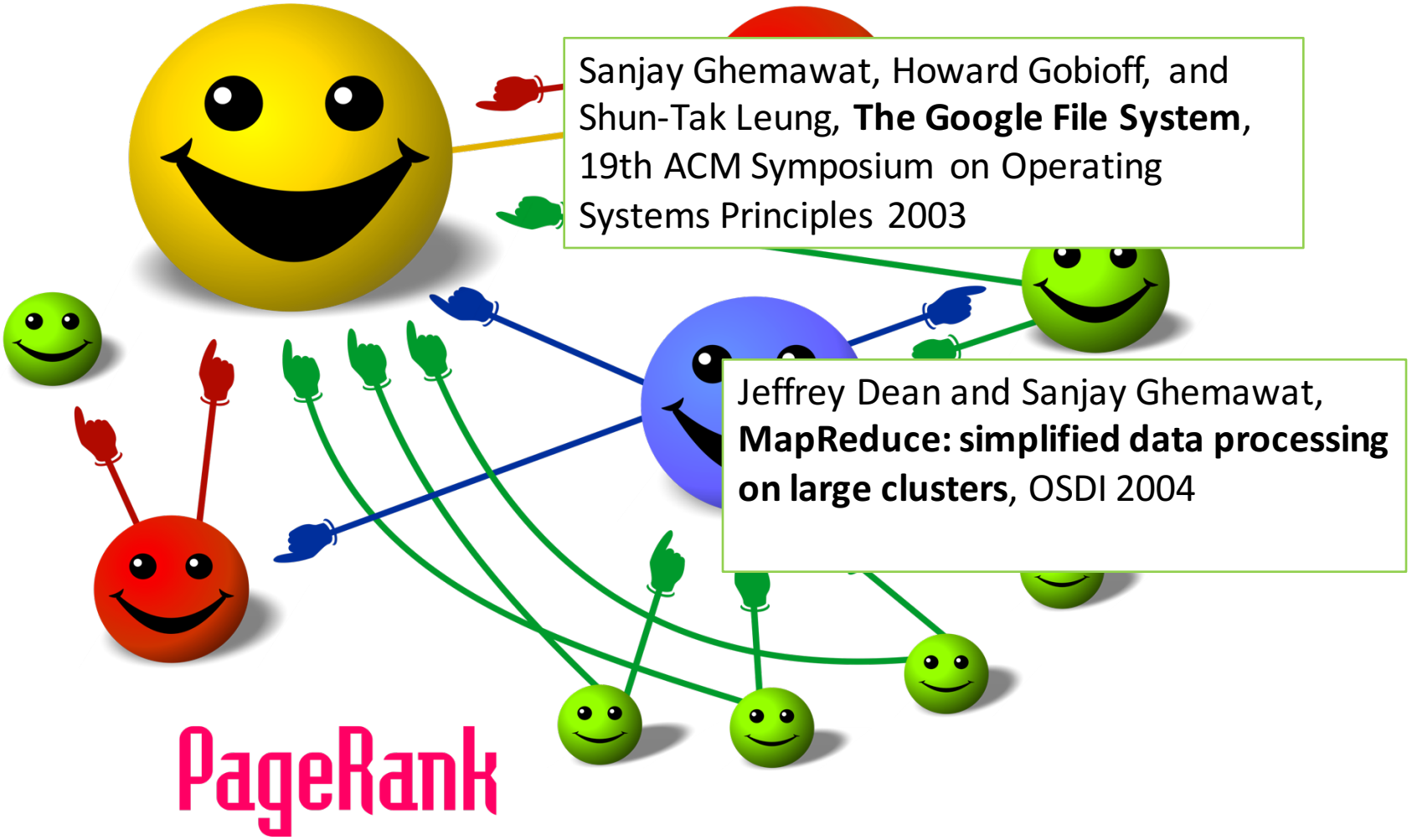
From Hadoop to Spark

Evolution of Big Data computing

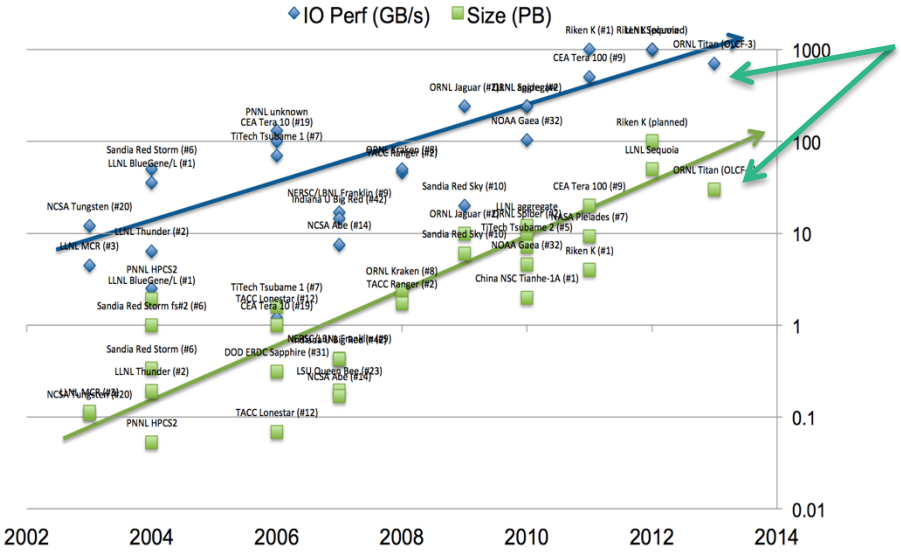
Piotr Szul | Senior Engineer Data61

April 2016

Search in Googles



Big Data vs HPC



Source: "Rock-Hard Lustre: Trends in Scalability and Quality". Nathan Rutman, Xyratex. Retrieved 2012-02-02.

Hadoop Clusters - ???

- May 2010: Facebook 2000 nodes – 21PB
- Jun 2012: Facebook 100PB
- Nov 2013: largest clusters grew from 3500 nodes to 5000
- Nodes fatter: 2x8 core CPU, 12-16x 3TB HD

Titan (Oak Ridge National Laboratory)

- 18,688 nodes
- **~299k CPU cores** + ~ 50M CUDA cores
- ~20 PetaFlops
- *In April 2013 "Herculean storage upgrade" from 10PB and 250Gb/s to*:**40PB and 1.4 TB/s**
- 20000 disks
- 36 x Datadirect Networks' SFA12K-40
- Each: 1.12PB and 40GB/s

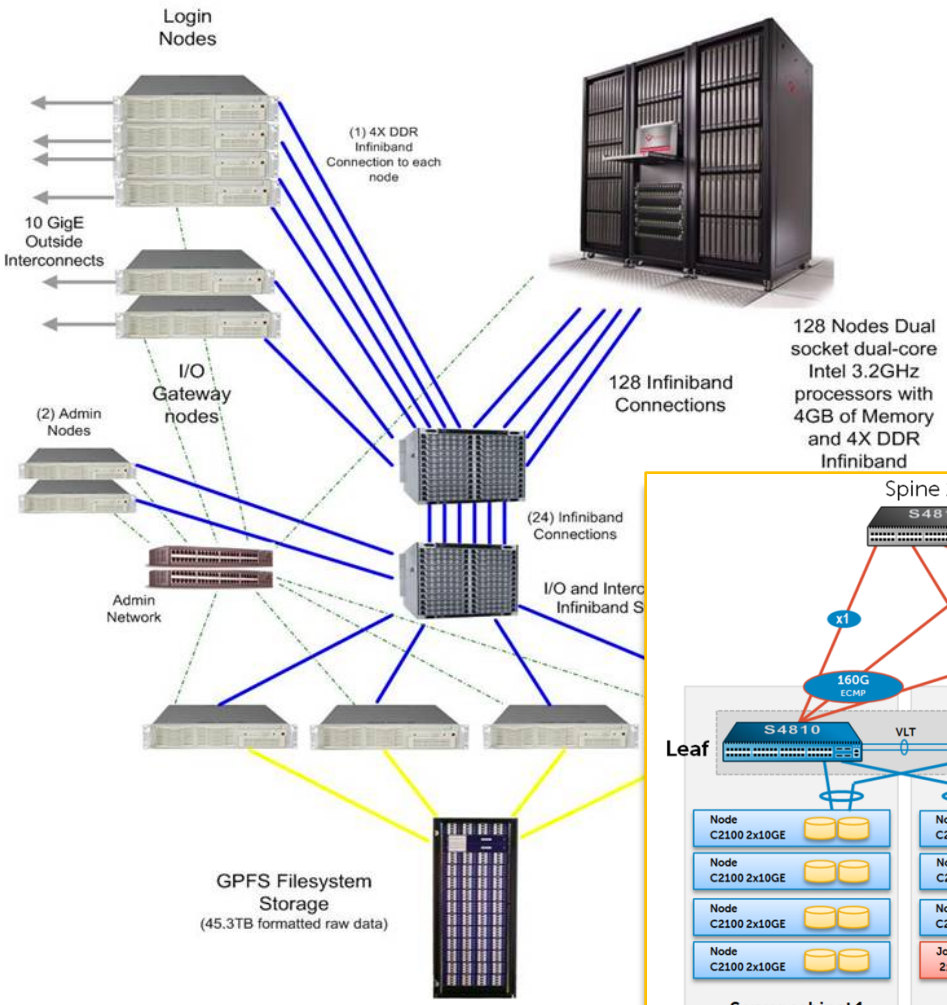
Hypothetical Hadoop Cluster (5000 nodes)
Each node:

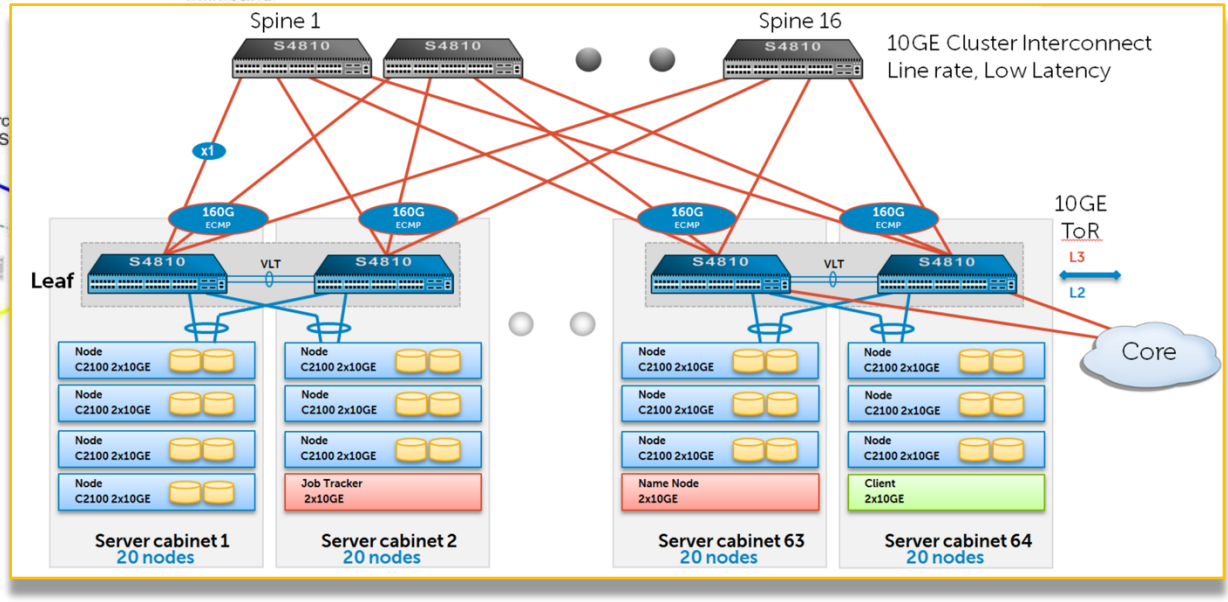
- 12 x 3TB HD and 16 x Intel E5 cores
- 12 TB storage (HDFS x3) and 2.4GB/s read throughput + 1.6 TeraFlops

Total:

- **60 PB of storage and 11TB/s (60000 HD)**
- **80k CPU cores**

Big Data vs HPC





Big Data vs HPC

Big Data: Petabytes

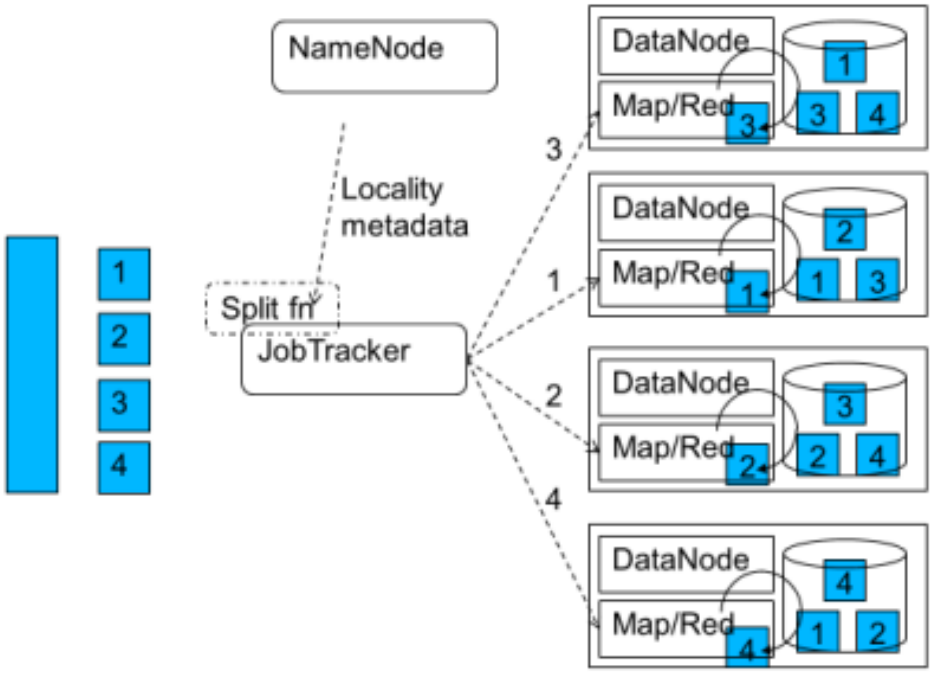
- Storage of low-value data
- H/W failure common
- Code: frequency, graphs, machine-learning, rendering
- Ingress/egress problems
- Dense storage of data
- Mix CPU and data
- Spindle:core ratio

HPC: Petaflops

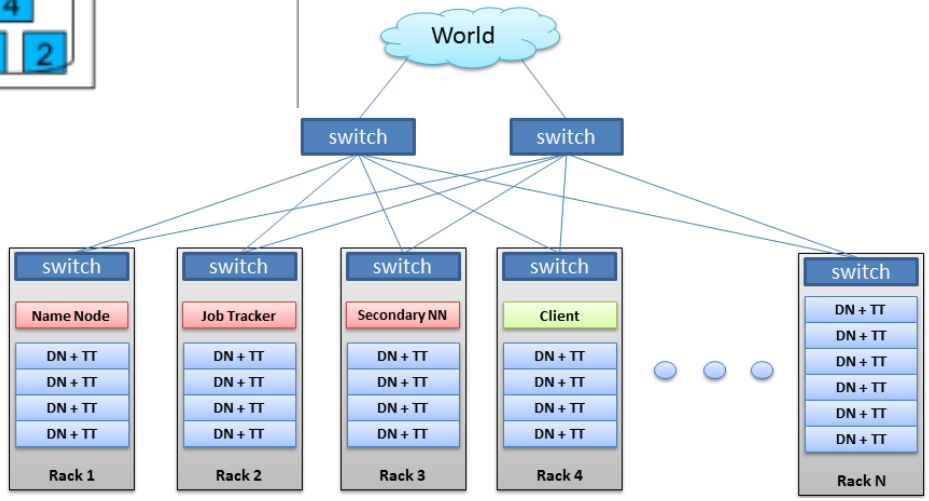
- Storage for checkpointing
- Surprised by H/W failure
- Code: simulation, rendering
- Less persistent data, ingress & egress
- Dense compute
- CPU + GPU
- Bandwidth to other servers

- *Failure is inevitable*
- *Bandwidth and IO is precious*
- *Linear scalability*
- *Hide the complexities from developers*

Distributed Filesystem

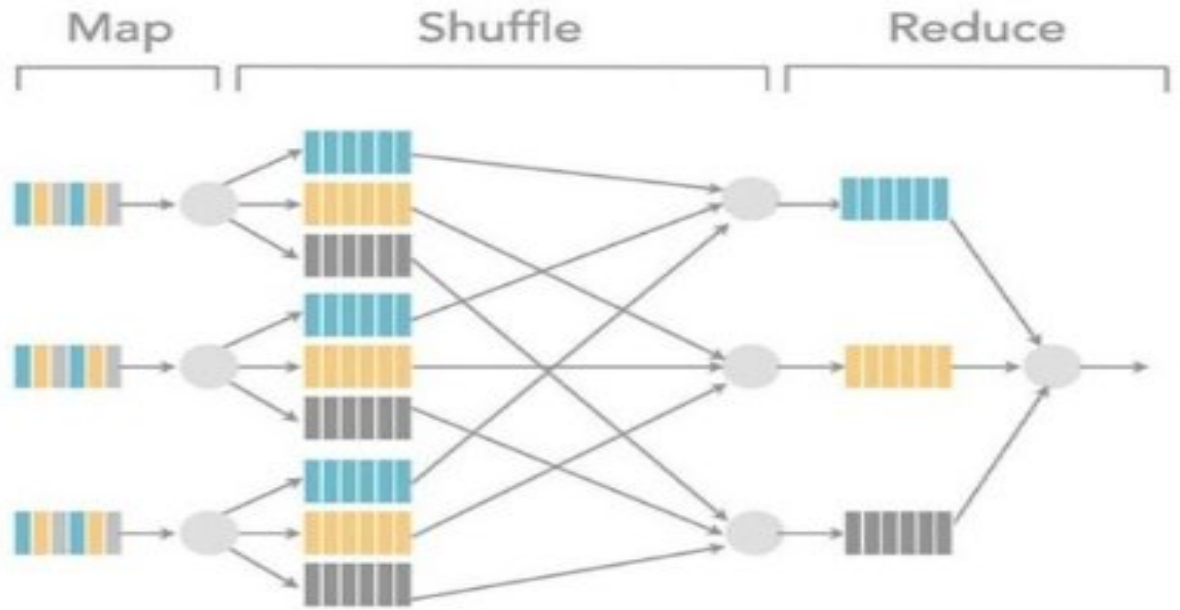


Reproducibility?



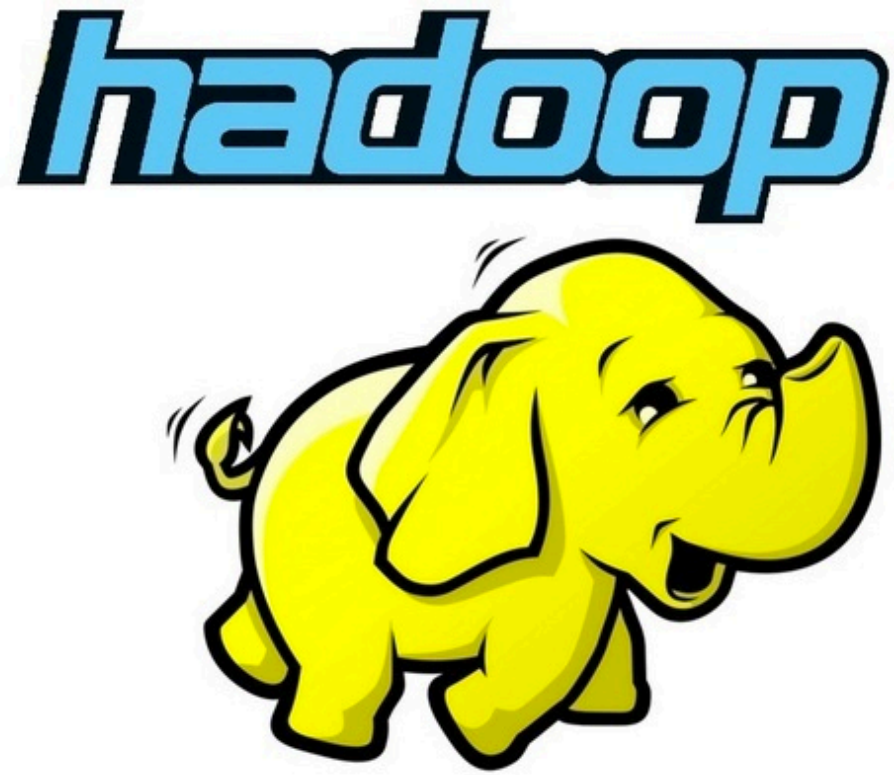
- *Linear scalabiliy → no central bottlenecks*
- *Bandwidth and IO is precious → topology aware scheduling*
- *Failure is inevitable → replication*

Map Reduce as Execution Paradigm



- *Linear scalability → massive parallelisation, minimal communication*
- *Failure is inevitable → fault tolerance build-in*
- *Hide the complexities from developers → simple programming model*

Map Reduce for everybody!

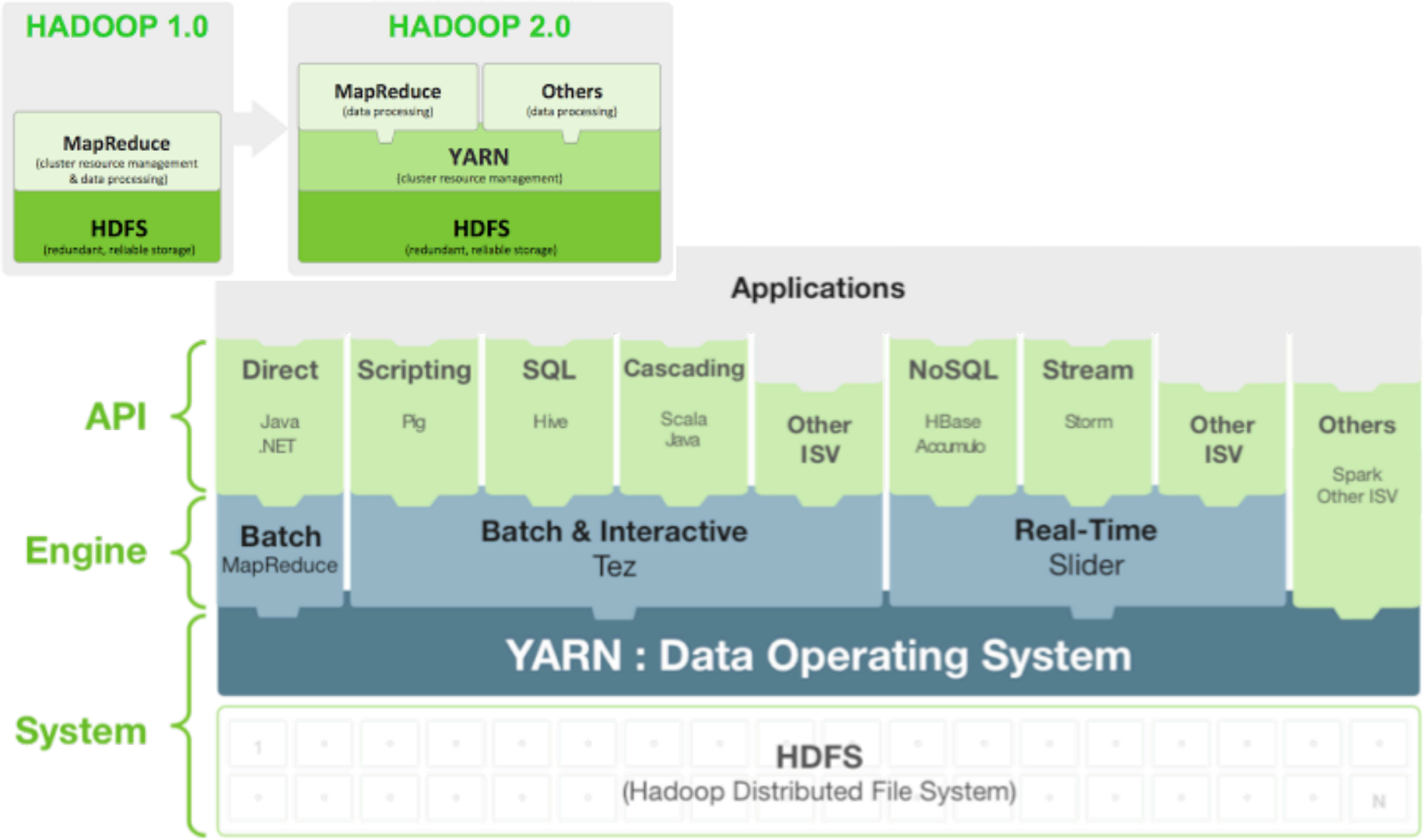


In early 2006 Doug Cutting creates the Hadoop project.
Hadoop 0.1.0 released in April 2006

Why Hadoop Was Not Enough?

