Parallel Algorithm Design

CS595, Fall 2010

Programming Models

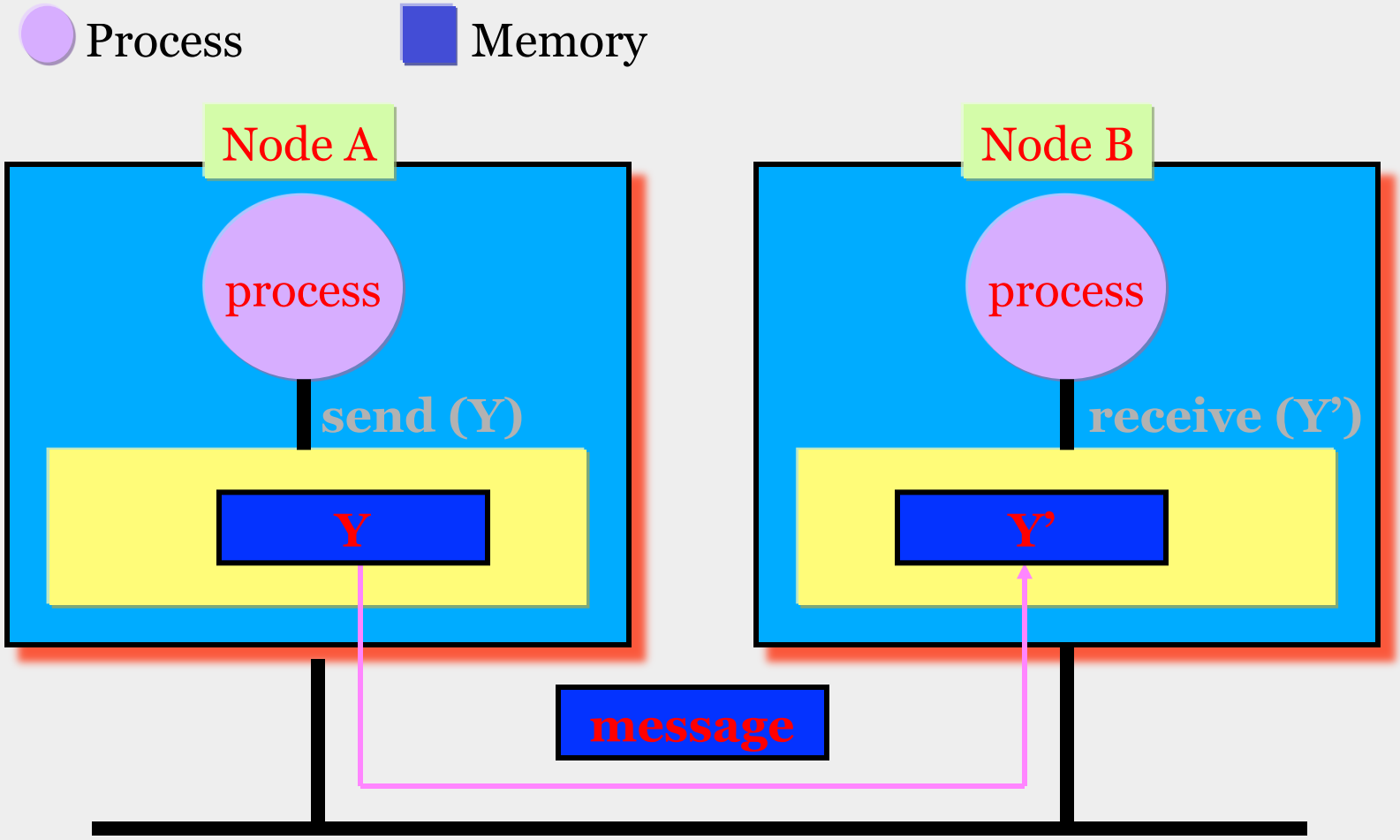
- The programming model
  - determines the basic concepts of the parallel implementation and
  - abstracts from the hardware as well as from the programming language or API.

- The names used for programming models differ in the literature.

Programming Models

1. Sequential Model: The sequential program is automatically parallelized.
   - Advantage: Familiar programming model
   - Disadvantage: Limitations in compiler analysis
2. 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

MP Programming Model

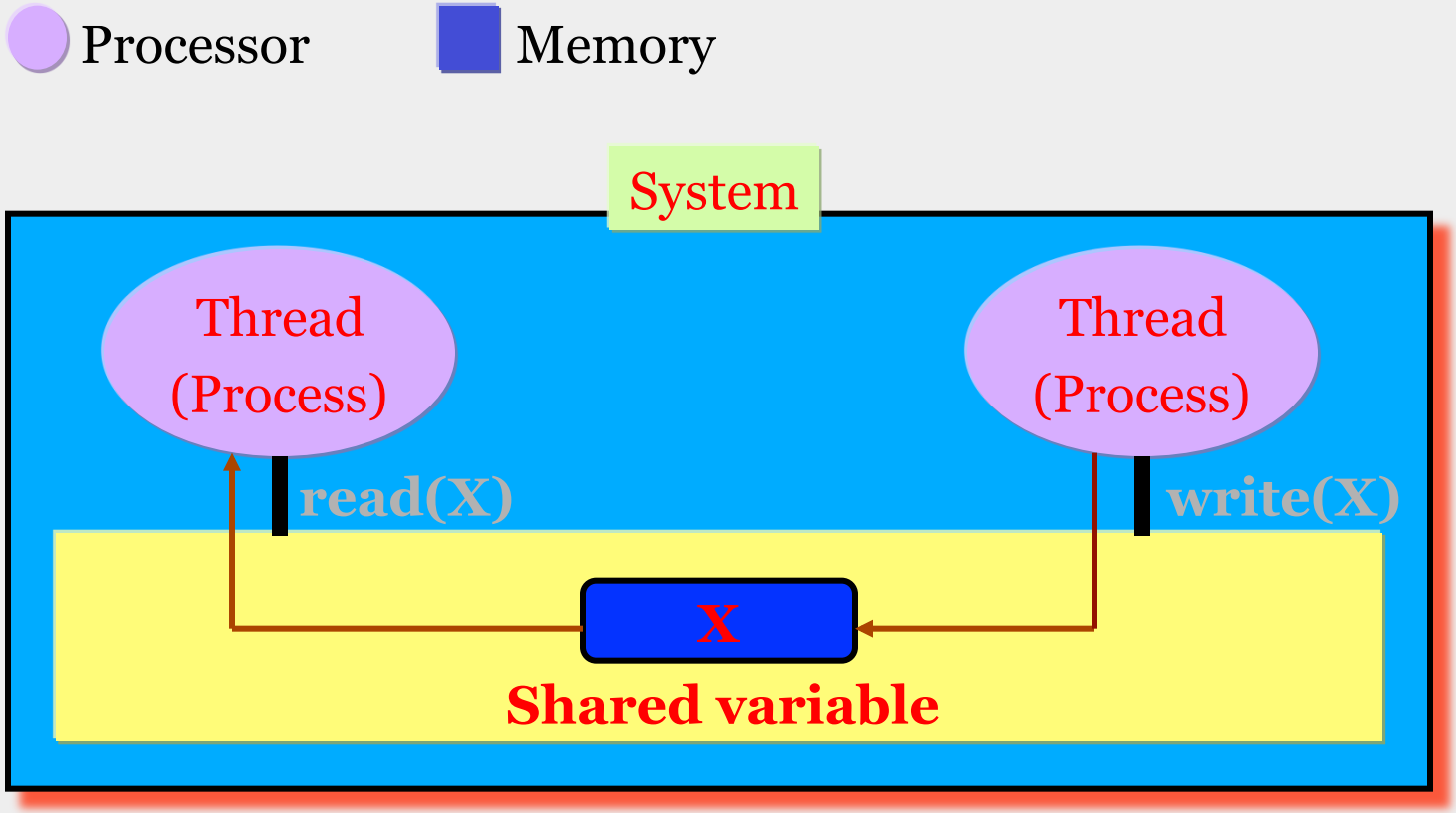


MP Programming Model

- locality is fully transparent

- error prone programming.
- difficult to build performance portable programs

3. Shared Memory Programming Model



Shared or Distributed Memory?

www.mcs.anl.gov/petsc/petsc-as/documentation/faq.html#computers

What kind of parallel computers or clusters are needed to use PETSc?

