Data Platforms

Data platform

Data platform

- An integrated set of technologies that collectively meets an organization's end-to-end data needs such as acquisition, storage, preparation, delivery, and governance, as well as a security layer for users and applications
- Rationale: relieve users from complexity of administration and provision
  - Not only technological skills, but also privacy, access control, etc.
  - Users should only focus on functional aspects

Data platform

Companies are collecting tons of data to enable advanced analytics

- Raw data is difficult to obtain, interpret, and maintain
- Data is more and more heterogeneous
- There is need for curating data to make it consumable

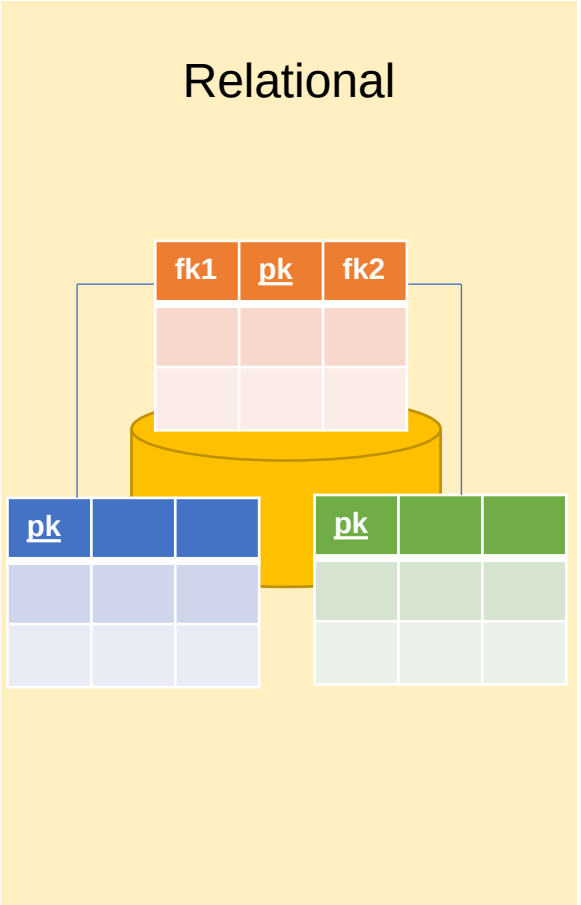
Where are we collecting/processing data?

- Getting value from data is not (only) a matter of storage
- Need integrated and multilevel analytical skills and techniques

Data platform

Database

"A database is a structured and persistent collection of information about some aspect of the real world organized and stored in a way that facilitates efficient retrieval and modification. The structure of a database is determined by an abstract data model. Primarily, it is this structure that differentiates a database from a data file."



Özsu M.T. (2018) Database. In: Encyclopedia of Database Systems. Springer, New York, NY. https://doi.org/10.1007/978-1-4614-8265-9_80734

ICT in companies

Up to some years ago, the main goal of databases in companies has been that of storing operational data, i.e., data generated by operations carried out within business processes

Computer science was seen as a subsidiary discipline that makes information management faster and cheaper, but does not create profits in itself

The evolution of information systems

- The role of computer science in companies has radically changed since the early 70's. ICT systems turned from simple tools to improve process efficiency into key factors of company organizations capable of deeply impacting on the structure of business processes

Auxiliary technology to manage the company information system

Organizational discipline that impacts on business processes, services, and company structure

The new role of computer science in decision making

An exponential increase in operational data has made computers the only tools suitable for providing data for decision-making performed by business managers

The massive use of techniques for analyzing enterprise data made information systems a key factor to achieve business goals

Big data vs small data

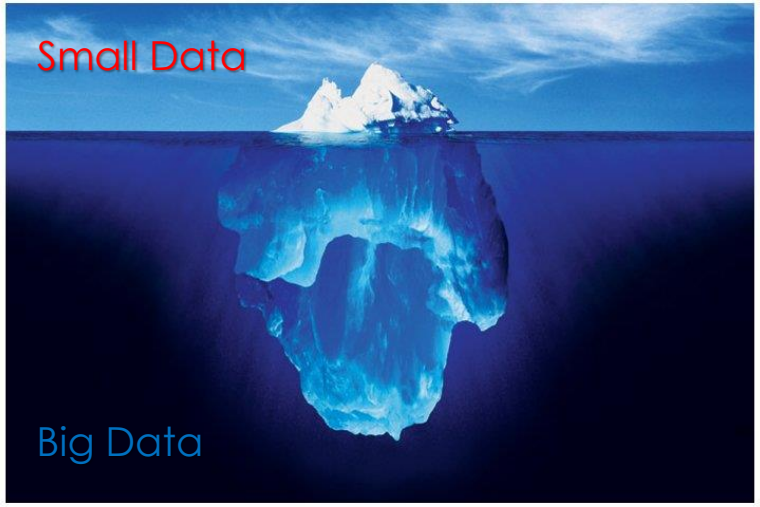
The progressive digitalization of services and systems generates an enormous mass of heterogeneous and real-time data

Big Data must be transformed into Small Data so that it can be exploited for decision-making purposes

Small data is data that is 'small' enough for human comprehension. It is data in a volume and format that makes it accessible, informative and actionable.

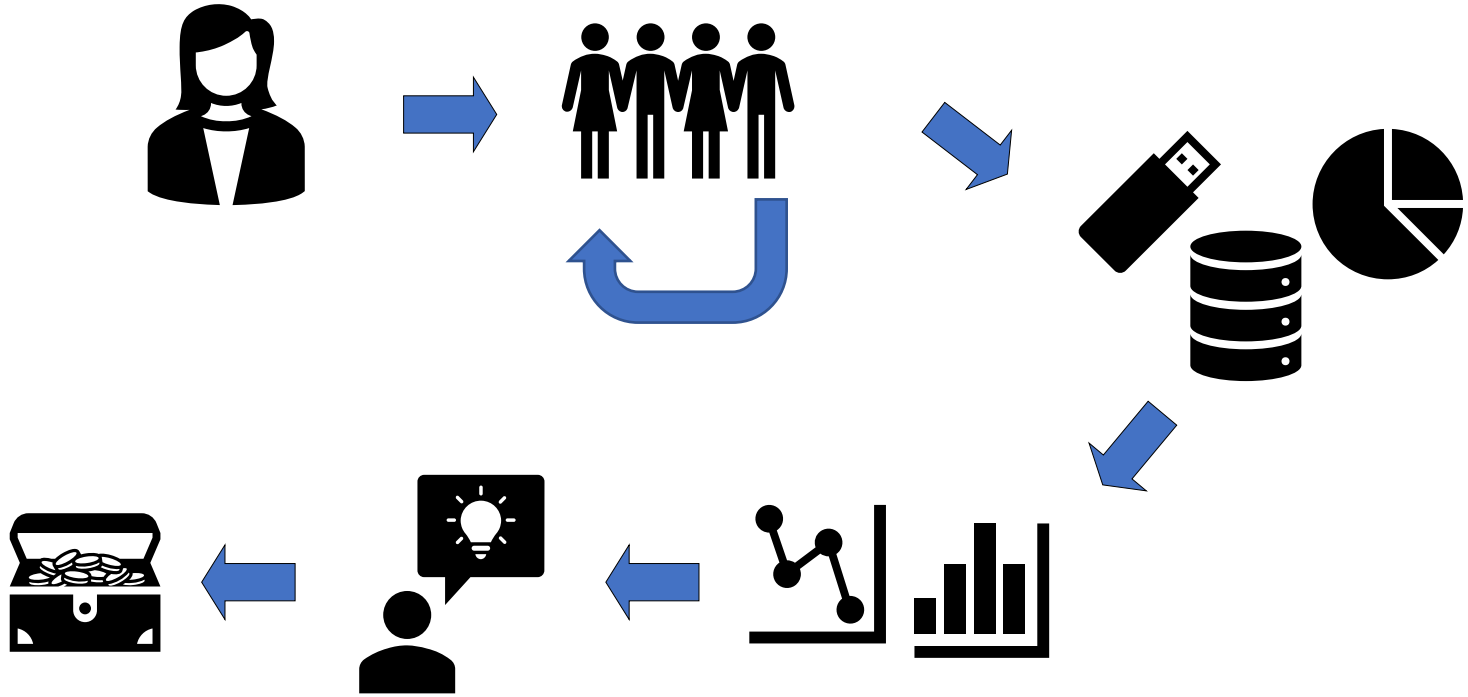
To manage the transformation, we need:

- Ad hoc Technology (e.g., NO SQL DBMS)
- Computing power (e.g., cloud & cluster computing)
- Automated systems (e.g., artificial intelligence)
- Digital culture
- The right processes (i.e., digital ready processes)



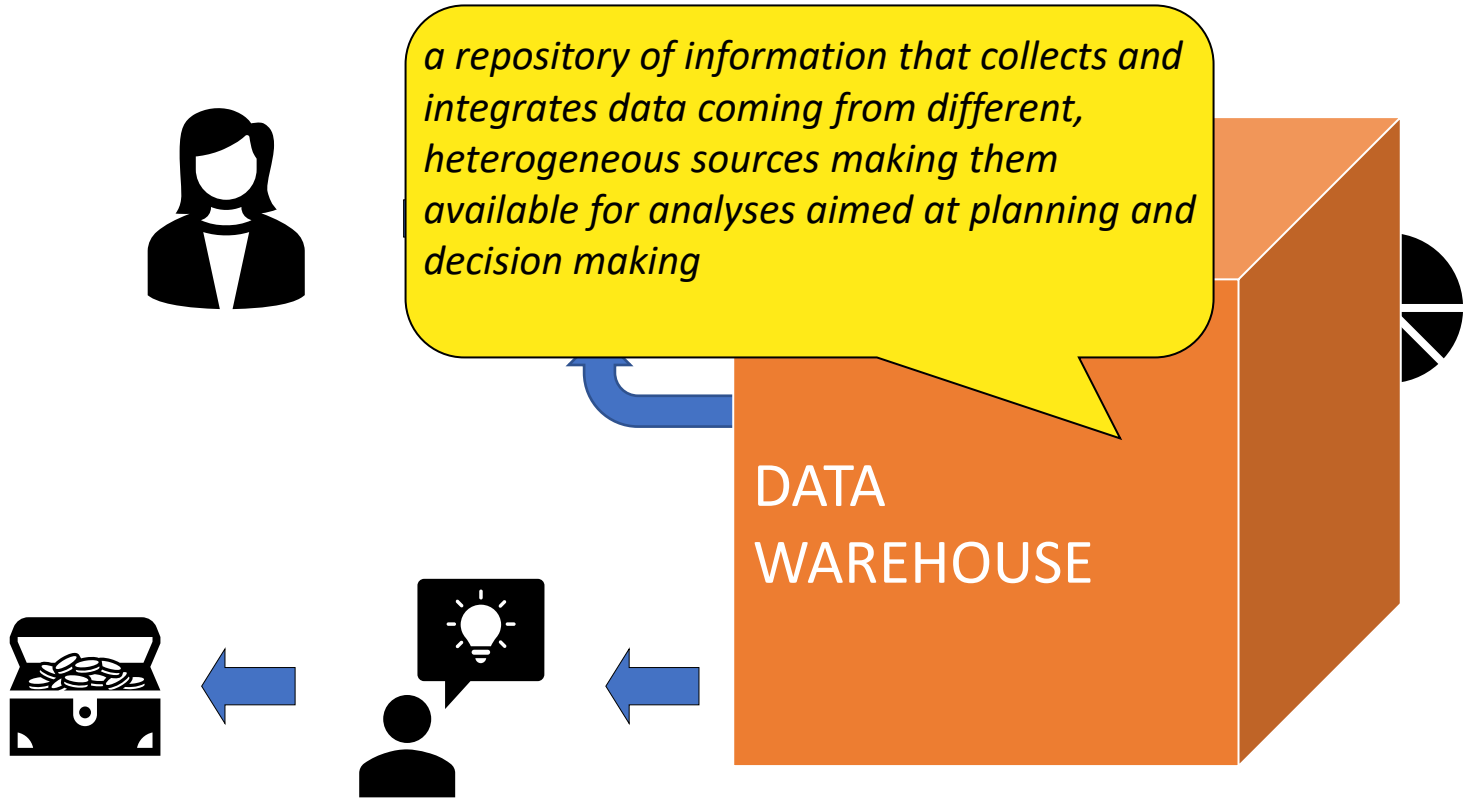
A typical scenario...

... is that of a large company, with several branches, whose managers wish to quantify and evaluate the contribution given from each branch to the global profit



A typical scenario...

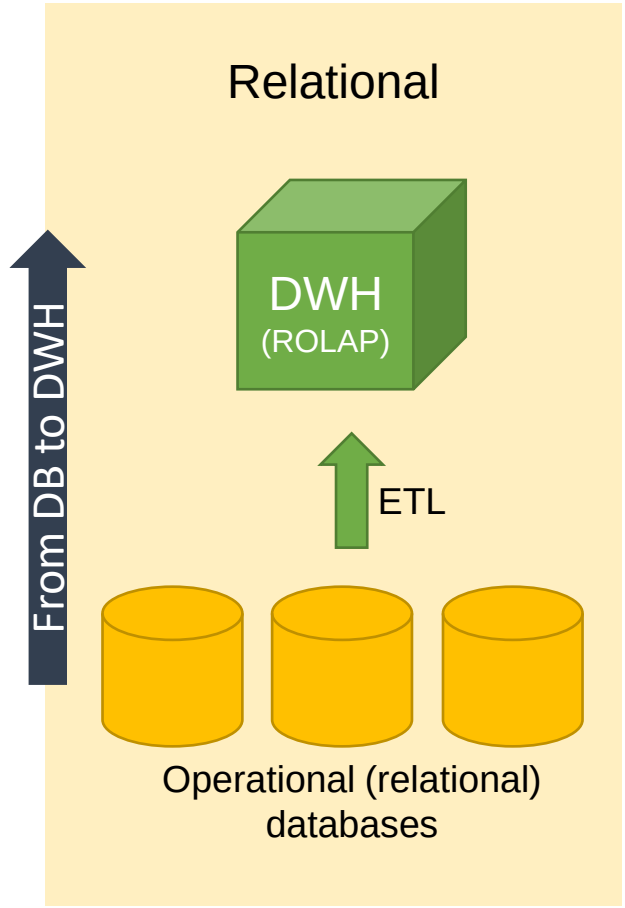
... is that of a large company, with several branches, whose managers wish to quantify and evaluate the contribution given from each branch to the global profit



Data platform

Data Warehouse

"A collection of data that supports decision-making processes. It provides the following features: subject-oriented, integrated and consistent, not volatile."



Matteo Golfarelli and Stefano Rizzi. *Data warehouse design: Modern principles and methodologies*. McGraw-Hill, Inc., 2009.

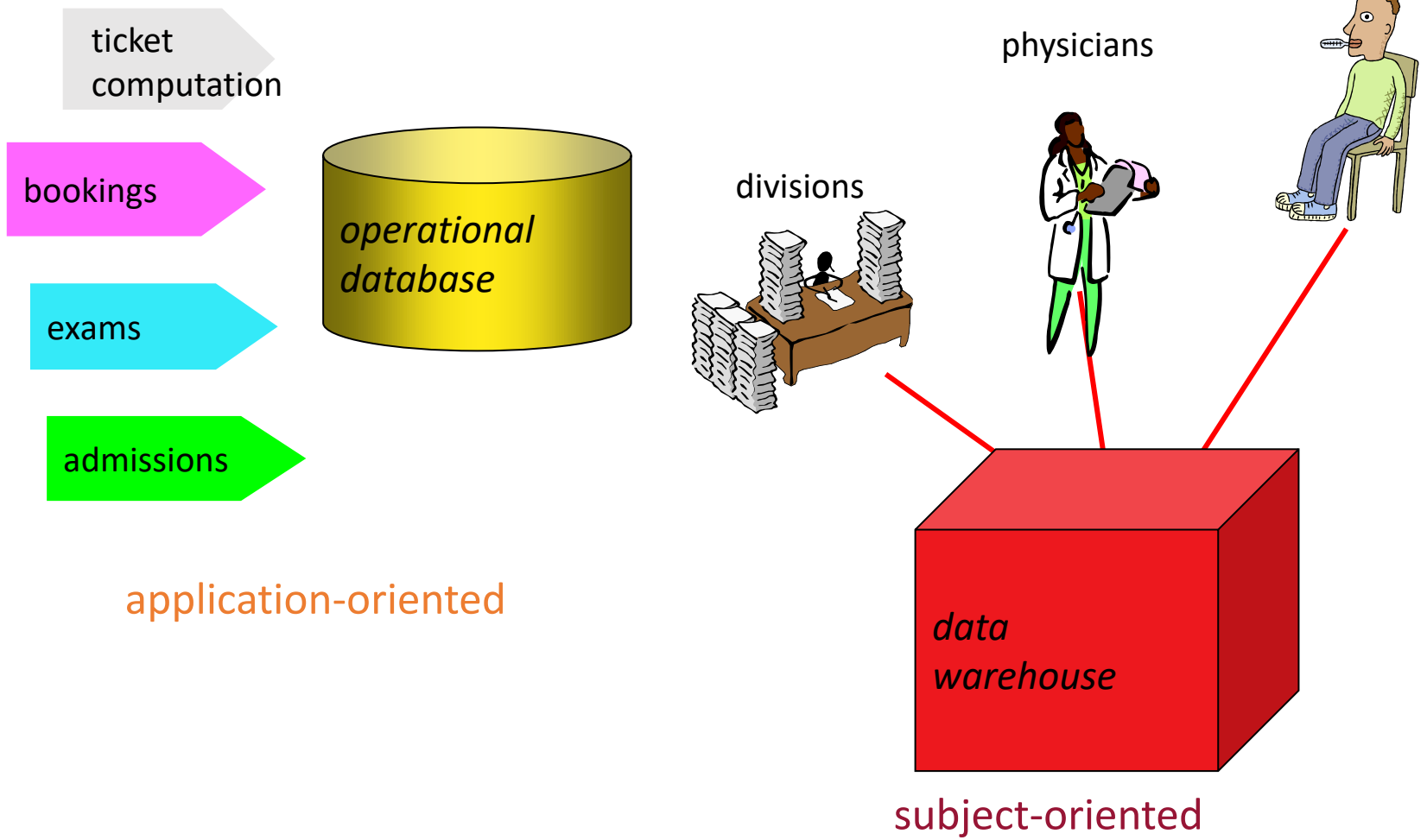
The Data Warehouse

In the middle of this process, a data warehouse is a data repository that fulfills the requirements

A data warehouse is a collection of data that supports decision-making processes. It provides the following features:

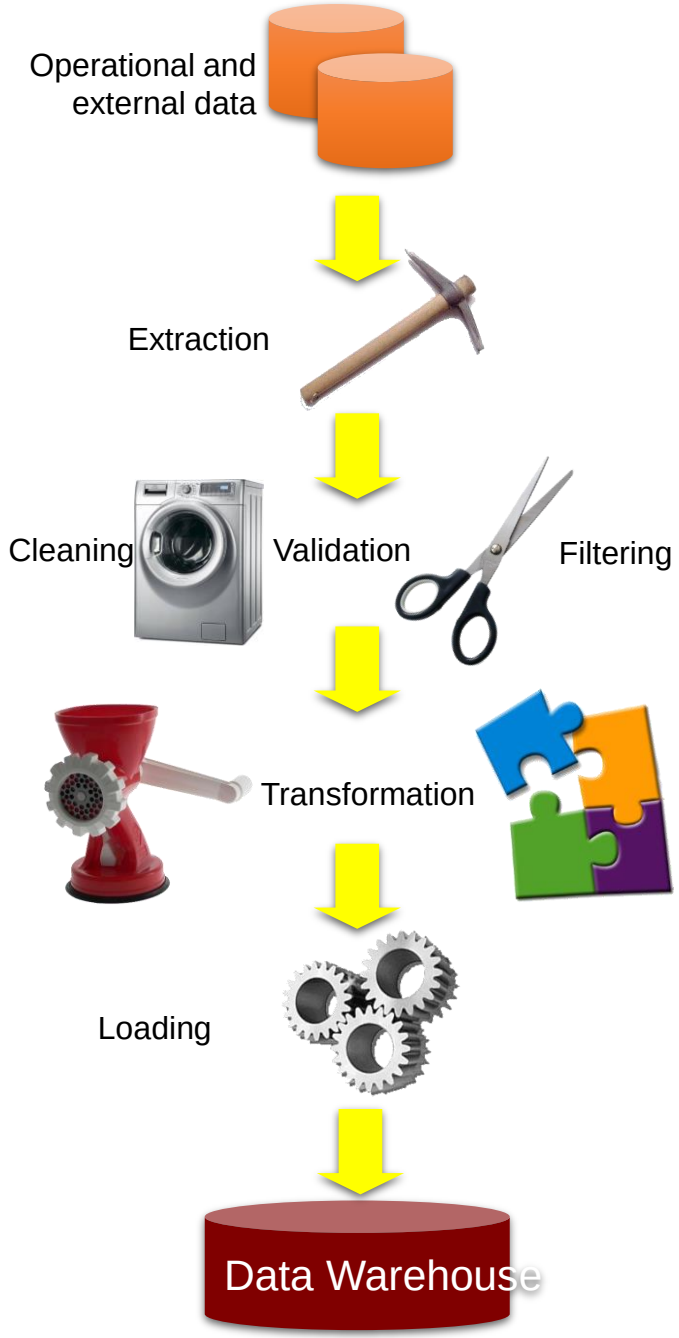
- It is subject-oriented;
- It is integrated and consistent;
- It shows its evolution over time and it is not volatile

...subject-oriented



...integrated and consistent

Data warehouses take advantage of multiple data sources, such as data extracted from production and then stored to enterprise databases, or even data from a third party's information systems. A data warehouse should provide a unified view of all the data.



