VELaSSCo overview : Big Data for Computational Engineering

Miguel Pasenau

http://www.velassco.eu

velassco@cimne.upc.edu

Goal of VELaSSCo

Visual Analysis

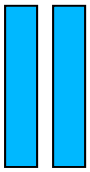
for **E**xtremely **La**rge-**S**cale

Scientific **Co**mputing

The Research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013)

Goal of VELaSSCo

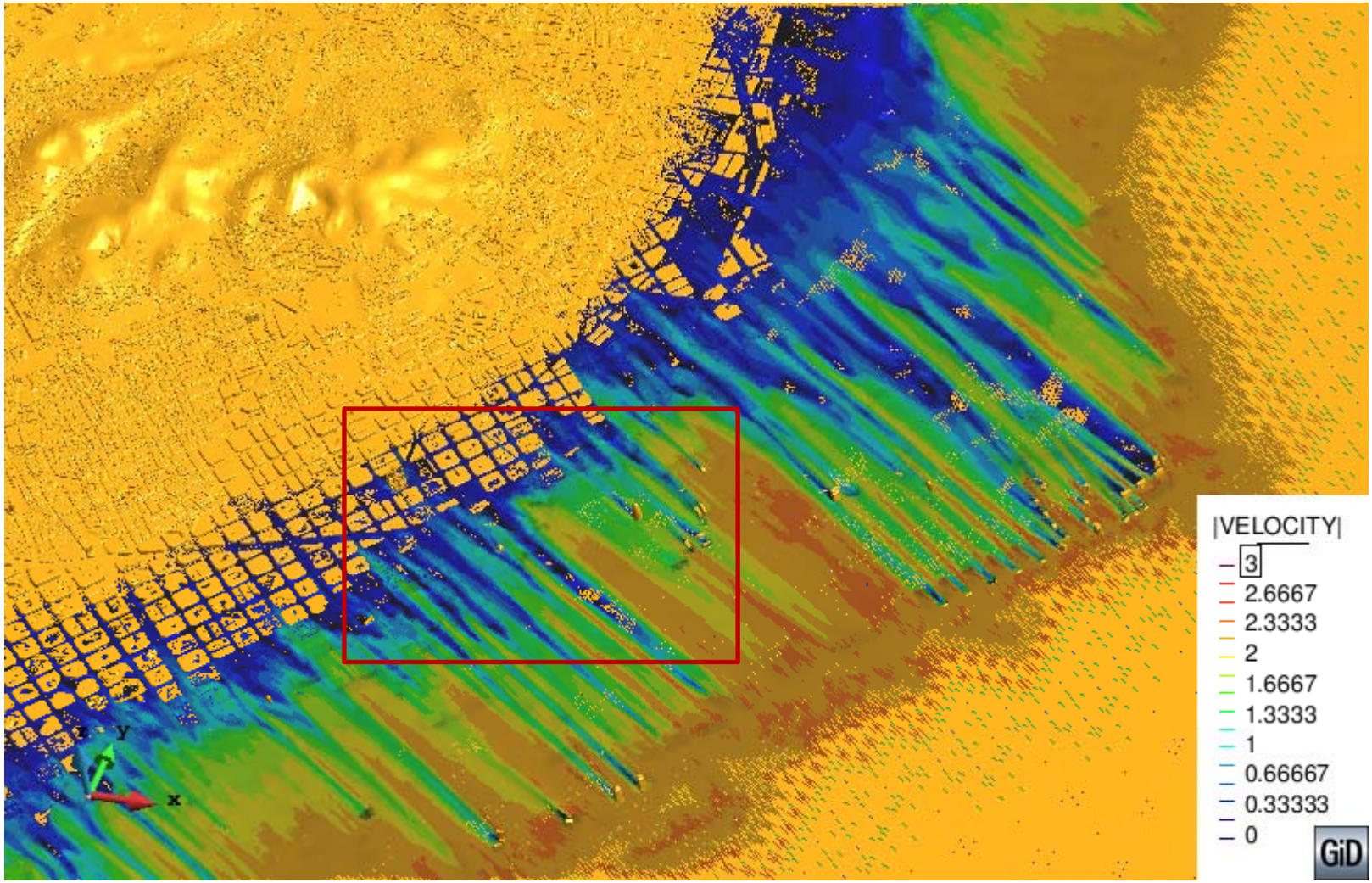
*The **Vision** of **VeLASSCo** is to provide new visual analysis methods for large-scale simulations serving the petabyte era and preparing the exabyte era by adopting Big Data tools/architectures for the engineering and scientific community leveraging new ways of in-situ processing for data analytics and hardware accelerated interactive visualization.*



- Integrate post-process analytics in a Big data framework embedded in the HPC, where the data is being calculated or stored.

