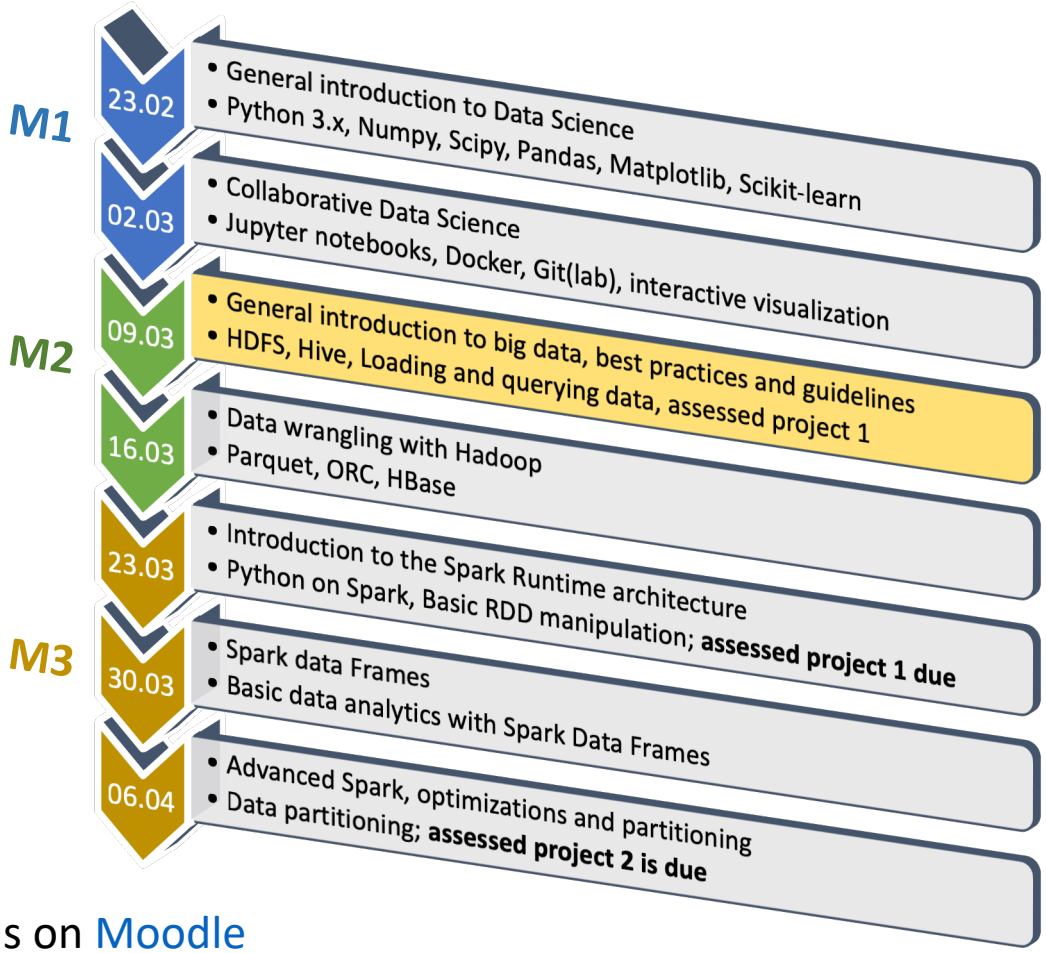
THE DATA SCIENCE LAB
General Introduction to Big Data

COM 490 – Spring 2022

Week 3

Agenda Spring 2022



*Details on Moodle

Week 2 – Questions?

Today's Agenda

- Big data journey
  - Bootstrapping your digitalization journey
  - A technology overview
    - Hadoop HDFS, MapReduce, and Hive
- Homework 1
- Exercises week 3
  - First steps with HDFS and Hive

Bootstrapping Your Digitalization Journey

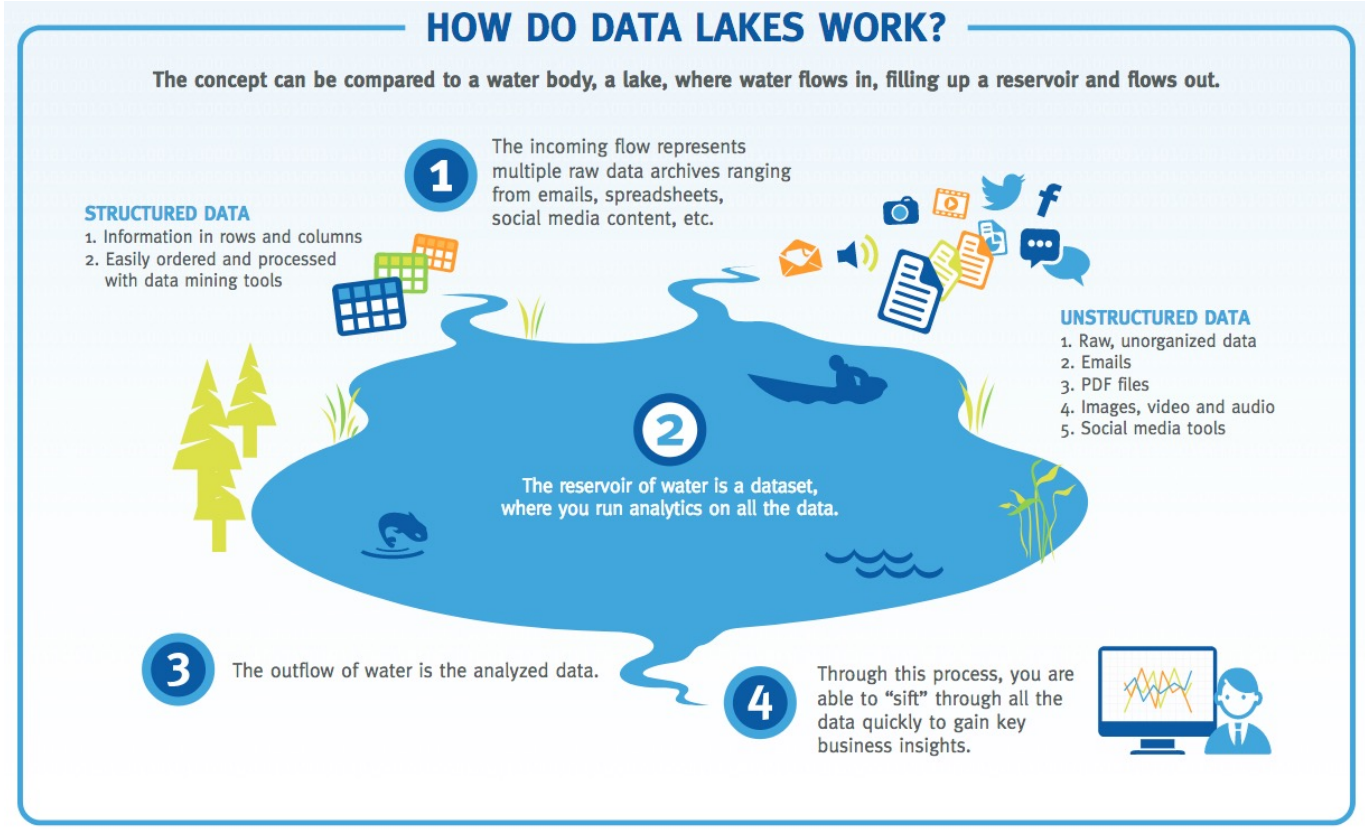
Big Data's 4Vs

How Shall I Start the Journey?

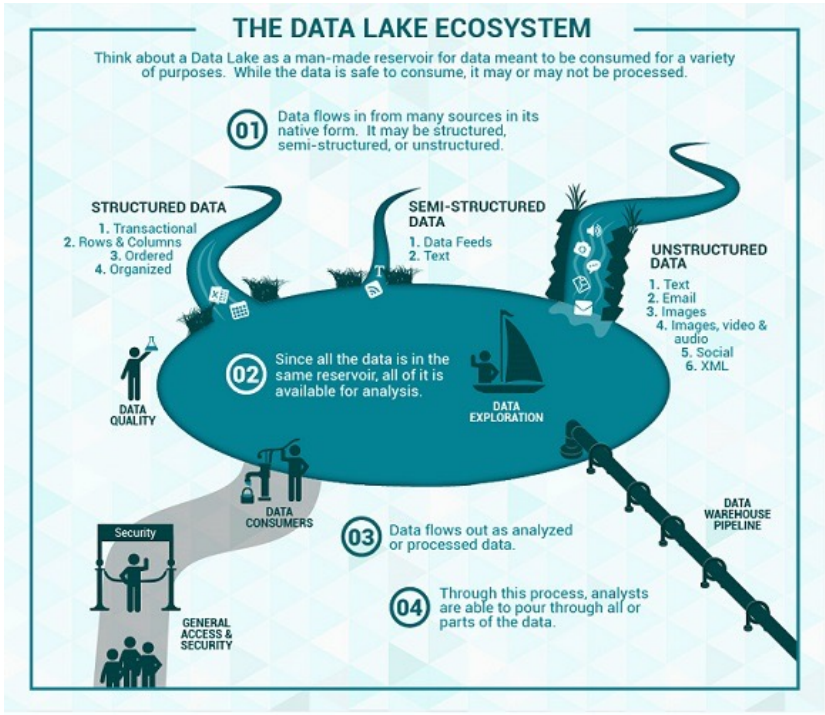
- ~~Easy! I'll go ahead and start building ML models, right?~~ Wrong!
- Instead, start by…
  - Ingesting data
  - Cleaning data
  - Integrating data
- In most companies, this actually represents **75%** of the work
- Only then can you make the last **25%** (*analytics*) successful
- Build a **data lake** to tame your data first!

What is a Data Lake?

