Concurrency and Parallel Programming

Super Computacion y Calculo Cientifico UIS

Carlos Jaime Barrios Hernandez, PhD.

EISI UIS

@carlosjaimebh

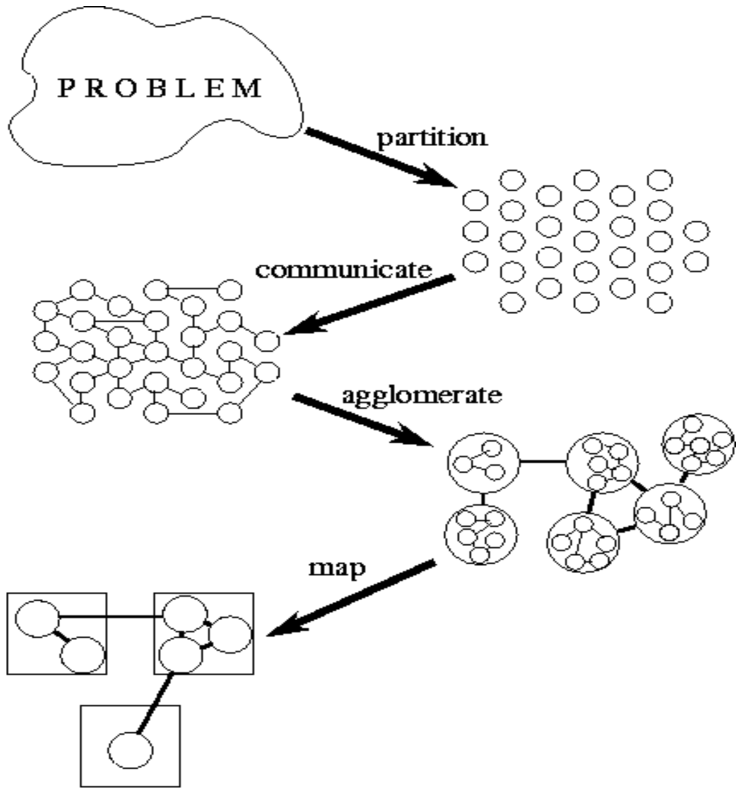
Concurrent and Parallel

La répétition sur la scène, 1874, Edgar Degas, Paris, Musée d'Orsay.

Plan

- The Traditional Way
- Design Spaces of Parallel Programming Recall
- Concurrent Programming
- Distributed Memory Vs. Shared Memory
- Design Models for Concurrent Algorithms
- Concurrent Algorithm Design Features and Forces
- Algorithm Structures
- Final Notes

Traditional Way



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

Design Spaces of Parallel Programming*

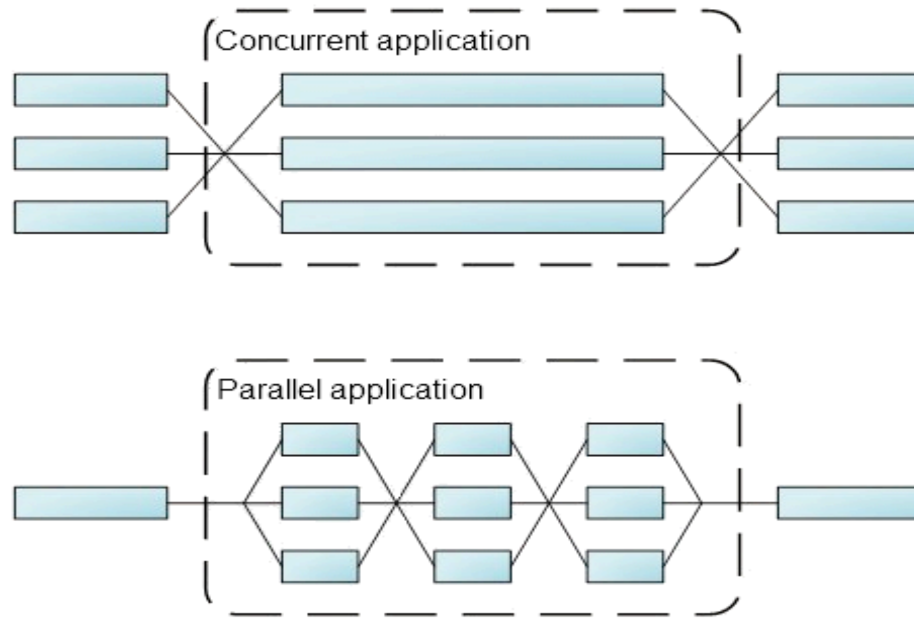
- FC — Finding Concurrency (Structuring Problem to expose exploitable concurrency)
- AS — Algorithm Structure (Structure Algorithm to take advantage of Concurrency)
- SS — Supporting Structures (Interfaces between Algorithms and Environments)
- IM — Implementation Mechanisms (Define Programming Environments)

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

Concurrency and Parallelism

- A system is “concurrent” if it can support two or more actions in progress at the same time

- A system is “parallel” if it can support two or more actions executing simultaneously



Concurrent Programming is all about independent computations that the machine can execute in any order.

