An Introduction to HPC and Advanced Computing

In 105 Slides - Part 3

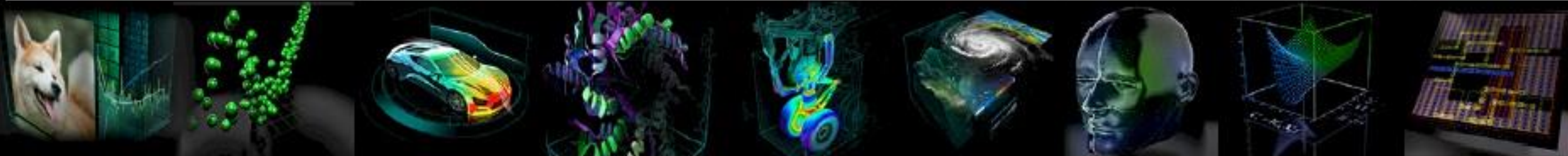
Carlos Jaime Barrios Hernández, PhD

An Example of Software Stack:

The NVIDIA Software Ecosystem

Software To Deliver Acceleration For HPC & AI Apps; 500+ New Updates

Machine Learning & Deep Learning	Computational Physics & Chemistry	Computational Fluid Dynamics	Life Sciences & Bioinformatics	Structural Mechanics	Weather & Climate	Geoscience, Seismology & Imaging	Numerical Analytics	Electronic Design Automation



600+ Apps

Linear Algebra | Parallel Algorithms | Signal Processing | Deep Learning | Machine Learning | Visualization

CUDA-X HPC & AI

40+ GPU Acceleration Libraries

CUDA

Desktop Development	Data Center	Supercomputers	GPU-Accelerated Cloud

www.nvidia.com

The Software/Applications Approach

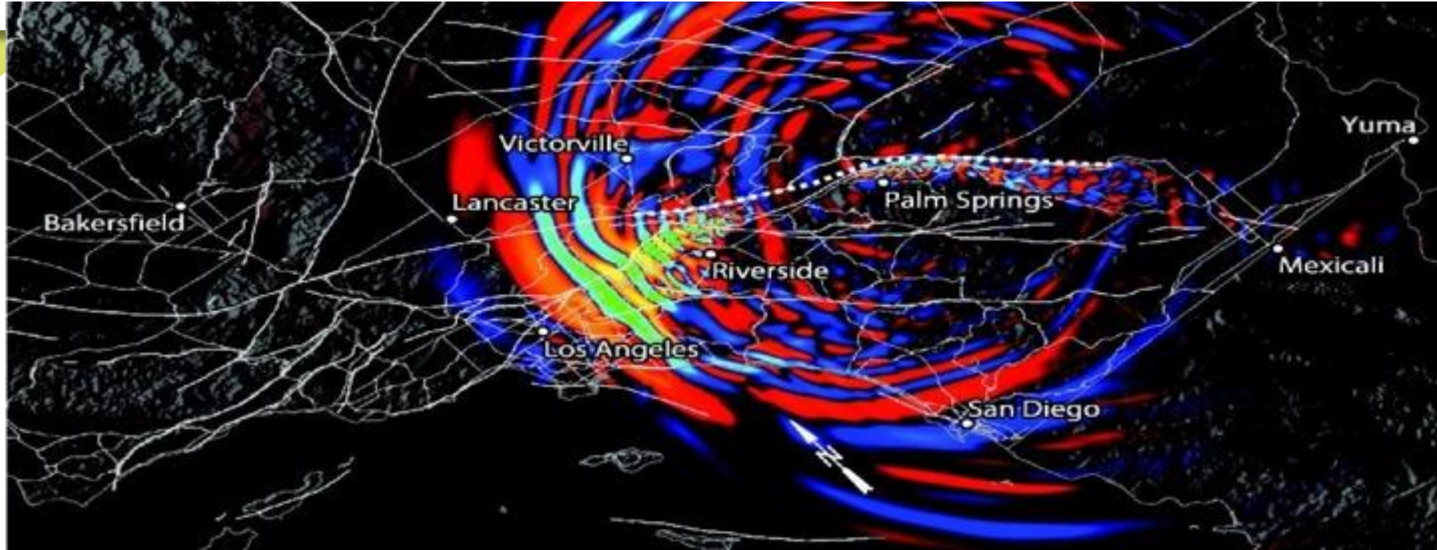
About High Performance Computing

- 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 Application Examples

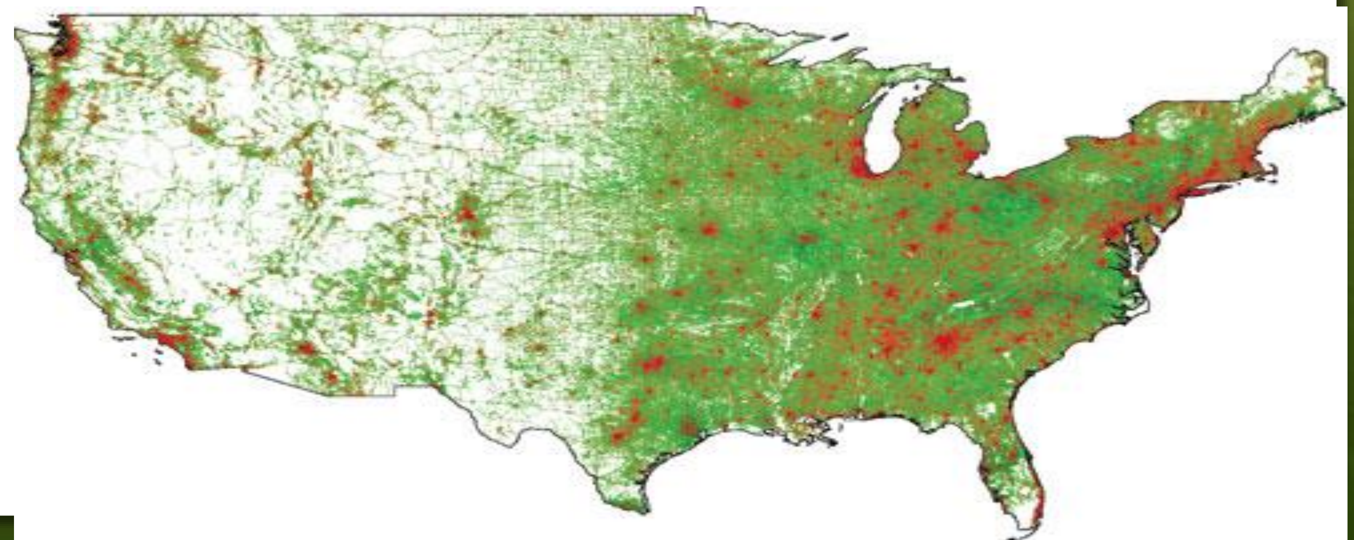


Earthquake simulation

Surface velocity 75 sec after earthquake

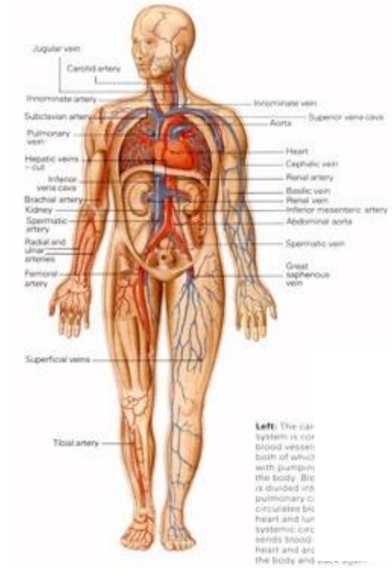
Flu pandemic simulation
300 million people tracked

