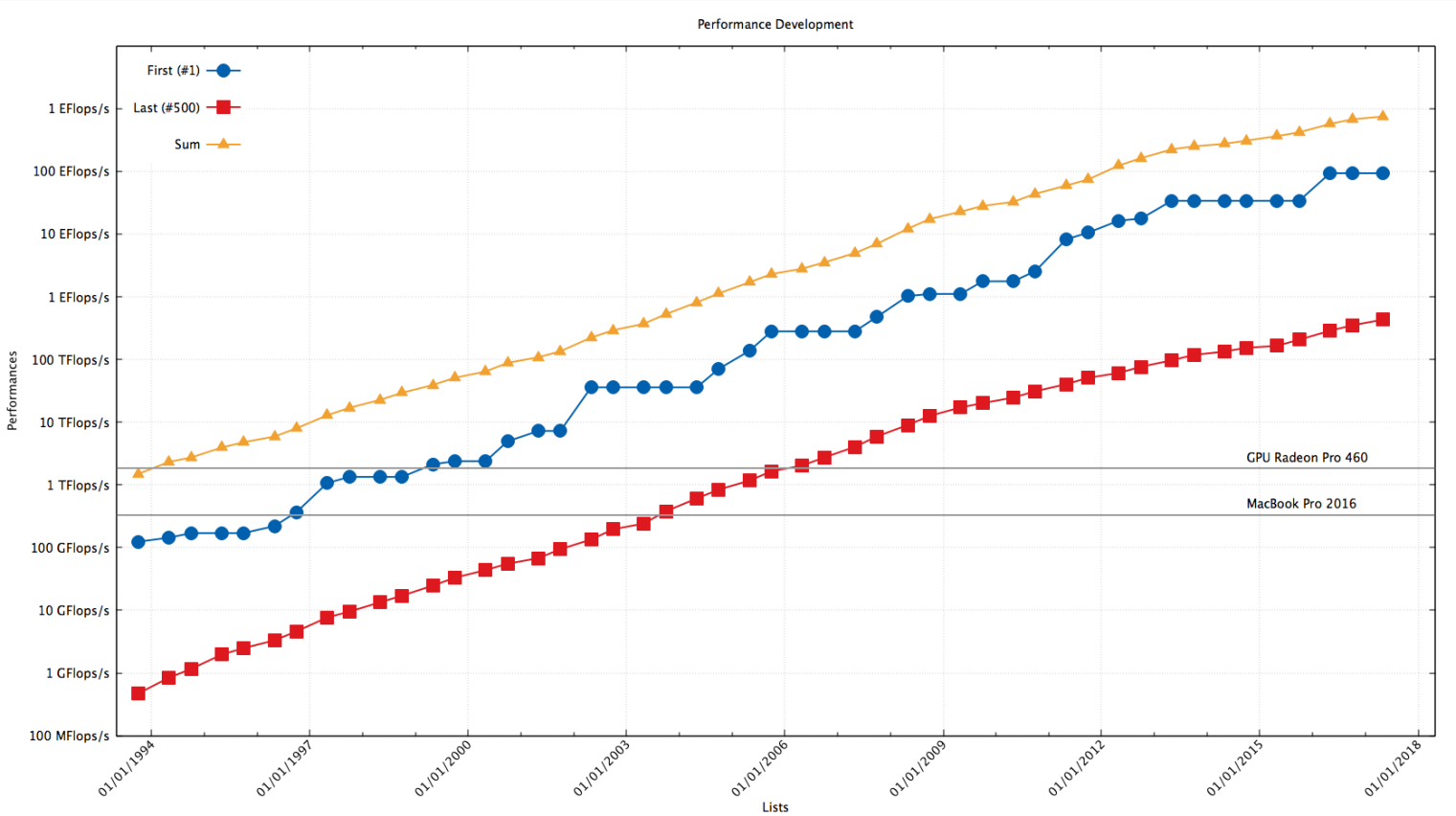
HPC and modeling

Chapter 1 – A first take on parallelism

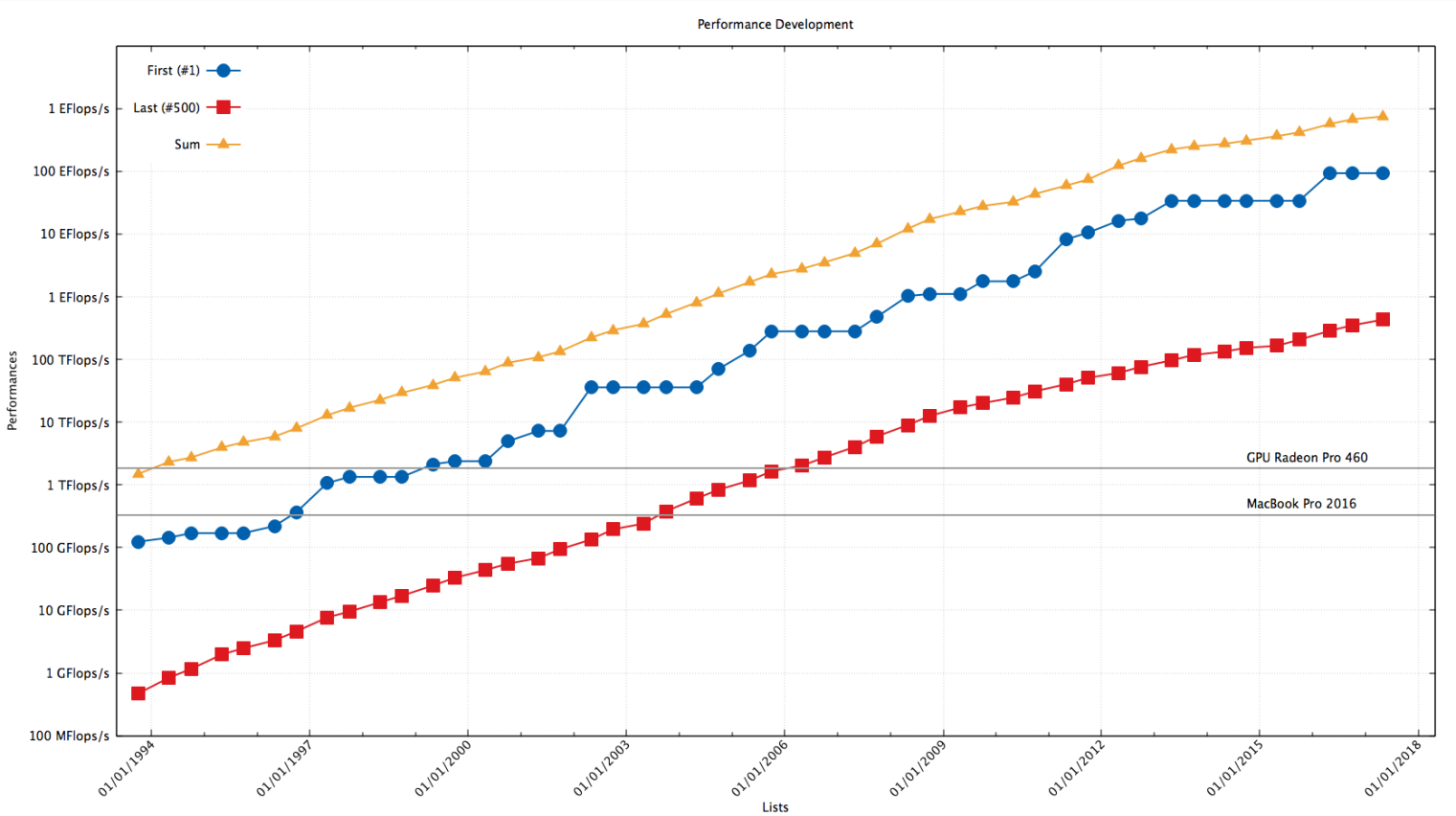
M2 – MSIAM

October 12, 2018

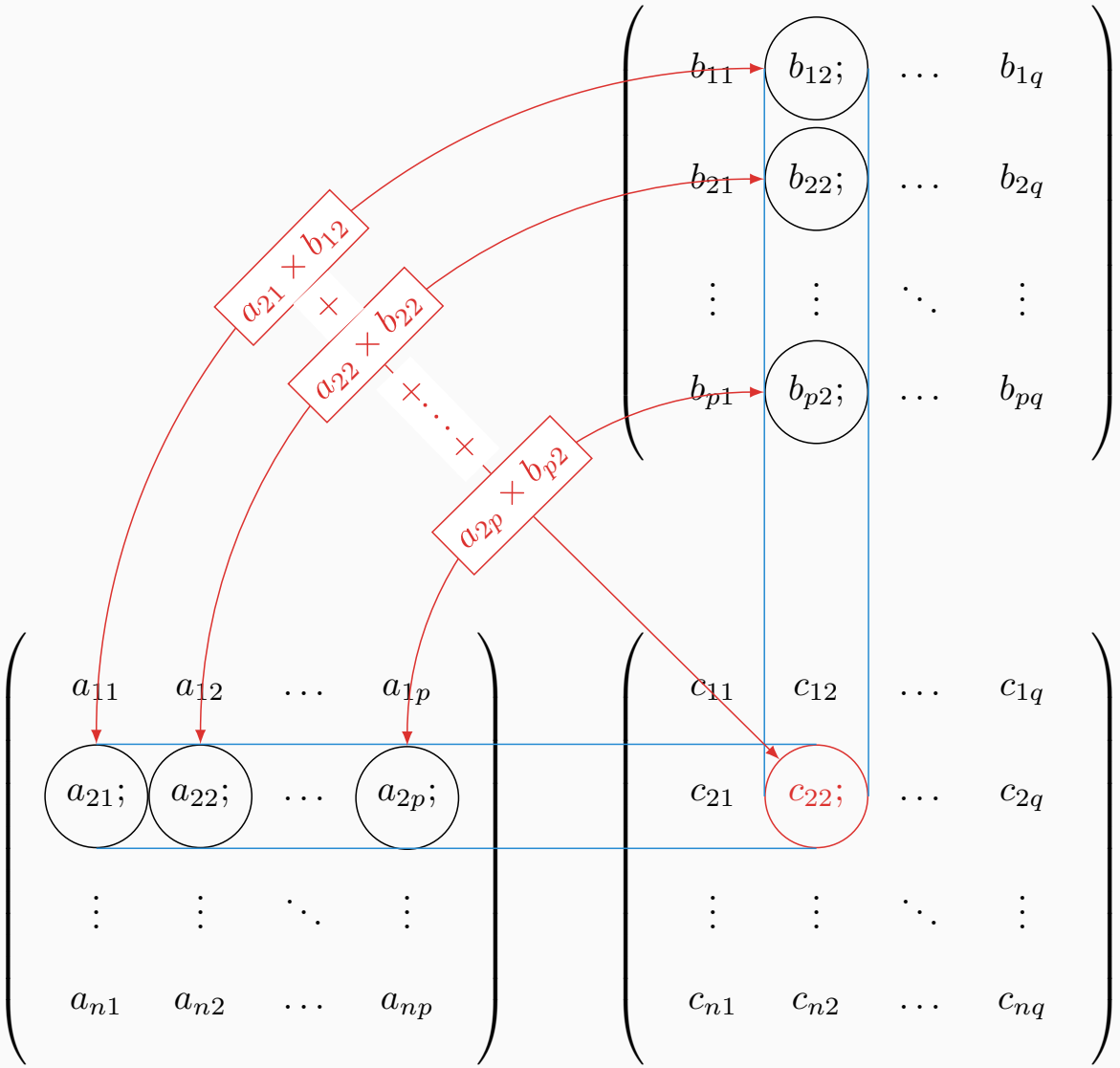
Top 500


Performance Development
First (#1)
Last (#500)
Sum
1 EFlops/s
100 EFlops/s
10 EFlops/s
1 EFlops/s
100 TFlops/s
10 TFlops/s
1 TFlops/s
100 GFlops/s
10 GFlops/s
1 GFlops/s
100 MFlops/s
Performances
GPU Radeon Pro 460
MacBook Pro 2016
01/01/1994
01/01/1997
