From Big Data to Data Platform

Research and Challenges

About me

Enrico Gallinucci, PhD

- Junior Assistant Professor
- Member of the Business Intelligence Group
- Doing research since 2013
- Teacher of Business Intelligence and Big Data
- Main research interests:
  - big data
  - data modeling
  - social bi, self-service bi
  - trajectory data
  - precision agriculture

Program

- Lecture 1: Introduction to Big Data
- Lecture 2: Polyglot Persistence
- Lecture 3: The Metadata Challenge

Exam

Write and send me a short essay

- About 5-10 pages long
- Discuss the relationships between your research interests and the topics presented in the course
- Explain how big data challenges may have an impact on your work

An introduction to Big Data

What we are going to do

- Define big data
- Why the hype behind big data
- Architectures
  - Hardware
  - Software
    - Data storage
    - Data processing (batch, interactive, streaming)
  - Reference architectures

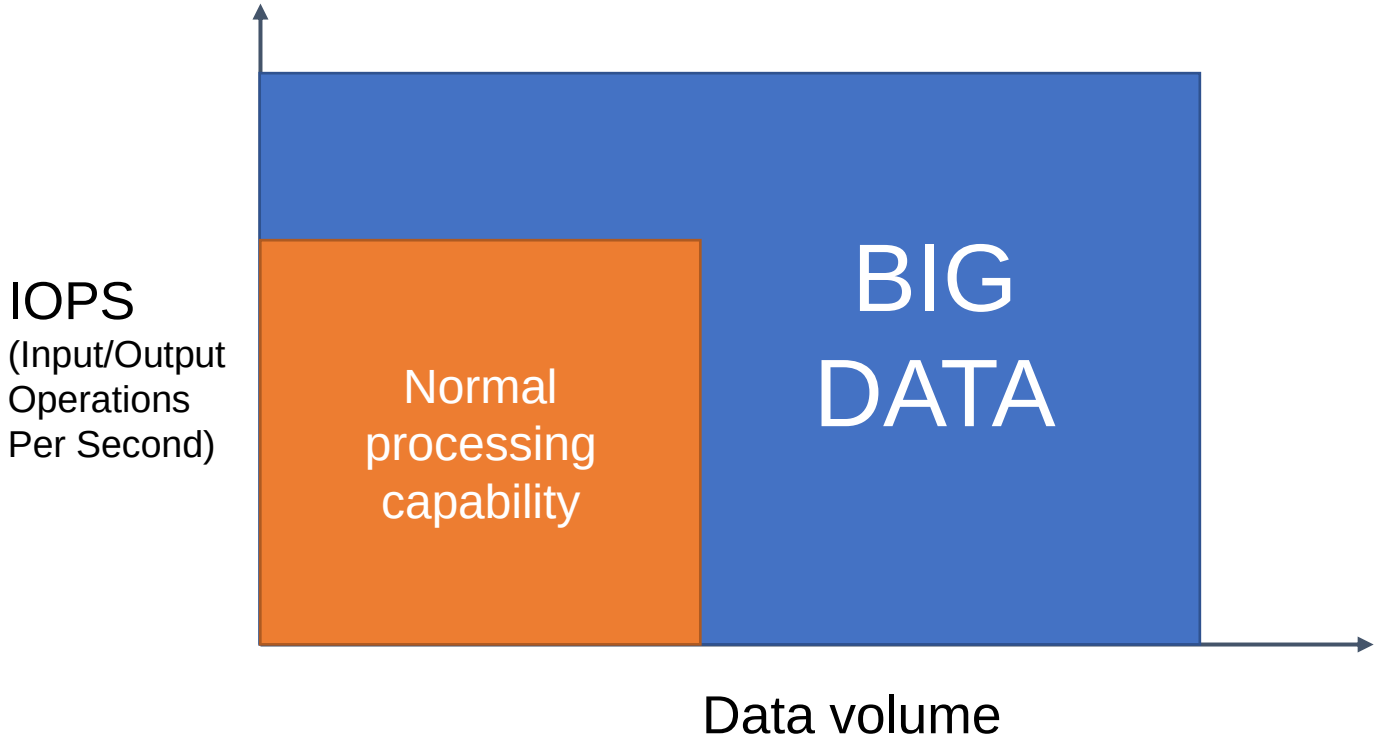
Define *big data*

The term *big data* comes in two flavors

- Noun: “We have *big data*"
- Adjective: “We use *big data* tools"

Big data as a noun

When does data become *big*?



Big data as a noun

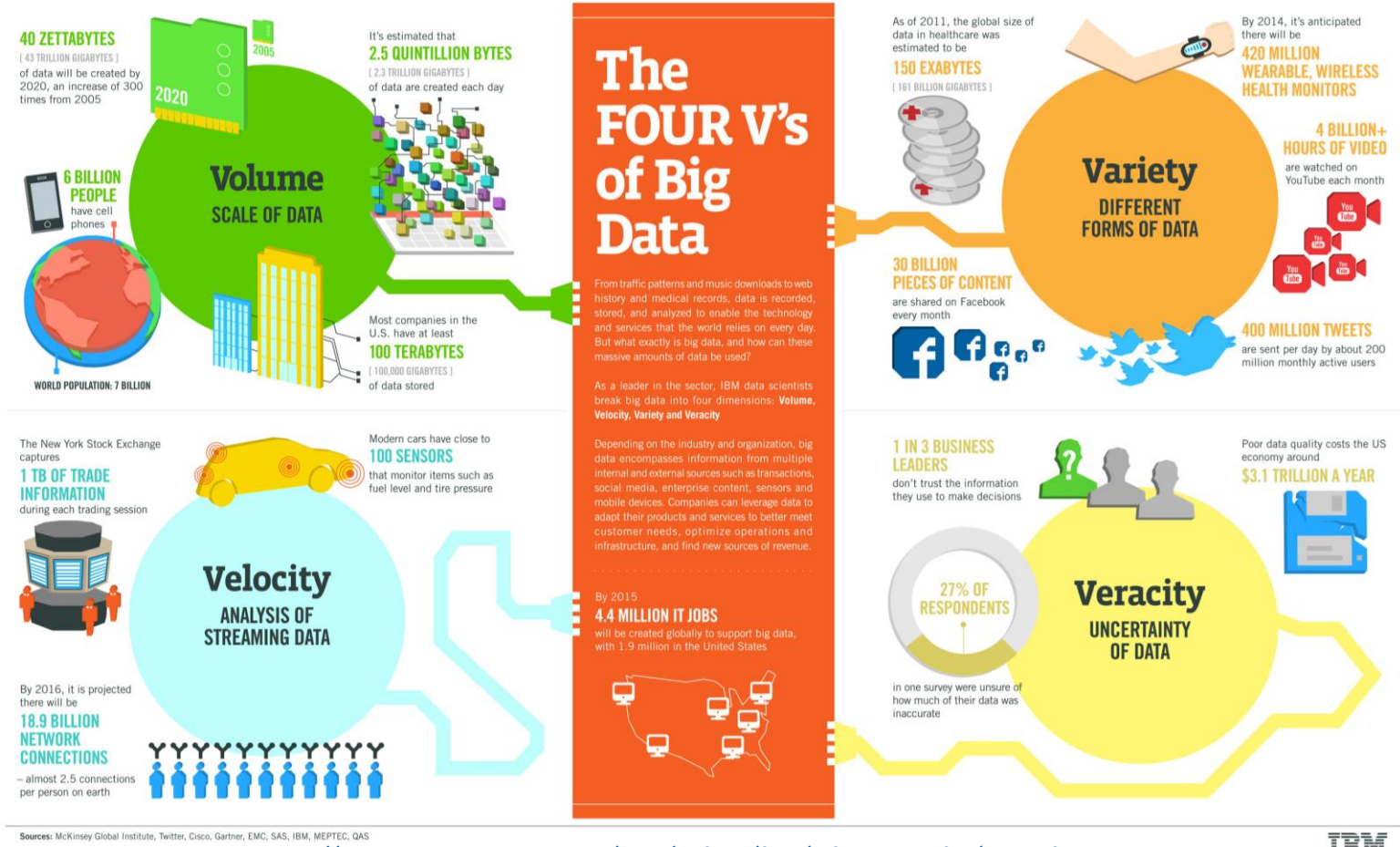
Common definitions

- "Big data **exceeds the reach of commonly used hardware** environments **and software** tools to capture, manage, and process it with in a tolerable elapsed time for its user population."
  Teradata Magazine article, 2011

- "Big data refers to data sets whose **size is beyond the ability of typical** database **software** tools to capture, store, manage and analyze."
  The McKinsey Global Institute, 2012

- "Big data is data sets that are **so voluminous and complex that traditional** data processing application **softwares are inadequate** to deal with them."
  Wikipedia

Big data as a noun – The V's



http://www.ibmbigdatahub.com/sites/default/files/infographic_file/4-Vs-of-big-data.jpg

Big data as a noun – The V’s

Volume

- Large quantity of data

Velocity

- Refers to the speed of data production…
- …and to the speed of consumption and analysis

Variety

- Structured, unstructured, multimedia

Veracity

- Refers to the trustworthiness of data
- Potentially inconsistent, incomplete, ambiguous, obsolete, false, approximate

Big data as a noun – The V’s

How many V’s?

Big data as an adjective

When used as a noun, the boundary between *normal* and *big* data is vague

When used as an adjective, its meaning is more specific

- Big data architecture (e.g., the Lambda architecture)
- Big data tools (e.g., Apache Spark)
- Big data paradigm (e.g., Map-Reduce)

Why the hype

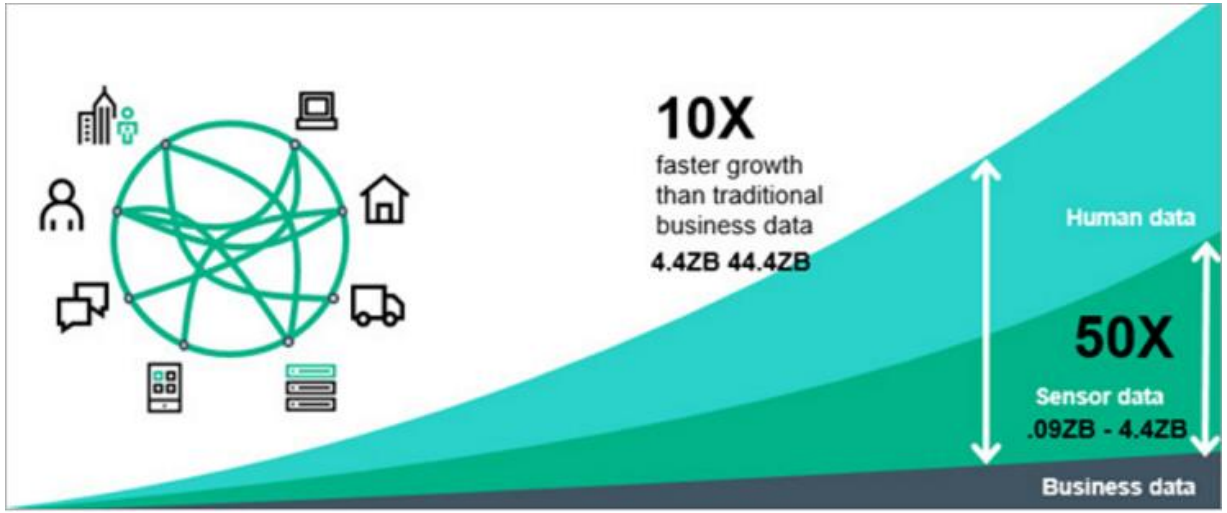
We have always known that data is powerful

What has changed?

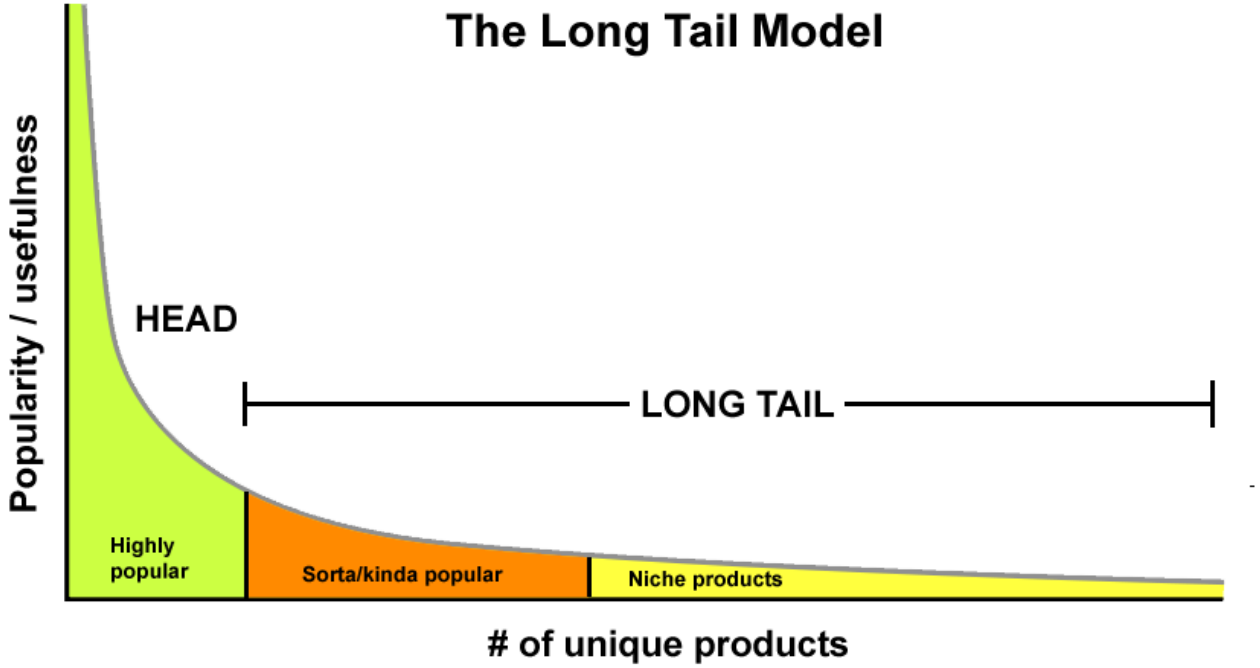
Data growth

New sources that drive an exponential growth of data

- Opens to new analytical opportunities



The long tail



"We sold more items today that didn't sell at all yesterday than we sold today of all the items that did sell yesterday" - *Amazon employee*

The highest value does not come from the small set of highly popular items, but from the long list of niche items

- Put together, the *insignificant* data is actually the most valuable
- The inverse of the 80-20 Pareto rule

Bigger = Smarter

Google Translate

- You collect snippets of translations
- You match sentences to snippets
- You continuously debug your system

Why does it work?

- There are tons of snippets on the Web
- The accuracy improves as the training set grows

Architectures

Hardware-wise

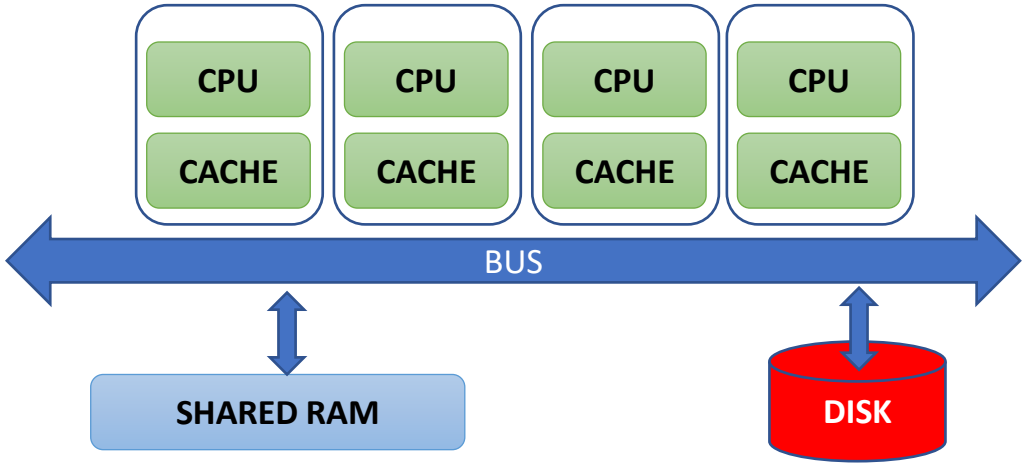
Software-wise

- Data storage (files, databases)
- Data processing (batch, interactive, streaming)
- Reference architecture (components, lambda vs kappa)

Scaling

First of all, big data is *big*

- It doesn't fit a single drive
- It doesn't fit a single (typical) machine
  - *Symmetric Multi Processing* (SMP): several processors share the same RAM, the same I/O bus and the same disk(s)
  - Limited number of mountable physical devices
  - BUS bottleneck



Second, processing big data requires a lot of computing resources

- Simply scaling only the disk is not an option

What do we do?

- Scale up
- Scale out

Scale up

Generally refers to adding more processors and RAM, buying a more expensive and robust server

Pros

- Less power consumption than running multiple servers
- Cooling costs are less than scaling horizontally
- Generally less challenging to implement
- Less licensing costs
- Less networking equipment

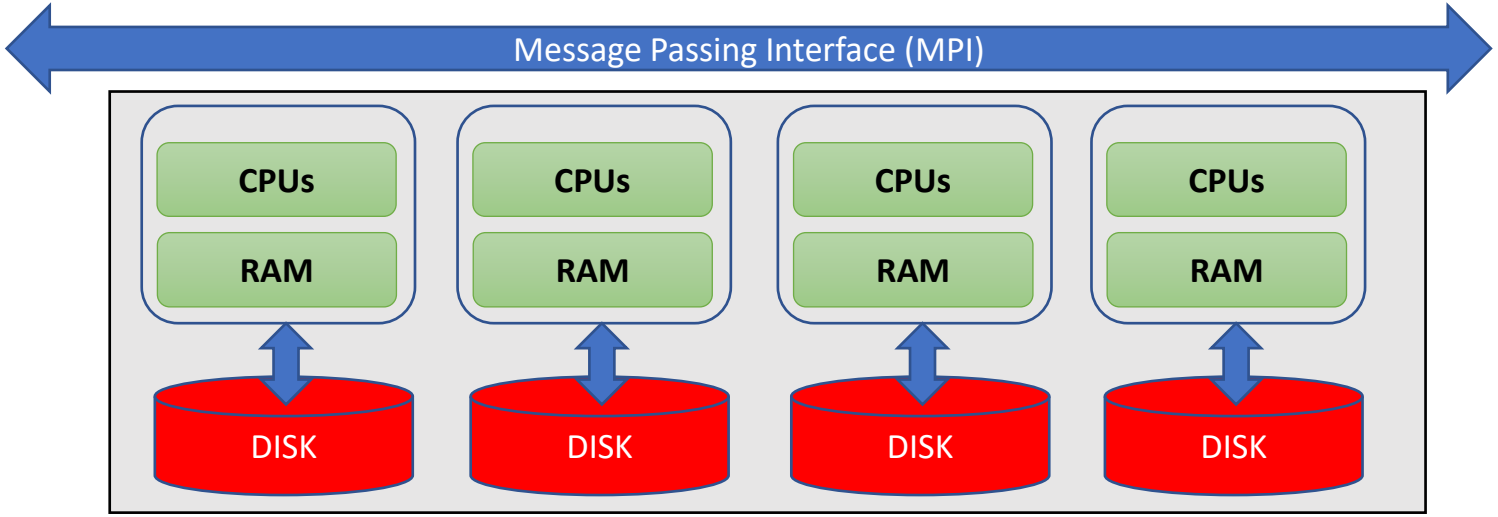
Cons

- PRICE
- Greater risk of hardware failure causing bigger outages
- Generally severe vendor lock-in
- Not long-term: limited upgradeability in the future

MPP

In a *Massively Parallel Processing* (MPP) architecture there are several processors, equipped with their own RAM and disks, collaborating to solve a single problem by splitting it in several independent tasks

- It is also called shared-nothing architecture
- Mainly used for Data Warehouse applications (e.g., Teradata, Nettezza, Vertica)



Scale out

Generally refers to adding more servers with less processors and RAM

Pros

- Much cheaper than scaling vertically
- New technologies simplify fault-tolerance and systems monitoring
- Easy to upgrade
- Usually cheaper
- Can literally scale infinitely

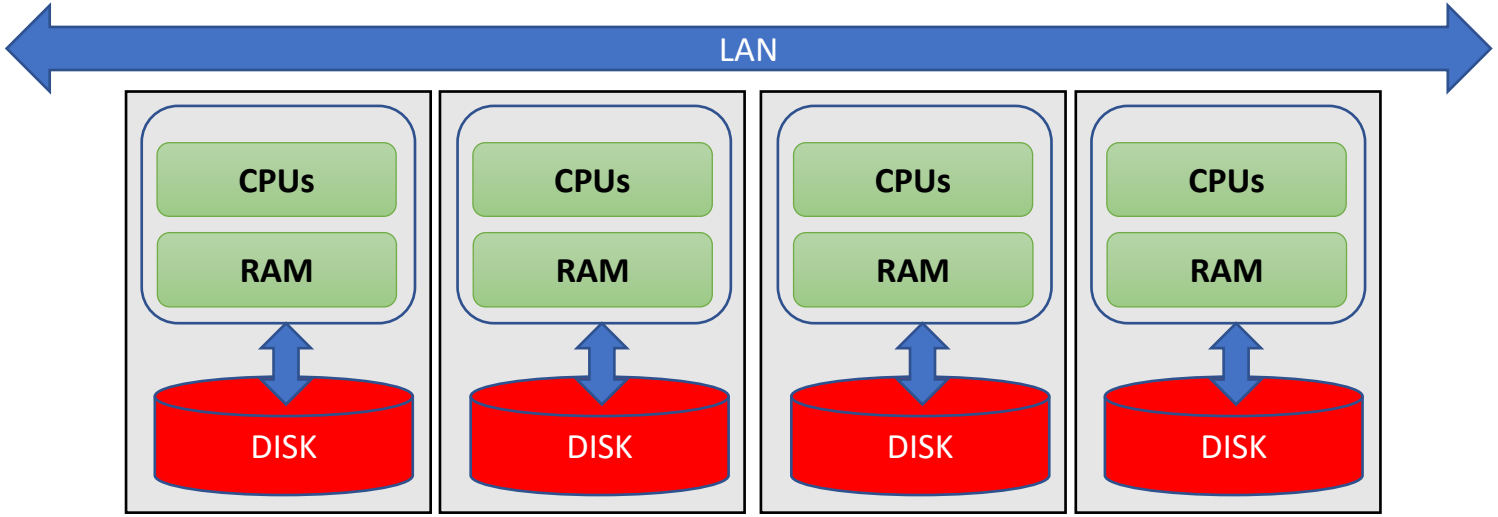
Cons

- More licensing fees
- Bigger footprint in the Data Center
- Higher utility cost (electricity and cooling)
- More networking equipment (switches/routers)

Cluster

A computer cluster is a group of linked computers (nodes), working together closely so that in many respects they form a single computer

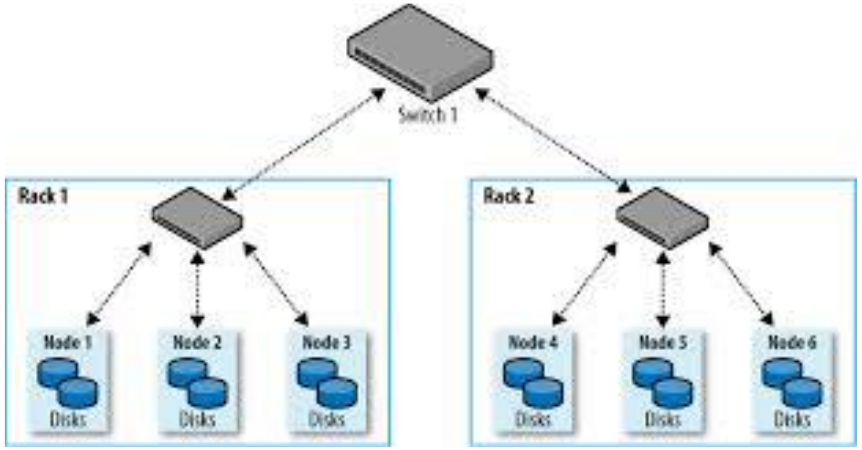
- Typically connected to each other through fast LAN (slower than MPI)
- Still shared-nothing, but every node is a system on its own, capable of independent operations
  - Unlimited scalability, no vendor lock-in
- Number of nodes in the cluster >> Number of CPUs in a node



Cluster

Compute nodes are stored on racks

- 8–64 compute nodes on a rack
- There can be many racks of compute nodes
- The nodes on a single rack are connected by a network (typically gigabit Ethernet)
- Racks are connected by another level of network (or a switch)
  - Intra-rack bandwidth >> inter-rack bandwidth



Commodity hardware

You are not tied to expensive, proprietary offerings from a single vendor

You can choose standardized, commonly available hardware from a large range of vendors to build your cluster

Commodity ≠ Low-end!

- Cheap components with high failure rate can be a false economy

Commodity hardware

Example of commodity hardware specifications:

- Processor: 12-core i7-8700 CPU @ 3.20GHz
- Memory: 64 GB RAM
- Storage: 3 × 4TB SATA disks
- Network: Gigabit Ethernet

Yahoo!’s Hadoop installation:

- > 100,000 CPUs in > 60,000 computers (as of 2017)
- Used to support research for Ad Systems and Web Search
- Also used to do scaling tests to support development of Hadoop

Multiple clusters

Having a single large cluster that is tantalizing to many organizations

- No data silos, simpler governance

Multiple clusters are inevitable within medium-large enterprise settings

- Resiliency
  - Every cluster sits within a single point of failure due to geography
- Software development
  - Mitigate the risk of impacting critical production environments by isolating configuration, integration, or evolution testing and deployment
- Workload isolation
  - Hardware resources tuned for specific workloads, less resource contention
- Legal separation
- Independent Storage and Compute

Multiple clusters

With the success of cloud services, the “independent storage and compute” solution for big data clusters is on the rise

- Network is no more a problem
  - Data locality is (?)
- Scalability is modularized
  - Disk and computation needs are different
- Cost-efficient
  - Data must be persistent, 24/7
  - Compute nodes are activated only on demand

Data storage

Distributed file system

Distributed databases

Distributed FS

Different implementations, relying on similar principles

- Apache Hadoop
- GCP Colossus
- AWS EFS
- Azure Data Lake

HDFS

HDFS is a filesystem designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware

- A typical file in HDFS is gigabytes to terabytes in size. There are Hadoop clusters running today that store petabytes of data
- Applications that run on HDFS need streaming access to their data sets. HDFS is designed more for batch processing rather than interactive use. The emphasis is on high throughput of data access rather than low latency of data access
- HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access
- Hardware failure is the norm rather than the exception. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS

HDFS: blocks

Block: minimum amount of data read/written

- Disk blocks are normally 512B
- Filesystem blocks are typically a few KB

HDFS blocks range between 64MB and 1GB (default: 128MB)

- If a file smaller, it will occupy the necessary disk space, not the full block size

Why blocks?

- Files can be larger than disks. This way, storage management is simplified and replication is easier

Why this big?

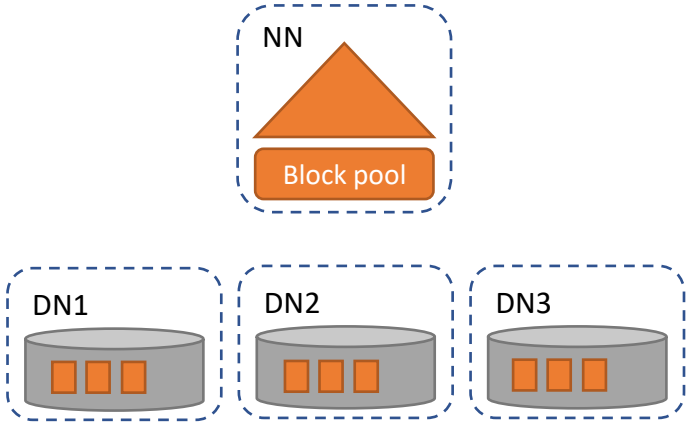
- Large files split into many small blocks require a huge number of seeks
- E.g.: block size = 4KB; file = 1GB; number of seeks = 250.000

HDFS: namenodes and datanodes

Nodes in an HDFS cluster operate in a master-slave pattern

Namenode (NN) - the master

- Persistently maintains the filesystem tree and all files' and directories' metadata
- Keeps in memory the location of each block for a given file (*block pool*)



Datanodes (DNs) - the slave

- Store and retrieve blocks
- Periodically report to the NN with the list of blocks they are storing; *heartbeats* are sent to the DN to signal their active state (every 10 minutes by default)

HDFS: the SPoF

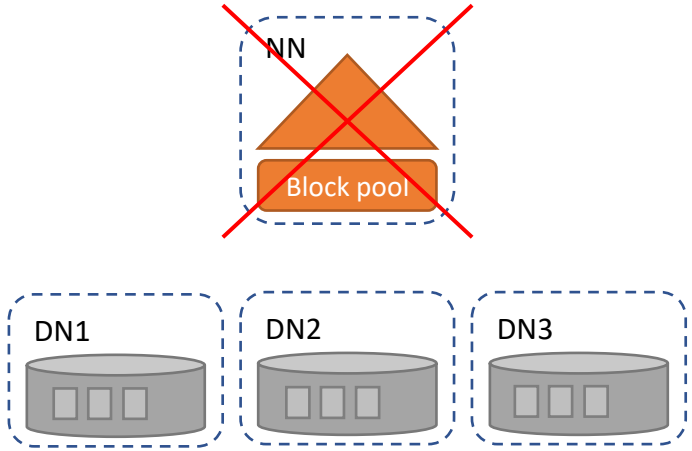
The NN is a single point of failure: without it, the filesystem cannot be used (no way to reconstruct the files from the blocks in the DNs)

Backup solution

- NN writes its persistent state to multiple filesystems, preventing loss of data

Secondary NN solution

- A separate machine regularly connects with the primary NN to build snapshots of its persistent data and saves them to local or remote directories.
- These *checkpoints* can be used to restart a failed NN without having to replay the entire journal of file-system actions.



High-Availability

High-Availability (HA) indicates a system that can tolerate faults

- The service never stops while the fault (be it hardware or software) is detected, reported, masked, and repaired off-line

Backups and secondary NNs protect against data loss..

- ..but restarting a NN could take 30 minutes or more on large clusters!

HA is supported by configuring two separate machines as NNs

- One is in an *active* state, the other is in a *standby* state
- In the event of a failure of the active NN, the standby NN takes over

The NN in standby keeps its metadata up-to-date by:

- Reading the *edit log* files written by the active NN in a shared storage
  - Edit logs are typically stored in *Journal Nodes*, which are also replicated
- Receiving the data block locations and the heartbeats directly from the DNs, which are configured to report to both NNs

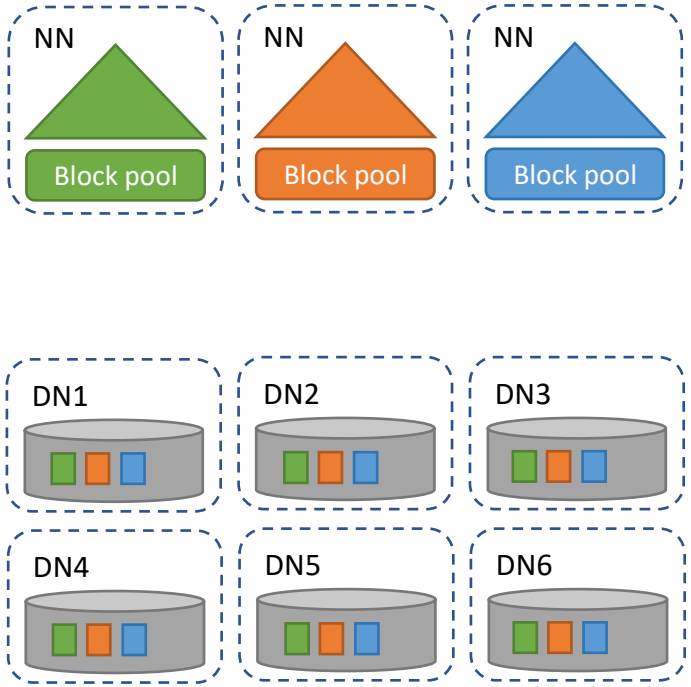
HDFS: federation

The size of the block pool is limited by the memory size of the NN

- May incur in scaling issues on very large clusters with many files

Solution: configure additional NNs

- Each NN manages a portion of the filesystem, i.e., a *namespace*
- NNs are independent of each other
- Introduced with Hadoop 2.0



HDFS: federation

Scalability

- Because the NN keeps all the namespace and block locations in memory, the size of the NN heap limits the number of files and also the number of blocks addressable. This also limits the total cluster storage that can be supported by the NN

Performance

- NN is the single point for storage and management of meta-data, it can become a bottleneck for supporting a huge number of files, especially a large number of small files

Availability

- Separate namespaces of different applications improve the overall availability of the cluster

Maintainability, Security & Flexibility

- Block pool abstraction allows other services to use the block storage with perhaps a different namespace structure. Each namespace is isolated and not aware of the others

Applications can read/write on more than one namespace

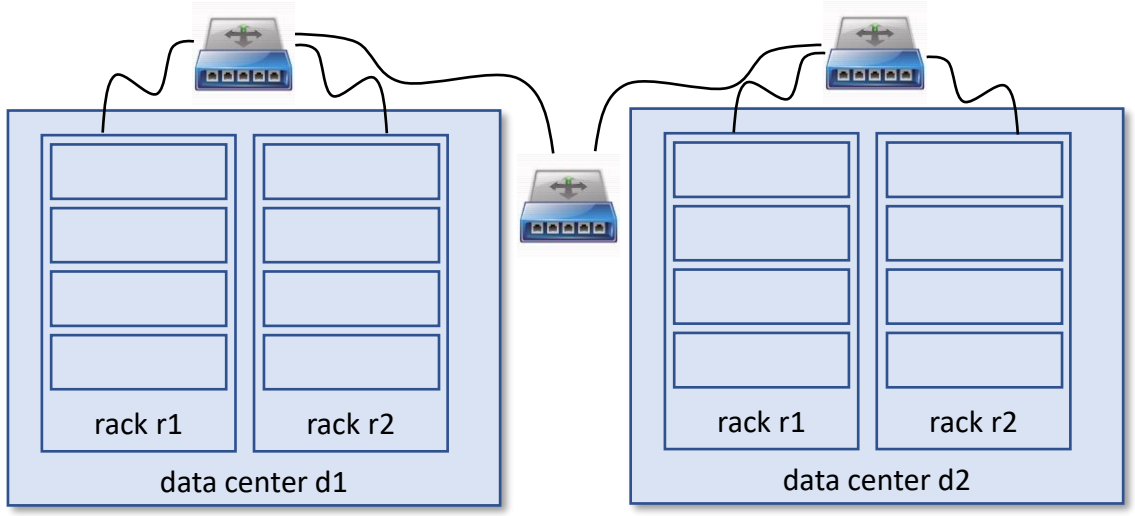
HDFS: cluster topology

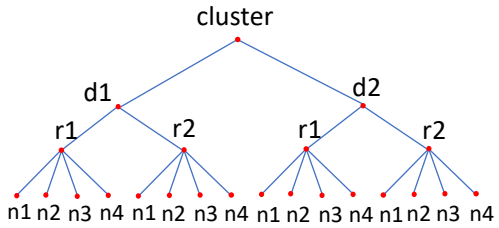
In order to carry out proper choices, Hadoop must be aware of the cluster topology that is defined during the cluster setting phase.

- Block storage and process allocation (data locality) need such information

Nodes are organized in racks, racks are organized in data centers

- Hadoop models such concepts in a tree-like fashion and computes the distance between nodes as their distance on the tree.





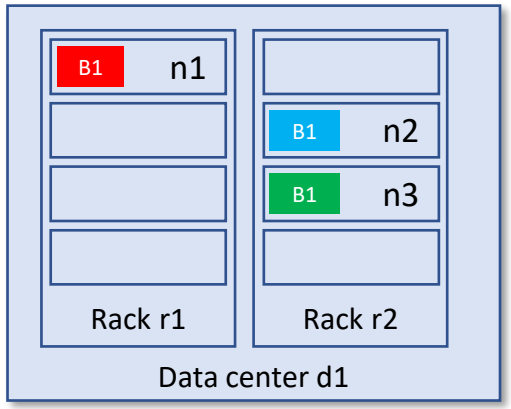
HDFS: replication (1)

Each data block is independently replicated at multiple DNs in order to improve performance and robustness

- Replication is aware of the cluster topology
- For each data block, the NN stores the list of DNs storing it

The default replication factor is 3:

- Replica 1 is stored on the node (n1) where the client issued the write command (if the client resides within the cluster)
- Replica 2 is stored on a node (n2) in a rack (r2) different from the one of n1 (off-rack)
- Replica 3 is stored on a node, different from n2, but that belongs to r2



Replicas can be rebalanced when nodes are added or unavailable

HDFS: replication (2)

Hadoop 3: support to Erasure Coding (EC) as an alternative to simple replication

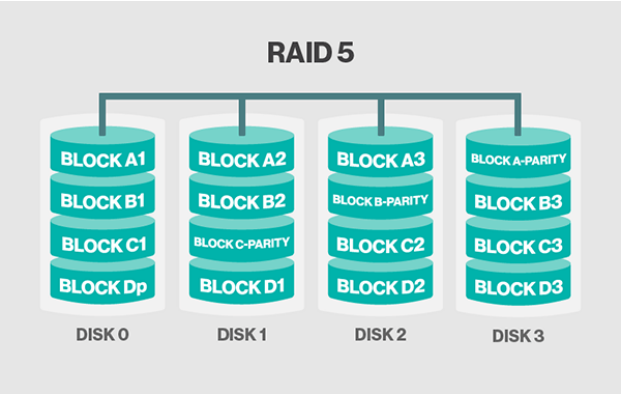
- I.e., a mechanism similar to Redundant Array of Inexpensive Disks (RAID) 5-6
- Each block is split (*striped*) across each data node
- Works best for warm and cold datasets with relatively low I/O activities

Main advantages

- Sensibly reduce data redundancy (from 200% to 50% with default setups)
- Faster writes

Main disadvantages

- Higher CPU cost
- Longer recovery time in case of failure
- **Loss of data locality**



HDFS: not always the best fit

Although this may change in the future, there are areas where HDFS is not a good fit today

Low-latency data access

- HDFS is optimized for delivering a high throughput of data, and this may be at the expense of latency. Applications that require access to data in the tens of milliseconds range will not work well with HDFS.

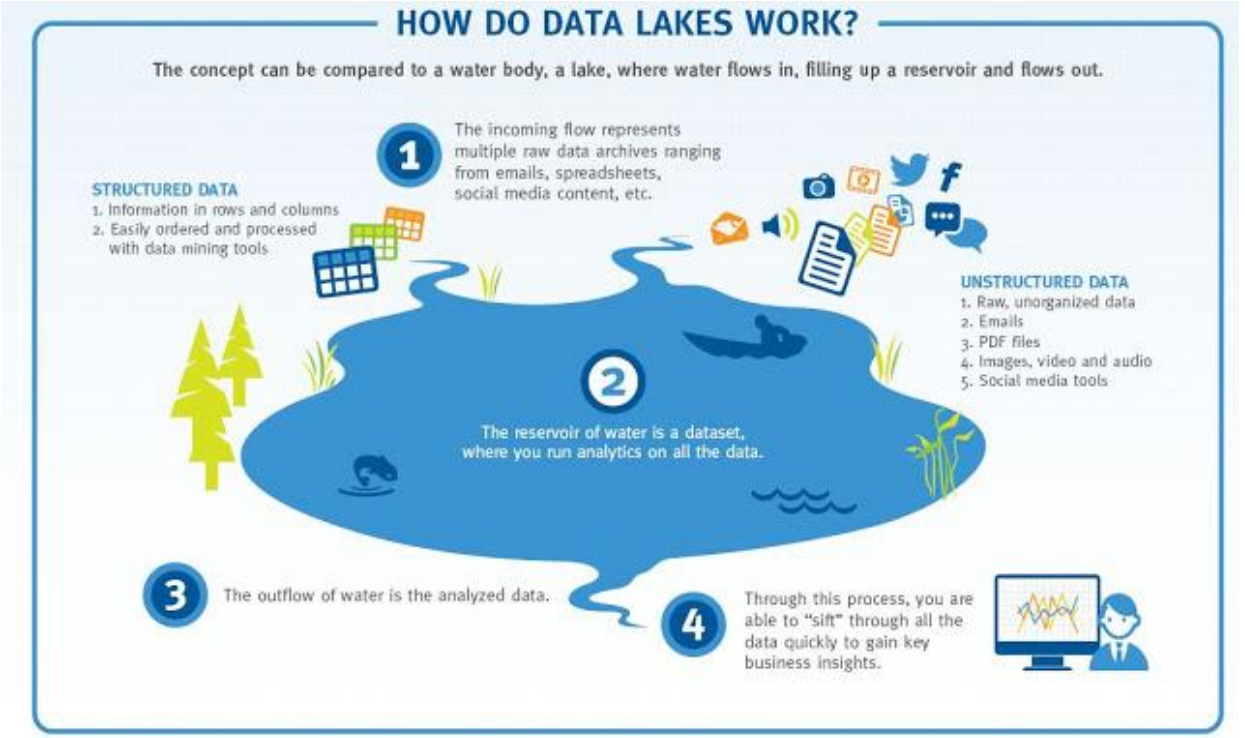
Lots of small files

- The limit to the number of files in a filesystem is governed by the amount of memory on the NN, because it holds filesystem metadata in memory.
- Although storing millions of files is feasible, billions is beyond the capability of current hardware.

The data lake

Big data stored in a DFS usually become a *data lake*

- A central location that holds a large amount of data in its native, raw format
- More on this the day after tomorrow...



File formats

Raw files

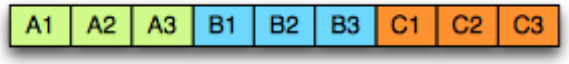
- From simple text to multimedia files
- No support besides reading/writing the whole file (e.g., no record compression, no splittability)

Row-oriented file formats

- Apache Avro, Google's Protocol Buffers, Facebook's Apache Thrift
- Better suited for row-level access to the data (e.g., OLTP)

Column-oriented file formats

- Apache Parquet, ORC
- Better suited for column-level access to the data (e.g., OLAP)
  - Better compression
  - Reduced I/O



Parquet

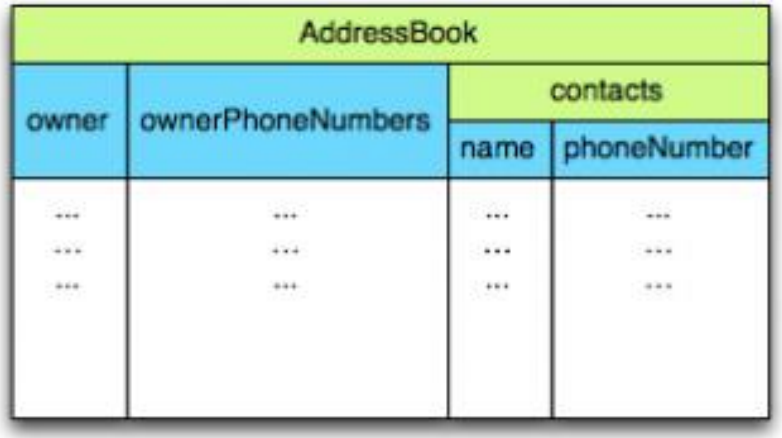
A columnar storage format for efficient querying of **nested structures** in a **flat format**

- Supported by most frameworks and query engines

One column per primitive type

- The structure of the record is captured for each value by two integers: *repetition level* and *definition level*
- Enough to fully reconstruct the nested structures

Column	Type
owner	string
ownerPhoneNumbers	string
contacts.name	string
contacts.phoneNumber	string



Parquet: definition level

Needed in presence of optional nested fields

- If there is a *null* value, at which level is it?

```
message ExampleDefLevel {
  optional group a {
    optional group b {
      optional string c;
    }
  }
}
```

Value	Definition Level
a: null	0
a: { b: null }	1
a: { b: { c: null } }	2
a: { b: { c: "foo" } }	3 (actually defined)

Parquet: repetition level

Needed in presence of repeated fields (i.e., arrays)

- To which array/record does the value belong to?

Schema:

```
message nestedLists {
   repeated group level1 {
      repeated string level2;
   }
}
```

Data: `[[a,b,c],[d,e,f,g]],[[h],[i,j]]`

```
{
      level1: {
            level2: a
            level2: b
            level2: c
      },
      level1: {
            level2: d
            level2: e
            level2: f
            level2: g
      }
}
{
      level1: {
            level2: h
      },
      level1: {
            level2: i
            level2: j
      }
}
```

The repetition level can be seen as a marker of when to start a new list and at which level.