Density of infected population, 45 days after breakout



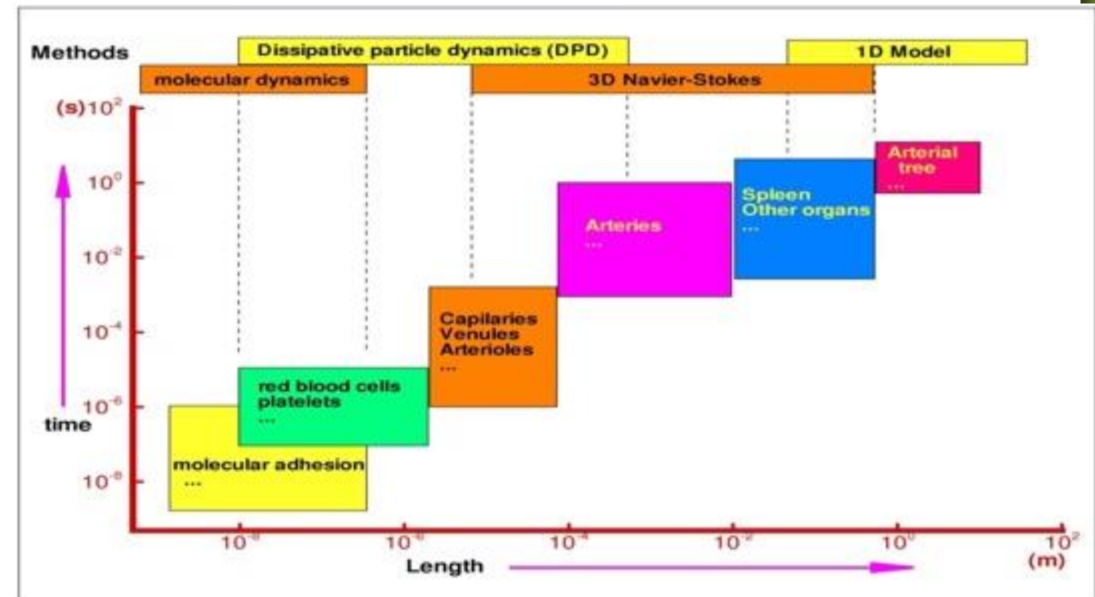
HPC Examples: Blood Flow in Human Vascular Network

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



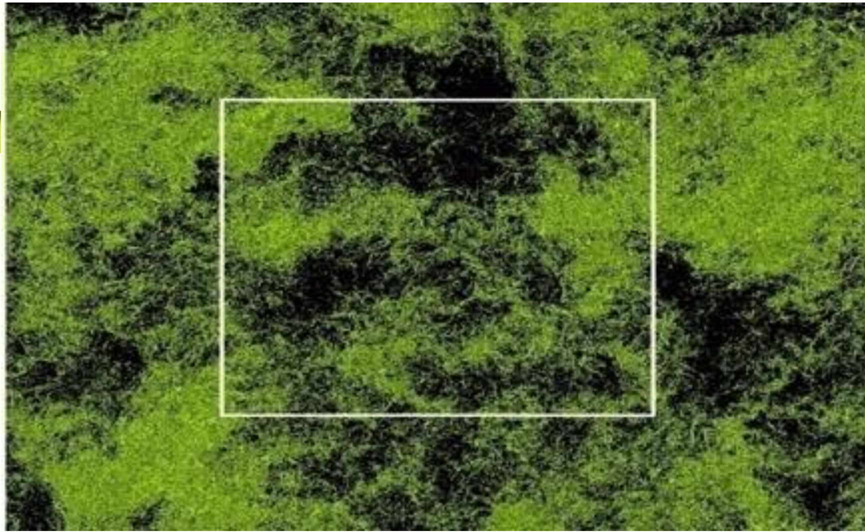
In one minute, the heart pumps the entire blood supply of 5 quarts through 60,000 miles of vessels, that is a quarter of the distance between the moon and the earth

Blood flow involves multiple scales

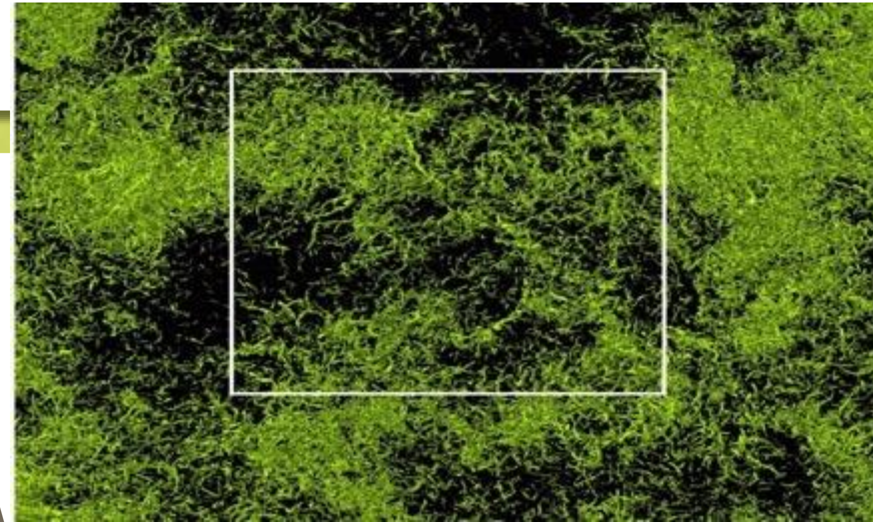


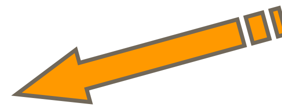
Computational challenges: Enormous problem size

