Aalto University
School of Science

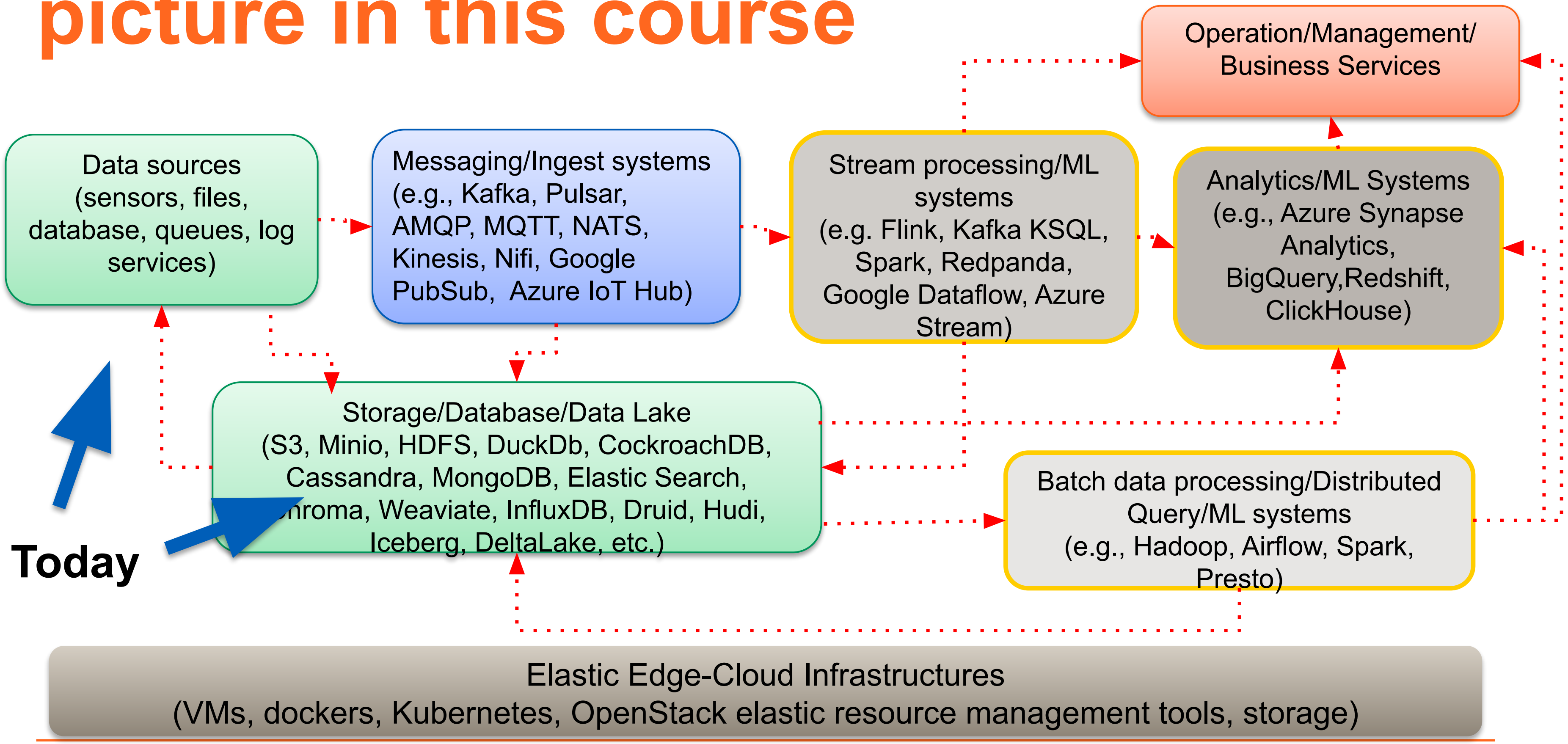
Data Services in Big Data Platforms

Hong-Linh Truong
Department of Computer Science
linh.truong@aalto.fi, https://rdsea.github.io

Learning objectives

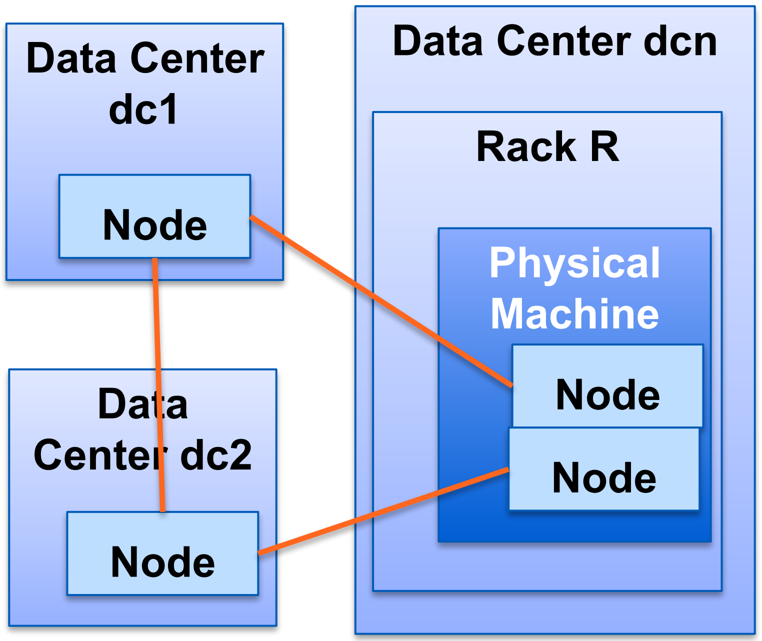
- Understand consistency, availability and partition tolerance issues in design and programming
- Study common data models and data management
- Understand composability: the need of polyglot persistence
- Understand the role of metadata management

Big data at large-scale: the big picture in this course


Operation/Management/ Business Services
Data sources (sensors, files, database, queues, log services)
Messaging/Ingest systems (e.g., Kafka, Pulsar, AMQP, MQTT, NATS, Kinesis, Nifi, Google PubSub, Azure IoT Hub)
Stream processing/ML systems (e.g. Flink, Kafka KSQL, Spark, Redpanda, Google Dataflow, Azure Stream)
Analytics/ML Systems (e.g., Azure Synapse Analytics, BigQuery,Redshift, ClickHouse)
Storage/Database/Data Lake (S3, Minio, HDFS, DuckDb, CockroachDB, Cassandra, MongoDB, Elastic Search, Chroma, Weaviate, InfluxDB, Druid, Hudi, Iceberg, DeltaLake, etc.)
Batch data processing/Distributed Query/ML systems (e.g., Hadoop, Airflow, Spark, Presto)
Today
Elastic Edge-Cloud Infrastructures (VMs, dockers, Kubernetes, OpenStack elastic resource management tools, storage)

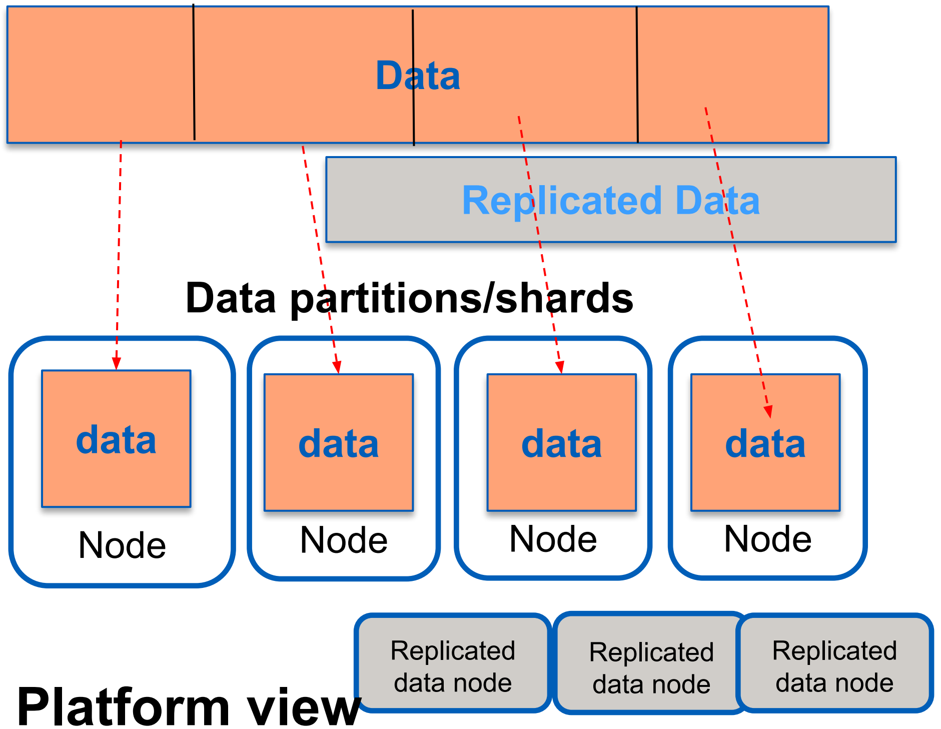
Consistency, Availability and Partition Tolerance