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)

Distributed Vs. Shared Memory Programming

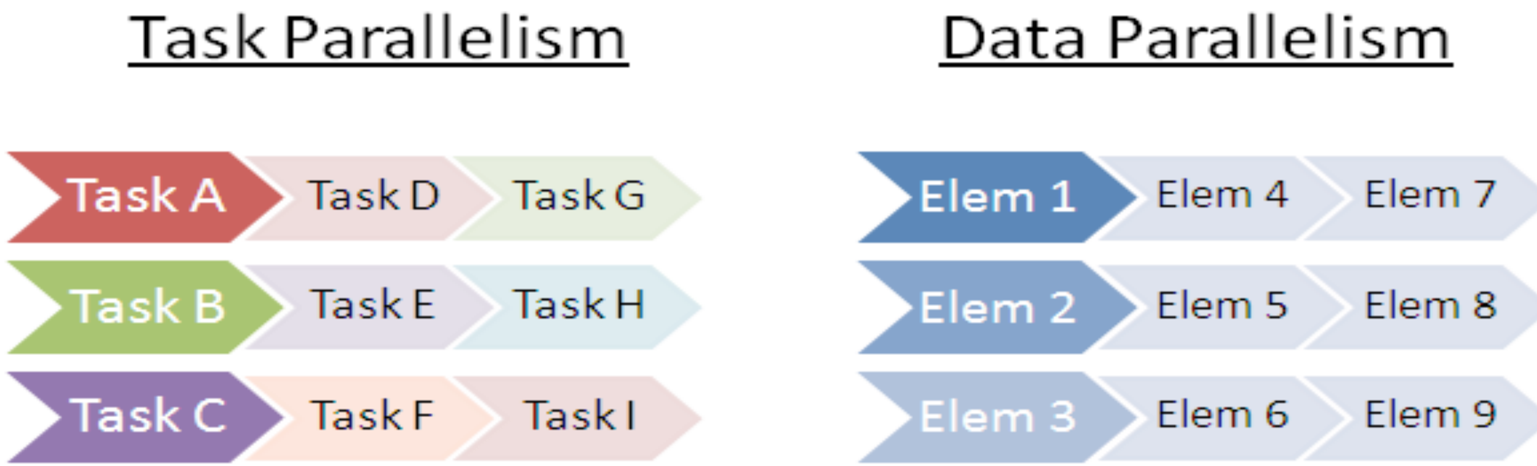
Common Features

- 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)

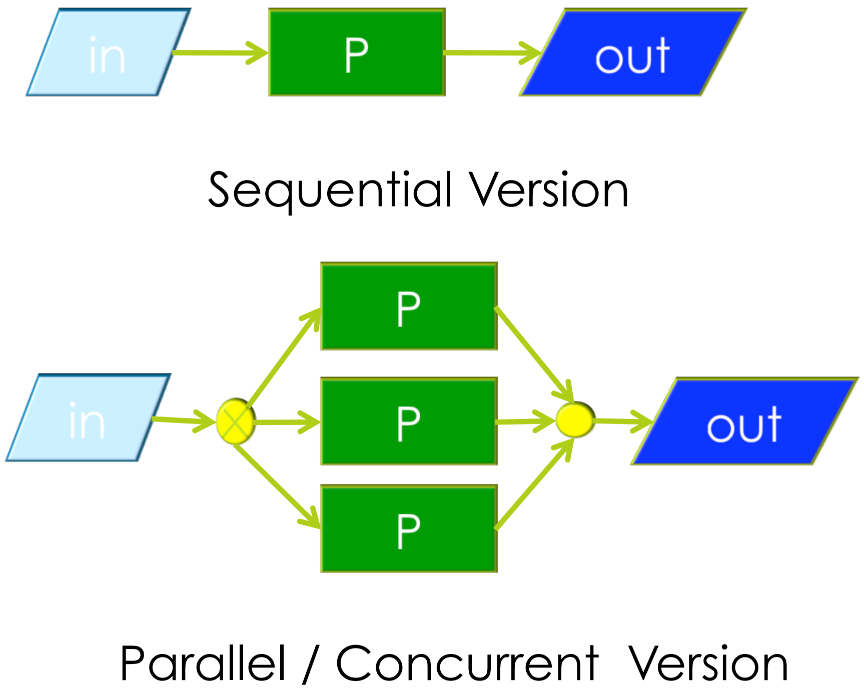
Tasks and Data Decomposition



- Tasks Decomposition : Task Parallelism
- Data Decomposition: Data Parallelism

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.