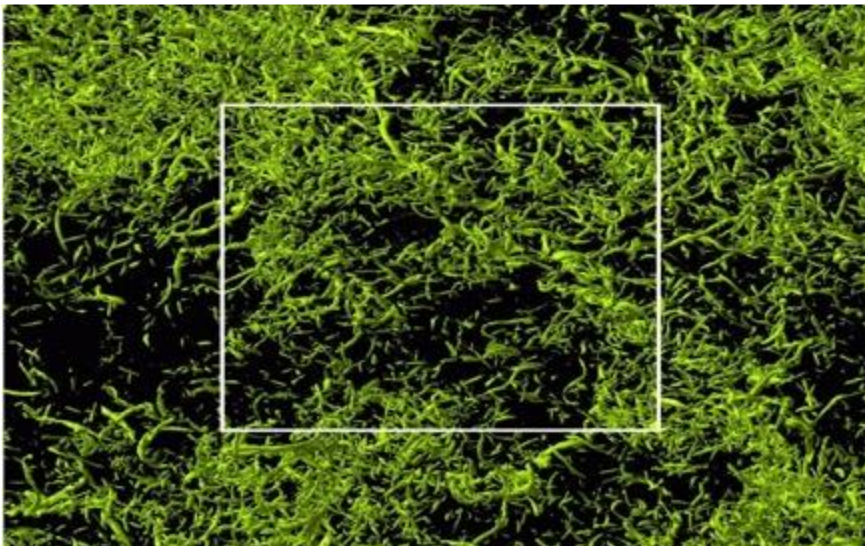
HPC Example: Homogeneous Turbulence



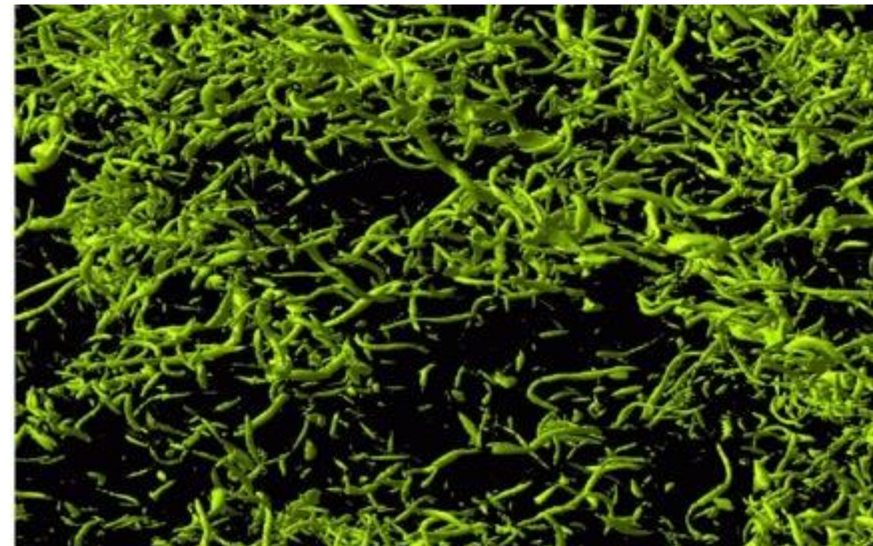
Zoom-in







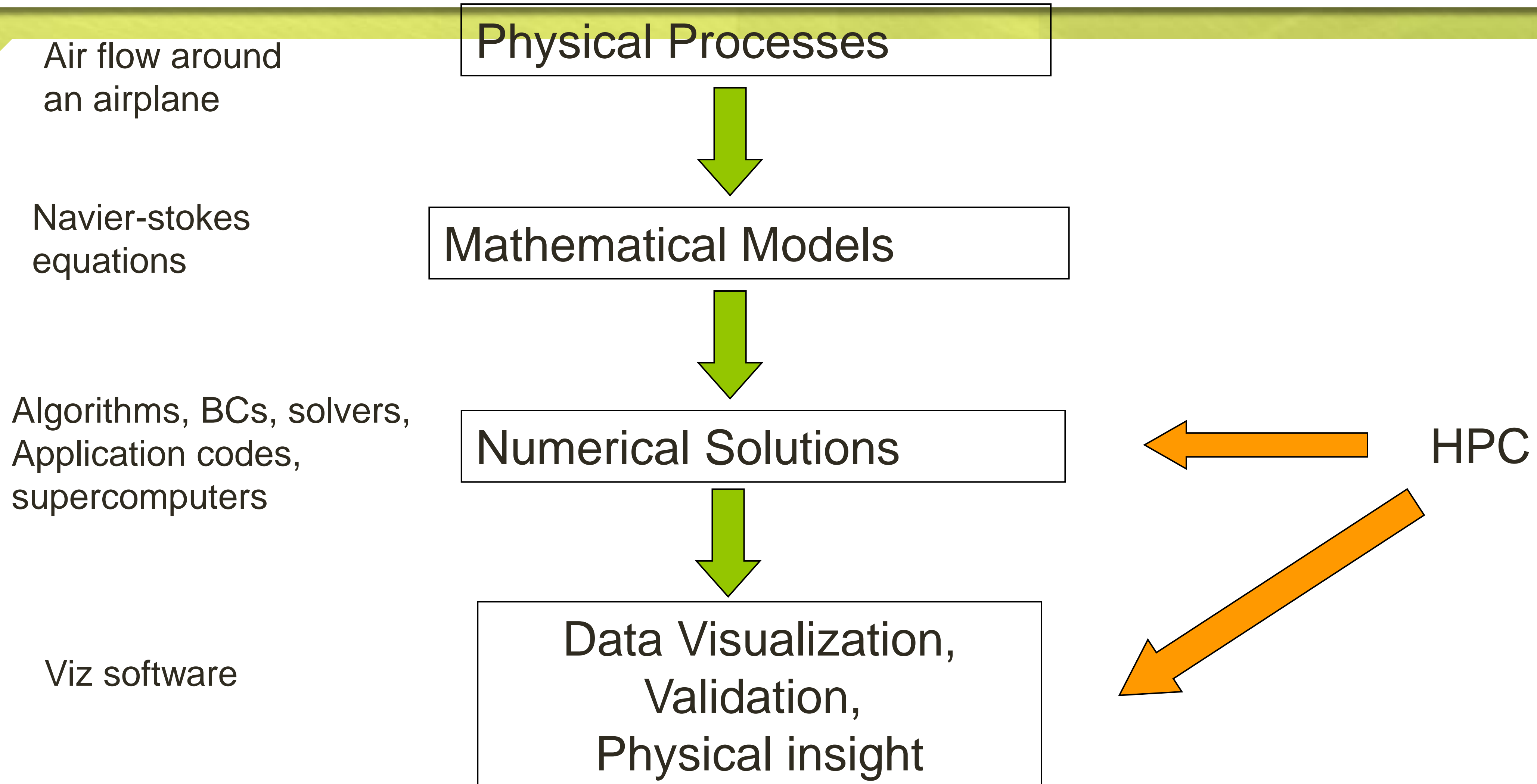
Zoom-in



Vorticity iso-surface

Direct Numerical Simulation of Homogeneous Turbulence: 4096^3

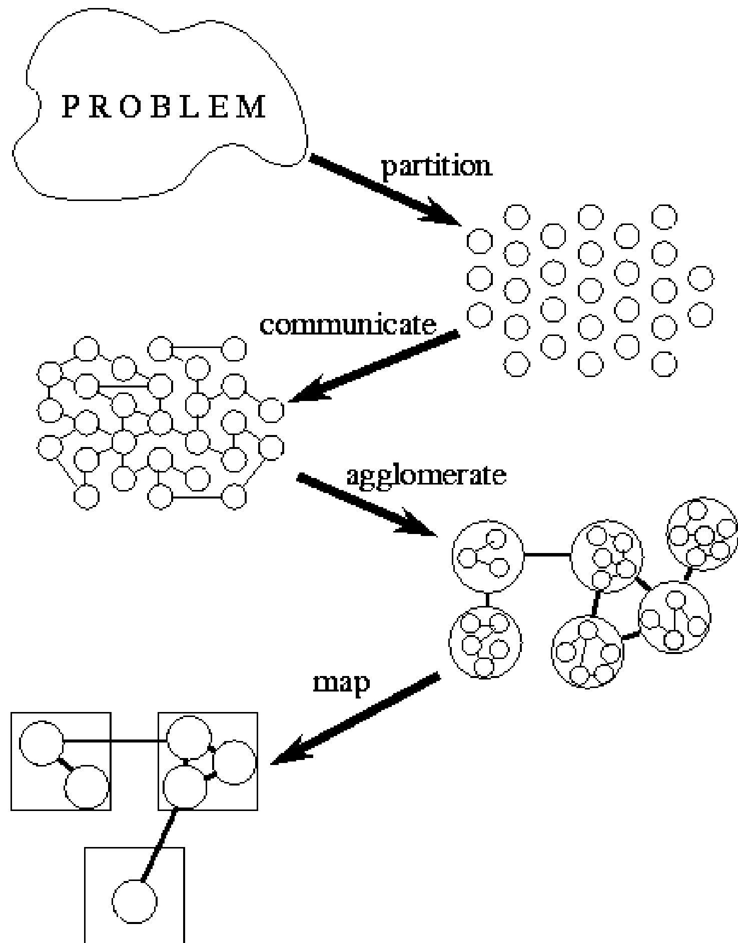
How HPC fits into Scientific Computing



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.

