THE DATA SCIENCE LAB
General Introduction to Big Data

COM 490 – Module 2a

Week 3

EPFL

Agenda 2025 - Module 2a

Date	Topic
19.02	Introduction to Data Science with Python
26.02	(Bigger) Data Science with Python
05.03	Introduction to Big Data Technologies
12.03	Big Data Wrangling with Hadoop
19.03	Advanced Big Data Queries
26.03	Introduction to Spark
02.04	Spark Data Frames
09.04	Advanced Spark
16.04	Introduction to Stream Processing
30.04	Stream Processing with Kafka
07.05	Advanced Stream Processing
14.06	Final Project Q&A
22.05	Final Project Videos Due before midnight
28.05	Oral Sessions

Week 2 – Questions?

Objectives Module 2a

- **Most of you have formed the groups**
- **You have access to the exercises of module 1b**
- **You understand the purpose of git and master the *most commons* commands**
- **You should be able to determine an efficient data storage format for your needs (Parquet, HDF5, …)**
- **You are aware of other (than pandas) python data processing technologies readily available to you (polars, dask, vaex, ray, duckdb, …)**

Solutions exercises Module 1b

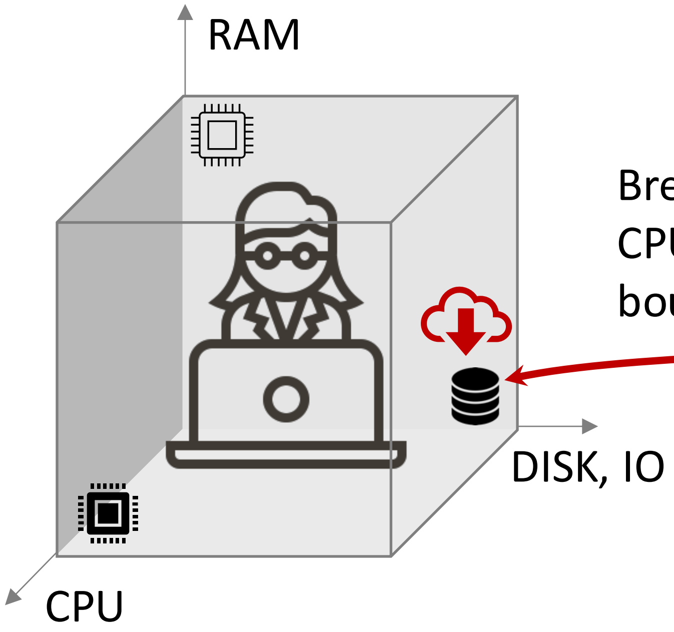
```
$ git branch -a
* main
  solutions
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
  remotes/origin/solutions
$ git checkout solutions
Switched to branch 'solutions'
Your branch is up to date with
'origin/solutions'.
```

Today's Agenda

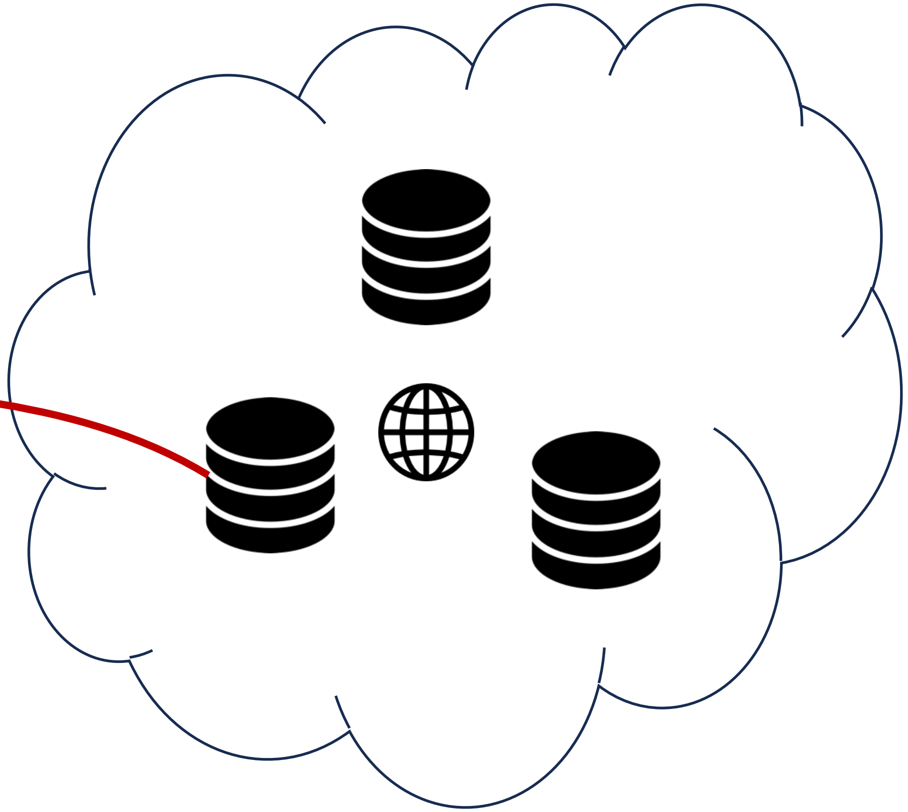
- Bootstrapping your digitalization journey
  - An overview and terminology of big data technology
    - Hadoop, HDFS, MapReduce, …
- Lab week 3
  - First steps with Hadoop Distributed File System (HDFS)
  - Start building your Data Lake

Bootstrapping Your Digitalization Journey

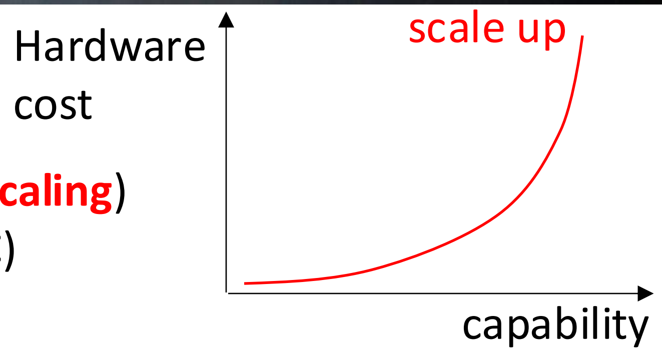
Intro - Addressing the Big Data Challenges



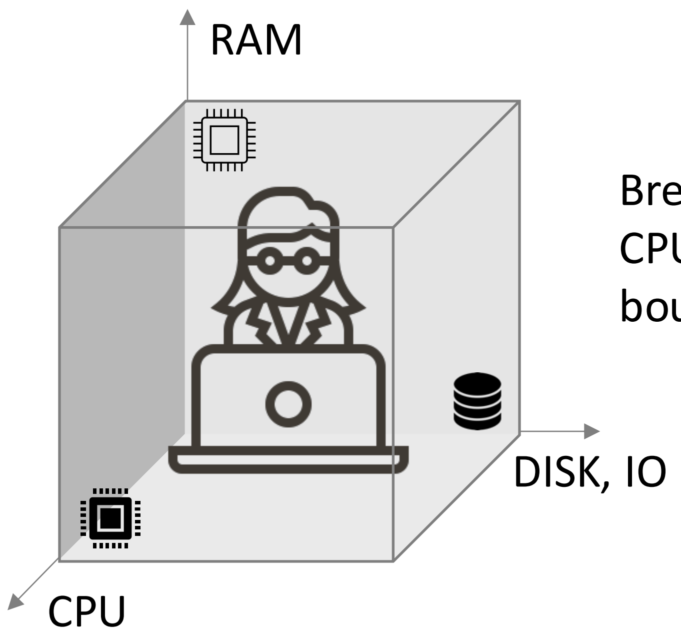
Break the CPU/RAM/Disk bounds



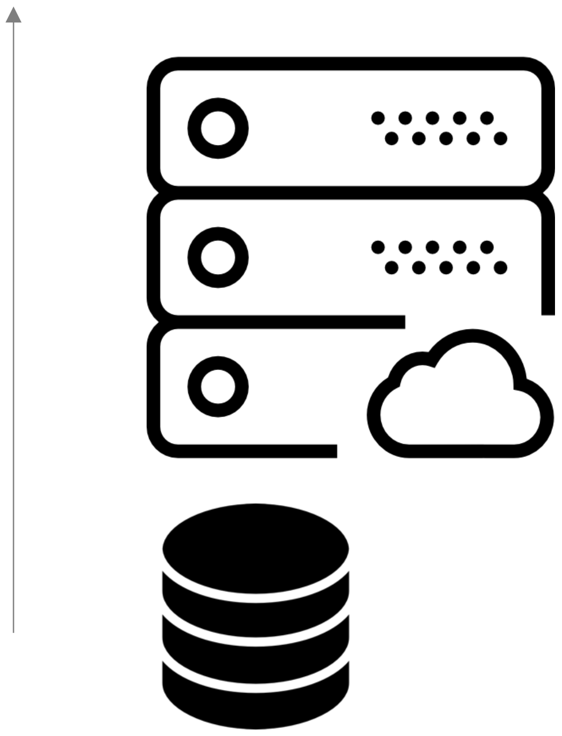
Intro - Addressing the Big Data Challenges



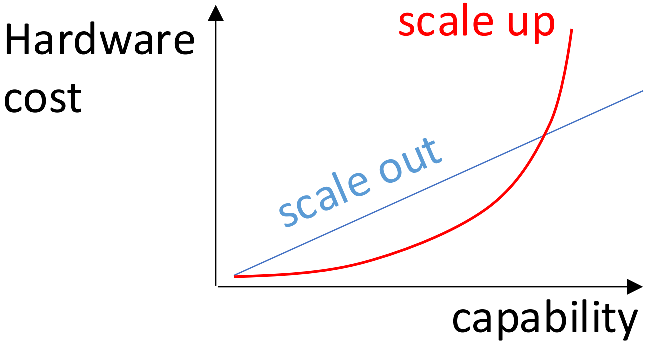
Use bigger machines: **scale up** (a.k.a **vertical scaling**)
… the High-Performance Computing way (HPC)

