Introduction to HPC Applications

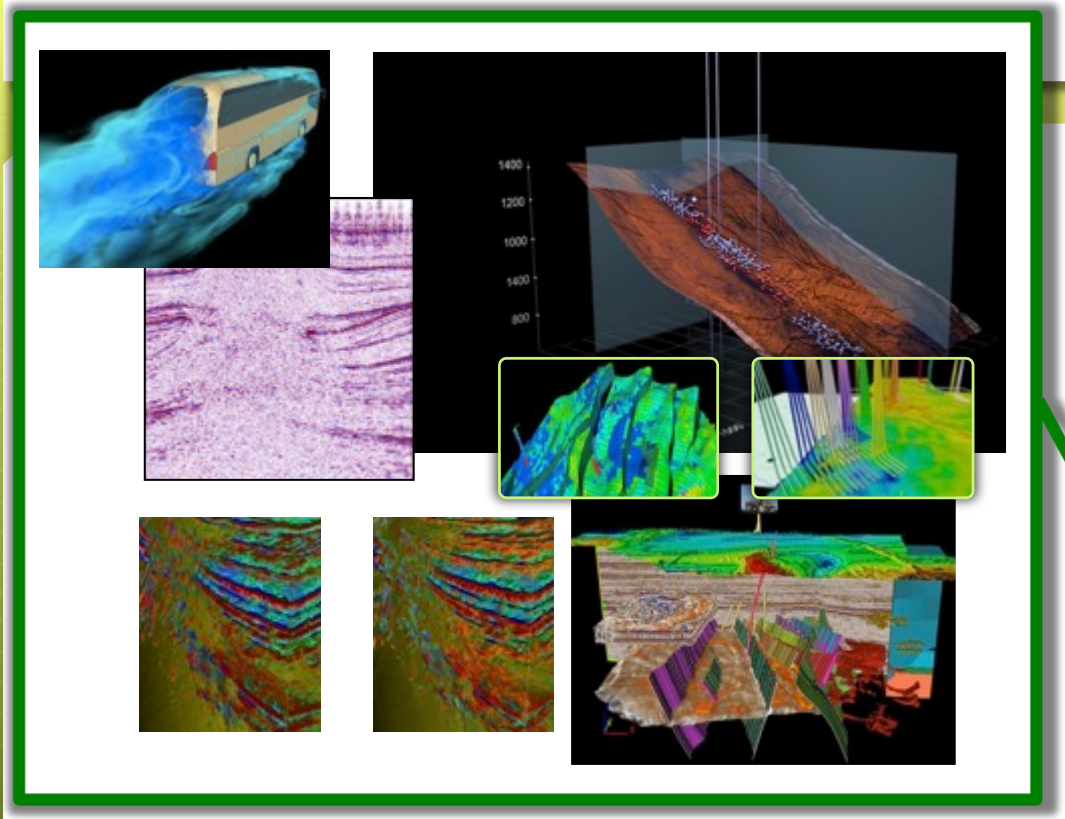
The Need of Scalable Architectures and more

Carlos Jaime Barrios Hernández, PhD
@carlosjaimebh

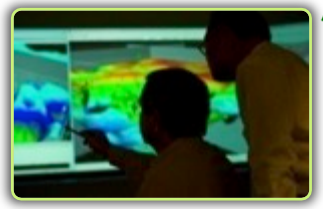
#SCCAMP2021

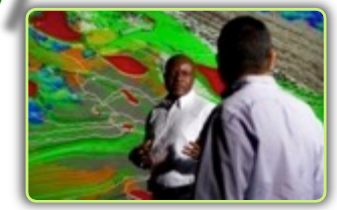
The (Big) Questions: What and How?

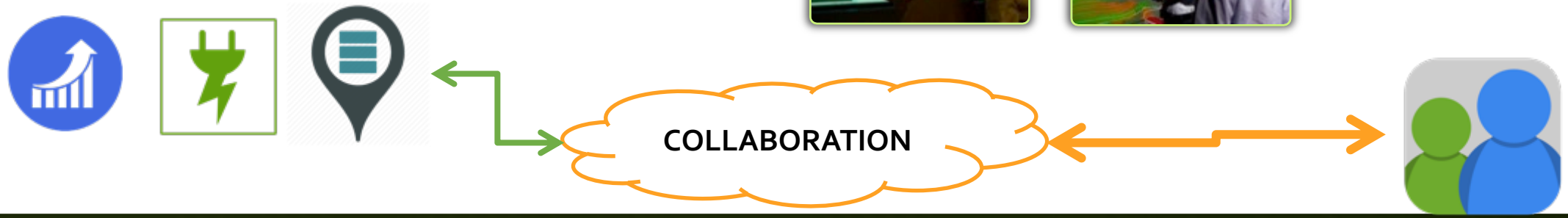
Why?



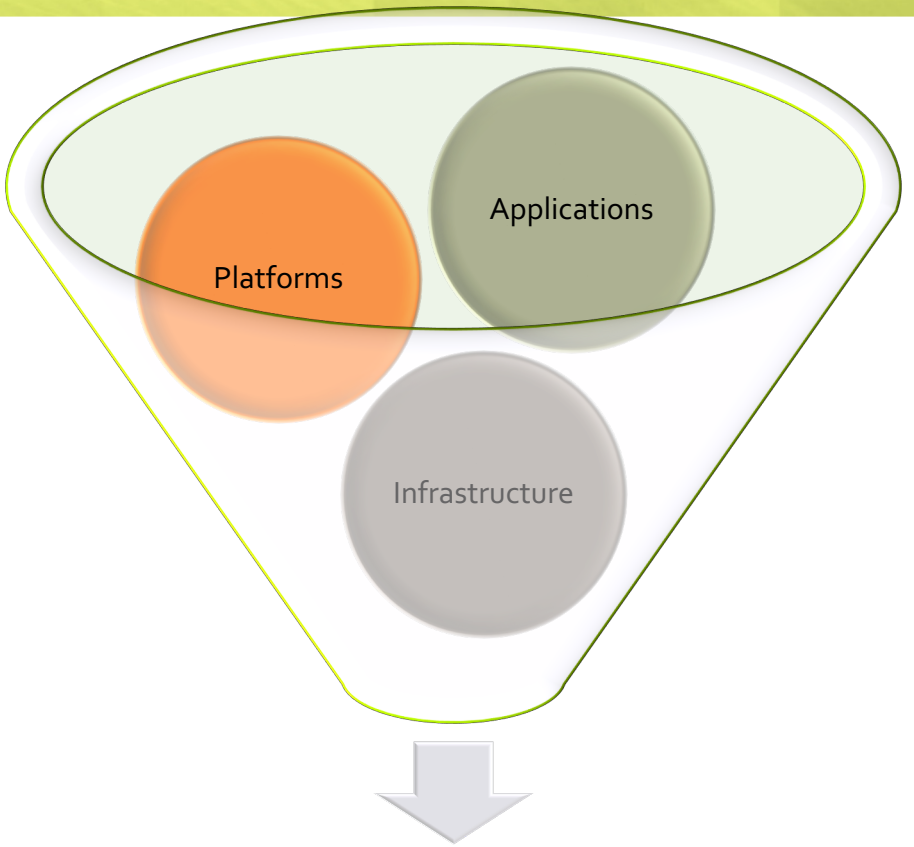
- Large Data Sets
- Complex Mathematics
- Complex Models
- Real Time
- Interaction and Confrontation
- Large Scale Visualization
- High Resolution
- High Performance and Capacity
  - VR Needs
  - Big Data and Deep Learning







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

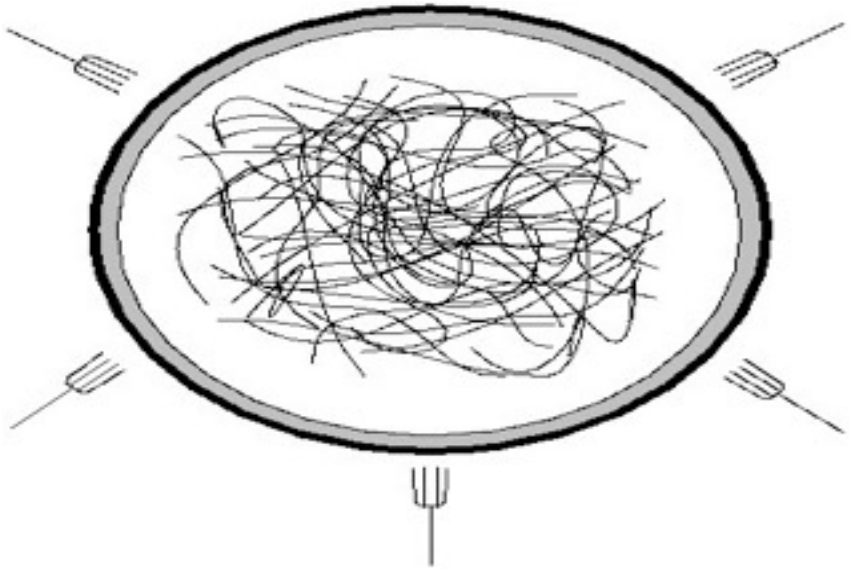
Platform

- 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.

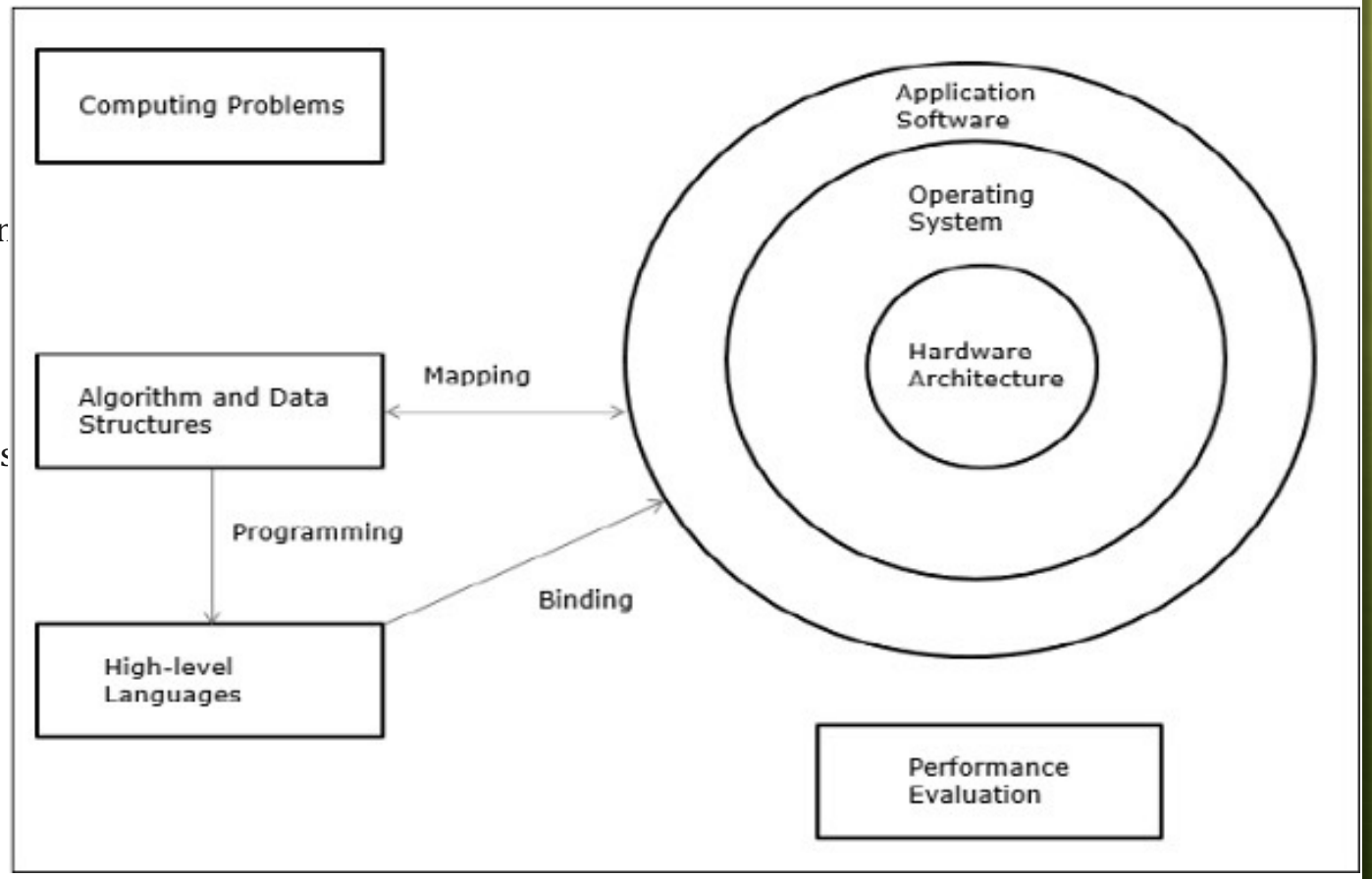
- **Implicit parallelism** is a characteristic of a programming language that allows a compiler or interpreter to automatically exploit the parallelism inherent to the computations expressed by some of the language's constructs.

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

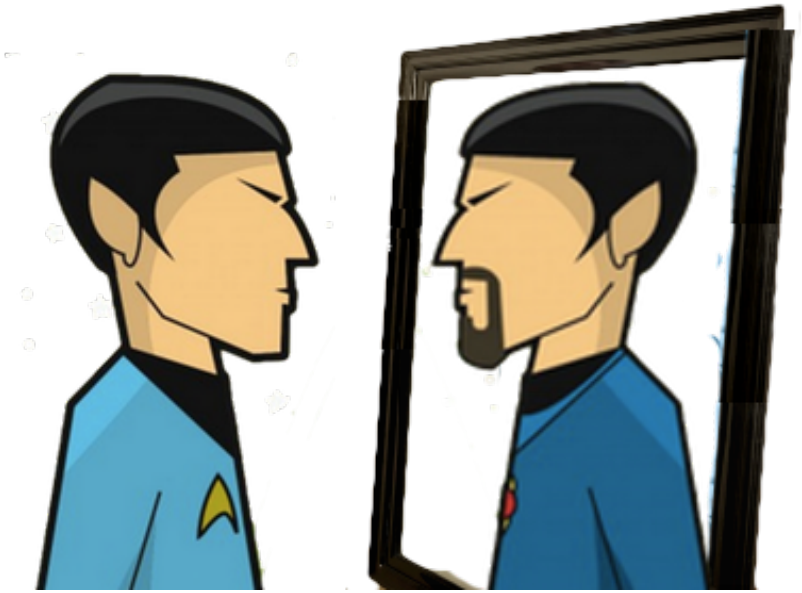
- We need two (mixed) approach in Architecture: Applications and Hardware (system).

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

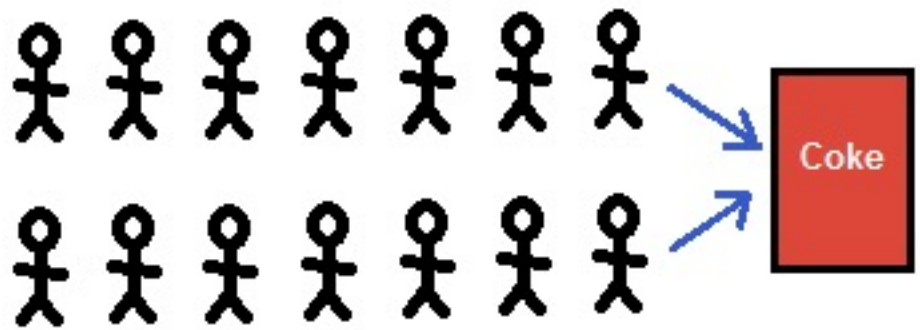


Pervasive and Thinking Parallelism

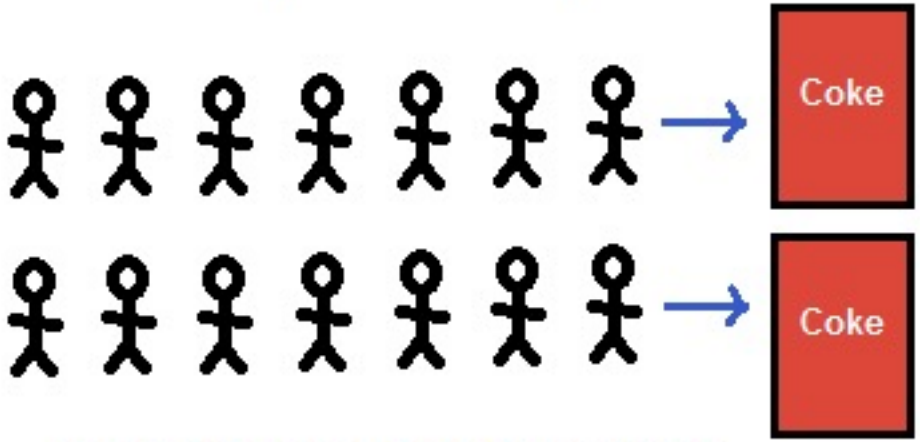


- It is not a question of « Parallel Universes » (Almost)
- Data Sources
- Processing and Treatment
- Resources (Available and Desire)
- Energy Consumption
- Natural “thinking” (Natural Compute?)

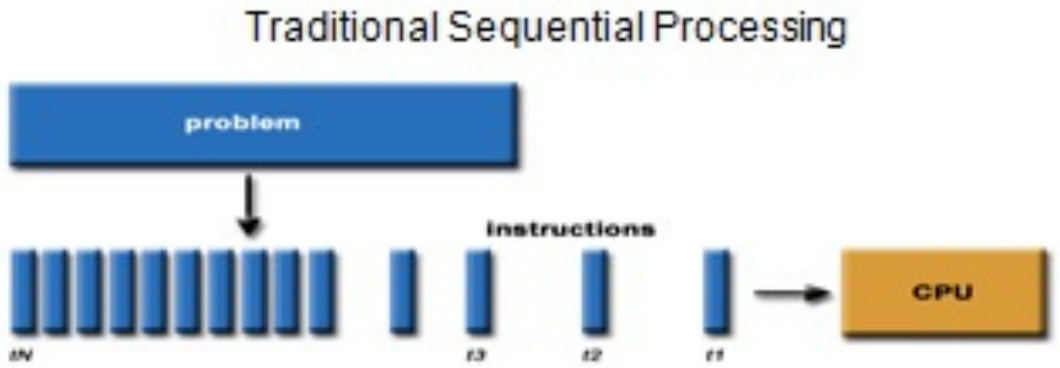
Thinking in Parallel (computing) – The Typical Visions

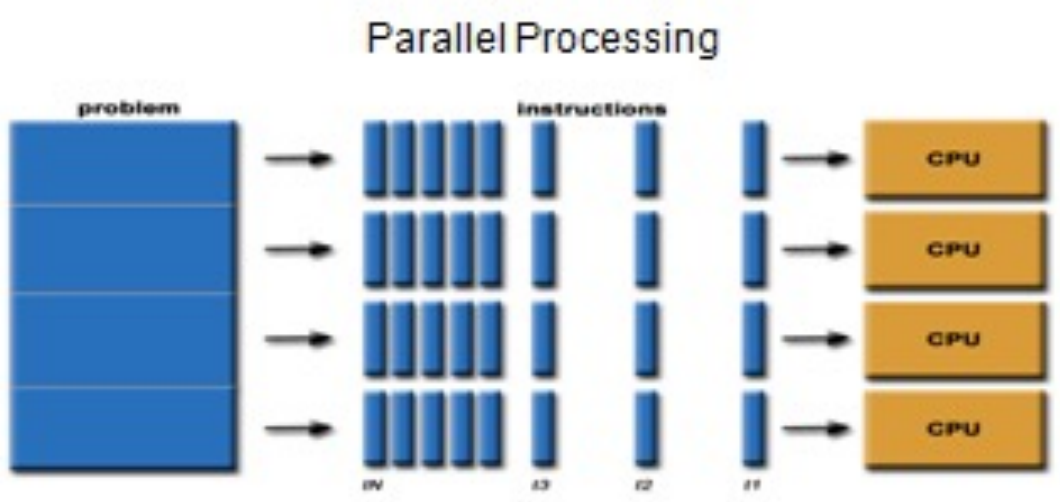


Concurrent: 2 queues, 1 vending machine

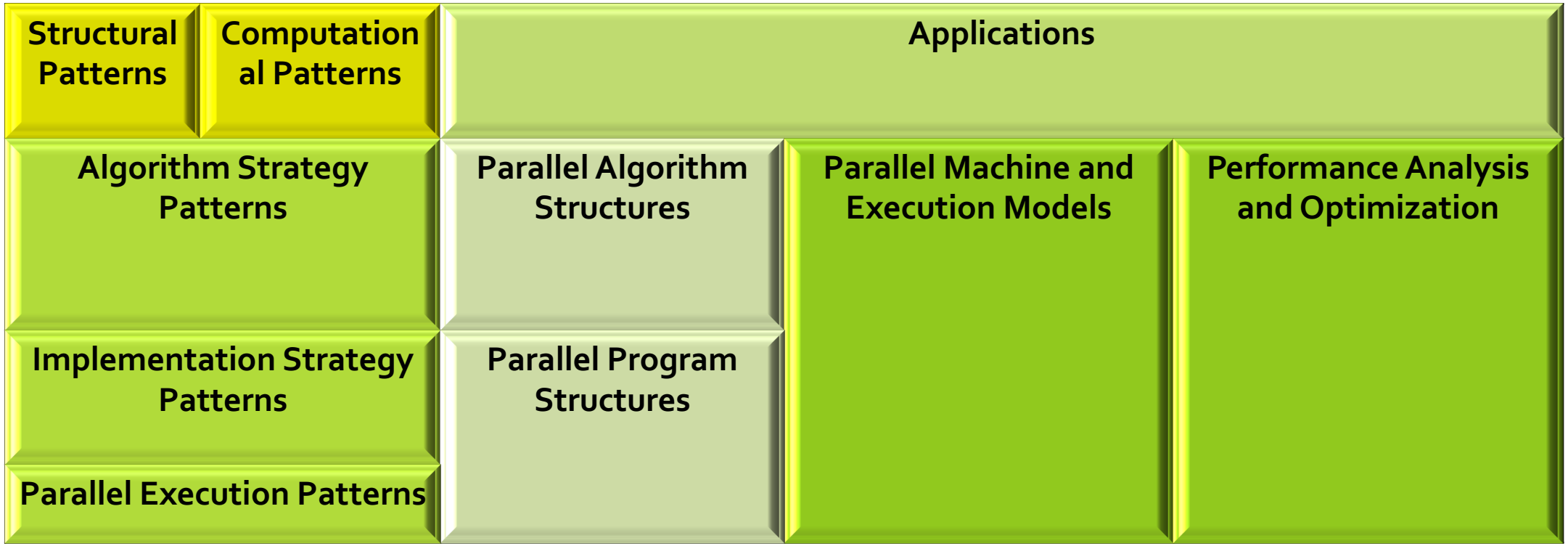


Parallel: 2 queues, 2 vending machines

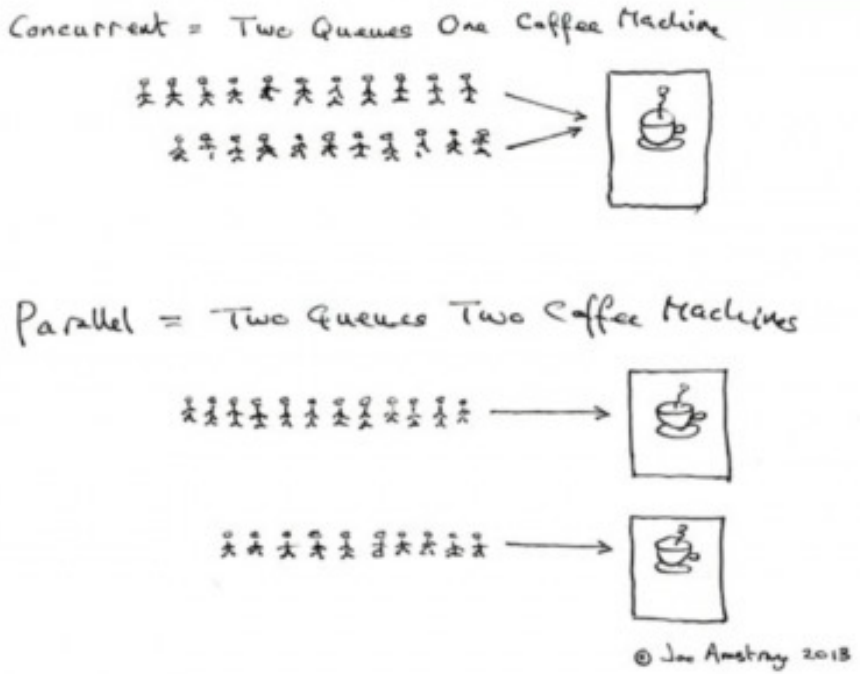


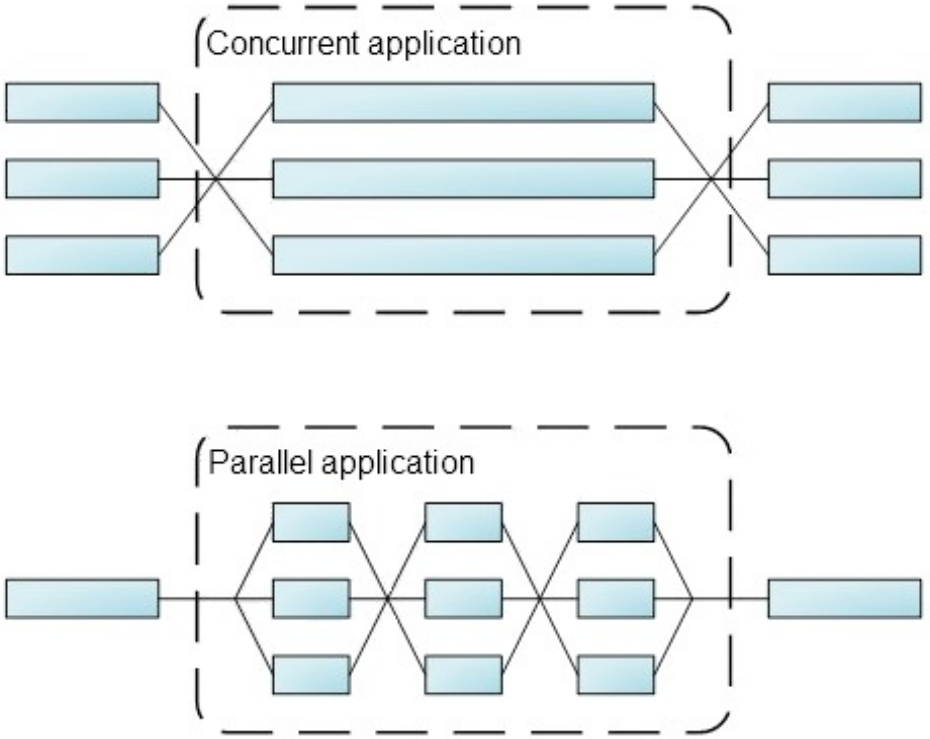


Thinking in Parallel (computing) – an OPL hierarchy



CONCURRENCY | PARALLELISM

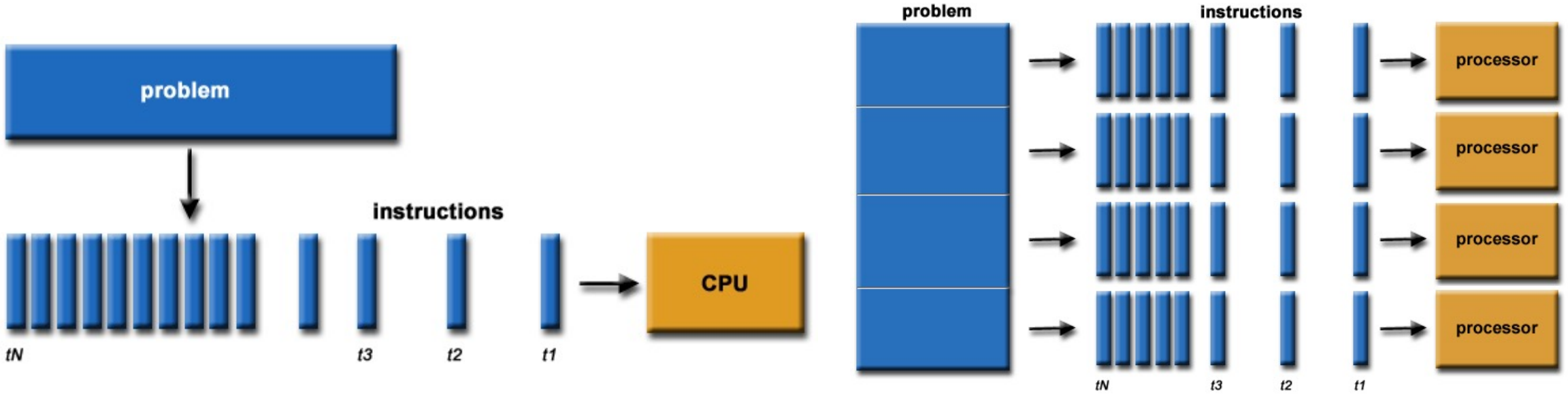




From J. Armstrong Notes: http://joearms.github.io/2013/04/05/concurrent-and-parallel-programming.html

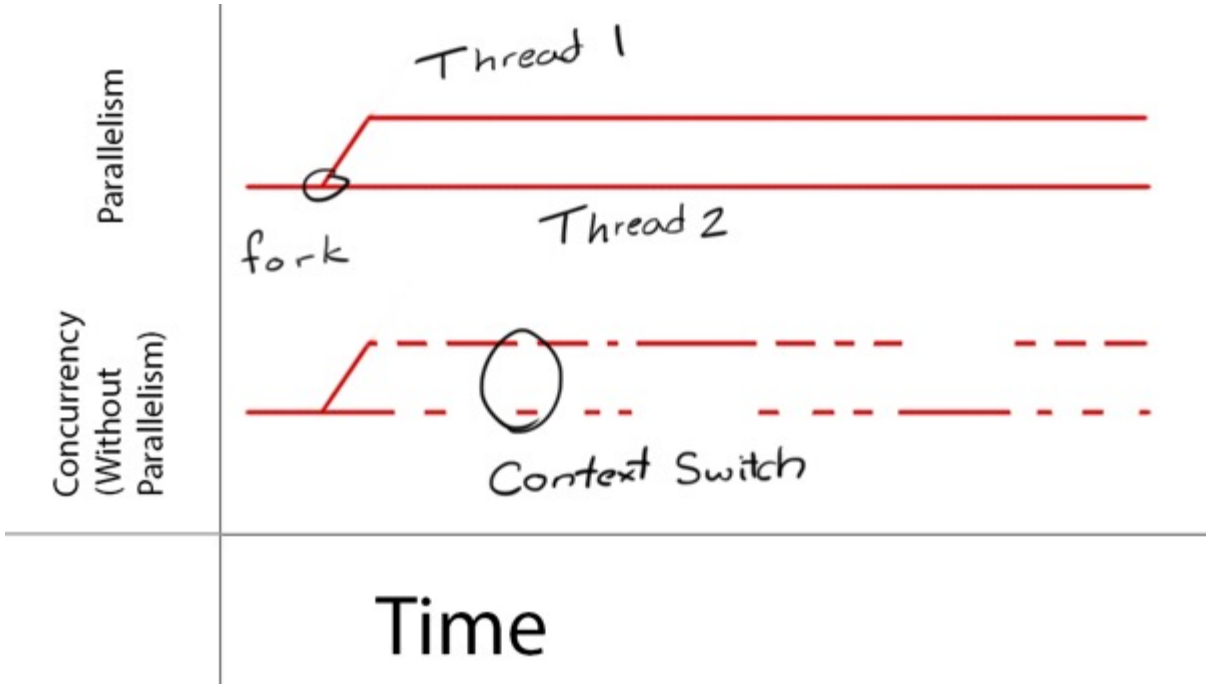
Any Parallel System is concurrent: Simulatenous Processing, Parallel but limited ressources.

