Parallel Programming Patterns

Parallel Patterns

- **Patterns**
  - a "**vocabulary**" for **designing** algorithms

- **Parallel Patterns**
  - A **recurring** combination of distribution of **tasks** and access of **data** pertaining to a specific problem in design of parallel algorithm.
  - aid in achieving **scalability** and convenience for developing **parallel** applications .
  - facilitates **comparison** between parallel and serial performance.
  - **Universal** in nature as patterns can be utilized any parallel programming models.

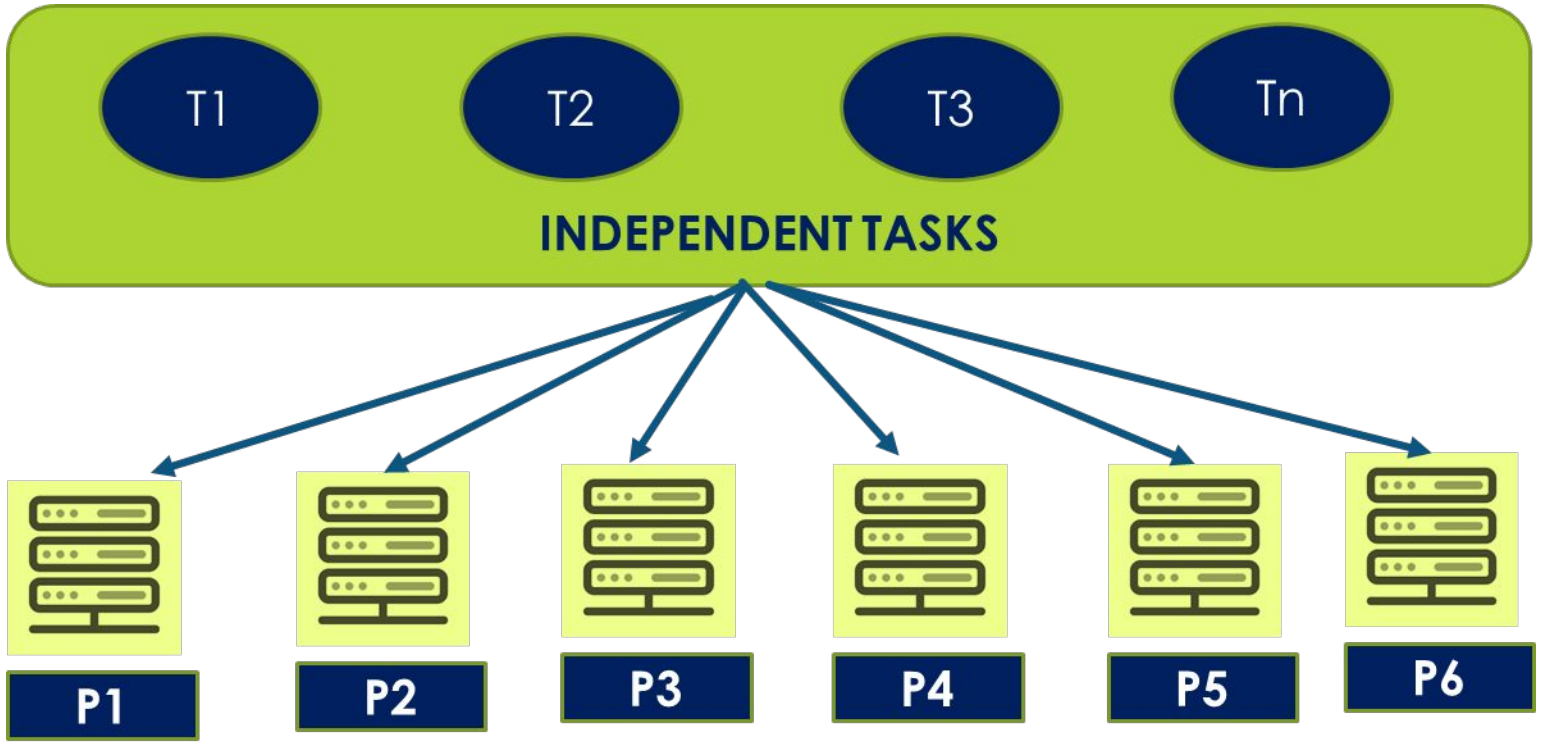
Parallel Patterns

- **Bag of Tasks**
- **Data Parallel**
- **Task Parallel**
- **Pipelining**
- **Divide and Conquer**

