A very basic Introduction to HPC

Trying to get the most out of the computer

ISYA 41
Socorro
2018

Juan Carlos Muñoz Cuartas
Instituto de Física
Universidad de Antioquia

Growth of the field in Astronomy has been parallel to the growth of computing power

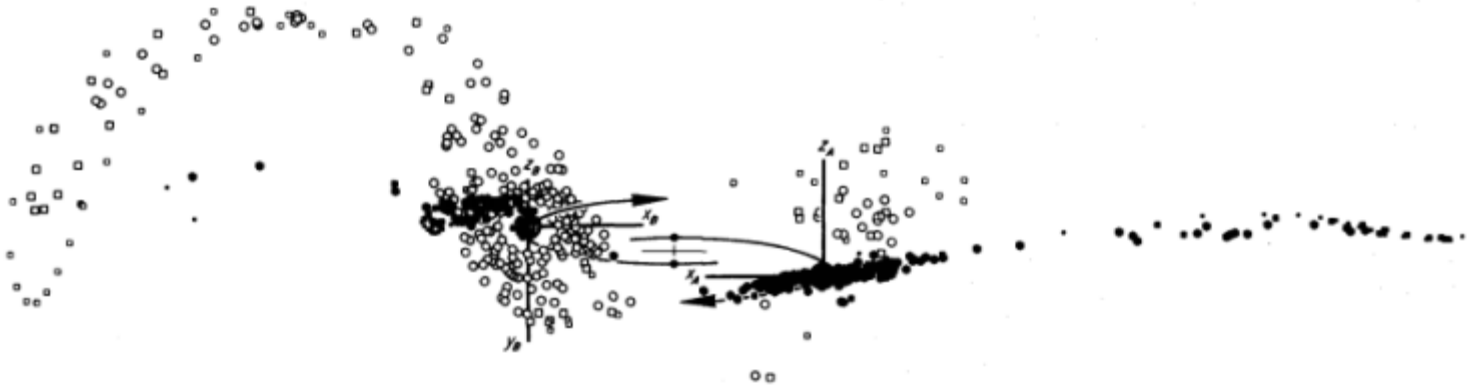
As computer power grows, our “astronomy knowledge” does the same way

(One might think is the other way around!)

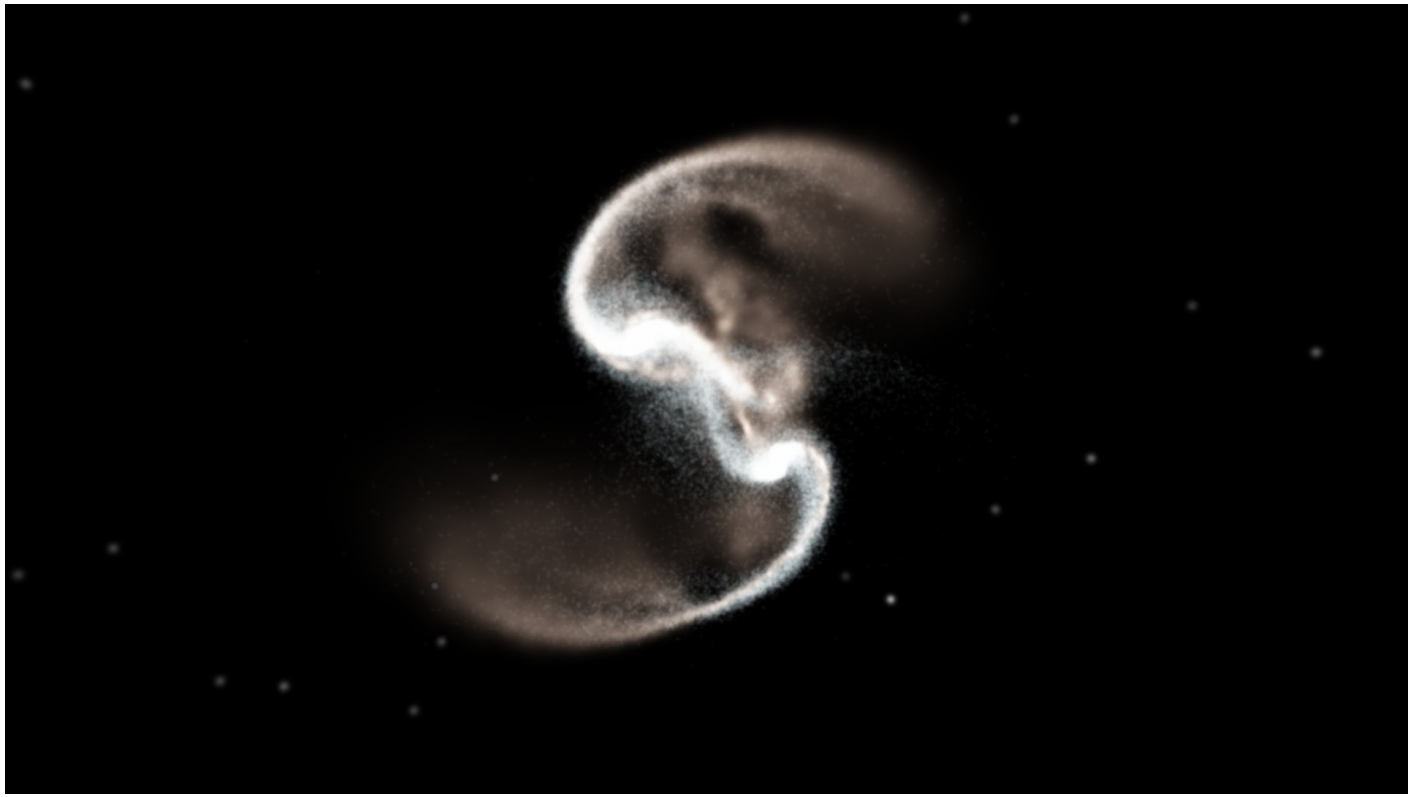
This has changed the reality of astronomy in a very deep way!

