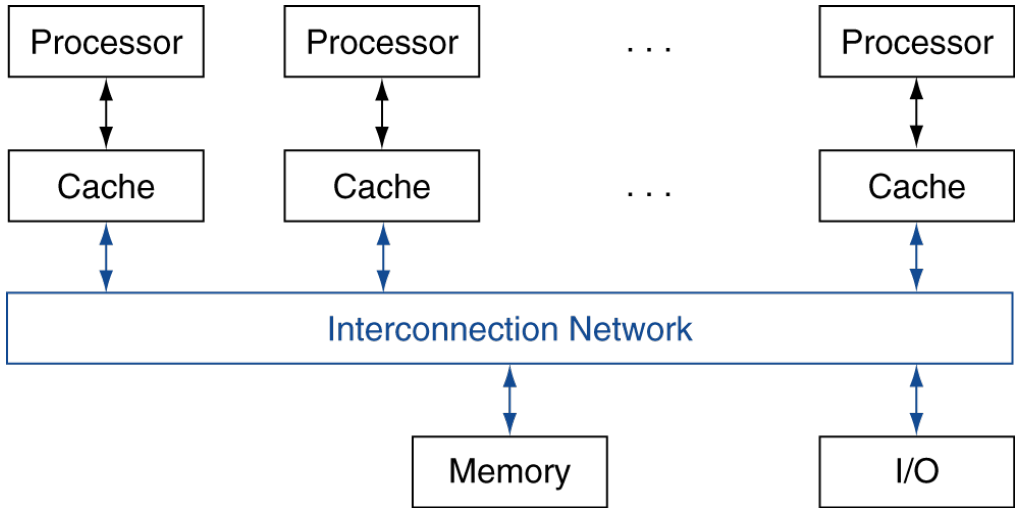
Shared Memory

- SMP: shared memory multiprocessor
  - Hardware provides single physical address space for all processors
  - Synchronize shared variables using locks
  - Memory access time
    - UMA (uniform) vs. NUMA (nonuniform)



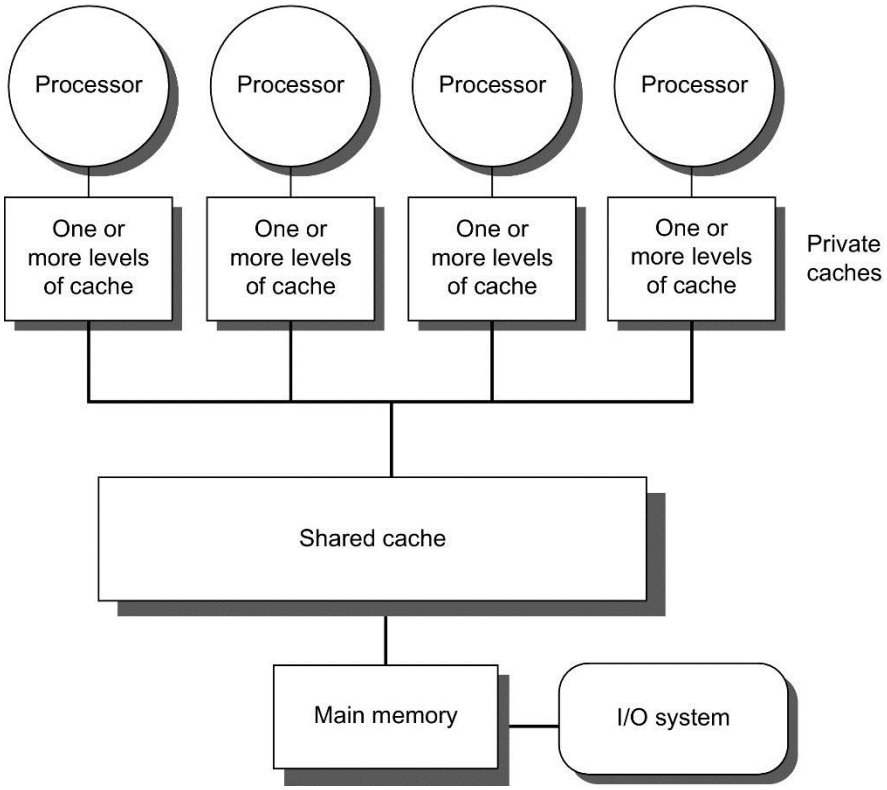


Figure 5.1 Basic structure of a centralized shared-memory multiprocessor based on a multicore chip. Multiple processor-cache subsystems share the same physical memory, typically with one level of shared cache on the multicore, and one or more levels of private per-core cache. The key architectural property is the uniform access time to all of the memory from all of the processors. In a multichip design, an interconnection network links the processors and the memory, which may be one or more banks. In a single-chip multicore, the interconnection network is simply the memory bus.