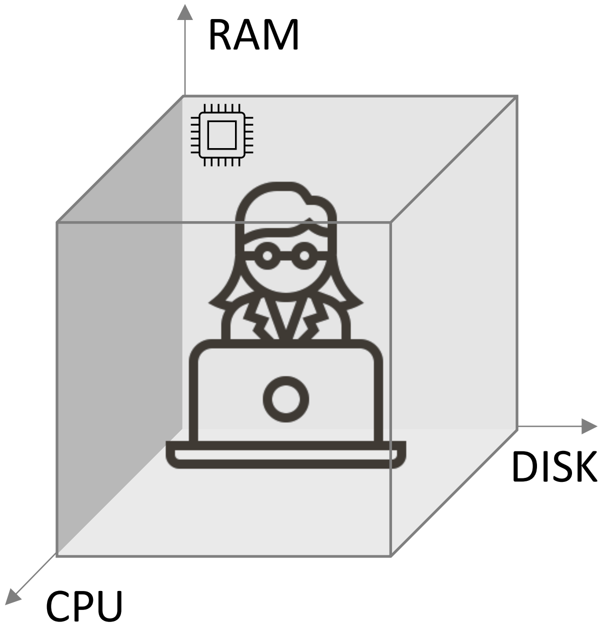


Break the CPU/RAM/Disk bounds

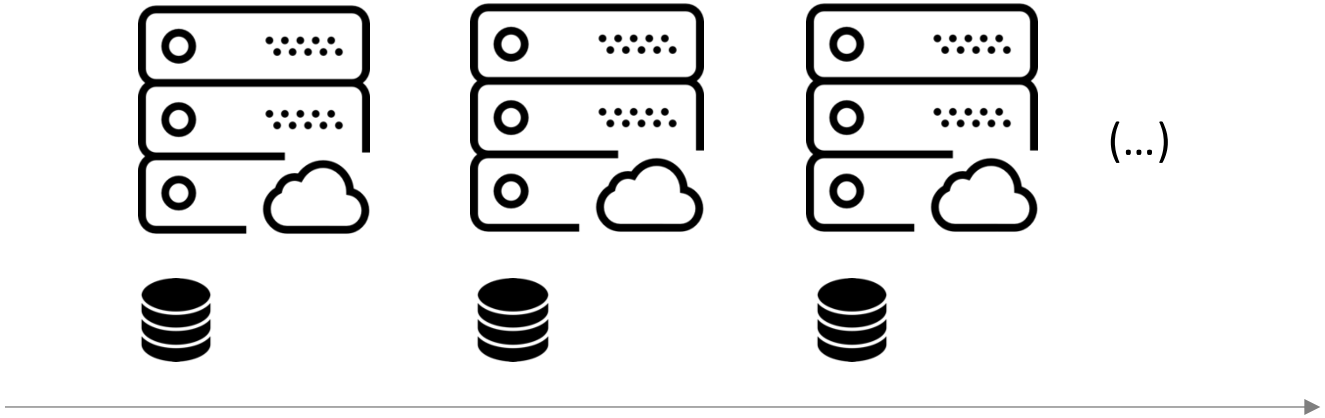


Intro - Addressing the Big Data Challenges





Break the CPU/RAM/Disk bounds



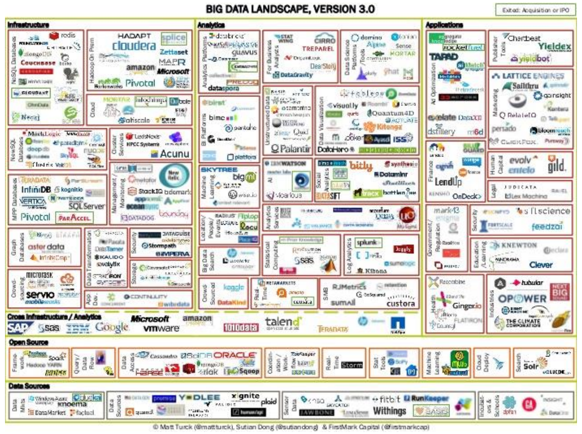
Use more machines: **scale out** (a.k.a **horizontal scaling**)
... the commodity hardware (cloud) way

Intro - Addressing the Big Data Challenge

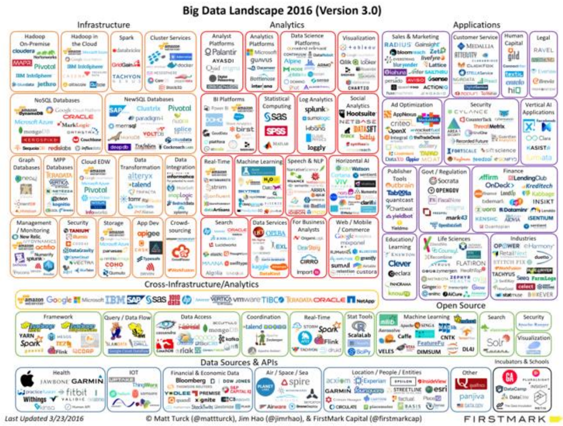
- Horizontal scaling entails (shared) distributed computing across a large number of compute servers
- Advantages of distributed computing are:
  - Parallel execution
  - Easier to run code closer the data - minimize data transfer
- Challenges of distributed computing are:
  - The same code should work seamlessly on 1, 10, or 10,000 servers
    - Assume the problem can be broken down into chunks, each chunk calculated locally
    - The data must be accessible from anywhere
  - Optimized resource utilization
    - Minimize hot-spots with an effective load-balancing strategy
    - Bring compute to data (data locality)

    => A resource manager is required to ensure fair and efficient use of resources
  - Fault tolerant and high availability
    - The system must handle one or more server failures with no impact on operations
  - Support for elastic scaling
    - Add/remove machines without requiring down-times or complex maintenance

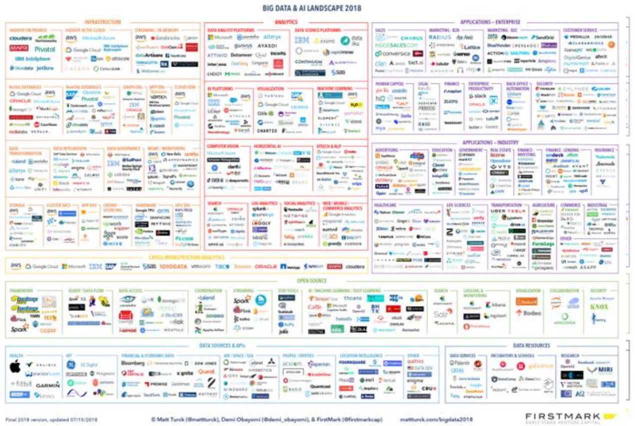
The ML, AI and Data (MAD) – A Moving Target



