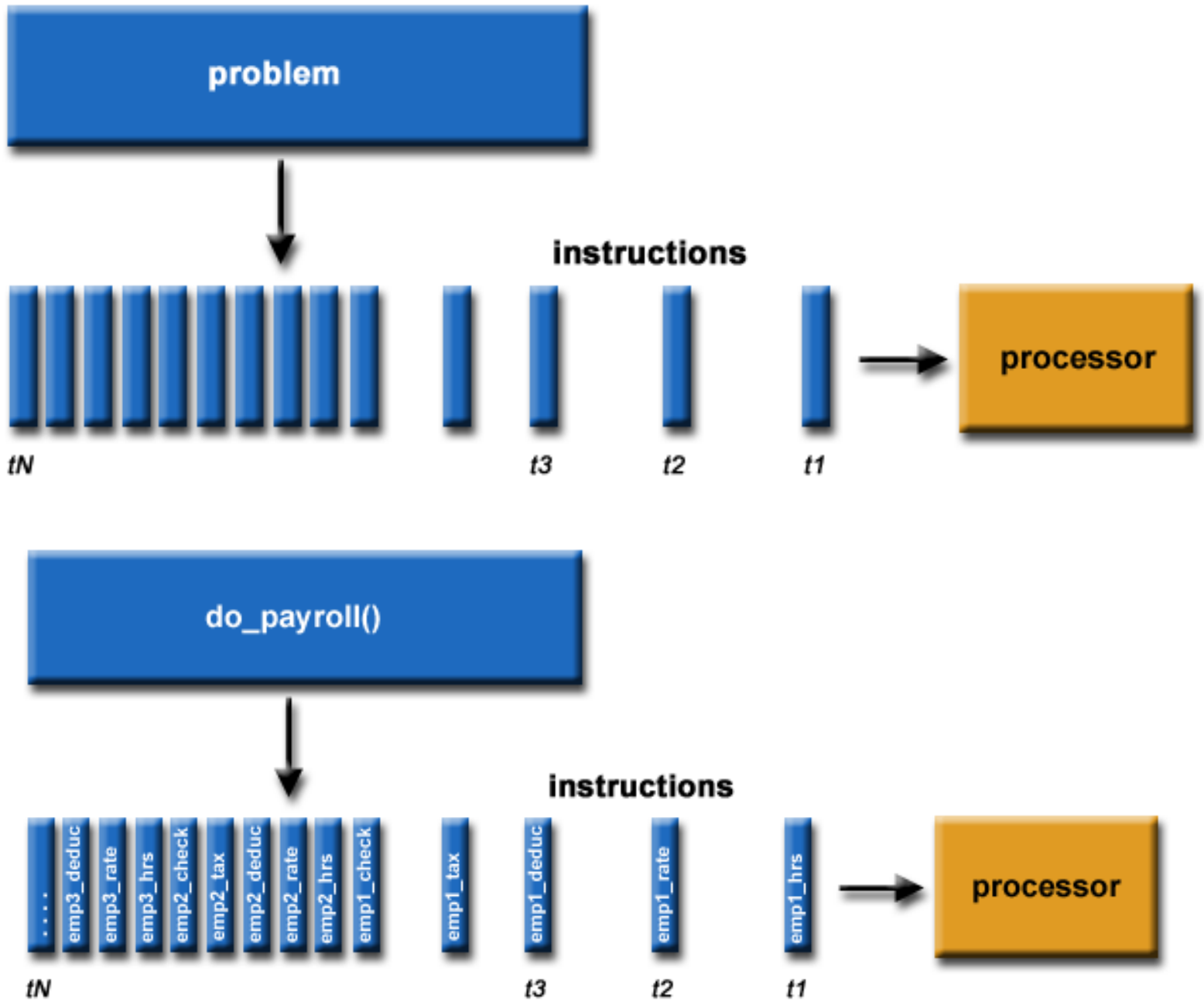
Basics of Parallel Computing

Contents

- What is Parallel Computing?
- Why Use Parallel Computing?
- Concepts and Terminology
- Parallel Computer Memory Architectures

