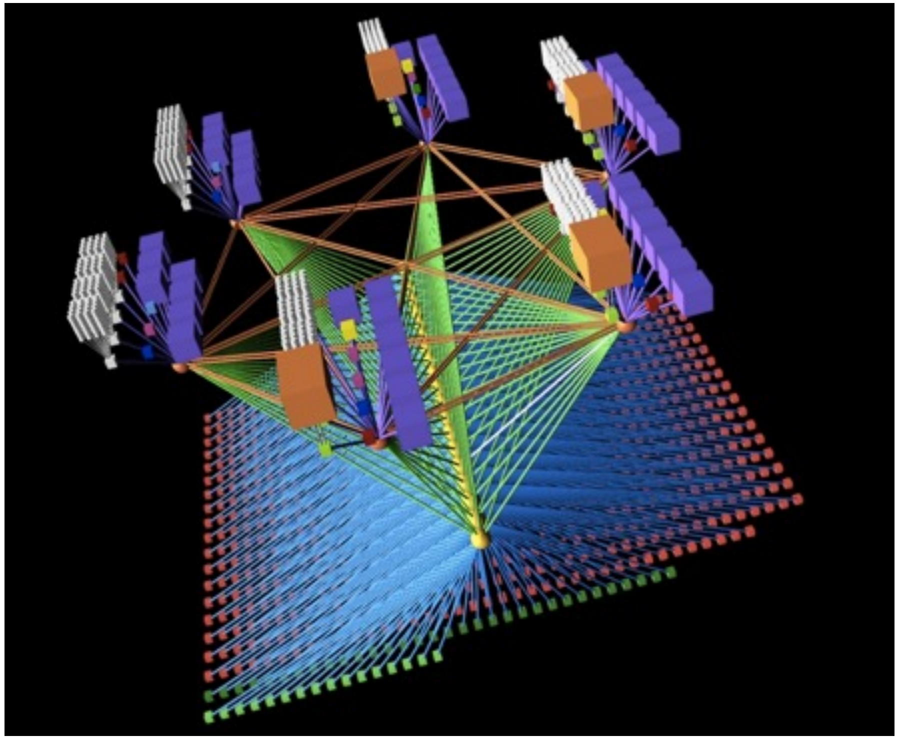
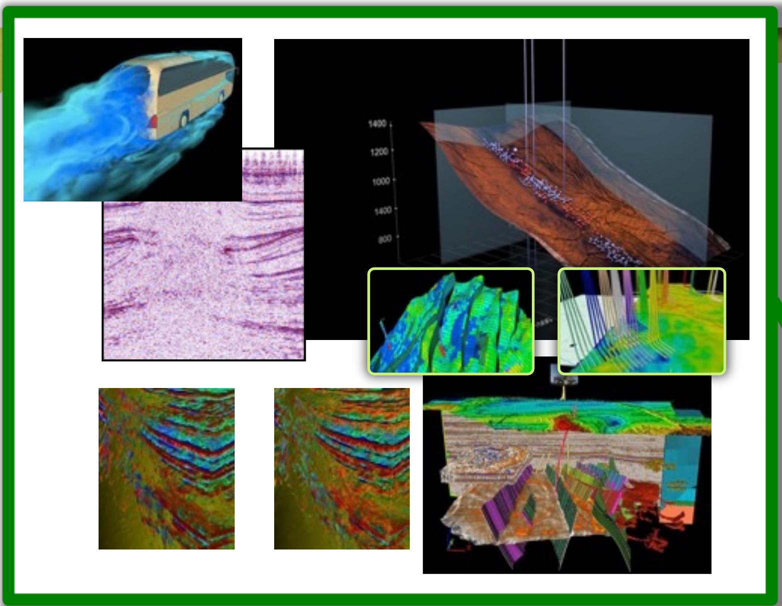


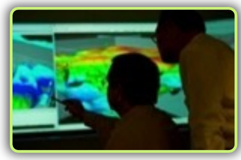
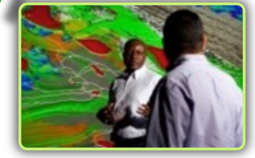
Parallel and Scalable Architectures

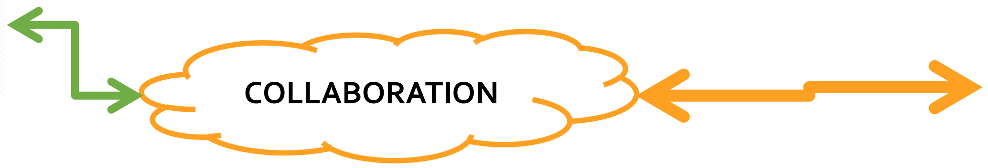
Carlos Jaime Barrios Hernandez
@carlosjaimebh

Why?

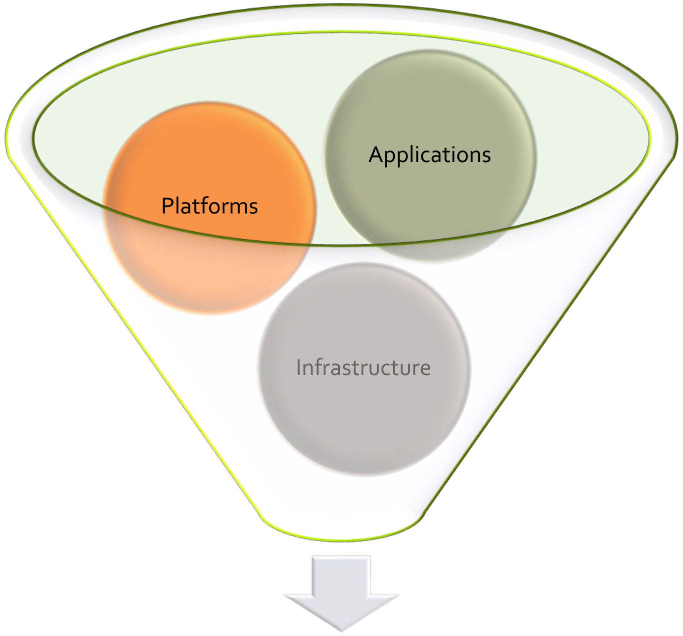


- Large Data Sets
- Complex Mathematics
- Complex Models
- Real Time
- Interaction and Confrontation
- Large Scale Visualization
- High Resolution
- High Performance and Capacity
  - VR Needs
  - Big Data and Deep Learning



Big Problems, Smart Solutions



High Performance (Computing) Knowledge

Challenges

Infrastructure

- Post Moore Era Architectures
  - Parallel Balancing, I/O, Memory Challenges
- Dark Sillico
- Exascale
  - Computer Efficiency (Processing/Energy Consumption)
- Hybrid Platforms (CISC+RISC+Others)
  - TPUs, ARM...
- Data Management
- Advanced Networks
- Fog/Edge
- HPC@Pocket
- ... Quantum Computing

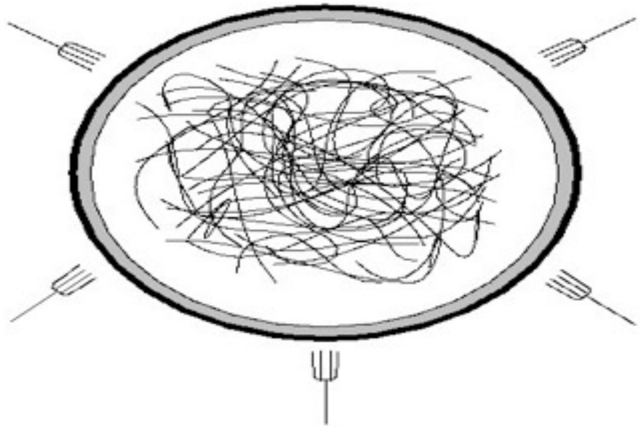
Platform

- Programmability
  - New Languages and Compilers
- Computing Efficiency
- Data Movement and Processing (In Situ, In Transit, Workflows)
- HPC as a Service
  - Science Gateways, Containers
- Viz as a Service (In Situ)
- Protocols
- IA and Deep Learning Frameworks
- Quantum Computing

Applications

- IA and Deep Learning
- Algorithms Implementation
- Use of Interpretators (as Python)
- Community versions
- Open Algorithms, Open Data
- Utra Scale Applicatons
- ...and more!

About Parallelism



- **Concurrency** is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other.

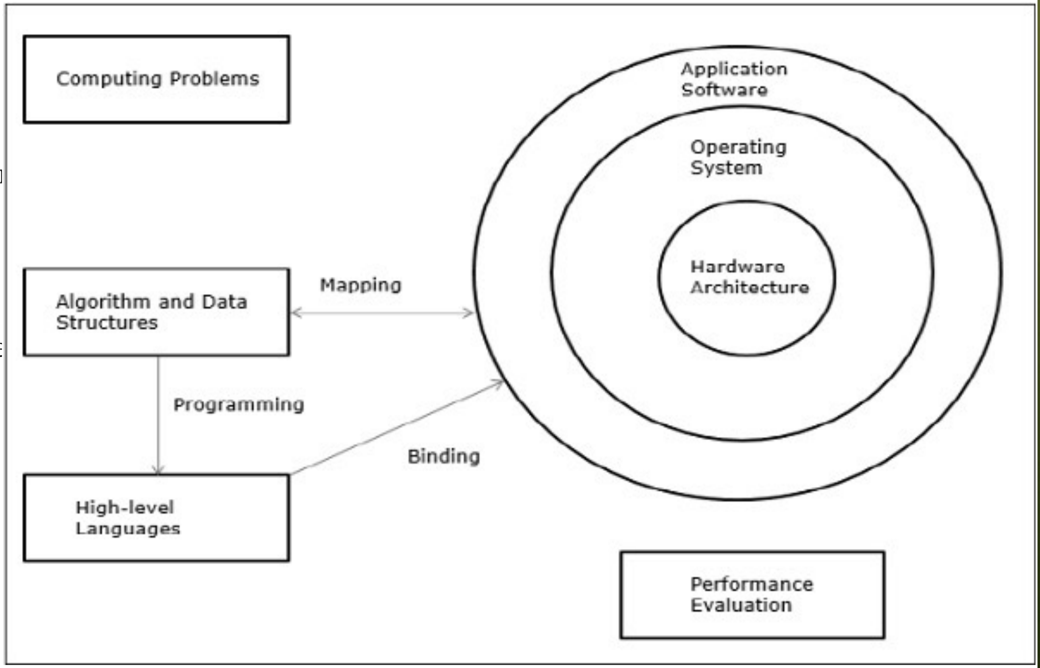
- **Implicit parallelism** is a characteristic of a programming language that allows a compiler or interpreter to automatically exploit the parallelism inherent to the computations expressed by some of the language's constructs.

- **Explicit parallelism** is the representation of concurrent computations by means of primitives in the form of special-purpose directives or function calls.

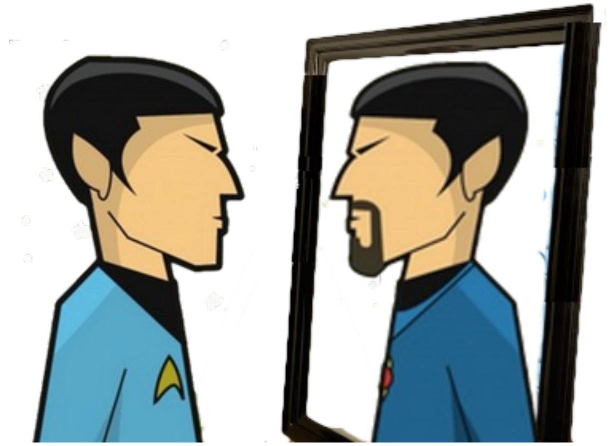
- We need two (mixed) approach in Architecture: Applications and Hardware (system).

Elements of Parallelism

1. Computing Problems
   - Numerical (Intensive Computing, Large Data Sets)
   - Logical (AI Problems)
2. Parallel Algorithms and Data Structures
   - Special Algorithms (Numerical, Symbolic)
   - Data Structures (Dependency Analysis)
   - Interdisciplinary Action (Due to the Computing Proble
3. System Software Support
   - High Level Languages (HLL)
   - Assemblers, Linkers, Loaders
   - Models Programming
   - Portable Parallel Programming Directives and Librarie
   - User Interfaces and Tools
4. Compiler Support
   - Implicit Parallelism Approach
     - Parallelizing Compiler
     - Source Codes
   - Explicit parallelism Approach
     - Programmer Explicitly
       - Sequential Compilers, Low Level Libraries
       - Concurrent Compilers (HLL)
     - Concurrency Preserving Compiler
5. Parallel Hardware Architecture
   - Processors
   - Memory
   - Network and I/O
   - Storage

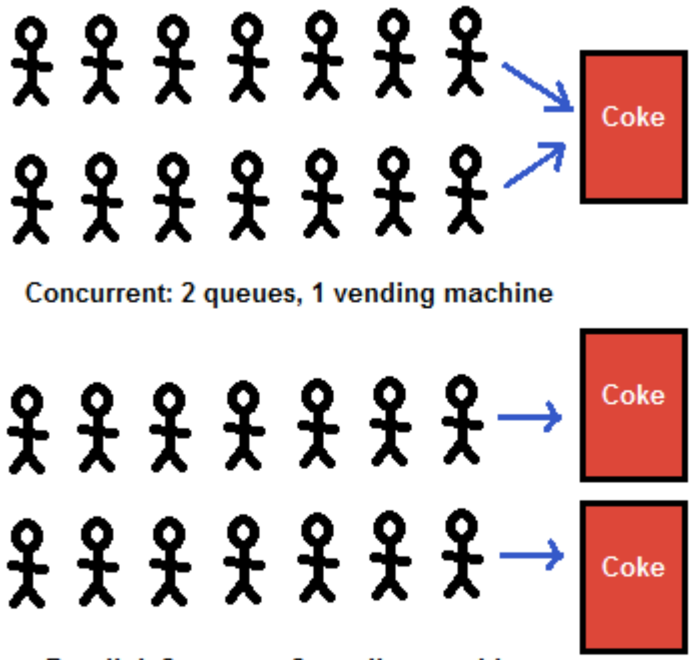


Pervasive and Thinking Parallelism



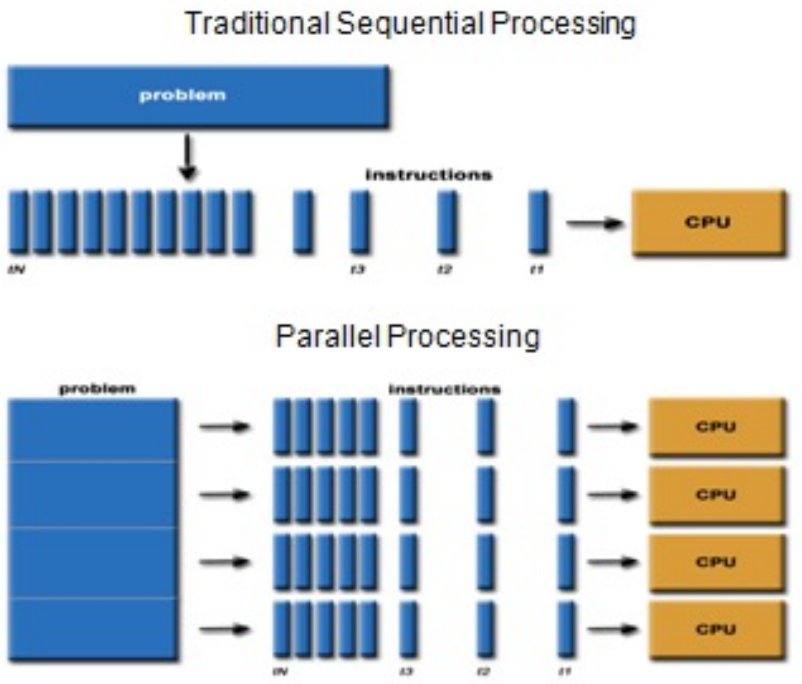
- It is not a question of « Parallel Universes » (Almost)
- Data Sources
- Processing and Treatment
- Resources (Available and Desire)
- Energy Consumption
- Natural “thinking” (Natural Compute?)

Thinking in Parallel (computing) – The Typical Visions

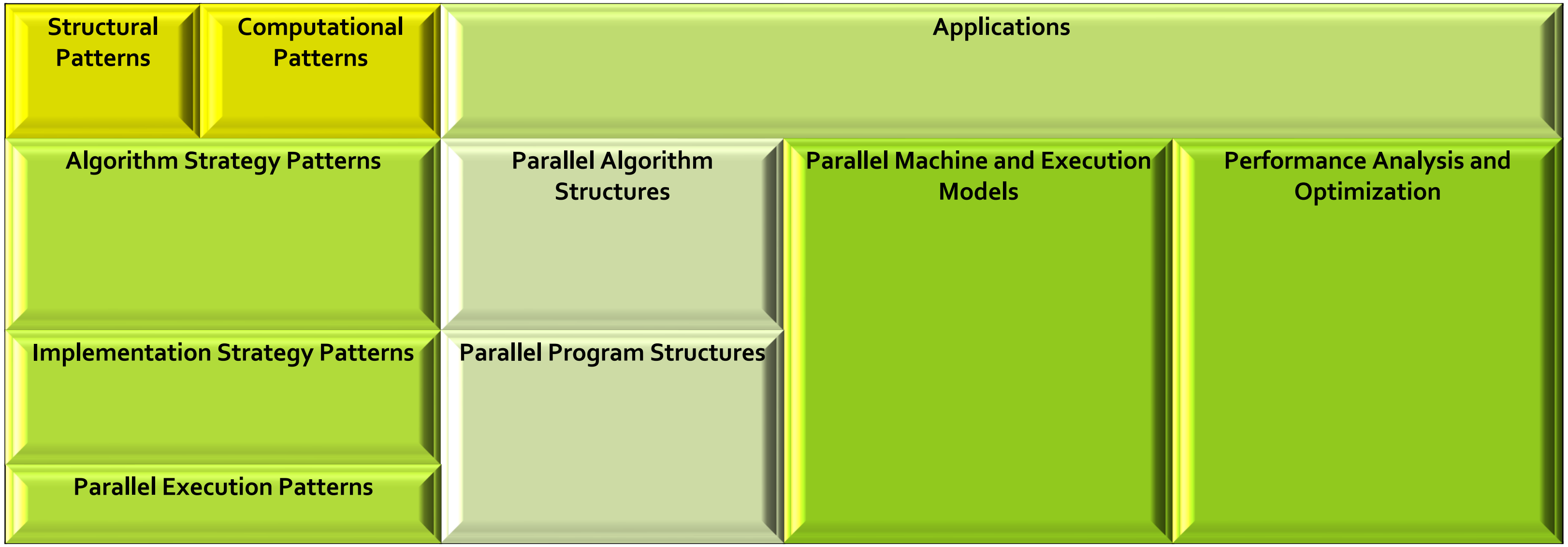


Concurrent: 2 queues, 1 vending machine

Parallel: 2 queues, 2 vending machines

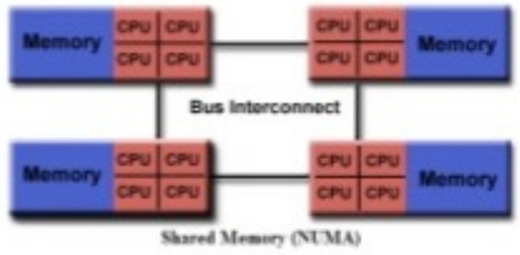


Thinking in Parallel (computing) – an OPL hierarchy

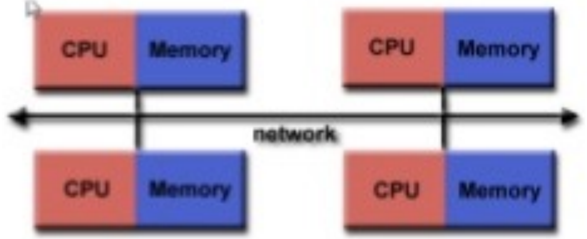

Structural Patterns
Computational Patterns
Applications
Algorithm Strategy Patterns
Parallel Algorithm Structures
Parallel Machine and Execution Models
Performance Analysis and Optimization
Implementation Strategy Patterns
Parallel Program Structures
Parallel Execution Patterns

Parallel Computing Architectures

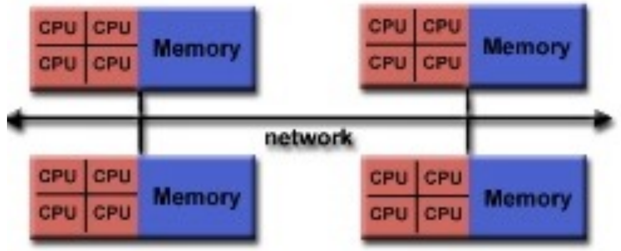
Shared Memory



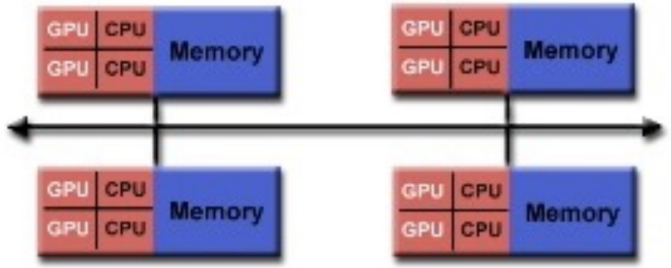
Distributed Memory



Homogeneous Architecture

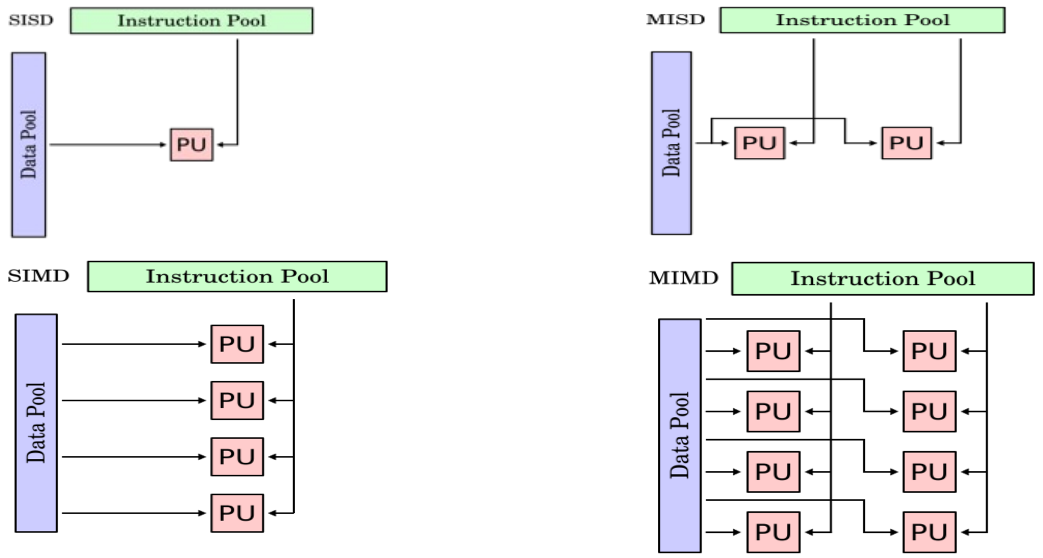


Heterogeneous Architecture



[redacted]

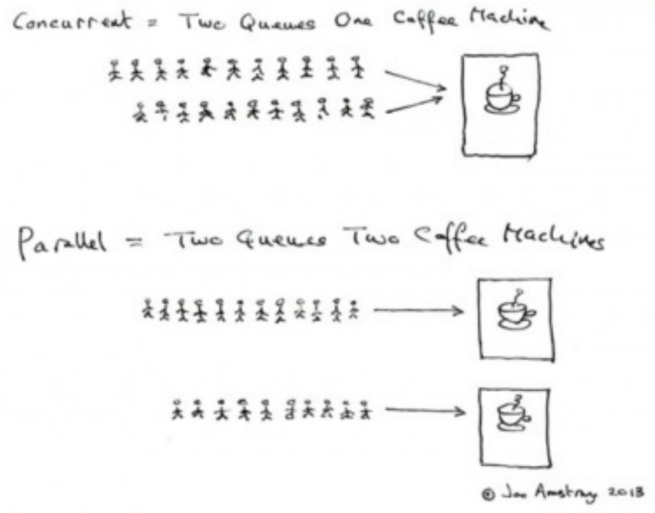
Flynn's Taxonomy*

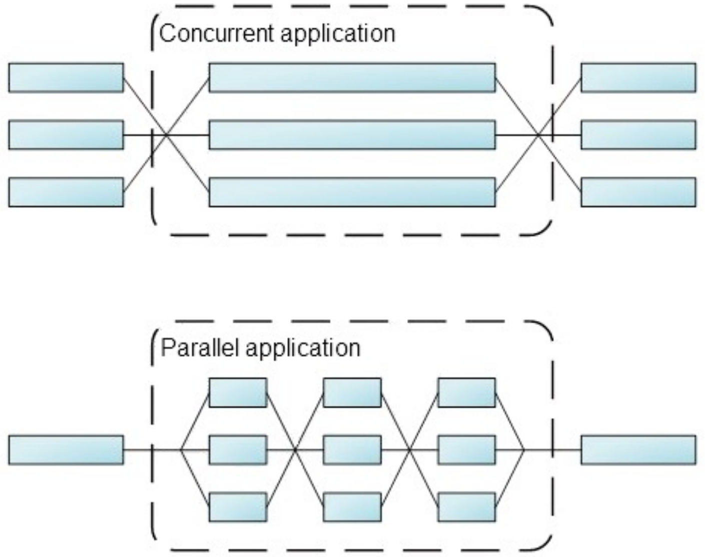


* Proposed by M. Flynn in 1966

[redacted]

CONCURRENCY | PARALLELISM


Concurrent = Two Queues One Coffee Machine
Parallel = Two Queues Two Coffee Machines
© Joe Armstrong 2013

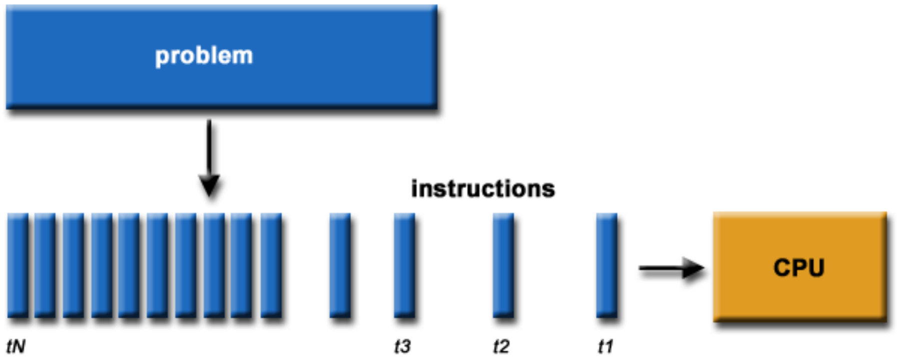


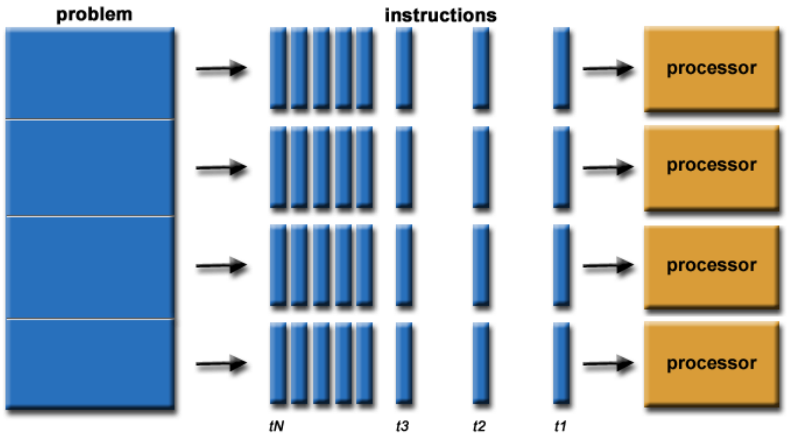
From J. Armstrong Notes: http://joearms.github.io/2013/04/05/concurrent-and-parallel-programming.html

Any Parallel System is concurrent: Simulatenous Processing, Parallel but limited ressources.

