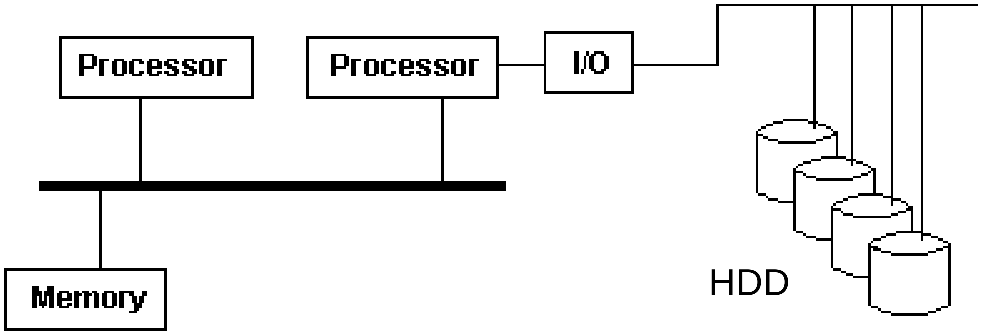
NUMA

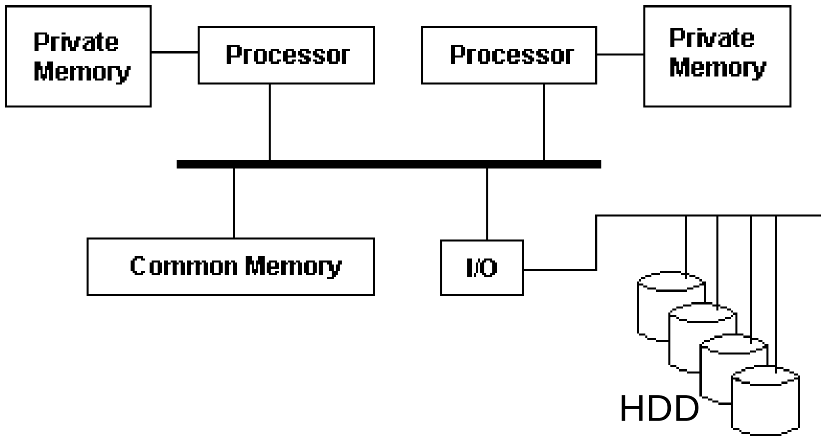


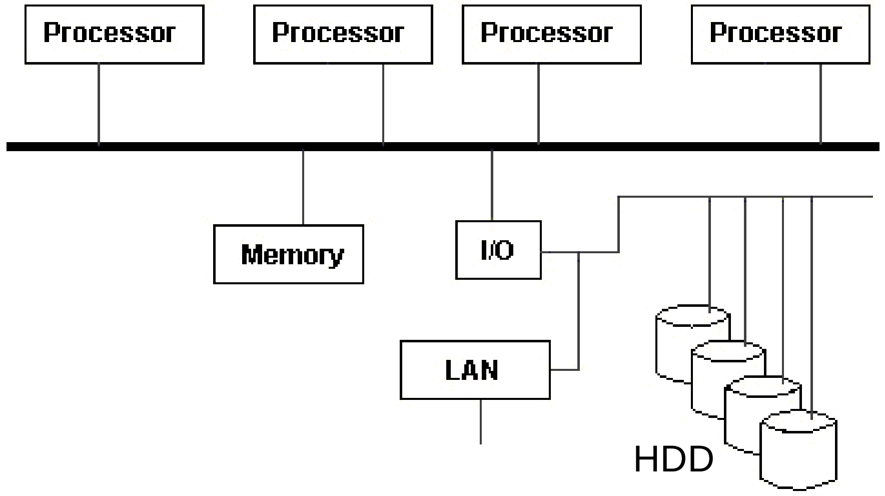
Interconnection Network

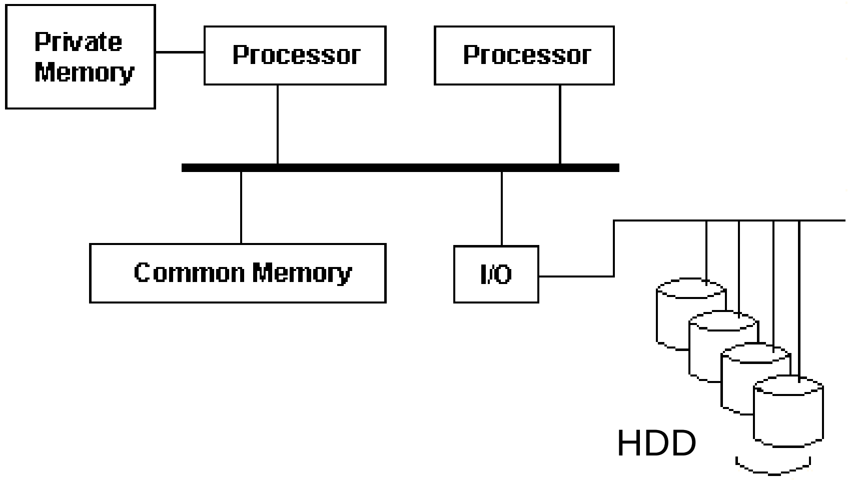
It can be a crossbar switch, or a dynamic multi-stage interconnection network, or some type of fast static network

Try to find SMP system ???



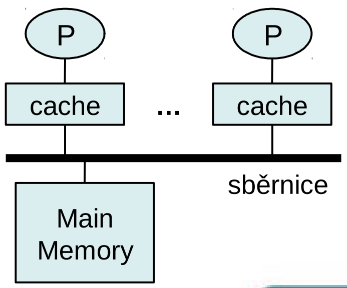


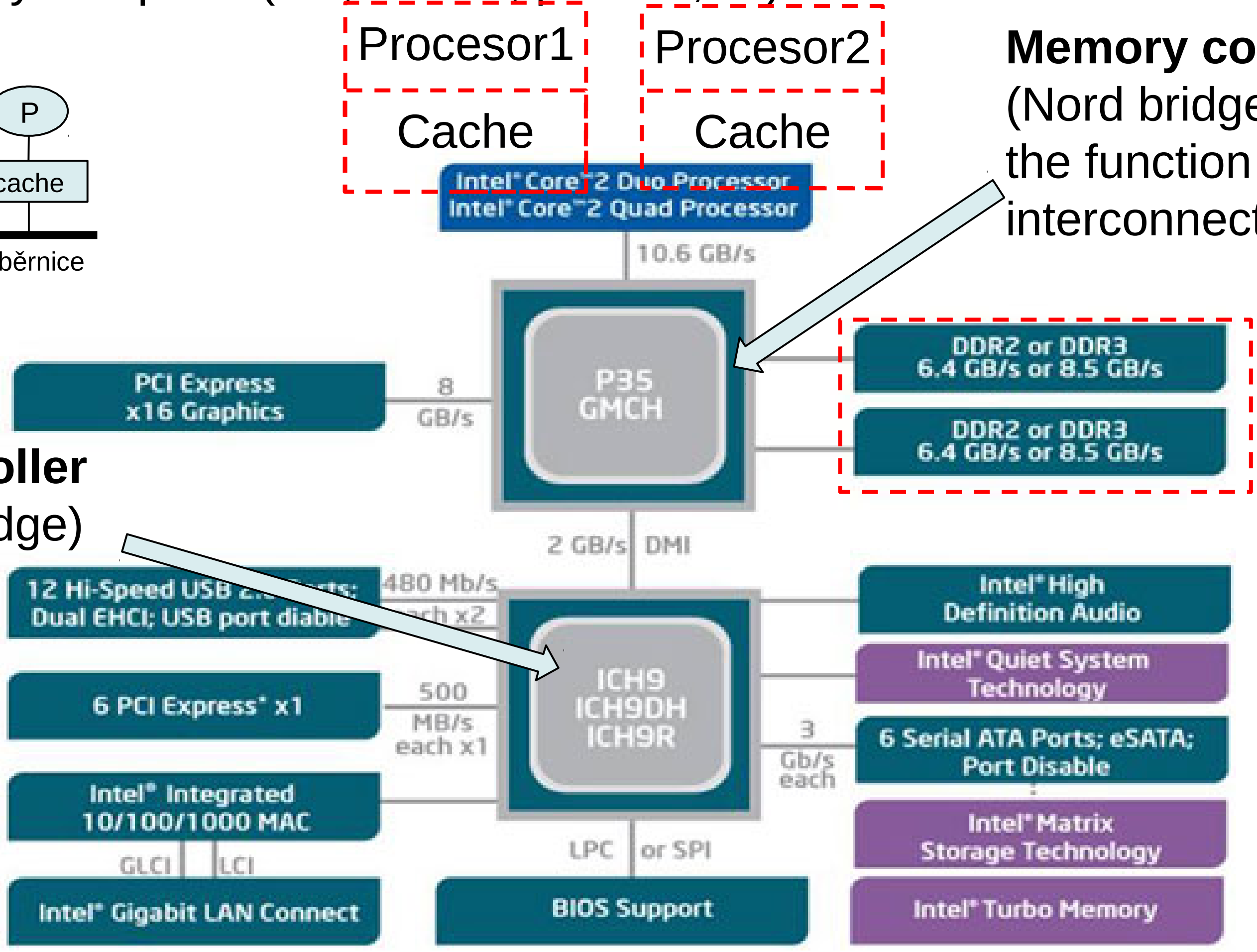




PC – from SMP to NUMA development

- Multiprocessor systems have existed for decades and have been used for demanding computing in science and commerce ...
- With the introduction of the first multi-core processors is the SMP available to ordinary computer (PC, tablets, phones, …) users




Procesor1
Procesor2
Cache
Cache
Intel® Core™2 Duo Processor
Intel® Core™2 Quad Processor
10.6 GB/s
P35 GMCH
PCI Express x16 Graphics
8 GB/s
DDR2 or DDR3 6.4 GB/s or 8.5 GB/s
DDR2 or DDR3 6.4 GB/s or 8.5 GB/s
2 GB/s DMI
12 Hi-Speed USB 2.0 Ports; Dual EHCI; USB port diable
480 Mb/s each x2
6 PCI Express* x1
500 MB/s each x1
ICH9 ICH9DH ICH9R
Intel® Integrated 10/100/1000 MAC
GLCI
LCI
Intel® Gigabit LAN Connect
LPC or SPI
BIOS Support
Intel® High Definition Audio
Intel® Quiet System Technology
3 Gb/s each
6 Serial ATA Ports; eSATA; Port Disable
Intel® Matrix Storage Technology
Intel® Turbo Memory

Memory controller (Nord bridge) – fulfills the function of bus/ interconnection network

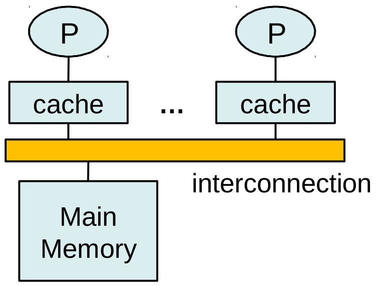
Main memory

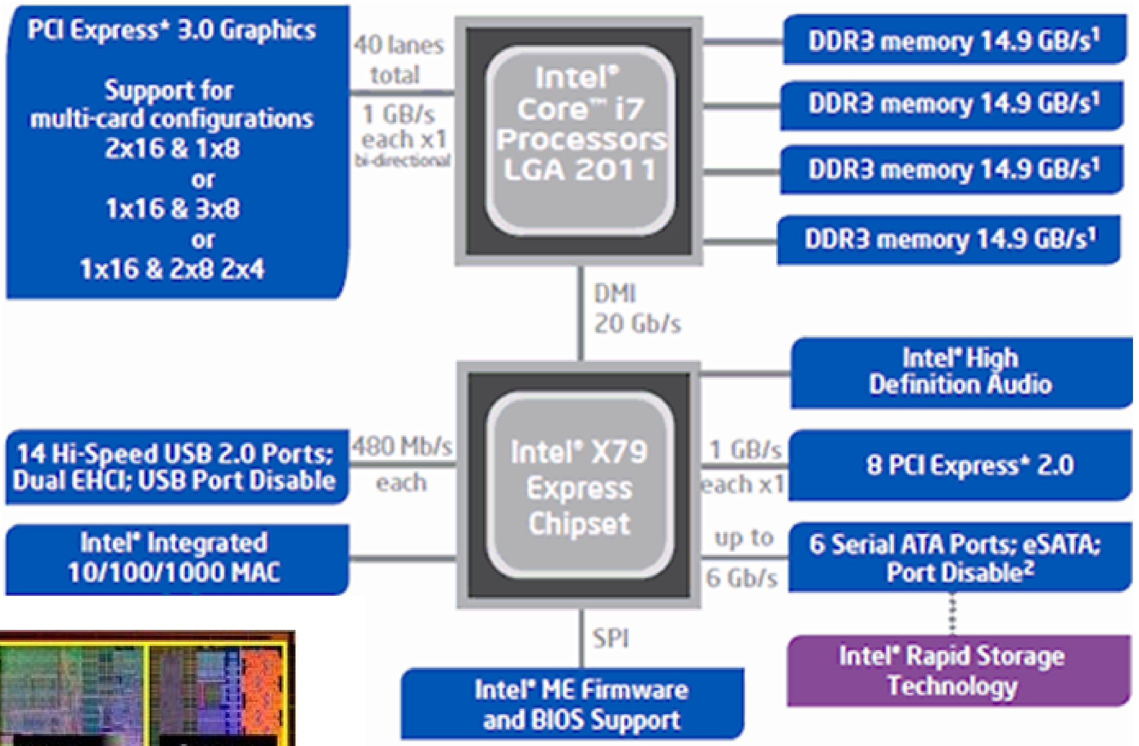
I/O controller (south bridge)

http://www.intel.de

PC – from SMP to NUMA development

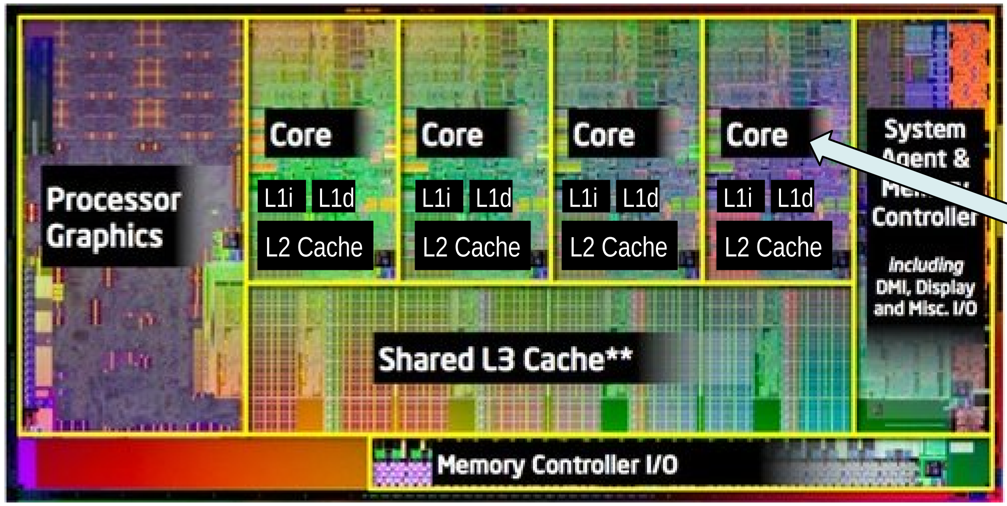
- Further development shifted the function of the northern bridge directly into the processor.
  The memory controller can
- be found there.




PCI Express* 3.0 Graphics
Support for multi-card configurations 2x16 & 1x8 or 1x16 & 3x8 or 1x16 & 2x8 2x4
40 lanes total
1 GB/s each x1 bi-directional
Intel® Core™ i7 Processors LGA 2011
DDR3 memory 14.9 GB/s¹
DDR3 memory 14.9 GB/s¹
DDR3 memory 14.9 GB/s¹
DDR3 memory 14.9 GB/s¹
DMI 20 Gb/s
Intel® High Definition Audio
14 Hi-Speed USB 2.0 Ports; Dual EHCI; USB Port Disable
480 Mb/s each
Intel® X79 Express Chipset
1 GB/s each x1
8 PCI Express* 2.0
Intel® Integrated 10/100/1000 MAC
up to 6 Gb/s
6 Serial ATA Ports; eSATA; Port Disable²
SPI
Intel® ME Firmware and BIOS Support
Intel® Rapid Storage Technology

http://www.intel.de

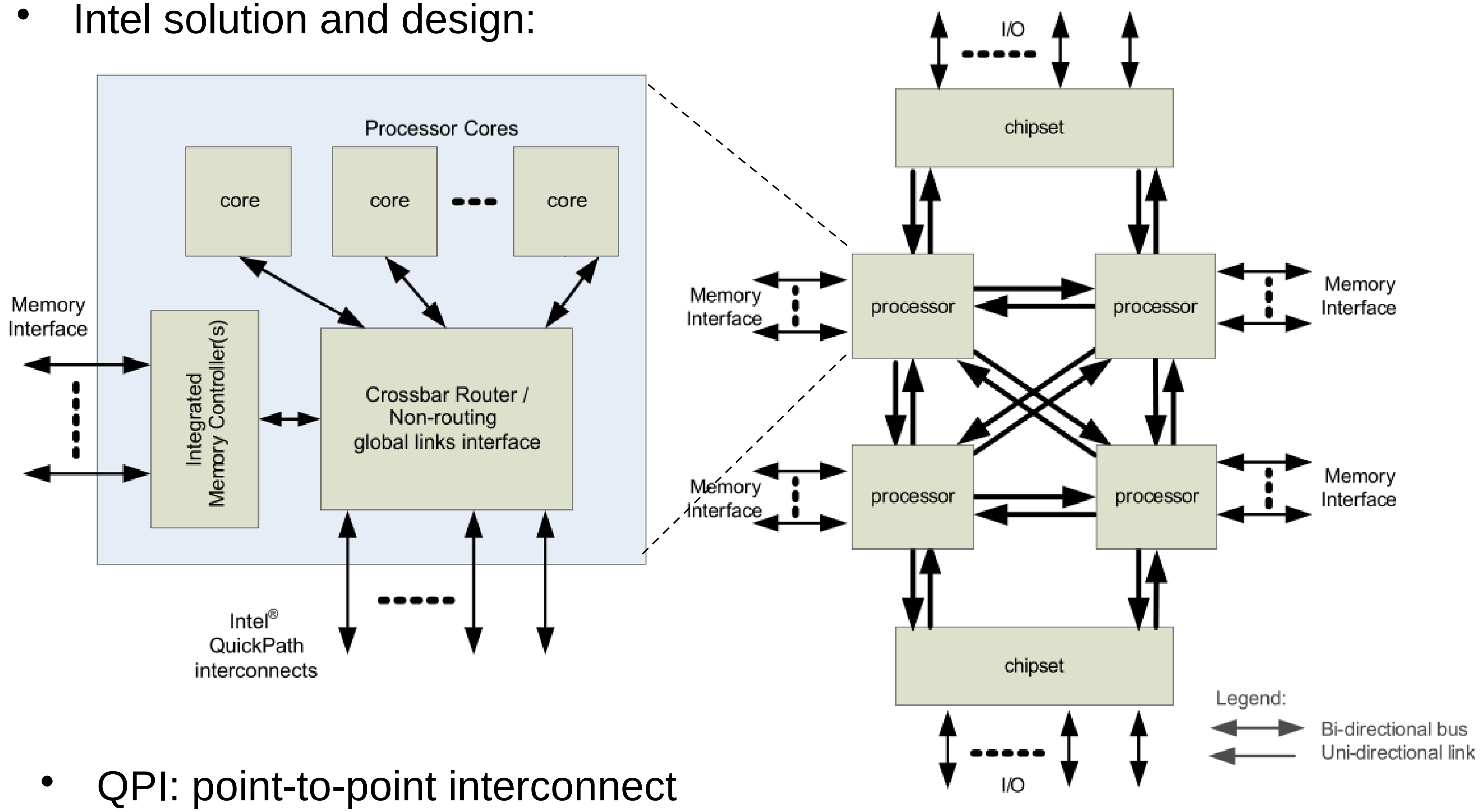
Core i7-2600K



The core can be seen as (de facto is) processor. If the L3 cache memory is inclusive then it can be seen as whole main memory.

