



Computer Engineering

Unit 3. Parallel computation

Concepts and motivation

#1 TOP500 (6/2018) - Sequoia

- Manufacturer: IBM
- Cores: 1572864
- OS: Linux
- Interconnecting network: ad-hoc
- Maximum performance: 16324,8 TF
- Peak performance: 20132,7 TF
- Power: 7890 kW
- Location: Lawrence Livermore National Laboratory (LLNL)
- <https://computing.llnl.gov/>
- Detailed specification:
https://computing.llnl.gov/?set=resources&page=OCF_resources#sequoia

Concepts and motivation

#5 TOP500 (6/2018) - Tianhe 2A

- Manufacturer: NUDT (National University of Defense Technology)
- Cores: 4,981,760
- OS: Linux
- Interconnecting network: proprietary
- Maximum performance: 61,444 TF
- Peak performance: 100,678 TF
- Power: 18,482 kW
- Location: National Supercomputing Center in Guangzhou
- <http://www.nscg-tj.gov.cn/en/>
- Detailed specification: <https://www.top500.org/site/50365>

Concepts and motivation

#6 TOP500 (6/2018) - Barcelona Supercomputing Center

- Manufacturer: Bull
- Núcleos: 153,216 (Xeon Platinum 8160 24C 2.1GHz)
- OS: Linux
- Interconnecting network: Intel Omni-Path
- Maximum performance: 6,470,2 TF
- Peak performance: 10,297 TF
- Power: 1,632 kW
- Location: Barcelona Supercomputing Center
- <http://www.bsc.es/>
- Detailed specification:
<https://www.bsc.es/discover-bsc/the-centre/marenostrum>

Concepts and motivation

Where is used supercomputation?

- Defense
 - Terrorism
 - Weapons program
 - Spatial program
- Engineering/Academic/Research
 - New energies prospection and simulation
 - Nanotechnology
 - Biological modeling
 - Deep learning
 - Genome research
 - Protein research
 - Astrophysical simulations
 - Weather forecasting
 - Drug design
 - Geological modeling
 - ...

Concepts and motivation

Parallel processing versus distributed processing:

Parallel processing:

- How and when to split an application into independent computing units to **speed-up** the execution? How are these units executed and how the application can be scalable in terms of complexity and performance?

Distributed processing:

- How to execute at the same time diverse applications using the same resources → distribution of time and resources (processor, memories, disks...)

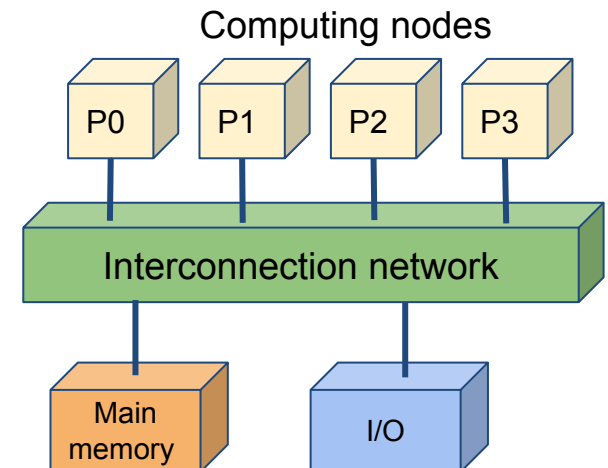
Concepts and motivation

Parallel computer classification (regarding their memory system)

- **Multiprocessor:** Nodes sharing the same addressing space (programmers needn't know where data are located)
- **Multicomputer:** Each node has its own addressing space (programmers problem → in which node are my data)

Parts of a parallel computer:

- Computing nodes
- Memory system
- Communication system
- Input/Output system



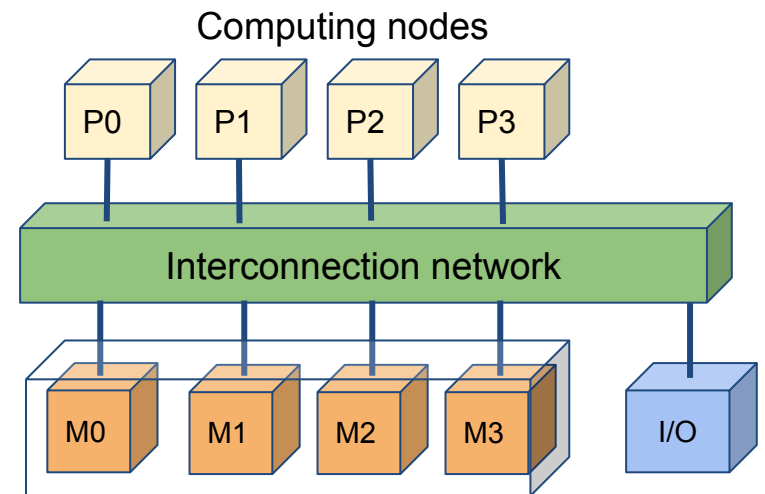
Computer Engineering

Unit 3: Parallel computation

Concepts and motivation

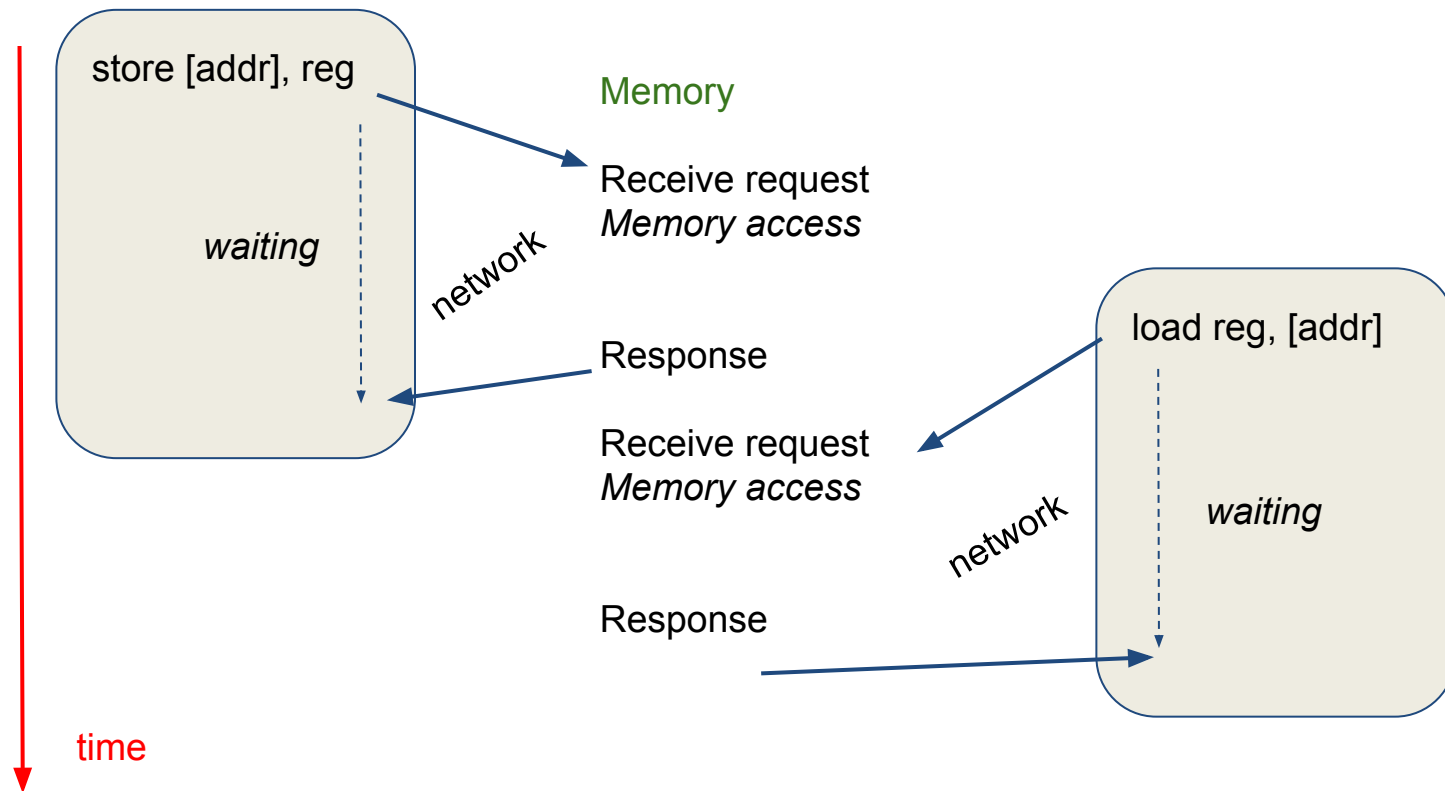
Multiprocessors: NUMA & SMP/UMA (Symmetric Shared Memory or Uniform Memory Access Multiprocessors)

- Higher latency
- Less scalable → Why?
- Communication by shared variables (no data replication)
- Synchronization primitives are needed.
- No need to distribute code and data → Why?
- Programming is usually friendly



Concepts and motivation

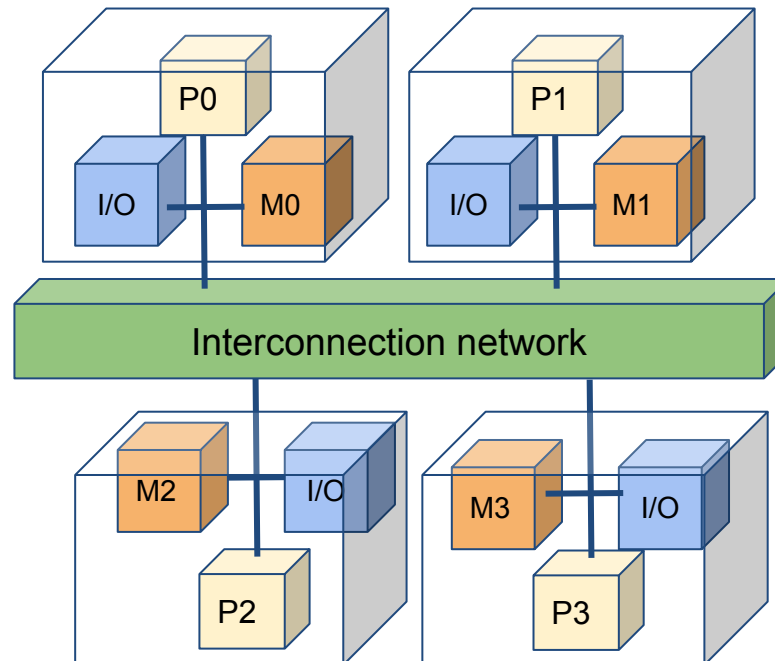
Communication in a multiprocessor



Concepts and motivation

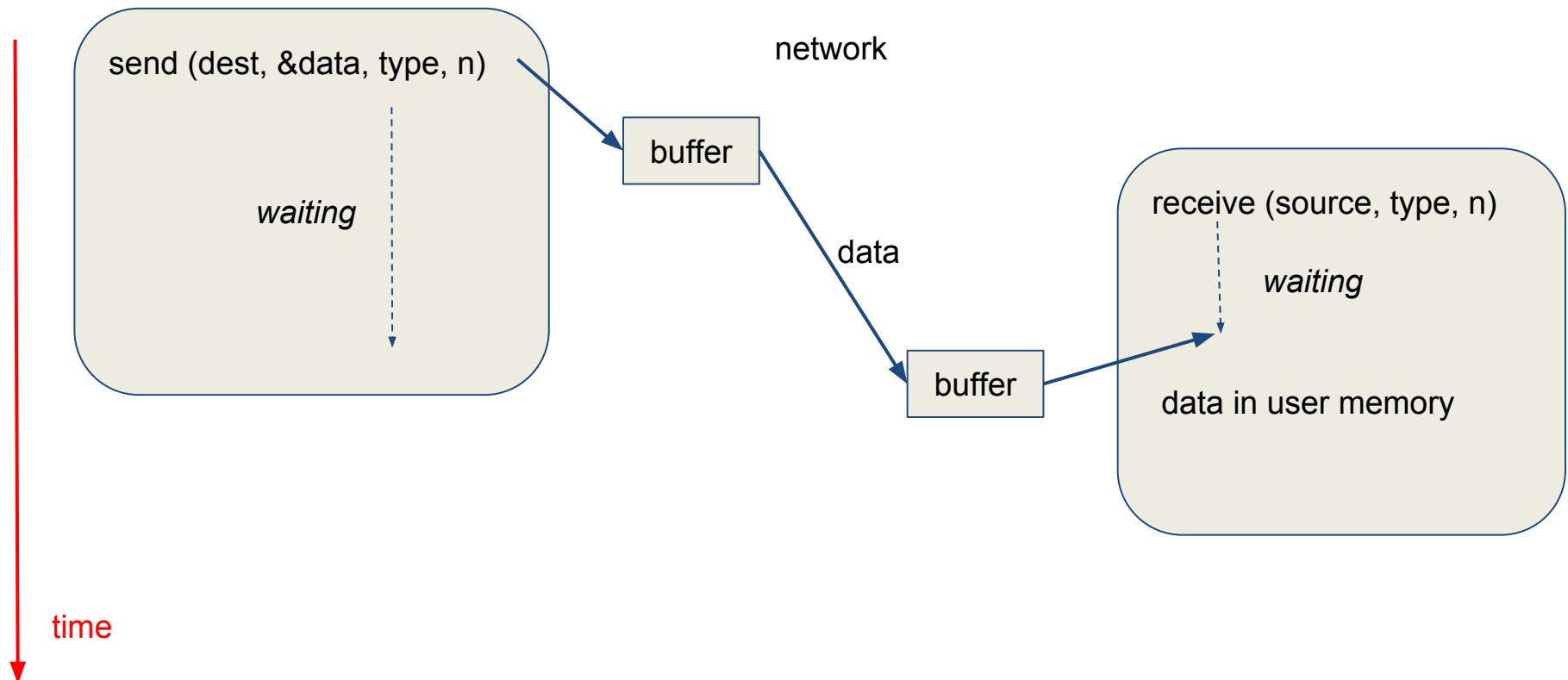
Multicomputer

- Lower latency
- Higher scalability
- Communication by message passing (duplicated data)
- Synchronization by means of communication mechanisms
- Workload (code and data) is distributed among processors
- Programming is a bit more complicated