[redacted]

Serial vs Concurrent/Parallel Approach

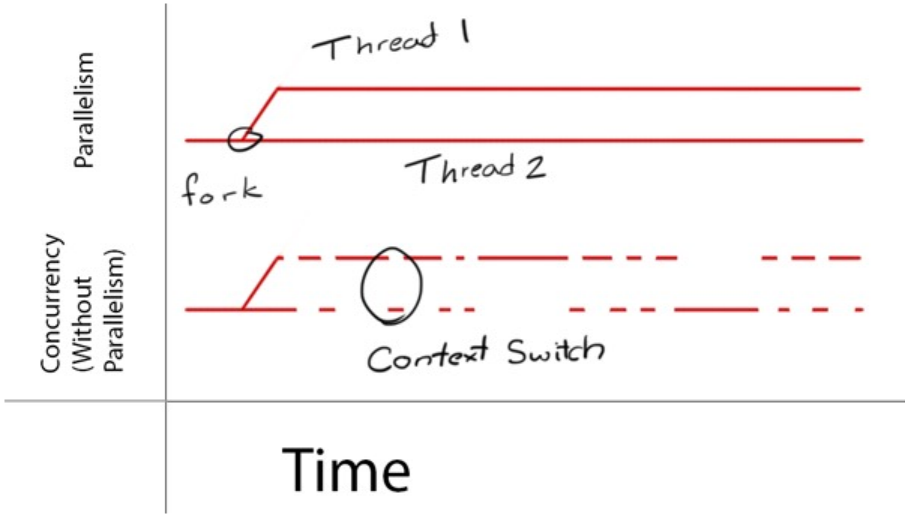




Reduction in Execution Time (However, overhead problem)
Instructions to Multithreading (To exploit Parallelism)
Syncrhonization (with all derivated concerns...)

[redacted]

Concurrency vs Concurreny/Parallelism Behavior

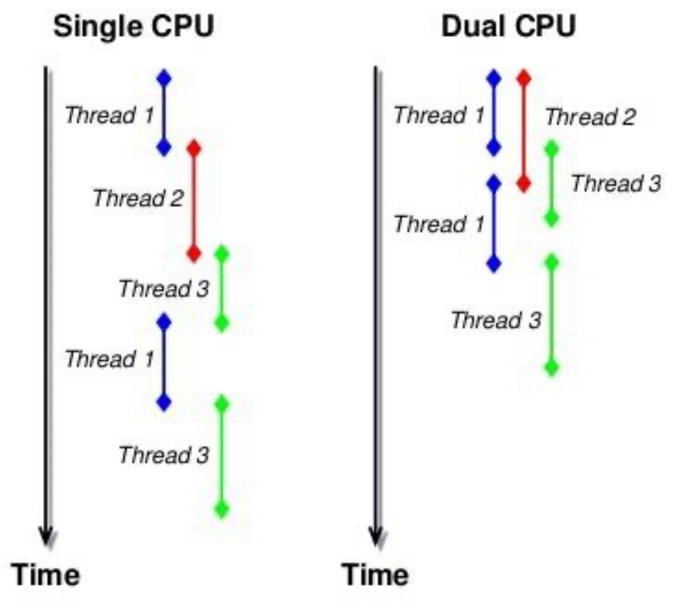


Non Shared Processing Ressources (However the Memory...)
Switching
Parallel Threards (Multitasking, Multithreading)

Shared Processing Ressources
Switching
Non Parallel Threards (Non Multitasking, Yes Multithreading)

[redacted]

Concurrency vs Concurreny/Parallelism Example



Single System
- Multiple Threads in Runtime
- Almost Synchronization Strategies
- Memory Allocation

Dual System
- Multiple Parallel Threads in Runtime
- Strategies to Paralellism following models (PRAM, LogP, etc) addressed to exploit memory and overhead reduction

+

Remember Computer Architecture Representation

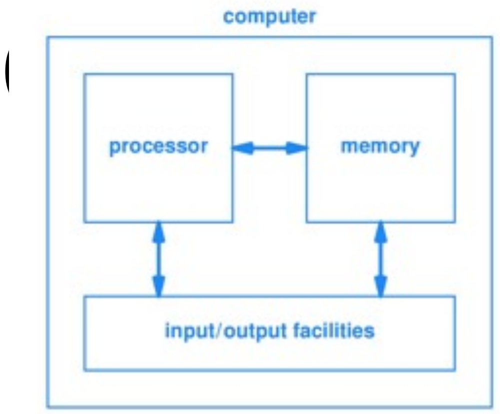


Illustration of the Von Neumann Architecture. Both programs and data can be stored in the same memory.

From https://eca.cs.purdue.edu/index.html

- Von Neumann Representation
  - Von Neumann Computer Machines
  - Classical Computers
- Input
- Processor
- Memory
- Output

Remember Computer Architecture Representation (

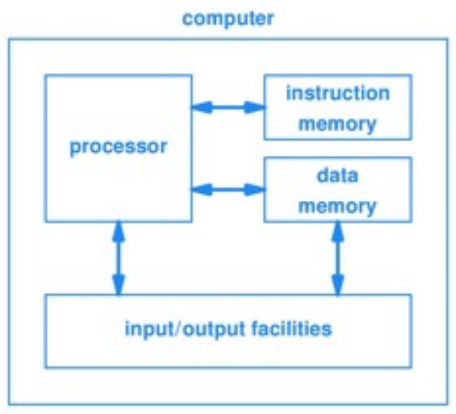


Illustration of the Harvard Architecture that uses two memories, one to hold programs and another to store data.

From https://eca.cs.purdue.edu/index.html

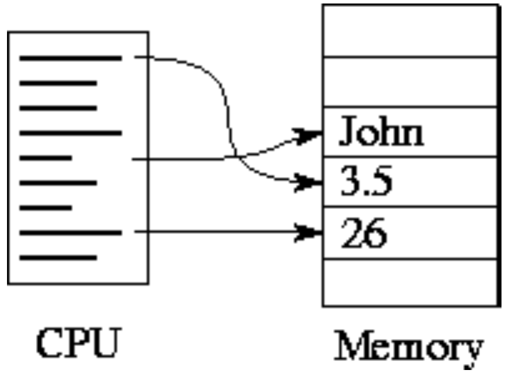
- Non- Von Newmann Representation
  - Non Von Newmann Computer Machines
    - Quantum Computers
    - Harvard Architectures
    - Hybrid Computers (or Post Moore Architectures)
- Processor
- Data Memory
- Instruction Memory
- Output

+ Machine Models

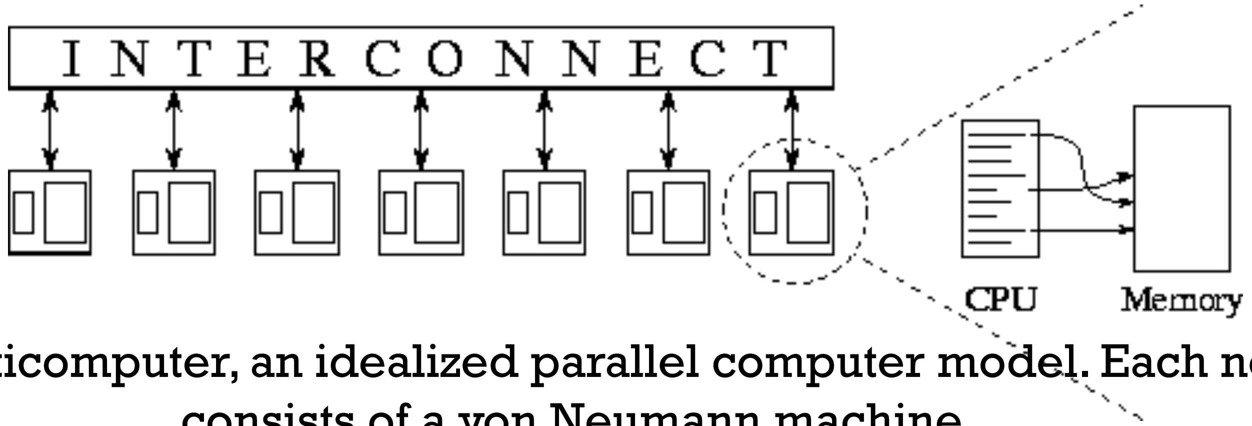
- What is a machine model?
  - A abstraction describes the operation of a machine.
  - Allowing to associate a value (cost) to each machine operation.

- Why do we need models?
  - Make it easy to reason algorithms
  - Hide the machine implementation details so that general results that apply to a broad class of machines to be obtained.
  - Analyze the achievable complexity (time, space, etc) bounds
  - Analyze maximum parallelism
  - Models are directly related to algorithms.

Parallel Machine Model



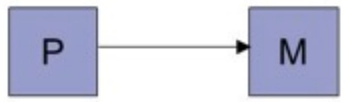
The von Neumann computer



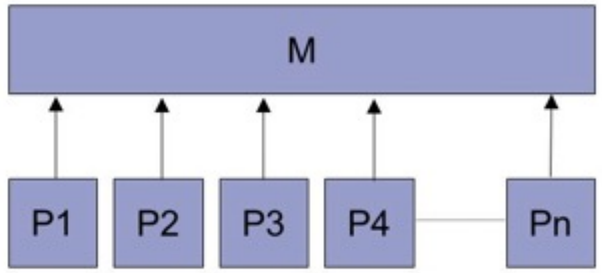
The multicomputer, an idealized parallel computer model. Each node consists of a von Neumann machine

RAM and PRAM Models

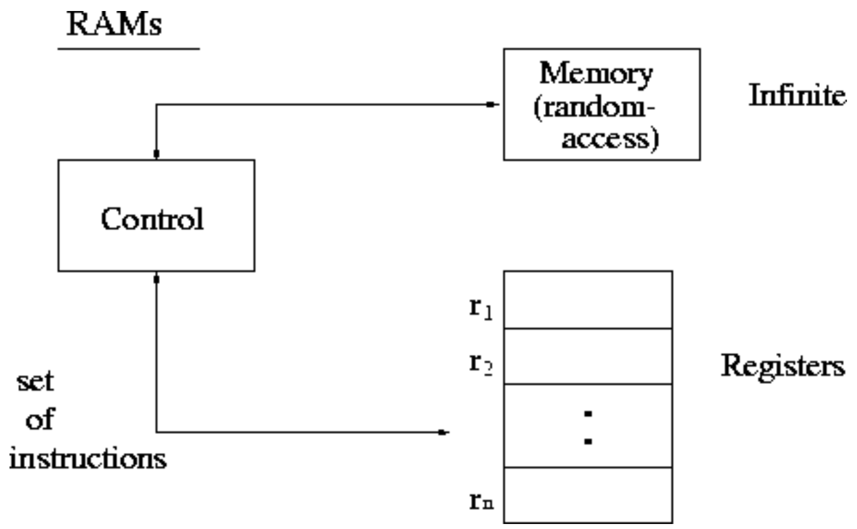
RAM. Random Access Machine



PRAM. Parallel Random Access Machine



+ RAM (Random Access Machine) model

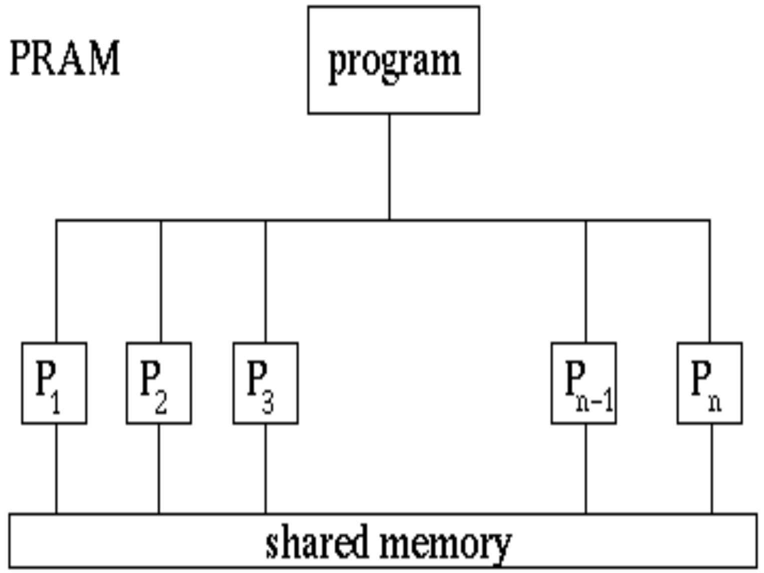


- Memory consists of infinite array (memory cells).
- Each memory cell holds an infinitely large number.
- Instructions execute sequentially one at a time.
- All instructions take unit time
  - Load/store
  - Arithmetic
  - Logic
- Running time of an algorithm is the number of instructions executed.
- Memory requirement is the number of memory cells used in the algorithm.

RAM (random access machine) model

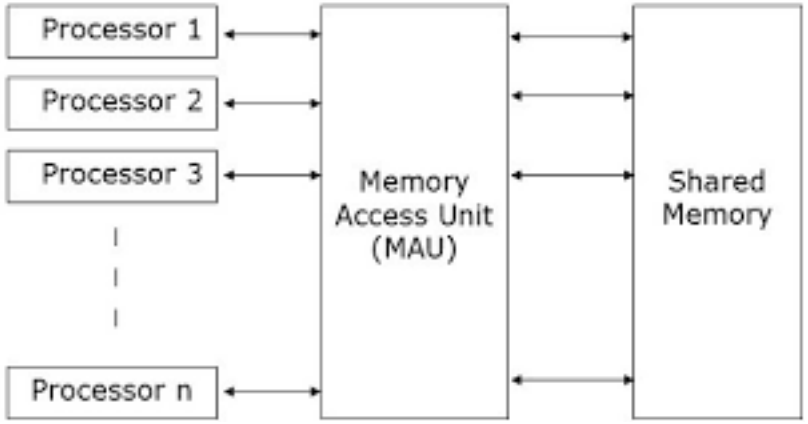
- The RAM model is the base of algorithm analysis for sequential algorithms although it is not perfect.
  - Memory not infinite
  - Not all memory access take the same time
  - Not all arithmetic operations take the same time
  - Instruction pipelining is not taken into consideration
- The RAM model (with asymptotic analysis) often gives relatively realistic results.

+
PRAM (Parallel RAM)



- A model developed for parallel machines
  - An unbounded collection of processors
  - Each processor has an infinite number of registers
  - An unbounded collection of shared memory cells.
  - All processors can access all memory cells in unit time (when there is no memory conflict).
  - All processors execute PRAM instructions synchronously
    - Somewhat like SIMD, except that different processors can run different instructions in the lock step.
    - Some processors may idle.

+ PRAM (Parallel RAM)



- A model developed for parallel machines
  - Each PRAM instruction executes in 3-phase cycles
    - Read from a share memory cell (if needed)
    - Computation
    - Write to a share memory cell (if needed)
    - Example: for all I, do A[i] = A[i-1]+1;
      - Read A[i-1], compute add 1, write A[i]

– The only way processors exchange data is through the shared memory.

+

PRAM (Parallel RAM)

- PRAM Provides an ideal model of a parallel machine (computer) for analyzing the efficiency of parallel algorithms.

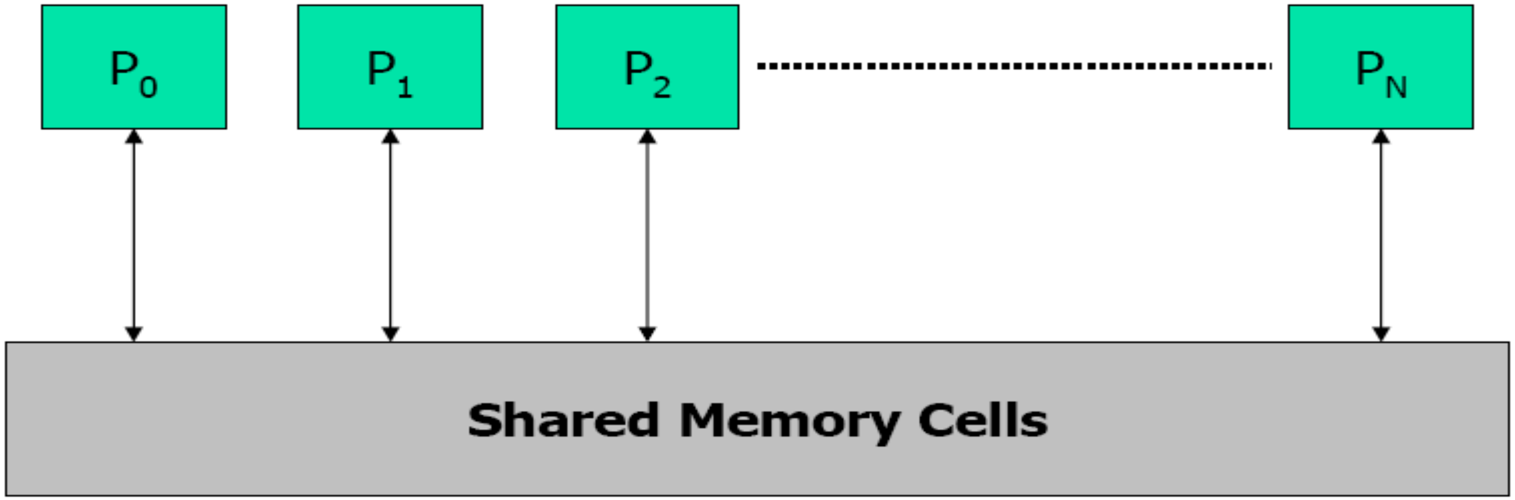
- PRAM composed of
  - P Unmodifiable programs, each composed of optionally labeled instructions
  - a single shared memory composed of a sequence of words, each capable of containing an arbitary integer
  - P accumulators, one associated with each program
    - A read-only input tape
    - A write-only output tape

Parallel time complexity: the number of synchronous steps in the algorithm

Space complexity: the number of share memory

Parallelism: the number of processors used

PRAM (A simple vision)



All processors can do things in a synchronous manner (with infinite shared Memory and infinite local memory), how many steps do it take to complete the task?

+ PRAM model variations

- EREW, CREW, CRCW (common), CRCW (random), CRCW (Priority)
  - Which model is closer to the practical SMP or multicore machines?
- Model A is computationally stronger than model B if and only if any algorithm written in B will run unchange in A.
  - EREW <= CREW <= CRCW (common) <= CRCW (random)pr
  - *And there are other models as BSP, LogP… (To Explore)*

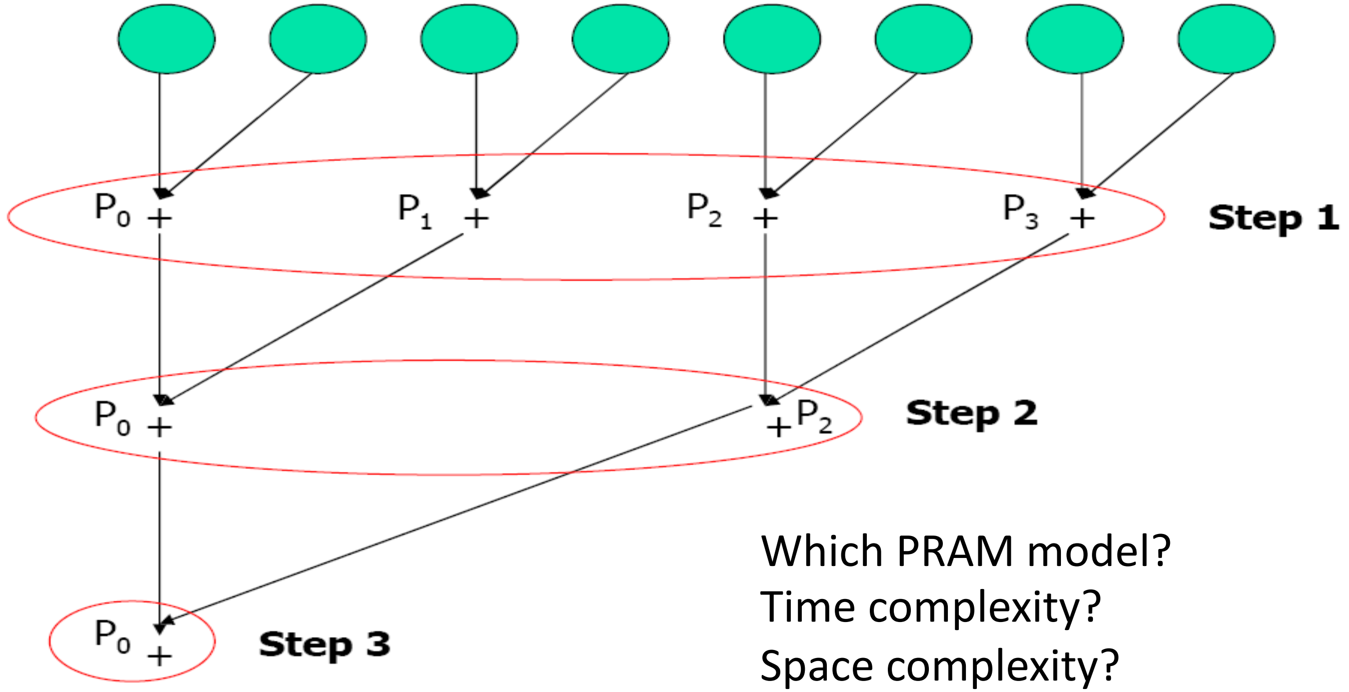
+ PRAM algorithm example

- SUM: Add N numbers in memory M[0, 1, …, N-1]
- Sequential SUM algorithm (O(N) complexity)

```
for (i=0; i<N; i++) sum = sum + M[i];
```

- PRAM SUM algorithm?

+ PRAM SUM algorithm



Which PRAM model?
Time complexity?
Space complexity?
Parallelism?
Speedup (.vs. sequential code)?

+ Parallel Adition

- Time complexity: log(n) steps
- Parallelism: n/2 processors
- Speed-up (vs sequential algorithm): n/log(n)

+ PRAM strengths

- Natural extension of RAM
- It is simple and easy to understand
  - Communication and synchronization issues are hided.
- Can be used as a benchmark
  - If an algorithm performs badly in the PRAM model, it will perform badly in reality.
  - A good PRAM program may not be practical though.
- It is useful to reason threaded algorithms for SMP/multicore machines.

