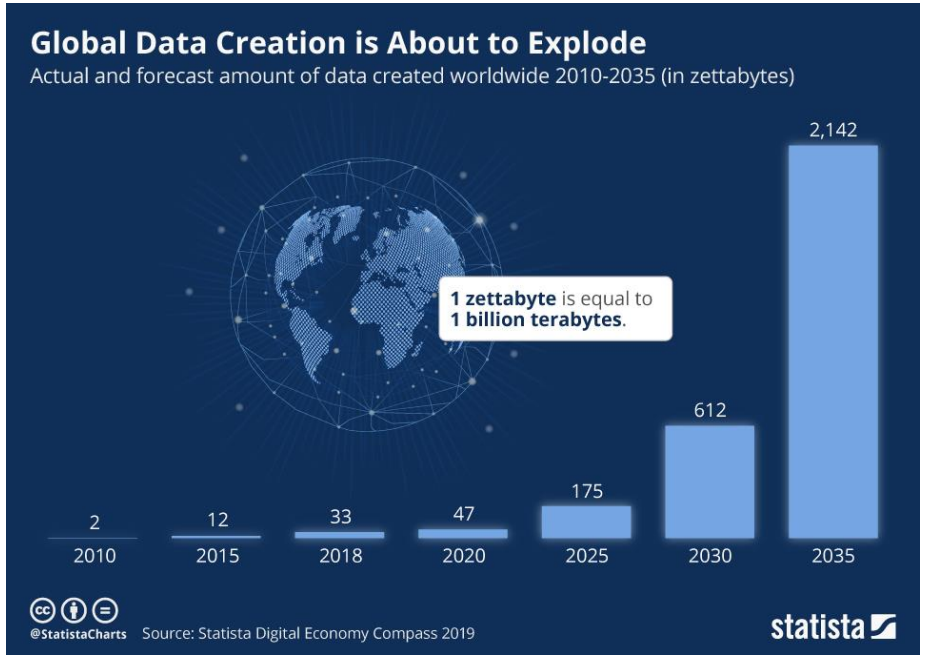
# Using GRICAD Ciment computing servers with R