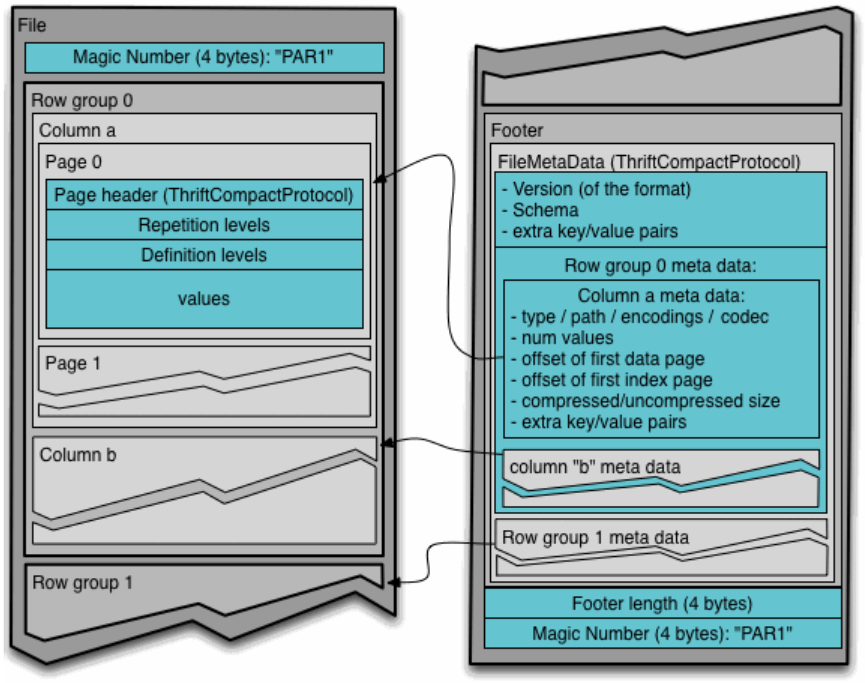
Repetition level	Value
0	a
2	b
2	c
1	d
2	e
2	f
2	g
0	h
1	i
2	j

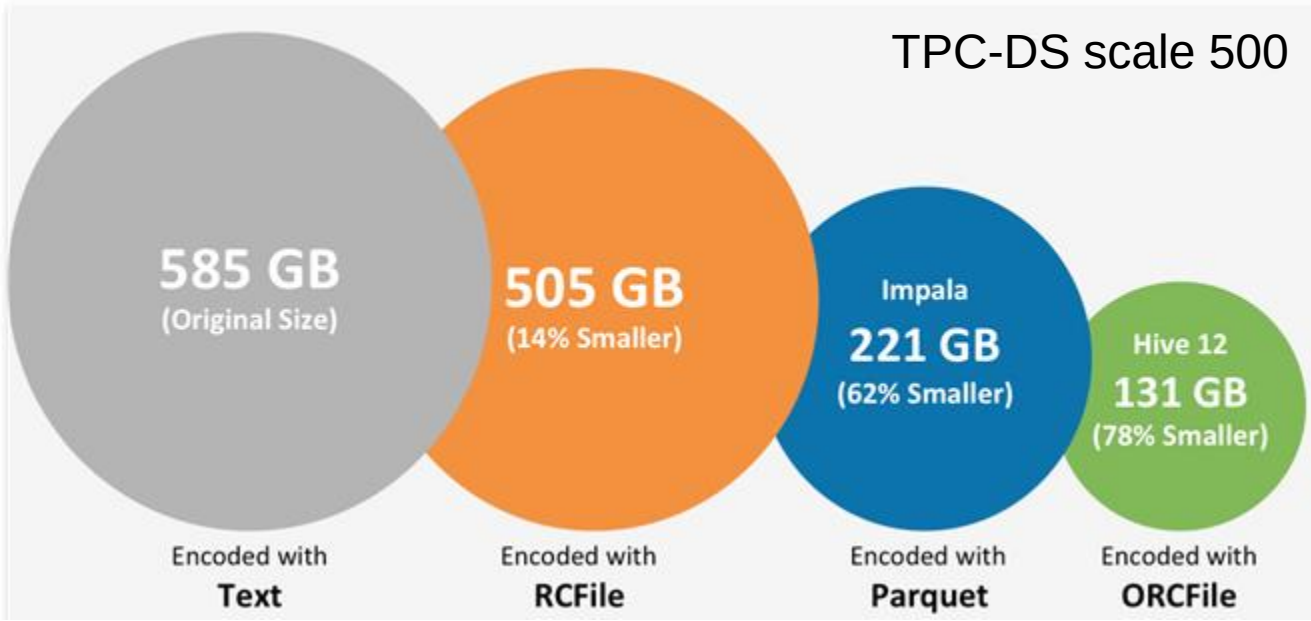
0 marks every new record;
implies creating a new level1 and level2 list

1 marks every new level1 list;
implies creating a new level2 list as well

2 marks every new element in a level2 list

Parquet: storage and compression





Distributed DB

New types of databases have emerged

RDBMSs are full of strengths

ACID properties

- Provides guarantees in terms of consistency and concurrent accesses

Data integration and normalization of schemas

- Several application can share and reuse the same information

Standard model and query language

- The relational model and SQL are very well-known standards
- The same theoretical background is shared by the different implementations

Robustness

- Have been used for over 40 years

RDBMSs have weaknesses as well

Impedance mismatch

- Data are stored according to the relational model, but applications to modify them typically rely on the object-oriented model
- Many solutions, no standard
  - E.g.: Object Oriented DBMS (OODBMS), Object-Relational DBMS (ORDBMS), Object-Relational Mapping (ORM) frameworks

Painful scaling-out

- Not suited for a cluster architecture
- Distributing an RDBMS is neither easy nor cheap (e.g., Oracle RAC)

Consistency vs latency

- Consistency is a must – even at the expense of latency
- Today's applications require high reading/writing throughput with low latency

Schema rigidity

- Schema evolution is often expensive

What is "NoSQL"

The term has been first used in '98 by Carlo Strozzi

- It referred to an open-source RDBMS that used a query language different from SQL

In 2009 it was adopted by a meetup in San Francisco

- Goal: discuss open-source projects related to the newest databases from Google and Amazon
- Participants: Voldemort, Cassandra, Dynomite, HBase, Hypertable, CouchDB, MongoDB

Today, NoSQL indicates DBMSs adopting a different data model from the relational one

- NoSQL = Not Only SQL
- According to Strozzi himself, NoREL would have been a more proper noun

The first NoSQL systems

LiveJournal, 2003

- Goal: reduce the number of queries on a DB from a pool of web servers
- Solution: Memcached, designed to keep queries and results in RAM

Google, 2005

- Goal: handle Big Data (web indexing, Maps, Gmail, etc.)
- Solution: BigTable, designed for scalability and high performance on Petabytes of data

Amazon, 2007

- Goal: ensure availability and reliability of its e-commerce service 24/7
- Solution: DynamoDB, characterized by strong simplicity for data storage and manipulation

NoSQL common features

Not just rows and tables

- Several data model adopted to store and manipulate data

Freedom from joins

- Joins are either not supported or discouraged

Freedom from rigid schemas

- Data can be stored or queried without pre-defining a schema (*schemaless* or *soft-schema*)

Distributed, shared-nothing architecture

- Trivial scalability in a distributed environment with no performance decay
- Each workstation uses its own disks and RAM

Not a farewell to SQL

NoSQL data models

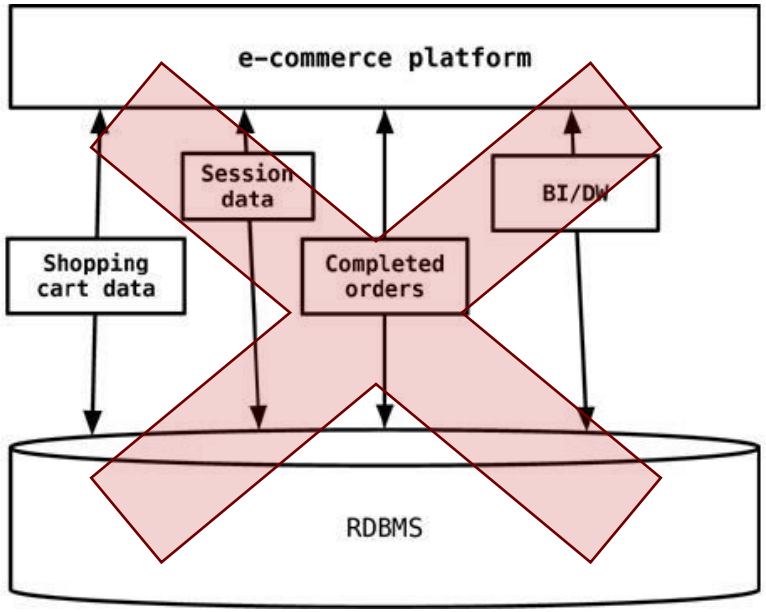
NoSQL databases mainly differ by the supported data model

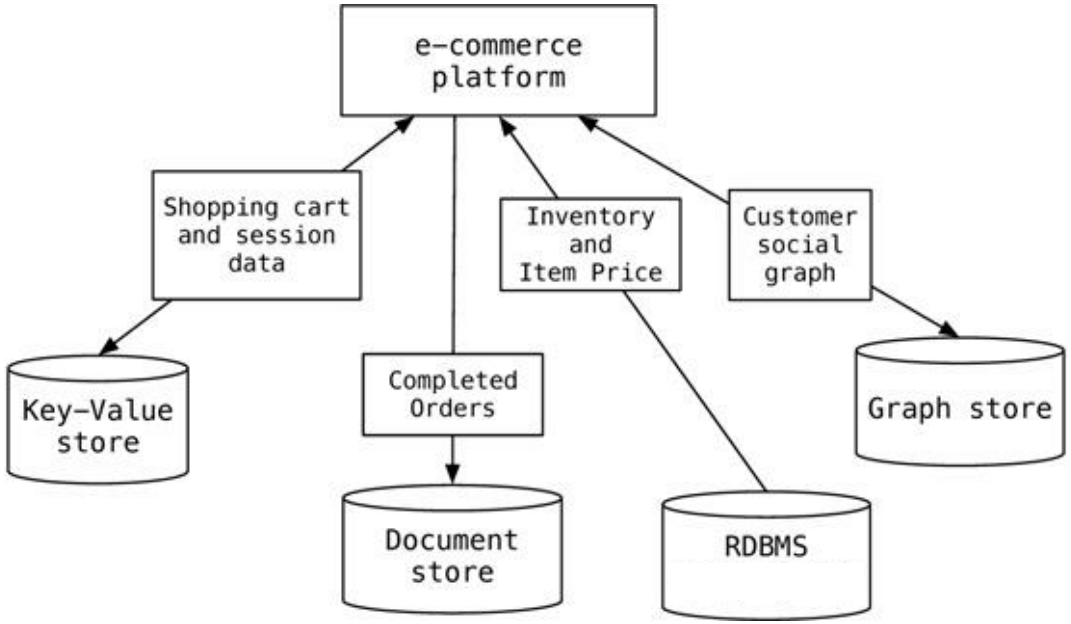
Model	Description	Use cases
Key-value	Associates any kind of value to a string	Dictionary, lookup table, cache, file and images storage
Document-based	Stores hierarchical data in a tree-like structure	Documents, anything that fits into a hierarchical structure
Wide-column	Stores sparse matrixes where a cell is identified by the row and column keys	Crawling, high-variability systems, sparse matrixes
Graph	Stores vertices and arches	Social network queries, inference, pattern matching

Polyglot persistence

The *one-size-fits-all* is “no more”

- More on this tomorrow...





Data processing

Three main kinds of data processing

- **Batch**
- Interactive
- Streaming

Batch

Simple/complex computations over large amounts of stored data

- Execution takes minutes to hours

What we need

- A cluster manager to negotiate resources
- An execution engine

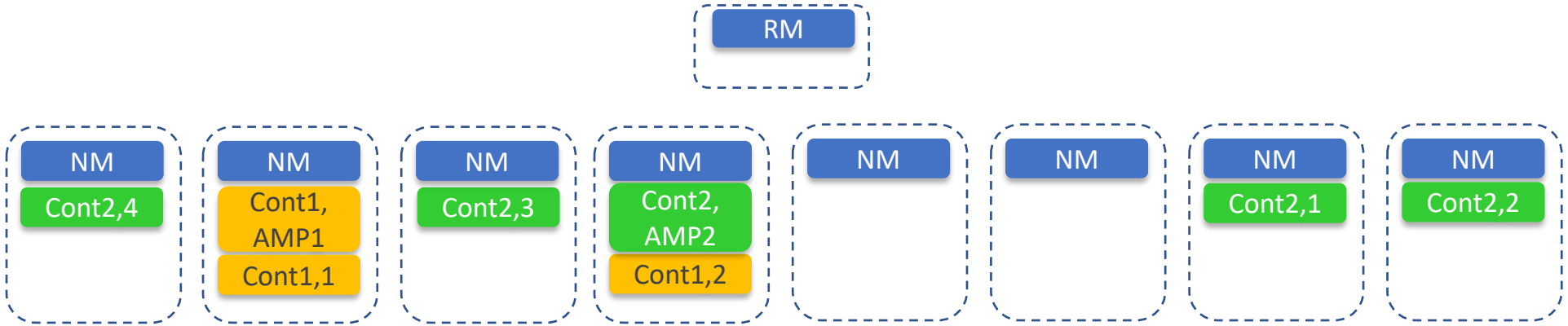
YARN

Yet Another Resource Negotiator

- Usage: assign resources for applications to run their processes

Two daemons

- Resource Manager (RM) - the master
  - The ultimate authority that arbitrates resources among all the applications
- Node Managers (NM) - the slaves
  - Responsible for the allocation and monitoring of containers



Data locality

Exploit cluster topology and data block replication to apply the data locality principle

When computations involves large set of data,
it is cheaper (i.e. faster) to move code to data
rather than data to code

The following cases respect the order the resource manager prefers:

1. Process and data on the same node
2. Process and data on the different node of the same rack
3. Process and data on different racks of the same data center
4. Process and data on different racks of the different data centers

YARN: Scheduler

YARN provides a choice of schedulers and configurable policies

FIFO Scheduler

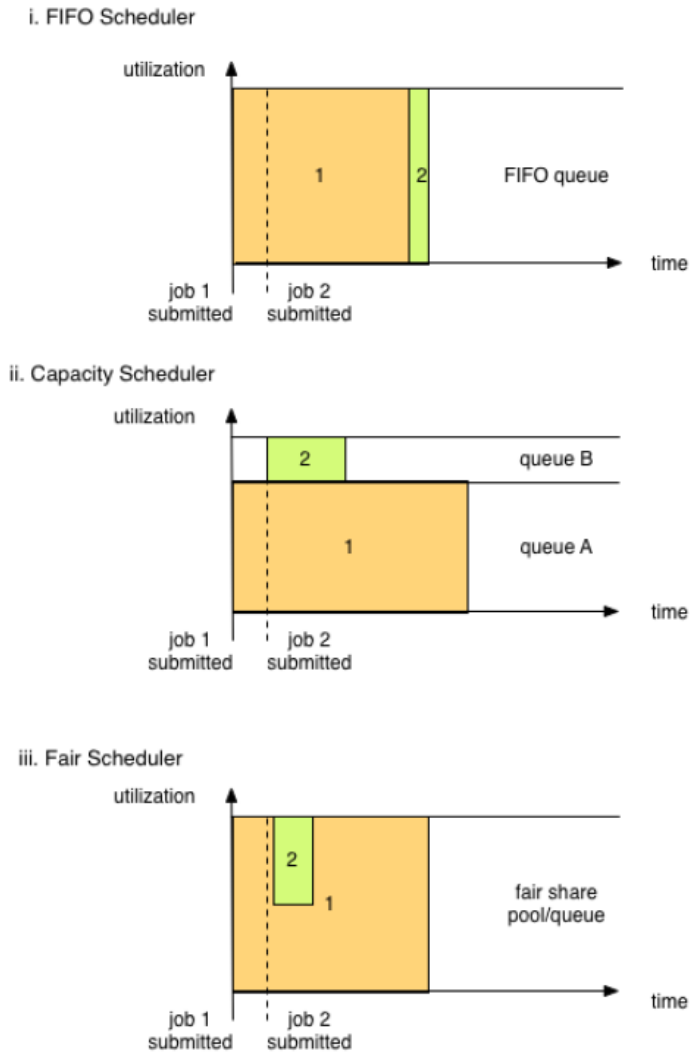
- Simple to understand, no configuration needed
- Not suitable for shared clusters

Capacity Scheduler (default)

- Reserves a fixed amount of capacity to each job

Fair Scheduler

- Dynamically balances the available resources between all running jobs



Typical Large-Data Problem

Iterate over a large number of records

Extract something of interest from each — Map

Shuffle and sort intermediate results

Aggregate intermediate results — Reduce

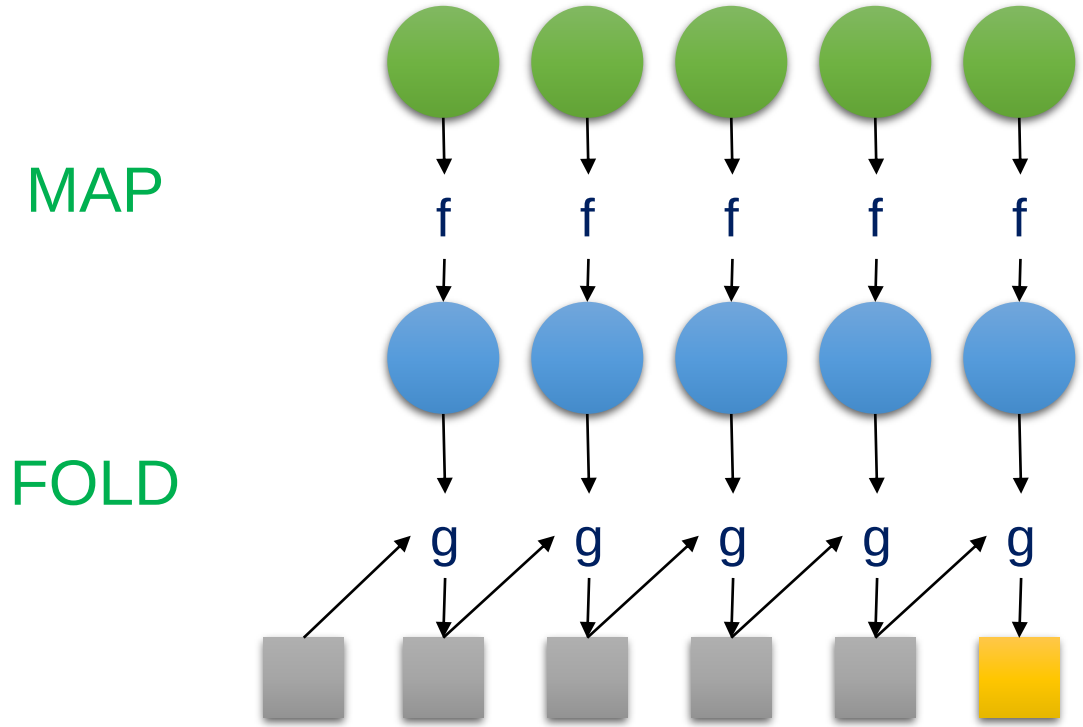
Generate final output

Key idea: provide a functional abstraction for these two operations

Roots in Functional Programming

MAP takes a function f and applies it to every element in a list,

FOLD iteratively applies a function g to aggregate results



Parallelization of Map and Reduce

The map operation (i.e., the application of *f* to each item in a list) can be parallelized in a straightforward manner, since each functional application happens in isolation

- In a cluster, these operations can be distributed across many different machines

The reduce operation has more restrictions on data locality

- Elements in the list must be "brought together" before the function *g* can be applied

However, many real-world applications do not require *g* to be applied to all elements of the list. If elements in the list can be divided into groups, the fold aggregations can proceed in parallel.

Disambiguation of MapReduce

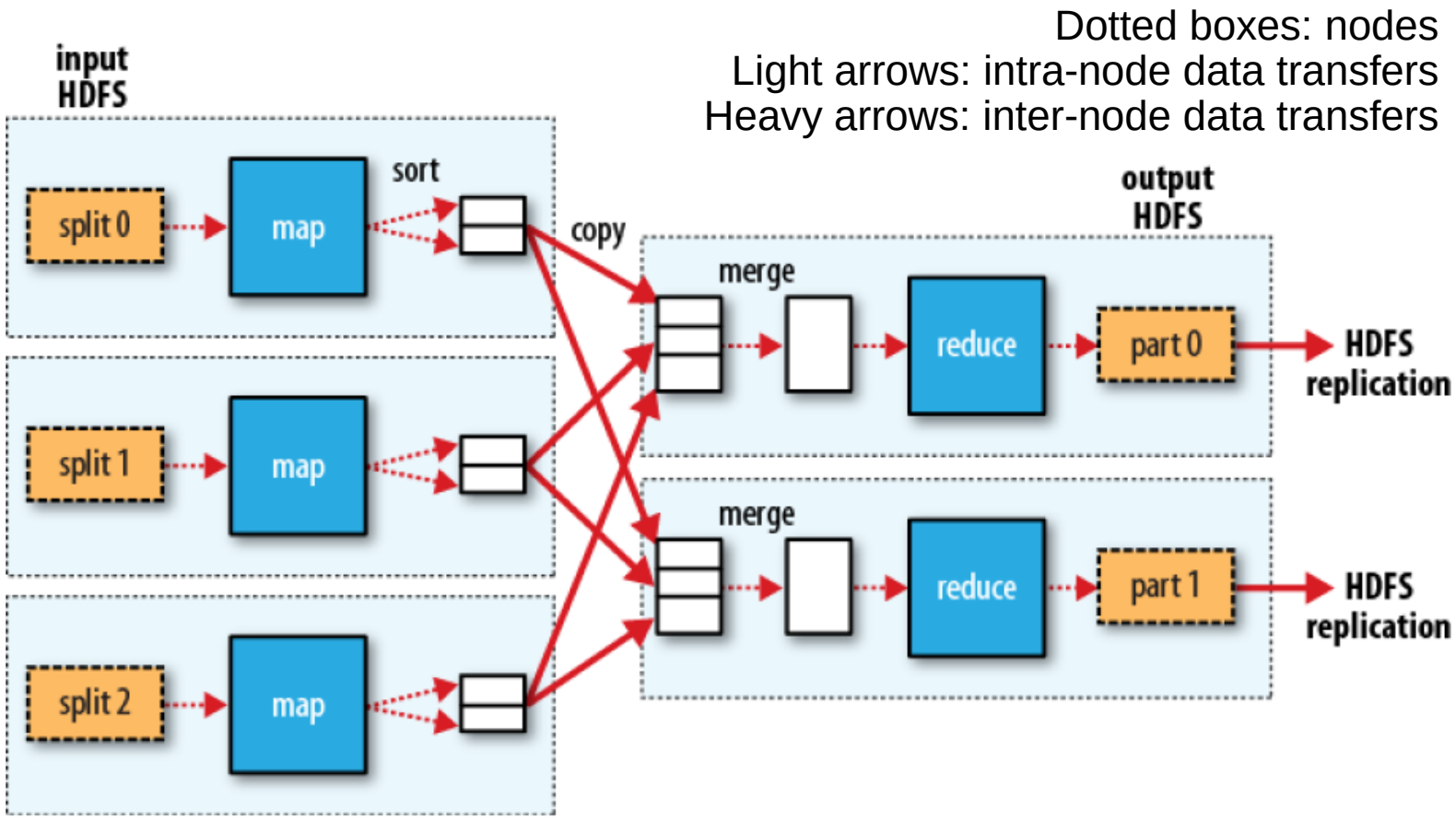
"MapReduce is a programming model and an associated implementation for processing and generating large data sets.

Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key."

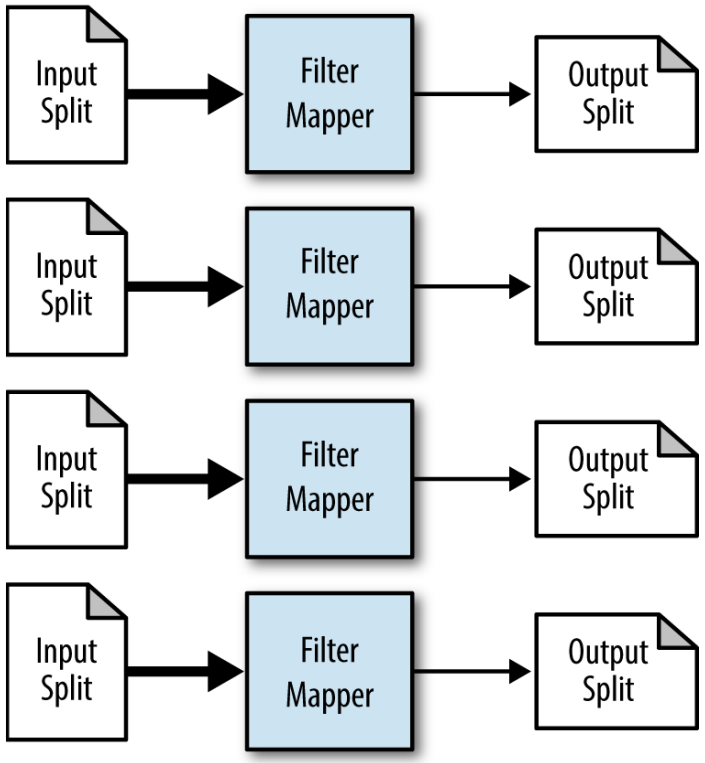
-- Dean J., Ghemawat S. (Google)

Hadoop MapReduce is an open-source implementation of the MapReduce programming model

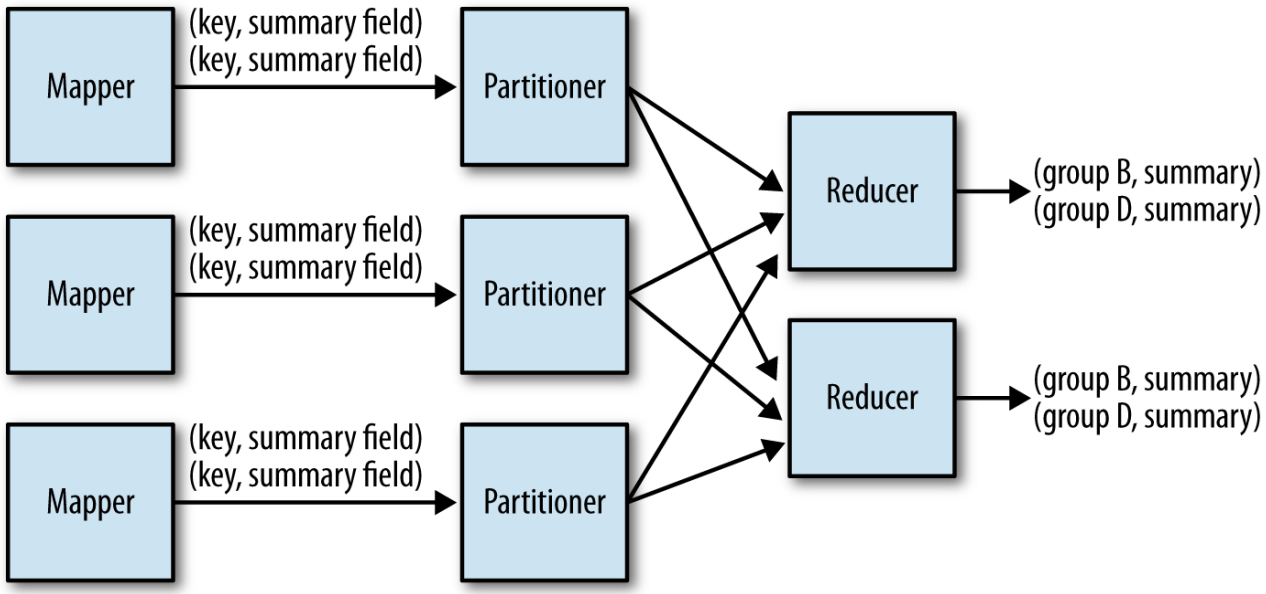
MapReduce process



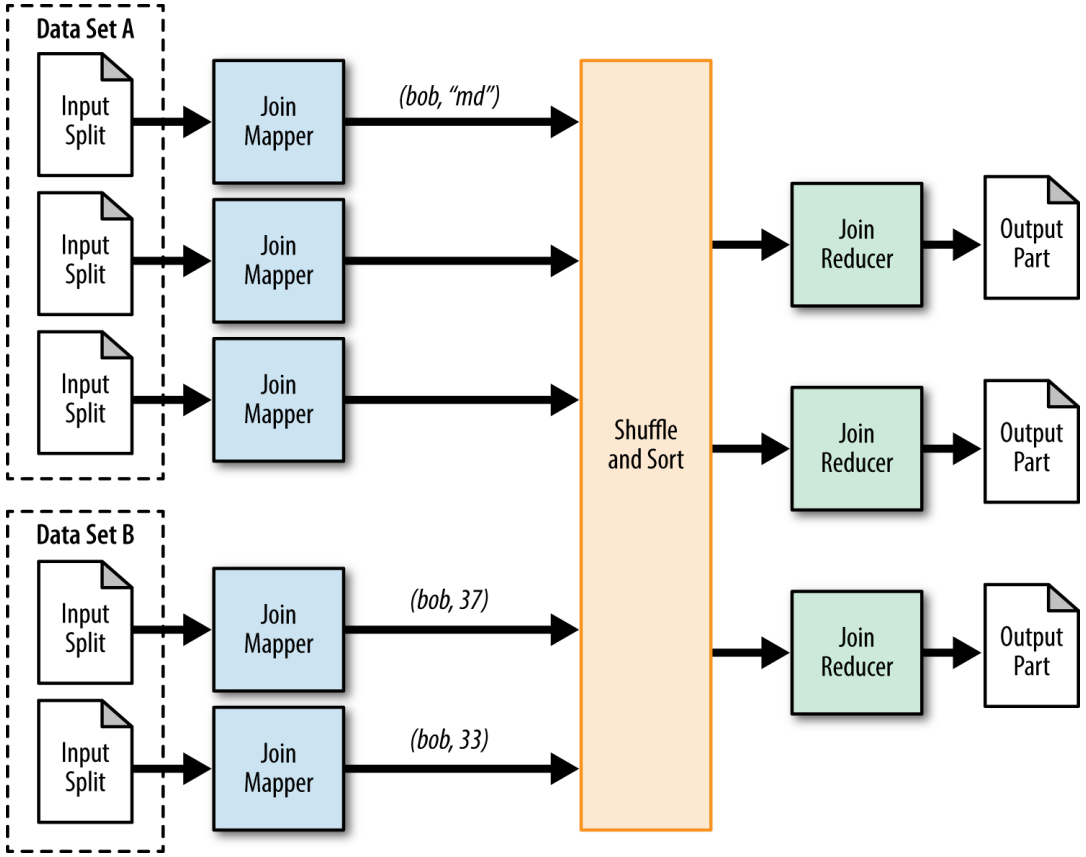
Filtering pattern



Summarization pattern



Join pattern



Sort

Goal: sort input

Examples:

- Return all the domains indexed by Google and the number of pages in each, ordered by the number of pages

The programming model (map, then reduce) does not support this per se

- But the implementations do: the shuffle stage performs grouping and ordering!

General pattern:

- map (key, record) → emit (sortKey, record)
- reduce (sortKey, records) → emit (sortKey, records[1]), ...

The Map and the Reduce do nothing

- With 1 reducer, we get sorted output
- With many reducers, we get partly sorted output (unless: TotalOrderPartitioner)

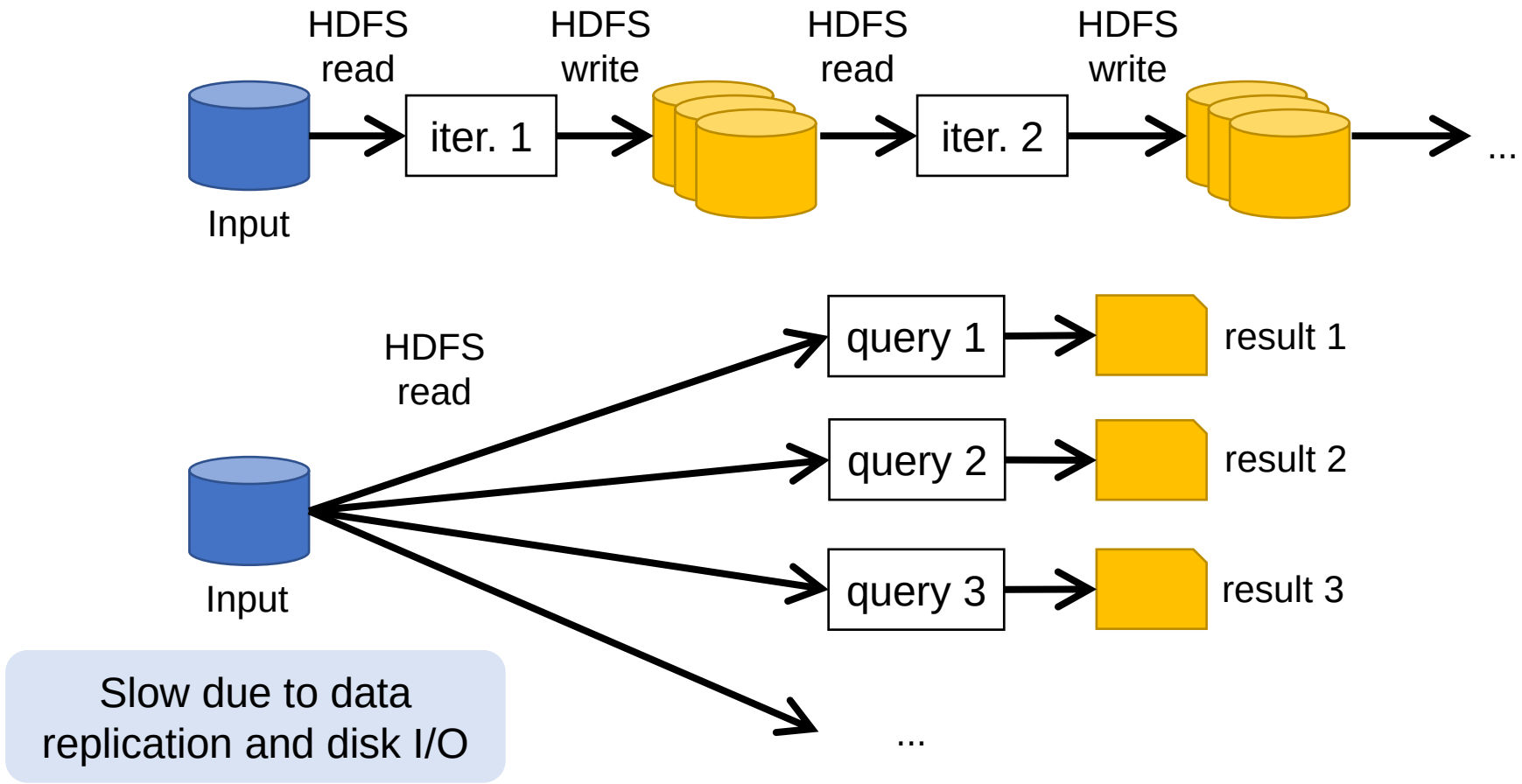
Two stage MapReduce

As map-reduce calculations get more complex, it’s useful to break them down into stages

- The output of the first stage serves as input to the next one
- The same output may be useful for different subsequent stages
- The output can be stored in the DFS, forming a materialized view

Early stages of map-reduce operations often represent the heaviest amount of data access, so building and saving them once as a basis for many downstream uses saves a lot of work!

Data sharing in MapReduce



Spark

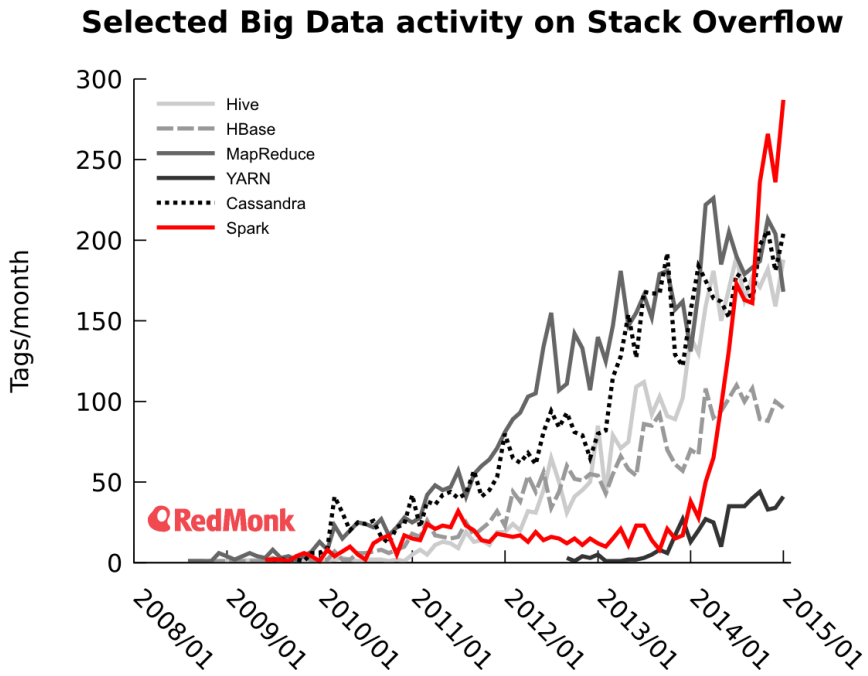
Spark project started in 2009

- Developed at UC Berkeley's AMPLab by Matei Zaharia
- Originally built to test how Apache Mesos works

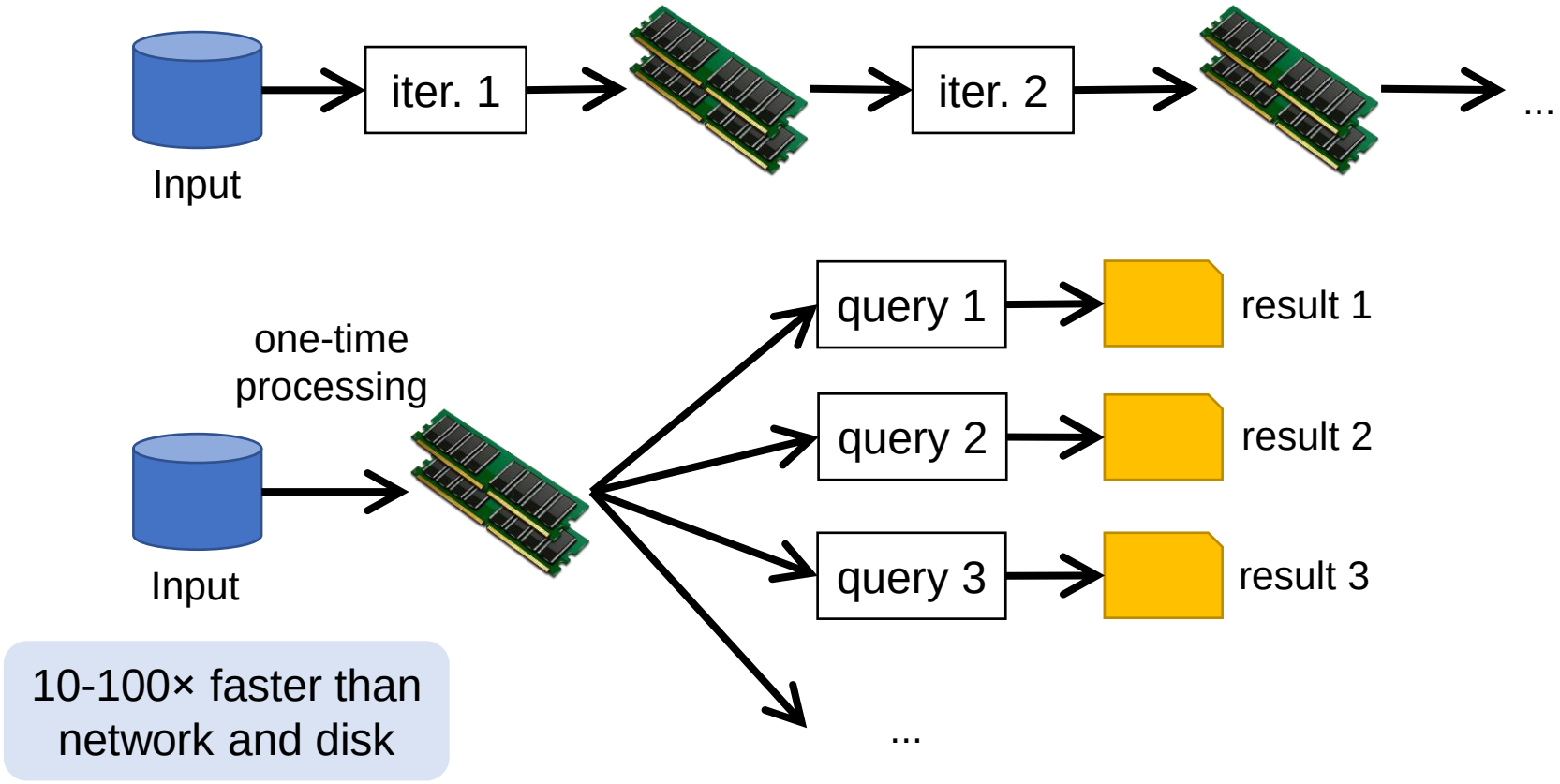
Open sourced 2010, Apache project from 2013

- In 2014, Zaharia founded Databricks
- Current version: 3.2
- Written in Scala; supports Java and Python

Currently the most used tool for batch analyses



Data sharing in Spark



Spark pillars: RDD

RDDs are immutable distributed collection of objects

- **Resilient**: automatically rebuild on failure
- **Distributed**: the objects belonging to a given collection are split into *partitions* and spread across the nodes
  - RDDs can contain any type of Python, Java, or Scala objects
  - Distribution allows for scalability and locality-aware scheduling
  - Partitioning allows to control parallel processing

Fundamental characteristics (mostly from *pure functional programming*)

- **Immutable**: once created, it can't be modified
- **Lazily evaluated**: optimization before execution
- **Cacheable**: can persist in memory, spill to disk if necessary
- **Type inference**: data types are not declared but inferred (≠ dynamic typing)

Spark pillars: DAG

Based on the user application and on the lineage graphs,
Spark computes a logical execution plan in the form of a DAG

- Which is later transformed into a physical execution plan

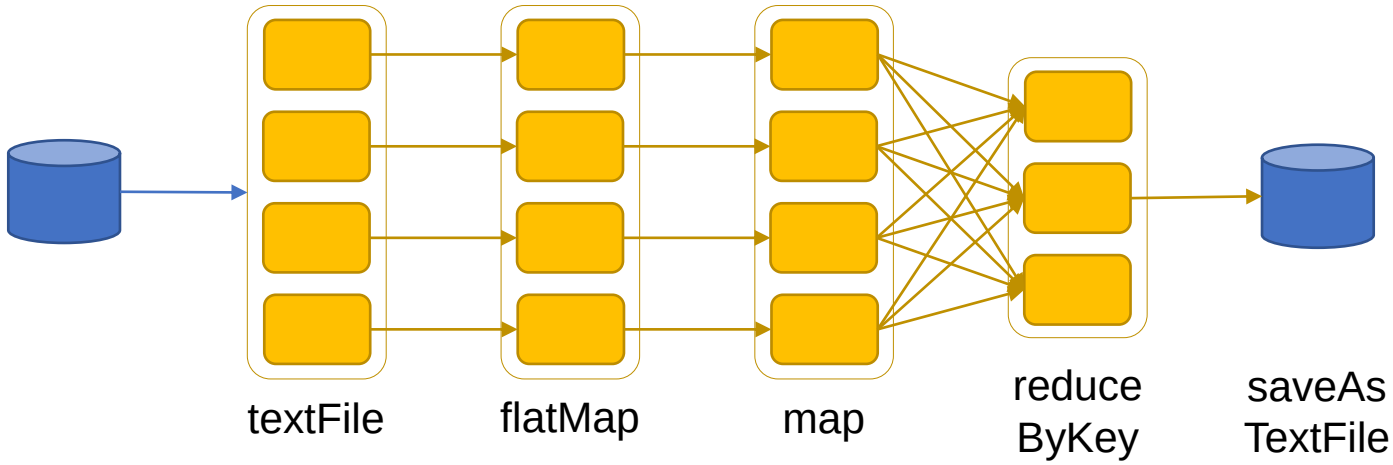
The DAG (Directed Acyclic Graph) is a sequence of computations performed on data

- Nodes are **RDDs**
- Edges are operations on RDDs
- The graph is Directed: transformations from a partition A to a partition B
- The graph is Acyclic: transformations cannot return an old partition

Spark sample DAG

Word count in Scala

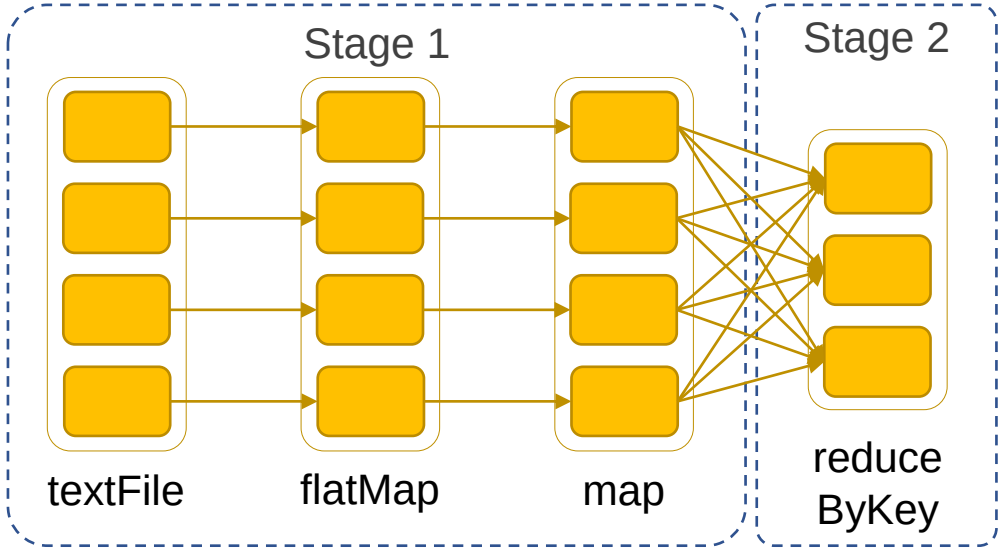
- textFile = sc.textFile("hdfs://...")
- counts = textFile.flatMap(line => line.split(" "))
  .map(lambda word: (word, 1))
  .reduceByKey(lambda a,b: a + b)
- **counts.saveAsTextFile("hdfs://...")**



DAG decomposed into stages

The execution plan is compiled into physical **stages**

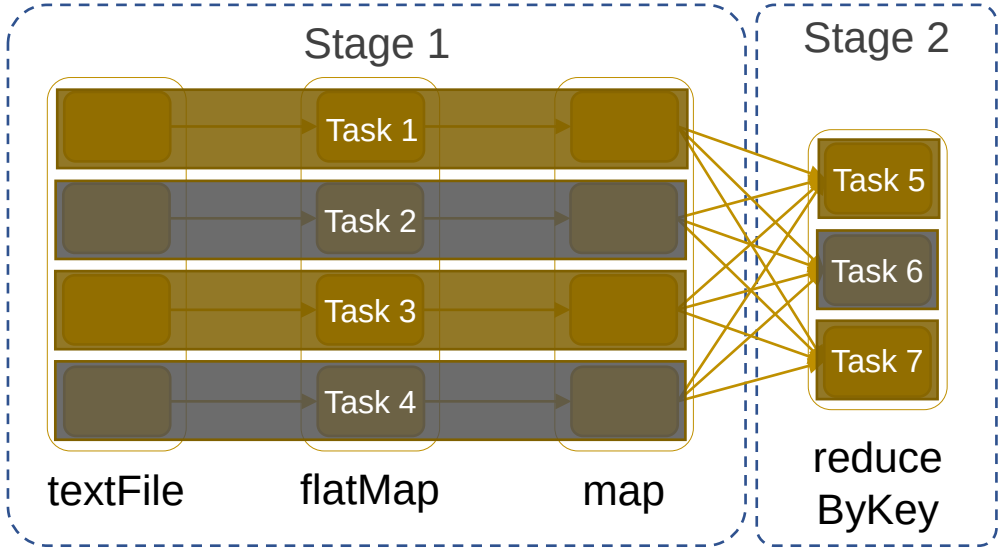
- Stages' boundaries are defined by shuffle operations
- Operations with narrow dependencies are pipelined as much as possible



Stages decomposed into tasks

The fundamental unit of execution is the one of **tasks**

- A task is created for each partition in the new RDD
- Tasks are scheduled and assigned to the worker nodes based on data locality
- The scheduler can run the same task on multiple nodes in case of stragglers (i.e., slow nodes)