Bag-of-Tasks

- **Bag of Tasks (BoT)**
  - Also called **Embarrassingly Parallel** or **loosely coupled** applications
  - Component **tasks** of application are **independent**
  - Tasks **don't communicate** with each other
  - Tasks can be executed in any order
  - **Minimal** parallel **overhead**

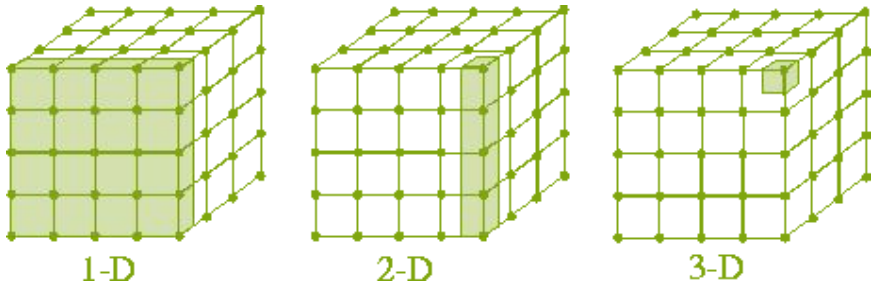
Bag-of-Tasks



Task vs Data Parallel Programs

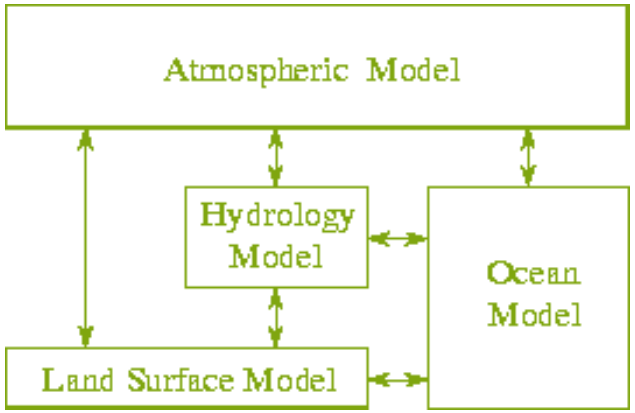
- ***Data Parallel Programs***

  - simultaneous execution of the same task on different data elements.