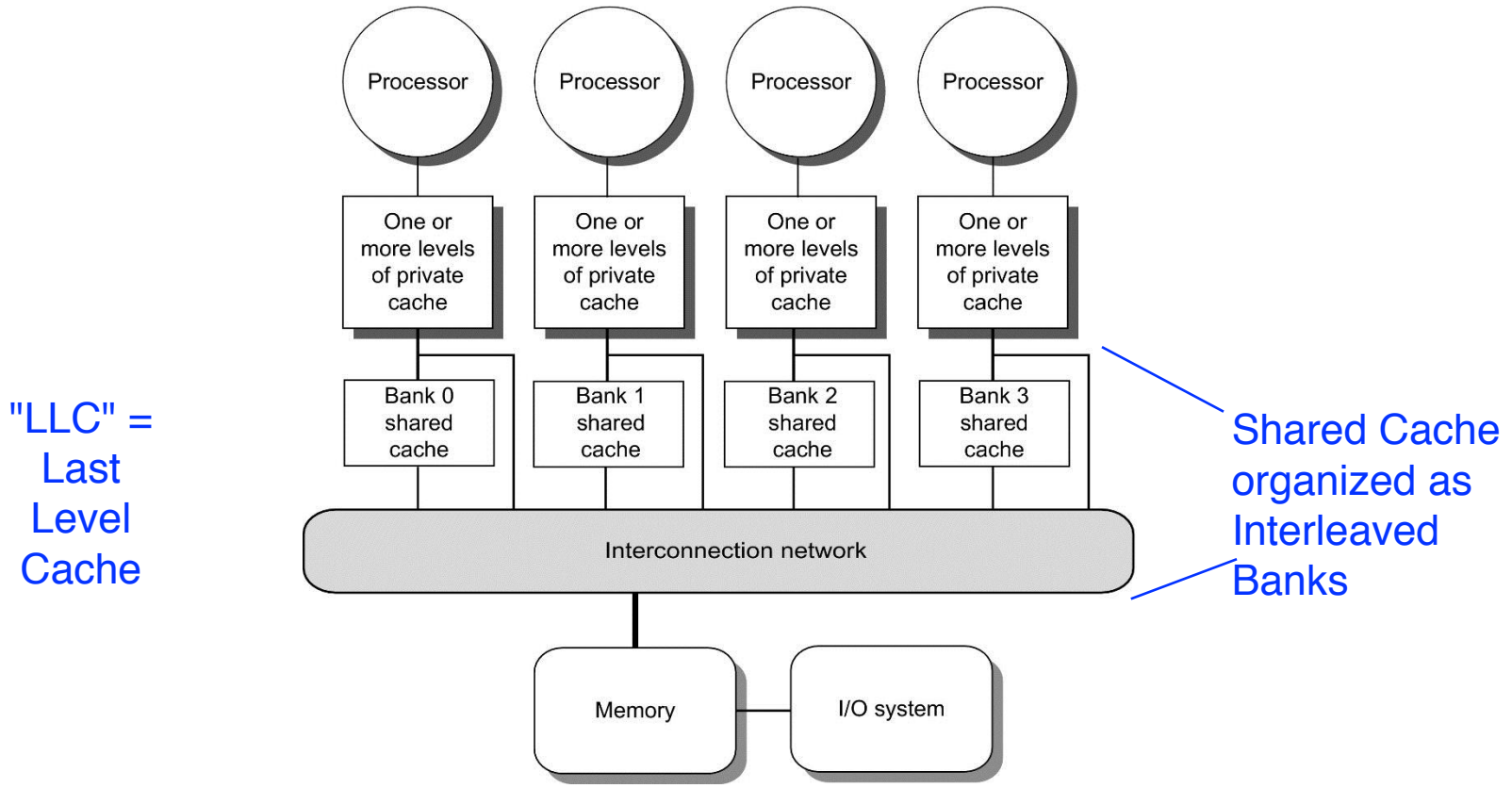


Figure 5.8 A single-chip multicore with a distributed cache. In current designs, the distributed shared cache is usually L3, and levels L1 and L2 are private. There are typically multiple memory channels (2–8 in today's designs). This design is NUCA, since the access time to L3 portions varies with faster access time for the directly attached core. Because it is NUCA, it is also NUMA.

Cache Coherence

- Coherence (Συνοχή)
  - All reads by any processor must return the most recently written value
  - Writes to the same location by any two processors are seen in the same order by all processors
- Consistency (Συνέπεια)
  - When a written value will be returned by a read
  - 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

Cache Coherence Problem

- Suppose two CPU cores share a physical address space
  - Write-through caches

Time step	Event	CPU A's cache	CPU B's cache	Memory
