ICTP
The Abdus Salam
International Centre
for Theoretical Physics

United Nations
Educational, Scientific and
Cultural Organization

IAEA
International Atomic Energy Agency

Basic Principles of Parallelism

Ivan Girotto – igirotto@ictp.it

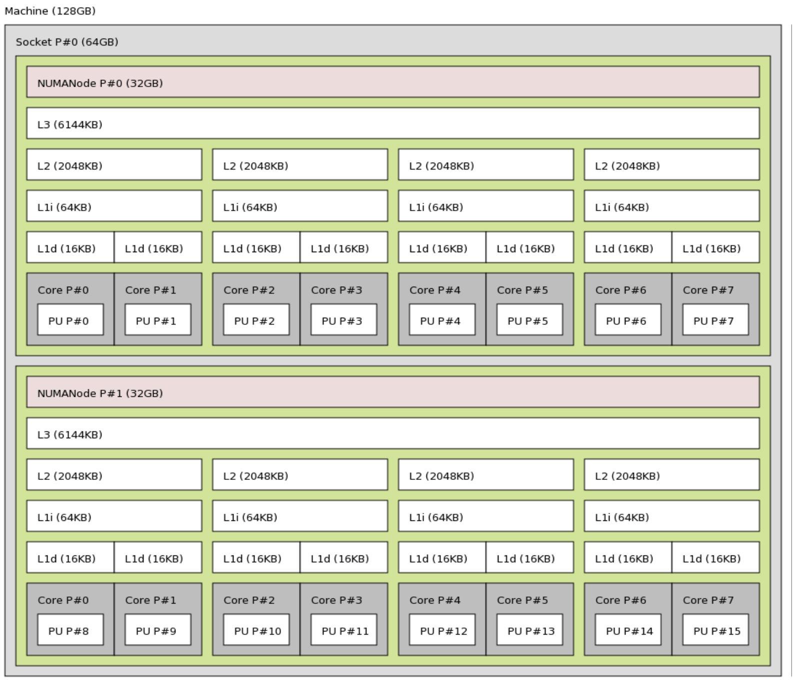
Information & Communication Technology Section (ICTS)

International Centre for Theoretical Physics (ICTP)

Why Parallel Programming

- Solve larger problems
- Run memory demanding codes
- Solve problems with greater speed
- Today?!
    - Processing of massive amount of data
    - Inescapable to exploit modern CPU systems

Modern CPUs Models



Machine (128GB)

Socket P#0 (64GB)

NUMANode P#0 (32GB)

L3 (6144KB)

L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)

L1i (64KB) L1i (64KB) L1i (64KB) L1i (64KB)

L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB)

Core P#0 PU P#0 | Core P#1 PU P#1 | Core P#2 PU P#2 | Core P#3 PU P#3 | Core P#4 PU P#4 | Core P#5 PU P#5 | Core P#6 PU P#6 | Core P#7 PU P#7

NUMANode P#1 (32GB)

L3 (6144KB)

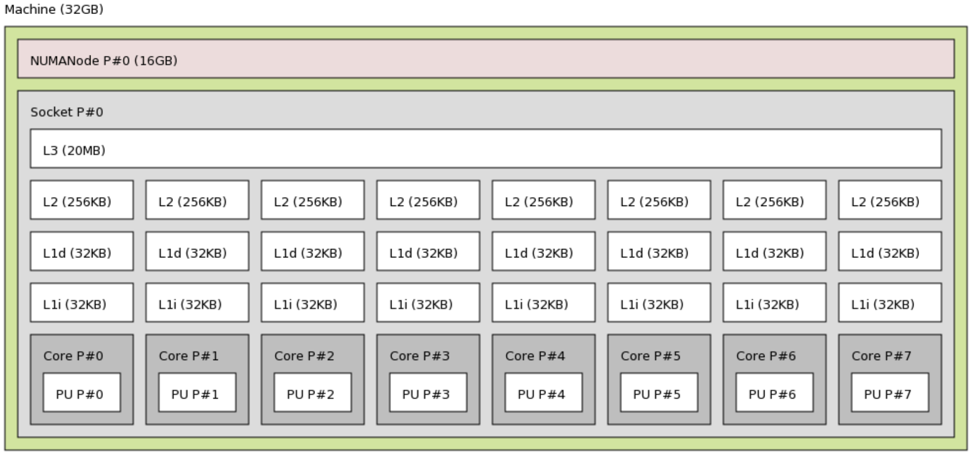
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)

L1i (64KB) L1i (64KB) L1i (64KB) L1i (64KB)

L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB)

Core P#0 PU P#8 | Core P#1 PU P#9 | Core P#2 PU P#10 | Core P#3 PU P#11 | Core P#4 PU P#12 | Core P#5 PU P#13 | Core P#6 PU P#14 | Core P#7 PU P#15

The AMD Opteron 6380 Abu Dhabi 2.5GHz



Machine (32GB)

NUMANode P#0 (16GB)

Socket P#0

L3 (20MB)

L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB)

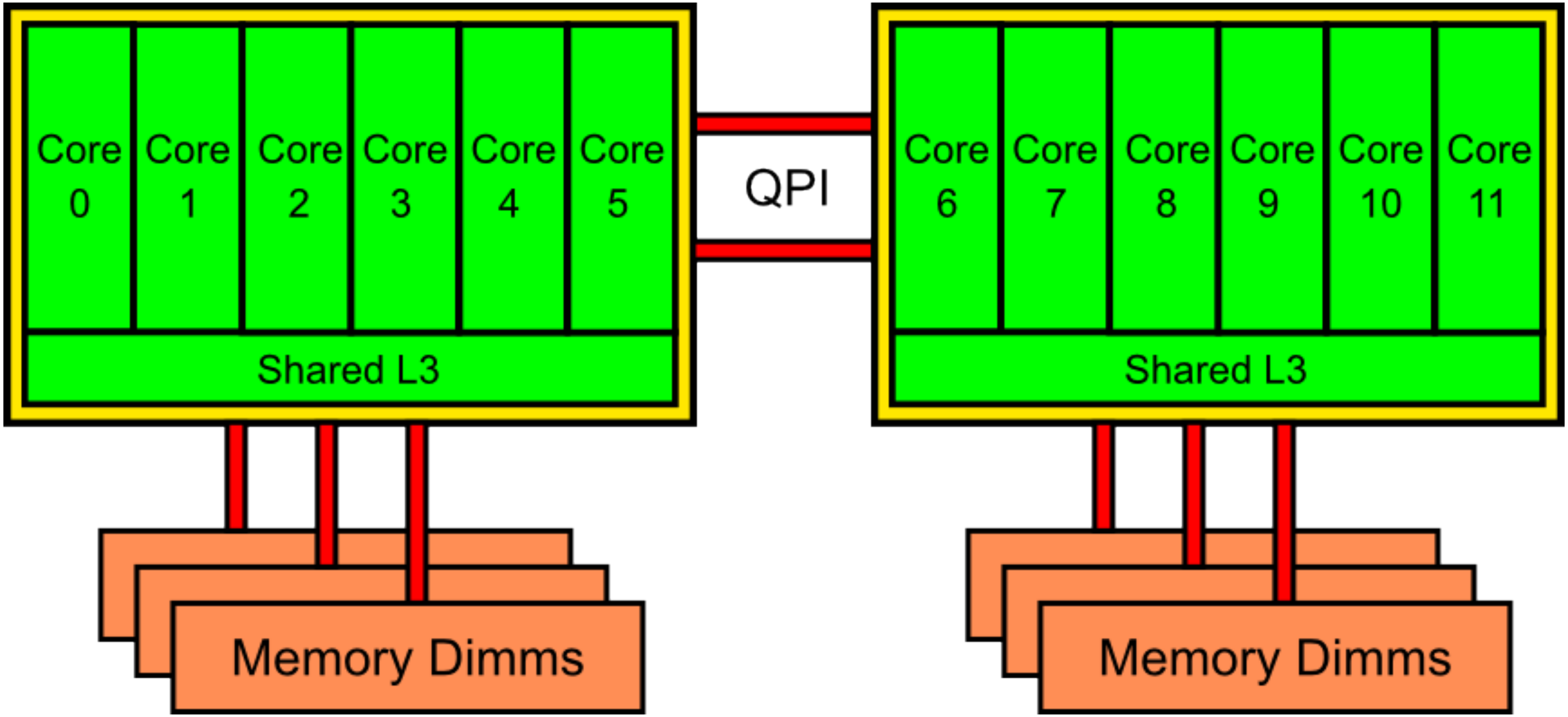
L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB)

L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB)

Core P#0 PU P#0 | Core P#1 PU P#1 | Core P#2 PU P#2 | Core P#3 PU P#3 | Core P#4 PU P#4 | Core P#5 PU P#5 | Core P#6 PU P#6 | Core P#7 PU P#7

