Principles of Parallel Computing

- Finding enough parallelism (Amdahl's Law)
- Granularity
- Locality
- Load balance
- Coordination and synchronization
- Performance modeling

All of these things makes parallel programming even harder than sequential programming.

What Is Parallelization ?

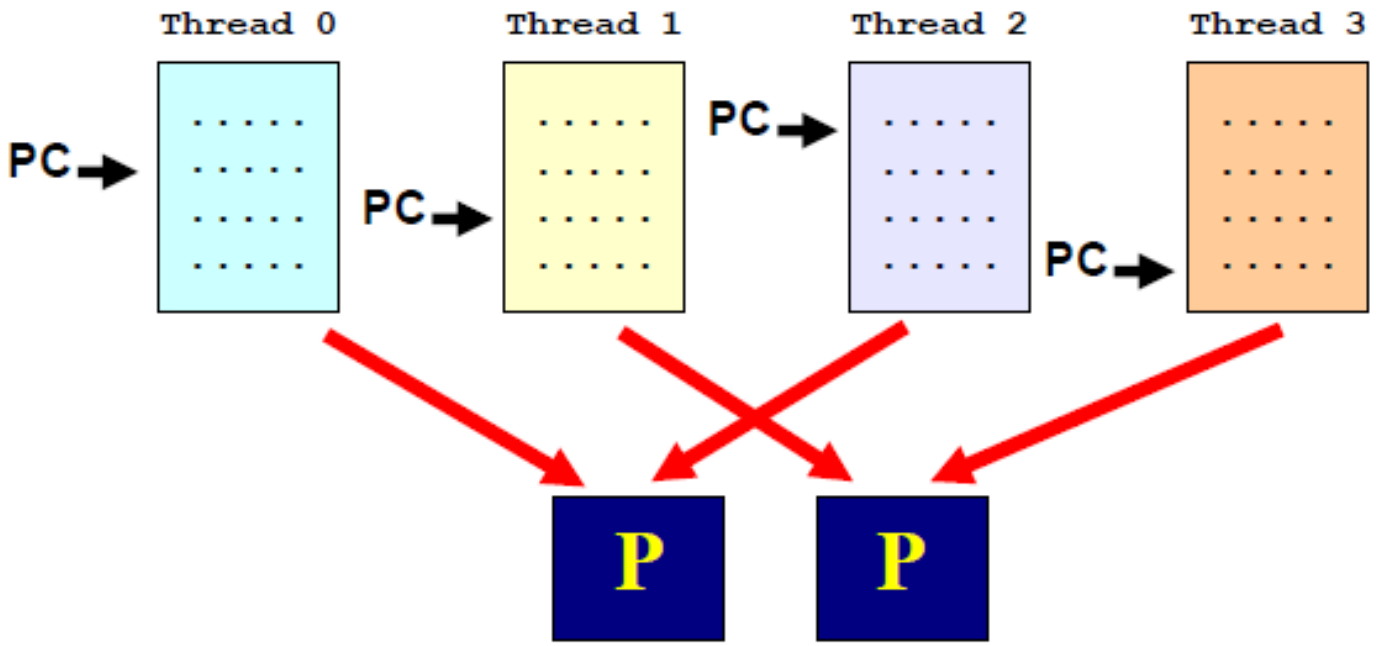
"Something" is parallel if there is a certain level of independence in the order of operations

In other words, it doesn't matter in what order those operations are performed

- *A sequence of machine instructions*
- *A collection of program statements*
- *An algorithm*
- *The problem you're trying to solve*

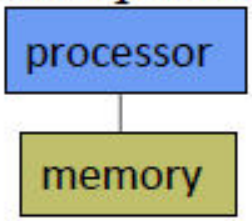
What is a Thread ?