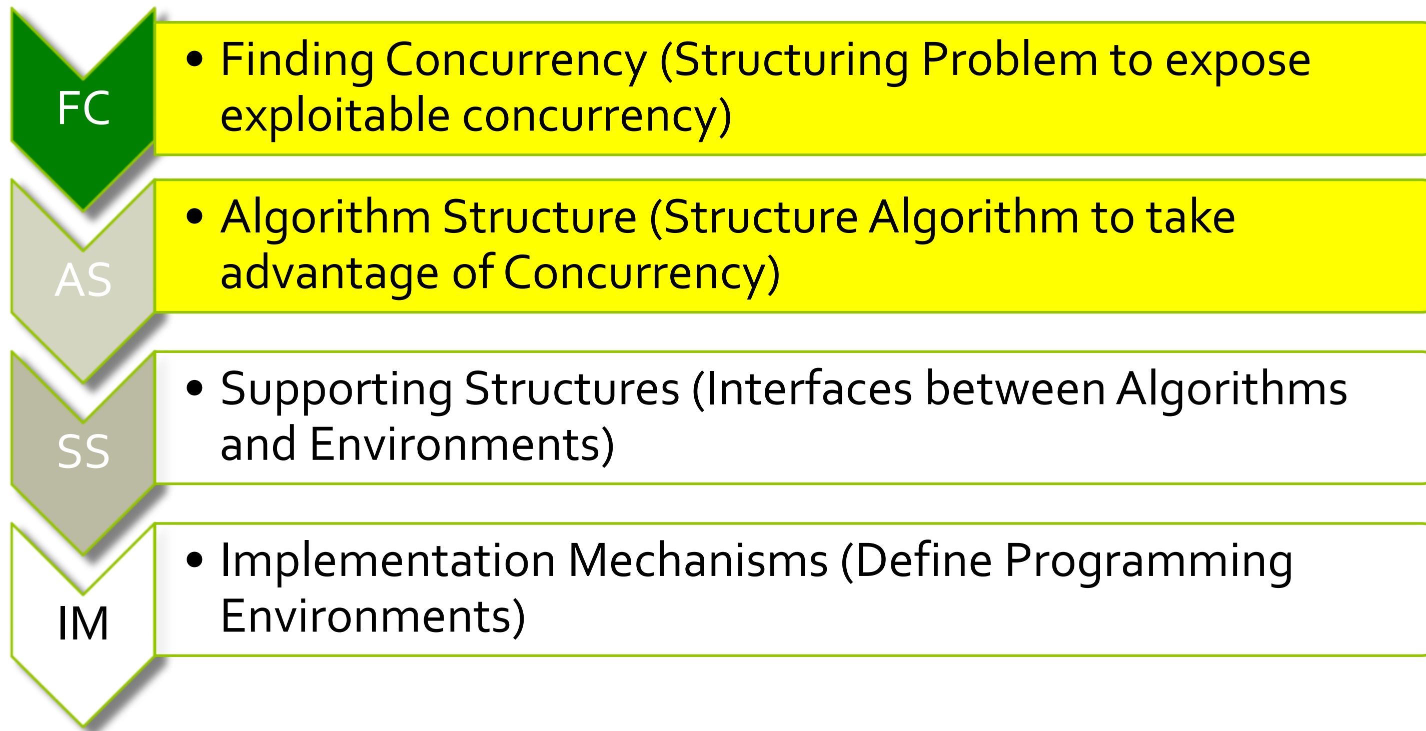
Addressed (mainly) for distributed memory applications

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)

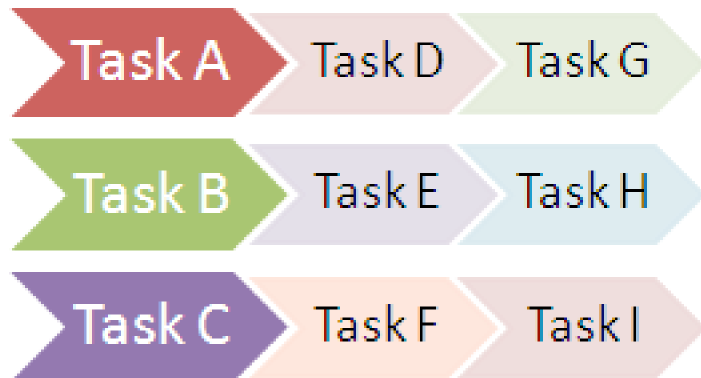
Common Features

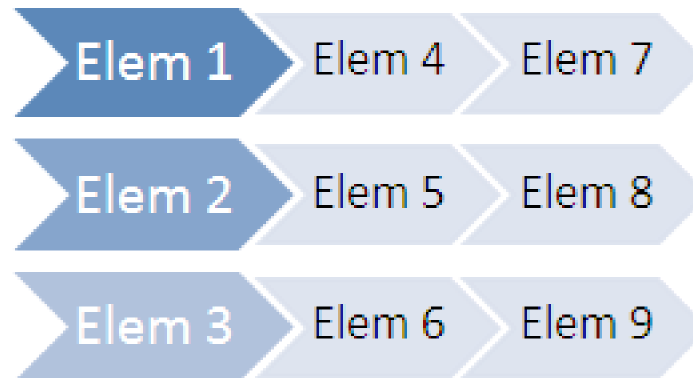
- Redundant Work
- Dividing Work
- Sharing Data (Different Methods)
- Dynamic / Static Allocation of Work
  - Depending of the nature of serial algorithm, resulting concurrent version, number of threads / processors

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





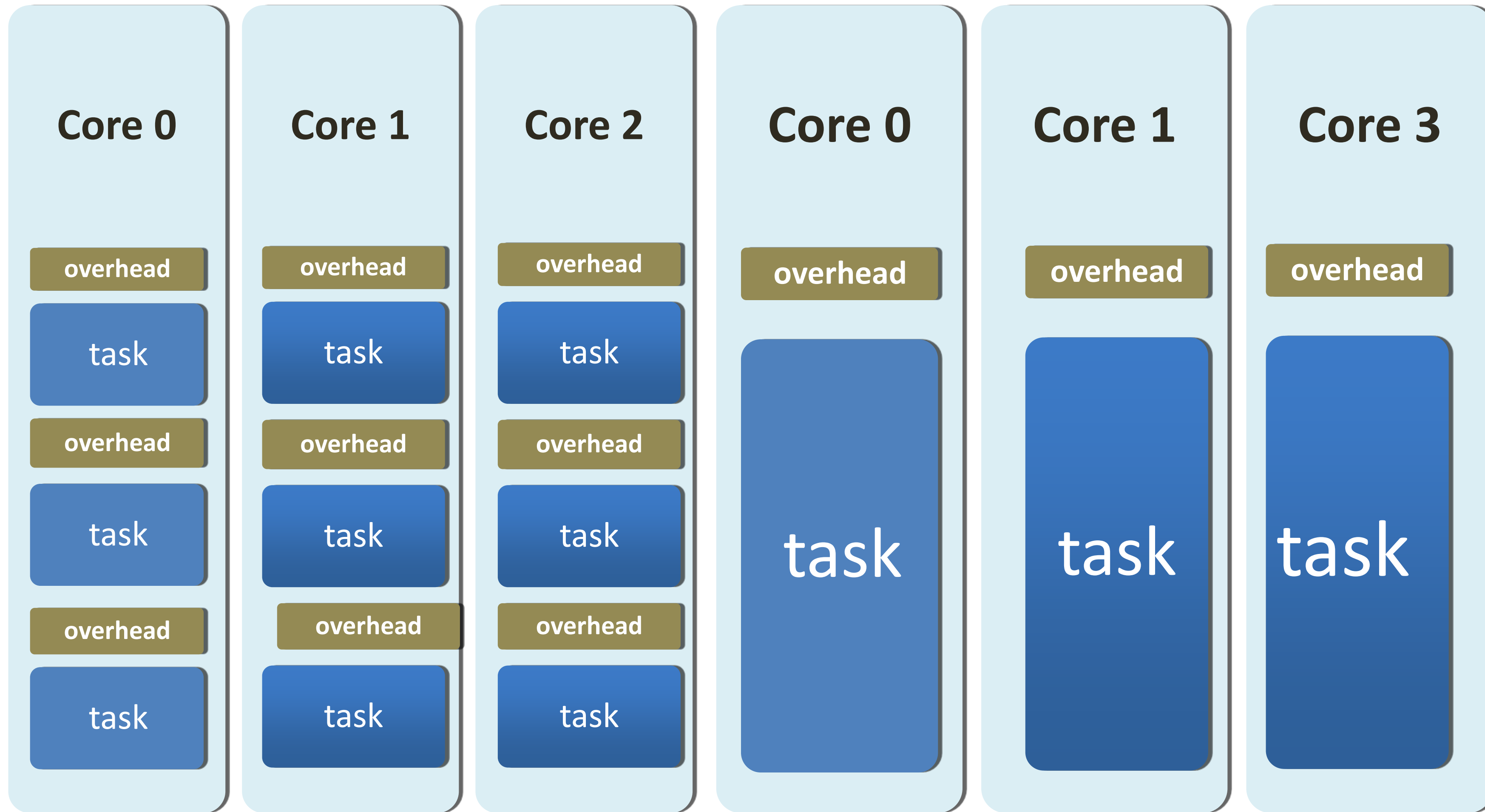
Tasks Decomposition : Task Parallelism
Data Decomposition: Data Parallelism /Geometric Parallelism