Serial vs Concurrent/Parallel Approach



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

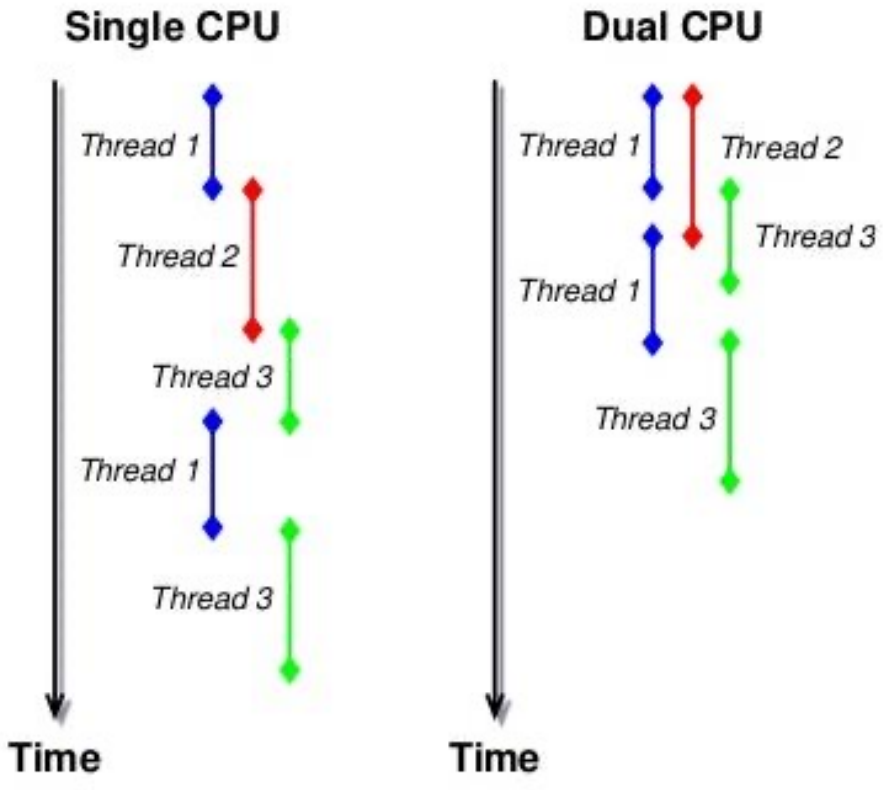
Concurrency vs Concurreny/Parallelism Behavior



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

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

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

Sequential Processing

- All of the algorithms we've seen so far are sequential:
  - They have one "thread" of execution
  - One step follows another in sequence
  - One processor is all that is needed to run the algorithm

- Concurrent Systems

- A system in which:
  - Multiple tasks can be executed at the same time
  - The tasks may be duplicates of each other, or distinct tasks
  - The overall time to perform the series of tasks is reduced

- **Advantages of Concurrency**

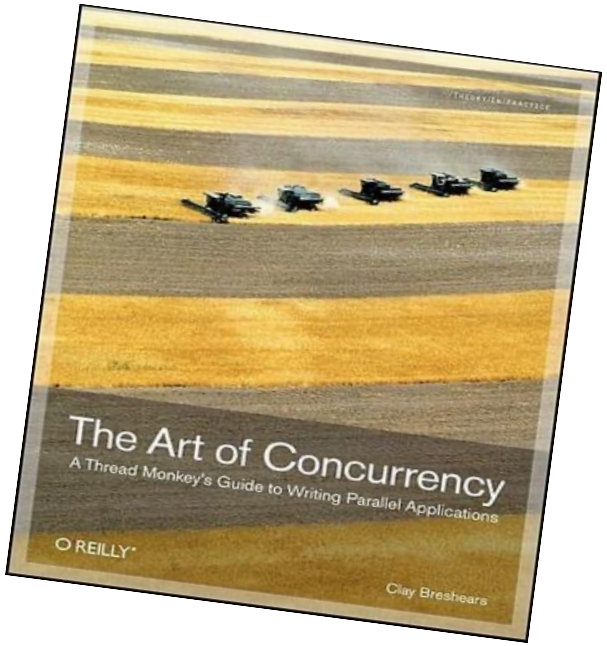
- **Concurrent processes can reduce duplication in code.**
- **The overall runtime of the algorithm can be significantly reduced.**
- **More real-world problems can be solved than with sequential algorithms alone.**
- **Redundancy can make systems more reliable.**

- **Disadvantages of Concurrency**

- **Runtime is not always reduced, so careful planning is required**
- **Concurrent algorithms can be more complex than sequential algorithms**
- **Shared data can be corrupted**
- **Communications between tasks is needed**

Parallel Computing

- Parallel Computing exploit Concurrency
  - In “system” terms, concurrency exists when a problem can be decomposed in sub problems that can safely executed at same time (in other words, concurrently)



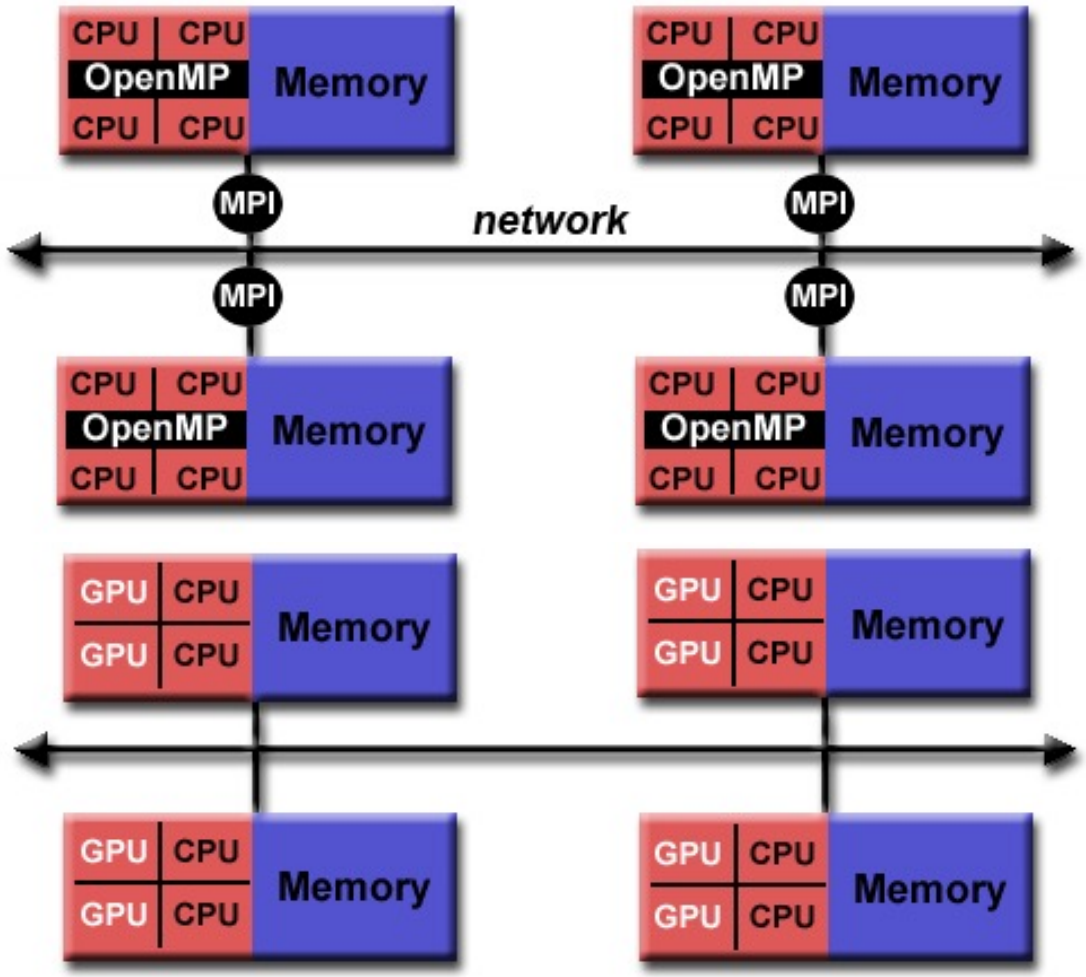
https://ignorelist.files.wordpress.com/2012/01/the-art-of-concurrency.pdf

How to Exploit (Better) Concurrency

- (Remember) Mixed Approach (Algorithms/Applications - Hardware/System.
- Good Techniques from Software Engineering
- Good Problem knowledge from scientific (domain) expertise
- Confrontation and Performance Evaluation

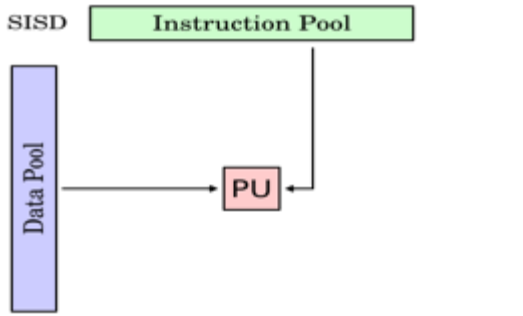
The Hardware/System Approach

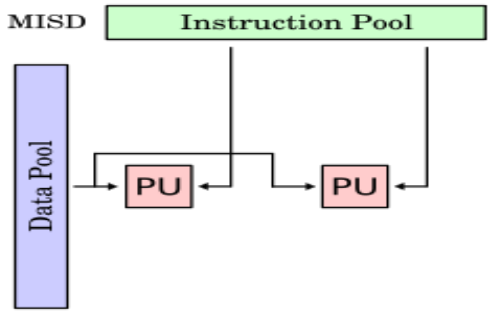
Shared, Distributed and Hybrid Memory Architectures

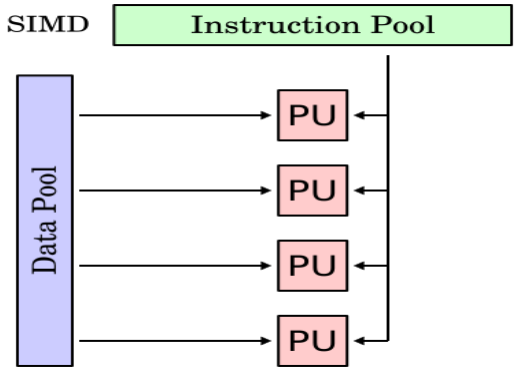


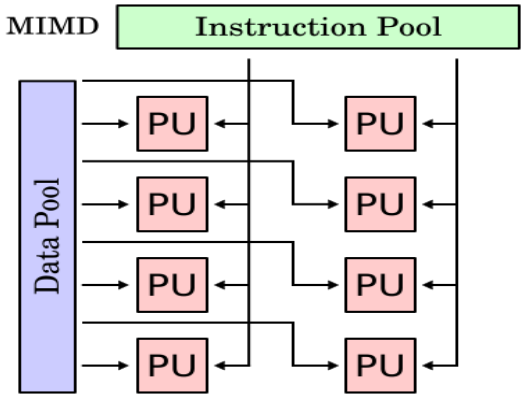
- Memory Exploitation involves Memory Hierarchy
  - Models as PRAM, BSP, etc..
- All modern architectures to HPC allows different memory models
  - Shared Memory (Inside Nodes)
  - Distributed Memory (Among Nodes)
  - Hybrid Memory
    - Using Accelerators (GPUs, MICs)
    - Interaction Nodes/Processors

Flynn's Taxonomy*









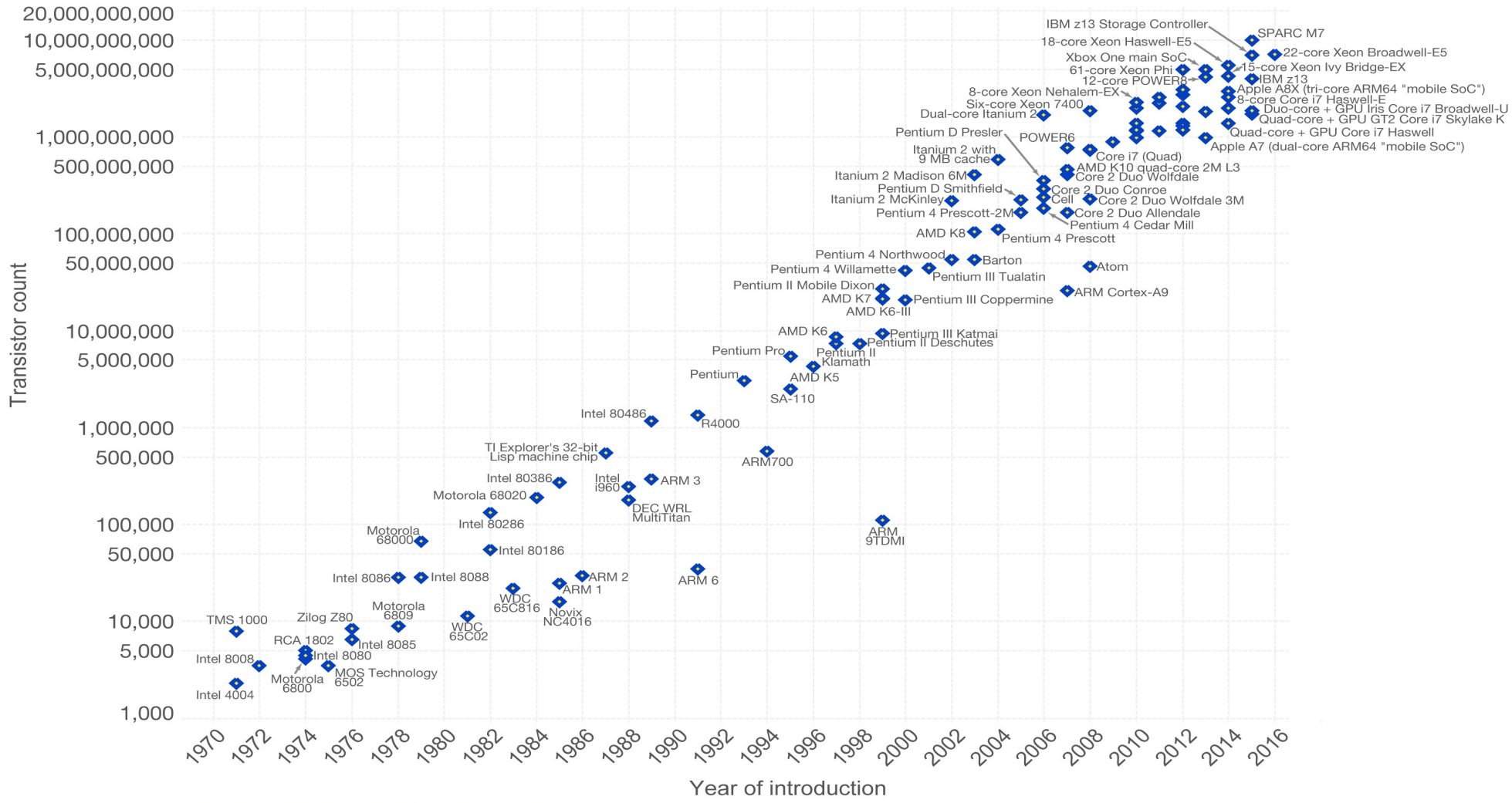
* Proposed by M. Flynn in 1966

The Moore Evolution

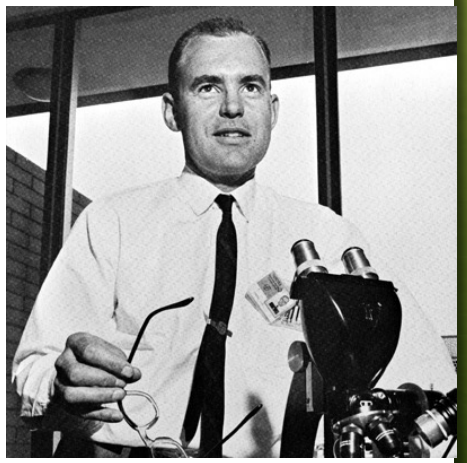
Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.

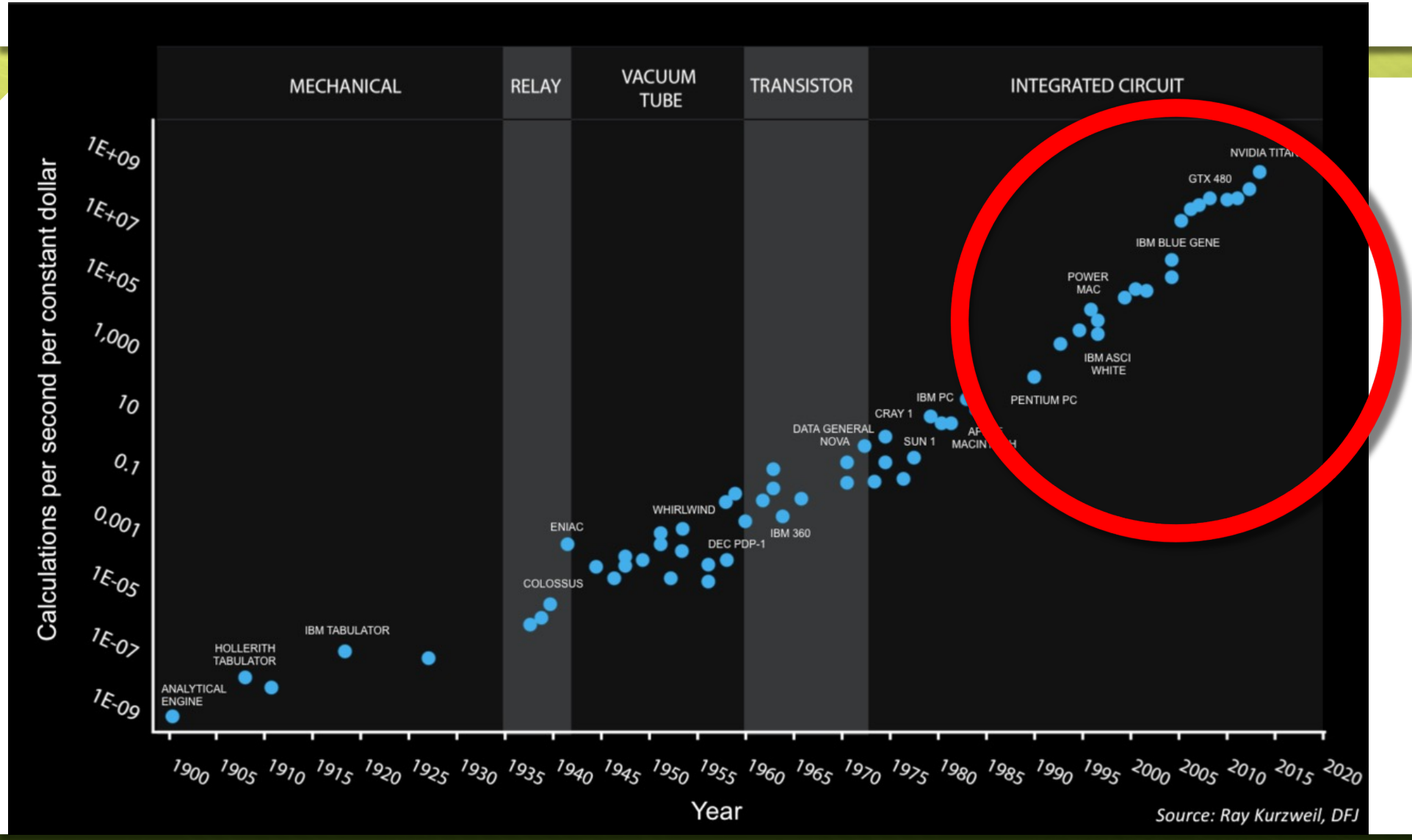


Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.



Gordon Moore (In the 60's)

The (Post) Moore Era



After 120 years… The Moore's Law is Dead

Jack Dongarra

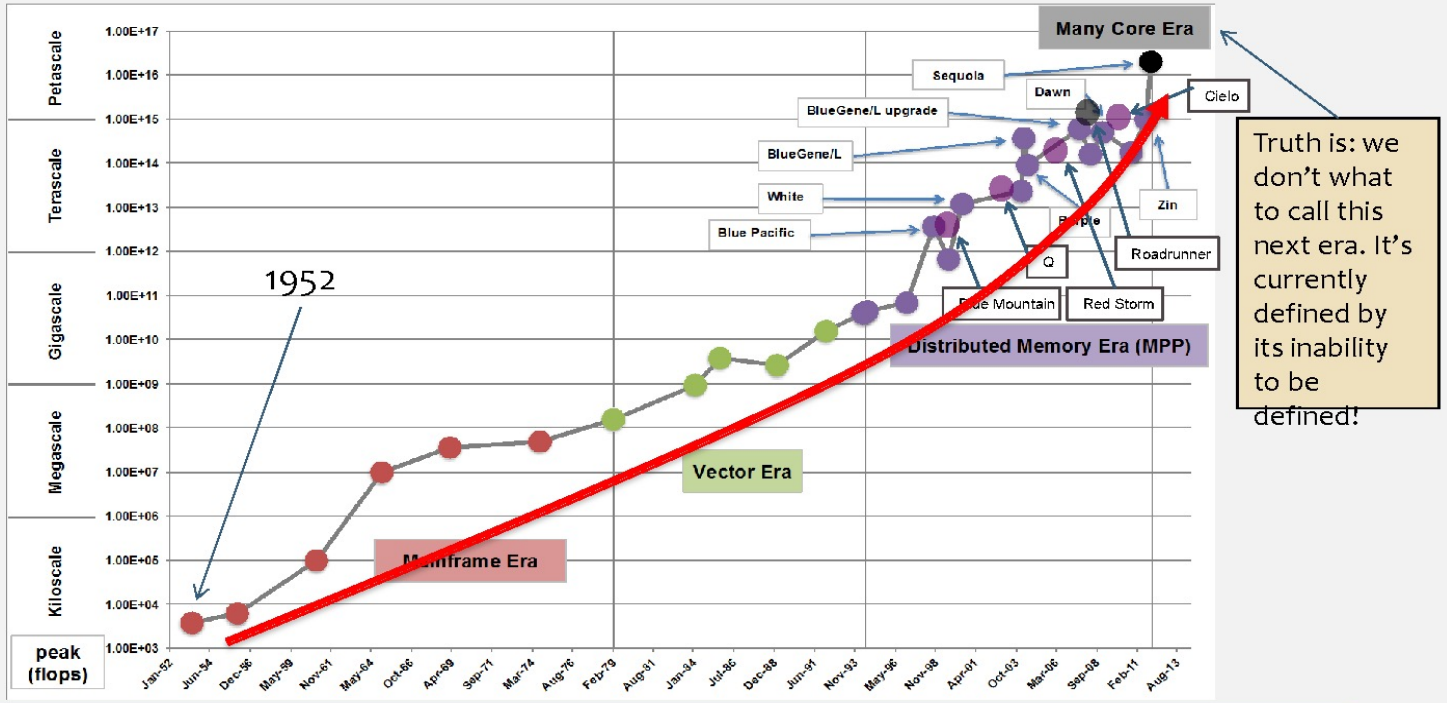
Parallel Computing Everywhere

It is more than a publicity!