PC – from SMP to NUMA development

- QuickPath Interconnect allows to mutually interconnect multiple processors… As a result, common PC becomes a **NUMA** system.
- Intel solution and design:

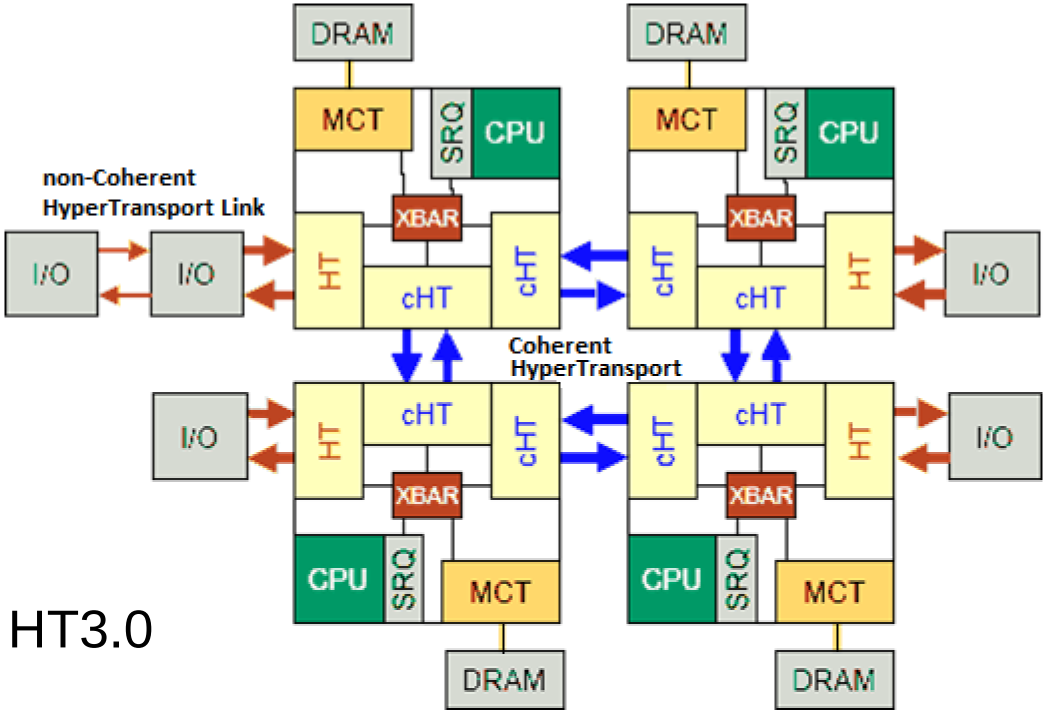


- QPI: point-to-point interconnect

PC – from SMP to NUMA development

- AMD solution and design (HT - HyperTransport):

Motherboard



- **Four AMD Opteron™** 6000 series processors (Socket G34) **16/12/8/4-Core ready**; HT3.0 Link support
- **Up to 1TB DDR3** 1600MHz
- 6x SATA2
- 1x PCI-E 2.0 x16 slot

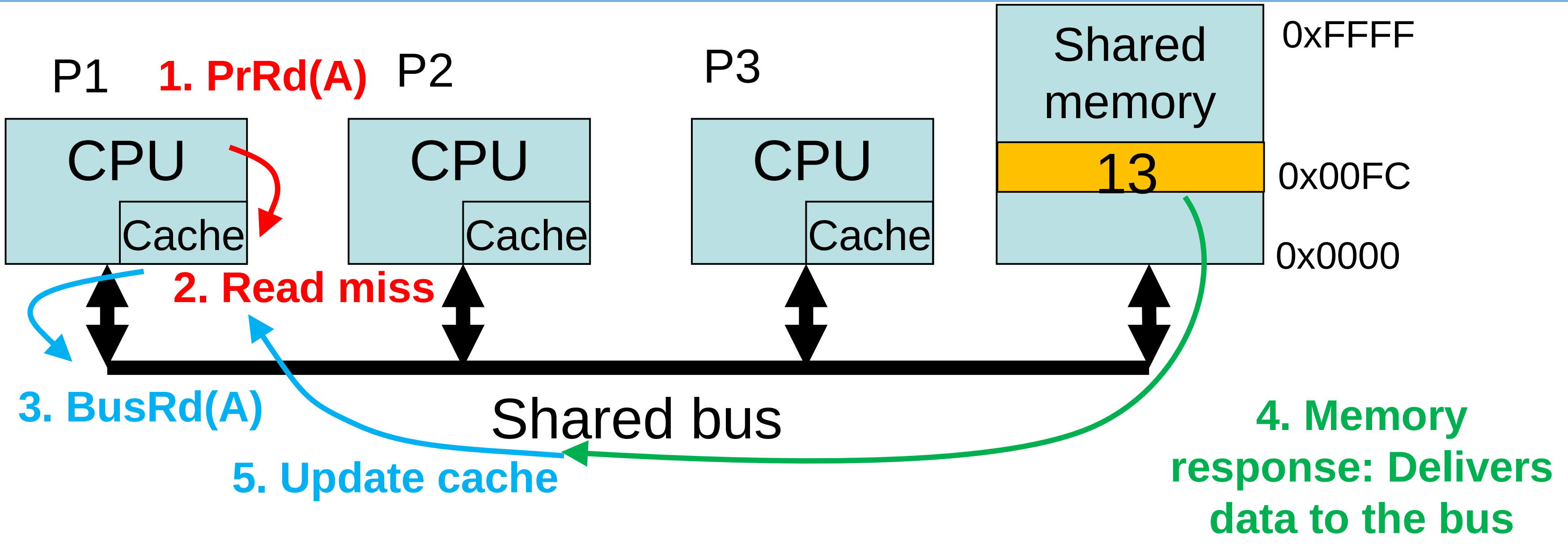
Shared memory systems

This lecture focus:

Shared memory systems

- Often developed as an extension of the single-processor computers by adding additional processors (or cores).
- Traditional single-processor OSs were extended to schedule processes for multiple processors.
- Multi-processor / multi-threaded program runs on single-processor system in timesharing mode but uses multiple processors, if they are available.
- A suitable model for tasks with significant data sharing, data are shared automatically. Sharing is solved transparently in HW. Be careful about synchronization/race conditions.
- Difficult to scale for larger numbers of processors.

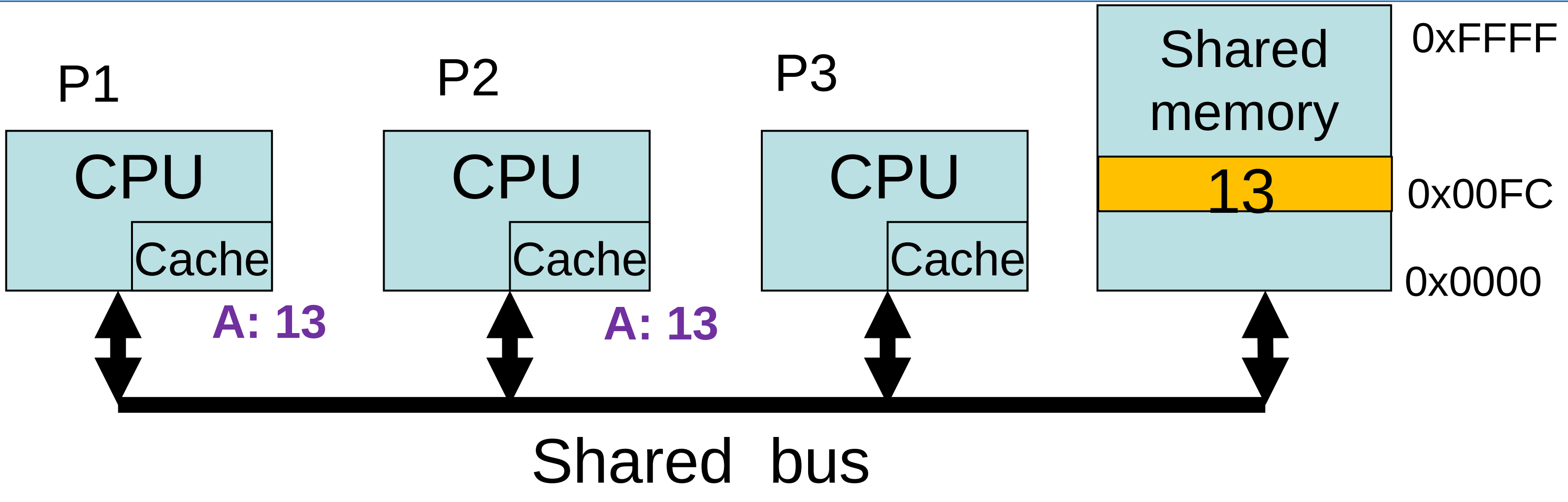
What is the basic problem? Consider the write-back cache



Processor P1 intent to read data from address A (0x00FC):

1. Processor P1 inquiry PrRd(A) to own cache.
2. Data are not found in the cache => cache read miss
3. Cache/bus controller of P1 sends BusRd(A) request to the bus.
4. The memory controller recognizes the request to read from given address (0x00FC) and provides data, value 13 for this example.
5. The cache controller of P1 receives bus data and stores them in cache.

What is the basic problem? Consider the write-back cache



Processor P1 requests data from address A (0x00FC). Result?

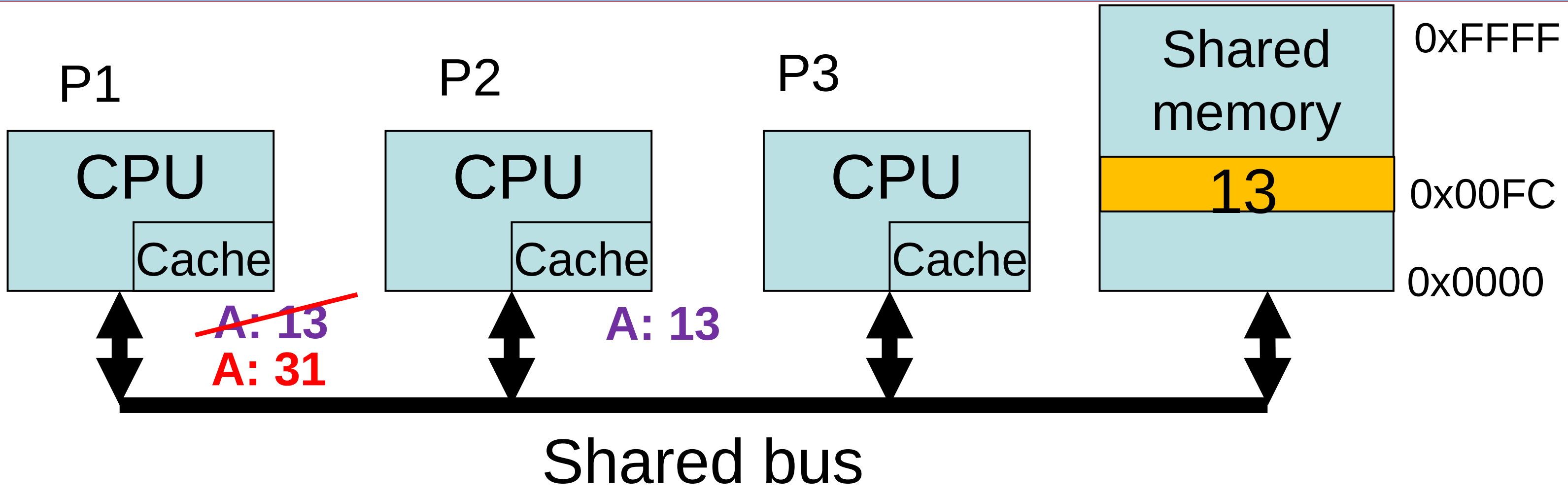
- P1 received data into its cache. A: 13 (address:value)

Processor P2 requests data from address A (0x00FC). Result?

- The same. P2 received data from memory stores then to cache. A: 13

The processors can read/access data from A independently from their caches, no need for bus activity, great scalability but…

What is the basic problem? Consider the write-back cache



Processor P1 requests data from address A (0x00FC). Result?

- P1 received data into its cache. A: 13 (address:value)

Processor P2 requests data from address A (0x00FC). Result?

- The same. P2 received data from memory stores then to cache. A: 13

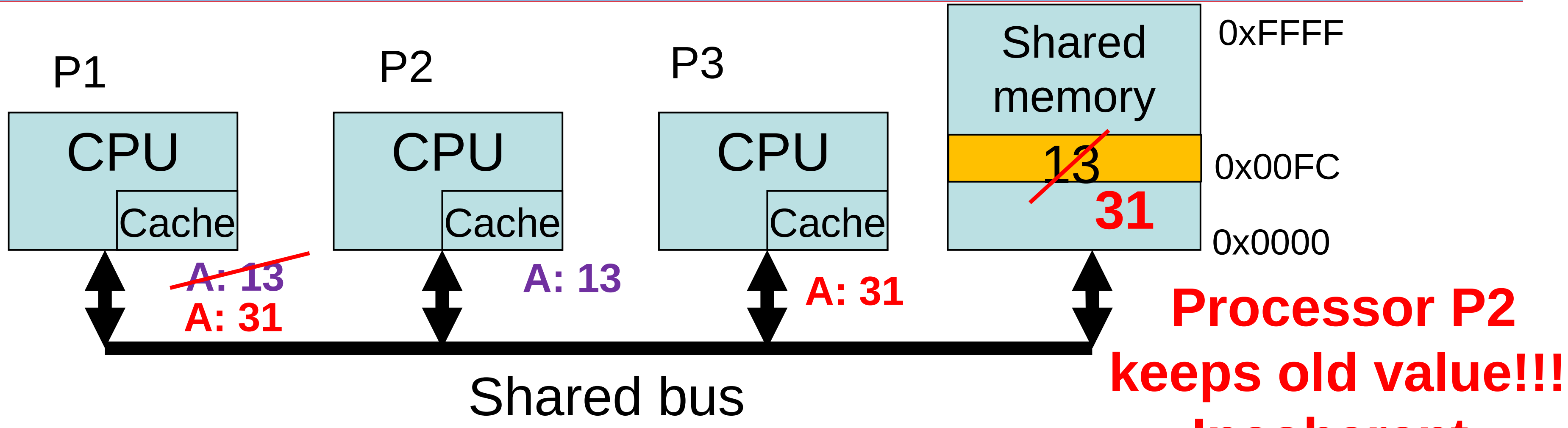
Processor P1 writes the new value into A. 31 for example.

- The value in its cache is modified.

Processor P3 requests to read from address A. Result value?

- Memory provides **13**. But processor P1 works with **31**. **> Incoherent**

What is the basic problem? Consider write-through cache



Processor P1 requests data from address A (0x00FC). Result?

- P1 received data into its cache. A: 13 (address:value)

Processor P2 requests data from address A (0x00FC). Result?

- The same. P2 received data from memory stores then to cache. A: 13

Processor P1 writes the new value into A. 31 for example.

- The value in its cache is modified and propagates into memory**.**

Processor P3 requests data from address A. Result?

- Memory provides value **31**. But P1 works with **31**. **> Coherent?**

The result of the previous analysis

- The problem lies in memory **incoherence**:
- Processor modified (appropriately) data in its cache
- Even immediate change in main memory is not enough.
- Cache memories of other processors can keep outdated data.

Important definitions:

- Memory coherence => toady lecture
  - Rules regarding access to individual memory locations
- Memory consistency => next lecture
  - Rules for all memory operations in the parallel computer

    Sequential consistency: "The computer is **sequentially consistent** if the result of the execution of the program is the same as if the operations on all the processors were performed in sequential order and the operations of each individual processor appear in that sequence in the order given by their program.”

Memory coherence

- Definition: We say that a multiprocessor memory system is **coherent** if
  - the results of any execution of a program are such that for each location, it is possible to construct a hypothetical serial order of all operations (reads and writes) to the location that is consistent with the results of the execution and in which:
  - 1. Memory operations to a given memory location for each process are performed in the order in which they were initiated by the process.
  - 2. The values returned by each read operation are the values of the most recent write operation in a given memory location with respect to the serial order.
- Methods, which ensures memory coherence, are called coherence protocols.

The formal definition of memory coherent system

Definition 1. The *memory system* is coherent, *if*

(1) preserves the order of accesses initiated by one of processors/processes P: *Op.* Read(M[x]) *executed by* P_i after *op.* Write(M[x],V_1) *executed by* P_i, among which there is no other Write(M[x],V_*) *executed by other* P_j, *always returns* V_1.

(2) cache memories maintain coherent views: *op.* Read(M[x]) *executed by* P_i which follows *op.* Write(M[x],V_1) *executed by other* P_j *returns* V_1 *if operations* Read *and* Write are sufficiently separated (by time, barrier instructions) and any other *op.* Write(M[x],V_*) to address x is not *executed by other* P_k *in between.*

(3) *ensures* serialization of operations Write: two *ops.* Write *targeting* same SM cell (address) executed by *two processors* are seen by all *processors in the same order*.

This definition is entirely equivalent to the definition on the previous slide. Its advantage is the formal definition of the terms "memory operations" and "serialization of write operations".

The methods for coherence are called cache coherence protocols.

Maintaining memory coherence in the multiprocessor system

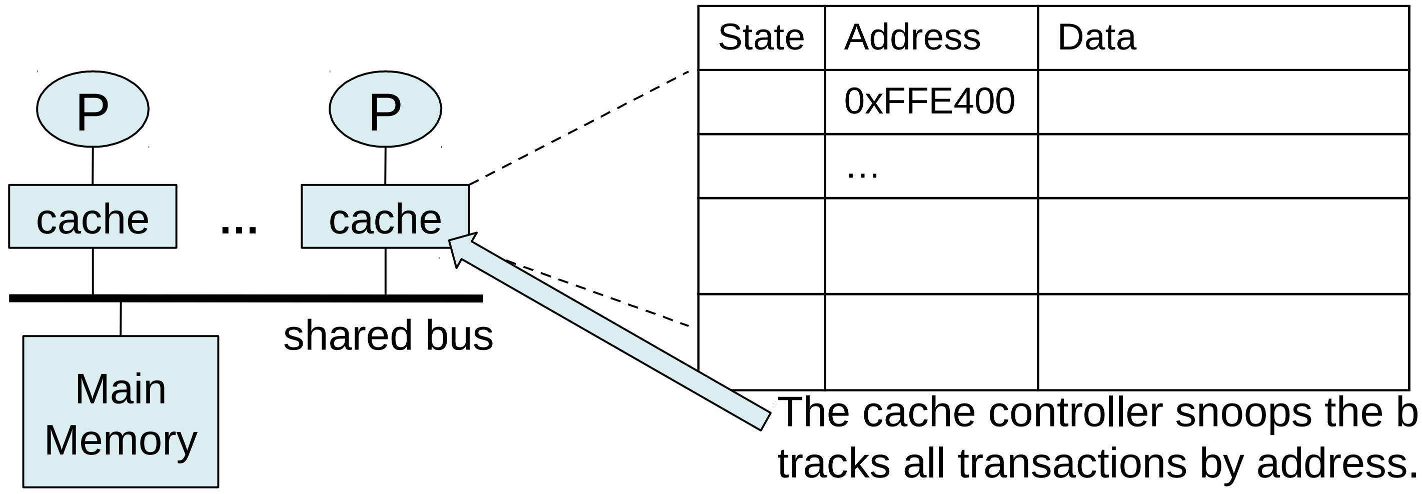
1. **Snooping**
   - Snooping = spy or (bus traffic) monitoring
   - Requires to supplement cache with HW which monitors transactions on the **bus** and
     - Detect operation in interest,
     - Actualizes state of relevant cache lines/data blocks,
     - Generates memory transactions
   - Protocols used in practice are MESI, MSI, MEI, MOESI, MESIF and some of their variants.

2. **Directory-based**
   - Used for larger scale systems where common shared bus cannot be used/implemented

Bus snooping

The variant where each processor knows which other processors have a copy of its cached data. It is too complex. Solution:

- Each cache controller snoops the bus for **write operations** relating to **addresses and data which are in its cache**
- Global shared bus is required to allow all cache controllers snoop transaction and receive operations is same order.
- The requirement for the global bus is the main problem for scalability.
- The variant using „Directory-based“ is better scalable.



State	Address	Data
	0xFFE400	
	…	

The cache controller snoops the bus and tracks all transactions by address. If the address matches the address in its cache, it responds appropriately.

Snooping protocols

Snooping protocols: write-update, write-invalidate

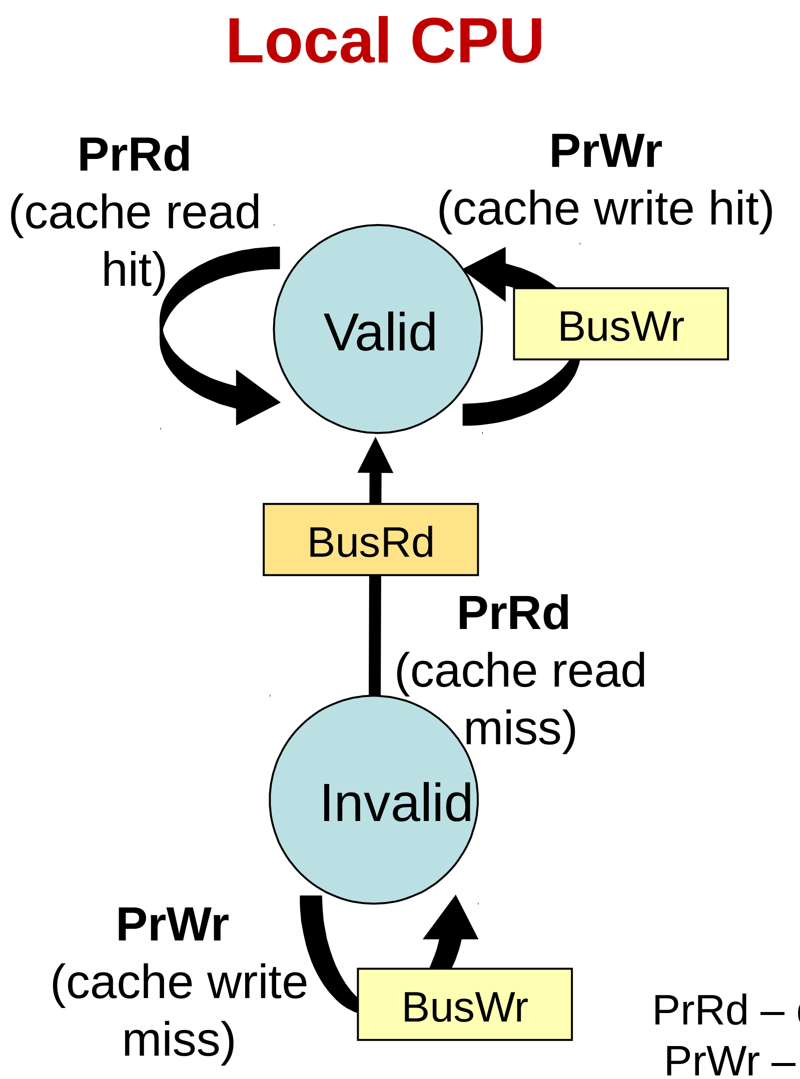
- Write-update
  - Before processor can write data, it has to acquire the bus. Then it sends new data. (All processors including memory are connected to the same bus)
  - **All snooping controllers actualize their data if address matches, and memory writes them unconditionally.**
  - The variant loads the bus heavily. I**t is not used.**

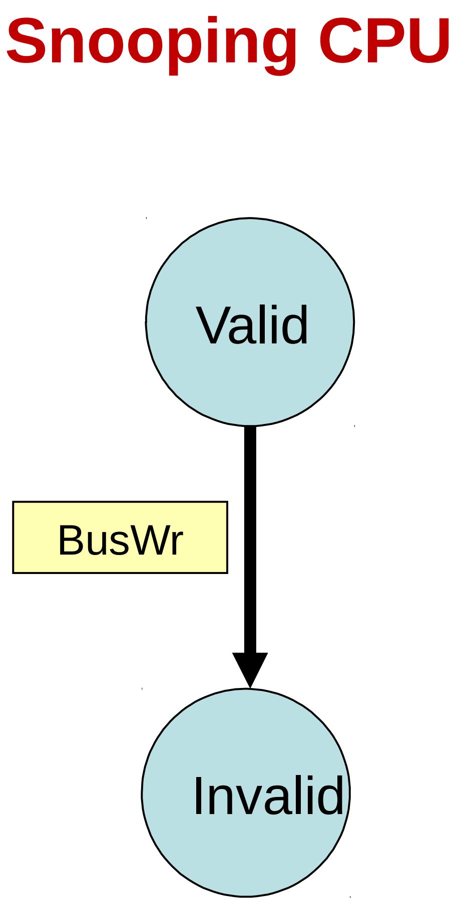
Snooping protocols

- Write-invalidate
  - The processor writes to some address. The **message** requesting **invalidation** of all places keeping data for that address is **sent**.
  - All snooping controllers invalidate address matching content in caches.
  - It ensures that there is an only a single copy of data corresponding to the cache line written by initiating processor. The processor can modify cache line without load the bus (strategy *Write allocate)*..
  - All other read requests initiated by other processors result in a cache miss, and the read request is visible on the bus.
  - **Write-invalidate protocol** requires to distinguish **at least two states** of the cache line **–** **valid (modified), invalid.**

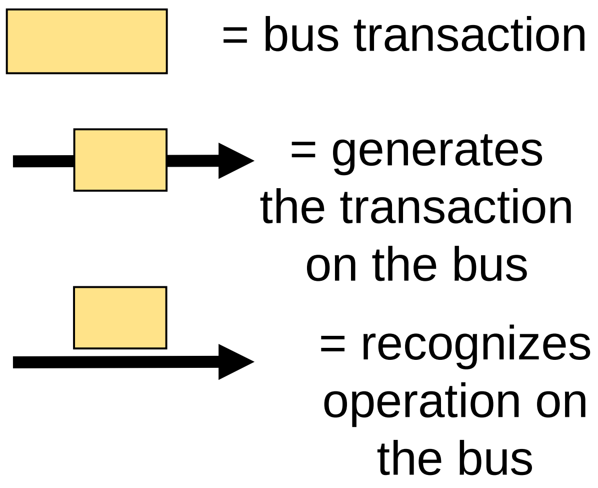
Protocol Write Through Write No Allocate

- WTWNA: **Invalidation protocol** with only **two cache-line states**





Legend:


= bus transaction
= generates the transaction on the bus
= recognizes operation on the bus

Observation: Each write to any address generates MemWrite transaction on bus…

PrRd – data read from given cache-line/block by processor
PrWr – data write to given cache-line/block by processor..

