Parallel Programming (CDSC 604)

Addisu G. Semie, PhD
Asst. Prof.,Computational Science Program,
Addis Ababa University
Email: addisu.semie@aau.edu.et

Objectives

- Provide an insight into the architecture of current HPC systems
- Provide a working knowledge of Parallel programming, which they can apply to solve a variety of scientific problems.
- Use OpenMP and MPI to solve various physical problems.

Course outline

- Basic concepts of HPC systems and Architecture
- Overview of Parallel Programming
- Introduction to OpenMP
- Introduction to MPI

What is High-Performance Computing(HPC)?

- Is the use of parallel processing for running advanced application programs efficiently, reliably and quickly.
- We use the term HPC when we care how fast we get an answer.

HPC can happen on:

- Workstations, desktop, laptop, smart-phone
- Supercomputer
- Linux Cluster
- Grid or cloud

Why would we care about HPC?

- Scientific simulation and modeling drive the need for greater computing power.
- Single core processors are too small and too slow to run simulations
- Making processors with faster clock speeds is difficult due to cost and power/heat limitations
- Expensive to put huge memory on a single processor
- Solution: **parallel computing** - divide up the work among numerous linked systems

What determines performance?

- How fast is my CPU?
- How fast can I move data around?
- How well can I split work into pieces?

How fast is my CPU?

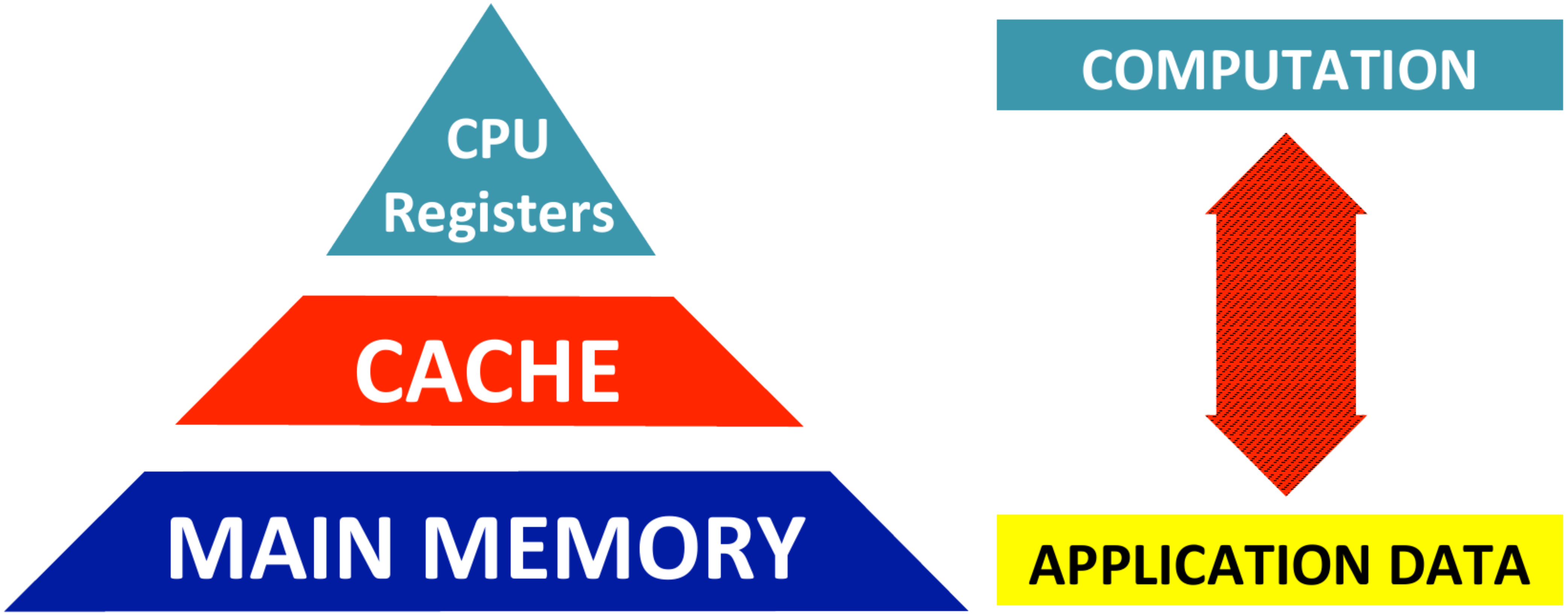
- CPU power is measured in FLOPS
  - number of floating point operations x second
  - FLOPS = number of cores x clock x FLOP/cycle

- FLOP/cycle is number of double precision addition and/or multiplications completed per clock
  - architectural limit (data load/store)
  - depend also by the instruction set supported

Ex. 2.5 Ghz x 8 FLOP/clock = 20 GigaFLOP/s

How fast is my CPU?

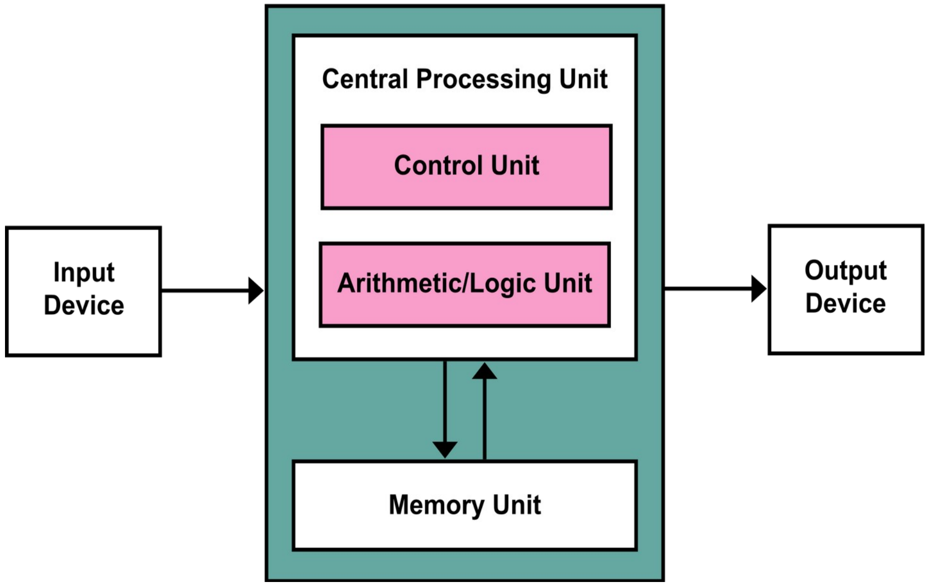
The CPU Memory Hierarchy



Von Neumann architecture

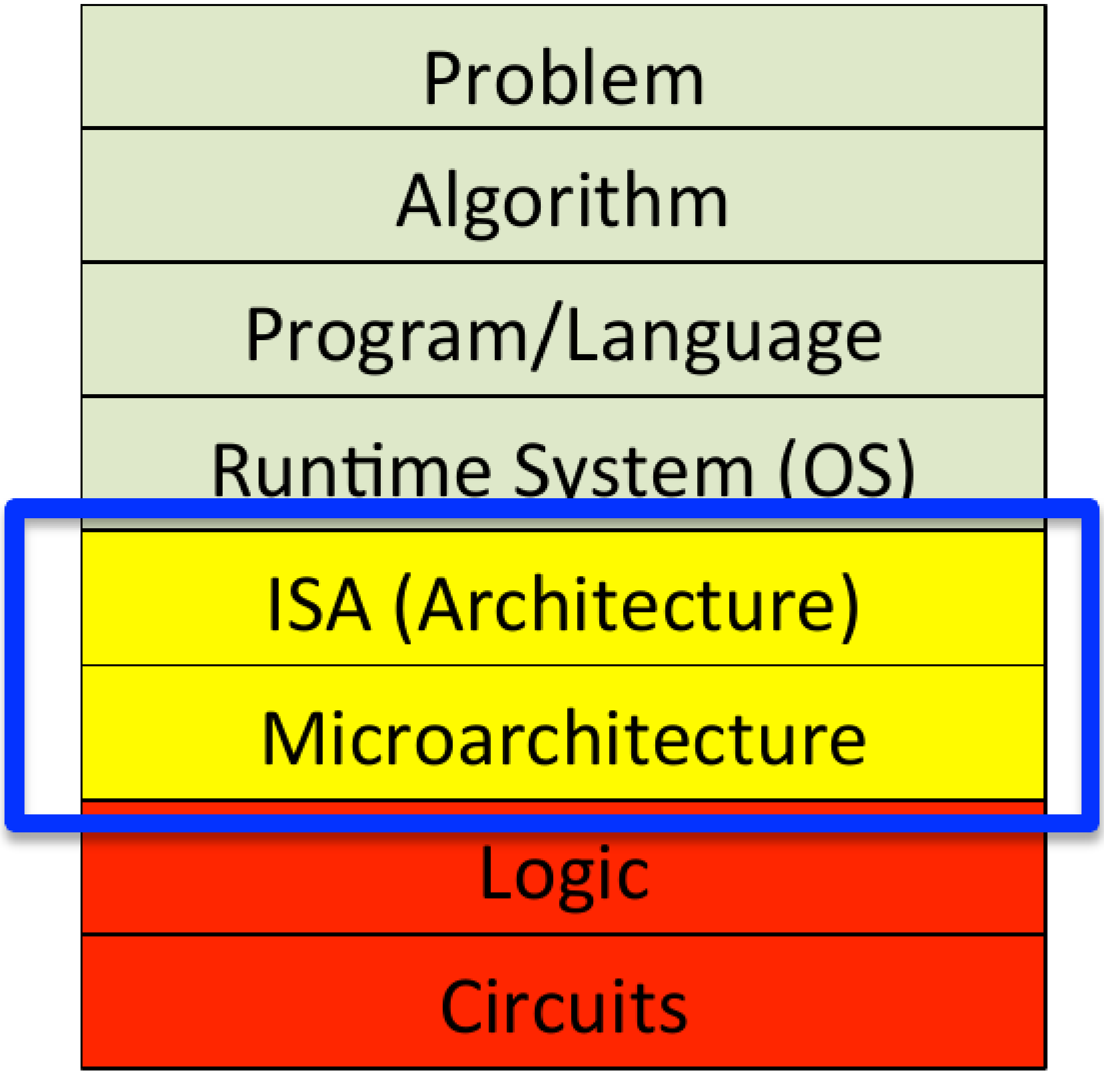
Comprises of four main components:

- Control Unit – fetch, decode, coordinates
- Arithmetic Logic Unit – basic arithmetic operations
- Memory – used to store instructions and data
- Input/Output – interface to the human operator

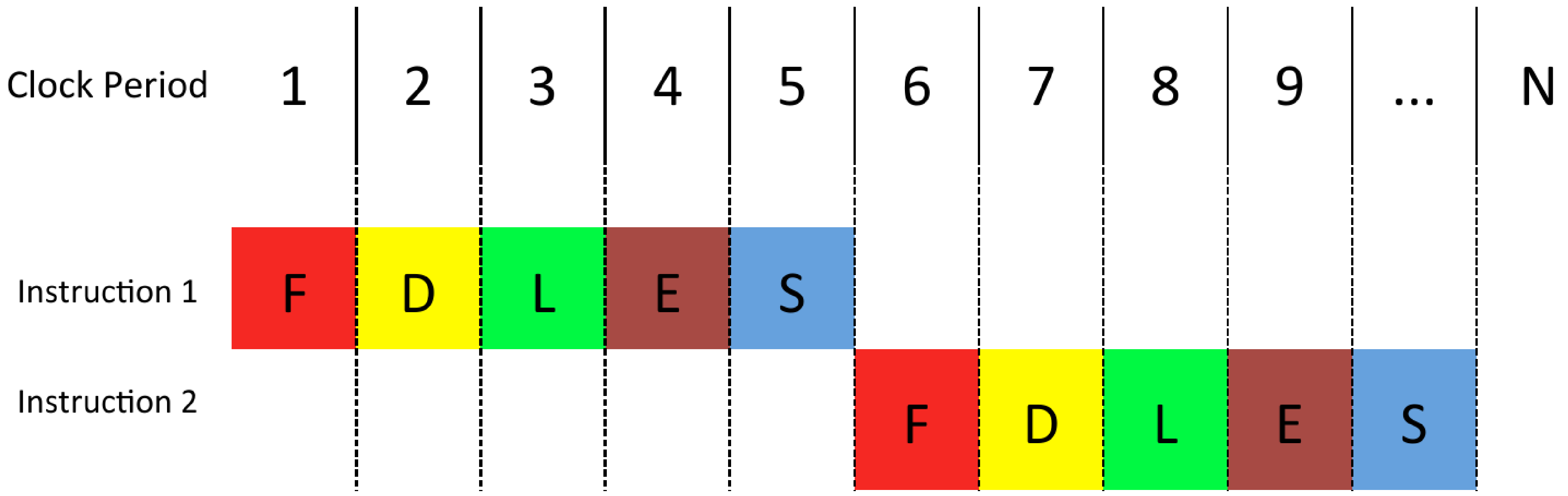


Instruction cycles

- Read an instruction and decode it
- Find any associated data that is needed to process the instruction
- Process the instruction
- Write the results out