Big data is not stored in a single machine & analyzed using a single machine



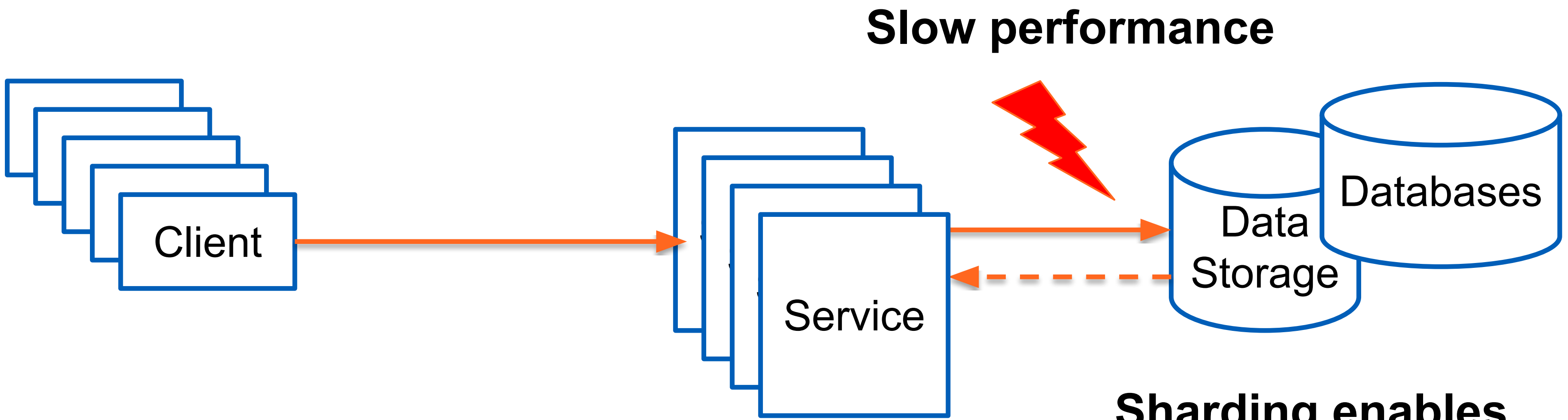
Cluster of nodes (virtual/physical machines) in multicloud, hybrid cloud and supercomputer

View from analytics application



Platform view

Performance problems between service serving request and data store

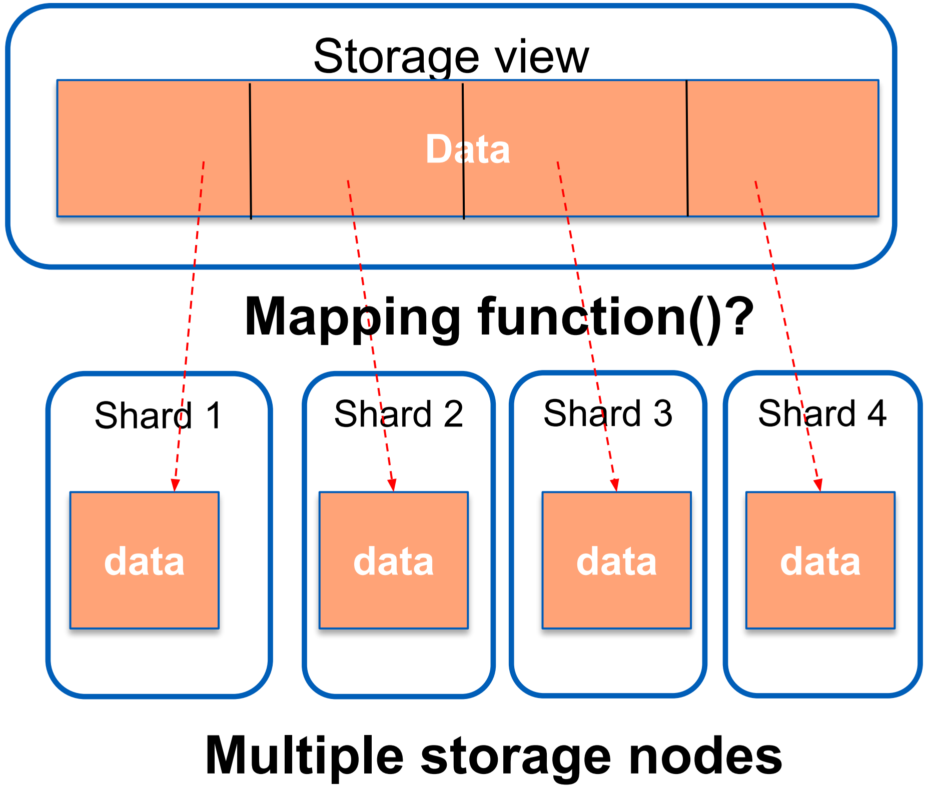


Sharding enables parallel access (data parallelism)

- **Big data grows ⇒ data explosion**
- **Concurrent contention, slow read, and slow query**

Principles

- **Partitioning data into different partitions/shards**
- **Making shards in different nodes ⇒ shared nothing, horizontal scaling!**
- **Strong links to the physical storage in disks**



Sharding Strategies

Key principles

- Determine partitioning attributes associated with data
- Each shard (where the data is stored) has a shard key mapped to partition attributes

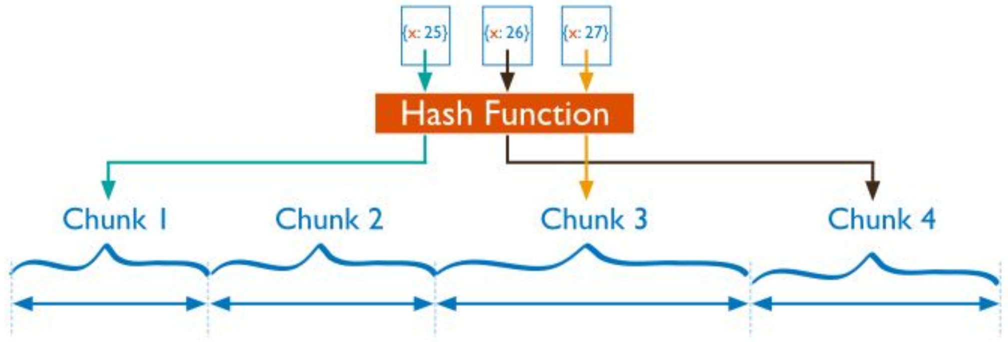
Different, common strategies

- Directory/Lookup: uses a lookup table to query partitioning attributes to find a shard
- Range: partitioning attributes are arranged into a range, each shard is responsible for a subrange
- Hash: use the hash of partitioning keys to determine the shard

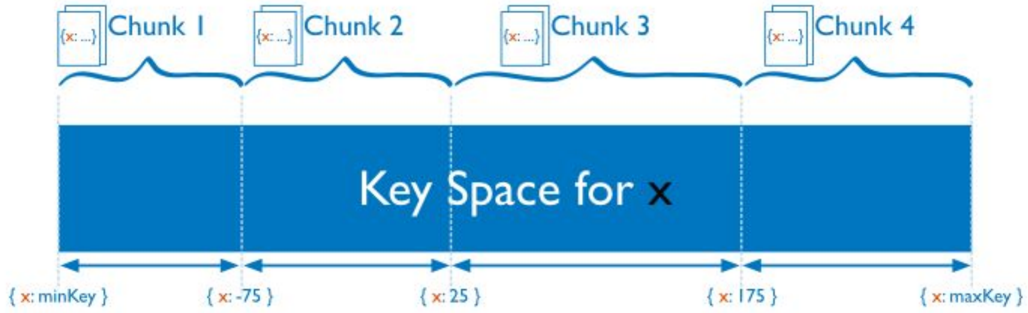
Sharding patterns/strategies reading: https://msdn.microsoft.com/en-us/library/dn589797.aspx

Example of strategies in MongoDB

Hash



Range



Figures source: https://docs.mongodb.com/manual/sharding/

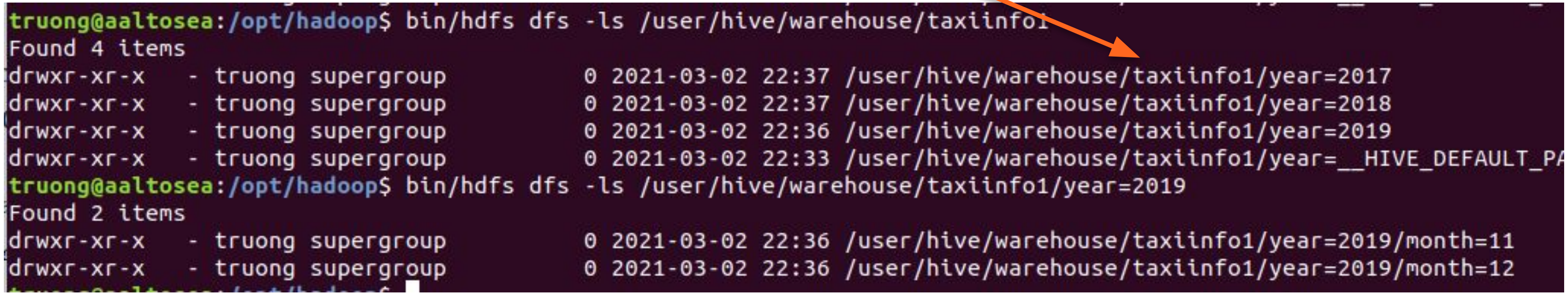
Example of partitions in Apache Hive

```
CREATE TABLE taxiinfo1 ( ….)
 PARTITIONED BY  (year int, month int)
…;
```