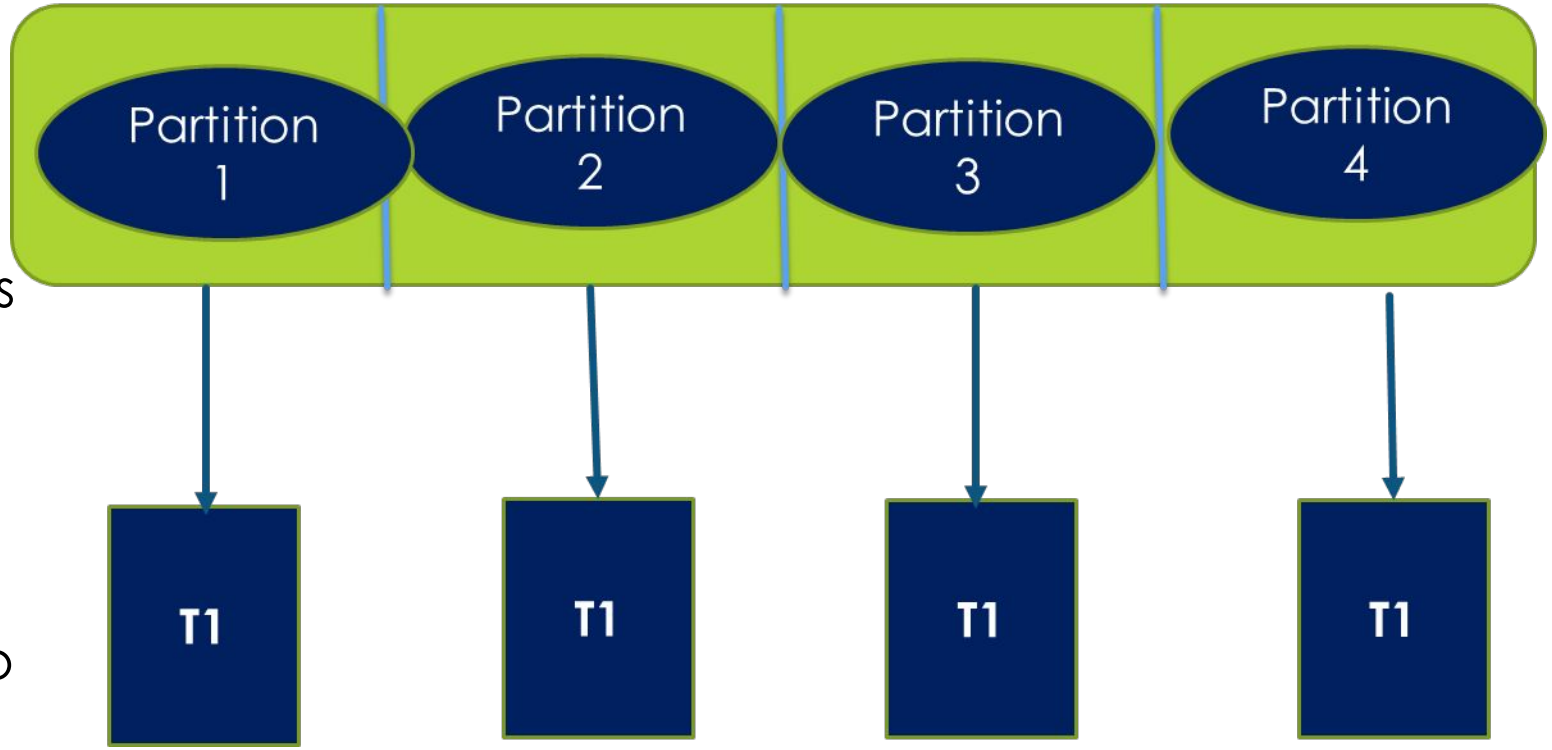
Task Parallel Programs

- simultaneous execution of different tasks on the same data elements.