01/01/2000
01/01/2003
01/01/2006
01/01/2009
01/01/2012
01/01/2015
01/01/2018
Lists

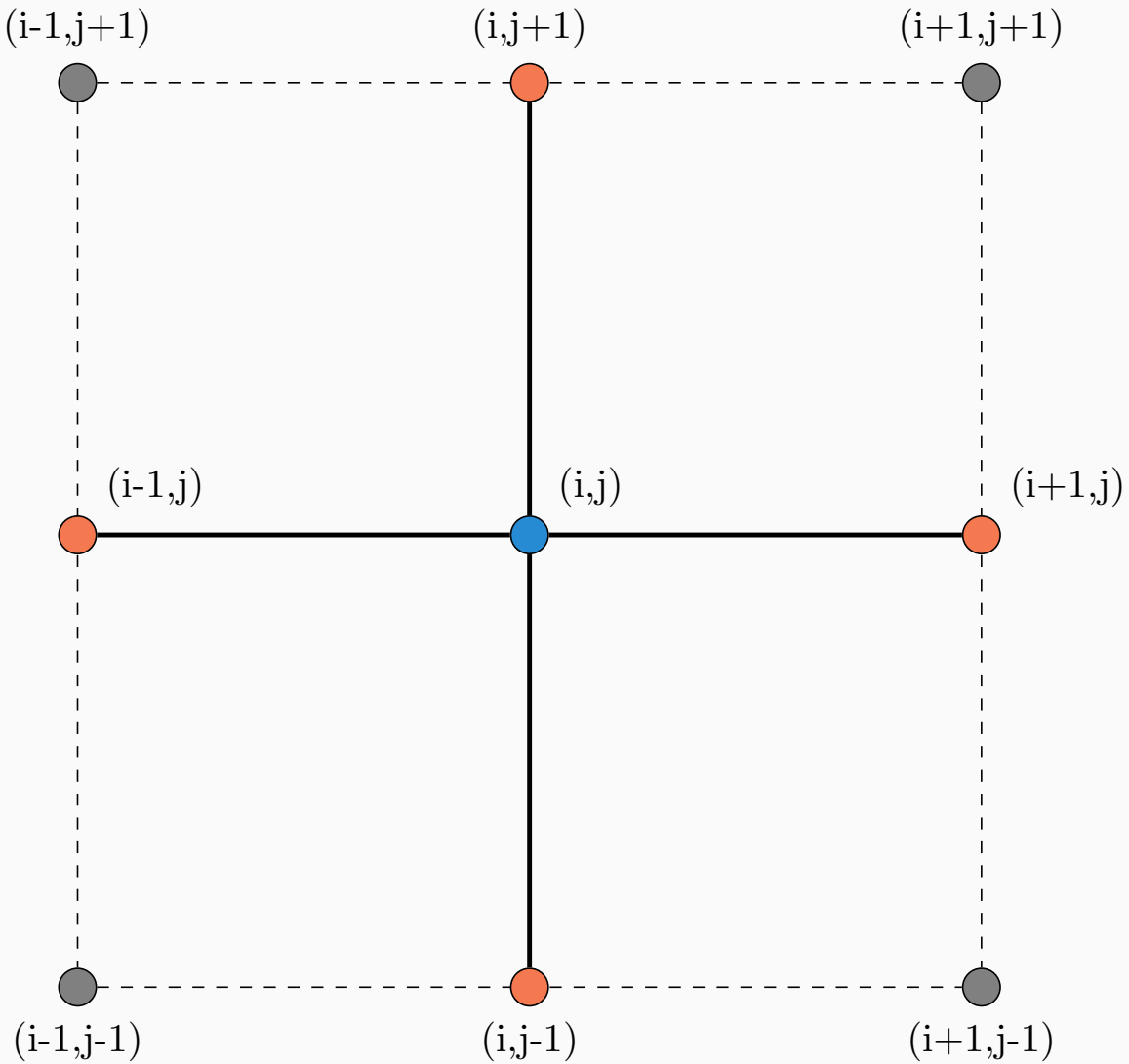
Top 500


Performance Development
First (#1)
Last (#500)
Sum
1 EFlops/s
100 EFlops/s
10 EFlops/s
1 EFlops/s
100 TFlops/s
10 TFlops/s
1 TFlops/s
100 GFlops/s
10 GFlops/s
1 GFlops/s
100 MFlops/s
Performances
GPU Radeon Pro 460
MacBook Pro 2016
01/01/1994
01/01/1997
01/01/2000
01/01/2003
01/01/2006
01/01/2009
01/01/2012
01/01/2015
01/01/2018
Lists

