Welcome!
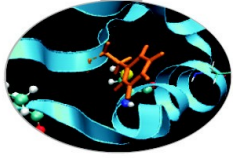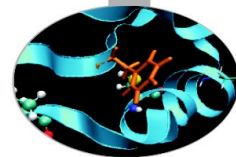
Emerging tools and techniques for massive data analysis

SuperComputing Applications and Innovation Department
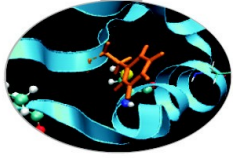
Goals

**During this three-day workshop you will learn:**

- the trends and challenges surrounding the BigData definition
- how the most relevant technologies and methods in this area work
    - Apache *Hadoop*
    - *Map-Reduce*
    - *Spark, SparkSQL*
    - *MLLIB*
- how to structure and program your code using Python

Quick poll

- How many of you already know part of the mentioned Big Data technologies?
- How many of you know any HPC methods (MPI, OpenMP, etc.)?
- How many of you know Python?
- Scala?
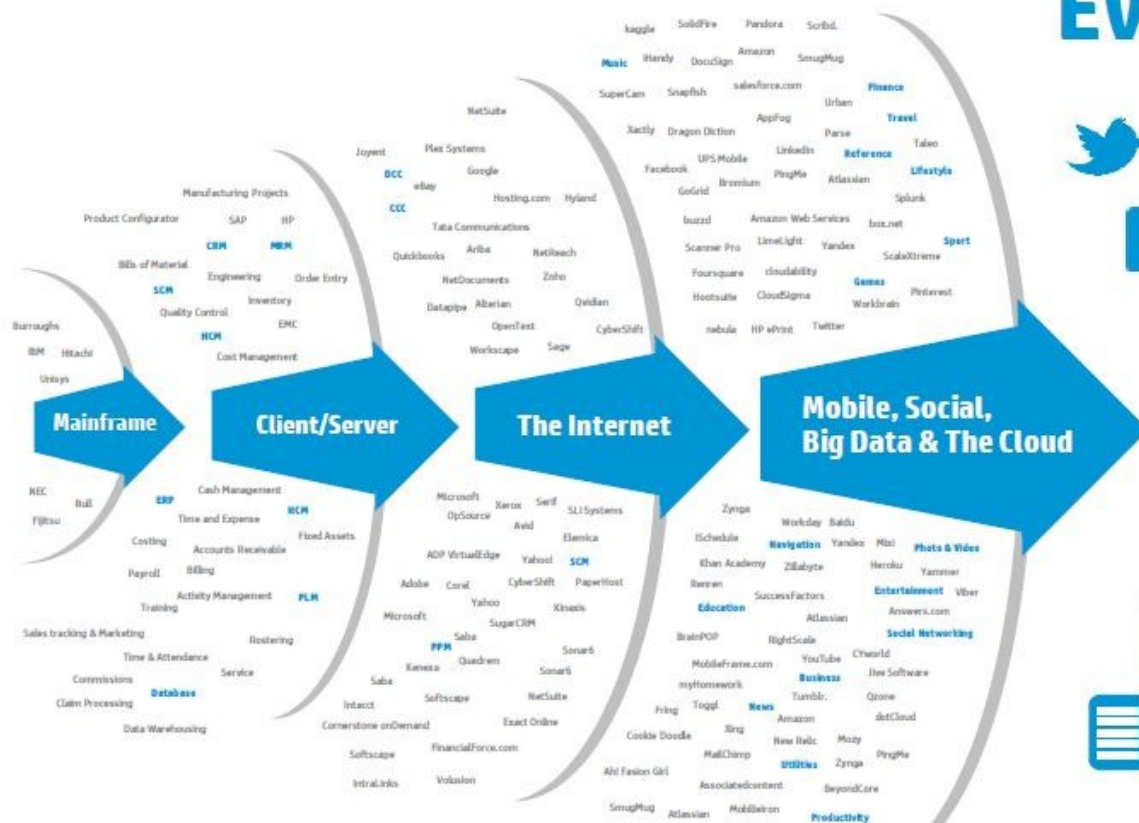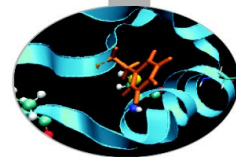
Why?

- **Overwhelming amounts of data generated** by all kinds of devices, networks and programs, *e.g. sensors, mobile devices, internet, social networks, computer simulations, satellites, radiotelescopes, LHC, etc.*
- **Increasing storage capacity**
  - *Storage capacity has doubled every 3 years since 1980 with prices steadily going down*
  - *1,8 zetabytes: an estimation for the data stored by humankind in 2011 (Digital Universe study of International Data Corporation)*
- **Massive data can produce high-value information and knowledge**
- **Critical for data analysis, decision support, forecasting, business intelligence, research, (data-intensive) science, etc.**

Big Data

A buzz word!

- With different meanings depending on your perspective - e.g. 100 TBs is big for a transaction processing system, but small for a world-wide search engine

A simple "definition" (Wikipedia)

- *Consists of data sets that grow so large that they become awkward to work with using on-hand database management tools*
  - Difficulties: capture, storage, search, sharing, analytics, visualizing

How big is big?

- Moving target: terabyte (10^{12} bytes), petabyte (10^{15} bytes), exabyte (10^{18}), zetabyte (10^{21})

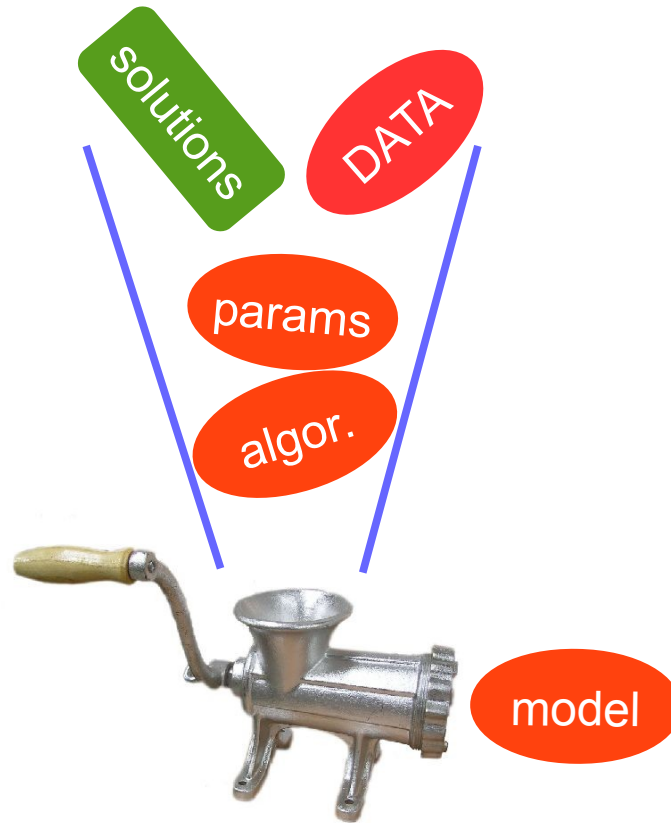**Scale is only one dimension of the problem!**
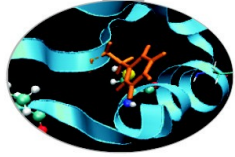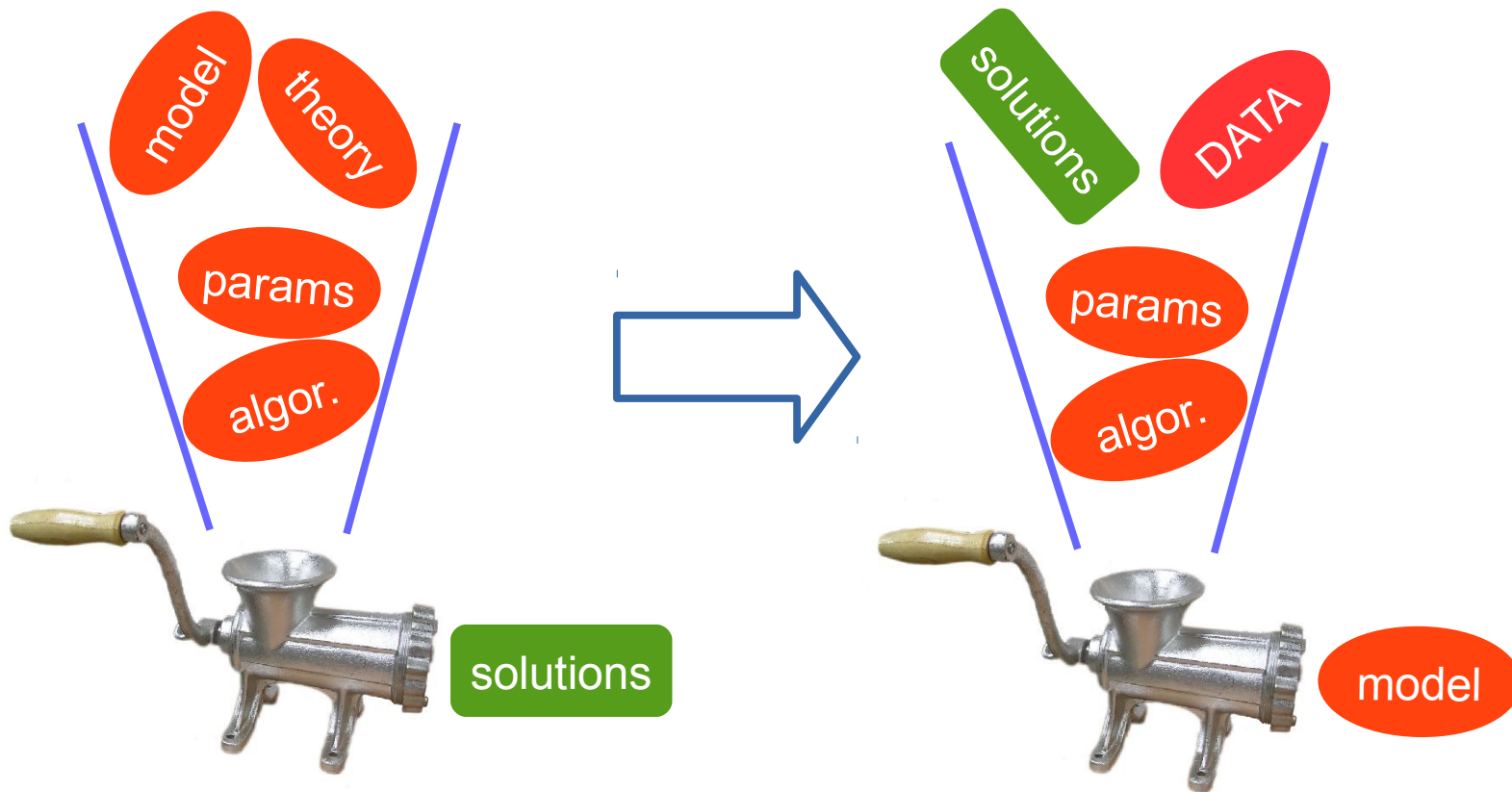
Operate without models
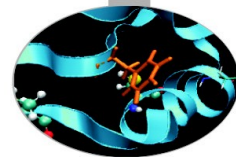## *forward problem*

