Running a typical ROOT HEP analysis on Hadoop/MapReduce

Stefano Alberto Russo – Michele Pinamonti – Marina Cobal

CHEP 2013 – Amsterdam – 14-18/10/2013

Topics

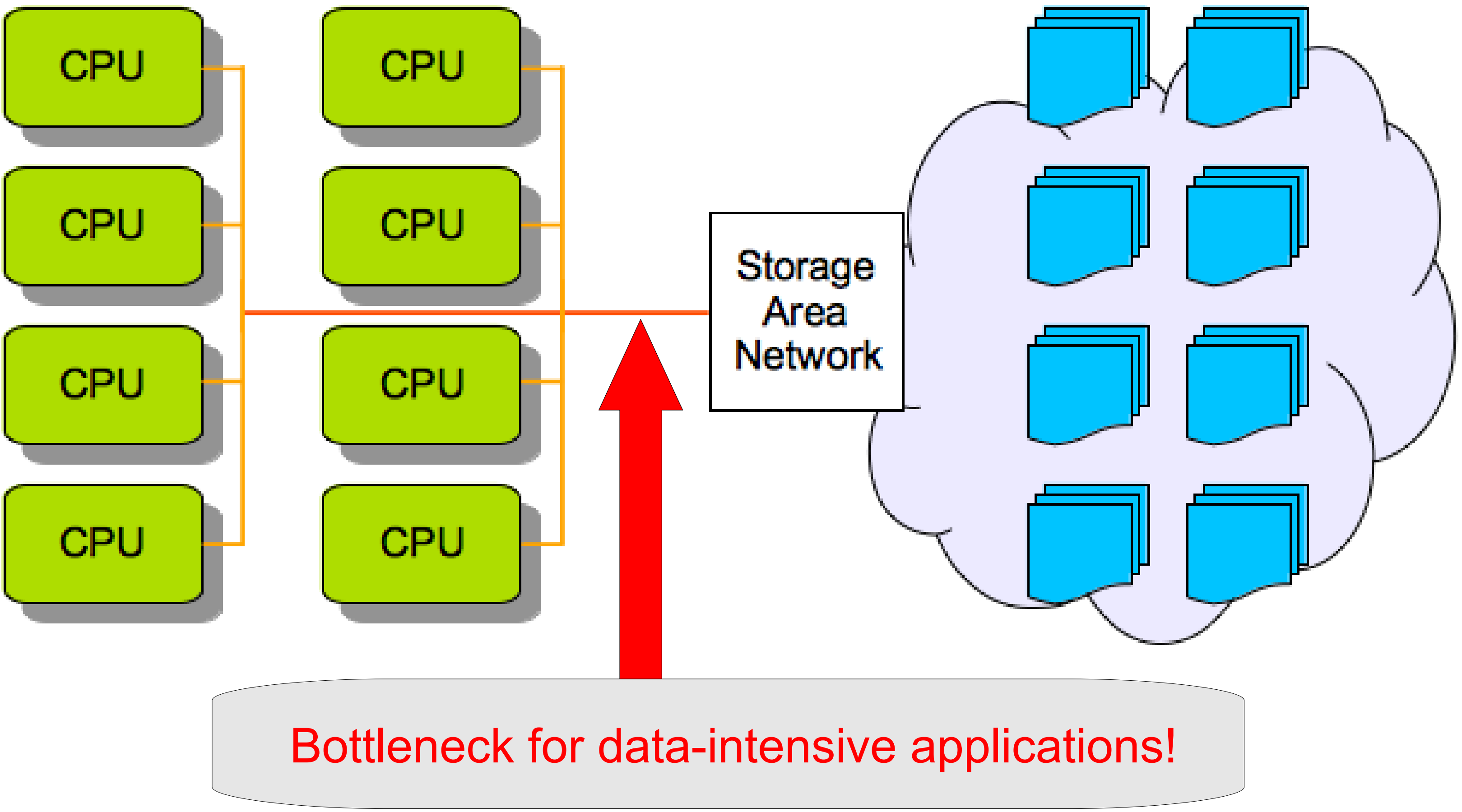
- The Hadoop/MapReduce model
- Hadoop and High Energy Physics
- How to run ROOT on Hadoop
- A real case: a top quark analysis
- Results and conclusions

DISCLAIMER:
This talk is about computing architecture, it is not a not performance study.

Background

"Standard" distributed computing model:

storage and computational resources of a cluster as two independent, well logically-separated components.



The Hadoop/MapReduce model

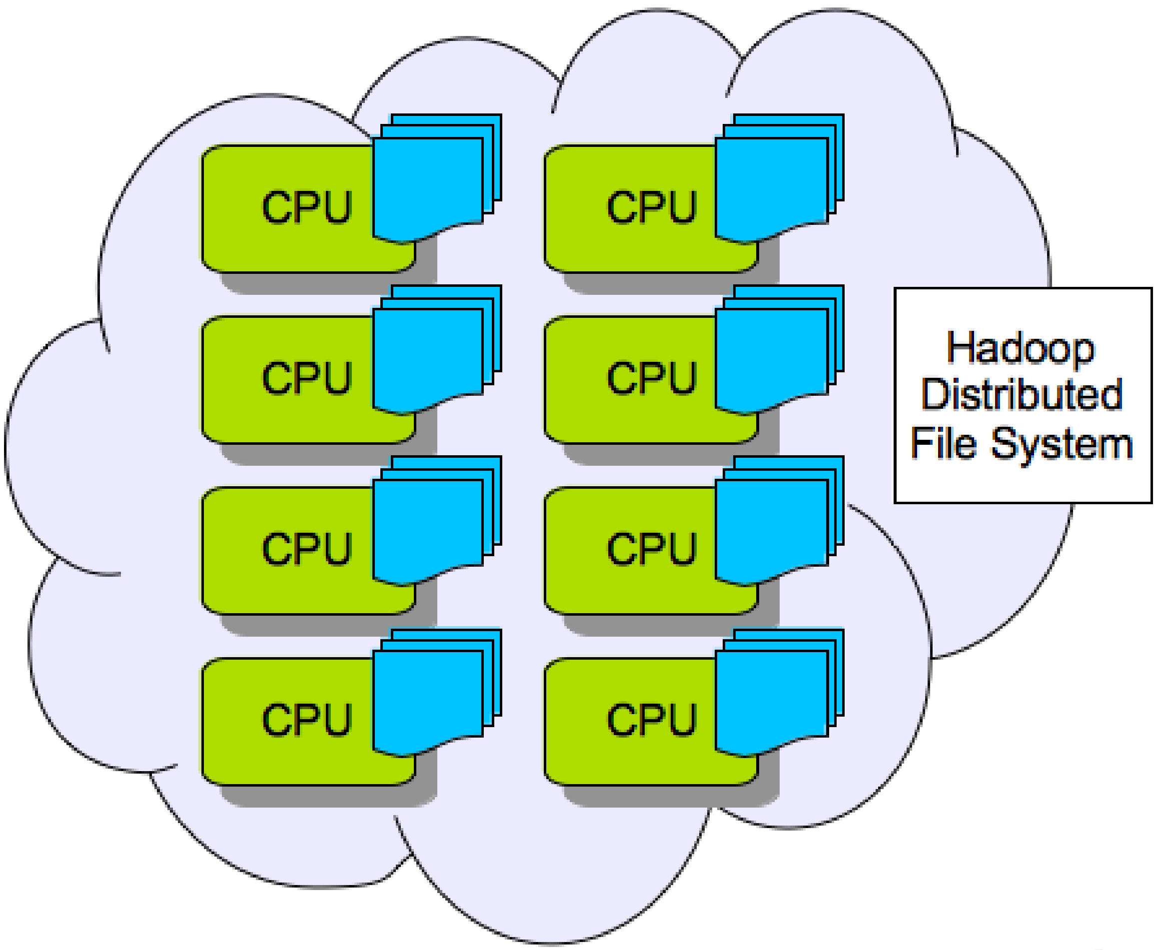
New idea: overlap storage elements with the computing ones

the computation can be scheduled on the cluster elements holding a copy of the data to analyze: *data locality*