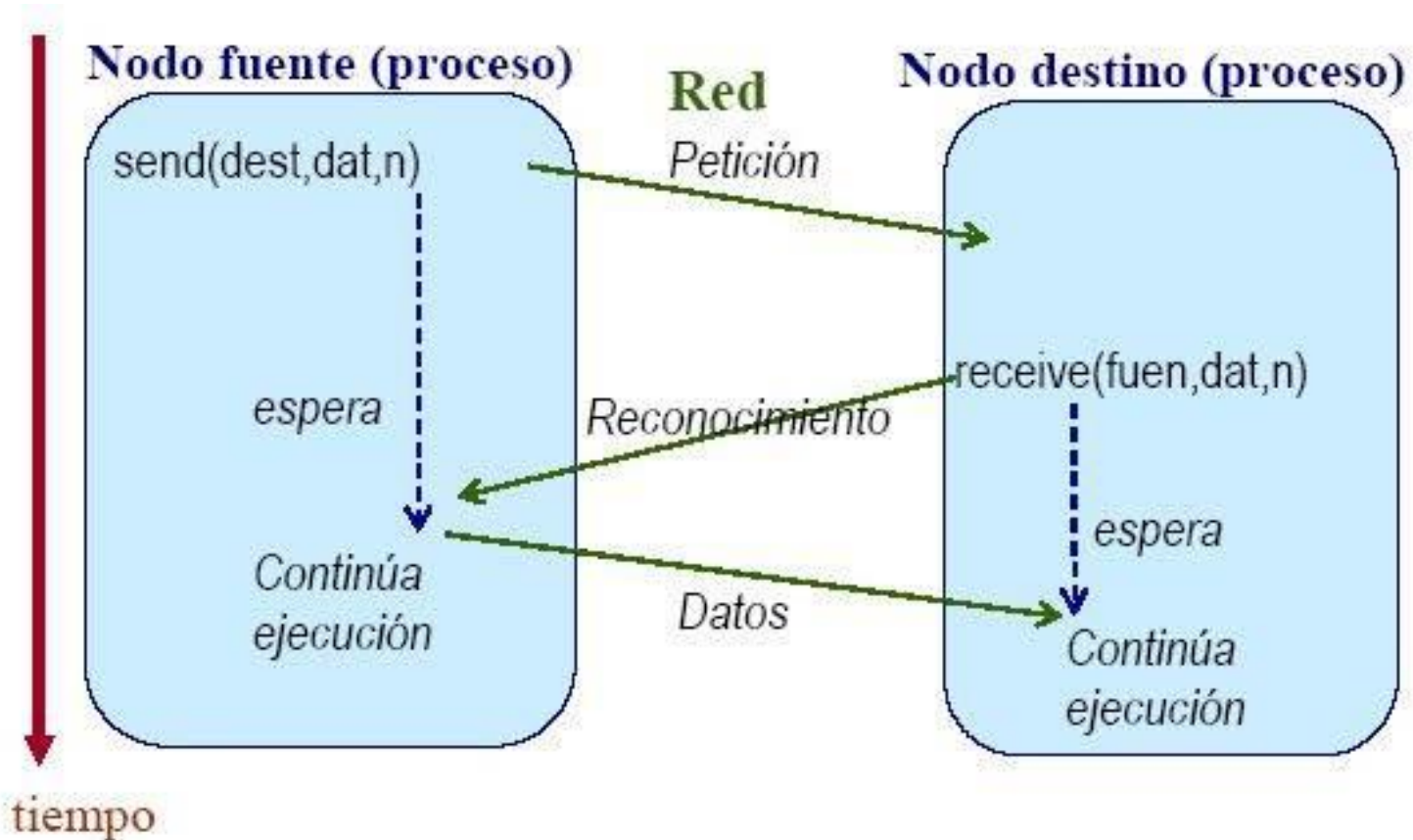
Concepts and motivation

Asynchronous communication in a multicomputer



Concepts and motivation

Synchronous communication in a multicomputer

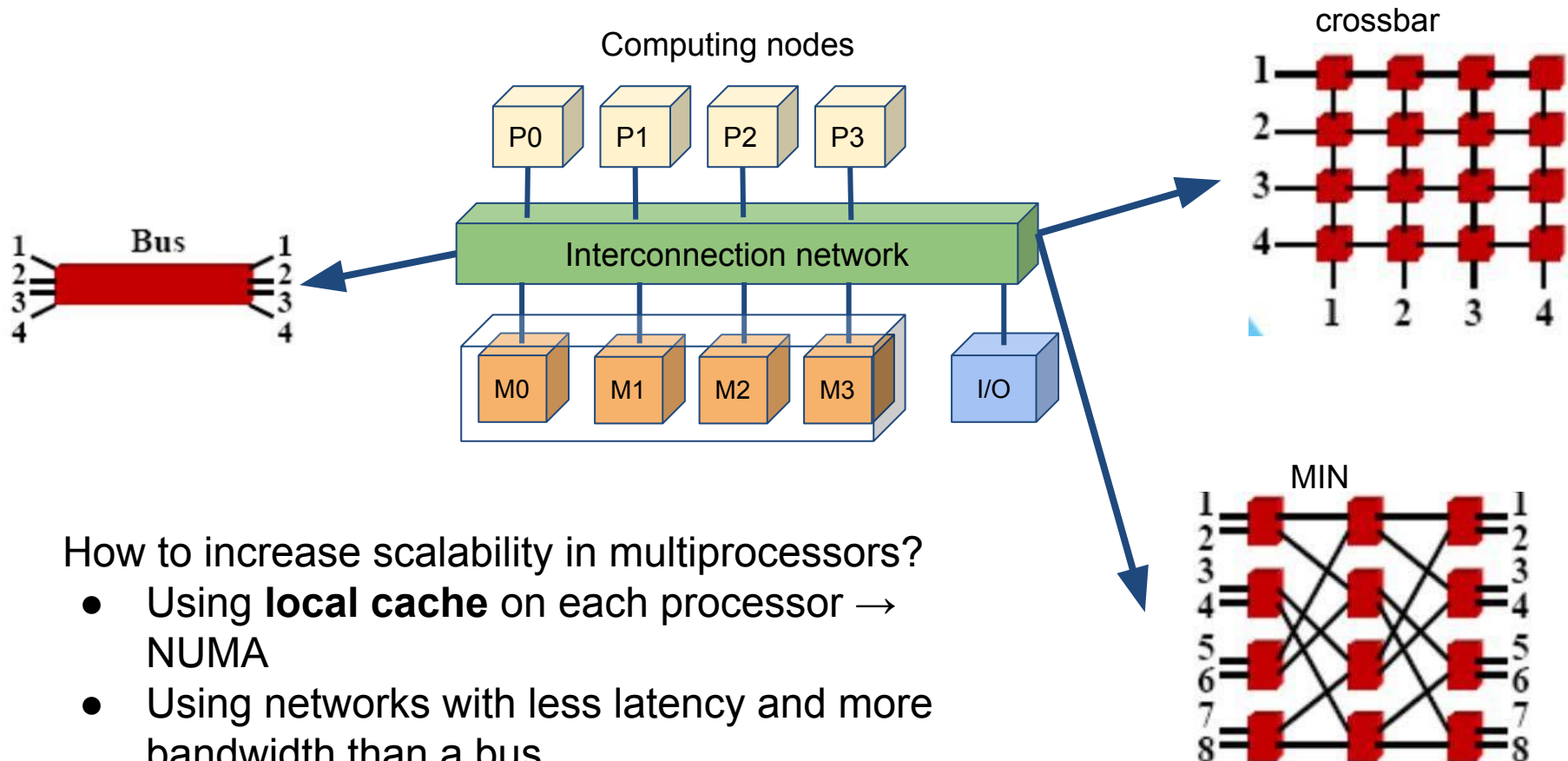


Computer Engineering

Unit 3: Parallel computation

Concepts and motivation

Interconnecting networks in multiprocessors



How to increase scalability in multiprocessors?

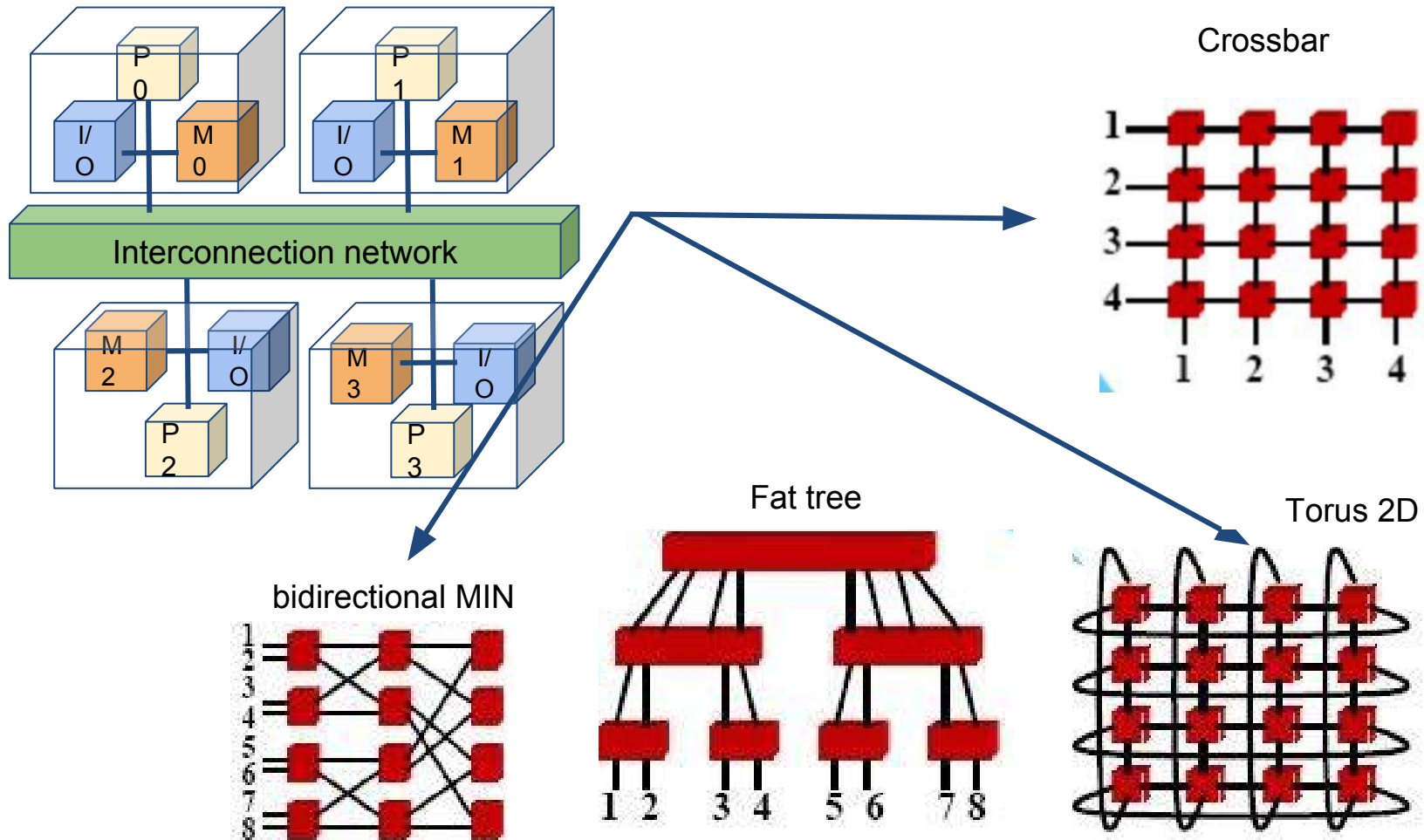
- Using **local cache** on each processor → NUMA
- Using networks with less latency and more bandwidth than a bus
- Turn UMA into NUMA

Computer Engineering

Unit 3: Parallel computation

Concepts and motivation

Interconnecting networks in multiprocessors



Concepts and motivation

Parallel computer and scalability

We can sort the parallel computer in terms of their increasing scalability:

UMA/SMP < COMA < CC-NUMA < NUMA < Multicomputers

Examples:

Multicomputers: IBM-SP2, HP AlphaServer SC45 (cluster of SMP)

NUMA : Cray T3E, Cray X1

CC-NUMA : Origin 3000, HP 9000 Superdome

COMA: KSR-1

SMP: Try to guess

Concepts and motivation

Other parallel computers:

- **MPP (Massive Parallel Processor):** # of processors > 200 . These systems have ad-hoc networks.
- **Cluster:** A set of
- A computer cluster is a set of connected computers that work together. They can be viewed as a single computer resource. The traffic is reduced to that caused by the application.
- **Cluster Beowulf:** Cluster using (Linux) and common hardware and software
- **Constellations:** Large-scale multiprocessors that use clustering of smaller-scale multiprocessors, typically with a DSM or SMP architecture and 32 or more processors.

