Multiprocessor Systems

- Tightly Coupled vs. Loosely Coupled Systems
  - tightly coupled system generally represent systems which have some degree of sharable memory through which processors can exchange information with normal load / store operations
  - Loosely coupled systems generally represent systems in which each processor has its own private memory and processor to processor information exchange is done via some message passing mechanism like a network interconnect or an external shared channel (FC, IB, SCSI, etc.) bus

Multiprocessor Systems

- The text presentation in chapters 16 and 17 deals primarily with tightly coupled systems in 2 basic categories:
  - uniform memory access systems (UMA)
  - non-uniform memory access systems (NUMA)
- Distributed systems (discussed beginning in chapter 4) are often referred to as no remote access or NORMA systems

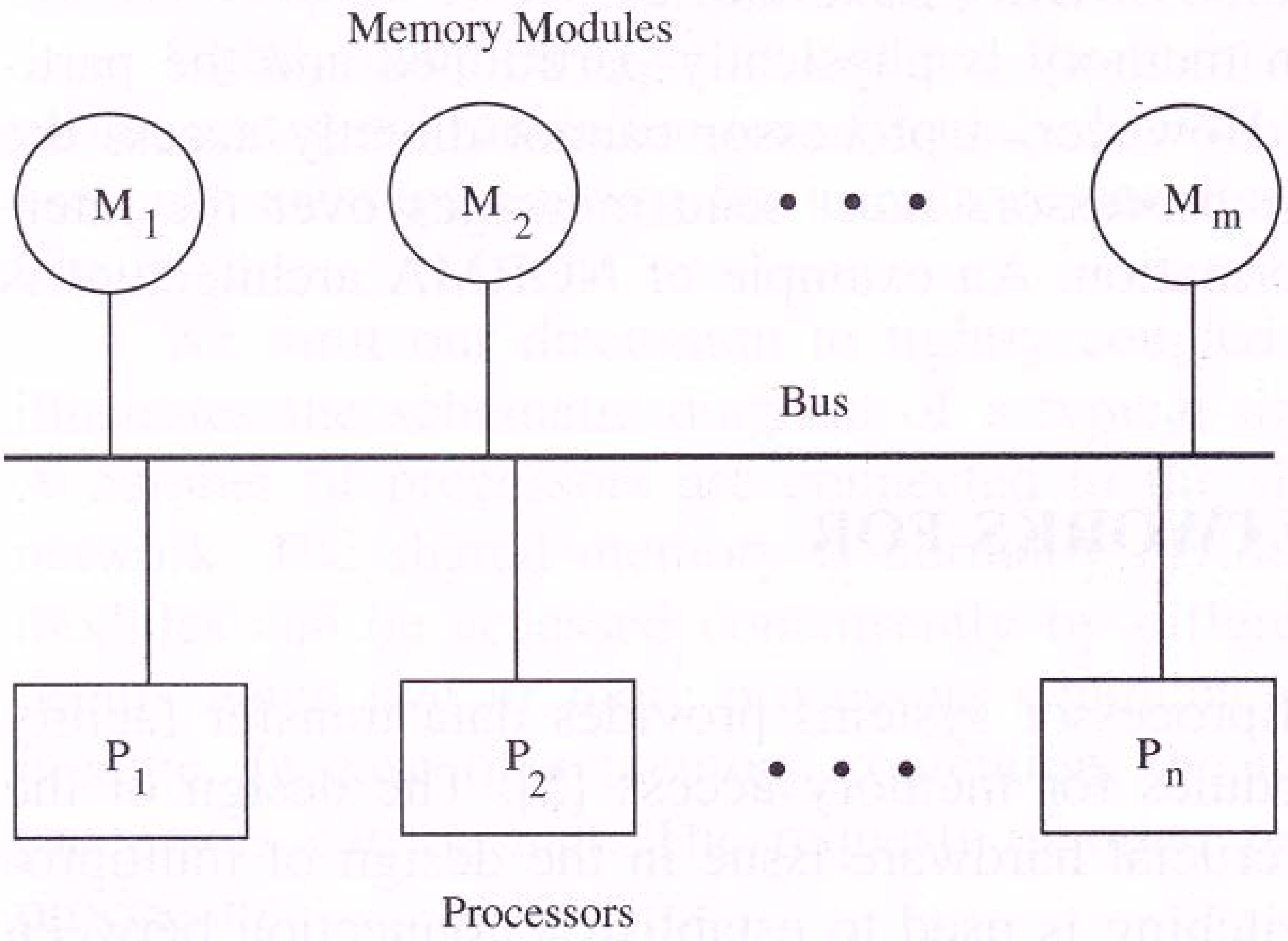
Multiprocessor Systems

- UMA and NUMA systems provide access to a common set of physical memory addresses using some interconnect strategy
  - a single bus interconnect is often used for UMA systems, but more complex interconnects are needed for scale-up
    - cross-bar switches
    - multistage interconnect networks
  - some form of fabric interconnect is common in NUMA systems
    - far-memory fabric interconnects with various cache coherence attributes

