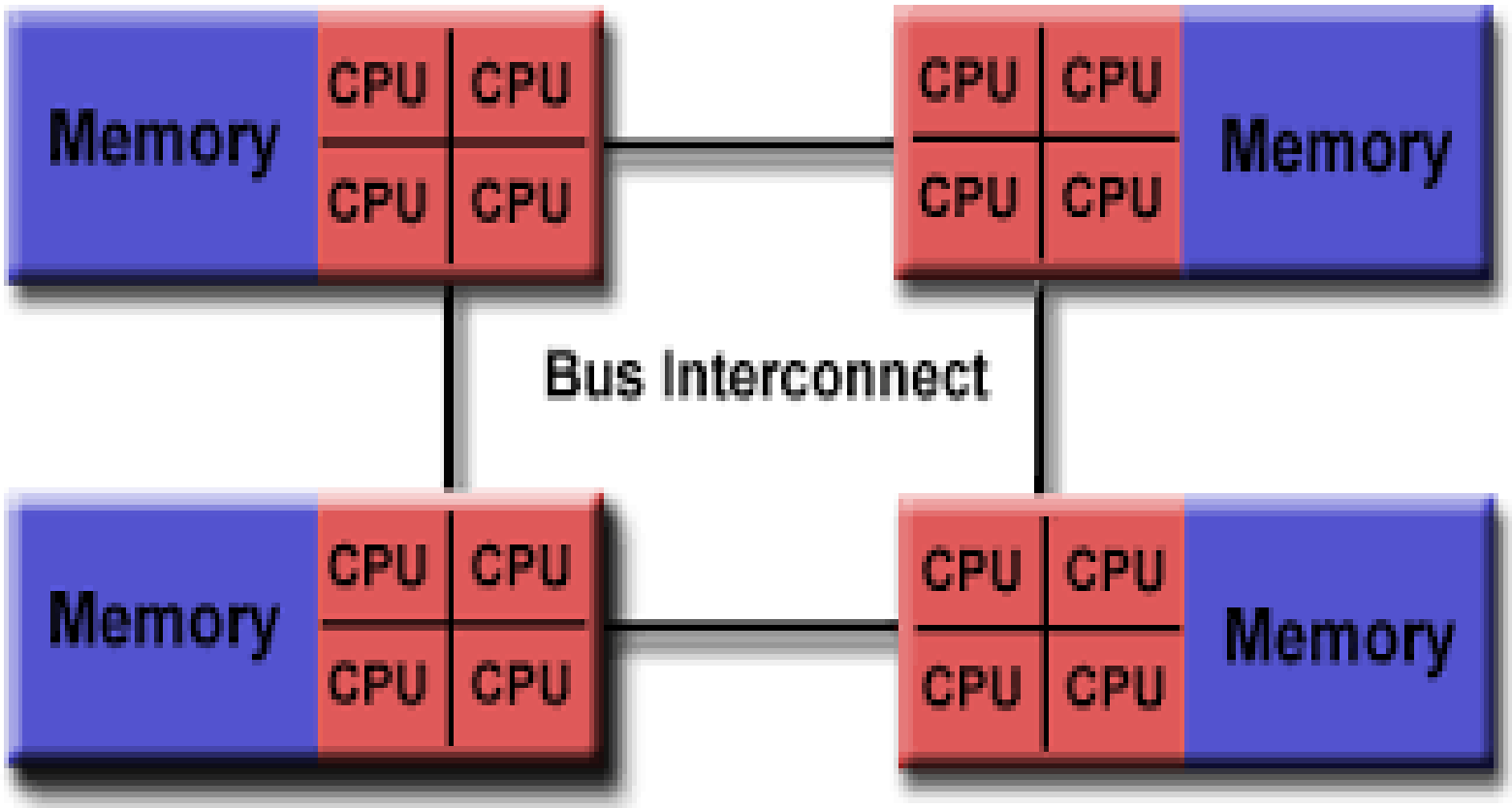
1)NUMA (Non Uniform Memory Access)-Module 3-
2)Inclusion-Module 3-pages723-724
3)Synchronization – Module 4- pages694-..
4)Memory Consistency-Module4- pages708-..
5)Cache Coherence Protocols(Patterson)-Module4-pages 374…