Conceptually: solution to co-locate and share data in its native form

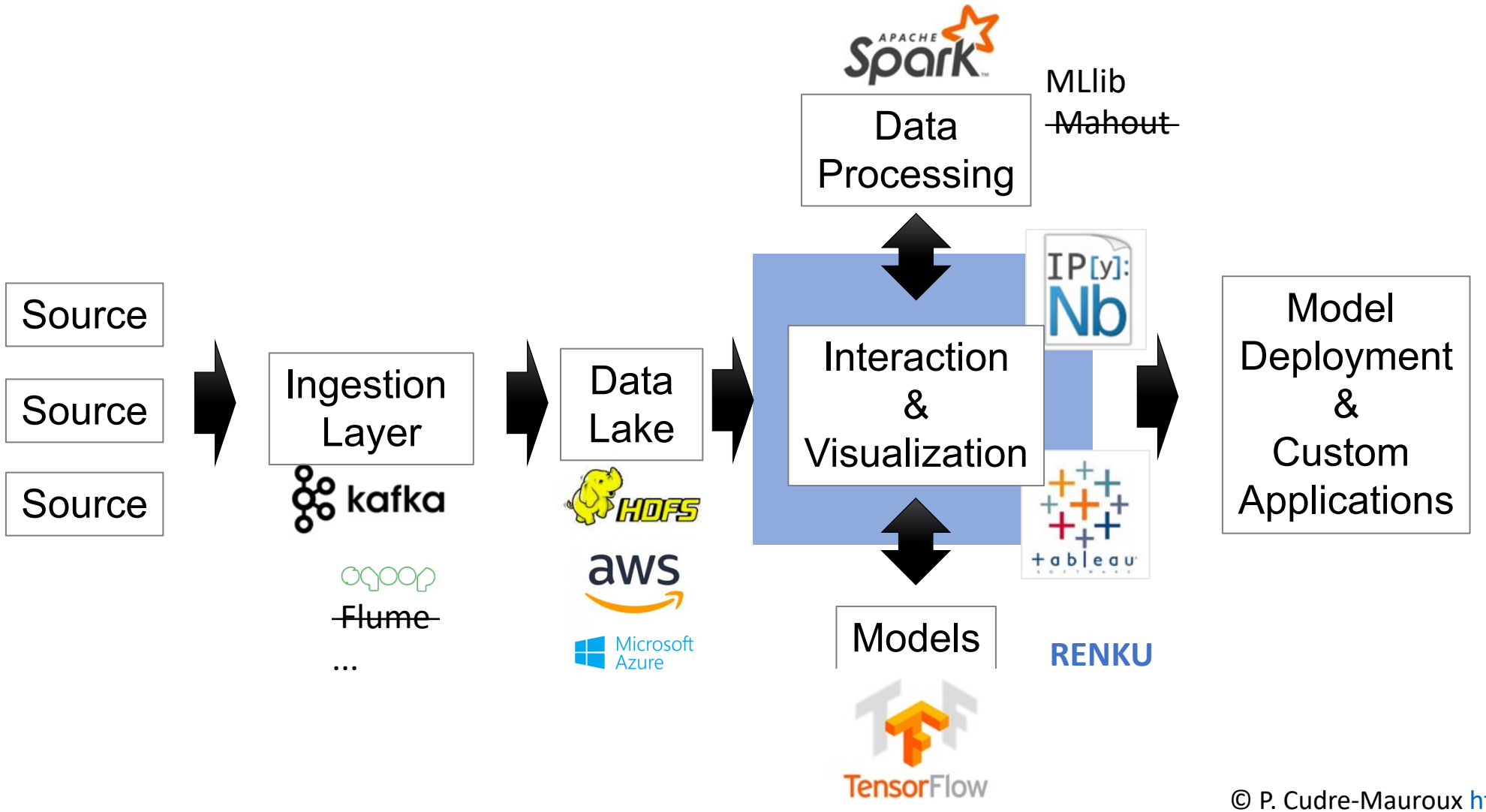


© Zaloni



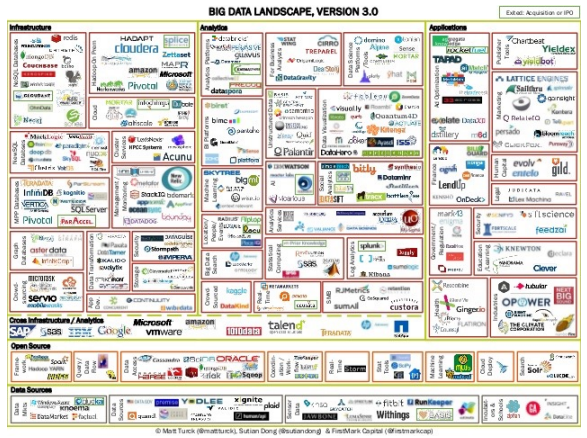
© DunnSolutions

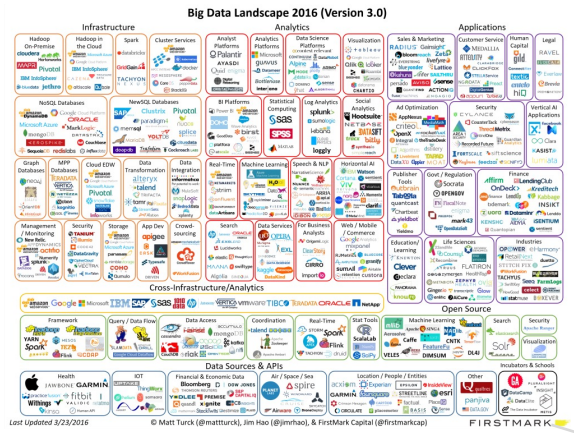
Typical Big Data Architecture (2021)

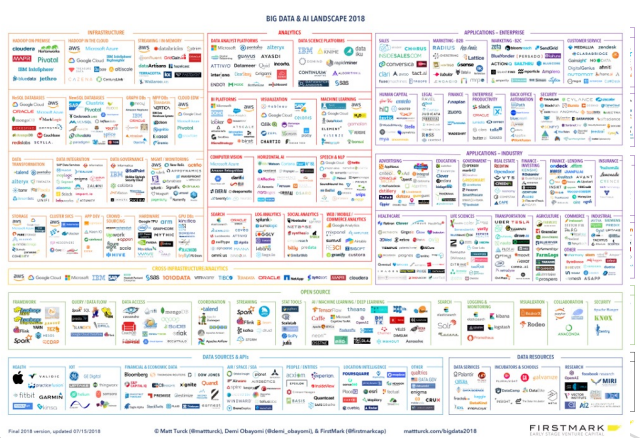


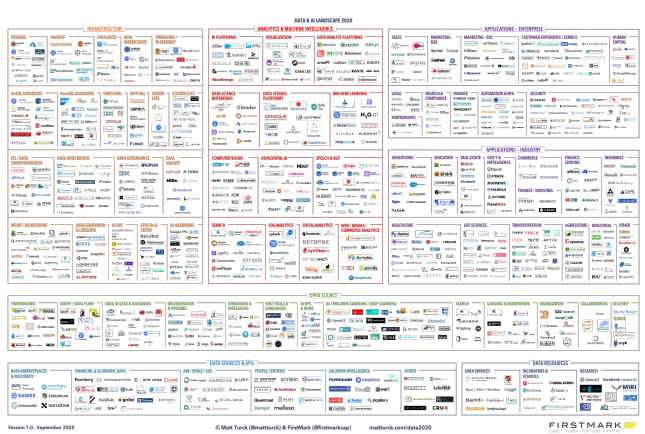
© P. Cudre-Mauroux https:\\exascale.info

Big Data Landscape Evolution – A Moving Target









2014 2016 2018 2020

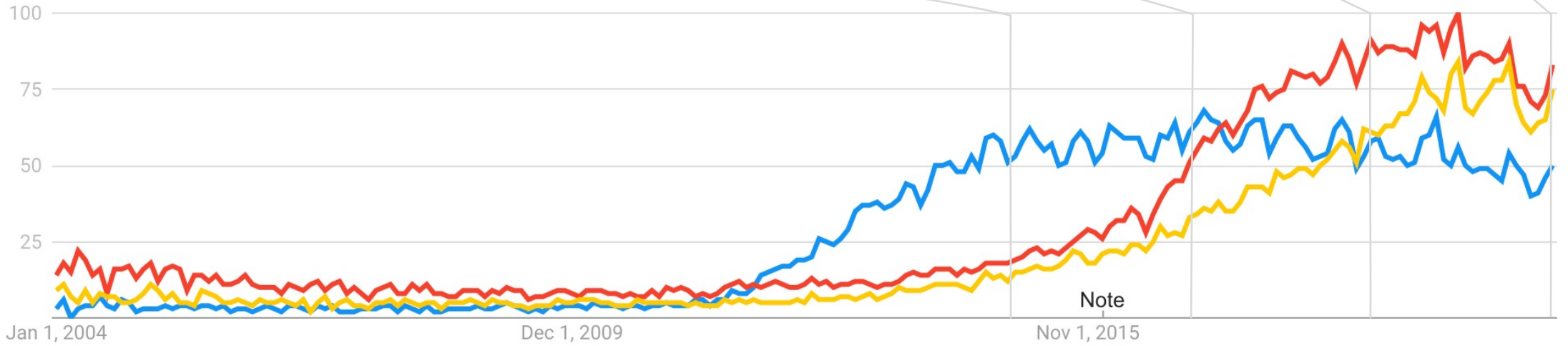
Google Trends

Big Data

Machine Learning

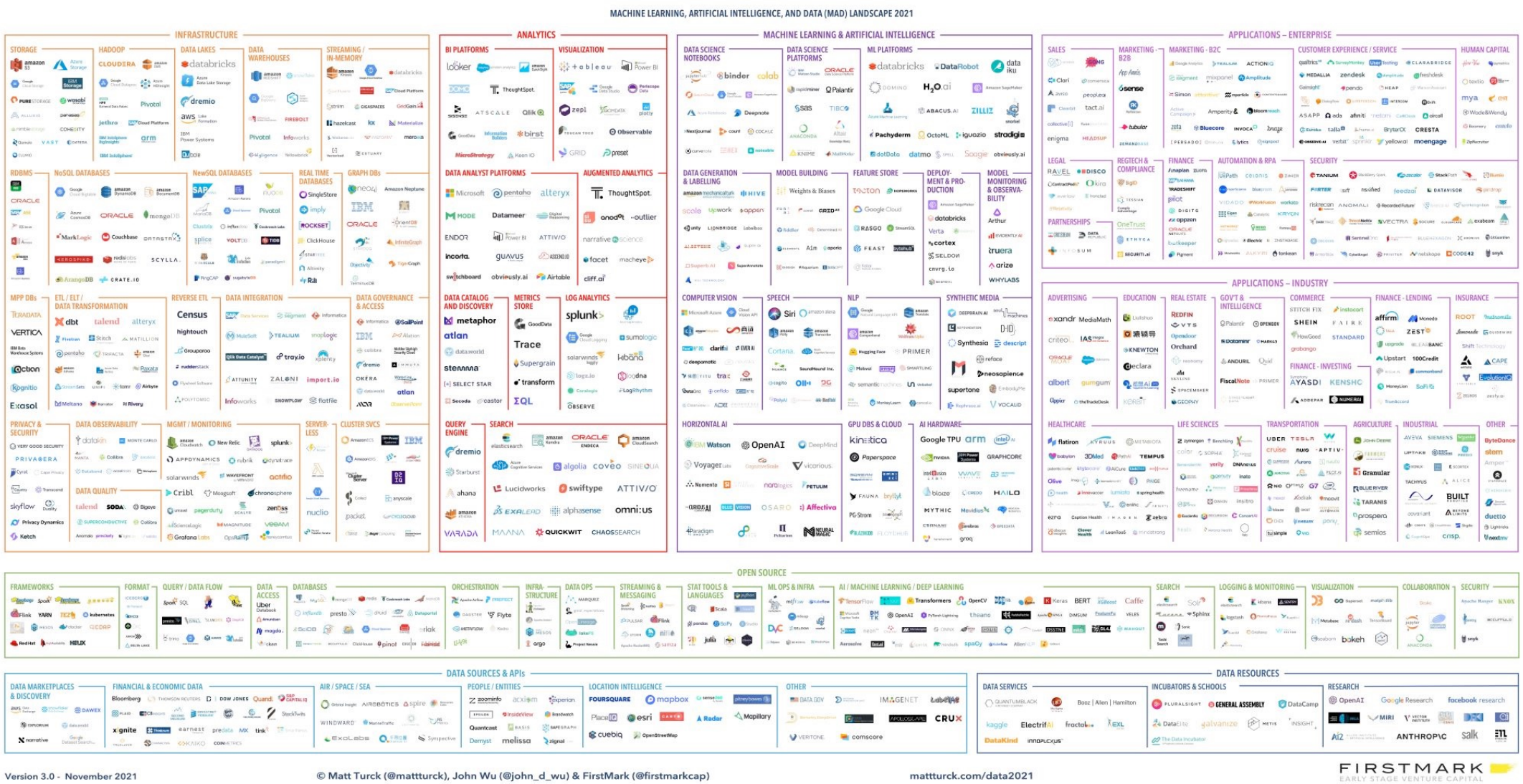
Data Science

Average



(Sources: Matt Turck - http://mattturck.com ; trends.google.ch search terms, all categories)

The 2021 ML, AI and Data (MAD) Landscape

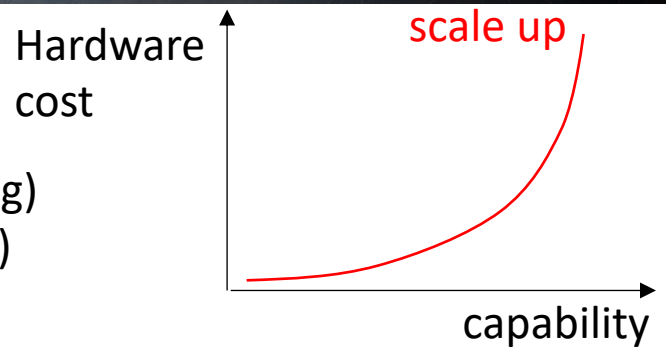


(Sources: Matt Turck - http://mattturck.com)

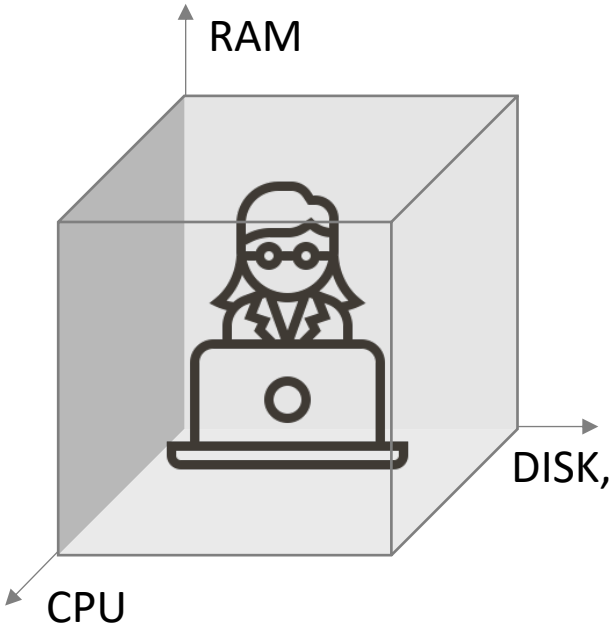
Today's Objectives

- Familiarize yourselves with the Big Data ecosystems
  - Find your way in the Big Data jungle, explore it more efficiently

