“Dependence and Parallelism”

To remember

- Statement order must not matter.
- Statements must not have dependences.
- Some dependences can be removed.
- Some dependences may not be obvious.

Summary

- No dependence (can run in parallel)
  S1: X=K+3;
  S2: Y=Z*5;
- True dependence (cannot run in parallel)
  S1: X=3;
  S2: Y=X*5;
- Anti dependence (cannot run in parallel)
  S1: Y=X*4;
  S2: X=3;
- Output dependence (cannot run in parallel)
  S1: X=Y*4;
  S2: X=3;
- Loop-carried dependence

Parallel algorithm Design

Programming Models

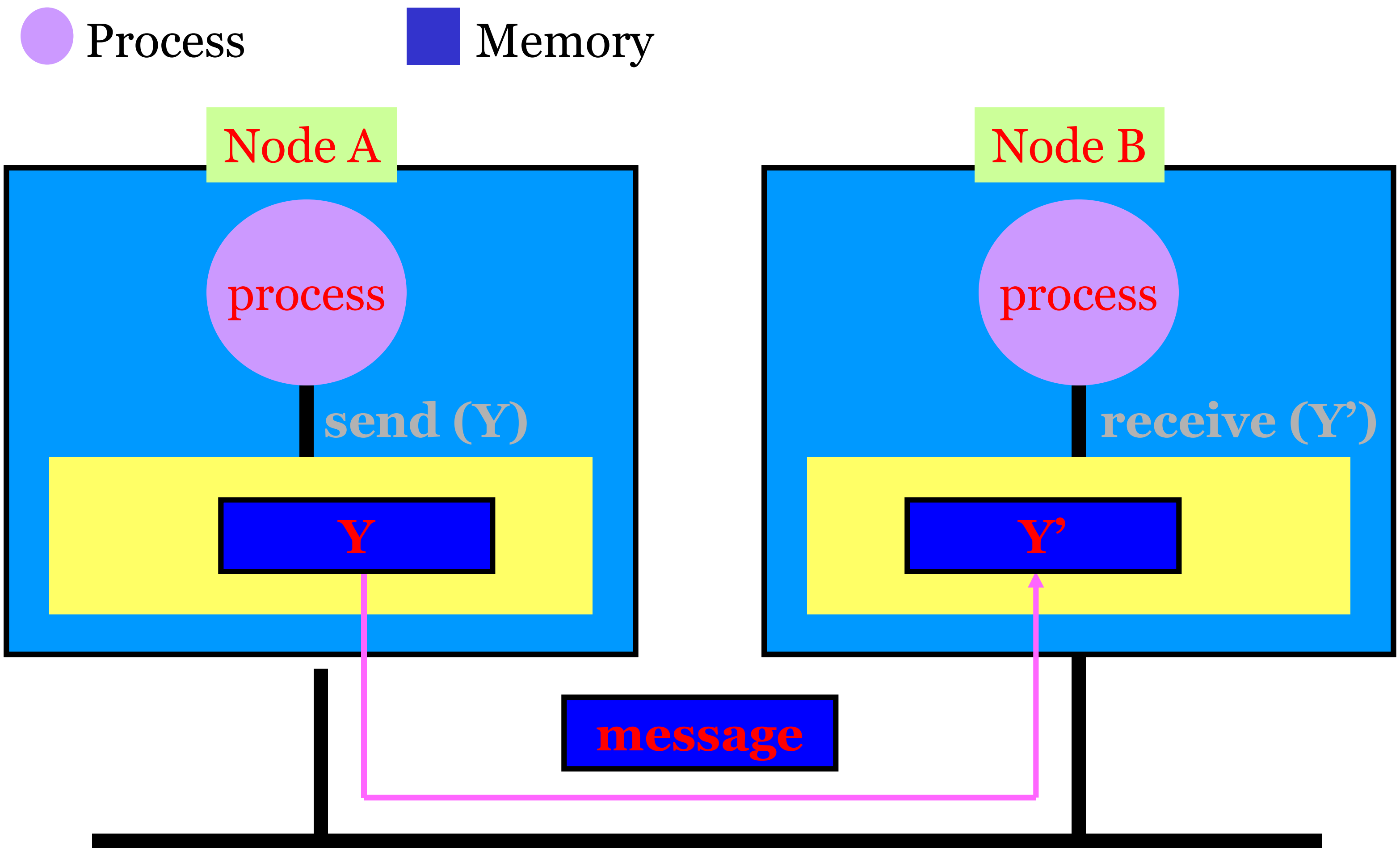
- The programming model
  - determines the basic concepts of the parallel implementation and
  - abstracts from the hardware as well as from the programming language or API.
- The names used for programming models differ in the literature.

Programming Models (2)

- Sequential Model: The sequential program is automatically parallelized.
  - Advantage: Familiar programming model
  - Disadvantage: Limitations in compiler analysis
- Message Passing Model: The application consists of a set of processes with separate address spaces. The processes exchange messages by explicit send/receive operations.
  - Advantage: Full control of performance aspects
  - Disadvantage: Complex programming

Assembly programming for parallel architectures!

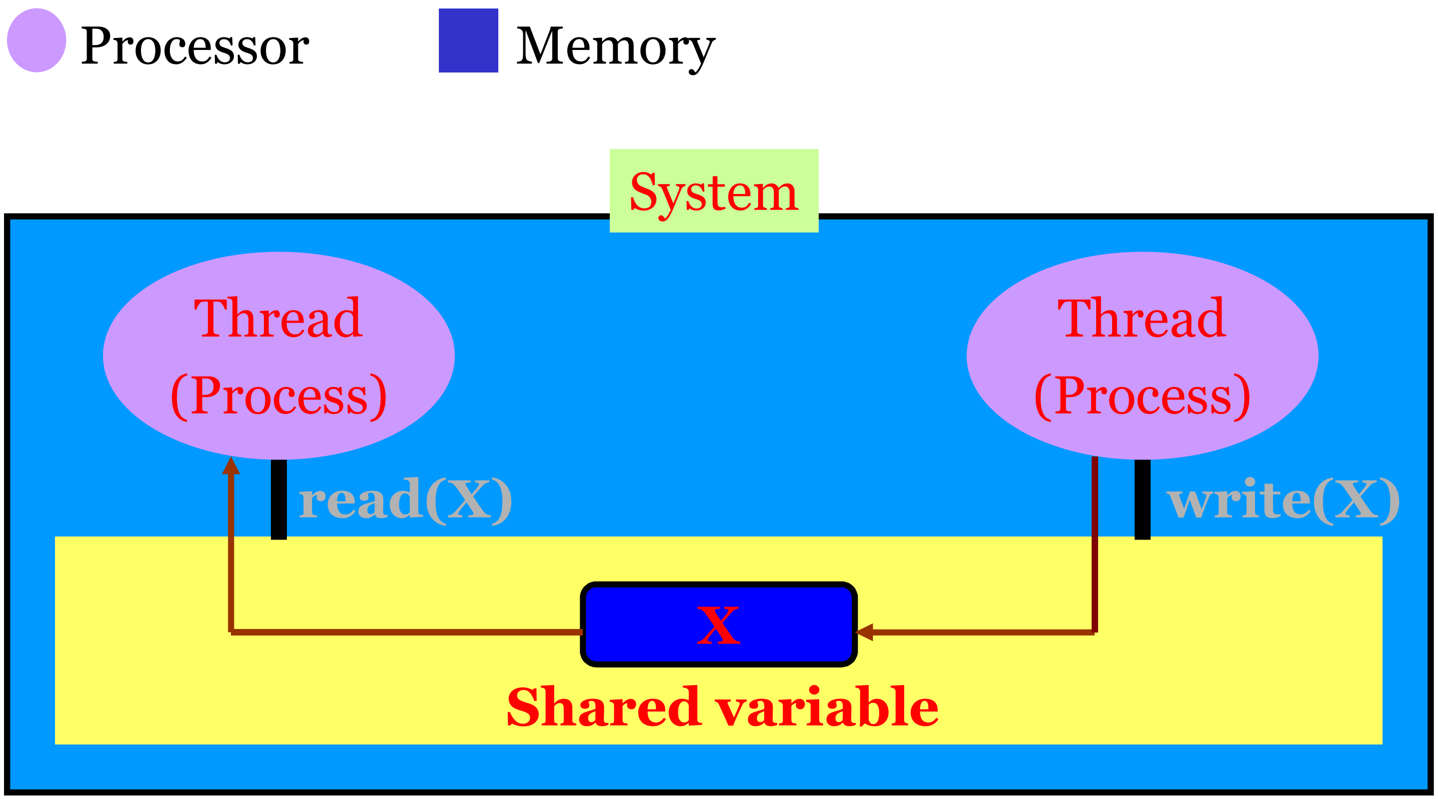
MP Programming Model



Programming Models (3)

- Shared Memory Model: Application consists of parallel threads, accessing shared data structures protected by synchronization operations.
  - Thread-Based SM Model: Explicit programming of cooperating threads.
    - Advantage: Portability
    - Disadvantage: Complex programming
  - Directive-Based SM Model: Parallel loops with implicit synchronization.
    - Advantage: Easier to program
    - Disadvantage: Difficult to control performance
  - Remote Memory Access Model: Parallel threads accessing mainly private data structures. Shared data structures are accessed via special operations.
    - Advantage: Can be easily combined with message passing
    - Disadvantage: Complex programming

Shared Memory Programming Model



Programming Models (4)

- Data Parallel Programming Model: Synchronized execution of parallel operations on large distributed data structures.
  - Advantage: High-level programming
  - Disadvantage: Performance of current implementations