Scalability of WTWNA protocol

Example: Consider the SMP system, processor clock frequency **f = 1.6** GHz. Executed code statistic and more parameters:

- Average **IPC = 1**,
- **s = 15%** of all instructions are Write operations,
- Each write operation stores **L = 8** B.

Let is the throughput of global shared bus **b = 8** GB/s. How many processors can be included in such a system without bus saturation?

Solution:

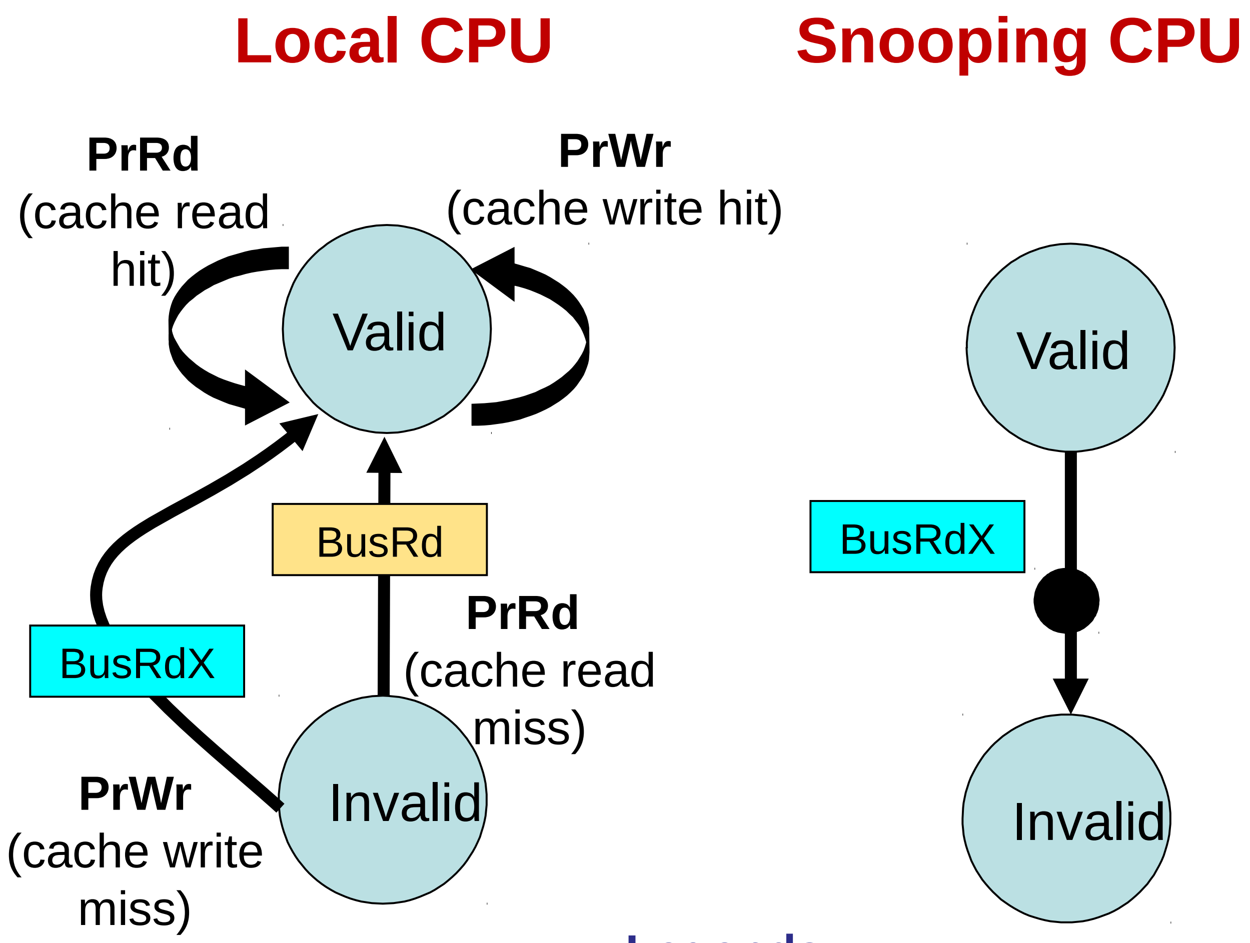
- Single P generates **w = s*IPC*f** Write operations per second
- Final bandwidth requirement of one P is **r = w*L** B/s.
- For our case, r=0.15*1*1.6G*8 = 1.92 GB/s
- Only four processors are satisfied even if we ignore bandwidth required for read miss situations

$$\mathbf{p = b/r = 8/1{,}92 \cong 4.}$$

Source slides by Prof. Ing. Pavel Tvrdík, CSc.

Write Back Write Allocate protocol

- WBWA: **Invalidation protocol** with only **two cache-line states**



Snooping core recognizes RWITM on the bus, if it has modified data, it aborts RWITM and writes data to memory (copy back). Initiating core repeats RWITM transaction. It is allowed this time and cache line state changes from *Invalid* to *Valid*.
Why so complicated?
Block has to be read the first (strategy *Write Allocate*) because data can be modified by other core on different offset in the cache-line. If we write the whole block, some bytes from snooping processor cache would be lost if it is in the valid state. That is why copy-back is required before invalidation.

Observation: Even if the processor **only reads** data, it has to write back to memory… = it writes back what is already in memory.

Legenda:

● = copy back

BusRdX is **RWITM** – Read with intent to modify.

WBWA protocol scalability

Example: Consider SMP with clock **f = 1.6** GHz and next parameters:

- Average **IPC = 1**,
- s_W **= 10%** of all operations are Write operations, s_R **= 10%** operace Read
- Each Write operations stores block of size **L = 64** B (cache line size).
- The program ran on given CPU results in read cache miss rate $M_R = 2\%$, and write cache miss rate $M_W = 3\%$, misses are distributed equally.
- Consider, that extending of the system by each other CPU results only in a constant value increase of write cache miss rate: **d=1%**.

Global bus bandwidth is **b = 8** GB/s. How many processors can be included in the system without a saturation situation?

Solution:

- Single P generates $w_W = s_W * IPC * f$ Write operations per second and $w_R = s_R * IPC * f$ Read operations per second, it is total $w = w_W + w_R$ operations per second.
- Complete required bandwidth for single P case is $r_1 = (w_W * M_W + w_R * M_R) * L$ B/s.
- Write miss rate increases if N processors are used: $M_{W,N} = M_W + d*(N-1)$.
- Aggregated bandwidth required for N processors is $r_N = N*(w_W * M_{W,N} + w_R * M_R) * L$ B/s.
- If bandwidth required for other transactions is ignored then

  $N = b/r_N$, after N evaluation and parameters substitution N=7 processors.

MESI protocol

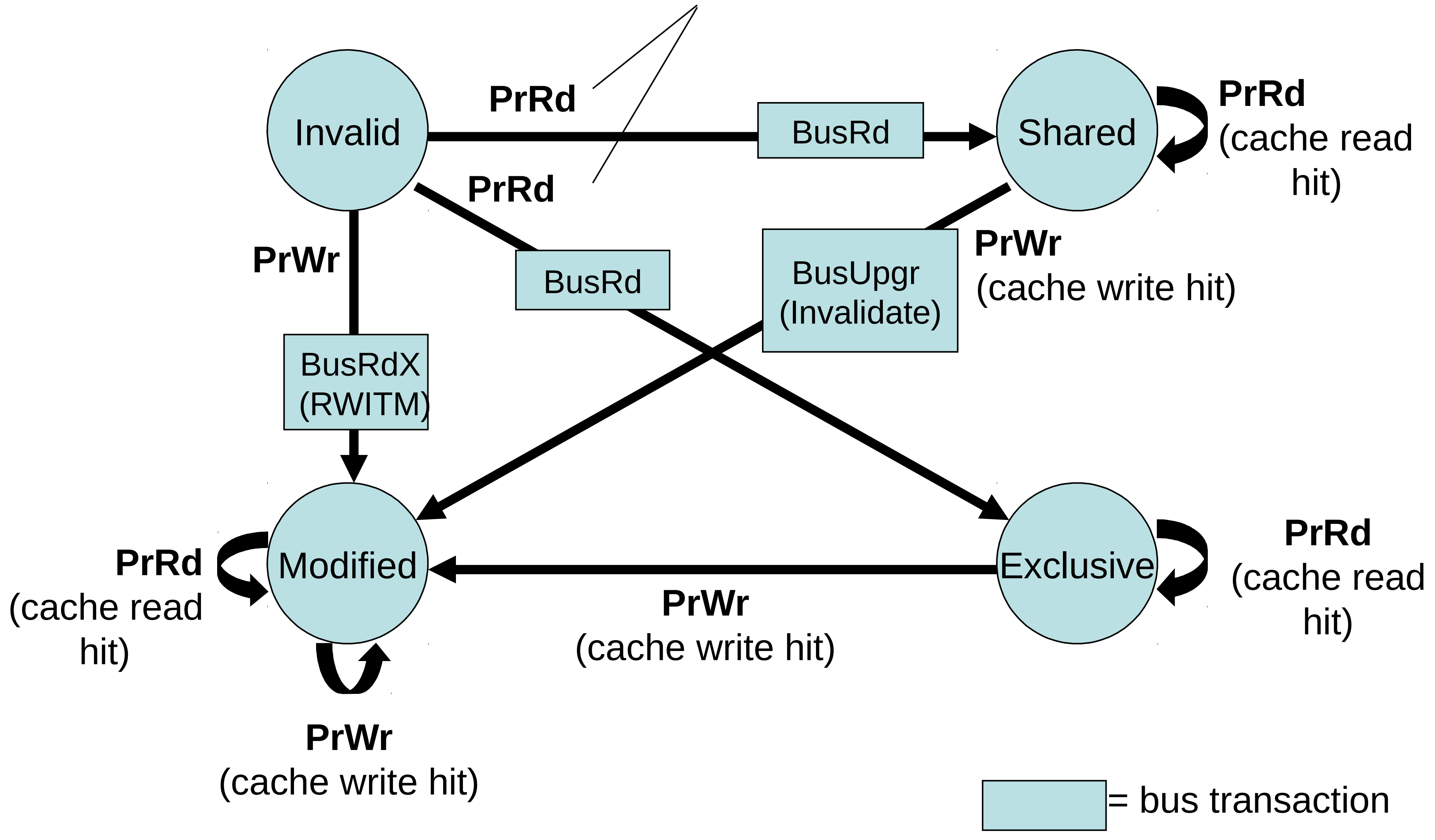
- It is the protocol which minimalizes multiprocessor system bus transactions in the case of invalidation operations.
- It is based on write-back; cache lines modification does not generate subsequent bus write transactions until there is an attempt to modify the corresponding cache-line on other CPU. Then write back occurs.
- Requires to extend cache-line flags (meta-data). Invalid and Dirty/Valid has been considered till now.
- **Each cache line occurs in one of 4 states (2 bits are enough for encoding)**
  - **M** – Modified. Cache-line content differs from data in corresponding memory cells, (it is equivalent to Dirty state),
  - **E** – Exclusive. Line content is in exactly only one processor cache and is the same as corresponding memory cells.
  - **S** – Shared. Line content is the same as corresponding memory, but content can be kept in multiple cache memories.
  - **I** – Invalid. The cache line is not used, no valid content or tag.

MESI

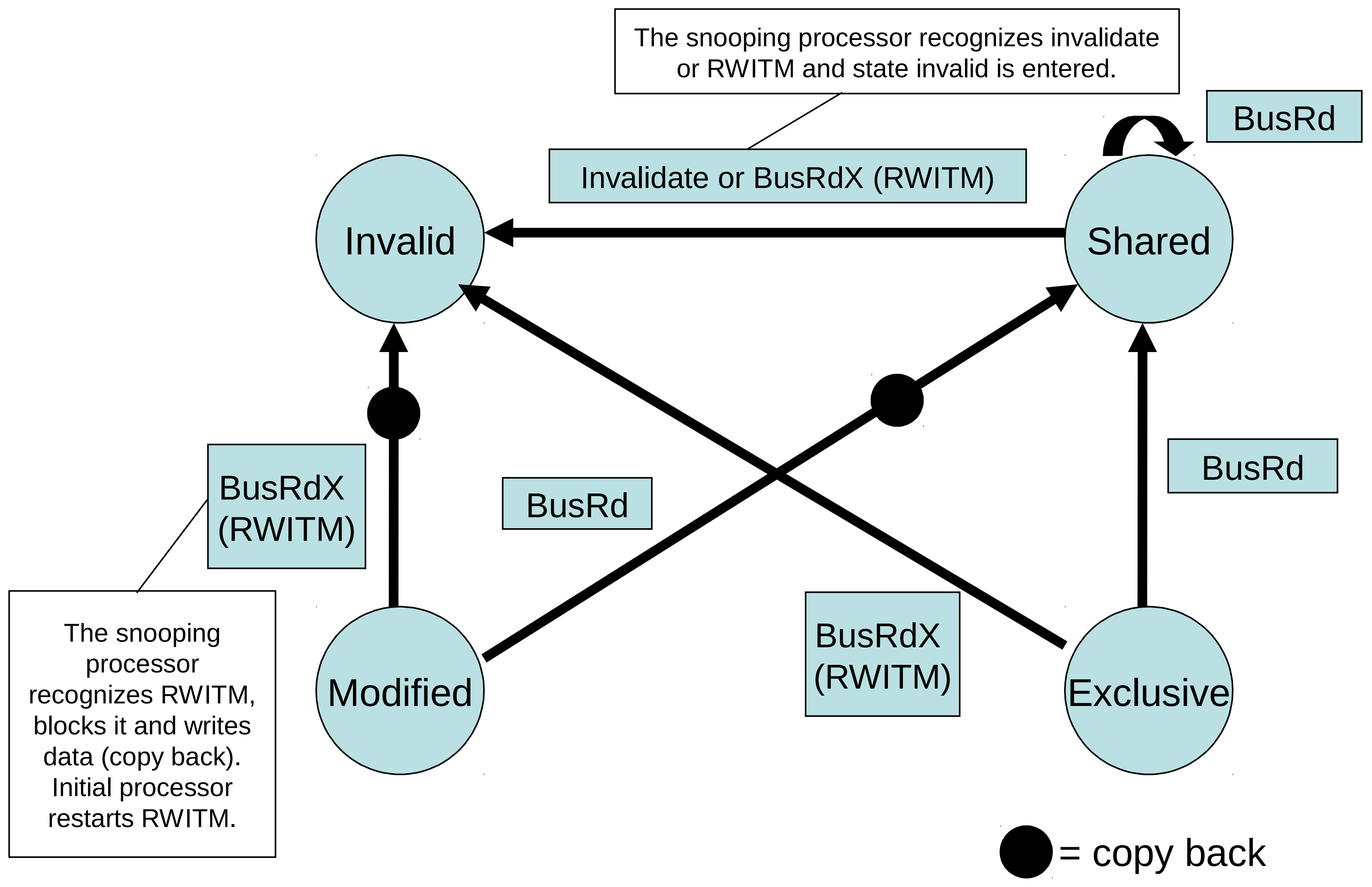
- Required actions are comprehensively summarized by the **state diagram** of transitions. It defines what happens with cache-line of the processor in a role of
  - accessing memory (read hit/miss, write hit/miss)
  - snooping processor which evaluates address/line match with accessing processor (Mem read, RWITM = Read With Intent To Modify, Invalidate).
- Operations of the local processor:
  - Read Hit (read value is available to the processor)
  - Read Miss (value is not in cache, bus transaction required)
  - Write Hit (successful write)
  - Write Miss (the corresponding line is not in the cache, bus transaction is required)

MESI – Local processor

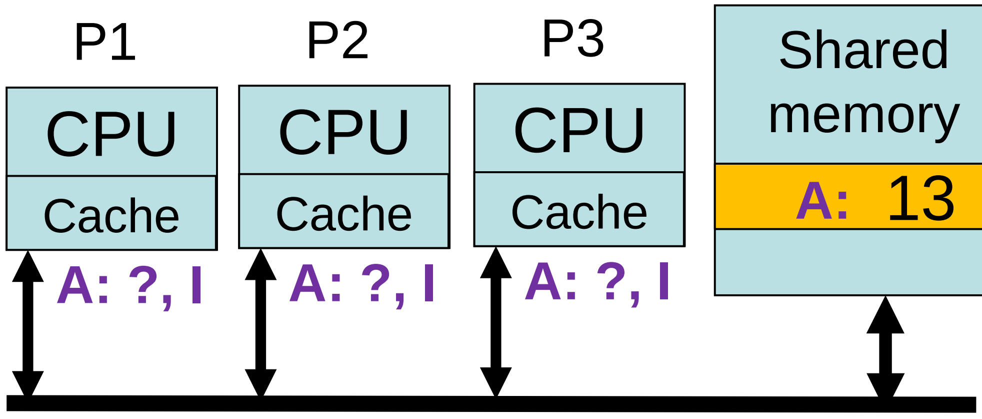
The decision, which edge is followed, is done after snoop interval expiration (snoop done) when it is sure that no copy exists in another cache. Can be implemented by additional bus signal

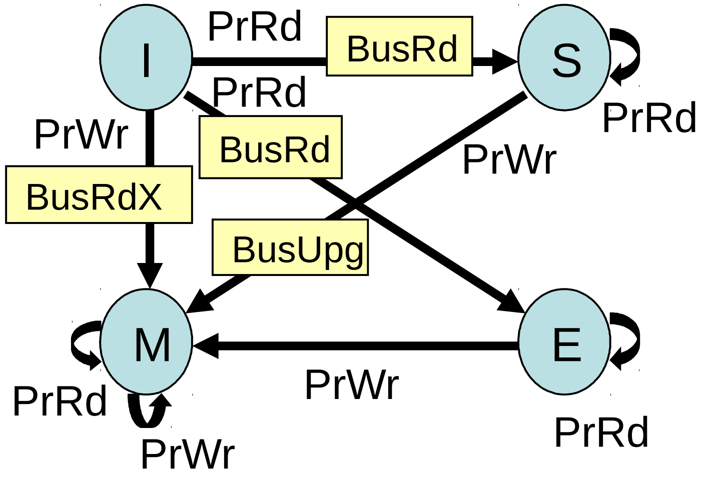


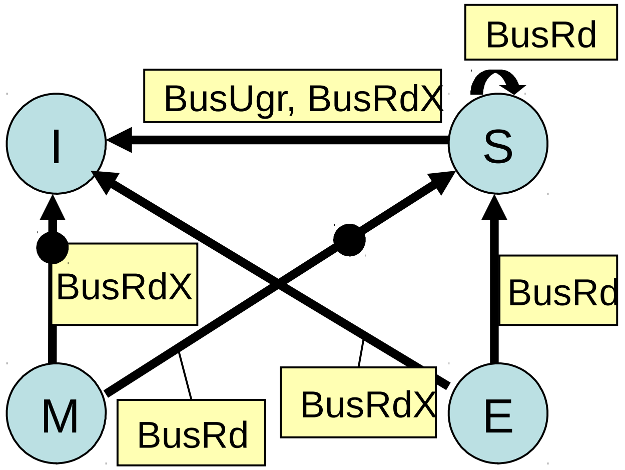
MESI – Snooping processor


The snooping processor recognizes invalidate or RWITM and state invalid is entered.
BusRd
Invalidate or BusRdX (RWITM)
Invalid
Shared
BusRdX (RWITM)
BusRd
BusRd
The snooping processor recognizes RWITM, blocks it and writes data (copy back). Initial processor restarts RWITM.
Modified
BusRdX (RWITM)
Exclusive
● = copy back

Example – MESI protocol

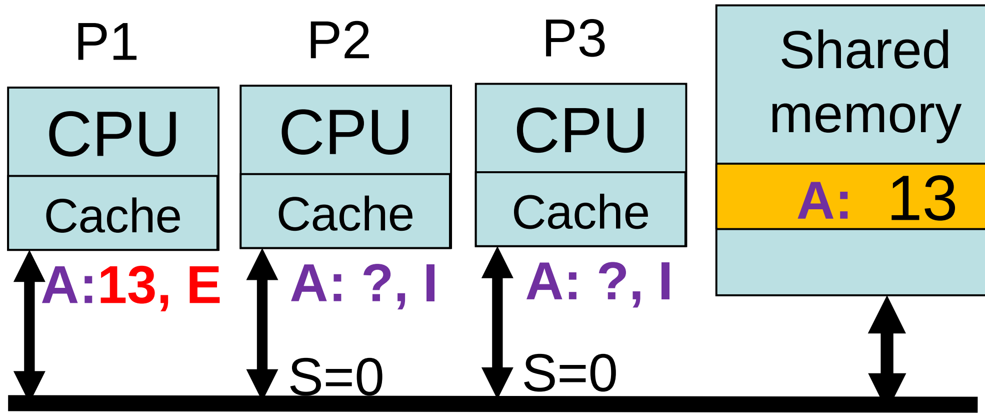


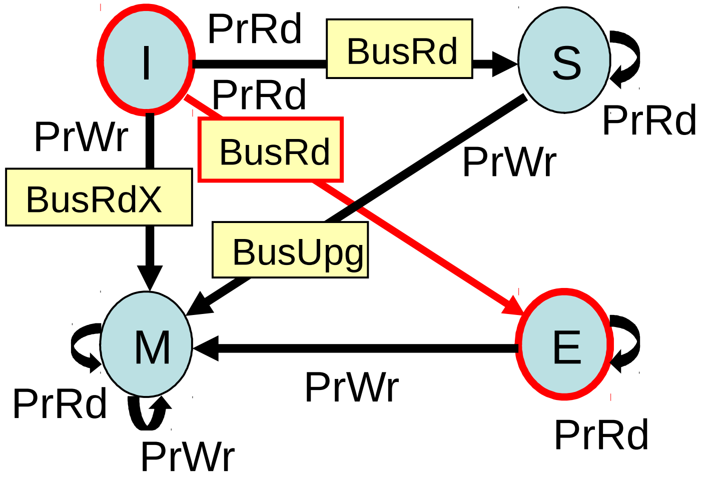




Initial state.

