ML-Ops intensive

Part 1: Managing Your Data

Alexander Myltsev[1], Vasily Safronov[1], Andrey Ustyuzhanin[1]

[1] HSE University

April 8, 2021

Skolkovo Institute of Science and Technology

Quick self intro

- Vasily Safronov head of BI-analytics and DataOps at WakeApp - leading mobile marketing company, researcher at the Laboratory of Methods for Big Data Analysis (LAMBDA), HSE University

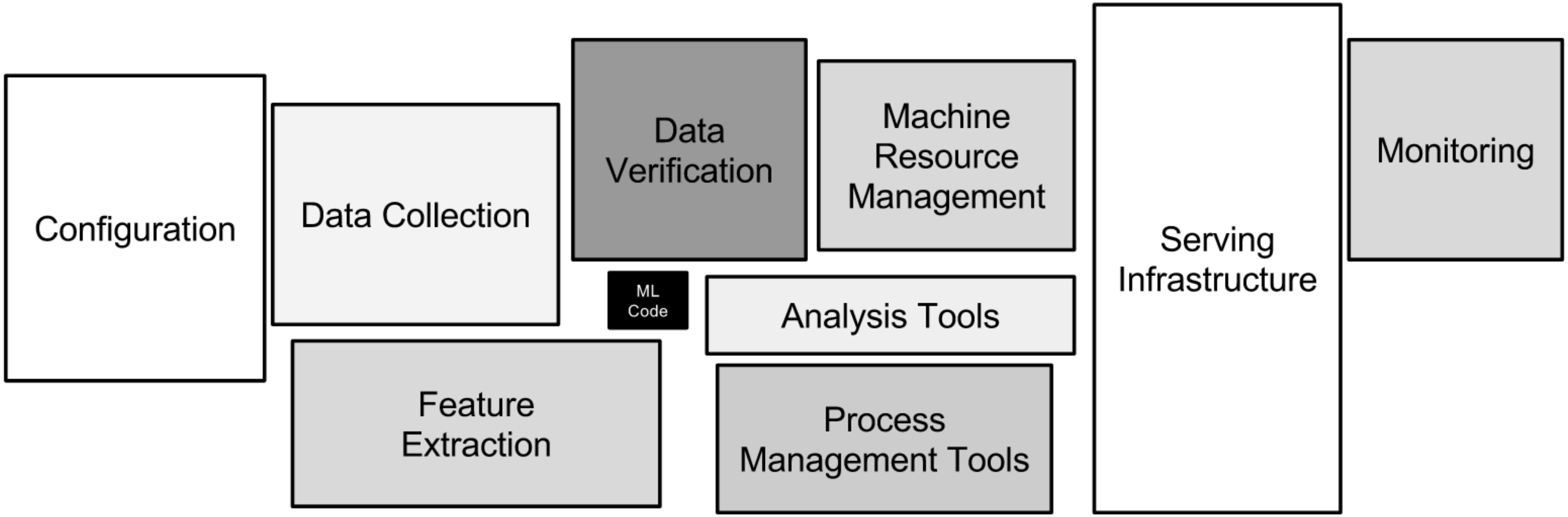
- Alexander Myltsev – researcher at LAMBDA, HSE University

- Andrey Ustyuzhanin – head of LAMBDA, HSE University

LAMBDA

- Laboratory of methods for Big Data Analysis at Higher School of Economics (HSE) est. 2015,
- Collaborates with LHCb, OPERA, SHiP, NICA and other fundamental science international experiments
- Mission: develop and apply machine learning (ML) methods for solving scientific challenges from various domains
- Co-organized data-intensice competitions at Kaggle and IDAO since 2015 (Flavours of Physics, TrackML)
- Education activities: 7 summer/autumn schools on ML, ML courses at ICL, Clermont Ferrand, URL Barcelona, Coursera

Modern ML researcher stack



https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf

Intensive outline

- Data management
  - Classic data management architectures
  - Data lakes
  - Data processing: MapReduce, Spark
  - Data and model versioning
- SWE4ML
  - (see tomorrow)

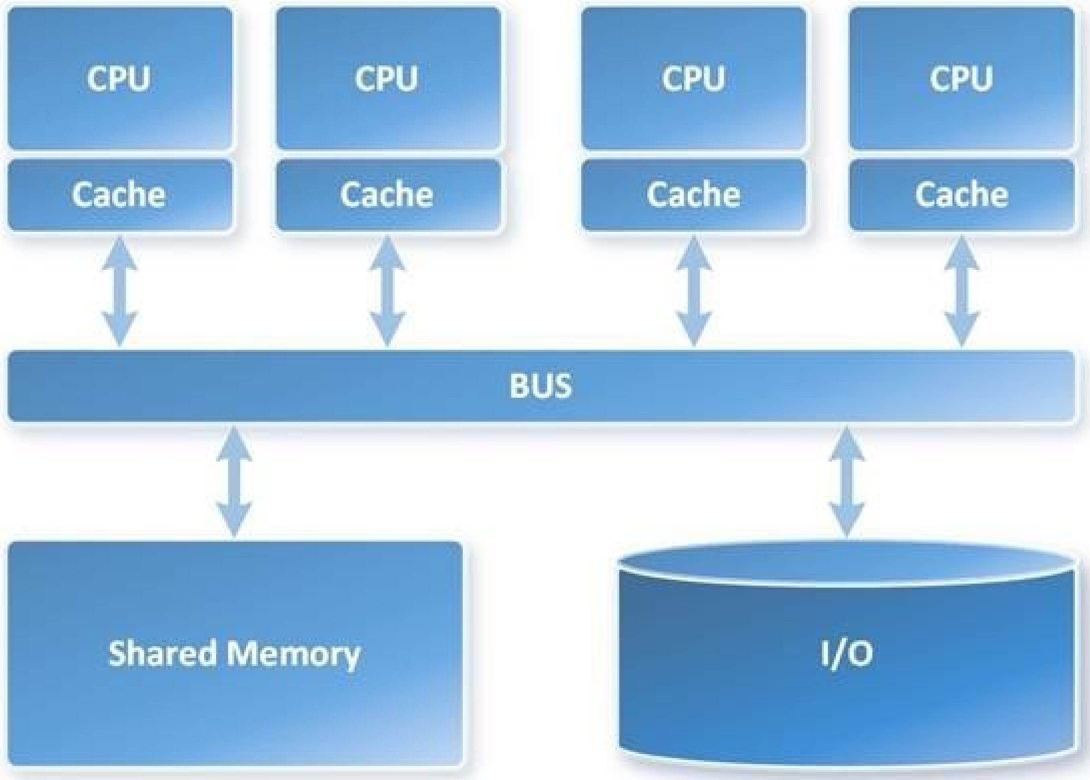
Dive into history: Classical architecture

Classical Architecture

Symmetric Multi-Processing

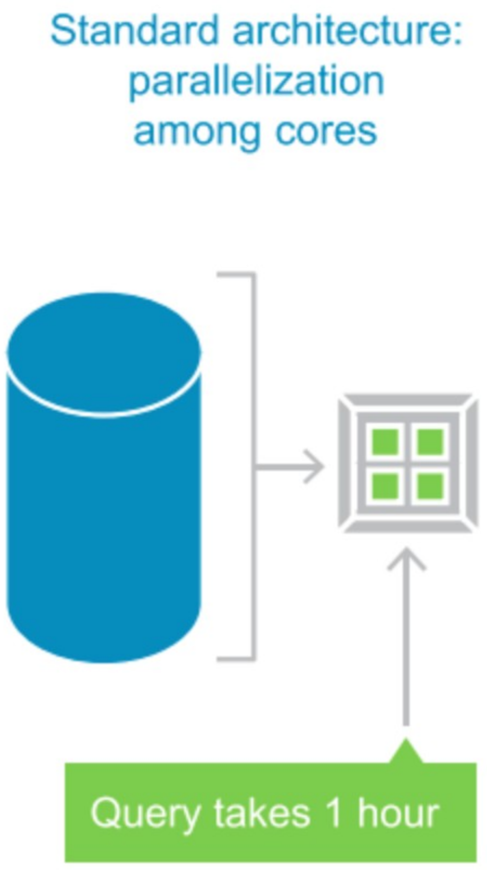
Single instances with common:

- Operating System
- Memory
- I/O devices
- Bus.