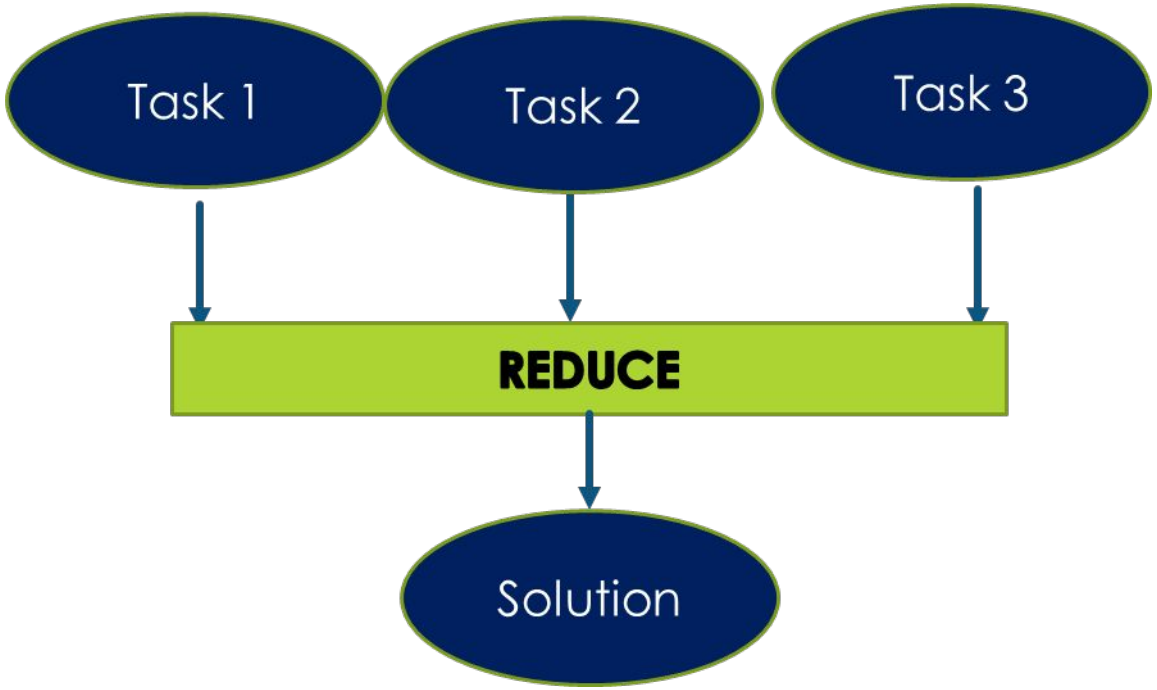
Data Parallel

- Most parallel work in these applications is focused on **performing tasks** on a **data set**.
- a **subset/partition** of the dataset is given to each **process**.
- each **process** performs the same tasks on the different subsets/partition of data.
- **Example:** each **task (T1) adds 3** to each data element.



Task Parallel

- a **subset of the tasks** is allocated to each **process**
- each process performs a different subset of tasks on the same data.
- at the end of the tasks, all of processes have to share the results of the tasks executed, (***global reduction)***.