Sequential Processing



Pipelining

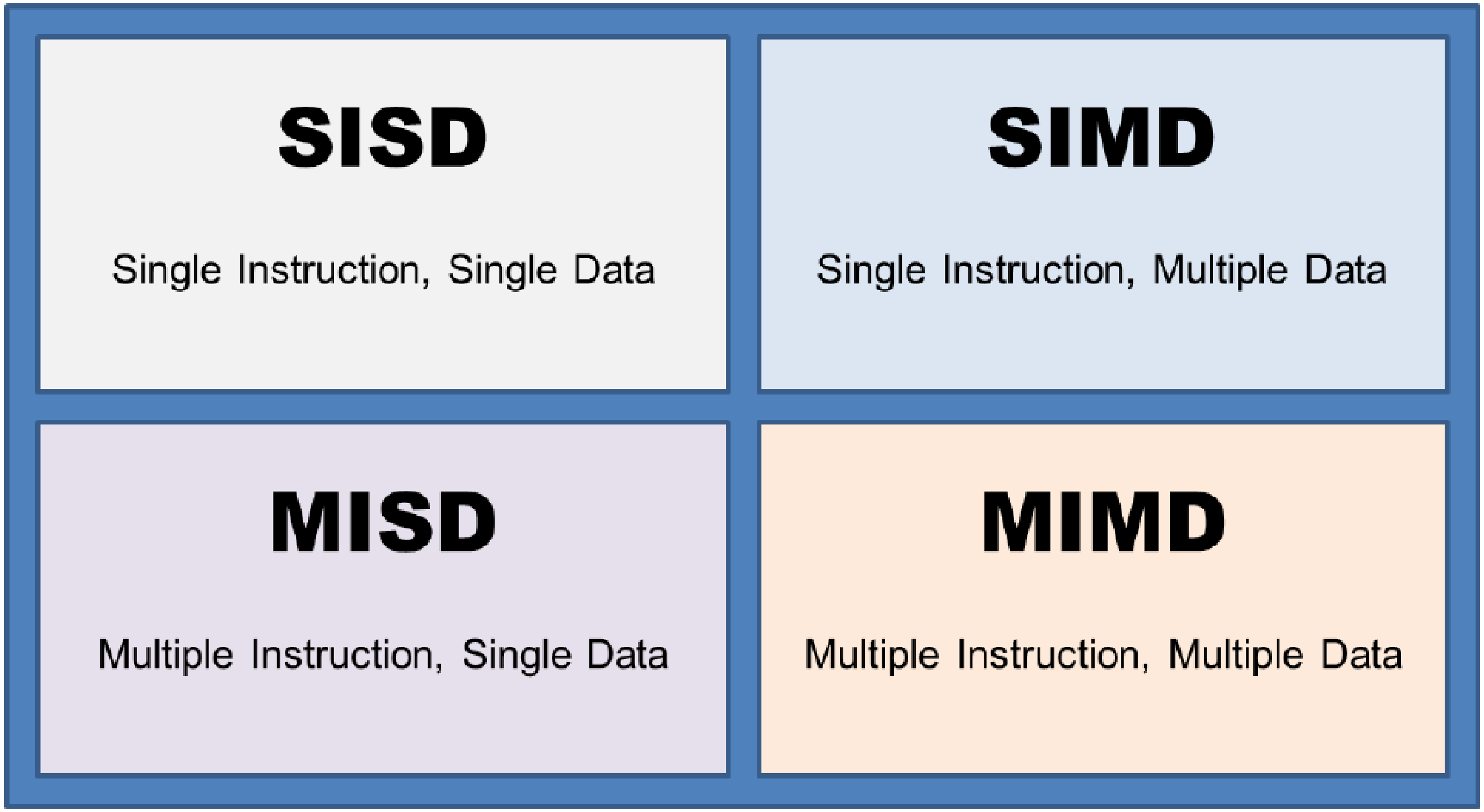
Clock Period	1	2	3	4	5	6	7	8	9	…	N
Instruction 1	F	D	L	E	S						
Instruction 2		F	D	L	E	S					
Instruction 3			F	D	L	E	S				
Instruction 4				F	D	L	E	S			
Instruction 5					F	D	L	E	S		

Superscalaring

Clock Period	1	2	3	4	5	6	7	8	9	...	N
Instruction 1	F	D	L	E	S						
Instruction 2	F	D	L	E	S						
Instruction 3		F	D	L	E	S					
Instruction 4		F	D	L	E	S					
Instruction 5			F	D	L	E	S				
Instruction 6			F	D	L	E	S				

Flynn's classical taxonomy

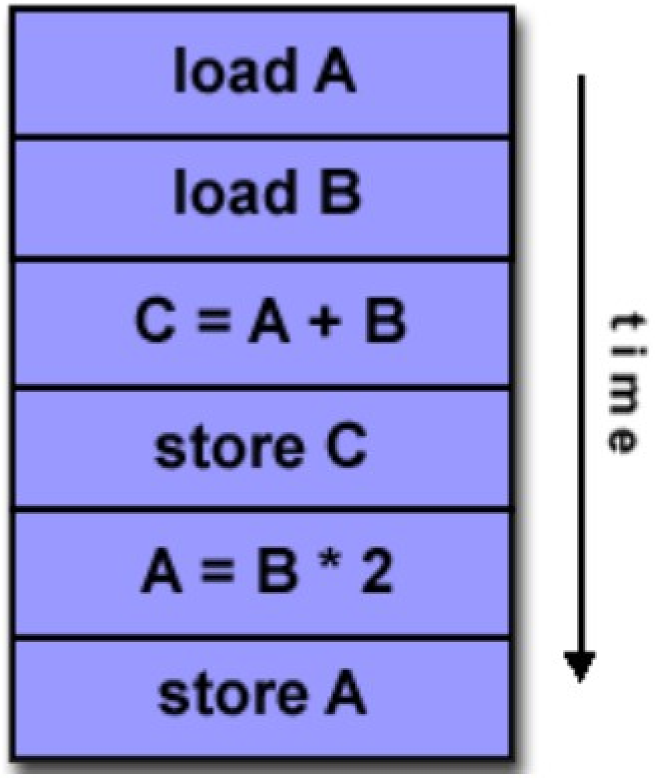
- It is a widely used classification since 1966
- Based on the number of concurrent instructions and data streams available in the architecture


SISD
Single Instruction, Single Data
SIMD
Single Instruction, Multiple Data
MISD
Multiple Instruction, Single Data
MIMD
Multiple Instruction, Multiple Data

Single Instruction, Single Data (SISD)

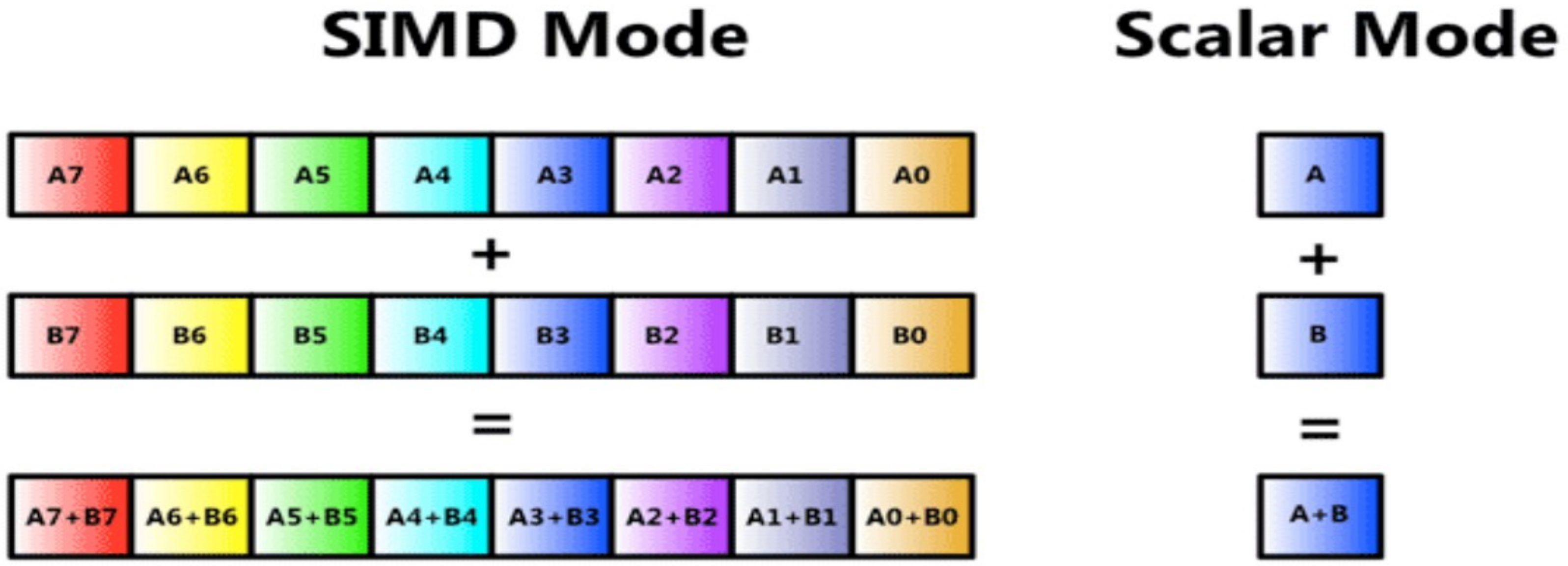
A serial (non-parallel) computer

- Single Instruction: Only one instruction stream is being acted on by the CPU during any one clock cycle
- Single Data: Only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest type of computer

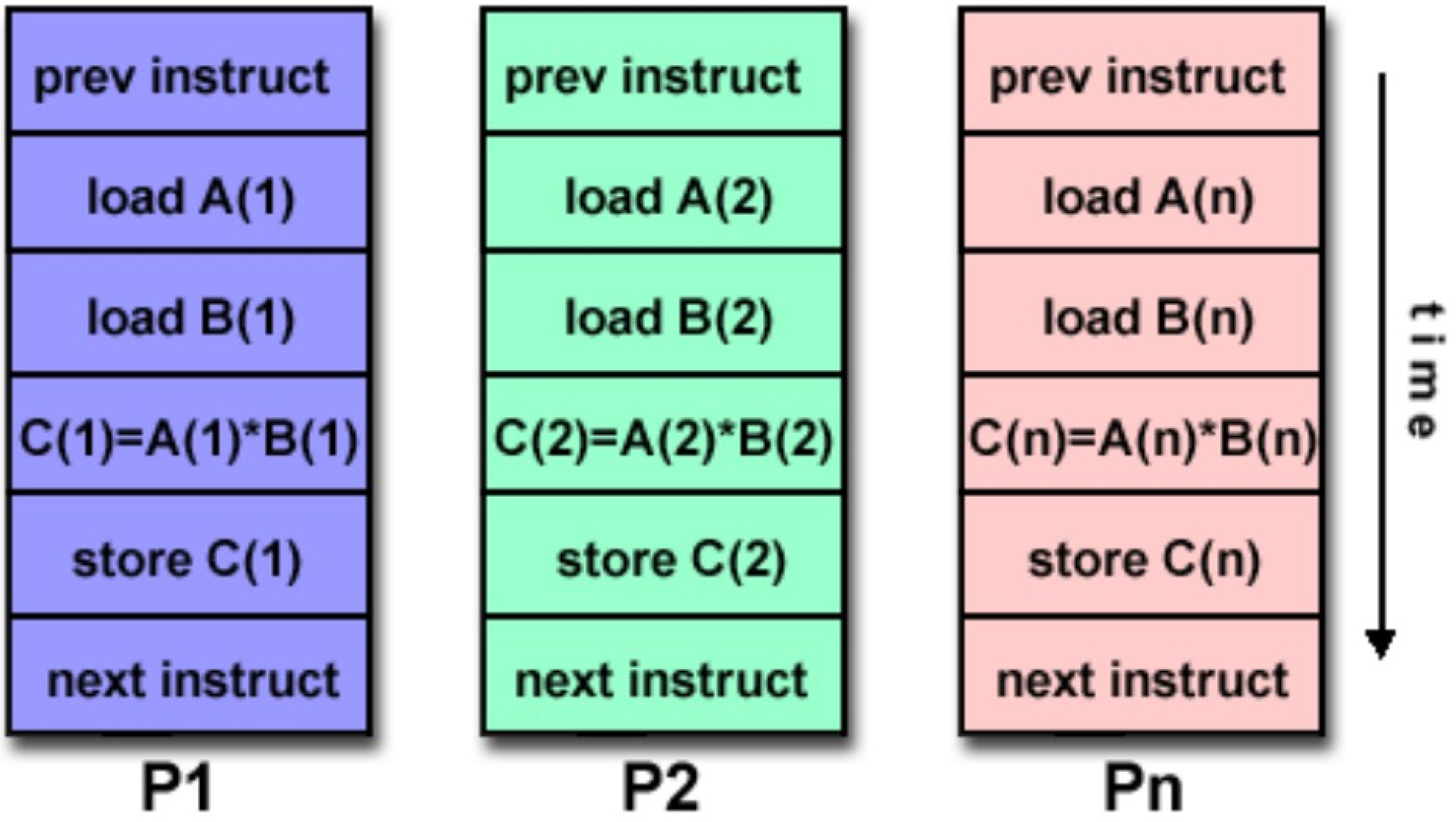


Single Instruction, Multiple Data (SIMD)