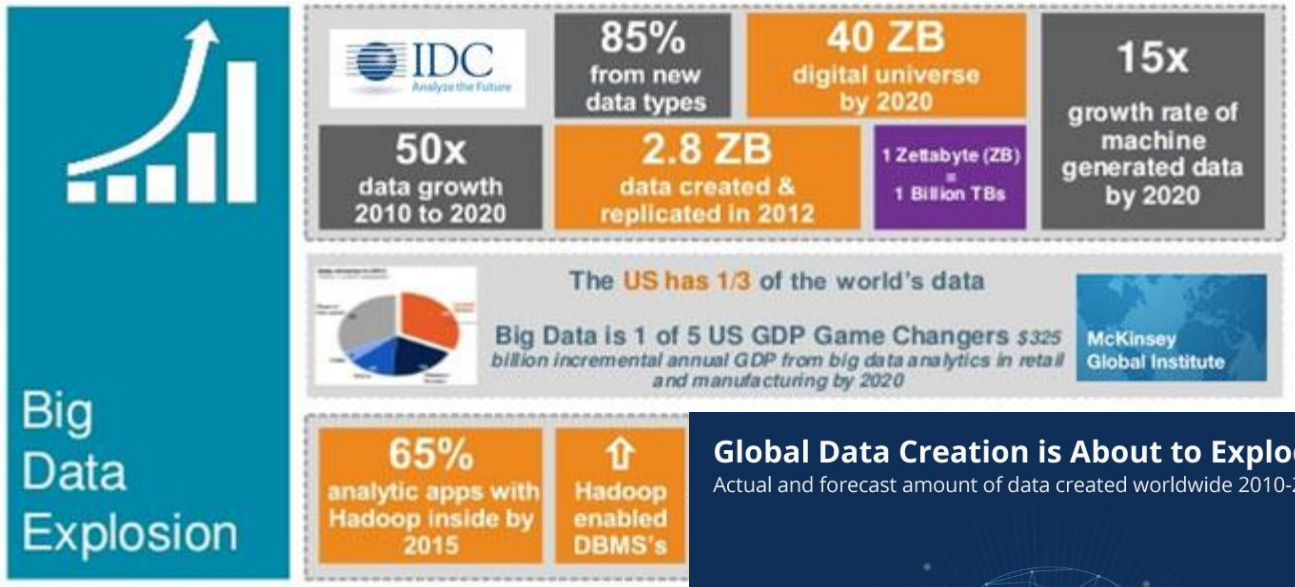
*Maya Guéguen*

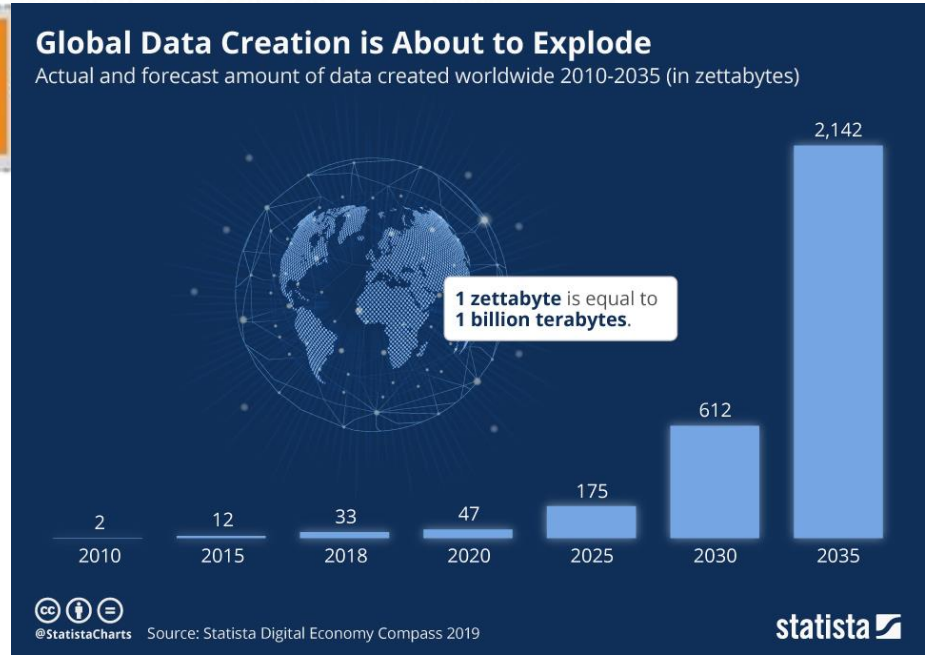
*LECA - Grenoble*

# Present context : data

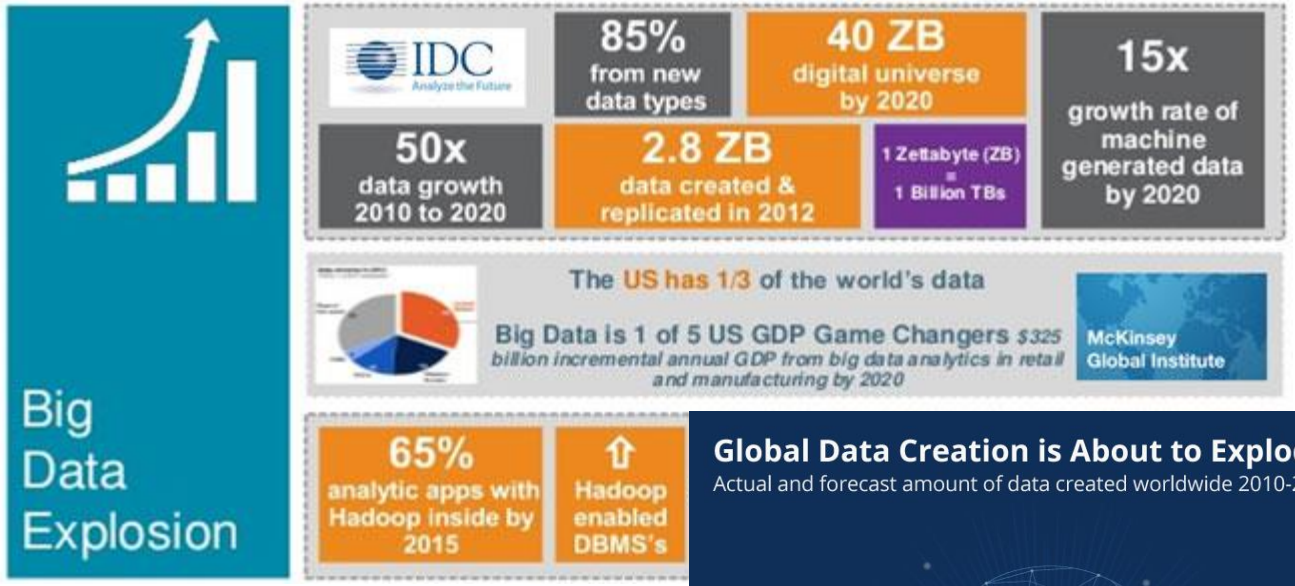


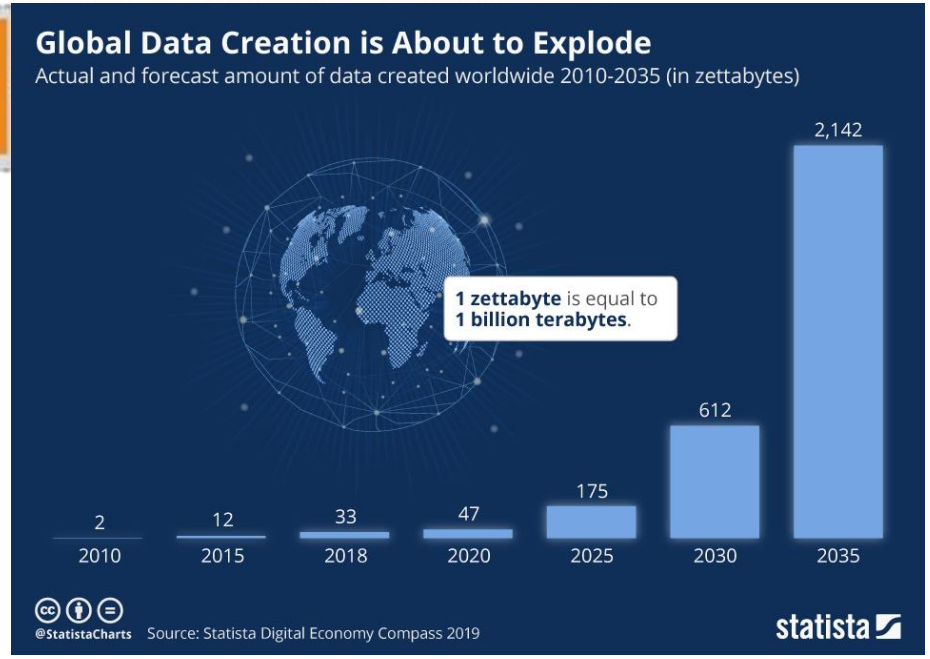
# Present context : data

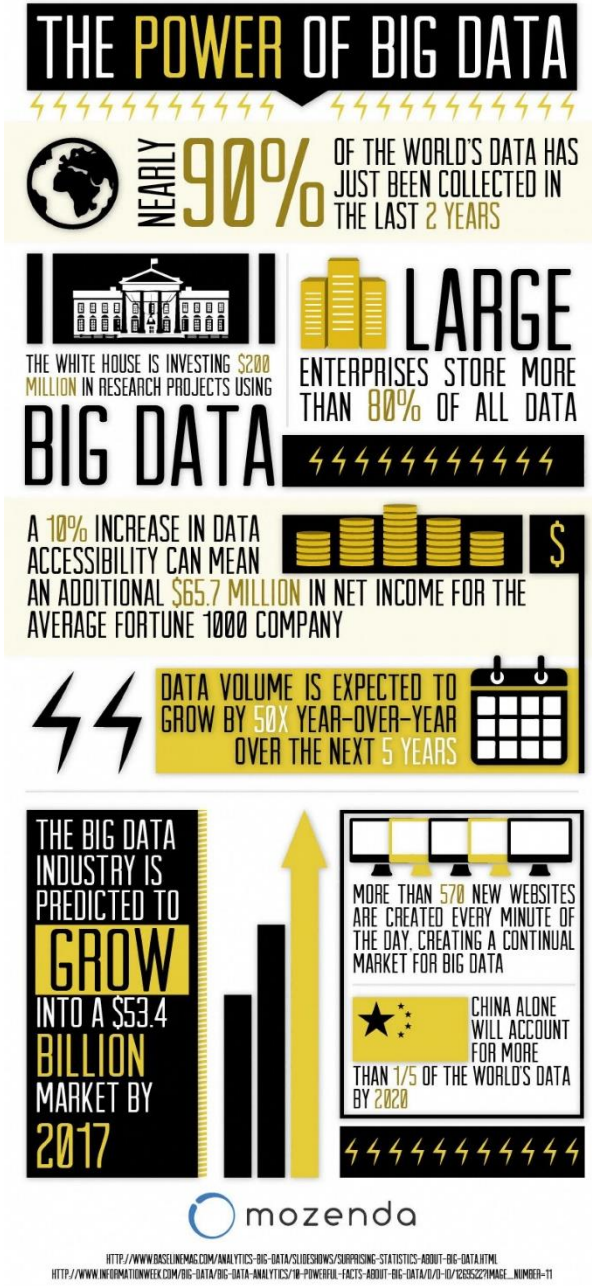