Together…

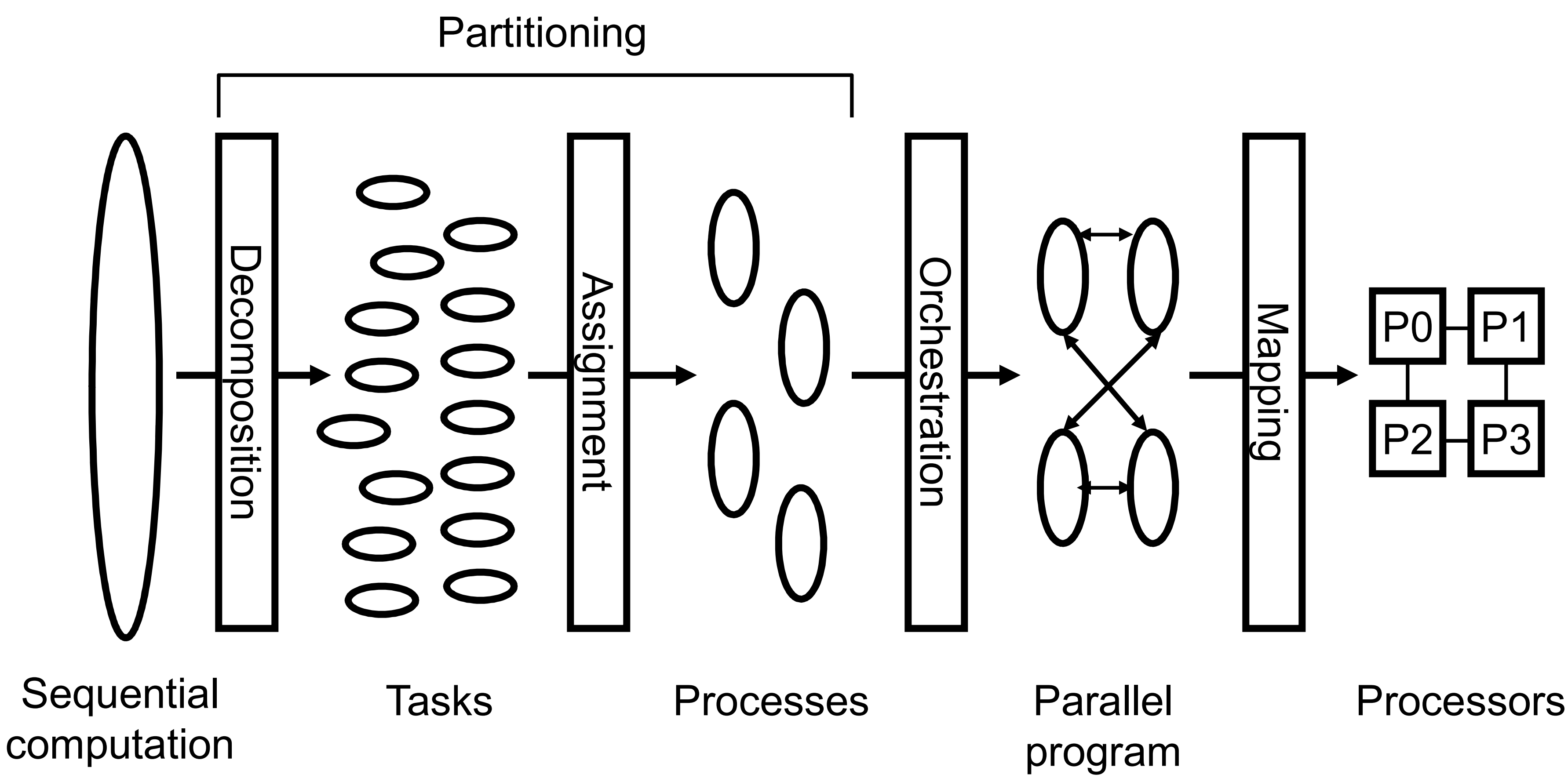
- Parallel programming models
  - Shared memory (Pthreads and openMP)
  - Message passing (MPI)
  - Massive data parallelism (GPU programming)
  - Hybrid

- Distributed programming models
  - Cloud programming, MapReduce, Spark

Concepts of Parallel Programming

- Concepts:
  - Task: arbitrary piece of work performed by a single process
  - Thread: is an abstract entity as part of a process that performs tasks. Defines a unit for scheduling.
  - Process: is active entity with resources that performs tasks.
  - Processor: is a physical resource executing processes

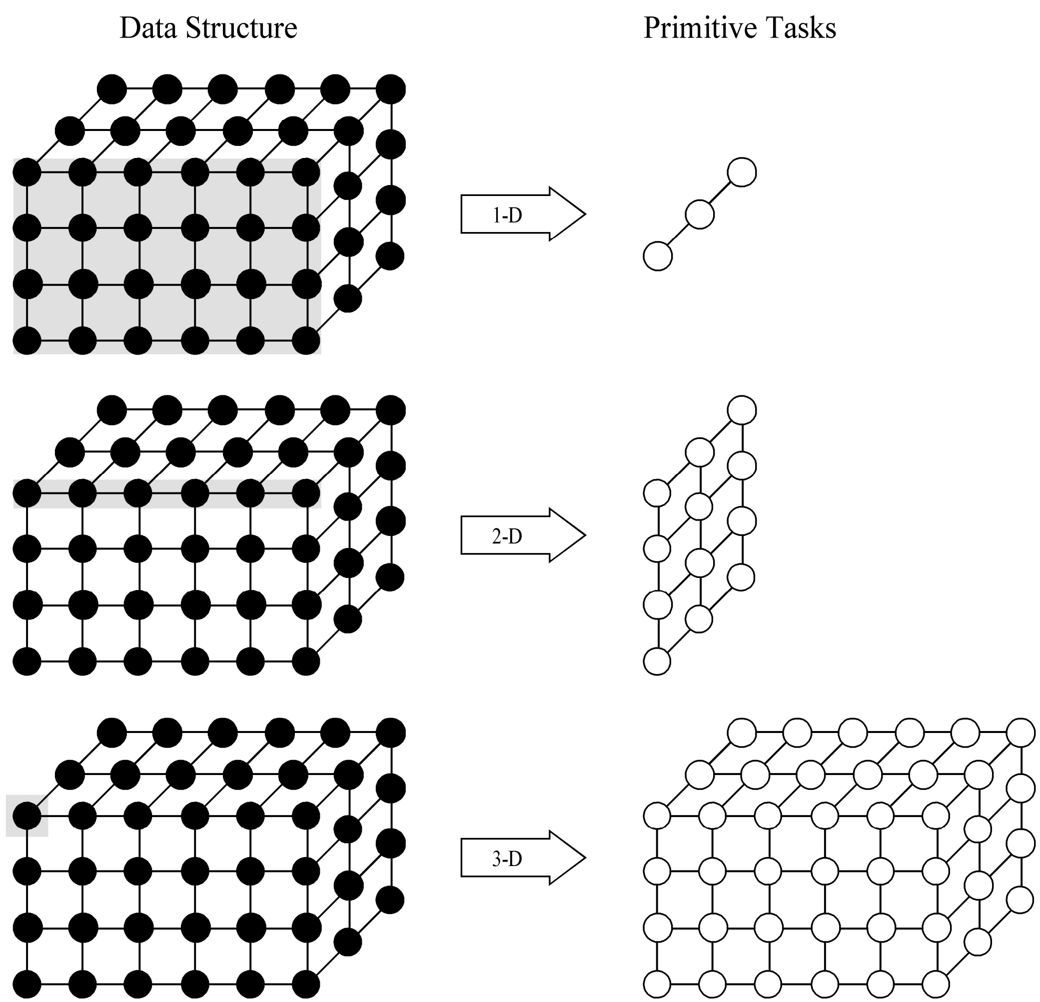
Phases in the Parallelization Process



Decomposition

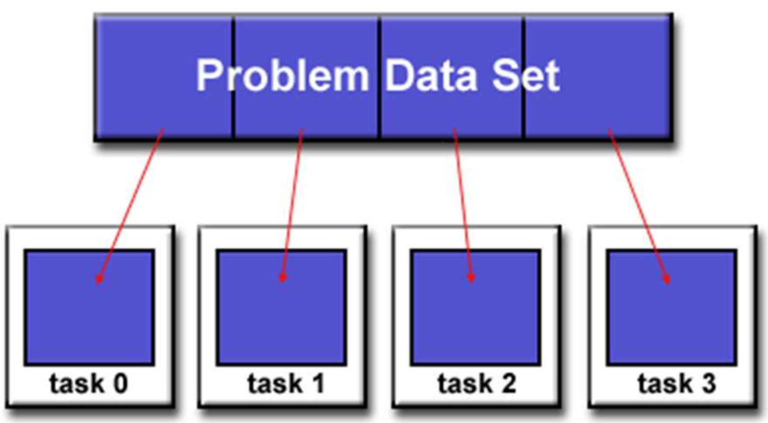
- Dividing computation and data into pieces
- Domain decomposition
  - Divide data into pieces
  - Determine how to associate computations with the data
- Functional decomposition
  - Divide computation into pieces
  - Determine how to associate data with the computations