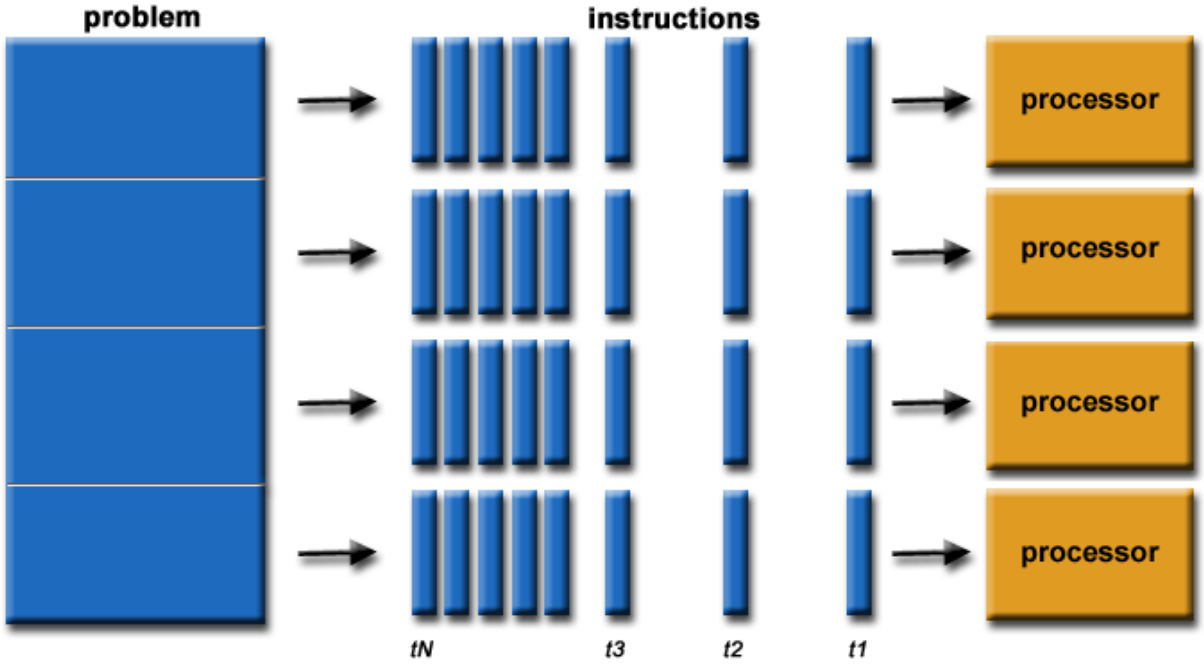
Serial Computing

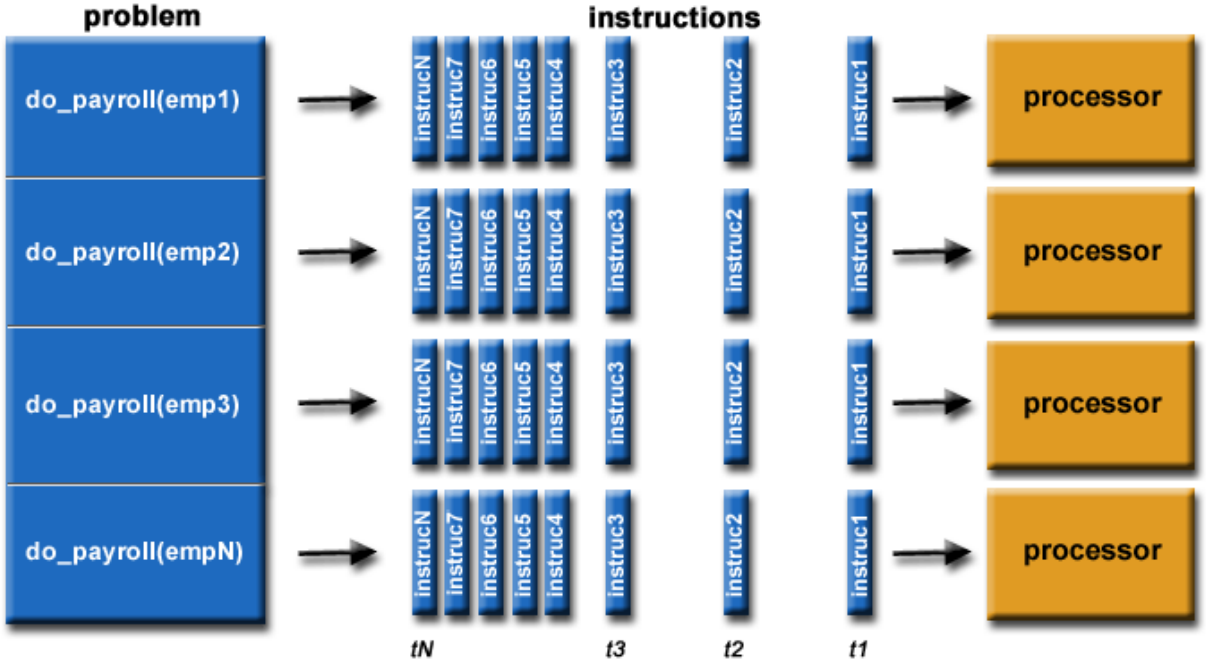
- Traditionally, software has been written for ***serial*** computation:
  - A problem is broken into a discrete series of instructions
  - Instructions are executed sequentially one after another
  - Executed on a single processor
  - Only one instruction may execute at any moment in time


problem
instructions
processor
tN
t3
t2
t1
do_payroll()
instructions
processor
. . . .
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
tN
t3
t2
t1

Parallel Computing

- ***parallel computing*** is the simultaneous use of multiple compute resources to solve a computational problem:
  - A problem is broken into discrete parts that can be 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

- 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.
- The compute resources are typically:
  - A single computer with multiple processors/cores
  - An arbitrary number of such computers connected by a network

Parallel Computers

- Virtually all stand-alone computers today are parallel from a hardware perspective:
  - Multiple functional units (L1 cache, L2 cache, branch, prefetch, decode, floating-point, graphics processing (GPU), integer, etc.)
  - Multiple execution units/cores
  - Multiple hardware threads

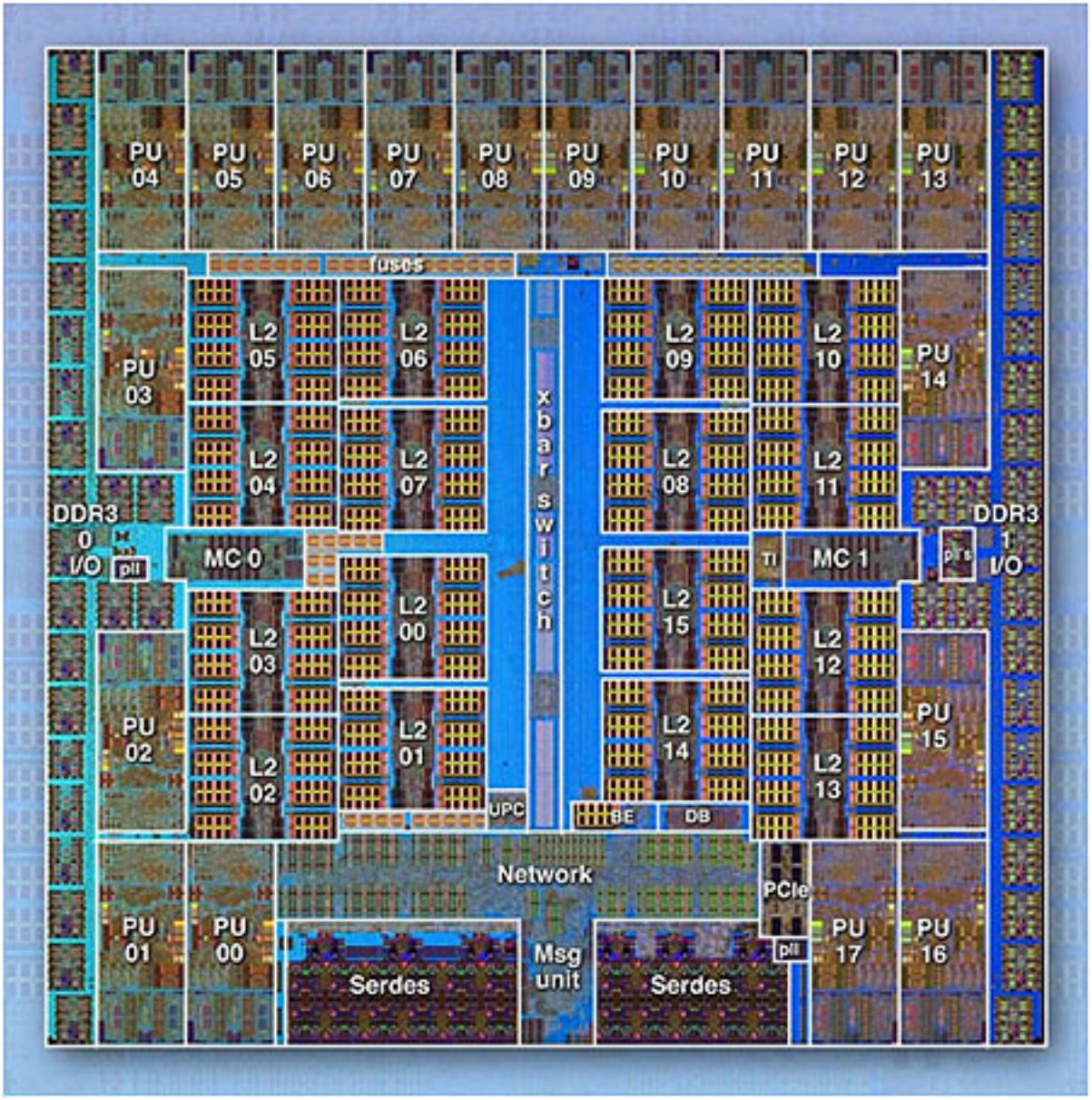
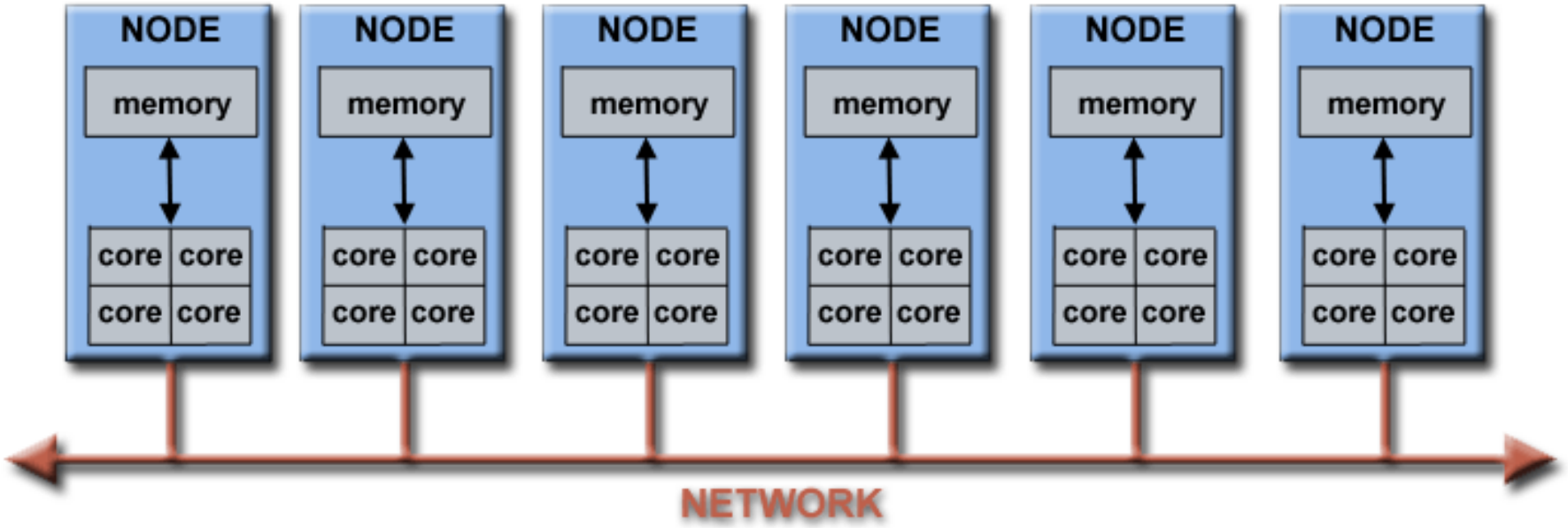
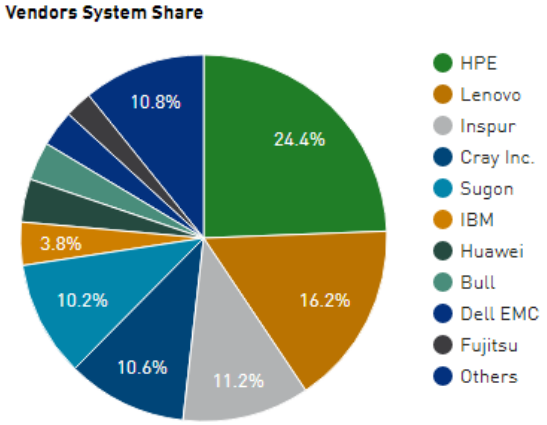