Parallelization in classic architecture

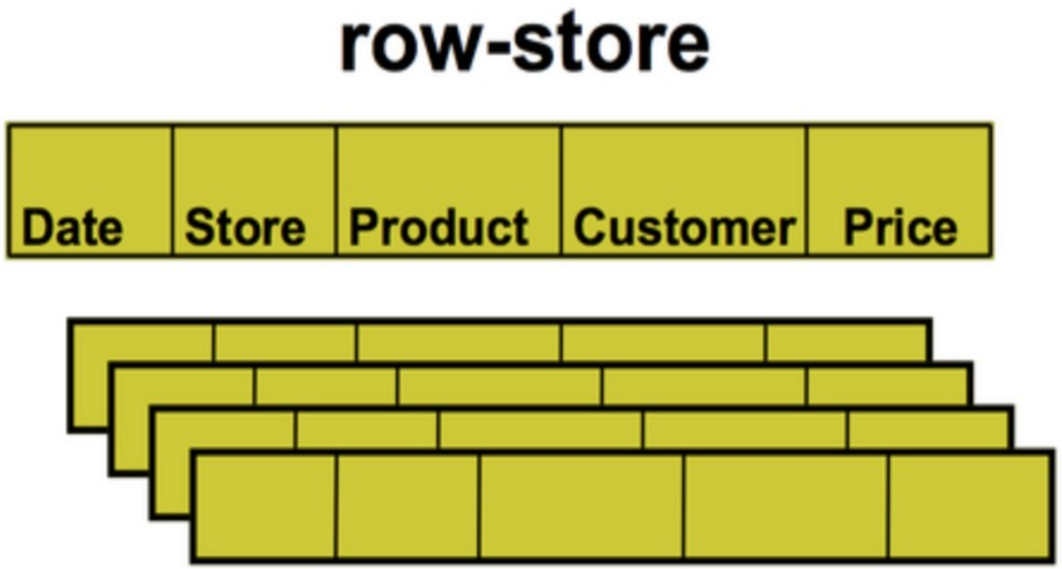
- All the processors available to all individual processes
- The processors share the workload



Centralized database

Database based on classic Symmetric Multi-Processing architecture:

- Typically has row-based store with relational schema
- Transactional (ACID):
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Optimised for running production systems



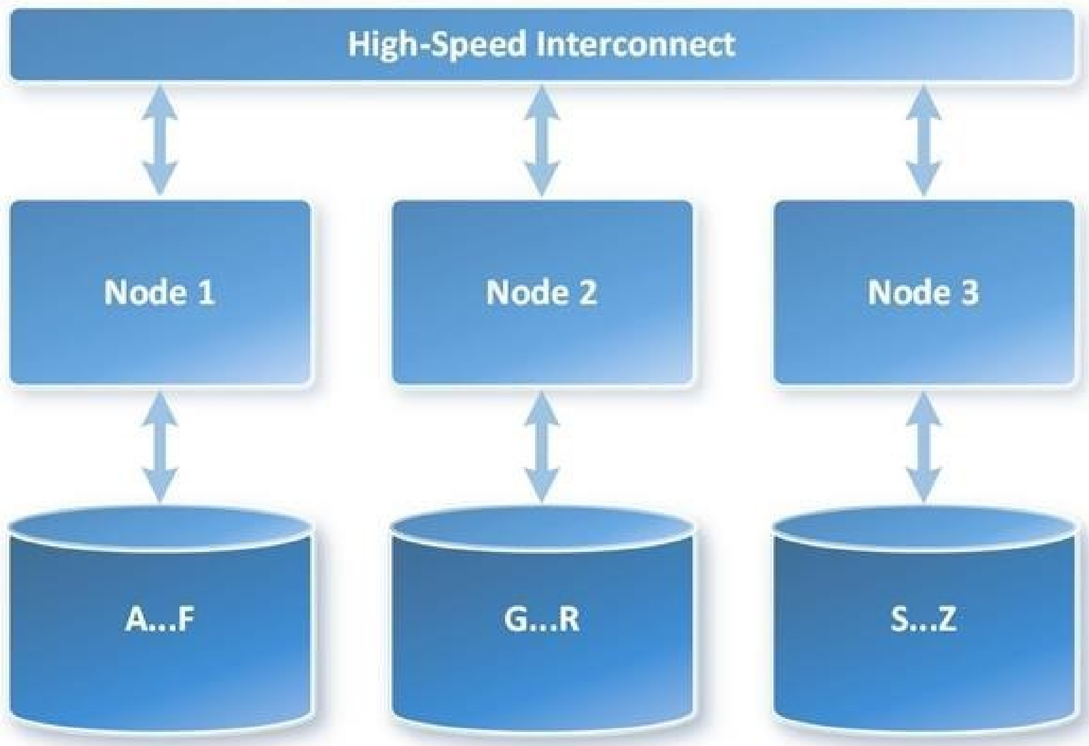
Challenges

- **High Data Volume and Data Growthing**
  With increasing data and a growing analytics base, the data volume has grown rapidly.
- **Time that needed for Data Loading**
  We finds the classic data loading speed inadequate to intake and process the increasing quantity of data flowing from other databases or non-relational systems.
- **Variety Data**
  The data collecting carried out from different data sources and in various formats with unstructed or semi-structed data that hard to store in relation format.
- **Long Time data processing**
  Query execution times are slowing down due to the increase of data and it is becoming increasingly difficult to generate insights.

Modern state:
Data Lake Architecture

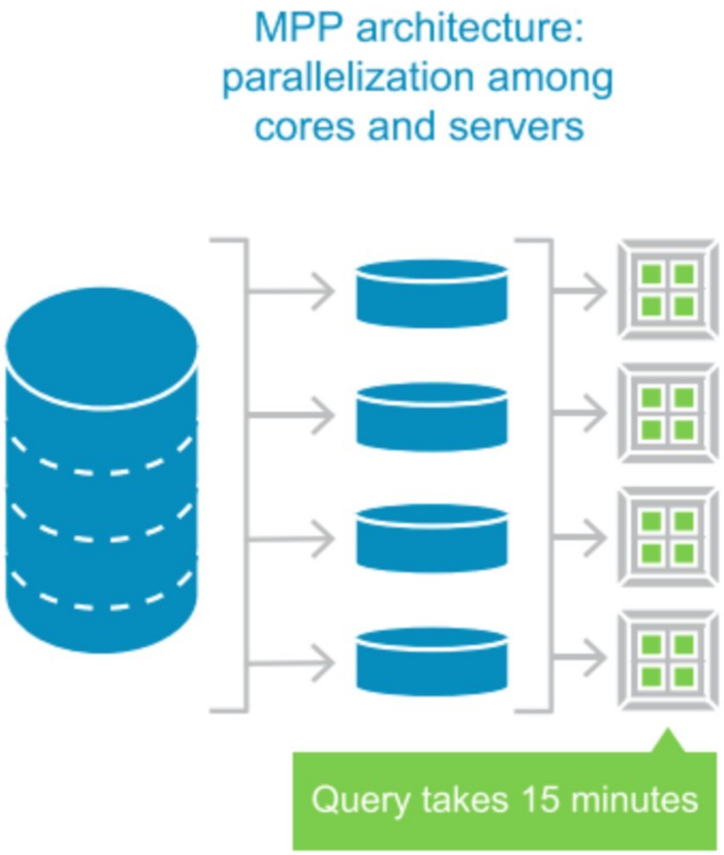
Massively Parallel Processing

- Large number of small homogeneous processing nodes
- Nodes are independent
- Typically do not share memory
- Typically each processor may run its own instance of an operating system



Parallelization in MPP

- All the processors available to all individual processes
- The processors share the workload



Data Gravity

Data Gravity

$$\frac{\left(\text{Data Mass} \times \text{Application Mass}\right) \times \text{Number of Requests per second}}{\left(\text{Latency in seconds} + \left(\text{Average Request Size in MBs} \,/\, \text{Bandwidth in MBs per second}\right)\right)^2}$$

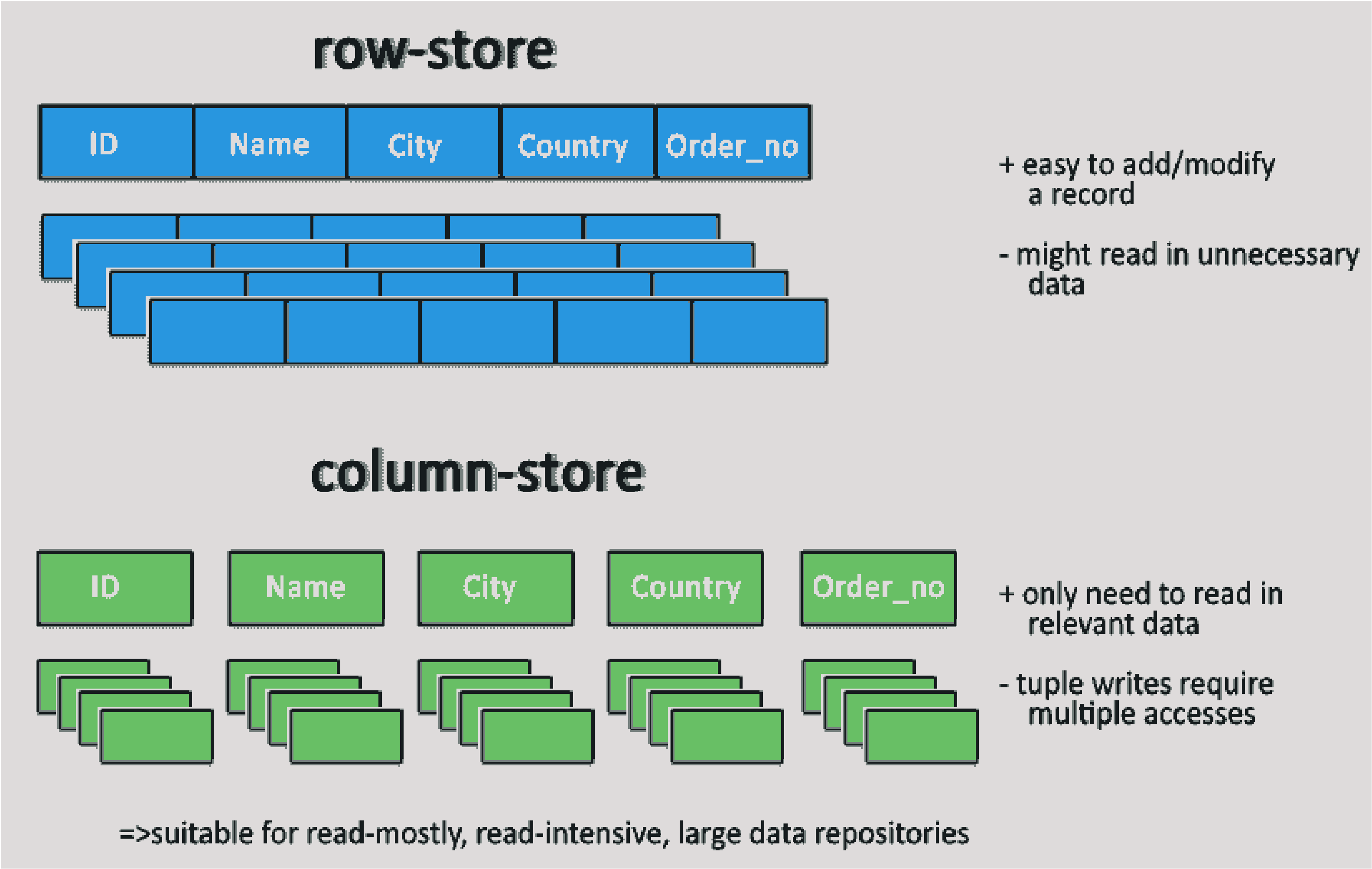
"It’s the gravity — and other things that are attracted to the data, like applications and processing power — that moves to where the data resides."

