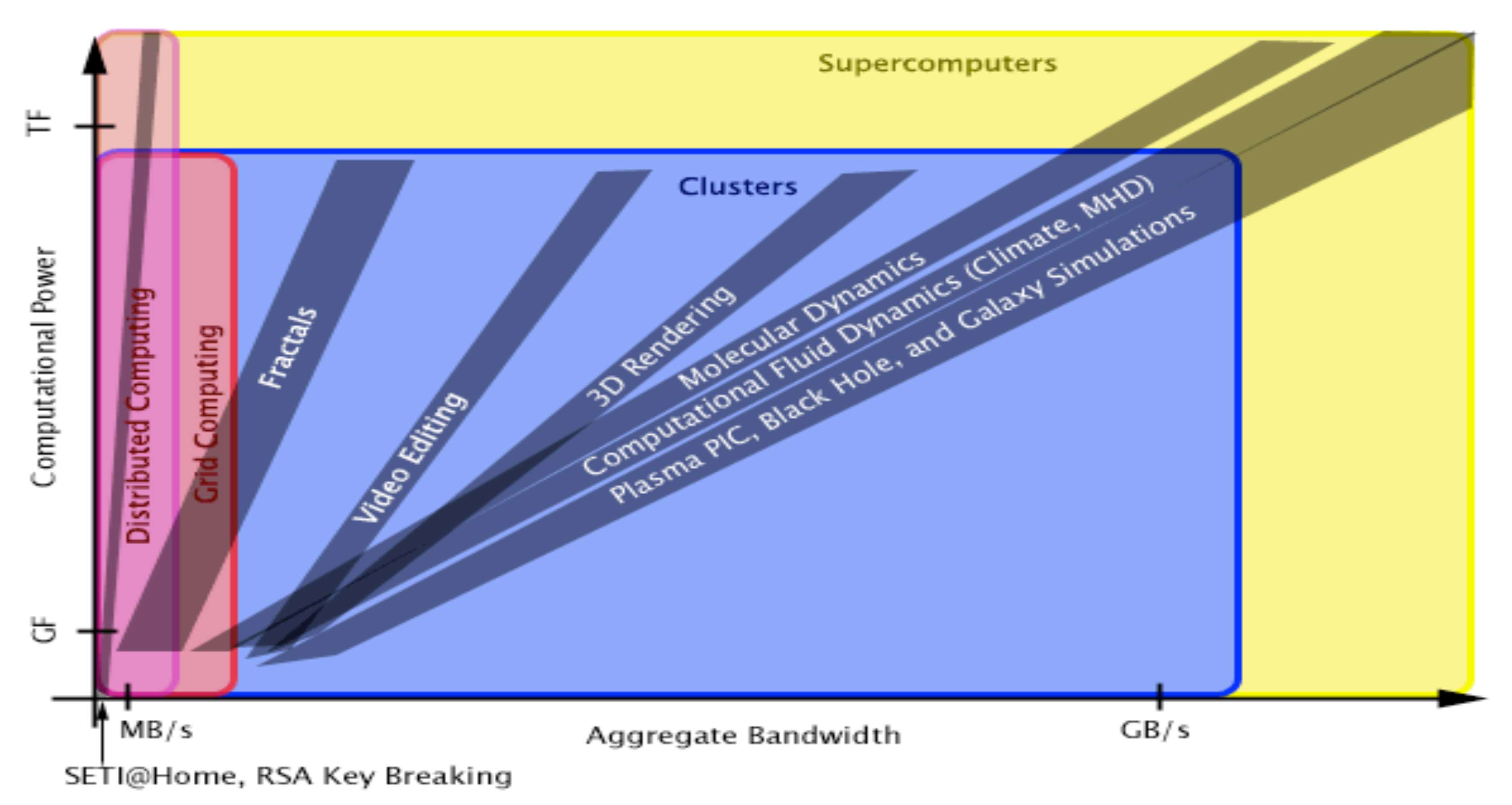
Parallel Programming Environments and Parallel Programming Computation

Carlos Jaime Barrios Hernandez, PhD.

SC-CAMP

Turrialba, Costa Rica 2011

"The exact region a parallel application occupies depends on the problem".



