CPSC 436C
Cloud Computing for Data Science

Data Management Systems

Maryam R.Aliabadi

mraiyata@cs.ubc.ca

Spring 2024

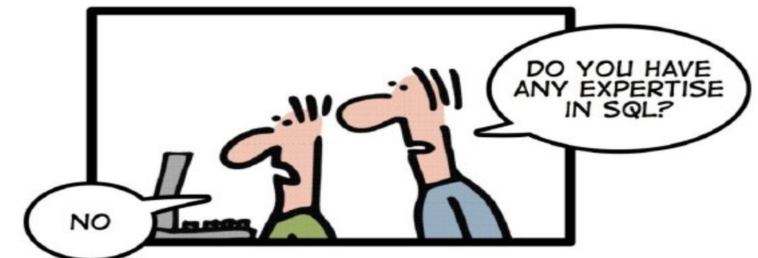
Last Class Review

- NoSQL Key-Value Store – DynamoDB
  - Consistent Hashing
  - Dynamo API
  - Version Branching
  - Membership management
  - Failure detection and handling

Dynamo

- Distributed key/value storage system
- Avoidance of unneeded complexity
- CAP : Availability and Partition tolerance
- Horizontal scalability

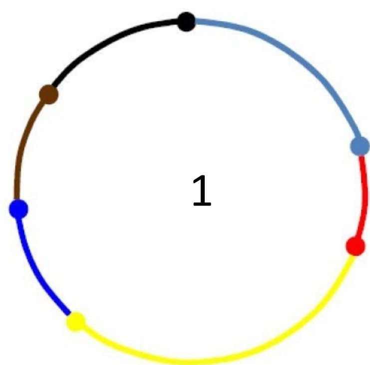
HOW TO WRITE A CV

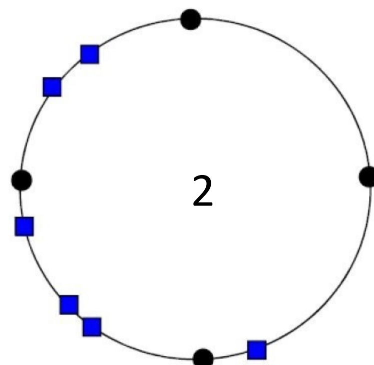


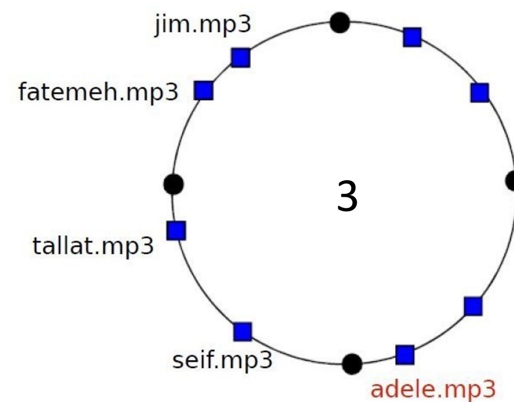
Leverage the NoSQL boom

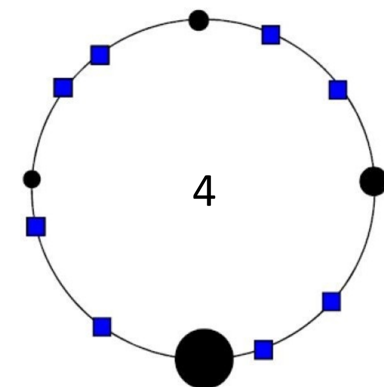
Consistent Hashing and Challenges

- Consistent hashing may lead to load imbalance.
  - 1-Node identifiers may not be balanced.
  - 2-Data identifiers may not be balanced.
  - 3-Hot spots.
  - 4-Heterogeneous nodes.



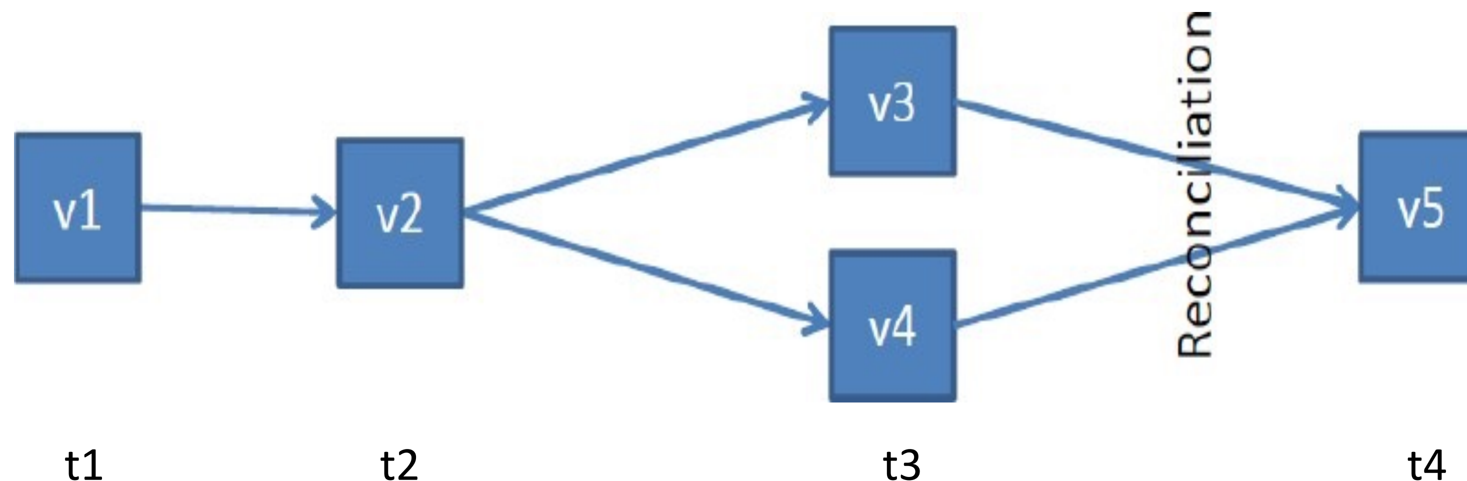






Data Versioning

- Version branching can happen due to node/network failures.
- Dynamo uses vector clocks for capturing causality, in the form of [node, counter] pairs



t1 t2 t3 t4

4-Dynamo API

- get(key)
  - Return single object or list of objects with conflicting version and context.

- put(key, context, object)
  - Store object and context under key.
  - Context encodes system metadata, e.g., version number.

Execution of Operations

- Client can send the request:
  - To the node responsible for the data (Partition coordinator)
    - save on latency, code on client
  - To a generic load balancer: extra hop
    - distribute incoming requests evenly across the Dynamo nodes
- The choice depends on the specific configuration and requirements of the Dynamo deployment.

Put Operation

- Coordinator generates new vector clock and writes the new version locally.
- Send to N nodes.
- Wait for response from W nodes.
- Using W=1
  - High availability for writes
  - Low durability

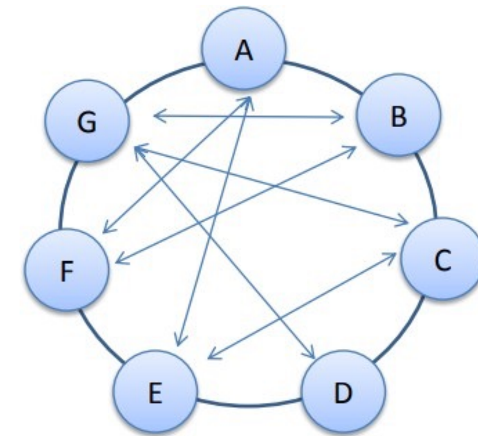
Get Operation

- Coordinator requests existing versions from N.
  - Wait for response from R nodes.