PETSc can be used with any kind of parallel system that supports MPI. BUT for any decent performance one needs

- a fast, low-latency interconnect; any Ethernet, even 10 gigE simply cannot provide the needed performance.
- high per-CPU memory performance. Each CPU (core in dual core systems) needs to have its own memory bandwith of roughly 2 or more gigabytes. For example, standard dual processor "PC's" will not provide better performance when the second processor is used, that is, you will not see speed-up when you using the second processor. This is because the speed of sparse matrix computations is almost totally determined by the speed of the memory, not the speed of the CPU.

Concepts of Parallel Programming

- Concepts:
  - Task:

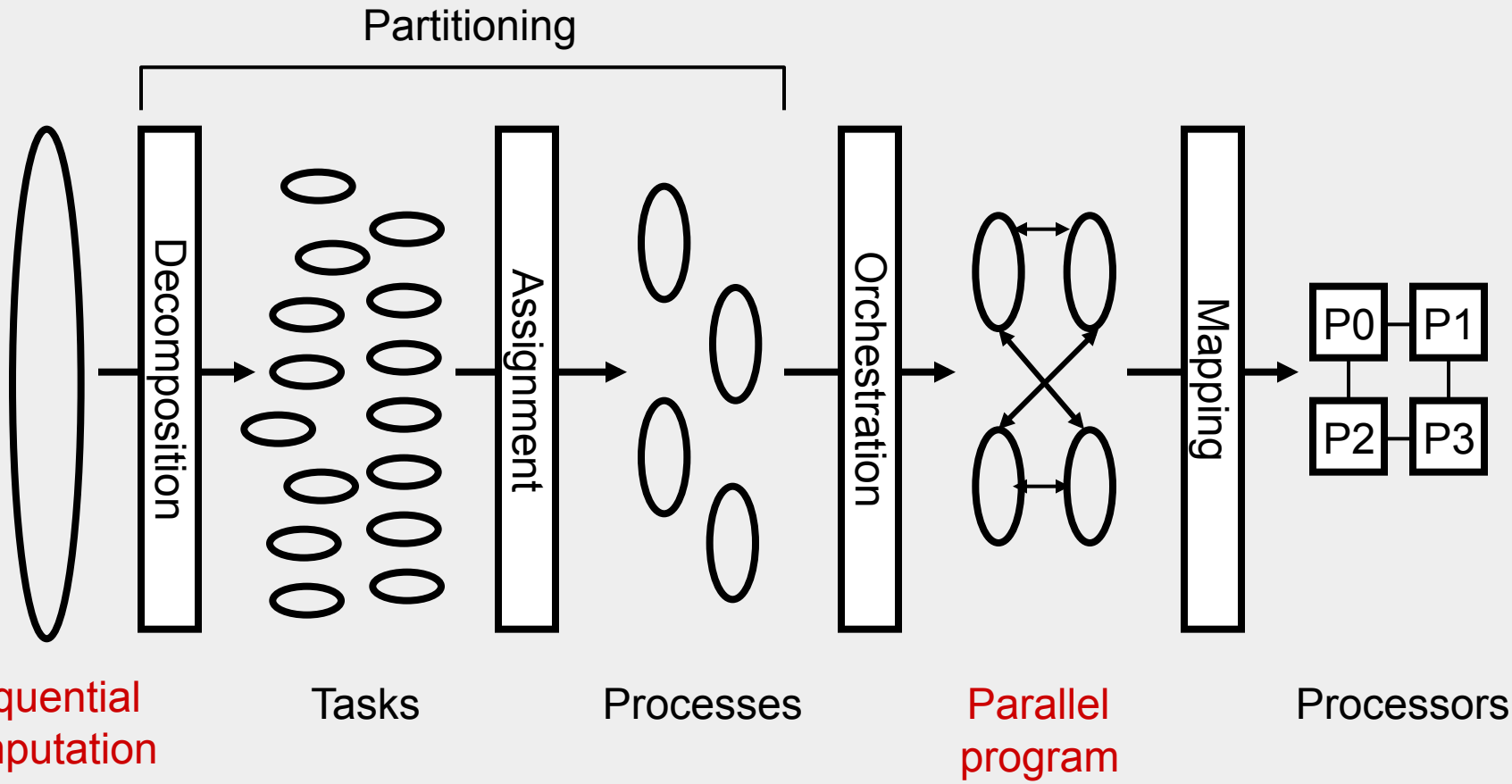
    arbitrary piece of work performed by a single process
  - Thread:

    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



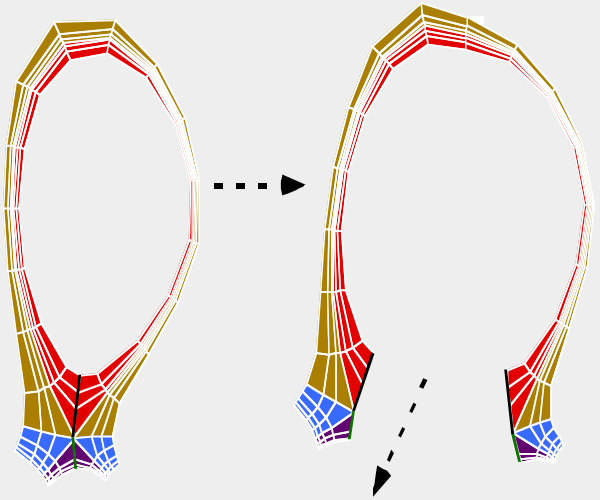
Decomposition

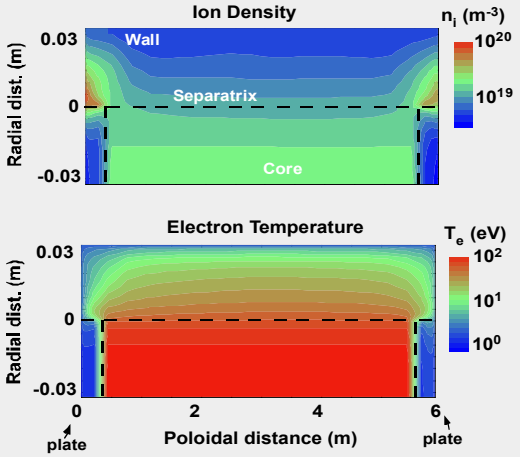
- Dividing computation and data into pieces
- Domain decomposition
  - Divide data into pieces
  - Determine how to associate computations with the data
- Functional decomposition
  - Divide computation into pieces
  - Determine how to associate data with the computations

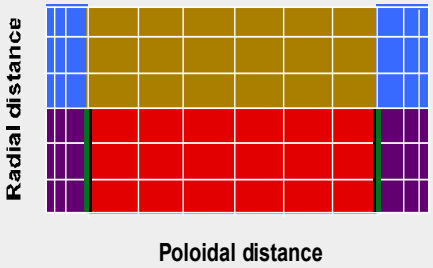
Example of Decomposition:

UEDGE: 2D decomposition

Physical mesh is based on magnetic flux surfaces (here DIII-D)