model

theory

params

algor.

solutions
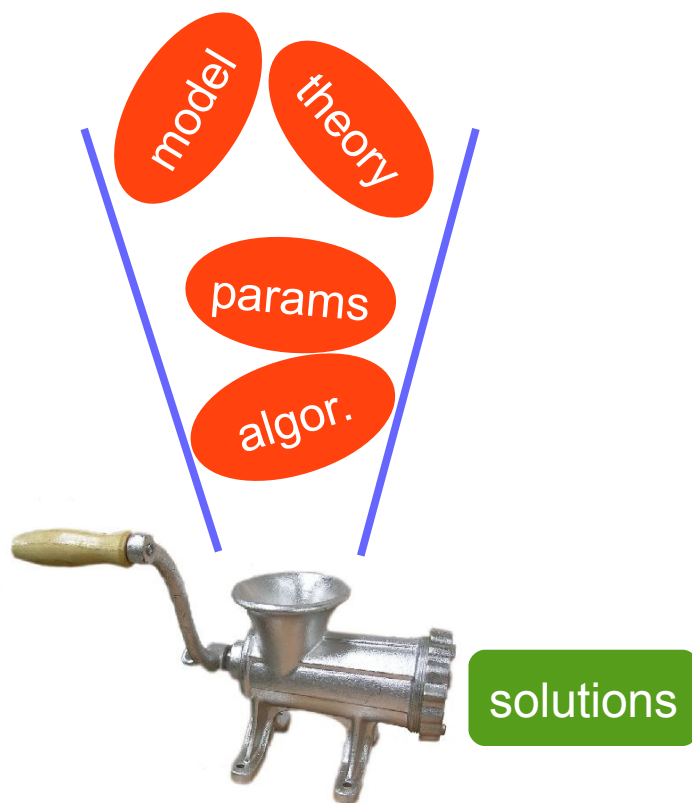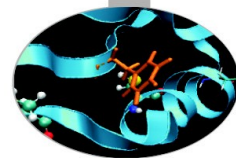
Before...

Operate without models
inverse problem

solutions

DATA

params

algor.

model

Now, future...

Operate without models (Big Data)

Where all began...a decade ago!

The FOURTH PARADIGM

DATA-INTENSIVE SCIENTIFIC DISCOVERY

EDITED BY TONY HEY, STEWART TANSLEY, AND KRISTIN TOLLE
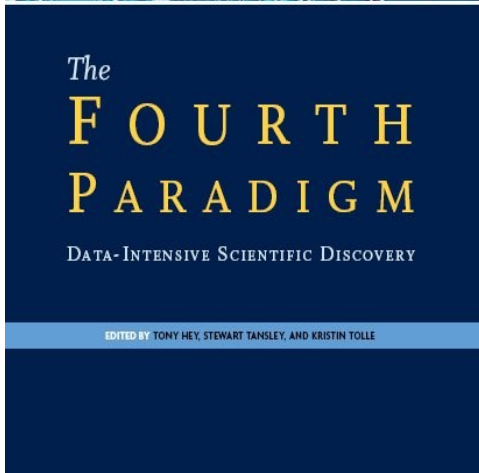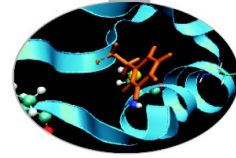
Science Paradigms

- Thousand years ago:
  science was **empirical**
  describing natural phenomena
- Last few hundred years:
  theoretical branch
  using models, generalizations
- Last few decades:
  a **computational** branch
  simulating complex phenomena

$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{4\pi Gp}{3} - K\frac{c^2}{a^2}$$

- Today: **data exploration** (eScience)
  unify theory, experiment, and simulation
  – Data captured by instruments
    or generated by simulator
  – Processed by software
  – Information/knowledge stored in computer
  – Scientist analyzes database/files
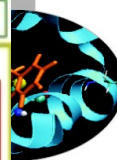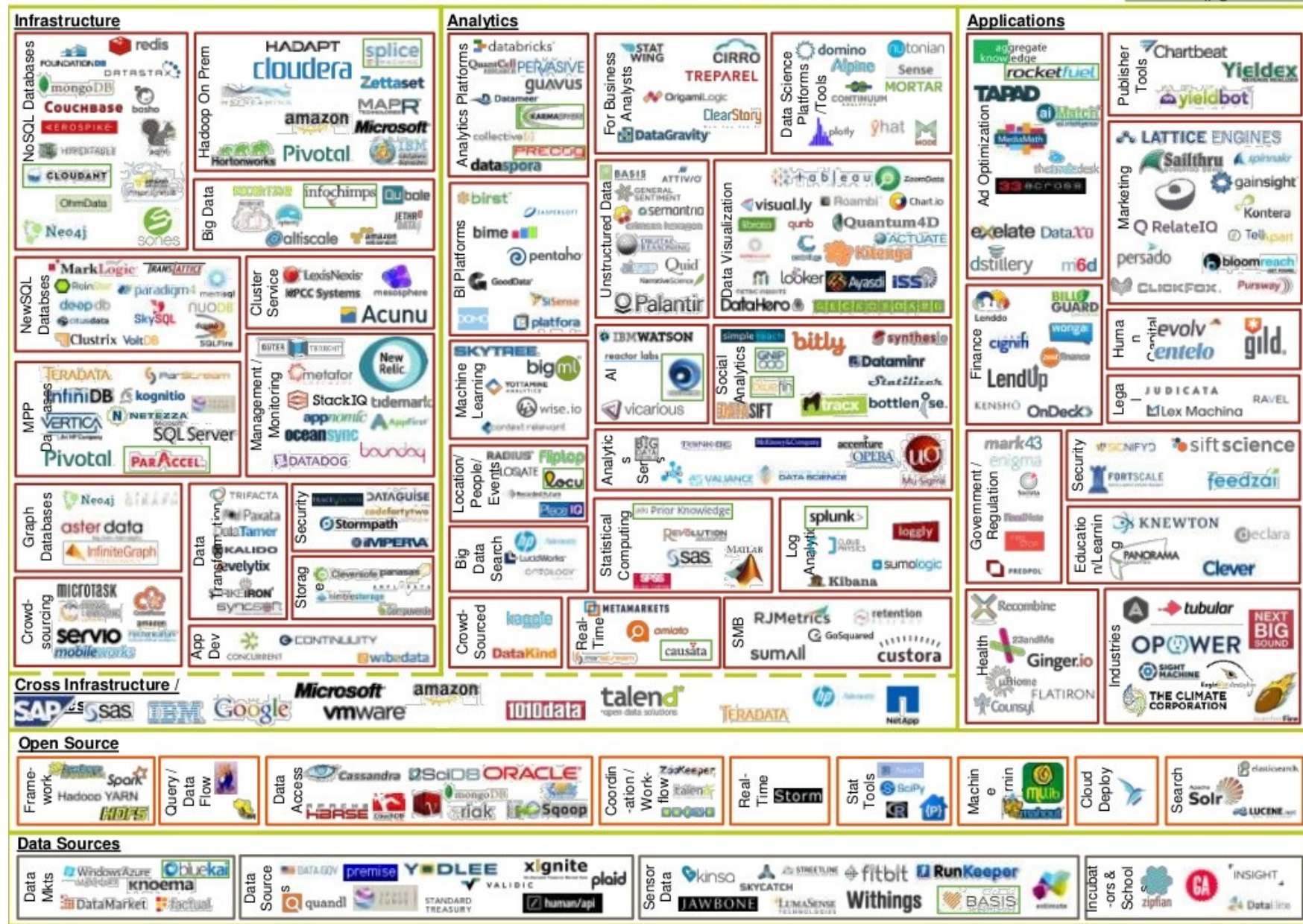   using data management and statistics
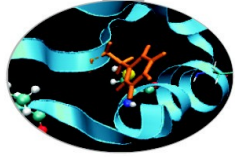
http://research.microsoft.com/en-us/collaboration/fourthparadigm/4th_paradigm_book_complete_lr.pdf