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)

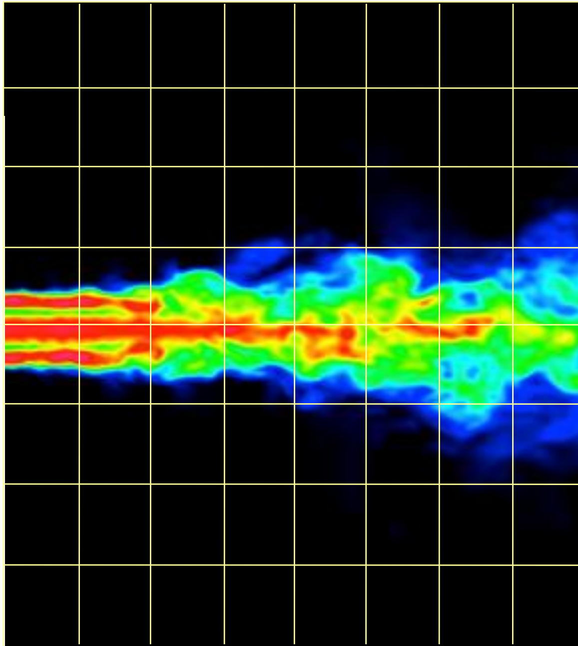
Granularity is the amount of computation done before synchronization is needed

Task Granularity



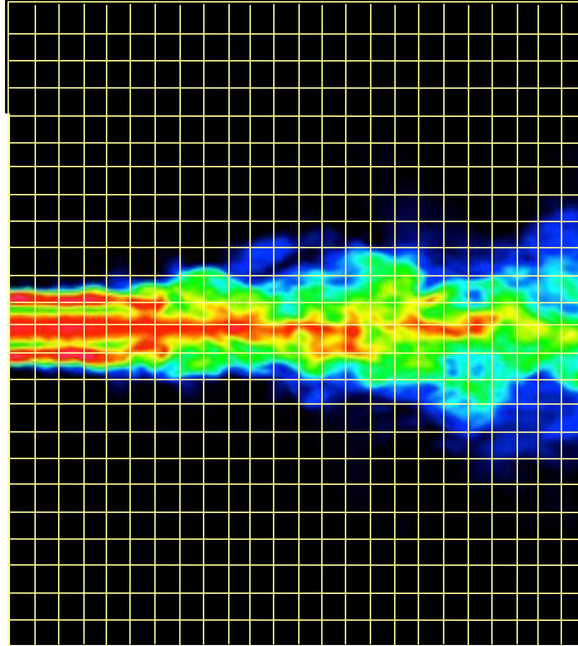
Granularity in Implementations

Coarse grid



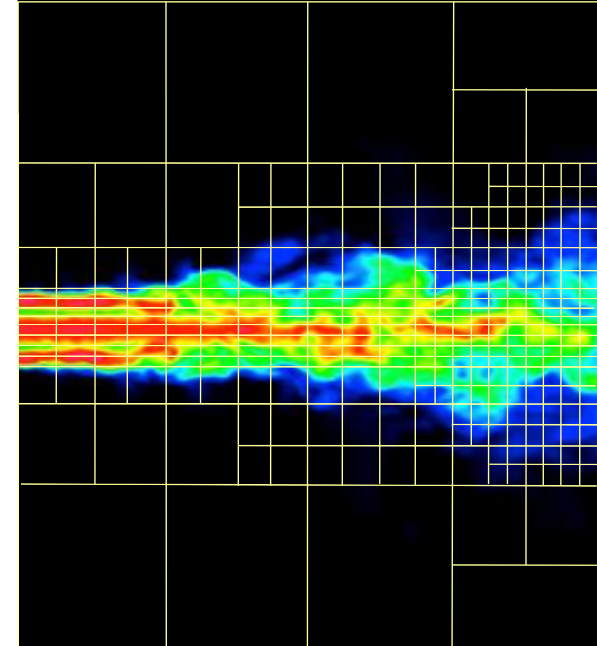
Higher Performance
Lower Accuracy
(Using Nodes)

Fine grid



Lower Performance
Higher Accuracy
(Using Processors)

Dynamic grid

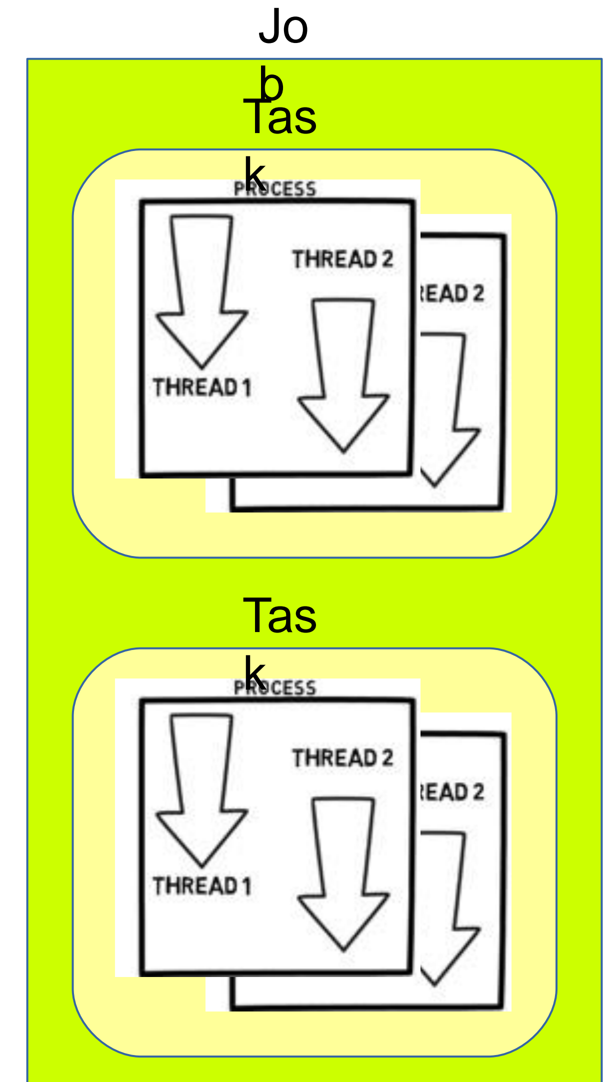


Target performance where accuracy is required (Using Processors and Nodes)