...shows its temporal evolution

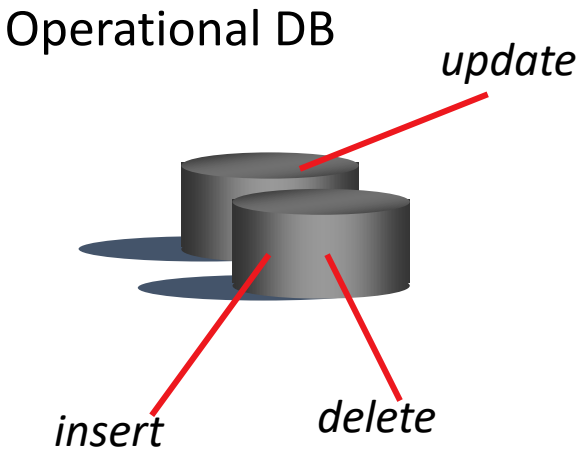
Operational DB

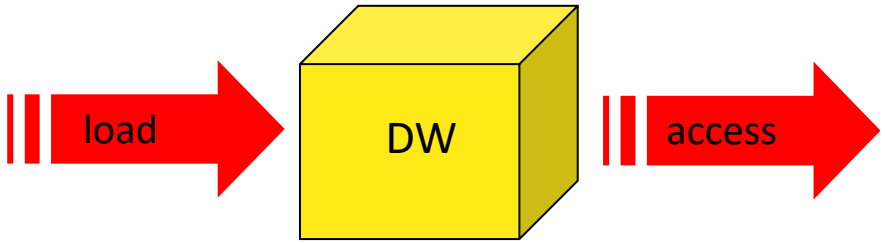
Limited historical content, time is often not part of the keys, data are updated

DW

Rich historical content, time is part of the keys, a snapshot of data taken at a given time cannot be updated

...non volatile

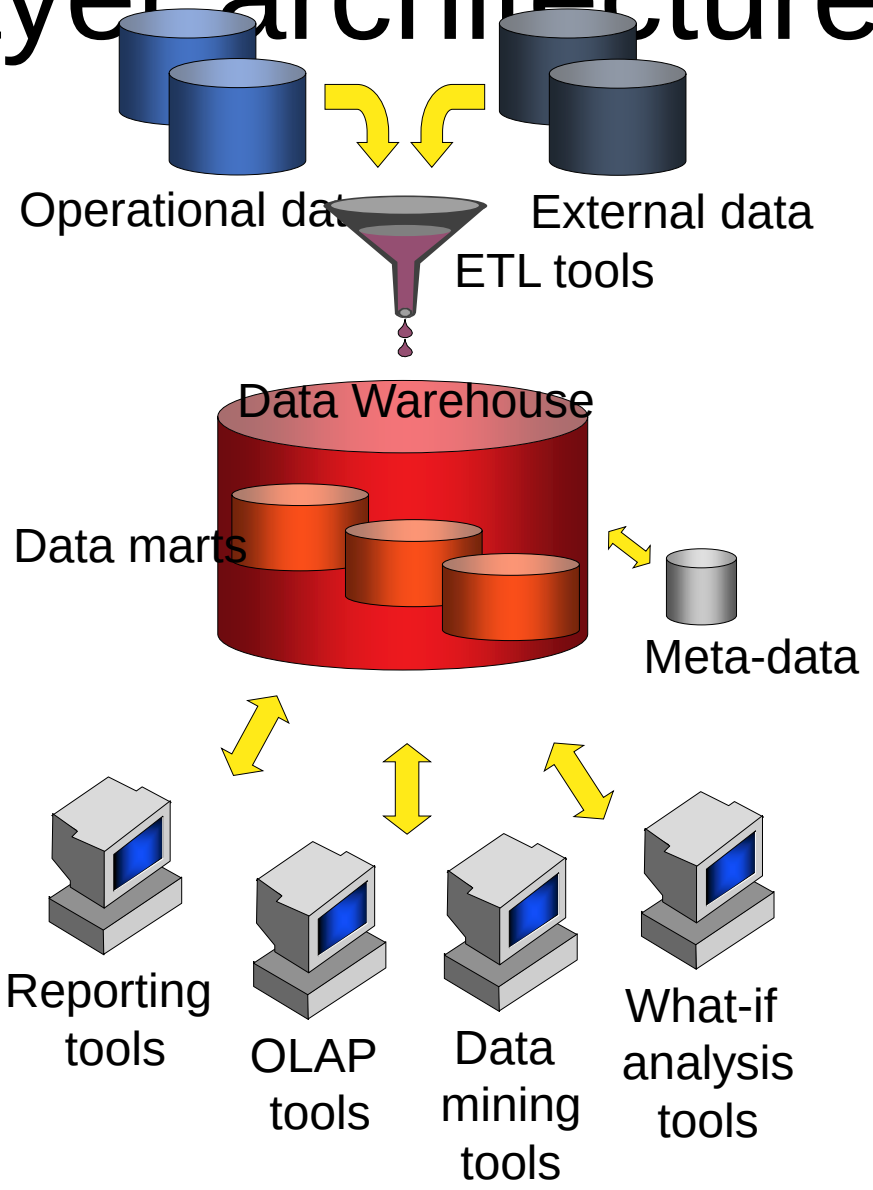




Huge data volumes:
from 50 GBs to some TBs
in a few years

- no need for advanced transaction management techniques required by operational applications
- key problems are query-throughput and resilience

Two-layer architectures



Source layer

Data staging

Data warehouse layer

Analysis

DATA MART:

A subset or an aggregation of the data stored to a primary data warehouse. It includes a set of information pieces relevant to a specific business area, corporate department, or category of users.

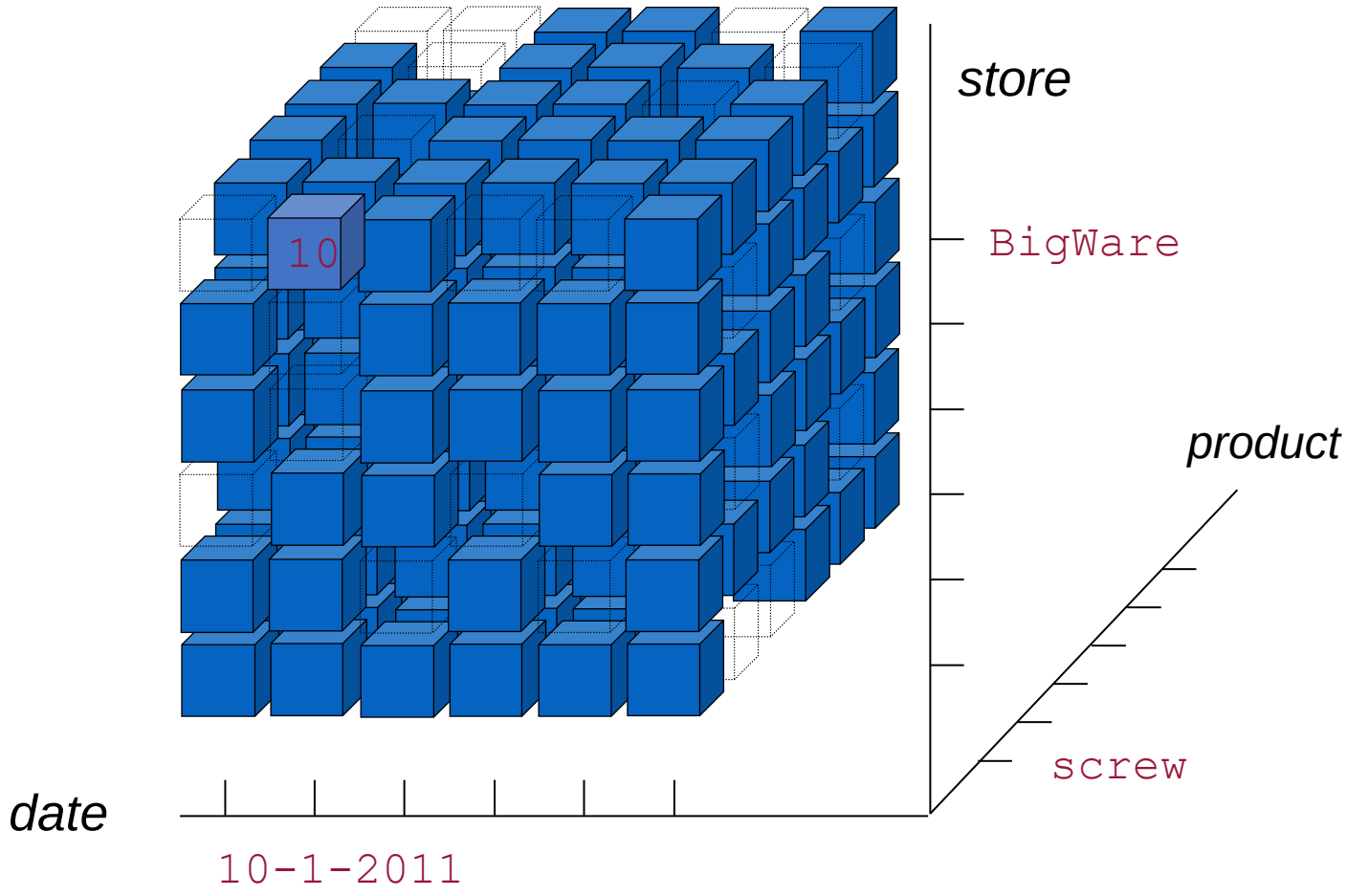
The multidimensional model

It is the key for representing and querying information in a DW

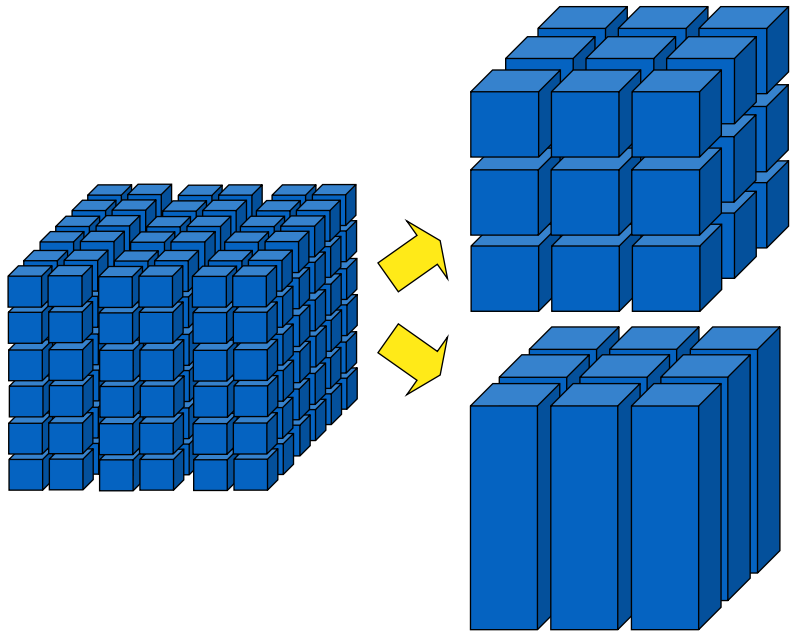
Facts of interest are represented in *cubes* where:

- each cell stores numerical *measures* that quantify the fact from different points of view;
- each axis is a *dimension* for analyzing measure values;
- each dimension can be the root of a *hierarchy* of attributes used to aggregated measure values

The Sales cube

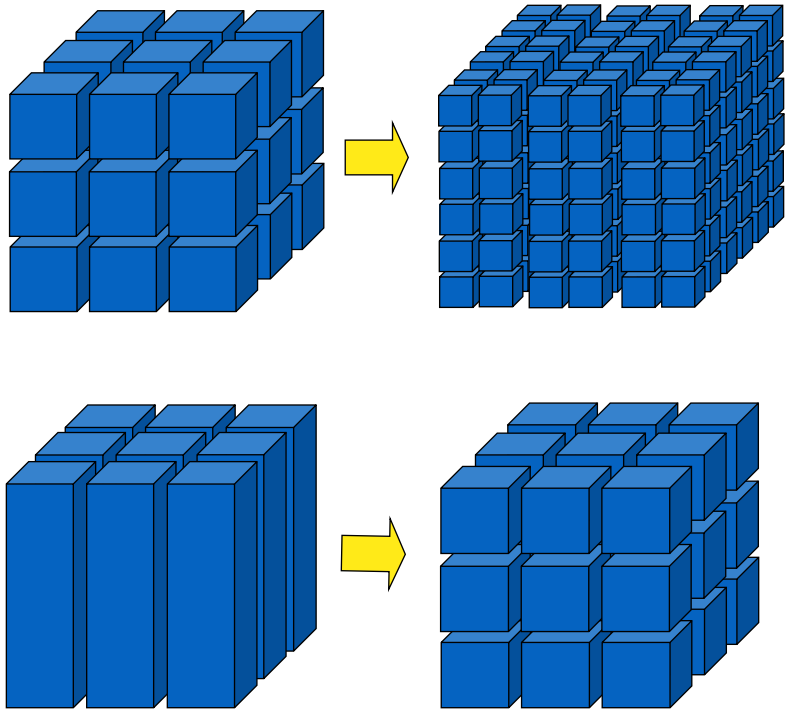


OLAP operators



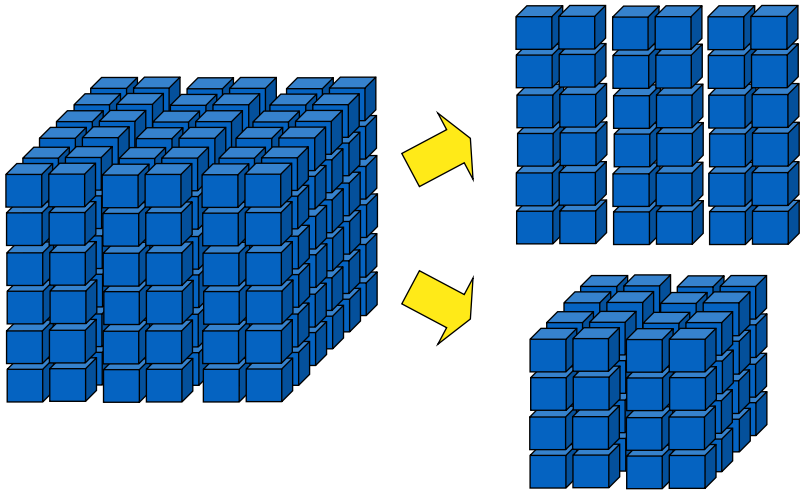
roll-up

OLAP operators



drill-down

OLAP operators



slice-and-dice

The Dimensional Fact Model

The DFM is a graphical conceptual model for data mart design, devised to:

1. lend effective support to conceptual design
2. create an environment in which user queries may be formulated intuitively
3. make communication possible between designers and end users with the goal of formalizing requirement specifications
4. build a stable platform for logical design (*independently of the target logical model*)
5. provide clear and expressive design documentation

The conceptual representation generated by the DFM consists of a set of *fact schemata* that basically model facts, measures, dimensions, and hierarchies

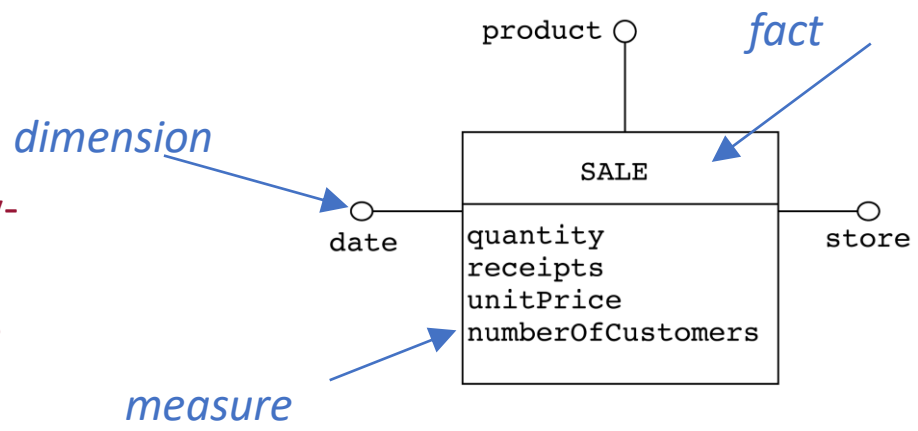
DFM: basic concepts

A fact is a concept relevant to decision-making processes. It typically models a set of events taking place within a company (e.g., sales, shipments, purchases, ...). It is essential that a fact have dynamic properties or evolve in some way over time

A measure is a numerical property of a fact and describes a quantitative fact aspect that is relevant to analysis (e.g., every sale is quantified by its receipts)

A dimension is a fact property with a finite domain and describes an analysis coordinate of the fact. Typical dimensions for the sales fact are products, stores, and dates

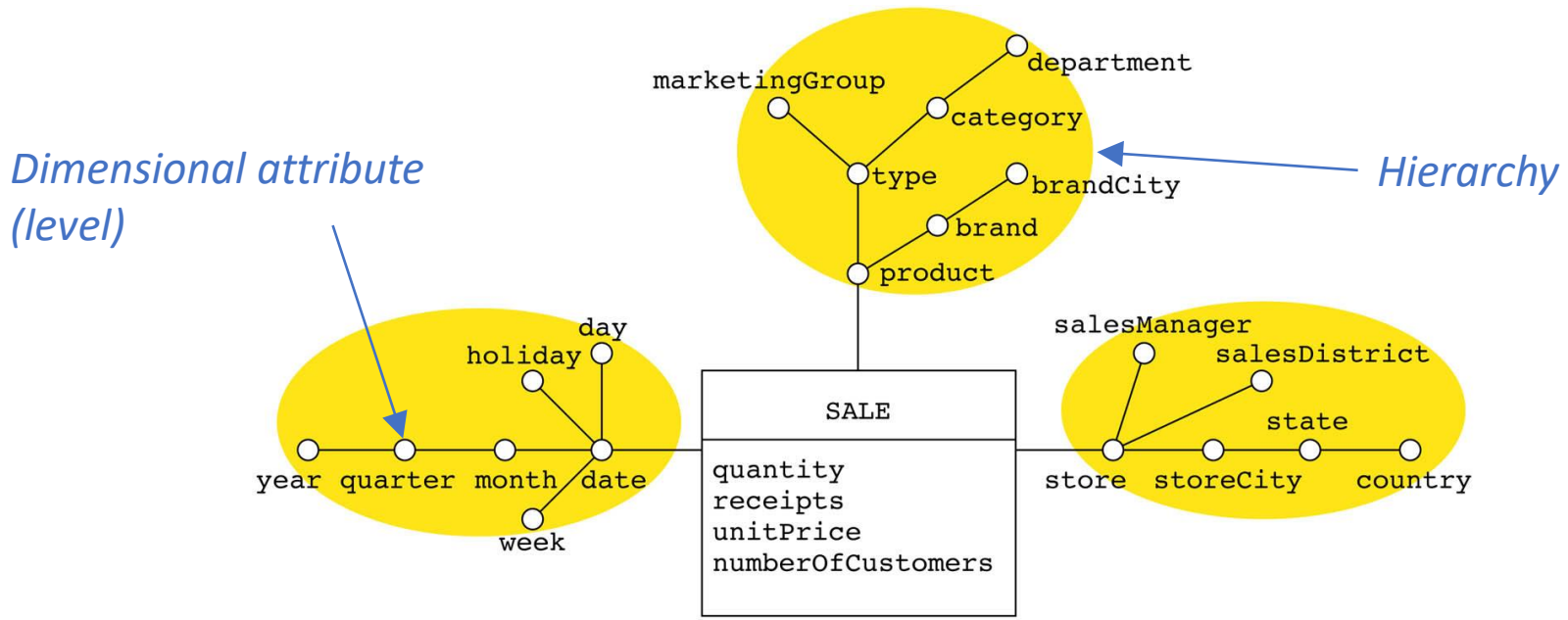
A fact expresses a many-to-many relationship between its dimensions



