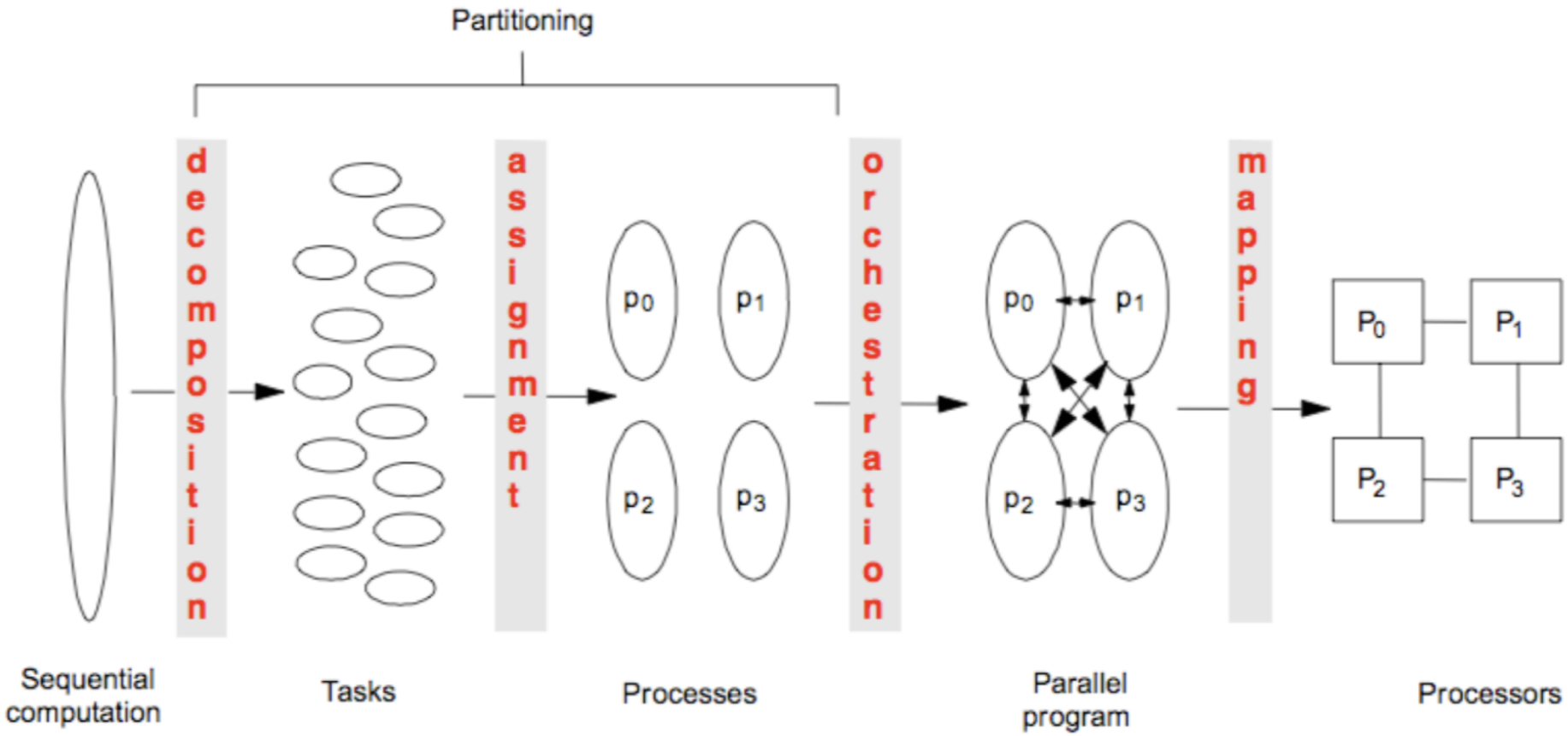
Parallel Algorithm Design

Parallel Algorithm Design

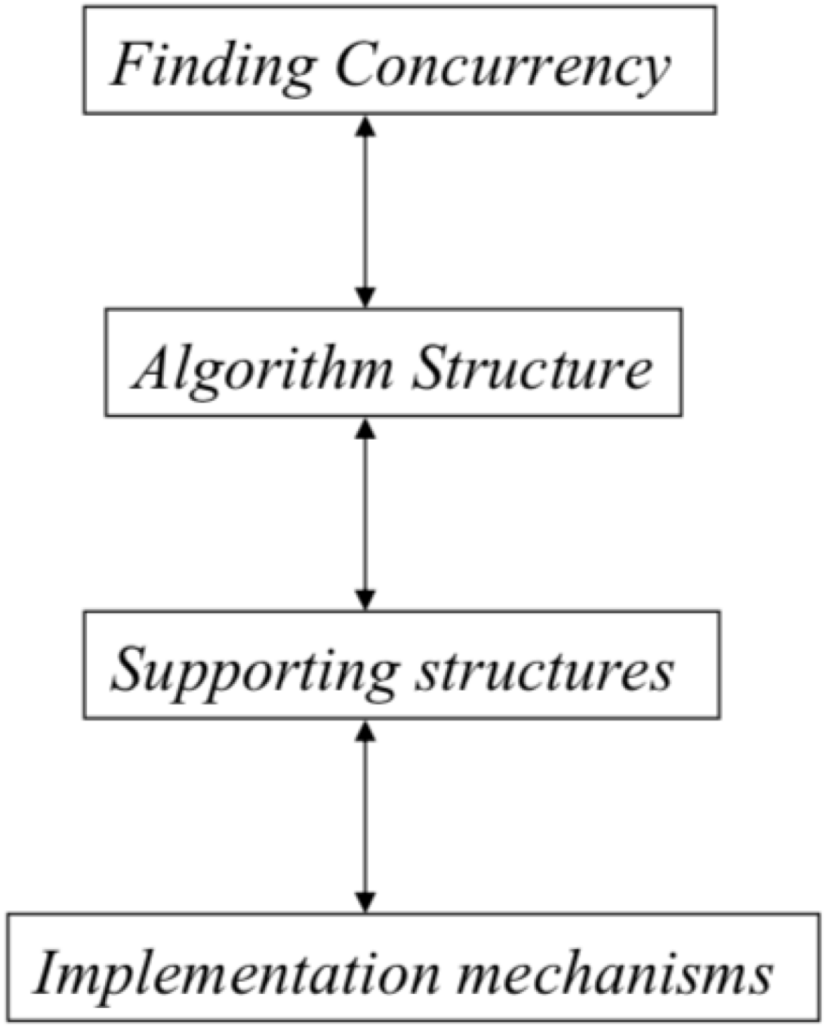


1. Study problem or code

2. Look for parallelism opportunities

3. Try to keep all cores busy doing useful work

Parallel Algorithm Design



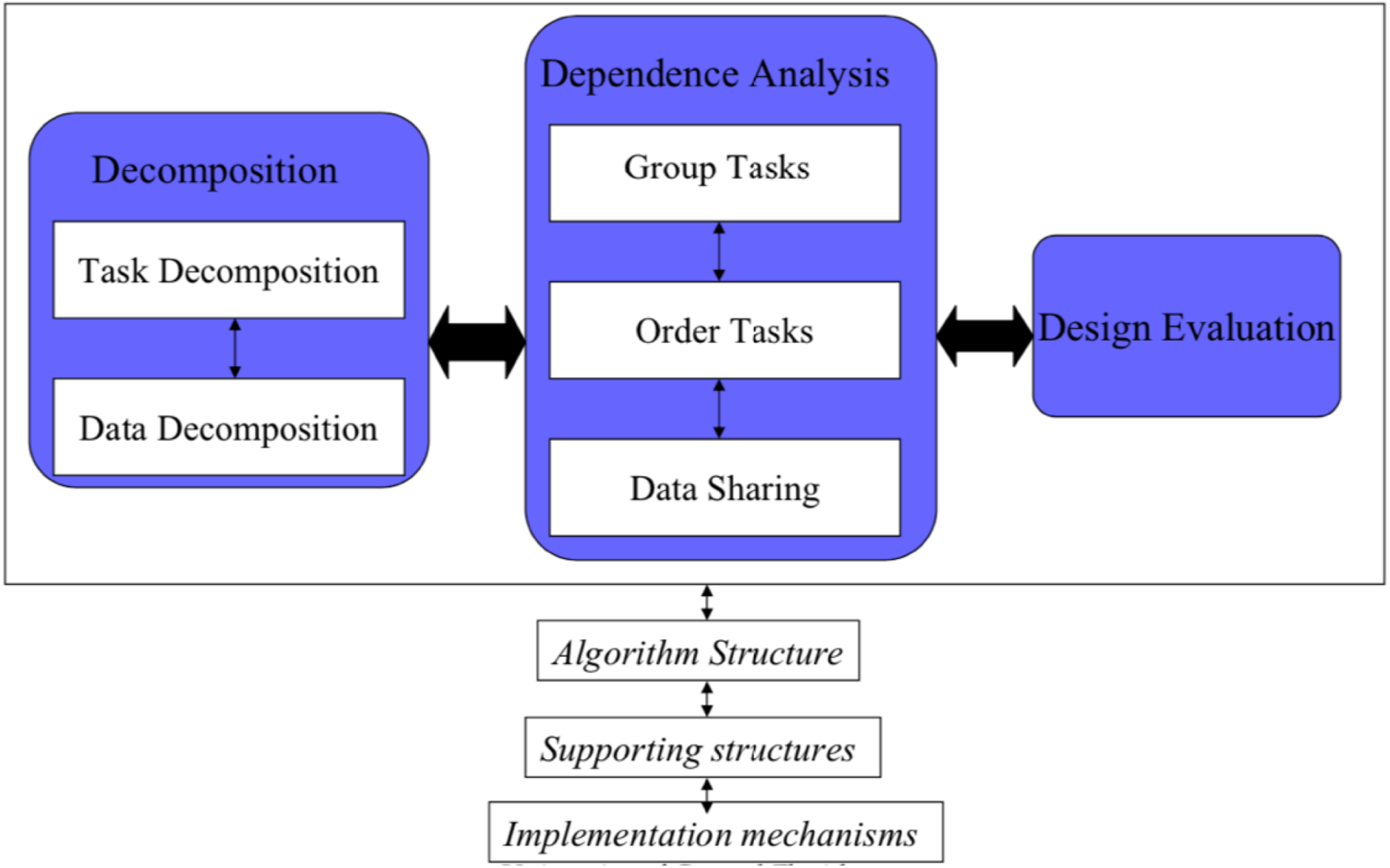
Exposing exploitable concurrency

Structuring algorithm to take advantage

Structuring program and shared data

Mapping to particular programming environment

Finding Concurrency



Decomposition Patterns

- Task decomposition: view problem as a stream of instructions that can be broken into sequences called tasks that can execute in parallel.
  - Key: Independent operations

- Data decomposition: view problem from data perspective and focus on how the can be broken into distinct chunks
  - Key: Data chunks that can be operated upon independently

- Task and data decomposition imply each other. They are different facets of the same fundamental decomposition