Dave McCrory, 2010

Transitioning to MPP

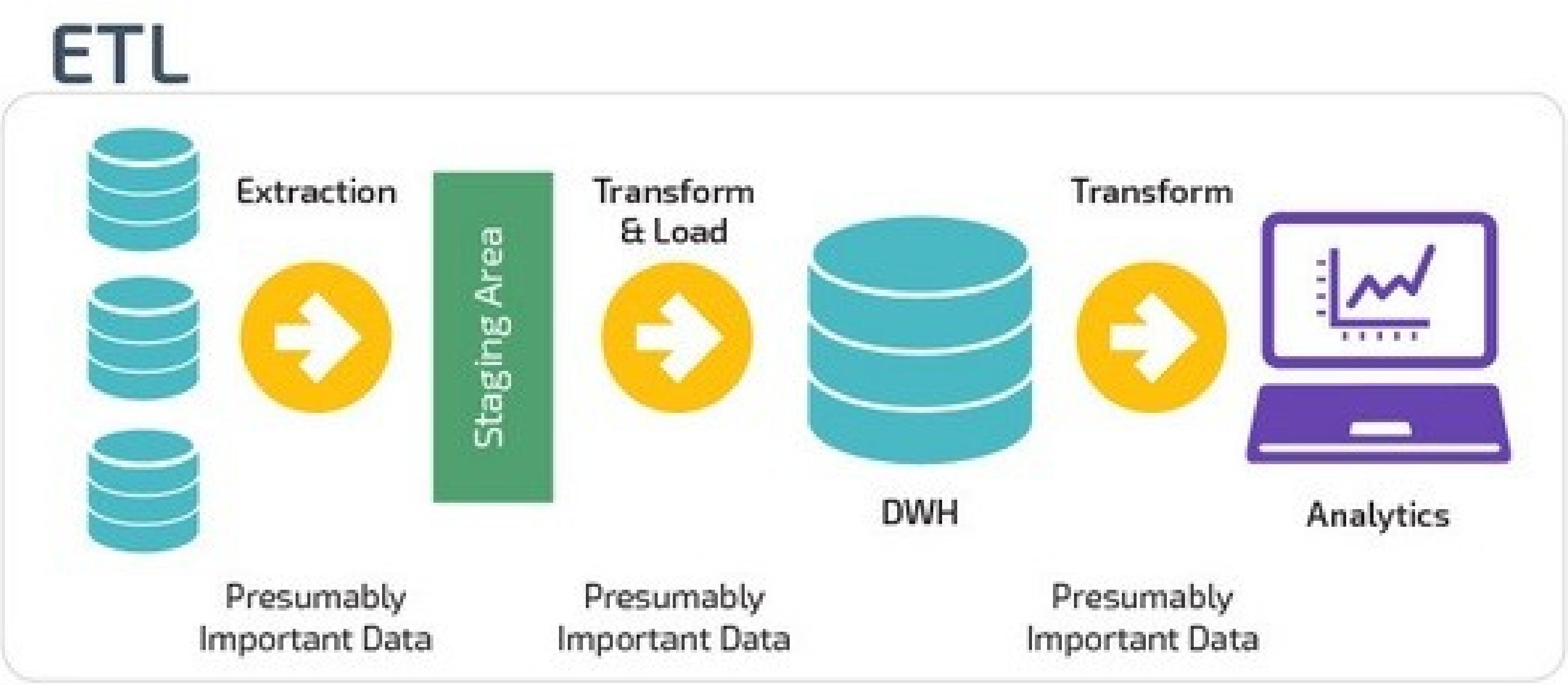
- "Shared nothing" MPP architecture
- Distribute large data across nodes

Data lakes: Distributed databases



Steps of data processing in classic way

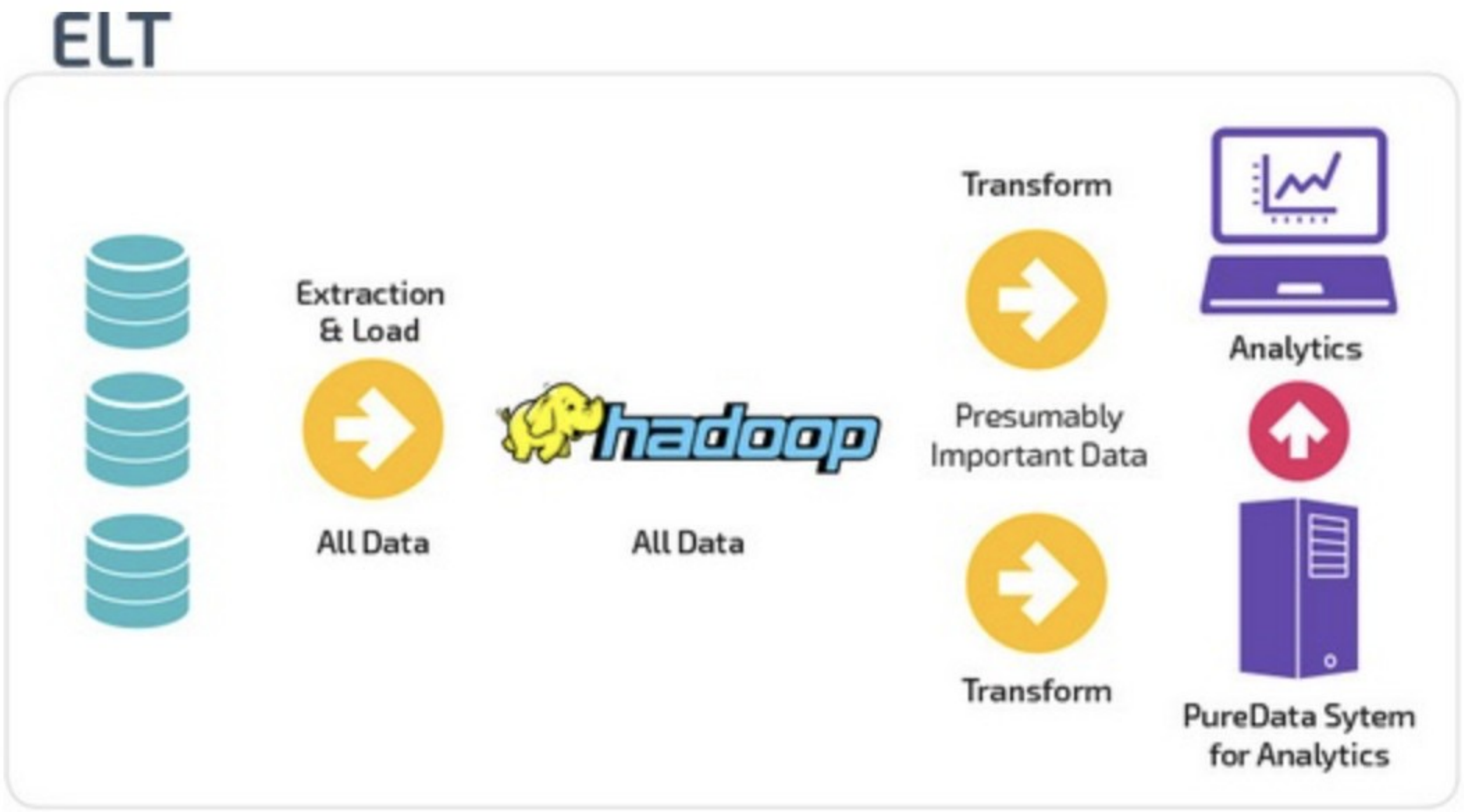
- Export
- Transform
- Load



https://sudonull.com/post/32564-Data-Warehouse-Architecture-Traditional-and-Cloud

Data lakes: Moving from ETL to ELT

- Export
- Load
- Transform



https://sudonull.com/post/32564-Data-Warehouse-Architecture-Traditional-and-Cloud