Above page references are from
Patterson 2nd Edition

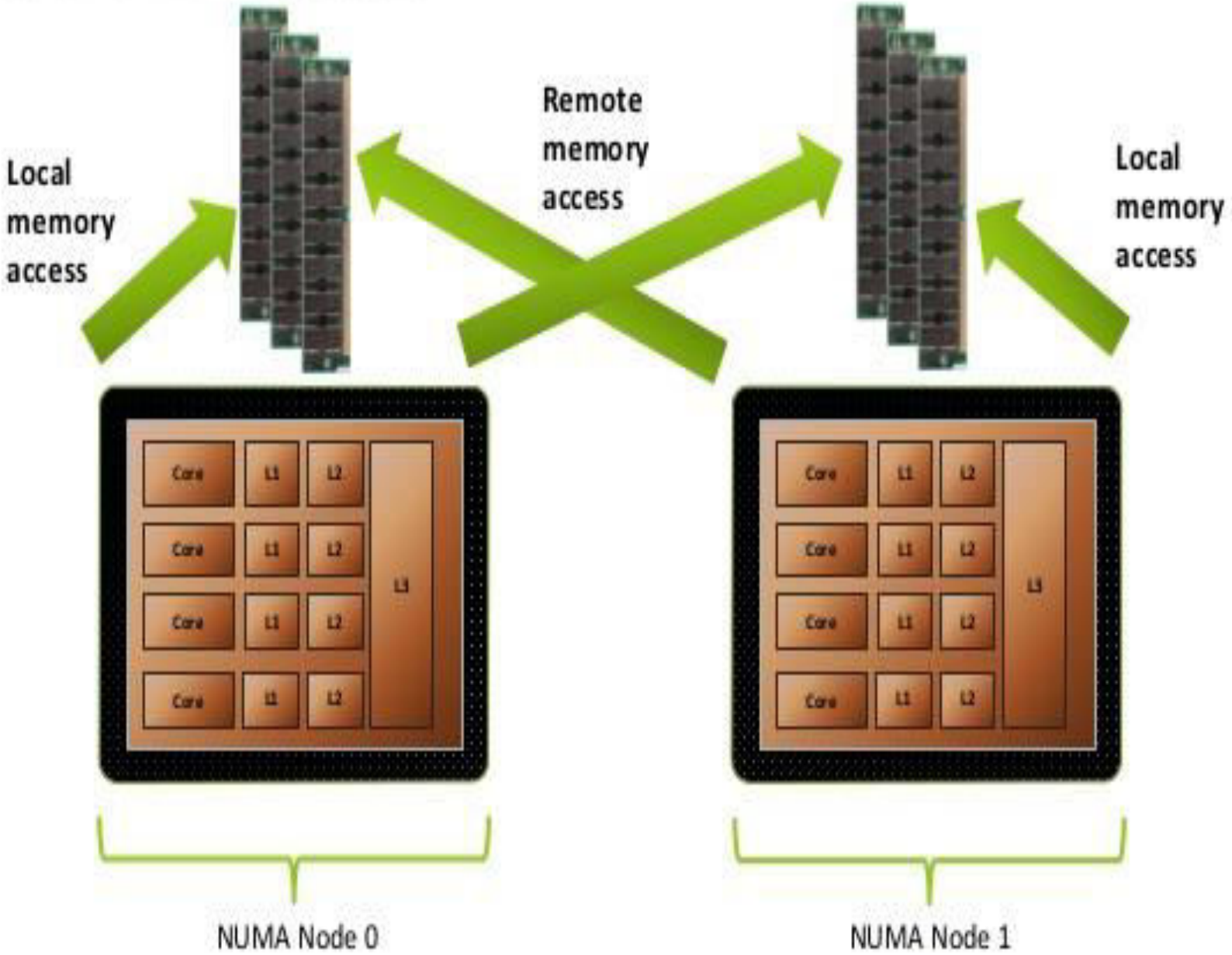
NUMA
Non Uniform Memory Access

- Is a method of configuring a cluster of microprocessor in a multiprocessing system so that they can share memory locally, improving performance and the ability of the system to be expanded.

- (What is NUMA: SISD or SIMD or MISD or MIMD?)