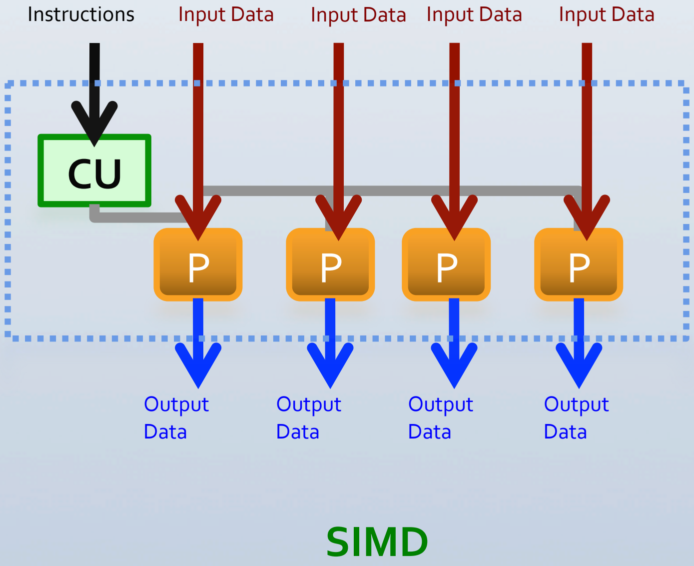
From http://daugerresearch.com

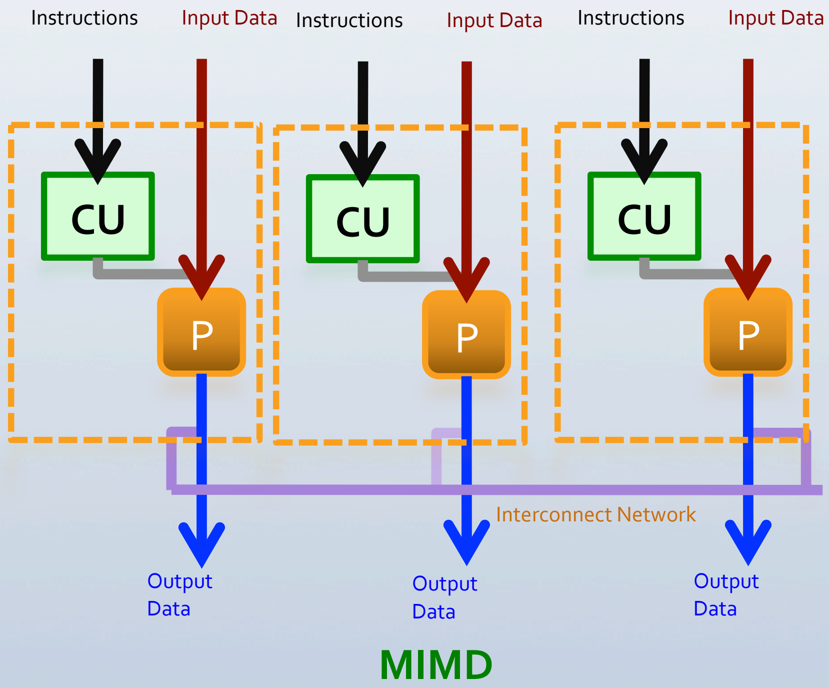
Plan

- Architectural Considerations to Parallel Programming (PP)
- Parallelism
- PP Models
- PP Quantitative Considerations
- Problem Decomposition
- Algorithm Selection
- PP Design Spaces
- Final Notes

Architectural Considerations to PP

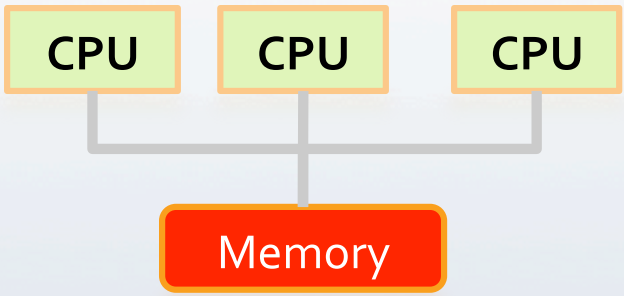
- Remember "concurrency": it exploits better the resources (shared) within a computer.
- Exploit SIMD and MIMD Architectures