What your mama' HOPES you are doing

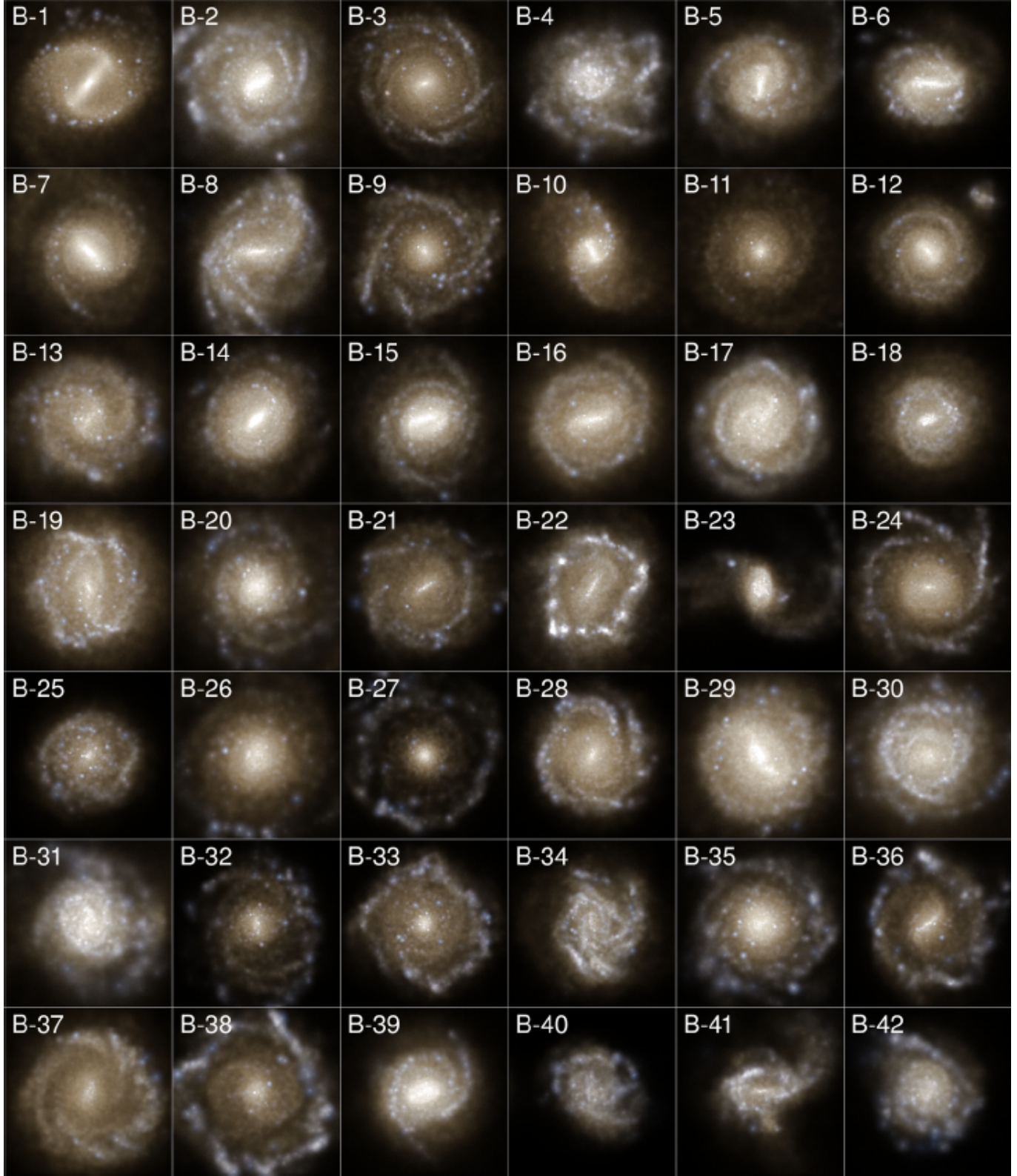
What you are really doing!!



Toomre & Toomre 1972



STSCI 2014



Illustris TNG project

But, why do we need computers in astronomy

If I still have to convince you that we need computers to do our work in astronomy... here a few scenarios where we really need them to do our job...

Data Acquisition

Did you know that CCD chips were created (at first!) as as computer memory devices?

Data acquisition is controlled by a computer: CCD detector is a silicon-semiconductors. Understanding how this computer works is needed to ensure you understand the data you are acquiring!

For example: read out noise is introduced by reading the CCD too fast!

Data management

Once you have your data, you realize you have looooooots of data, more than you can handle…

Organizing this data is a “must” if we pretend to make any kind of “smart” use of all this data.

Project	Size	Dir Count	File Count
APOGEE			
	75.2 Tb	122364	18278403
EBOSS			
	89.4 Tb	225765	107386379
MANGA			
	5.8 Tb	4739	375964
SEGUE			
	0.5 Tb	3190	32639
SPECDATA			
	8.1 Tb	56139	13067388
	179 Tb	412197	139140773

What will we do with the data from LSST? Several Tb per night?

Think on a pile 3.5m tall of hard drives just with a copy of SDSS data

Next you would like to analyze your data.

Assume you already solved the issue of swimming in Tb of data in you database.

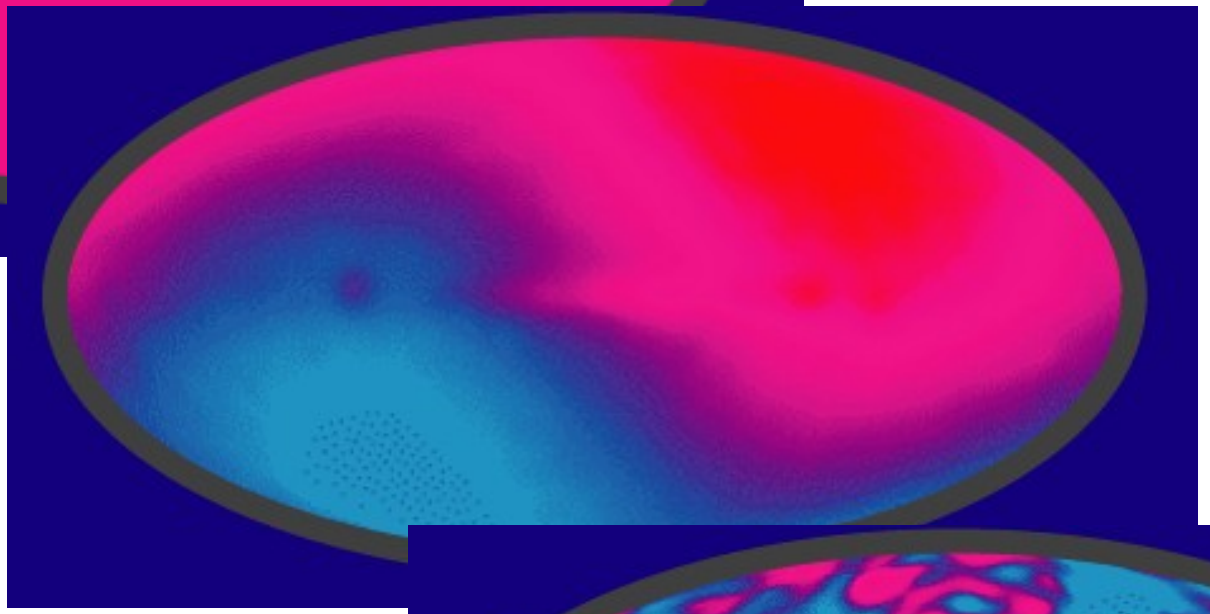
You already have a plan on how to manage your data!