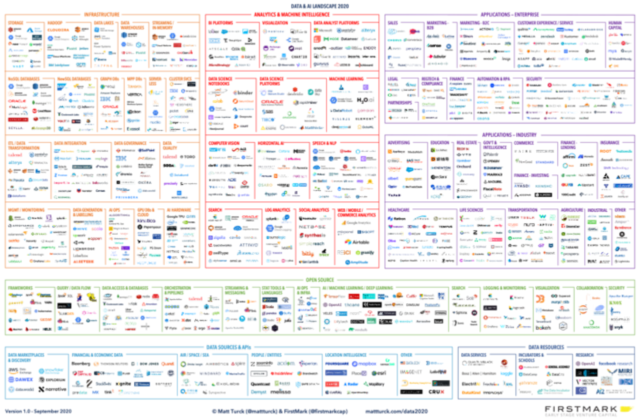
2014



2016



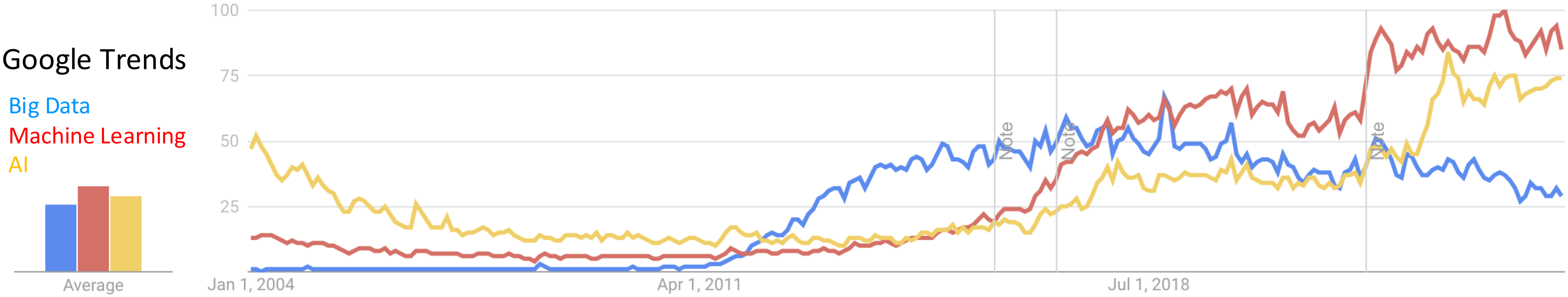
2018



2020

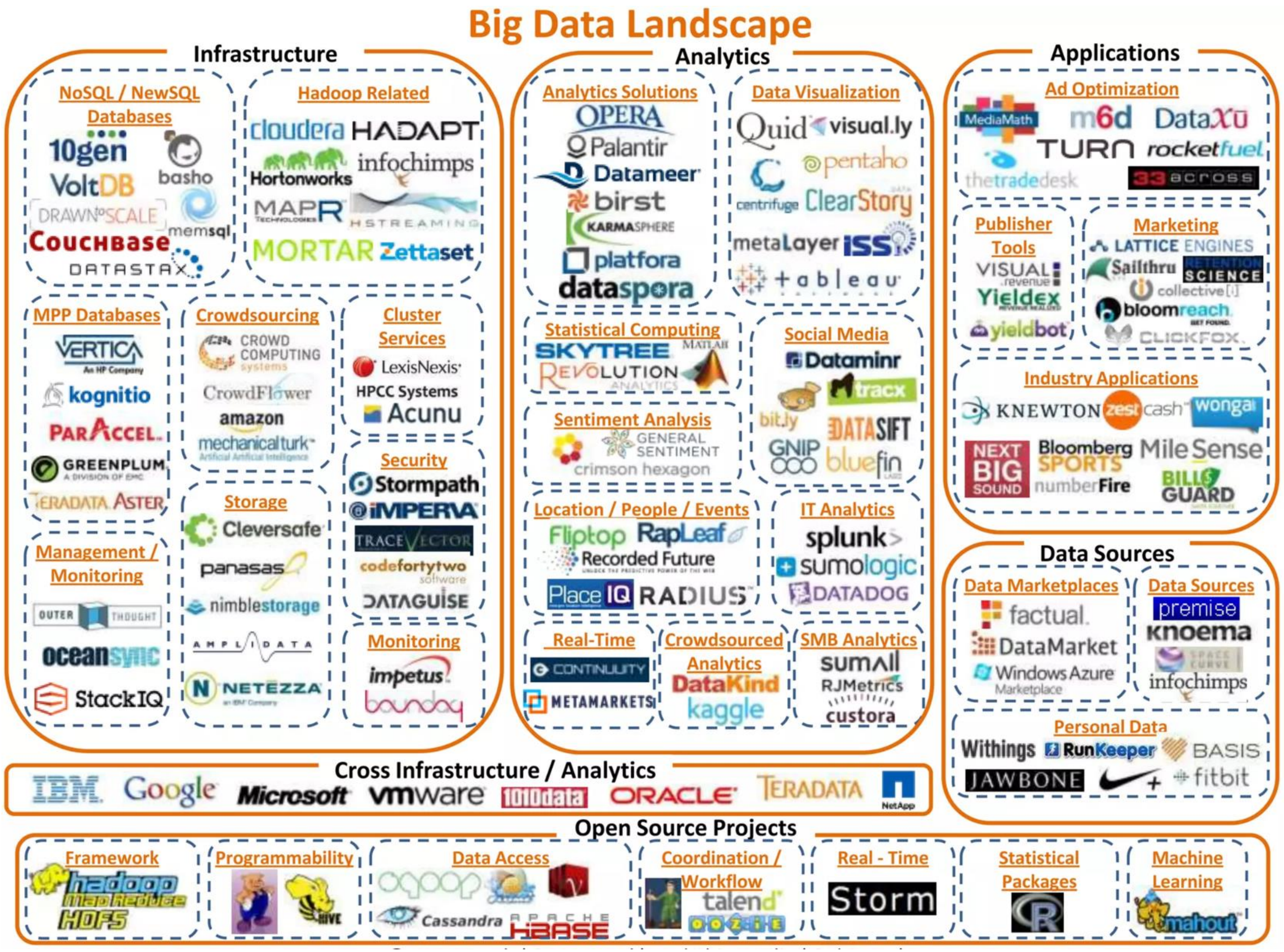
Google Trends

Big Data
Machine Learning
AI



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

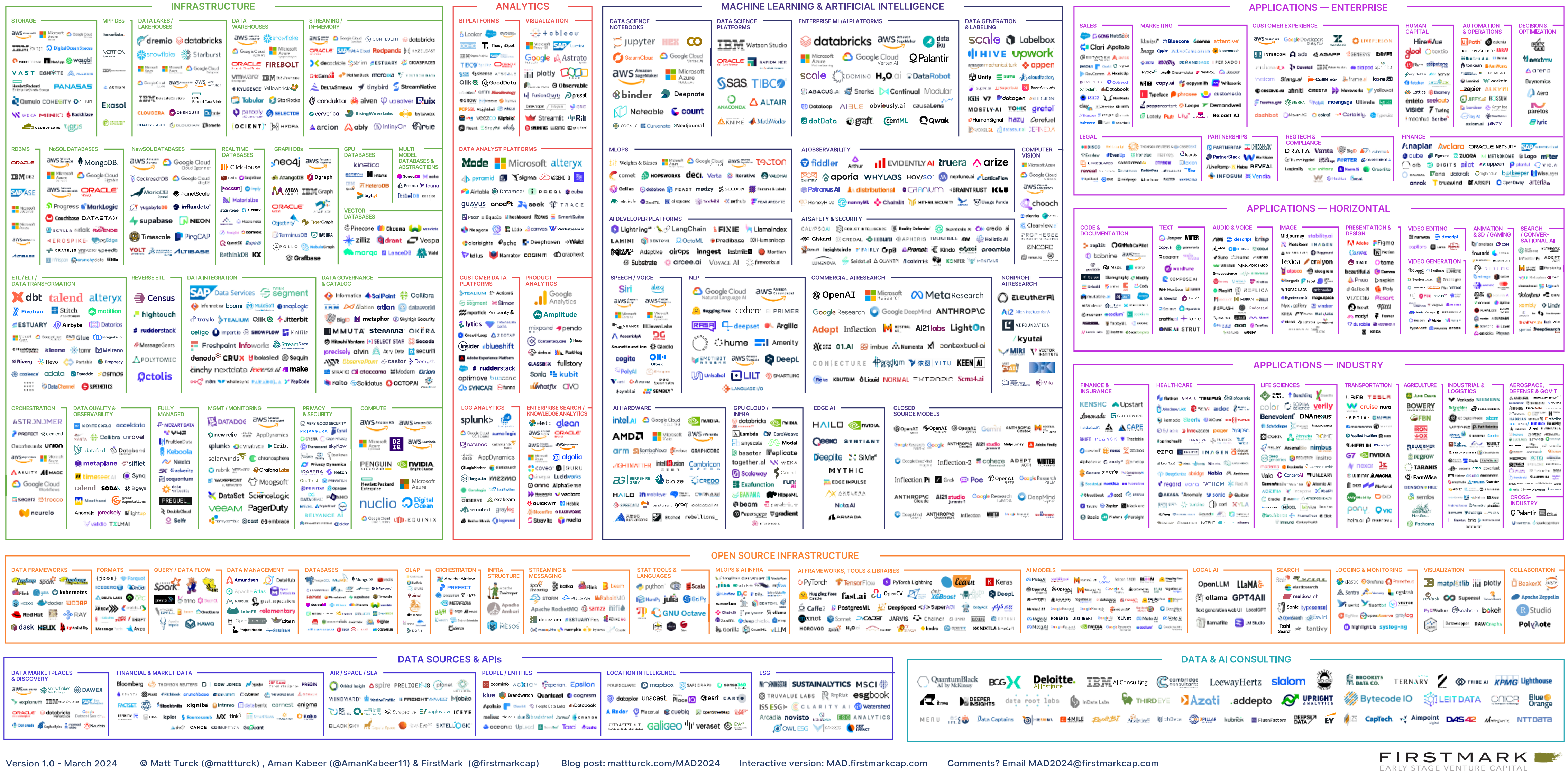
The ML, AI and Data (MAD) – 2012



EPFL

The ML, AI and Data (MAD) – 2024

THE 2024 MAD (MACHINE LEARNING, ARTIFICIAL INTELLIGENCE & DATA) LANDSCAPE



Source: https://mattturck.com/mad2024/

Today's First Objective

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

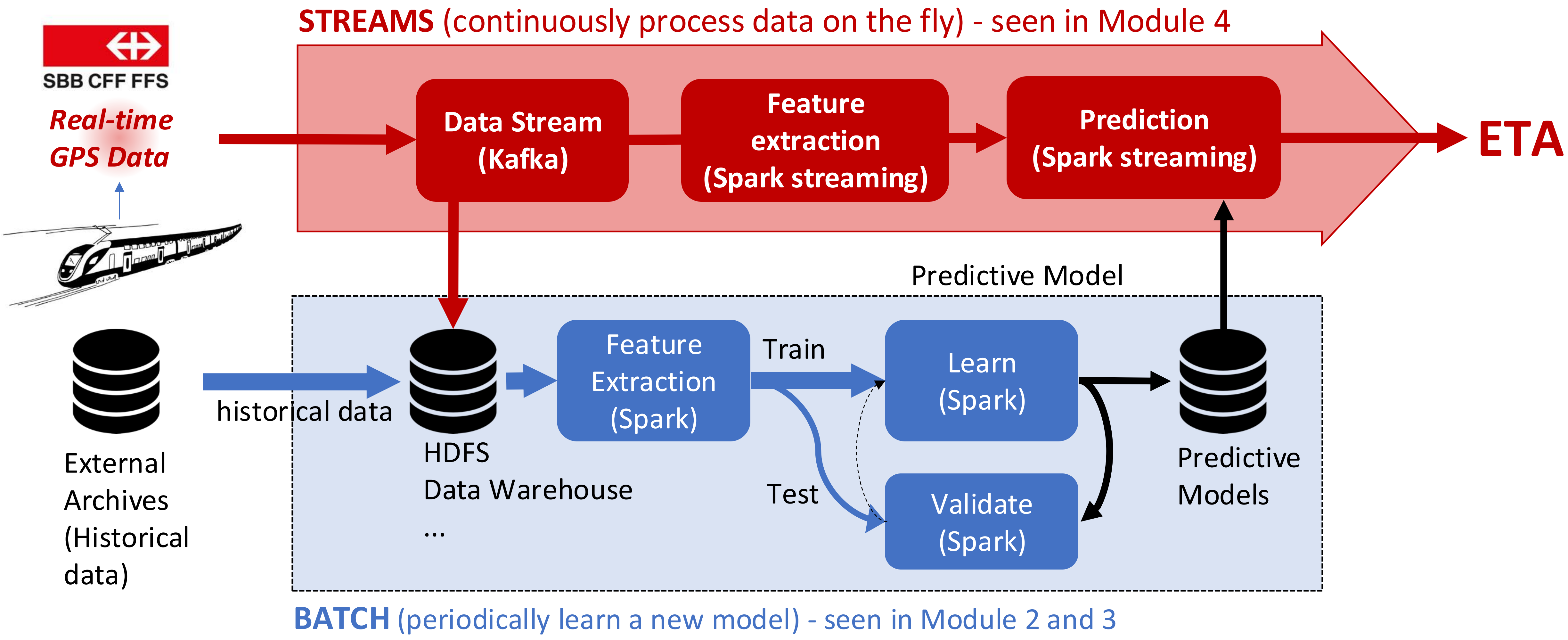
Big Data's 4Vs

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**
  - **Method**: Operates on finite size data sets (e.g. monthly update), and terminate after all data has been processed, repeat.
  - **Application**: create reports, training models, …
  - **Data warehouses (Hive, …), Hadoop Map Reduce**, **Spark Batch**

- Your application needs results as soon as more information becomes available? **STREAMS**
  - AKA: **Data in motion**, or **Fast data**
  - **Method**: 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
  - **Application**: Often used in critical systems, where fast response time to event is essential
  - **Spark Streaming**, **Kafka**, **Flink**, **Storm**

STREAMS and BATCH Illustrated



In-Memory Versus Out-of-Core Processing

In-Memory

- Entire dataset (or at least the part being processed) fits into memory (RAM) during computation
- Faster, once data is in memory
- Limited by available RAM
- E.g. **Pandas**

Out-of-Core

- Data is loaded in **chunks** into memory during processing
- More **Disk/Network I/O**: needed to retrieve chunks of data
- Used when data set exceeds RAM
- E.g. Vaex, Polars, DuckDB, pyarrow

Understanding Push-down In Data Processing

- Delegate operations to the underlying data source (database, storage, etc.)
  - Operation is performed **closer to the data**
  - Reduce the volume of data transferred (over network, from disk)
  - Leverage the data source's native optimizations

- Example:
  - Parquet and ORC formats store column statistics (e.g., min/max values)
  - With DuckDB and PyArrow's predicate pushdown, queries like the following read only relevant data:

```
SELECT Date,Temp FROM weather WHERE Date > '2025-01-01'
```

Understanding Hive Partitioning In Data Storage

- **Organizes data** into directories based on column values, e.g., year, month, day

```
…/weather/year=2025/month=01/day=01/*.parquet
                            /day=02/*.parquet
                            …
                   /month=02/day=01/*.parquet
```

- **Reduces I/O** by skipping irrelevant partitions during queries

- Can be used in **predicate pushdown** to further optimize queries

E.g. … `WHERE year=2025 AND month>6`

(1) Hive partitioning is not the only scheme for partitioning data in the file system, but it is one of the most widely supported schemes.

Understanding Tradeoffs of Distributed Data Stores – C.A.P

C.A.P Theorem "It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees" (Brewer's Conjecture, 2000)

- **C**onsistency: All clients see the same data (last update) [1]
- **A**vailability: Every request get a response, even if partition [2] happens (node, or network failure)
- **P**artition tolerance: C ~~and~~/**or** A holds despite messages being dropped or delayed between partitions

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

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

- C+P: System refuses to answer, and thus forfeit **A**vailability but guarantee **C**onsistency
- A+P: Proceed with the operation and thus provide **A**vailability but risks Inconsistency

Choose your technology based on whether consistency or availability is more important for your application!