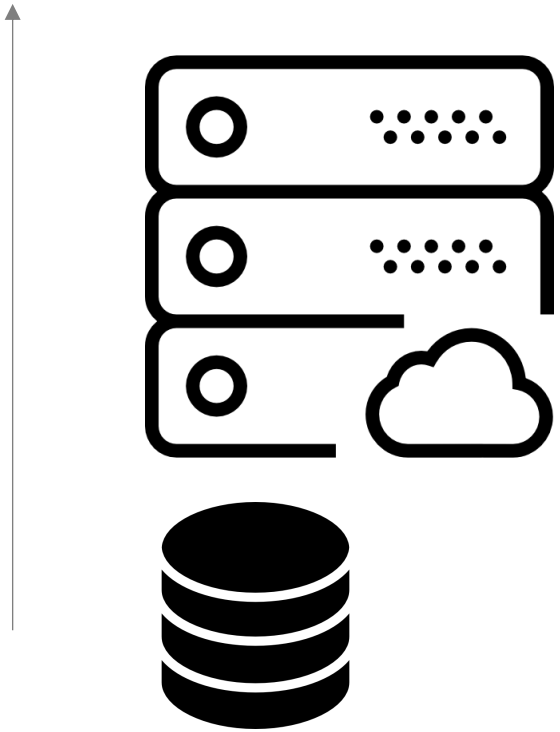
Addressing the Big Data Challenges



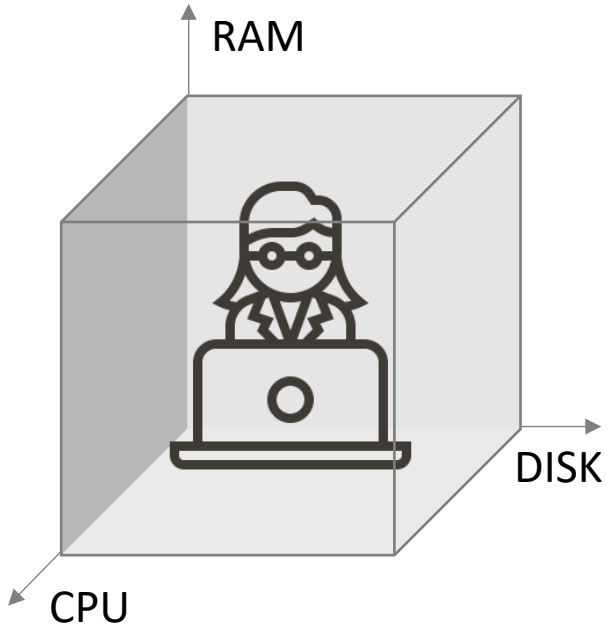
Bigger machines: **scale up** (a.k.a vertical scaling)
… the High Performance Computing way (HPC)



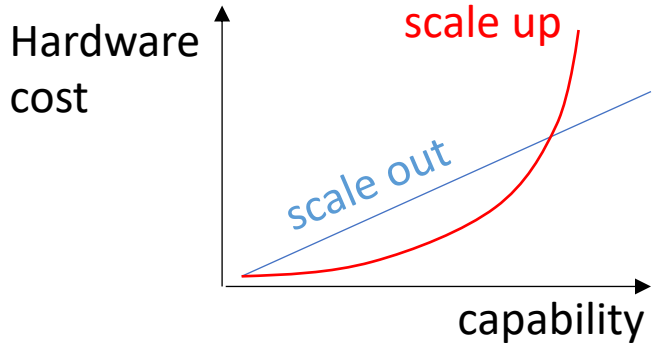
Break the CPU/RAM/Disk bounds

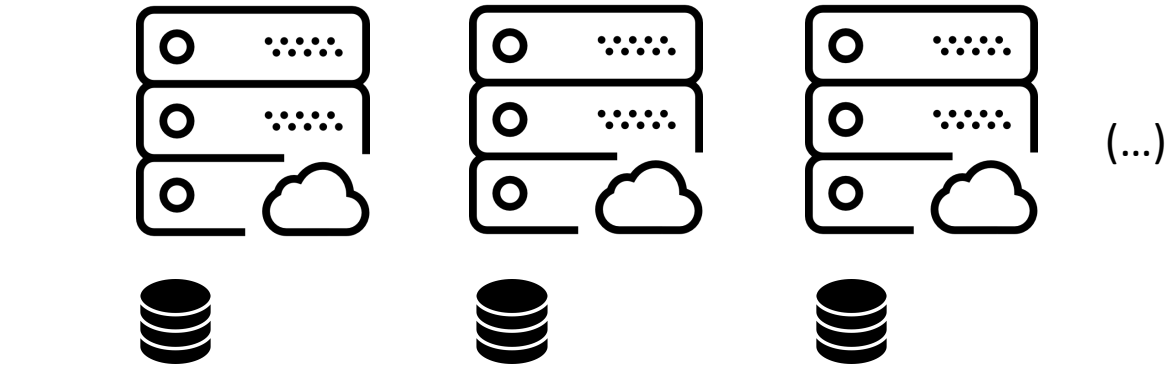


Addressing the Big Data Challenges



Break the CPU/RAM/Disk bounds





More machines: **scale out** (a.k.a horizontal scaling)
… the commodity hardware (cloud) way

Addressing the Big Data Challenge

- Horizontal scaling (scaling out) entails distributed computing across a large number of compute servers
- Challenges of distributed computing are:
  - The same code should transparently work on 1, 10, or 10,000 servers
    - Assume problem can be broken down into chunks and each chunk calculated locally
    - The data must be accessible from anywhere
  - Fault tolerant and high availability
    - The systems must survive one or more server failures with no impact on operations
  - Resource utilization must be optimized
    - Minimize hot-spots with a good load-balancing strategy
    - Bring compute to data
  - The systems should support elastic scaling
    - Add/remove machines without requiring maintenance down-times

CAP Theorem of Distributed Data Stores

"It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees"

- **C**onsistency: All clients see the same data at the same time [(1)]
- **A**vailability: Every RW request succeeds (even if a node goes down. i.e. partition happens)
- **P**artition tolerance: C ~~and~~/**or** A holds despite messages being dropped or delayed

In distributed systems (**scale out**), partition tolerance (**P**) is unavoidable.

We therefore must choose between Consistency (**C**+**P**) or Availability (**A**+**P**):

- C+P: Cancel the operation and thus decrease the **A**vailability but ensure **C**onsistency
- A+P: Proceed with the operation and thus provide **A**vailability but forfeit **C**onsistency

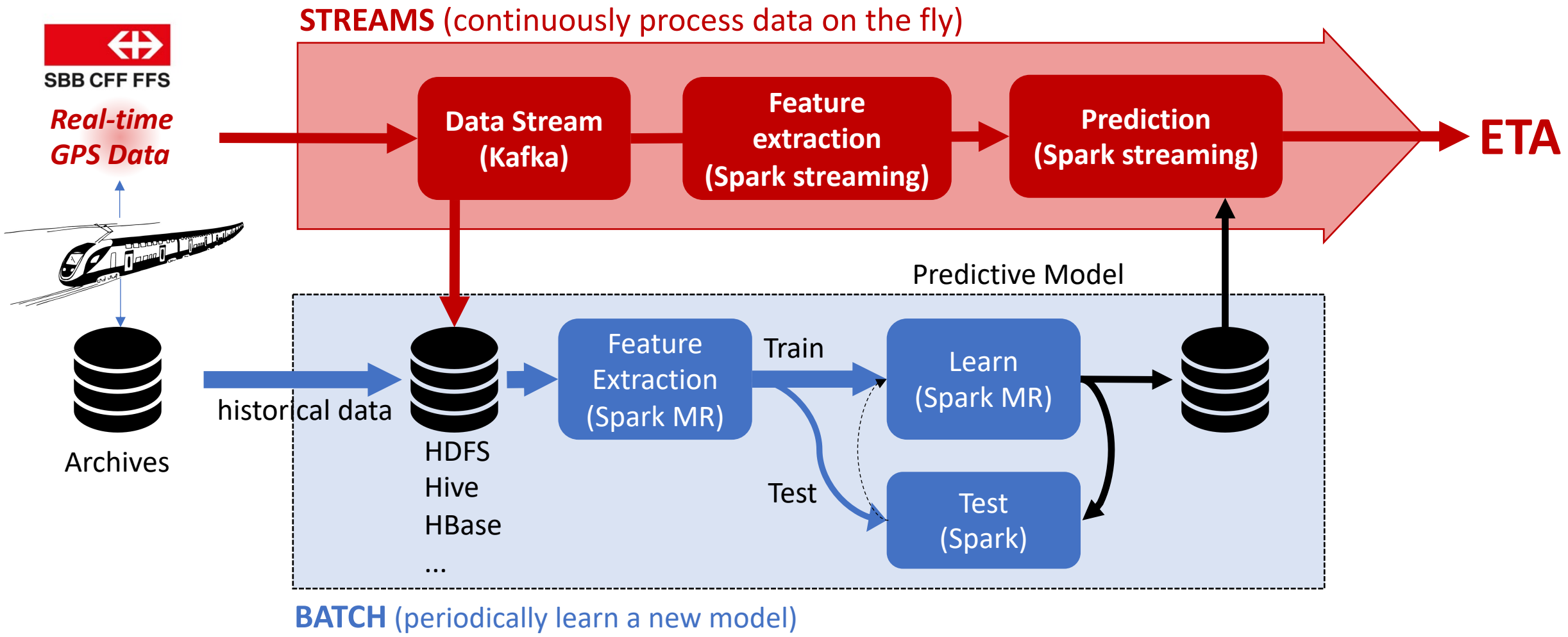
[(1)]Defined differently from strict consistency of **A**tomicity **C**onsistency **I**solation **D**urability (ACID transaction)

The Clash: Should I BATCH, or should I STREAM?

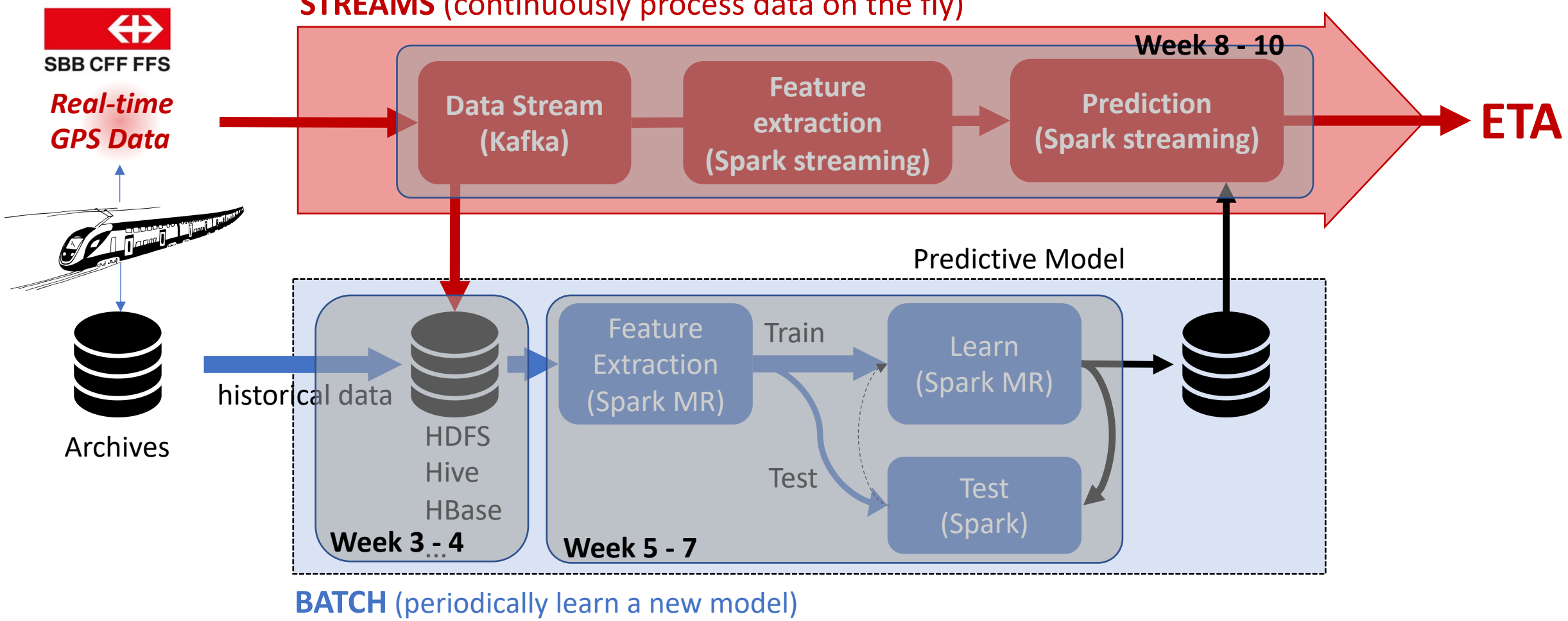
- Your application can wait until all information is available for a complete answer? **BATCH**
  - AKA: **Data at rest**
  - Operates on finite size data sets, that terminate after all data has been processed
  - **Hadoop MR**, **Spark**