You would like to clean up your data from all kind of “noise”.

Will see it with Karín in the next days and you know you don’t do it by hand!

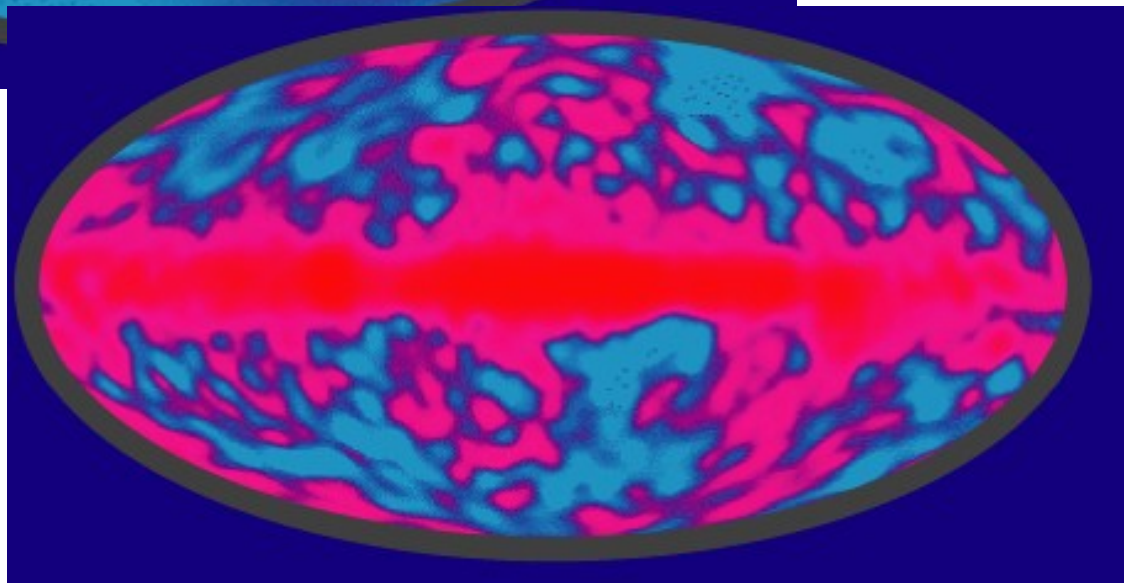
Other thing you may like to do is to take your data **and see if it fits a model**

This is what the data shows associated to the dipole contribution



This is what an actual CMB map “looks like”

This is what the processing shows once you remove the dipole contribution



Science is driven by data… but you want to get your “theory”

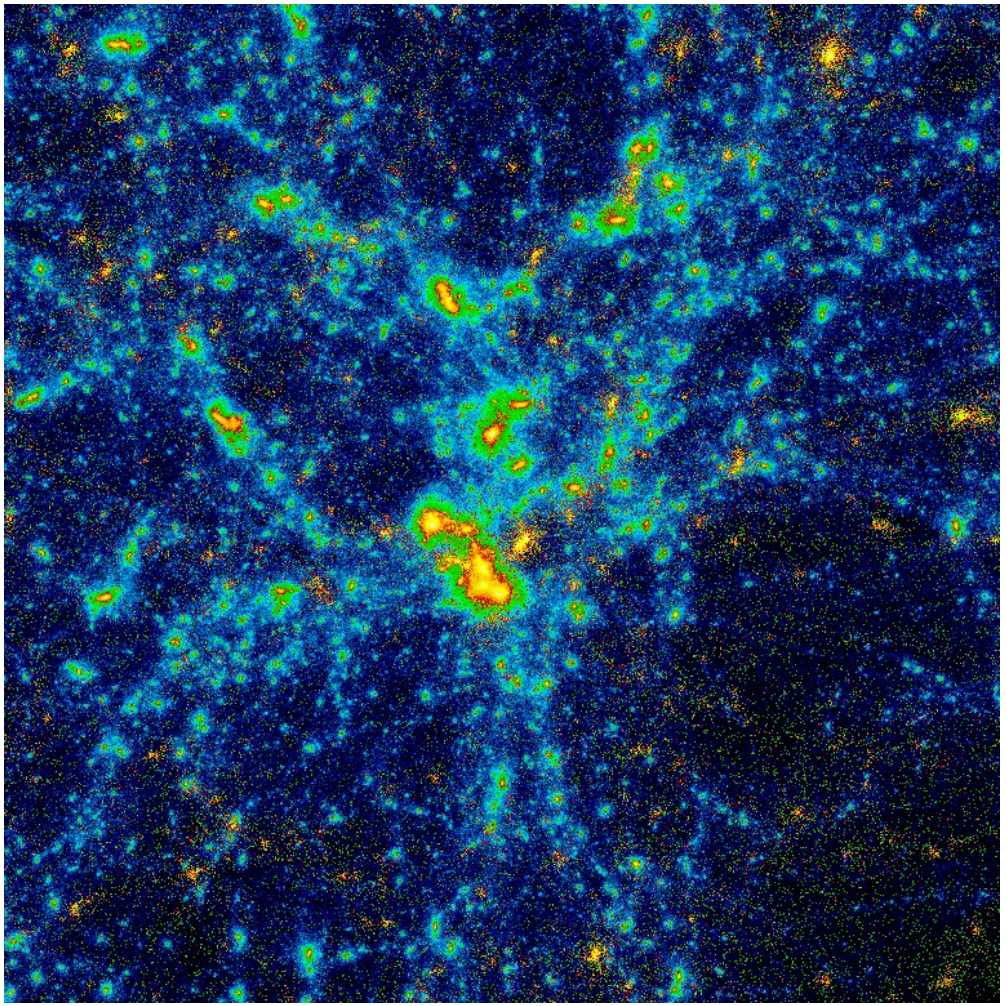
Assume you want to see if the model you used to fit the CMB data actually reproduces all the observables you have. Then you have your model and you want to “**predict**” things with your model.

This is actually an issue for many realistic models, you simply can not do the math…!