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

[1] This is server partitioning, which is not the same as file or data partitioning such as seen in Hive partitioning

Addressing the Big Data Challenge – Big Data Stack

Machine Learning
- Spark MLlib
- H2O.ai
- MAHOUT
- TensorFlow

Batch Processing
- Spark
- apache Ignite
- Map Reduce

SQL Data Warehouse
- Apache Impala
- HIVE
- trino
- presto

NoSQL Database
- APACHE HBASE
- cassandra
- accumulo

Stream processing
- STORM
- Apache Flink
- kafka

Resource Management
- hadoop YARN
- kubernetes
- MESOS

Storage
- HDFS
- ALLUXIO
- S3

Real time data flow orchestration
- nifi

Batch tasks scheduling (ETL)
- OOZIE
- Apache Airflow

Security
- Apache Ranger

Addressing the Big Data Challenge - Other Technologies

DASK

https://dask.org/

Familiar for python users

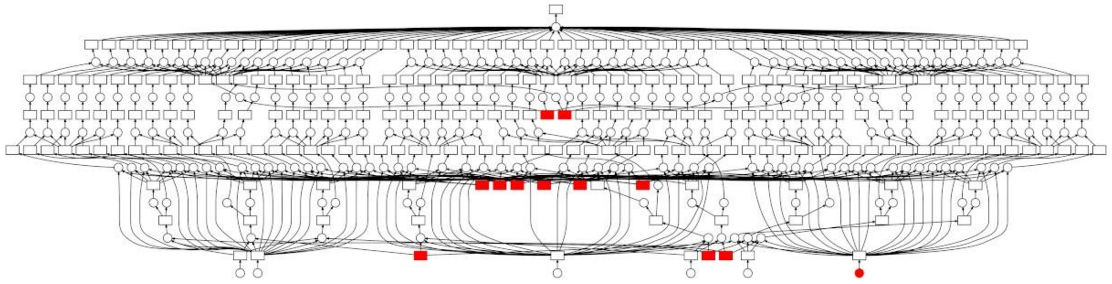
```python
# Arrays implement the NumPy API
import dask.array as da
x = da.random.random(size=(10000, 10000),
                     chunks=(1000, 1000))
x + x.T - x.mean(axis=0)
```

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

```python
# 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

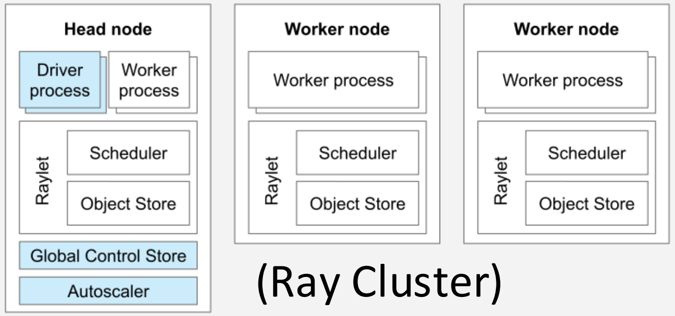
Scale up to clusters (including Apache YARN-managed)



RAY

https://ray.io/