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0	1

Coherence Defined

- Informally: Reads return most recently written value
- Formally:
  - P writes X; P reads X (no intervening writes)
    ⇒ read returns written value
  - P_1 writes X; P_2 reads X (sufficiently later)
    ⇒ read returns written value
    - c.f. CPU B reading X after step 3 in example
  - P_1 writes X, P_2 writes X
    ⇒ all processors see writes in the same order
    - End up with the same final value for X

Cache Coherence Protocols

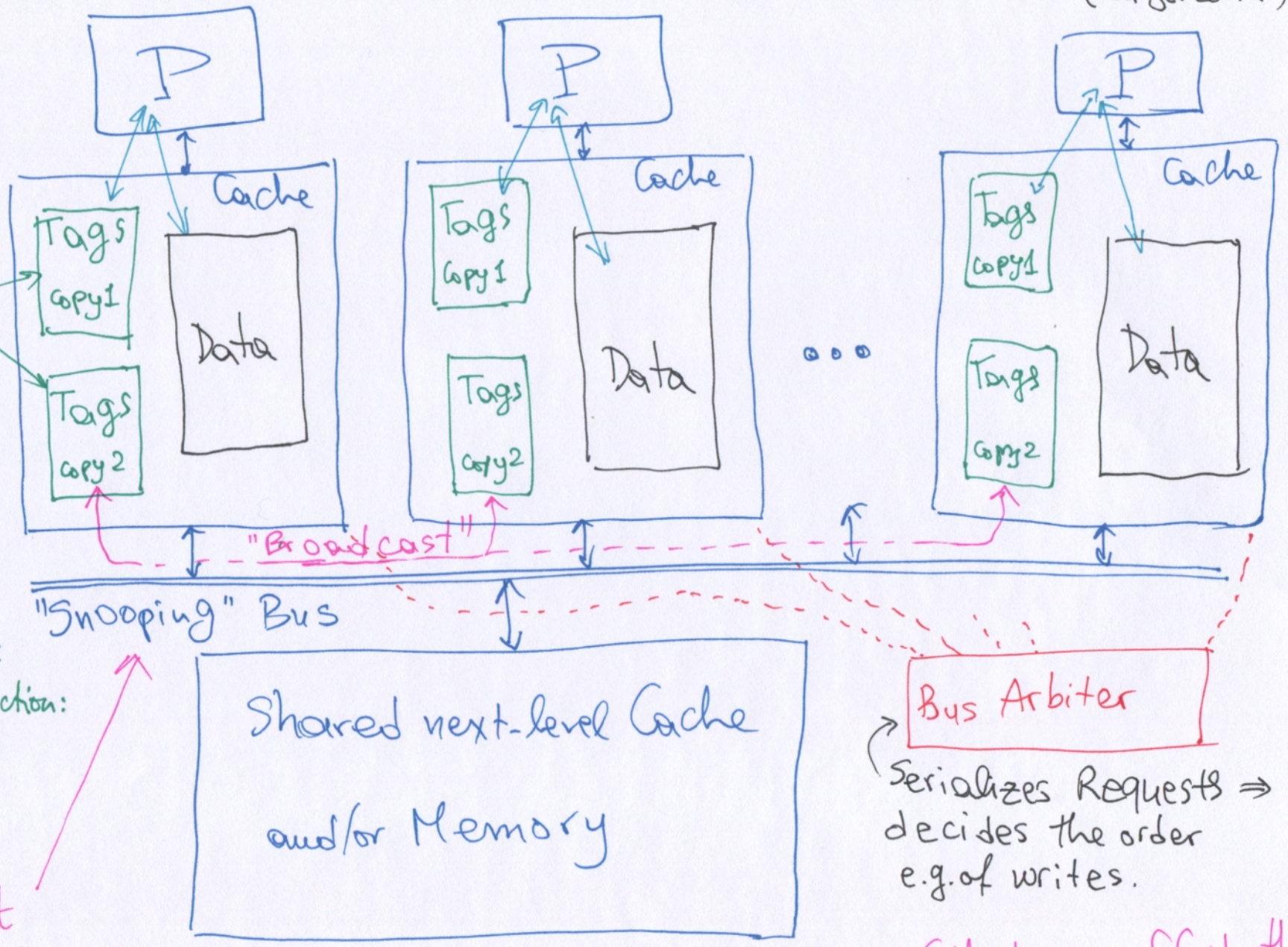
- Operations performed by caches in multiprocessors to ensure coherence
  - Migration of data to local caches
    - Reduces bandwidth for shared memory
  - Replication of read-shared data
    - Reduces contention for access
- Snooping protocols
  - Each cache monitors bus reads/writes
- Directory-based protocols
  - Caches and memory record sharing status of blocks in a directory

