The Abdus Salam
International Centre
for Theoretical Physics

United Nations
Educational, Scientific and
Cultural Organization

IAEA
International Atomic Energy Agency

2494-3

Workshop on High Performance Computing (HPC) Architecture and Applications in the ICTP

14 - 25 October 2013

Introduction to SW parallelization

Ivan Girotto
ICTP, Trieste

The Abdus Salam
International Centre
for Theoretical Physics

Introduction to SW parallelization

Ivan Girotto – igirotto@ictp.it

Information & Communication Technology Section (ICTS)

International Centre for Theoretical Physics (ICTP)

Outline

- Principles of Parallelism
- Overview of the Programming Paradigms
- The software environment
- Conclusions

Design of Parallel Algorithm

- 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

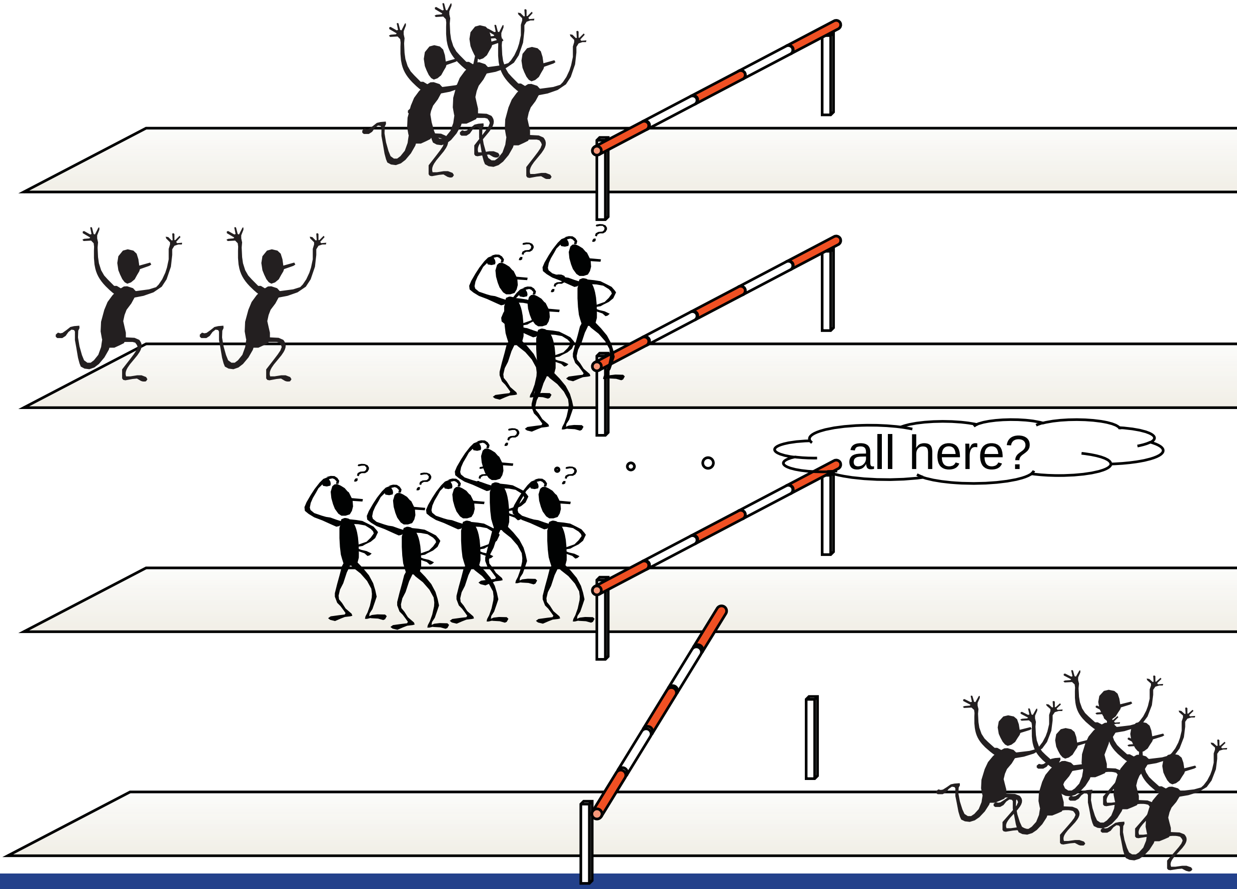
- **Functional (or task) parallelism**: different people are performing different task at the same time