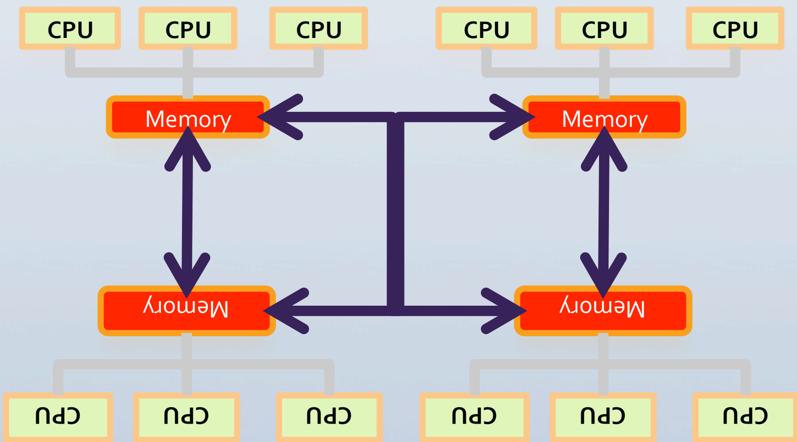


Architectural Considerations to PP

Shared Memory

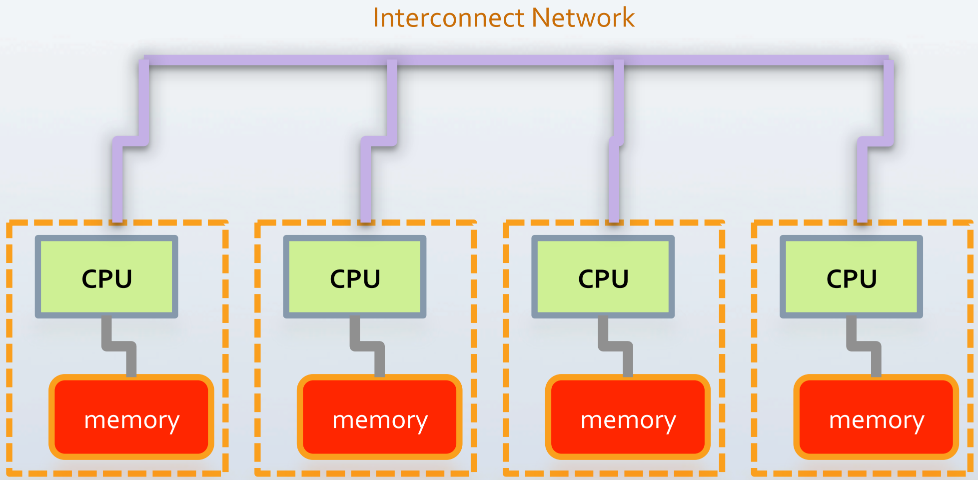


SMP



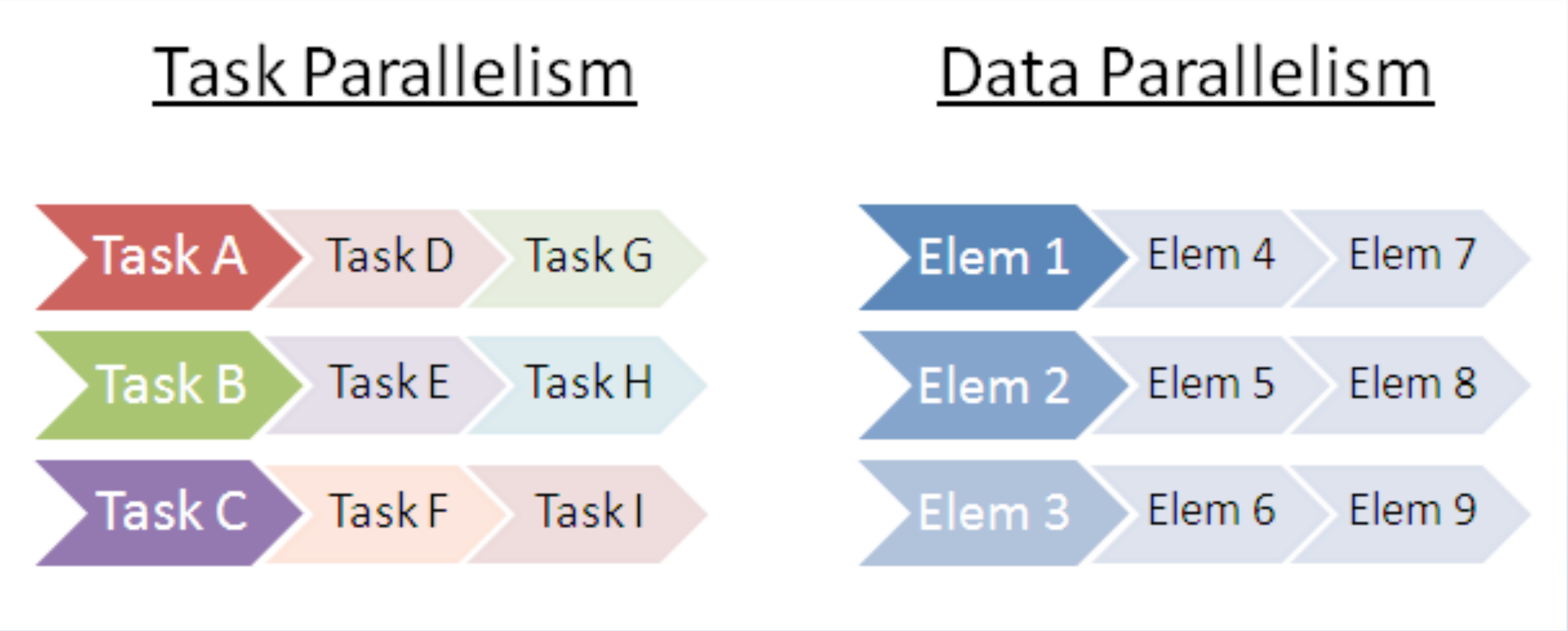
NUMA

- Distributed (Shared) Systems



MPP / Clusters

Parallelism



Parallel Programming Models

- PP Environments
  - Basic Tools
  - Language Features
  - APIs (Application Programming Interfaces)
- Parallel Programs are extremely dependents of Parallel Systems (Architectures)
  - Often, Parallel Programs are not portable between parallel computers

