Parallel Algorithms

Tomàs Margalef
Departamento de Arquitectura de Computadores y Sistemas Operativos
Universidad Autónoma de Barcelona
tomas.margalef@uab.es

Universitat
Autònoma
de Barcelona

Contents

- **Parallel Programming: Introduction**
- **Parallel application design:**
  - Parallel programming paradigms
  - Parallel algorithms

Contents

- **Parallel Programming: Introduction**
- Parallel application design:
  - Parallel programming paradigms
  - Parallel algorithms

Introduction

- Performance is the main reason of parallel/distributed processing.
- When a user develops a parallel application, he/she expects reaching certain performance improvements.
- These improvements allow reducing execution time, increasing the problem size, maximizing the throughput, etc.

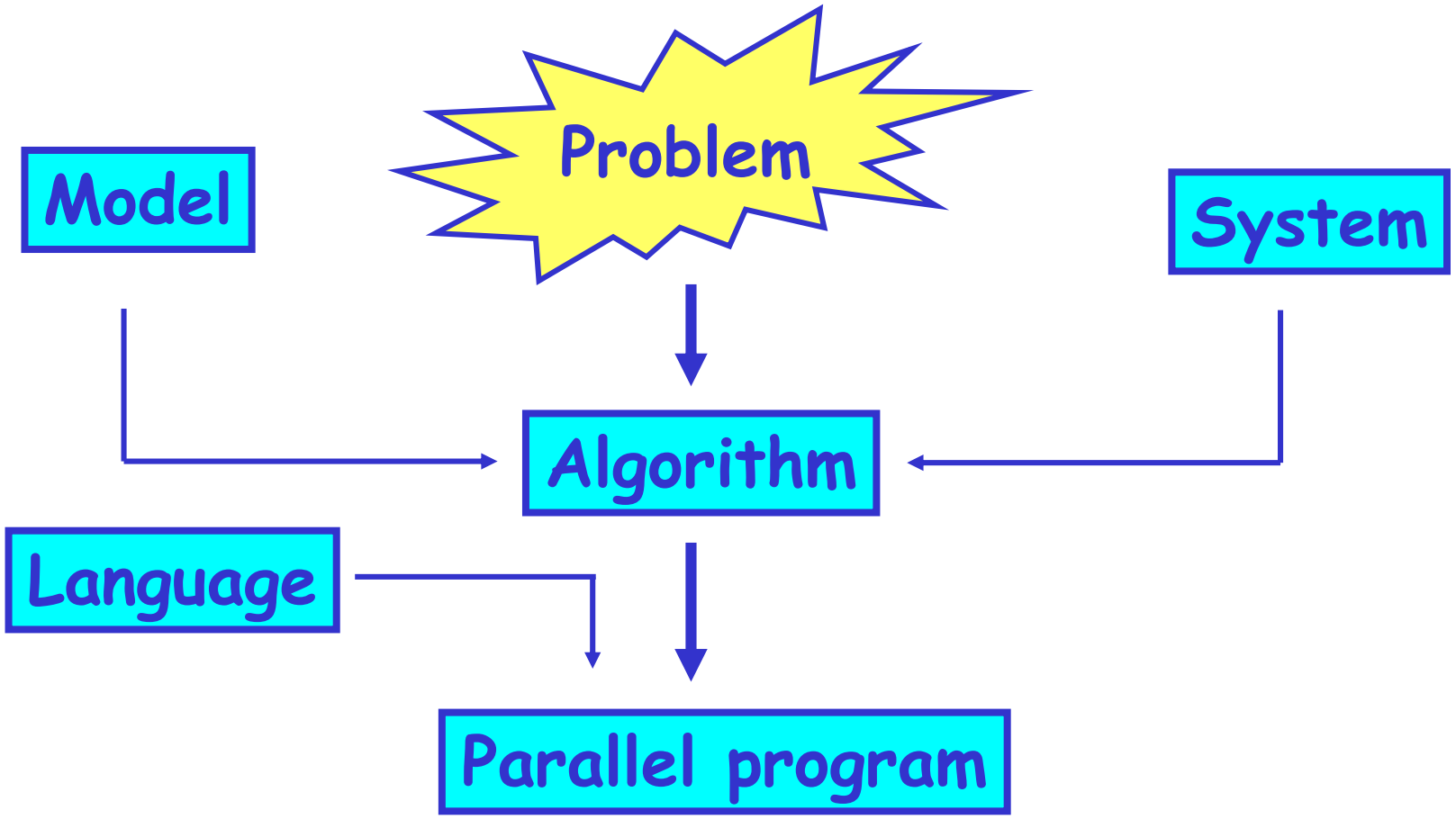
Introduction

- Parallel Application design is strongly connected to the system where the application will run and to the programming model used.
- Application, system and programming model are closely related.

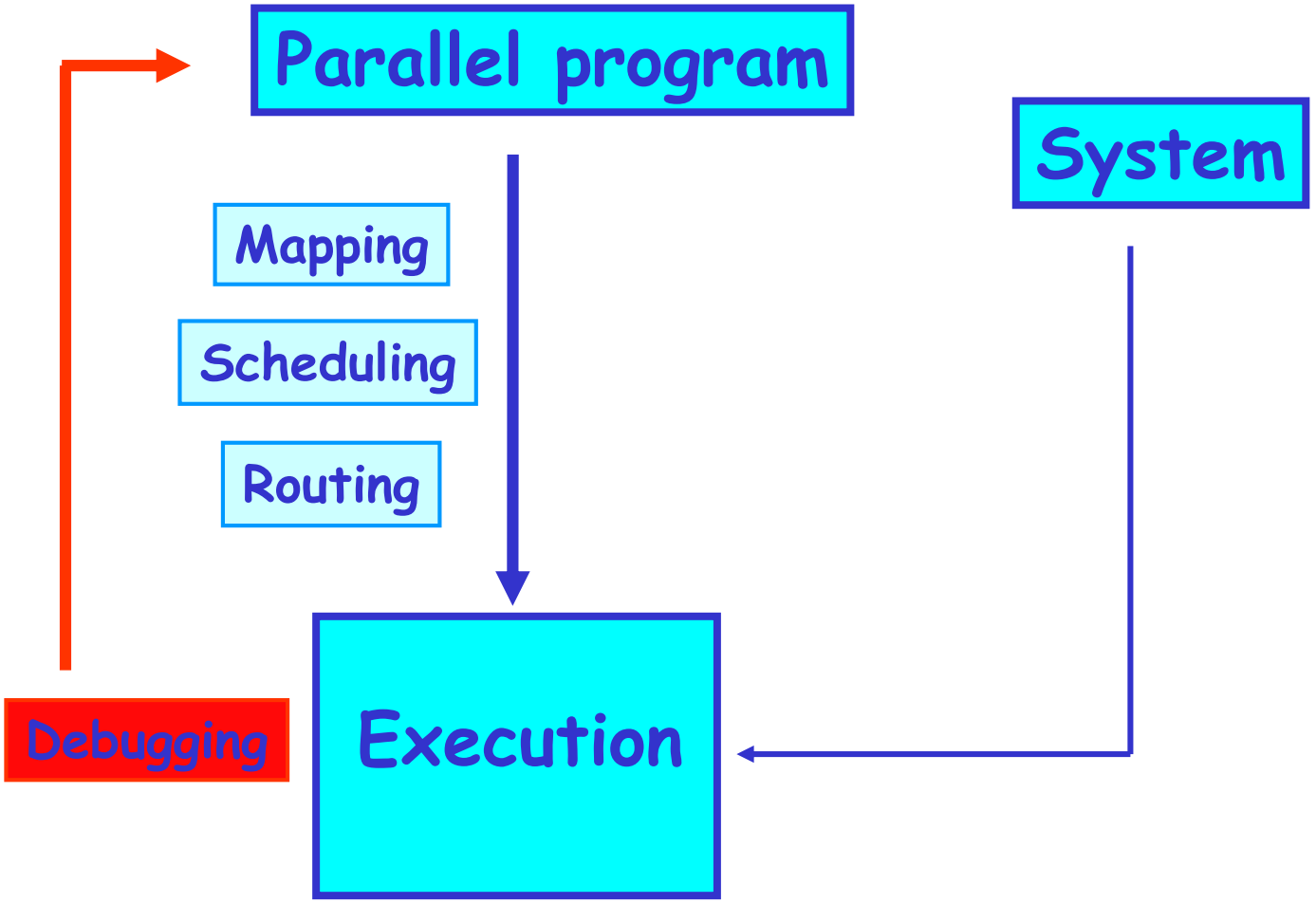
Parallel programming models

- **Programming models:**
  - Automatic parallelism extraction
  - Message passing
  - Shared memory: "Threads"