Two components:

1. The Hadoop Distributed File System (HDFS)

2. The MapReduce computational model and framework



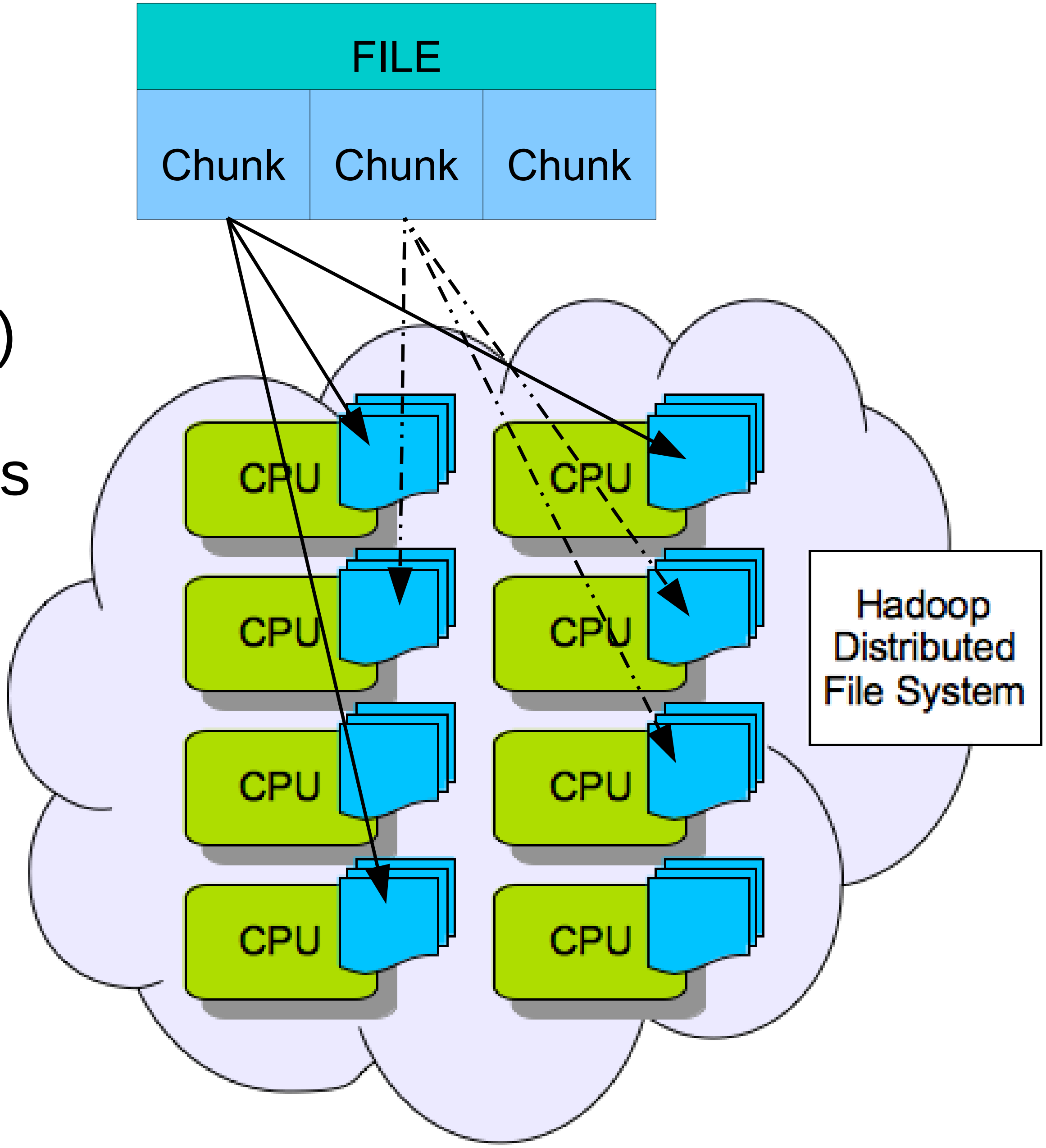
The Hadoop Distributed File System (HDFS)

On HDFS, files are:

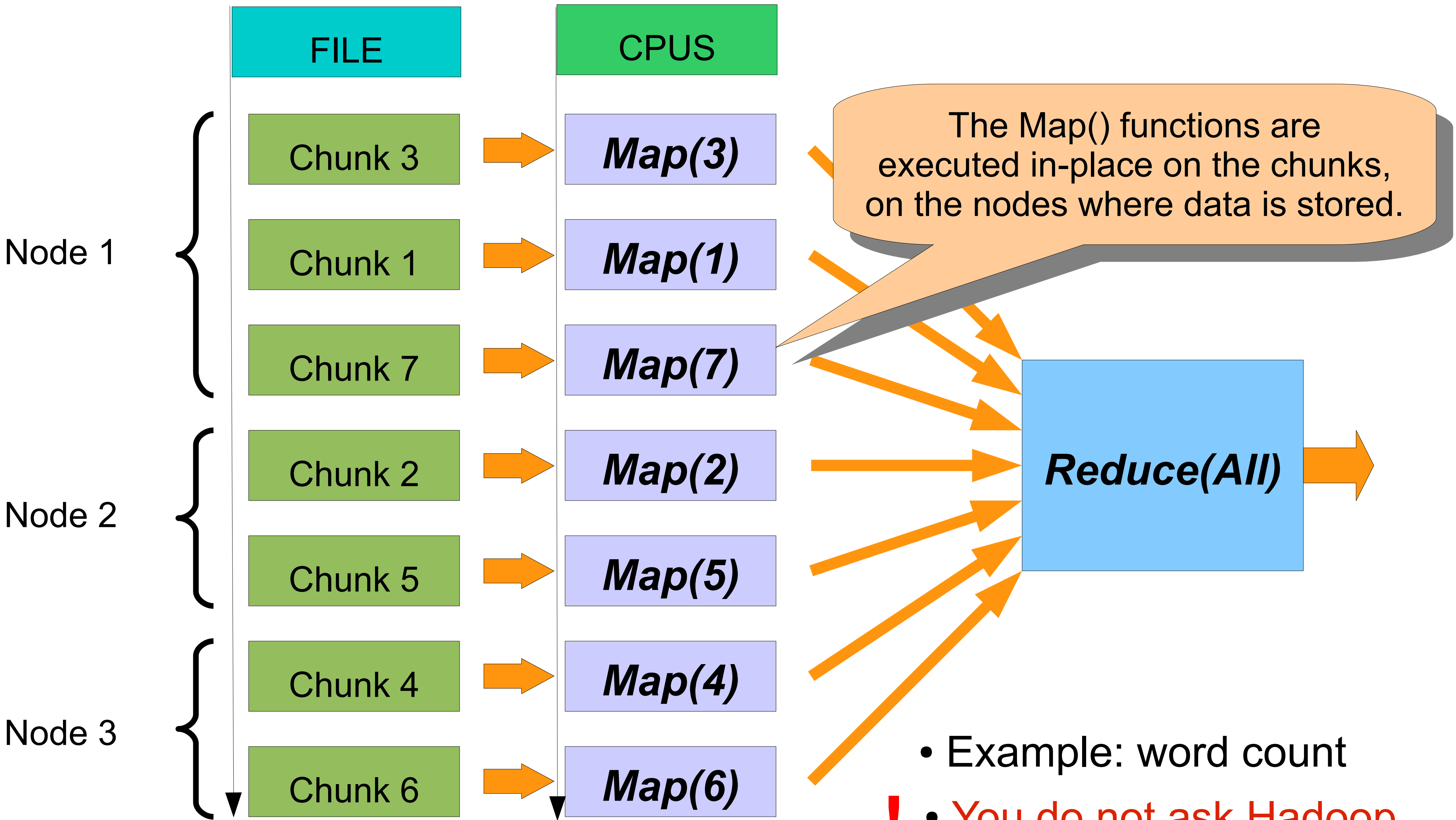
- Stored by slicing them in **chunks** (i.e. 64 MB, 1 GB)