PP Models

- Shared Memory
  - OpenMP

- Distributed Memory (Shared)
  - MPI

- Hybrid Combination
  – OpenMP + MPI
  – CUDA, OpenCL, JAVA

Common Problems:

• ***Thrashing***: Simultaneous Processors are simultaneous Attempting to write into the same cache line, which can cause the cache link to ping pong between two different caches.
•***False Sharing***: Data Structures are attended in the same cache blocks
•***Cache coherence*** (Overloading)
•***Data Placement*** (Overlapping)
•***Latency Growing*** (in accordance with the distance among processors)

Some Definitions (Reminder)

- Task: Sequence of Instructions that operates together as a group.
- Unit Execution (UE): Unit where a task is mapped (Process or Thread)
  - Process: Collection of Resources that enables the execution of program instructions
- Processing Elements (PEs): Hardware element that executes a system of instructions

Some Concepts (To Explore)

- Load Balancing: (In terms of distribution among PEs)
  - Static or Dynamic
- Synchronization: (In terms of Task and Process)
  - Synchronous or Asynchronous
- Race Condition:
  - Outcome changes as the relative UE scheduling
- Deadlock:
  - Each task of the cycle of tasks is blocked waiting for another proceed.

PP Quantitative Bases

- Total Running Time (Sequential-Serial View):

$$T_T(1) = T_{setup} + T_{compute} + T_{finalization}$$

- Total Running Time (Parallel Idealized View):

$$T_T(P) = T_{setup} + \frac{T(1)_{compute}}{P} + T_{finalization}$$

PP Quantitative Considerations

- Relative Speedup (S): How much faster a problem runs in a way that normalizes away to running time.
- Efficiency: Speedup Normalized by the number of PEs.
  - Perfect Linear Speedup: S(P) = PEs.
- Serial Terms: Terms that cannot be run concurrently.
- Serial Fraction (γ): Running times of the serial terms.
  - Parallelizable fraction of a program is (1 –γ)

Amdahl's and Gustafson's Laws

- Amdahl's law:
  - Speedup application due to parallel computing is limited by the sequential part of the application.
    - Motivates task-level parallelization
- Gustafson's law:
  - Any sufficiently large problem can be effectively parallelized.

John L. Gustafson: Reevaluating Amdahl's law. In Communication of ACM, Volume 31(5) 532-533, 1988

Parallel Programming and Computation Thinking

- PP Goals (Reminder)
  - Solve a problem in less time.
  - Solve bigger problems in a determined time.
  - Achieve better solutions for problems in a determined time.

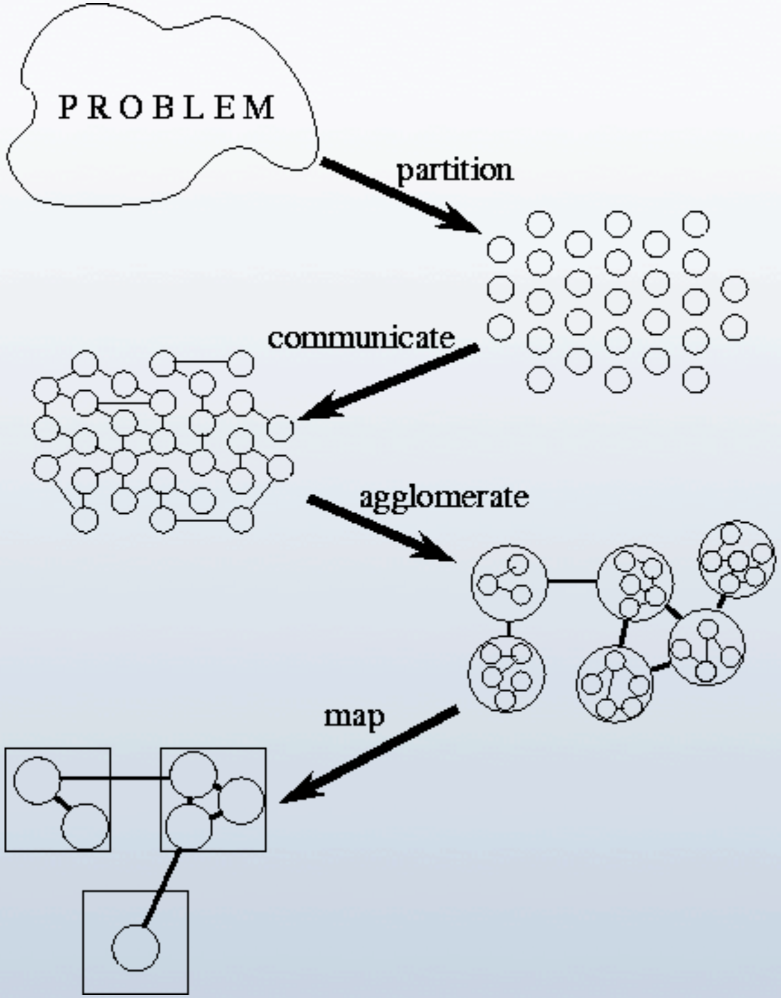
- Good Candidates for Parallel Computing (Reminder):
  - Large Problem Sizes (Large Scale Problems)
  - High Complexity
  - High Modeling

Programmers "build code" and "organize data" to solve subproblems concurrently.