Indicate partition info

Define partition names

```
LOAD DATA LOCAL INPATH …… INTO TABLE taxiinfo1
PARTITION (year=2019, month=11);
```

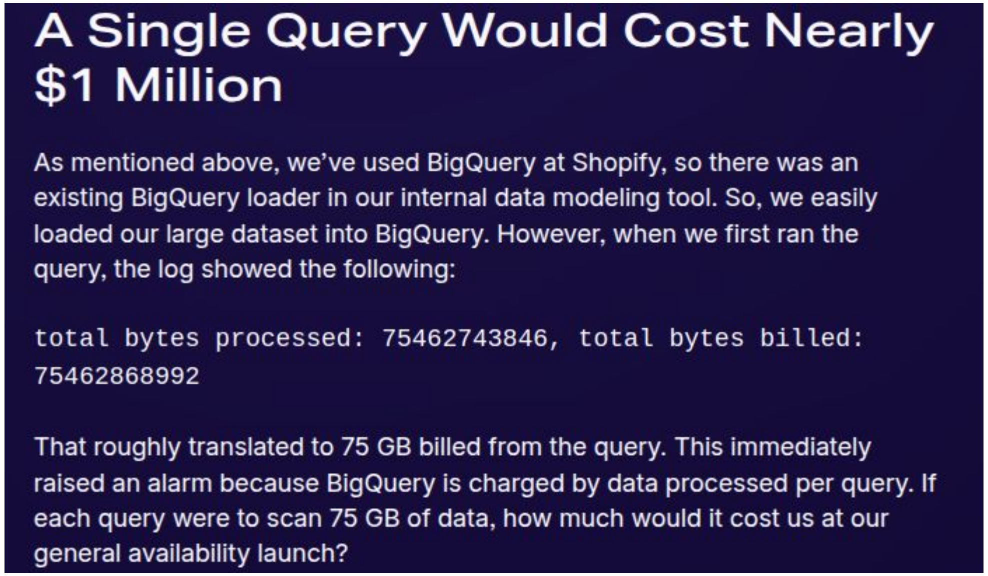


```
truong@aaltosea:/opt/hadoop$ bin/hdfs dfs -ls /user/hive/warehouse/taxiinfo1
Found 4 items
drwxr-xr-x   - truong supergroup          0 2021-03-02 22:37 /user/hive/warehouse/taxiinfo1/year=2017
drwxr-xr-x   - truong supergroup          0 2021-03-02 22:37 /user/hive/warehouse/taxiinfo1/year=2018
drwxr-xr-x   - truong supergroup          0 2021-03-02 22:36 /user/hive/warehouse/taxiinfo1/year=2019
drwxr-xr-x   - truong supergroup          0 2021-03-02 22:33 /user/hive/warehouse/taxiinfo1/year=__HIVE_DEFAULT_PA
truong@aaltosea:/opt/hadoop$ bin/hdfs dfs -ls /user/hive/warehouse/taxiinfo1/year=2019
Found 2 items
drwxr-xr-x   - truong supergroup          0 2021-03-02 22:36 /user/hive/warehouse/taxiinfo1/year=2019/month=11
drwxr-xr-x   - truong supergroup          0 2021-03-02 22:36 /user/hive/warehouse/taxiinfo1/year=2019/month=12
```

Example: partitioning and clustering benefits

Source and example from:

https://shopify.engineering/reducing-bigquery-costs



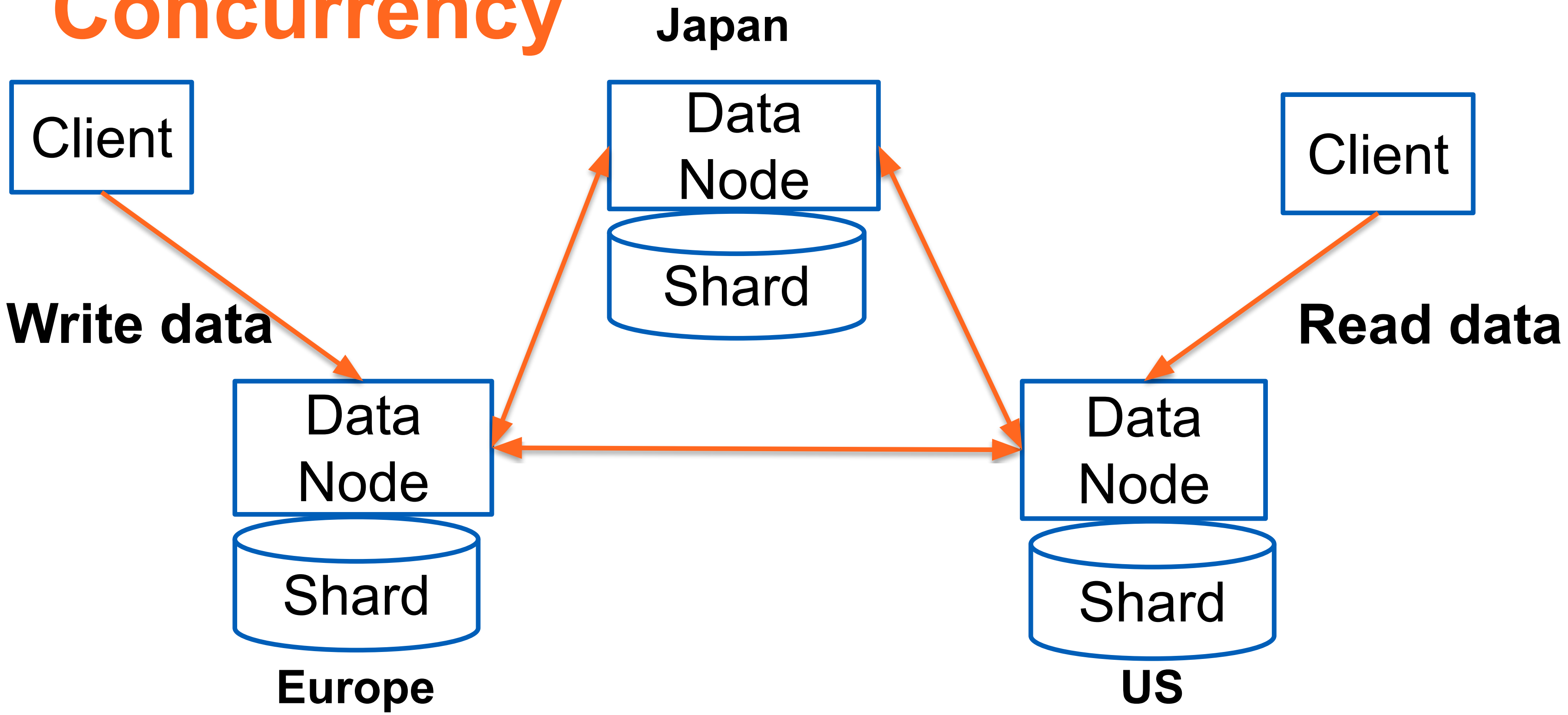
A Single Query Would Cost Nearly $1 Million

As mentioned above, we've used BigQuery at Shopify, so there was an existing BigQuery loader in our internal data modeling tool. So, we easily loaded our large dataset into BigQuery. However, when we first ran the query, the log showed the following:

```
total bytes processed: 75462743846, total bytes billed:
75462868992
```

That roughly translated to 75 GB billed from the query. This immediately raised an alarm because BigQuery is charged by data processed per query. If each query were to scan 75 GB of data, how much would it cost us at our general availability launch?

Distribution, Replication & Concurrency



Problems due to data replication/sharding and distributed data nodes

- **Can every client see the same data when accessing any node in the platform?**
- **Can any request always receive a response?**
- **Can the platform serve clients under network failures?**

Well-known ACID properties for transactional systems

- **Atomicity: with a transaction**
  - either all statements succeed or nothing
- **Consistency:**
  - transactions must ensure consistent states
- **Isolation:**
  - no interferences among concurrent transactions
- **Durability:**
  - data persisted even in the system failure

