Aalto University
School of Science

Data Services in Big Data Platforms

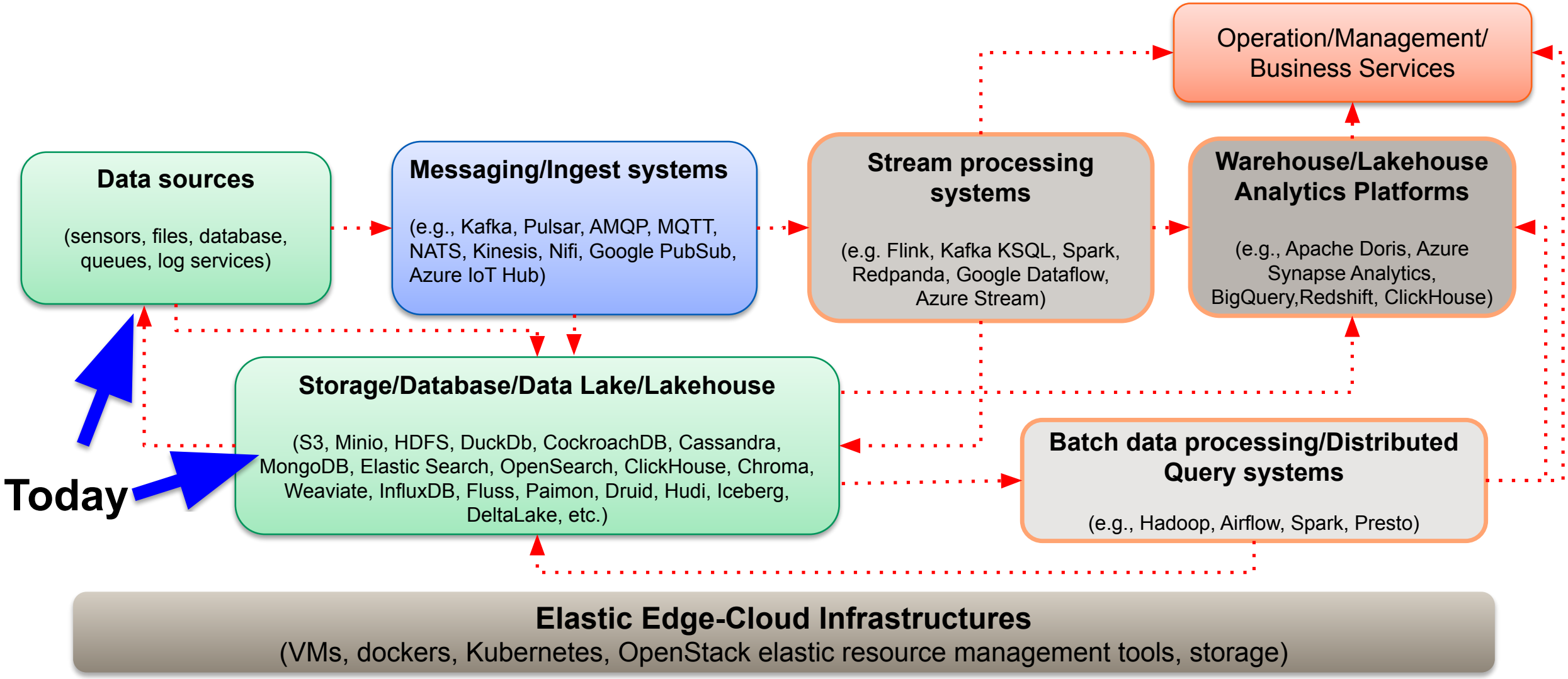
Hong-Linh Truong
Department of Computer Science
linh.truong@aalto.fi

21.1.2026

Learning objectives

- Understand consistency, availability and partition tolerance issues in design and programming
- Study designs of data modeling and organization based access frequency and values of data
- Understand composability in big data services
- Understand the role of metadata management