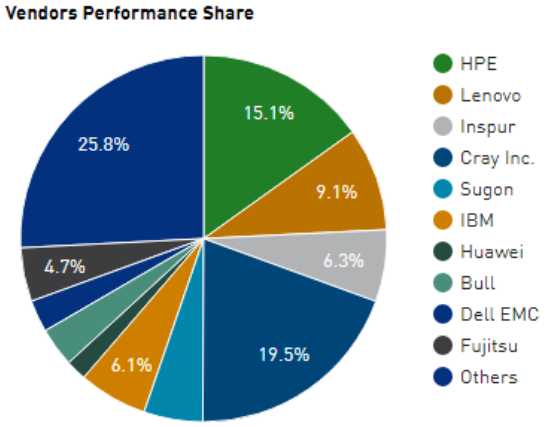
Fig: IBM BG/Q Compute Chip with 18 cores (PU) and 16 L2 Cache units (L2)

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



- The majority of the world's large parallel computers (supercomputers) are clusters of hardware produced by a handful of (mostly) well known vendors





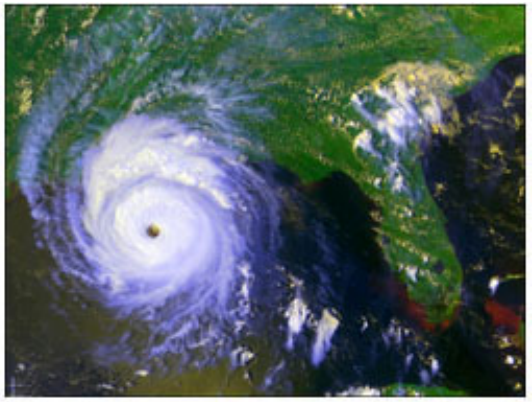
Vendors	Count	System Share (%)
HPE	122	24.4
Lenovo	81	16.2
Inspur	56	11.2
Cray Inc.	53	10.6
Sugon	51	10.2
IBM	19	3.8
Huawei	19	3.8
Bull	17	3.4
Dell EMC	16	3.2
Fujitsu	12	2.4
Penguin Computing	10	2
NUDT	4	0.8
PEZY Computing / Exascaler Inc.	4	0.8
NEC	4	0.8
Atipa	3	0.6
Lenovo/IBM	3	0.6
Nvidia	2	0.4
Dell EMC / IBM-GBS	2	0.4
T-Platforms	2	0.4
IBM/Lenovo	2	0.4
Self-made	1	0.2
T-Platforms, Intel, Dell	1	0.2
ClusterVision	1	0.2
Supermicro	1	0.2
ExaScaler	1	0.2
E4 Computer Engineering S.p.A.	1	0.2
RSC Group	1	0.2

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

Galaxy Formation

Planetary Movments



Climate Change

Rush Hour Traffic

Plate Tectonics

Weather

Auto Assembly

Jet Construction

Drive-thru Lunch

Main Reasons

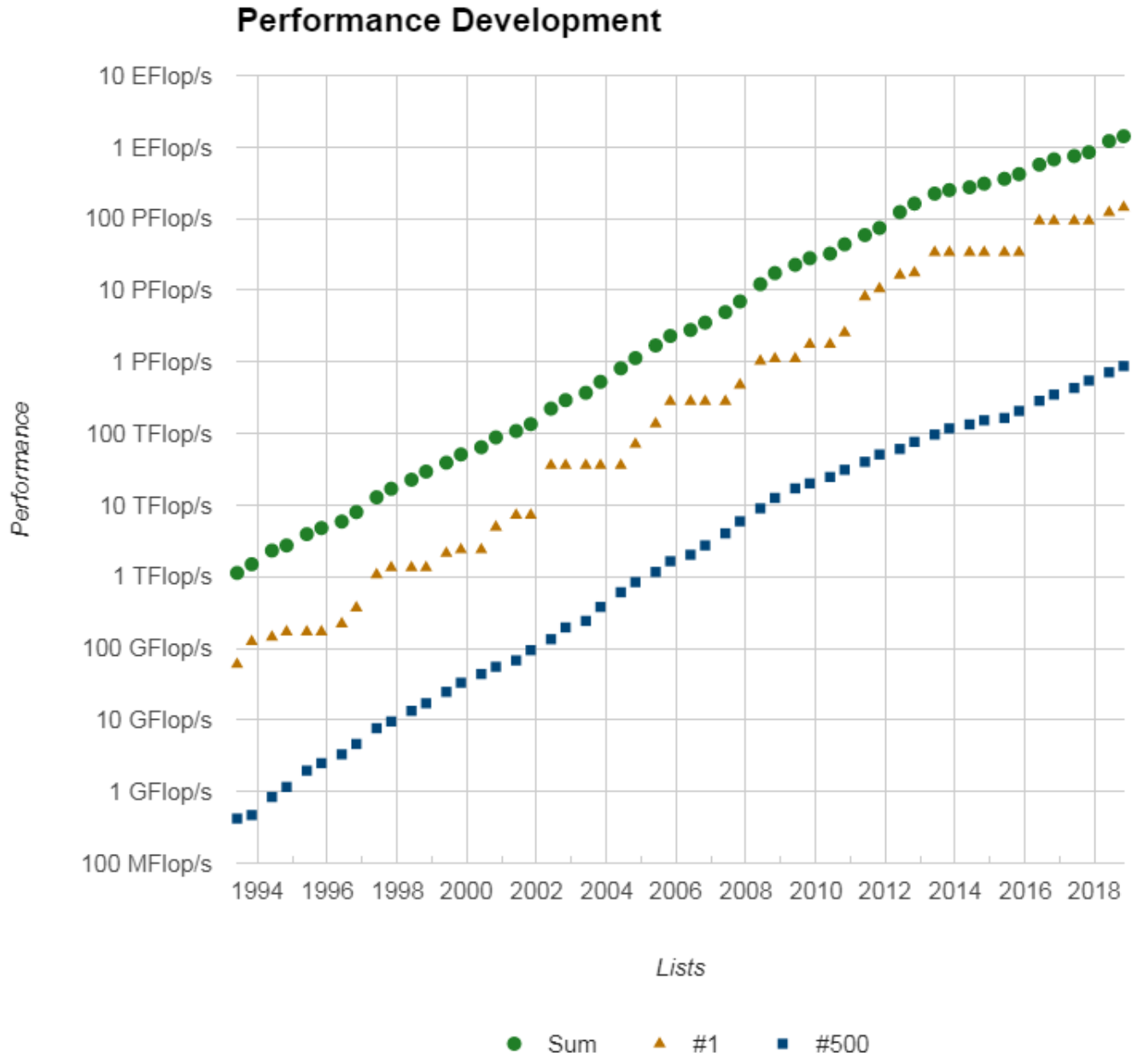
- **SAVE TIME AND/OR MONEY:**
  - In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings.
  - Parallel computers can be built from cheap, commodity components.
- **SOLVE LARGER / MORE COMPLEX PROBLEMS:**
  - Many problems are so large and/or complex that it is impractical or impossible to solve them on a single computer, especially given limited computer memory.
  - Example: Web search engines/databases processing millions of transactions every second

- **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".
- **TAKE ADVANTAGE OF NON-LOCAL RESOURCES:**
  - Using compute resources on a wide area network, or even the Internet when local compute resources are scarce or insufficient.
  - Example: SETI@home (setiathome.berkeley.edu)
  - Example: Folding@home (folding.stanford.edu)

- **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.