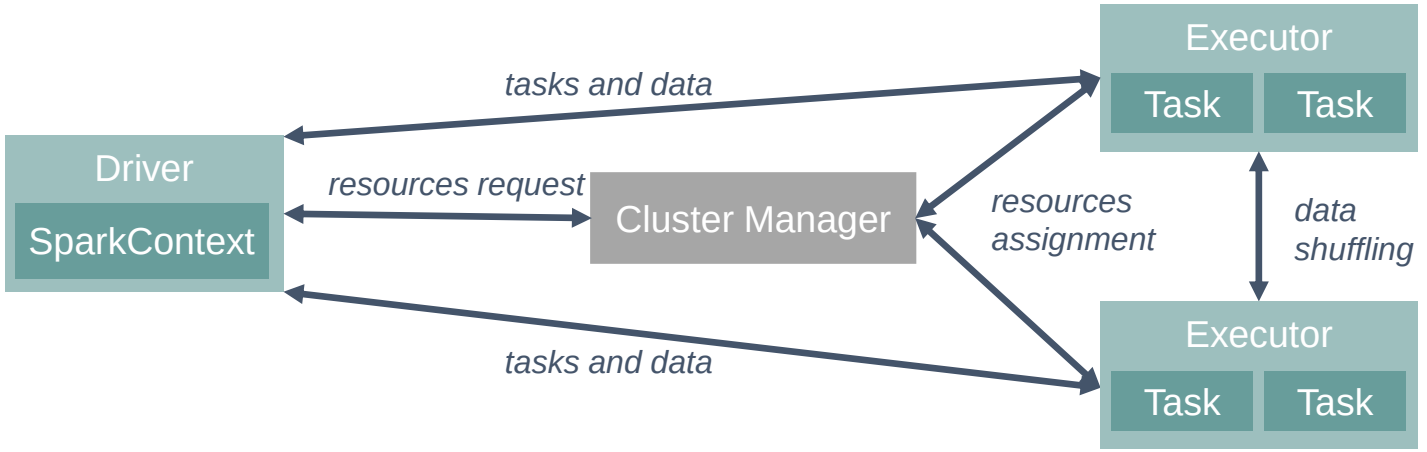


Spark architecture

Spark uses a *master/slave architecture* with one central coordinator (*driver*) and many distributed workers (*executors*)

- The driver and each executor are independent Java processes
- Together they form a Spark *application*

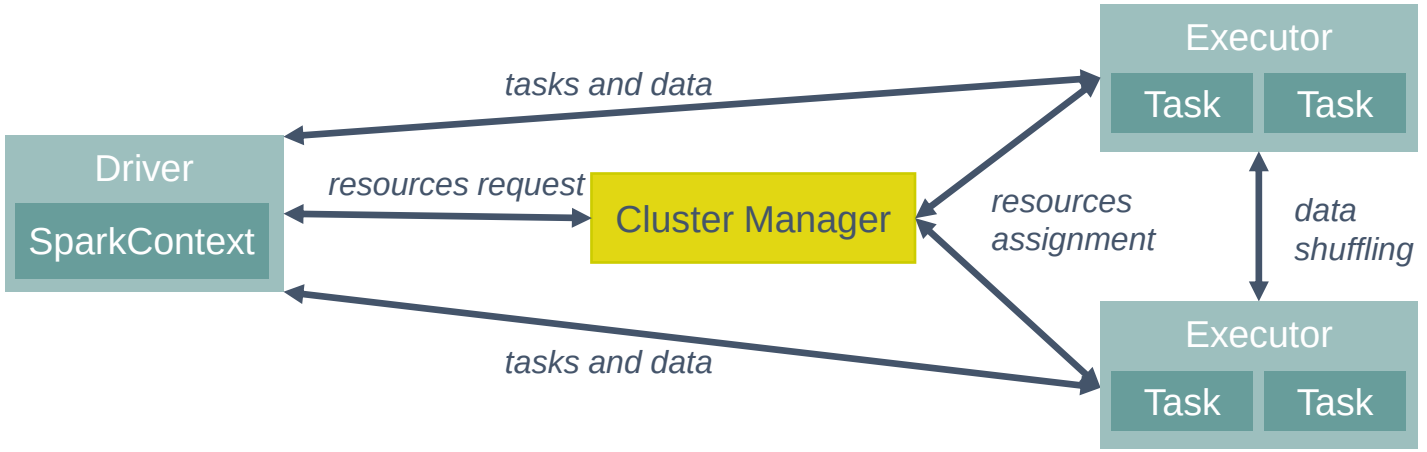
The architecture is independent of the cluster manager that Spark runs on



Spark architecture

Cluster Manager: the component responsible for assigning and managing the cluster's resources (e.g., memory, processor time, …)

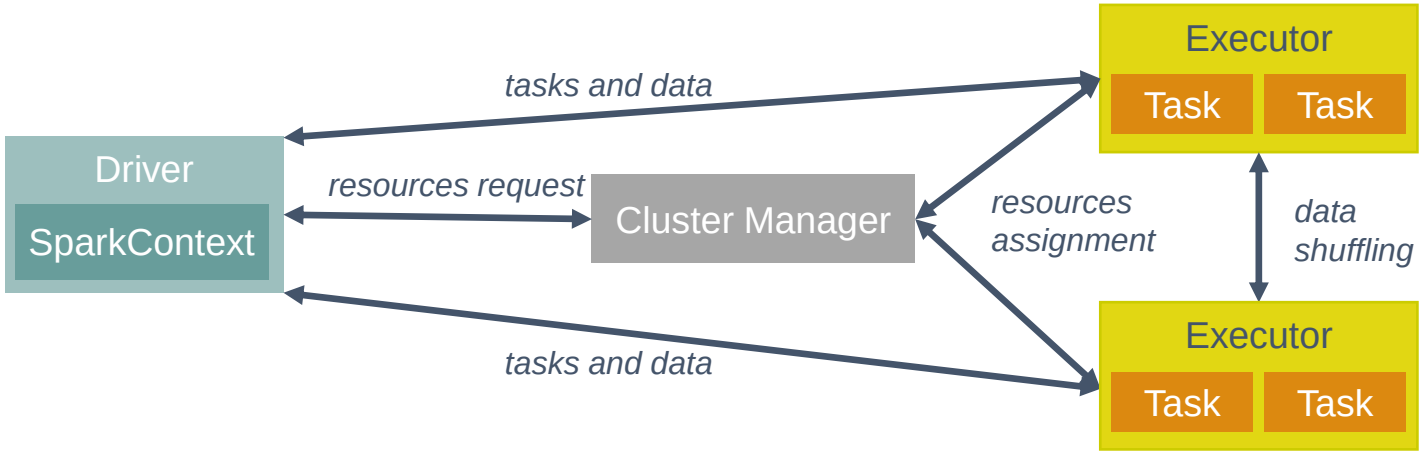
- The cluster manager can be either the Spark standalone manager or any other compatible manager (e.g., YARN)
- Launches executor processes on behalf of the driver



Spark architecture

Executor: a process responsible for executing the received tasks

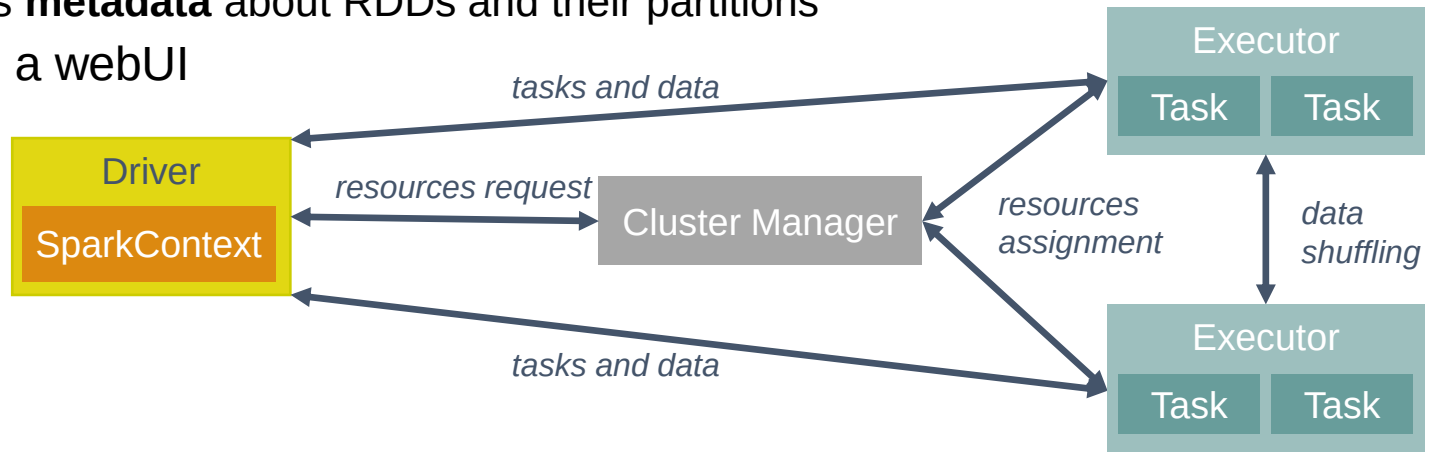
- Each spark application can have (and usually has) multiple executors, and each worker node can host many executors
- Typically runs for the entire duration of the application
- Stores (caches) RDD data in JVM heap
- Tasks are the smallest unit of work and are carried out by executors



Spark architecture

Driver Program (a.k.a. *Spark Driver*, or simply *Driver)*

- Each spark application can only have one driver (entry point of Spark Shell)
- Converts user program into tasks
  - Creates the **SparkContext**, i.e., the object that handles communications
  - Computes the logical **DAG** of operations and converts it into a physical **execution plan**
- Schedules tasks on executors
  - Has a **complete view** of the available executors and schedules tasks on them
  - Stores **metadata** about RDDs and their partitions
- Launches a webUI



Spark architecture (in YARN)

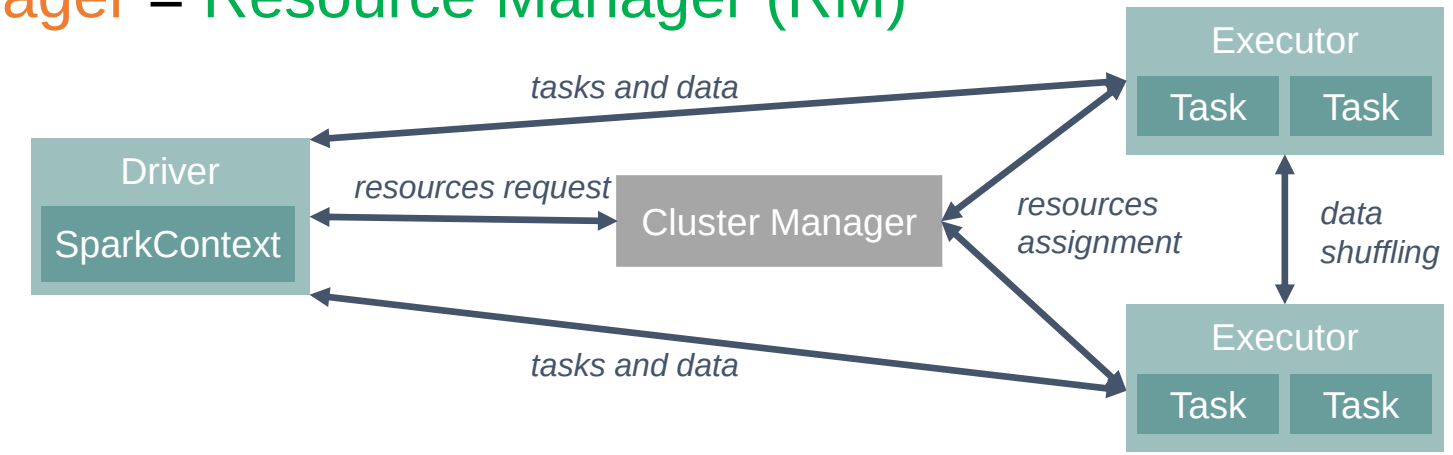
Driver Program ≅ Application

- Driver can be run *externally* from the client (e.g., spark-shell) or *internally* from the AMP (e.g., for production jobs)
- The Application master process (AMP) is not shown for simplicity

Executor = Container

- Executors are run and monitored by the Node Manager (NM)

Cluster Manager = Resource Manager (RM)



Interactive

Simple computations over small/large amounts of stored data

- Execution takes milliseconds to minutes

What we need

- An MPP-like execution engine

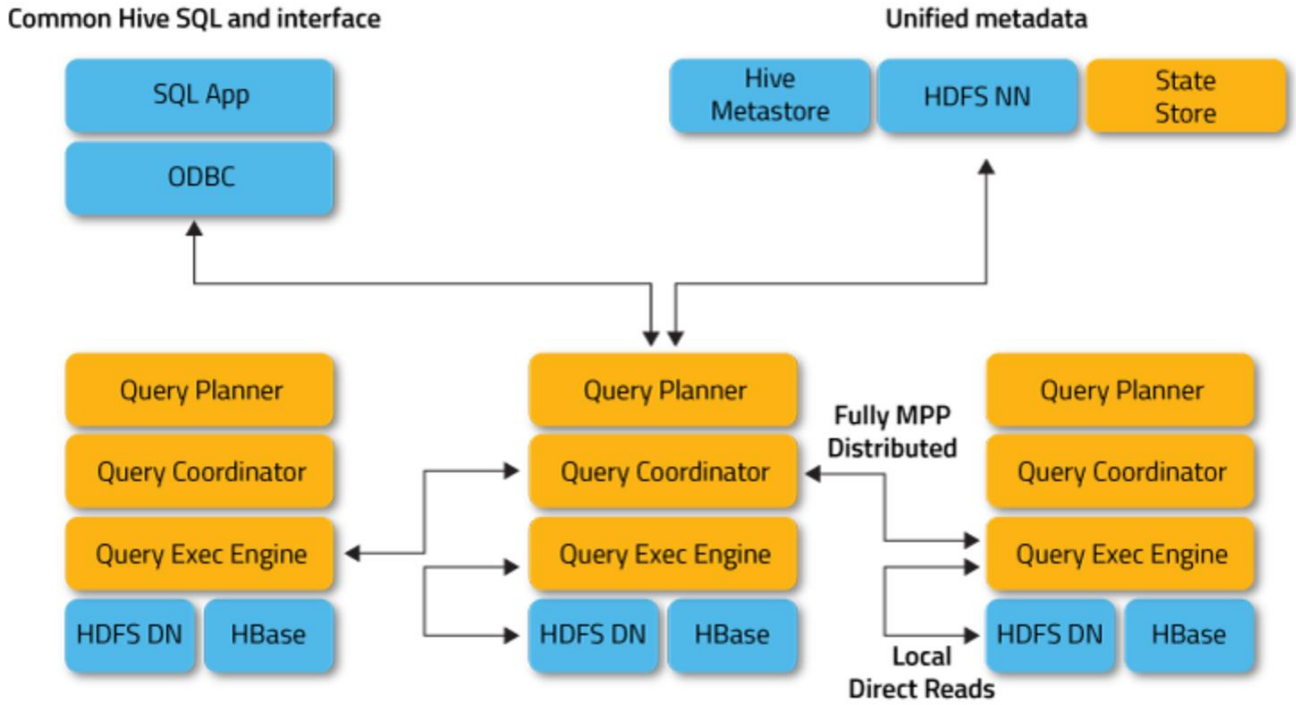
Interactive

MPP-like architecture

- Always-running daemons
- Distributed engine: submit queries to any node
- Highly rely on caching and in-memory computation
- Optimistic query execution

Several tools

- Apache Impala (from Cloudera)
- Apache Presto (from Facebook)
- Apache Drill



Streaming

Simple computations over small amounts of continuously incoming data

- Execution is in near-real-time

What we need

- A message queueing service
- An execution engine

Streaming system definition

The term "streaming" has been used to mean a variety of different things

- Video streaming
- Streaming systems
- Streaming algorithms

In our context, a *system for data streaming* is

***a type of* data processing engine**
that is designed with* infinite datasets *in mind

Remember that

- Batch engines can be (and have been) used to process infinite datasets
- Streaming engines can be (and have been) used to process finite datasets

Data streaming characteristics

Infinite dataset

- Data is always being generated
- No control over the order in which data elements arrive

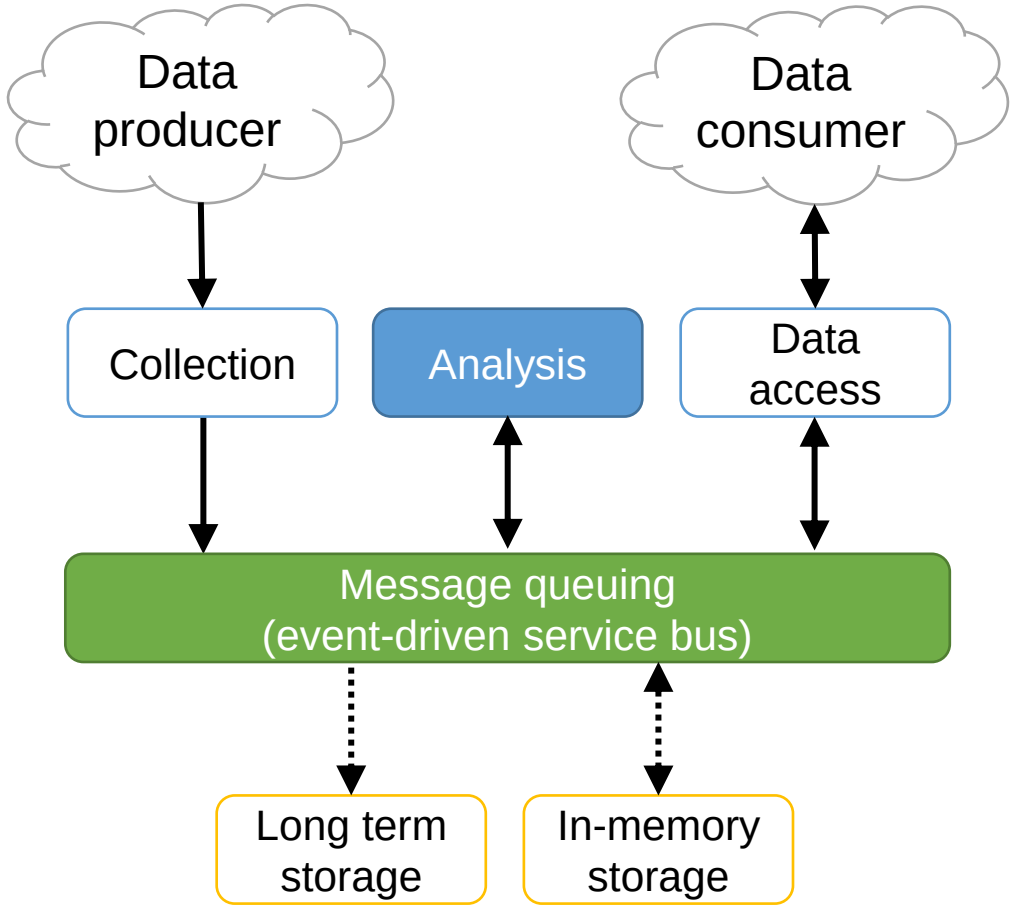
Infinite computation

- The system must be always on and must be able to keep up with the data
  - There must be a plan to avoid *overflowing* (e.g., auto-scalability)

Low-latency, approximate and/or speculative results

- Data can usually be processed a single time (*one pass*)
- Only a fraction of the dataset can be kept in memory for analyses
- Approximation may be required to accommodate the low-latency requirement

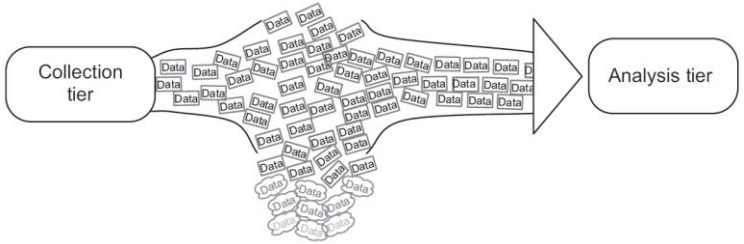
Architecture



Message queuing tier

The message queuing tier handles the transportation of data between different tiers

- Most importantly, between collection to analysis tiers



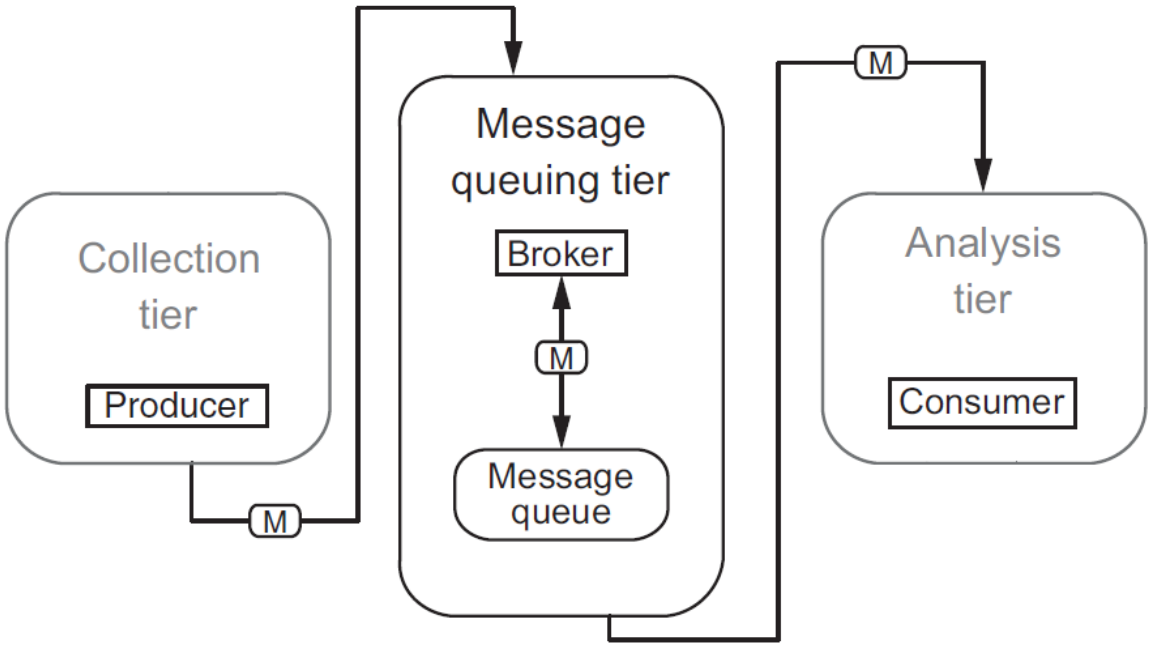
Why message queuing?

- By decoupling the pipeline of operations (collection, analysis, data access), each node in the cluster will do *one* job only
- Message queuing provides a solid framework for a safe communication between such nodes
- Message queuing handles funneling of *n* data streams to *m* consumers

Producer-Broker-Consumer

Core concepts

- The Producer and the Consumer of data
- The Broker, which manages one or more queues of data



Message delivery semantics

Exactly once: a message is never lost and is read once and only once

- Required in applications where data means money (financial/ad systems)
- Performance is sacrificed to provide safety mechanisms

At most once: a message may get lost, but it will never be read twice

- Allowed where not all data is required (monitoring systems, down-sampling)
- Fast and trivial

At least once: a message will never be lost, but it may be read twice

- Balances the two previous semantics
- Easier to provide than *exactly once*
- The consumer can still check for duplicates to adopt *exactly once*

Beware: guarantees depends on the chosen tools + application logic on the whole pipeline

Analysis: continuous queries

A continuous query is a query that is issued once and then is continuously executed against the data

- In contrast, traditional queries are simply executed once when issued
- A continuous query may need to maintain a **state**

State: an intermediate result that is continuously updated by the query

- A query is *stateless* if each execution is independent from the other

What makes continuous queries different from traditional ones?

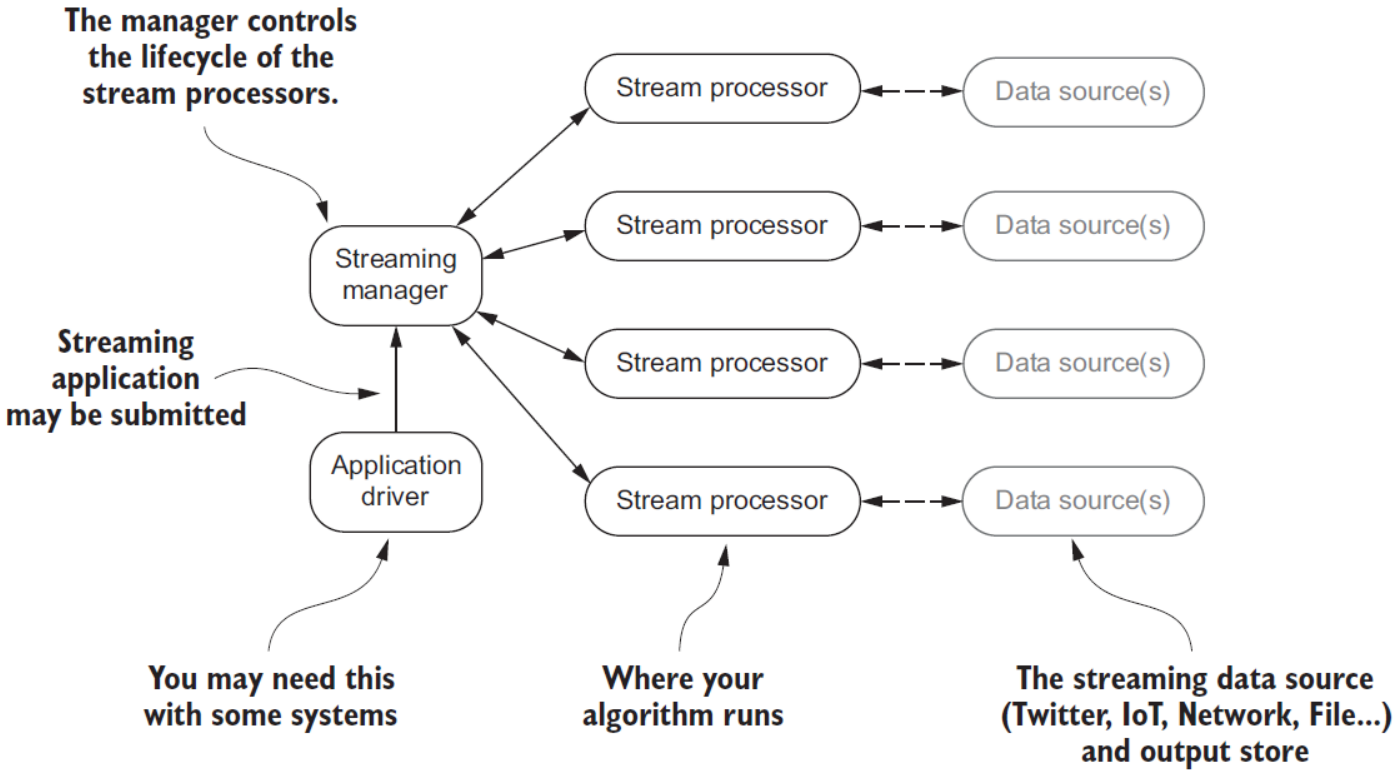
- Memory constraint (for both data processing and state maintenance)
  - Data cannot be processed altogether (***one pass* algorithms**)
  - Not much space to store the state
- Time constraint
  - Data that can't be processed in time may have to be dropped (***load shedding***)
  - Algorithms (e.g., predictive models) may lose efficacy over time (***concept drift***)

Continuous query model

	Typical query	Continuous query
Query model	Queries are based on a one-time model and a consistent state of the data **Pull model**: the user executes a query and gets an answer, and the query is forgotten	The query is continuously executed based on the data that is flowing into the system **Push model**: a user registers a query once, and the results are regularly pushed to the client
Query state	If the system crashes while a query is being executed, it must be re-issued on the whole dataset	Registered continuous queries may or may not need to continue where they left off; in the first case, a **state** has to be maintained

Distributed execution

An architecture that is common to different frameworks



Sliding windows

Sliding windows define length and period in terms of stream time

- Beware of the difference between event time and stream time

Depending on the comparison of length and period, specialized versions of sliding windows can be defined

- Fixed windows: when length and period are the same
  - E.g.: analyze the last 5 minutes of data every 5 minutes
  - *Tumbling windows* are a special case of fixed windows, where length and period are expressed in terms of number of items (instead of time)
- Overlapping windows: when the length is greater than the period
  - E.g.: analyze the last 5 minutes of data every 2 minutes
- Sampling windows: when the length is smaller than the period
  - E.g.: analyze the last 2 minutes of data every 5 minutes

Data-driven windows

Data-driven windows define the length in terms of the content that comes with the data

- In this case, the period determines the *update interval*

Typical use case: sessions

- A session is a sequence of events terminated by a gap of inactivity grater than some timeout
  - Goal: determine the average amount of traffic generated by sessions

Main characteristic: the lengths cannot be defined apriori

- How can I know when the session of a user has ended?
- How can I know when the event of a user's session are going to stop coming?

Algorithms

Consider *n* the space of events captured by the data stream

Streaming algorithm have the following requirements:

- *One-pass*; once examined, items must be discarded
- Use small space for the internal state: (O(polylog(n)))
- Fast update of the internal state: O(1) to O(polylog(n))
- Fast computation of answers
- Provide approximated answers with (ε, δ)-guarantees
  - Let R be the exact result, ε and δ > 0
  - With probability at least 1 − δ, the algorithm outputs R' such that $(1 - \varepsilon)R \leq R' \leq (1 + \varepsilon)R$
  - For instance, take ε=0.02 and δ=0.01
  - Then, I have a 99% probability that the obtained result R' equals the real result ± 2%

Algorithms (examples)

Given any list as an input

- Count the number of elements
- Find the n^{th} element

Given a list of numbers

- Find the *k* largest or smallest elements (*k* given in advance)
- Find the sum/mean/variance/st.dev. of the elements of the list

Given a list of symbols from an alphabet of k symbols (given in advance)

- Count the number of times each symbol appears in the input (*frequency*)
- Find the most or least frequent elements (*heavy hitters*)
- Sort the list according to some order on the symbols
- Find the maximum gap between two appearances of a given symbol

Algorithms (examples)

Some problems are **not** solvable by one-pass algorithms

Given any list as an input

- Find the middle element of the list

Given a list of numbers

- Find the median
- Find the most frequent symbol
- Sort the list

Approximated algorithms

Analyzing a data stream is challenging due to:

- Space constraints
- Time constraints
- Algorithm unfeasibility in one-pass

One solution is to rely on *approximated algorithms*

Approximation can be achieved by relying on two main concepts:

- Sampling
- Random projections

Approximation by sampling

Many different sampling methods have been proposed

- *E.g., Distinct sampling, Quantile sampling, Reservoir sampling*

Sampling algorithms are known on time series and cash register models to

- Find the number of distinct items
- Find the quantiles
- Find frequent items

Notes

- Some systems (e.g., IP packet sniffers) already do sampling
- Sampling is not a powerful primitive for many problems
  - Too many samples for performing sophisticated analyses
- Sampling is more challenging in the turnstile model

Approximation with random projections

An approach that relies on dimensionality reduction, using projection along random vectors

- Project high-dimensional data into a suitable lower-dimensional space..
- ..in a way which approximately preserves the distances between the points
- Projections are called *sketches*

Random projection algorithms are known on (also) turnstile models to

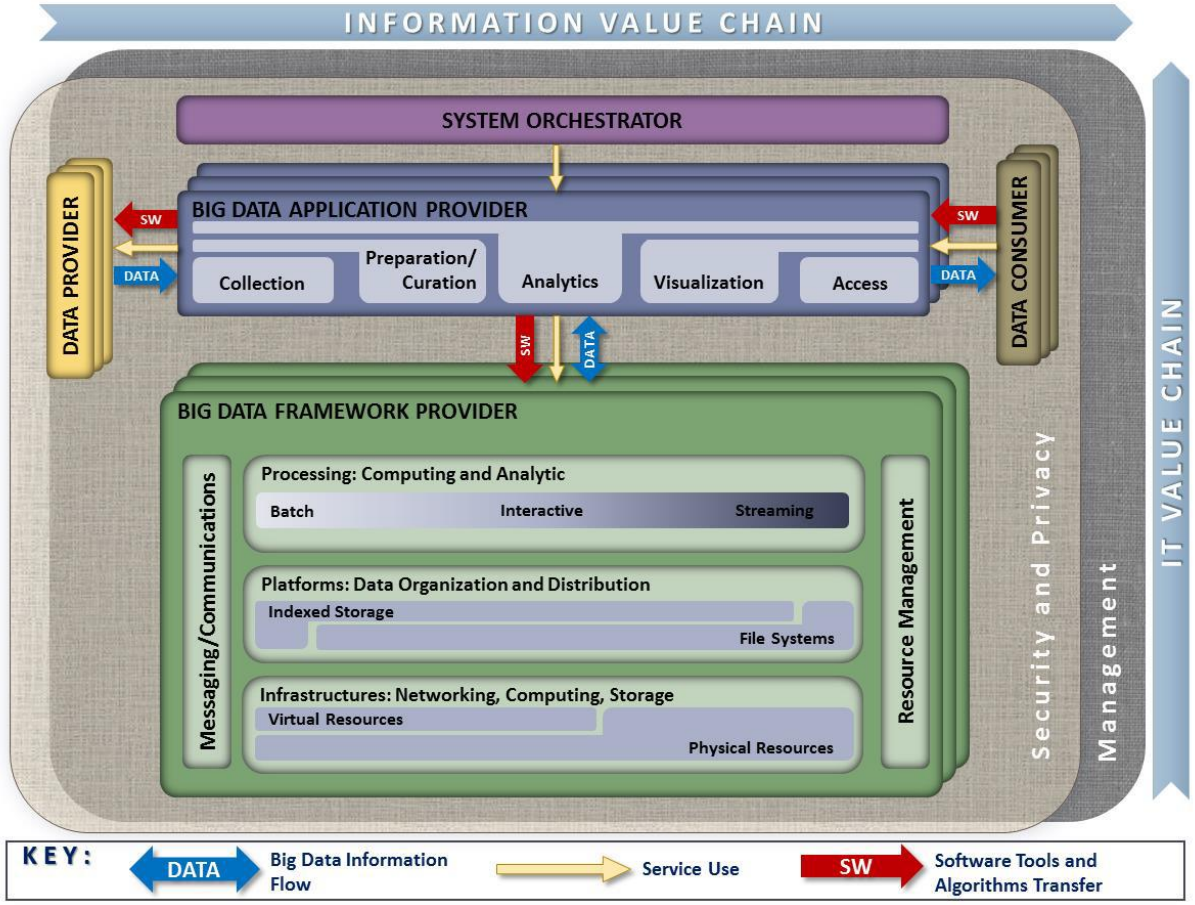
- Estimate the number of distinct elements at any time
- Estimate the quantiles at any time
- Track most frequent items, wavelets, histograms, etc.
- Random subset sums, counting sketches, Bloom filters

Reference architectures

A stack of many components

- NIST’s reference
- Microsoft’s reference
- Coexistence of batch and streaming (Lambda vs Kappa)
- The technological stack

NIST's reference



NIST's reference

System Orchestrator

- **Integrate the required application activities into an operational vertical system**
- Configure and manage the other components of the Big Data architecture to implement one or more workloads
- Monitor workloads and system to verify the meeting of quality requirements
- Elastically assign and provision additional physical or virtual resources
- Performed by human and/or software components

NIST's reference

Data Provider

- **Introduces new data or information feeds into the Big Data system**
- It can be anything from a sensor to a human or another Big Data system
- Includes the following activities:
  - Collecting and persisting the data
  - Providing transformation functions for scrubbing sensitive information
  - Creating the metadata describing the data source, usage policies/access rights, etc.
  - Enforcing access rights on data access and establishing (in)formal contracts for data access authorizations
  - Making the data accessible through suitable programmable push or pull interfaces and mechanisms
  - Publishing the availability of the information and the means to access it

Data Consumer

- **Uses the interfaces or services provided by the Big Data Application Provider to get access to the information of interest**
- It can be an actual end user or another Big Data system

NIST's reference

Big Data Application Provider

- **Executes a specific set of operations along the data life cycle**
  - Meet the requirements established by the System Orchestrator
  - Meet security and privacy requirements
- Includes the following activities:
  - Collection, Preparation, Analytics, Visualization, Access
- Each activity
  - Is specific to the application
  - Can be implemented by independent stakeholders and deployed as a stand-alone service
- There may be multiple and differing instances of each activity, or a single program may perform multiple activities
- Each of the functions can run on a separate Big Data Framework Provider or all can use a common Big Data Framework Provider

NIST's reference

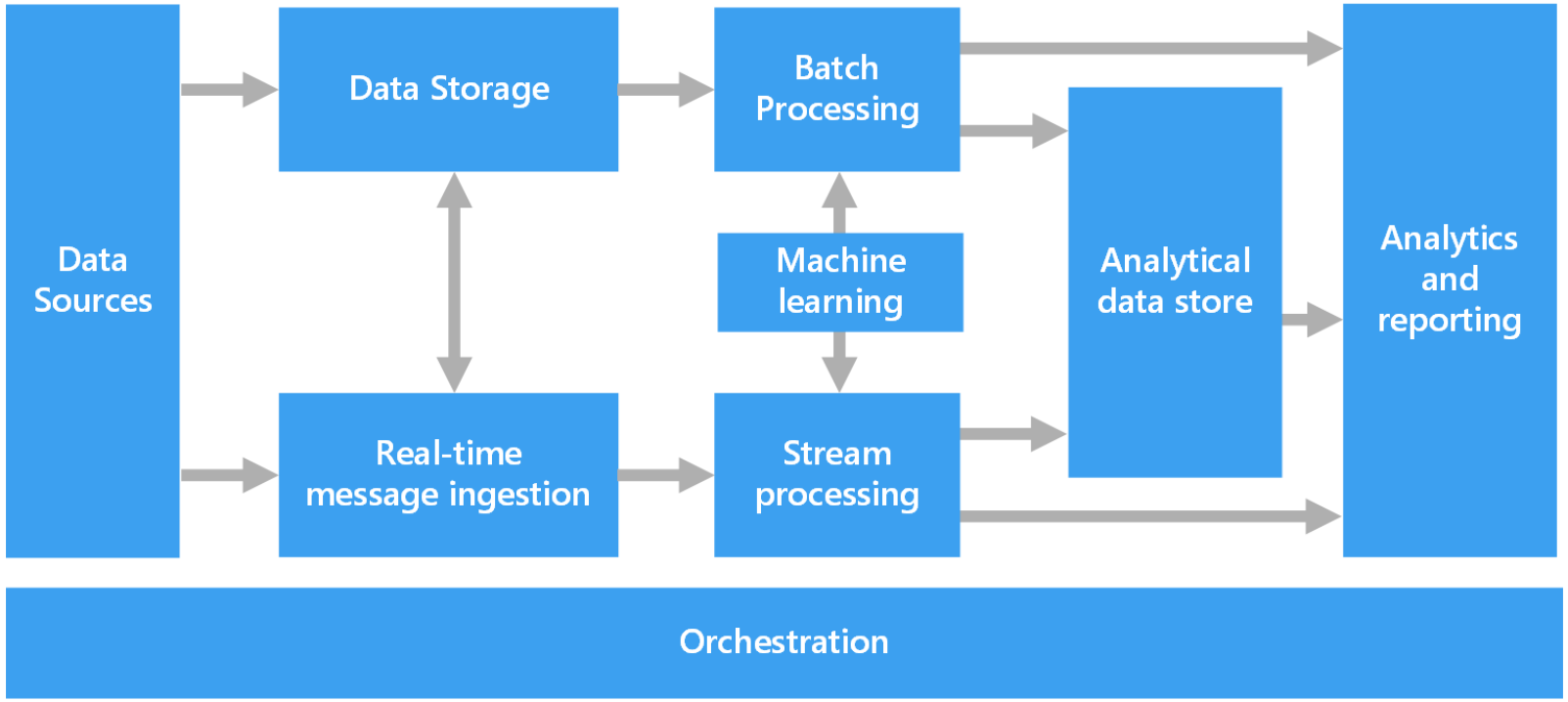
Big Data Framework Provider

- **Provides general resources or services to be used by the Big Data Application Provider in the creation of the specific application**
  - Handles the communication with the Big Data Application Provider and the assignment of physical resources to the respective activities
- Consists of Infrastructure frameworks..
  - Support the underlying computing, storage, and networking functions required to implement the overall system
  - Associated with physical or virtual infrastructure resources
- .. Data Platform frameworks..
  - Manage the organization and distribution of data (file System, databases, etc.)
- .. and Processing frameworks
  - How data will be processed in support of Big Data applications (e.g., batch/streaming framework)

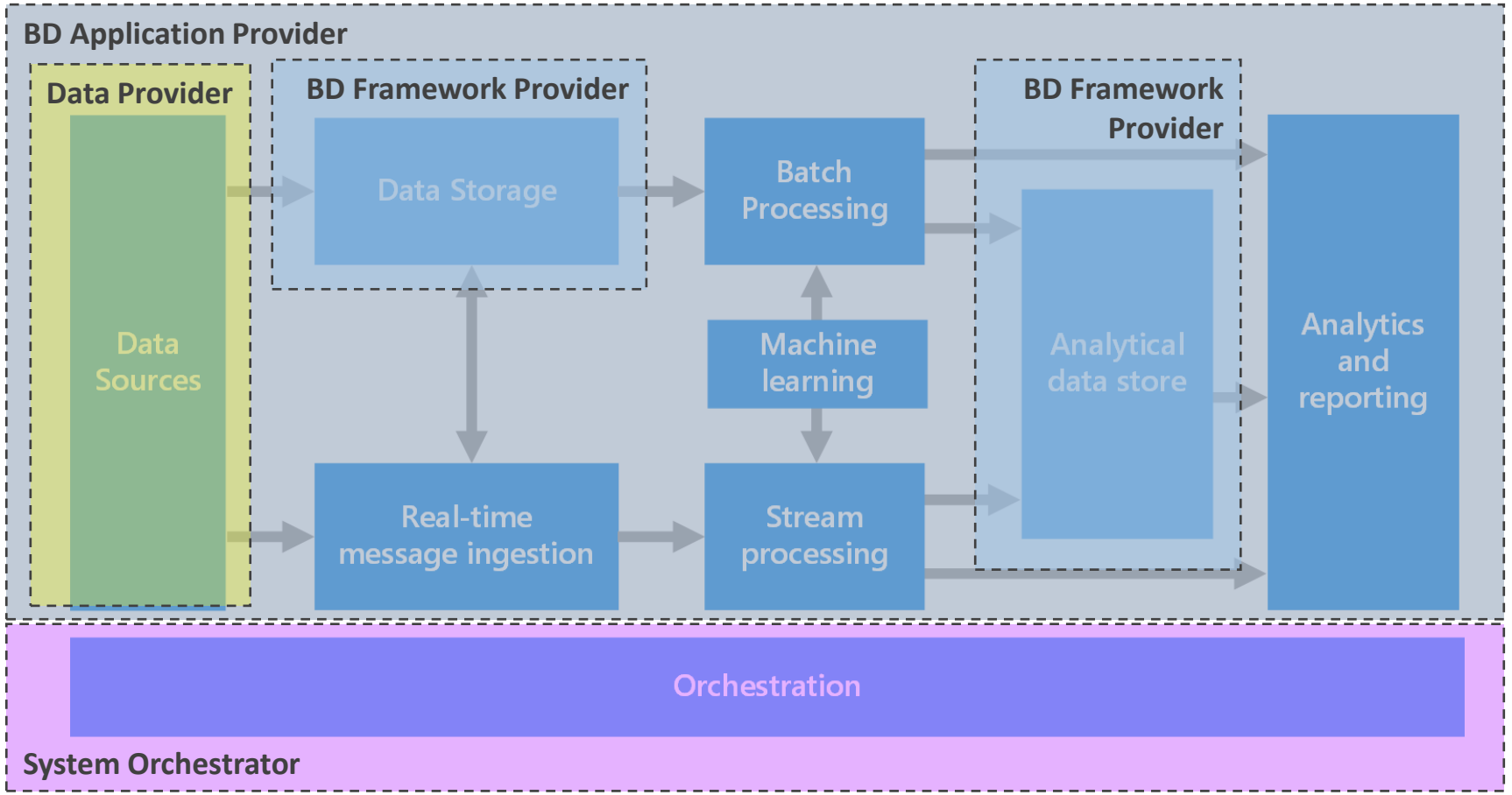
Microsoft's reference

Logical components that fit into a big data architecture

- Individual solutions may not contain every item



Microsoft's reference



Microsoft's reference

Data sources

- Data stores, static files, real-time sources

Data storage

- Distributed file store (data lake) or database

Batch processing

- Long-running batch jobs to filter, aggregate, and prepare the data for analysis

Analytical data store

- Serve the processed data in a structured format that can be queried using analytical tools

Analysis and reporting

- Traditional OLAP and BI tools, interactive exploration, analytical notebooks

Microsoft's reference

Real-time message ingestion

- Capture and store real-time messages for stream processing
- Act as a buffer for messages
- Support scale-out processing, reliable delivery, message queuing semantics

Stream processing

- Filter, aggregate, and prepare the data for analysis
- Processed stream data is then written to an output sink

Orchestration

- Automation of repeated data processing operations, encapsulated in workflows
- E.g., transforming source data, moving data between multiple sources and sinks, loading into an analytical data store, pushing results to a dashboard

Lambda vs Kappa

Batch and streaming analyses follow different principles

- Batch: large amounts of data at-rest with fault-tolerance
- Streaming: small amount of continuously incoming data without fault-tolerance

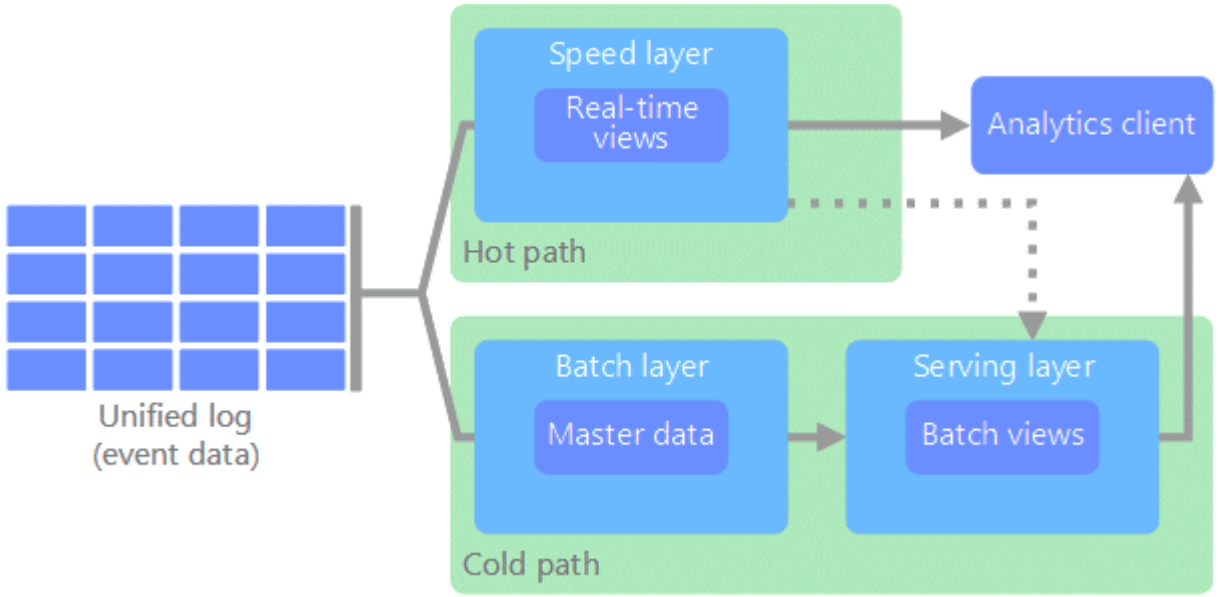
Two philosophies to make them coexist

- Lambda
- Kappa

Lambda architecture

All data coming into the system goes through two paths

- *Hot path*: for timely, yet potentially less accurate data in real time
- *Cold path*: for less timely but more accurate data



Lambda architecture

Pros of Lambda

- Emphasizes retaining the input data unchanged
- Highlights the problem of reprocessing data (bugs, evolution, etc.)