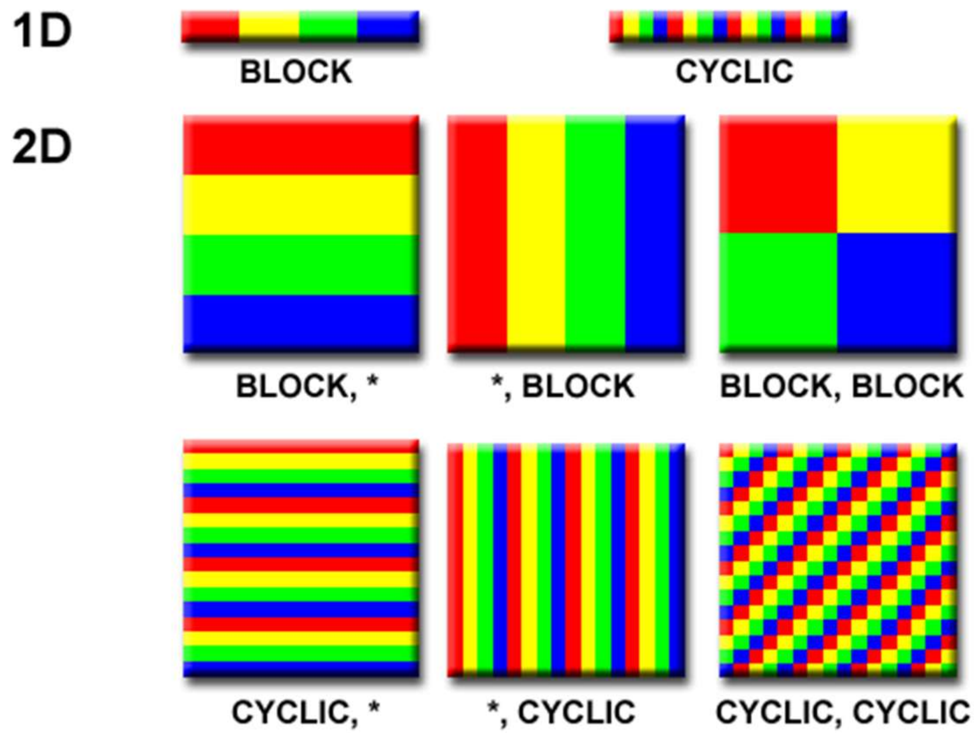
Example Domain Decompositions



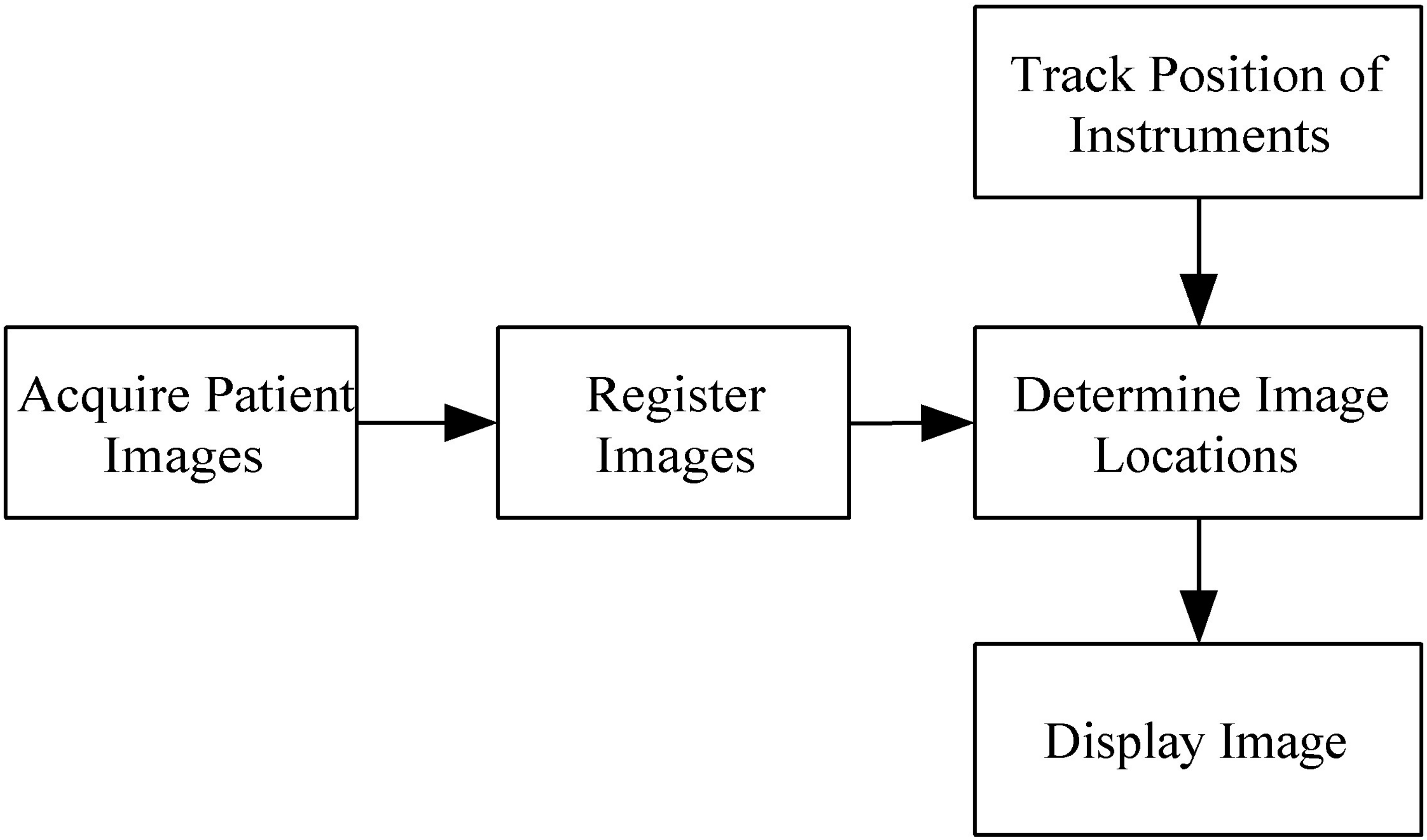
Domain Decomposition

- The **data** associated with a problem is decomposed!
  - If possible, divide data into small pieces of equal size
- Each parallel task then works on a portion of the data
  - Generates number of tasks, each with some data and set of operations on data





Example Functional Decomposition



Functional Decomposition

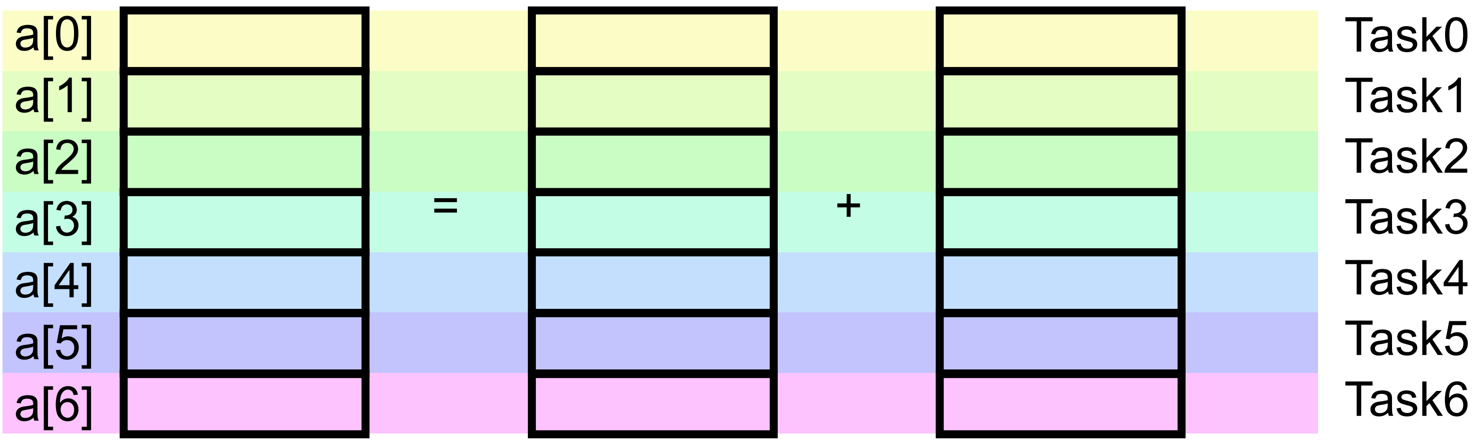
- Breaking the computation into a collection of tasks
  - Tasks may be of variable length
  - Tasks require data to be executed
  - Tasks may become available dynamically

Data vs Functional Parallelism

- Data parallelism
  - The same operations are executed in parallel for the elements of large data structures, e.g. arrays.
  - Tasks are the operations on each individual element or on subsets of the elements.
  - Whether tasks are of same length or variable length depends on the application. Quite some applications have tasks of same length.

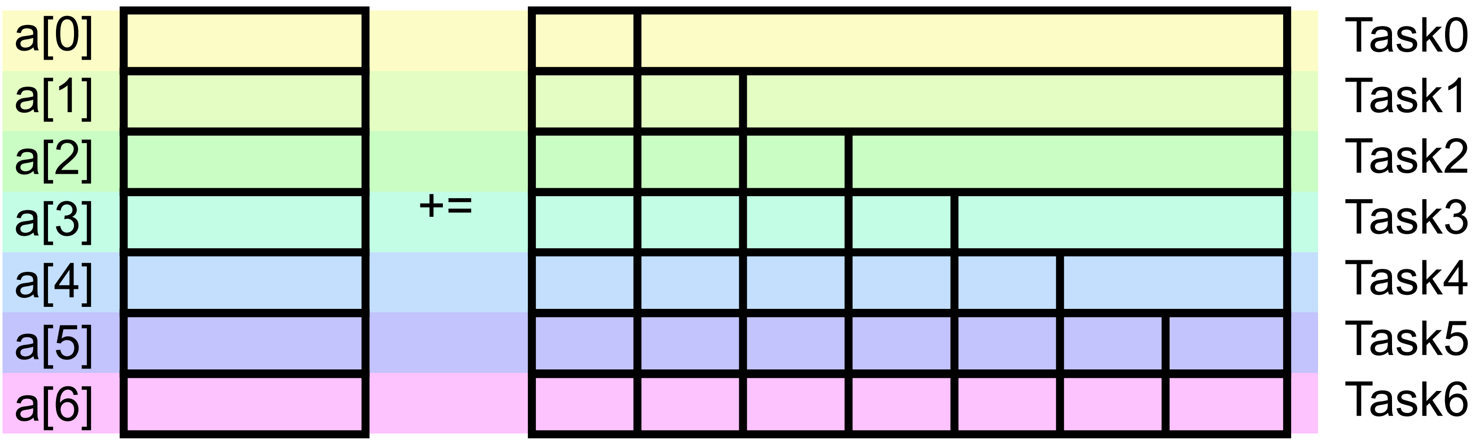
Example: Data Parallelism

```
for (i=0;i<n;i++)
   a[i]=b[i]+c[i]
```



Example: Data Parallelism, Variable Length

```
for (i=0;i<n;i++)
   for (j=0;j<=i;j++)
      a[i]=a[i]+b[i][j]
```



Data vs Functional Parallelism

- Functional parallelism
  - Entirely different calculations can be performed concurrently on either the same or different data.
  - The tasks are usually specified via different functions or different code regions.
  - The degree of available functional parallelism is usually modest.
  - Tasks are of different length in most cases.