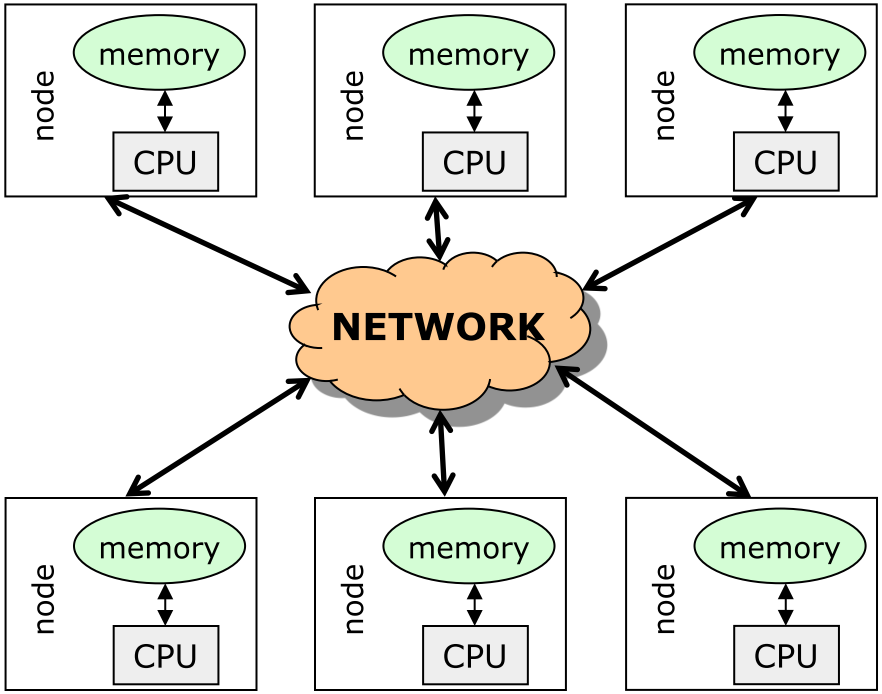
The Intel Xeon E5-2665 Sandy Bridge-EP 2.4GHz

Multiple Socket CPUs

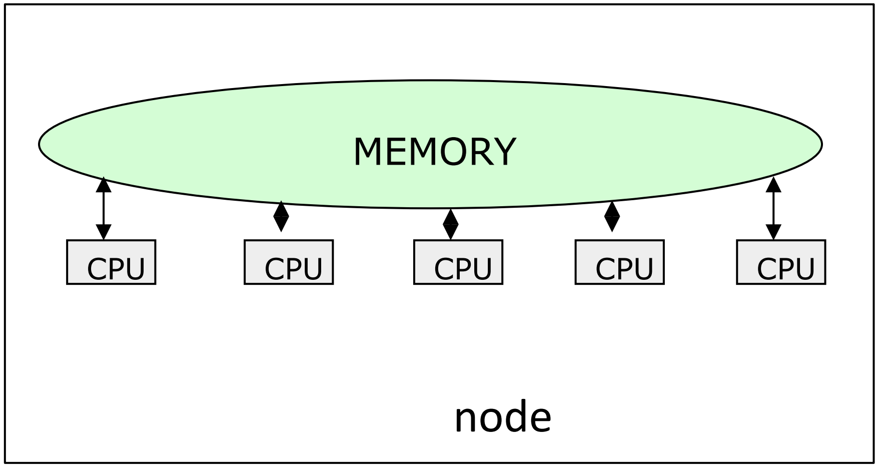


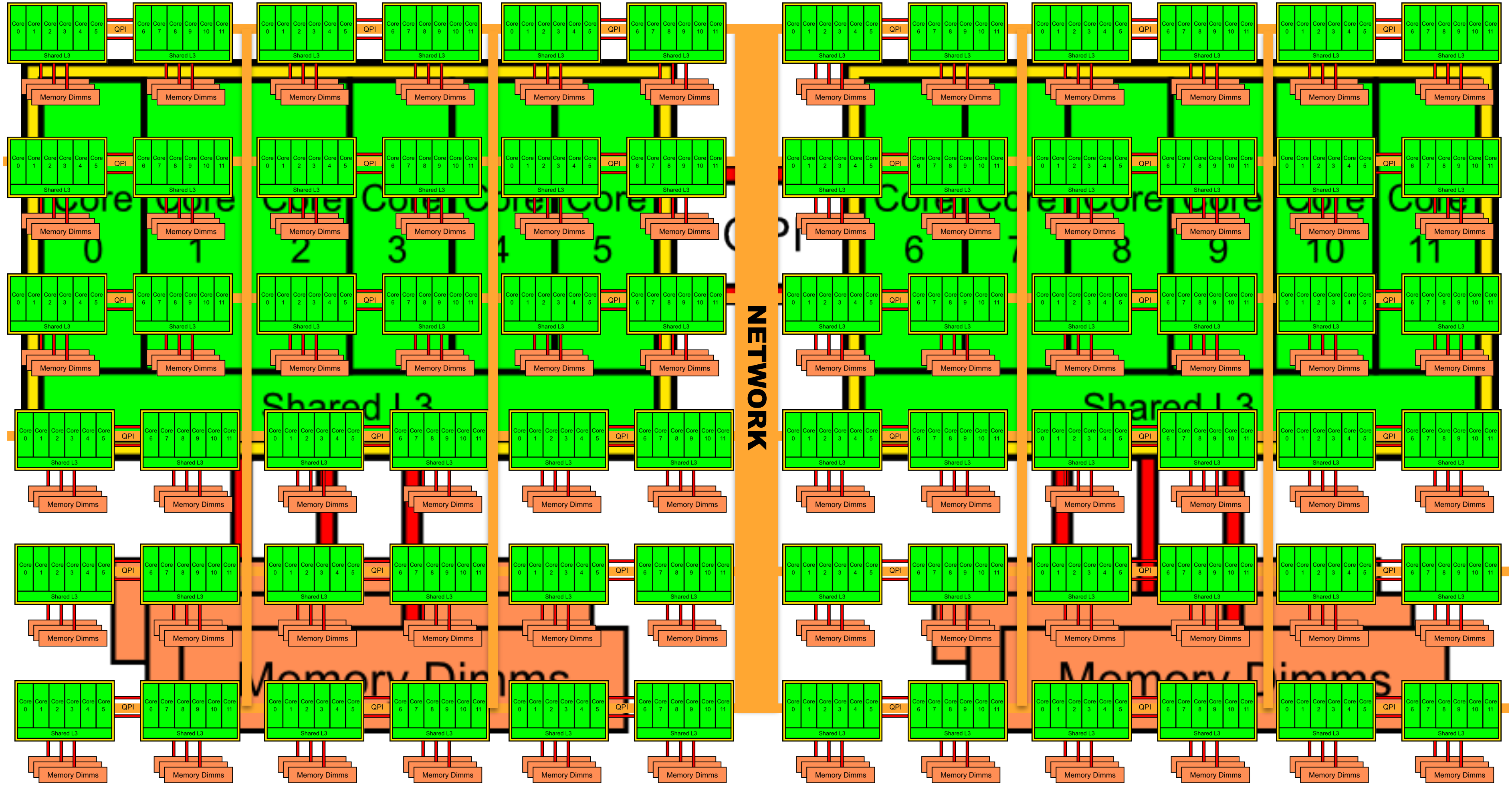
Parallel Architectures

- Distributed Memory

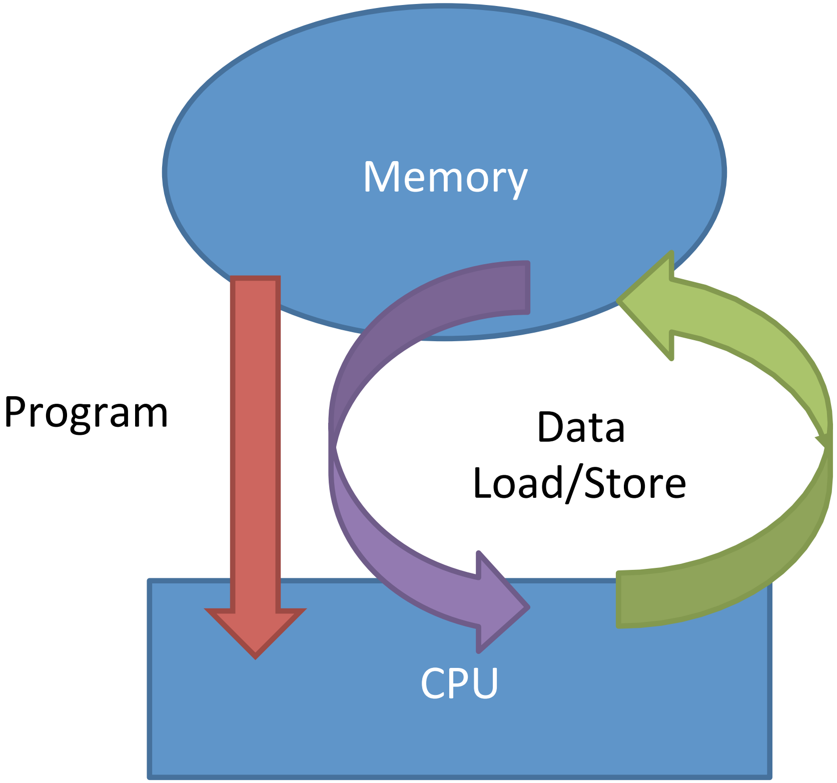


- Shared Memory

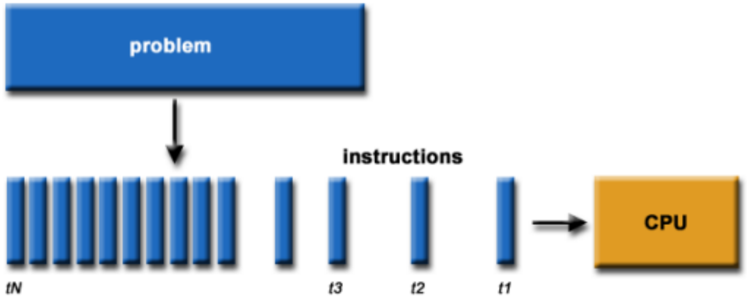



Core 0
Core 1
Core 2
Core 3
Core 4
Core 5
Core 6
Core 7
Core 8
Core 9
Core 10
Core 11
QPI
Shared L3
Memory Dimms
NETWORK