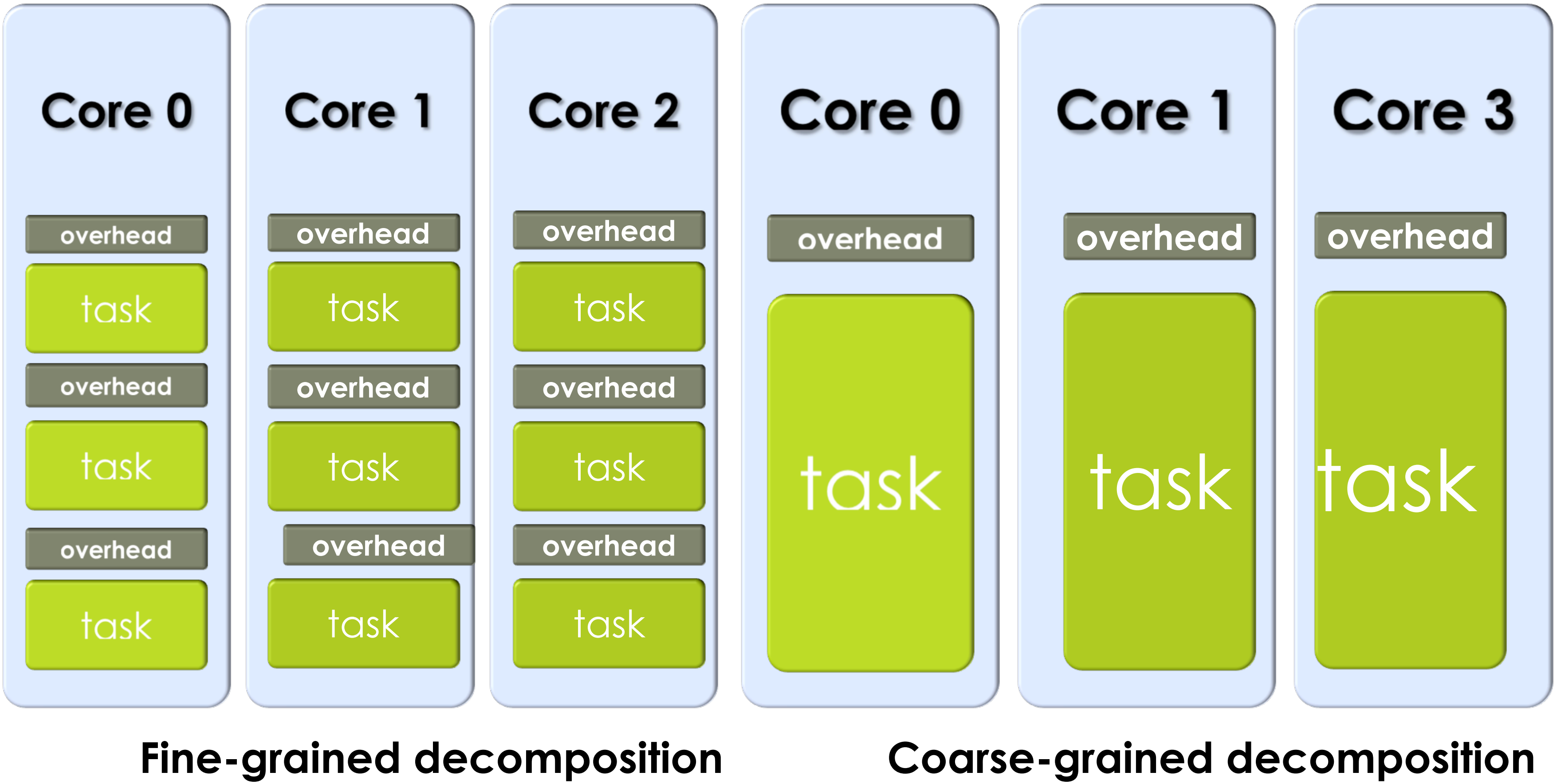


What are the tasks and how are defined?

