departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Parallel Programming Overview

Concurrency and Parallelism — 2019-20

Master in Computer Science

(Mestrado Integrado em Eng. Informática)

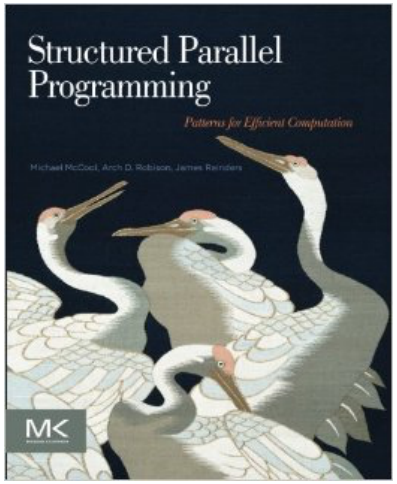
Joao Lourenço <joao.lourenco@fct.unl.pt>

Source: Parallel Computing, CIS 410/510, Department of Computer and Information Science

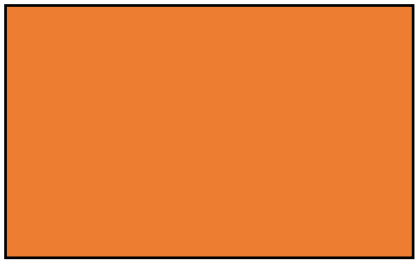
Outline

- Structured programming patterns overview
  - Concept of programming patterns
  - Serial and parallel control flow patterns
  - Serial and parallel data management patterns

  - Bibliography:
    - **Chapter 3** of book
      McCool M., Arch M., Reinders J.;
      Structured Parallel Programming: Patterns for Efficient Computation;
      Morgan Kaufmann (2012);
      ISBN: 978-0-12-415993-8

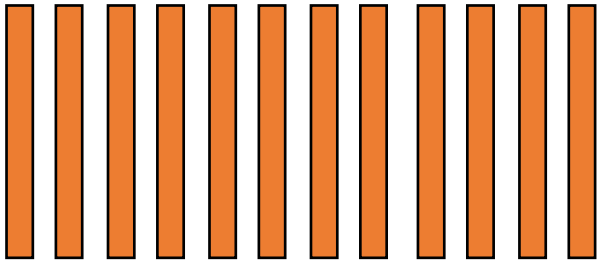


How to Create a Parallel Application



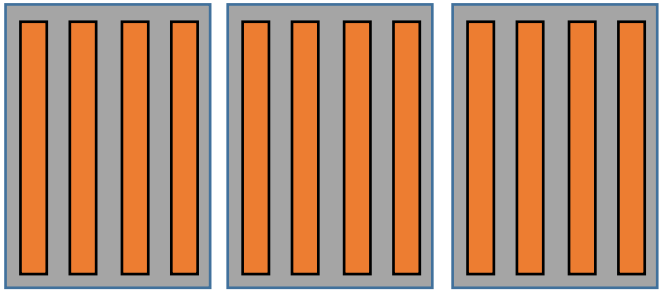
Original problem

Decompose into tasks



Tasks, local and shared data

Group into execution units



Units if execution + new shared data for extracted dependencies

Code with a parallel prog, env.

```
program thread 0 {
}
program thread 1 {
}
program thread 2 {
    …
    for (;;) {
        …
    }
}
```