Benefits of MPP and ELT

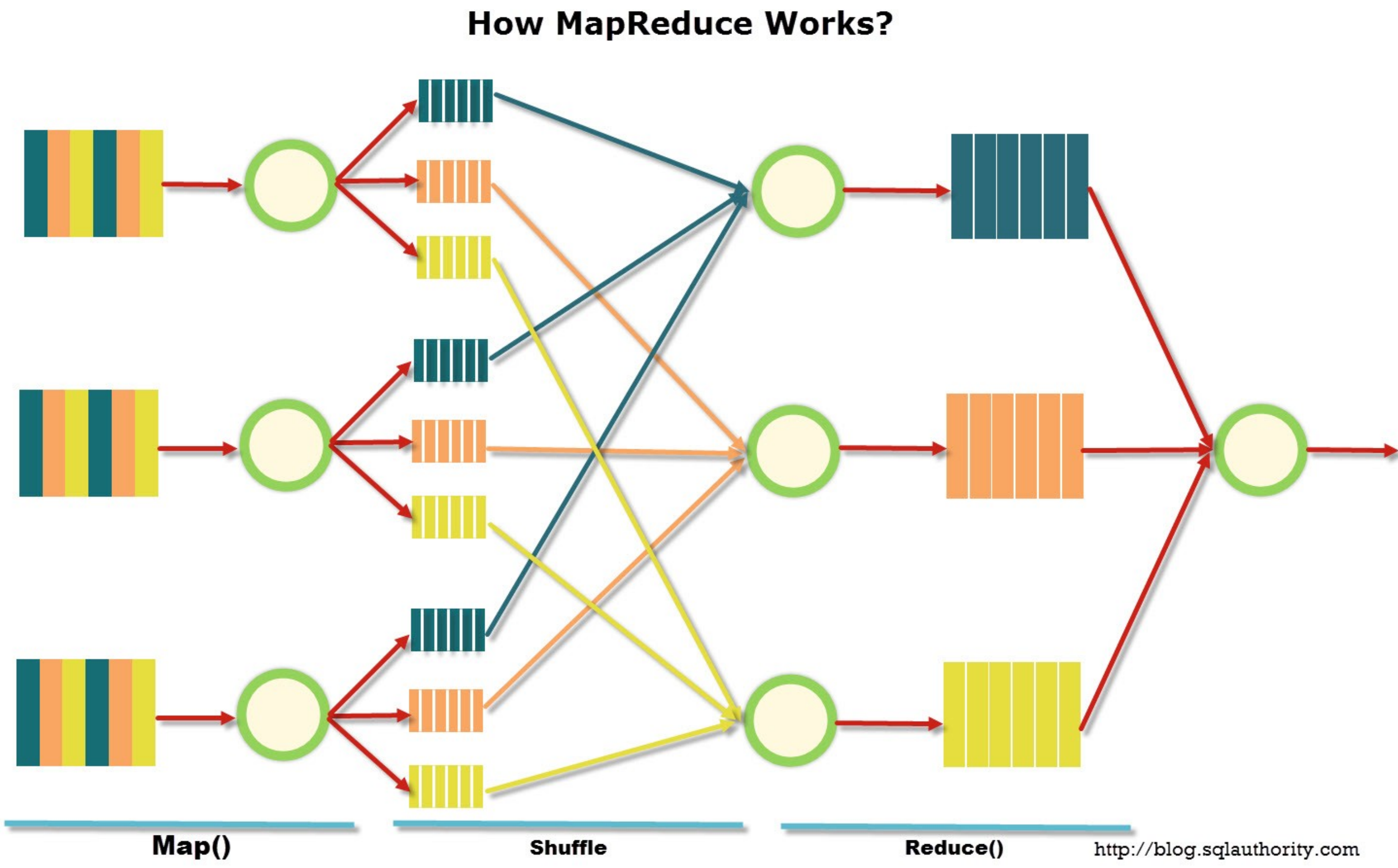
- We can now feel confident handling the data volume and growth with the ability to scale the data storage as needed.
- With ELT and the power of parallel processing in analytics platform, we can load data into faster and within the expected time-window.
- By aligning with MPP design, we can achieve breakthrough query performance, allowing for real-time reporting and insight into their data.

MapReduce

How MapReduce works?

MapReduce has five different steps:

- Preparing **Map()** Input
- Executing User Provided **Map()** Code
- **Shuffle** Map Output to Reduce Processor
- Executing User Provided **Reduce** Code
- Producing the Final Output



Map() Procedure

- Input: source objects
- Output: set of pairs (key $\rightarrow$ value)

Shuffle() Procedure

- Sort objects
- Distributed by reducers

Reduce() Procedure

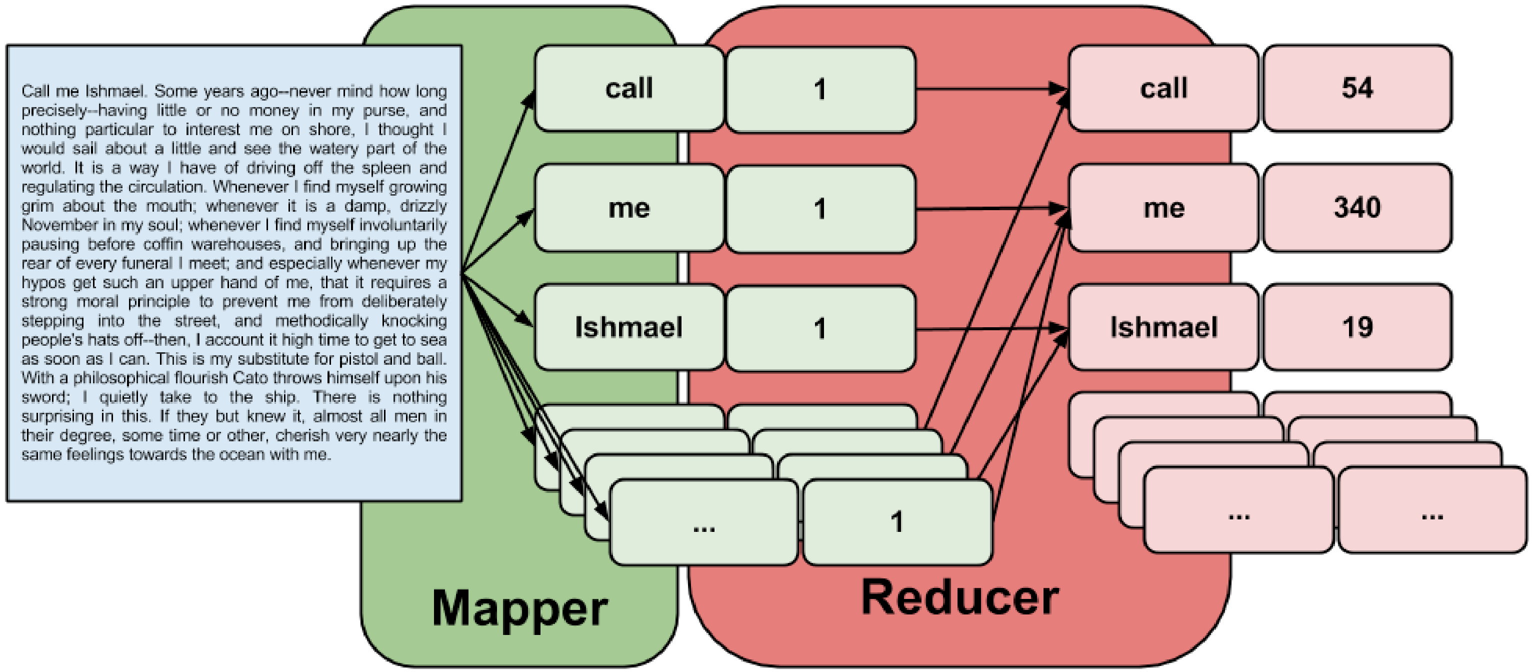
- Input: one key → set of all values
- Output: one key → one value

Word count: task

Task:

- File with any text.
- One row - one document.
- We need count of each word in the text

Word count: illustration



https://glennklockwood.com/data-intensive/hadoop/streaming.html

Word count: python realization

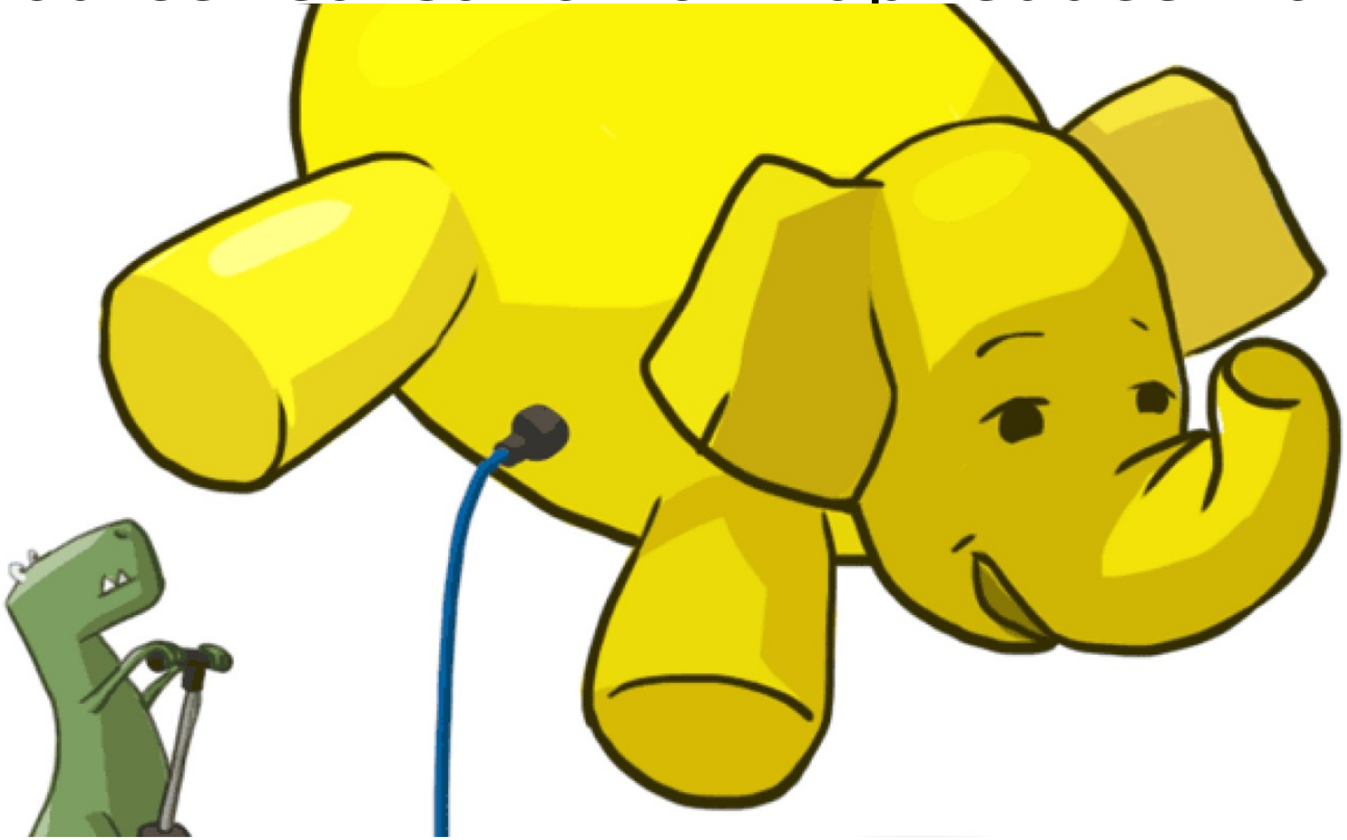
```
def map(string):
    for token in string.split():
        return token, 1

def reduce(key, values):
    return key, sum(values)
```