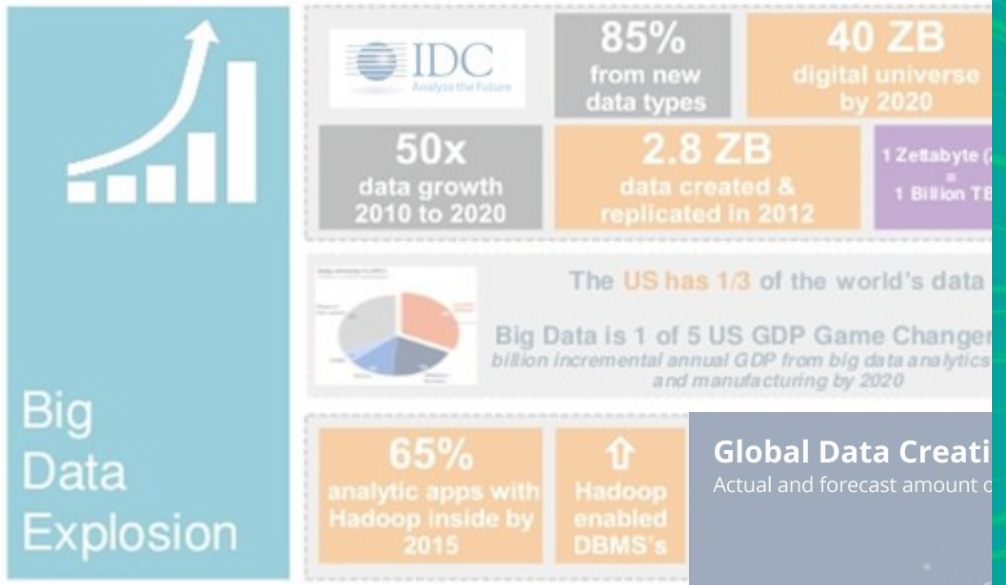
# Present context : data



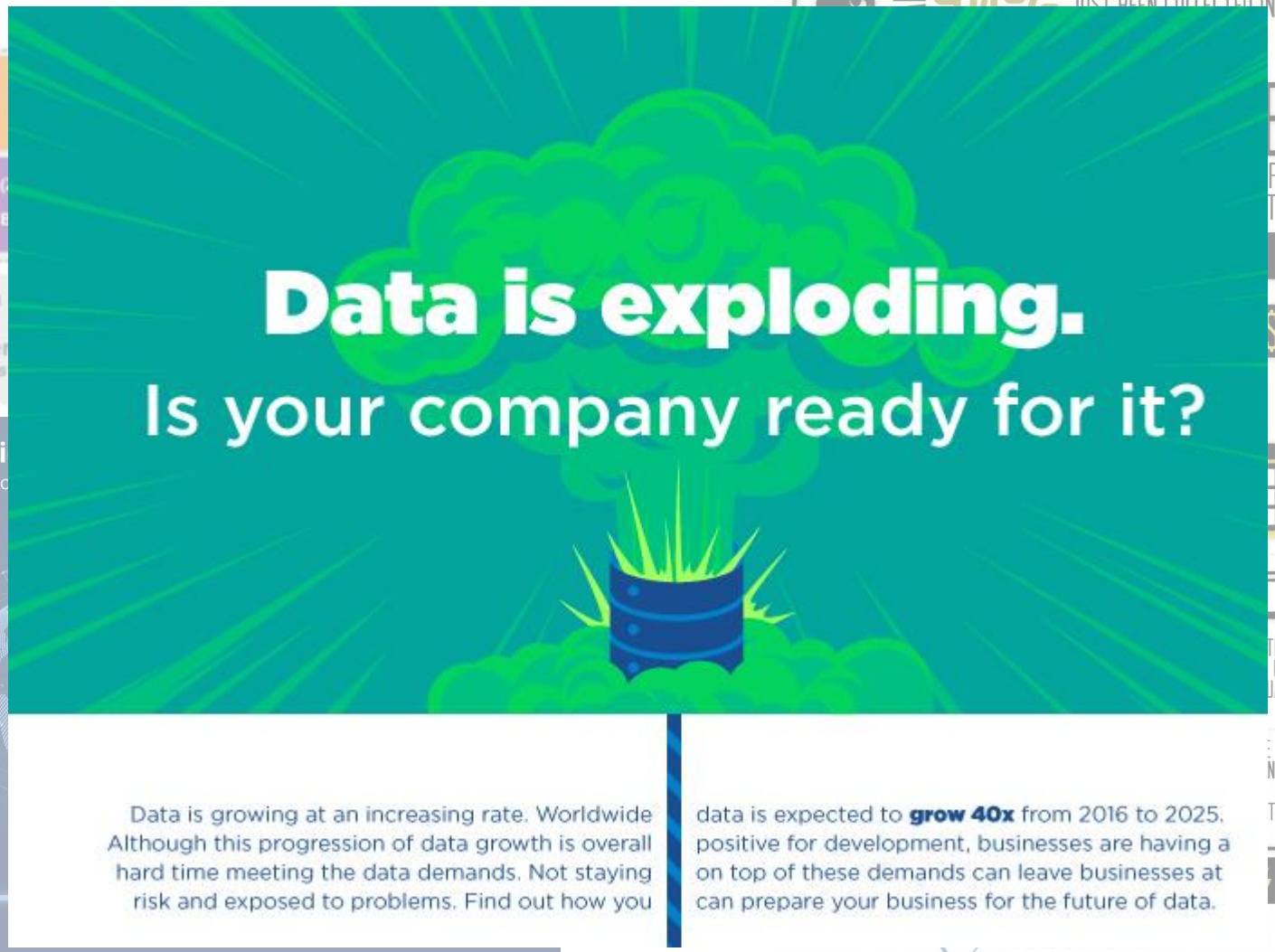



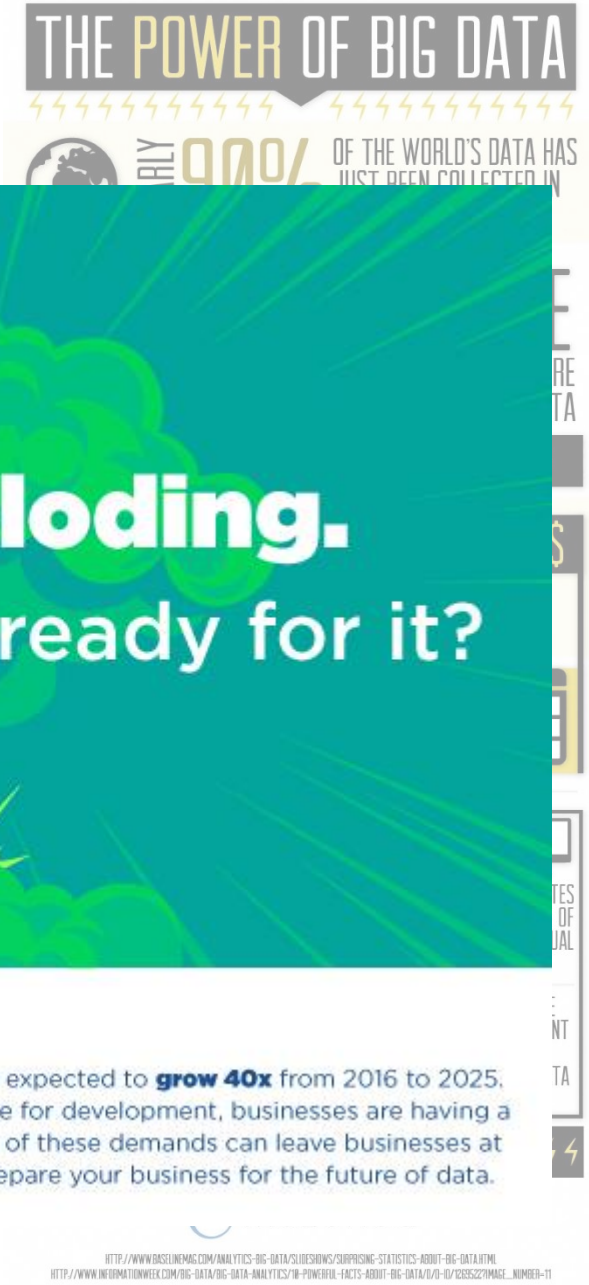



# Present context : data



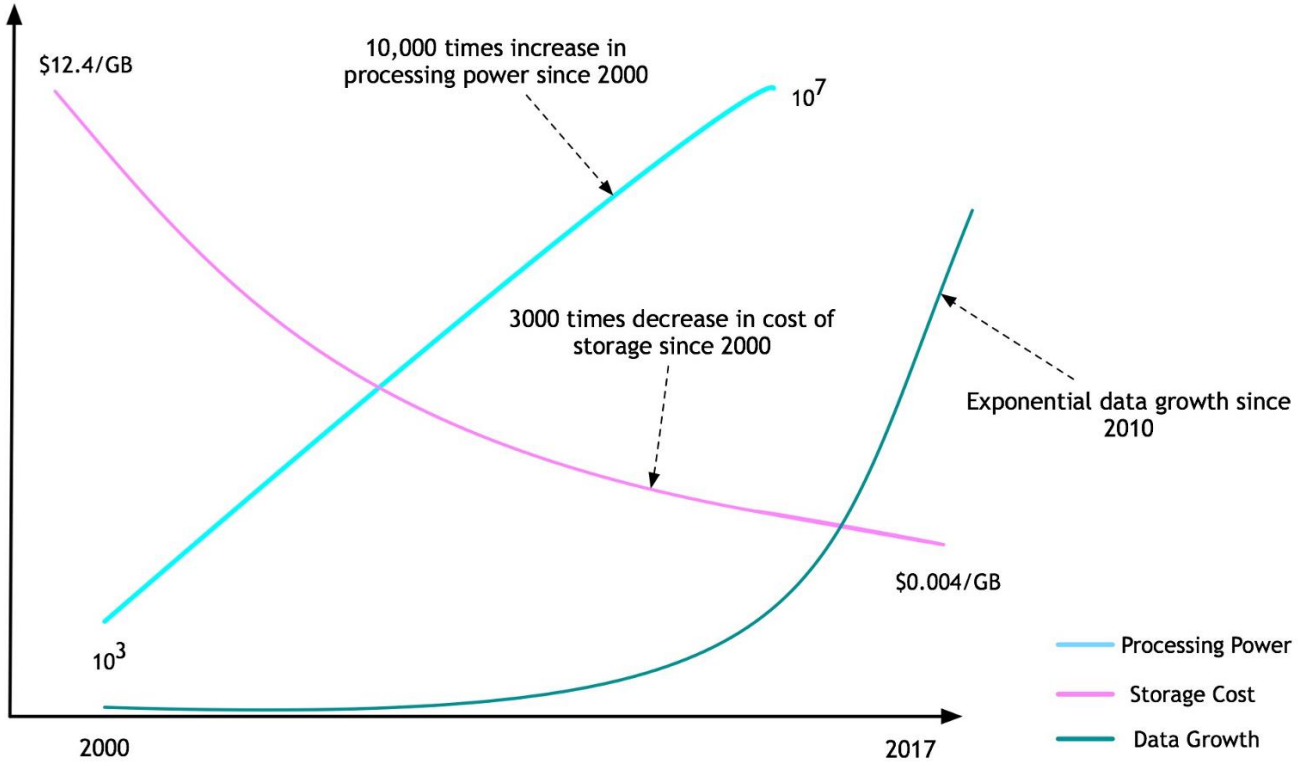




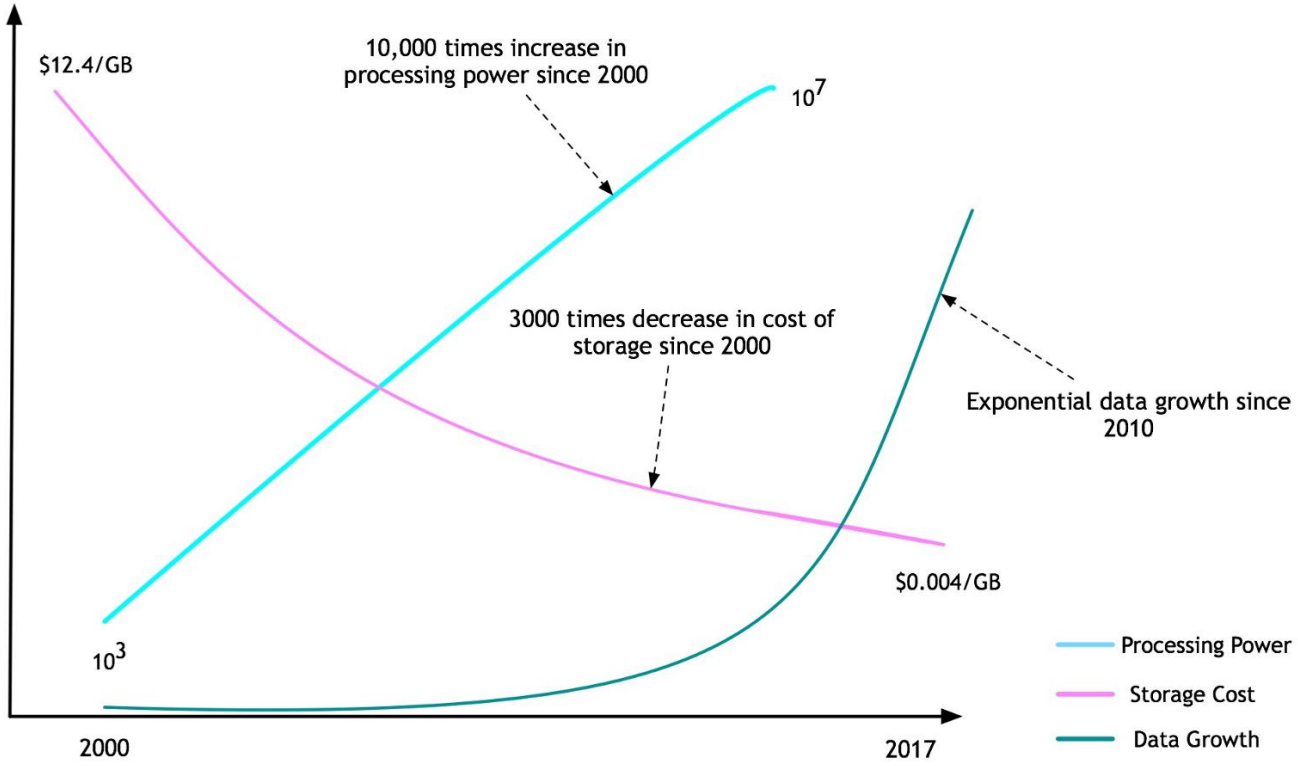
# Present context :