- *Loosely said, a thread consists of a series of instructions with it's own program counter ("PC") and state*
- *A parallel program executes threads in parallel*
- *These threads are then scheduled onto processors*

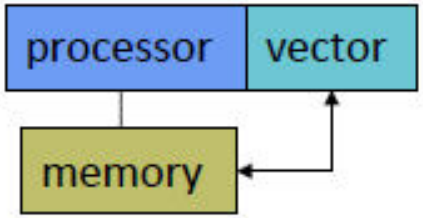


Parallel Architecture Types

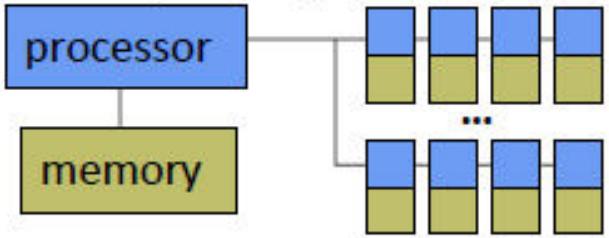
- Uniprocessor
  - Scalar processor



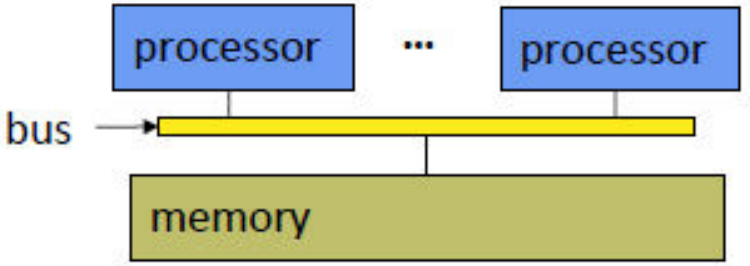
  - Vector processor



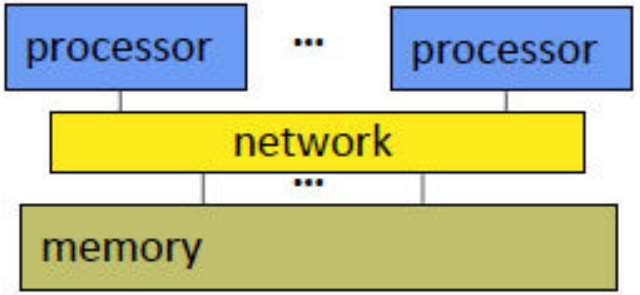
  - Single Instruction Multiple Data (SIMD)



- Shared Memory Multiprocessor (SMP)
  - Shared memory address space
  - Bus-based memory system

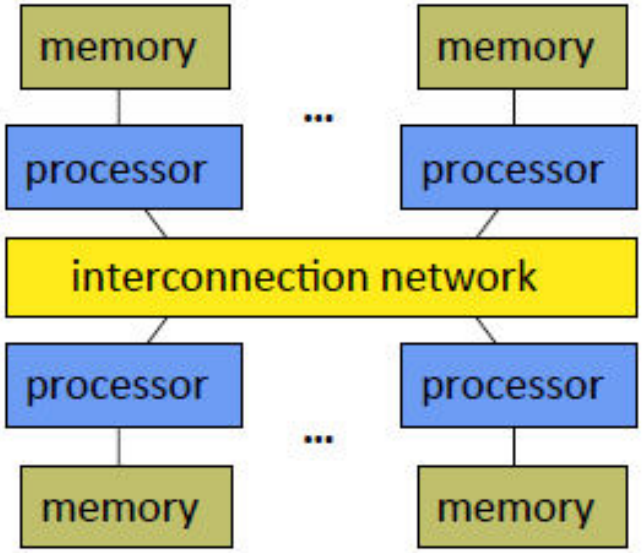


  - Interconnection network



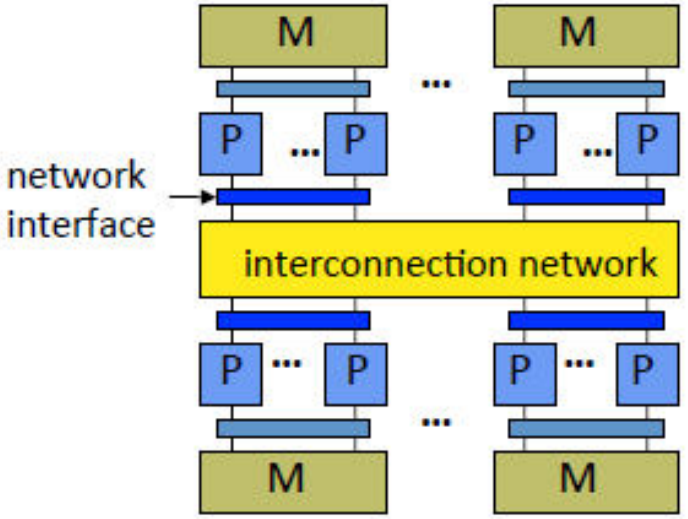
Parallel Architecture Types (2)