- If multiple versions, return all versions that are causally unrelated.
- Divergent versions are then reconciled.
- Reconciled version written back.
- Using R=1
  - High performance read engine.

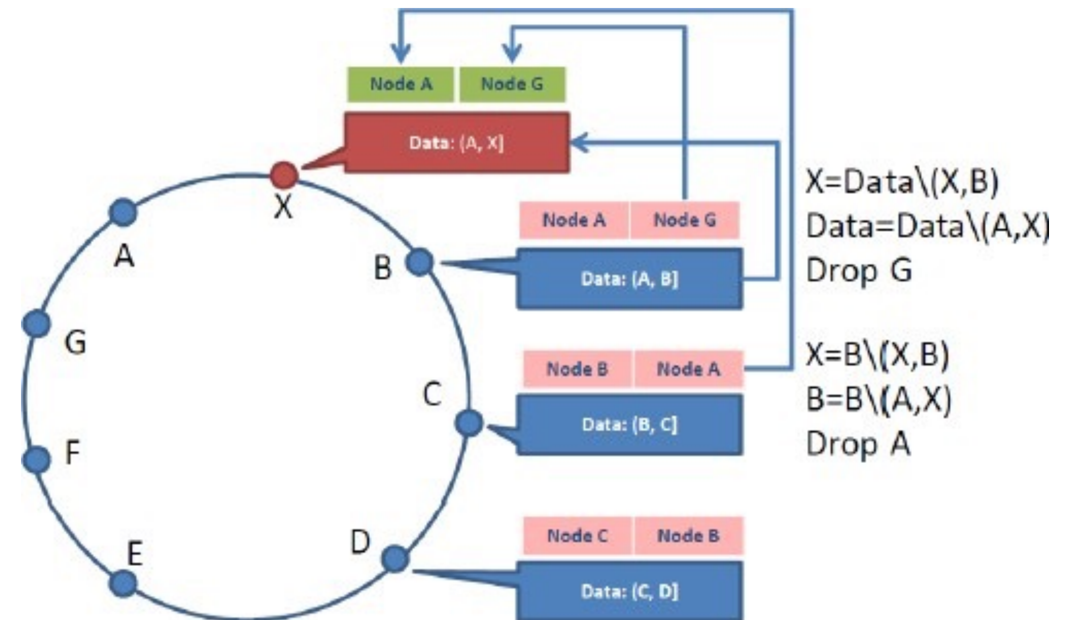
5-Membership Management

- Administrator explicitly adds and removes nodes.
- Gossiping to propagate membership changes.
  - periodically, each node contacts another randomly selected node.
- Eventually consistent view.
- O(1) hop overlay.
  - data access operations can be performed with low latency (constant number of hops).



Adding Nodes

- A new node X added to system.
  - X is assigned key ranges w.r.t. its virtual servers.
  - For each key range, it transfers the data items.



Removing Nodes

- Reallocation of keys is a reverse process of adding nodes.
  - Determining new owners
  - Assigning new key ranges to new owners
  - Data transfer to new owners
  - Synchronization

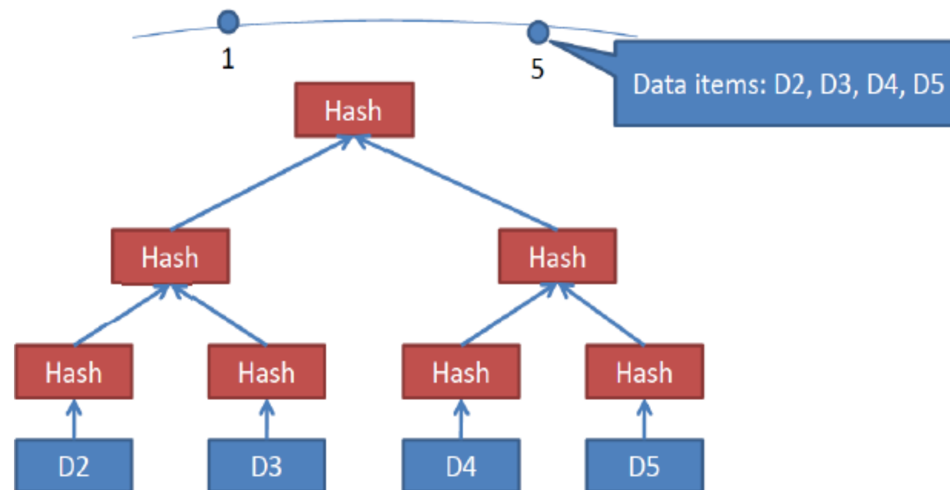
6 - Failure Detection and Handling

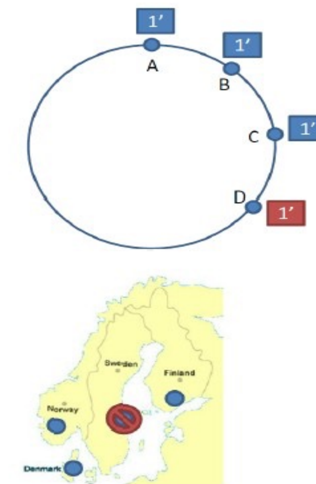
- Passive failure detection.
  - Use pings only for detection from failed to alive.

- In the absence of client requests, node A doesn’t need to know if node B is alive.

Handling Permanent Failure

- ►Anti-entropy for replica synchronization.
- ►Use Merkle trees for fast inconsistency detection and minimum transfer of data.
  - Nodes maintain Merkle tree of each key range.
  - Exchange root of Merkle tree to check if the key ranges are updated.





Today’s Topics

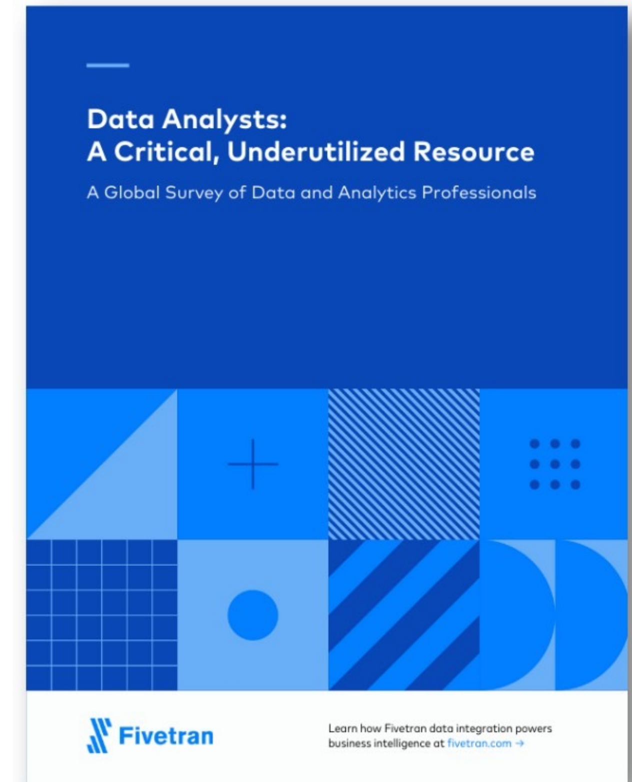
- Data Management systems
  - Data Warehouse
  - Data Lake
  - Lakehouse