Example: Function Parallelism

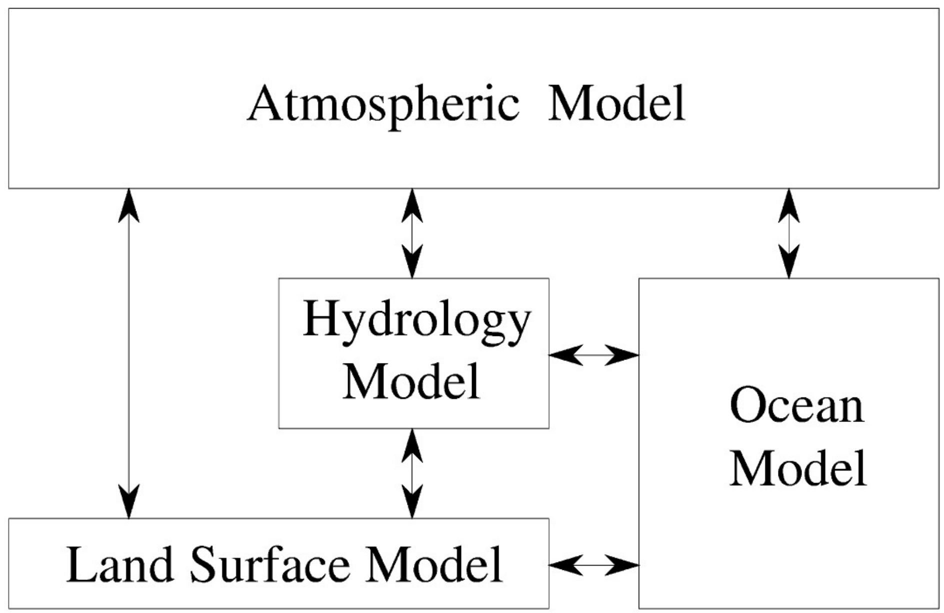
```
f(a);
g(b);
```

```
i=i+10;
j=2*k+z**2;
```

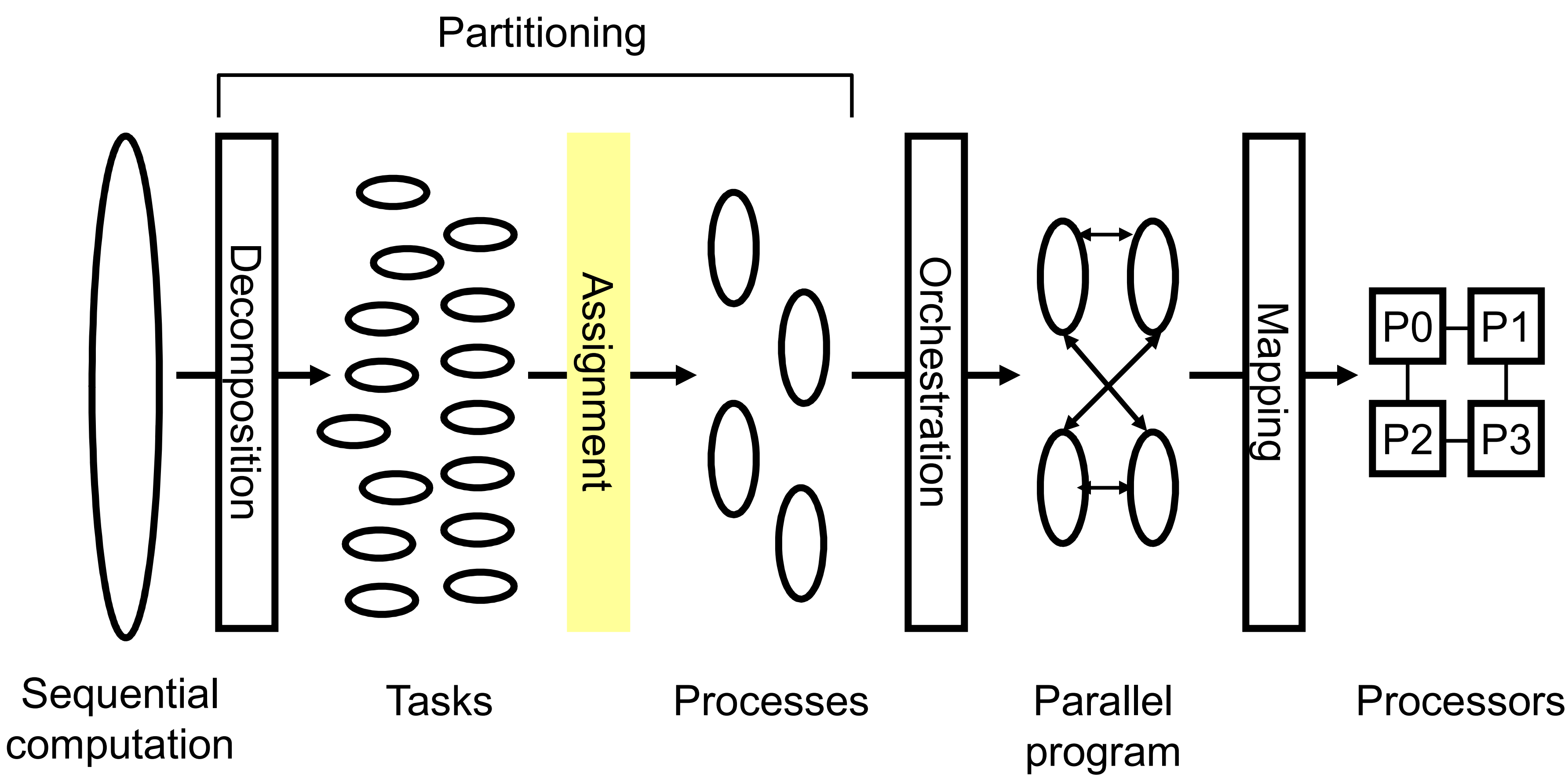
- The functions or statements can be executed in parallel.
- These are different operations → functional parallelism

Functional Decomposition Example

- A simulation of the earth's climate comprises atmosphere, ocean, hydrology, etc.



Phases in the Parallelization Process

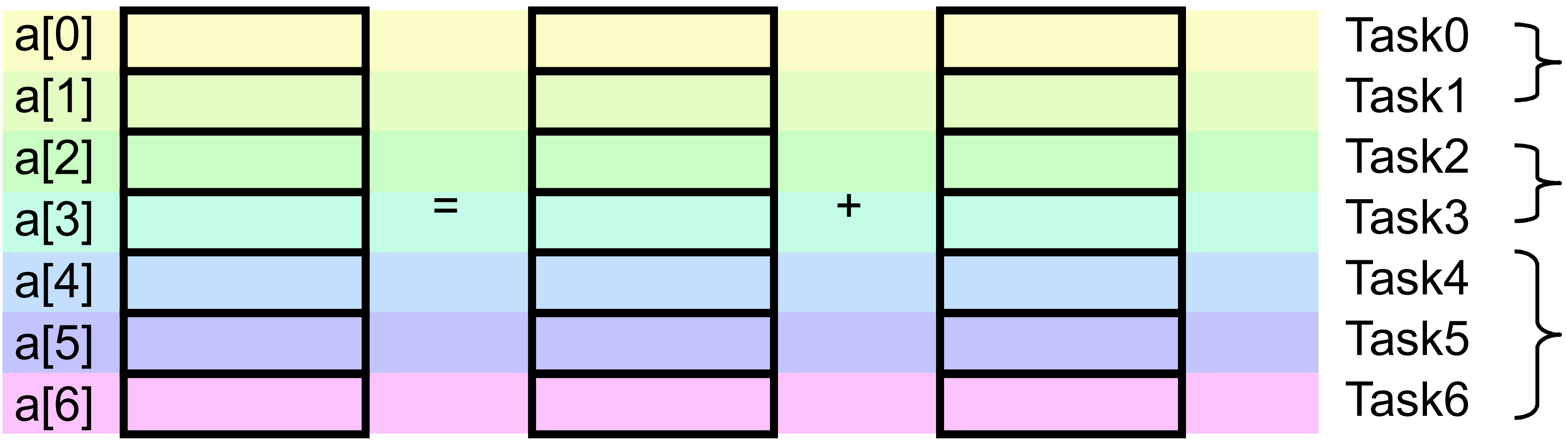


Assignment

- Assignment means specifying the mechanism by which tasks will be distributed among processes.
- Goals:
  - Balance workload
  - Reduce interprocess communication
  - Reduce assignment overhead
- Assignment time
  - Static: fixed assignment during compilation or program creation
  - Dynamic: adaptive assignment during execution

