Hadoop as a solution for data-intensive scientific computing

Stefano Alberto Russo

CERN IT Department

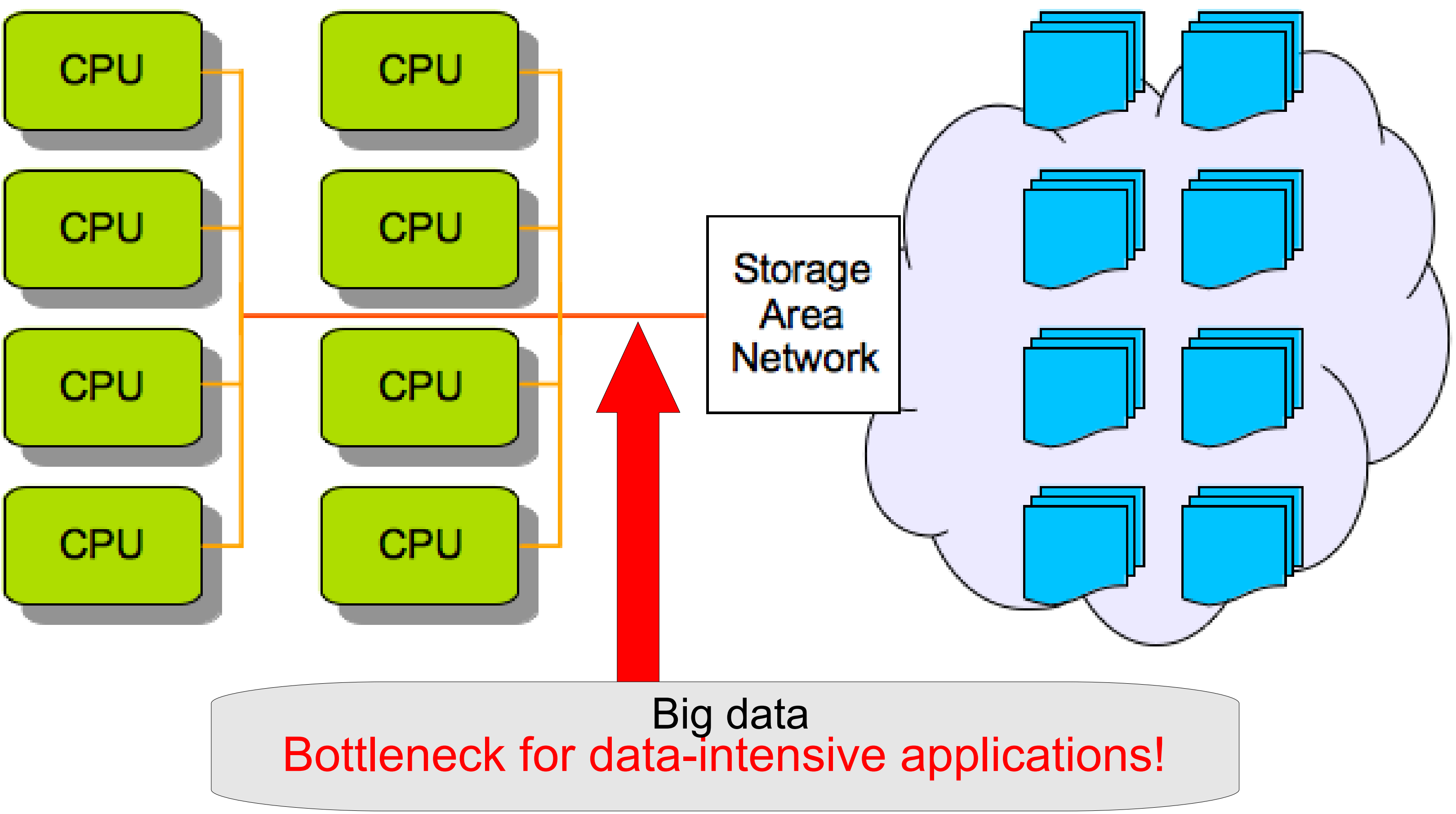
“Data-Day” – ICTP – Trieste – 05/09/2013

Topics

- What is Hadoop/MapReduce?
- Scientific codes and Hadoop - limitations
- Scientific codes and Hadoop - solutions
- A real case: high energy physics analysis
- Conclusions

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

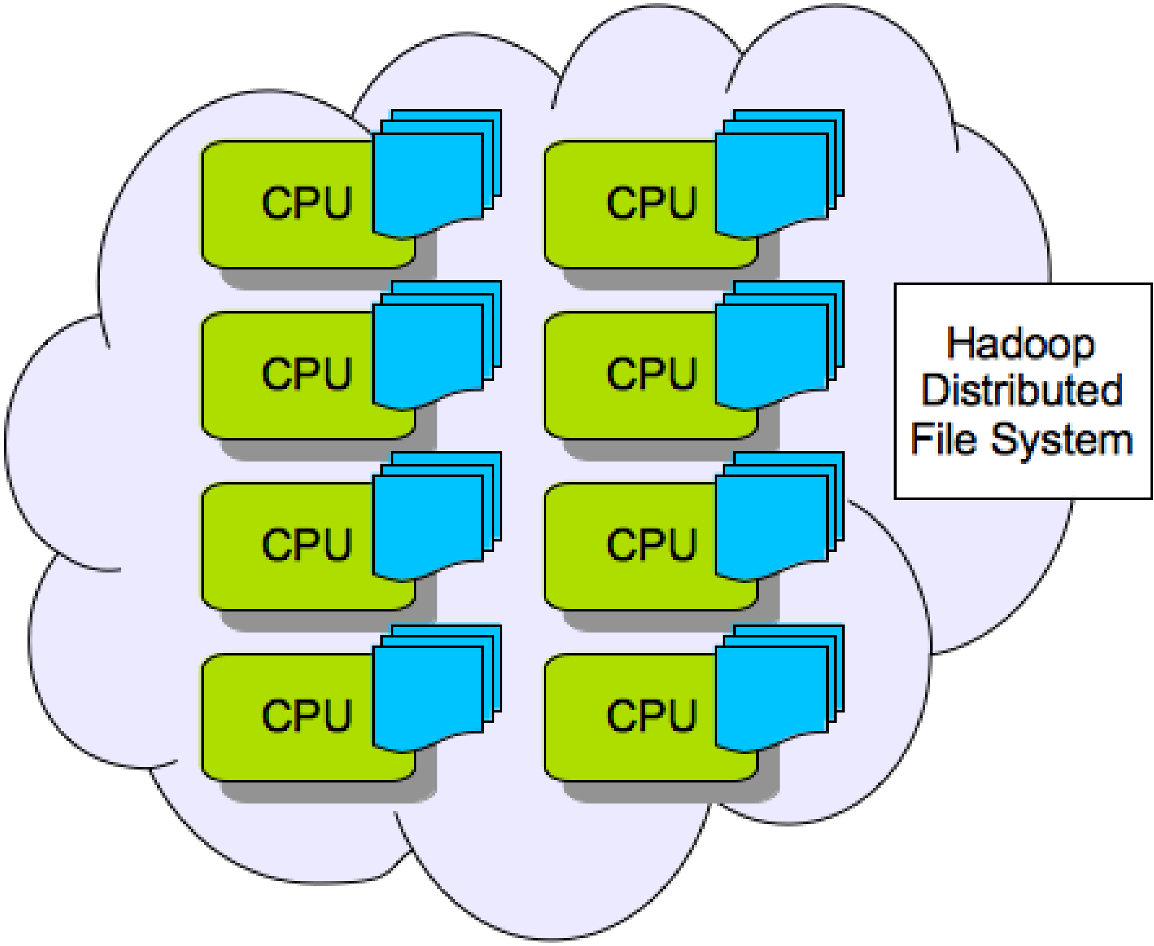
➡ the computation can be scheduled on the cluster elements holding a copy of the data to analyze: *data locality*