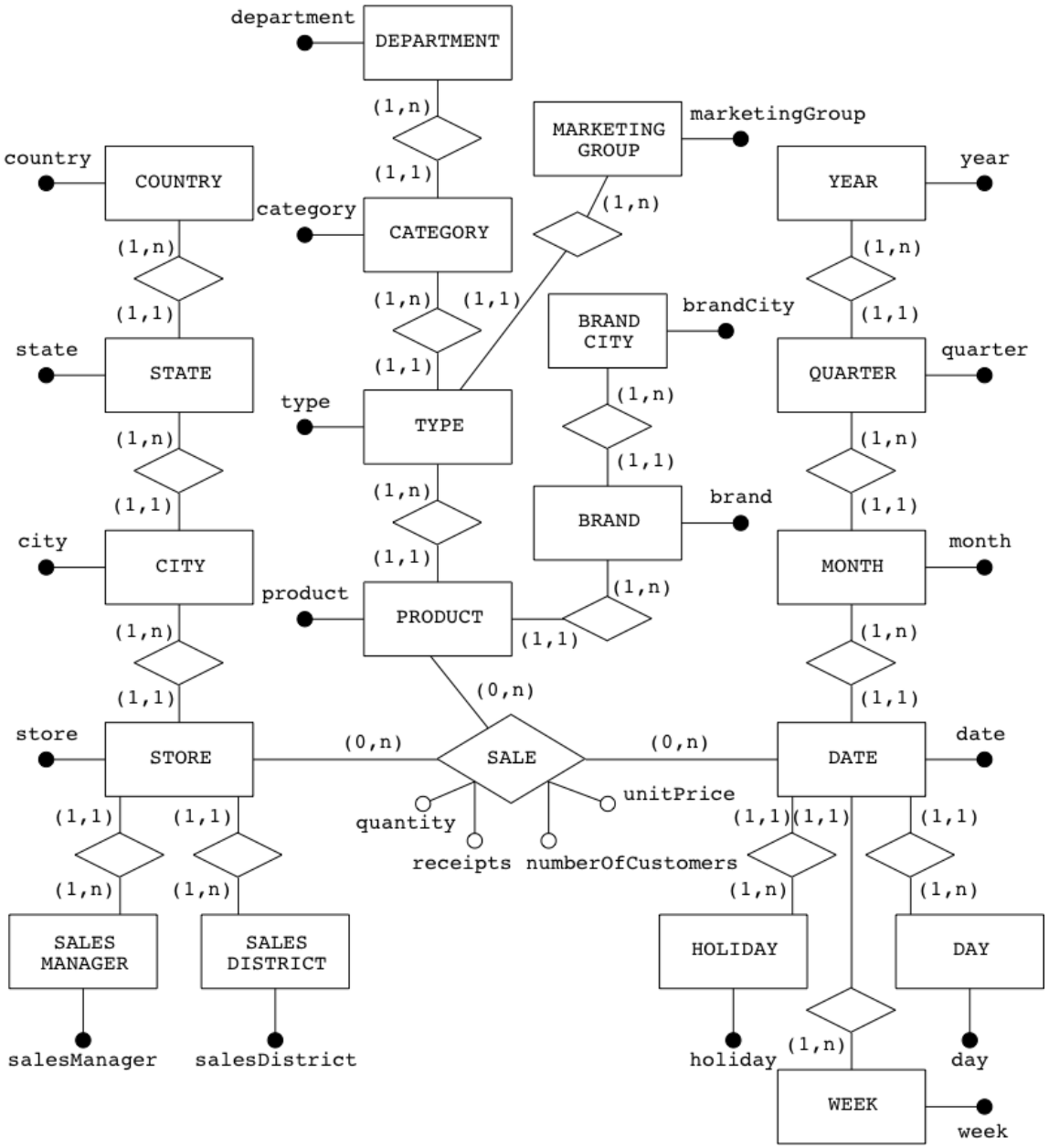
DFM: basic concepts

The general term dimensional attributes stands for the dimensions and other possible attributes, always with discrete values, that describe them (e.g., a product is described by its type, by the category to which it belongs, by its brand, and by the department in which it is sold)

A hierarchy is a directed tree whose nodes are dimensional attributes and whose arcs model many-to-one associations between dimensional attribute pairs. It includes a dimension, positioned at the tree's root, and all of the dimensional attributes that describe it



DFM vs. ERM


department
DEPARTMENT
(1,n)
(1,1)
MARKETING GROUP
marketingGroup
country
COUNTRY
category
CATEGORY
YEAR
year
(1,n)
(1,1)
(1,n)
(1,1)
(1,n)
(1,1)
BRAND CITY
brandCity
state
STATE
QUARTER
quarter
(1,n)
(1,1)
type
TYPE
(1,n)
(1,1)
(1,n)
(1,1)
BRAND
brand
city
CITY
MONTH
month
(1,n)
(1,1)
(1,n)
(1,1)
product
PRODUCT
(1,n)
(1,1)
(1,n)
(1,1)
(0,n)
store
STORE
(0,n)
SALE
(0,n)
DATE
date
unitPrice
quantity
receipts
numberOfCustomers
(1,1)
(1,1)
(1,1)
(1,1)
(1,1)
(1,n)
(1,n)
(1,n)
(1,n)
SALES MANAGER
SALES DISTRICT
HOLIDAY
DAY
salesManager
salesDistrict
holiday
day
(1,n)
WEEK
week

Summarizing

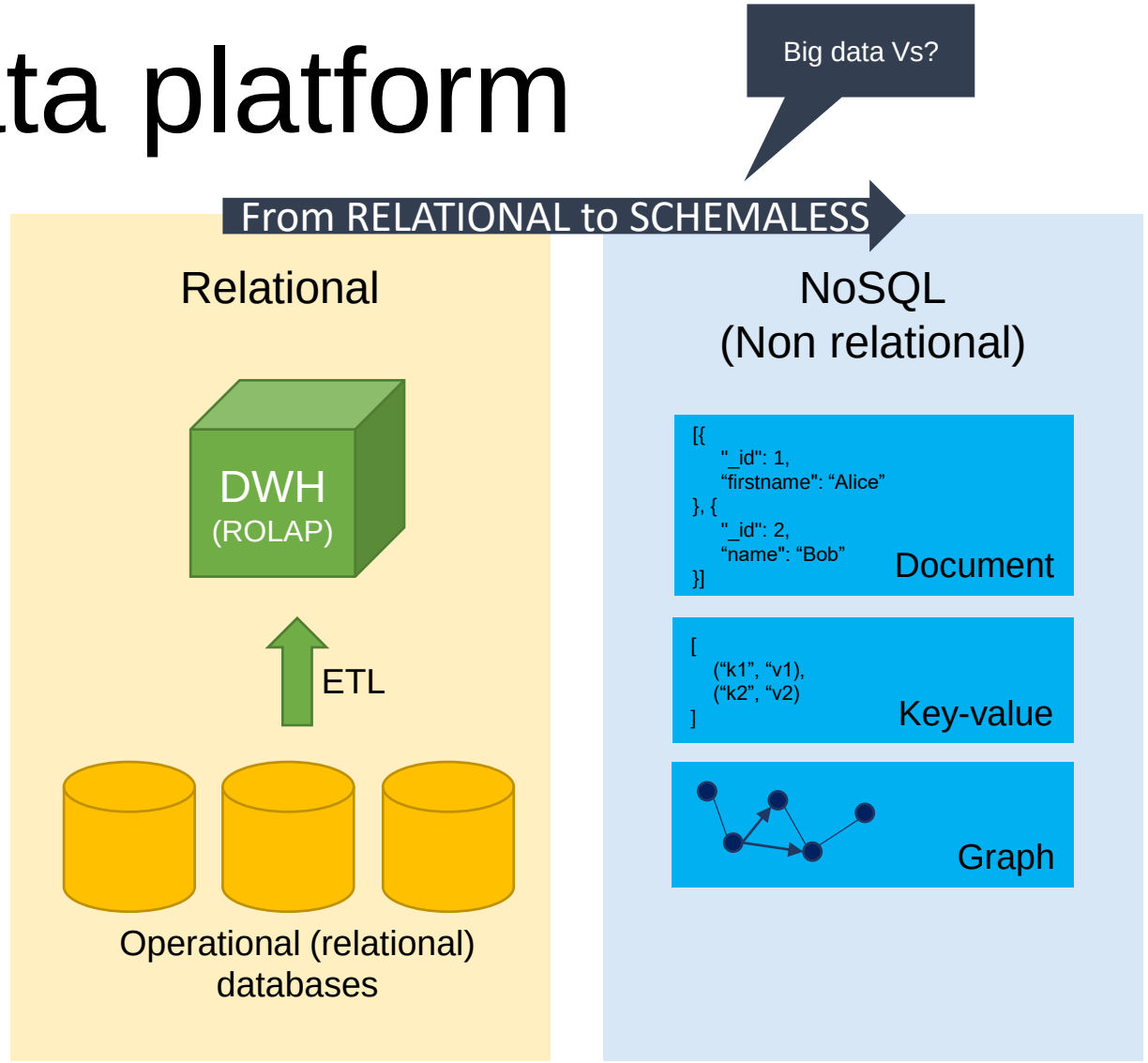
	Operational DBs	*Data warehouses*
users	thousands	hundreds
workload	predefined transactions	*ad hoc* analysis queries
access	to hundreds of records, read and write	to millions of records, mostly read-only
goal	application-dependent	decision support
data	elementary, numeric and alphanumeric	aggregated, mostly numeric
lata integration	application-based	subject-based
quality	in terms of integrity	in terms of consistency
temporal span	current data	current and historical data
update	continuous	periodic
model	normalized	multidimensional
optimization	for OLTP accesses on a fraction of database	for OLAP accesses on a large part of database

Data platform: OLTP vs OLAP

Data platform: OLTP vs OLAP

Characteristic	OLTP	OLAP
Nature	Constant transactions (queries/updates)	Periodic large updates, complex queries
Examples	Accounting database, online retail transactions	Reporting, decision support
Type	Operational data	Consolidated data
Data retention	Short-term (2-6 months)	Long-term (2-5 years)
Storage	Gigabytes (GB)	Terabytes (TB) / Petabytes (PB)
Users	Many	Few
Protection	Robust, constant data protection and fault tolerance	Periodic protection

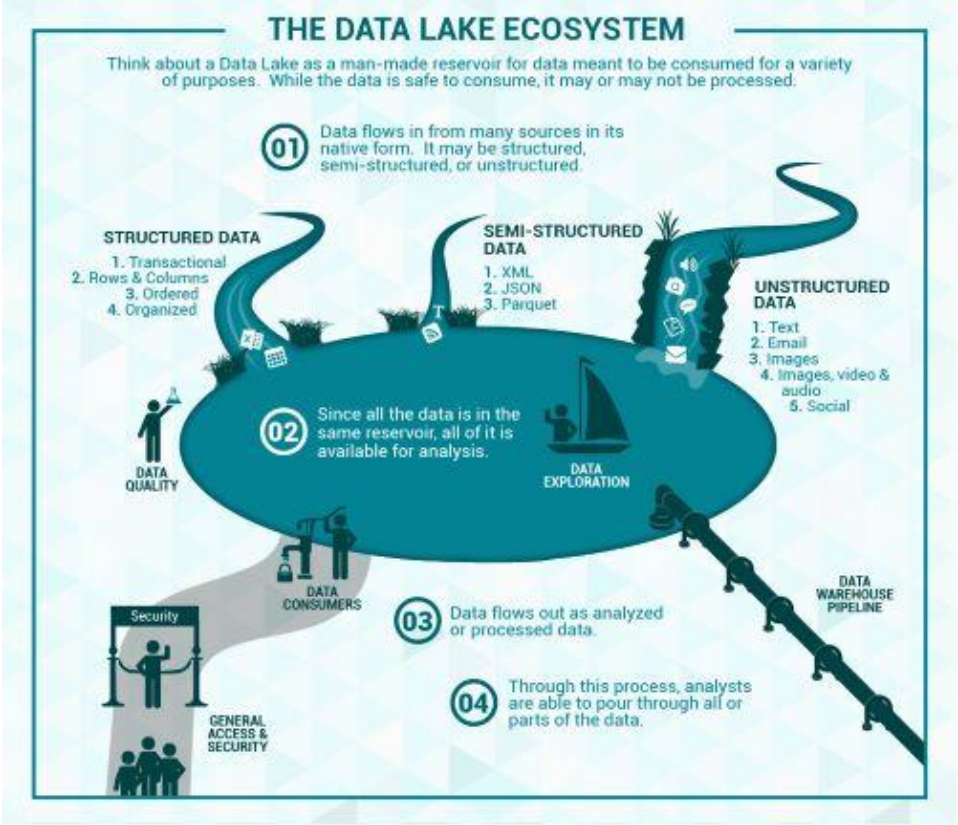
Data platform



Data platform

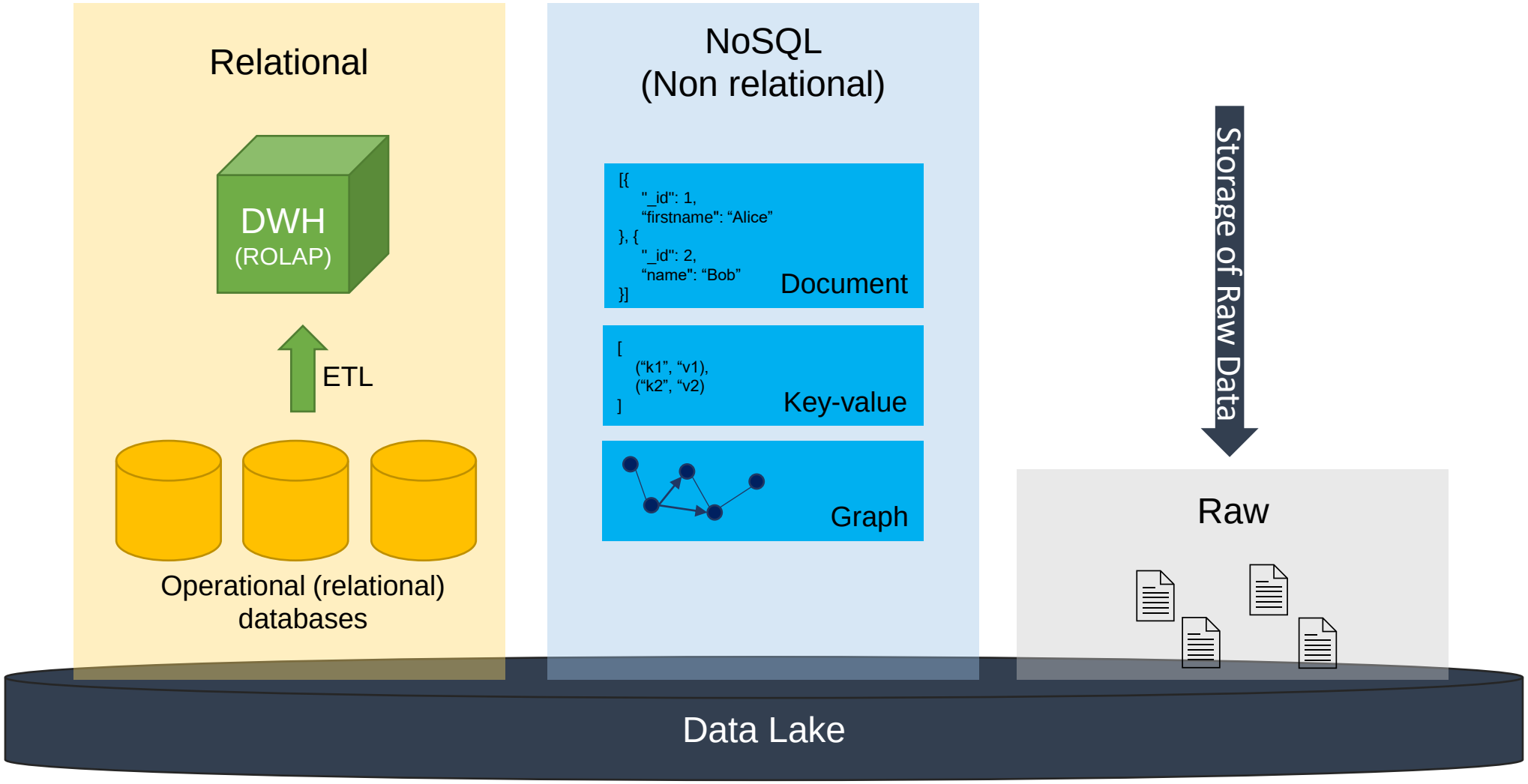
Data lake

Couto et al.: *“A DL is a central repository system for storage, processing, and analysis of raw data, in which the data is kept in its original format and is processed to be queried only when needed. It can store a varied amount of formats in big data ecosystems, from unstructured, semi-structured, to structured data sources”*



Couto, Julia, et al. "A Mapping Study about Data Lakes: An Improved Definition and Possible Architectures." *SEKE*. 2019.
https://dunnsolutions.com/business-analytics/big-data-analytics/data-lake-consulting

Data platform



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 "ETL-ed" 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 platform: DWH vs Data Lake

Data platform: DWH vs Data Lake

Characteristics	Data warehouse	Data lake
Data	Relational	Non-relational and relational
Schema	Designed prior to implementation (schema-on-write)	Written at the time of analysis (schema-on-read)
Price/ performance	Fastest query results using higher cost storage	Query results getting faster using low-cost storage
Data quality	Highly curated data that serves as the central version of the truth	Any data, which may or may not be curated (e.g., raw data)
Users	Business analysts	Data scientists, data developers, and business analysts (using curated data)
Analytics	Batch reporting, BI, and visualizations	Machine learning, predictive analytics, data discovery, and profiling.

Data platform

Data lakes have increasingly taken the role of data hubs

- Eliminate up-front costs of ingestion and ETL since data are stored in original format
- Once in DL, data are available for analysis by everyone in the organization

Drawing a sharp line been storage/computation/analysis is hard

- Is a database just storage?
- What about SQL/OLAP?