Concepts and motivation

Other parallel computers:

- **Network of computers:** Set of computer connected using a LAN.
- **GRID:** A computer made of widely distributed computer resources to reach a common goal.

Concepts and motivation

Types of parallelism

Functional parallelism

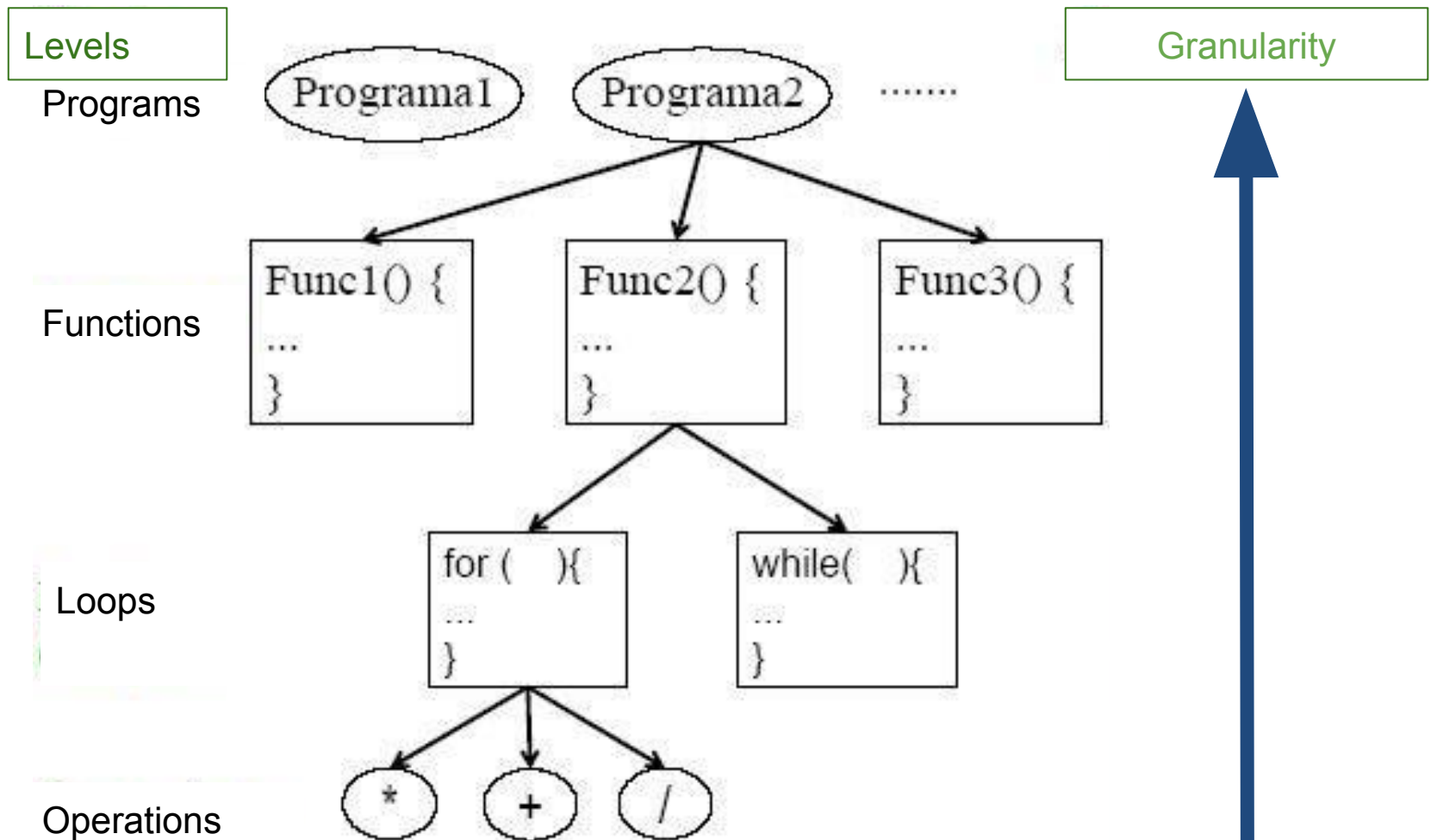
- Obtained from a reorganization of the logic structure of an application. Different levels based in the different software structures (software blocks) taken into account (basic blocks, loops, functions, set of functions, ...). How many software blocks can I run in parallel?

Data parallelism

- Implicit when operating with structures such as vectors or matrices. Related with operations over a big amount of independent data.

Concepts and motivation

Functional parallelism



Concepts and motivation

Types of parallelism regarding its visibility

Explicit parallelism

- If we must express parallelism because it is not present in the language common structure. (I.e. MPI application)

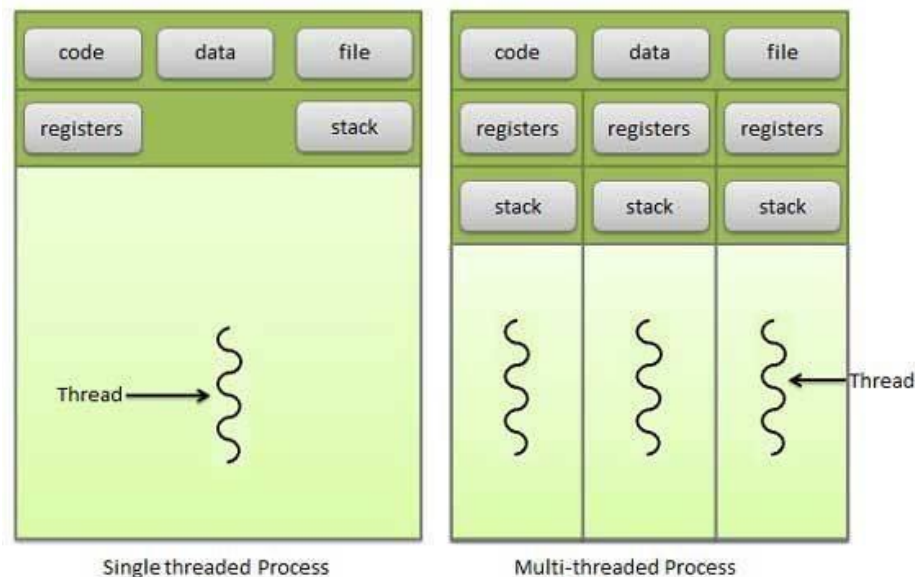
Implicit parallelism

- If we have language structures (loops, vectors, matrices) that naturally can express parallelism (although it can be unexploited).

Concepts and motivation

Execution units for parallelism: threads and processes

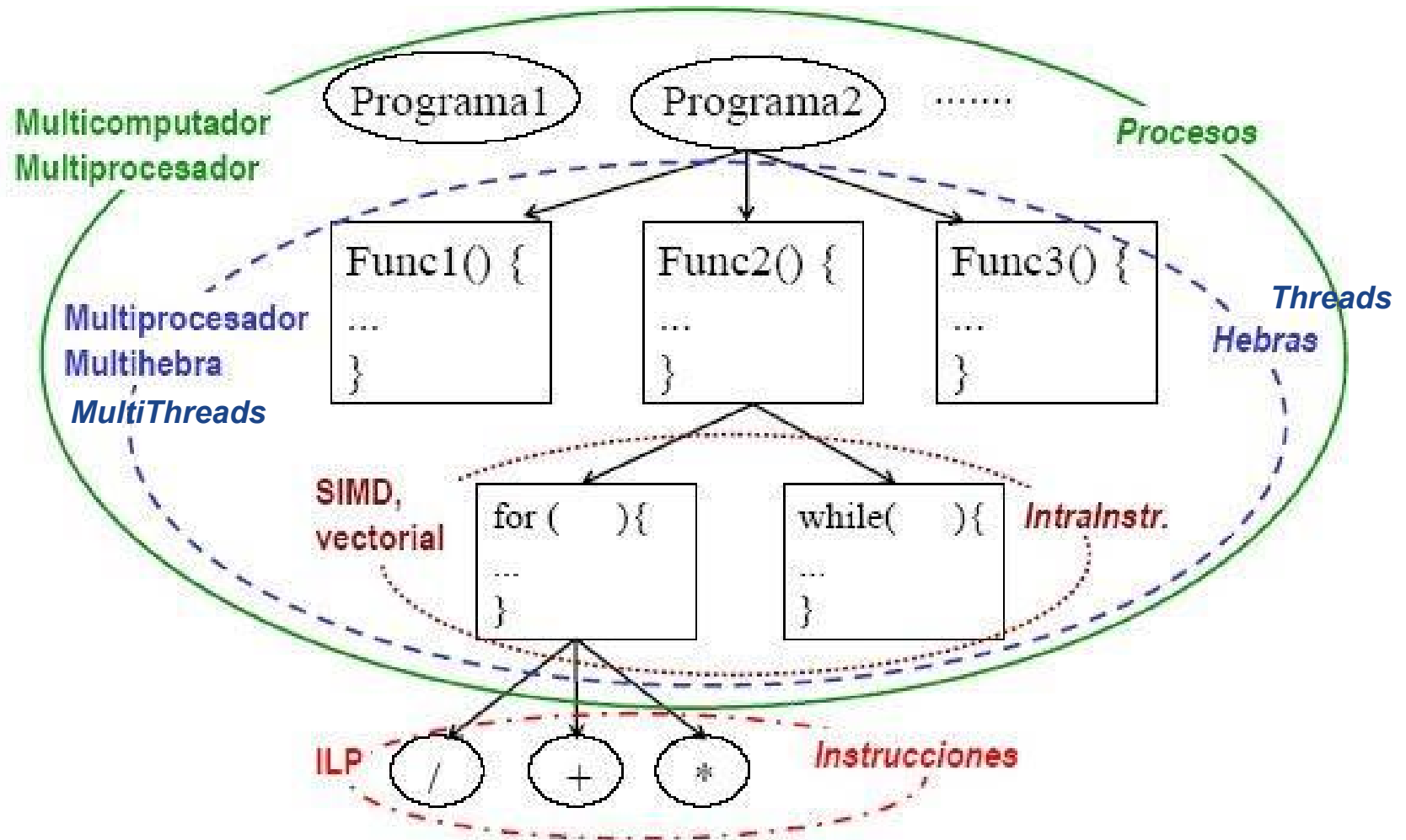
- Hardware (processor): Manage the execution of instructions.
- Software (O.S.): Manage the execution of threads and processes.
 - **Processes:** Have their own virtual addressing space.
 - **Threads:** share virtual addresses, but can be created and destroyed faster than processes and also, the communication is faster.