Computational Thinking thought process of formulating domain problems in terms of computation steeps and algorithms.

Methodical Design of Parallel Algorithms

- *Partitioning:* Decomposition into small tasks.
- *Communication:* Coordination of task executions.
- *Agglomeration.* Task and communication structures defined in the first two stages of a design are evaluated with respect to performance requirements and implementation costs.
- *Mapping.* Each task is assigned to a processor in a manner that attempts to satisfy the competing goals of maximizing processor utilization and minimizing communication costs.



From http://www.mcs.anl.gov/~itf/dbpp/

Problem Decomposition

- Decomposing calculation work into units of parallel executions
- Finding parallelism in large computational problems is often conceptually simple but can turn out to be difficult in practice.
- Key is to identify the work to be performed by each unit of parallel executions
  - Inherent parallelism of the problem is well utilized

Problem Decomposition

- Decomposition is necessary to mapped
- Decomposition defines granularity
- Decomposition produces:
  - Latency Costs
  - More Complexity (In general terms)
  - Load balancing needs
  - Scheduling needs (To allocate, communicate, transferring)
  - Data Management needs

Algorithm Selection

- Algorithm Properties
  - Definiteness:
    - Each step is precisely stated.
  - Effective Computability
    - Each step can be carried out by a computer.
  - Finiteness
    - Algorithm must be guaranteed to terminate.
- Algorithm Features
  - Number of steeps
  - Degree of parallel execution
  - Bandwidth (Among Process)

Algorithm Selection

- Computer Architecture
  - Memory Organization, Caching and locality, Memory Bandwidth, Instructions mode (single-instruction or multiple thread or single programs), data classes (single data or multiple data executions), floating point precision, accuracy (tradeoffs between algorithms).
- Programming Models and Compilers
  - Parallel Execution Models, Types of available memories, array data layouts, loop transformations (Data structures and loop strructures)

Algorithm Selection

- Algorithm Techniques
  - Tiling, cutoff, binning (scalability, efficiency)
- Domain Knowledge
  - Numerical Methods, Models, accuracy requirements, mathematical properties, phenomena/problem knowledge (Application of Algorithms)