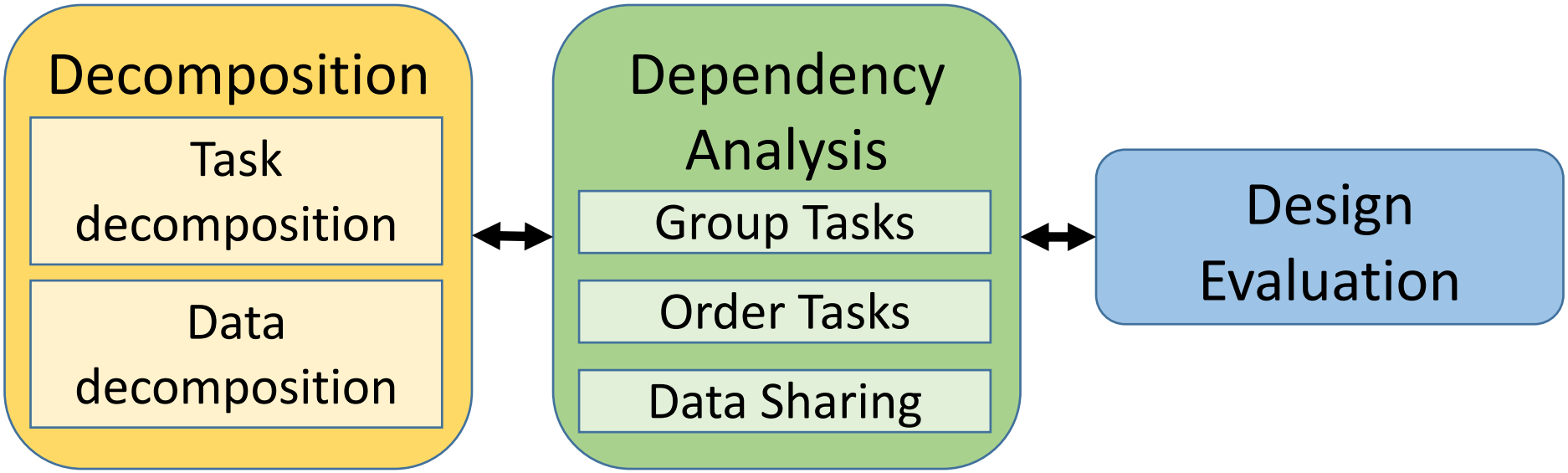
Before writing parallel programs

- Parallel programs often start as sequential programs
  - Easy to write and debug
  - Already developed/tested
- Identify program hot spots
- Parallelization
  - Start with hot spots first
  - Make sequences of small changes, each followed by testing
  - Patterns provide guidance

Steps to Parallel Programming

- Step 1: Find concurrency
- Step 2: Structure the algorithm so that concurrency can be exploited
- Step 3 : Implement the algorithm in a suitable programming environment
- Step 4: Execute and tune the performance of the code on a parallel system

1. Finding Concurrency



- Things to consider: Flexibility, Efficiency, Simplicity

Guidelines for Task Decomposition

- Flexibility
  - Program design should afford flexibility in the number and size of tasks generated
    - Tasks should not tie to a specific architecture
    - Fixed tasks vs. Parameterized tasks
- Efficiency
  - Tasks should have enough work to amortize the cost of creating and managing them
  - Tasks should be sufficiently independent so that managing dependencies doesn't become the bottleneck
- Simplicity
  - The code must remain readable and easy to understand and debug

Guidelines for Data Decompostion

- Data decomposition is often implied by task decomposition
- Programmers need to address task and data decomposition to create a parallel program
  - Which decomposition to start with?
- Data decomposition is a good starting point when
  - Main computation is organized around manipulation of a large data structure
  - Similar operations are applied to different parts of the data structure