Computer Engineering

Unit 3: Parallel computation

Concepts and motivation

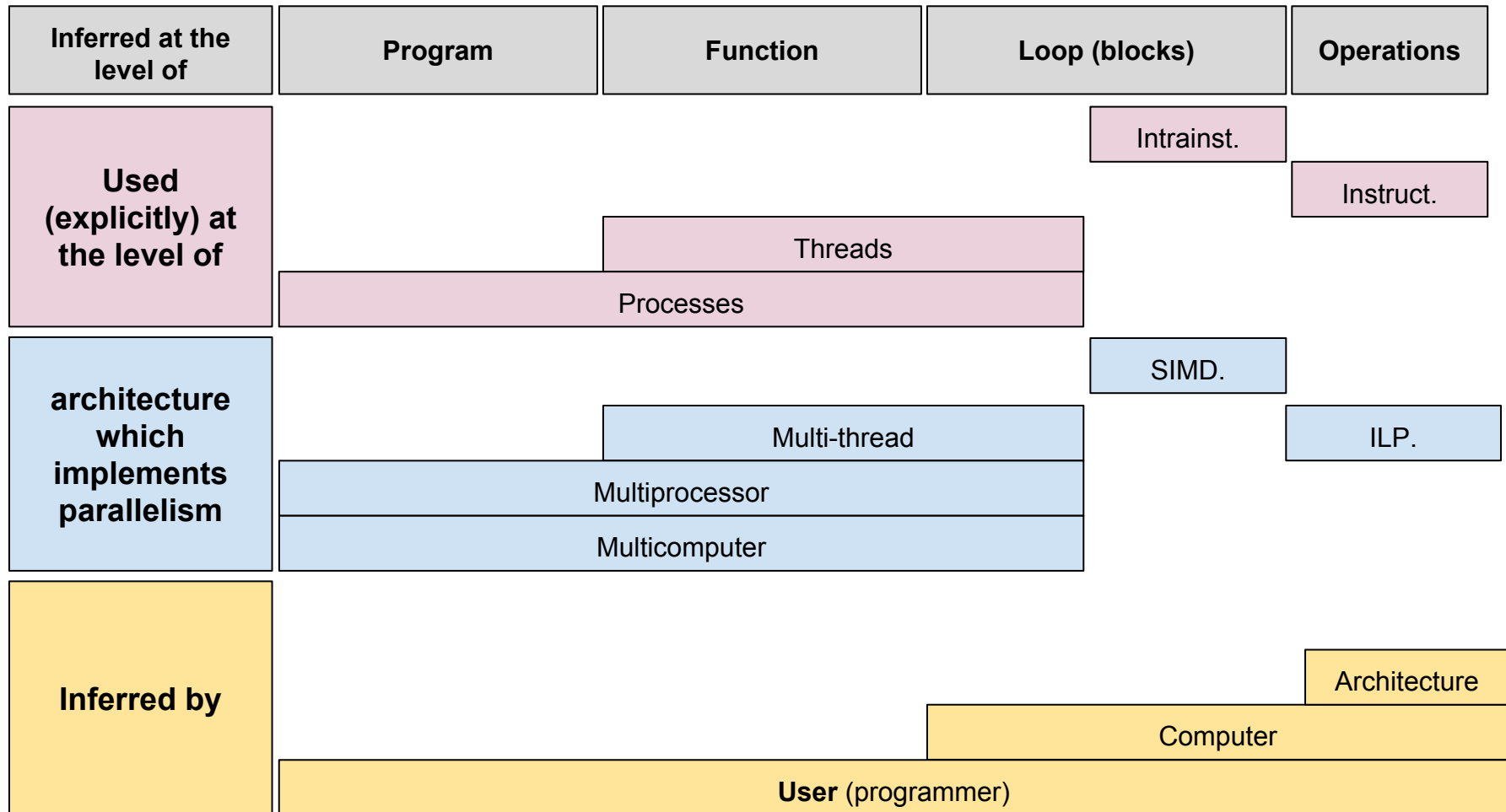


Computer Engineering

Unit 3: Parallel computation

Concepts and motivation

Parallelism detection and extraction



Techniques and procedures

Parallel programming tasks/challenges:

- Code splitting into independent computing units (tasks)
- Task grouping (code & data) into processed/threads
- Computing units (cores) assignment
- Synchronization & Communication

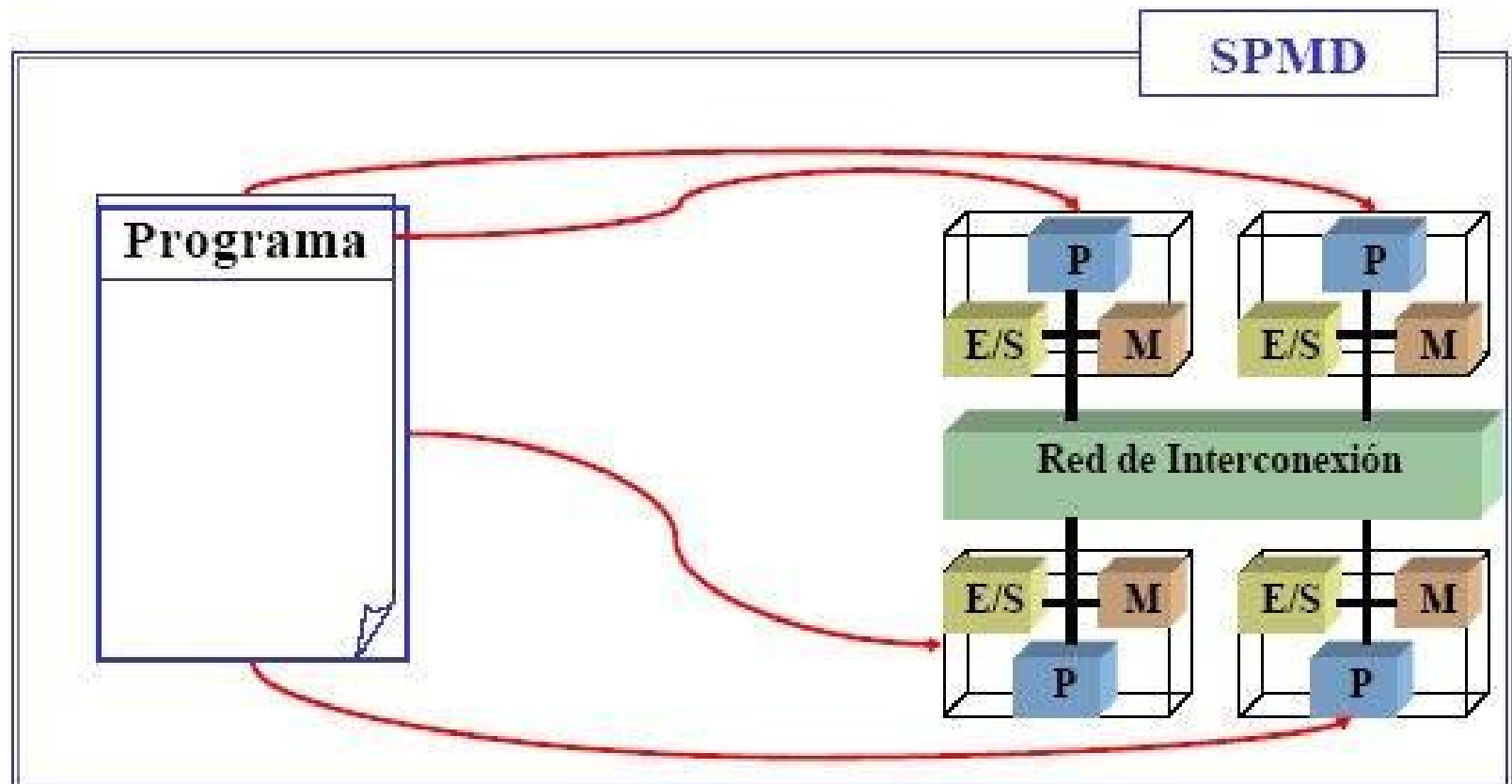
Usual starting point

- Sequential version of the code to be parallelized
- Functional description of the code
- Supporting elements:
 - Parallel code resolving similar problem
 - Parallel libraries (BLAS, ScalaPACK, OpenMP, Intel TBB, ...)

Techniques and procedures

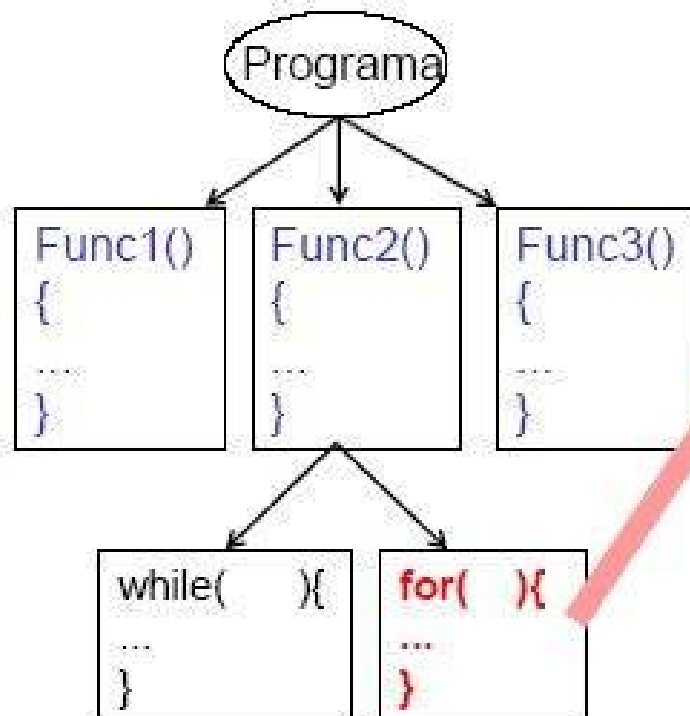
Parallel programming modes:

- **SPMD** (Single Program Multiple Data)



Techniques and procedures

SPMD Example



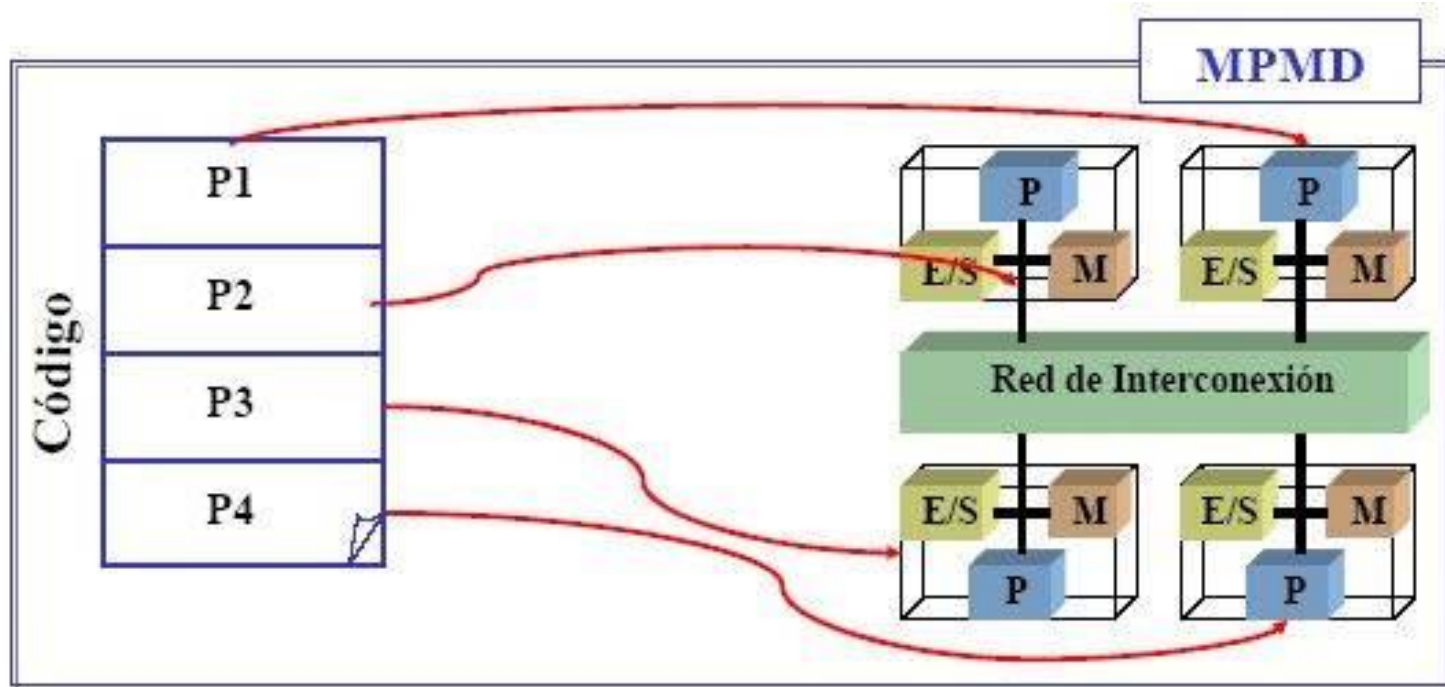
SPMD

```
Func1() {  
...}  
Func2() {  
...  
for (i=ithread;i<N;i=i+nthread){  
    código para la iteración i }  
...}  
Func3() {  
...}  
Main () {  
...  
switch (iproc) {  
    case 0: Func1(); break;  
    case 1: Func2(); break;  
    case 2: Func3(); break; }  
...}
```

Techniques and procedures

Parallel programming modes:

- **MPMD** (Multiple Program Multiple Data)



- **Hybrid SPMD-MPMD**

Techniques and procedures

Tools supporting parallel programming

- Parallel compiler. Automatic inference of parallelism.
- Compiler directives/pragmas/artifacts (OpenMP)
- Parallel languages (HPF, Occam, Ada)
- Libraries: PThreads, MPI, PVM, Intel TBB (sequential code + interfacing functions)

Techniques and procedures

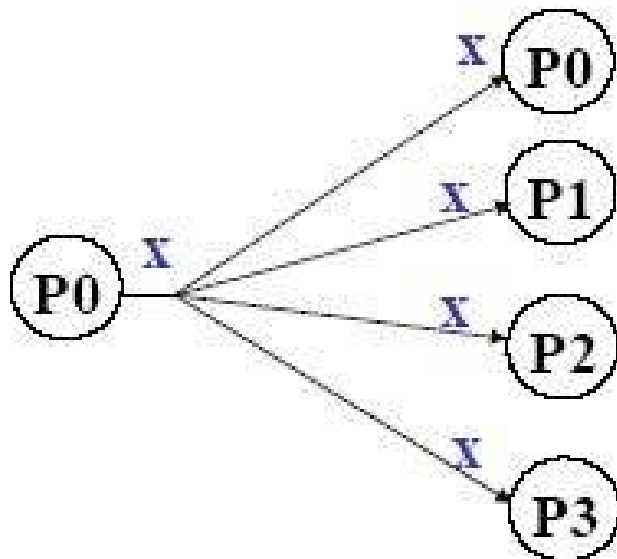
Collective communication

- **One to all** : broadcasting (MPI_Bcast), scattering (MPI_Scatter)
- **All to one**: Reduction (MPI_Reduce), gathering (MPI_Gather)
- **All to all complete exchange**: All-broadcast (MPI_AllBroadcast), All-scattering (MPI_AllScatter)
- **Multiple one-by-one**: Permutations, Shifting
- **Composed services**: barrier synchronization (MPI_Barrier), scan (MPI_Scan)