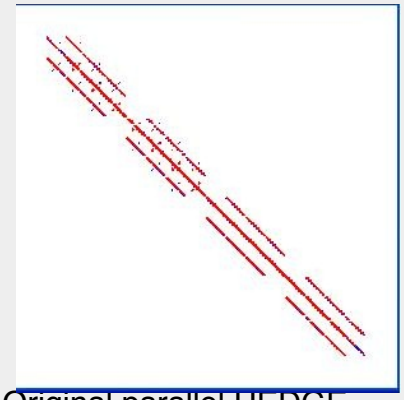




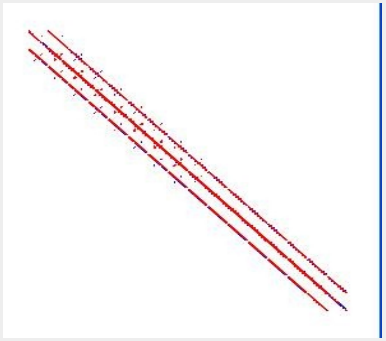


Recent new capability:

Computing sparse parallel Jacobian using matrix coloring



Original parallel UEDGE Jacobian (block Jacobi only)



Recent progress: Complete Jacobian data, enabling use of many different preconditioners

Functional Decomposition

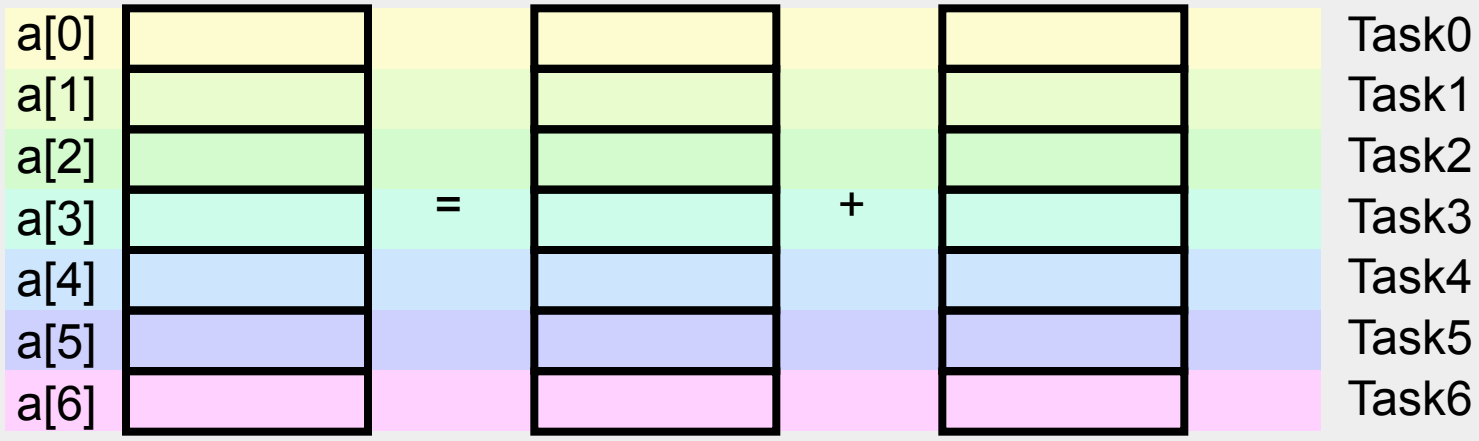
- Breaking the computation into a collection of tasks
  - Tasks may be of variable length
  - Tasks require data to be executed
  - Tasks may become available dynamically

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. Many applications have tasks of same length.

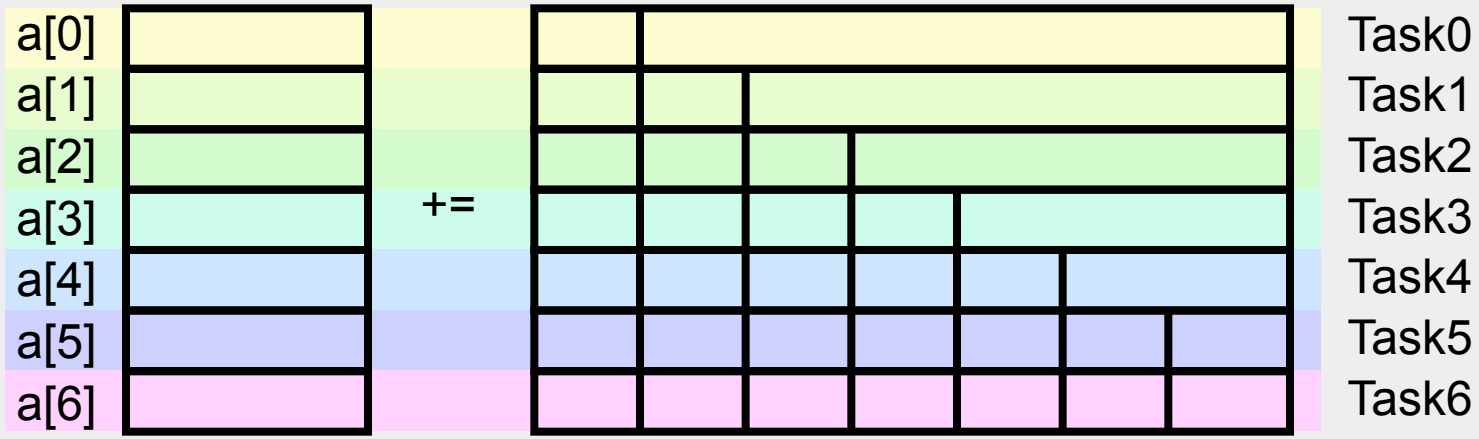
Example: Data Parallelism

```
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);
```