Biggest Challenges With Data

- Data quality
- Staleness
- Data volume
- Scale

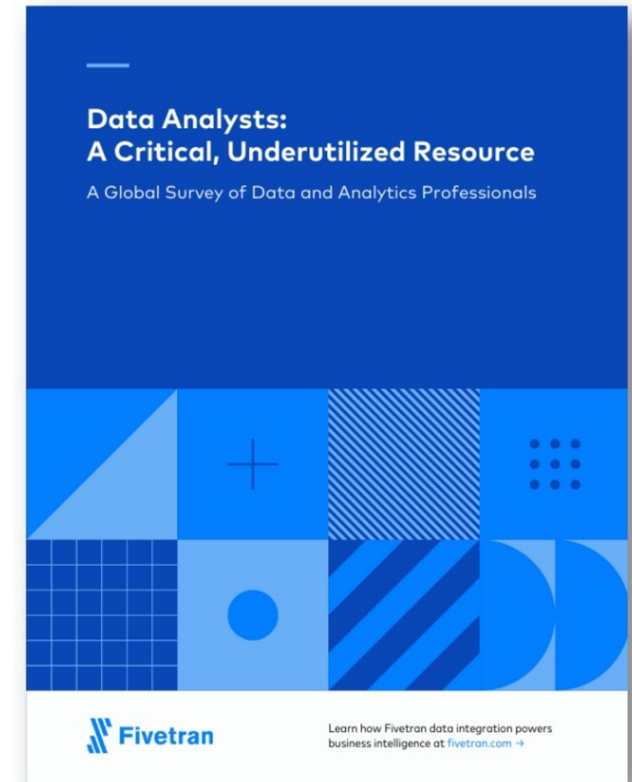
Fivetran Data Analyst Survey

- 60% reported data quality as top challenge.
- 86% of analysts had to use stale data, with 41% using data that is > 2 months old.
- 90% regularly had unreliable data sources over the last 12 months



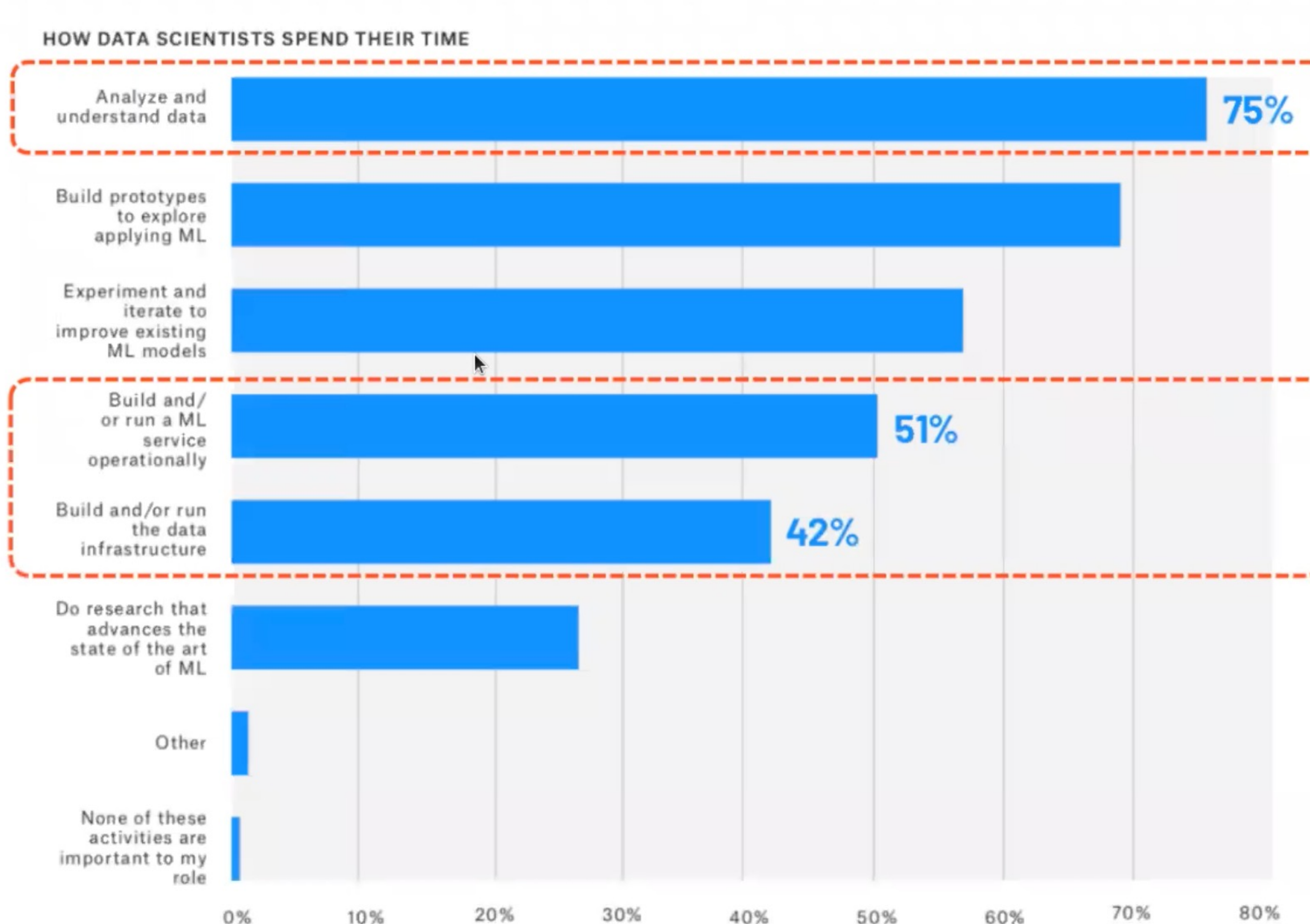
Fivetran Data Analyst Survey

- 60% reported data quality as top challenge.
- 86% of analysts had to use stale data, with 41% using data that is >2 months old.
- 90% regularly had unreliable data sources over the last 12 months



Getting high-quality, timely data is hard!

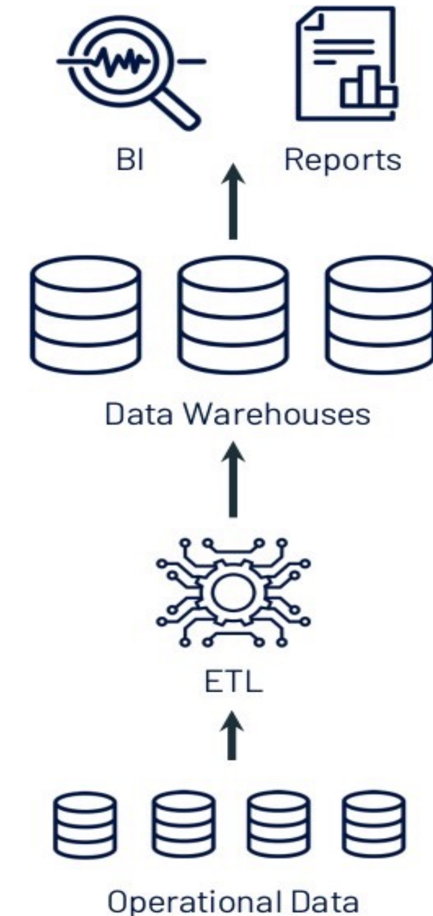
Kaggle Data Analyst Survey



The Evolution of Data Management Systems

Data Warehouses (1980s)

- ETL (Extract, Transform, Load) data directly from operational database systems.
- Purpose-built for SQL analytics and BI: schemas, indexes, caching, etc.
- Powerful management features such as ACID transactions and time travel
- Data Warehouse defines the schema before data is stored (Schema on write).