BIG DATA LANDSCAPE, VERSION 3.0

© Matt Turck (@mattturck), Sutian Dong (@sutiandong) & FirstMark Capital (@firstmarkcap)

Dimensions of the problem

- **Volume**
  - Refers to massive amounts of data
  - Makes it hard to store and manage, but also to analyze (big analytics)
- **Velocity**
  - Continuous data streams are being captured (e.g. from sensors or mobile devices) and produced
  - Makes it hard to perform online processing
- **Variety**
  - Different data formats (sequences, graphs, arrays, …), different semantics, uncertain data (because of data capture), multiscale data (with lots of dimensions)
  - Makes it hard to integrate and analyze
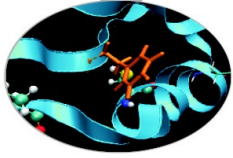
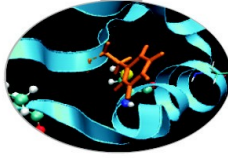# What do we do when there is too much data to process?

Parallel data processing!

- Exploit a massively parallel computer
- A computer that interconnects lots of CPUs, RAM and disk units
- To obtain
  - High performance through data-based parallelism
    - High throughput for transaction-oriented (OLTP) loads
    - Low response time for decision-support (OLAP) queries
  - High availability and reliability through data replication
  - Extensibility with the ideal goals
    - *Linear speed-up*
    - *Linear scale-up*

Speed-up

- Linear increase in performance for a constant database size and load, and proportional increase of the system components (CPU, memory, disk)
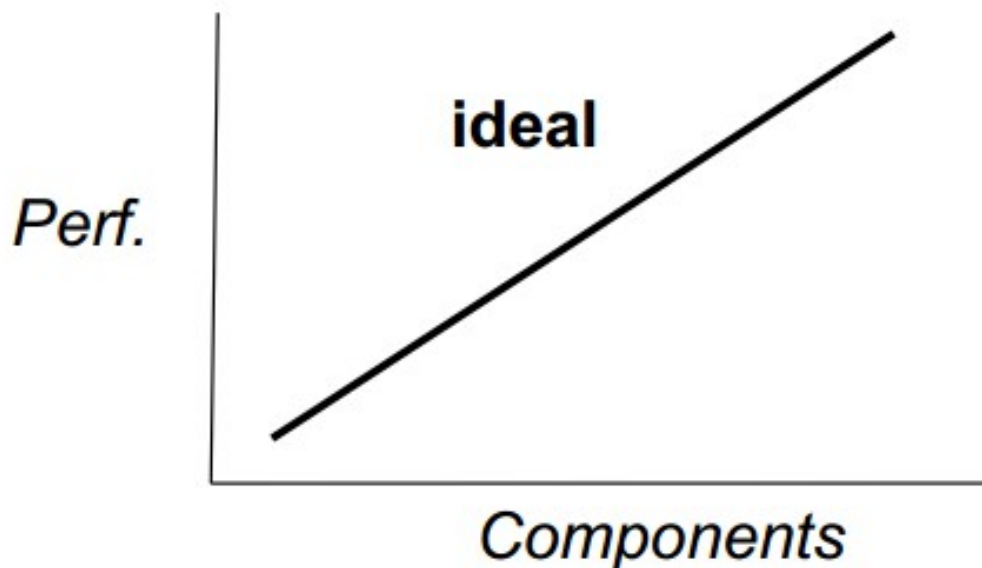
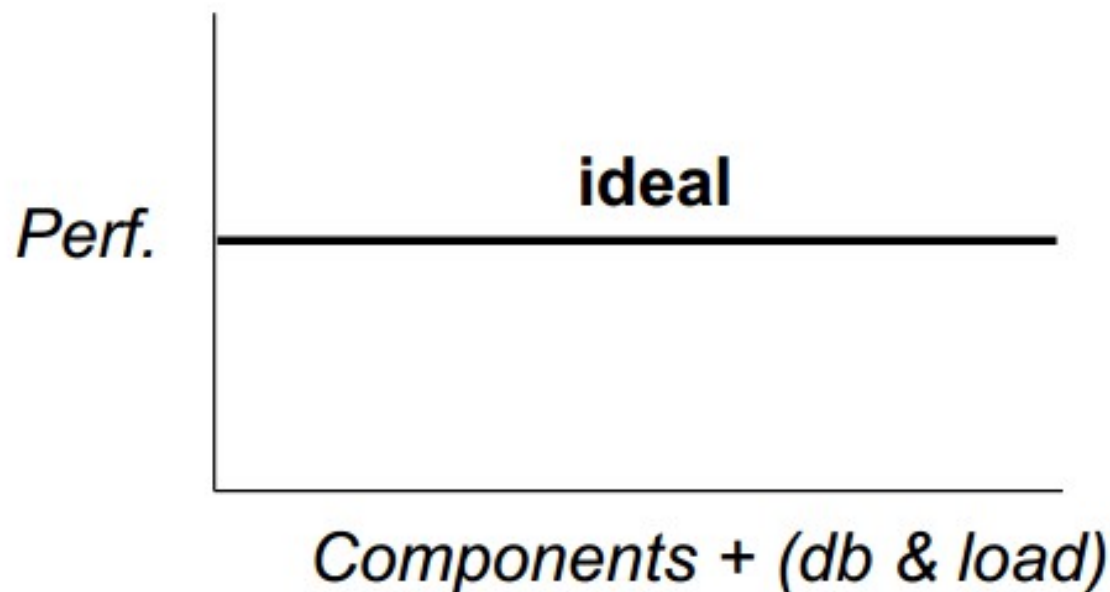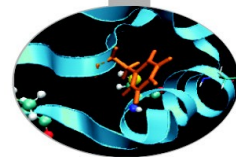# Scale-up

- Sustained performance for a linear increase of database size and load, and proportional increase of components

Perf. — ideal

Components + (db & load)

Parallel Architectures for Data Processing
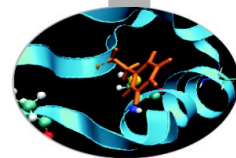
- Three main alternatives, depending on how processors, memory and disk are interconnected
  - *Shared-memory computer*
  - *Shared-disk cluster*
  - *Shared-nothing cluster*
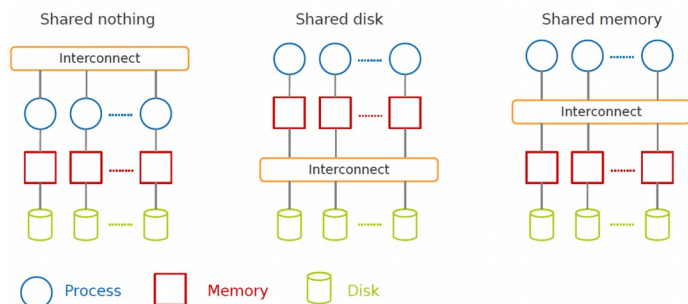
DeWitt, D. and Gray, J. "Parallel database systems: the future of high performance database systems". ACM Communications, 35(6), 85-98, 1992.