Example

- Matrix multiplication
  - Task decomposition
    - Considering the computation of each element in the product matrix as a separate task
    - Performs poorly => group tasks pattern
  - Data decomposition
    - Decompose the product matrix into chunks, e.g., one row a chunk, or a small submatrix (or block) per chunk

Dependency Analysis Pattern

- Group tasks: group tasks that have the same dependency constraints; identify which tasks must execute concurrently
  - Reduced synchronization overhead – all tasks in the group can use a barrier to wait for a common dependence
  - All tasks in the group efficiently share data loaded into a common on-chip, shared storage (Shard Memory)
  - Grouping and merging dependent tasks into one task reduces need for synchronization

- Order task pattern: identifying order constraints among task groups.
  - Control dependency: Find the task group that creates it
  - Data dependency: temporal order for producer and consumer relationship

Dependency Analysis Pattern

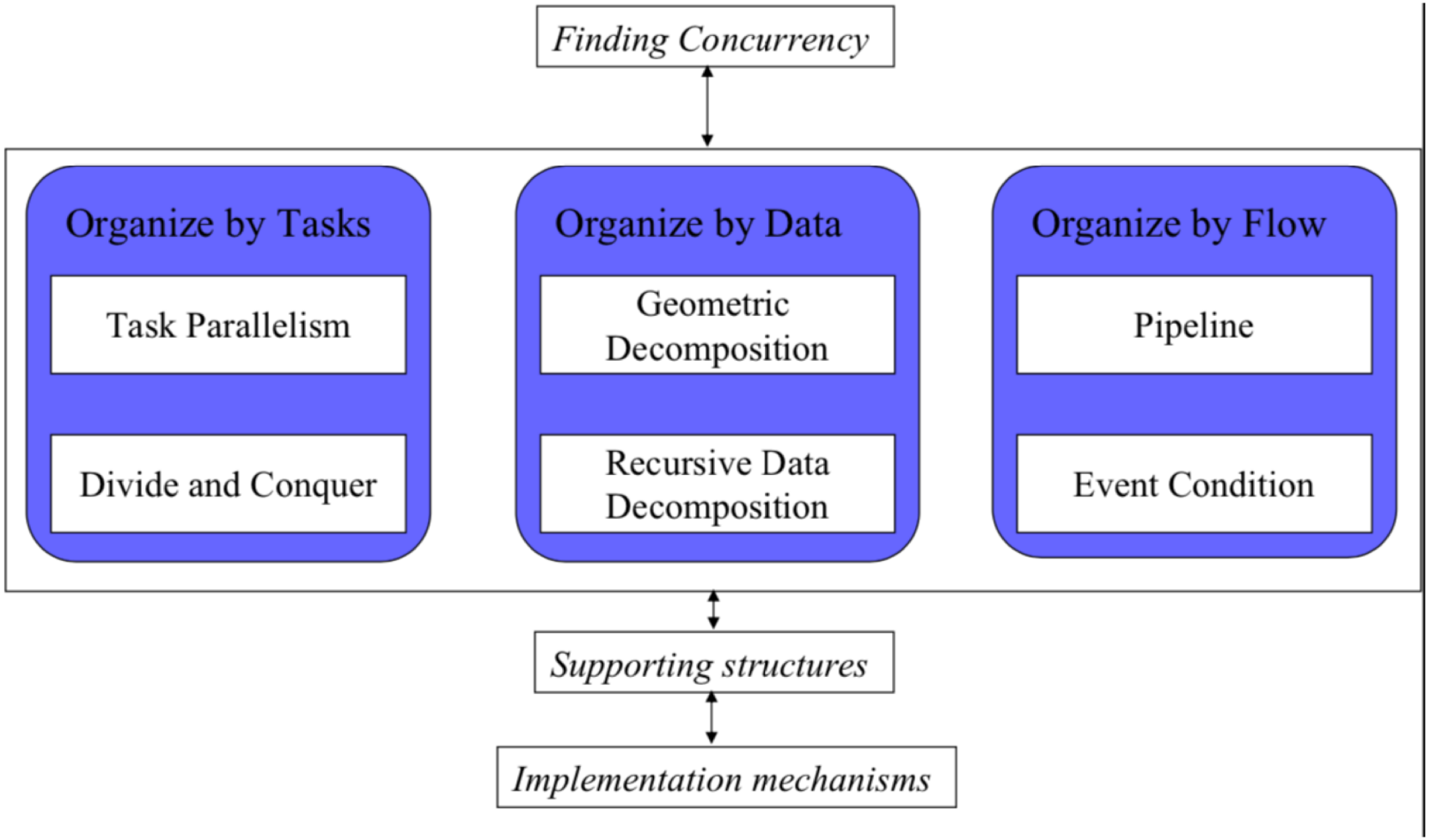
- Data sharing pattern: how data is shared among the tasks?
  - Read only: make own local copies
  - Effectively local: the shared data is partitioned into subsets, each of which is accessed (for read or write) by only one task a time.
  - Read-write: the data is accessed by more than one task. Need exclusive access mechanisms.
  - Example: the use of the shared memory among threads in a thread block.