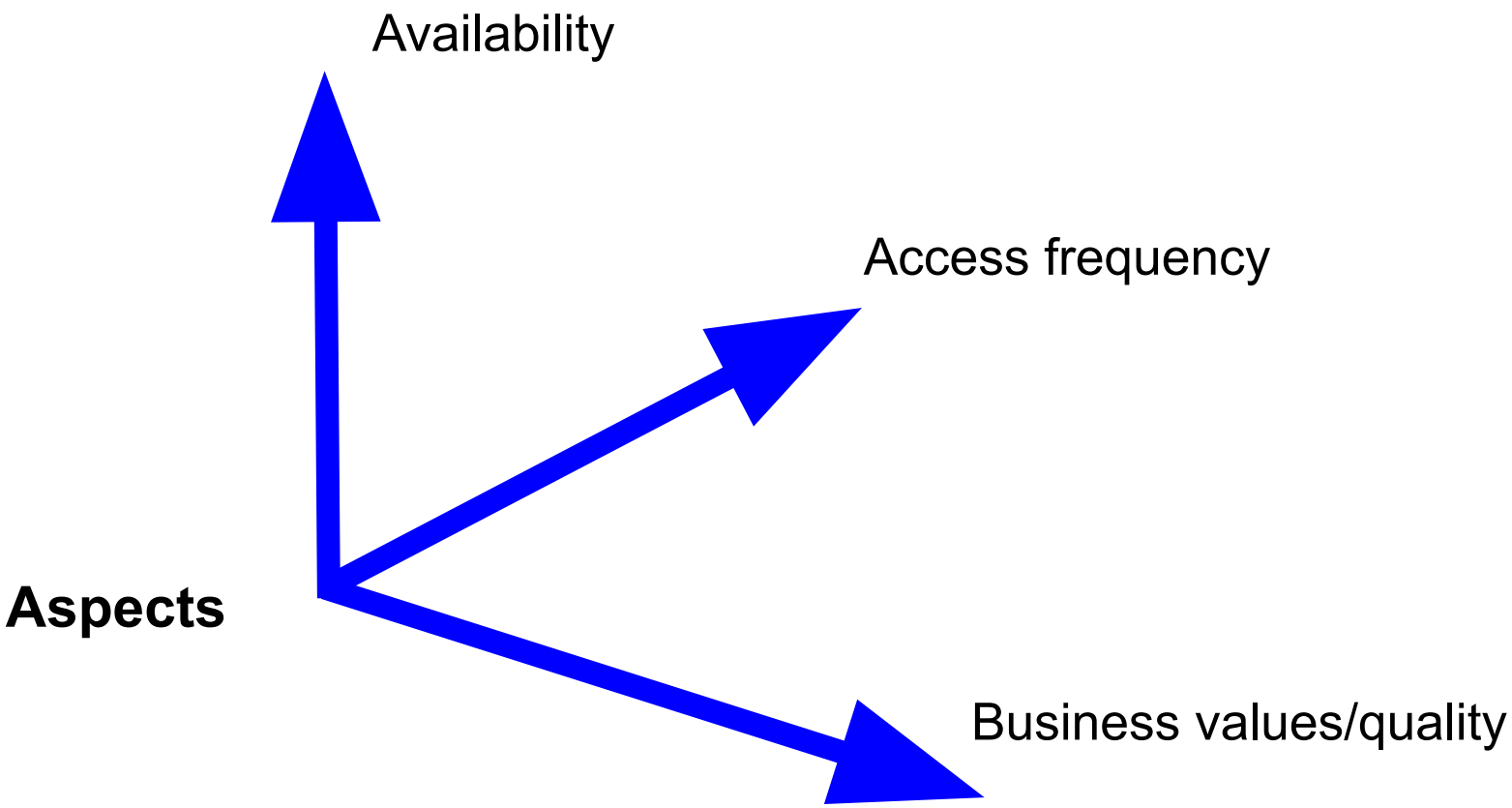
Our big data at large-scale: the big picture in this course



Understanding requirements from tenants

- **Identify data organization strategies**
  - support tenants to categorize, arrange and manage data
- **Identifying data models**
  - first focus on data models representing data in big data platforms
    - before deciding technology that can help to implement the required 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