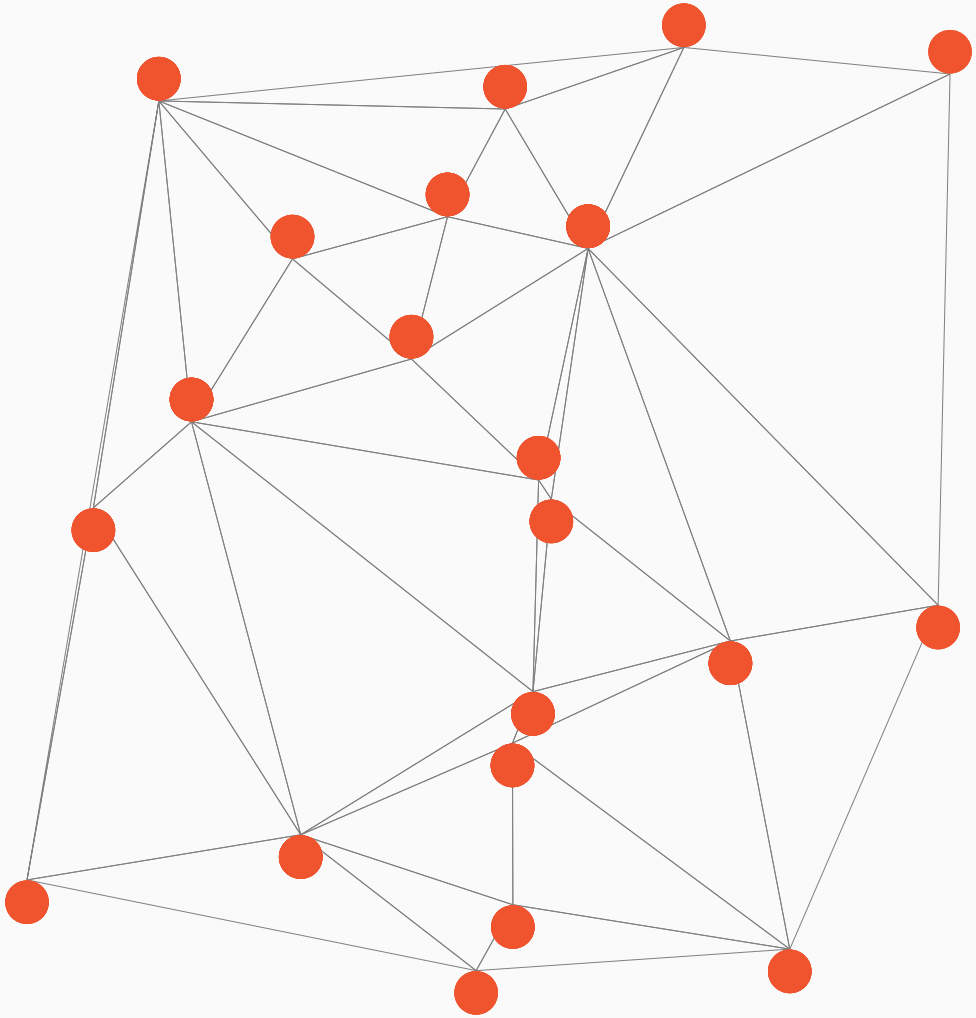
Matrix Multiplication



Images



Graph



Organization

General information

Contact
Christophe Picard: christophe.picard@imag.fr

Office 174 – Imag Building – 1st floor

Email headers: [MHPC]

Course website:

`http://chamilo.grenoble-inp.fr/courses/ENSIMAGWMM9MO16`

Class organization
Lectures, labs and project.

Context

- Decrease time to solution.
- Solve larger problem.
- Combine resources of several processing units: gain access to more memory and more processing power.
- Harness the processing power of modern architectures.
- Use idle computer to perform embarrassing parallelism computation (SETI@home).
- Improve the precision of computations in a limited time (weather forecast).

Objectives

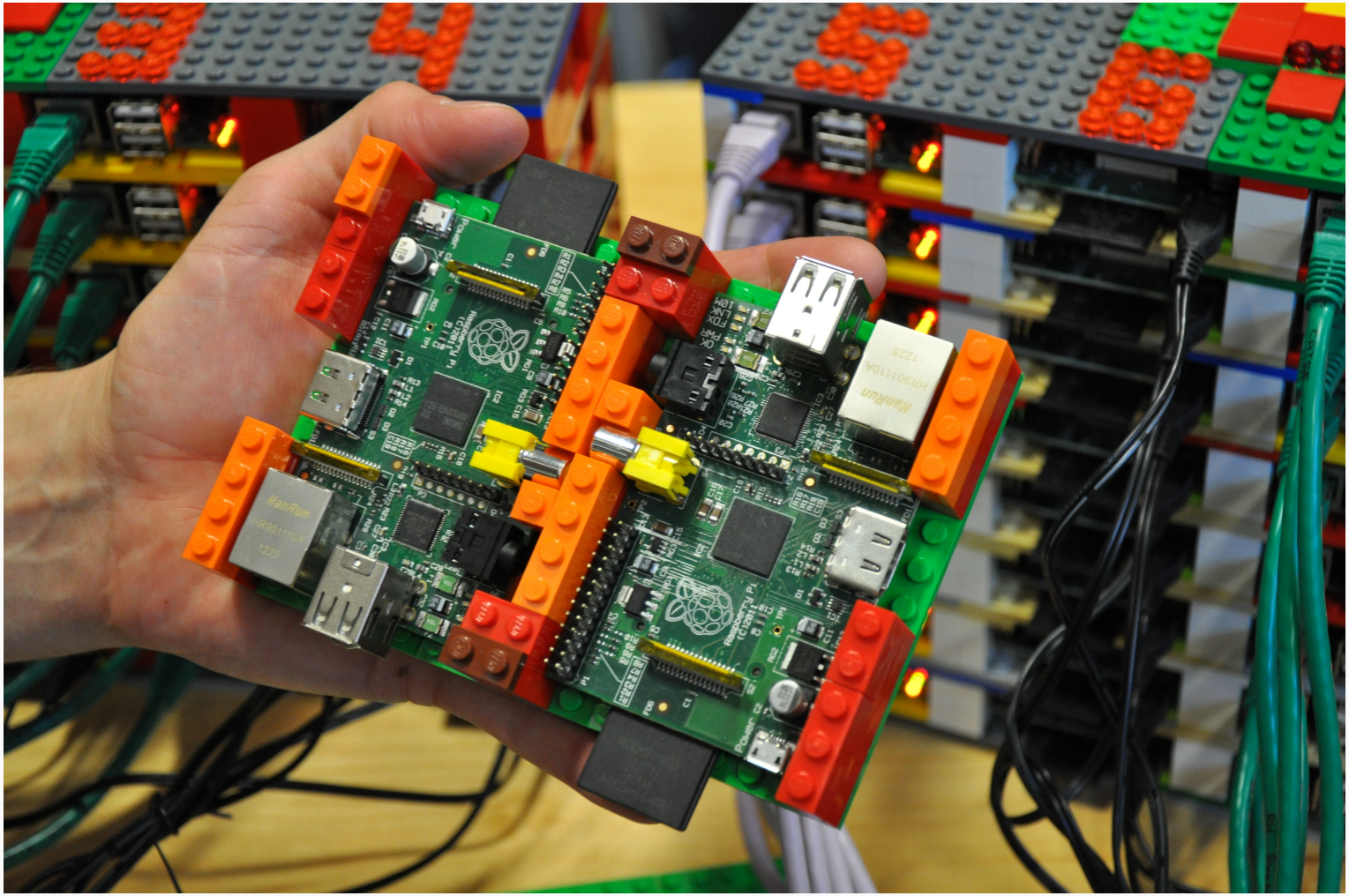
- Understand the different level of parallelism.
- Apprehend the concepts required for real-life applications.
- Experiment with different tools of parallel computing.
- Put into practice the theoretical concepts through an application.

Course content

- Introduction – Scale of parallelism (1 class)
- Programming pattern in parallelism – Part 1 (1 class)
- Labs – Introduction to OpenMP (1 lab)
- Programming pattern in parallelism – Part 2 (1 class)
- Labs – Introduction to MPI (1 lab)
- Small scale project (6 classes)
  - Introduce the problem.
  - Analysis of the code.
  - Strategies for parallelism.