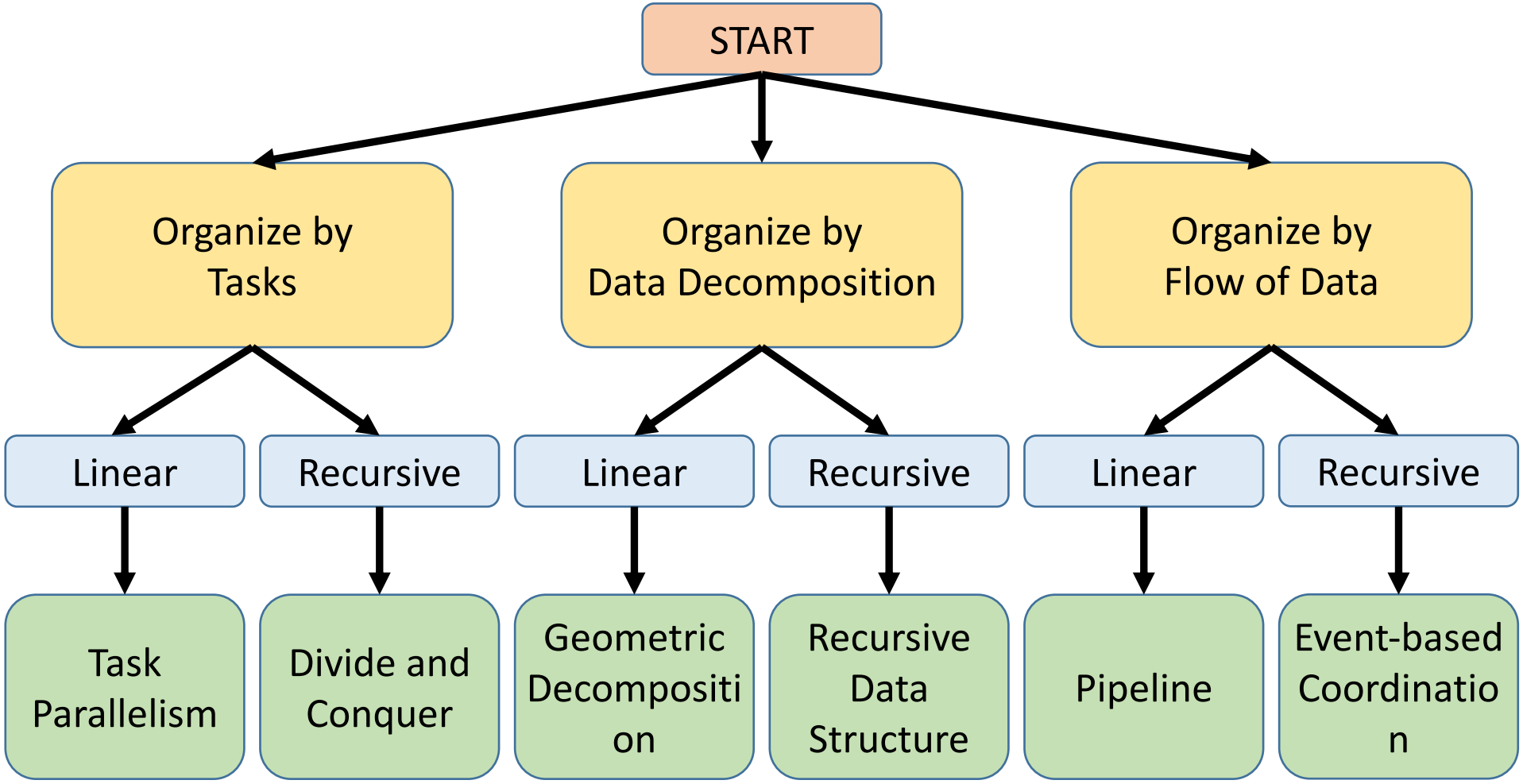
Guidelines for Data Decomposition

- Flexibility
  - Size and number of data chunks should support a wide range of executions
- Efficiency
  - Data chunks should generate considerable amounts of work (adequate grain), to minimize impact of communication and management
  - Data chunks should generate comparable amounts of work, for load balancing
- Simplicity
  - Complex data compositions can get difficult to manage and debug