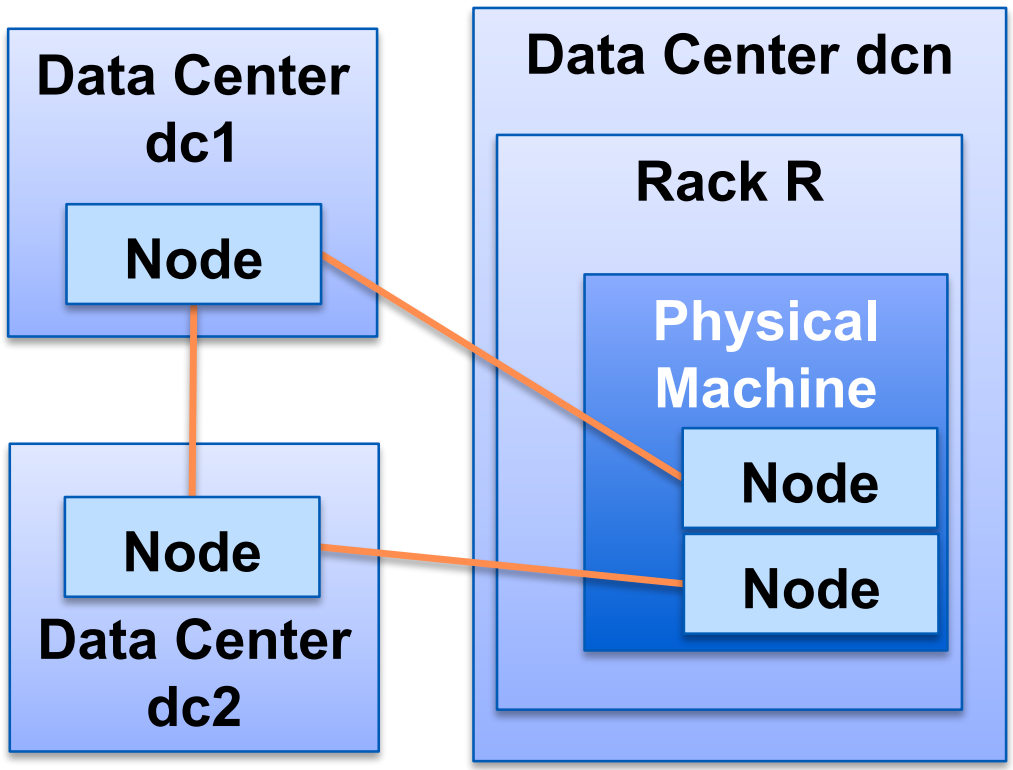
Data Organization



Consistency, Availability and Partition Tolerance

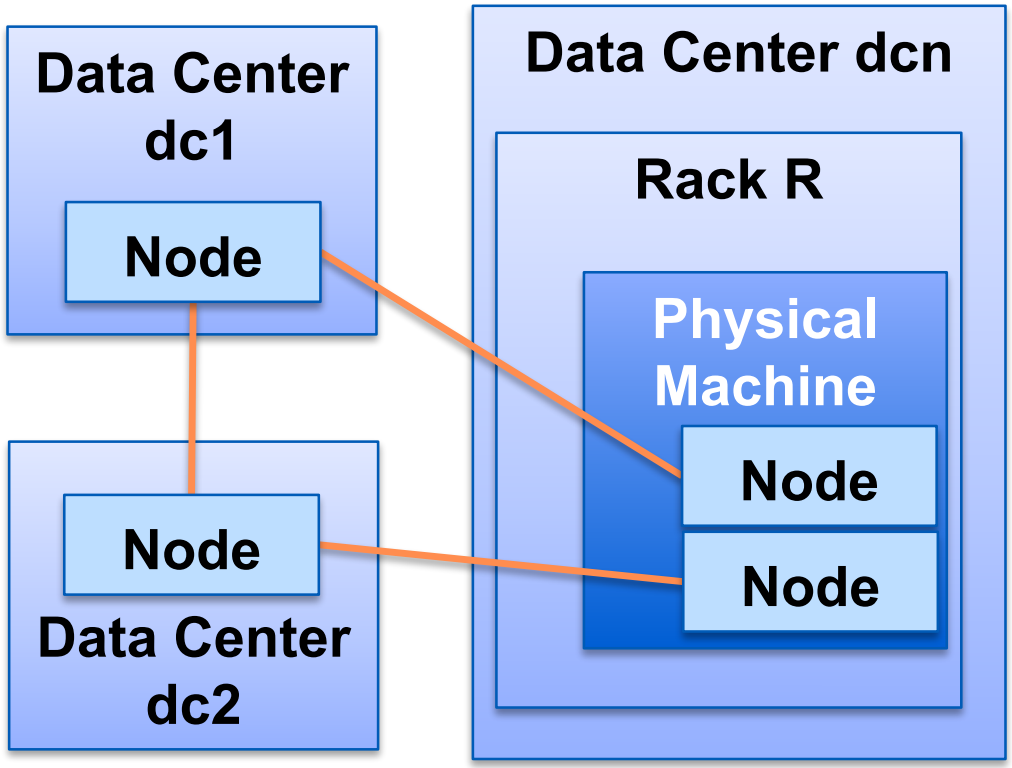
Data and Compute coupling

- Bring data to compute approach
  - computation/processing is intensive
  - high performance CPU/GPU, memory, and low latency communications among compute nodes.
- Bring compute to data (data-centric) approach
  - large-scale data is hard to move
  - so move/schedule processing tasks into machines storing data
- In practice - a hybrid configuration with data gravity problems
  - voluminous and intensive data/datasets accessed, but compute/processing is also intensive (e.g., need many cores and GPUs)
  - compute nodes are located next to data nodes
  - data is moved and mounted to compute nodes via low latency, high bandwidth networks