- 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)

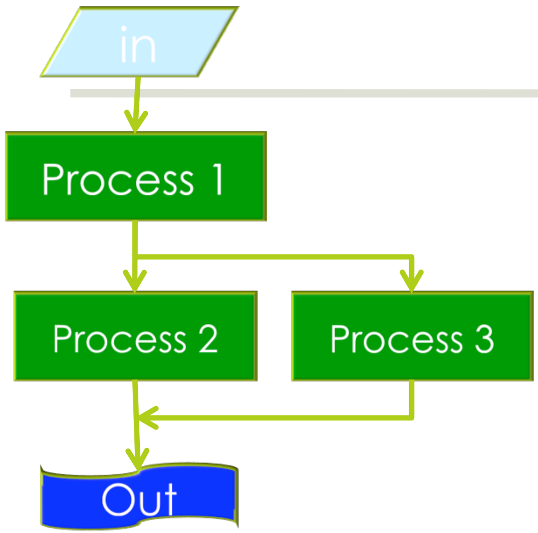
Granularity is the amount of computation done before synchronization is needed

Task Granularity

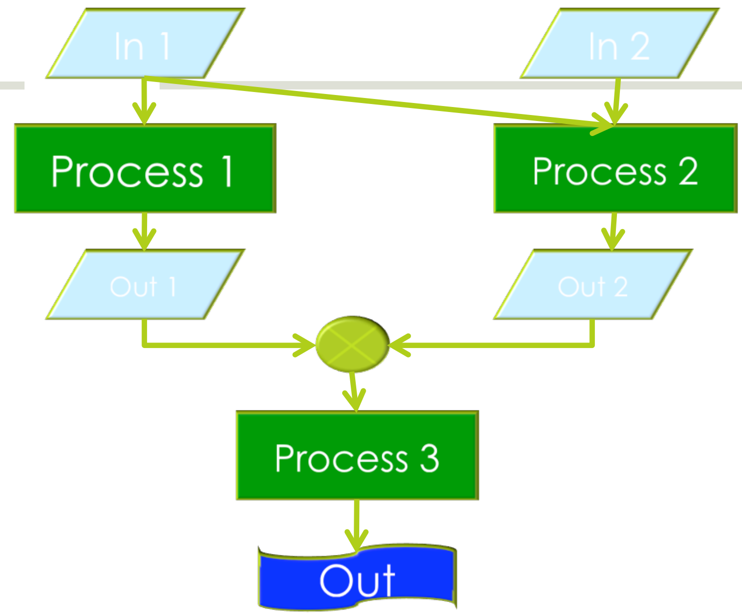


Task Dependencies

Order Dependency

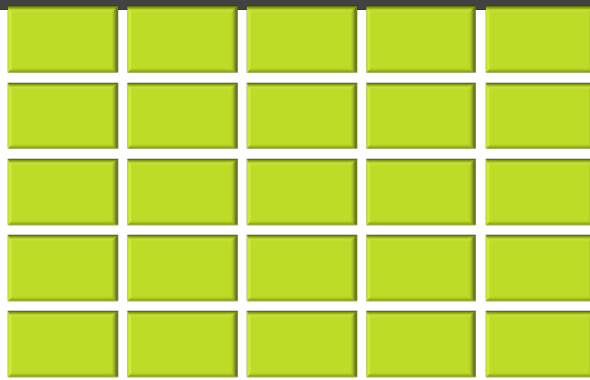


Data Dependency

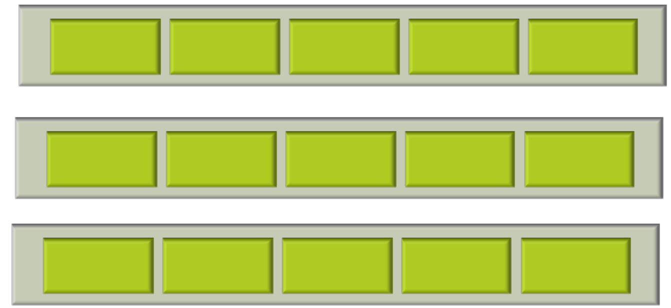


Enchantingly Parallel Code: Code without dependencies

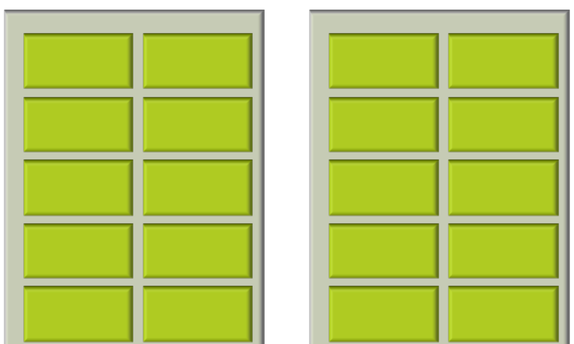
How should you divide data into chunks?



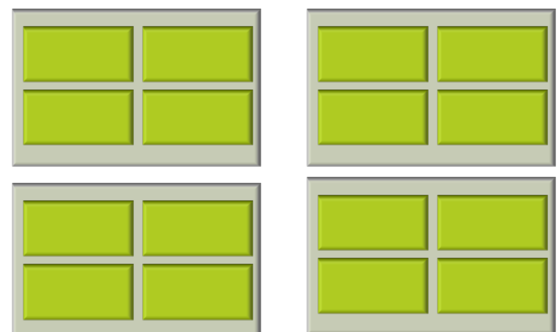
By individual elements



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.