- Distributed Memory Multiprocessor
  - Message passing between nodes



  - Massively Parallel Processor (MPP)
    - Many, many processors

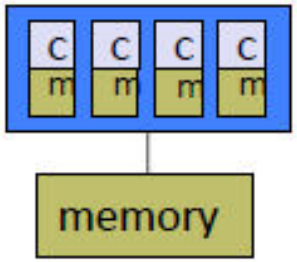
- Cluster of SMPs
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes



  - Can also be regarded as MPP if processor number is large

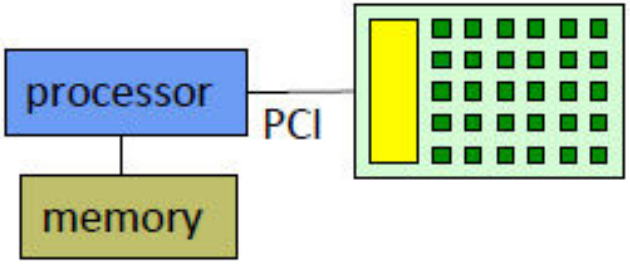
Parallel Architecture Types (3)

- Multicore
  - Multicore processor

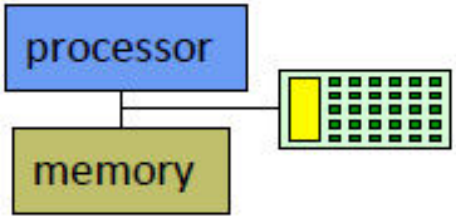


cores can be hardware multithreaded (hyperthread)