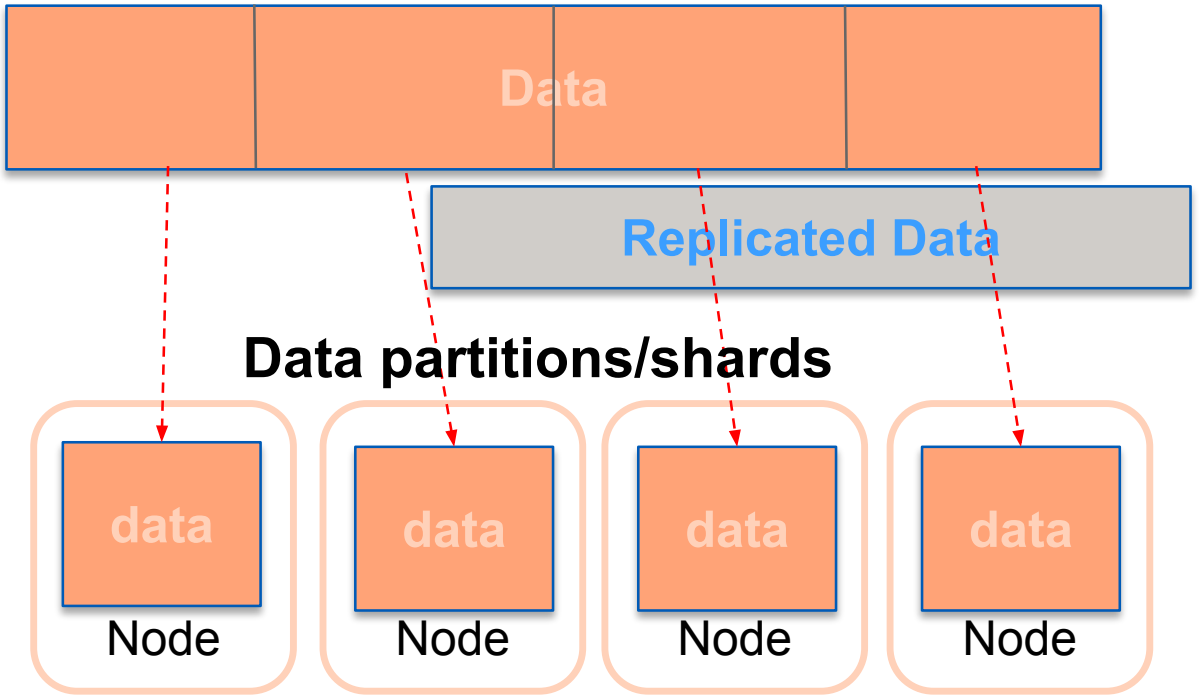
Edge-cloud, multicloud, hybrid on-prem - cloud and supercomputer

Data is not stored in and analyzed using a single machine

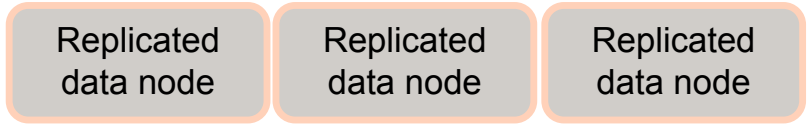


Cluster of nodes (virtual/physical machines) in multicloud, hybrid cloud and supercomputer