Parallel Computing Evolution

(From the LLNL Vision by Rob Neely)

Advancements in (High Performance) Computing Have Occurred in Several Distinct "Eras"



Each of these eras define not so much a common hardware architecture, but a common programming model

Rob Neely

Configurable Architectures

Large Scale Cores
(High Single Thread Performance)

Dual Cores
(Symmetric Multithreading)

MultiCore Arrays

Scalar + Many Cores
(Highly threaded workloads)

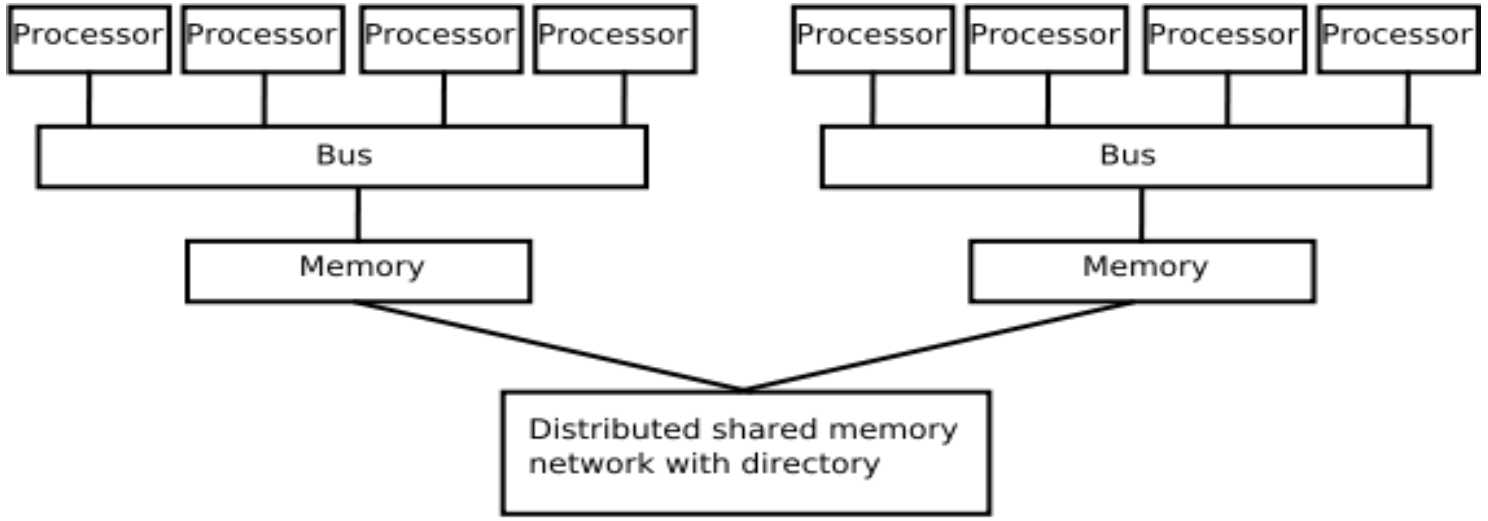
Manycore arrays

Further Taxonomy

(Derivate from MIMD for distributed memory programming)

- SPMD (Single Program, Multiple Data Streams or Single Process, Multiple Data)
  - Multiple autonomous processors simultaneously executing the same program on different data.
  - It is the most common taxonomy.
- MPMD (Multiple Program Multiple Data)
  - Multiple autonomous processors simultaneously operating at least 2 independent programs.

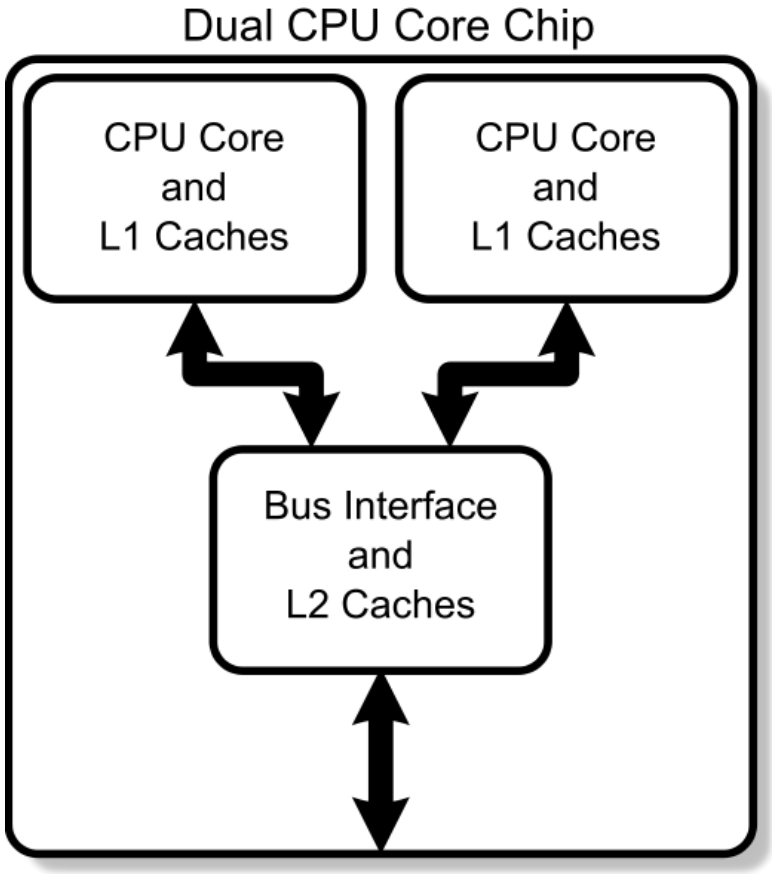
The Distributed Shared Memory Access



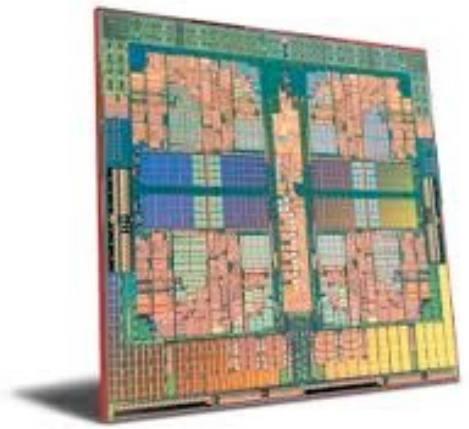
- Main Memory in Parallel Machines is a hybrid between shared memory and distributed memory.
- Uniform Memory Access (UMA) is proposed for End-User Systems (Generally)
- Distributed memory systems have Non-Uniform Memory Access (NUMA) architecture

Multicores

- Multicore Computer:
  - Multicore Processor includes multiple execution units (cores)
  - Minimal (Physic) Processing unit is the core. The Core support processing of threads (almost one thread)
  - Cache memory is important to threads exchange between cores and memory.

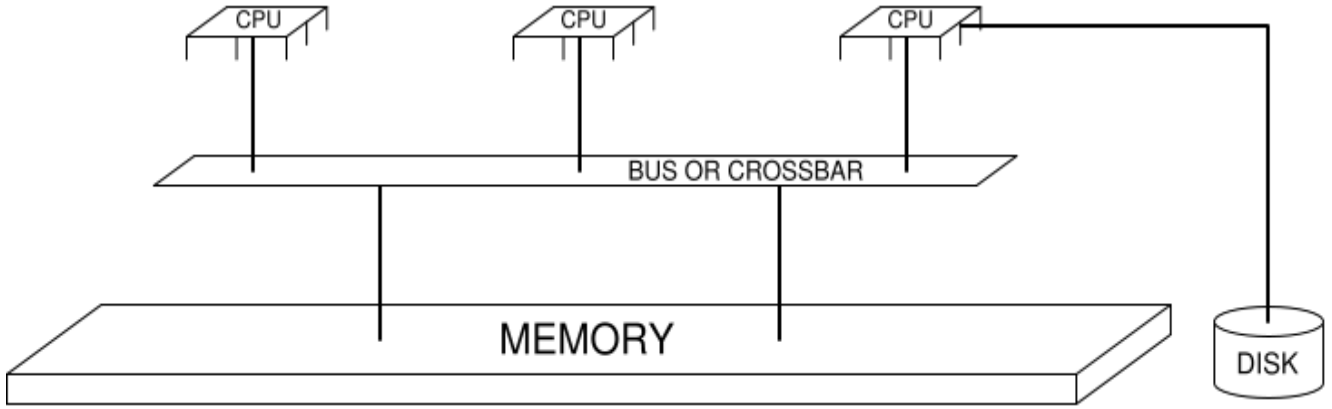


Dual CPU example



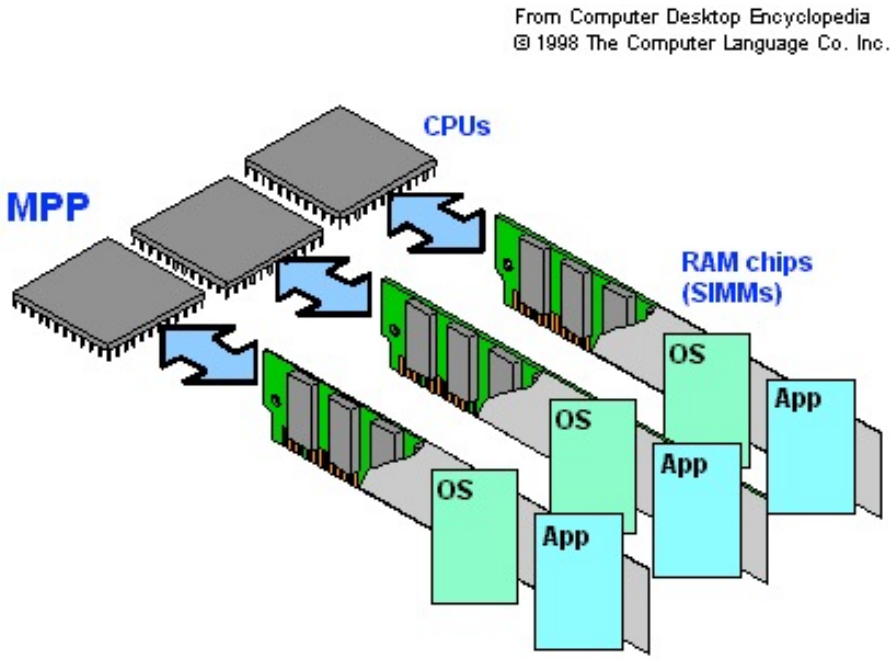
Symetric Multiprocessing

- Symmetric multiprocessors:
  - A symmetric multiprocessor (SMP) is a computer system connected to a main shared memory.
  - Intel's Xeon is the most known SMP system.
  - Sun Microsystems UltraSPARC was the first multiprocessing system.



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.
- GPGPU Computing exploit MPP.



More of Parallel Computers

- Reconfigurable Computing with Field-Programmable Gate Arrays (FPGA).
- General-Purpose Computing on Graphics Processing Units (GPGPU).
  - Programmin with CUDA and OpenCL (i.e.)
- Application-Specific Integrated Circuits (ASIC).
- Vector Processors. (SIMD)

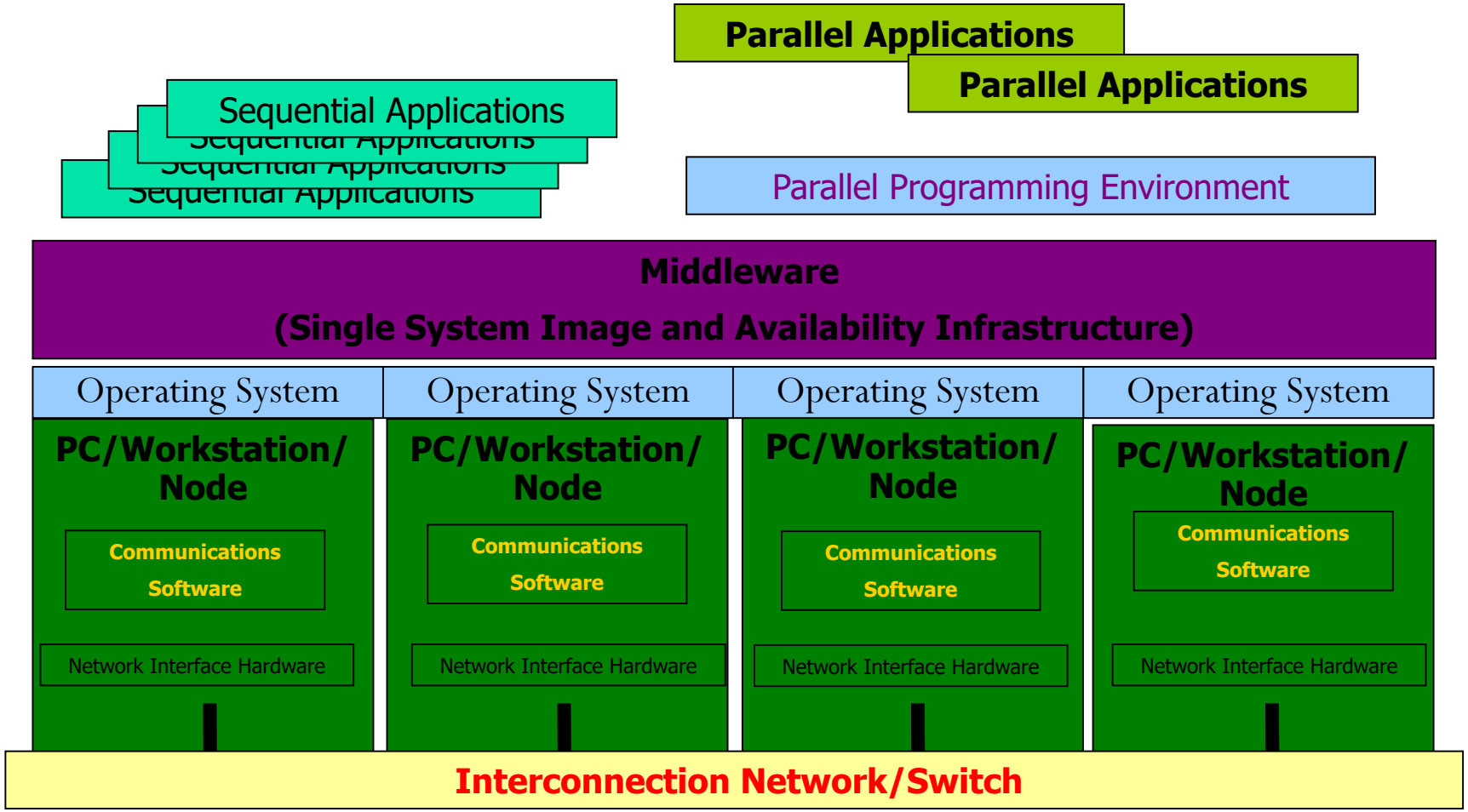
Distributed Computers

- Distributed computing:
  - Distributed Computing is a Distributed Memory Multiprocessor System connected by a network.
  - Distributed computers are **highly scalable!**
  - Cases of Distributed Computing:
    - Cluster computing (Parallel Distributed Computing)
    - Grid Computing

Large Scale Architectures

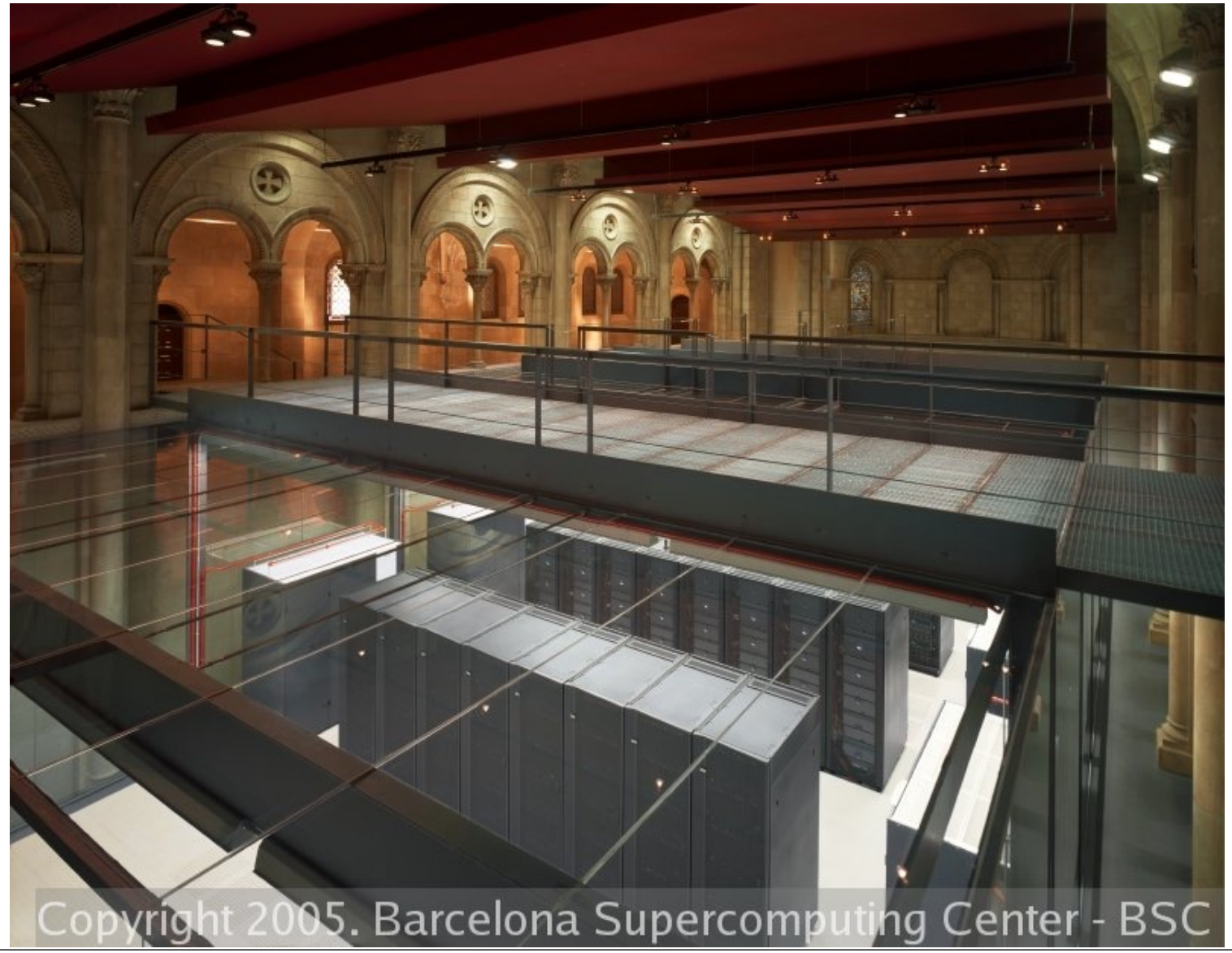
- Large Scale Architecture (LSA) allows to trait large scale problems.
  - LSAs need Large Scale Sotware
  - LSAs are distributed systems.
    - Cluster Computing Platform
    - Grid Computing Infrastructure
  - The Fault tolerance is a critical problem in LSA systems.

Cluster Computing Architecture



An Spain Exemple: BSC-CNS Marenostrum

www.bsc.es

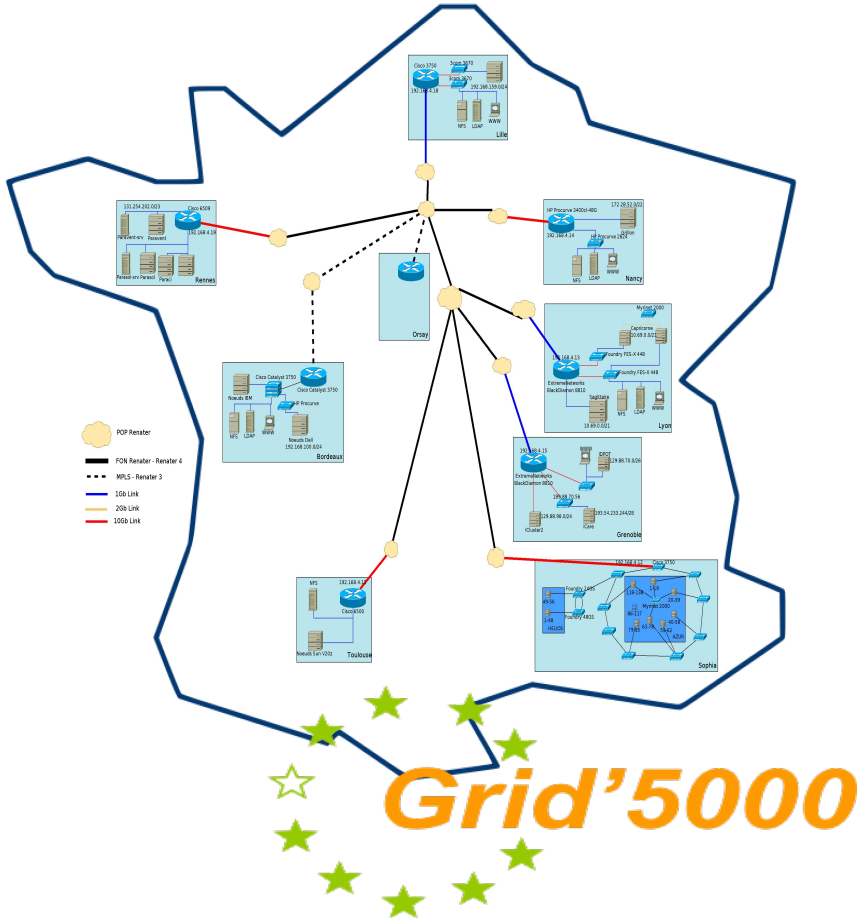


- 11.15 Petaflops
- 384.75 TB Memory
- 3.465 Computing Nodes
  - 2x Intel Xeon Plantium 8160 24C /2.1Ghz
  - 216 Nodes with 12x32GB DDR4 2667 DIMMS (8GB Cores)
  - 3240 Nodes with 12x8GB DDR4-2667 DIMMS (2GB Cores)
- Network
  - 100Gb Intel Omni-Path Full Fat Tree
  - 100Gb Ethernet
- Operating System
  - Suse Linux Enterprise Server 12SP2

Grid Computing

- Grid Computing implies technology, technics and methodology to support Parallel*/Distributed Computing.
- Grid Computing needs Grid Computing Infrastructure and dedicated and high disponibility networks or interconexion.
- Different Types or Possibilities:
  - Experimental Testbeds
  - Production Grids
  - Lightweigth Grids
  - Desktop Grid Computing (May be Lightweigth too)
- **Grid Computing is in the back of Cloud Computing Systems (from Infrastructure Point of View)**

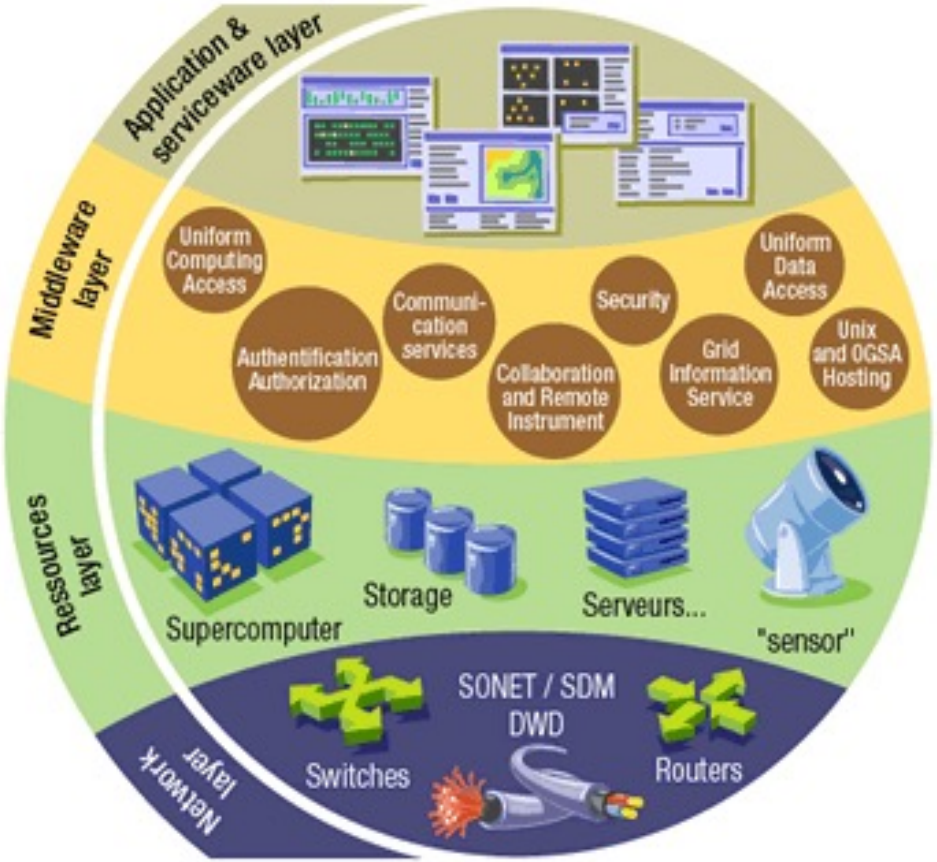
Grid Computing Feautures

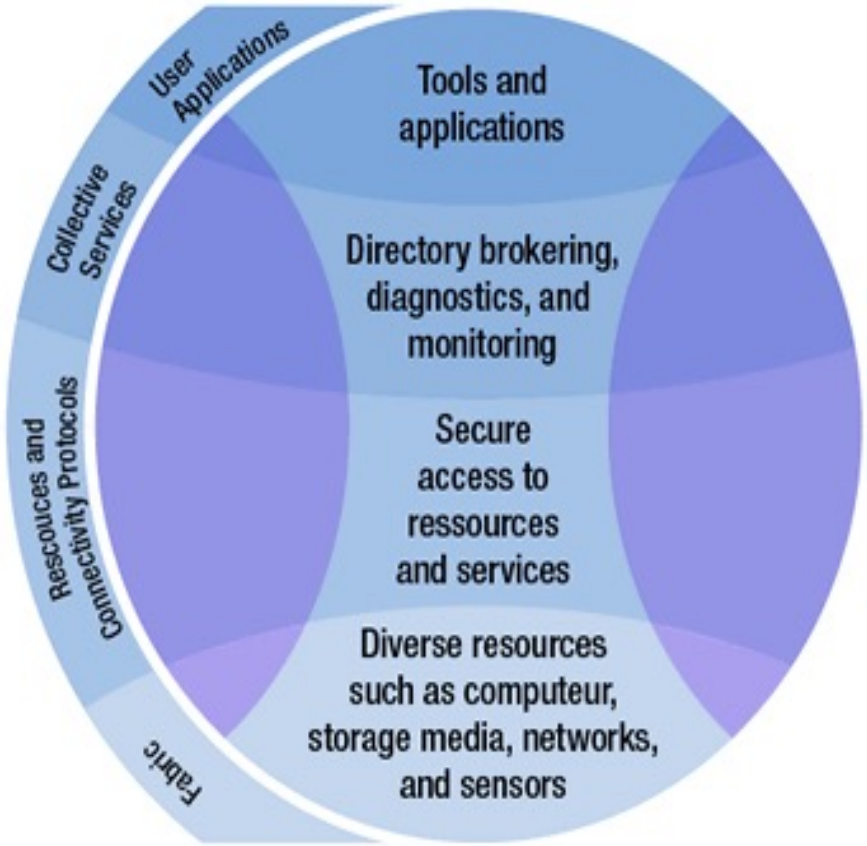


http://www.grid5000.fr

- Grid Computing Features:
- Infrastructure
  - High Availability
  - High Performance
  - Heterogeneity
  - Pervasive
  - Scalability
- Methodology
  - Different User Levels
  - Multi Administration
- Politics
  - Security
  - Use
  - Privacy