Blurring of the architectural borderlines

- DL is often replaced by “data platform” or “data ecosystem”
- Encompass systems supporting data-intensive storage, computation, analysis

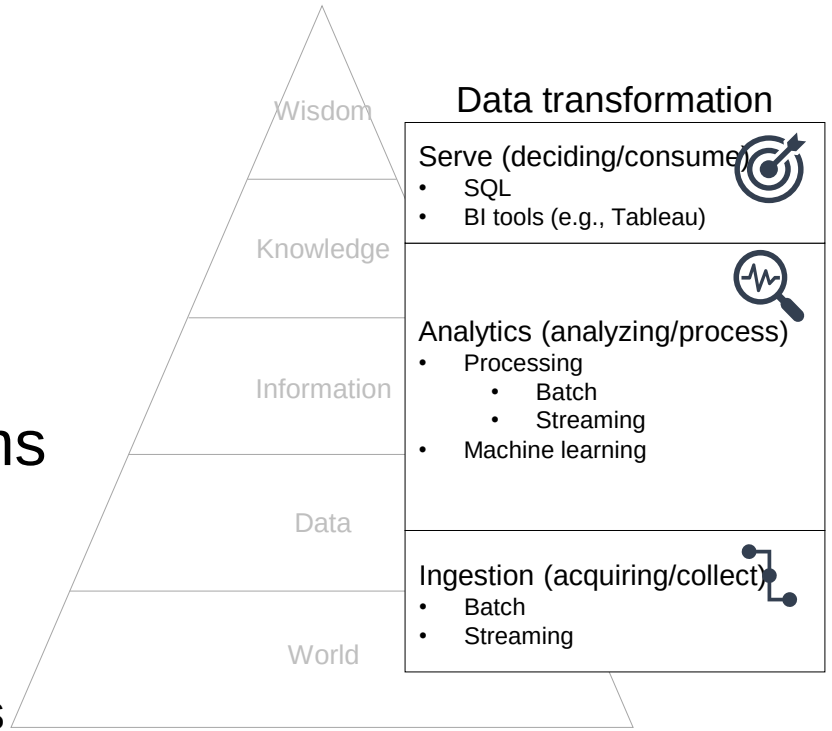
Data platform

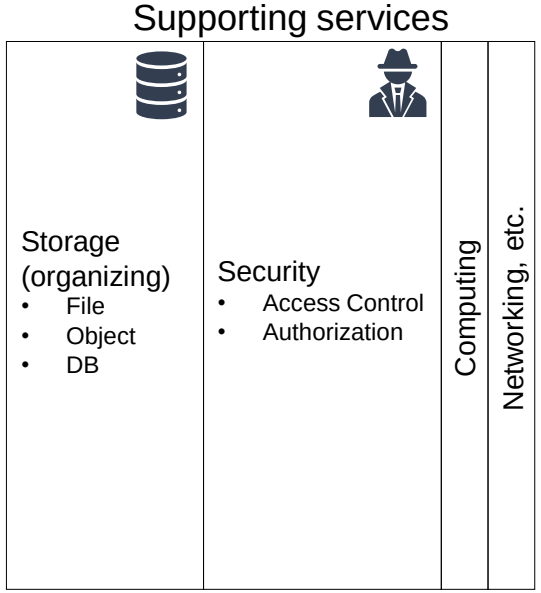
We have services

- To transform data
- To support the transformation

The (DIKW) pyramid abstracts many techniques and algorithms

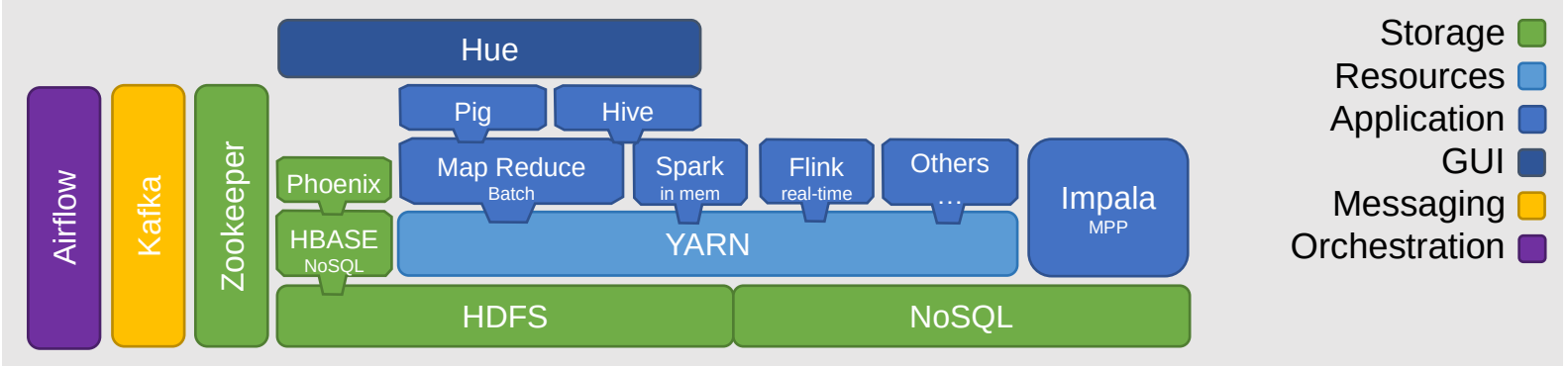
- Standardization
- Integration
- Orchestration
- Accessibility through APIs





Example of data platform: Hadoop-based

A data platform on the Hadoop stack requires several tools



How many levels of complexity are hidden here?

How do you provision it?

- Manual provisioning on-premises
- Semi-automatic provisioning on-premises
- Automatic provisioning in the cloud

Data Lakehouse

Data warehouse architecture as we know today will replaced by a new architectural pattern, the 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
- Data governance
  - All processes ensuring that data meet high quality standards throughout the whole lifecycles
  - Including availability, usability, consistency, integrity, security
- Support for diverse workloads (e.g., data science, ML, SQL, analytics)

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

Data Lakehouse

Data warehouse architecture as we know today will replaced by a new architectural pattern, the Lakehouse

1st generation systems: data warehousing started with helping business leaders get analytical insights

- Data in these warehouses would be written with schema-on-write, which ensured that the data model was optimized for downstream BI consumption
- Several challenges
  - They typically coupled compute and storage into an on-premises appliance
    - This forced enterprises to provision and pay for the peak of user load and data under management, very costly
  - More and more datasets were completely unstructured, e.g., video, audio, and text documents, which data warehouses could not store and query at all

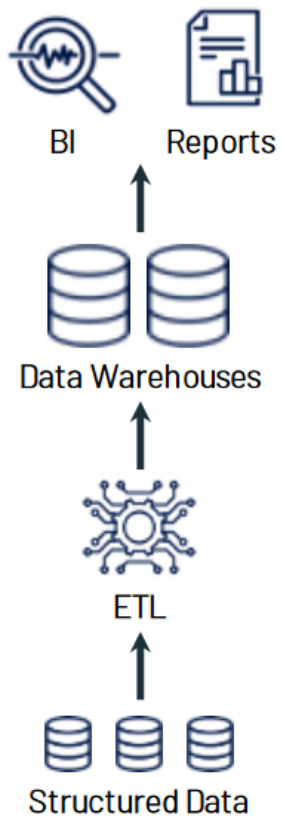
Armbrust, Michael, et al. "Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics." *CIDR*. 2021.

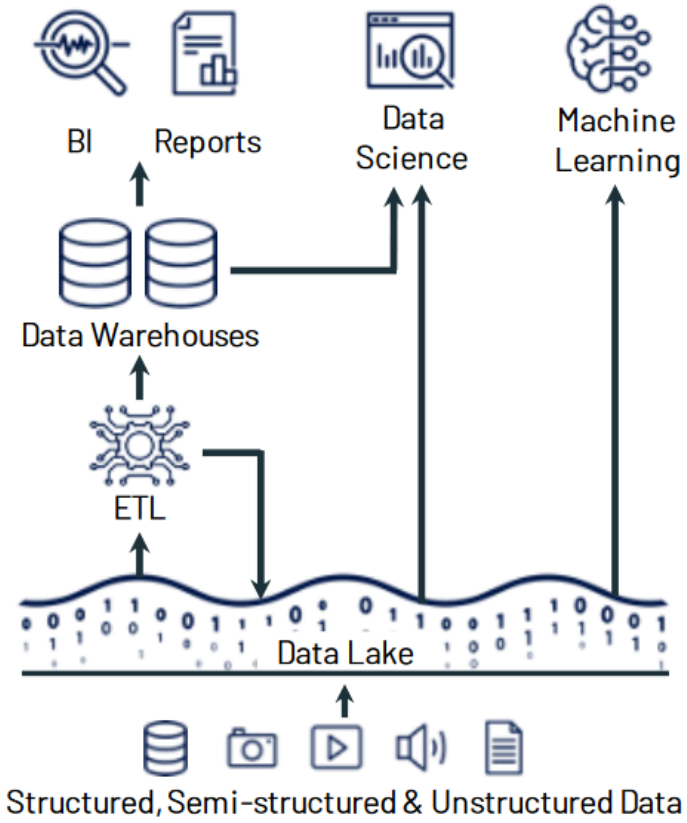
Data Lakehouse

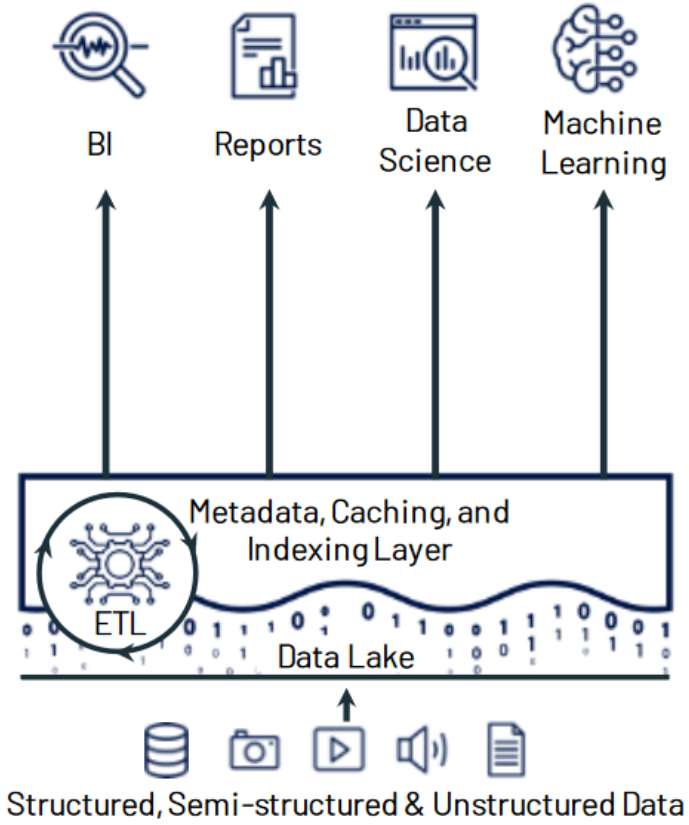
2nd generation: started offloading all the raw data into data lakes

- The data lake is a schema-on-read architecture that enabled the agility of storing any data at low cost, but on the other hand, punted the problem of data quality and governance
- In this architecture, a small subset of data in the lake would later be ETLed to a downstream data warehouse
- The use of open formats also made data lake data directly accessible to a wide range of other analytics engines, such as machine learning systems
- From 2015 onwards, cloud data lakes, such as S3, ADLS and GCS, started replacing HDFS. They have superior durability (often >10 nines), geo-replication, and most importantly, extremely low cost with the possibility of automatic, even cheaper, archival storage, e.g., AWS Glacier

Data Lakehouse







Dataset Search for Data Discovery, Augmentation, and Explanation

- Recent years have seen an explosion in our ability to collect and catalog immense amounts of data about our environment, society, and populace
- Governments, and organizations are increasingly making structured data available on the Web and in various repositories and data lakes
- Combined with advances in analytics and machine learning, the availability of such data should in theory allow us to make progress on many of our most important scientific and societal questions
- This opportunity is often missed due to a central technical barrier: it is currently nearly impossible for domain experts to weed through the vast amount of available information to discover datasets that are needed for their specific application
- While search engines have addressed the discovery problem for Web documents, there are many new challenges involved in supporting the discovery of structured data---from crawling the Web in search of datasets, to the need for dataset-oriented queries and new strategies to rank and display results

Juliana Freire, keynote @ EDBT 2023

Data Lakehouse

While the cloud data lake and warehouse architecture is ostensibly cheap due to separate storage (e.g., S3) and compute (e.g., Redshift), a two-tier architecture is highly complex for users.

- Data is first ETLed into lakes, and then again ELTed into warehouses
- Enterprise use cases now include advanced analytics such as machine learning, for which neither data lakes nor warehouses are ideal
- (Some) main problems:
  - **Reliability**. Keeping the data lake and warehouse consistent is difficult and costly
  - Data **staleness**. The data in the warehouse is stale compared to that of the data lake, with new data frequently taking days to load
  - **Limited support for advanced analytics**. Businesses want to ask predictive questions using their warehousing data, e.g., “which customers should I offer discounts to?” None of the leading machine learning systems directly work well on top of warehouses
    - Process large datasets using complex non-SQL code

Data Lakehouse

Lakehouse is a data management system based on low-cost and directly-accessible storage

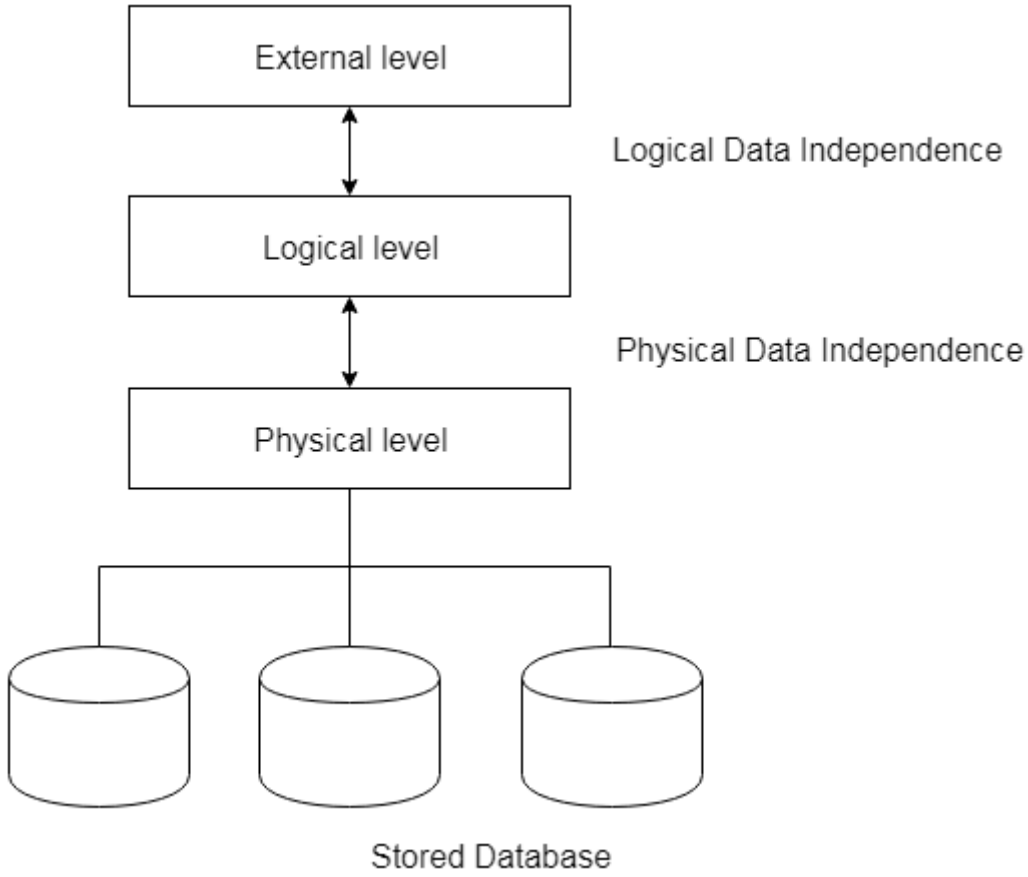
- Combine the key benefits of data lakes and data warehouses: low-cost storage in an open format accessible by a variety of systems from the former, and powerful management and optimization features from the latter.
- ACID transactions, data versioning, auditing, indexing, caching, and query optimization.

Key question: can we combine these benefits in an effective way?

- Direct access means that they **give up some aspects of data independence**, which has been a cornerstone of relational DBMS design
- **Lakehouses are an especially good fit for cloud environments with separate compute and storage**: different computing applications can run on-demand on completely separate computing nodes (e.g., a GPU cluster for ML) while directly accessing the same storage data

Data Independence

- Data independence can be explained using the three-schema architecture
- Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level



Data Lakehouse

- Have the system store data in a low-cost object store (e.g., Amazon S3) using a standard file format such as Apache Parquet
- Implement a transactional metadata layer on top of the object store that defines which objects are part of a table version.
- Implement management features such as ACID transactions or versioning within the metadata layer
- Although a metadata layer adds management capabilities, it is not sufficient to achieve good SQL performance
  - Data warehouses use several techniques to get state-of-the-art performance, such as storing hot data on fast devices such as SSDs, maintaining statistics, building efficient access methods such as indexes, and co-optimizing the data format and compute engine
  - In a Lakehouse based on existing storage formats, it is not possible to change the format, but it is possible to implement other optimizations that leave the data files unchanged, including caching, auxiliary data structures such as indexes and statistics, and data layout optimizations

Delta Lake

Achieving performant and mutable table storage over data lakes and object storage is challenging, making it difficult to implement data warehousing capabilities over them

- Most cloud object stores are merely key-value stores, with no cross-key consistency
- The most common way to store relational datasets in cloud object stores is using columnar file formats such as Parquet and ORC, where each table is stored as a set of objects (Parquet “files”), possibly clustered into “partitions” by some fields (e.g., separate objects by date)

However, it creates both correctness and performance challenges for more complex workloads

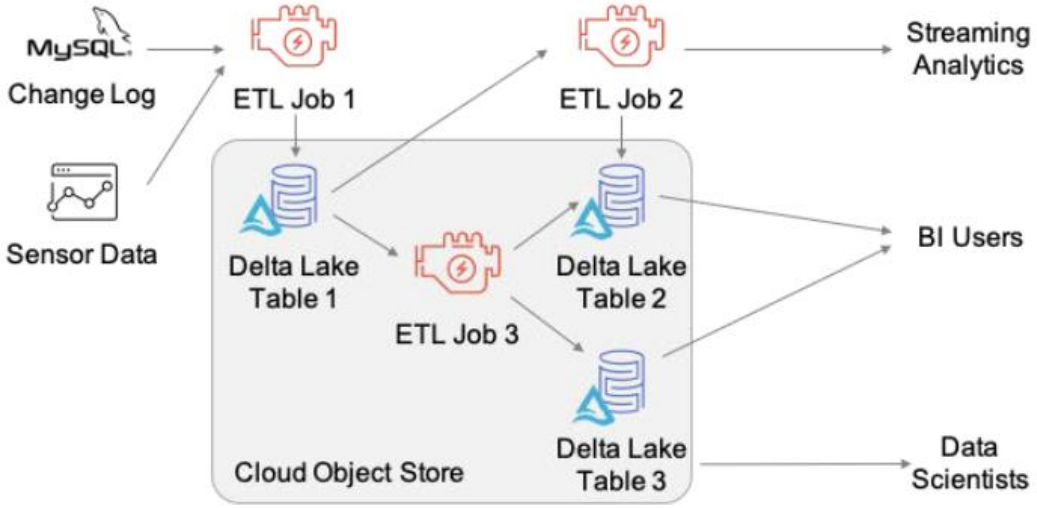
- **Multi-object updates are not atomic**, there is no isolation between queries: for example, if a query needs to update multiple objects in the table readers will see partial updates as the query updates each object individually
- For large tables with millions of objects, **metadata operations are expensive**. The latency of cloud object stores is so much higher that these data skipping checks can take longer than the actual query

Armbrust, Michael, et al. "Delta lake: high-performance ACID table storage over cloud object stores." Proceedings of the VLDB Endowment 13.12 (2020): 3411-3424.

Delta Lake

Delta Lake uses a transaction log that is compacted into Apache Parquet format to provide ACID properties, time travel, and significantly faster metadata operations for large tabular datasets (e.g., the ability to quickly search billions of table partitions for those relevant to a query)

- The log is stored in the _delta_log subdirectory within the table
- It contains a sequence of JSON objects with increasing, zero-padded numerical IDs to store the log records, together with occasional checkpoints for specific log objects that summarize the log up to that point



Delta Lake

Each log record object (e.g., 000003.json) contains an array of actions to apply to the previous version of the table to generate the next one

Examples of actions are:

- Change Metadata
- Add or Remove Files

It is necessary to compress the log periodically into checkpoints

- Checkpoints store all the non-redundant actions in the table's log up to a certain log record ID, in Parquet format
- Some sets of actions are redundant and can be removed Read the _last_checkpoint object in the table's log directory, if it exists, to obtain a recent checkpoint ID.