```python
from ray.autoscaler.sdk import request_resources
# Request 1000 CPUs.
request_resources(num_cpus=1000)
# Request 64 CPUs and also fit a 1-GPU/4-CPU task.
request_resources(
    num_cpus=64, bundles=[{"GPU": 1, "CPU": 4}])
# Same as requesting num_cpus=3.
request_resources(
    bundles=[{"CPU": 1}, {"CPU": 1}, {"CPU": 1}])
```



(Ray Cluster)

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!

Hadoop Distributed File Systems HDFS

Technology Overview

Hadoop Distributed File Systems Top Features

- **Large Data Sets**
  - Size of a file only limited to total 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 using parity blocks instead of redundancy
same level of fault tolerance, but less replicas - best for rarely accessed data)

Hadoop Distributed File Systems Essentials

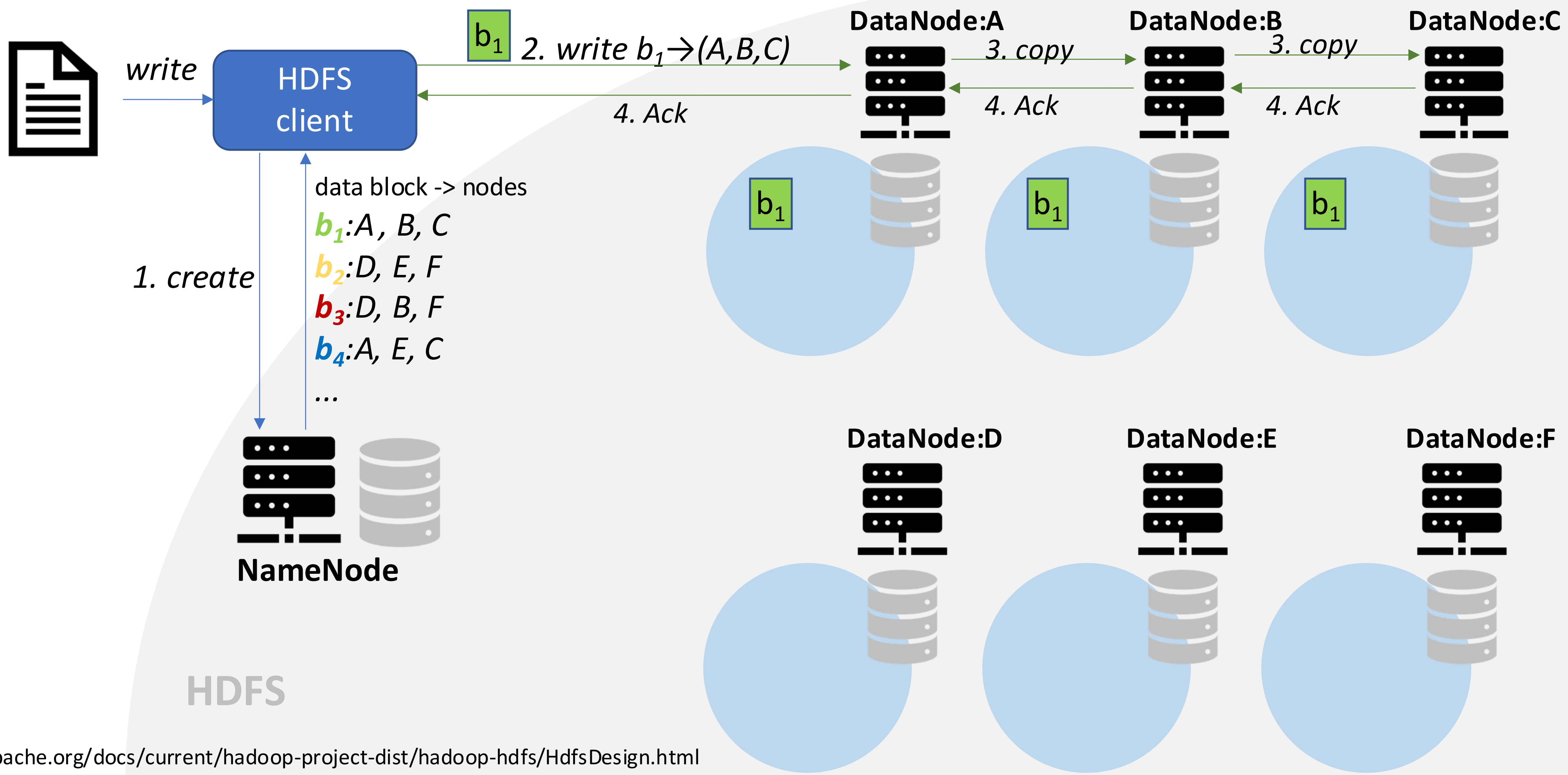
What about other storages like block storage (e.g AWS S3) ?

It works too, main differences:

- Block storage is more cost-efficient
- Block storage scales better and is more elastic than HDFS
- HDFS has better latency and performances than S3

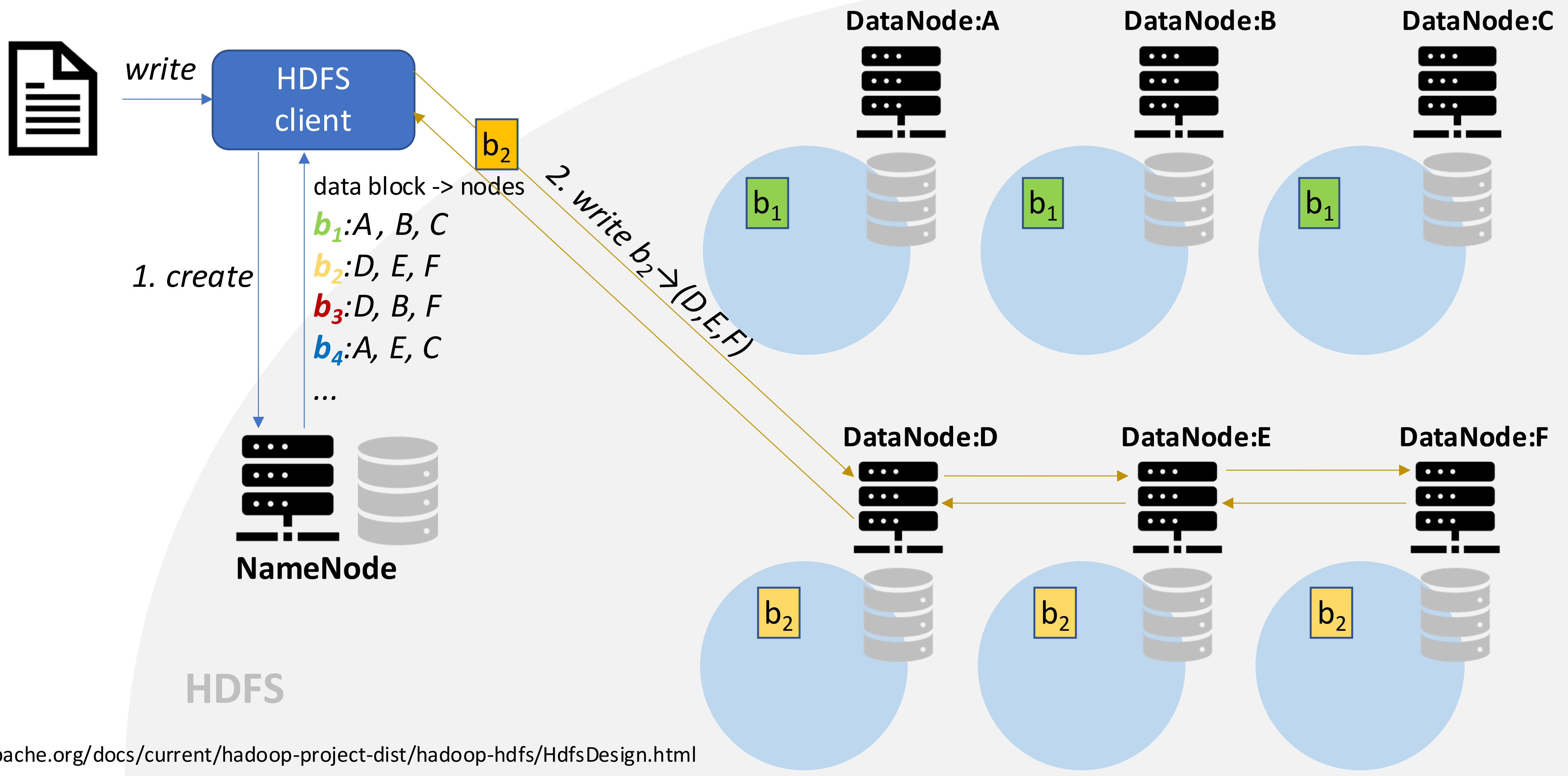
Others: security, durability, persistence, ... depends on the providers, on-premise vs in cloud etc.

Hadoop Distributed File Systems (HDFS) Essentials



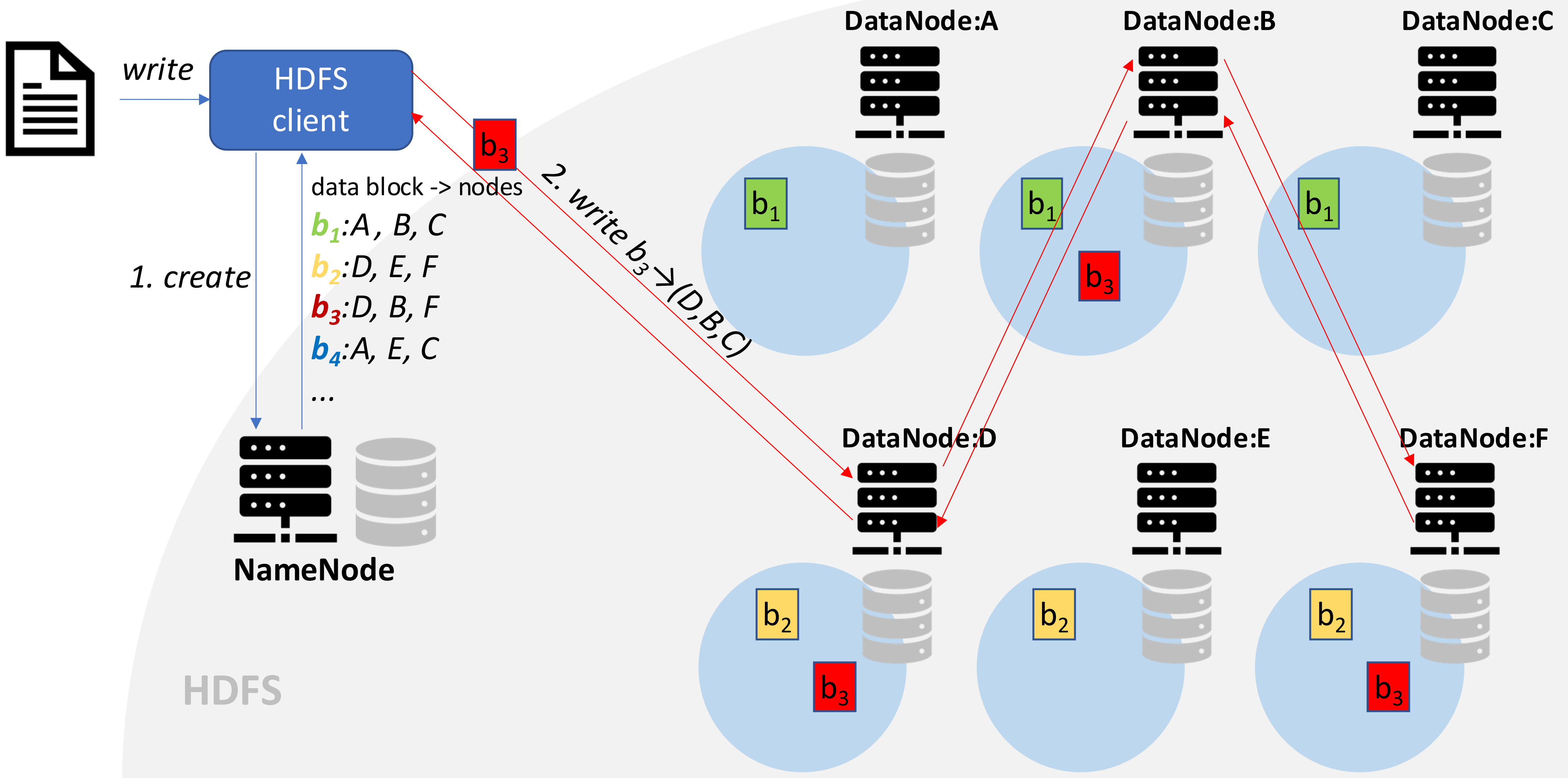
https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