- Your application needs results as soon as more information becomes available? **STREAMS**
  - AKA: **Data in motion**, or **Fast data**
  - Continuous computation that never stops, processes infinite amount of data on the fly
    - Designed to keep size of in-memory state bounded, regardless of how much data is processed
  - Operates on small time windows
  - Update the answer as more data becomes available
  - Often used in critical systems, where fast response time to event is essential
  - **Spark Streaming**, **Flink**, **Storm**, **Kafka Streams**

STREAMS and BATCH Illustrated



STREAMS and BATCH Illustrated



Addressing the Big Data Challenge - Technologies

DASK

https://dask.org/

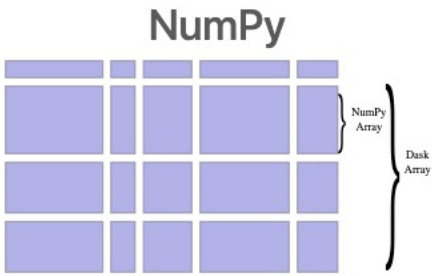
Familiar for python users

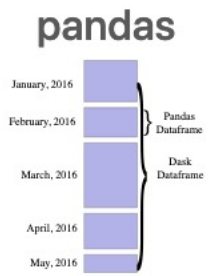
```
# Arrays implement the NumPy API
import dask.array as da
x = da.random.random(size=(10000, 10000),
                     chunks=(1000, 1000))
x + x.T - x.mean(axis=0)
```

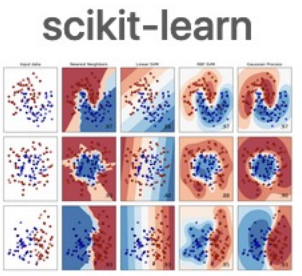
```
# Dataframes implement the pandas API
import dask.dataframe as dd
df = dd.read_csv('s3://.../2018-*-*.csv')
df.groupby(df.account_id).balance.sum()
```

```
# Dask-ML implements the scikit-learn API
from dask_ml.linear_model \
  import LogisticRegression
lr = LogisticRegression()
lr.fit(train, test)
```

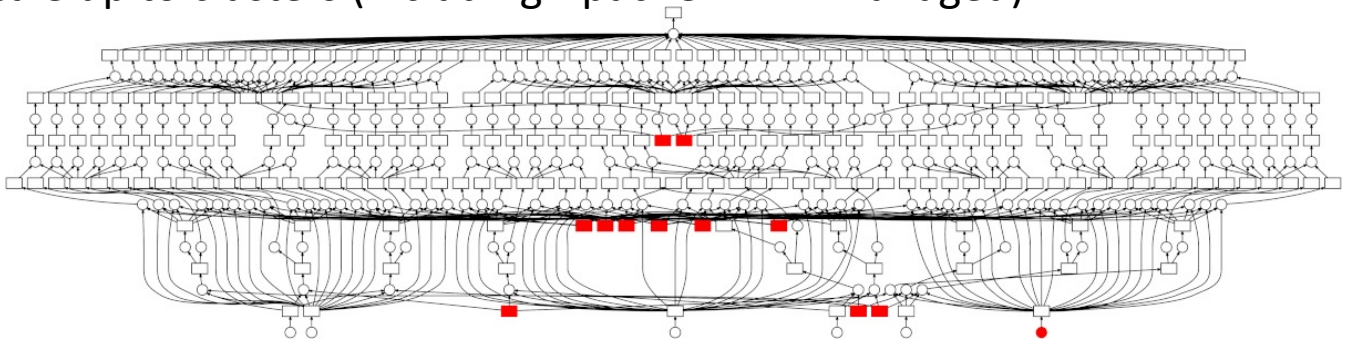
Integrate with existing python projects







Scale up to clusters (including Apache YARN-managed)

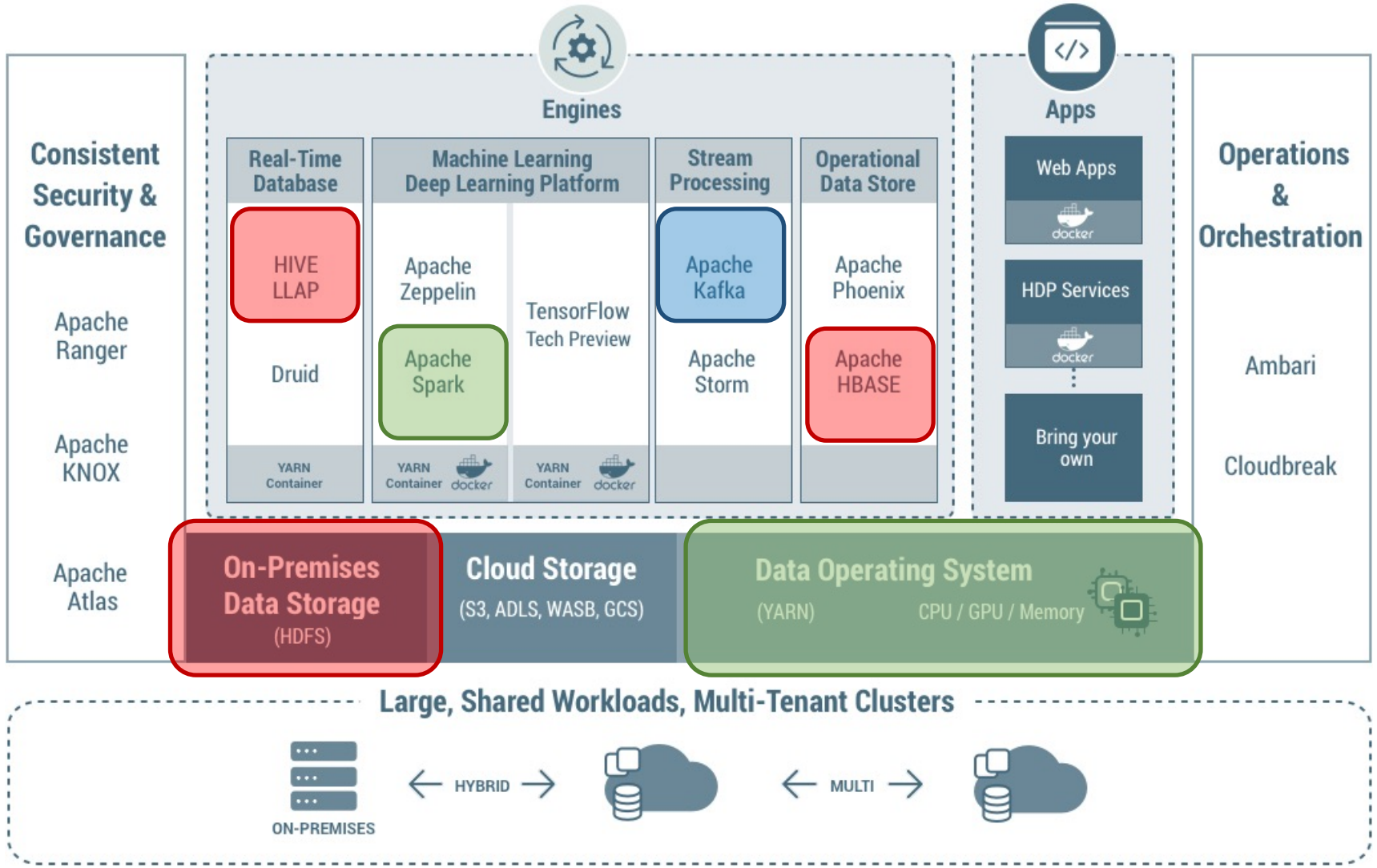


Addressing the Big Data Challenge - Technologies

Weeks 3-4

Weeks 5-8

Weeks 9-10



https://www.cloudera.com/products/hdp.html

Technology Overview

Hadoop Distributed File Systems - HDFS

Map Reduce

Hive warehouse

Hadoop Distributed File Systems Top Features

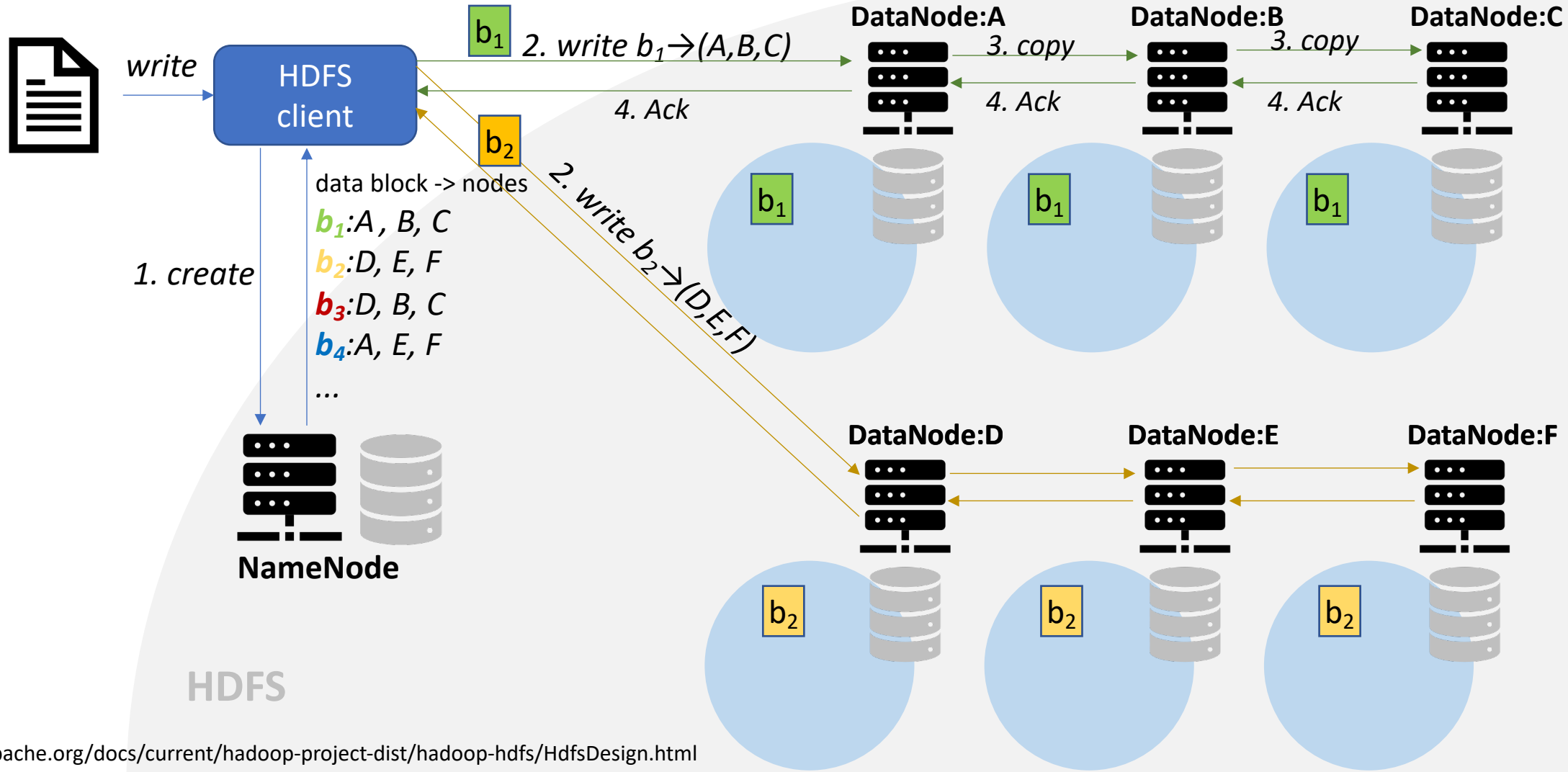
- **Large Data Sets**
  - Size of a file only limited to HDFS cluster capacity, and can exceed the size of its largest disks
- **Horizonal scalability** (cost effective)
  - Need more space? add more machines with more disks
- **Fault Tolerance** & **High Availability**
  - Redundance guarantees that if a disk fail, copies of lost data blocks can be found on another disk
- **High Throughput**
  - Support parallel file I/O and processing with “end-to-end” partitioning from input data to results
- **Data Locality**
  - Moving computation to the data instead of moving data to the computation (less network bottleneck)
- **Data Integrity**
  - Checksums are used to detect corrupted data
- **Data security**
  - Access Control Lists (ACL)
  - Transparent end-to-end encryption (multi encryption zones, i.e. multi-tenant)