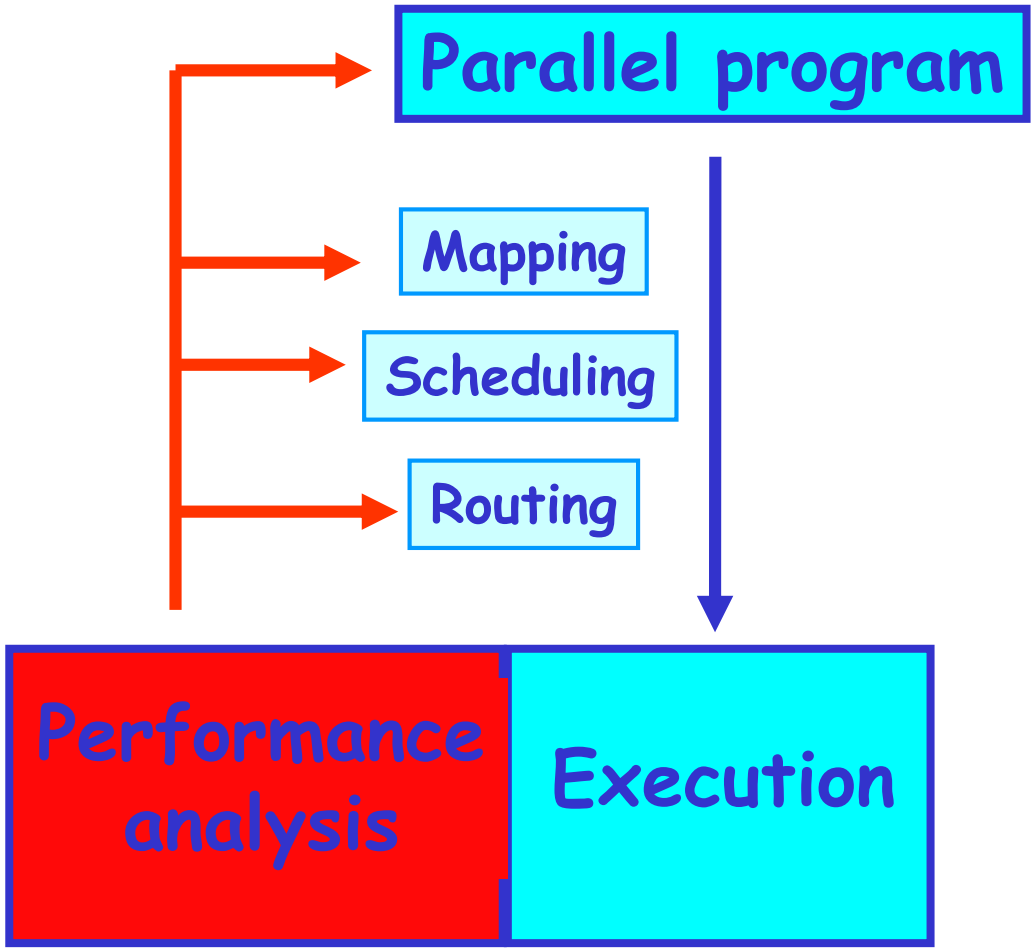
Parallel programming life cycle



Parallel programming life cycle



Parallel programming life cycle



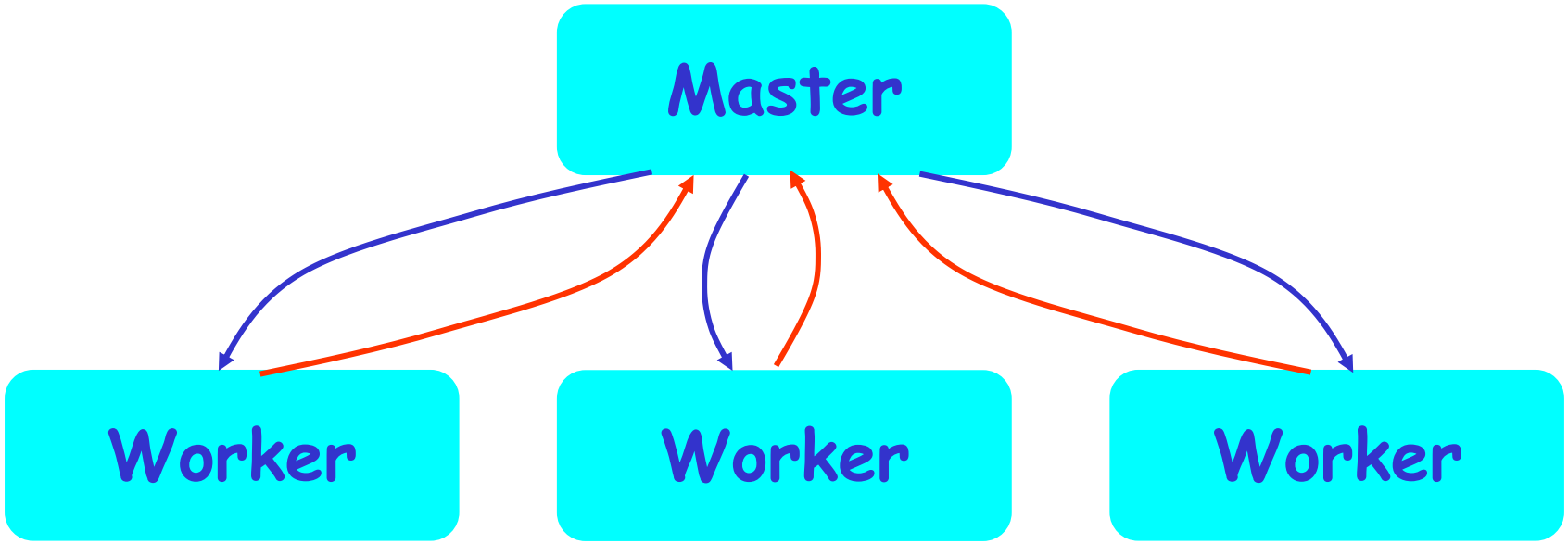
Contents

- Parallel Programming: Introduction
- **Parallel application design:**
  - Parallel programming paradigms
  - Parallel algorithms

Programming paradigms

- There are many algorithms in the literature that can be used to solve many different problems.
- Most algorithms have been designed for a particular system architecture.
- However, there are certain basic programming paradigms that can be used to solve many problems.

