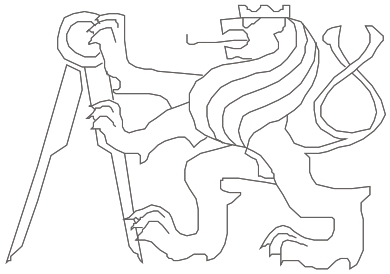
Advanced Computer Architectures

Multiprocessor systems and memory coherence problem



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

Content

- What is cache memory?
- What is SMP?
- Other multi-processor systems?

  UMA, NUMA, aj.
- Consistence and coherence
- Coherence protocol
- Explanation of states of cache lines

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**. Unique instance of OS on each node (processor, group of processors - hybrid). Network protocols, RPC, Standard: MPI. Sometime labelled 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:

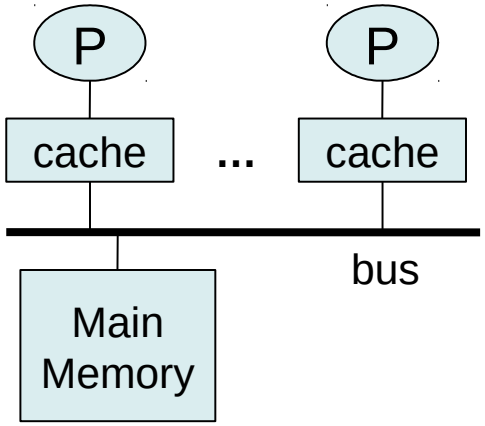
- Shared memory systems – single/common physical address-space – SMP: UMA
- Distributed memory system – memory physically distributed to multiple nodes, address-space private to node (cluster) or global i.e. NUMA, then more or less hybrid memory organization

Definitions:

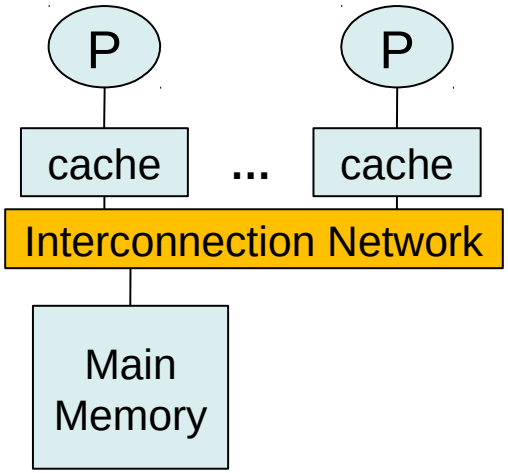
- **SMP** – Symmetric multiprocessor – processors are connected to cetral common memory. Processors are **identical** and „access“ memory and rest of the system same way (by same address, port, instructions).
- **UMA** – Uniform Memory Access – all memory ranges are accessed from different CPUs in same 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** – group of cooperating computers with fast interconnection network and SW which allows to view group as a single computing system.

Multiprocessor systems - examples

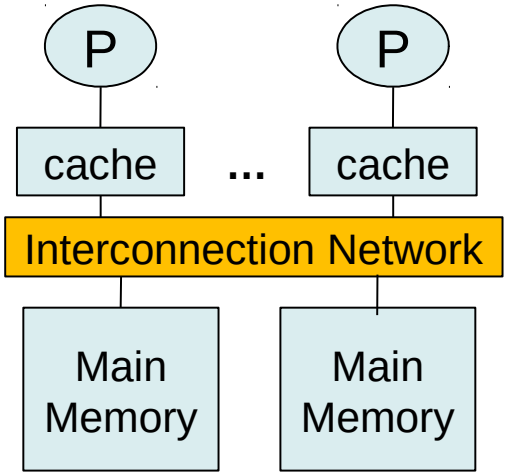
Traditional SMP, and UMA



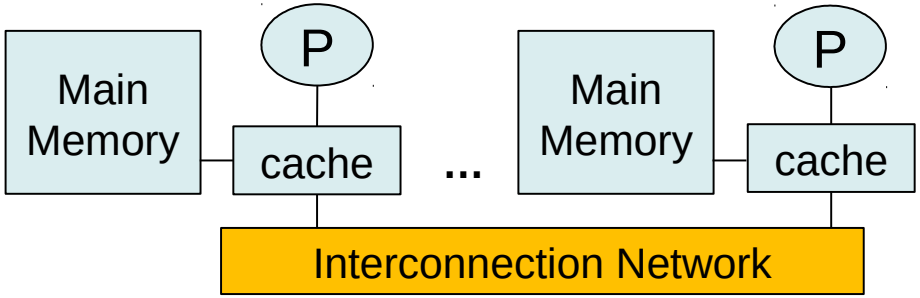
UMA



UMA



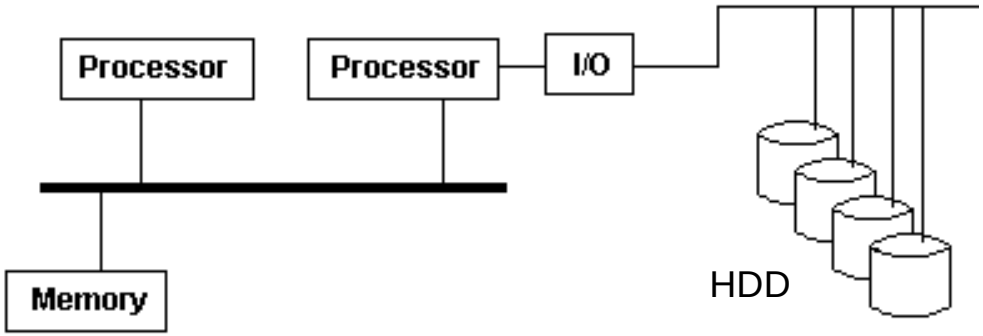
NUMA

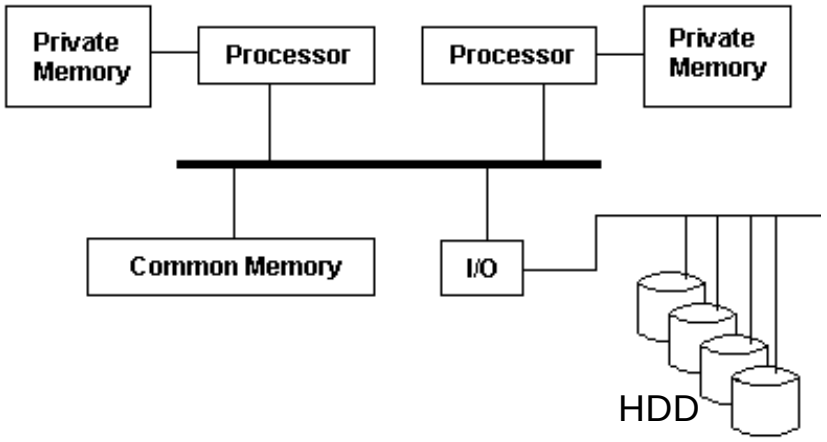


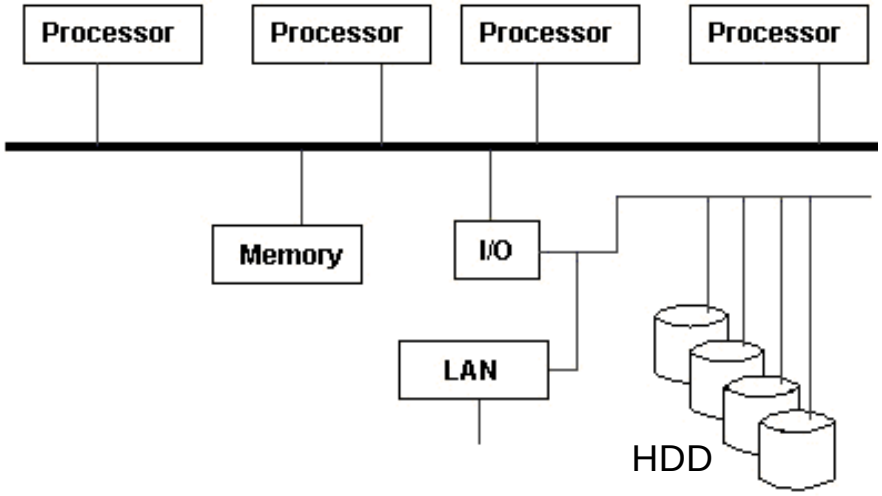
Interconnection Network

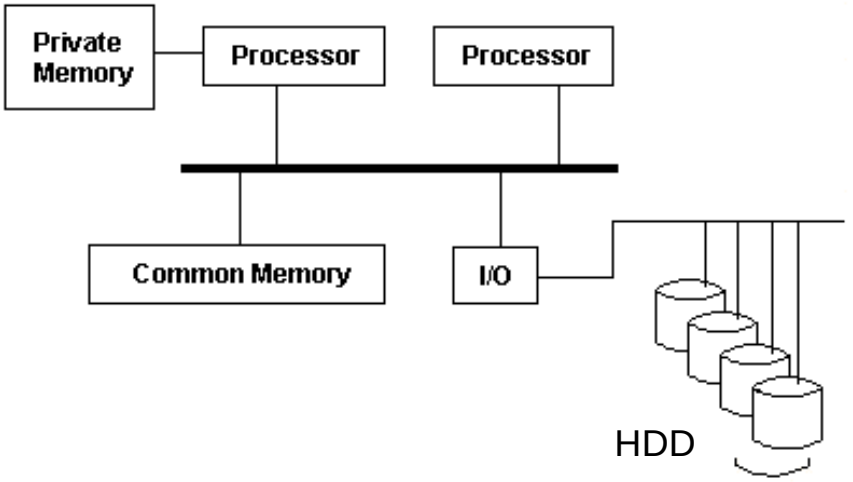
It can be a crossbar switch, or a dynamic multi-stage interconnection network, or some type of fast static network

Try to find SMP system ???



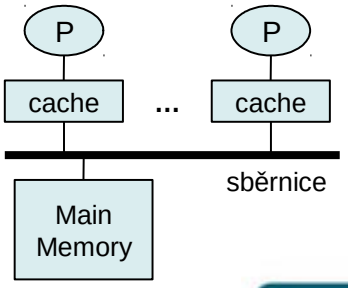


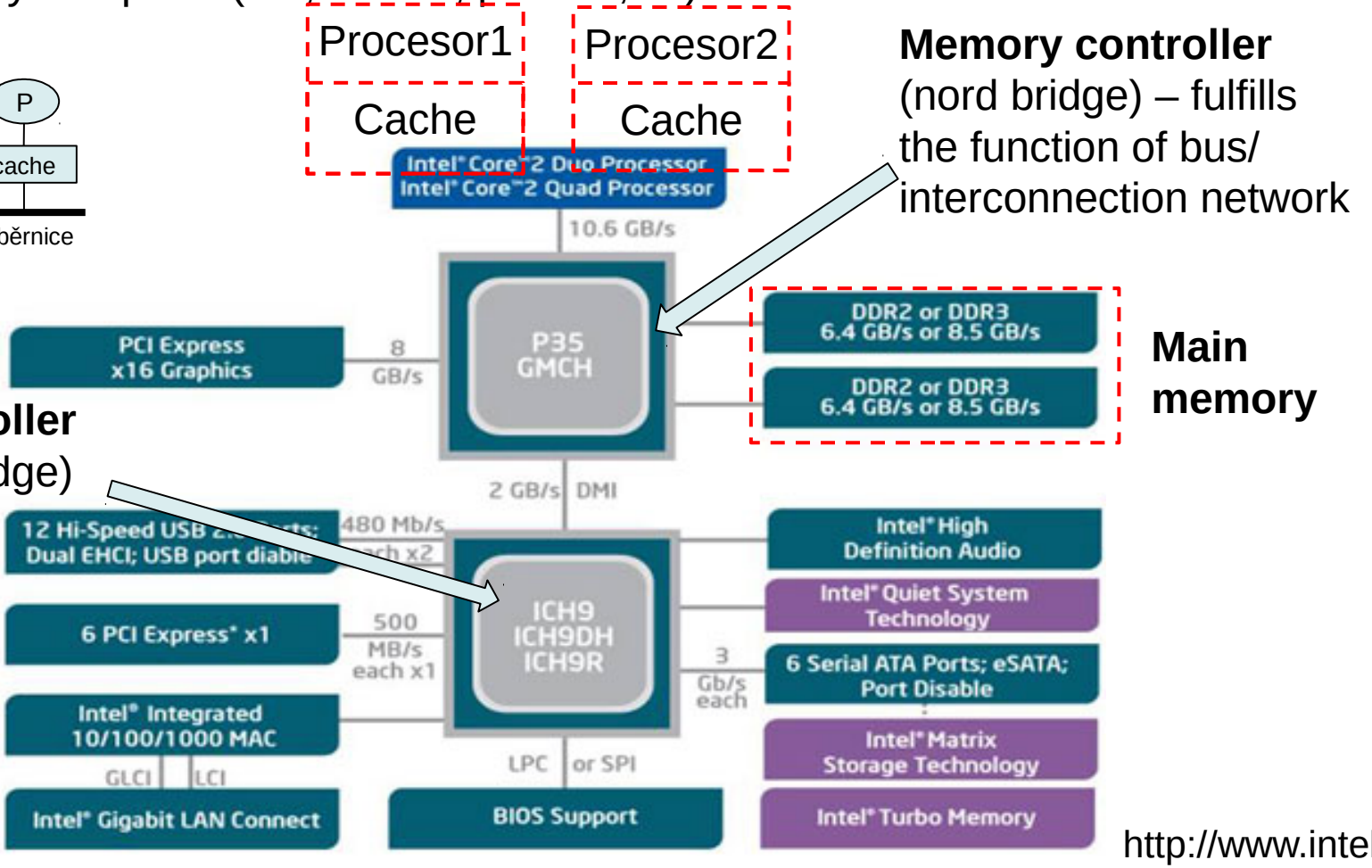




PC – from SMP to NUMA development

- 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





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

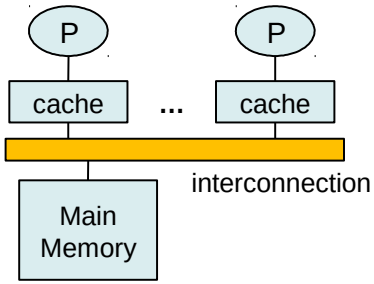
Main memory

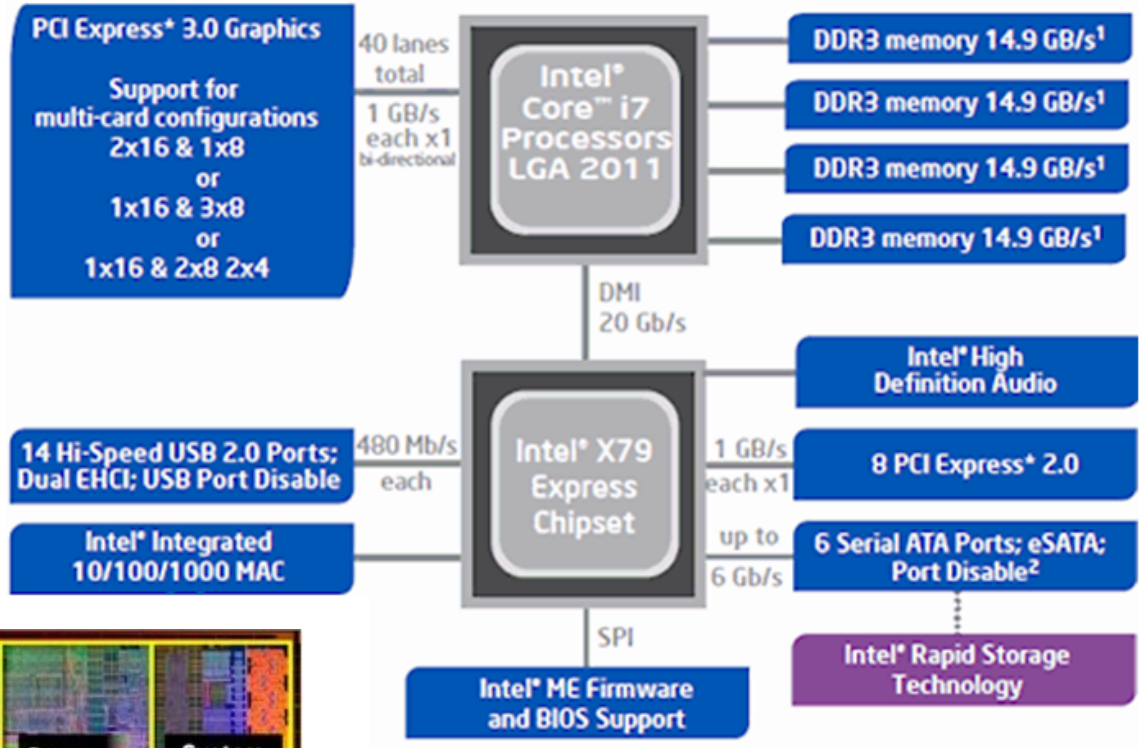
I/O controller (south bridge)

http://www.intel.de

PC – from SMP to NUMA development

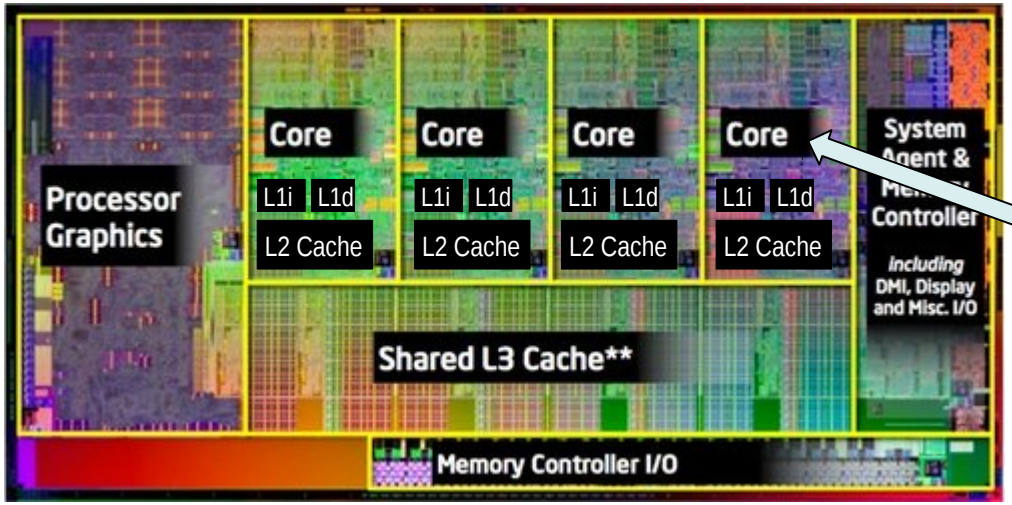
- Further development shifted the function of the northern bridge directly into the processor.
- The memory controller can be found there.