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

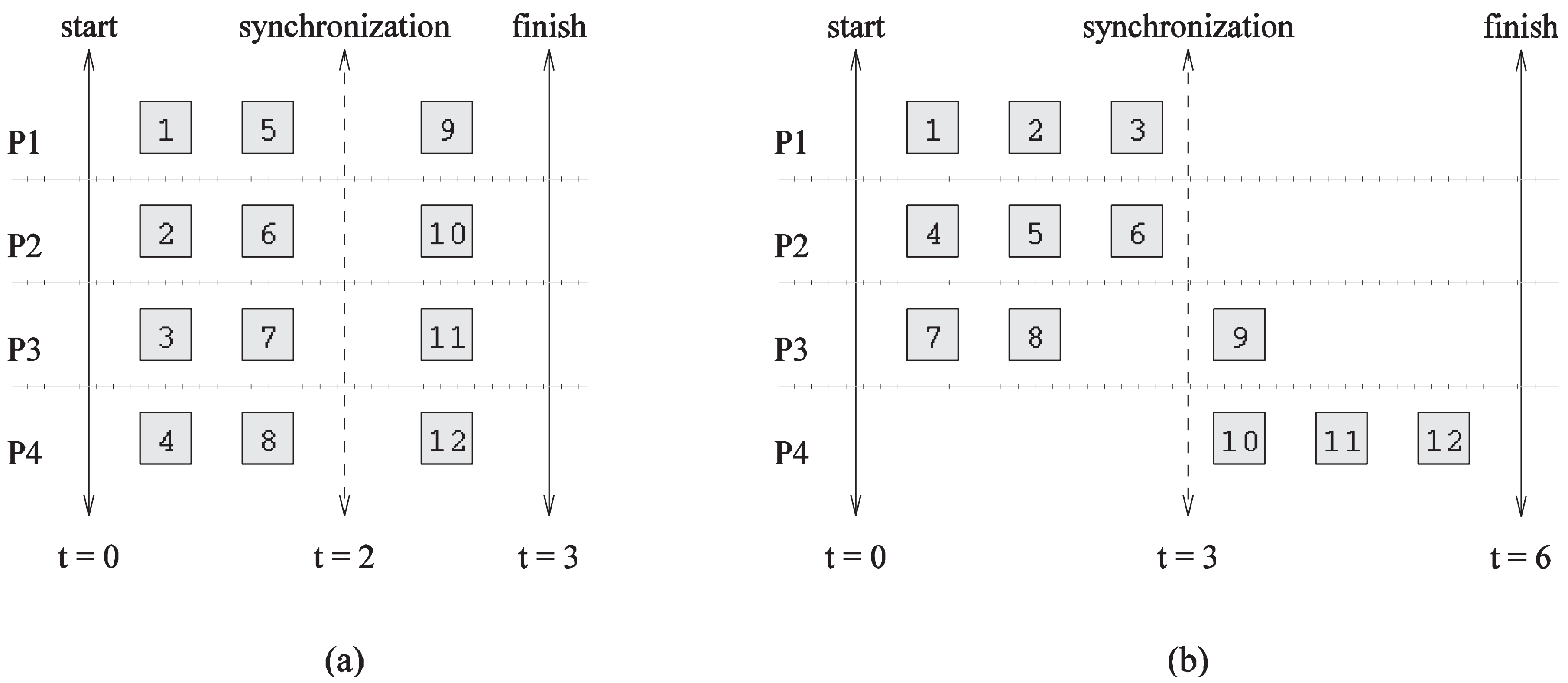
Process Interactions /1

- 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**)

What happens if someone is left behind?



Mapping and Synchronization



Granularity

- Granularity is determined by the decomposition level (number of task) on which we want divide the problem

- The degree to which task/data can be subdivided is limit to concurrency and parallel execution

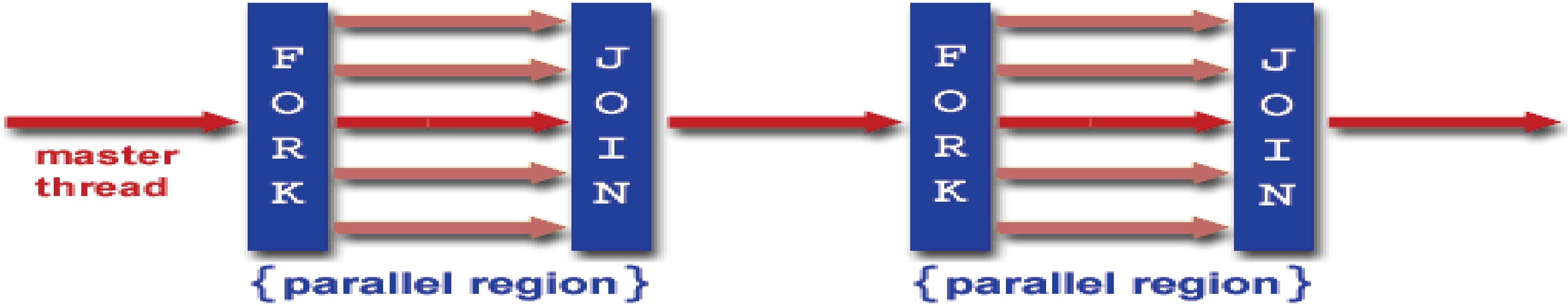
- Parallelization has to become "topology aware"
  - coarse grain and fine grained parallelization has to be mapped to the topology to reduce memory and I/O contention

Programming Parallel Paradigms

- Are the tools we use to express the parallelism for on a given architecture
- They differ in how programmers can manage and define key features like:
  - parallel regions
  - concurrency
  - process communication
  - synchronism

OpenMP (*Open spec. for Multi Processing*)

- OpenMP is not a computer language
  - Rather it works in conjunction with existing languages such as standard Fortran or C/C++
- Application Programming Interface (API)
  - that provides a portable model for shared memory // applications.
  - Three main components:
    - Compiler directives
    - Runtime library routines
    - Environment variables
- Three main advantages:
  - Incremental parallelization, Ease of use, Standardised



- Thread-based Parallelism
- Explicit Parallelism
- Fork-Join Model
- Compiler Directive Based
- Dynamic Threads

*Source: http://www.llnl.gov/computing/tutorials/openMP/#ProgrammingModel

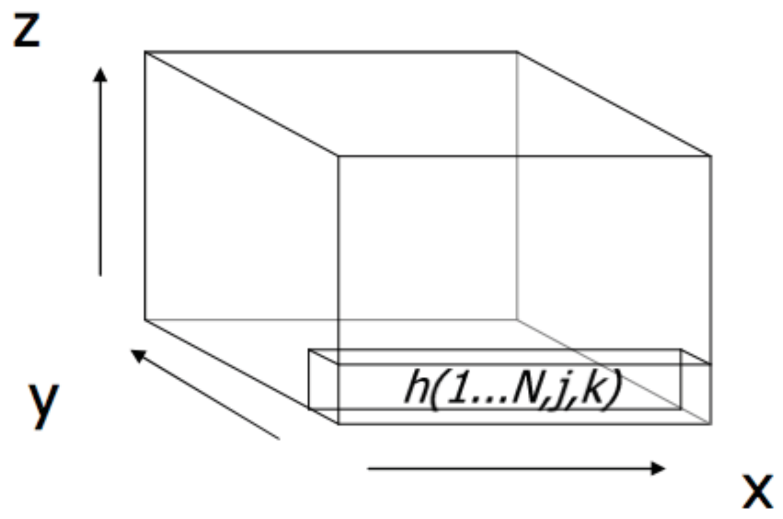
The Message Passing Interface (MPI)