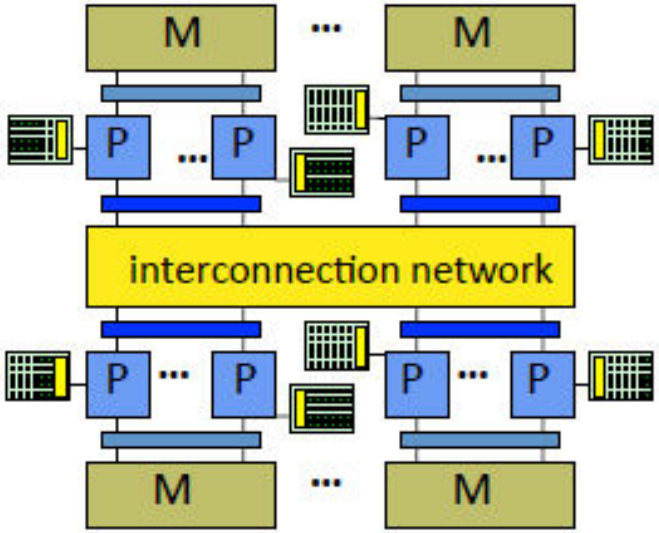
  - GPU accelerator



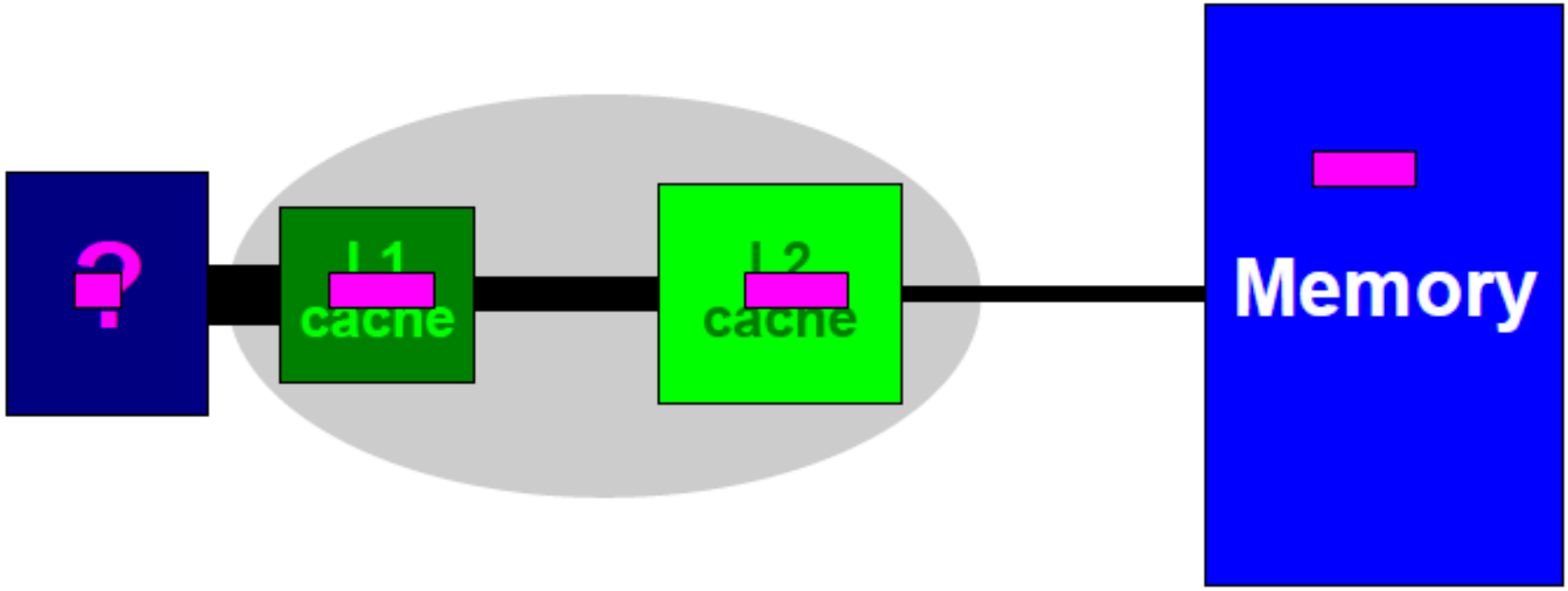
  - "Fused" processor accelerator



- Multicore SMP+GPU Cluster
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes
  - GPU accelerators attached