Design Evaluation Pattern

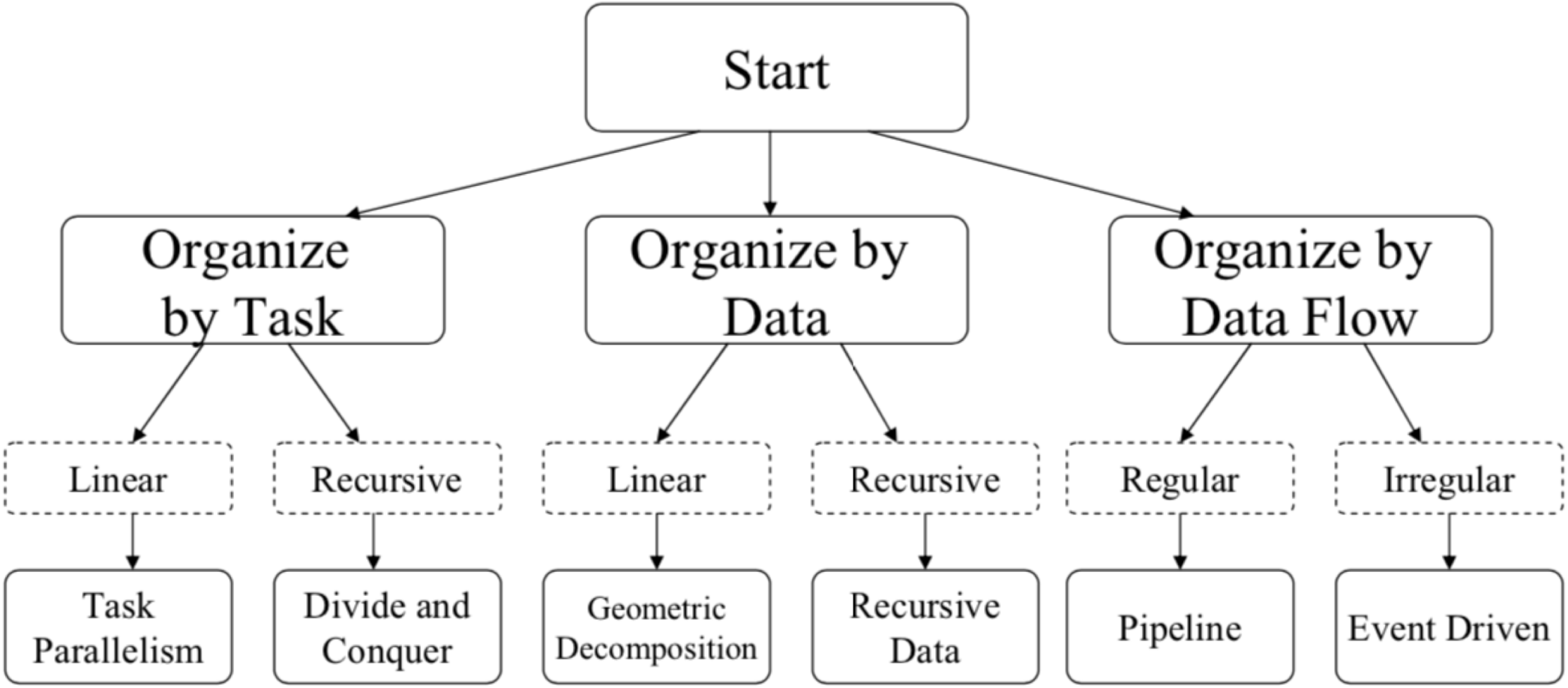
- Whether the partition fits the target hardware platform?

- Key questions to ask
  - How many threads can be supported?
  - How many threads are needed?
  - How are the data structures shared?
  - Is there enough work in each thread between synchronizations to make parallel execution worthwhile?