Shared Memory
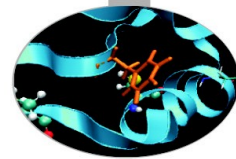
- All memory and disk are shared
  - Symmetric Multiprocessor (SMP)
  - Recent: Non Uniform Memory

+ Simple for apps, fast com., load balancing

- Complex interconnect limits extensibility, cost

- For write-intensive workloads, not for big data
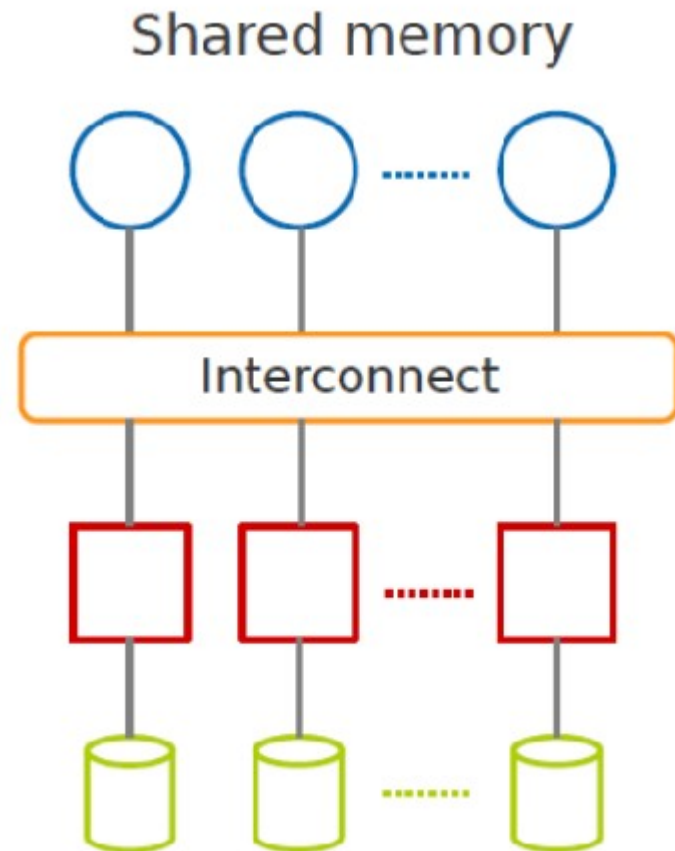
Shared memory
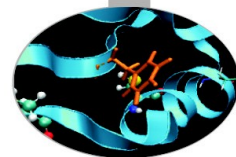
Shared Disk

- Disk is shared, memory is private
    - Storage Area Network (SAN) to interconnect memory and disk (block level)
    - Needs distributed lock manager (DLM) for cache coherence

+ Simple for apps, extensibility

- Complex DLM, cost

- For write-intensive workloads or big data

Shared disk

Interconnect
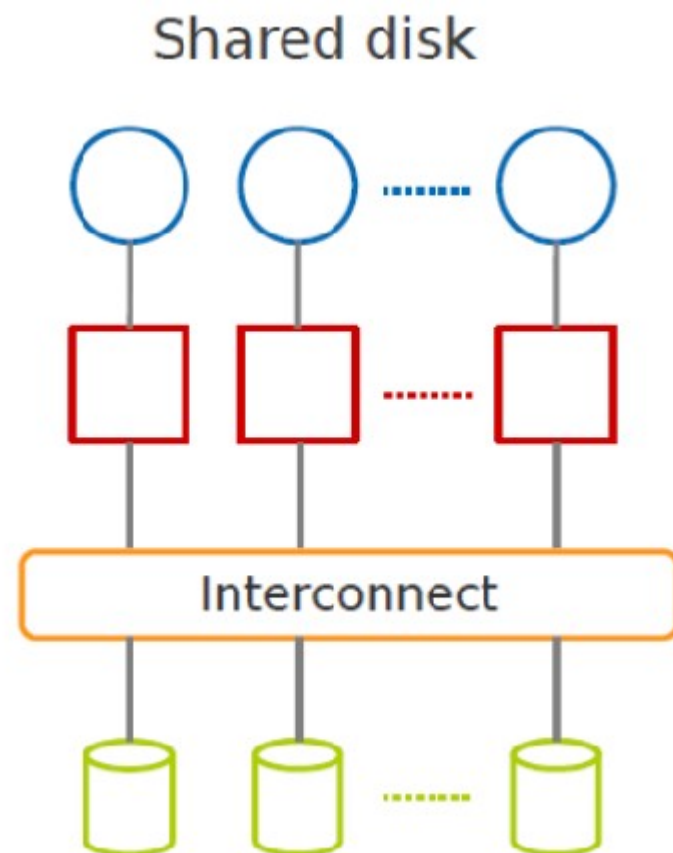
Shared Nothing

- No sharing of memory or disk across nodes
    - No need for DLM
    - But needs data partitioning

+ highest extensibility, cost

- updates, distributed trans.

- For **big data** (read intensive)

Shared nothing

Interconnect
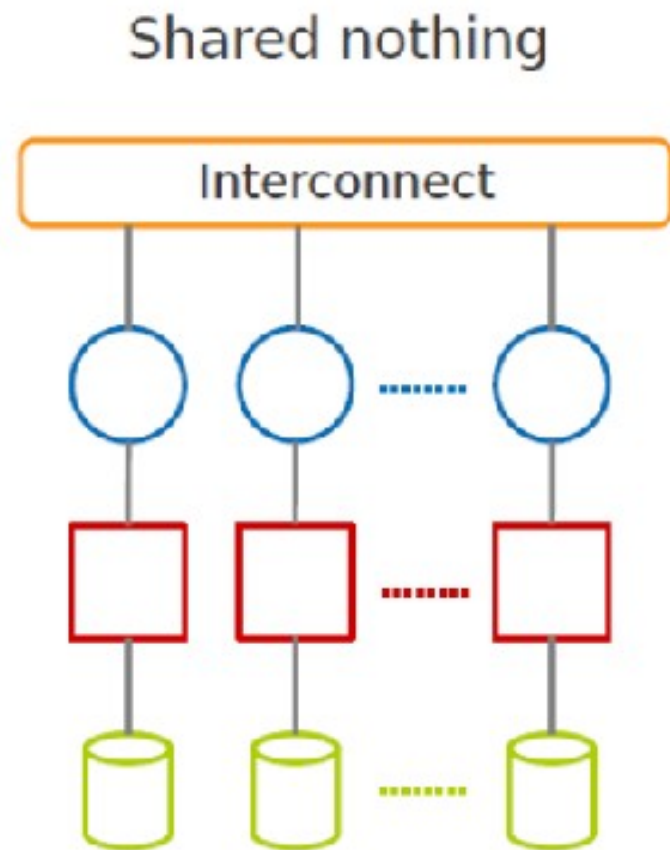
Parallel Architectures

Hadoop/Map-Reduce
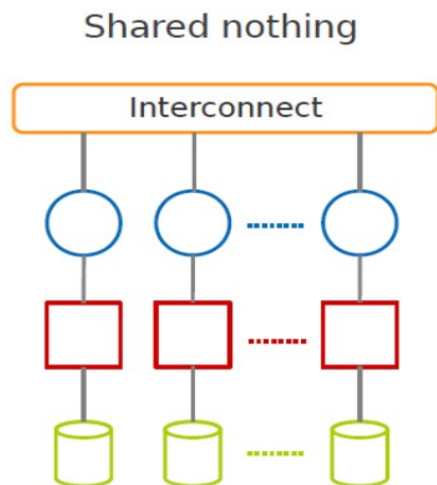
MPI
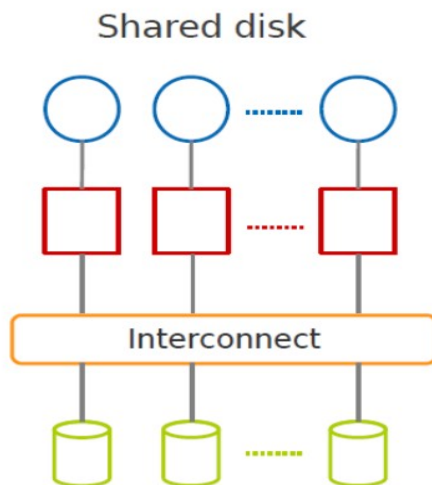
OpenMP
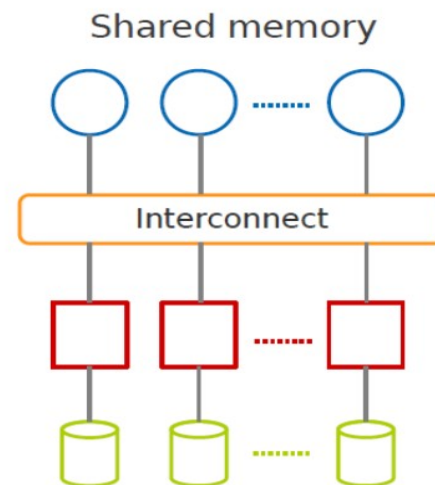
Different classes of applications

- **MPI (Message Passing Interface)**
  - A **shared disk** infrastructure for processing large data sets with a parallel algorithm on clusters

- **OpenMP (Open MultiProcessing)**
  - A **shared memory** infrastructure for processing large data sets with a parallel algorithm on a node

- **Map Reduce/Hadoop**
  - A **shared nothing** architecture for processing large data sets with a distributed algorithm on clusters

Simple Model for Parallel Data
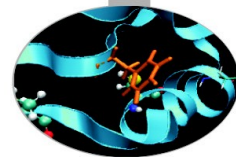
Shared-nothing architecture
  - The most general and scalable

Set-oriented
  - Each dataset D is represented by a table of rows

Key-value
  - Each row is represented by a <key, value> pair, where
    - ➳ Key uniquely identifies the value in D
    - ➳ Value is a list of (attribute name : attribute value)

Can represent structured (relational) data or NoSQL data
  - But graph is another story (see Pregel or DEX)