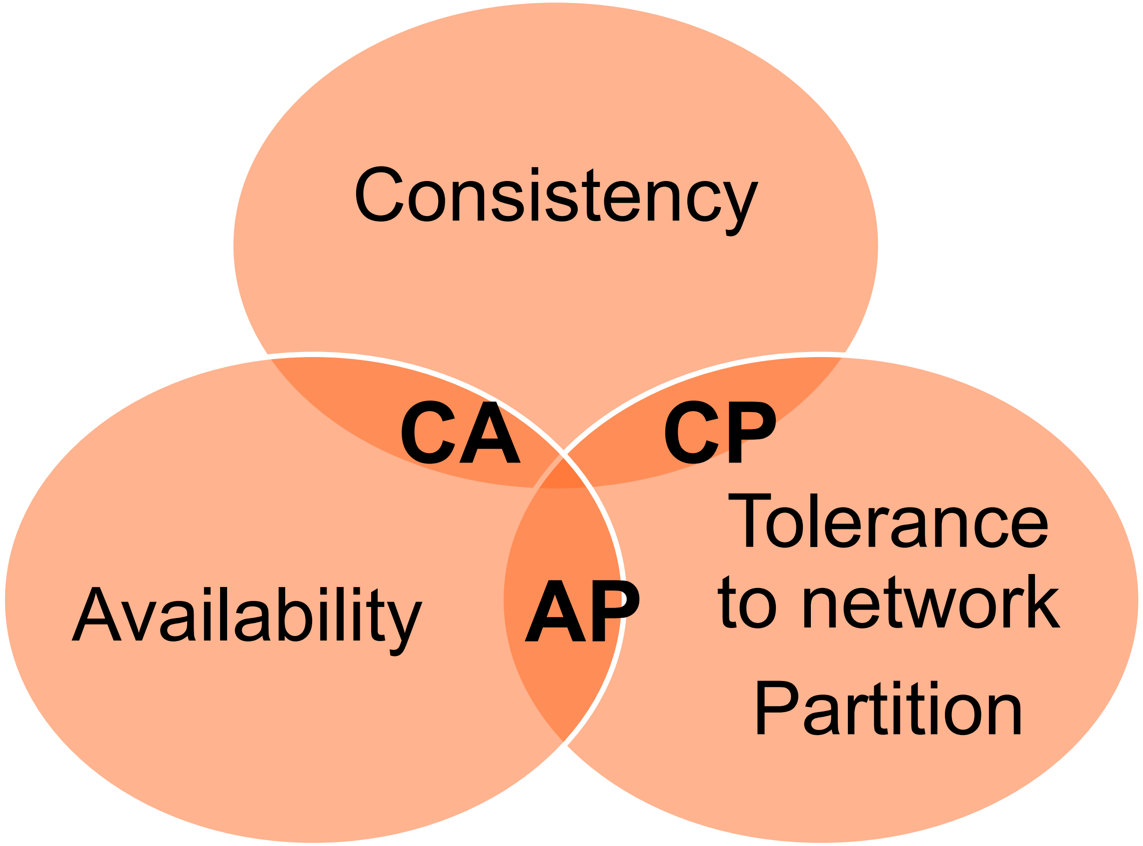
We must carefully study how such properties are supported in big data storage/databases

Issues in managing big data nodes

- **Tolerance to Network Partition**
  - if any node fails, the system is still working ⇒ a very strong constraint in our big data system design
- **High Consistency**
  - every read from a client must get the most up-to-date result
  - if the network fails, the newest write might not be updated to all nodes
- **High Availability**
  - every request must get a response (and with the most recent write)

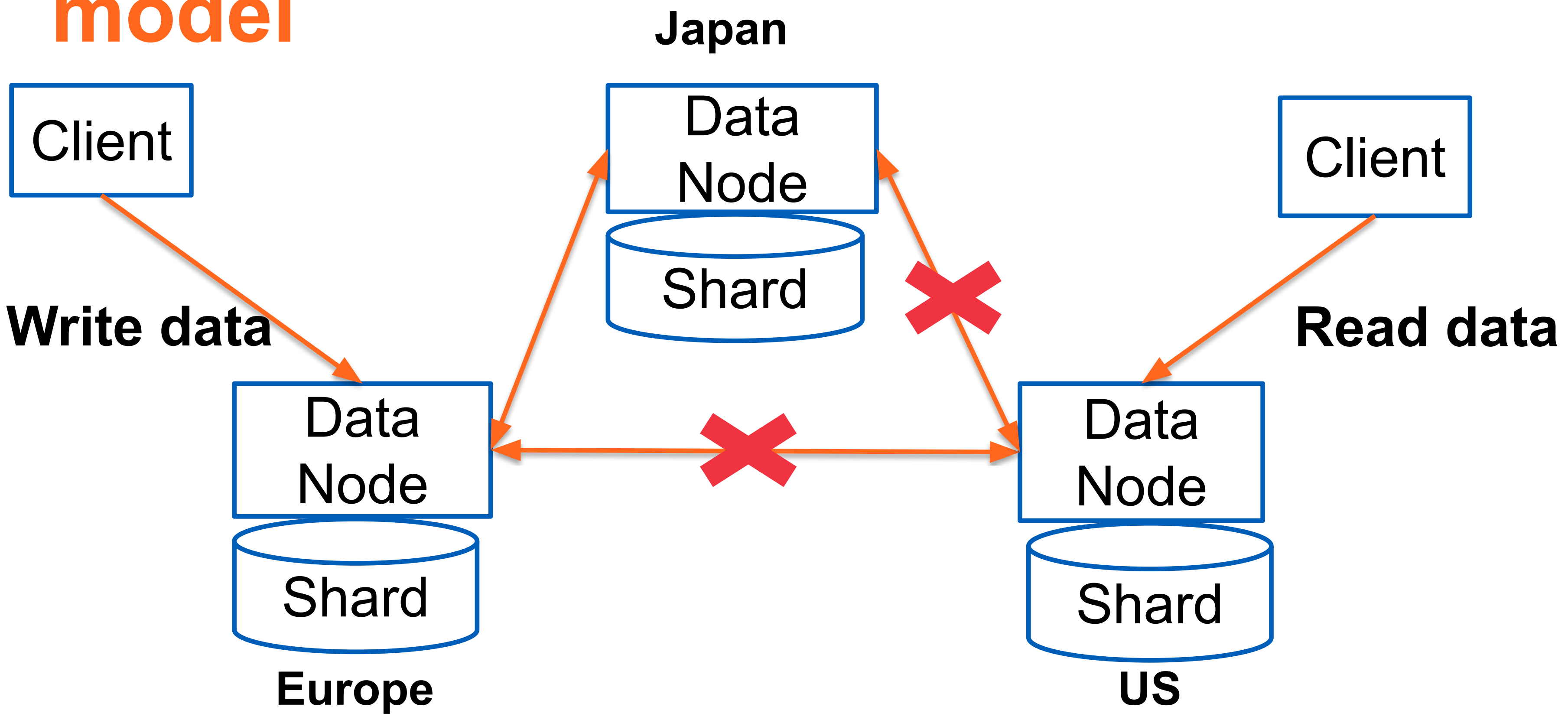
CAP Theorem

CAP theorem "you can only have 2 of out of three highly C,A,P"



CAP Theorem: E. Brewer, "CAP twelve years later: How the "rules" have changed," in Computer, vol. 45, no. 2, pp. 23-29, Feb. 2012, doi: 10.1109/MC.2012.37.

Think about CAP with this simple model



Programming consistency levels

- **Partition tolerance and availability are important for many big data applications**
  - allow different consistency levels to be configured and programmed
- **Data consistency strongly affects data accuracy and performance**
  - very much depending on technologies/specific systems and designs

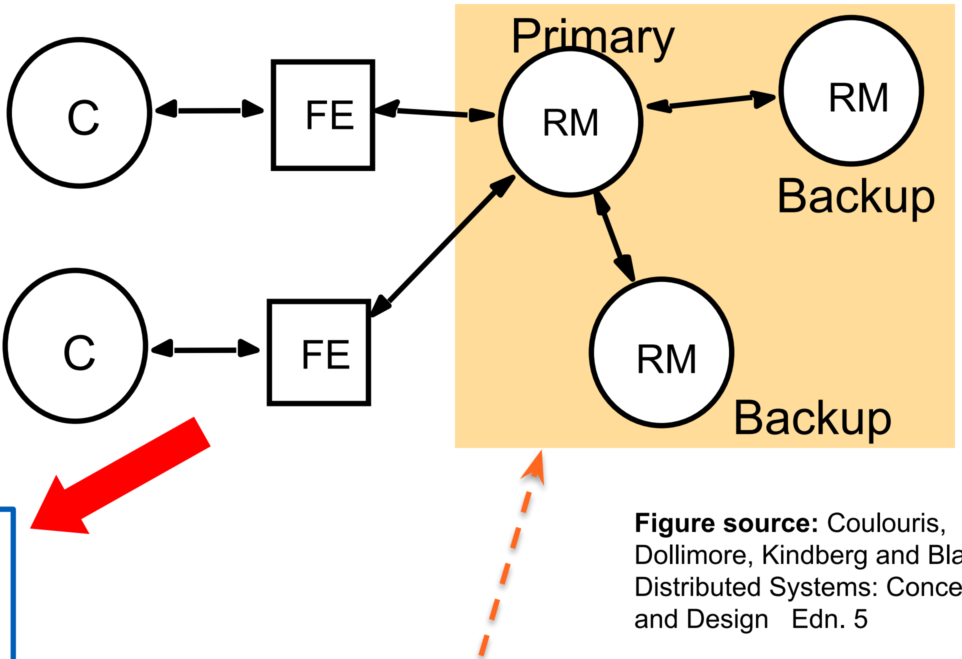
BASE (Basically Available, Soft state, Eventual consistency)

- **Focus on balance between high availability and consistency**
- **Key ideas**
  - given a data item, if there is no new update on it, eventually the system will update the data item in different places ⇒ consistent
  - allow read and write operations as much as possible, without guaranteeing consistency

Single-leader replication architecture

Passive (Primary backup) model:

- FE (Front-end) can interface to a Replication Manager (RM) to serve requests from clients.
- E.g., in MongoDB



For causal consistency

Figure source: Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5

Replica set: easy to deploy, globalize, manage and replace using cloud resources

Example of different levels of consistency

- **Consistency level for WRITE operations**
  - One node in the replica set is the primary node
  - All writes are done at the primary node
  - Write consistency is guaranteed as “majority”: data has been written into a majority in the replica set, before confirming the write
- **Consistency levels for READ operations**
  - READ from a single replica
  - READ from a quorum and return the most updated result
  - READ from ALL replicas