Introduction

Legoland

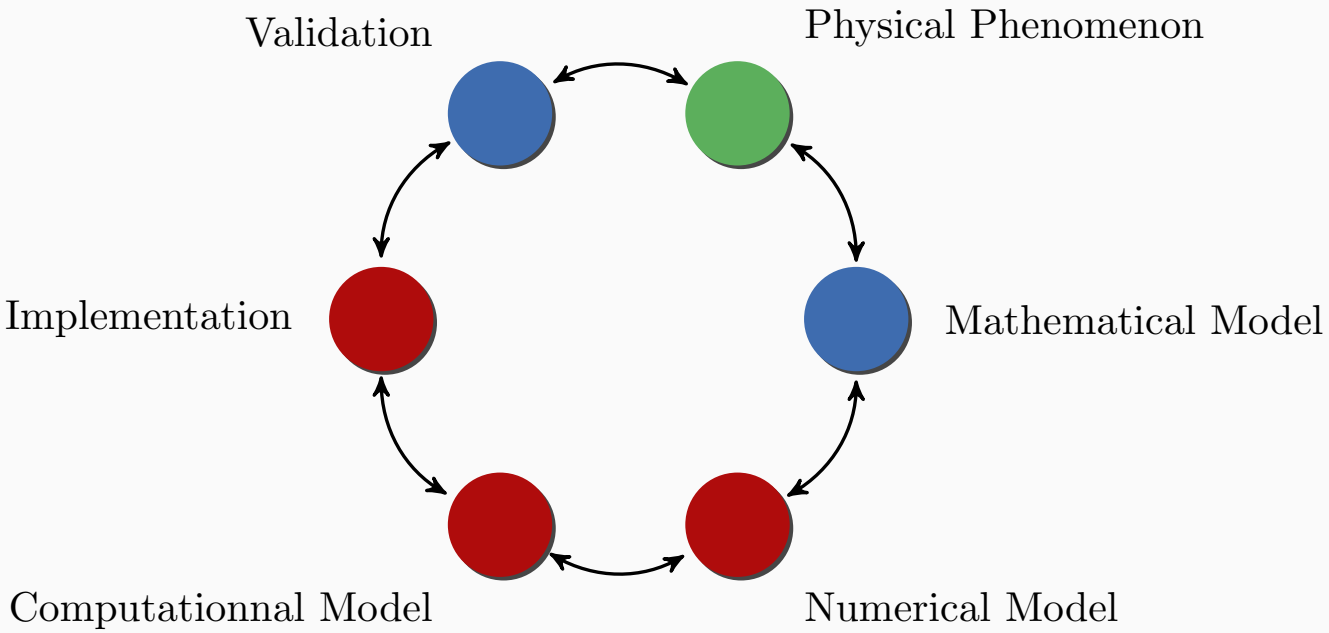


Curie

Why HPC?

- Sequential programming is limited
  - applications are parallel.
  - access to memory is limited.
  - engineering/cost limitations: it is easier to increase the number of unit that the frequency of a processor.
- Performances are evolving
  - Hardware is faster.
  - Algorithms are more efficients.

Modeling and computation



Goals of HPC

Intensive computation

- Solve a problem faster.
- Models are more sophisticated.
- Increase resolution of models.
- Increase interactivity.

Example

- Improve the rate: compute N problems simultaneously
- Decrease response time: solve a problem with N times faster.
- Increase the size of the problem: compute a problem N times larger.

How to increase performances?

HPC attempts to speed solution by dividing task into sub-tasks and executing simultaneously on different processing units.

- Identify where parallelism will be the most effective.
- Know the set of technological constraints.
- Design solution adapted to the problem and the constraints.

Limitations of sequential solutions

Clock speed limitation

- Current leakage.
- Power consumption.
- Heat dissipation.

} Not compatible with mobile devices

Standard optimisations

- Instruction prefetching
- Instruction reordering
- Pipelined functions units
- Branch prediction
- Functional unit allocation
- Hyperthreading