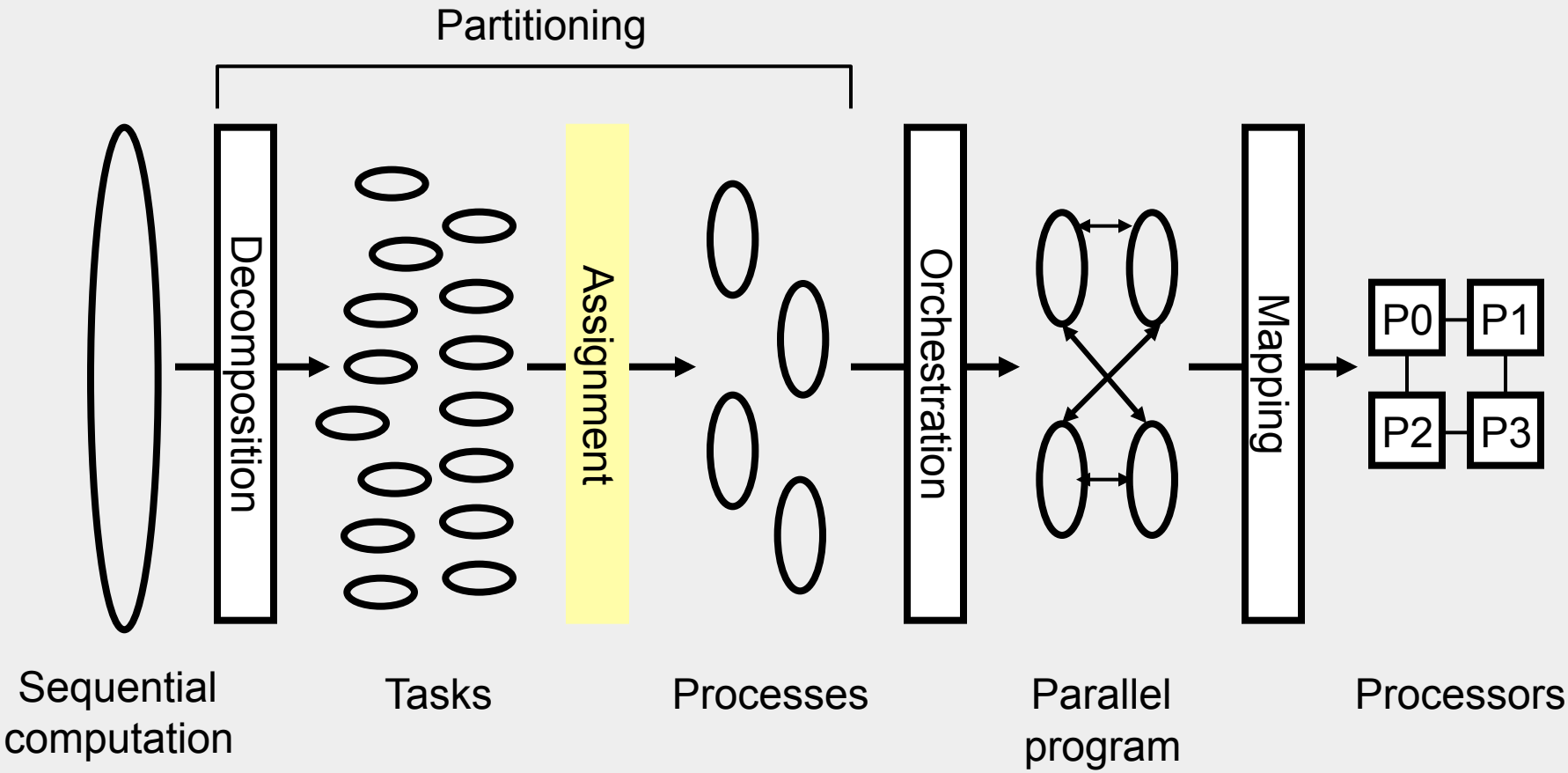
```
i=i+10;
j=2*k+z**2;
```

- The functions or statements can be executed in parallel.