Key expectations for designing big data services

- **Check the consistency, availability and partition tolerance when you use existing systems**
  - Very hard subject!
  - Also link to partitioning, scaling, service discovery and consensus (previous lectures)
- **Support the right ones when you design and implement big data systems**
  - Based on your data/use cases/applications

Key expectations for designing big data services

- **Designers: which one do you support?**
  - ACID or BASE ?
  - support programmable consistency guarantees?
- **Programmers**
  - how do big data management services support ACID/BASE
  - can I program with different consistency levels?
- **Able to explain why we have data accuracy problems and other tradeoffs w.r.t. performance and consistency!**

Data services: Data Models and Composability

Data sources and domains

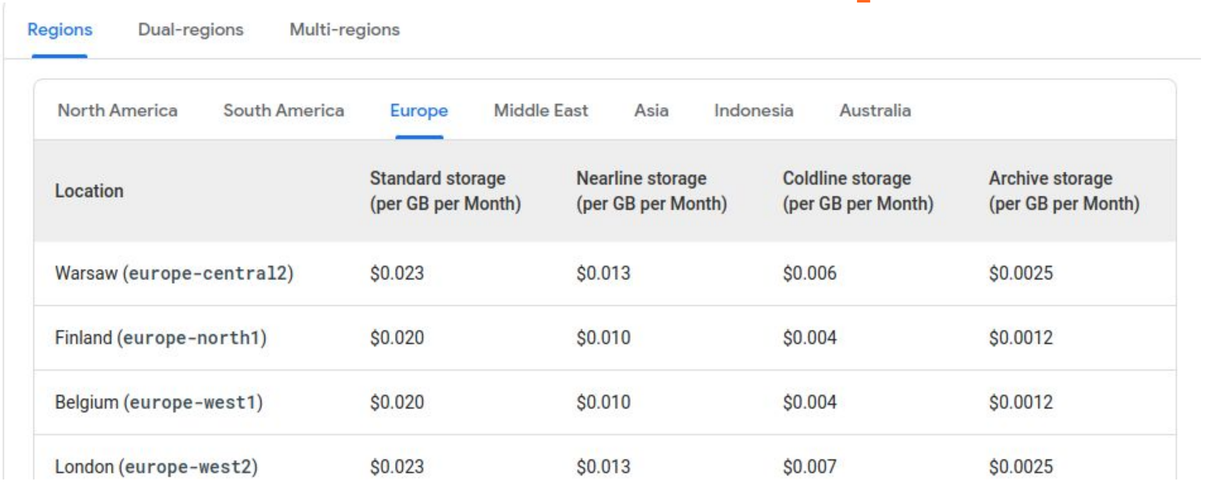
- **Social media data generated by human activities**
  - Facebook/Meta, Twitter, Instagram, etc.
- **Internet of Things (IoT)/Machine-to-Machine (M2M)/Industry 4.0**
  - data generated from monitoring of equipment, infrastructures and environments
- **Advanced sciences data generated by advanced instruments**
  - Earth observation from Sentinel satellites
- **Personal and disease information**
  - E.g. COVID data
- **Business-related customer data**
- **Asset management and lodging**
  - E.g., bookings, cars, accommodations
- **Software systems**
  - E.g., logs and test results

Data at rest

- **At rest**
  - Distributed file systems/object storages
    - *In big data we have a lot of files with different data formats*
  - Data in a set of databases
- **Multiple types of big data analytics with high concurrent/parallel data writes/reads**
- **Dealing with different data access/analytics frequencies:**
  - Organize data into **hot, warm and cold data**
  - **Tenant-wide vs data product-wide** configuration

Example from the service viewpoint

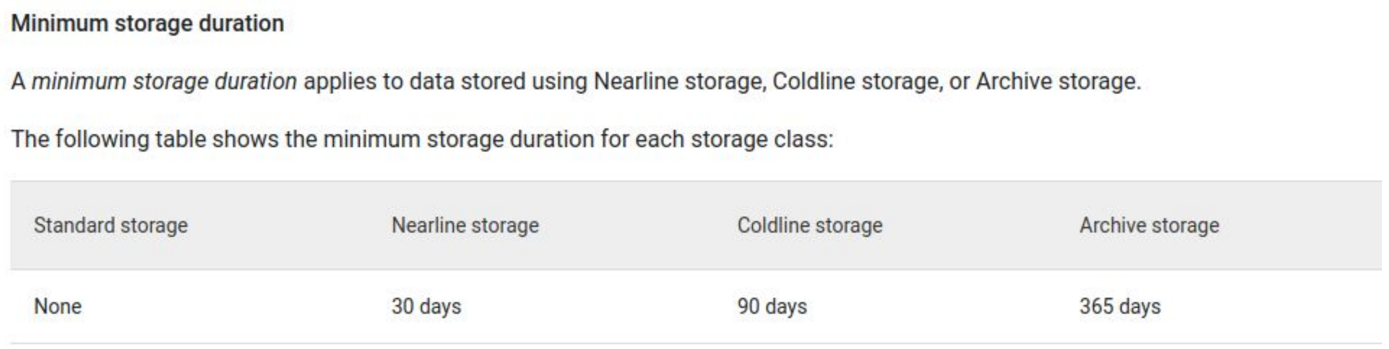
Google Storage: https://cloud.google.com/storage/pricing#europe



Regions | Dual-regions | Multi-regions

North America | South America | Europe | Middle East | Asia | Indonesia | Australia

Location	Standard storage (per GB per Month)	Nearline storage (per GB per Month)	Coldline storage (per GB per Month)	Archive storage (per GB per Month)
Warsaw (`europe-central2`)	$0.023	$0.013	$0.006	$0.0025
Finland (`europe-north1`)	$0.020	$0.010	$0.004	$0.0012
Belgium (`europe-west1`)	$0.020	$0.010	$0.004	$0.0012
London (`europe-west2`)	$0.023	$0.013	$0.007	$0.0025



Minimum storage duration

A *minimum storage duration* applies to data stored using Nearline storage, Coldline storage, or Archive storage.

The following table shows the minimum storage duration for each storage class:

Standard storage	Nearline storage	Coldline storage	Archive storage
None	30 days	90 days	365 days

Understanding developer concerns

- **Identifying data models**
  - first focus on data models representing data in big data platforms
    - *Before deciding technology that can help to implement the data model*
  - how *many data models* does the platform need to support?
- **Identifying data management technologies**
  - based on "multi-dimensional service and data properties" a technology for data management is selected
    - performance, scalability, interoperability, extensibility, etc.
  - costs and expertises and team requirements for management

Data models vs data access technologies

- **Data models explain structure and organization of the data to be stored and analyzed**
    - very important for deciding technologies and techniques used for data analytics
    - Complex analytics might require use to deal with different models
- **(Generic)Data connectors**
    - allow analysis programs to access data from different sources
    - do heavy lifting work for data load/extract

Data models

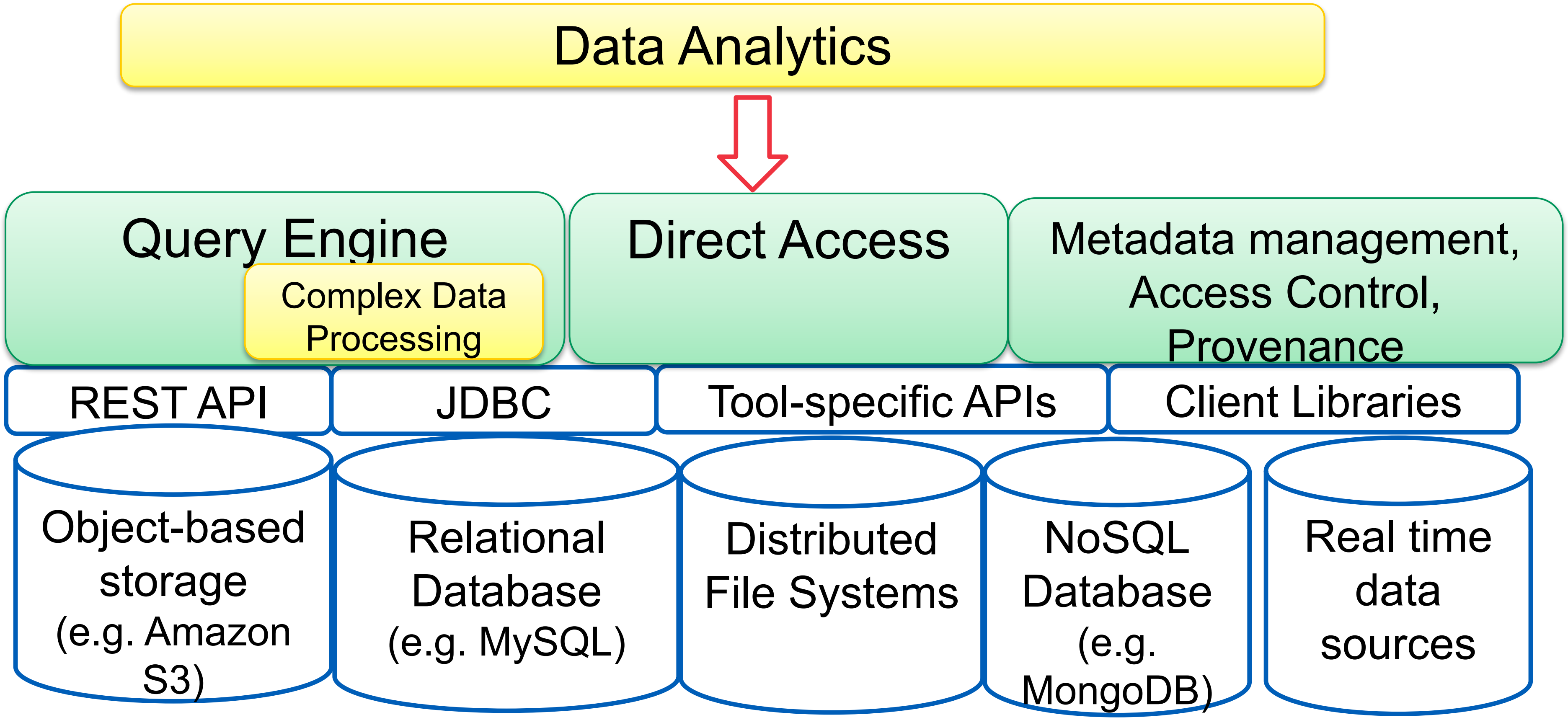
- **Data models**
  - File with different structures
  - Relational data model
  - Key-Value data model
  - Document-oriented model
  - Column family model
  - Graph model
  - Vectorization