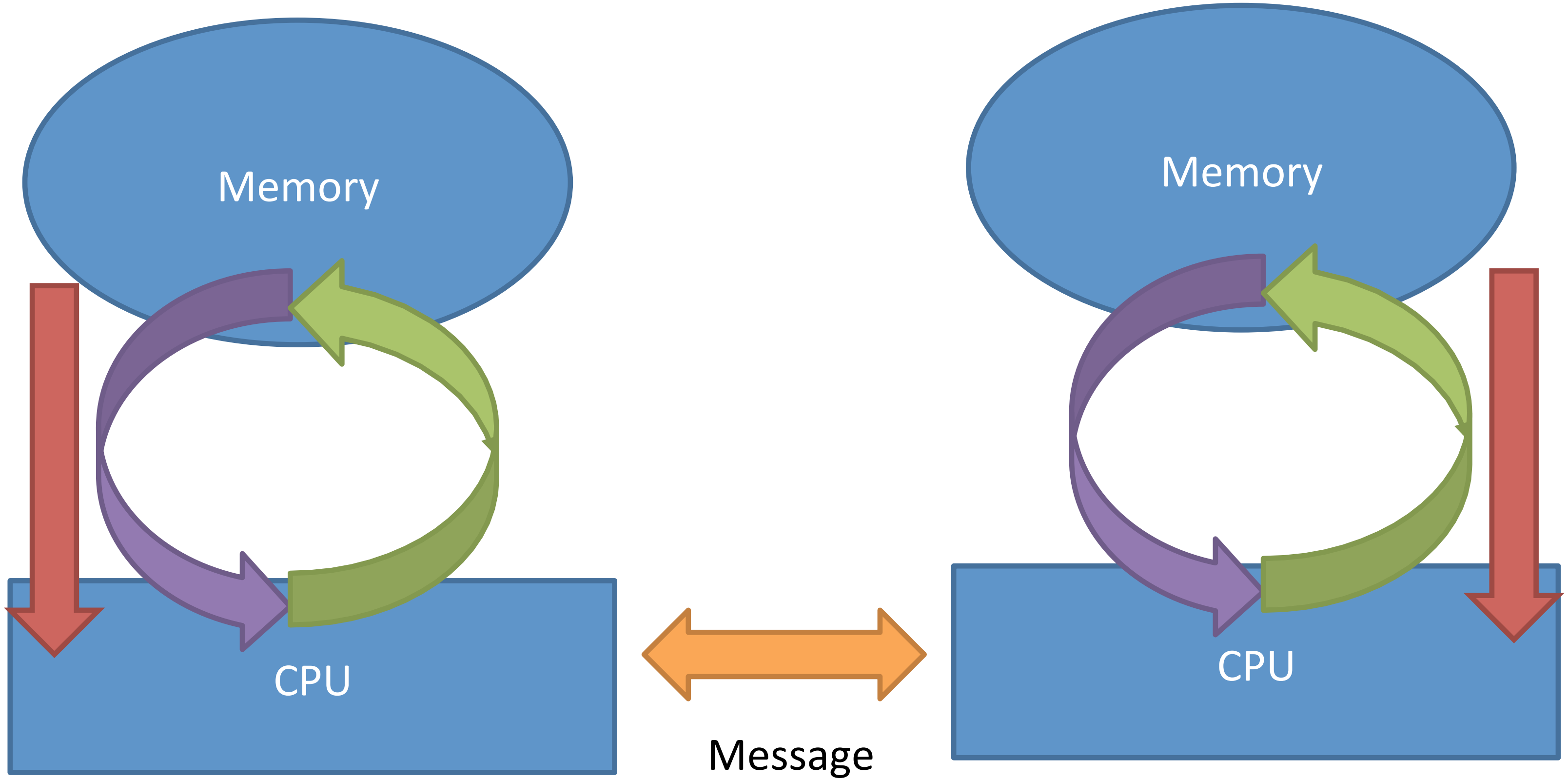
Serial Programming



A problem is broken into a discrete series of instructions.
Instructions are executed one after another.
Only one instruction may execute at any moment in time.



Parallel Programming



Logical Machine Organization

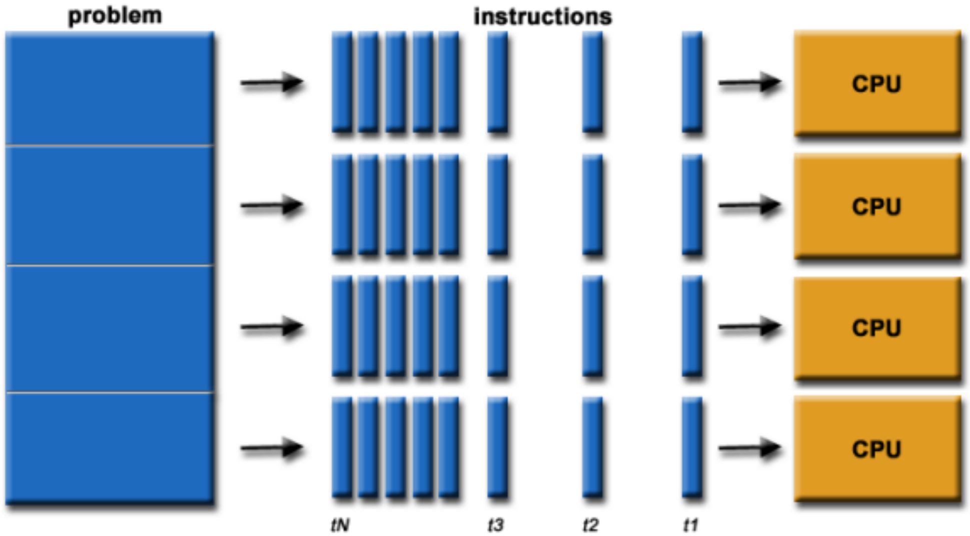
- Developers can have a different view of the hardware architecture due to the abstraction of the logical organization (how access to memory).
- It is easy to logically partition a Shared Memory system as a Distributed Memory system.
- The opposite is not trivial.

Design of Parallel Algorithm /1

- A serial algorithm is a sequence of basic steps for solving a given problem using a single serial computer
- Similarly, a parallel algorithm is a set of instruction that describe how to solve a given problem using multiple (>=1) parallel processors
- The parallelism add the dimension of concurrency. Designer must define a set of steps that can be executed simultaneously!!!

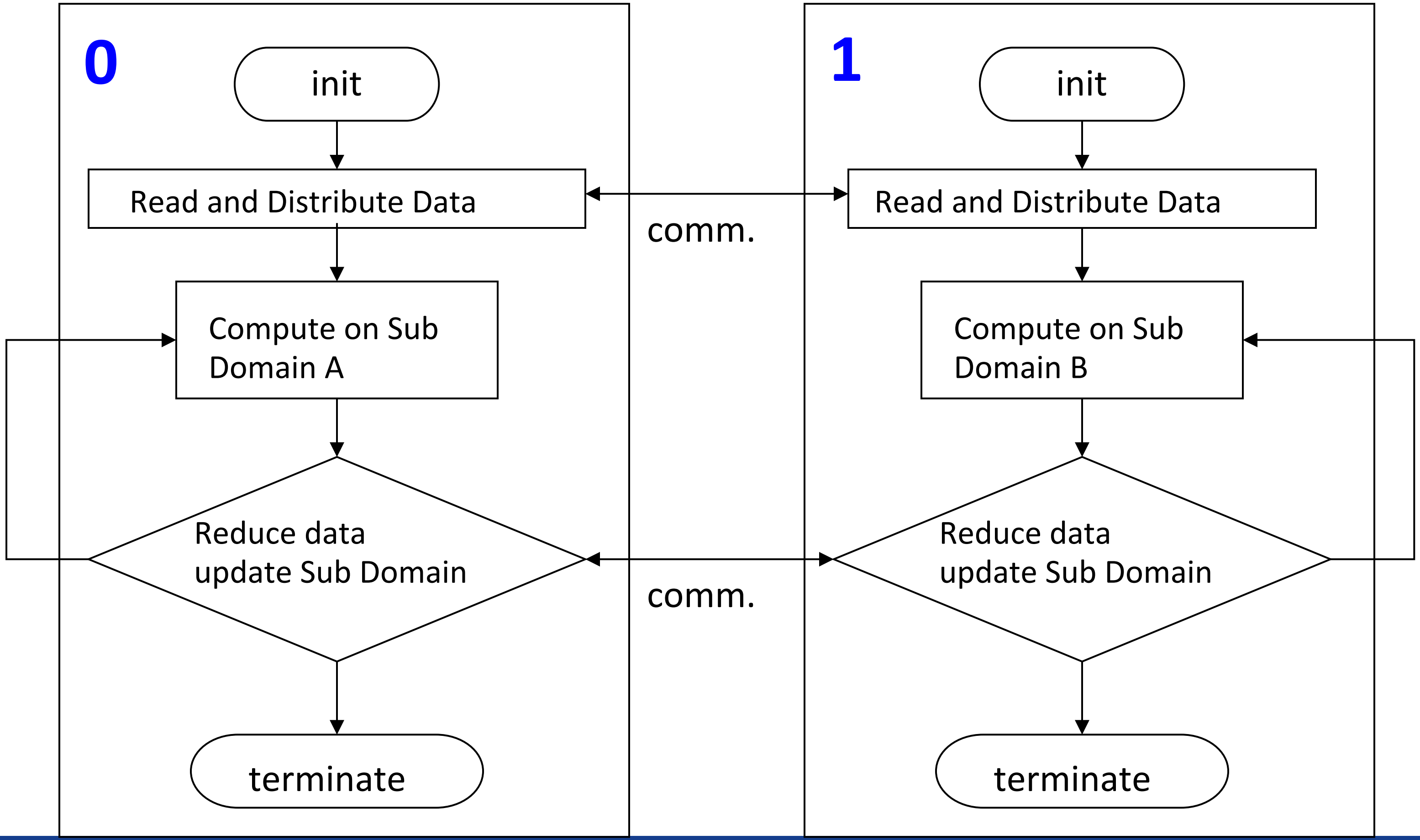
Concurrency

The first step in developing a parallel algorithm is to decompose the problem into tasks that can be executed concurrently



- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control / coordination mechanism is used

What is a Parallel Program



Design of Parallel Algorithm /2

- Identify portions of the work that can be performed concurrently
- Mapping the concurrent pieces of work onto multiple processes running in parallel
- Distributing the input, output and intermediate data associated within the program
- Managing accesses to data shared by multiple processors
- Synchronizing the processors at various stages of the parallel program execution

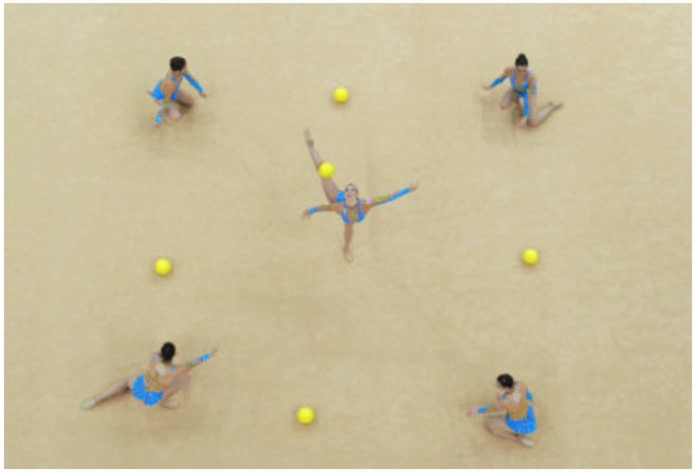
Type of Parallelism

- **Functional (or task) parallelism**: different people are performing different task at the same time

- **Data Parallelism**: different people are performing the same task, but on different equivalent and independent objects

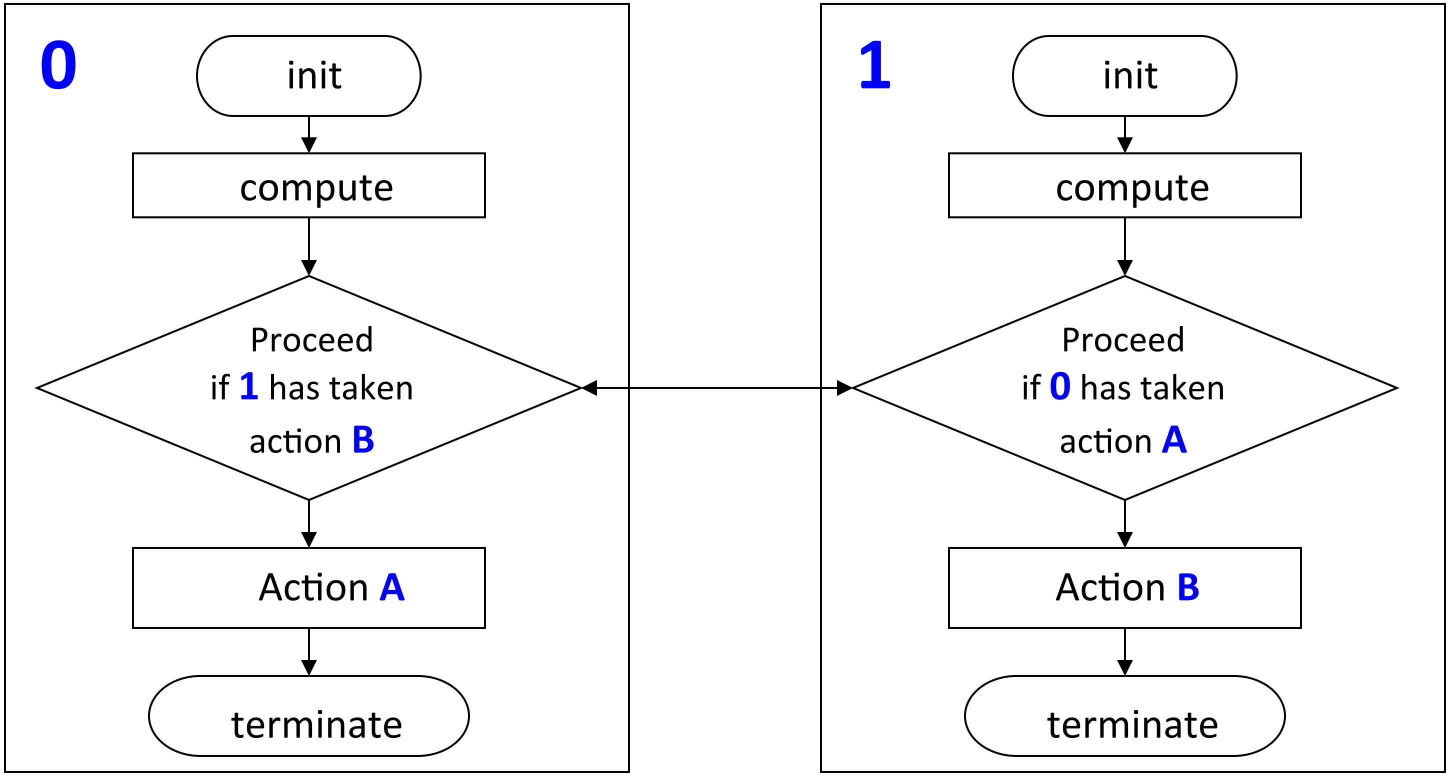
Process Interactions

- The effective speed-up obtained by the parallelization depend by the amount of overhead we introduce making the algorithm parallel
- There are mainly two key sources of overhead:
    1. Time spent in inter-process interactions (**communication**)
    2. Time some process may spent being idle (**synchronization**)

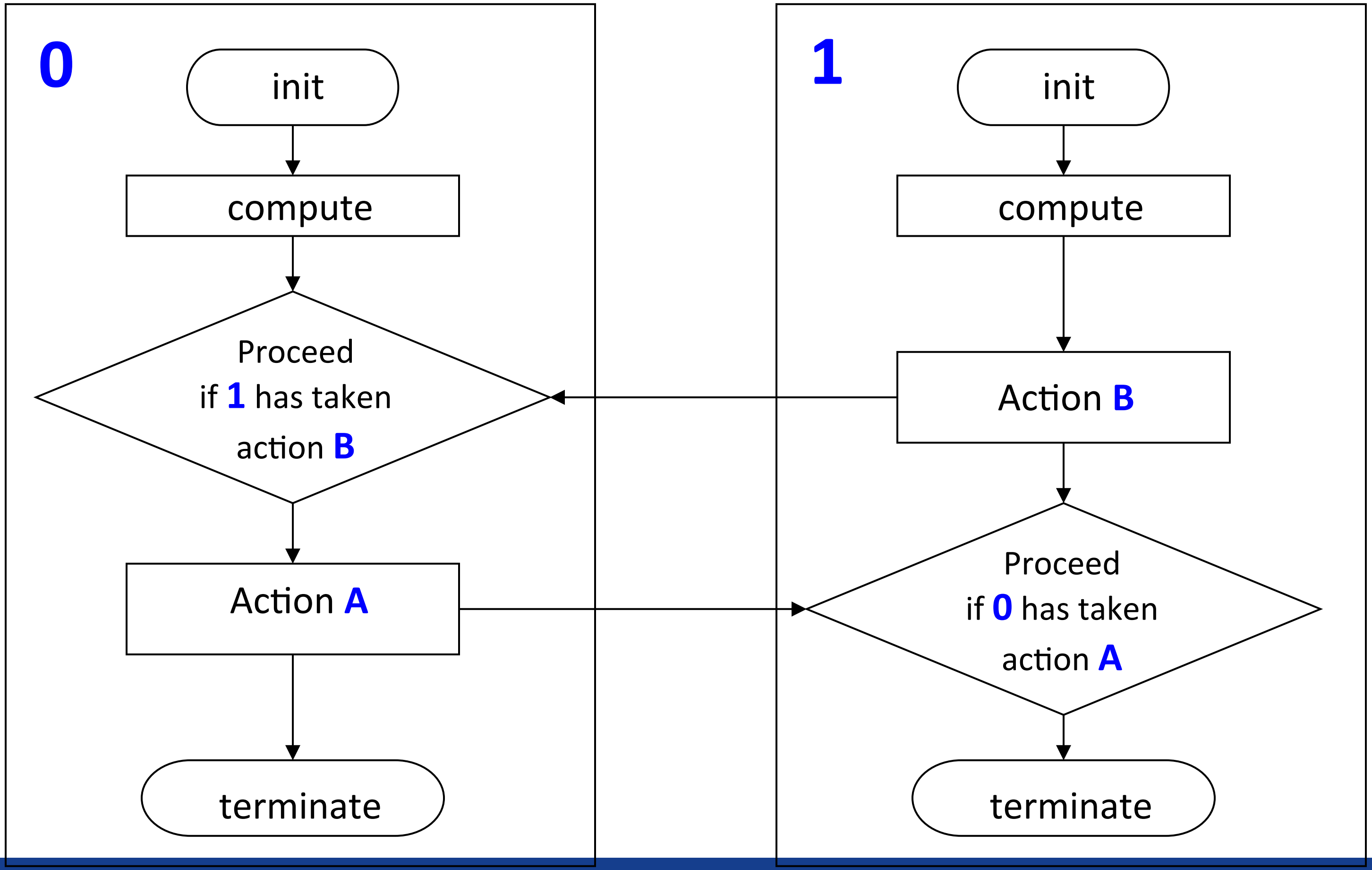


DEADLOCK

- occurs when 2^+ processes are blocked and each is waiting for the other to progress.


