Eötvös Loránd University (ELTE)
Faculty of Informatics (IK)
Pázmány Péter sétány 1/c
1117 Budapest, Hungary

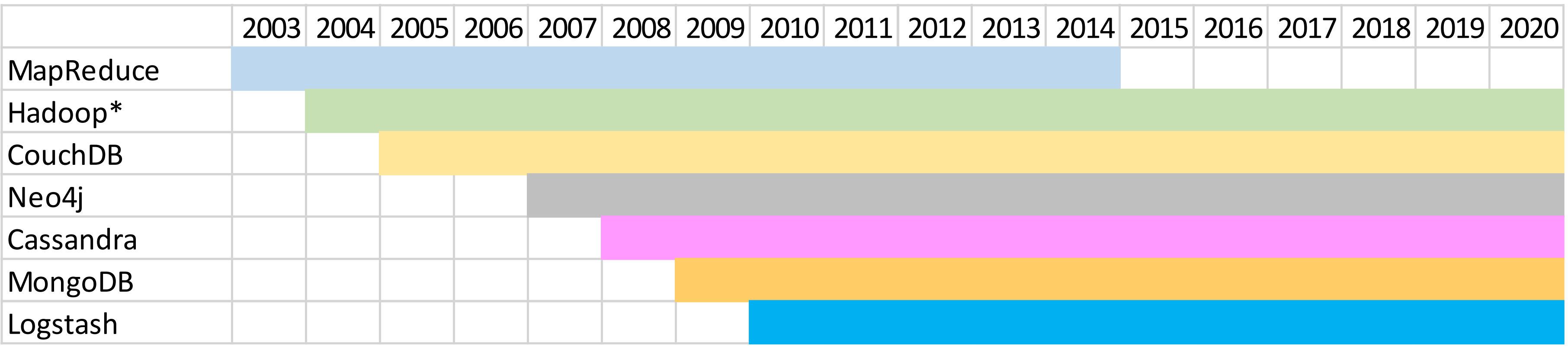
DATA STORAGE #2: THE HADOOP STORAGE ECOSYSTEM

Open Source Technologies for Real-Time Data Analytics

Imre Lendák, PhD, Associate Professor

2020
Budapest, Hungary

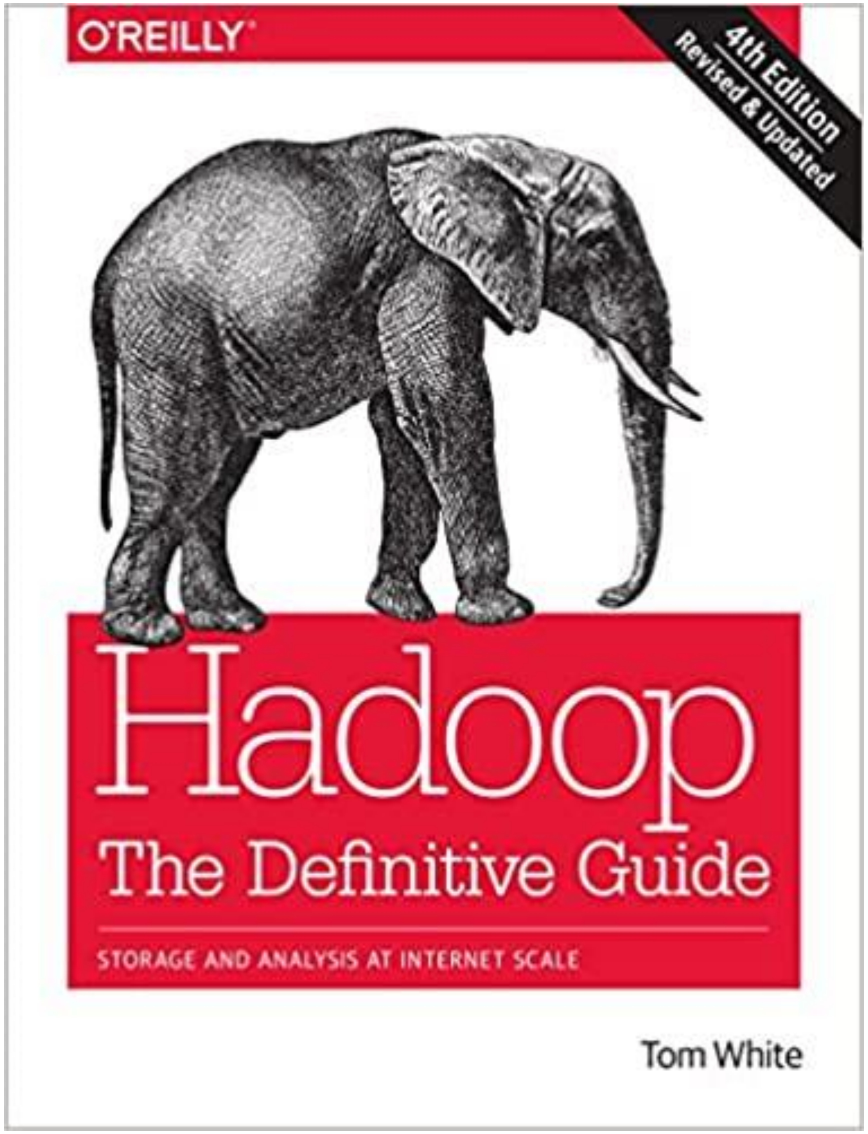
‘Big data’ storage timeline



* Hadoop lives as a distributed data storage platform, not as data processor

Chosen technologies

- Discussed earlier:
  - **MongoDB** is a general purpose, document-based, distributed database
  - **Logstash** is the log management element of the ELK stack
  - **Cassandra** is a decentralized structured storage system
- **Hadoop** is an open source software framework designed for data storage and processing
  - Hadoop Distributed Filesystem (**HDFS**)
  - **HBase** non-relational database, (usually) runs on top of HDFS
  - **Zookeeper** coordination service
- Tom White, Hadoop – The definitive guide, O'Reilly, 4th Edition, 2015



Hadoop Distributed Filesystem

Hadoop in the timeline

	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
MapReduce																		
Hadoop*																		
CouchDB																		
Neo4j																		
Cassandra																		
MongoDB																		
Logstash																		

* Hadoop lives as a distributed data storage platform, not as data processor

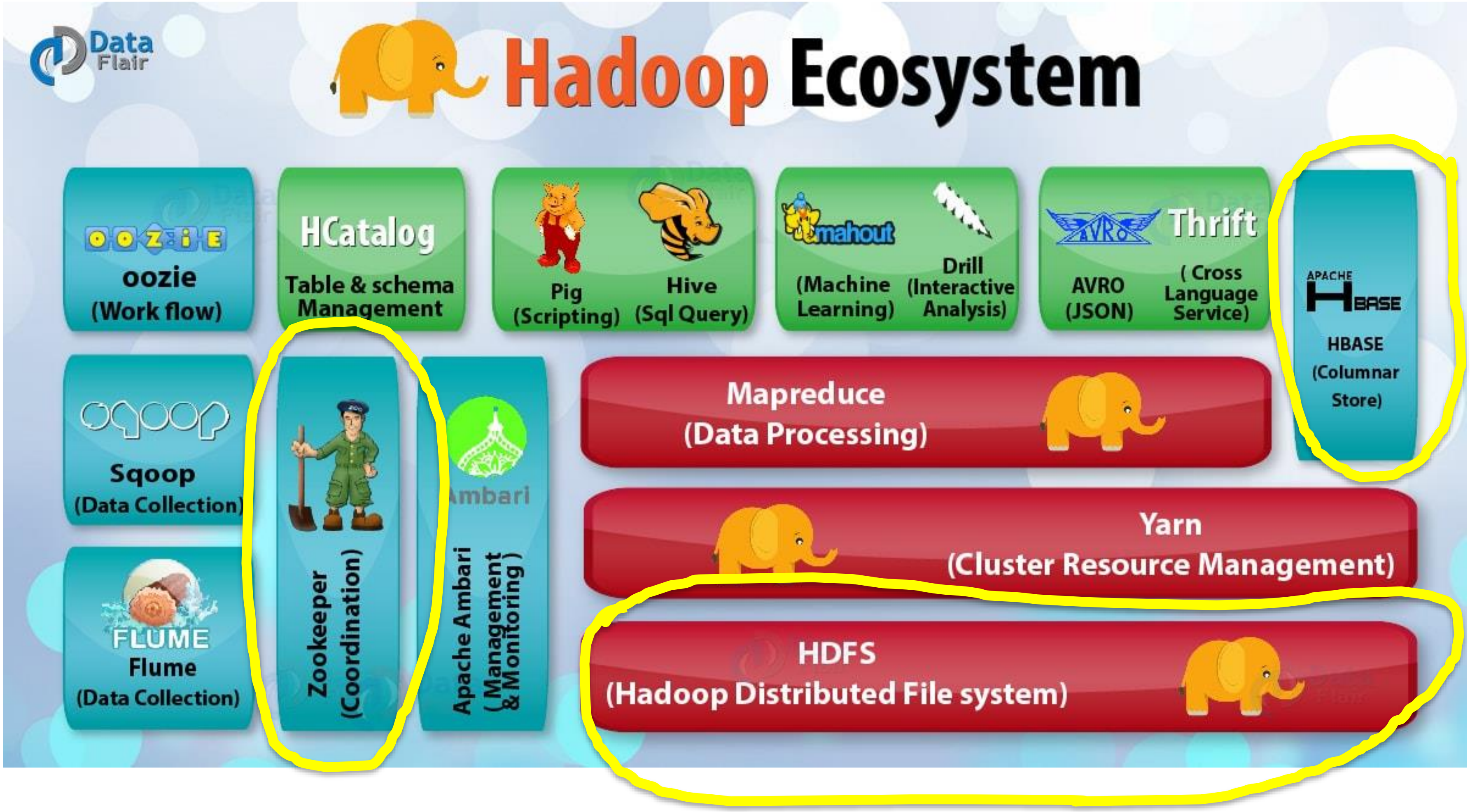
Why Hadoop @ OST?