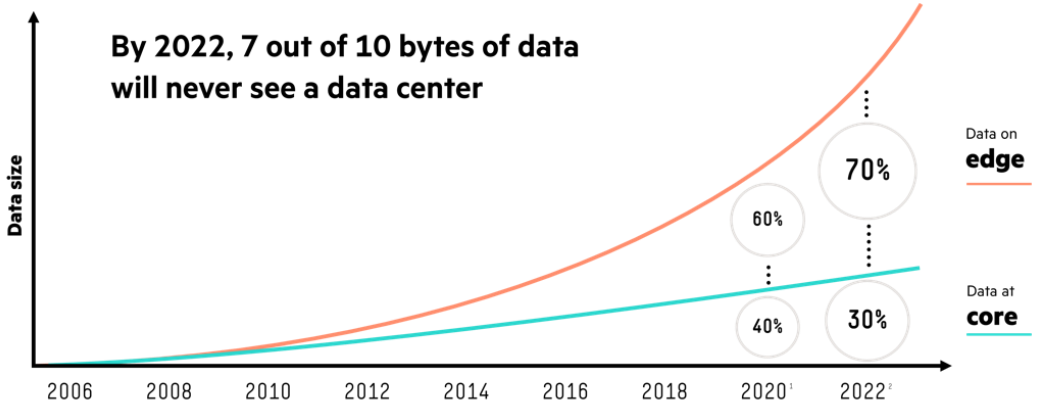
## processing power and price



# Present context :

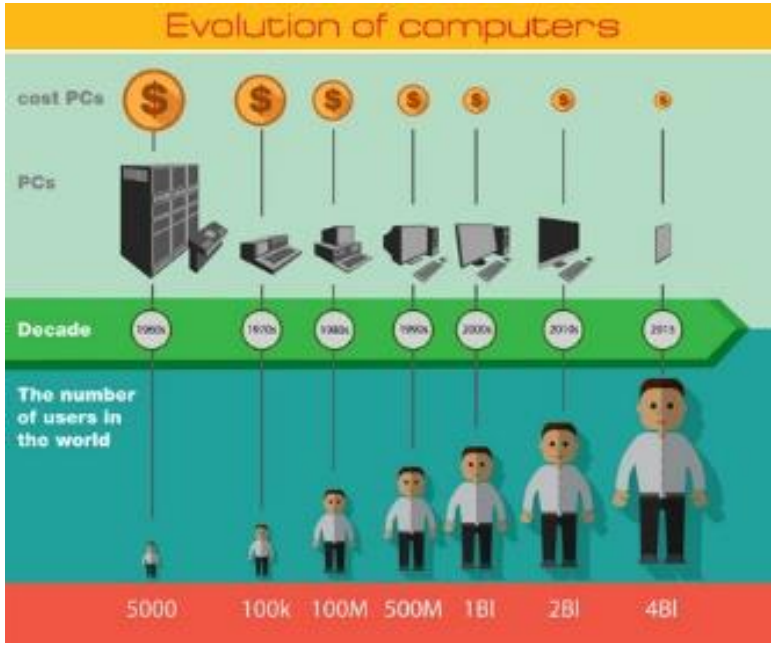
## processing power and price

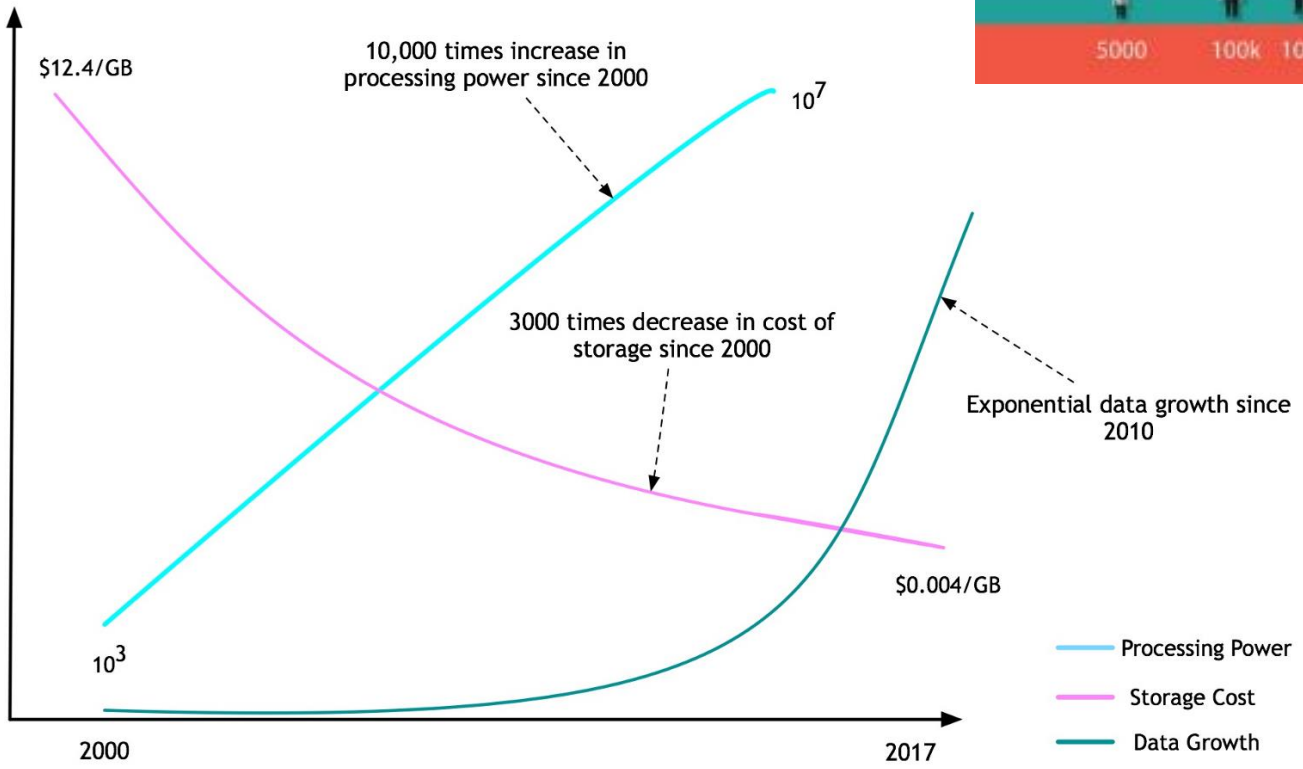


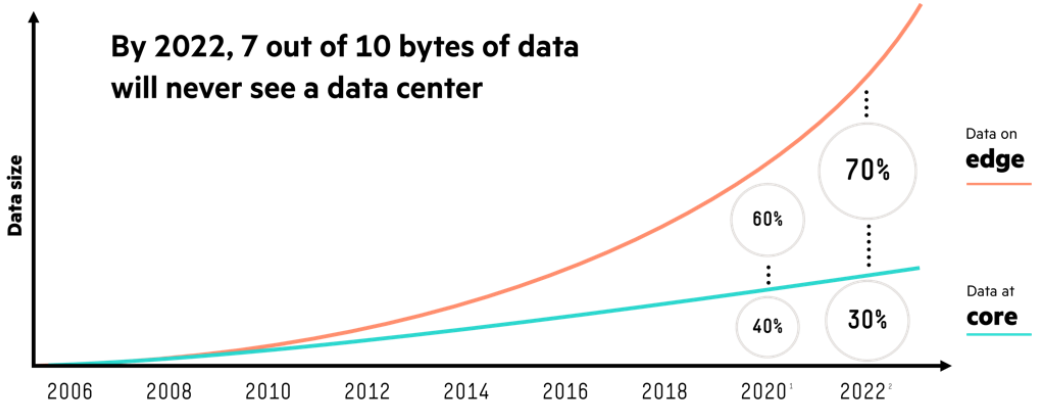



# Present context :

## processing power and price

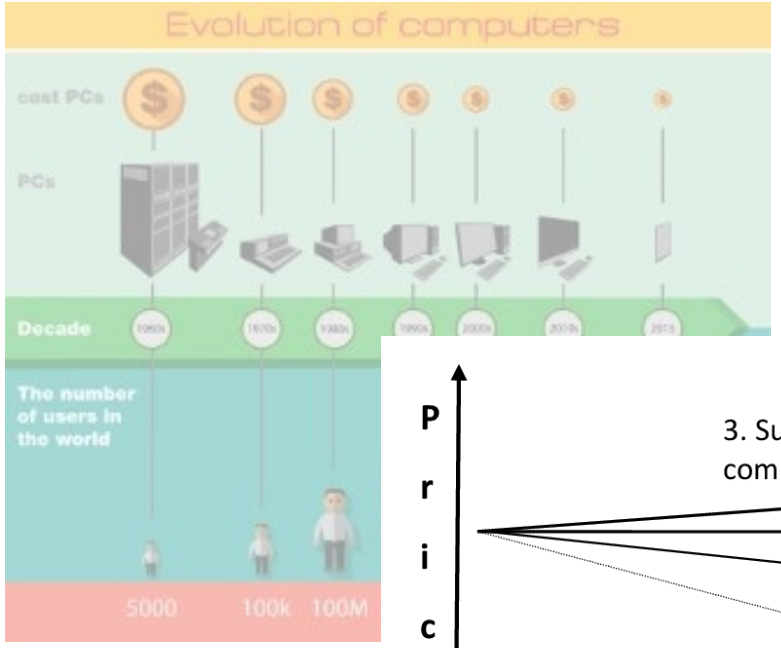


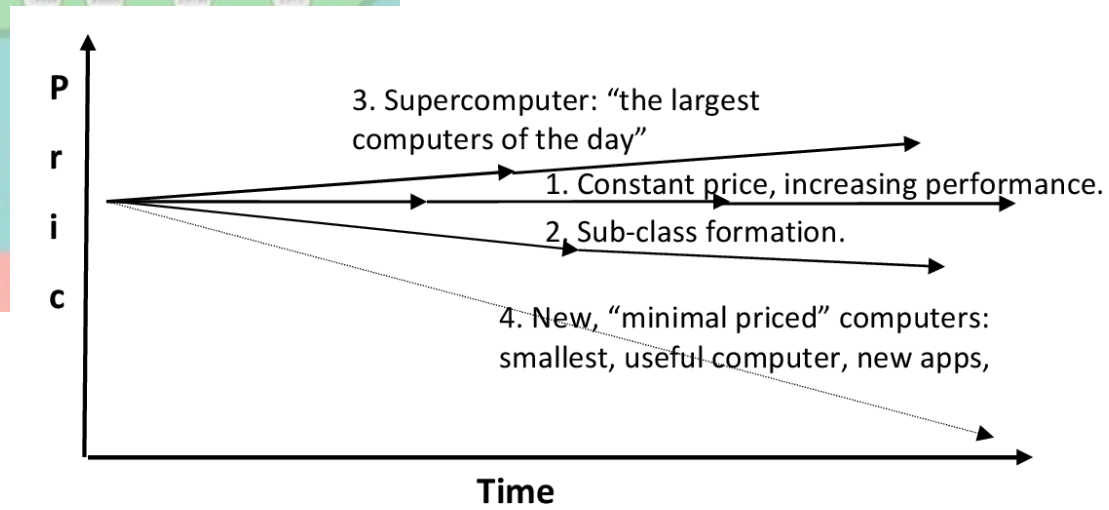



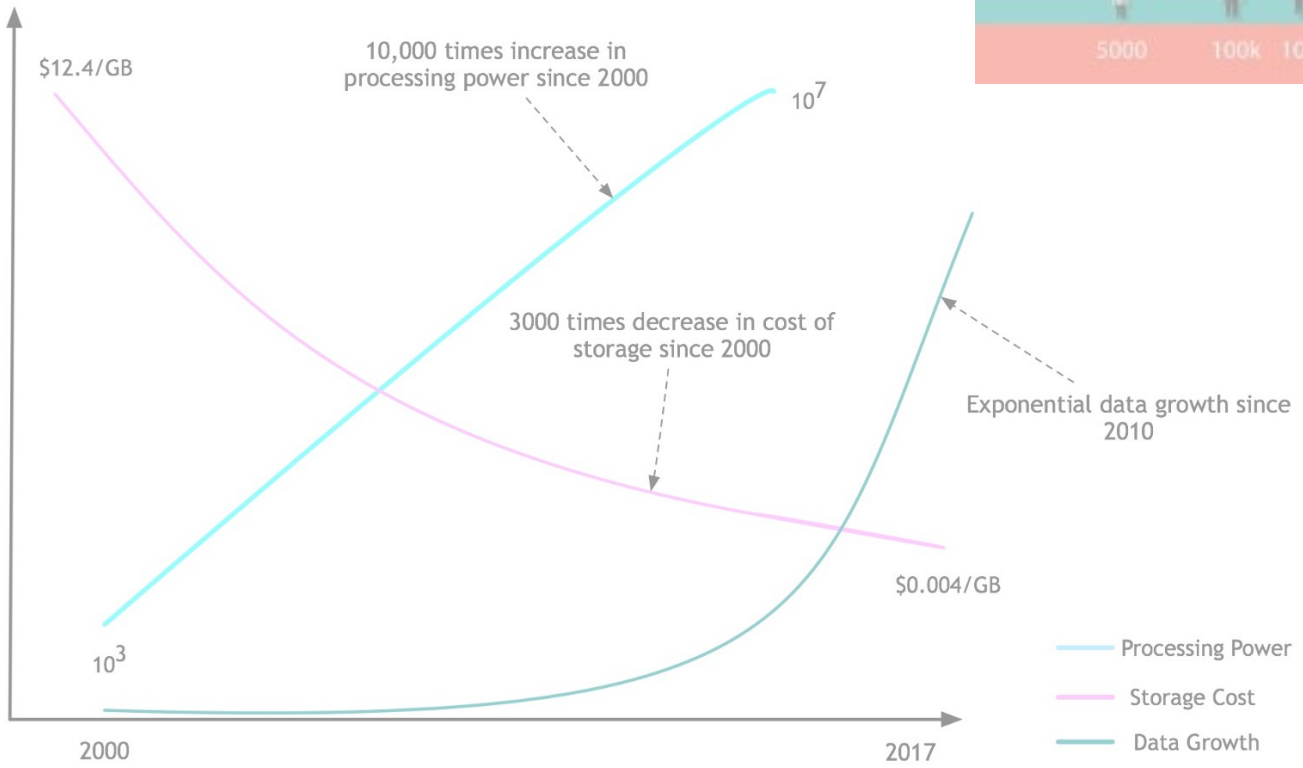



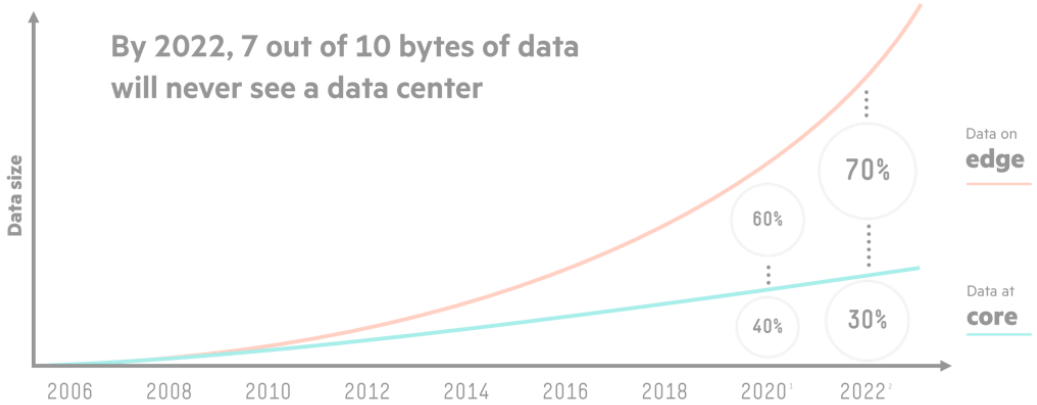
# Present context :