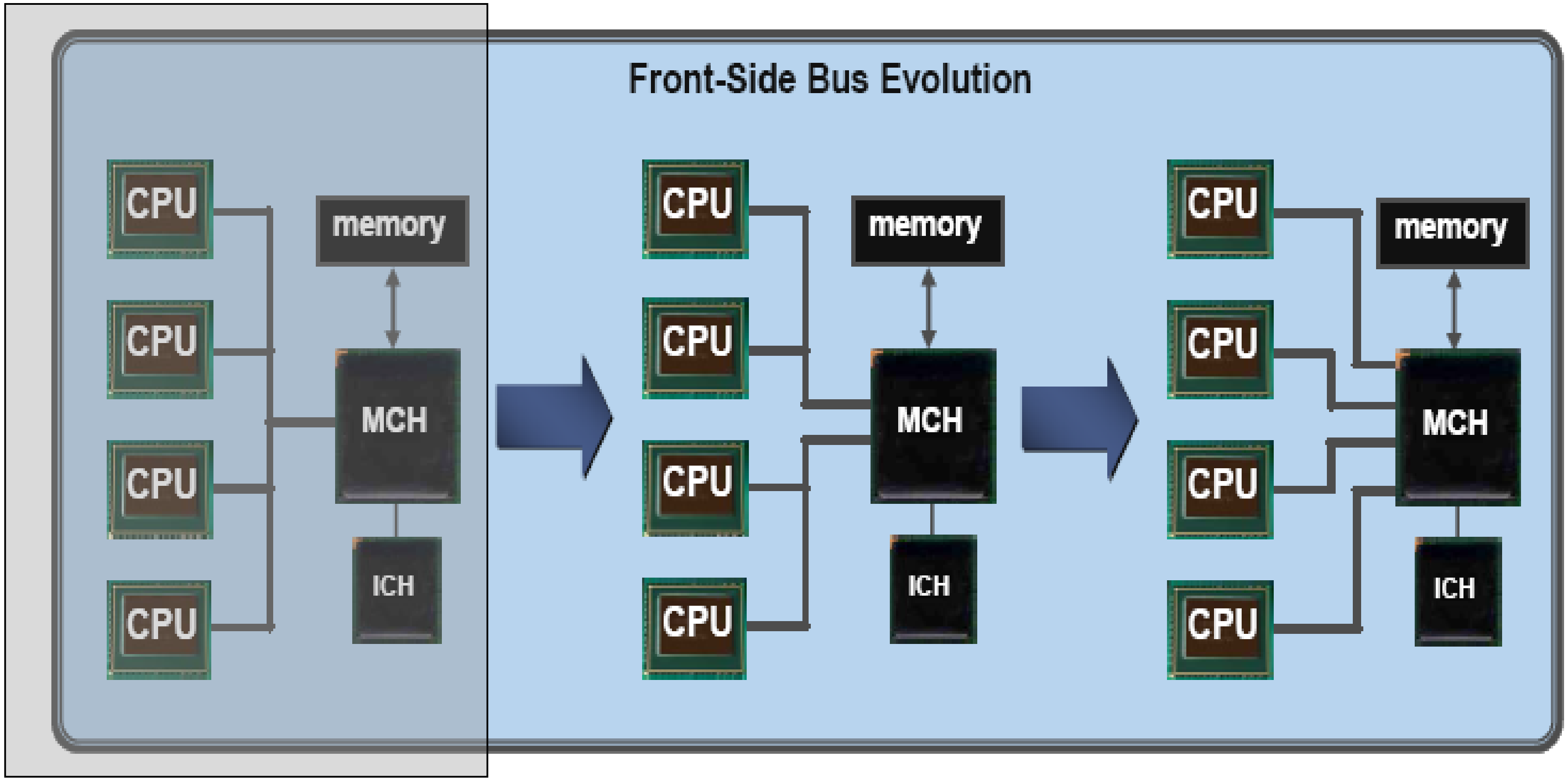
UMA Bus-Based SMP Architectures

- The simplest multiprocessors are based on a single bus.
  - Two or more CPUs and one or more memory modules all use the same bus for communication.
  - If the bus is busy when a CPU wants to access memory, it must wait.
  - Adding more CPUs results in more waiting.
  - This can be mitigated to some degree by including processor cache support