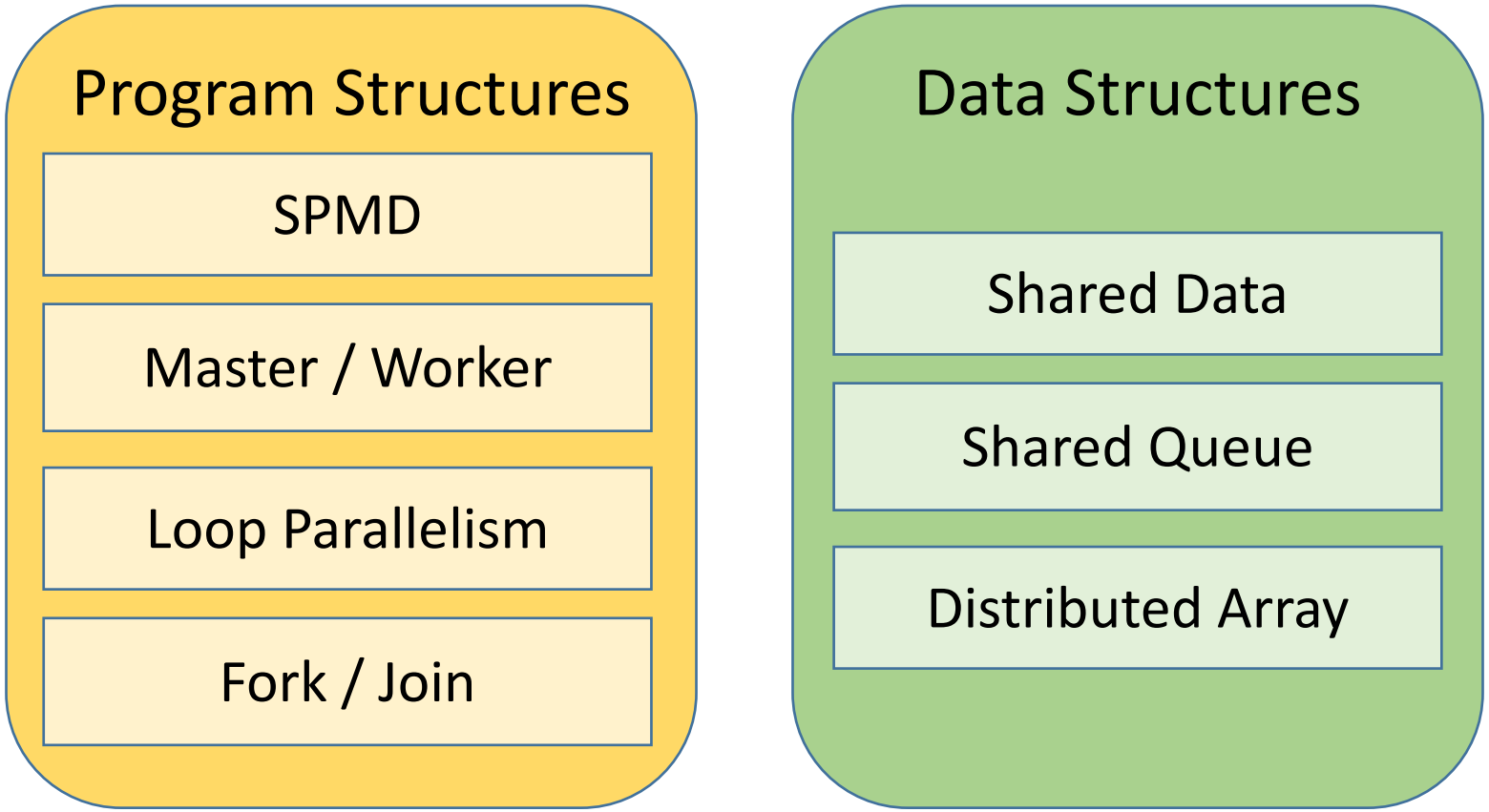
Common Data Decomposition

- Geometric data structures
  - Decomposition of n-dimensional arrays along rows, column, blocks
- Recursive data structures
  - Example: list, tree, graph