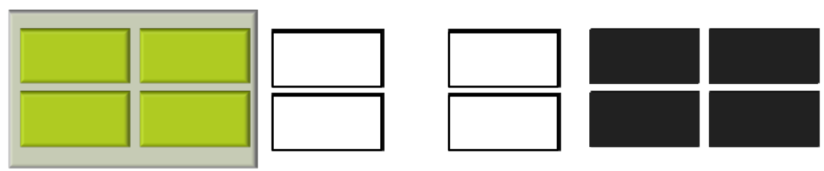
2 Shared Borders

5 Shared Borders

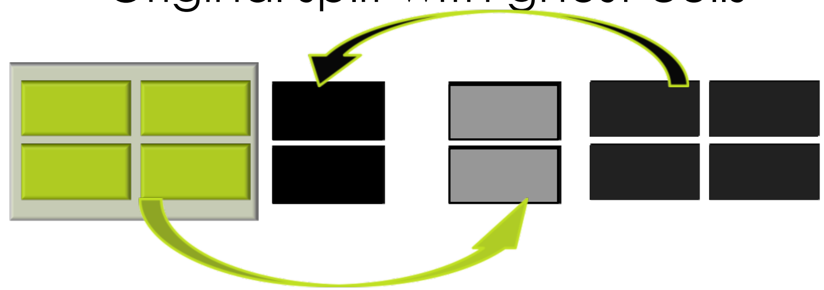
- Regular shapes : Common Regular data organizations.
- Irregular shapes: may be necessary due to the irregular organizations of the data.

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

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.

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.

Tasks and Domain Decomposition Patterns

- Task Decomposition Pattern
  - 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 Pattern
  - 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.

Group and Order Tasks Patterns

- Group Tasks Pattern
  - Simplify the problem dependency analysis
    - If a group of tasks must work together on a data shared structure
    - If a group of tasks are dependent
- Order Tasks Pattern
  - Find and correctly account for dependencies resulting from constraints on the order of execution of a collection of tasks.
    - Temporal dependencies
    - Specific Requirements of the tasks

Data Sharing Pattern

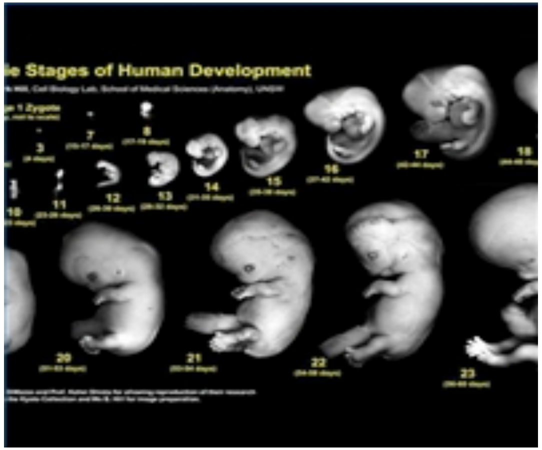
- Data decomposition might define some data that must be shared among the tasks.
- Data dependencies can also occur when one task needs access to some portions of the another task's local data.
  - Read Only
  - Effectively Local (Accessed by one of the tasks)
  - Read Write
    - Accumulative
    -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