Memory
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
Memory
Bus Interconnect
Memory
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
Memory

A Basic NUMA Architecture



NUMA

- is used in a symmetric multiprocessing (SMP) system.
- An SMP system is a multiple processors system
  - "tightly-coupled," i.e"share everything" system, access each other's memory over a common bus or "interconnect" path.
  - working under a single operating system

Limitation of SMP

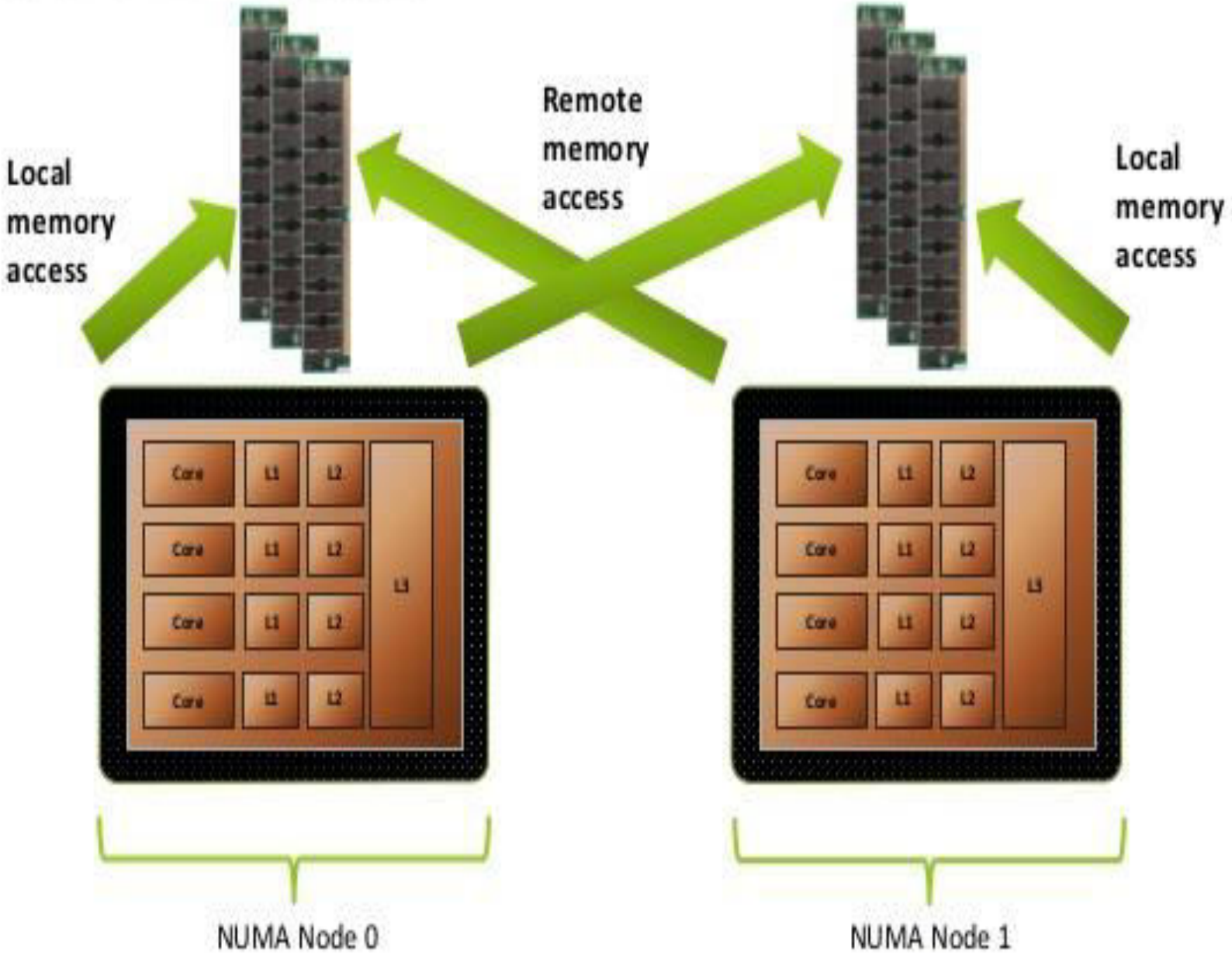
(symmetric multiprocessing)

- is that as microprocessors are added, the shared bus or data path get overloaded and becomes a performance bottleneck
- NUMA adds an intermediate level of memory shared among a few microprocessors so that all data accesses don't have to travel on the main bus.

NUMA

- cluster typically consists of four microprocessors (for example, four Pentium microprocessors)
- **interconnected on a local bus** (for example, a Peripheral Component Interconnect bus) to a shared memory (called an "L3 cache ") on a single motherboard (it could also probably be referred to as a card).

A Basic NUMA Architecture



NUMA contd..

- This unit can be added to similar units to form a symmetric multiprocessing system(SMP)
- in which a *common SMP bus* interconnects all of the clusters.
  - Such a system typically contains from 16 to 256 microprocessors. To an application program running in an SMP system, all the individual processor memories look like a single memory.