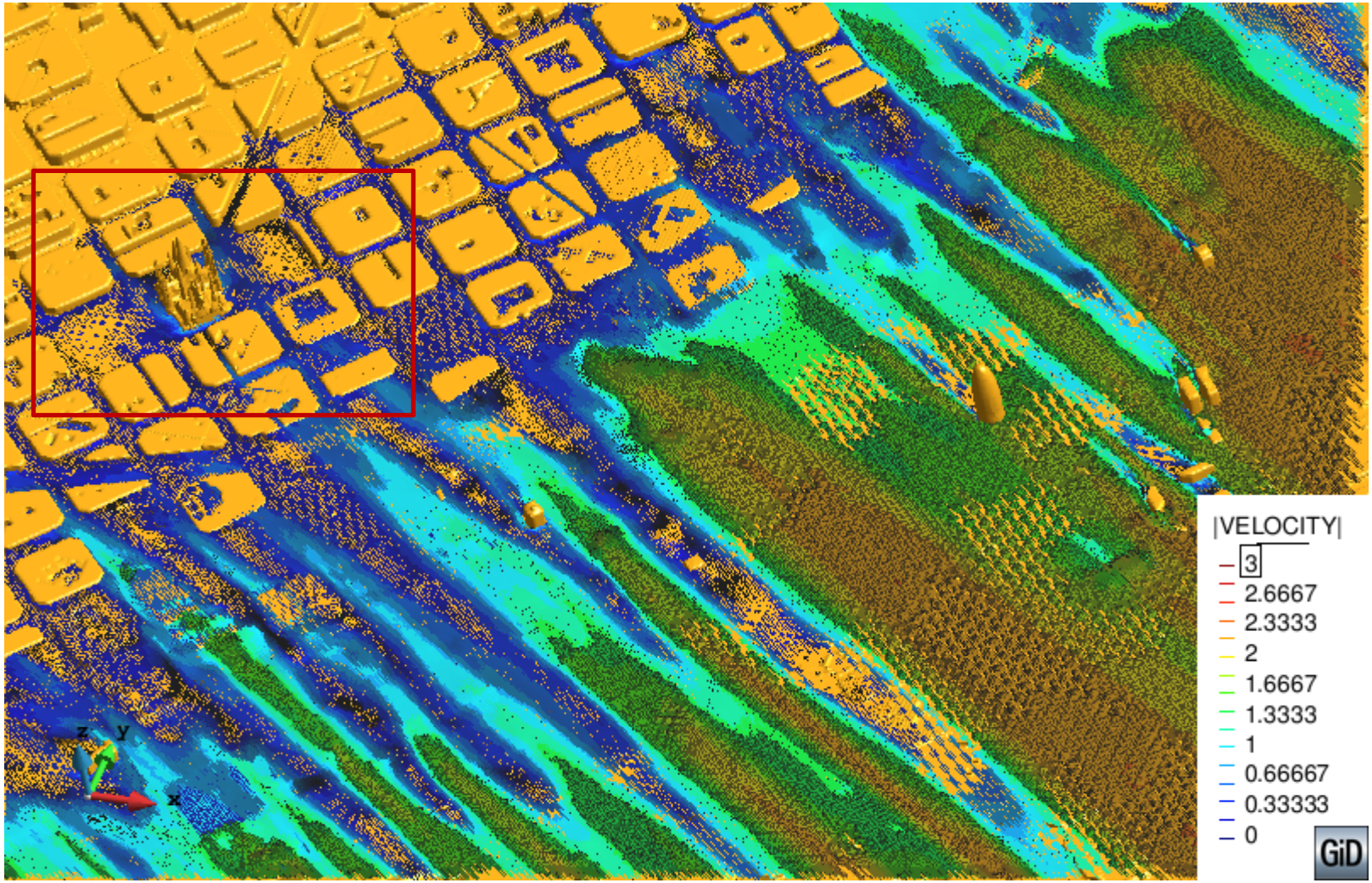
Wind flow around buildings @ 40 m

- 8m resolution: 100 M tetrahedrons



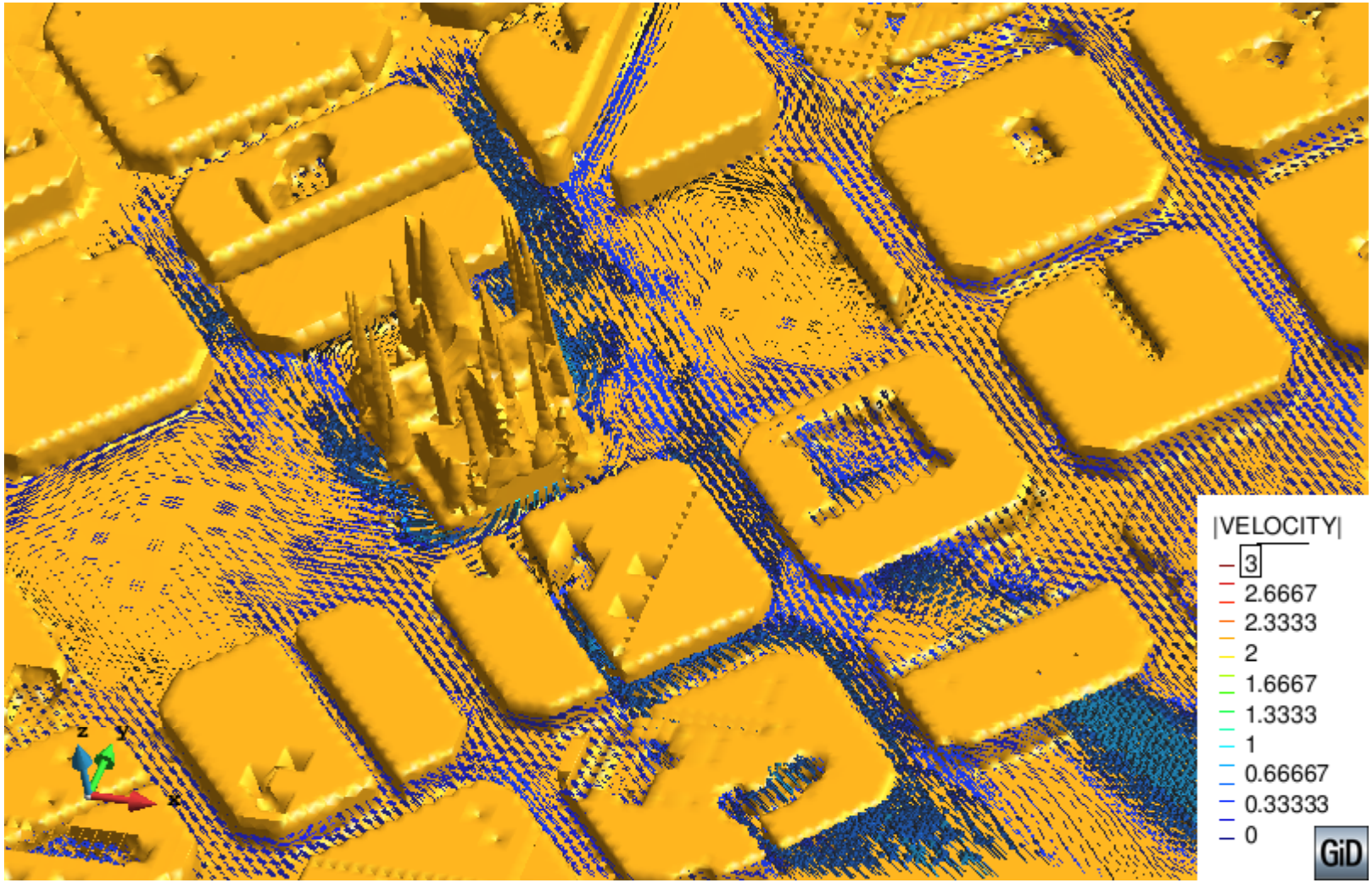
Wind flow around buildings @ 40 m

- 8m resolution: 100 M tetrahedrons



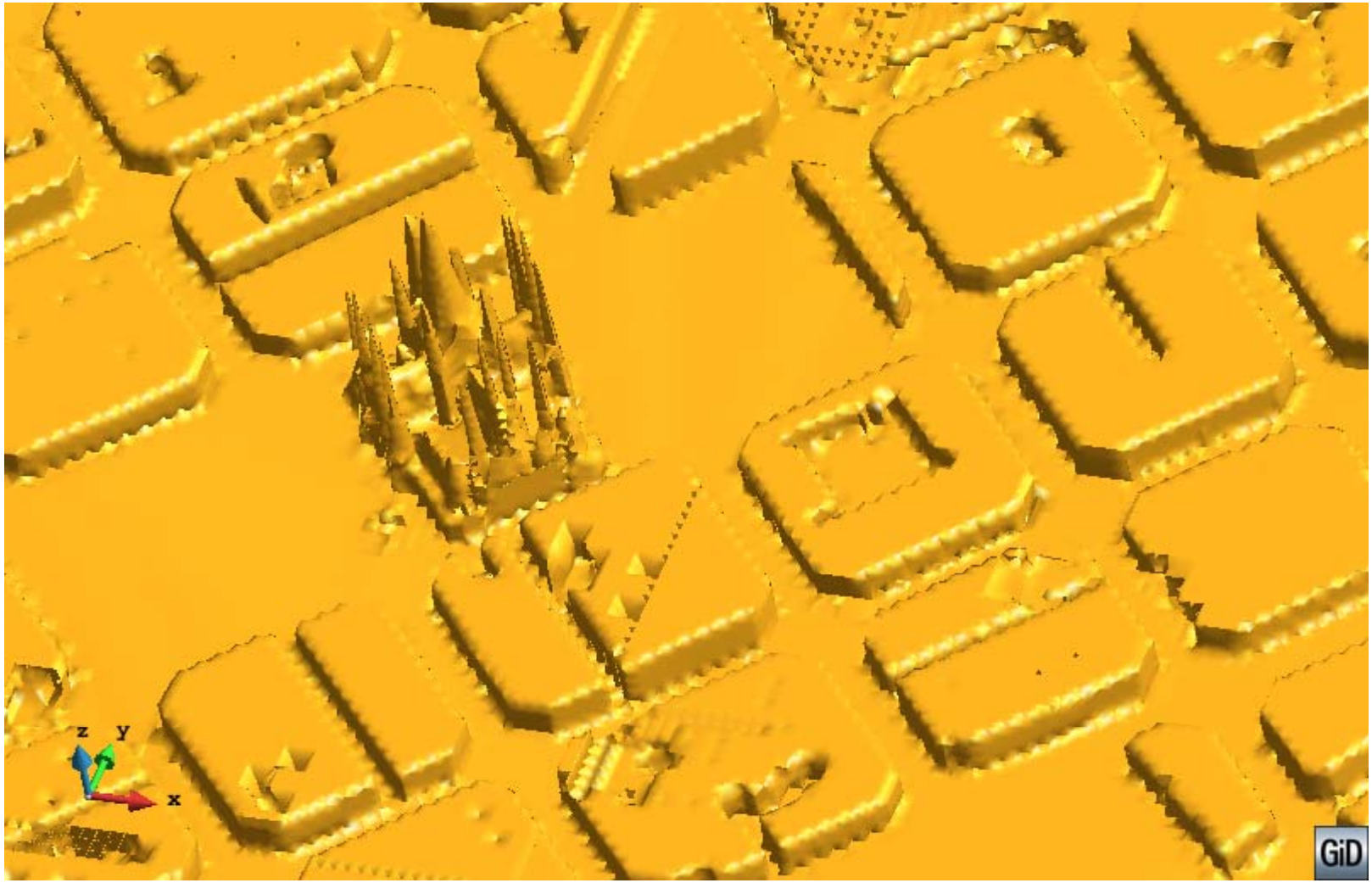
Wind flow around buildings @ 40 m

- 8m resolution: 100 M tetrahedrons



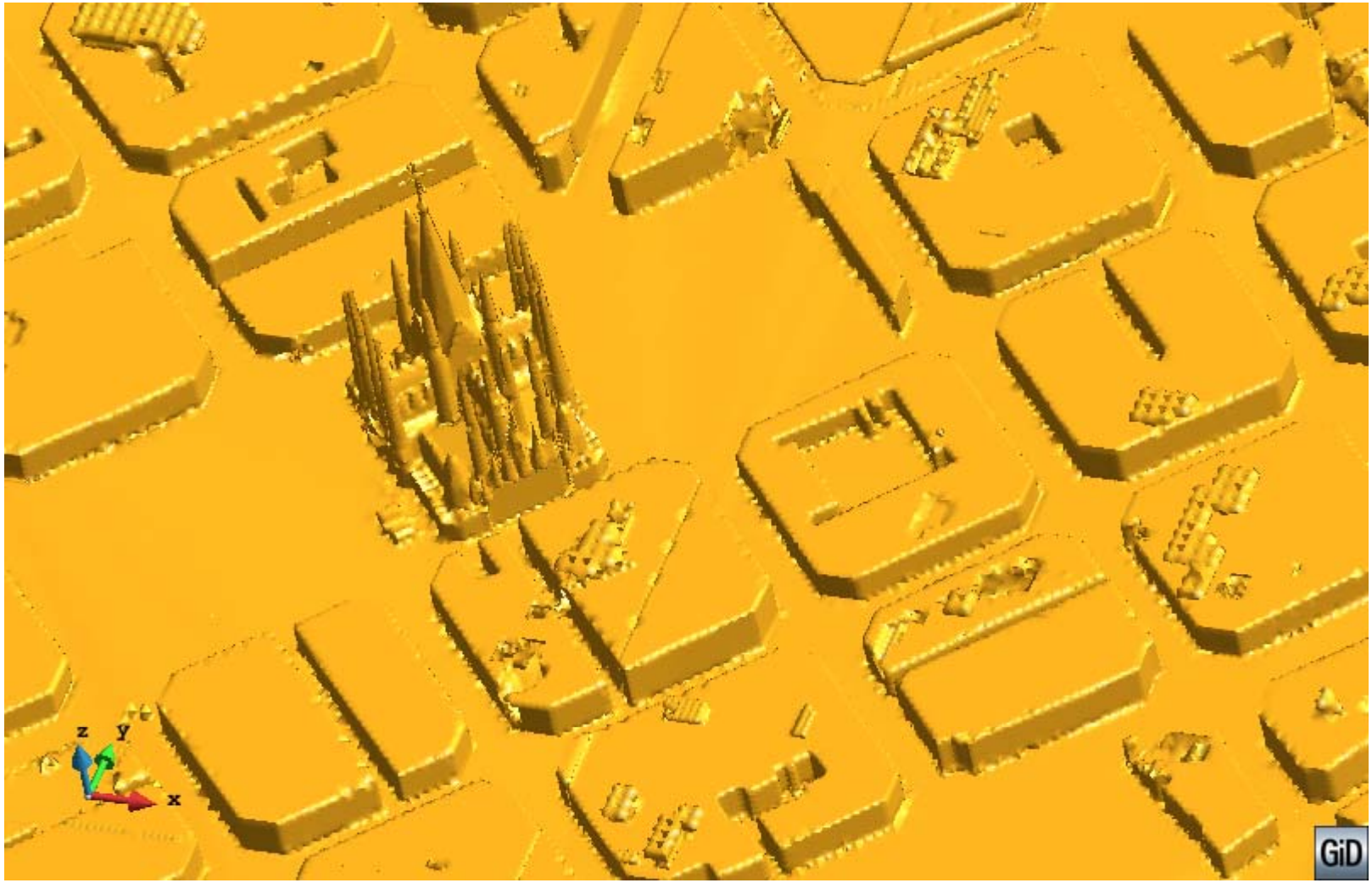
Wind flow around buildings @ 40 m

- 8m resolution: 100 M tetrahedrons



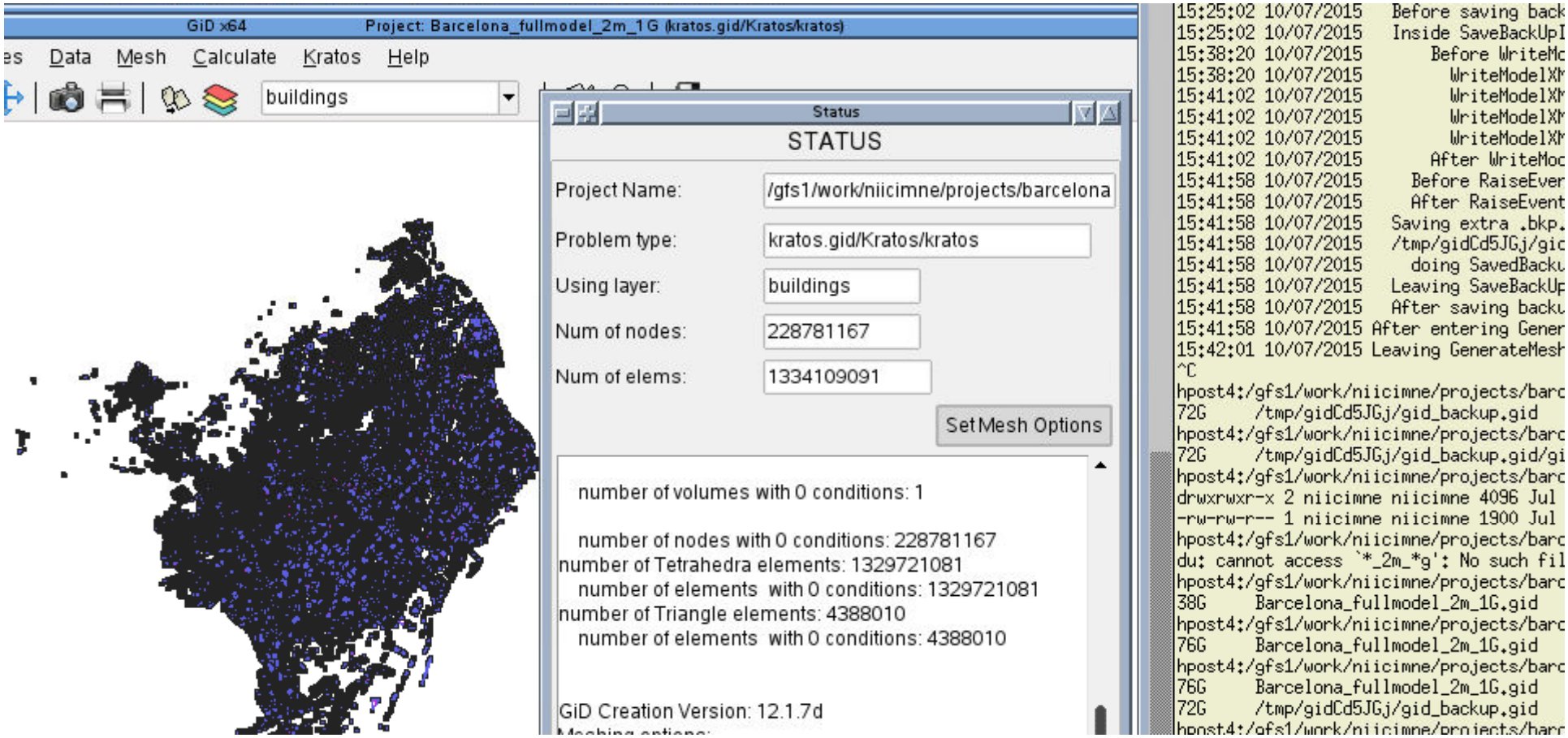
Wind flow: traditional post-process limits

- 4m resolution: 340 M tetrahedrons



1.3 billion tetrahedrons

- 2m resolution: several difficulties

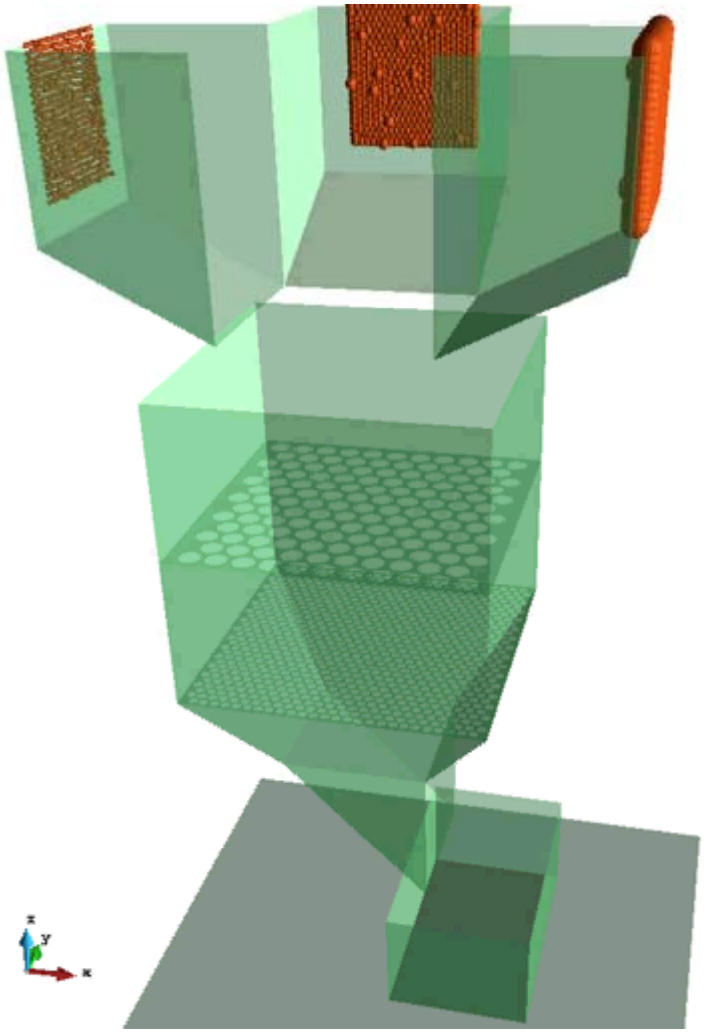


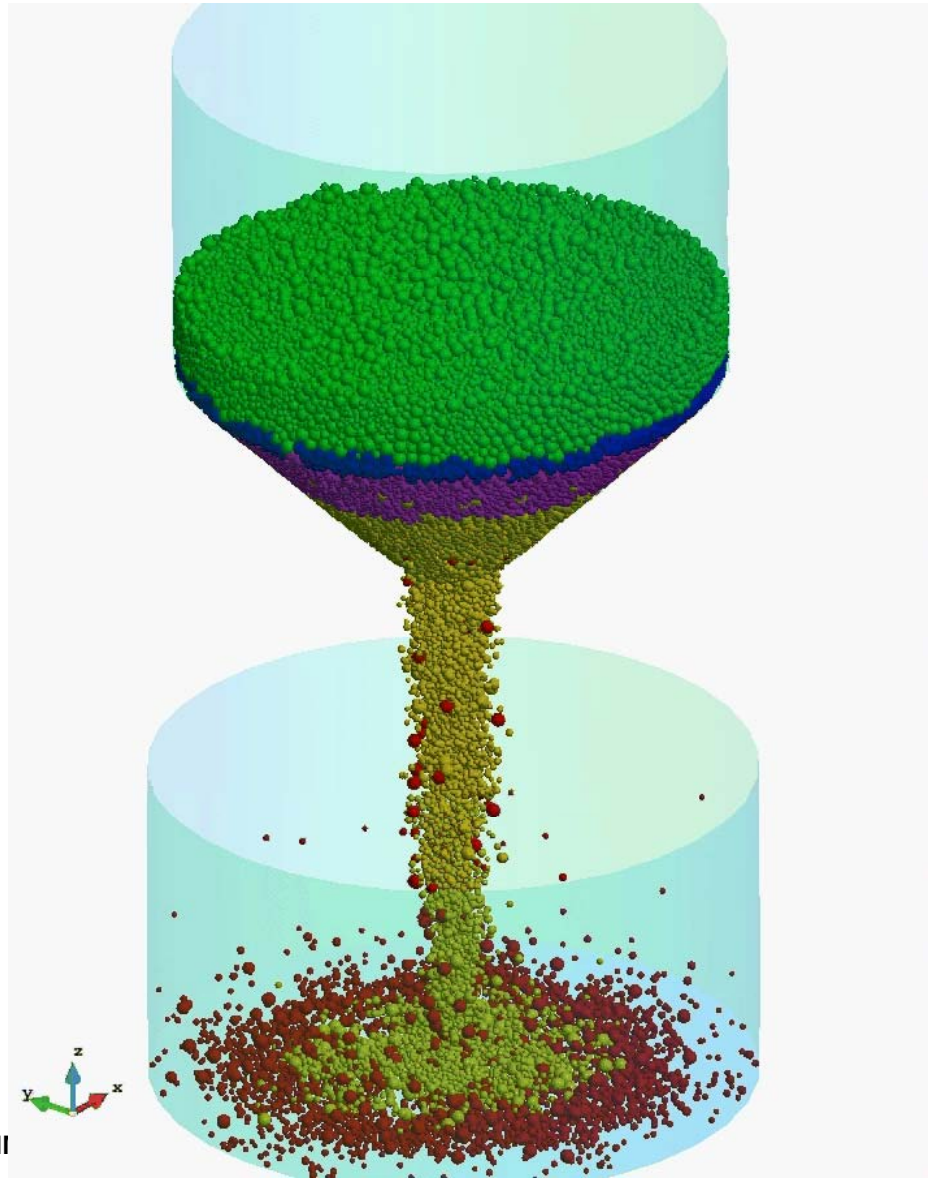
Remote cluster

- Barcelona 2m resolution: 1.3 billion tetrahedrons (2015)
  - 72 GB mesh only
  - 1 time-step (pressure, velocity) ~ 9 GB
  - 100 time-steps = 972 GB = 55 h @ 5MB/s
  - 1 time-step (distance, partition index, pressure, reaction, velocity) ~ 22 GB
-

Motivation

- Particle methods





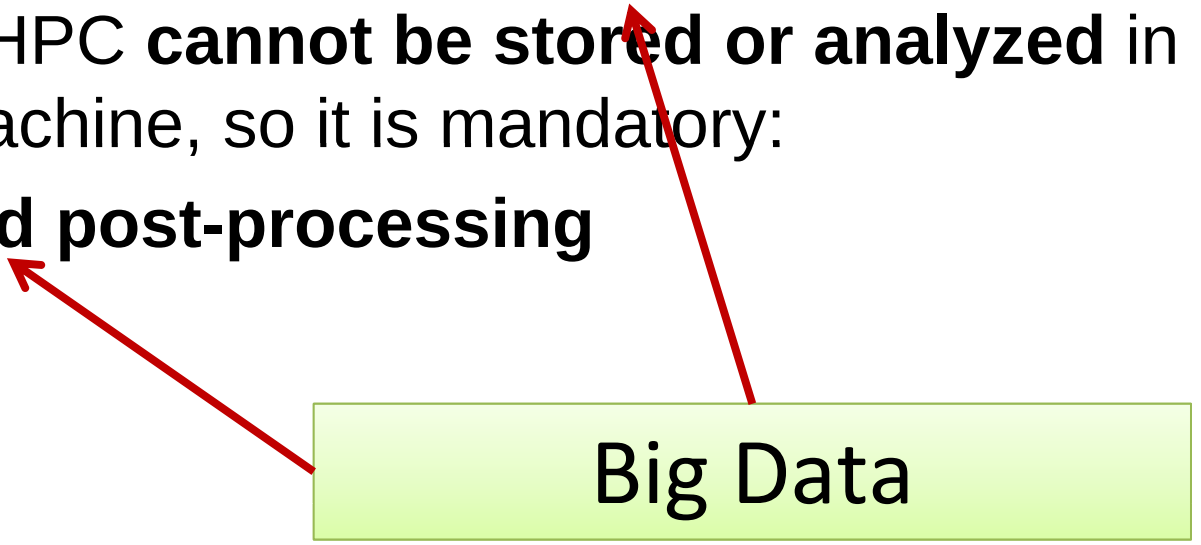
Target data

- Estimated data size in a couple of years

	DEM	FEM
Total size	**50 GB → 1 PB**	**30 GB → 50 TB**
Partitions		1 → 10,000
Particles / elements	10 million	8 million → 1 billion
Time-steps	1 billion	40 → 25,000
Variables per node	10 variables	2-8 scalars, 1-2 vectors, ?1 tensor?