View from analytics application



Platform view

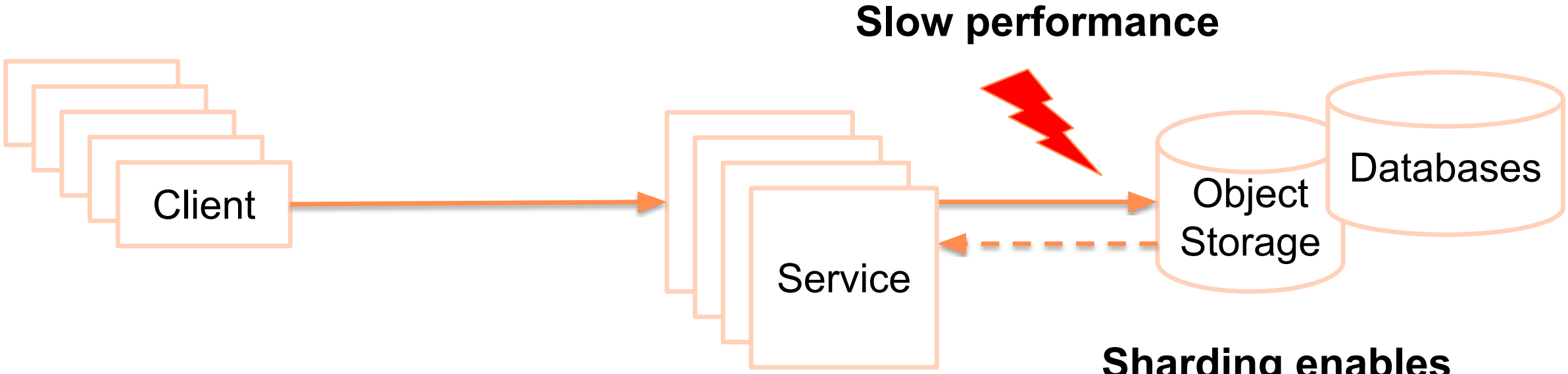


Application level vs system data organization

- Application level view
  - e.g., files and databases
  - types, characteristics and semantics of data are dependent on the application
- System level view
  - (distributed and parallel) file systems, e.g., POSIX-compliant systems
    - like what you see in your computer
  - object storage
  - block storage

⇒ The concepts of replication, consistency, management, etc. apply for all levels but with different techniques/tools