Hadoop

At the beginning:

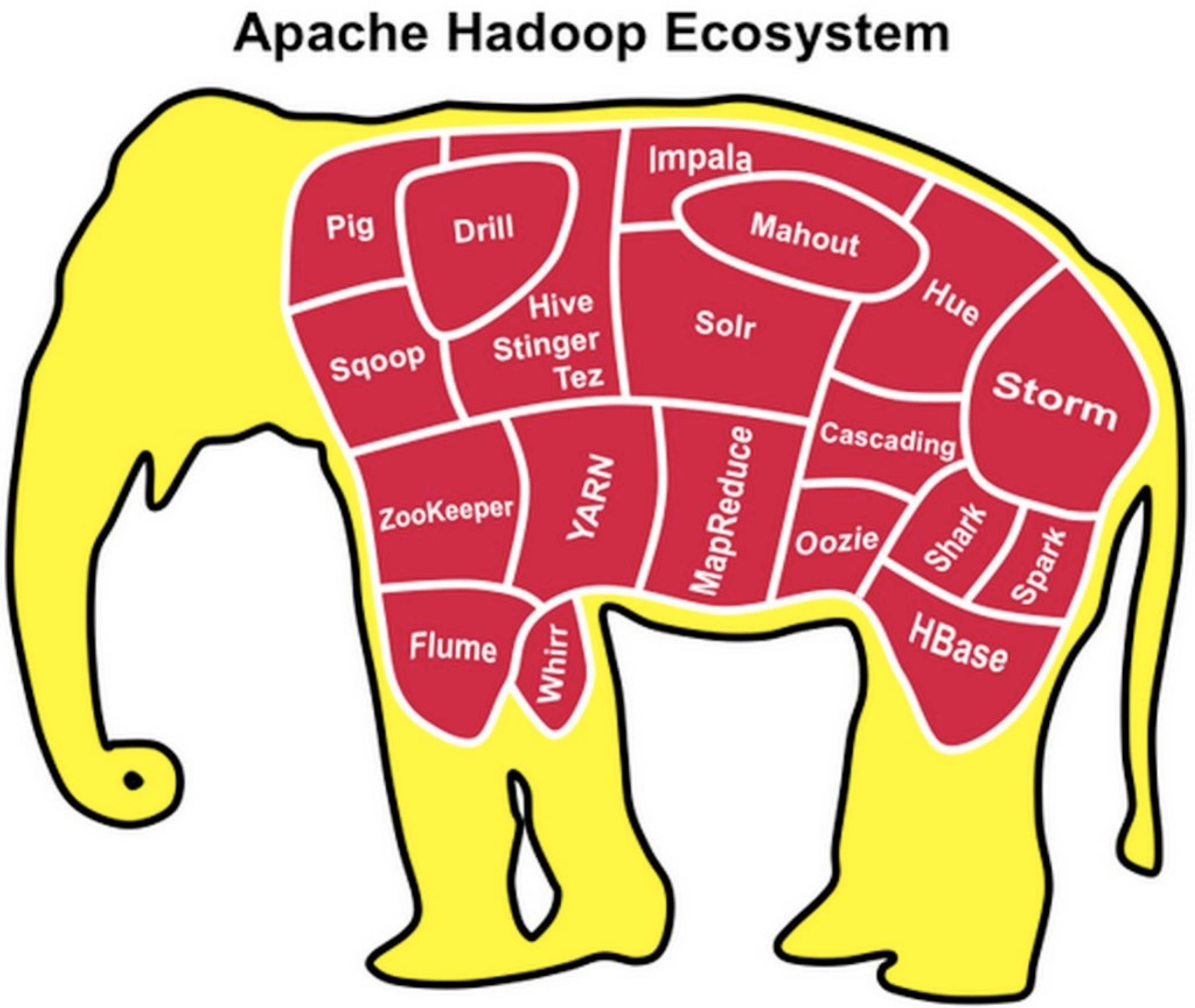
Opensource realisation of MapReduce Framework



Hadoop ecosystem

And now it's

large ecosystem

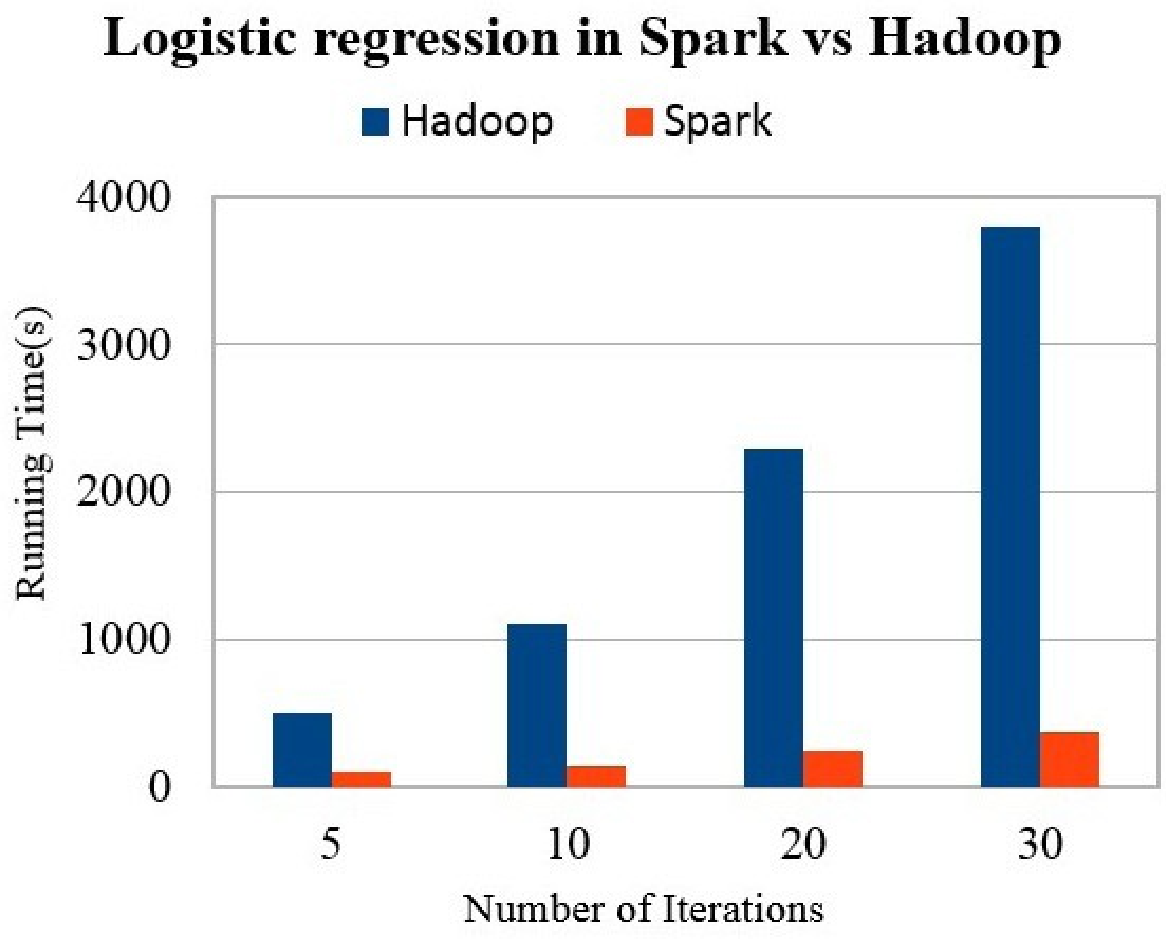


Spark

What is Spark?

- General engine for large-scale data processing
- Supports cyclic data flow and in-memory computing
- Java, Scala, Python, R interfaces
- Libraries: SQL and DataFrames, MLlib, GraphX, and Spark Streaming.