Algoritmic Structure Design Space



3. Implement the algorithm in a suitable progr. environment

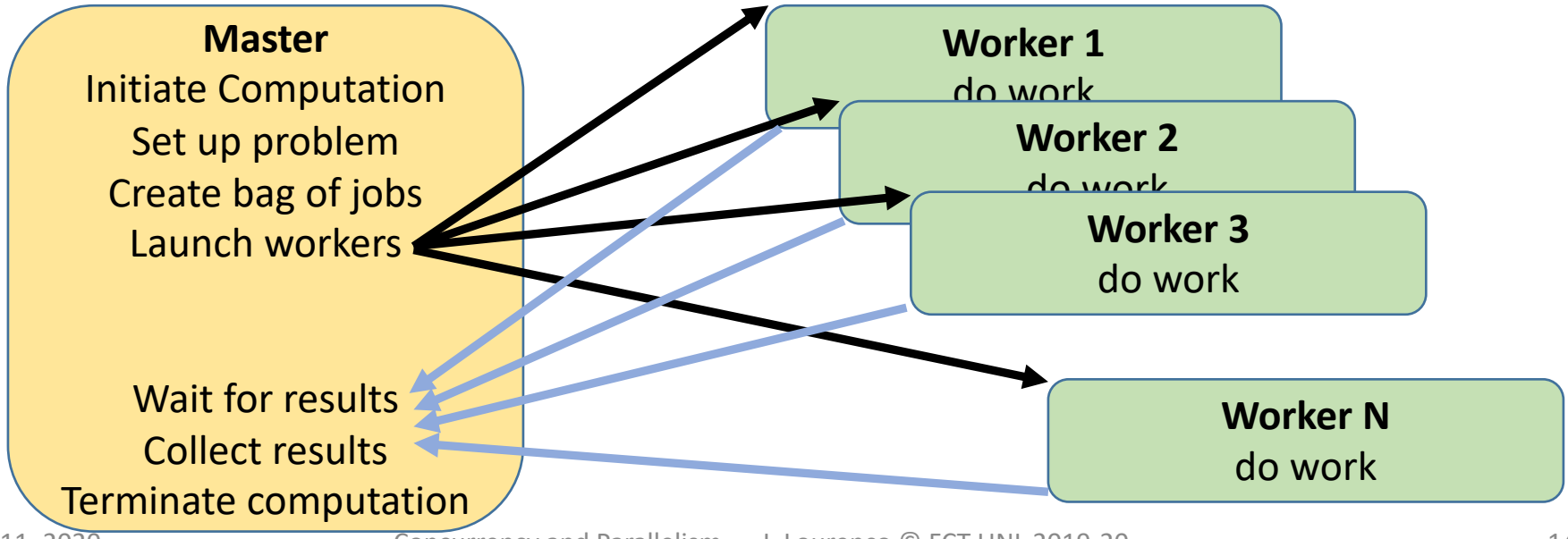


SPMD Pattern

- Single program, multiple data
- All tasks execute the same program in parallel, but each has its own set of data
  - Initialize
  - Obtain a unique identifier
  - Run the same program each processor
    - Operate on distributed data
  - Finalize
- CUDA

Master / Worker Pattern

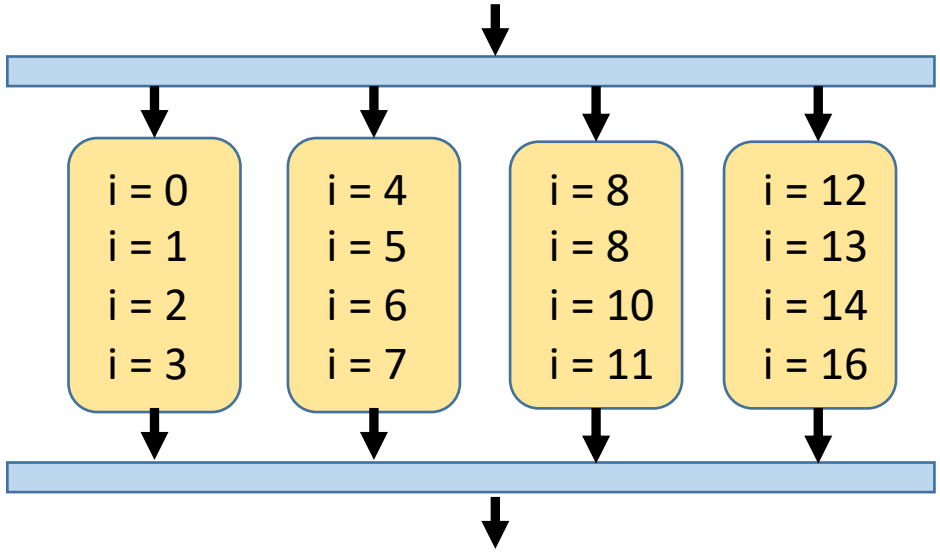
- A master process or thread set up a pool of worker processes of threads and a bag of tasks
- The workers execute concurrently, with each worker repeatedly removing a tasks from the bag of the tasks
- Embarrassingly parallel problems