Hadoop Distributed File Systems Essentials

- Main Concepts
  - **HDFS** is a DISTRIBUTED (networked) cost-efficient file systems
  - **NameNode**: (master node) manages namespace, must have at least one, preferably two for high availability
  - **DataNode**: (worker nodes) serves the data, one per server
  - **Data blocks** are in units of 128MB max (default Hadoop 2)
    - E.g. 500 MB file is 3 x 128 MB blocks + 116 MB block
  - **Write-once** & read many times: a file cannot be modified in place, it must be replaced (but append is possible)
  - **Redundancy**, all blocks replicated x3 by default (200% overhead **)
    - Redundancy against failures
    - Statistically easier to move computation next to the data and load-balance the CPU usage
  - HDFS command line, a POSIX-like file systems interface (Hadoop2):
    - `hdfs dfs [--help]`

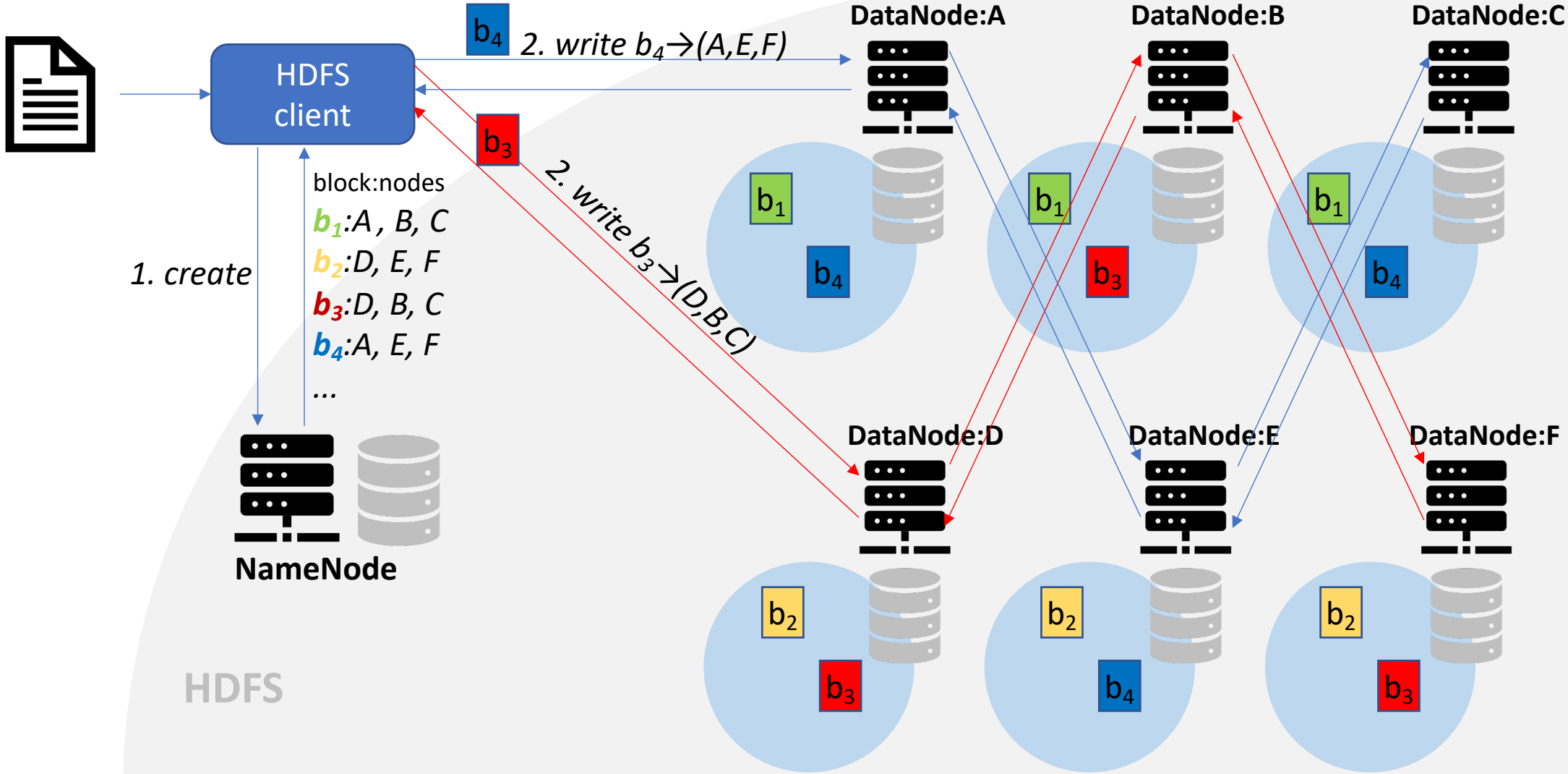
** more recently Hadoop 3 uses code erasure with 50% overhead

Hadoop Distributed File Systems (HDFS) Essentials

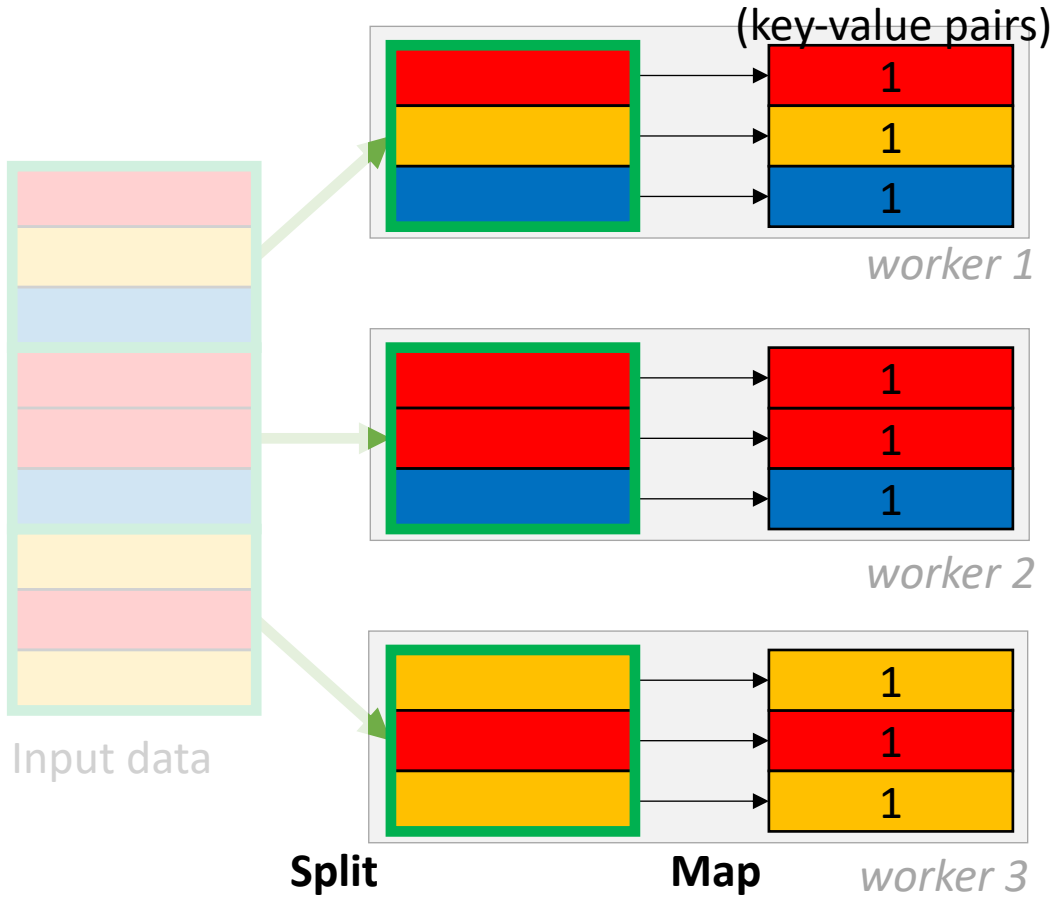


https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

Hadoop Distributed File Systems (HDFS) Essentials



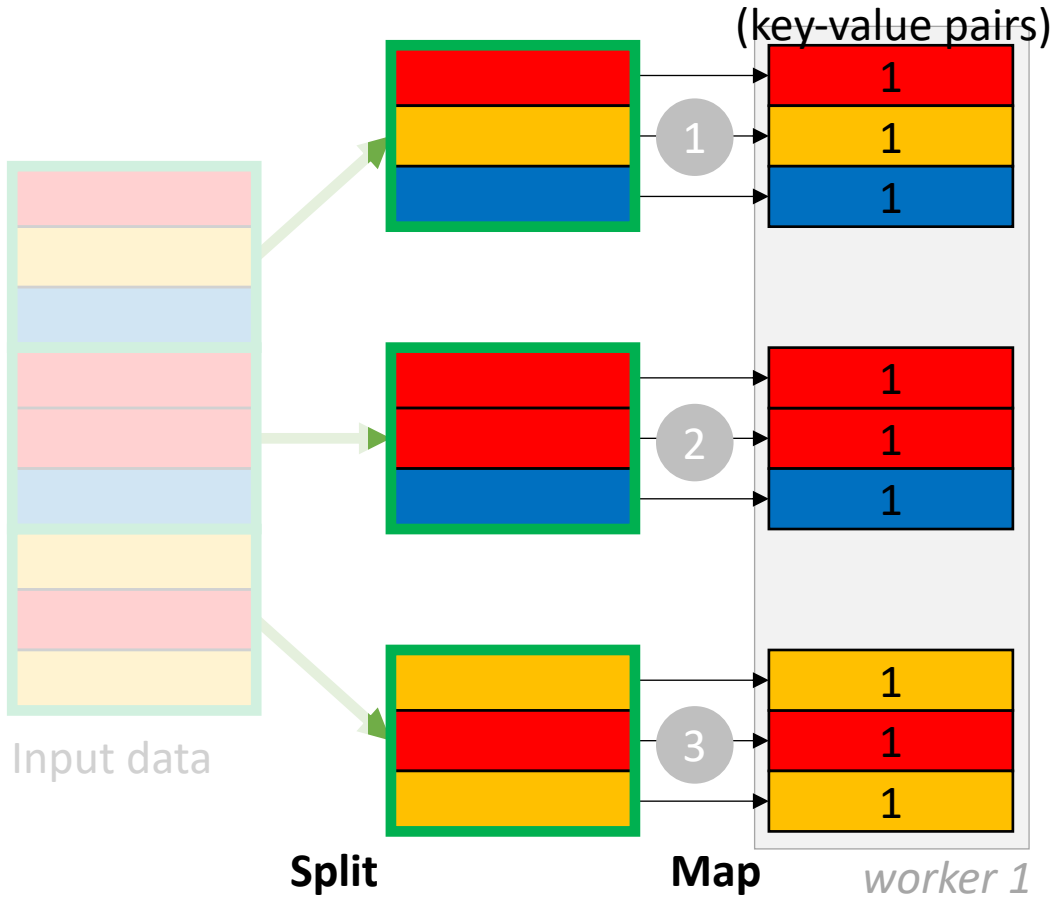
MapReduce in a Nutshell



- **HDFS data** is already split into blocks !
  - "Mapping" can be done in parallel on worker nodes placed closest to datanodes with blocks

Intermediate results on HDFS file systems or in-memory

MapReduce in a Nutshell

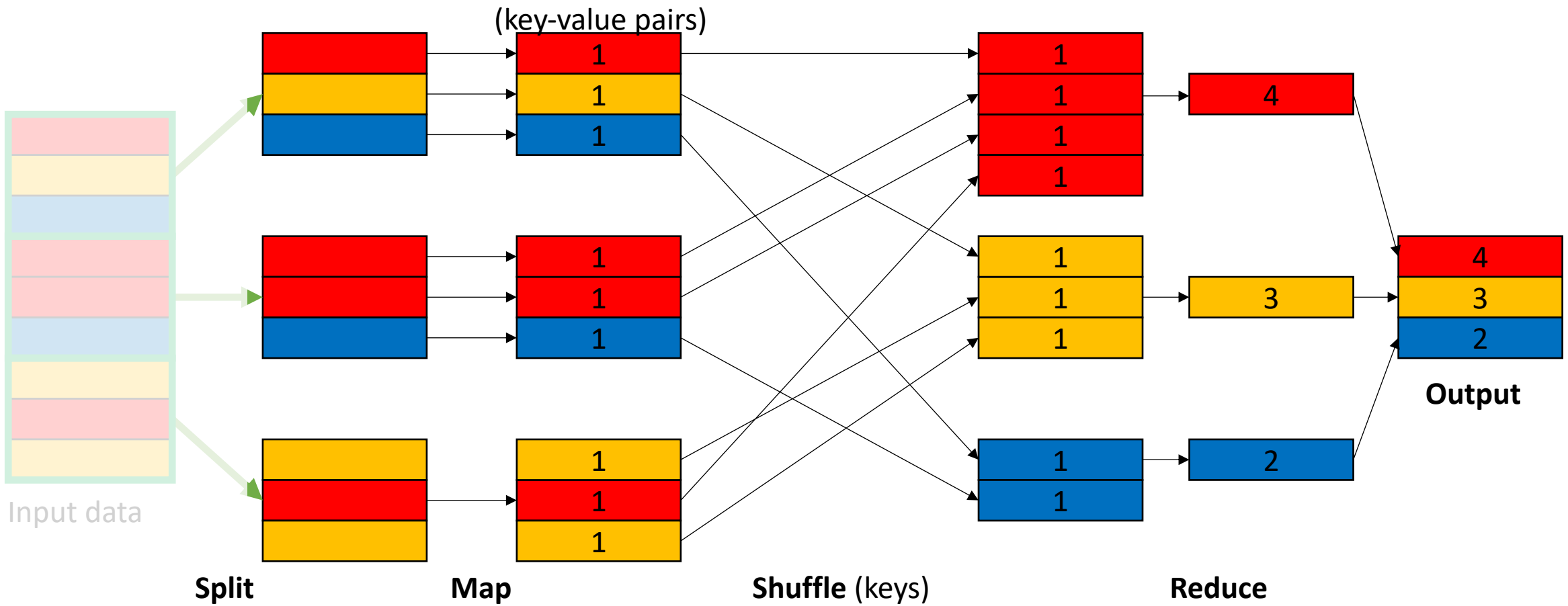


- **HDFS data** is already split into blocks !
  - "Mapping" can be done in parallel on worker nodes placed closest to datanodes with blocks