Master/Worker



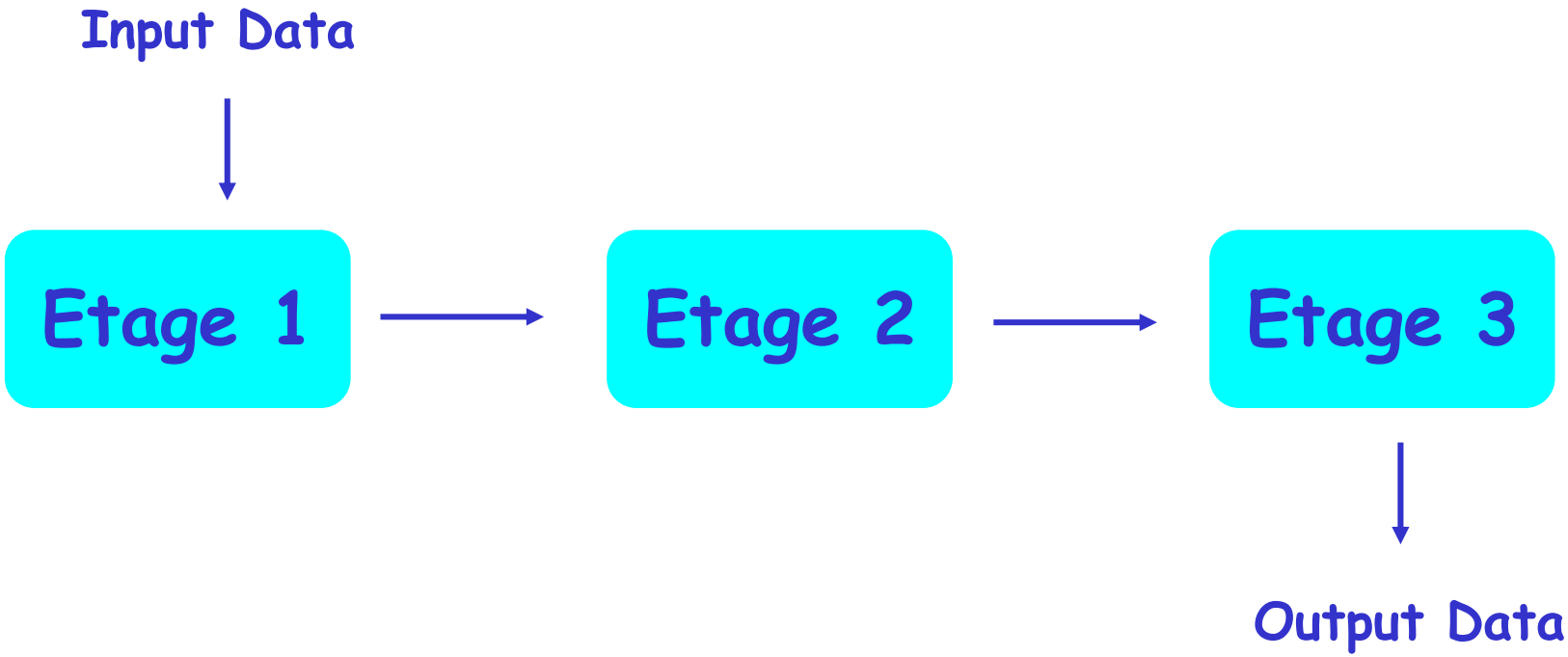
Master/Worker

- **Iteration dependence**
  - Dependence between iterations: Master process requires the results of "all" the workers" to generate the next set of data.
  - Independent data input: Data continuously arrives to the Master process and it does not require the results of the previous set of data.

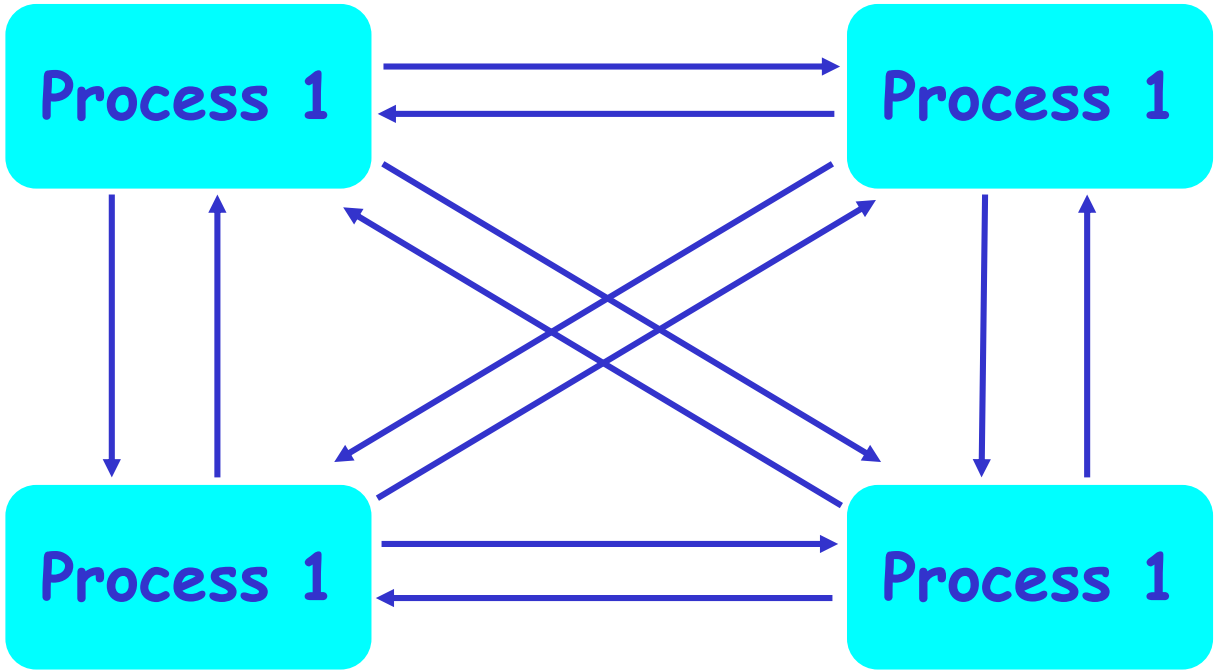
Master/Worker

- **Data partitioning and distribution**
  - Distribute all data among the Workers: When Master has prepared all the data, it sends them to the Worker processes according to certain strategy.
  - Distribute data on-demand: Master process distributes a subset of data to the Workers and then these workers ask for more data when they finish the previous chunk.