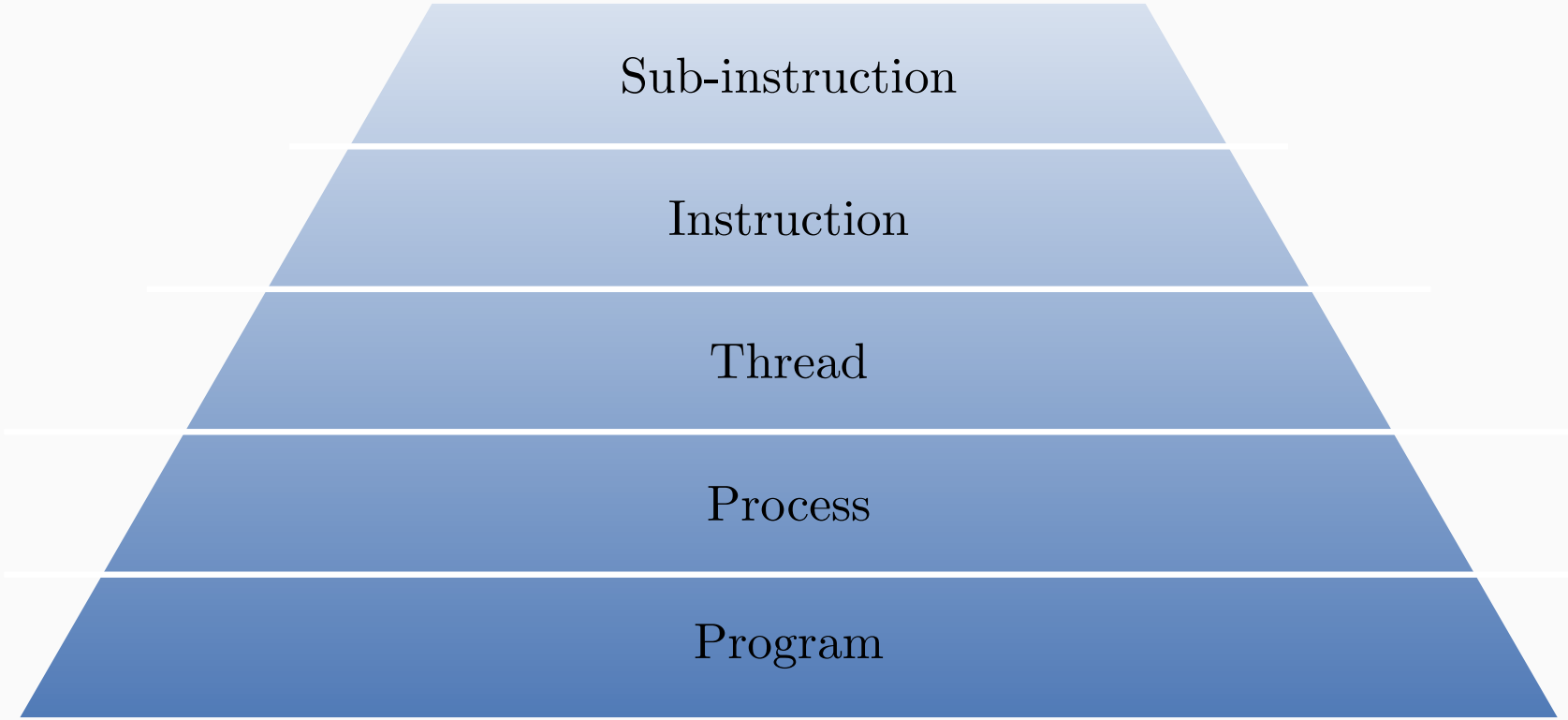
} No control of the programmer

How to achieve parallelism

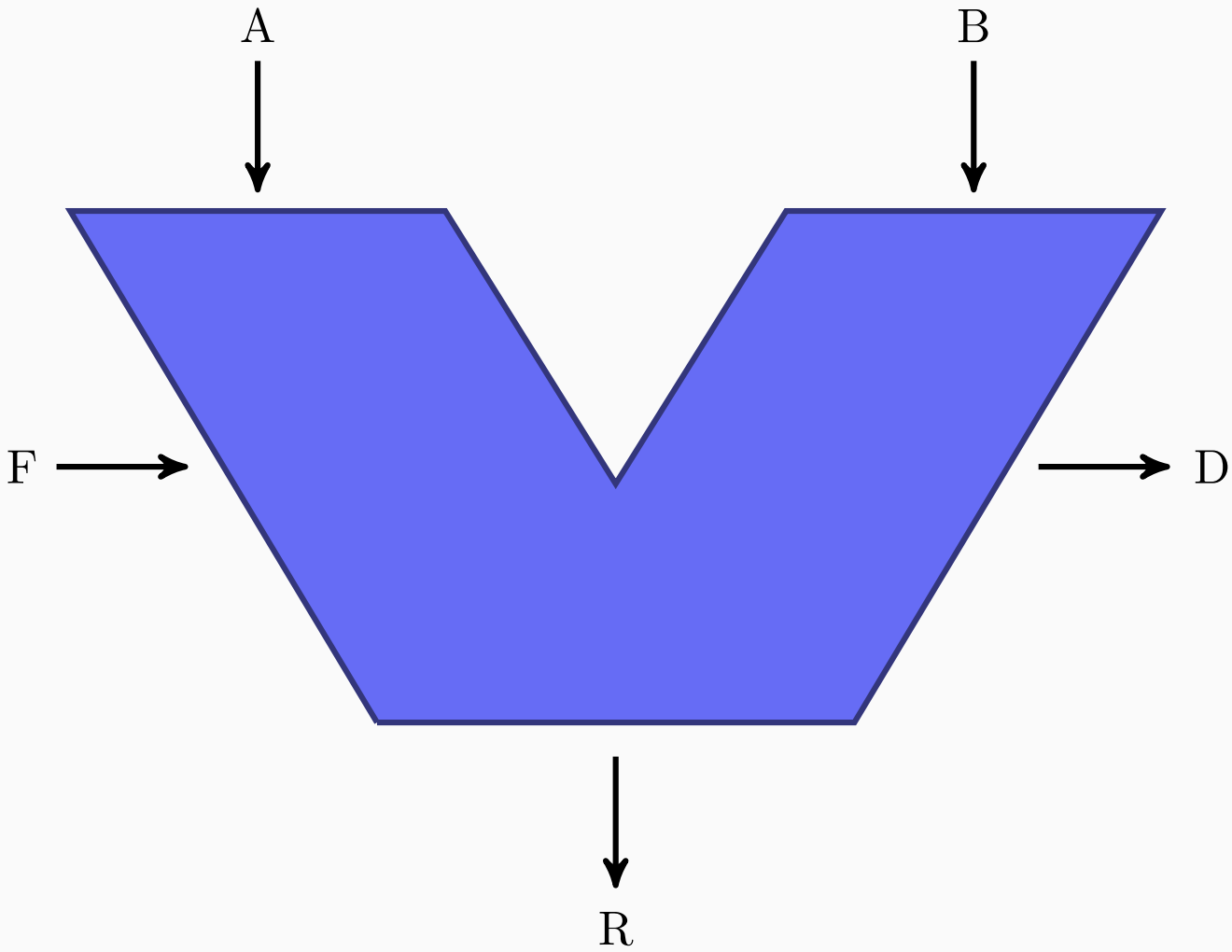
- Processors: multicore, memory, network, accelerators, instructions.
- Compilers: dedicated library, automatic parallelism.
- Algorithms: tailored algorithms.
- Mathematics: adapted numerical methods, evolutionary methods.