- These are different operations

  → functional parallelism

Phases in the Parallelization Process

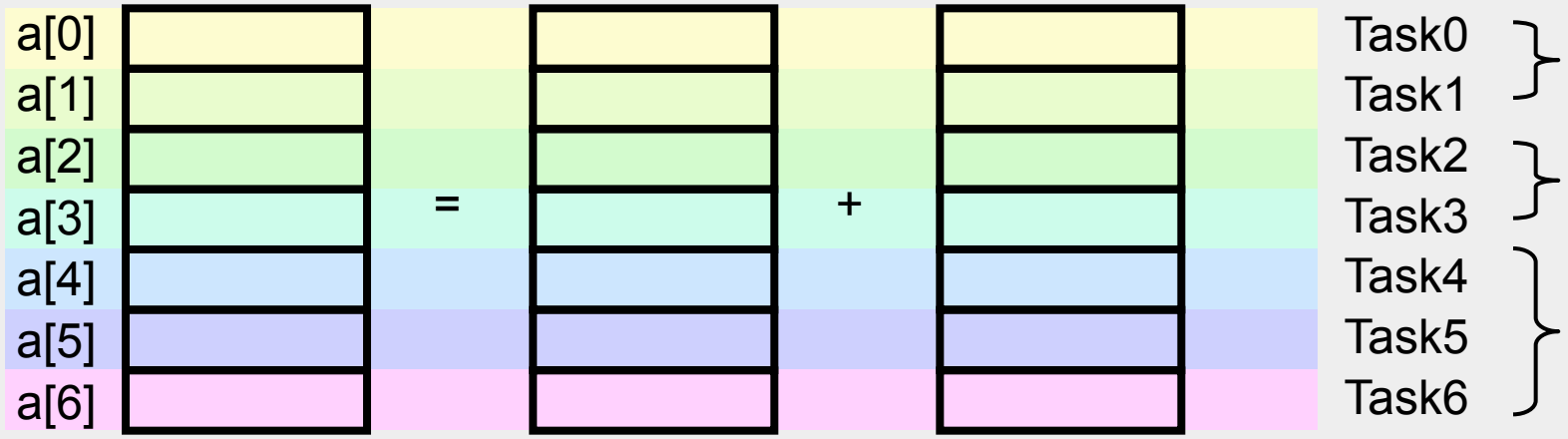


Assignment

- 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

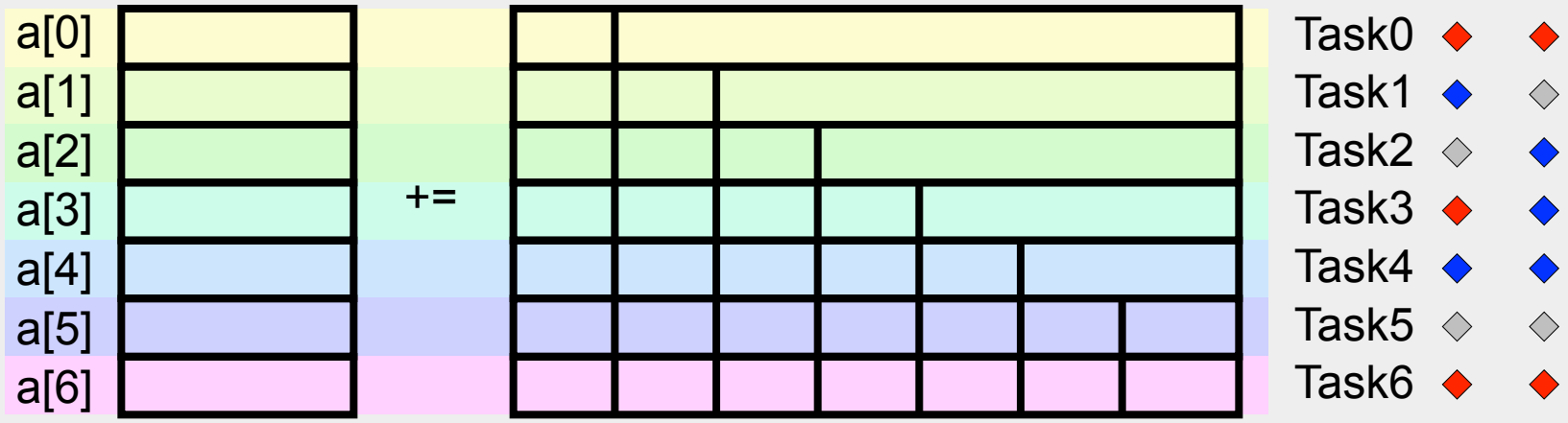
Example: Balance Workload

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



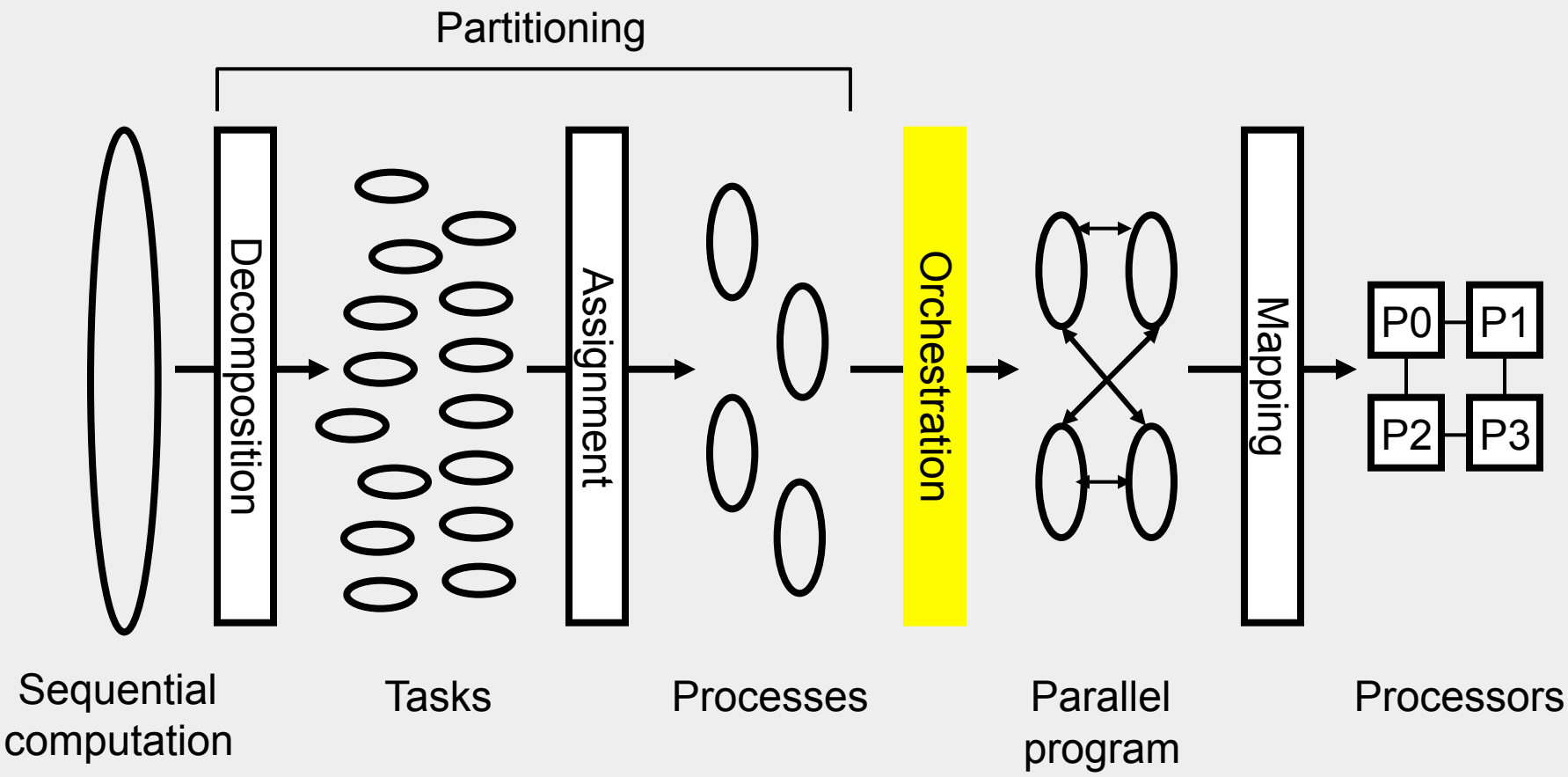
Example: Balance Workload

```
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

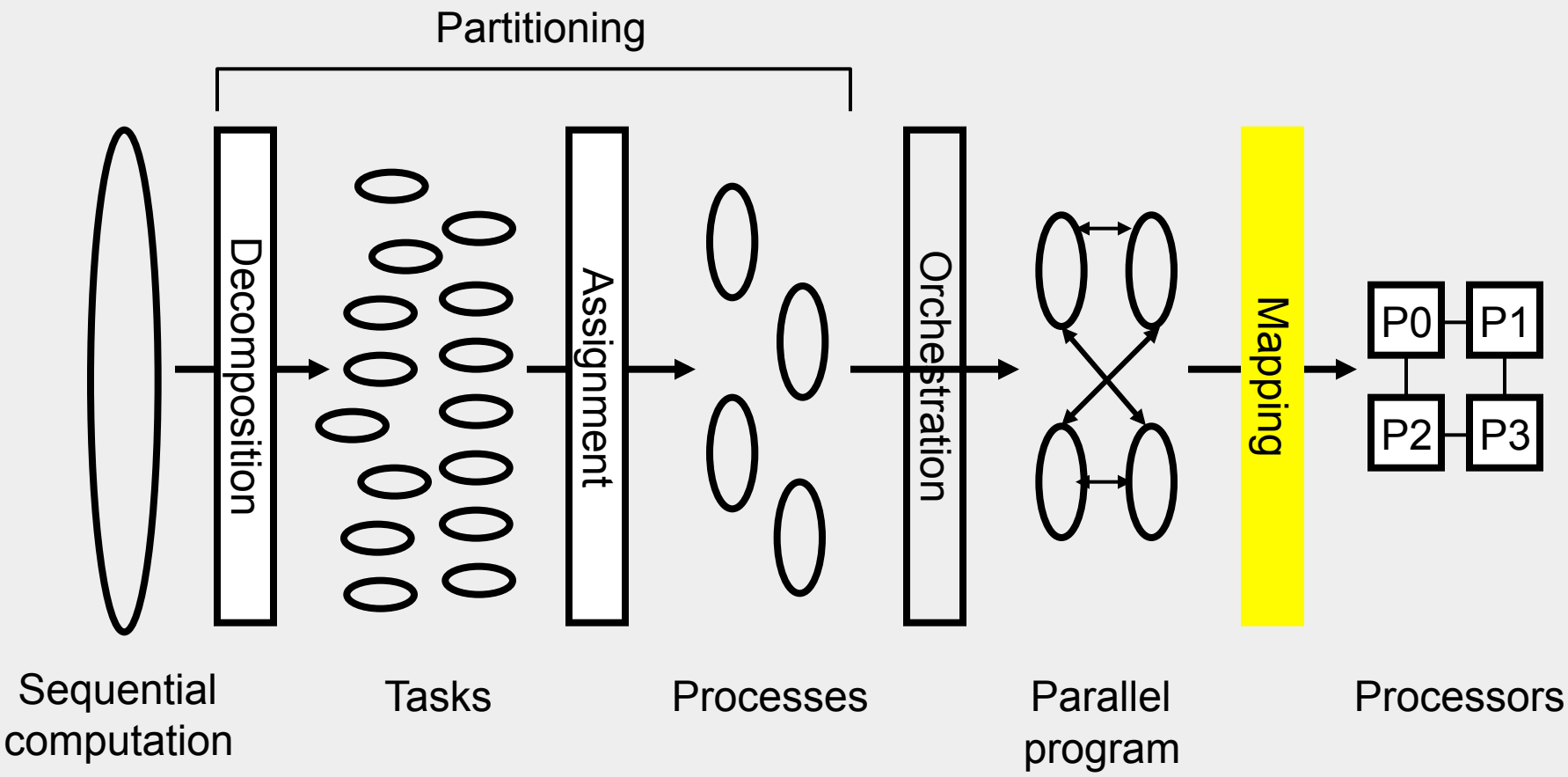
Phases in the Parallelization Process



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
  - Load balancing
- Goals can be conflicting
  - reduction of communication vs. load balancing

Phases in the Parallelization Process



Mapping

- Mapping processes to processors

  Done by the program and/or operating system

- Shared memory system:

  done by operating system

- Distributed memory system:

  done by user or

  by runtime library such as MPI

Mapping

- Goal
  - If there are more processes than processors:
    put multiple related processes on the same processor
  - This may also be an option for heavily interacting processes no matter how many processors are available.
  - Exploit locality in network topology.
    - place processes close to needed data
  - Maximize processor utilization
    Minimize interprocessor communication

Optimal Mapping

- Finding optimal mapping is NP-hard
- Must rely on heuristics