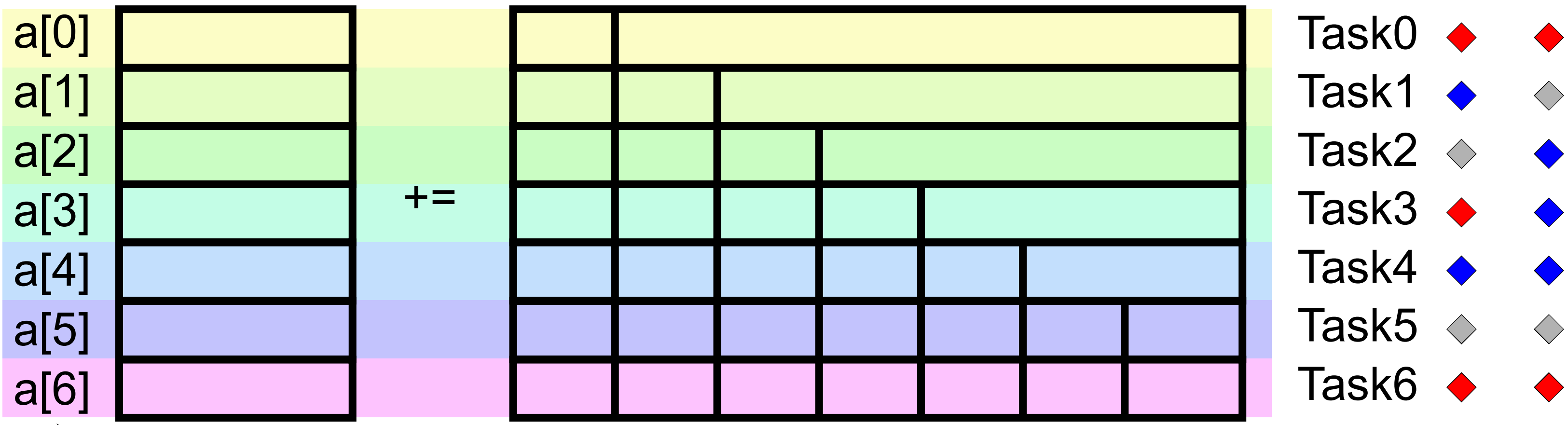
Example: Balance Workload

```
for (i=0;i<n;i++)
    a[i]=b[i]+c[i]
```



Example: Balance Workload (2)

```
for (i=0;i<n;i++)
    for (j=0;j<i;j++)
        a[i]=a[i]+b[i][j]
```

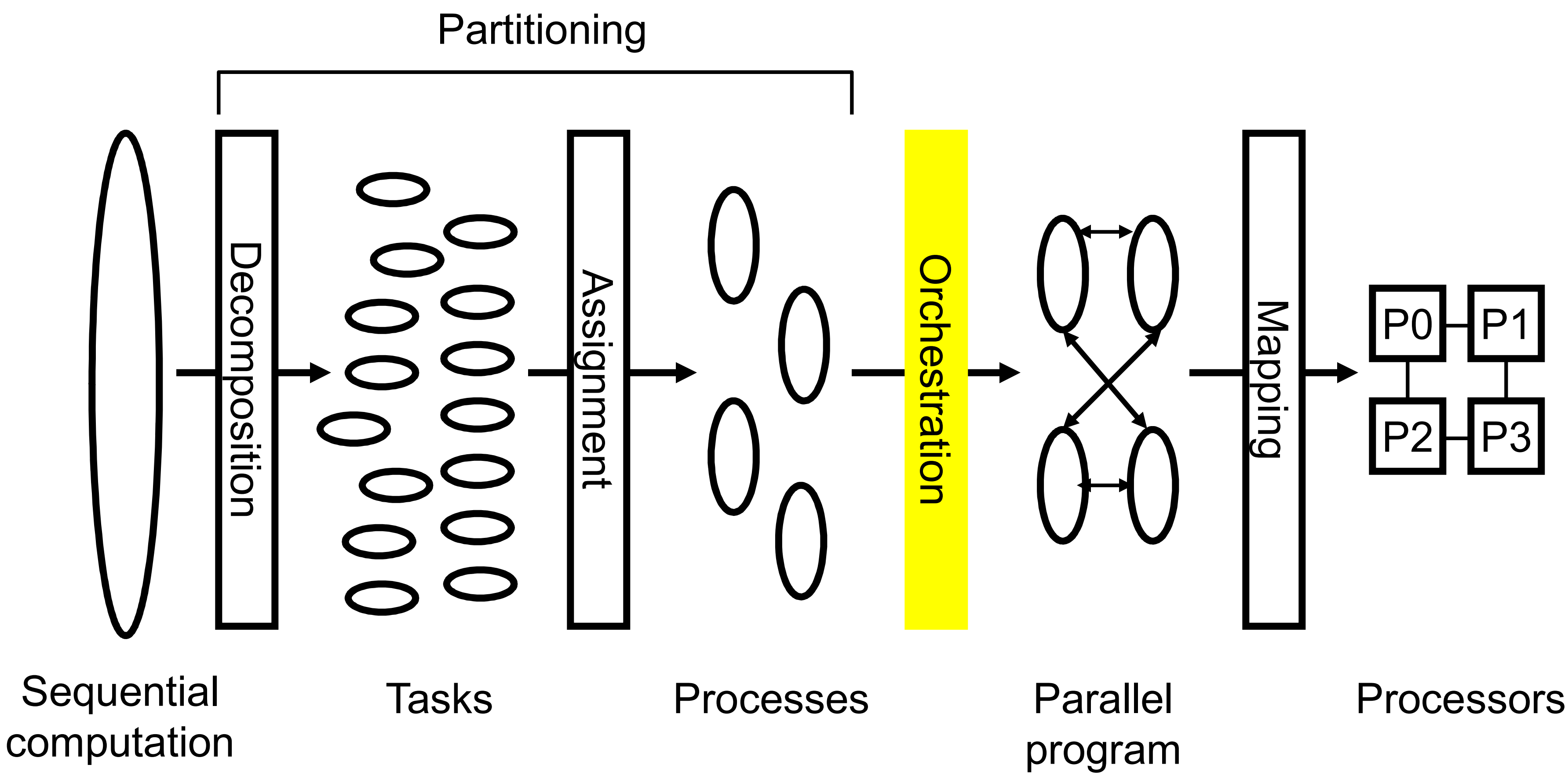


- Static Load Balancing
  - o 2 different assignments
- Dynamic Load Balancing
  - o postpone assignment until execution time

Partitioning Checklist

- Does your partition define at least an order of magnitude more tasks that there are processors?
- Does your partition avoid redundant computation and storage requirements?
- Are tasks of equal size?
- Does no. of tasks scale with problem size?
- Have you identified alternate partitions?

Phases in the Parallelization Process



Orchestration

- Implementation in a given programming model
- Means for
  - Naming and accessing data
  - Exchanging data
  - Synchronization
- Questions
  - How to organize data structures?
  - How to schedule assigned tasks to improve locality?
  - Whether to communicate in large or small messages?
- Performance goal
  - Reduction of communication and synchronization overhead
  - Reduction of parallelization overhead
  - Reduction of idle time
  - Ideal load balancing
- Sometimes in some cases goals can be conflicting
  - reduction of communication versus reduction of parallelization overhead

Communication (MPI)

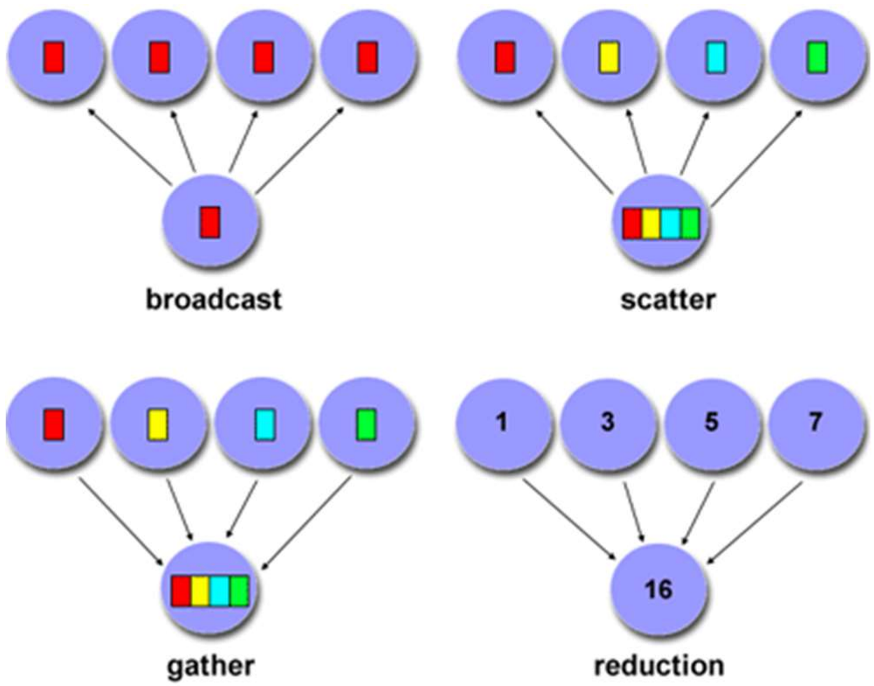
- Tasks often execute concurrently, but not independently
  - The computation to be performed in one task requires data from another
  - Information flow is specified in communication phase
- We can conceptualize a need for comm between two tasks as a channel linking tasks
- Comm is easy for functional decomposition, not easy for domain decomposition
- Latency vs Bandwidth
  - Sending many small messages can cause latency to dominate communication overhead.
  - Package small messages into a larger message (aka increasing the effective comm bandwidth)

Categories of Communication

- Local versus global
  - Local comm: ?
  - Global comm: ?
- Structured versus unstructured
  - Structured comm: ?
  - Unstructured comm: ?
- Static versus dynamic
  - Static: ?
  - Dynamic: ?
- Synchronous versus asynchronous
  - Synchronous: ?
  - Asynchronous: ?