- ..which are **replicated** across the cluster for redundancy and workload distribution.

- No RAID
- Commodity hardware:
  a disk can (and will) fail, sooner or later



The MapReduce model and framework



- Example: word count

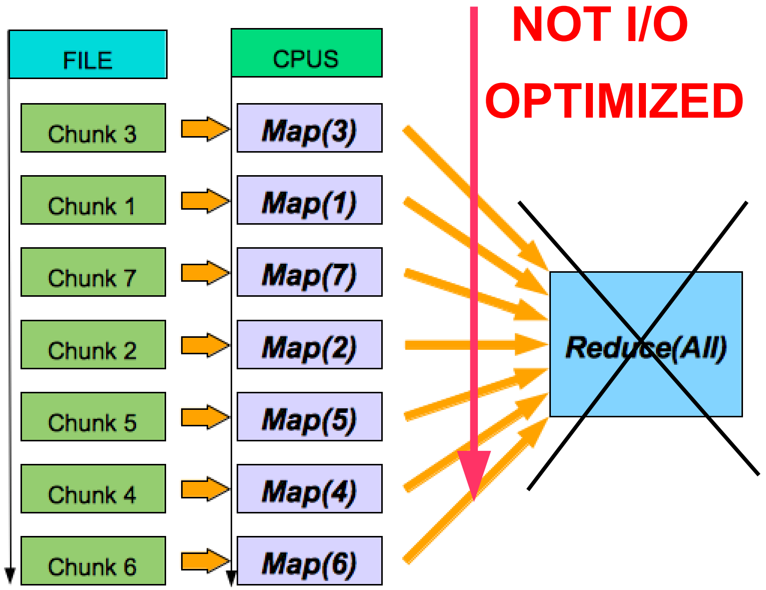
! • You do not ask Hadoop for cpu slots, you ask to analyze a dataset

The MapReduce model and framework

MapReduce requires an *embarrassing parallel* problem.

No communication between Maps...

Another basic assumption: a trivial Reduce phase.
→ easy to compute and almost I/O free



Hadoop and HEP (1)

In High Energy Physics (HEP):

Particle collision events are *independent*:
embarrassing parallel problems

Simple merging operations: **sum numbers, sum historgrams..**

Usually, data to analyse accessed *over and over again* to finalize physics results**: potential advantage from data localiy**

(Store once, read many)

Hadoop and HEP (2)

“Natural” approach:

- **Map:** processes a chunk of the data set, analysing it *event by event*
- **Reduce:** collect Map's partial result merging them.

Hadoop and HEP (2)

“Natural” approach:

- **Map:** processes a chunk of the data set, analysing it *event by event*
- **Reduce:** collect Map's partial result merging them.

Drowbacks:

Not column-based storage ...

1) Events in plain text, CSV style: lot of unnecessary I/O reads
(typical HEP analysis requires only a few out of the many variables available)

→ Ref: Maaike Limper, *An SQL-based approach to Physics Analysis,* CHEP2013

Hadoop and HEP (2)

"Natural" approach:

- **Map:** processes a chunk of the data set, analysing it *event by event*
- **Reduce:** collect Map's partial result merging them.

Drowbacks:

Not column-based storage ...

1) Events in plain text, CSV style: lot of unnecessary I/O reads
(typical HEP analysis requires only a few out of the many variables available)

→ Ref: Maaike Limper, *An SQL-based approach to Physics Analysis,* CHEP2013

2) Frameworks for HEP developed, maintained and used by large communities over several years (ROOT):
- porting code could be very challenging and time consuming
...and non-optimised MapReduce code can easily lead to waste CPU

→ Ref: Zbigniew Baranowski, et Al, *Sequential Data access with Oracle and Hadoop: a performance comparison,* CHEP2013

Hadoop and HEP (3)

IDEA:

- run **ROOT on Hadoop**, and
- use its **original data format** which provides column-based storage.

GOALS:

1) Transparency for the data:
let binary datasets be uploaded on HDFS without changing format;

2) Transparency for the code:
let the original code run without having to modify a single line;

3) Transparency for the user:
avoid the users to have to learn Hadoop/MapReduce, and let them interact with Hadoop in a classic, batch-fashioned behavior.

Hadoop and HEP (4)

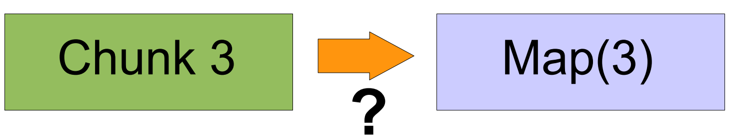
PROBLEMS:

- The Hadoop/MapReduce framework and its native API are written in the Java programming language.

- Support for other programming languages is provided, **but**:
  serious limitations on the input/output side when working with binary data sets. *(Hadoop was developed with textual analyses in mind)*

- ROOT data is binary

...chunking binary files without corrupting data is NOT possible!



ROOT on Hadoop/MapReduce (1)

SOLUTIONS: Transparency for the (binary) data

NO chunking:

One Map = One file = one HDFS block (chunk)

(set chunk size >= file size per file)

- Map tasks will be in charge of analyzing one file, in its entirety
  - Corruptions due to chunking binary data are avoided
    - Data can be stored on the Hadoop cluster without conversions, in its original format.