http://www.intel.de

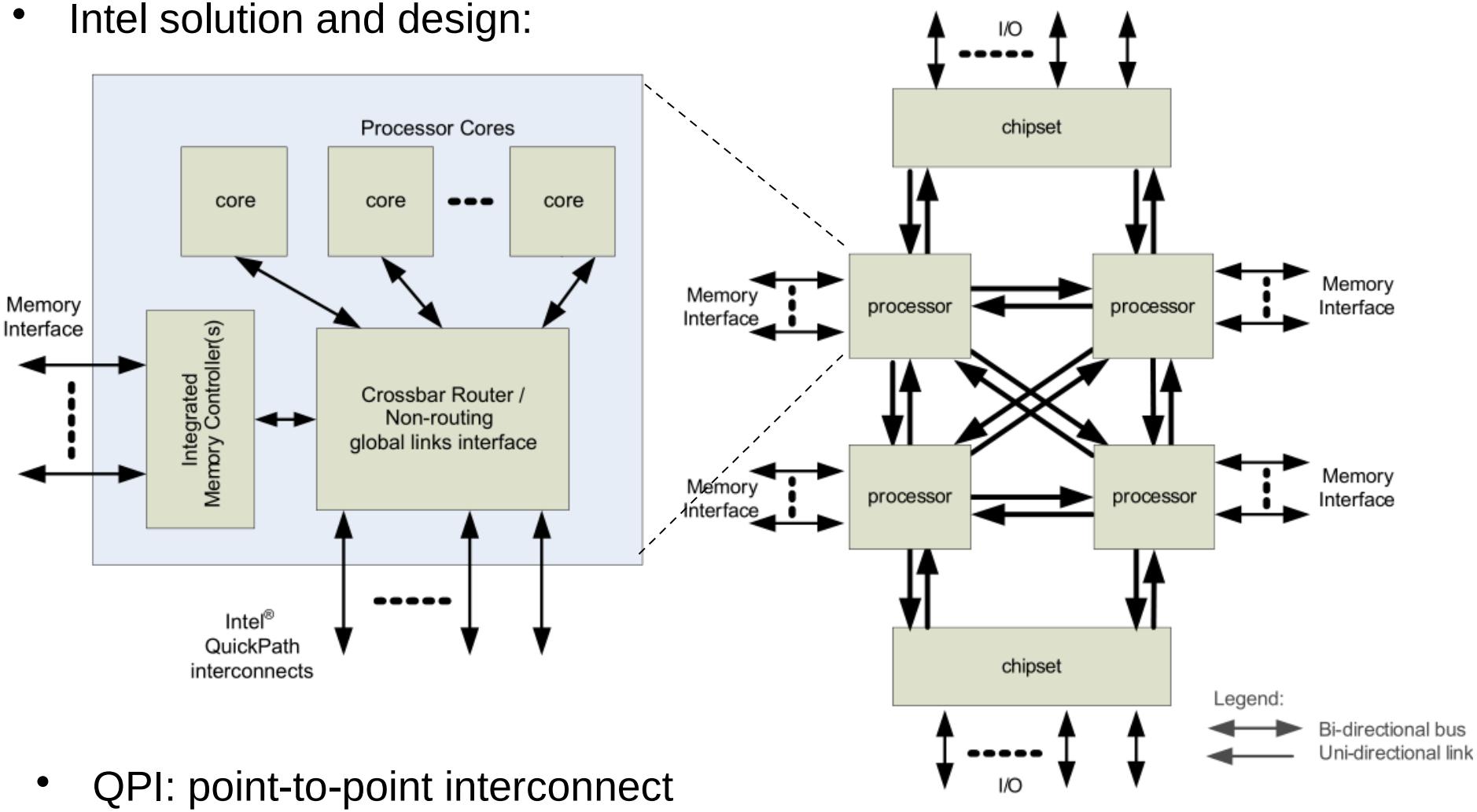
Core i7-2600K



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 **NUMA** system.
- Intel solution and design:

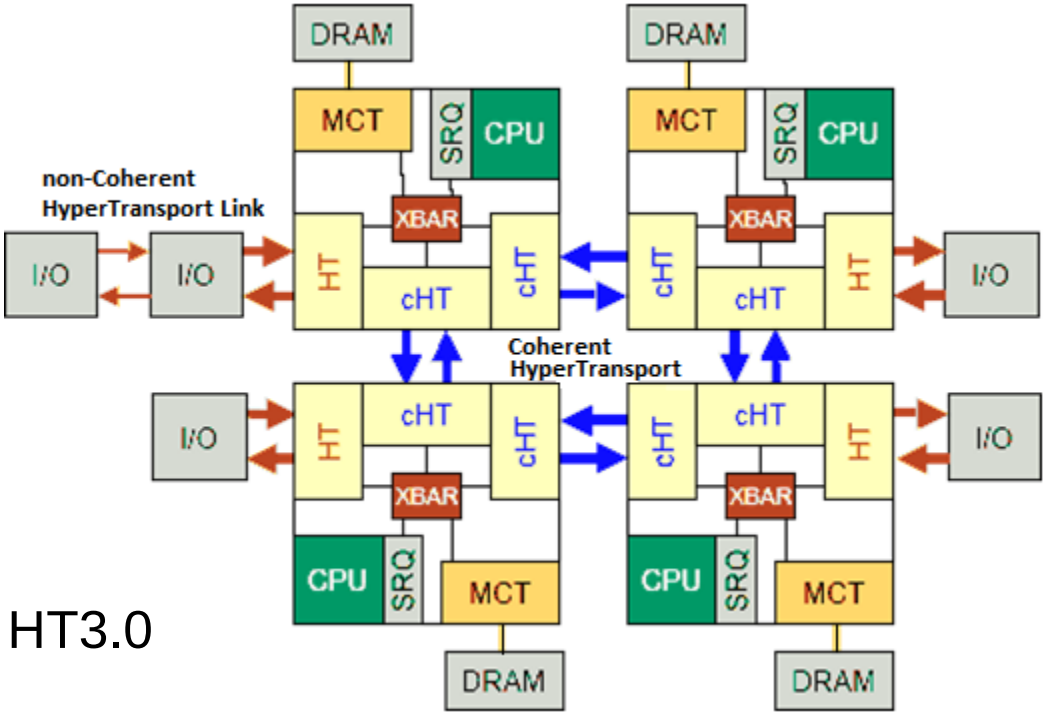


- QPI: point-to-point interconnect

PC – from SMP to NUMA development

- AMD solution and design (HT - HyperTransport):

Motherboard



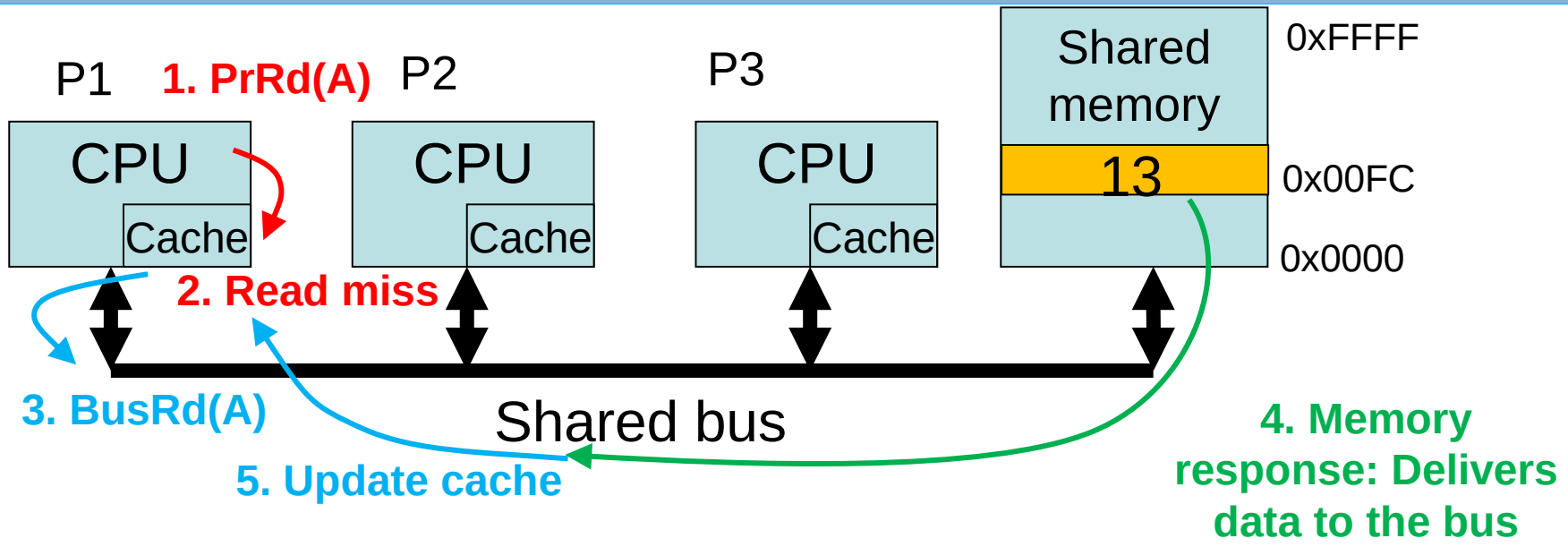
- **Four AMD Opteron**™ 6000 series processors (Socket G34) **16/12/8/4-Core ready**; HT3.0 Link support
- **Up to 1TB DDR3** 1600MHz
- 6x SATA2
- 1x PCI-E 2.0 x16 slot

This lecture foccus:

Shared memory systems

- Often developed as extension of 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. This solves transparently HW. Be careful about synchronization/race conditions.
- Difficult to scale for larger numbers of processors.

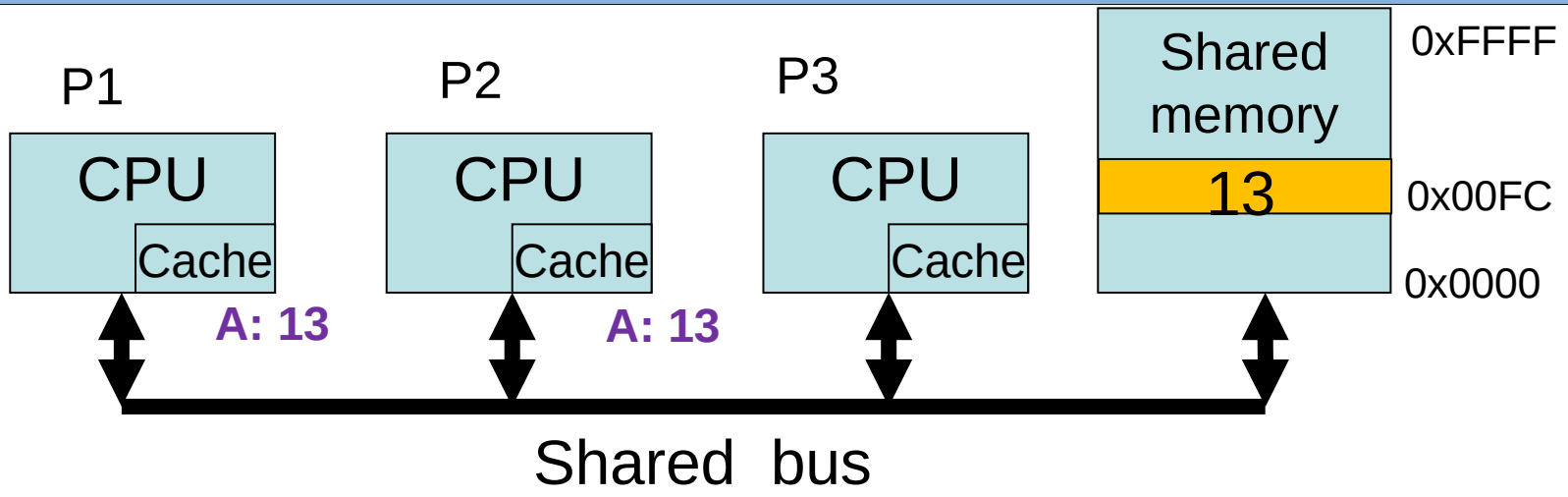
What is the basic problem? Consider the write-back cache



Intent of processor P1 to data read 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. Memory controller recognizes request to read from given address (0x00FC) and provides data, value 13 for this example.
5. 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?

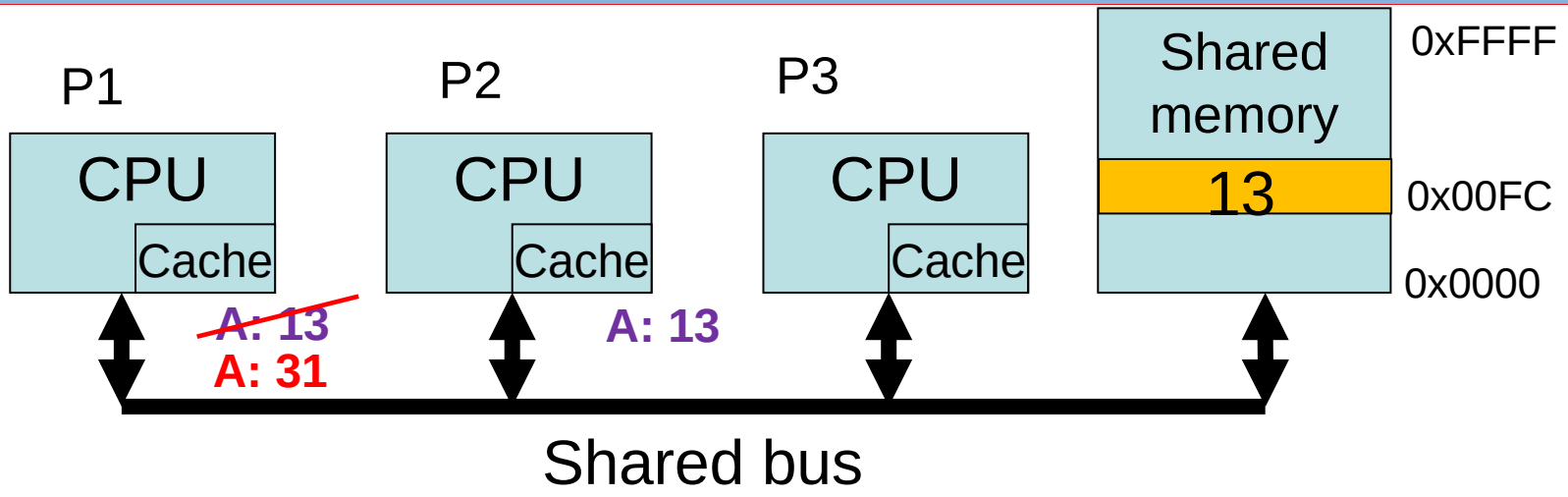
- P1 received data into its cache. A: 13 (address:value)

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

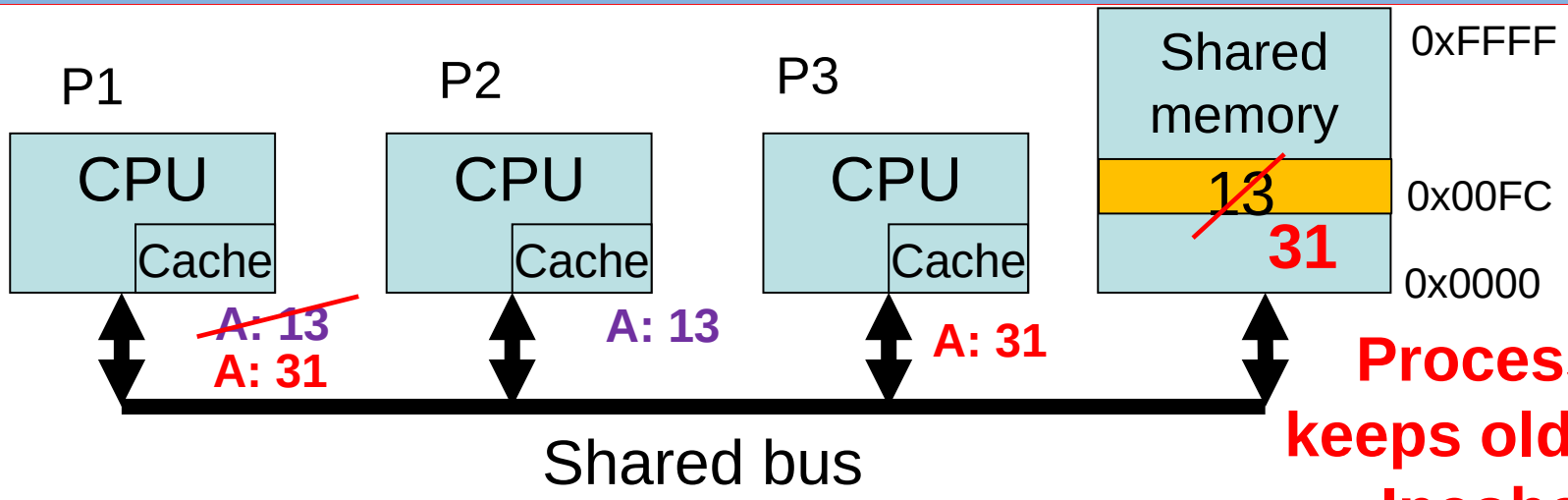
Processor P1 writes new value into a A. 31 for example.

- Value in its cache is modified.

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

- Memory provides **13**. But processor P1 woks 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 new value into a A. 31 for example.

- 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?**

Result of previous analysis

- The problem lies in memory **incoherence**:
- Processor modified (appropriately) data in its cache
- Even immediate change in main memory is not enough.
- Chache 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 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 a 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.

Formal definition of memory coherent system

Definition 1. M*emory 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 *op.* Write *targeting* same SM cell (address) executed by *two processors* are seen by all *processors in same order*.

This definition is entirely equivalent to the definition of the previous slide. Its advantage is the formal 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 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 their variants.

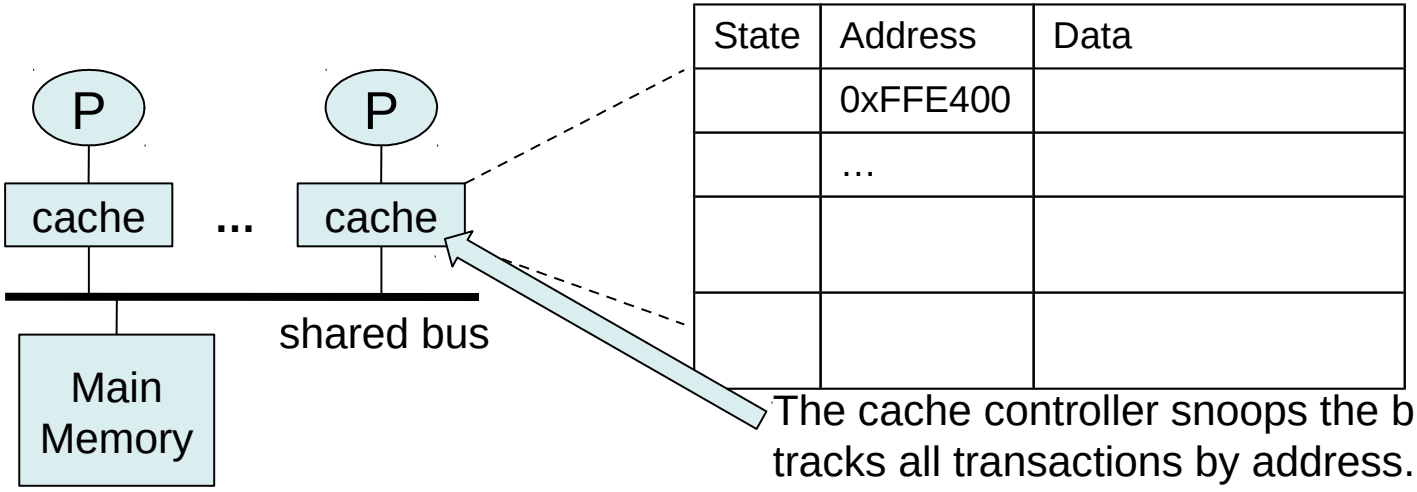
2. Directory based

- Used for larger scale systems where common shared bus cannot be used/implemented

Bus snooping

Variant where each processor knows which other processors have 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.
- Requirement for global bus is main problem for scalability.
- The variant using „Directory based“ is better scalable.