Attempts: 40 out of 40

Please rate you past experience in using the Hadoop for data storage:

No prior experience	26 respondents	65 %
Heard/learned about it in an online or university course	12 respondents	30 %
My course project team used it		0 %
Used it myself in a course project	1 respondents	3 %
Used it professionally, i.e. in a for money project	1 respondents	3 %

The Hadoop Ecosystem



https://data-flair.training/blogs/hadoop-ecosystem-components/

Hadoop introduction

Definitions

- **DEF:** Hadoop is an open source software framework designed for storage and processing of large-scale data on clusters of commodity hardware
- Created by Doug Cutting and Mike Carafella in 2005.
- **Trivia:** Cutting named the program after his son’s toy elephant.

Use cases

- Data-intensive text processing
- Assembly of large genomes
- Graph mining
- Machine learning and data mining
- Large scale social network analysis

Hadoop motivation

Early large-scale computing

- Historically computation was processor-bound
  - Data volumes were relatively limited in size
  - Complicated computations were performed on that data
- Advances in computer technology was historically centered on improving the power of a single machine
- Today single machines (aka hosts/nodes) cannot handle the storage and processing needs of may use cases

Modern, distributed systems challenges

- “You know you have a distributed system when the crash of a computer you’ve never heard of stops you from getting any work done.” – Leslie Lamport
- Synchronizing data exchanges
- Managing a finite bandwidth
- Controlling computation timing is complicated
- Partial system failures
- More data in modern systems
- Data Node → Compute Node data copies are slow

HDFS defined

- **DEF:** Distributed filesystem = a filesystem which is capable to manage the storage of files (or generally data) across a distributed system consisting of networked computers (either cluster or grid)
  - Storage distribution is necessary when the datasets outgrows the storage capacity of the underlying hardware
- **DEF:** The Hadoop Distributed Filesystem (HDFS) is a filesystem designed for storing **very large**, bulk datasets on **commodity hardware** with **streaming data access** patterns
  - **Very large datasets** → up to terabytes in size
  - **Streaming data access** → write-once, read many times → read the whole or large segments of the data sequentially (streaming)
  - **Commodity hardware** → commonly available, inexpensive (server) HW available from multiple vendors, e.g. HP, Lenovo, Dell, etc.

File storage

Files are split into blocks

- The usual block size is 128 MB, but can be configured to be larger
- **Note:** A 1 MB piece of data does not consume 128 MB of space even if the block size is 128 MB

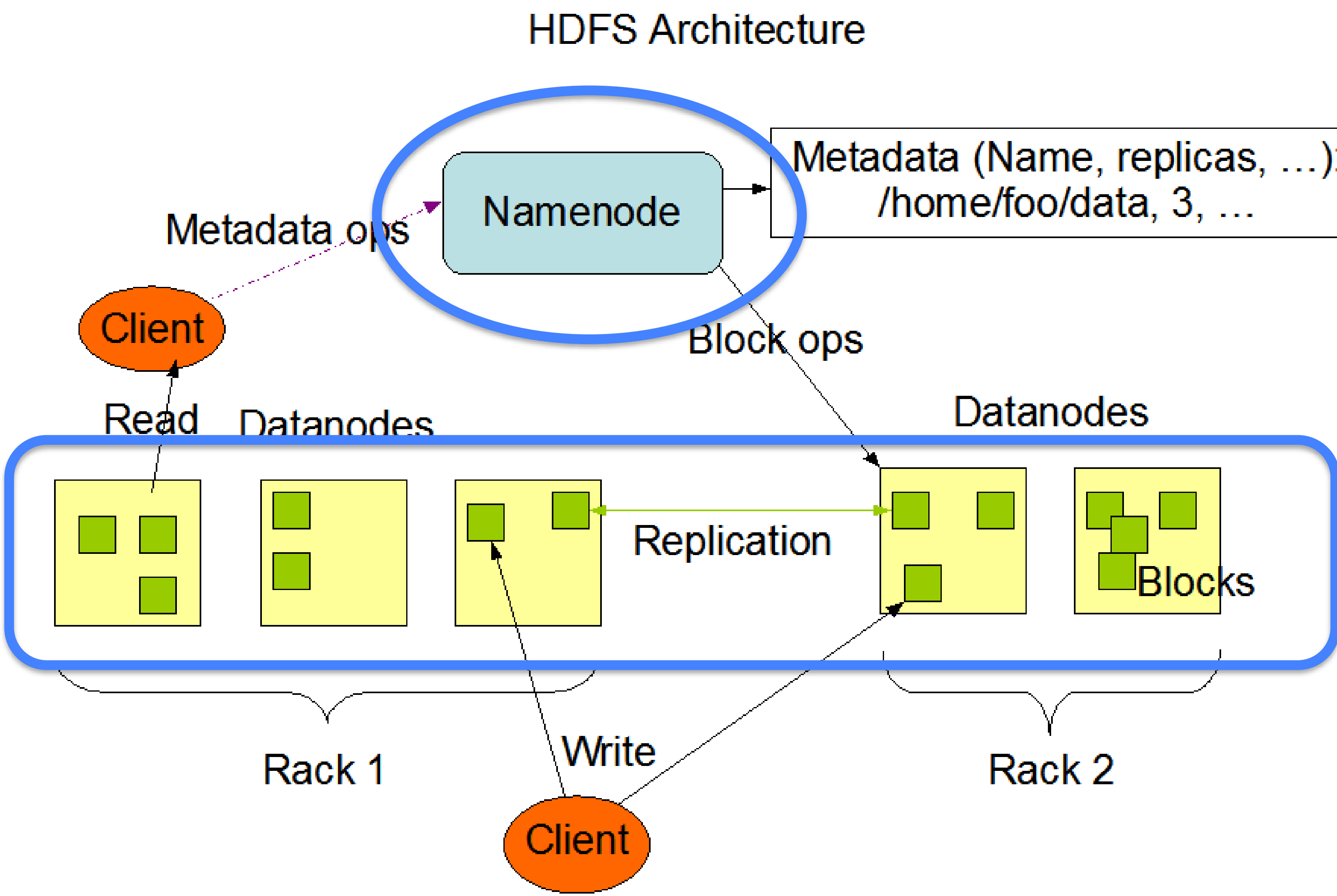
Blocks are split across many machines at load time