Delta Lake

Example of a write transaction

- Identify a log record ID (i.e., looking forward from the last checkpoint ID). The transaction will then read the data at table version r (if needed) and attempt to write log record r + 1
- Read data at table version r, if required combine previous checkpoint and further log records
- Write any new data objects that the transaction aims to add to the table into new files in the correct data directories, generating the object names using GUIDs.
  - This step can happen in parallel
  - At the end, these objects are ready to reference in a new log record.
- Attempt to write the transaction's log record into the r + 1 .json log object, if no other client has written this object. **This step needs to be atomic**. If the step fails, the transaction can be retried; depending on the query's semantics
- Optionally, write a new .parquet checkpoint for log record r + 1

Creating the r + 1 .json record, needs to be atomic: only 1 client should succeed. Not all large-scale storage systems have an atomic put operation

- Google Cloud Storage and Azure Blob Store support atomic put-if-absent operations
- HDFS, we use atomic renames to rename a temporary file to the target name
- Amazon S3 need ad-hoc protocols

Lakehouse

Implement SQL optimizations in a Lakehouse independent of the chosen data format

Format-independent optimizations are

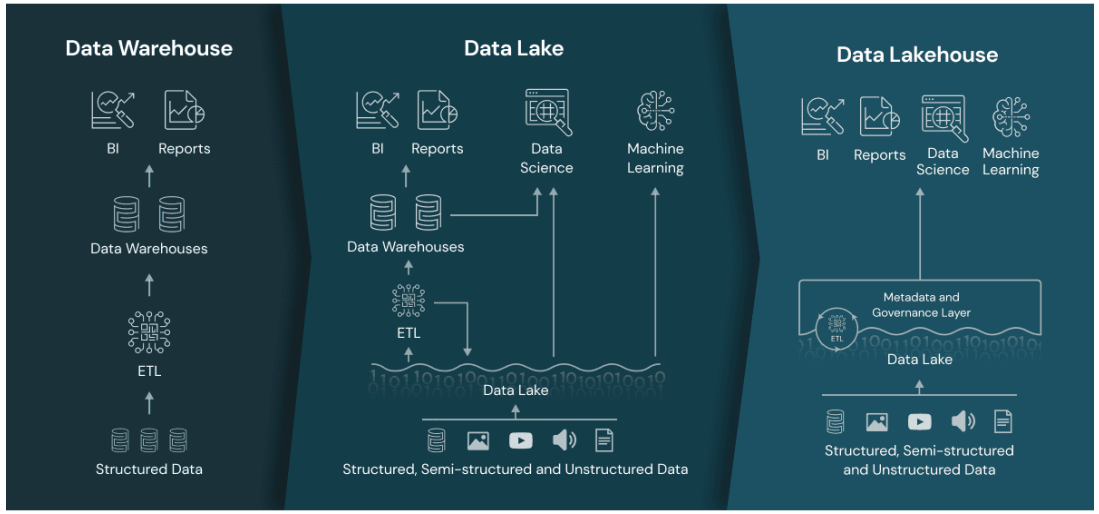
- **Caching**: When using a transactional metadata layer such as Delta Lake, it is safe for a Lakehouse system to cache files from the cloud object store on faster storage devices such as SSDs and RAM on the processing nodes
- **Auxiliary data**: maintain column min-max statistics for each data file in the table within the same Parquet file used to store the transaction log, which enables data skipping optimizations when the base data is clustered by particular columns
- **Data layout**:
  - record ordering: which records are clustered together and hence easiest to read together, e.g. ordering records using individual dimensions or space-filling curves such as Z-order
  - compression strategies differently for various groups of records, or other strategies [28].

One approach that we've had success with is offering a declarative version of the DataFrame APIs used in these libraries, which maps data preparation computations into Spark SQL query plans and can benefit from logical optimizations

Data lakehouse

Data lakehouse

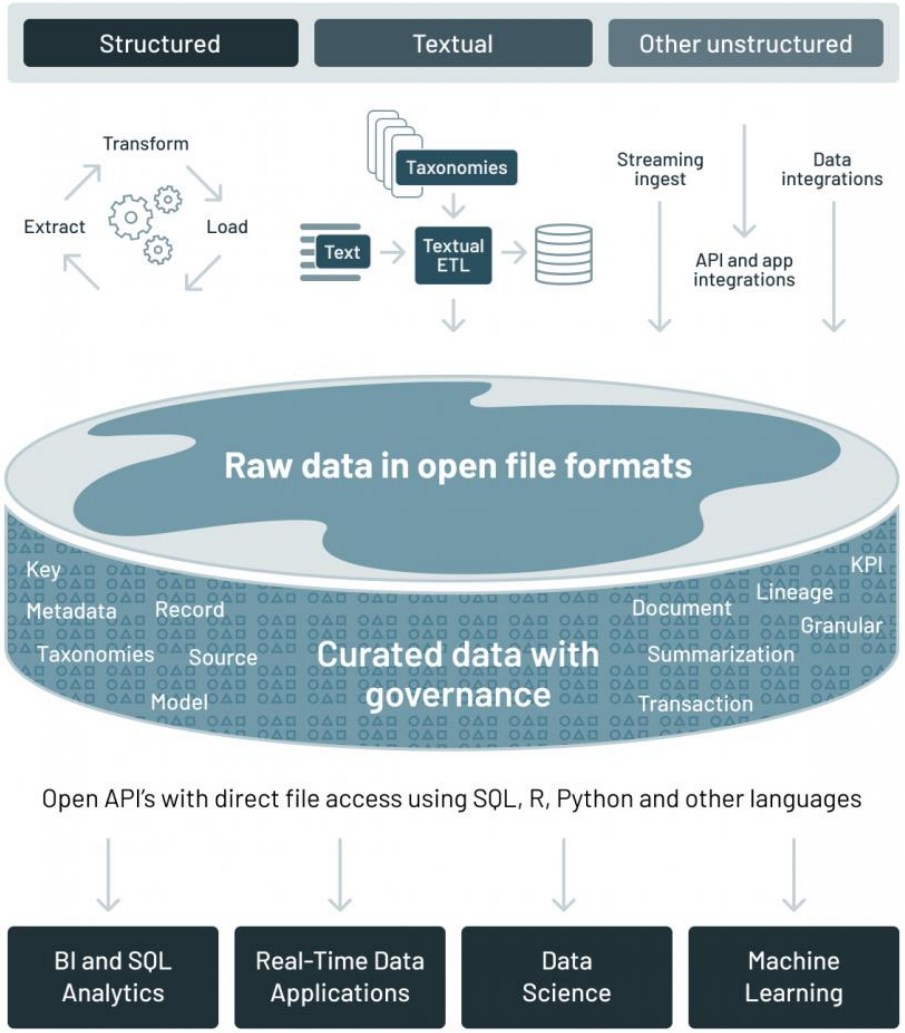
- Data management architecture that combines the flexibility, cost-efficiency, and scale of data lakes with the data management and ACID transactions of data warehouses, enabling business intelligence (BI) and machine learning (ML) on all data
- Vendor lock in



https://www.databricks.com/glossary/data-lakehouse

Data lakehouse

	Data warehouse	Data lake	Data lakehouse
Data format	Closed, proprietary format	**Open format** (e.g., Parquet)	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

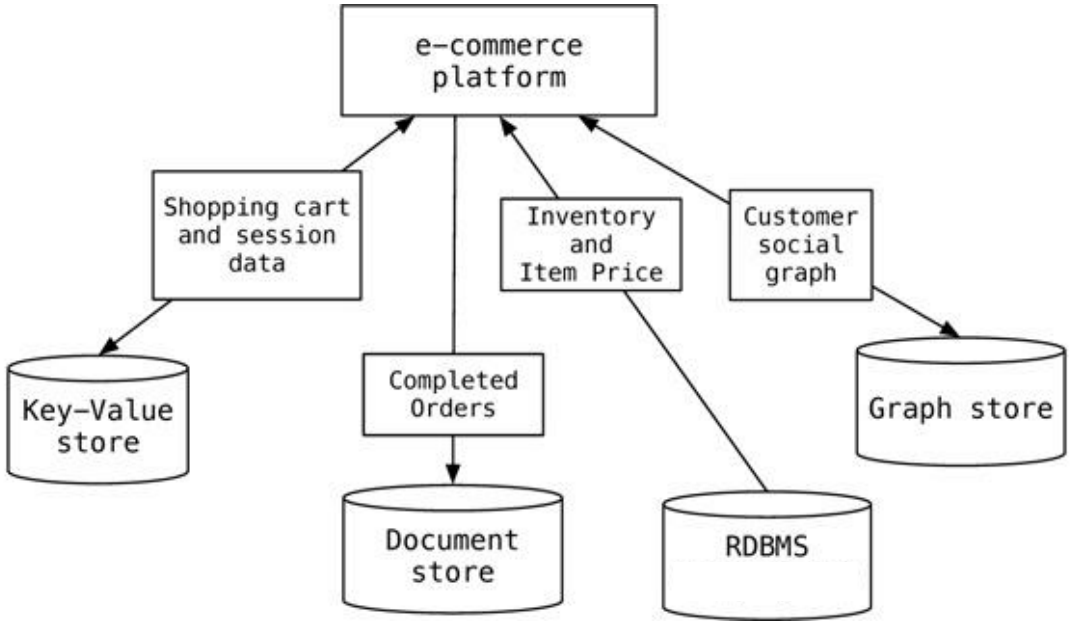


https://databricks.com/blog/2021/05/19/evolution-to-the-data-lakehouse.html

Polyglot Persistence

Polyglot persistence

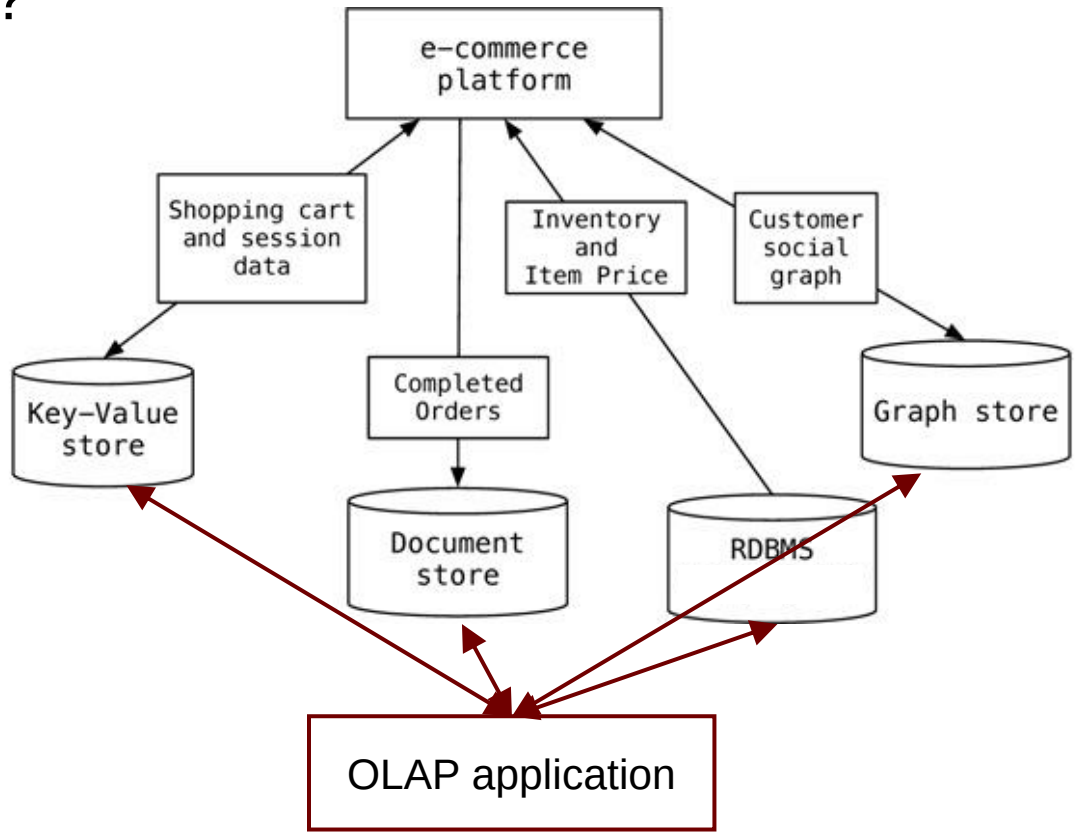
To each application the appropriate DBMS



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

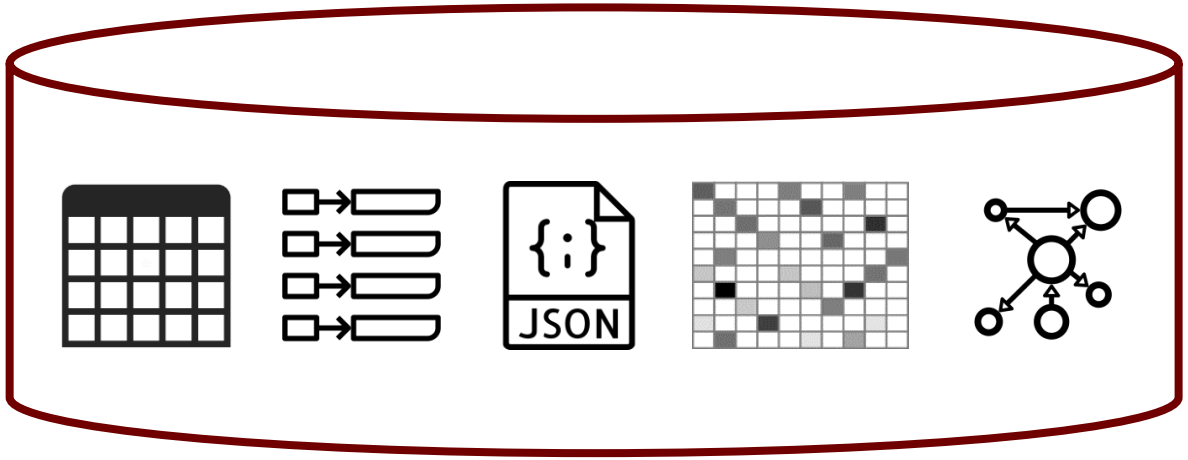
Schema heterogeneity

- 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.

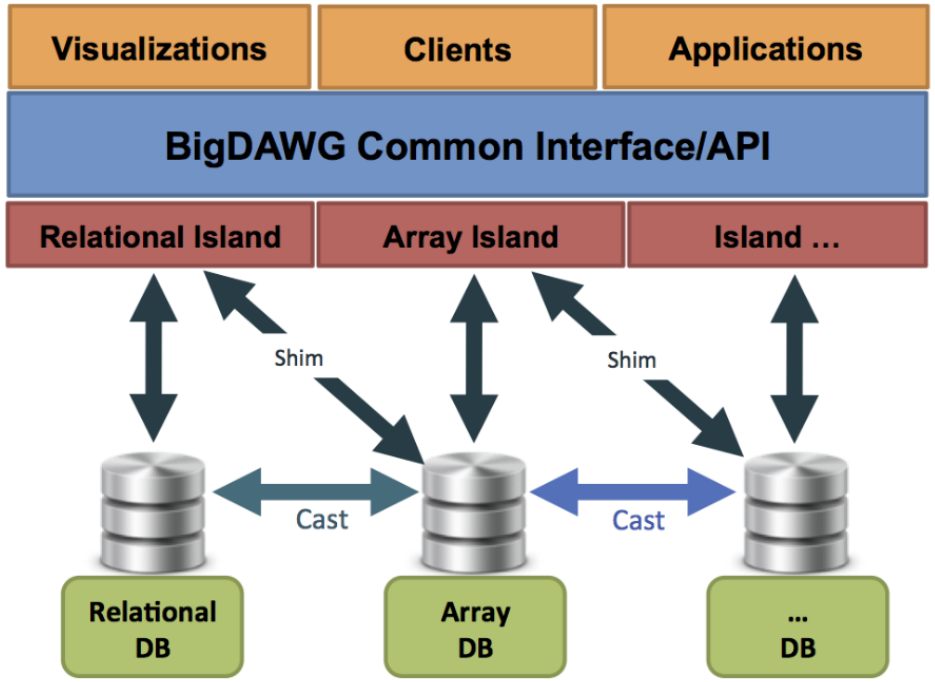
Advanced solutions

The challenge is to balance two often conflicting forces.

- **Location Independence**: A query is written and the system figures out which storage engine it targets
- **Semantic Completeness**: A query can exploit the full set of features provided by a storage engine

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

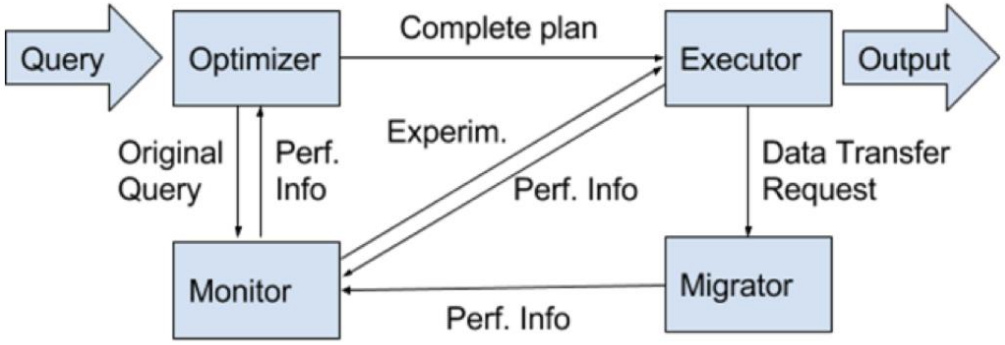


Vijay Gadepally, Kyle O'Brien, Adam Dziedzic, Aaron J. Elmore, Jeremy Kepner, Samuel Madden, Tim Mattson, Jennie Rogers, Zuohao She, Michael Stonebraker: Version 0.1 of the BigDAWG Polystore System. CoRR abs/1707.00721 (2017)

Advanced solutions

BigDAWG middleware consists of

- **Optimizer**: parses the input query and creates a set of viable query plan trees with possible engines for each subquery
- **Monitor**: uses performance data from prior queries to determine the query plan tree with the best engine for each subquery
- **Executor**: figures out how to best join the collections and then executes the query
- **Migrator**: moves data from engine to engine when the plan calls for such data motion



… and of course we have metadata

- **Catalog**: stores metadata about the system
  - Databases: Databases, their engine membership, and connection authentication information.
  - Objects: Data objects (i.e., tables), field-names, and object-to-database membership.