NUMA(Non-Uniform Memory Access)

- Each of these clusters is viewed by NUMA as a "node" in the interconnection network.
- NUMA maintains a hierarchical view of the data on all the nodes.

NUMA
SCI coordinates Cache Coherence

- Data is moved on the bus between the clusters of a NUMA SMP(Symmetric Multiprocessing) system
  - using **scalable coherent interface** (SCI) technology
- SCI coordinates "cache coherence" or consistency across the nodes of the multiple clusters.

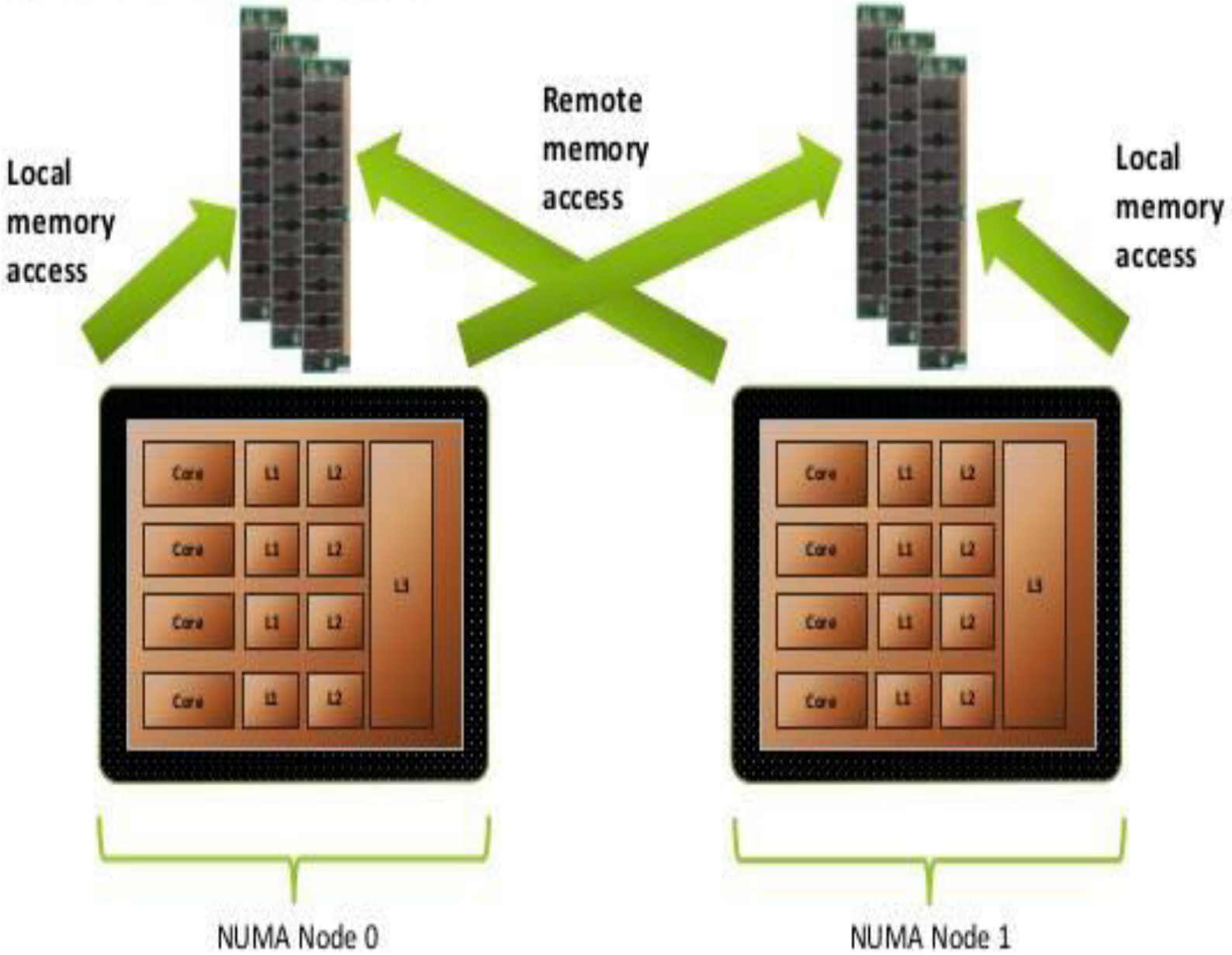
Searching sequence in Memory

- When a processor looks for data at a certain memory address,
  - it **first looks in** the **L1 cache line** on the microprocessor itself,
  - then on a somewhat larger L2 **cache chip** nearby,
  - then on a **third level L3 of cache** and memory that the NUMA configuration provides
  - At the end seeking the data in the **"remote memory"** located near the other microprocessors