Hadoop Distributed File Systems (HDFS) Essentials

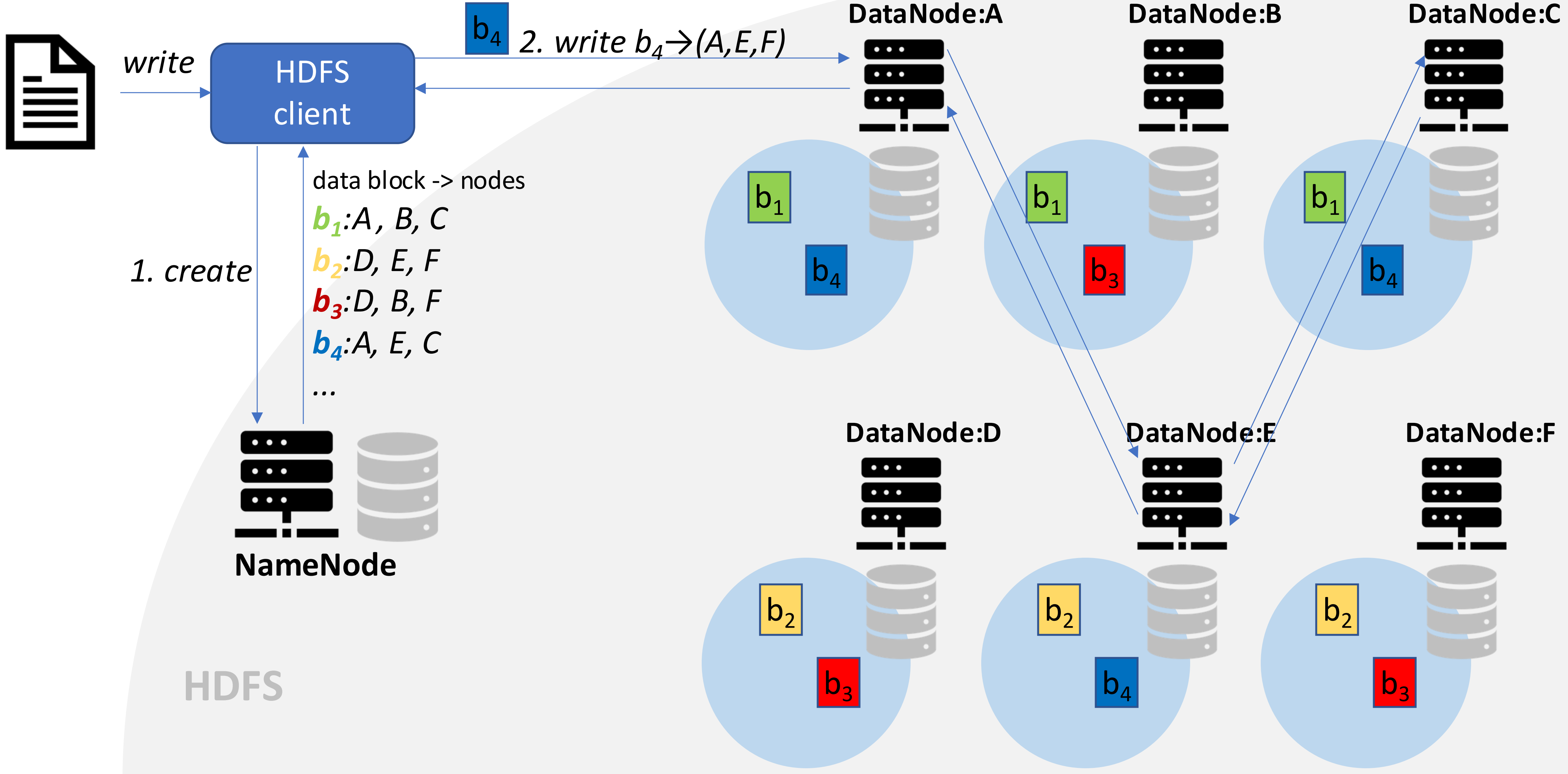


https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

Hadoop Distributed File Systems (HDFS) Essentials



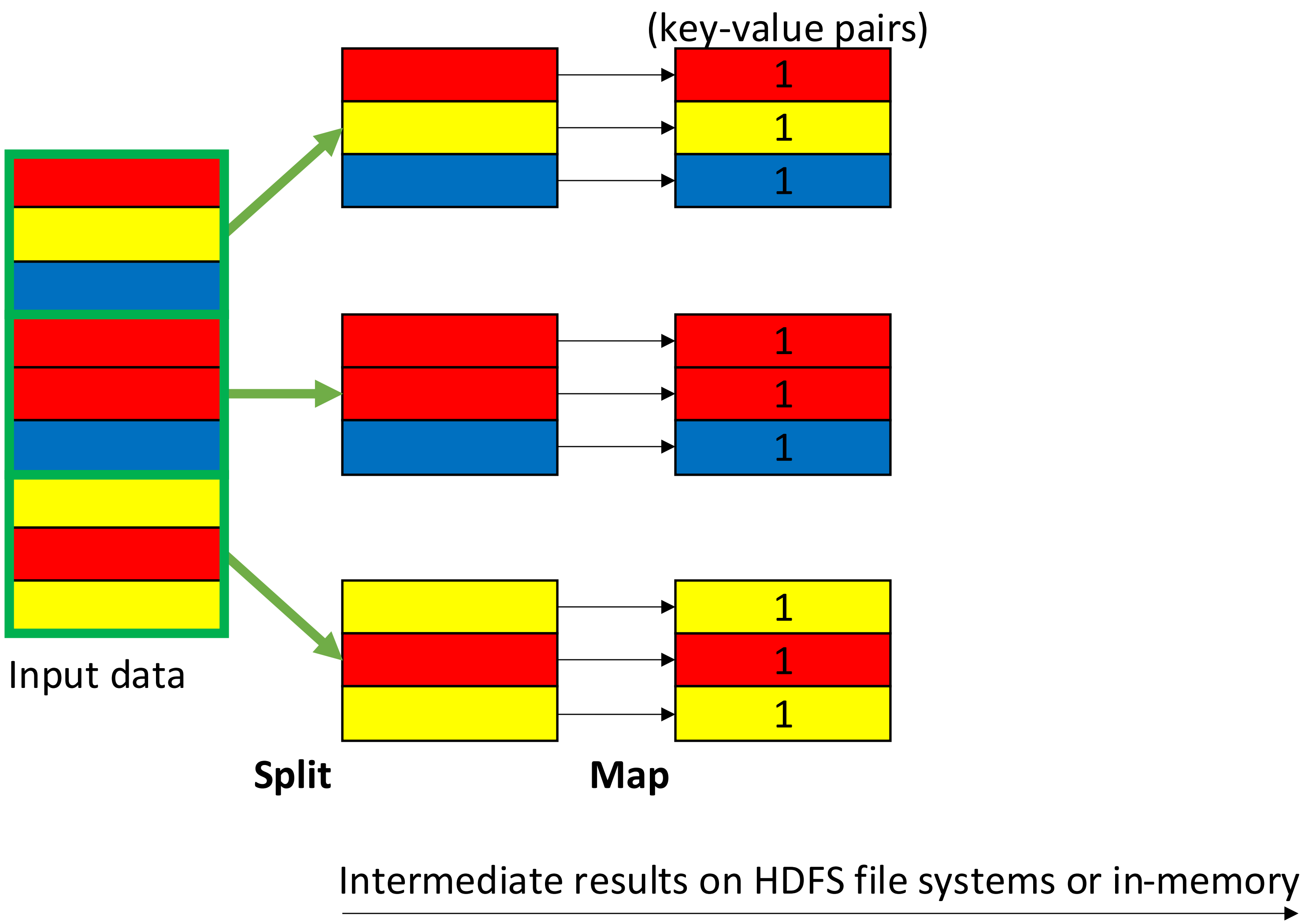
Hadoop Distributed File Systems (HDFS) Essentials



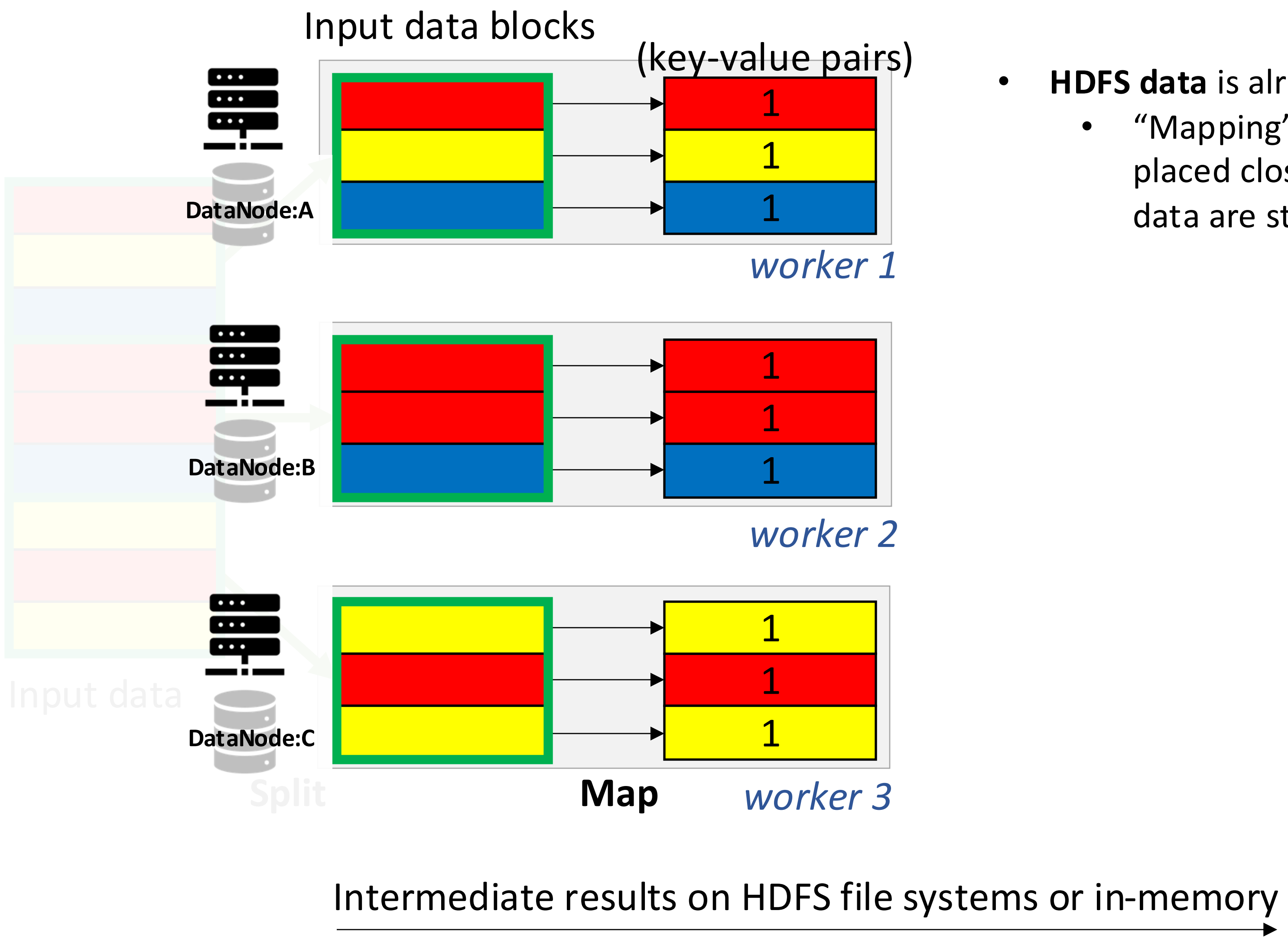
Hadoop Map Reduce

Algorithm Overview

Map Reduce in a Nutshell

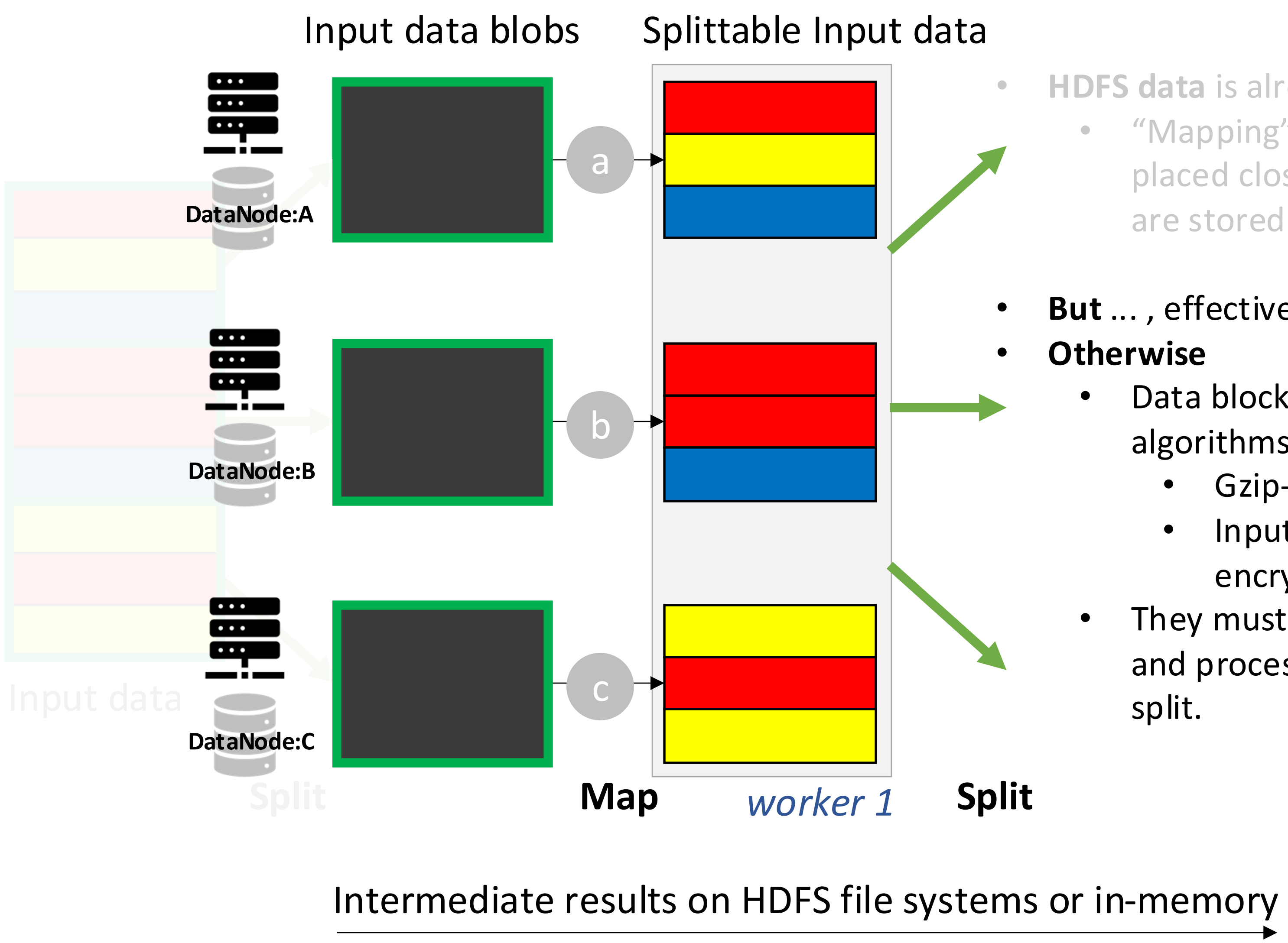


Map Reduce in a Nutshell



- **HDFS data** is already split into HDFS blocks !
  - "Mapping" can be done in <u>parallel</u> on worker nodes placed closest to datanodes where blocks of input data are stored

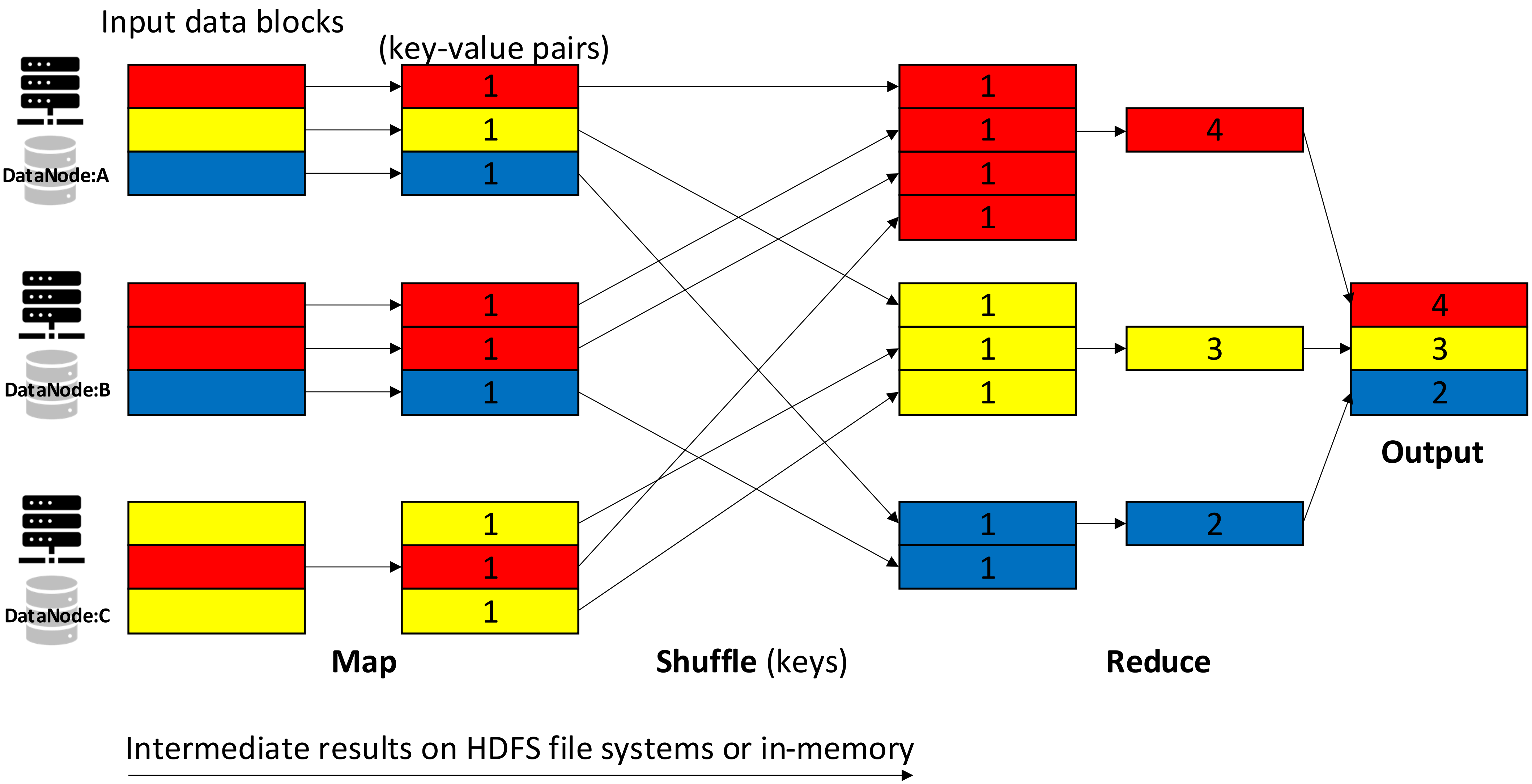
Map Reduce in a Nutshell – Hadoop Splittable



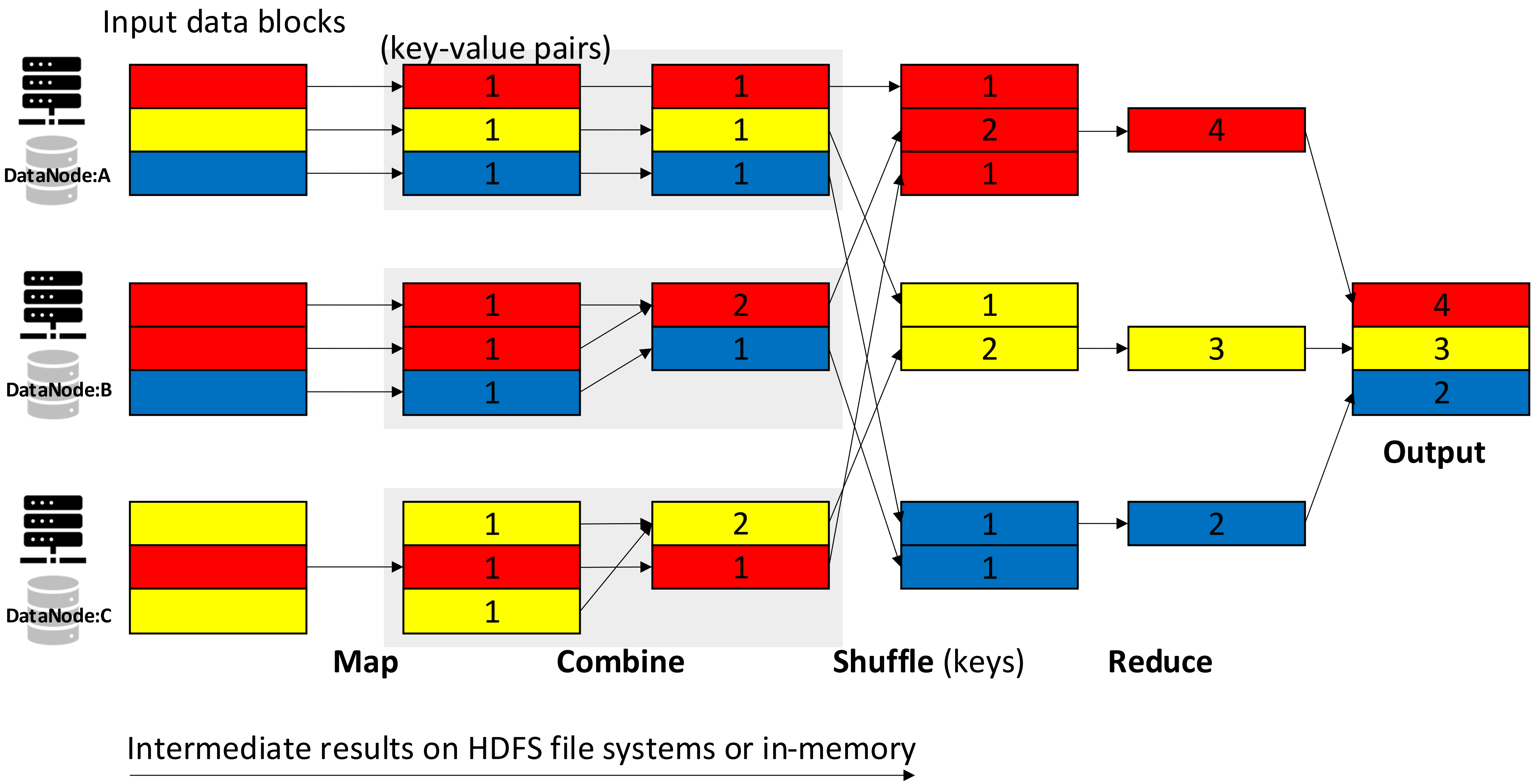
- **HDFS data** is already split into blocks !
  - "Mapping" can be done in parallel on worker nodes placed closest to datanodes where blocks of input data are stored

- **But** ... , effective only if input data is **Hadoop splittable** !
- **Otherwise**
  - Data blocks of Input data encoded using non-splittable algorithms are meaningless binary blobs, e.g.:
    - Gzip-compressed input data
    - Input data encrypted without using HDFS native encryption
  - They must be copied and reassembled in a worker node, and processed sequentially (e.g. decompressed), then split.