Cons of Lambda

- Parallel development and maintenance of two parallel pipelines
- Same goal, different languages, different frameworks

Streaming: a superset of batch

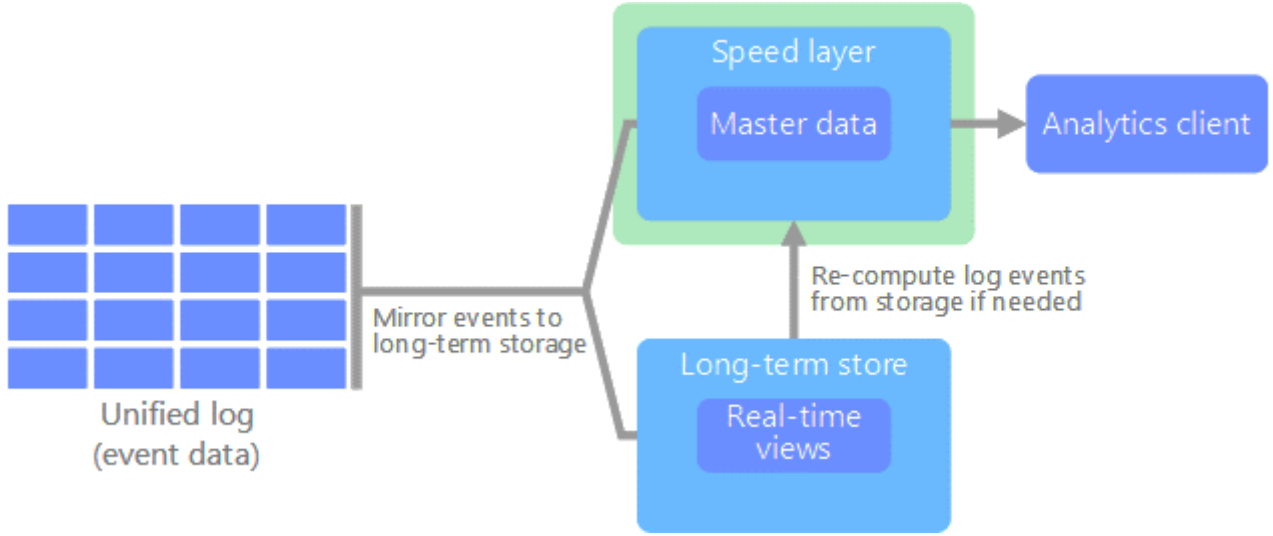
Can we do better?

- Well-designed streaming systems provide a strict superset of batch functionality
- Introduce correctness (exactly-once semantics, strong consistency) and the result of a streaming job is the same of batch job
- A streaming engine can handle a bounded dataset

Kappa architecture

All data flows through a single path, using a stream processing system

- Similar to a lambda architecture's speed layer, all event processing is performed on the input stream and persisted as a real-time view
- To recompute the entire data set (equivalent to what the batch layer does in lambda), simply replay the stream



Kappa architecture

~~Modeling everything under the streaming paradigm is not trivial~~

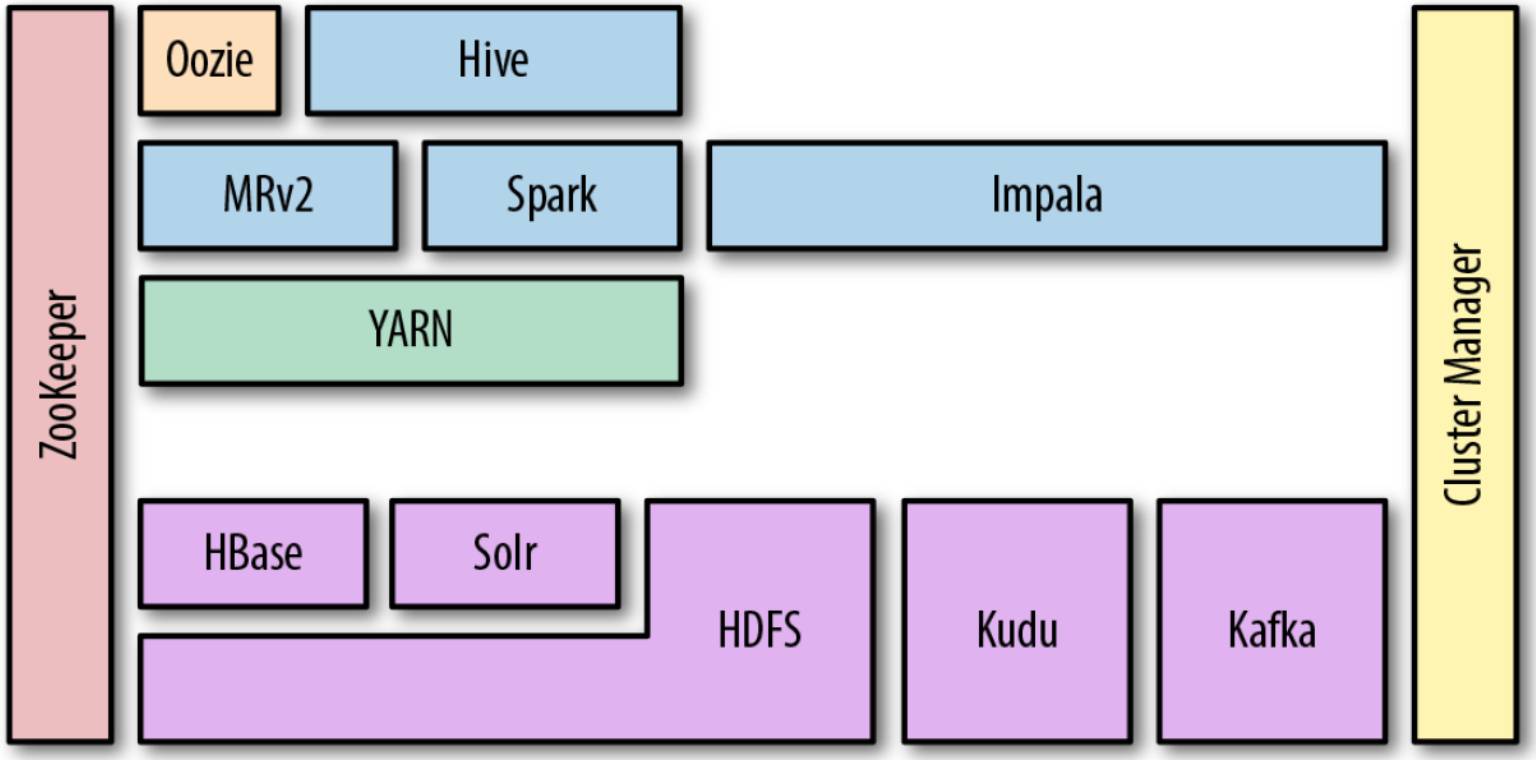
Advanced tools provide a single framework and unified APIs for writing and executing both batch and streaming jobs

- Google Cloud Dataflow
- Apache Flink

“Broad maturation of streaming systems combined with robust frameworks for unbounded data processing will in time allow for the relegation of the Lambda Architecture to the antiquity of big data history where it belongs”

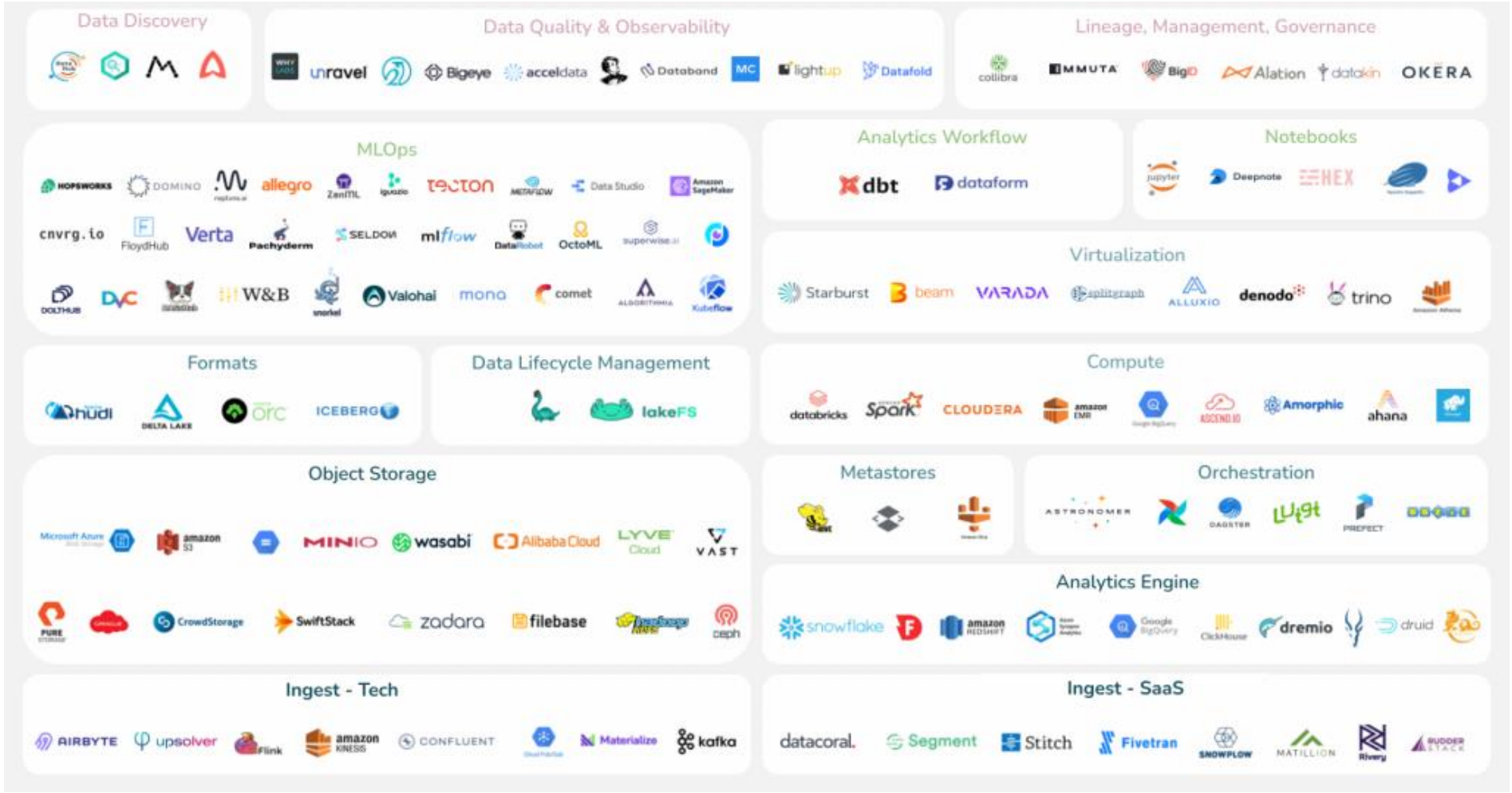
The technological stack

Sample Hadoop-based deployment



The technological stack

Beyond Hadoop



On-premises vs cloud

First installations where primarily on-premises

- **Installation**: how do I set up a new machine?
- **Networking**: how do I cable dozens of machines?
- **Management**: how do I replace a broken disk?
- **Upgrade**: how do I extend the cluster with new services/machines?
- (energy and cooling, software licenses, insurance...)

Today, cloud providers are mature and offer comprehensive support for big data ecosystems

- Forget all that is above

Cloud computing

A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, services) that can be rapidly provisioned and released with minimal management effort or service provider interaction

- On-demand self-service (consume services when you want)
- Broad network access (consume services from anywhere)
- Broad network deployment (deploy services anywhere)
- Resource pooling (infrastructure, virtual platforms, and applications)
- Rapid elasticity (enable horizontal scalability)
- Measured service (pay for the service you consume as you consume)
- Reliability (fault-tolerance handled by the provider)

Conclusions

Learned the basis of big data (a summary of the Master's degree course)

- Definitions and motivations
- Main components – focus on storage and processing
- Hardware and software architectures

Up next: an overview of some of the main challenges

- Polyglot persistence
- The metadata challenge

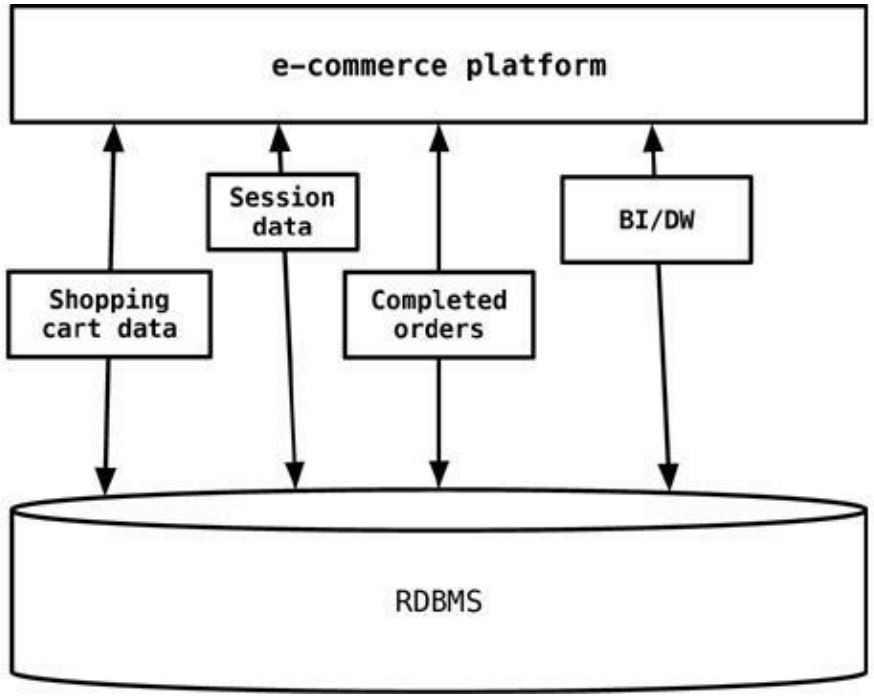
Polyglot persistence

What we are going to do

- Define polyglot persistence
- Define challenges
- Discuss current solutions and future directions

Polyglot persistence

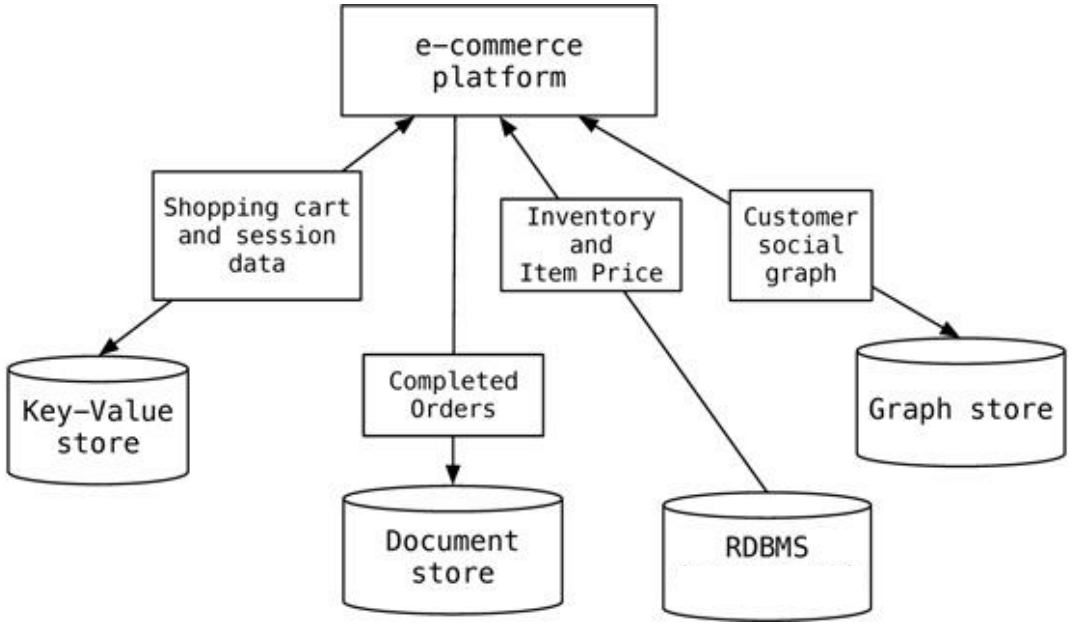
The *one-size-fits-all* solution



Polyglot persistence

~~The *one-size-fits-all* solution~~

Replaced by the *polyglot* solution



NoSQL databases

Emerged around the 2010's, NoSQL databases offer efficient solutions to specific problems

- Mainly distinguished by the supported data model

Model	Description	Use cases
Key-value	Associates any kind of value to a string	Dictionary, lookup table, cache, file and images storage
Document-based	Stores hierarchical data in a tree-like structure	Documents, anything that fits into a hierarchical structure
Wide-column	Stores sparse matrixes where a cell is identified by the row and column keys	Crawling, high-variability systems, sparse matrixes
Graph	Stores vertices and arches	Social network queries, inference, pattern matching

NoSQL databases: key-value

Each DB contains one or more collections (corresponding to tables)

Each collection contains a list of key-value pairs

- Key: a unique string
  - E.g.: ids, hashes, paths, queries, REST calls
- Value: a BLOB (binary large object)
  - E.g.: text, documents, web pages, multimedia files

Looks like a simple dictionary

- The collection is indexed by key
- The value may contain several information: one or more definitions, synonyms and antonyms, images, etc.

Key	Value
image-12345.jpg	Binary image file
http://www.example.com/my-web-page.html	HTML of a web page
N:/folder/subfolder/myfile.pdf	PDF document
9e107d9d372bb6826bd81d3542a419d6	The quick brown fox jumps over the lazy dog
view-person?person-id=12345&format=xml	<Person><id>12345</id .</Person>
SELECT PERSON FROM PEOPLE WHERE PID="12345"	<Person><id>12345</id .</Person>

NoSQL databases: document-based

Each DB contains one or more collections (corresponding to tables)

Each collection contains a list of documents (usually JSON)

- Documents are hierarchically structured

Each document contains a set of fields

- The ID is mandatory

Each field corresponds to a key-value pair

- Key: unique string in the document
- Value: either simple (string, number, boolean) or complex (object, array, BLOB)
  - A complex field can contain other field

```
{
    "_id": 1234,
    "name": "Enrico",
    "age": 33,
    "address": {
        "city": "Ravenna",
        "postalCode": 48124
    },
    "contacts": [ {
        "type": "office",
        "contact": "0547-338835"
    }, {
        "type": "skype",
        "contact": "egallinucci"
    } ]
}
```

NoSQL databases: wide-column

Each DB contains one or more column families (corresponding to tables)

Each column family contains a list of rows in the form of a key-value pair

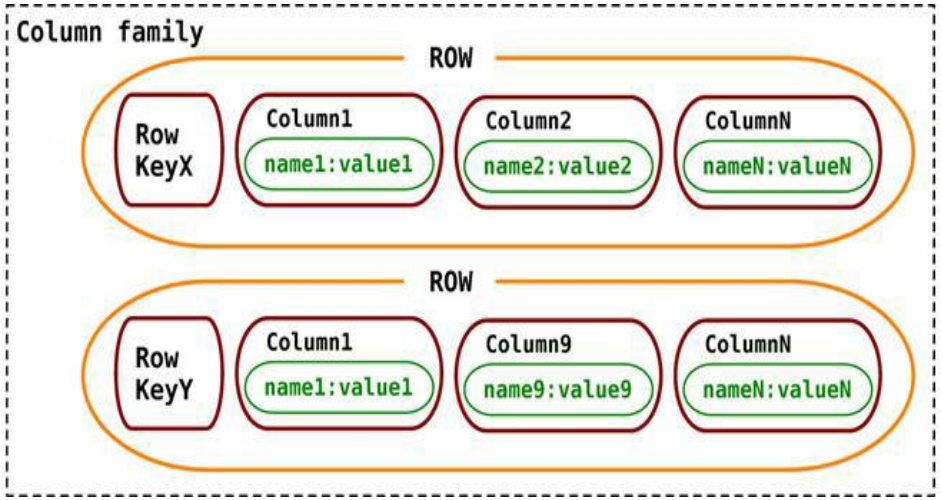
- Key: unique string in the column family
- Value: a set of columns

Each column is a key-value pair itself

- Key: unique string in the row
- Value: simple or complex (*supercolumn*)

With respect to the relational model:

- Rows specify only the columns for which a value exists
  - Particularly suited for sparse matrixes
- Timestamps can be used to defines *versions* of column values



NoSQL databases: graph

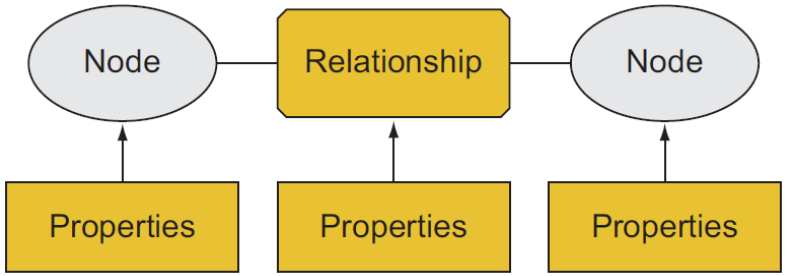
Each DB contains one or more graphs

Each graph contains vertices and arcs

- Vertices: usually represent real-world entities
  - E.g.: people, organizations, web pages, workstations, cells, books, etc.
- Arcs: represent directed relationships between the vertices
  - E.g.: friendship, work relationship, hyperlink, ethernet links, copyright, etc.
- Vertices and arcs are described by properties
- Arcs are stored as physical pointers

Most known specializations:

- Reticular data model
  - Parent-child or owner-member relationships
- Triplestore
  - Subject-predicate-object relationships (e.g., RDF)



Data modelling

Key-value, document and wide column are called aggregate-oriented

- Aggregate = key-value pair, document, row (respectively)
- The aggregate is the atomic block (no guarantees for multi-aggregate operations)

Based on the concept of encapsulation

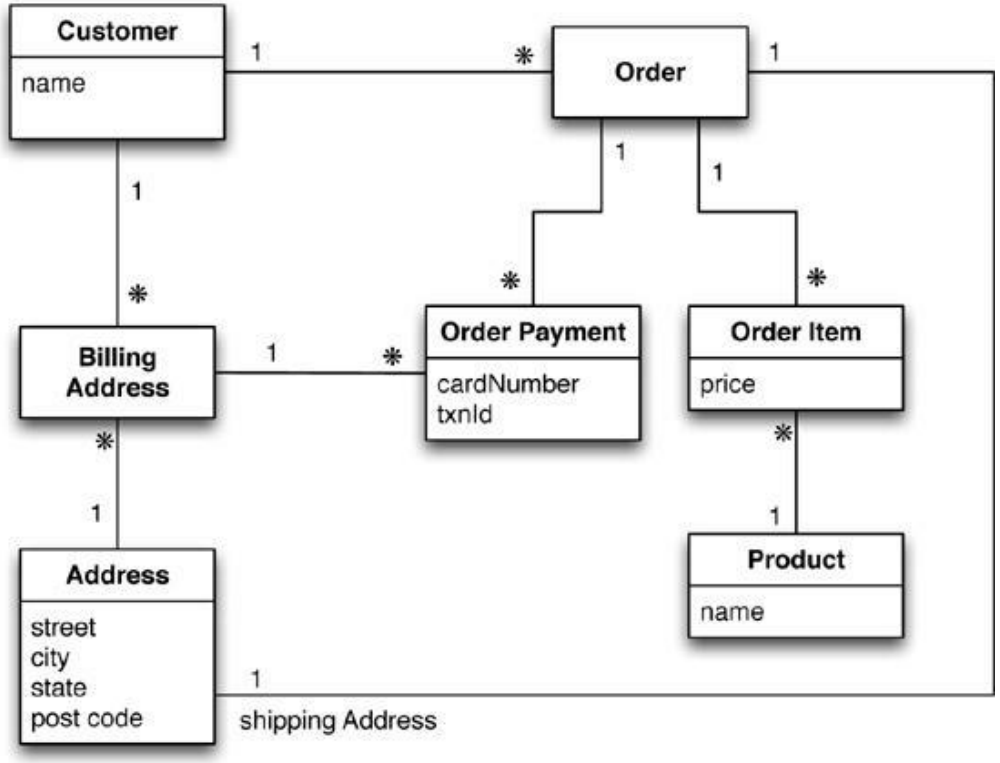
- Pro: avoid joins as much as possible → achieve **high scalability**
- Con: data denormalization → **potential inconsistencies in the data**
- Query-driven modeling

The graph data model is intrinsically different from the others

- Focused on the relationships rather than on the entities per-se
- **Limited scalability**: it is often impossible to shard a graph on several machines without "cutting" several arcs (i.e. having several cross-machine links)
  - Batch cross-machine queries: don't follow relationships one by one, but "group them" to make less requests
  - Limit the depth of cross-machine node searches
- Data-driven modeling

Data modelling

Typical use case: customers, orders and products



Relational data modelling

Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Address

Id	City
77	Chicago

OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

NoSQL data modelling (kv)

Customer collection

key	value
cust-1:name	Martin
cust-1:adrs	[{"street":"Adam", "city":"Chicago", "state":"Illinois", "code":60007}, {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}]

cust-1:ord-99	{
	"orderpayments": [
	{"card":477, "billadrs":
	{"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007} },
	{"card":457, "billadrs":
	{"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
	],
	"products": [
	{"id":1, "name":"Cola", "price":12.4},
	{"id":2, "name":"Fanta", "price":14.4}
	],
	"shipAdrs": {"street":"9th", "city":"NewYork", "state":"NewYork", code":10001}
	}

Product collection

key	value
p-1:name	Cola
p-2:name	Fanta

NoSQL data modelling (document)

Customer collection

```
{
  "_id": 1,
  "name": "Martin",
  "adrs": [
    {"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007},
    {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
  ],
  "orders": [ {
    "orderpayments":[
      {"card":477, "billadrs": {"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007}},
      {"card":457, "billadrs": {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}}
    ],
    "products":[
      {"id":1, "name":"Cola", "price":12.4},
      {"id":2, "name":"Fanta", "price":14.4}
    ],
    "shipAdrs": {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
  }]
}
```

Product collection

```
{
  "_id":1,
  "name":"Cola",
  "price":12.4
}
```

```
{
  "_id":1,
  "name":"Fanta",
  "price":14.4
}
```

NoSQL data modelling (document)

```
{
  "_id": 1,
  "name": "Martin",
  "adrs": [
    {"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007},
    {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
  ]
}
```

Customer collection

```
{
  "_id":1,
  "name":"Cola",
  "price":12.4
}
```

```
{
  "_id":1,
  "name":"Fanta",
  "price":14.4
}
```

Product collection

```
{
  "_id": 1,
  "customer":1,
  "orderpayments":[
    {"card":477, "billadrs":{"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007}},
    {"card":457, "billadrs":{"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}}
  ],
  "products": [
    {"id":1, "name":"Cola", "price":12.4},
    {"id":2, "name":"Fanta", "price":14.4}
  ],
  "shipAdrs": {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
}
```

Order collection

NoSQL data modelling (wide-column)

Order table > Order details column family

Ord	CustName	Pepsi	Cola	Fanta	…
1	Martin		12.4	14.4	
2	…	…			…

Order table > Order payments column family

Ord	OrderPayments
1	Card / Steet / City / State / Code 477 / 9th / NewYork / NewYork / 10001 457 / Adam / Chicago / Illinois / 60007
2	…

NoSQL data modelling

The *aggregate* term comes from Domain-Driven Design

- An aggregate is a group of tightly coupled objects to be handled as a block
- Aggregates are the basic unit for data manipulation and consistency management

Advantages

- Can be distributed trivially
  - Data the should be used together (e.g., orders and details) are stored together
- Facilitate the developer's job
  - By surpassing the impedance mismatch problem

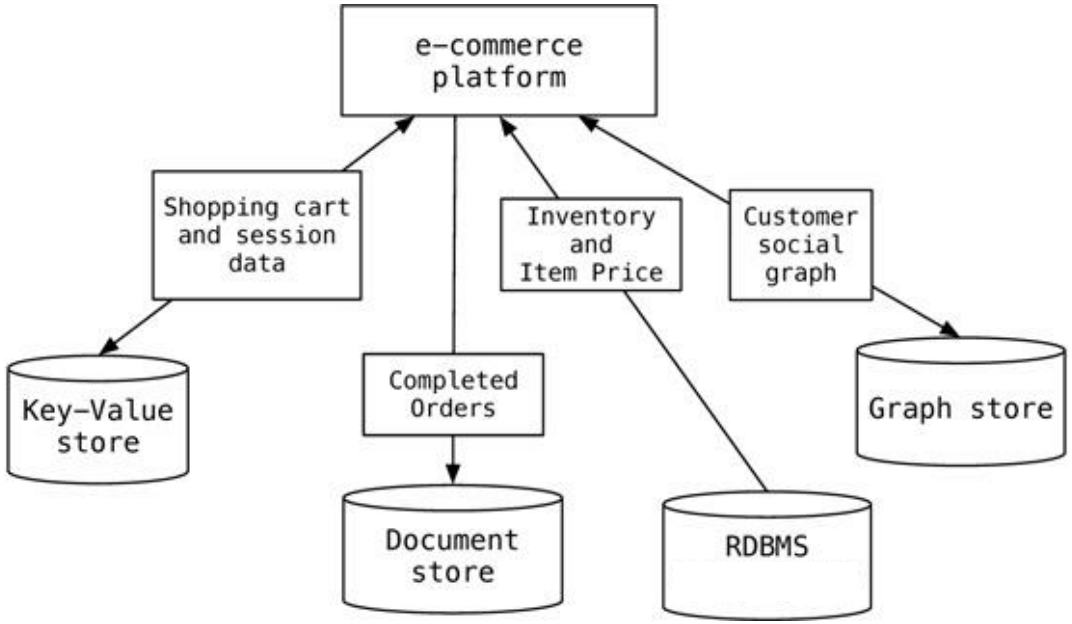
Disadvantages

- **No design strategy exists for aggregates**
  - It only depends on how they are meant to be used
- Can optimize only a limited set of queries
- Data denormalization → possible inconsistencies

RDBMSs are agnostic from this point of view

Polyglot persistence

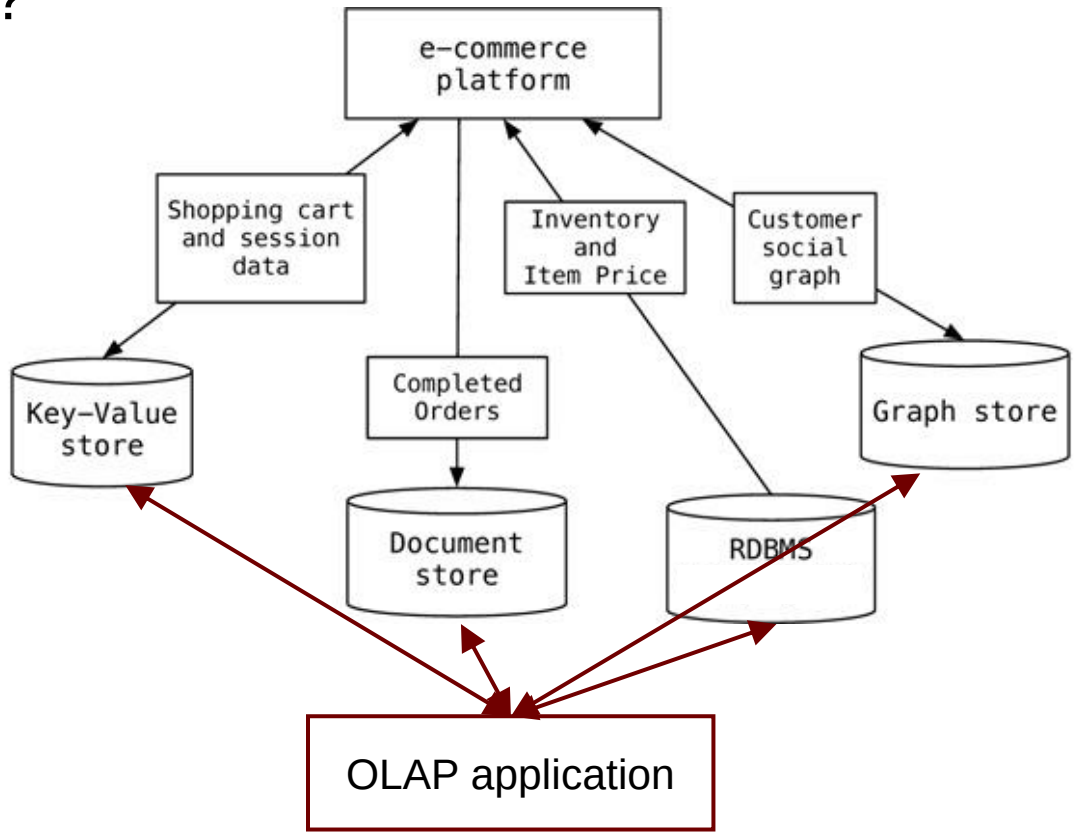
To each application the appropriate DBMS



Polyglot persistence

To each application the appropriate DBMS works well for OLTP

What about OLAP?



Polyglot persistence: main challenges

Data model heterogeneity

- Support multiple models in the same database
- Or integrate data from different databases using different query languages

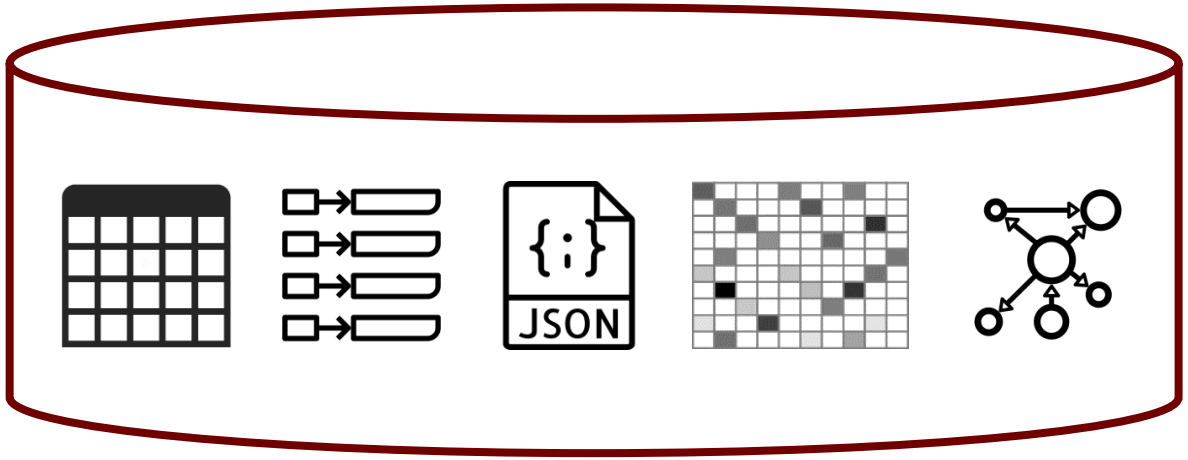
Schema heterogeneity

- Inter-collection: different records in different collections have different schemas
  - Not a new problem: think federated databases, corporate mergers, etc.
- Intra-collection: different records in the same collection have different schemas
  - Emerged with NoSQL databases

Data inconsistency

- Reconcile inconsistent versions of the same data (inter- or intra-collection)

Data model heterogeneity



Basic solutions

Some DBMSs offer multi-model support

- Extended RDBMSs
  - KV implementable as a table with two fields: a string key, and a blob value
  - Cypher query language on top of a relational implementation of a graph
  - Hstore data type in PostgreSQL for wide-column-like implementation
  - **Scalabilty issue remains**
- Multi-model NoSQL DBMSs
  - ArangoDB, OrientDB
  - **Support all NoSQL data models, but not the relational one**

Some approaches suggest strategies to model everything within RDBMSs

- DiScala, M., Abadi, D.J.: Automatic generation of normalized relational schemas from nested key-value data. In: 2016 ACM SIGMOD Int. Conf. on Management of Data, pp. 295-310. ACM (2016)
- Tahara, D., Diamond, T., Abadi, D.J.: Sinew: a SQL system for multi-structured data. In: 2014 ACM SIGMOD Int. Conf. on Management of Data, pp. 815-826. ACM (2014)

A taxonomy for distributed solutions

Federated database system

- **Homogeneous** data stores, exposes a **single** standard query interface
- Features a mediator-wrapper architecture, employs schema-mapping and entity-merging techniques for integration of relational data

Polyglot system

- **Homogeneous** data stores, exposes **multiple** query interfaces
- Takes advantage of the semantic expressiveness of multiple interfaces (e.g., declarative, procedural)

Multistore system

- **Heterogeneous** data stores, exposes a **single** query interface
- Provides a unified querying layer by adopting ontologies and applying schema-mapping and entity-resolution techniques

Polystore system

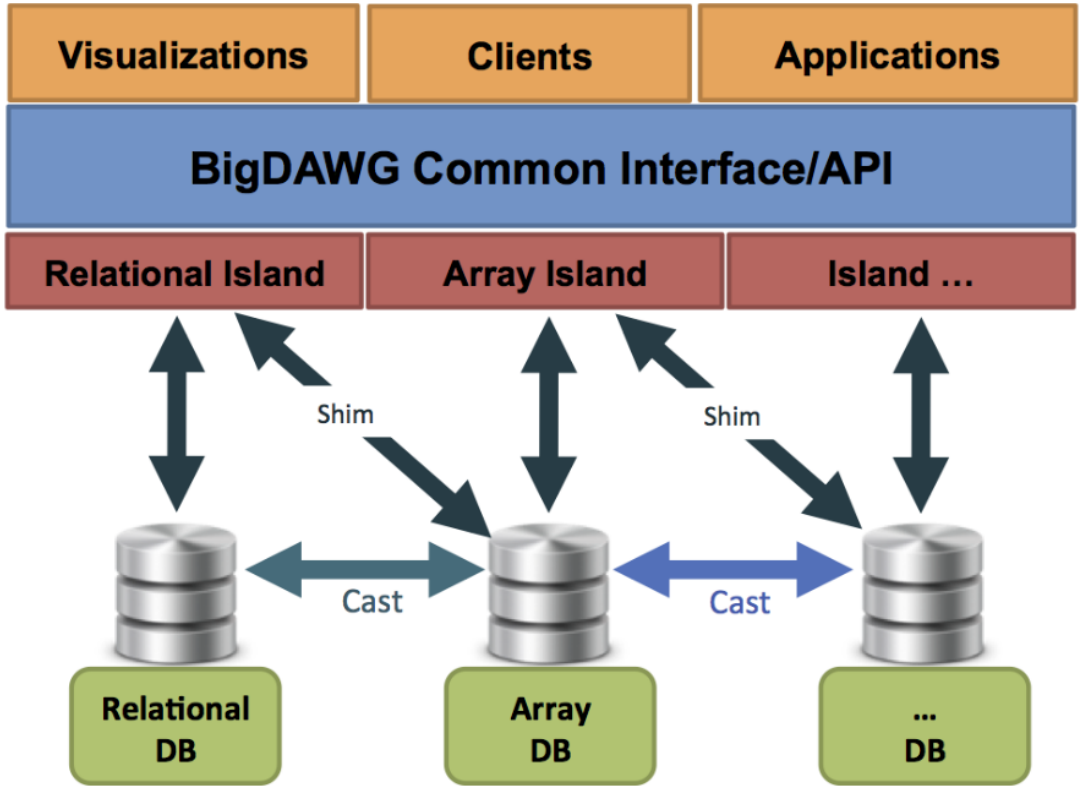
- **Heterogeneous** data stores, exposes **multiple** query interfaces
- Choose from a variety of query interfaces to seamlessly query data residing in multiple data stores

R. Tan, R. Chirkova, V. Gadepally and T. G. Mattson, "Enabling query processing across heterogeneous data models: A survey," *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 3211-3220, doi: 10.1109/BigData.2017.8258302.

Advanced solutions

Example of a polystore

- Island = a middleware application to support a set of operations on a given data model
- Shim = a wrapper to convert from the island's query language to the target DB's query language



Vijay Gadepally, Kyle O'Brien, Adam Dziedzic, Aaron J. Elmore, Jeremy Kepner, Samuel Madden, Tim Mattson, Jennie Rogers, Zuohao She, Michael Stonebraker: Version 0.1 of the BigDAWG Polystore System. CoRR abs/1707.00721 (2017)

Advanced solutions

Most notable multistore/polystore proposals

- BigDAWG
  - V. Gadepally et al. Version 0.1 of the BigDAWG Polystore System. CoRR abs/1707.00721 (2017)
  - Focus on the ability to “move” data from one DB to another to improve query efficiency
- Estocada
  - R. Alotaibi et al. ESTOCADA: Towards Scalable Polystore Systems. Proc. VLDB Endow. 13(12): 2949-2952 (2020)
  - Focus on taking advantage of possible (consistent) redundancy and previous query results
- Awesome
  - S. Dasgupta. Analytics-driven data ingestion and derivation in the AWESOME polystore. IEEE BigData 2016: 2555-2564
  - Focus on supporting common analytical functions
- CloudMdsQL
  - B. Kolev et al. CloudMdsQL: querying heterogeneous cloud data stores with a common language. Distributed Parallel Databases 34(4): 463-503 (2016)
  - Focus on taking advantage of local data store native functionalities

Beyond data model heterogeneity

What else is there?