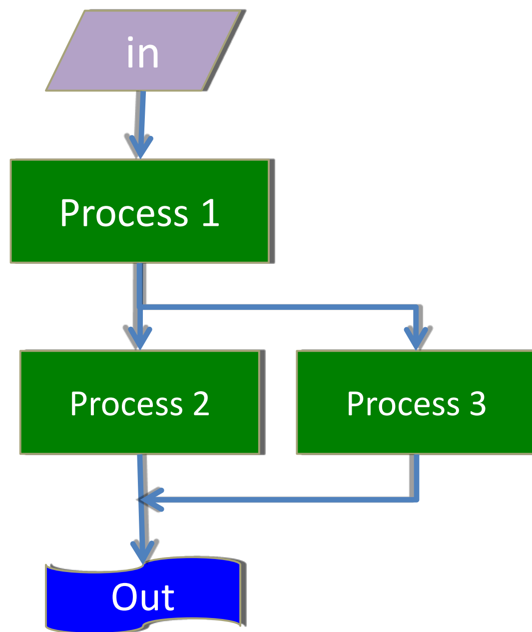
Task Decomposition Considerations

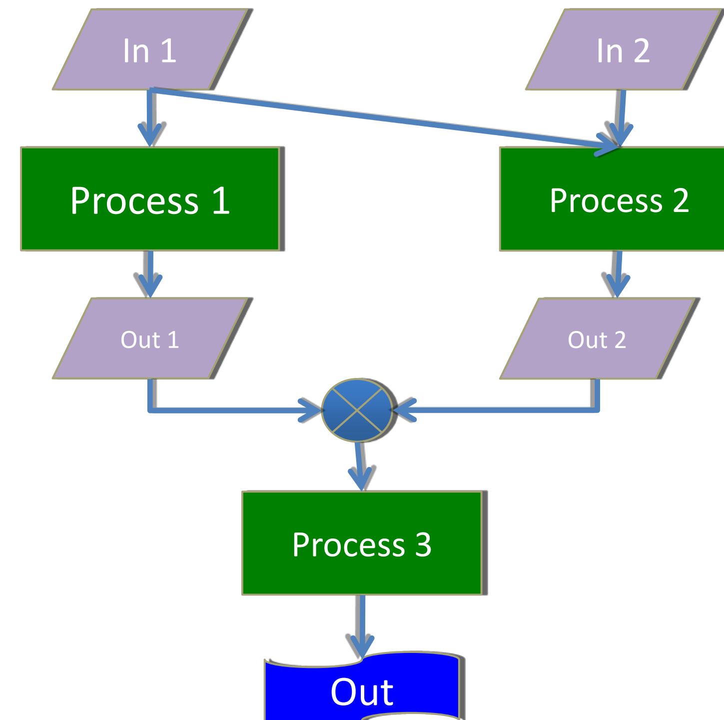
- What are the tasks and how are defined?
- What are the dependencies between task and how can they be satisfied?
- How are the task assigned to threads?

Tasks must be assigned to threads for execution



Task Dependencies

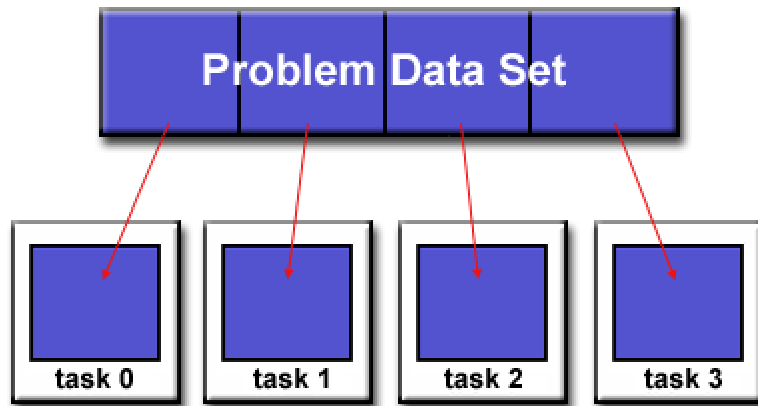




Enchantingly Parallel Code: Code without dependencies

Data Decomposition Considerations
(Geometric Decomposition)

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

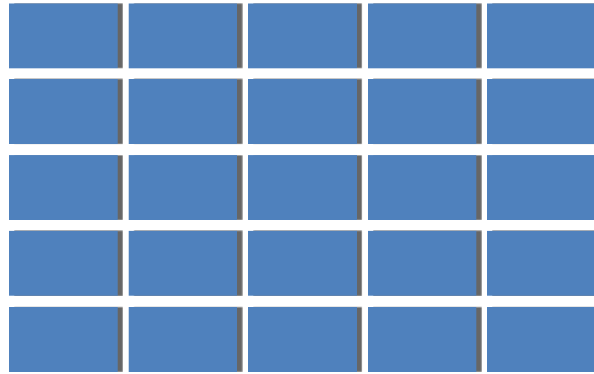


- How should you divide the data into chunks?
- How should you ensure that the tasks for each chunk have access to all data required for update?
- How are the data chunks assigned to threads?

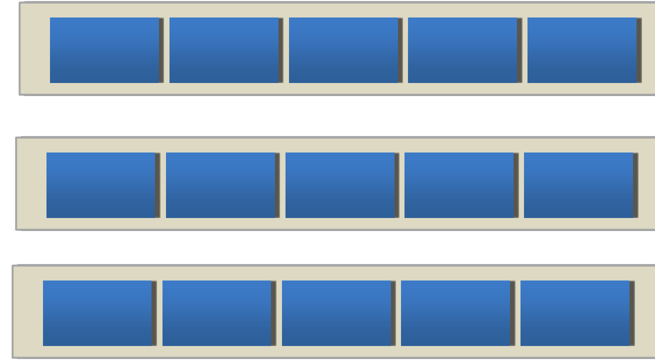
How are the data chunks (and tasks) assigned to threads?

- 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?

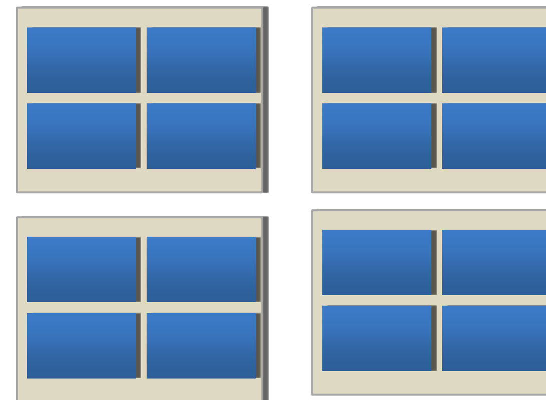


By individual elements



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.

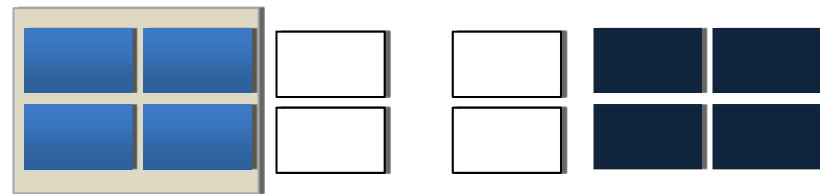
2 Shared Borders

5 Shared Borders

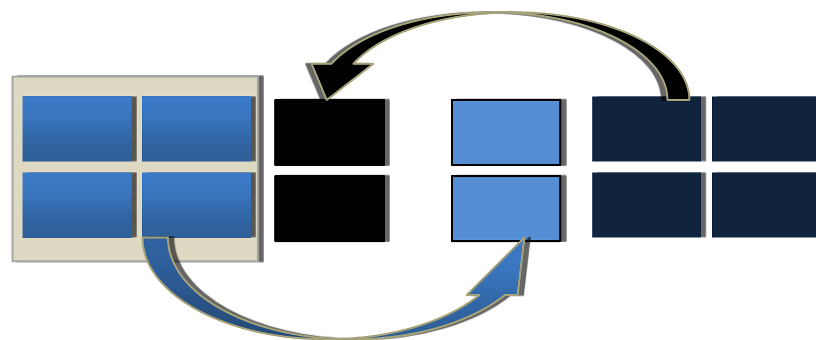
- Regular shapes : Common Regular data organizations.
- Irregular shapes: may be necessary due to the irregular organizations of the data.

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

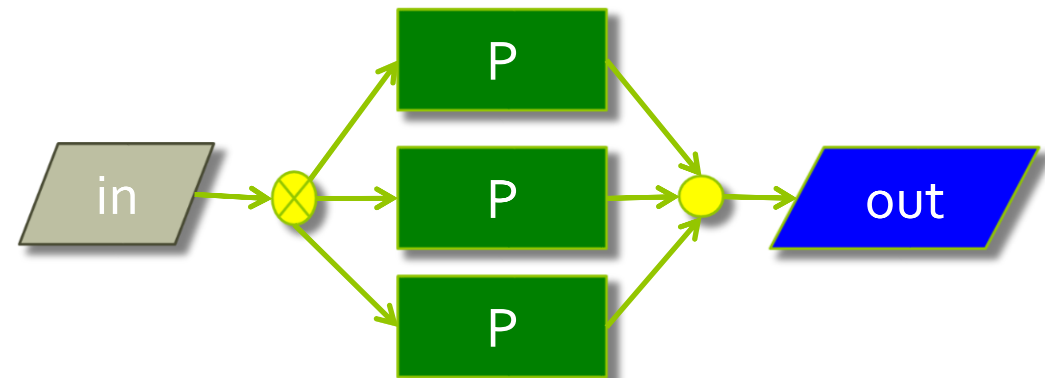
Tasks and Domain Decomposition Patterns

- **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.

