(Apache) Spark for physicists

C. Arnault, G. Barrand, J.E. Campagne, J. Peloton and S. Plaszczynski

LAL, Univ. Paris-Sud, CNRS/IN2P3, Université Paris-Saclay, Orsay, France

May 13, 2019

High Performance Computing (HPC)

- since ~ 2 decades CPU freq was frozen (power consumption)
- but more and more data...
- →more and more complex computer *architectures*
- multi core+machine parallelization often on *super-computers*
- `OpenMP, MPI, C++11/14/17, SIMD, GPU,FPGA...:`

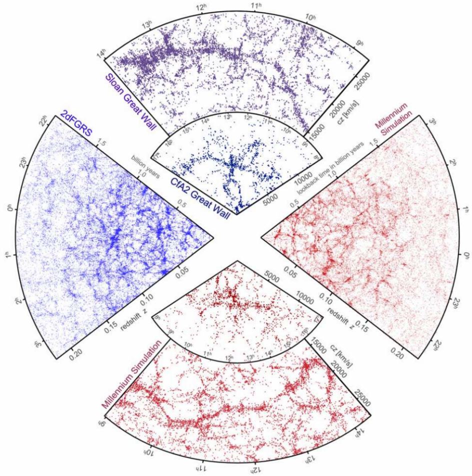
complicated...

High Throughput Computing (HTC) aka ”Big Data”

- 2004 `Google`: `mapReduce` programming model foundation of *distributed computing*
- 2006 `Hadoop` open-source framework (ecosystem) `HDFS,Hive, YARN...`
- 2004 `scala` (`java` ecosystem)
- 2009 `Spark`: research project at UC. Berkeley
- 2015 `Spark` SQL (dataframes)
- today: (`Apache`) `Spark` used by $\gtrsim$ 1000 companies

Meanwhile in cosmology...

Springel et al. (2006)



SDSS

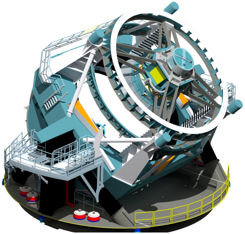
`BOSS` $\mathcal{O}(10^6)$ galaxies, `DESI` $\mathcal{O}(10^7)$

DES

$\mathcal{O}(10^7)$

LSST

- start 2022 for 10 years
- 8.4m primary mirror
- 3.2 Gpixels camera
- 18000 deg^2
- 15 TB raw data/night
- +mocks...→big data



(may 2018, Cerro Pachón)

So what is Spark about?

A framework to work as *close* as possible to the data

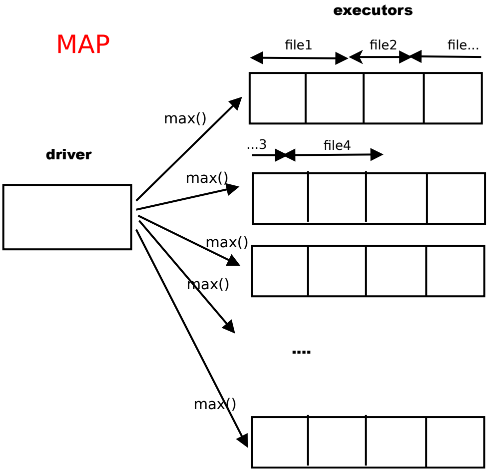
in practice: a set of functions: `scala`, (`java`), `python`, `R`

```
data.transform1().transform2()....action()
```

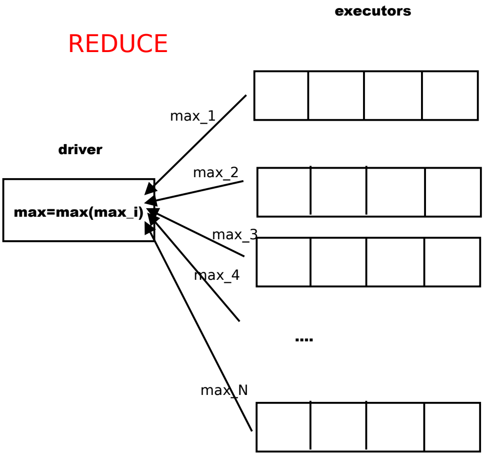
This is Functional Programming (but you don't need to know it!)