Examples
  - *<row-id#5, (part-id:5, part-name:iphone5, supplier:Apple)>*
  - *<doc-id#10, (content:<html> html text … </html>)>*
  - *<akeyword, (doc-id:id1, doc-id:id2, doc-id:id10)>*

Considerations

Big datasets
- Data partitioning and indexing
  - Problem with skewed data distributions
- Parallel algorithms for algebraic operators
  - Select is easy, Join is difficult
- Disk is very slow (10K times slower than RAM)
  - Exploit main memory data structures and compression

Query parallelization and optimization
- Automatic if the query language is declarative (e.g. SQL)
- Programmer assisted otherwise (e.g. MapReduce)

Transaction support
- Hard: need for distributed transactions (distributed locks and 2PC)
  - NoSQL systems don't provide transactions

Fault-tolerance and availability
- With many nodes (e.g. several thousand), node failure is the norm

Programming Models: What is MPI?
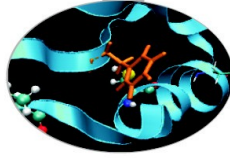
- **Message Passing Interface (MPI)**
  - World's most popular distributed API
  - MPI is "de facto standard" in scientific computing
  - C and FORTRAN, ver. 2 in 1997

- **What is MPI good for?**
  - Abstracts away common network communications
  - Allows lots of control without bookkeeping
  - Freedom and flexibility come with complexity
    - 300 subroutines, but serious programs with fewer than 10

- **Basics:**
  - One executable run on every node
  - Each node process has a rank ID number assigned
  - Call API functions to send messages
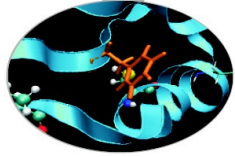
Challenges with MPI

- **Deadlock is possible...**
  - Blocking communication can cause deadlock
    - "crossed" calls when trading information
    - example:
    - `Proc1: MPI_Receive(Proc2, A);  MPI_Send(Proc2, B);`
    - `Proc2: MPI_Receive(Proc1, B);  MPI_Send(Proc1, A);`
    - There are some solutions - MPI_SendRecv()
- **Large overhead from comm. mismanagement**
  - Time spent blocking is wasted cycles
  - Can overlap computation with non-blocking comm.
- **Load imbalance is possible! Dead machines?**
- **Things are starting to look hard to code!**

Data Partitioning

Vertical partitioning

- Basis for column stores (e.g. MonetDB): efficient for OLAP queries
- Easy to compress, e.g. using Bloom filters
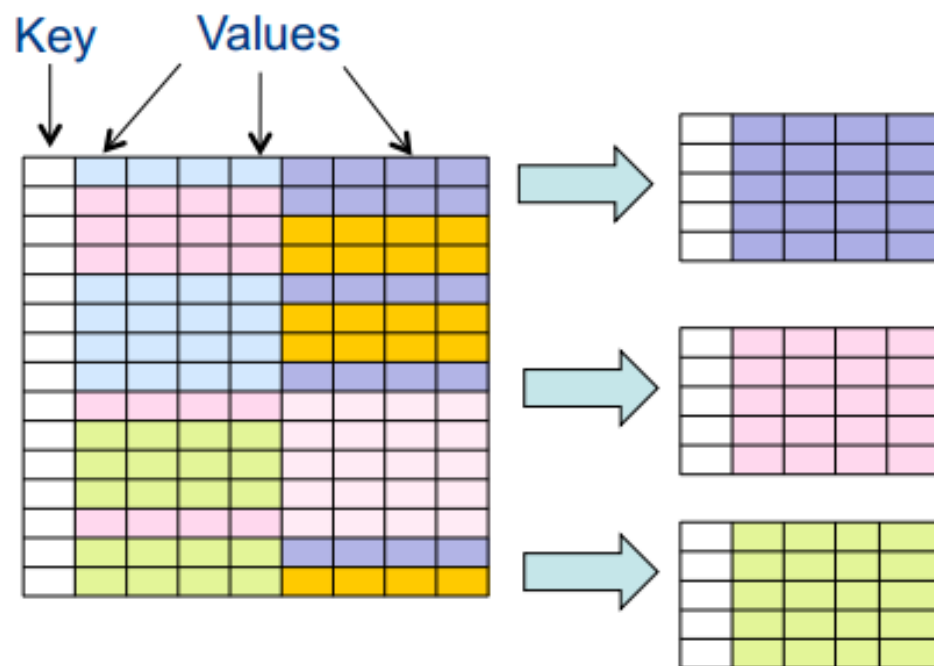
Horizontal partitioning (sharding)

- Shards can be stored (and replicated) at different nodes

Sharding Schemes
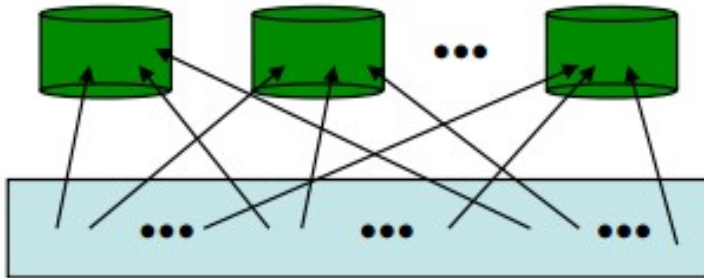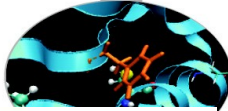
Round-Robin

- i*th* row to node (*i mod n*)
- perfect balancing
- but full scan only
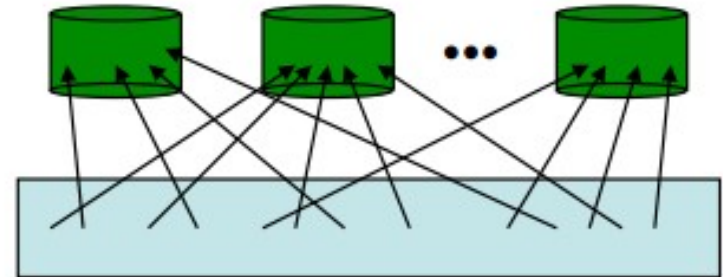
Hashing

- (k,v) to node h(k)
- exact-match queries
- but problem with skew

Range

a-g	h-m	•••	u-z
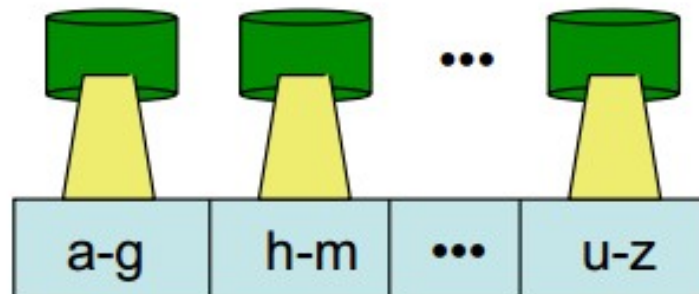
Range

- (k,v) to node that holds k's interval
- exact-match and range queries
- deals with skew
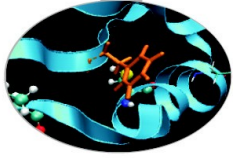
Are emerging data analytics techniques the new El Dorado?

Where and When using Hadoop
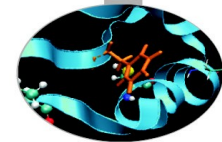
Where

- Batch data processing, not real-time

- Highly parallel data intensive distributed applications

- Very large production deployments

When

- Process lots of unstructured data

- When your processing can easily be made parallel

- Running batch jobs is acceptable

- When you have access to lots of cheap hardware

Advantages/Disadvantages
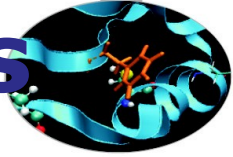
- **Now it's easy to program for many CPUs**
  - Communication management effectively gone
    - I/O scheduling done for us
  - Fault tolerance, monitoring
    - machine failures, suddenly-slow machines, etc are handled
  - Can be much easier to design and program!