Algorithm Structure



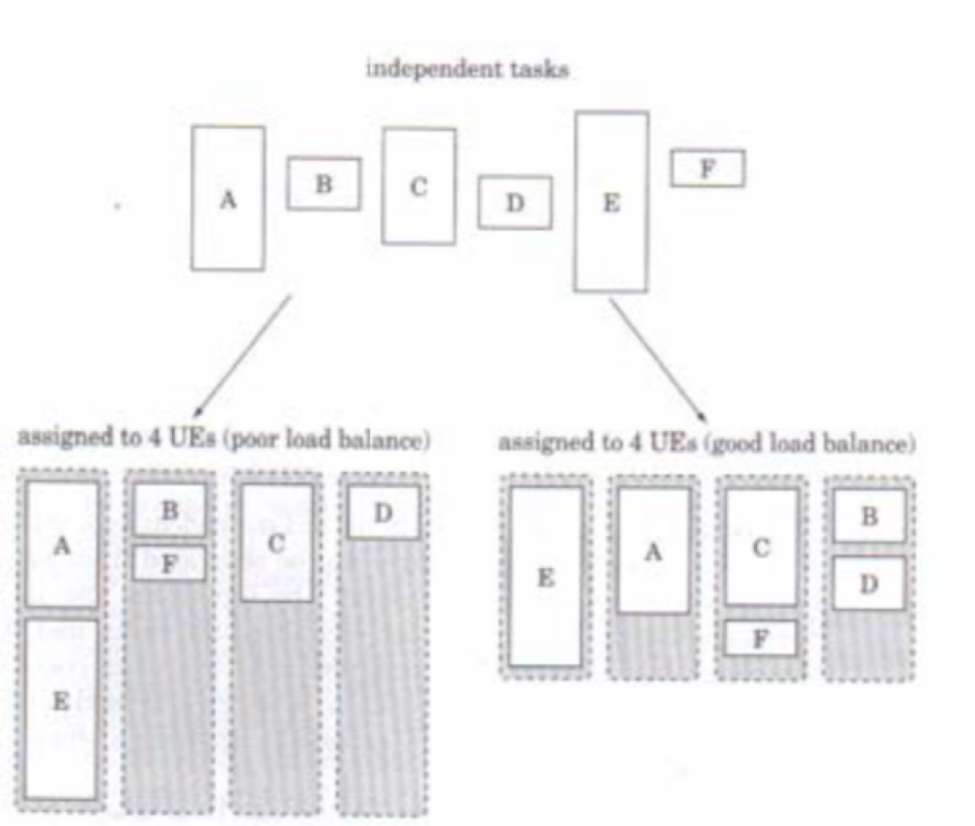
Organizing Principle



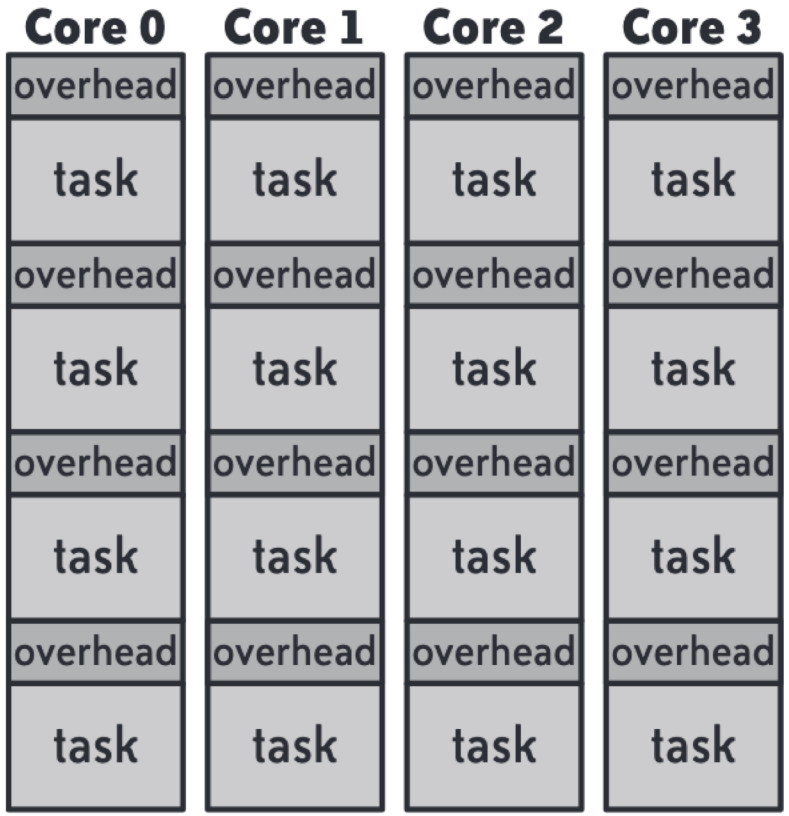
Task Parallelism Pattern

- After the problem is decomposed into a collection of tasks that can execute concurrently, how to exploit this concurrency efficiently?