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

Finding Concurrency

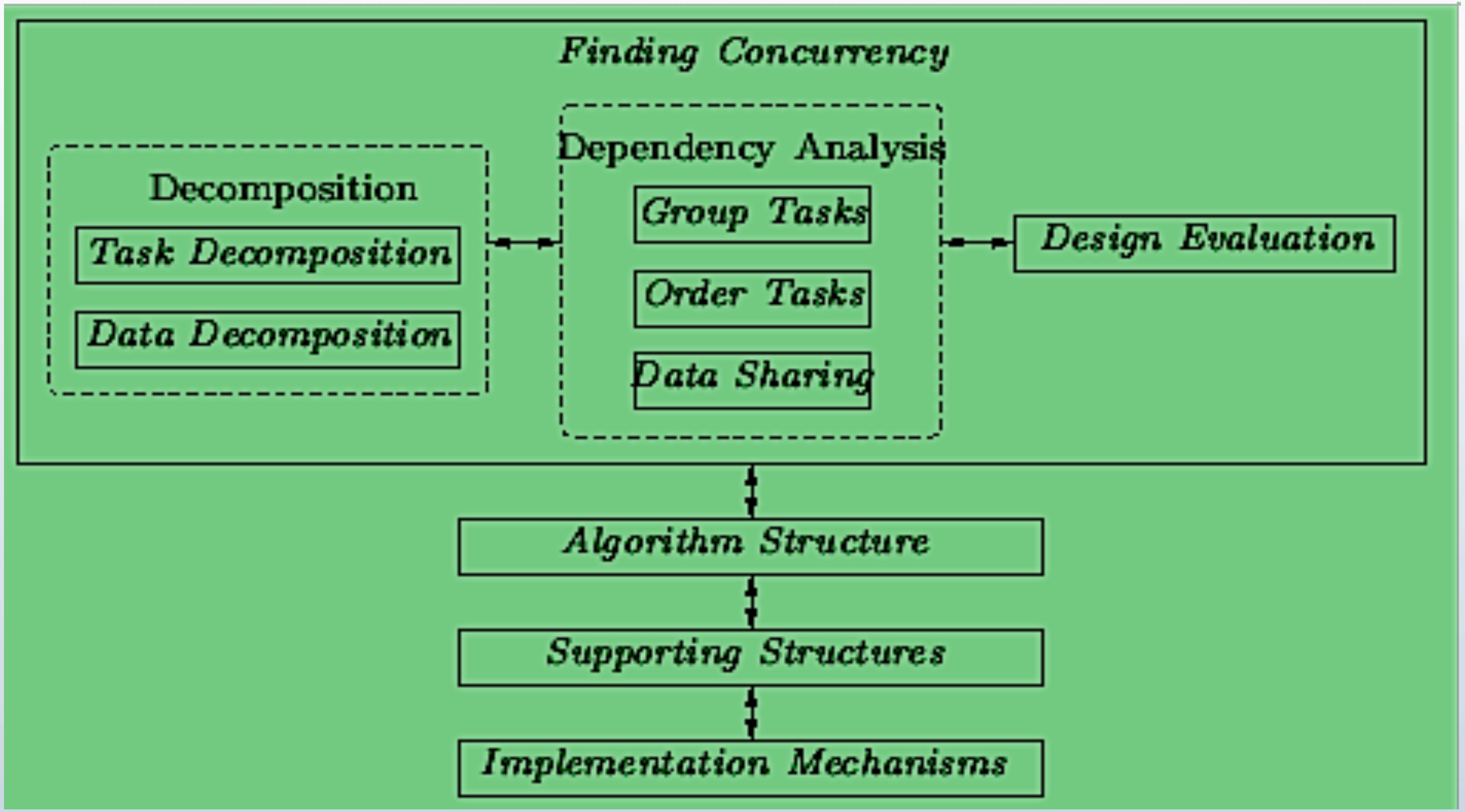


Image from: Patterns for Parallel Programming, Timoty Mattson, Beverly A. Sanders and Berna L. Massingill, Software Pattern Series, Addison-Wesley 2004