- Different blocks from the same file will be stored on different machines

Suboptimal use cases

- Low-latency (random) data access
- Lots of small files

HDFS Architecture

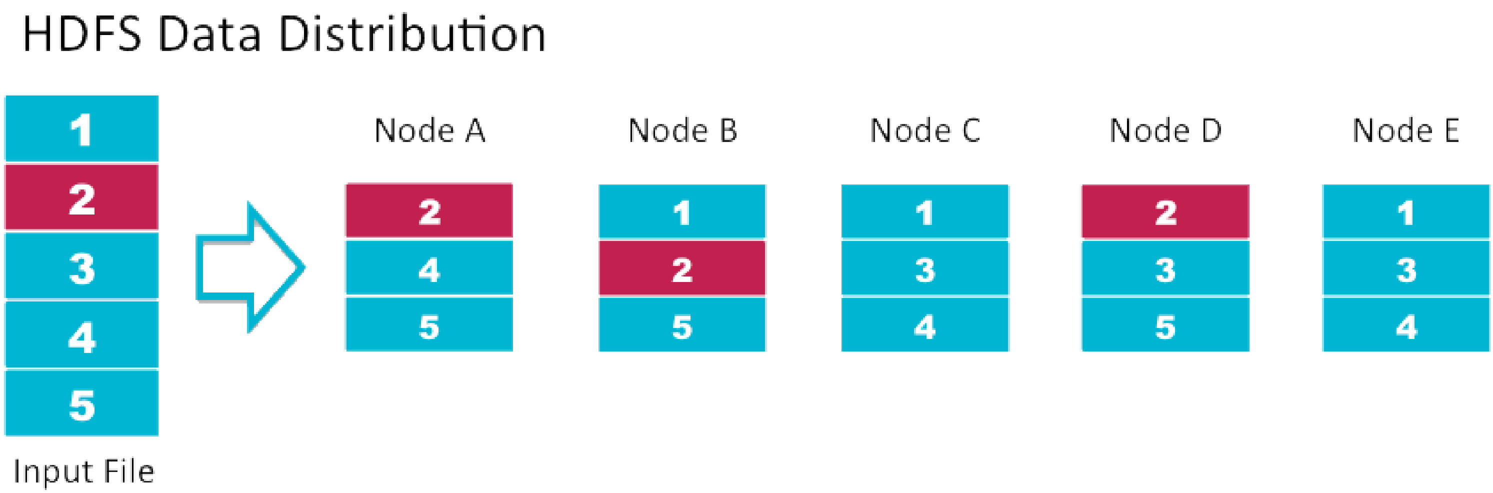


Namenodes

- HDFS namenodes manage the filesystem namespace
- The namenode **tracks the datanodes** on which the blocks for a given file are located
  - Block location information is not persisted → it is re-created when the system starts (simple!)
- The maintain an **in-memory** copy of
  - the filesystem tree, and
  - metadata for all files and directories in the tree.
- The **namenode state** is persisted on the local disk and consists of the following elements
  - Namespace image
  - Edit log

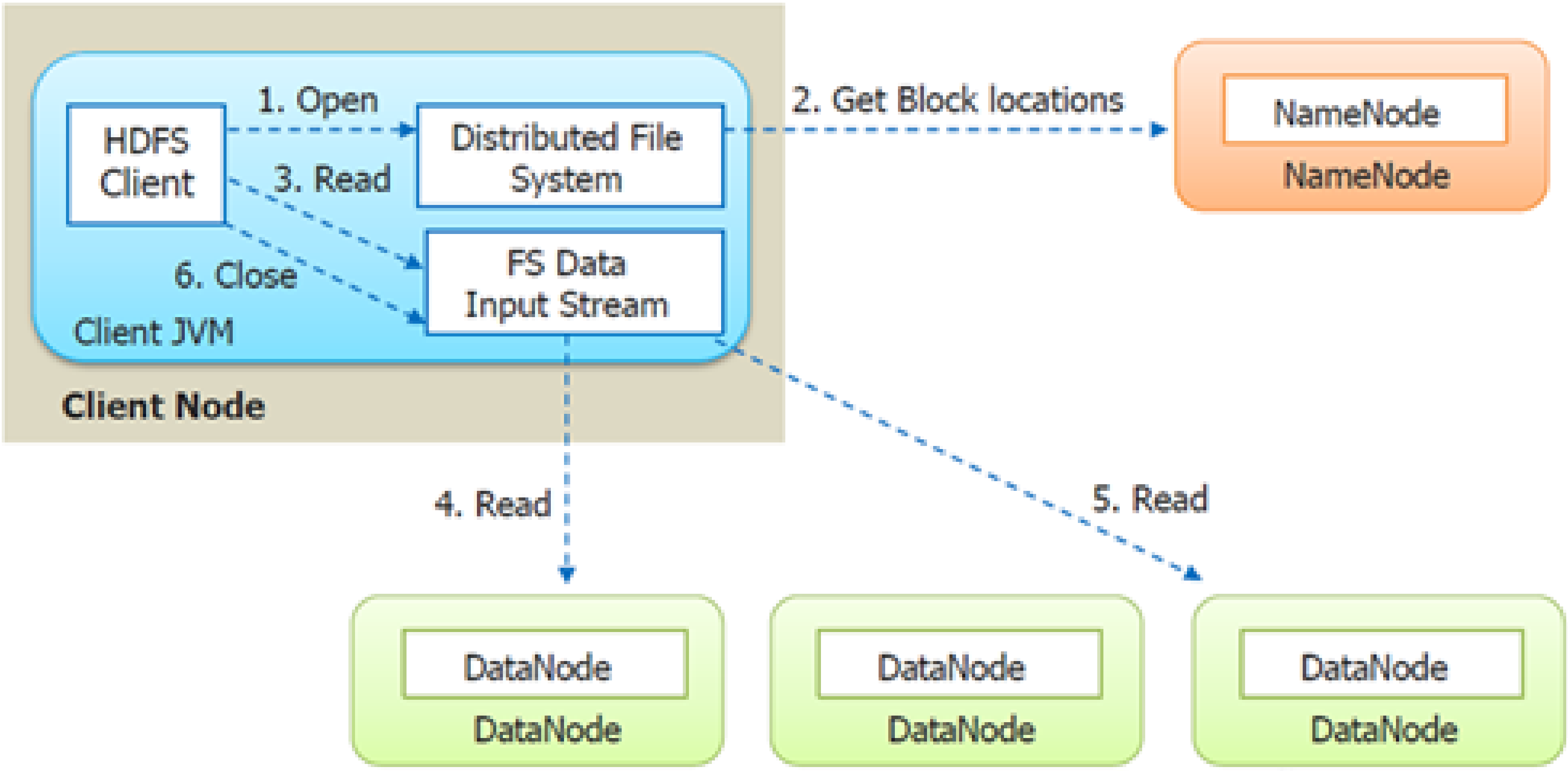
Datanodes

- Datanodes store and retrieve blocks
- Datanodes periodically report the list of blocks managed to the Namenode(s)



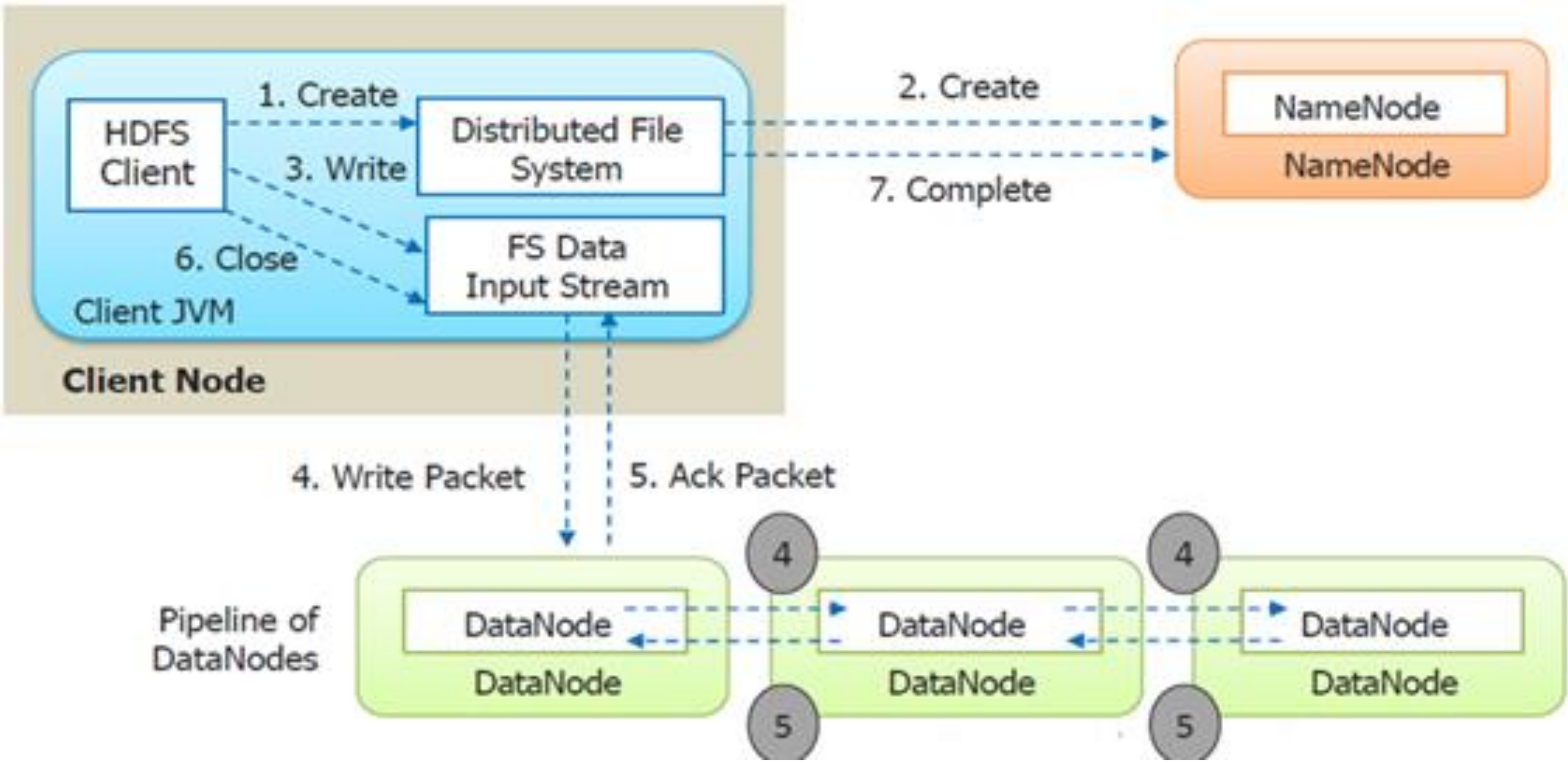
- Note: a completely failed Namenode would destroy the contents of the filesystem as it could not be reconstructed without the metadata stored on the Namenode