Multilevel **Inclusion**

- Every level of cache hierarchy **is a subset** of *the **level further away** from the processor*
  - Then we can use the multilevel structure to reduce the contention between coherence traffic and processor traffic
    - This is **possible when block size is same** at all levels.
  - To maintain inclusion **with multiple block size** , check higher level of hierarchy, when replacement is done at the lower level to ensure that any word replaced at lower level is invalidated in the higher cache

A Basic NUMA Architecture



Cache Coherence Problem

- In multi-processor systems,
  - data can reside in multiple levels of cache, as well as in main memory.
  - This can cause problems if all CPUs don't see the same value for a given memory location.

Cache Coherence contd..

- To ensure coherence and consistency, you want all caches to see all memory accesses in program order.
- A memory system is **coherent**
  - **if it sees memory** accesses to a single location in order
  - A read to P following a write to P returns P, regardless of which processor reads/writes
    - Writes are serialized so each processor sees them in the same order

Cache Consistency

- A memory system is **consistent**
  - **if it sees memory** accesses to different locations in order

Examples of Oder Maintenance

First, a write does not

complete (and allow the next write to occur) until all processors have seen the effect of that write.

Second, the processor does not change the order of any write with respect to any other memory access. These two conditions mean that if a

processor writes location A followed by location B, any processor that sees the new value of B

must also see the new value of A.

Syncronization

- To maintain the order of read-write (Serialization of critical sections)
  - Done **by hardware spinlocks**
  - **software queuelocks** or semaphore type of things from programmers point of view.
- To maintain order of memory access among processors
  - Fences are set to ensure no read or write is moved across the fences
    - Used to ensure writes /reads of a portion have completed

Cache Coherence

Two classes of protocols to ensure

cache coherence

- Directory based: a central location (directory) contains the sharing status of all blocks in all caches

(more scalable).

- Snooping: each cache listens on a shared bus for events regarding cache blocks (less scalable).

Snooping Protocol

Two ways to ensure cache coherence

- Write invalidate protocol: on a write, **broadcast a block invalidation message on the bus so everyone knows their local copy is dirty.**
- Write update protocol: on a write, the value is broadcast on the bus and everyone updates their local copy.

Snooping Protocol

- We use write-invalidate + write-back caches
- Track block sharing status
  - When writing to a shared block, must first acquire the bus to broadcast the invalidation
- Finding data on a read
  - most recent version of the block might be in another cache (with *write-back caches)*
  - all caches must monitor the address line for *dirty* blocks they might have and update the status of those blocks so they don't use those values

Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	**0**		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

An example of an invalidation protocol working on a snooping bus for a single cache block (X) with write-back caches

Snooping Protocols 1

- Write Invalidate
  - CPU wanting to write to an address, grabs a bus cycle and sends a ‘write invalidate’ message
  - All snooping caches invalidate their copy of appropriate cache line
  - CPU writes to its cached copy (assume for now that it also writes through to memory)
  - Any shared read in other CPUs will now miss in cache and re-fetch new data.