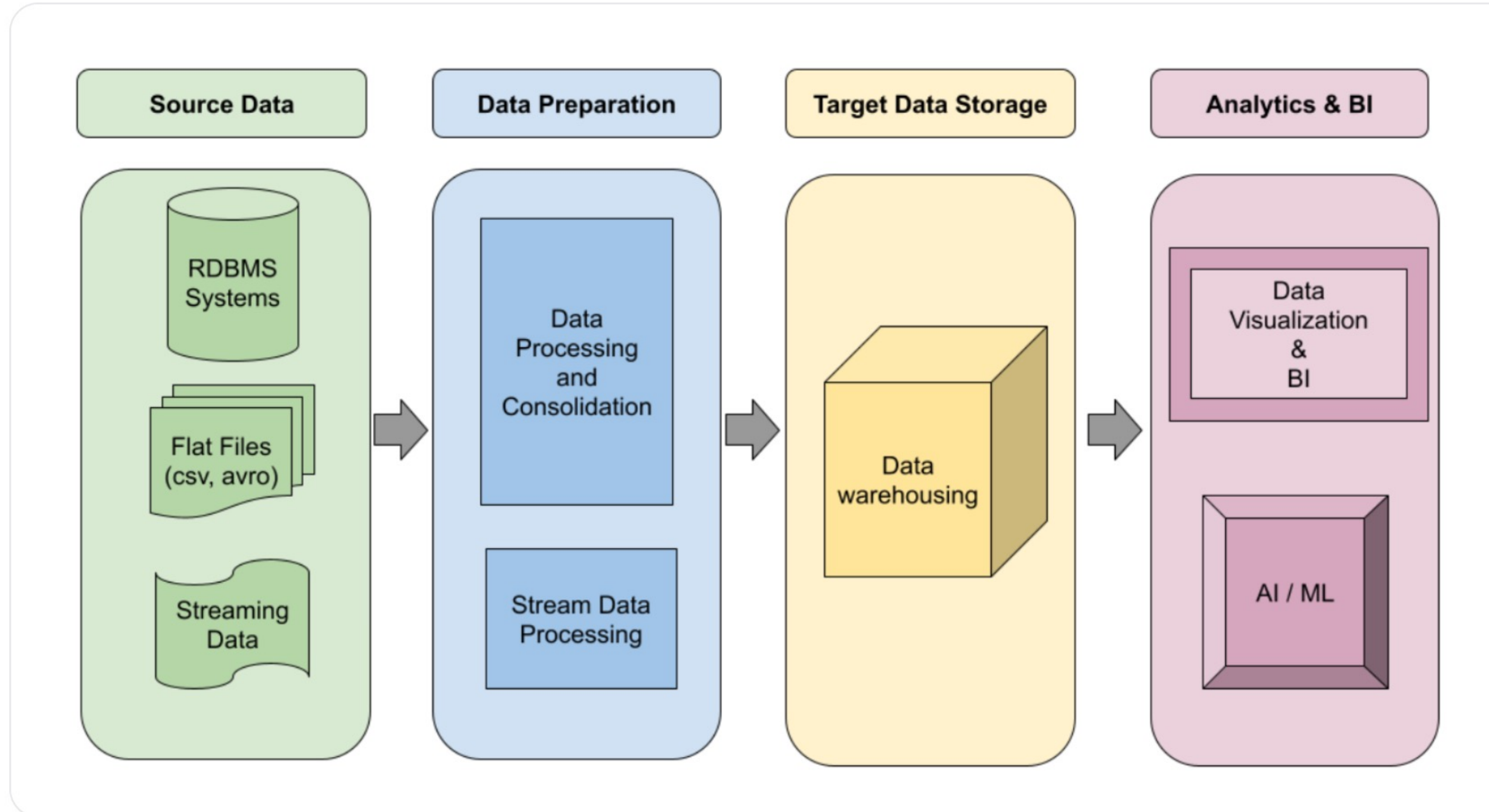


ETL (Extract – Transform – Load)

- ETL stands for:
  - Extract: bringing in data from different sources
  - Transform: filtering, cleaning, imputing, aggregation, calculations
  - Load: moving transformed data to a staging area

- ETL is the process by which legacy data warehouses ingest and transform data.

Data Warehousing Pipeline - ETL

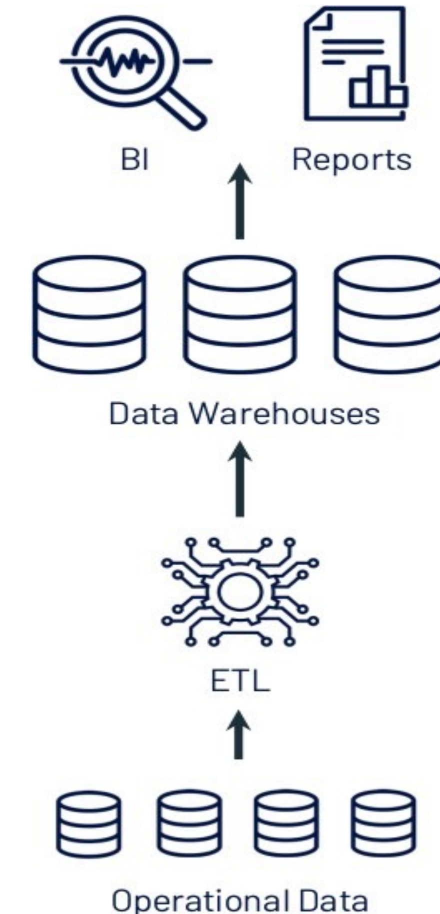


Data Warehouses - Problems (2010s)

- Could not support rapidly growing unstructured and semi-structured data: time series, logs, images, documents, etc.

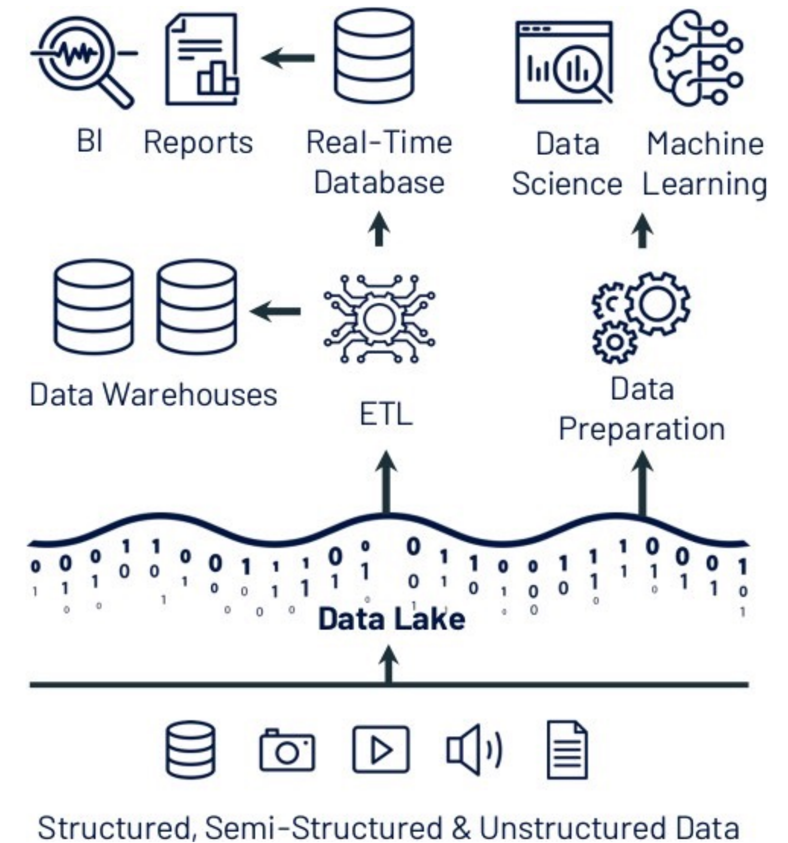
- High cost to store large datasets.

- No support for data science and ML.



Data Lakes (2010s)

- Low-cost storage to hold all raw data, e.g., Amazon S3, and HDFS.
- ETL jobs then load specific data into warehouses, possibly for further ELT.
- Directly readable in ML libraries (e.g., TensorFlow and PyTorch) due to open file format.