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

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 “out-of-scope”

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
    - E. 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

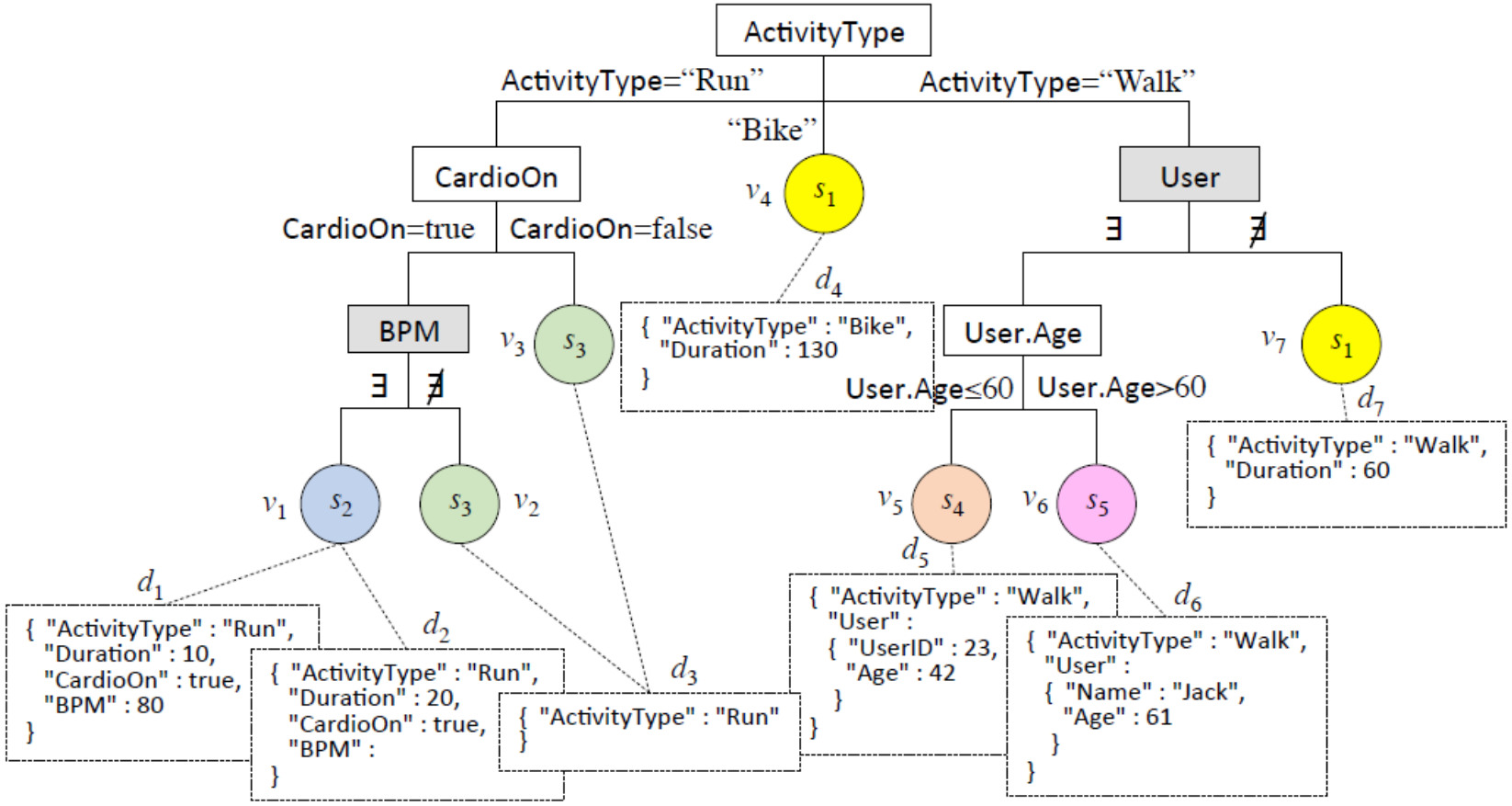
Documents / Observations

Schema / Class

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

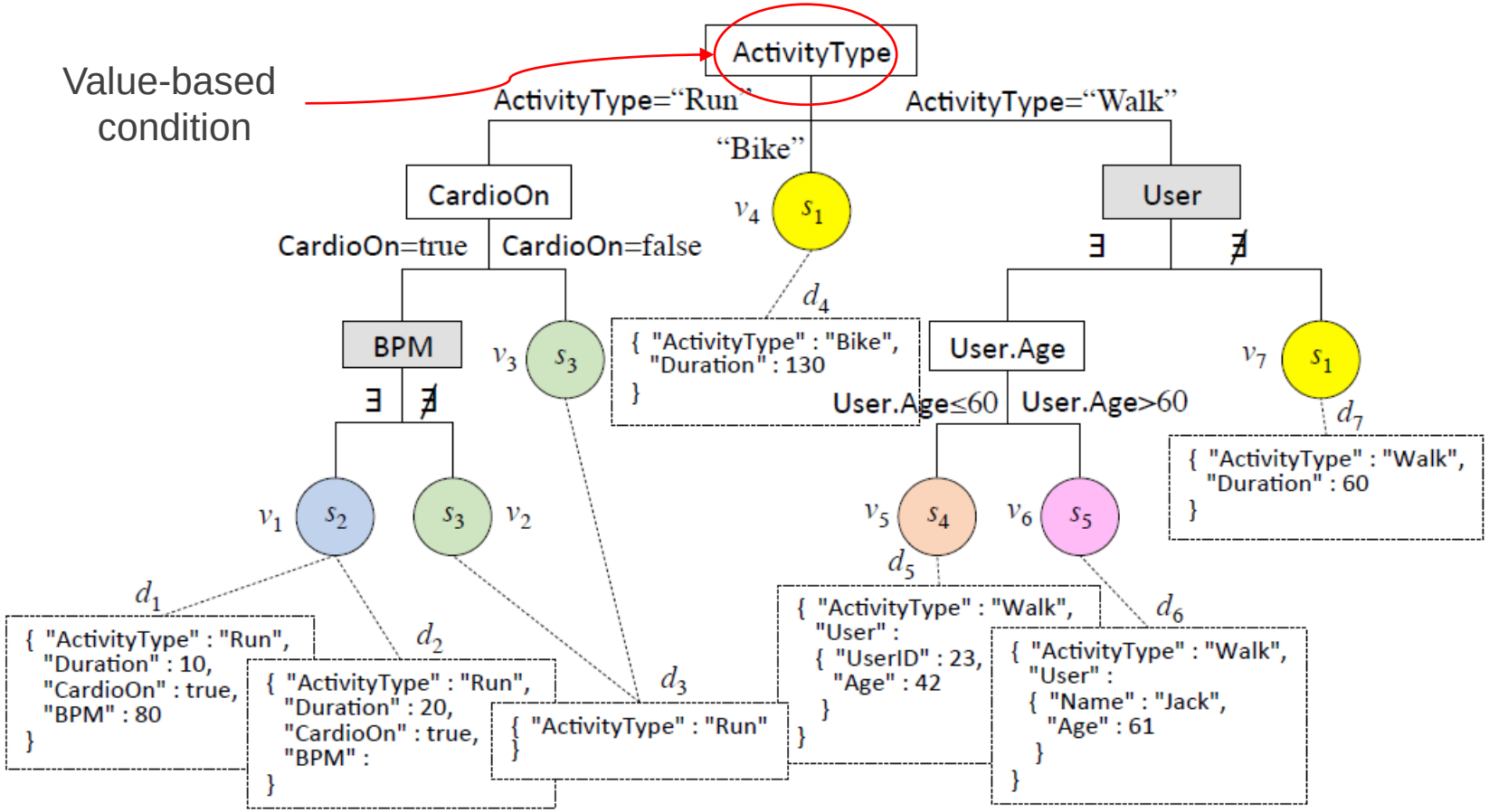
- Classifiers are also used to describe the rules for assigning a class to an observation based on the other observation features
- Based on the requirements collected from potential users, **decision trees** emerged as the most adequate

Schema profiling

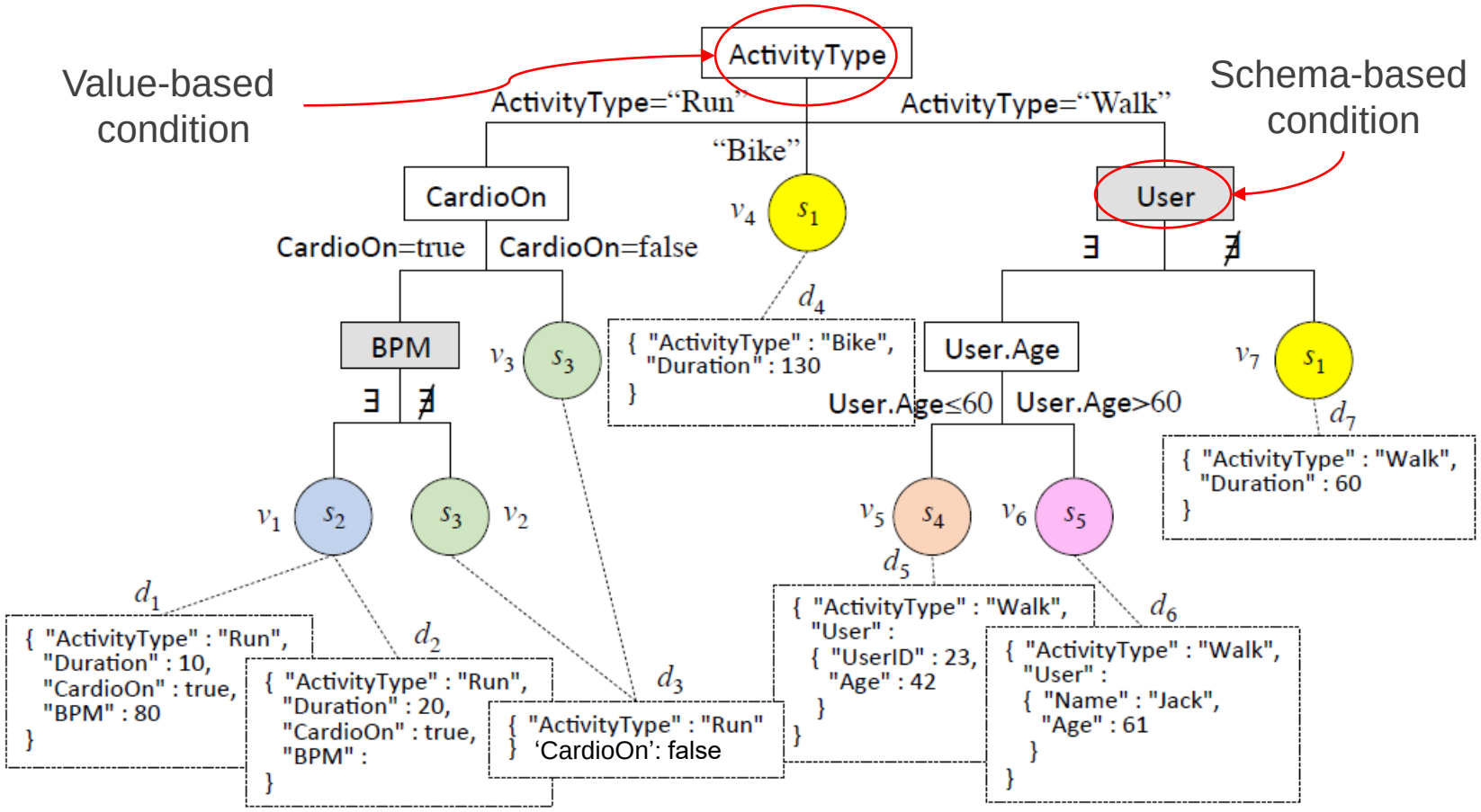


The **documents** are the observations
The **schema** are the classes

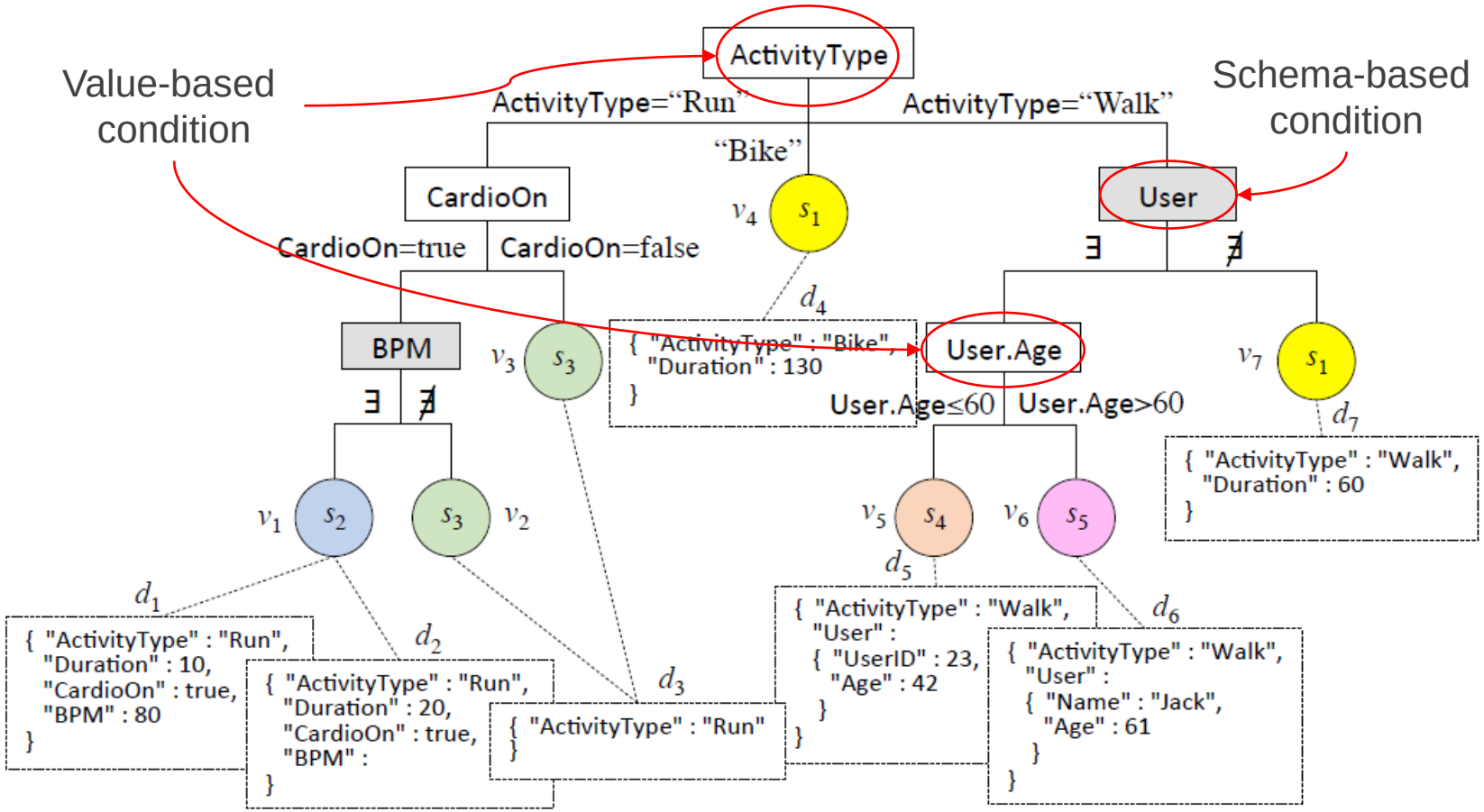
Schema profiling



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

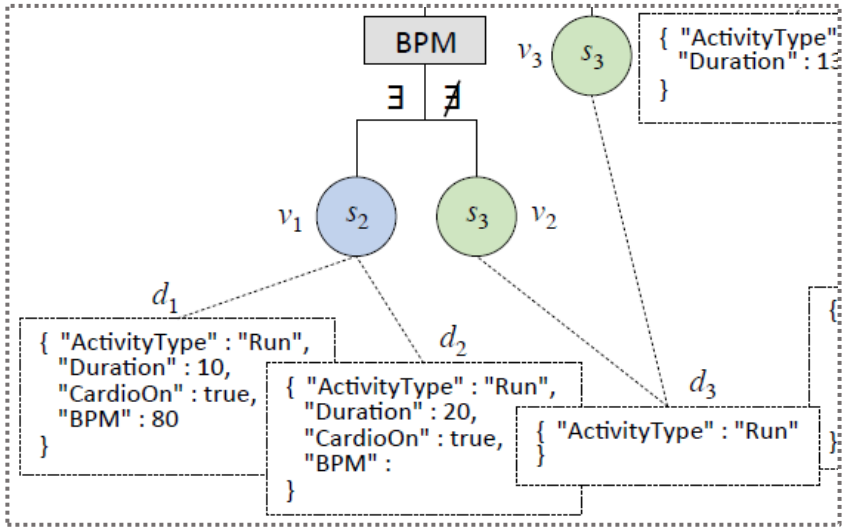
Definition of schema profile characteristics

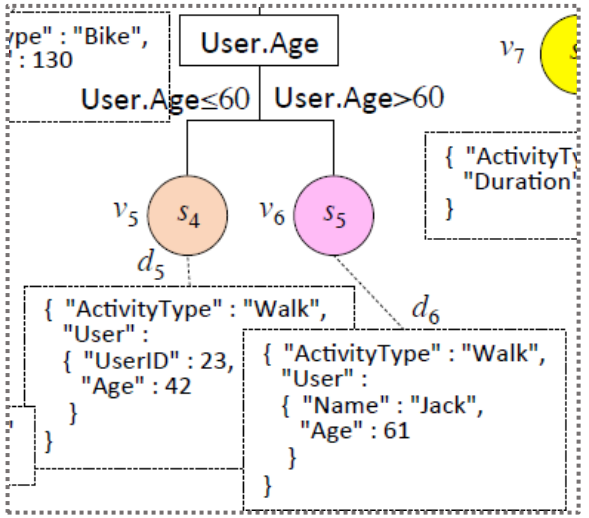
- Explicativeness
- Precision
- Conciseness

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

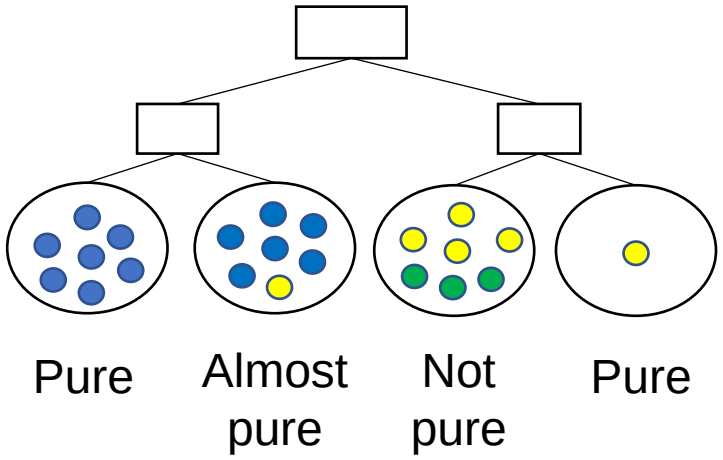
Precision

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



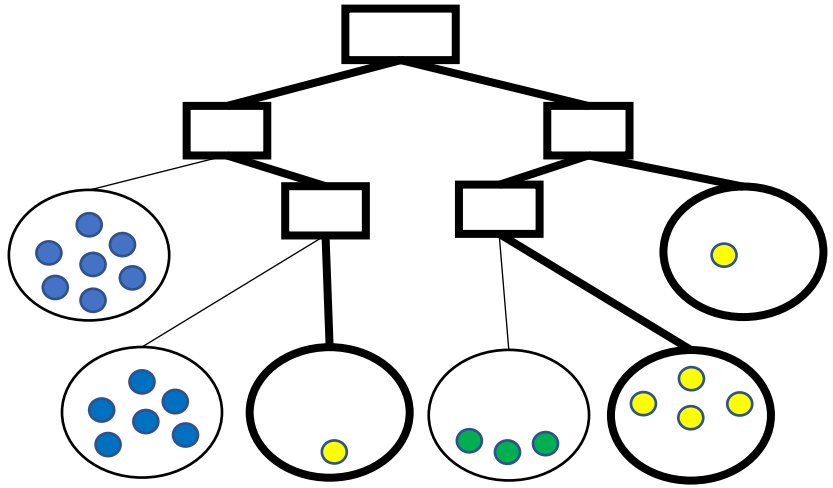
$$entropy(v_j) = -\sum_{s \in S(D_v)} \frac{|D_{v_j}|_s}{|D_{v_j}|} \log \frac{|D_{v_j}|_s}{|D_{v_j}|}$$

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**

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

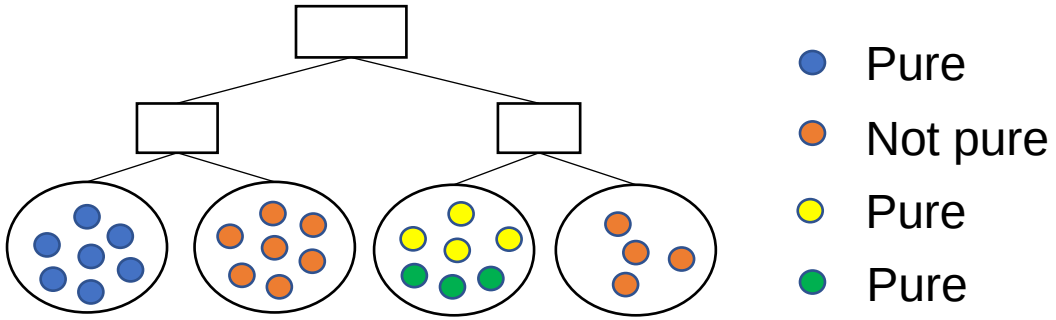
Conciseness

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

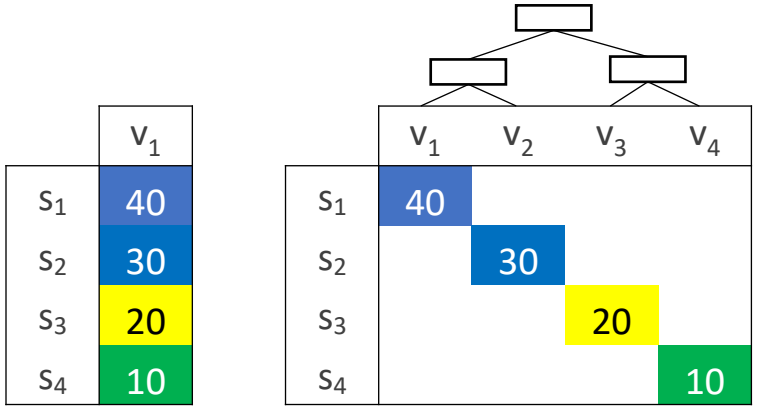
- Entropy:
  a leaf is pure if
  it contains only documents
  with the **same class**