Performance Goals

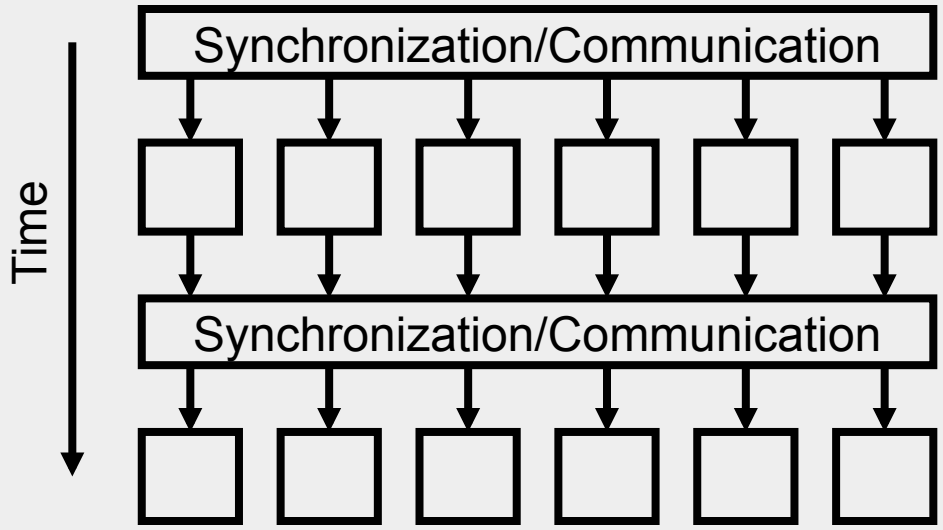
Step	Architecture-Dependent?	Major Performance Goal
Decomposition	Mostly No	➢Expose enough concurrency but not too much
Assignment	Mostly no	➢Balance workload ➢Reduce communication volume
Orchestration	Yes	➢Reduce unnecessary communication via data locality ➢Reduce communication and synchronization cost 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
  - Embarrassingly Parallel
  - Master / Slave
  - Work Pool
  - Divide and Conquer
  - Pipeline
  - Competition

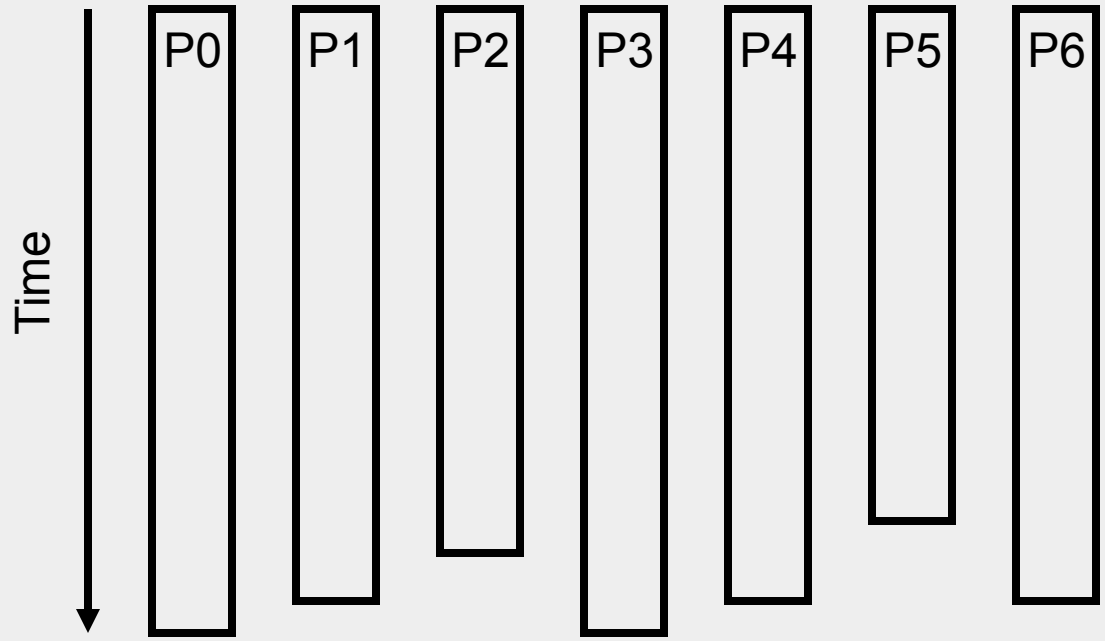
Structure: Single Program Multiple Data

- Single program is executed in a replicated fashion.
- Processes or threads execute same operations on different data.
- Loosely-synchronous: Sequence of phases of computation and communication/synchronization.



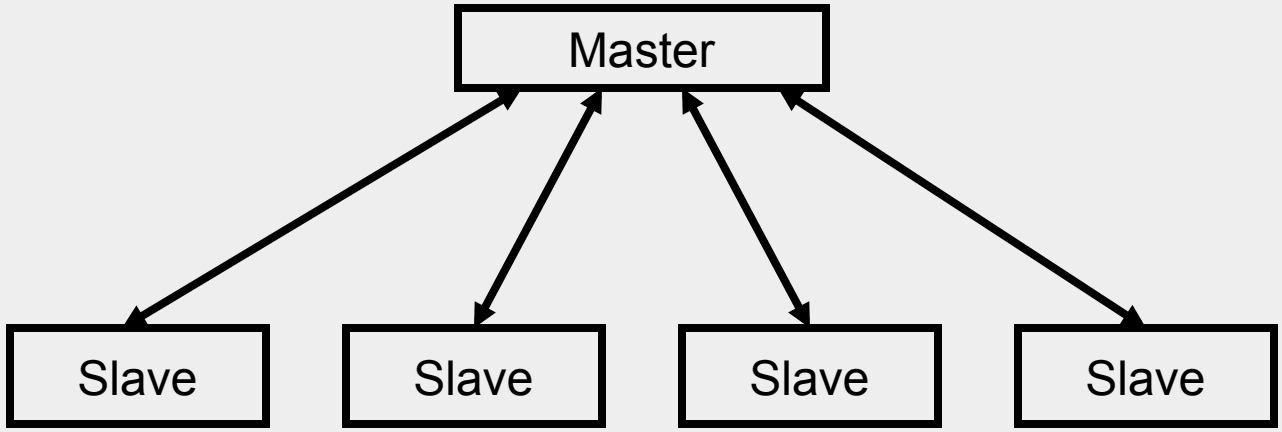
Structure: Embarrassingly Parallel

- Multiple processes are spawned at the beginning.
- They execute totally independent of each other.
- Application terminates after all processes terminated.



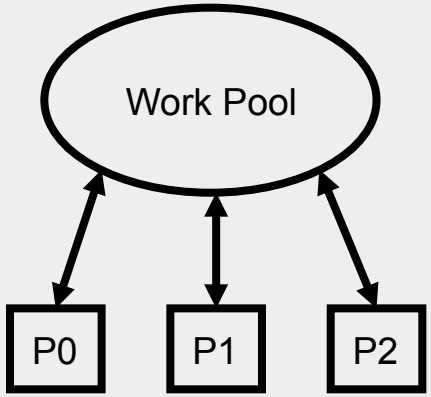
Structure: Master / Slave

- One process executes as a master. It distributes tasks to the slaves and receives the results from the slaves.
- Slaves execute the assigned tasks usually independent of the other slaves.
- Frequently used on workstation networks.



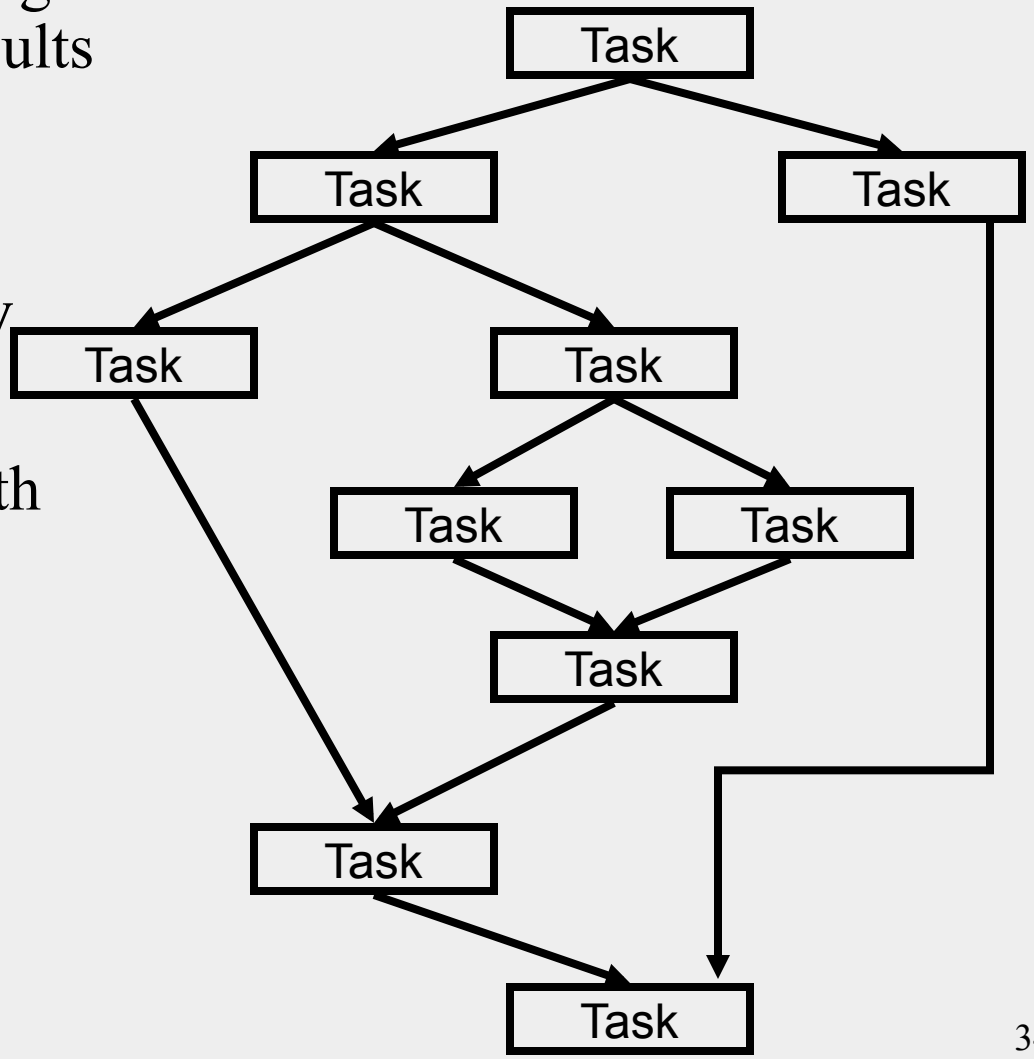
Structure: Work Pool