- Programming tool (library) based on the concept of messages
- Driver for parallel execution of independent processes
- Standard defined by the mpi-forum.org of which is possible to find different implementations

Task Farming

- Many independent programs (tasks) running at once
  - each task can be serial or parallel
  - “independent” means they don’t communicate directly
- Common approach for using cycles in a loosely-connected cluster
  - how does it relate to HPCx and Capability Computing?
- Often needed for pre or post-processing
- Tasks may contribute to a single, larger calculation
  - parameter searches or optimisation
  - enhanced statistical sampling
  - ensemble modelling

Multidimensional FFT



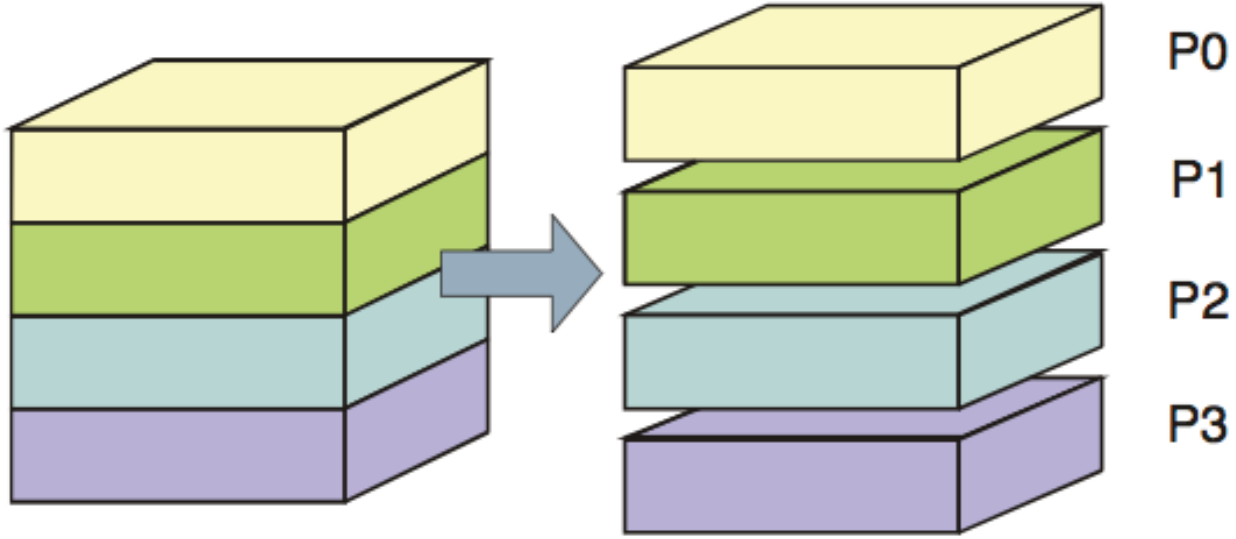
$$f(x,y,z) = \frac{1}{N_z N_y N_x} \underbrace{\sum_{z=0}^{N_z-1} (\underbrace{\sum_{y=0}^{N_y-1} (\underbrace{\sum_{x=0}^{N_x-1} F(u,v,w)\, e^{-2\pi i \frac{xu}{N_x}})}_{\text{DFT long x-dimension}}\, e^{-2\pi i \frac{yv}{N_y}})}_{\text{DFT long y-dimension}}\, e^{-2\pi i \frac{zw}{N_z}}}_{\text{DFT long z-dimension}}$$

1) For any value of ***j*** and ***k*** transform the column (1…N, j, k)

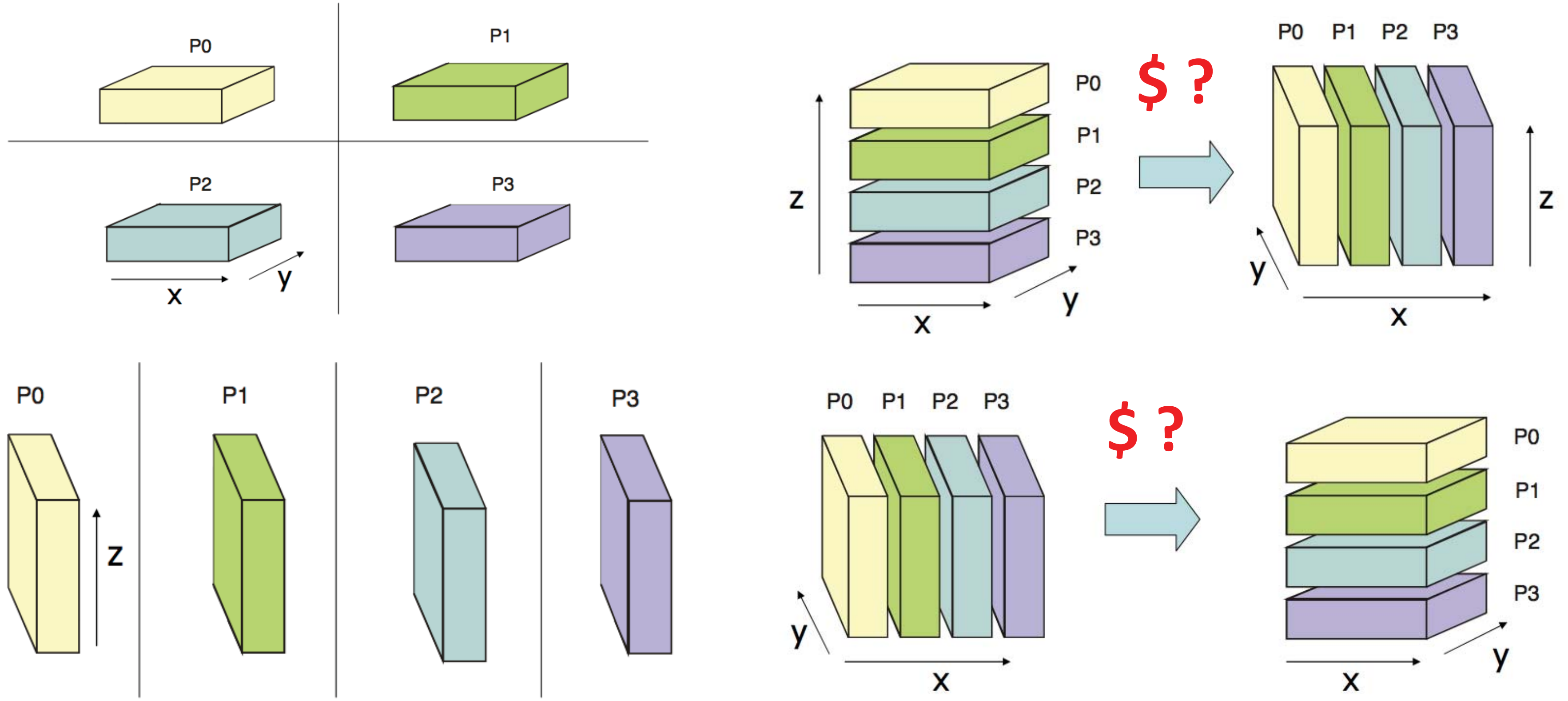
2) For any value of ***i*** and ***k*** transform the column (i, 1…N, k)