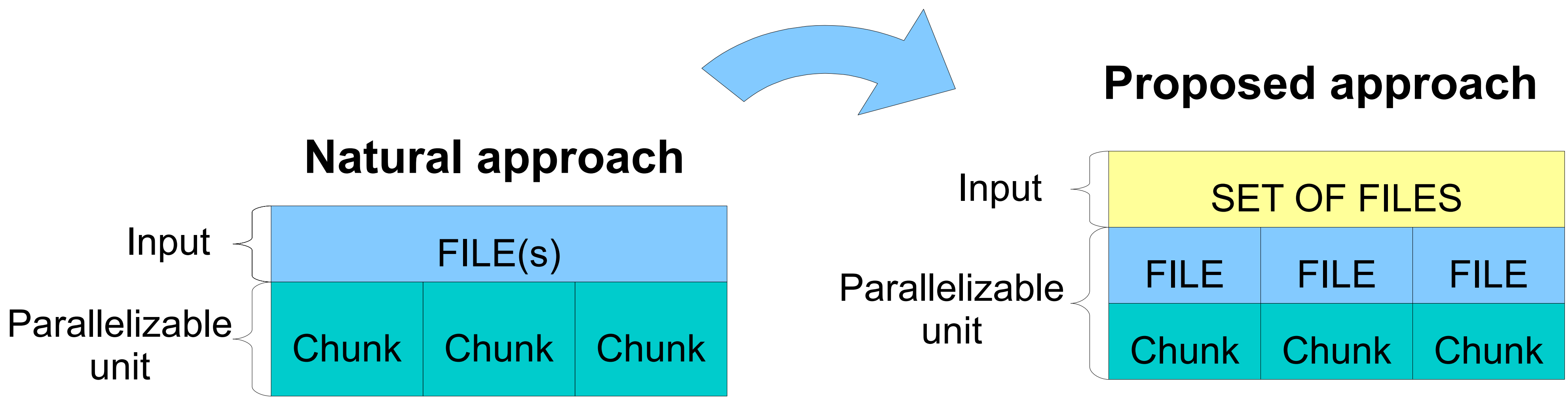
Other approaches are possible, but much more effort required

ROOT on Hadoop/MapReduce (1.1)

SOLUTIONS: ...and what about parallelism?

Working conditions imposed:

One Map Task = One chunk = one file to analyze



Now the parallelization degree goes with the number of files!

ROOT on Hadoop/MapReduce (1.2)

SOLUTIONS: ...and what about parallelism?

HEP datasets are usually composed by *several* files

I.e. ATLAS D3PD's storage schema:

Object	Order of Magnitude	Type	On Hadoop/Mapreduce
Event	1	ROOT data	Unknown (binary)
File	10^2 - 10^4	ROOT file	One chunk
Luminosity block	10^4	Set of Files	Directory
LHC Run	10^5 - 10^6	Set of Lum. blocks	Directory
Data set	10^5 - 10^9	Set of LHC Runs	Directory (input dataset)

Dataset: ~ 10^3-10^5 files ✓

ROOT on Hadoop/MapReduce (2)

SOLUTIONS: Transparency for the code

Bottom line: bypass Hadoop

1. Java Map and Reduce tasks as *wrappers for ROOT*

2. Let ROOT access the data from a *standard file system*

For every Map task:

- **Local replica available:**

  → HDFS file (block) to analyze can be found and therefore accessed on the local, standard file system, i.e. Ext3.

- **Local replica *not* available**:

  → access the file to analyze via network using Hadoop's file system tools

or.. use FUSE

ROOT on Hadoop/MapReduce (3)

SOLUTIONS: Transparency for the user

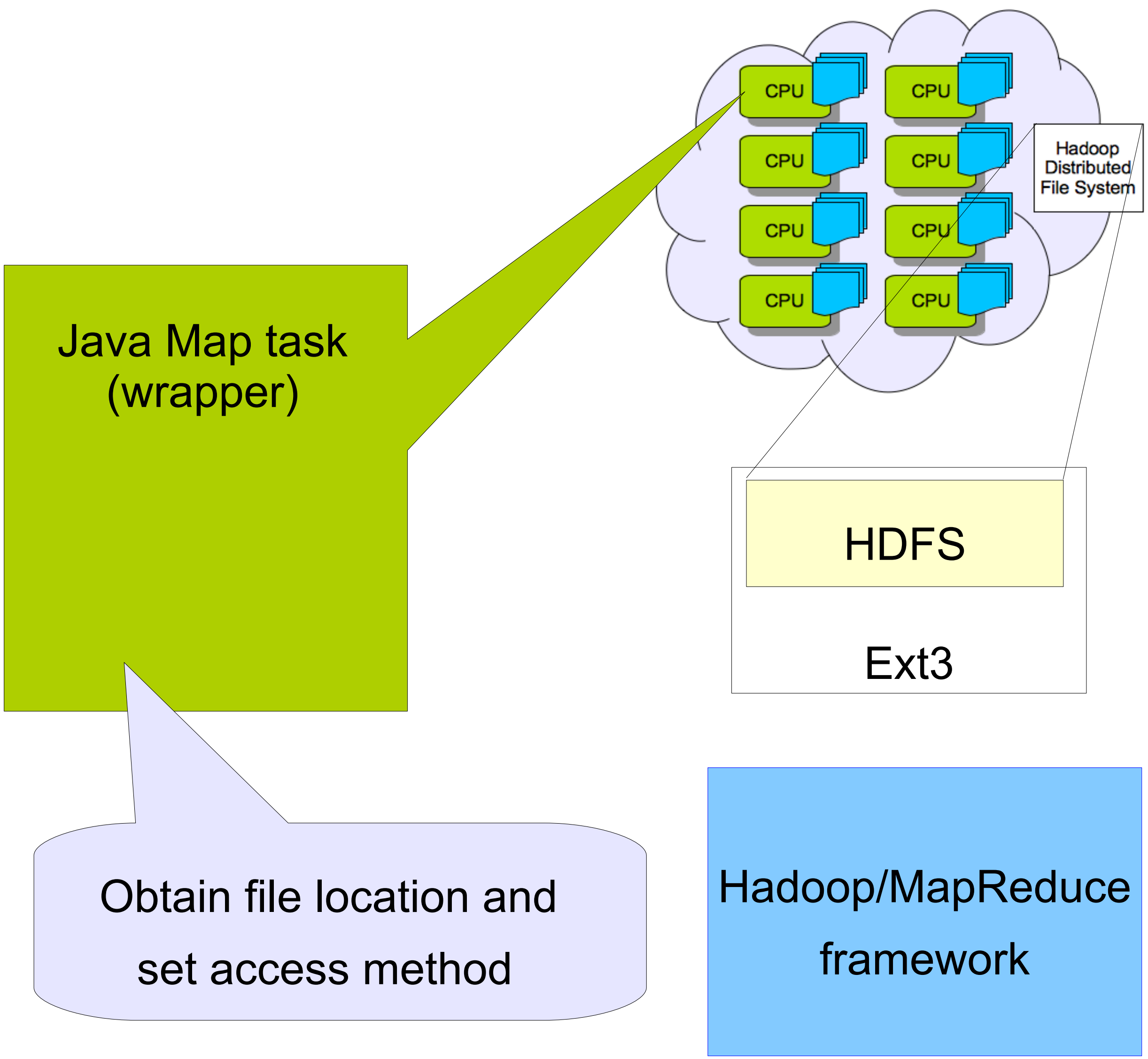
Easy to write a Java MapReduce job acting as a wrapper for user's code, i.e **RootOnHadoop.java**:

```
# hadoop run RootOnHadoop “user Map code” “user Reduce
   code” “HDFS input dataset“ “HDFS output location”
```