Algorithm Structure

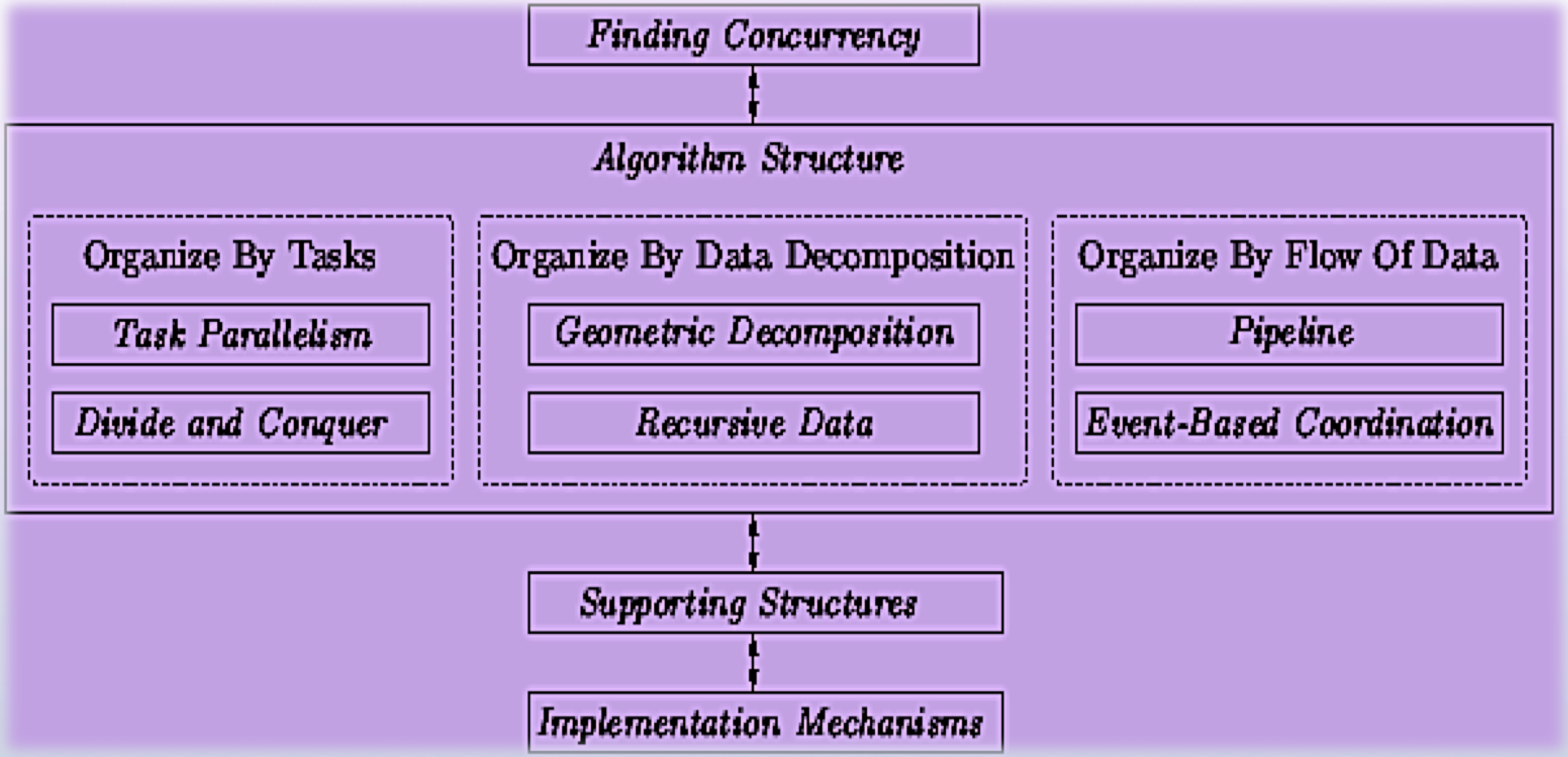


Image from: Patterns for Parallel Programming, Timoty Mattson, Beverly A. Sanders and Berna L. Massingill, Software Pattern Series, Addison-Wesley 2004

Supporting Structures

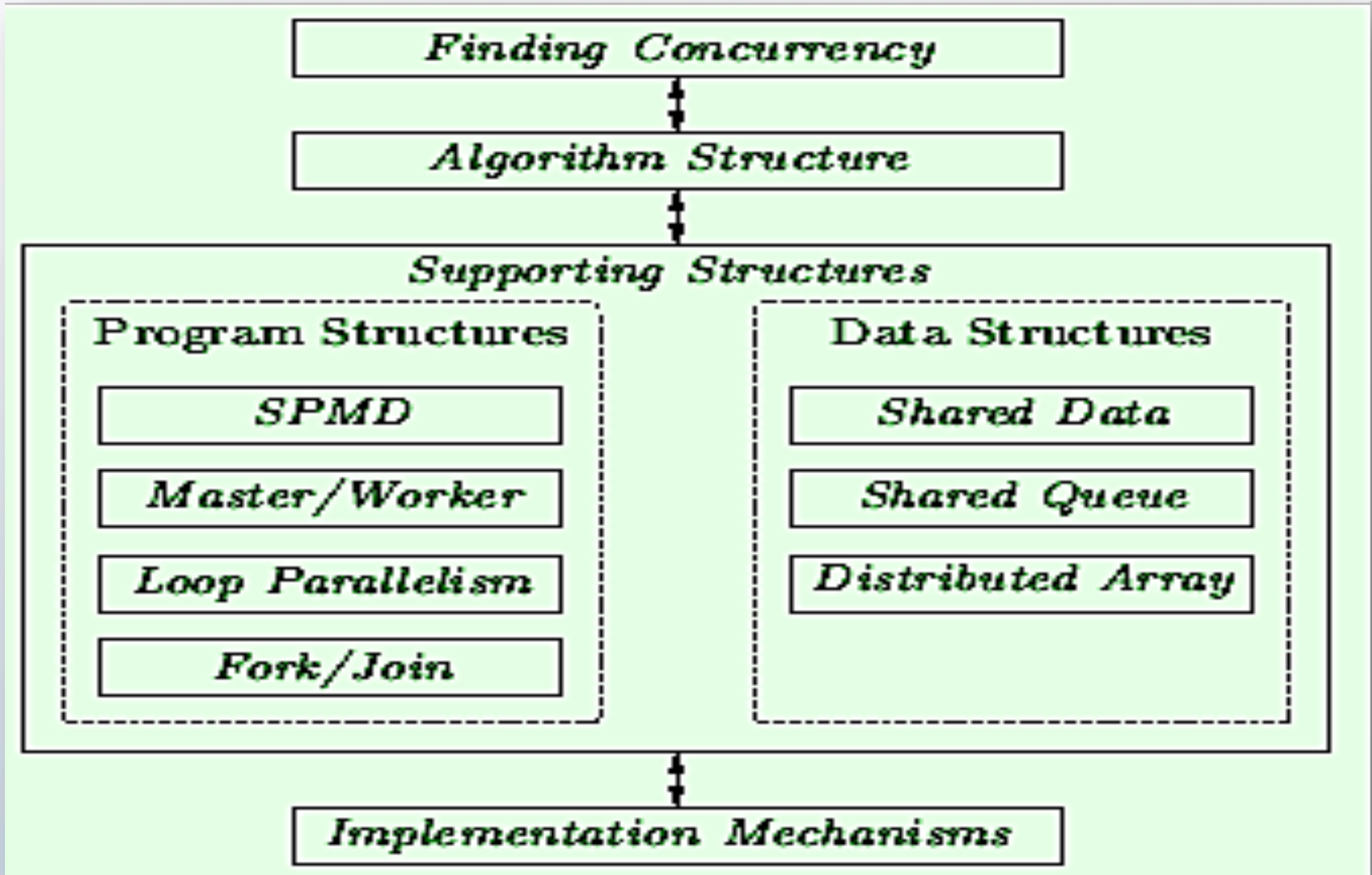


Image from: Patterns for Parallel Programming, Timoty Mattson, Beverly A. Sanders and Berna L. Massingill, Software Pattern Series, Addison-Wesley 2004