Two components:

1. The Hadoop Distributed File System (HDFS)

2. The MapReduce computational model and framework

- Open Source
- Widely used (Facebook, Yahoo..)



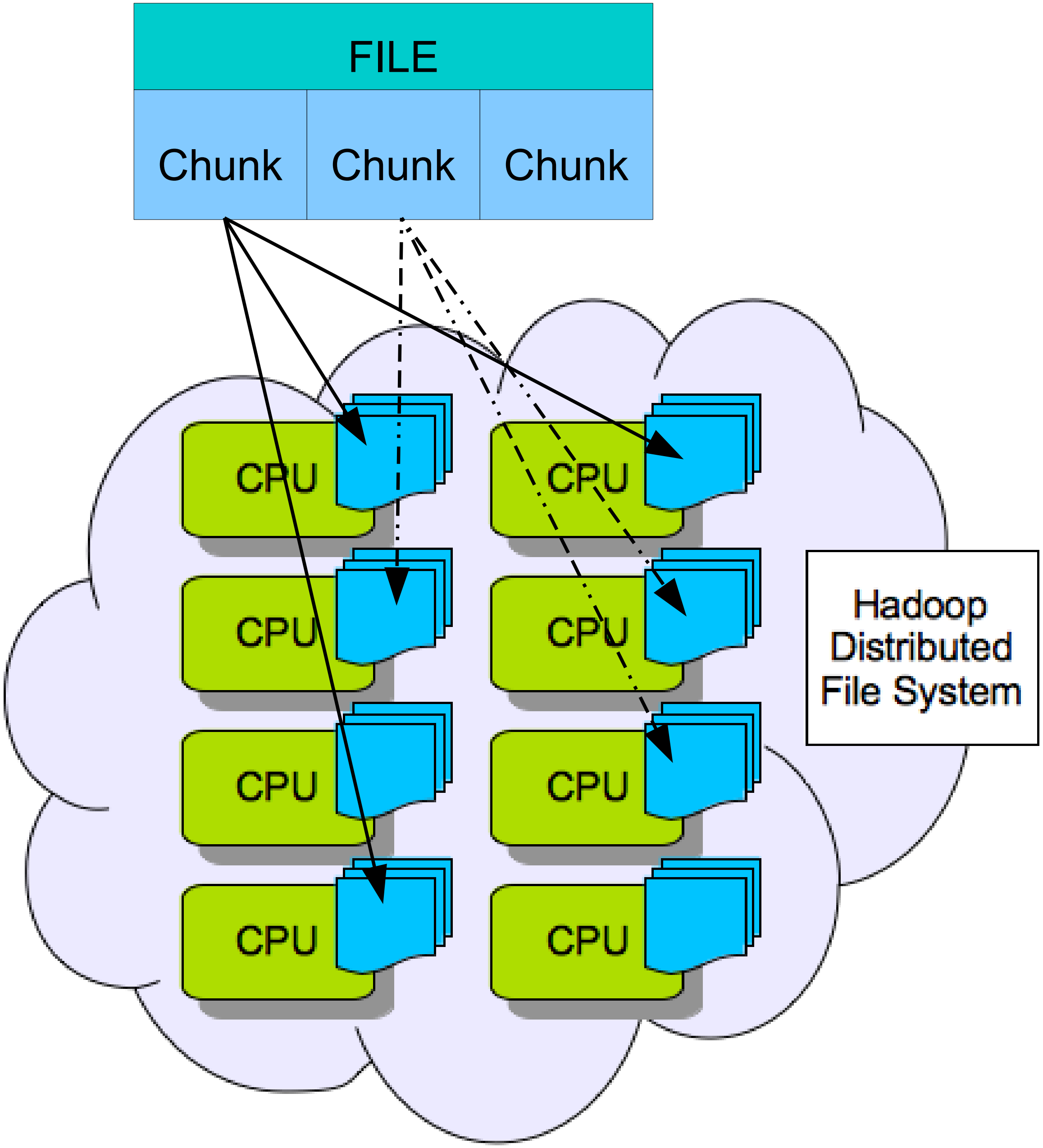
The Hadoop Distributed File System (HDFS)

On HDFS, files are:

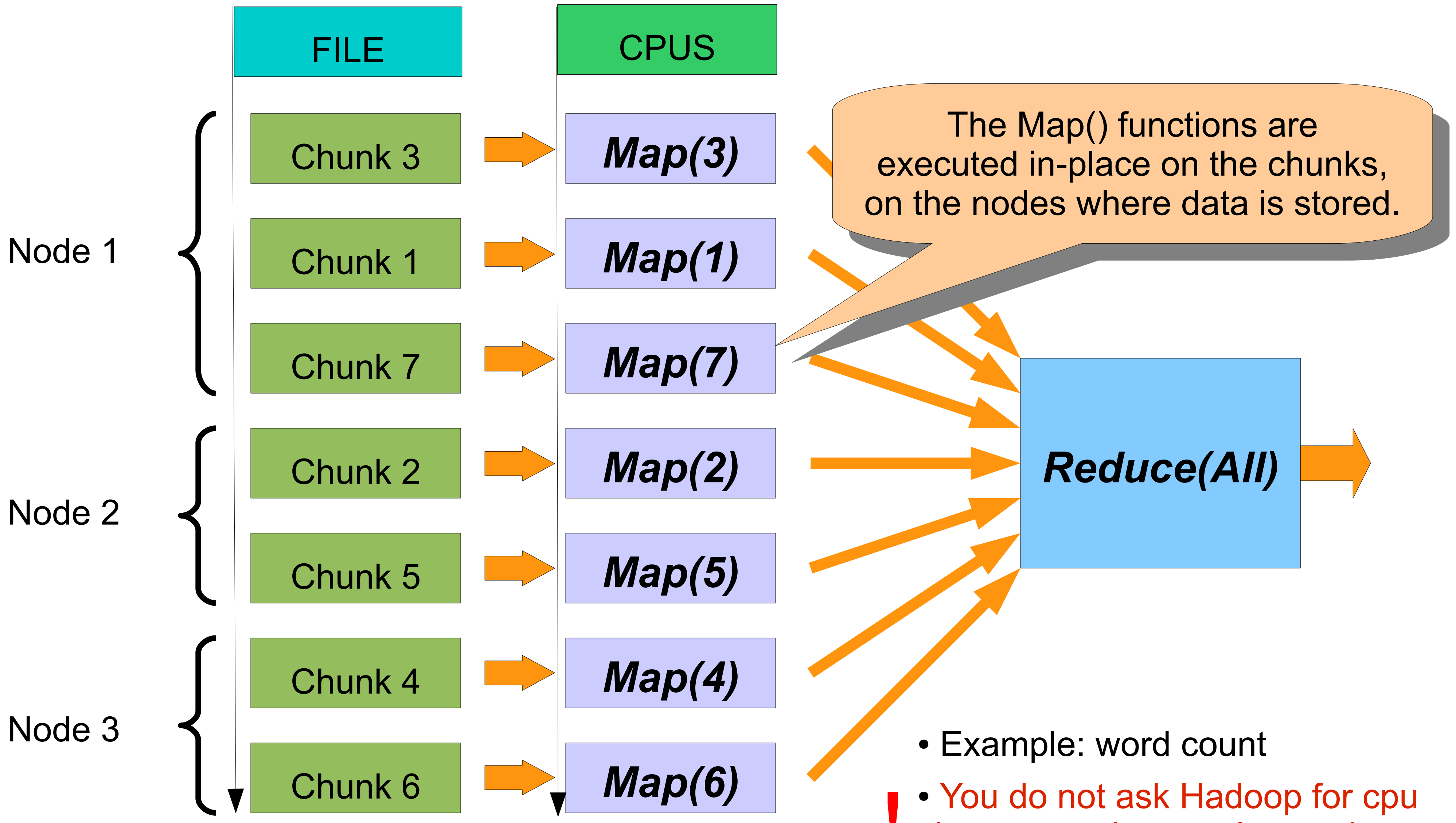
- Stored by slicing them in chunks (i.e. 64 MB)

- ..which are placed across the Hadoop cluster in a configured number of replicas (usually 3) for data redundancy and workload distribution.

- No RAID
- Commodity hardware
  (Low cost disks)



The MapReduce model and framework



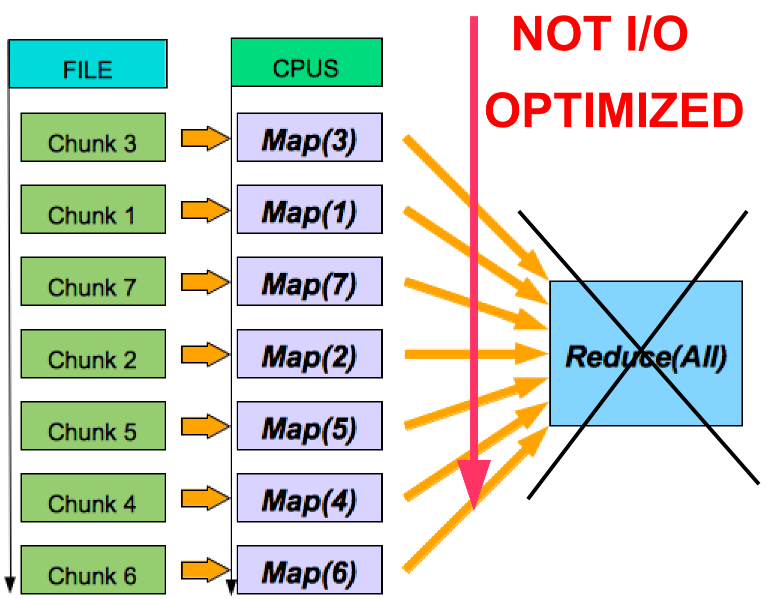
- Example: word count
- ! You do not ask Hadoop for cpu slots, you ask to analyze a dataset
- Fault tolerance

The MapReduce model and framework

MapReduce requires an *embarrassing parallel* problem.

... a problem which can be split in
independent subproblems

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



data locality can be exploited also for codes which produce huge amounts of data like ***preprocessing*** (first replica on the node), **but** this data should not be processed by a Reduce

- *No problems to run without a Reduce*

The MapReduce model and framework

- 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)*

Hadoop streaming: allows to run a custom code which reads data from stdin, and which returns data from stdout.

- *Dataset has to be in plain text!*

Scientific codes on Hadoop

Scientific codes:

- In general in Fortran, C, C++: not Java
- Often developed for years to model complex scientific processes, possibly by a joint effort of a community

porting on Java is not an option

+

non-Java code on Hadoop only on textual datasets (via Streaming)

=

scientific codes in Fortran, C, C++ etc. which have to operate on complex (binary) data sets, just cannot be executed on Hadoop/MapReduce with the current implementation.

Scientific codes on Hadoop (2)

“Scientific code” definition onwards: *a code which cannot be ported to Java and that has to operate on a binary dataset.*

..and we restrict to the class of embarrassing parallel problems.

How to run them on Hadoop?

What to ensure when looking for a solution?

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.

Mission: transparency (1)

Transparency for the (binary) data:

- Binary data cannot be read in chunks (corruption)
- **One Map = One file** vanishes data locality
- **One Map = One file = one HDFS block is fine**

(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.