Loop Parallelism Pattern

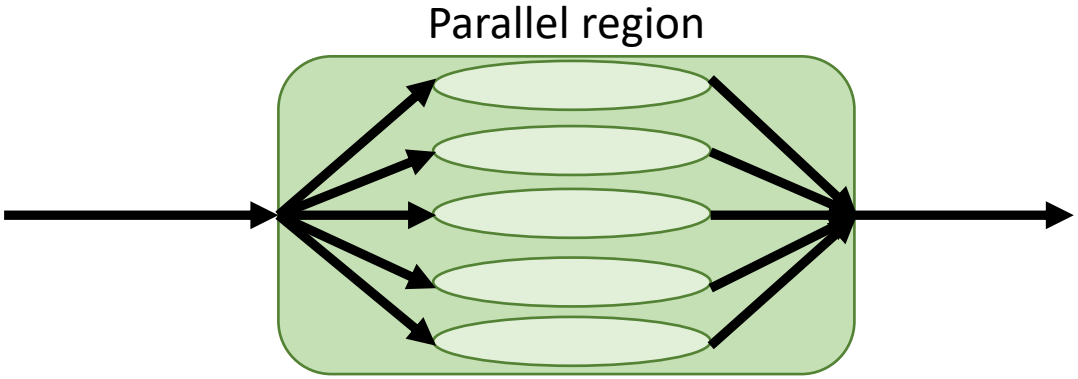
- Many programs are expressed using iterative constructs
  - Programming models like OpenMP provide directives to automatically assign loop iteration to execution units
  - Especially good when code cannot be massively restructured

```
#pragma omp parallel for
for (i = 0; i < 16; i++)
    c[i] = A[i]+B[I];
```



Fork / Join Pattern

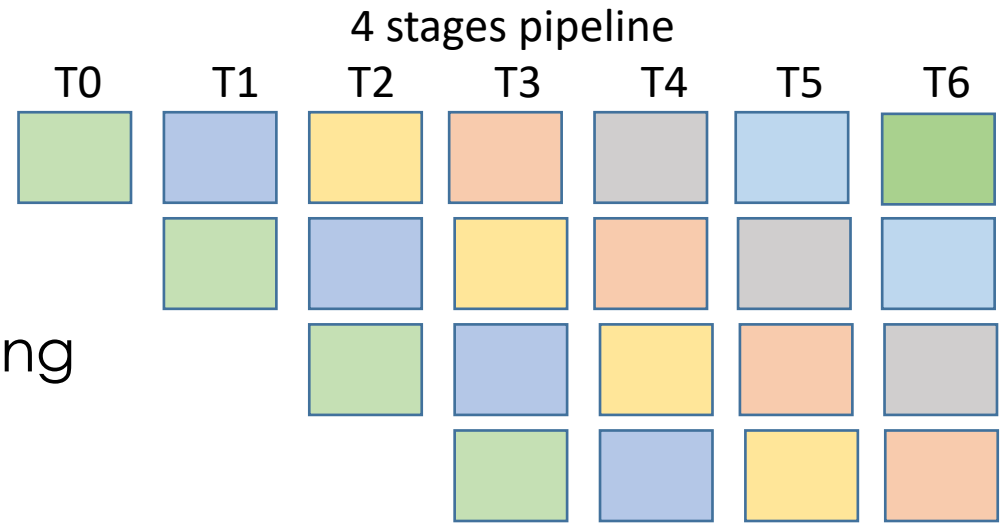
- A main task forks off some number of other tasks that then continue in parallel to accomplish some portion of the overall work
- Parent tasks creates new task (fork) then waits until all they complete (join) before continuing with the computation