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: write-update, write-invalidate

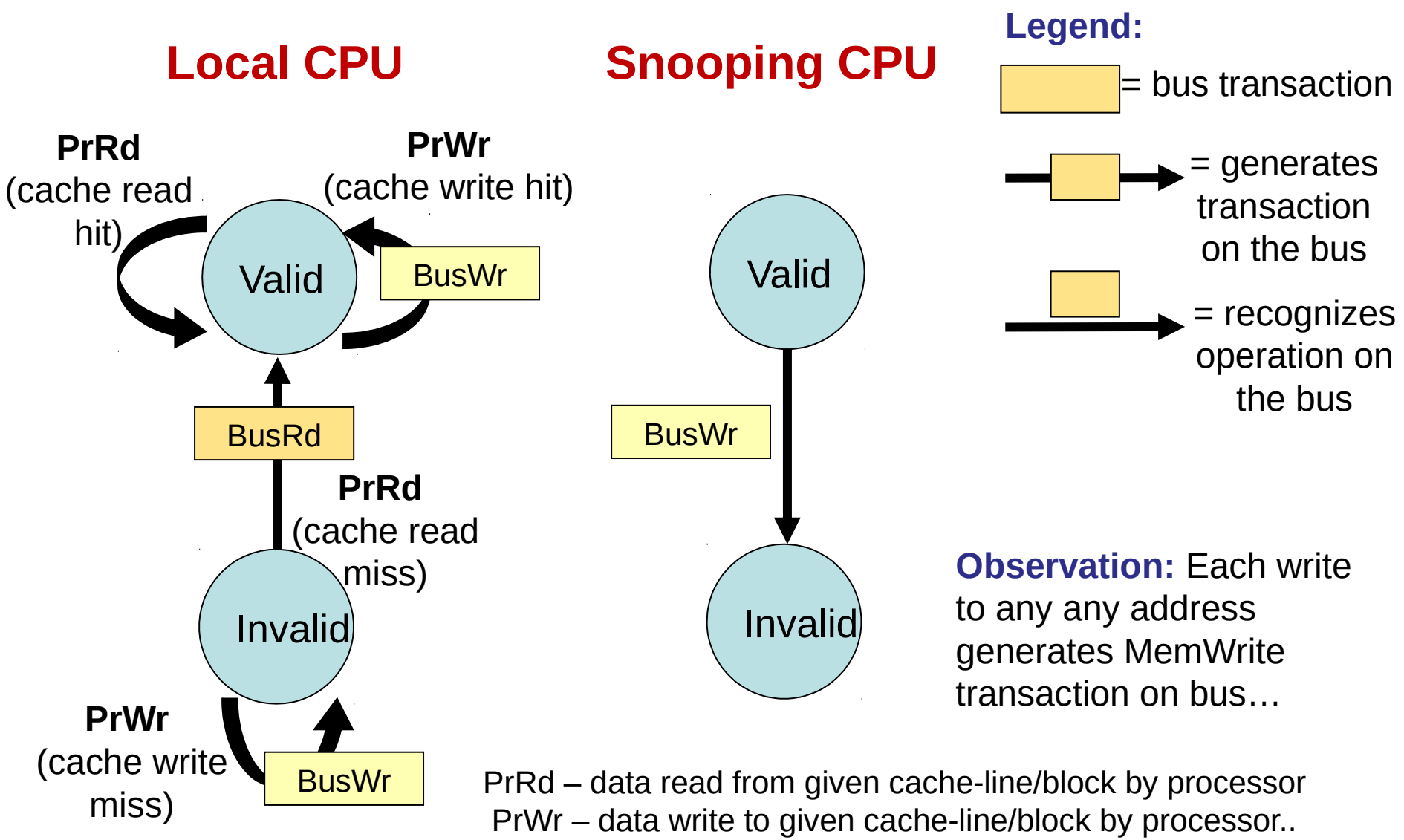
- Write-update
  - Before processor can write data, it has to acquire bus. Then it sends new data. (All processors including memory are connected to the same bus)
  - **All snooping controllers actualize their data if address matches and memory writes them unconditionally.**
  - The varinat loads the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

Snooping protocols

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

Protocol Write Through Write No Allocate

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



Scalability of WTWNA protocol

Example: Consider 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 Write oparation stores **L = 8** B.

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

Solution:

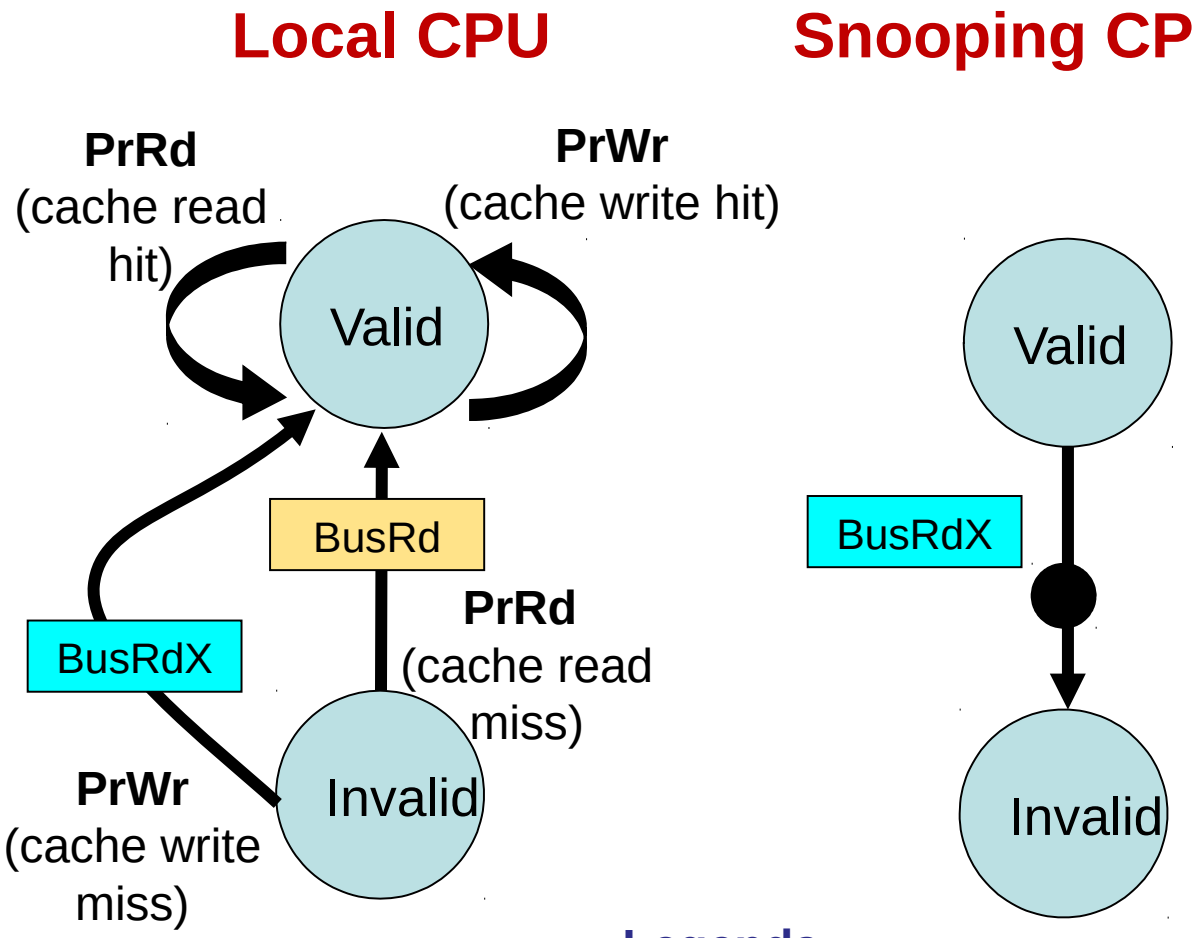
- Single P generates **w = s*IPC*f** Write operations per second
- Final bandwidth requirement of one P is **r = w*L** B/s.
- For our case, r=0.15*1*1.6G*8 = 1.92 GB/s
- Only four processors are satisfied even if we ignore bandwidth required for read miss situations

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

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 whole block, some bytes from snooping processor cache would be lost if it is in valid state. That is why copy-back is required before invalidation.

Observation: Even if processor **only read** 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).
- 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 system by each other CPU results only in constatnt 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 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.

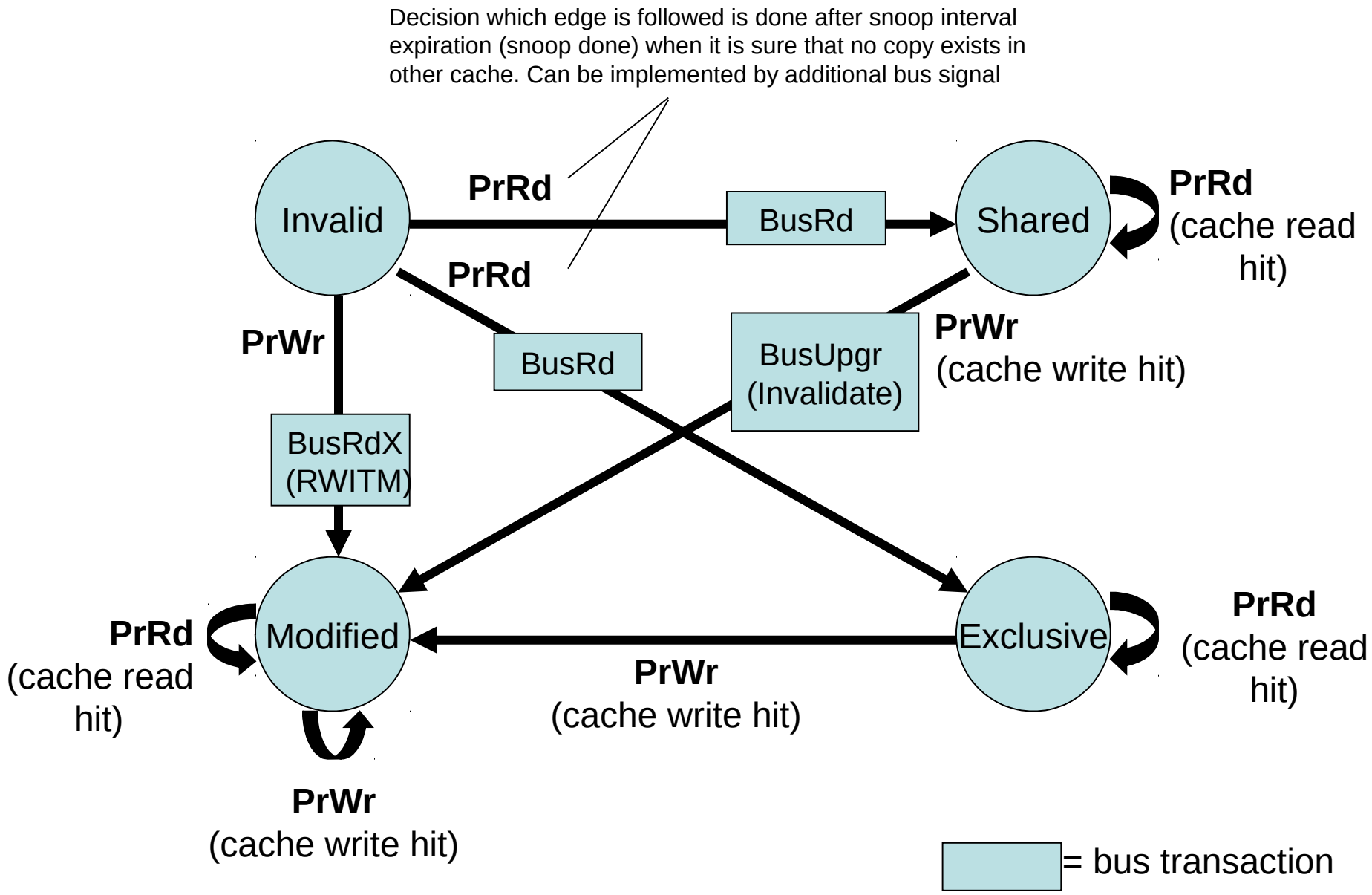
MESI protocol

- It is protocol which minimalize 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 attempt to modify 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 ocurs 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 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.

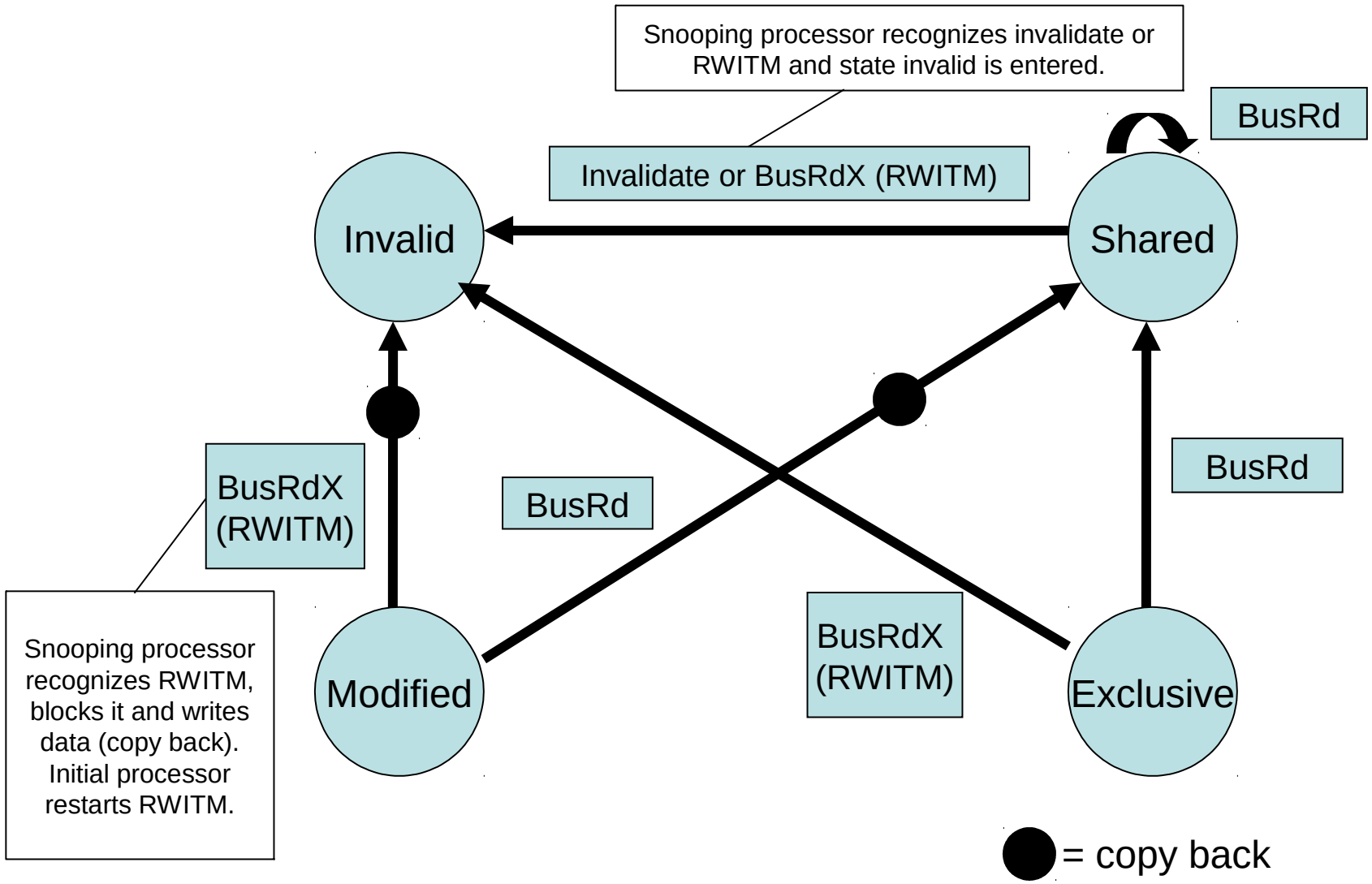
MESI

- Required actions are comprehensivelly sumarized by **state diagram** of transitions. It defines what happens with cache-line of 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 processor)
  - Read Miss (value is not in cache, bus transaction required)
  - Write Hit (successful write)
  - Write Miss (corresponding line is not in cache, bus transaction is required)

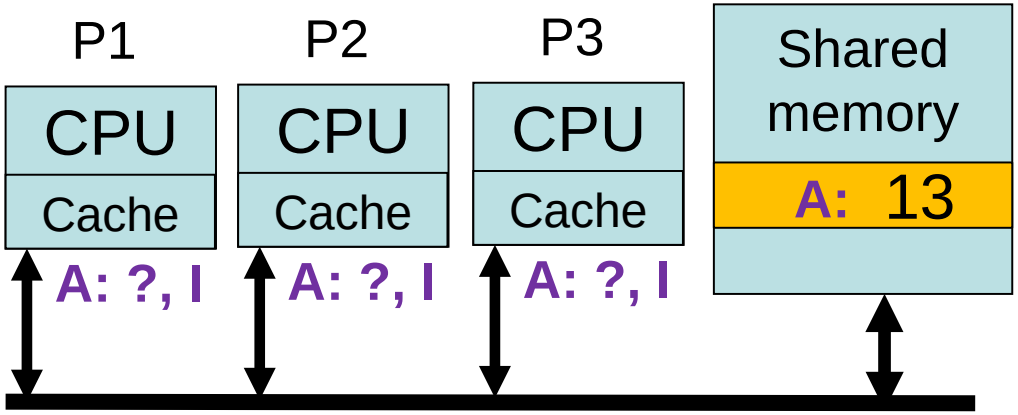
MESI – Local processor

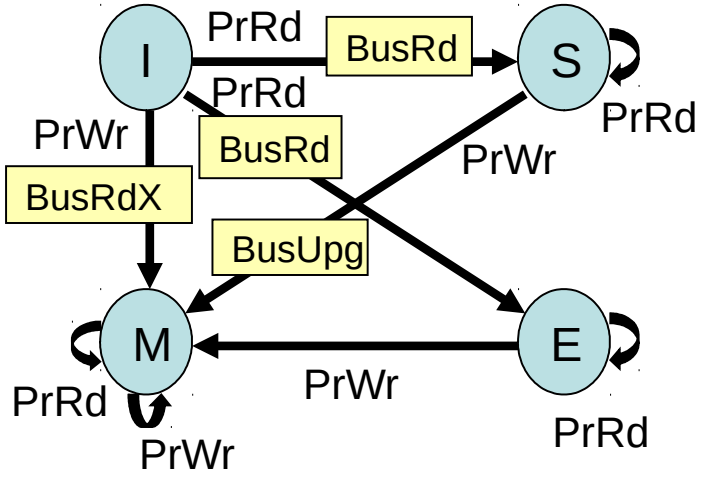


MESI – Snooping processor

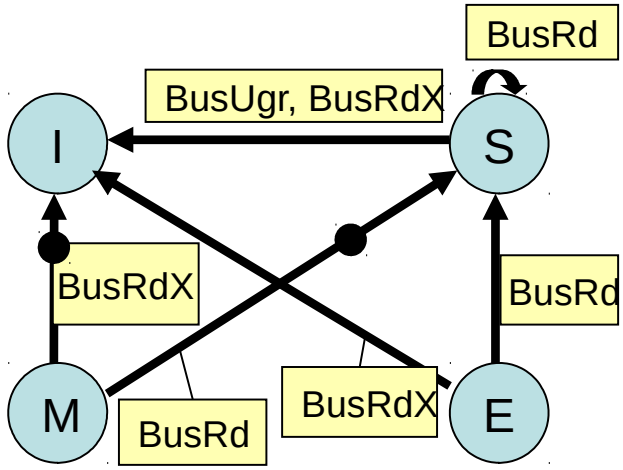


Example – MESI protocol

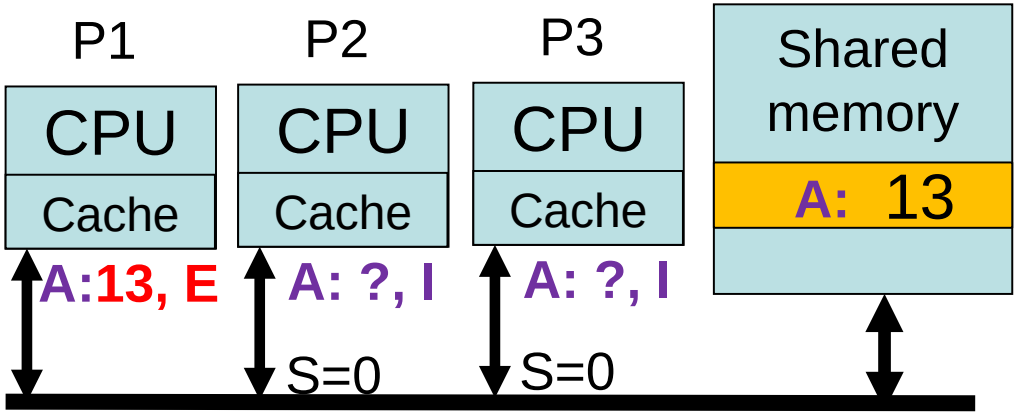


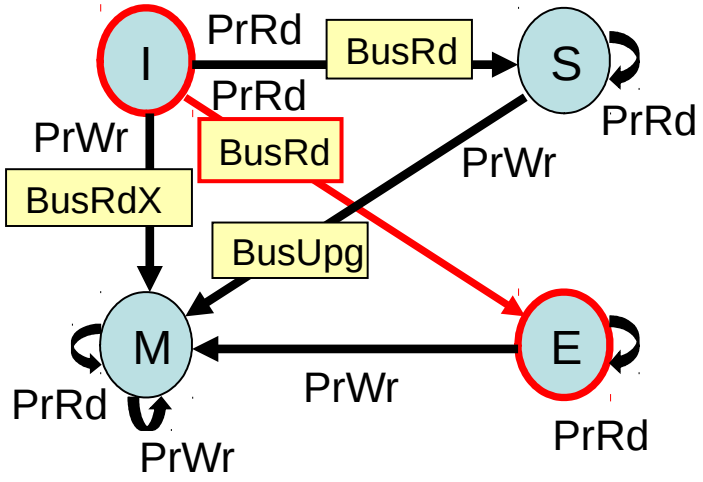


Initial state.