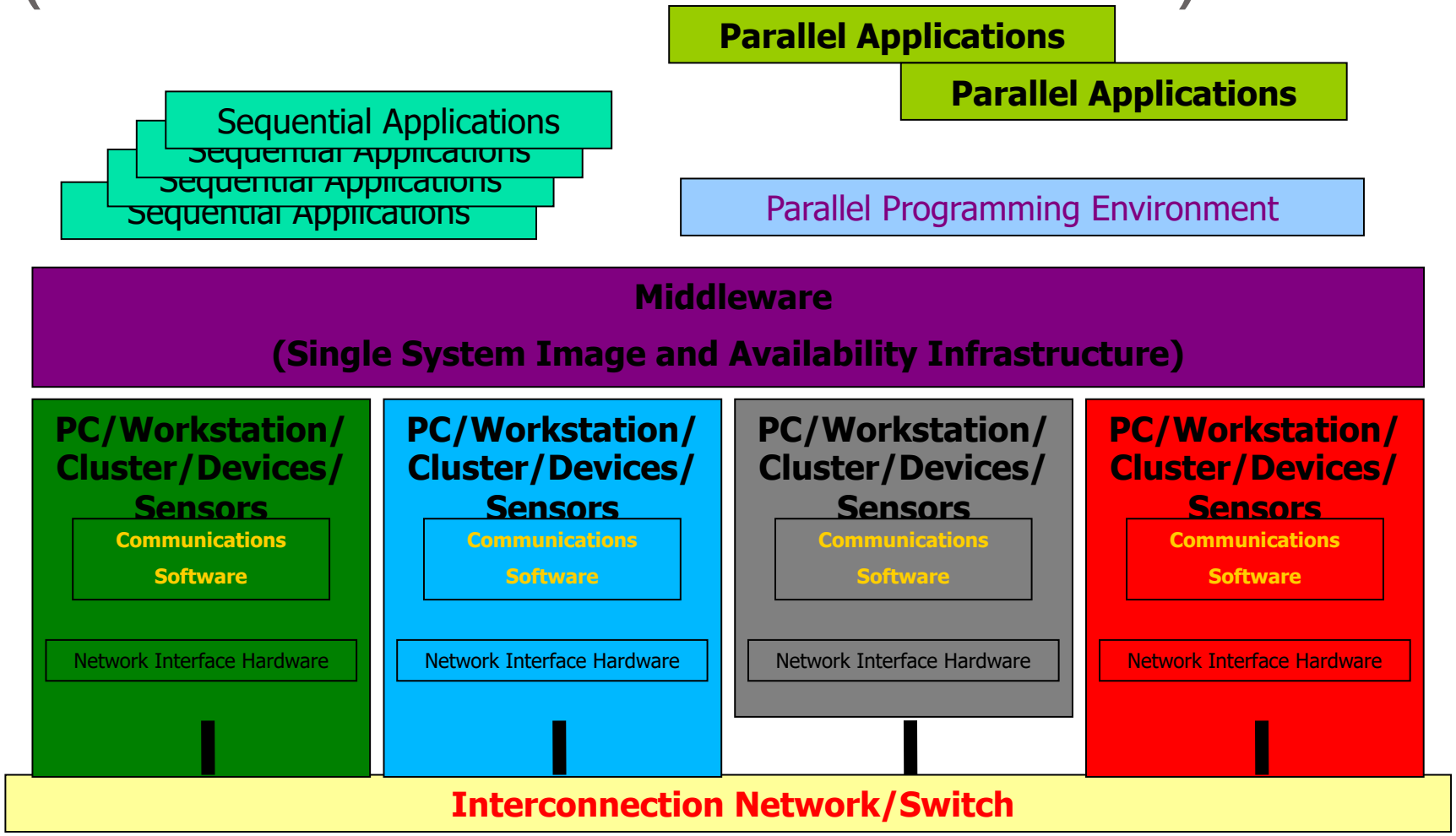
Grid Computing Architecture (Typical Diagram)





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

Grid Computing Architecture (Remember the Cluster Architecture)



An Example: The French Aladdin Grid5000 (G5K)

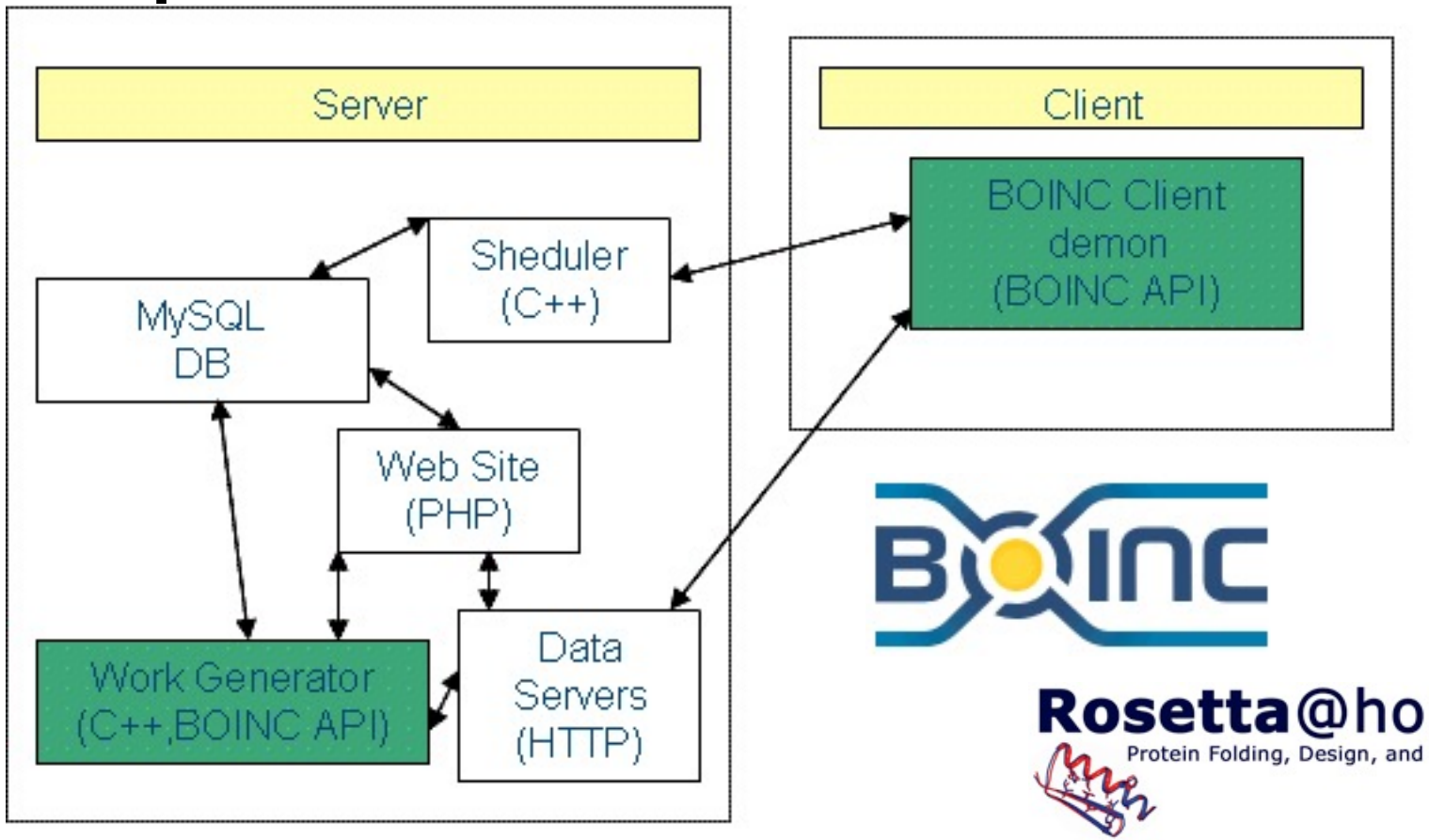
www.grid5000.org

- G5K has 5000 processors distributed in 9 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
- All clusters will be connected to Renater with a 10Gb/s link (or at least 1 Gb/s, when 10Gb/s is not available yet).
  - IntraCluster
    - Myrinet
    - GigaEthernet / Infiniband
  - Grid
    - Giga Ethernet (Best case 10GB/s, Nate case: 1GB/s)
  - Inter-Grid
    - Ethernet (~1GB/s)

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 "projects".
  - BOINC (Seti@home)
  - Xgrid
  - GridMP
- Associated with P2P
- Can be associated with High Throughput Computing (HTC) or High Performance Computing (HCP)

An Example: The BOINC Architecture

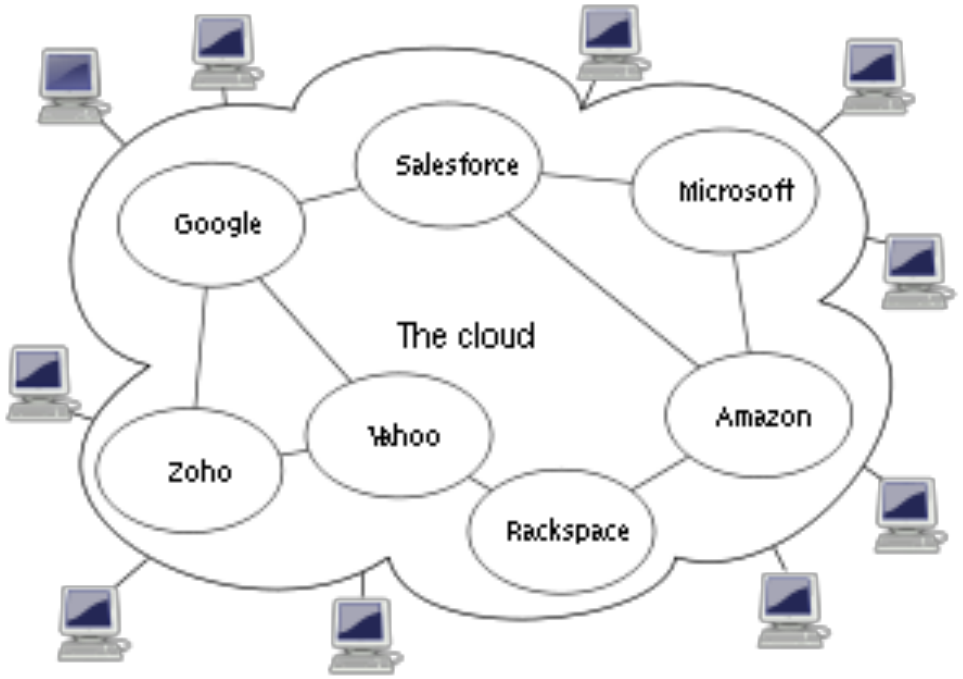


BOINC

Rosetta@home
Protein Folding, Design, and Docking

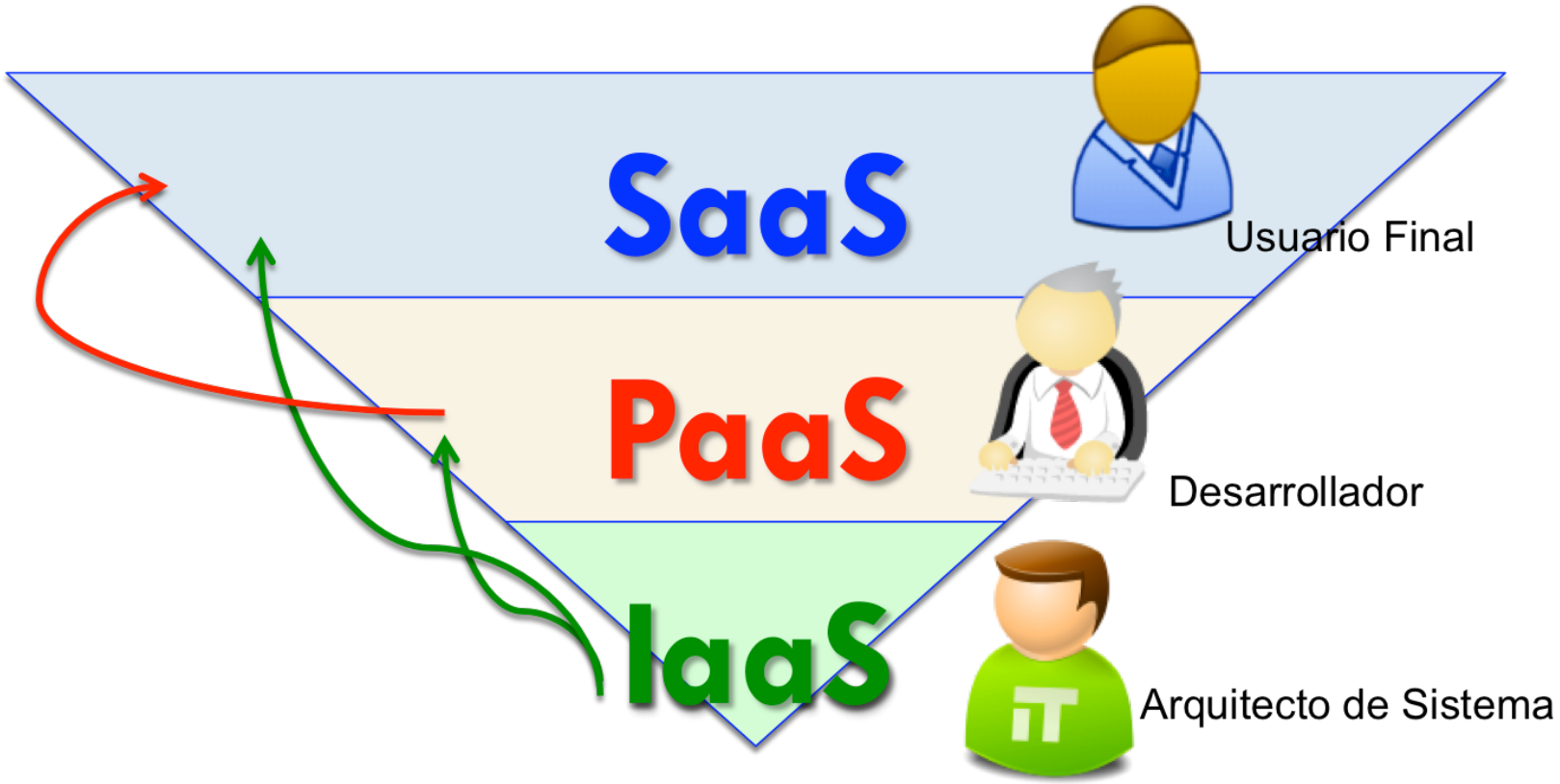
Cloud Computing

- Internet-based computing, whereby shared resources, software, and information are provided to computers and other devices on demand.
- Cloud computing describes a new supplement, consumption, and delivery model for IT services based on the Internet, and it typically involves over the-Internet-provision of dynamically scalable and often virtualized resources



Logical-Services Cloud View

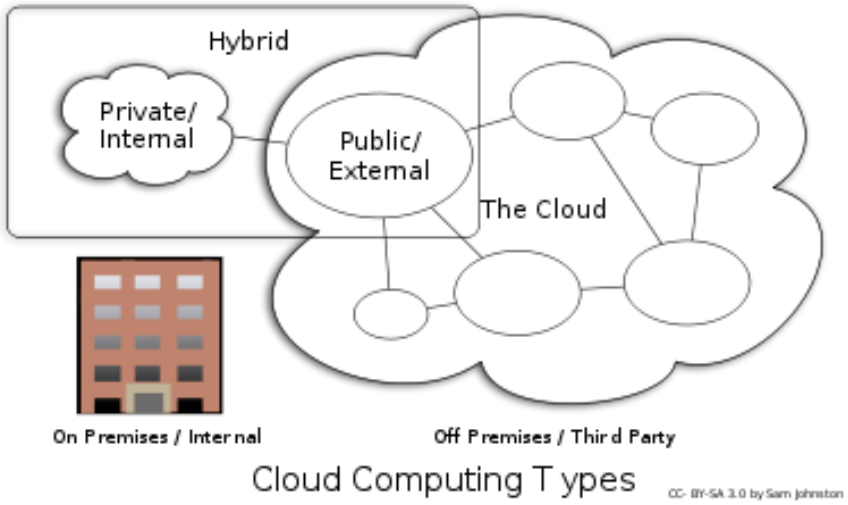
Cloud Computing Visibility



Visit: http://prezi.com/i0sretldeyk7/computacion-en-la-nube-y-sus-implicaciones-para-la-industria-del-software-en-colombia/

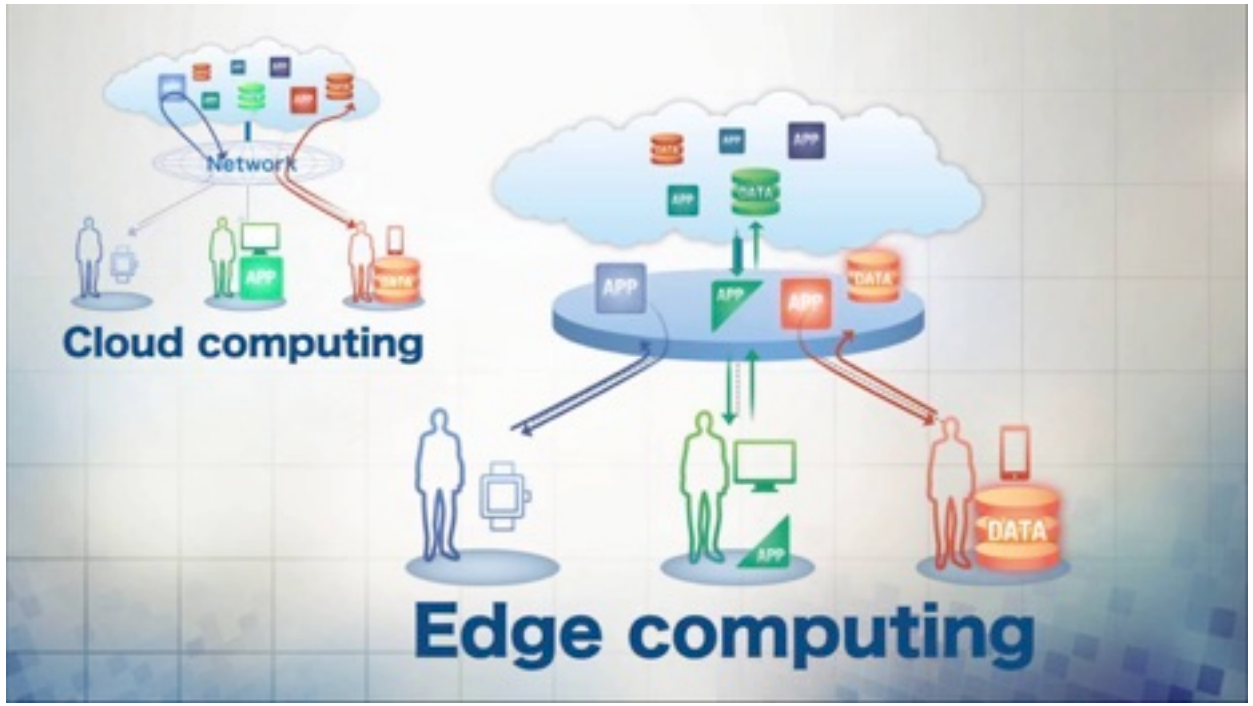
Cloud Computing Deployment Types

- Private Cloud
- Public Cloud
  - Resources are dynamically provisioned on a fine-grained, self-service basis over the Internet, via web applications/web services
- Community Cloud
  - Established where several organizations have similar requirements and seek to share infrastructure
- Hybrid Cloud
- InterCloud
  - Cloud of Clouds

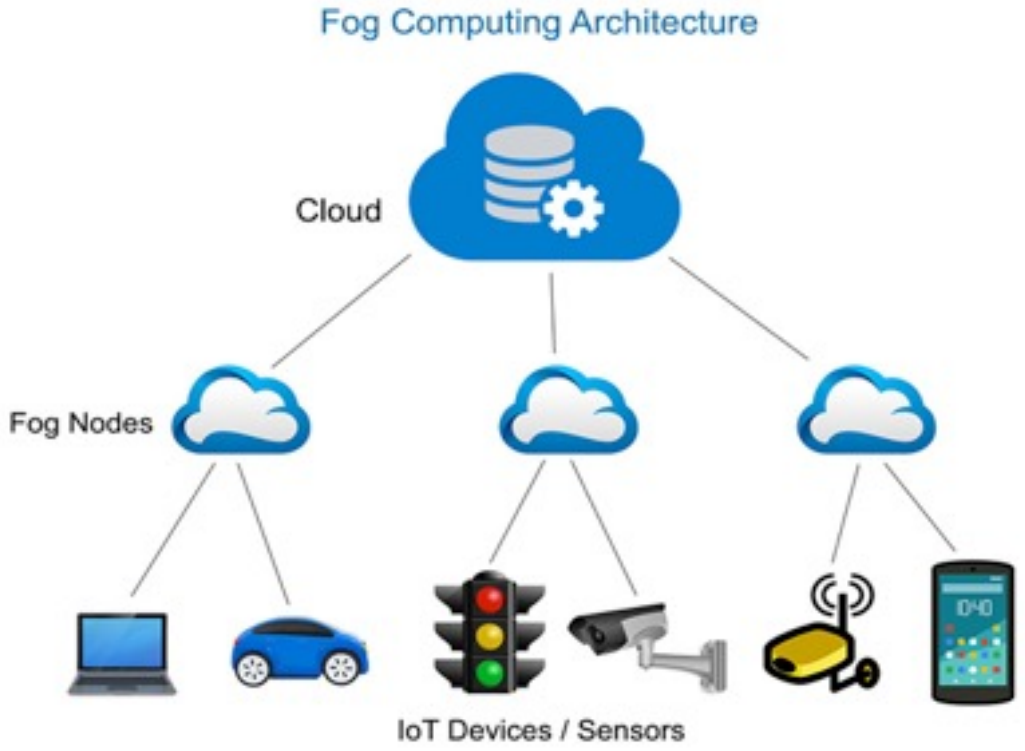


Different very known examples: AWS, MS Azure, Google Cloud..

UltraScale Systems



Mesh network of micro data centers that process or store critical data locally



Extends Cloud computing and services to the edge of the network

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.)

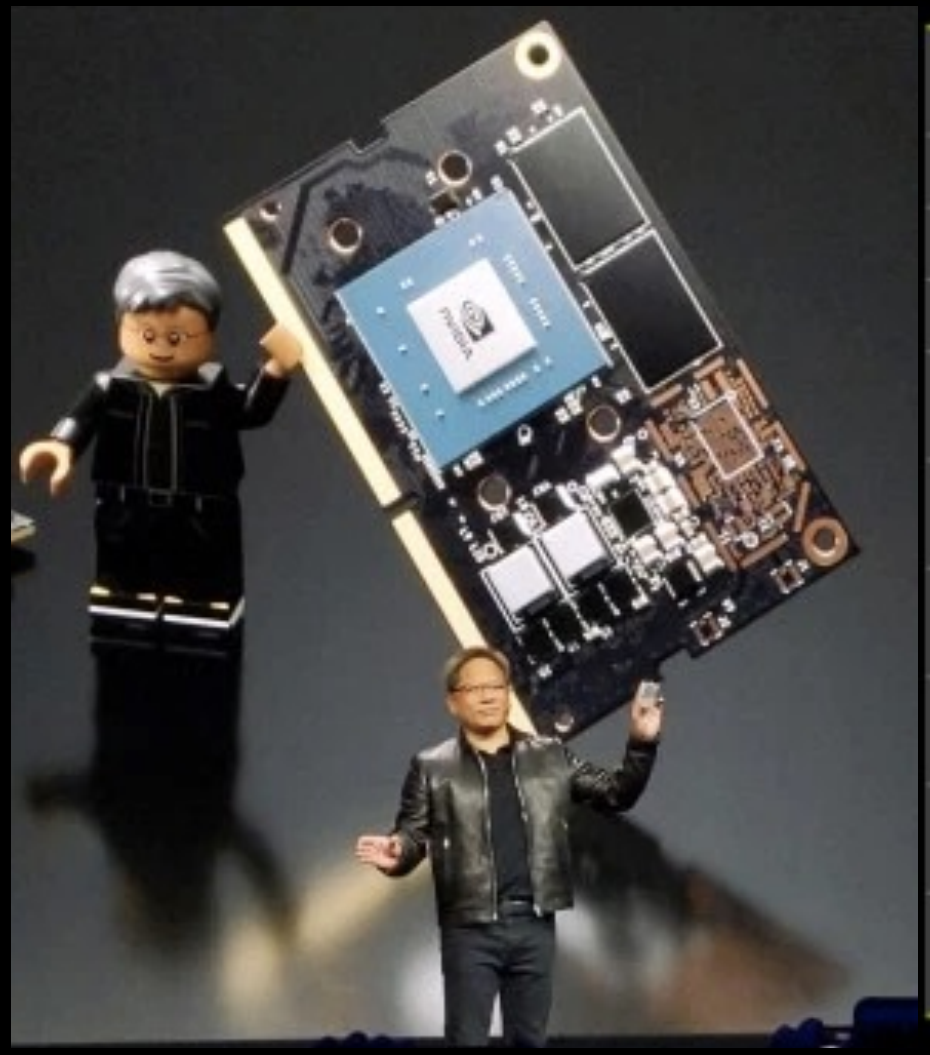
HPC Hybrid Systems (HPC@Pocket)

- 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

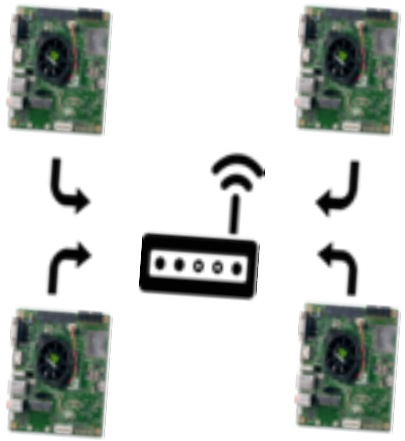
An Example: NVIDIA Jetson Nano

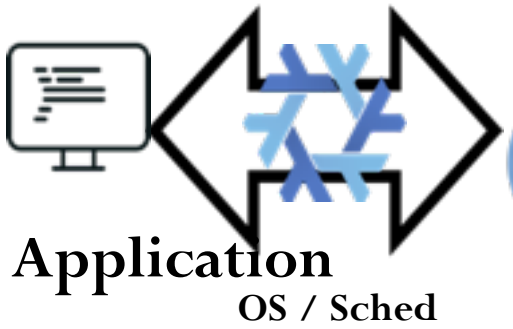
JETSON NANO SPECIFICATIONS

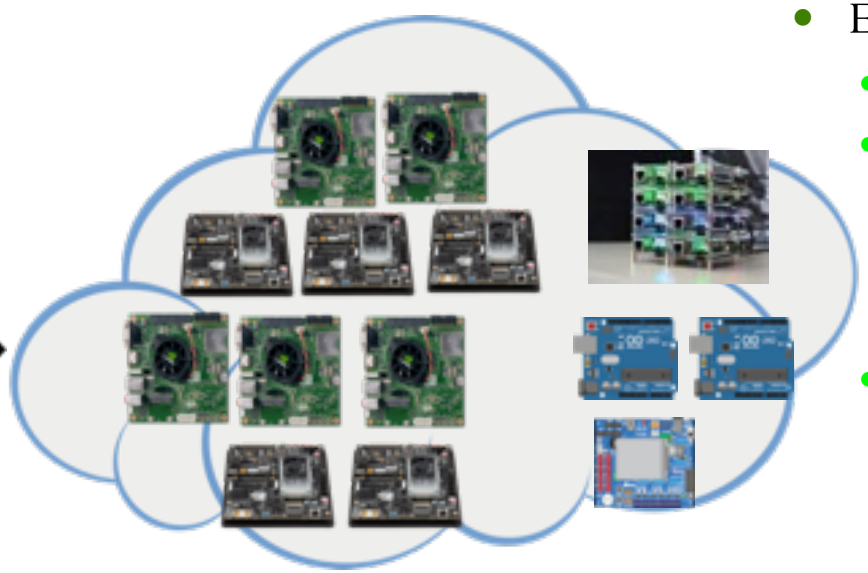


GPU	128 Core Maxwell 472 GFLOPs (FP16)
CPU	4 core ARM A57 @ 1.43 GHz
Memory	4 GB 64 bit LPDDR4 25.6 GB/s
Storage	16 GB eMMC
Video Encode	4K @ 30 | 4x 1080p @ 30 | 8x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 16x 720p @ 30 | (H.264/H.265)
Camera	12 (3x4 or 4x2) MIPI CSI-2 DPHY 1.1 lanes (1.5 Gbps)
Display	HDMI 2.0 or DP1.2 | eDP 1.4 | DSI (1 x2) 2 simultaneous
UPHY	1 x1/2/4 PCIE 1 USB 3.0
SDIO/SPI/SysIOs/GPIOs/I2C	1x SDIO / 2x SPI / 5x SysIO / 13x GPIOs / 6x I2C