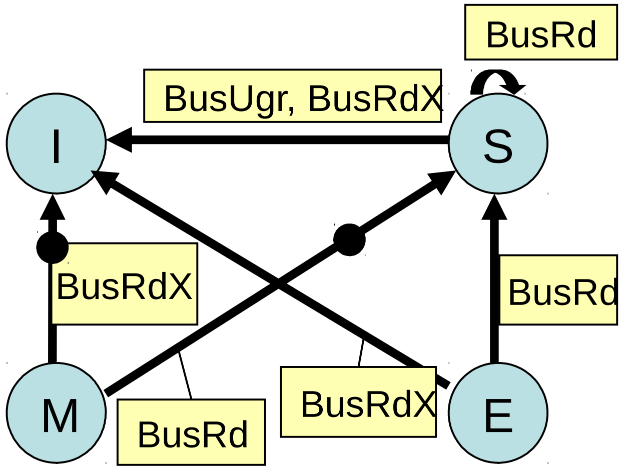
Example – MESI protocol



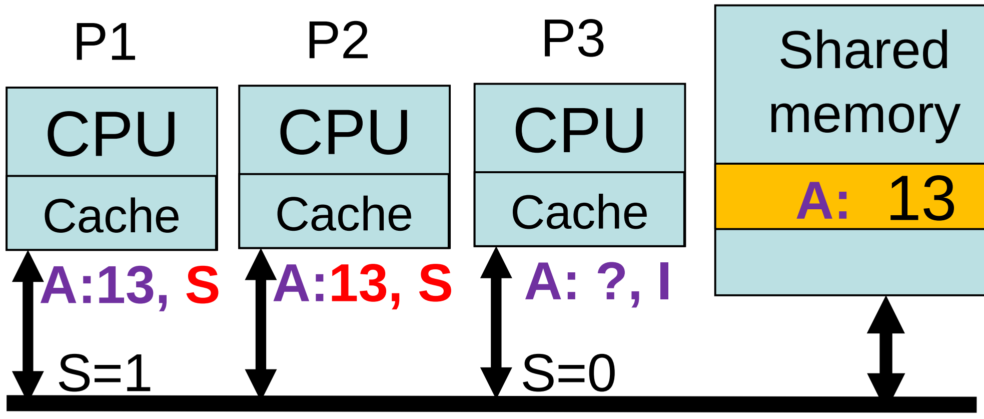


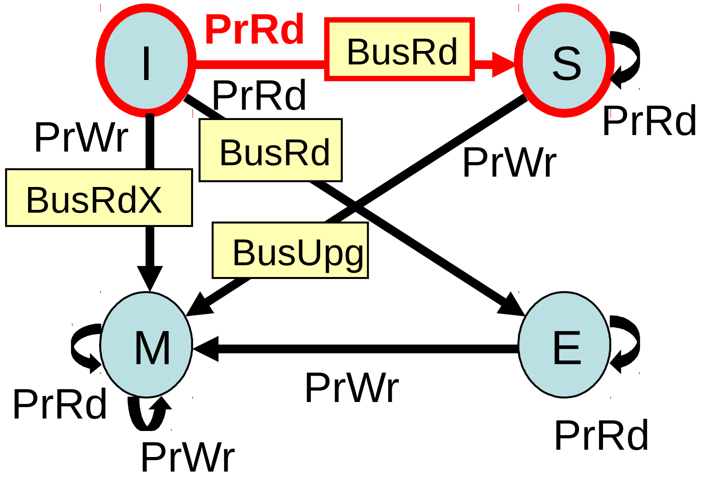
Processor P1 requests read from address A.

- P1 send PrRd(A) to its cache controller, but the line is ale Invalid, i.e., read miss occurs.
- Cache controller needs to read data from memory – issues BusRd(A) request.
- None of snooping cache controllers evaluates match, no line data copy in other caches.
- When memory delivers data which fill cache line in requesting processor P1 and line state is changed according to edge I->E.



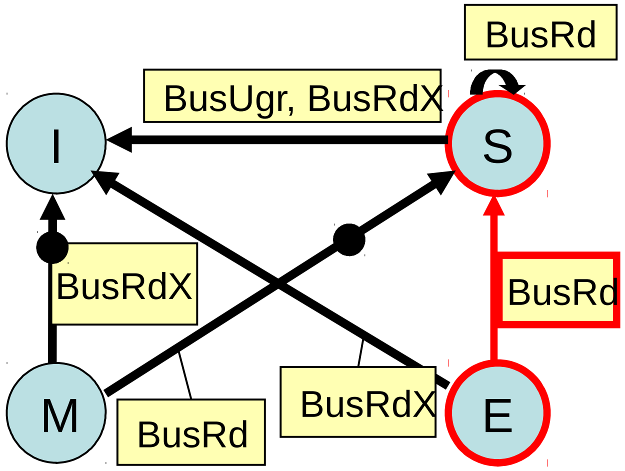
Example – MESI protocol



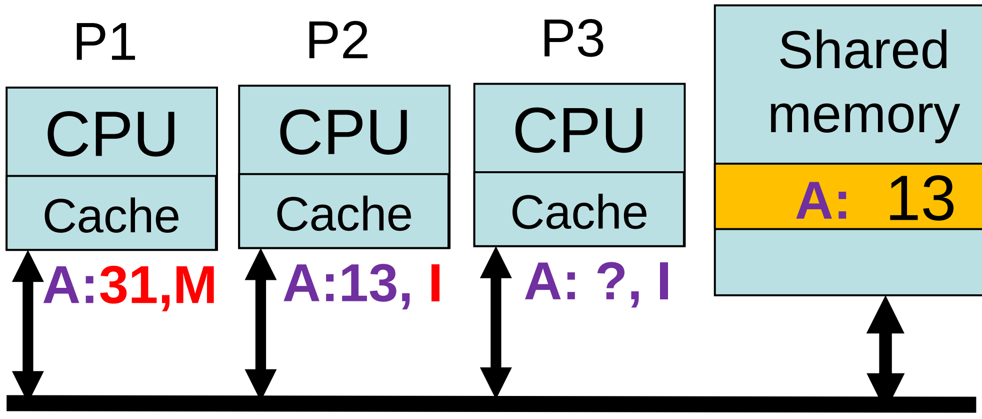


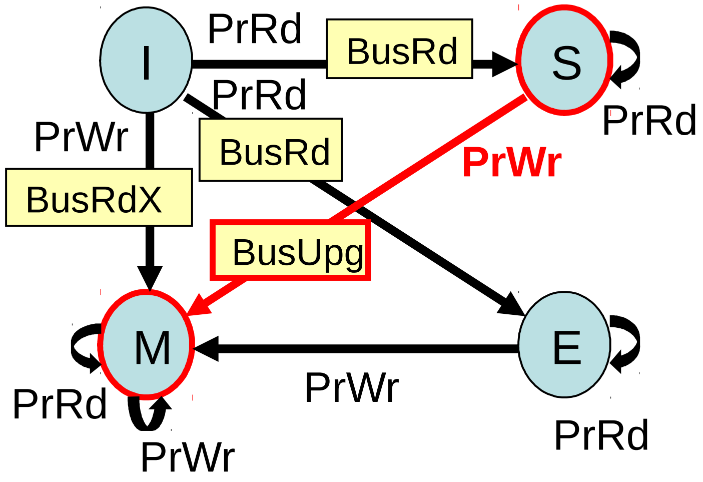
Processor P2 requests read from address A.

- P2 sends PrRd(A) to its cache, but the line is Invalid, i.e., read miss occurs.
- P2 cache controller sends BusRd(A) to the bus.
- Snooping cache controller of P1 indicates that it has data copy in its cache (asserts signal S=1) aborts read request and delivers data from the cache.
- Both processors/cache controllers advance to state S. (P1: E->S; P2: I->S)



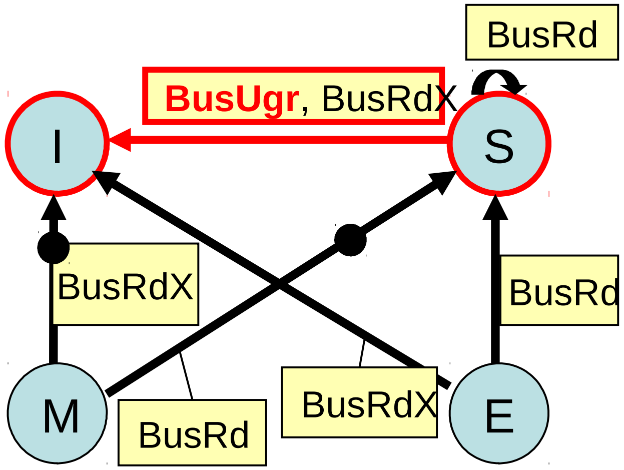
Example – MESI protocol



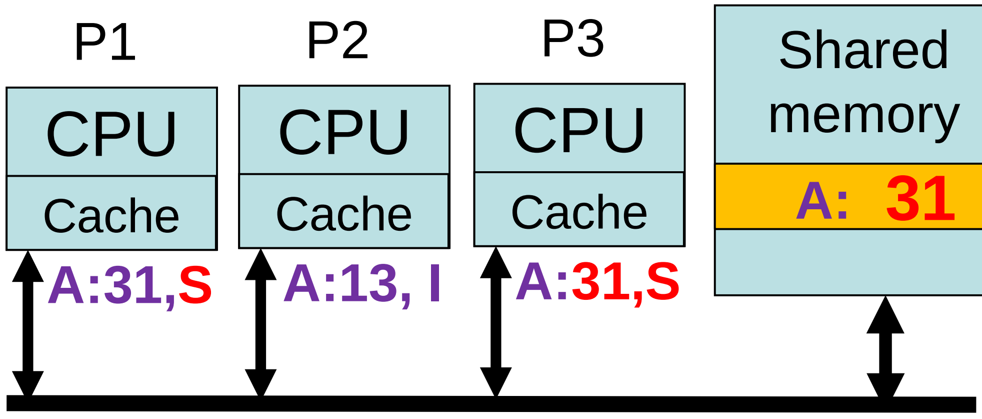


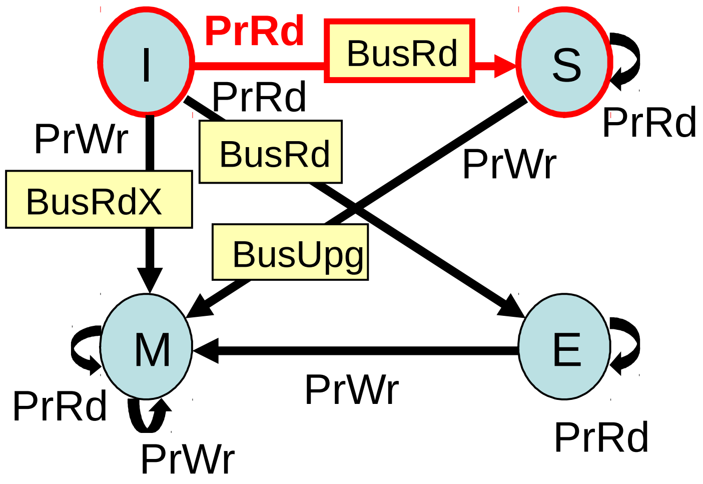
Processor P1 writes a new value to address A. For example 31.

- P1 send PrWr to cache. But block is in the S state.
- Cache controller sends BusUpgrade(A).
- All snooping cache controllers (in the example only one) recognize match with BusUpgrade(A) and activate edge S->I.
- Memory is not updated.
- P1 cache-line state changes S->M and write is finished.



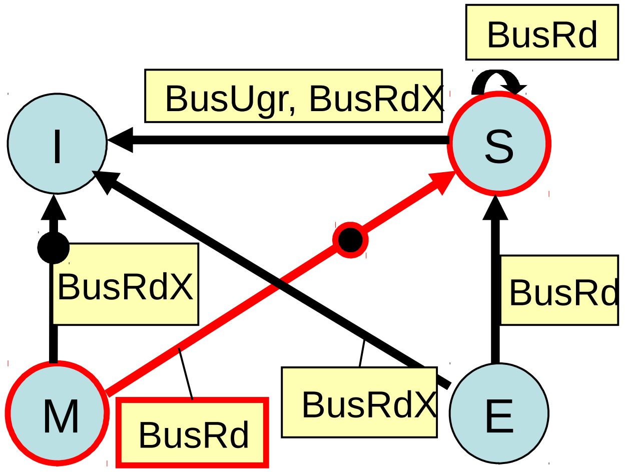
Example – MESI protocol



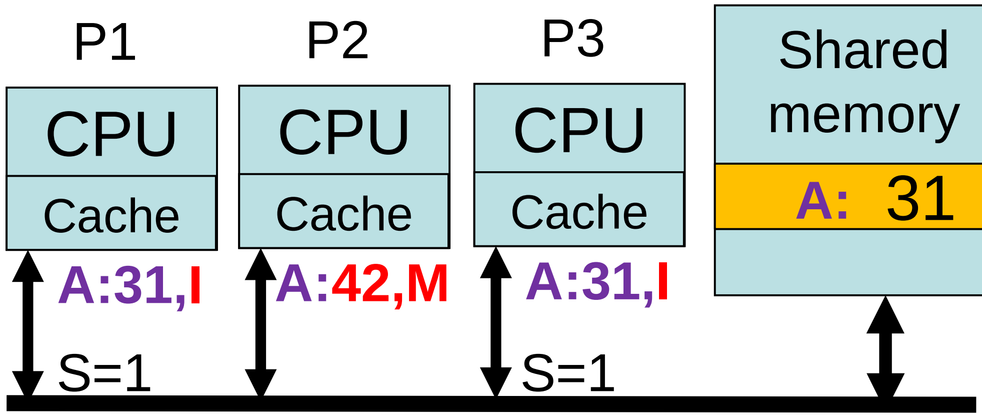


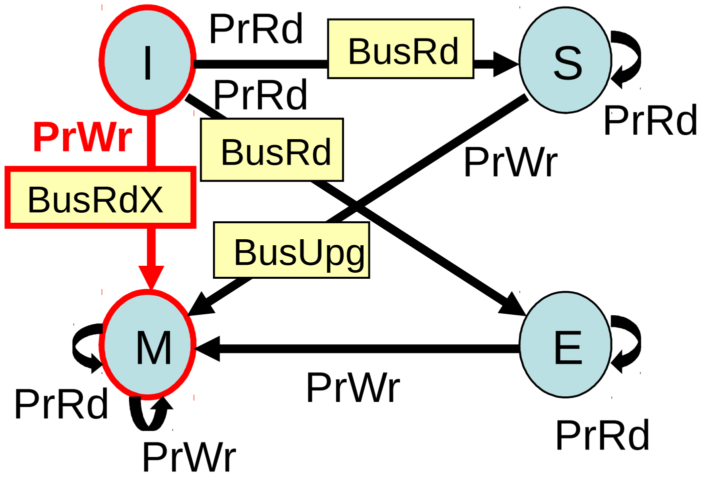
Processor P3 reads data from address A. Which value does it read?

- P3 sends PrRd(A) to its cache, but the line is Invalid. Cache controller responds by sending BusRd(A).
- Snooping cache controller of P1 indicates the match, asserts signal S=1 and delivers data from its cache to the bus and that way to P3.
- Both change state to S. (P1: M->S; P3: I->S)
- There is copy-back at M->S edge. Memory is updated.



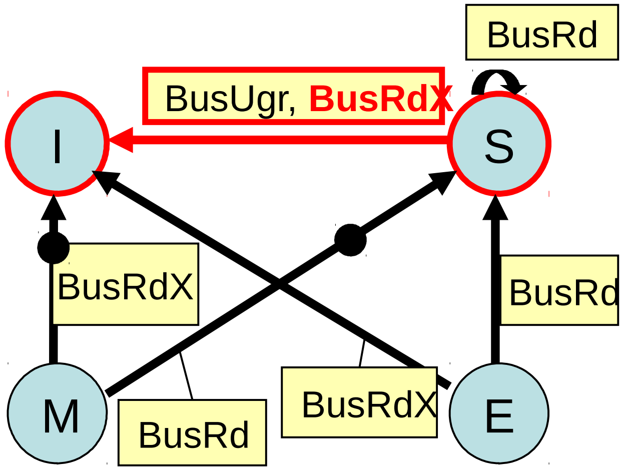
Example – MESI protocol





Processor P2 writes data to address A. Value 42 for example.

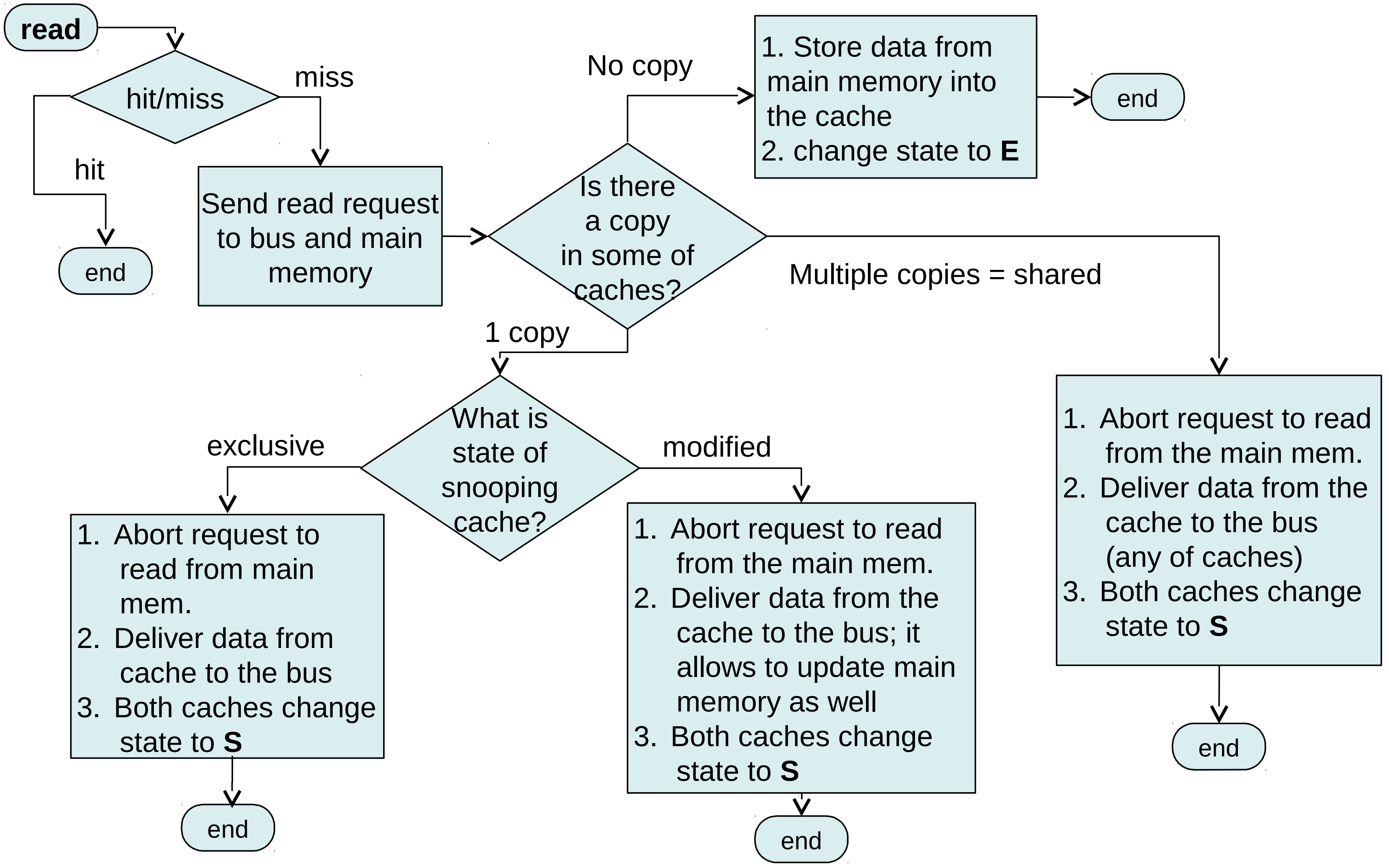
- P2 send PrWr(A) to its cache, but the line is Invalid. Cache controller issues BusRdX(A).
- Snooping controllers indicate that they have data copy. Abort memory read request, and some of the controllers deliver data.
- Both snooping controllers follow edge do S->I.
- Data requesting P2 follows edge I->M.
- Memory is not updated.