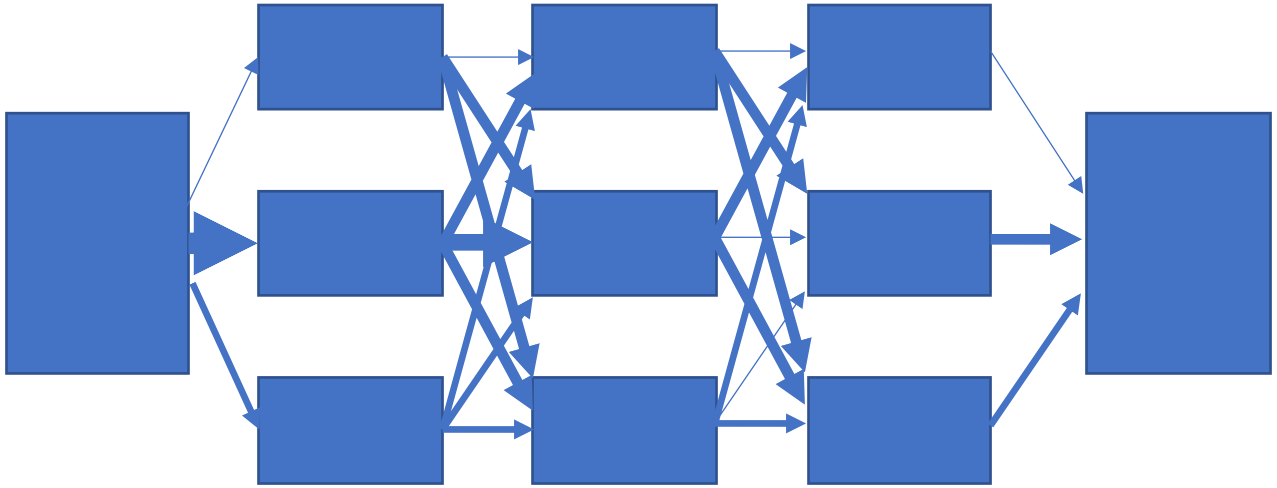
Map Reduce in a Nutshell

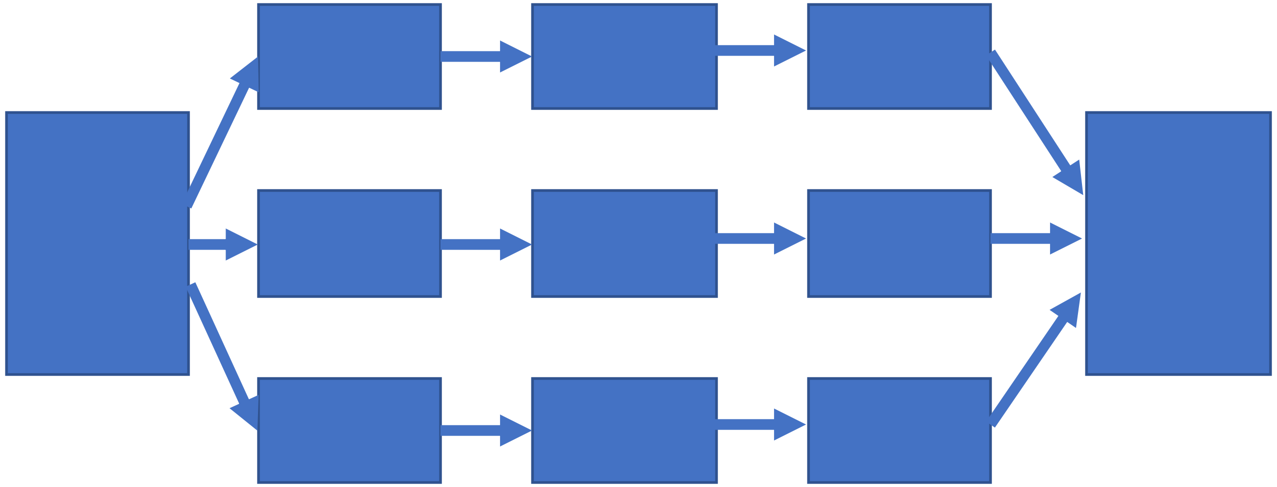


Map Reduce in a Nutshell

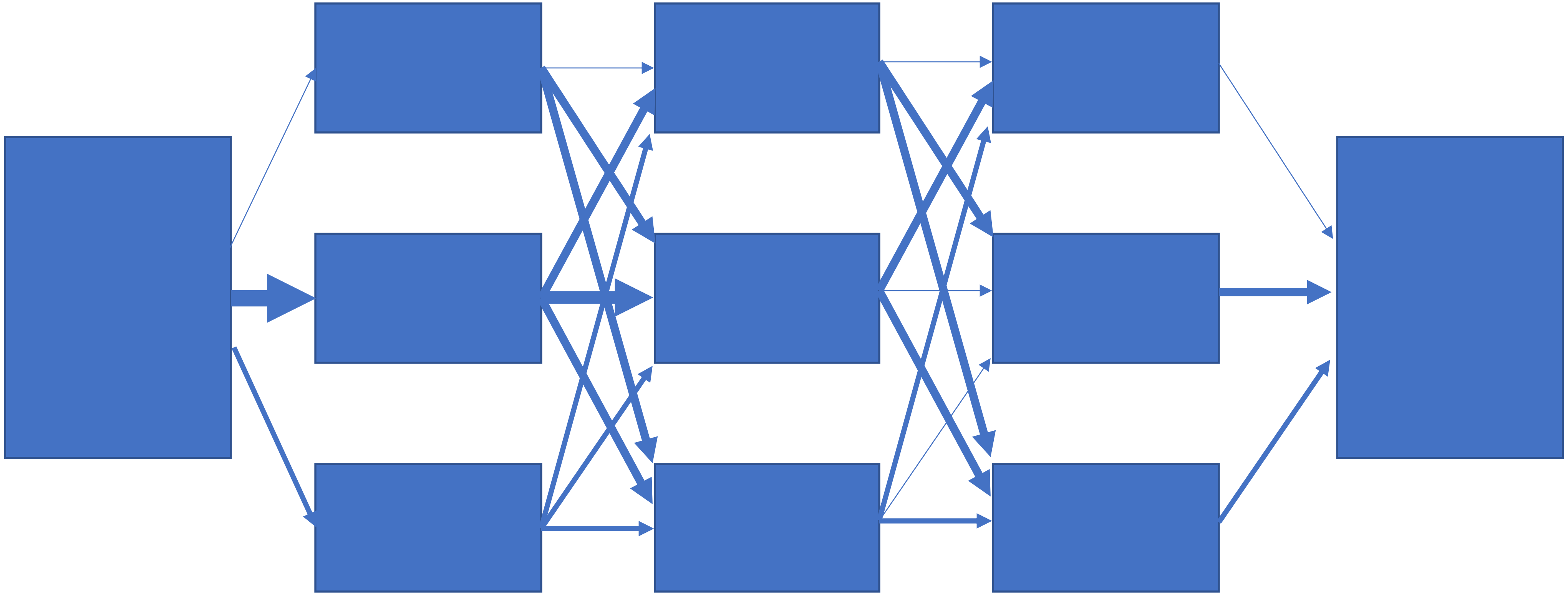


Quiz – Which one is best ?



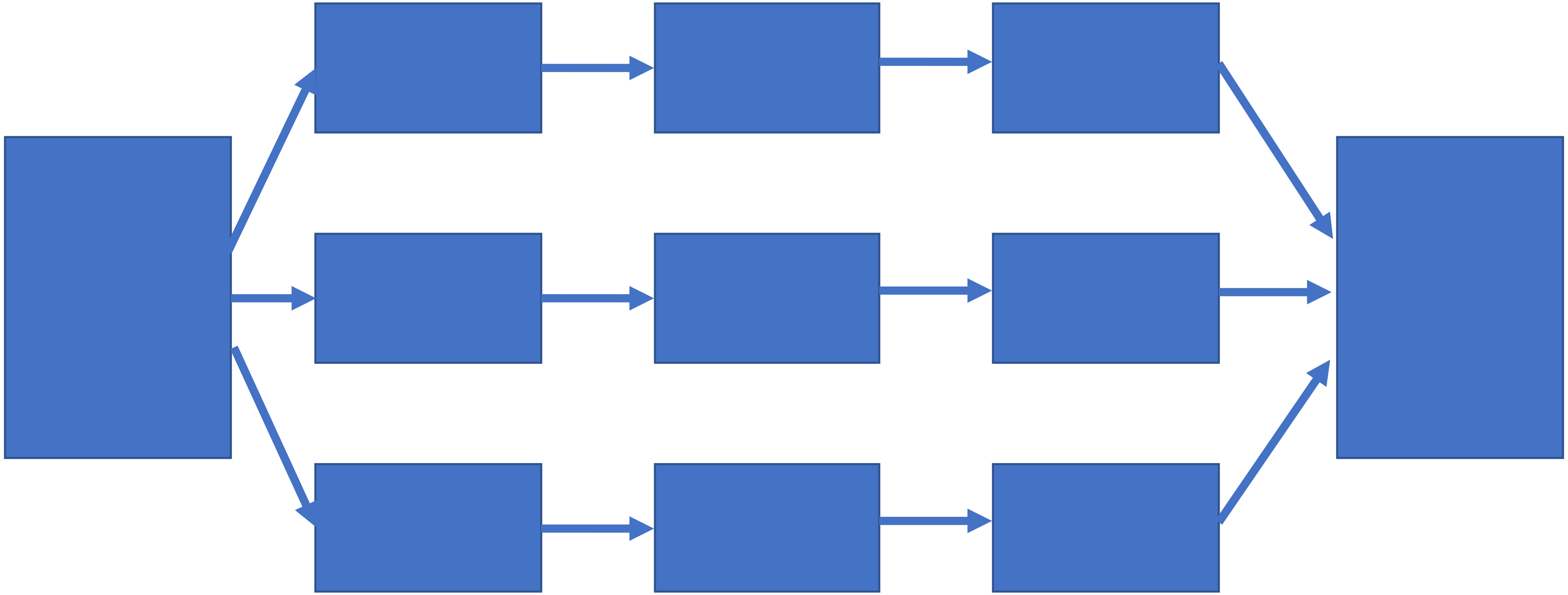


MapReduce gotchas



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

MapReduce Best Practices



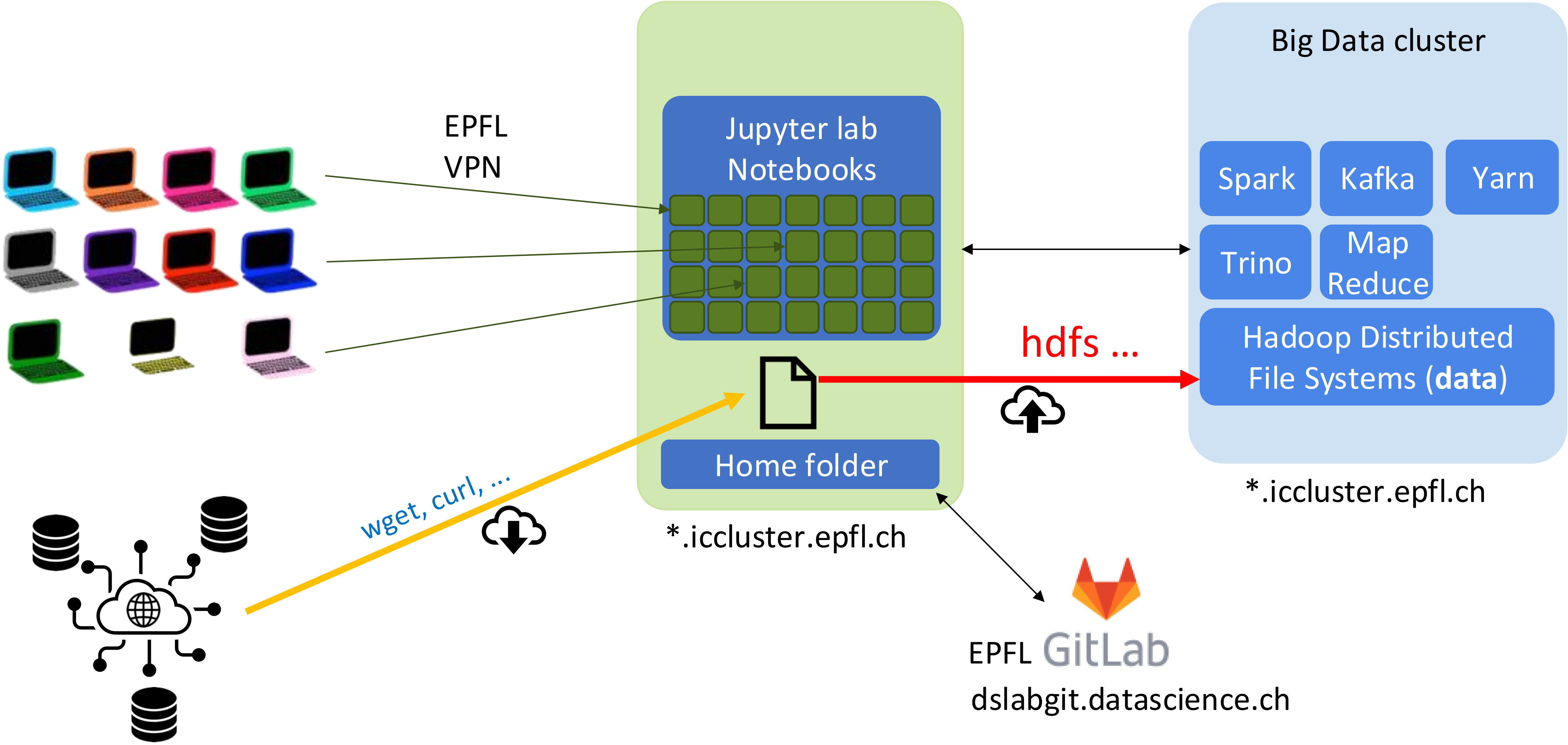
Optimization starts with good data partitioning practices to (1) better balance the load on CPU and RAM, and (2) minimize data shuffling and expensive network IO.

Today’s check list – key objectives

- **You have formed the groups**
  - Otherwise contact us
- **You have access to the exercises of module 2a**
- **You understand some of the fundamental concepts presented in class**
  - Scale-out vs Scale-up, challenges of scaling out, Hive Partitioning, Predicate Push down, HDFS, Splittable data format, Map Reduce, consistency availability tradeoffs, out-of-core computing (and what to do when pandas runs out-of-memory)
  - You have a clearer understanding of the various Hadoop technologies and their use cases, and you can recognize when other technologies offer similar features.
- **You can navigate HDFS and manage data on HDFS**

Start your engines

Uploading and managing data on HDFS



Uploading and managing data on HDFS - CLI

Purpose of today's exercises: upload data to HDFS