Parallel platforms

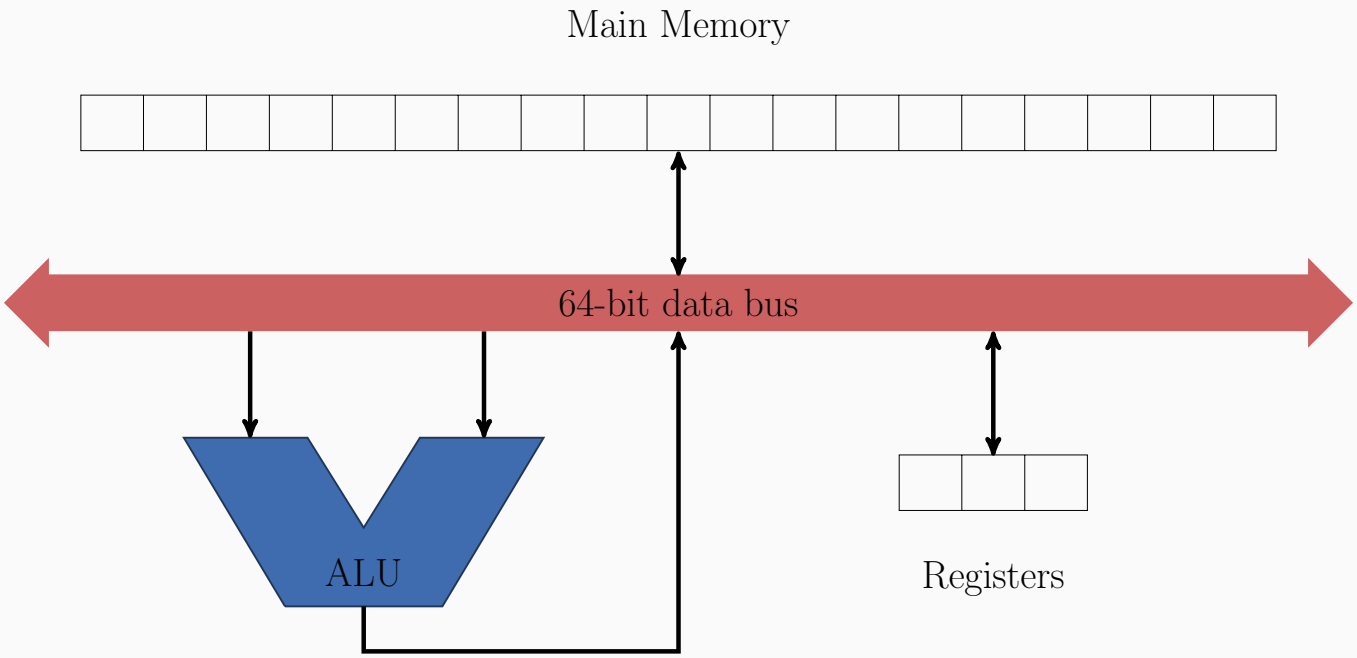
Parallel hierarchy



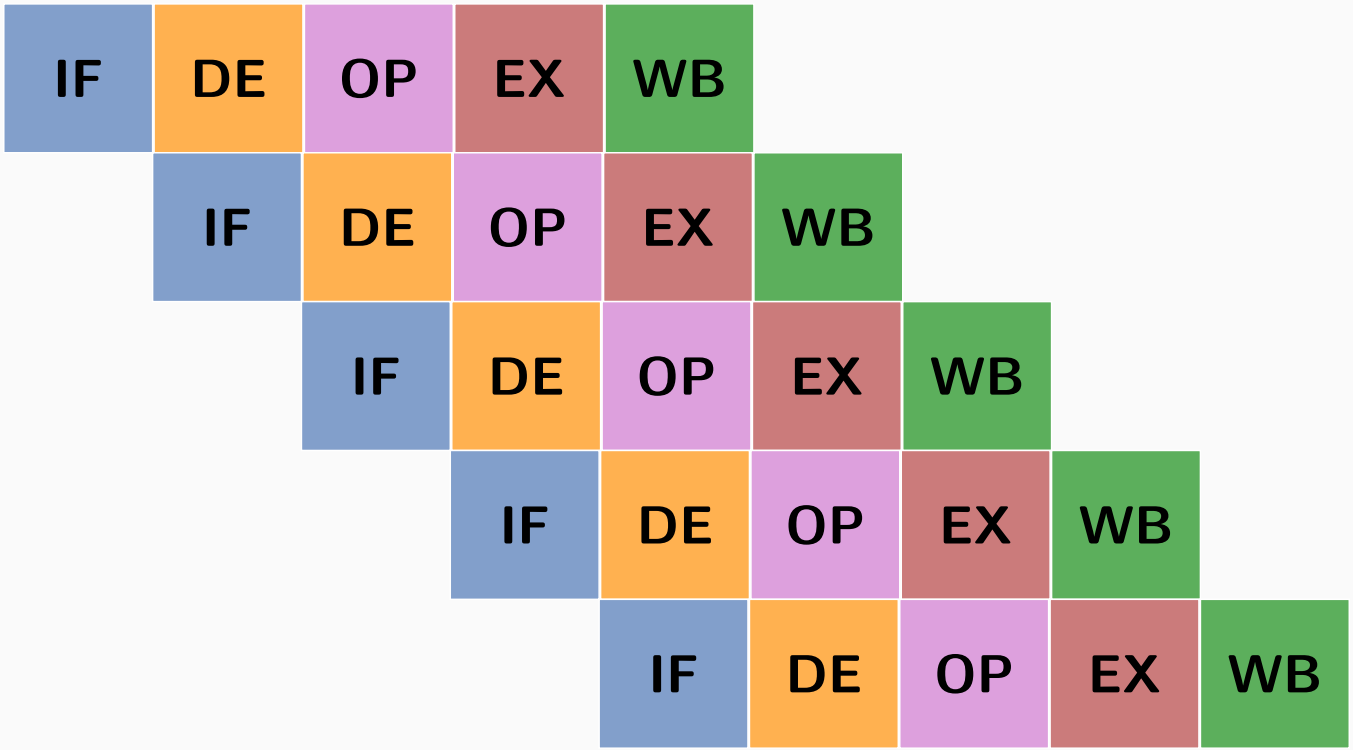
Low-level hardware



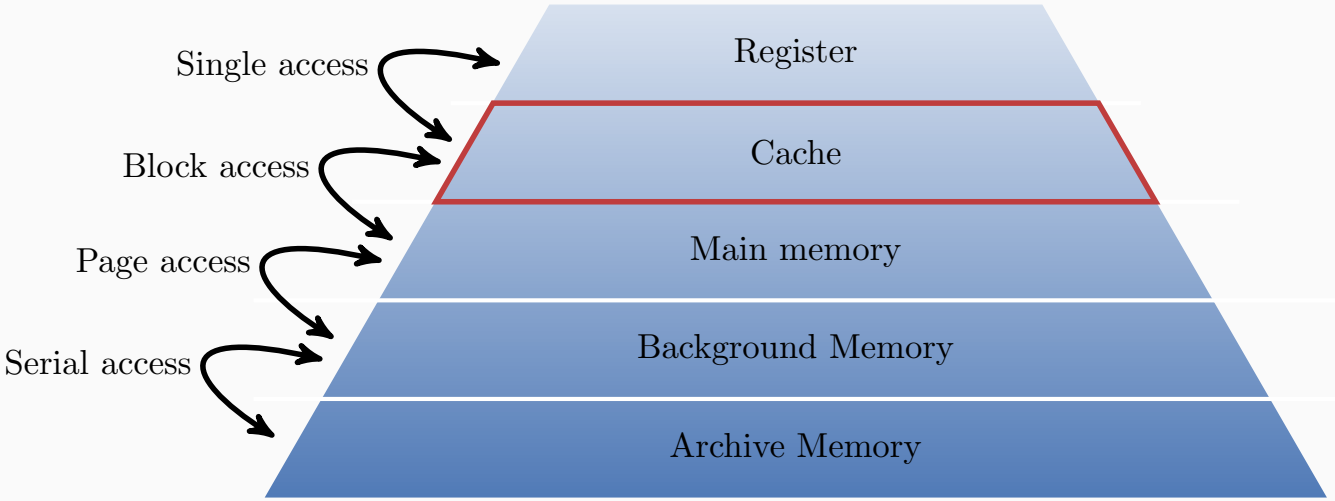
Abstract processor



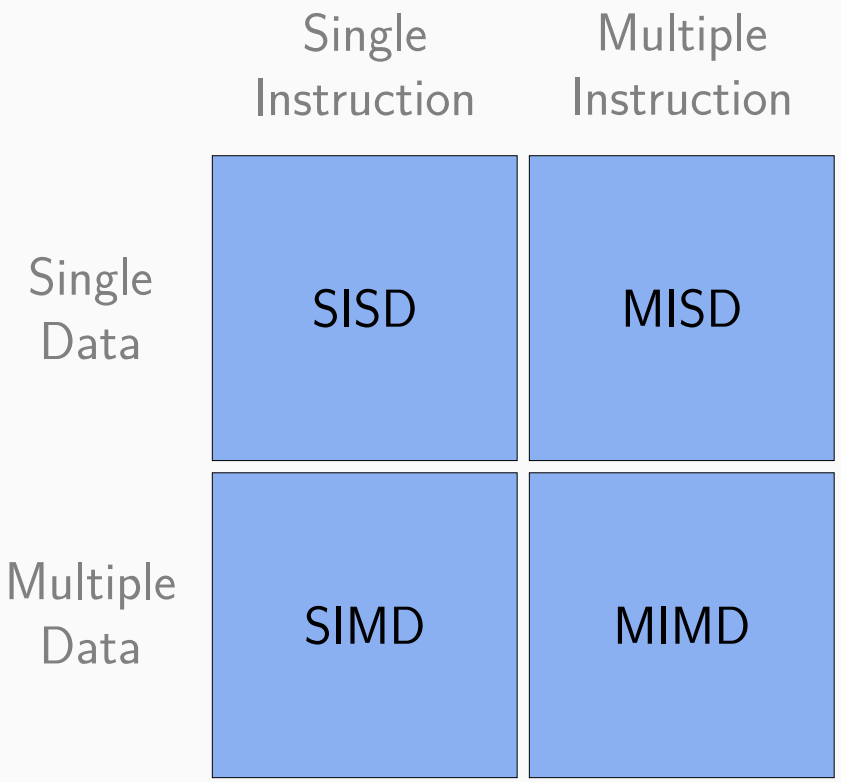
Pipeline architecture



Memory hierarchy

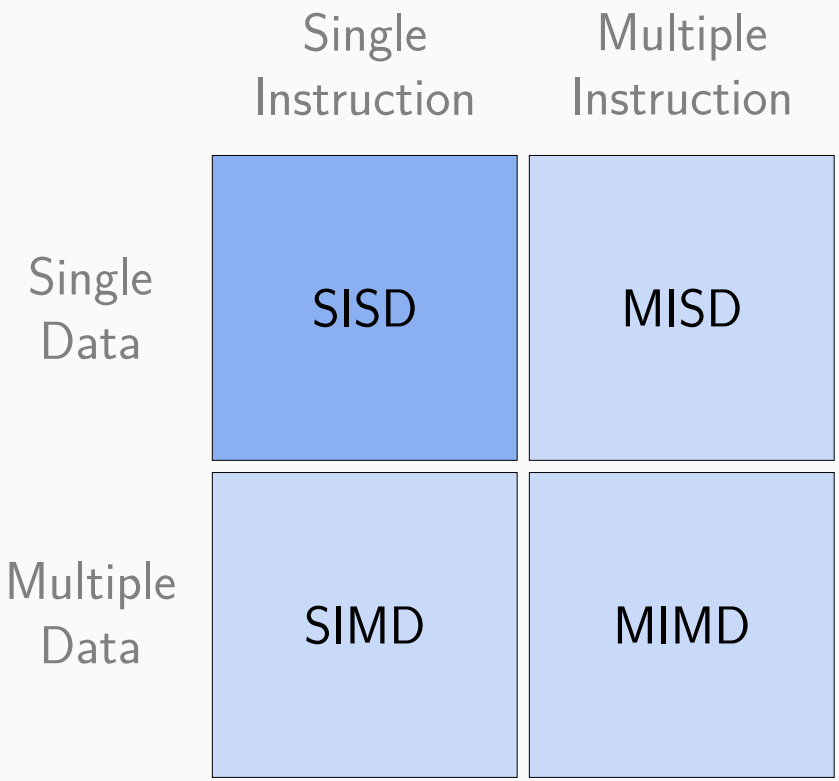


Flynn classification



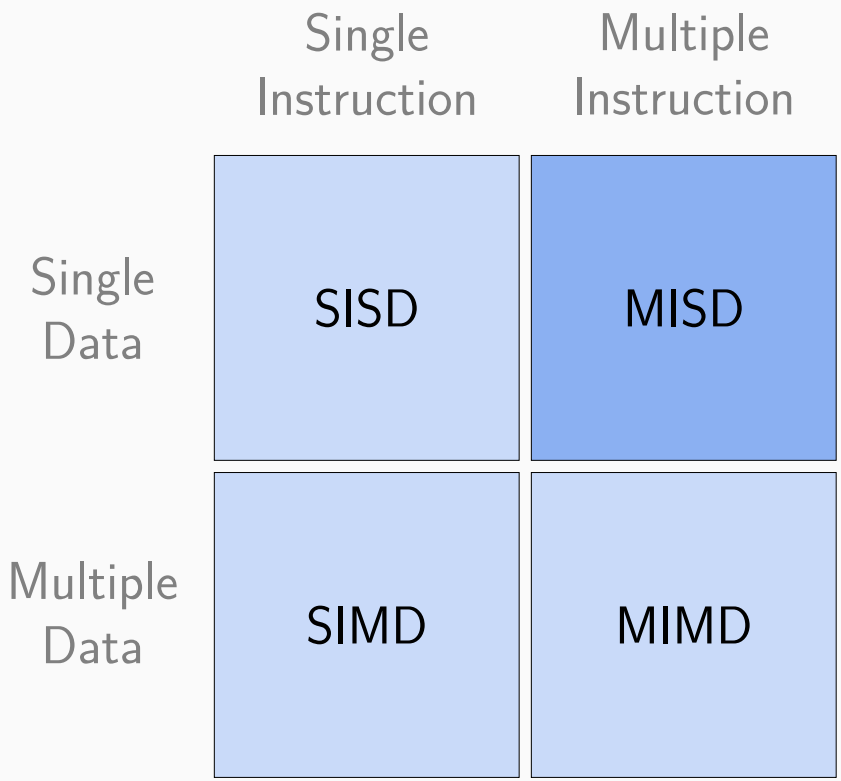
- Classification depends on two parameters: the nature of the data flow and the sequence of instruction apply to them.