Speed/Iterative processes



https://media.neliti.com/media/publications/265040-big-data-in-the-cloud-environment-and-en-3d8f6ad1.pdf

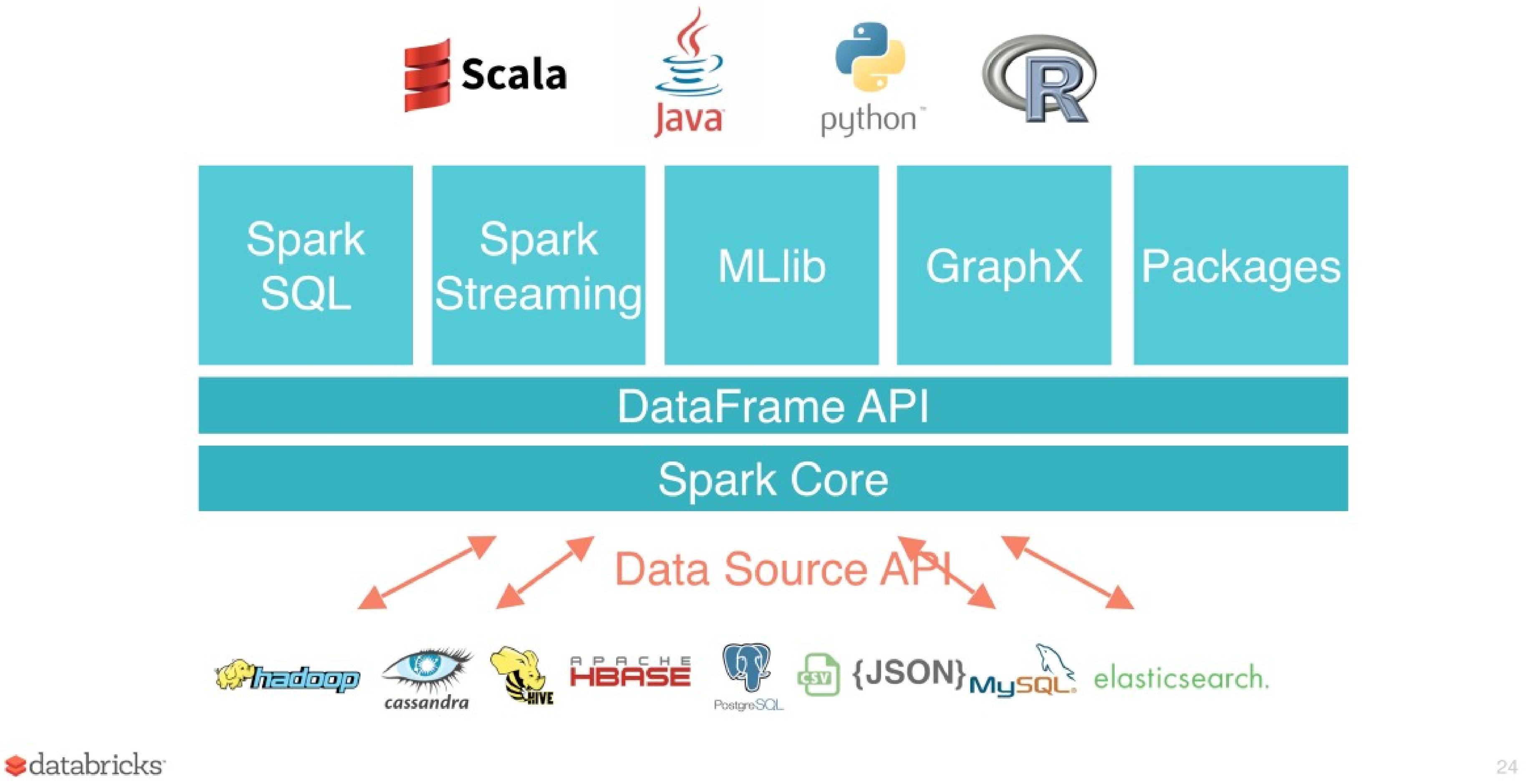
Speed/Iterative processes

Spark vs Hadoop MapReduce

Factors	Spark	Hadoop MapReduce
Speed	100x times than MapReduce	Faster than traditional system
Written In	Scala	Java
Data Processing	Batch / real-time / iterative / interactive /graph	Batch processing
Ease of Use	Compact & easier than Hadoop	Complex & lengthy
Caching	Caches the data in-memory & enhances the system performance	Doesn't support caching of data

https://data-flair.training/blogs/spark-vs-hadoop-mapreduce/

Spark stack



https://spark.apache.org/docs/latest/cluster-overview.html

Resilient Distributed Dataset

RDD

- Distributed collection of objects in memory
- Fault-tolerant: RDD can be reconstructed automatically
- RDD can be cached to save computations

RDD operations

Transformations:

- are lazy and executed when an action is run
- operations on RDDs that return a new RDD

`map(), flatMap(), filter(), mapPartitions(), mapPartitionsWithIndex(), sample(), union(), distinct(), groupByKey(), reduceByKey(), sortByKey(), join(), cogroup(), pipe(), coalesce(), repartition(), partitionBy(), …`

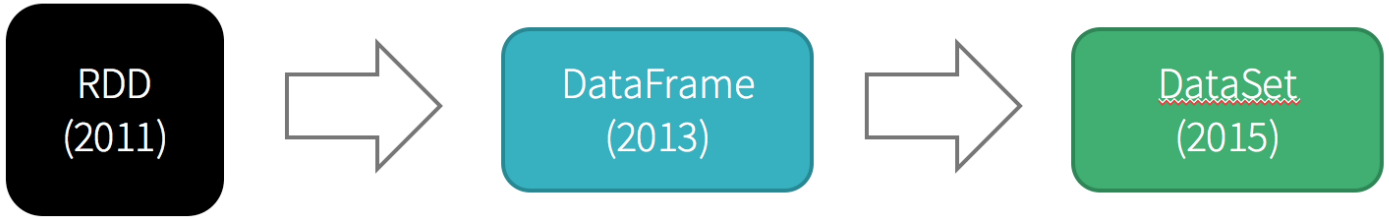
Actions:

- return a result to the driver program or write it to storage, and kick off a computation

`reduce(), collect(), count(), first(), take(), takeSample(), takeOrdered(), saveAsTextFile(), saveAsSequenceFile(), saveAsObjectFile(), countByKey(), foreach(), ...`

Spark data structures

History of Spark APIs



RDD (2011)

Distribute collection of JVM objects

Functional Operators (map, filter, etc.)

DataFrame (2013)

Distribute collection of Row objects

Expression-based operations and UDFs

Logical plans and optimizer

Fast/efficient internal representations

DataSet (2015)

Internally rows, externally JVM objects

Almost the "Best of both worlds": type safe + fast

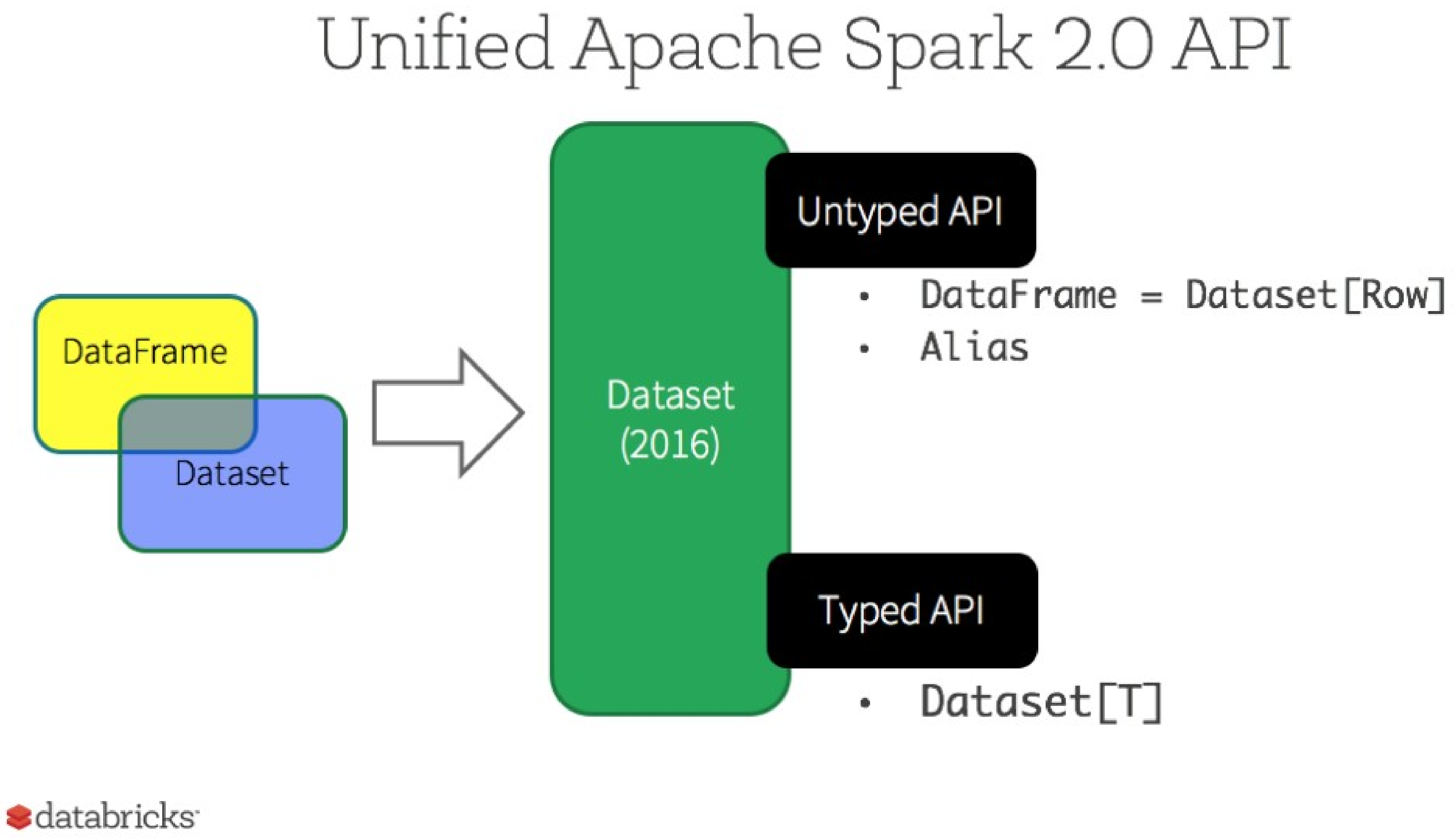
But slower than DF
Not as good for interactive analysis, especially Python

databricks

https://www.slideshare.net/databricks/jump-start-into-apache-spark-and-databricks