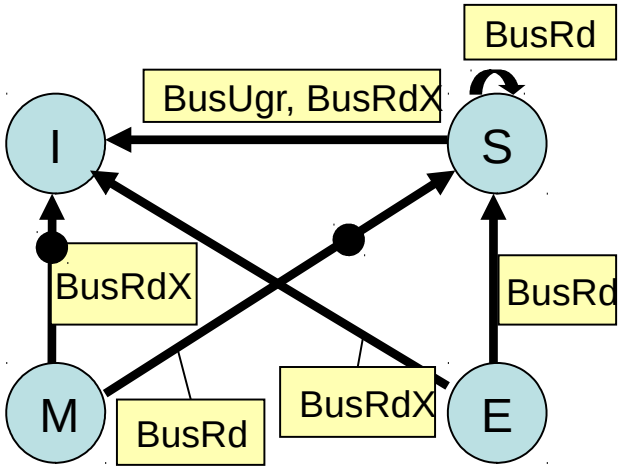
Example – MESI protocol



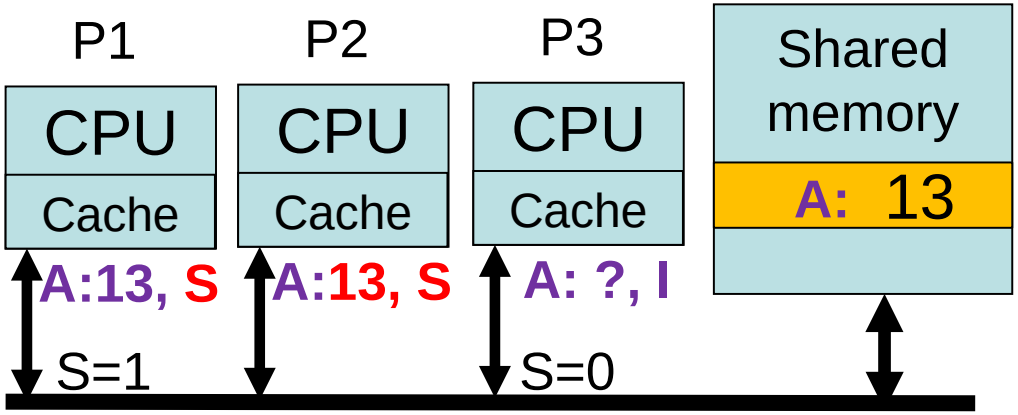


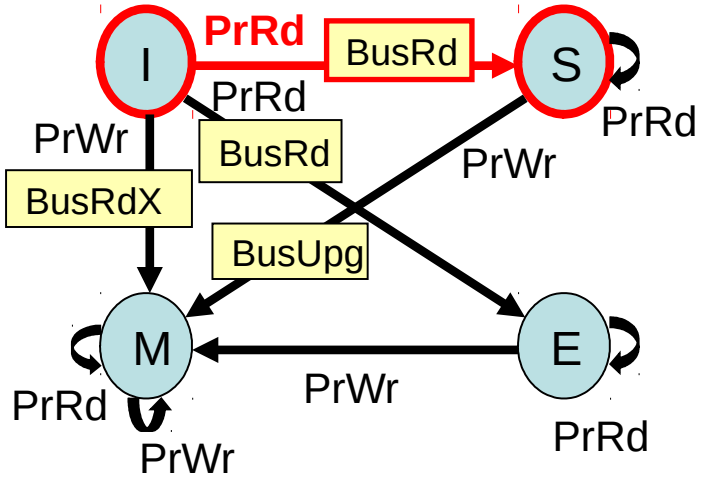
Processor P1 requests read from address A.

- P1 send PrRd(A) to its cache controller but 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.



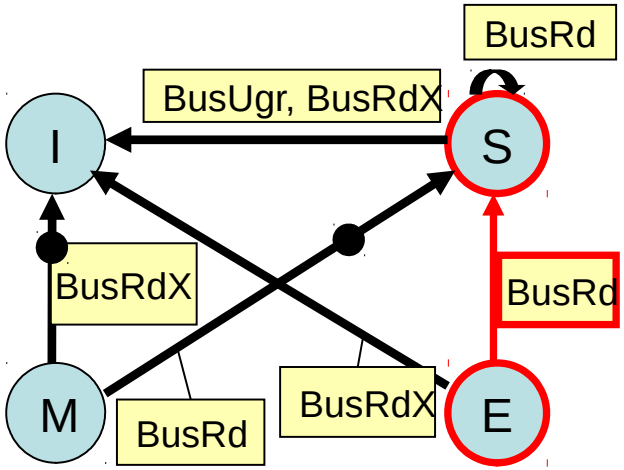
Example – MESI protocol



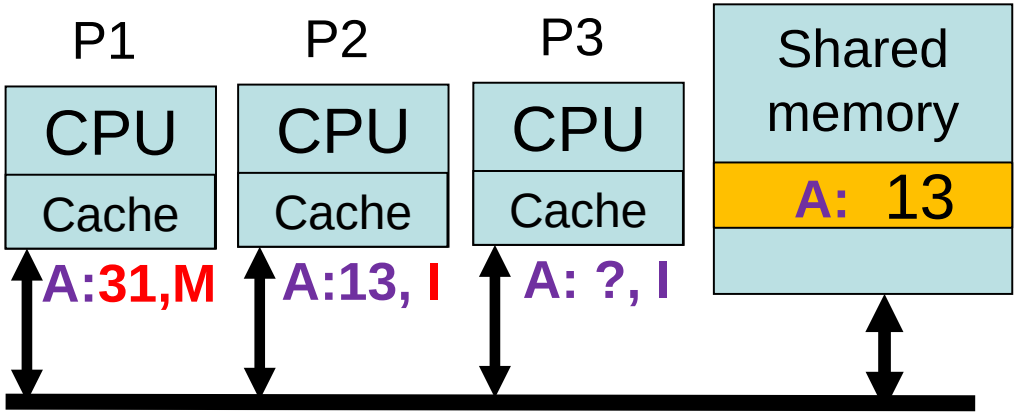


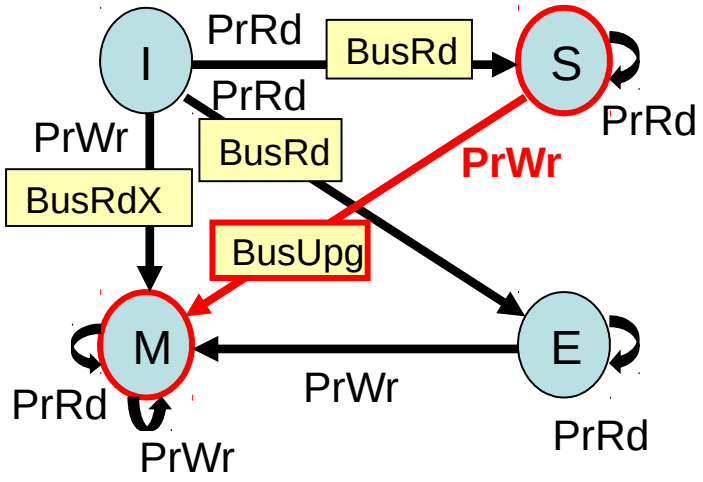
Processor P2 requests read from address A.

- P2 sends PrRd(A) to its cache but 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 (assersts signal S=1) aborts read request and delivers data from cache.
- Both processors/cache controllers advance to state S.
  (P1: E->S; P2: I->S)



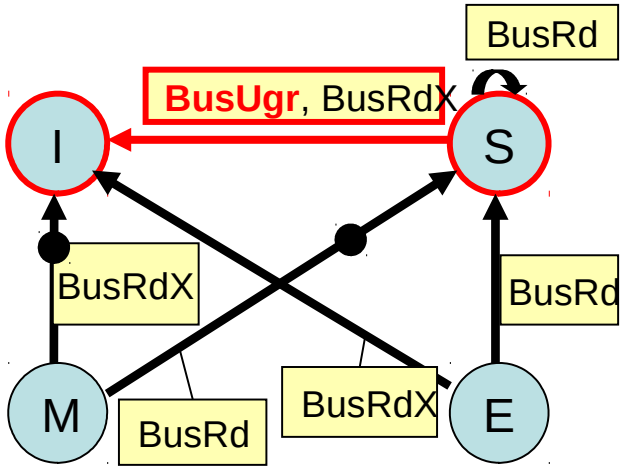
Example – MESI protocol



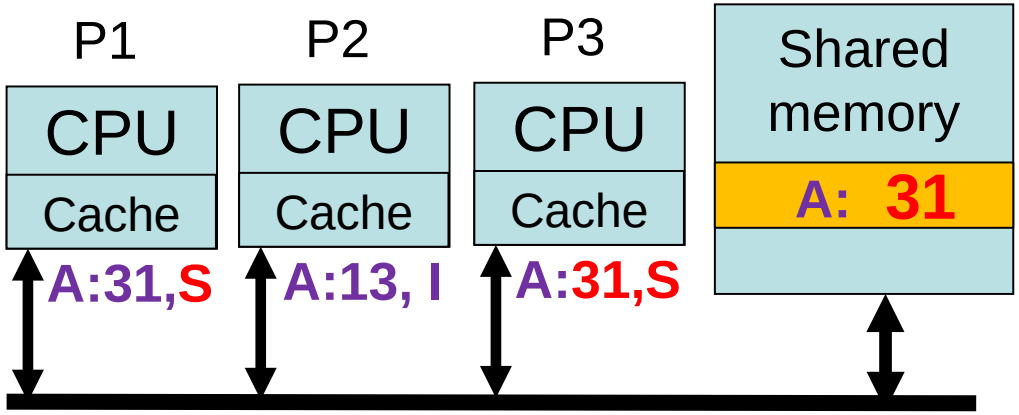


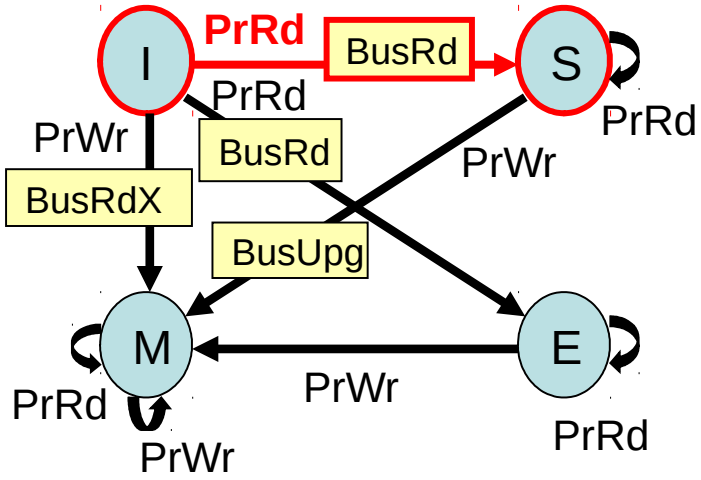
Processor P1 writes new value to address A. For example 31.

- P1 send PrWr to cache. But block is in the S state.
- Cache controller send BusUpgrade(A).
- All snooping cache controllers (in 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.



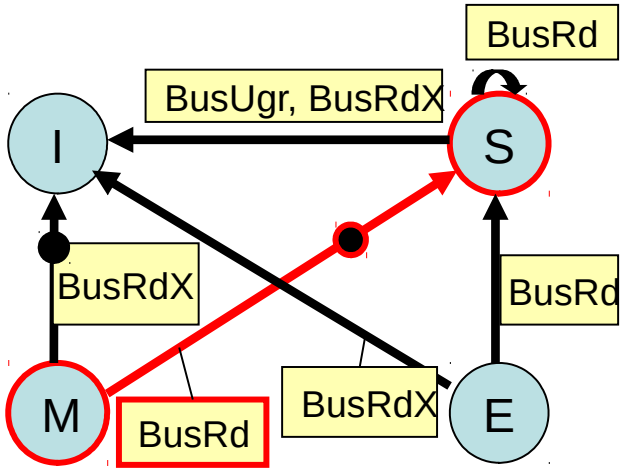
Example – MESI protocol



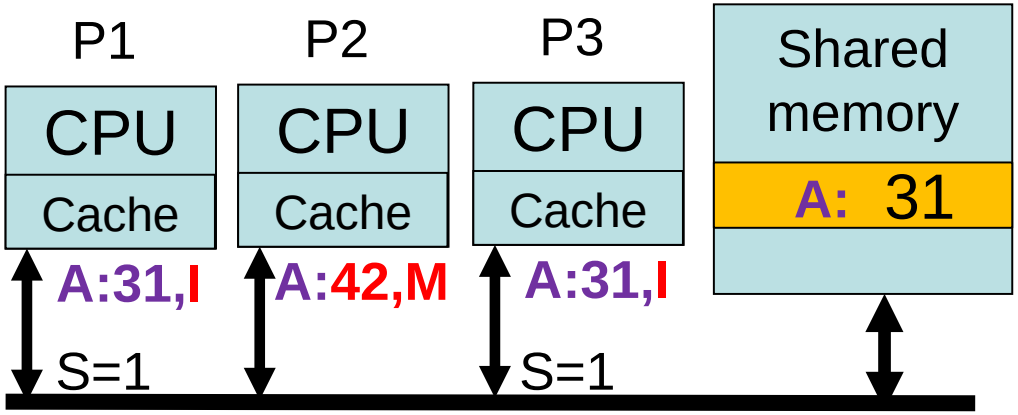


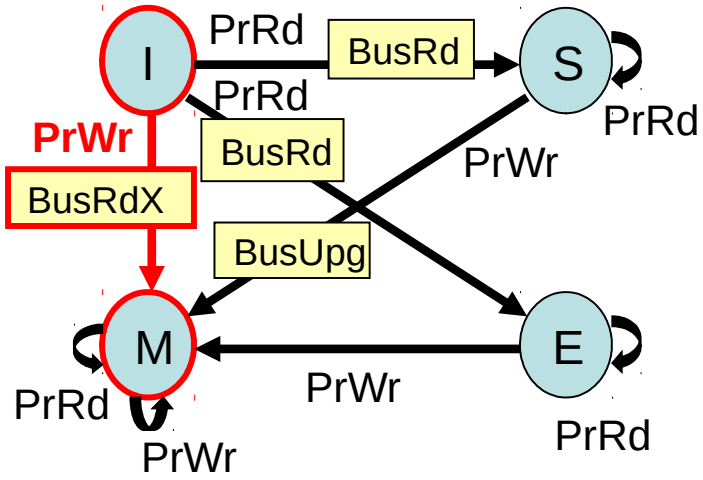
Processor P3 reads data from address A. Which value does it read?

- P3 sends PrRd(A) to its cache but line is Invalid. Cache controller responds by sending BusRd(A).
- Snooping cache controller of P1 indicates 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.



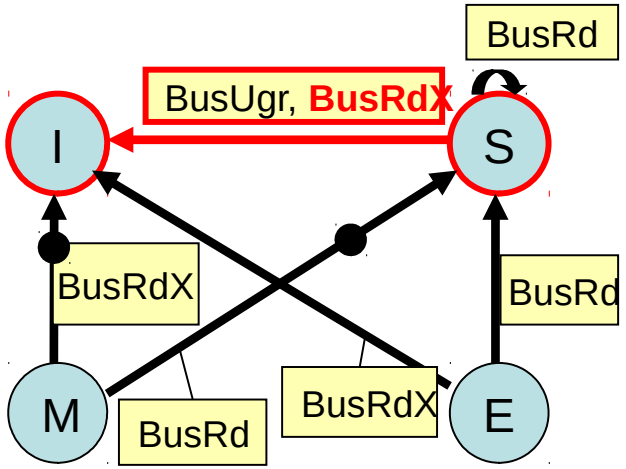
Example – MESI protocol





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

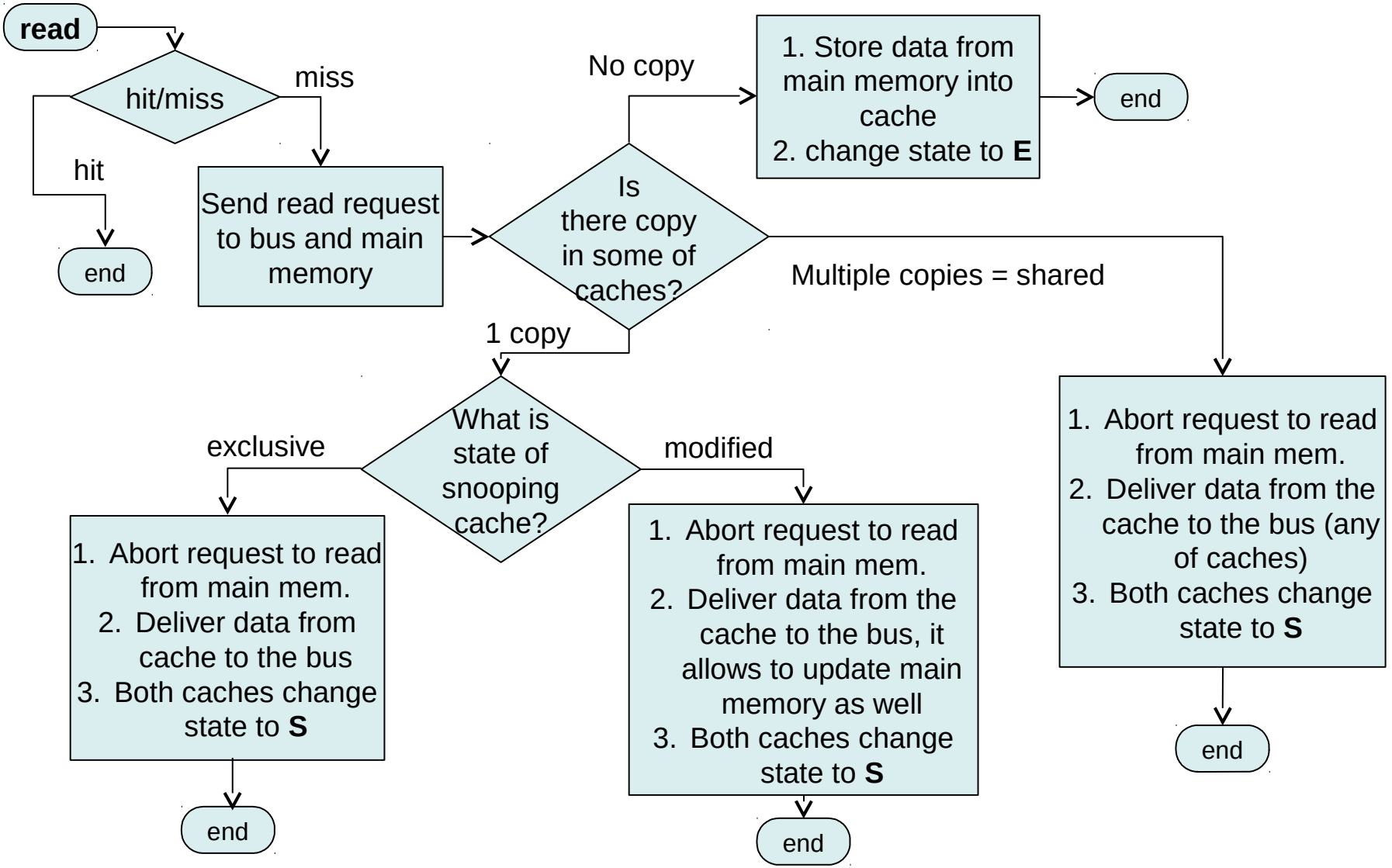
- P2 send PrWr(A) to its cache but line is Invalid. Cache controller issues BusRdX(A).
- Snooping controllers indicate that they have data copy. Abort memory read request and some of controllers delivers data.
- Both snooping controllers follow edge do S->I.
- Data requesting P2 follows edge I->M.
-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



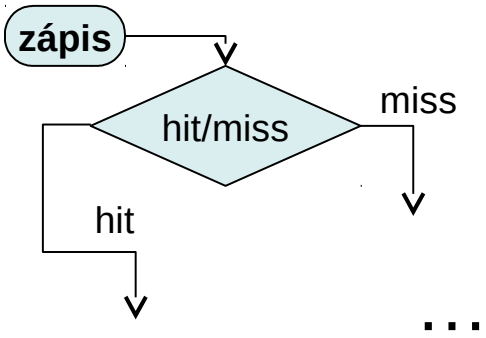
Remarks to simplify implementation

- If 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 occurs during Read and or Write events (memory access)
- Event is caused by
  - Cache controller local/connected processor activity/code execution (cache access), or
  - As result of successful bus snooping (address matching) of other processor initiated bus activity.
- Changes of cache-line state occurs only in the cache of corresponding transaction and cache-line address (index+tag+address) match.