Flynn classification



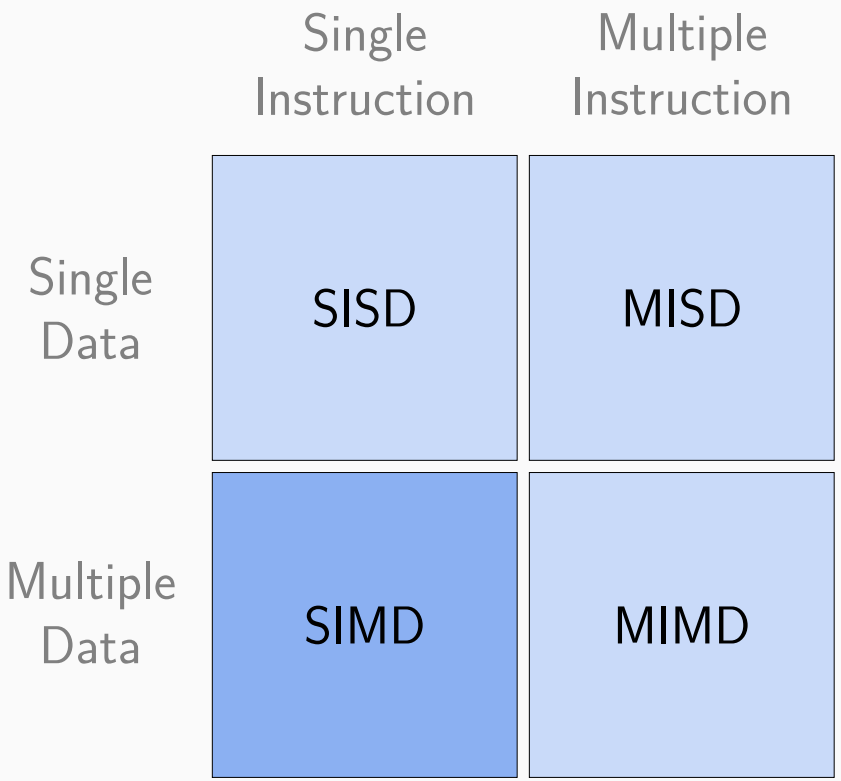
- One flow of data supply one processing unit.
- Uniprocessor, execution is determinist.

Flynn classification



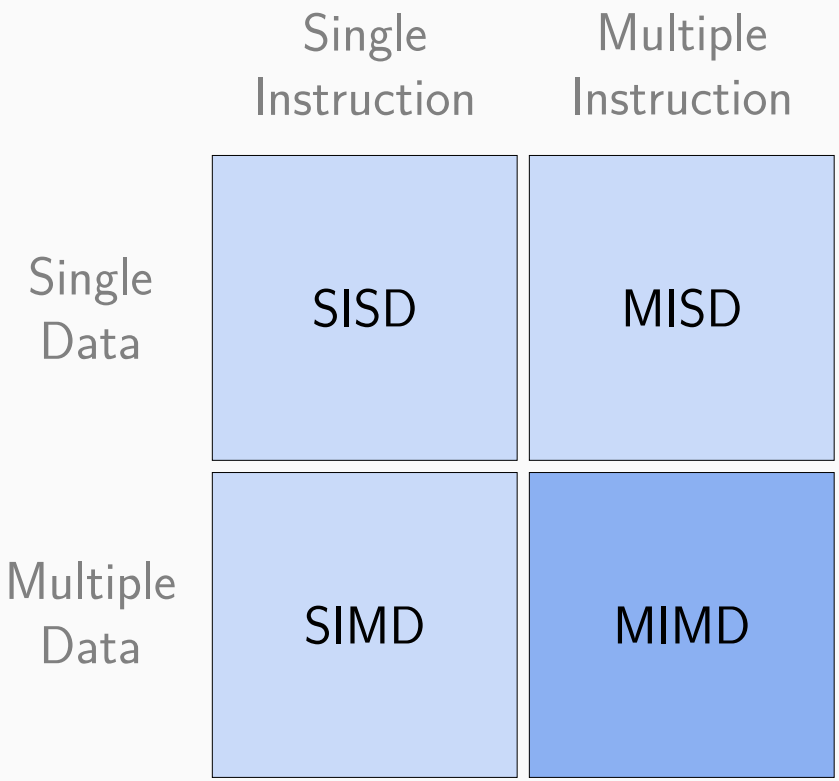
- One flow of data supply several processing unit.
- Fault tolerant. Use in cryptography or filtering.

Flynn classification



- All PU execute the same instructions on different data.
- Execution are pipelined. Vector processors. GPUs.

Flynn classification



- PU may execute different instructions on different data.
- All moderns multiprocessors.

Parallel computational units

Implicit parallelism

- Parallel execution of different processor instructions.
- Happens almost automatically
- Can only be influenced indirectly by the programmer.

Multi-core/Multi-CPU/Many-Core

- Found in commodity hardware today
- Computational units share the same memory

Clusters

- Aggregation off compute units
- Independent systems linked using fast interconnect
- Each system has its own memory

Parallel computational units

Accelerators

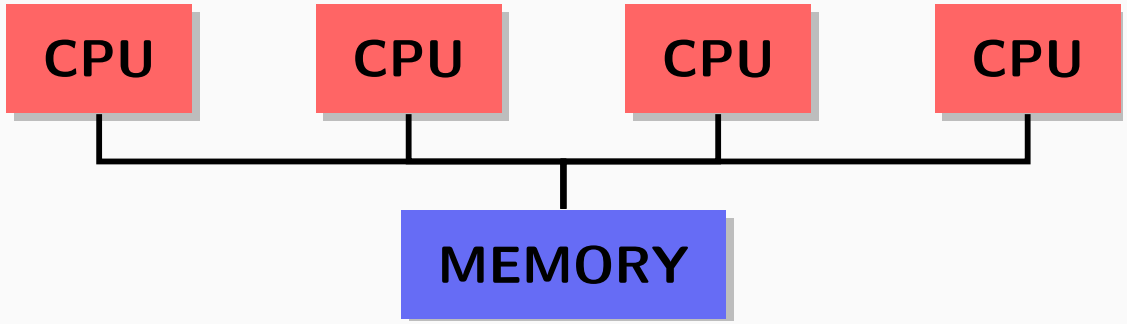
- Offer specialize features
- Often have their own memory
- Often not autonomous

Vector processors/Vector Units

- Perform same operations on multiple pieces of data simultaneously.
- Need a well though layout of the data