Distributed computing

MAP



Advantage 1= simple parallelization




```
dataframe.select(max("variable"))
```

Advantage 2 ?

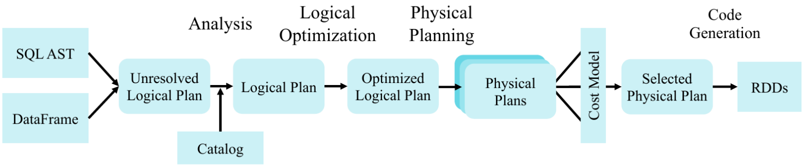
```
> data=spark.read.format("fits")\
              .load("path/to/110GB/of/fits/files")
> data.show(5)
+---------+---------+---------+---------+
|       RA|      Dec|        z|     zrec|
+---------+---------+---------+---------+
|225.80168|18.519966|2.4199903| 2.414322|
|225.73839|18.588171|2.4056022|2.2913096|
|225.79999|18.635067| 2.396816|2.3597262|
|225.49783|18.570776|2.4139786|2.3434482|
|225.57983|18.638515|2.3995044|2.3826954|
+---------+---------+---------+---------+
```

5s !/?

Lazy evaluation →optimization

- you are used to *imperative* languages (C/C++/FORTRAN...)
- here lazy evaluation: code is an ‘expression-language’ that allows to build a Direct Acyclic Graph (DAG)
- transformations (`load`,`map`,`filter..`) →update DAG
- actions (`count`,`collect`,`show..`) →optimize DAG (`Catalyst`) and run

Advantage 2= Automatic pipeline optimization



the Machine does it better than you! →`Spark` reason of success

Advantage 3= in memory work (cache)

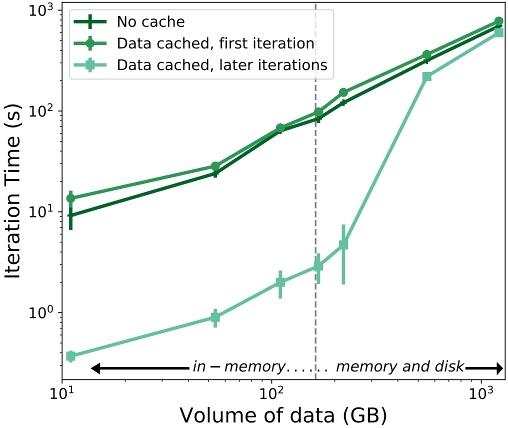
Put the data in cache as if you had a **huge** RAM

- ex: 110 GB on a small cluster (8 workers)
- 1TB at NERSC

```
dataframe.cache()
```

Then you can work interactively

Advantage 4= scaling



A use-case in cosmology

- generate LSST 10Y of galaxies with fast sim `https://github.com/damonge/CoLoRe.git`
- →110GB of FITS files. 6. 10^9 galaxies
- goal is to have a quick *interactive* look at what was generated (`python`)
- this is different from *developing* software (`scala`)

The U-PSUD cluster

- 9 machines: 18 cores+ 32 GB RAM each
- cache $= 0.6 \times mem = 144GB$ $\rightarrow$enough to hold our dataset
- HDFS

Data sources

- `Spark` was rather developed to ~~spy~~ study your habits: poorly structured data (text, although `avro, parquet`)
- need to develop support for more complex structures
- popular formats in astronomy: FITS, HDF5
- but no good native FITS/HDF5 `Spark` reader exists...

`spark-fits` high performance connector (+lib) Peloton et al. (2018)

Reading a FITS file

Nothing to do on the user-side: just copy your standard FITS file to your cluster and then

```
> df=spark.read.format("fits").option("hdu",1)\
      .load("hdfs:path/to/fits/dir/")

> df.printSchema()

root
 |-- TYPE: integer (nullable = true)
 |-- RA: float (nullable = true)
 |-- DEC: float (nullable = true)
 |-- Z_COSMO: float (nullable = true)
 |-- DZ_RSD: float (nullable = true)
```

Dataframe similar to R/pandas
('n-tuple' in HEP since the 70's, 'binTable' in FITS since the 80's...)