The future

- During the past 20+ years, the trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures (even at the desktop level) clearly show that ***parallelism is the future of computing***.
- In this same time period, there has been a greater than **500,000x** increase in supercomputer performance, with no end currently in sight.
- ***The race is already on for Exascale Computing!***
  - Exaflop = 10^{18} calculations per second


Performance Development
10 EFlop/s
1 EFlop/s
100 PFlop/s
10 PFlop/s
1 PFlop/s
100 TFlop/s
10 TFlop/s
1 TFlop/s
100 GFlop/s
10 GFlop/s
1 GFlop/s
100 MFlop/s
Performance
1994 1996 1998 2000 2002 2004 2006 2008 2010 2012 2014 2016 2018
Lists
Sum
#1
#500

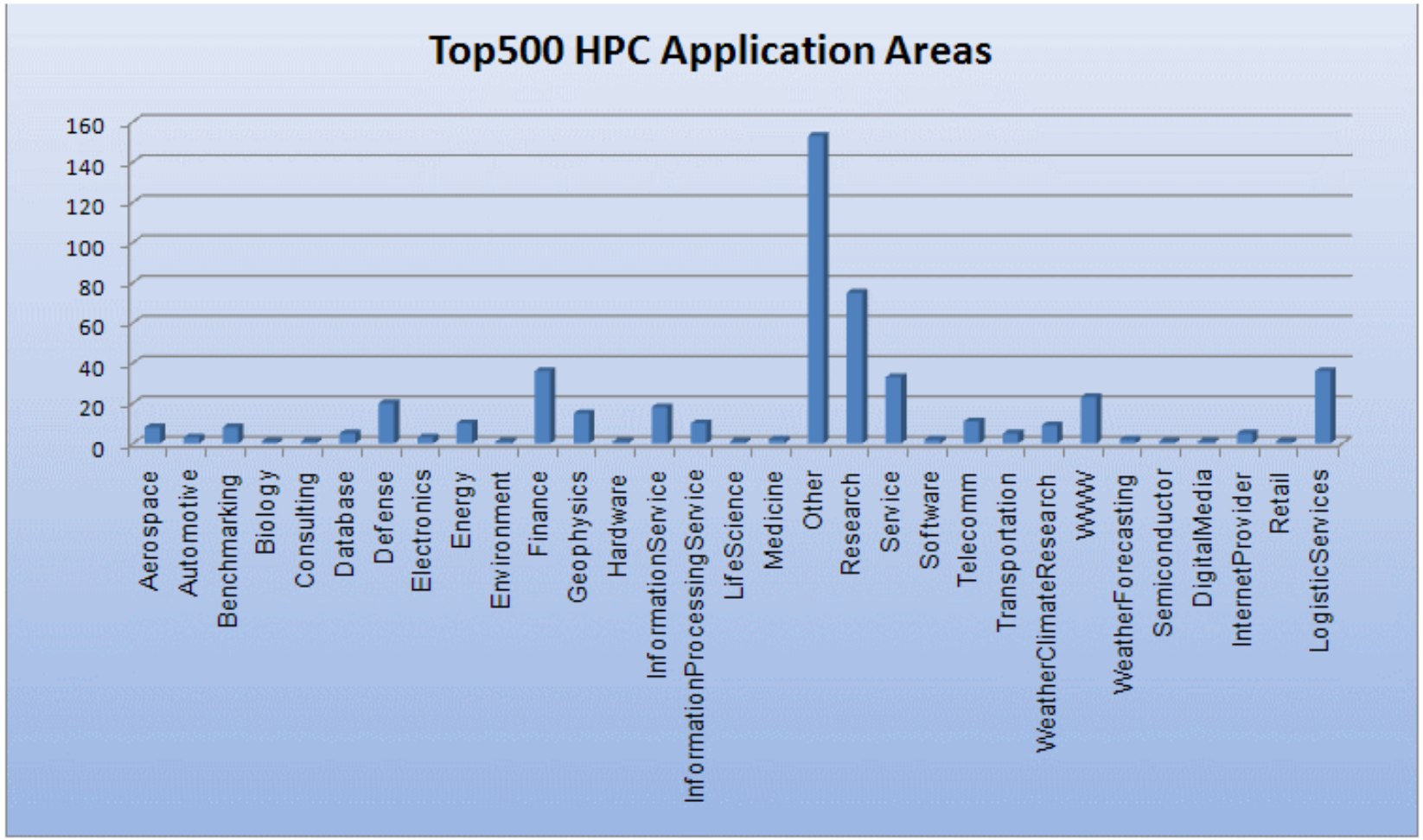
Who is Using Parallel Computing?

- **Science and Engineering:**
- parallel computing 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
  - Chemistry, Molecular Sciences
  - Geology, Seismology
  - Mechanical Engineering - from prosthetics to spacecraft
  - Electrical Engineering, Circuit Design, Microelectronics
  - Computer Science, Mathematics
  - Defence, Weapons

- **Industrial and Commercial:**
- These applications require the processing of large amounts of data in sophisticated ways.
  - "Big Data", databases, data mining
  - Artificial Intelligence (AI)
  - Web search engines, web based business services
  - Medical imaging and diagnosis
  - 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
  - Oil exploration

Global Applications

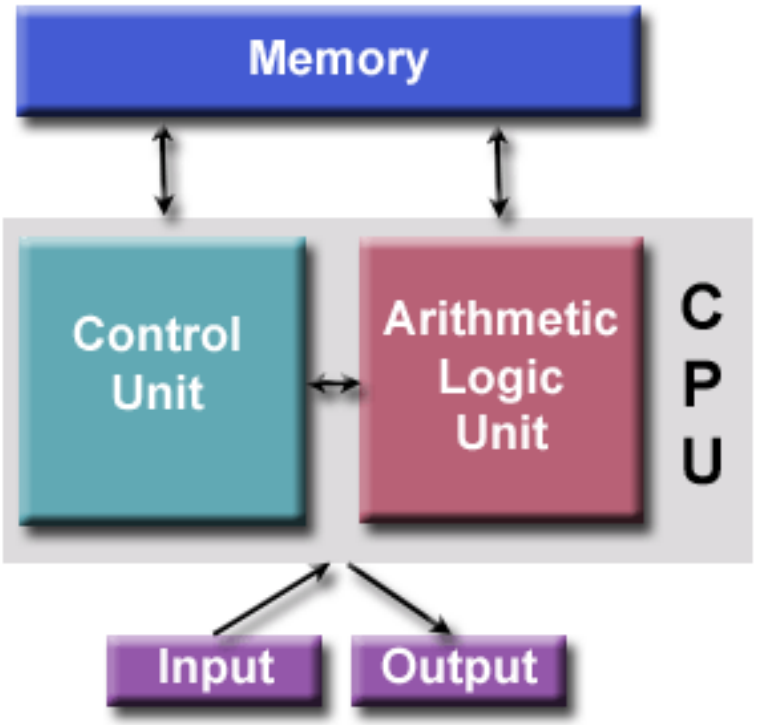
- Parallel computing is now being used extensively around the world, in a wide variety of applications.