- A type of parallel computer that exploits multiple data sets against a single instruction
- Single Instruction: All processing units execute the same instruction at any given clock cycle
- Multiple Data: Each processing unit can operate on a different data element

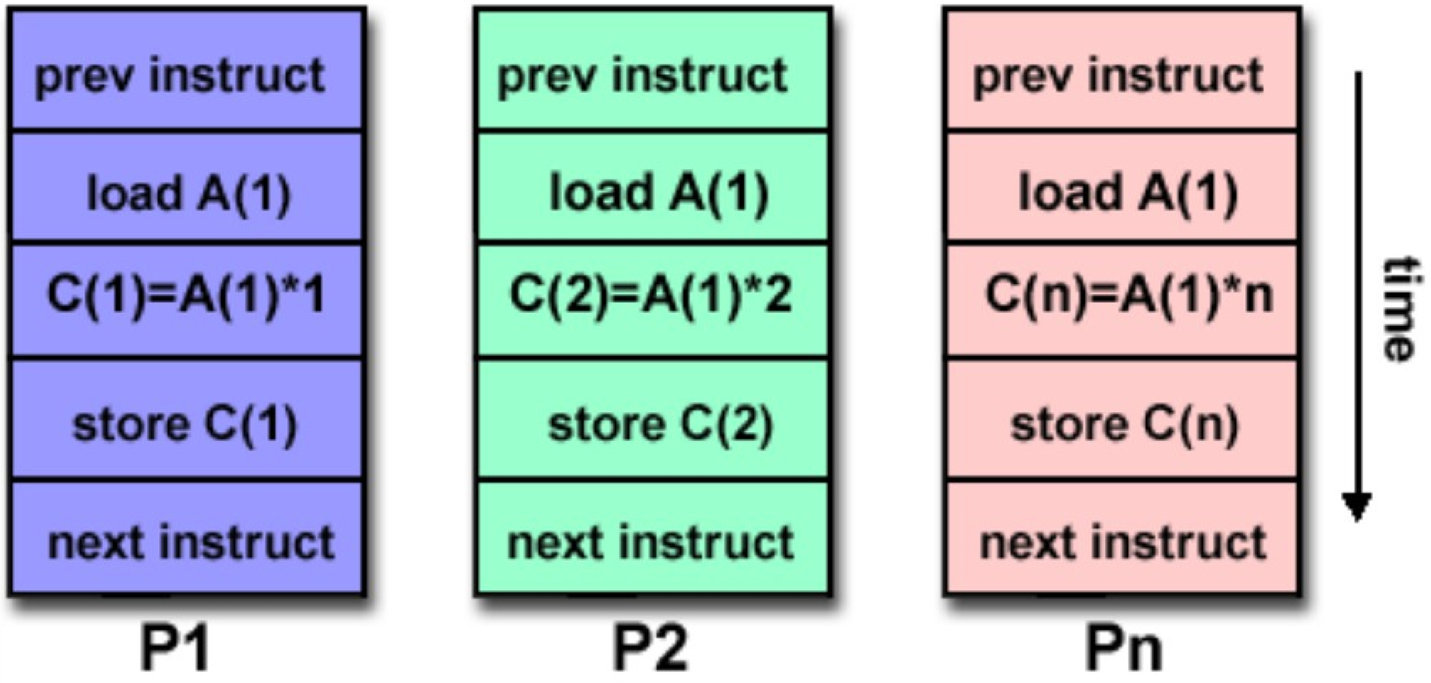


Single Instruction, Multiple Data (SIMD)



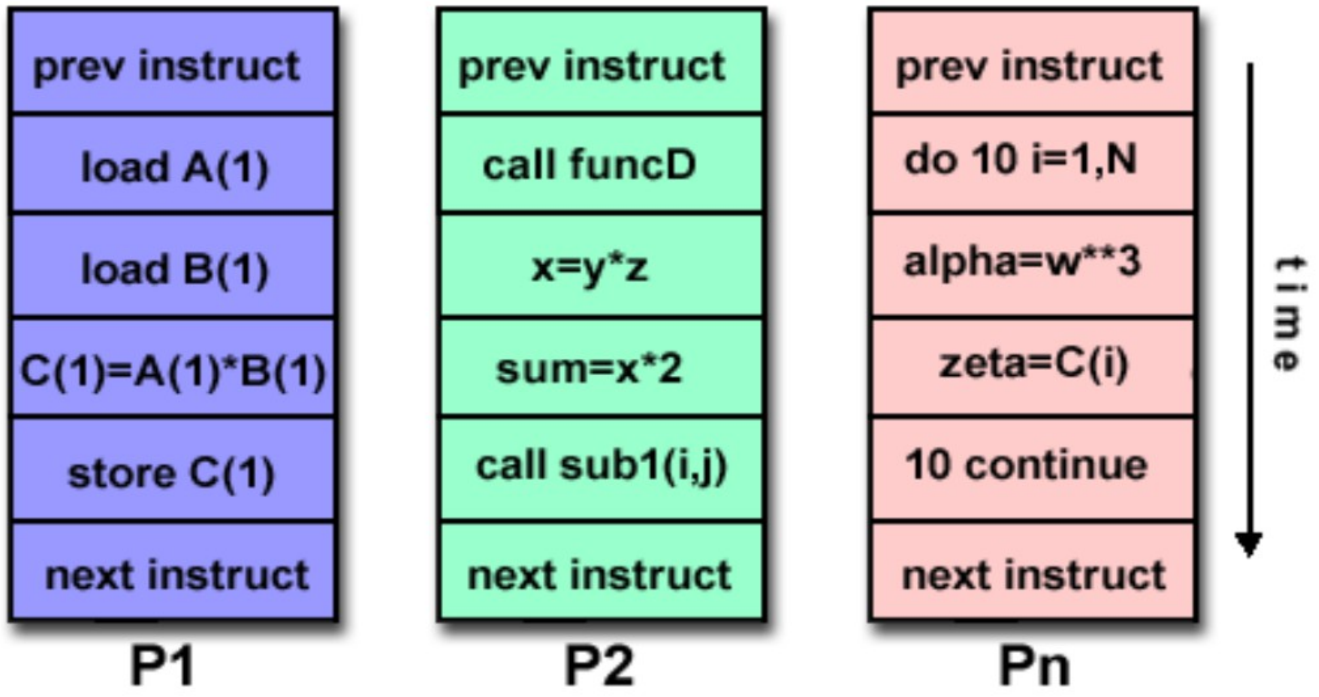
Multiple Instruction, Single Data (MISD)

- Multiple Instruction: Each processing unit operates on the data independently via separate instruction streams.
- Single Data: A single data stream is fed into multiple processing units.



Multiple Instruction, Multiple Data (MIMD)

- Multiple autonomous processors simultaneously executing different instructions on different data
- Multiple Instruction: Every processor may be executing a different instruction stream
- Multiple Data: Every processor may be working with a different data stream



Multiple Instruction, Multiple Data (MIMD)

- Most current supercomputers, networked parallel computer clusters, grids, clouds, multi-core PCs
- Many MIMD architectures also include SIMD execution sub-components

IBM POWER5

HP/Compaq Alphaserver

Intel IA32

AMD Opteron

Cray XT3

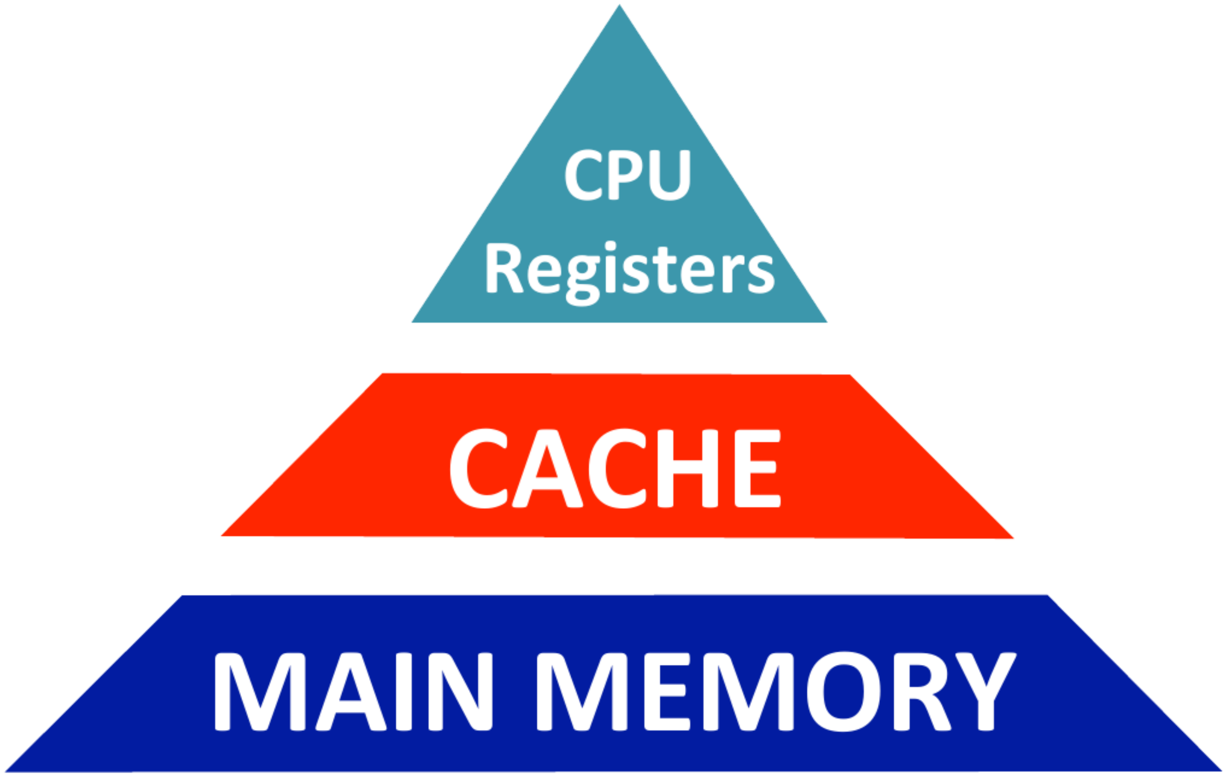
IBM BG/L

Performance Metrics

- When all CPU component work at maximum speed that is called peak of performance
  - Tech-spec normally describe the theoretical peak
  - Benchmarks measure the real peak
  - Applications show the real performance value
- CPU performance is measured as:
  - Floating point operations per seconds GFLOP/s
- But the real performance is in many cases mostly related to the memory bandwidth (GBytes/s)

Cache Memory

- Expensive (SRAM) high-speed memory
- Relatively low-capacity in regards to RAM
- Cache Memory are for Instructions (L1I) and for Data (L1D)



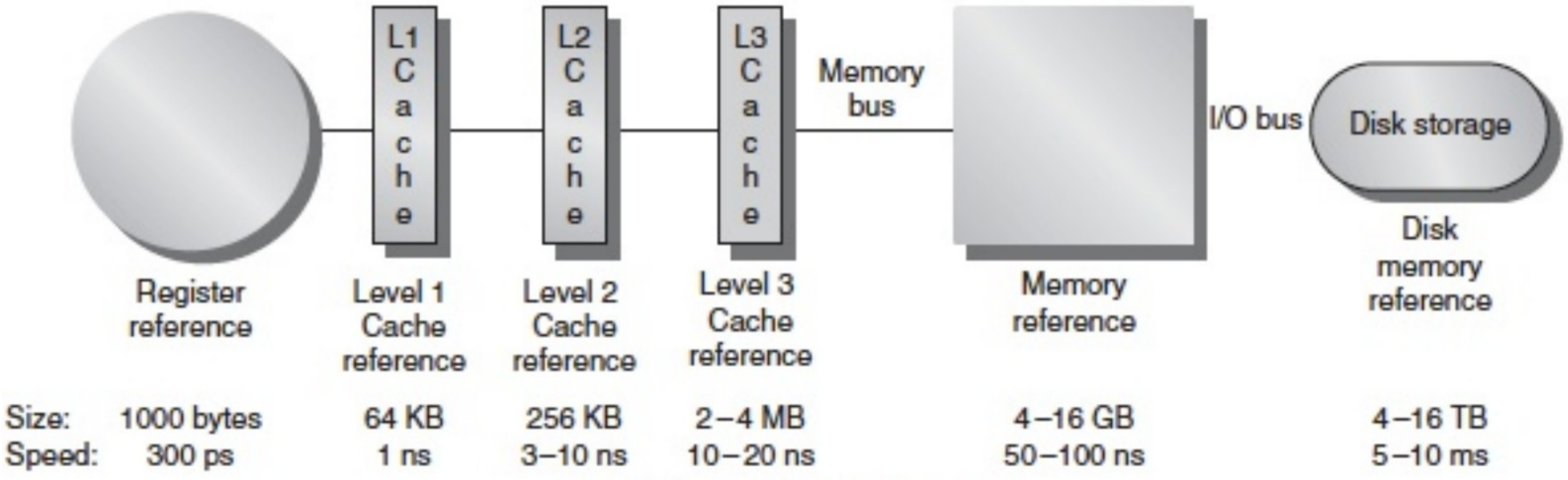
Cache Memory

- Modern CPU are designed with several levels of cache memories
- Designed for temporal/spatial locality
- Data is transferred to cache in blocks of fixed size, called cache lines.
- Operation of LOAD/STORE can lead at two different scenario:

  - cache hit

  - cache miss

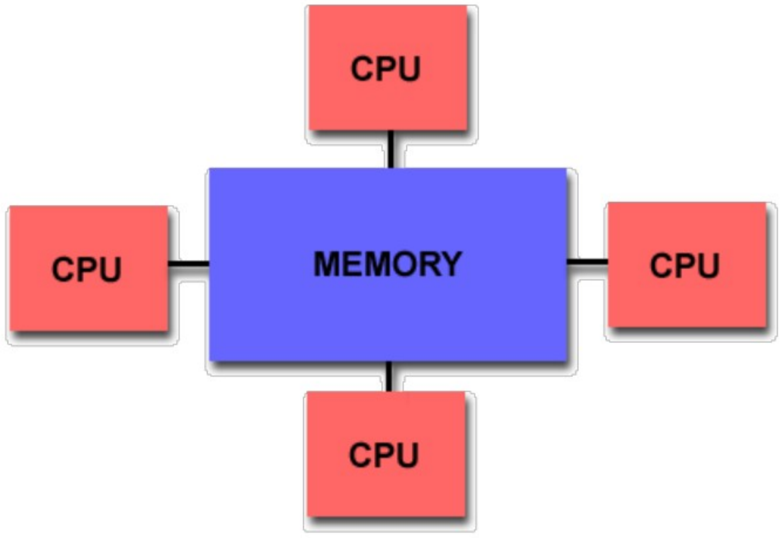
CPU Memory Hierarchy



	Register reference	Level 1 Cache reference	Level 2 Cache reference	Level 3 Cache reference	Memory reference	Disk memory reference
Size:	1000 bytes	64 KB	256 KB	2–4 MB	4–16 GB	4–16 TB
Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	5–10 ms

Parallel Computer Memory Architectures

- **Shared memory** - all processors access all memory as global address space.
- Multiple processors can operate independently but share the same memory resources.
- Changes in a memory location effected by one processor are visible to all other processors.



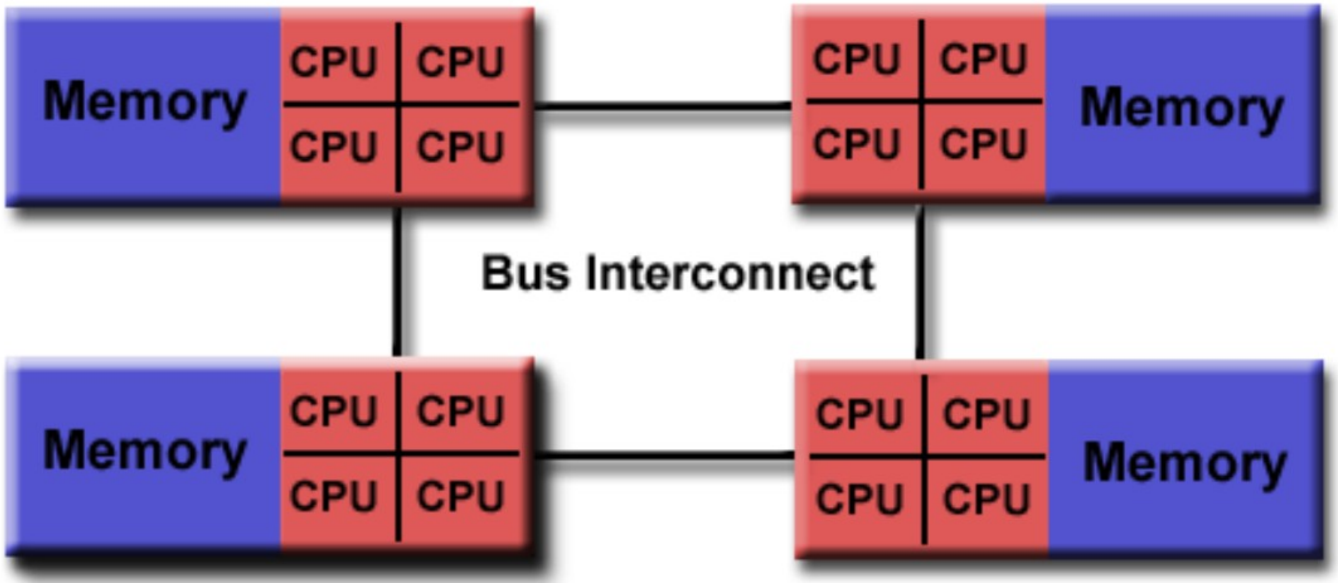
Shared memory machines can be divided into two main classes based upon memory access times: **UMA** and **NUMA**.

Uniform Memory Access (UMA)

- Most commonly represented today by Symmetric Multiprocessor (SMP) machines Identical processors
- Equal access and access times to memory
- **Cache coherence** is the uniformity of shared resource data that ends up stored in multiple local caches.
- Cache Coherent UMA (CC-UMA) - means if one processor updates a location in shared memory, all the other processors know about the update.
- Cache coherency is accomplished at the hardware level.

Non-Uniform Memory Access (NUMA)

- Often made by physically linking two or more SMPs
- One SMP can directly access memory of another SMP
- Not all processors have equal access time to all memories
- Memory access across link is slower
- If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA



Shared memory: Advantages

- A type of parallel computer that exploits multiple data sets against a single instruction Global address space provides a user-friendly programming perspective to memory

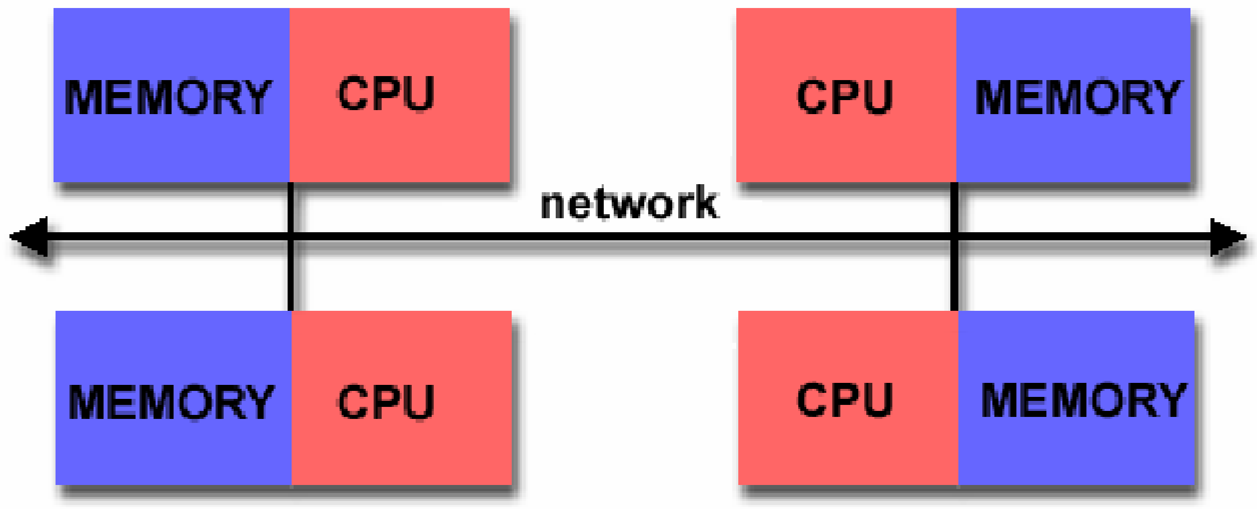
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

Shared memory: Disadvantages

- Primary disadvantage is the lack of scalability between memory and CPUs.
- Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
- Programmer responsibility for synchronization constructs that insure "correct" access of global memory.

Distributed memory

- Each processor has its own private memory
- Changes to processor’s local memory have no effect on the memory of other processors.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated.



Distributed memory: Advantages

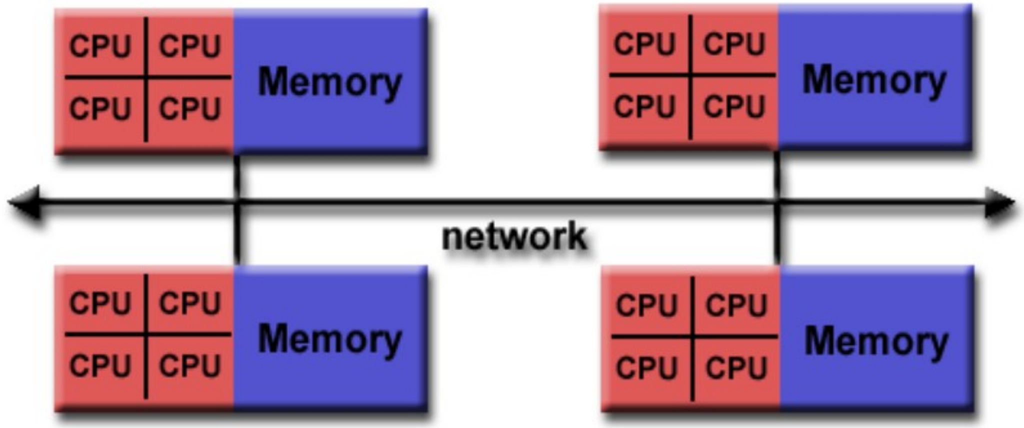
- Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
- Cost effectiveness: can use commodity, off-the-shelf processors and networking.

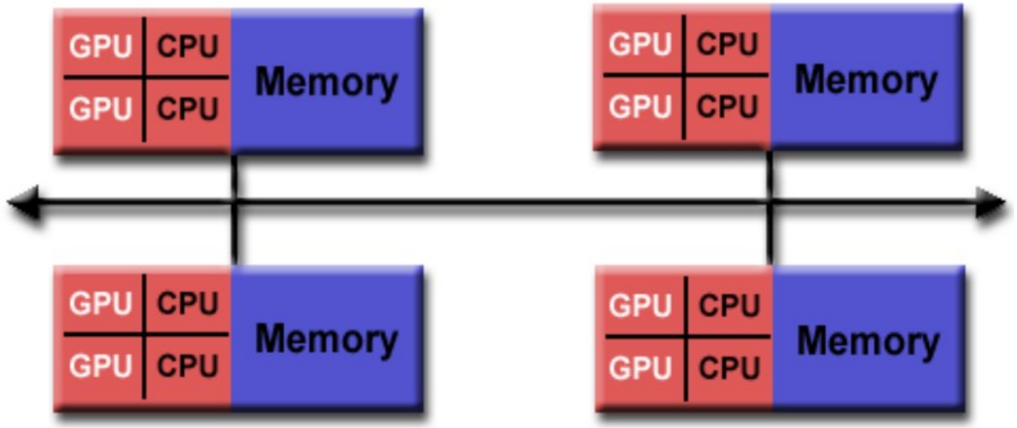
Distributed memory: disadvantages

- The programmer is responsible for many of the details associated with data communication between processors.
- It may be difficult to map existing data structures, based on global memory, to this memory organization.
- Non-uniform memory access (NUMA) times

Hybrid distributed-shared memory

- The largest and fastest computers in the world today employ both shared and distributed memory architectures.
- The shared memory component is usually a cache coherent SMP machine or graphics processing units.
- The Hybrid distributed memory component is the networking of multiple SMPs or GPU machines.