+ PRAM weaknesses

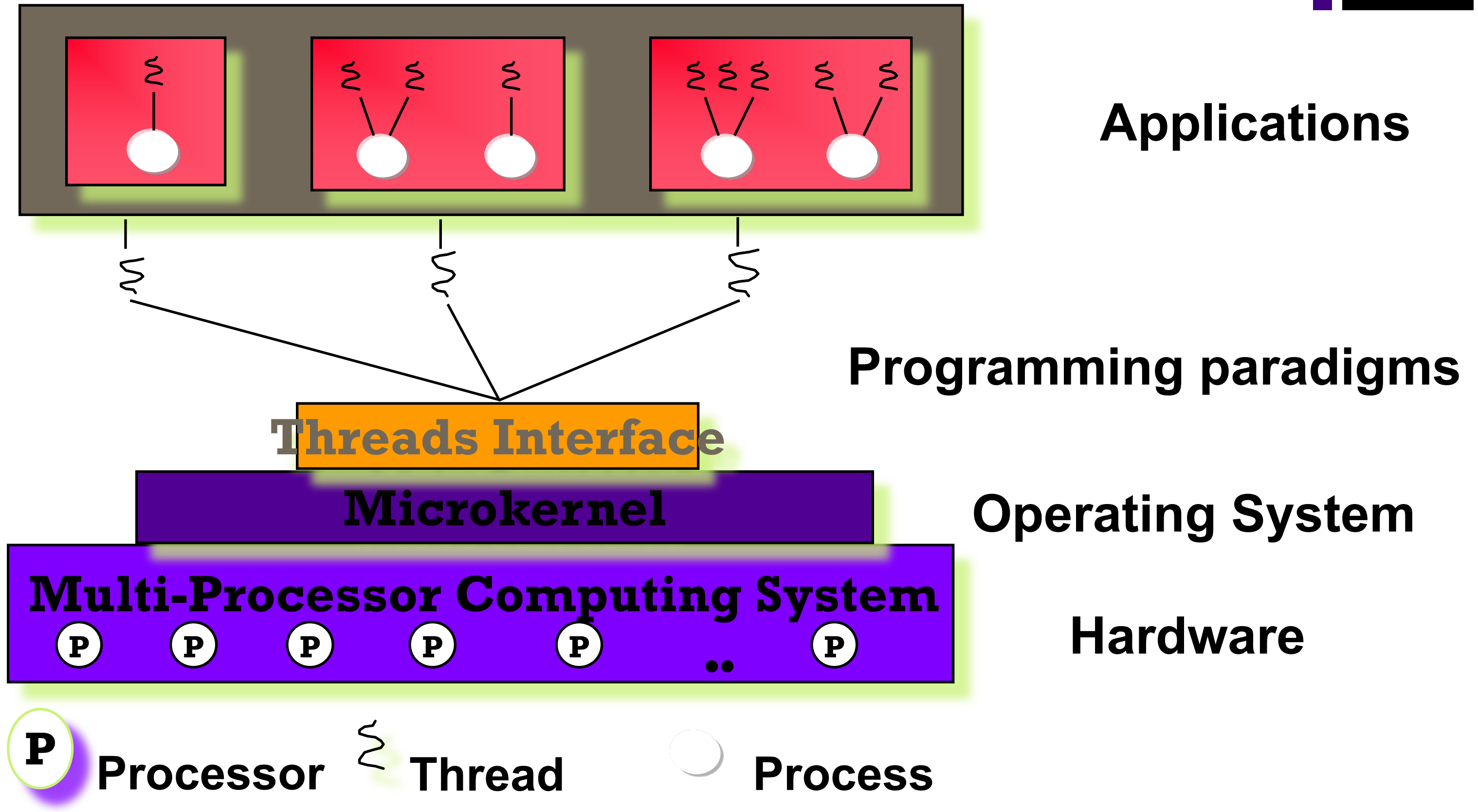
- Model inaccuracies
  - Unbounded local memory (register)
  - All operations take unit time
  - Processors run in lock steps
- Unaccounted costs
  - Non-local memory access
  - Latency
  - Bandwidth
  - Memory access contention

PRAM summary

- The RAM model is widely used.
- PRAM is simple and easy to understand
  - This model never reaches beyond the algorithm community.
  - It is getting more important as threaded programming becomes more popular.
- The BSP (bulk synchronous parallel) model is another try after PRAM.
  - Asynchronously progress
  - Model latency and limited bandwidth

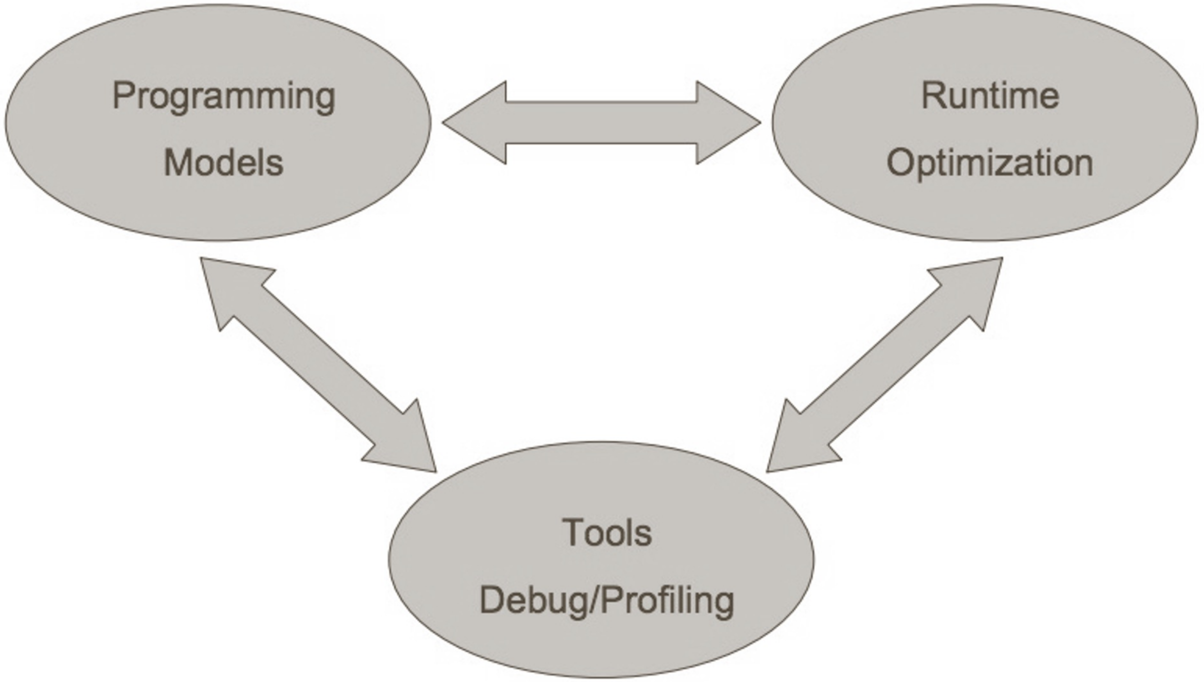
+

Remember…Computing Elements



+

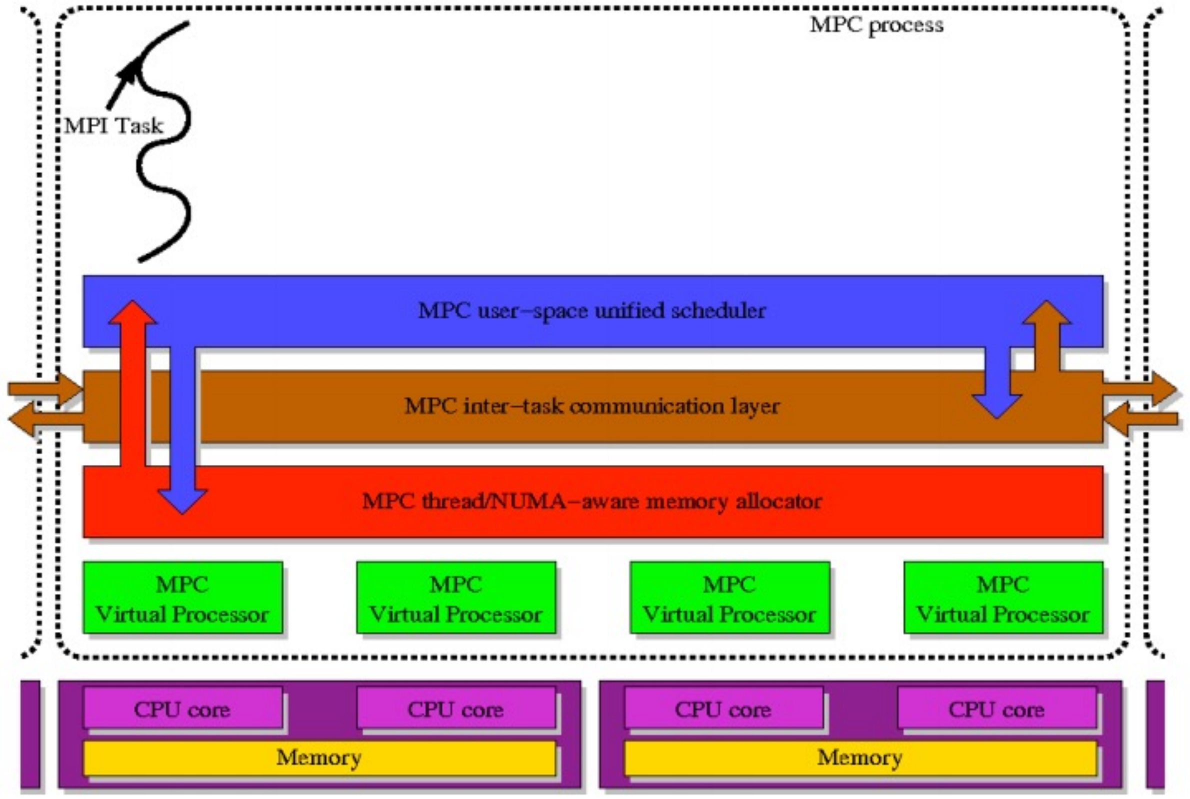
Multi Processor Computing Framework



+

MPC Execution Model

(An Example)

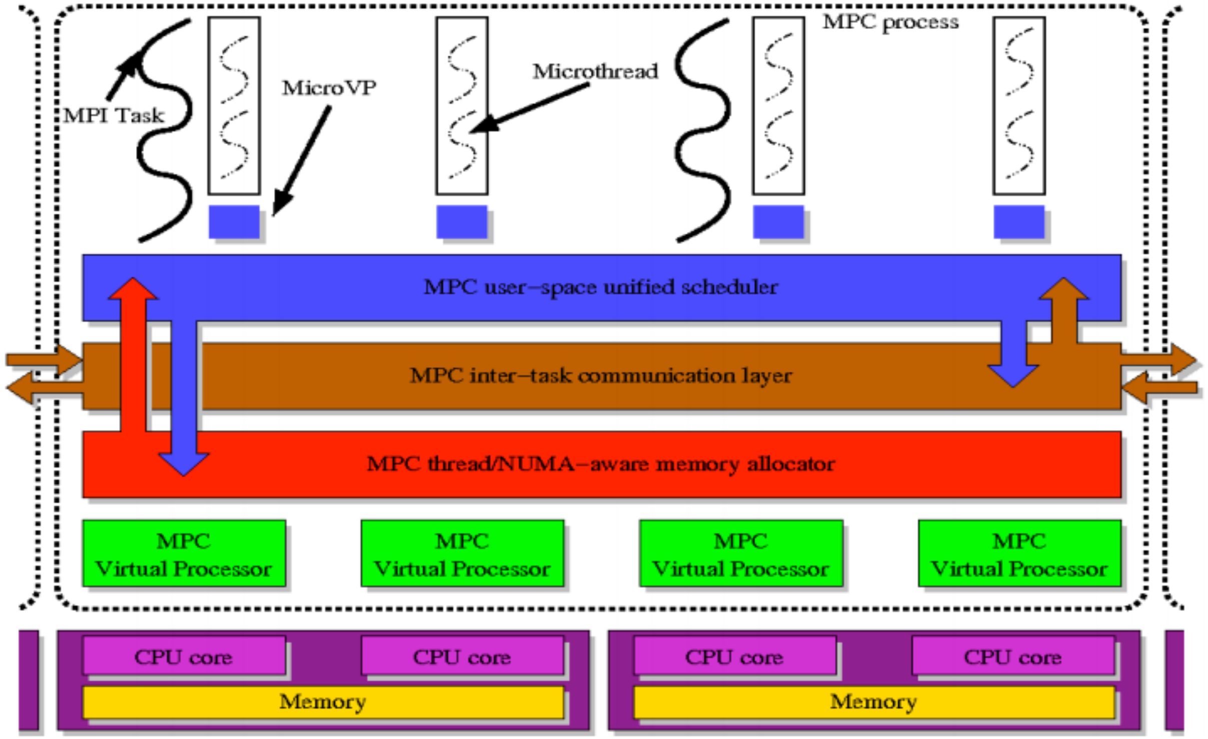


http://calcul.math.cnrs.fr/IMG/pdf/Ecole_hybride.pdf

+
MPC Execution Model

(An example of 4 Tasks and 4 threads)

2 MPI tasks + OpenMP parallel region w/ 4 threads (on 2 cores)

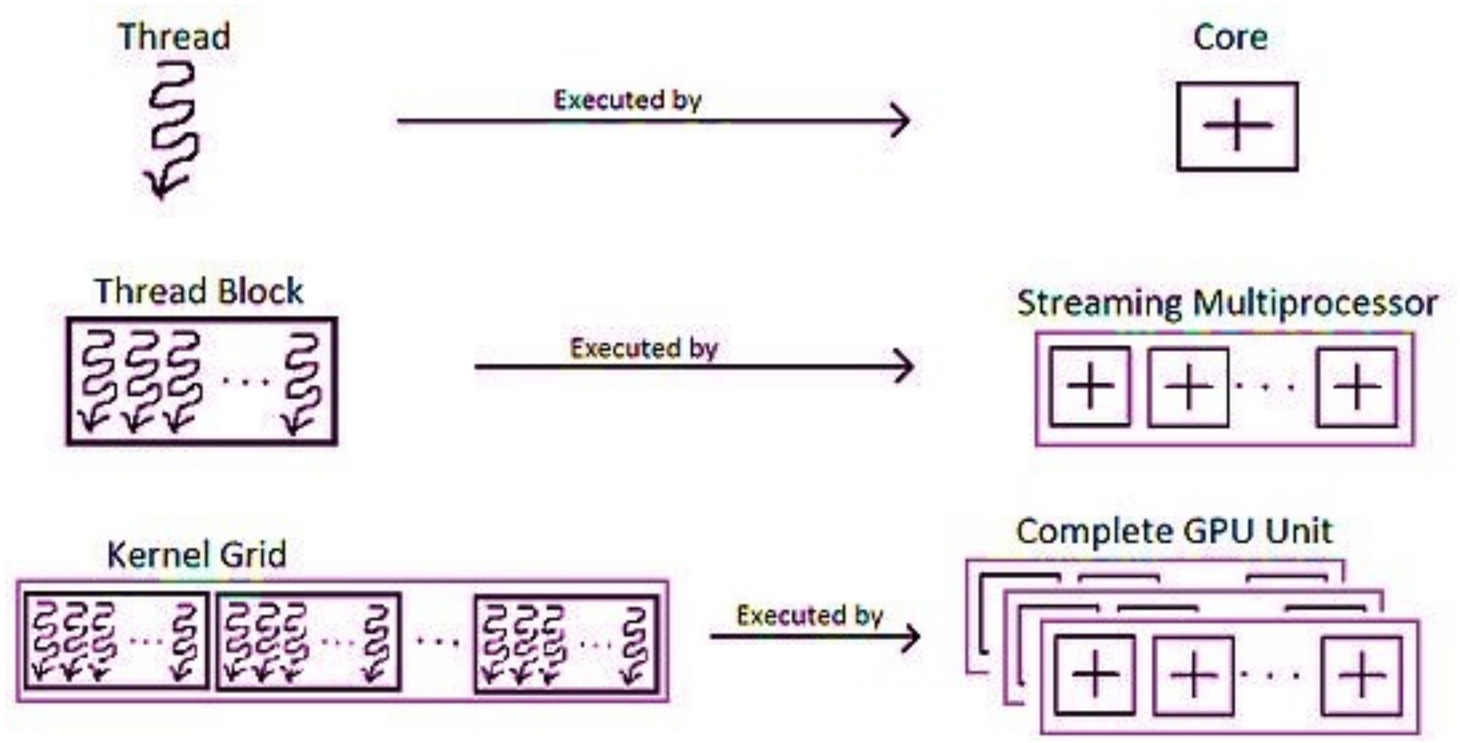


http://calcul.math.cnrs.fr/IMG/pdf/Ecole_hybride.pdf

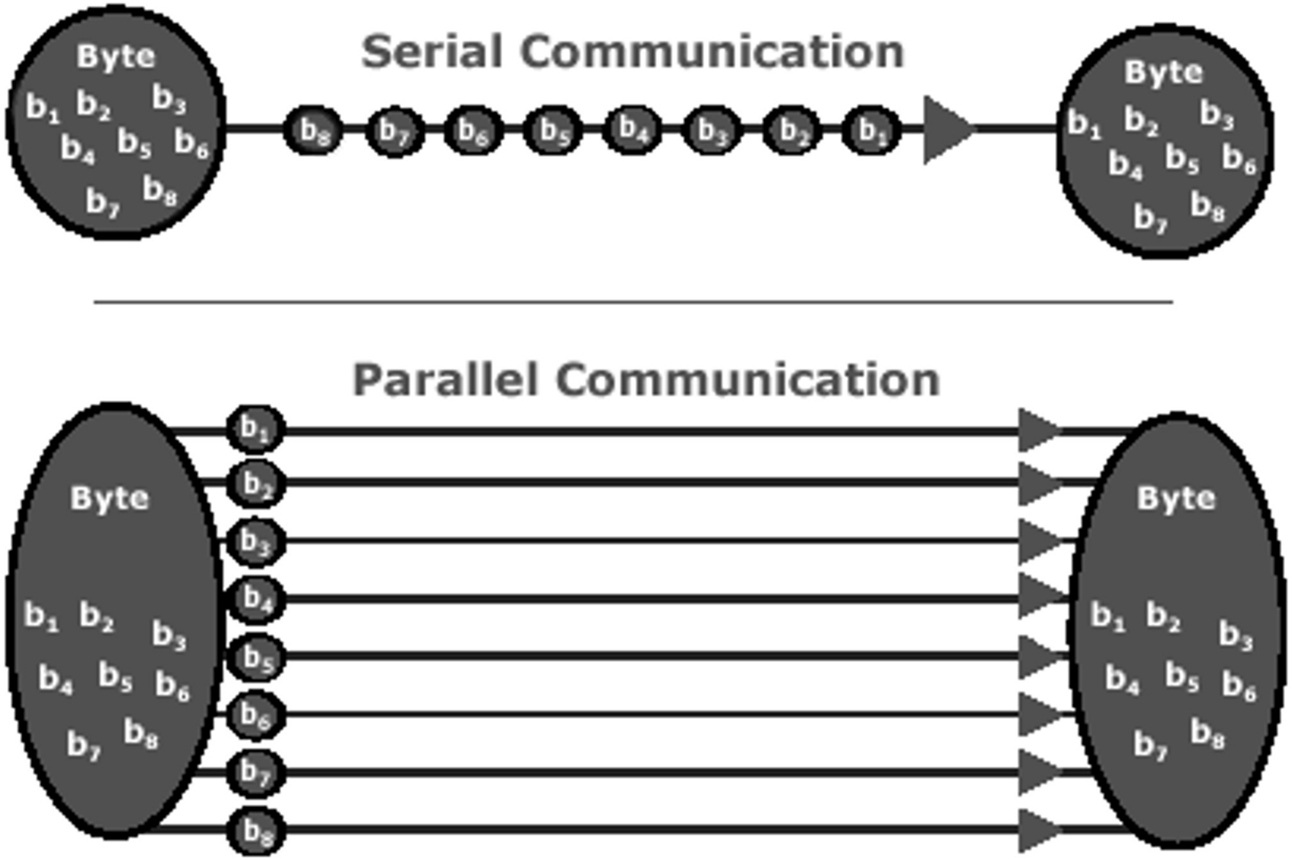
+

MPC Execution Model

(An Example with GPUs)

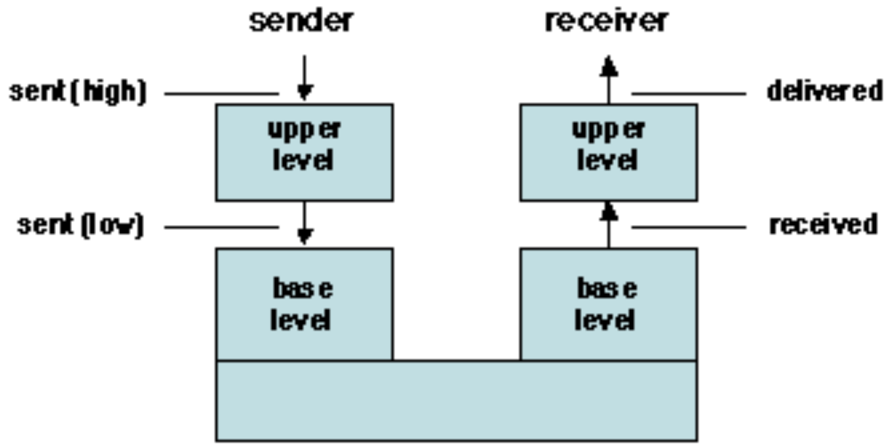


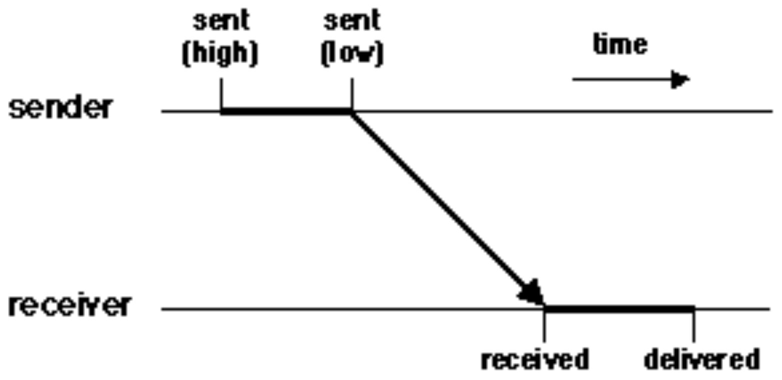
Parallel Communication


Serial Communication
Byte
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8
b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1
Byte
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8
Parallel Communication
Byte
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8
Byte
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8

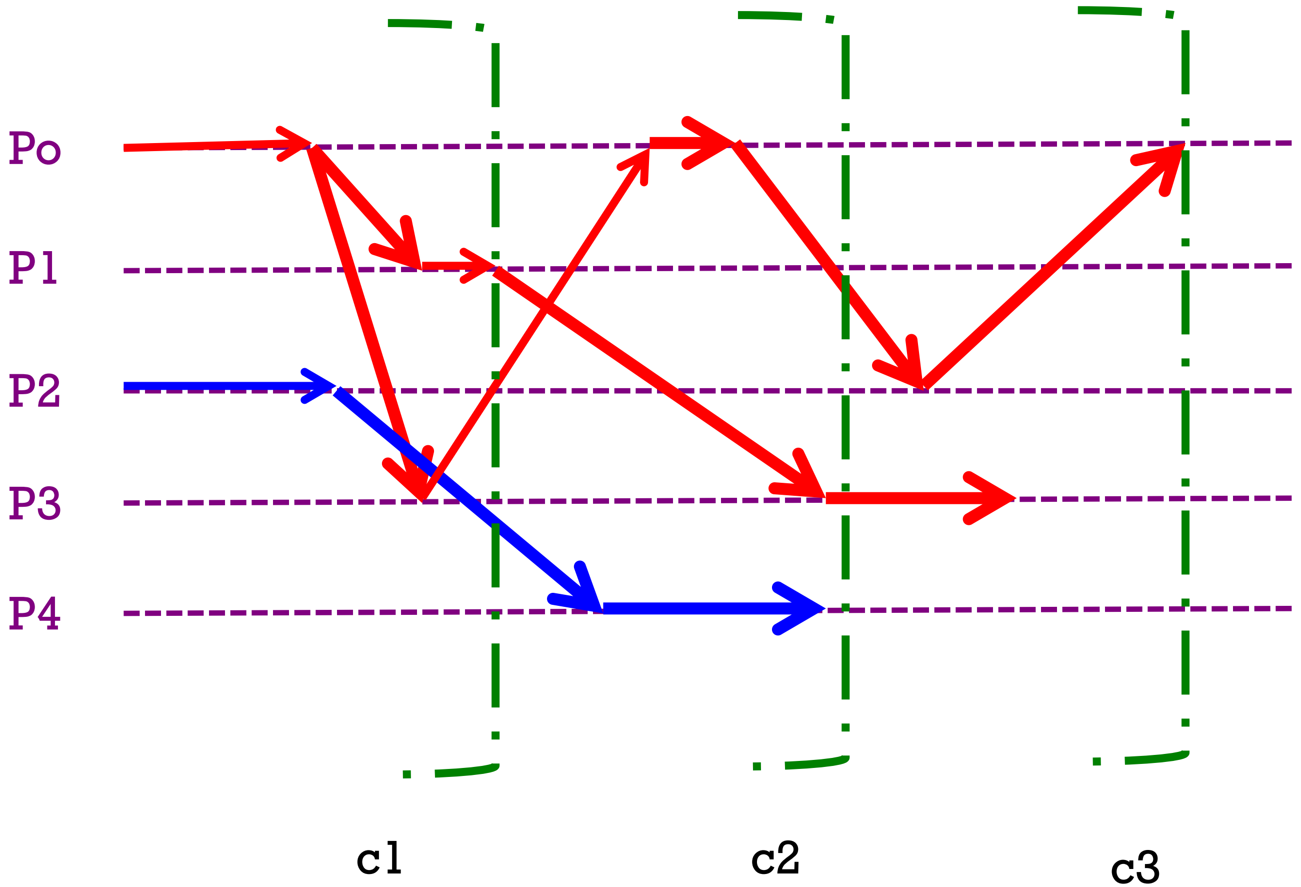
+

A General Communication – processing Model

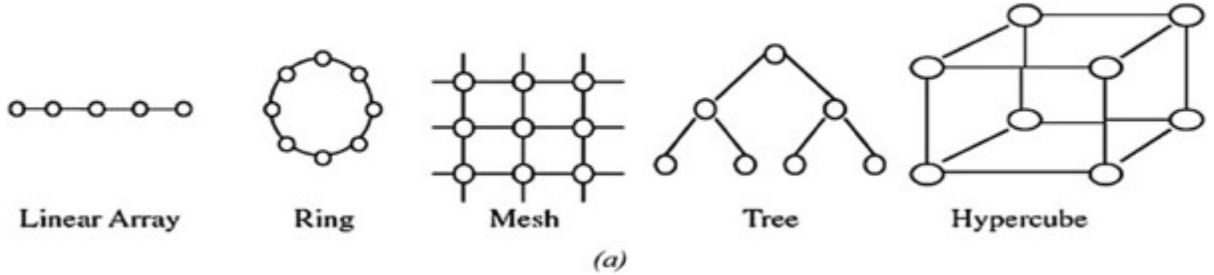


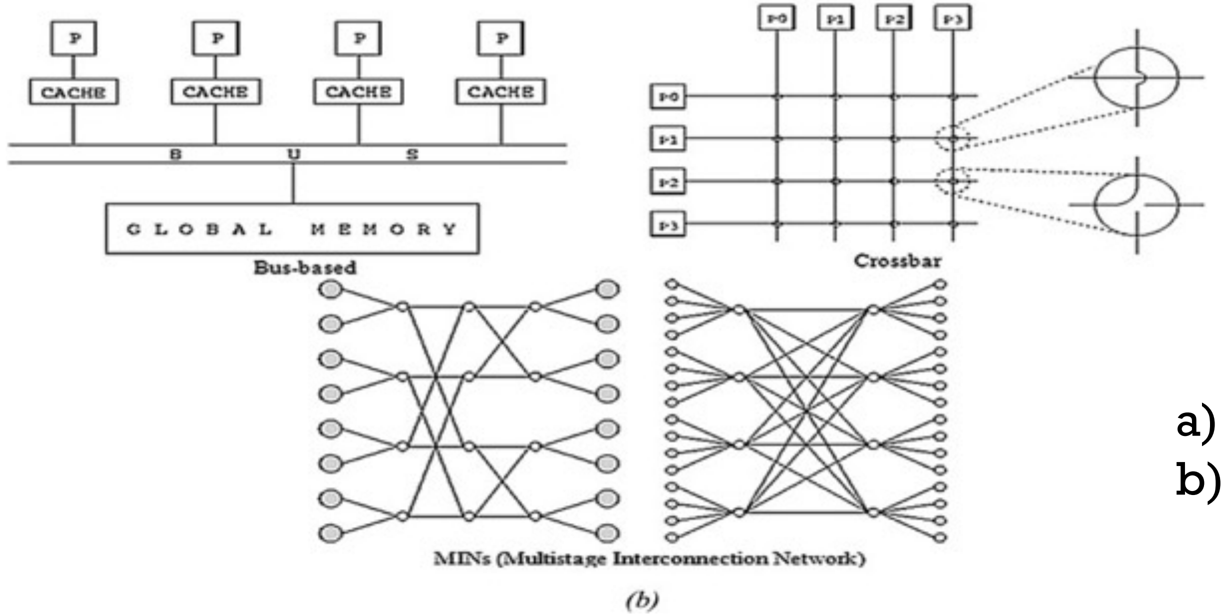


+ In Parallel…



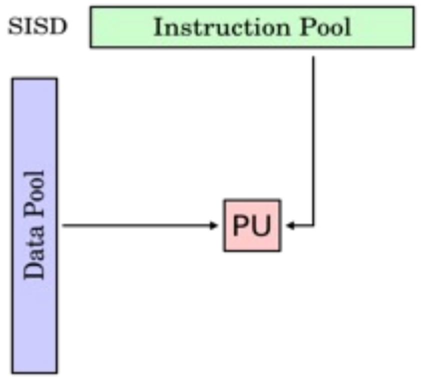
Interconnection Networks Topologies

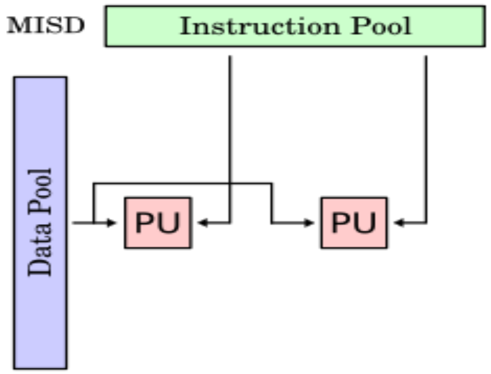


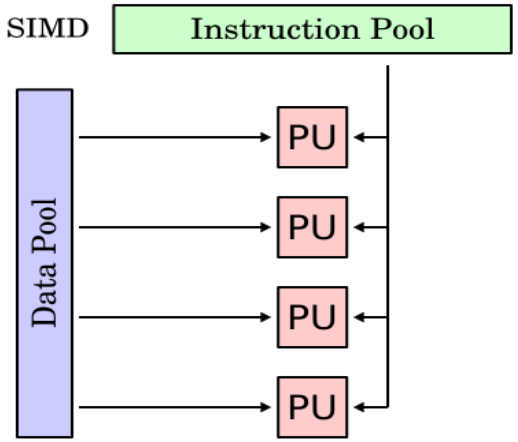


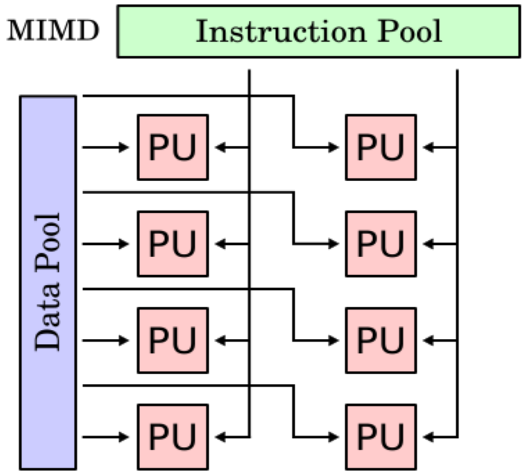
a) Static
b) Dynamic

+ ## Flynn's Taxonomy









* Proposed by M. Flynn in 1966

Some Glosary

- Any scalable system is a distributed system.
- Parallel computing uses Scalable Systems
  - Many instructions are carried out simultaneously-concurrently.
  - High Performance Computing (HPC) implies Parallel Computing
- Scalable Systems may be describe in terms of Scalable Architectures.
- Scalable Architectures (hardware point of view) have the characteristics of Scalable Systems.
  - Concurrency, Distribution
- Scalable Architectures support Parallel Computing.
- Of course, Parallel Computing implies parallelism.
- Obviously, Parallel Computing demads Parallel Machines.