Is in these cases where you need Hardcore solution to the problems

Solve the Fluid equations? Can you?
Solve the MHD equations? In general cases?
Solve the equations of motion for a set of self-gravitating particles

One example: Solve Poisson's equation (for the problem you like...!)



$$\nabla^2 \delta\Phi = 4\pi G \delta\rho$$

Fourier magic!

$$\Phi(\mathbf{k}, t) = -4\pi G a^2 \overline{\rho}(t) \frac{\Delta(\mathbf{k}, t)}{k^2}$$

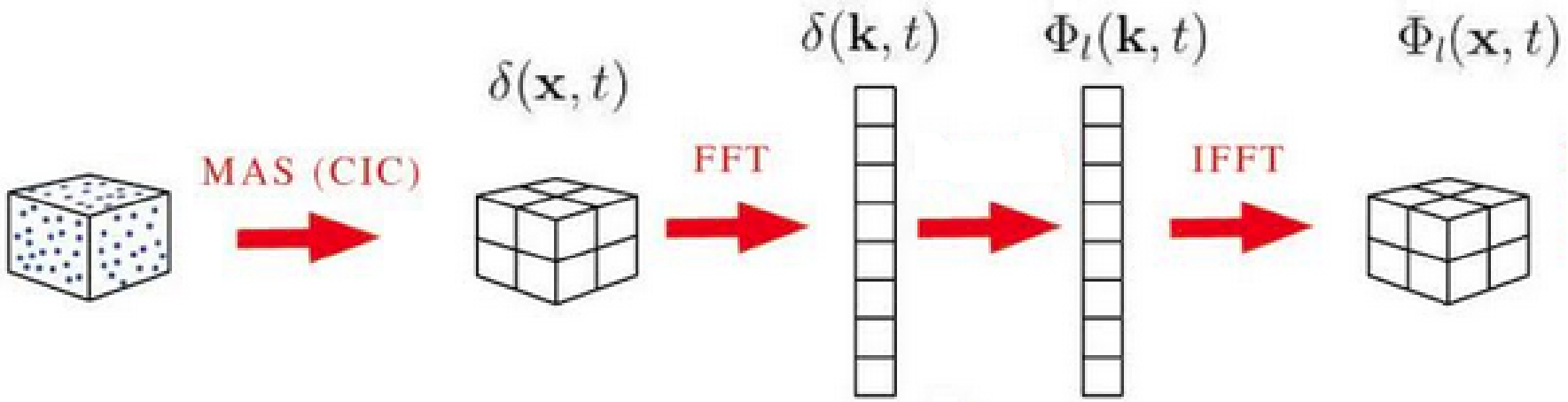
3N

2*Ng

2*Ng

Ng

3N+5Ng
(8Gb)



1) Build a regular grid and use it to build a density field from the particle distribution. For that, use a CIC or NGP scheme.

2) Apply a Fourier transform to the density field and Solve Poisson equation in Fourier space for the gravitational potential

3) Then transform back the potential from Fourier space and use it to compute its derivatives in the grid

4) Use the grid to interpolate back the force field at the position of every particle.

What are we going to do?

- We focus our attention on our problem: **SCIENCE**
- Programming is just a tool we use to get our science done
- The language is another tool, don’t get in love with just one language: *There are different tools for different problems*

Architecture of the computer

This will be our concept of computer

The best way to know how to improve the operation of any tool, is knowing how it works!

This is our working hypothesis!

Based in this premise, we will talk a little bit about very basic (dummy!) aspects of the workings of “a computer”.

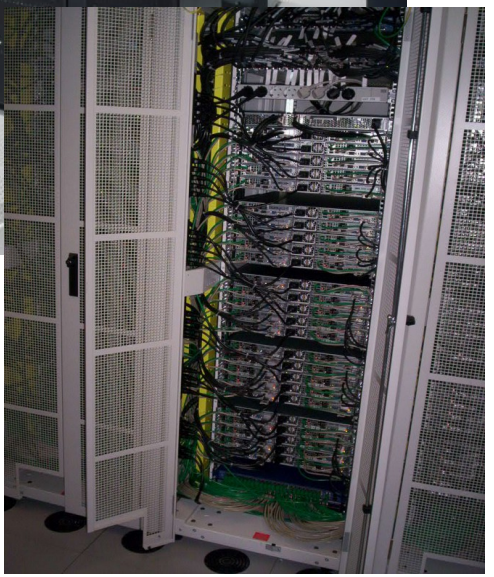
Sorry if this is too basic for you, feel free to fall to sleep for a few minutes if it is the case!


Altix


JuGene


JÜLICH
Sun
HPC – FF
JUROPA
JUROPA


Marenostrum
Copyright 2005. Barcelona Super




WHAT DO WE WANT?
FASTER RUNS
DO YOU KNOW HOW?
....
imgflip.com

My opinions:

HPC is nothing more than applying common sense to the development of your code, once you know the basics of the workings of the computer!

You don’t need to have a cluster or a fancy computer to apply HPC. Do it for your thesis, it may help to get you through faster!

You could see that “*this*” problem you considered **was to big** to be solved with your resources, may be “solvable”. *May be you have not optimized ...*

Our computer has these basic three components

1) Processing units
2) I/O units
3) Storage units

All of them work together connected in the board to make your job run the way you like.

The performance of the job depends on the job, but also on the way you use each of these resources.

The Computer Processing Unit

- This is the heart of the machine…
  - *you work for this guy while developing!*
    It will work for you latter!

- Here you do all operations of the system: SO operations, applications, and of course, run your numerics!
- The ideal CPU processor: the fastest one
  The largest the CPU clock frequency, the largest the speed of the calculations
  - Increase in the speed in current CPU units depends on the number of electronics (transistors) in the chip.
    Technical restrictions to get as many as you like!

The Computer Processing Unit

Core, core... core?

A way to “speedup” performance of the CPU is to “divide and conquer” including more transistors in connected CPU units

Single core: *Just one processor (this thing does not exist anymore!)*

Multi-core: *This is one BIG chip with two or more physical CPUs connected between them.*

Multi-Thread: *Multi-Threading is not the same as Multi-Core. Multi-Threading is a way to use, in the same CPU “streams” in a way that might seem to have two CPUs reducing IDLE time.*

Speed up of about 30%

Storage

You have long term storage and short term storage

Long term storage devices

- Low rate of IO operations (10, 100 or more times slower than RAM)

Try to get something from ARCH!

- Large capacity
- “Permanent” storage of the data

Hard drives

Tape drives

Storage

Short term storage

- Random Access Memory
- CPU registers

- Both types of storage are temporal, they have limited capacity.

  They are there to store data used during execution.

- Size of RAM memory depends on memory frequency. Large in size but slower in speed than registers.