VELaSSCo approach

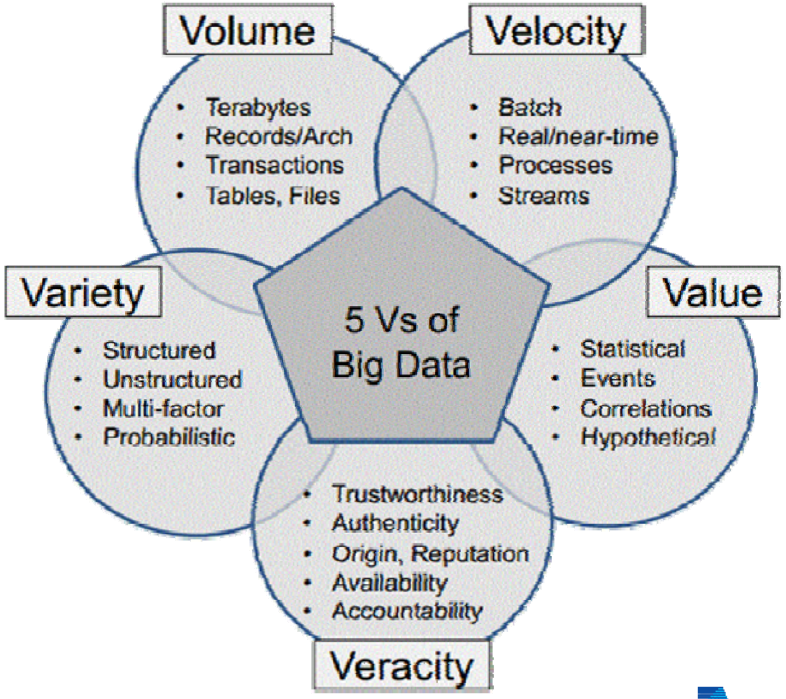
- Nowadays the **huge amount of data** provided by the solver in HPC **cannot be stored or analyzed** in one single machine, so it is mandatory:
  - **Distributed post-processing**



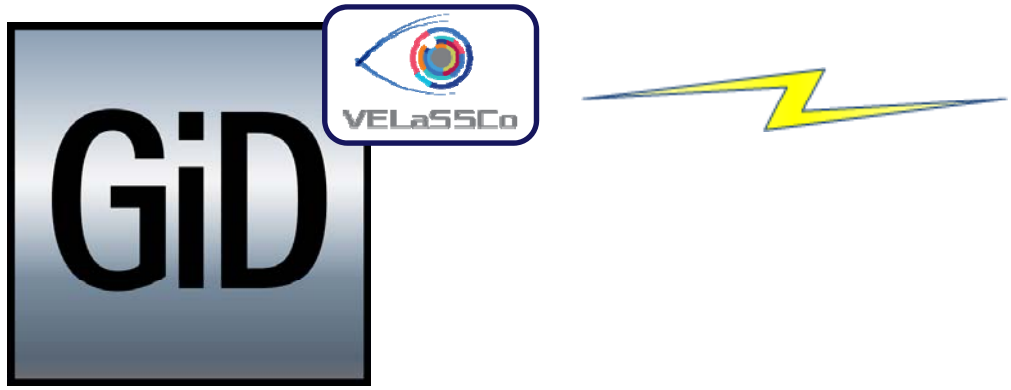
- Integrate post-process analytics in a Big Data framework embedded in the HPC, where the data is being calculated.

Big Data for what

- Store data in a distributed way: robust, redundancy
- Perform analytics in a distributed way:
  - Post-process = extract information + visualize
- High scalability



GiD point of view



- User will:
  - Launch GiD
  - Connect to VELaSSCo platform
  - Select model
  - Interact with a (simplified) version
  - Request results view: get a (simplified) view, while full-detailed view is received

Solver's point of view

- Output the results to the platform
- Using GiDpost:
  - GiD_PostInit(VELaSSCo, username, pass)
  - GiD_fOpen(…)
  - GiD_fWrite(…) // write mesh & results
  - GiD_fClose()
  - GiD_PostDone()

Contents

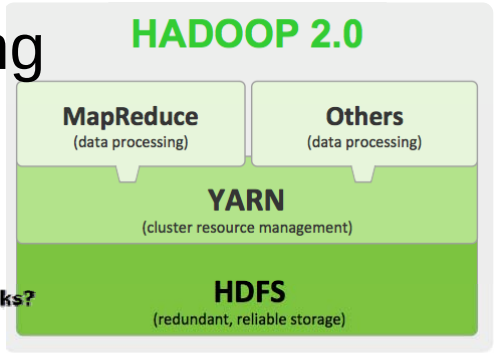
- VELaSSCo architecture
- Hadoop framework
- Current development status

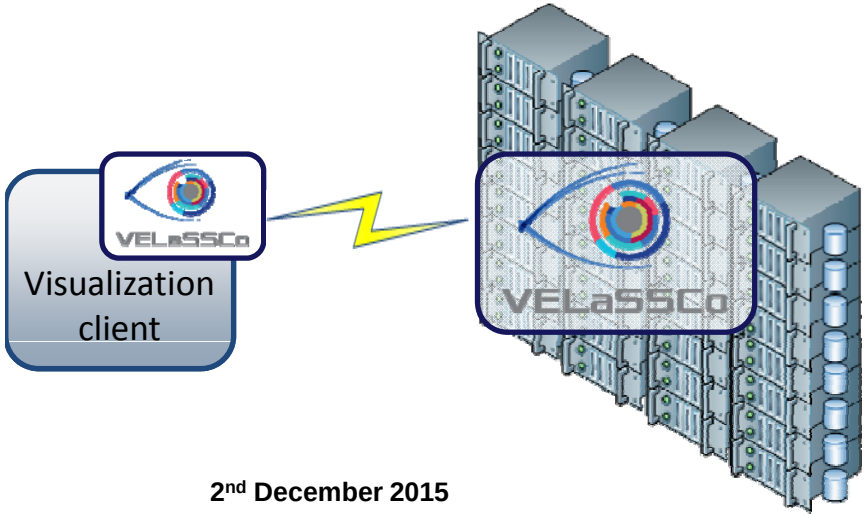
Contents

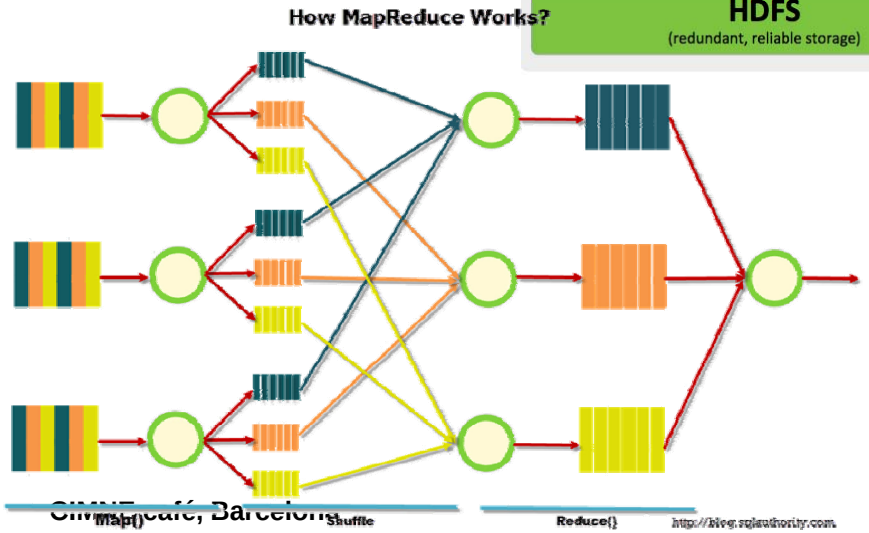
- VELaSSCo architecture
- Hadoop framework
- Current development status

VELaSSCo key-points

- Based on open source Hadoop
  - and also on Jotne's EDM
- Analyze the data where they are: in the HPC
- Post-process algorithms are distributed using Yarn and MapReduce
- Visualize on local machine







VELaSSCo architectures

- Two versions:
  - OpenSource: using Hadoop, HDFS, Hbase, YARN
  - CloseSource: using JOTNE's EDM
- Visualization client communication using Thrift
- Data ingestion using Apache flume:
  - From files or running simulation
- Runs on an HPC with:
  - Local storage on some nodes: Vnodes
  - Dedicated queue for Vnodes
  - Vnodes can also be used to calculate when idle

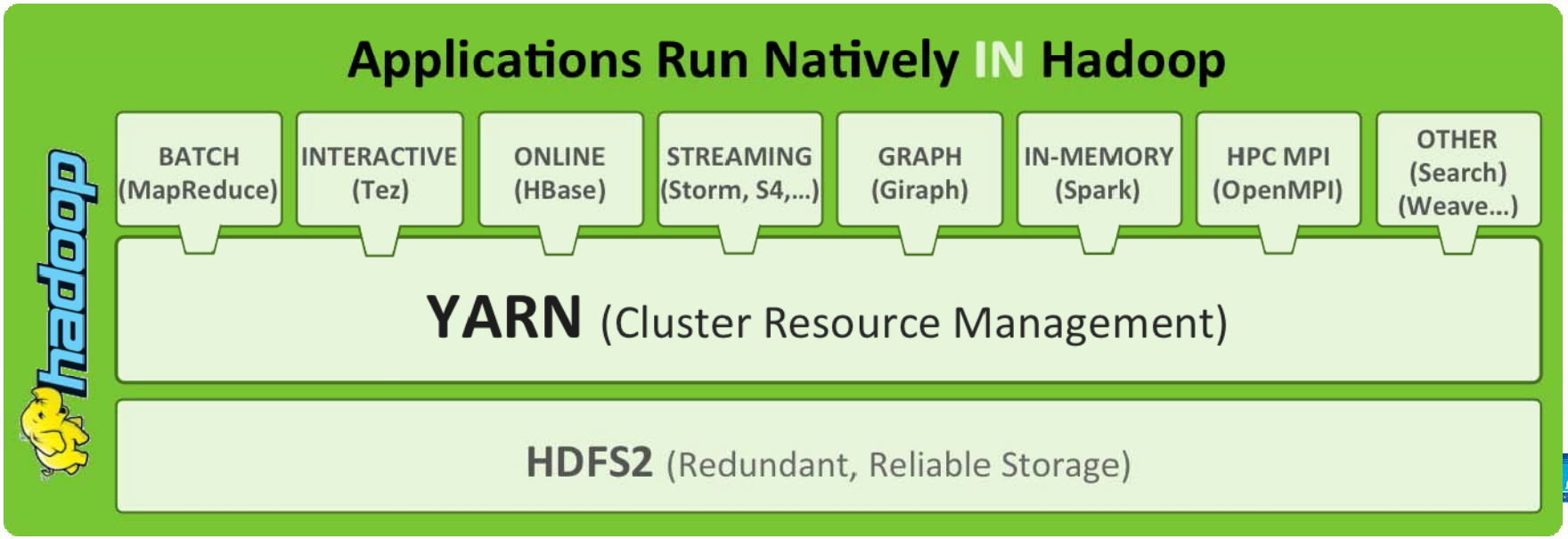
Hadoop

- “Apache Hadoop is an **open-source** software framework written in **Java** for **distributed storage** and distributed **processing** of very large data sets on computer clusters built from commodity hardware.” (Wikipedia)

- Core blocks:
  - HDFS: Hadoop Distributed File System
  - YARN: Yet Another (computing) Resource Manager, application scheduler
  - MapReduce: programming model
  - Common: tool-set, scripts for extensions

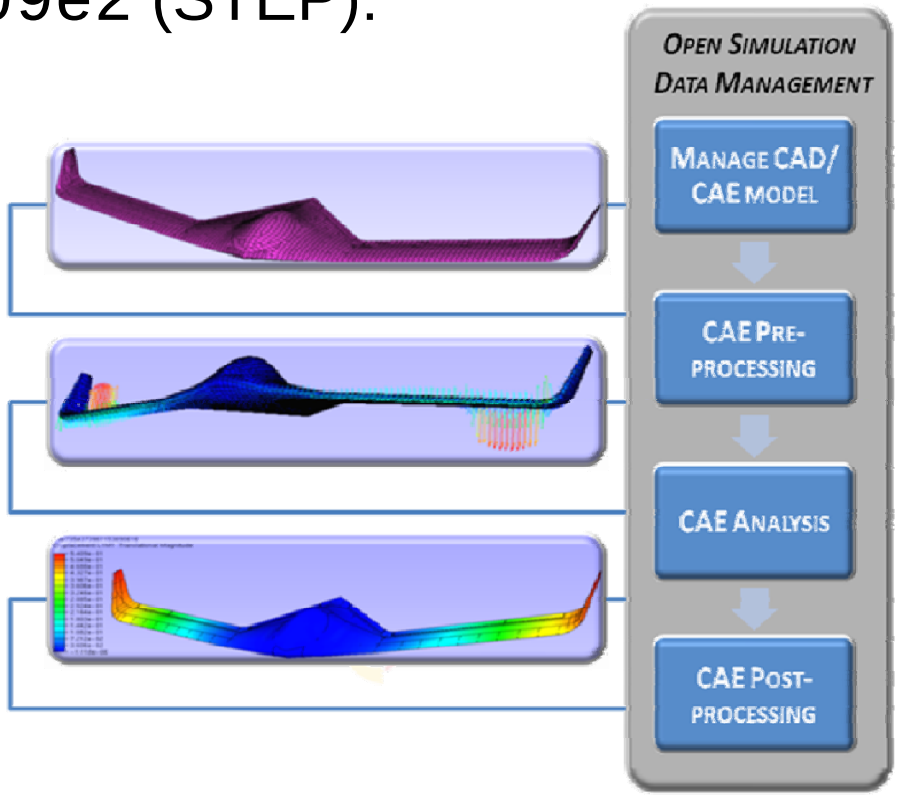
Hadoop

- Extensions:
  - Hbase: distributed (table) database
  - Hive: data access, query and analysis language over Hbase / HDFS, SQL-like
  - Spark: in-memory MapReduce engine also over YARN/HDFS