- Why Hadoop was great and why we need another solution?
- Good
  - Finally analytics at scale
  - Fault tolerance etc.
- Bad
  - Foreign and low level programming model
  - Rigid data structure
  - Hard to support multistep processes
  - Everything goes to disk
  - Slow scheduling
  - Constrained execution model

Hadoop 1.0 → YARN



Why do we need another Hadoop?

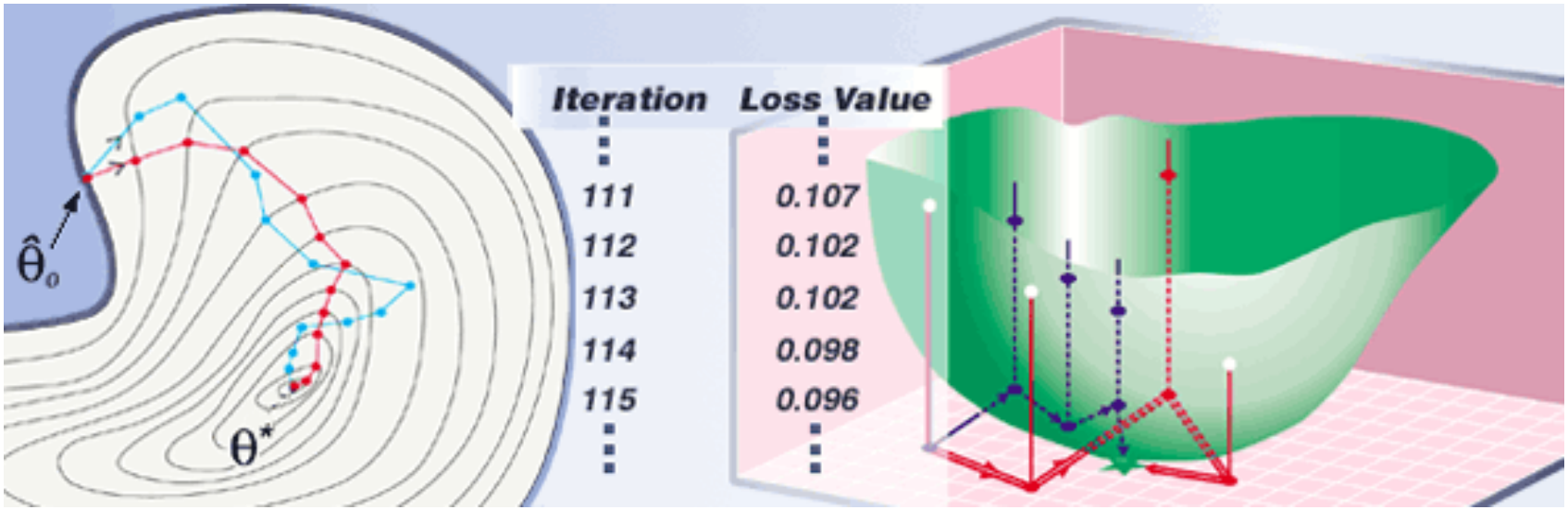
```java
public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

≠

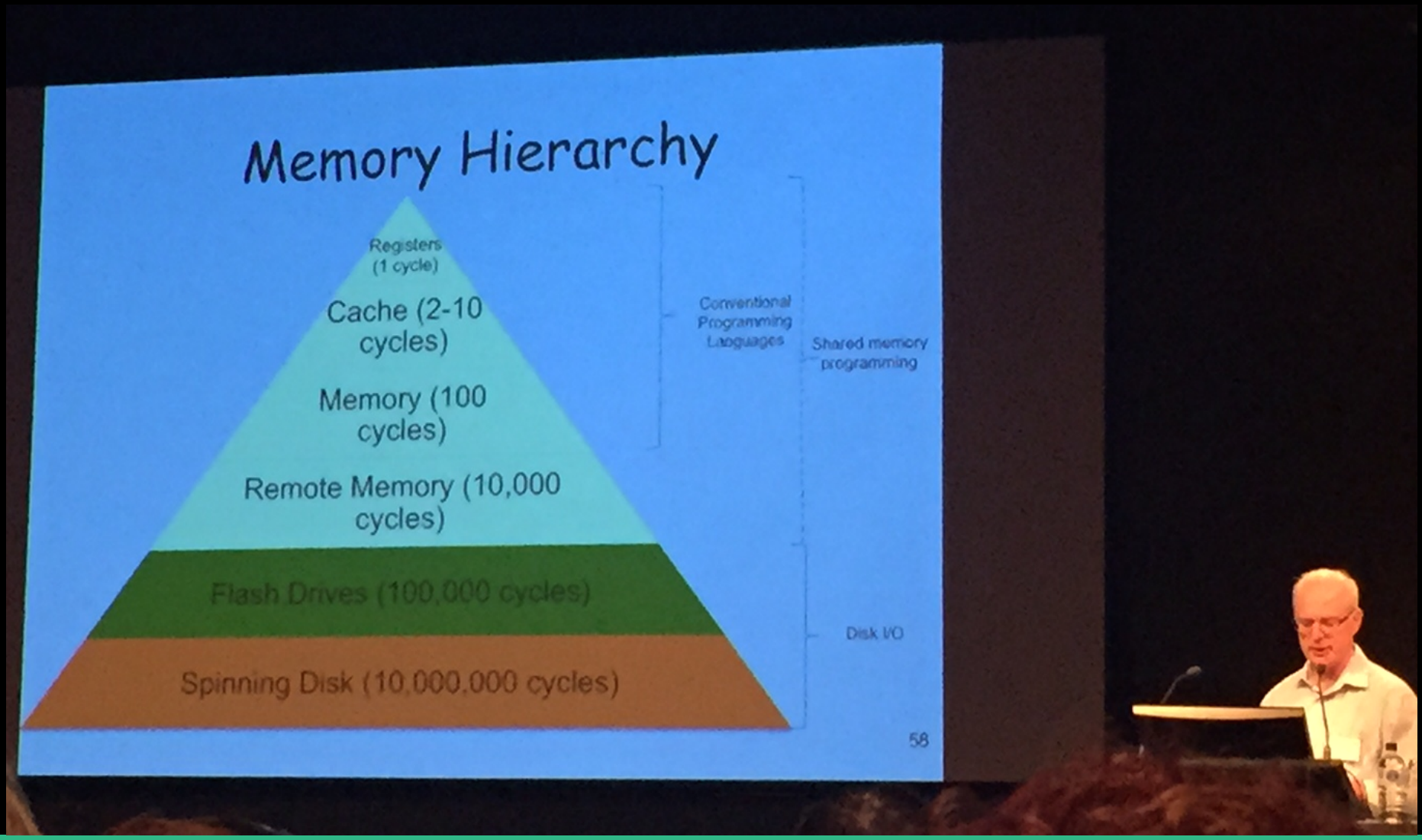
```
1  SELECT word,COUNT(*) FROM words GROUP BY word
```

Why do we need another Hadoop?

Realisation of potential in big data machine learning and predictive analytics.
More data (usually) better predictions with basic models.



Quintessential challenge of CS

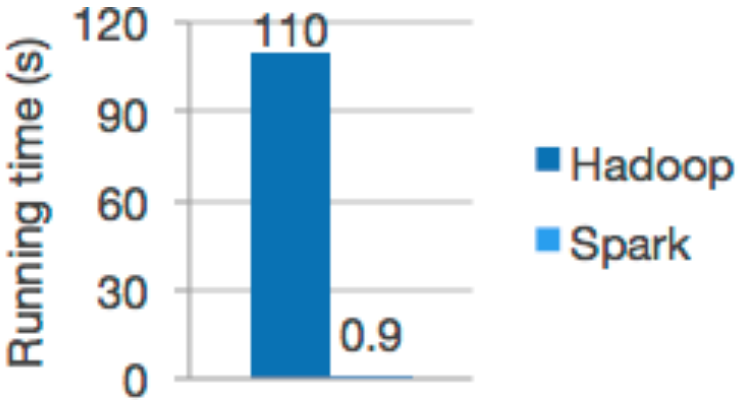


From Keynote: David Abramson, Research Computing Centre, QT

What is [Apache] Spark?

Fast, expressive cluster computing system compatible with Apache Hadoop

- Improves **efficiency** through:
  - In-memory computing primitives
  - General computation graphs
- Improves **usability** through:
  - Rich APIs in Java, Scala, Python,R
  - Interactive shell Scala and Python
  - Can work with iPython notebook

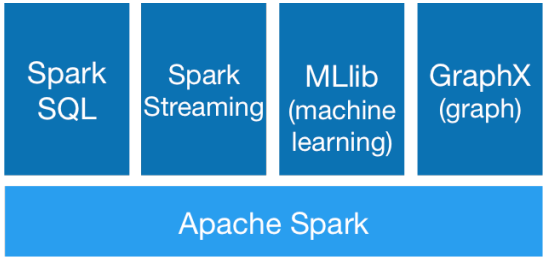


→ Often 2-10× less code

From "Parallel Programming with Spark" by Matei Zaharia, UC Berkeley

Why Spark?

- Generality
  - Combine SQL, streaming, and complex analytics.
  - Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.



- Runs Everywhere
  - Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.
  - You can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, or on Apache Mesos. Access data in HDFS, Cassandra, HBase, Hive, Tachyon, and any Hadoop data source.

Let' talk about loops

```
1  input = ["10 20 50", "hello 10 there,", "it's 11 pm"]
2  output = []
3
4  for s in input:
5      output.append(s.split(" ")) # [ ["10", "20", "50"], [ "hello", "10", "there"], ...]
6
7  input = output
8  output = []
9
10 for l in input:
11     for s in l:
12         output.append(s)        # [ "10", "20", "50", "hello", "10", "there", ...]
13 ...
14
15 for s in input:
16     if is_int(s):
17         output.append(s)        # [ "10", "20", "50", "10", "11" ]
18
19 ...
20
21 for s in input:
22     output.append(int(s))       # [ 10, 20, 50, 10, 11 ]
23
24 ...
25
26 result = 0
27 for i in input:
28     result += i                 # 101
```

Map

Flatten

Filter

Fold

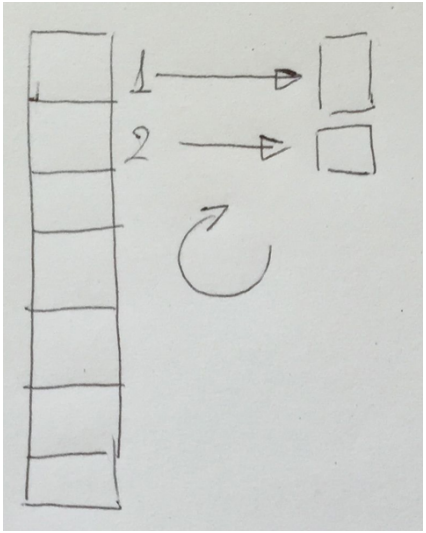
The other “functional” way

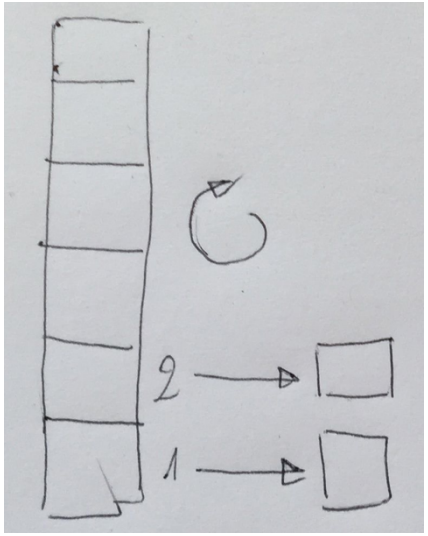