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

Not Parallelizable Jobs, Tasks and Algorithms

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

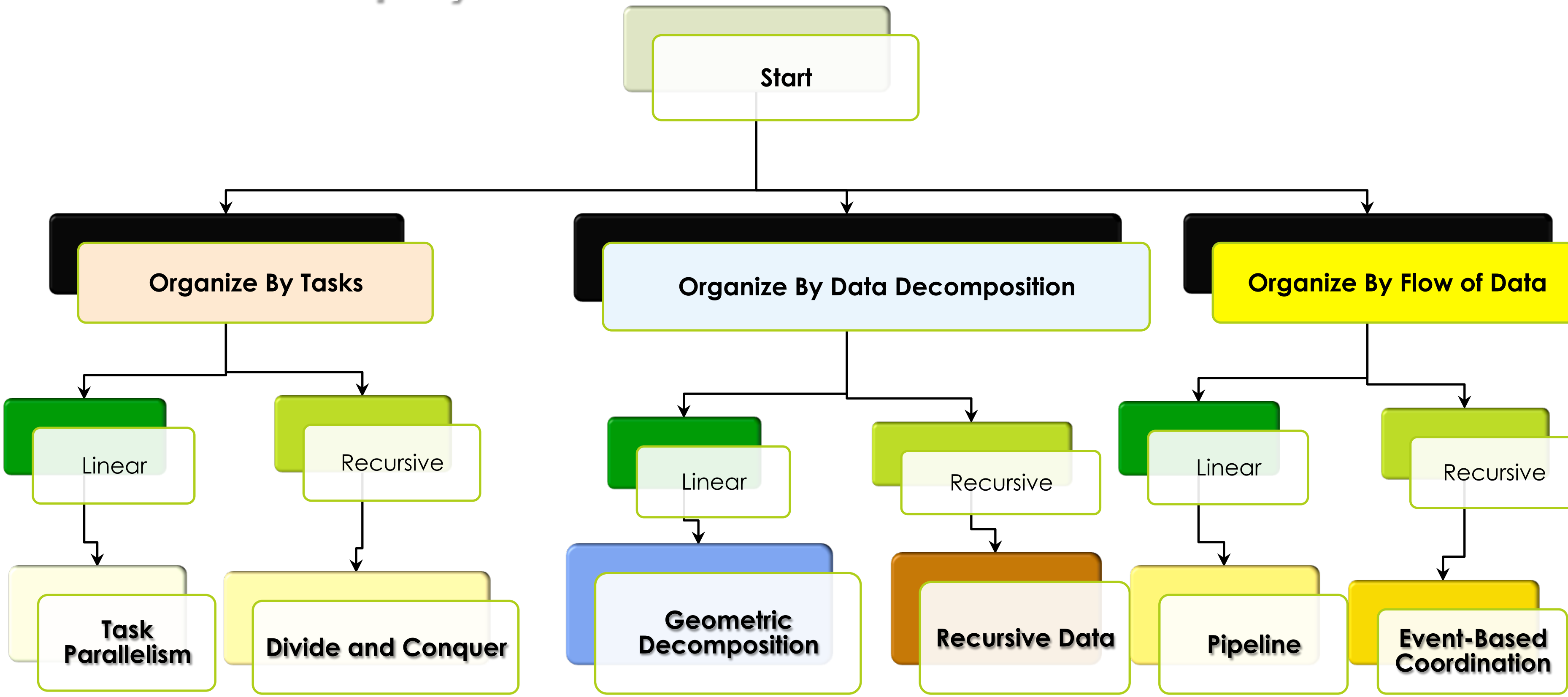


The Mythical Man-Month: Essays on Software Engineering. By Fred Brooks. Ed Addison-Wesley Professional, 1995