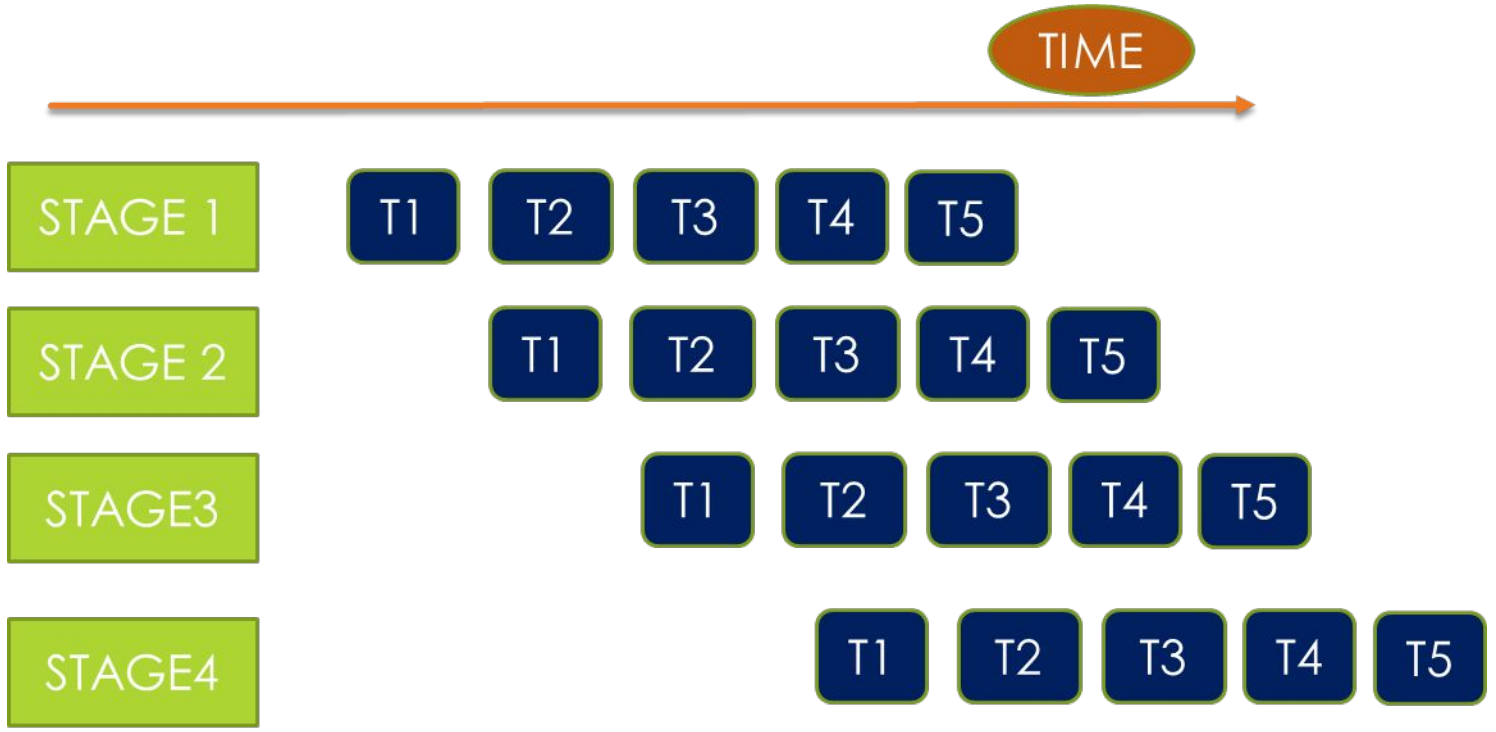


Pipelining

- A **stream of data** is passed through a **succession of processes**, each of which perform some task on it.
- A pipeline is composed of several **computations** called **stages**.
- Computation stages performed on data are **ordered** but **independent**.
- Computation **stages** run **independently** for each item.
- Each **output** of computation becomes **input** to the following computation.

Pipelining

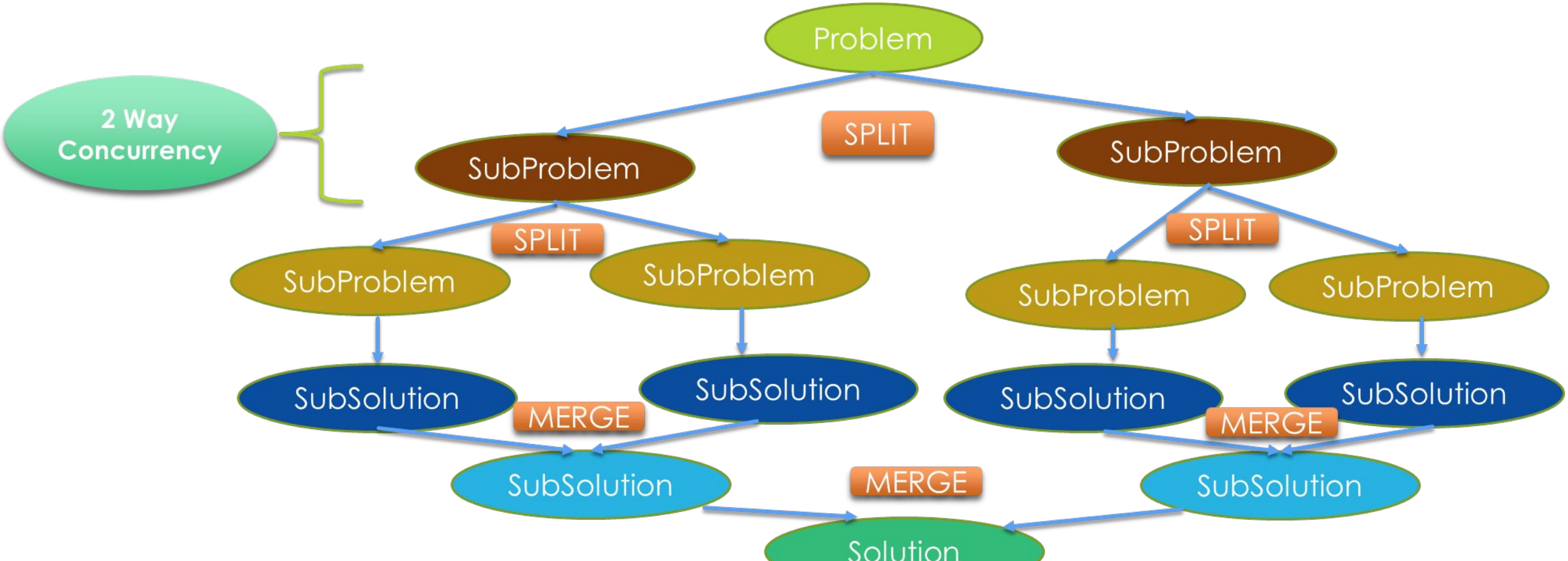
- Enable **modular** design
- Conceptually **simple**
- **Serial** computation still a **bottleneck**



Divide and Conquer

- **Concurrency is obtained by:**
  - **Splitting** the problem into **subproblems**
  - **concurrently** solving the subproblems
  - **merging** the solved subproblems solutions into a solution for the whole problem

Divide and Conquer



Thank You!