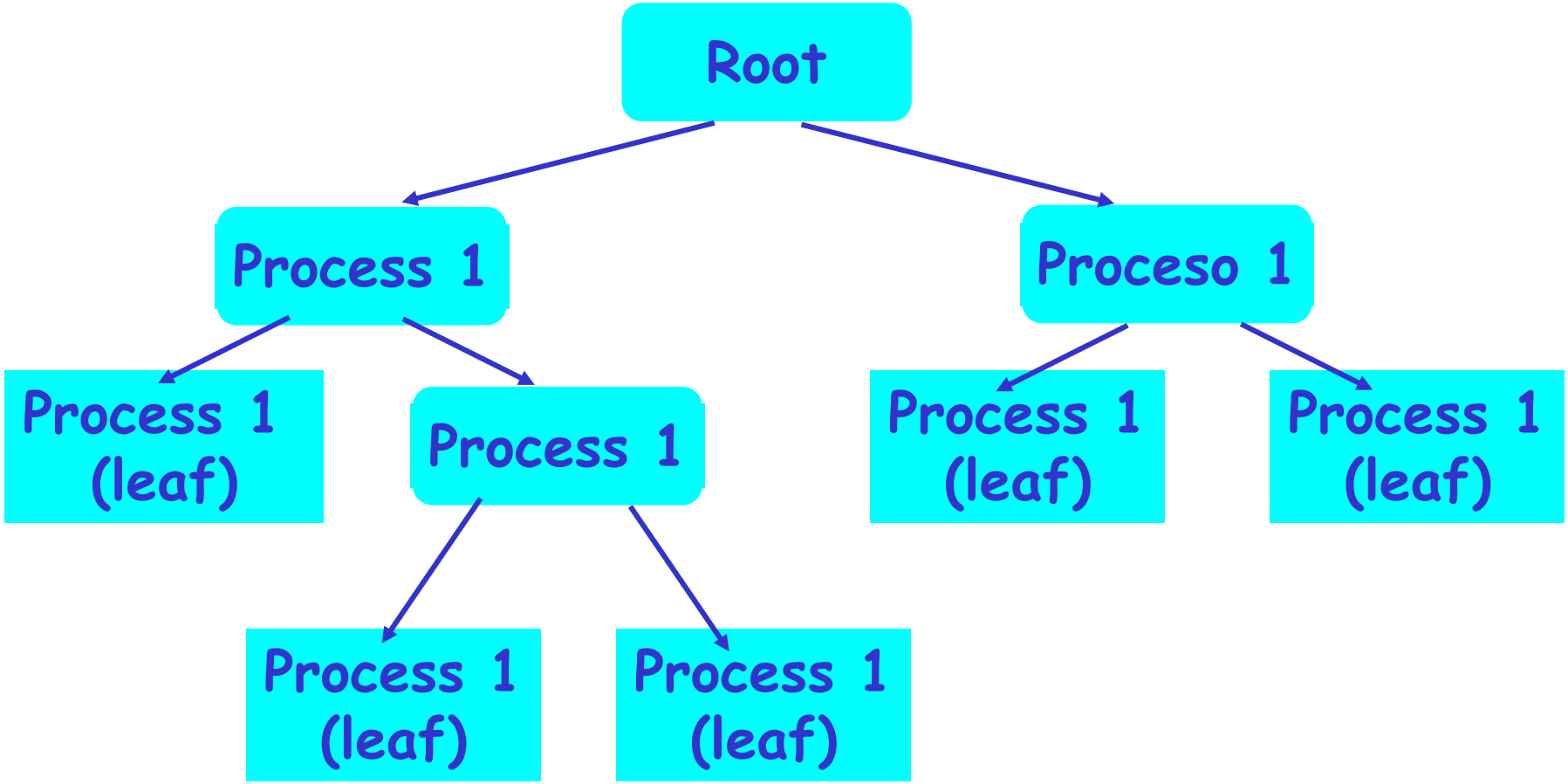
Pipeline



Single Program Multiple Data SPMD



Divide and Conquer



Divide and Conquer

- **All processes, but the root, are exactly equal. Each process:**
  - Receive a subset of data.
  - If it can process them, it process them.
  - If it cannot process them, it creates a certain number of child processes and distribute the subset of data.

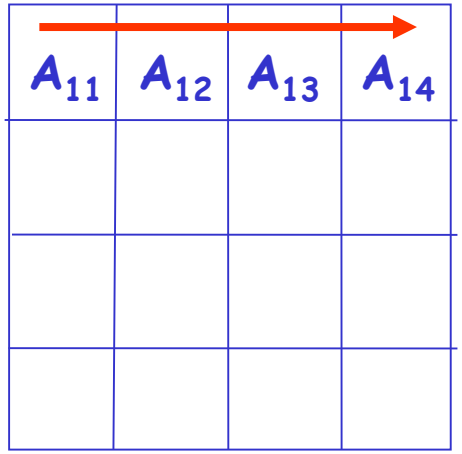
Programming paradigms

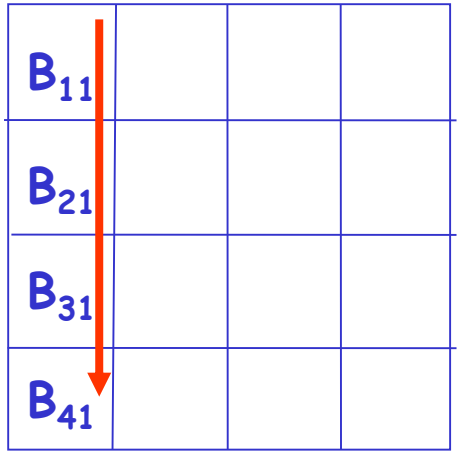
- Usually, these paradigms simplify the programmers task.
- The programmer just develops one copy of the code and, when it is executed, multiple instances are generated.
- Communication patterns are always the same for each paradigm.

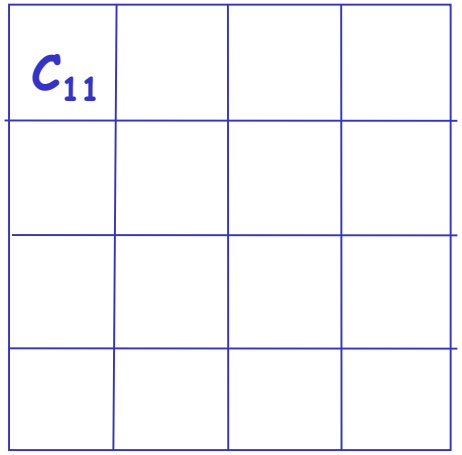
Contents

- Parallel Programming: Introduction
- Parallel application design:
  - Parallel programming paradigms
  - Parallel algorithms

Matrix multiplication







$$C_{ij} = \sum_{\forall k} A_{ik} * B_{kj}$$

Matrix multiplication

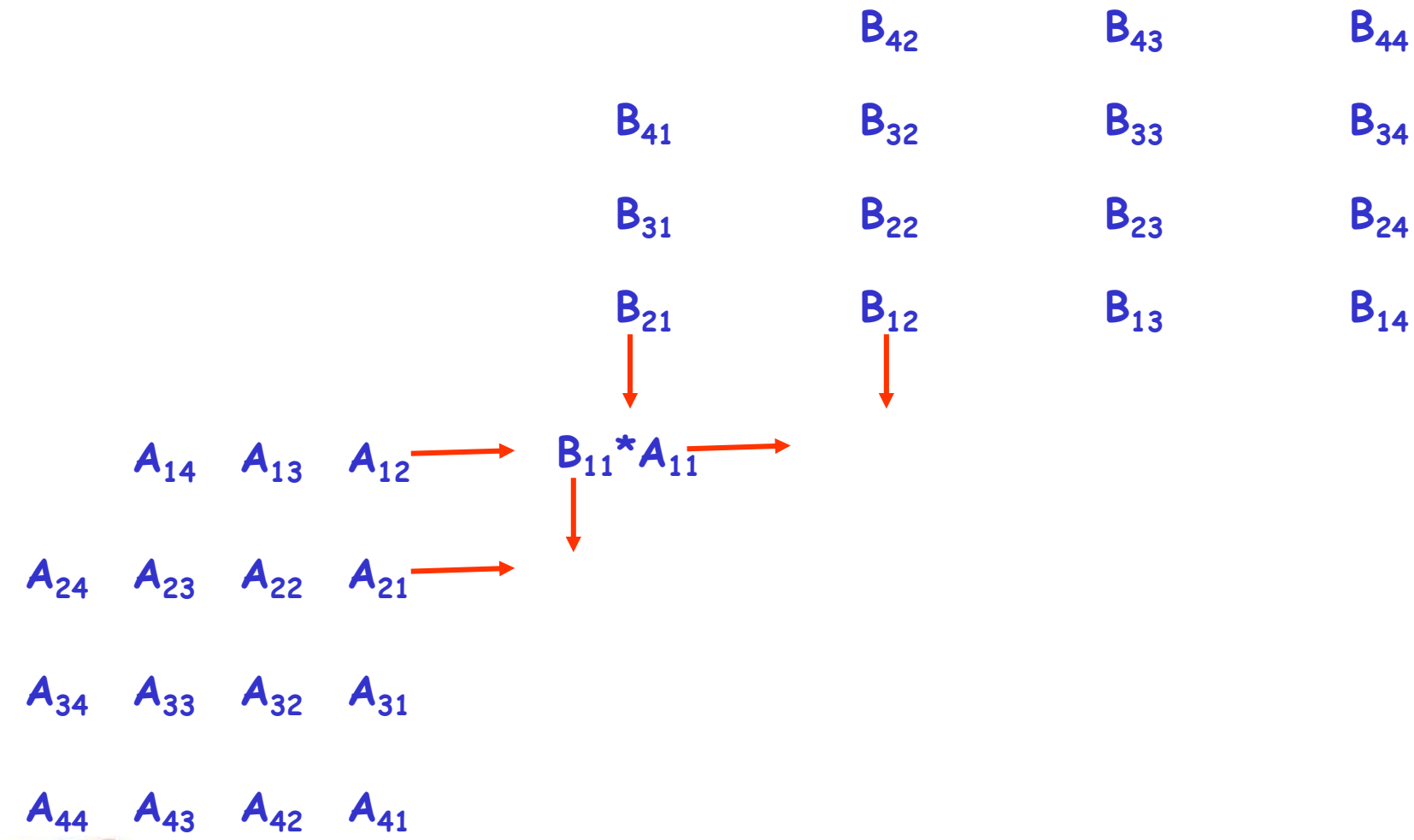
- There is no dependence among the calculations required to determine the values of the resulting matrix
- All C_{ij} terms can be calculated in parallel.
- The set of multiplications that must be done to evaluate one term of the resulting matrix are also independent. Therefore, each C_{ij} calculation can be parallelised.