```
1  val input = List("10 20 50", "hello 10 there,", "it's 11 pm")
2
3  val result = input.map( { s => s.split("") } ) // same as: lambda s: s.split("")
4      .flatten()
5      .filter( s => s.isInt() )
6      .map( s => s.toInt )
7      .fold( (a,b) => a + b)
8
9  // a compacted scala version
10
11 val result = input.flatMap( _.split("") )
12     .filter( _.isInt() )
13     .map( _.toInt )
14     .fold( _+_ )
```

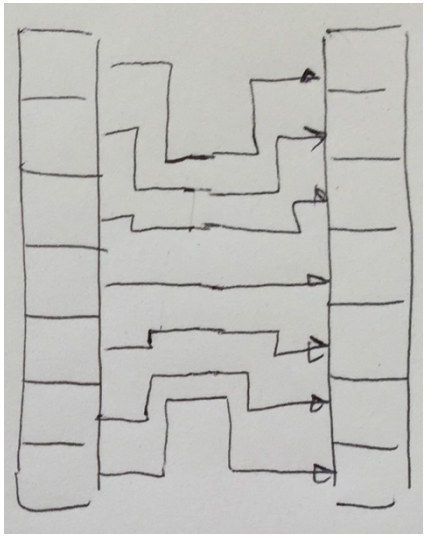
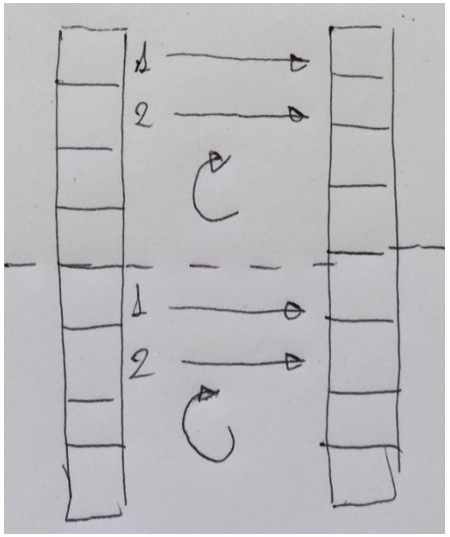
And this is neat, because each operation has explicit semantics, e.g. explicit way it transforms a list.

The less obvious consequence

- **for** loop in an example of the imperative programming (especially the explicit indexed loop)
  - tell the computer how to loop through a collection
- **map, filter etc**., are more "declarative"
  - tell the computer what the result should look like
  - how it loops through it is irrelevant (mostly)



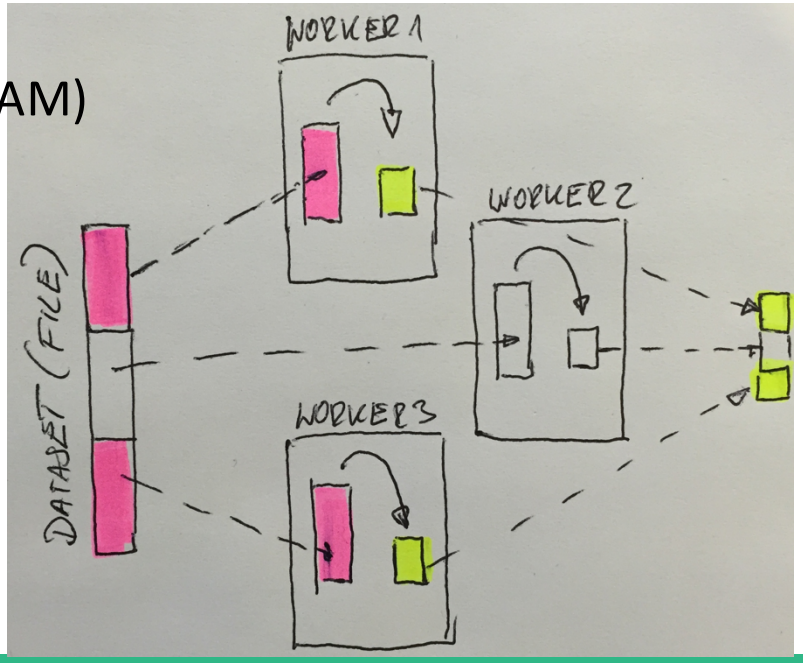


Key Idea

- **Work with distributed collections as you would with local ones**

- Concept: resilient distributed datasets (RDDs)
  - Immutable collections of objects spread across a cluster
  - Built through parallel transformations (map, filter, etc)
  - Automatically rebuilt on failure
  - Controllable persistence (e.g. caching in RAM)



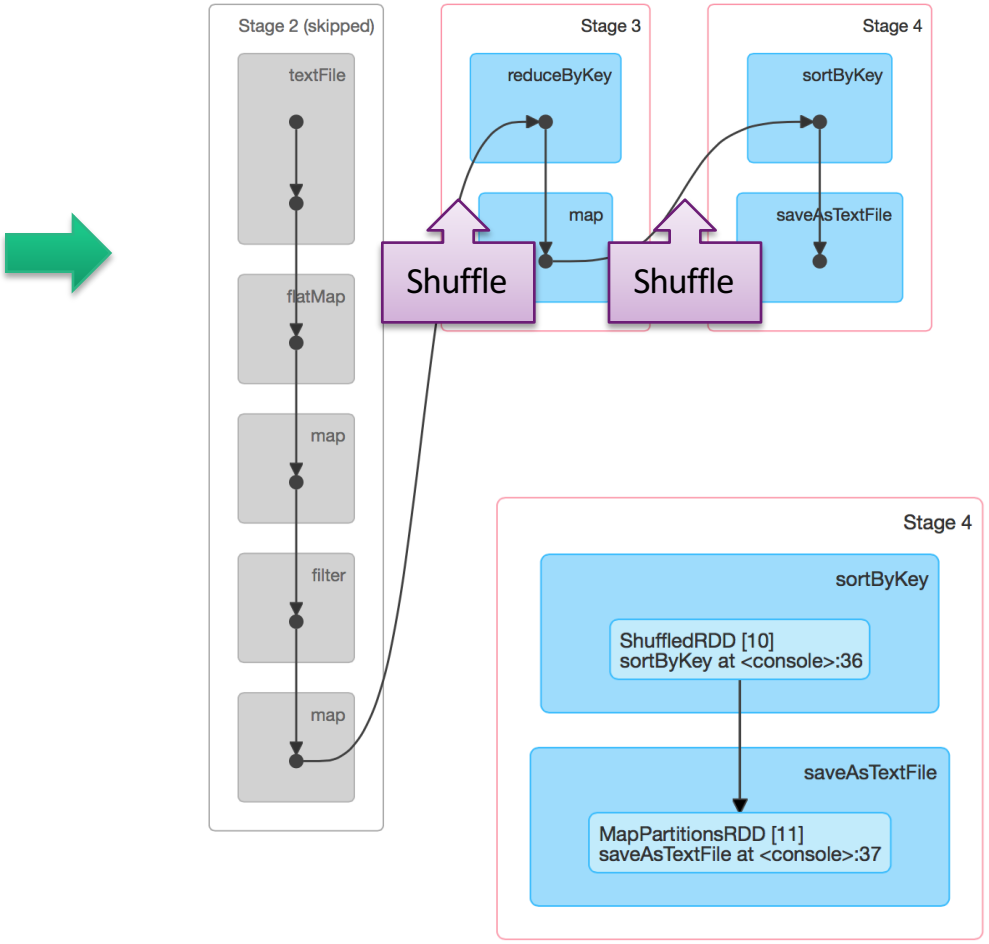
From "Parallel Programming with Spark" by Matei Zaharia, UC Berkeley

Example: Spark Word Count

```
3  import org.apache.spark.SparkConf
4  import org.apache.spark.SparkContext
5  import org.apache.spark.SparkContext._
6
7  object WordCount {
8    def main(args: Array[String]) {
9
10     val sc = new SparkContext(new SparkConf().setAppName("WordCount"))
11     val file = sc.textFile(args(0))
12     file.flatMap(_.split("[^\\w]"))
13       .map(_.toLowerCase().trim())
14       .filter(!_.isEmpty())
15       .map(word => (word, 1))
16       .reduceByKey(_ + _)
17       .coalesce(1,true)
18       .saveAsTextFile(args(1))
19   }
20 }
```

Transformation Graphs

```
3  sc.textFile("../data/input.txt")
4        .flatMap(_.split("[^\\w]"))
5        .map(_.toLowerCase().trim())
6        .filter(!_.isEmpty())
7        .map(word => (word, 1))
8        .reduceByKey(_ + _)
9        .map(_.swap)
10       .sortByKey()
11       .saveAsTextFile("output.txt")
```

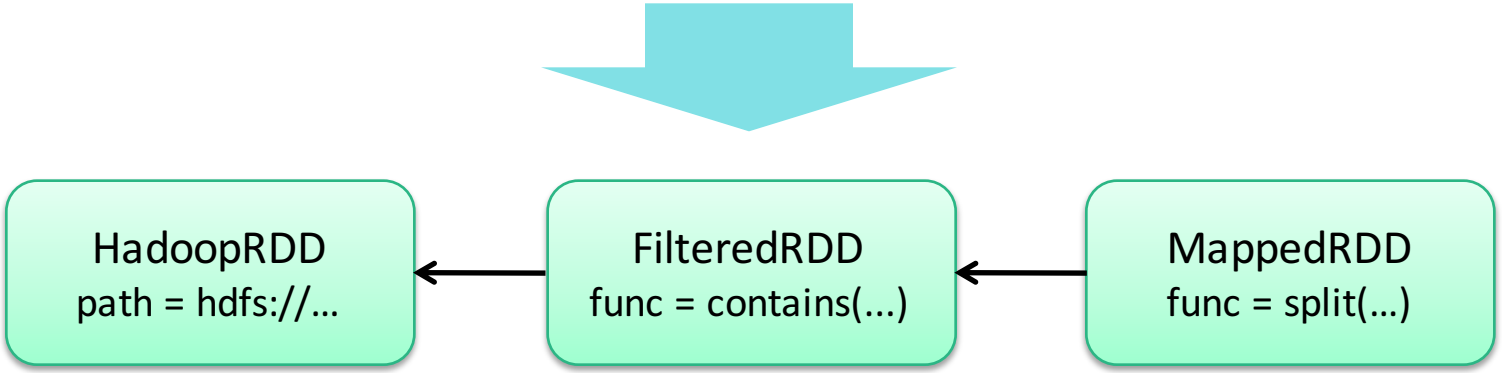


RDD Fault Tolerance

RDDs track the transformations used to build them (their *lineage*) to recompute lost data

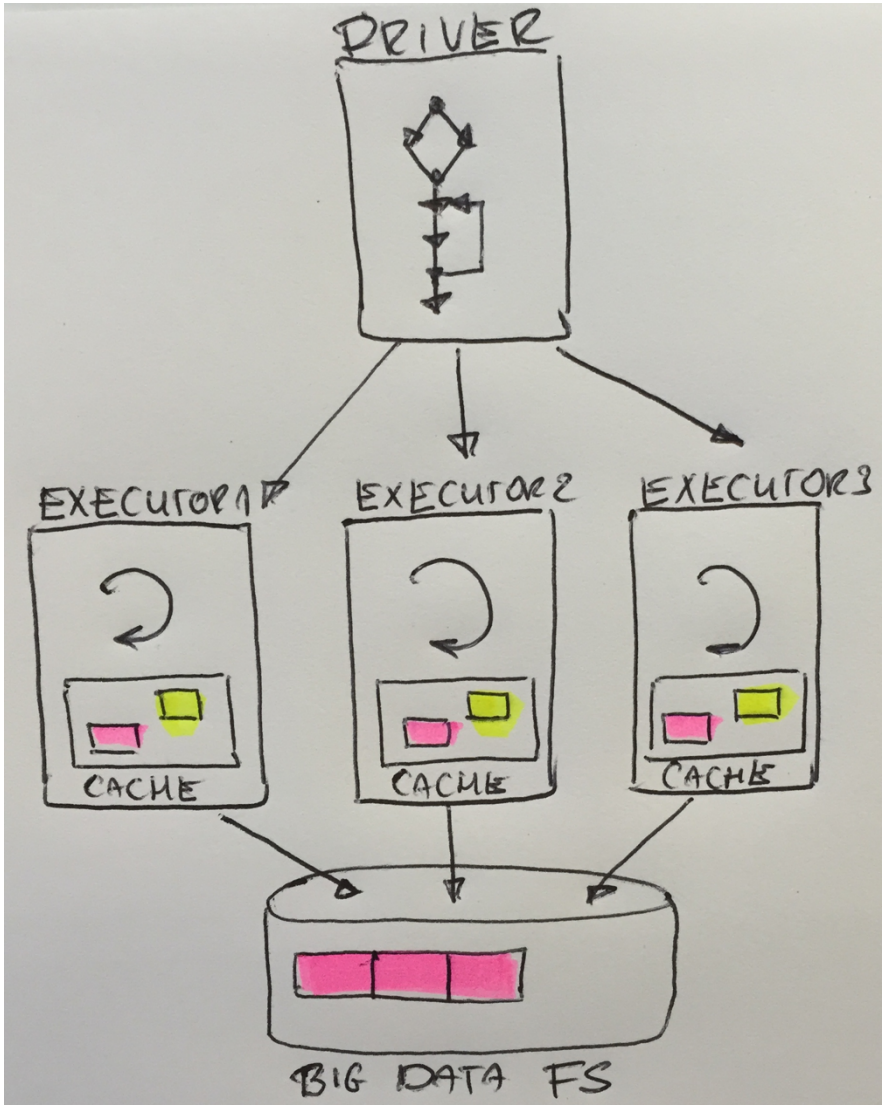
E.g:

```
messages = textFile(...).filter(lambda s: s.contains("ERROR"))
                        .map(lambda s: s.split('\t')[2])
```



From "Parallel Programming with Spark" by Matei Zaharia, UC Berkeley

Anatomy of Spark Runtime

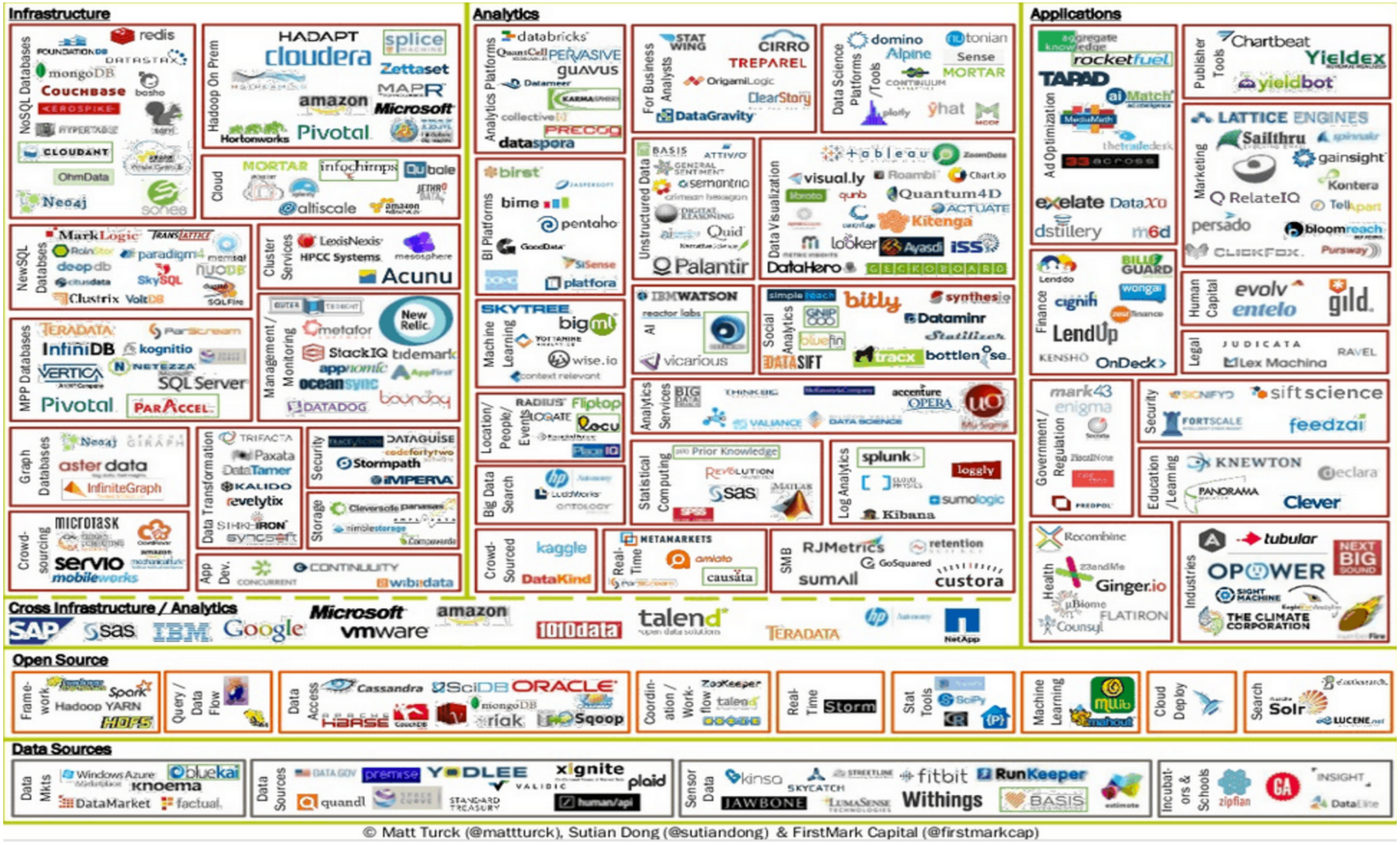


- Driver
  - Creates execution graph
  - Schedules and sends out task to executors
  - Receives and combines results
  - Manages executors
  - Can perform ‘small data’ computation
- Executors
  - Accept and execute computational tasks
  - Read and write big data blocks
  - Maintain in-memory cache of blocks

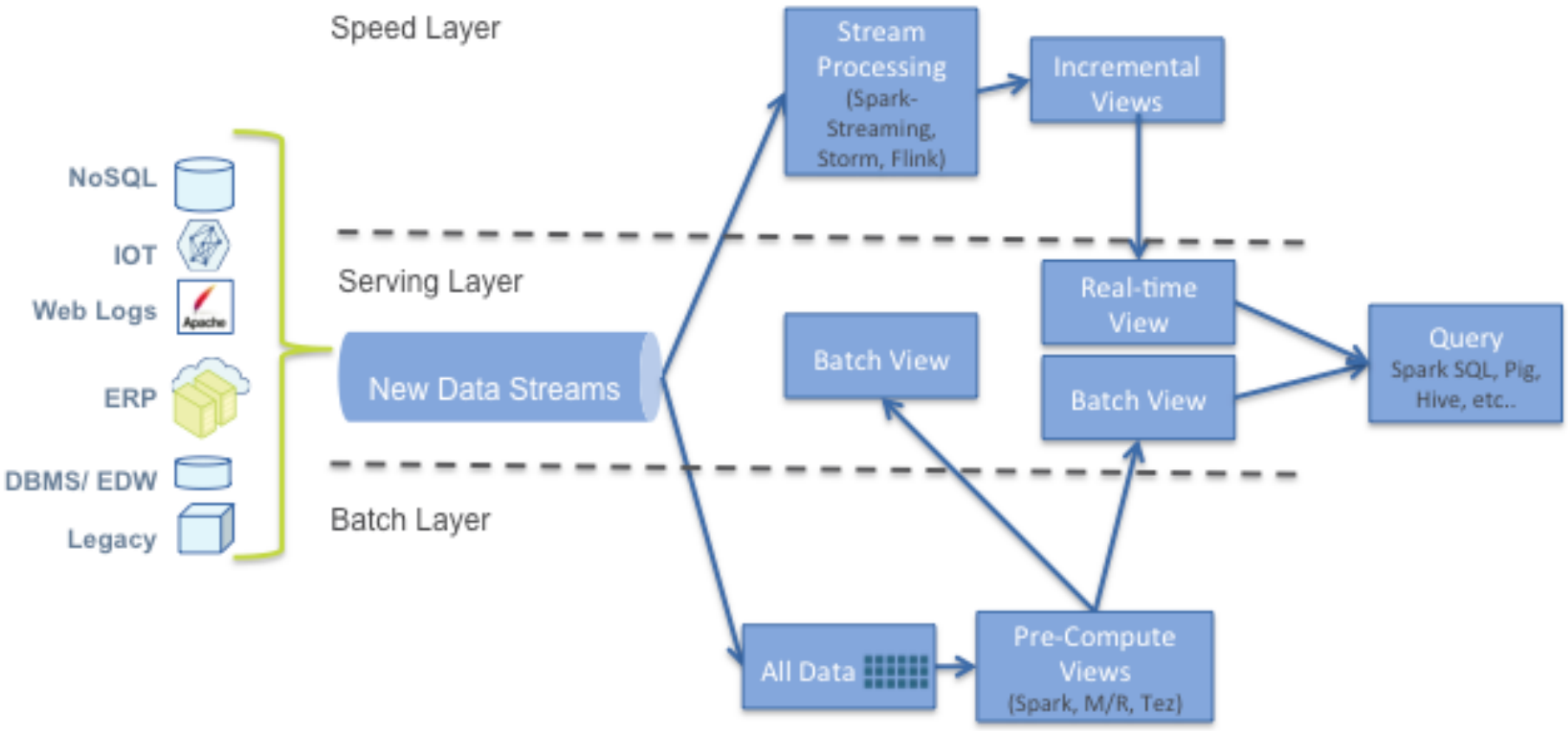
Spark Runtime Environments

- Spark supports multiple runtime environments/deployments including:
  - local: single and multithreaded
  - Standalone cluster
  - YARN: client and cluster
  - Mesos
  - AWS
- Selection of the runtime environment through *--master* parameter or *spark.master* jvm property, e.g.:
  - ***> spark-submit --master local***
  - ***> spark-submit --master local[4]*** # also local[*] use either 4 or all CPU cores
  - ***> spark-submit --master yarn-client --num-executors 100***

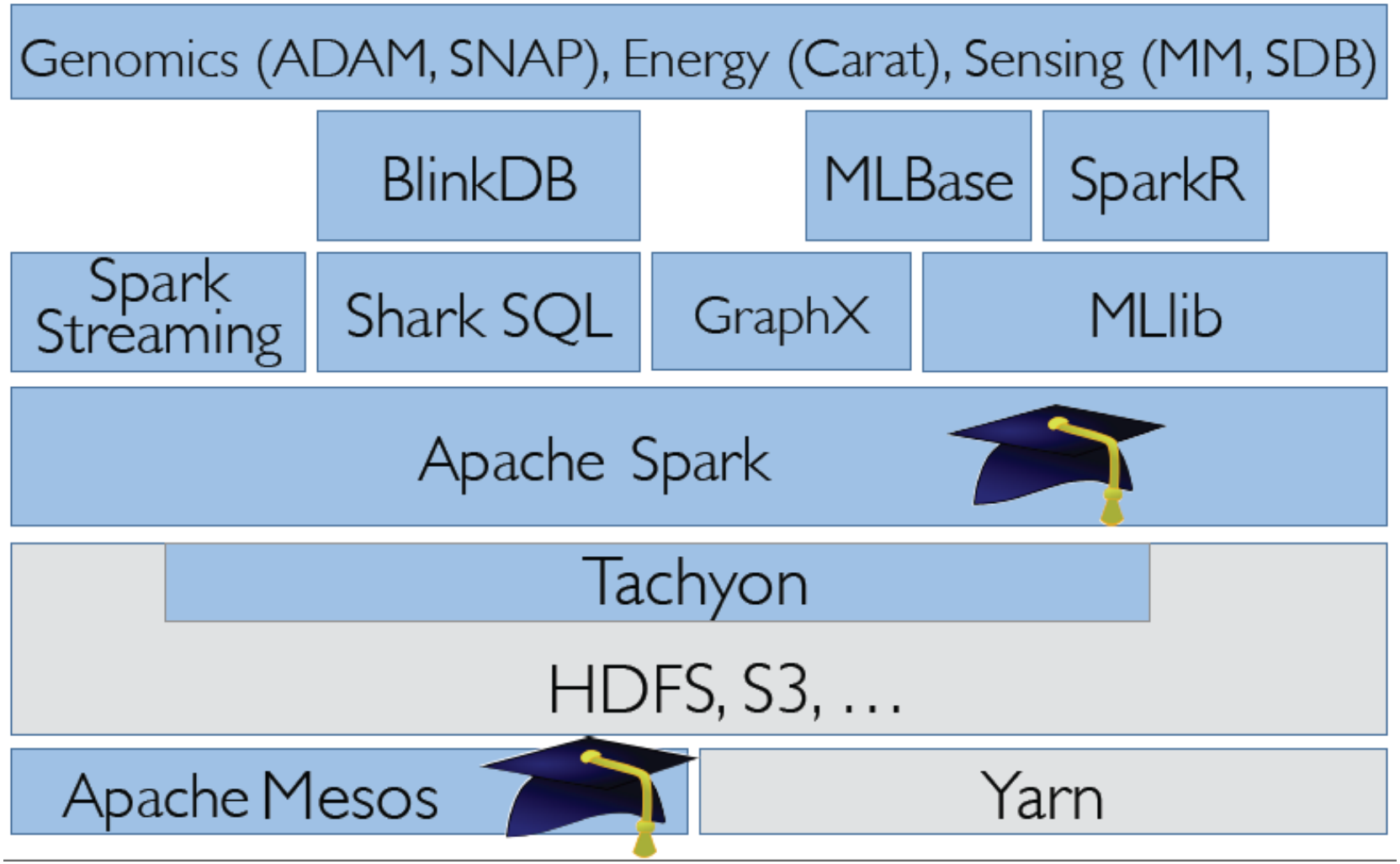
Back to Hadoop Evolution



Lambda Architecture



All in one - BDAS



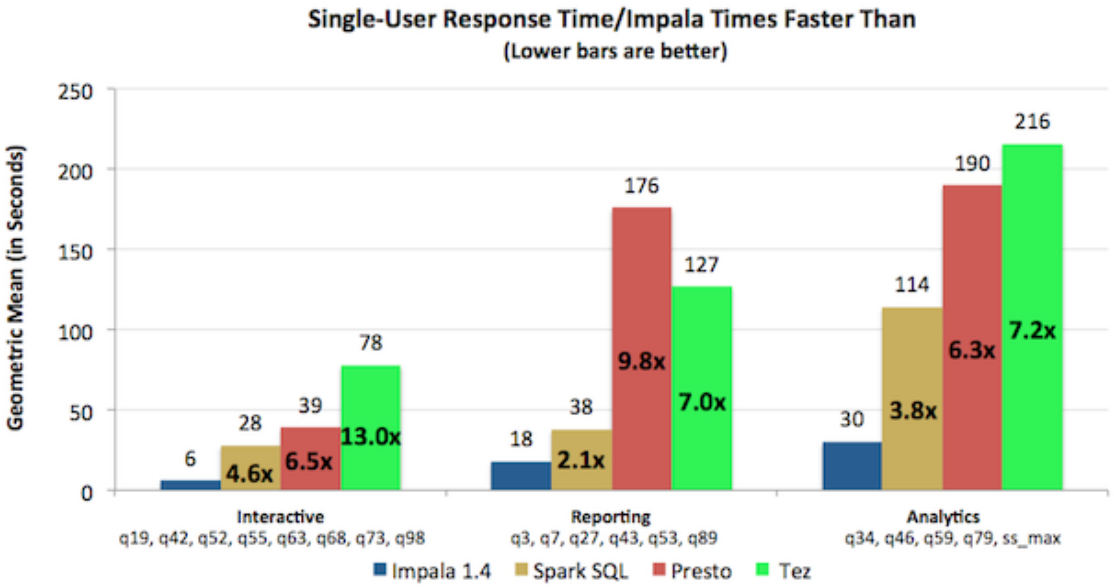
Apache Hive – Big Data Warehouse

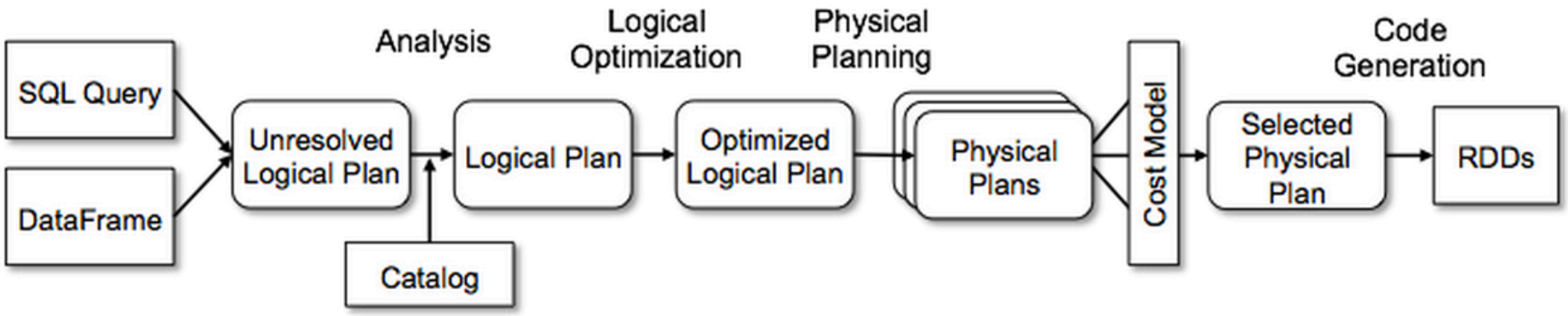