1. Selecting columns

```
> gal=df.select("RA","Dec", \
      (col("Z_COSMO")+col("DZ_RSD")).alias("z"))

> gal.show(5)

+----------+-----------+-----------+
|        RA|        Dec|          z|
+----------+-----------+-----------+
|  265.1168|  -79.96222|  0.5590986|
|  258.0575|  -79.84589| 0.55854577|
| 261.24503| -80.293274| 0.56063706|
| 279.49026|  -80.23766| 0.56124765|
|  285.2853|  -79.96391| 0.56244487|
+----------+-----------+-----------+
```

2. Put them in cache

```
> gal.cache().count()

5926764680
```

$\simeq$100s

3. A first quick look

```
> gal.describe('z').show()

+-------+--------------------+
|summary|                   z|
+-------+--------------------+
|  count|            11713638|
|   mean|0.277558705877229775|
| stddev|  0.163914089685891 1|
|    min|        -0.001773748 5|
|    max|           0.5673544|
+-------+--------------------+
```

4s

4.Histograms (the distributed way)

starting from z add a column of bin numbers

```
> zbin=gal.select("z",
        ((gal['z']-zmin-dz/2)/dz).astype('int')\
          .alias('bin'))

+----------+----+
|         z|bin|
+----------+----+
| 0.5590986| 98|
|0.55854577| 97|
|0.56063706| 98|
|0.56124765| 98|
|0.56244487| 98|
|0.55902207| 98|
...
```

Histograms (the distributed way) 2

GroupBy this column

```
> zbin=zbin.groupBy("bin")...
+-----------+-----------+--------------------+-----+
|          z|      group                     | bin|
+-----------+-----------+--------------------+-----+
| {0.5590986,0.56063706,0.56124765,...}     | 98  |
| {0.55854577. ...                       }  | 97  |
...
```

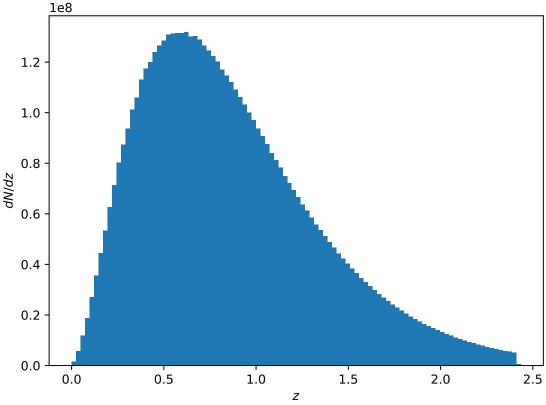
and count by group

```
> zbin=zbin.groupBy("bin").count()
+---+-------+
|bin| count|
+---+-------+
| 98|116607|
| 97|117410|
```

Histograms (the distributed way) 3

- sort in ”bin” ascending order
- add locations (bin centers)

```
> h=zbin.sort("bin",ascending=True)
> histo=h.select((zmin+dz/2+h['bin']*dz)\
                .alias('zbin')\
              ,"count")
+--------------------+-------+
|                zbin|  count|
+--------------------+-------+
|0.001071892101317644|237445|
|0.006763173397630453|178469|
| 0.01245445469394326|132612|
| 0.01814573599025607|102854|
| 0.02383701728656888| 96153|
...
```



$\boxed{\simeq 11\text{s}}$

on 6.10^9 data! imperative way (sequential): 45 mins

5. User-Defined Functions (UDF)

```
binNum=udf(lambda z: int((z-zmin-dz/2)/dz))
zbin=gal.select(gal.z,\
     binNum(gal.z).alias('bin'))
```

115s !

```
@pandas_udf("float", PandasUDFType.SCALAR)
def binNumber(z):
    return pandas.Series((z-zmin)/dz)

zbin=gal.select(gal.z,\
     binNumber("z").astype('int').alias('bin'))
```