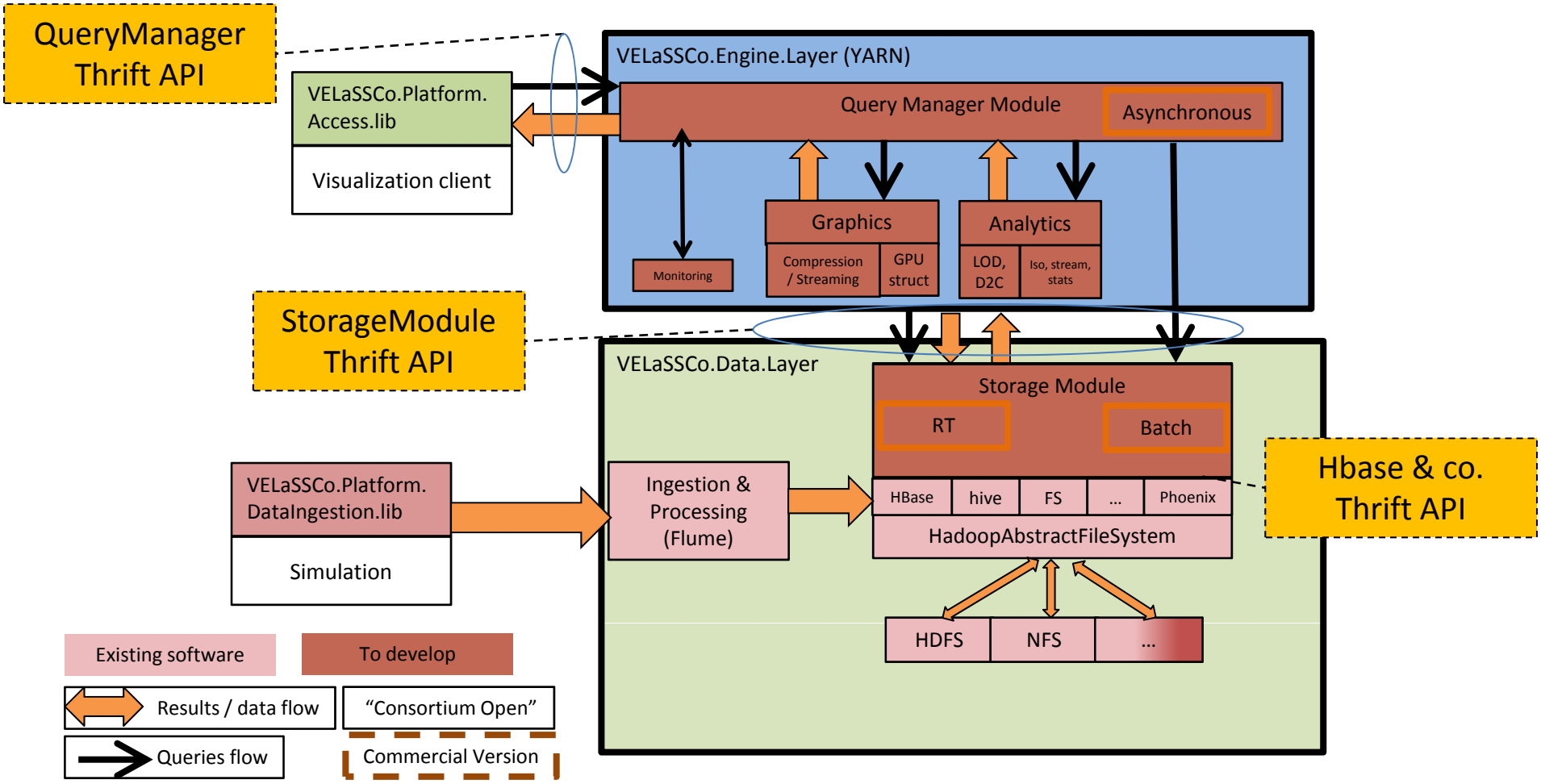
JOTNE's EDM

- Express Data Manager
- Open Simulation Data Management (SDM) using ISO standards, ISO 10303-209e2 (STEP):
  - Object Oriented DB
  - Conceptual models



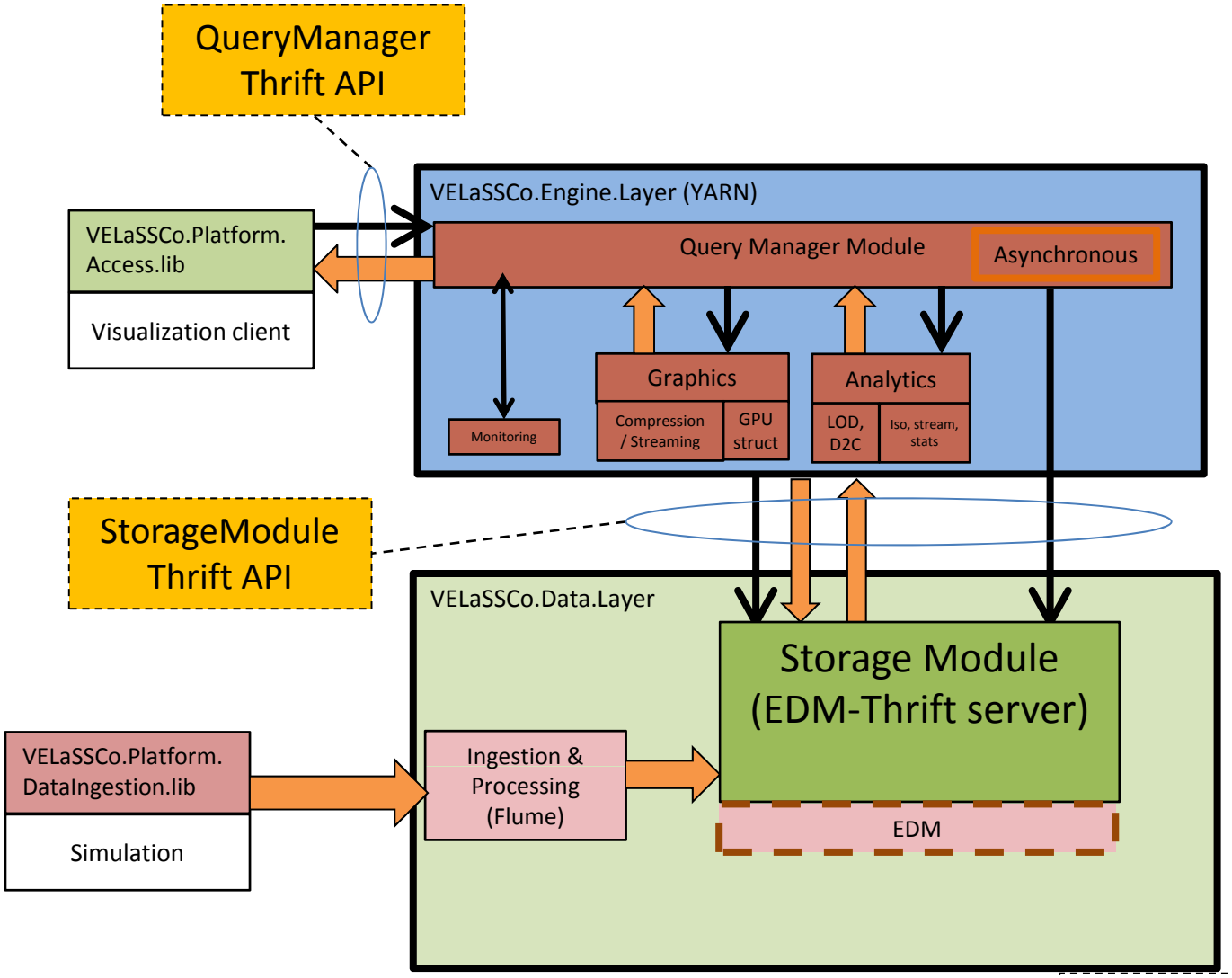
EXPRESS Data Manager
OpenSimDM™

VELaSSCo OpenSource architecture



Oslo 2015.06.16-19

VELaSSCo CloseSource architecture

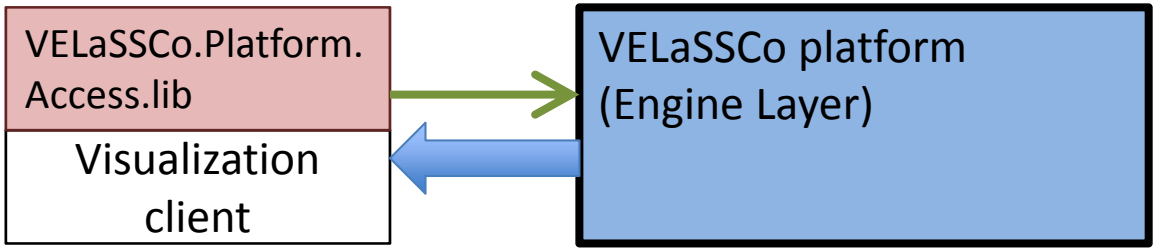


2015.11.18

VELaSSCo platform

- Engine Layer a stand-alone application with:
  - The Query Manager Module as thrift server
  - Graphics as CLI program and later as library
  - Analytics using: Hive, YARN MapReduce/… later eventually with Spark
  - Communicates with CLI and Thrift with Data Layer (Storage module)
- Data Layer:
  - Storage Module as thrift server
  - Will connect to Hbase (using CURL or Thrift) or EDM thrift server

User interaction: Vqueries



- User interaction generates VELaSSCo-queries
  - Show me the model
    → getBoundaryMesh() + RenderBoundaryMesh()
  - Visualize sea surface with velocity vectors
    → getIsoSurface() + interpolateResult(velocity) + RenderData()
  - Traveling from point A to B (*streaming version*)
    → loop of getMeshData(BoundingBox/UserView)
  - For costly queries:
    →doQuery(simplified model) + doQuery(full model)

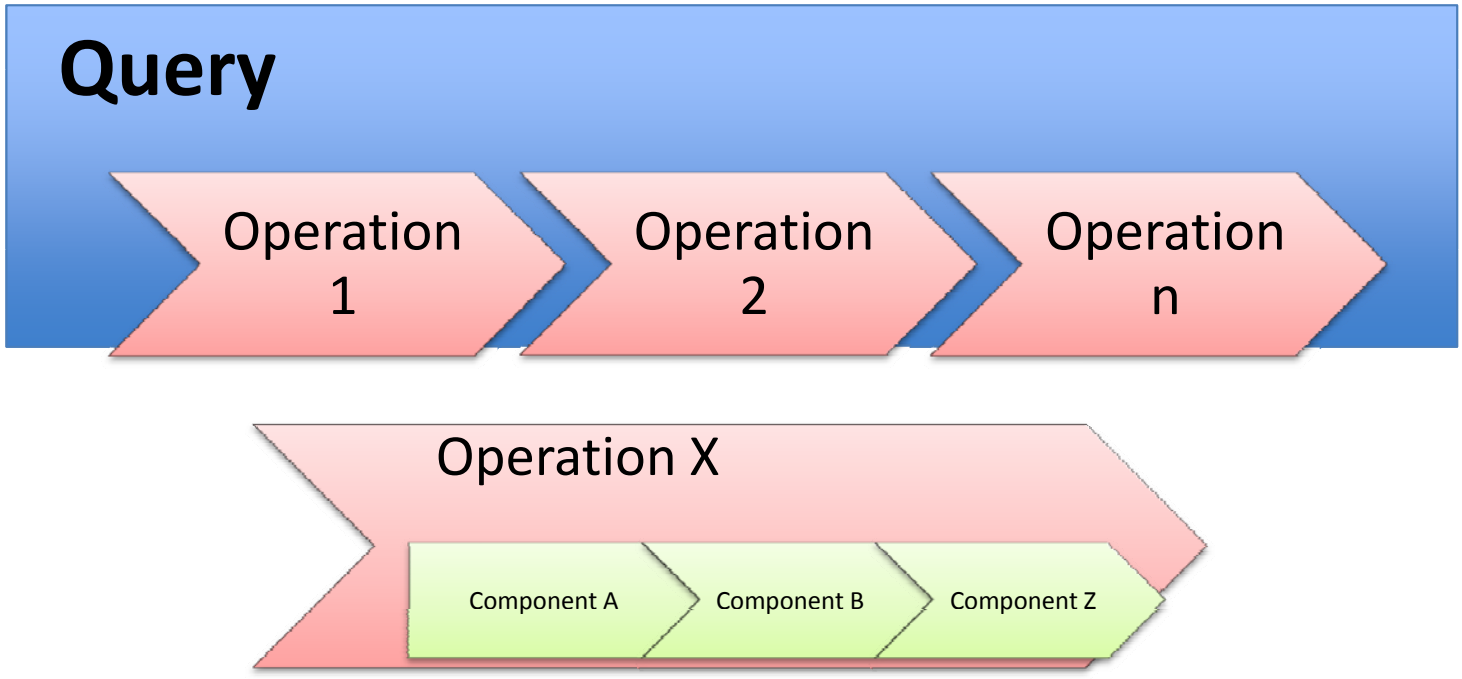
→ Communication using Thrift

VELaSSCo-queries families

- **Session Queries (SQ):** User connection, model selection, thumbnails and validation status
- **Direct Result Queries (DRQ):** Get mesh and results information from the data without analytics, such as: get result of node or element number n
- **Result Analysis Queries (RAQ):** Postprocess operations on data (data analytics), such as: calculate skin mesh, streamlines, isosurface or cut
- **Data Ingestion Queries (DIQ):** Insert data in the Data Layer, and merge into a Hbase data table

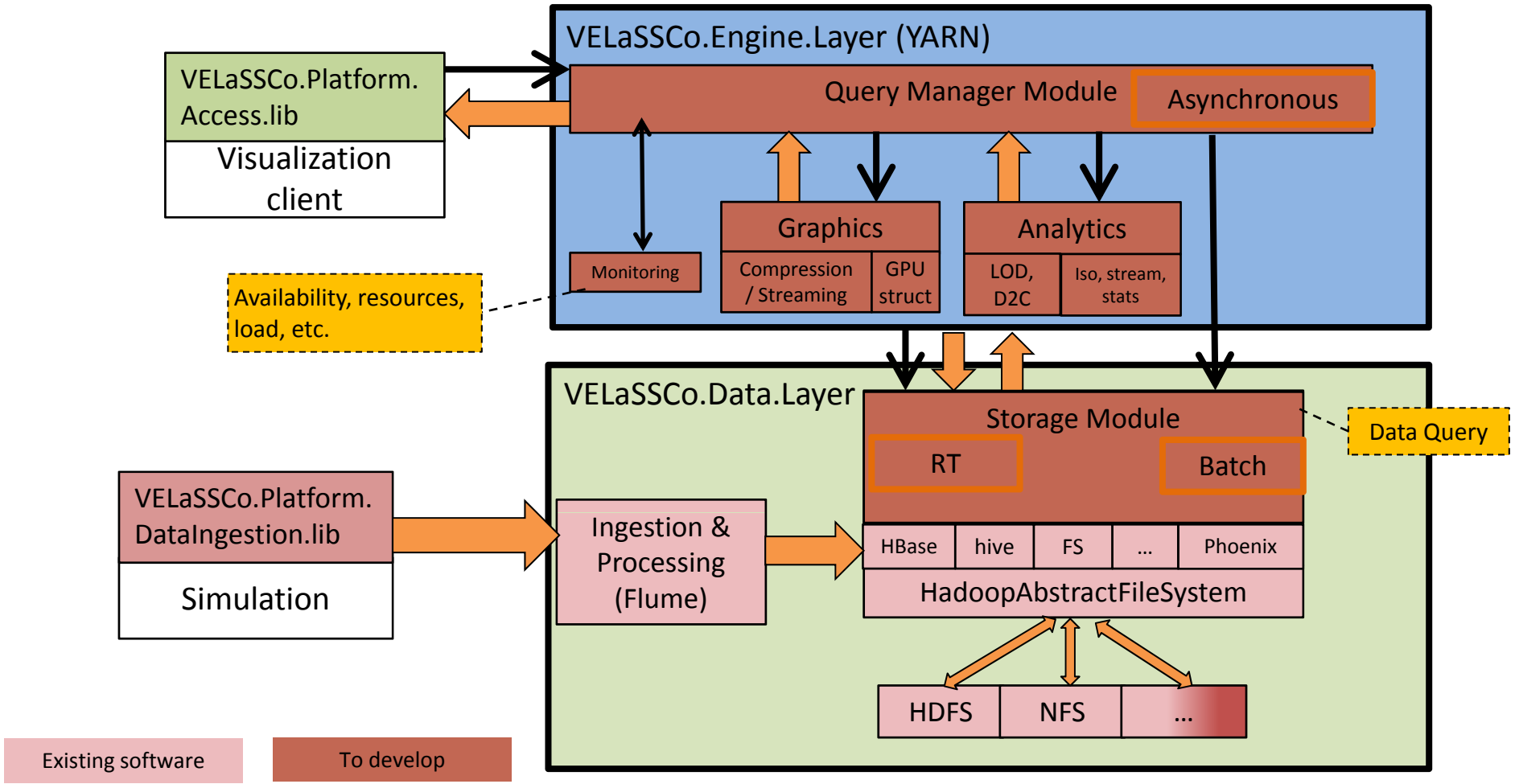
Development: Vqueries

- VQuery decomposition into operation and components:



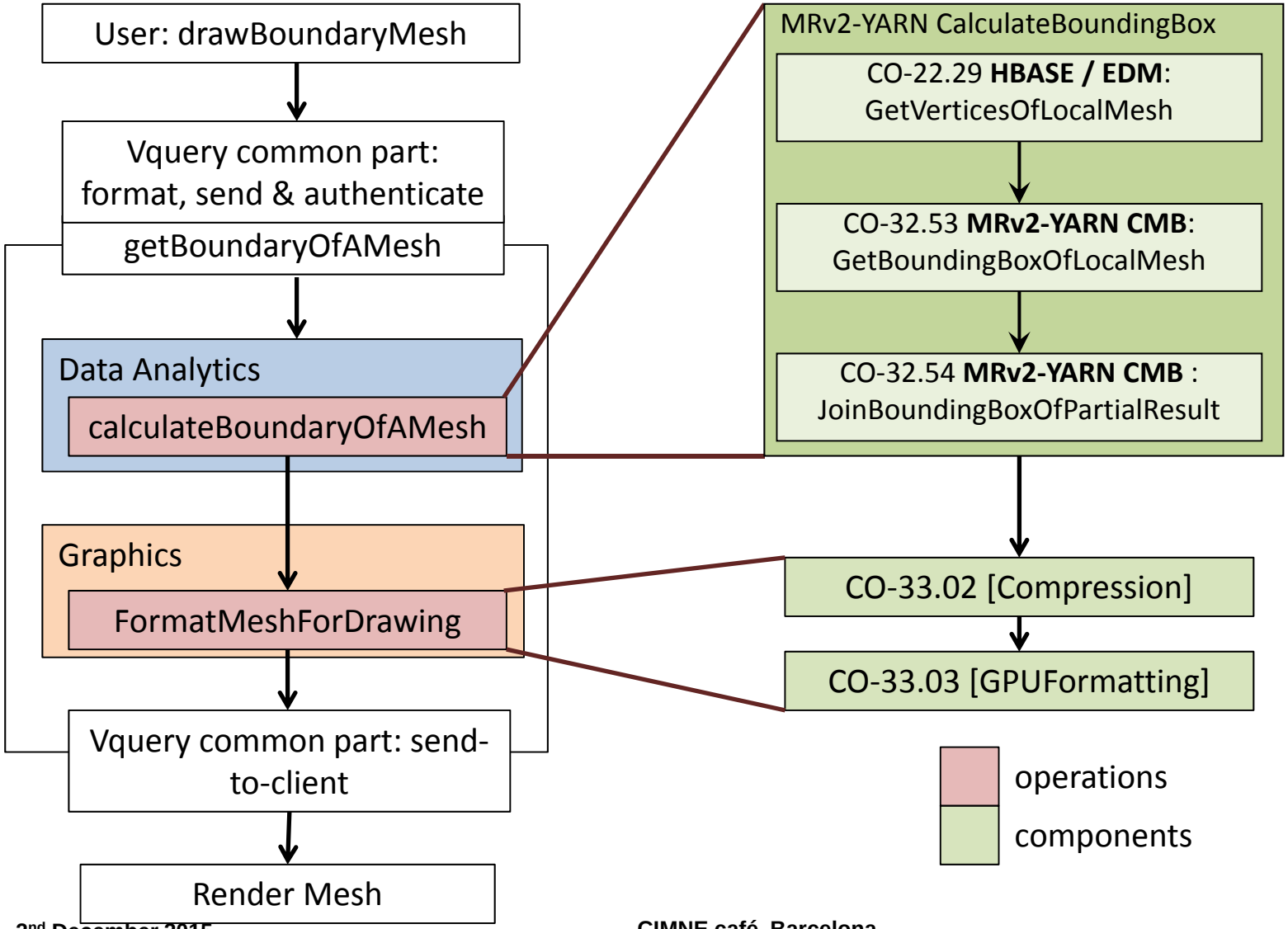
- Allows to monitor the progress of the platform

VELaSSCo OpenSource architecture

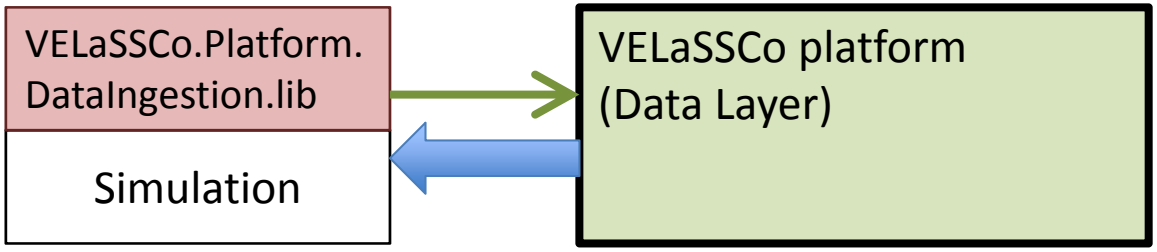


Oslo 2015.06.16-19

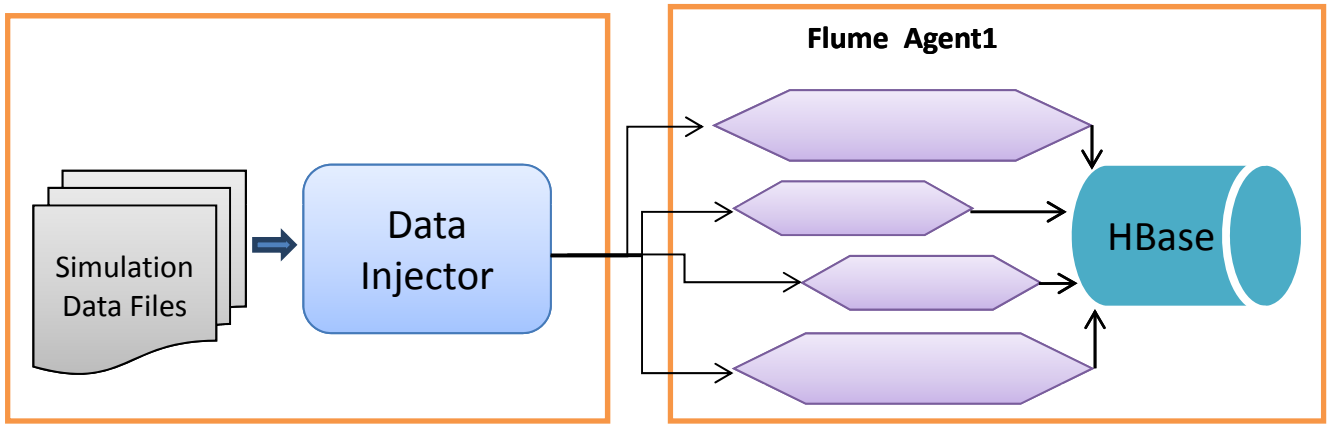
Vquery example: getBoundaryOfAMesh()



Data Ingestion



- Through Flume agents, as RESTfull service
- Injector applications using Hbase-thrift
- From already existing data
- From a running simulation: using GiDpost



Development plan

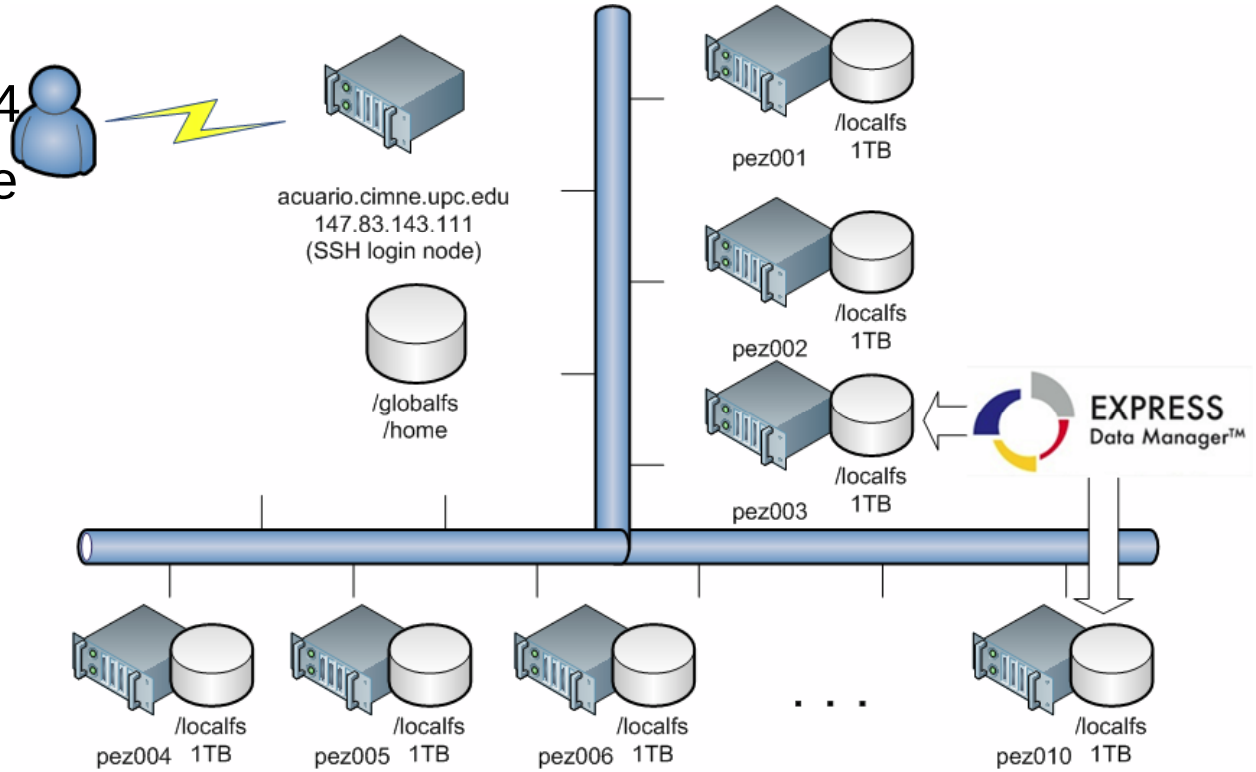
- First prototype with simple queries:
  - User login, model information, summaries,
  - Mesh view: surfaces, skins,
  - Results views: results on nodes,
  - Analytics:
    - Bounding box, boundary/skin, point interpolation, discrete 2 continuum transformation, Coarser model
    - Cut planes with interpolation (segments),
    - (Isosurfaces with interpolation)
  - Visualization:
    - On demand
    - 1-LOD

Development plan

- Final prototype with complex queries:
  - Analytics:
    - Isosurfaces with interpolation
    - Results statistics, stream lines,
    - Bspline representation for lines, surfaces, ?volumes?
  - Pre-computed queries: detection of most used ones
  - Visualization:
    - Navigation, streaming
    - Several LOD

Done so far

- Ingestion: almost done
- Few data information retrieval and analytics queries
- Deployed in acuario: 8 + 2 nodes
  - 2 x QuadCores E5410 @ 2.33 GHz
  - RAM: 16 GB
  - Infiniband DDRx4
  - 1TB local storage



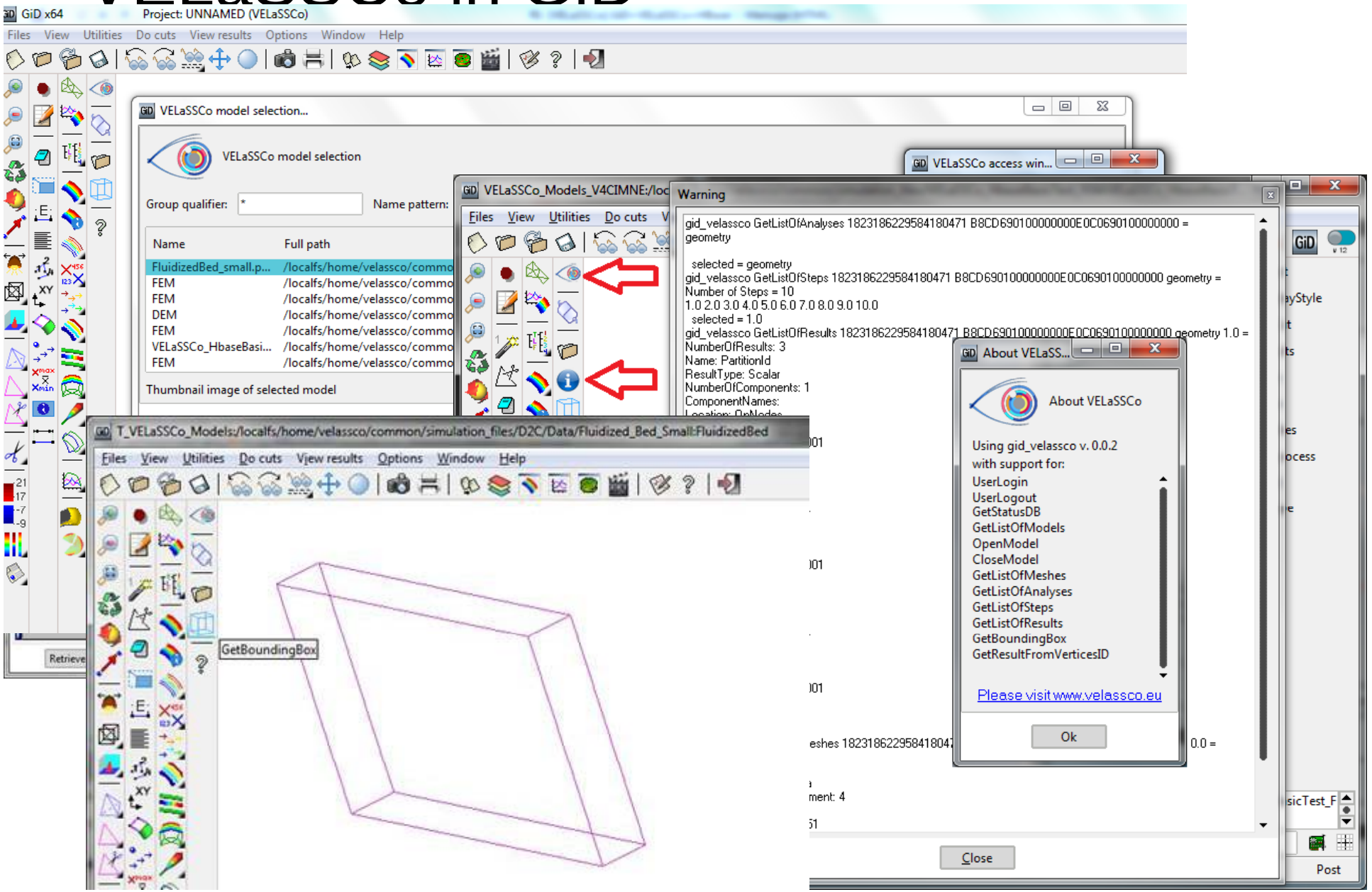
Some preliminary numbers (Hadoop)

- Ingestion:
  - Current speed is 1.4 MB/s (Hbase-thrift), 2.6 MB/s (EDM-thrift)
  - Initial speed is 122 KB/s, 523 KB/s (Hbase-REST)
- Information:
  - Getting model information: 34,406 rows in 3.7 s.
- GetBoundingBox (MapReduce):
  - 24 M tetrahedra in 128 sub-domains:
    - Job time: 38 s. / wall clock: 63 s.
  - 485 M particles in 40,801 steps:
    - Job time: 334 s. / wall clock: 346 s.