Entity resolution

- Every approach needs some kind of integrated knowledge
- Ample research from federated database systems
- Usually “hidden”

Management of **schema heterogeneity** and **data inconsistency**

- Usually addressed as different problems in the literature

Schema heterogeneity

Heterogeneous data stored with variant schemata and structural forms

- Missing/additional attributes
- Different names/types of attributes
- Different nested structures

Two main problems

- Understand the data
- Query the data

Understanding the data

Early work on XML

- To deal with the widespread lack of DTDs and XSDs
- Extract regular expressions to described the content of elements in a set of XML documents

Recent work on JSON

- **Concise view**: a single representation for all schema variations
  - Union of all attributes
    - M. Klettke et al. Schema extraction and structural outlier detection for JSON-based NoSQL data stores., in: Proc. BTW, volume 2105, 2015, pp. 425-444.
  - A *skeleton* as the smallest set of core attributes according to a frequency-based formula
    - L. Wang et al. Schema management for document stores, Proc. VLDB Endowment 8 (2015) 922-933.
- **Comprehensive view**: multiple representations (a different schema for every document)
    - D. S. Ruiz, et al. Inferring versioned schemas from NoSQL databases and its applications, in: Proc. ER, 2015, pp. 467-480.
- **Schema profile**: *explain why* there are different schemas
    - Enrico Gallinucci et al. Schema profiling of document-oriented databases. Inf. Syst. 75: 13-25 (2018)

Schema profiling

Schema profiles explain

- What are the differences between schemas
- When/why is one schema used instead of the other

SchemaID	User	Activity	Weight	Duration	Repetitions
S1	Jack	Run		108	
S2	John	Leg press	80	4	23
S1	Kate	Walk		42	
S3	John	Push-ups		8	40

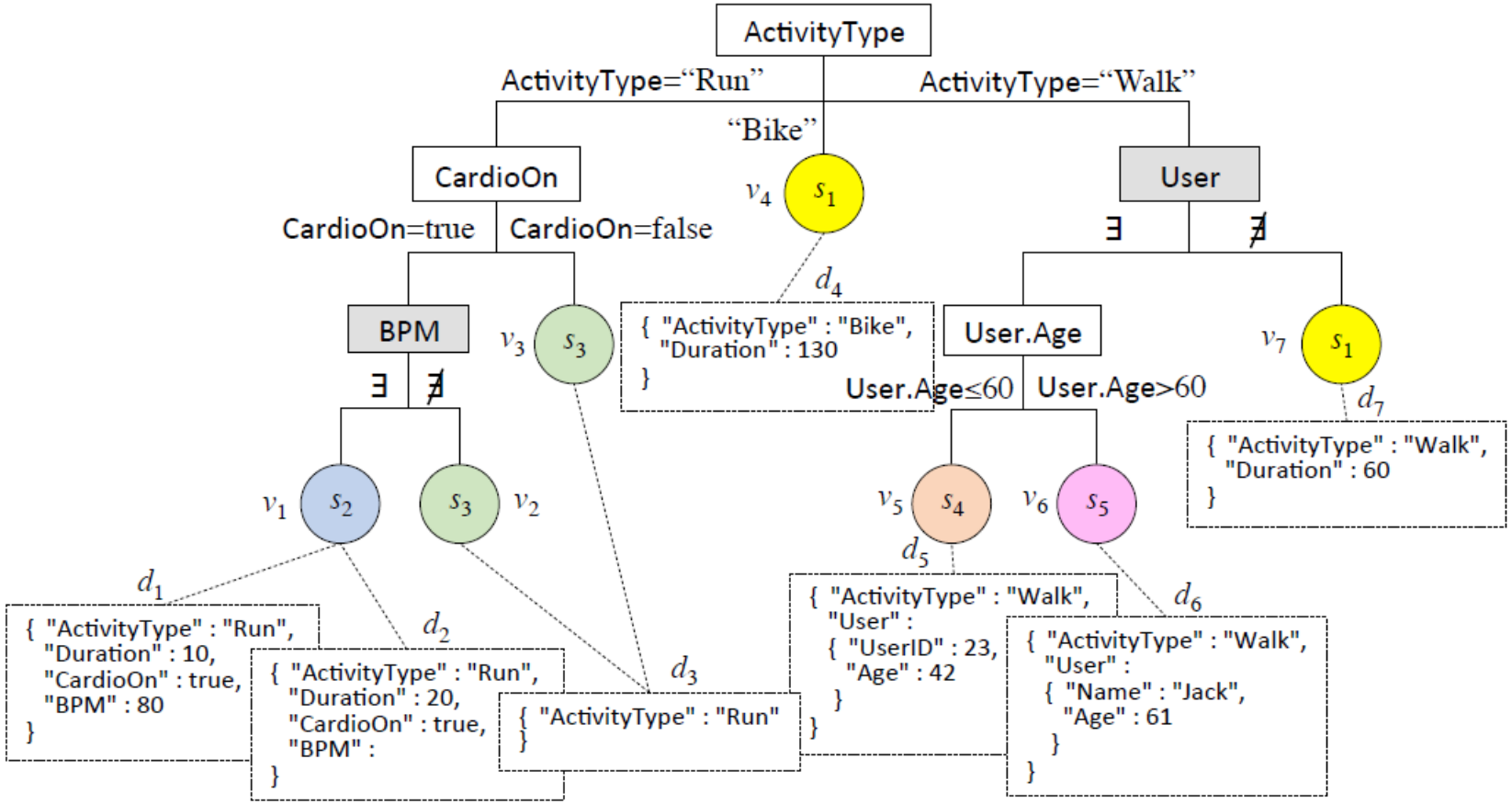
Documents / Observations

Schema / Class

The problem of schema profiling is quite similar to a classification problem

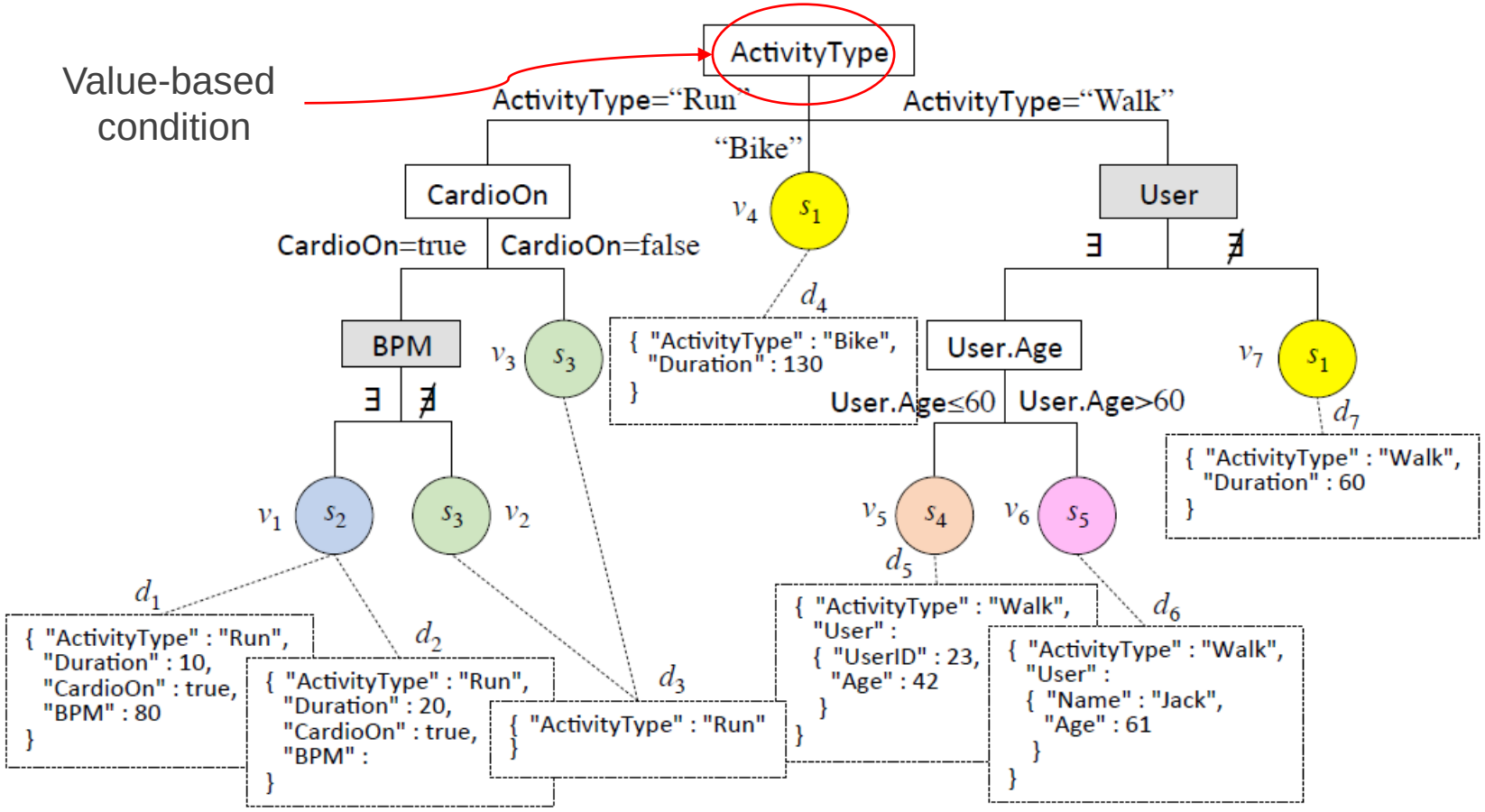
- Classifiers are also used to describe the rules for assigning a class to an observation based on the other observation features
- Based on the requirements collected from potential users, **decision trees** emerged as the most adequate

Schema profiling

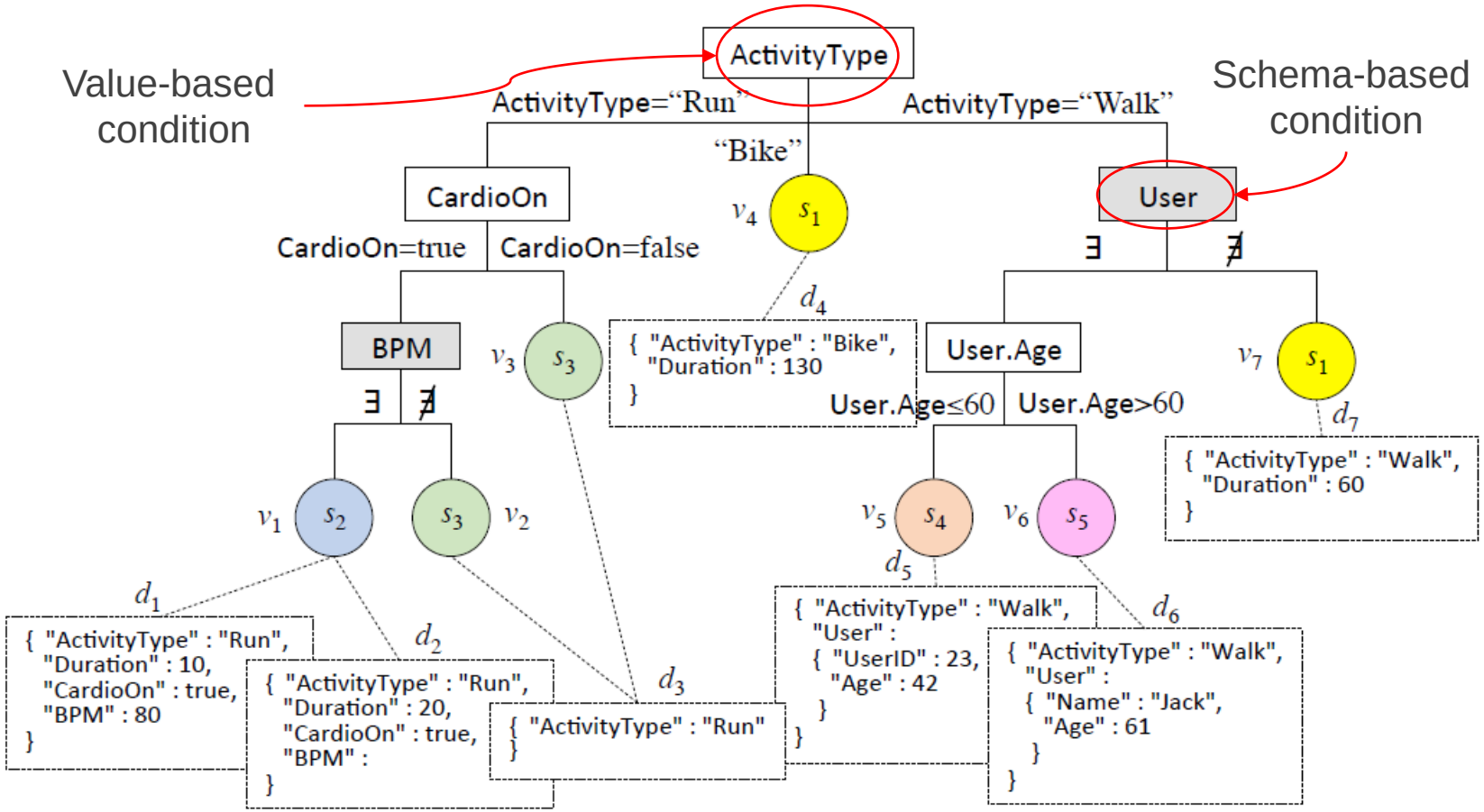


The **documents** are the observations
The **schemata** are the classes

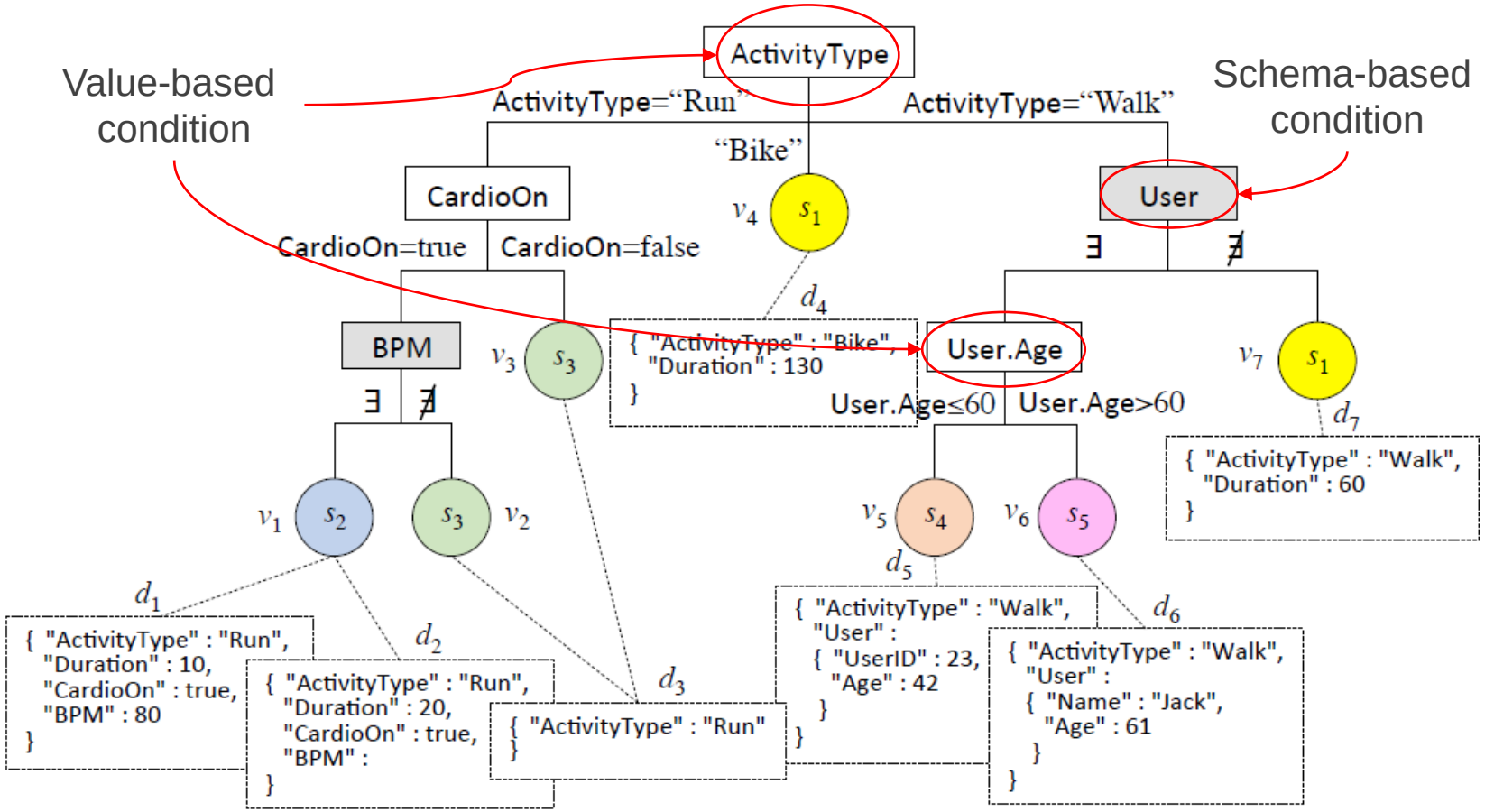
Schema profiling



Schema profiling



Schema profiling



Preliminary activities

Semi-structured interviews with 5 users

- Application domains: fitness equipment sales, software development
- Understand goals, requirements, visualization format
- Not one complete/correct dataset description

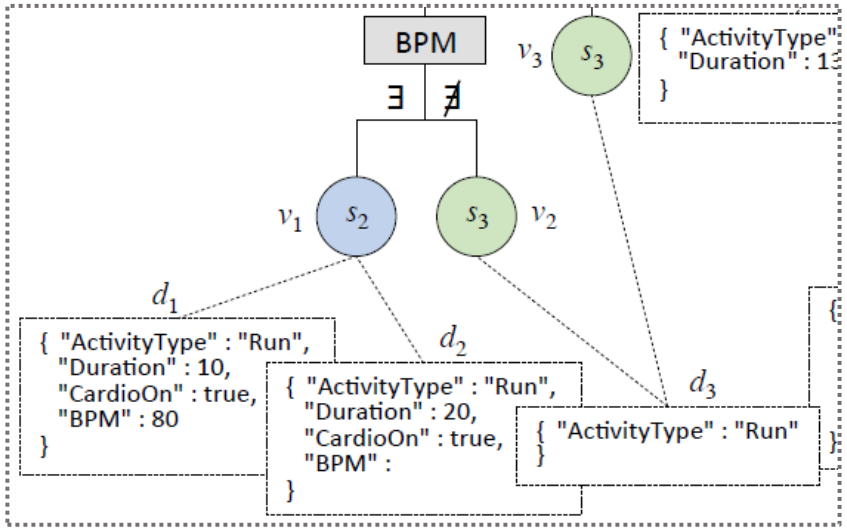
Definition of schema profile characteristics

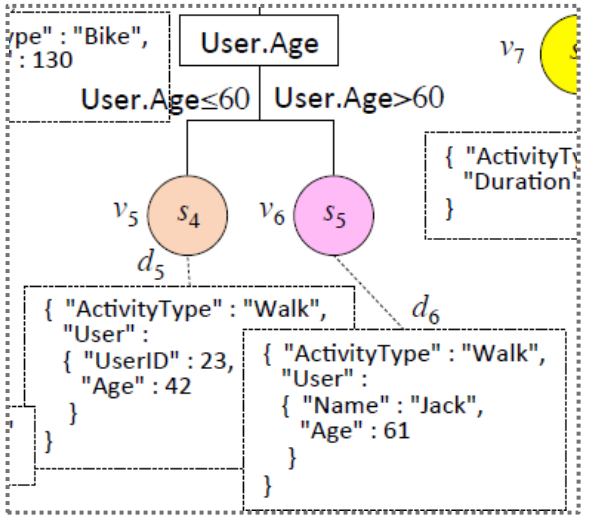
- Explicativeness
- Precision
- Conciseness

Explicativeness

Value-based (VB) conditions are preferred to schema-based (SB) ones

- SB: **acknowledge** a difference between schemata
- VB: **explain** it in terms of the values taken by an attribute





The less SB conditions, the more explicativeness

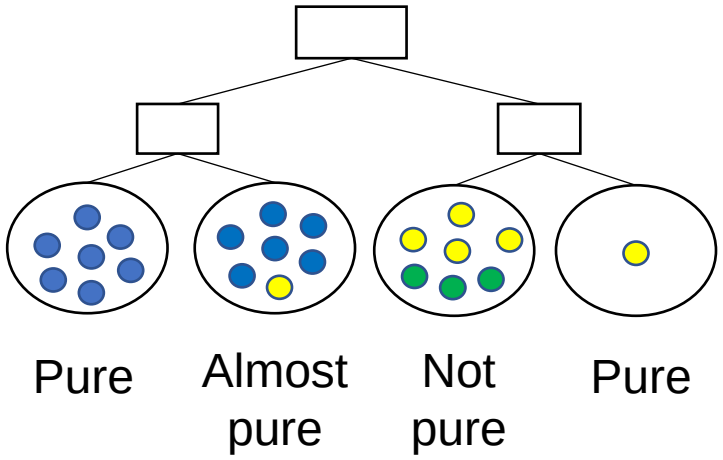
Precision

A decision tree is precise if all the leaves are pure

- A leaf is **pure** if all its observations belong to the **same class**
- Leaf v_j is pure if *entropy(v_j) = 0*

Entropy is strictly related to **precision**

- Divisive approaches typically stop only when the leaves are all pure



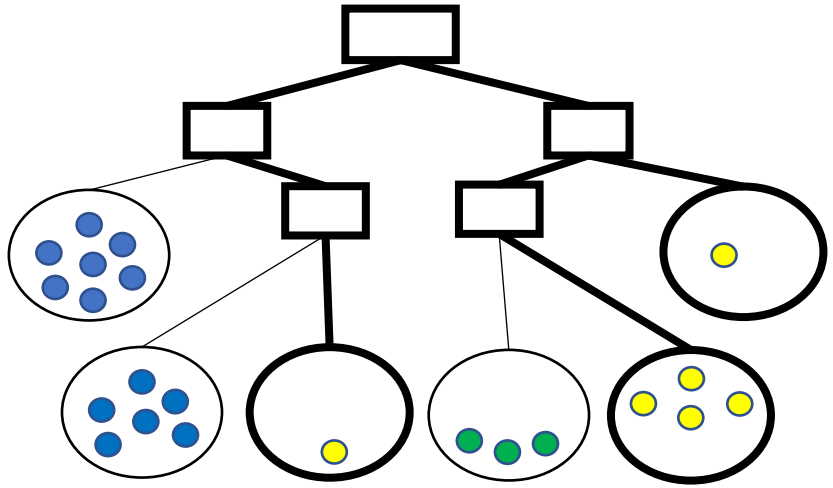
$$entropy(v_j) = -\sum_{s \in S(D_v)} \frac{|D_{v_j}|_s}{|D_{v_j}|} \log \frac{|D_{v_j}|_s}{|D_{v_j}|}$$

probability of schema *s* within leaf v_j

Precision and conciseness

Minimization of entropy often leads to **splitting observations** of the same class **among several leaves**

- Entropy's sole focus is on node purity
- More frequent when the number of classes is high



Typically, precision is more important than readability

In schema profiling, this is a critical problem

- **It conflicts with the conciseness requirement**

Conciseness

A maximally concise schema profile is one where there is **a single rule for each schema**

Schema entropy: inverts the original definition of entropy, relating it to the **purity of the schemata** instead of the purity of the leaves

- Entropy:
  a leaf is pure if
  it contains only documents
  with the **same class**

$$entropy(v_j) = -\sum_{s \in S(D_v)} \frac{|D_{v_j}|_s}{|D_{v_j}|} \log \frac{|D_{v_j}|_s}{|D_{v_j}|}$$

- Schema entropy:
  a schema is pure if
  all its documents
  are in the **same leaf**

$$sEntropy(s) = \sum_{j=1}^{m} \frac{|D_{v_j}|_s}{|D|_s} \log \frac{|D_{v_j}|_s}{|D|_s}$$

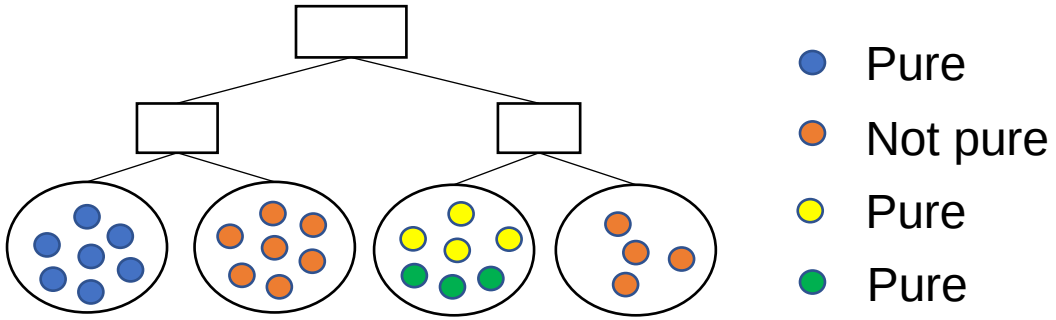
Conciseness

A maximally concise schema profile is one where there is
a single rule for each schema

Schema entropy: inverts the original definition of entropy, relating it to the **purity of the schemata** instead of the purity of the leaves

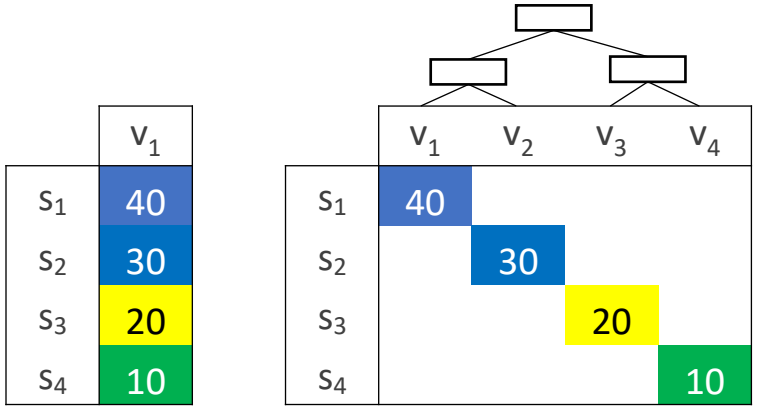
- Entropy:
  a leaf is pure if
  it contains only documents
  with the **same class**

- Schema entropy:
  a schema is pure if
  all its documents
  are in the **same leaf**



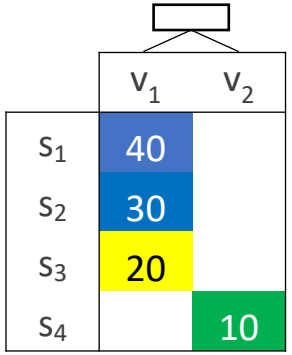
Schema profiling example

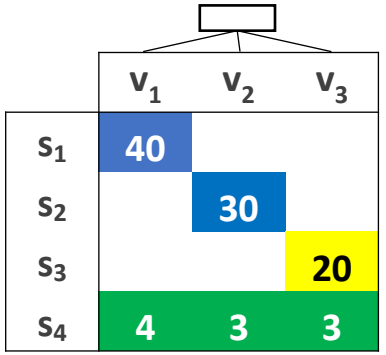
Starting situation
E = 1,85 (maximum)
SE = 0 (minimum)



Best outcome
E = 0
SE = 0

E = 1,38
SE = 0





E = 0,46
SE = 0,16

Querying the data

One thing to understand the data, another thing is enabling querying over heterogeneous data

What we need

- Integration techniques to solve schema heterogeneity and produce a global knowledge
- Query rewriting techniques to translate queries on the global knowledge to queries on the actual schemas

(Focus on OLAP queries)

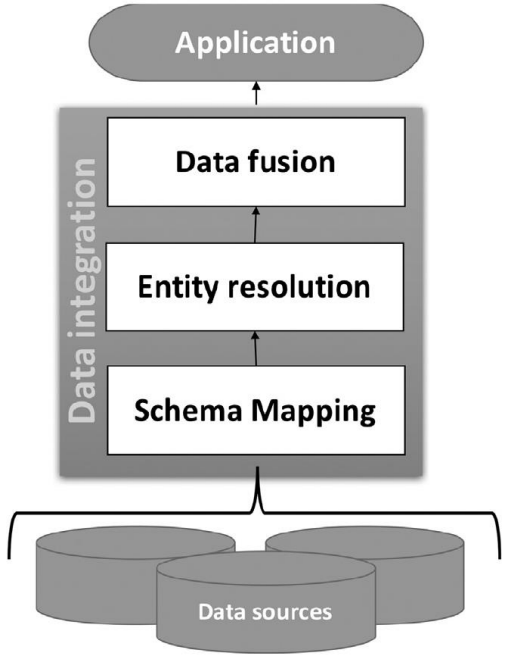
Integration techniques

Integration at the intensional level

- Schema matching and mapping
  - A match is a correspondence between attributes
  - A mapping is a function to explain the relationship between attributes
    - E.g., S1.FullName = CONCAT(S2.FirstName, S2.LastName)

Integration at the extensional level

- Entity resolution (a.k.a. record linkage or duplicate detection)
  - Identifying (or linking, or grouping) different records referring to the same real-world entity
  - Aims at removing redundancy and increasing conciseness
- Data fusion
  - Fuse records on the same real-world entity into a single record and resolve possible conflicts
  - Aims at increasing correctness of data

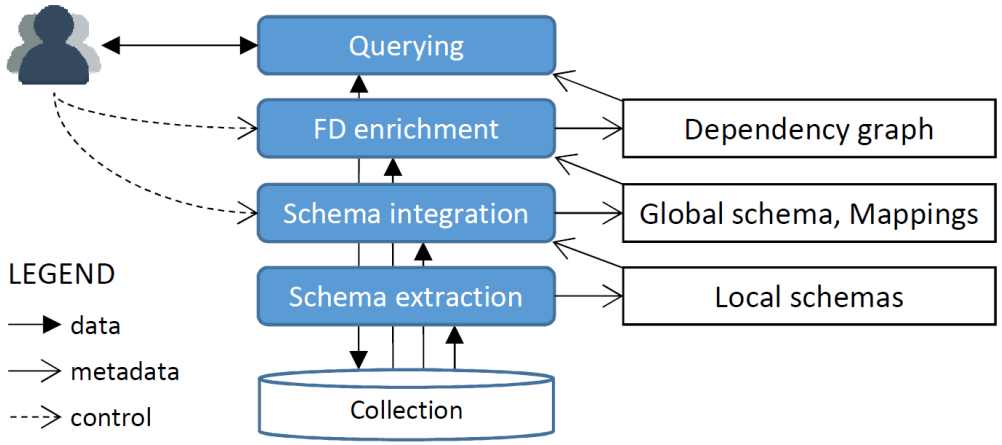


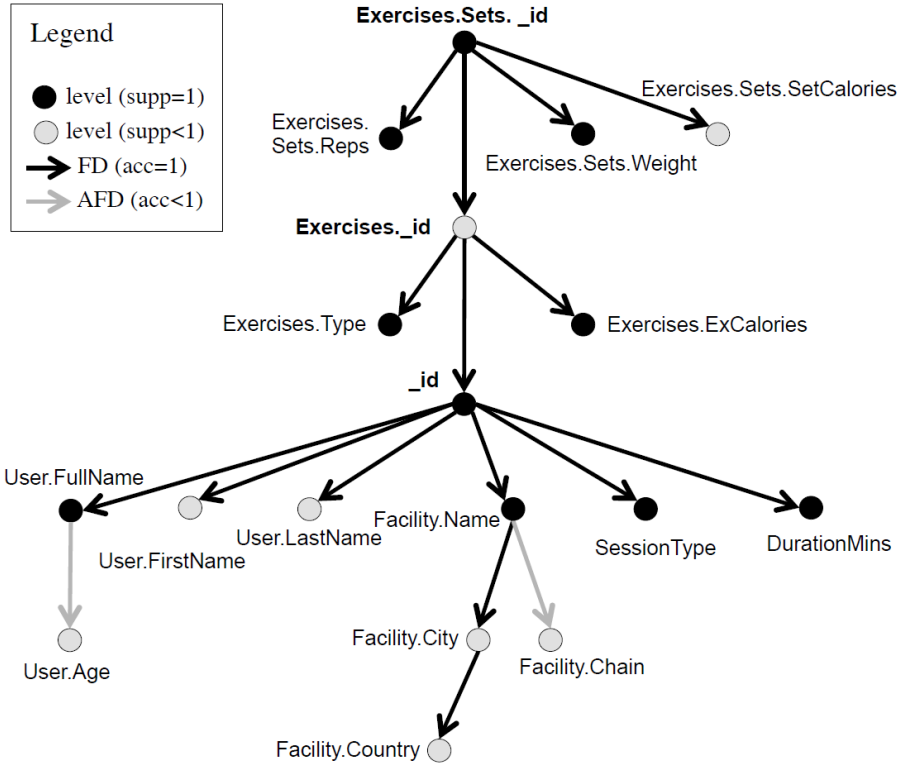
E. Rahm, P.A. Bernstein, A survey of approaches to automatic schema matching, VLDB J. 10 (4) (2001)

Mandreoli, F., & Montangero, M. (2019). Dealing with data heterogeneity in a data fusion perspective: models, methodologies, and algorithms. In *Data Handling in Science and Technology* (Vol. 31, pp. 235-270). Elsevier.

OLAP querying

A first approach to OLAP on heterogeneous data





Gallinucci, E., Golfarelli, M., & Rizzi, S. (2019). Approximate OLAP of document-oriented databases: A variety-aware approach. Information Systems, 85, 114-130.

OLAP querying

Some limitations

- Expensive querying
  - Does not scale well with the number of schemas
- Expensive integration
  - High levels of heterogeneity imply complex rewriting rules (requiring knowledge and time)
  - Assuming to be *always* able to obtain a global schema is a bit pretentious
    - *“One does not simply define a global schema”*

New integration techniques

Replace the global schema with a *dataspace*

- A dataspace is a lightweight integration approach providing basic query expressive power on a variety of data sources, bypassing the complexity of traditional integration approaches and possibly returning best-effort or approximate answers
  - Franklin, M., Halevy, A., & Maier, D. (2005). From databases to dataspaces: a new abstraction for information management. *ACM Sigmod Record*, *34*(4), 27-33.

Replace traditional integration with a *pay-as-you-go* approach

- The system incrementally understands and integrates the data over time by asking users to confirm matches as the system runs
  - Jeffery, S. R., Franklin, M. J., & Halevy, A. Y. (2008, June). Pay-as-you-go user feedback for dataspace systems. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (pp. 847-860).

New integration techniques

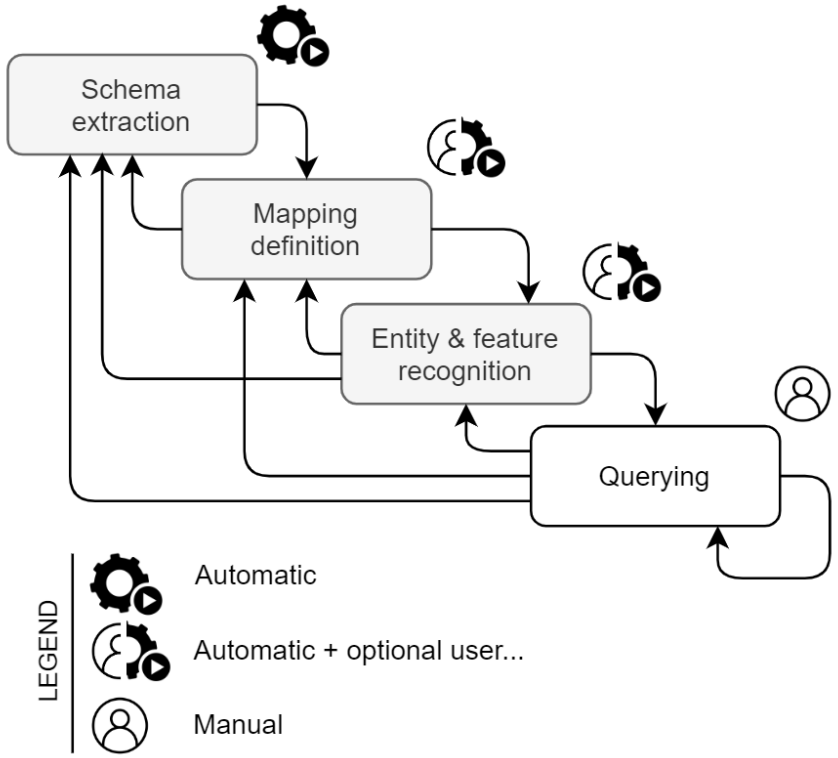
Introducing new concepts

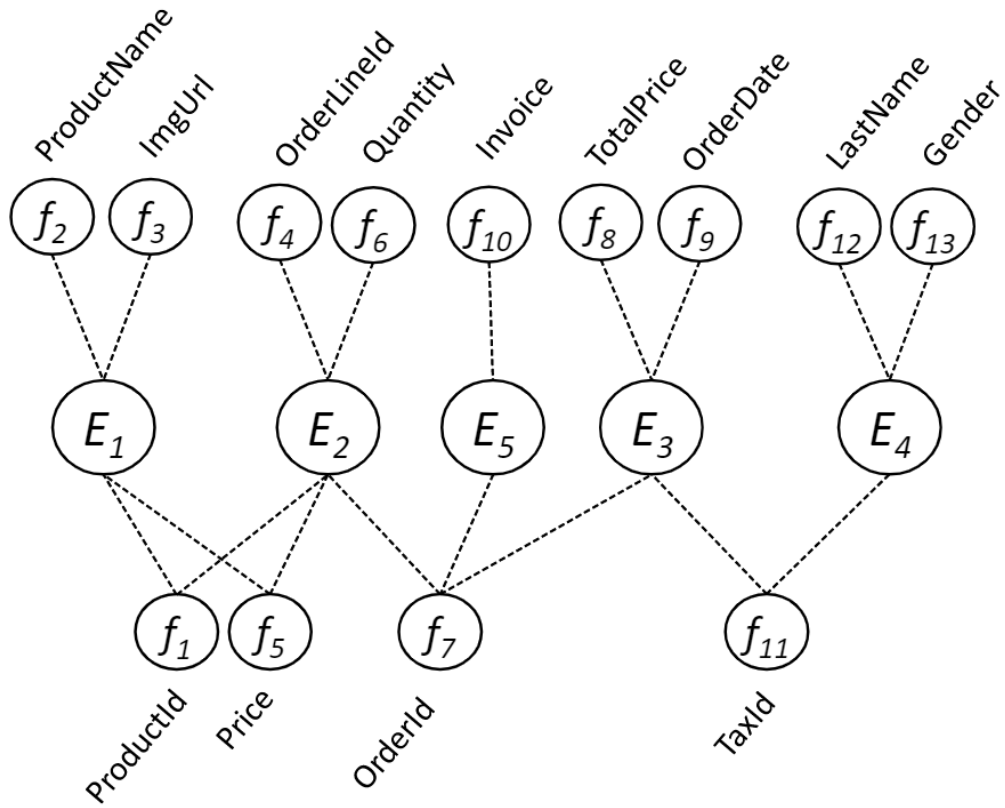
- Features: univocal representation of a group of semantically equivalent attributes
  - E.g., CustomerName = { S1.name, S2.fullname, S3.customer, S4.cName, … }
  - Mapping functions must be defined/definable between every couple
- Entities: representation of a real-world entity
  - E.g., customers, products, orders, etc.

The dataspace becomes an abstract view in terms of features and entities

New OLAP querying

What it looks like



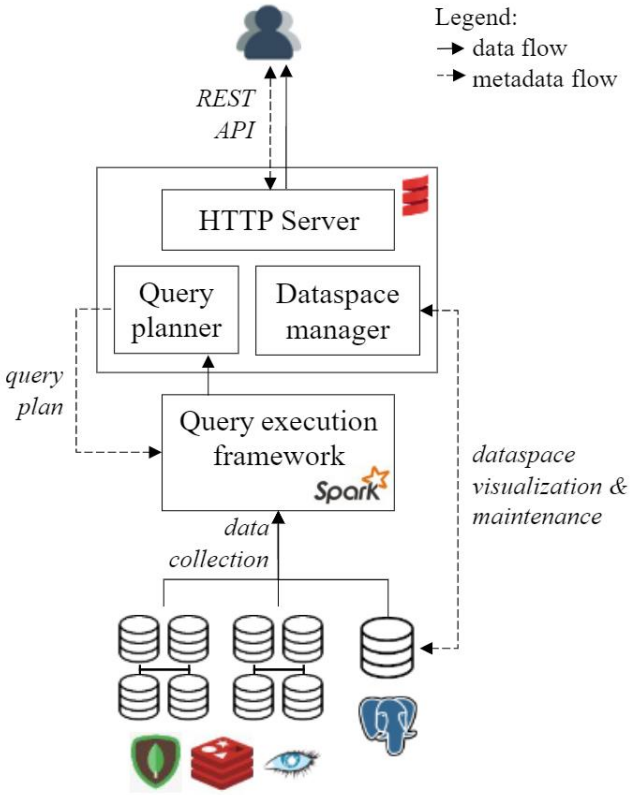


Forresi, C., Gallinucci, E., Golfarelli, M., & Hamadou, H. B. (2021). A dataspace-based framework for OLAP analyses in a high-variety multistore. *The VLDB Journal*, *30*(6), 1017-1040.

New OLAP querying

Previous issues

- ~~Expensive querying~~
  - Schema heterogeneity solved at query time
  - Requires complex - but feasible - algorithms
- ~~Expensive integration~~
  - Pay-as-you-go approach is quicker, iterative, and more flexible
  - Dataspace is conceptual, untied to logical data modeling



Now we have a multistore dealing with multiple data models and schema heterogeneity

Forresi, C., Gallinucci, E., Golfarelli, M., & Hamadou, H. B. (2021). A dataspace-based framework for OLAP analyses in a high-variety multistore. *The VLDB Journal*, *30*(6), 1017-1040.

Data inconsitency

Intra-collection

- Due to denormalized data modeling

Inter-collection

- Due to analytical data offloading
  - To reduce costs and optimize performance, the historical depth of databases is kept limited
  - After some years, data are offloaded to cheaper/bigger storages, e.g., cloud storages, data lakes
  - Offloading implies a change of data model, a change of schema, and obviously an overlapping of instances with the original data
- Due to multi-cloud architectures
  - Enables the exploitation of data spread across different providers and architectures, all the while overcoming data silos through data virtualization
  - Typical in presence of many company branches

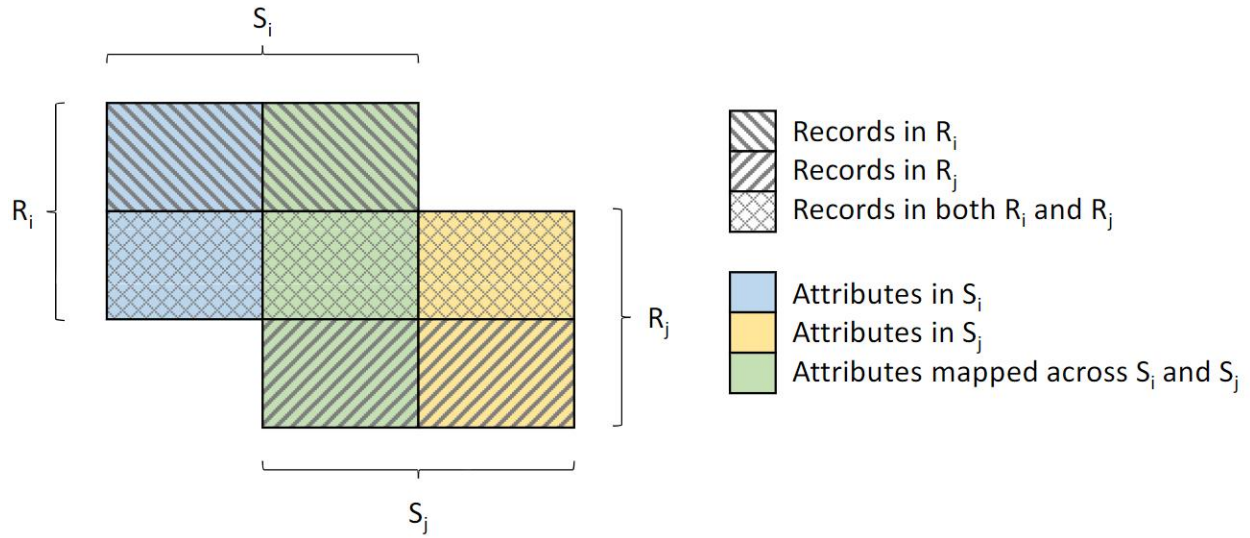
Solutions?

- Traditional integration approach
- Solve inconsistencies on-the-fly

Data fusion

Merge operator

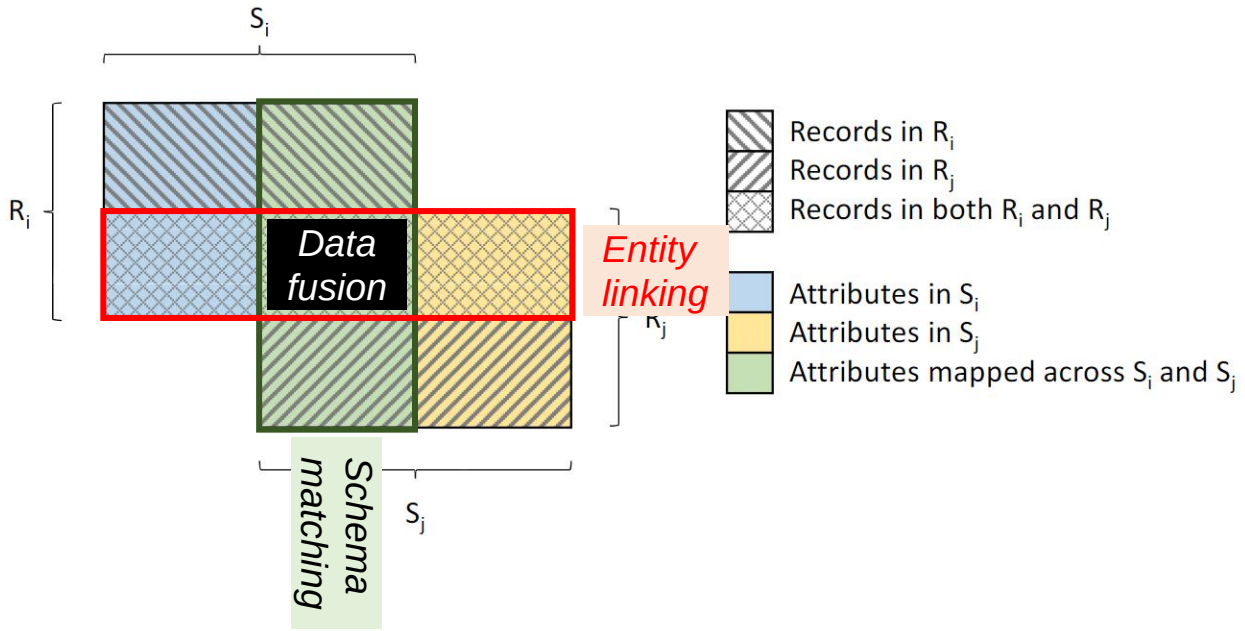
- Originally introduced as “full outer join merge”
  - Naumann, F., Freytag, J. C., & Leser, U. (2004). Completeness of integrated information sources. *Information Systems*, *29*(7), 583-615.
- Aims to keep as much information as possible when joining the records of two schemas
  - Avoid any loss of records
  - Resolve mappings by providing transcoded output
  - Resolving conflicts whenever necessary



Data fusion

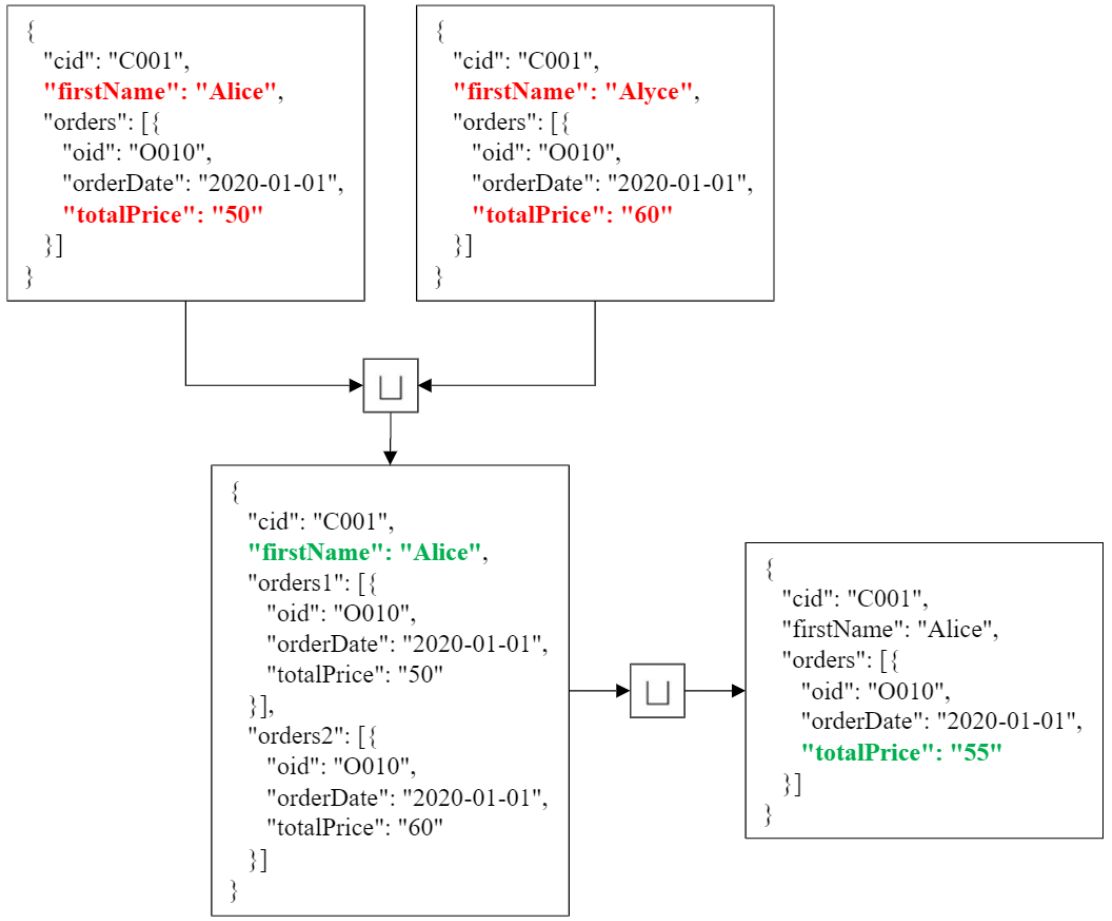
Merge operator

- Originally introduced as “full outer join merge”
  - Naumann, F., Freytag, J. C., & Leser, U. (2004). Completeness of integrated information sources. *Information Systems*, *29*(7), 583-615.
- Aims to keep as much information as possible when joining the records of two schemas
  - Avoid any loss of records
  - Resolve mappings by providing transcoded output
  - Resolving conflicts whenever necessary



Data fusion

Merge operator



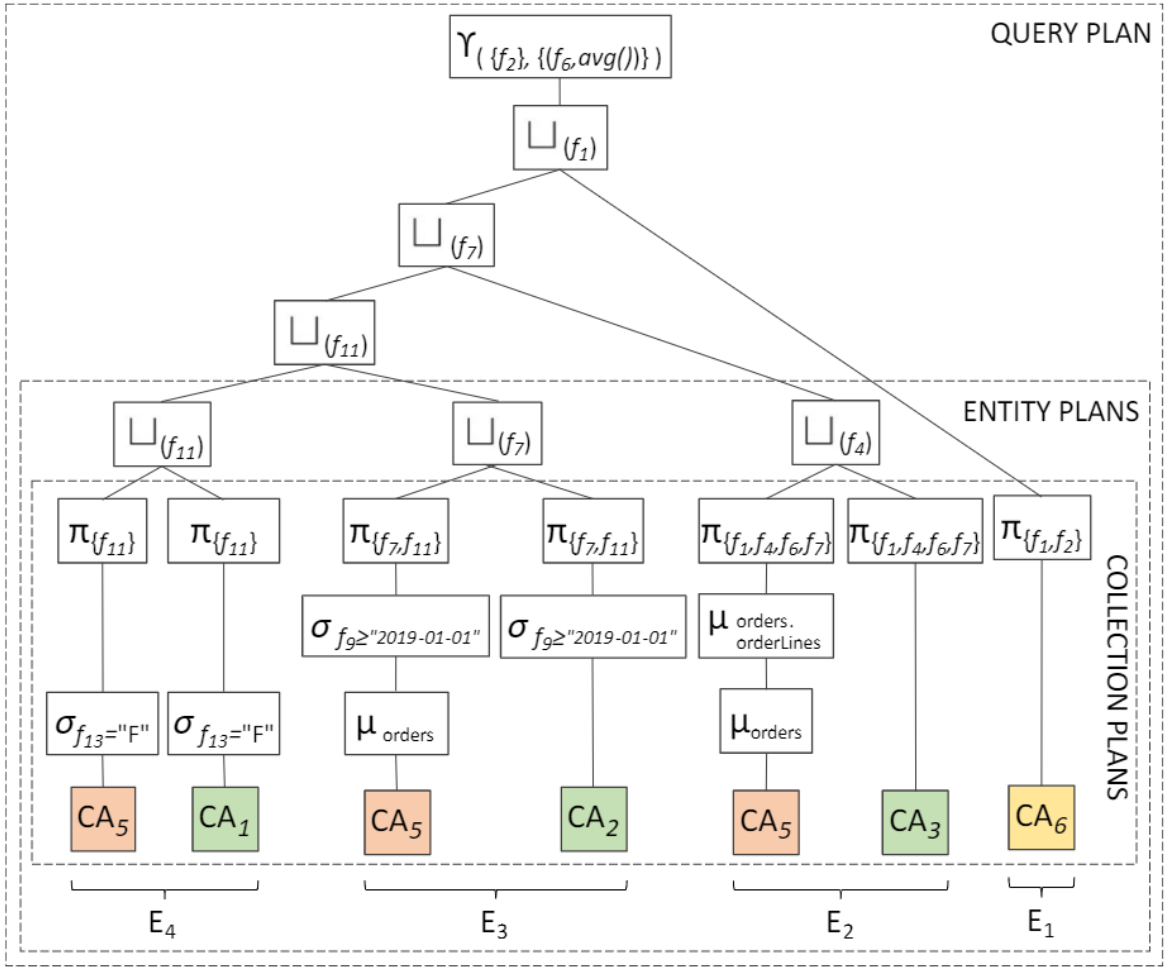
On-the-fly data fusion

Merge operator in a query plan

- Take the data from heterogeneous sources (in different colors)
- Extract records of the single entites (e.g., customer, products)
- Merge each entity
- Join and produce the final result

Now we have a multistore dealing with multiple data models, schema heterogeneity, and data inconsistency

- Are we done? No!