## processing power and price

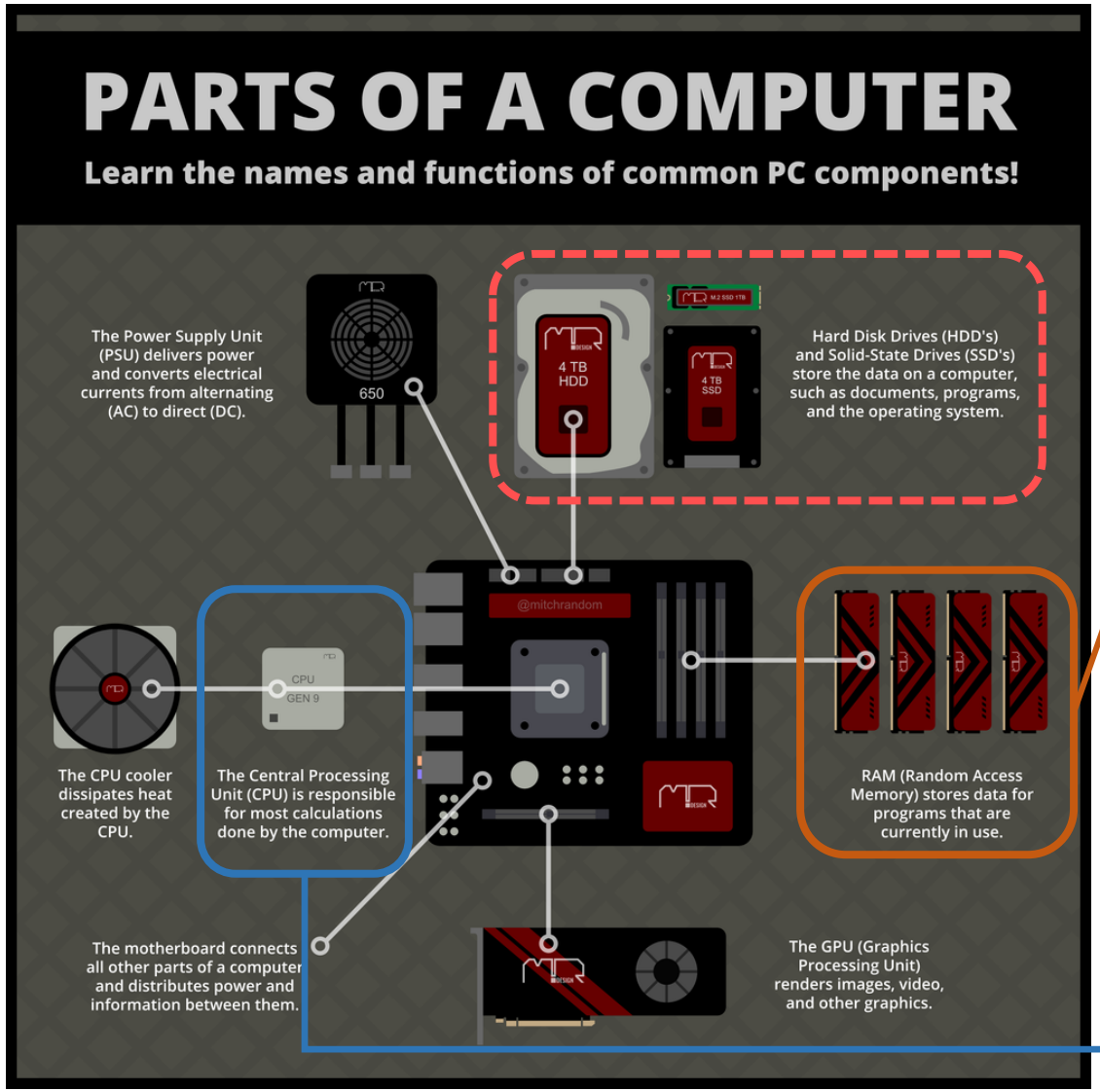


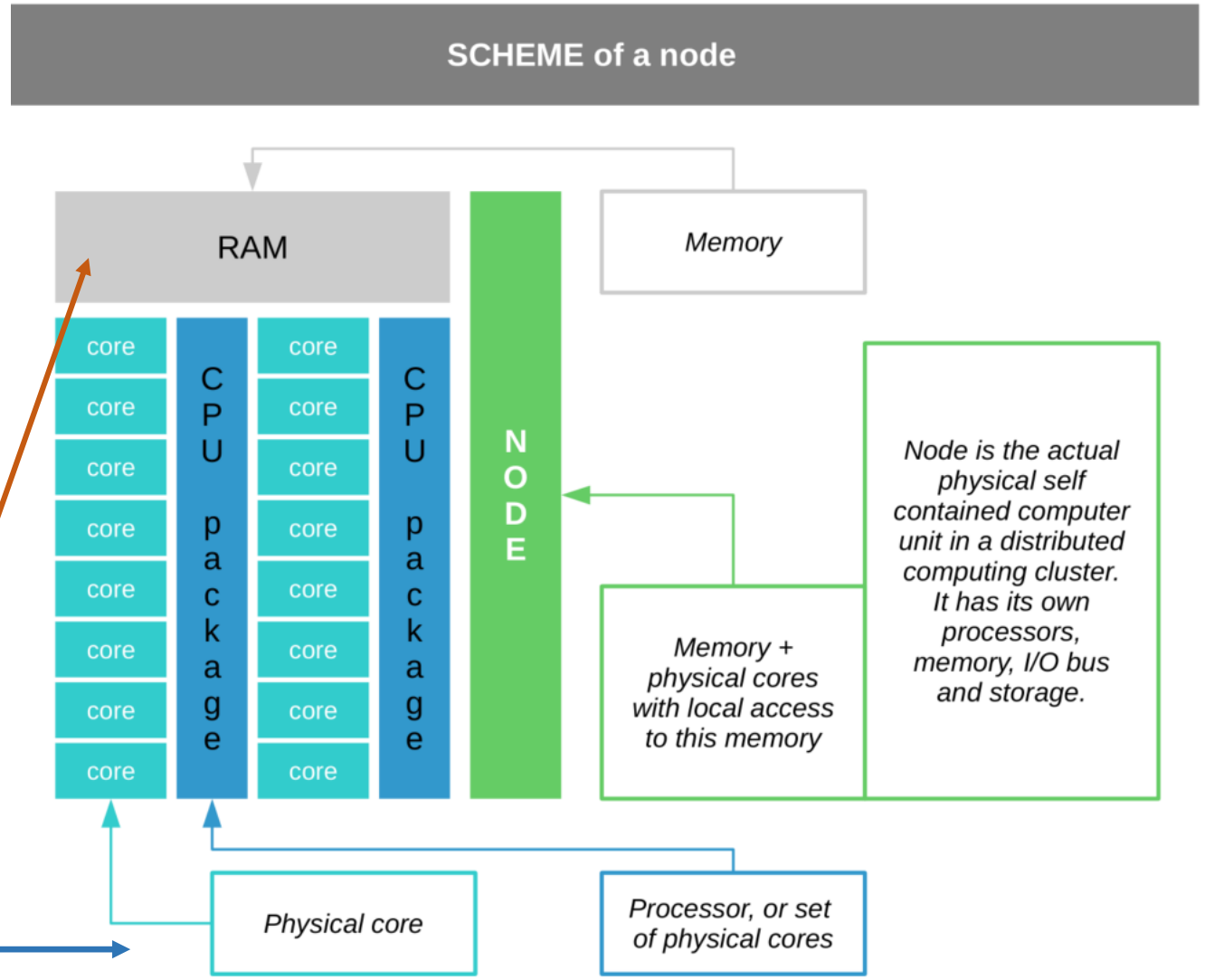



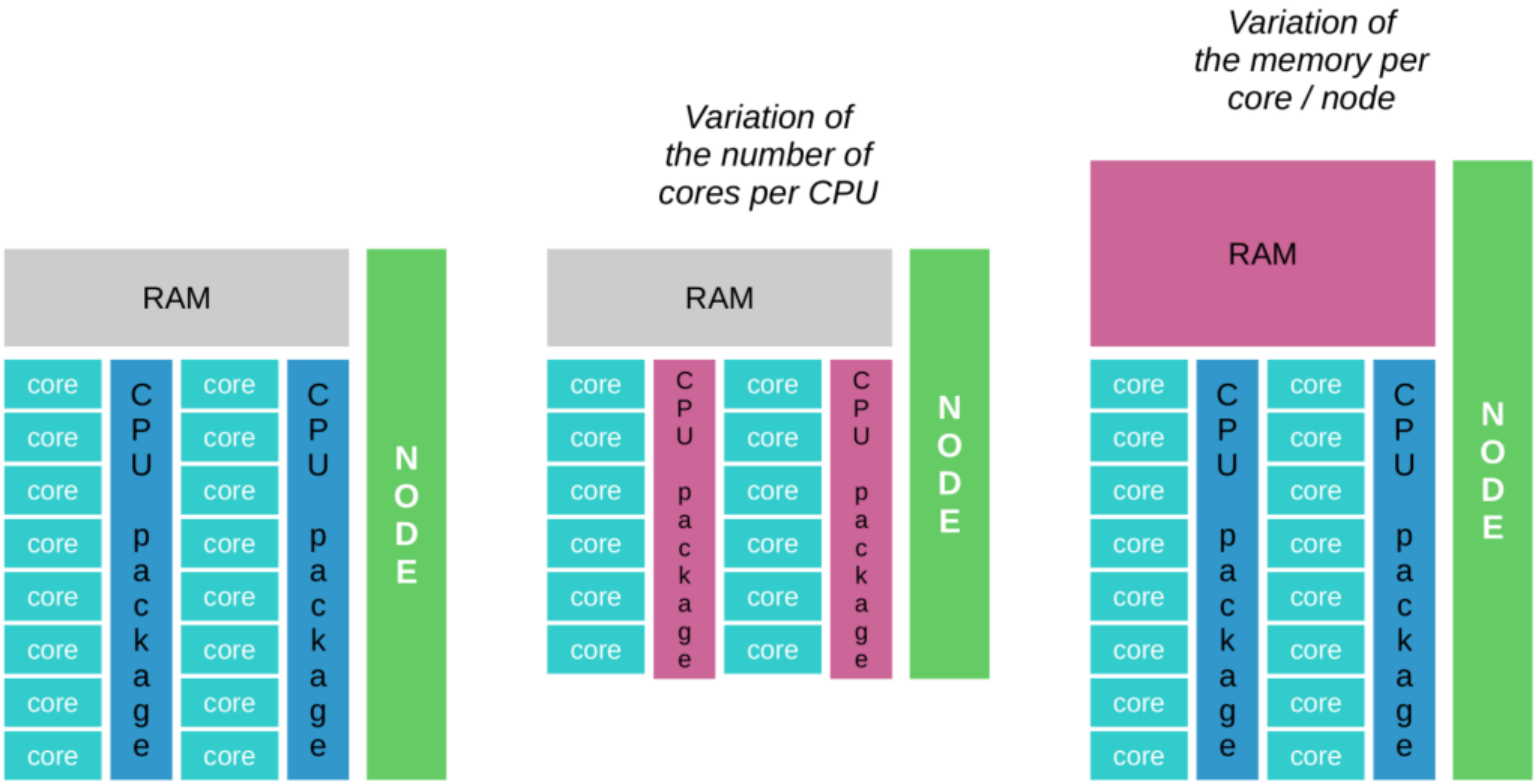



# Why would I need a supercomputer instead of my laptop ?

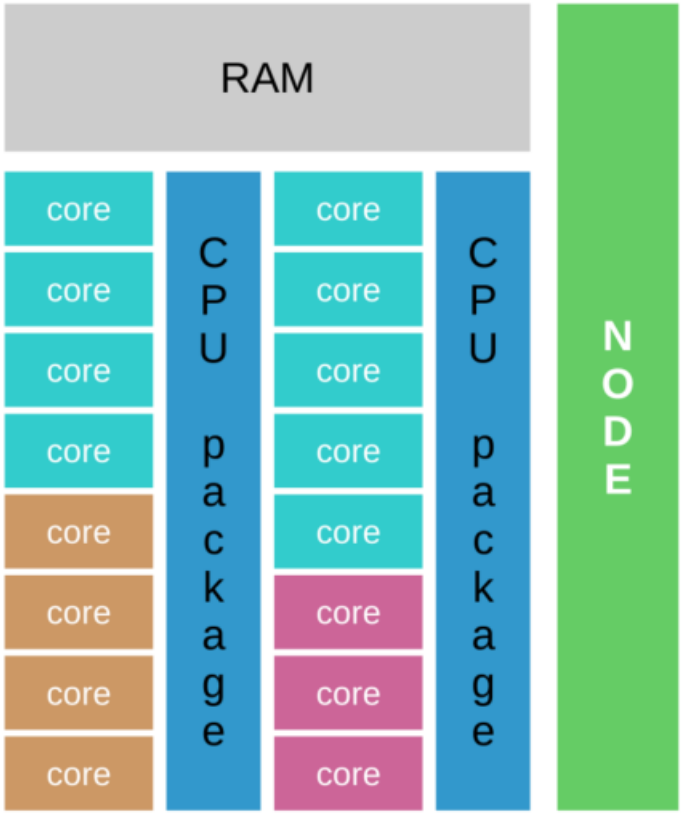

PARTS OF A COMPUTER
Learn the names and functions of common PC components!
The Power Supply Unit (PSU) delivers power and converts electrical currents from alternating (AC) to direct (DC).
650
4 TB HDD
4 TB SSD
Hard Disk Drives (HDD's) and Solid-State Drives (SSD's) store the data on a computer, such as documents, programs, and the operating system.
@mitchrandom
CPU GEN 9
The CPU cooler dissipates heat created by the CPU.
The Central Processing Unit (CPU) is responsible for most calculations done by the computer.
RAM (Random Access Memory) stores data for programs that are currently in use.
The motherboard connects all other parts of a computer and distributes power and information between them.
The GPU (Graphics Processing Unit) renders images, video, and other graphics.


SCHEME of a node
RAM
Memory
core
CPU package
NODE
Node is the actual physical self contained computer unit in a distributed computing cluster. It has its own processors, memory, I/O bus and storage.
Memory + physical cores with local access to this memory
Physical core
Processor, or set of physical cores

# But depending on your machine….



# How do I use my computer's ressources ?

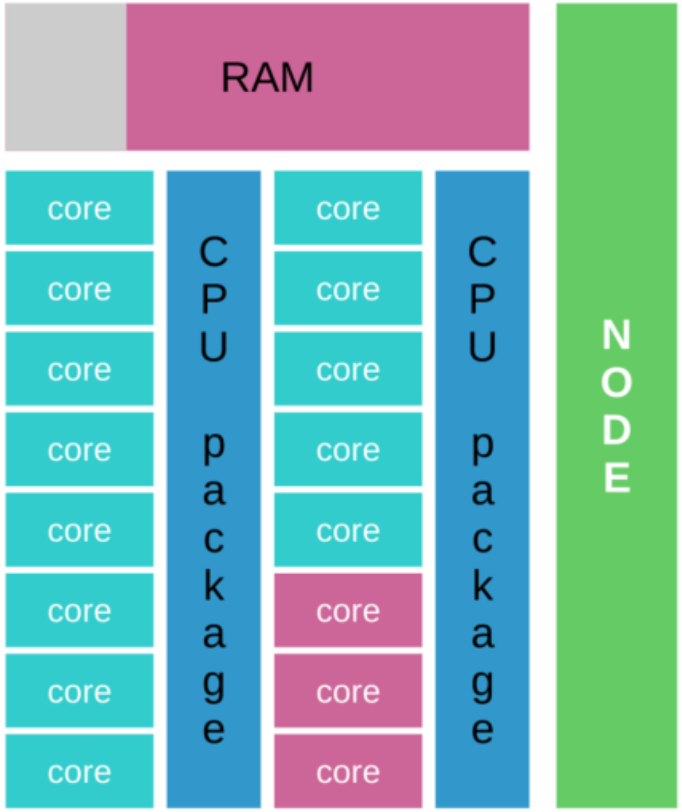



To summarize roughly, 2 cases :

1. **Certain amount of jobs to run**

→ the limitation comes from the <u>number of processing units</u> of the machine

# How do I use my computer's ressources ?




To summarize roughly, 2 cases :

1. **Certain amount of jobs to run**

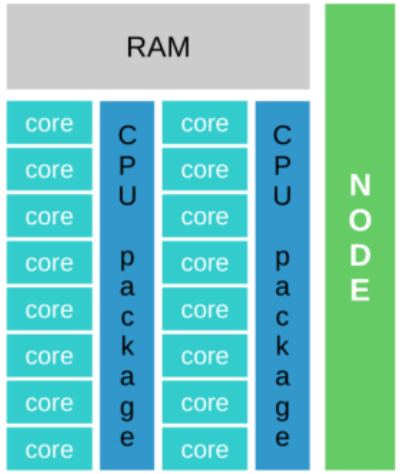
   → the limitation comes from the <u>number of processing units</u> of the machine

2. **One (or several) job *memory-hungry* to run**

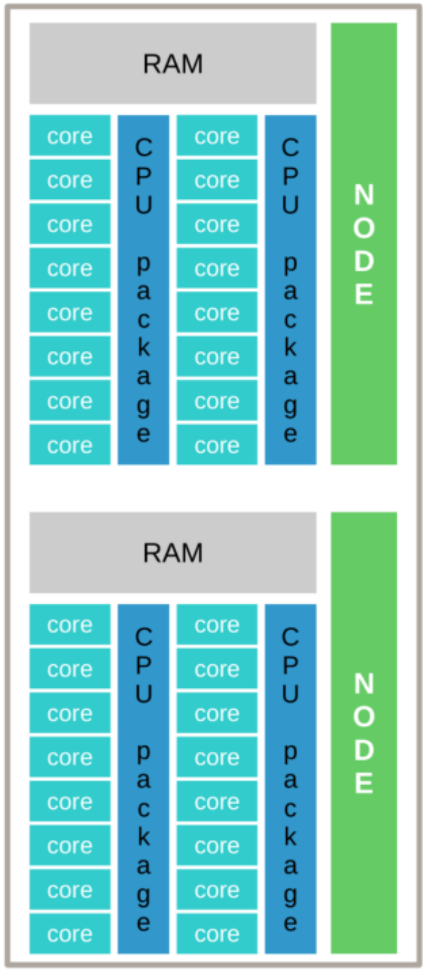
   → the limitation comes from the <u>available memory</u> of the machine

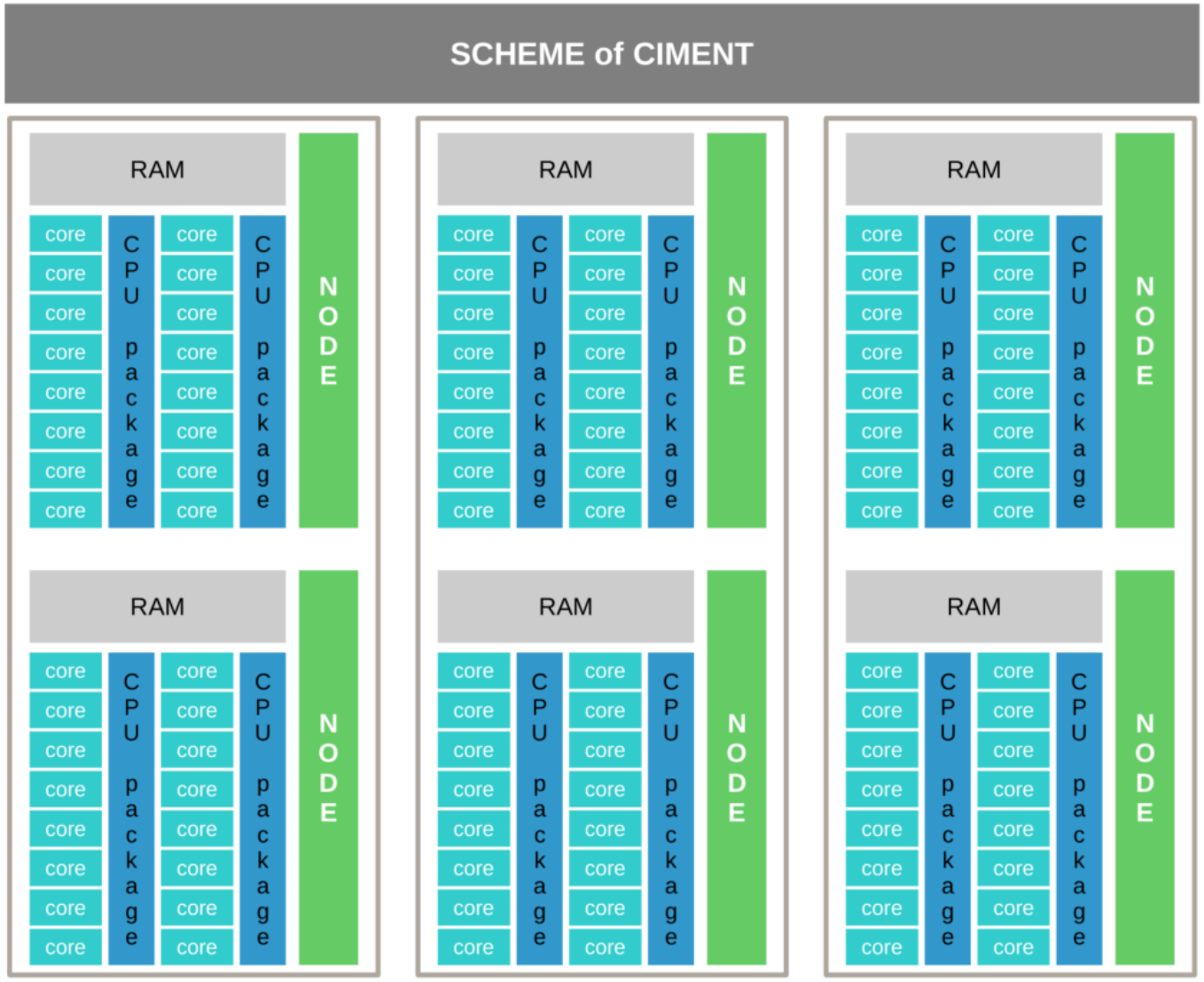
# Which kind of resources can I have access to through supercomputing servers ?