Concepts and Terminology

- **von Neumann Architecture**
  - Hungarian mathematician/genius John von Neumann - requirements for an electronic computer (1945 papers)
  - Also known as "stored-program computer" - both program instructions and data are kept in electronic memory.
  - Differs from earlier computers which were programmed through "hard wiring".

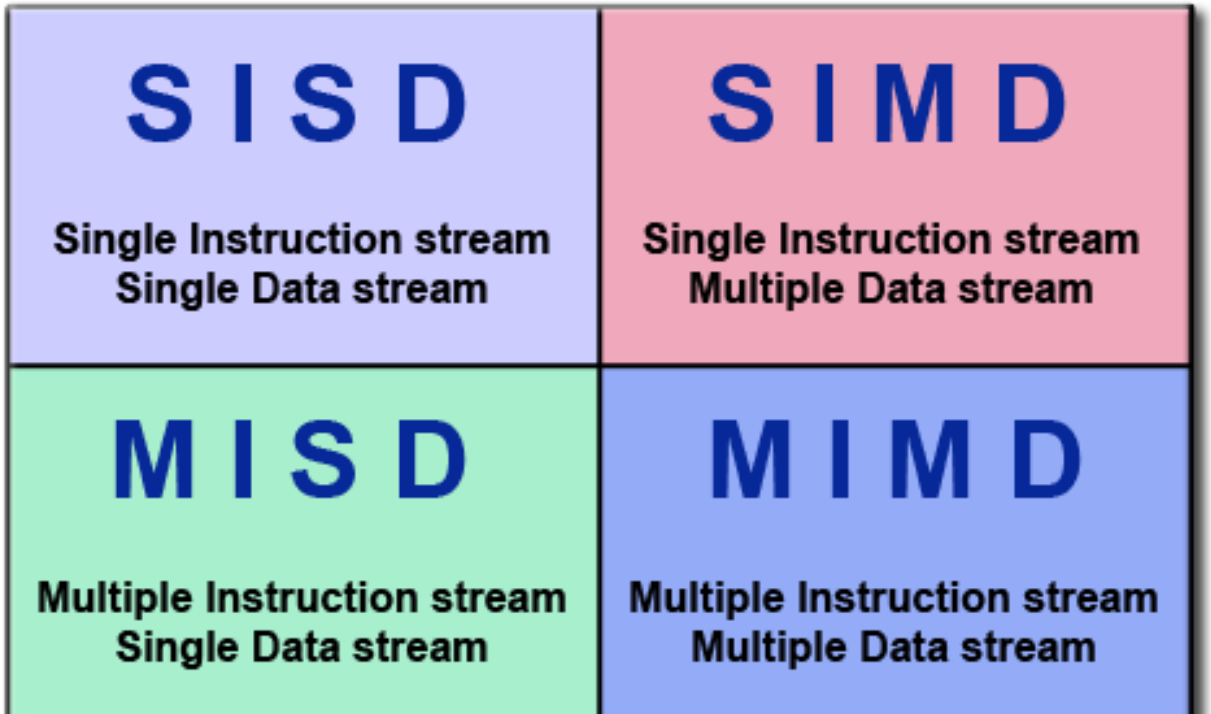
- **von Neumann Architecture -** Comprised of four main components:
  - Memory
  - Control Unit
  - Arithmetic Logic Unit
  - Input/Output



NOTE: parallel computers still follow this basic design, just multiplied in units. The basic, fundamental architecture remains the same.

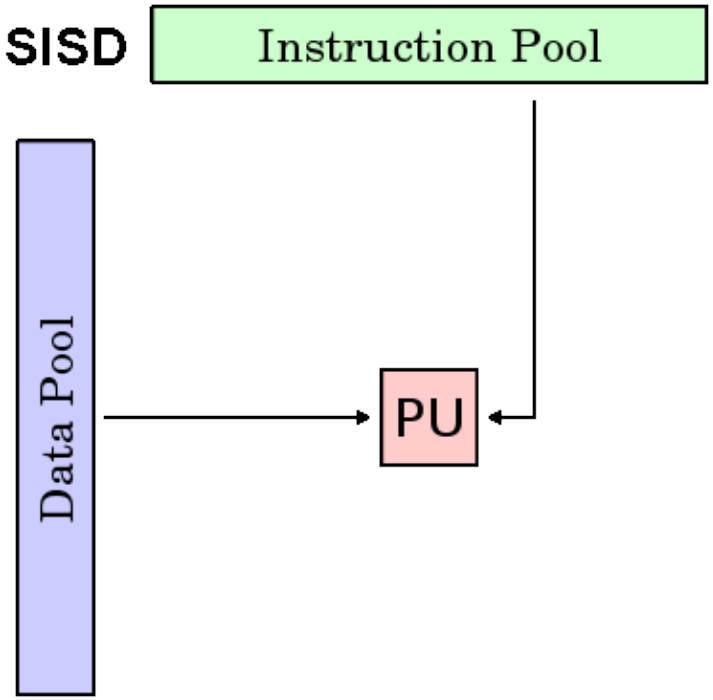
Flynn's Classical Taxonomy

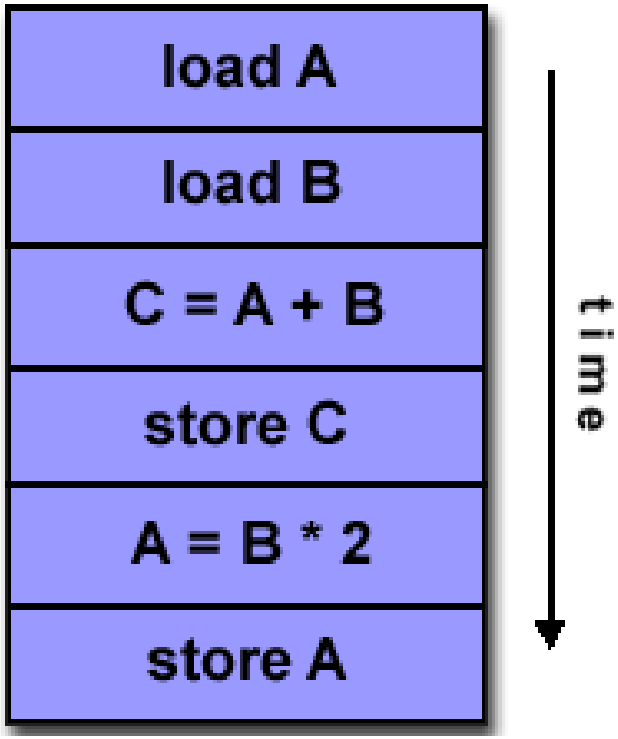
- Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of ***Instruction Stream*** and ***Data Stream***. Each of these dimensions can have only one of two possible states: ***Single*** or ***Multiple***.