Typical cache based system

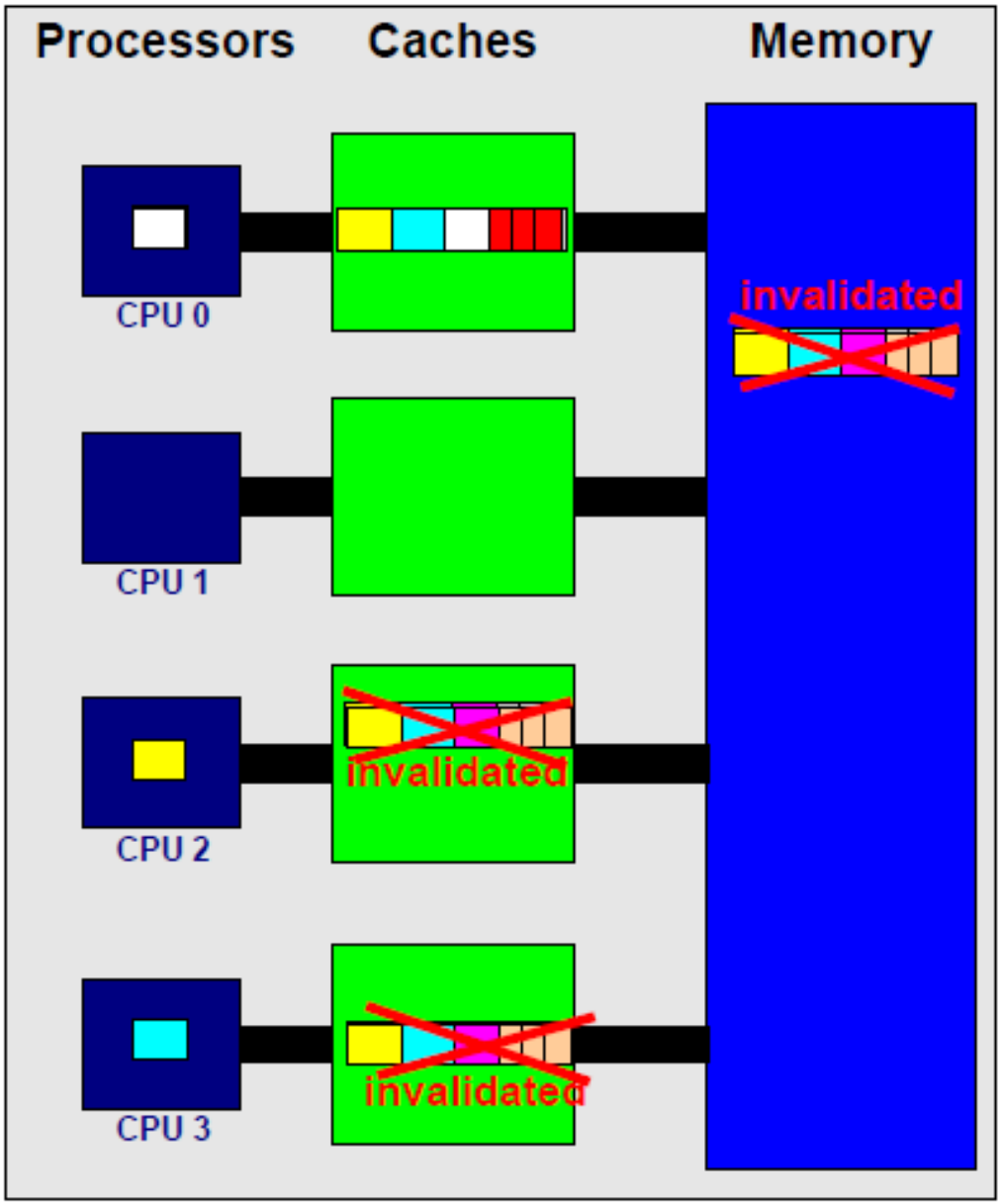


Caches in a parallel computer

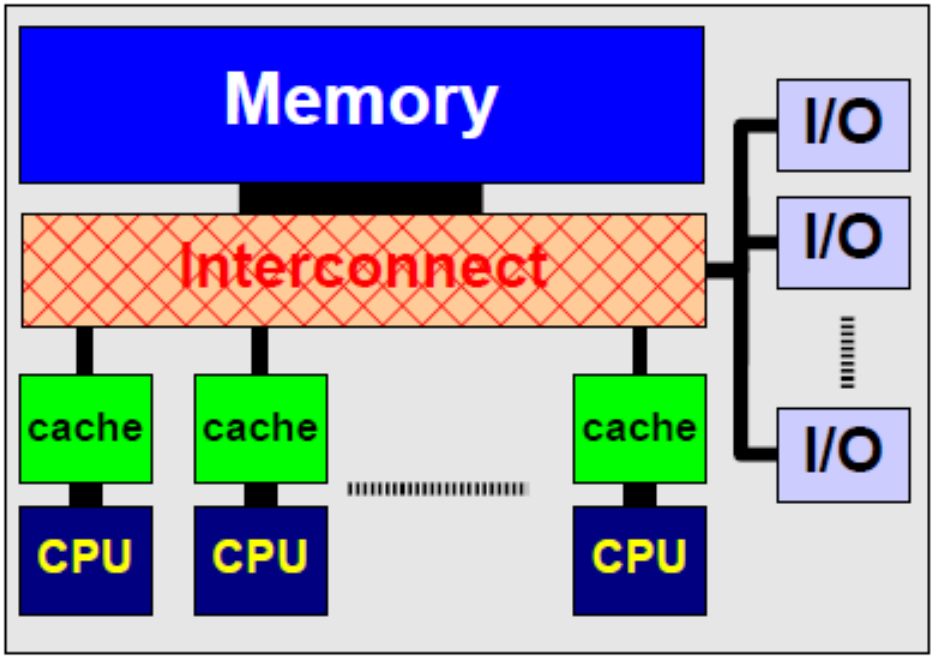
- *A cache line starts in memory*
- *Over time multiple copies of this line may exist*

Cache Coherence ("cc"):

- ✔ *Tracks changes in copies*
- ✔ *Makes sure correct cache line is used*
- ✔ *Different implementations possible*
- ✔ *Need hardware support to make it eff cient*



Uniform Memory Access (UMA)



- *Also called "SMP" (Symmetric Multi Processor)*
- *Memory Access time is Uniform for all CPUs*
- *CPU can be multicore*
- *Interconnect is "cc":*
  - *Bus*
  - *Crossbar*
- *No fragmentation - Memory and I/O are shared resources*