File read



Tom White, Hadoop – The definitive guide, O'Reilly, 4th Edition, 2015

File write



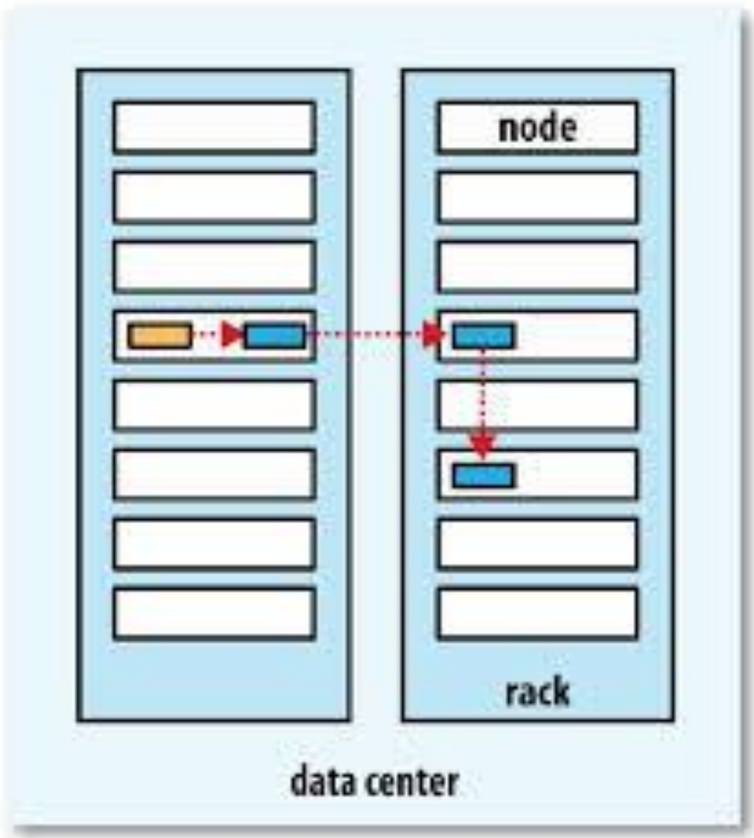
Tom White, Hadoop – The definitive guide, O'Reilly, 4th Edition, 2015

Namenode fault tolerance

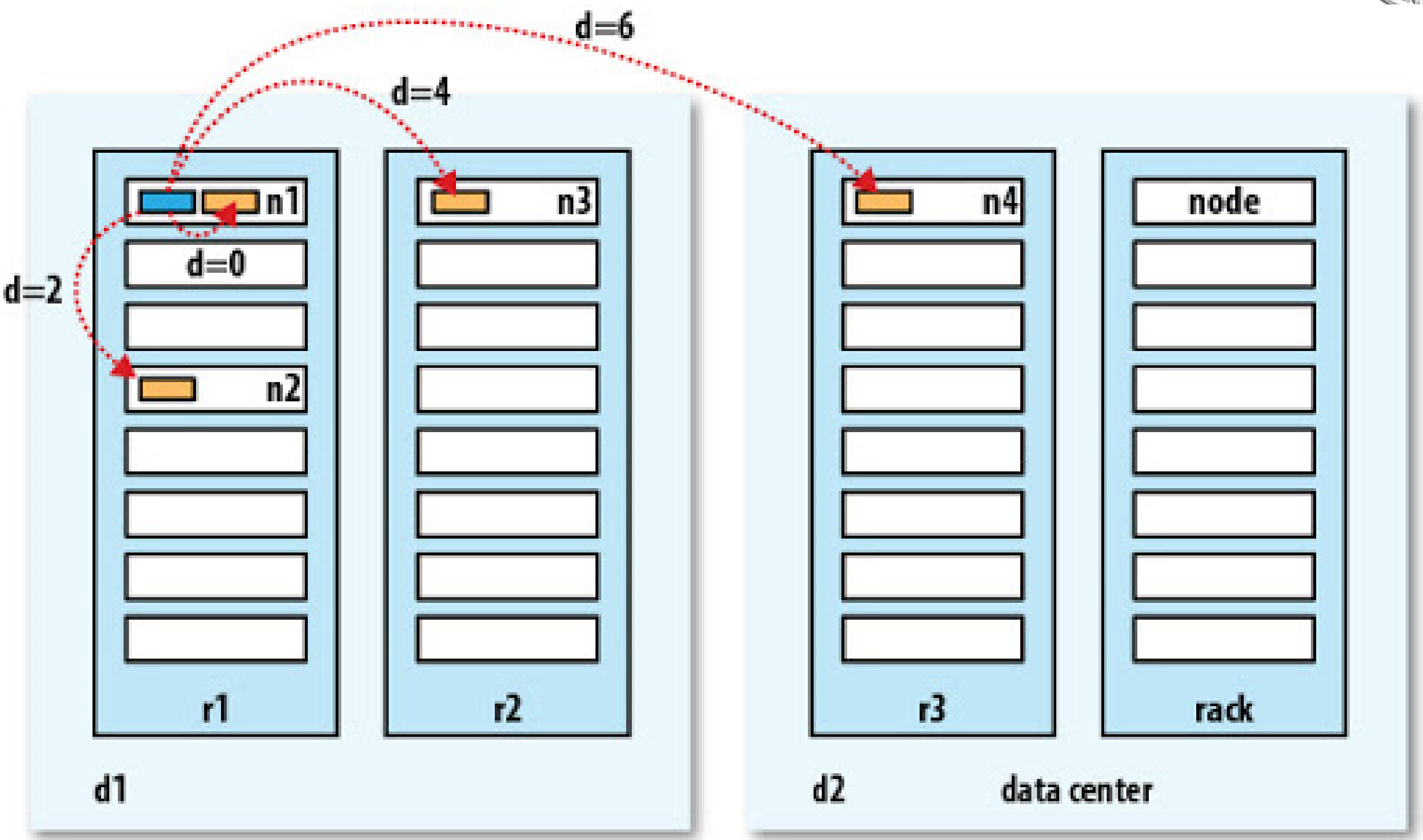
- Namenode instances are potential single points of failure (SPoF) → backup mechanisms are introduced
  - **Persist metadata to multiple filesystems**, usually local disk and a remote NFS mount
  - Introduce a **secondary namenode** which runs on a separate (physical) host which maintains a separate namespace image into which it merges the edit log → strong HW necessary
  - Clients need to be configured to handle namenode failover
- Namenode recovery steps in the case of primary failure:
  1. An administrator initializes a new primary namenode
  2. The namespace image is loaded into memory
  3. The (unmerged part of the) edit log is replayed
  4. Sufficient amount of block reports are received from the datanodes
  5. The (new) primary namenode leaves safe mode

Data replication

- How does the namenode decide where to store data replicas?
- Hadoop's default replication strategy is 3-fold (and implemented in the namenodes):
  - Place the 1st replica on the same node
  - Place the 2nd replica off-rack, i.e. in a randomly chose different rack
  - Place the 3rd replica in the rack of the 2nd replica but on a different node
- **Note:** HDFS clusters are usually limited to single data center



HDFS replication & distances



Tom White, Hadoop – The definitive guide, O'Reilly, 4th Edition, 2015

Who uses Hadoop/HDFS?

facebook
The New York Times
ebY
hp
IBM
JPMorganChase
eHarmony
twitter
intel
NETFLIX
rackspace HOSTING
amazon.com
VISA
NING
SAMSUNG
YAHOO!