Spark data structures

- **DataFrame**
  DataFrame is a distributed collection of data organized into named columns.
- **Dataset**
  The goal is to provide users easily express transformations on domain objects. Starting in Spark 2.0 DataFrame API will merge with Datasets API, unifying data processing capabilities across all libraries

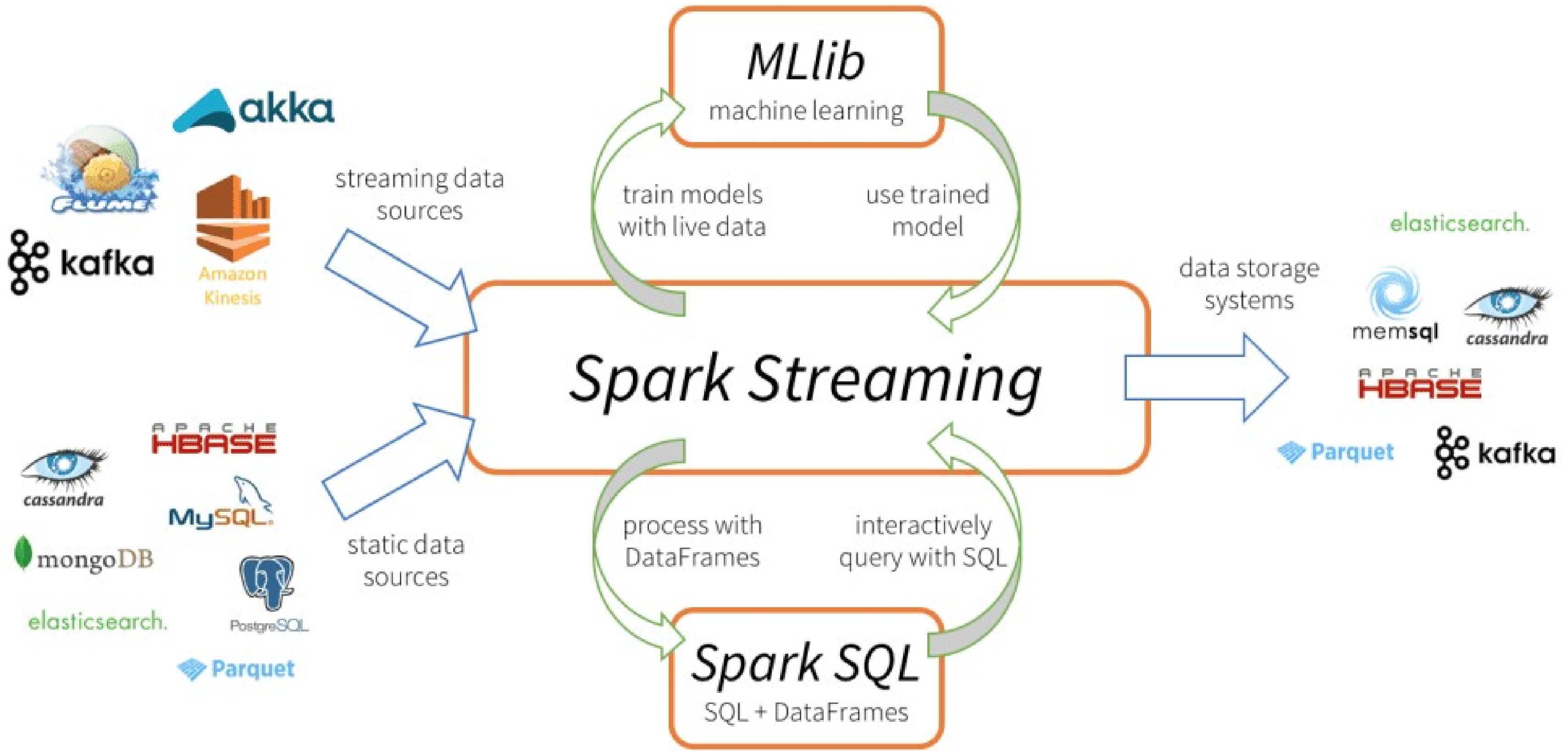


https://databricks.com/blog/2016/06/22/apache-spark-key-terms-explained.html

Spark use-cases

- Hadoop-like
  map-reduse operations (addition / replacement)
- Data scientist/analyst's workplace
  great with jupyter notebooks or something similar
- Distributed python environment
  easily run your own tasks on the cluster

Spark Streaming



https://glennklockwood.com/data-intensive/hadoop/streaming.html

MLlib

- spark.mllib contains the original API built on top of RDDs.
- spark.ml provides higher-level API built on top of DataFrames for constructing ML pipelines.

Data Versioning Control (DVC)

Data files hell problem

- 1. Data files stored in different places, not in your repository.
- 2. Tons of data file versions:
  - model.pkl
  - model_L7_e120.pkl
  - model_vgg16_L5tune_e120.pkl
  - model_L7_e160_cleansed.pkl
  - model_vgg16_L45tune_e120.pkl
  - model_vgg16_L45tune_e160_noempty.pkl
  - ...
- 3. Data files stored separately of your code files.

```
$ git checkout finetune_head # creates even more mess
```

Data files hell in a team

- How to create a reproducible ML project?
- How to scale ML process in a team
- How to pass ML model to deployment or revert a model (to devops)

Methodology mismatch

“Data science as different from software as software was different from hardware”

https://dominodatalab.wistia.com/medias/fq0l4152sh

What is special about Data Science?

New artifacts to manage:

- Experiment: Code + Data files.
- Metrics.
- ML pipelines and reproducibility.

Different process:

- R&D like. A lot of trials and errors, progress should be measured in a different way.
- Ephemerality. Hard to communicate and track the progress.

DVC project motivation

“Git extension for data scientists – mange your code and data together”

Open source tool Data Version Control

to manage ML projects: http://dvc.org

DVC is a data science platform on top of open source stack. GitHub repo:

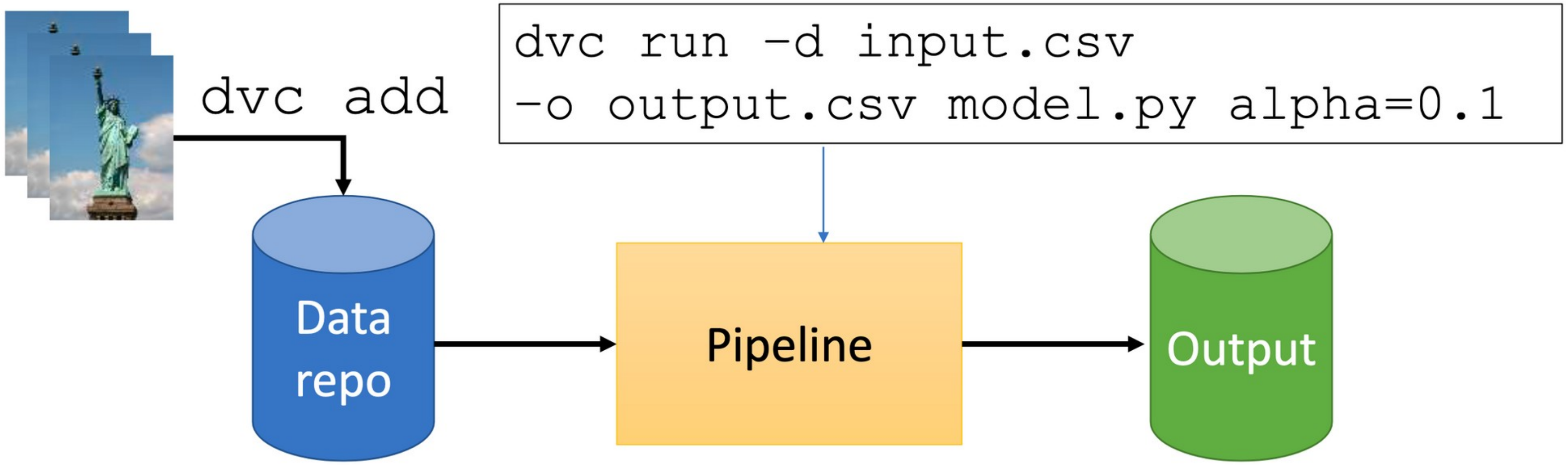
https://github.com/iterative/dvc

Download binaries (Mac, Linux, Windows)

or `$ pip install dvc`

It extends Git by commands: `dvc add`, `dvc run`, `dvc repro`, `dvc remote`

What DVC does?



https://www.slideshare.net/joshlk100/reproducible-data-science-review-of-pachyderm-data-version-control-and-git-lfs-tools