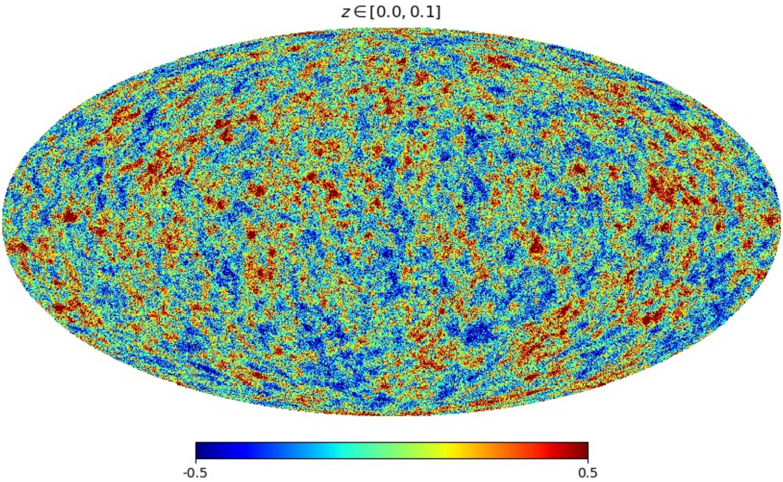
40s

6. Tomography

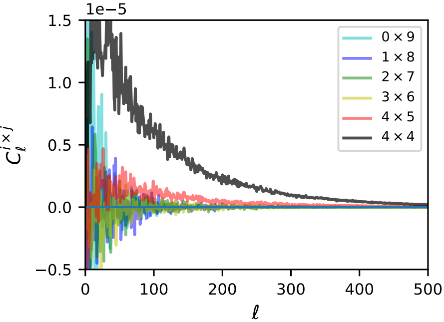
- compute over-densities in redshift regions (shells)
- project onto a map (`HEALPix`)
- compute cross/auto power-spectra
- $P(k, z)$+shear= powerful probe for cosmology (DES 1Y Troxel et al. (2018))

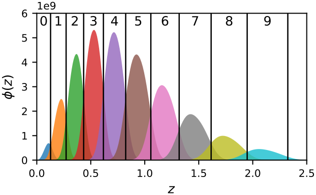
Implementation

```
@pandas_udf('int', PandasUDFType.SCALAR)
def Ang2Pix(ra,dec):
    theta=np.radians(90-dec)
    phi=np.radians(ra)
    return pandas.Series(\
           healpy.ang2pix(nside,theta,phi)
           )\

shell=gal.filter(gal['z'].between(z1,z2))

map=shell.select(Ang2Pix("RA","Dec")\
    .alias("ipix"))\
    .groupBy("ipix").count()
```


$z \in [0.0, 0.1]$
-0.5
0.5

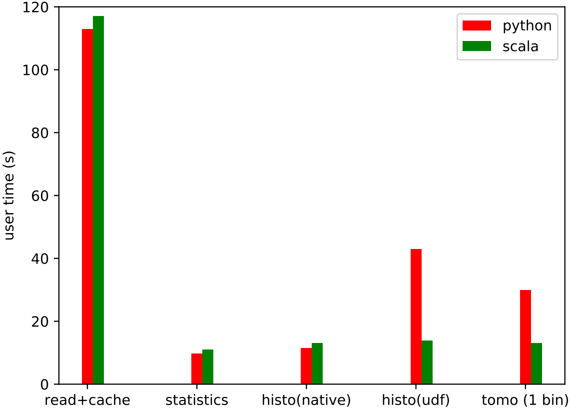
Power spectra



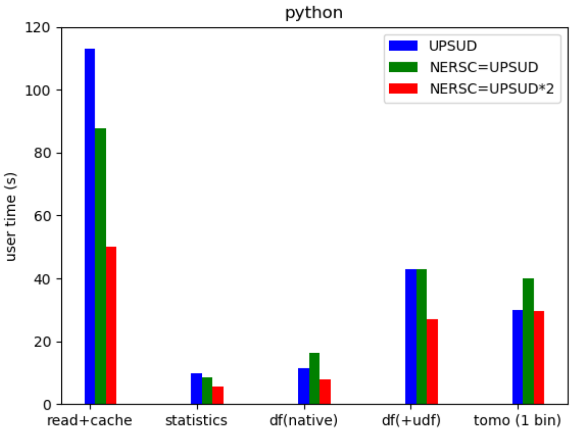


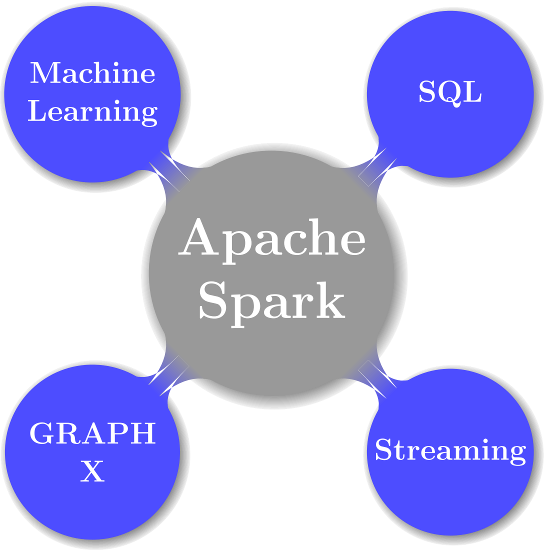
$\simeq$30 s/shell

python or scala?