Hybrid distributed-shared memory

Advantages and Disadvantages:

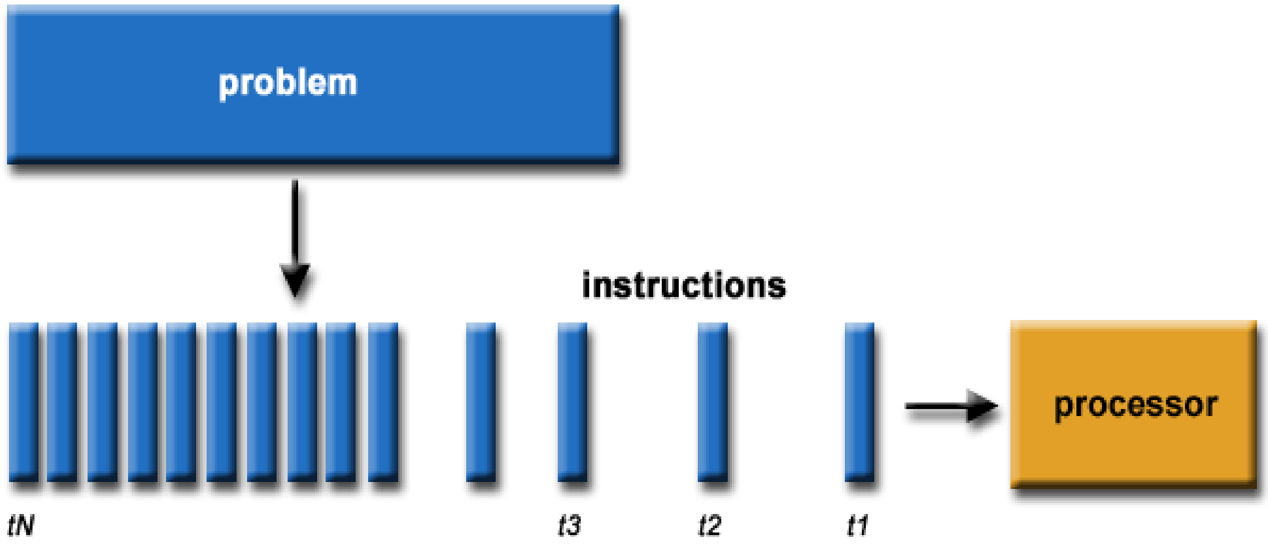
- Whatever is common to both shared and distributed memory architectures.
- Increased scalability is an important advantage
- Increased programmer complexity is an important disadvantage

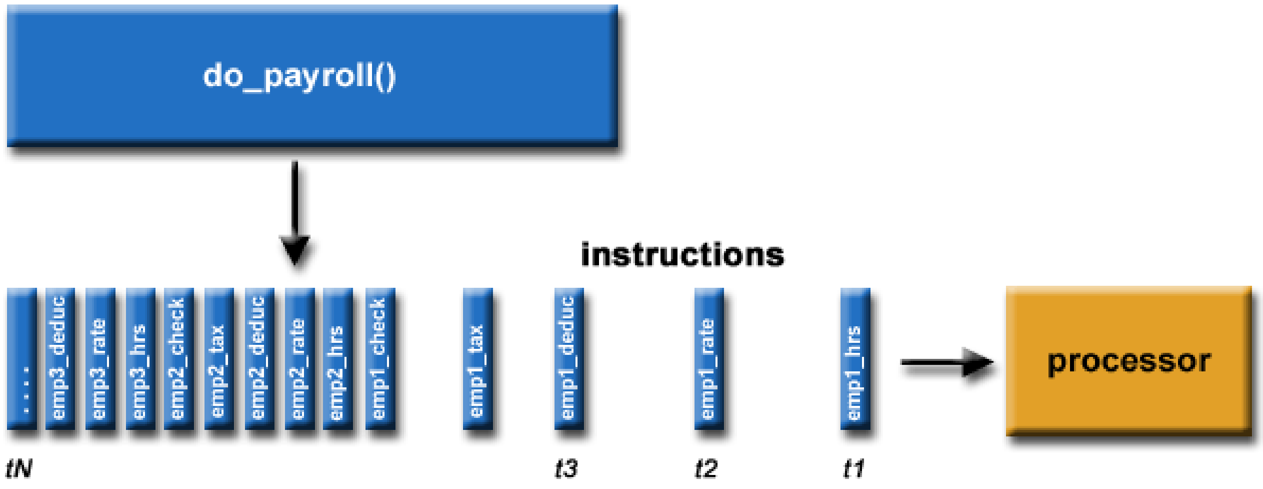
Overview of Parallel Programming

Serial Computing

Traditionally, software has been written
for ***serial*** computation: A problem is broken into a discrete series of instructions

- Instructions are executed sequentially one after another
- Executed on a single processor
- Only one instruction may execute at any moment in time

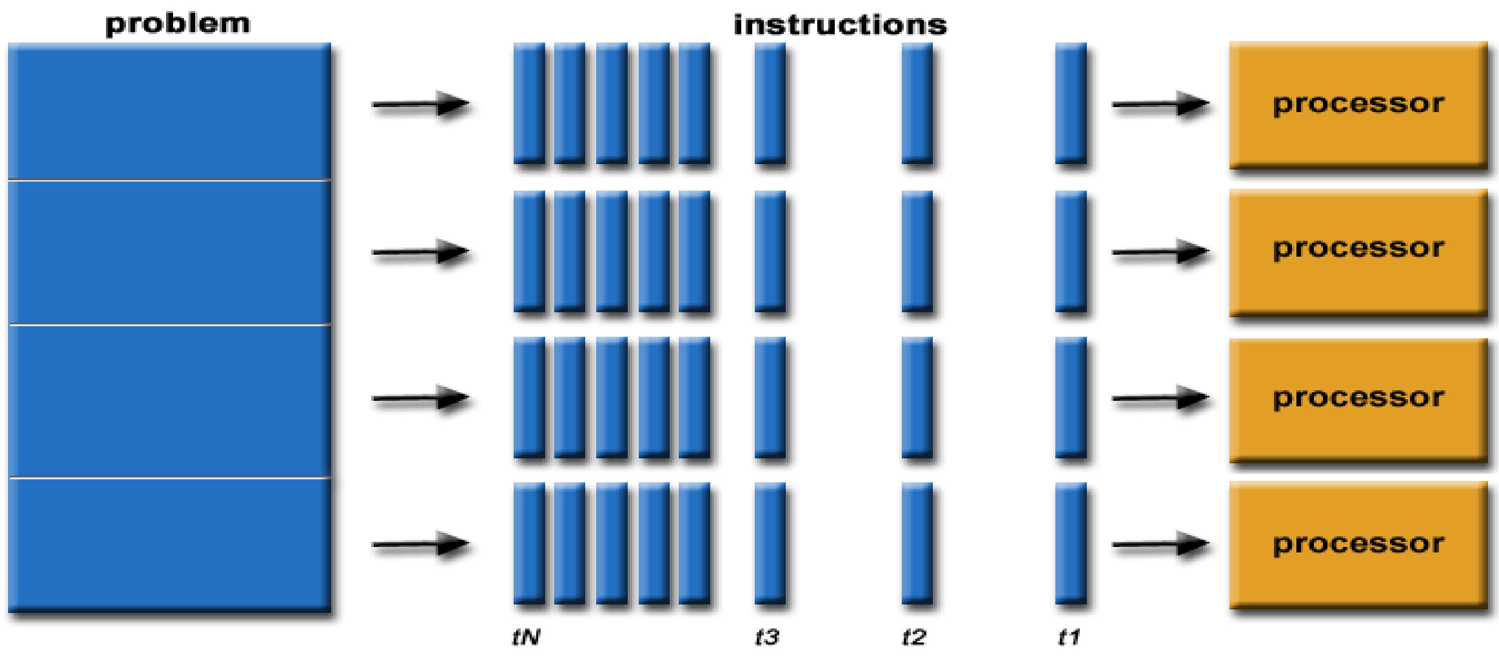

problem
instructions
processor
tN
t3
t2
t1

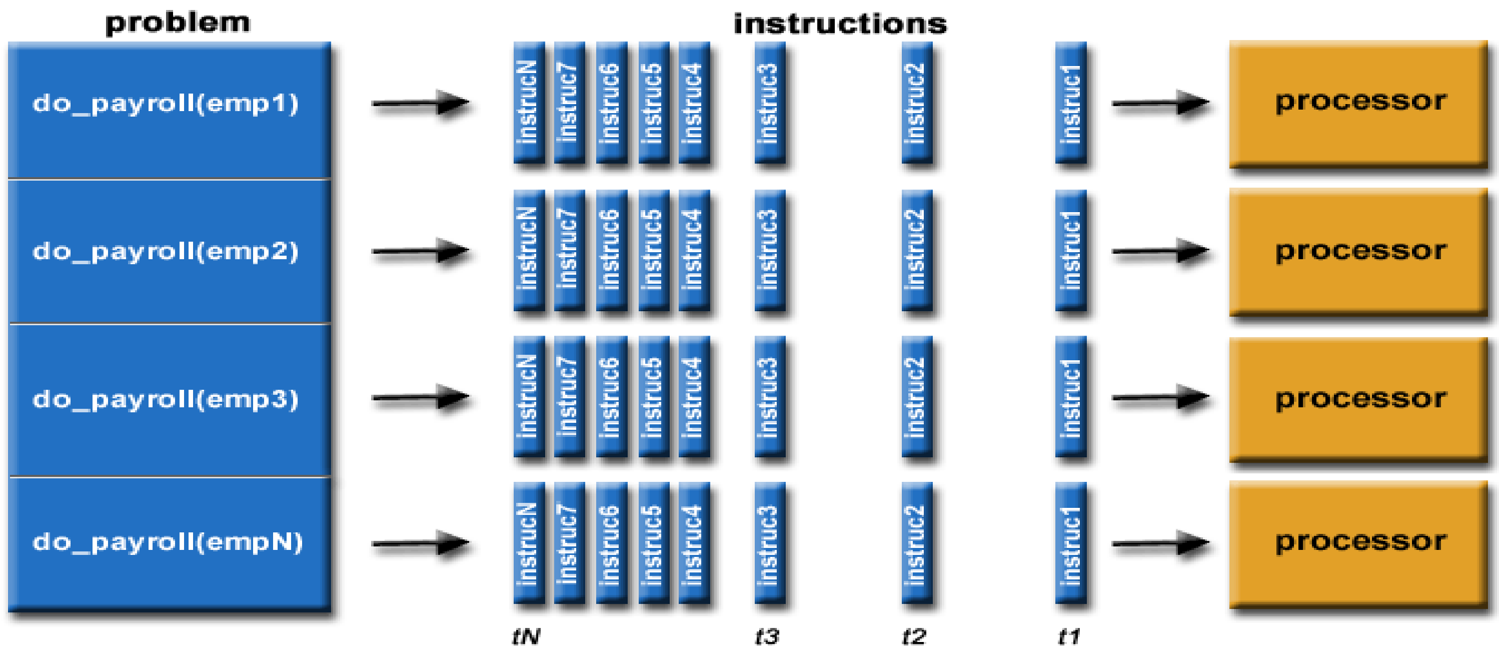

do_payroll()
instructions
. . . .
emp3_deduc
emp3_rate
emp3_hrs
emp2_check
emp2_tax
emp2_deduc
emp2_rate
emp2_hrs
emp1_check
emp1_tax
emp1_deduc
emp1_rate
emp1_hrs
processor
tN
t3
t2
t1

Parallel computing

It is the simultaneous use of multiple compute resources to solve a computational problem that can be broken into discrete parts and solved concurrently

- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control/coordination mechanism is employed


problem
instructions
processor
processor
processor
processor
tN
t3
t2
t1


problem
instructions
do_payroll(emp1)
do_payroll(emp2)
do_payroll(emp3)
do_payroll(empN)
instrucN
instruc7
instruc6
instruc5
instruc4
instruc3
instruc2
instruc1
processor
processor
processor
processor
tN
t3
t2
t1

Parallel computing

1 The computational problem should be able to:

- Be broken apart into discrete pieces of work that can be solved simultaneously
- Execute multiple program instructions at any moment in time;
- Be solved in less time with multiple compute resources than with a single compute resource.