+ Evolution of Configurable Architecture

Large Scale Cores
(High Single Thread Performance)

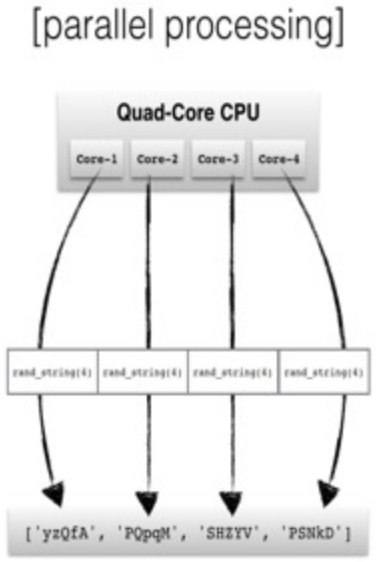
Dual Cores
(Symmetric Multithreading)

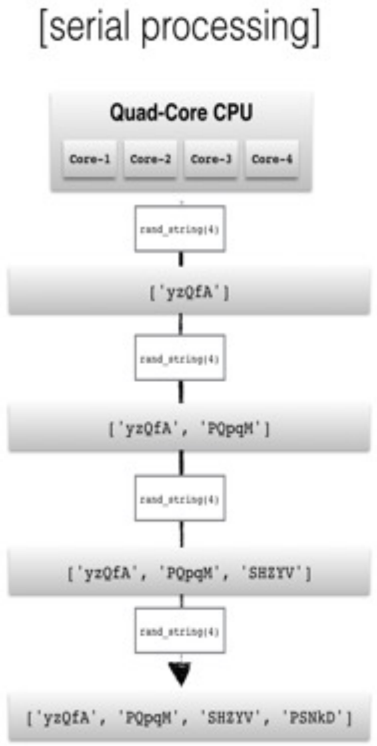
MultiCore Arrays

Scalar + Many Cores
(Highly threaded workloads)

Manycore arrays

+ MultiProcessing

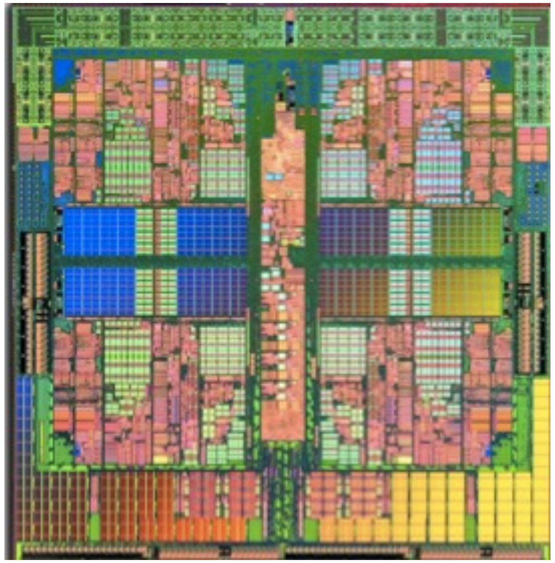




From https://sebastianraschka.com/Articles/2014_multiprocessing.html

- Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system.
- In a multiprocessing system, all CPUs may be equal, or some may be reserved for special purposes. A combination of hardware and operating system software design considerations determine the symmetry (or lack thereof) in a given system.
  - Symmetric multiprocessing (SMP).
  - Asymmetric multiprocessing (ASMP).
  - Uniform memory access (UMA) processing.
  - Non-uniform memory access (NUMA) processing.
  - Clustered multiprocessing.
- Tightly coupled multiprocessor system
  - CPUs may have access to a central shared memory (Muticore processors).
- Loosely coupled multiprocessor system
  - Based on multiple standalone single or dual processor commodity computers interconnected via a high speed communication system (As Beowulf Clusters)

+ Multi-core Processor


AMD QuadCore

- A multi-core processor is a single computing component with two or more independent processing units called cores, which read and execute program instructions.

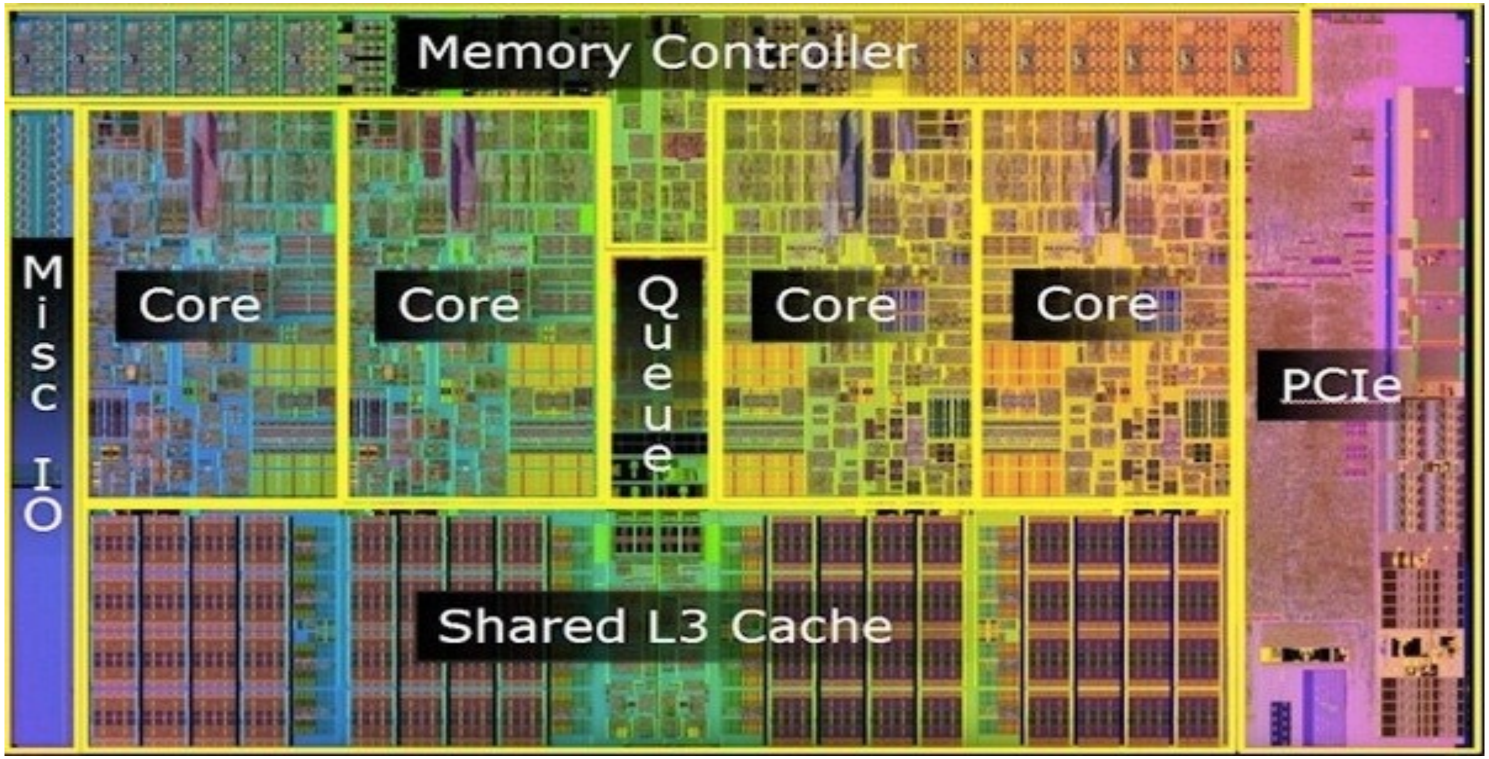
- The technical motivation behind multicore is based on the observation that, for a given semiconductor process technology, power dissipation of modern CPUs is proportional to the third power of clock frequency fc (actually it is linear in fc and quadratic in supply voltage Vcc , but a decrease in fc allows for a proportional decrease inVcc)

- A multi-core processor implements multiprocessing in a single physical package.

- The improvement in performance gained by the use of a multi-core processor depends very much on the software algorithms used and their implementation.

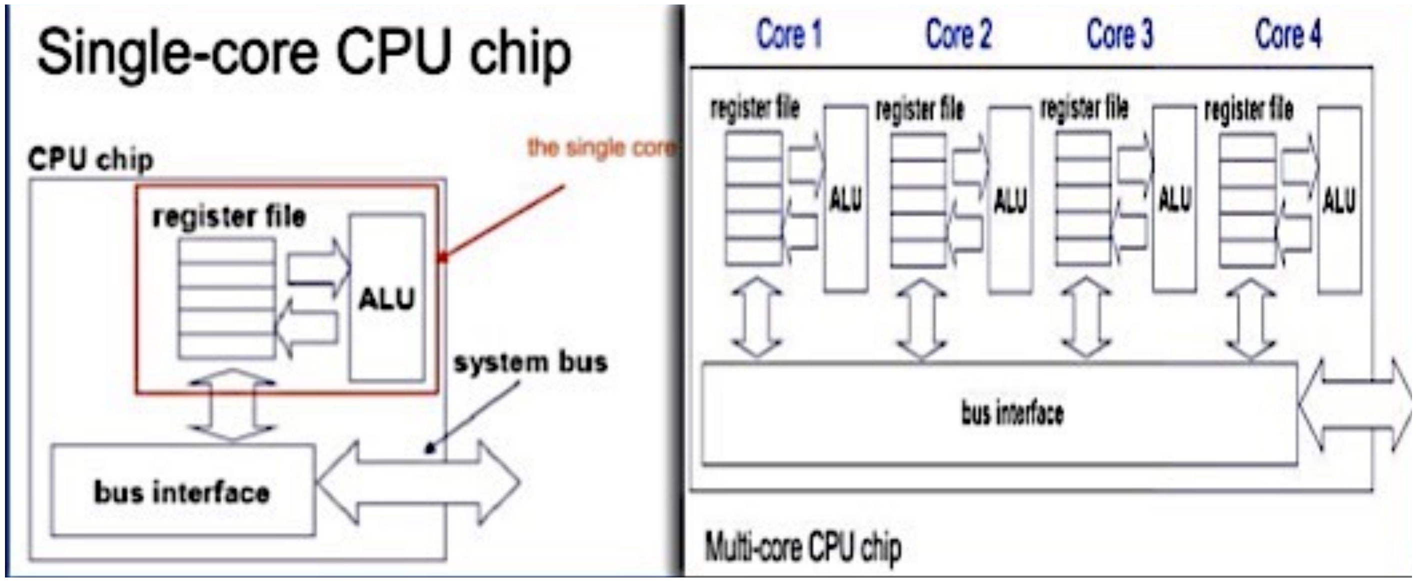
+

MultiCore Processor

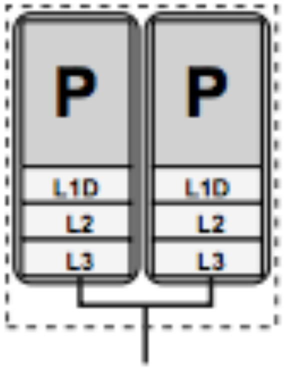


Intel QuickPath Architecture

+ Single-Core Vs MultiCore



+ MultiCore Processor Cache Groups



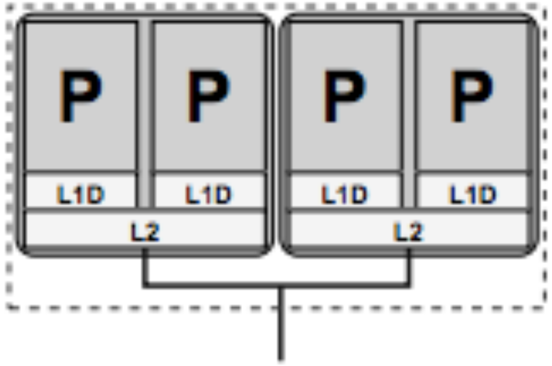
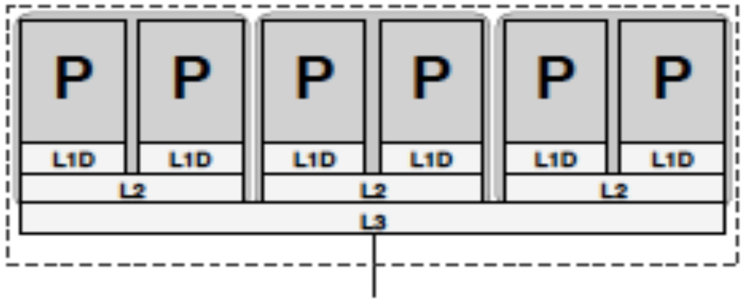


Figure 1.15: Dual-core processor chip with separate L1, L2, and L3 caches (Intel "Montecito"). Each core constitutes its own cache group on all levels.

Figure 1.16: Quad-core processor chip, consisting of two dual-cores. Each dual-core has shared L2 and separate L1 caches (Intel "Harpertown"). There are two dual-core L2 groups.

From : "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein,

MultiCore Processor Memory



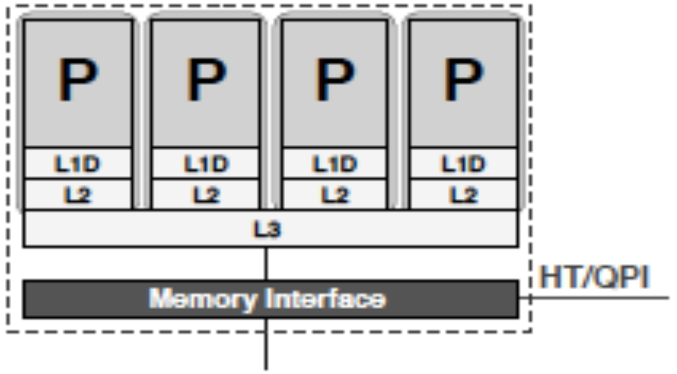
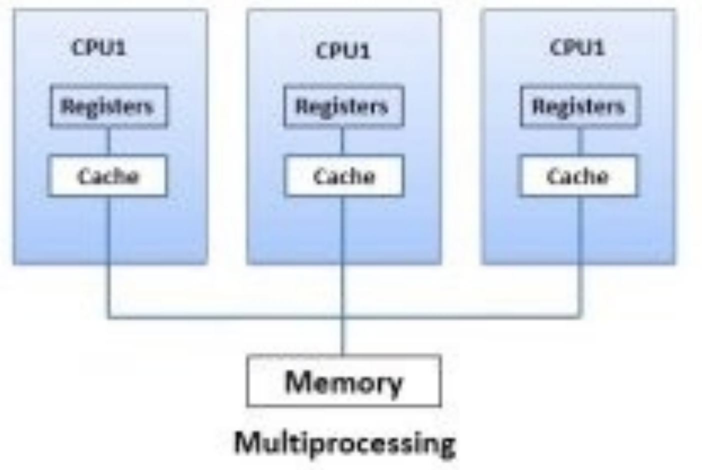


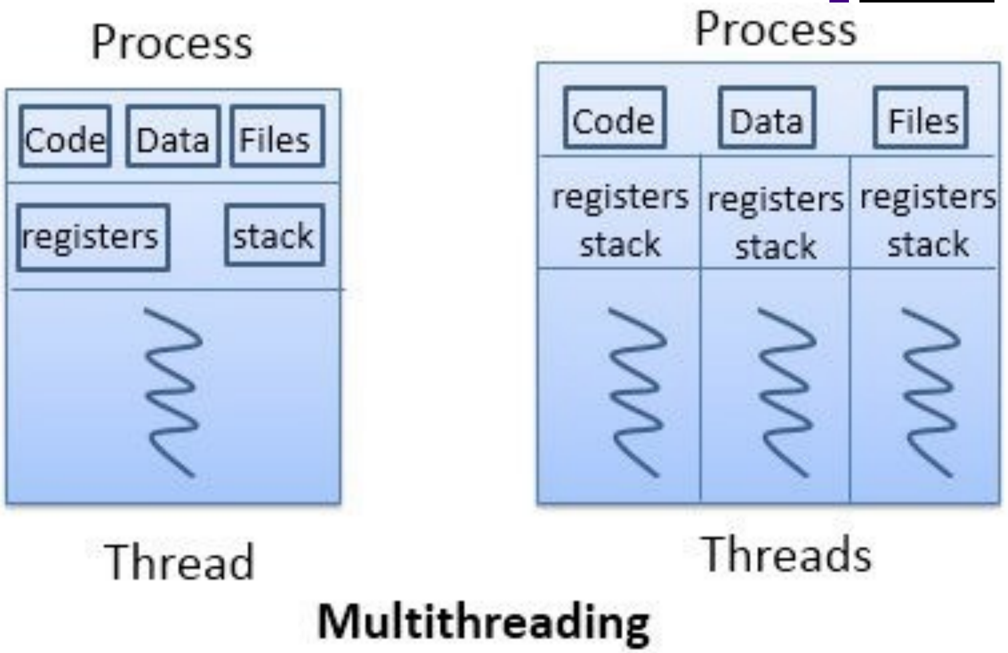
Figure 1.17: Hexa-core processor chip with separate L1 caches, shared L2 caches for pairs of cores and a shared L3 cache for all cores (Intel "Dunnington"). L2 groups are dual-cores, and the L3 group is the whole chip.

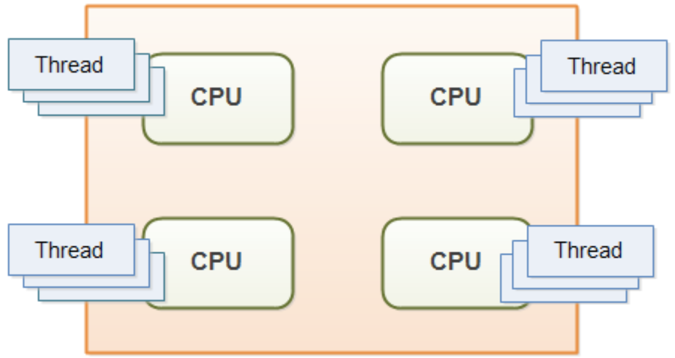
Figure 1.18: Quad-core processor chip with separate L1 and L2 and a shared L3 cache (AMD "Shanghai" and Intel "Nehalem"). There are four single-core L2 groups, and the L3 group is the whole chip. A built-in memory interface allows to attach memory and other sockets directly without a chipset.

From : "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein,

A Simple view of Multithreading (Before a Formal







Multithread Processors

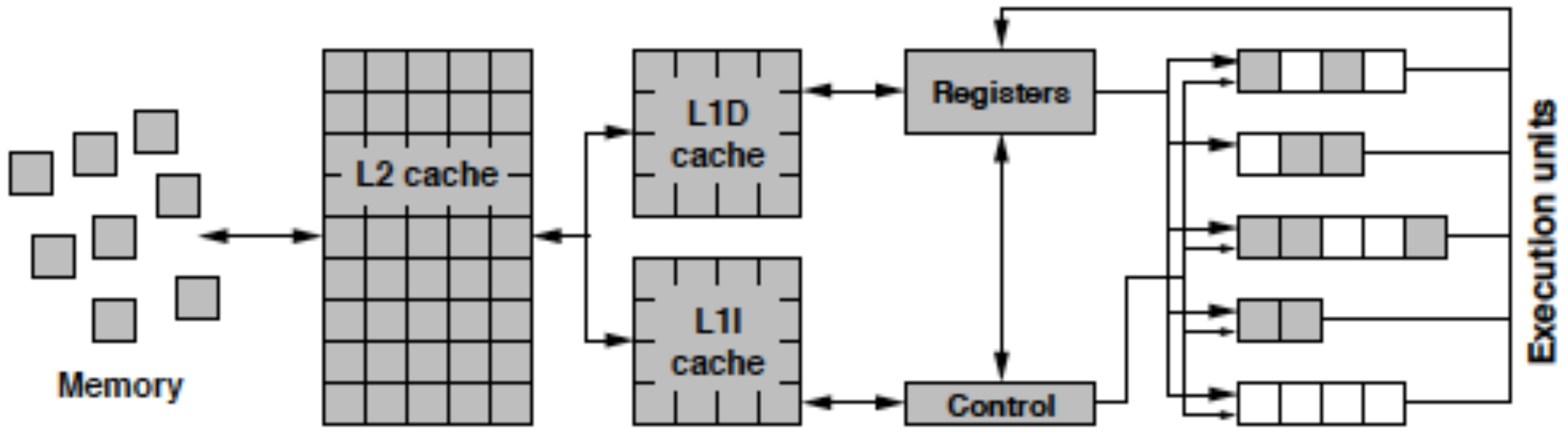


Figure 1.19: Simplified diagram of control/data flow in a (multi-)pipelined microprocessor without SMT. White boxes in the execution units denote pipeline bubbles (stall cycles). Graphics by courtesy of Intel.

From : "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein,

Vector Processors

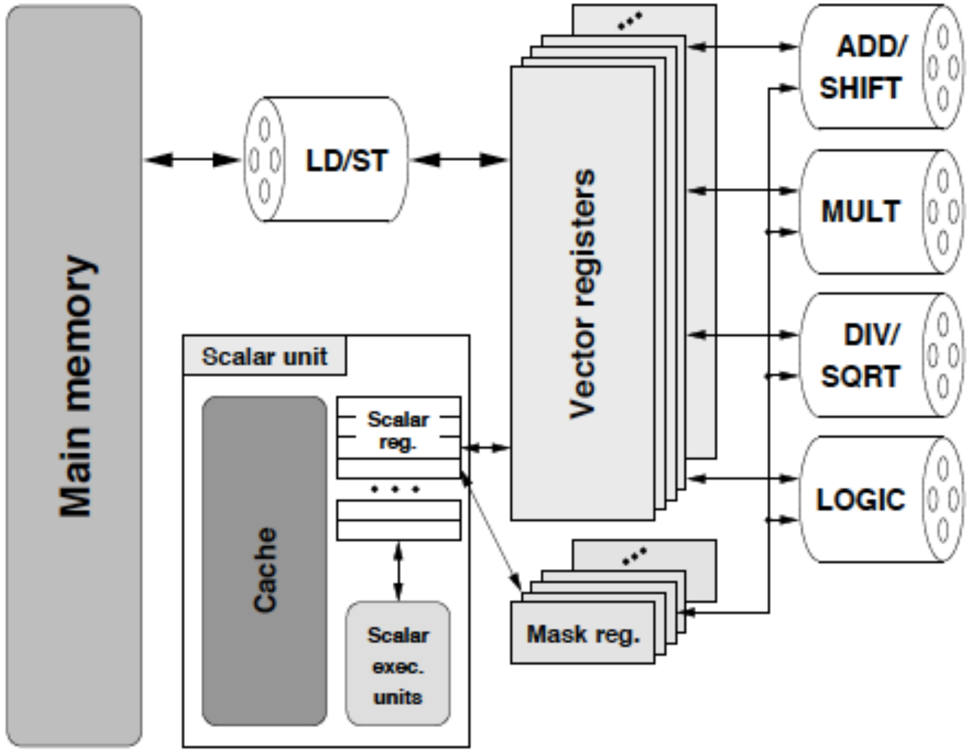
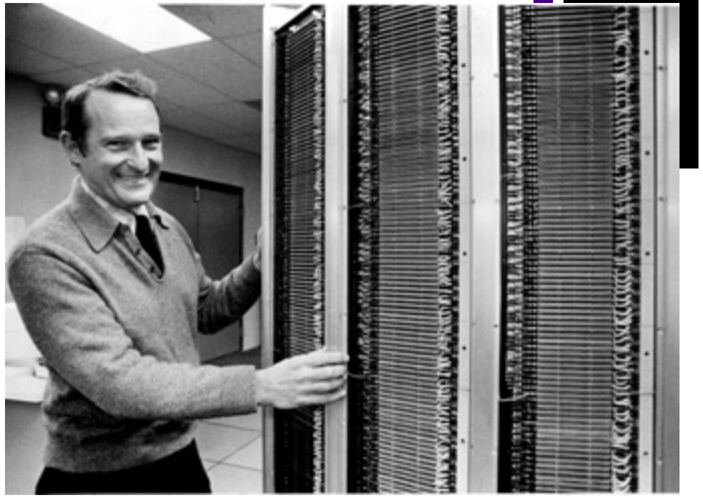


Figure 1.21: Block diagram of a prototypical vector processor with 4-track pipelines.