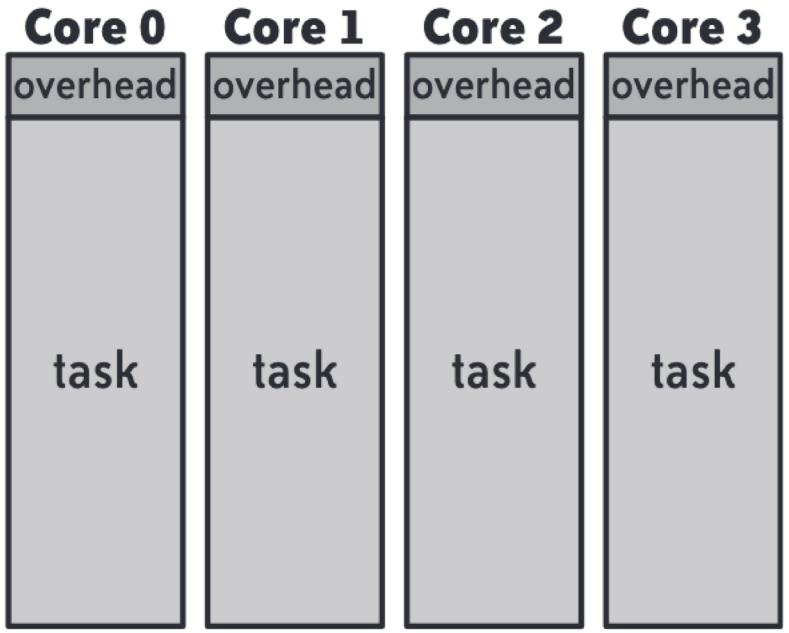
- Load balancing



Task Granularity Example



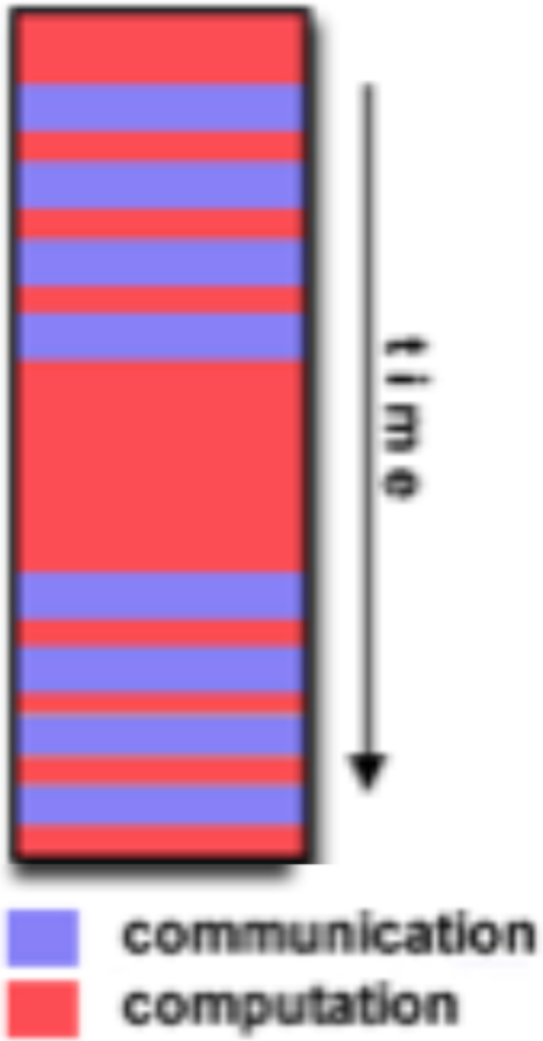
(a) Fine-grained decomposition



(b) Coarse-grained decomposition

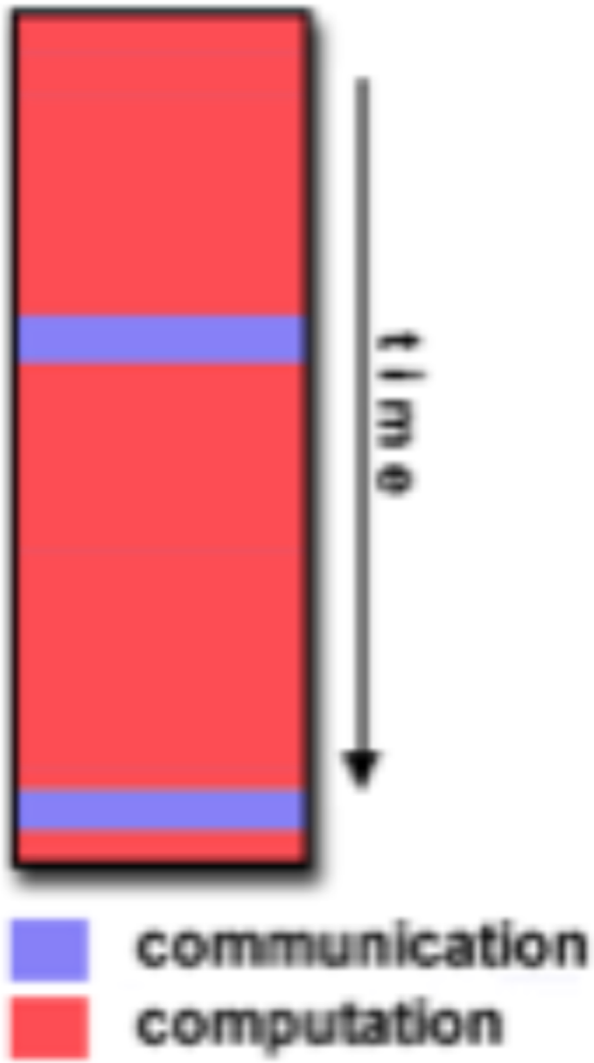
Task Granularity Example

- Tasks execute little comp. between comm.
- Easy to load balance
- If ***too fine***, comm. may take longer than comp.