On-the-fly data fusion

Main issue: performance

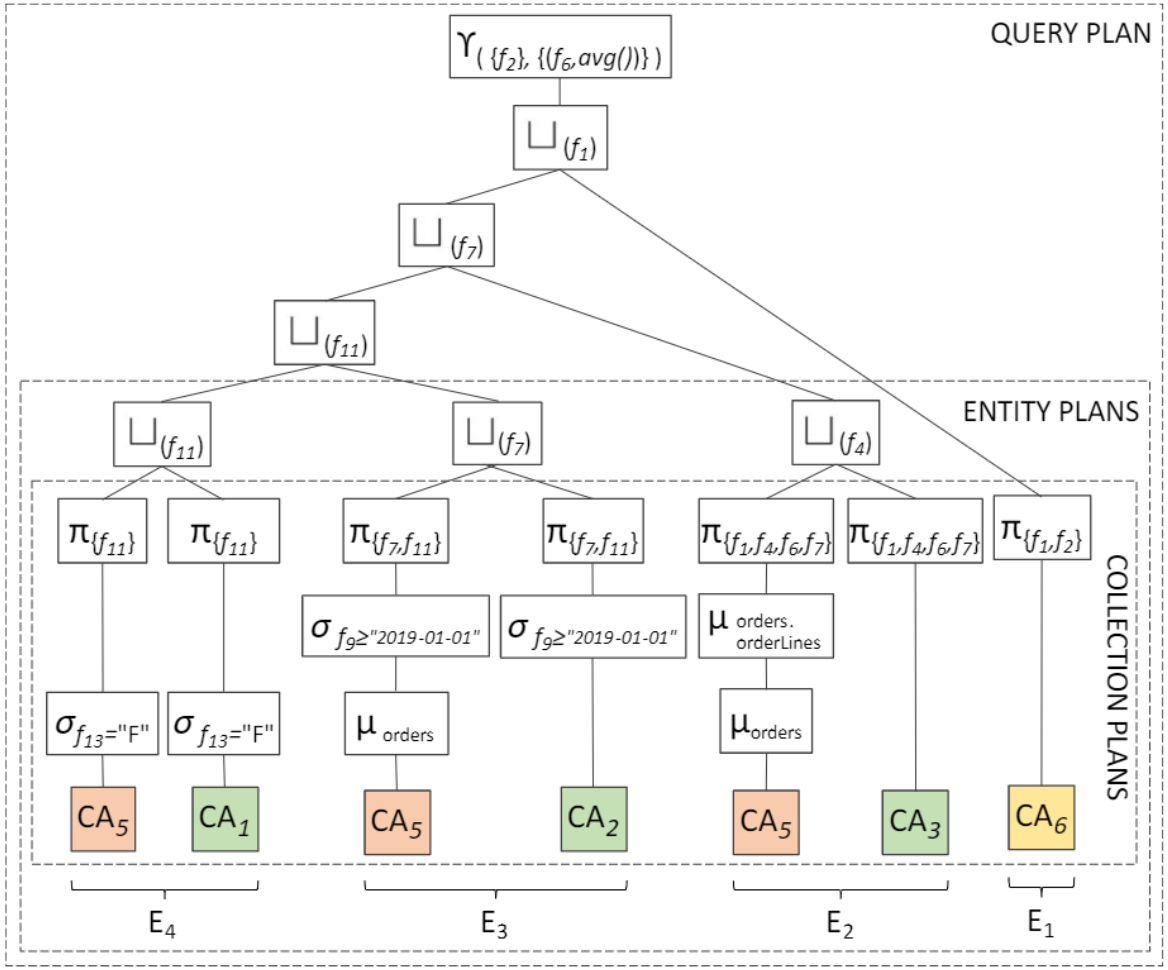
- Collections accessed more than once
- Most effort pulled to the middleware

What can we do about it?

- Exploit more the local DBMSs
- Exploit local data modelling
- Carry out multi-entity merges

Issues

- Several query plans could be devised
- Hard to find the most efficient one



Same query, several query plans

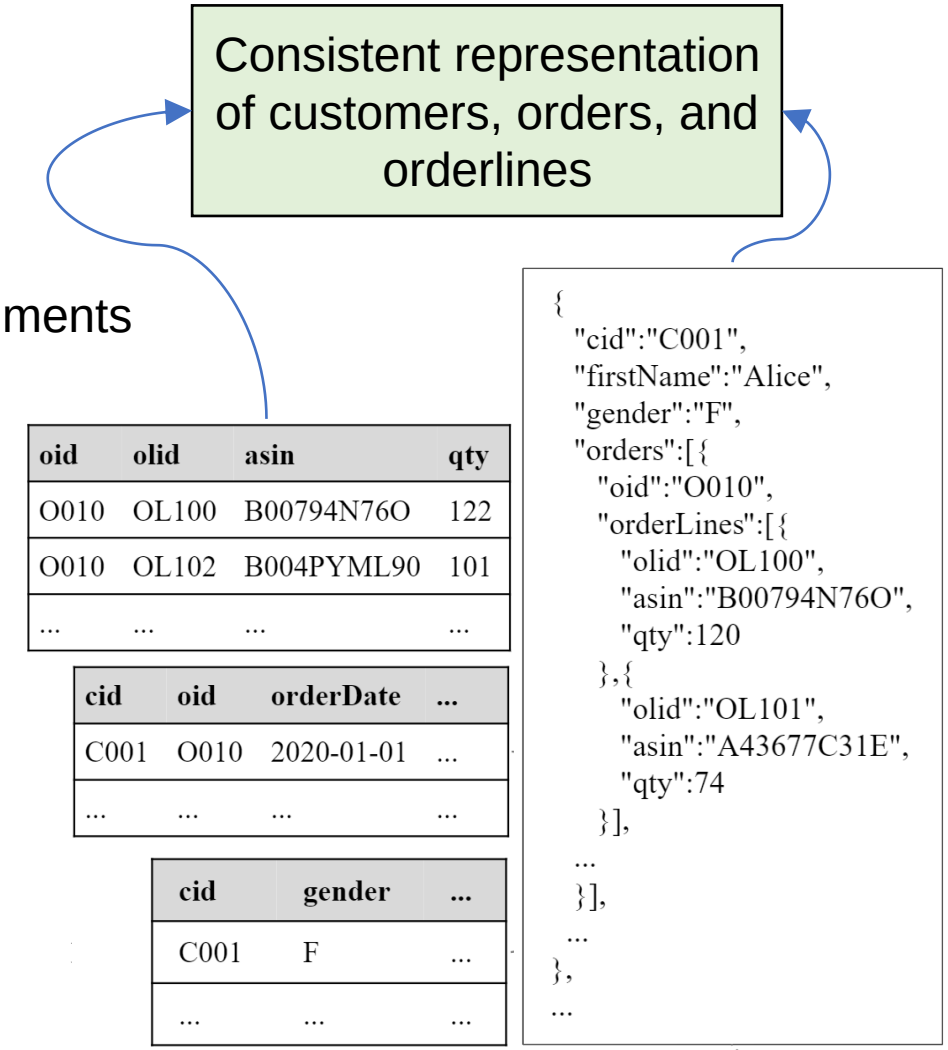
What is the most efficient solution?

- Single-entity merge and subsequent joins
- Nest relational data and multi-merge with documents
- Join relational data and multi-merge with flattened documents

Depends on several factors

- On the capabilities of each DBMS/middleware
- On the presence of indexes and statistics
- On the resources available to each DBMS/middleware
- On the number of records involved on each side

..which can change over time



Logical optimization

Logical rules to transform a query plan into a more efficent one

- Predicate push-down: applying selection predicates as close to the source as possible
  - Not always feasible (e.g., in presence of inconsistent data)
- Column pruning: extracting the only attributes relevant for the query
  - Not for granted when writing a custom query language
- Join sequence reordering: changing the order to do binary joins
  - Not so easy when merges are involved as well
  - Not so easy when data comes from different sources

Cost modelling

Cost-based evaluation of different plans

- White-box cost modelling
  - Associate theoretical formulas to each query operators, then build up the cost of a query by summing the cost of each operation
  - Cost can be determined in terms of disk I/O, CPU, network
  - Requires an enormous effort to effectively model the many factors that contribute to query costs in a complex and heterogeneous environment like a multistore
- Black-box cost modelling
  - Hide the behavior of an execution engine within a black-box, where the known information is mostly limited to the issued queries and the given response times
  - Cost is determined in terms of time
  - Easily adapts to evolving environments
  - Suffers from cold-start

Cost modelling

White-box cost modelling example

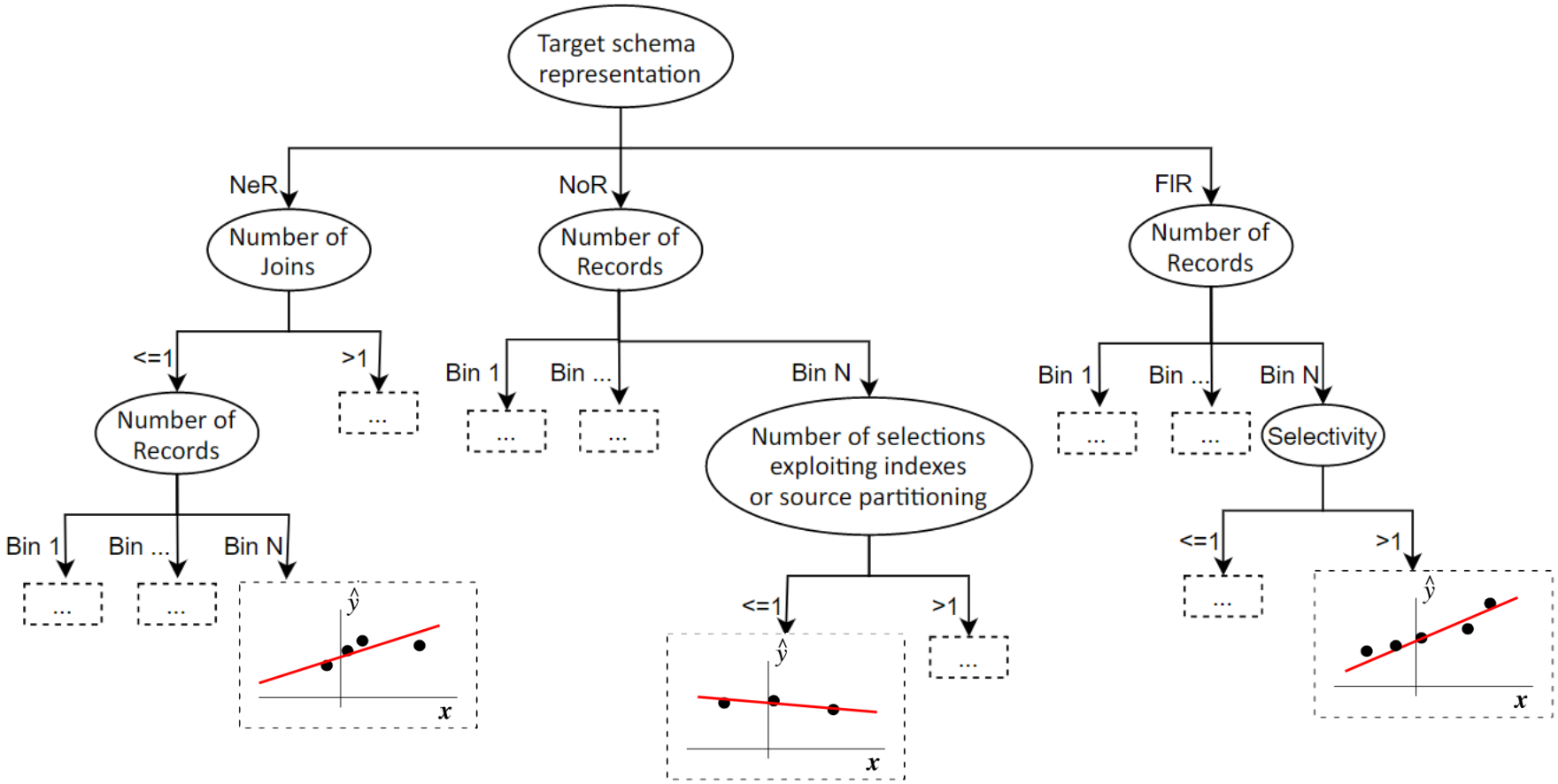
Parameter/Function	Description
$NR(C)$	Number of records in C
$NNR(C)$	If C is nested, number of nested records
$Len(C)$	Average byte length of a record in C
DPS	Size of a disk page
$PT(C)$	Number of partitions into which records in C are organized
NB	Number of memory buffers
$NP(C)$	Number of disk pages occupied by C: $\lceil \frac{NR(C) \cdot Len(C)}{DPS} \rceil$
$Part(C, nPart)$	Number of disk pages occupied by one of $nPart$ partitions of C

Operation	Description	Estimated cost	Sup.
$CA(C)$	Collection access	$NP(C)$	RMCS
$\pi(C)$	Projection	$NP(C)$	RMCS
$\gamma(C)$	Aggregation	$Sort(C) + NP(C)$	RM*S
$v(C)$	Nest	$Sort(C) + NP(C)$	RM-S
$\mu(C)$	Unnest	$NP(C)$	RM-S
$\overline{\mu}(C)$	Simult. unnest	$NR(C) \cdot SM(Part(C, \lceil \frac{NNR(C)}{NR(C)} \rceil))$	---S
$\overline{\cup}(C)$	Array union	$NP(C)$	RM-S
$(C_1) \bowtie (C_2)$	Join	$min(NLJ(C, C'),\ SMJ(C, C'),\ HJ(C, C'))$	RM-S
$(C_1) \sqcup (C_2)$	Merge	$min(NLJ(C, C'),\ SMJ(C, C'),\ HJ(C, C'))$	RM-S
$(C_1) \cup (C_2)$	Union	$NP(C) + NP(C')$	RM-S
$Shf(C)$	Data shuffle	$3 \cdot NP(C)$	-M-S
$SM(C)$	Sort-Merge	$2 \cdot NP \cdot (\lceil log_{NB-1} NP \rceil + 1)$	RM-S
$Sort(C)$	Data sort (central.)	$SM(C)$	RM--
	Data sort (distrib.)	$Shf(R) + PT(C) \cdot SM(Part(C, PT(C)))$	---S
$HJ(C, C')$	Hybrid hash Join	$3 \cdot (NP(C) + NP(C'))$	R---
$NLJ(C, C')$	Nested Loops Join	$NP(C) + NR(C) \cdot NP(C')$	R---
		$NP(C) + NR(C) \cdot NP(C') + Unnest(C'')$	-M--
$SMJ(C, C')$	Sort-Merge Join	$Sort(C) + Sort(C') + NP(C) + NP(C')$	R--S

Golfarelli, M. (2021, August). Optimizing Execution Plans in a Multistore. In Advances in Databases and Information Systems: 25th European Conference, ADBIS 2021, Tartu, Estonia, August 24–26, 2021, Proceedings (Vol. 12843, p. 136). Springer Nature.

Cost modelling

Black-box cost modelling example



(work in progress)

Polyglot persistence - Conclusions

Two main issues

- Query performance
  - Improving caching techniques
  - Improving cost model effectiveness
- Data integration
  - Improving effectiveness (ever-lasting direction)
  - Improving efficiency on big data scales
    - Reduce the number of comparison, introduce approximation
    - Parallelize computation
    - Exploit additional information (e.g., temporal entity resolution models)
  - Human-in-the-loop (pay-as-you-go, crowdsourcing)
  - Frictionless integration within a big data platform
    - A metadata challenge that we explore tomorrow!

Mandreoli, F., & Montangero, M. (2019). Dealing with data heterogeneity in a data fusion perspective: models, methodologies, and algorithms. In *Data Handling in Science and Technology* (Vol. 31, pp. 235-270). Elsevier.

The metadata challenge

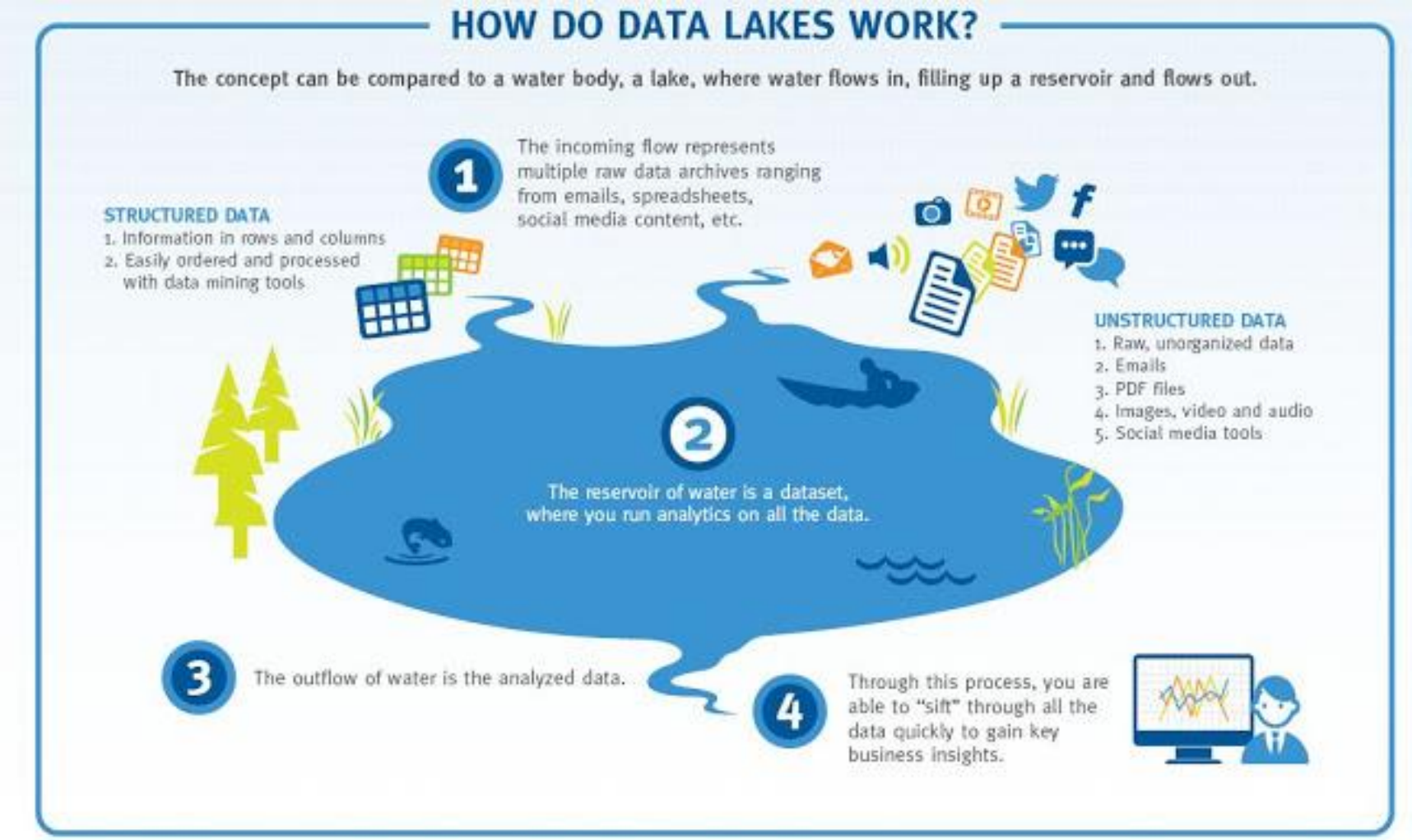
What we are going to do

Definitions: from the data lake to the data platform

Define challenges

Discuss current solutions and future directions

Data lake


HOW DO DATA LAKES WORK?
The concept can be compared to a water body, a lake, where water flows in, filling up a reservoir and flows out.
1
The incoming flow represents multiple raw data archives ranging from emails, spreadsheets, social media content, etc.
STRUCTURED DATA
1. Information in rows and columns
2. Easily ordered and processed with data mining tools
UNSTRUCTURED DATA
1. Raw, unorganized data
2. Emails
3. PDF files
4. Images, video and audio
5. Social media tools
2
The reservoir of water is a dataset, where you run analytics on all the data.
3
The outflow of water is the analyzed data.
4
Through this process, you are able to "sift" through all the data quickly to gain key business insights.

Data lake

"*If you think of a datamart as a store of bottled water – cleansed and packaged and structured for easy consumption – the data lake is a large body of water in a more natural state.*"

- James Dixon, 2010

"*A large storage system for raw, heterogeneous data, fed by multiple data sources, and that allows users to explore, extract and analyze the data.*"

- Sawadogo, P., Darmont, J. On data lake architectures and metadata management. J Intell Inf Syst 56, 97–120 (2021)

"*A data lake is a central location that holds a large amount of data in its native, raw format.*"

- Databricks, 2021

Data lake

The data lake started with the Apache Hadoop movement, using the Hadoop File System (HDFS) for cheap storage

- Schema-on-read architecture
- Agility of storing any data at low cost
- Eludes the problems of quality and governance

A two-tier data lake + warehouse architecture is dominant in the industry

- HDFS replaced by cloud data lakes (e.g., S3, ADLS, GCS)
- Data lake data directly accessible to a wide range of analytics engines
- A subset of data is ETLed to a data warehouse for important decision support and BI apps

Armbrust, M., Ghodsi, A., Xin, R., & Zaharia, M. (2021). Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. CIDR.

Data lake

Downsides of data lakes

- Security
  - All the data is stored and managed as files
  - No fine-grained access control on the contents of files, but only coarse-grained access governing who can access what files or directories
- Quality
  - Hard to prevent data corruption and manage schema changes
  - Challenging to ensure atomic operations when writing a group of files
  - No roll-back mechanism
- Query performance
  - Formats are not optimized for fast access

It is often said that the *lake* easily turns into a *swamp*

Data lakehouse

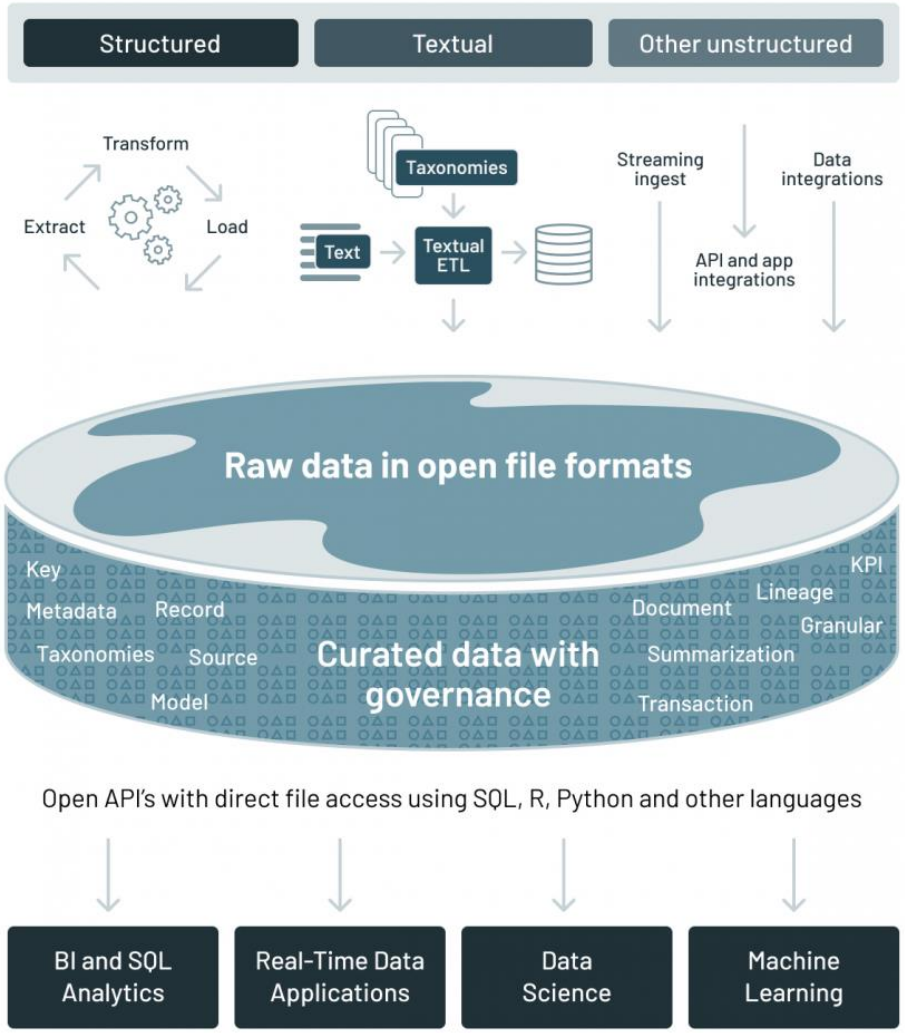
The data lakehouse enables storing all your data once in a data lake and efficiently doing AI and BI on that data directly at a massive scale

- ACID transaction support
- Schema enforcement and governance
  - Support to schema enforcement and evolution, supporting DW schema architectures
  - Reason about data integrity
  - Robust governance and auditing mechanisms
- BI support
- Storage decoupled from compute
  - Ability to scale to many more concurrent users and larger data sizes
- Open storage formats (e.g., Parquet)
- Support for diverse data types ranging from unstructured to structured data
- Support for diverse workloads (e.g., data science, ML, SQL, analytics)
- End-to-end streaming

https://databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html

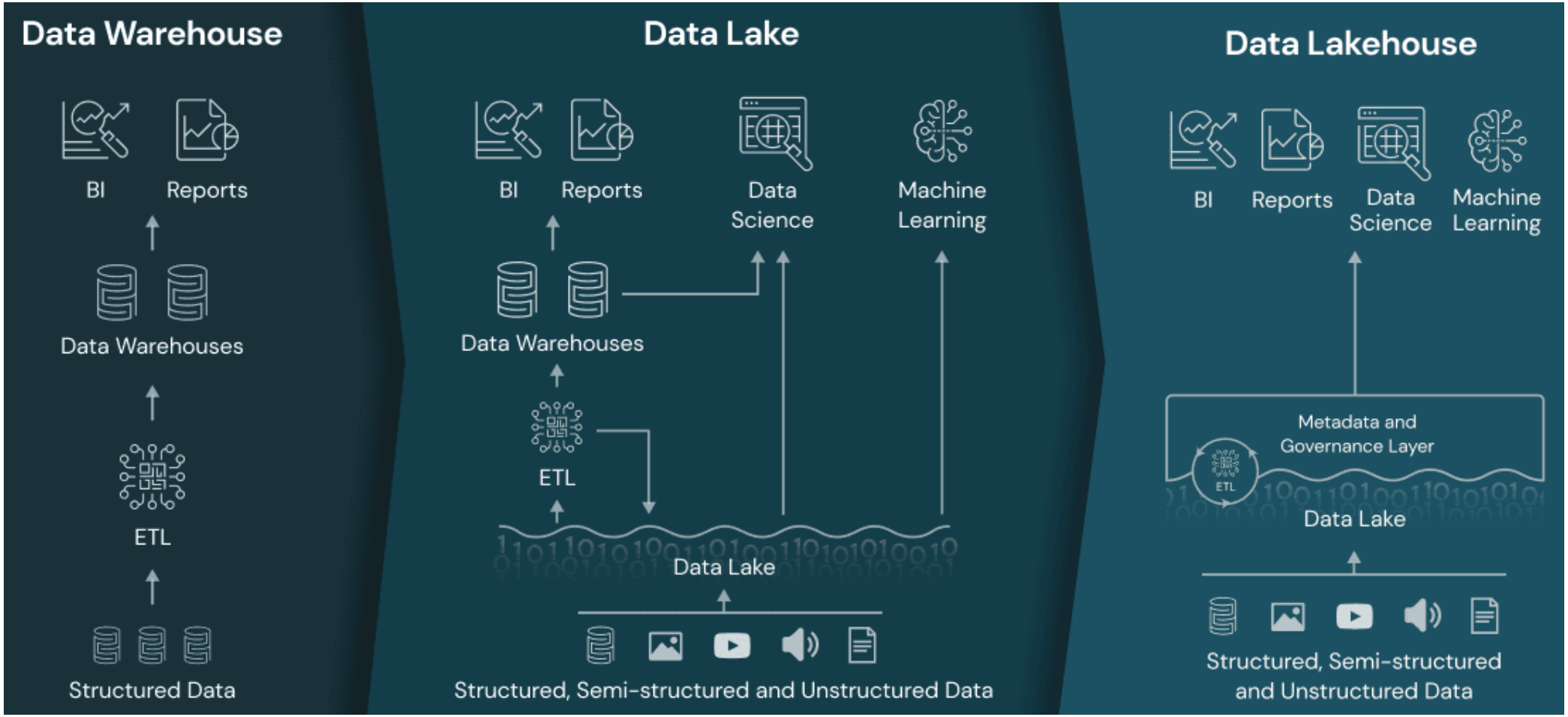
Data lakehouse

	Data warehouse	Data lake	Data lakehouse
Data format	Closed, proprietary format	**Open format**	Open format
Types of data	Structured data, with limited support for semi-structured data	**All types**: Structured data, semi-structured data, textual data, unstructured (raw) data	All types: Structured data, semi-structured data, textual data, unstructured (raw) data
Data access	SQL-only, no direct access to file	**Open APIs** for direct access to files with SQL, R, Python and other languages	Open APIs for direct access to files with SQL, R, Python and other languages
Reliability	**High quality**, reliable data with ACID transactions	Low quality, data swamp	High quality, reliable data with ACID transactions
Governance and security	**Fine-grained** security and governance for row/columnar level for tables	Poor governance as security needs to be applied to files	Fine-grained security and governance for row/columnar level for tables
Performance	**High**	Low	High
Scalability	Scaling becomes exponentially more expensive	**Scales** to hold any amount of data at low cost, regardless of type	Scales to hold any amount of data at low cost, regardless of type
Use case support	Limited to BI, SQL applications and decision support	Limited to machine learning	One data architecture for BI, SQL and machine learning

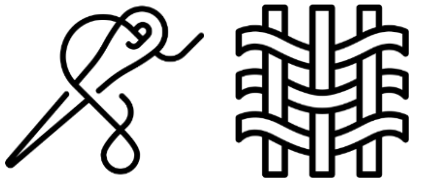


https://databricks.com/blog/2021/05/19/evolution-to-the-data-lakehouse.html

Data lakehouse



Data fabric



Data fabric enables frictionless access and sharing of data in a distributed data environment

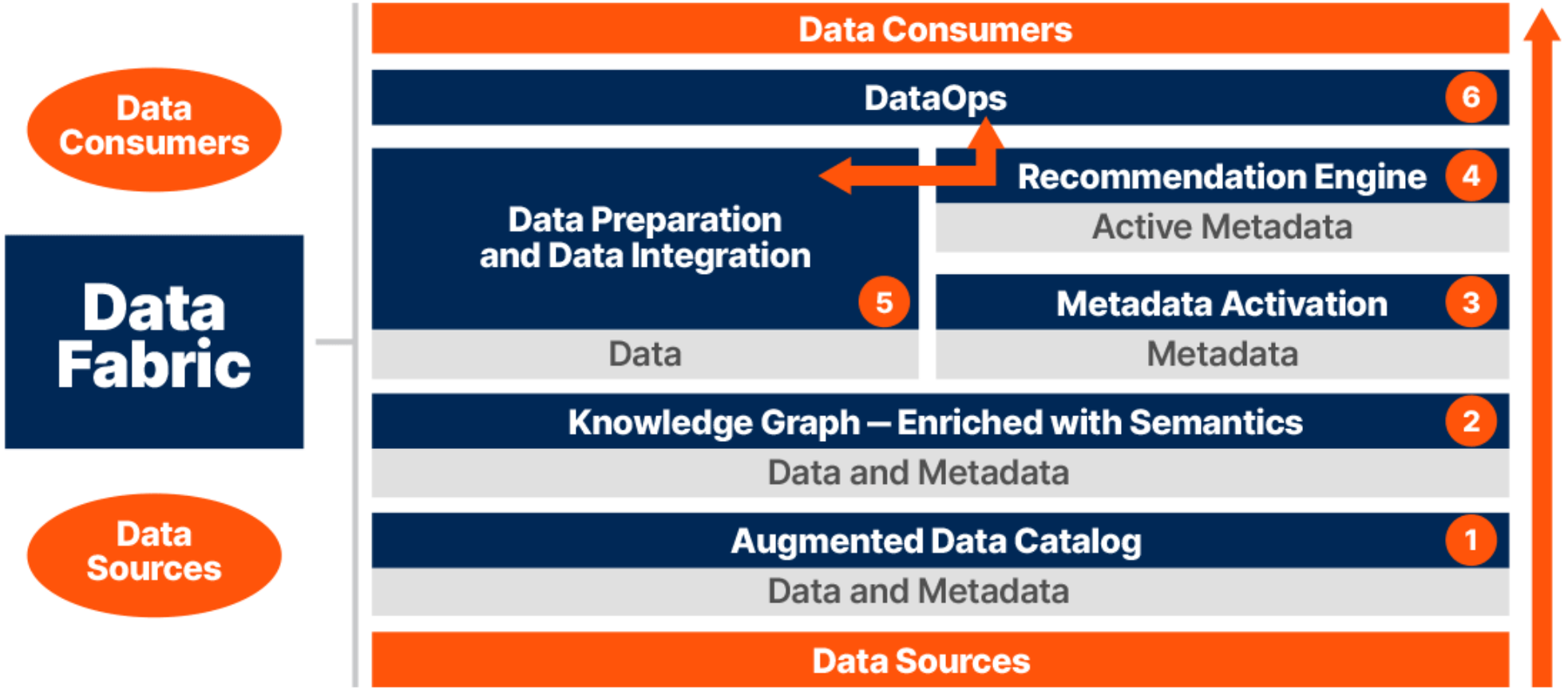
- It enables a single and consistent data management framework, which allows seamless data access and processing by design across otherwise siloed storage
- Leverages both human and machine capabilities to access data in place or support its consolidation where appropriate
- Continuously identifies and connects data from disparate applications to discover unique, business-relevant relationships between the available data points

It is a unified architecture with an integrated set of technologies and services

- Designed to deliver integrated and enriched data – at the right time, in the right method, and to the right data consumer – in support of both operational and analytical workloads
- Combines key data management technologies – such as data catalog, data governance, data integration, data pipelining, and data orchestration

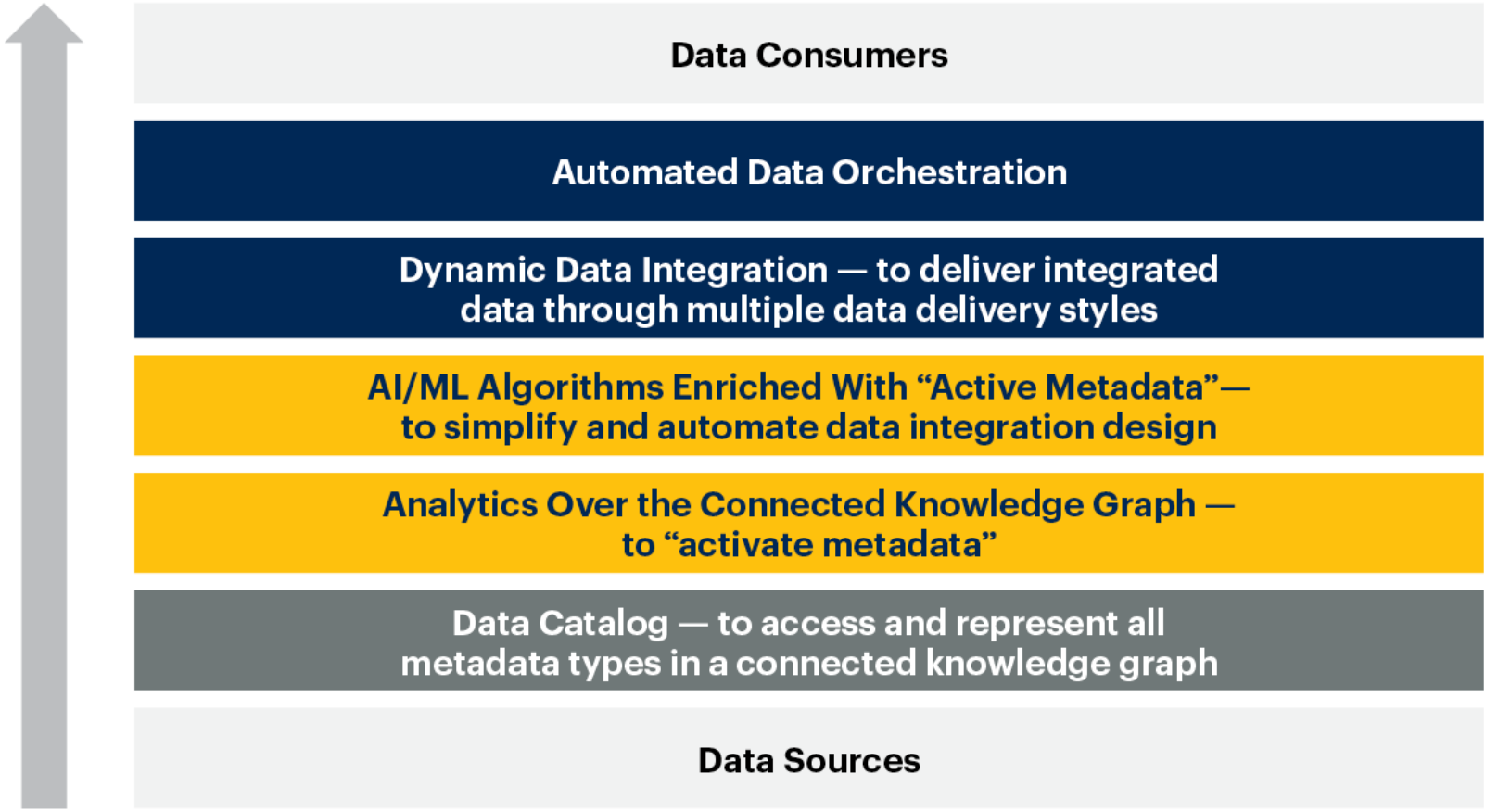
Gartner, 2019 https://www.gartner.com/en/newsroom/press-releases/2019-02-18-gartner-identifies-top-10-data-and-analytics-technolo
Gartner, 2021 https://www.gartner.com/smarterwithgartner/data-fabric-architecture-is-key-to-modernizing-data-management-and-integration
K2View Whitepaper: What is a Data Fabric? The Complete Guide, 2021

Data fabric



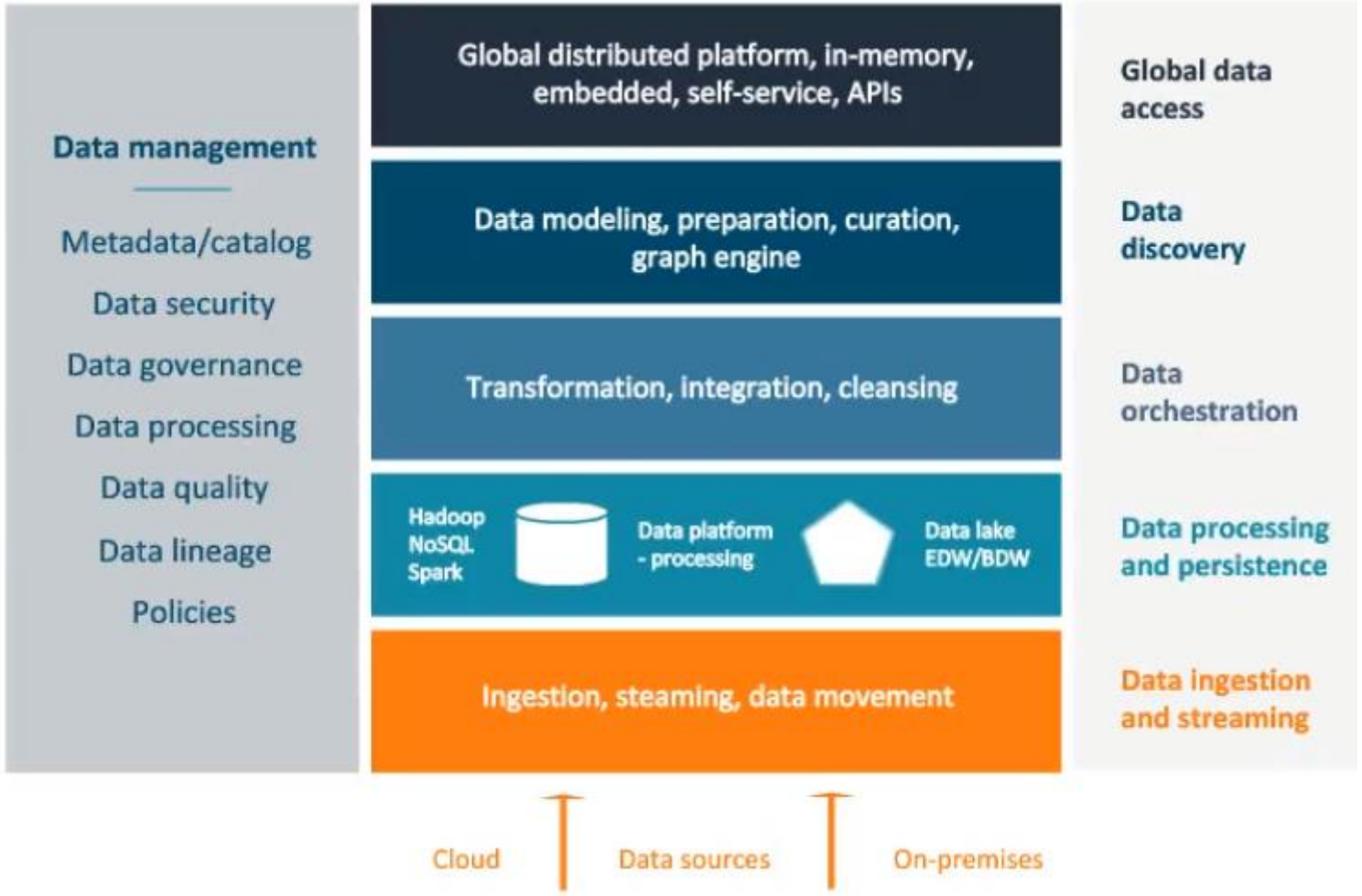
https://www.irion-edm.com/data-management-insights/gartner-data-summit-irion-representative-vendor-for-data-fabric-technology/

Data fabric



Gartner, 2021 https://www.gartner.com/smarterwithgartner/data-fabric-architecture-is-key-to-modernizing-data-management-and-integration

Data fabric


Data management
Metadata/catalog
Data security
Data governance
Data processing
Data quality
Data lineage
Policies
Global distributed platform, in-memory, embedded, self-service, APIs
Data modeling, preparation, curation, graph engine
Transformation, integration, cleansing
Hadoop NoSQL Spark
Data platform - processing
Data lake EDW/BDW
Ingestion, steaming, data movement
Global data access
Data discovery
Data orchestration
Data processing and persistence
Data ingestion and streaming
Cloud
Data sources
On-premises

Data fabric

It is a design concept

- It optimizes data management by automating repetitive tasks
- According to Gartner estimates, 25% of data management vendors will provide a complete framework for data fabric by 2024 – up from 5% today

Gartner, 2021 https://www.gartner.com/smarterwithgartner/data-fabric-architecture-is-key-to-modernizing-data-management-and-integration