Seymour Cray

From : "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein,

Programming Vector Architectures

A vector CPU can issue a single instruction for a whole array if it is shorter than the vector length:

```
vload V1(1:N) = B(1:N)
vload V2(1:N) = C(1:N)
vadd V3(1:N) = V1(1:N) + V2(1:N)
vstore A(1:N) = V3(1:N)
```

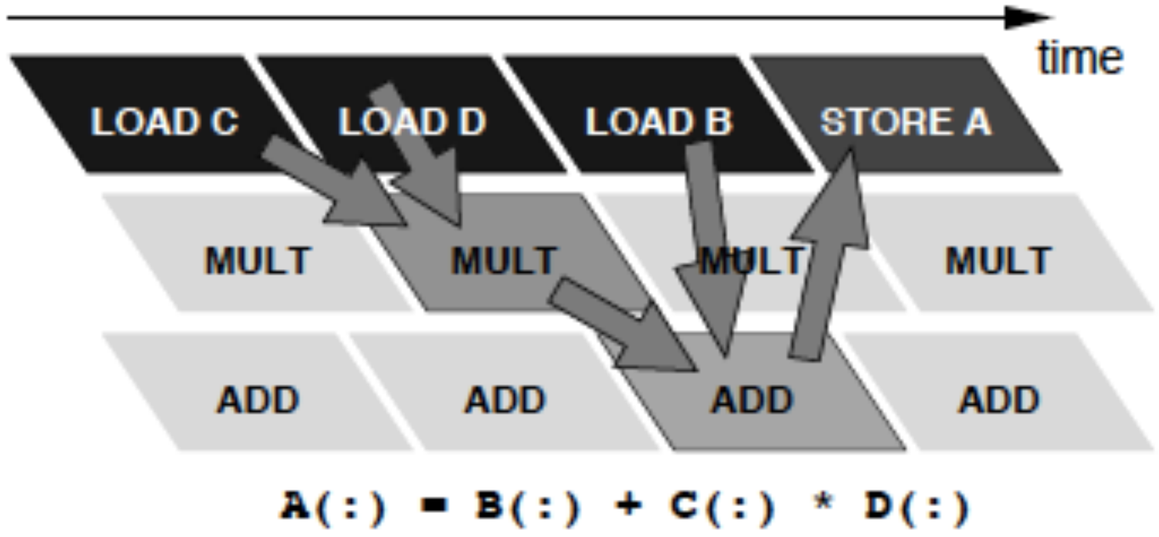
Here, V1, V2, and V3 denote vector registers. The distribution of vector indices across the pipeline tracks is automatic. If the array length is larger than the vector length, the loop must be stripmined, i.e., the original arrays are traversed in chunks of the vector

```
do S = 1,N,Lv
  E = min(N,S+Lv-1)
  L = E-S+1
  vload V1(1:L) = B(S:E)
  vload V2(1:L) = C(S:E)
  vadd V3(1:L) = V1(1:L) + V2(1:L)
  vstore A(S:E) = V3(1:L)
enddo
```

This is done automatically by the compiler.

From : "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein,

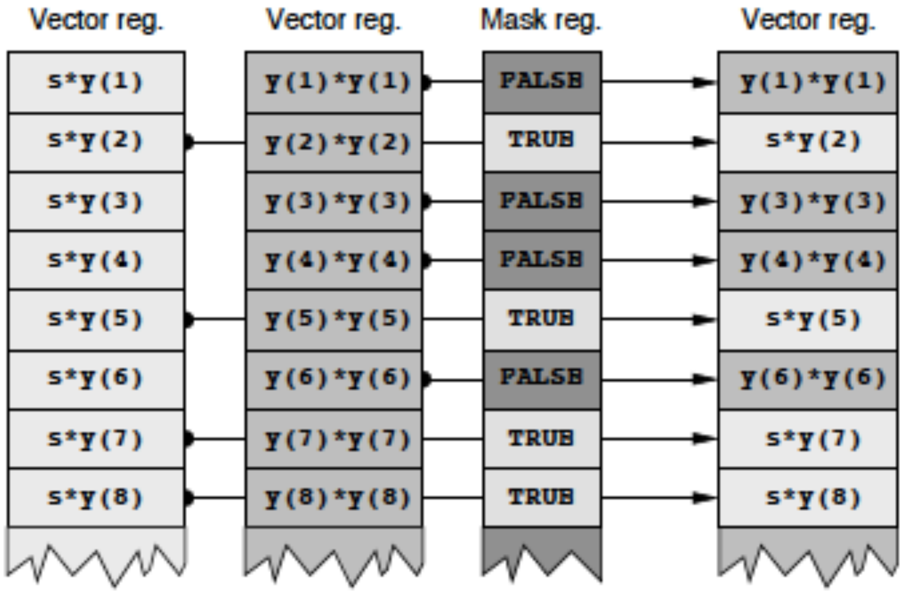
Pipeline Utilization Timeline



Pipeline utilization timeline for execution of the vector Triad on the vector processor shown in Figure 1.21. Light gray boxes denote unused arithmetic units.

From : "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein,

Mask Registers and Vectorization



Loop with if/else branch can be vectorized using Mask Registers:

```
do i = 1,N
  if(y(i) .le. 0.d0) then
    x(i) = s * y(i)
  else
    x(i) = y(i) * y(i)
  endif
enddo
```

Conditional false:

```
do i = 1,N
  if(y(i) .ge. 0.d0) then
    x(i) = sqrt(y(i))
  endif
enddo
```

From : "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein,

Vectorization with Gather/Scatter Method

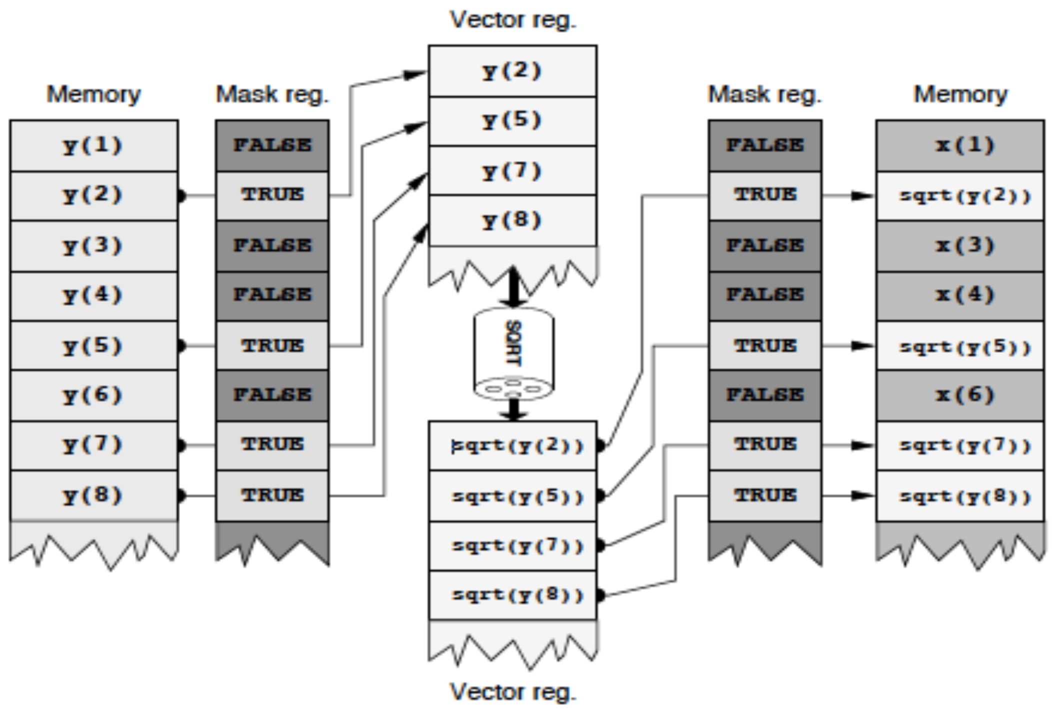
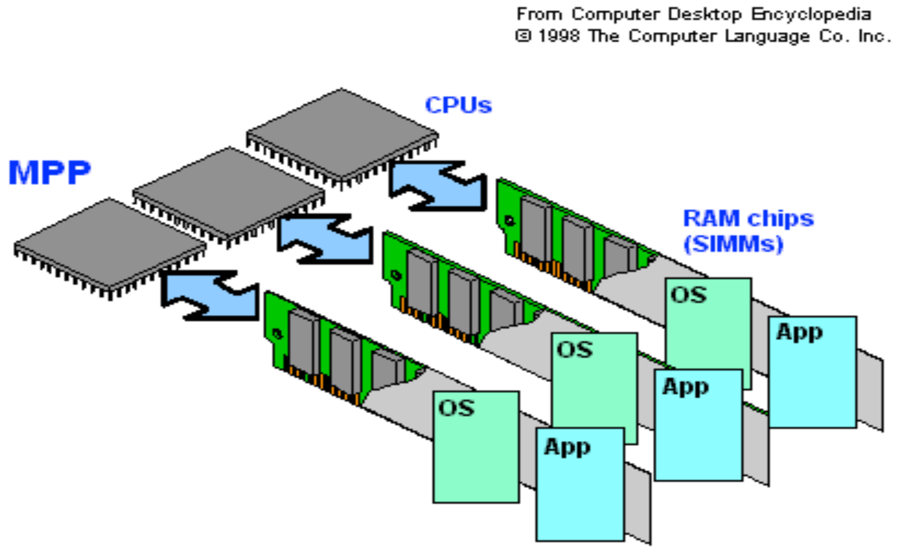


Figure 1.24: Vectorization by the gather/scatter method. Data transfer from/to main memory occurs only for those elements whose corresponding mask entry is true. The same mask is used for loading and storing data.

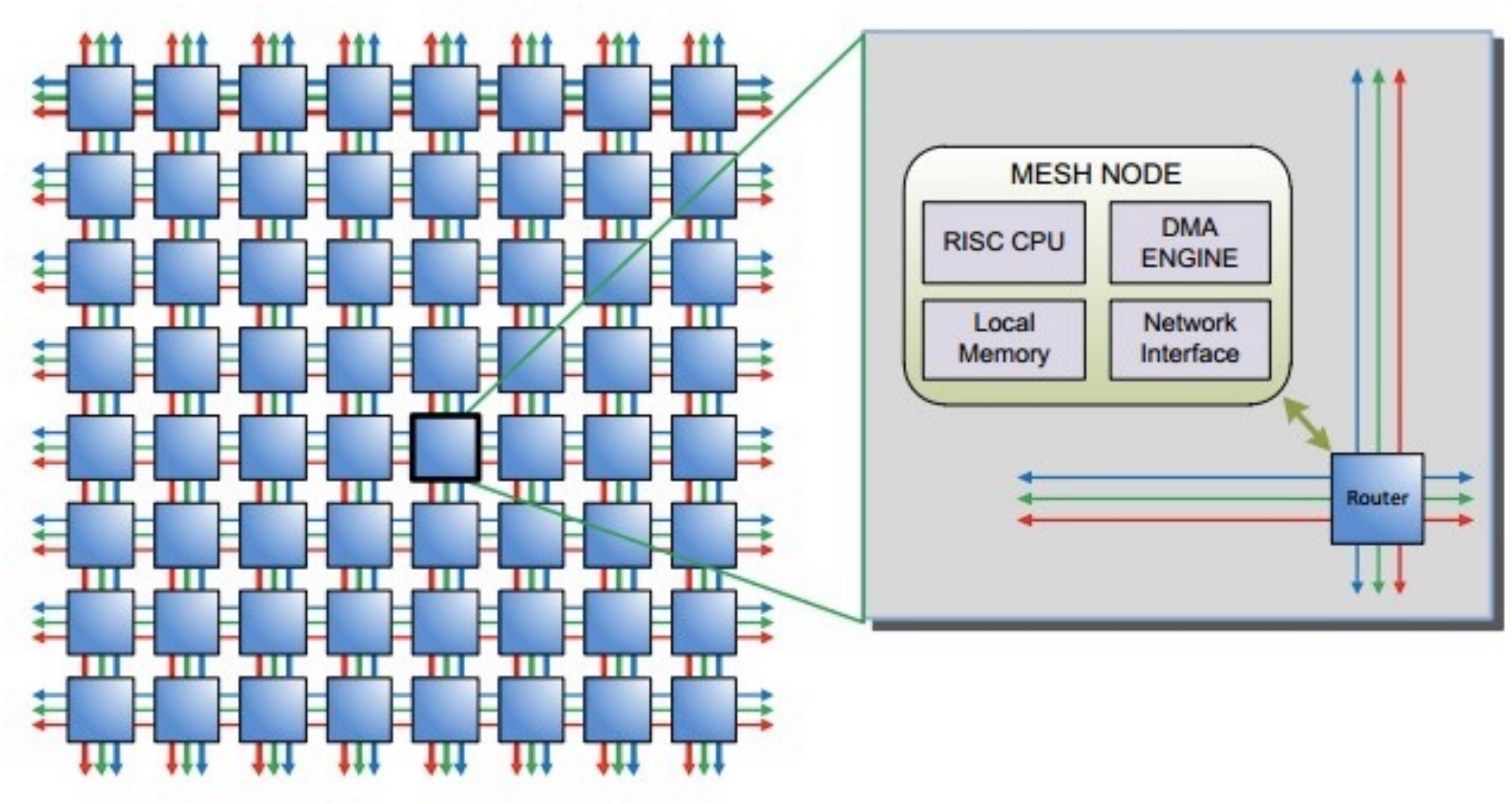
From : "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein,

Massive Parallel Processing (MPP)

- Computer system with many independent arithmetic units or entire microprocessors, that run in parallel
- MPPA is a MIMD (Multiple Instruction streams, Multiple Data) architecture, with distributed memory accessed locally, not shared globally


From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.
CPUs
MPP
RAM chips (SIMMs)
OS
App
OS
App
OS
App

Parallel Massive Processors (Manycores)

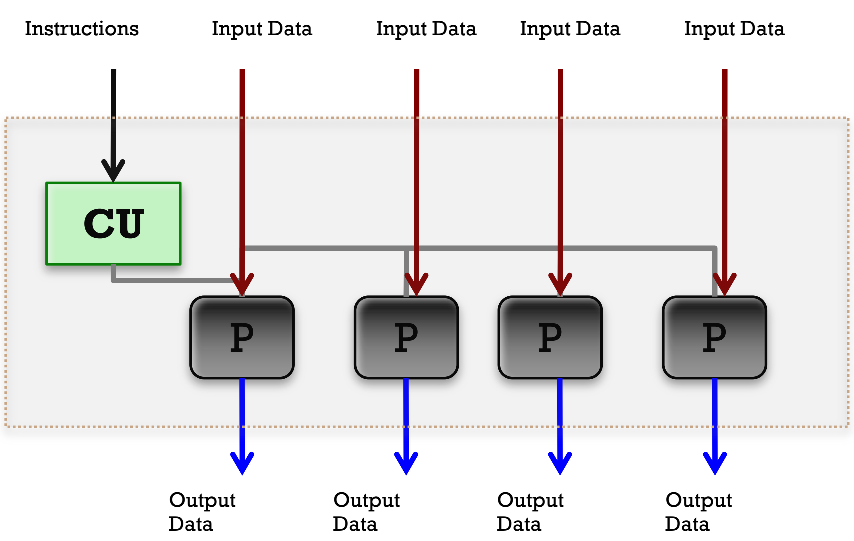


From https://www.parallella.org/

Remember Architectural Flynn's Taxonomy

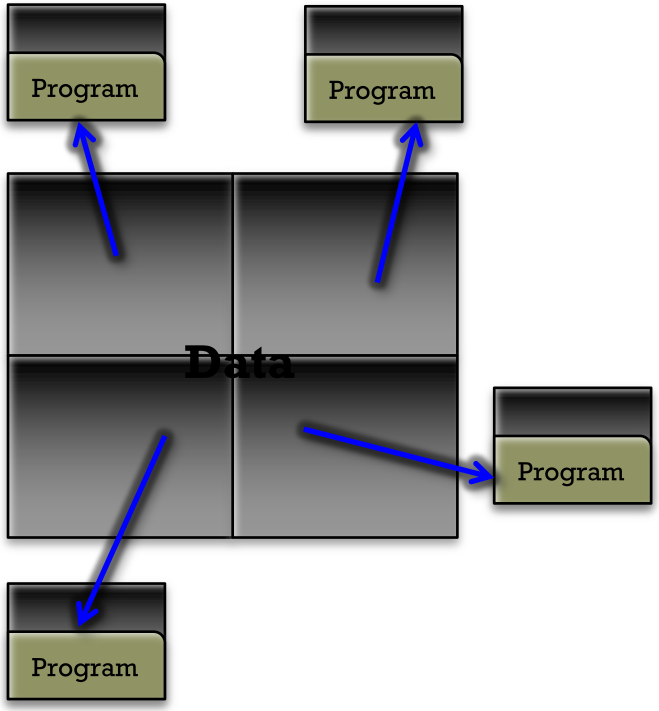
SPMD: Parallel Processing Units execute the same program on multiple parts of data

SIMD: All processors units are executing the same instructions in any instant.



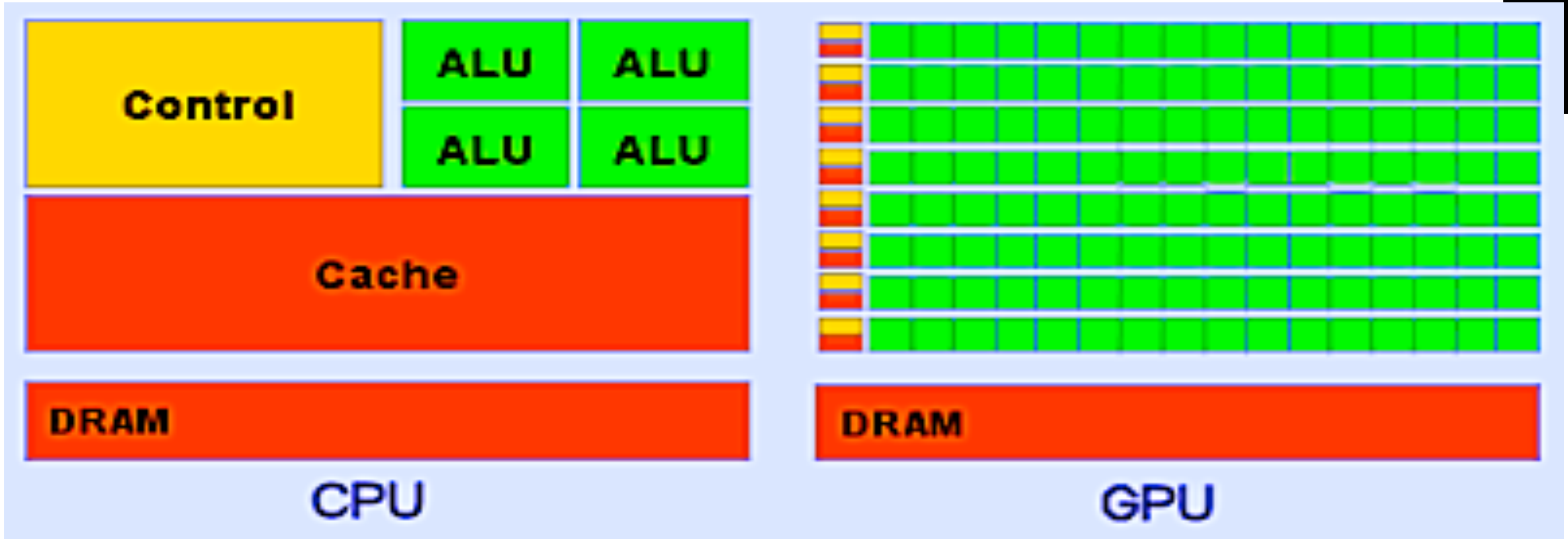
SIMD

+



Processor

Comparing CPU (Multicore) and GPU (Manycore/massive Multiprocessing)



- Manycore processors are distinct from multi-core processors in that they are optimised from the outset for a higher degree of explicit parallelism, and for higher throughput (or lower power consumption) at the expense of latency and lower single thread performance.
- Manycore processors are specialist multi-core processors designed for a high degree of parallel processing, containing a large number of simpler, independent processor cores.
- Manycore processors are used extensively in embedded computers and high-performance computing.

Parallel Massive Processors Motivation

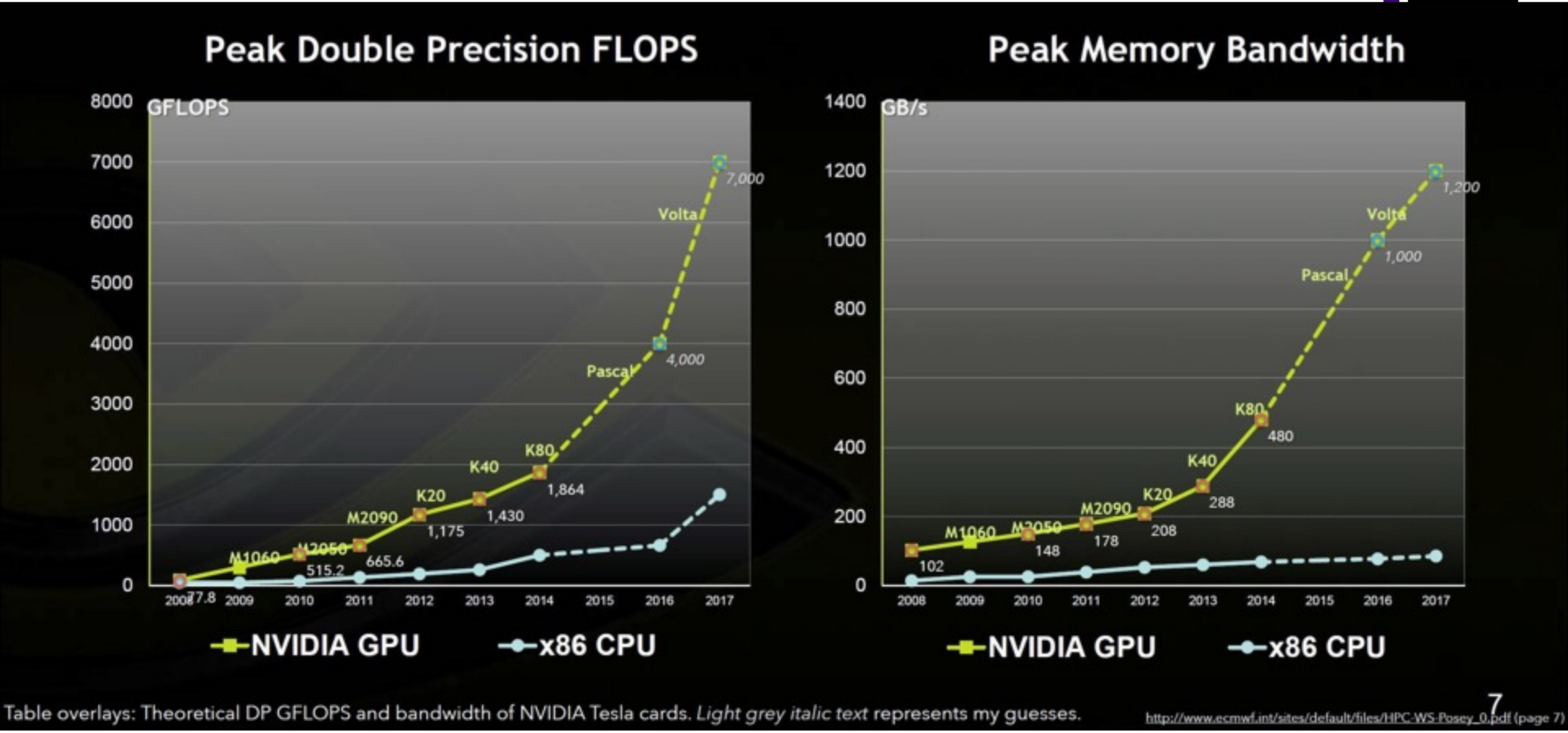


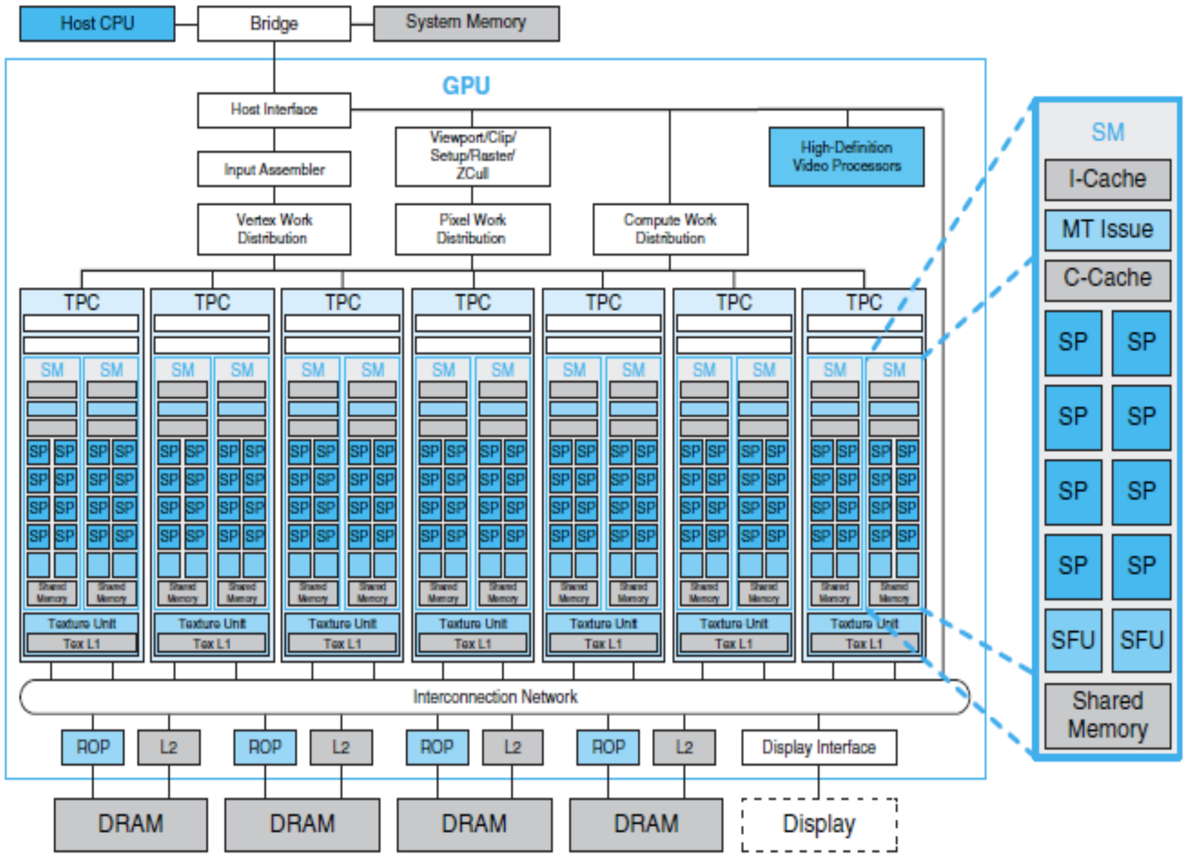
Table overlays: Theoretical DP GFLOPS and bandwidth of NVIDIA Tesla cards. *Light grey italic text* represents my guesses.