Development status

- First prototype:
  - ready by December / January
  - deployed in UEDIN's cluster using 20 nodes with:
    - 2 x Xeon E5-2630v3 2.4 GHz = 16 cores
    - 64 GB RAM
    - 700 GB local storage = 14 TB → Project goal 200 TB
  - Evaluation event at mid January
- Revision of design and architecture
- Final prototype September 2016

VELaSSCo in GiD



Visual Analysis for Extremely Large-Scale Scientific Computing

Thanks for your attention

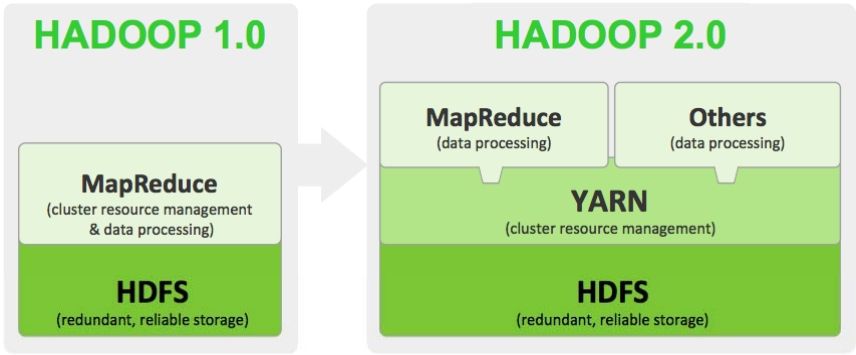
Questions or comments ?

Contents

- VELaSSCo architecture
- Hadoop framework
- Current development status

Hadoop eco-system

- A wide variety of plug-ins
  - Support of SQL
  - Storage (columns, files, …)
  - File System
  - Computational model
  - Query model
  - Management
  - Deployment ….
- A main evolution with YARN
  - Up to 10.000 nodes per cluster
  - Efficient cluster usage with Yarn
  - HDFS 2
  - Compatible with existing extensions
- Hadoop distributions: Cloudera, Ambari, Intel's hadoop, Apache Hadoop, Hortonworks (HDP), …



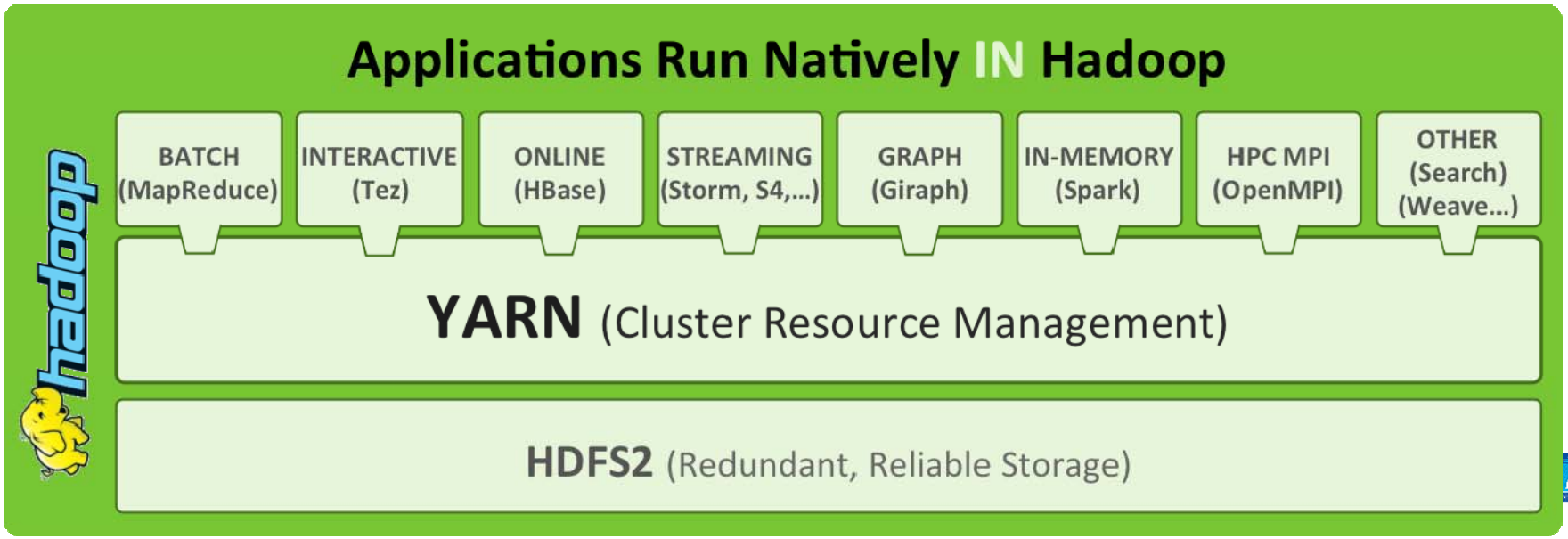
Hadoop

- “Apache Hadoop is an **open-source** software framework written in **Java** for **distributed storage** and distributed **processing** of very large data sets on computer clusters built from commodity hardware.” (Wikipedia)

- Core blocks:
  - HDFS: Hadoop Distributed File System
  - YARN: Yet Another (computing) Resource Manager, application scheduler
  - MapReduce: programming model
  - Common: tool-ser, scripts for extensions

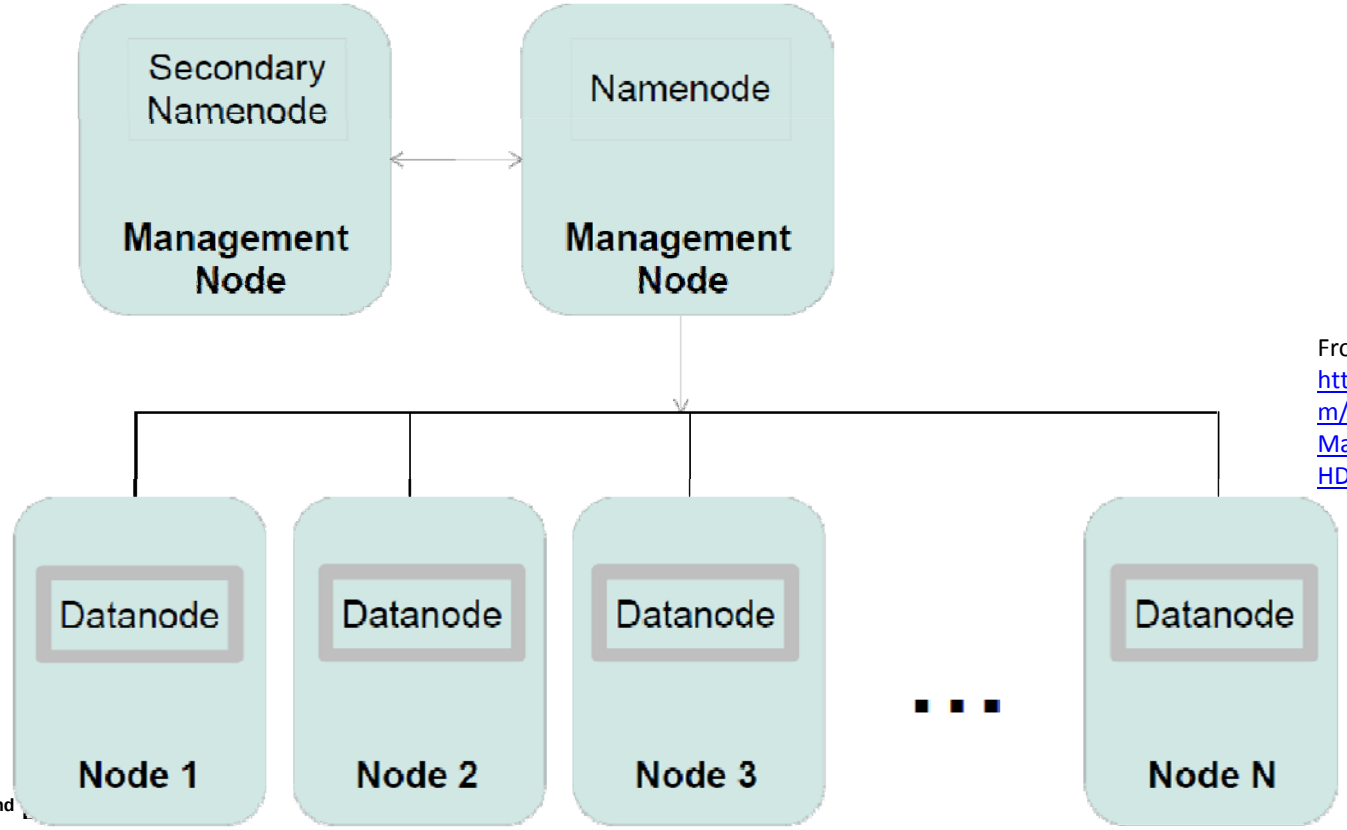
Hadoop

- Extensions:
  - Hbase: distributed (table) database
  - Hive: data access, query and analysis language over Hbase / HDFS, SQL-like
  - Spark: in-memory MapReduce engine also over YARN/HDFS



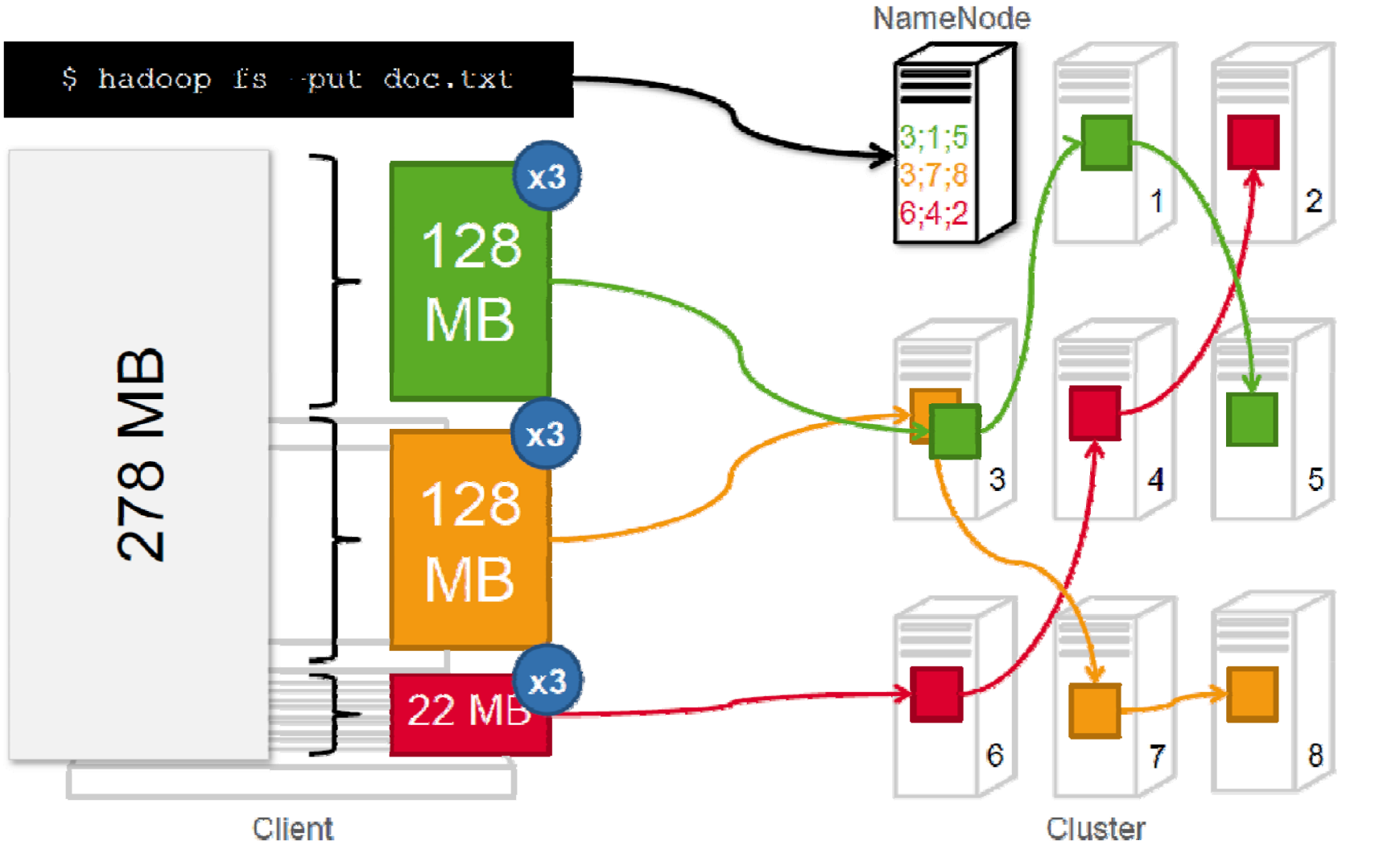
HDFS

- Store files in a distributed, split and redundant way.
- Default chunk size = 64/128 MB, redundancy level = 3
- 1 Name node = metadata (Files → Blocks → Nodes)



From "HDFS overview" http://courses.coreservlets.com/Course-Materials/pdf/hadoop/02-HDFS_1-Overview.pdf