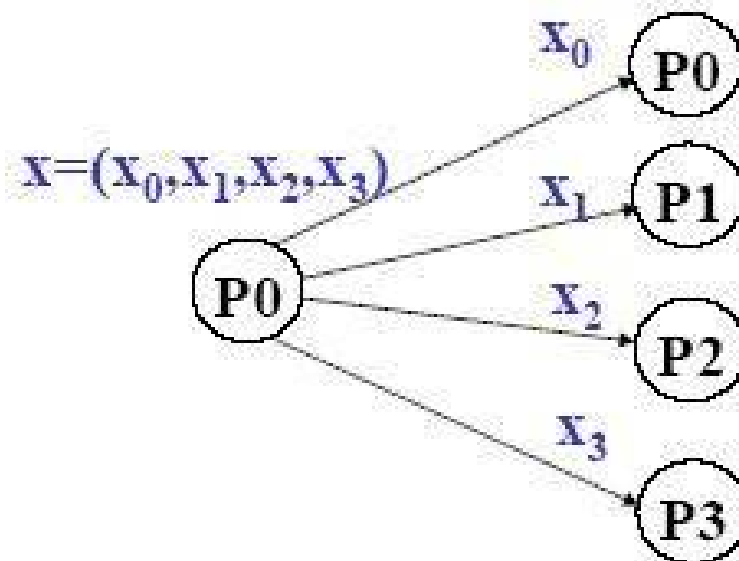
Techniques and procedures

Collective communication: **one by all**

Difusión (*broadcast*)



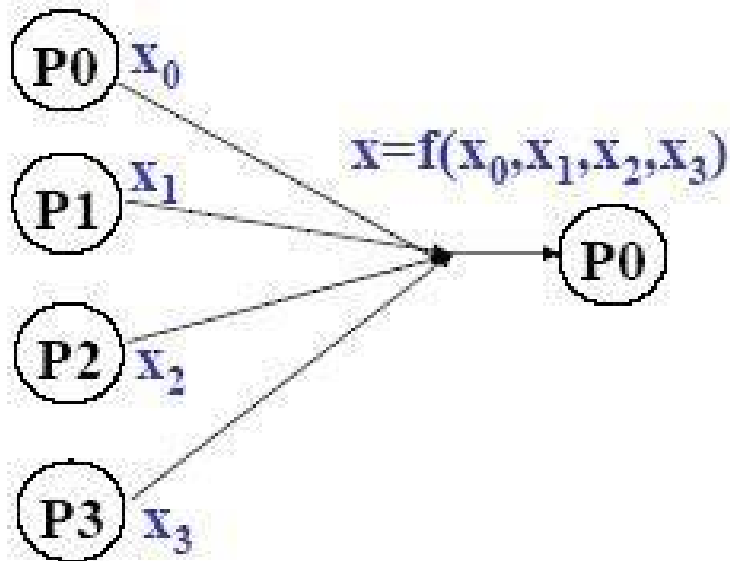
Dispersión (*scatter*)



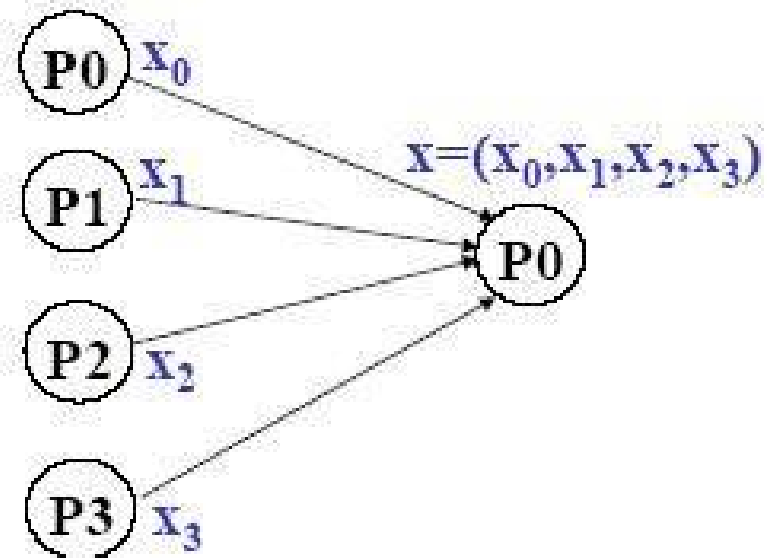
Techniques and procedures

Collective communication: **all to one**

Reducción

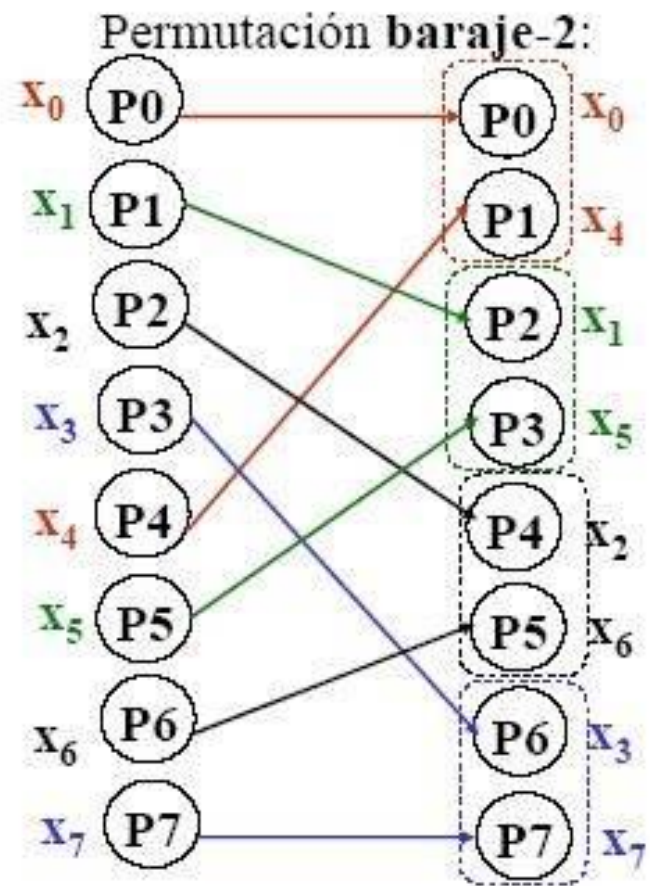
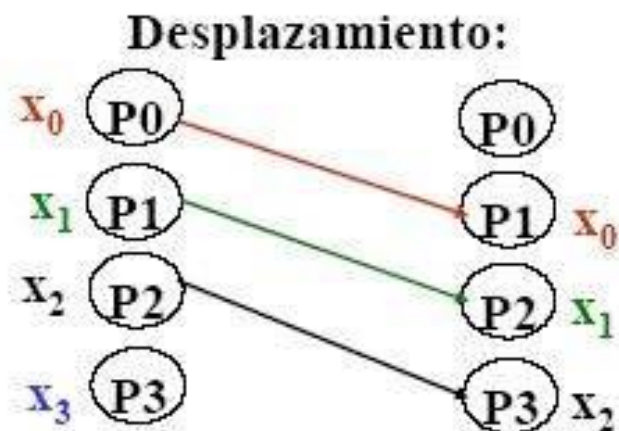
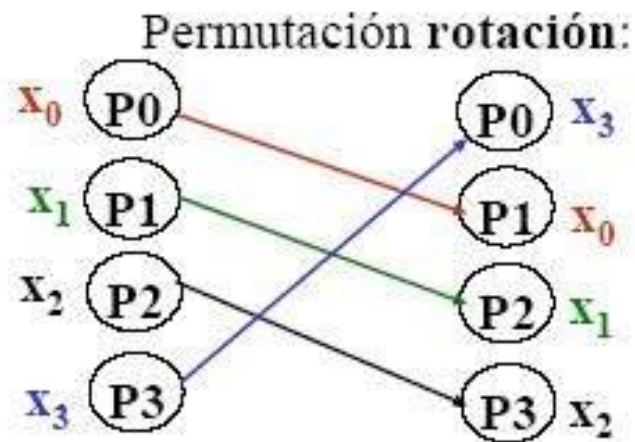


Acumulación (*gather*)



Techniques and procedures

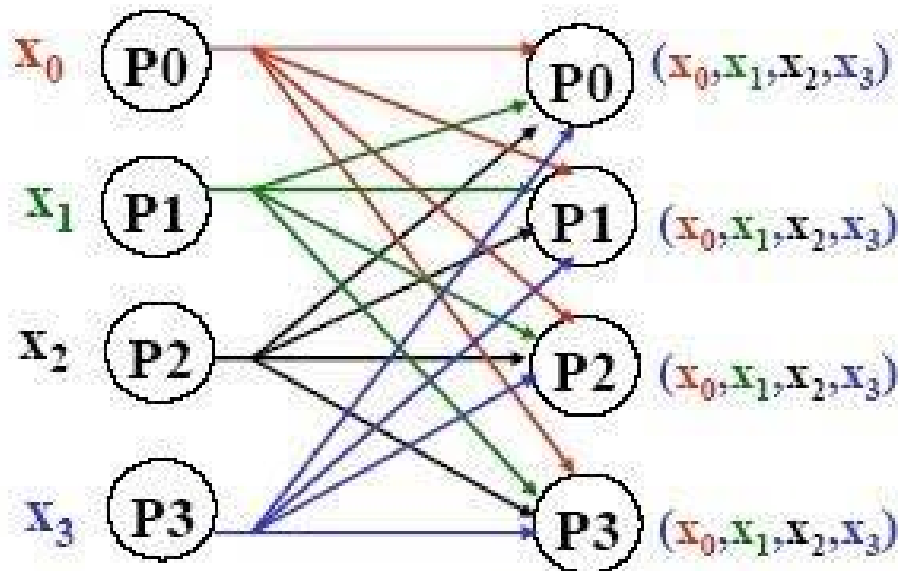
Collective communication: **multiple one-by-one**



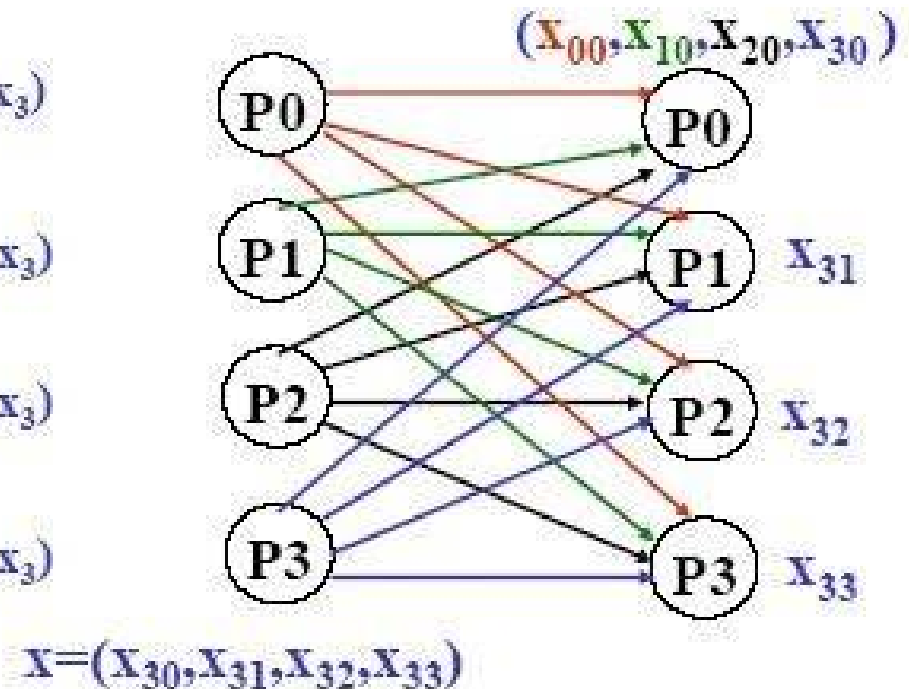
Techniques and procedures

Collective communication: **all to all**

Todos Difunden (*all-broadcast*)
o chismorreo (*gossiping*)