3) For any value of ***i*** and ***j*** transform the column (i, j, 1…N)

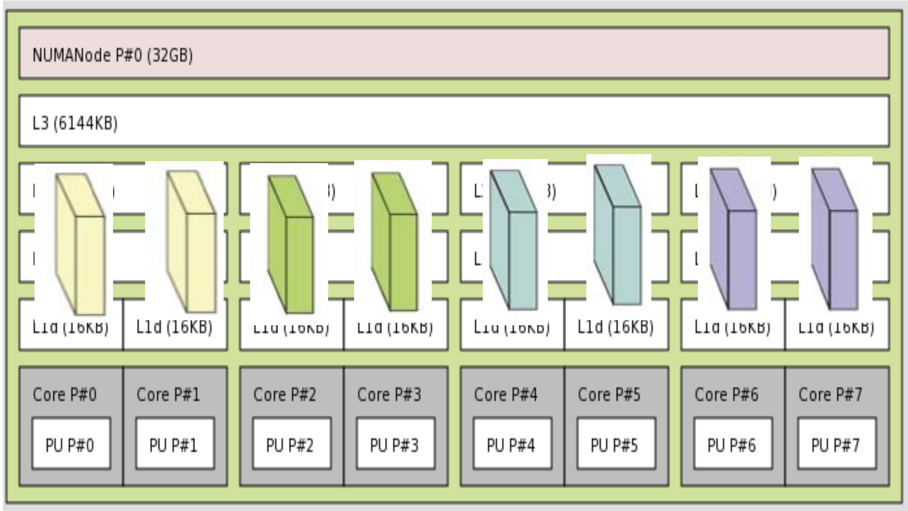
Parallel 3DFFT / 1



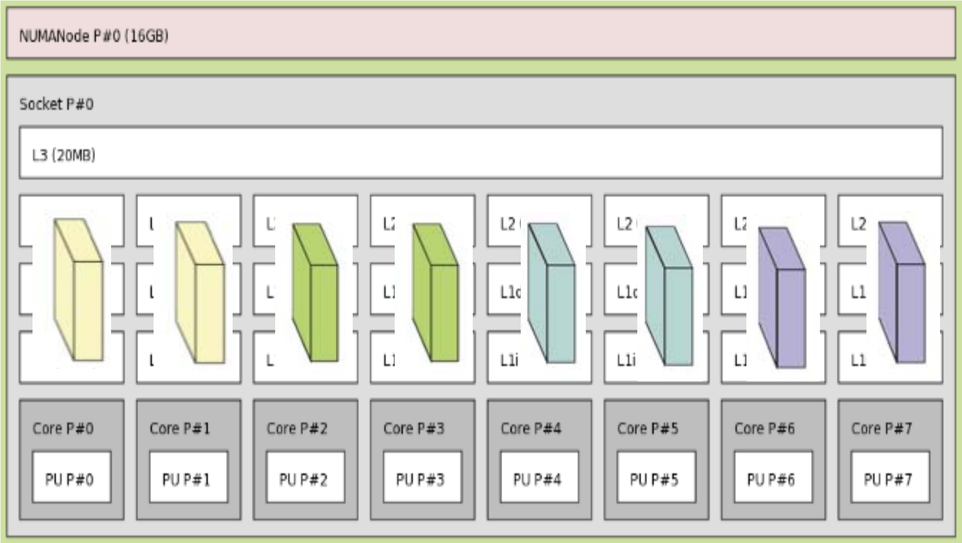
Parallel 3DFFT / 2



Parallel 3DFFT on Multicore CPUs

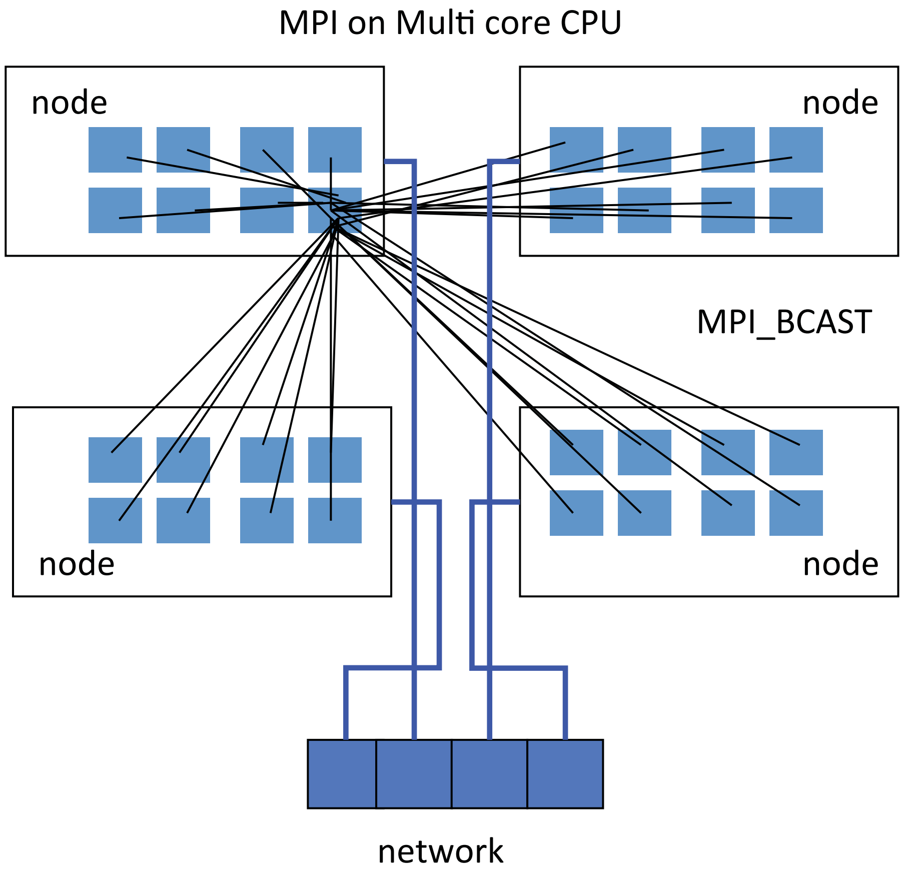