Integration and Interaction





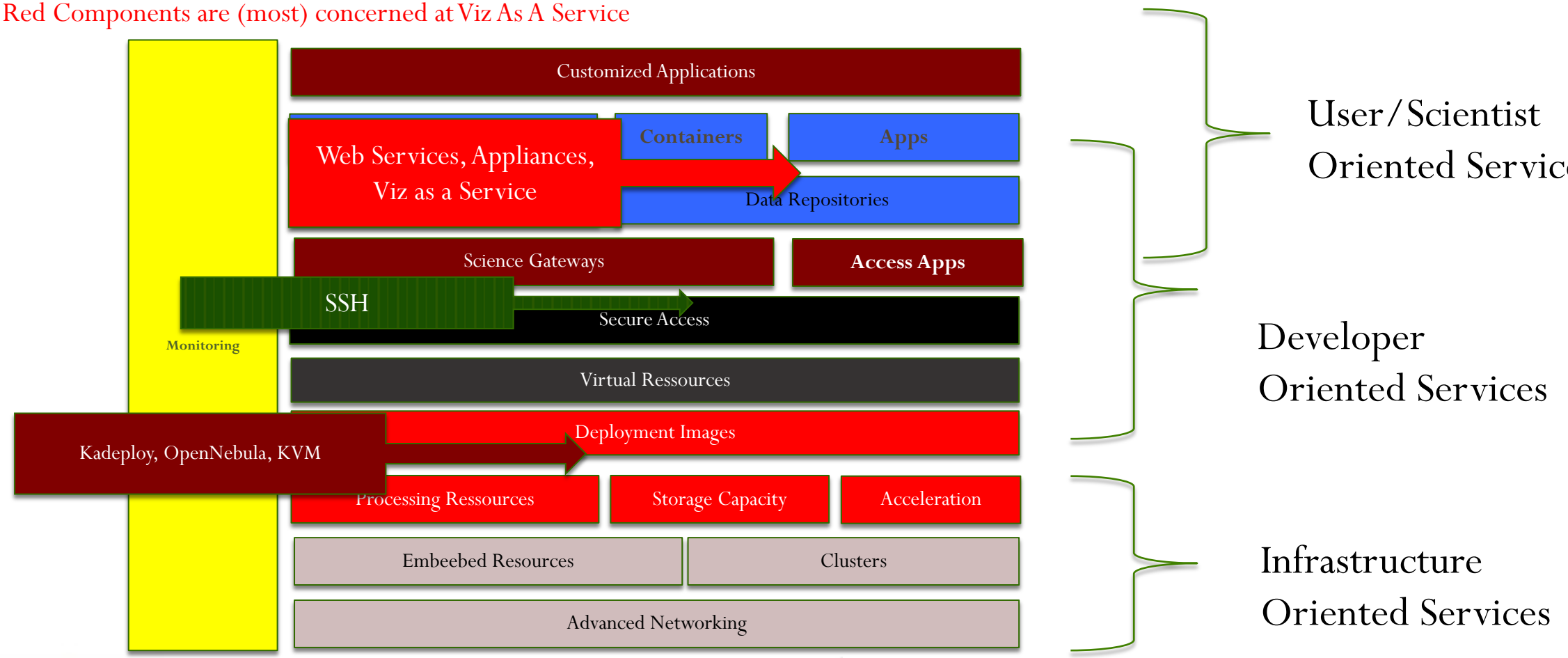


- Typical Protocols
  - TCP/UDP
  - Interaction with Large Scale Systems
- Redundancy
  - Availability
  - Easy Performance Monitoring
  - Fault Tolerance
- Embedded O.S. and Package Management
  - Scheduling Resources and Uses (i. e. NIX/NIX OS)
  - Containers
    - Fluidity (Data and Applications)
    - Micro-Architectures
    - Usability
  - Interaction with Large Scale Systems with Big Heterogeneous

Currently is the MSc These of Carlos Gomez (Co-Advising with O. Richard)

How Exploit HPC Architectures with Cloud Visibility Models?
HPC as A Service Model

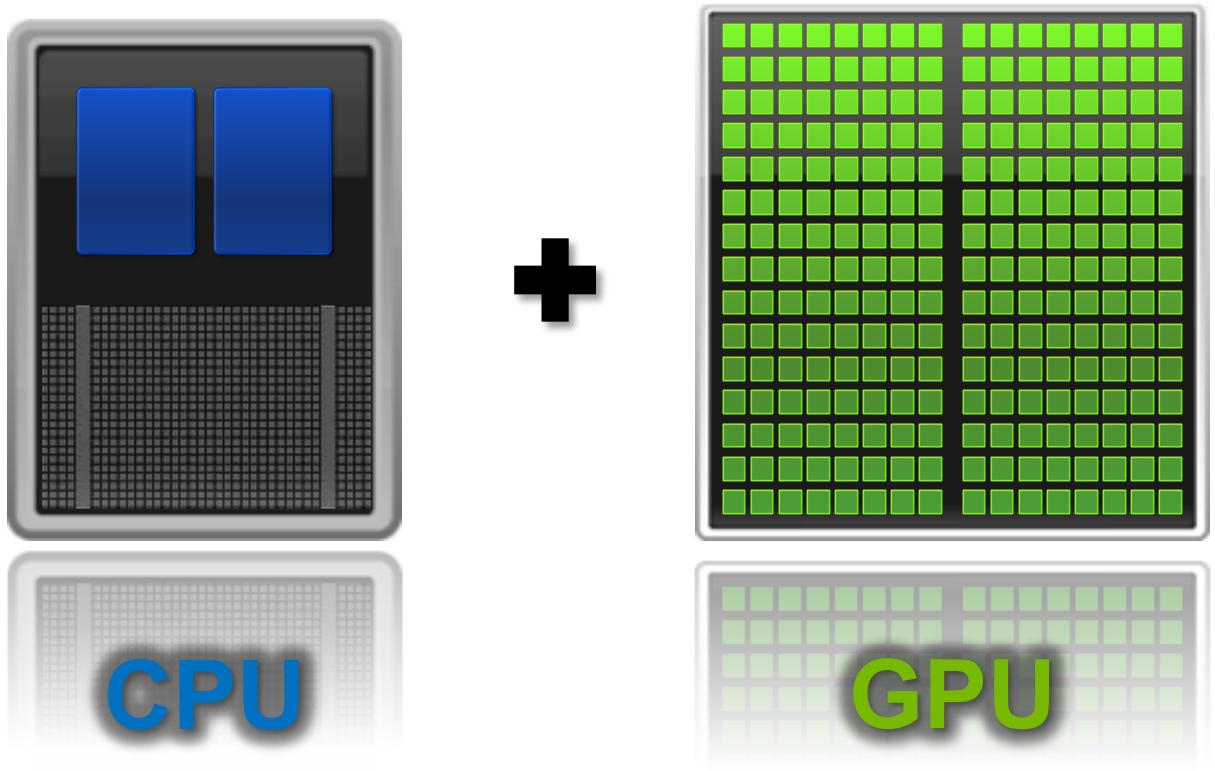
* Red Components are (most) concerned at Viz As A Service



Universidad Industrial de Santander
CONSTRUIMOS FUTURO
Super Computación y Cálculo Científico UIS
Universidad de los Andes
Inria
SCAAC
RICAP
NVIDIA

GPGPU Accelerate Computing Architecture

Latency Processor + Throughput processor



More Detailed and Explained in the Thursday Session)

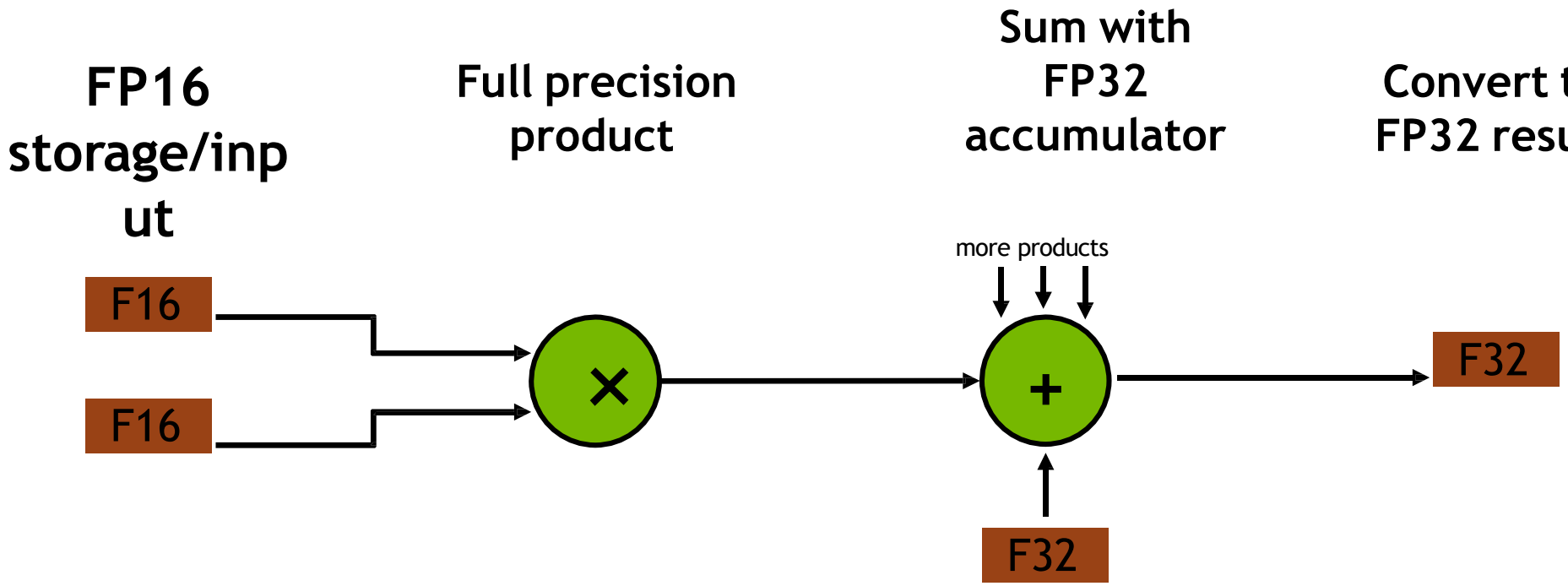
NVIDIA Tensor Cores Implementation Architecture

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

FP16 or FP32 — FP16 — FP16 — FP16 or FP32

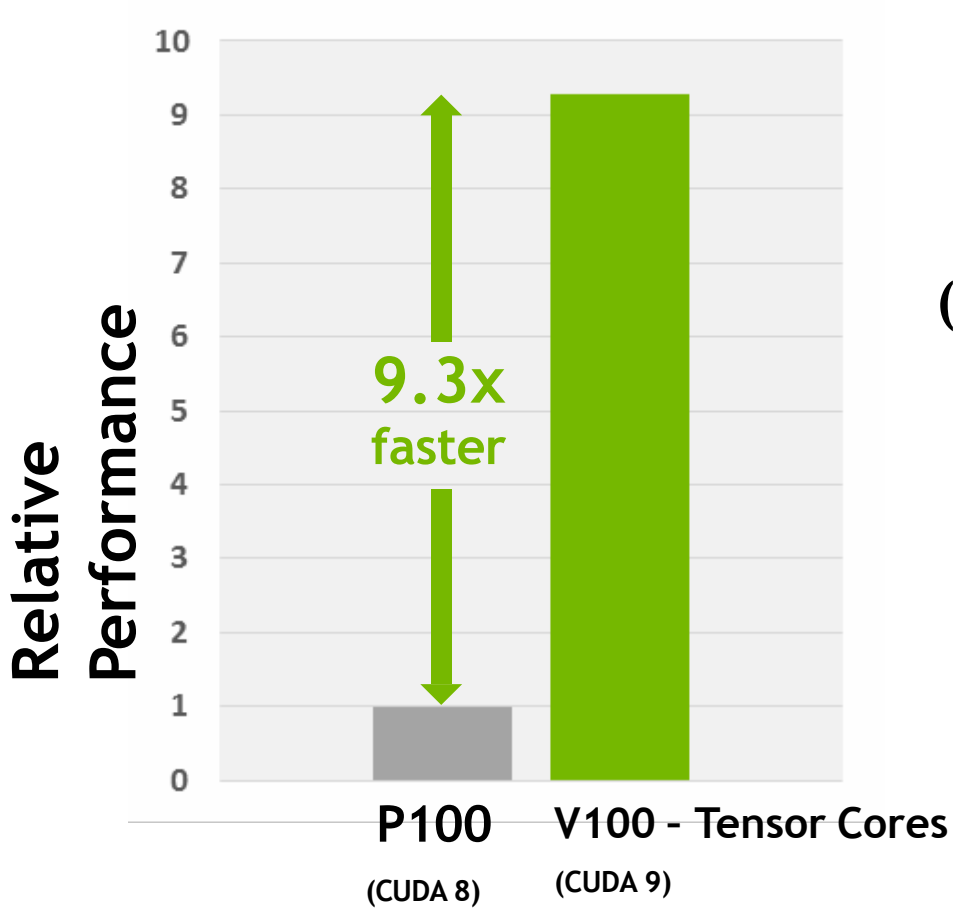
$$D = AB + C$$

VOLTA TENSOR OPERATION



Also supports FP16 accumulator mode for inferencing

Deep Learning Accomplished Promises



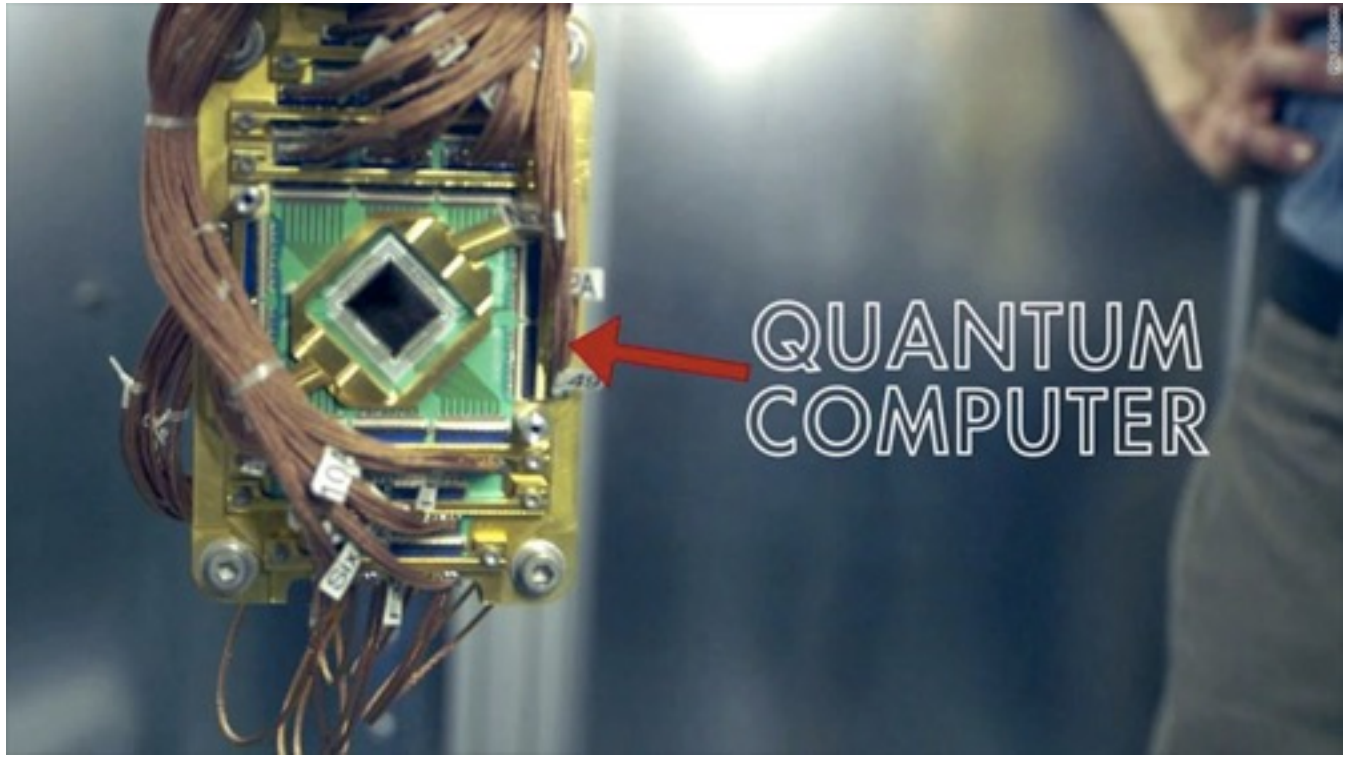
cuBLAS Mixed Precision
(FP16 input, FP32 compute)
Matrix Multiply (M=N=K=2048)

V100 measured on pre-production hardware.

And… Quantum computing?

(Advanced Computing Point of view)

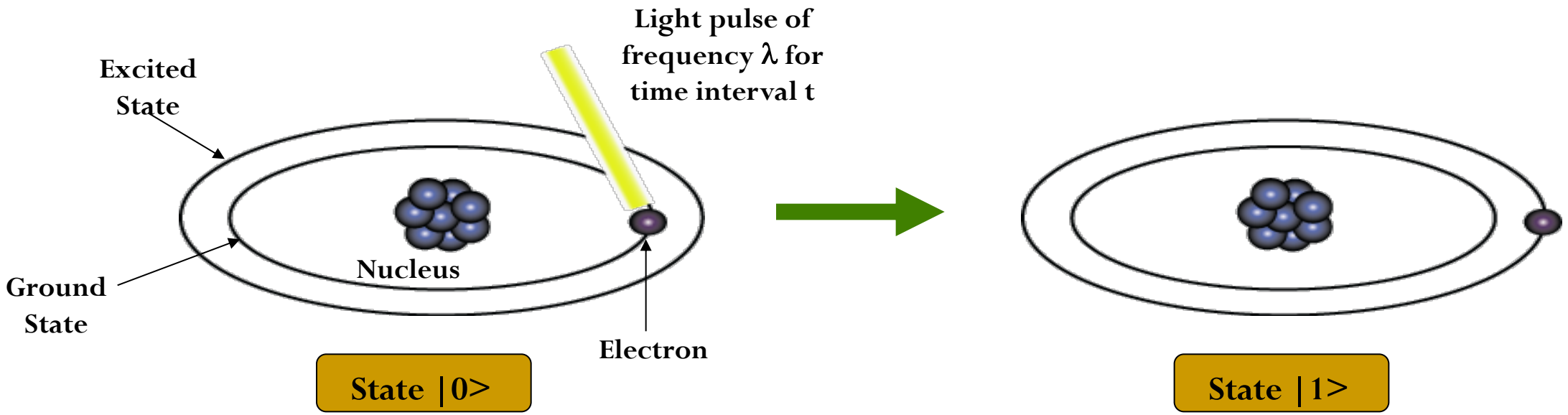
- A quantum computer is a machine that performs calculations based on the laws of quantum mechanics, which is the behavior of particles at the sub-atomic level.



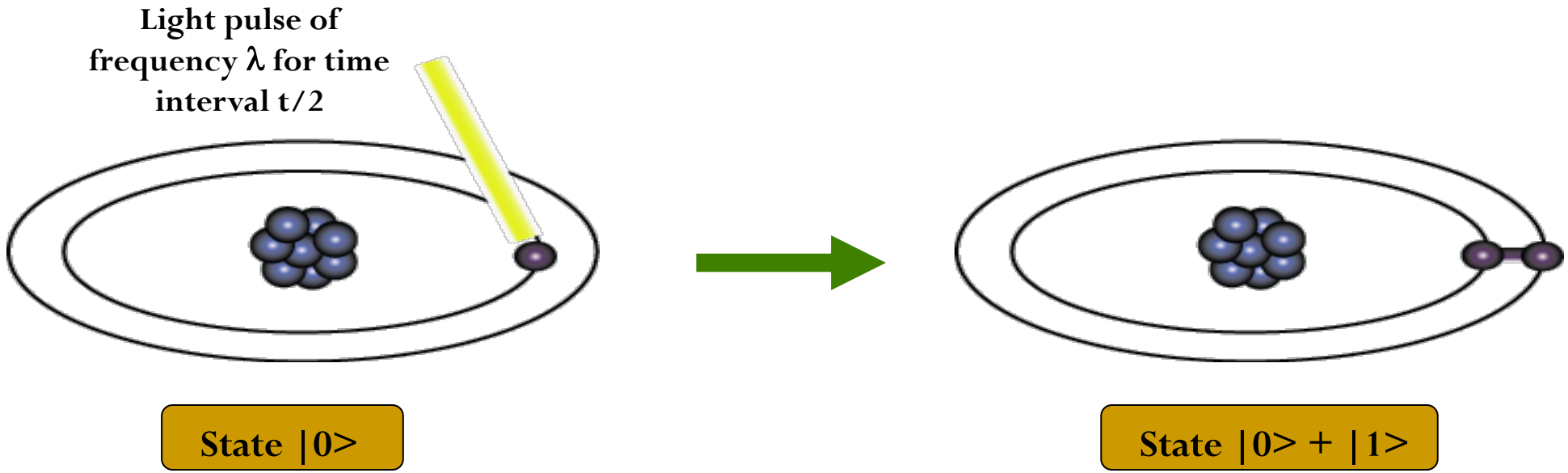
Representation of Data - Qubits

A bit of data is represented by a single atom that is in one of two states denoted by |0> and |1>. A single bit of this form is known as a ***qubit***

A physical implementation of a qubit could use the two energy levels of an atom. An excited state representing |1> and a ground state representing |0>.



Representation of Data - Superposition



- Consider a 3 bit qubit register. An equally weighted superposition of all possible states would be denoted by:

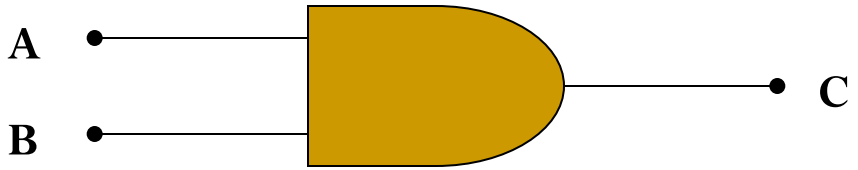
$$|\psi\rangle = \frac{1}{\sqrt{8}}|000\rangle + \frac{1}{\sqrt{8}}|001\rangle + \ldots + \frac{1}{\sqrt{8}}|111\rangle$$

Operations on Qubits - Reversible Logic

- Due to the nature of quantum physics, the destruction of information in a gate will cause heat to be evolved which can destroy the superposition of qubits.

Ex.

The AND Gate



Input		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

In these 3 cases, information is being destroyed

- This type of gate cannot be used. We must use ***Quantum Gates***.

Quantum Computers Today

- Enterprises produce Quantum Computing Laboratory Infrastructure (Non for production)
  - (Real) Quantum Computing (D-Wave, IBM)
  - Quantum Computing Simulators (Atos)

Quantum Computing Tomorrow

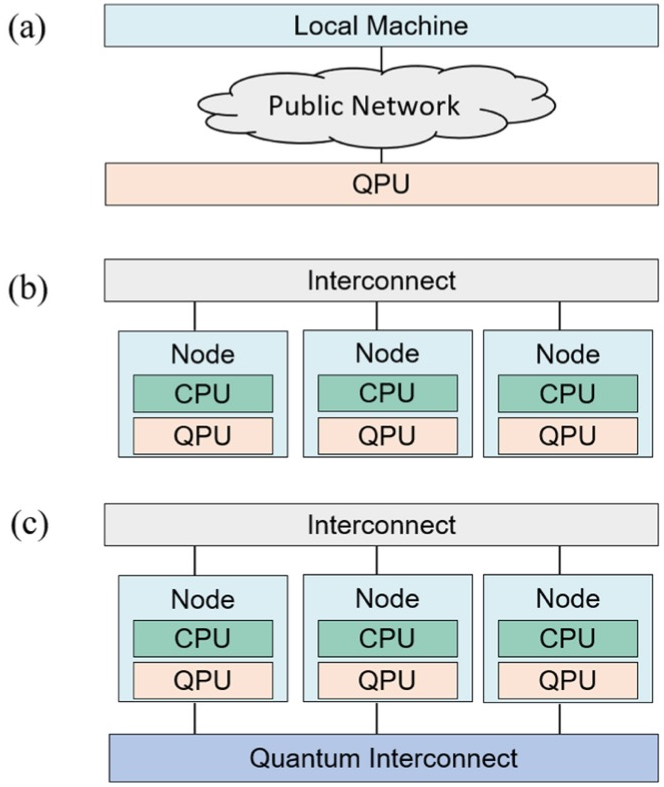


FIGURE 2. Three macroarchitectures for integrating quantum computing with conventional computing. (a) A local machine remotely accesses a QPU through public cloud network. (b) A network of quantum-accelerated nodes communicate through a common interconnect. (c) A network of quantum-accelerated nodes communicate through both conventional and quantum networks.

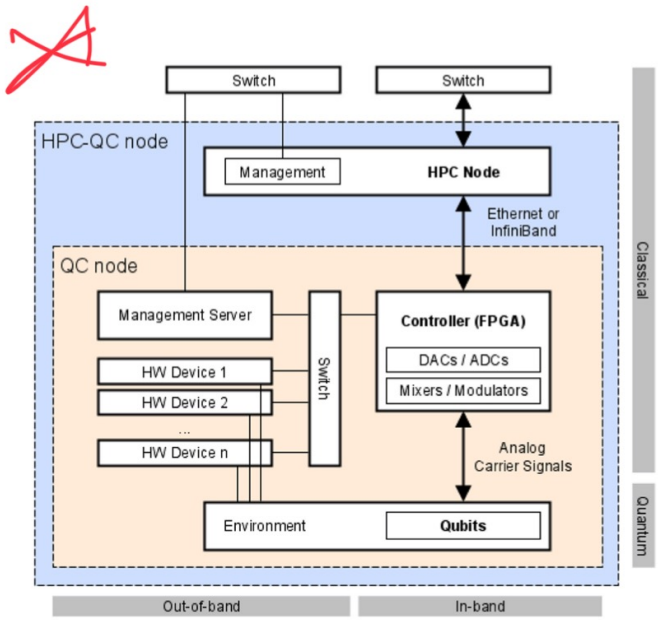


FIGURE 3. A component diagram representing the microarchitecture of a HPC-QC node with a common interconnect as depicted in Figure 2(b). The diagram shows the major components needed for the operation of a QPU within the HPC node infrastructure. Individual components are grouped into so-called out-of-band and in-band scopes and are placed on the left-hand and right-hand side of the figure, respectively. The QPU, which contains the qubits and is capable of processing quantum information, is depicted at the lower part, whereas classical information processing components are shown in the upper part of the figure. Several hardware (HW) devices control the QPU environment, which has a direct effect on qubit properties and thus the quality of execution of instructions.

DEPARTMENT: EXPERT OPINION

Quantum Computers for High-Performance Computing

Travis S. Humble, Alexander McCaskey, Dmitry I. Lyakh, and Meenambika Gowrishankar, *Quantum Computing Institute, Oak Ridge National Laboratory, Oak Ridge, TN, 37830, USA*
Albert Frisch, *Alpine Quantum Technologies, Innsbruck, 6020, Austria*
Thomas Monz, *Alpine Quantum Technologies, Innsbruck, 6020, Austria and also Institut für Experimentalphysik, Universität Innsbruck, Innsbruck, 6020, Austria*

Quantum computing systems are developing rapidly as powerful solvers for a variety of real-world calculations. Traditionally, many of these same applications are solved using conventional high-performance computing (HPC) systems, which have progressed sharply through decades of hardware and software improvements. Here, we present a perspective on the motivations and challenges of pairing quantum computing systems with modern HPC infrastructure. We outline considerations and requirements for the use cases, macroarchitecture, microarchitecture, and programming models needed to integrate near-term quantum computers with HPC system, and we conclude with the expectation that such efforts are well within reach of current technology.

High-performance computing (HPC) systems define the pinnacle of modern computing by drawing on massively parallel processing. This leading paradigm for HPC often relies on specialized accelerators and highly tuned networks to optimize data movement and application performance, whereby many computational nodes are connected by high-bandwidth networks to support shared information processing tasks. Existing computational nodes also support highly concurrent execution with multithreaded processing, and technology trends indicate that future node designs will integrate heterogeneous processing paradigms that include conventional central processing units (CPUs), graphics processing units (GPUs), field-programmable gate arrays (FPGAs), and other specialized processors.[1] The components of these future computational nodes must be tightly integrated to balance data movement with processing power and workload in order to optimize overall system performance.