- **But ... it further restricts solvable problems**
  - Might be hard to express problem in MapReduce
  - Data parallelism is key
  - Need to be able to break up a problem by data chunks
  - MapReduce is closed-source (to Google) C++
  - Hadoop is open-source Java-based rewrite

Parallel Computing Model
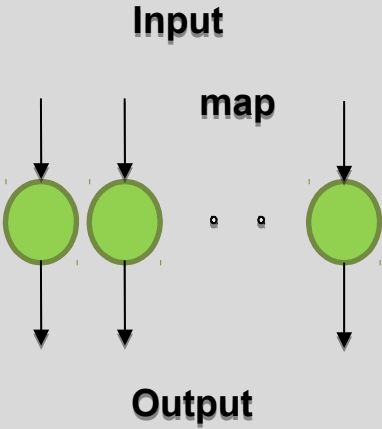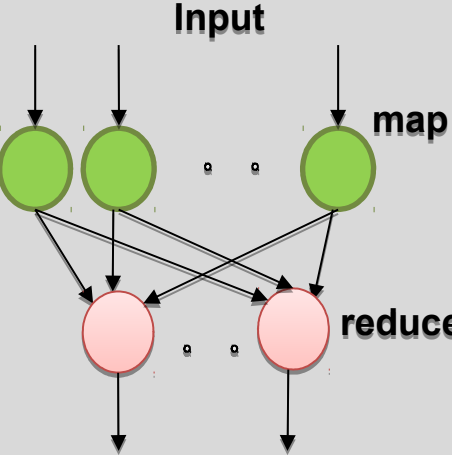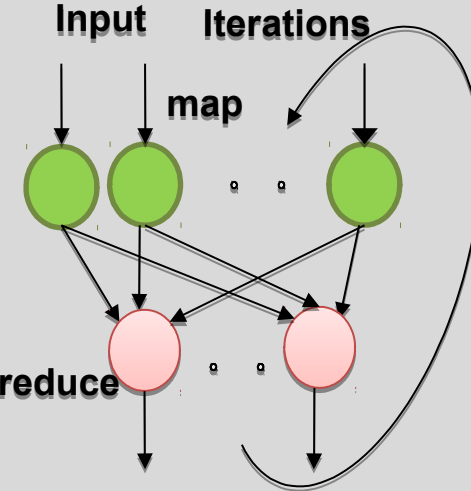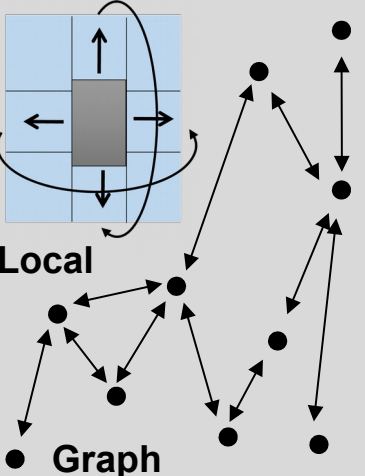
MapReduce can be classified as a SIMD (single-instruction, multiple-data) problem.

- Indeed, the map step is highly scalable because the same instructions are carried out over all data. Parallelism arises by breaking the data into independent parts with no forward or backward dependencies (side effects) within a Map step; that is, the Map step may not change any data (even its own).

- The reducer step is similar, in that it applies the same reduction process to a different set of data (the results of the Map step).

- In general, the MapReduce model provides a functional, rather than procedural, programing model. Similar to a functional language, MapReduce cannot change the input data as part of the mapper or reducer process, which is usually a large file. Such restrictions can at first be seen as inefficient; however, the lack of side effects allows for easy scalability and redundancy.

An HPC cluster, on the other hand, can run SIMD and MIMD (multiple-instruction, multiple-data) jobs.

- The programmer determines how to execute the parallel algorithm. Users, however, are not restricted when creating their own MapReduce application within the framework of a typical HPC cluster.

(1) Map Only	(2) Classic MapReduce	(3) Iterative Map Reduce or Map-Collective	(4) Point to Point or Map-Communication
BLAST Analysis Local Machine Learning Pleasingly Parallel	High Energy Physics (HEP) Histograms Distributed search Recommender Engines	Expectation maximization Clustering e.g. K-means Linear Algebra, PageRank	Classic MPI PDE Solvers and Particle Dynamics Graph Problems
MapReduce and Iterative Extensions (Spark, Twister)			MPI, Giraph
Integrated Systems such as Hadoop + Harp with Compute and Communication model separated			
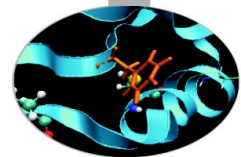
Word count
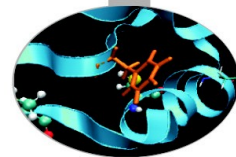
First example

- **Map function**: processes data and generates a set of intermediate key/value pairs.
- **Reduce function**: merges all intermediate values associated with the same intermediate key.
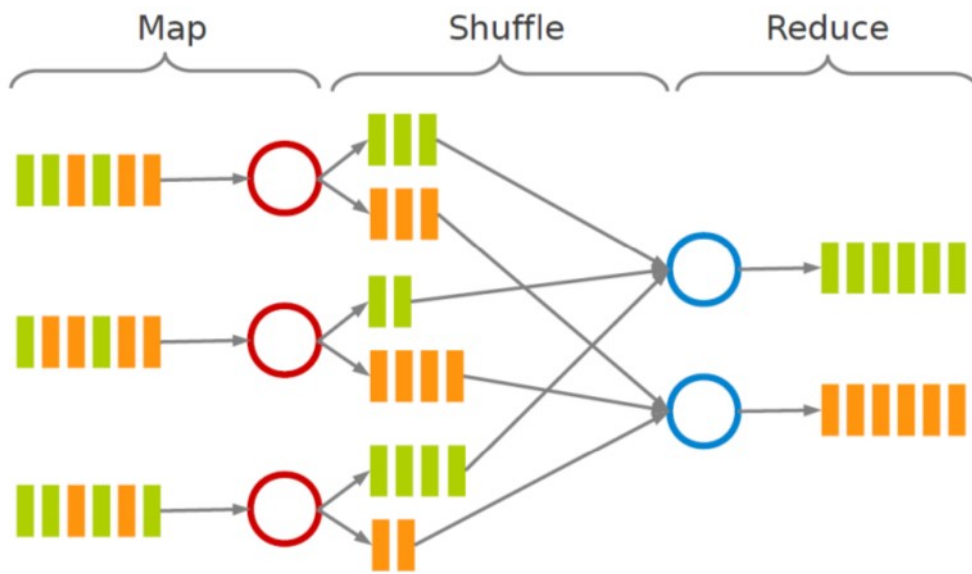
Word count execution
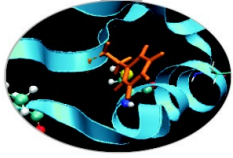
- Consider doing a word count of the following file using MapReduce:

  - **Hello World Bye World**
  - **Hello Hadoop Goodbye Hadoop**

Word count
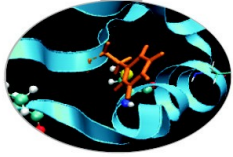
- The map function reads in words one a time and outputs (word, 1) for each parsed input word.

- The map function output is:

(Hello, 1)
(World, 1)
(Bye, 1)
(World, 1)
(Hello, 1)
(Hadoop, 1)
(Goodbye, 1)
(Hadoop, 1)

Word count
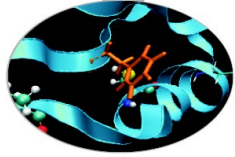
- The shuffle phase between map and reduce phase creates a list of values associated with each key.

- The reduce function input is:

(Bye, (1))
(Goodbye, (1))
(Hadoop, (1, 1)
(Hello, (1, 1))
(World, (1, 1))

Word count
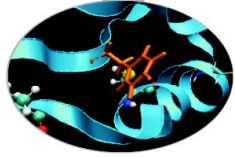
- The reduce function sums the numbers in the list for each key and outputs (word, count) pairs.
- The output of the reduce function is the output of the MapReduce job:

(Bye, 1)
(Goodbye, 1)
(Hadoop, 2)
(Hello, 2)
(World, 2)

MRJob code
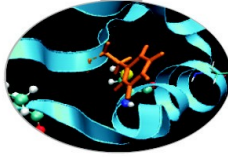
```python
from mrjob.job import MRJob
class MRWordCount(MRJob):

    def mapper(self, key, line):
        for word in line.split(' '):
            yield word.lower(),1


    def reducer(self, word, occurrences):
        yield word, sum(occurrences)

if __name__ == '__main__':
    MRWordCount.run()
```

Testing the code

$ git clone **https://github.com/gfiameni/course-exercises.git**

$ docker run -v ~/course-exercises:/course-exercises -i -t **cineca/hadoop-mrjob:1.2.1** /etc/bootstrap.sh -bash

root$ show-exercises

root$ source /root/mrjob0.4.2/bin/activate

root$ python mrjob/word_count.py data/txt/2261.txt.utf-8 (-r hadoop)

Word count (combiner)

```python
from mrjob.job import MRJob
class MRWordCount2(MRJob):

    def mapper(self, key, line):
        for word in line.split(' '):
                        yield word.lower(),1

    # Combiner step
    def combiner(self, word, occurrences):
        yield word, sum(occurrences)

    def reducer(self, word, occurrences):
        yield word, sum(occurrences)

if __name__ == '__main__':
    MRWordCount2.run()
```

How to execute jobs with MRJob
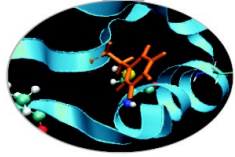
By default, output will be written to stdout.

- `$ python my_job.py input.txt`

You can pass input via stdin, but be aware that mrjob will just dump it to a file first:
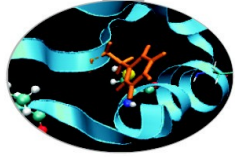
- `$ python my_job.py < input.txt`

You can pass multiple input files, mixed with stdin (using the – character)

- `$ python my_job.py input1.txt input2.txt - < input3.txt`

By default, mrjob will run your job in a single Python process. This provides the friendliest debugging experience, but it's not exactly distributed computing!