Do you need a supecomputer?




Machine Learning
SQL
Apache Spark
GRAPH X
Streaming

AstroLab software

https://astrolabsoftware.github.io

spark-fits
FITS data source for Spark SQL and DataFrames
Scala ★ 10 4

spark3D
Spark extension for processing large-scale 3D data sets: Astrophysics, High Energy Physics, Meteorology, ...
Scala ★ 9 3

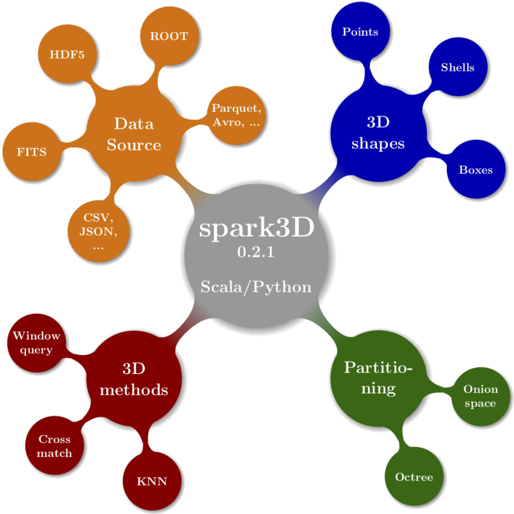
fink-broker
Astronomy Broker based on Apache Spark
Python ★ 2 3

spark-tutorials
PySpark notebooks to learn Apache Spark (WIP)
Jupyter Notebook ★ 5

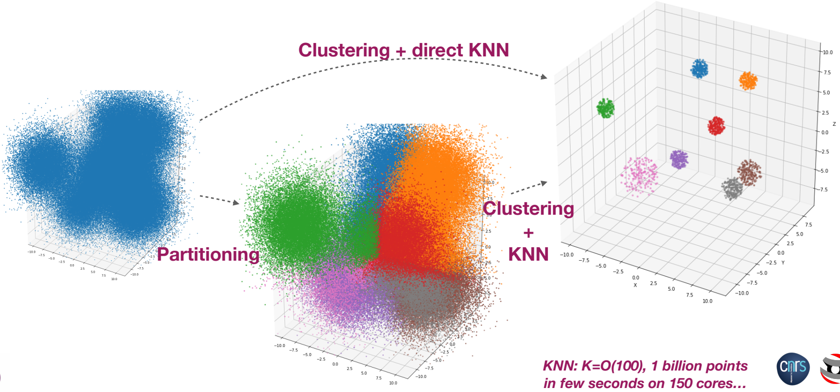
scala-tutorials
Tutorials on functional programming & Scala
Dockerfile ★ 2

Interfaces
How to interface different languages implied in the process of scientific programming especially in the context of the AstroLab Sofware organization, or developments using it.
Scala ★ 4

Spark-3D


spark3D
0.2.1
Scala/Python
Data Source
HDF5
ROOT
Parquet, Avro, ...
FITS
CSV, JSON, ...
3D shapes
Points
Shells
Boxes
3D methods
Window query
Cross match
KNN
Partitio-ning
Onion space
Octree

spark3D: K Nearest Neighbours



clustering algorithm =`k-Means` →`DBSCAN`

Where do I start?

- download and play with Spark on your laptop
- read Plaszczynski et al. (2018) + notebook
  https://github.com/astrolabsoftware/1807.03078

clusters:

- NERSC?
- not at CCIN2P3 →ask!
- U-PSud
- CERN/openlab?

openlab Big Data Analytics

in collaboration with Intel

hadoop, Apache Spark, SERVICE IT-DB: Provides infrastructure, knowledge, consultancy and integration with the rest of the IT services

Ensures that industry, CERN IT and the Experiments are effectively connected

CERN openlab

Provides resources and consultancy on big data technologies and optimization

intel

Provides the relevant use cases and physics analysis code

CMS

https://cernbox.cern.ch/index.php/s/6B89Z3wQgfZci7h

CMS Data Reduction Facility - Motivation

Why Data Analytics & Reduction with Spark?