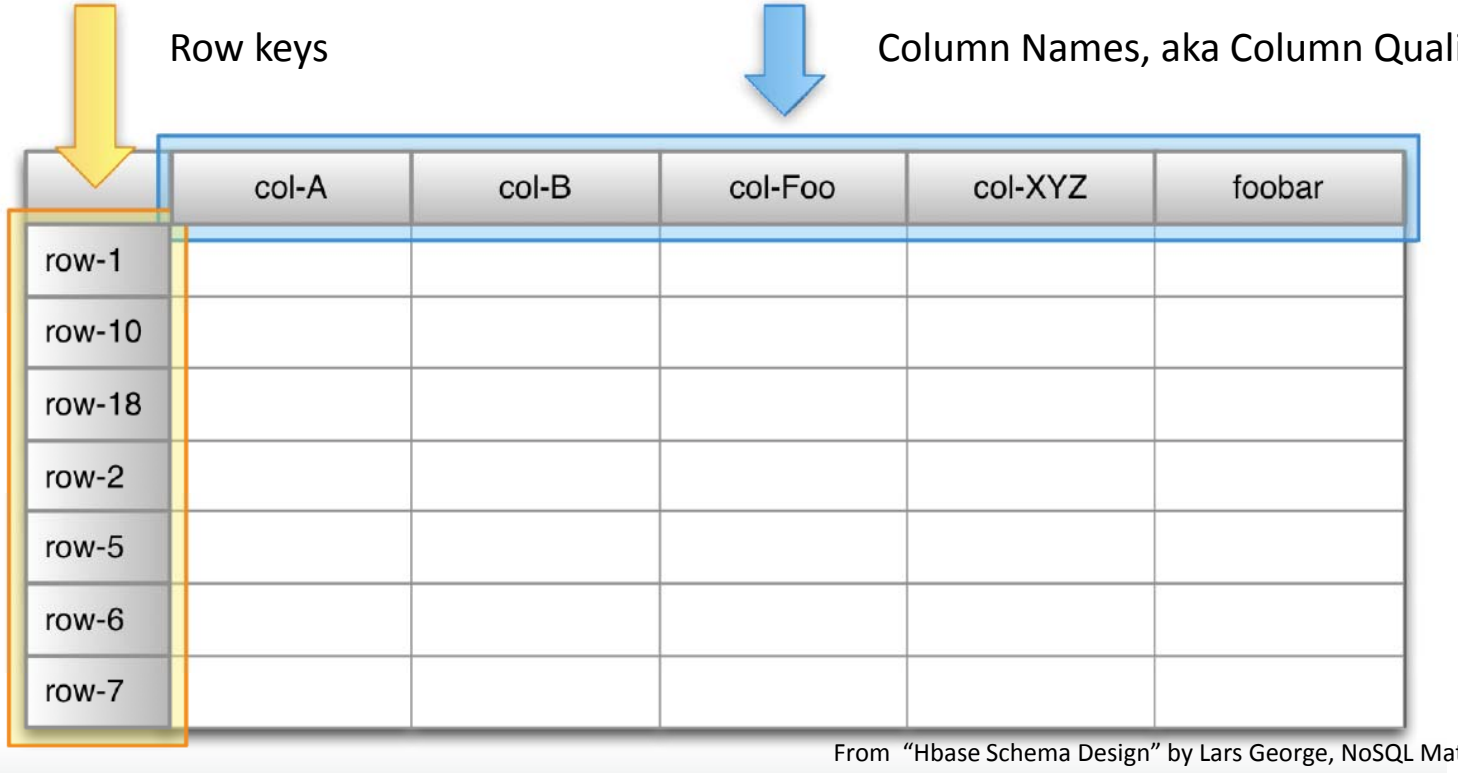
HDFS



From "Hadoop in a Nutshell"
http://www.metafinanz.de/sites/default/files/Hadoop_in_a_Nutshell.pdf

HBase

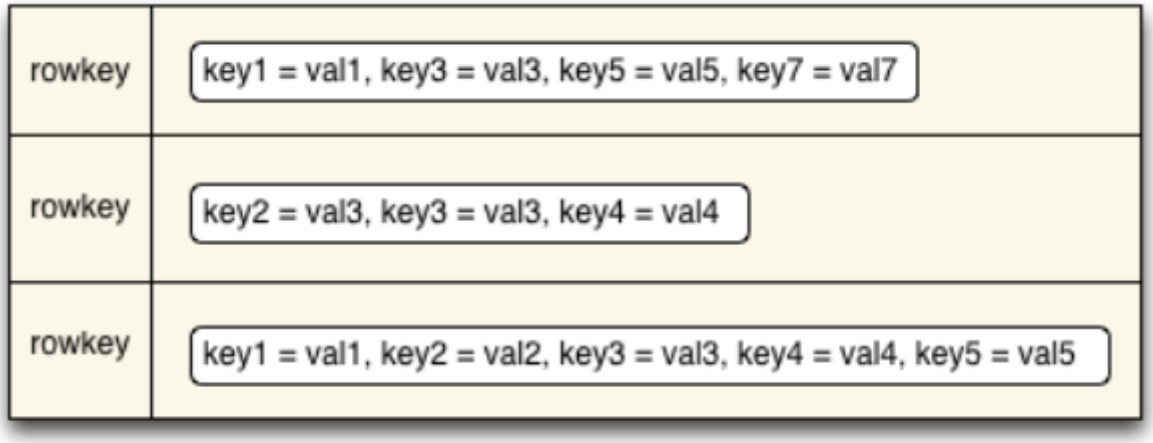
- Table (column) oriented distributed data store, over HDFS
- In Java providing thrift, rest, … services
- Integrated in MapReduce



From "Hbase Schema Design" by Lars George, NoSQL Matters, 2013

Hbase basic

- **Table**: design-time namespace, has many rows.
- **Row**: atomic key/value container, with one row key
- **Column**: a key in the k/v container inside a row
- **Value**: a value in the k/v container inside a row



From "Hbase Schema Design" by Ian Varley, HBaseCon 2012

Hbase column families

- **Table**: design-time namespace, has many rows.
- **Row**: atomic key/value container, with one row key
- **Column Family**: groups ‘related columns’ / divide rows into physical files.
- **Column-qualifiers**: a key in the k/v container inside a row

- **Value**: a value in the k/v container inside a row

- → Row is atomic: flushed periodically
  - Colum-Families: groups of columns with ‘related’ data:
    - to store in separate files, do compression, in-memory options, …
- → Family definitions are static, limited to small number of families
- → Column-qualifiers may vary from row to row.

From “Hbase Schema Design” by Ian Varley, HBaseCon 2012

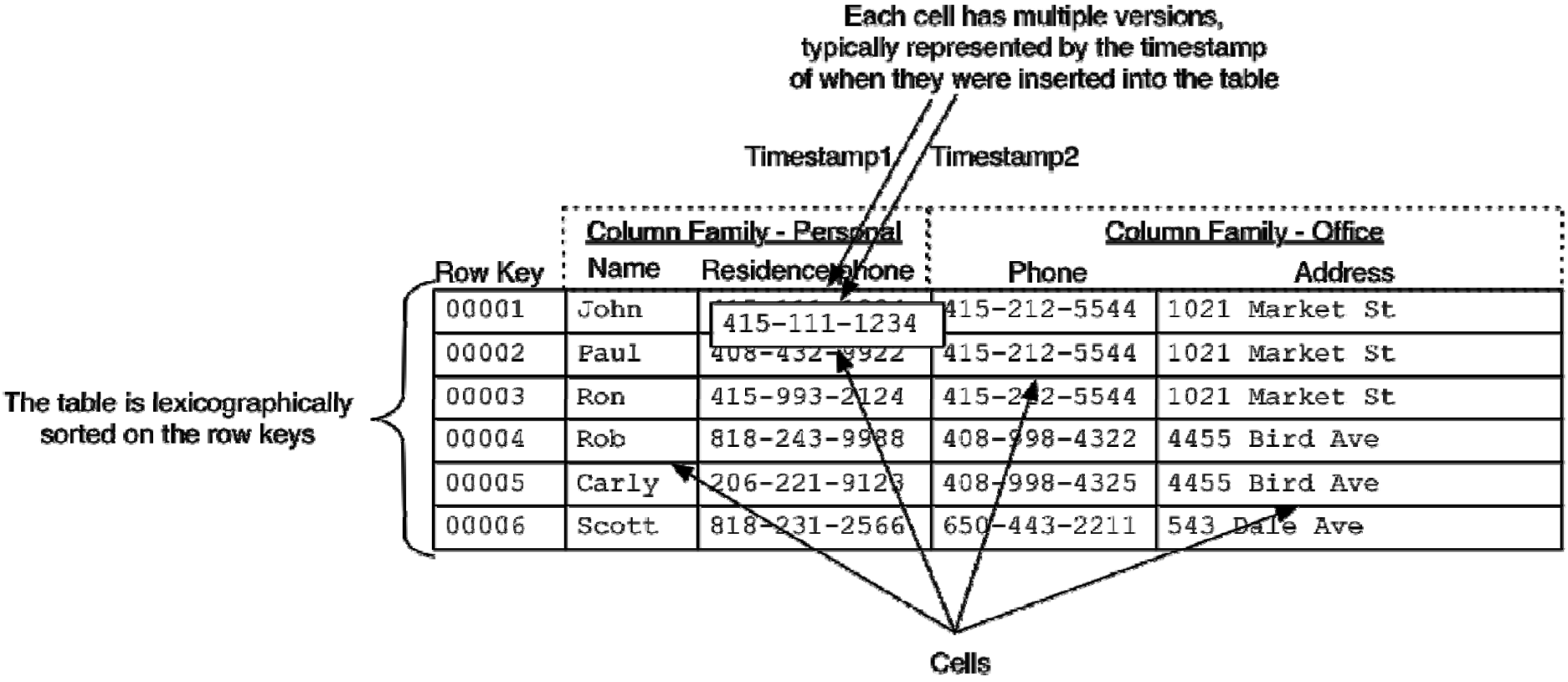
Hbase time-stamp

- **Table**: design-time namespace, has many rows.
- **Row**: atomic key/value container, with one row key
- **Column Family**: groups ‘related columns’ / divide rows into physical files.
- **Column-qualifiers**: a key in the k/v container inside a row
- **Timestamp**: long milliseconds, sorted descending
- **Value**: a time-versioned value in the k/v container inside a row

- → Row is atomic: flushed periodically
  - Colum-Families: groups of columns with ‘related’ data:
    - to store in separate files, do compression, in-memory options, …
- → Family definitions are static, limited to small number of families
- → Column-qualifiers may vary from row to row.

From “Hbase Schema Design” by Ian Varley, HBaseCon 2012

Hbase sumary



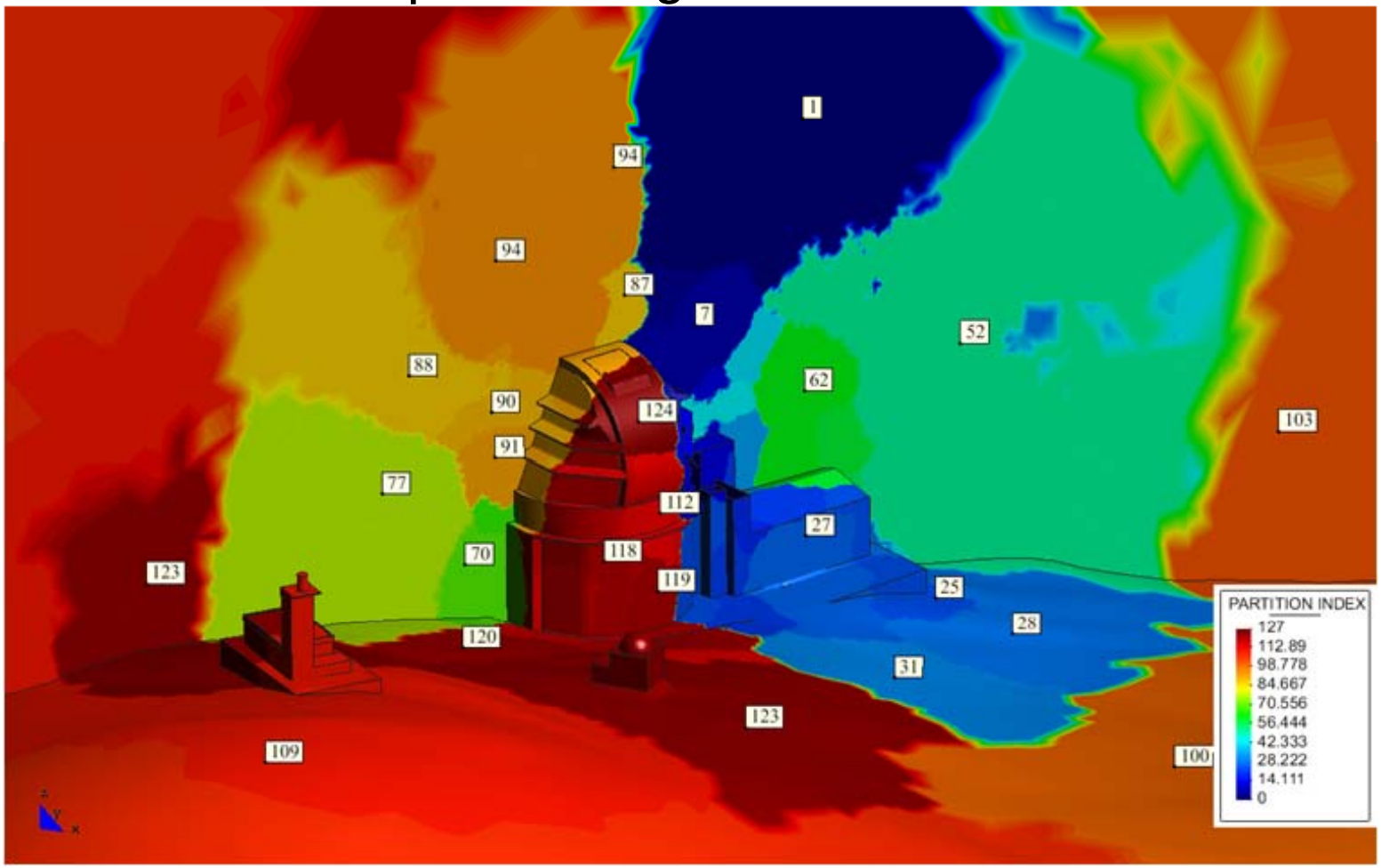
From "Introduction to Hbase Schema Design" by Amandeep Khurana

Hbase summary

Row Key	CF: Personal	CF: Office
00001	Name:t1 = John Residence_phone:t1 = 415-111-1224 Residence_phone:t2 = 415-111-1234	Phone:t1 = 415-212-5544 Address:t1 = 1012 Market St
00002	Name:t10 = Paul Residence_phone:t10 = 408-432-9922	Phone:t11 = 415-212-5544 Address:t11 = 1012 Market St
…	…	…
…	…	…
0000N	Name:t1234 = Eventually Residence_phone:t1234 = 555-111-111 Residence_phone_alternative:t1234 = 666-666-666	