Parallel compute resources

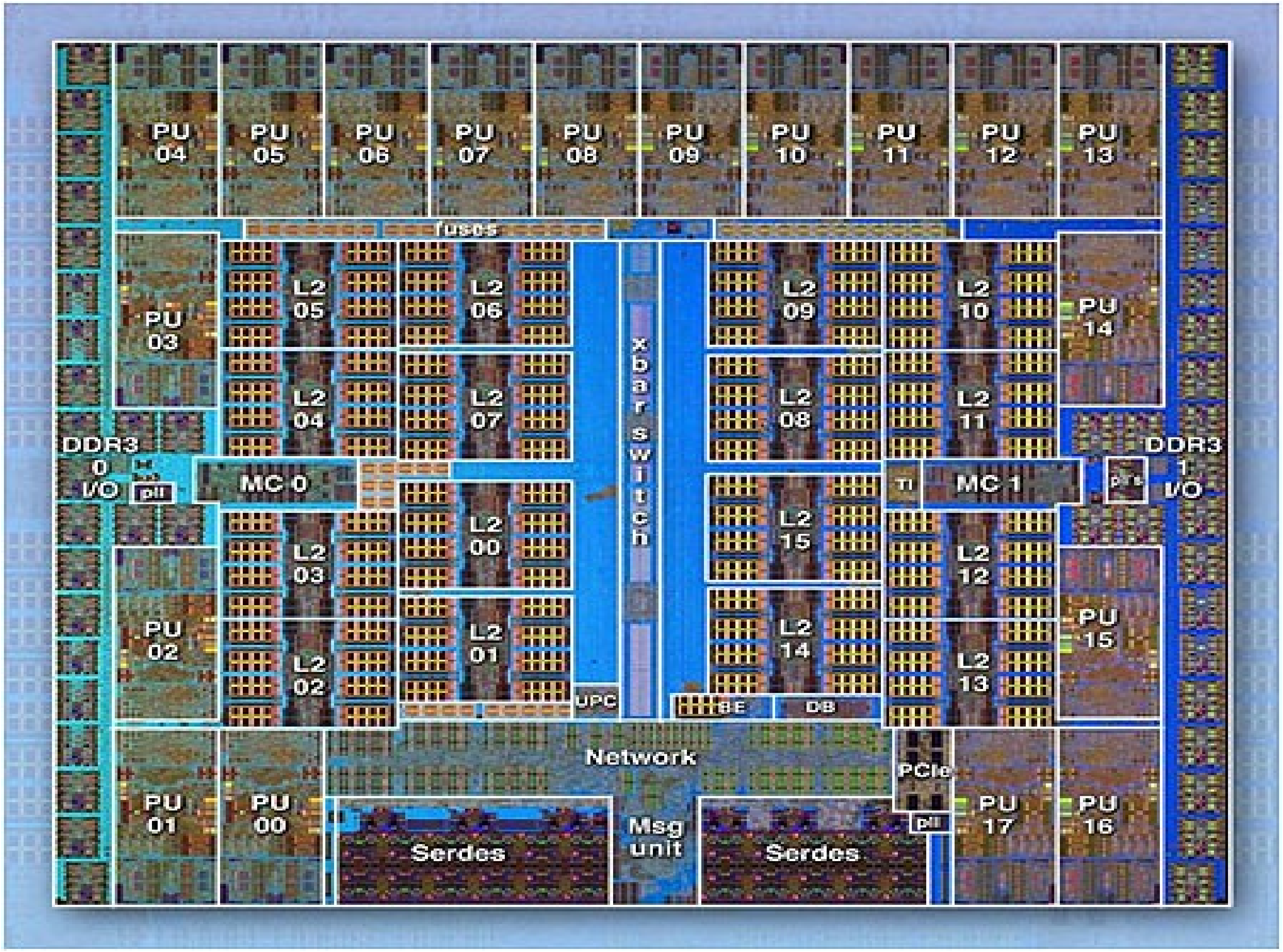
A single computer with multiple processors/cores

1 An arbitrary number of such computers connected by a network

2 These days almost all computers are parallel from a hardware perspective: Multiple functional units (L1 cache, L2 cache, branch, prefetch, decode, floating-point, graphics processing (GPU), etc.)

- Multiple execution units/cores
- Multiple hardware threads

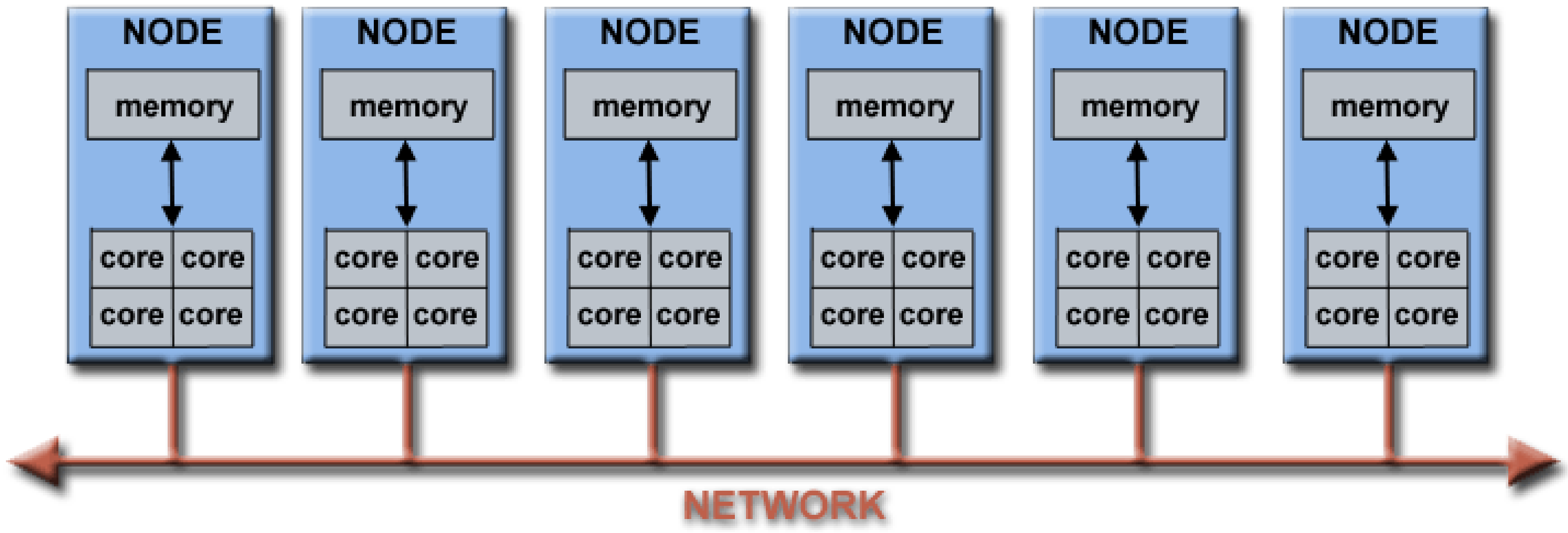
Parallel compute resources



- IBM BG/Q Compute Chip with 18 cores (PU) and 16 L2 Cache units (L2)

Parallel compute resources

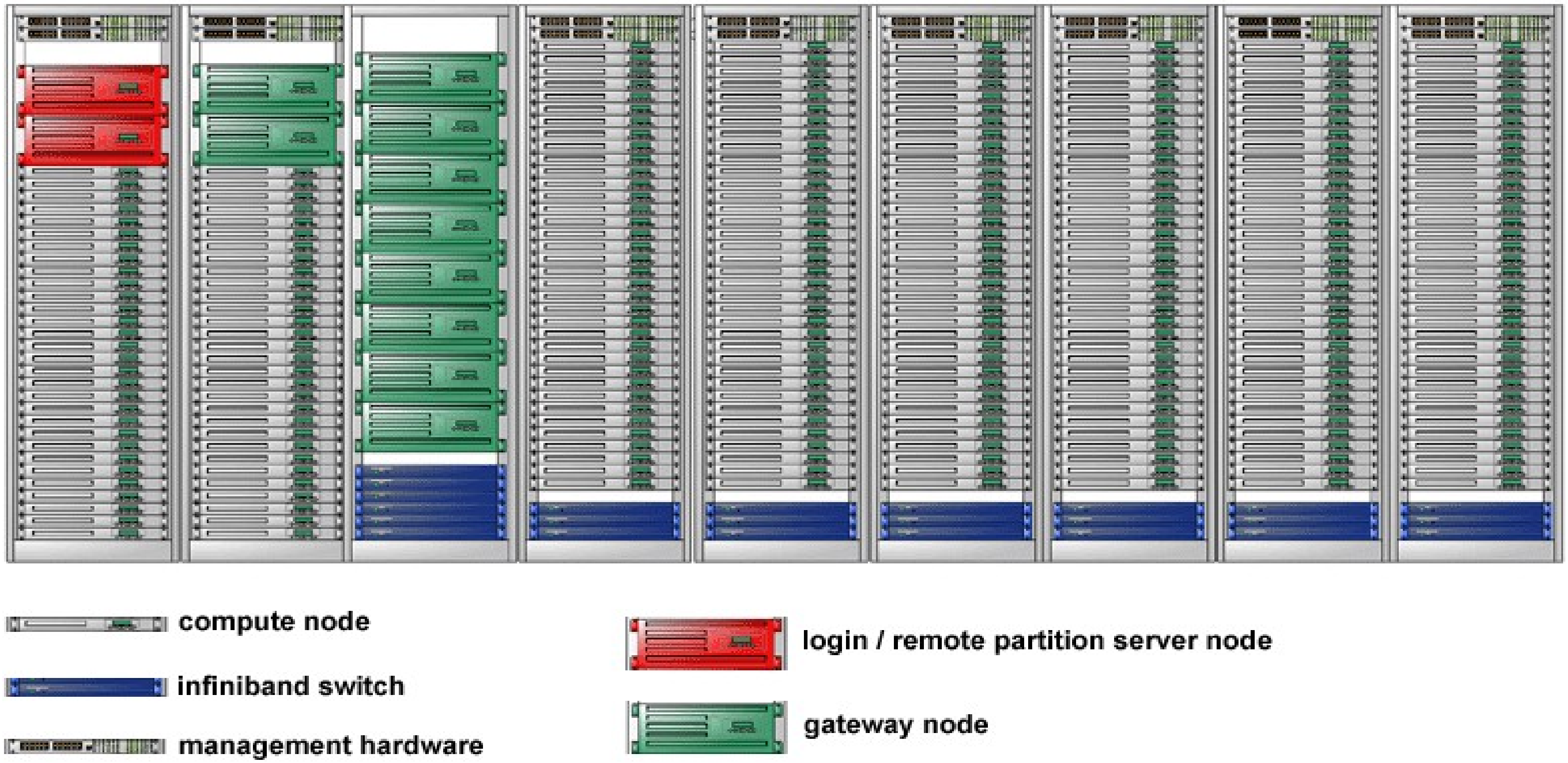
Networks connect multiple stand-alone computers (nodes) to make larger parallel computer clusters.



Parallel compute resources

For example, the schematic below shows a typical Lawrence Livermore National Laboratory (LLNL) parallel computer cluster:

- Each compute node is a multi-processor parallel computer in itself
- Multiple compute nodes are networked together with an Infiniband network



Why use Parallel computing?

The Real World is Massively Parallel

- In the natural world, many complex, interrelated events are happening at the same time, yet within a temporal sequence.

Auto Assembly

Jet Construction

Drive-thru Lunch

Why use Parallel computing?

The Real World is Massively Parallel

- Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena.

Rush Hour Traffic

Plate Tectonics

Weather

Galaxy Formation

Planetary Movments

Climate Change

Benefits of using Parallel computing

SAVE TIME AND/OR MONEY

- In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings.
- Parallel computers can be built from cheap, commodity components.

Benefits of using Parallel computing

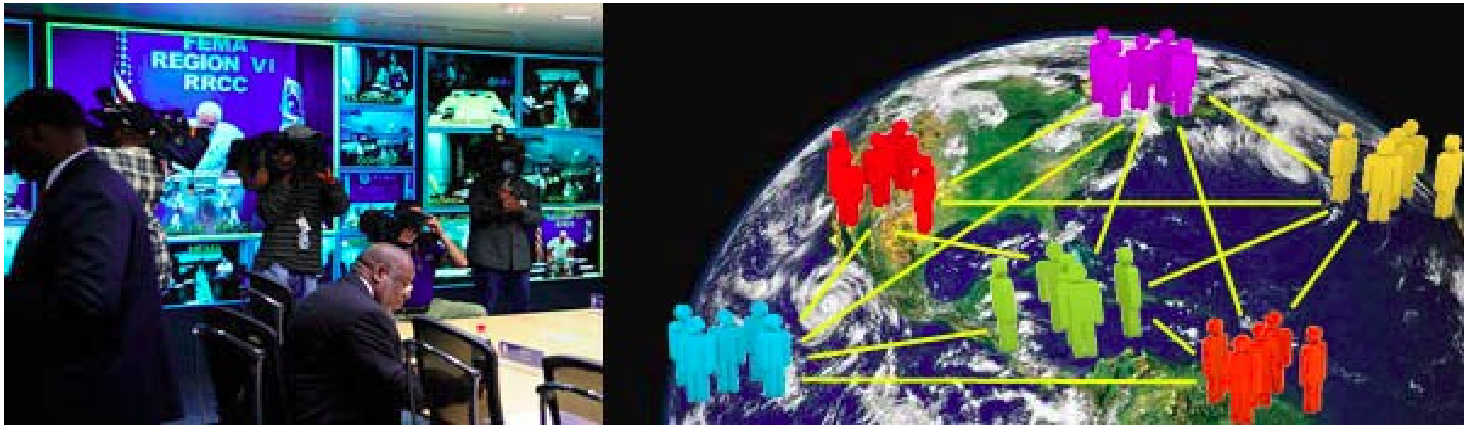
SOLVE LARGER / MORE COMPLEX PROBLEMS

- Many problems are so large and/or complex that it is impractical or impossible to solve them using a serial program, especially given limited computer memory.
- Example: "Grand Challenge Problems" (en.wikipedia.org/wiki/Grand_Challenge) requiring petaflops and petabytes of computing resources.
- Example: Web search engines/databases processing millions of transactions every second