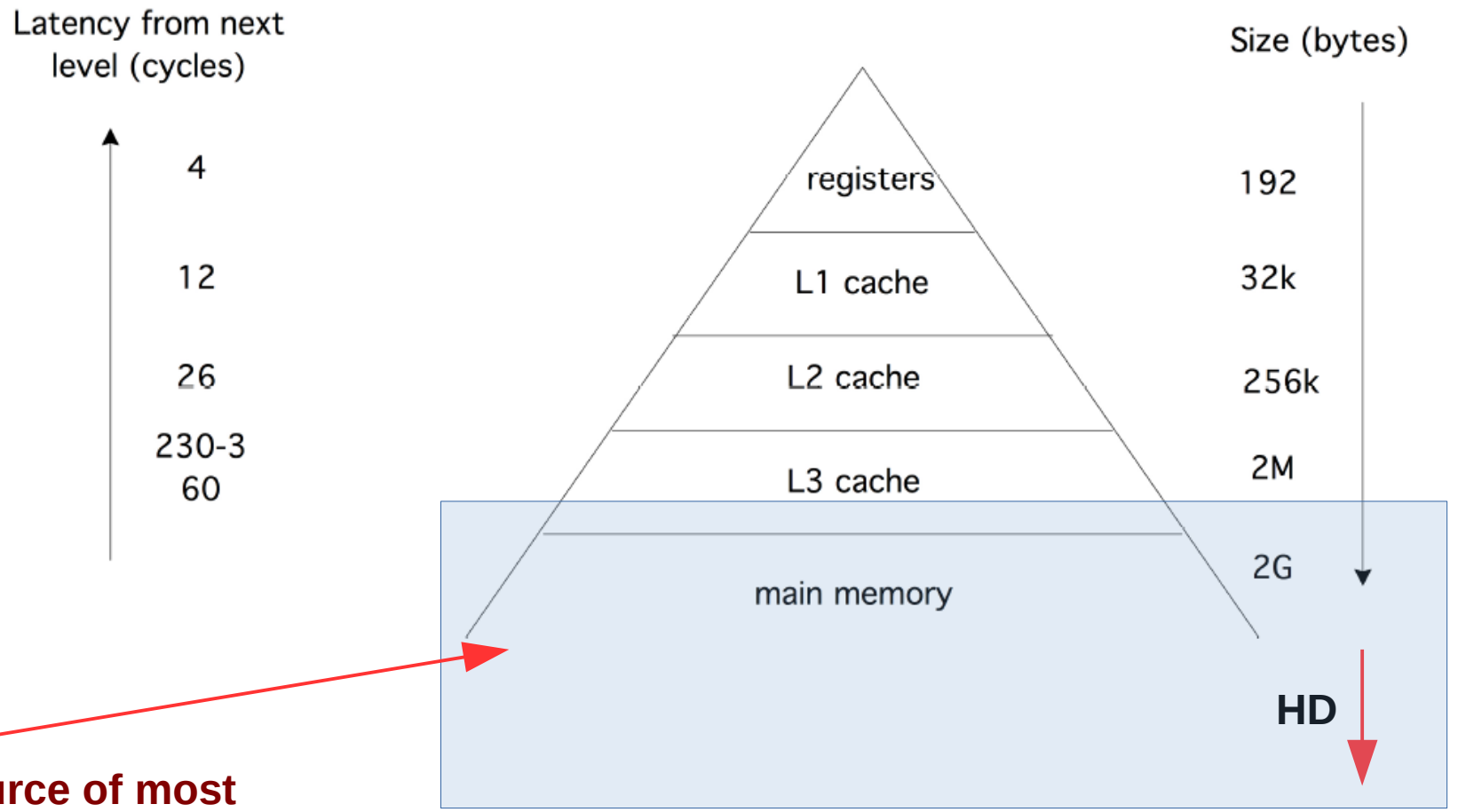
  Also on $$$

- Size of cache registers depends on the size of circuitry, but are faster (closer to CPU) than RAM memory

The ideal sort term storage: Large capacity, large speed.