```
1  CREATE EXTERNAL TABLE page_view_stg(viewTime INT, userid BIGINT,
2                      page_url STRING, referrer_url STRING,
3                      ip STRING COMMENT 'IP Address of the User',
4                      country STRING COMMENT 'country of origination')
5  STORED AS TEXTFILE
6  LOCATION '/user/data/staging/page_view';
7
8  FROM page_view_stg pvs
9  INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-08', country='US')
10 SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url, null, null, pvs.ip
11 WHERE pvs.country = 'US';
```

- Data organisation/management
  - Logical (databases, tables)
  - Physical (partitions)
- Schema/structure (also on demand)
- Multiple storage formats (text, native, avro, parquet)
- Data languages: HQL, DML, DDL
- Extensibility through user defined functions
- SQL server with JDCB connectors

Spark SQL

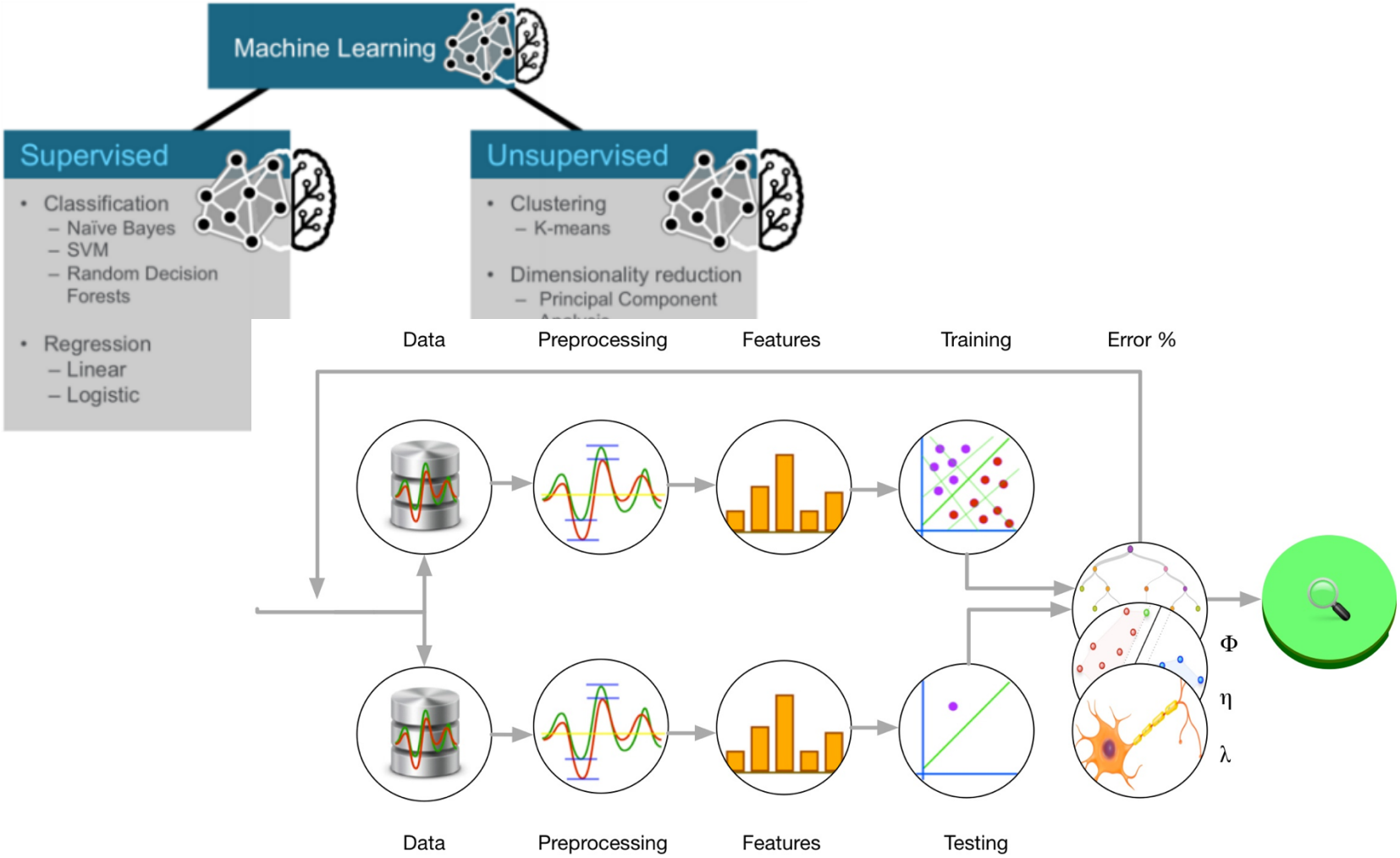

python
Scala
Java
HIVE QL
SQL-92
Spark SQL
SchemaRDD
Parquet
{JSON}
HIVE
cassandra


Single-User Response Time/Impala Times Faster Than
(Lower bars are better)
Geometric Mean (in Seconds)
250
200
150
100
50
0
6
28
4.6x
39
6.5x
78
13.0x
18
38
2.1x
176
9.8x
127
7.0x
30
114
3.8x
190
6.3x
216
7.2x
Interactive
q19, q42, q52, q55, q63, q68, q73, q98
Reporting
q3, q7, q27, q43, q53, q89
Analytics
q34, q46, q59, q79, ss_max
Impala 1.4
Spark SQL
Presto
Tez


Analysis
Logical Optimization
Physical Planning
Code Generation
SQL Query
DataFrame
Unresolved Logical Plan
Catalog
Logical Plan
Optimized Logical Plan
Physical Plans
Cost Model
Selected Physical Plan
RDDs

Spark Streaming

Spark ML

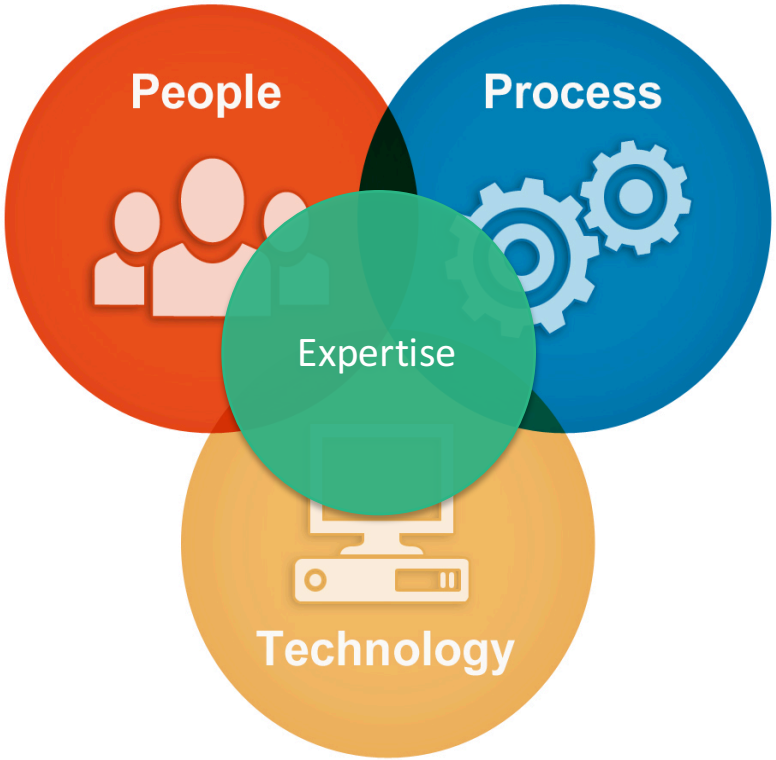

Machine Learning
Supervised
• Classification
– Naïve Bayes
– SVM
– Random Decision Forests
• Regression
– Linear
– Logistic
Unsupervised
• Clustering
– K-means
• Dimensionality reduction
– Principal Component
Data
Preprocessing
Features
Training
Error %
Φ
η
λ
Data
Preprocessing
Features
Testing

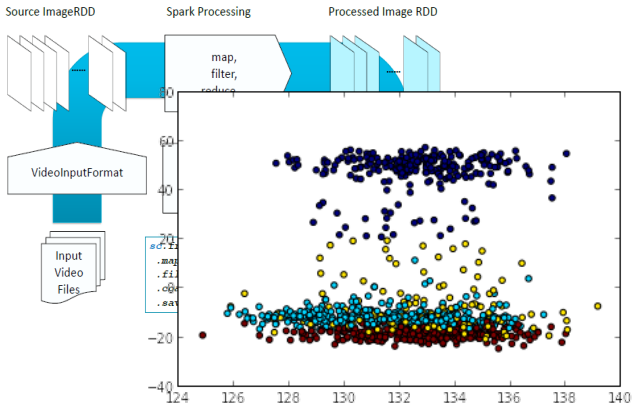
More Info

- Spark Programming Guide: http://spark.apache.org/docs/latest/programming-guide.html
- Code examples: https://github.com/piotrszul/spark-intro
- “Parallel programming with Spark” by Matei Zaharia
  - Presented at UC Berkeley AmpLab 2013: https://www.youtube.com/watch?v=e-56inQL5hQ
  - http://www.docfoc.com/parallel-programming-with-spark-matei-zaharia-uc-berkeley
- Spark SQL: http://www.slideshare.net/jeykottalam/spark-sqlamp-camp2014

Platform for Big Data and Visual Analytics