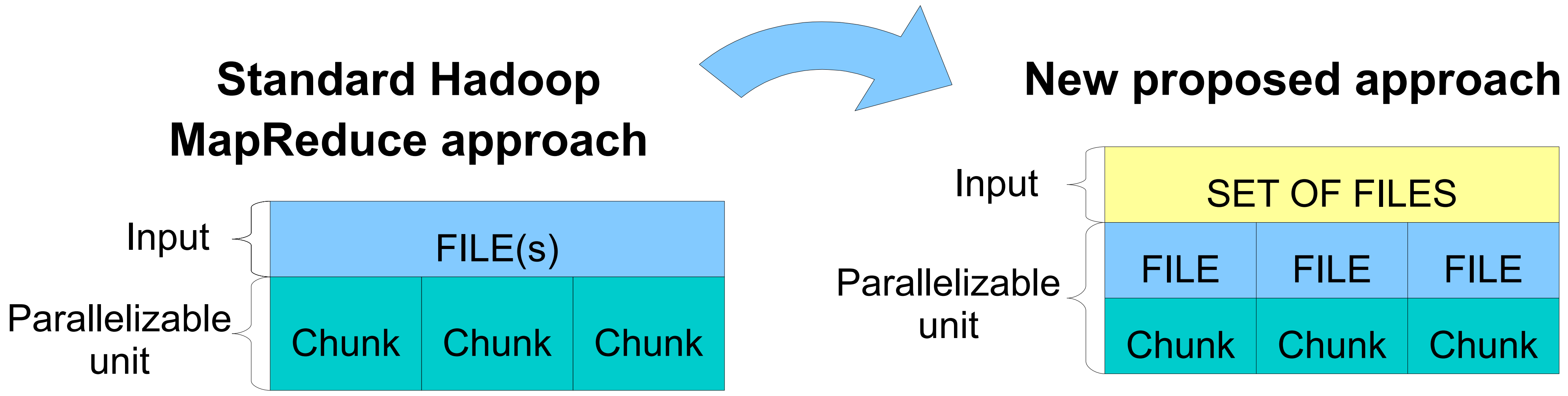
Other approaches are possible, but much more effort required

Mission: transparency (1.1)

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!

Mission transparency (2)

Transparency for the code:

Bottom line: bypass Hadoop

1. Hadoop's Java Map and Reduce tasks as wrappers for the <u>real code</u>

2. Let the <u>real code</u> 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

Mission: transparency (3)

Transparency for the user:

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

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

Reminder:

on Hadoop you do not ask for cpus, you ask to analyze a dataset.

Mission: transparency (3)

Transparency for the user:

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

Few guidelines:

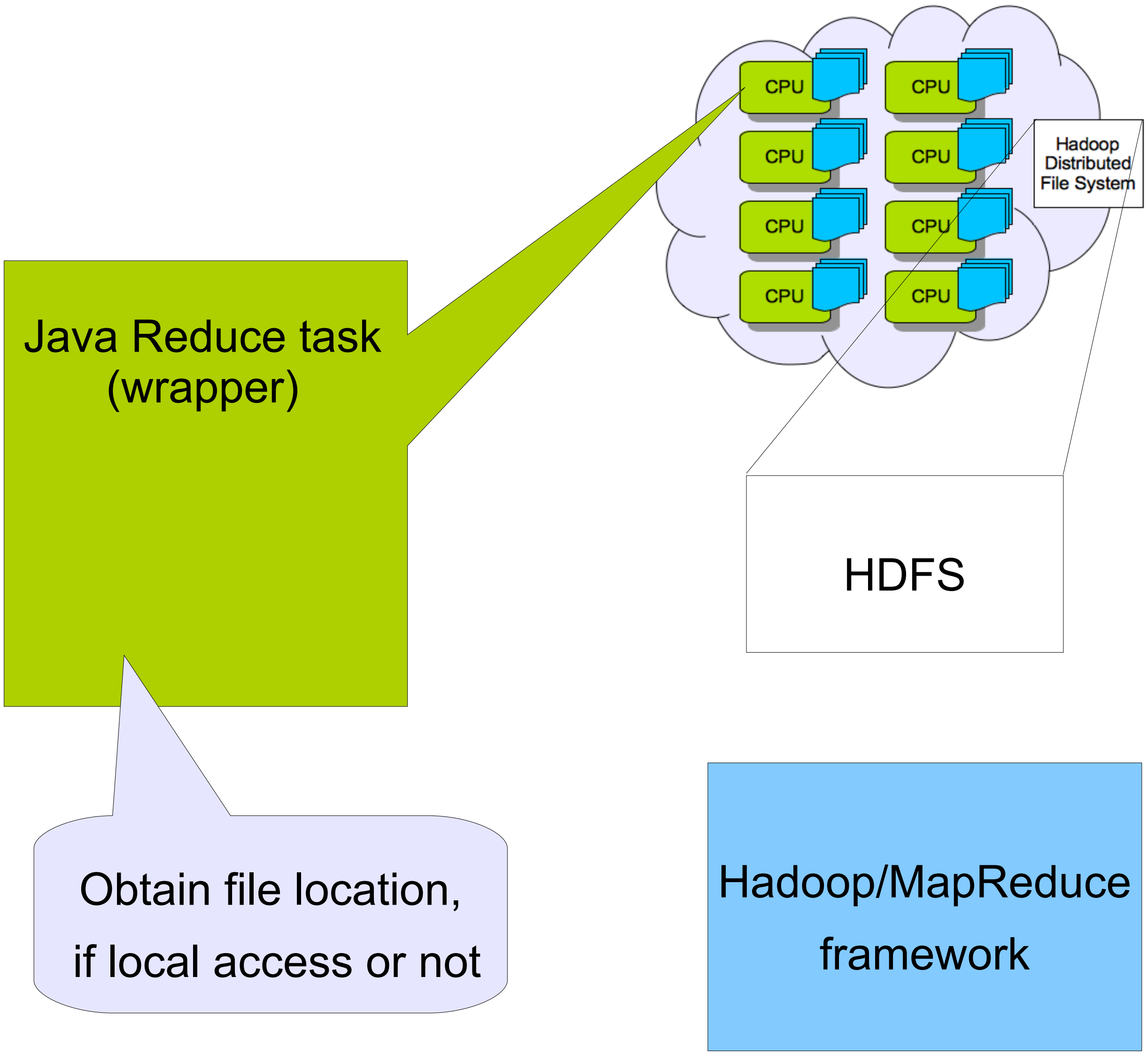
- **User Map** will receive as the first argument the file on which to operate on
- **User Map** output has to follow a conventional naming schema

  to be accessed from the Reduce

- **User Reduce** will receive from the standard input (one per line) the locations on HDFS of the files to merge in the final result.