HBase

Introduction

- DEF: **HBase** is a distributed column-oriented data store built on top of HDFS
  - HBase is the Hadoop solution for real-time, read/write random access to very large datasets
  - Development started in 2006 by Chad Walters & Jim Kellerman @ Powerset
  - Initial release: 2008
  - Preview release: Aug 2020
- The HBase data model is not relational and it natively does not support SQL
- Canonical use case: webtables of (billions of) crawled web pages

HDFS and HBase compared

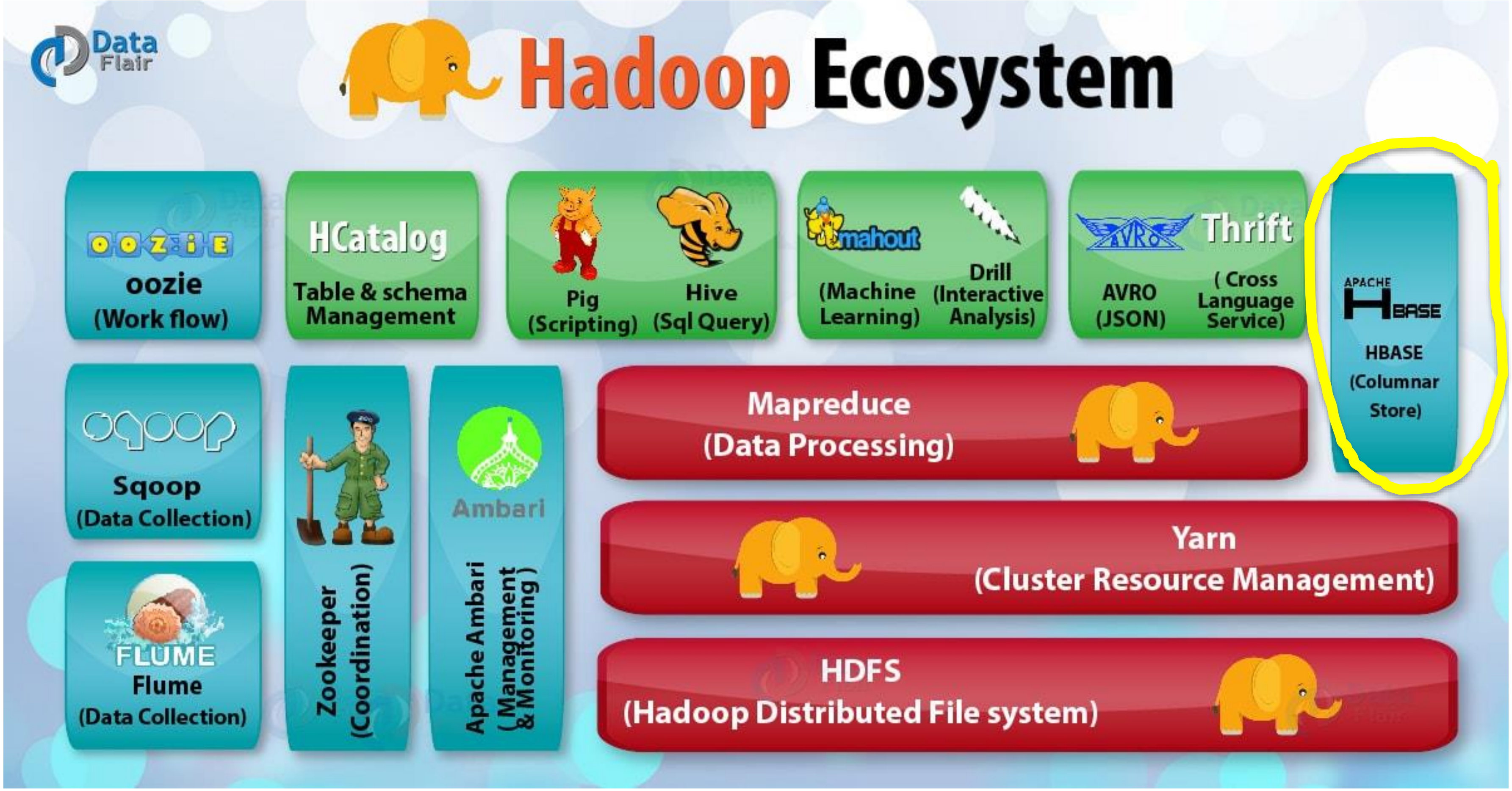
Plain old HDFS

- Designed for batch processing relying on scans over (very) big files
- Not good for record lookup → slow seek
- Not good for incremental addition of small batches → designed for bulk load
- Not good for updates → append or reload (!)

HBase on HDFS

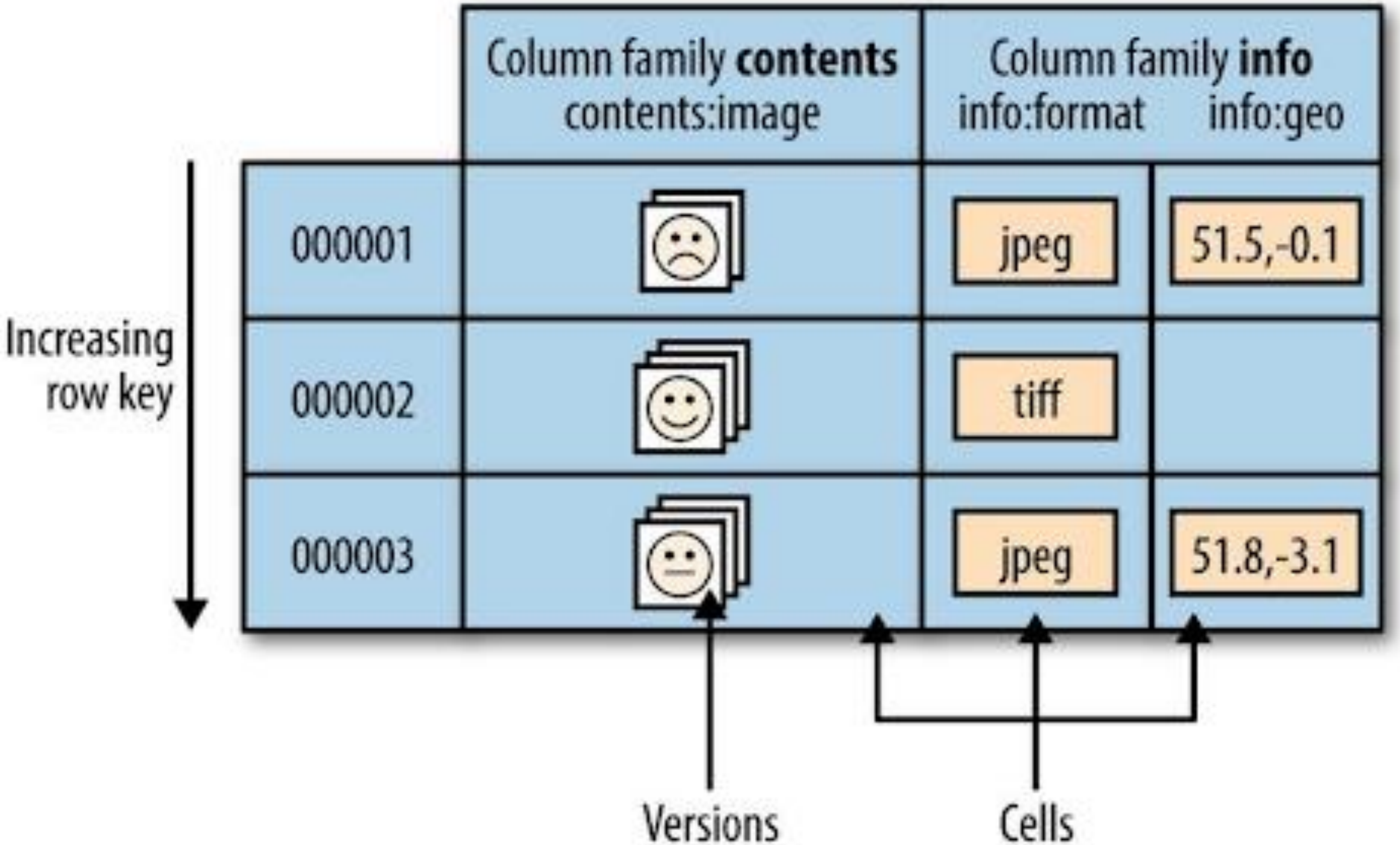
- Fast record lookup
- Support for record-level insertion → cell versioning
- Support for updates with versioning → row-level atomic updates