Scope of Communications

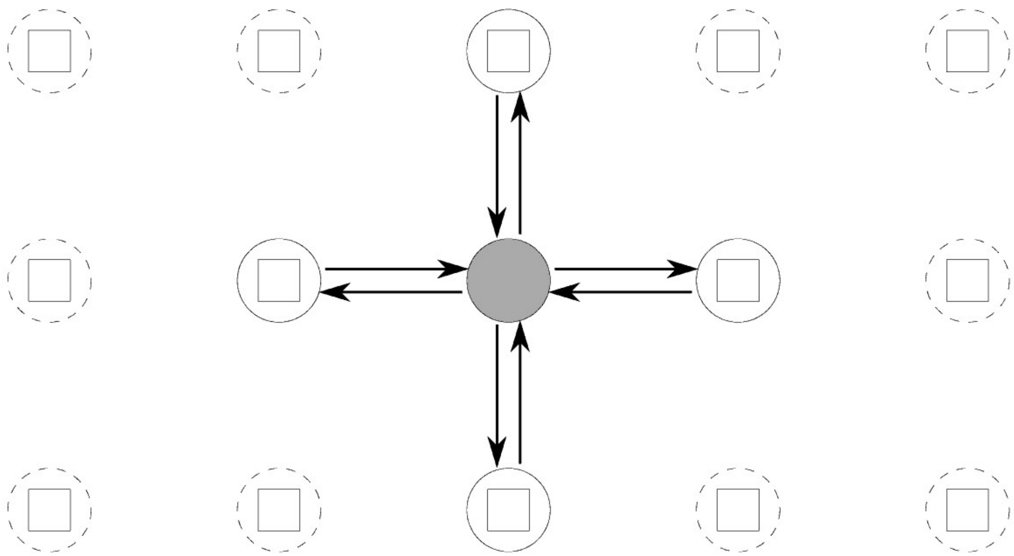
- Point-to-point
  - involves two tasks with one acting as the sender and the other acting as the receiver
- Collective
  - Involves data sharing among more than two tasks



Local Communication

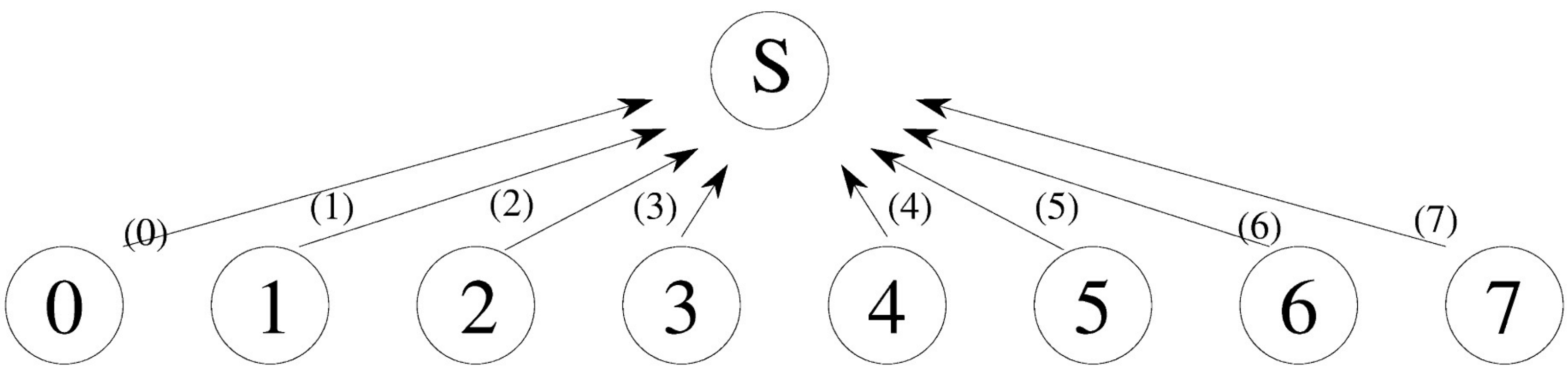
- Consider comm requirements of Jacobi finite difference scheme

$$x_{i,j}^{(t+1)} = \frac{4x_{i,j}^{(t)} + x_{i-1,j}^{(t)} + x_{i+1,j}^{(t)} + x_{i,j-1}^{(t)} + x_{i,j+1}^{(t)}}{8}$$



Global Communication

- Consider problem of performing a parallel reduction of N numbers



Communication Design Checklist

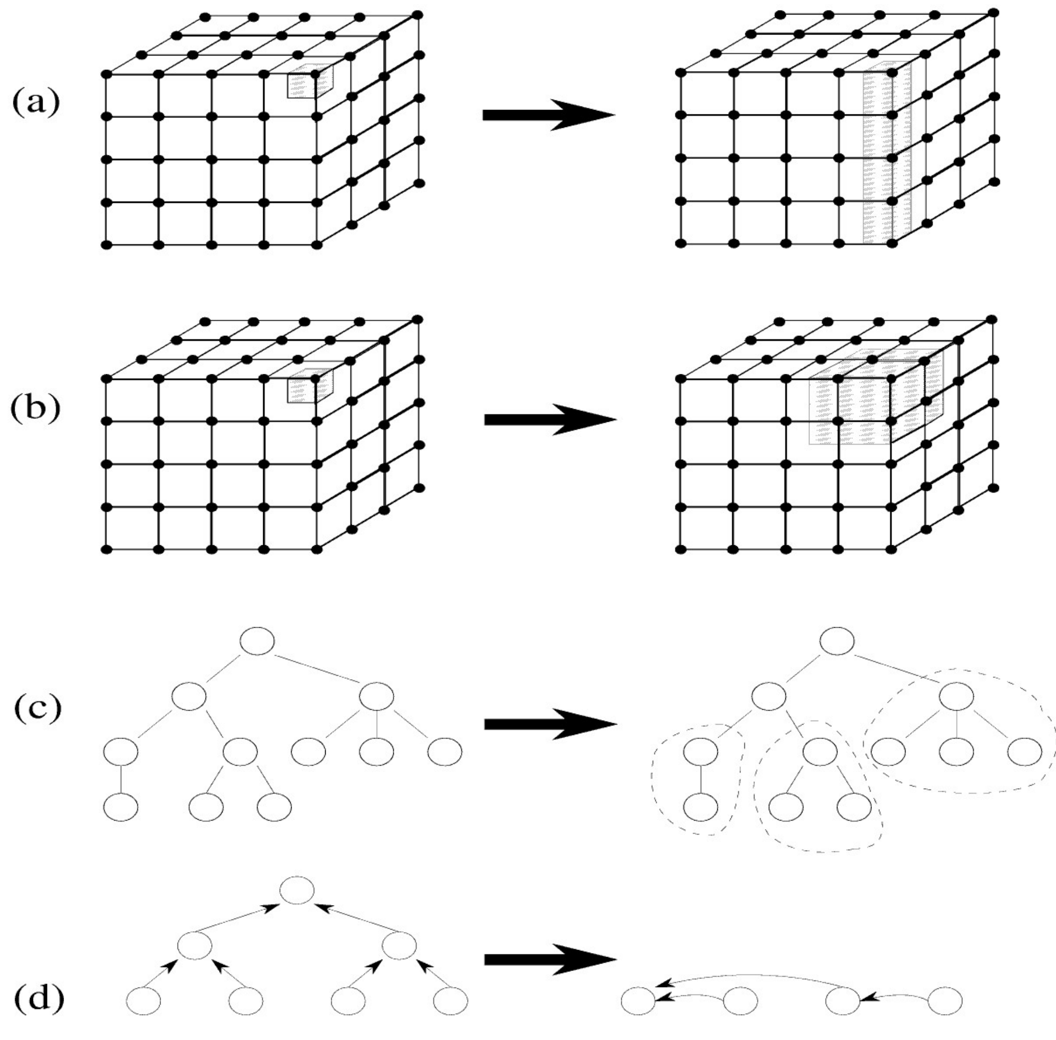
- Do all tasks perform same amount of communication operations?
- Does each task communicate with small number of neighbors?
- Are communication operations able to proceed concurrently?
- Is computation for different tasks able to proceed concurrently?

Agglomeration (MPI)

- In first two phases, we partitioned the computation to be performed into a set of tasks and introduced comm to provide data required by these tasks

- In agglomerate stage, we consider if it is useful to combine tasks to provide smaller number of tasks

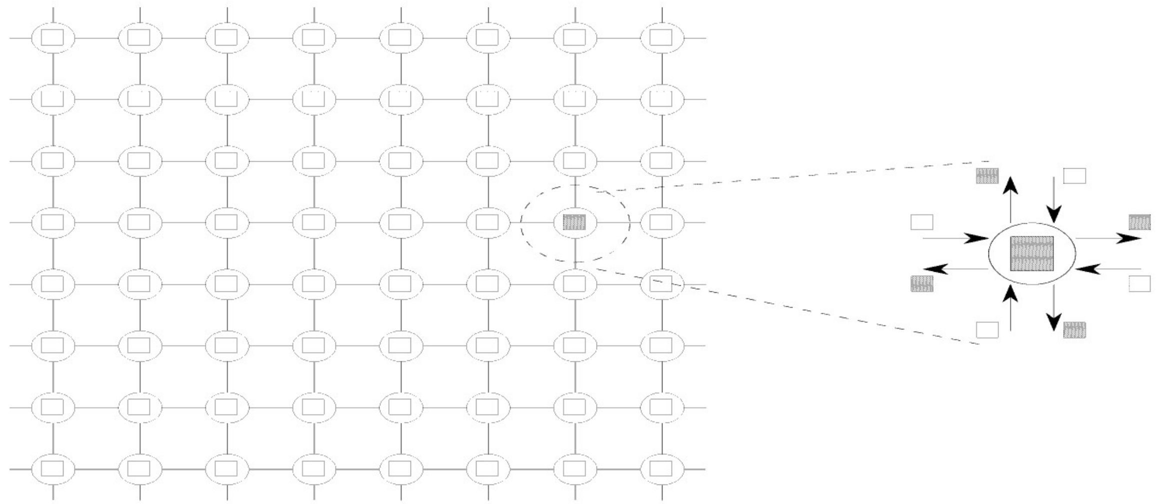
- Determine if worthwhile to replicate data or computation


(a)
(b)
(c)
(d)