- **Or** ... , maybe not in parallel if input data is compressed
  - Not all compression algorithm are splittable
  - If not splittable, blocks must be processed sequentially
  - **Splitable**:
    - BZIP2
    - LZO (indexed)
  - **Not-splittable**:
    - GZIP
    - Snappy (generally)
    - LZO

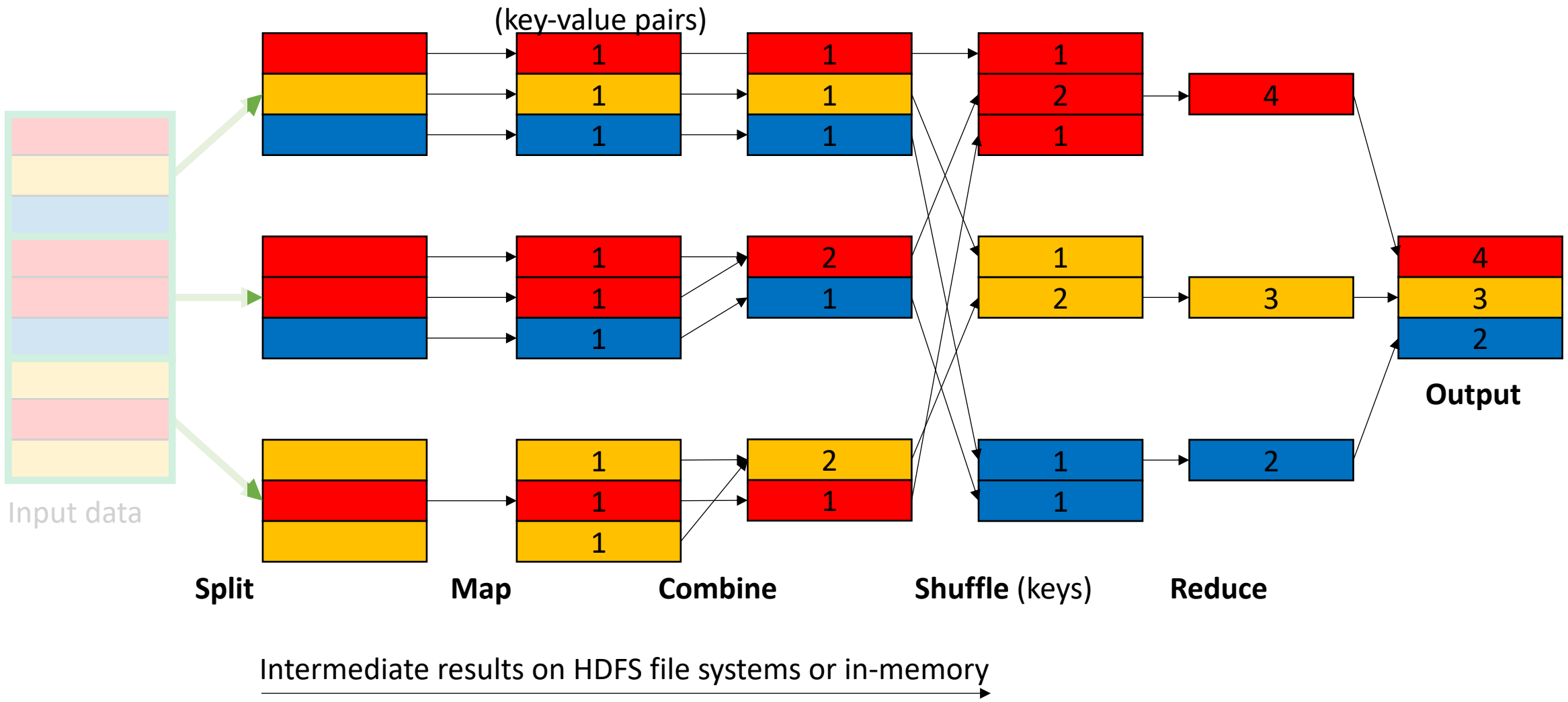
Intermediate results on HDFS file systems or in-memory

MapReduce in a Nutshell

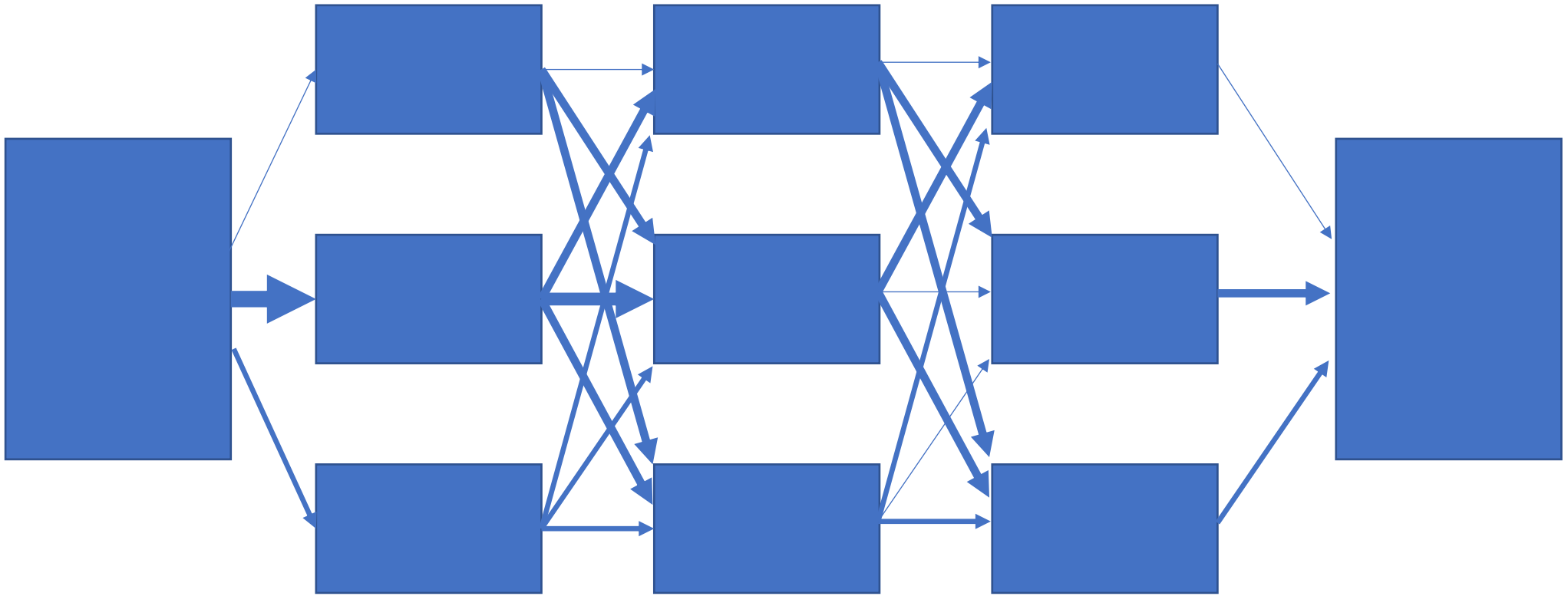


Intermediate results on HDFS file systems or in-memory

MapReduce in a Nutshell

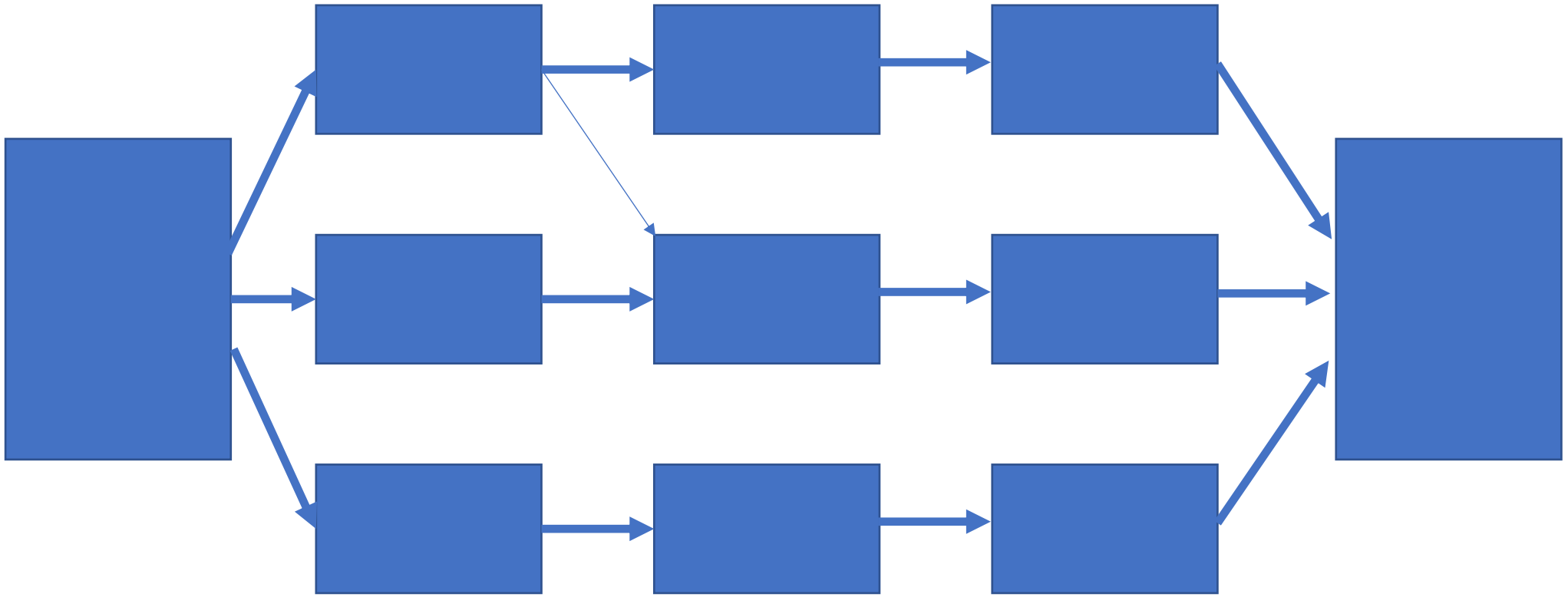


MapReduce gotchas



Shuffling is the network bottleneck of MapReduce operations, because placement of "reducers" cannot be optimized based on data locality.