Summary of previous slides – Read

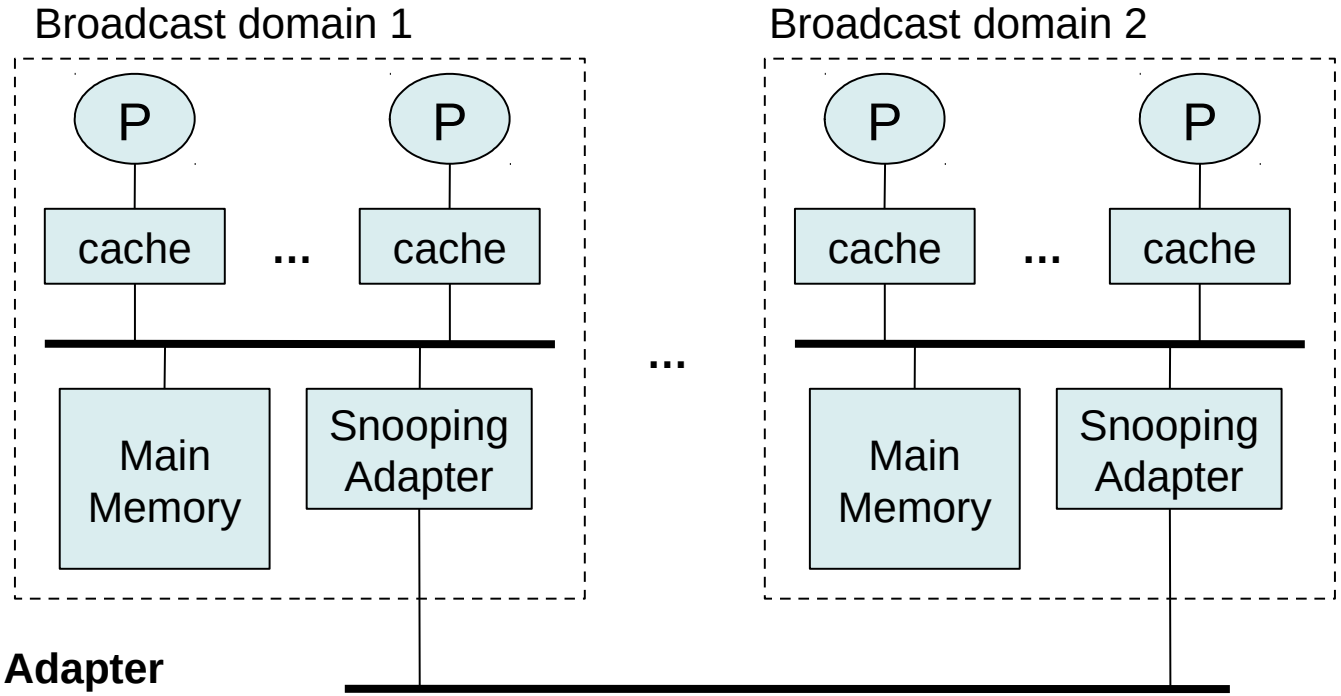


Summary of previous slides – Write



Broadcast extension/scaling for more processors

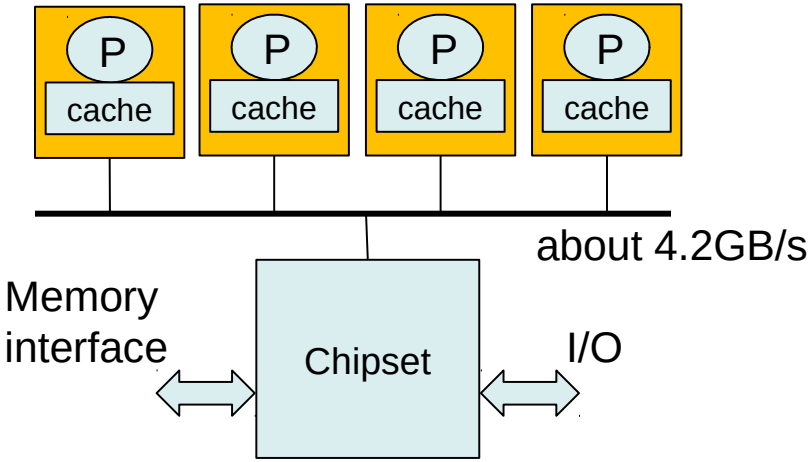
- Systems based on Broadcast (snooping) can be practically scalled to about 8-10 nodes where each node can be multicore processor today
- Option for more processors is hierarchical Hierarchical Snooping.
- But shared buses are generally replaced by point-to-point interconnection (on next slides) so snooping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



Snooping Adapter
Separates busses

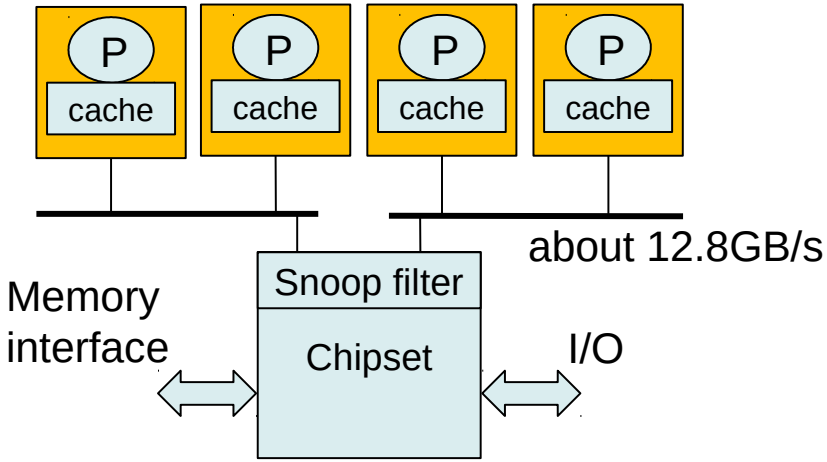
Development inside of processor as a chips/packages

Year 2004



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

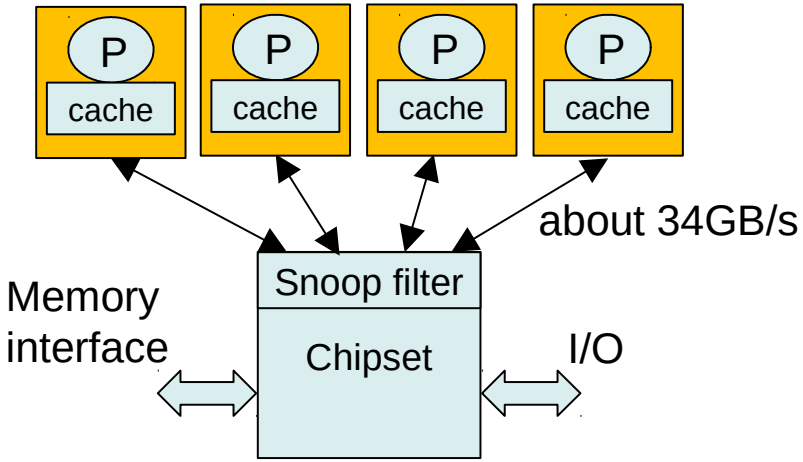
Year 2005



- 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 irreleveant transactions

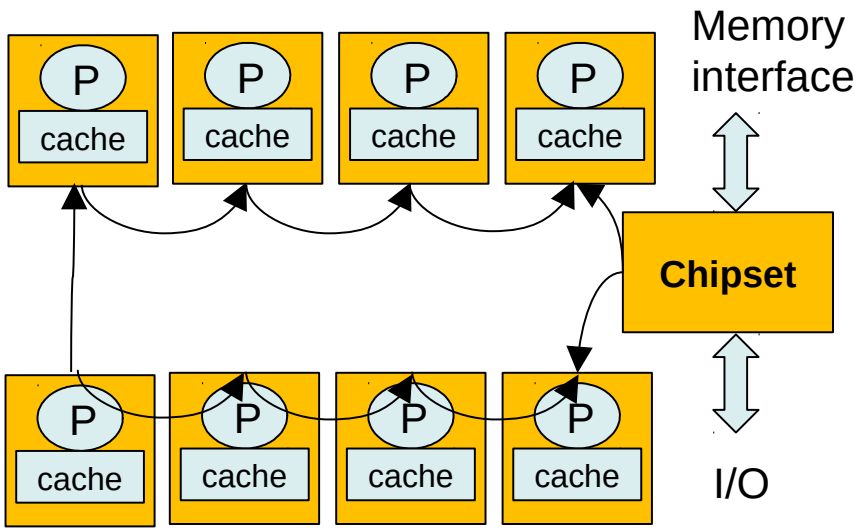
Development inside of processor as a chips/packages

Year 2007



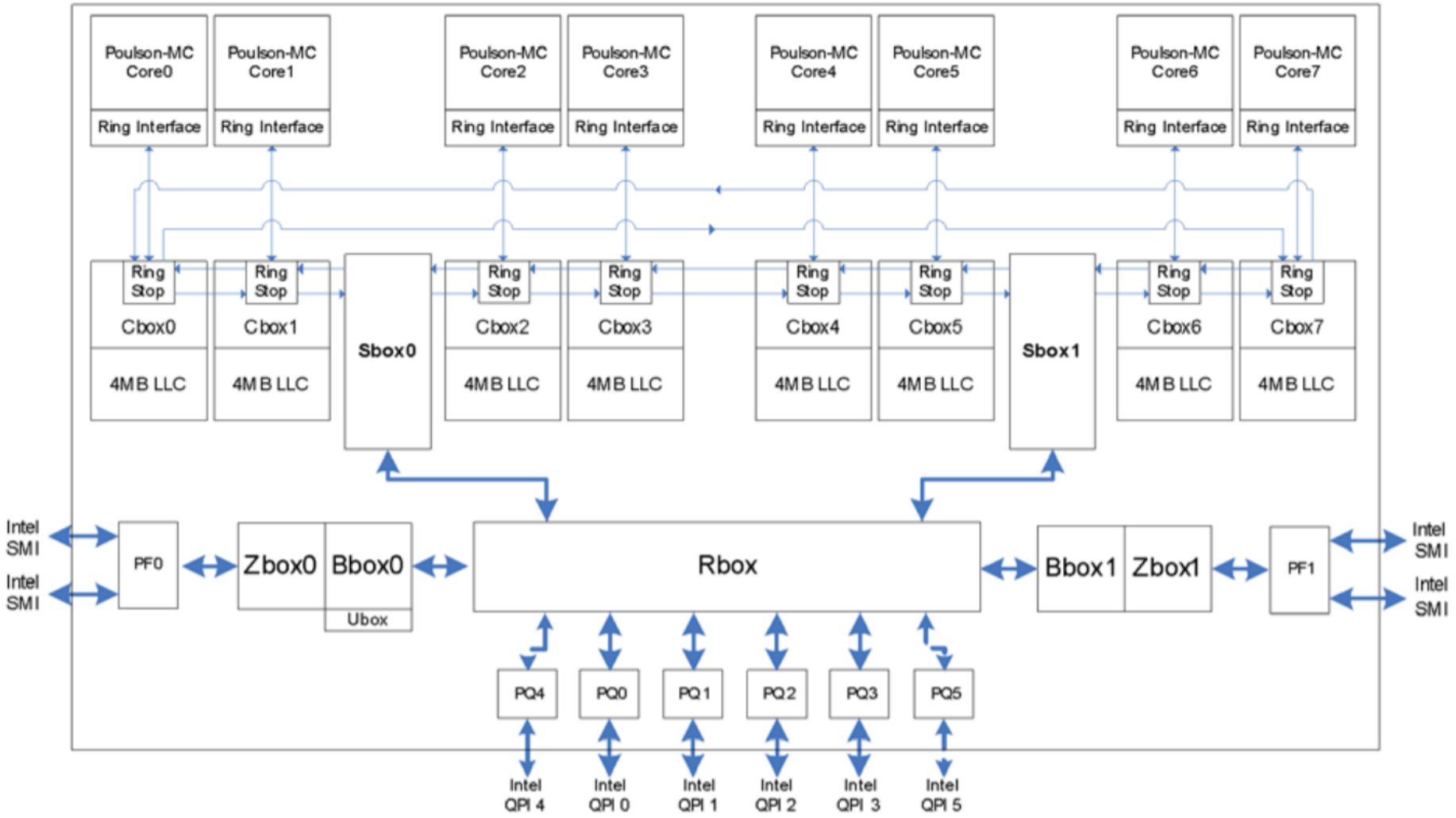
- Introduction high-speed point-to-point interconnects. But snoop filter becomes the bottleneck of the system. It is too centralized

2009 till today

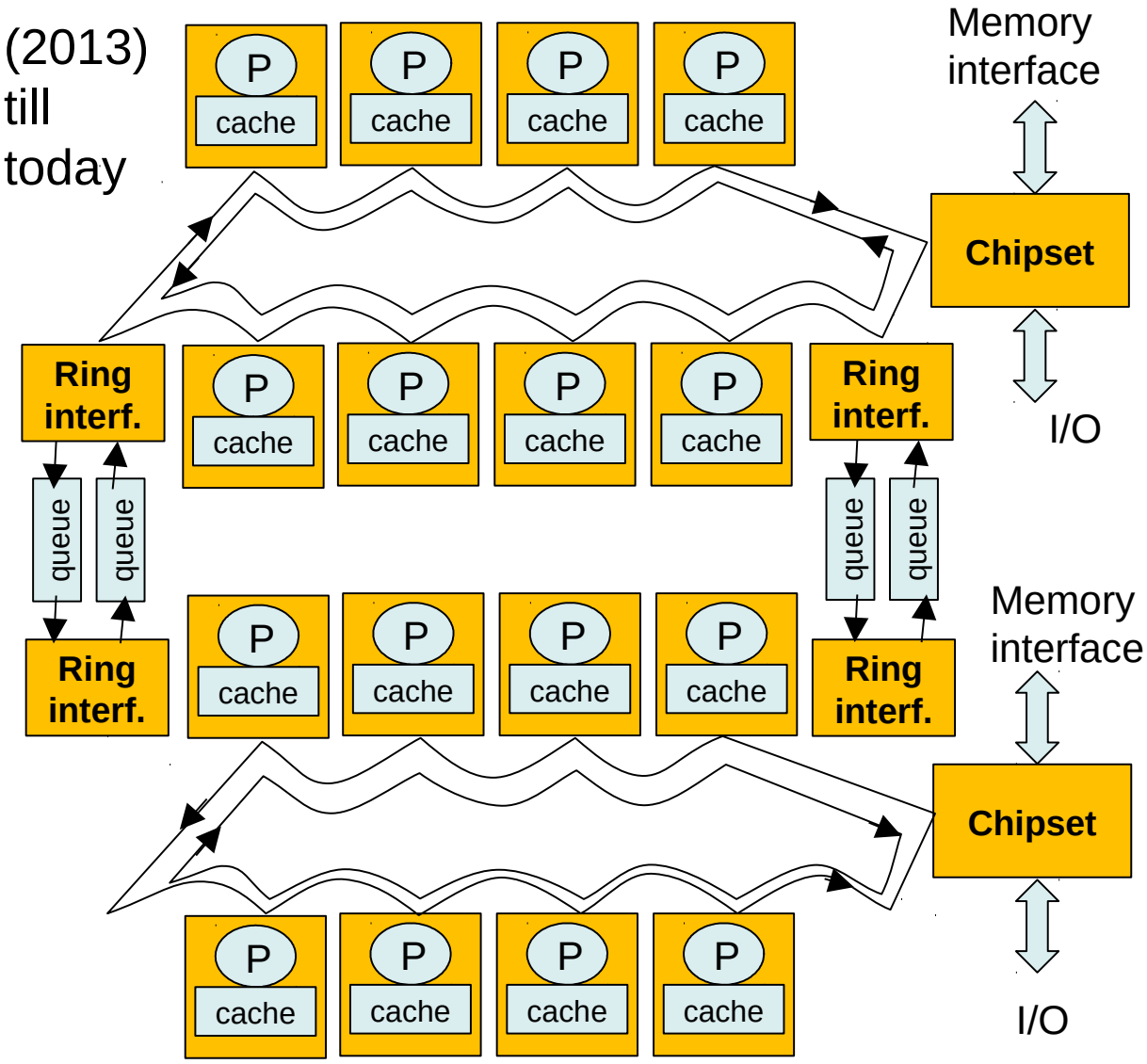


- Ring. Single direction ring (in the picture) – all meseges are delivered between nodes in same direction and node order is fixed. Two single direction links placed in opisite direction form bidirectional ring often used today. Ring routers can be simple – same as on roundabout which is leaved by car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