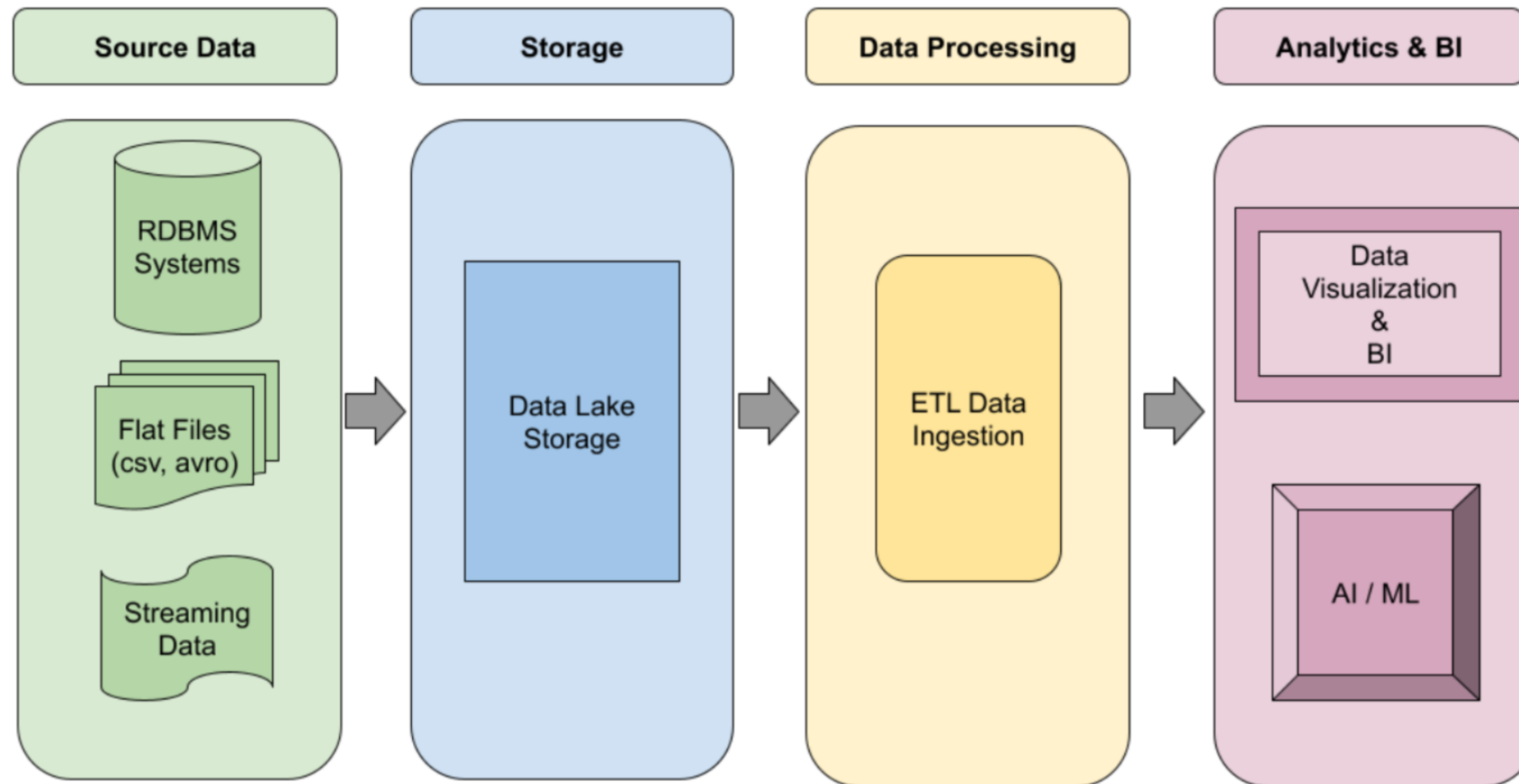


ELT (Extract – Load – Transform)

- A modern approach used in data lakes, which loads raw data from the source without transforming it, preserving the original format

  - Defines the schema after data is stored (Schema on read).
  - Removes associated compute and maintenance costs
  - Prevents information loss when converting data from its raw format to a conformed structure.
  - Transformation can then be performed selectively

Data Lake Pipeline - ELT

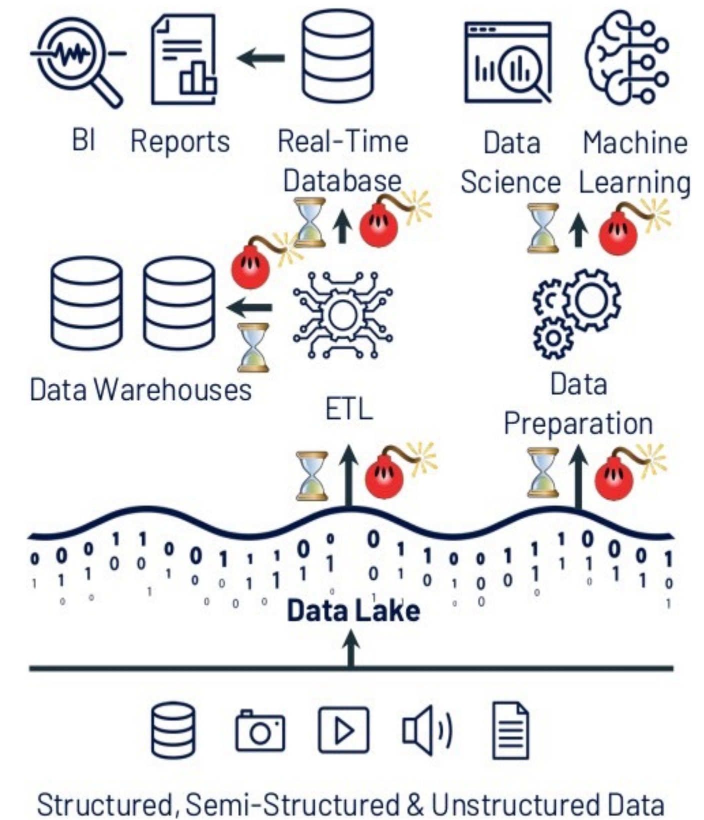


Data Lakes

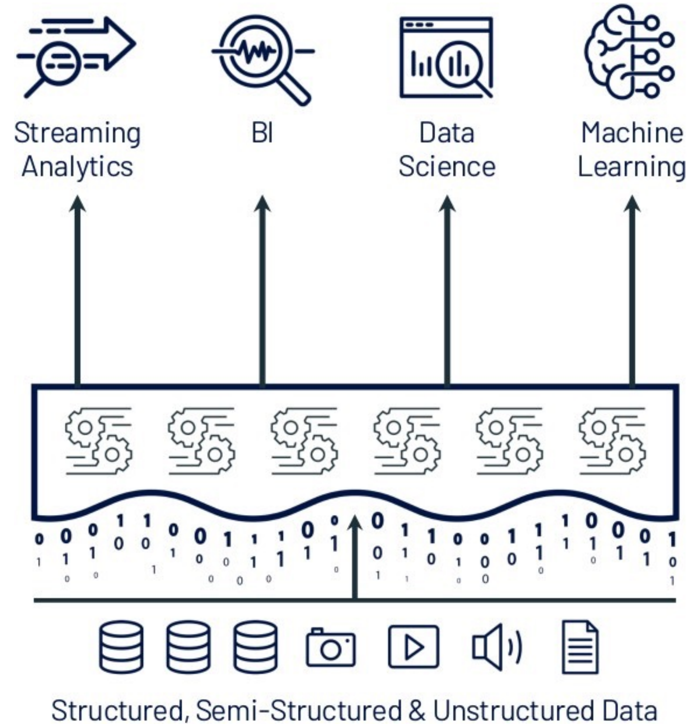
- Reduce up-front effort by ingesting data in any format, any size, without requiring a schema initially
- Make acquiring new data easy, so it can be available for data science and analysis quickly
- Store large volume of multi-structured data in its native format
- Storage for additional types of data which were historically difficult to obtain or score
- Reduce the long-term ownership cost of data management & storage