http://www.ecmwf.int/sites/default/files/HPC-WS-Posey_0.pdf (page 7)

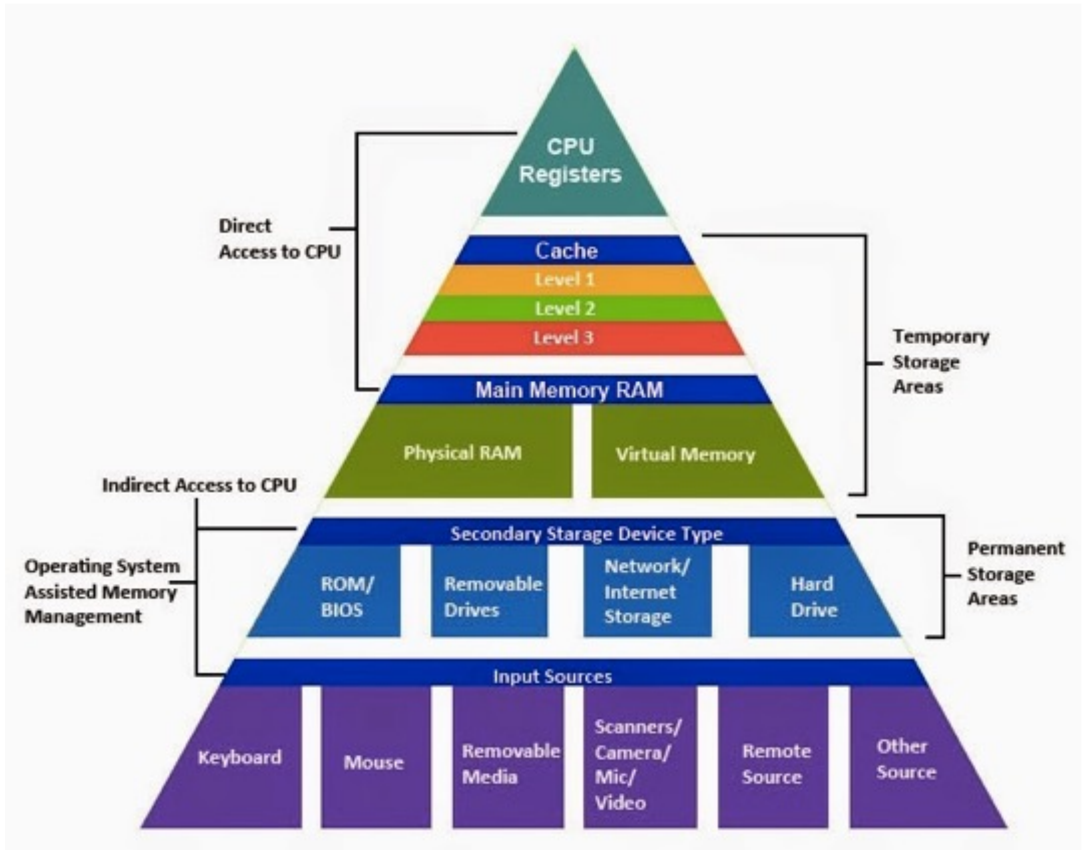
From https://www.nvidia.com

+

NVIDIA TESLA® Architectur



Memory Hierarchy (Quickly View)



- Programming Model and Paradigms observe Hierarchy of Memory
- Memory Hierarchy is an active problem in research and development.
  - Model costs of processing and memory use.
  - Energy Models.
  - Model costs of both interprocessor communication and memory hierarchy traffic

Cache Mapping (Direct Mapped)

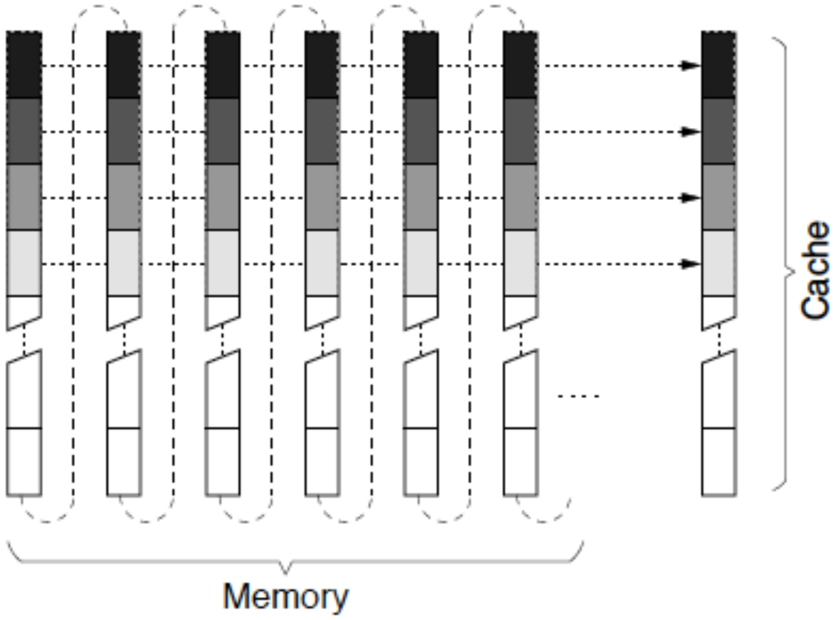


Figure 1.10: In a direct-mapped cache, memory locations which lie a multiple of the cache size apart are mapped to the same cache line (shaded boxes).

```
do i=1, N, CACHE_SIZE_IN_BYTES/8
          A(i)= B(i)+C(i)*D(i)
enddo
```

From : "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein,

Cache Mapping (Different Cache Ways)

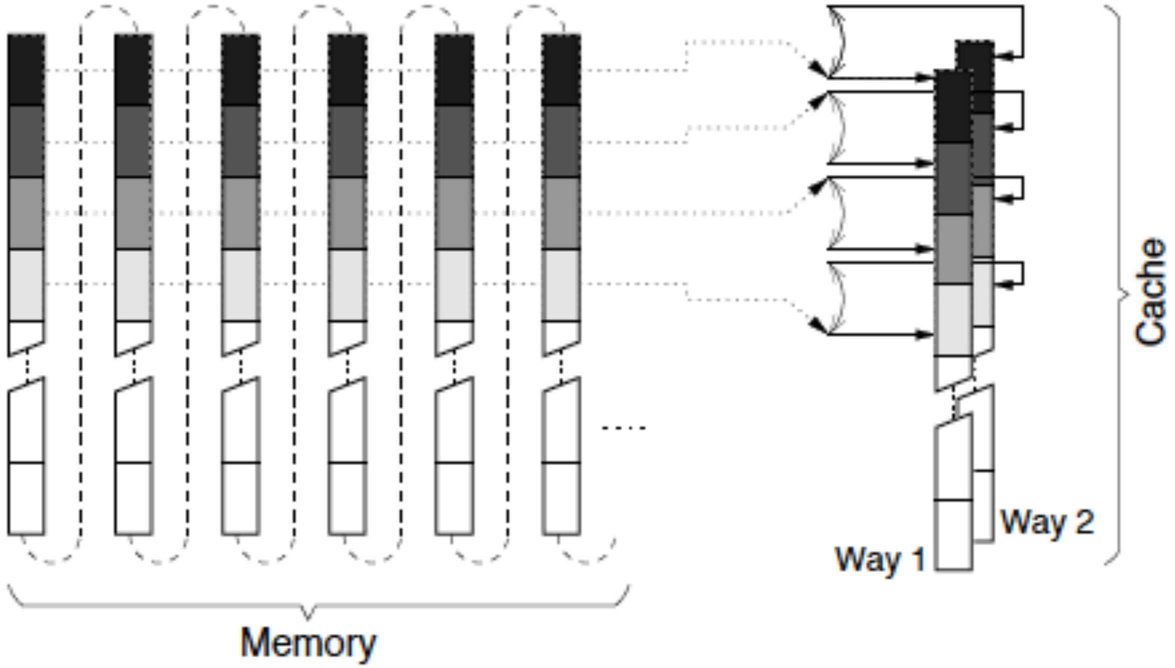


Figure 1.11: In an m-way set-associative cache, memory locations which are located a multiple of $\frac{1}{m}$th of the cache size apart can be mapped to either of m cache lines (here shown for $m = 2$).

From : "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein,

Latency Penalties in Memory

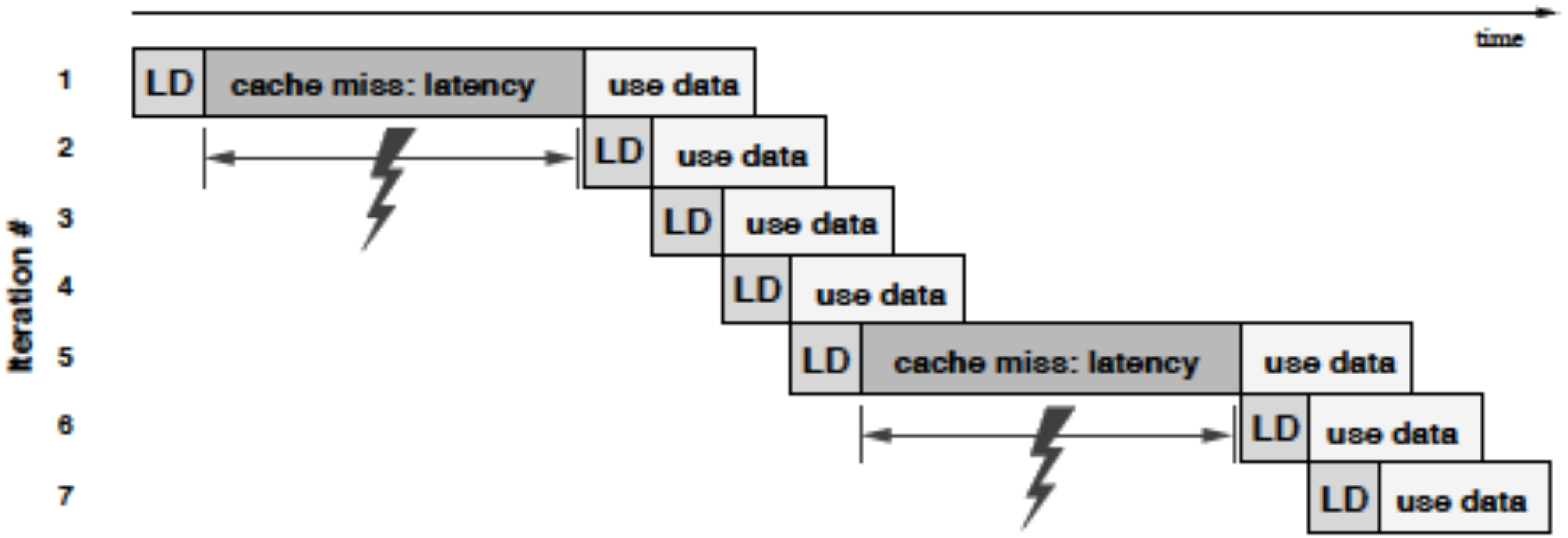


Figure 1.12: Timing diagram on the influence of cache misses and subsequent latency penalties for a vector norm loop. The penalty occurs on each new miss.

From : "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein,

Prefetch

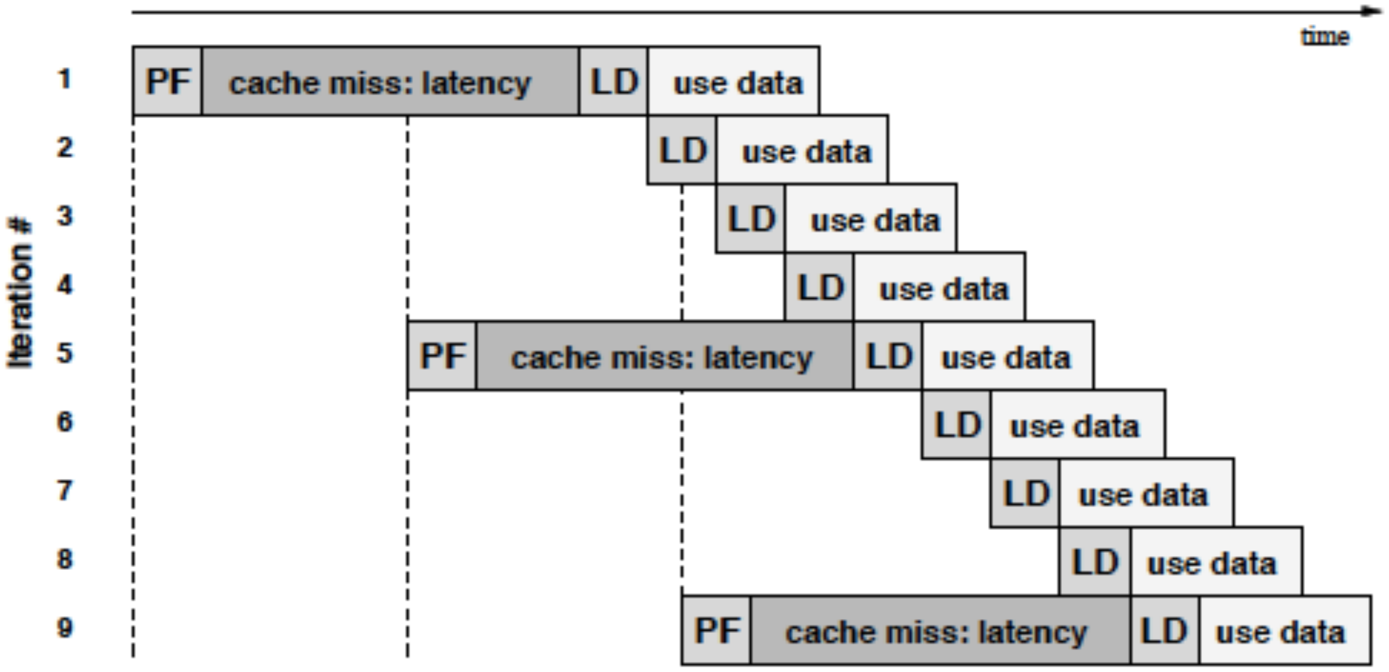
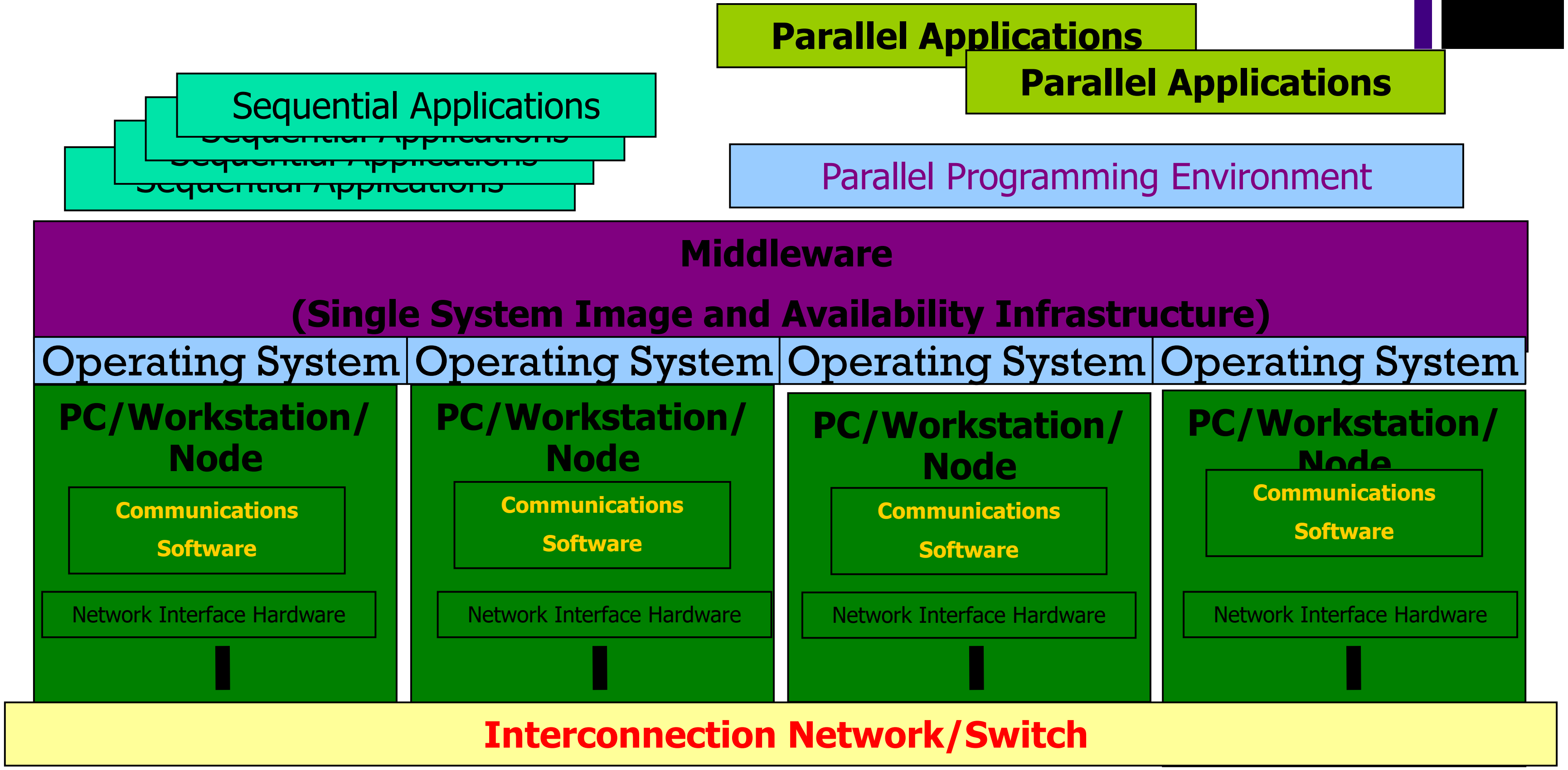


Figure 1.13: Computation and data transfer can be overlapped much better with prefetching. In this example, two outstanding prefetches are required to hide latency completely.

From : "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein,

+ Cluster Computing Architecture



GUANE-1

- Supercomputing Platform based on GPGPU 128 TESLA FERMI NVIDIA GPUs
- Peak 205 Tflops (Double)
- 16 Nodes (Hybrid Platform)
- Since 2012 (Obsolete but it still the most powerful machine in Colombia)
- … however, it exits some different HPC and HPC@Pocket platforms…

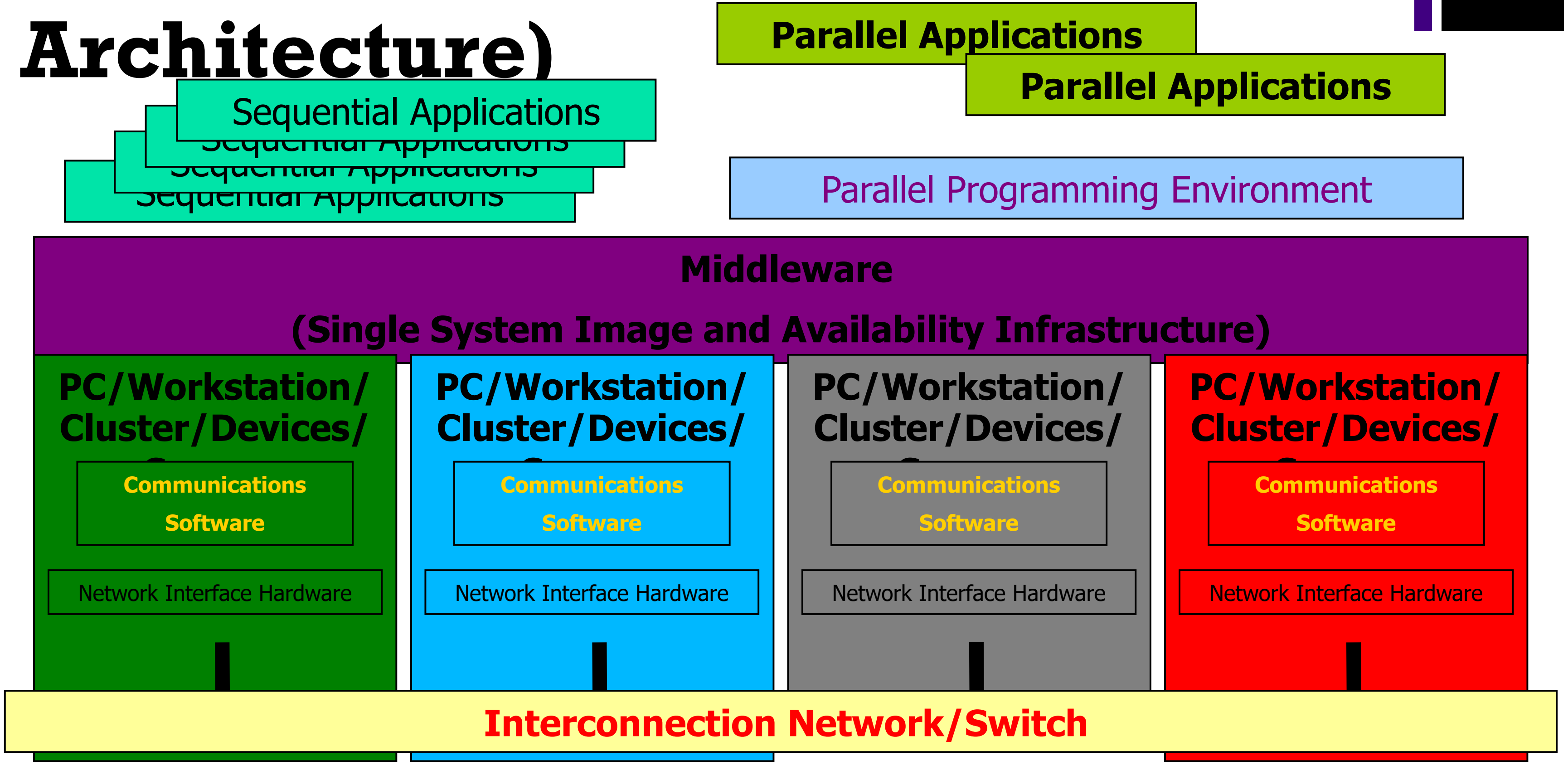
+

HPC Hybrid Systems (HPC@Pocket)

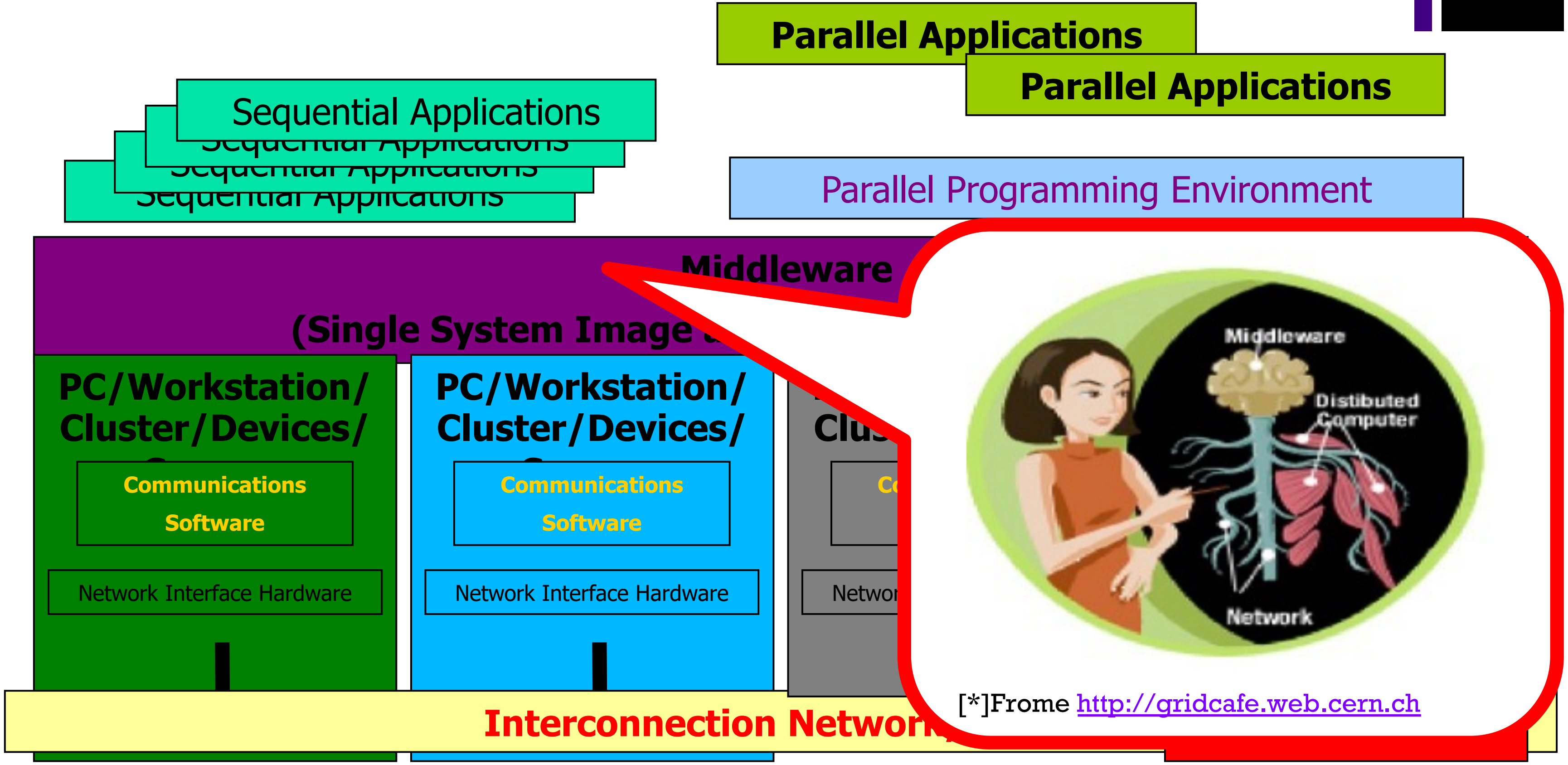
- High Performance Capabilities
  - Multiple Cores (i.e. more than 192 cores in Jetson)
- Co-Design Architecture
  - Allowing multiple networks and protocols
  - Software Implementation Mechanisms (Now, very known, i.e. CUDA, OpenCL… same Python)
  - Low Power
- Low Cost
  - Depending of the device… (≈1 € per core)
- However, Integration/interaction demands efficiency

nVidia® Jetson TK1/TX1

Grid Computing Architecture (Remember the Cluster Architecture)