K2View, 2021 https://www.k2view.com/top-data-fabric-vendors

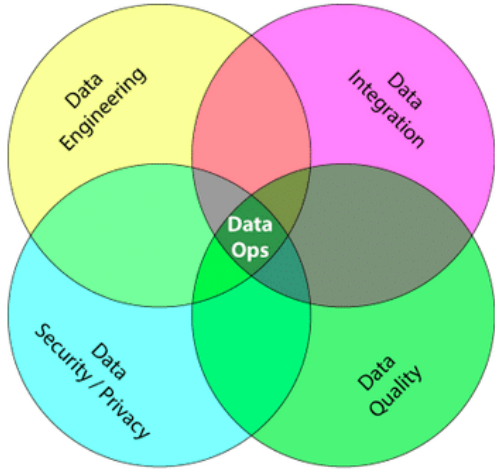
DataOps

From DevOps to DataOps

- “A collaborative data management practice focused on improving the communication, integration and automation of data flows between data managers and data consumers across an organization”
- Data analytics improved in terms of velocity, quality, predictability and scale of software engineering and deployment

Some key rules

- Establish progress and performance measurements at every stage
- Automate as many stages of the data flow as possible
- Establish governance discipline (*governance-as-code*)
- Design process for growth and extensibility



Gartner, 2020 https://www.gartner.com/smarterwithgartner/how-dataops-amplifies-data-and-analytics-business-value
Andy Palmer, 2015 https://www.tamr.com/blog/from-devops-to-dataops-by-andy-palmer/
William Vorhies, 2017 https://www.datasciencecentral.com/profiles/blogs/dataops-it-s-a-secret

Data mesh

An intentionally designed distributed data architecture, under centralized governance and standardization for interoperability, enabled by a shared and harmonized self-serve data infrastructure

- Domain-oriented decentralized data ownership
  - Decentralization and distribution of responsibility to people who are closest to the data, in order to support continuous change and scalability
  - Each domain exposes its own op/analytical APIs
- Data as a product (*quantum*)
  - Products must be discoverable, addressable, trustworthy, self-describing, secure
- Self-serve data infrastructure as a platform
  - High-level abstraction of infrastructure to provision and manage the lifecycle of data products
- Federated computational governance
  - A governance model that embraces decentralization and domain self-sovereignty, interoperability through global standardization, a dynamic topology, automated execution of decisions by the platform

Zhamak Dehghani, 2019 https://martinfowler.com/articles/data-monolith-to-mesh.html
Zhamak Dehghani, 2020 https://martinfowler.com/articles/data-mesh-principles.html

Data mesh vs Data fabric

A data fabric and a data mesh both provide an architecture framework to access data across multiple technologies and platforms

- Data fabric
  - Attempts to centralize and coordinate data management
  - Tackles the complexity of data and metadata in a smart way that works well together
- Data mesh
  - Emphasis on decentralization and data domain autonomy
  - Focuses on organizational change; it is more about people and process

They are concepts, not things

- They are *not* mutually exclusive
- They are architectural frameworks, not architectures
  - The frameworks must be adapted and customized to your needs, data, processes, and terminology

Alex Woodie, 2021 https://www.datanami.com/2021/10/25/data-mesh-vs-data-fabric-understanding-the-differences/
Dave Wells, 2021 https://www.eckerson.com/articles/data-architecture-complex-vs-complicated

Data platform

A data platform is a complete solution for ingesting, processing, analyzing, and presenting the data generated by the systems, processes, and infrastructures of the modern digital organization

- It is a single platform that can be used across an entire organization, preventing silos and providing actionable insights based on a holistic view of the organization's data

Data fabric and mesh are different architectural frameworks for a data platform

Metadata challenges

Knowledge representation

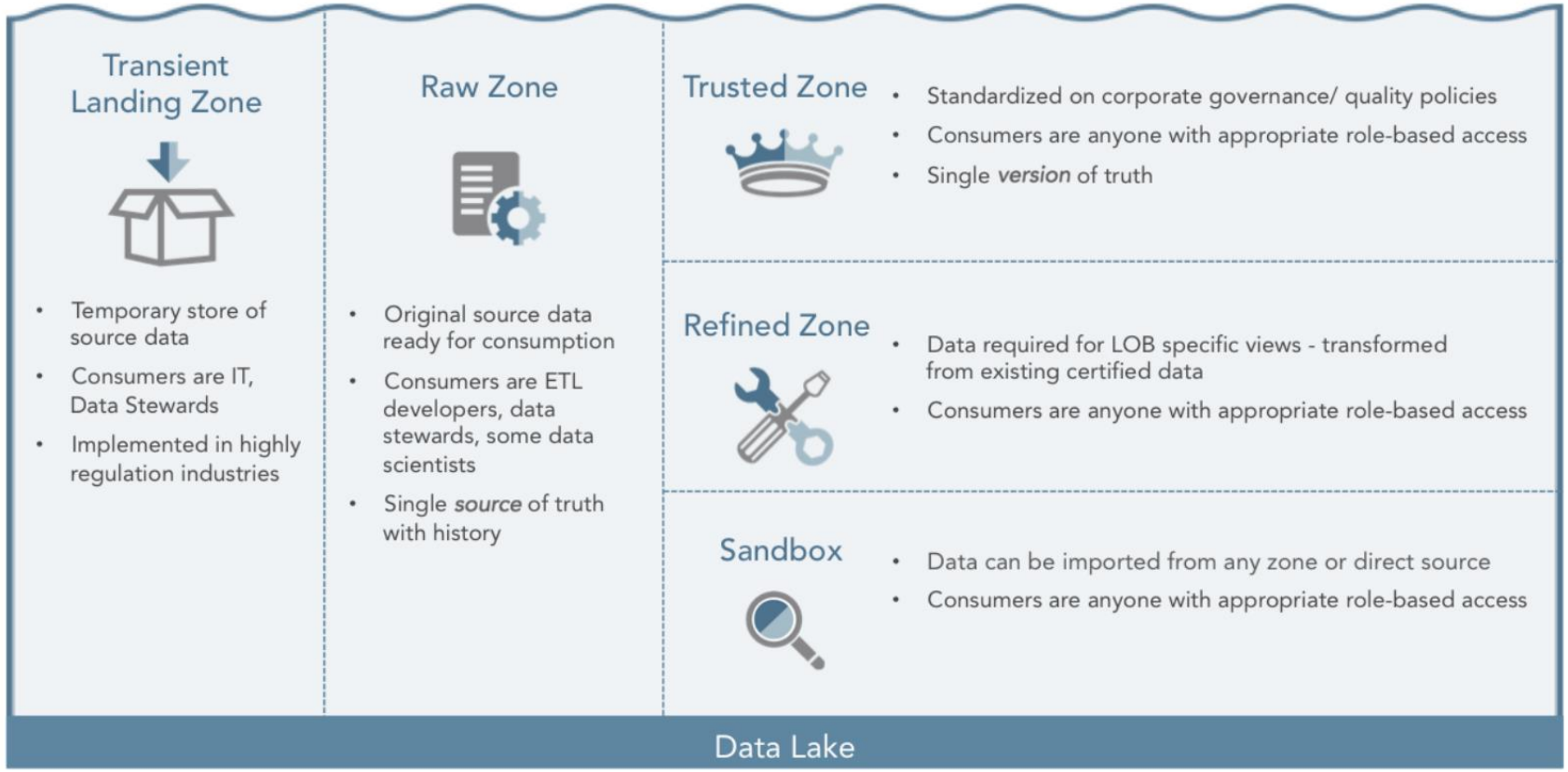
- Which metadata must be captured
- How should metadata be organized

Knowledge exploitation

- Which features do metadata enable

Knowledge representation

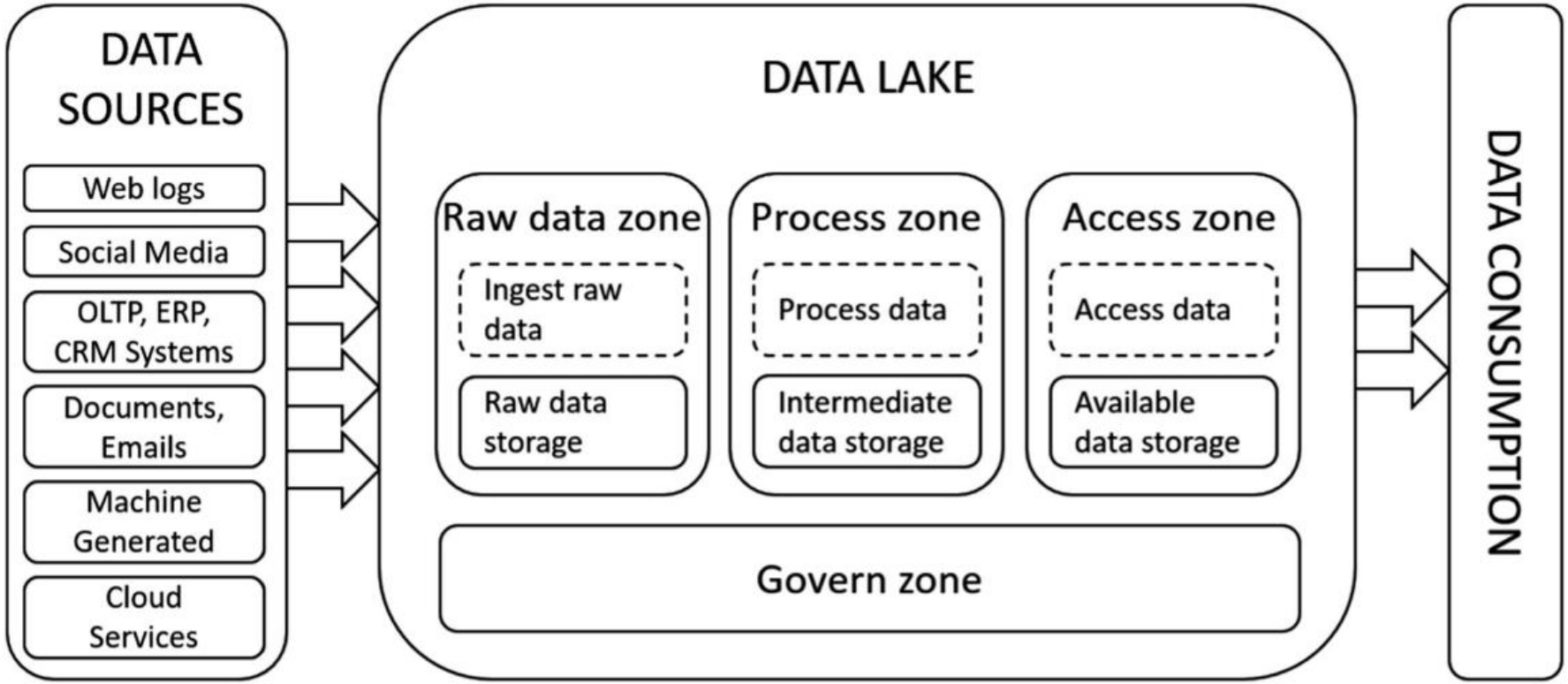
A first layer of metadata: structuring the data lake



A. LaPlante, B. Sharma, Architecting Data Lakes, O'Reilly Media, Sebastopol, 2018.

Knowledge representation

A first layer of metadata: structuring the data lake



F. Ravat, Y. Zhao, Data lakes: Trends and perspectives, in: Proc. DEXA, Linz, Austria, 2019, pp. 304–313.

Knowledge representation

A classification of functionalities

- Semantic enrichment
  - Generating a description of the context of data, e.g., with tags, to make them more interpretable and understandable
- Data indexing
  - Data structures to retrieve datasets based on specific characteristics (keywords or patterns)
- Link generation and conservation
  - Detecting similarity relationships or integrating preexisting links between datasets
- Data polymorphism
  - Storing multiple representations of the same data to avoid repeating preprocessings and speed up analyses
- Data versioning
  - Support data changes while conserving previous states
- Usage tracking
  - Records the interactions between users and the data

Sawadogo, P. N., Scholly, E., Favre, C., Ferey, E., Loudcher, S., & Darmont, J. (2019, September). Metadata systems for data lakes: models and features. In European conference on advances in databases and information systems (pp. 440-451). Springer, Cham.

Knowledge representation

A classification of metadata

- **Technical** metadata
  - Capture the form and structure of each dataset
  - E.g.: type of data (text, JSON, Avro); structure of the data (the fields and their types)
- **Operational** metadata
  - Capture lineage, quality, profile, and provenance of the data
  - E.g.: source and target locations of data, size, number of records, and lineage
- **Business** metadata
  - Captures what it all means to the user
  - E.g.: business names, descriptions, tags, quality, and masking rules for privacy

A. LaPlante, B. Sharma, Architecting Data Lakes, O'Reilly Media, Sebastopol, 2018.

Knowledge representation

Another classification of metadata

- **Intra-object** metadata
  - *Properties* provide a general description of an object in the form of key-value pairs
  - *Summaries and previews* provide an overview of the content or structure of an object
  - *Semantic metadata* are annotations that help understand the meaning of data
- **Inter-object** metadata
  - *Objects groupings* organize objects into collections, each object being able to belong simultaneously to several collections
  - *Similarity links* reflect the strength of the similarity between two objects
  - *Parenthood relationships* reflect the fact that an object can be the result of joining several others
- **Global** metadata
  - *Semantic resources*, i.e., knowledge bases (ontologies, taxonomies, thesauri, dictionaries) used to generate other metadata and improve analyses
  - *Indexes*, i.e., data structures that help find an object quickly
  - *Logs*, used to track user interactions with the data lake

Sawadogo, P. N., Scholly, E., Favre, C., Ferey, E., Loudcher, S., & Darmont, J. (2019, September). Metadata systems for data lakes: models and features. In European conference on advances in databases and information systems (pp. 440-451). Springer, Cham.

Knowledge representation

Table 1: Features provided by data lake metadata systems

System	Type	SE	DI	LG	DP	DV	UT
SPAR (Fauduet and Peyrard, 2010) [10]	♦♯	✓	✓	✓			✓
Alrehamy and Walker (2015) [1]	♦	✓		✓			
Terrizzano et al. (2015) [27]	♦	✓	✓			✓	✓
Constance (Hai et al., 2016) [11]	♦	✓	✓				
GEMMS (Quix et al., 2016) [22]	◊	✓					
CLAMS (Farid et al., 2016) [8]	♦	✓					
Suriarachchi and Plale (2016) [26]	◊				✓		✓
Singh et al. (2016) [24]	♦	✓	✓	✓	✓		
Farrugia et al. (2016) [9]	♦			✓			
GOODS (Halevy et al., 2016) [12]	♦	✓	✓	✓		✓	✓
CoreDB (Beheshti et al., 2017) [3]	♦		✓				✓
Ground (Hellerstein et al., 2017) [13]	◊♯	✓	✓			✓	✓
KAYAK (Maccioni and Torlone, 2018) [17]	♦	✓	✓	✓			
CoreKG (Beheshti et al., 2018) [4]	♦	✓	✓	✓	✓		✓
Diamantini et al. (2018) [5]	◊	✓		✓	✓		

♦ : Data lake implementation ◊ : Metadata model
♯ : Model or implementation assimilable to a data lake

Sawadogo, P. N., Scholly, E., Favre, C., Ferey, E., Loudcher, S., & Darmont, J. (2019, September). Metadata systems for data lakes: models and features. In European conference on advances in databases and information systems (pp. 440-451). Springer, Cham.

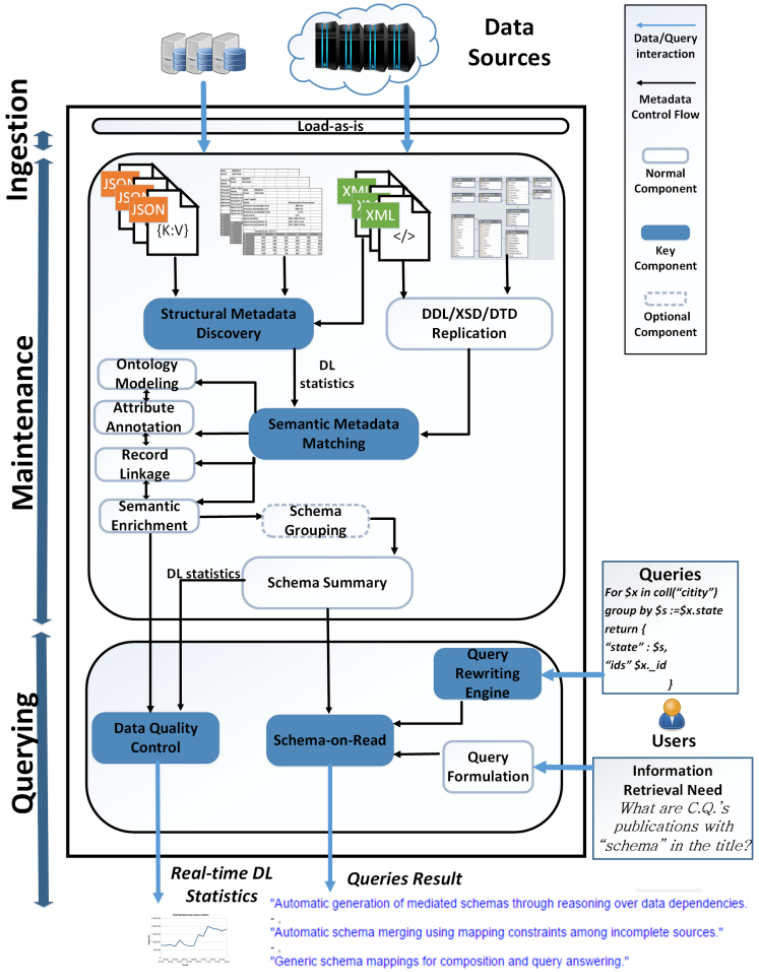
Knowledge representation

Table 1: Features provided by data lake metadata systems

System	Type	SE	DI	LG	DP	DV	UT
SPAR (Fauduet and Peyrard, 2010) [10]	♦♯	✓	✓	✓			✓
Alrehamy and Walker (2015) [1]	♦	✓		✓			
Terrizzano et al. (2015) [27]	♦	✓	✓			✓	✓
Constance (Hai et al., 2016) [11]	♦	✓	✓				
GEMMS (Quix et al., 2016) [22]	◊	✓					
CLAMS (Farid et al., 2016) [8]	♦	✓					
Suriarachchi and Plale (2016) [26]	◊				✓		✓
Singh et al. (2016) [24]	♦	✓	✓	✓	✓		
Farrugia et al. (2016) [9]	♦			✓			
GOODS (Halevy et al., 2016) [12]	♦	✓	✓	✓		✓	✓
CoreDB (Beheshti et al., 2017) [3]	♦		✓				✓
KA							✓
Diamantini et al. (2018) [5]	◊	✓		✓	✓		

Few details given on metamodel and functionalities.
No metadata collected on operations.

♦ : Data lake implementation ◊ : Metadata model
♯ : Model or implementation assimilable to a data lake



Hai, R., Geisler, S., & Quix, C. (2016, June). Constance: An intelligent data lake system. In *Proceedings of the 2016 international conference on management of data* (pp. 2097-2100).

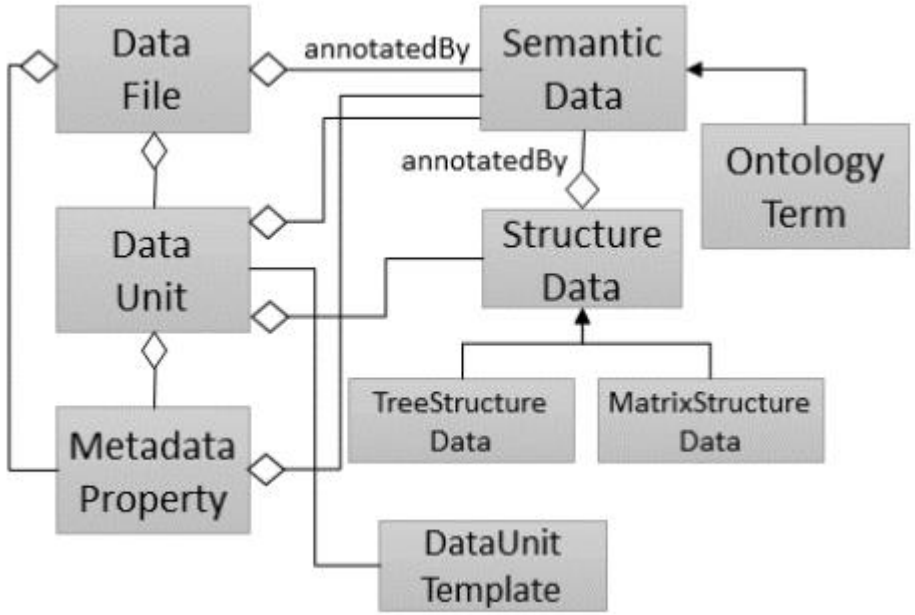
Knowledge representation

Table 1: Features provided by data lake metadata systems

System	Type	SE	DI	LG	DP	DV	UT
SPAR (Fauduet and Peyrard, 2010) [10]	♦♯	✓	✓	✓			✓
Alrehamy and Walker (2015) [1]	♦	✓		✓			
Terrizzano et al. (2015) [27]	♦	✓	✓			✓	✓
Constance (Hai et al., 2016) [11]	♦	✓	✓				
GEMMS (Quix et al., 2016) [22]	◊	✓					
CLAMS (Farid et al., 2016) [8]	♦	✓					
Suriarachchi and Plale (2016) [26]	◊				✓		✓
Singh et al. (2016) [24]	♦	✓	✓	✓	✓		
Farrugia et al. (2016) [9]	♦			✓			
GOODS (Halevy et al., 2016) [12]	♦	✓	✓	✓		✓	✓
CoreDB (Beheshti et al., 2017) [3]	♦		✓				✓
KA						✓	✓
							✓
Diamantini et al. (2018) [5]	◊	✓		✓	✓		

No discussion about the functionalities provided.
No metadata collected on operations and agents.

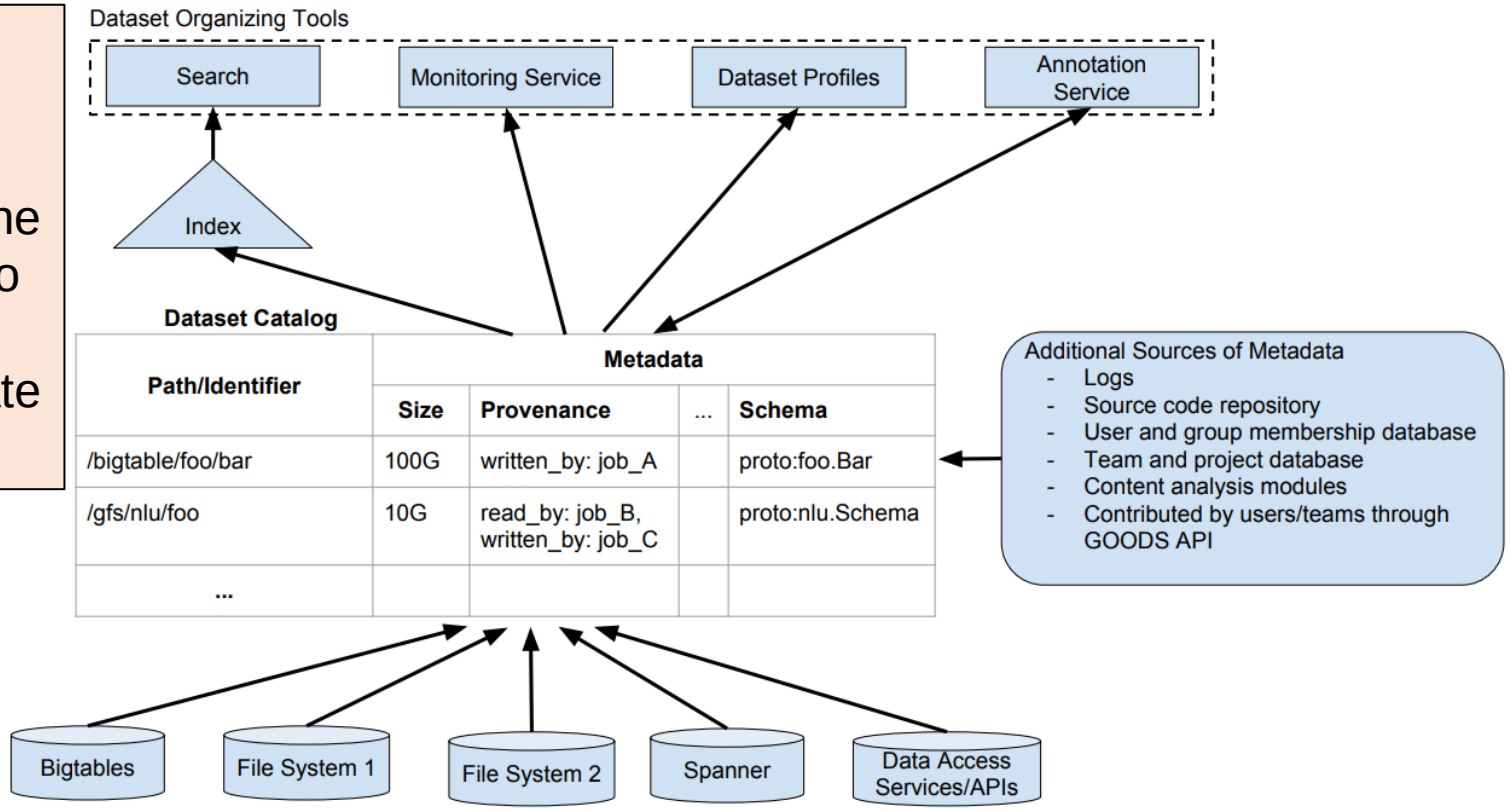
♦ : Data lake implementation ◊ : Metadata model
♯ : Model or implementation assimilable to a data lake



Quix, C., Hai, R., & Vatov, I. (2016). GEMMS: A Generic and Extensible Metadata Management System for Data Lakes. In *CAiSE forum* (Vol. 129).

Knowledge representation

Crawls Google's storage systems to extract basic metadata on datasets and their relationship with other datasets.
Performs metadata inference, e.g., to determine the schema of a non-self-describing dataset, to trace the provenance of data through a sequence of processing services, or to annotate data with their semantics.



Strictly coupled with the Google platform.
Mainly focuses on object description and searches.
No formal description of the metamodel.

Farrugia et al. (2016) [9] ♦
GOODS (Halevy et al., 2016) [12] ♦ ✓
CoreDB (Beheshti et al., 2017) [3] ♦
✓
✓
✓
✓

♯ : Model or implementation assimilable to a data lake

Halevy, A. Y., Korn, F., Noy, N. F., Olston, C., Polyzotis, N., Roy, S., & Whang, S. E. (2016). Managing Google's data lake: an overview of the Goods system. *IEEE Data Eng. Bull.*, *39*(3), 5-14.

Knowledge representation

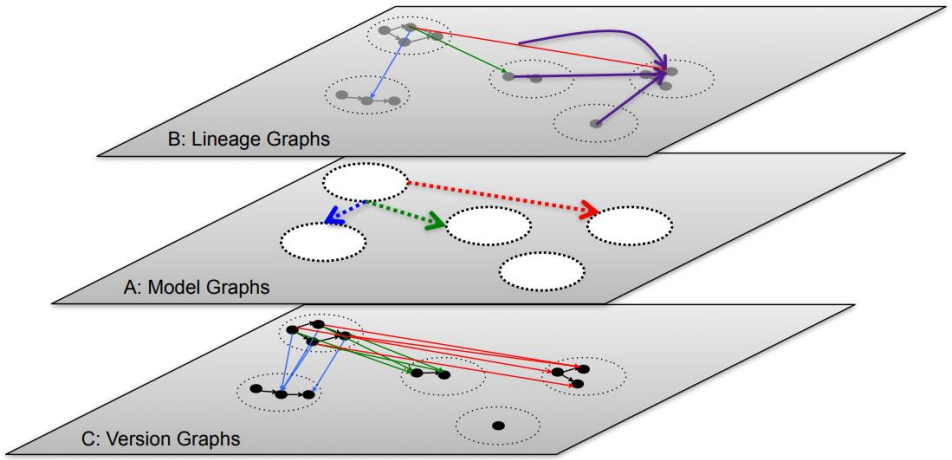
Version graphs represent data versions.
Model graphs represent application metadata, i.e., how data are interpreted for use.
Lineage graphs capture usage information.

Not enough details given to clarify which metadata are actually handled.
Functionalities are described at a high level.

Table 1: Features provided by data lake metadata systems

				DI	LG	DP	DV	UT
				✓	✓			✓
					✓			
				✓			✓	✓
				✓				
GEMMS (Quix et al., 2016) [22]	◊	✓						
						✓		✓
				✓	✓	✓		
					✓			
GOODS (Halevy et al., 2016) [12]	♦	✓	✓	✓			✓	✓
CoreDB (Beheshti et al., 2017) [3]	♦		✓					✓
Ground (Hellerstein et al., 2017) [13]	◊♯	✓	✓				✓	✓
KAYAK (Maccioni and Torlone, 2018) [17]	♦	✓	✓	✓				
CoreKG (Beheshti et al., 2018) [4]	♦	✓	✓	✓	✓			✓
Diamantini et al. (2018) [5]	◊	✓		✓	✓			

♦ : Data lake implementation ◊ : Metadata model
♯ : Model or implementation assimilable to a data lake



Hellerstein, J. M., Sreekanti, V., Gonzalez, J. E., Dalton, J., Dey, A., Nag, S., ... & Sun, E. (2017, January). Ground: A Data Context Service. In *CIDR*.

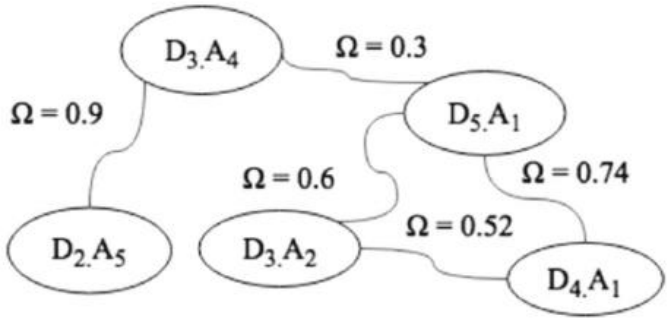
Knowledge representation

Table 1: Features provided by data lake m

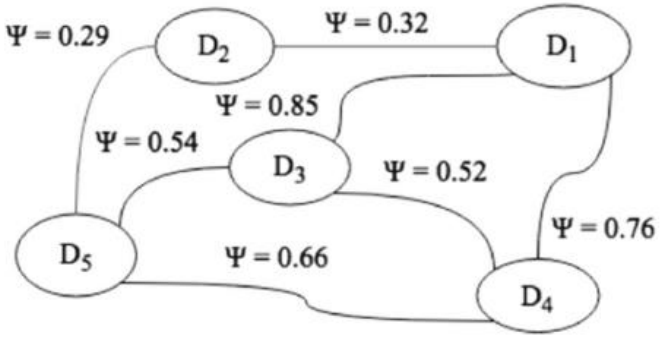
System	Type	SE	I					
SPAR (Fauduet and Peyrard, 2010) [10]	♦♯	✓						
Alrehamy and Walker (2015) [1]	♦	✓						
Terrizzano et al. (2015) [27]	♦	✓						
Constance (Hai et al., 2016) [11]	♦	✓						
GEMMS (Quix et al., 2016) [22]	◊	✓						
CLAMS (Farid et al., 2016) [8]	♦	✓						
		✓	✓	✓	✓			
				✓				
		✓	✓	✓		✓	✓	
CoreDB (Beheshti et al., 2017) [3]	♦		✓				✓	
Ground (Hellerstein et al., 2017) [13]	◊♯	✓	✓			✓	✓	
KAYAK (Maccioni and Torlone, 2018) [17]	♦	✓	✓	✓				
CoreKG (Beheshti et al., 2018) [4]	♦	✓	✓	✓	✓		✓	
Diamantini et al. (2018) [5]	◊	✓		✓	✓			

♦ : Data lake implementation ◊ : Metadata model
♯ : Model or implementation assimilable to a data lake

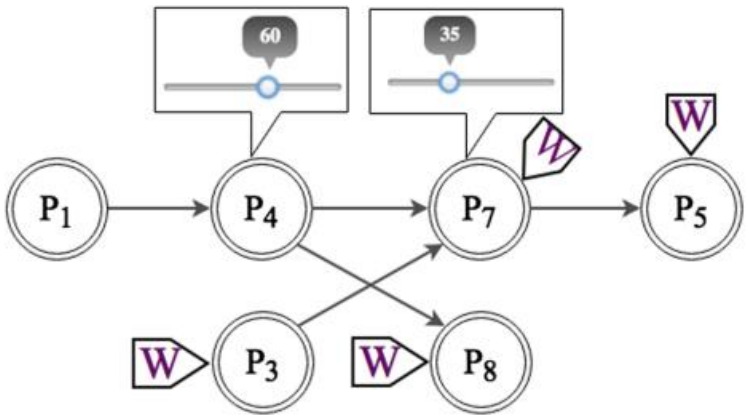
Support users in creating and optimizing the data processing pipelines.
Only goal-related metadata are collected.



(a) Affinity of attributes A_j



(b) Joinability of datasets D_i



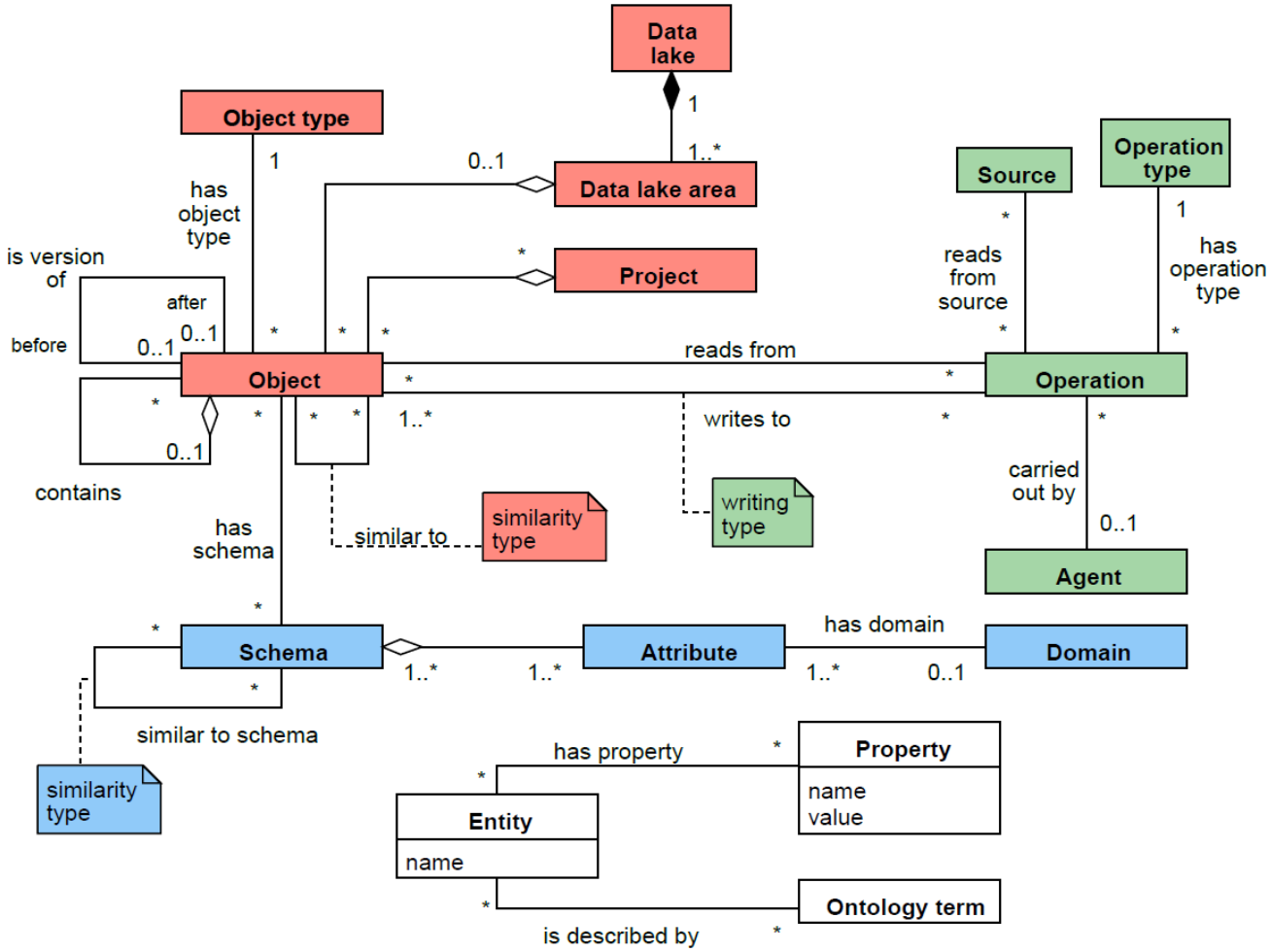
Maccioni, A., & Torlone, R. (2018, June). KAYAK: a framework for just-in-time data preparation in a data lake. In *International Conference on Advanced Information Systems Engineering* (pp. 474-489). Springer, Cham.

Knowledge representation

Technical

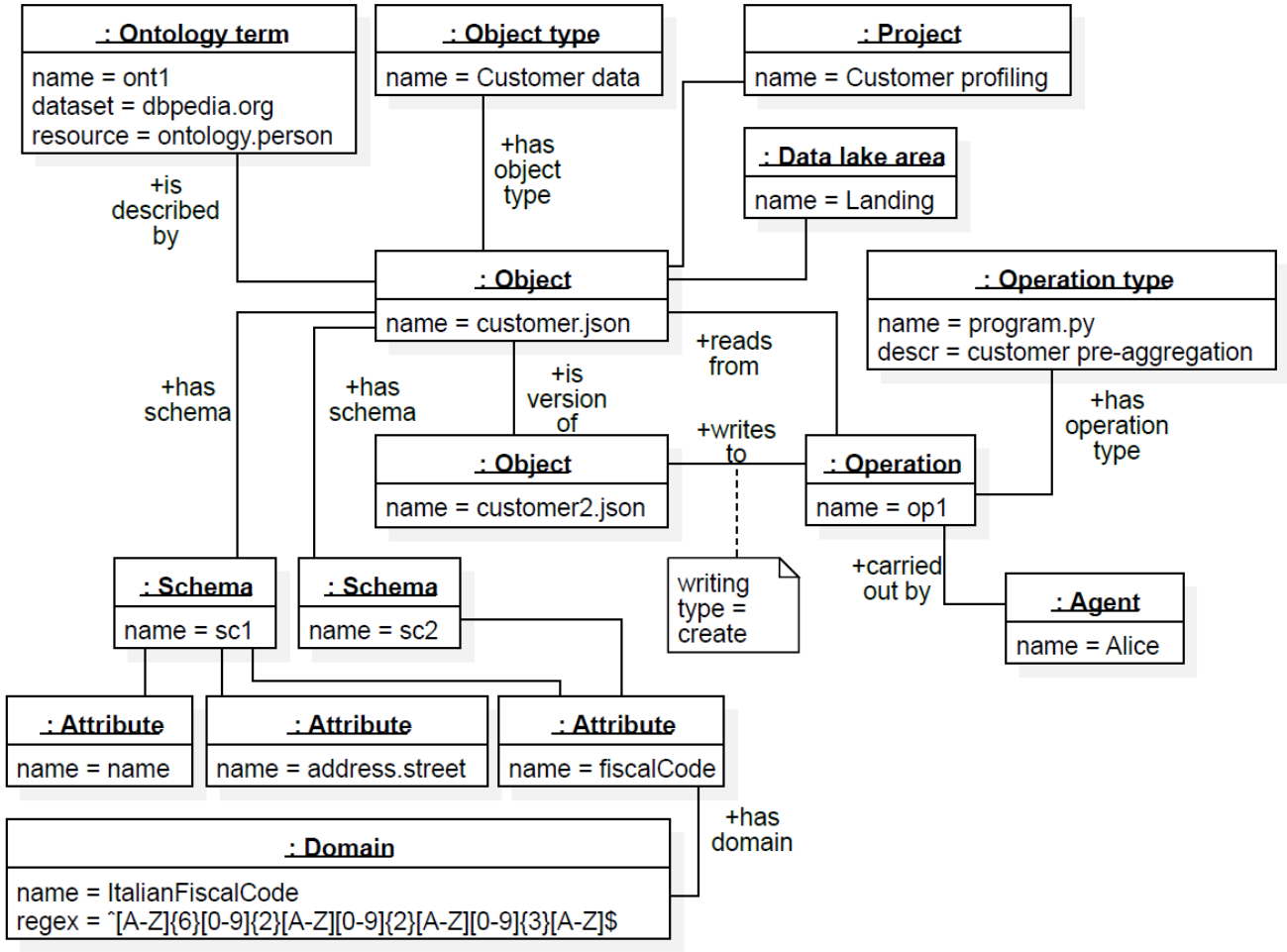
Operational

Business



Francia, M., Gallinucci, E., Golfarelli, M., Leoni, A. G., Rizzi, S., & Santolini, N. (2021). Making data platforms smarter with MOSES. Future Generation Computer Systems, 125, 299-313.

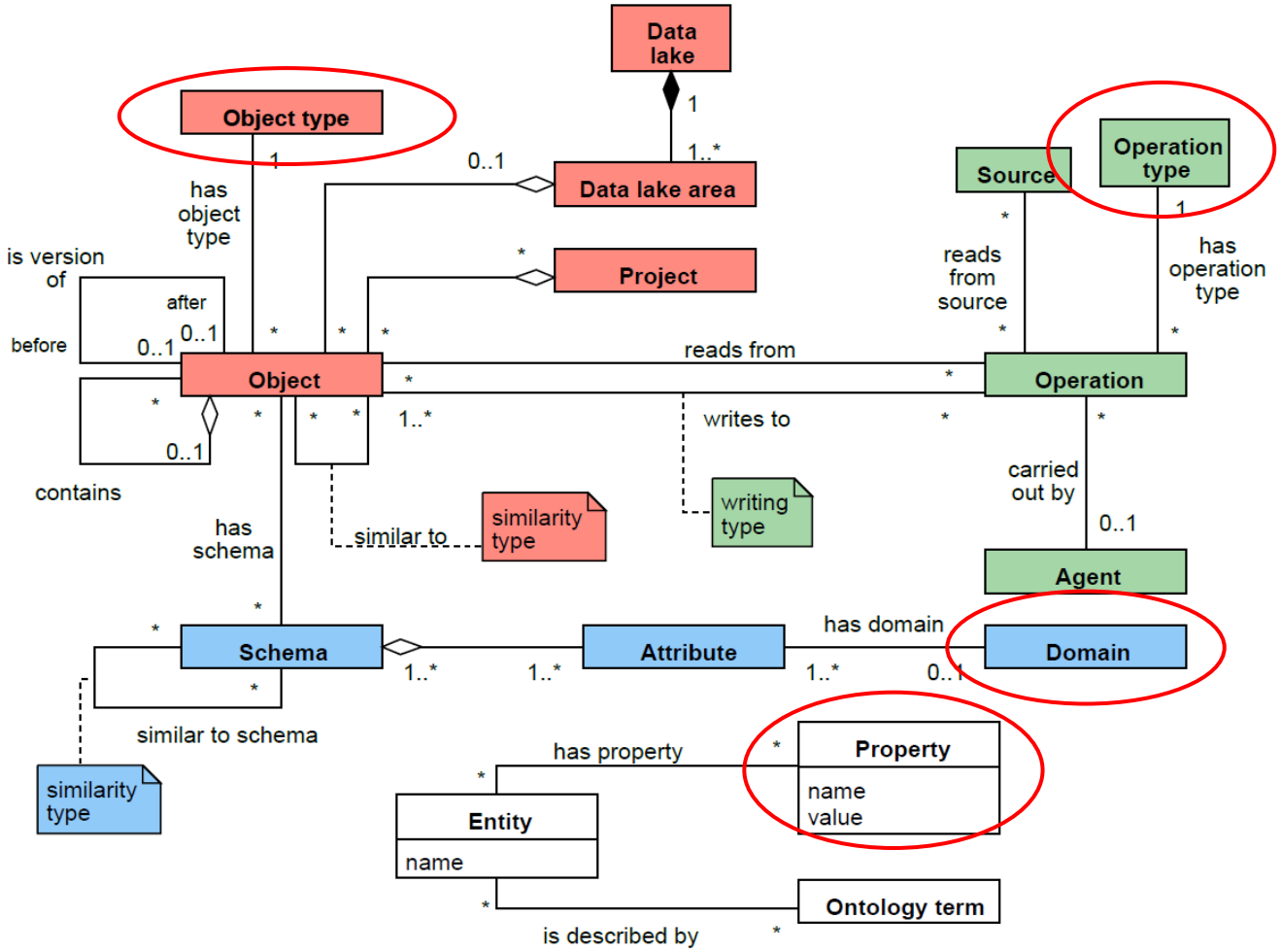
Knowledge representation



Francia, M., Gallinucci, E., Golfarelli, M., Leoni, A. G., Rizzi, S., & Santolini, N. (2021). Making data platforms smarter with MOSES. Future Generation Computer Systems, 125, 299-313.