- Just few guidelines for the user code to make it work

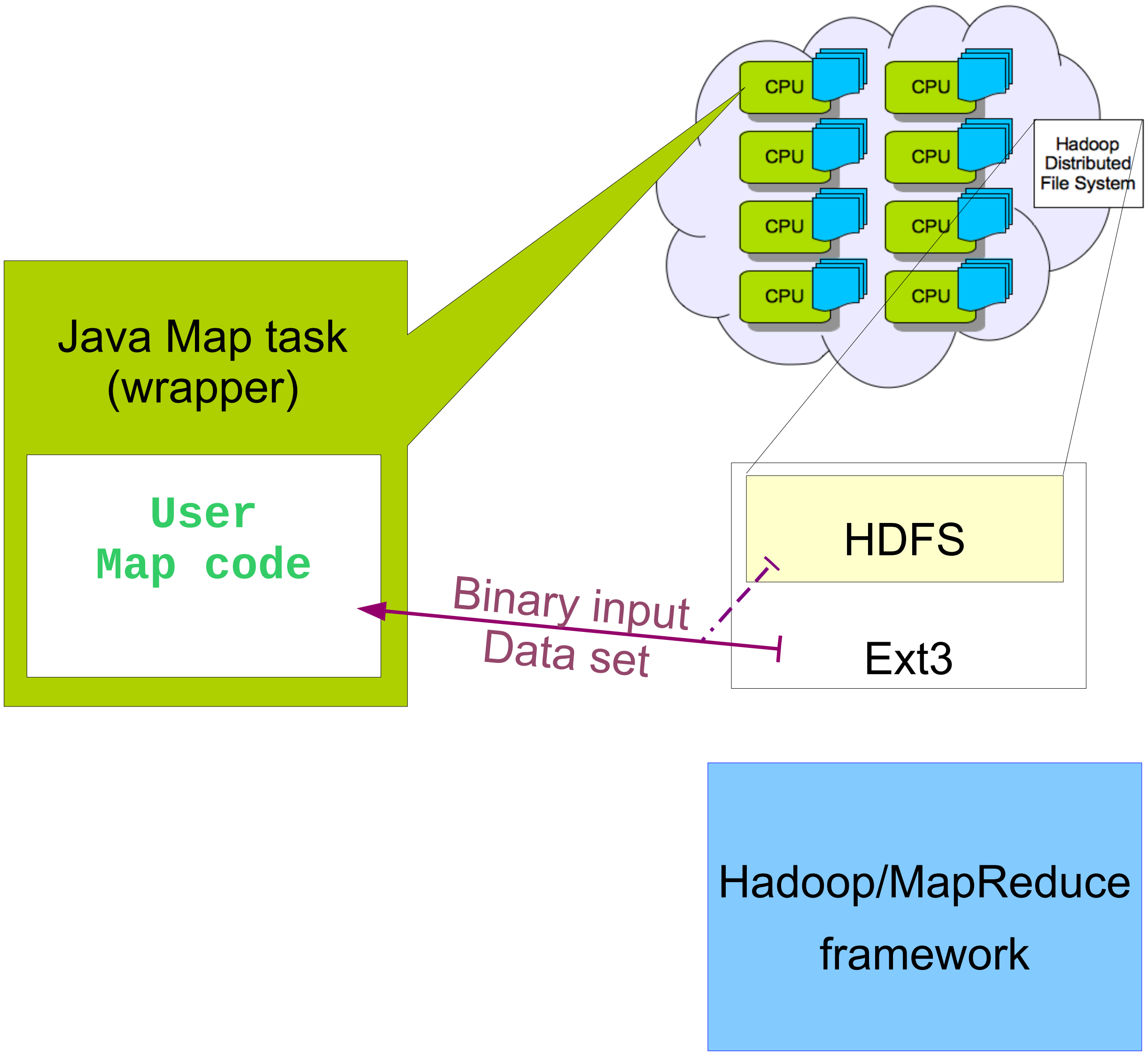
Under the hood..

```
# hadoop run RootOnHadoop "user Map code" "user Reduce code" "HDFS input dataset" "HDFS output location"
```



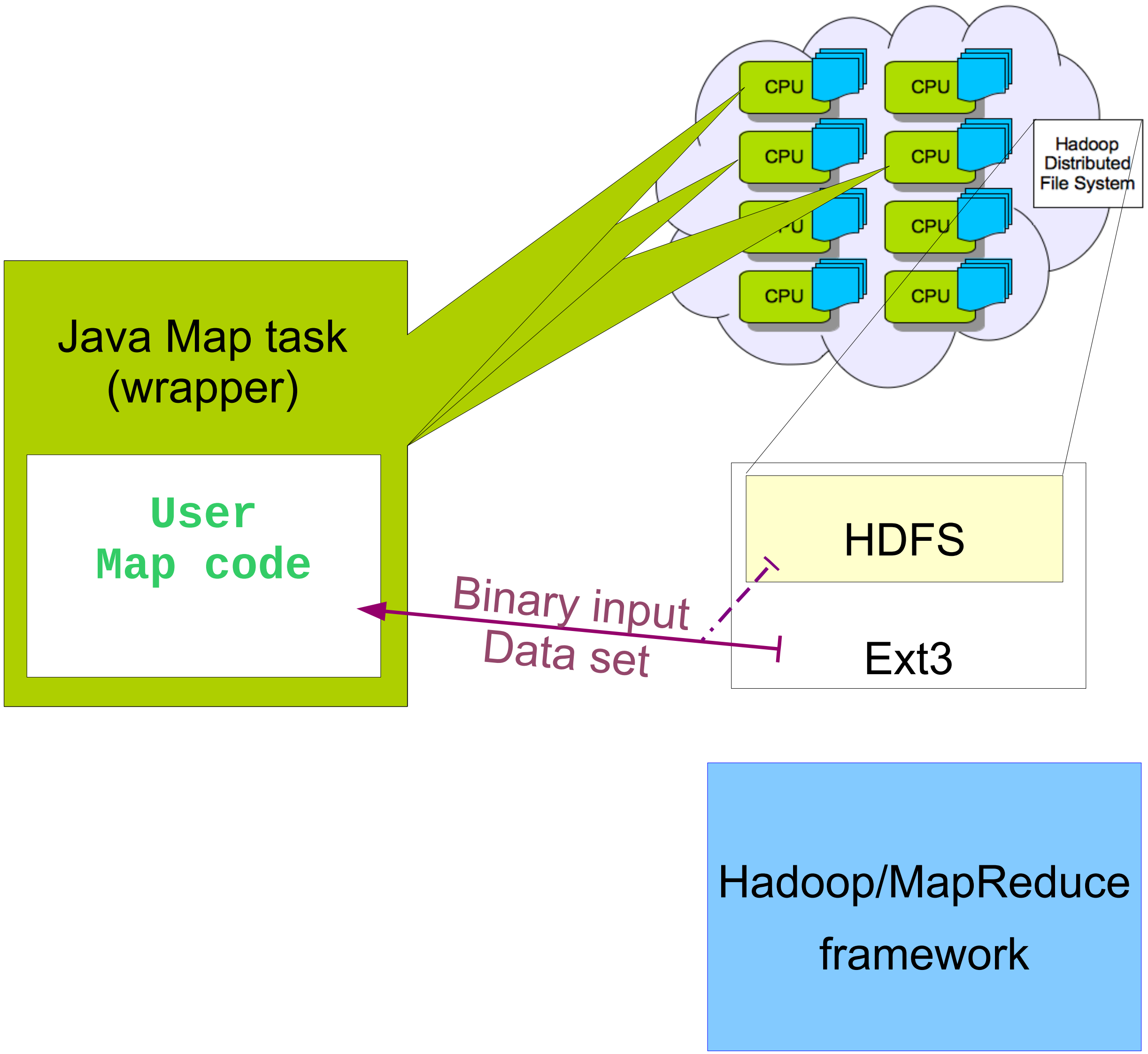
Under the hood..

```
# hadoop run RootOnHadoop "user Map code" "user Reduce code" "HDFS input dataset" "HDFS output location"
```