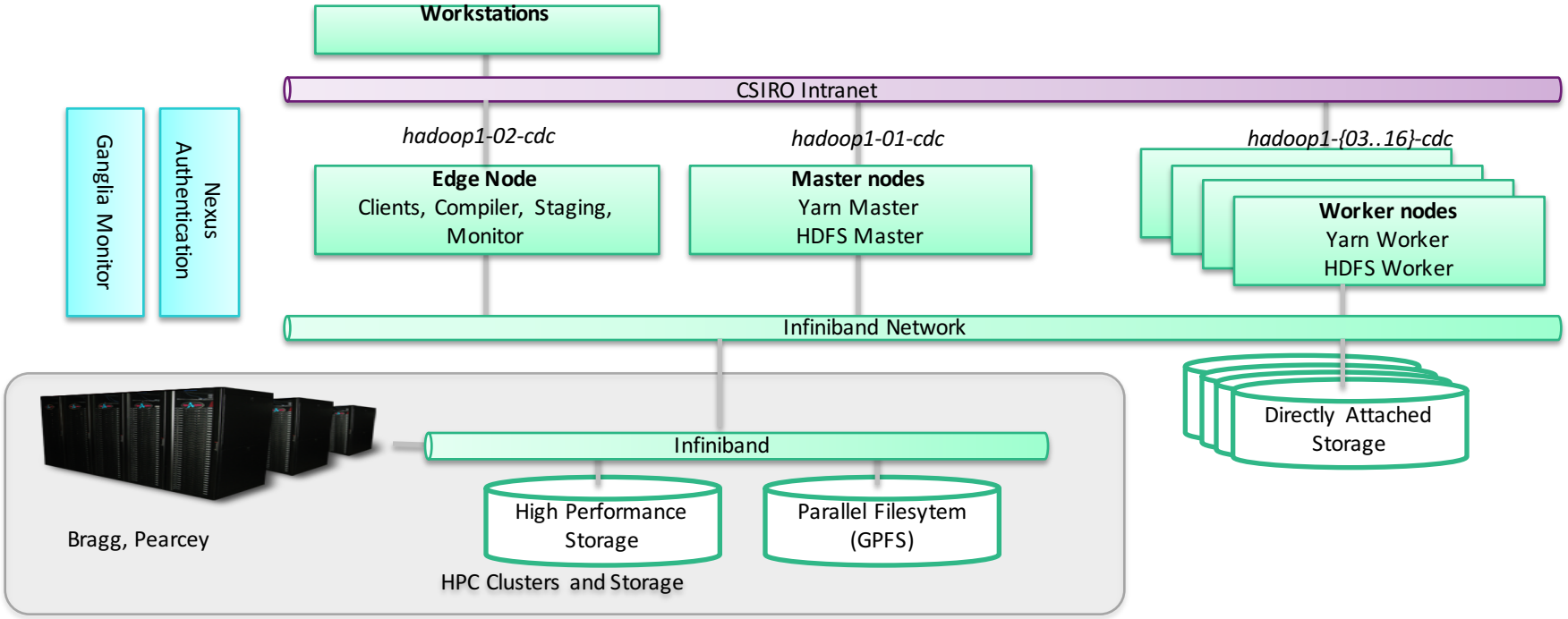
CSS FSP project orientated at providing incremental, use-case driven development of technical capabilities including skills, software and infrastructure to facilitate scientists' access to big data processing





Research Big Data Cluster

Research Big Data Cluster *supports big data related research and provides experimental big data processing capabilities.*



- Buld in collaboration with IMT on Bowen Cloud
- 14 workers: 2 x Intel Xeon E5-2660@2.20GHz CPU (8 cores), 128GB RAM
- 100 TB of disk storage in various configurations (HDFS, GPFS)
- connected with Infiniband to CSIRO HPC clusters: Bragg (GPGPU) and Pearcey (CPU)

Emergency Situation Awareness (ESA)

The Emergency Situation Awareness (ESA) system analyses tweets from the Australian and New Zealand region to detect information reported by the general public about emergency events. Since March 2011 it has been licensed to more than 50 policing and emergency management agencies, government departments, commercial organisations and universities.

Burst visualisation for a small 4.4 magni- tude earthquake in Melbourne in July 2011

with Mark Cameron, Rober Power and Bella Robinson

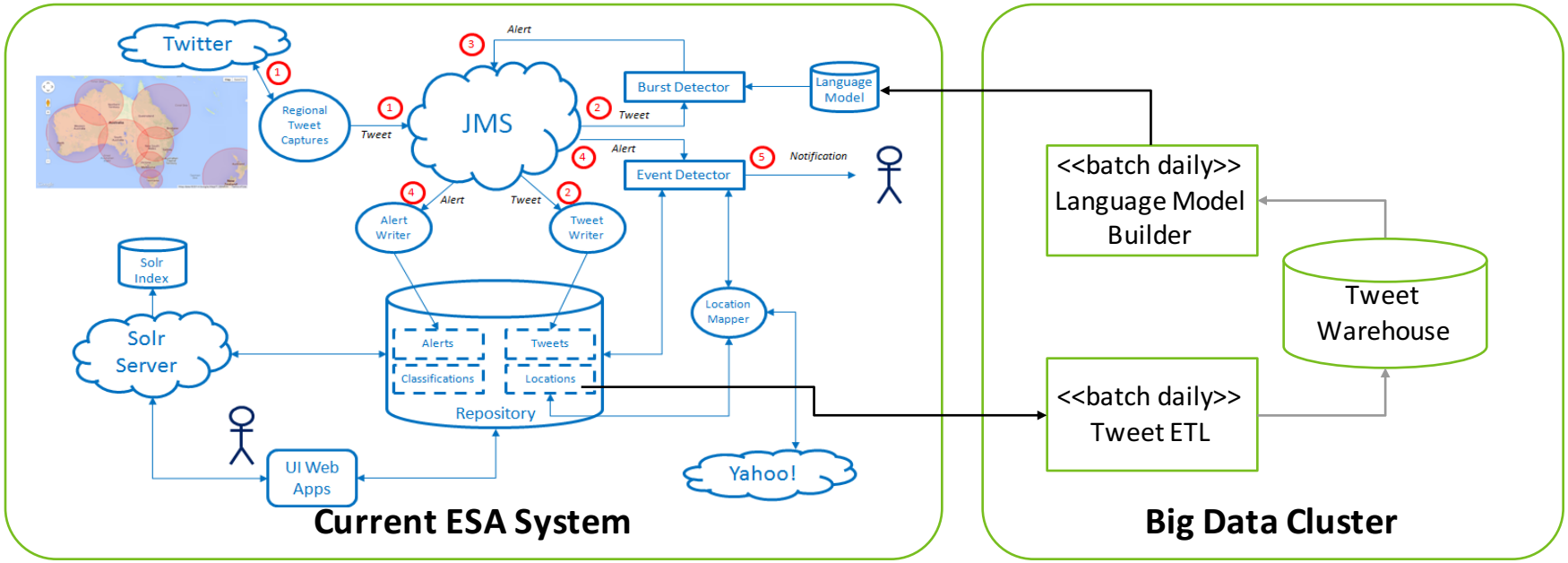
ESA – The Use Case

At the heart of ESA is a **statistical language model** used to detect unusual spikes of word frequencies in the online tweet stream.

- offline analysis to generate a language model over a 12-month tweet collection currently **takes 3 days**;
- as further human analysis is needed to identify and mark spurious outliers, it can take up **to 2 weeks** to build a single model;
- ideally, offline and online analysis would be closely coupled in time, so that the online monitoring is based on an up to date model.

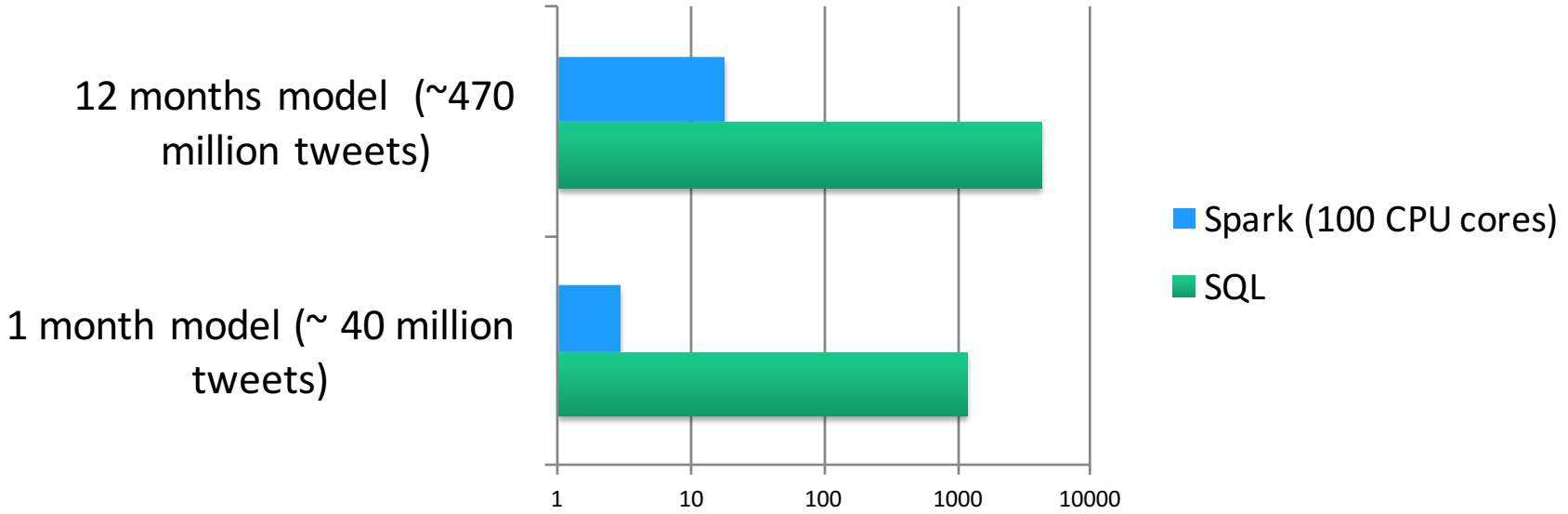
The primary objective of the use-case was to demonstrate that a significant reduction of model building time could be achieved by employing capabilities of the Research Big Data Cluster.

Solution Architecture



- using Apache Hive to create a warehouse of tweets in the cluster storage;
- using Parquet to efficiently store tweets;
- Apache Spark to implement the model building process;
- Spark SQL to select tweets to be included in the model;
- daily ETL processes to update the warehouse with new tweets captured by the ESA application

Results



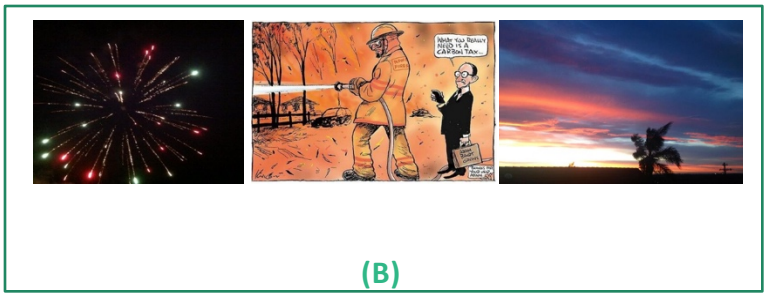