The AMD Opteron 6380 Abu Dhabi 2.5GHz



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

MPI inter process communications



1 MPI proces / core
Stress network
Stress OS

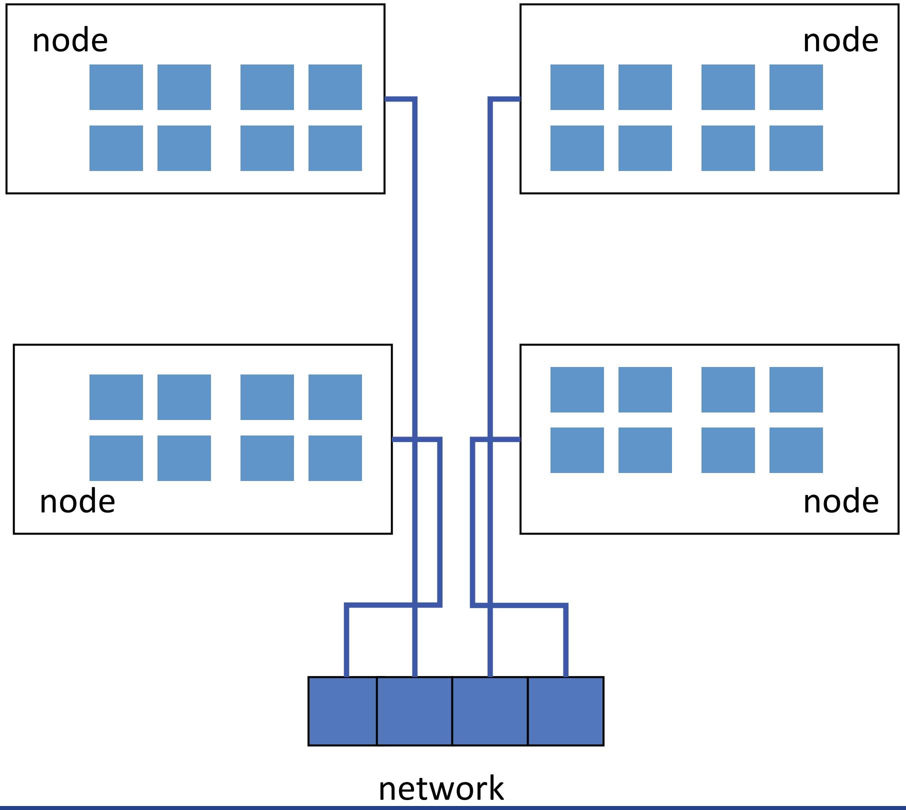
Many MPI codes (QE) based on
ALLTOALL
Messages = processes * processes