Benefits of using Parallel computing

PROVIDE CONCURRENCY

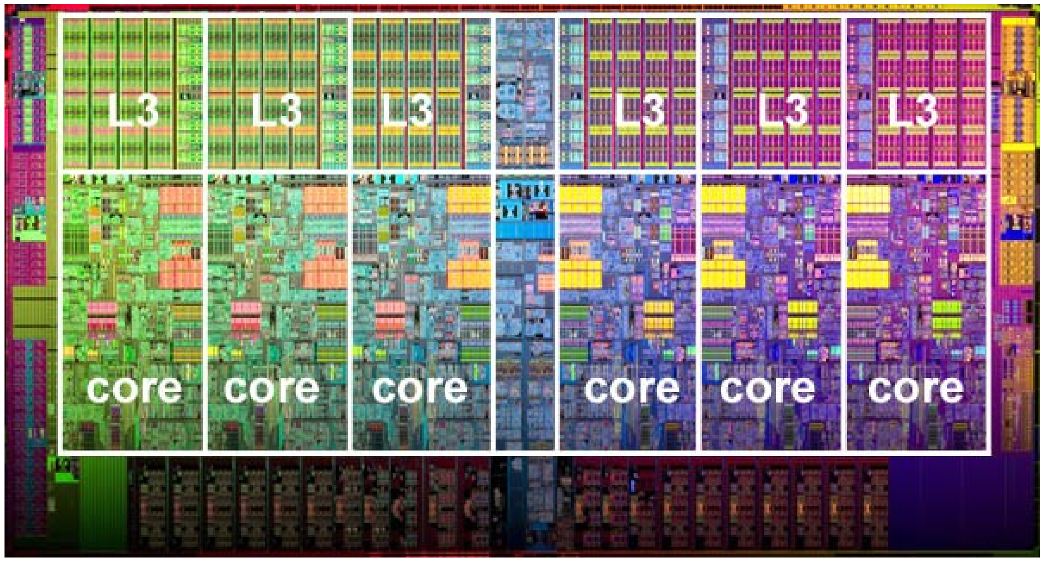
- A single compute resource can only do one thing at a time. Multiple compute resources can do many things simultaneously.
- Example: Collaborative Networks provide a global venue where people from around the world can meet and conduct work "virtually"



Benefits of using Parallel computing

MAKE BETTER USE OF UNDERLYING PARALLEL HARDWARE

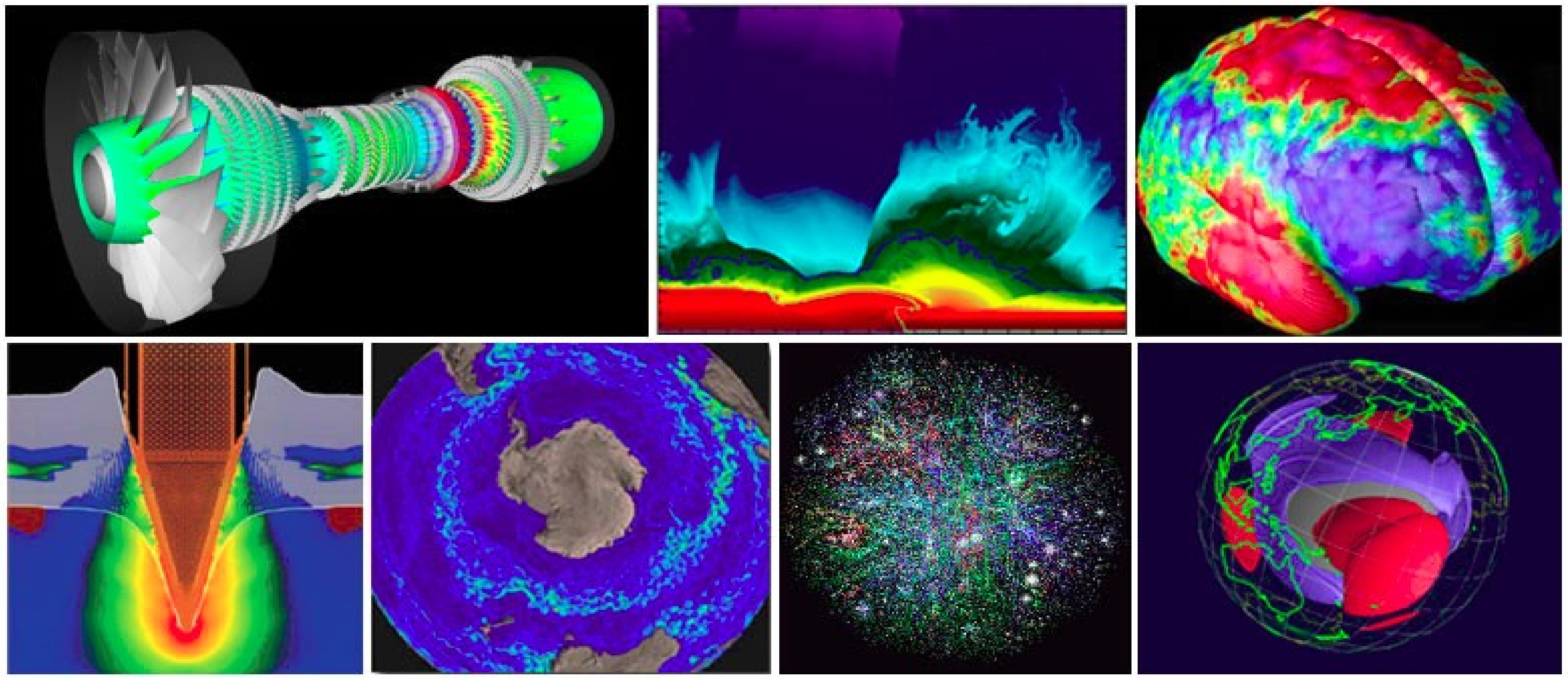
- Modern computers, even laptops, are parallel in architecture with multiple processors/cores.
- Parallel software is specifically intended for parallel hardware with multiple cores, threads, etc.
- In most cases, serial programs run on modern computers "waste" potential computing power.



Who is using Parallel computing?

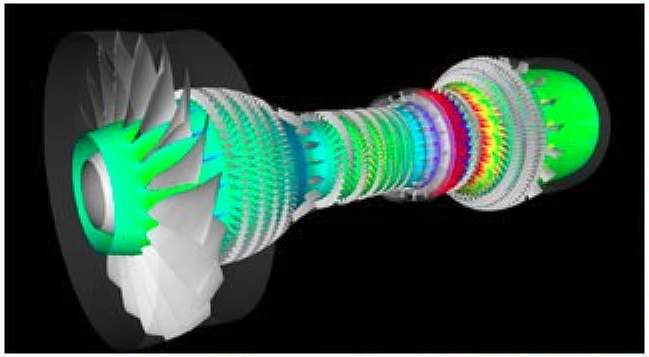
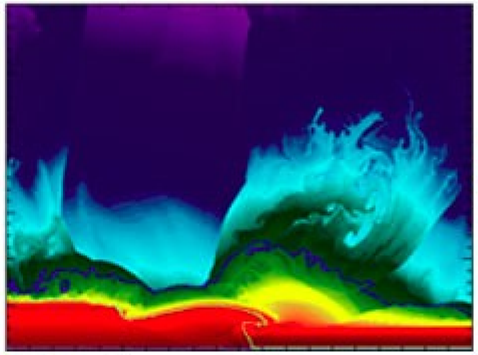
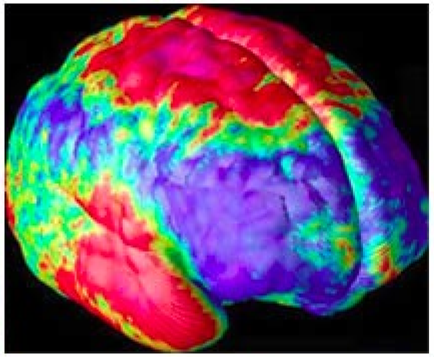
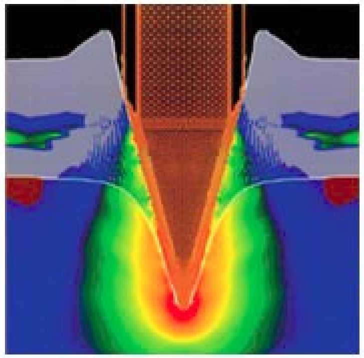
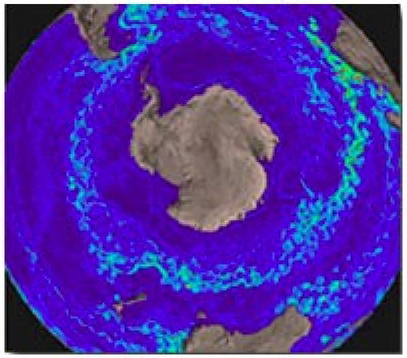
It has been used to model difficult problems in many areas of **science and engineering:**

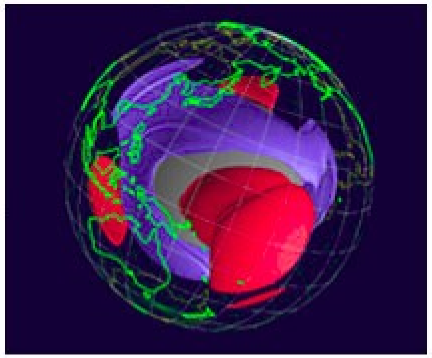
- Atmosphere, Earth, Environment
- Physics - applied, nuclear, particle, condensed matter, high pressure, fusion, photonics
- Bioscience, Biotechnology, Genetics



Who is using Parallel computing?

- Chemistry, Molecular Sciences
- Geology, Seismology
- Mechanical Engineering - from prosthetics to spacecraft
- Electrical Engineering, Circuit Design, Microelectronics
- Computer Science, Mathematics
- Defense, Weapons

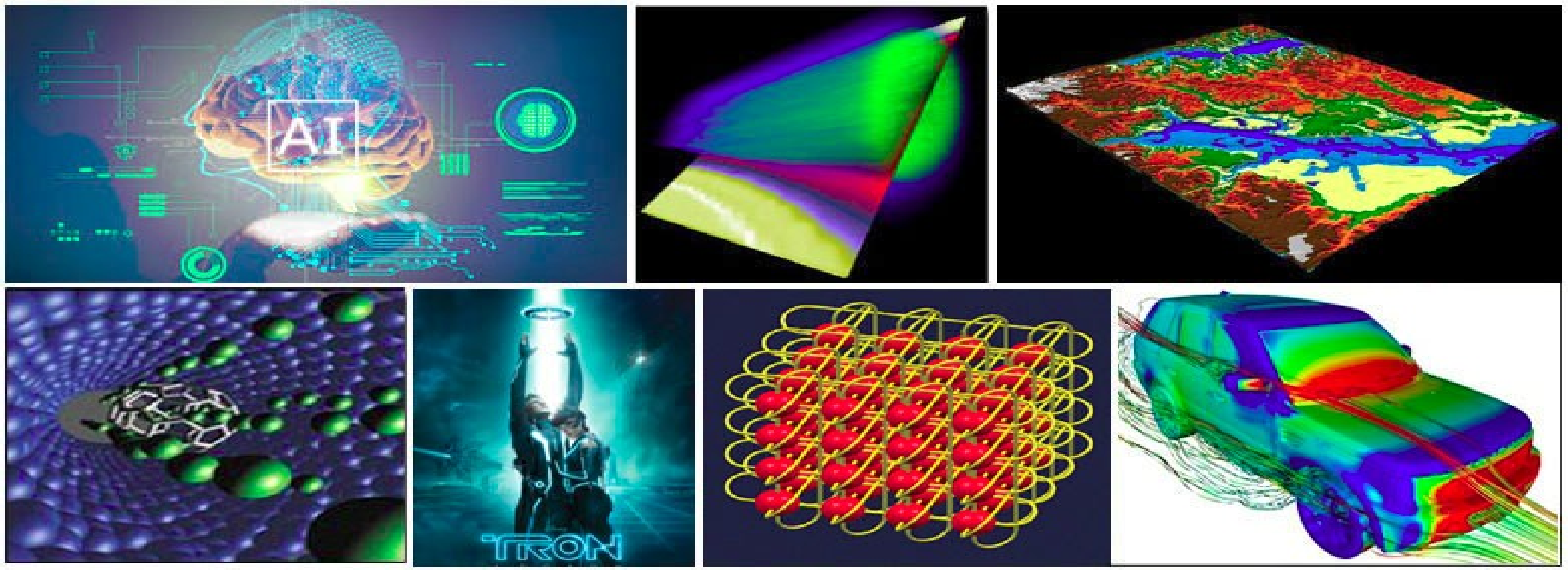








Who is using Parallel computing?