0
init
compute
Proceed if 1 has taken action B
Action A
terminate
1
init
compute
Proceed if 0 has taken action A
Action B
terminate

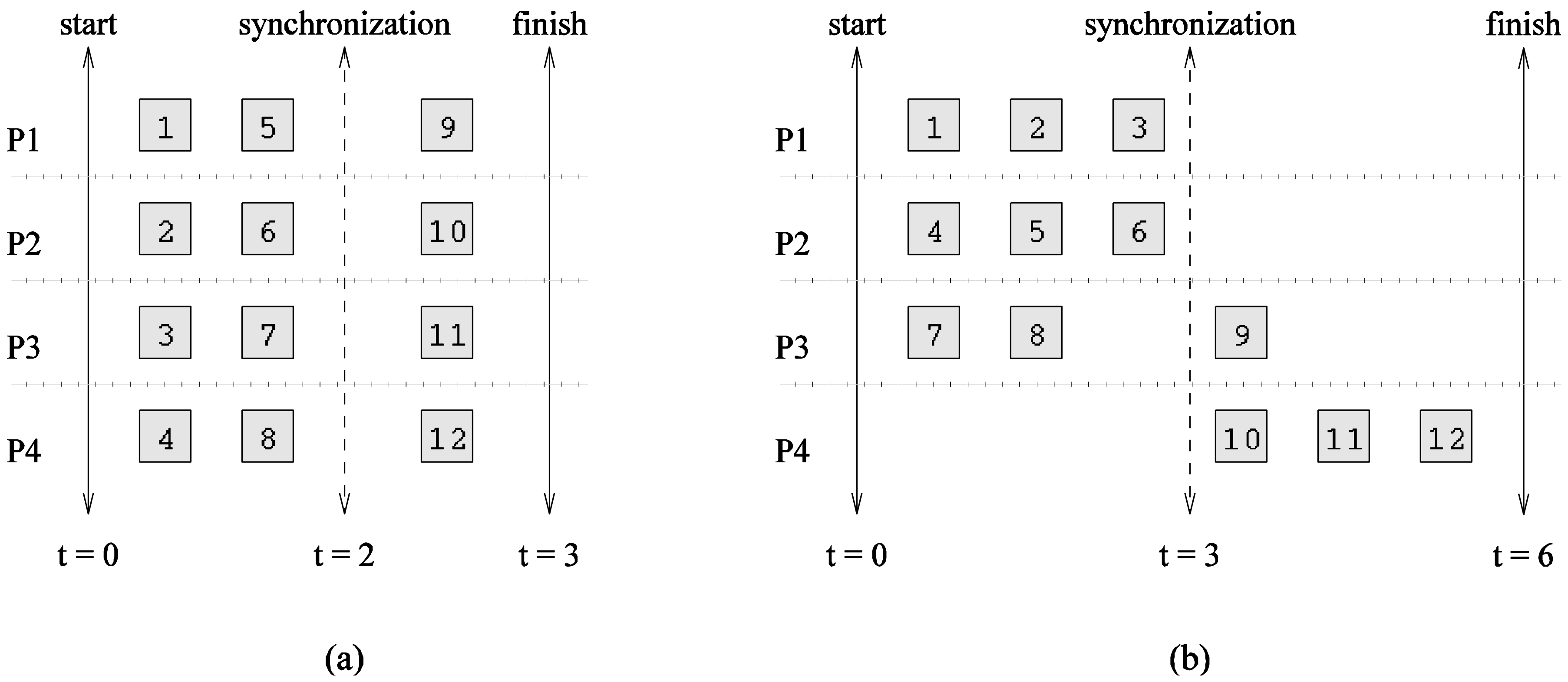
Avoiding DEADLOCK


0
init
compute
Proceed if 1 has taken action B
Action A
terminate
1
init
compute
Action B
Proceed if 0 has taken action A
terminate

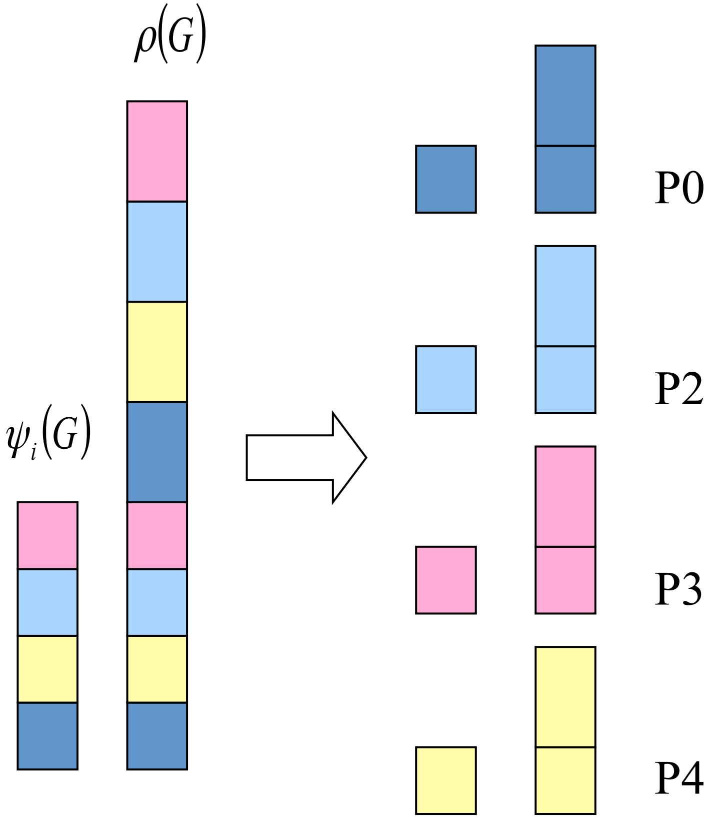
Load Balancing

- Equally divide the work among the available resource: processors, memory, network bandwidth, I/O, ...
- This is usually a simple task for the problem decomposition model
- It is a difficult task for the functional decomposition model