Example: Intel Itanium Processor 9500

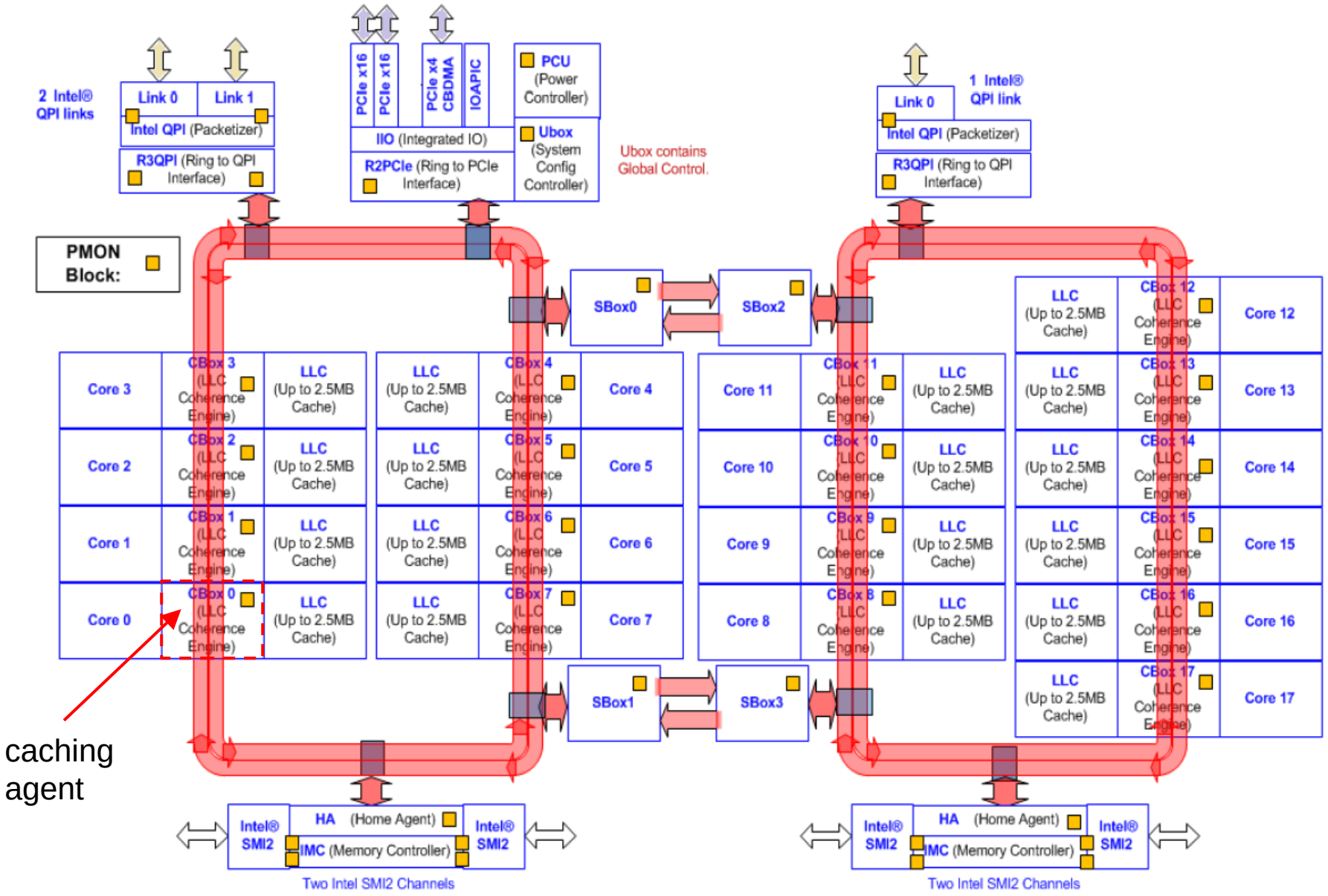


Development inside of processor as a chips/packages



- Interconnected rings. Even single package multi-processor becomes NUMA system.
- Ring offers fast point-to-point interconnection and removes 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 insers and or remove message thanks to distributed arbitration. But ring does not ensure global ordering of events (which is natural on 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)



Practical example: Broadwell-EP (Intel Xeon)

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

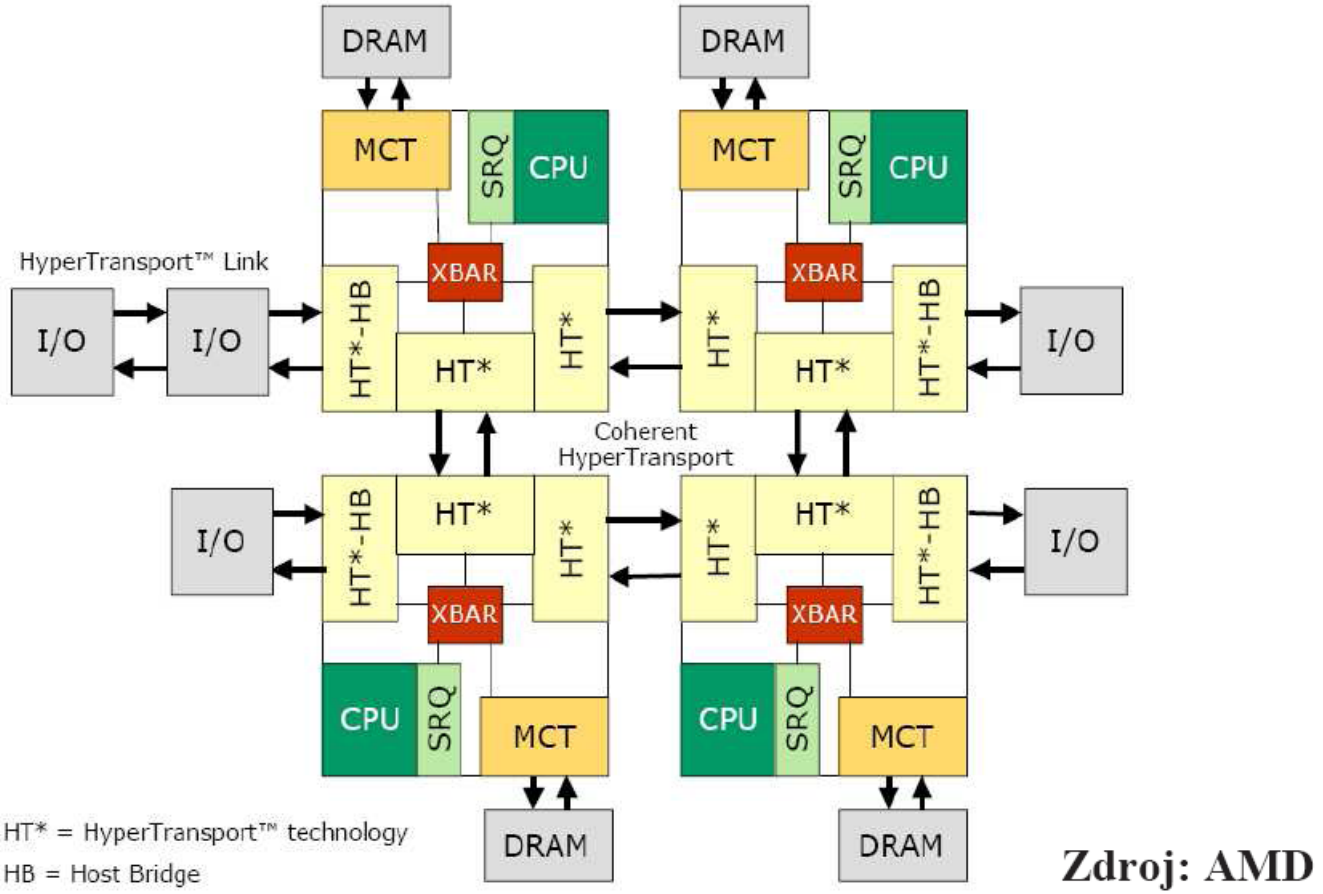
- **Directory Cache** – it is 14KB cache each HA (home agent). It stores 8-bit vector which inform which CA (caching agent) can deliver copy of cache-line. Directory cache is integrated in chip. Directory cache hit means whom we should ask to provide data for given address.
- **Directory**. Directory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

Remarks:

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

How to snoop without 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**:

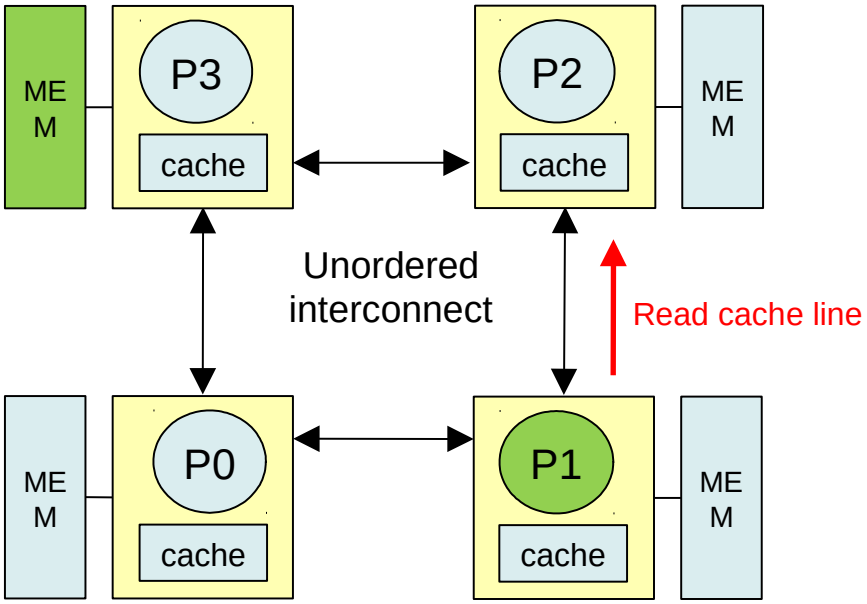


Zdroj: AMD

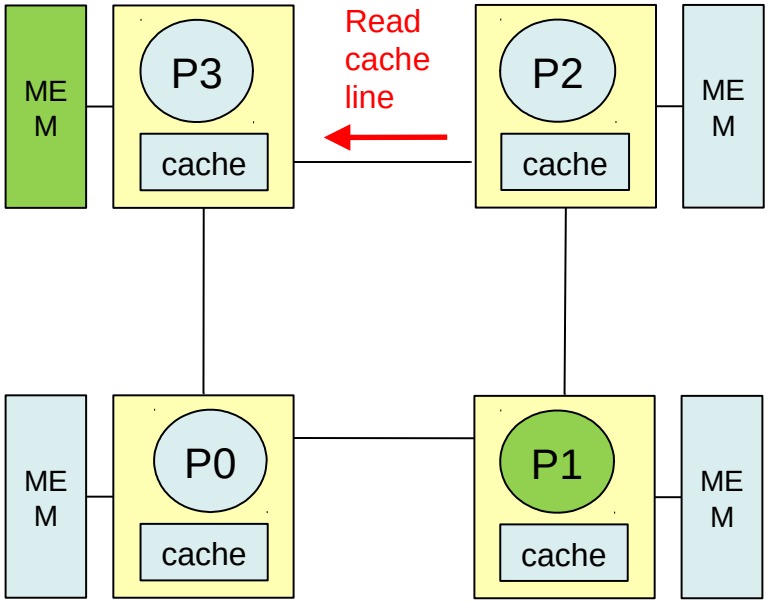
How to snoop without shared bus? Broadcast protocol.

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

Step 1



Step 2

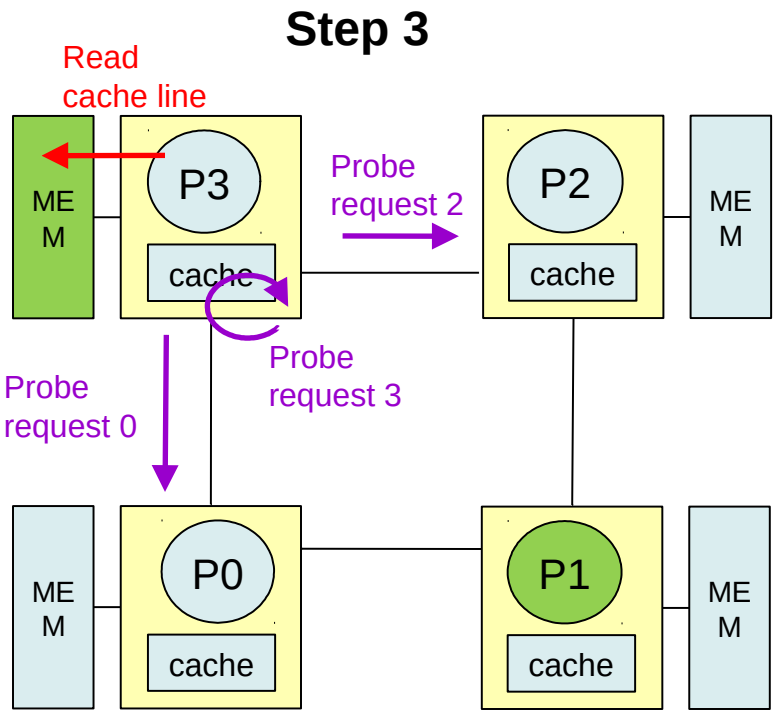


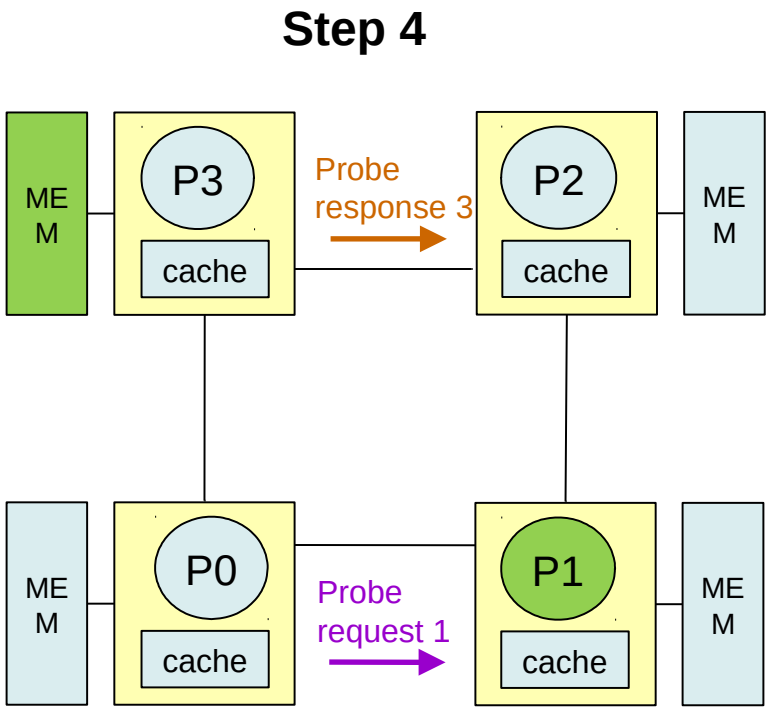
- Reaction on last-level cache miss: query sent to home node
- There memory controller decides order (of processing) of all queries to same cache line

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

How to snoop without shared bus? Broadcast protocol.

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



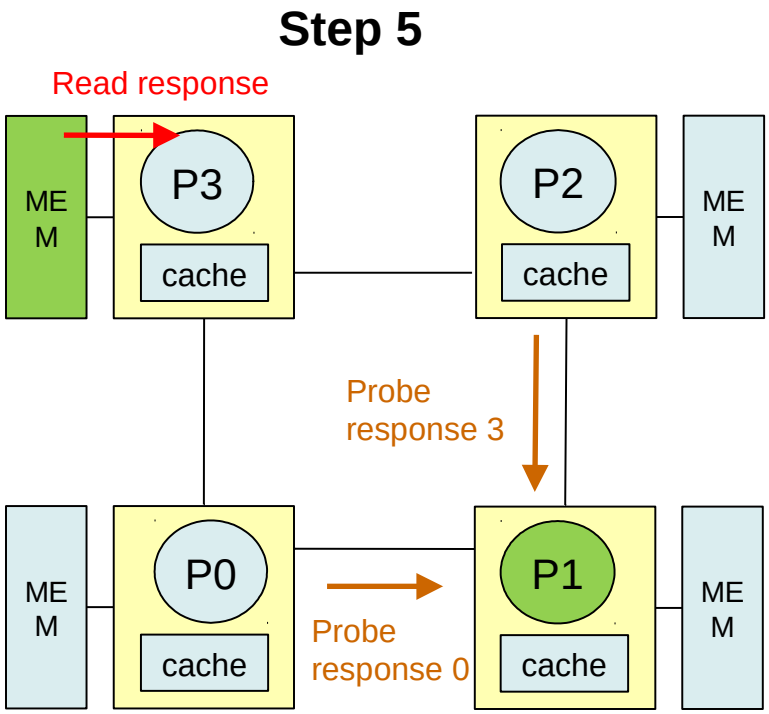


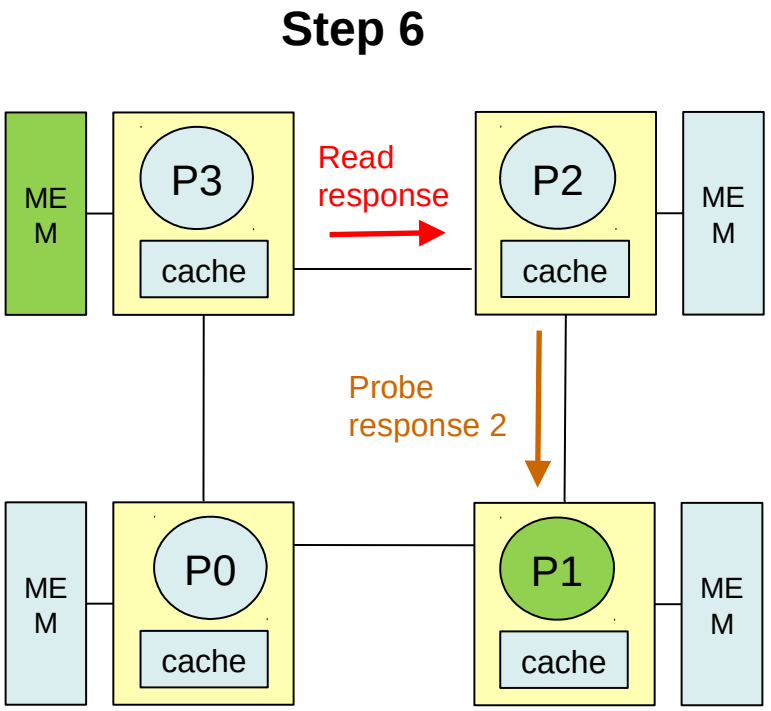
- When query reaches nome node and starts to be served , cache probes requests are sent to all processors and access to RAM is initiated in parallel

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

How to snoop without shared bus? Broadcast protocol.

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

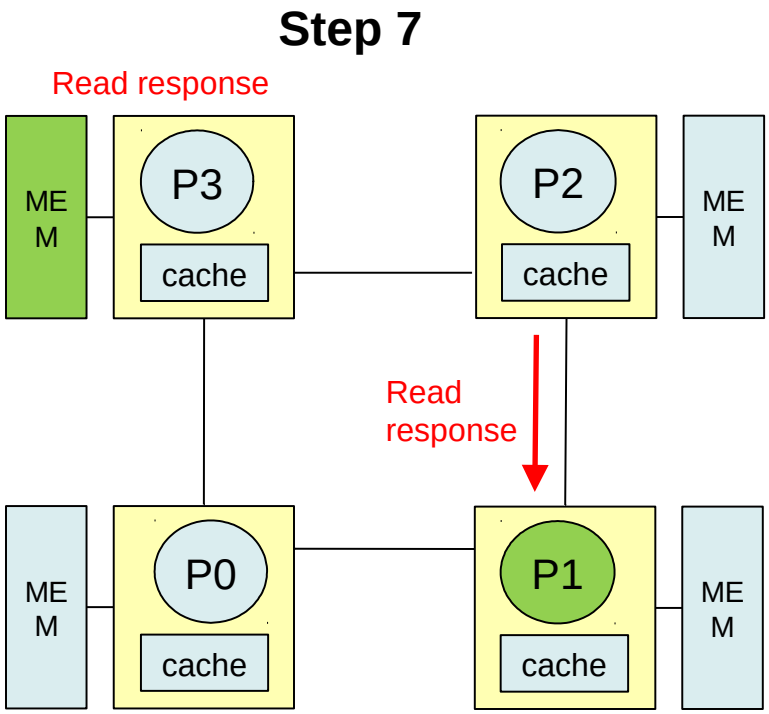


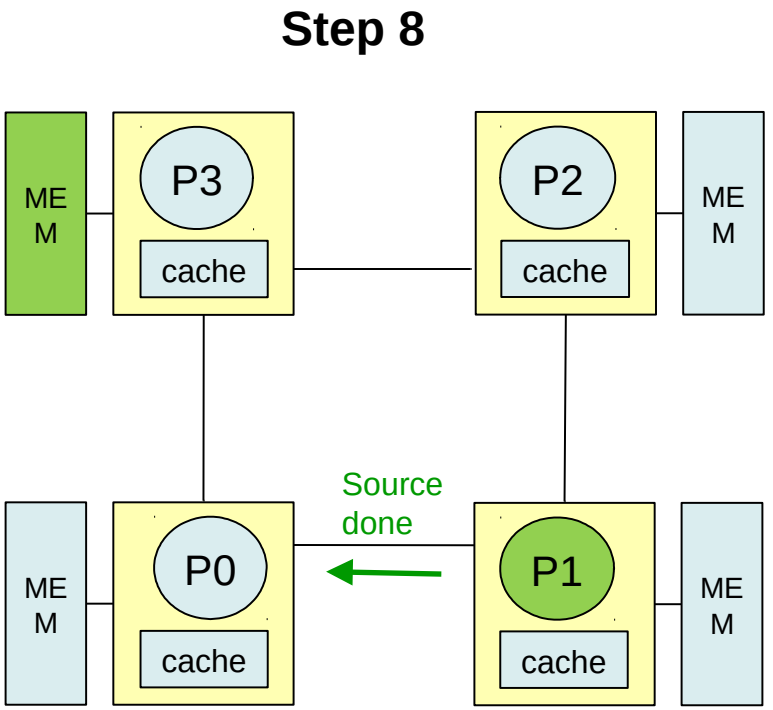


- Querying processor P1, collect responses from all processors in time as they arrive

How to snoop without shared bus? Broadcast protocol.