Parallel Applications
Parallel Applications
Sequential Applications
Sequential Applications
Sequential Applications
Sequential Applications
Parallel Programming Environment
Middleware
(Single System Image and Availability Infrastructure)
PC/Workstation/ Cluster/Devices/
Communications Software
Network Interface Hardware
PC/Workstation/ Cluster/Devices/
Communications Software
Network Interface Hardware
PC/Workstation/ Cluster/Devices/
Communications Software
Network Interface Hardware
PC/Workstation/ Cluster/Devices/
Communications Software
Network Interface Hardware
Interconnection Network/Switch

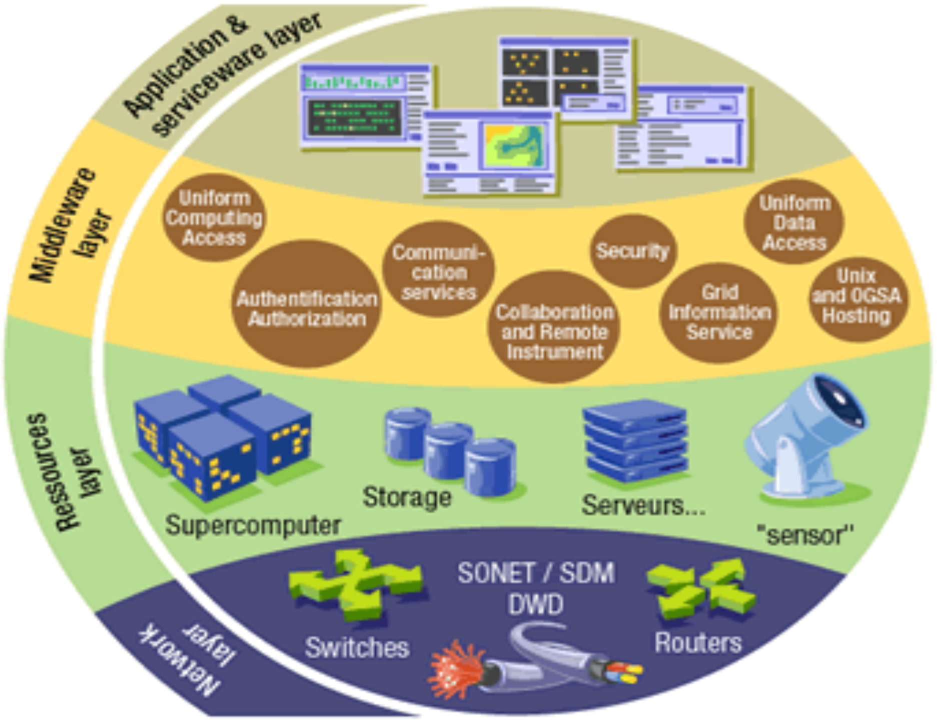
Grid Computing Architecture and the Middleware

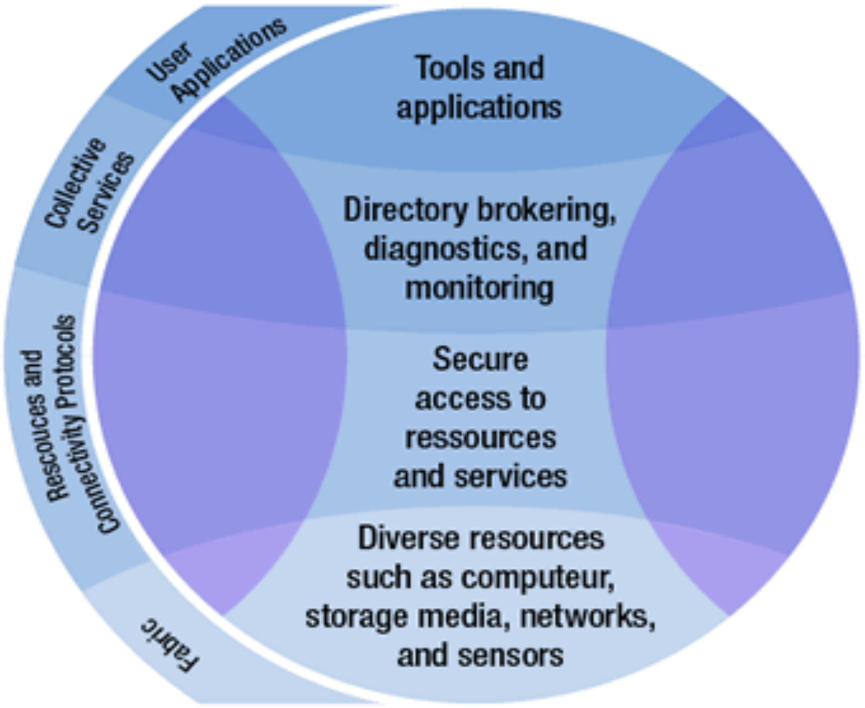


[*]Frome http://gridcafe.web.cern.ch

+

Grid Computing Architecture (Typical Diagram)

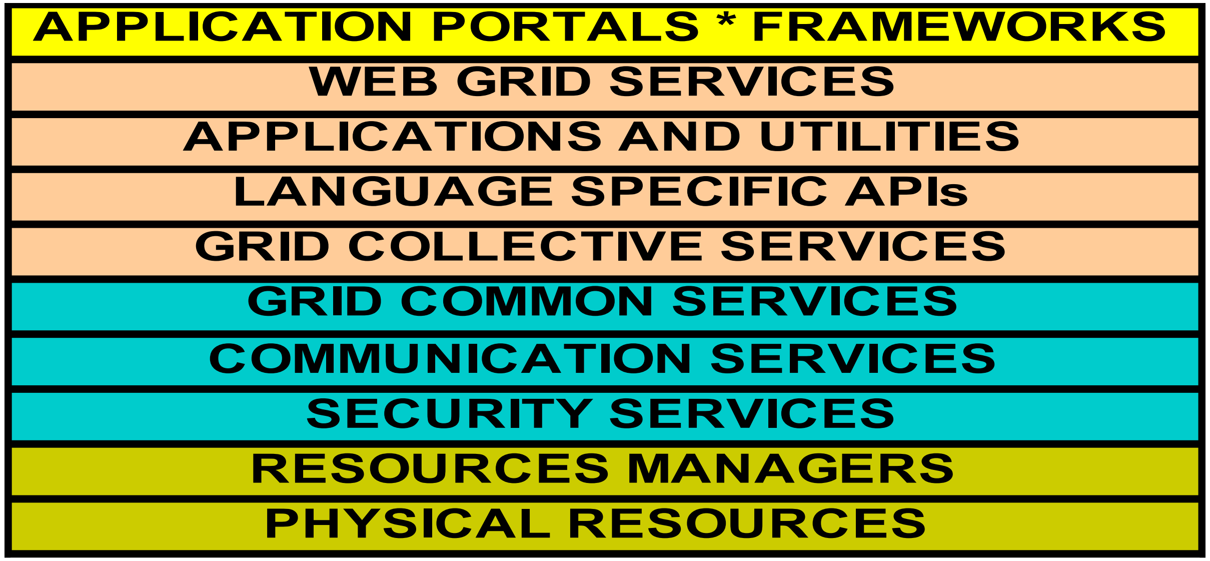




User Applications
Tools and applications
Collective Services
Directory brokering, diagnostics, and monitoring
Rescouces and Connectivity Protocols
Secure access to ressources and services
Fabric
Diverse resources such as computeur, storage media, networks, and sensors

[*]From http://gridcafe.web.cern.ch

Grid Computing Architecture Views

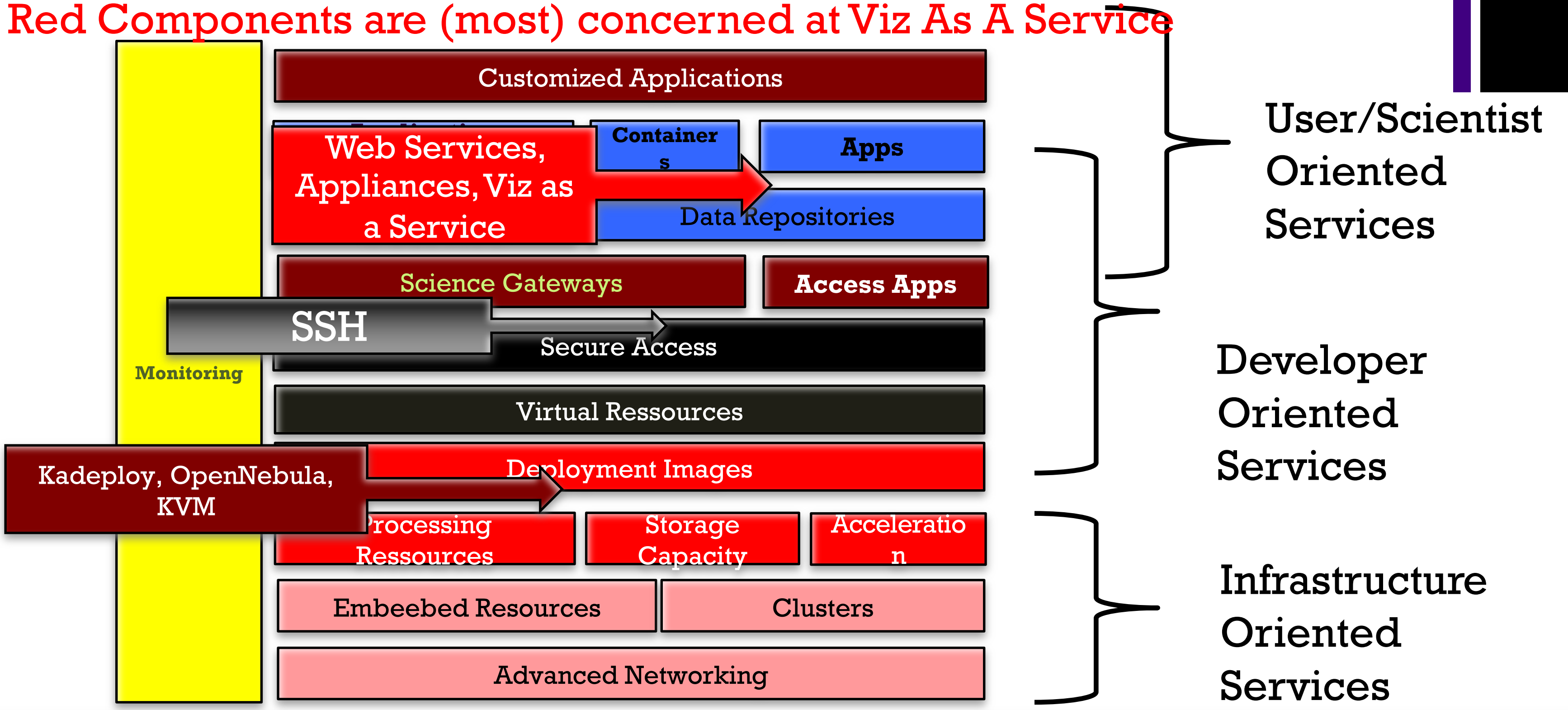


[*]From: Grid Computing: Making The Global Infrastructure a Reality

+

HPC as A Service Model

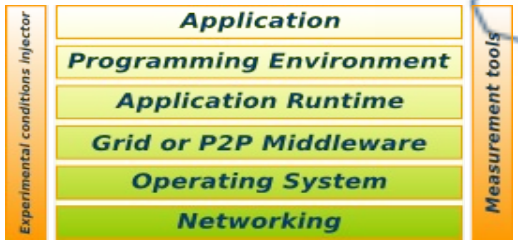
* Red Components are (most) concerned at Viz As A Service



+

A Grid Exemple: Grid5000 (G5K)

- G5K has 5000 processors distributed in 11 sites France wide, for research in Grid Computing, eScience and Cyber-infrastructures
- G5K project aims at building a highly reconfigurable, controlable and monitorable experimental Grid platform



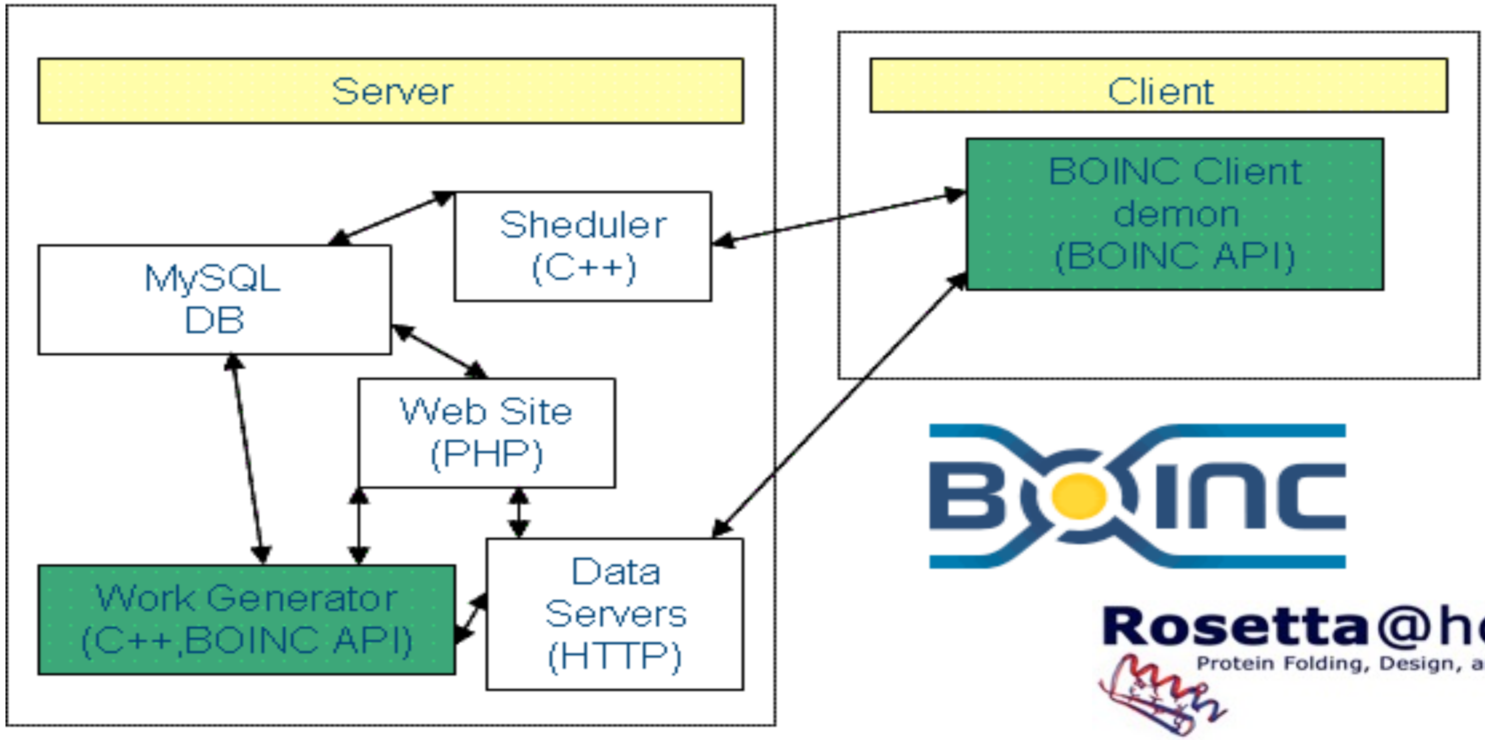
www.grid5000.org

Volunteer Computing

- Volunteer computing is a type of distributed computing in which computer owners donate their computing resources (such as processing power and storage) to one or more "projec
    - •BOINC (Seti@home)
    - •Xgrid
    - •GridMP
- Associated with P2P
- Can be associated with High Throughput Computing (HTC) or High Performance Computing (HCP)

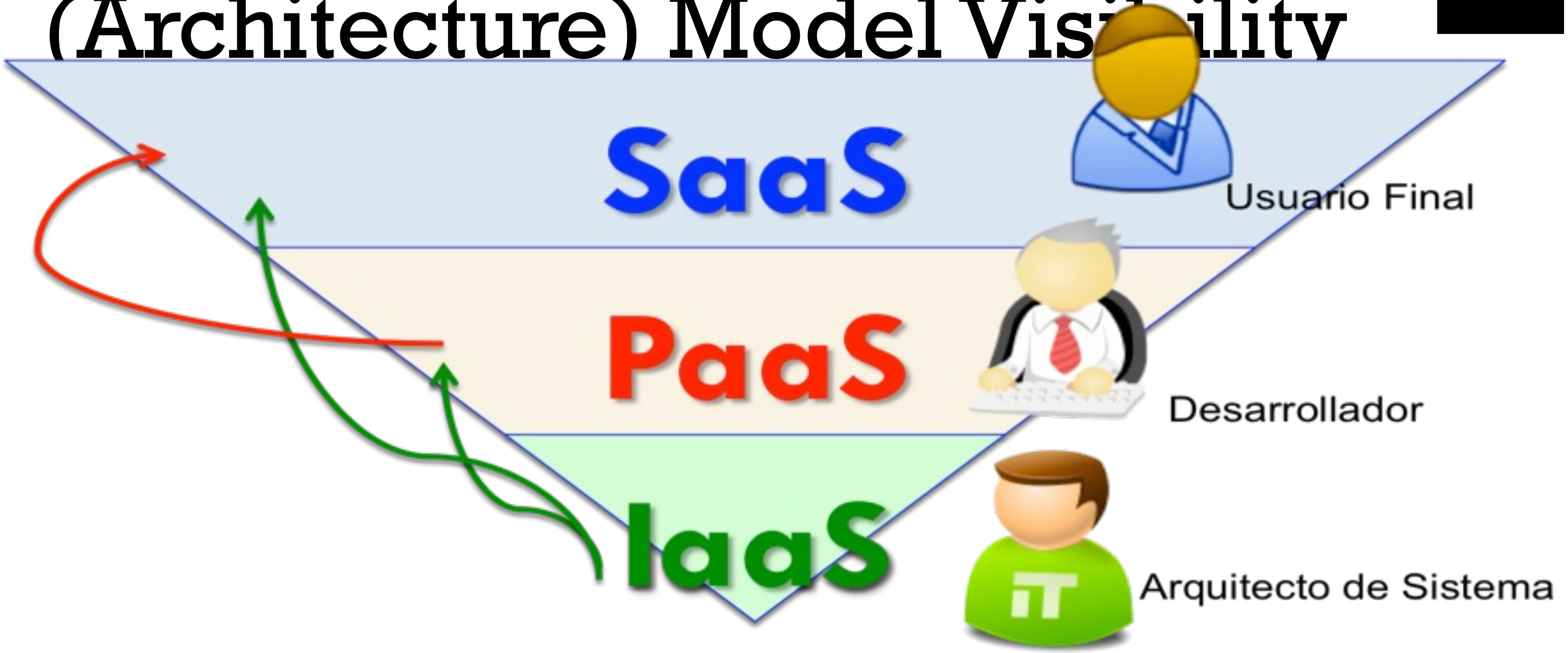
+

BOINC Architecture



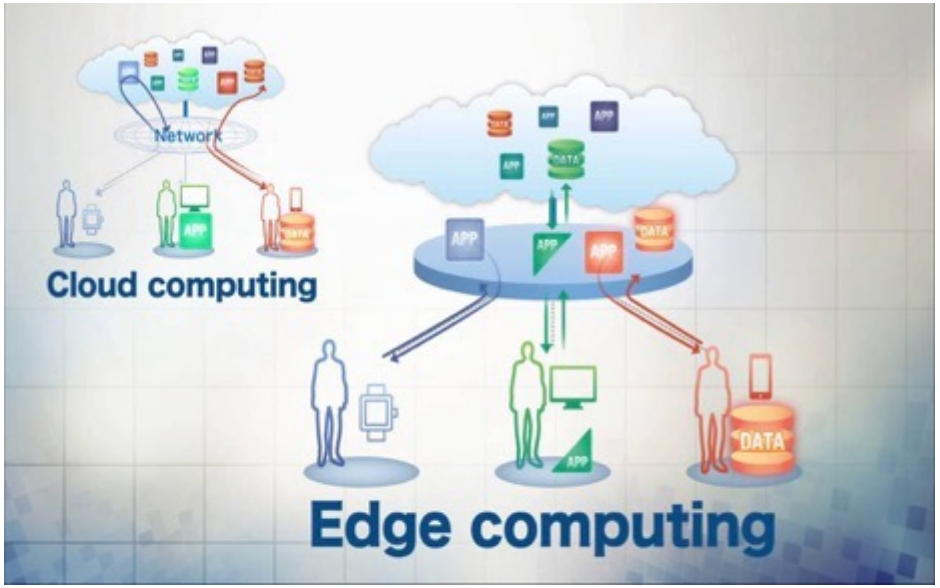
+

Cloud Computing (Architecture) Model Visi ility

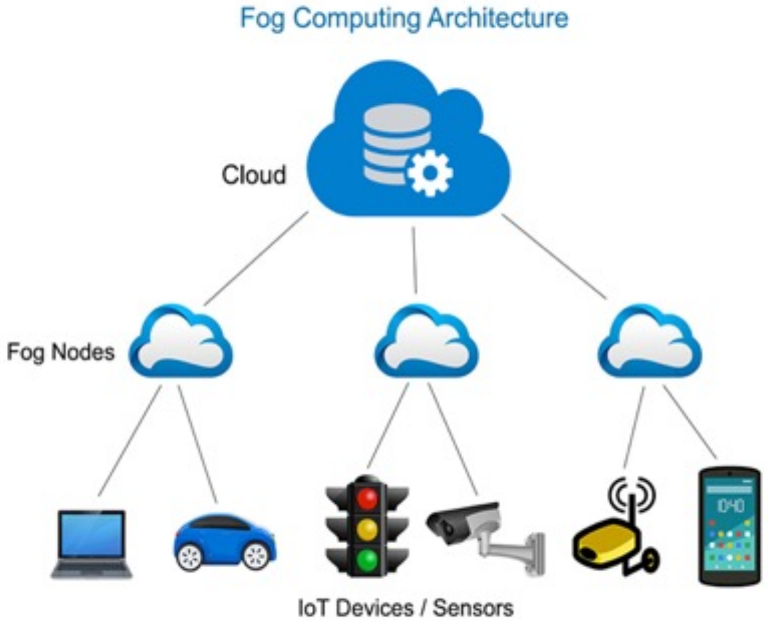


+

UltraScale Systems



Mesh network of micro data centers that process or store
ocally

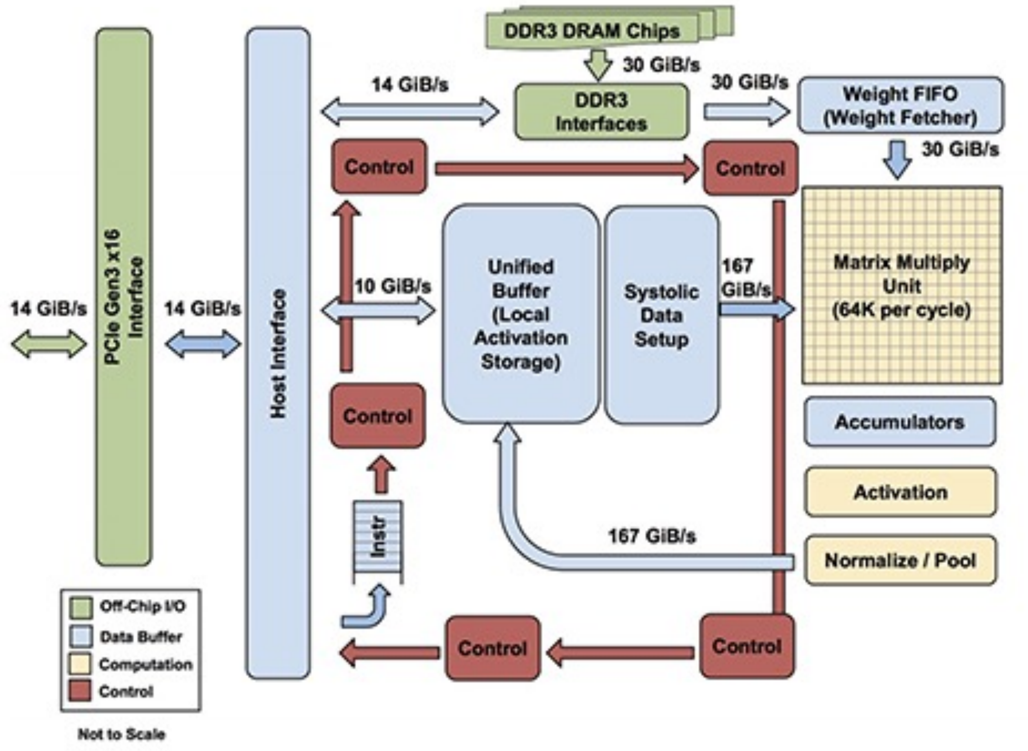


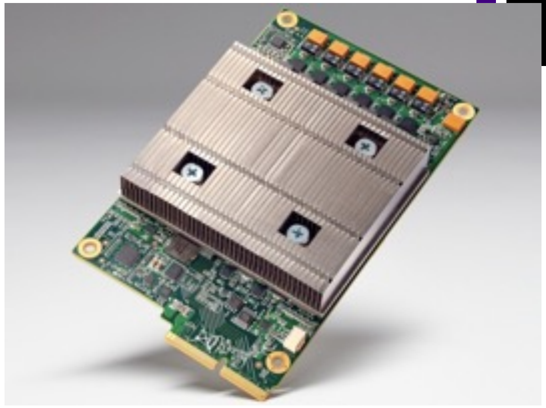
Extends Cloud computing and services to the edge of the

NESUS COST IC1305 Network for Sustainable Ultrascale Computing

« Ultrascale systems are envisioned as large-scale complex systems joining parallel and distributed computing systems that will be two to three orders of magnitude larger that today's systems » (Carretero et al.)