Comparison of model-building times for SQL and Spark based implementations (in minutes with ***log scale***).

The Spark based implementation is easily scalable to hundreds of CPU cores - running on 100 CPU cores is able to build the language model in under 20 minutes, which constitutes a speedup of almost 250 times compared to to 3 days of the SQL implementation.

Bushfires in tweeted images (ESA)

We explored on-line classification of tweeted images both in general and specifically for the purpose of tracking development of bushfires.

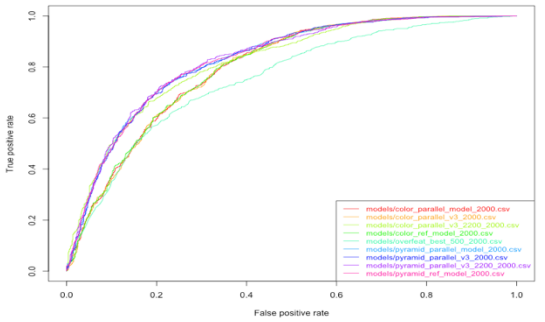
(A)



(B)

- integrated Spark with OpenIMAJ for image processing
- implemented a scalable training module for image classification with visual bag of words approach using Spark MLLIb
- Spark Streaming for online classification of images

We also investigated employing pre-trained general image recognizers such as OverFeat and Clarifai and compared classification performance of all approaches.



Automated Big Video Analysis

Integrated video camera systems have been installed on fishing boats to trial for the 24/7 fishery monitoring of tuna longline operations in Australia.

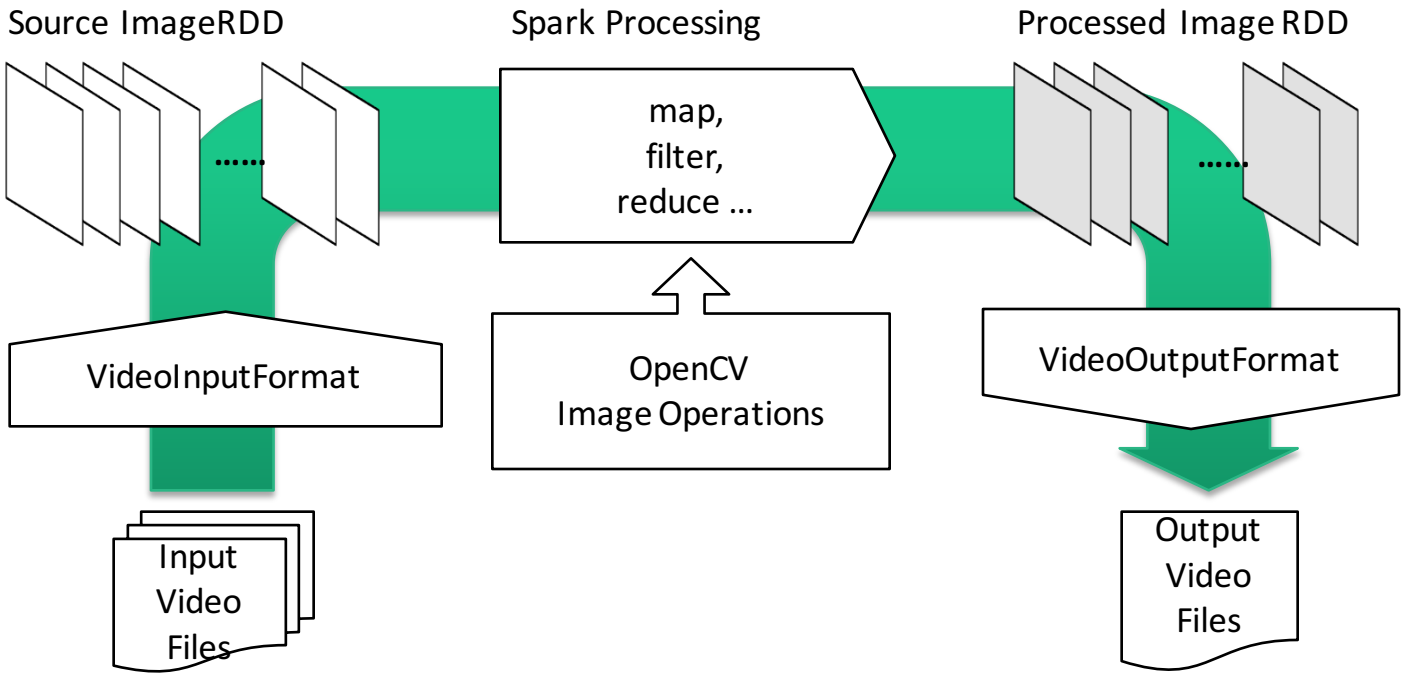
A single 30 day trip produces 720 hours or 180 GB of video footage

- These need to be watched and fish counted

As the first step of the automated video processing, CSIRO has developed method to extract significant frames

- sample video been reduced from 1h59min to 19 seconds without losing fish information
- single CPU processing takes about 9 hours

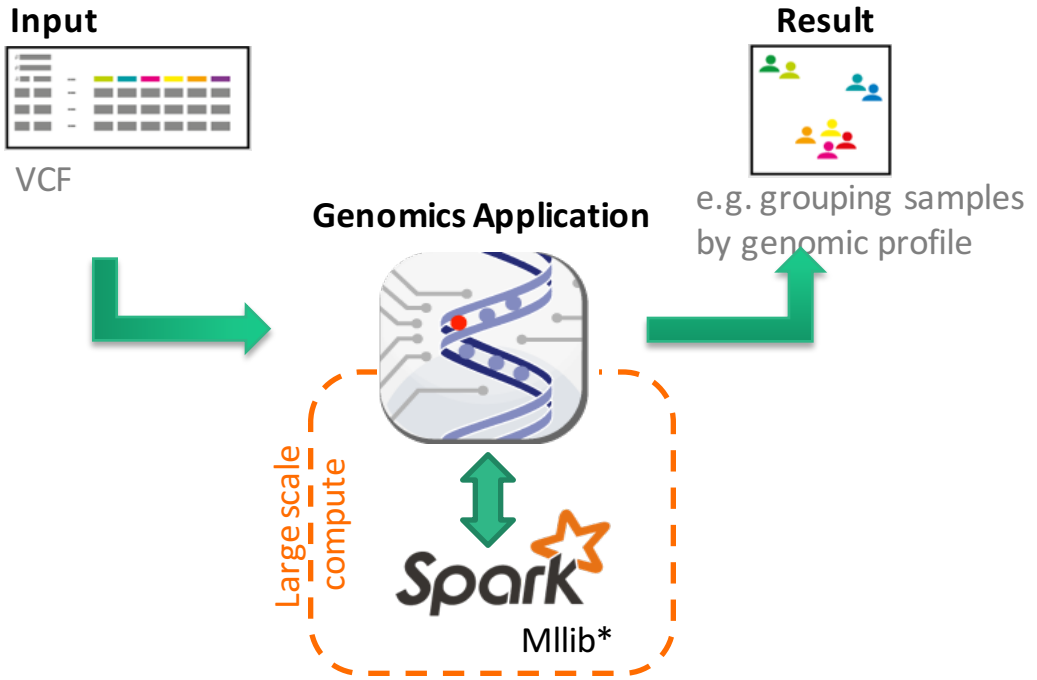
Automated Big Video Analysis

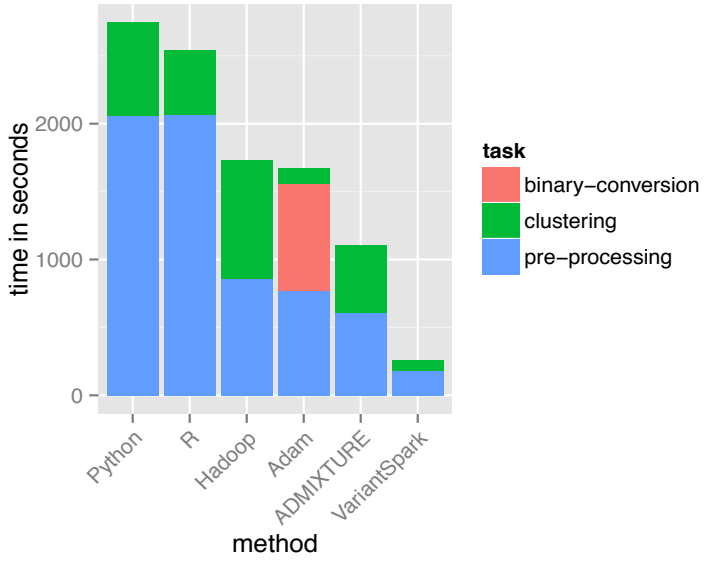


- Intergrate OpenCV image processing withSpark (image RDDs) and provides splittable Hadoop I/O formats for reading/writing video files
- a distributed version of video reduction scales well at least up to 512 processing cores, and allows processing of 100GB of video input in just a few minutes (tested on Bragg)

Machine Learning for Genomic Variants

VariantSpark is the interface enabling Spark's MLlib machine learning algorithms to be applied to genomics data being developed by Transformational Bioinformatics team led by Denis Bauer.





Variant Spark deployed on Big Data Cluster is 80% faster then ADAM – the second best implementation. Check it out at http://tinyurl.com/j8cxc3v

Hive, Spark SQL, Parquet

See: https://github.com/piotrszul/spark-intro

Thank you

Piotr Szul
Senior Engineer
E piotr.szul@csiro.au