Data Lakes - Problems (Todays)

- Cheap to store all the data, but system architecture is much more complex!
- Data reliability suffers:
  - Multiple storage systems with different semantics, SQL dialects, etc.
  - Extra ETL steps that can go wrong.
- Timeliness suffers and high cost:
  - Extra ETL steps before data is available in data warehouses.
  - Continuous ETL, duplicated storage



Lakehouse



30 / 55

Single platform for every use case

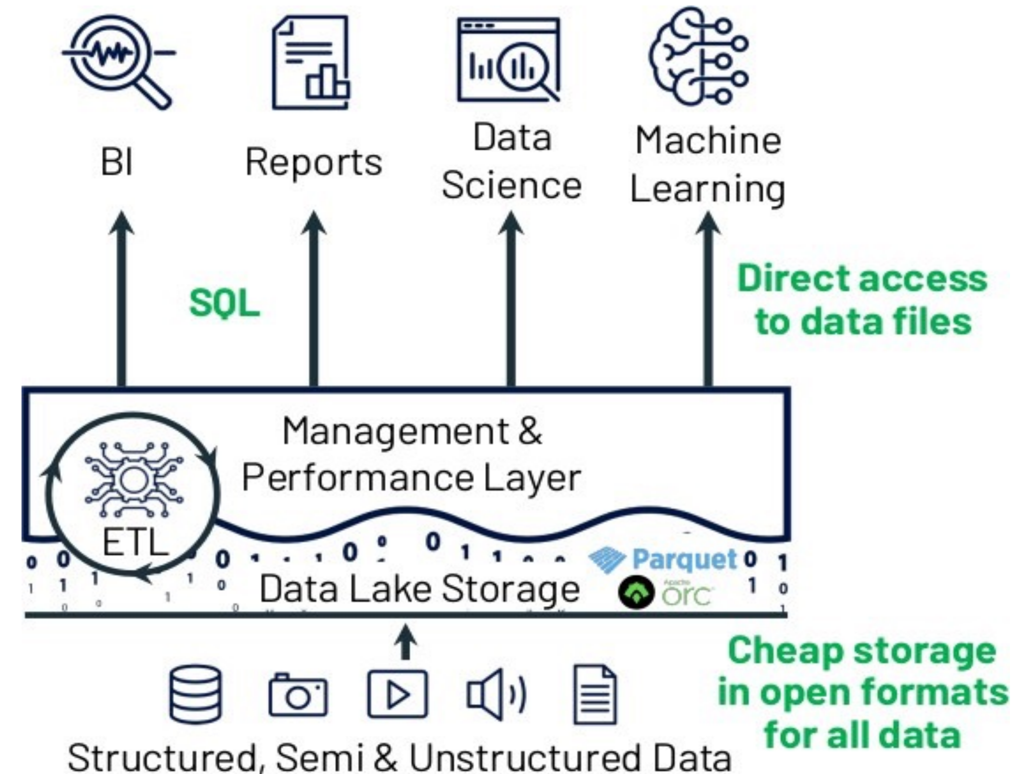
Management features (transactions, versioning, etc.)

Data lake storage for all data

- Lakehouse combines the benefits of Data Warehouses and Data Lakes while simplifying enterprise data architectures.

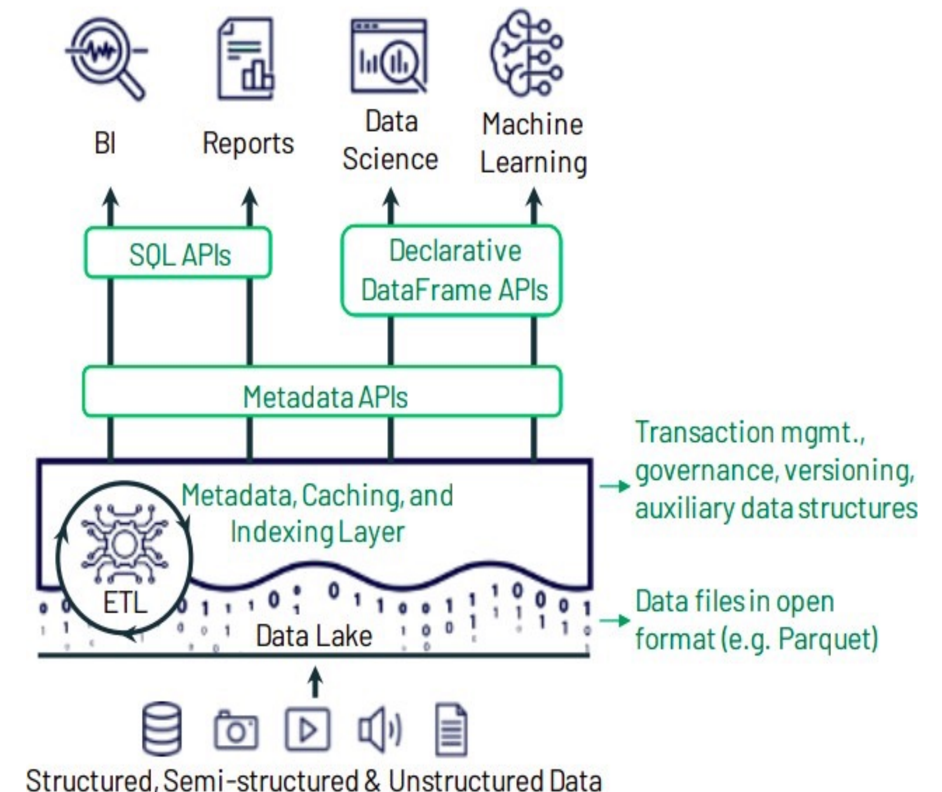
Lakehouse System

- Implements Data Warehouse management and performance features on top of directly-accessible data in open formats.
- Can we get state-of-the-art performance and governance features with this design?



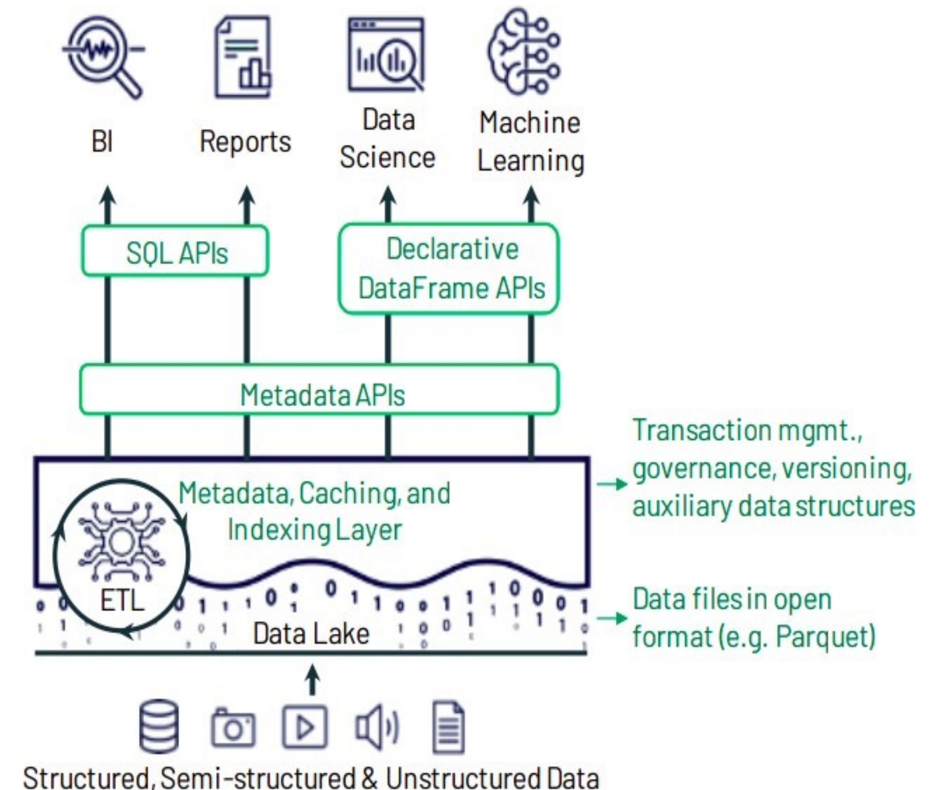
Key Technologies Enabling Lakehouse

- Metadata layers for Data Lakes
- New query engine designs
- Declarative access for data science and Machine Learning