The Hadoop Ecosystem



https://data-flair.training/blogs/hadoop-ecosystem-components/

Data model



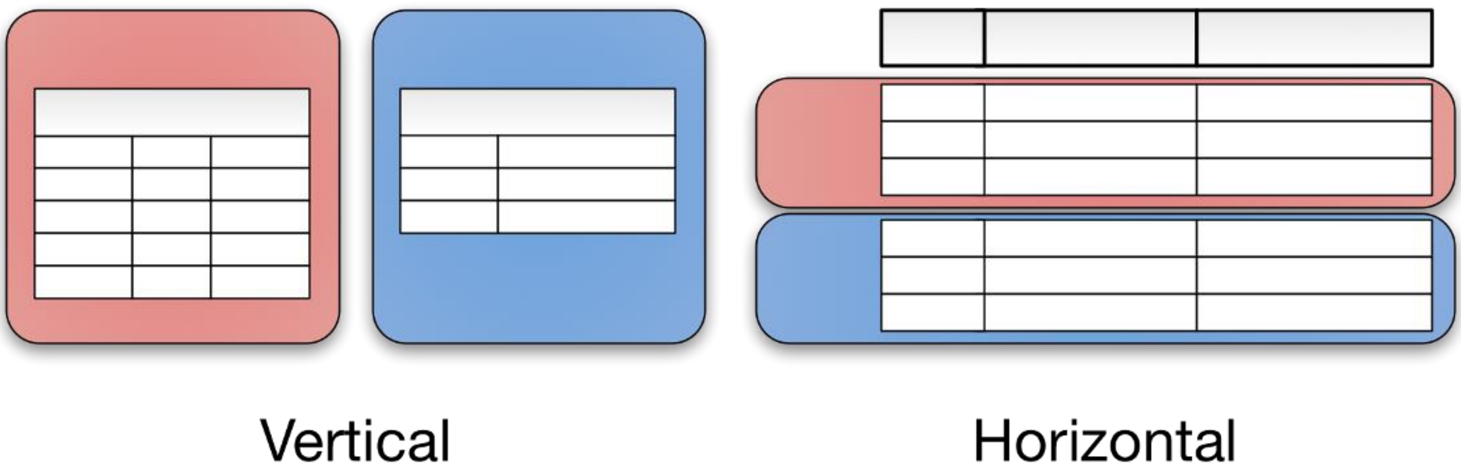
Tom White, Hadoop – The definitive guide, O'Reilly, 4th Edition, 2015

Data model described

- Tables are made of rows and columns similarly to RDBMS
- All table accesses are via **row keys**, aka primary keys
  - Row keys are byte arrays → keys can be of any data format
  - Table rows are sorted by row keys → the sort is byte-ordered
  - Row updates are atomic regardless of the column count → simple!
- Columns are grouped in **column families**
  - Share a common prefix, e.g. info:format, info:geo are members of the ‘info’ column family
  - Column family names consist of printable, human-readable characters
  - Column families are specified during schema definition
  - Columns can be added dynamically to column families
- **Table cells**
  - contain arrays of bytes
  - are versioned, usually timestamp assigned by HBase on entry

Data storage

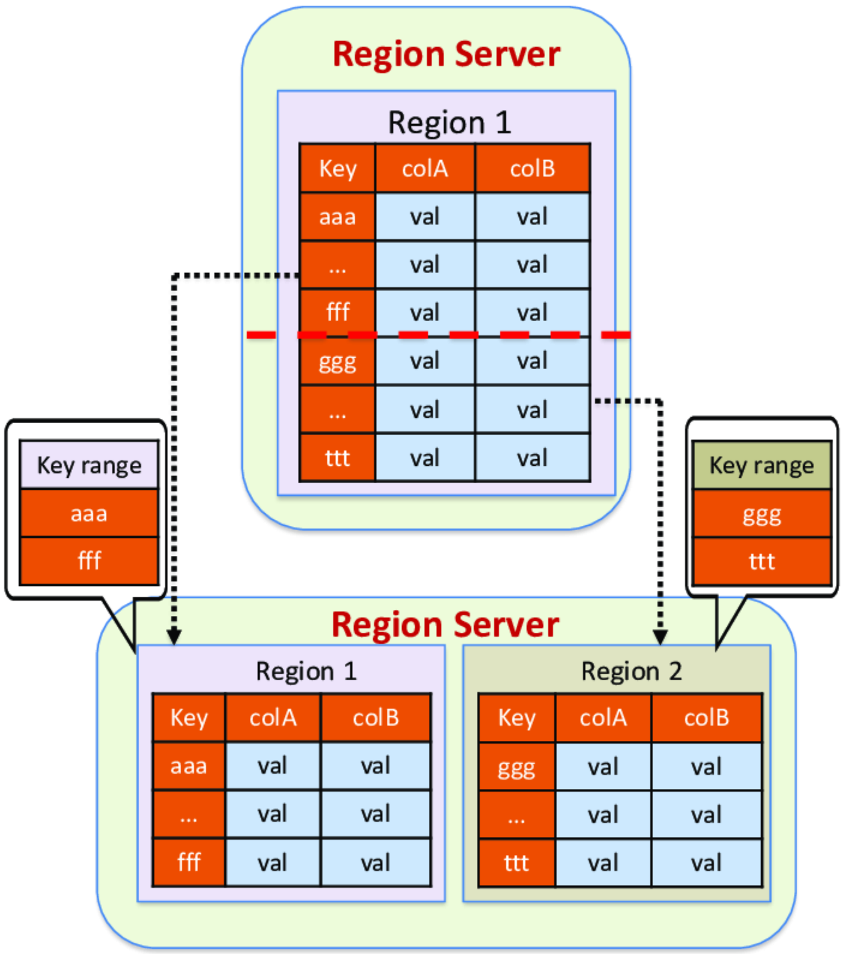
- Column families are stored together → column family-oriented store (!)
- HBase performs automatic horizontal partitioning of tables into **regions** as the table (out)grows (a single host)
  - A region is defined by the table name, first row (inclusive) and last row (exclusive)



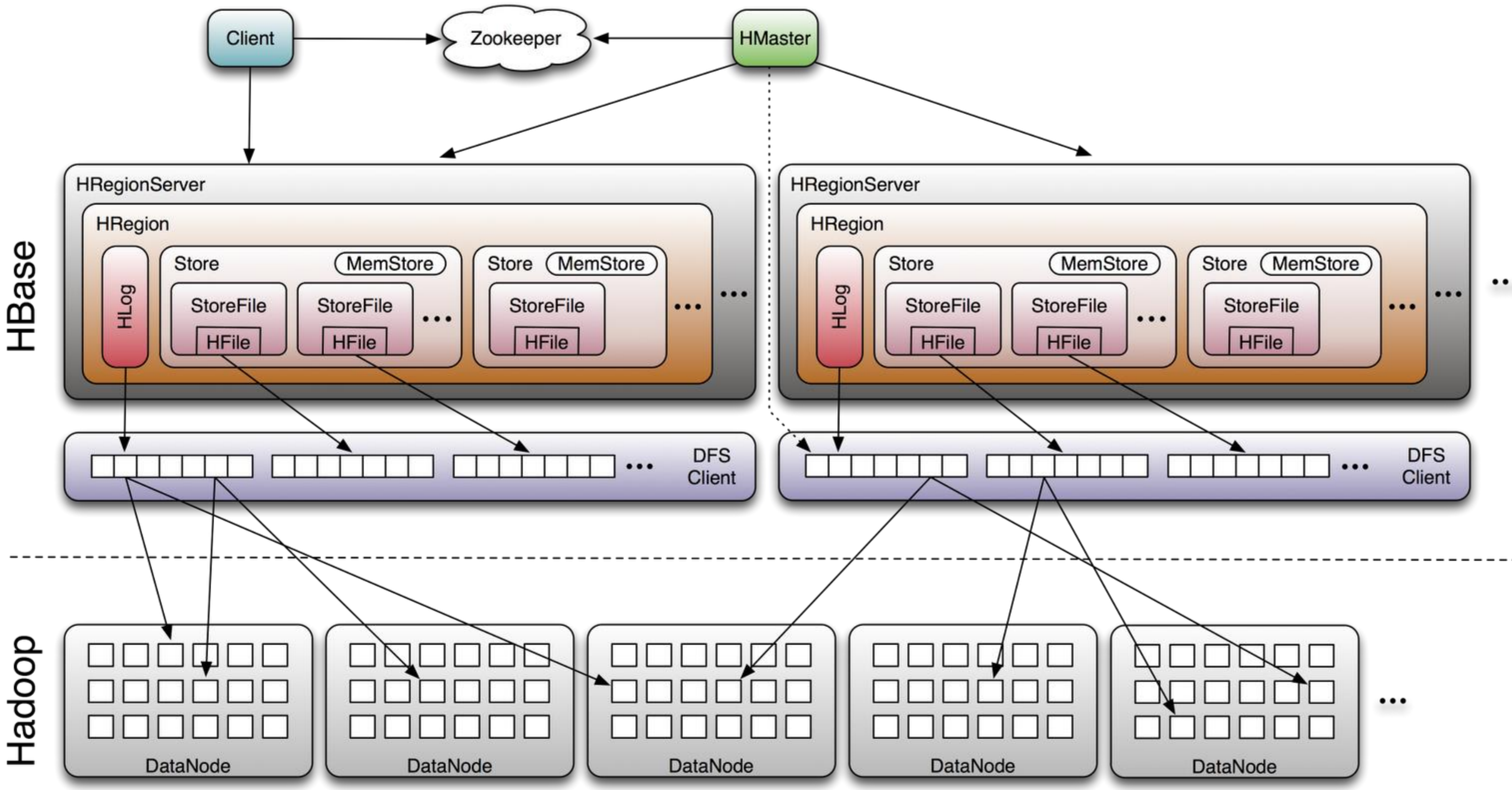
https://medium.com/@jeeyoungk/how-sharding-works-b4dec46b3f6