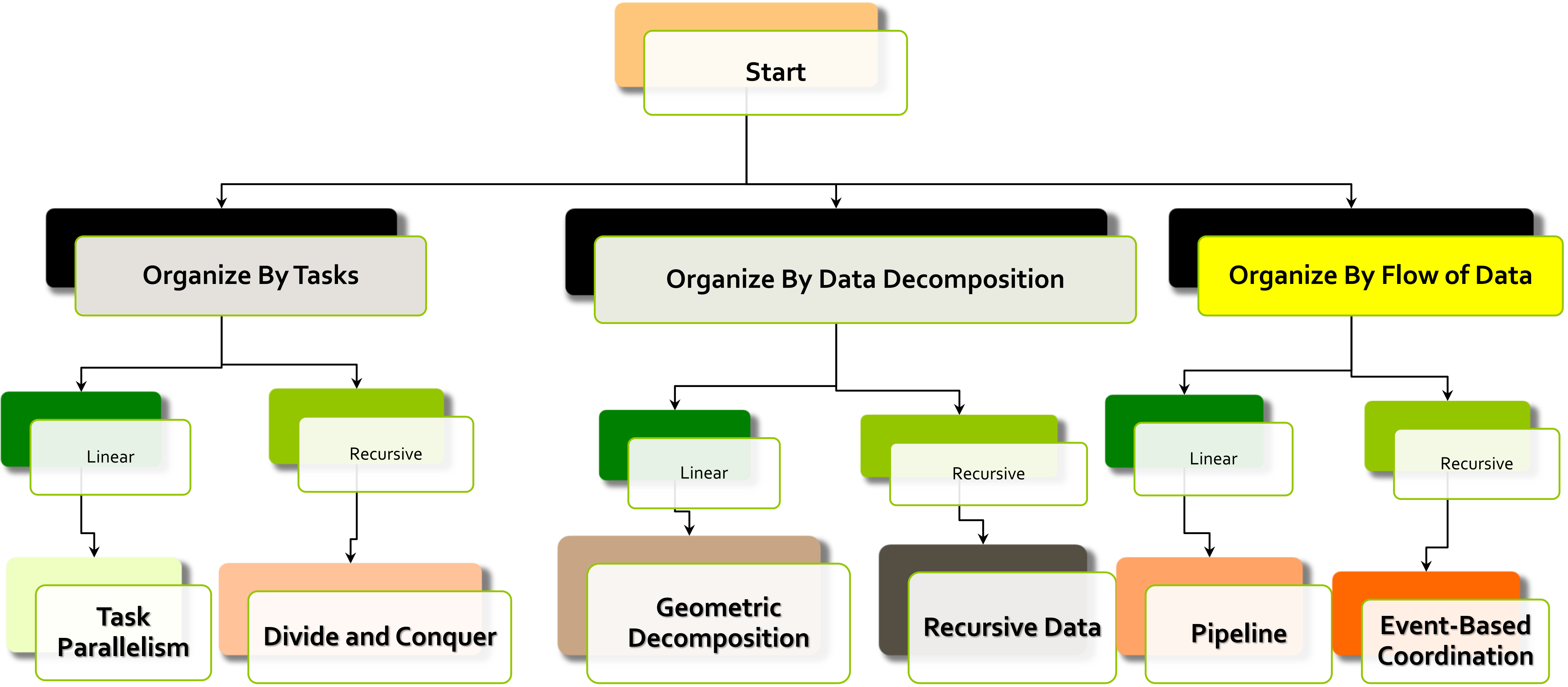
Design Evaluation Pattern

- 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

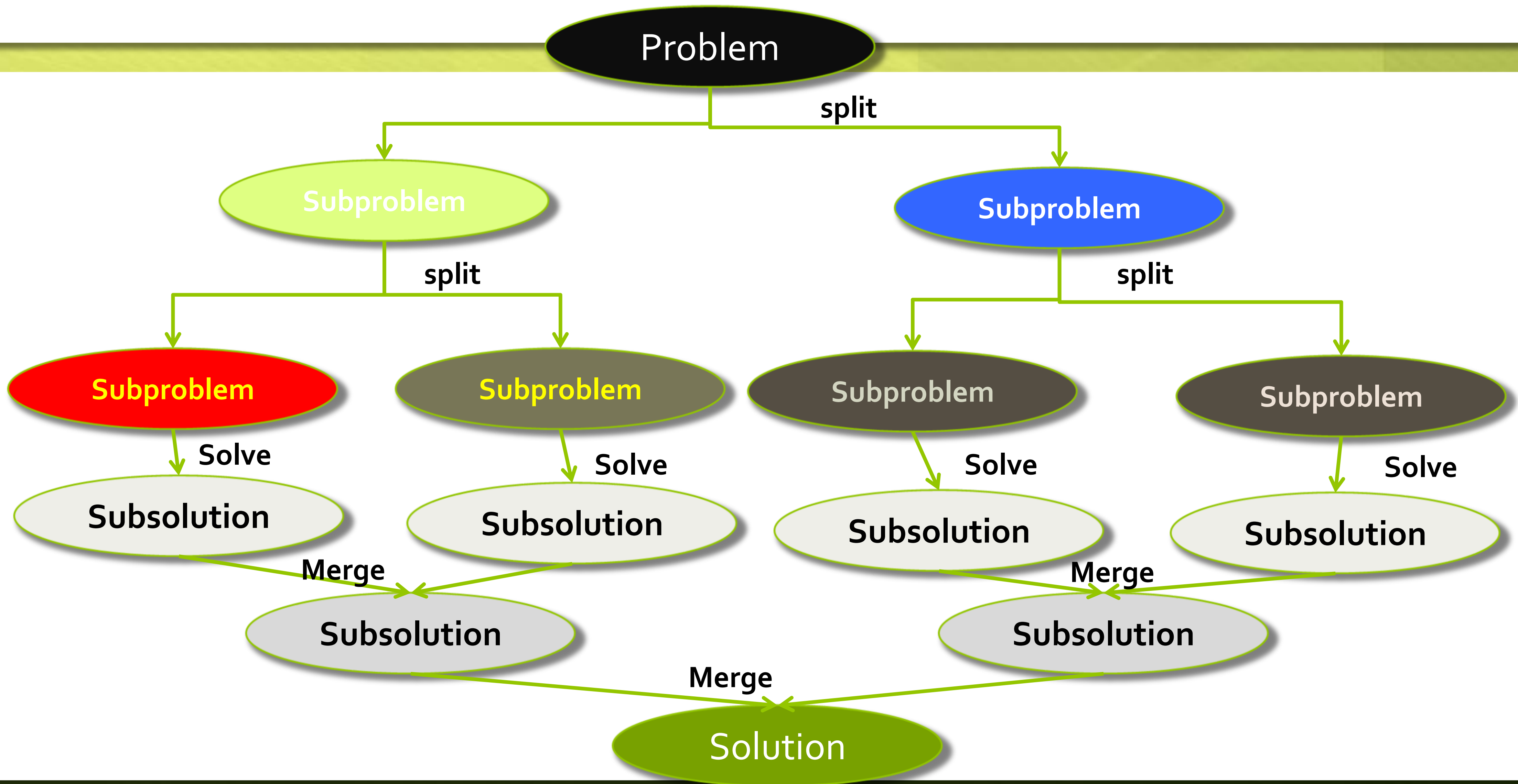
Algorithm Structures

- 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

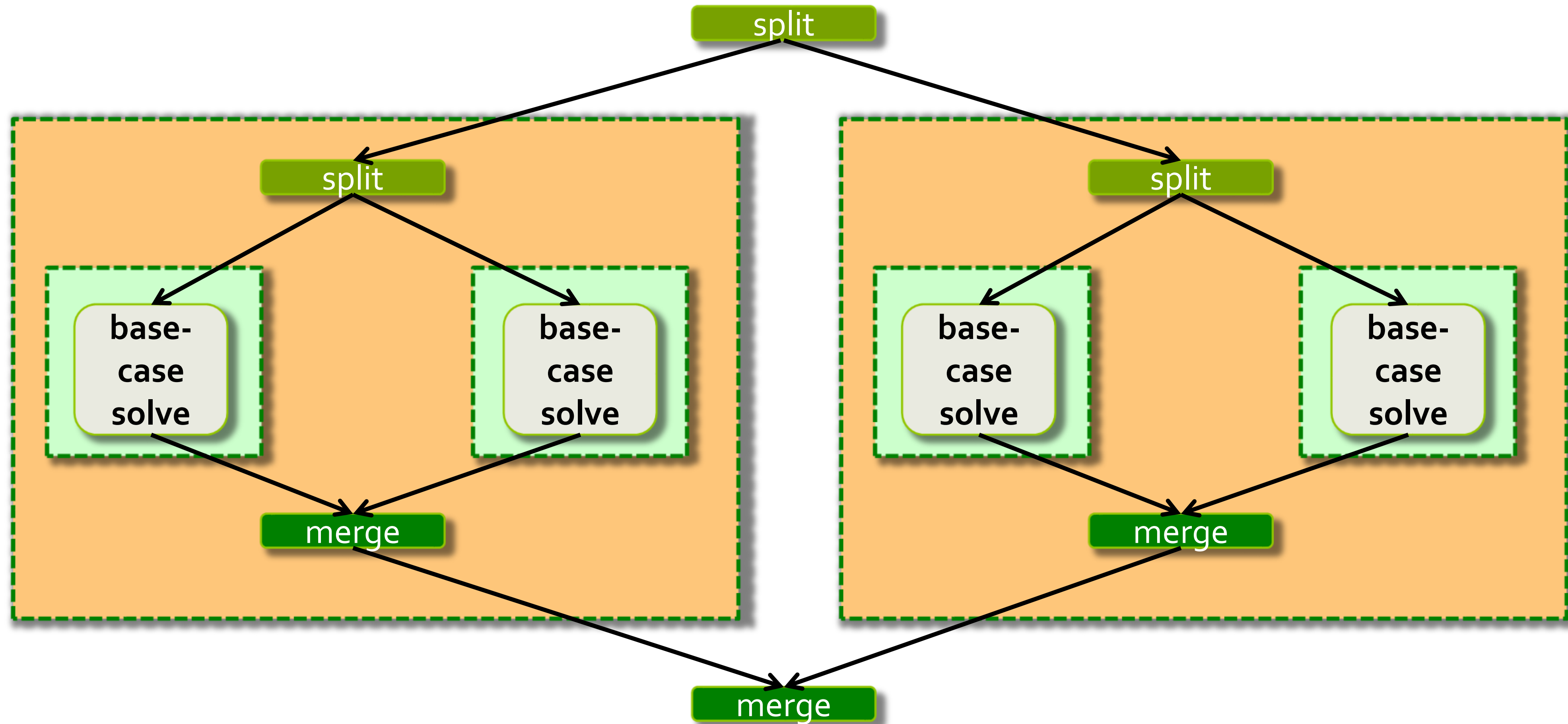
Algorithm Structure Decision Tree
(Major Organizing Principle)



Divide and Conquer Strategy



Divide and Conquer Parallel Strategy



Each dashed-line box represents a task

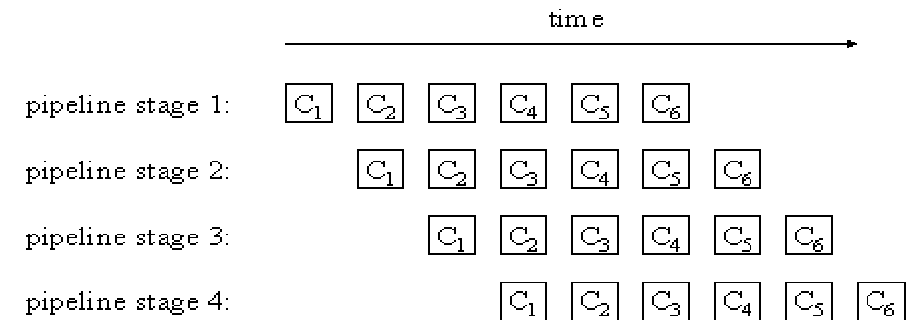
Recursive Data Strategy

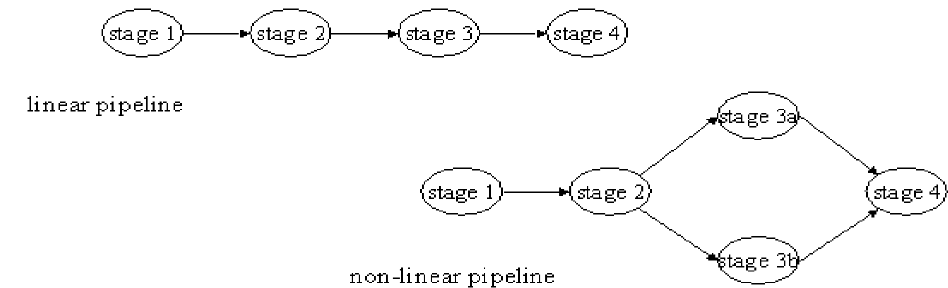
- Involves an operation on a recursive data structure that appears to require sequential processing:
  - Lists