# SCHEME of a node

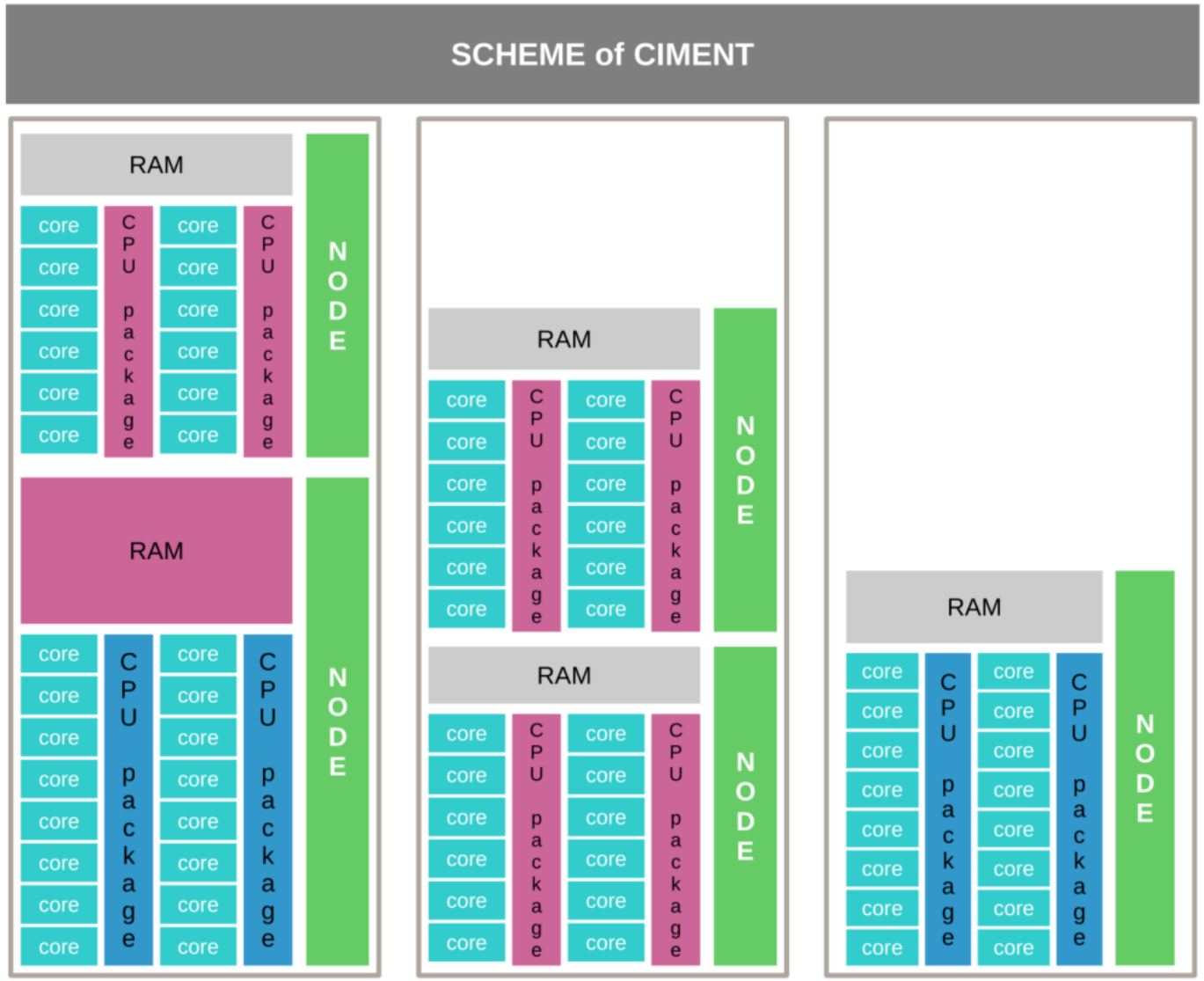


# SCHEME of a cluster






GRICAD
Grenoble Alpes Recherche
INFRASTRUCTURE DE CALCUL INTENSIF ET DE DONNÉES

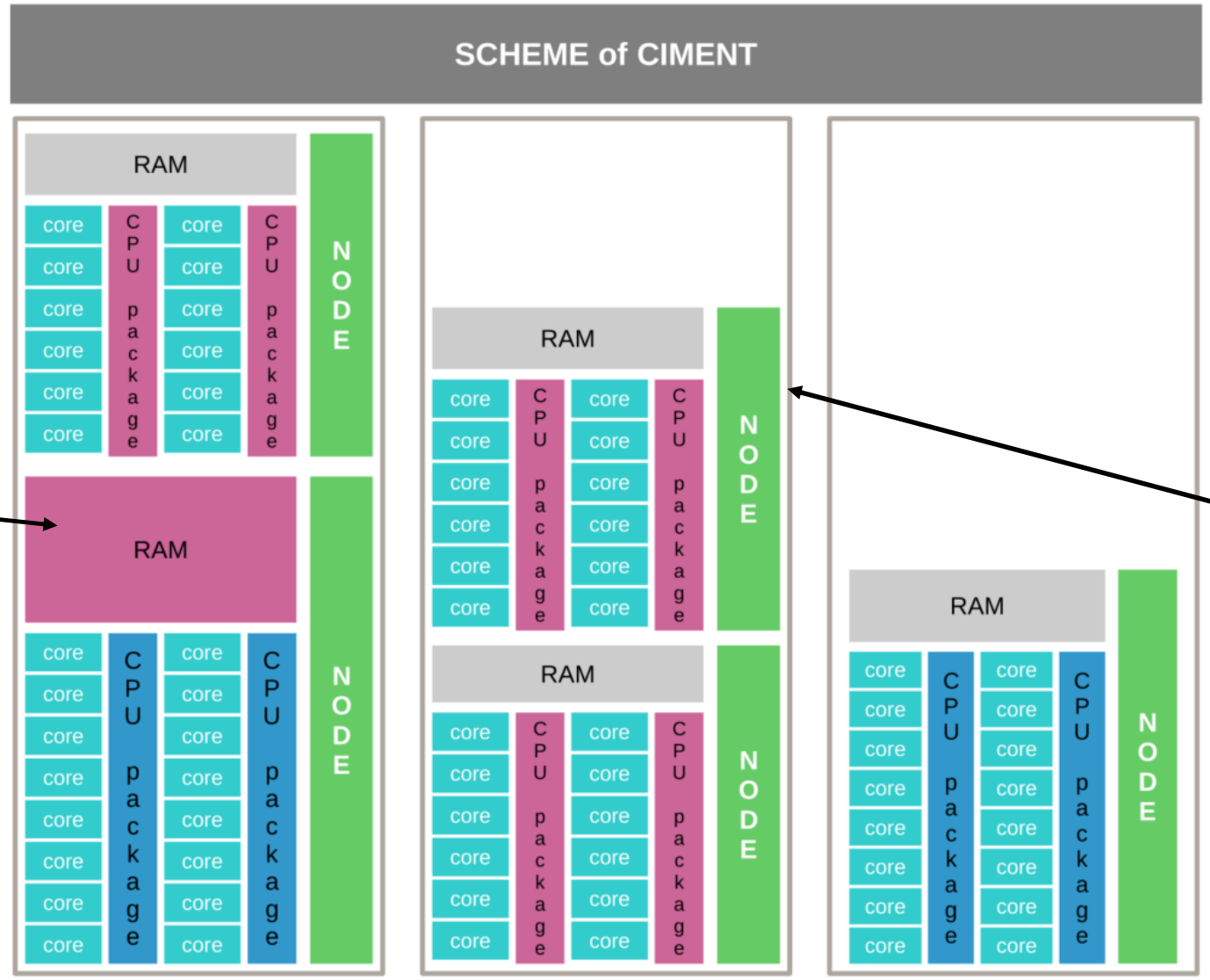



For example : LUKE

For example : DAHU

GRICAD
Grenoble Alpes Recherche
INFRASTRUCTURE DE CALCUL INTENSIF ET DE DONNÉES

For example, if I have only one job but requiring a lot of memory




Or if I have a huge number of small jobs

For example : LUKE

For example : DAHU

# So, what do I need exactly ?

- ❑ access to the supercomputing (HPC) resources
- ❑ to upload my data and my scripts onto the cluster
- ❑ to recreate my working environment
  *(here : R software, and R packages needed for my job)*
- ❑ to define the resources needed by my job
  *(memory, processing units, time)*

# In Grenoble, example of GRICAD Ciment computing servers

# So, what do I need exactly ?

- ❑ access to the supercomputing (HPC) resources
- ❑ to upload my data and my scripts onto the cluster
- ❑ to recreate my working environment
  *(here : R software, and R packages needed for my j*
- ❑ to define the resources needed by my job
  *(memory, processing units, time)*

- Create an account :
  *https://perseus.univ-grenoble-alpes.fr*
- Be part of / write a project
  explaining why you want access and use the GRICAD resources
- Connect to the cluster :
  - ✓ Windows users :
    WinSCP (*https://winscp.net/eng/download.php*) is your super friend !
  - ✓ Unix / Mac users:
    Geek way : with SSH and terminal
- Upload data / scripts :
  - ✓ Windows users :
    WinSCP again !
  - ✓ Unix / Mac users:
    Geek way : with SCP and terminal

# So, what do I need exactly ?

- ❑ access to the supercomputing (HPC) resources
- ❑ to upload my data and my scripts onto the clus
- ❑ to recreate my working environment
  *(here : R software, and R packages needed for my job)*
- ❑ to define the r
  *(memory, proc*

- <u>NIX environment !</u>

Into my `/home/username/` folder :
I can install from pre-compiled files all the softwares I need.

*You need to do this only once per cluster : then you just have to load your NIX session each time you need to use the soft you installed.*

Load my NIX session :

```
> source /applis/site/nix.sh
```

Look for a specific pattern among the pre-compiled software list :

```
> nix-env -qaP | grep boost
```

Install the desired software :

```
> nix-env -i boost
```

# So, what do I need exactly ?

- ❑ access to the supercomputing (HPC) resources
- ❑ to upload my data and my scripts onto the clus
- ❑ to recreate my working environment
  *(here : R software, and R packages needed for my job)*
- ❑ to define the resources needed by my job
  *(memory, processing units, time)*

- <u>Specific case of R (softwares) :</u>

```
> source /applis/site/nix.sh
> nix-env -i R
> nix-env -i proj
> nix-env -i gdal
> nix-env -i netcdf
> nix-env -i sqlite
> nix-env -i curl
> nix-env -i openssl
> nix-env -i boost
> nix-env -i xml2
> nix-env -i libxml2
```

- <u>Specific case of R (packages) :</u>

1. Create a file `/home/username/.config/nixpkgs/config.nix` :

```
{
   packageOverrides = super: let self = super.pkgs; in
   {
      rEnv = super.rWrapper.override {
         packages = with self.rPackages; [
                  devtools
                  ggplot2
                  reshape2
         ];
};};}
```

- ❑ to recreate my working environment
  *(here : R software, and R packages needed for my job)*

- ❑ to define the resources needed by my job
  *(memory, processing units, time)*

- Specific case of R (packages) :

1. Create a file `/home/username/.config/nixpkgs/config.nix` :

```
{
   packageOverrides = super: let self = super.pkgs; in
   {
      rEnv = super.rWrapper.override {
         packages = with self.rPackages; [
                  devtools
                  ggplot2
                  reshape2
         ];
};};}
```

2. Install all the packages at once :

```
> nix-env -f "<nixpkgs>" -iA rEnv
```

*Advantage : update at the same time the R version !*

- ❑ to recreate my working environment
  *(here : R software, and R packages needed for my job)*

- ❑ to define the resources needed by my job
  *(memory, processing units, time)*

# So, what do I need exactly ?

- ❑ access to the supercomputing (HPC) resources
- ❑ to upload my data and my scripts onto the clus
- ❑ to recreate my working environment
  *(here : R software, and R packages needed for my job)*
- ❑ to define the r
  *(memory, proc*

- <u>NIX environment !</u>

Into my `/home/username/` folder :
I can install from pre-compiled files all the softwares I need.

*You need to do this only once per cluster : then you just have to load your NIX session each time you need to use the soft you installed.*

Each time you want to work or start a job onto the cluster, load your NIX session :

```
> source /applis/site/nix.sh
```

# So, what do I need exactly ?

- ❑ access to the supercomputing (HPC) resources
- ❑ to upload my data and my scripts onto the cluster
- ❑ to recreate my working environment
  *(here : R software, and R packages needed for my j*
- ❑ to define the resources needed by my job
  *(memory, processing units, time)*

For each type of job I want to run,
I need to estimate :

- How much memory it needs
- How many processing units it uses
- How many time it takes

This is NOT an optional step !

Necessary to know
what kind and how many
resources I will have to ask for.

# So, now I can run my job(s) ?

- YES ! With the help of…

- a **Batch Scheduler** : task and resource manager for HPC

Computer application for controlling execution of non-interactive jobs through a job queue :

- adaptation to different contexts
- set default values (walltime, queue, CPUs...)
- access control (users, time slots...)

# So, now I can run my job(s) ?

- 2 types of OAR jobs :

```
> oarsub -I --project projectname -l /cpu=1,walltime=01:00:00
```

1. **Interactive job**

*This is not to be overlooked !*
It allows you to connect directly to a set of resources, through a terminal.
Very useful to :
test your script, check data upload, softwares & packages installation

2. Passive job

# So, now I can run my job(s) ?

- 2 types of OAR jobs :

1. Interactive job

2. **Passive job**

*This is to start your job / or your campaign job !*
It is based on a bash file (below `.oar` file) containing instructions to be executed in the OAR allocated resources.

```
> oarsub -S ./RUN_script.oar
```

```
> oarsub -S ./RUN_script.oar --array-param-file RUN_param.txt
```

```
#!/bin/bash

## OAR instructions ##
#OAR -n TEST_job_A
#OAR --project projectname
#OAR -l /nodes=1,walltime=40:00:00
#OAR -O log_TEST_job_A.%jobid%.stdout
#OAR -E log_TEST_job_A.%jobid%.stderr

## define some bash options
## exit script as soon as a function return error
set -e

## load ciment environment and required softwares
source /applis/site/nix.sh

## run our R script
echo `date`
R CMD BATCH /bettik/username/JOB_A/script_A.R
/dev/stdout
echo `date`

## quit the script
exit $?

## oarsub -S ./RUN_script.oar
```

## 2. Passive job

*This is to start your job / or your campaign job !*
It is based on a bash file (below `.oar` file) containing instructions to be executed in the OAR allocated resources.

```
> oarsub -S ./RUN_script.oar
```

```bash
#!/bin/bash

## OAR instructions ##
#OAR -n TEST_job_A
#OAR --project projectname
#OAR -l /nodes=1,walltime=40:00:00
#OAR -O log_TEST_job_A.%jobid%.stdout
#OAR -E log_TEST_job_A.%jobid%.stderr

## define some bash options
## exit script as soon as a function return error
set -e

## load ciment environment and required softwares
source /applis/site/nix.sh

## run our R script
echo `date`
R CMD BATCH "--args $1 $2"
/bettik/username/JOB_A/script_A.R /dev/stdout
echo `date`

## quit the script
exit $?

## oarsub -S ./RUN_script.oar --array-param-file
RUN_params_job_A.txt
```

## 2. Passive job

*This is to start your job / or your campaign job !*
It is based on a bash file (below `.oar` file) containing instructions to be executed in the OAR allocated resources.

```
> oarsub -S ./RUN_script.oar
```

```
> oarsub -S ./RUN_script.oar --array-param-file RUN_param.txt
```

```
SPECIES_1 14
SPECIES_2 3
SPECIES_3 9
SPECIES_4 11
```

# Laboratory of Alpine Ecology

***Example 1***

***(one BIG job)***

- Climate downscaling

*(refining maps of temperature and precipitation to a higher resolution taking topography into account)*

- Over the whole Alps

*(one map = 10 Go, combining several input maps)*

***Example 2***

***(a lot of small jobs)***

- Species distribution modelling

*(predicting from occurrences and corresponding environmental data a presence probability map of a species)*

- Over the whole french Alps

*(around 3,000 plant species)*

1

```
library(raster)

## Load data
ras_dem = raster('French_Alps_dem.img')
ras_temp = raster('French_Alps_temp.img')

## Compute stuff
ras_res = superFunction(ras_dem, ras_temp)

## Save results
writeRaster(ras_res, filename = 'French_Alps_dwn.img')
```

1

Install all the required packages

```
library(raster)

## Load data
ras_dem = raster('French_Alps_dem.img')
ras_temp = raster('French_Alps_temp.img')

## Compute stuff
ras_res = superFunction(ras_dem, ras_temp)

## Save results
writeRaster(ras_res, filename = 'French_Alps_dwn.img')
```

- <u>Specific case of R (packages) :</u>

1. Create a file `/home/username/.config/nixpkgs/config.nix` :

```
{
   packageOverrides = super: let self = super.pkgs; in
   {
      rEnv = super.rWrapper.override {
         packages = with self.rPackages; [
                  devtools
                  ggplot2
                  reshape2
                  raster
         ];
};};}
```

2. Install all the packages at once :

```
> nix-env -f "<nixpkgs>" -iA rEnv
```

*Advantage : update at the same time the R version !*

- ❑ to recreate my working environment
  *(here : R software, and R packages needed for my job)*

- ❑ to define the resources needed by my job
  *(memory, processing units, time)*

1

```
library(raster)

## Load data
ras_dem = raster('French_Alps_dem.img')
ras_temp = raster('French_Alps_temp.img')

## Compute stuff
ras_res = superFunction(ras_dem, ras_temp)

## Save results
writeRaster(ras_res, filename = 'French_Alps_dwn.img')
```

Be sure to upload all your data. And to put it into the right place !