- Processes fetch tasks from a pool and insert new tasks into the pool.
- Pool requires synchronization.
- Large parallel machines require a distributed work pool.
- Leads to better load balancing.



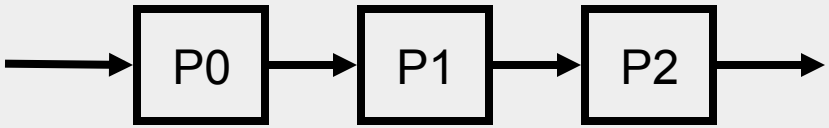
Structure: Divide and Conquer

- Recursive partitioning of tasks and collection of results
- Problems:
  - load balancing
  - least granularity
- Used on systems with background load



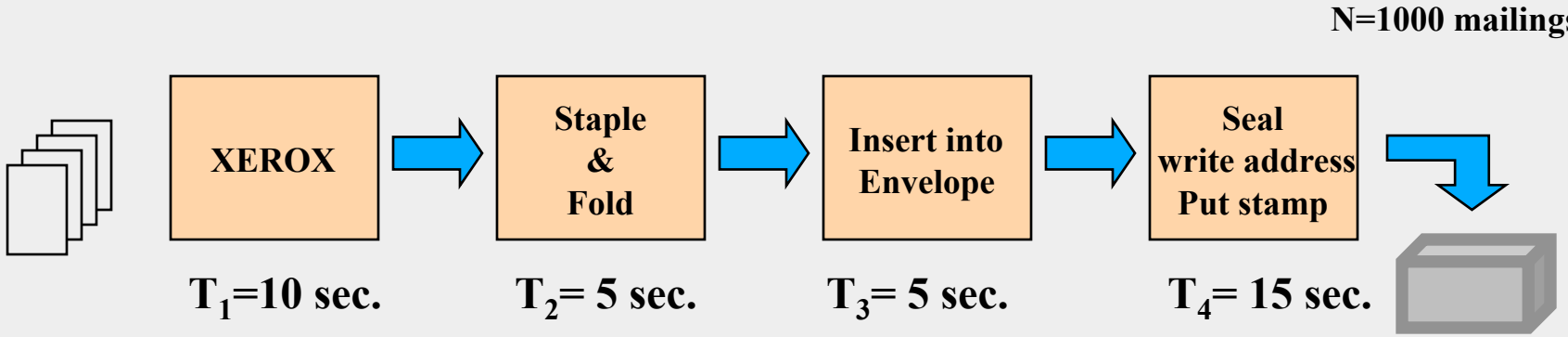
Structure: Pipeline

- Examples
  - Different functions are applied to data: functional decomposition
  - Parallel execution of functions for different data.
  - Signal and image processing
  - Groundwater flow, flow of pollutants, visualization
  - Almost no example of high parallelism



Pipelining: Example

We would like to prepare and mail 1000 envelopes each containing a document of 4 pages to members of an association.



- At what intervals, do we see a new envelope prepared for mailing?

  $\text{Max}(T_1, T_2, \ldots, T_k) = T_{max} = 15$ sec.

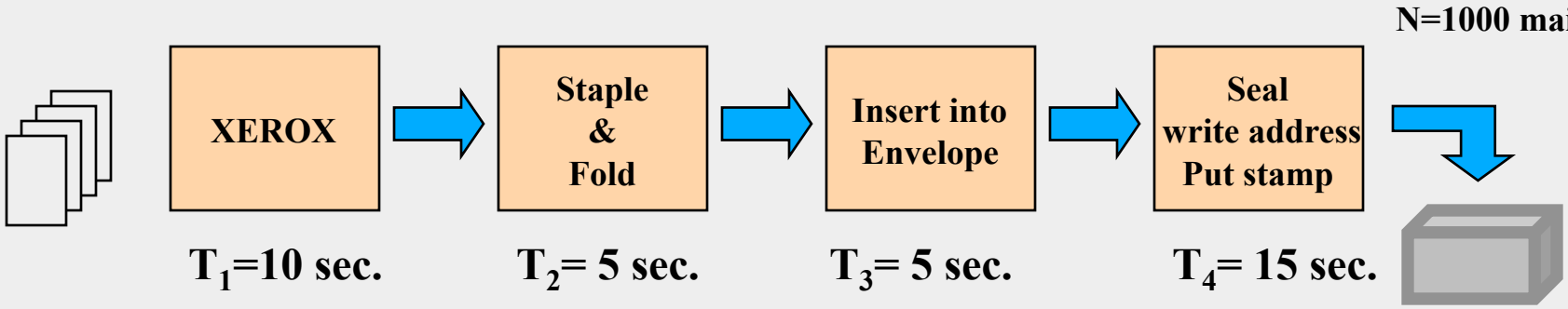
- What is the total time to get N envelopes prepared?

  $\text{Time} = \text{Cold\_Start\_Time} + T_{max} * (N-1) \cong N * T_{max}$

- What is the total time we would have spent if pipelining is not used?

  $N * \Sigma_i T_i$

Pipelining: Example (contd.)



How much speedup do we get?

Speedup = $T_{seq}/T_{pipe} = [\ N*\sum_i T_i\ ]/\ N*\ T_{max} = \sum_i T_i\ /\ T_{max}$

Speedup = 35/15