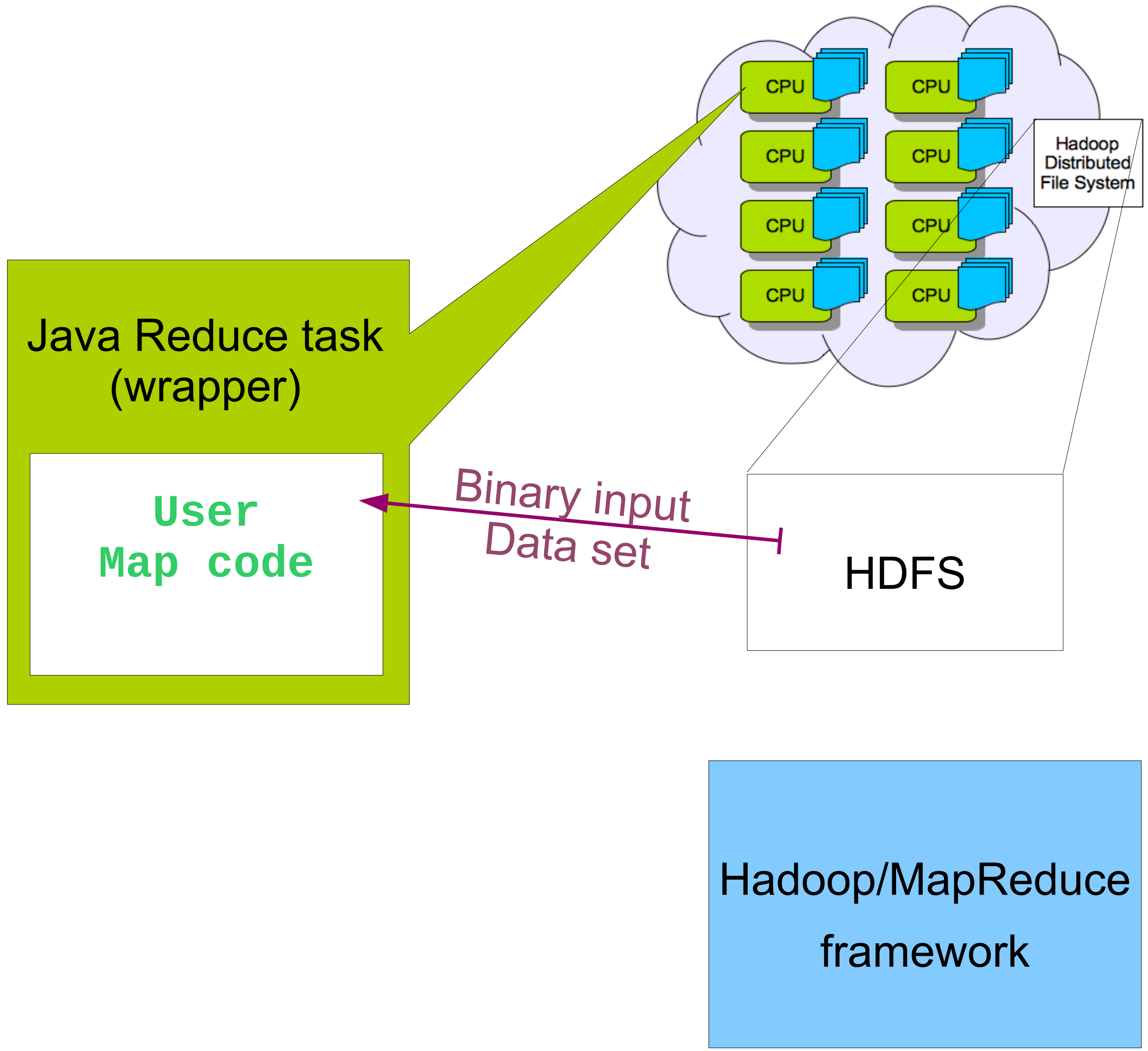
Under the hood..

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



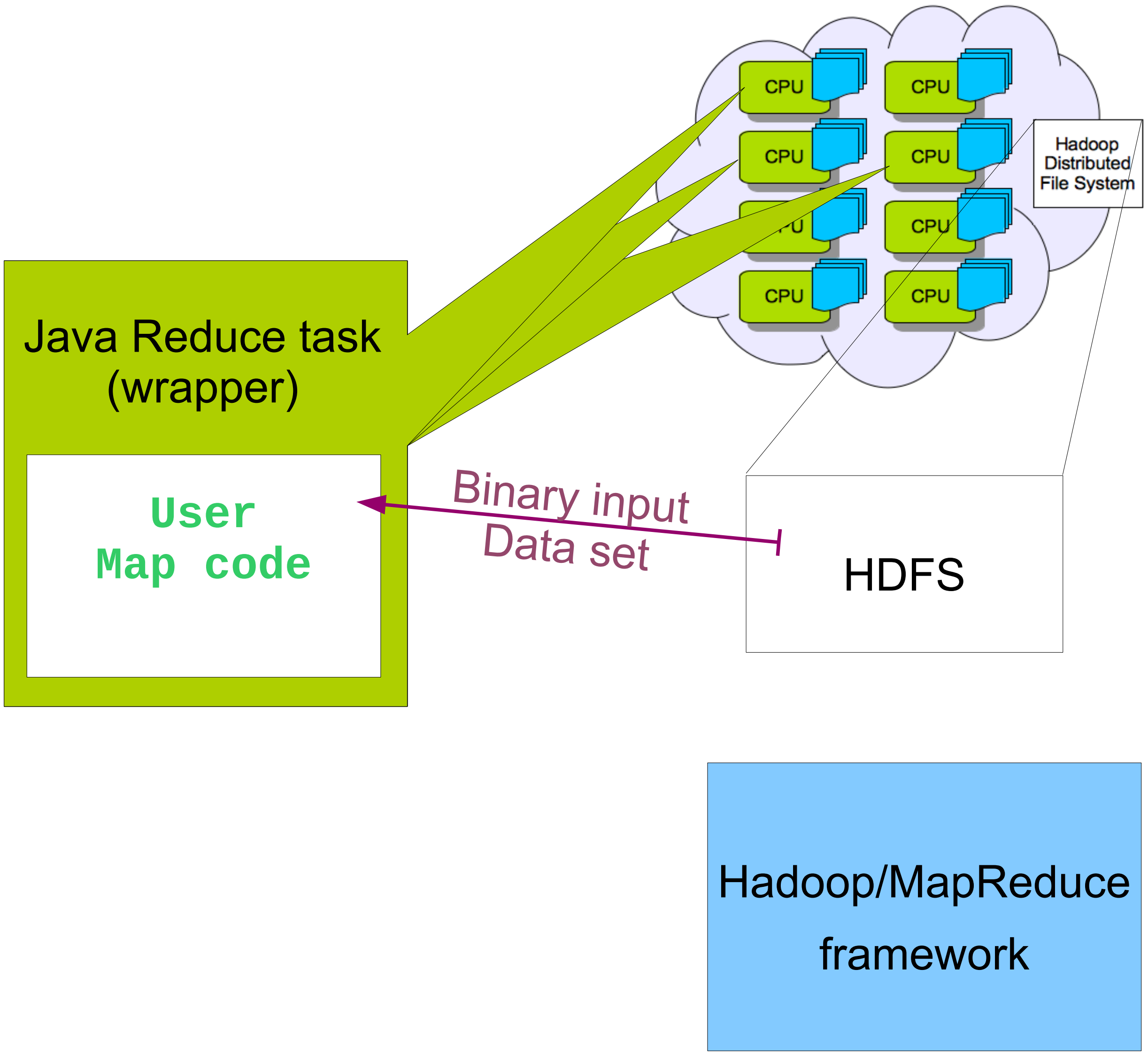
Under the hood..