You change the way the job is run with the -r/--runner option (`-r` **inline**, `-r local, -r hadoop, or -r emr`)

Use "`--verbose`" to show all the steps
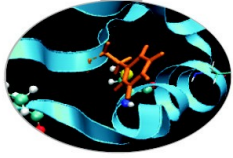
When to use MR + Hadoop

When to use MR + Hadoop
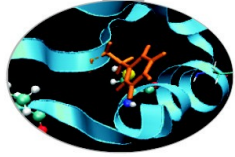
- **Your Data Sets Are Really Big**
    - *Don't even think about Hadoop if the data you want to process is measured in MBs or GBs.* If the data driving the main problem you are hoping to use Hadoop to solve is measured in GBs, save yourself the hassle and use Excel, a SQL BI tool on Postgres, or some similar combination. On the other hand, if it's several TB or (even better) measured in petabytes, Hadoop's superior scalability will save you a considerable amount of time and money

- **You Celebrate Data Diversity**
    - One of the advantages of the Hadoop Distributed File System (HDFS) is it's really flexible in terms of data types. It doesn't matter whether your raw data is structured, semi-structured (like XML and log files), unstructured (like video files).

When to use MR + Hadoop

- **You Find Yourself Throwing Away Perfectly Good Data**
    - One of the great things about Hadoop is its capability to store petabytes of data. If you find that you are throwing away potentially valuable data because its costs too much to archive, you may find that setting up a Hadoop cluster allows you to retain this data, and gives you the time to figure out how to best make use of that data.

When to NOT use MR + Hadoop
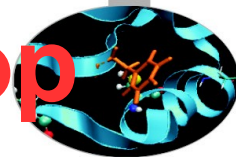
- **You Need Answers in a Hurry**
  - Hadoop is probably not the ideal solution if you need really fast access to data. The various SQL engines for Hadoop have made big strides in the past year, and will likely continue to improve. But if you're using Map-Reduce to crunch your data, expect to wait days or even weeks to get results back.

- **Your Queries Are Complex and Require Extensive Optimization**
  - Hadoop is great because it gives you a massively parallel cluster for low-cost Lintel servers and scads of cheap hard disk capacity. While the hardware and scalability is straightforward, getting the most out of Hadoop typically requires a hefty investment in the technical skills required to optimize queries.

When to NOT use MR + Hadoop
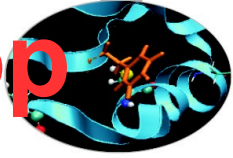
- **You Require Random, Interactive Access to Data**
  - The pushback from the limitations of the batch-oriented MapReduce paradigm in early Hadoop led the community to improve SQL performance and boost its capability to serve interactive queries against random data. While SQL on Hadoop is getting better, in most cases it's not a reason in of itself to adopt Hadoop.

- **You Want to Store Sensitive Data**
  - Hadoop is evolving quickly and is able to do a lot of things that it couldn't do just a few years ago. But one of the things that it's not particularly good at today is storing sensitive data. Hadoop today has basic data and use access security. And while these features are improving by the month, the risks of accidentally losing personally identifiable information due to Hadoop's less-than-stellar security capabilities is probably not worth the risk.

Overall limitations
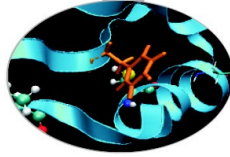
- *MapReduce provides an easy-to-use framework for parallel programming, but is it the most efficient and best solution to program execution in datacenters?*

- ***DeWitt and Stonebraker: "MapReduce: A major step backwards"*** – MapReduce is far less sophisticated and efficient than parallel query processing

- **MapReduce is a parallel processing framework, not a database system, nor a query language**
    - It is possible to use MapReduce to implement some of the parallel query processing function

- **What are the real limitations?**
    - Inefficient for general programming (and not designed for that)
    - Hard to handle data with complex dependence, frequent updates, etc.
    - High overhead, bursty I/O, difficult to handle long streaming data
    - Limited opportunity for optimization

What if

- If you have access to a Hadoop cluster and you want a one-off quick-and-dirty job…
  - *Hadoop Streaming*
- If you don't have access to Hadoop and want to try stuff out…
  - *MrJob*
- If you're heavily using AWS…
  - *MrJob*
- If you want to work interactively…
  - *PySpark*
- If you want to do in-memory analytics…
  - *PySpark*
- If you want to do anything…*
  - *PySpark*
- If you want ease of Python with high performance
  - *Impala + Numba*
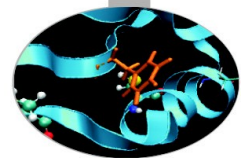
The PICO system

The PICO system
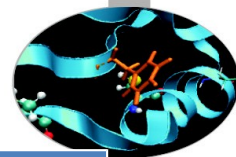
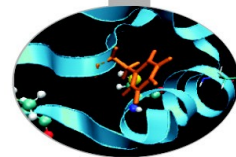|	Total Nodes	CPU	Cores per Nodes	Memory (RAM)	Notes
Compute login node	66	Intel Xeon E5 2670 v2 @2.5Ghz	20	128 GB	
Visualization node	2	Intel Xeon E5 2670 v2 @ 2.5Ghz	20	128 GB	2 GPU Nvidia K40
Big Mem node	2	Intel Xeon E5 2650 v2 @ 2.6 Ghz	16	512 GB	1 GPU Nvidia K20
BigInsight node	4	Intel Xeon E5 2650 v2 @ 2.6 Ghz	16	64 GB	32TB of local disk
SSD Storage					40 TB

http://www.hpc.cineca.it/hardware/pico

PICO: how to log in
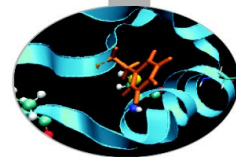
- Establish a ssh connection

  `ssh <username>@login.pico.cineca.it`

- Notes:

  - **ssh** available on all linux distros
  - **Putty** (free) or **Tectia** ssh on Windows
  - *secure shell plugin* for Google Chrome!
  - login nodes are swapped to keep the load balanced
  - important messages can be found in the *message of the day*

71

Working environment

$HOME:

- Permanent, backed-up, and local to PICO.
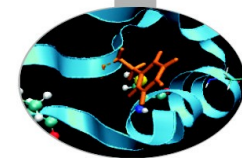- For source code or important input files.

$CINECA_SCRATCH:

- Large, parallel filesystem (GPFS).
- No quota. Run your simulations and calculations here.

- use the command `cindata` command to get info on your disk occupation

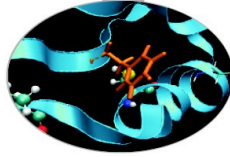   http://www.hpc.cineca.it/content/data-storage-and-filesystems-0

"module", my best friend

- **All the optional software on the system is made available through the "module" system**
  - provides a way to rationalize software and its environment variables

- Modules are divided in 2 *profiles*
  - **profile/base** (stable and tested modules)
  - **profile/advanced** (software not yet tested or not well optimized)

- Each profile is divided in 4 categories
  - **compilers** (GNU, intel, openmpi)
  - **libraries** (e.g. LAPACK, BLAS, FFTW, ...)
  - **tools** (e.g. Hadoop, GNU make, VNC, ...)
 - **applications** (software for chemistry, physics, ...)

Modules

- CINECA's work environment is organized in modules, a set of installed libraries, tools and applications available for all users.

- "loading" a module means that a series of (useful) shell environment variables will be set

- E.g. after a module is loaded, an environment variable of the form "<MODULENAME>_HOME" is set
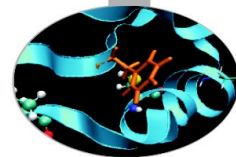
```
[amarani0@fen07 ~]$ module load namd
[amarani0@fen07 ~]$ ls $NAMD_HOME
backup   flipbinpdb   flipdcd   namd2   namd2_plumed   namd2_remd   psfgen   sortreplicas
```

Module commands
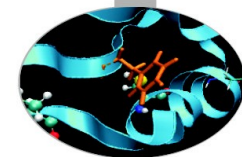
COMMAND	DESCRIPTION
`module avail`	list all the available modules
`module load <module_name(s)>`	load module <module_name>
`module list`	list currently loaded modules
`module purge`	unload all the loaded modules
`module unload <module_name>`	unload module <module_name>
`module help <module_name>`	print out the help (hints)
`module show <module_name>`	print the env. variables set when loading the module

Launching a Job

- Now that we have our executable, it's time to learn how to prepare a job for its execution

- PICO uses **PBS scheduler**.
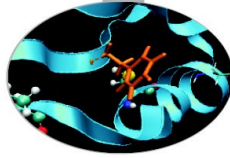
- The job script scheme is:

```
#!/bin/bash
#PBS keywords

variables environment

execution line
```
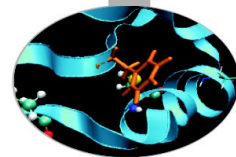
PBS keywords

```
#PBS -N jobname # name of the job
#PBS -o job.out    # redirect stdout (output file)
#PBS -e job.err    # redirect stderr (error file)
#PBS -l select=1:ncpus=20::mem=96gb   # resources
#PBS -l walltime=1:00:00      # hh:mm:ss
#PBS -q <queue-name>  # chosen queue
#PBS -A <my_account>     # name of the account
```

select = number of chunk requested

ncpus = number of cpus per chunk requested

mem = RAM memory per chunk
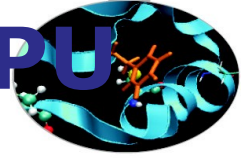
PBS keyword - resource