Single Instruction, Single Data (SISD)

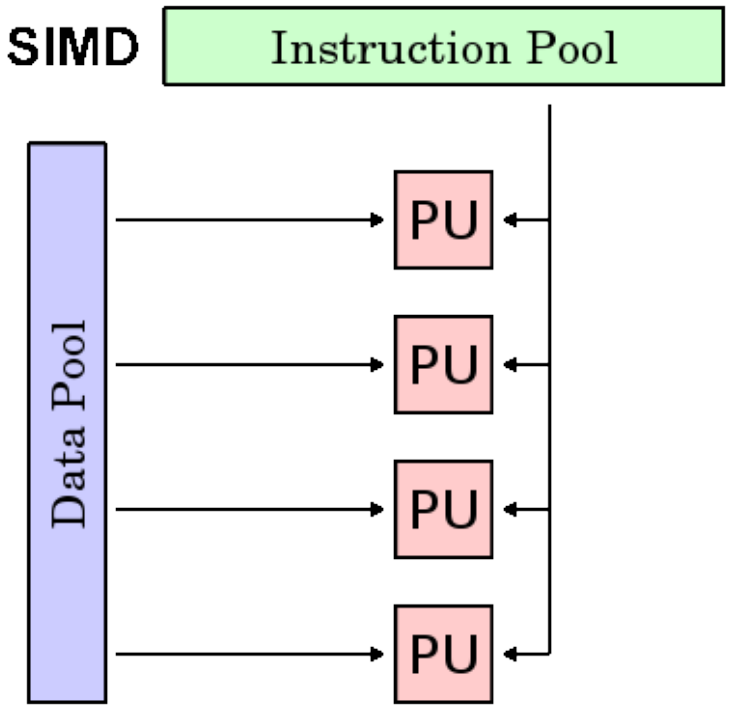
- A serial (non-parallel) computer
- **Single Instruction:** Only one instruction stream is being acted on by the CPU during any one clock cycle
- **Single Data:** Only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest type of computer
- Examples: older generation mainframes, minicomputers, workstations and single processor/core PCs. (**UNIVAC1, IBM 360, CRAY1, CDC7600, PDP1, Dell Laptop)**

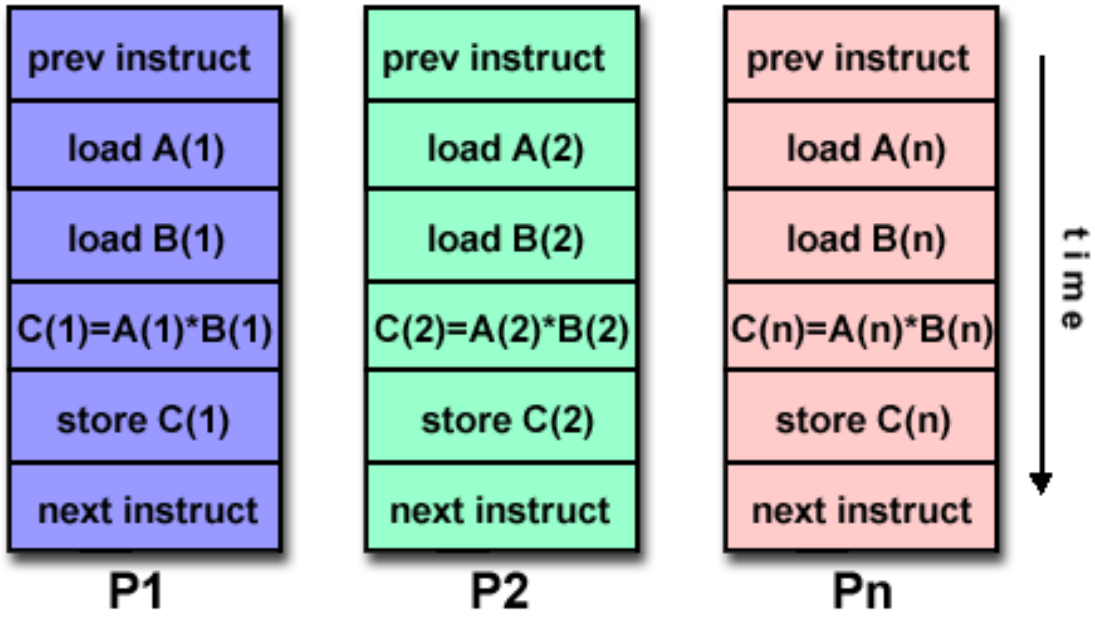

SISD
Instruction Pool
Data Pool
PU

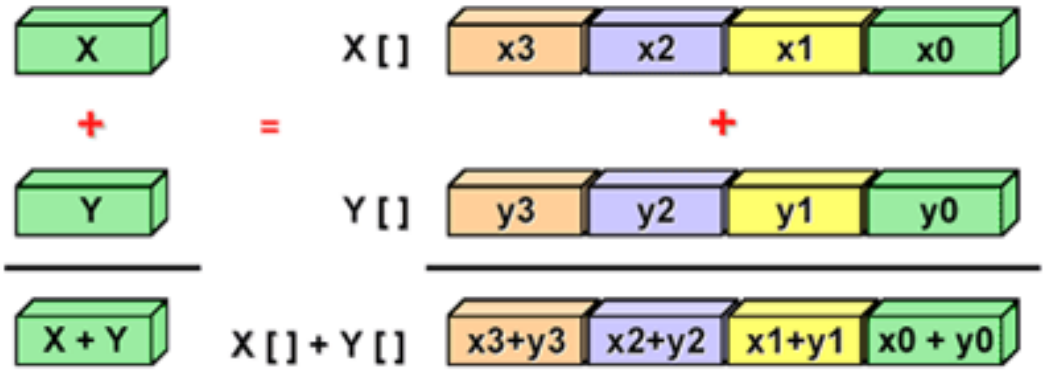

load A
load B
C = A + B
store C
A = B * 2
store A
time

Single Instruction, Multiple Data (SIMD)

- A type of parallel computer
- **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
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Examples:
  - Processor Arrays: Thinking Machines CM-2, MasPar MP-1 & MP-2, ILLIAC IV
  - Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10
- Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.