All activities that potentially affect other caches are broadcast onto the shared bus; all caches monitor ("snoop") that shared bus.

OK for few (4, 8, 16?) sharers, but too much traffic beyond ~8.

A central Directory (may consist of interleaved banks) records which caches have copies of which blocks => only "bother" those caches that are affected

Snooping Cache Coherence (Συνοχή κρυφής Μνήμης με κρυφοκοίταγμα/καταπολέμηση (κουτσομπολιό στη γειτονιά))



Two copies of tags, or 2-port TagsMemory: mostly read-only; only need to write infrequently: on misses or when affected by bus transaction: then, have to write into both copies.

Broadcast Medium: everybody has to listen to everything going on (that may affect others) — when does it affect others ???

Serializes Requests ⇒ decides the order e.g. of writes.

Upon Writes: to Invalidate or to Update the Others?

- **Invalidate-based Protocols:**

  When I write (modify) something that I have, and there is a danger others may have copies of it, tell them to invalidate their copies!

  (like telling them: "If you ever need it again, come back to me and ask me for its latest value")

  - Advantage: from that point on, I know that I have the only copy, therefore I can freely "play with it", as long as nobody ask me for a copy again.

  (most usual)

- **Update-based Protocols:**

  When I write (modify) something that I have, and there is a danger others may have copies of it, broadcast the new value so as to update the values of all copies!

  - Advantage: if others will need the value again soon, they will have it in their caches
  - Disadvantage: multiple copies will keep existing, hence the need to continuously keep updating them

  (rarely (if ever) used)

(statistics showed that disadvantage is important in the general case (unless there is hint about some data by the algorithm/software)

Invalidating Snooping Protocols

- Cache gets exclusive access to a block when it is to be written
  - Broadcasts an invalidate message on the bus
  - Subsequent read in another cache misses
    - Owning cache supplies updated value

CPU activity	Bus activity	CPU A's cache	CPU B's cache	Memory
				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 1 to X	Invalidate for X	1		0
CPU B read X	Cache miss for X	1	1	1

Write-Back

Snoopy Coherence Protocols

- Locating an item when a read miss occurs
  - In write-back cache, the updated value must be sent to the requesting processor
- Cache lines marked as shared or exclusive/modified
  - Only writes to shared lines need an invalidate broadcast
    - After this, the line is marked as exclusive

Κατάσταση (State) κάθε Γραμμής: ορολογία "MOESI"

Obliged to Write–back

my copy is up to date;
the memory version is outdated;
I am responsible to write back to memory, and also to provide this up to date version to other caches