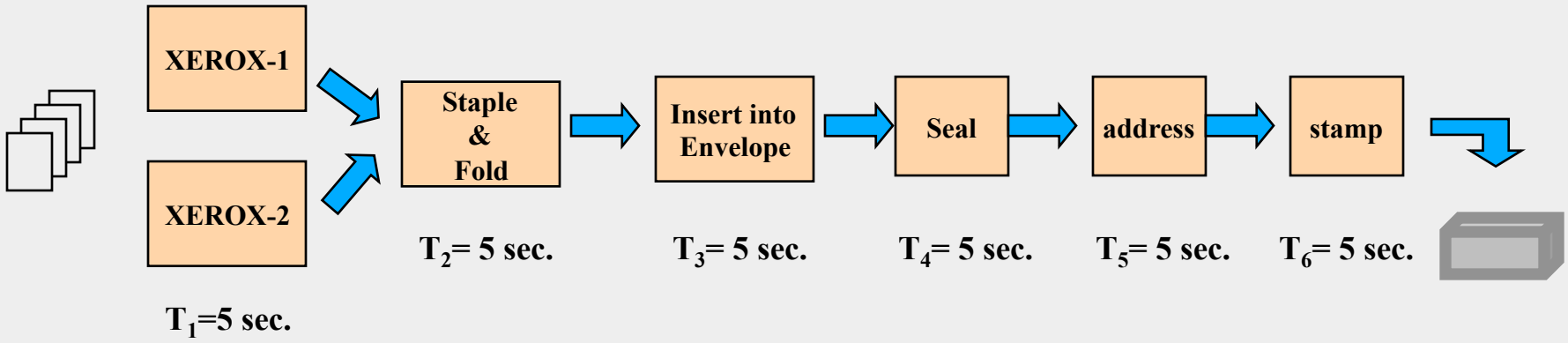
If you can not do much about the completion time for one task (i.e. $\sum_i T_i$); what can you do to maximize the speedup?

(i) Create as many stations (stages) as possible
and

(ii) Try to balance the load at each station, i.e. $T_1 = T_2 = \ldots = T_k$

Pipelining: Example (contd.)

One possible configuration to maximize speedup:



- At what intervals, do we see a new envelope prepared for mailing?

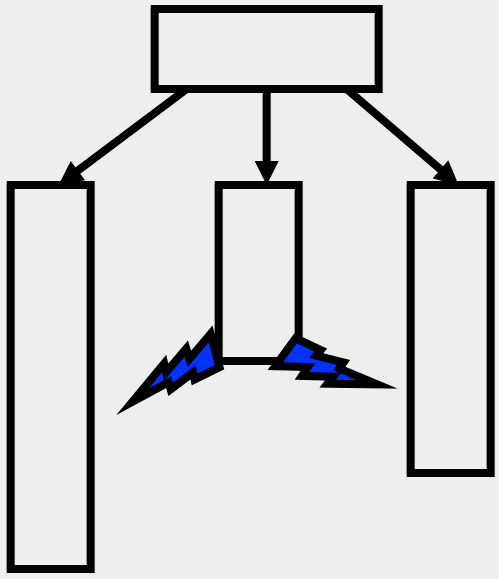
  $\text{Max}(T_1, T_2, \ldots, T_k) = T_{max} = 5$ sec.

- What is the speedup now?

  Speedup = 30/5 = 6 = number of stages in the pipeline !

Structure: Competition

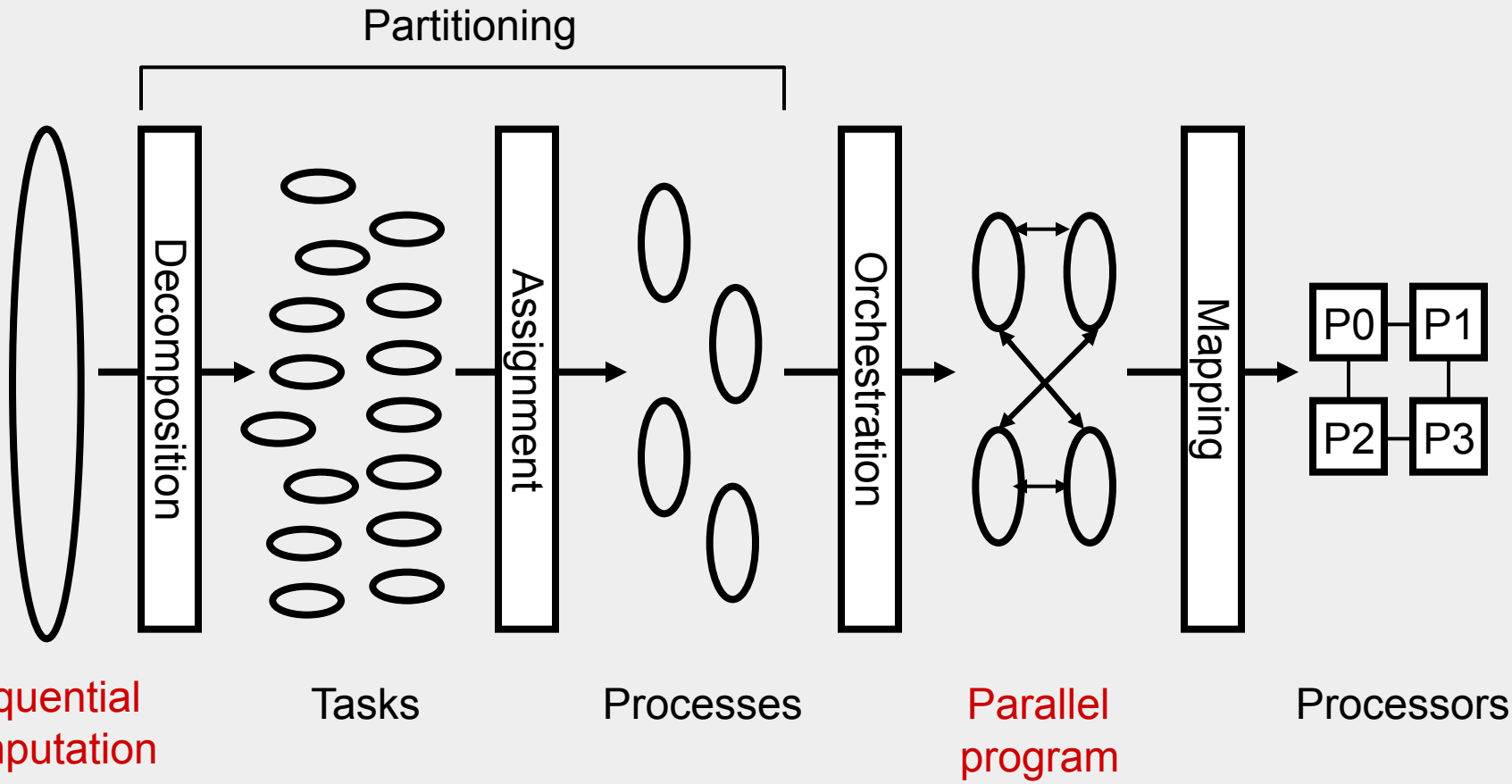
- Evaluation of multiple solution strategies in parallel.
- It might be unknown which strategy is successful or which one is the fastest.
- With k processors, k strategies can be tested. If one of the additional strategies - not tested in the sequential program - is very fast, the speedup can be more than k (Superlinear speedup)



Summary Parallel Programming

- Programming models for SM and DM systems
  - Sequential, Message Passing, Shared Memory
- Phases in the parallelization process
  - Decomposition, assignment, orchestration, mapping

Phases in the Parallelization Process



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
  - Embarrassingly Parallel
  - Master / Slave
  - Work Pool
  - Divide and Conquer
  - Pipeline
  - Competition