Remarks to simplify implementation

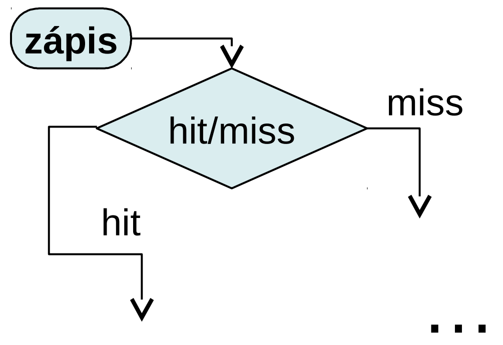
- If the cache line is in E (Exclusive) state, then there is no need to send any bus transaction (for both read or write)
- State changes of cache-line occur during Read and or Write events (memory access)
- The event is caused by
  - Cache controller local/connected processor activity/code execution (cache access), or
  - As a result of successful bus snooping (address matching) of other processor initiated bus activity.
- Changes of cache-line state occur only in the cache of the corresponding transaction and cache-line address (index+tag+address) match.

Summary of previous slides – Read



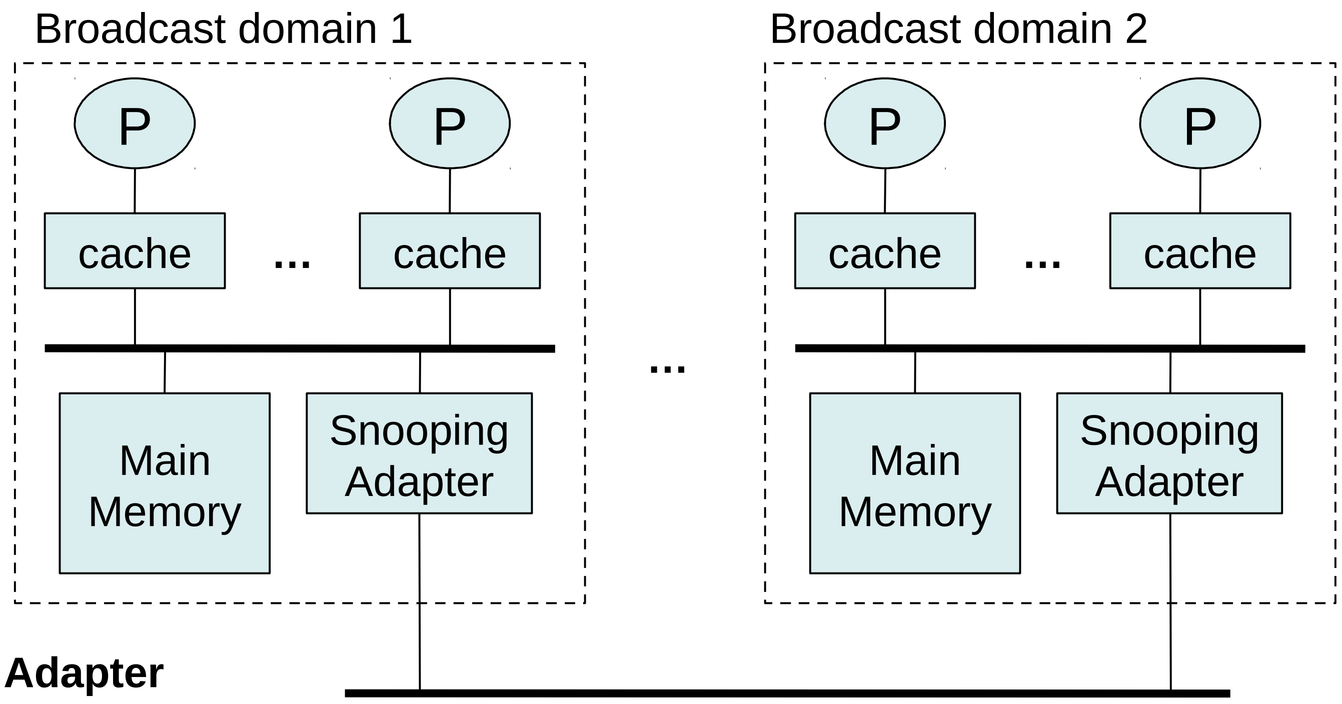
read
hit/miss
miss
hit
end
Send read request to bus and main memory
Is there a copy in some of caches?
No copy
1. Store data from main memory into the cache
2. change state to **E**
end
Multiple copies = shared
1 copy
What is state of snooping cache?
exclusive
modified
1. Abort request to read from main mem.
2. Deliver data from cache to the bus
3. Both caches change state to **S**
end
1. Abort request to read from the main mem.
2. Deliver data from the cache to the bus; it allows to update main memory as well
3. Both caches change state to **S**
end
1. Abort request to read from the main mem.
2. Deliver data from the cache to the bus (any of caches)
3. Both caches change state to **S**
end

Summary of previous slides – Write



Broadcast extension/scaling for more processors

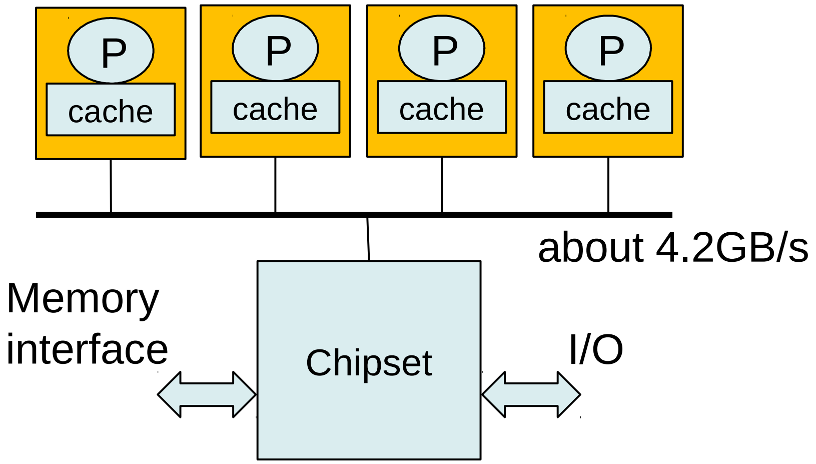
- Systems based on Broadcast (snooping) can be practically scaled to about 8-10 nodes where each node can be multicore processor today
- Option for more processors is hierarchical Hierarchical Snooping.
- But shared buses are generally replaced by point-to-point interconnection (on next slides) → snooping is not so much important today.



Snooping Adapter Separates buses

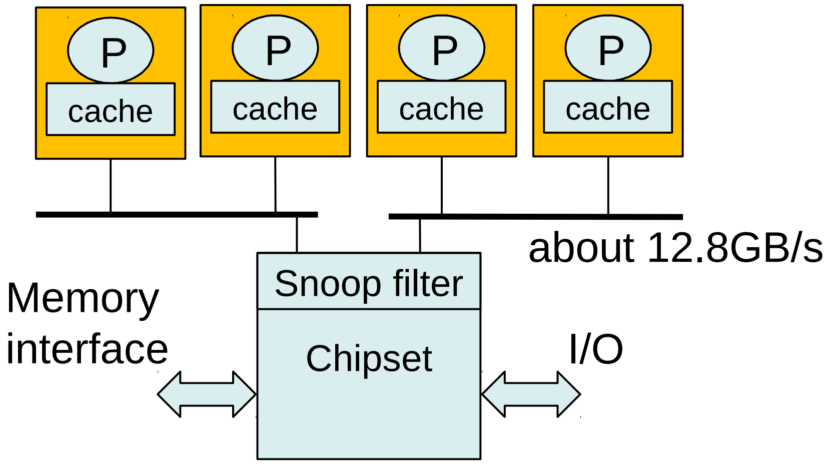
Development inside of processor as a chips/packages

Year 2004



- Classical bus limitation … Problem: increase bus frequency. **Important**: Bus ensures serialization of all requests (access arbitration used) – any two processors cannot modify the same cache-line in the same instant of time

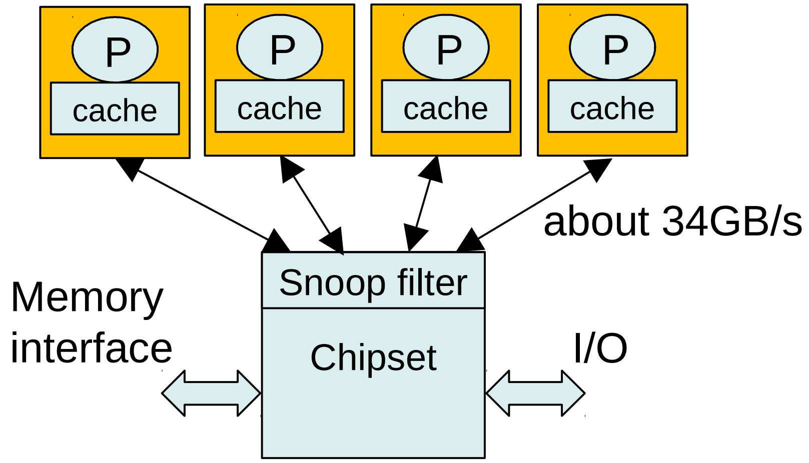
Year 2005



- Two independent buses – DIB (dual independent buses). Snoop principle has to be preserved. If all traffic is propagated, throughput is degraded. That is why snoop filter filters requests and stores snoop information and does not propagate irrelevant transactions

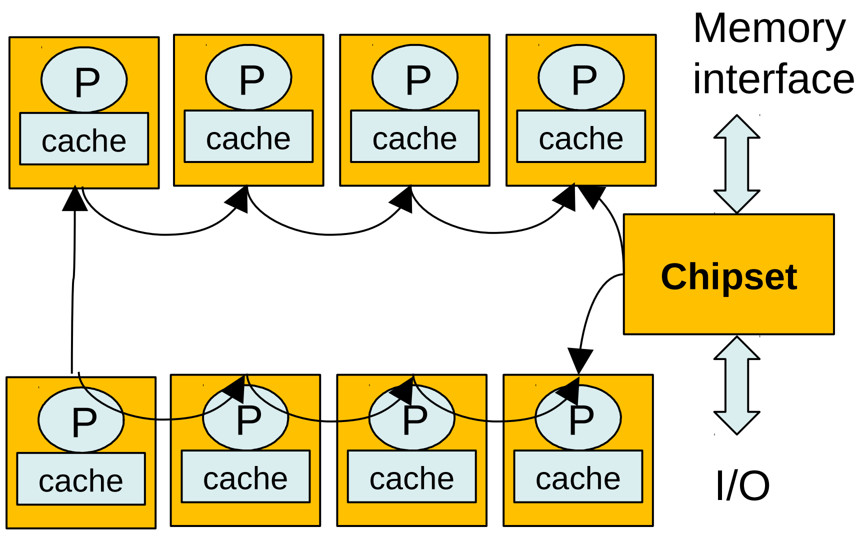
Development inside of processor as a chips/packages

Year 2007



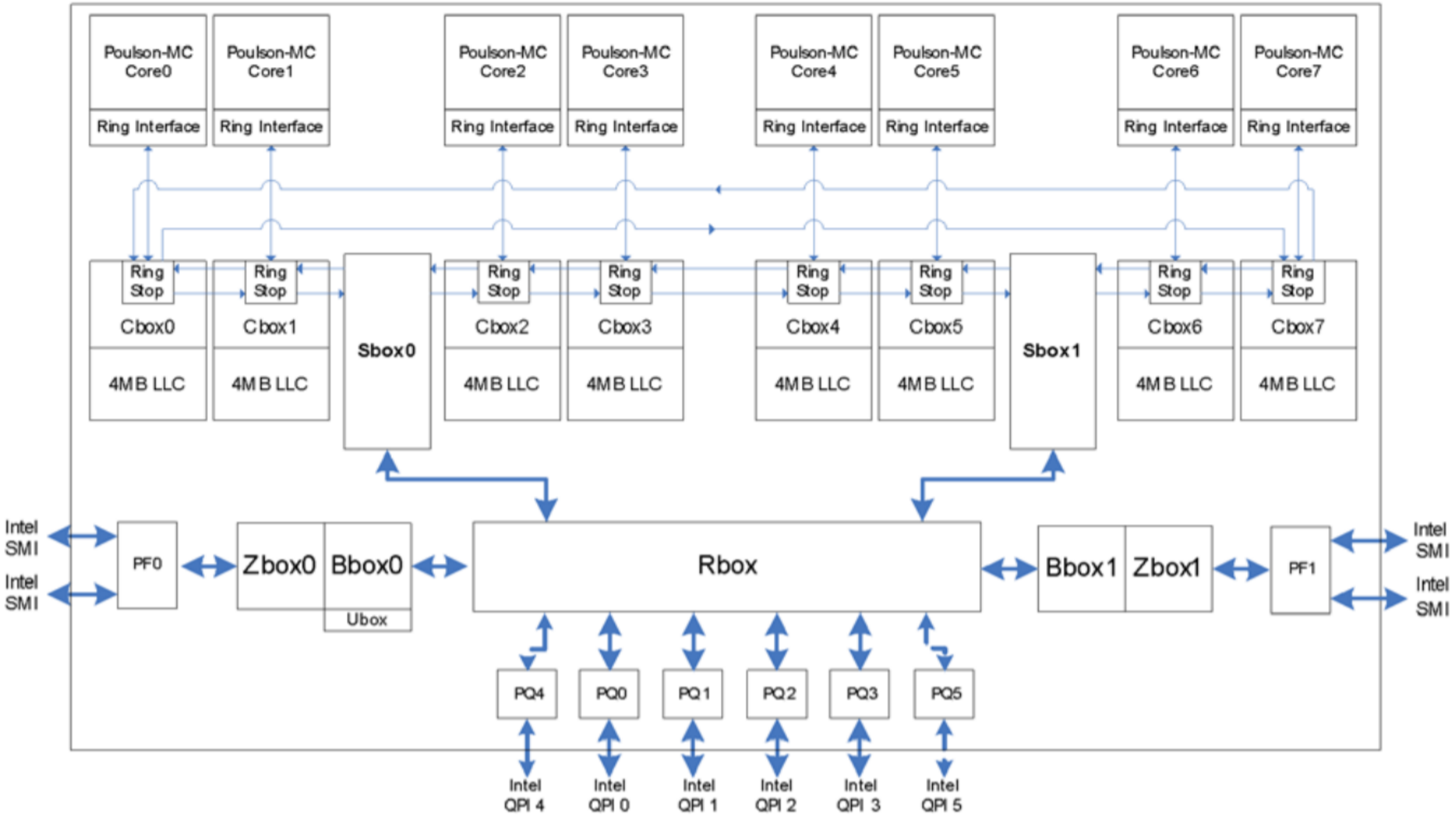
- Introduction high-speed point-to-point interconnects. But snoop filter becomes the bottleneck of the system. It is too centralized.

2009 till today



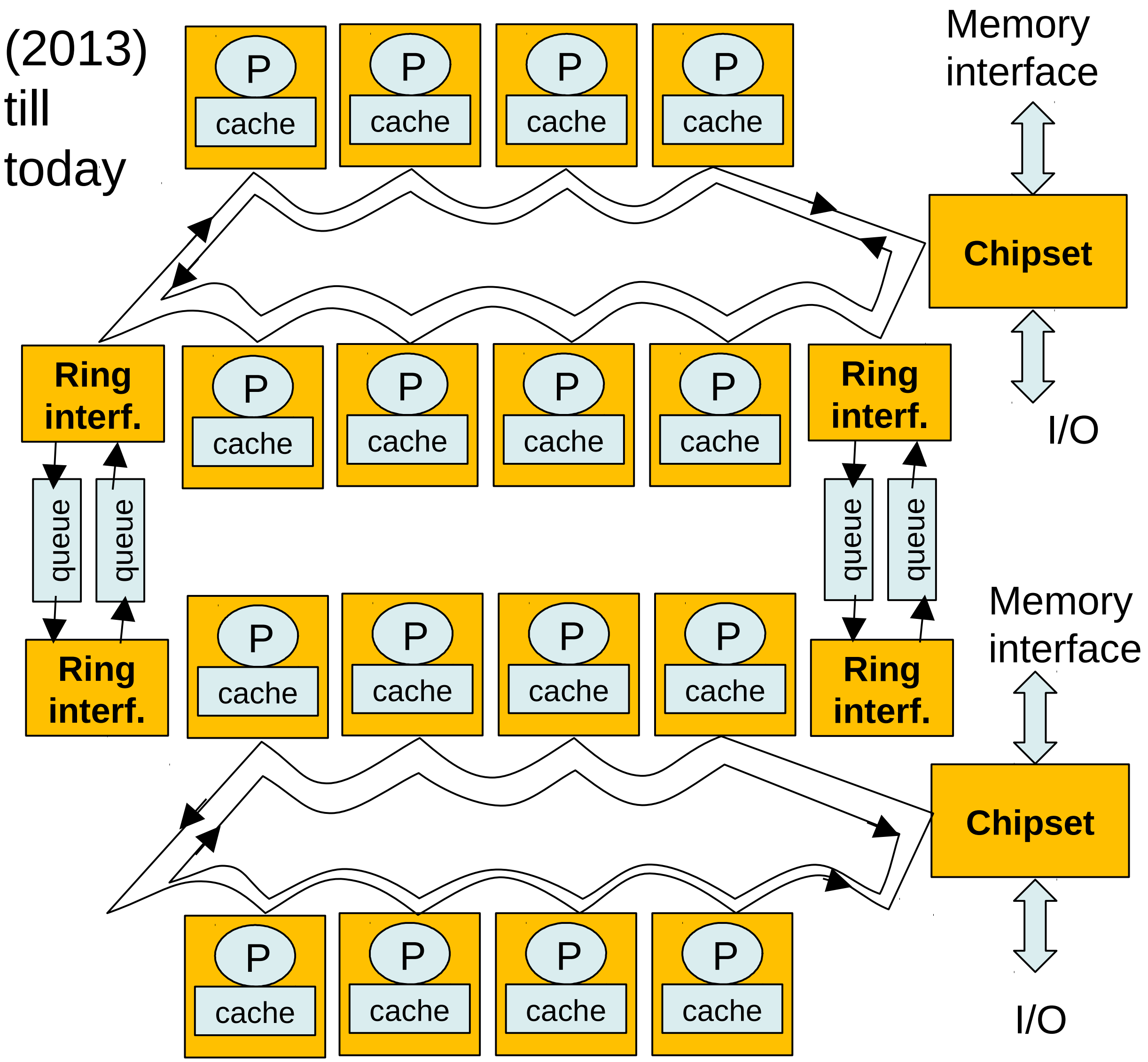
- Ring. Single direction ring (in the picture) – all messages are delivered between nodes in the same direction and node order is fixed. Two single direction links placed in opposite direction form bidirectional ring often used today. Ring routers can be simple – same as on roundabout which is left by a car when you need/reach the target node direction.