We need to exploit the hierarchy

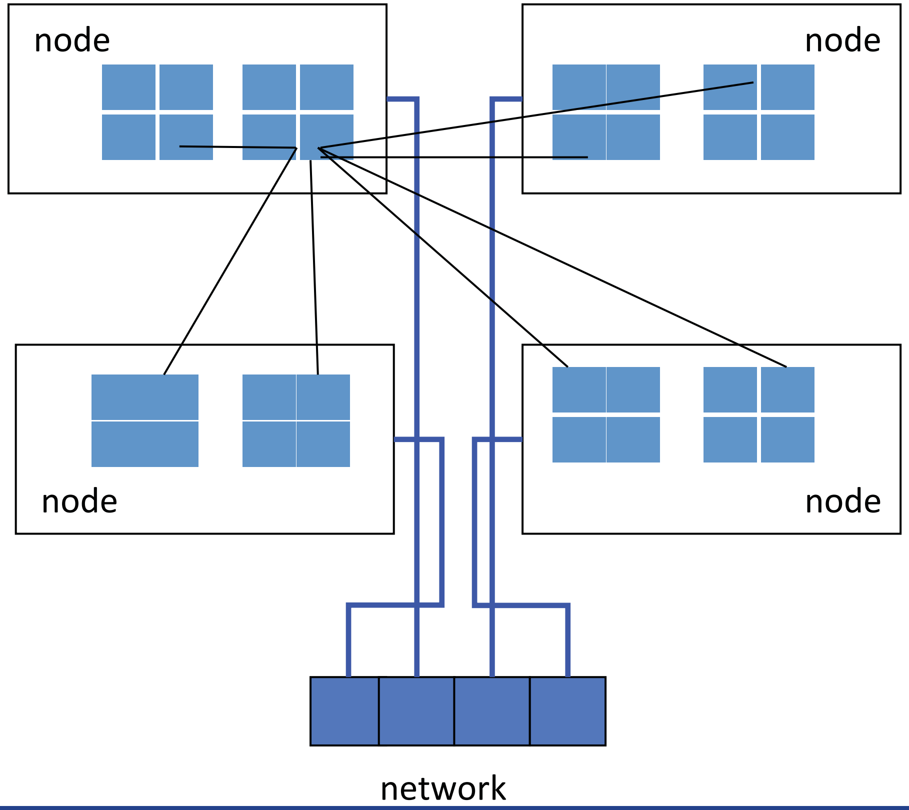
Re-design applications

Mix message passing And multi-threading

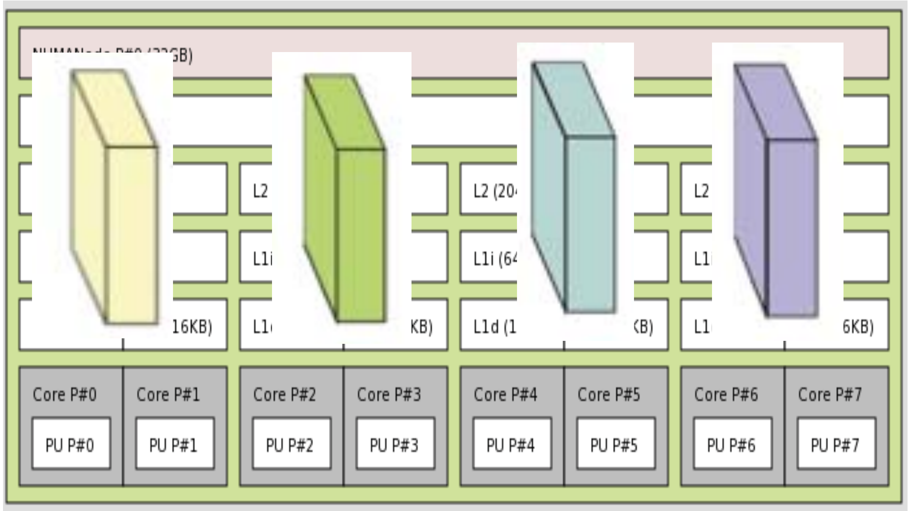
The Hybrid Mode



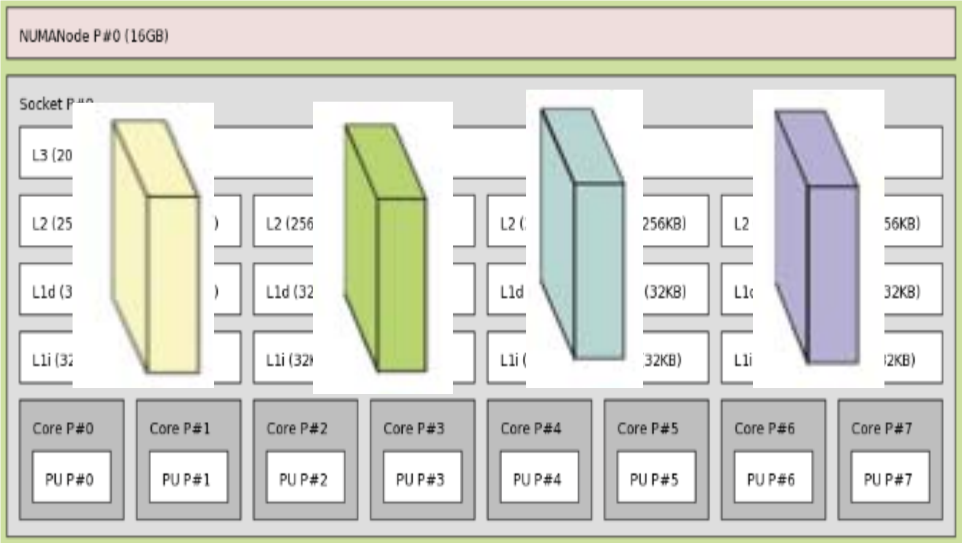
The Hybrid Mode



Parallel 3DFFT on Multicore CPUs



The AMD Opteron 6380 Abu Dhabi 2.5GHz



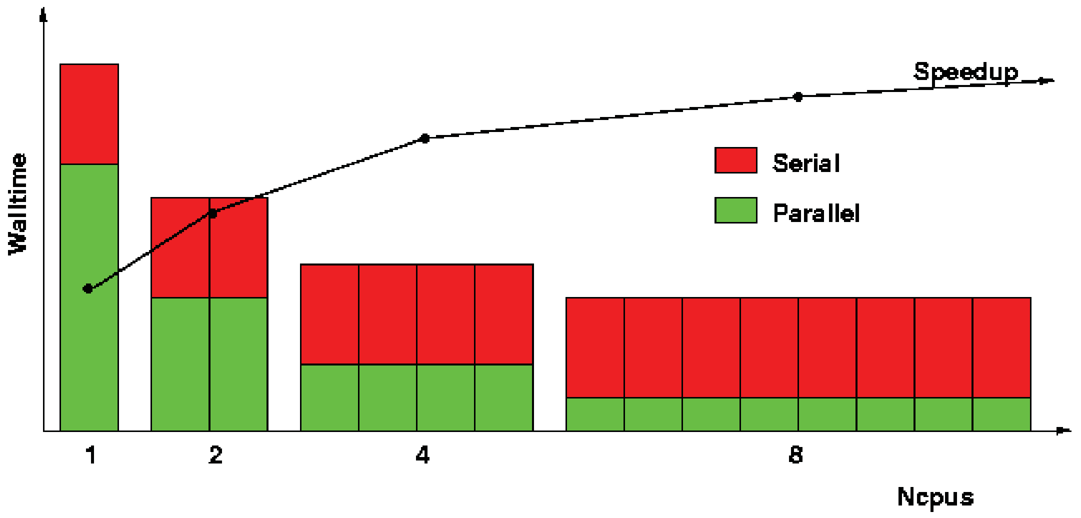
The Intel Xeon E5-2665
Sandy Bridge-EP 2.4GHz

OpenCL

- *O*pen *C*ompute *L*anguage
- Open, royalty-free standard for cross-platform,
- For heterogeneous parallel-computing systems
- Cross-platform. Implementations for
  - ATI GPUs
  - NVIDIA GPUs
  - x86 CPUs

What about Applications?