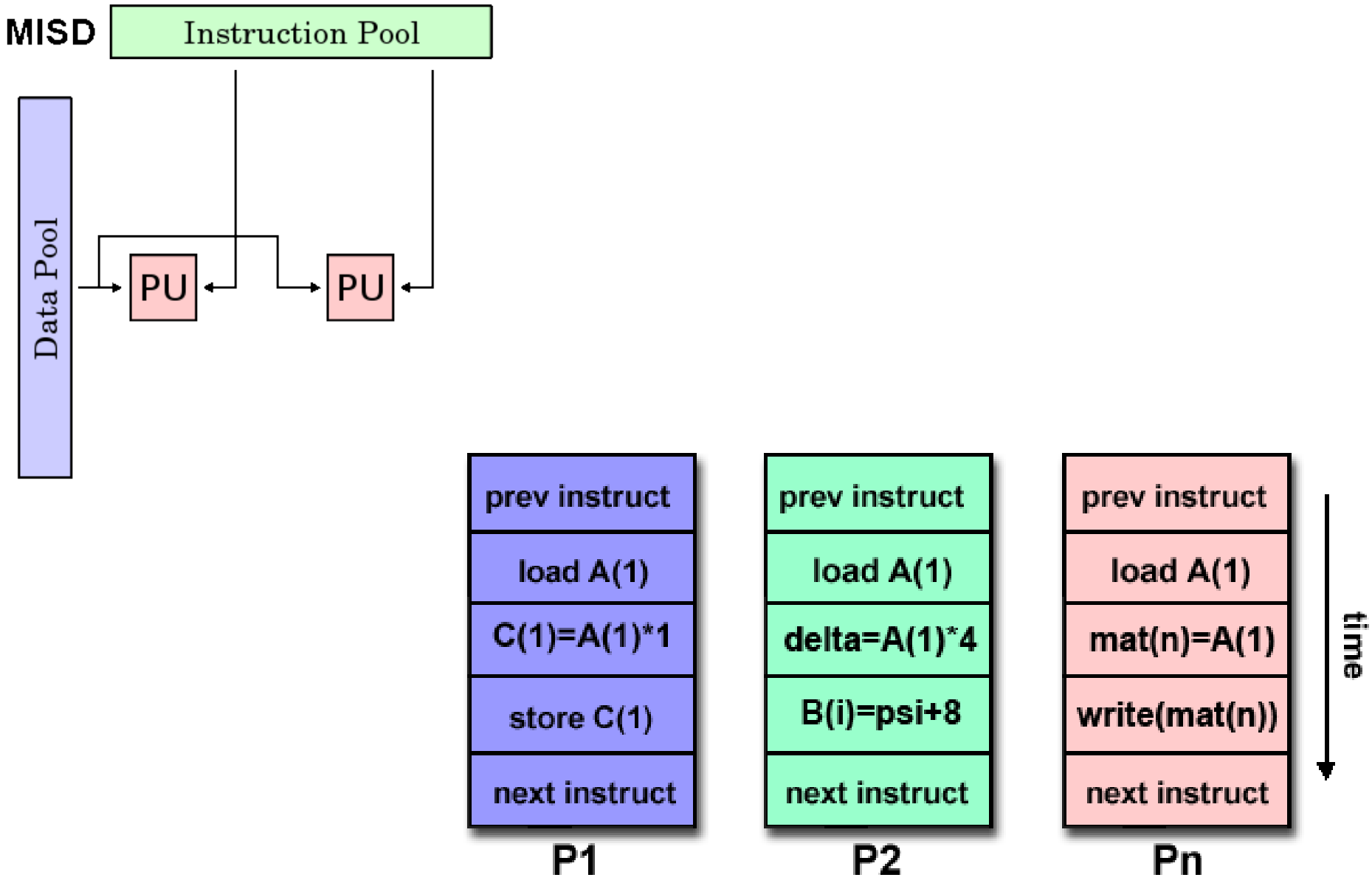

SIMD
Instruction Pool
Data Pool
PU
PU
PU
PU


prev instruct
load A(1)
load B(1)
C(1)=A(1)*B(1)
store C(1)
next instruct
P1
prev instruct
load A(2)
load B(2)
C(2)=A(2)*B(2)
store C(2)
next instruct
P2
prev instruct
load A(n)
load B(n)
C(n)=A(n)*B(n)
store C(n)
next instruct
Pn
time


X
+
Y
X + Y
=
X [ ]
x3
x2
x1
x0
+
Y [ ]
y3
y2
y1
y0
X [ ] + Y [ ]
x3+y3
x2+y2
x1+y1
x0 + y0

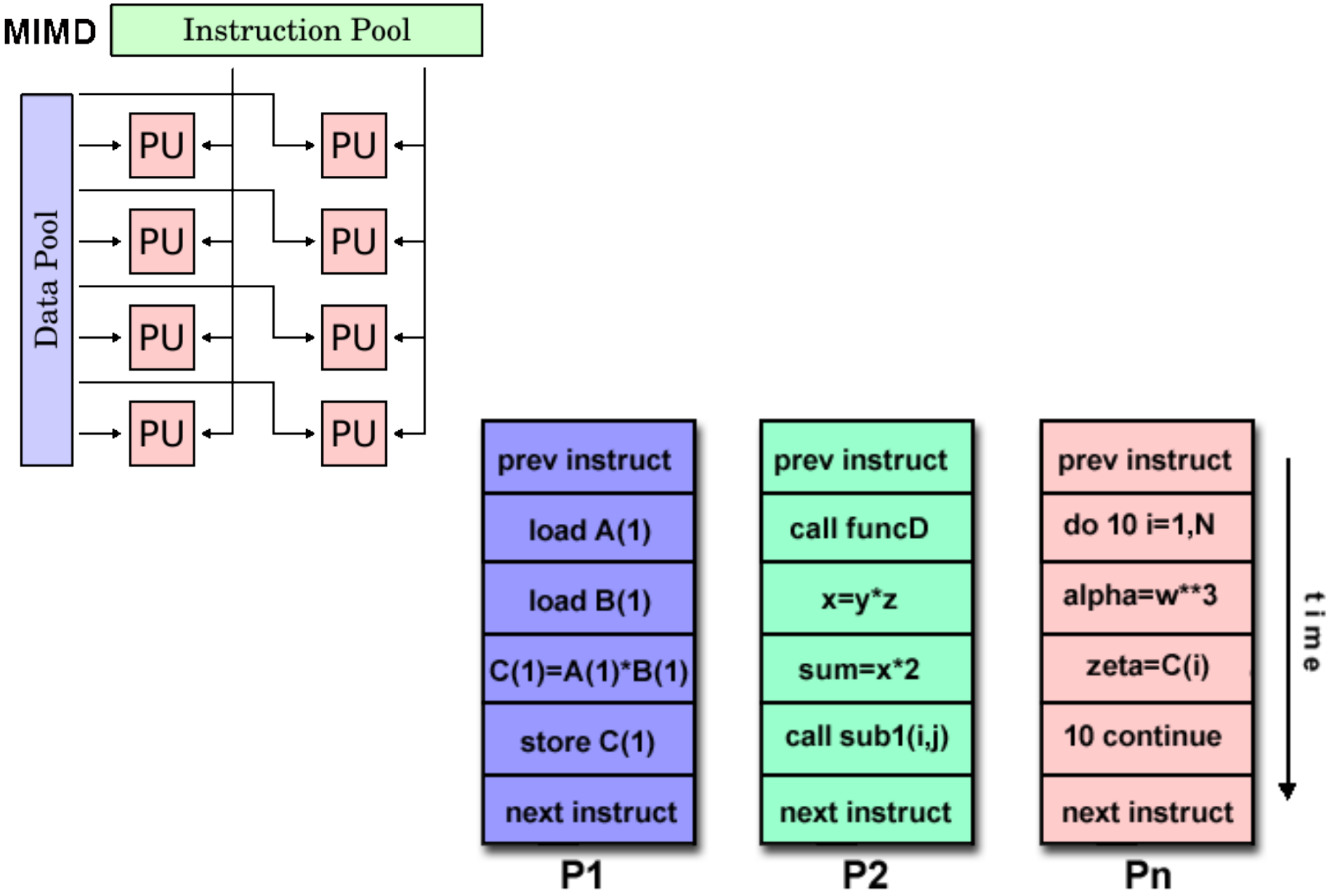
Multiple Instruction, Single Data (MISD)

- A type of parallel computer
- **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.
- Few (if any) actual examples of this class of parallel computer have ever existed.
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.


MISD
Instruction Pool
Data Pool
PU
PU
prev instruct
load A(1)
C(1)=A(1)*1
store C(1)
next instruct
P1
prev instruct
load A(1)
delta=A(1)*4
B(i)=psi+8
next instruct
P2
prev instruct
load A(1)
mat(n)=A(1)
write(mat(n))
next instruct
Pn
time

Multiple Instruction, Multiple Data (MIMD)

- A type of parallel computer
- **Multiple Instruction:** Every processor may be executing a different instruction stream
- **Multiple Data:** Every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Currently, the most common type of parallel computer - most modern supercomputers fall into this category.
- Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.
- **Note: many MIMD architectures also include SIMD execution sub-components**


MIMD
Instruction Pool
Data Pool
PU
PU
PU
PU
PU
PU
PU
PU
prev instruct
load A(1)
load B(1)
C(1)=A(1)*B(1)
store C(1)
next instruct
P1
prev instruct
call funcD
x=y*z
sum=x*2
call sub1(i,j)
next instruct
P2
prev instruct
do 10 i=1,N
alpha=w**3
zeta=C(i)
10 continue
next instruct
Pn
time

Some General Parallel Terminology

- **Supercomputing / High Performance Computing (HPC)**
- **Node:** A standalone "computer in a box". Usually comprised of multiple CPUs/processors/ cores, memory, network interfaces, etc. Nodes are networked together to comprise a supercomputer.
- **CPU / Socket / Processor / Core**
- **Task:** A logically discrete section of computational work
- **Pipelining:** Breaking a task into steps performed by different processor units, with inputs streaming through, much like an assembly line; a type of parallel computing

- **Shared Memory:** a computer architecture where all processors have direct (usually bus based) access to common physical memory
- **Symmetric Multi-Processor (SMP):** Shared memory hardware architecture where multiple processors share a single address space and have equal access to all resources
- **Distributed Memory:** refers to network based memory access for physical memory that is not common
- **Communications:** through a shared memory bus or over a network

- **Synchronization:** The coordination of parallel tasks in real time, very often associated with communications.
  - Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.
- **Granularity:** In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.
  - ***Coarse:*** relatively large amounts of computational work are done between communication events
  - ***Fine:*** relatively small amounts of computational work are done between communication events