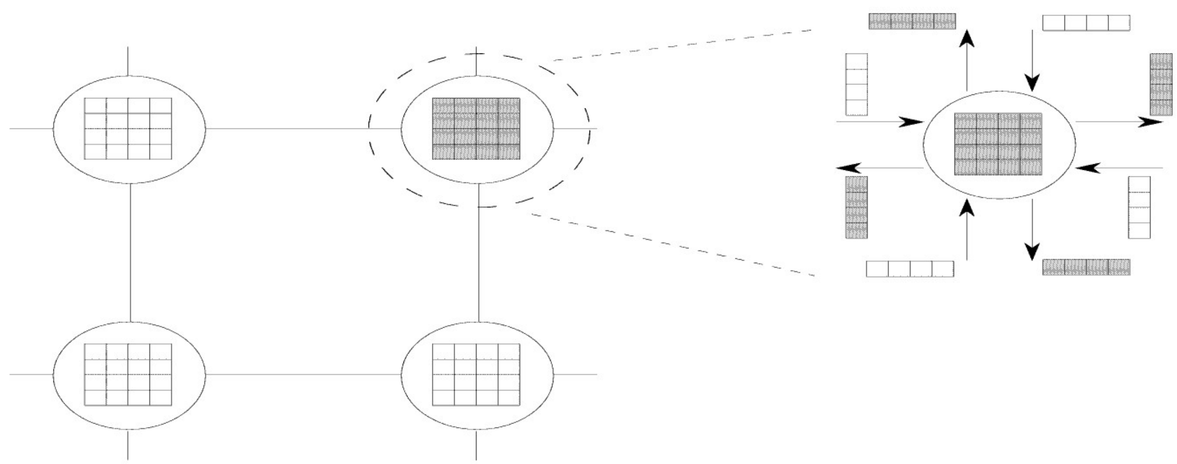
Increasing Granularity

- In partitioning phase, efforts focused on exposing parallelism
- Large number of parallel tasks does not result in efficient parallel algorithm
  - For comm, send same data in less number messages
  - Less task creation costs and task scheduling costs
- **Fine-grain** vs **coarse-grain** parallelism
  - Dependent on the algorithm and the hardware environment in which it runs

(a)

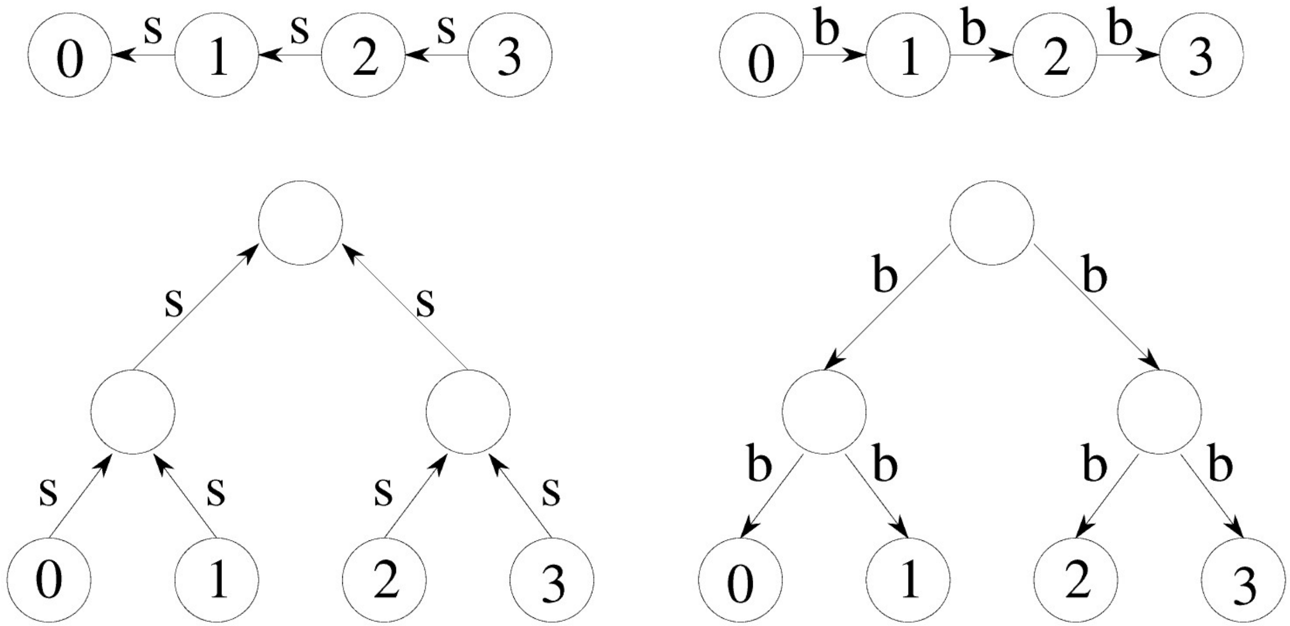


(b)



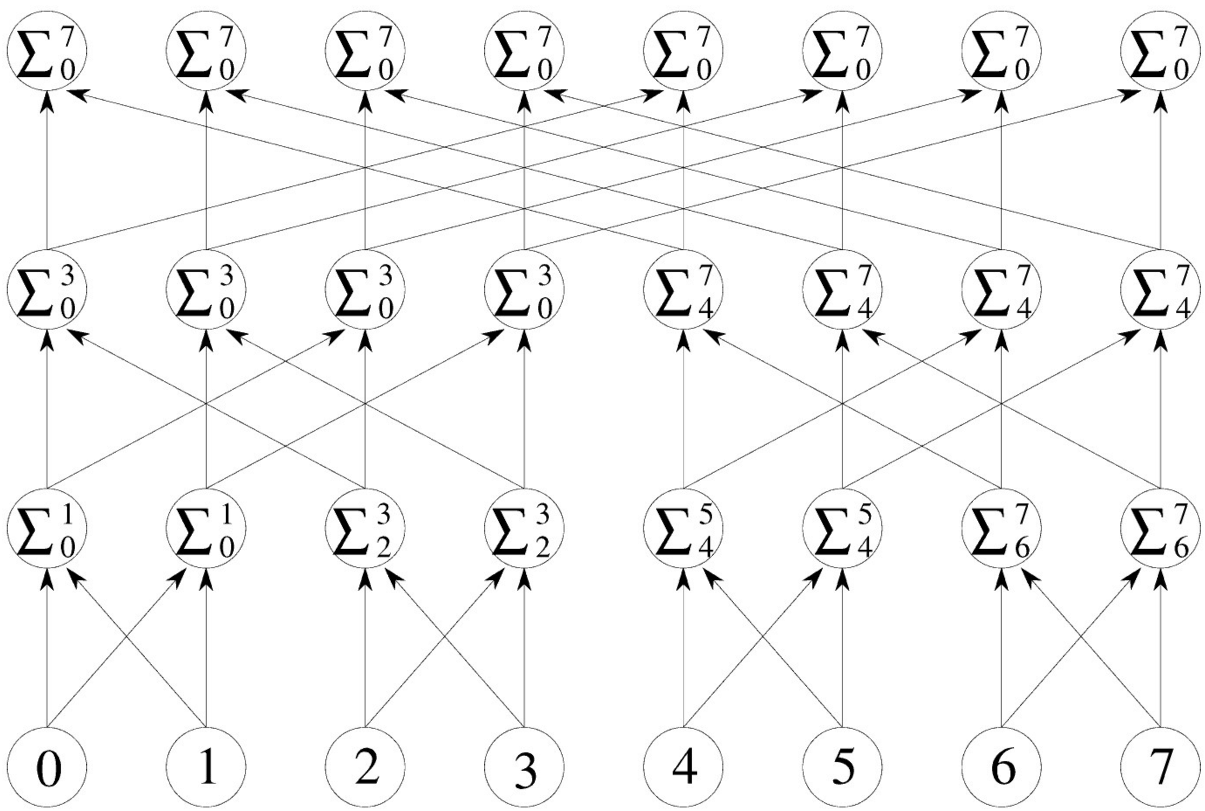
Replicating Computations

- Consider problem of replicating the sum of N numbers in N processors
- Use a sum reduction followed by a broadcast takes 2(N-1) in ring and 2logN for a tree