Matrix multiplication

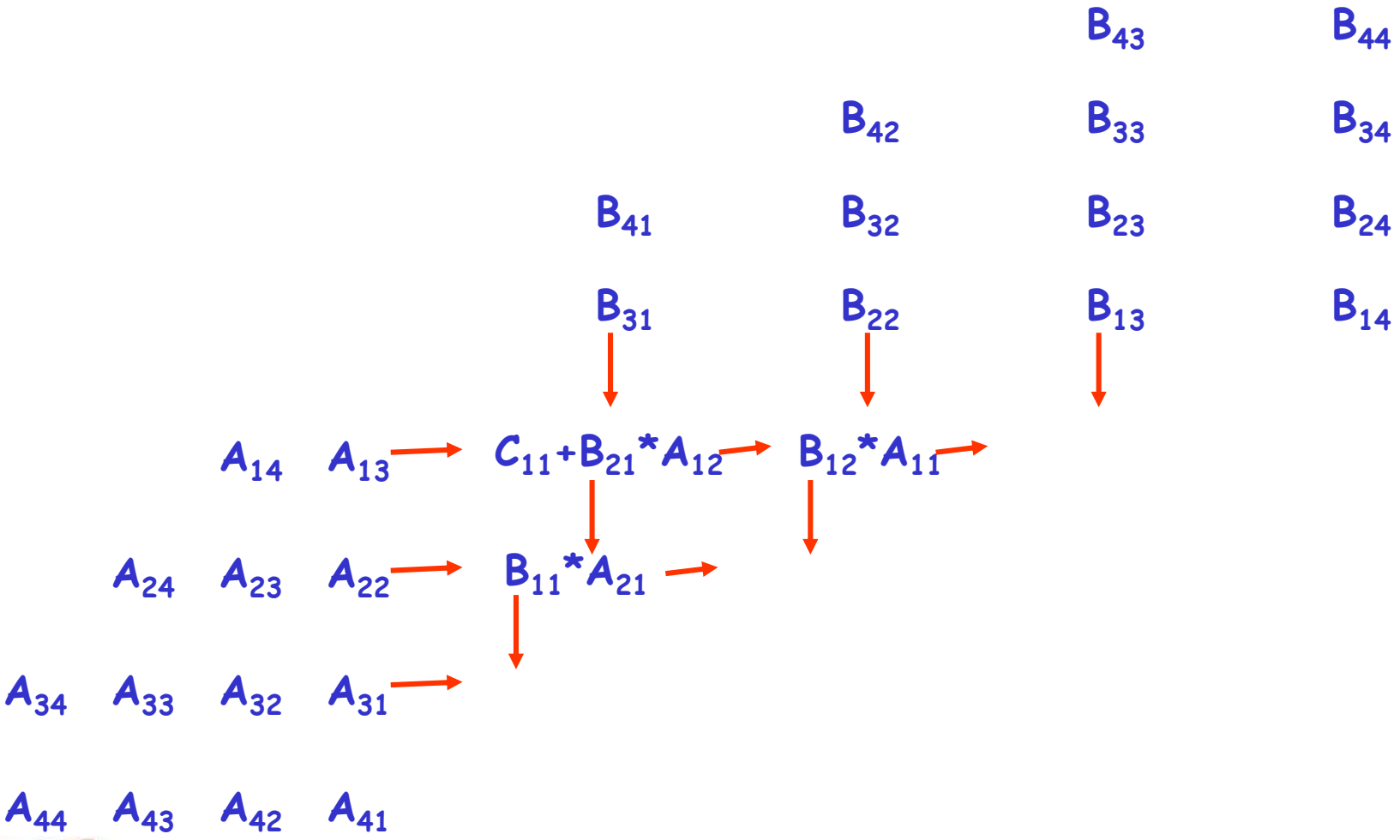
$$\begin{matrix} B_{41} & B_{42} & B_{43} & B_{44} \\ B_{31} & B_{32} & B_{33} & B_{34} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ B_{11} & B_{12} & B_{13} & B_{14} \end{matrix}$$

$$\begin{matrix} A_{14} & A_{13} & A_{12} & A_{11} \\ A_{24} & A_{23} & A_{22} & A_{21} \\ A_{34} & A_{33} & A_{32} & A_{31} \\ A_{44} & A_{43} & A_{42} & A_{41} \end{matrix}$$