Free to Evict

the memory or another cache have the up to date version, and I have a copy of that, which I am free to evict at any time

Guaranteed Exclusive

it is guaranteed that my copy is the only copy currently existing in any cache

Potentially Shared

other caches may have copies of this line (not known for sure)

Invalid

nothing in this cache line

- Επέκταση των Valid-Dirty (per line) bits: τι ξέρω γιά την γραμμή

Simplification: MSI Protocol

(No "E" info when Clean; If Dirty, must be Exclusive (no "O" state))

M (Modified): Dirty and Exclusive

- with invalidate-based protocol, when I write into my copy, I have to invalidate all others, hence I am guaranteed to have the ONLY copy ⇒ Exclusive

S (Shared): (potentially) Shared and guaranteed Clean

- when first reading from memory → S (no "E" info maintained)
- for simplicity, I do not keep track whether or not others too may have copies
- if I had the line as M, and another cache misses it, then:
  - I have to write it back to memory (no "O" state, for simplicity)
  - the other cache gets a copy automatically on the bus
- I may have had the line as S, and all other caches may have evicted their copies, so I may be the only one having a copy, but I do NOT know that for sure...
- If I have the line as S, and I want to write into it, I must broadcast an Invalidate command, since I have no "E" info!

I (Invalid)

from MSI to MESI protocol:

- Add and maintain E (exclusive and clean) info:
  - When I read-miss, if no other cache responds (faster than memory) providing me the data, and I have to wait for the memory to bring me the data ⇒ then I know I am "E".
  - Advantage versus MSI: If the line is "E" and I want to write into it, I do NOT need to broadcast any invalidate: save bus traffic.

MOESI protocol:

- Add "O" (Owned) info: Line is shared, Memory is "old", and the responsibility to write back is mine!
  - When the line is "M" (Modified: dirty and exclusive), and another cache read-misses on it, I supply the data to the other cache (faster than memory), but I do not spend the time to write-back to memory (now): save time (for now).
  - The other caches that have copies, have them in S state, hence they are allowed to evict their copies without writing back: I have the (only) "O" copy, hence the responsibility to write-back is mine!

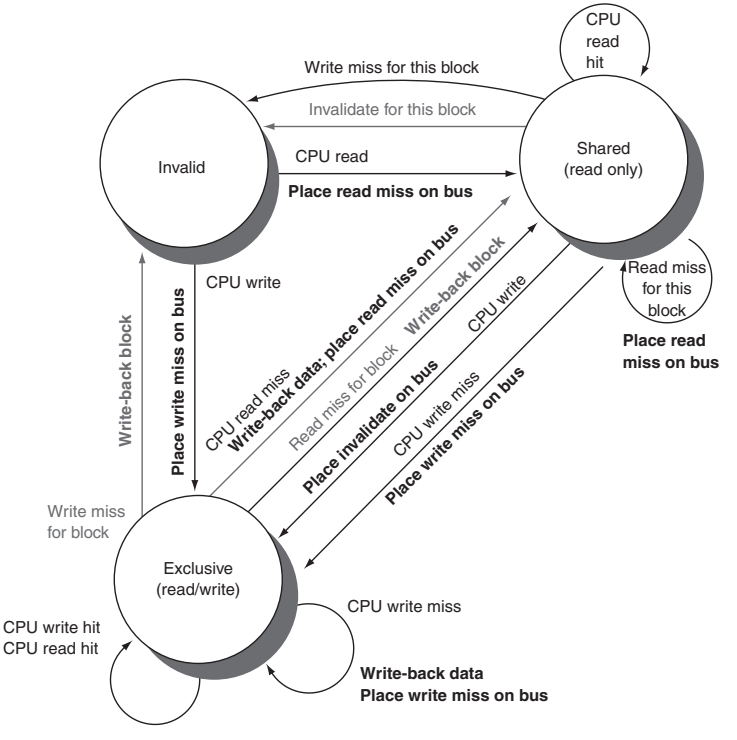


FIGURE e5.12.11 Cache coherence state diagram with the state transitions induced by the local processor shown in black and by the bus activities shown in gray. As in Figure e5.12.10, the activities on a transition are shown in bold.