HBase table splitting

- Tables are initially hosted on a single node
- Tables are split horizontally when a table outgrows the confines of a single node, e.g. if there is insufficient storage capacity
  - The horizontal partitions are ‘regions’
- Region servers are responsible for the storage of regions



Satellite view



http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html

Old(ie-goldie) meets new

RDBMS

1. Initial public launch of web server with a traditional RDBMS data store
2. Service popular → too many RDBMS reads → **memcache** common queries
3. Service even more popular → too many RDBMS writes → buy custom, **expensive HW**
4. New website features (requested by the millions of customers) increase (SQL) query complexity → **de-normalize data**
5. RDBMS server(s) overloaded, the website is too slow → **avoid joins, pre-materialize data** and keep in-memory
6. Reads work, writes are still slow → **no way forward** with an RDBMS

HBase(d) solution

- **No indexing** → performance independent of table size
- **Automatic partitioning** → as the data grows, it is split into regions
- **Linear scalability** → regions automatically utilize new (server) nodes added to the HW cluster
- **Commodity hardware** → runs on clusters of servers worth up to 5000 USD
- **Fault tolerance** → numerous cheap nodes lessen the worry caused by node downtime

Zookeeper

Introduction

Definition

- Zookeeper is a highly-available process coordination services in distributed systems (DS).
- Designed and written to allow computer scientists and programmers to **not worry about coordination** in DS

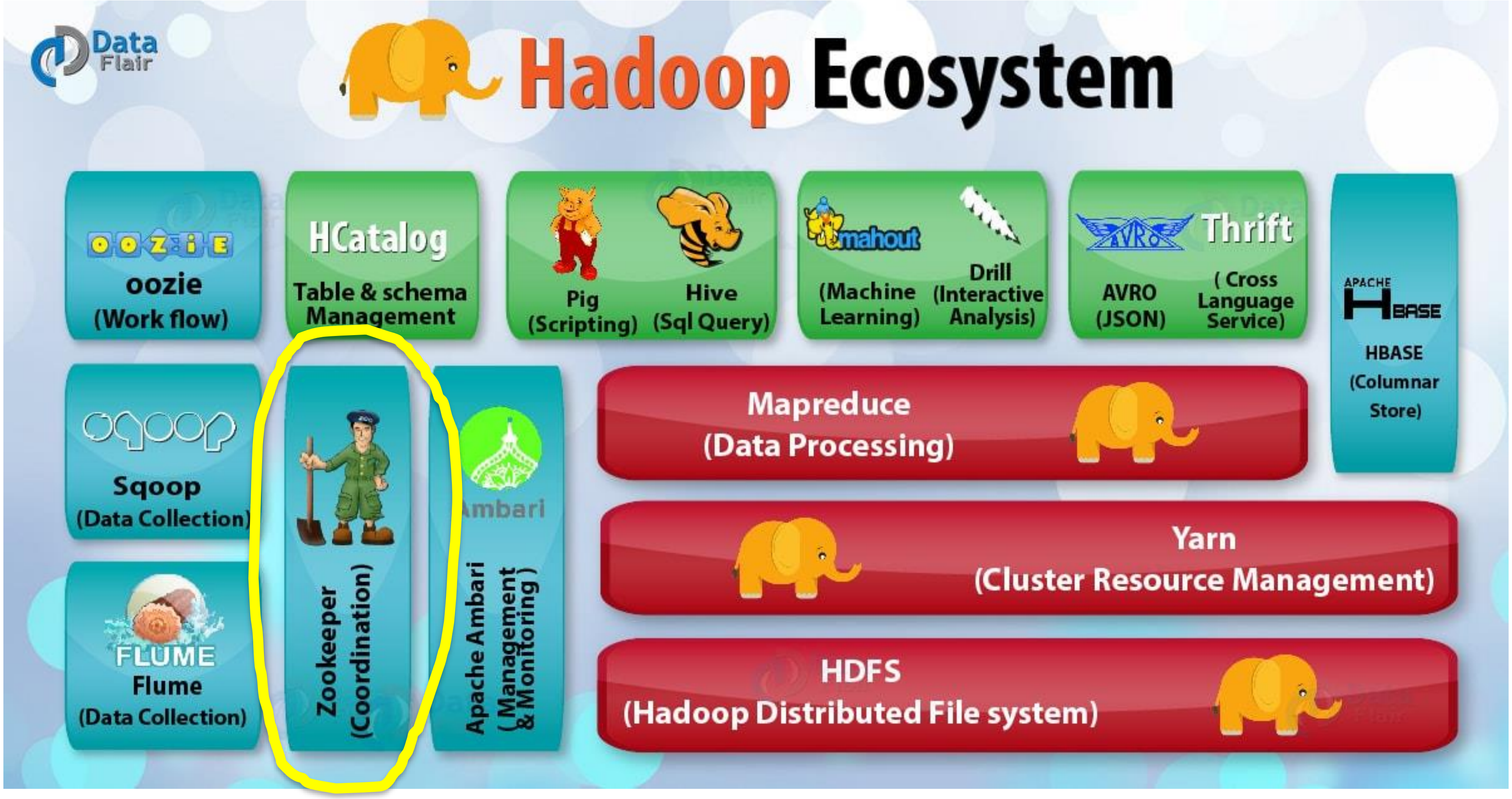
Motivation

- No single computer can manage the vast amounts of data on the web scale → necessary to use distributed systems
- Solve (some of) the ages old challenges in distributed systems
  - Network reliability
  - Latency
  - Topology changes
  - Heterogenous systems
  - Limited bandwidth
  - Faults

Challenges to be solved

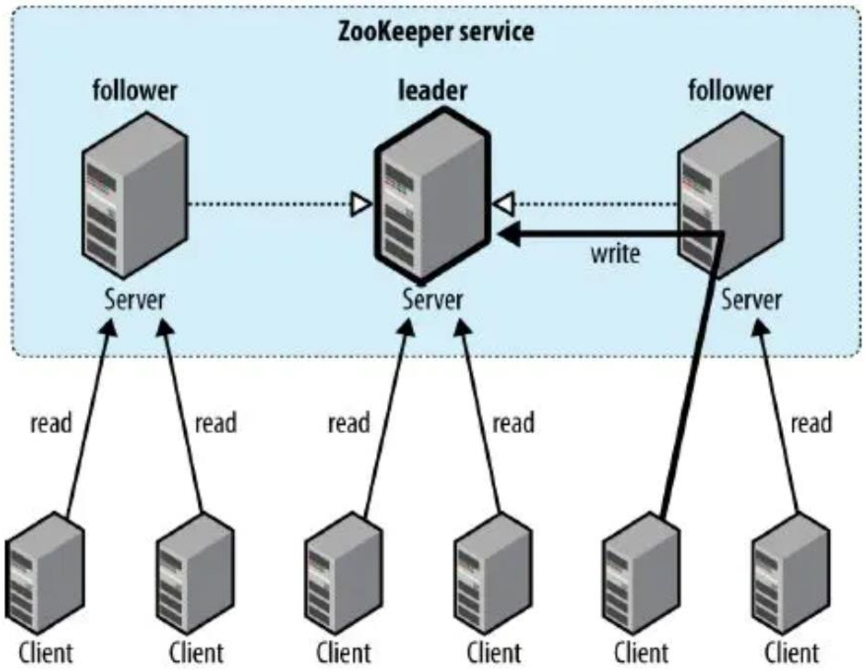
- **Coordinator selection** → how to select a process from a group of equal processes
- **Locking** → how to coordinate the use of limited resources, e.g. off-site storage available via a slow link
- **Configuration management** → how to ensure that each node in a 150-node computing cluster receives the same configuration update

The Hadoop Ecosystem



https://data-flair.training/blogs/hadoop-ecosystem-components/

Zookeeper system architecture

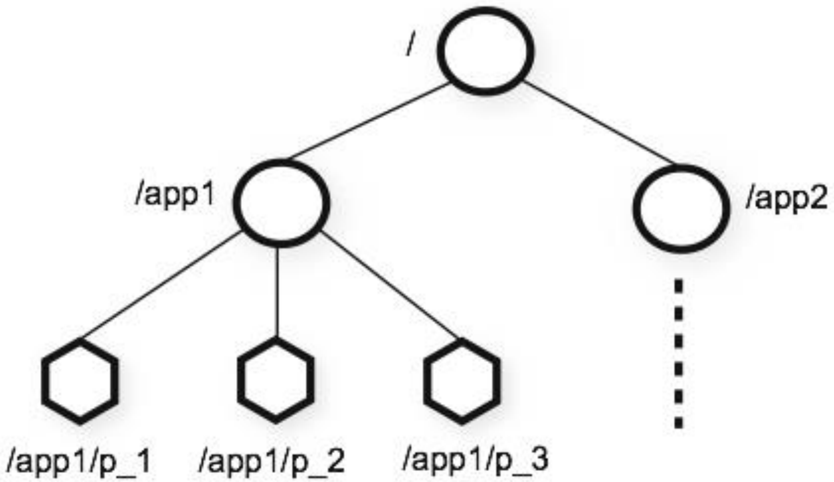


(Figre 21-2) from Tom White, Hadoop – The definitive guide, O'Reilly

- ZooKeeper service → clients observe it as a single component
- Ensemble → cluster of zookeeper servers
- Server → provides access to Zookeeper
  - Leader → write point
  - Follower → replicas
- Client → ZK service client, usually a node in a(ny) cluster