Single Bus Topology



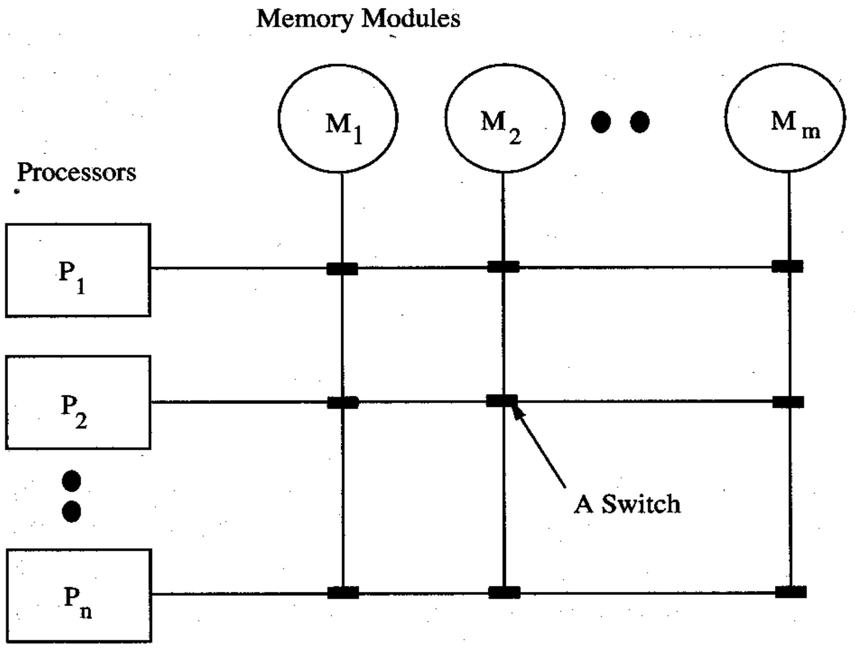
Single Bus Topology

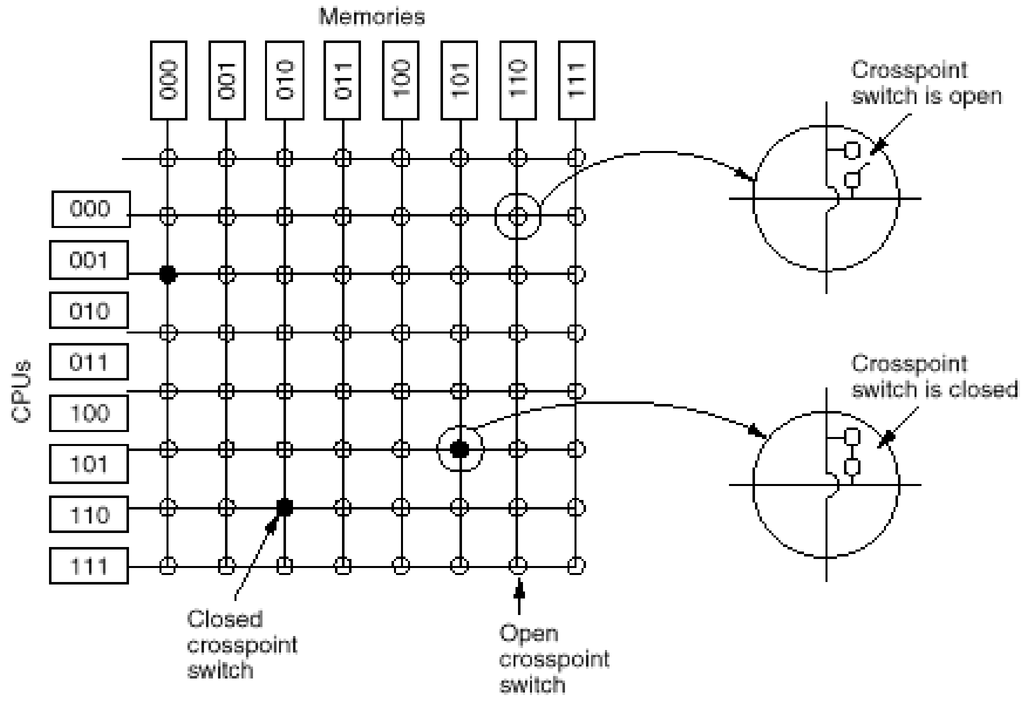


UMA Multiprocessors Using Crossbar Switches

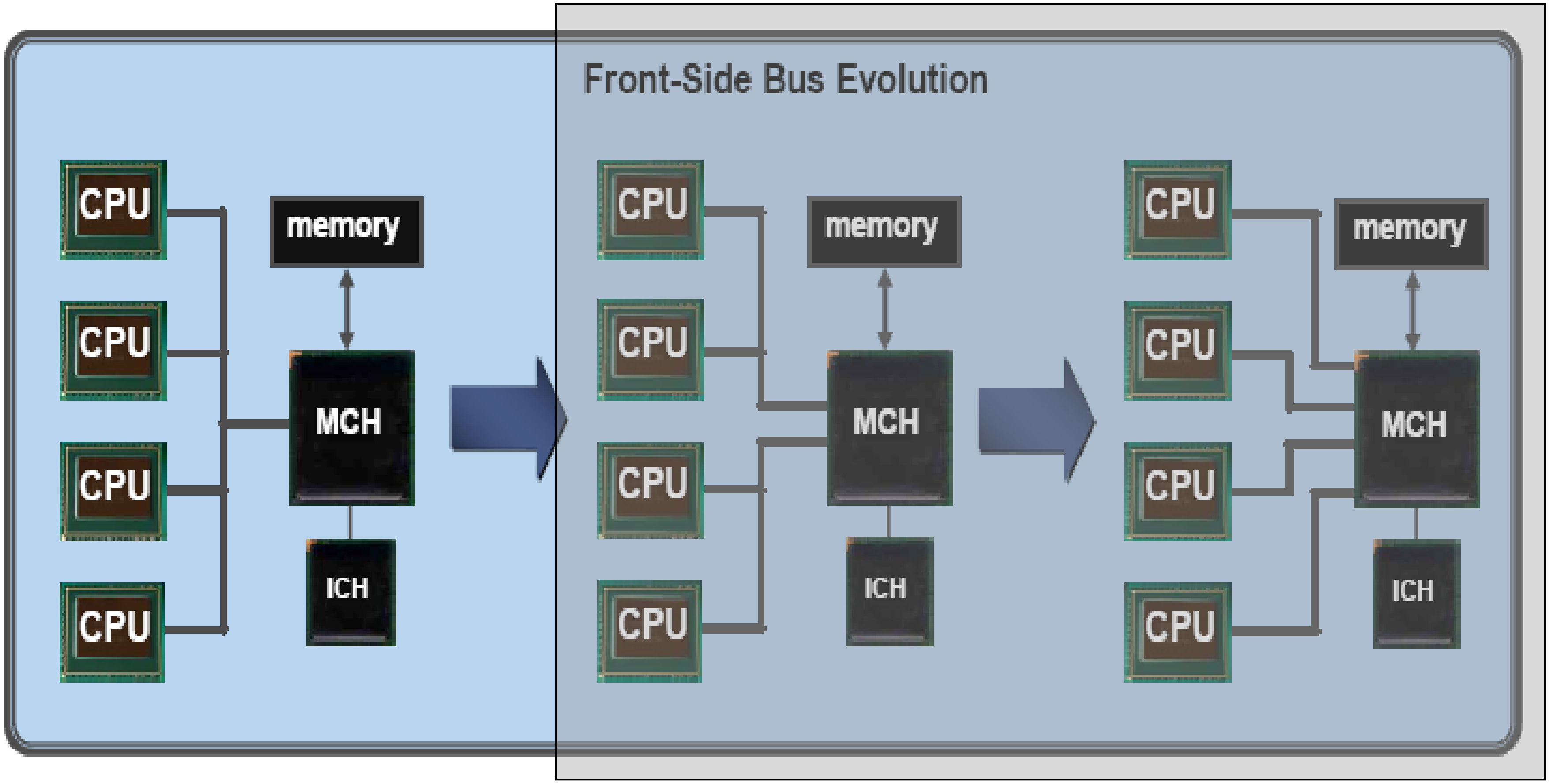
- Even with all possible optimizations, the use of a single bus limits the size of a UMA multiprocessor to about 16 CPUs.
  - To go beyond that, a different kind of interconnection network is needed.
  - The simplest circuit for connecting n CPUs to k memories is the **crossbar switch**.
    - Crossbar switches have long been used in telephone switches.
    - At each intersection is a **crosspoint** - a switch that can be opened or closed.
    - The crossbar is a **nonblocking** network