Implementation Mechanisms

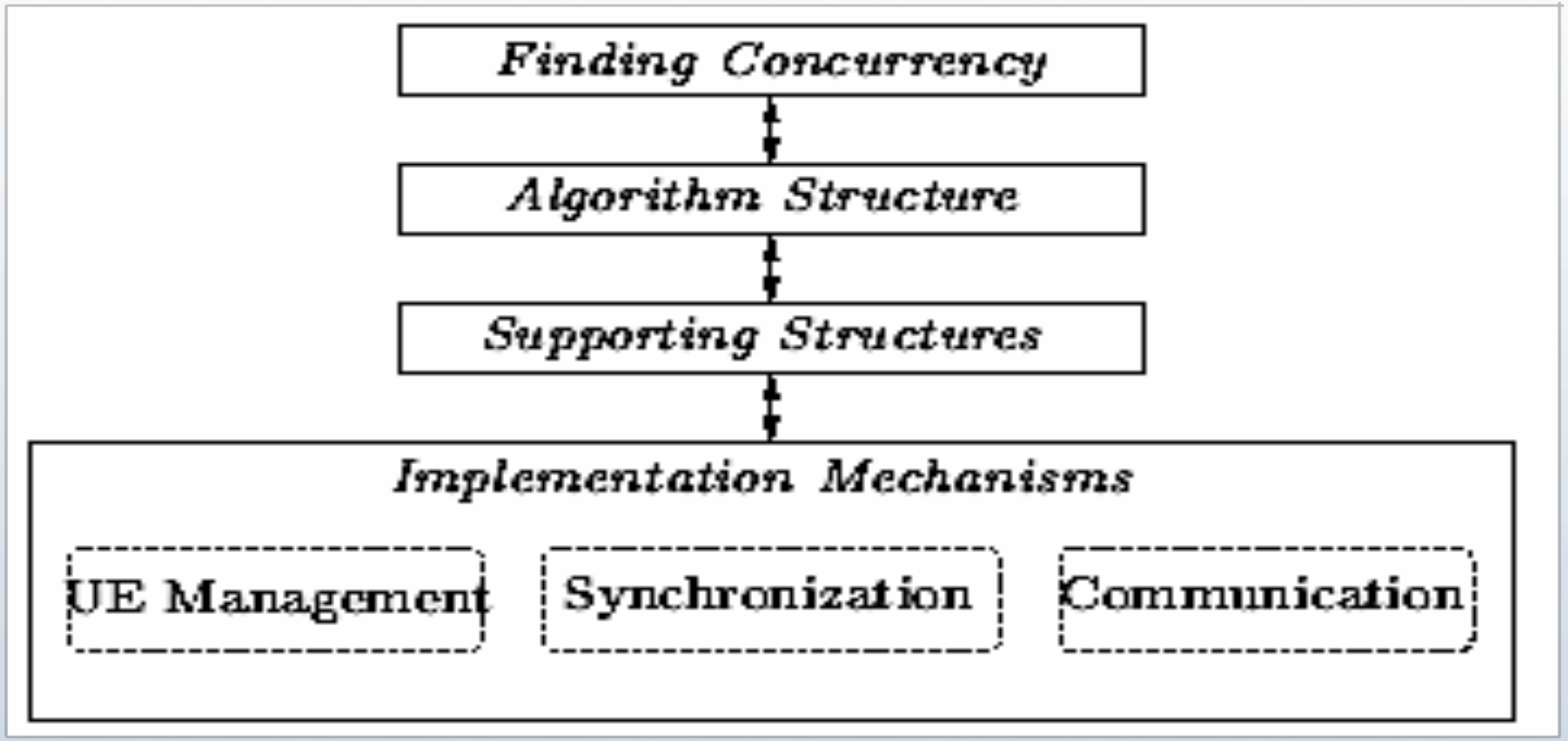


Image from: Patterns for Parallel Programming, Timoty Mattson, Beverly A. Sanders and Berna L. Massingill, Software Pattern Series, Addison-Wesley 2004

Final Notes

- Often, Computer Science is Parallel Computing
- The key to parallel computing is exploit concurrency
  - Concurrency was first exploited in Computing to better utilize or share resources within a computer
- Same, if parallel computing brooks paradigms, a methodological design is necessary to build algorithms and code programs

Recommended Lectures

- **Designing and Building Parallel Programs**, by *Ian Foster in http://www.mcs.anl.gov/~itf/dbpp/*
- **Patterns for Parallel Programming**, *by Timothy G. Mattson, Beverly A. Sanders and Berna L. Massingill. Software Patterns Series, Addison Wesley Ed., USA. 2009.*