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).



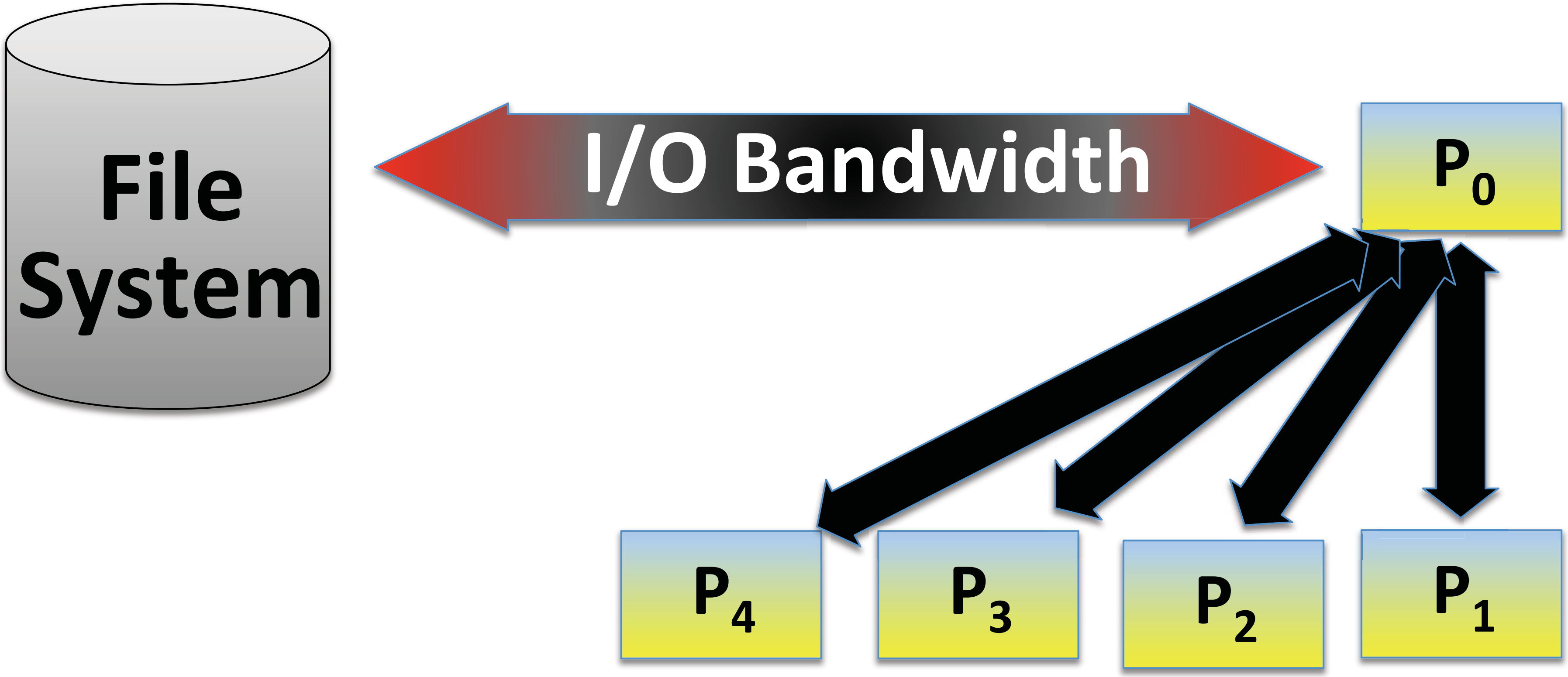
maximum speedup tends to
1 / (1 − P)
P= parallel fraction