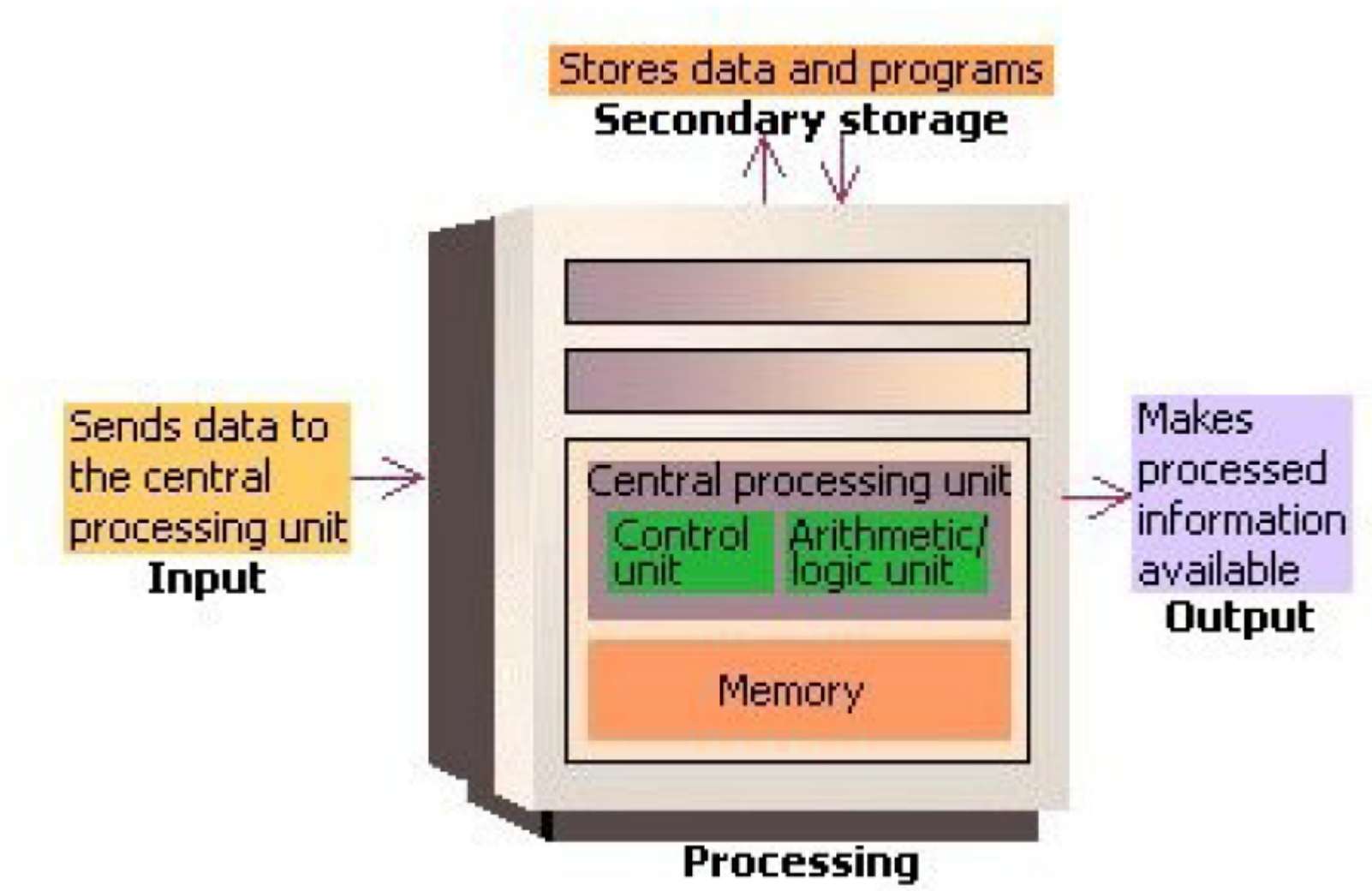
The problem? $$$$

Memory hierarchy!



This is the source of most problems in scientific computing

How the computer does his job?



Control Unit does not execute, it drives the execution

Control unit drives the execution:

- *Takes data from the memory*
- *Manages the execution of pieces of code*
- *Manages the operation of registers*

Arithmetic Logic Unit executes instructions (code)

ALU is in charged of the execution of instructions

- ***Arithmetic operations implemented in the ALU:*** *addition, difference, product, division*
- ***Logical operations:*** *=, >, < (and combinations!).*

How is the code executed in the CPU?

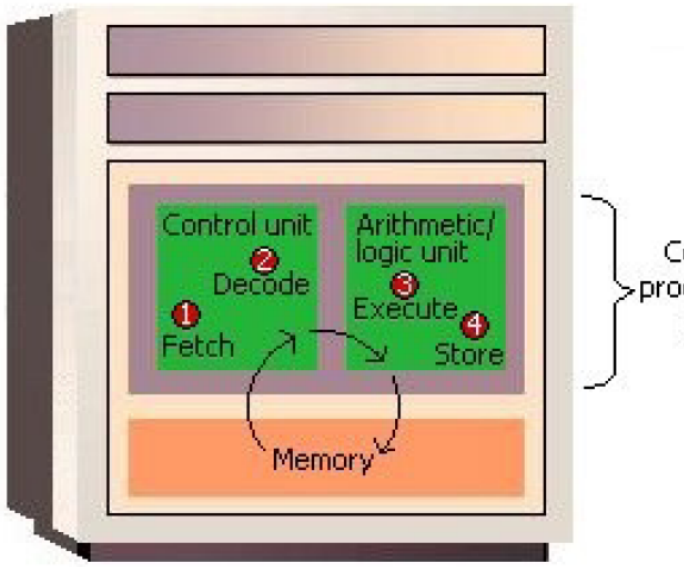
1) First the executable is loaded to memory and starts execution. Memory registers and Cache register are created (./exec)

1) Control takes instructions to memory

2) Control decodes the instructions for the execution in the ALU (Instruction)

2) Control moves the required data to the ALU for the execution (Instruction)

3) ALU executes the instructions (execution)
4) ALU the returns to registers, and they start to move in the hierarchy of memory "like a bubble!" (execution)



Control manages the way the data is moving across the storage hierarchy (Cache, RAM, HD, stdout, etc)

Intensive or extensive problems

Intensive computations

- Low memory requirement
- Required CPU execution
- Requires fast execution times
- Speed of data transfer between CPU and RAM is not a constraint

Extensive computations

- Extensive use of memory
- Equal or lower CPU requirement
- Data transfer between RAM and CPU really matters

Floating point operations

- Much of the power of the CPU is invested attending the requirements of floating point operations

(They are more interesting for us, they use more memory than integer operations, they require more operations)

- CPUs come with a finite set of Float Point Operation Units (FPU) that allow the simultaneous execution of multiple float point operations (flops)

- Of special interest are the units devoted to addition and product, they are more frequent than division units.

*CPU is more efficient a doing + and * than /*

Floating point operations

- There are also fused MultiplyAdd, doing

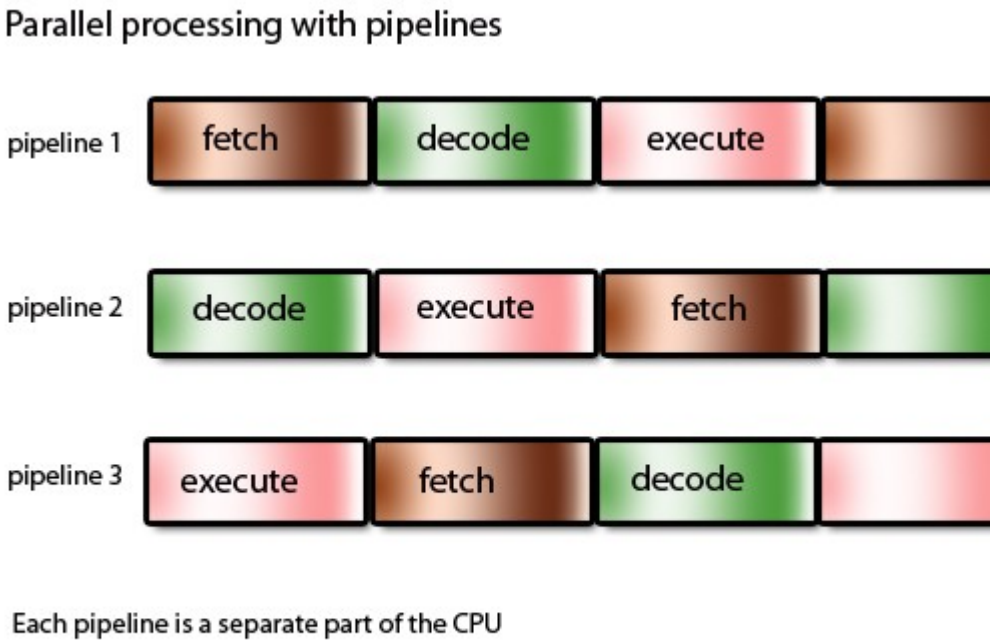
$$x=x*a+b$$

is faster than doing

$$c=x*a$$
$$x=c+b$$

*It is not the same for divisions, in the CPU divisions are around 20 times slower than * or +*