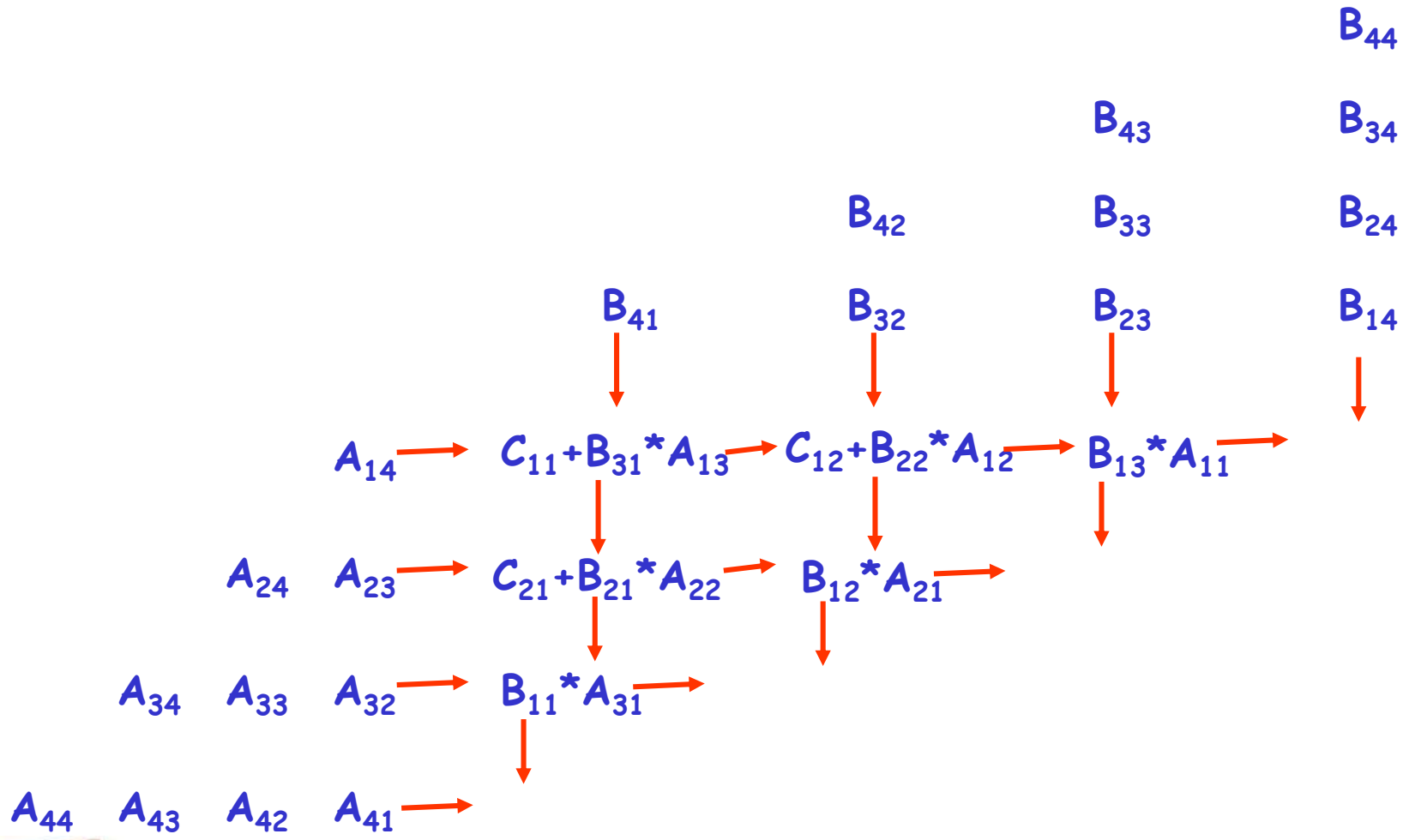
Matrix multiplication



Matrix multiplication



Matrix multiplication



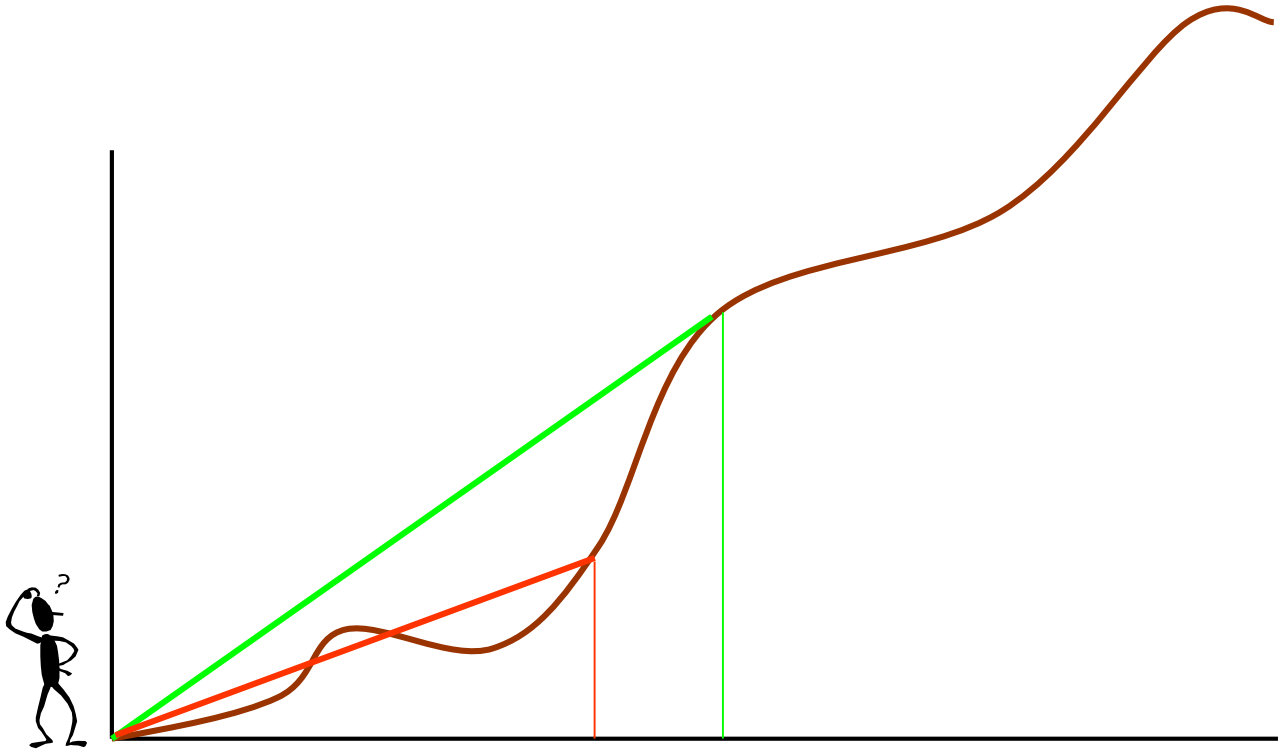
Thinkings about the algorithm

- The algorithm works properly on a system with a mesh interconnectin network.
- On other systems (PC clusters on ethernet) the performance degrades substancialy.

Other considerations

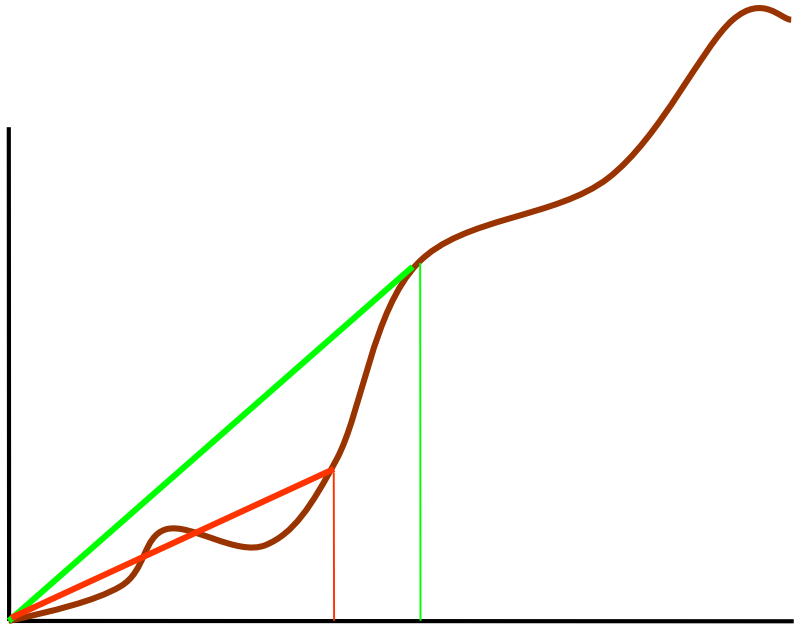
- Besides the topology of the interconnection network, the computation and communication capabilities will also affect the performance of the algorithm.
- It is necessary to determine the granularity of the tasks/data to balance computation and communication.

Which points are visible?



Which points are visible?