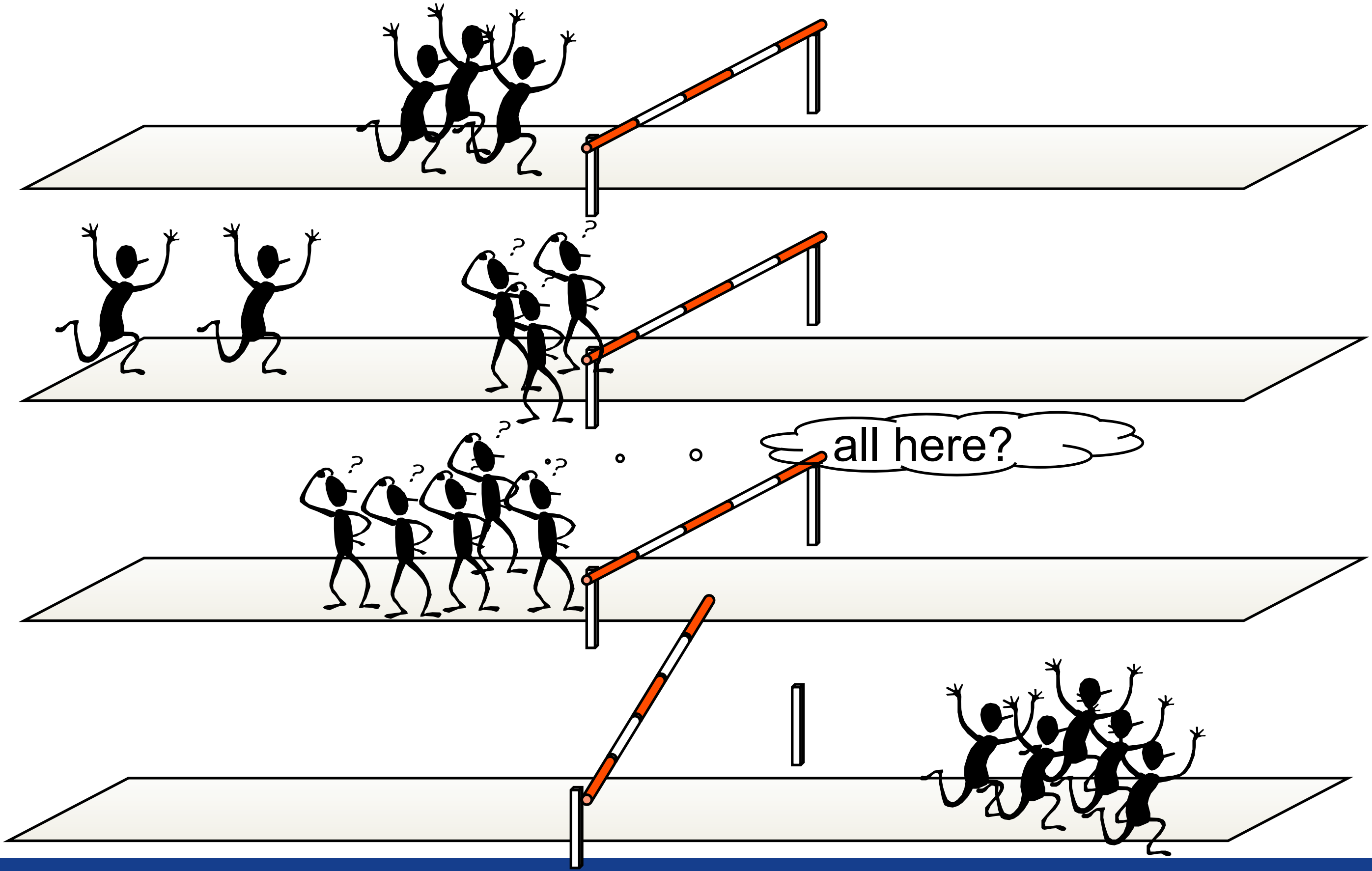
Mapping and Synchronization



Reciprocal Space distribution

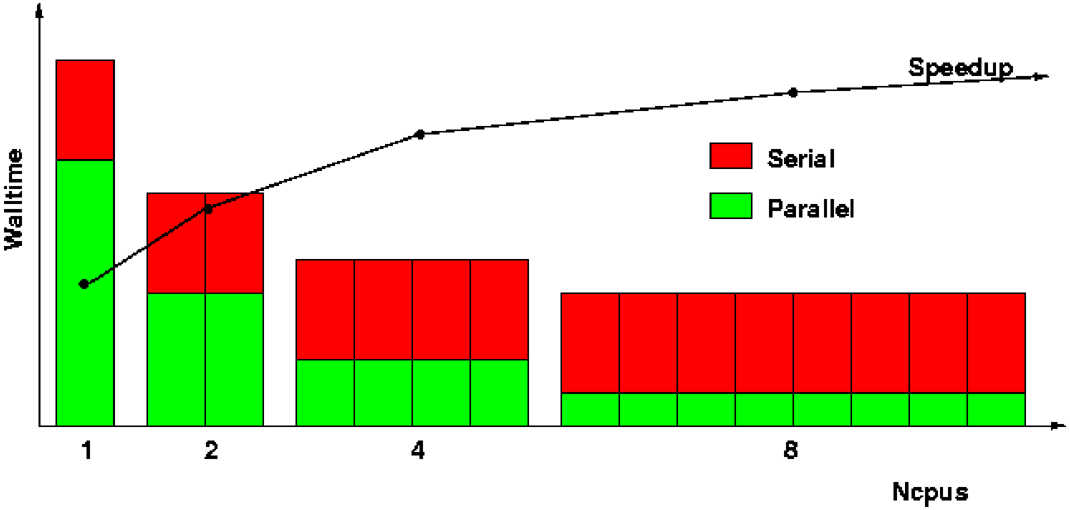


Effect of load-unbalancing



Amdahl's law

In a massively parallel context, an upper limit for the scalability of parallel applications is determined by the fraction of the overall execution time spent in non-scalable operations (Amdahl's law).



maximum speedup tends to
$1 / (1 - P)$
P= parallel fraction

1000000 core

$P = 0.999999$

serial fraction= 0.000001

Minimizing Communication

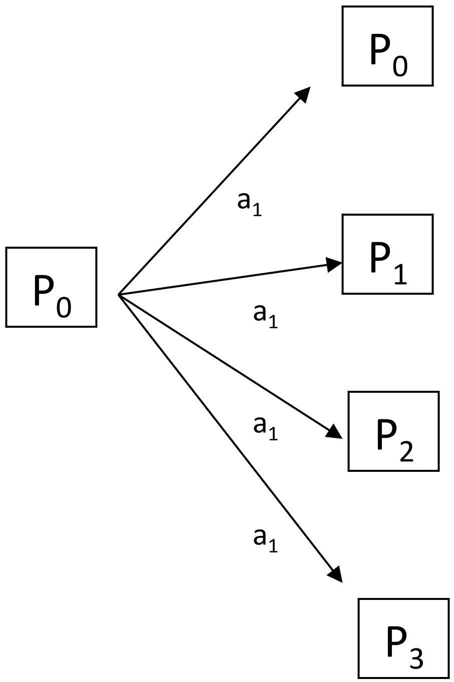
- When possible reduce the communication events:
  - Group lots of small communications into large one
  - Eliminate synchronizations as much as possible. Each synchronization level off the performance to that of the slowest process

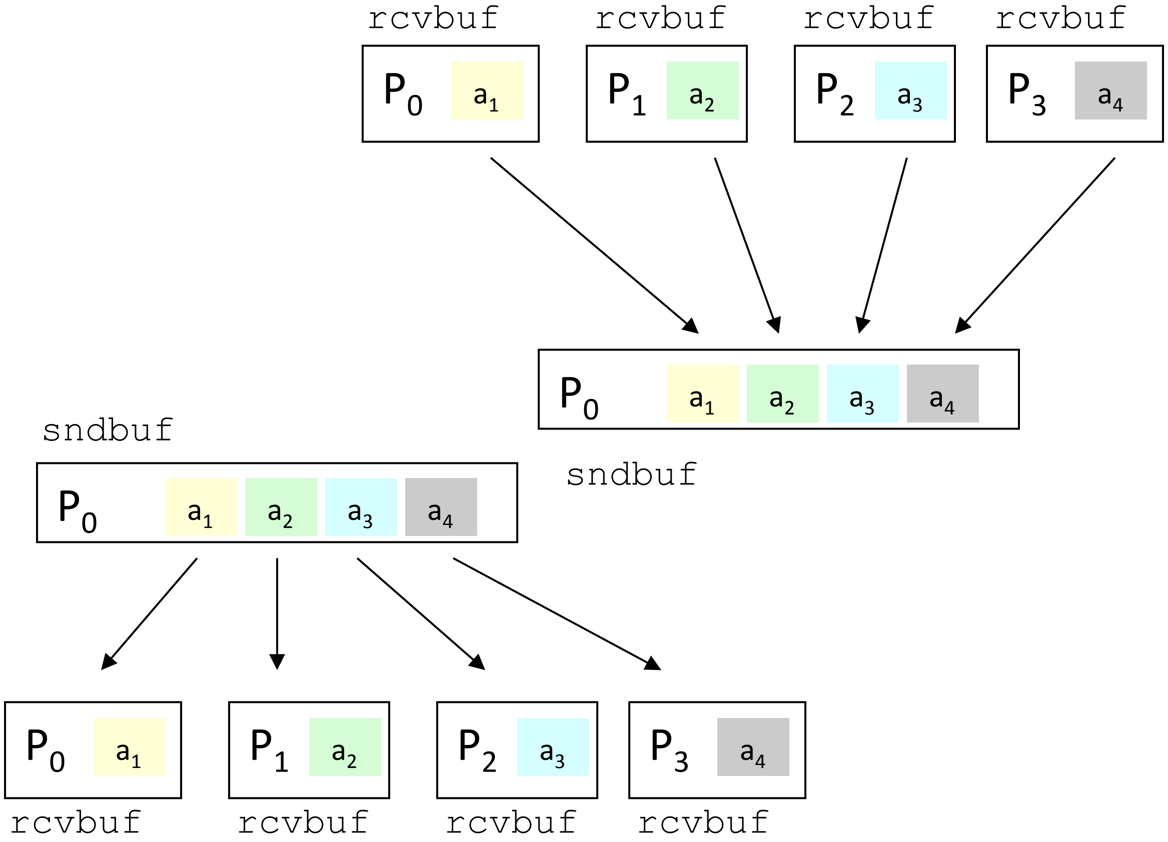
Overlap Communication and Computation

- When possible code your program in such a way that processes continue to do useful work while communicating
- This is usually a non trivial task and is afforded in the very last phase of parallelization.
- A successful implementation can bring a relevant improvement