1

```
library(raster)
setwd('/bettik/username/JOB_1/')

## Load data
ras_dem = raster('French_Alps_dem.img')
ras_temp = raster('French_Alps_temp.img')

## Compute stuff
ras_res = superFunction(ras_dem, ras_temp)

## Save results
writeRaster(ras_res, filename = 'French_Alps_dwn.img')
```

Be sure to upload all your data. And to put it into the right place !

1

```
library(raster)
setwd('/bettik/username/JOB_1/')

## Load data
ras_dem = raster('French_Alps_dem.img')
ras_temp = raster('French_Alps_temp.img')

## Compute stuff
ras_res = superFunction(ras_dem, ras_temp)

## Save results
writeRaster(ras_res, filename = 'French_Alps_dwn.img')
```

SAVE YOUR RESULT !

# So, now I can run my job(s) ?

- 2 types of OAR jobs :

```
> oarsub -I --project projectname -l /cpu=1,walltime=01:00:00
```

1. **Interactive job**

*This is not to be overlooked !*
It allows you to connect directly to a set of resources, through a terminal.
Very useful to :
test your script, check data upload, softwares & packages installation

2. Passive job

```
#!/bin/bash

## OAR instructions ##
#OAR -n TEST_job_1
#OAR --project projectname
#OAR -l /nodes=1,walltime=40:00:00
#OAR -O log_TEST_job_1.%jobid%.stdout
#OAR -E log_TEST_job_1.%jobid%.stderr

## define some bash options
## exit script as soon as a function return error
set -e

## load ciment environment and required softwares
source /applis/site/nix.sh

## run our R script
echo `date`
R CMD BATCH /bettik/username/JOB_1/script_1.R
/dev/stdout
echo `date`

## quit the script
exit $?

## oarsub -S ./RUN_script_1.oar
```

## 2. Passive job

*This is to start your job / or your campaign job !*
It is based on a bash file (below `.oar` file) containing instructions to be executed in the OAR allocated resources.

```
> oarsub -S ./RUN_script_1.oar
```

2

```r
library(foreach)
library(data.table)

## Load species list
list.species = paste0('species_', 1:48)

## Do the job for each species
RES.sp = foreach (sp = list.species) %do%
{
   occ = read.txt(paste0(sp, '_occ.txt'))
   res = superFunction(sp, occ)
   return(res)
}

## Save results
save(RES.sp, file = 'RES.sp.Rdata')
```

2

Install all the required packages

```
library(foreach)
library(data.table)

## Load species list
list.species = paste0('species_', 1:48)

## Do the job for each species
RES.sp = foreach (sp = list.species) %do%
{
   occ = read.txt(paste0(sp, '_occ.txt'))
   res = superFunction(sp, occ)
   return(res)
}

## Save results
save(RES.sp, file = 'RES.sp.Rdata')
```

- <u>Specific case of R (packages) :</u>

1. Create a file `/home/username/.config/nixpkgs/config.nix` :

```
{
   packageOverrides = super: let self = super.pkgs; in
   {
      rEnv = super.rWrapper.override {
         packages = with self.rPackages; [
                  devtools
                  ggplot2
                  reshape2
                  raster
                  foreach
                  data.table
         ];
};};}
```

2. Install all the packages at once :

```
> nix-env -f "<nixpkgs>" -iA rEnv
```

*Advantage : update at the same time the R version !*

- ❑ to recreate my working environment
  *(here : R software, and R packages needed for my job)*

- ❑ to define the resources needed by my job
  *(memory, processing units, time)*

2

```r
library(foreach)
library(data.table)

## Load species list
list.species = paste0('species_', 1:48)

## Do the job for each species
RES.sp = foreach (sp = list.species) %do%
{
   occ = read.txt(paste0(sp, '_occ.txt'))
   res = superFunction(sp, occ)
   return(res)
}

## Save results
save(RES.sp, file = 'RES.sp.Rdata')
```

OPTION 1 :

Ask for one node with lot of cores and do some intern parallelisation

2

```r
library(foreach)
library(data.table)
library(doParallel)
registerDoParallel(cores = 10)

## Load species list
list.species = paste0('species_', 1:48)

## Do the job for each species
RES.sp = foreach (sp = list.species) %dopar%
{
   occ = read.txt(paste0(sp, '_occ.txt'))
   res = superFunction(sp, occ)
   return(res)
}

## Save results
save(RES.sp, file = 'RES.sp.Rdata')
```

OPTION 1 :

Ask for one node with lot of cores and do some intern parallelisation

```
#!/bin/bash

## OAR instructions ##
#OAR -n TEST_job_2
#OAR --project projectname
#OAR -l /nodes=1,walltime=40:00:00
#OAR -O log_TEST_job_2.%jobid%.stdout
#OAR -E log_TEST_job_2.%jobid%.stderr

## define some bash options
## exit script as soon as a function return error
set -e

## load ciment environment and required softwares
source /applis/site/nix.sh

## run our R script
echo `date`
R CMD BATCH /bettik/username/JOB_2/script_job_2.R
/dev/stdout
echo `date`

## quit the script
exit $?

## oarsub -S ./RUN_script_2.oar
```

## 2. Passive job

*This is to start your job / or your campaign job !*
It is based on a bash file (below `.oar` file) containing instructions to be executed in the OAR allocated resources.

```
> oarsub -S ./RUN_script_2.oar
```

2

```r
library(foreach)
library(data.table)

## Load species list
list.species = paste0('species_', 1:48)

## Do the job for each species
RES.sp = foreach (sp = list.species) %do%
{
   occ = read.txt(paste0(sp, '_occ.txt'))
   res = superFunction(sp, occ)
   return(res)
}

## Save results
save(RES.sp, file = 'RES.sp.Rdata')
```

OPTION 2 :

Run each species separately.
Meaning, have as many jobs as species

2

```r
library(foreach)
library(data.table)

## Load species list
list.species = paste0('species_', 1:48)

## Do the job for each species
## RES.sp = foreach (sp = list.species) %do%
{
   occ = read.txt(paste0(sp, '_occ.txt'))
   res = superFunction(sp, occ)
   ## return(res)
}

## Save results
## save(RES.sp, file = 'RES.sp.Rdata')
```

Keep only the executive commands for one iteration

OPTION 2 :

Run each species separately.
Meaning, have as many jobs as species

Replace the species list with function to get only one value from external parameter(s).

2

```
library(foreach)
library(data.table)

## Load species list
## list.species = paste0('species_', 1:48)
args = commandArgs(trailingOnly = TRUE)
sp = as.character(args[1])

## Do the job for each species
## RES.sp = foreach (sp = list.species) %do%
{
   occ = read.txt(paste0(sp, '_occ.txt'))
   res = superFunction(sp, occ)
   ## return(res)
}

## Save results
## save(RES.sp, file = 'RES.sp.Rdata')
```

OPTION 2 :

Run each species separately.
Meaning, have as many jobs as species

2

```r
library(foreach)
library(data.table)

## Load species list
## list.species = paste0('species_', 1:48)
args = commandArgs(trailingOnly = TRUE)
sp = as.character(args[1])

## Do the job for each species
## RES.sp = foreach (sp = list.species) %do%
{
   occ = read.txt(paste0(sp, '_occ.txt'))
   res = superFunction(sp, occ)
   save(res, file = paste0('RES.', sp, '.Rdata'))
   ## return(res)
}

## Save results
## save(RES.sp, file = 'RES.sp.Rdata')
```

SAVE YOUR RESULT !

OPTION 2 :

Run each species separately.
Meaning, have as many jobs as species

```bash
#!/bin/bash

## OAR instructions ##
#OAR -n TEST_job_2
#OAR --project projectname
#OAR -l /nodes=1,walltime=40:00:00
#OAR -O log_TEST_job_2.%jobid%.stdout
#OAR -E log_TEST_job_2.%jobid%.stderr

## define some bash options
## exit script as soon as a function return error
set -e

## load ciment environment and required softwares
source /applis/site/nix.sh

## run our R script
echo `date`
R CMD BATCH "--args $1"
/bettik/username/JOB_2/script_2.R /dev/stdout
echo `date`

## quit the script
exit $?

## oarsub -S ./RUN_script_2.oar --array-param-file
RUN_params_2.txt
```

## 2. Passive job

*This is to start your job / or your campaign job !*
It is based on a bash file (below `.oar` file) containing instructions to be executed in the OAR allocated resources.

```
> oarsub -S ./RUN_script_2.oar --array-param-file RUN_params_2.txt
```

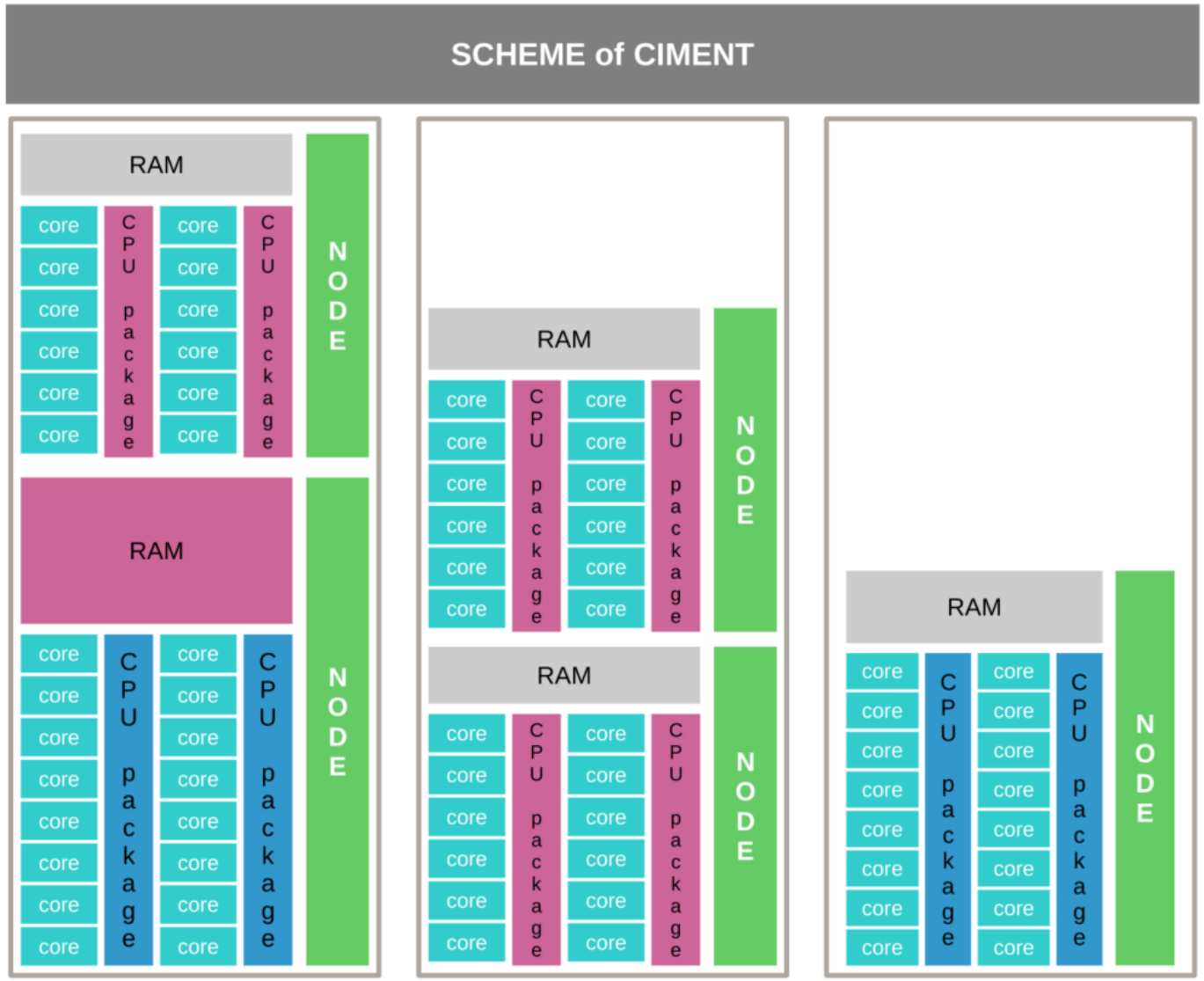
```
species_1
species_2
species_3
species_4
species_5
species_6
```

2

```r
library(foreach)
library(data.table)

## Load species list
## list.species = paste0('species_', 1:48)
args = commandArgs(trailingOnly = TRUE)
sp = as.character(args[1])

## Do the job for each species
## RES.sp = foreach (sp = list.species) %do%
{
   occ = read.txt(paste0(sp, '_occ.txt'))
   res = superFunction(sp, occ)
   save(res, file = paste0('RES.', sp, '.Rdata'))
   ## return(res)
}

## Save results
## save(RES.sp, file = 'RES.sp.Rdata')
```

OPTION 3 :