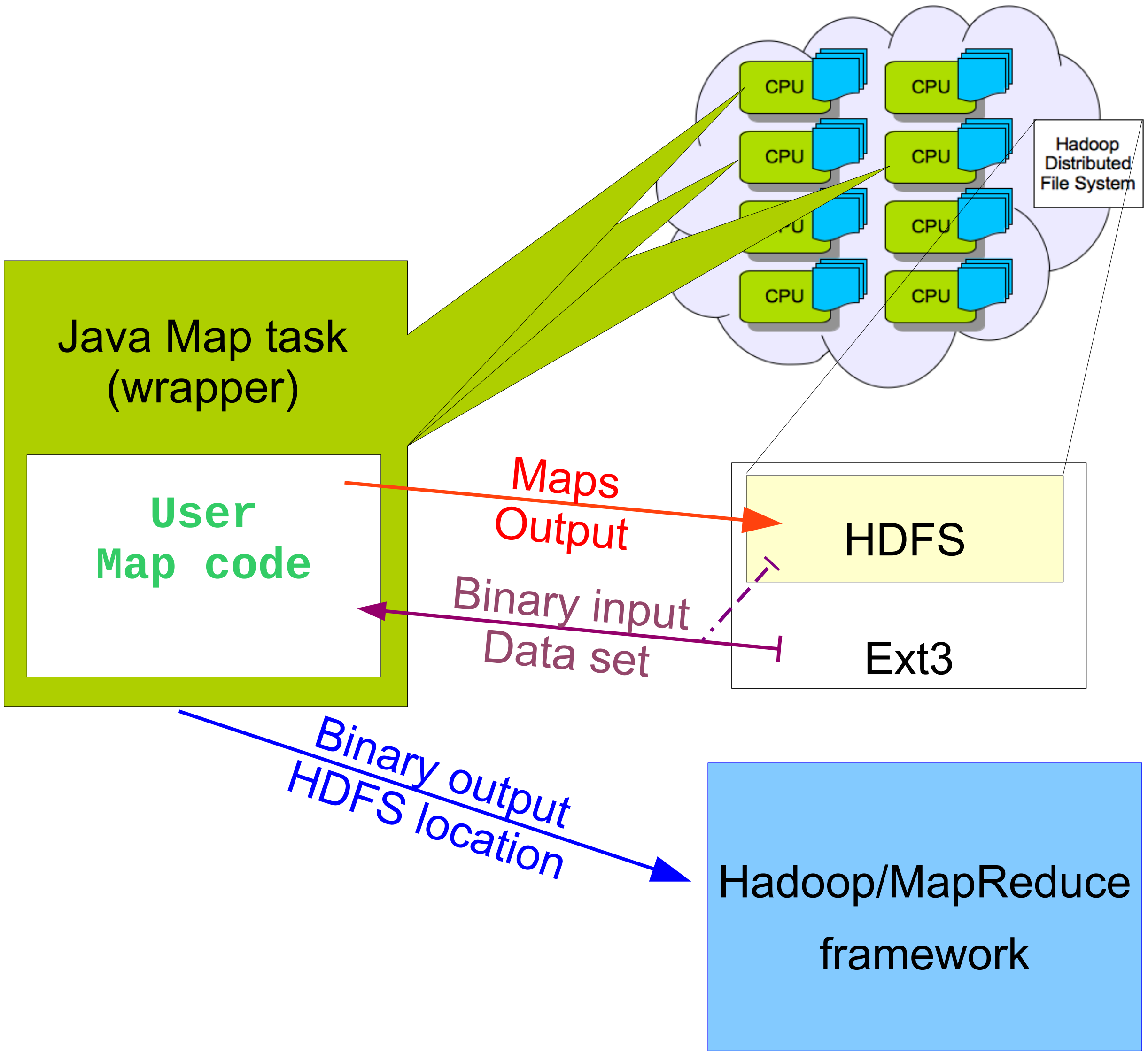
Under the hood..

```
# hadoop run RootOnHadoop “user Map code” “user Reduce
 code” “HDFS input dataset“ “HDFS output location”
```



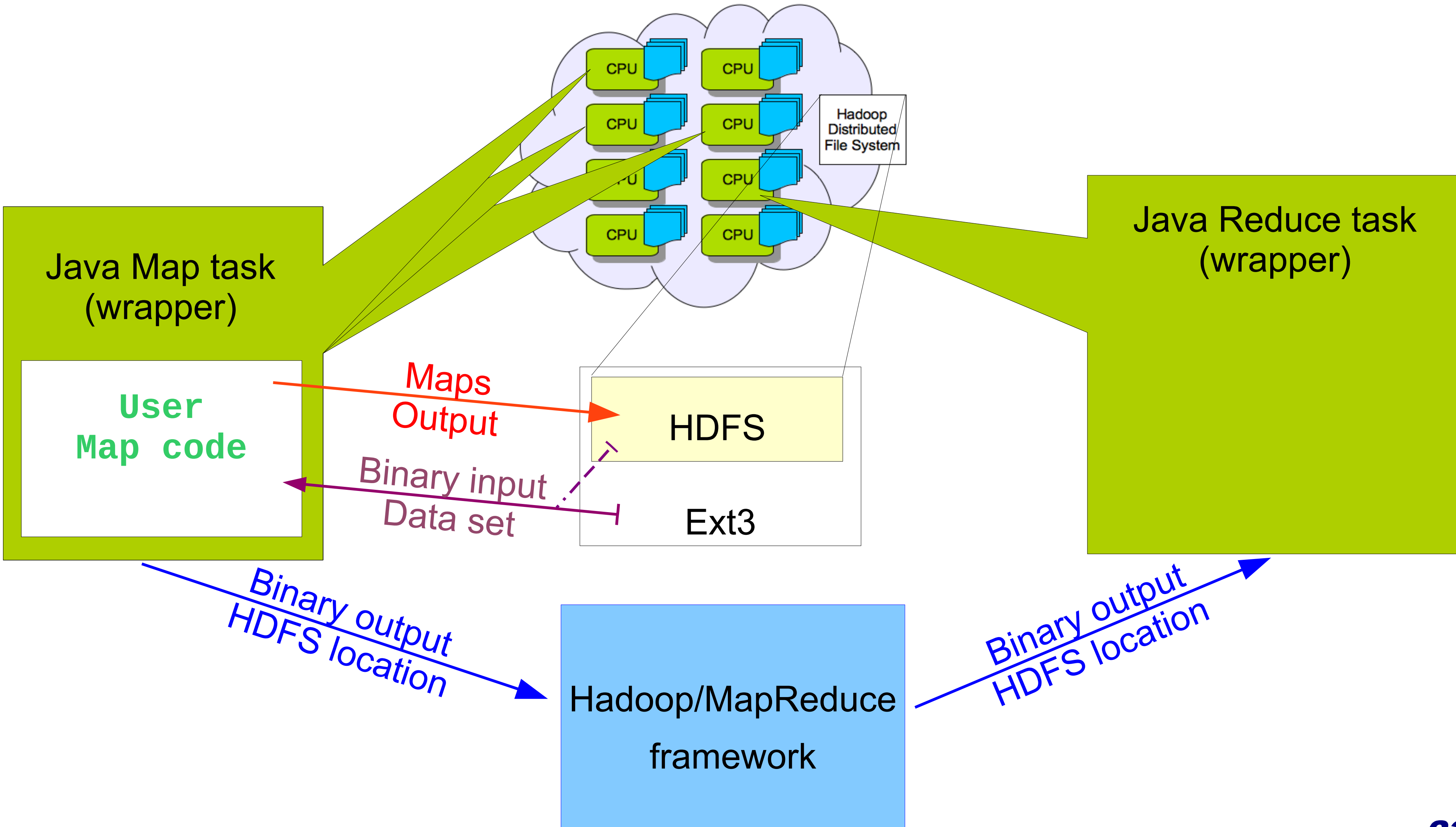
Under the hood..

```
# hadoop run RootOnHadoop "user Map code" "user Reduce code" "HDFS input dataset" "HDFS output location"
```