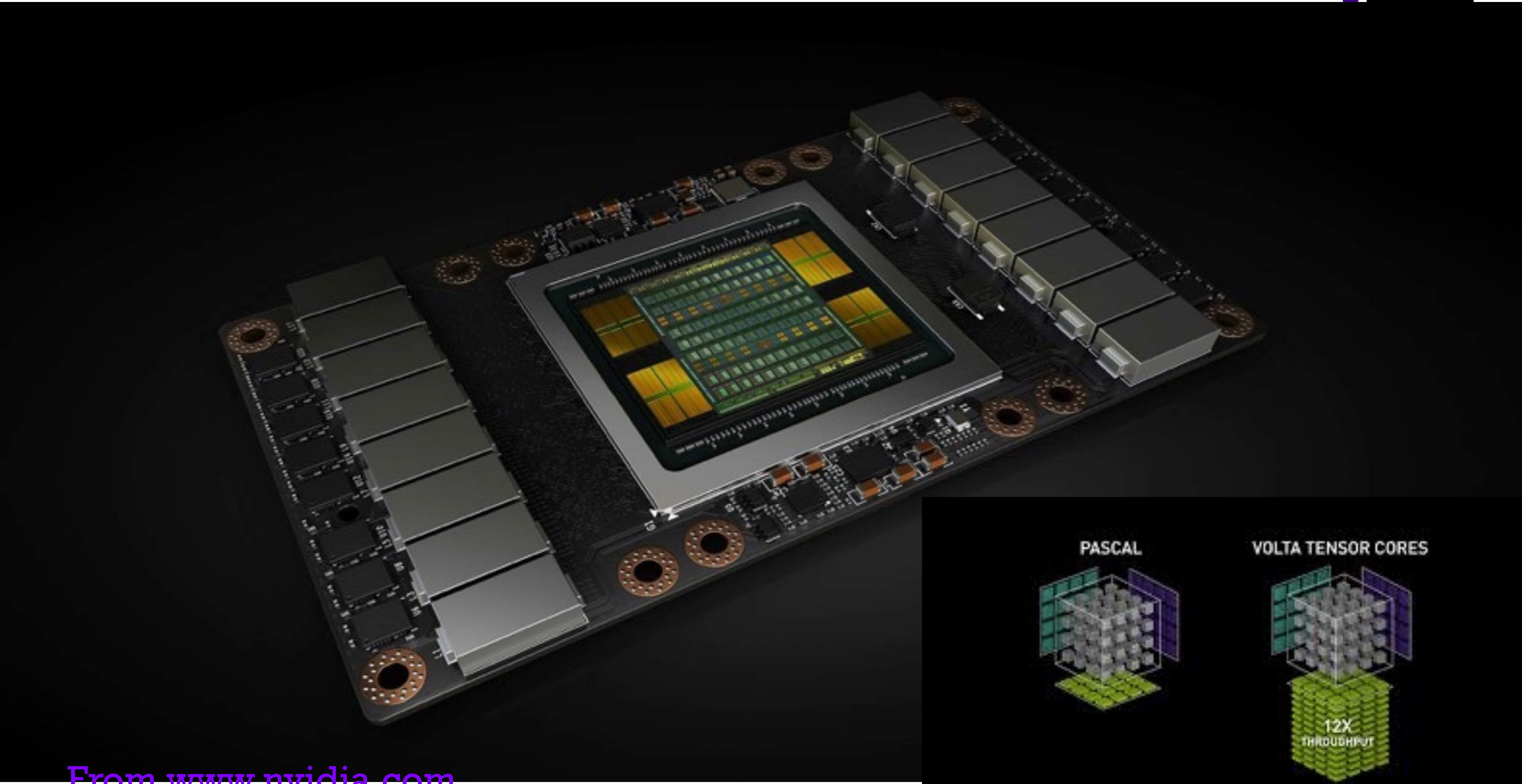
Tensor Processing Units (TPUs)

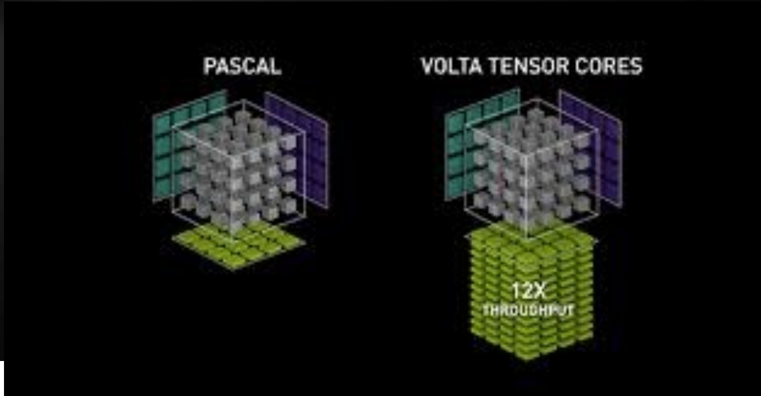




https://cloud.google.com/tpu/

+Volta Tensor Core GPU by NVIDIA®





From www.nvidia.com

TENSOR CORE

Mixed Precision Matrix Math 4x4 matrices

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

FP16 or FP32	FP16	FP16	FP16 or FP32

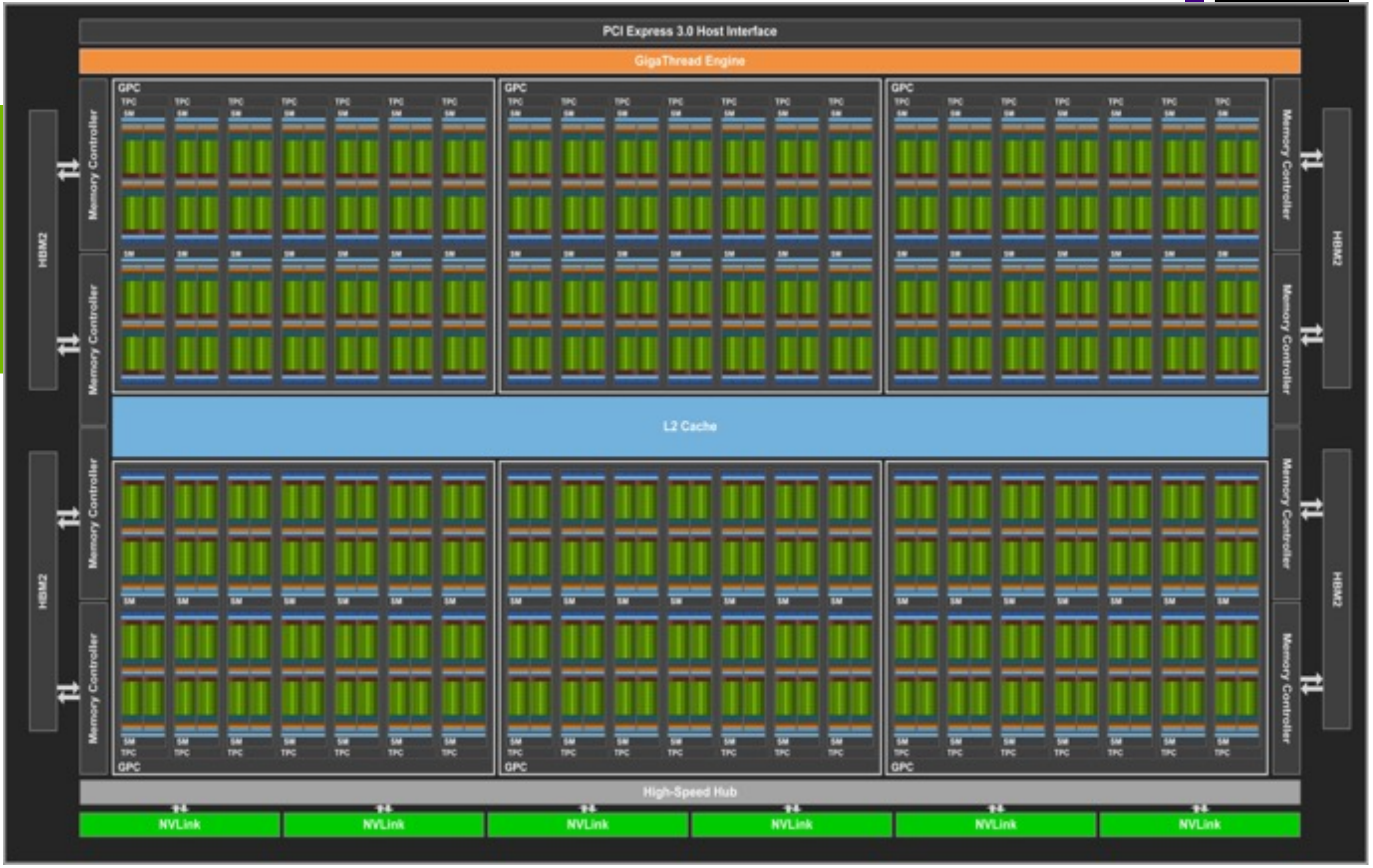
$$D = AB +$$

INSIDE A TESLA V100

21B transistors
815 mm^2

80 SM
5120 CUDA Cores

16 GB HBM2
900 GB/s HBM2
300 GB/s NVLink



*full GV100 chip contains 84 SMs

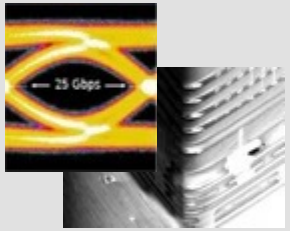
+

INTRODUCING TESLA V100

Volta Architecture

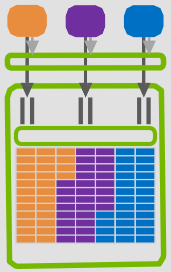
Most Productive GPU

Improved NVLink & HBM2



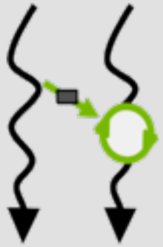
Efficient Bandwidth

Volta MPS



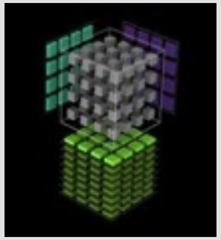
Inference Utilization

Improved SIMT Model



New Algorithms

Tensor Core



120 Programmable TFLOPS Deep Learning

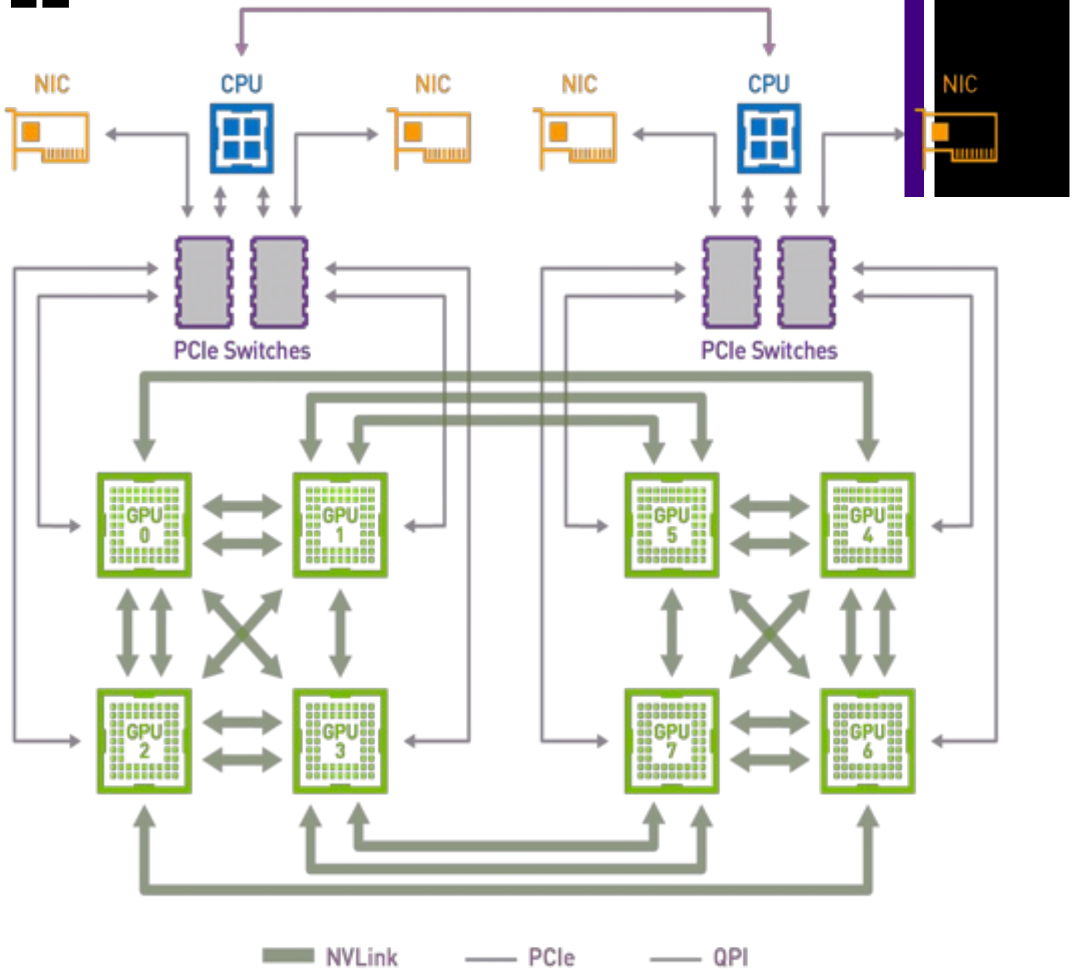
The Fastest and Most Productive GPU for Deep Learning and HPC

+
VOLTA NVLINK

300GB/sec

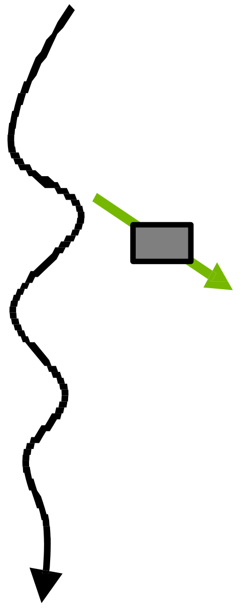
50% more links

28% faster signaling



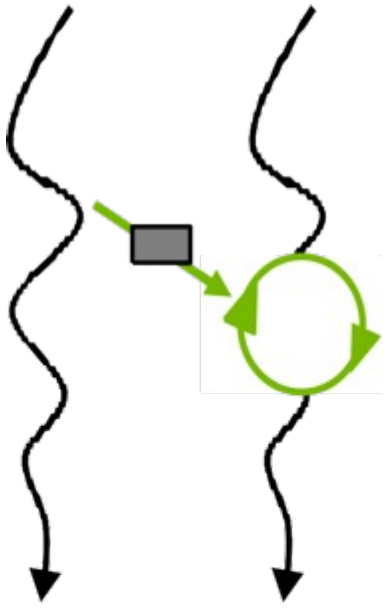
VOLTA: INDEPENDENT THREAD SCHEDULI

Communicating Algorithms



Pascal: Lock-Free Algorithms

Threads cannot wait for messages

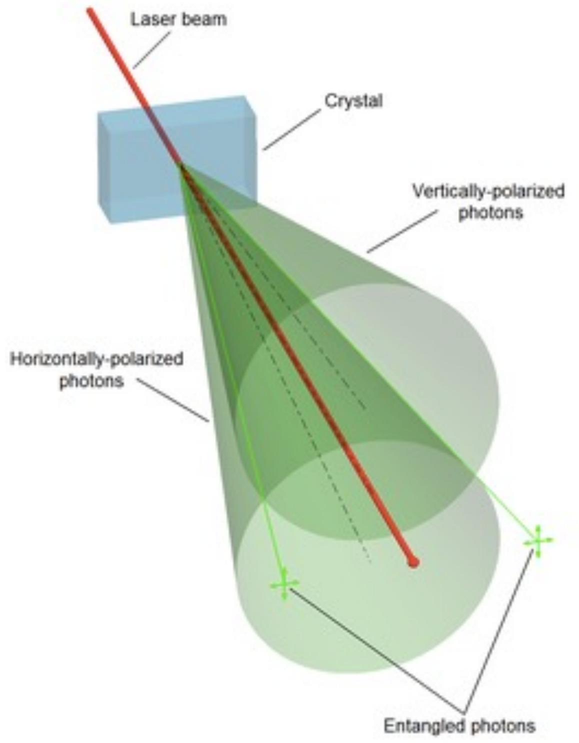


Volta: Starvation Free Algorithms

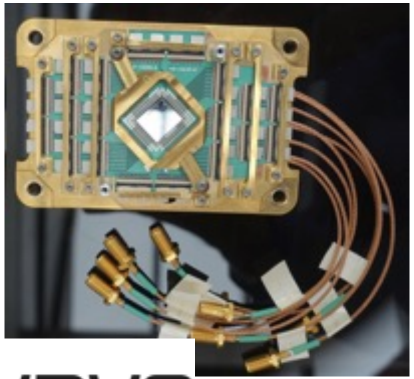
Threads may wait for messages

Quantum Unit Processing

- Quantum computing is computing using quantum-mechanical phenomena, such as superposition and entanglement
- A quantum computer is a device that performs quantum computing
- A quantum computer with a given number of qubits is fundamentally different from a classical computer composed of the same number of classical bits. For example, representing the state of an n-qubit system on a classical computer requires the storage of 2n complex coefficients, while to characterize the state of a classical n-bit system it is sufficient to provide the values of the n bits, that is, only n numbers



Quantum Computing Hardware and Simulators

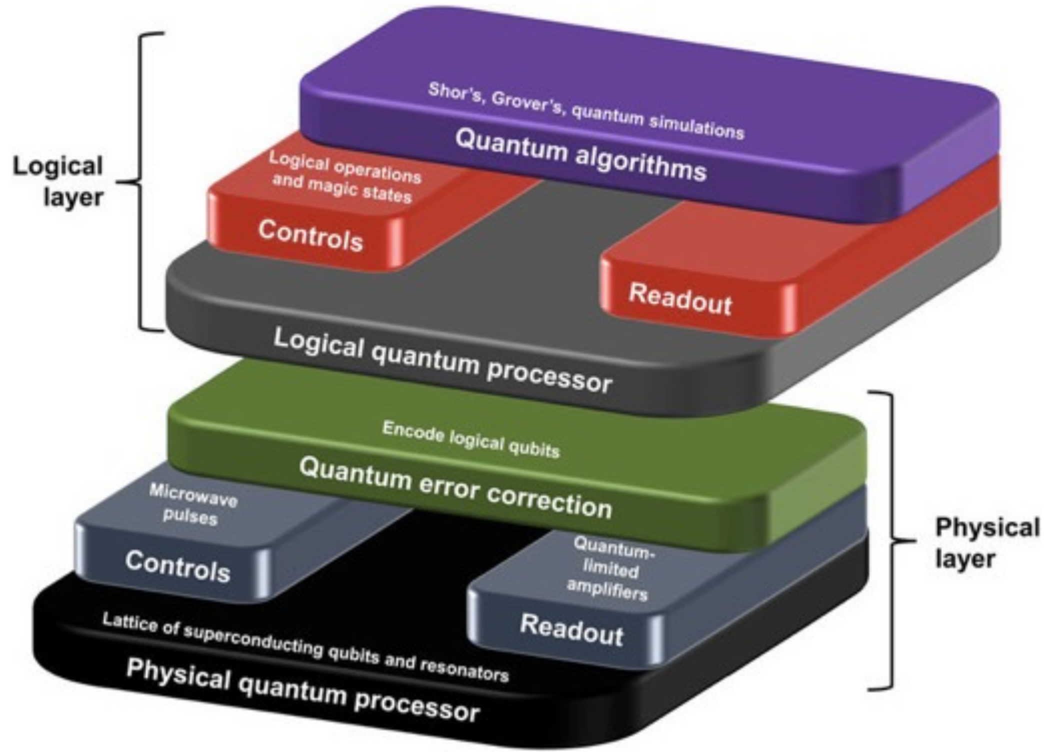


D:WAVE
The Quantum Computing Company™

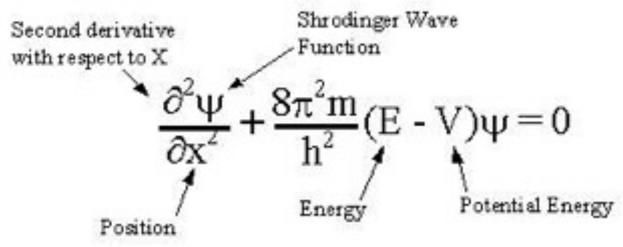
https://www.dwavesys.com/tutorials/background-reading-series/introduction-d-wave-quantum-hardware

https://atos.net/en/insights-and-innovation/quantum-computing/atos-quantum

Quantum Computing Architecture

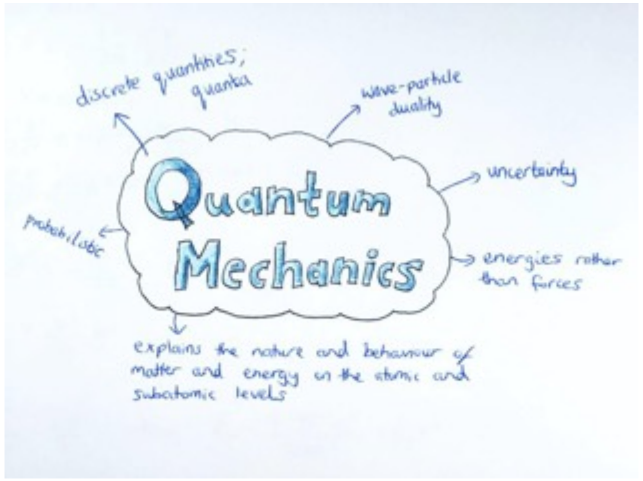

Logical layer
Shor's, Grover's, quantum simulations
Quantum algorithms
Logical operations and magic states
Controls
Readout
Logical quantum processor
Encode logical qubits
Quantum error correction
Microwave pulses
Controls
Quantum-limited amplifiers
Readout
Lattice of superconducting qubits and resonators
Physical quantum processor
Physical layer

From https://www.nextbigfuture.com/2018/04/improved-quantum-error-correction-could-enable-universal-quantum-computing.html



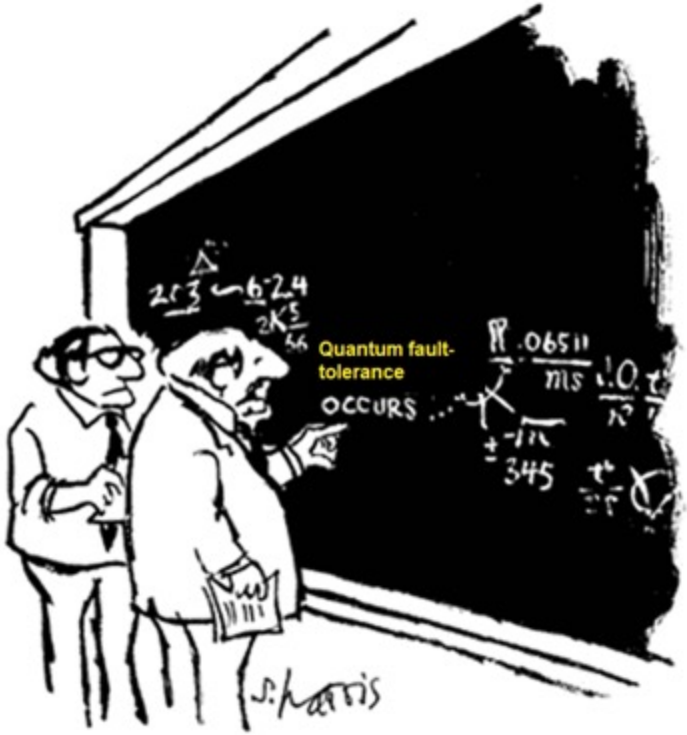
$$\frac{\partial^2\psi}{\partial x^2} + \frac{8\pi^2 m}{h^2}(E - V)\psi = 0$$

However, the mathematical aspect is very important!

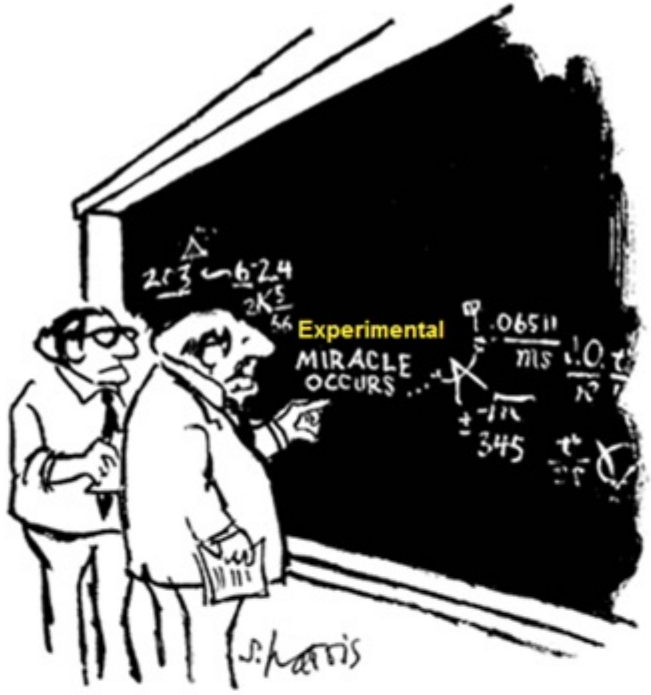

discrete quantities; quanta
wave-particle duality
uncertainty
probabilistic
Quantum Mechanics
energies rather than forces
explains the nature and behaviour of matter and energy on the atomic and subatomic levels

Conclusions

- Abstractions of Parallel Architectures are addressed to understand execution models (to see in detail after for parallel programming execution models).
  - Related with Algorithms
  - Related to Implementation Mechanisms
- Scalability is a condition to understand in some aspects:
  - Hardware
  - Data
  - Processing
- Future Directions in Computer Architecture is Parallel Computing Architecture!



Quantum Fault-tolerance occurs



Experimental Miracle occurs

Questions? (Yes & No)

Thank you!

@SC3UIS