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

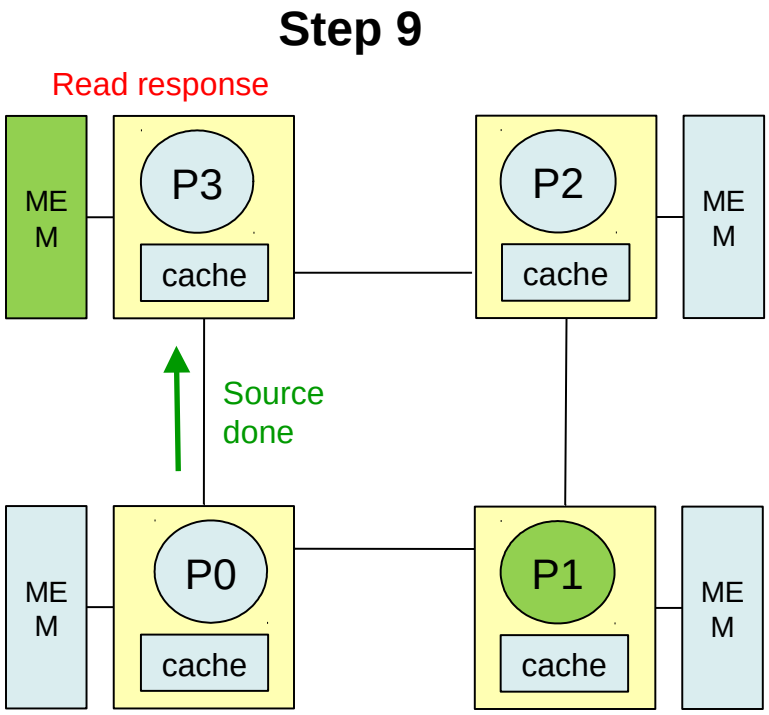




- As requesting processor receives all responses, it sends message to home node which informs that cache line request is handled then home node (memory controller) can service next request to the same cache-line.

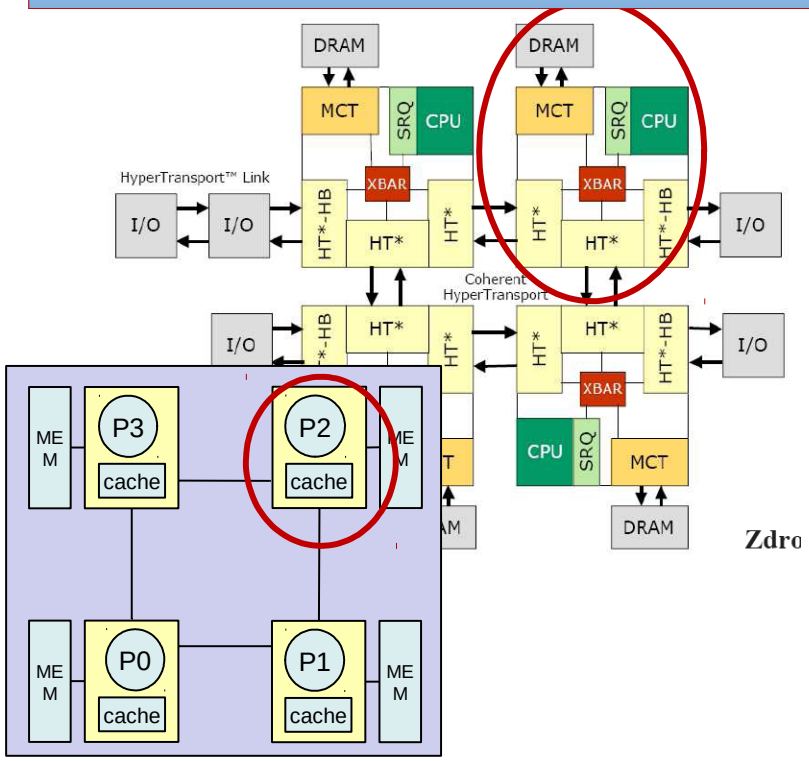
How to snoop without shared bus? Broadcast protocol.

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

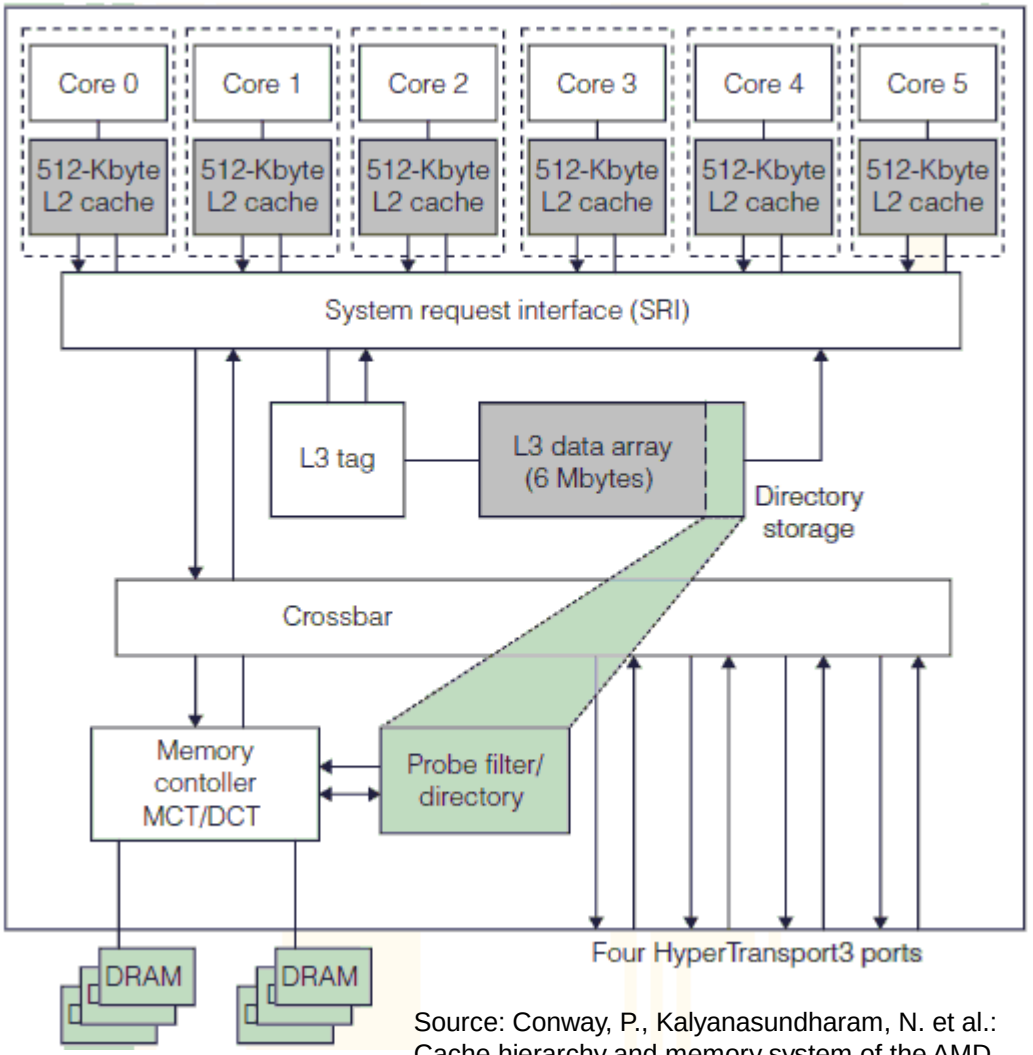


- Notice complexity of communication for single read?
- Next generation of AMD processors included **HT assist** (HyperTransport Assist directory protocol) – about 2010 year
- HT assist uses **directory** (in home node) which maintains information which cache-lines are cached in another CPUs
- This allows reduce inte-processor communication significantly. Instead of 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 line
- broadcast probe – fallback case

This is case of AMD Quad – Coherent HyperTransport



Zdro



- 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 bus is not enough and broadcast to all is limiting

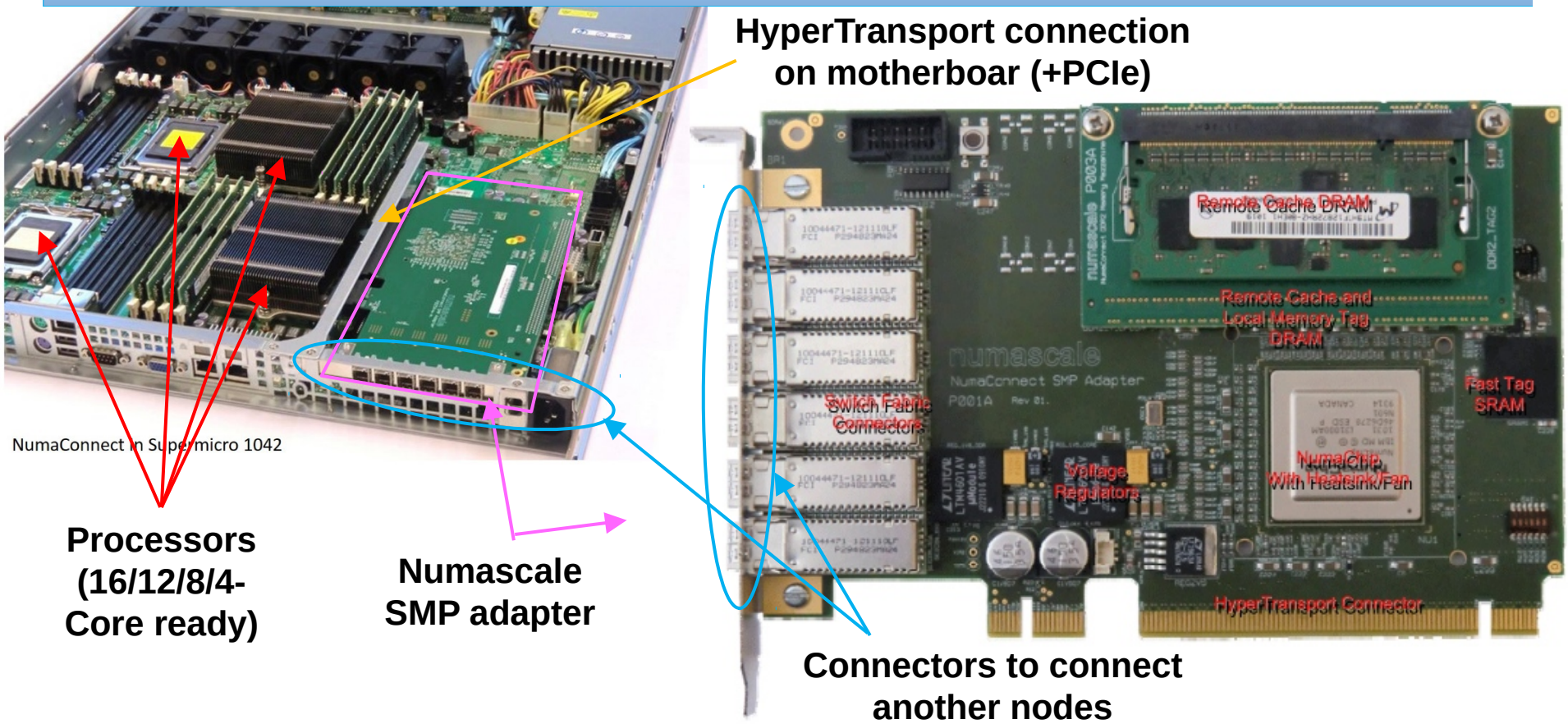
- Sending information to everyone else (or listening to all) is not a scalable solution ...
- We noticed that removing buses (where monitoring/snooping is not a problem) and by change to the point-to-point interconnection between CPU cores (nodes) and the increase in the number of cores, the solution 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.

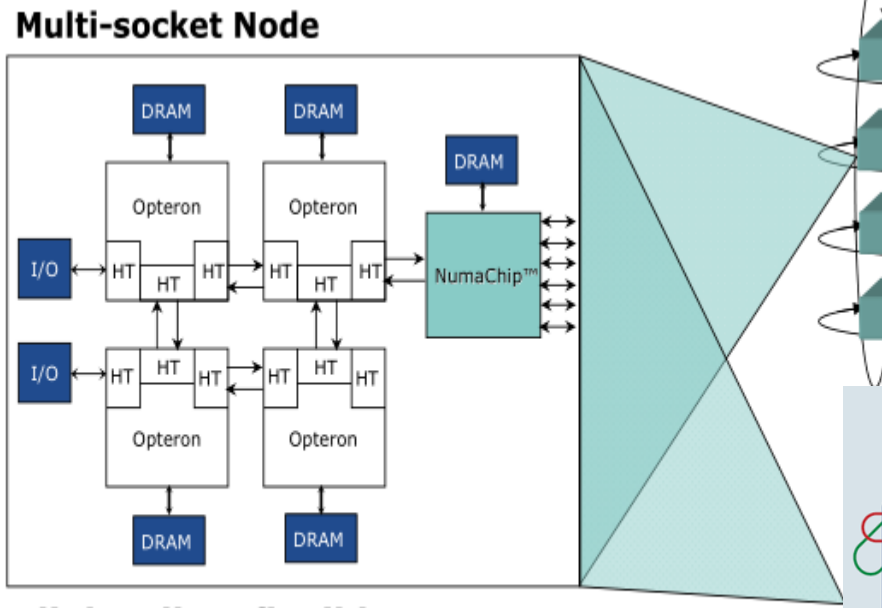
- This approach is suitable for interconnection of hundreds and or thousands of processors.
- Directories use is not new idea. The new is only that their use has spread even to todays common desktop CPUs.

Example from practice – Numascale

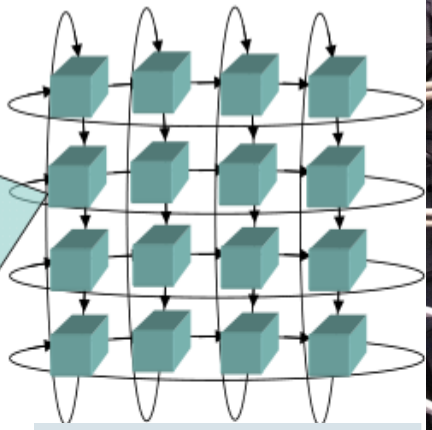


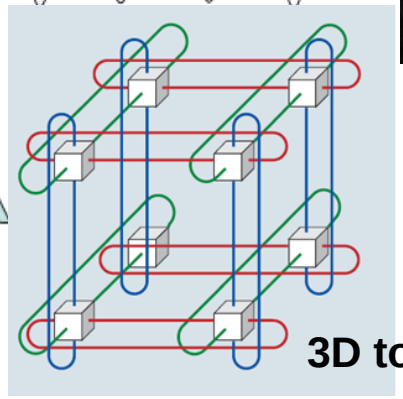
- Support cache coherence in HW (directory based)
- Max. 4096 nodes. One node: 4 processors (on left-top picture)
- node: 16 DIMM sockets x 32 GB = max. 512 GB
- Max. 256 TB total memory. Limited by 48-bit physical address-space of CPU
- Example: 4GB Cache, 8 GB Tag (supports 240 GB Local Node RAM)

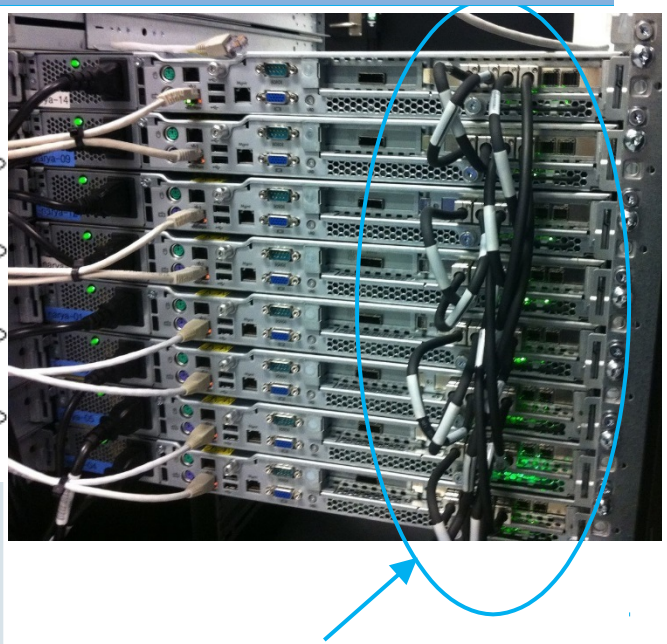
Example from practice – Numascale – Nodes interconnection



6 links allow flexible system configurations in multi-dimensional topologies





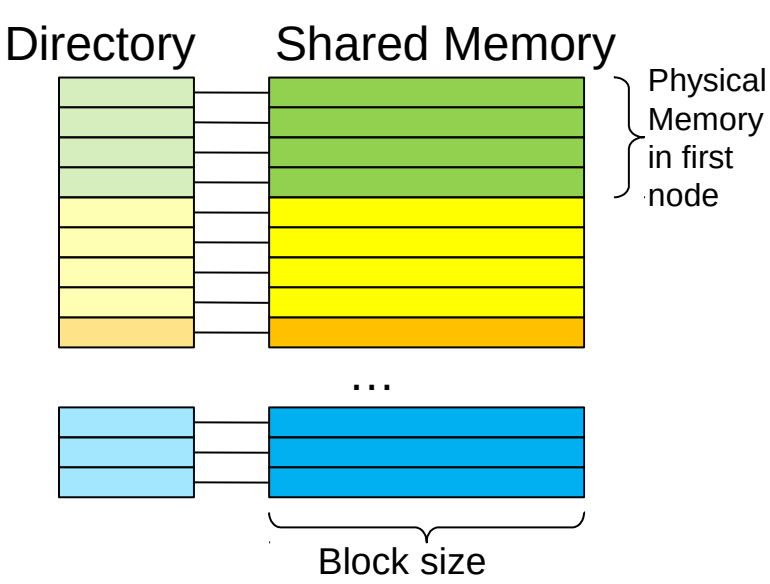


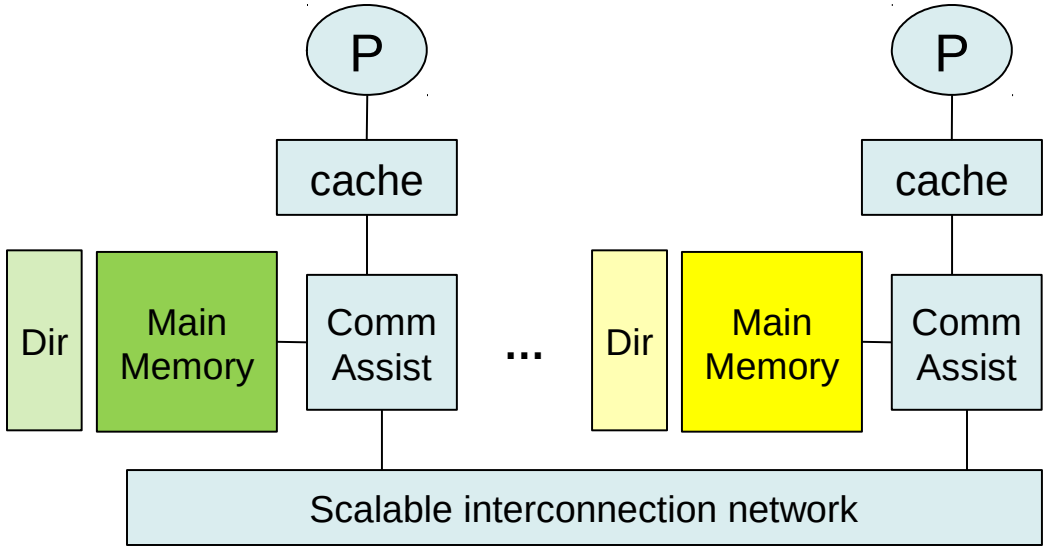
Nodes interconnected by 2D torus

- 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
- Program, which is written to run on single node, can run on 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 Boolean vector of length n+1. If bit i (i=1,2,…n) is set then corresponding (i) cache holds copy of the line. Bit 0 indicates if the line is in clean or disrty state (only one other bit can be set for dirty state = line is only in one cache)
- In NUMA system, each node implements only part of the directory with informations corresponding to that lines which are stored in its memory = **home node**, the rest are **remote node** for this part of memory.
- For cache miss case, request is send only to 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 capacity of shared memory, but for 64 processors it is 13% (65/(64*8)=0.127) . Bad scalability for thousands processors.