Big data: both single type of data and combined multiple types of data with very large scale

- **Some are also seen in "no big data"**
- **Some are *specifically designed to address big data and ML***

Some important aspects when deciding data models

- **Structured data, semi-structured data and unstructured data**
  - diverse types of data
- **Schema flexibility and extensibility**
  - cope with requirement changes
- **Normalization and denormalization**
  - do we have to normalize data when dealing with big data (and storage is cheap)?
  - but data consistency maybe a problem!
- **Making data available in large-scale analysis infrastructure**
  - data is for analytics


Data Analytics
Query Engine
Complex Data Processing
Direct Access
Metadata management, Access Control, Provenance
REST API
JDBC
Tool-specific APIs
Client Libraries
Object-based storage (e.g. Amazon S3)
Relational Database (e.g. MySQL)
Distributed File Systems
NoSQL Database (e.g. MongoDB)
Real time data sources

Query engines with SQL-style

- **Analytics with big data databases**
  - NoSQL or NewSQL but they are very scale
  - E.g., Aurora, Cosmos, BigQuery
- **Analytics with federated databases**
  - Using scalable analytics engines to connect to different databases
  - Analytics using SQL-style queries or workflows
- **From the analytics: the developer is familiar with the traditional way**
  - SQL-on-Hadoop, SQL for data stream, etc. (covered in other lectures)

Presto + other as an example

- **Presto (used by Facebook and many others)**
  - distributed query engine
  - decoupled from storage
  - integration with different databases
  - very large-scale with many nodes
- **Analytics: interactive analytics, seconds – minutes**
  - SQL style

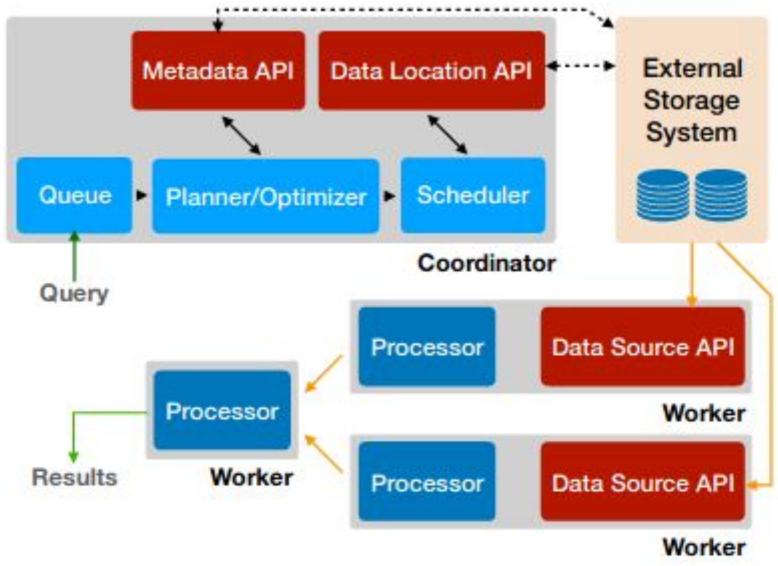


Figure source: *Presto: SQL on Everything* *https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8731547&tag=1*

Composability: support multiple types of data

- **Real-world applications need different types of databases!**
  - it is easier to use a single type of database, but it might not work for real projects
- **New use cases required different datasets and different analytics**
  - E.g. machine learning/AI
- **Strong set of APIs, connectors and client libraries**
  - for providing data to different analytics frameworks

Polyglot Big Data models/systems

- **A platform might need to provide multiple supports for different types of data → different data models**
  - single, even complex, storage/database/data service cannot support very good multiple types of data
- **A single complex application/service needs multiple types of data**
  - examples: logs of services, databases for customers, real-time log-based messages

Polyglot persistence is inevitable for many use cases

Example: monitring/maintenance situations

- **Subjects to be monitored (e.g., equipment, house, animal)**
  - usually in relational/document databases with different updates/management
- **Their monitoring data**
  - are time series measurements, video, images, etc. updated in real time
- **Complex analytics**

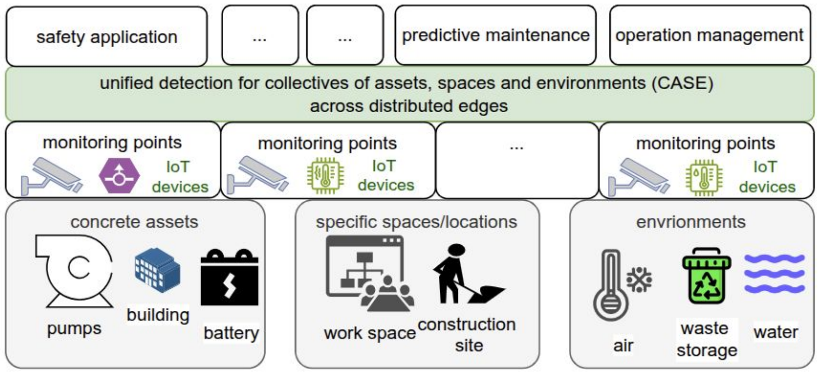


Fig. 1. Unified detection management for collectives of assets, spaces, and environments at the edge. Applications need to analyze multiple related entities in a CASE.

Figure source: Truong & Nguyen, 2024
https://research.aalto.fi/en/publications/analytics-feature-space-a-novel-framework-for-interoperable-edge-

Using a combination of different databases/storage

- **Pros**
  - start with simple, reasonable solutions
  - incrementally extend the platform according to the need/new requirements
  - use the best features from individual software
- **Cons**
  - different types of data must be linked
    - *each type requires a different model*
  - provide a collection of APIs
  - many software stacks

Example: blob (binary large object) for images/video

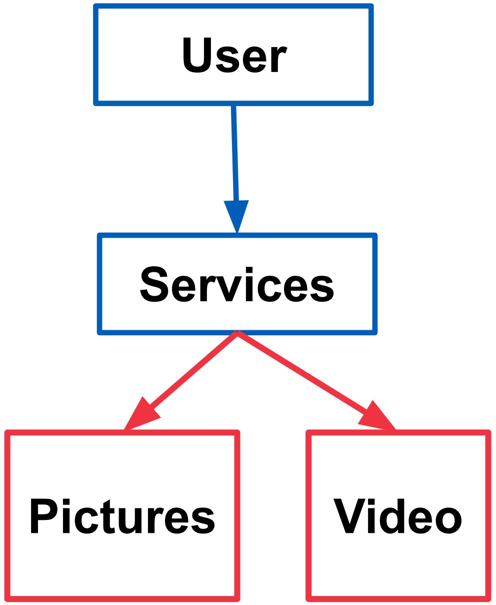
- Pictures, documents, big log files, images, video, backup data

Storage

- Distributed file systems or blob/object storage

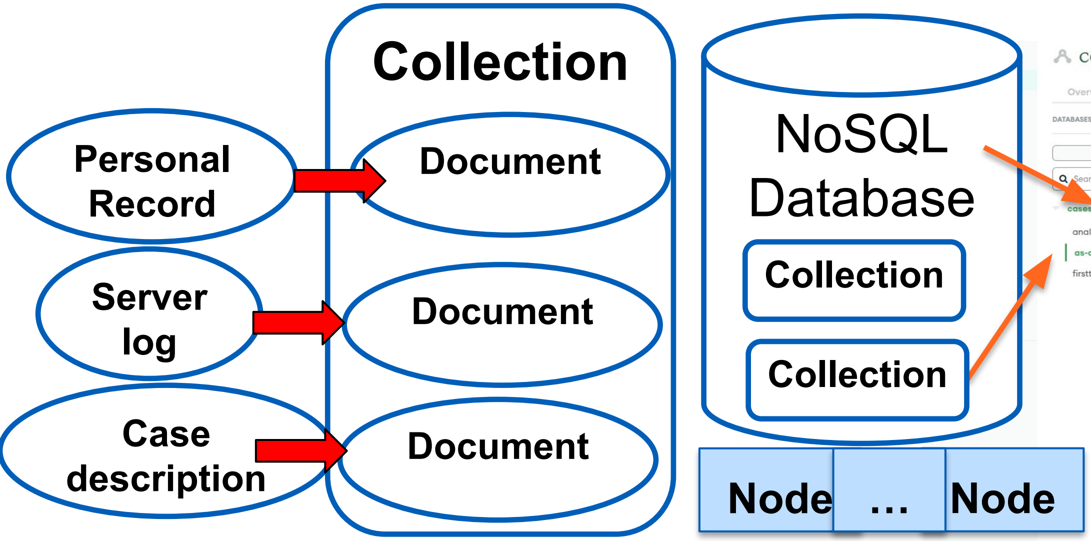
Implementations

- File systems: NFS, GPFS, Lustre (http://lustre.org/), Hadoop File systems
- Blob storage: Amazon S3, Azure Blob storage, OpenStack Swift, Minio
- Simple API for direct access (GET/PUT)