Cross-bar Switch Topology
Scale-up is ~N^2

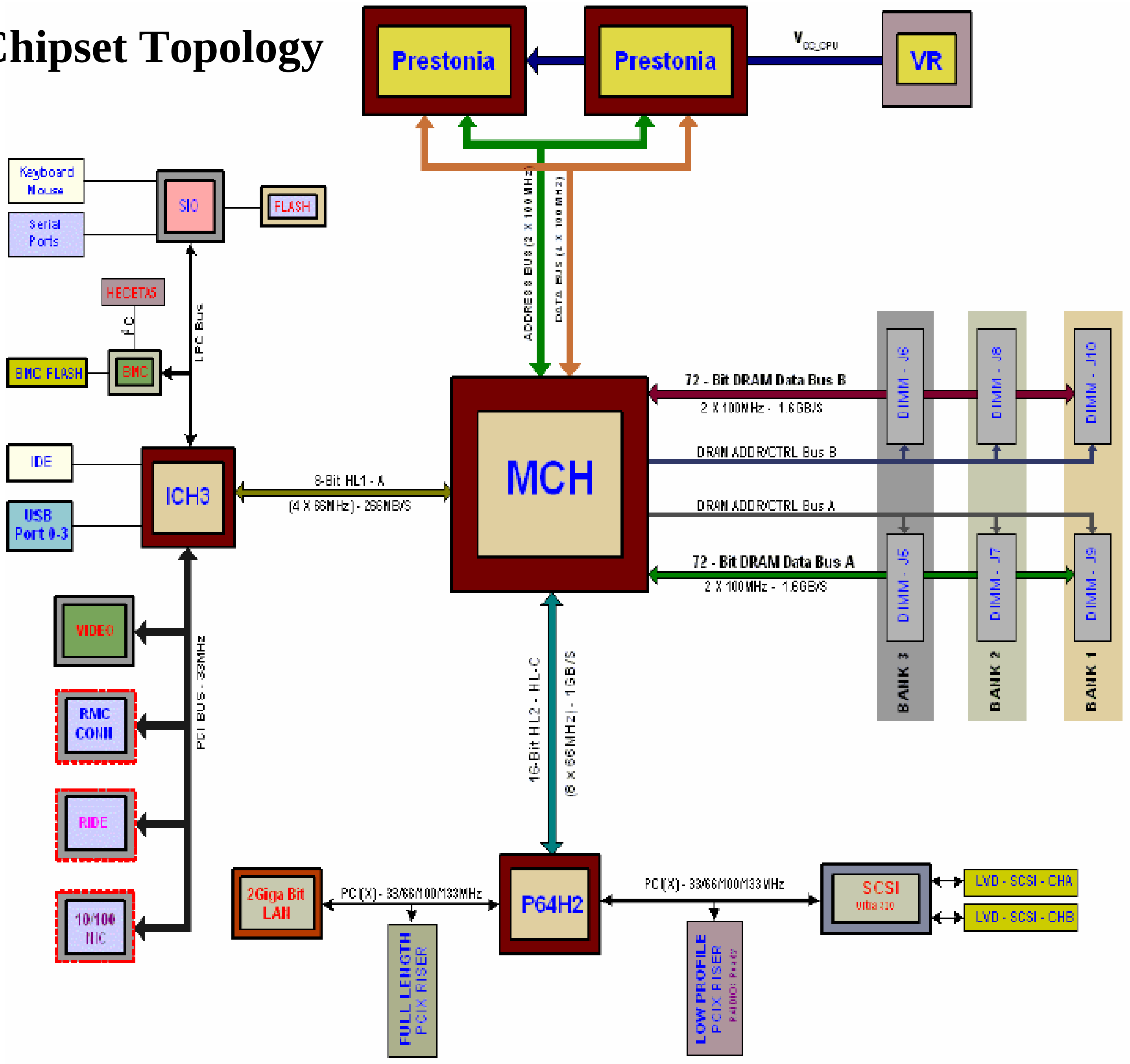




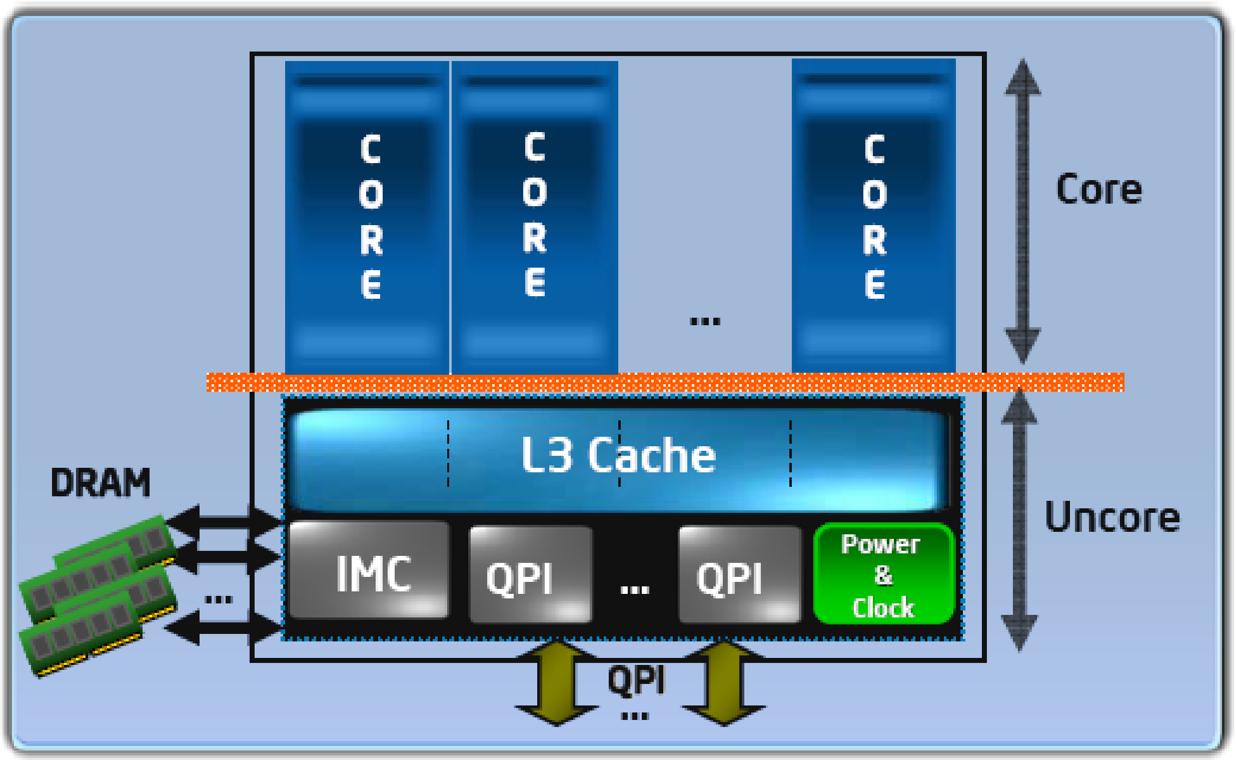
Crossbar Chipset Topology



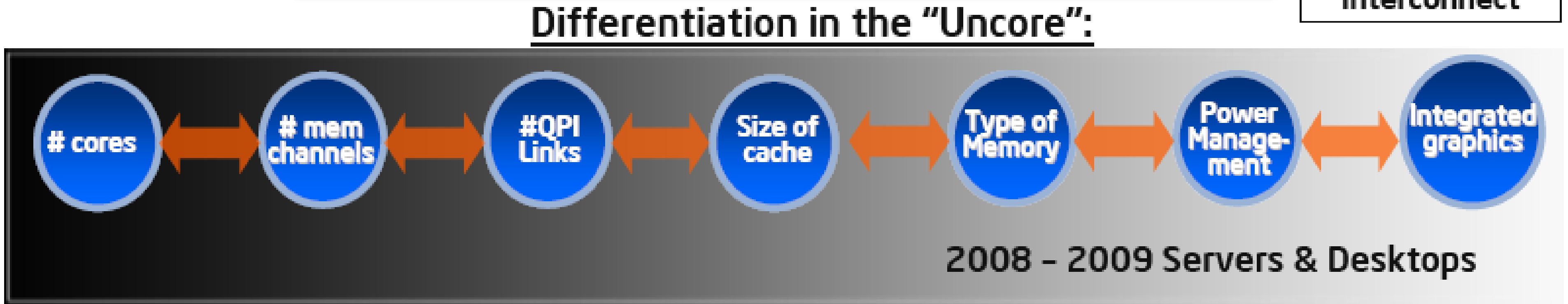
Crossbar Chipset Topology



Crossbar On-Die Topology
Nehalem Core Architecture



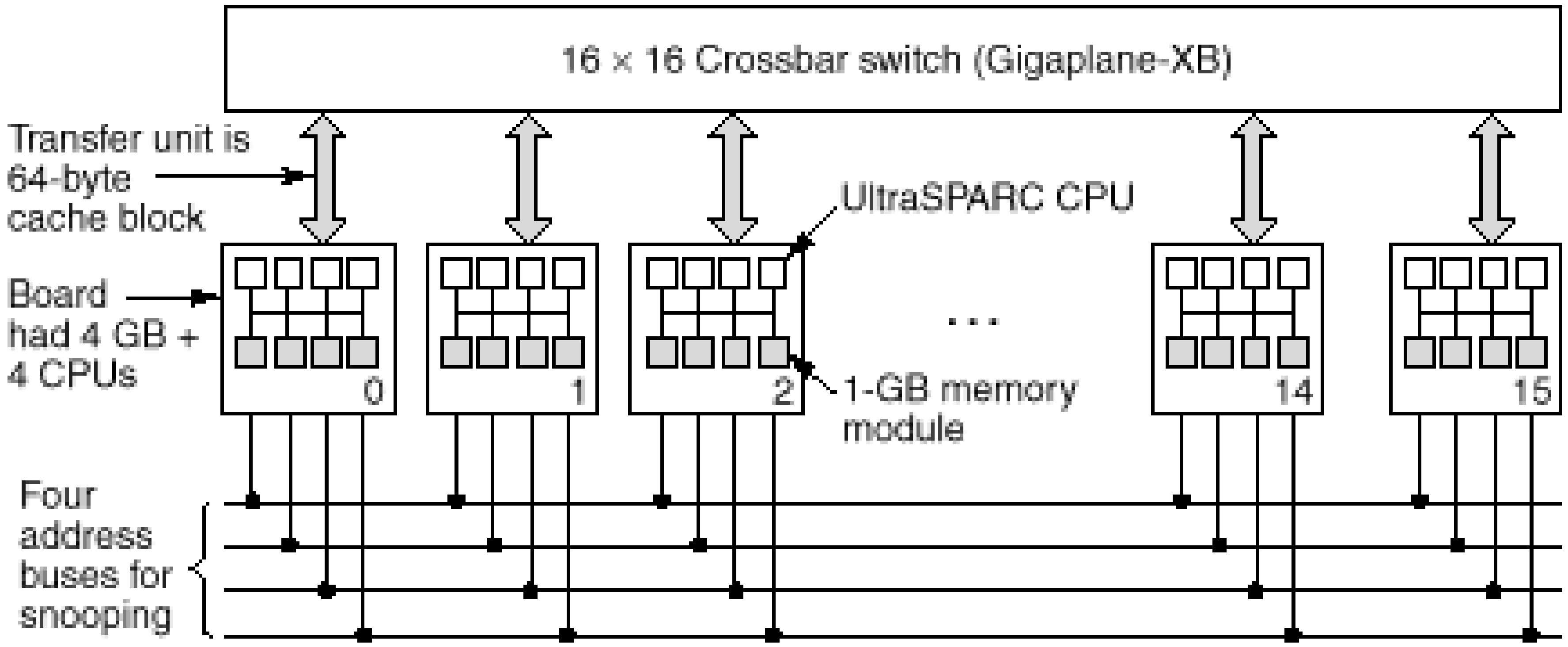
QPI: Intel® QuickPath Interconnect



Sun Enterprise 1000

- An example of a UMA multiprocessor based on a crossbar switch is the Sun Enterprise 1000.
  - This system consists of a single cabinet with up to 64 CPUs.
  - The crossbar switch is packaged on a circuit board with eight plug in slots on each side.
  - Each slot can hold up to four UltraSPARC CPUs and 4 GB of RAM.
  - Data is moved between memory and the caches on a 16 X 16 crossbar switch.
  - There are four address buses used for snooping.