Pro

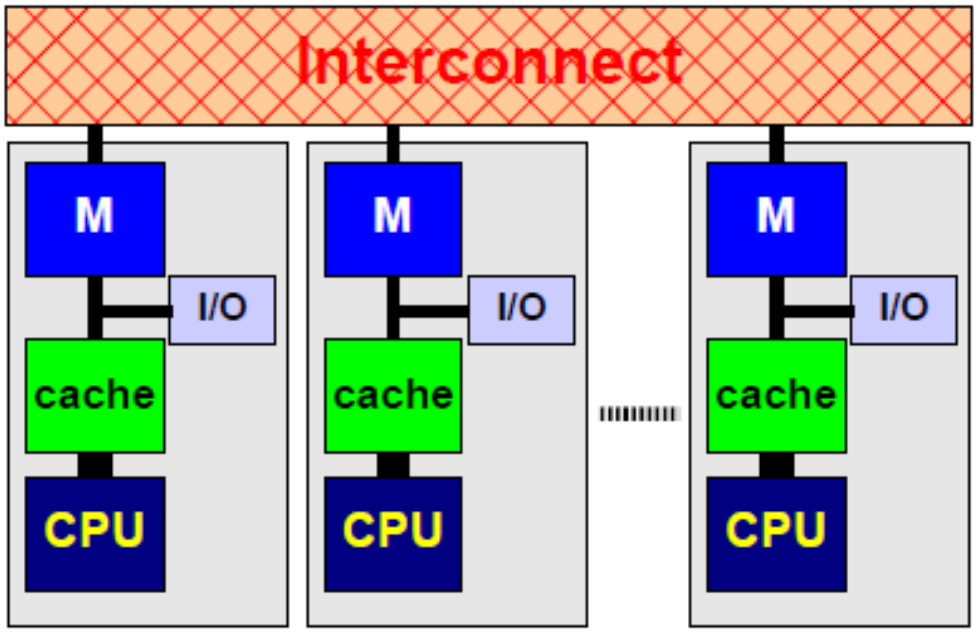
- ✔ *Easy to use and to administer*
- ✔ *Eff cient use of resources*

Con

- ✔ *Said to be expensive*
- ✔ *Said to be non-scalable*

- **Uniform memory access** (**UMA**) is a shared memory architecture used in parallel computers.
- All the processors in the UMA model share the physical memory uniformly.
- In an UMA architecture, access time to a memory location is independent of which processor makes the request or which memory chip contains the transferred data.
- Uniform memory access computer architectures are often contrasted with non uniform (NUMA) architectures.
- In the UMA architecture, each processor may use a private cache. Peripherals are also shared in some fashion.
- The UMA model is suitable for general purpose and time sharing applications by multiple users.
- It can be used to speed up the execution of a single large program in time critical applications.

NUMA



- *Also called "Distributed Memory" or NORMA (No Remote Memory Access)*
- *Memory Access time is Non-Uniform*
- *Hence the name "NUMA"*
- *Interconnect is not "cc":*
  - *Ethernet, Inf niband, etc, ...*
- *Runs 'N' copies of the OS*
- *Memory and I/O are distributed resources*