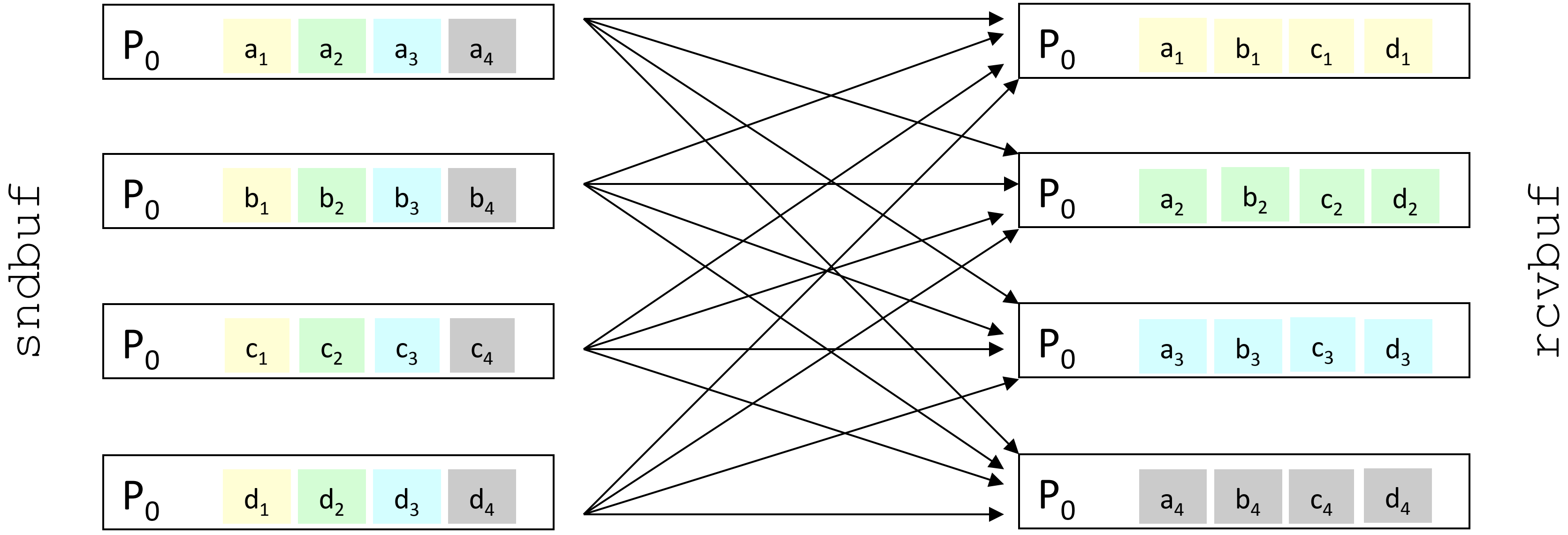
Scatter/Gather

Broadcast





All to all



sndbuf

P_0 a_1 a_2 a_3 a_4

P_0 b_1 b_2 b_3 b_4

P_0 c_1 c_2 c_3 c_4

P_0 d_1 d_2 d_3 d_4

rcvbuf

P_0 a_1 b_1 c_1 d_1

P_0 a_2 b_2 c_2 d_2

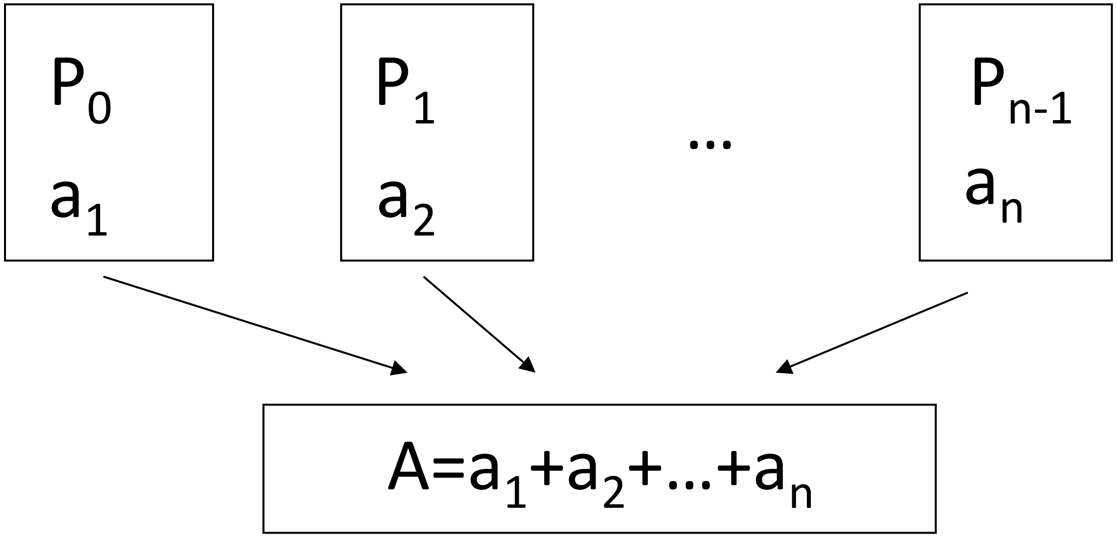
P_0 a_3 b_3 c_3 d_3

P_0 a_4 b_4 c_4 d_4

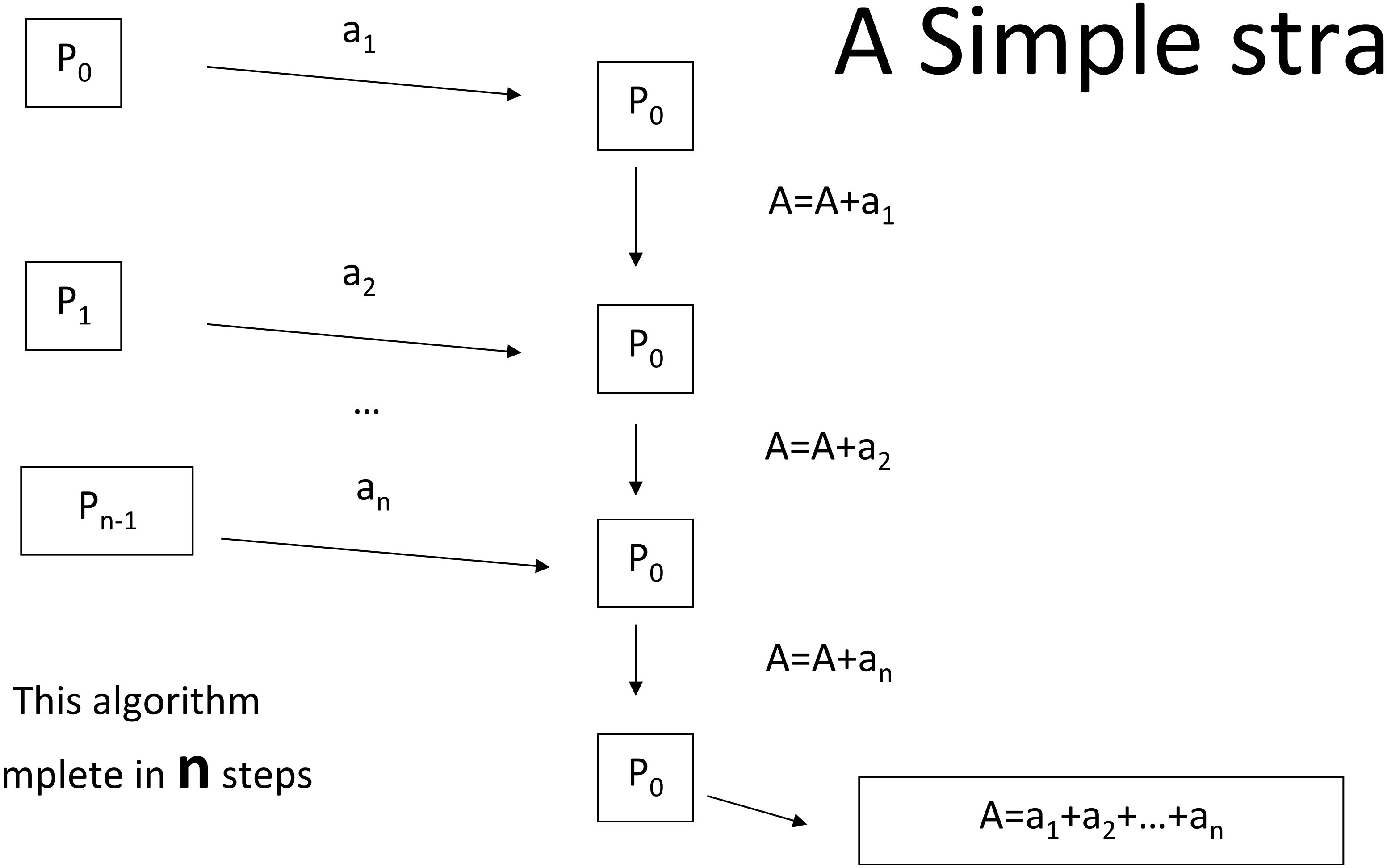
Very useful to implement data transposition

Reduction

- Reduction: sum up the partial results of different process (maybe the most common parallel operation required in a parallel program)



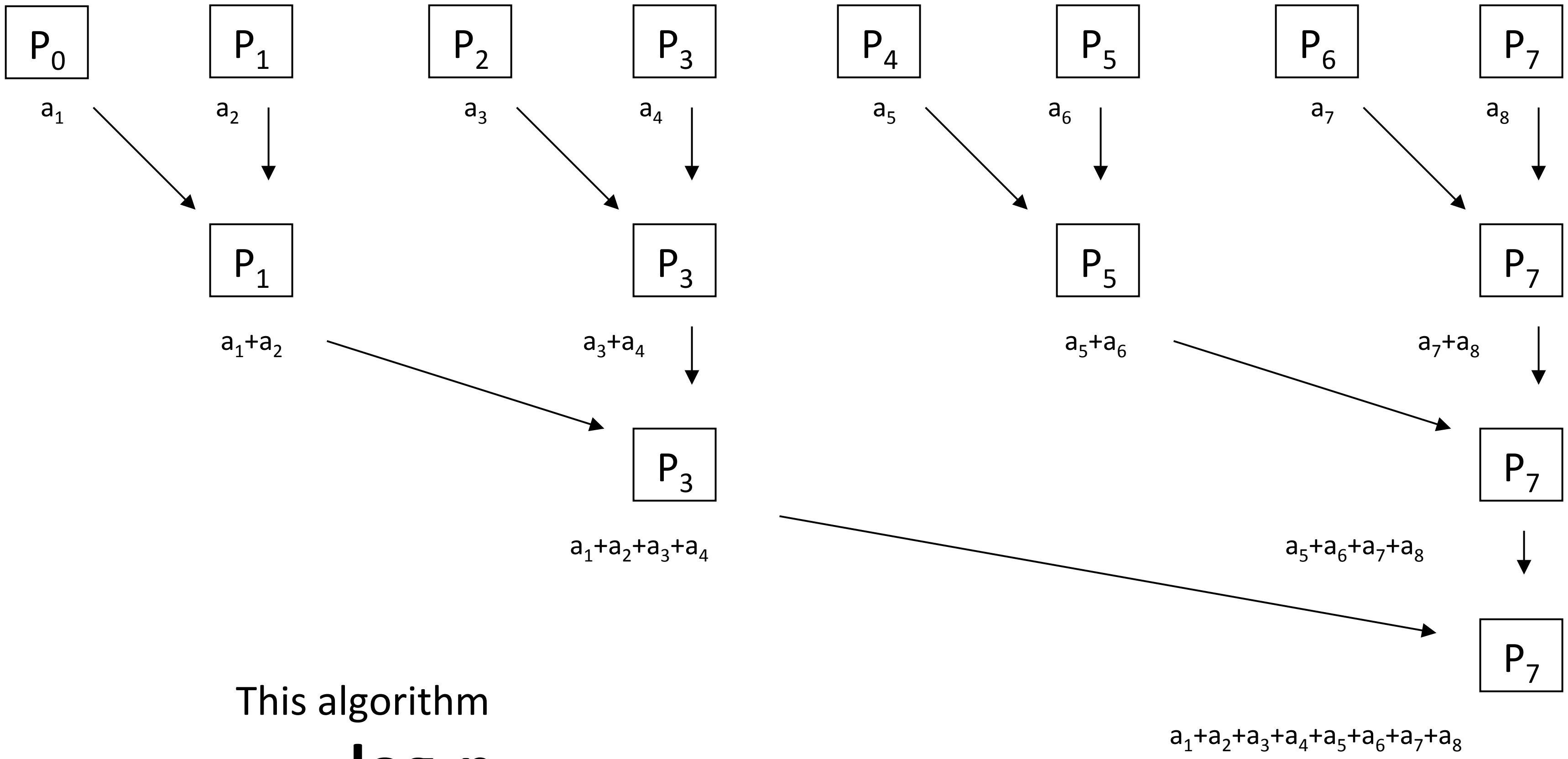
A Simple strategy



This algorithm

complete in **n** steps

Binary Tree



This algorithm

complete in $\mathbf{\log_2 n}$ steps

Binary Tree

- Finding the right partner:

	Sender	Receiver
Step 1	`MOD(myid,2)=0`	`MOD(myid,2)=1`
Step 2	`MOD(myid,4)=1`	`MOD(myid,4)=3`
Step 3	`MOD(myid,8)=3`	`MOD(myid,8)=7`
…		
Step n	`MOD(myid,2**n)=`	`MOD(myid,2**n)=`
	`2**(n-1)-1`	`2**n-1`

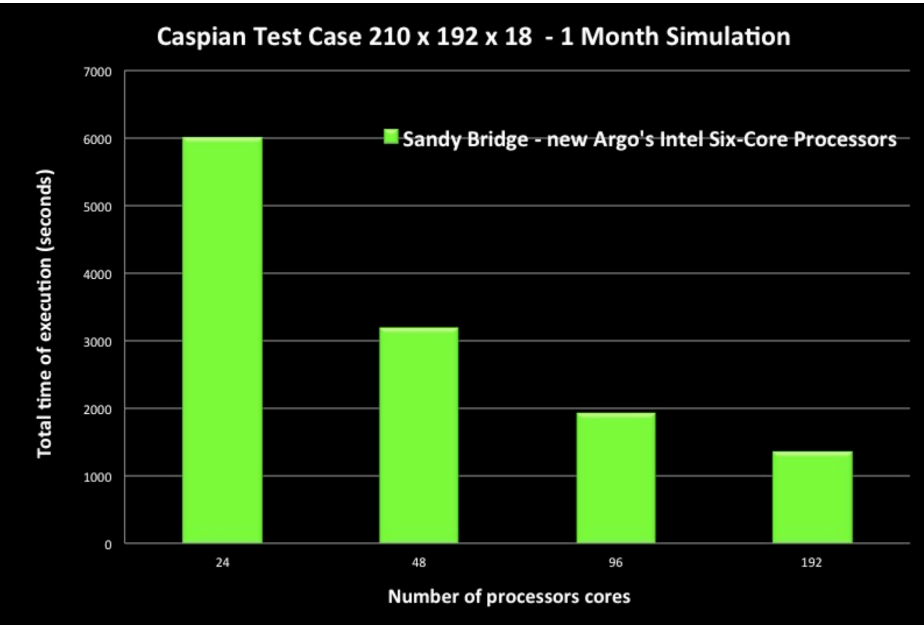
- `myid:` processor index

How do we evaluate the improvement?

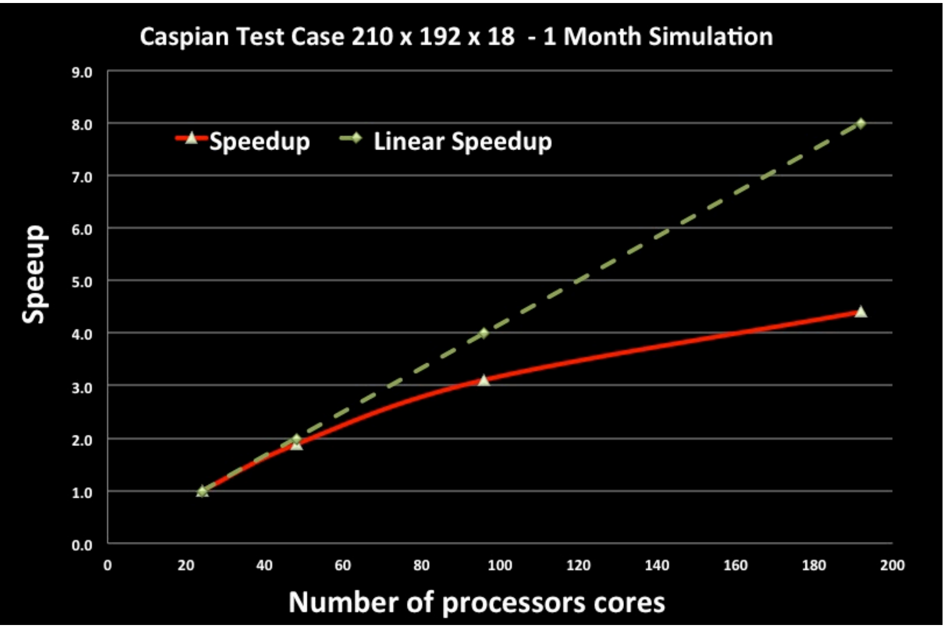
- We want estimate the amount of the introduced overhead => $T_o = n_{pes} T_P - T_S$
- But to quantify the improvement we use the term **Speedup**:

$$S_P = \frac{T_S}{T_P}$$

Speedup



Caspian Test Case 210 x 192 x 18 - 1 Month Simulation



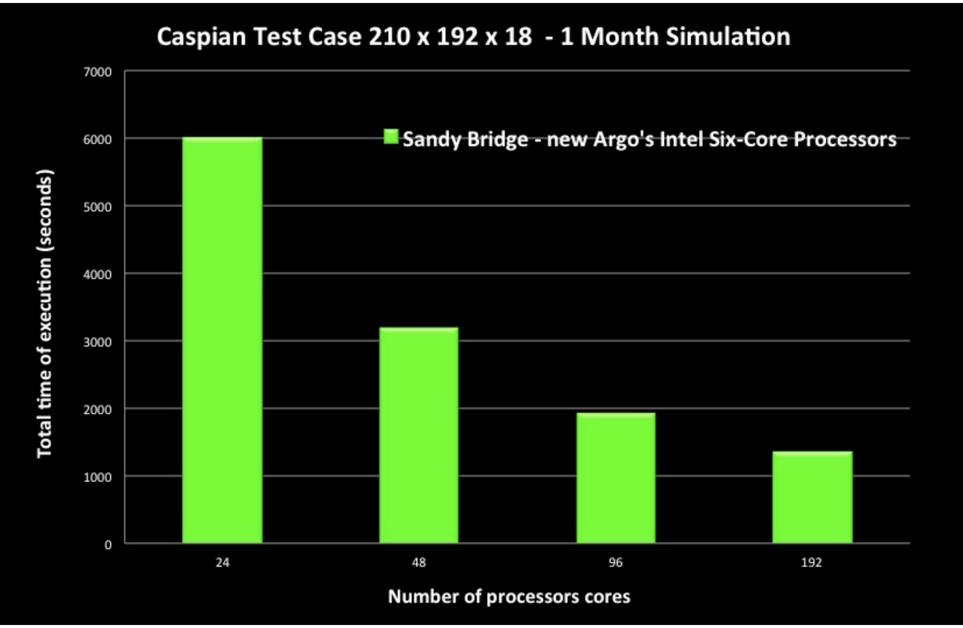
Caspian Test Case 210 x 192 x 18 - 1 Month Simulation

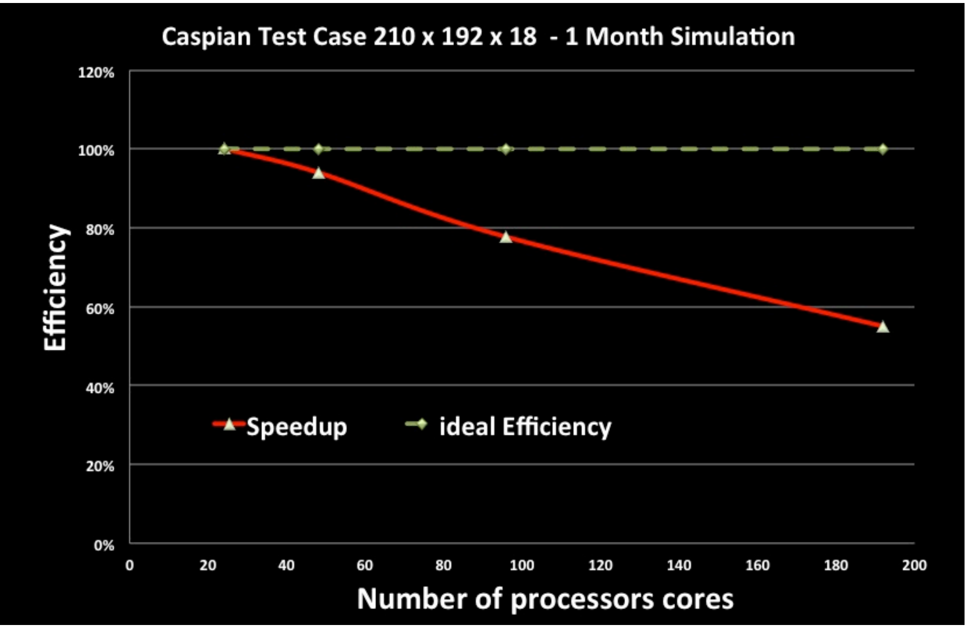
Efficiency

- Only embarrassing parallel algorithm can obtain an ideal Speedup
- The **Efficiency** is a measure of the fraction of time for which a processing element is usefully employed:

$$E_P = \frac{S_P}{p}$$

Efficiency


Caspian Test Case 210 x 192 x 18 - 1 Month Simulation
Total time of execution (seconds)
7000
6000
5000
4000
3000
2000
1000
0
Sandy Bridge - new Argo's Intel Six-Core Processors
24
48
96
192
Number of processors cores


Caspian Test Case 210 x 192 x 18 - 1 Month Simulation
Efficiency
120%
100%
80%
60%
40%
20%
0%
Speedup
ideal Efficiency
0
20
40
60
80
100
120
140
160
180
200
Number of processors cores

Thanks for your attention!!