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



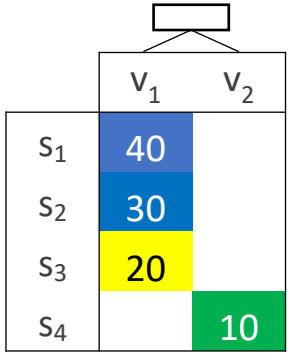
Schema profiling example

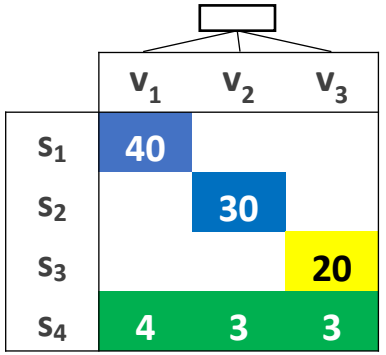
Starting situation
E = 1,85 (maximum)
SE = 0 (minimum)



Best outcome
E = 0
SE = 0

E = 1,38
SE = 0





E = 0,46
SE = 0,16

Schema profiling algorithm

Introduced the notion of *schema entropy loss*

$$loss(\sigma_a(v_j)) = sEntropy(T') - sEntropy(T)$$

Defined a criterion for comparing two splits in the decision tree

Definition 9. *Given two splits $\sigma(v)$ and $\sigma'(v)$ (possibly on different attributes), we say that $\sigma(v)$ is* better *than $\sigma'(v)$ (denoted $\sigma(v) \prec \sigma'(v)$) if either (i) $loss(\sigma(v)) < loss(\sigma'(v))$, or (ii) $loss(\sigma(v)) = loss(\sigma'(v))$ and $gain(\sigma(v)) > gain(\sigma'(v))$.*

Querying the data

One thing is understanding 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)

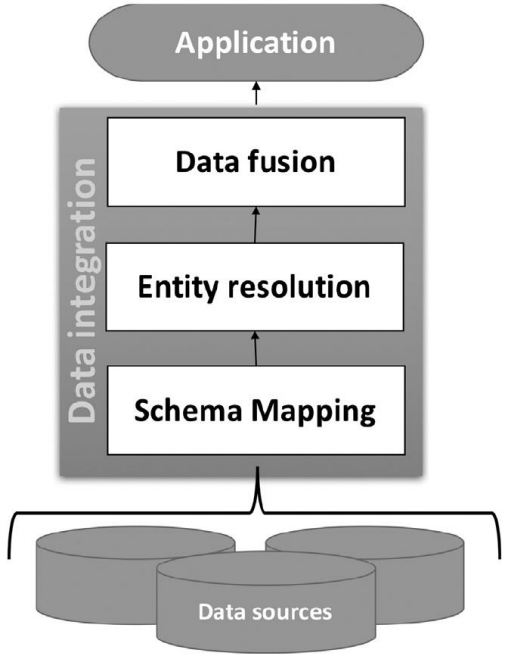
Integration techniques

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

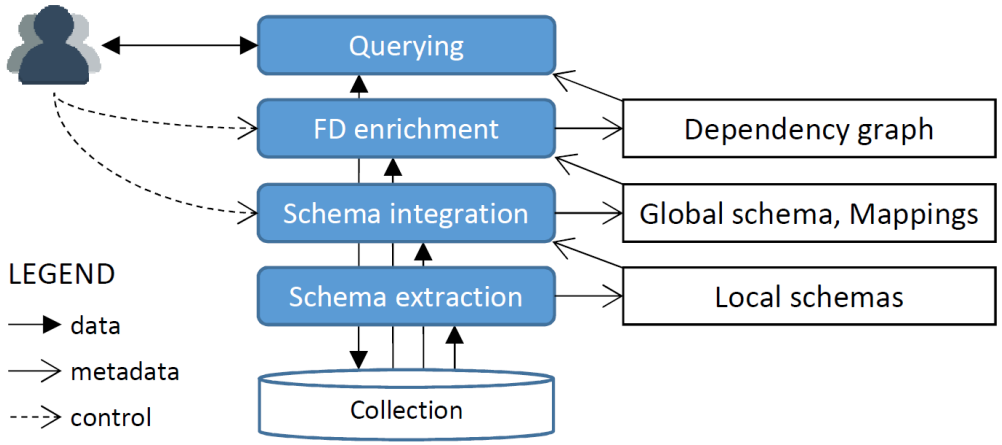


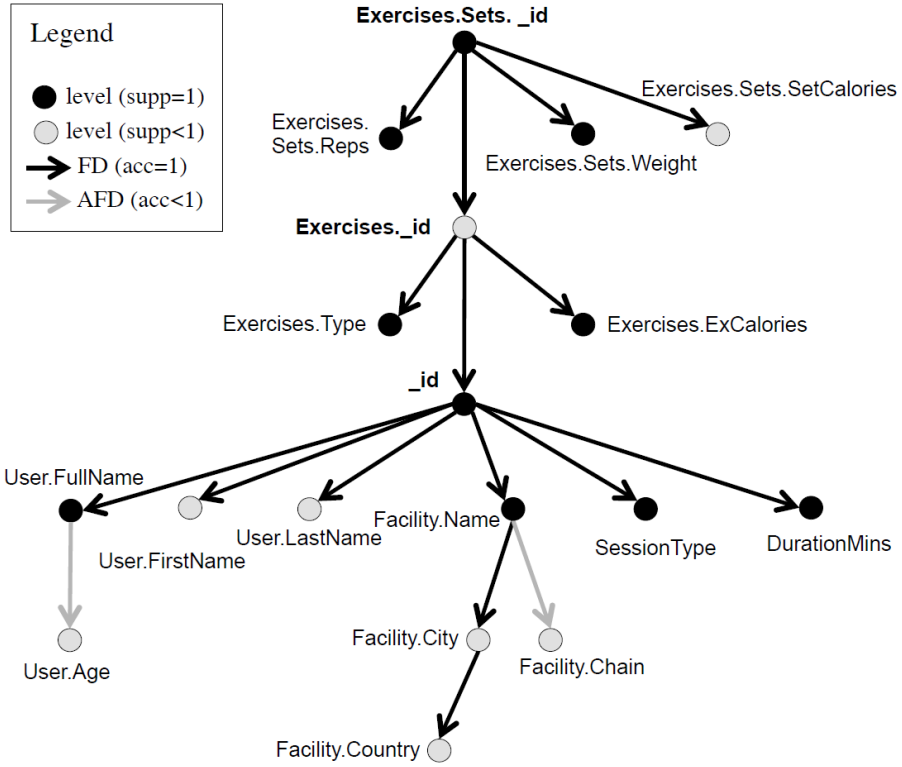
E. Rahm, P.A. Bernstein, **A survey of approaches to automatic schema matching**, *VLDB J.* 10 (4) (2001)

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

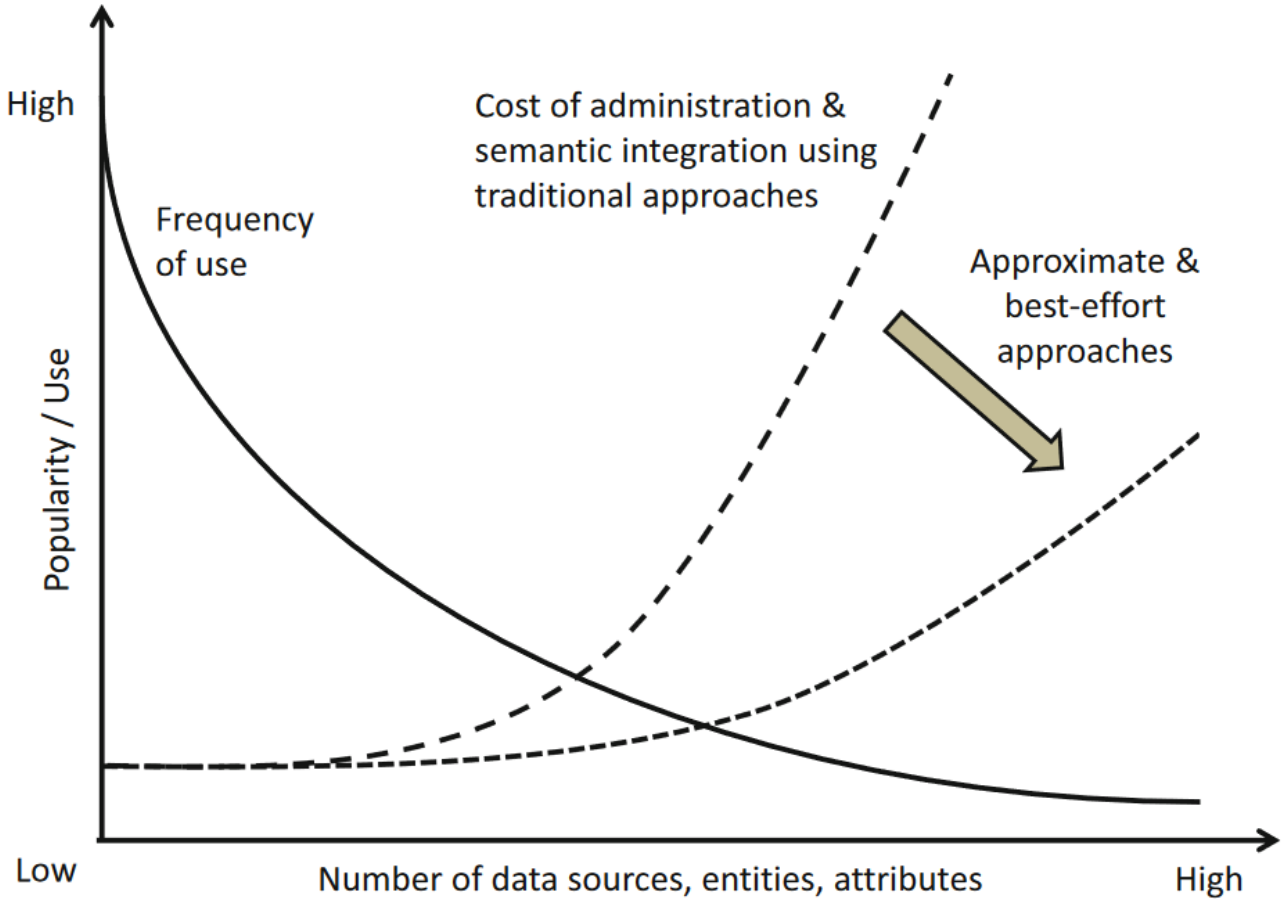
- Expensive querying
  - Does not scale well with the number of schemas
- Expensive integration
  - High levels of heterogeneity imply complex rewriting rules (requiring knowledge and time)
  - Assuming to be *always* able to obtain a global schema is a bit pretentious

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



Curry, E. (2020). Dataspaces: Fundamentals, Principles, and Techniques. *Real-time Linked Dataspaces: Enabling Data Ecosystems for Intelligent Systems*, 45-62.

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).

Curry, E. (2020). Dataspaces: Fundamentals, Principles, and Techniques. *Real-time Linked Dataspaces: Enabling Data Ecosystems for Intelligent Systems*, 45-62.

New integration techniques

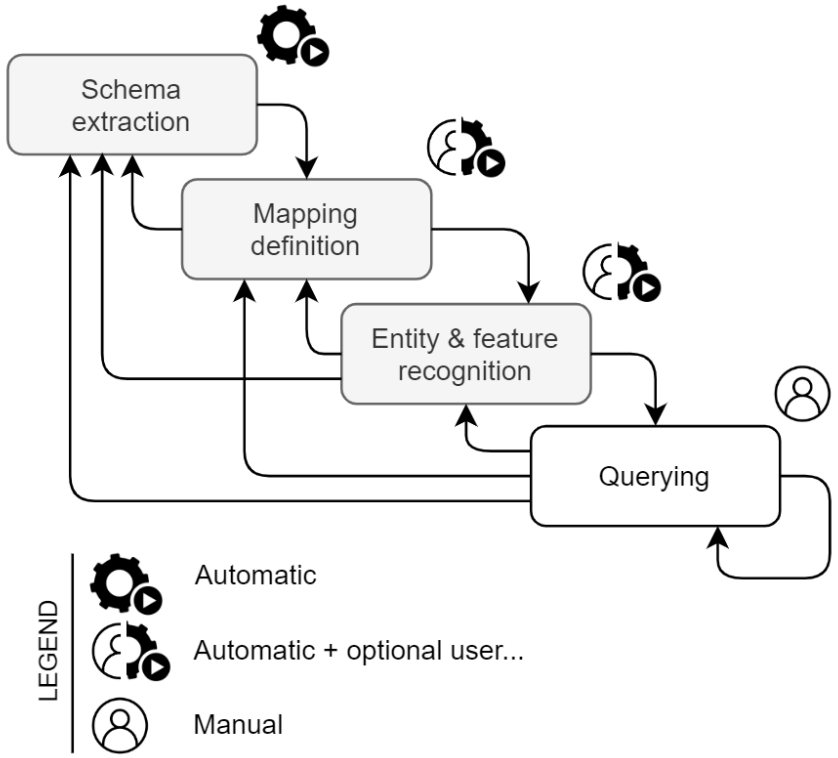
Introducing new concepts

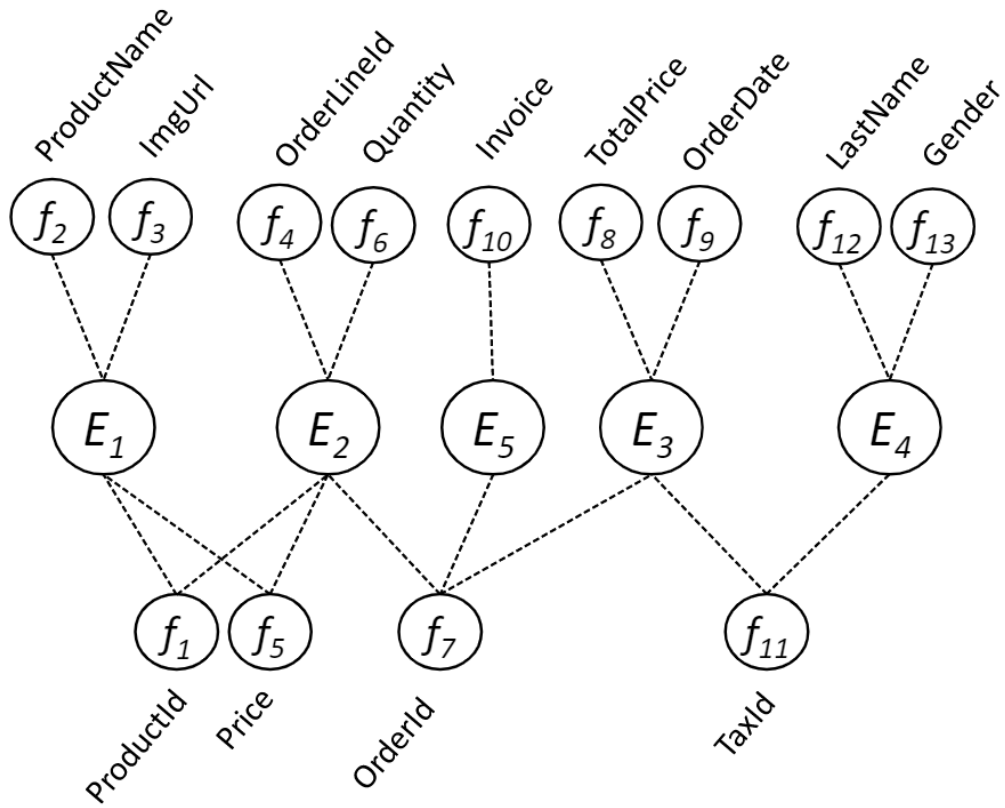
- Entities: representation of a real-world entity
  - E.g., customers, products, orders, etc.
- Features: univocal representation of a group of semantically equivalent attributes
  - E.g., CustomerName = { S1.name, S2.fullname, S3.customer, S4.cName, … }
  - Mapping functions must be defined/definable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

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

New OLAP querying

What it looks like



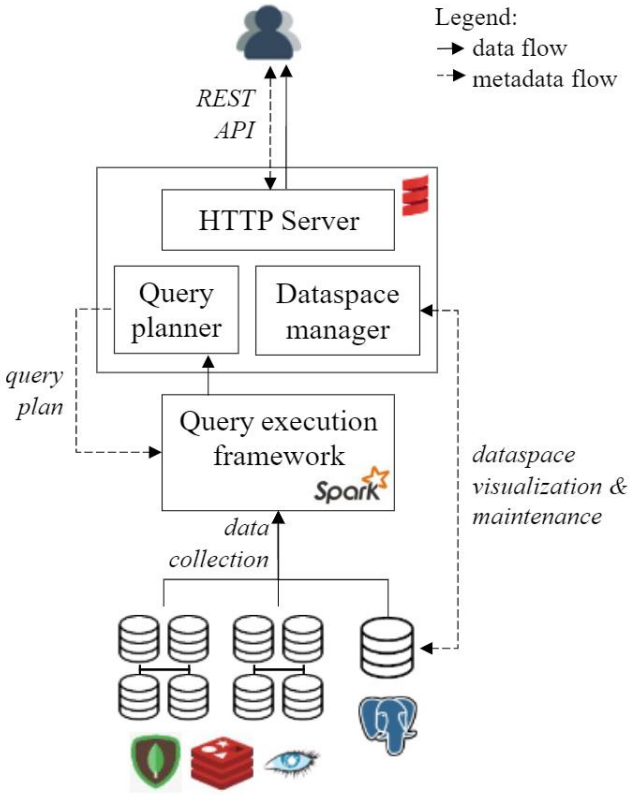


Forresi, C., Gallinucci, E., Golfarelli, M., & Hamadou, H. B. (2021). **A dataspace-based framework for OLAP analyses in a high-variety multistore**. *The VLDB Journal*, *30*(6), 1017-1040.

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 inconsistency

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

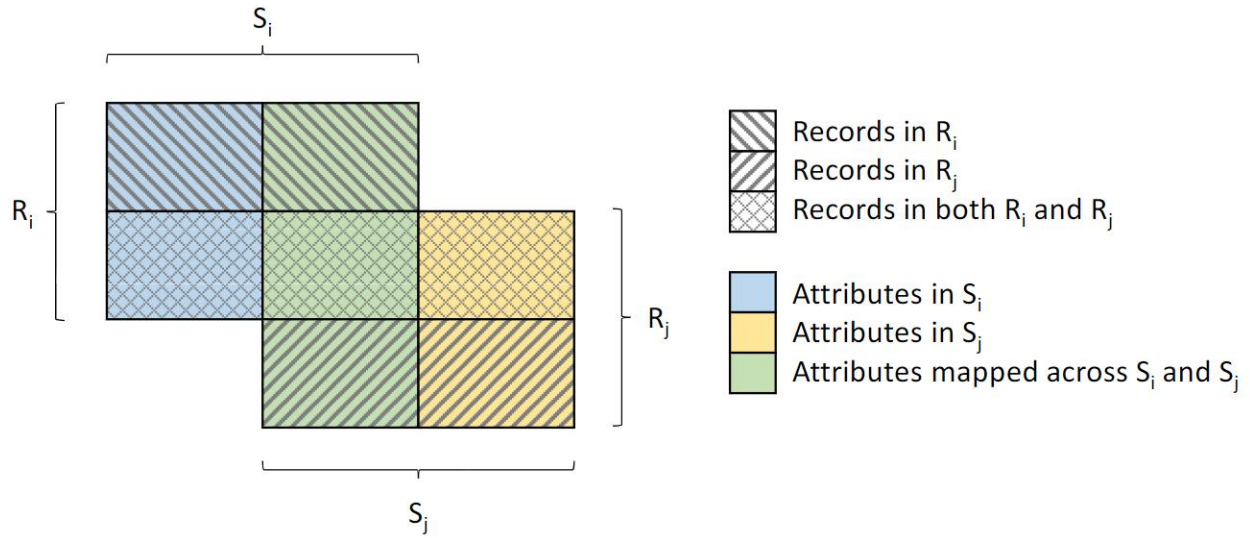
Solutions?