- **Speedup:**
  - speedup = time of serial execution / time of parallel execution
- **Parallel Overhead:** The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as:
  - Task start-up time
  - Synchronizations
  - Data communications
  - Software overhead imposed by parallel languages, libraries, operating system, etc.
  - Task termination time

- **Massively Parallel:** Hardware having many processing elements. The meaning of "many" keeps increasing, but currently, the largest parallel computers having **hundreds of thousands to millions** “processing elements”.
- **Embarrassingly Parallel:** Solving many similar, but independent tasks simultaneously -need for coordination between the tasks.
- **Scalability:** ability to demonstrate a proportionate increase in parallel speedup. Factors that contribute to scalability include:
  - Hardware - particularly memory-cpu bandwidths and network communication properties
  - Application algorithm
  - Parallel overhead related
  - Characteristics of your specific application

Limits and Costs of Parallel Programming

- Amdahl's Law: **Amdahl's Law** states that potential program speedup is defined by the fraction of code (P) that can be parallelized
  - **Speedup = [1/ (1 - P)]**
- If none of the code can be parallelized, P = 0 and the speedup = 1 (no speedup).
- If all of the code is parallelized, P = 1 and the speedup is infinite (in theory).
- If 50% of the code can be parallelized, maximum speedup = 2, meaning the code will run twice as fast.
- Introducing the number of processors performing the parallel fraction of work, the relationship can be modelled by:
  - **Speedup = [1/ (P/N + S)]**
- where P = parallel fraction, N = number of processors and S = serial fraction.

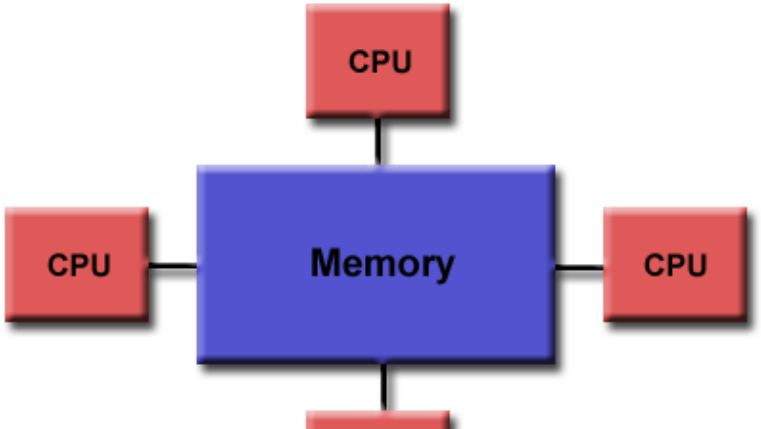
- It soon becomes obvious that there are limits to the scalability of parallelism.

Speedup

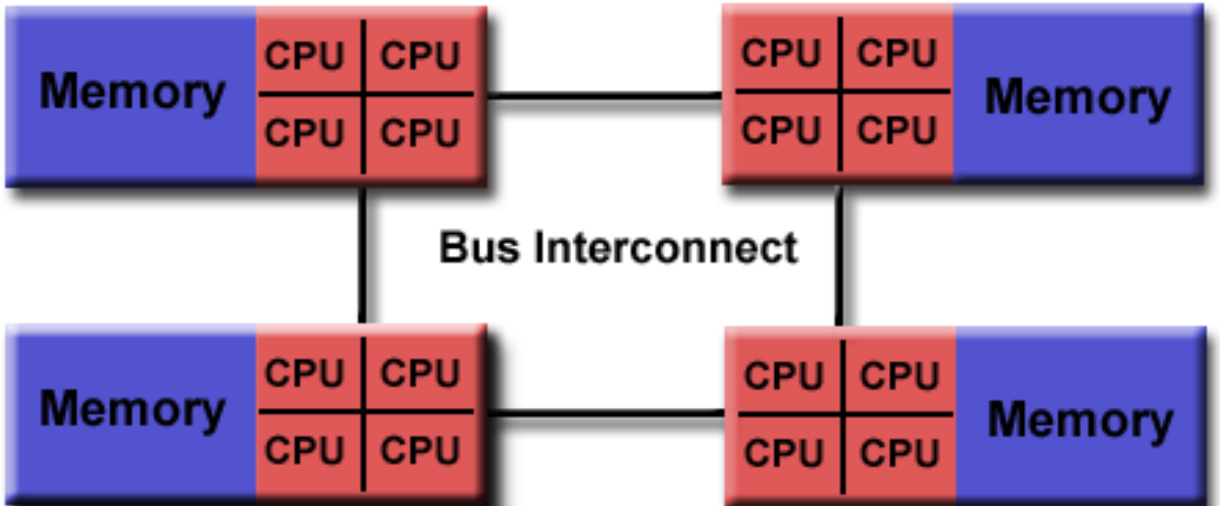
N	P = .50	P = .90	P = .95	P = .99
10	1.82	5.26	6.89	9.17
100	1.98	9.17	16.80	50.25
1,000	1.99	9.91	19.62	90.99
10,000	1.99	9.91	19.96	99.02
100,000	1.99	9.99	19.99	99.90

Parallel Computer Memory Architectures

- **Shared Memory : UMA , NUMA**
- Uniform Memory Access (UMA)
  - Most commonly represented today by ***Symmetric Multiprocessor (SMP)*** machines , identical processors
  - Sometimes called CC-UMA - Cache Coherent UMA. Cache coherent 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



- **Advantages:**
  - 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
- **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 ensure "correct" access of global memory.

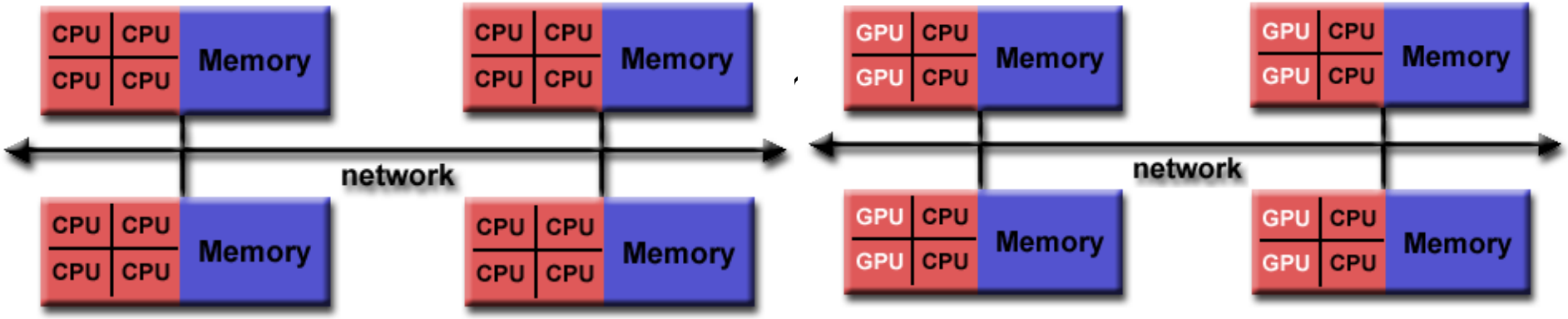
Distributed Memory

- Distributed memory systems require a communication network to connect inter-processor memory.
- Processors have their own local memory. No concept of global address space across all processors.
- Processor has its own local memory, it operates independently. Hence, the concept of cache coherency does not apply.
- 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. Synchronization between tasks is likewise the programmer's responsibility.

- **Advantages:**
  - Memory is scalable with the 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 global cache coherency.
  - Cost effectiveness: can use commodity, off-the-shelf processors and networking.
- **Disadvantages:**
  - Programmer is responsible for 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 times - data residing on a remote node takes longer to access than node local data.

Hybrid Distributed-Shared Memory

- The largest and fastest computers in the world today employ both shared and distributed memory architectures.
- The shared memory component can be a shared memory machine and/or graphics processing units (GPU).
- The distributed memory component is the networking of multiple shared memory/GPU machines, which know only about their own memory - not the memory on another machine.



- **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