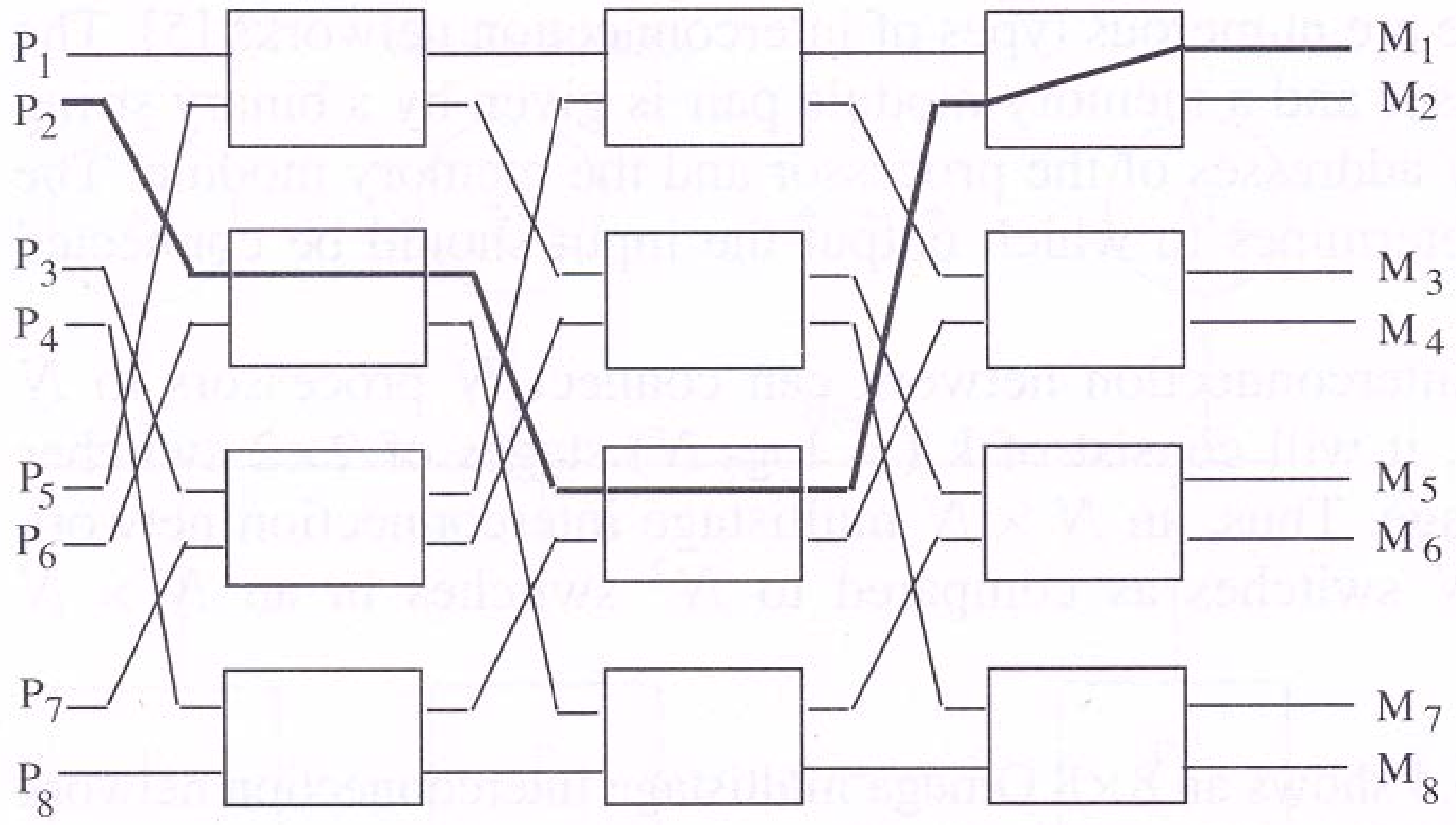
Sun Enterprise 1000 (cont'd)


16 × 16 Crossbar switch (Gigaplane-XB)
Transfer unit is 64-byte cache block
UltraSPARC CPU
Board had 4 GB + 4 CPUs
...
1-GB memory module
0
1
2
14
15
Four address buses for snooping

UMA Multiprocessors Using Multistage Switching Networks

- In order to go beyond the limits of the Sun Enterprise 1000, we need to have a better interconnection network.
- We can use 2 X 2 switches to build large **multistage switching networks**.
  - One example is the **omega network**.
  - The wiring pattern of the omega network is called the perfect shuffle.
  - The labels of the memory can be used for routing packets in the network.
  - The omega network is a **blocking network**.

Multistage Interconnect Topology
Scale-up is N (log N)



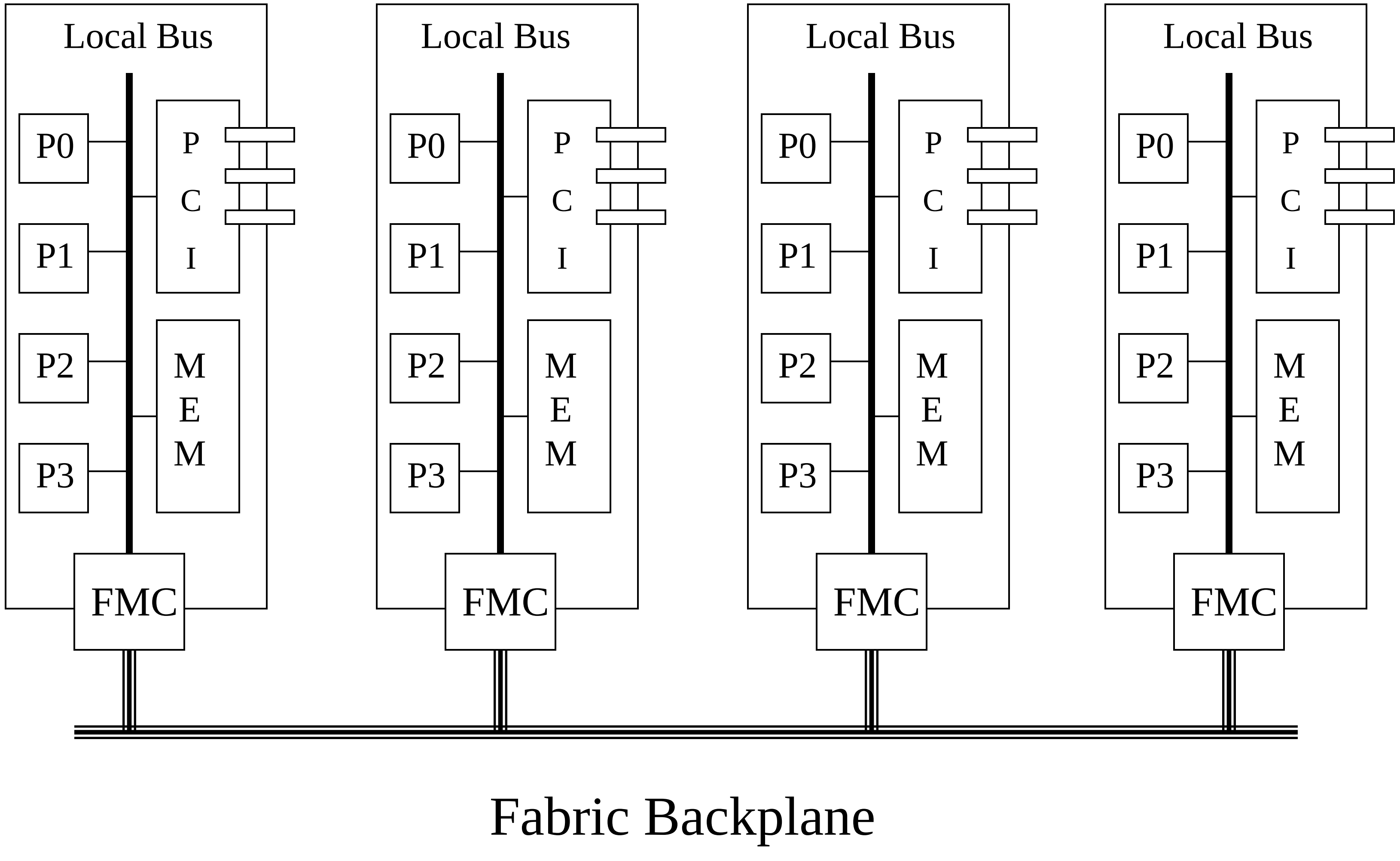
NUMA Multiprocessors

- To scale to more than 100 CPUs, we have to give up uniform memory access time.
- This leads to the idea of **NUMA** (**NonUniform Memory Access**) multiprocessors.
  - They share a single address space across all the CPUs, but unlike UMA machines local access is faster than remote access.
  - All UMA programs run without change on NUMA machines, but their performance may be worse.
    - When the access time to the remote machine is not hidden (by caching) the system is called **NC-NUMA.**