Data model

- ZK maintains a tree of nodes
  - Nodes are named ‘znodes’
- znodes are referenced by paths
  - znodes path namespaces are similar to filenames on Linux
- znode types
  - Ephemeral → tied to a client session & deleted when the session ends
  - Persistent → not tied to a client session, explicit delete
- Zookeeper is designed for coordination, not data storage
  - Data size is limited to 1 MB



Operations

- create = create a znode – the parent must exist
- delete = remove a znode without children nodes
- exists = chek node existence
- getChildren = get list of children nodes
- getData, setData = access/modify the data associated with a node
- getACL, setACL = access/modify the access control list (ACL) of a node
- sync = synchronize a client’s view with a znode

- multi = batch together multiple primitive operations into a transaction

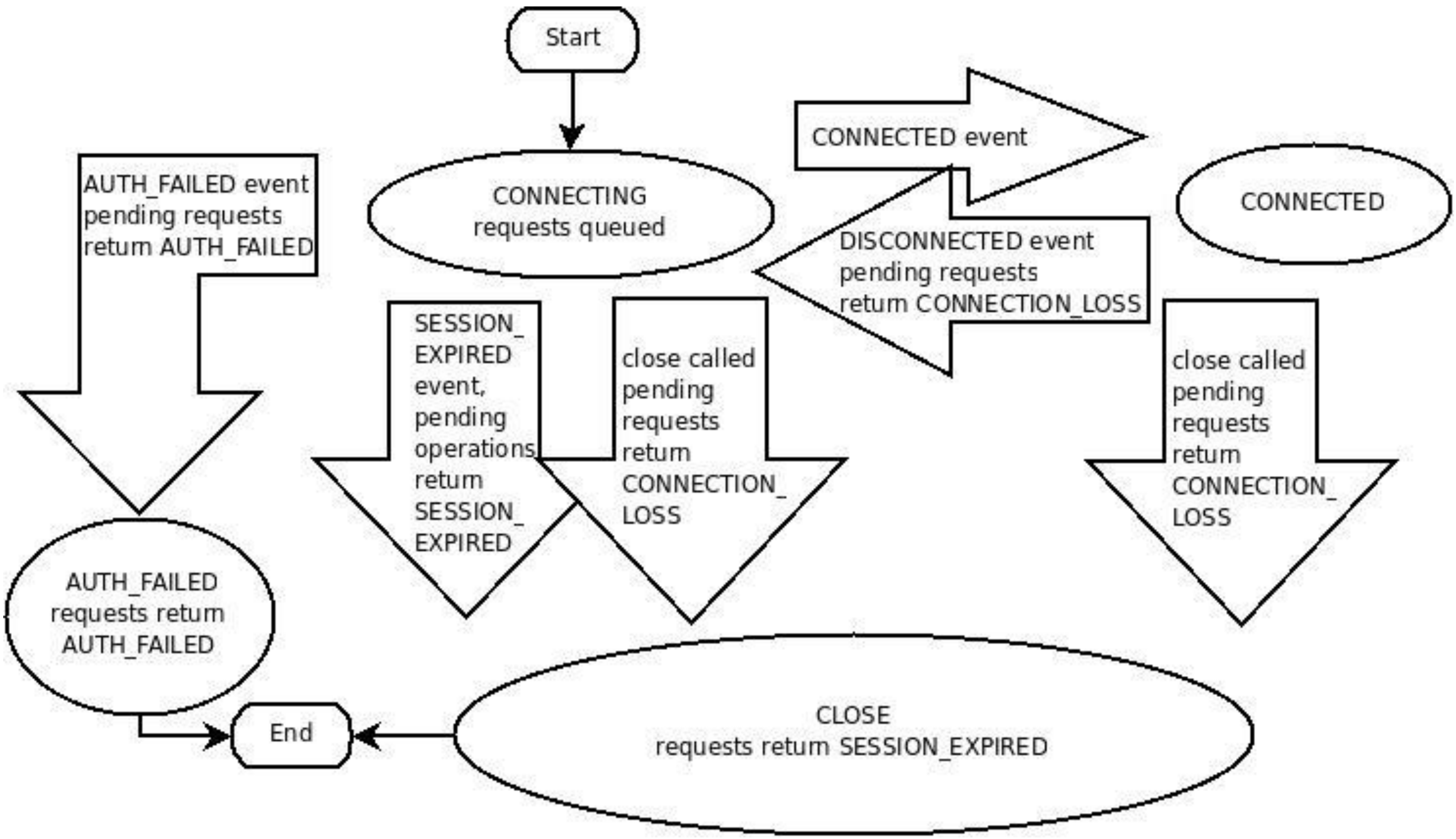
Keeping clients updated

- Watch triggers are used to notify clients
- Watches can be set on:
  - ‘exists’ → triggered when the watched znode is created, deleted or its data is updated
  - ‘getData’ → triggered when the data watched is deleted or updated
  - ‘getChildren → triggered when any child of the watched znode is modified

Sessions

- ZK clients are configured with a (sub)set of servers in the ensemble
- Clients try to connect to the configured servers iteratively
- Once a successful connection is made, a session is established
  - Sessions are long-lasting, but can time out
  - Clients need to maintain sessions by sending heartbeat signals to the server
  - If the timeout period expires, the server tears down the session and all ephemeral znodes are deleted
- Failover is handled by the ZK client
  - Sessions and ephemeral nodes are persisted in the ensemble and survive partial failures

Sessions depicted



https://zookeeper.apache.org/doc/r3.3.3/zookeeperProgrammers.html

Synchronization and consistency

- ZK ensures that every modification is replicated to the majority of the servers in the ensemble
  - The ZooKeeper Atomic Broadcast (ZAB) protocol (2008) is used
    - Gossip or Paxos are not used, although they are similar
  - Zab operates in two phases:
    1. Leader election → the servers in the ensemble select a ‘leader’ – this phase is over when the majority of servers is synced with the leader
    2. Atomic broadcast → all write requests are forwarded to the ‘leader’, which broadcasts them to the followers
- Zookeeper ensembles consist of odd numbers of servers
  - The ensemble is operational if more than half of its servers are up → in a 5-node ensemble two nodes might fail wo operational disruption

Data consistency guarantees

- **Sequential consistency** → updates from a particular client are applied sequentially, i.e. in the order they are observed by the ZK ensemble
- **Atomicity** → updates either succeed or fail (on the ensemble level)
- **Single system image** → clients see the same view of the system regardless of the ZK server chosen and connected
- **Durability** → successful updates survive server failures
- **Timeliness** → delays in synchronization are guaranteed to be short and measured in multiples of 10 seconds
  - Servers with stale data shut down → they avoid to serve clients with 'old' data

Security

Authentication

- Authentication is optional
- Authentication variants
  - 'digest' → authentication with username & password
  - 'sasl' → Kerberos
  - 'ip' → IP address-based auth

Authorization

- Authorization via Access Control Lists (ACL)
- Access levels:
  - CREATE – node creation
  - READ – read children/data
  - WRITE – allowed to setData
  - DELETE – delete child node
  - ADMIN – ability to setACL

Who uses ZooKeeper?

PMC Members

ZooKeeper's active PMC members are

Username	Name	Organization	Time Zone
tdunning	Ted Dunning	MapR Technologies	-8
camille	Camille Fournier	RentTheRunway	-5
phunt	Patrick Hunt	Cloudera Inc.	-8
fpj	Flavio Junqueira	Confluent	+0
ivank	Ivan Kelly	Midokura	+2
mahadev	Mahadev Konar	Hortonworks Inc.	-8
michim	Michi Mutsuzaki	Nicira	-8
cnauroth	Chris Nauroth	Hortonworks Inc.	-8
breed	Benjamin Reed	Facebook	-8
henry	Henry Robinson	Cloudera Inc.	-8
rgs	Raul Gutierrez Segales	Pinterest	-8
rakeshr	Rakesh Radhakrishnan	Intel	+5:30
hanm	Michael Han	Twitter	-8
andor	Andor Molnar	Cloudera Inc.	+1

Summary

- Hadoop Distributed Filesystem (**HDFS**)
- **HBase** non-relational database, (usually) runs on top of HDFS
- **ZooKeeper** (cluster) coordination service

Thank you for your attention!