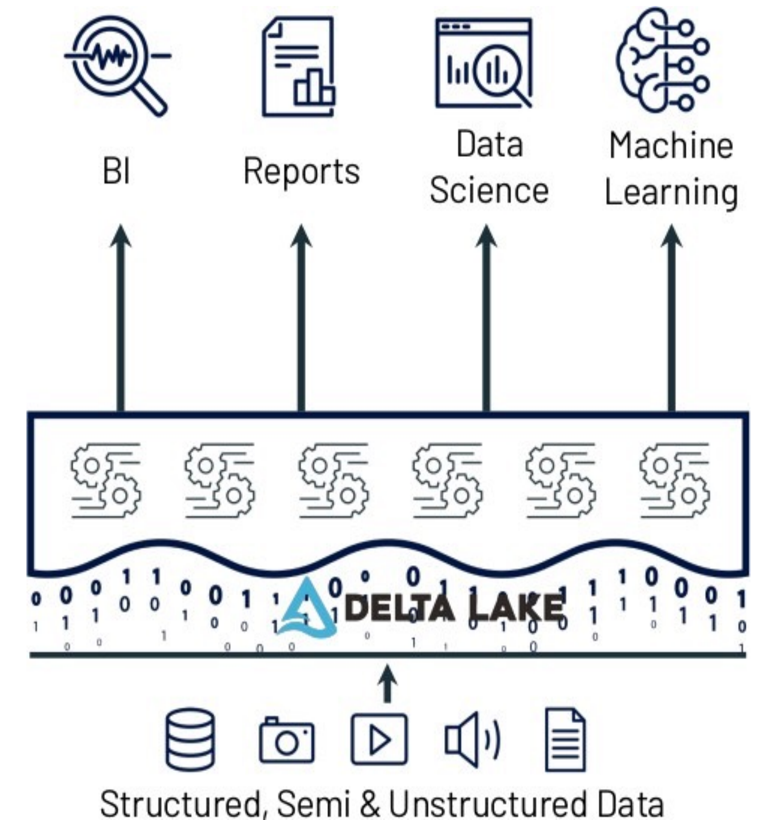
Metadata Layers for Data Lakes

- Add transactions, versioning, and more …
- Track which files are part of a table version to offer rich management features like transactions.
  - Clients can then access the underlying files at high speed.
  - Optimistic concurrency



Metadata Layers for Data Lakes

- Implemented in multiple systems, such as Delta Lake.
  - Table view for unstructured data
  - ACID properties

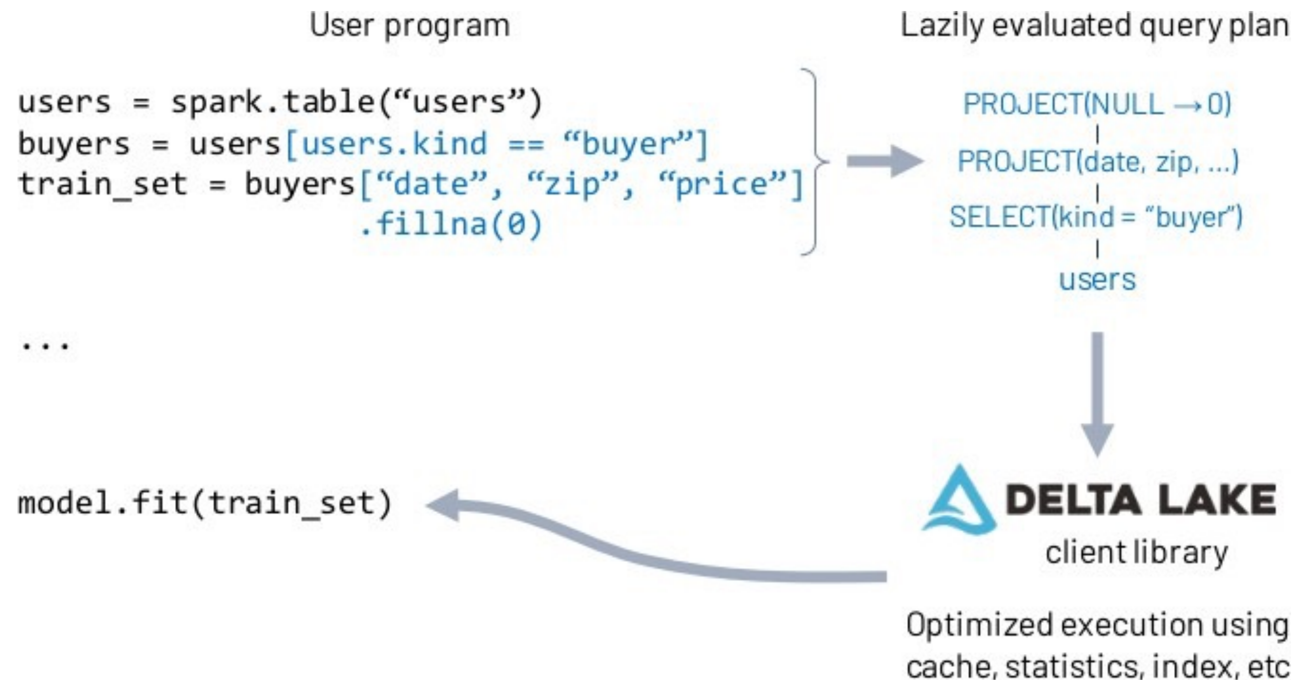


New Query Engine Designs

- Great SQL performance on Data Lake storage systems and file formats.
- Directly-accessible file storage optimizations can enable high SQL performance:
  - Caching hot data in RAM/SSD
  - Data layout within files to cluster co-accessed data (e.g., sorting or multi-dimensional clustering)
  - Auxiliary data structures like statistics and indexes

Declarative Access for Data Science and ML

- New declarative interfaces for I/O enable further optimization.
- Example: Spark DataFrame API compiles to relational algebra.