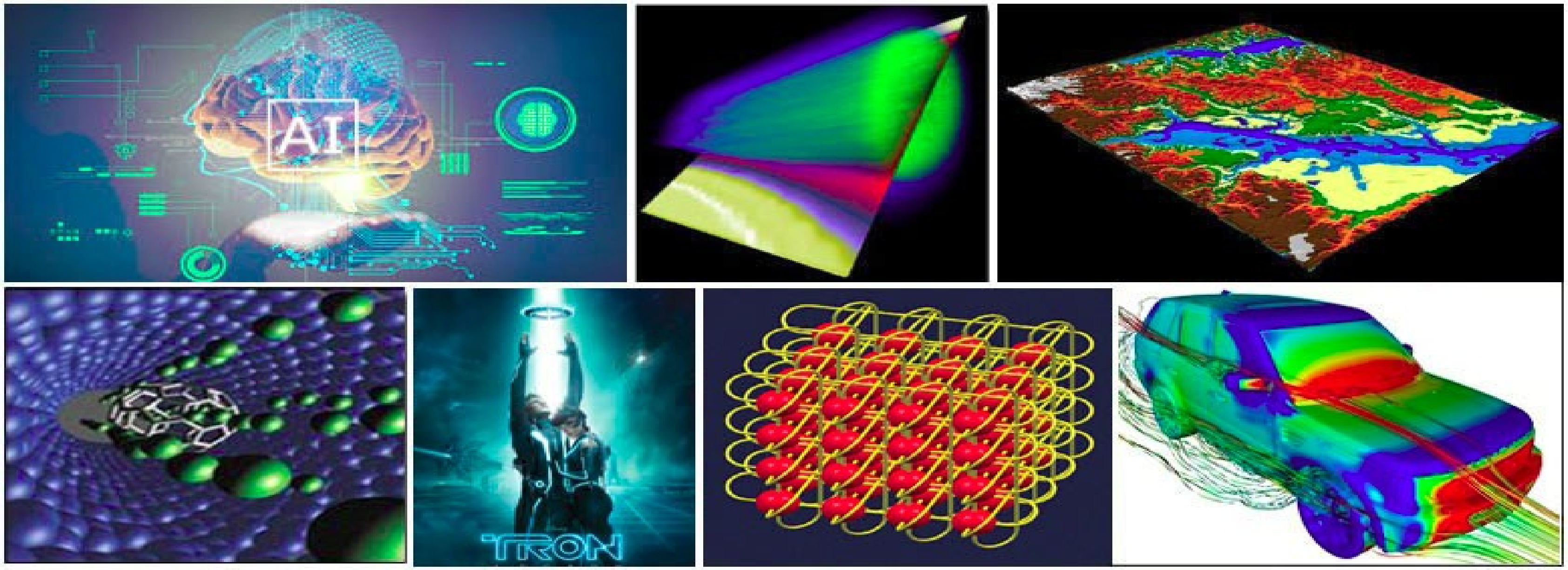
Industrial and commercial applications require the processing of large amounts of data in sophisticated ways. For example:

- "Big Data", databases, data mining
- Artificial Intelligence (AI)
- Oil exploration
- Web search engines, web based business services
- Medical imaging and diagnosis



Who is using Parallel computing?

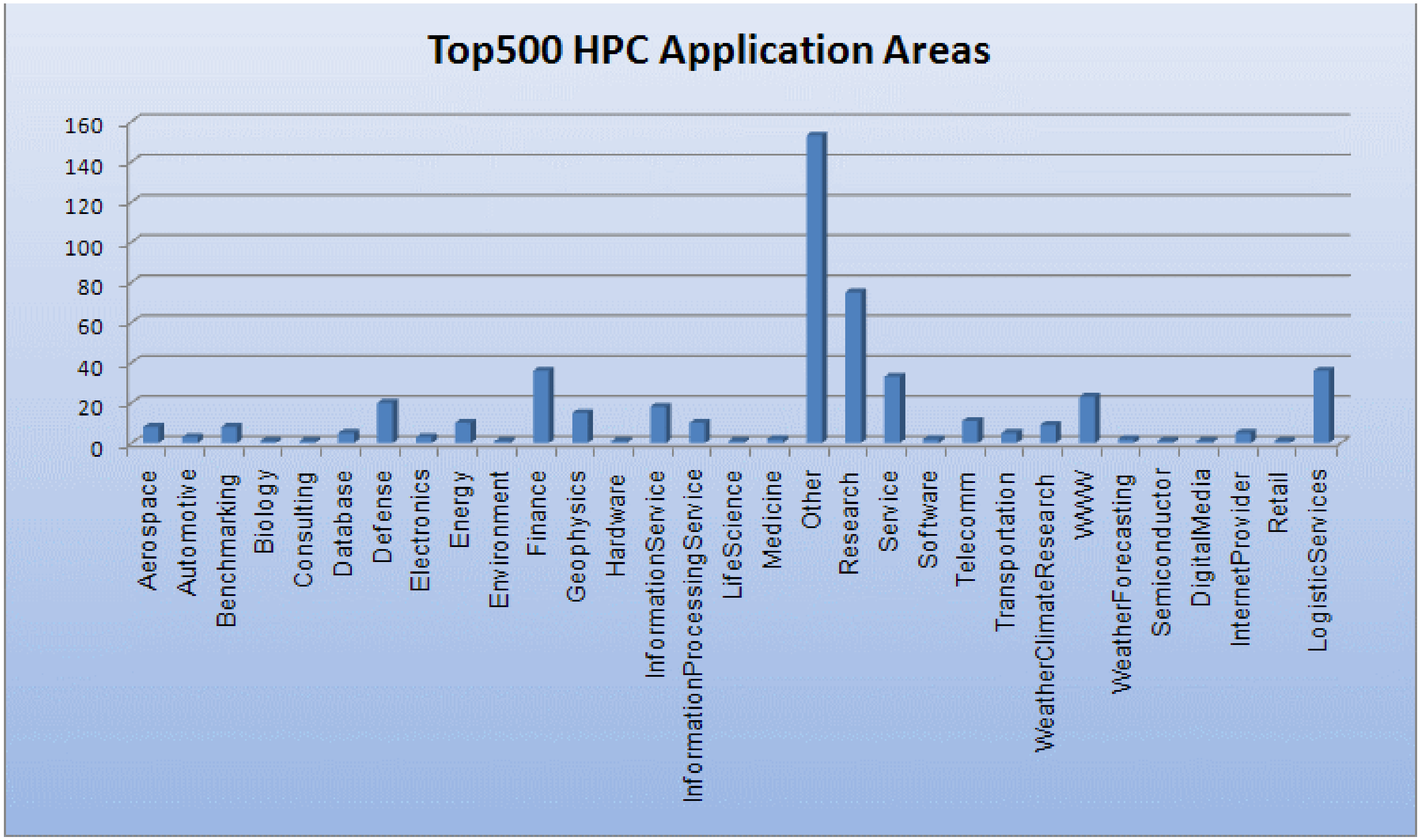
- Pharmaceutical design
- Financial and economic modeling
- Management of national and multi-national corporations
- Advanced graphics and virtual reality, particularly in the entertainment industry
- Networked video and multi-media technologies
- Collaborative work environments



Who is using Parallel computing?

Global Applications

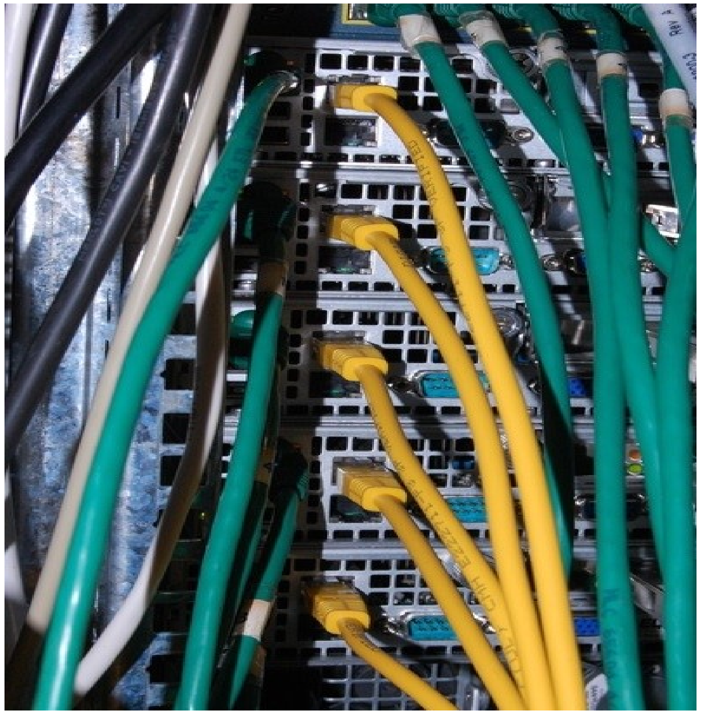
- Parallel computing is now being used extensively around the world, in a wide variety of applications.



Cluster Computing

Cluster Computing is a computing infrastructure that is comprised of a group of linked computers working together closely in order to improve performance

Cluster computing will have hybrid or distributed memory architecture and usually designed to mimic MIMD or MISD or MISD parallel program model.



Cluster Computing

The primary reasons for Cluster computing:

- It improves the performance of the system (wall clock time)
- It can solve larger problems
- Provide concurrency (do multiple things at the same time)
- Enable us to use distributed resources such as lab machines, which save cost, time, and overcome memory constraints

Cluster Computing

In cluster computing environment, a logically discrete section of computational work which is typically a program or program-like set of instructions that is given to be executed by a processor on a given data is called **Task** .

Factors to measure performance of parallel programs

Latency and Bandwidth measure the performance of a cluster environment

1. Bandwidth tells us the rate of data transmission per second from one process to another in a given network
2. Latency tells how much time will be spent in order to establish a communication link between processes over the network
3. Latency and bandwidth don’t depend on the actual application but on the networking facility

Factors to measure performance of parallel programs

It is also possible to measure the performance of a parallel program designed to solve specific application

Such performance depends on several factors such as

- Communication requirement of the Application
- Embarrassingly parallel problem (a problem that couldn’t be affected by the communication facility)
- Highly communication dependent problem
- Architecture of the cluster (specification of nodes, processors, heterogeneousness, etc)
- The algorithm implemented
- The Compiler used
- The amount of data need to be communicated
- Model of the parallel computing environment

Factors to measure performance of parallel programs

The performance improvement obtained using an application developed in a cluster environment can be measured using the parameter like

- Absolute Speedup (S_{an}) = time of equivalent sequential program/ time of parallel program in N processor
- Absolute Efficiency (E_{an})= S_{an}/N
- Relative Speedup (S_{rn}) = time required for the parallel code in 1 processor/ time of parallel program in N processor
- Relative Efficiency (E_{rn})= S_{rn}/N

Factors to measure performance of parallel programs

Question:

1. What do you say if Absolute Speedup (S_{an}) is less than one?
2. What do you say if Absolute Efficiency (E_{an})= 1
3. What do you say if Absolute Efficiency (E_{an}) > 1
4. What do you observe by looking at Relative Speedup (S_{rn}) for different value of N
5. What do you observe by looking at Relative Efficiency (E_{rn}) for different value of N

1.

Parallel Programming Model

Parallel programming models exist as an abstraction above hardware and memory architectures.

- The models basically differs in:
  - how the application divided into parallel tasks (the computational task)
  - how mapping is done between computational tasks and processing elements
  - how data distribution with in memory handled
  - how mapping of communication between inter-connected network is done
  - how inter-task synchronization is implemented

Parallel Programming Model

There are several parallel programming models in common use:

- Shared Memory (without threads)
- Threads
- Distributed Memory / Message Passing
- Data Parallel
- Hybrid
- Single Program Multiple Data (SPMD)
- Multiple Program Multiple Data (MPMD)

Parallel Programming Model

- Though various parallel programming models exist, in this course we will focus on two of the most common model :
  - OpenMP (Shared memory Architecture) and
  - MPI (Message passing programming models in a distributed or Hybrid memory Architecture).

MPI vs OpenMP: Parallel Programming Model

No	Factor	OpenMP	MPI
1	Task identification and mapping	implicit	explicit
2	Communication mapping	implicit	explicit
3	Synchronization	implicit	explicit
4	Data distribution	Implicit (directive based)	Explicit (programmer)
5	Environment	SMP	SMP, Distributed or hybrid memory architecture
6	Implementation	Multi-threading (forking)	Mainly a process on processor

Advantages of MPI

- Does not require shared memory architectures which are more expensive than distributed memory architectures
- Can be used on a wider range of problems since it exploits both task parallelism and data parallelism
- Can run on both shared memory and distributed memory architectures
- Highly portable with specific optimization for the implementation on most hardware

Disadvantage of MPI

- Requires more programming changes to go from serial to parallel version
- Can be harder to debug

Advantages of OpenMP

- Considered by some to be easier to program and debug (compared to MPI)
- Data layout and decomposition is handled automatically by directives.
- Allows incremental parallelism: directives can be added incrementally, so the program can be parallelized one portion after another and thus no dramatic change to code is needed.
- Unified code for both serial and parallel applications: OpenMP constructs are treated as comments when sequential compilers are used.
- Original (serial) code statements need not, in general, be modified when parallelized with OpenMP. This reduces the chance of introducing bugs and helps maintenance as well.

Disadvantages of OpenMP

- Currently only runs efficiently in shared-memory multiprocessor platforms
- Require compiler that supports OpenMP.
- Scalability is limited by memory architecture.
- Reliable error handling is missing.
- Lacks fine-grained mechanisms to control thread-processor mapping.
- Synchronization between subsets of threads is not allowed.
- Mostly used for loop parallelization
- Can be difficult to debug, due to implicit communication between threads via shared variables