Pro

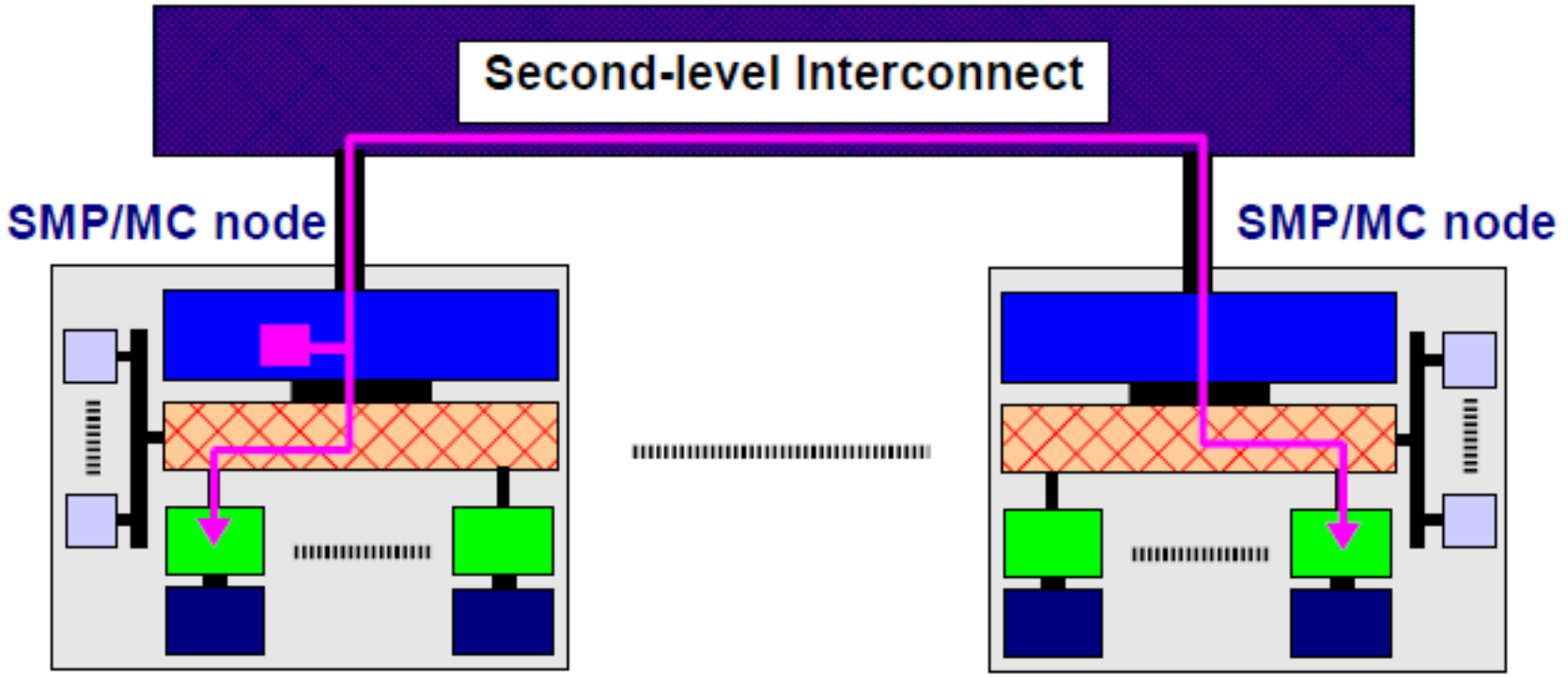
- ✔ *Said to be cheap*
- ✔ *Said to be scalable*

Con

- ✔ *Diff cult to use and administer*
- ✔ *In-eff cient use of resources*

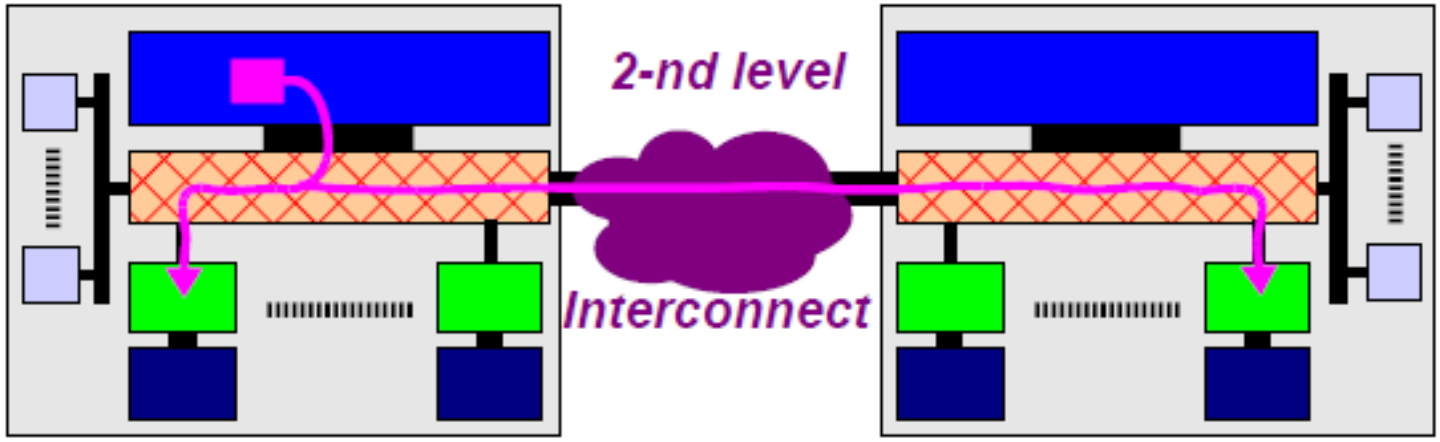
- **Non-uniform memory access** (**NUMA**) is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor.
- Under NUMA, a processor can access its own local memory faster than non-local memory (memory local to another processor or memory shared between processors).
- The benefits of NUMA are limited to particular workloads, notably on servers where the data is often associated strongly with certain tasks or users.