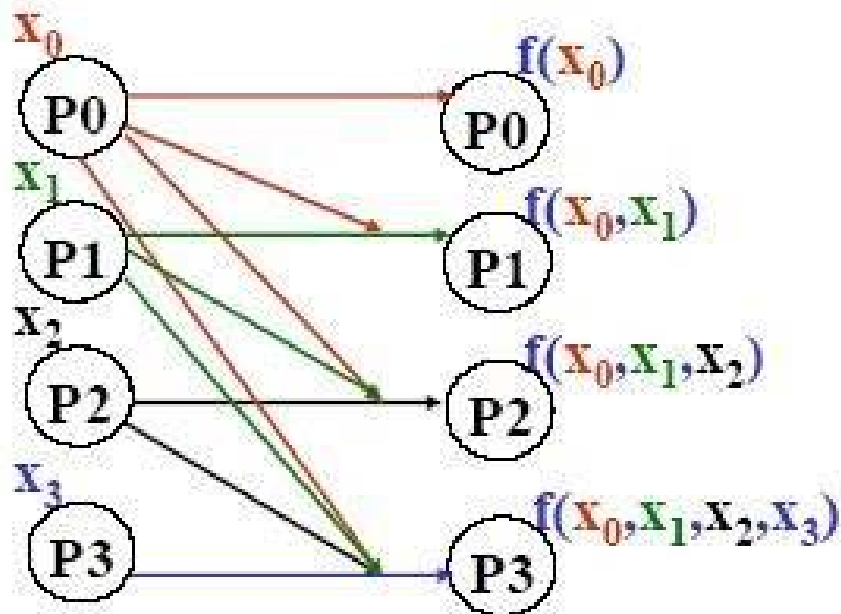
Todos Dispersan (*all-scatter*)



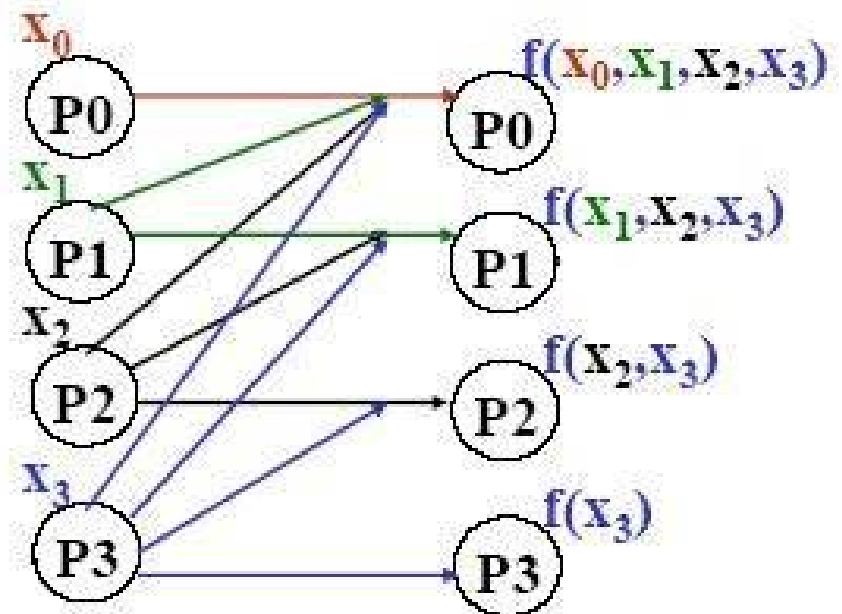
Techniques and procedures

Collective communication: **compound services**

Recorrido prefijo paralelo



Recorrido sufijo paralelo



Techniques and procedures

Parallel programming styles: **message passing**

- Software tools:
 - Programming languages (Ada2012) and libraries (MPI, PVM, Intel TBB)
- Workload distribution:
 - Using libraries: explicitly with language sequential sentences (if, for, ...)
 - Using a parallel language: explicitly with language special primitives (l.e. Occam'par)
- Basic communicating primitives : Send and Receive
- Collective communication functions
- Synchronization
 - Blocking receive. Barriers (l.e. MPI_Barrier)
 - Blocking Send-receive (l.e ADA)

Techniques and procedures

Parallel programming styles: **shared variables**

- Programming languages (Ada), libraries (OpenMP, Intel TBB), compiler directives+functions (OpenMP)
- Workload distribution
 - Libraries: explicit within the sequential code
 - Directives: implicit synchronization (higher abstraction level)
 - Languages: built-in language constructions
- Basic communication: load & store
- Collective communication
- Synchronization
 - Semaphores, conditional variables, locks

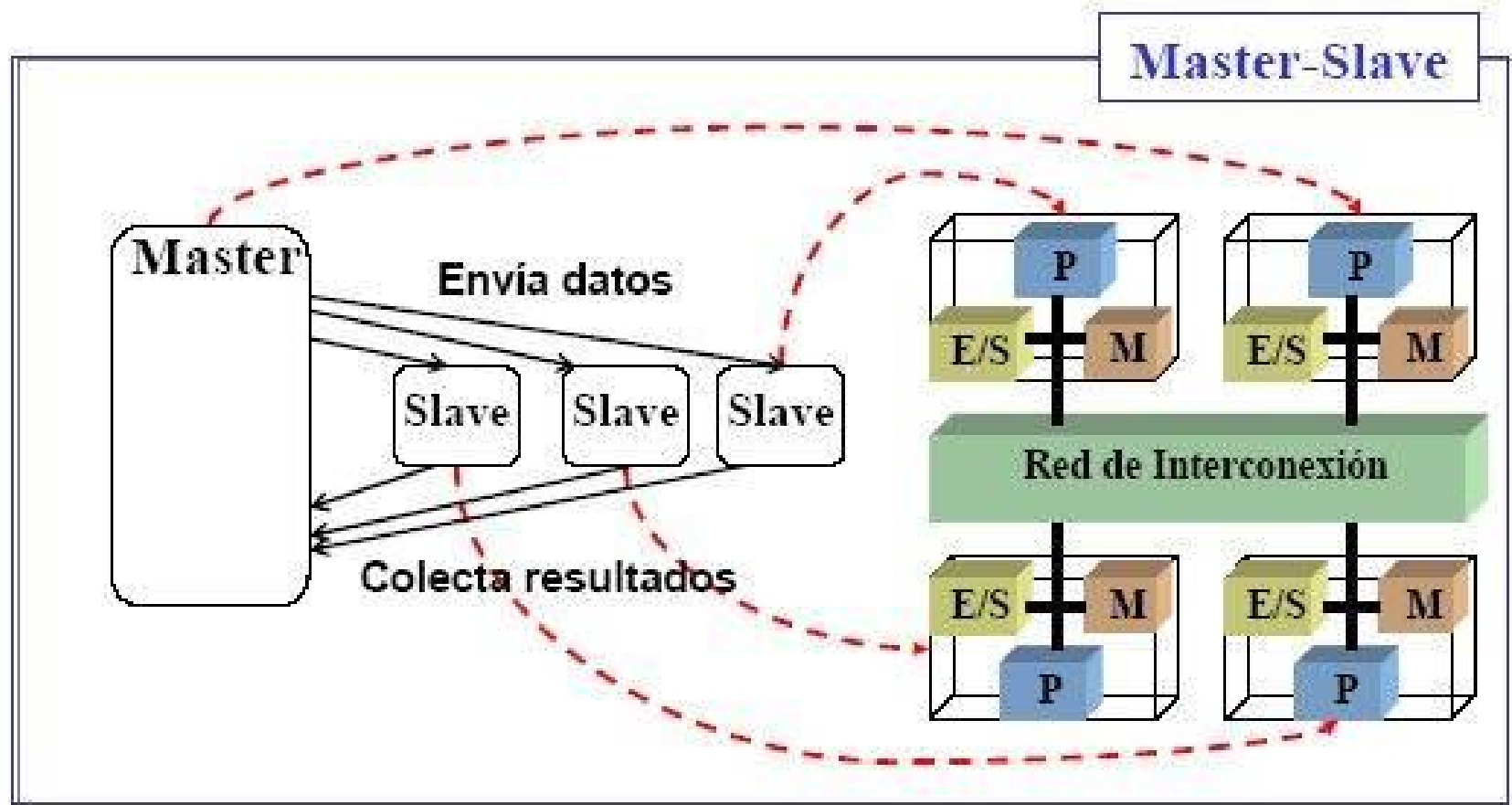
Techniques and procedures

Parallel programming styles: **data parallelism**

- Tools
 - Programming languages (HPF – High Performance Fortran)
- Workload distribution
 - Directives to distribute data among processors
 - Directives for loop parallelization
- Basic communication: implicit $\rightarrow A(i) = A(i-1)$
- Collective communication
 - Implicit in the functions used by the programmer (i.e. rotations – CSHIFT)
- Synchronization: implicit

Techniques and procedures

Parallelism structures: **master-slave**



Techniques and procedures

Parallelism structures: master slave (using 2 different parallel modes)

Master-Slave como MPMD– SPMD

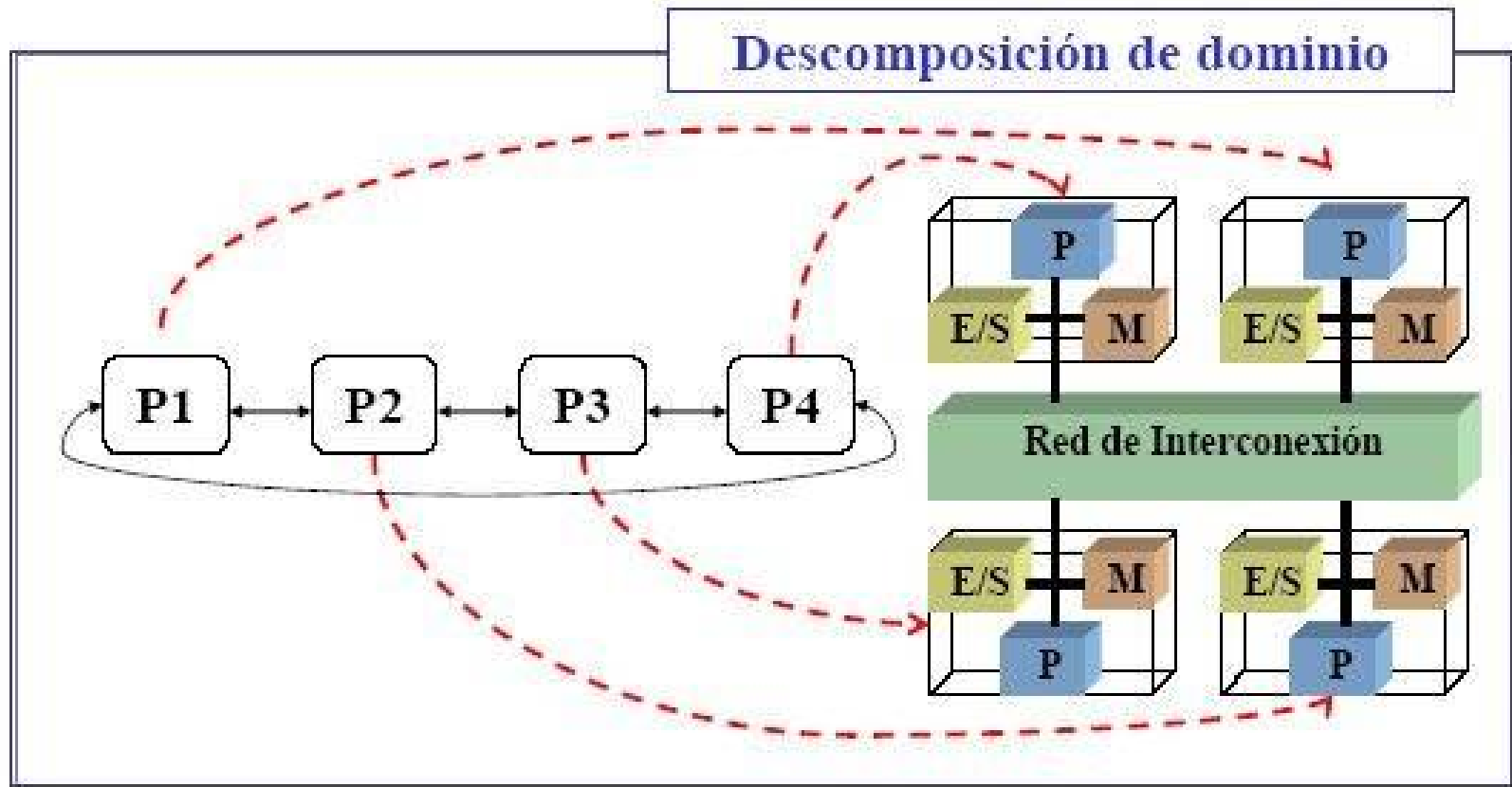
```
main ()  
{ código Master  
}  
-----  
main ()  
{ código Slaves  
}
```

Master-Slave como SPMD

```
main ()  
{ if (iproc=id_Master) {  
    código Master  
  } else {  
    código Slaves  
  }  
}
```