Pipelining



Assume that an operation like addition can be split up in a sequence of simpler operations that can be executed in individual FPUs

If you can have 4 independent additions, in sequence one FPU can handle 4 operations in a cycle

Net effect → ~1 operation/cycle

This can only take place for independent operations

- Pipelines stop when the code has conditions

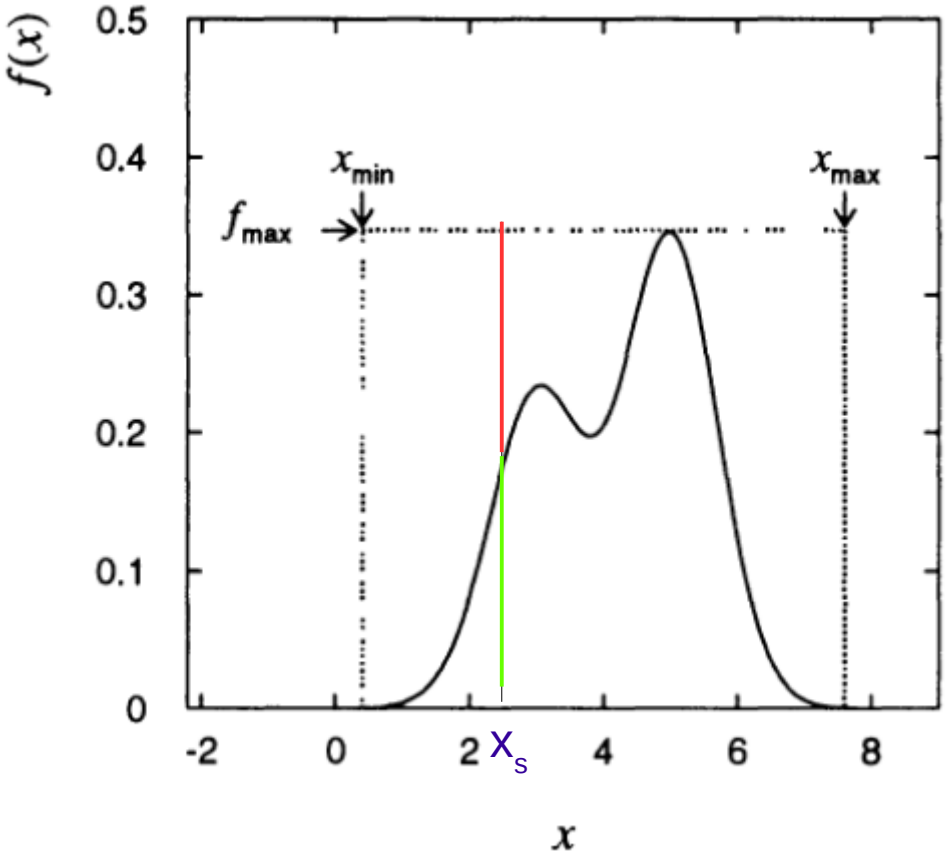
The CPU won't stop the execution and continues executing the code assuming that the answer of the conditional is positive.

Once there is an answer for the conditional, the pipeline continues (if the answer was indeed positive) otherwise, the full stream is repeated!

Dependency, conditions reduce the efficiency of pipelines.

Example:

Simple rejection algorithm for generation of random deviates



1) Generate a random number x_s uniformly distributed between x_{min} and x_{max}

2) Evaluate $f(x_s)$ and generate a random number *u* uniformly distributed between 0 and $f(x_s)$

3) If $u < f(x_s)$ accept x_s, otherwise, repeat.

If you already know the region in red (rejection) is larger than the region in green (accept), you know the condition $u < f(x_s)$ will be in disadvantage.

Choose in your implementation the way you write the conditional.

Get the most probable!

Optimizing the use of cache registers

The CPU has a very limited number of registers (of the order of 50)

```
a = b + c
d = a + e
```

In this case the optimization is native. But here are dependency issues!!
However variables stay in cache during operation!

```
t1 = sin(alpha) * x + cos(alpha) * y;
t2 = -cos(alsph) * x + sin(alpha) * y;
```

Here you have made things independent, at the time that all variables stay in cache registers

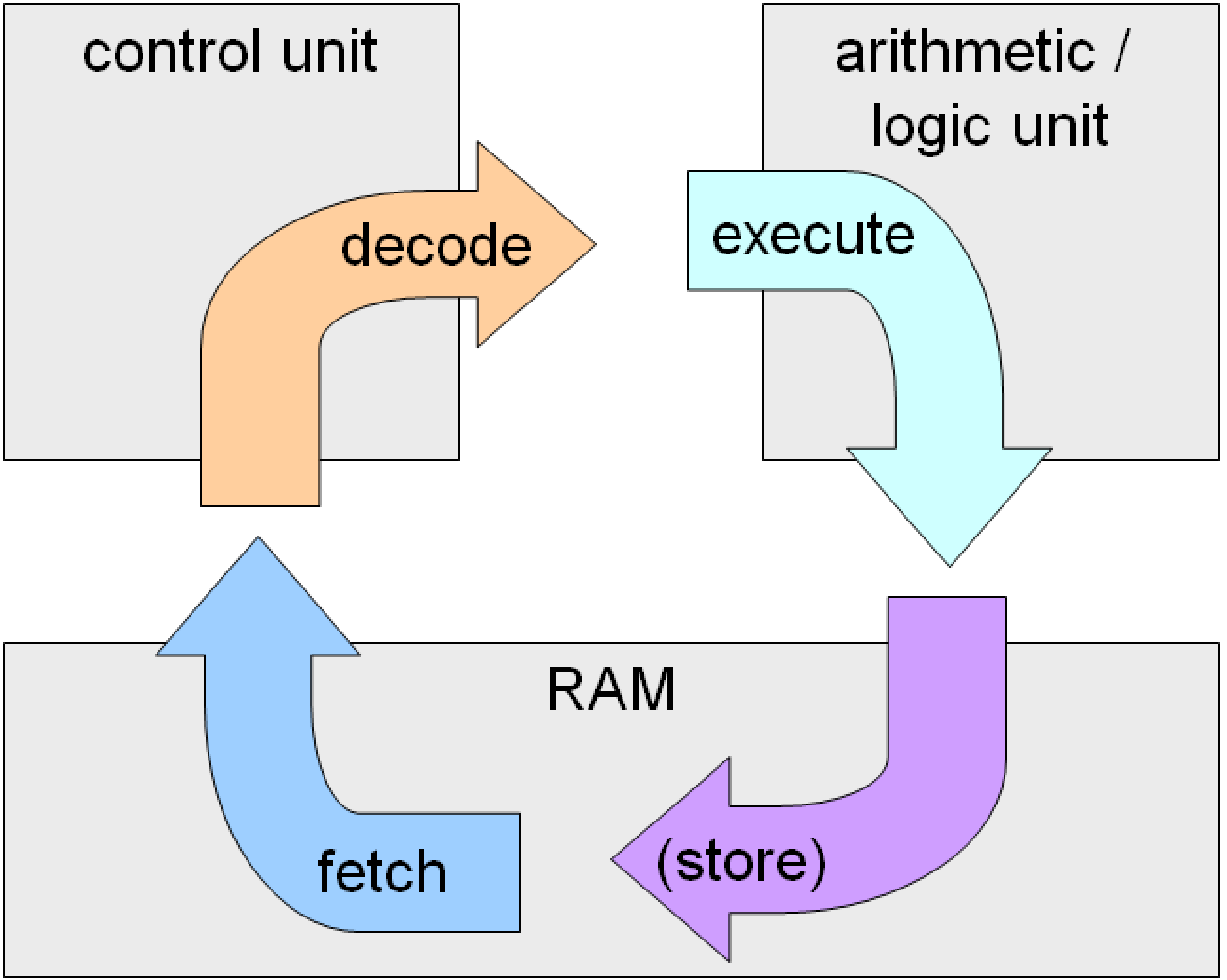
```
s = sin(alpha); c = cos(alpha);
t1 = s * x + c * y;
t2 = -c * x + s * y
```