- Investigate new ways to analyse physics data
- Improve resource utilization and time-to-physics
- Adopt new technologies widely used in the industry
  - Open the HEP field to a larger community
  - Improve chance of researchers on the job market outside academia

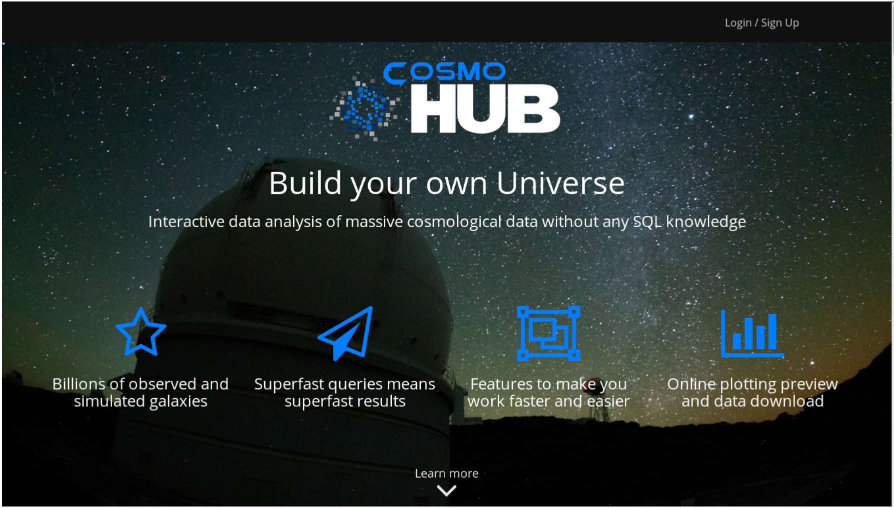
Full refs

Peloton, J., Arnault, C., & Plaszczynski, S. 2018, ArXiv e-prints, arXiv:1804.07501

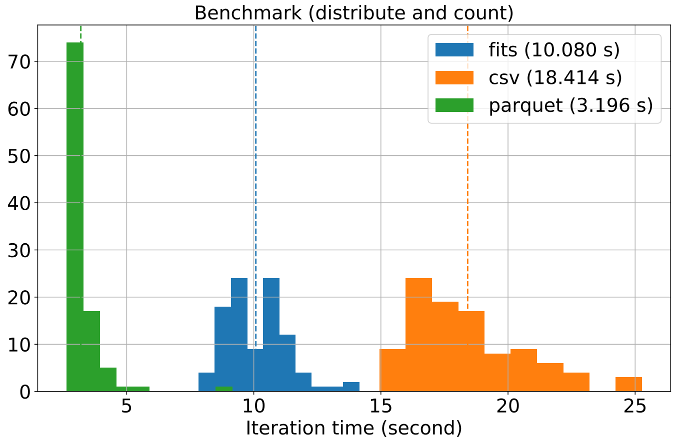
Plaszczynski, S., Peloton, J., Arnault, C., & Campagne, J. E. 2018, ArXiv e-prints, arXiv:1807.03078

Springel, V., Frenk, C. S., & White, S. D. M. 2006, Nature, 440, 1137, arXiv:astro-ph/0604561

Troxel, M. A., MacCrann, N., Zuntz, J., et al. 2018, Phys. Rev. D, 98, 043528, arXiv:1708.01538


Login / Sign Up
COSMO HUB
Build your own Universe
Interactive data analysis of massive cosmological data without any SQL knowledge
Billions of observed and simulated galaxies
Superfast queries means superfast results
Features to make you work faster and easier
Online plotting preview and data download
Learn more

Performances



FP

- concepts from math logic theory ($\lambda-$calculus): Curry-Howard correspondence
- *imperative* languages rather developed (Turing machines)
- Lisp, Haskell..
- `scala` used by some scientific communities (genomics)

main ideas:

- functions are fundamental basic types
- Referential transparency $\implies f(x) + f(x) = 2f(x)$
- no idea of 'state' (but monades introduced in `scala`)
- immutability (no side-effect), recursivity...

quite clear /concise/robust codes but not very used until `scala/Spark`