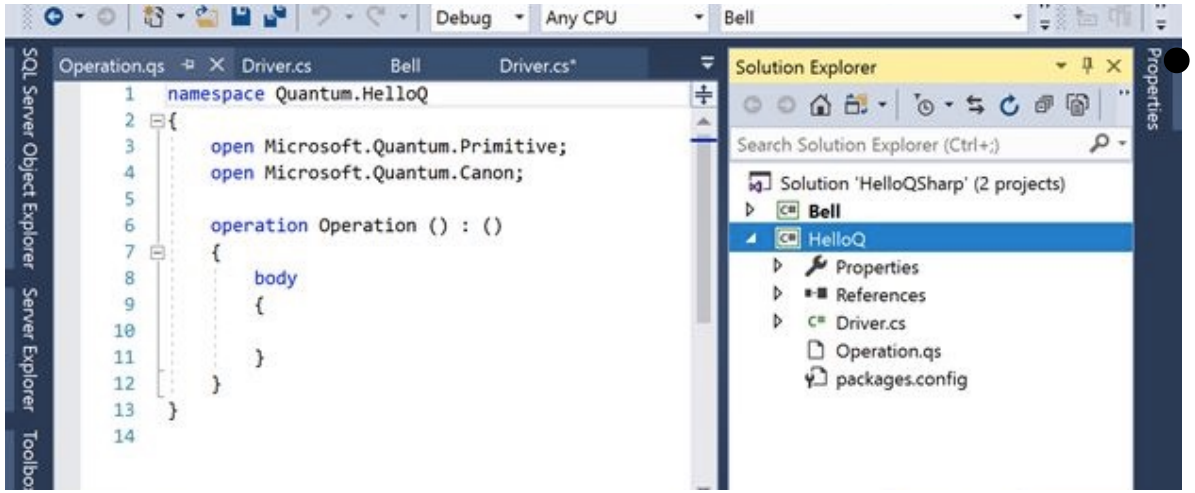
By comparison, quantum computers (QCs) represent a young yet remarkable advance in the science and technology of computation that are often cited as rivals or successors to state-of-the-art conventional high-performance computing (HPC) systems. The source of this proposed advantage of QCs derives from quantum information processing in which information is encoded in the quantum state of physical systems such as atoms, electrons, and photons.[2] These quantum physical systems present the unique features of quantum coherence and quantum entanglement that permit quantum computing to reduce exponentially the computational time and memory needed to solve many problems from chemistry, materials science, finance, and cryptanalysis among other application domains. The advantage afforded to quantum computing is therefore aptly named the "quantum computational advantage," and there is now a fervent effort to realize quantum computing systems that demonstrate this advantage. Notably, recent efforts have focused on besting the world's leading HPC systems to great effect.[3,4,5]

Many of the most promising applications of quantum computing overlap strongly with existing applications of HPC,[6] which begs the question of how QCs may be integrated with modern HPC to accelerate these

Please see the Acknowledgements section at the end of the article for a special statement regarding the copyright.

0272-1732 © 2021 IEEE
Digital Object Identifier 10.1109/MM.2021.3099140
Date of current version 14 September 2021.

About Q… Algorithms and Code…



Q# Interface

- Consortiums propose Quantum Frameworks
  - IBM, Microsoft, ATOS
  - Quantum Computing Community…
    - However without a good use fo real Quantum mathematical abstraction.

And the software …

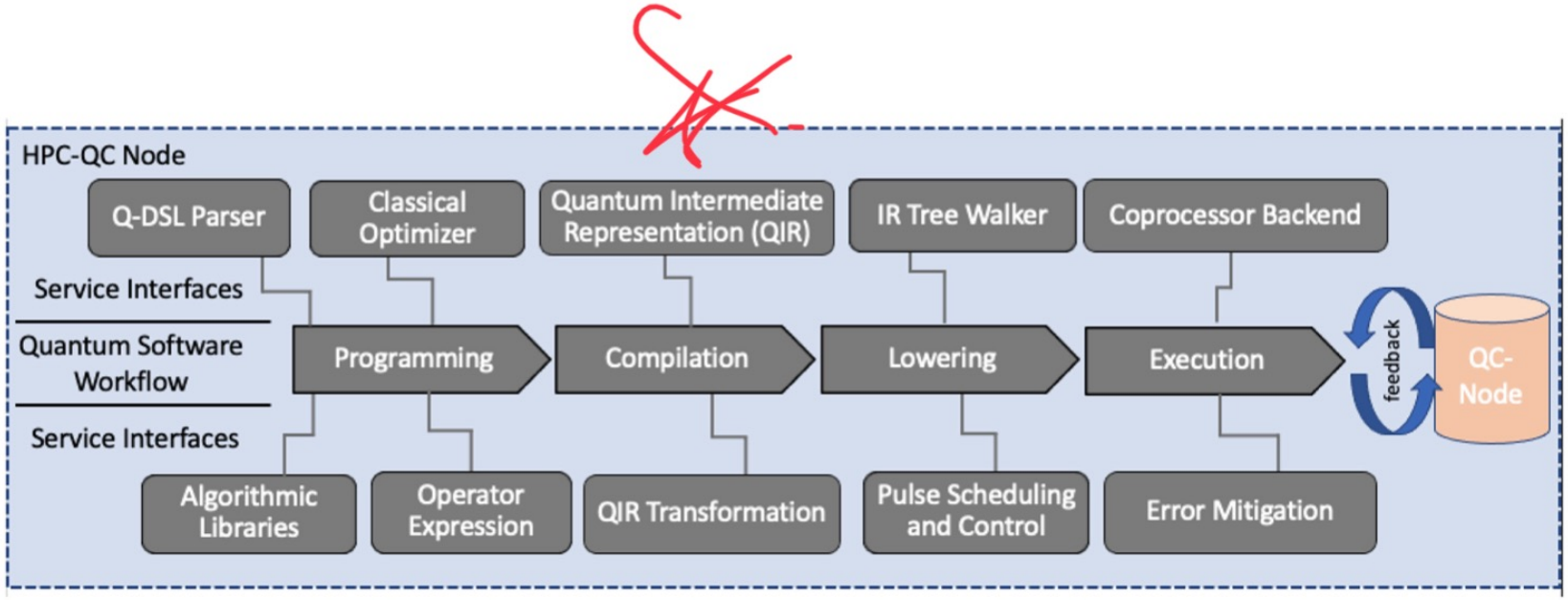
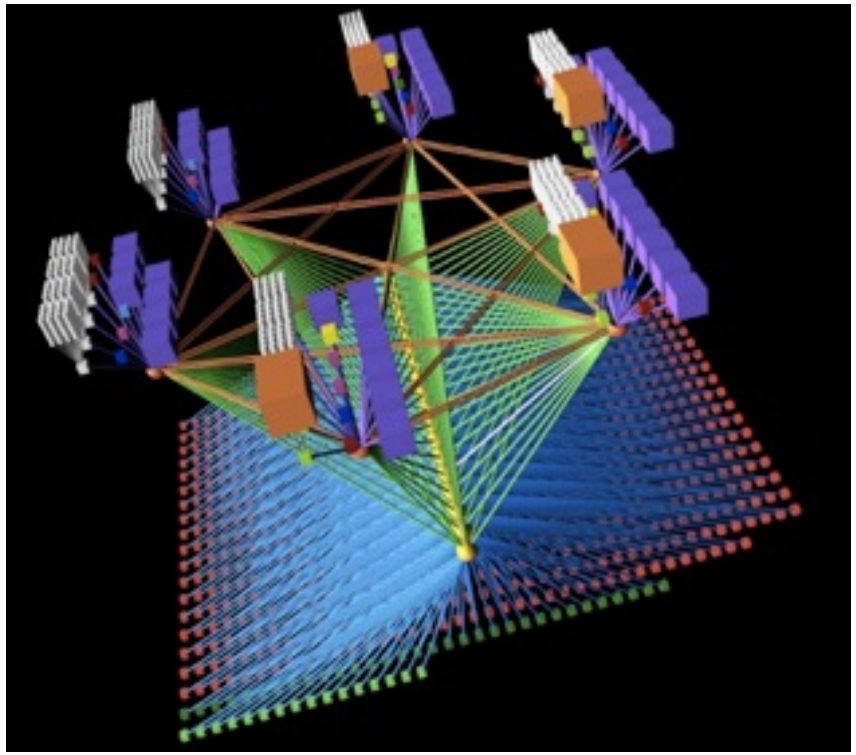


FIGURE 4. Decomposition of the software architecture required for quantum-HPC integration into a series of workflow steps, each exposing a unique set of service interfaces. This architecture maximizes the flexibility, modularity, and extensibility of the suggested integration strategy. Here we show the workflow decomposed into programming, compilation, IR lowering, and execution components. Each component exposes a series of service interfaces intended for the implementation of concrete use cases.

Then..... The challenges (Are now for computing people not for physicists)

- A « Real » Abstraction of Quantum Architecture
  - Quantum Memory (Optimized « in-Memory » System)
  - Software/Application Elements
    - Definition of a Language with the « Assembly » possibilities
    - The Concept of a Operating System
- The « Production » Applications
  - Quantum Algorithms
  - Quantum Application Design and Code Structure
  - The Concept of Optimization (and compilation) in Quantum Computing

The Challenges in Detail: Post Moore Era Architectures



A post-moore architecture schema

- Sustainable-Hybrid Technology
  - RISC/CISC
    - GPUs, Hybrid ARM/FPGAs, Accelerators, CPUS….
    - I/O's and Memory Management
- The "Data Treatment" Goal
  - Large Scale Data Sets (Supported by the Architecture
  - However scale capabilities changes
  - In-Situ and In-Transit Problem
- Very Known Schedulers, O.S. and Package Management
  - However, it is important to observe the architecture
- Exascale constrains
  - Computer Efficiency (Energy Consumption / Energy Aware)

The Software/Applications Approach

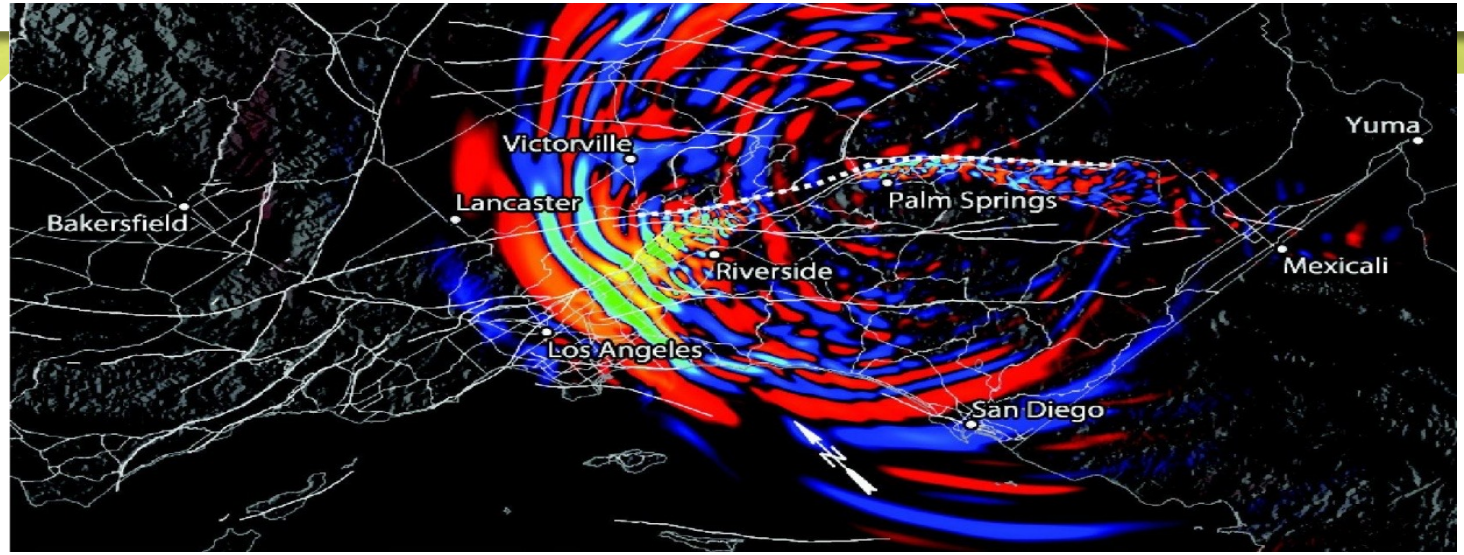
About High Performance Computing

- HPC is useful to being faster, more precise overall, to solve large problems and to treat, intrinsically, parallelism in essence.
- However allows
  - Technological Advantage
  - Technological Independency
  - Competitively
  - Energy Savings
- But, HPC is expensive

What & Why

- **What is high performance computing (HPC) from Parallel Programming Approach?**
  - The use of the most efficient algorithms on computers capable of the highest performance to solve the most demanding problems.
- **Why HPC?**
  - Large problems – spatially/temporally
    - 10,000 x 10,000 x 10,000 grid → 10^12 grid points → 4x10^12 double variables → 32x10^12 bytes = 32 Tera-Bytes.
    - Usually need to simulate tens of millions of time steps.
    - On-demand/urgent computing; real-time computing;
  - Weather forecasting; protein folding; turbulence simulations/CFD; aerospace structures; Full-body simulation/ Digital human …
  - And Remember the slides 2 and 3…

HPC Examples

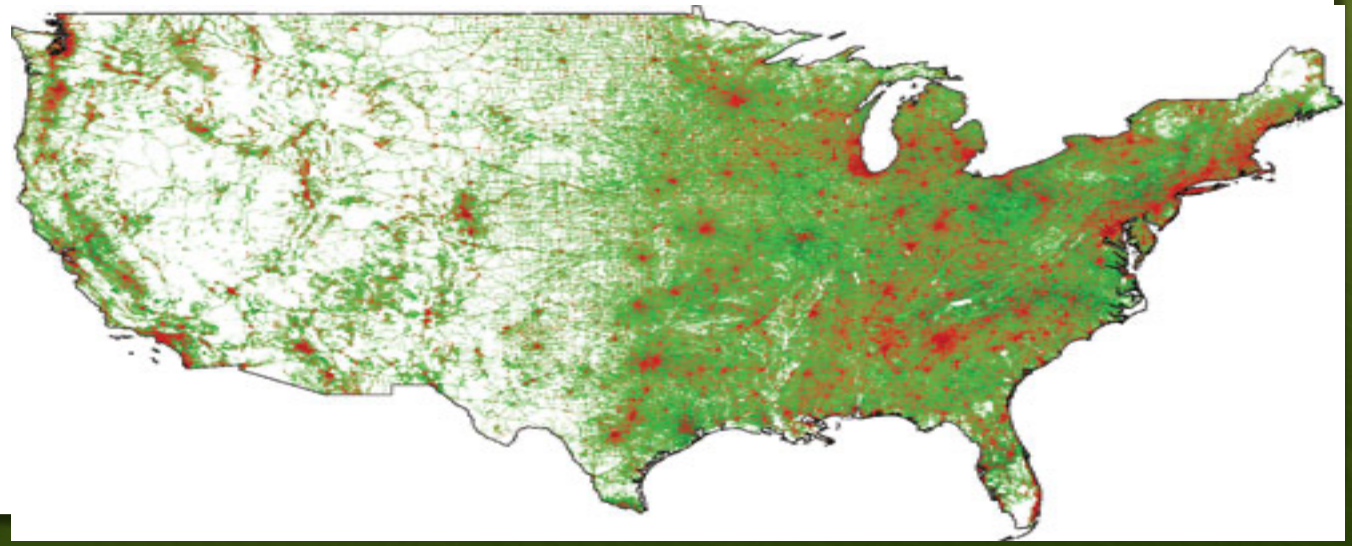


Earthquake simulation

Surface velocity 75 sec after earthquake

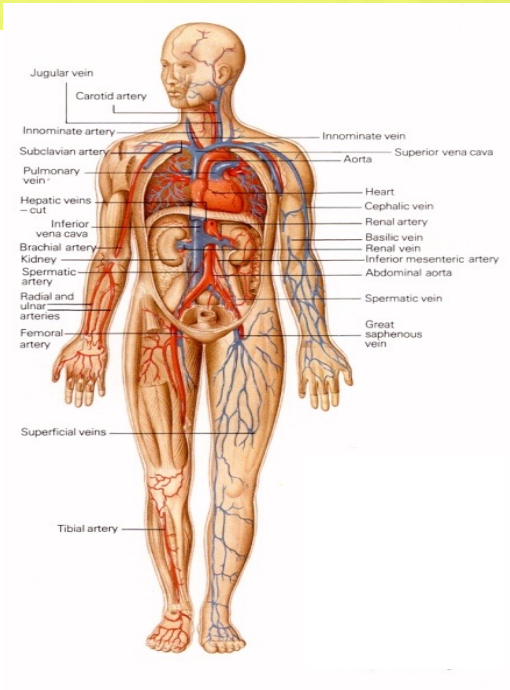
Flu pandemic simulation
300 million people tracked

Density of infected population, 45 days after breakout



HPC Examples: Blood Flow in Human Vascular Network

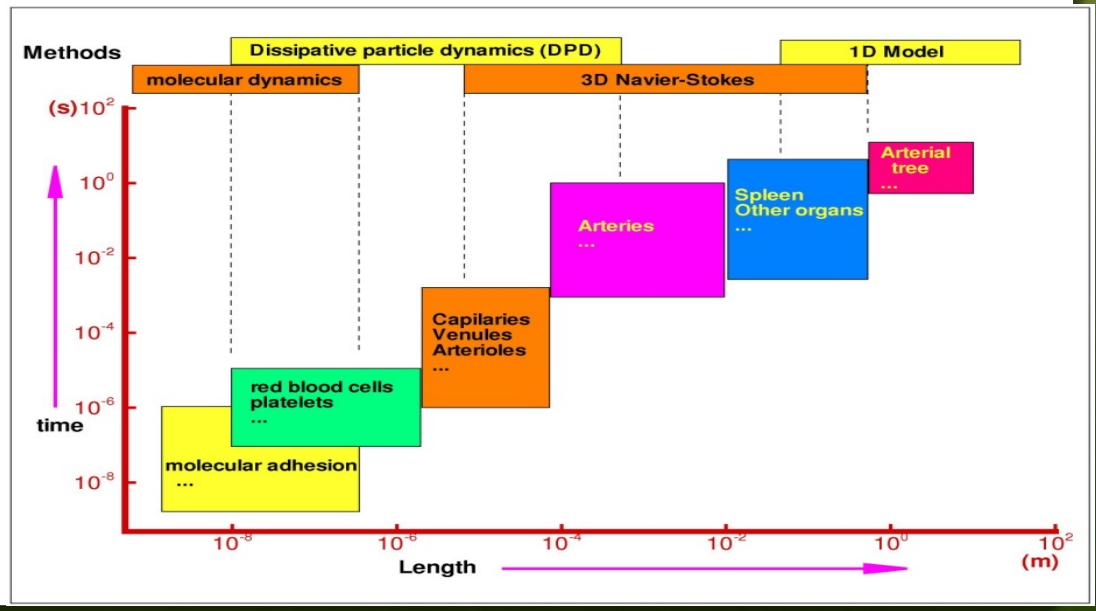
- Cardiovascular disease accounts for about 50% of deaths in western world;
- Formation of arterial disease strongly correlated to blood flow patterns;



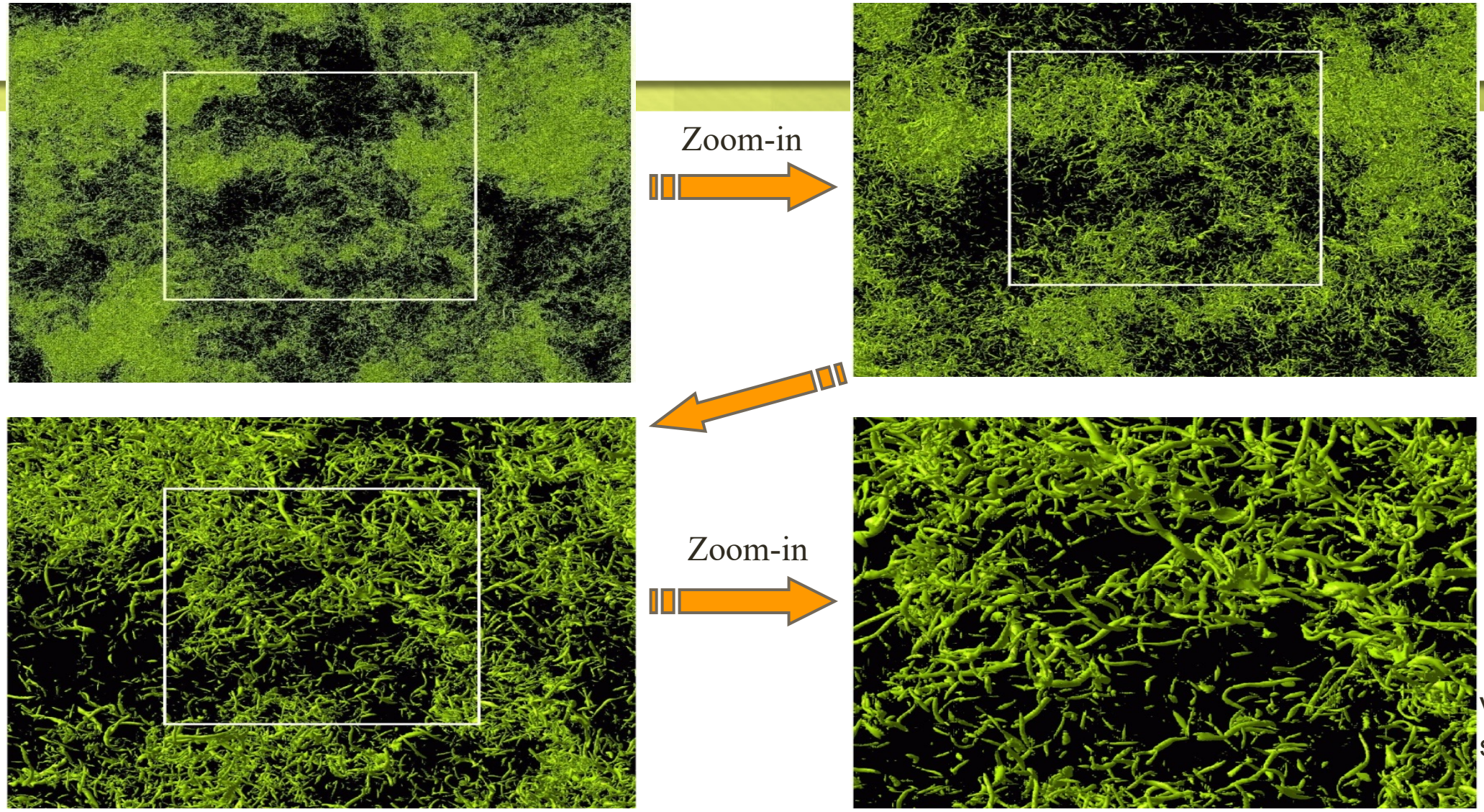
In one minute, the heart pumps the entire blood supply of 5 quarts through 60,000 miles of vessels, that is a quarter of the distance between the moon and the earth

Computational challenges: Enormous problem size

Blood flow involves multiple scales



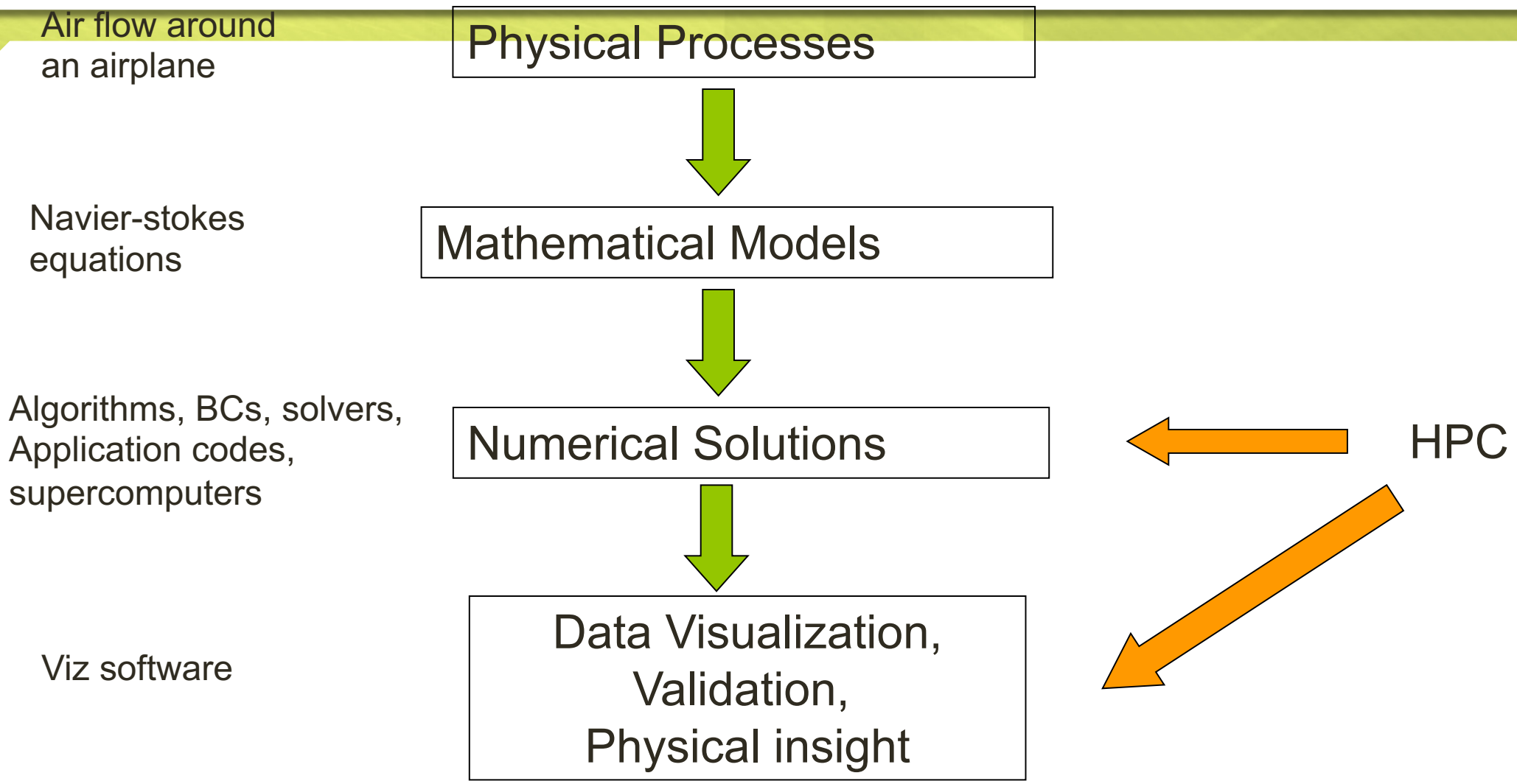
HPC Example: Homogeneous Turbulence



Vorticity iso-surface

Direct Numerical Simulation of Homogeneous Turbulence: 4096^3

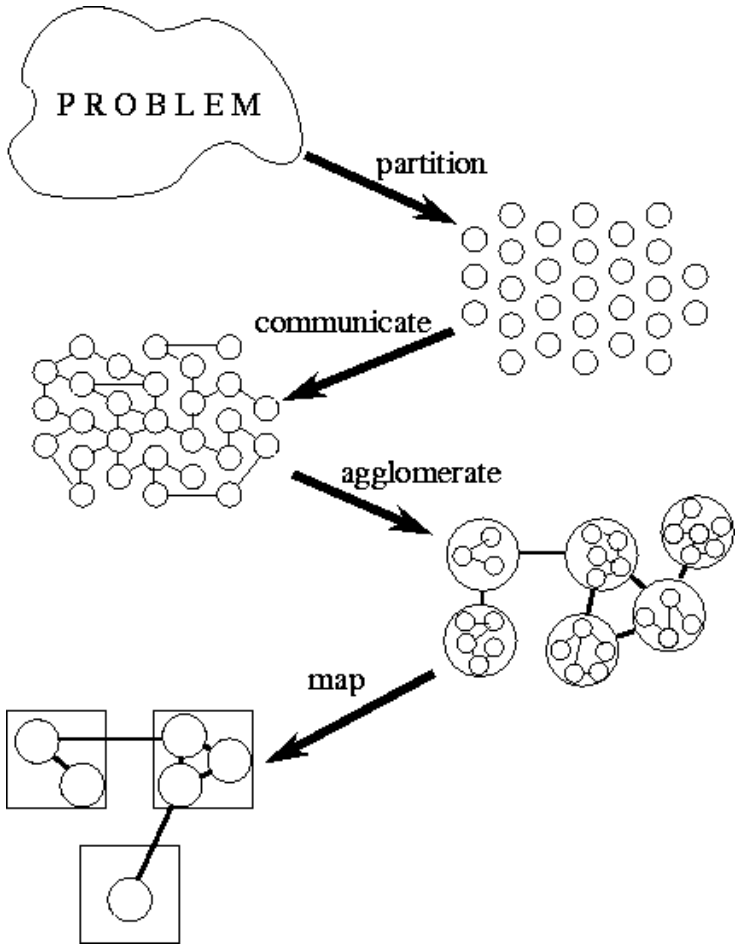
How HPC fits into Scientific Computing



Advantages of Parallelization

- Cheaper, in terms of Price/Performance Ratio
- Faster than equivalently expensive uniprocessor machines
- Handle bigger problems
- More scalable: the performance of a particular program may be improved by execution on a large machine
- More reliable: In theory if processors fail we can simply use others

How to Parallelize?: Traditional Way



Actually applied for current well-known applications with sequential implementations.