Example: Intel Itanium Processor 9500



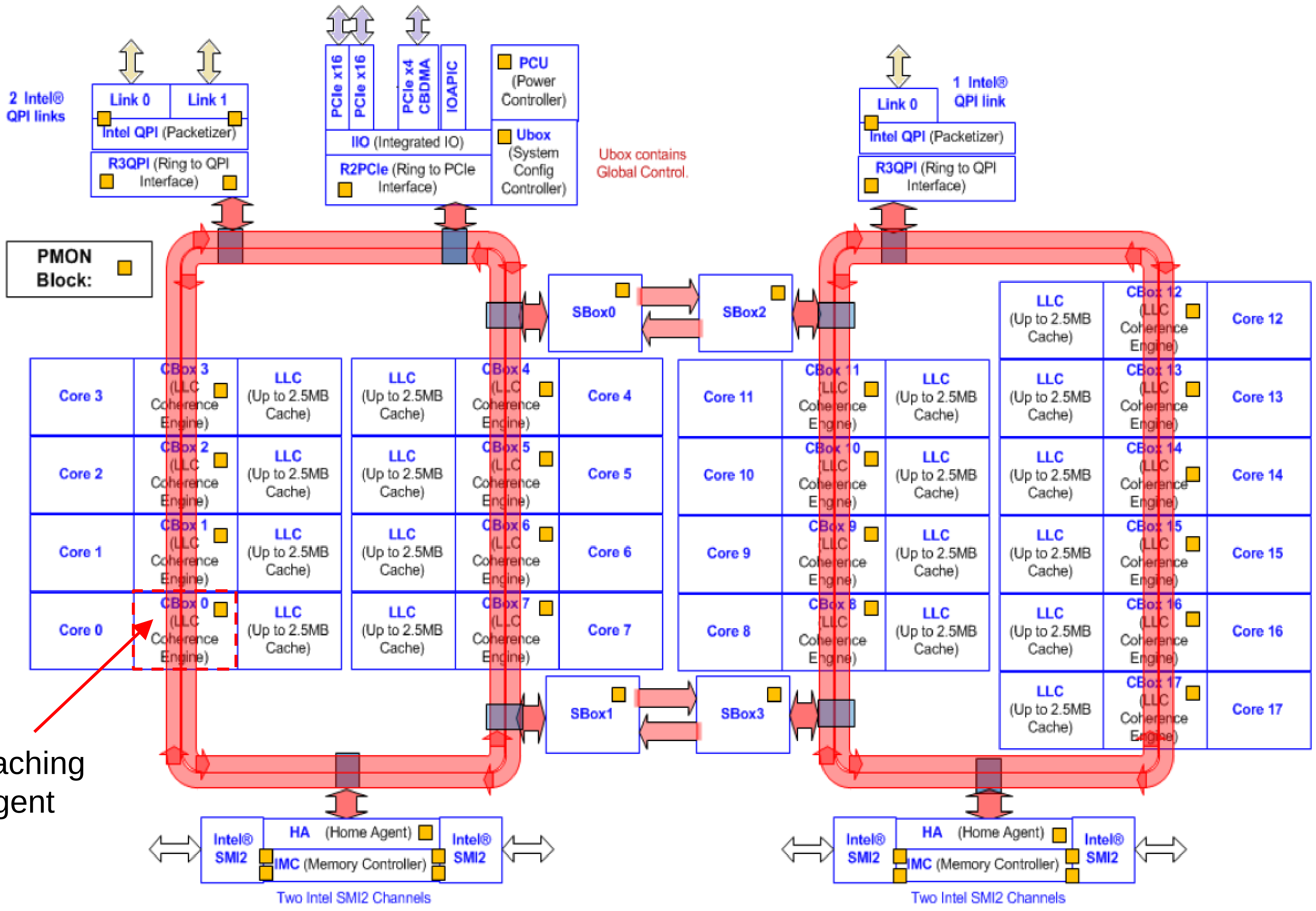
Development inside of processor as a chips/packages

(2013) till today



- Interconnected rings. Even single package multi-processor becomes NUMA system.
- The ring offers fast point-to-point interconnection and removes the complexity of packed oriented interconnections with general topology. Routing in each node is simple – includes only single input and output port for each ring. Nodes can insert and or remove message thanks to distributed arbitration. But the ring does not ensure global ordering of events (which is natural on the bus) – order depends on observer position… => Greedy snooping (IBM Power4/5), everything to everybody **or selective with use of Directories**.

Practical example: Broadwell-EP (Intel Xeon)



caching agent

Practical example: Broadwell-EP (Intel Xeon)

Broadwell-EP uses next means to speedup „snooping“ and lowering communication load:

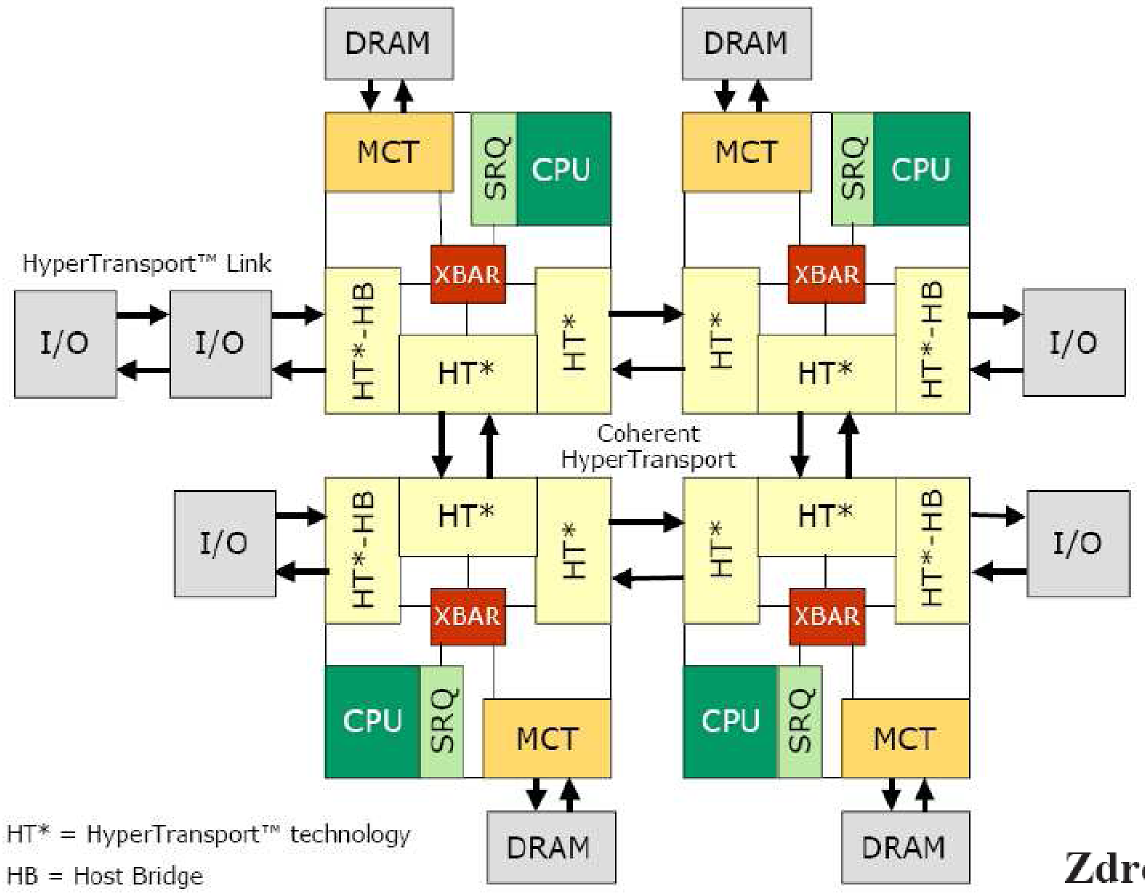
- **Directory Cache** – it is 14KB cache each HA (home agent). It stores 8-bit vector which informs which CA (caching agent) can deliver a copy of cache-line. Directory cache is integrated into the chip. Directory cache hit means whom we should ask to provide data for the given address.
- **Directory** – the directory is placed in the memory controller and requires only 2 bits (**directory bits**) for each block (cache line) – states Local/Invalid, SnoopAll, Shared. The directory is consulted only in case of Directory cache miss.

Remarks:

- **Directory cache** extension to DAS protocol. Speedups access to cache lines, which are (re)sent from the cache of other nodes.
- **Directory assisted snoop broadcast protocol** (DAS) is the extension of commonly used **MESIF** protocol (F state means Forwarding – i.e., who is responsible for forwarding). DAS uses the directory to store auxiliary information. It reduces the number of queries to HA that way.

How to snoop without a shared bus?

- Example: AMD Quad – Coherent HyperTransport
- Instead of a bus HT (AMD) or QPI (Intel) is used
- Example of interconnection of **four multi-core processors**:

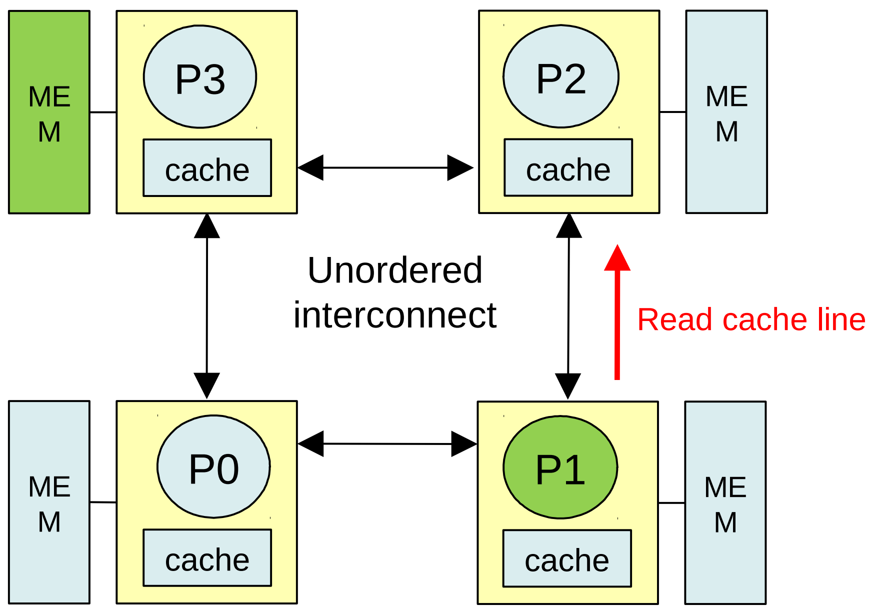


Zdroj: AMD

How to snoop without a shared bus? Broadcast protocol.

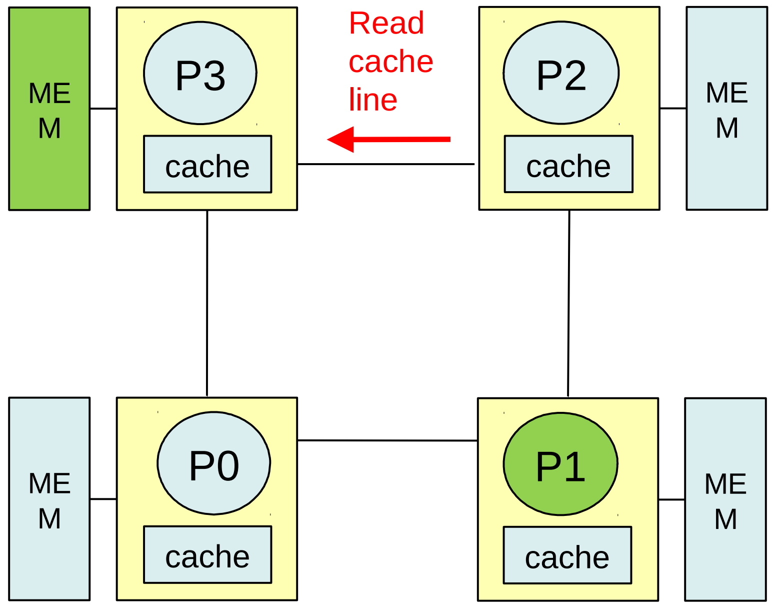
Example: CPU1 reads memory location which is homed in CPU3 (it is in memory controlled by P3 memory controller)

Step 1



- Reaction on last-level cache miss: query sent to the home node
- This memory controller decides order (of processing) of all queries to the same cache line

Step 2

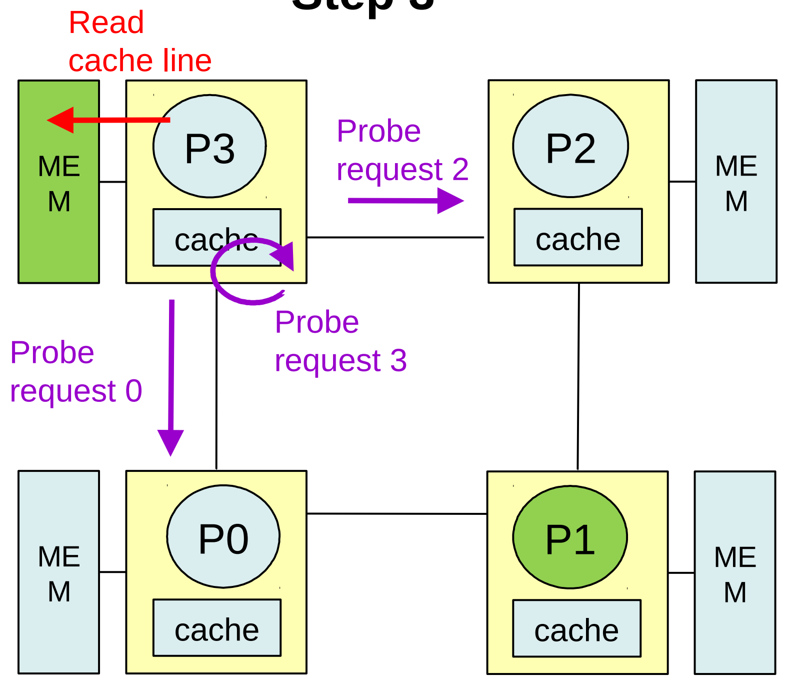


- There is no direct connection between P1 and P3. Querry is set over P2.

How to snoop without a shared bus? Broadcast protocol.

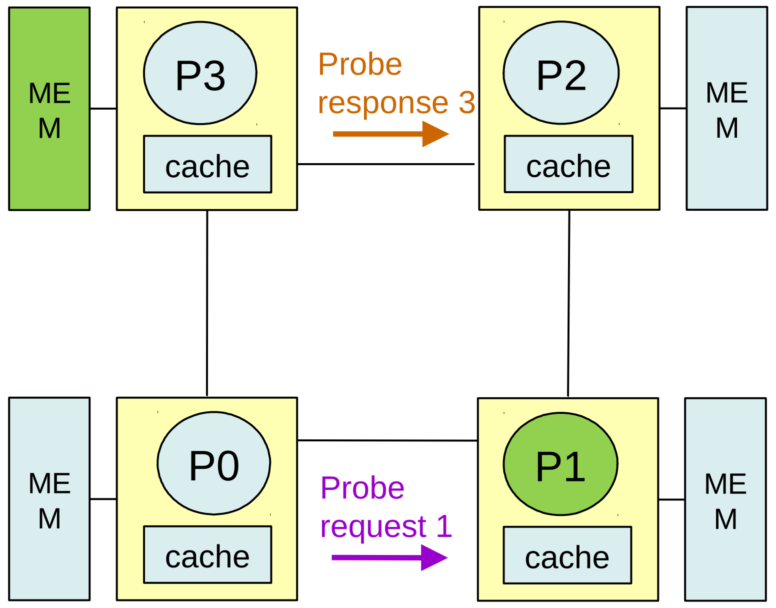
Example: CPU1 reads memory location which is homed in CPU3 (it is in memory controlled by P3 memory controller)

Step 3



- When query reaches home node and starts to be served, cache probes requests are sent to all processors and access to RAM is initiated in parallel

Step 4

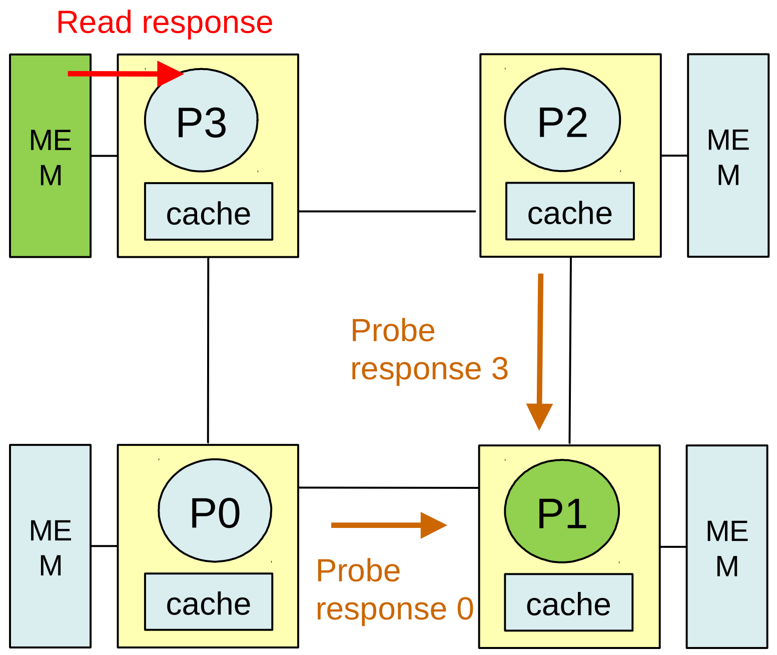


- All processors send probe responses (immediately as they obtain them) to querying processor – P1.
- Memory controller also sends message, when data from RAM are available – step 5

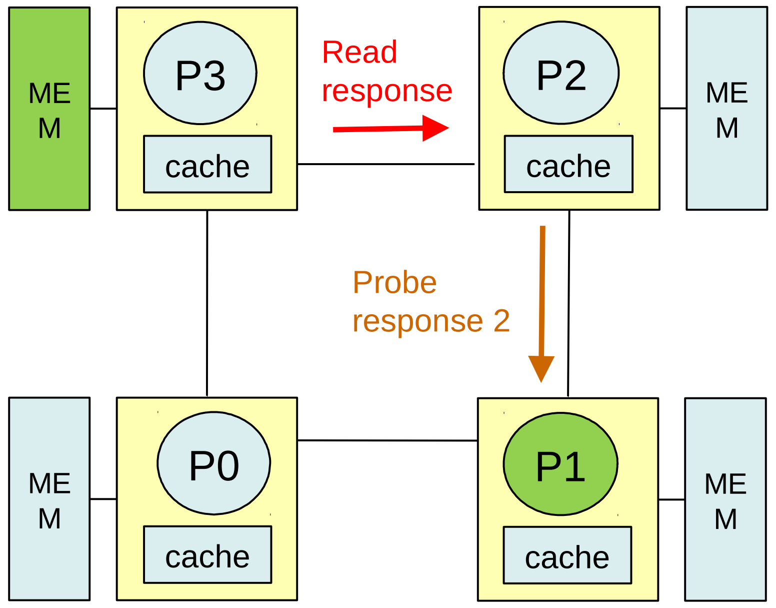
How to snoop without a shared bus? Broadcast protocol.

Example: CPU1 reads memory location which is homed in CPU3 (it is in memory controlled by P3 memory controller)

Step 5



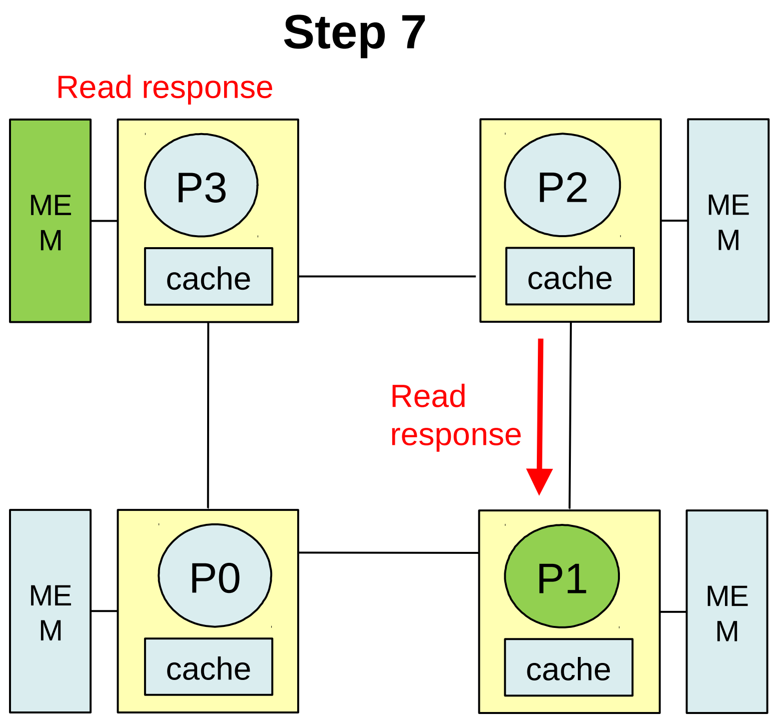
Step 6

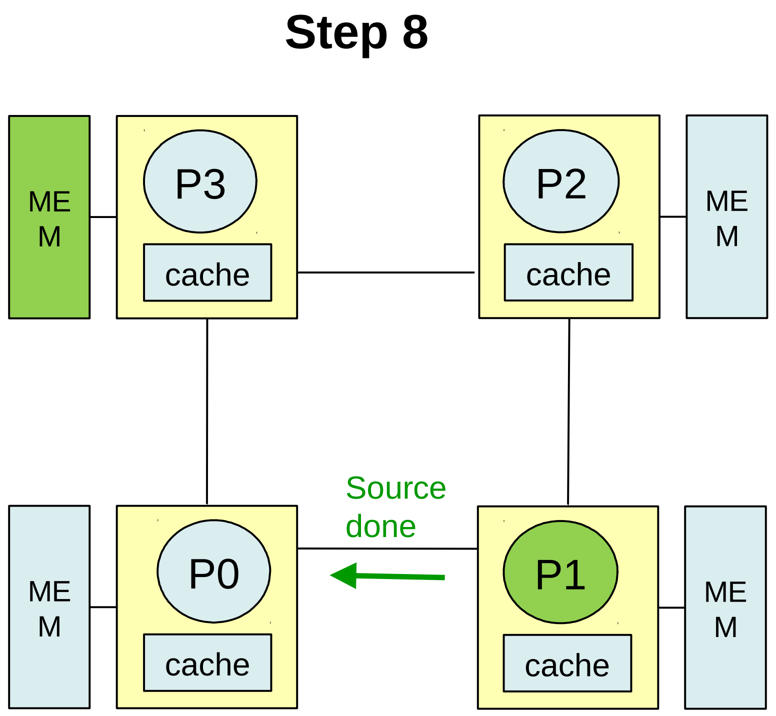


- Querying processor P1, collect responses from all processors in time as they arrive

How to snoop without a shared bus? Broadcast protocol.