```
hdfs dfs -ls hdfs-path

hdfs dfs -mkdir hdfs-path

hdfs dfs {-copyFromLocal|-put|-moveFromLocal} local-file(s) hdfs-dest

hdfs dfs -mv hdfs-from-path hdfs-dest-path

hdfs dfs -chmod permissions hdfs-from-path hdfs-dest-path

hdfs dfs -setfacl acl-spec hdfs-from-path hdfs-dest-path

hdfs dfs -getfacl hdfs-dest-path

hdfs dfs -rm hdfs-dest-path

hdfs dfs -du hdfs-dest-path
```

Processing data on HDFS programmatically

PANDAS

- Python application programming interface (API) convenient library, can be used read or write data on different file systems, including HDFS (based on pyarrow)

Pyarrow

- Arrow: Low level API to abstract operations on different file systems, used to integrate data processing technologies and storage or data transfer systems
- Pyarrow: is the Python API wrapper of Arrow (others for C++, Rust, etc)

DuckDB

- Query data on HDFS (using pyarrow)

Reminder - Popular Storage Formats

- **Plain text (**csv, json, xml, …),
  - Row-oriented (most common)
  - Often sourced externally
  - Best for OLTP
  - Compression: None, Gzip, Bzip2, ...
  - Batch and stream processing
  - Splittable (if one line per record, depend on compression)
- **Parquet**
  - Column-oriented, ideal for OLAP workload
  - Integrated compression: SNAPPY, ZLIB, ZSTD, …
  - Splittable
  - Best suited for write once, read many (WORM)
  - Batch processing only
- **ORC**
  - Column-oriented, optimized for OLAP
  - Data stored in stripes (typically 250MB)
  - Indexed, splittable
  - Integrated compression: SNAPPY, ZLIB, ZSTD, …
  - Optimized for WORM
  - Batch processing only
- **Avro**
  - Row-oriented,
  - Splittable
  - Block level compression
  - Best for OLTP
  - Support schema evolution
- HDF5 / NetCDF4
  - Hierarchical, Multidimensional (D > 2)
  - Optimized for large datasets
  - Compression: ZLIB, SZIP, …
  - Splittable (with chunks)
  - Best for scientific and high-performance computing