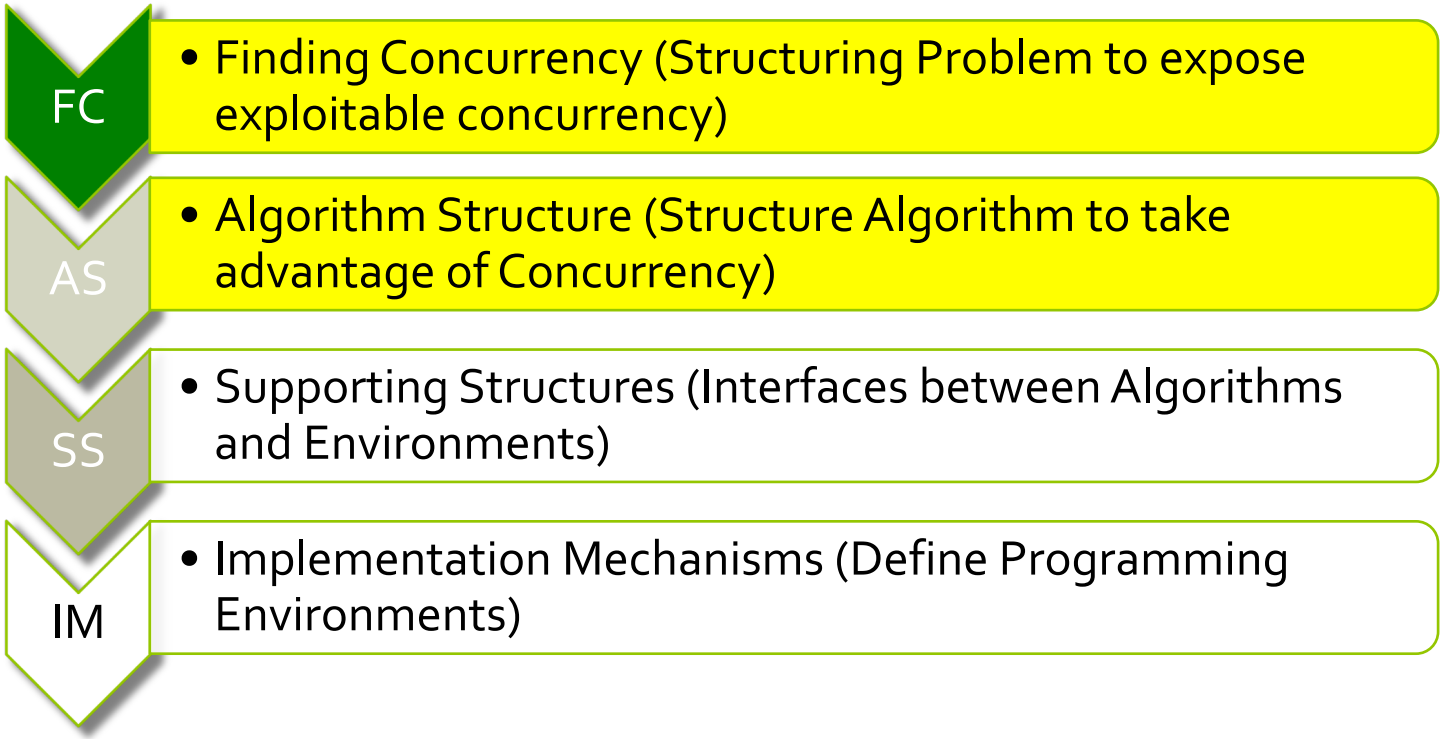
Addressed (mainly) for distributed memory applications

It's good as first approach of scientific computing algorithm for (alone) scientists programmers.

However this is not a traditional course…

Designing and Building Parallel Programs, by Ian Foster in http://www.mcs.anl.gov/~itf/dbpp/

Design Spaces of Parallel Programming*



•Patterns for Parallel Programming, Timoty Mattson, Beverly A. Sanders and Berna L. Massingill, Software Pattern Series, Addison-Wesley 2004

Concurrent Programming General Steps

1. **Analysis**
   - Identify Possible Concurrency
     - Hotspot: Any partition of the code that has a significant amount of activity
     - Time spent, Independence of the code…
2. **Design and Implementation**
   - Threading the algorithm
3. **Tests of Correctness**
   - Detecting and Fixing Threading Errors
4. **Tune of Performance**
   - Removing Performance Bottlenecks
     - Logical errors, contention, synchronization errors, imbalance, excessive overhead
     - Tuning Performance Problems in the code (tuning cycles)

- **From: Patterns for Parallel Programming**., by T. Mattson., B. Sanders and B. MassinGill (Ed. Addison Weslley, 2009) Web Site: http://www.cise.ufl.edu/research/ParallelPatterns/

Distributed Vs. Shared Memory Programming
(Remember Architecture Features)

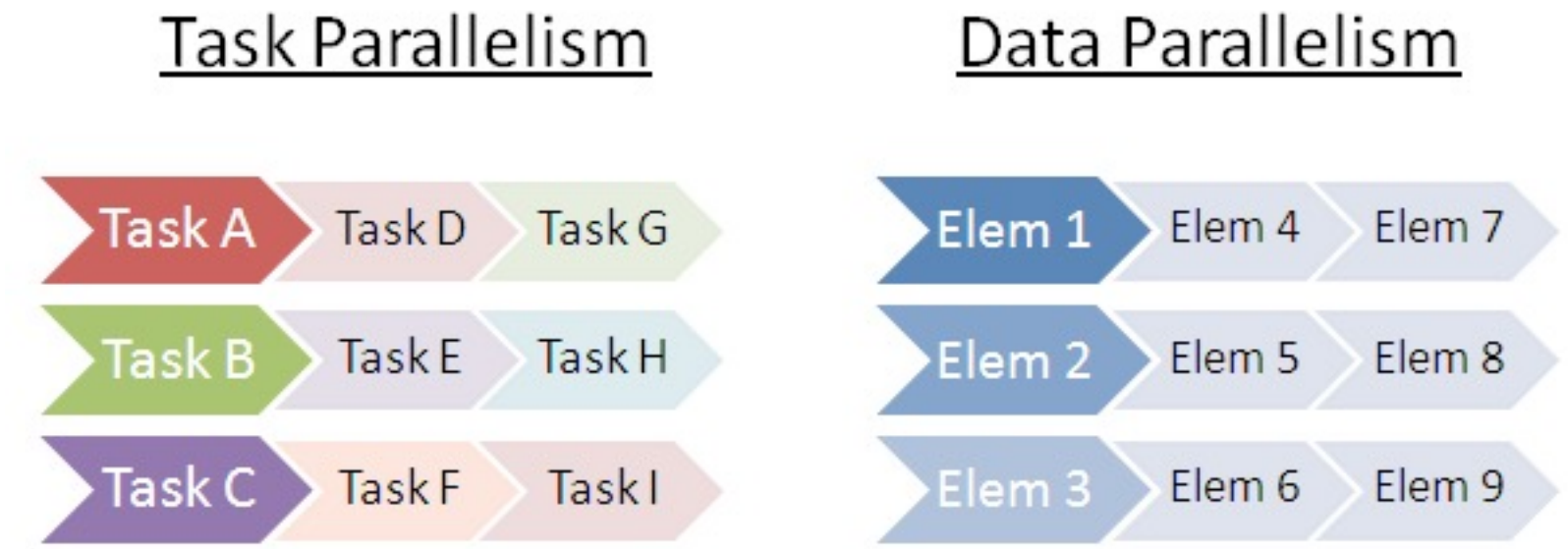
Common Features

- Redundant Work
- Dividing Work
- Sharing Data (Different Methods)
- Dynamic / Static Allocation of Work
  - Depending of the nature of serial algorithm, resulting concurrent version, number of threads / processors

Only to Shared Memory

- Local Declarations and Thread-Local Storage
- Memory Effects:
  - False Sharing
- Communication in Memory
- Mutual Exclusion
- Producer / Consumer Model
- Reader / Writer Locks (In Distributed Memory is Boss / Worker)

Decomposition



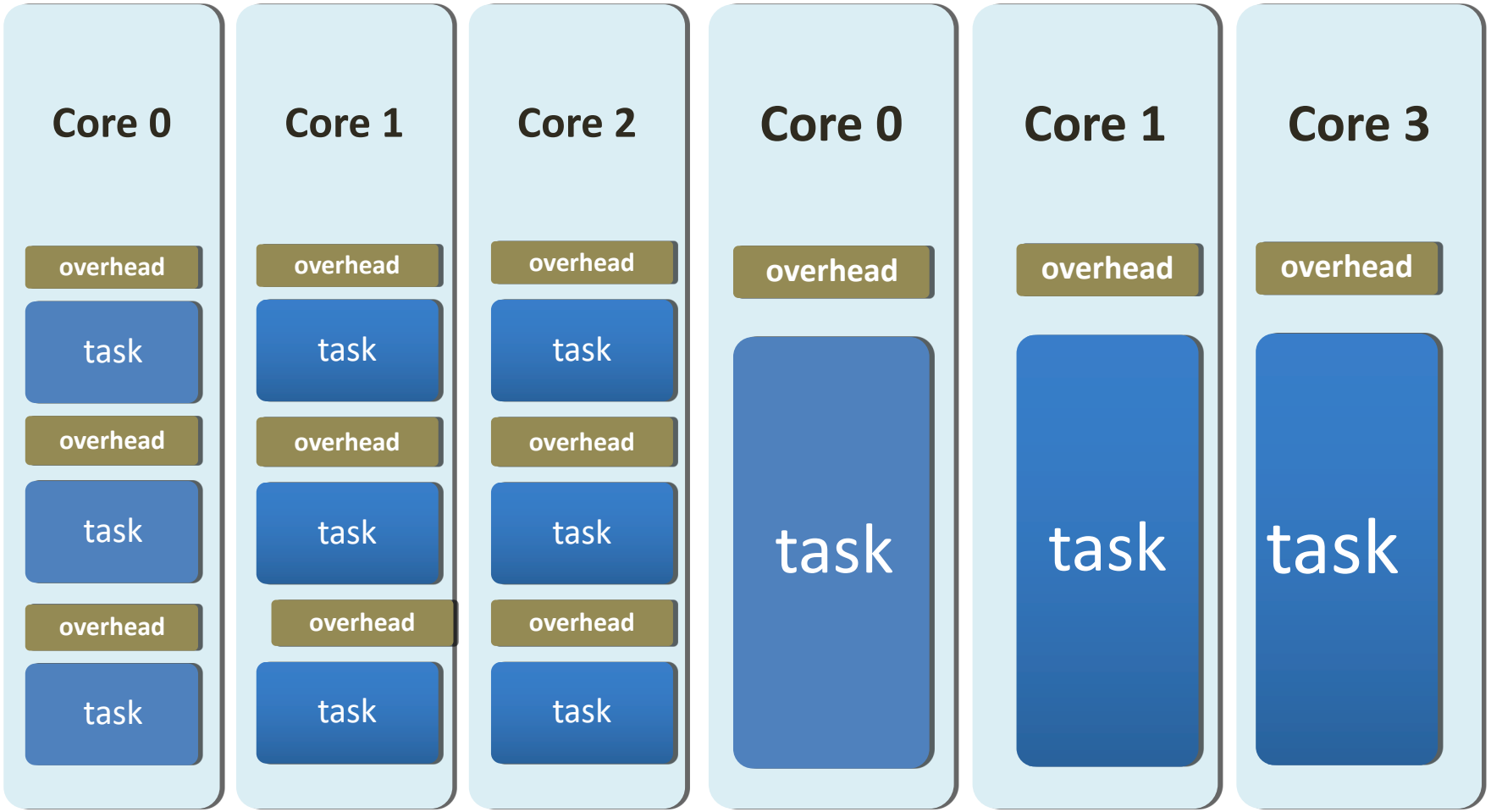
Tasks Decomposition : Task Parallelism
Data Decomposition: Data Parallelism /Geometric Parallelism

Task Parallelism : What are the tasks and how are defined?

- There should be at least as many tasks as there will be threads (or cores)
  - It is almost always better to have (many) more tasks than threads.
- **Granularity** must be large enough to offset the overhead that will be needed to manage the tasks and threads
  - More computation: higher granularity (coarse-grained)
  - Less Computation: lower granularity (fine-grained)

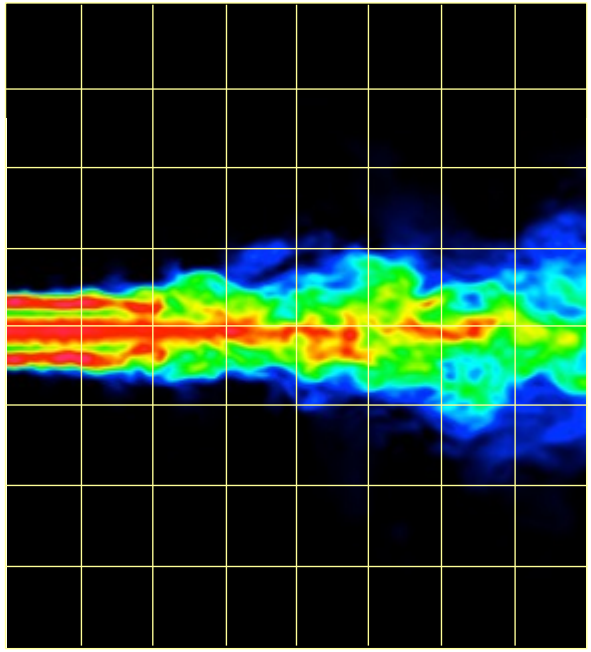
Granularity is the amount of computation done before synchronization is needed

Task Granularity



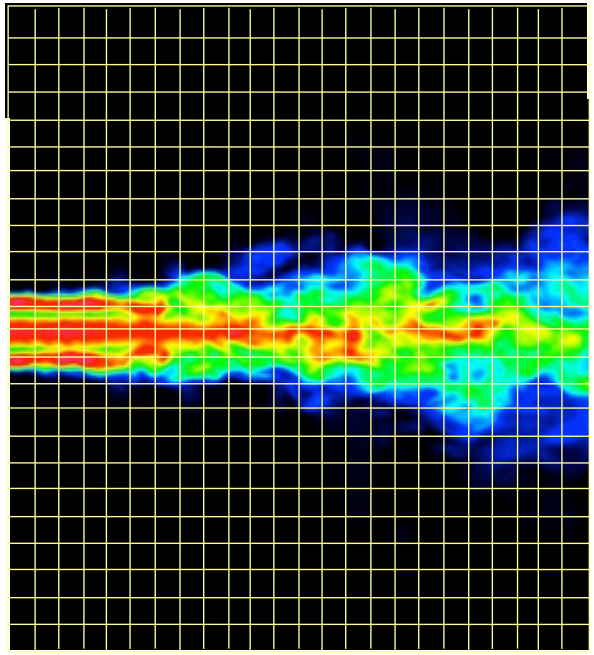
Granularity in Implementations

Coarse grid



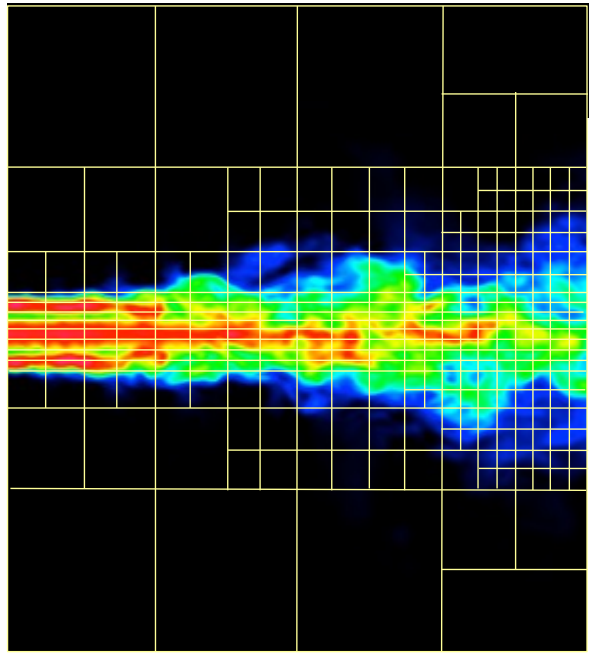
Higher Performance
Lower Accuracy
(Using Nodes)

Fine grid



Lower Performance
Higher Accuracy
(Using Processors)

Dynamic grid

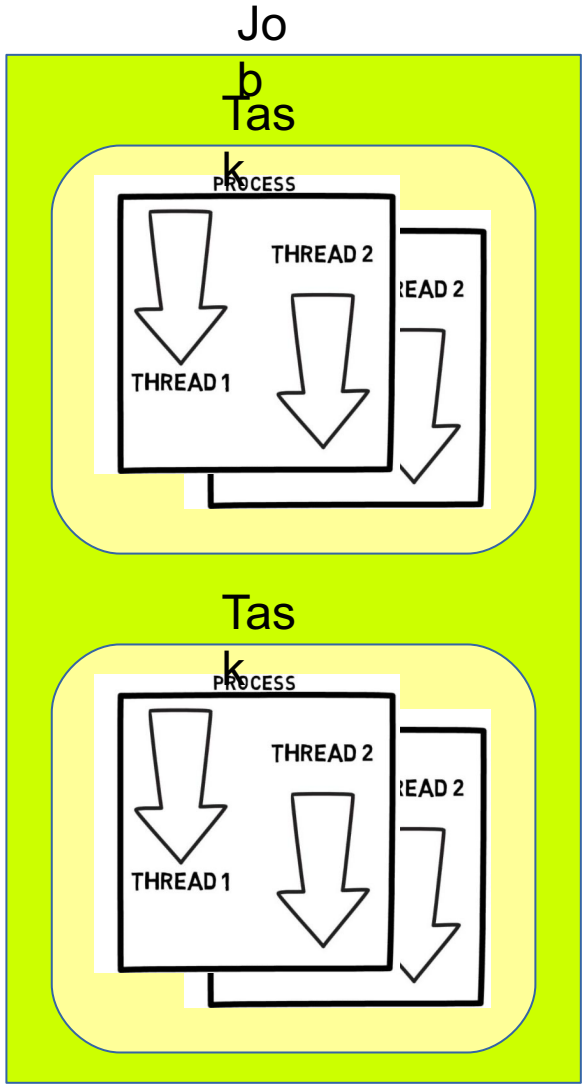


Target performance where accuracy is required (Using Processors and Nodes)

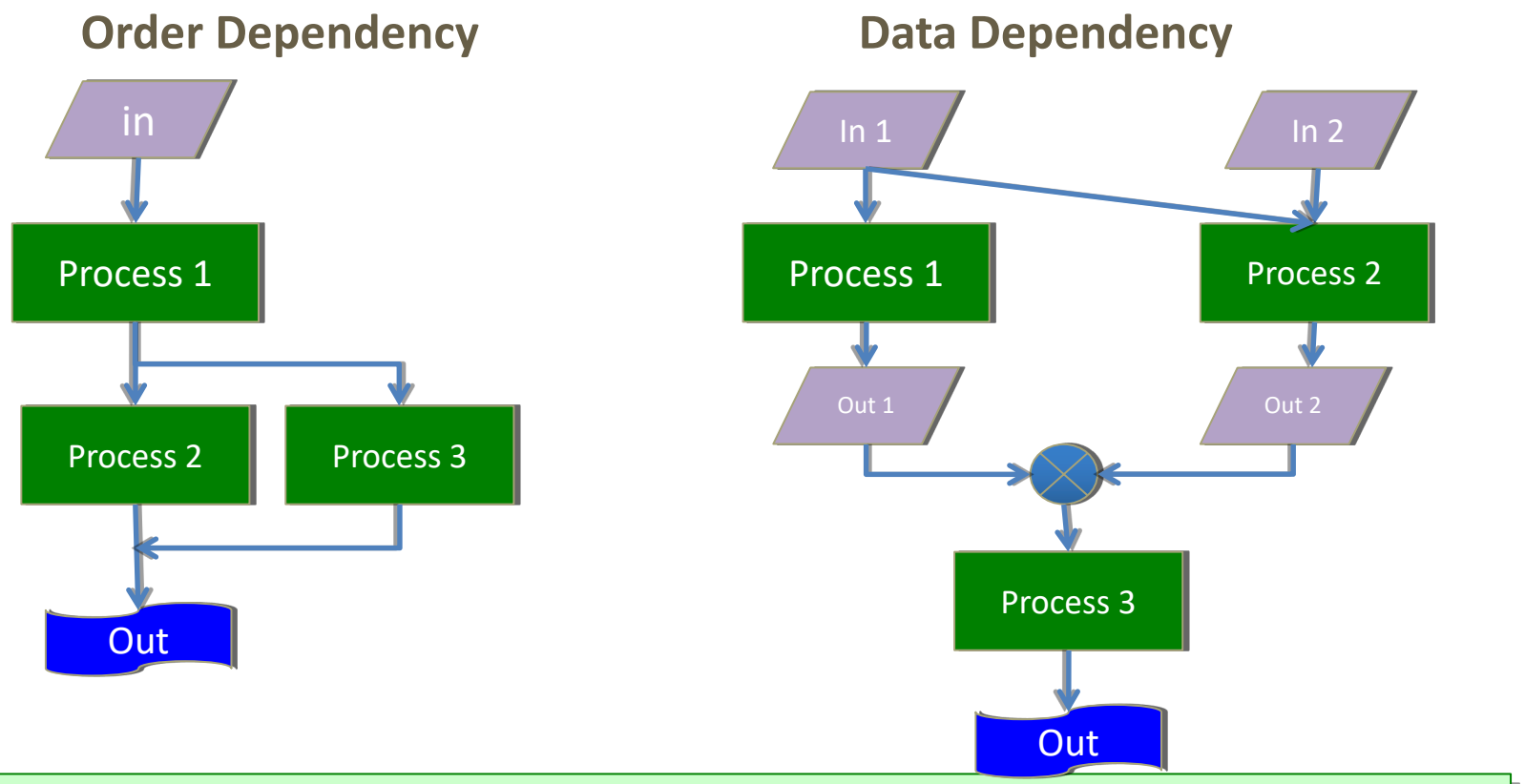
Task Decomposition Considerations

- What are the tasks and how are defined?
- What are the dependencies between task and how can they be satisfied?
- How are the task assigned to threads?

Tasks must be assigned to threads for execution



Task Dependencies

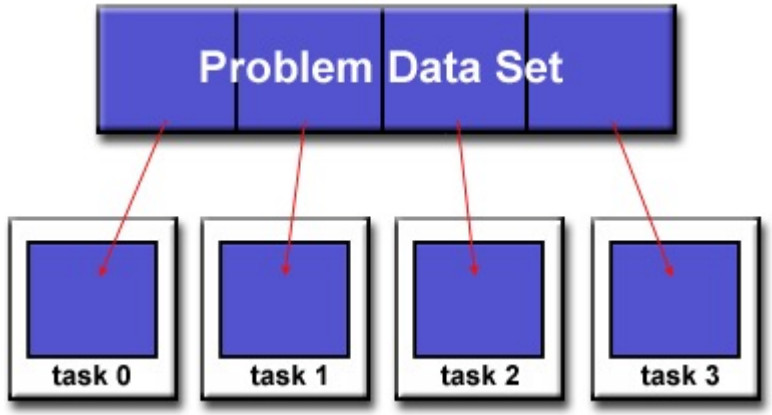


Enchantingly Parallel Code: Code without dependencies

Data Decomposition Considerations

(Geometric Decomposition)

Data Structures must be (commonly) divided in arrays or logical structures.

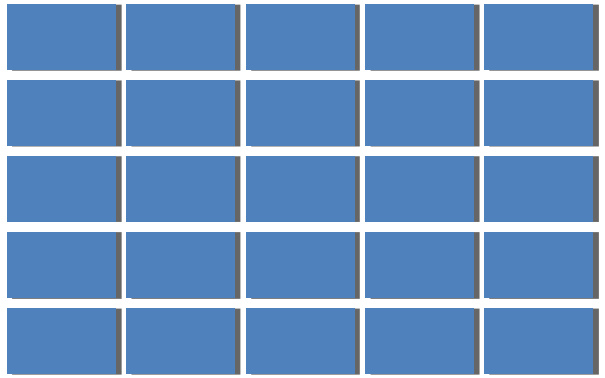


- How should you divide the data into chunks?
- How should you ensure that the tasks for each chunk have access to all data required for update?
- How are the data chunks assigned to threads?

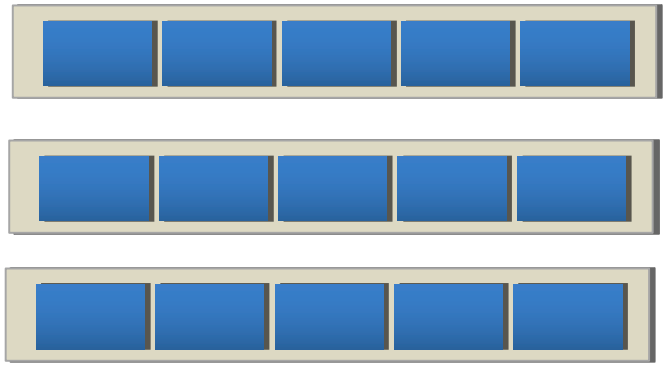
How are the data chunks (and tasks) assigned to threads?

- Data Chunks are associated with tasks and are assigned to threads statically or dynamically
- Via Scheduling
  - Static: when the amount of computations within tasks is uniform and predictable
  - Dynamic: to achieve a good balance due to variability in the computation needed by chunk
    - Require many (more) tasks than threads.

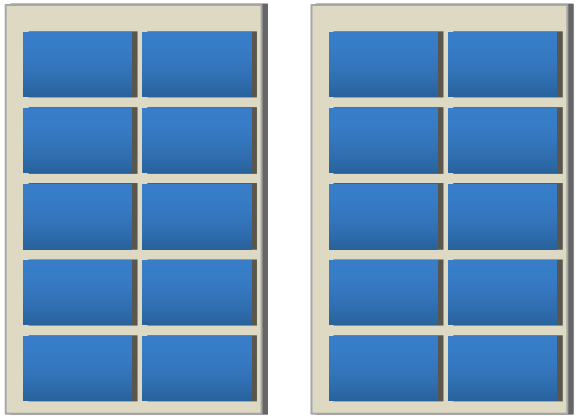
How should you divide data into chunks?