NUMA Multiprocessors (cont'd)

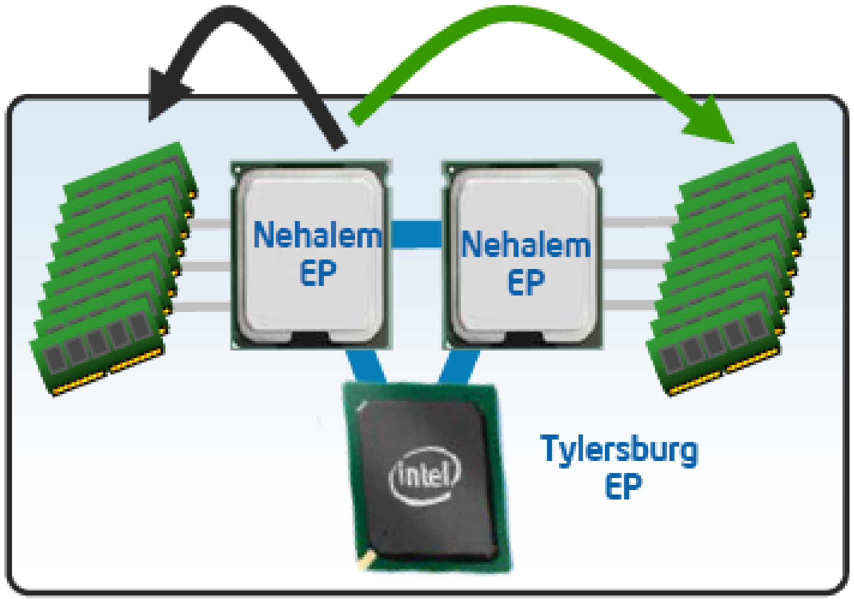
  - When coherent caches are present, the system is called **CC-NUMA**.
  - It is also sometimes known as **hardware DSM** since it is basically the same as software distributed shared memory but implemented by the hardware using a small page size.
- One of the first NC-NUMA machines was the Carnegie Mellon Cm*.
  - This system was implemented with LSI-11 CPUs (the LSI-11 was a single-chip version of the DEC PDP-11).
  - A program running out of remote memory took ten times as long as one using local memory.
  - Note that there is no caching in this type of system so there is no need for cache coherence protocols

In a full NUMA system memory and peripheral space is visible to any processor on any node



NUMA On-Die Topology
Intel Nehalem Core Architecture
(i3, i5, i7 family)

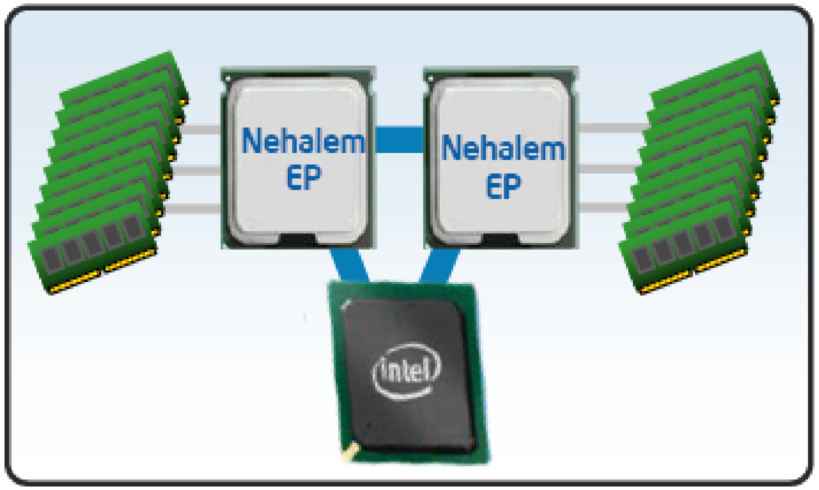
- FSB architecture
  - All memory in one location
- Starting with Nehalem
  - Memory located in multiple places
- Latency to memory dependent on location
- Local memory
  - Highest BW
  - Lowest latency
- Remote Memory
  - Higher latency