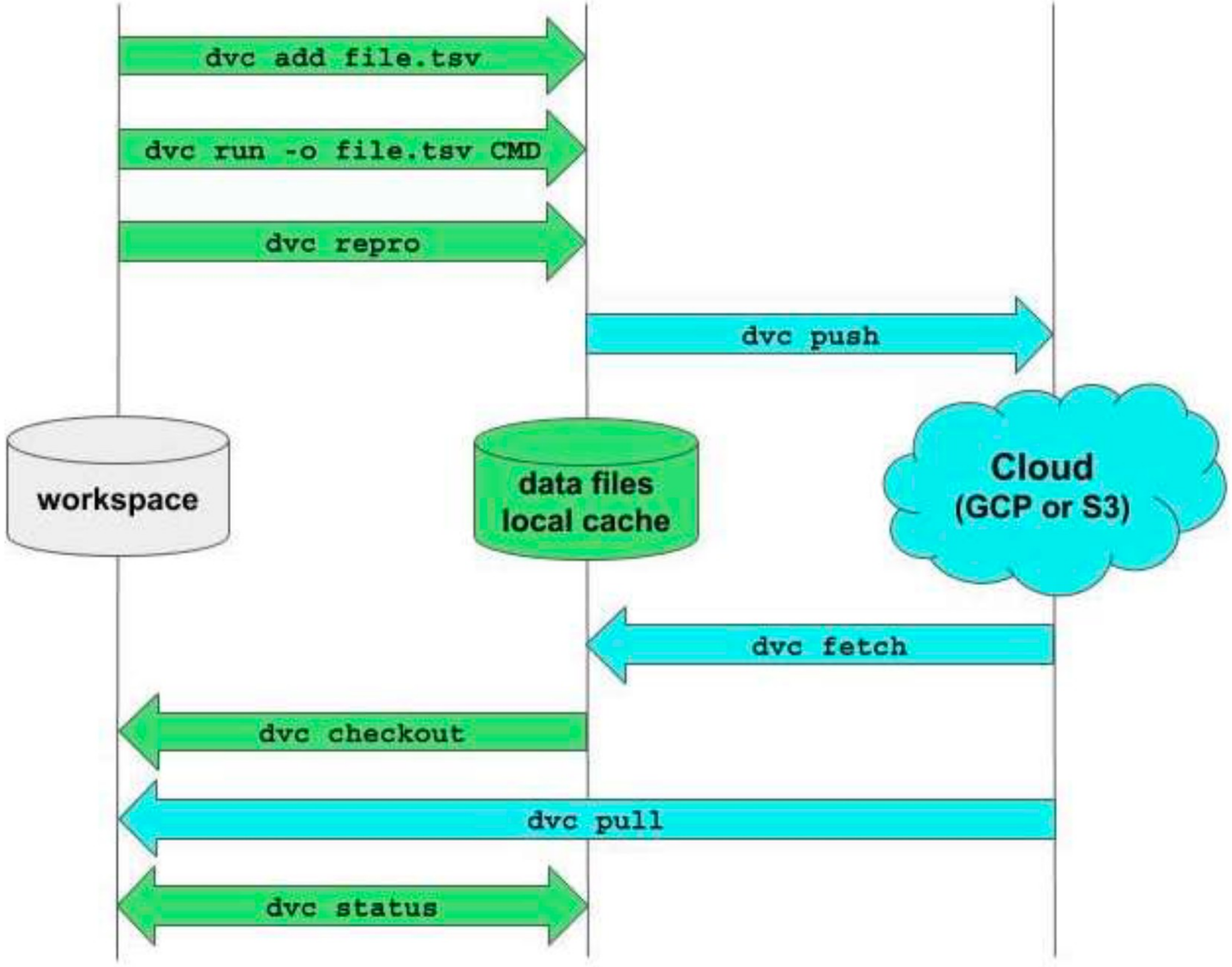
What DVC does?

- Experiment as commit/branch: Code + Data files.
- Large data files:
  - Local cache.
  - Optimized for 1Gb - 100Gb file size.
  - Data remotes: S3, GCP, SSH.
- Metrics per experiment.
- ML pipelines.
- Reproducibility.

Existing solutions

	Git-LFS	DVC
A single file size	< 2Gb	1Gb - 100Gb
Workspace size (all files)	Slow if 5Gb+	Unlimited
Not garbage collector for data	20 experiments by 5Gb each ~= 100Gb	Remove data files from some of experiments
Data storage	Proprietary and paid: only GitHub and GitLab.	S3, GCP or custom server (rsync, SFTP)

Workflow



https://www.slideshare.net/joshlk100/reproducible-data-science-review-of-pachyderm-data-version-control-and-git-lfs-tools

DVC: checkout and optimization

Optimizations:

- No data file copying - hardlinks copy instead.
- Checksum caching and timesteps tracking.
- Supports reflinks (CoW - Copy on Write) in modern file systems: BTRFS, ReFS, XFS.
  As a result: 100Gb data file checkout works instantaneously.

Benefits

- Integration with GIT
- Interlinked data-pipeline-output version control
- Easy to install
- Environment agnostic

A simple pipeline

Pipeline: images.zip → images/ → model.p → plots.jpg

S

```
(ML) ~/src/segment$ dvc run -d images.zip -o images unzip -q images.zip
Using 'images.dvc' as a stage file
Running command:
        unzip -q images.zip
(ML) ~/src/segment$ git status -s
 M .gitignore
?? images.dvc
```

Special DVC scenarios

- Tracking data files - like Git-LFS but S3/GCP/SSH backend.
- ML model deployment tool.
- Experimentation on HDFS/Apache Spark.

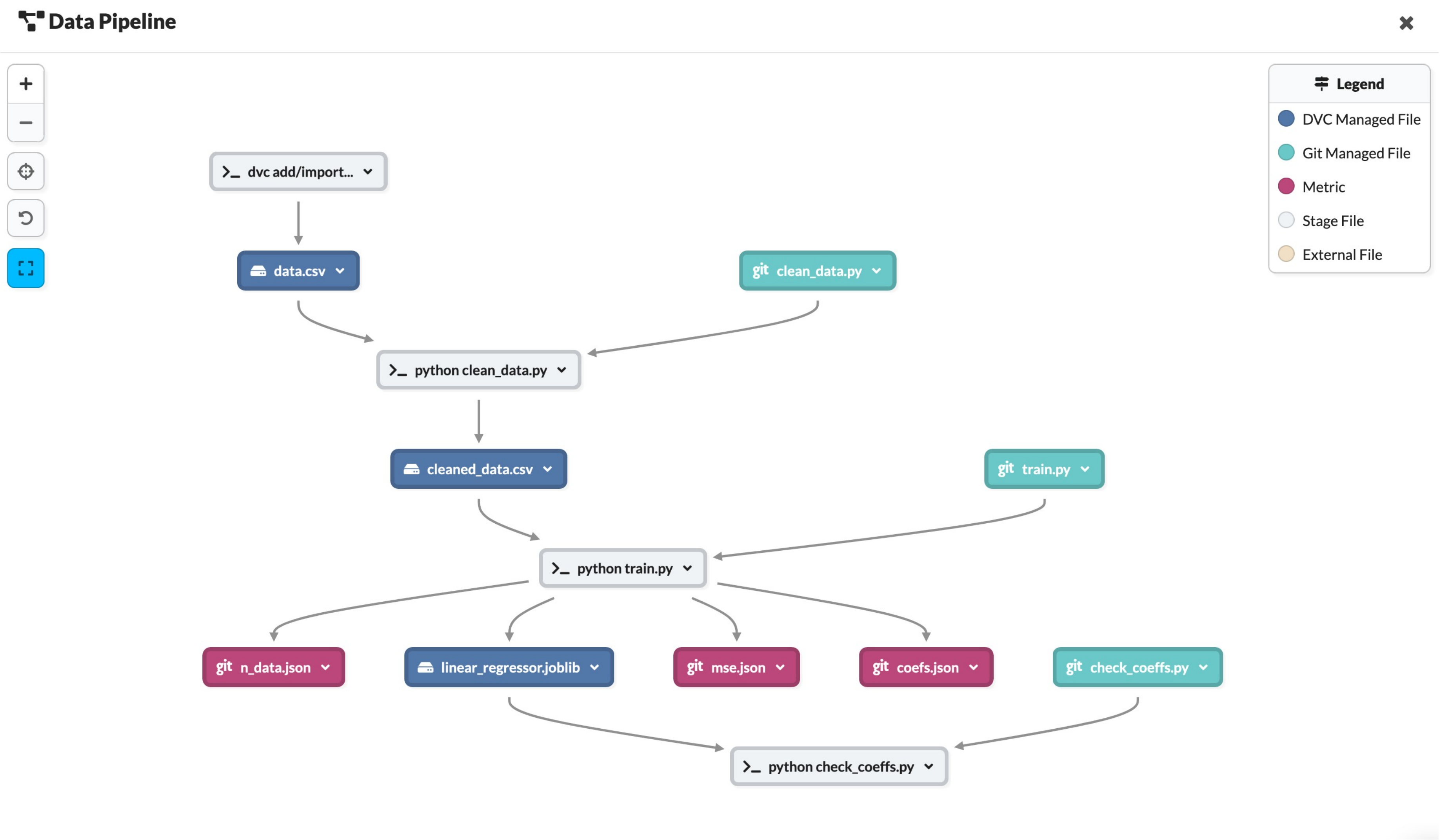
Reproducibility

DVC reproduces ML pipeline in a single command:

```
$ dvc repro
```

Any DAG (Directed acyclic graph) is supported.

DAGs Hub



https://dagshub.com/jmhsi/DVC_example

When you need DVC?

- DVC is a data science platform on top of open source stack.
- It uses some ideas from existing data science platforms but uses open source stack and Git as a foundation.
- Data science platforms helps creating ML projects in teams (3+ members).

Conclusion

- When you need Hadoop ecosystem, distributed DBs and Spark?

  It depends on how much data you have and how fast they are growing.
  If your data is so huge that can't be storing in a classic architecture and it difficult to processing them – it is a good idea to use the stack of tools for distributed data storaging and processing.

- When you need data and models management tools?

  It depends from how are you plan to deploy and maintain your ML model.

  If you are working with a lot of different experiments that use various data, especially if you are working on an ML problem in a team – you need tool like DVC that will prevent chaos in your data and models.

  And of course you need tools to manage ML pipelines, we will talk about in the next part.

Thank you!

Interested in collaboration?

anaderiRu

hse_lambda

Andrey Ustyuzhanin