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



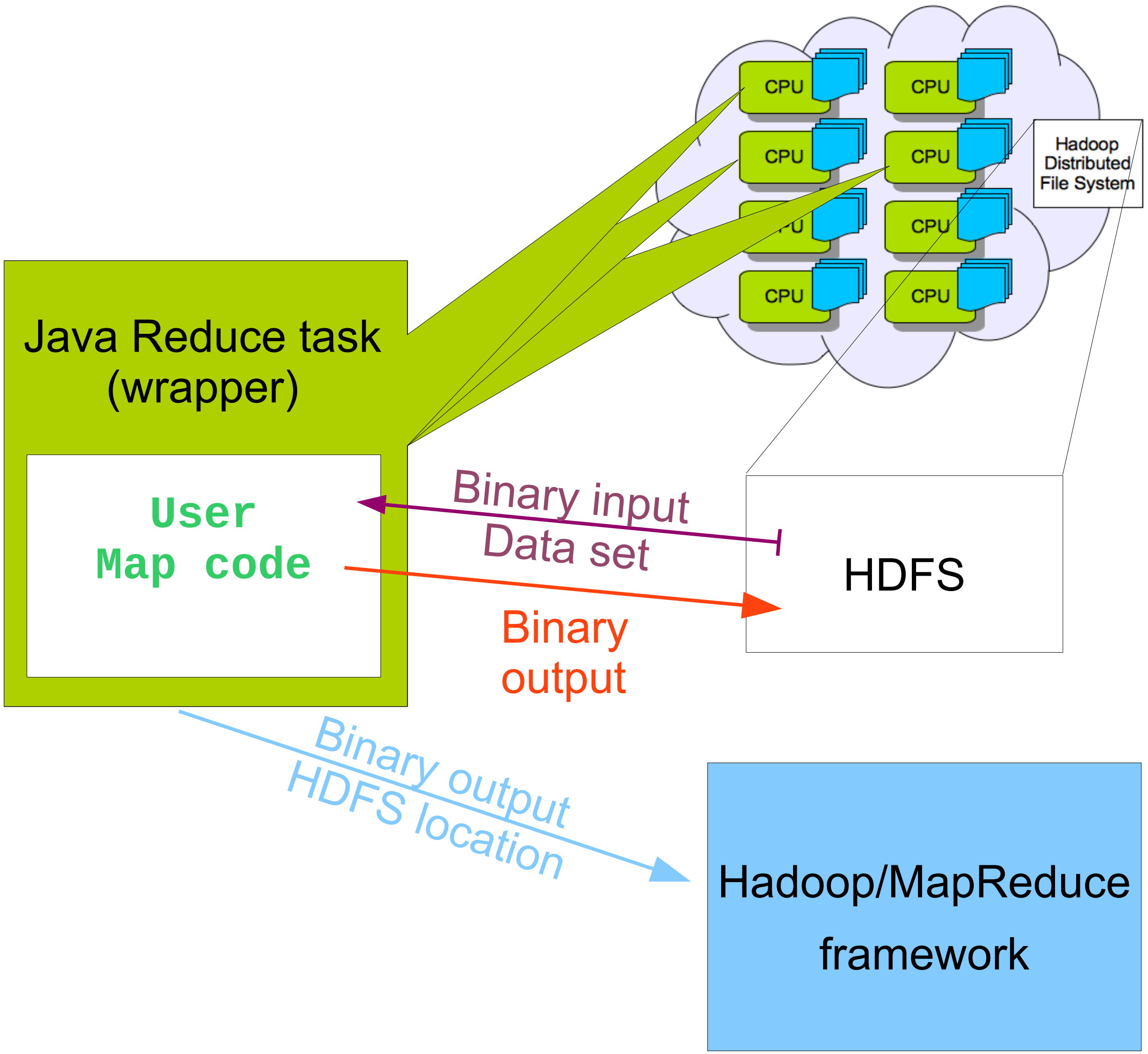
Under the hood..