Memory per node:

- The default memory is 1 GB per node (for the classes debug, parallel and longpar).
- The user can specify the requested memory up to 128 GB, on 58 nodes

```
#PBS -l select=NN:ncpus=CC:mem=128GB
```
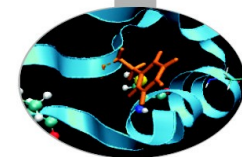
PBS job script – Serial using 1 GPU

```
#!/bin/bash
#PBS -l walltime=30:00
#PBS -l select=1:ncpus=1
#PBS -o job.out
#PBS -e job.err
#PBS -q debug
#PBS -A train_cmda2014

cd $PBS_O_WORKDIR

./myProgram
```
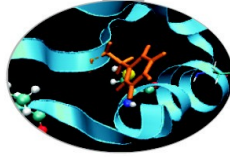
PBS Commands

qsub

qsub <job_script>

Your job will be submitted to the PBS scheduler and executed when there will be nodes available (according to your priority and the queue you requested)

qstat

qstat

Shows the list of all your scheduled jobs, along with their status (idle, running, closing, …) Also, shows you the job id required for other qstat options
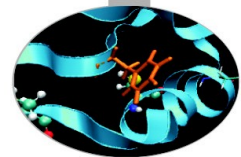
PBS Commands

qstat

qstat -f <job_id>

Provides a long list of informations for the job requested.

In particular, if your job isn't running yet, you'll be notified about its estimated start time or, if you made an error on the job script, you will learn that the job won't ever start
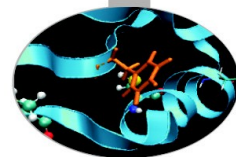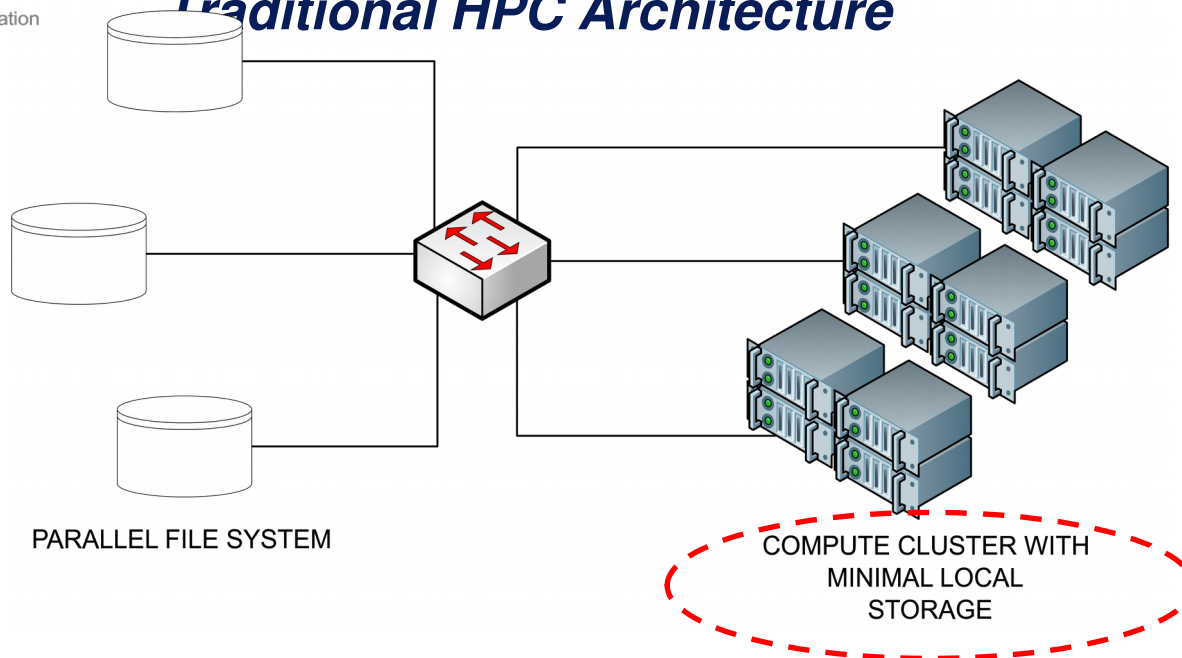
qdel

qdel <job_id>
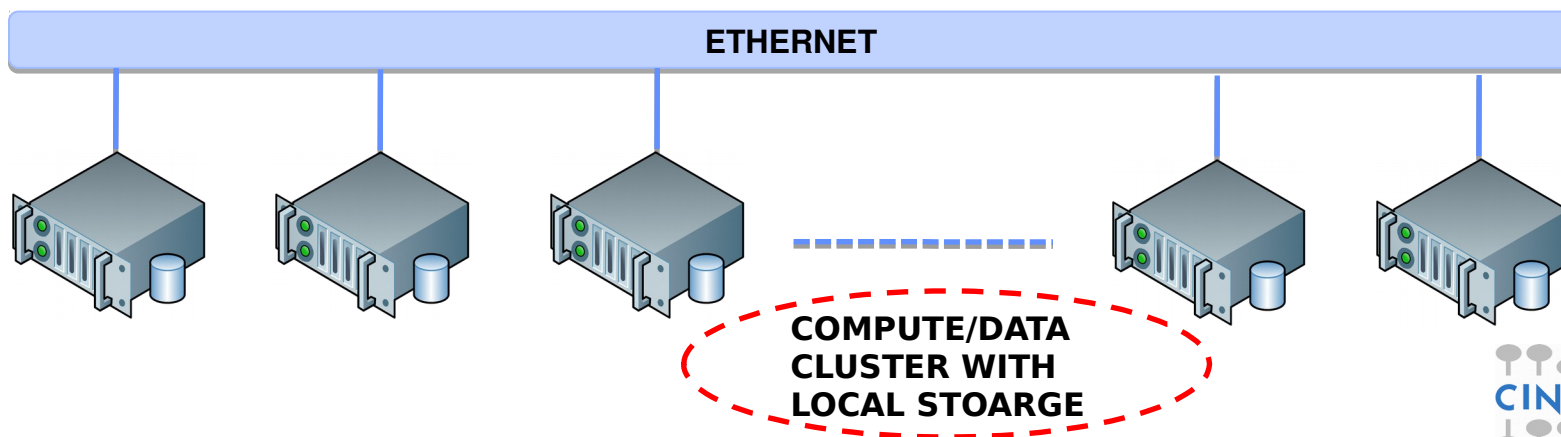
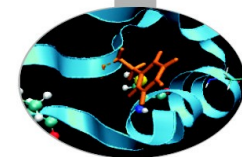Removes the job from the scheduled jobs by killing it

Hadoop on PICO

Traditional HPC Architecture

PARALLEL FILE SYSTEM

COMPUTE CLUSTER WITH MINIMAL LOCAL STORAGE

Shared-nothing (MapReduce-style) Architectures

ETHERNET

COMPUTE/DATA CLUSTER WITH LOCAL STOARGE
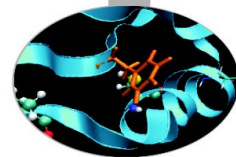
PBS Script

```
#!/bin/bash
#PBS -A <account>
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=20:mem=96GB
#PBS -q parallel

## Environment configuration
module load profile/advanced hadoop/1.2.1
# Configure a new HADOOP instance using PBS job information
$MYHADOOP_HOME/bin/myhadoop-configure.sh -c $HADOOP_CONF_DIR
# Start the Datanode, Namenode, and the Job Scheduler
$HADOOP_HOME/bin/start-all.sh
###################################

# Your job goes here

# Stop HADOOP services
$MYHADOOP_HOME/bin/myhadoop-shutdown.sh
```
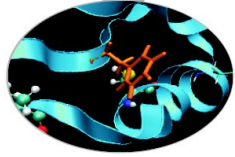
Sample execution

- Login on PICO
    - `ssh login.pico.cineca.it -l <username>`

- Download source codes within **$HOME** or **$CINECA_SCRATCH**

- Change the selected PBS script accordingly to the destination directory

- *qsub $HOME/course-exercises/pbs-scripts/mrjob/wordcount/word-count.hadooop.pbs*
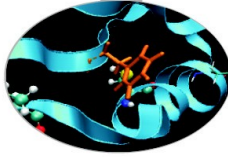
- *qstat*

Sample execution (cont.)

Output:

- word-count.hado.o3041 **// std output**
- word-count.hado.o3042 **// std error**

Credits

- **Geoffrey C. Fo**x – Indiana University
- **Hanspeter Pfister and Joe Blitzstein** – Harvard University
- **Borja Sotomayor** – University of Chicago
- **Glenn K. Lockwood** – High-Performance and Data-Intensive Computing San Francisco Bay Area