MapReduce Best Practices



Optimization starts with good data partitioning practices to better balance the load (on CPU, RAM), and minimize data shuffling (network bottlenecks)

Popular HDFS Storage Format

- Storage format - this is how data is physically stored in the HDFS blocks

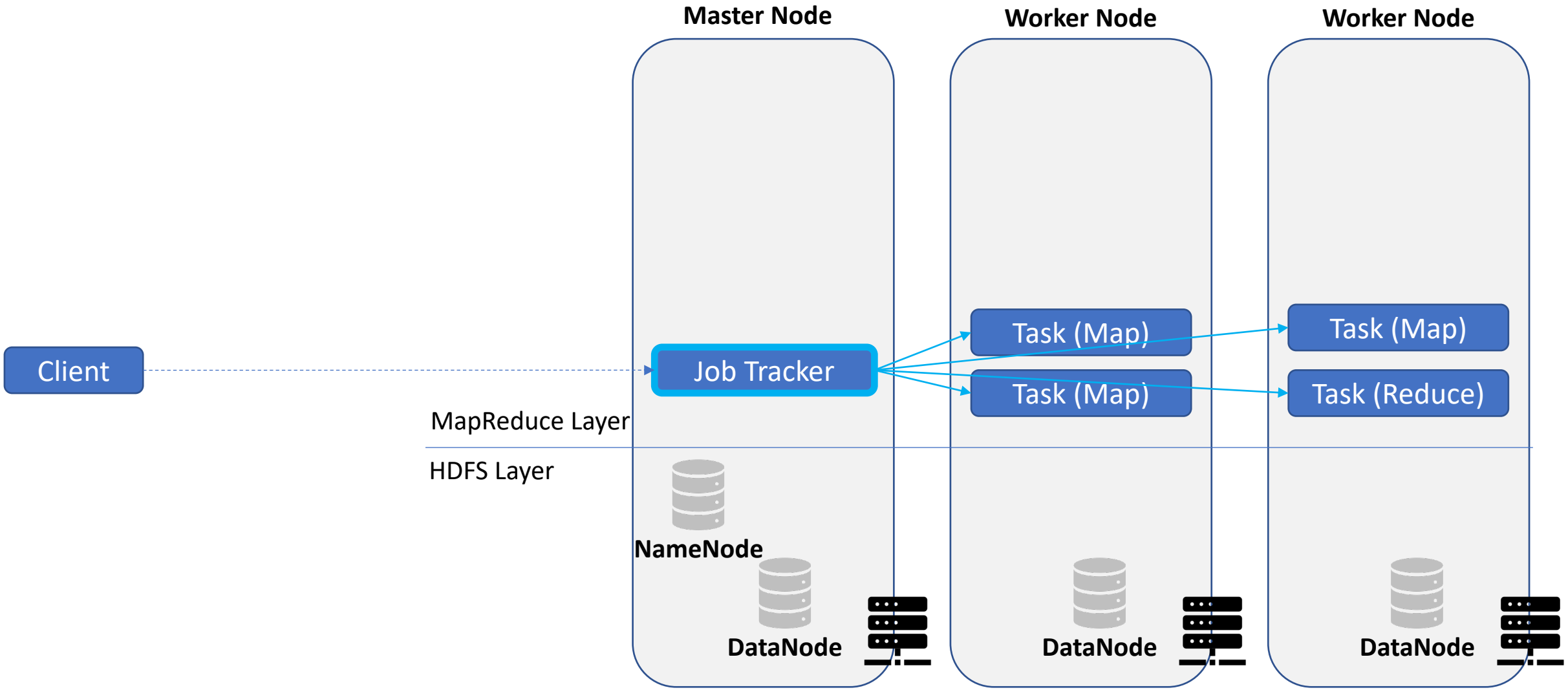
- **Plain text (**csv, json, xml, …),
  - Row-oriented (most of the time)
  - This often the format you get from external sources
  - Best for OLTP
  - Batch and (often) stream processing
  - Splittable (if one line per record)

- **Parquet**
  - Column-oriented, best for OLAP
  - Integrated compression: None, SNAPPY, ZLIB
  - Splittable
  - For write once, read many (WORM)
  - Batch processing only

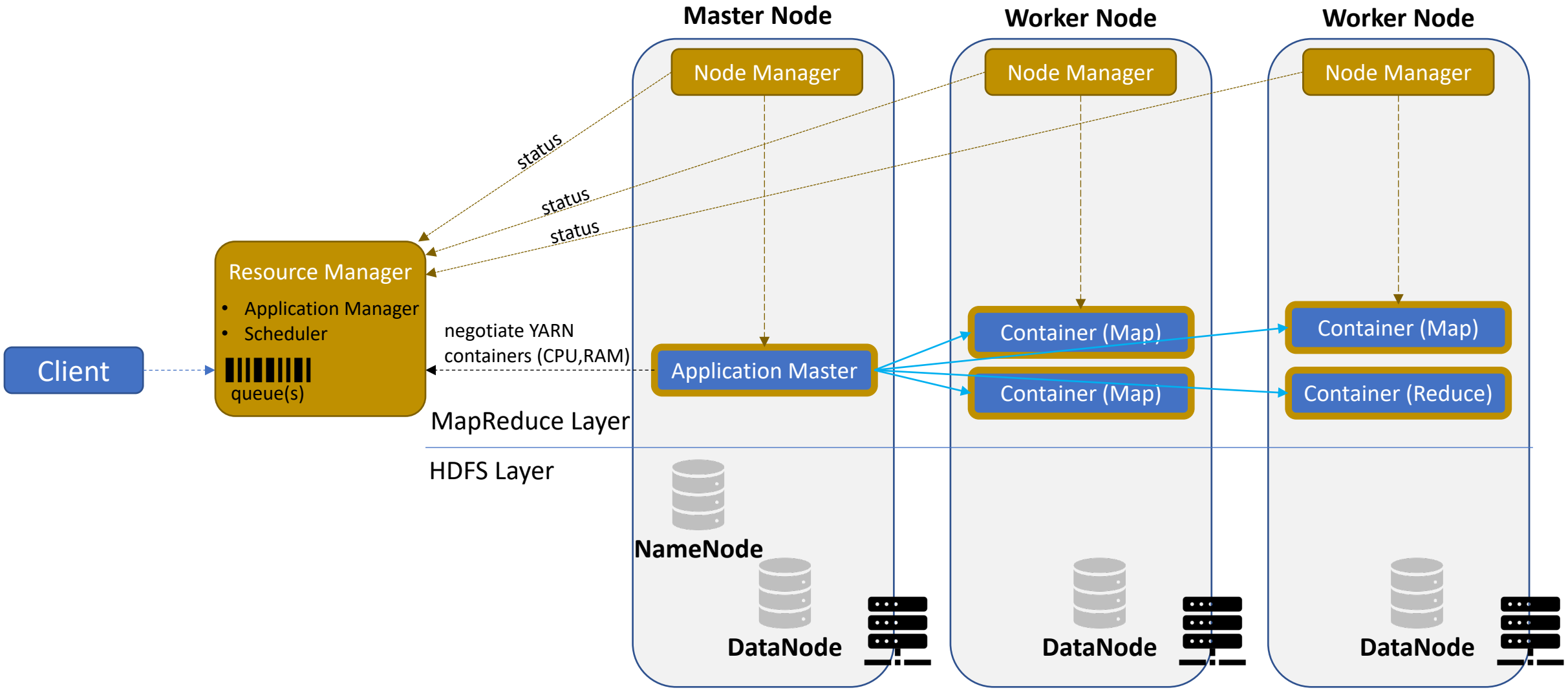
- **ORC**
  - Column-oriented, in collections of rows (stripes of 250MB), best for OLAP
  - Indexed
  - Splittable (per stripes).
  - integrated compression: None, SNAPPY, ZLIB
  - Optimized for WORM
  - Batch processing only

- **Avro**
  - Row-oriented,
  - Splittable
  - Support block level compression
  - Best for OLTP
  - Support schema evolution

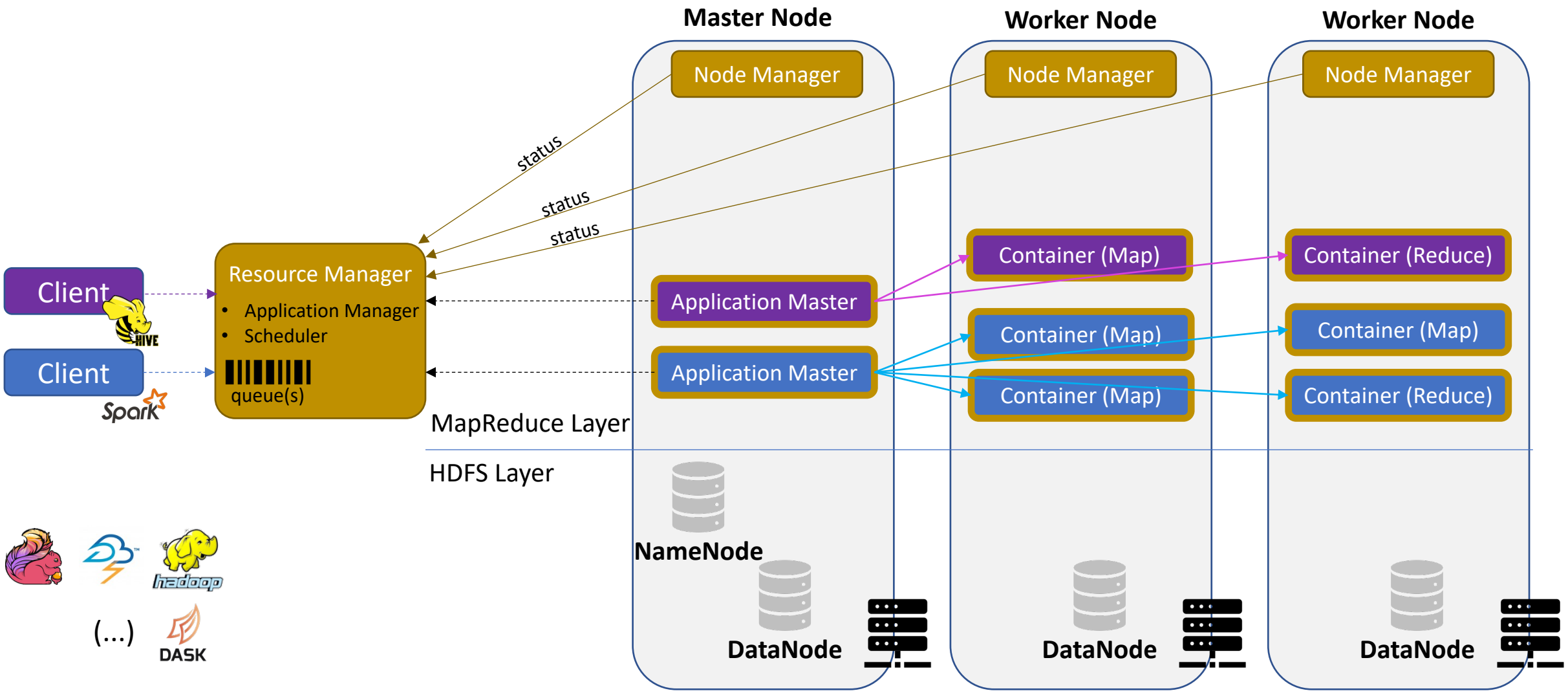
MapReduce Architecture (1.0)



MapReduce Architecture with YARN (2.0)



MapReduce Architecture with YARN (2.0)

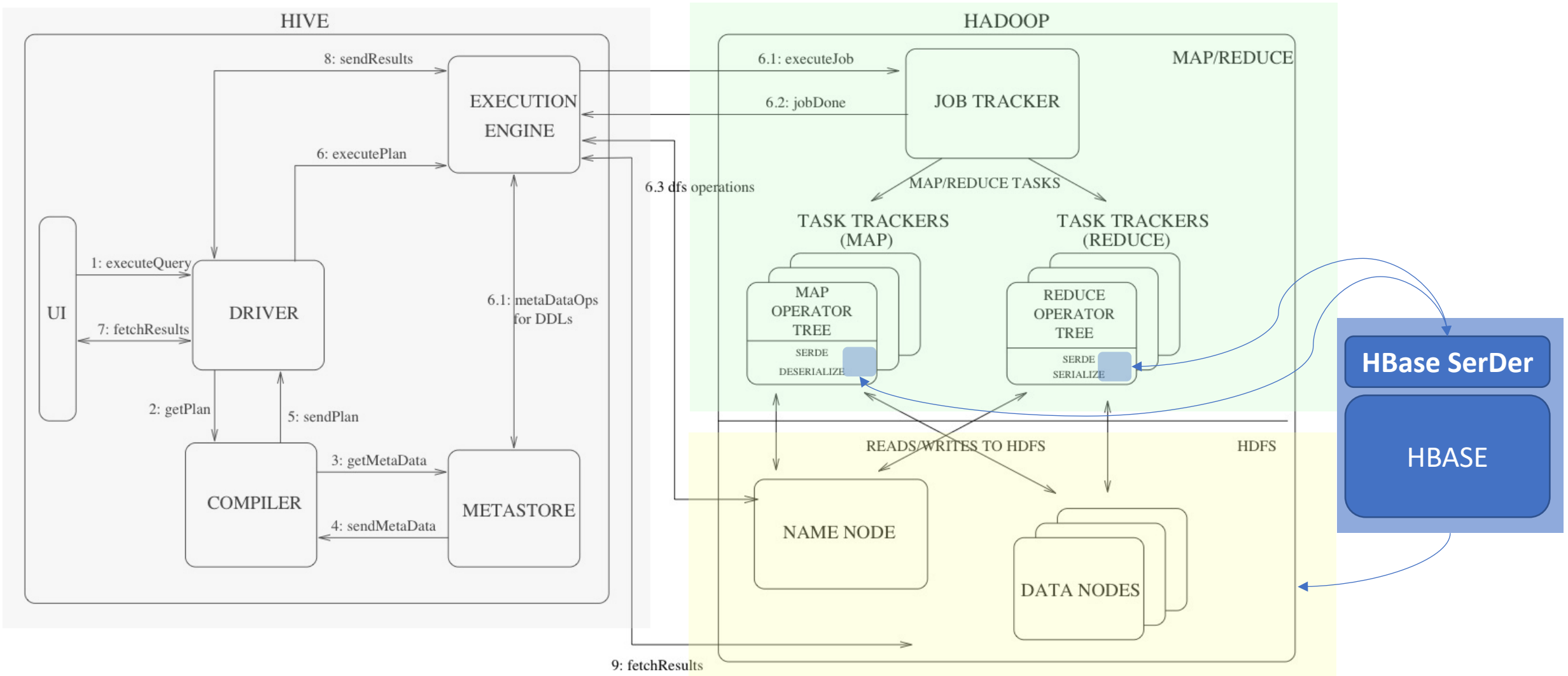


HIVE Hadoop Data Warehouse

- **Important Concepts**
  - Data warehouse software
  - Facilitates reading, writing, and managing arbitrarily large datasets
  - Built on top of **Hadoop**
  - Easy SQL-like interface to query to data stored in **HDFS** or other data storage systems (1)
  - Distributed SQL Query engine execution on top of Tez, Spark, or plain MapReduce.