Pipeline Pattern

- Tasks are applied in sequence to data

- Examples:
  - Instruction pipeline in modern CPUs
  - Algorithm level pipelining
  - Signal processing
  - Graphics
  - Shell programs
    - cat sampleFile | grep “word” | wc



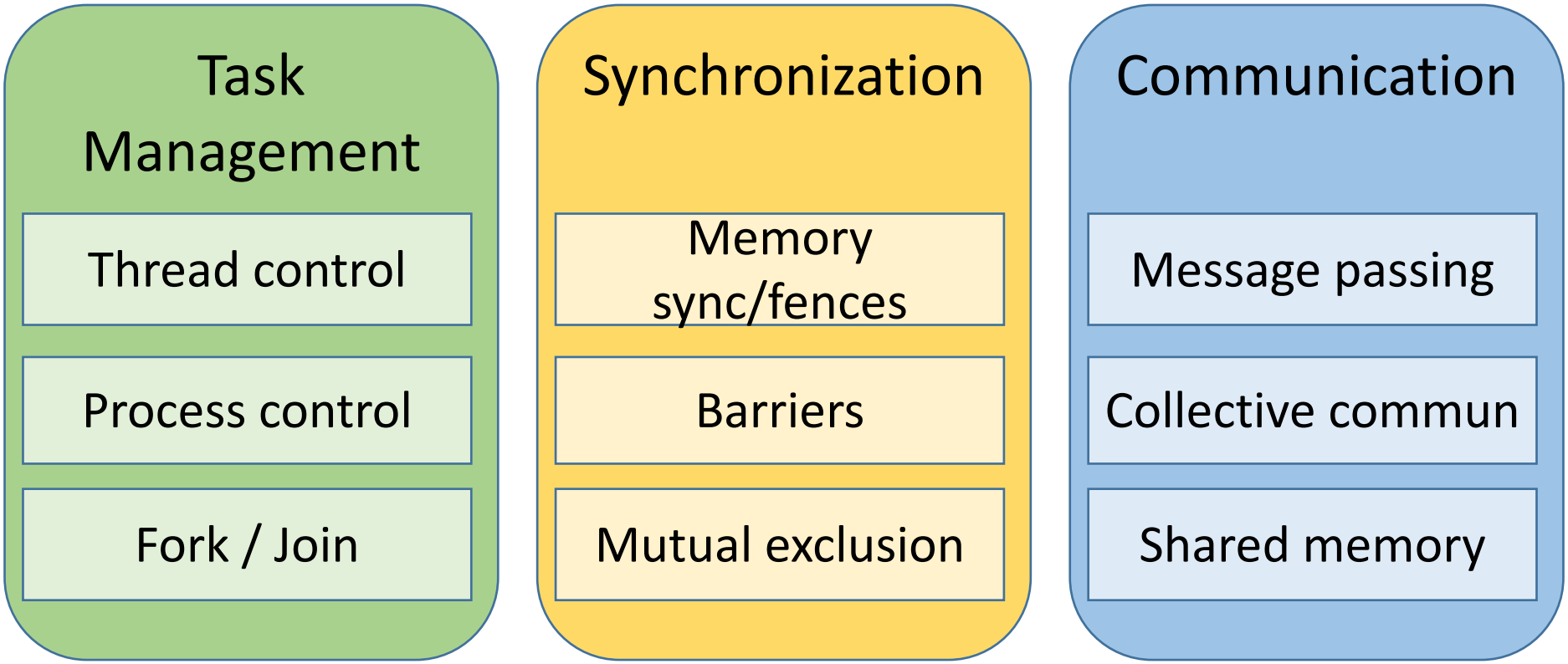
Choosing the Patterns

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

Choosing the Programming Environment

Prog. Env. / Pattern	OpenMP	MPI	CUDA
SPMD	☺ ☺ ☺	☺ ☺ ☺ ☺	☺ ☺ ☺ ☺ ☺
Loop parallel	☺ ☺ ☺ ☺	☺	
Master / Worker	☺ ☺	☺ ☺ ☺	
Fork / Join	☺ ☺ ☺		

3. The Implementations Mechanisms Design Space



The END