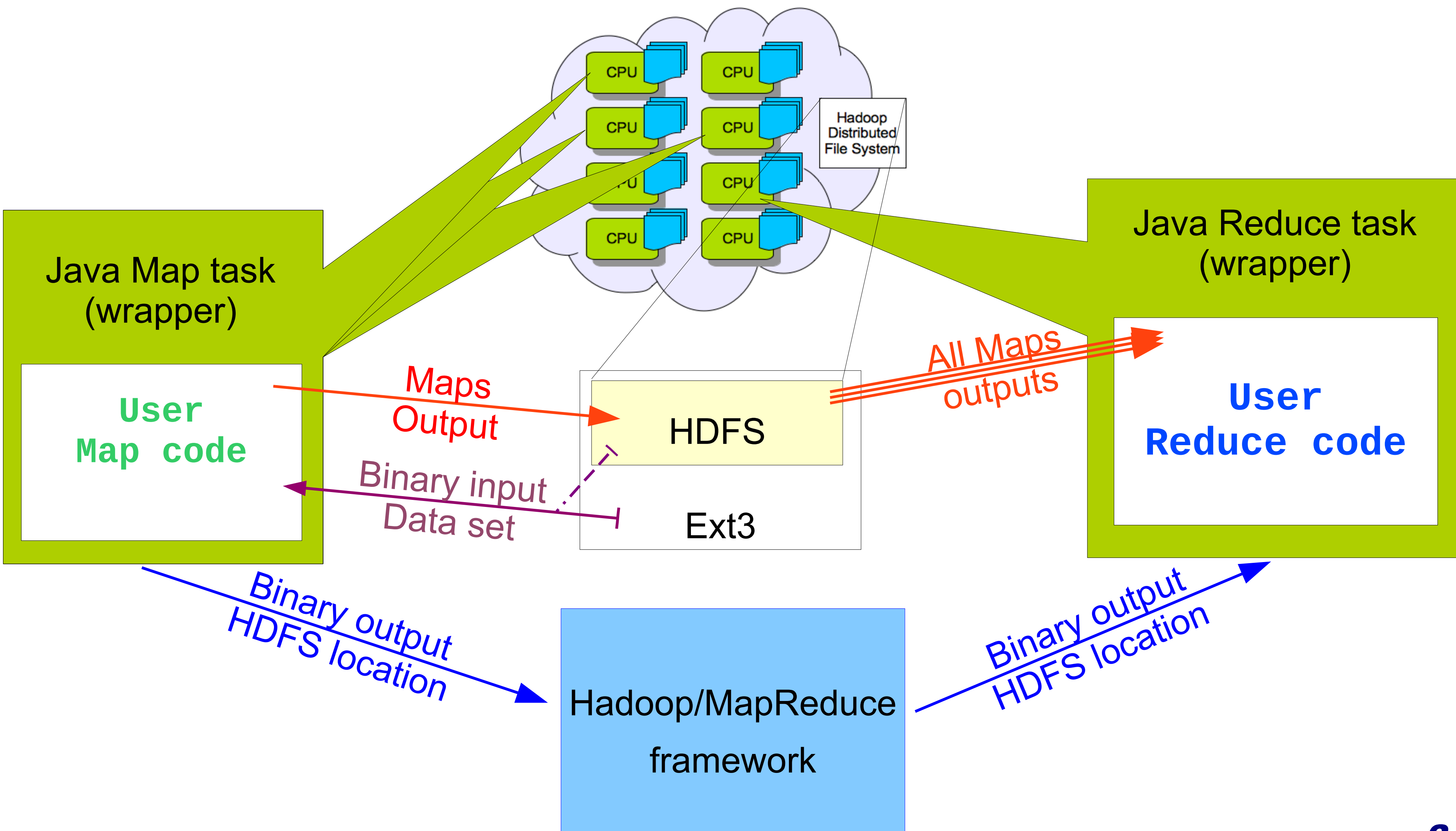
Under the hood..

hadoop run **RootOnHadoop** "user Map code" "user Reduce code" "HDFS input dataset" "HDFS output location"