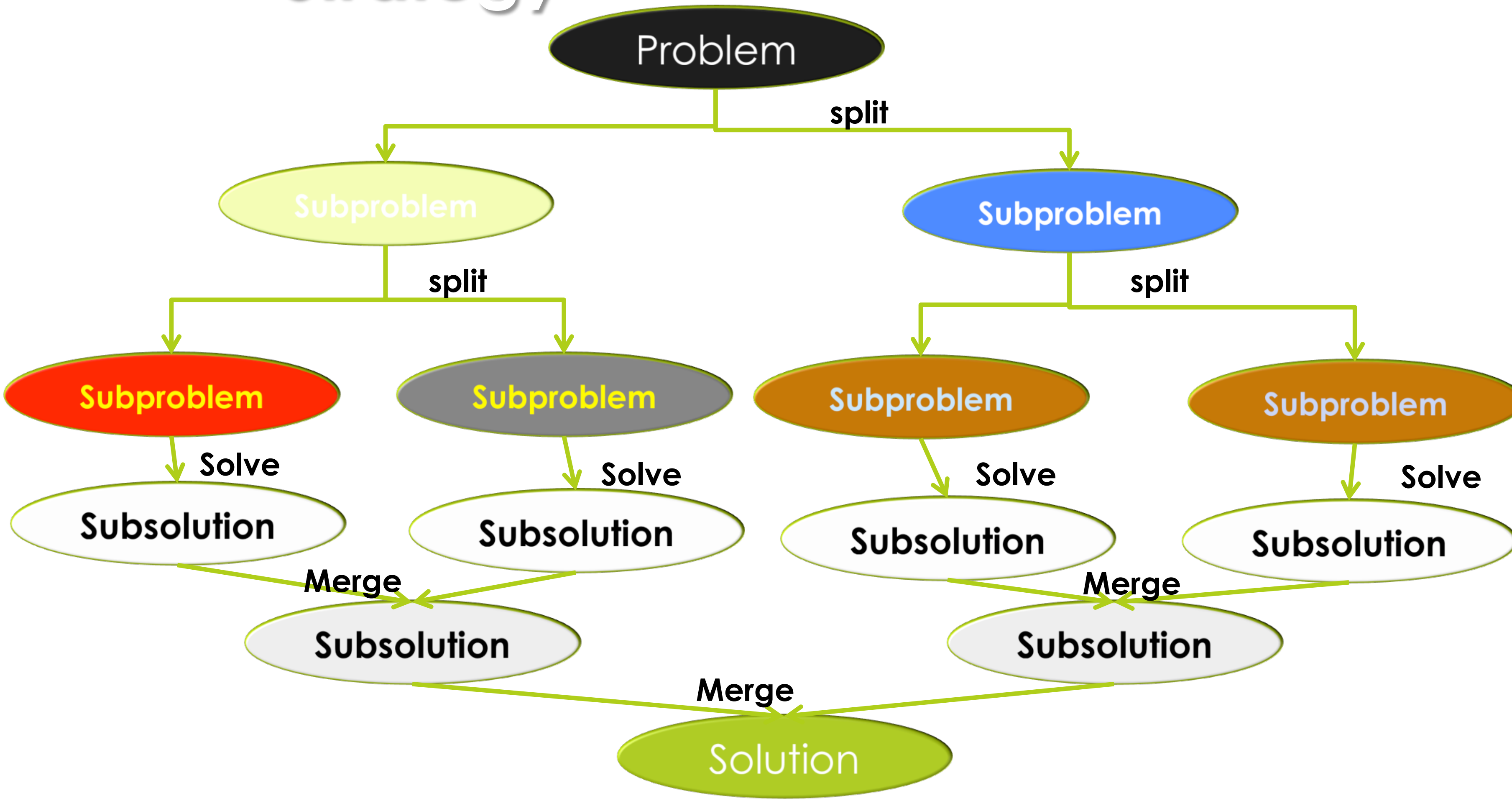
Algorithm Structures

- 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

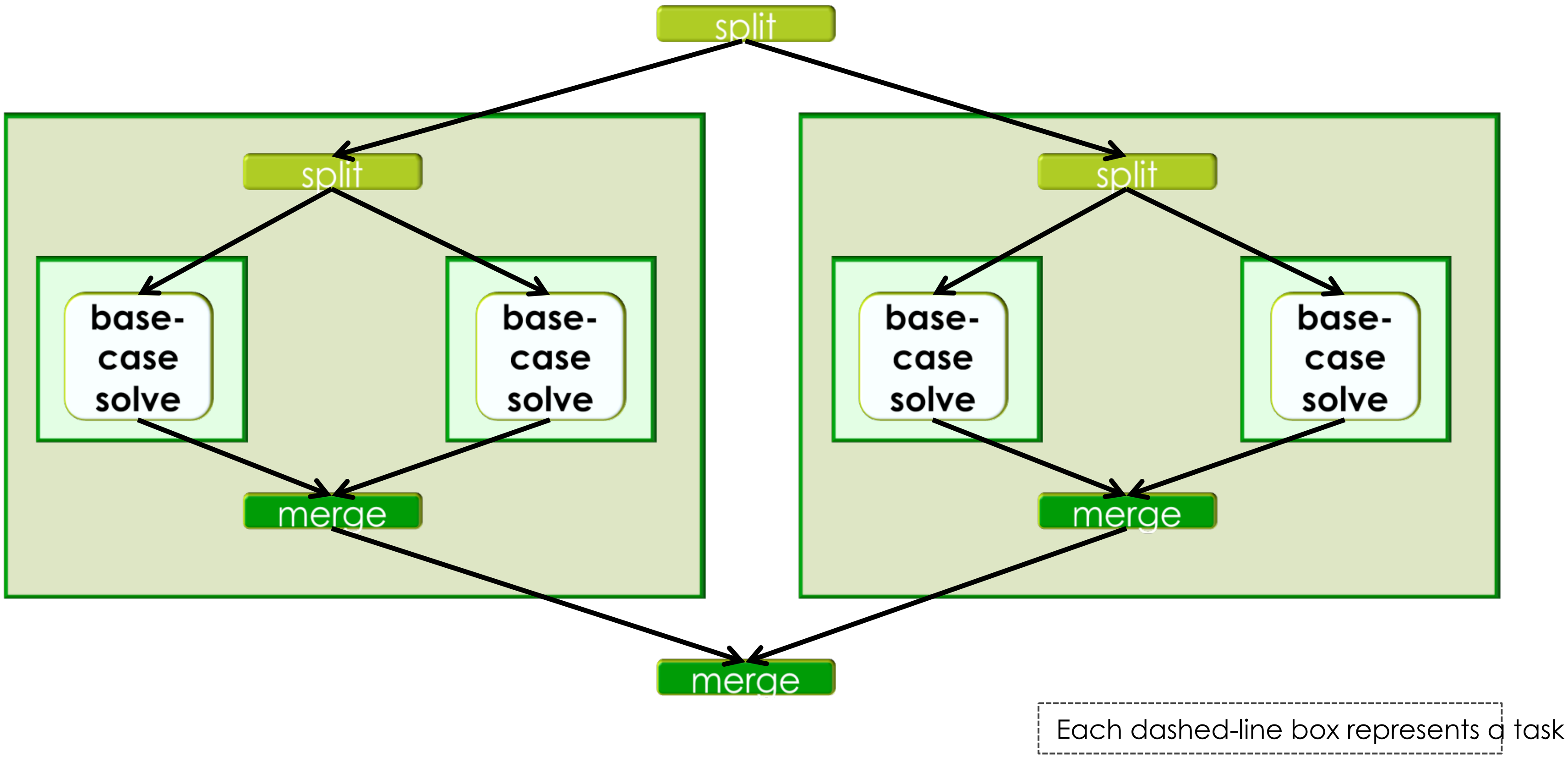
Algorithm Structure Decision Tree (Major Organizing Principle)