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



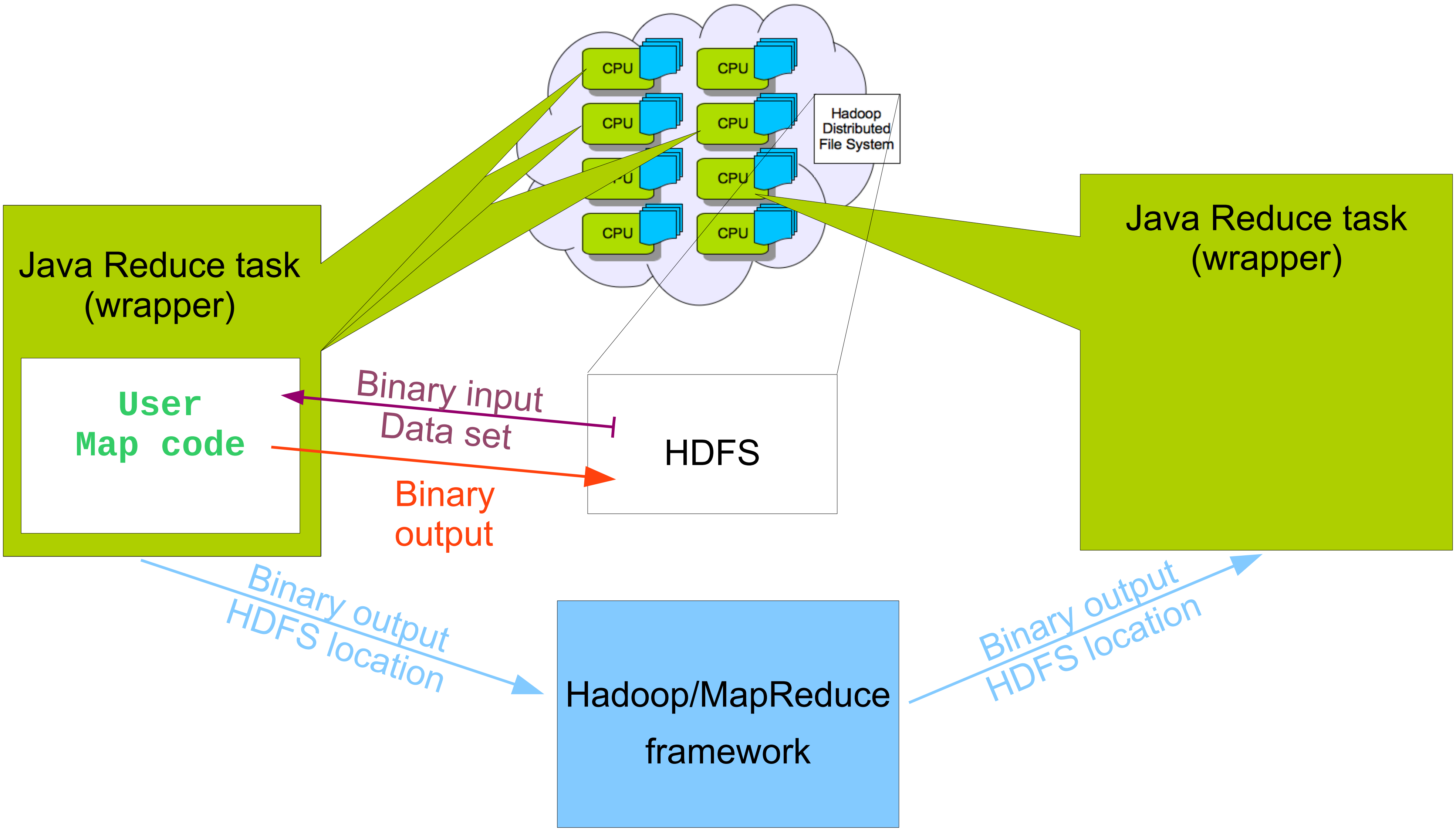
Under the hood..

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



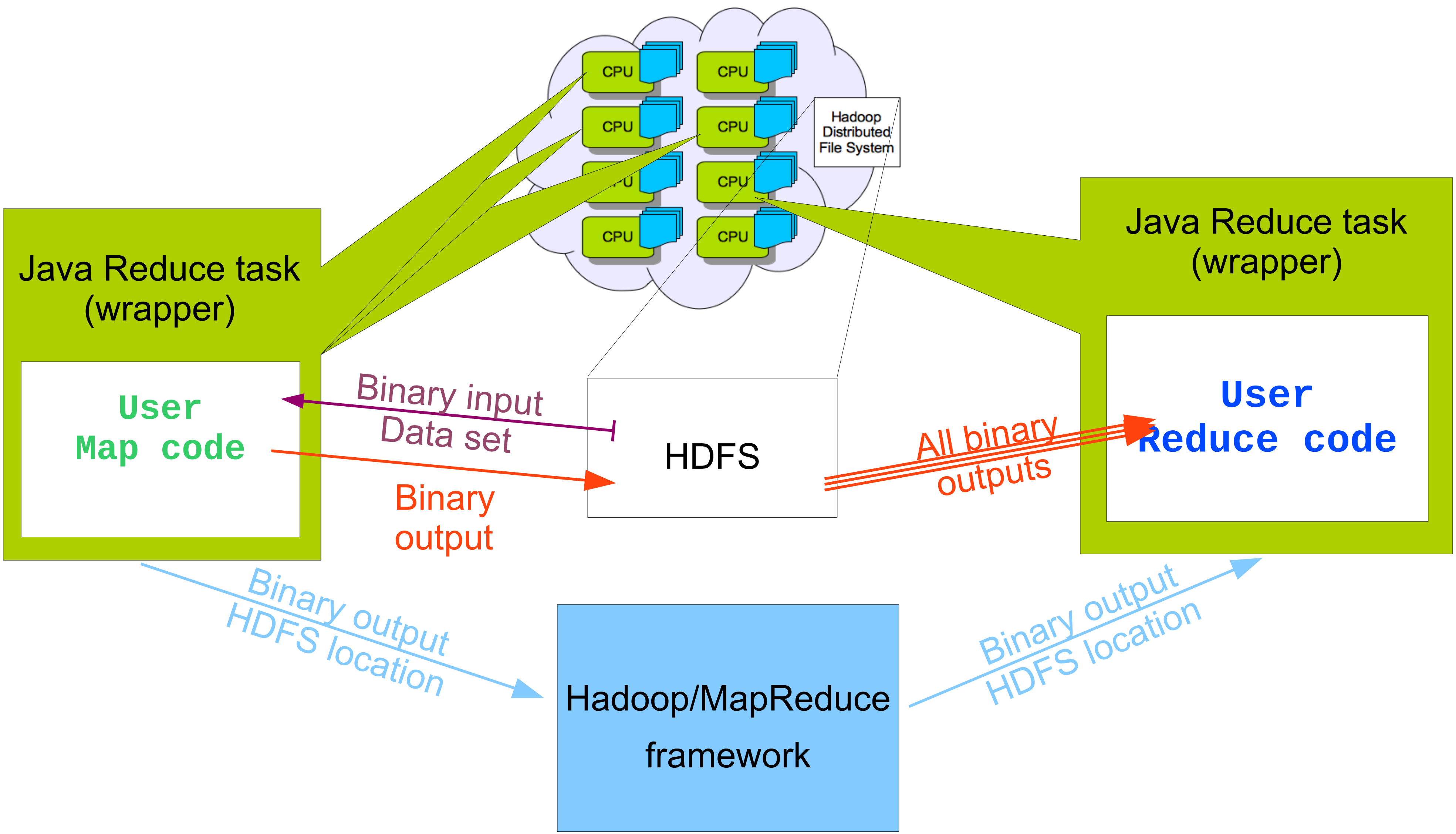
Under the hood..