Snooping Protocols 2

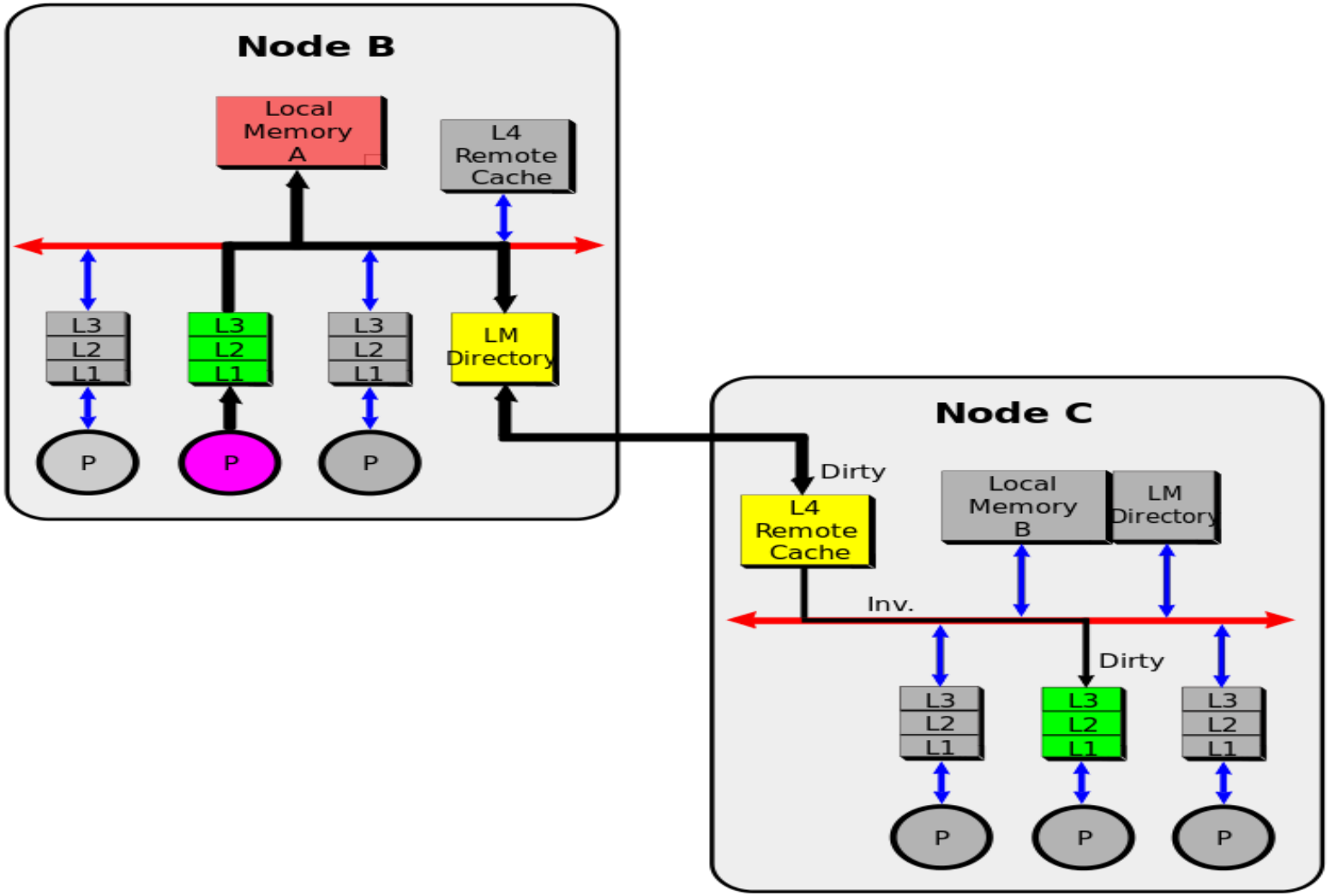
(not used since too many updates required)

- Write Update
  - CPU wanting to write grabs bus cycle and broadcasts new data as it updates its own copy
  - All snooping caches update their copy
- Note that in both schemes, problem of simultaneous writes is taken care of by bus arbitration - only one CPU can use the bus at any one time.

Update or Invalidate?

- Due to both spatial and temporal locality, previous cases occur often.
- Bus bandwidth is a precious commodity in shared memory multi-processors
- Experience has shown that invalidate protocols use significantly less bandwidth.
- Will consider implementation details only of invalidate.

cc-NUMA - Local Memory Read and Invalidate (RWITM) (Dirty in Remote Node)



By Ferruccio Zulian - Milan.Italy

Software Cache Solution

- Relies on operating system and compiler
- Compiler based coherence mechanism performs an analysis on the code to **determine which data items become unsafe for caching and mark them**
- Operating system prevents any non cacheable item being cached
- **Advantage of Software solution**: overhead of detecting potential problems **is transferred** from run time **to compile time**

NUMA Applications

- SMP and NUMA systems
- used for applications such as data mining and decision support system in which processing can be parceled out to a number of processors that collectively work on a common database

(Sequent, Data General, and NCR are among companies that produce NUMA SMP systems)

Causes of High Miss Rates

the three Cs model sorts all misses into three simple categories:

• *Compulsory—The very first access to a block cannot be in the cache, so the* block must be brought into the cache. Compulsory misses are those that occur even if you had an infinite cache.

• *Capacity—If the cache cannot contain all the blocks needed during execution* of a program, capacity misses (in addition to compulsory misses) will occur **because of blocks being discarded and later retrieved**.

• *Conflict—If the block placement strategy is not fully associative, conflict* misses (in addition to compulsory and capacity misses) will occur because a . **block may be discarded and later retrieved if conflicting blocks map to its set**