How can I make my code run faster?

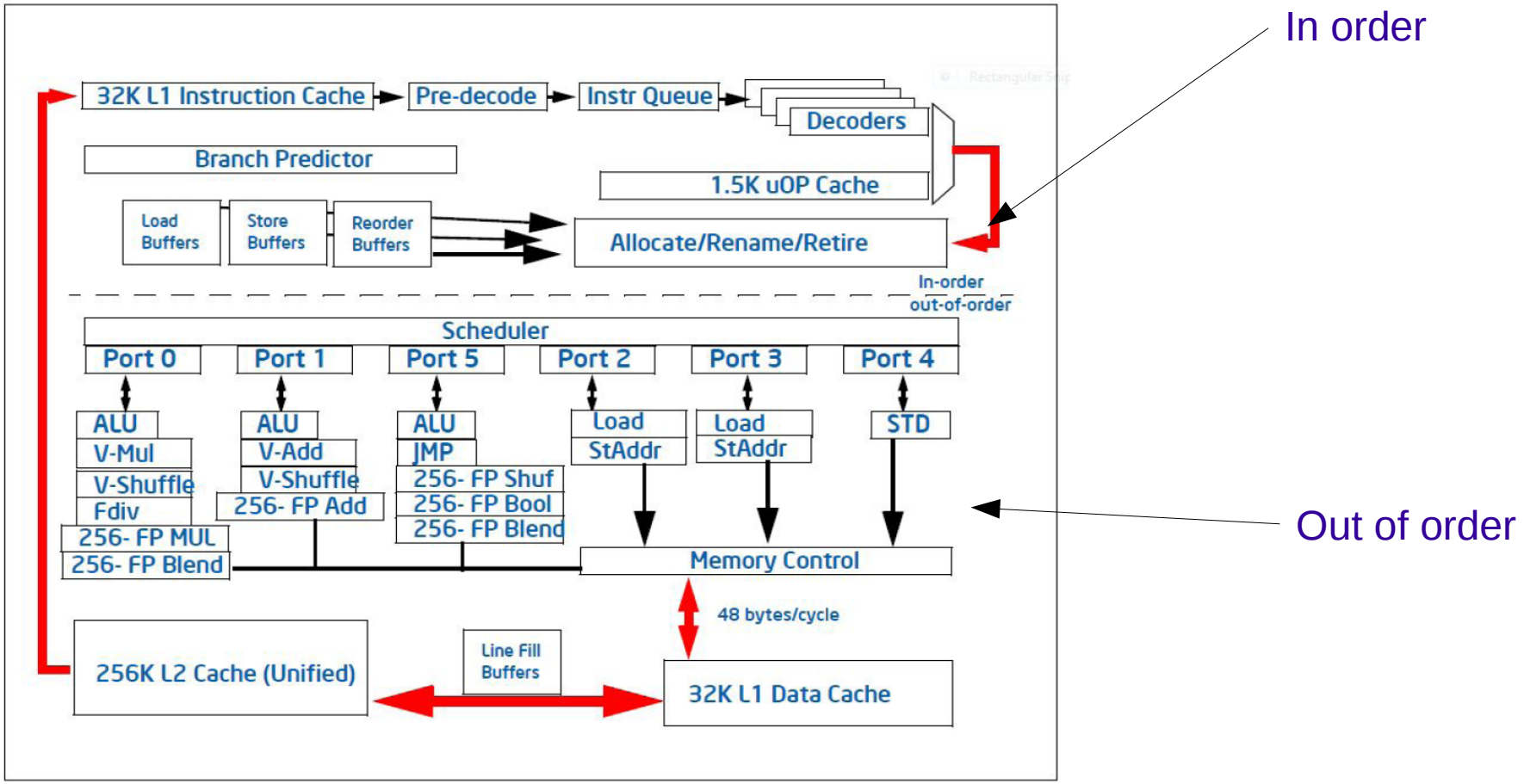
- Avoid as much as possible the use of IO to HD
- Avoid as much as possible the use of division. Especially in long loops!
- If possible, add+multiply (*0.5 is much than /2)
- Try as much as possible to implement independent operations.
- Reduce the use of conditionals (if) or study your problem to give priority to the most probable condition
- Optimize the use of memory: use the variables you need for the problem you have

  This actually reduces the use of RAM and speedups calculations an data transfer

- Keep spatial and temporal locality in the data to optimize cache access


control unit
arithmetic /
logic unit
decode
execute
RAM
fetch
(store)

Executions In order or out of order?



- Out of order optimizes the use of CPU time reducing IDLE, but it is at expenses of energy and Circuitry.

- Out of order executions can only be executed if the result of the operations are independent.

Some simple messages...

- ***Clarify: memory is not the same as storage HD***
  - *Memory is of temporal use*
  - *Has very limited capacity*
  - *Memory can have large IO velocity*
- *The larger the size of the memory, the larger the size of the problem you can handle*
- *It may be important to keep in mind synchrony between RAM chip speed and CPU speed...*