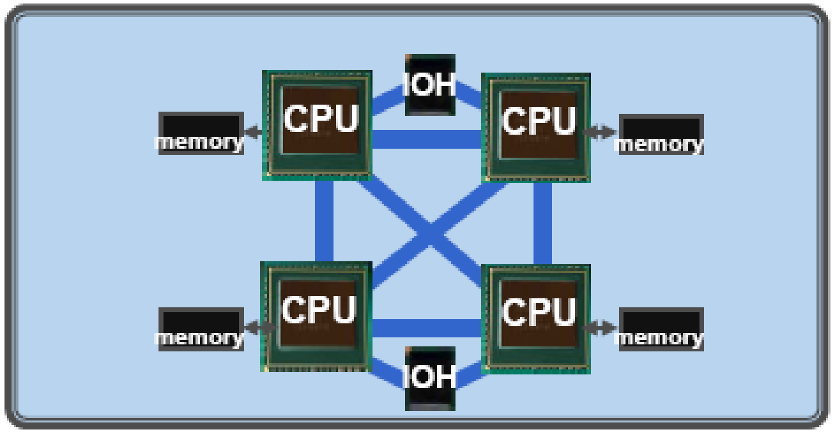


Ensure software is NUMA-optimized for best performance

NUMA QPI Support
Nehalem Core Architecture

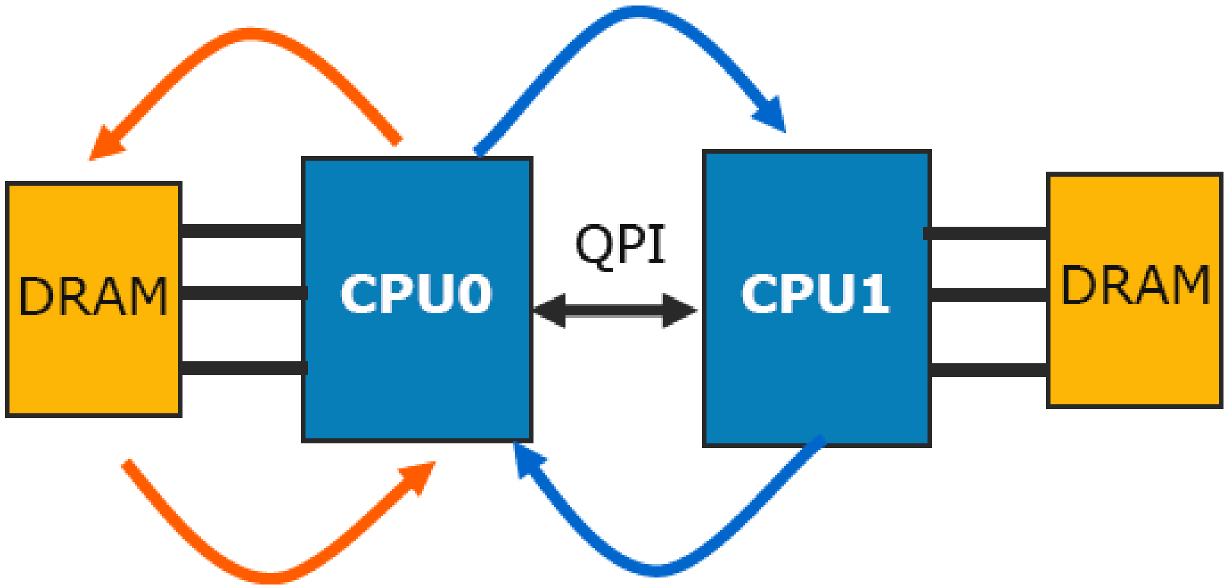
- Nehalem introduces new Intel® QuickPath Interconnect (QPI)
- ***High bandwidth, low latency*** point to point interconnect
- Up to 6.4 GT/sec initially
  - 6.4 GT/sec -> 12.8 GB/sec
  - Bi-directional link -> 25.6 GB/sec per link
  - Future implementations at even higher speeds
- Highly ***scalable*** for systems with varying # of sockets





Local Memory Access

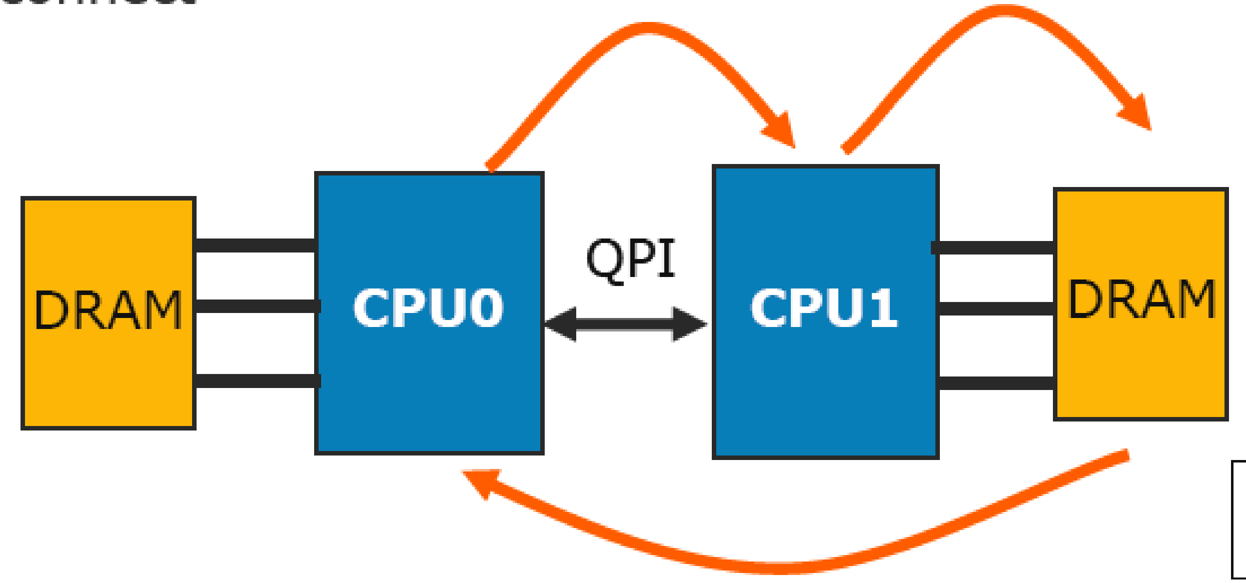
- CPU0 requests cache line X, not present in any CPU0 cache
  - CPU0 requests data from its DRAM
  - CPU0 snoops CPU1 to check if data is present
- Step 2:
  - DRAM returns data
  - CPU1 returns snoop response
- Local memory latency is the maximum latency of the two responses
- Nehalem optimized to keep key latencies close to each other



QPI: Intel® QuickPath Interconnect

Remote Memory Access

- CPU0 requests cache line X, not present in any CPU0 cache
  - CPU0 requests data from CPU1
  - Request sent over QPI to CPU1
  - CPU1's IMC makes request to its DRAM
  - CPU1 snoops internal caches
  - Data returned to CPU0 over QPI
- Remote memory latency a function of having a low latency interconnect



QPI: Intel® QuickPath Interconnect

Multiprocessor Operating Systems

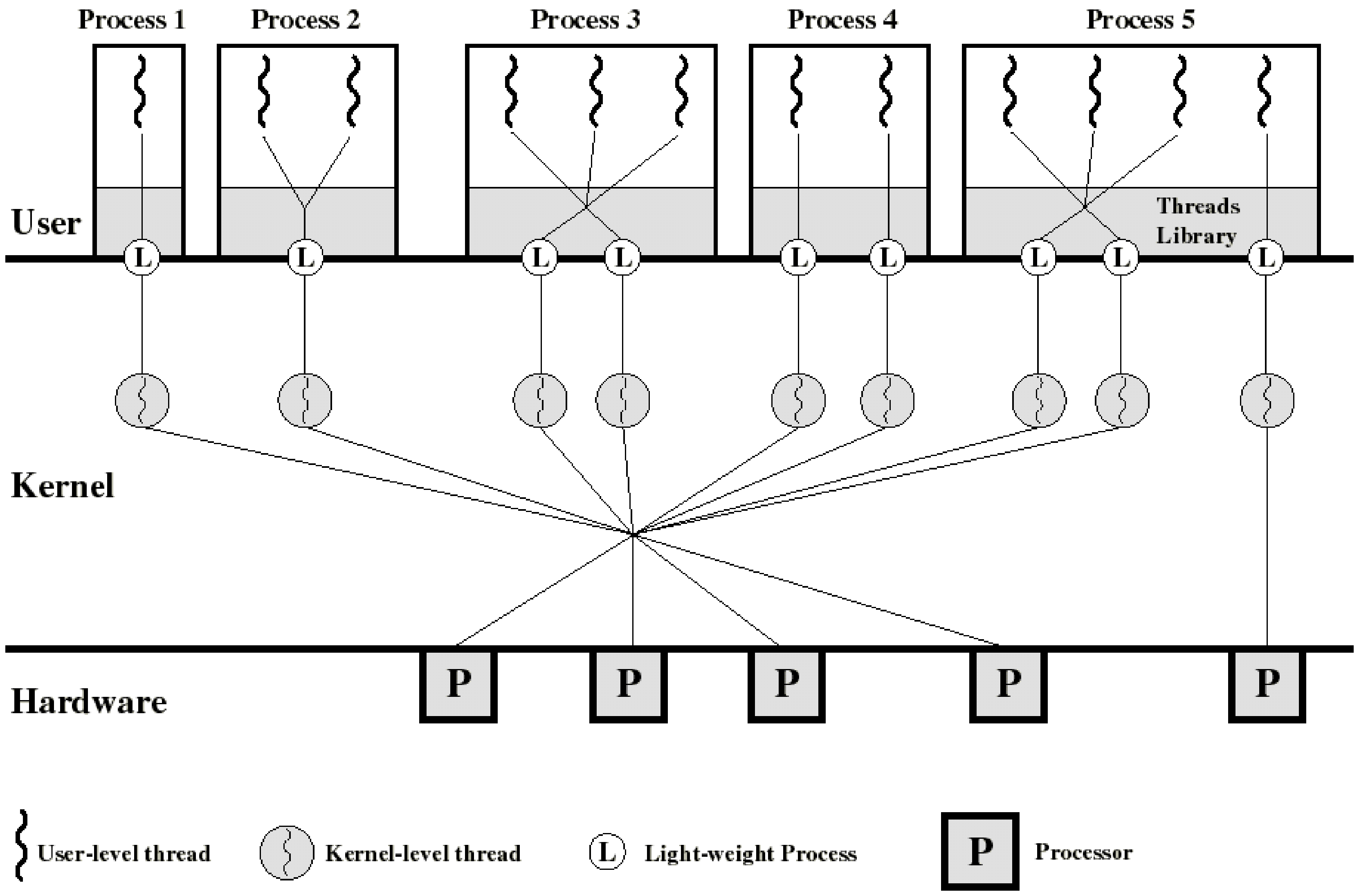
- Common software architectures of multiprocessor systems:
  - Separate supervisor configuration
    - Common in clustered systems
    - May only share limited resources
  - Master-Slave configuration
    - One CPU runs the OS, others run only applications
    - OS CPU may be a bottleneck, may fail
  - Symmetric configuration (SMP)
    - One OS runs everywhere
    - Each processor can do all (most) operations