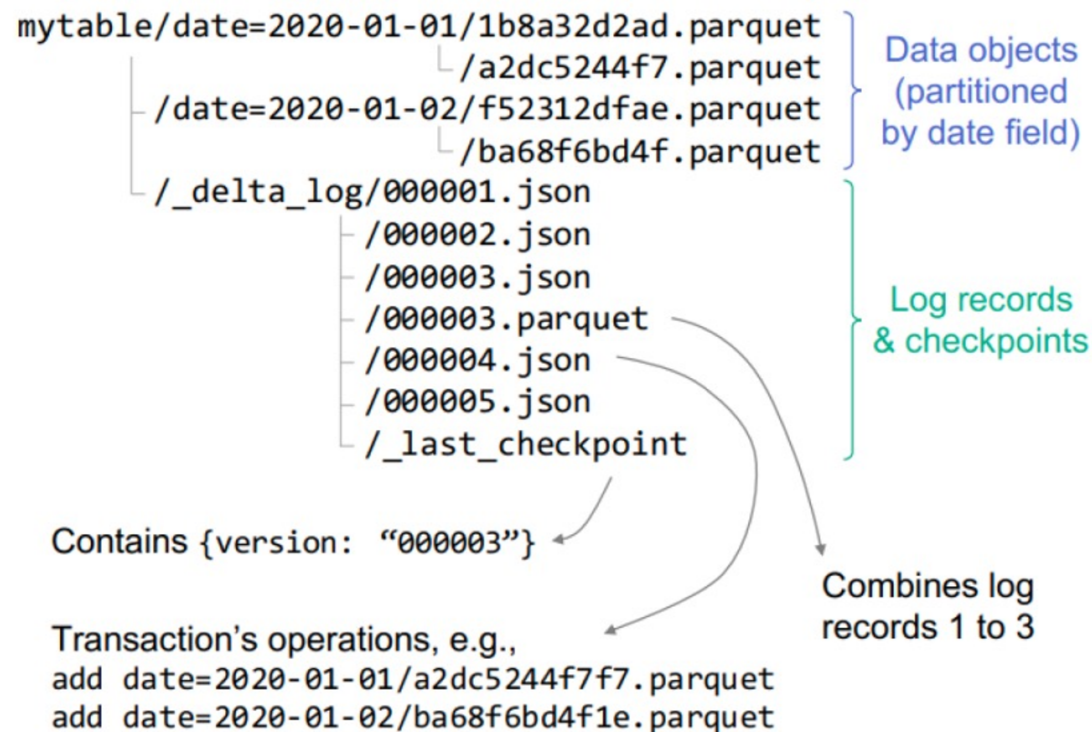
Delta Lake

Delta Lake

- Delta Lake is an open source storage layer that brings reliability to Data Lakes.
- Provides ACID transactions.
- Provides scalable metadata handling.
- Provides time travel and versioning.
- Unifies streaming and batch data processing.

Delta Lake

- Delta Lake Table is a directory (e.g., mytable) that holds data objects and a log of transaction operations.



DeltaLog

- DeltaLog is a transaction log that tracks all changes that users make to the table.

- Delta Lake uses the DeltaLog for many features including ACID transactions, scalable metadata handling, time travel, etc.

DeltaLog

- When a user creates a Delta Lake Table, its DeltaLog is automatically created in the delta log subdirectory.
- Any changes to that table are then recorded as ordered, atomic commits in the DeltaLog.
- Each commit is written out as a JSON file, starting with 000000.json.
- Additional changes to the table generate subsequent JSON files in ascending numerical order, e.g., 000001.json, 000002.json, and so on.

Time Travel

- Every table is the result of the sum of all of the commits recorded in the Delta Lake DeltaLog.
- The DeltaLog provides a step-by-step instruction guide, detailing exactly how to get from the table's original state to its current state.
- Thus, we can recreate the state of a table at any point in time.
- Starting with an original table, and processing only commits made prior to that point.
- This ability is known as time travel or data versioning.

Data Lineage

- The Delta Lake DeltaLog offers users a verifiable data lineage.
- It is is useful for governance, audit and compliance purposes.
- It can also be used to trace the origin of an inadvertent change or a bug in a pipeline back to the exact action that caused it.

Schema Enforcement and Evolution

- Delta Lake supports schema on read and write operations.
- Schema enforcement: prevents users from accidentally polluting their tables with mistakes or garbage data.
- Schema evolution: enables automatic addition of columns when desired.

Schema Enforcement

- With Delta Lake, the table's schema is saved in JSON format inside the DeltaLog.

```
schemaString: {"type":"struct","fields":[
  {"name":"loan_id","type":"long","nullable":false,"metadata":{}},
  {"name":"funded_amnt","type":"integer","nullable":true,"metadata":{}},
  {"name":"paid_amnt","type":"double","nullable":true,"metadata":{}},
  {"name":"addr_state","type":"string","nullable":true,"metadata":{}}
]}
```

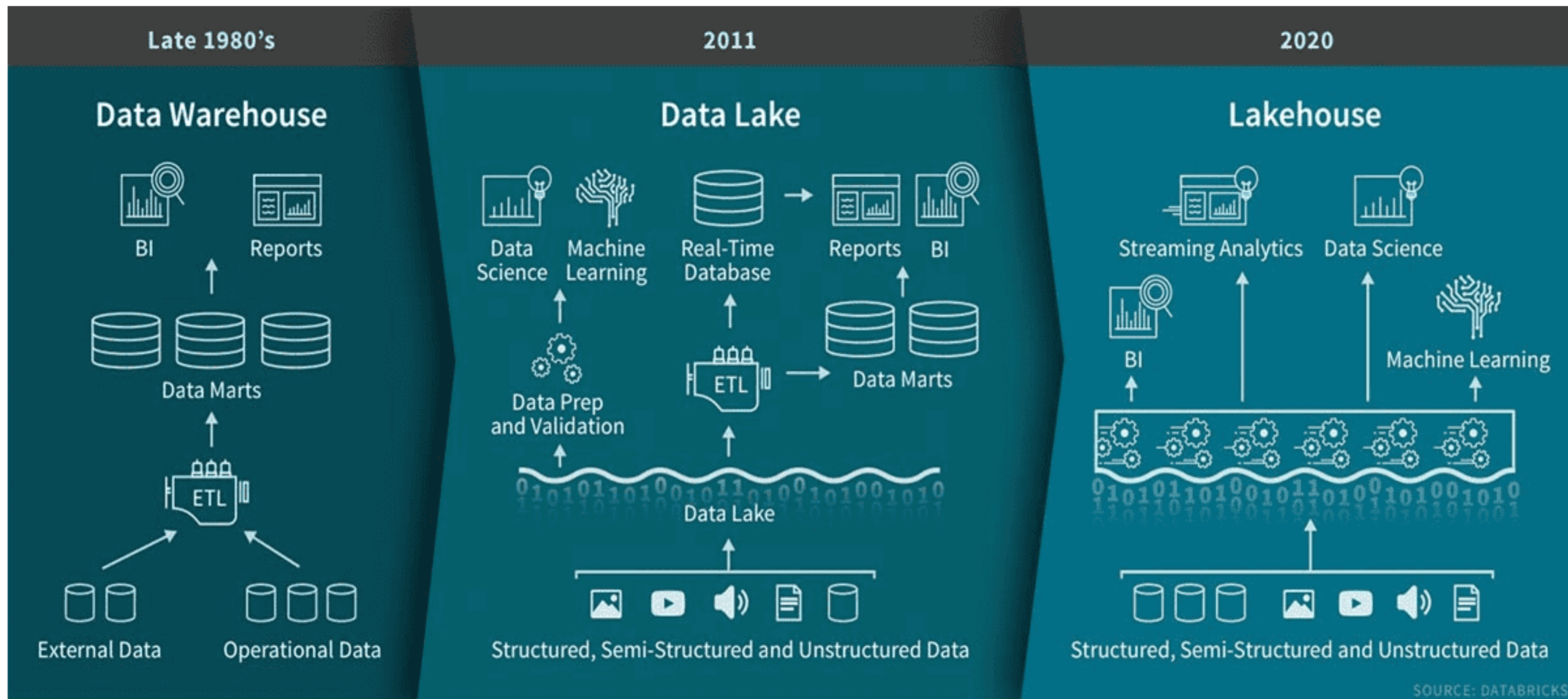
- Schema enforcement (a.k.a schema validation) occurs on write.
- If the schema is not compatible, Delta Lake cancels the transaction, i.e., no data is written.
- Delta Lake raises an exception to let the user know about the mismatch.

Schema Evolution

- Schema evolution allows users to change a table's current schema to accommodate data that is changing over time.

- Most commonly used operations for append and overwrite.

Recap



Next Topic

Data Processing – MapReduce