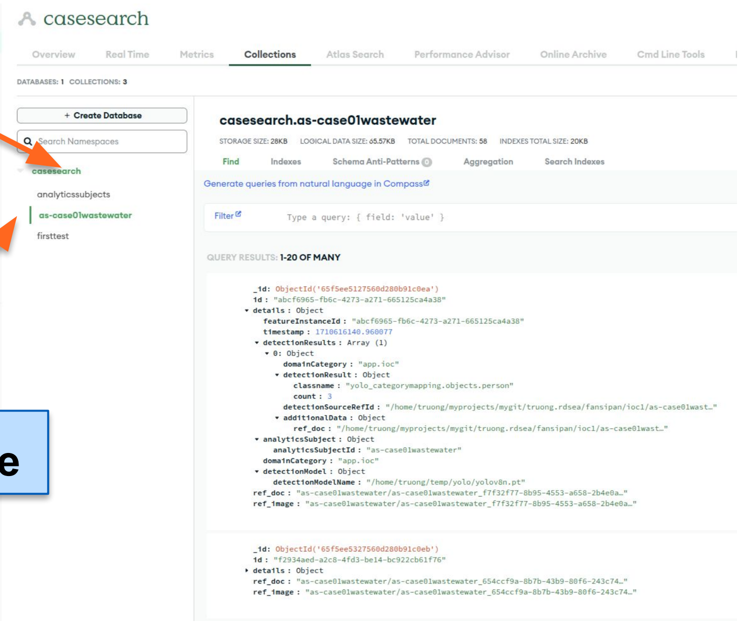
Aalto University
School of Science

Example: document-oriented model for detailed information about cases



Flexible schema

(Cluster)



DATABASES: 1 COLLECTIONS: 3

casesearch.as-case01wastewater

STORAGE SIZE: 28KB LOGICAL DATA SIZE: 65.57KB TOTAL DOCUMENTS: 58 INDEXES TOTAL SIZE: 20KB

QUERY RESULTS: 1-20 OF MANY

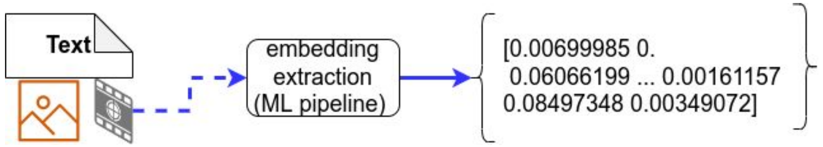
```
  _id: ObjectId('65f5ee5127560d280b91c0ea')
  id : "abcf6965-fb6c-4273-a271-665125ca4a38"
▾ details : Object
    featureInstanceId : "abcf6965-fb6c-4273-a271-665125ca4a38"
    timestamp : 1710616140.960077
  ▾ detectionResults : Array (1)
    ▾ 0: Object
        domainCategory : "app.ioc"
      ▾ detectionResult : Object
          classname : "yolo_categorymapping.objects.person"
          count : 3
        detectionSourceRefId : "/home/truong/myprojects/mygit/truong.rdsea/fansipan/ioc1/as-case01wast…"
      ▾ additionalData : Object
          ref_doc : "/home/truong/myprojects/mygit/truong.rdsea/fansipan/ioc1/as-case01wast…"
  ▾ analyticsSubject : Object
      analyticsSubjectId : "as-case01wastewater"
    domainCategory : "app.ioc"
  ▾ detectionModel : Object
      detectionModelName : "/home/truong/temp/yolo/yolov8n.pt"
  ref_doc : "as-case01wastewater/as-case01wastewater_f7f32f77-8b95-4553-a658-2b4e0a…"
  ref_image : "as-case01wastewater/as-case01wastewater_f7f32f77-8b95-4553-a658-2b4e0a…"

  _id: ObjectId('65f5ee5327560d280b91c0eb')
  id : "f2934aed-a2c8-4fd3-be14-bc922cb61f76"
▸ details : Object
  ref_doc : "as-case01wastewater/as-case01wastewater_654ccf9a-8b7b-43b9-80f6-243c74…"
  ref_image : "as-case01wastewater/as-case01wastewater_654ccf9a-8b7b-43b9-80f6-243c74…"
```

Many servers hosting database

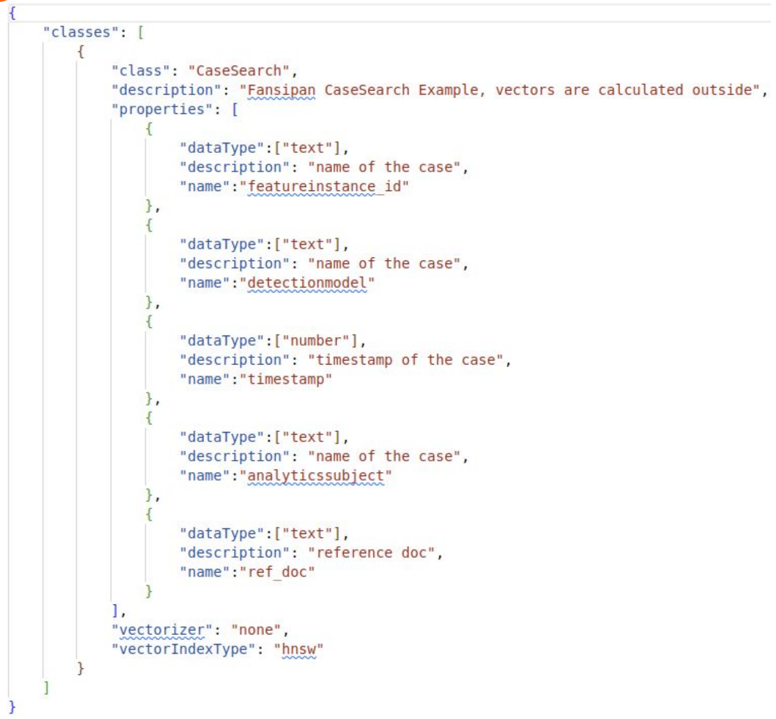
Example: Vector databases for similarity

Example of a Weaviate schema

- **Emerging in the age of LLMs/MLs**
- **Store vector embeddings**
  - vector embeddings are output of ML embedding models



- **Key designs**
  - vector management
  - distance functions
  - similarity search



```
{
    "classes": [
        {
            "class": "CaseSearch",
            "description": "Fansipan CaseSearch Example, vectors are calculated outside",
            "properties": [
                {
                    "dataType":["text"],
                    "description": "name of the case",
                    "name":"featureinstance_id"
                },
                {
                    "dataType":["text"],
                    "description": "name of the case",
                    "name":"detectionmodel"
                },
                {
                    "dataType":["number"],
                    "description": "timestamp of the case",
                    "name":"timestamp"
                },
                {
                    "dataType":["text"],
                    "description": "name of the case",
                    "name":"analyticssubject"
                },
                {
                    "dataType":["text"],
                    "description": "reference doc",
                    "name":"ref_doc"
                }
            ],
            "vectorizer": "none",
            "vectorIndexType": "hnsw"
        }
    ]
}
```

Using large-scale multi-model database services

- **Multi-model database services**
- **a data service can host different data models**
  - able to store data using different types of data models
    - *relational tables, documents, graphs, etc.*
  - can be a virtual service atop other database services
- **Benefits**
  - the same system (query, storage engine)
- **Example**
  - Microsoft Azure Cosmos, OrientDB, ArangoDB, Virtuoso

Using Data Lake/Lakehouse

- **Principles**
  - Massive of datasets, in different collections, in different formats, in different types of data storages
    - *Internal/external data, operational/analytical data*
    - *Raw/clean/training data*
  - For multiple types of analytics/ML
- **Example of technologies:** Apache Hudi, Delta Lake, Iceberg
- **Related concepts**
  - Data mesh (data, infrastructures, services and governance for domain-oriented data products)