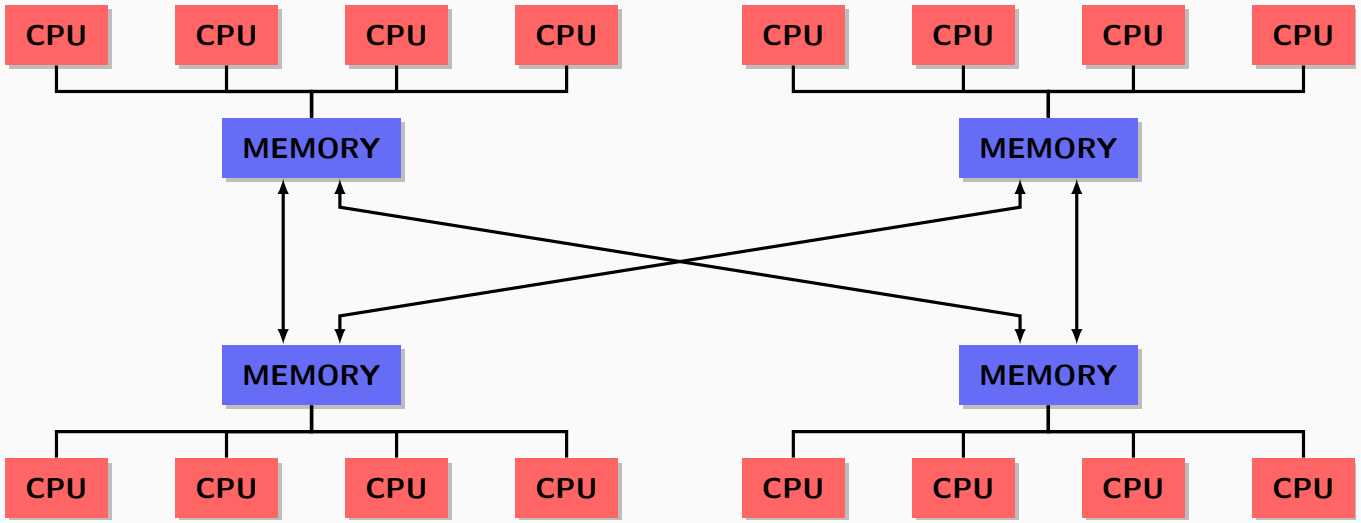
Shared memory (1)

- All the processors shares the same memory space. They communicate using reading and writing shared variables.
- Each processing unit carry out its task independently but modification of shared variables are instantaneous.
- Two kind of shared memories
  - SMP (Symmetric MultiProcessor) – All the processors share a link to the memory. Access to the memory is uniform.



Shared memory (2)

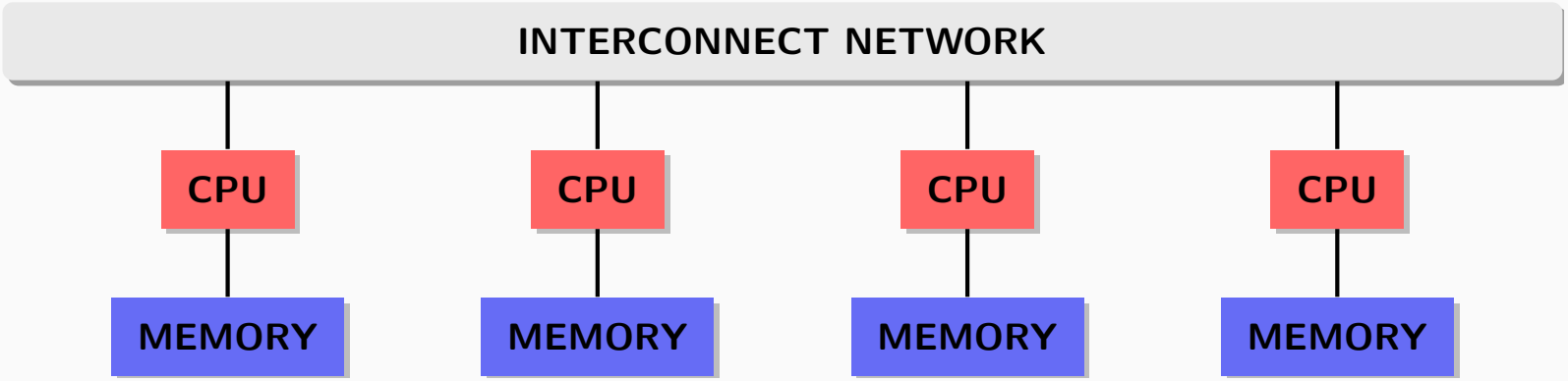
- NUMA (NonUniform Memory Access) – All the processors can access to the memory but not uniformly. Each processor has a preferred access to some memory part.



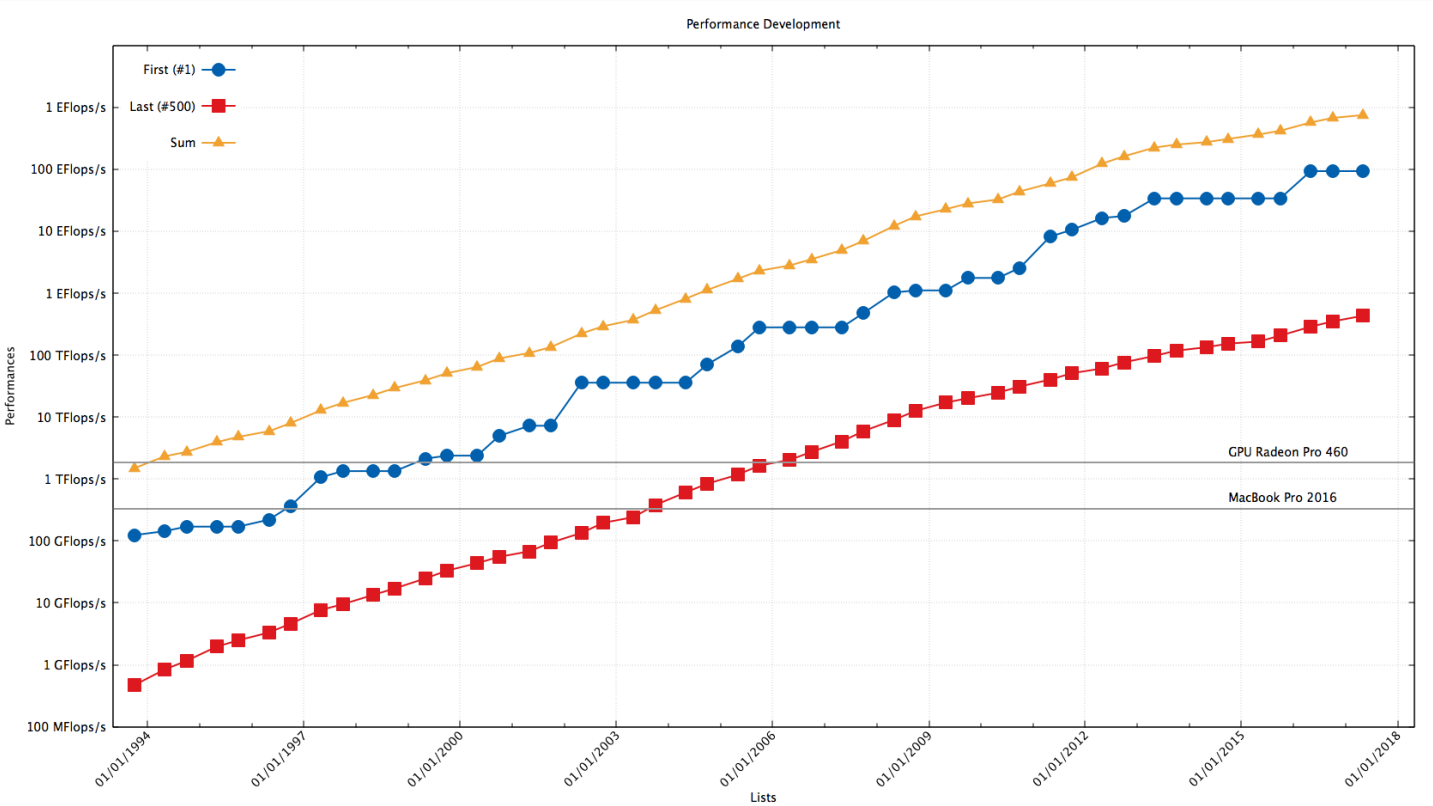
  - Decrease the risk of bottleneck to memory access.
  - Local memory cache on each processor to mitigate the effect of non-uniform access.

Distributed memory

- Each processor has its own memory. There is no global memory space.
- Each processor communicate with the others using messages.
  - Modification of variables are local and only the processor managing the memory can access it.
  - Each processor work independently on its own set of variables.
  - The speed of the resolution depends on the architecture: network, topology, processors.
  - Can scale easily.



Top 500


Performance Development
First (#1)
Last (#500)
Sum
1 EFlops/s
100 EFlops/s
10 EFlops/s
1 EFlops/s
100 TFlops/s
10 TFlops/s
1 TFlops/s
100 GFlops/s
10 GFlops/s
1 GFlops/s
100 MFlops/s
Performances
GPU Radeon Pro 460
MacBook Pro 2016
01/01/1994
01/01/1997
01/01/2000
01/01/2003
01/01/2006
01/01/2009
01/01/2012
01/01/2015
01/01/2018
Lists

Introduction to shared memory computing