Divide and Conquer Strategy



Divide and Conquer Parallel Strategy

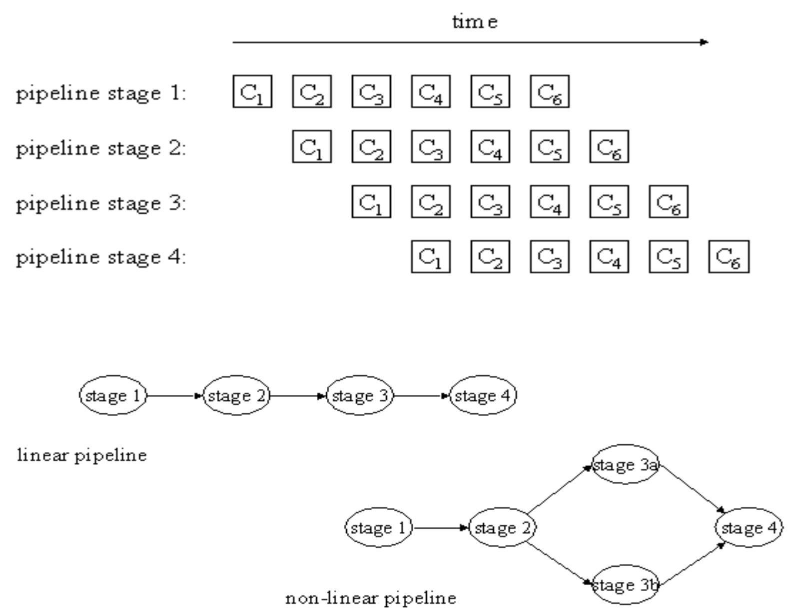


Recursive Data Strategy

- 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.

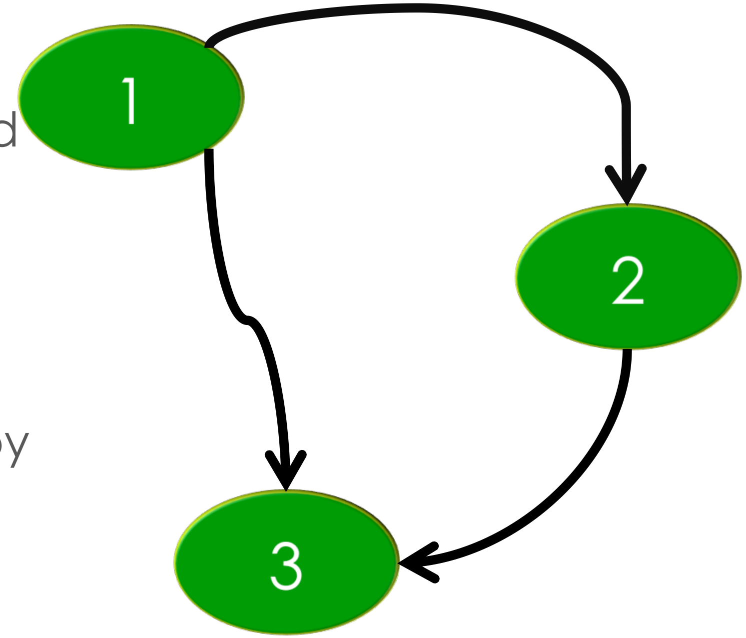
Pipeline Strategy

- 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



Final Notes

- Every Parallel Algorithm involves a collection of tasks that can execute concurrently
  - The key is finding tasks (and collect them)
- Data-based decomposition is good if:
  - The most computationally intensive part of the problem is organized around the manipulation of a large data set structure.
  - Similar operations are being applied to different parts of the data structure with independency.
- However the desired features of a concurrent/parallel program (efficiency, simplicity, portability and scalability):
  - Efficiency conflicts with portability
  - Efficiency conflicts with simplicity
- Thus a good algorithm design must strike a balance between abstraction and portability and suitability for a particular target architecture.

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/