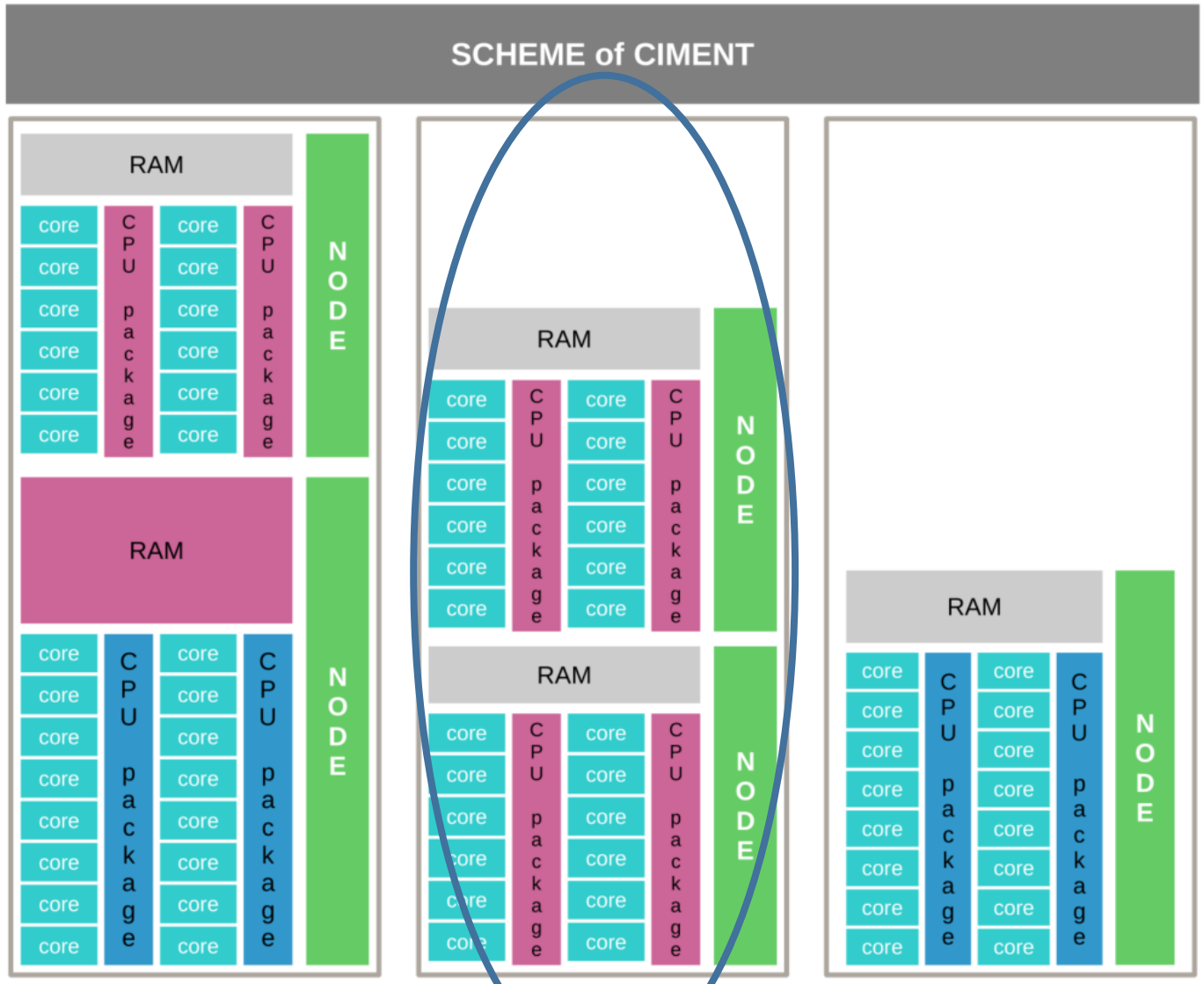
NOT SEEN TODAY :
Run each species separately.
Meaning, have as many jobs as species BUT ON DIFFERENT CLUSTERS.




For example : LUKE

For example : DAHU

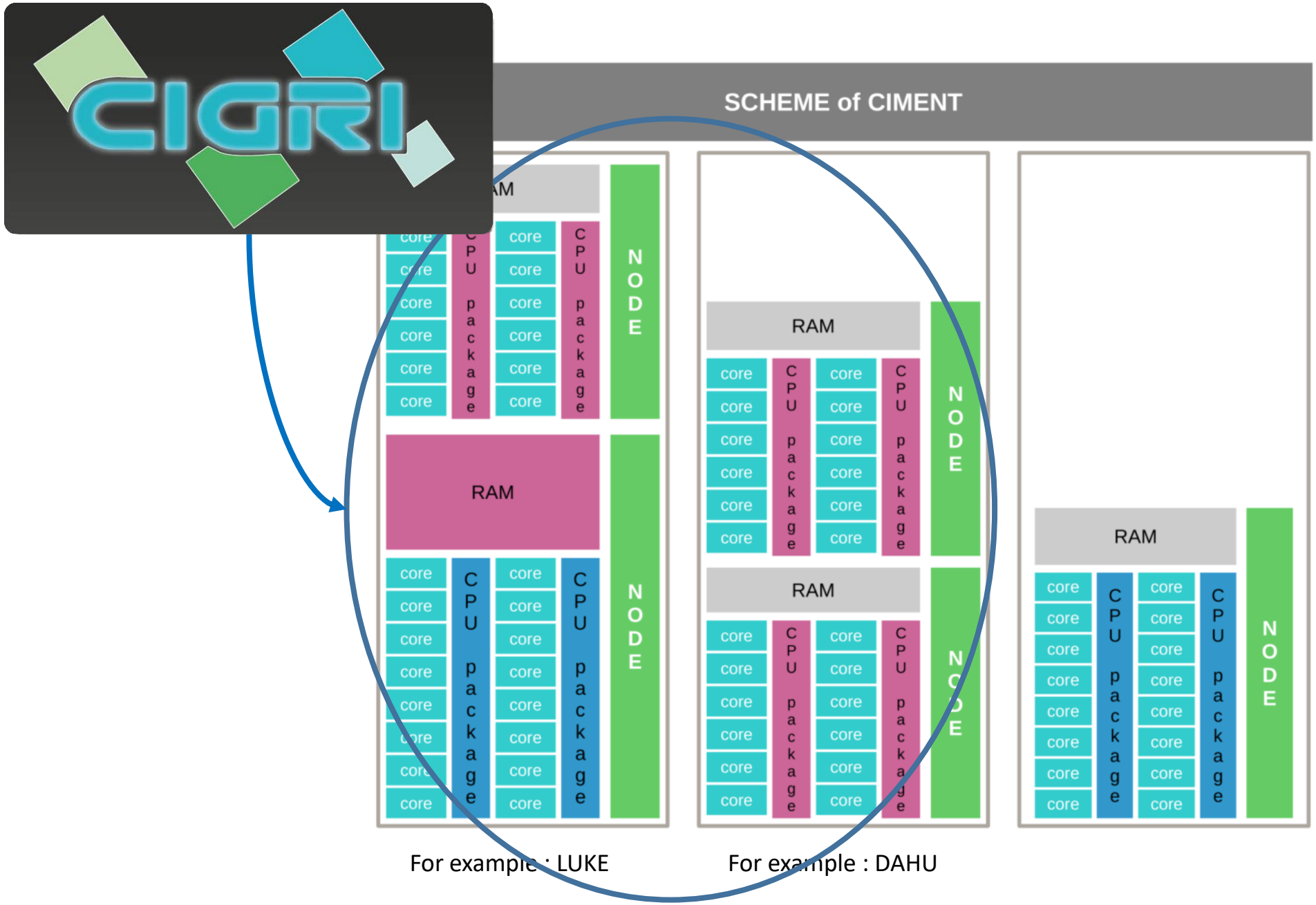
GRICAD
Grenoble Alpes Recherche
INFRASTRUCTURE DE CALCUL INTENSIF ET DE DONNÉES

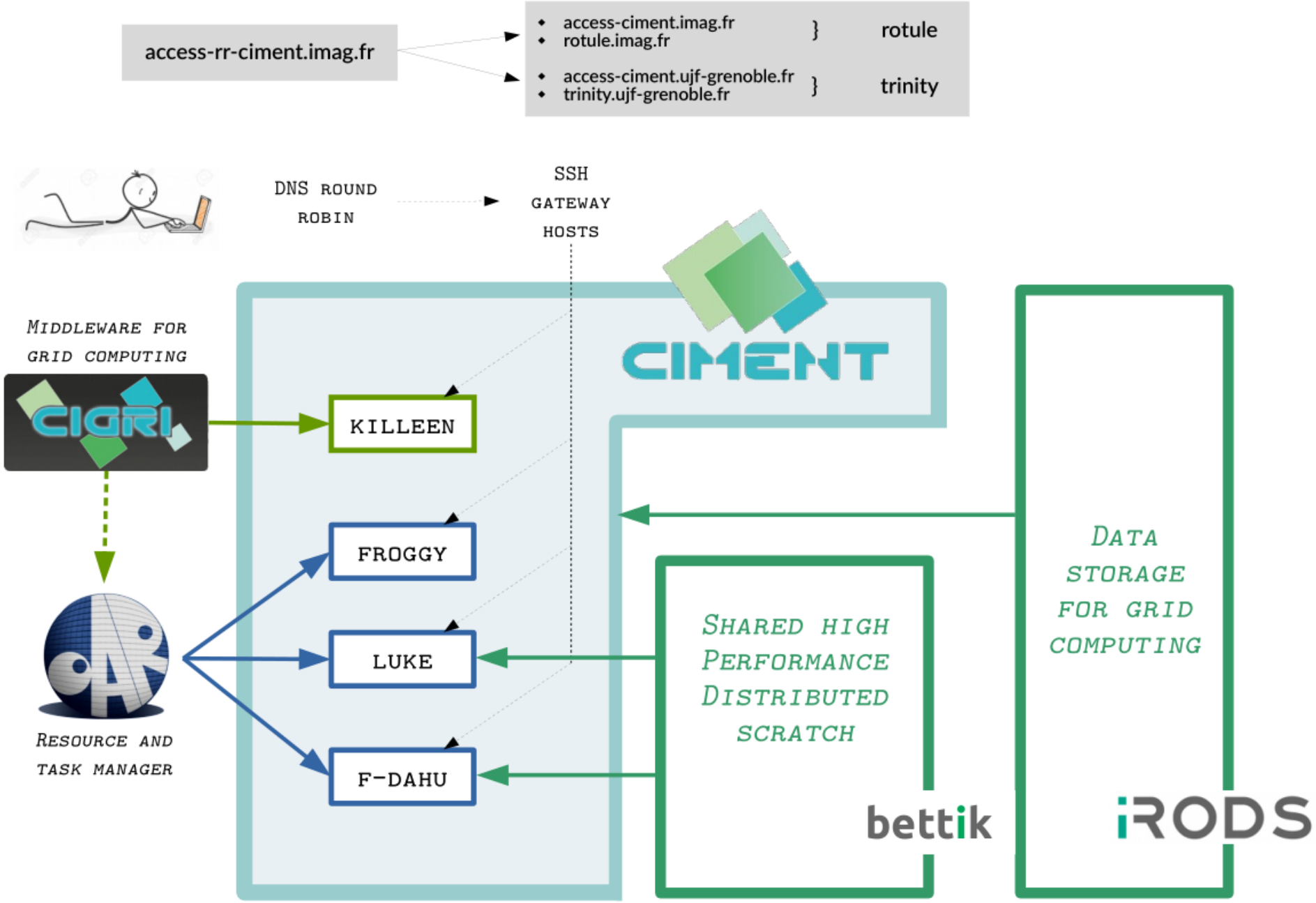



For example : LUKE

For example : DAHU

GRICAD
Grenoble Alpes Recherche
INFRASTRUCTURE DE CALCUL INTENSIF ET DE DONNÉES


SCHEME of CIMENT
RAM
core
CPU package
NODE
For example : LUKE
For example : DAHU


access-rr-ciment.imag.fr
access-ciment.imag.fr
rotule.imag.fr
rotule
access-ciment.ujf-grenoble.fr
trinity.ujf-grenoble.fr
trinity
DNS ROUND ROBIN
SSH GATEWAY HOSTS
CIMENT
MIDDLEWARE FOR GRID COMPUTING
CIGRI
KILLEEN
FROGGY
LUKE
F-DAHU
RESOURCE AND TASK MANAGER
SHARED HIGH PERFORMANCE DISTRIBUTED SCRATCH
bettik
DATA STORAGE FOR GRID COMPUTING
iRODS

I thought this was a ‘*R in Grenoble*’ speech !
Why is there not a lof of **R** in this presentation ?!

# Because this is not really something only R-dependent…

- GRICAD clusters are not used only by R users…

  But a lot of scientific communities, using different softwares, different languages (python, C, matlab…).
  Plus, you need to learn how it works and how to use (nobody is going to take your script and your data and do the job for you !).

- This is NOT really a question of R script…

  No need for super efficient and optimized scripts… as long as it runs ! The servers are here to provide resources that you don’t have. Run 100,000 simulations instead of 200 onto your laptop ; run an analysis over the whole country instead of only your region, etc.

# Advantages & drawbacks

- ✓ Access to bigger resources
  (no need to buy a supermachine, or to take care of it)
- ✓ Free of charge for all UGA researchers
  (but it is strongly advised, in a long term view, by allocating a share of the budget to GRICAD machines when looking for funding)
- ✓ GRICAD members are here to help you !

- ○ No graphical interface
  (out of the connection through WinSCP, everything goes through terminal and bash commands)
- ○ Some softwares / packages versions are not available, or hard to install onto the cluster by yourself
  (but once again, you can ask for help to GRICAD)
- ○ This is NOT straightforward. You need to put your head into this.
  To clean your scripts, to estimate the resources you need, to build the OAR scripts. This is not like just installing a new R package.