Hbase in VELasSCo

- Solver: Domain partitioning for distributed calculation

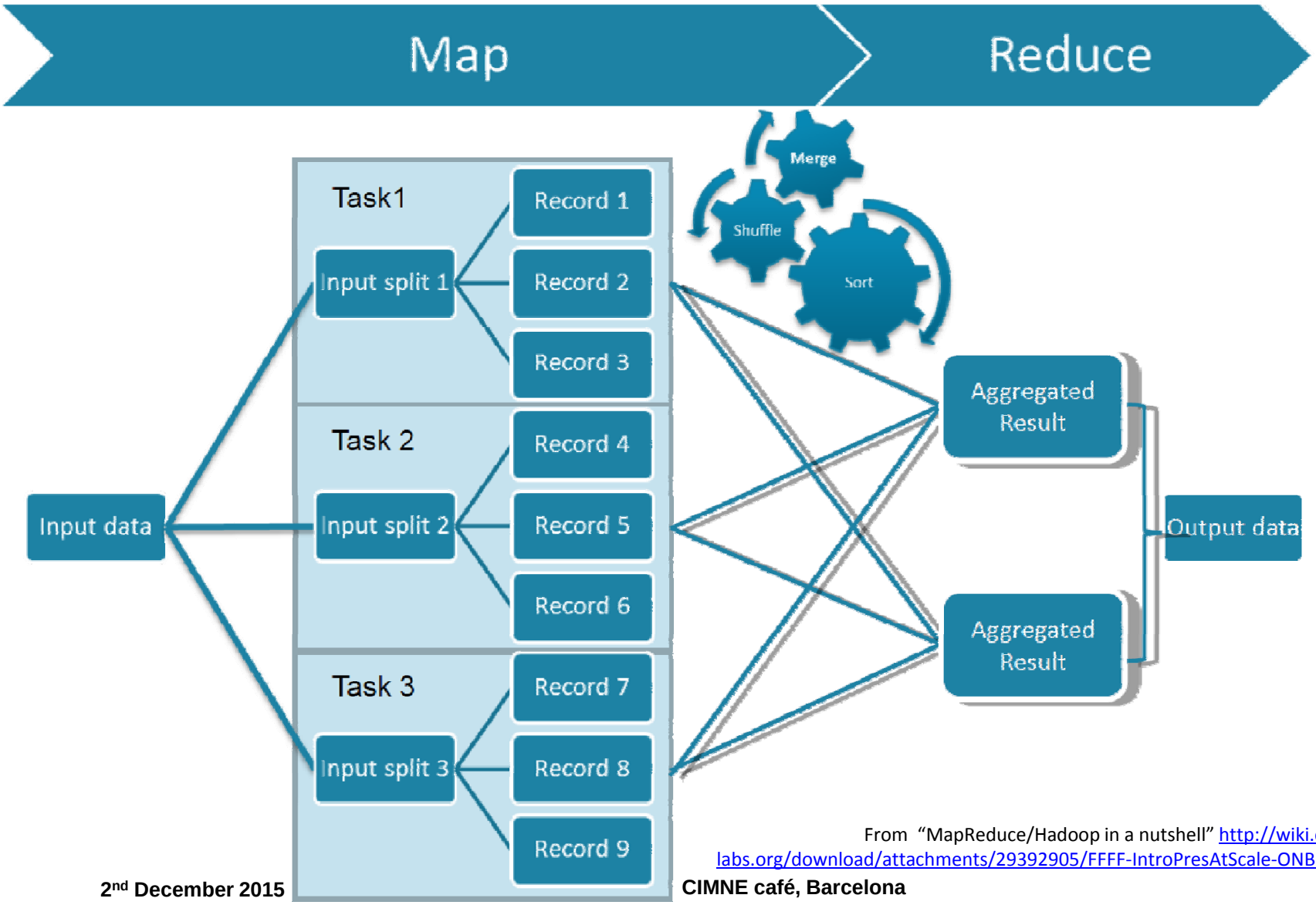


Hbase in VELaSSCo

- 3 tables: model list, metadata and data
- Simulation_Data:
  - Results for 1 time-step and 1 partition in one row
  - 2 column families: M(esh) and R(esults)
  - Each coordinate or result value in one column qualifier

Key: SimulationId + *Analysis* + *Step* + *PartitionID*	CF: *M*	CF: *R*
0x0001 + 0 + "" + 0.0 + 0	c000001_0 = (0.0, 0.0, 0.0), c000001_339185 = (1.0, 1.0, 1.0), m000001cn_0 = (1, 2, 3, 4), m000001gr_0 = (1), m000001cn_717793 = (339184, 339183, 339182, 339181), m000001gr_717793 = (2), m000002cn_717794 = (1, 2, 3) m000002cn_739513 = (7,66, 8)	(empty)
0x0001 + 6 + "RANSOL" + 91.5 + 0	(empty)	r000001vl_1 = (4), r000001vl_339184 = (324), r000002vl_1 = (1.1, 2.2, 3.3), r000002vl_339184 = (10.0, 10.0, 9.1), …
0x0001 + 6 +"RANSON" + 200.0 + 127	(empty)	…

MapReduce



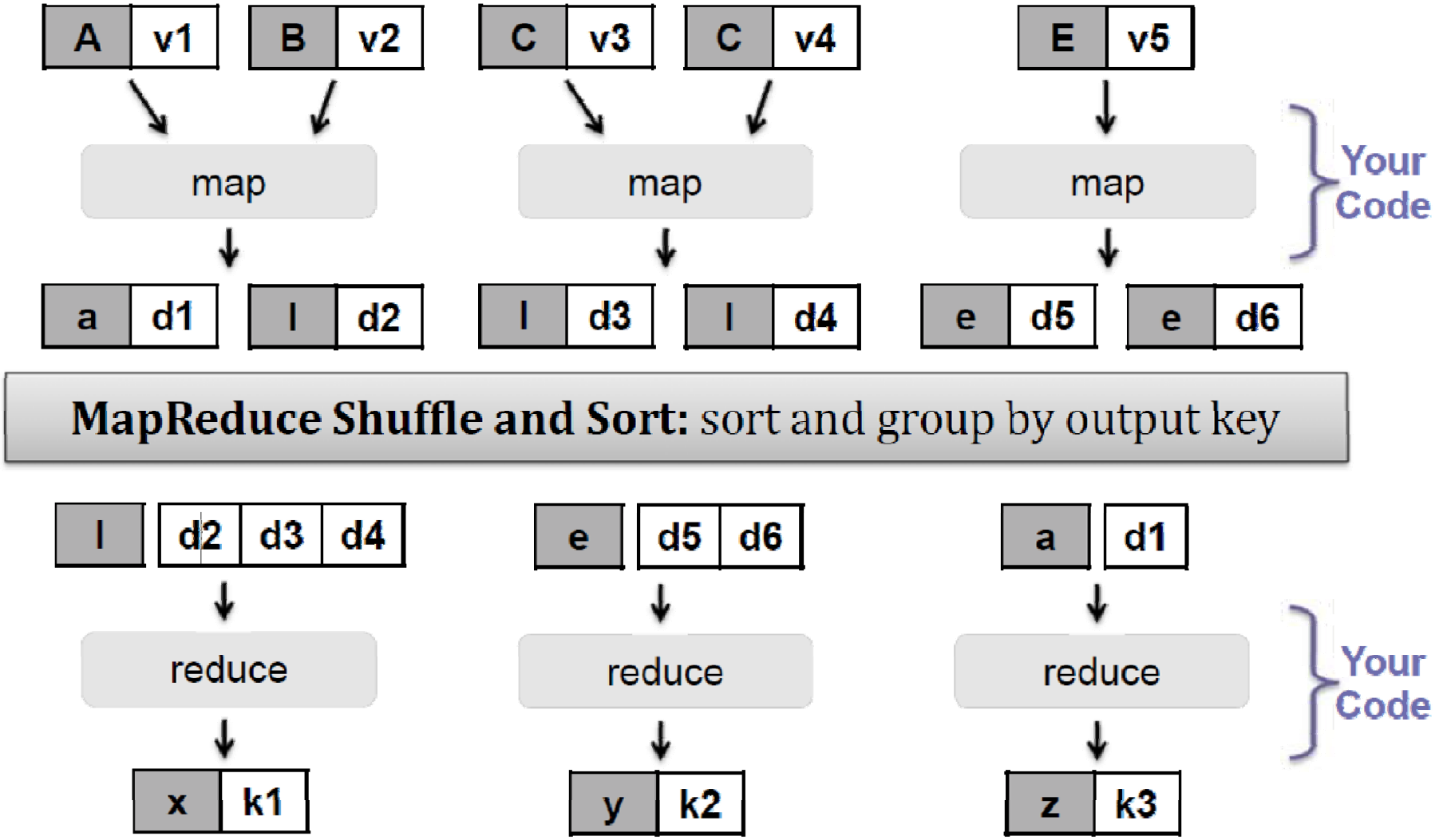
From "MapReduce/Hadoop in a nutshell" http://wiki.opf-labs.org/download/attachments/29392905/FFFF-IntroPresAtScale-ONB.pdf

MapReduce

- Key-value Input/Output
- Uses map and reduce functions:
  - Map: (k1, v1) → list (k2, v2)
  - Reduce: (k2, list(v2)) → list(k3, v3)
  - Map function applied to every input k-v pair
  - Map function generates intermediate k2-v2 pairs
  - Intermediate results are sorted and grouped by key
  - Reduce applied to sorted and grouped intermediate results
  - Reduce emits result key-value s

From "MapReduce overview" http://courses.coreservlets.com/Course-Materials/pdf/hadoop/04-MapRed-1-OverviewAndInstall.pdf

MapReduce



From "MapReduce overview" http://courses.coreservlets.com/Course-Materials/pdf/hadoop/04-MapRed-1-OverviewAndInstall.pdf

MapReduce: color count with Java API

```java
package de.inovex.academy.hadoop.candy;
import org.apache.hadoop...;

public class ColorMapper // extends Mapper< k_i, v_i, k_t, v_t>
       extends Mapper<LongWritable, Text, Text, IntWritable> {
       private static IntWritable one = new IntWritable( 1 );
       /**
        * in: ( byte-pos, "color flavor size batch bag" )
        * out: ( color, 1 )
        */
       @Override
       protected void map( LongWritable key_in, Text va_in,
                           Context context )
                     throws IOException, InterruptedException {
              String[] fields = va_in.toString().split( "\\t" );
              context.write( new Text( fields[0] ), one );
       }
}

public class IntSumReducer // ext. Reducer< k_t, v_t, k_o, v_o>
       extends Reducer<Text, IntWritable, Text, IntWritable> {

       @Override
       protected void reduce( Text key_in, Iterable<IntWritable>
                                  values_in, Context context )
                     throws IOException, InterruptedException {
              int sum = 0;
              for ( IntWritable partialCount : values_in ) {
                     sum += partialCount.get();
              }
              context.write( key_in, new IntWritable( sum ));
       }
}
```

```java
package de.inovex.academy.hadoop.candy;
import org.apache.hadoop...;

public class ColorCount
       extends Configured implements Tool {
       @Override
       public int run(String[] args) throws Exception {
              Job job = new Job();
              job.setJarByClass( ColorCount.class );
              job.setJobName( "color-count" );

              FileInputFormat.addInputPath( job,
                                           new Path( args[0] ));
              FileOutputFormat.setOutputPath( job,
                                           new Path( args[1] ));

              job.setMapperClass( ColorMapper.class );
              job.setReducerClass( IntSumReducer.class );
              job.setNumReduceTasks( 6 ); // 6 different colors

              job.setOutputKeyClass( Text.class );
              job.setOutputValueClass( IntWritable.class );

              return job.waitForCompletion( true ) ? 0 : 1;
       }

       public static void main( String[] args )
                     throws Exception {
              int exitCode = ToolRunner.run( new ColorCount(),
                                                  args);
              System.exit(exitCode);
       }
}
```

YARN

- Thrift API

Others

- Hive, FLUME, …

Distributed post-processing: VELaSSCo

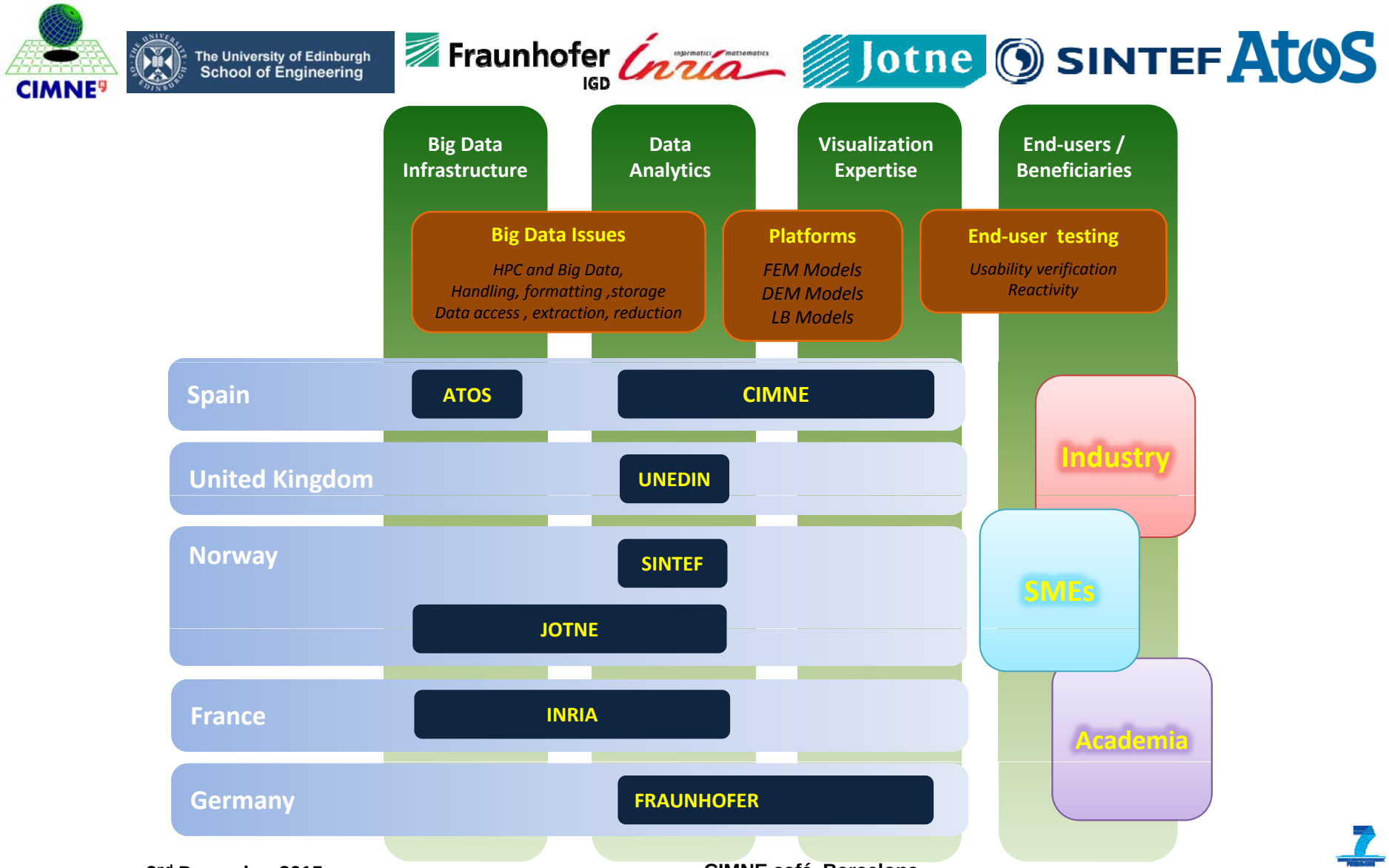
- **V**isual Analysis for **E**xtremely **La**rge-**S**cale **S**cientific **Co**mputing
  - grant agreement # 619439 (FP7/2007-2013)
  - 3 Years: January 2014 – December 2016
  - collaborates with HPC projects: **NUMEXAS**, **Fortissimo** and **CloudFlow**
- Integrate post-process analytics in a Big Data framework embedded in the HPC, where the data is being calculated.
- http://www.velassco.eu

VELaSSCo

European Union

SEVENTH FRAMEWORK PROGRAMME

Partners


Big Data Infrastructure
Data Analytics
Visualization Expertise
End-users / Beneficiaries
Big Data Issues
HPC and Big Data,
Handling, formatting ,storage
Data access , extraction, reduction
Platforms
FEM Models
DEM Models
LB Models
End-user testing
Usability verification
Reactivity
Spain
ATOS
CIMNE
United Kingdom
UNEDIN
Norway
SINTEF
JOTNE
France
INRIA
Germany
FRAUNHOFER
Industry
SMEs
Academia

Partners

- CIMNE: FEM data analytics
- UEDIN: DEM data analytics
- SINTEF: high degree spline representation
- FRAUNHOFER: GPU leverage
- INRIA: Big Data framework
- JOTNE: Big Data framework + STEP format
- ATOS: Big Data framework, user efectiveness

VELaSSCo

Thanks for your attention

… questions & comments …