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



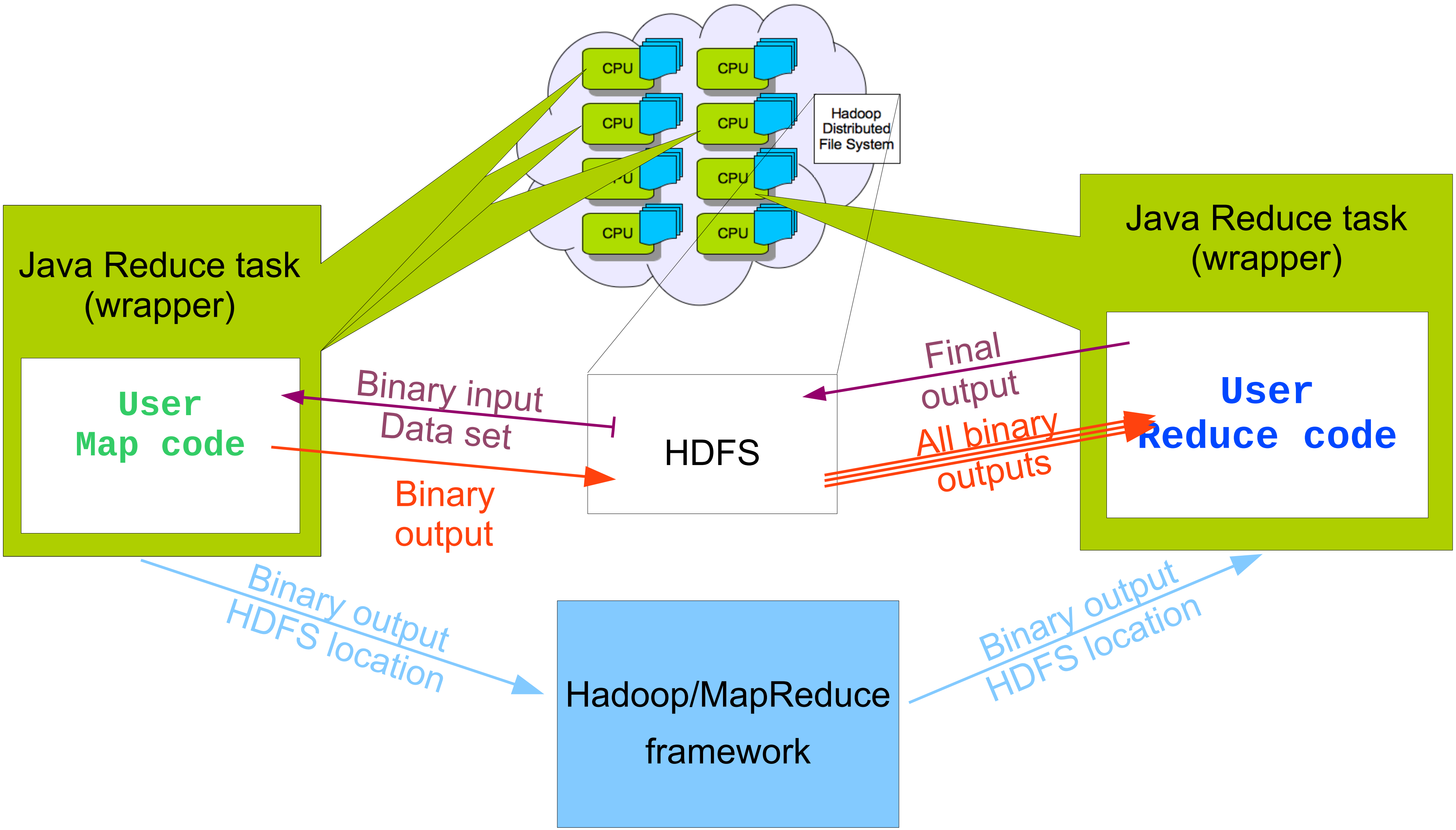
Under the hood..

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



Under the hood..

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



A real case: a top quark analysis (1)

The approach has been tested on a real case: the top quark pair production search and cross section measurement analysis performed by the ATLAS Udine Group on LHC data

PARTICLE COLLISIONS EVENTS ARE INDEPENDENT

Basics of the analysis:

- *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 the number of selected events with the total. (**Reduce**)

+ luminosity, efficiency in selection of signal events, expected background events.

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** (considering only data related to this analysis)
- **8830 files**
- average size: ~ 38 MB
- maximum file size: ~ **48 MB**

LHC produces 15 Petabytes/year!

Every file fits in the default Hadoop chunk size of 64 MB!

→ Data copied straightforward from 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
- Other details are not relevant

Preparing the top quark analysis code:

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

Results (1)

AGAIN: transparency

- **For the data:** Data has been stored on the Hadoop cluster without conversions, in its original format.

- **For the code:** An arbitrary executable (ROOT) has been run without any modification

- **For the user:** User's Map and Reduce code had to follow just few guidelines, but then:

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

Results (2)

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%,**

  Using the Delayed Fair Scheduler By Facebook, which has been designed for (and tested to) give data locality ratios close to 100% in the majority of the use-cases.

Results (3)

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

Conclusions – Pros and Cons

- *Network usage* for accessing the data *reduced by several orders of magnitude* thanks to Hadoop's 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
- Hadoop provides an *easy to manage, robust and scalable infrastructure*
- Project *open source* widely used and well maintained

Conclusions – Pros and Cons

- Only embarrassing parallel problems
  (MPI etc to be investigated)
- Hadoop forced to work unnaturally
  bugs when working with blocksize > 2 Gb to be fixed
  (already investigated by the community)

...worth to investigate!

...positive feedback received (i.e. Uni Muenchen)

My take: *with Hadoop you have a distributed file system which is interesting from various points of view*

..and you can spot data locality for embarrassing parallel problems

Conclusions – Pros and Cons

- *Network usage* for accessing the data *reduced by several orders of magnitude* thanks to
- Hadoop's 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
- Hadoop provides an *easy to manage, robust and scalable infrastructure*
- Project *open source* widely used and well maintained

- Only embarrassing parallel problems (MPI etc to be investigated)
- Hadoop forced to work unnaturally

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

Thanks for your attention!

stefano.alberto.russo@cern.ch

...questions?