Data Lake/Lakehouse: example

Allows different analytics

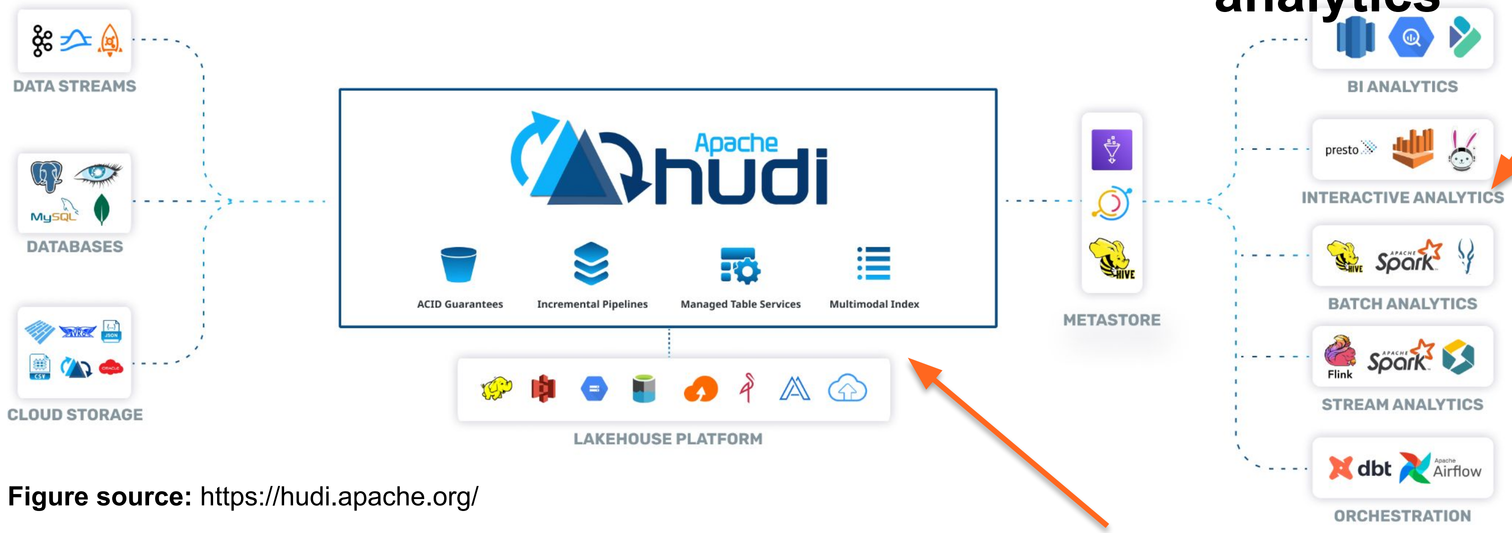


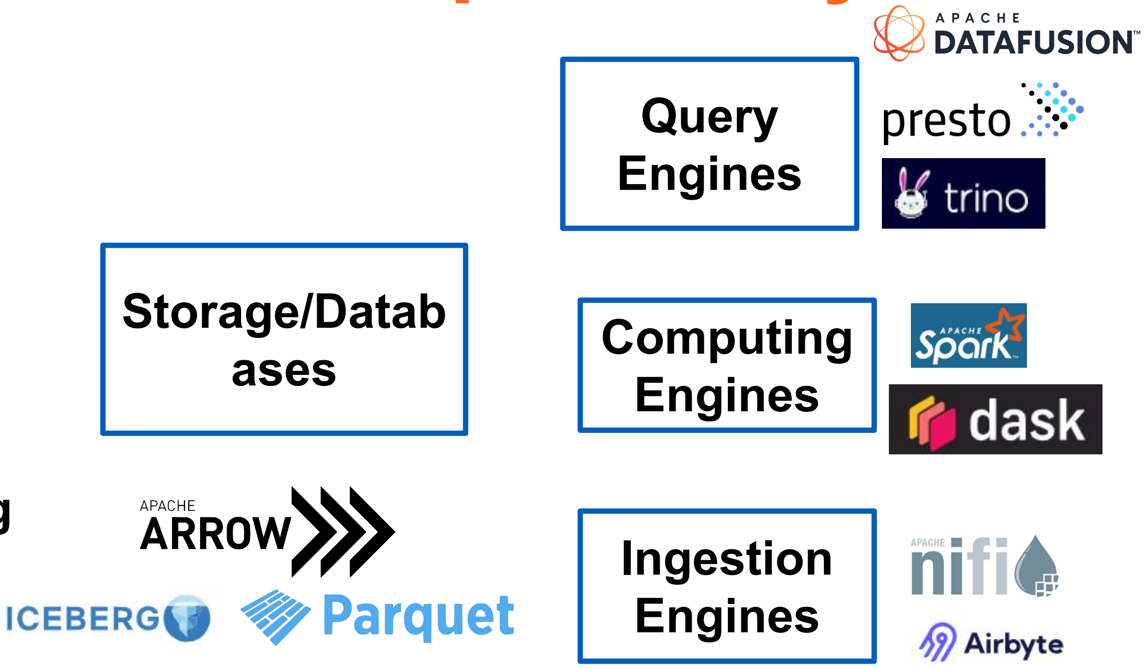
Figure source: https://hudi.apache.org/

Massive of datasets, in different collections, in different formats, in different types of data storages for multiple types of analytics/ML

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

Trends in modernizing data services: composibility and interoperability

- **Breakdown important building blocks:**
  - storage/databases, query, ingestion, etc.
- **Support open standards and open sources**
  - Apache Icerberg, Parquet, Arrow
- **Combine them according to workloads, costs, interoperability, etc.**



Metadata and Data Resources

Metadata about data resources

- **Metadata characterizes data assets stored in databases/storage/data lake or data products delivered by services**
  - for management, liability, fairness, regulation compliance
- **Important types of metadata**
  - governance (creators, update, retention, security setting, etc.), quality of data (accuracy, completeness, etc.)
  - designed for common and specific cases
- **Remember metadata is data!**
  - techniques and models for ingestion, collection and management
- **Tools:** Google Data Catalog, Apache Atlas, Amundsen, Linkedin DataHub, OpenLineage

Example of Metadata

Key design:

- Metadata comes from different sources via well-design processes
- Different access models for metadata
- Complex ingestion of metadata
- Graph view of metadata

Amundsen

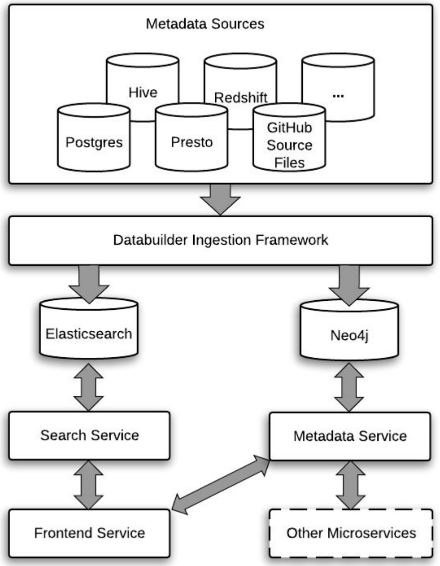


Figure source: https://www.amundsen.io/amundsen/architecture/

DataHub

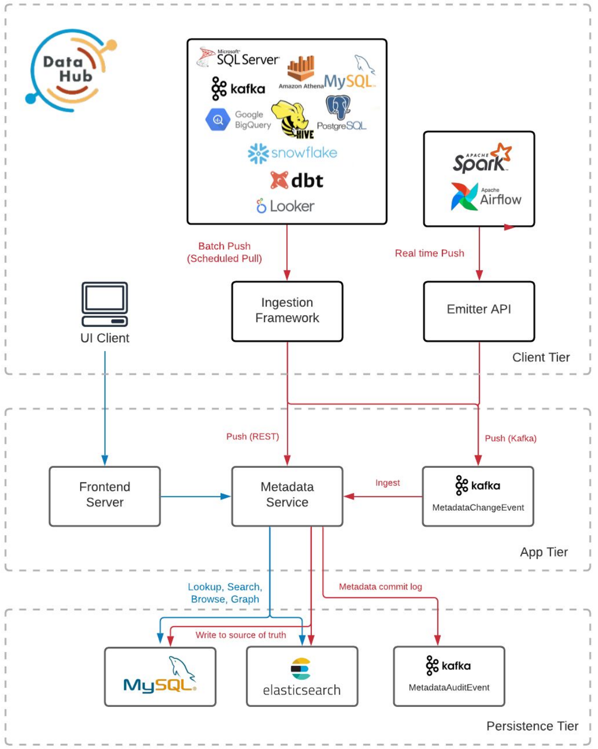


Figure source: https://datahubproject.io/docs/architecture/architecture

Using a centralized data discovery

- **Approach 3:**
  - use a dedicated data discovery service and ask the service for metadata about data
- **Design your metadata schemas**
  - reuse common metadata schemas
    - *e.g., DataHub Metadata (https://datahubproject.io/docs/metadata-modeling/metadata-model),*
  - design and add new entities/schemas
- **Integration metadata ingestion**
  - capturing metadata in service deployment, data ingestion pipelines, etc.
  - ingest metadata to the discovery service; manage changes
  - query of metadata

Key points

- **Spend your time to think about data sharding strategies**
  - common concepts and concrete implementations in your choice of database technologies, also in connection to data nodes deployment
- **Work on understanding the relationships in big database provisioning**
  - distribution (multiple nodes, data centers, geo-distributed locations), sharding/replication, security, multitenancy, availability
- **Focus on understanding features for programming consistency**
- **Practice with some key database/store technologies**
  - individual, federated, multi-model, and data lake/lakehouse

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io