Techniques and procedures

Parallelism structures: **domain decomposition** → **data parallelism**

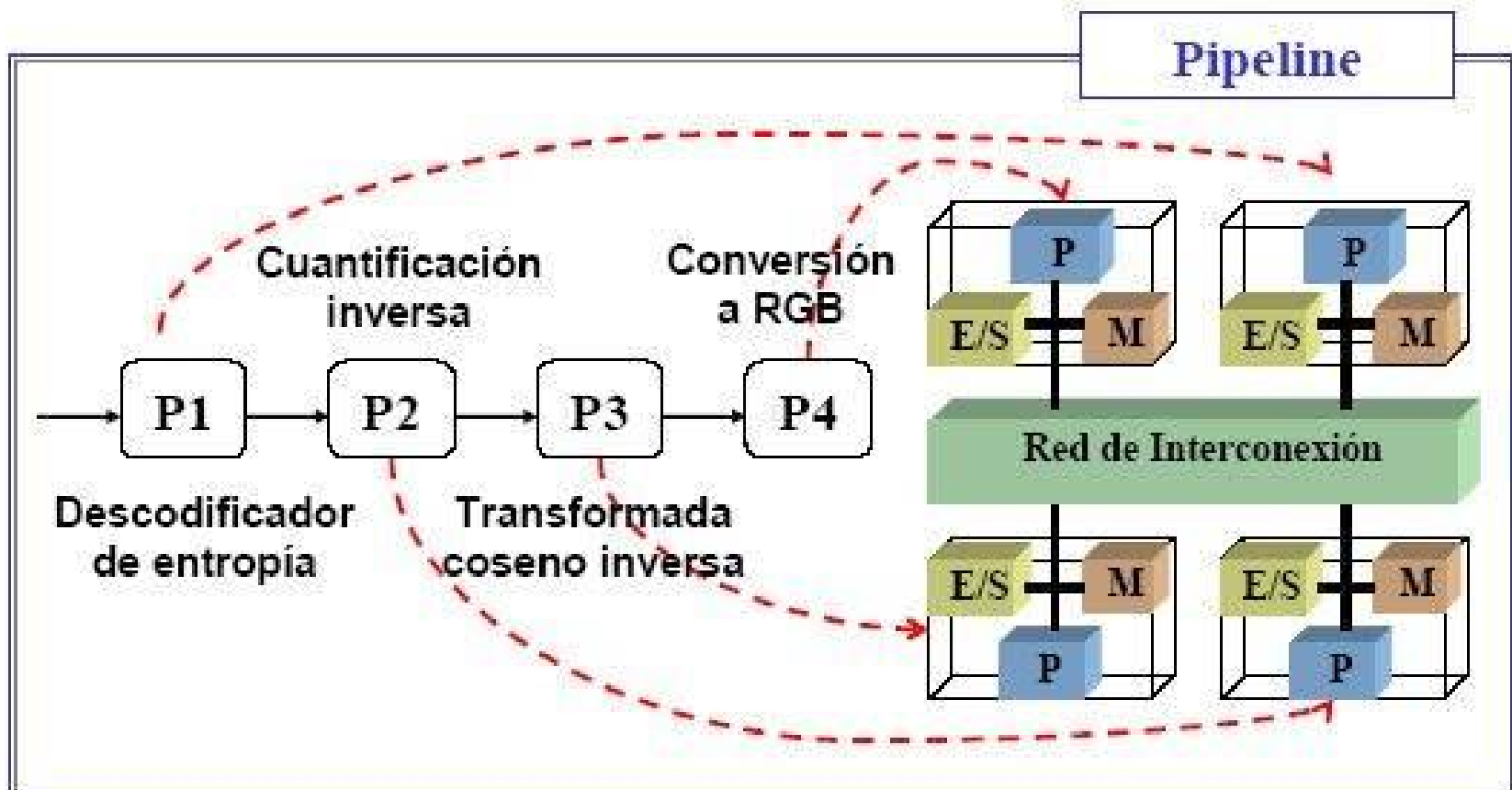


Computer Engineering

Unit 3: Parallel computation

Techniques and procedures

Parallelism structures: pipeline → data flow

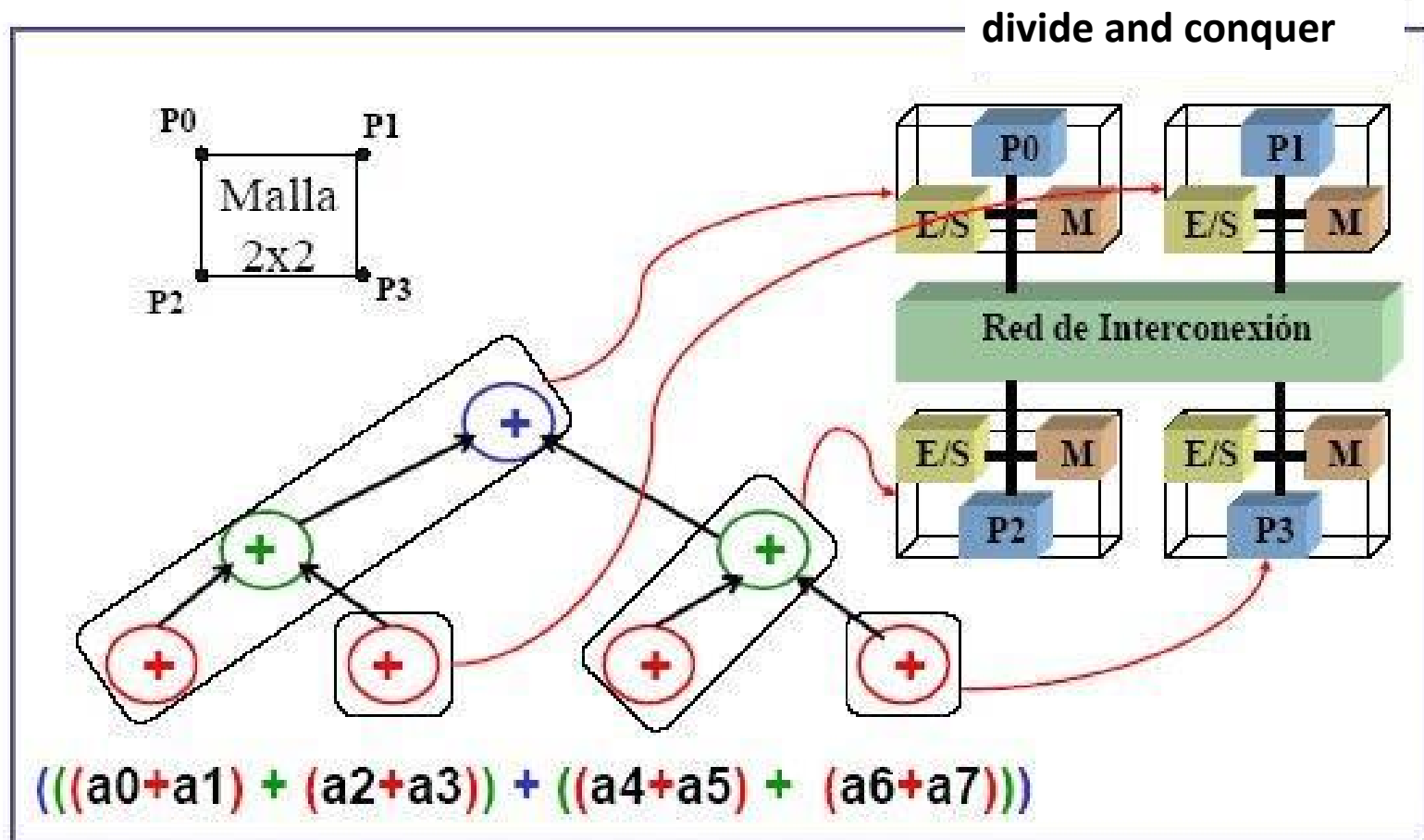


Computer Engineering

Unit 3: Parallel computation

Techniques and procedures

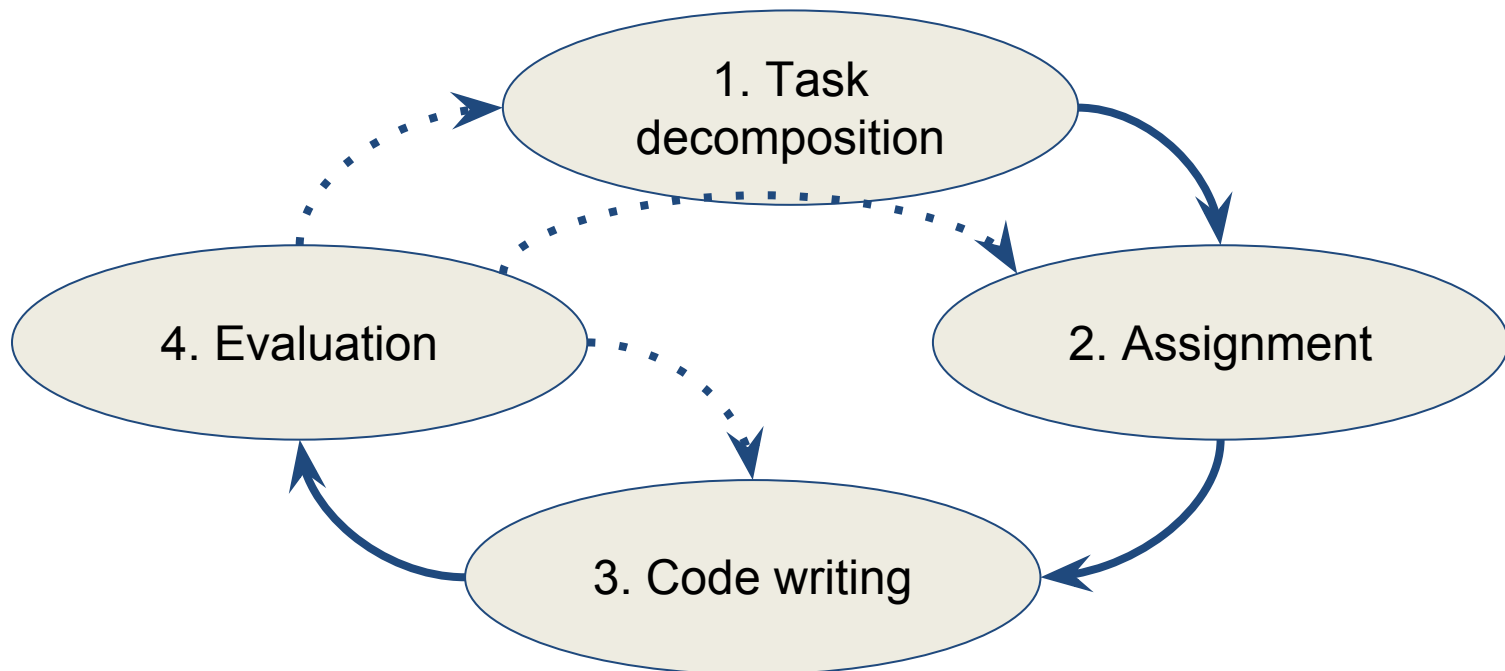
Parallelism structures: **divide and conquer** (recursive decomposition)



Techniques and procedures

Parallelization process

1. Independent tasks **decomposition**
2. Task **assignment** to threads and/or processes.
3. Writing the parallel **code**
4. Performance evaluation



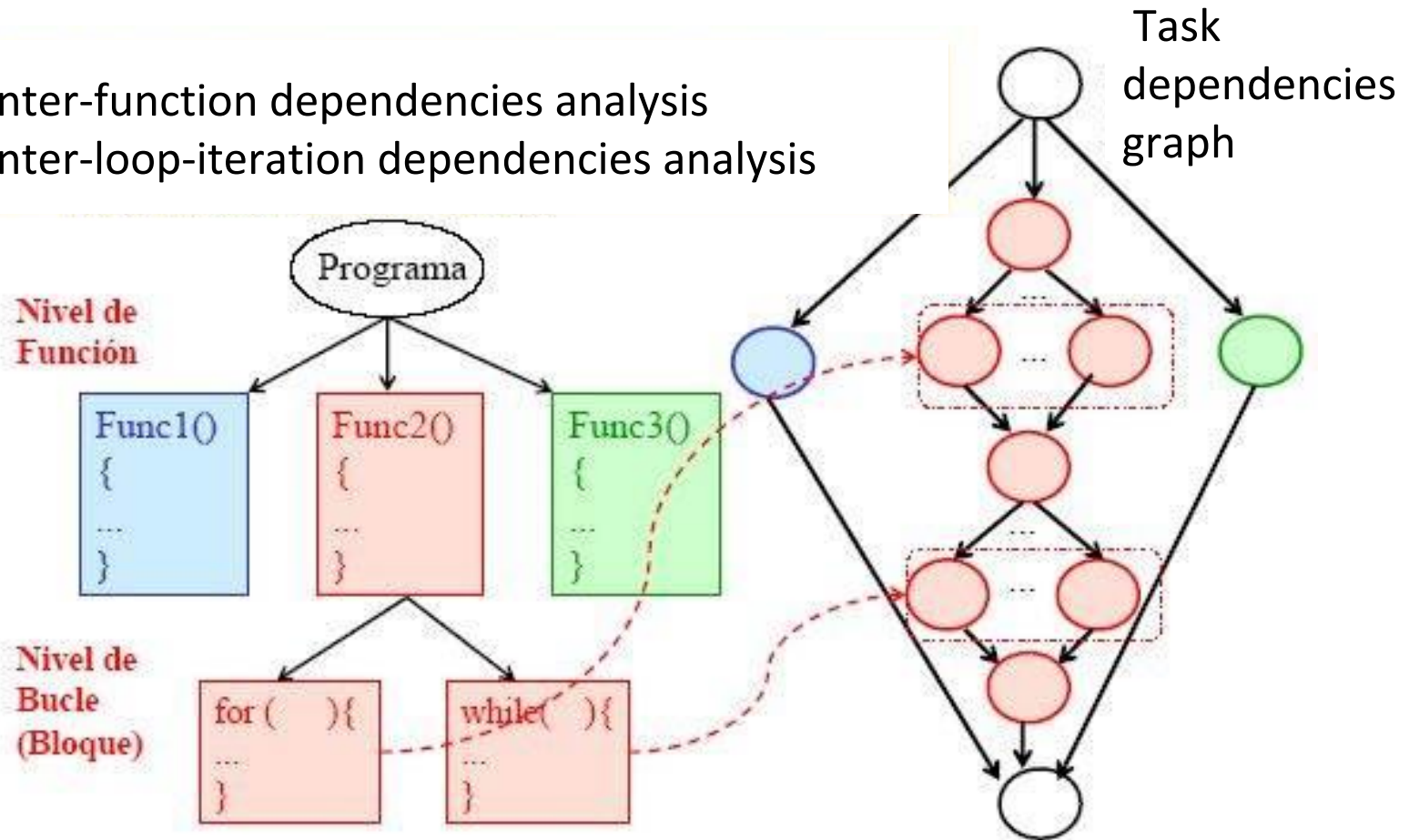
Computer Engineering

Unit 3: Parallel computation

Techniques and procedures

Parallelization process: independent tasks decomposition

- Inter-function dependencies analysis
- Inter-loop-iteration dependencies analysis