Multiprocessor Operating Systems

- SMP systems are most popular today
  - Clustered systems are generally a collection of SMP systems that share a set of distributed services
- OS issues to consider
  - Execution units (threads)
  - Synchronization
  - CPU scheduling
  - Memory management
  - Reliability and fault tolerance

Multiprocessor Operating Systems

- Threads (execution units)
  - Address space utilization
  - Platform implementation
    - User level threads (M x 1 or M x N, Tru64Unix, HP-UX)
      - Efficient
      - Complex
      - Course grain control
    - Kernel level threads (1 X 1, Linux, Windows)
      - Expensive
      - Less complex
      - Fine grain control



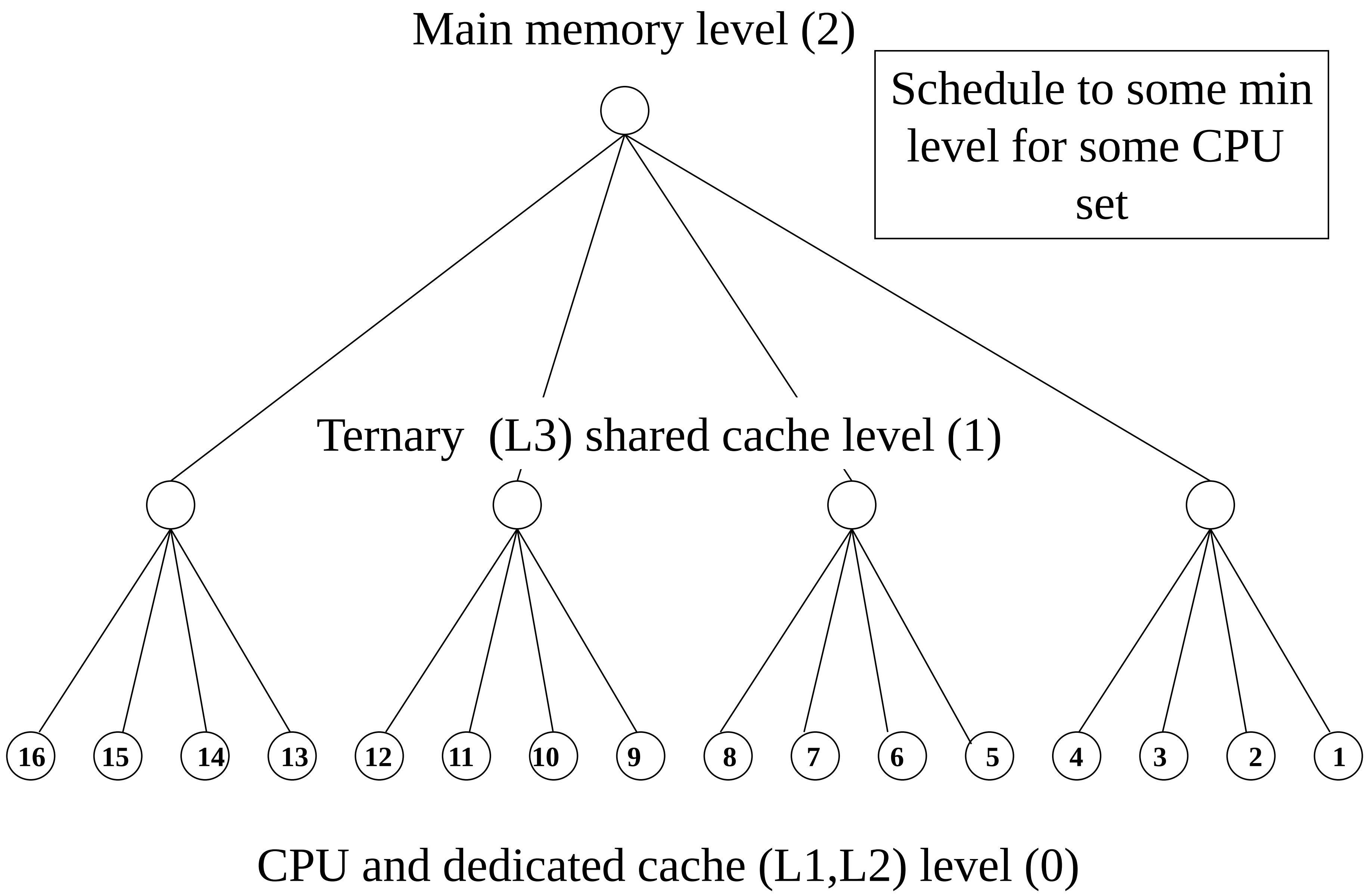
Process 2 is equivalent to a pure ULT approach
Process 4 is equivalent to a pure KLT approach
We can specify a different degree of parallelism (process 3 and 5)

Multiprocessor Operating Systems

- Synchronization issues
  - Interrupt disable no longer sufficient
  - Spin locks required
    - Software solutions like Peterson's algorithm are required when hardware platform only offers simple memory interlock
    - Hardware assist needed for efficient synchronization solutions
      - Test-and-set type instructions
        - » Intel XCHG instruction
        - » Motorola 88110 XMEM instruction
      - Bus lockable instructions
        - » Intel CMPXCHG8B instruction

Multiprocessor Operating Systems

- Processor scheduling issues
  - Schedule at the process or thread level ?
  - Which processors can an executable entity be scheduled to ?
    - Cache memory hierarchies play a major role
      - Affinity scheduling and cache footprint
    - Can the OS make good decisions without application hints ?
      - Applications understand how threads use memory, the OS does not
    - What type of scheduling policies will the OS provide ?
      - Time sharing, soft/hard real time, etc.


Main memory level (2)
Schedule to some min level for some CPU set
Ternary (L3) shared cache level (1)
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
CPU and dedicated cache (L1,L2) level (0)

Multiprocessor Operating Systems

- Memory management issues
  - Physical space deployment (UMA, NUMA)
  - Address space organization
  - Types of memory objects
    - Text, data/heap, stack, memory mapped files, shared memory segments, etc.
    - Shared vs private objects
    - Anonymous vs file objects
      - Swap space allocation strategies
  - Paging strategies and free memory list(s) configurations
  - Kernel data structure formats and locations

Reliability and Fault Tolerance Issues

- Operating systems must keep complex systems alive
  - Simple panics no longer make sense
- OS must be proactive in keeping the system up
  - Faulty components must be detected and isolated to the degree enabled by HW
  - The system must map its way around failed HW and continue to run
  - To the extent that the HW supports hot repair, the OS must provide recovery mechanisms