Example of Full directory realization

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

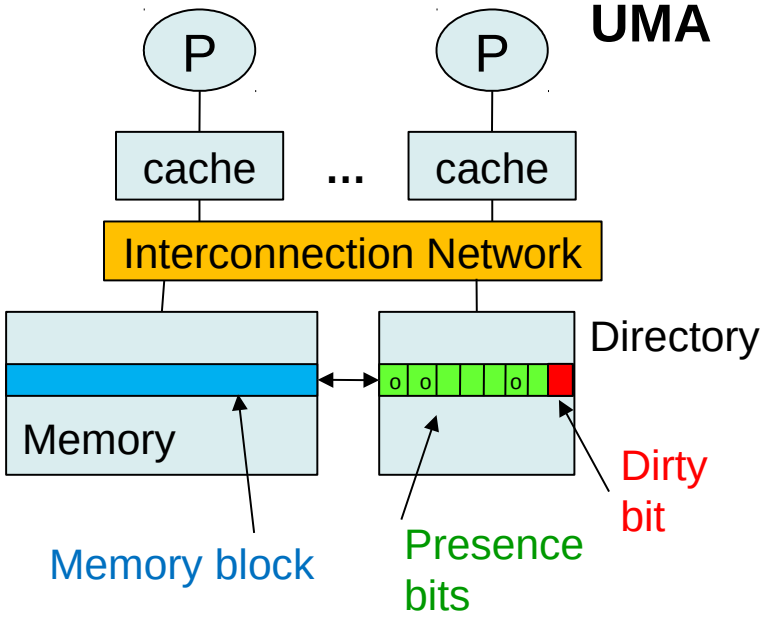
Read block by processor „i“ (after read miss):

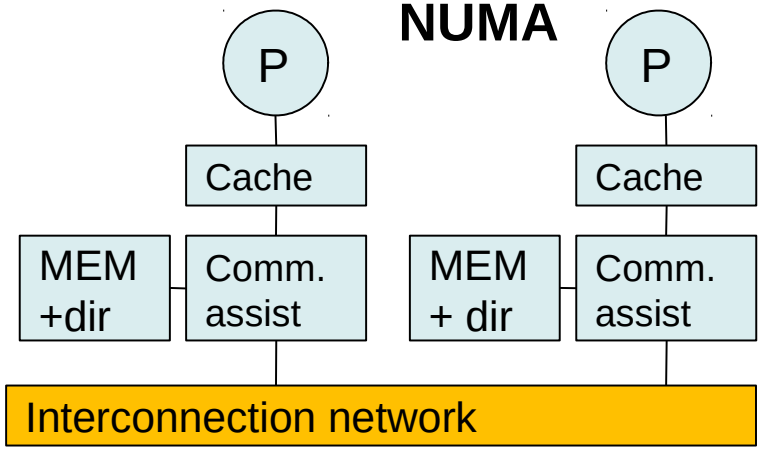
- If dirty-bit OFF the { read from main memory; set p[i] ON; }
- If dirty-bit ON then { request/stole dirty line from corresponding processor, update memory; set dirty-bit OFF; set p[i] ON; send data to processor i;}

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

- If dirty-bit OFF then { send invalidation to all shared copies; send data to ii, clear all p[j] in directory and set only p[i] to ON; dirty bit ON; }
- If dirty-bit ON then {acquire (with invalidation) block from corresponding processor; its p[j] bit clearr; 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 cahe has: MESI, MOESI, another state corresponding to coherence solution between multiple cores/processors on given node with common L3 cache.





Full directories use

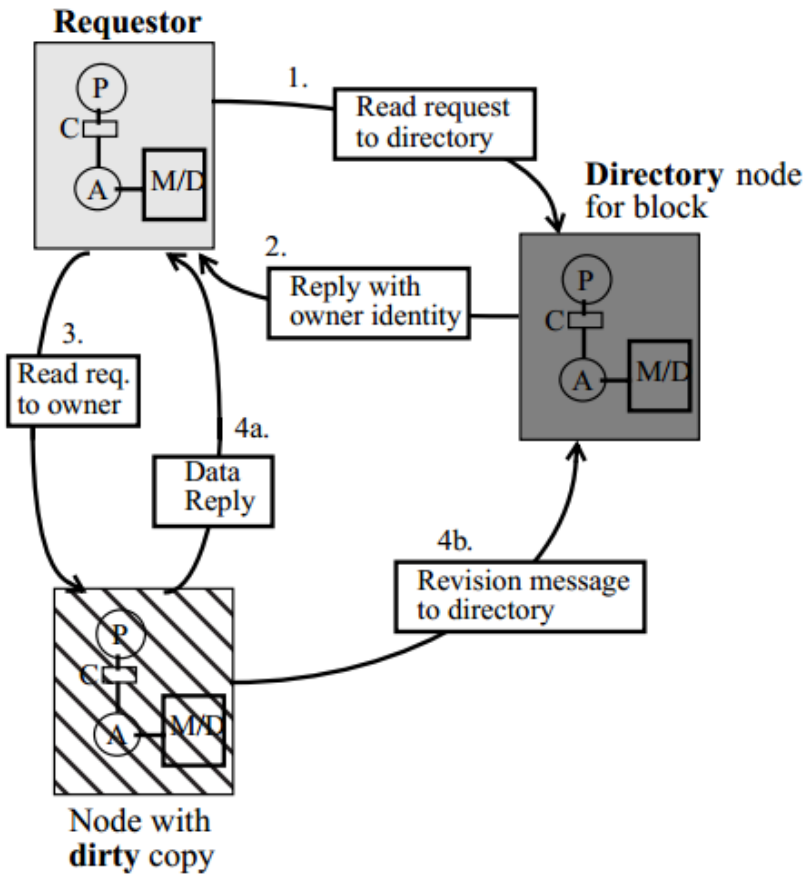
- CC-NUMA with Directories is solution for large scale systems – Directory Based CC-NUMA (Cache-Coherent NUMA).
- Directory and memory are distributed between nodes.
- Example SGI Origin2000 – 512 nodes x 2 processors = 1024 x MIPS R10K
- Idea is based on the fact that information about state of each block is available (maintained). This information is stored in directory. Broadcast are not necessary for such case and limited number of point-to-point transactions is necessary for each miss.
- **Home node** is that node which memory contains requires data, other nodes are **remote nodes** for that address range.
- **Numebr 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

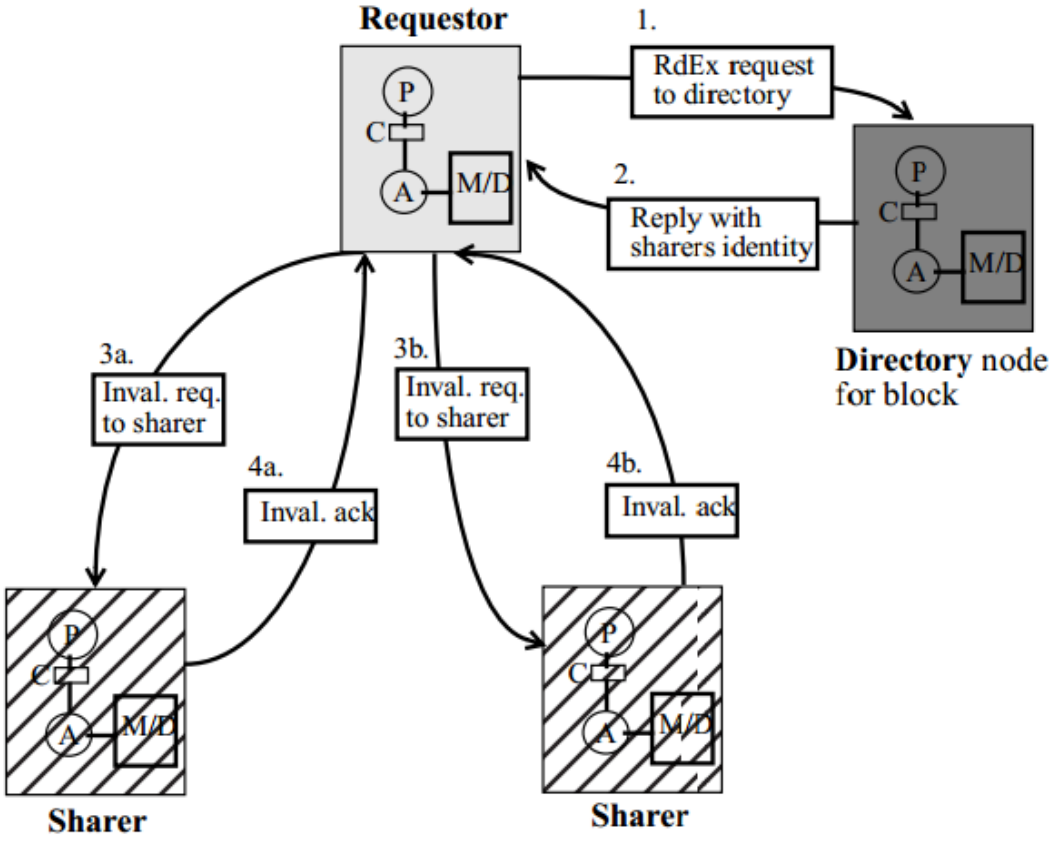
All of them related to each block in memory or cache:

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

More detailed directory supported operations



(a) Read miss to a block in dirty state



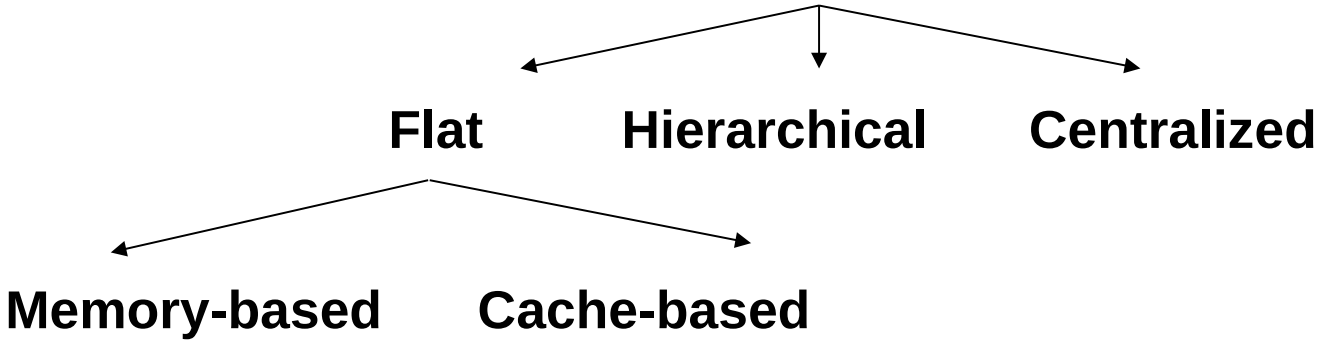
(b) Write miss to a block with two sharers

Directory Storage Schemes

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

- The most common is **Flat Directory** where this part of directory is placed on fixed location – usually on home node. It can be obtained/derived directly from block address (address from which CPU wants read or write)
- Other option hierarchical directory where meory is distributed between nodes usual way but 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 childs have or do not have copy of the block. Cache miss is then realized as wanking thro the tree in direction to parents. In the practice, tree nodes are distributed between system ndes (processors) and each miss generates usually multiple transactions between system nodes before required information is found.
- Centralized directory – advantage – single place to send querries – can be used only for small systems – example is coherency maintenance inside multi-core processors (in the fact inclusive cache hierarchy fall between central and tree solution). Such node can be member of larger system.

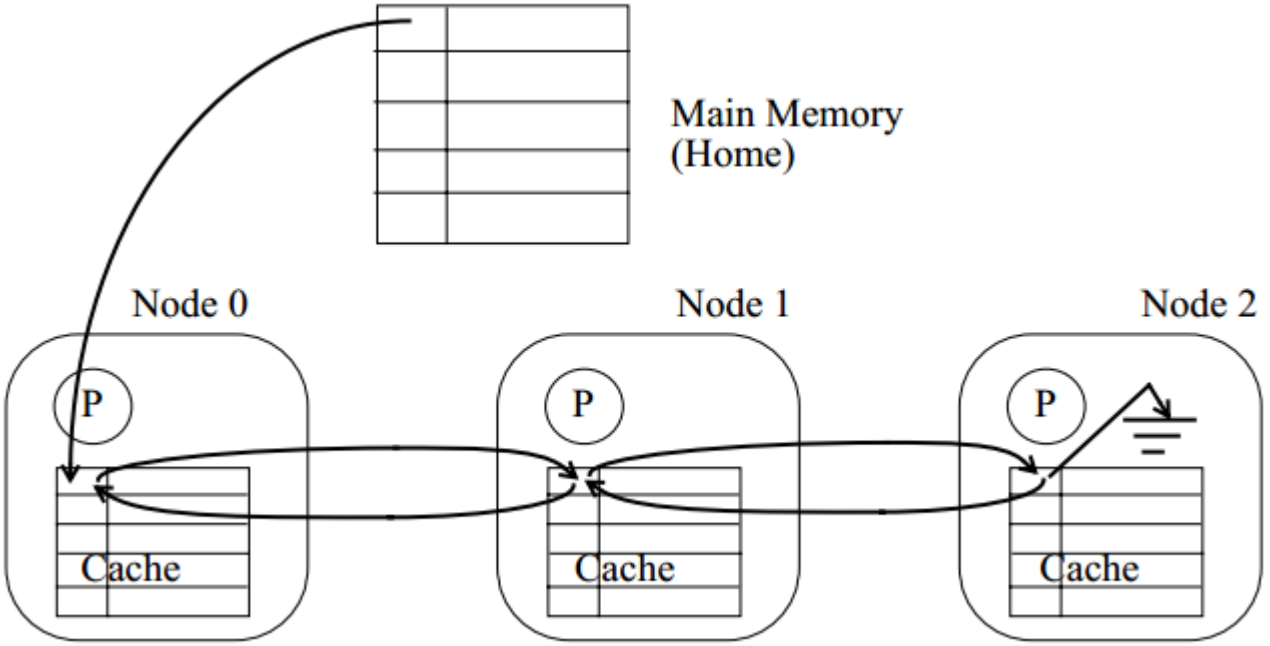
Directory Storage Schemes



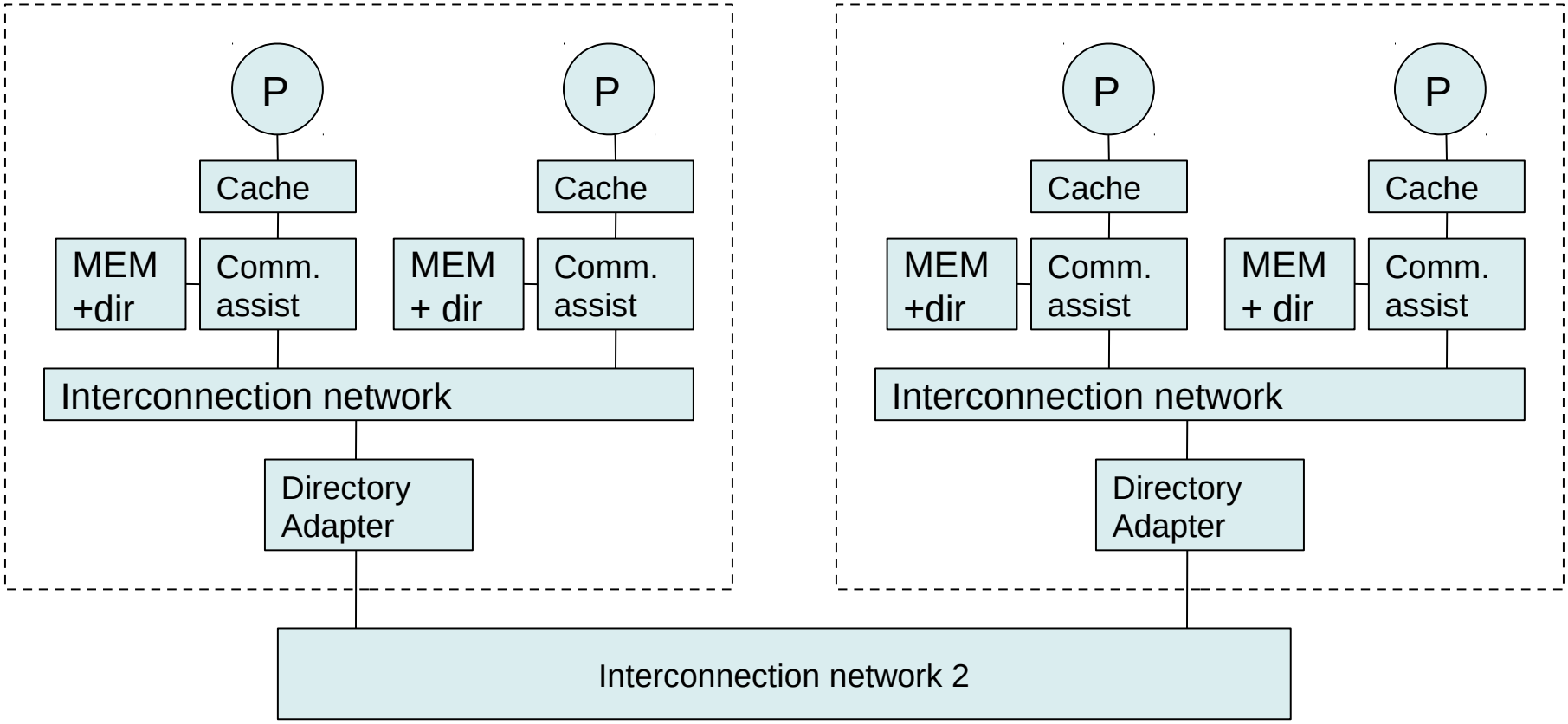
- Flat directory - **Memory based** – record for dirty node or all sharing nodes is kept in home node (record in memory). Example is discussed full-directory solution. An alternative is solution where remote nodes (processors) are recorded by their number instead of bitmap (number of sharing nodes is then limited – usual situation). System has to be prepared for situation when number of remote nodes requesting single block is above limit (example solution is forced invalidation in remote cache on oldest age basis …)
- Flat directory - **Cache based** – record in home node does not keep information about all sharing nodes but only pointer to the first sharing node is kept (plus state bits). Record 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 ove the list. Cache which contains copy of the block stores pointer to next and previous node s well.

Distributed directory – bidirectional linked list of sharing nodes

- IEEE standard SCI
- **Scalable Coherent Interface**
- 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-Direcory, Directory-Snooping

Summary and conclusions

- 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 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 significan for development of scalable solutions using multiprocessors.
- Debugging and performance tuning is not easy.
- Large scale multirocessor systems typically ensure memory coherence in individual compuational nodes (more multi-core CPUs) – OpenMP. Data echange 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