- Traditional ETL
- Solve inconsistencies on-the-fly

Data fusion

Merge operator

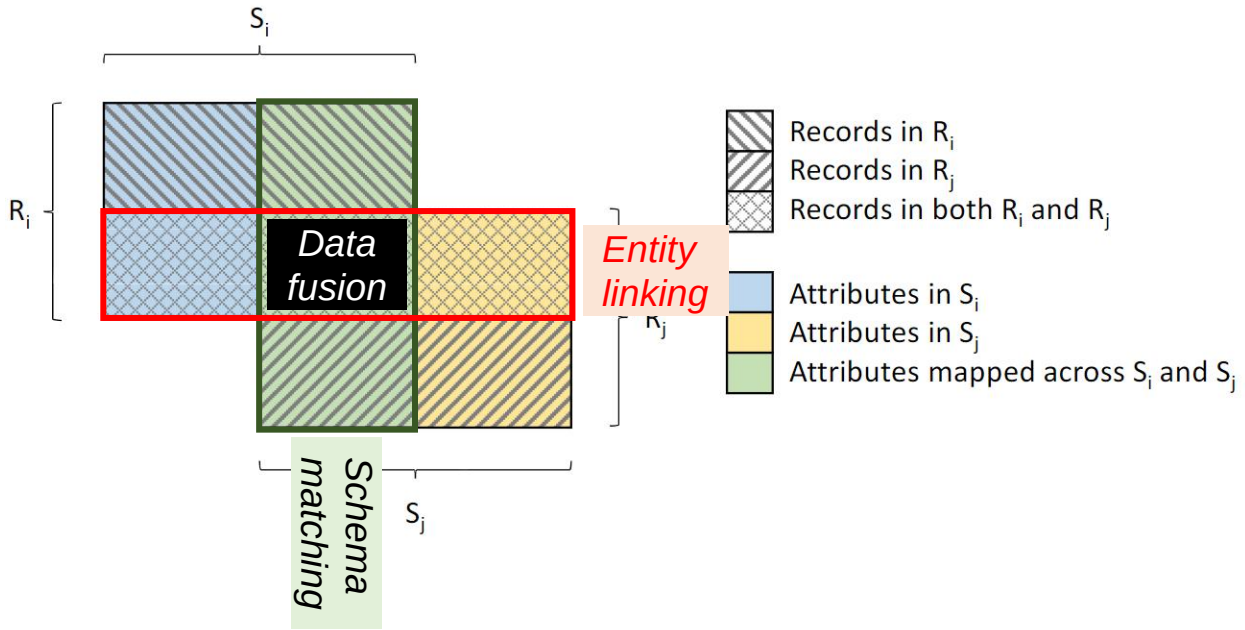
- Originally introduced as “full outer join merge”
  - Naumann, F., Freytag, J. C., & Leser, U. (2004). **Completeness of integrated information sources**. *Information Systems*, *29*(7), 583-615.
- Aims to keep as much information as possible when joining the records of two schemas
  - Avoid any loss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



Data fusion

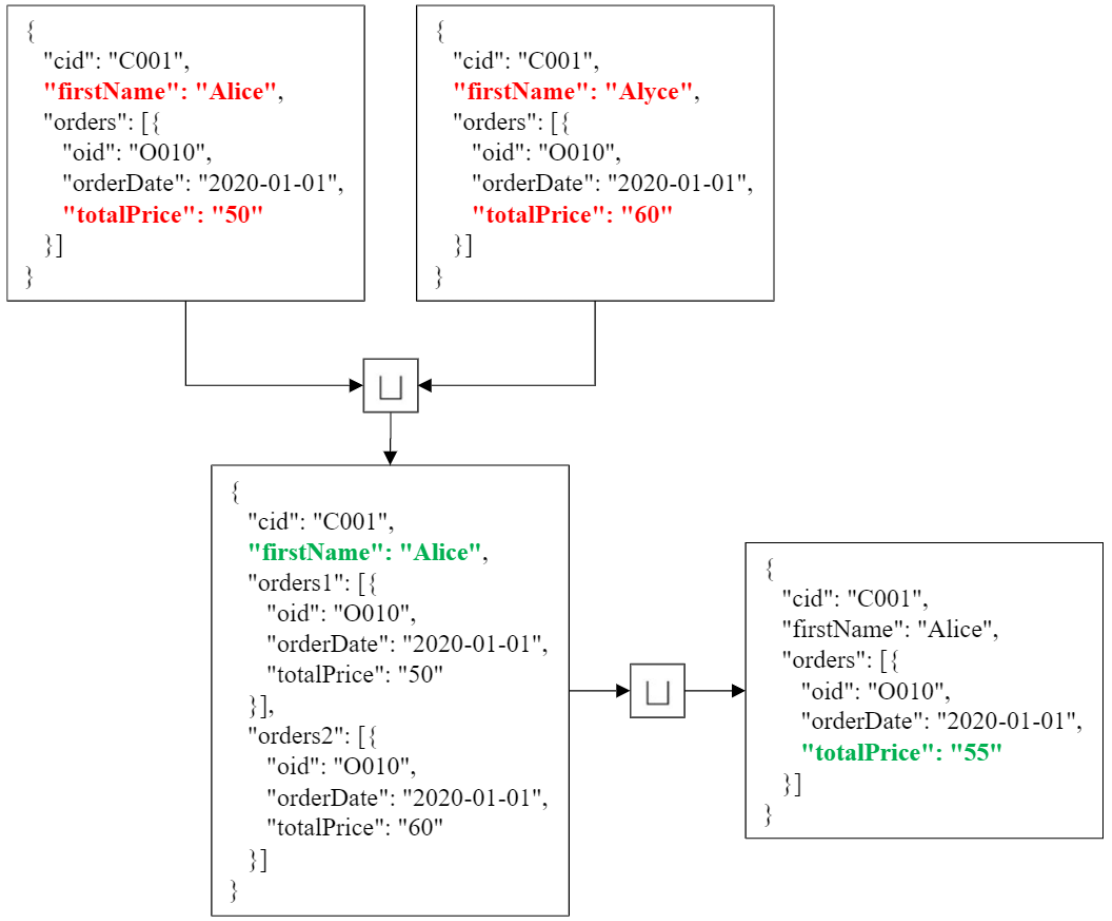
Merge operator

- 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



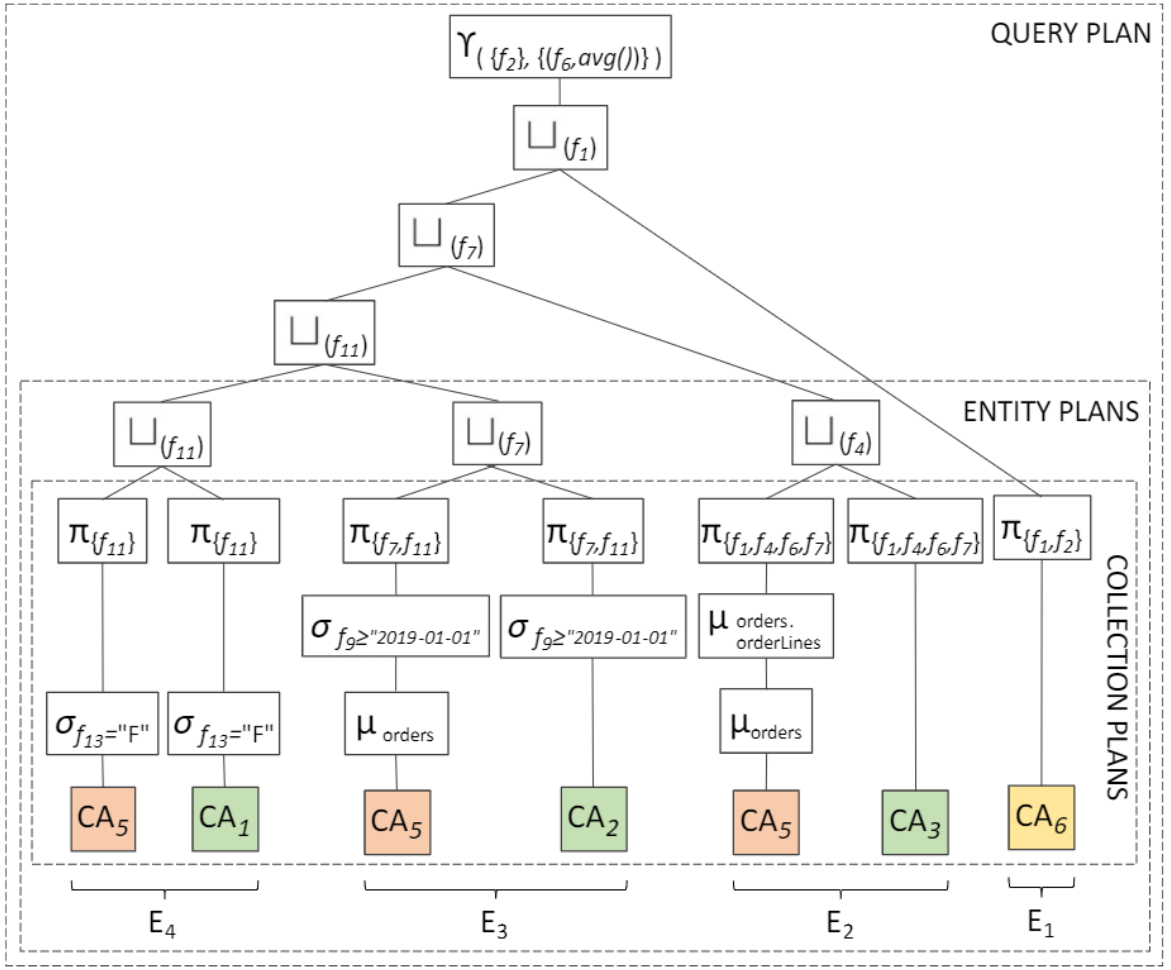
On-the-fly data fusion

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? Not yet!



On-the-fly data fusion

Main issue: performance

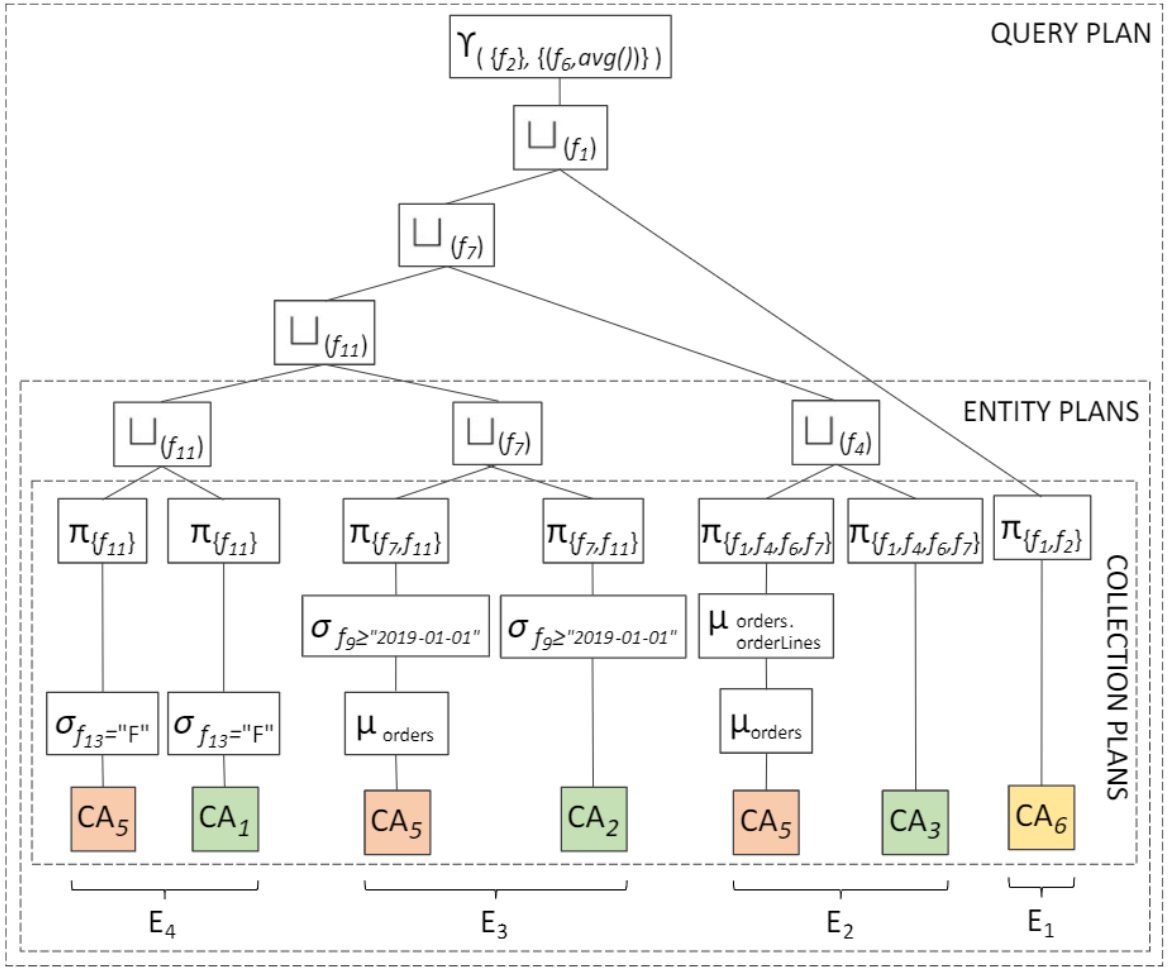
- Collections accessed more than once
- Most effort pulled to the middleware

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



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

Same query, several query plans

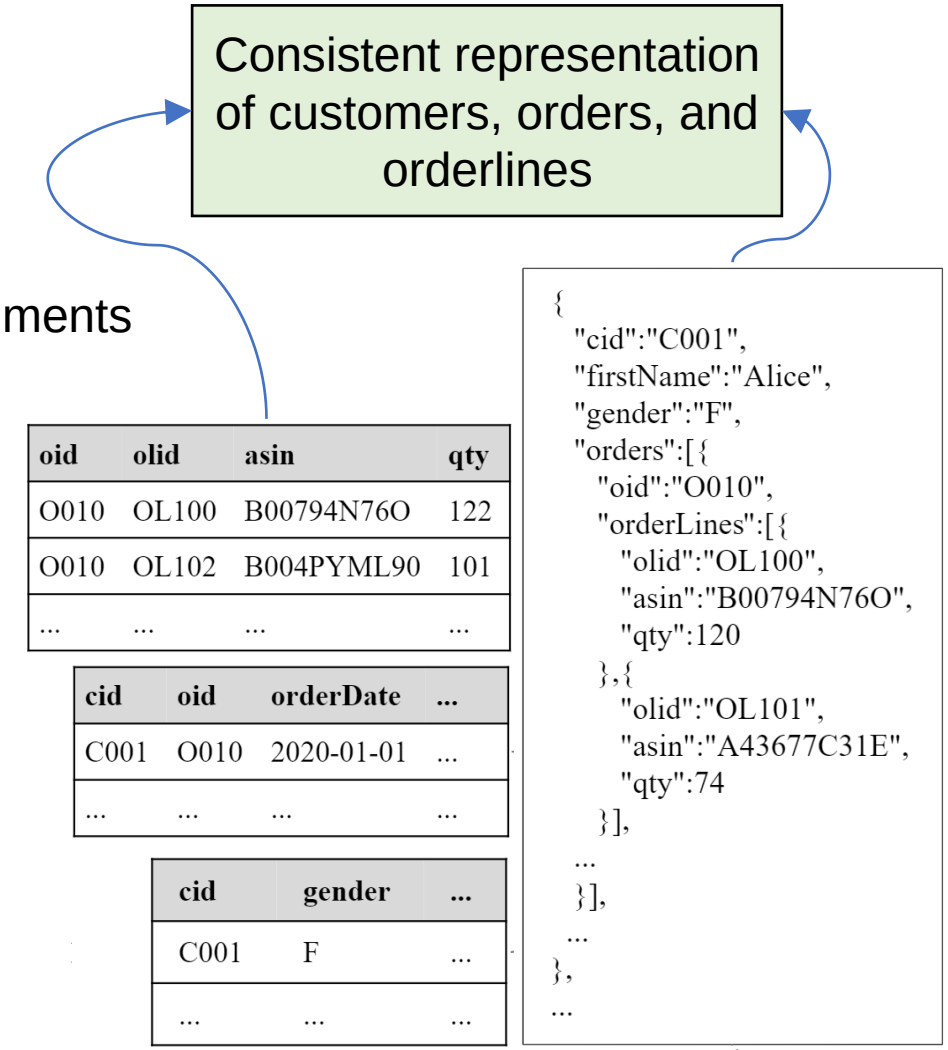
What is the most efficient solution?

- 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



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

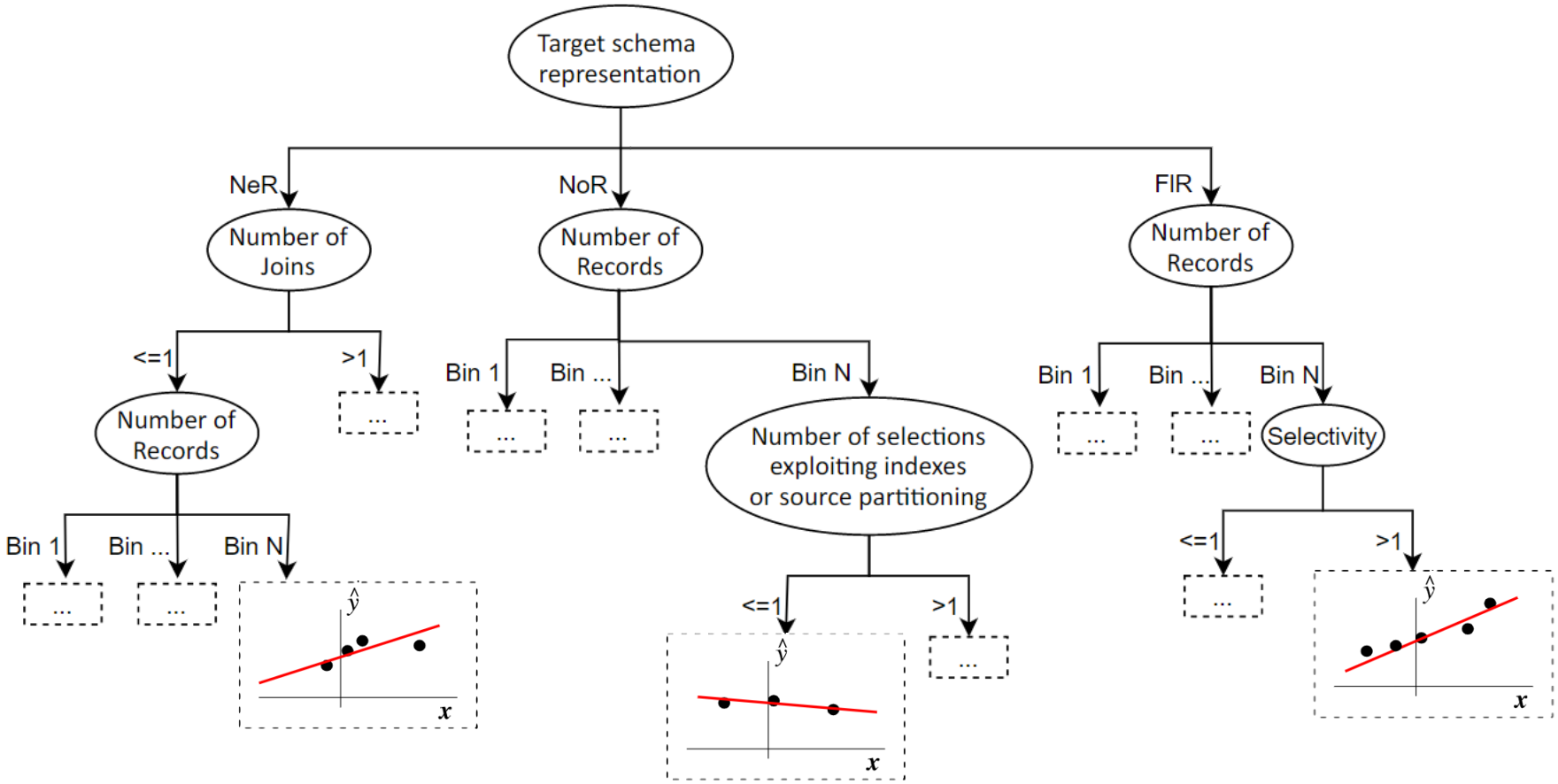
Parameter/Function	Description
$NR(C)$	Number of records in C
$NNR(C)$	If C is nested, number of nested records
$Len(C)$	Average byte length of a record in C
DPS	Size of a disk page
$PT(C)$	Number of partitions into which records in C are organized
NB	Number of memory buffers
$NP(C)$	Number of disk pages 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

Forresi, C., Francia, M., Gallinucci, E., & Golfarelli, M. (2021). Optimizing execution plans in a multistore. In Advances in Databases and Information Systems: 25th European Conference, ADBIS 2021.

Cost modelling

Black-box cost modelling example



Forresi, C., Francia, M., Gallinucci, E., & Golfarelli, M. (2022). Cost-based Optimization of Multistore Query Plans. Information Systems Frontiers, 1-27.