Replicating Computations

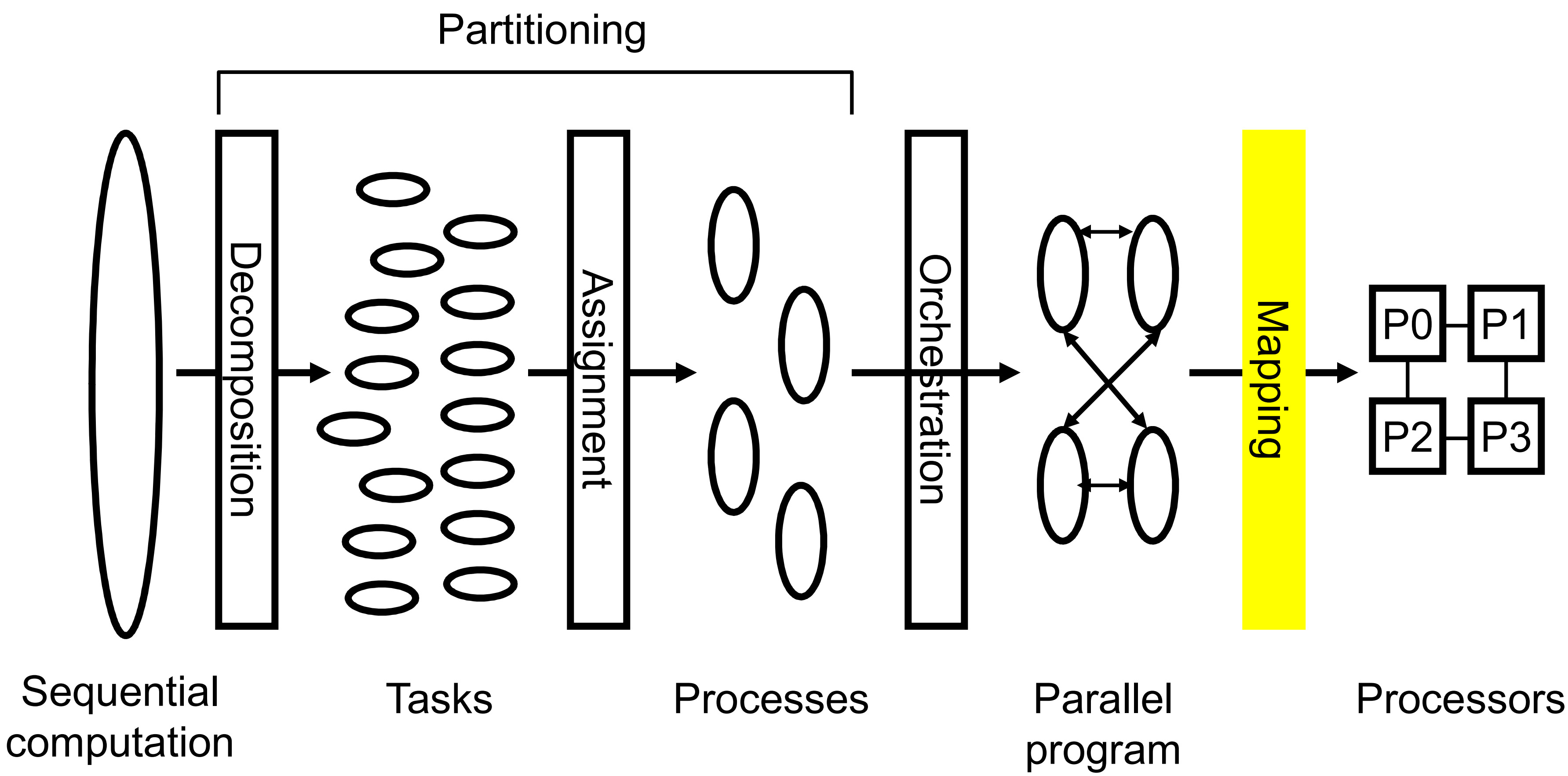
- Can accomplish in log(N) steps in butterfly algorithm



Agglomeration Checklist

- Has agglomeration reduced comm costs by increasing locality?
- If agglomeration uses replicated computations, do benefits outweight costs?
- If agglomeration replicates data, is scalability affected?
- Has agglomeration yielded tasks of similar computation and comm costs?
- Does number of tasks still scale with problem size?

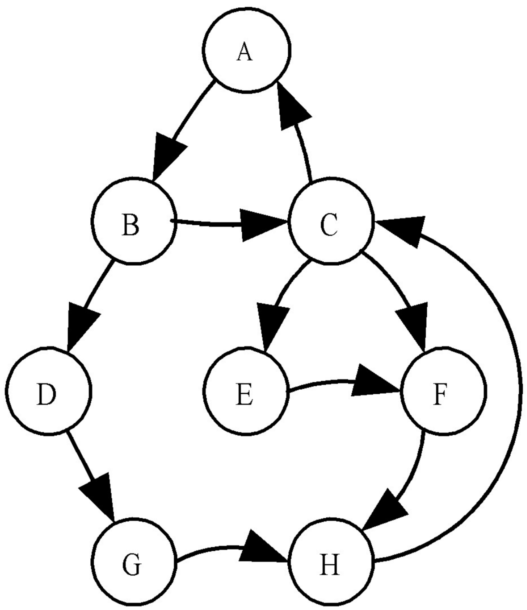
Phases in the Parallelization Process



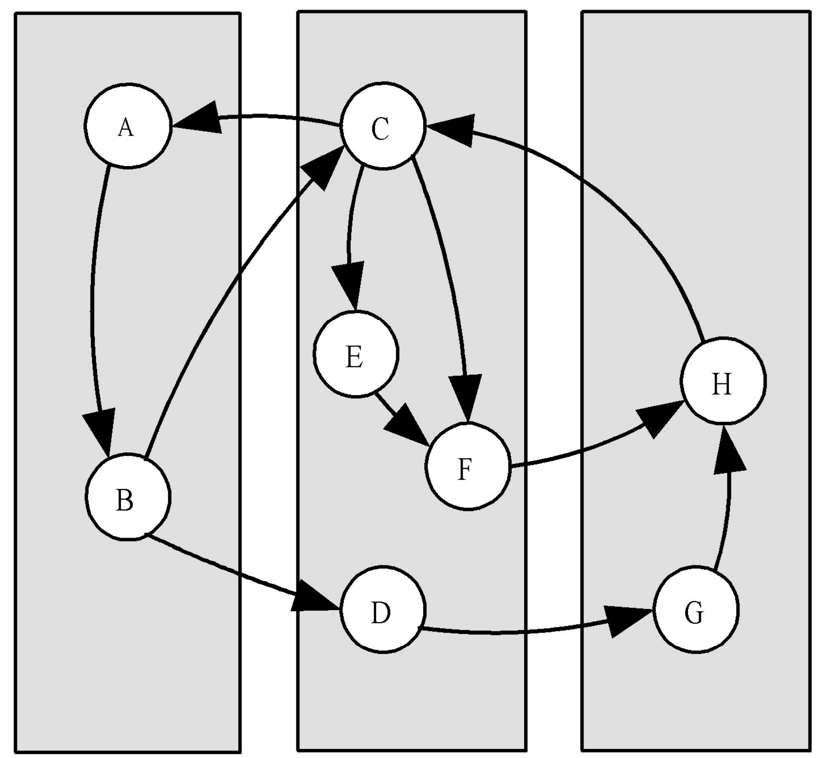
Mapping

- Mapping processes to processors
- Done by the program and/or operating system
- Shared memory system: mapping done by operating system
- Distributed memory system: mapping done by user
- Conflicting goals of mapping
  - Maximize processor utilization
  - Minimize interprocessor communication

Mapping Example



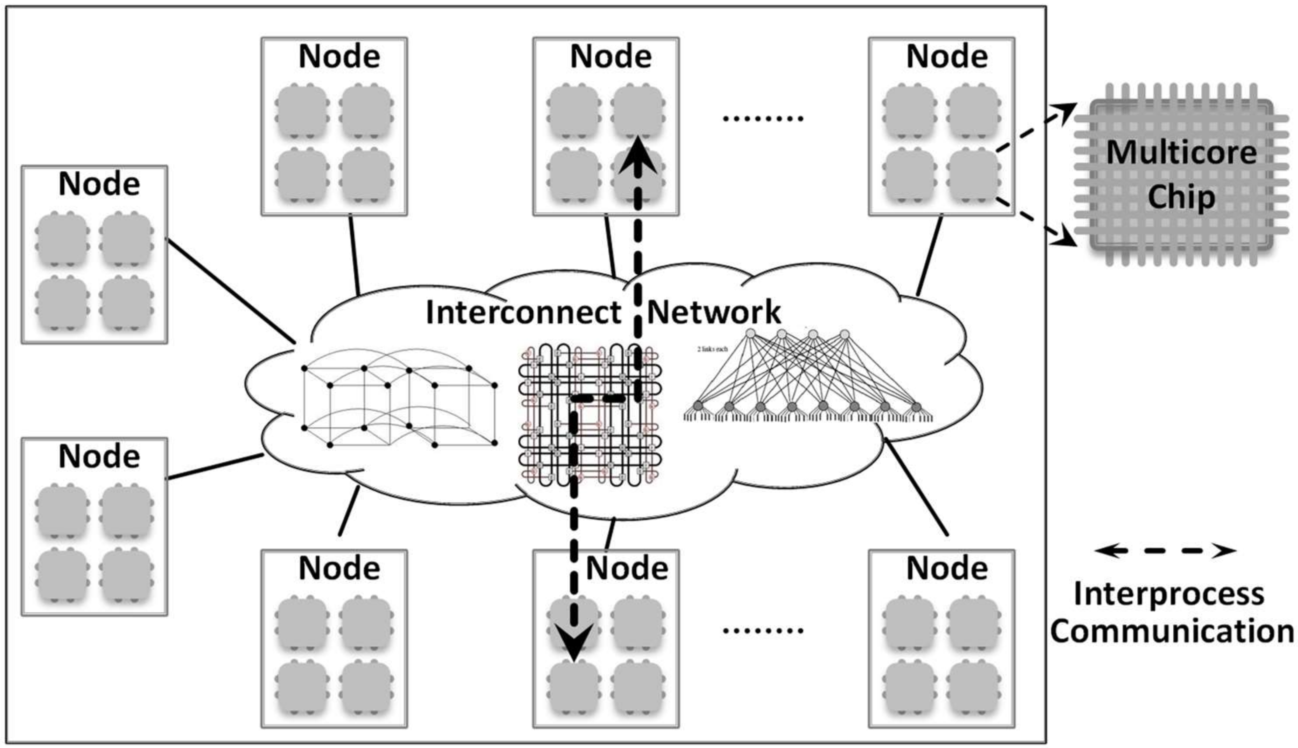
(a)



(b)

Mapping Problem

- Mapping problem to minimize execution time is **NP-complete** (definition?)
  - Hence resort to heuristics
- On modern computer systems, it is also a multilevel problem



Performance Goals

Step	Architecture-Dependent?	Major Performance Goal
Decomposition	Mostly No	➢ Expose enough concurrency but not too much
Assignment	Mostly no	➢ Balance workload ➢ Reduce communication volume
Orchestration	Yes	➢ Reduce unnecessary communication via data locality ➢ Reduce communication and synchronization cost as seen by the processor ➢ Reduce serialization at shared resources
Mapping	Yes	➢ Put related processes on the same processor if necessary ➢ Exploit locality in network topology

Application Structure

- Frequently used patterns for parallel applications:
  - Single Program Multiple Data – SPMD (Domain Decomposition)
  - Embarrassingly Parallel
  - Master / Slave
  - Work Pool
  - Divide and Conquer
  - Pipeline
  - Competition