Techniques and procedures

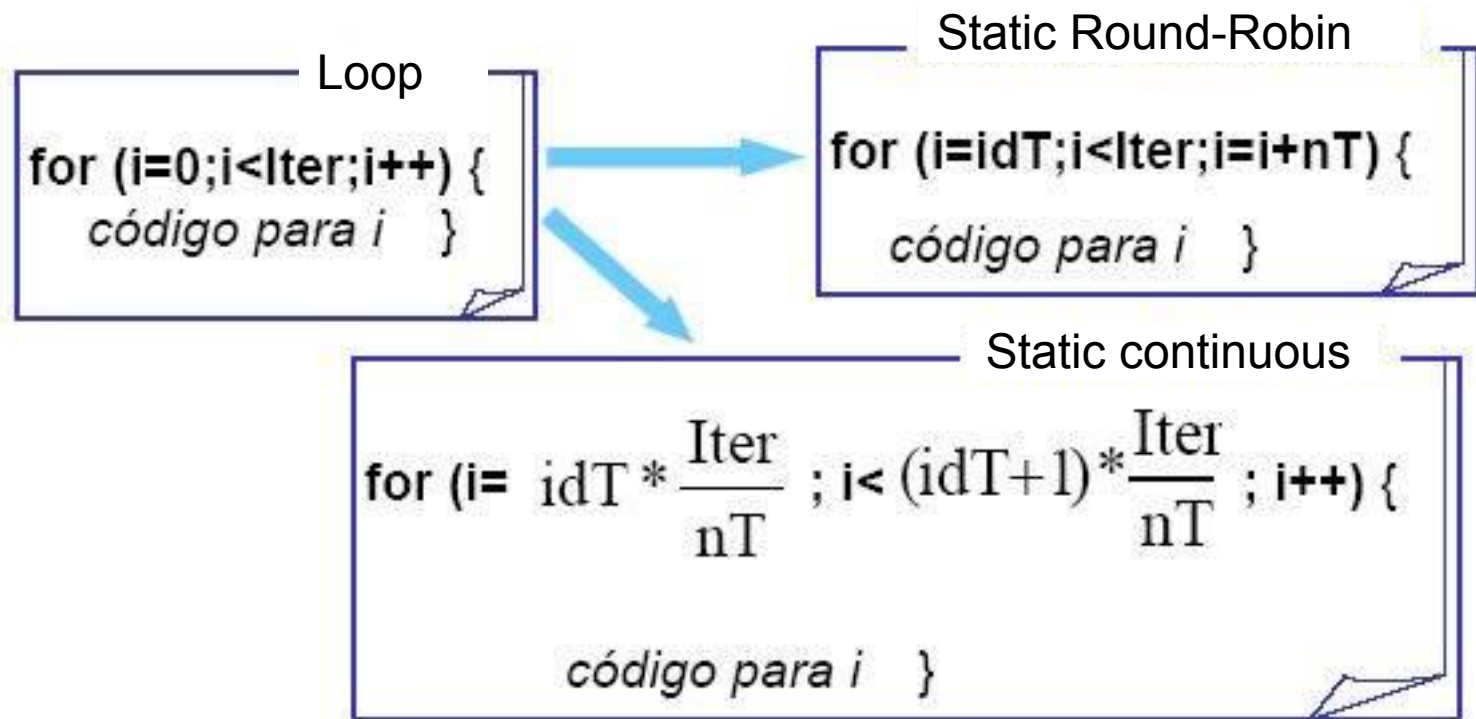
Parallelization process: Tasks assignment

- Usually: loop iteration iteraciones → threads hebras and functions/methods -> processes
- Granularity depends on
 - # processors
 - Relation between communication & synchronization time and bare computing time
- Inherent tradeoff in the workload: (no processors should wait to any other)
- Assignment types:
 - Dynamic (while executing) → Usefull if we don't know the # of tasks.
 - Static (designed by programmer or compiler)

Techniques and procedures

Parallelization process: Tasks assignment

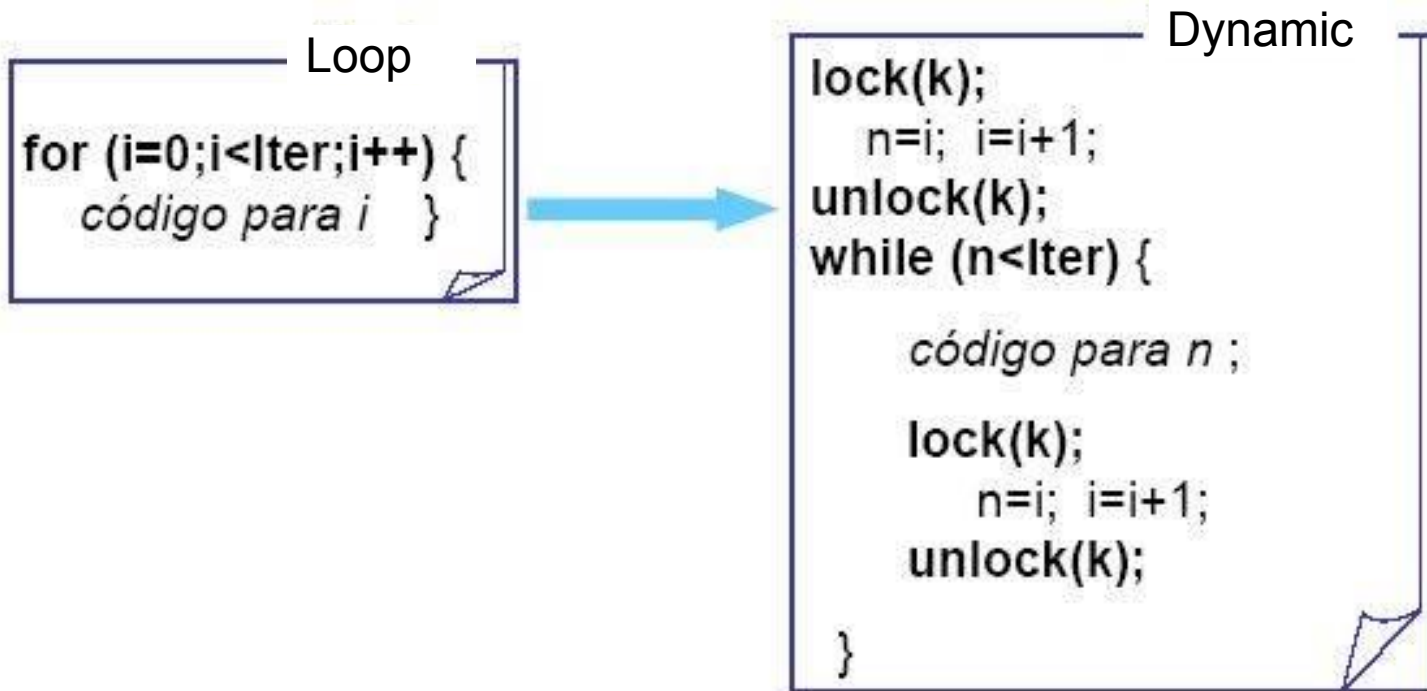
- Static assignment



Techniques and procedures

Parallelization process: Tasks assignment

- Dynamic assignment



Techniques and procedures

Parallelization process: writing parallel code

- It depends on:
 - Programming style: message passing, etc.
 - Programming mode
 - Starting point
 - Tool used for parallelism
 - Program structure
- A parallel program should include:
 - Processes/threads creation and destruction
 - Workload assignation
 - Communication and synchronization routines

Computer Engineering

Unit 3: Parallel computation

Techniques and procedures

π calculus by task decomposition

$$\left. \begin{array}{l} \operatorname{arctg}'(x) = \frac{1}{1+x^2} \\ \operatorname{arctg}(1) = \frac{\pi}{4} \\ \operatorname{arctg}(0) = 0 \end{array} \right\} \Rightarrow \int_0^1 \frac{1}{1+x^2} = \operatorname{arctg}(x) \Big|_0^1 = \frac{\pi}{4} - 0$$

Pi can be calculated by means of numerical integration



Computer Engineering

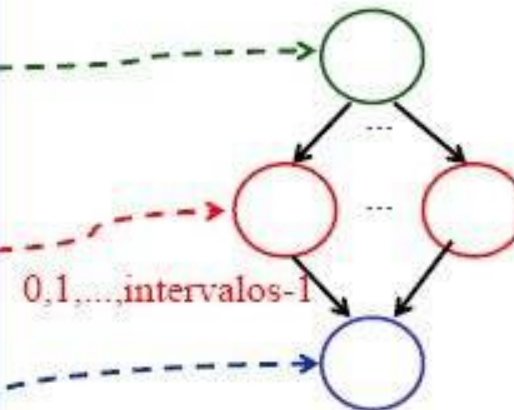
Unit 3: Parallel computation

Techniques and procedures

π calculus by task decomposition

```
main(int argc, char **argv) {  
    double ancho, sum;  
    int intervalos, i;  
  
    intervalos = atoi(argv[1]);  
    ancho = 1.0/(double) intervalos;  
    for (i=0; i< intervalos; i++){  
        x = (i+0.5)*ancho;  
        sum = sum + 4.0/(1.0+x*x);  
    }  
    sum* = ancho;  
}
```

Task dependencies graph



Techniques and procedures

π calculus by task decomposition

- Static **assignment** of each loop iteration (Round Robin assign.)
- Parallel code signature
 - Programming style: Message passing
 - Programming mode: SPMD
 - Starting from: sequential version of the code
 - Tool: MPI
 - Program structure: Data parallelism or Divide and Conquer

Computer Engineering

Unit 3: Parallel computation

Techniques and procedures

π calculus by task decomposition using MPI

```
#include <mpi.h>
main(int argc, char **argv) {
    double ancho,x, sum, tsum; int intervalos, i; int nproc, iproc;
    if (MPI_Init(&argc, &argv) != MPI_SUCCESS) exit(1);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);
    intervalos = atoi(argv[1]); ancho = 1.0 / (double) intervalos; lsum = 0;
    for (i=iproc; i<intervalos; i+=nproc) {
        x = (i + 0.5) * ancho; lsum += 4.0 / (1.0 + x * x);
    }
    lsum *= ancho;
    MPI_Reduce(&tsum, &sum, 1, MPI_DOUBLE,
              MPI_SUM,0,MPI_COMM_WORLD);
    MPI_Finalize();
}
```

Enrolar

Asignar/
Localizar

Comunicar/
sincronizar

Desenrolar

Techniques and procedures

π calculus by task decomposition

- Dynamic **assignment** of each loop iteration
- Parallel code signature:
 - Programming style: directives + API (OpenMP)
 - Programming mode: SPMD
 - Starting from: sequential version of the code
 - Tool: OpenMP
 - Program structure: Data parallelism or Divide and Conquer

Techniques and procedures

π calculus by task decomposition using OpenMP

```
#include <omp.h>
#define NUM_THREADS 4
main(int argc, char **argv) {
    double ancho, x, sum; int intervalos, i;
    intervalos = atoi(argv[1]); ancho = 1.0/(double) intervalos;

    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel
    #pragma omp for reduction(+:sum) private(x)
    schedule(dynamic)
    for (i=0; i< intervalos; i++) {
        x = (i+0.5)*ancho; sum = sum + 4.0/(1.0+x*x);
    }
    sum* = ancho;
}
```

Crear/ Terminar

Comunicar/ sincronizar

Localizar

Asignar