  - Trees
  - Graphs
- Recursive Data structure is completely decomposed into individual elements.
- Structure in the form of a loop (top-level structure)
- Simultaneously updating all elements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

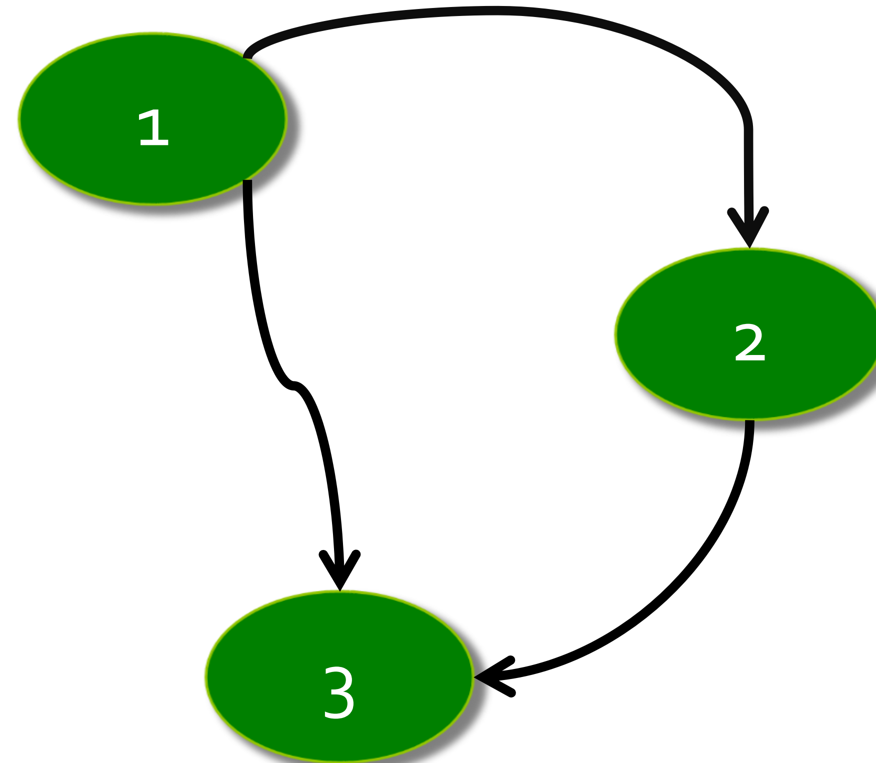
- Involves performing a calculation on many sets of data, where the calculation can be viewed in terms of data flowing through a sequence of stages
  - Instruction pipeline in modern CPUs
  - Vector Processing (Loop-level pipelining)
  - Algorithm-level Pipelining
  - Signal Processing
  - Graphics
  - 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: @SC3UIS

Or visit: www.sc3.uis.edu.co