Example: CPU1 reads memory location which is homed in CPU3 (it is in memory controlled by P3 memory controller)

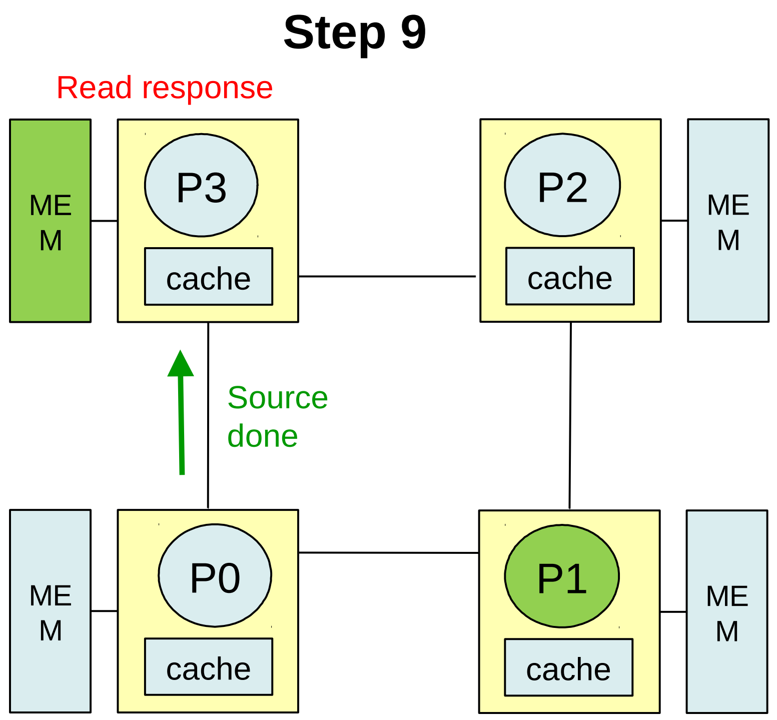




- As requesting processor receives all responses, it sends the message to home node which informs that cache line request is handled then home node (memory controller) can service next request to the same cache-line.

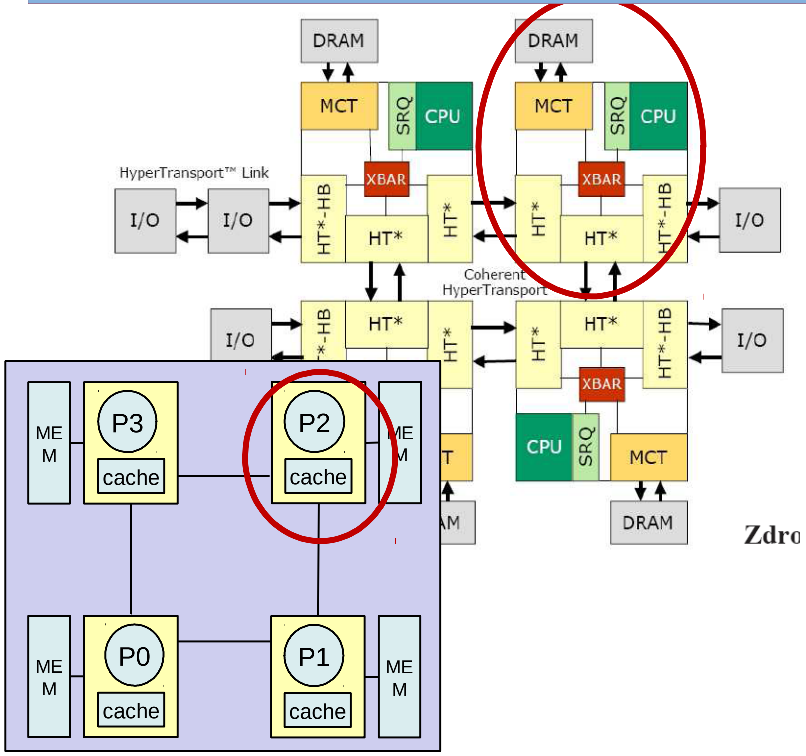
How to snoop without a shared bus? Broadcast protocol.

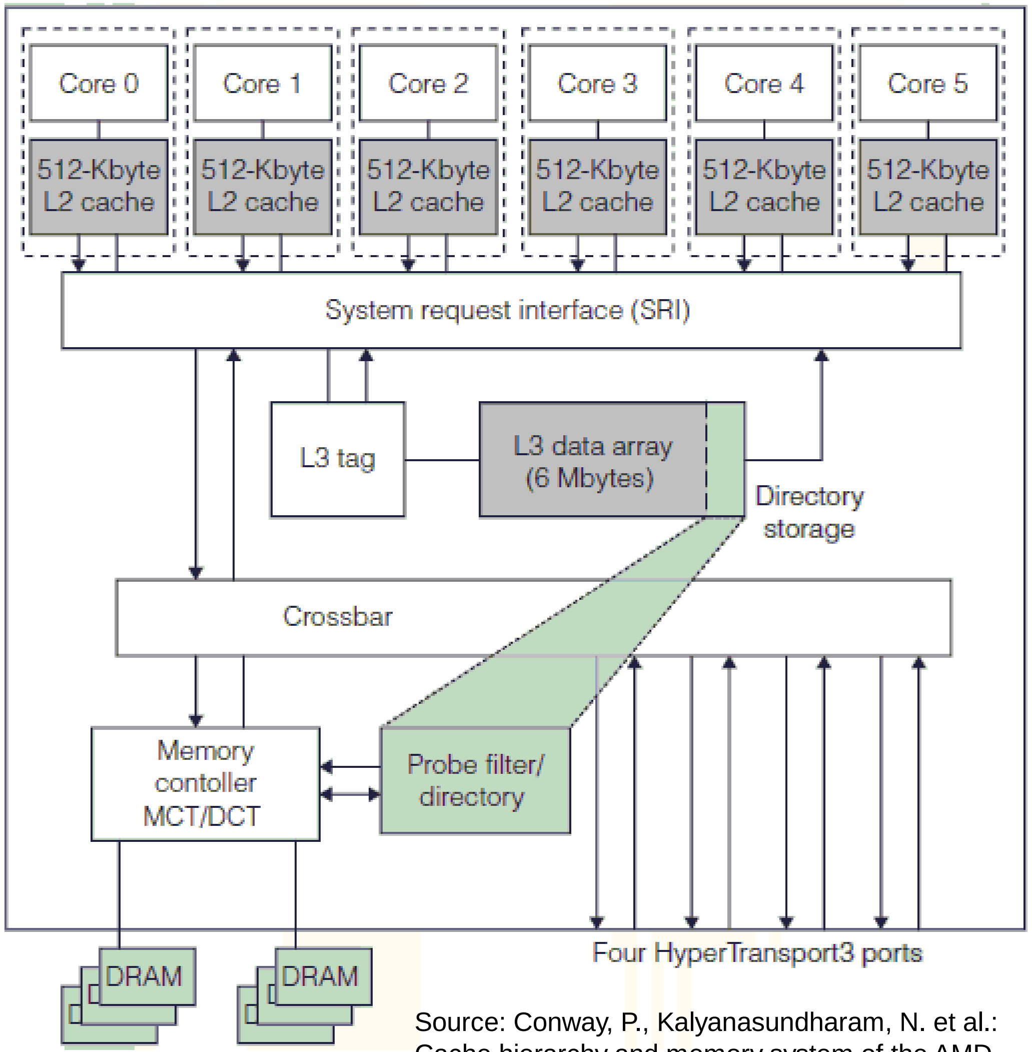
Example: CPU1 reads memory location which is homed in CPU3 (it is in memory controlled by P3 memory controller)



- Notice complexity of communication for single read?
- Next generation of AMD processors included **HT assist** (HyperTransport Assist directory protocol) – about 2010 year.
- HT assist uses **directory** (in the home node) which maintains information which cache-lines are cached in another CPUs
- It allows reducing inter-processor communication significantly. Instead of the previous broadcast to all processors, next 3 cases can appear:
- no probe – data only in RAM, nobody else
- directed probe (query to single CPU only) – it is not in RAM only one has the line
- broadcast probe – fallback case

This is the case of AMD Quad – Coherent HyperTransport





- Probe Filter (HT Assist) – uses part of L3 cache as directory cache in which it monitors cached lines. Instead of generating many requests (cache probes), processor searches this part of L3 cache.

Source: Conway, P., Kalyanasundharam, N. et al.: Cache hierarchy and memory system of the AMD Opteron processor, IEE Micro, March/April 2010.

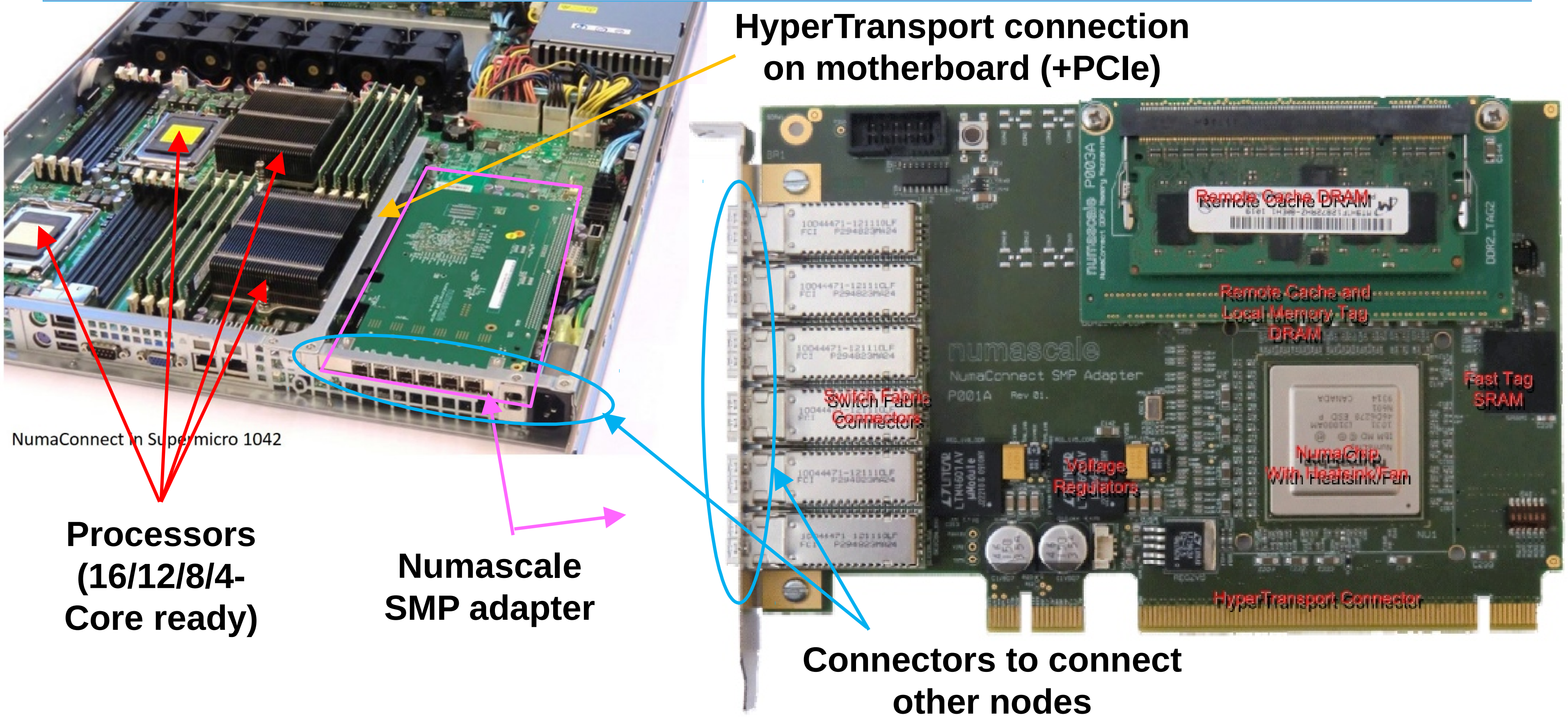
When the bus is not enough and broadcast to all is limiting

- Sending information to everyone else (or listening to all) is not a scalable solution ...
- We noticed that removing buses (where monitoring/snooping is not a problem) and by the change to the point-to-point interconnection between CPU cores (nodes) and the increase in the number of cores, the solution how to “snoop” but not burden the system with excessive communication becomes crucial.
- **Directories**

(More projects aiming to solve problems related to shared memory started by end of 80. and start of 90. years. One of then was “SCI” (Scalable Coherent Interface) - HP, Apple, Data General, Dolphin, US Navy,.. The next one was “DASH” – Stanford. Both use similar technologies – directory-based cache coherence architecture.

- This approach is suitable for interconnection of hundreds and or thousands of processors.
- Directories use is not a new idea. The new is only that their use has spread even to today's common desktop CPUs.

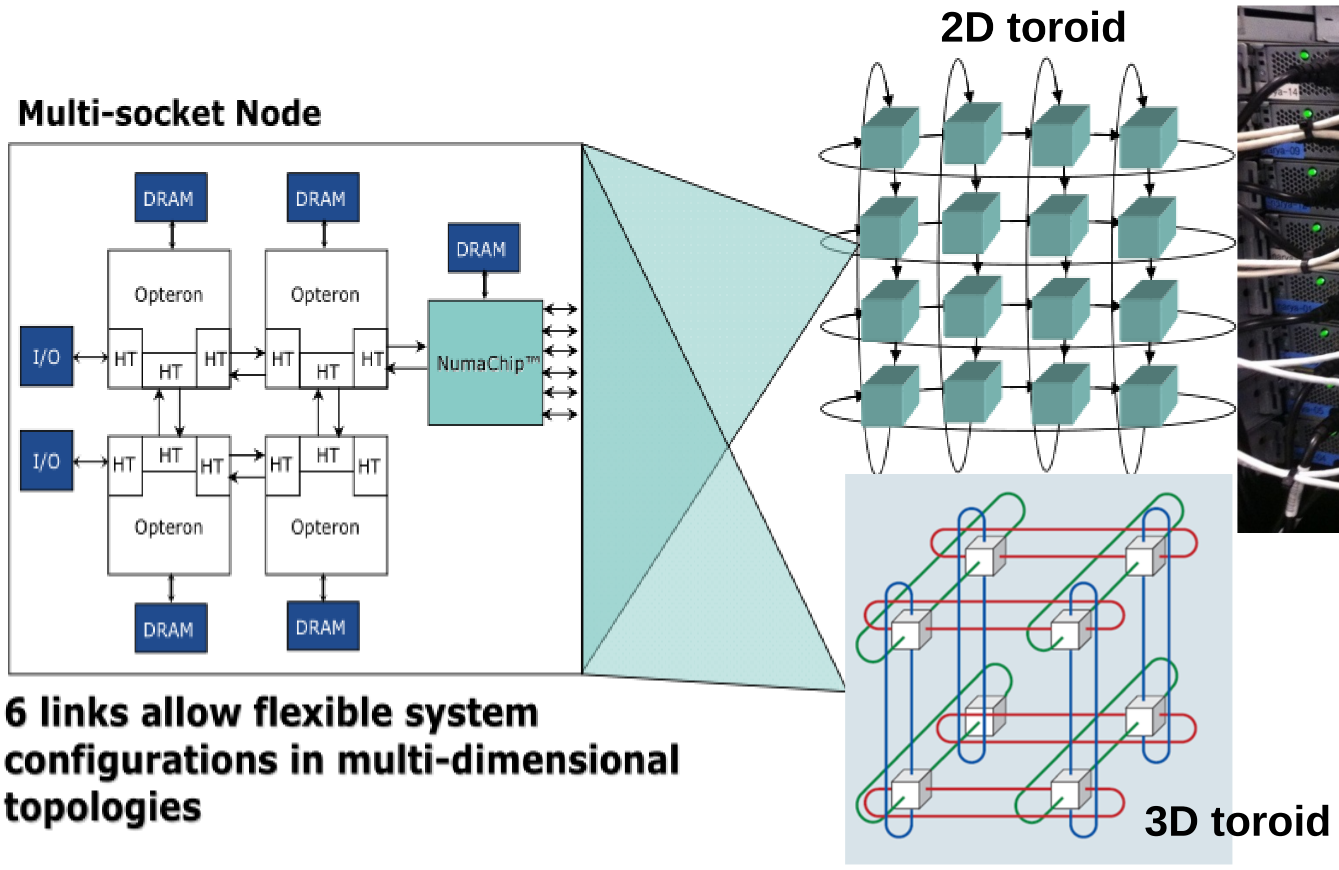
Example from practice – Numascale



HyperTransport connection on motherboard (+PCIe)
NumaConnect in Supermicro 1042
Processors (16/12/8/4-Core ready)
Numascale SMP adapter
Connectors to connect other nodes
Remote Cache DRAM
Remote Cache and Local Memory Tag DRAM
Fast Tag SRAM
Switch Fabric Connectors
Voltage Regulators
NumaChip With Heatsink/Fan
HyperTransport Connector
numascale
NumaConnect SMP Adapter

- Support cache coherence in HW (directory-based)
- Max. 4096 nodes. One node: 4 processors (on left-top picture)
- node: 16 DIMM sockets x 32 GB = max. 512 GB
- Max. 256 TB total memory. Limited by 48-bit physical address-space of CPU
- Example: 4GB Cache, 8 GB Tag (supports 240 GB Local Node RAM)

Example from practice – Numascale – Nodes interconnection



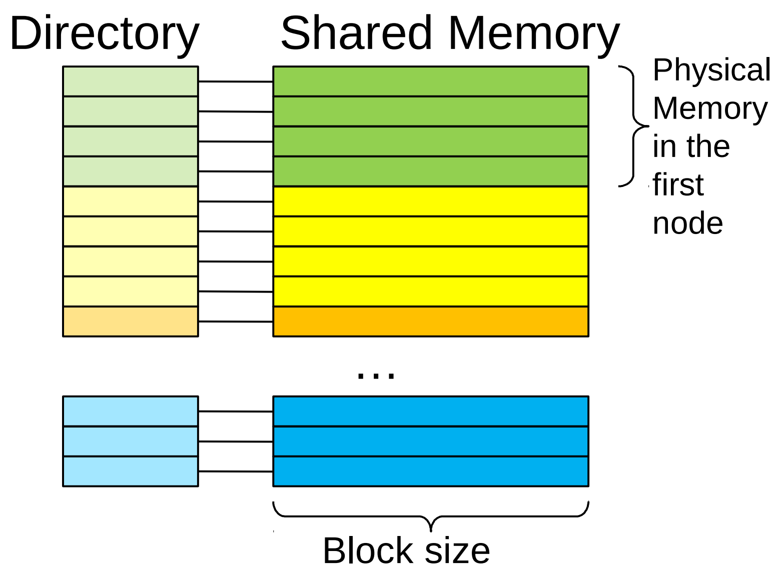
6 links allow flexible system configurations in multi-dimensional topologies

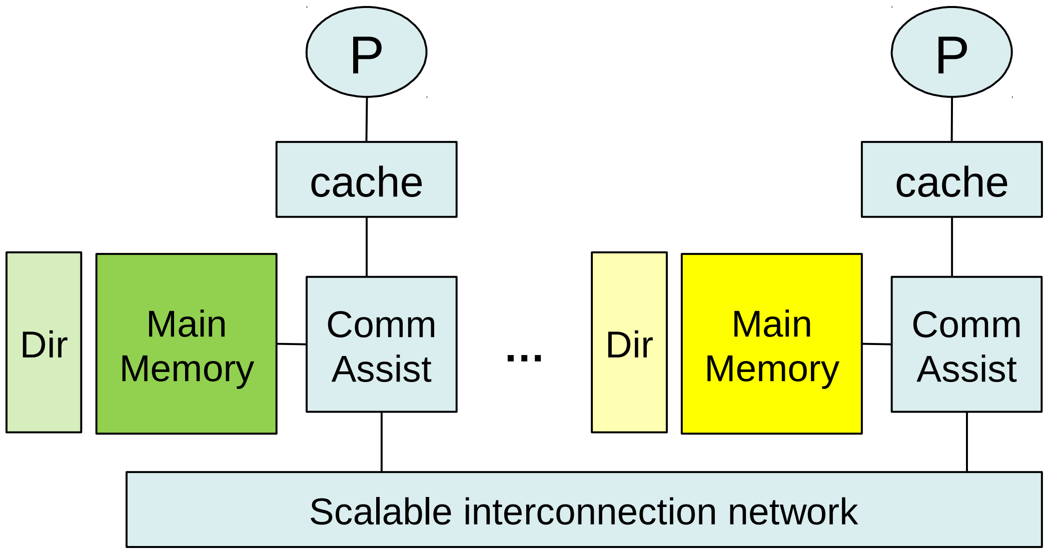
Nodes interconnected by 2D torus

- 2D torus – each node with 4 neighbors
- 3D torus – each node with 6 neighbors
- Max. 4096 nodes x 4 processor/node = 16 384 multi-core processors
- The program, which is written to run on a single node, can run on the whole system without change if written with scaling in mind (OpenMP, MPI, Threads)

Back to Directory solution from beginning …

- If broadcast (multicast) cannot be easily realized (not a bus)
- Core idea: Introduce Directory, which remembers for each line-block of memory:
  - If it is in the cache (at least one)
  - In which cache(s) it is present
  - If it is clean or dirty in the cache





Directory

- **Full directory** remembers complete information for each line of the memory. For n processor system it is a Boolean vector of length n+1. If bit i (i=1,2,…n) is set then corresponding (i) cache holds a copy of the line. Bit 0 indicates if the line is in the clean or dirty state (only one other bit can be set for dirty state = line is only in one cache)
- In NUMA system, each node implement only part of the directory with information corresponding to that lines which are stored in its memory = **home node**; the rest are **remote nodes** for this part of memory.
- For cache miss case, the request is sent only to the home node
- Full directory – disadvantage to big directory size. Example: for 8 processor system with L3 cache line size 64 B (consideration: coherence resolved at level L3), directory size is 2% (9/(64*8)=0.018) of the capacity of shared memory, but for 64 processors it is 13% (65/(64*8)=0.127). Bad scalability for thousands of processors.

Example of Full directory realization

- Consider K processors. Each memory line/block is equipped by 1 Dirty-bit, K Presence-bits.