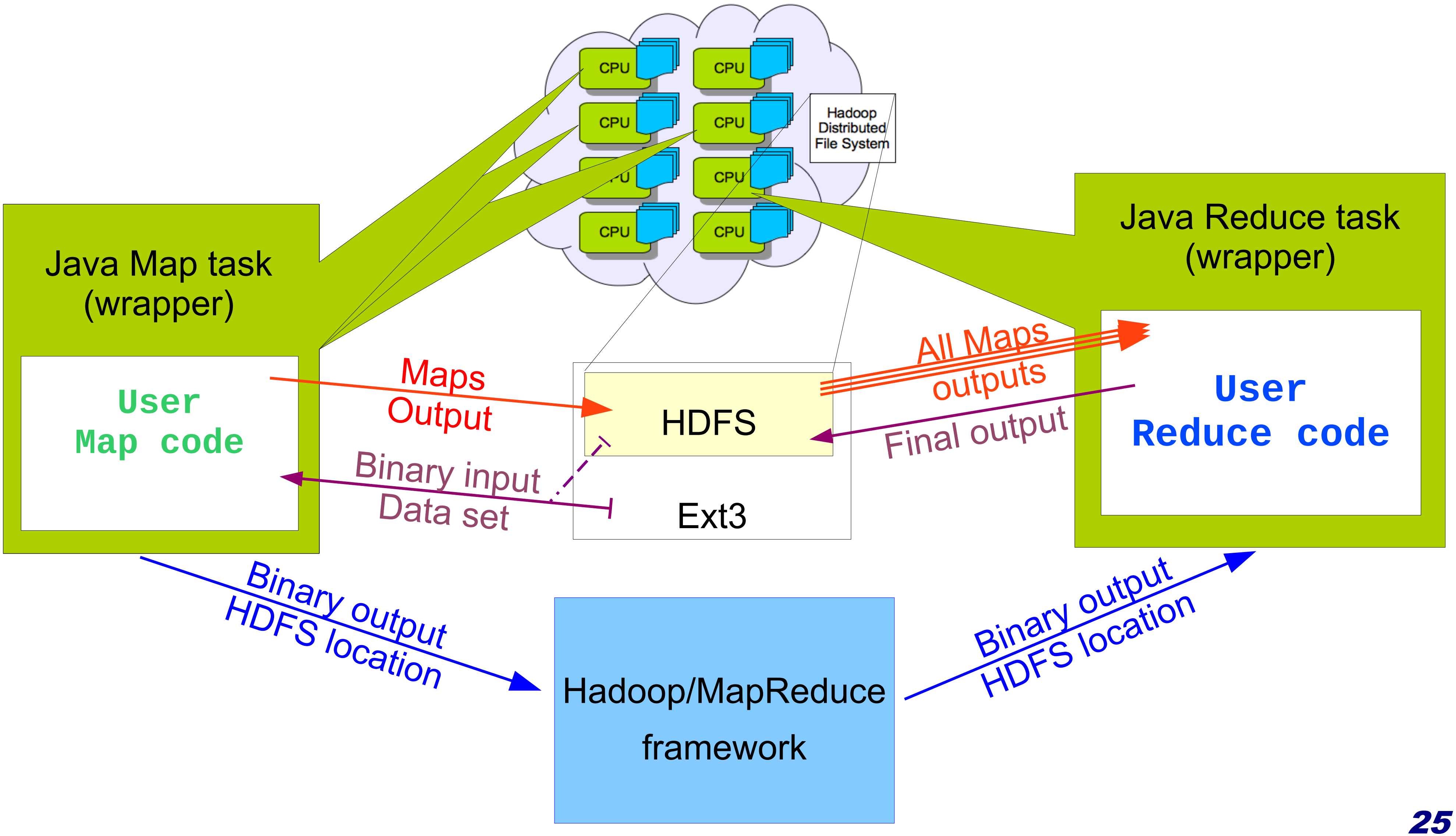
Under the hood..

hadoop run **RootOnHadoop** “**user Map code**” “**user Reduce code**” “**HDFS input dataset**“ “**HDFS output location**”



Under the hood..

```
# hadoop run RootOnHadoop "user Map code" "user Reduce code" "HDFS input dataset" "HDFS output location"
```



A real case: a top quark analysis (1)

ROOT on Hadoop has been tested on a real case: the top quark pair production search and cross section measurement analysis performed by the ATLAS collaboration

Basics of the analysis:

- Based on a cut-and-count code: every event undergoes a series of selection criteria, and at the end is accepted or not.

Map

- Cross section obtained by comparing numbers (number of selected events with the luminosity, the efficiency in the selection of signal events, and the expected background events.)

Reduce

A real case: a top quark analysis (2)

The dataset, data taking conditions:

Data has been taken with all the subsystems of the ATLAS detector in fully operational mode, with the LHC producing proton-proton collisions corresponding to a centre of mass energy of 7 TeV with stable beams condition during the 2011 run up to August.

The dataset, in numbers:

- **338,6 GB** (only electron channel D3PDs)
- **8830 files**
- average size: ~ 38 MB
- maximum file size: ~ **48 MB**

Every file fits in a default HDFS chunk size of 64 MB!

➡ Data copied straightforward from CERN Tier-0 to the Hadoop Cluster

A real case: a top quark analysis (3)

The test cluster:

- Provided by CERN IT-DSS group
- **10 nodes**, 8 cpus per node
- Max 10 Map tasks per node
- **2 replicas per file**

The top quark analysis code:

- ROOT-based, treated as a black magic box
- Compiled without any modification!
- Has ben stored on the Hadoop File System as well

Results (1)

Worked as expected:

Kind	% Complete	Num Tasks	Pending	Running	Complete
map	48.33%	8830	4462	100	4268
reduce	16.07%	1	0	1	0

- **Data locality ratio: 100%** *(every file is read locally)*

  Using the *Delayed Fair Scheduler By Facebook*

  designed for (and tested to) give data locality ratios close to 100% in the majority of the use-cases.

Results (2)

Data locality 100% and data transfers at runtime:

Data transfers:	Hadoop Computing Model	Standard Computing Model
Code	0,12 GB	0,12 GB
Infrastructure overhead	1,17 GB	-
Input data set	0 GB	336,6 GB
Output events count	-	-
Total:	1,29 GB	336,72 GB

- **Performance in terms of time still to be evaluated**

  → **...coparision is hard (apples Vs bananas issue)**

Conclusions – Pros and Cons

- *Typical HEP analyses can be easily ported to a MapReduce model*
- In Hadoop *network usage* for accessing the data *reduced by several orders of magnitude* thanks to the data locality feature
- *Transparency* can be achieved quite easily
- Bypassing some Hadoop components permits to:
  - run standard code on standard, local file systems at maximum speed
  - fine tuning (SSD caching, BLAS/LAPACK..)

  ..while:

  exploiting the innovative features of Hadoop/MapReduce and HDFS

- *easy to manage, fault tollerant and scalable infrastructure (plug/unplug)*
- *open source,* widely used and well maintained

...and the method actually works, **positive feedback received**

i.e. Uni LMU ATLAS group, poster here at CHEP 2013

"Evaluation of Apache Hadoop for parallel data analysis with ROOT"

Conclusions – Pros and Cons

- Java and ROOT overhead to start many jobs

  Performance to be evaluated

  Tuning: - JVM reuse, Map startup improvement;
  - Latency (Heartbeat) optimization...

- Bottomline: Hadoop forced to work unnaturally

  bugs when working with blocksize > 2 Gb to be fixed

  (already investigated by the community)

...worth to investigate, spend time for tuning, find a metric to measure performance?

Conclusions – Pros and Cons

- *Typical HEP analyses can be easily ported to a MapReduce model*
- *Network usage* for accessing the data *reduced by several orders of magnitude* thanks to
- Hadoop's data locality feature. Same data accessed over and over.
- *Transparency* can be achieved quite easily
- Bypassing some Hadoop components permits to:
  - run standard code on standard, local file systems at maximum speed
  - fine tuning (SSD caching, BLAS/LAPACK..)

  ..while:

  exploiting the innovative features of Hadoop/MapReduce and HDFS
- *easy to manage, fault tollerant and scalable infrastructure*
- ..and is *open source,* widely used and well maintained

- Hadoop and ROOT overhead to start many jobs (Performance to be evaluated)
- Hadoop forced to work unnaturally

  bugs when working with blocksize > 2 Gb to be fixed (already investigated)

Thanks for your attention!

Demo code → stefano.alberto.russo@cern.ch

...questions?