1000000 core

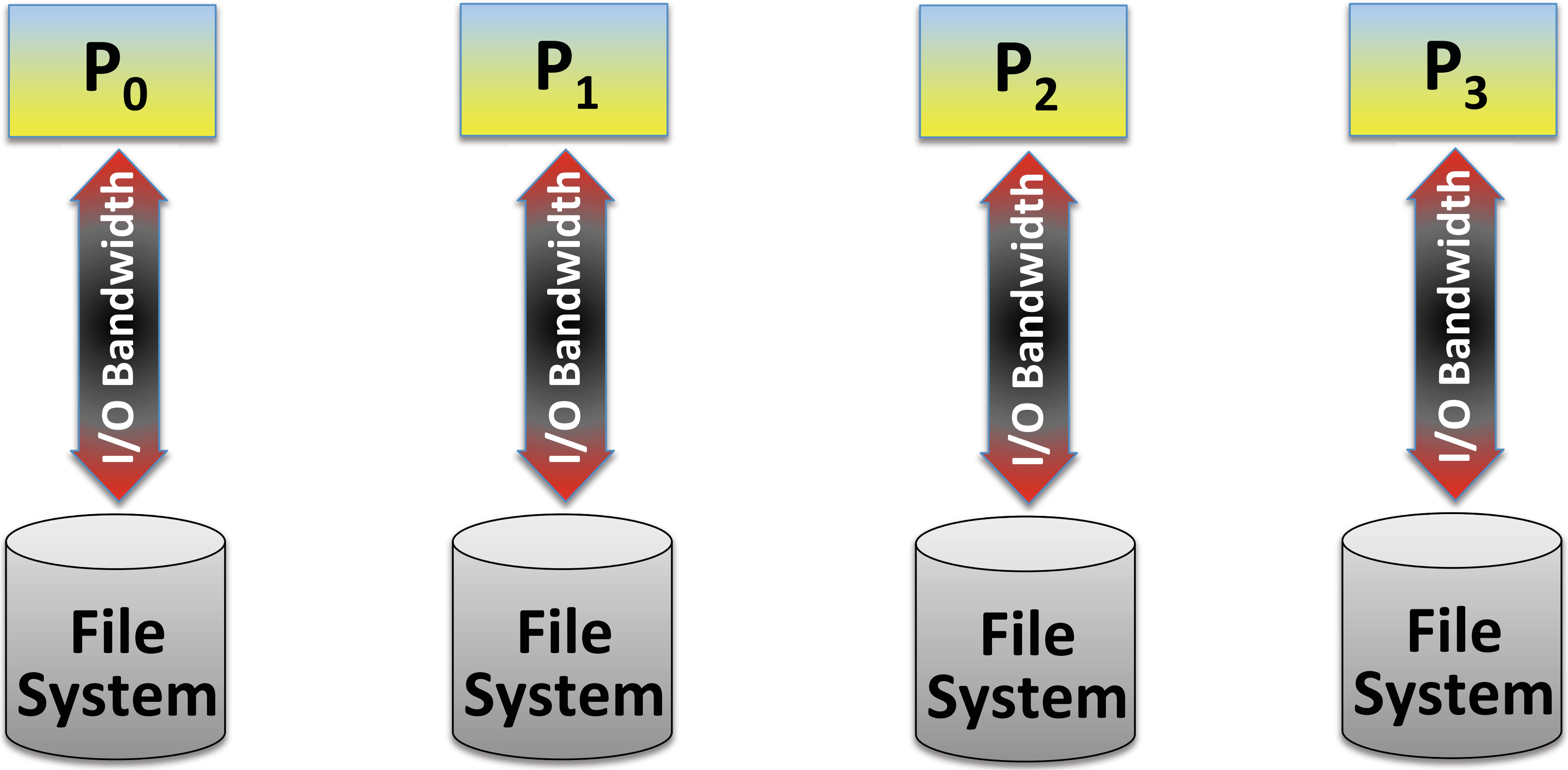
P = 0.999999

serial fraction= 0.000001

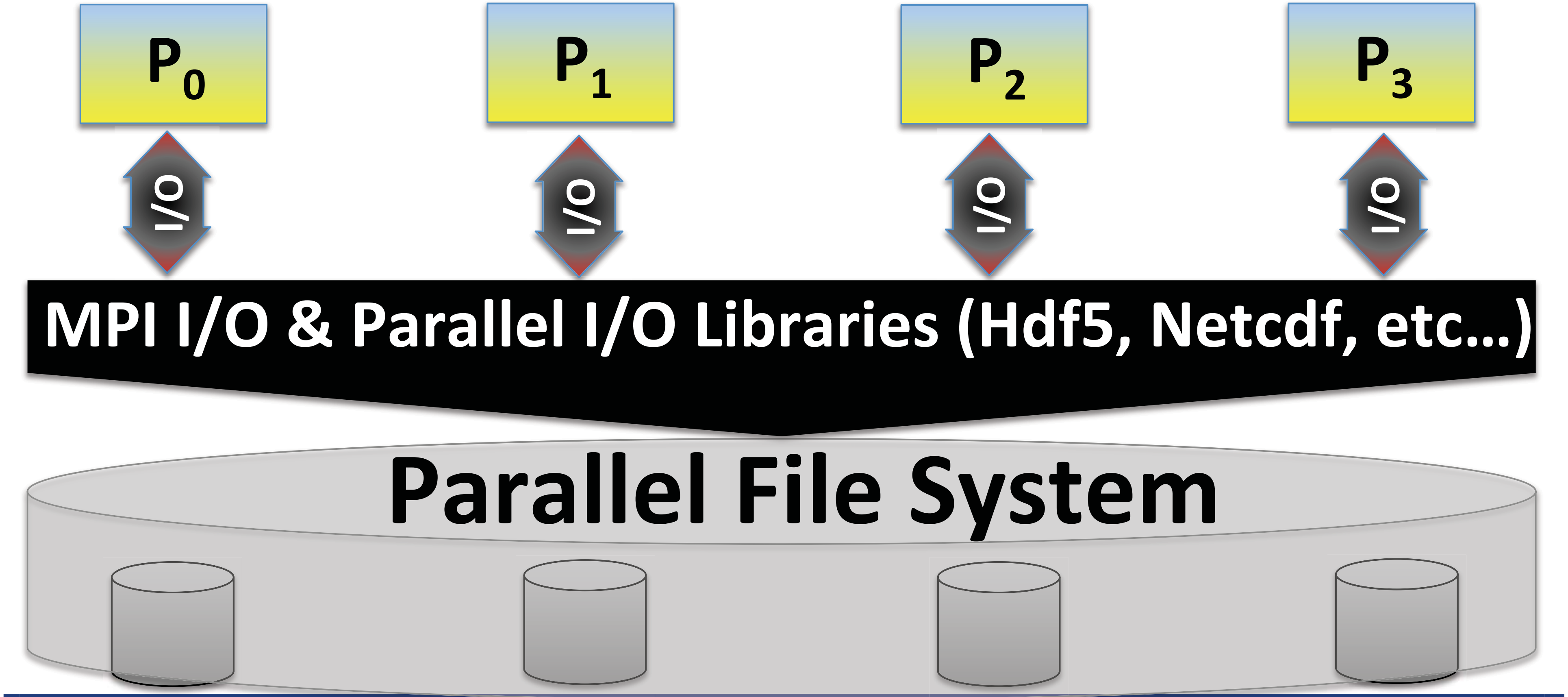
Parallel I/O



Parallel I/O


P_0
P_1
P_2
P_3
I/O Bandwidth
I/O Bandwidth
I/O Bandwidth
I/O Bandwidth
File System
File System
File System
File System

Parallel I/O


P_0
P_1
P_2
P_3
I/O
I/O
I/O
I/O
MPI I/O & Parallel I/O Libraries (Hdf5, Netcdf, etc...)
Parallel File System

Conclusions

- The technology evolution/revolution do not allow to longer work around the parallelism
- Higher granularity enhance large parallelism but it increases concurrency
- Codes are being modularized and parameterized to enhance different levels of granularity and consequently to become more “platform adaptable”
- HW and SW architectural knowledge is needed to handle complexity

Thanks for your attention!!