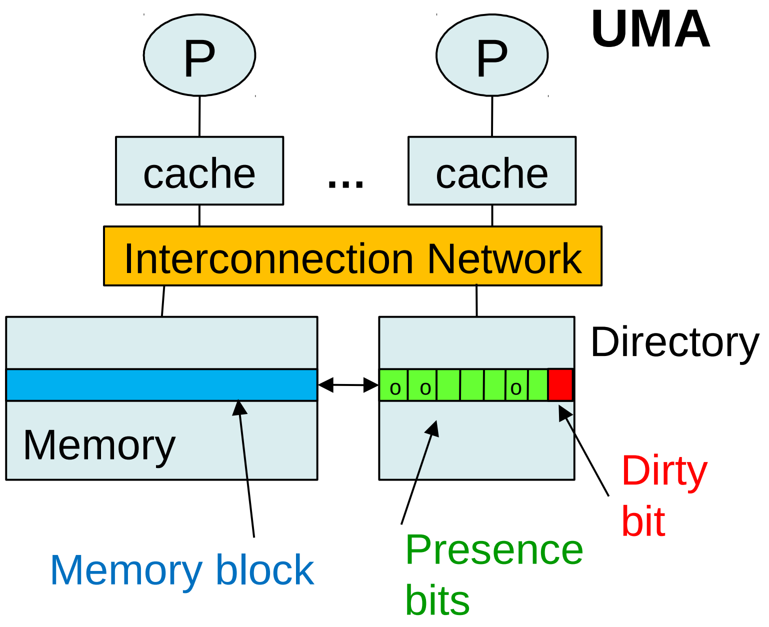
Read block by processor „i“ (after a read miss):

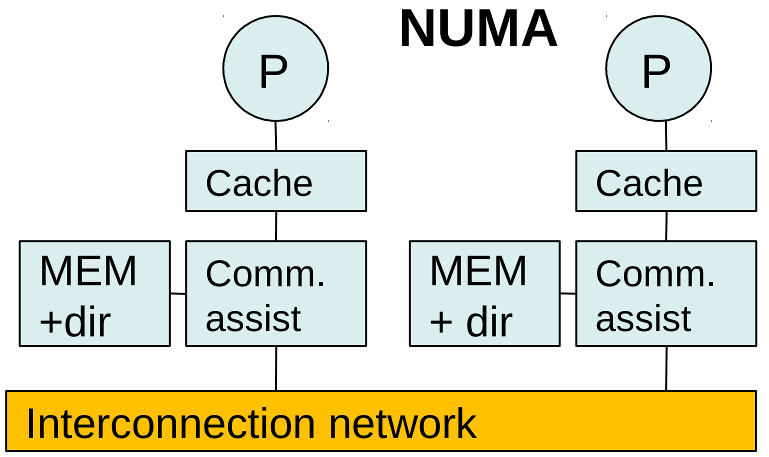
  - If dirty-bit OFF the { read from main memory; set p[i] ON; }
  - If dirty-bit ON then { request/stole the dirty line from the corresponding processor, update memory; set dirty-bit OFF; set p[i] ON; send data to processor I }

Write to memory by processor „i” (after write miss):

  - If dirty-bit OFF then { send invalidation to all shared copies; send data to i, clear all p[j] in the directory and set only p[i] to ON; dirty bit ON; }
  - If dirty-bit ON then {acquire (with invalidation) block from the corresponding processor; its p[j] bit clear; set p[i] to ON – only for new dirty node}

- Remark 1: If bit dirty is ON, then only one node (dirty node) can cache given block, and only single presence bit is ON
- Remark 2: Each block in the cache has: MESI, MOESI, another state corresponding to coherence solution between multiple cores/processors on the given node with common L3 cache.





Full directories use

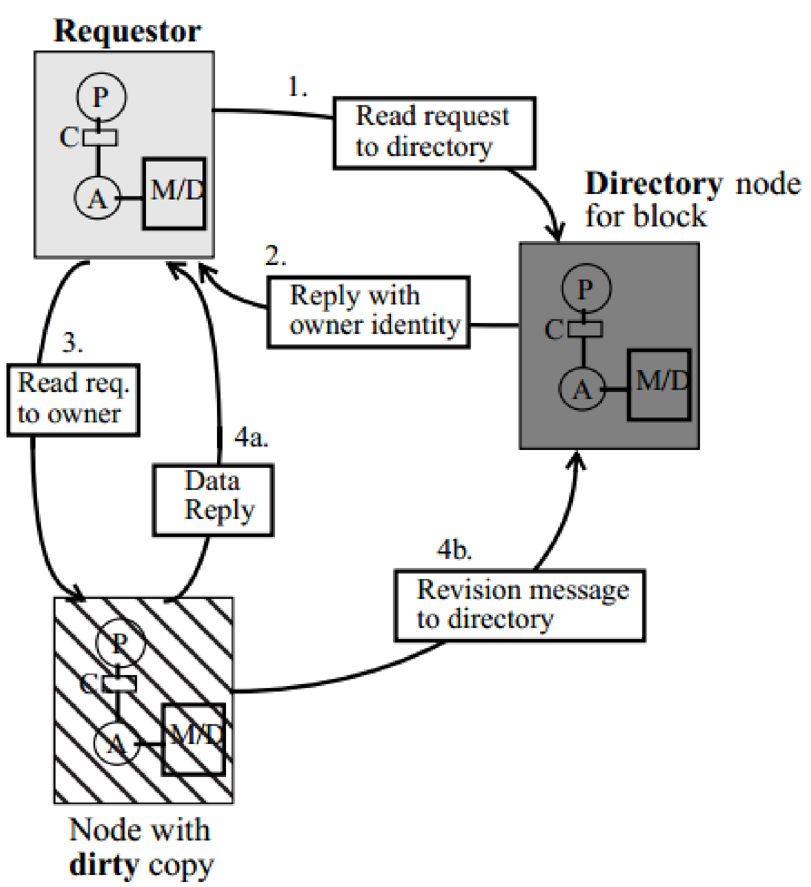
- CC-NUMA with Directories is the solution for large scale systems – Directory-Based CC-NUMA (Cache-Coherent NUMA).
- Directory and memory are distributed between nodes.
- Example SGI Origin2000 – 512 nodes x 2 processors = 1024 x MIPS R10K
- The idea is based on the fact that information about the state of each block is available (maintained). This information is stored in the directory. Broadcast are not necessary for such case, and a limited number of point-to-point transactions is necessary for each miss.
- The **home node** is that node which memory contains requires data, other nodes are **remote nodes** for that address range.
- **Number of shared copies is usually small even in large systems, and this ensures significant communication reduction when compared with Broadcast based solution.**
- The price to pay is requirement to include additional resource – addres (Directory): 4GB node, block 128B, 256 processors => Directory size per node 32 M x 256b (bit-map) = 256 MB.

Definitions

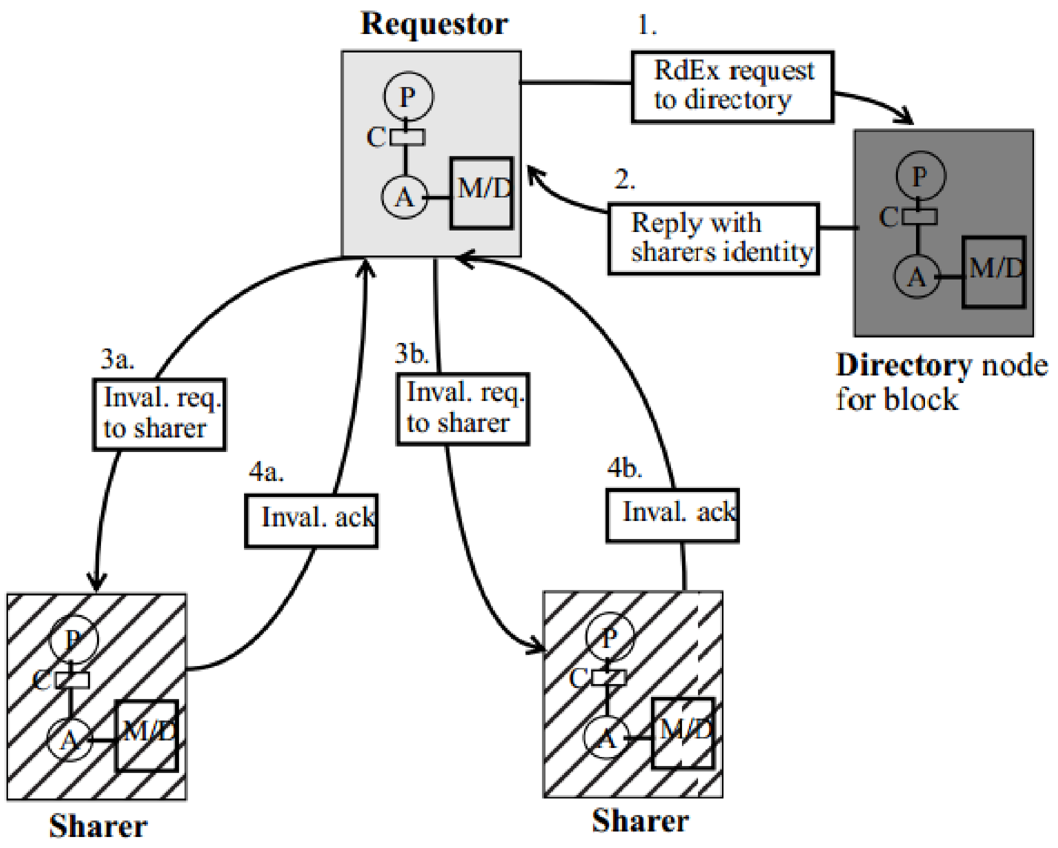
All of them related to each block in memory or cache:

- **Home node** – node which provides given memory block – where it is allocated and corresponding memory connected
- **Dirty node** – node which owns a copy of given memory block in its cache in the modified state (M)
- **Owner node** – node with valid memory block copy in its cache and is responsible for delivering data when they are requested (can be home or dirty node) (M, O, E)
- **Local node – requester** for data: node witch send the request to shared or exclusive access to the memory block
- **Remote node** – all other nodes which than local node for the given memory block

More detailed directory supported operations



(a) Read miss to a block in dirty state



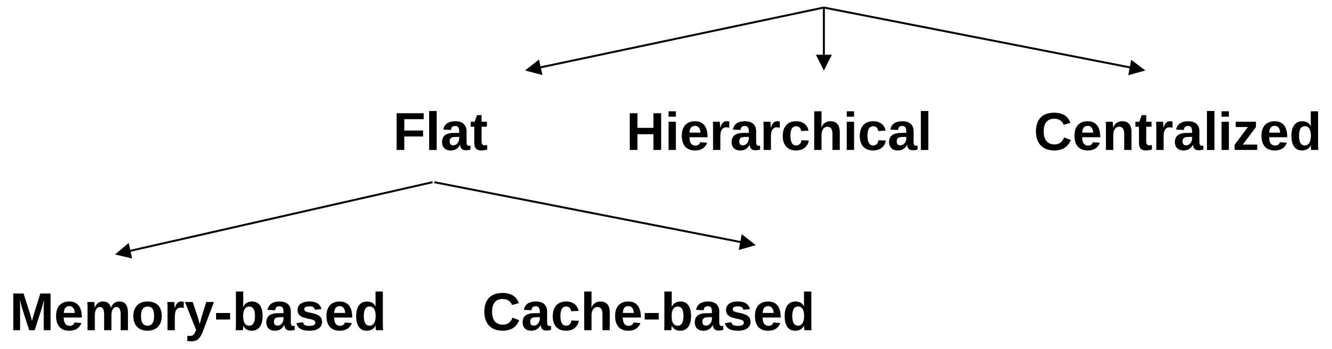
(b) Write miss to a block with two sharers

Directory Storage Schemes

How to find where is the part of directory corresponding to the given memory block address?

- The most common is **Flat Directory** where this part of the directory is placed on fixed location – usually on home node. It can be obtained/derived directly from block address (address from which CPU wants to read or write)
- Other option hierarchical directory where memory is distributed between nodes usual way, but the directory is stored in form tree (logic structure). Nodes (processors) are located in three leaves, and nodes of the tree keep information about given block: if its children have or do not have the copy of the block. The cache miss is then realized as walking through the tree in the direction to parents. In practice, tree nodes are distributed between system nodes (processors) and each miss generates usually multiple transactions between system nodes before required information is found.
- Centralized directory – advantage – a single place to send queries – can be used only for small systems – the example is coherency maintenance inside multi-core processors (in the fact inclusive cache hierarchy fall between central and tree solution). Such node can be a member of the larger system.

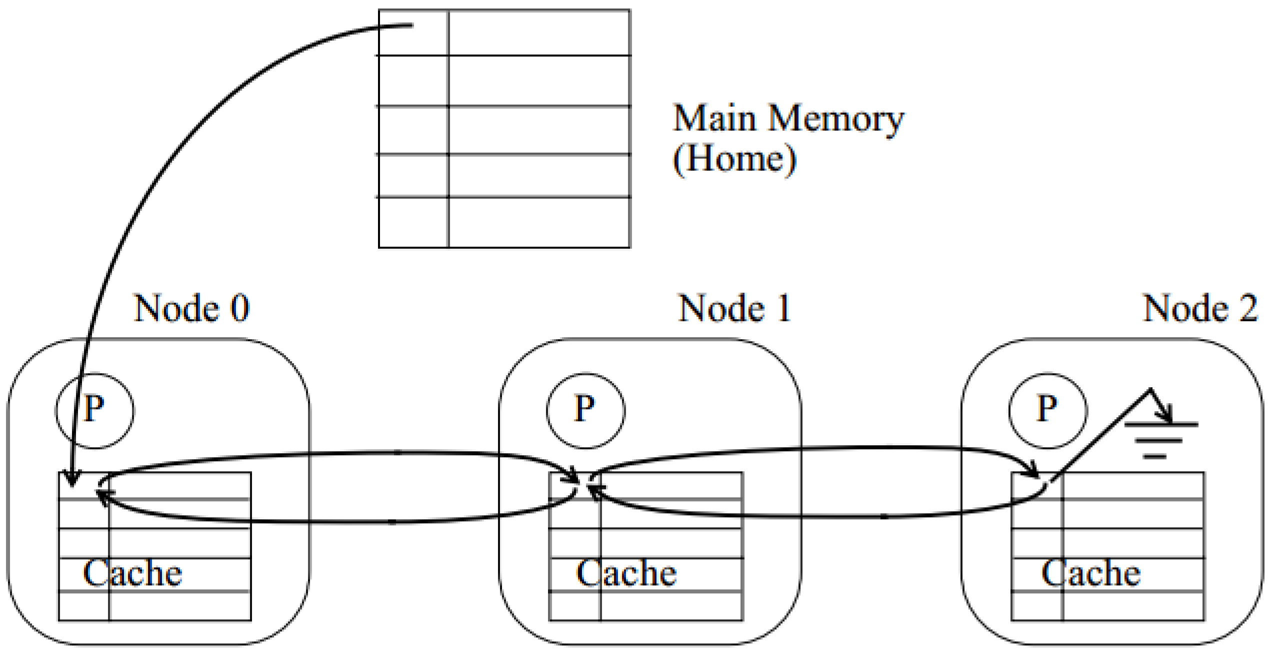
Directory Storage Schemes



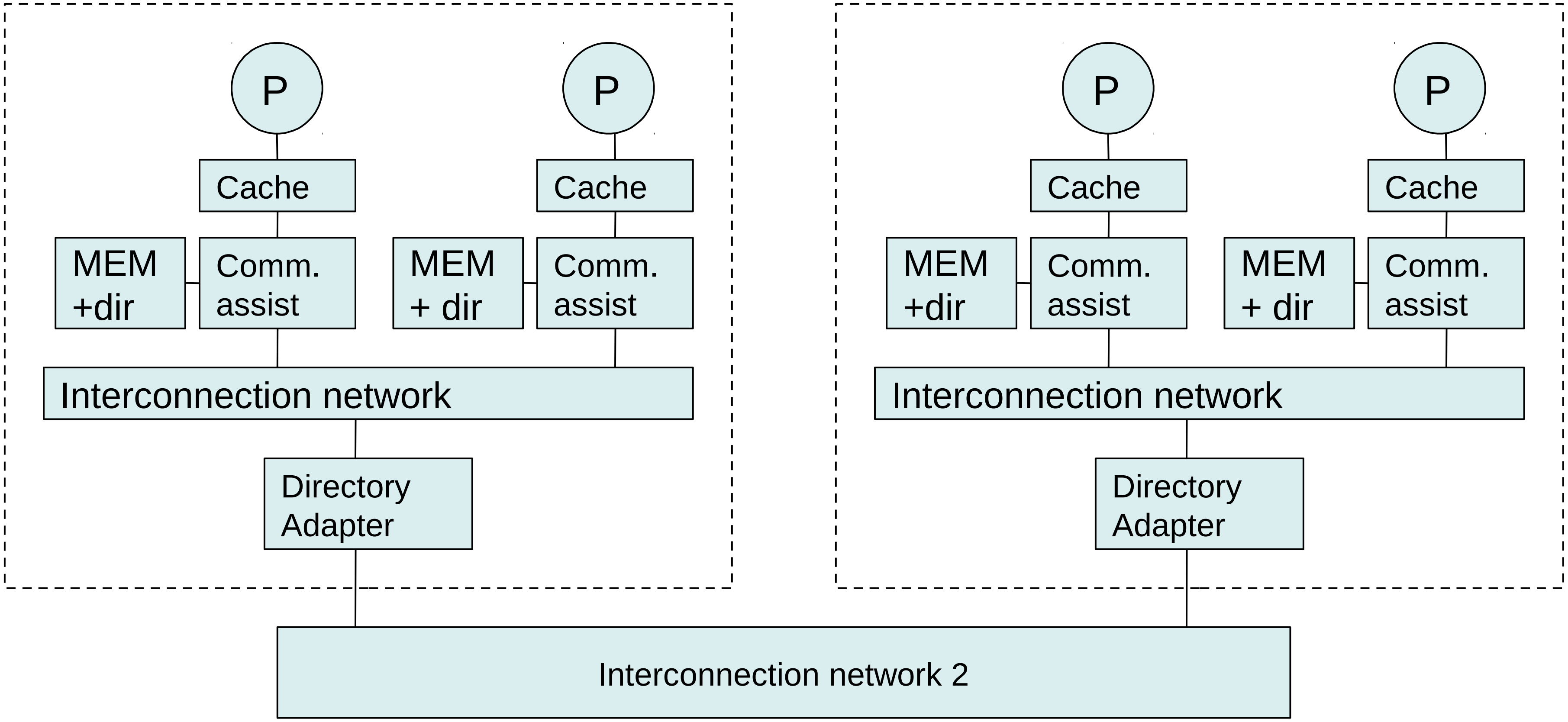
- Flat directory - **Memory based** – a record for a dirty node or all sharing nodes is kept in the home node (record in memory). The example is discussed the full-directory solution. An alternative is the solution where remote nodes (processors) are recorded by their number instead of the bitmap (number of sharing nodes is then limited – usual situation). The system has to be prepared for the situation when a number of remote nodes requesting single block is above the limit (example solution is forced invalidation in remote cache on oldest age basis …)
- Flat directory - **Cache based** – the record in home node does not keep information about all sharing nodes but the only pointer to the first sharing node is kept (plus state bits). Records about additional sharing nodes are stored in distributed bidirectional linked list (its entries in remote caches). The second and additional sharing nodes are found by iteration over the list. Cache which contains the copy of the block stores pointer to next and previous node as well.

Distributed directory – bidirectional linked list of sharing nodes

- IEEE standard SCI
- <u>**Scalable Coherent Interface**</u>
- The protocol based on rules for enlisting and removal of entries from linked list… for example used in SEQUENT NUMA Q, Convex Exemplar.



Two-level cache coherent system



- Directory-directory
- Alternatives: Snooping-Snooping, Snooping-Directory, Directory-Snooping

Summary and conclusions

- The basic solution for coherence maintenance is snooping – protocol MESI or newer MESIF (better for point-to-point interconnect)
- Snooping request message are used today instead of snooping on the shared bus due to change to point-to-point interconnect – typically ring, or 2D mesh
- Directories are used today for larger systems (>8 nodes)
- Hybrid and hierarchical solutions – snoop+directory systems
- Programmer model and competence are still significant for the development of scalable solutions using multiprocessors.
- Debugging and performance tuning is not easy.
- Large scale multiprocessor systems typically ensure memory coherence in individual computational nodes (more multi-core CPUs) – OpenMP. Data exchange in the cluster of nodes is under programmer control – MPI (Message Passing Interface). => Combination of OpenMP + MPI.

References and literature:

1. Shen, J.P., Lipasti, M.H.: Modern Processor Design: Fundamentals of Superscalar Processors, First Edition, New York, McGraw-Hill Inc., 2005
2. Bečvář M: Přednášky Pokročilé architektury počítačů.
3. https://www.cs.utexas.edu/~pingali/CS395T/2009fa/lectures/mesi.pdf
4. D.E.Culler, J.P. Singh, A. Gupta: Parallel Computer Architecture: A HW/SW Approach, Morgan Kaufmann Publishers, 1998.
5. Einar Rustad: Numascale. Coherent HyperTransport Enables the Return of the SMP
6. https://www.numascale.com/numa_pdfs/numaconnect-white-paper.pdf
7. Rajesh Kota: HORUS: Large Scale SMP using AMD Opteron processors. Newisys Inc., a Sanmina-SCI Company. http://download.microsoft.com/download/5/d/6/5d6eaf2b-7ddf-476b-93dc-7cf0072878e6/LargeScaleSMP.doc
8. http://www.intel.com/content/dam/doc/white-paper/quick-path-interconnect-introduction-paper.pdf
9. David Culler, Jaswinder Pal Singh, Anoop Gupta: Parallel Computer Architecture. A Hardware / Software Approach.
10. http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/itanium-9300-9500-datasheet.pdf
11. https://tu-dresden.de/zih/forschung/ressourcen/dateien/abgeschlossene-projekte/benchit/2015_ICPP_authors_version.pdf?lang=de
12. Michael R. Marty: Cache Coherence Techniques for Multicore Processors, 2008.
13. http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/itanium-9300-9500-datasheet.pdf