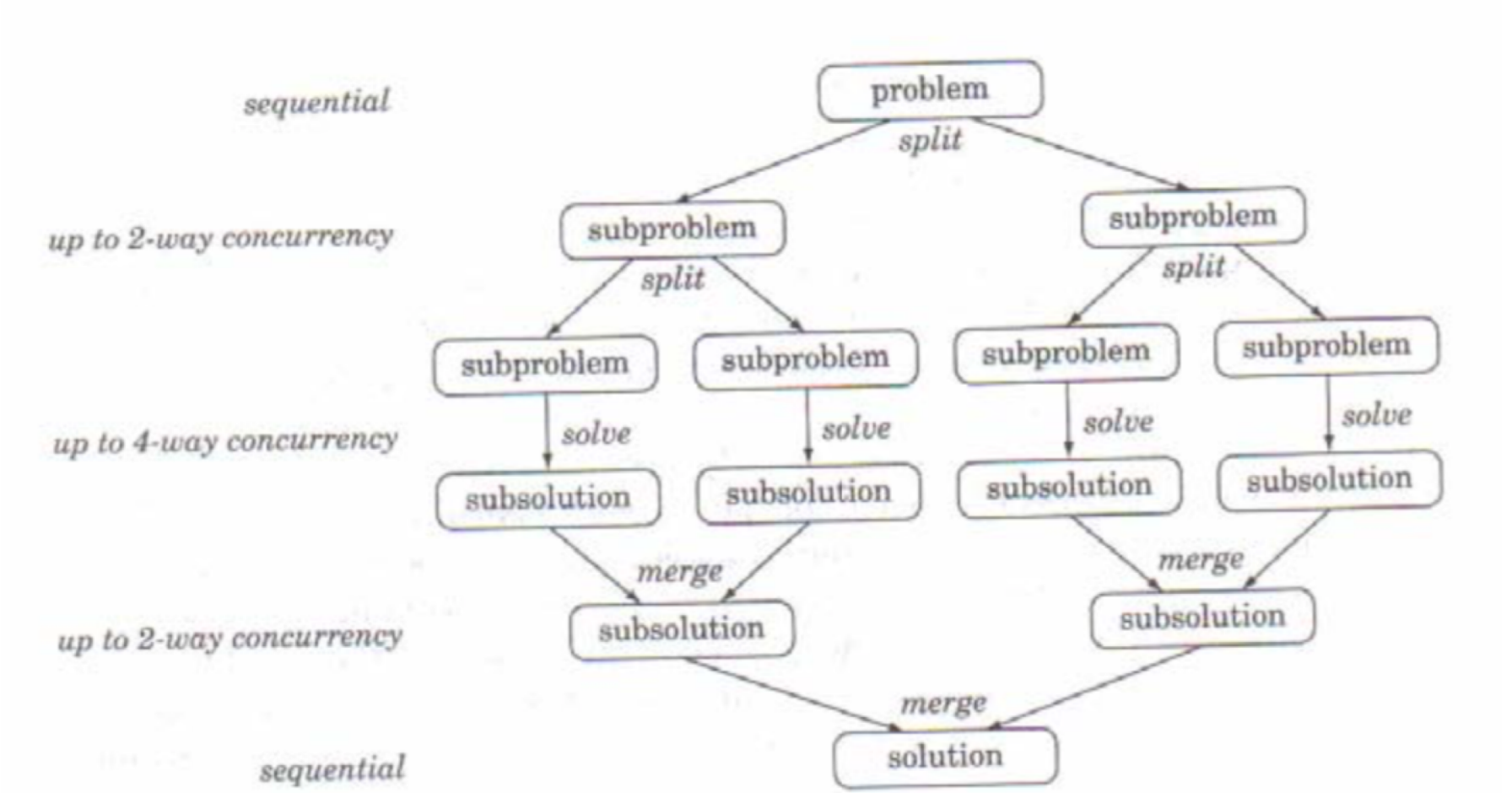
Task Granularity Example

- Long computations between communication
- More opportunity for performance increase
- Harder to load balance



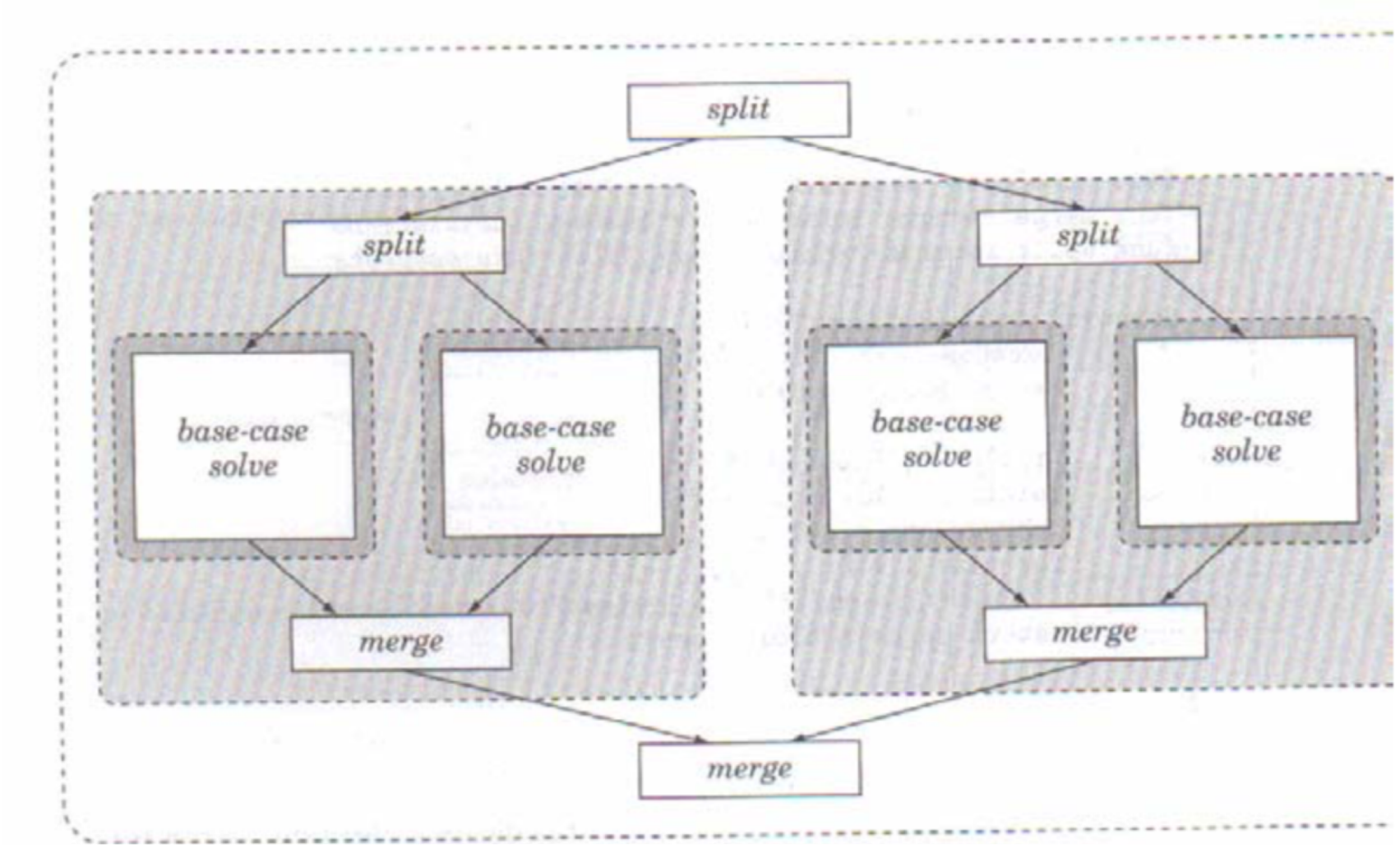
Divide and Conquer Pattern

- If the problem is formulated using the sequential divide-and-conquer strategy, how to exploit the potential concurrency?