The Hybrid Architecture



- ***Second-level interconnect is not cache coherent***
  - ***Ethernet, Inf niband, etc, ...***
- ***Hybrid Architecture with all Pros and Cons:***
  - ***UMA within one SMP/Multicore node***
  - ***NUMA across nodes***

cc-NUMA



- *Two-level interconnect:*
  - *UMA/SMP within one system*
  - *NUMA between the systems*
- *Both interconnects support cache coherence i.e. the system is fully cache coherent*
- *Has all the advantages ('look and feel') of an SMP*
- *Downside is the Non-Uniform Memory Access time*

"Automatic" Parallelism in Modern Machines

- Bit level parallelism
  - —within floating point operations, etc.
- Instruction level parallelism (ILP)
  - —multiple instructions execute per clock cycle
- Memory system parallelism
  - —overlap of memory operations with computation
- OS parallelism
  - —multiple jobs run in parallel on commodity SMPs

Limits to all of these -- for very high performance, need user to identify, schedule and coordinate parallel tasks

Amdahl's Law

Amdahl's Law places a strict limit on the speedup that can be realized by using multiple processors. Two equivalent expressions for Amdahl's Law are given below:

$t_N = (f_p/N + f_s)t_1$ Effect of multiple processors on run time

$S = 1/(f_s + f_p/N)$ Effect of multiple processors on speedup

Where:

f_s = serial fraction of code

f_p = parallel fraction of code = 1 - f_s

N = number of processors

Overhead of Parallelism

- Given enough parallel work, this is the biggest barrier to getting desired speedup
- Parallelism overheads include:
  - —cost of starting a thread or process
  - —cost of communicating shared data
  - —cost of synchronizing
  - —extra (redundant) computation
- Each of these can be in the range of milliseconds (=millions of flops) on some systems
- Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (I.e. large granularity), but not so large that there is not enough parallel work

Load Imbalance

- Load imbalance is the time that some processors in the system are idle due to
  - —insufficient parallelism (during that phase)
  - —unequal size tasks
- Examples of the latter
  - —adapting to “interesting parts of a domain”
  - —tree-structured computations
  - —fundamentally unstructured problems
- Algorithm needs to balance load

Parallelism Is Everywhere

Multiple levels of parallelism:

Granularity	*Technology*	*Programming Model*
Instruction Level	*Superscalar*	*Compiler*
Chip Level	*Multicore*	*Compiler, OpenMP, MPI*
System Level	*SMP/cc-NUMA*	*Compiler, OpenMP, MPI*
Grid Level	*Cluster*	*MPI*

Loosely Coupled Multiprocessor System

- It is a type of multiprocessing system in which, There is distributed memory instead of shared memory.
- In loosely coupled multiprocessor system, data rate is low rather than tightly coupled multiprocessor system.
- In loosely coupled multiprocessor system, modules are connected through MTS (Message transfer system) network.
- Data dependency will be less, so that maximum parallelism can be achieved.

Tightly Coupled Multiprocessor System:

- It is a type of multiprocessing system in which, There is shared memory.
- In tightly coupled multiprocessor system, data rate is high rather than loosely coupled multiprocessor system.
- Data dependency will be high, so that parallelism will be minimized.

S.NO	LOOSELY COUPLED	TIGHTLY COUPLED
1.	There is distributed memory in loosely coupled multiprocessor system.	There is shared memory, in tightly coupled multiprocessor system.
2.	Loosely Coupled Multiprocessor System has low data rate.	Tightly coupled multiprocessor system has high data rate.
3.	The cost of loosely coupled multiprocessor system is less.	Tightly coupled multiprocessor system is more costly.
4.	In loosely coupled multiprocessor system, modules are connected through **Message transfer system** network.	While there is PMIN, IOPIN and ISIN networks.
5.	In loosely coupled multiprocessor, Memory conflicts don't take place.	While tightly coupled multiprocessor system have memory conflicts.
6.	Efficient when tasks running on different processors, has minimal interaction.	Efficient for high-speed or real-time processing.