Knowledge representation

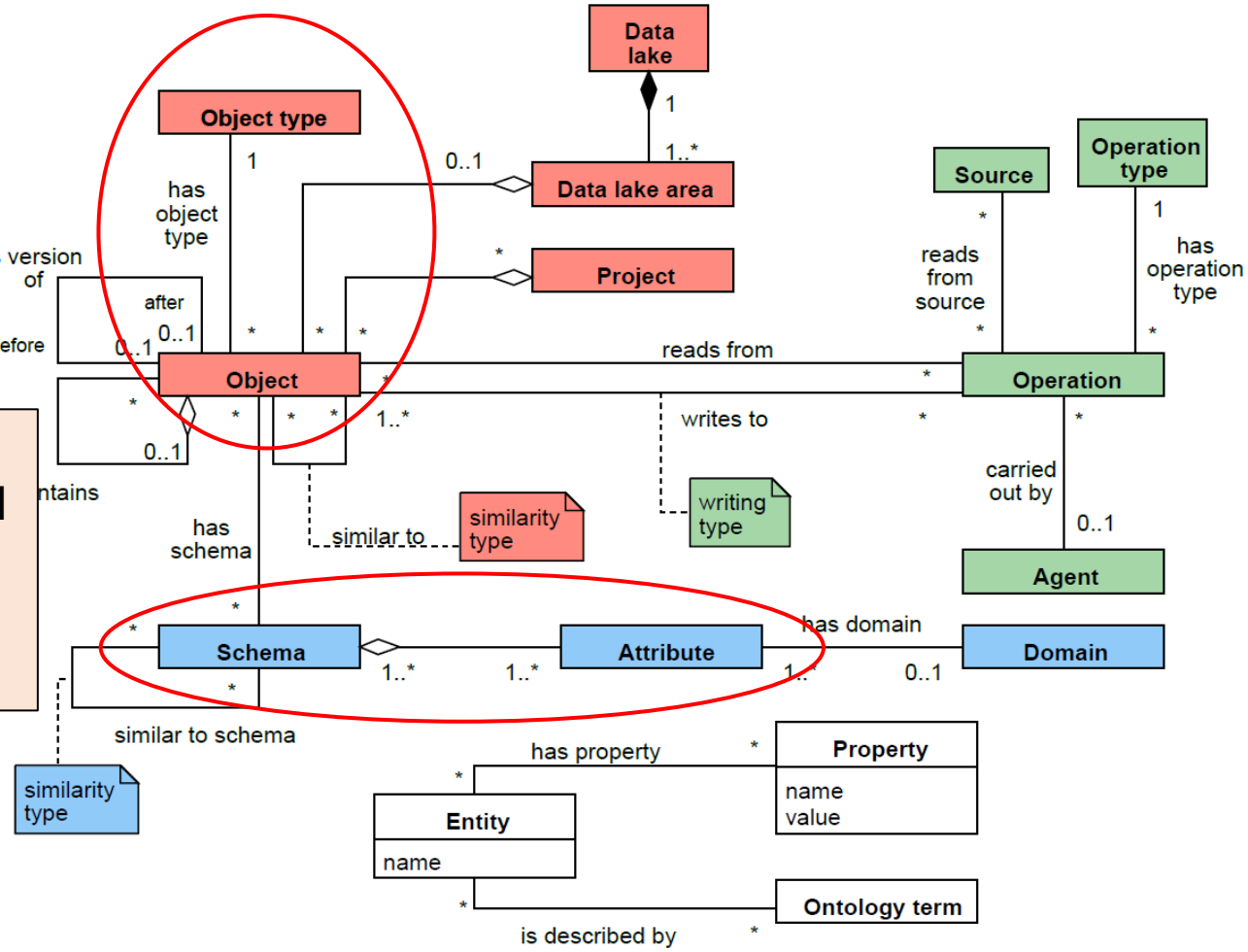
Not pre-defined
Domain-independent, extensible



Francia, M., Gallinucci, E., Golfarelli, M., Leoni, A. G., Rizzi, S., & Santolini, N. (2021). Making data platforms smarter with MOSES. Future Generation Computer Systems, 125, 299-313.

Knowledge representation

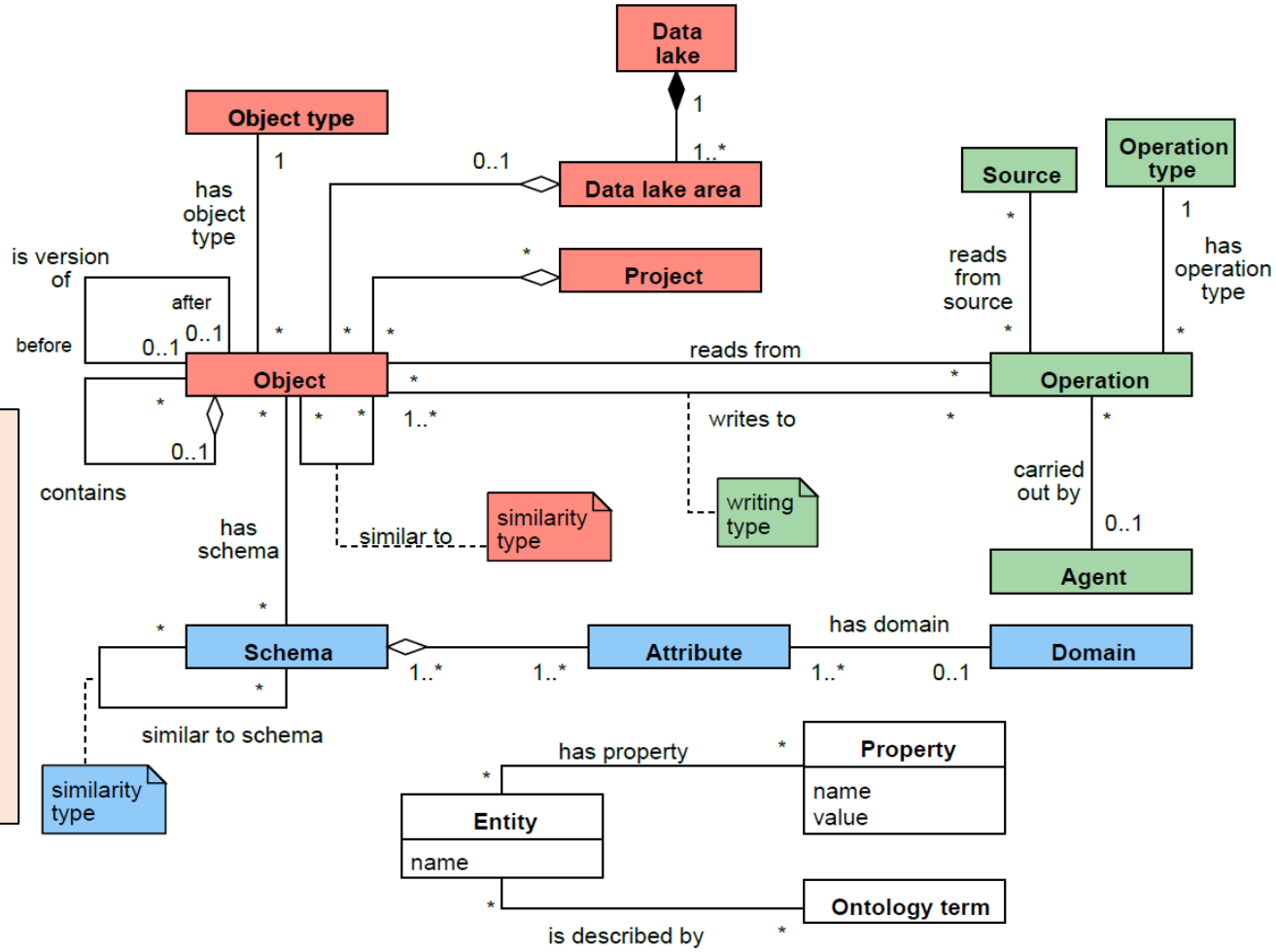
Tune the trade-off between the level of detail of the functionalities and the required computational effort



Francia, M., Gallinucci, E., Golfarelli, M., Leoni, A. G., Rizzi, S., & Santolini, N. (2021). Making data platforms smarter with MOSES. Future Generation Computer Systems, 125, 299-313.

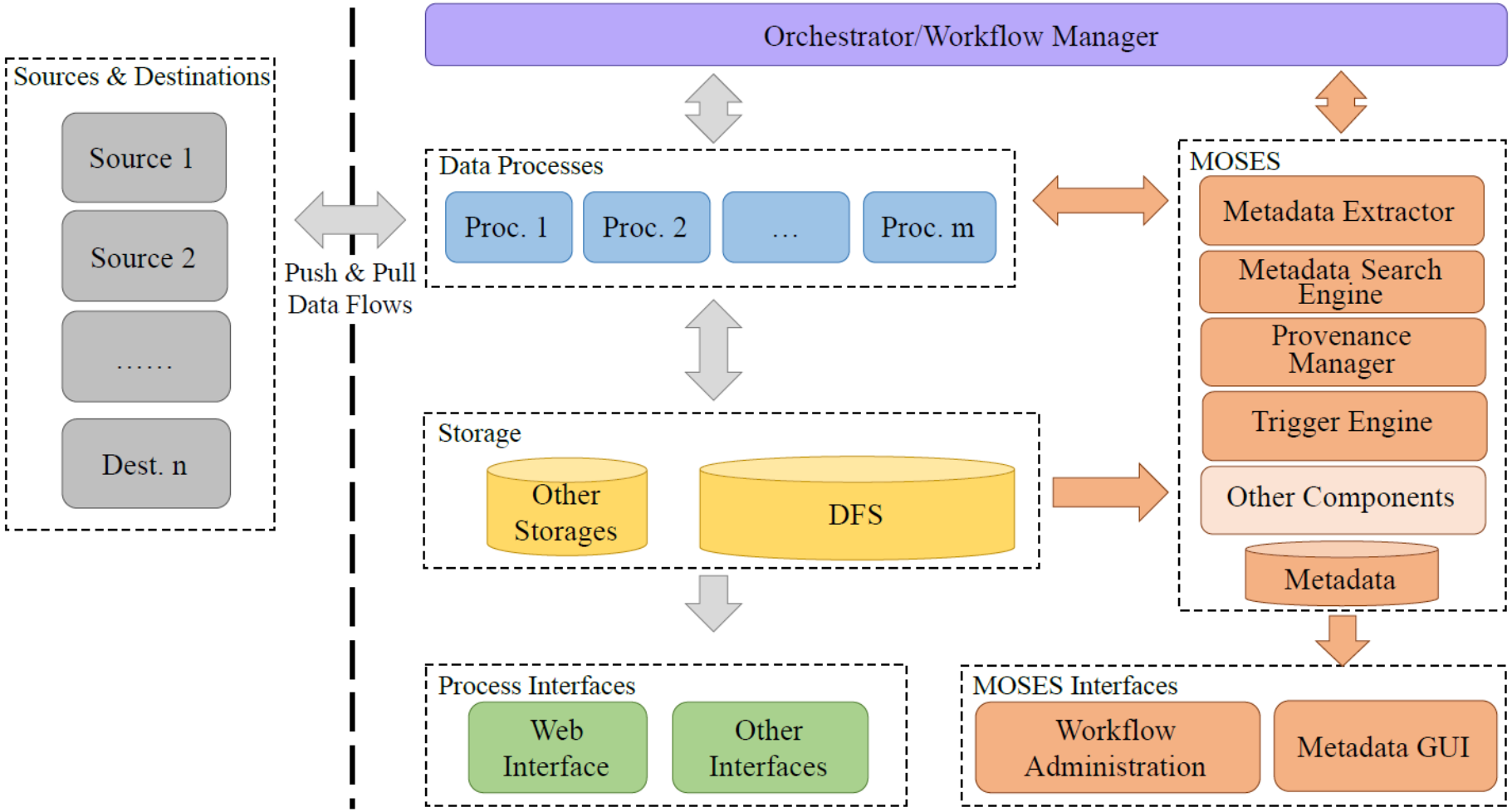
Knowledge representation

Functionalities
- ✓ Semantic enrichment
- x Data indexing
- ✓ Link generation
- ✓ Data polymorphism
- ✓ Data versioning
- ✓ Usage tracking



Francia, M., Gallinucci, E., Golfarelli, M., Leoni, A. G., Rizzi, S., & Santolini, N. (2021). Making data platforms smarter with MOSES. Future Generation Computer Systems, 125, 299-313.

Architectural reference



Knowledge exploitation

Capturing the metadata

Object profiling and search

Provenance and versioning

Orchestration support

Capturing the metadata

Pull strategy

- The system actively collects new metadata
- Requires scheduling: when does the system activate itself?
  - Event-based (CRUD)
  - Time-based
- Requires wrappers: what does the system capture?
  - Based on data type and/or application
  - A comprehensive monitoring is practically unfeasible

Push strategy

- The system passively receives new metadata
- Requires an API layer
- Mandatory for operational metadata

Object profiling and search

Discoverability is a key requirement for data platforms

- Simple searches to let users locate “known” information
- Data exploration to let users uncover “unknown” information
- Common goal: identification and description of Objects

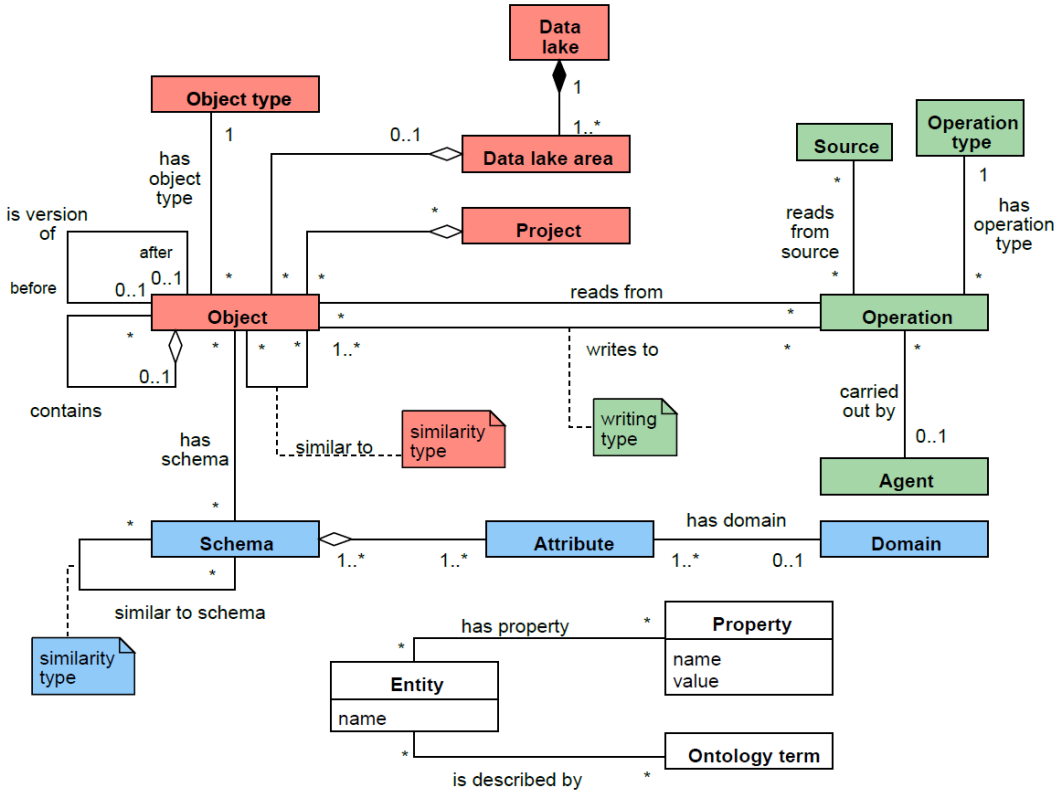
Two levels of querying

- Metadata level (most important)
- Data level (can be coupled with the first one)

Object profiling and search

Basic search

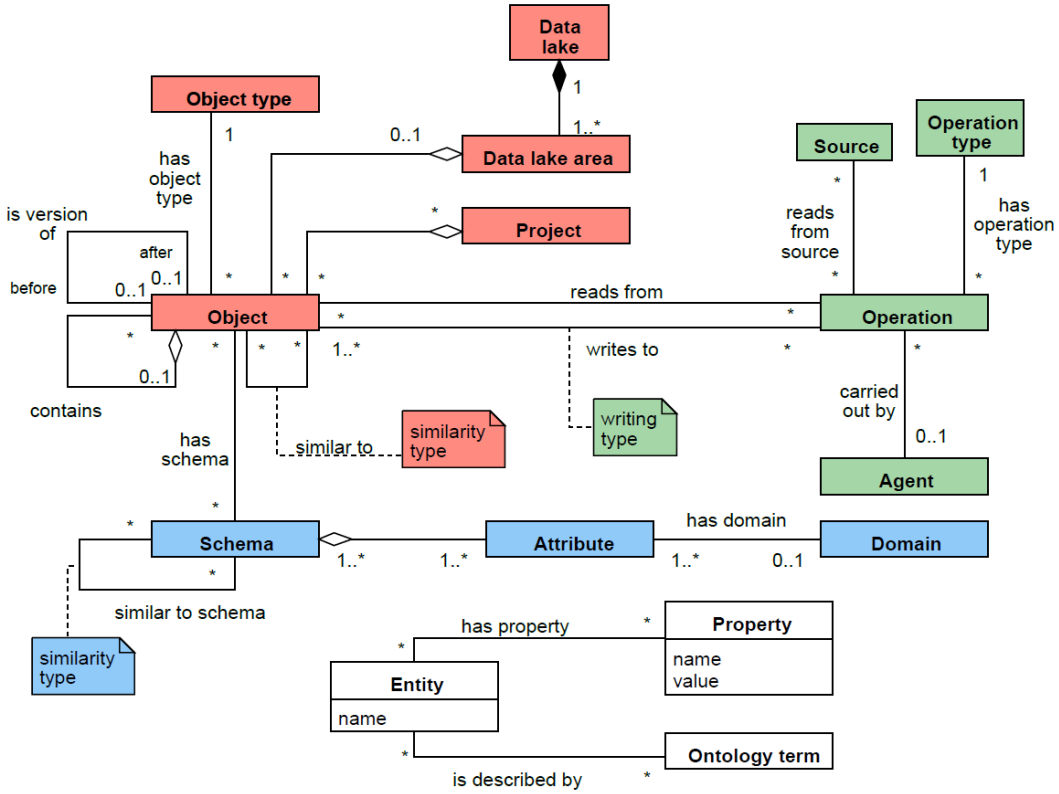
- MATCH (o:Object)-[]-(:Project {name:"ABC"})
  RETURN o
  - Return all objects of a given project
- MATCH (o:Object)-[]-(d:DataLakeArea)
  WHERE d.name = "Landing"
  AND o.name LIKE "2021_%"
  AND o.size < 100.000
  RETURN o
  - Return small objects with a given name pattern in the landing area



Object profiling and search

Schema-driven search

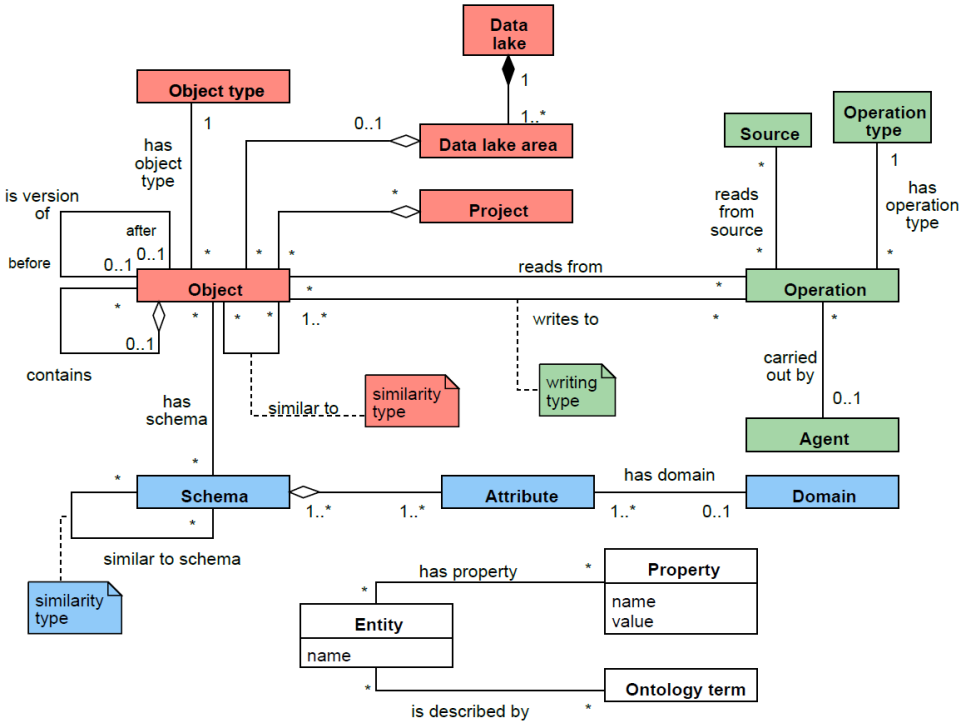
- MATCH (o:Object)-[]-(:Schema)-[]-(a:Attribute),
  (a)-[]-(:Domain {name: "FiscalCode"})
  RETURN o
  - Return objects that contain information referring to a given Domain



Object profiling and search

Provenance-driven search

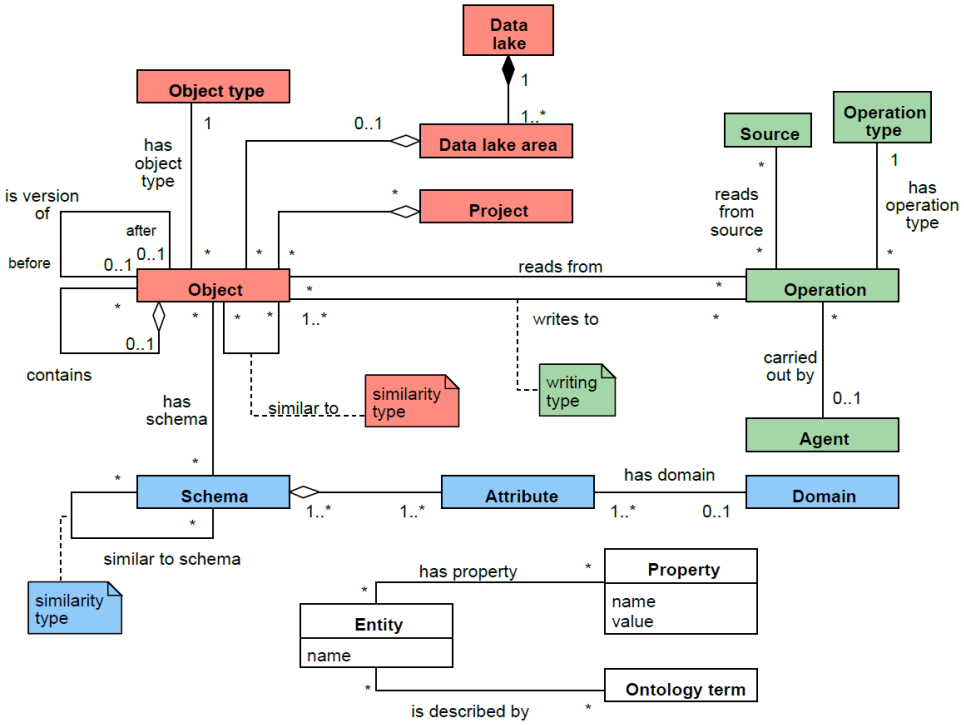
- MATCH (obj1:Object)-[:readsFrom]-(o:Operation)-[:writesTo]-(obj2:Object)
  CREATE (obj1)-[:ancestorOf]->(obj2)
- MATCH (:Object {id:123})-[:ancestorOf*]-(obj:Object)
  RETURN obj
  - Discover objects obtained from a given ancestor
- MATCH (obj:Object)-[:ancestorOf*]-(:Object {id:123})
  RETURN obj
  - Discover object(s) from which another has originated
- Example: a ML team wants to use datasets that were publicized as *canonical* for certain domains, but they find these datasets being too “groomed” for ML
  - Provenance links can be used to browse upstream and identify the less-groomed datasets that were used to derive the canonical datasets



Object profiling and search

Similarity-driven search

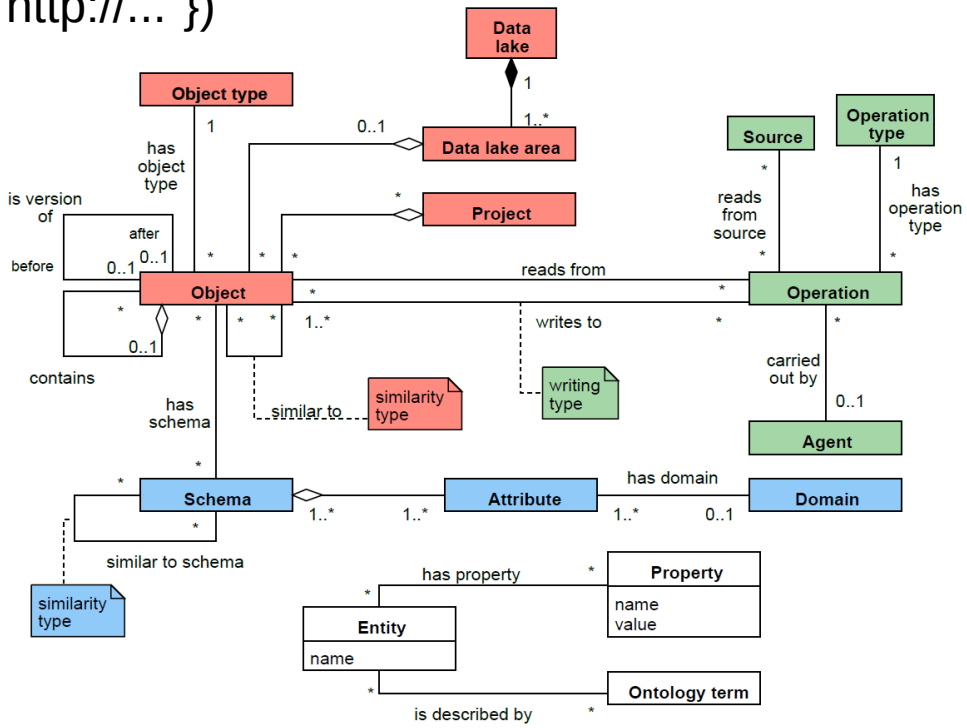
- MATCH (:Object {id:123})-[r:similarTo]-(o:Object)
  WHERE r.similarityType="affinity"
  RETURN o
  - Discover datasets to be merged in a certain query
- MATCH (:Object {id:123})-[r:similarTo]-(o:Object)
  WHERE r.similarityType="joinability"
  RETURN o
  - Discover datasets to be joined in a certain query
- Group similar objects and enrich the search results
  - List the main objects from each group
  - Restrict the search to the objects of a single group



Object profiling and search

Semantics-driven search

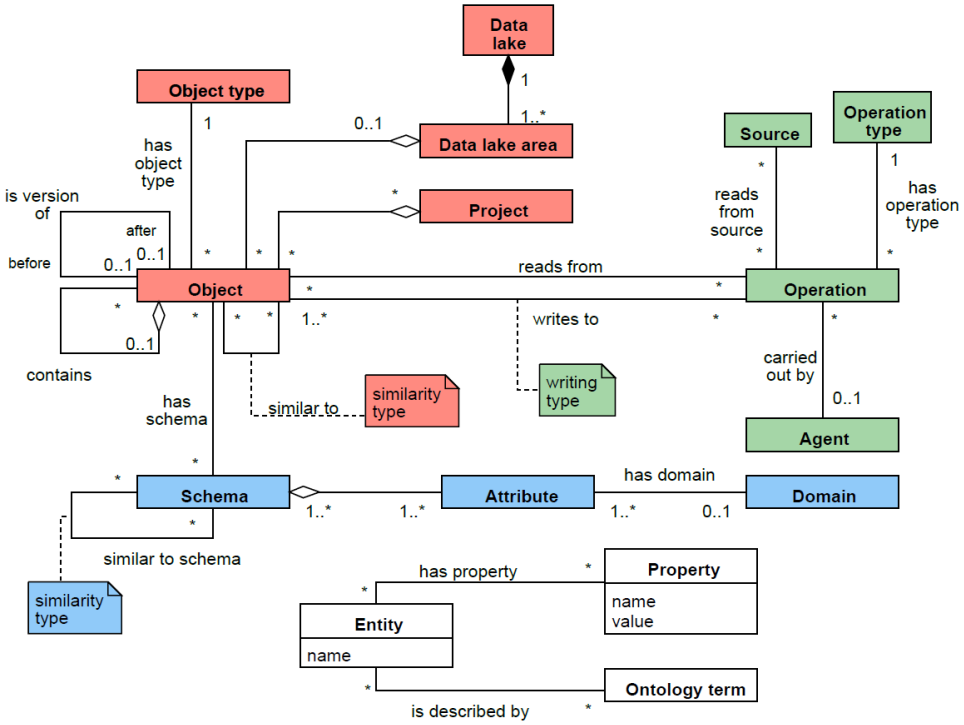
- MATCH (o:Object)-[:isDescribedBy]-(:OntologyTerm {uri:"http://..."})
  RETURN o
- MATCH (o:Object)-[*]-(any),
  (any)-[:isDescribedBy]-(:OntologyTerm {uri:"http://..."})
  RETURN o
  - Search objects without having any knowledge of their physical or intensional properties, but simply exploiting their traceability to a certain semantic concept



Object profiling and search

Profiling

- MATCH (o:Object)-[]-(:OntologyType {name:"Table"}),
  (o)-[]-(s:Schema)-[]-(a:Attribute),
  (o)-[r:similarTo]-(o2:Object),
  (o)-[:ancestorOf]-(o3:Object),
  (o4:Object)-[:ancestorOf]-(o)
  RETURN o, s, a, r, o2, o3, o4
  - Shows an object's properties, list the relationships with other objects in terms of similarity and provenance
  - Compute a representation of the intensional features that mostly characterize a group of objects (see slides on schema heterogeneity)



Provenance and versioning

Provenance: metadata pertaining to the history of a data item

- Any information that describes the production process of an end product
- Encompasses meta-data about entities, data, processes, activities, and persons involved in the production process
- Essentially, it describes a transformation pipeline, including the origin of objects and the operations they are subject to

J.Wang, D. Crawl, S. Purawat, M. H. Nguyen, I. Altintas, Big data provenance: Challenges, state of the art and opportunities, in: Proc. BigData, Santa Clara, CA, USA, 2015, pp. 2509–2516.

M. Herschel, R. Diestelk¨amper, H. Ben Lahmar, A survey on provenance: What for? What form? What from?, VLDB J. 26 (6) (2017) 881–906.

Provenance and versioning

Several use cases

- Supply chain
  - Assess food's quality and gain trust in food products
- Scientific experiments
  - Experiments on big data scales are not easily repeatable (e.g., 100 PB workloads @CERN, SKA)
  - Ensuring the reproducibility and accessibility of the scientific results is essential
- Complex data processing
  - The development of complex data processing pipelines is iterative/incremental
  - Provenance analysis leads to faster and higher-quality analysis, debugging, and refinement

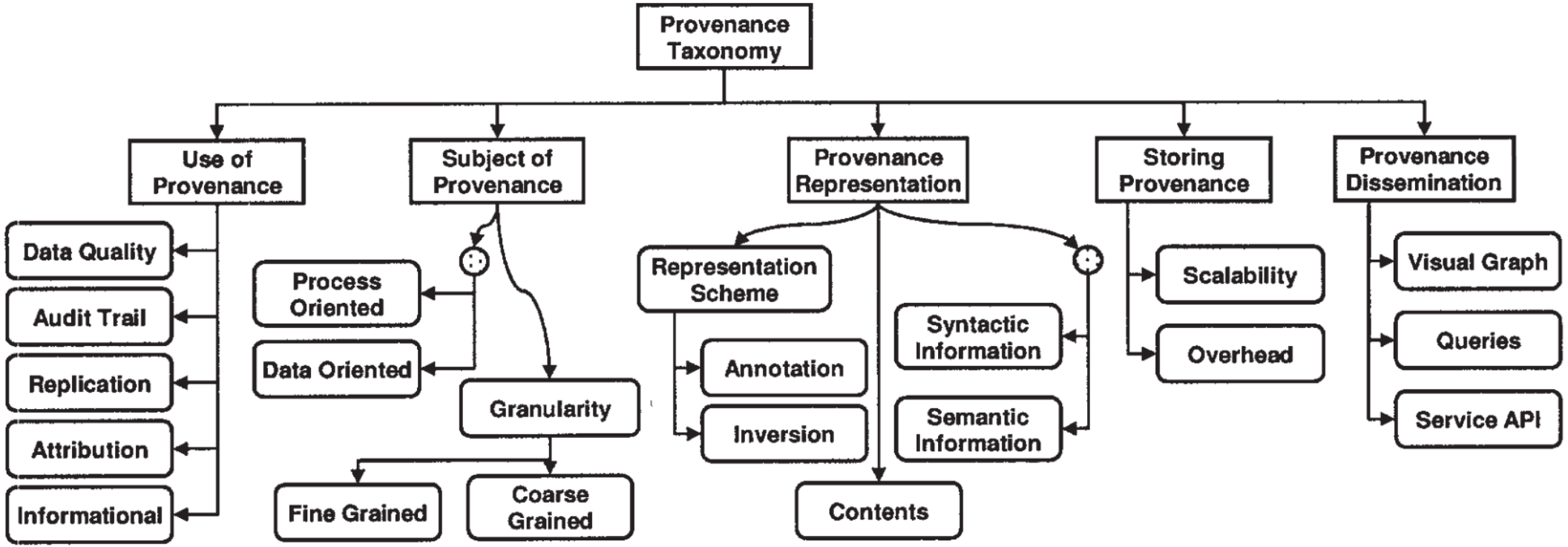
New, S.: The transparent supply chain. Harvard Bus. Rev.88, 1–5(2010)

Cranmer, K., Heinrich, L., Jones, R., South, D.M.: Analysispreservation in ATLAS. J. Physi.664(3) (2015).

Bourhis, P., Deutch, D., Moskovitch, Y.: POLYTICS: provenance-based analytics of data-centric applications. In: IEEE Interna-tional Conference on Data Engineering (ICDE), pp. 1373–1374(2017)

Provenance and versioning

A taxonomy of provenance techniques



Y. Simmhan, B. Plale, D. Gannon, A survey of data provenance in e-science,SIGMOD Rec. 34 (3) (2005) 31–36.

Provenance and versioning

An important aspect is the granularity of provenance

- Fine-grained provenance is typically used for single vertical applications
- It requires to collect huge amounts of detailed information to enable a very detailed tracing
- Coarse-grained provenance is appropriate to ensure a broad coverage of highly heterogeneous transformations possibly involving several applications and datasets

Choosing a granularity is the result of a trade-off between accuracy and computational effort

- Storing only the name and the version of a clustering algorithm enables an approximate reproducibility of the results
- Storing all its parameters makes this functionality much more accurate

Provenance and versioning

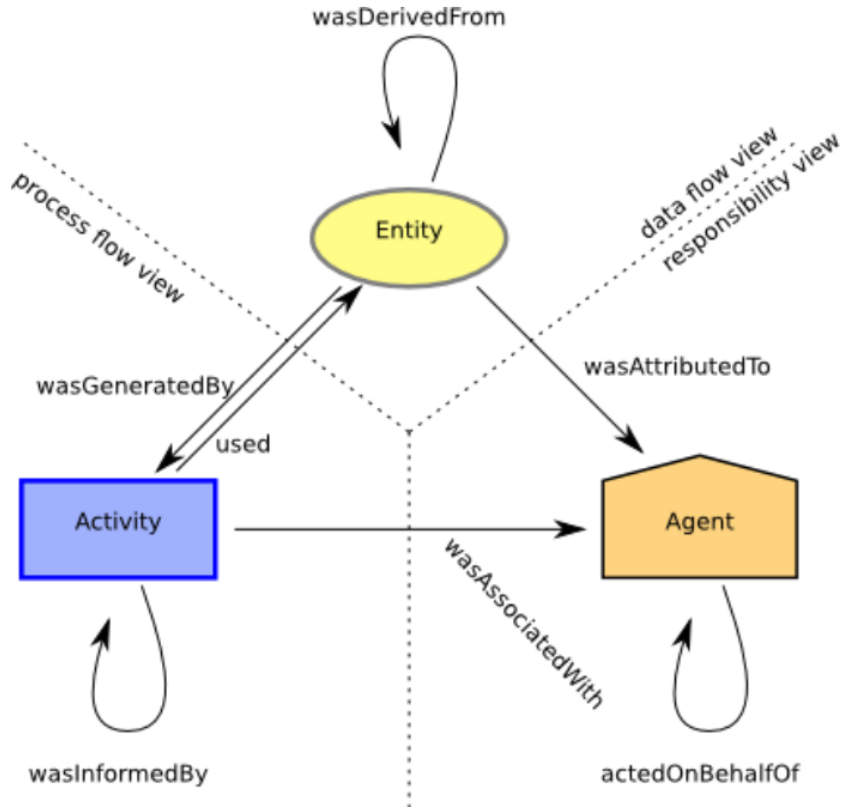
Provenance functionalities

- Data quality
  - Monitoring accuracy, precision, and recall of produced objects to notify the data scientist when a transformation pipeline is not behaving as expected
- Debugging
  - Inferring the cause of pipeline failures is challenging and requires an investigation of the overall processing history, including input objects and the environmental settings
- Reproducibility
  - Re-execution of all or part of the operations belonging to a pipeline
- Trustworthiness
  - Help data scientists to trust the objects produced by tracing them back to their sources and storing the agents who operated on those objects
- Versioning
  - Marking a generated object and its versions (e.g., due to changes in a database schema) helps in identifying relevant objects along with their semantic versions, and to operate with legacy objects

Provenance and versioning

PROV: a standard for provenance modeling

- Several tools exists for managing PROV metadata
  - https://openprovenance.org/services/view/translator
  - https://lucmoreau.github.io/ProvToolbox/
  - https://prov.readthedocs.io/en/latest/
- Compilance with PROV ensures integration with existing tools for querying and visualization



L. Moreau, P. T. Groth, Provenance: An Introduction to PROV, Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool Publishers, 2013.

Provenance and versioning

Some current research directions

- Expand PROV to better suite big data scenarios
  - Y. Gao, X. Chen and X. Du, "A Big Data Provenance Model for Data Security Supervision Based on PROV-DM Model," in IEEE Access, vol. 8, pp. 38742-38752, 2020.
- Define provenance-based approaches to measure the quality of big data
  - Taleb, I., Serhani, M.A., Bouhaddioui, C. et al. Big data quality framework: a holistic approach to continuous quality management. J Big Data 8, 76 (2021).
- An outline of the challenges, including granularity identification, integration, security concerns
  - A. Chacko and S. D. Madhu Kumar, "Big data provenance research directions," *TENCON 2017 - 2017 IEEE Region 10 Conference*, 2017, pp. 651-656, doi: 10.1109/TENCON.2017.8227942.
- Blockchain-based provenance systems
  - Dang, T. K., & Duong, T. A. (2021). An effective and elastic blockchain-based provenance preserving solution for the open data. *International Journal of Web Information Systems*.
  - Ruan, P., Dinh, T. T. A., Lin, Q., Zhang, M., Chen, G., & Ooi, B. C. (2021). LineageChain: a fine-grained, secure and efficient data provenance system for blockchains. *The VLDB Journal*, *30*(1), 3-24.

Orchestration support

The orchestrator is the component in charge of controlling the execution of computation activities

- Either through a regular scheduling of the activities
- Or by triggering a process in response to a certain event

Several entities (either processes or human beings) can cover this role to activate some data processes

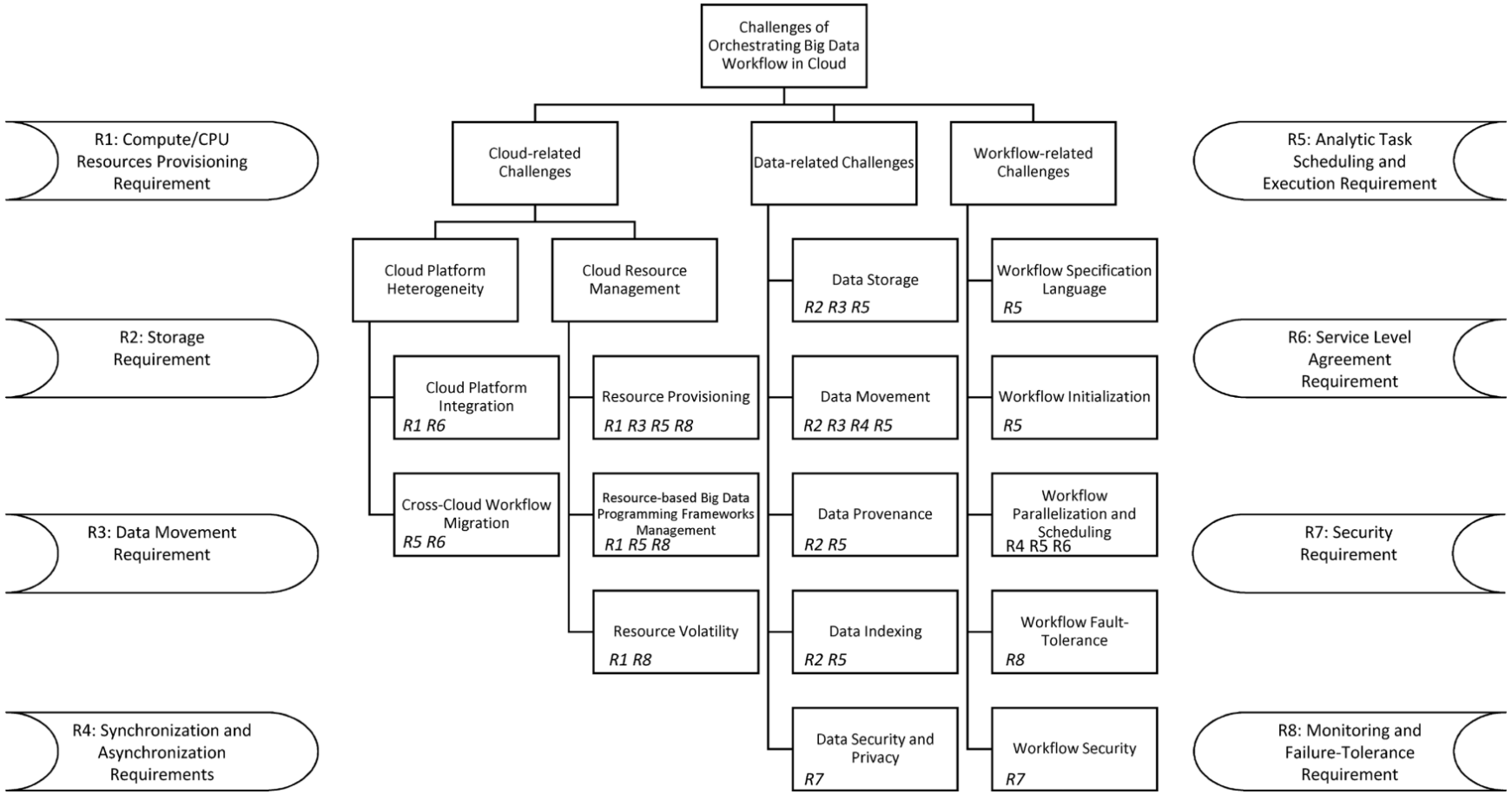
Orchestration support

Orchestration functionalities

- Dynamic/condition-based behavior
  - Decide *what* data process should be activated under different conditions
  - Decide *how* to tune the parameters in case of parametric data processes
- Triggering
  - Decide *when* to trigger a certain data process
- Scoping
  - Assess the trustworthiness of Objects to decide *if* a certain data process should be activated or not
- Resource estimation/prediction
  - Decide the optimal amount of resources required to terminate successfully while leaving sufficient resources to the other concurrent process, based on previous executions and current settings
  - Negotiate the resources with the cluster's resource manager

Orchestration support

Orchestration requirements & challenges



Barika, M., Garg, S., Zomaya, A. Y., Wang, L., Moorsel, A. V., & Ranjan, R. (2019). Orchestrating big data analysis workflows in the cloud: research challenges, survey, and future directions. *ACM Computing Surveys (CSUR)*, *52*(5), 1-41.

Orchestration support

Orchestration requirements

- R1 Compute/CPU resource provisioning
  - Determine the right amount of resources
  - Continuously monitor and manage them in a dynamic execution environment
- R2 Storage
  - Choose the right cloud storage resource, data location, and format (if the application is parametric)
- R3 Data movement
  - Dynamically transfer large datasets between compute and storage resources
- R4 Synchronization and asynchronization
  - Manage the control and data flow dependencies across analytics tasks

Barika, M., Garg, S., Zomaya, A. Y., Wang, L., Moorsel, A. V., & Ranjan, R. (2019). Orchestrating big data analysis workflows in the cloud: research challenges, survey, and future directions. *ACM Computing Surveys (CSUR)*, *52*(5), 1-41.

Orchestration support

Orchestration requirements

- R5 Analytic task scheduling and execution
  - Scheduling and coordinating the execution of workflow tasks across diverse sets of big data programming models
  - Tracking and capturing provenance of data
- R6 Service Level Agreement
  - Executions may need to meet user-defined QoS requirements (e.g., a strict execution deadline)
- R7 Security
  - Beyond standard encryption approaches: private (anonymous) computation, verification of outcomes in multi-party settings, placement of components according to security policies
- R8 Monitoring and Failure-Tolerance
  - Ensure that everything is streamlined and executed as anticipated
  - As failures could happen at any time, handle those failures when they occur or predicting them before they happen

Barika, M., Garg, S., Zomaya, A. Y., Wang, L., Moorsel, A. V., & Ranjan, R. (2019). Orchestrating big data analysis workflows in the cloud: research challenges, survey, and future directions. *ACM Computing Surveys (CSUR)*, *52*(5), 1-41.

Orchestration support

Orchestration challenges

- Cloud Platform Heterogeneity
  - **Integration** (different APIs, virtualization formats, pricing policies, hardware/software configurations)
  - **Workflow Migration** (e.g., to aspire to specific QoS features in the target cloud or better price)
- Cloud Resource Management
  - **Resource Provisioning** (selecting the right configuration of virtual resources; the resource configuration search space grows exponentially, and the problem is often NP-complete)
  - **Resource-based Big Data Programming Frameworks Management** (automatically select the configurations for both IaaS-level resource and PaaS-level framework to consistently accomplish the anticipated workflow-level SLA requirements, while maximizing the utilization of cloud datacenter resources)
  - **Resource Volatility** (at different levels: VM-level, big data progressing framework-level and workflow task-level)

Barika, M., Garg, S., Zomaya, A. Y., Wang, L., Moorsel, A. V., & Ranjan, R. (2019). Orchestrating big data analysis workflows in the cloud: research challenges, survey, and future directions. *ACM Computing Surveys (CSUR)*, *52*(5), 1-41.

Orchestration support

Orchestration challenges

- Data-related
  - **Storage** (where the data will be residing, which data format will be used)
  - **Movement** (minimize transfer rates, exploit *data locality* in task-centric or worker-centric way)
  - **Provenance** (tradeoff expressiveness with overhead)
  - **Indexing** (which dataset is worth indexing and how)
  - **Security and Privacy** (cryptography, access control, integrity, masking, etc.)

Barika, M., Garg, S., Zomaya, A. Y., Wang, L., Moorsel, A. V., & Ranjan, R. (2019). Orchestrating big data analysis workflows in the cloud: research challenges, survey, and future directions. *ACM Computing Surveys (CSUR)*, *52*(5), 1-41.

Orchestration support

Orchestration challenges

- Workflow-related
  - **Specification Language** (devising a high level, technology-/cloud-independent workflow language)
  - **Initialization** (subdivision into fragments considering dependencies, constraints, etc.)
  - **Parallelization and Scheduling** (with super-workflows defined at application and task level)
  - **Fault-Tolerance** (thing can go wrong at workflow-, application-, and cloud-level)
  - **Security** (securing workflow logic and computation)

Barika, M., Garg, S., Zomaya, A. Y., Wang, L., Moorsel, A. V., & Ranjan, R. (2019). Orchestrating big data analysis workflows in the cloud: research challenges, survey, and future directions. *ACM Computing Surveys (CSUR)*, *52*(5), 1-41.

Metadata challenge - Conclusions

Some predictions for 2022

- Data lakes and data warehouses will become indistinguishable
- Analytics will merge with SQL-based systems within data platforms
- Universal standards for governance, lineage, and metrics will begin to emerge
- Predictive analytics will evolve dramatically
- Knowledge graphs will be in high demand
- The next generation of data sharing will require domain-oriented governance within (and between) organizations

Barr Moses 2021, https://towardsdatascience.com/the-future-of-the-modern-data-stack-2de175b3c809