Divide and Conquer Pattern

- Parallelization Strategy



Geometric Decomposition Pattern

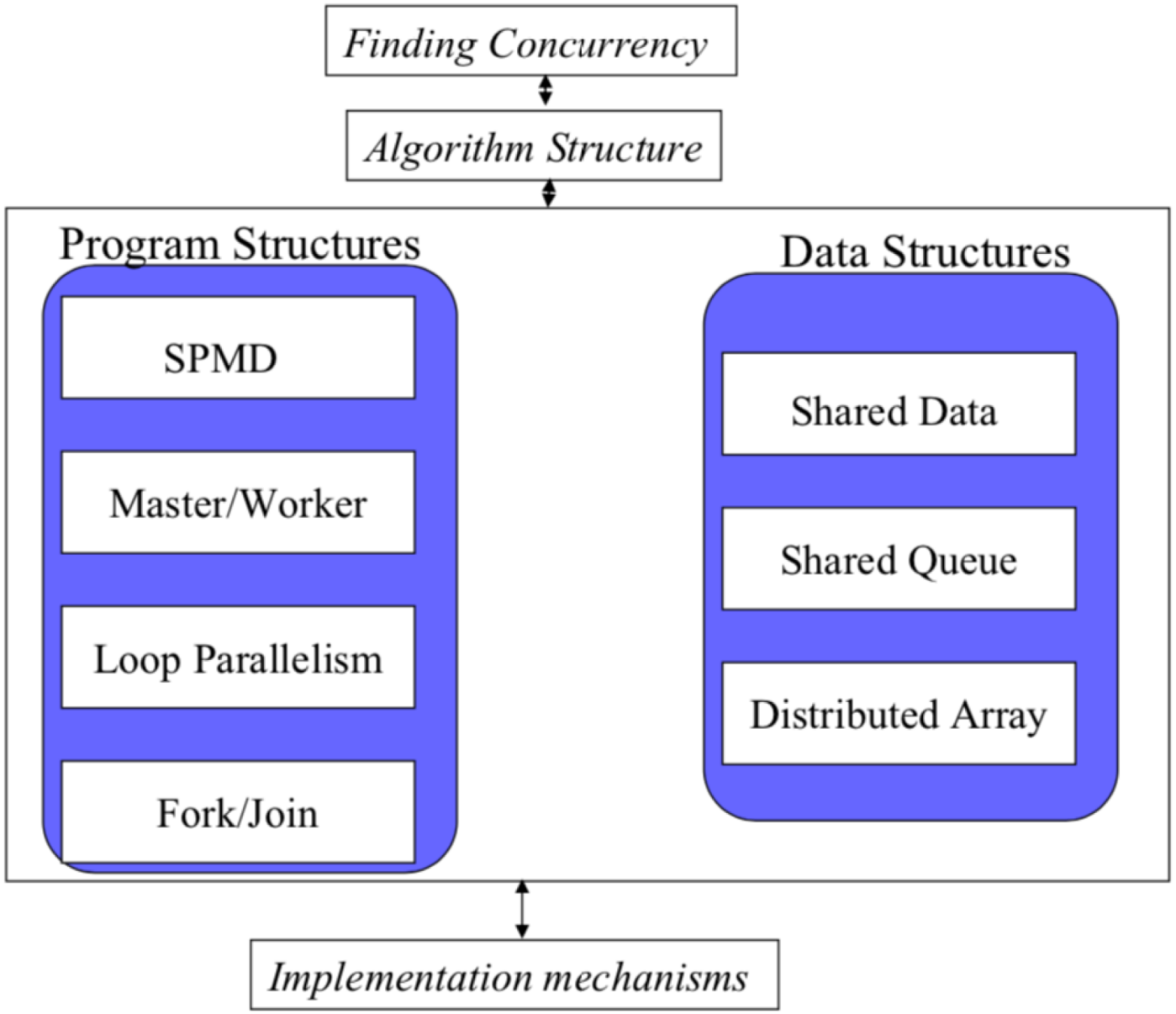
- How to organize the algorithm after the data has been decomposed into concurrently updatable chunks?

- Decomposition to minimize the data communication and dependency among tasks
- Care needs to be taken when update non-local data, e.g., exchange operations

Recursive Data Pattern

- Suppose the problem involves an operation on a recursive data structure that appears to require sequential processing. How to make the operations on these data structures parallel?
- Check whether divide-and-conquer pattern works
- If not, may need to transform the original algorithm.

Supporting Structures



Relationship between supporting program structure pattern and Algorithm structure pattern

	Task Parallel.	Divide/Conquer	Geometric Decomp.	Recursive Data	Pipeline	Event-based
SPMD	☺ ☺ ☺ ☺	☺ ☺ ☺	☺ ☺ ☺ ☺	☺ ☺	☺ ☺ ☺	☺ ☺
Loop Parallel	☺ ☺ ☺ ☺	☺ ☺	☺ ☺ ☺			
Master/Worker	☺ ☺ ☺ ☺	☺ ☺	☺	☺	☺	☺
Fork/Join	☺ ☺	☺ ☺ ☺ ☺	☺ ☺		☺ ☺ ☺ ☺	☺ ☺ ☺ ☺

Relationship between supporting program structure pattern and programming environment

	OpenMP	MPI	Java	Brook+/ CUDA	Cell
SPMD	☺ ☺ ☺	☺ ☺ ☺ ☺	☺ ☺	☺ ☺ ☺ ☺ ☺	☺ ☺ ☺ ☺
Loop Parallel	☺ ☺ ☺ ☺	☺	☺ ☺ ☺		☺ ☺ ☺
Master/ Slave	☺ ☺	☺ ☺ ☺	☺ ☺ ☺		☺ ☺ ☺ ☺
Fork/Join	☺ ☺ ☺		☺ ☺ ☺ ☺		☺ ☺

Implementation Mechanism