$$\alpha_i = \operatorname{arctg} \frac{y_i}{x_i}$$



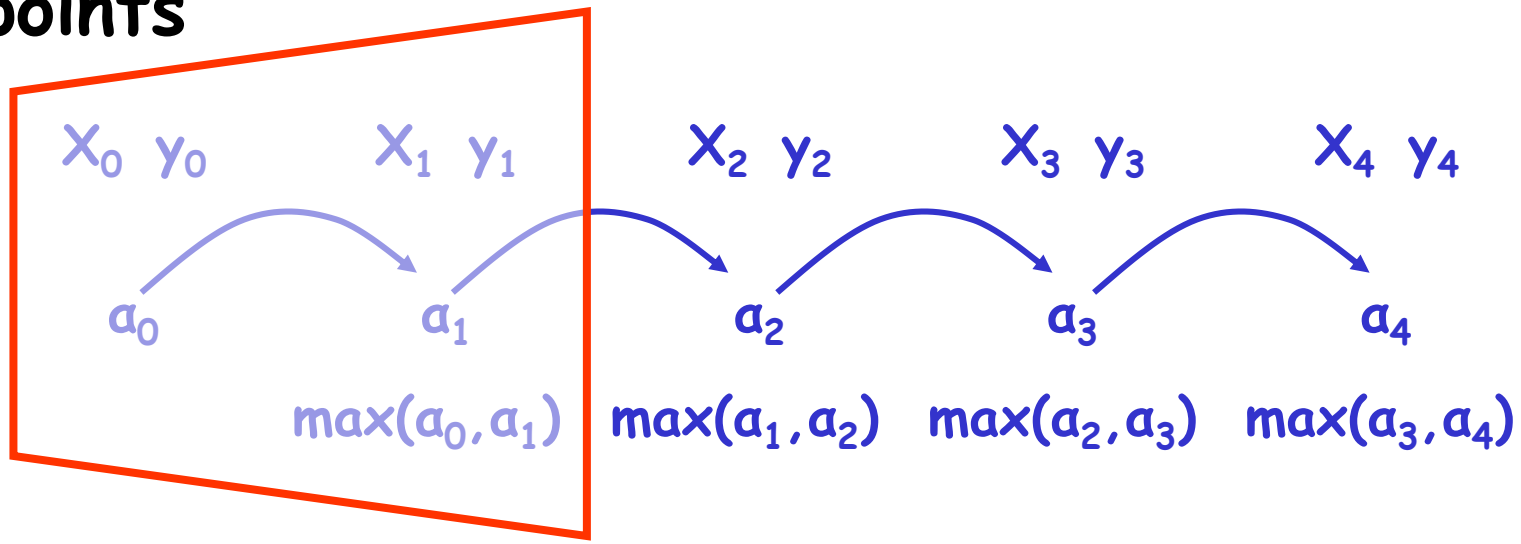
Visibility condition

$$\alpha_i \geq \alpha_k \forall k < i$$

$$max(\alpha_k)_{k=0}^{i} = \alpha_i$$

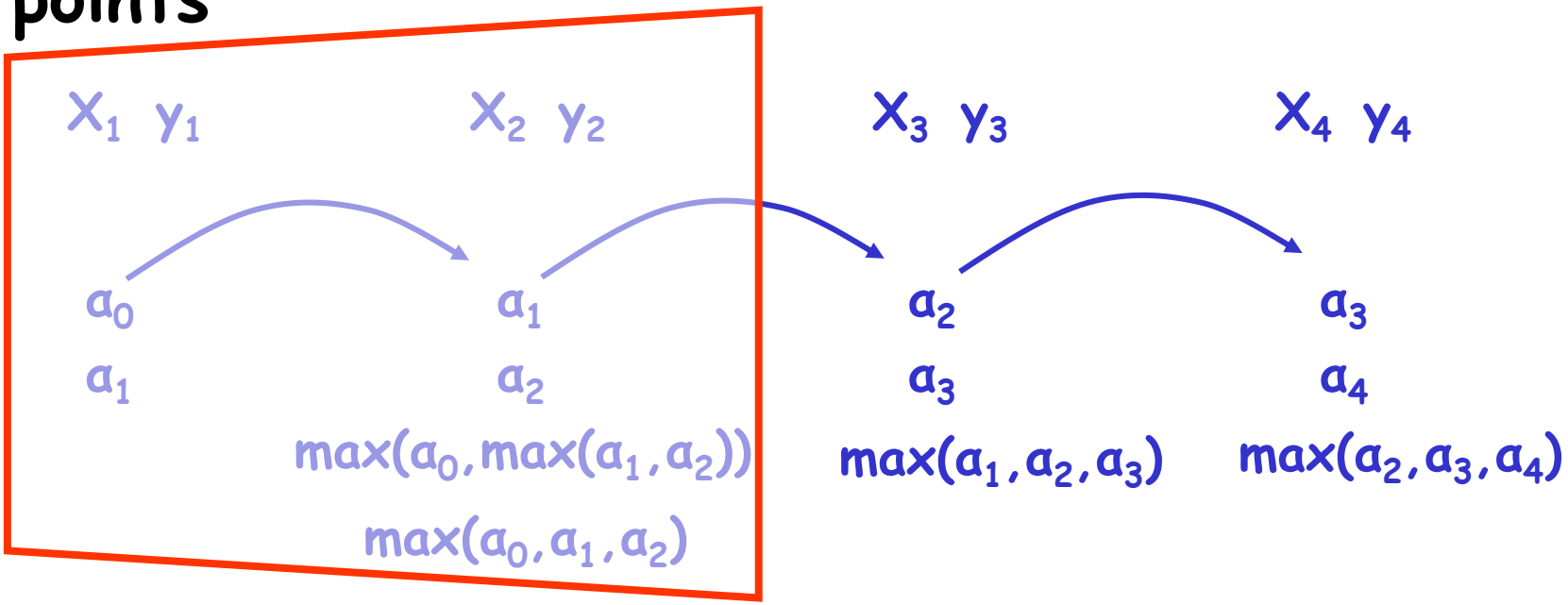
A parallel algorithm

- To determine if one point is visible, it is necessary to know the angle of all previous points



A parallel algorithm

- To determine if one point is visible, it is necessary to know the angle of all previous points



A parallel algorithm

- To determine if one point is visible, it is necessary to know the angle of all previous points