(1) Requires specialized Serializer Deserialized (SerDer)

HIVE Under the Hood



source: https://cwiki.apache.org/confluence/display/Hive/Design

Start your engines

https://dslab2022-renku.epfl.ch/projects/com490/lab-course (Fork)

GRADED ASSIGNMENT 1

CarboSense - Data Science with CO2

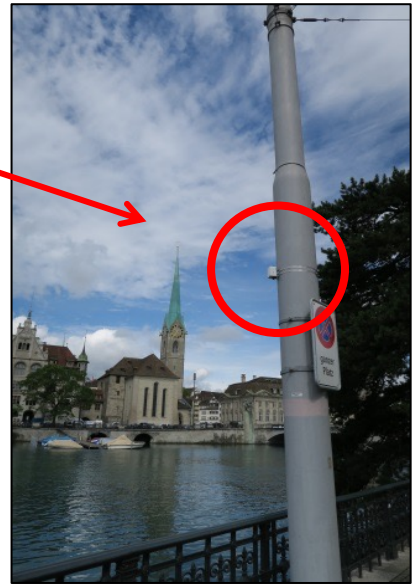
Time Series Modeling, Data Visualization

EPFL

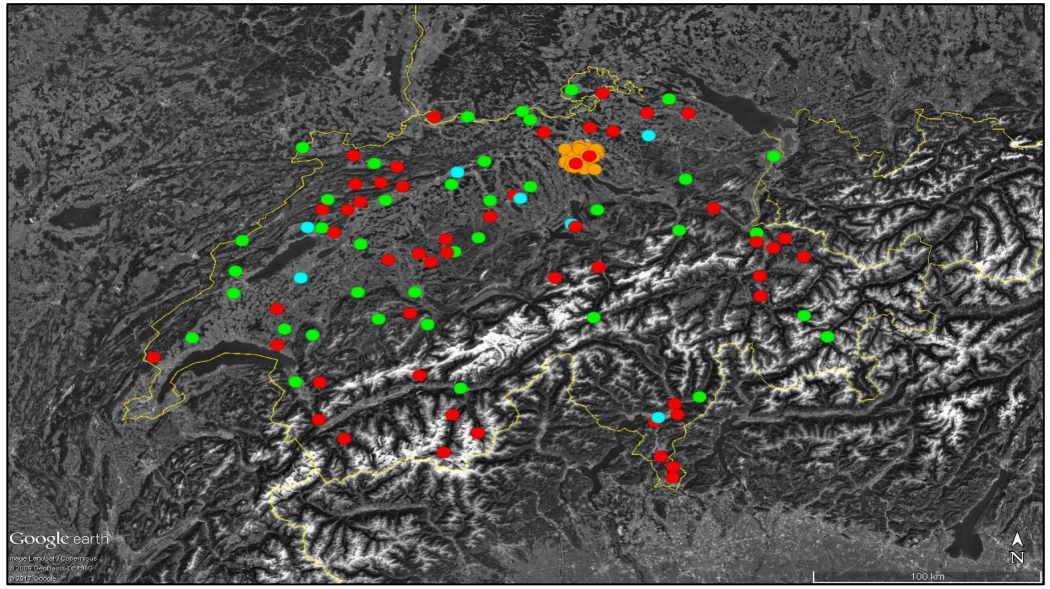
CarboSense: A low-cost low-power CO2 network

- Motivation
  - Improve the quantification of anthropogenic CO2 emissions and CO2 fluxes of the biosphere accounting for their high variability in space and time
  - Provide near-real time information on man-made emissions

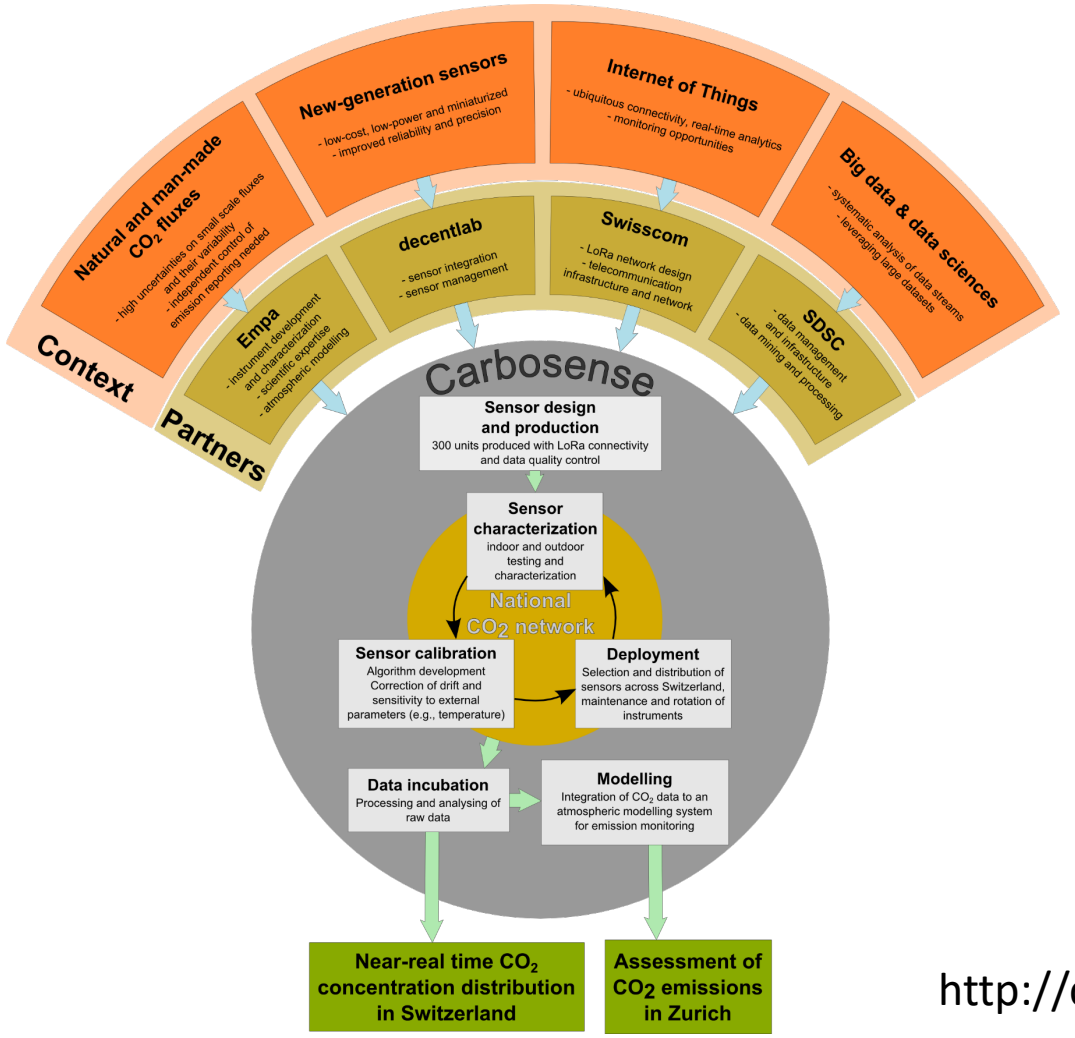
Different sensor units



Sensor in Zurich



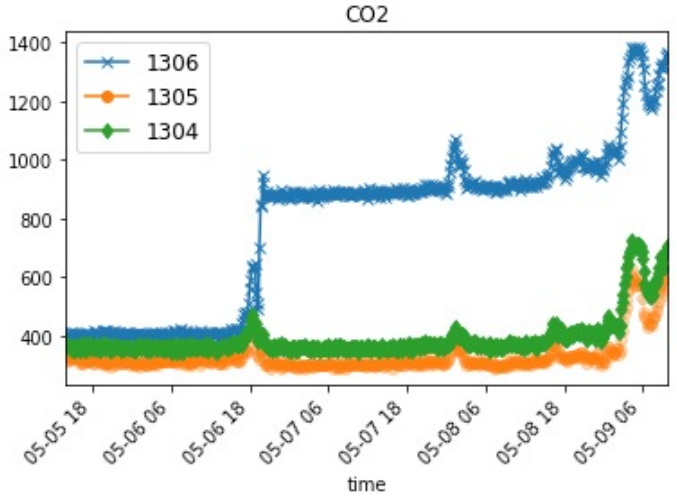
CarboSense: Project Overview

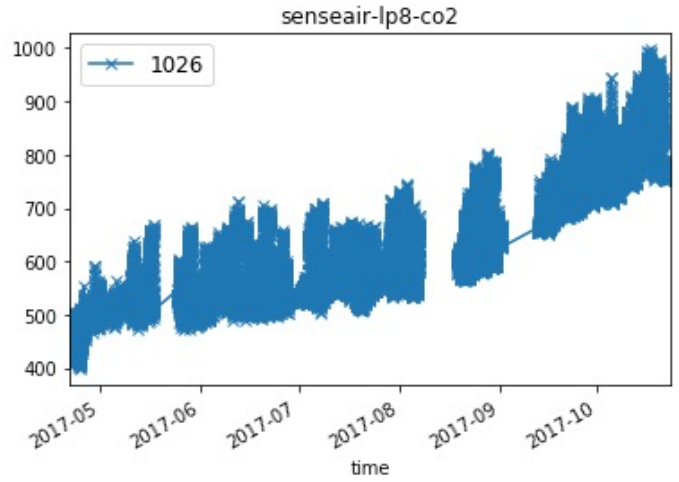


http://carbosense.wikidot.com

CarboSense: Sensor's measurements

- Inaccurate sensor measurements
  - All sensor devices have been calibrated (CO2, Temperature, Humidity) in climate and pressure chambers and under ambient conditions before deployment
  - Sensors behavior may change over time (sudden/single discontinuities, slower changes/drifts)
  - External parameters such as traffic area, altitude may affect this change





GRADED ASSIGNMENT 1

- https://dslab2022-renku.epfl.ch/projects/com490/homework1

- We will focus only in the area of Zurich
  - 46 sites located in different part of the city

- Measurements for each site
  - CO2 (ppm)
  - Temperature
  - Humidity

- Additional metadata
  - Altitude at which each sensor is located
  - Division of the city into zones with different anthropogenic emissions (e.g., industrial, residential, forest, mountain, etc.)

GRADED ASSIGNMENT 1

- Curate the CO2 measurements, by processing jointly the sensor measurements
  - Fit a robust regression model to the CO2 measurements, that takes into account all the different parameters
  - Use this model to detect possible inaccurate values

- Prior knowledge
  - There is a strong dependence of the CO2 measurements, on temperature, humidity, altitude, traffic, etc.

GRADED ASSIGNMENT 1

- Due **22.03.2022 23:59:59**
- Checklist
  - Create your group in gitlab (contact us if you do not have a group)
  - Fork homework-1 under your group name
    - https://dslab2022-renku.epfl.ch/projects/com490/homework1
  - Answer questions in homework's notebook
  - Work in team (take advantage of git, gitlab and renku)
  - Only commit the code and comments
    - Do not commit results of notebook execution (in history or in final version)
      - Do a cell output clear before committing
  - Always check your result
    - Restart kernel and run all cells (then clear the cells)
  - **Submit before deadline: git add + git commit + git push**