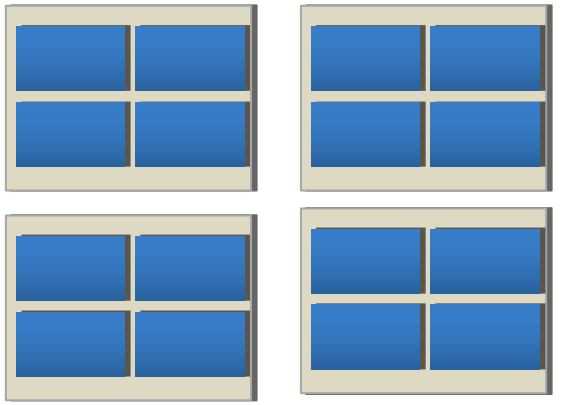
By individual elements



By rows



By groups of columns



By blocks

The Shape of the Chunk

- Data Decomposition have an additional dimension.
- It determines what the neighboring chunks are and how any exchange of data will be handled during the course of the chunk computations.

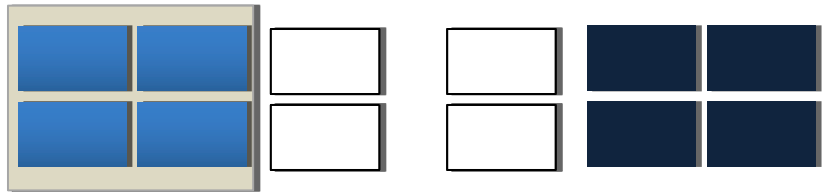
2 Shared Borders

5 Shared Borders

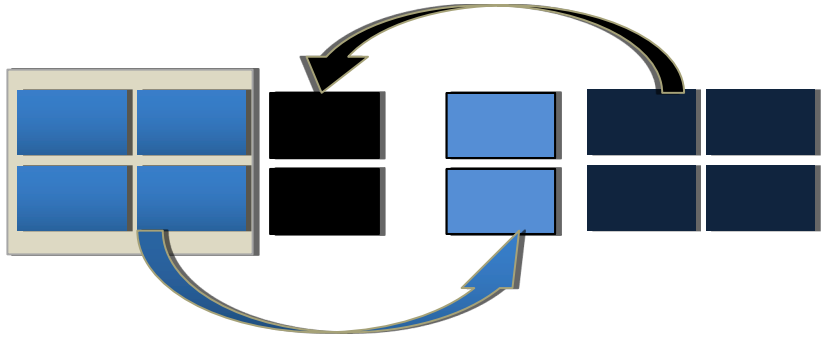
- Regular shapes : Common Regular data organizations.
- Irregular shapes: may be necessary due to the irregular organizations of the data.

How should you ensure that the tasks for each chunk have access to all data required for update?

- Using Ghost Cells
  - Using ghost cells to hold copied data from a neighboring chunk.



Original split with ghost cells



Copying data into ghost cells

Data Sharing Pattern

- Data decomposition might define some data that must be shared among the tasks.
- Data dependencies can also occur when one task needs access to some portions of the another task's local data.
  - Read Only
  - Effectively Local (Accessed by one of the tasks)
  - Read Write
    - Accumulative
    - Multiple read / Single Write

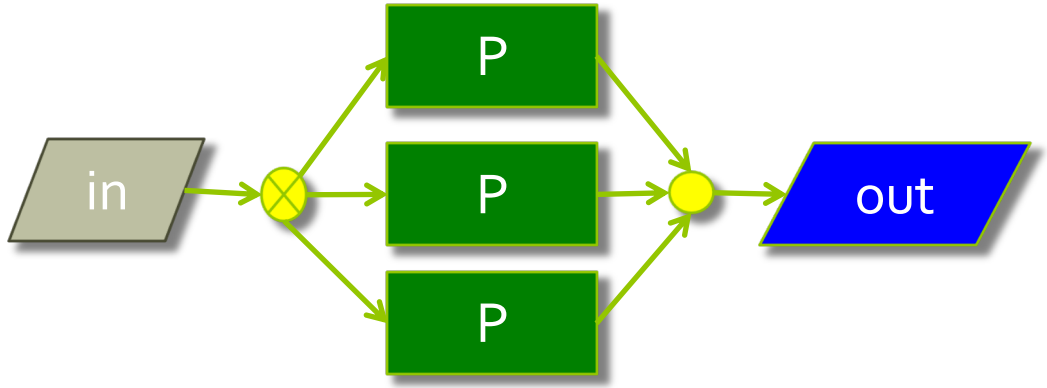
Tasks and Domain Decomposition Patterns

- **Task Decomposition Patterns**
  - Understand the computationally intensive parts of the problem.
  - Finding Tasks (as much…)
    - Actions that are carried out to solve the problem
    - Actions are distinct and relatively independent.
- **Data Decomposition Patterns**
  - Data decomposition implied by tasks.
  - Finding Domains:
    - Most computationally intensive part of the problem is organized around the manipulation of large data structure.
    - Similar operators are being applied to different parts of the data structure.
  - In shared memory programming environments, data decomposition will be implied by task decomposition (To see in detail in the OpenMP session).

Concurrent Computation from Serial Codes

- **Sequential Consistency Property:** Getting the same answer as the serial code on the same input data set, comparing sequence of execution in concurrent solutions of the concurrent algorithms.

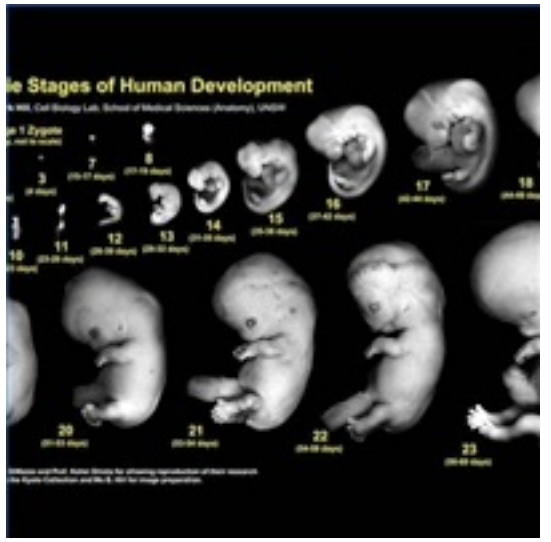
Sequential Version



Parallel / Concurrent Version

Not Parallelizable Jobs, Tasks and Algorithms

- Algorithms with state
- Recurrences
- Induction Variables
- Reductions
- Loop-carried Dependencies



The Mythical Man-Month: Essays on Software Engineering. By Fred Brooks. Ed Addison-Wesley Professional, 1995

Concurrent Design Models Features

- **Efficiency**
  - Concurrent applications must run quickly and make good use of processing resources.
- **Simplicity**
  - Easier to understand, develop, debug, verify and maintain.
- **Portability**
  - In terms of threading portability.
- **Scalability**
  - It should be effective on a wide range of number of threads and cores, and sizes of data sets.

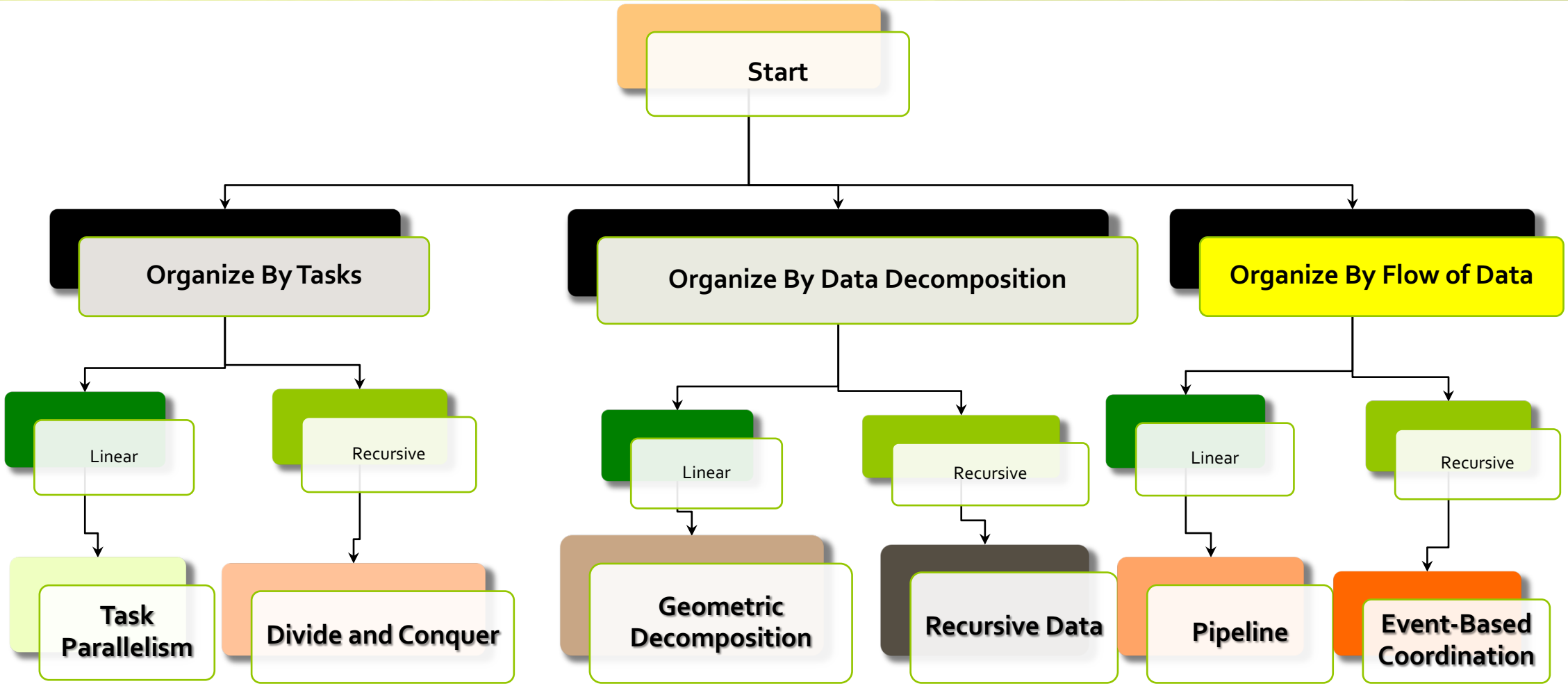
Design Evaluation Pattern

- Production of analysis and decomposition:
  - Task decomposition to identify concurrency
  - Data decomposition to indentify data local to each task
  - Group of task and order of groups to satisfy temporal constraints
  - Dependencies among tasks
- Design Evaluation
  - Suitability for the target platform
  - Design Quality
  - Preparation for the next phase of the design

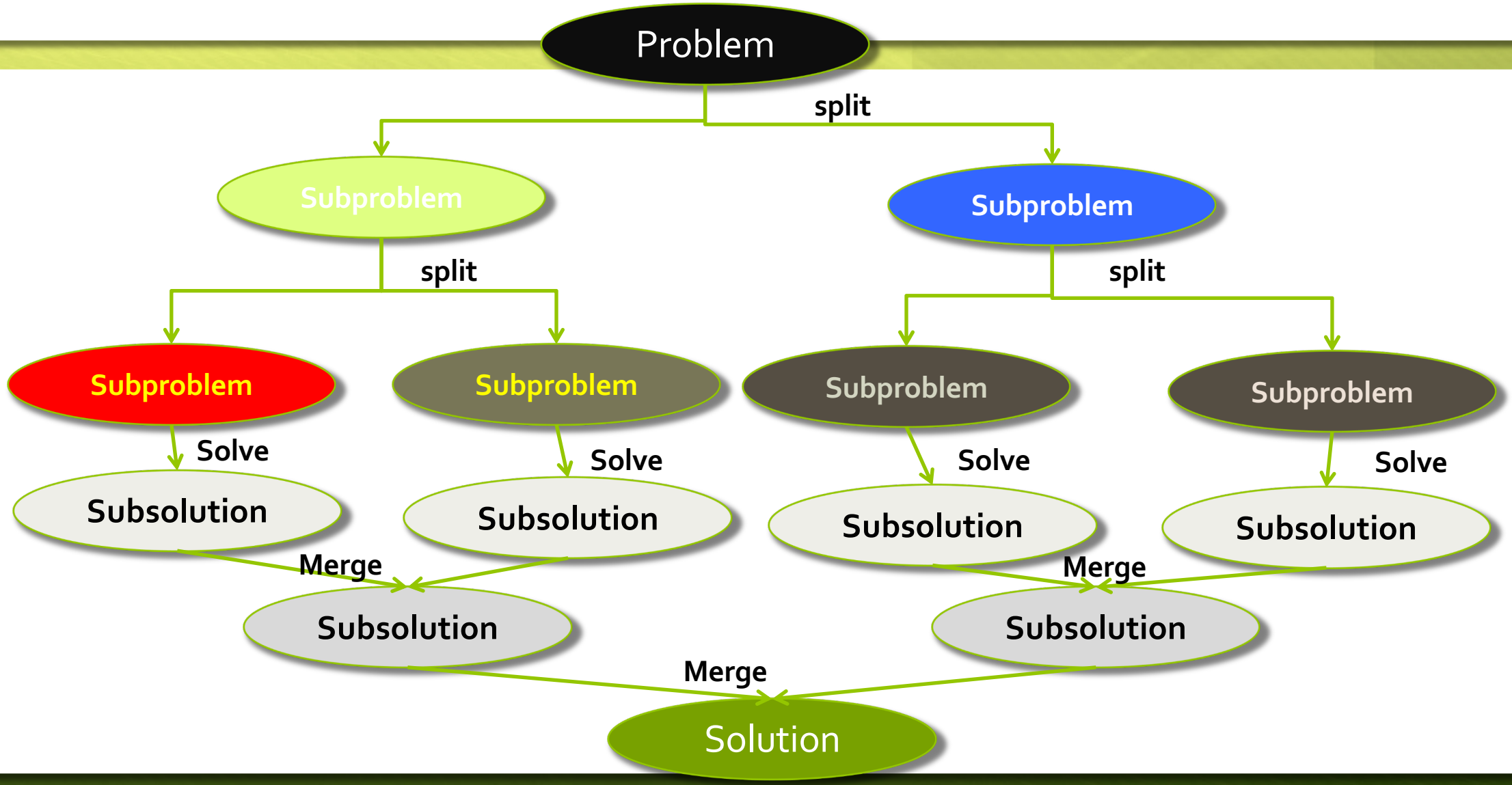
Algorithm Structures

- Organizing by Tasks
  - Task Parallelism
  - Divide and Conquer
- Organizing by Data Decomposition
  - Geometric Decomposition
  - Recursive Data
- Organizing by Flow of Data
  - Pipeline
  - Event-Based Coordination

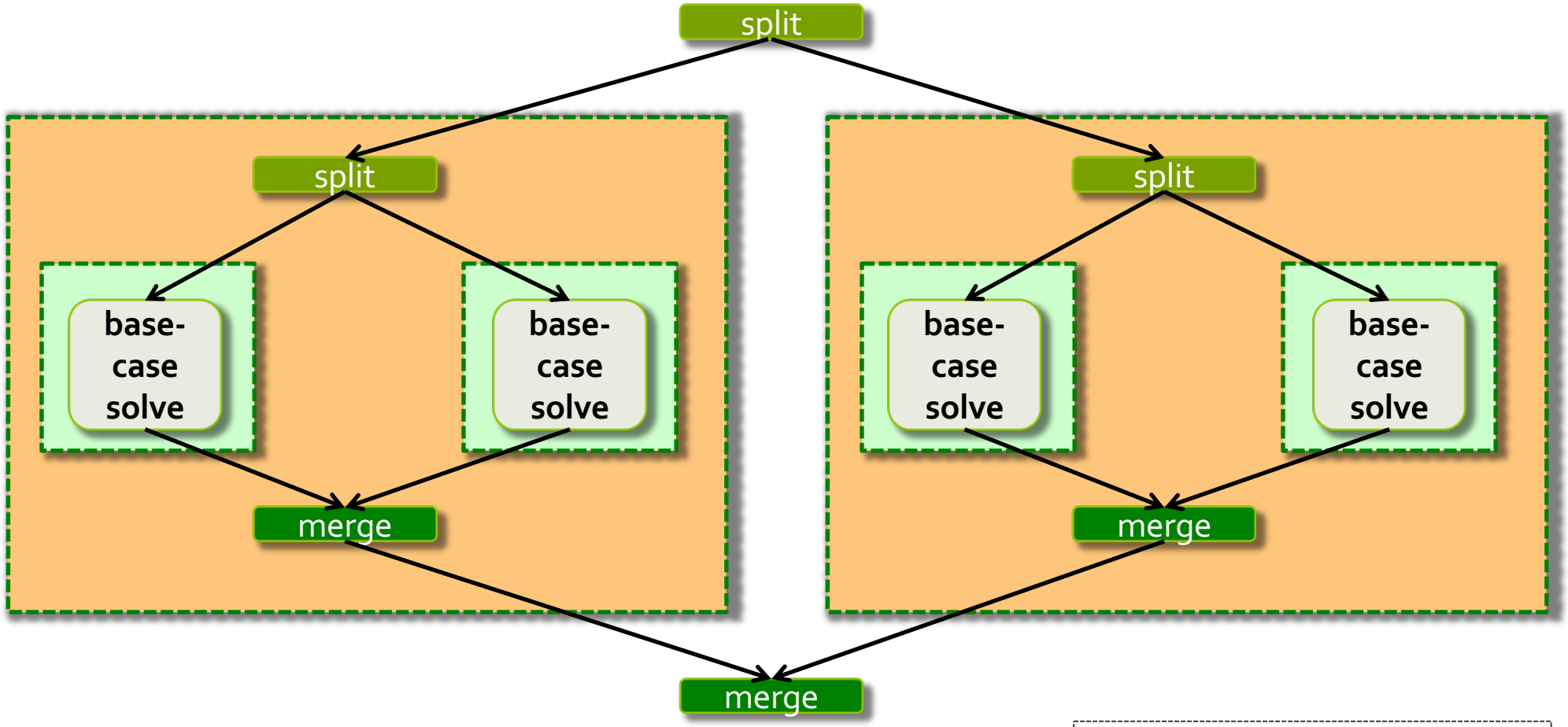
Algorithm Structure Decision Tree
(Major Organizing Principle)



Divide and Conquer Strategy



Divide and Conquer Parallel Strategy



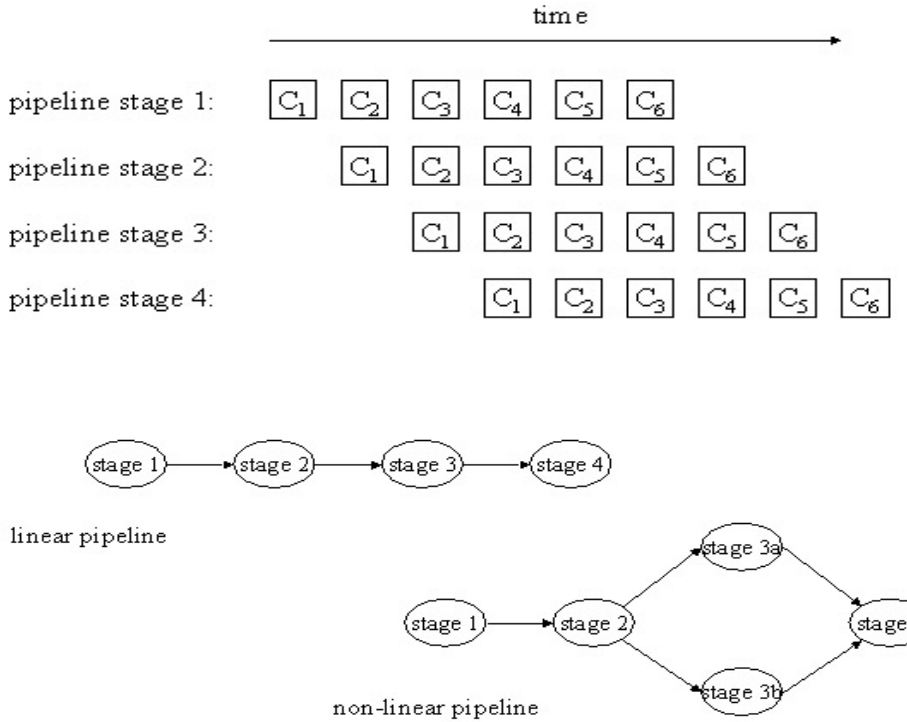
Recursive Data Strategy

- Involves an operation on a recursive data structure that appears to require sequential processing:
  - Lists
  - Trees
  - Graphs
- Recursive Data structure is completely decomposed into individual elements.
- Structure in the form of a loop (top-level structure)
- Simultaneously updating all elements of the data structure (Synchronization)

- Examples:
  - Partial sums of a linked list.
- Uses:
  - Widely used on SIMD platforms (HPF77)
  - Combinatorial optimization Problems.
  - Partial sums
  - List ranking
  - Euler tours and ear decomposition
  - Finding roots of trees in a forest of rooted directed trees.

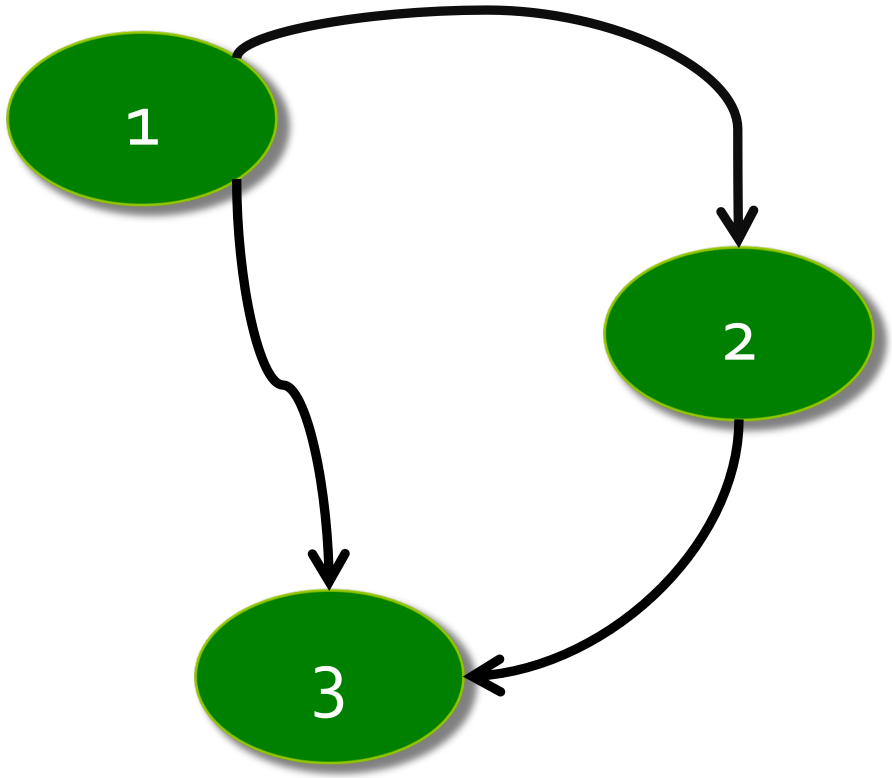
Pipeline Strategy

- Involves performing a calculation on many sets of data, where the calculation can be viewed in terms of data flowing through a sequence of stages
  - Instruction pipeline in modern CPUs
  - Vector Processing (Loop-level pipelining)
  - Algorithm-level Pipelining
  - Signal Processing
  - Graphics
  - Shell Programs in Unix



Event-Based Coordination Strategy

- Application decomposed into groups of semi-independent tasks interacting in an irregular fashion.
- Interaction determined by a flow of data between the groups, implying ordering constraints between the tasks



Some Conclusions

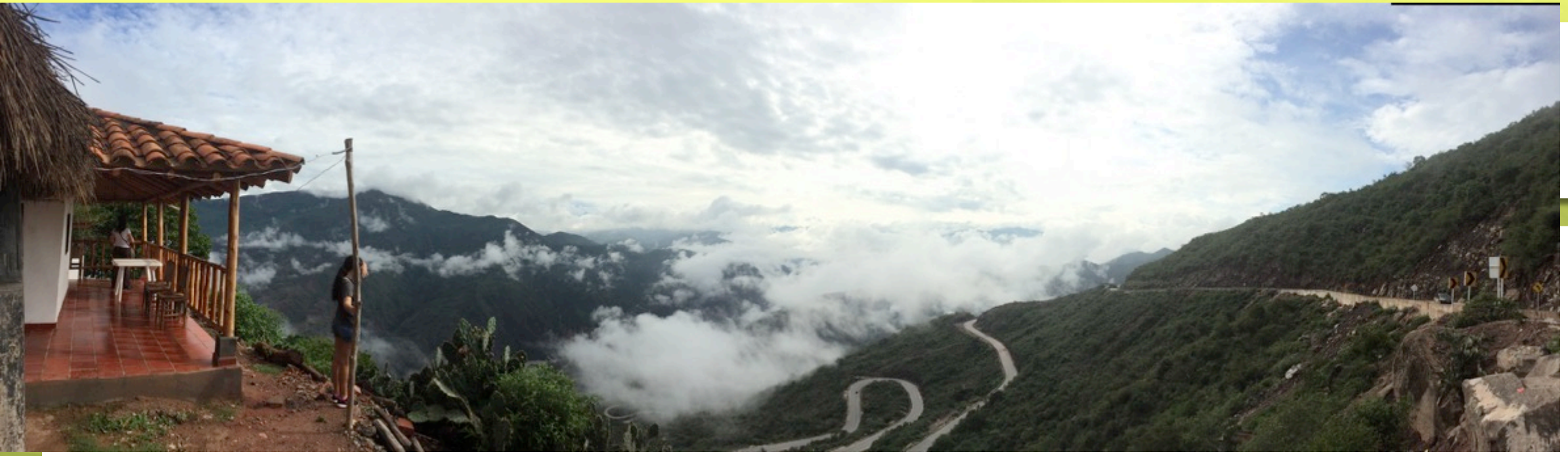
- High Performance Computing allows science and mathematics dreams and implementations… i.e. Artificial Intelligence Implementation, data analytics, blockchain and more…
- Computer systems involve different technologies and hybrid architectures, demanding sustainability, dynamicity and they need support changes in the scale of data and processing…. And all processing in parallel.
  - Of course, observing requirements of the applications and large scale behavior (i.e. IoT platforms)
- Power consumption, energy aware and computational efficiency reach sustainability. It is proposed from the design of the architecture and it must be dynamic.
  - Exascale challenges : Co-Design
- Big and little (embedded) HPC Architectures with the same challenges (memory contention, stable speed – up, parallel coherence) follows same kind of solutions, but with different scale of treatment observing the data level.
  - Involving Software Engineering, Computer Architecture, Data Analytics and Performance Evaluation.
- HPC is expensive (but It is more expensive to not have HPC Knowledge and Resources)
- Parallel Computing is not a tendency. (From 2015 is mandatory in all universities and colleges in USA parallel computing, scientific computing and advanced computing courses in science and engineering programs (programming computing is mandatory also in high school from 2009).

Recommended Lectures

- **The Art of Concurrency "A thread Monkey's Guide to Writing Parallel Applications"**, by *Clay Breshears* (Ed. O Reilly, 2009)
- **Writing Concurrent Systems. Part 1.**, by *David Chisnall* (InformIT Author's Blog: http://www.informit.com/articles/article.aspx?p=1626979)
- **Patterns for Parallel Programming**., by T. Mattson., B. Sanders and B. MassinGill (Ed. Addison Weslley, 2009) Web Site: http://www.cise.ufl.edu/research/ParallelPatterns/
- Designing and Building Parallel Programs, by Ian Foster in http://www.mcs.anl.gov/~itf/dbpp/
- Lectures in the site: www.sc-camp.org

Class work

- Revision of Chapter 2 of Designing and Building Parallel Programs, by Ian Foster in http://www.mcs.anl.gov/~itf/dbpp/
- Solve in the Exercises Section the 1 and 2 numerals.
- Imagine a solution for a real-world high complex problem to solve in the campus (conceptually)
- Read http://www.cs.wisc.edu/multifacet/papers/ieeecomputer08_amdahl_multicore.pdf



Questions?

Follow us: @SuperCCamp

Or visit: www.sc-camp.org

#SCCAMP2021