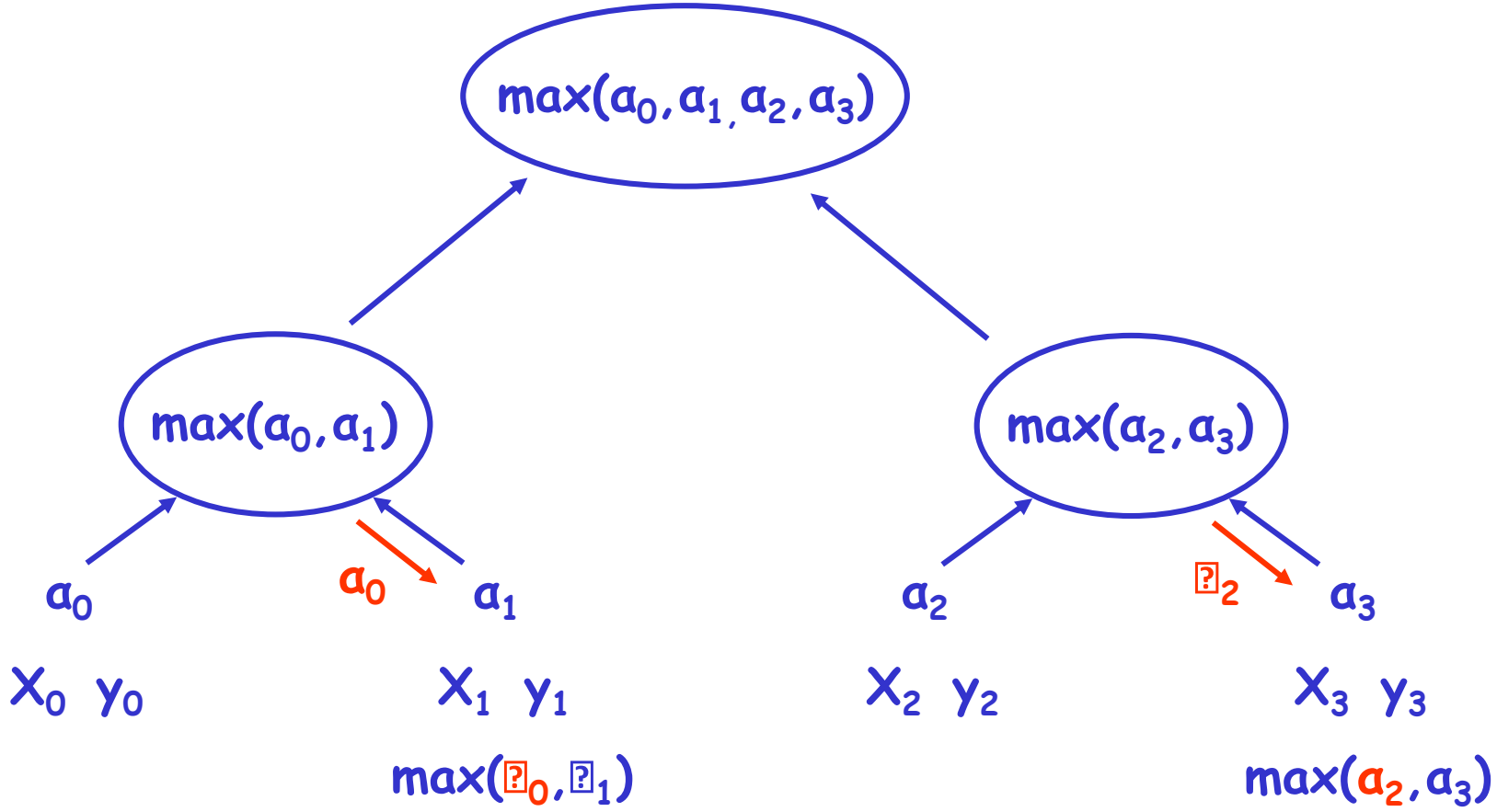
$x_i \ y_i$

$\max(a_0, a_1, \ldots, a_i)$

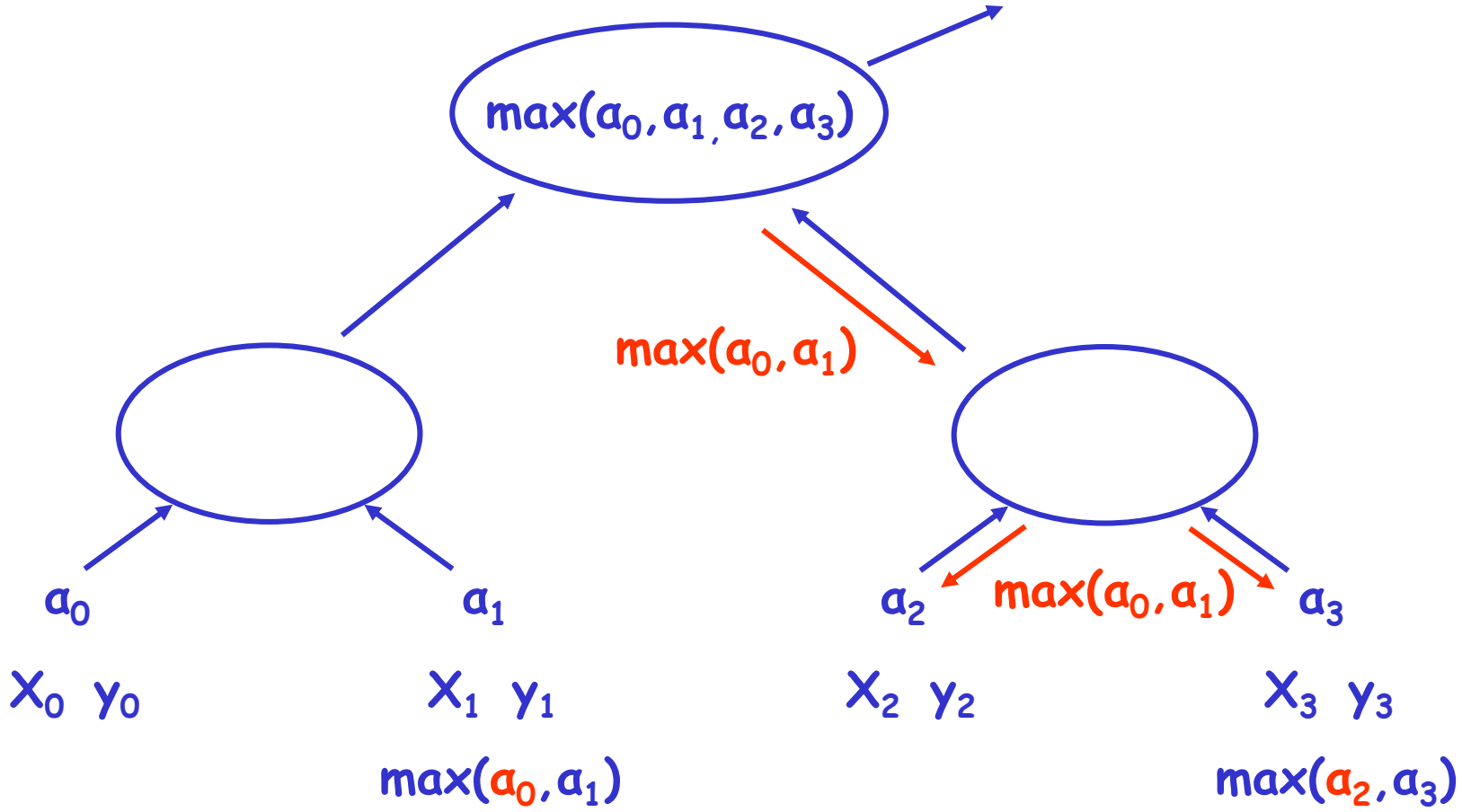
- If there *n* points, we need (*n-1*) steps.

O(n)

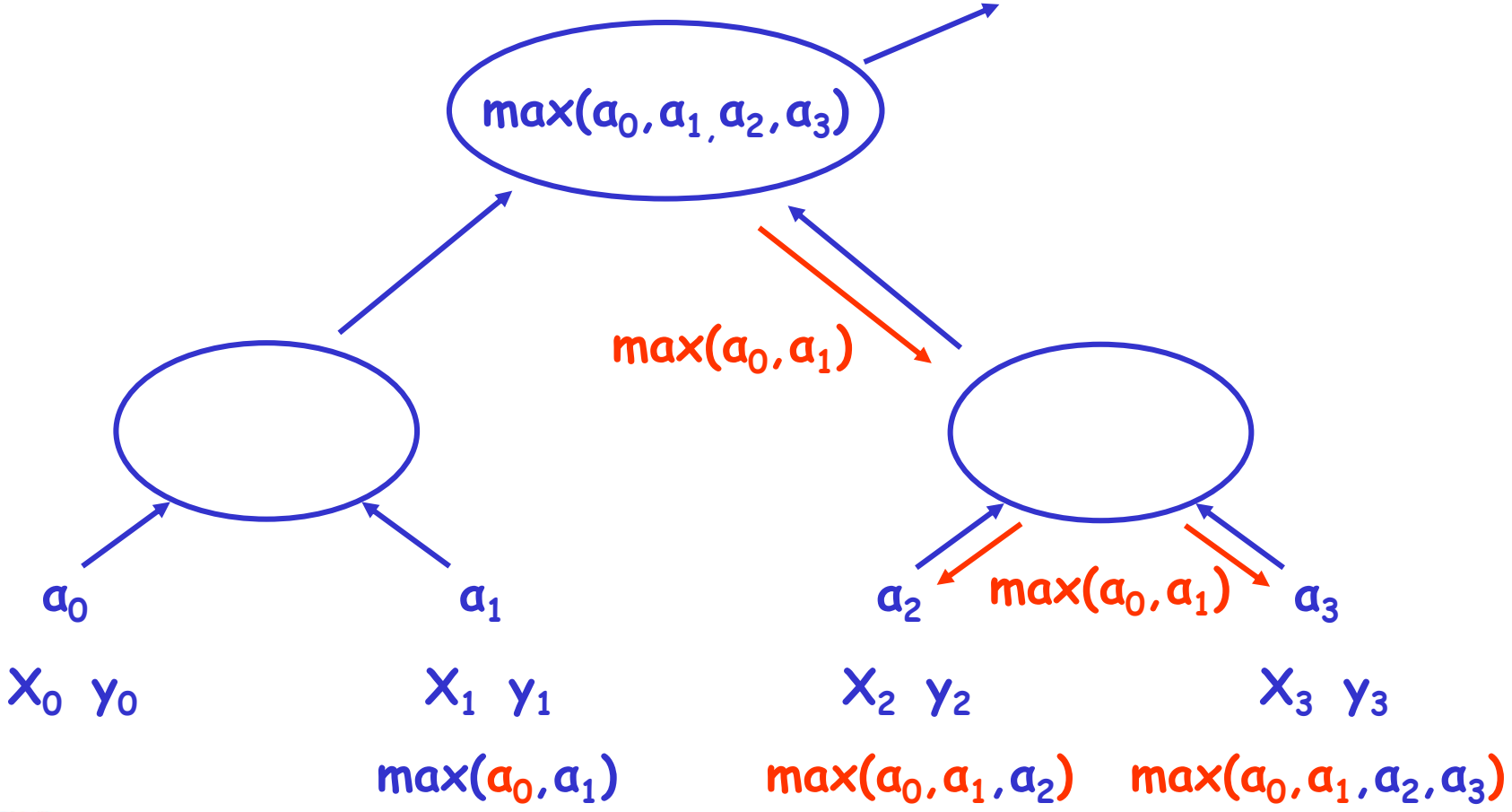
Parallel Prefix algorithm



Parallel Prefix algorithm



Parallel Prefix algorithm



Parallel Prefix algorithm

- The complexity of this algorithm is logarithmic.

$$O(\log_2 n)$$

- When the number of points increases, the improvement in the execution time is significant.