Performance problems between service serving request and data store

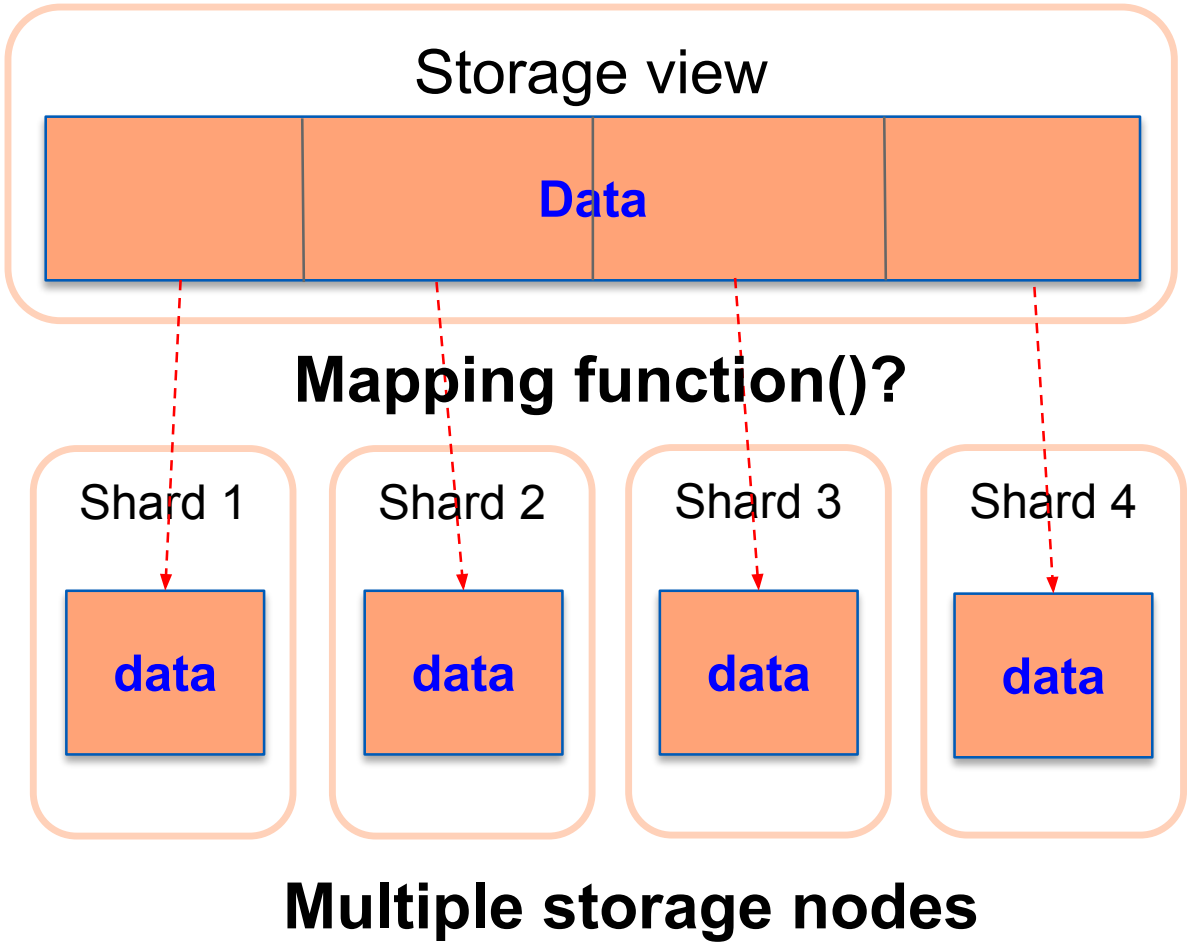


Sharding enables parallel access (data parallelism)

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

Data Partitioning: Principles

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



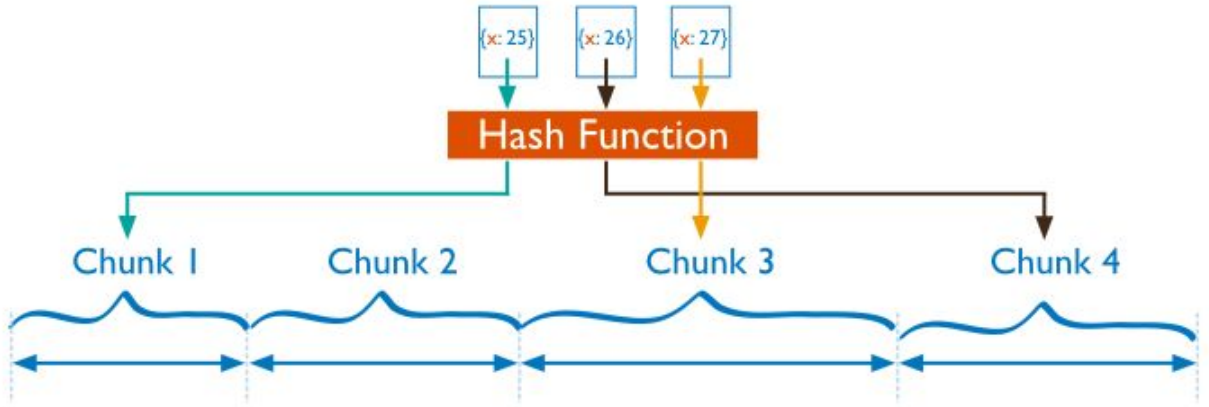
Sharding Strategies

- 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

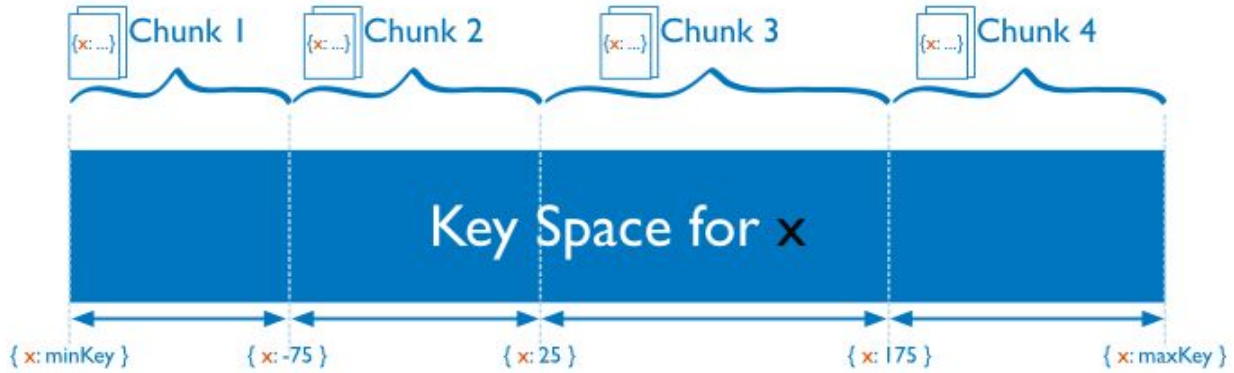
Sharding patterns/strategies reading: *https://msdn.microsoft.com/en-us/library/dn589797.aspx*

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)
…;
```

Define partition names

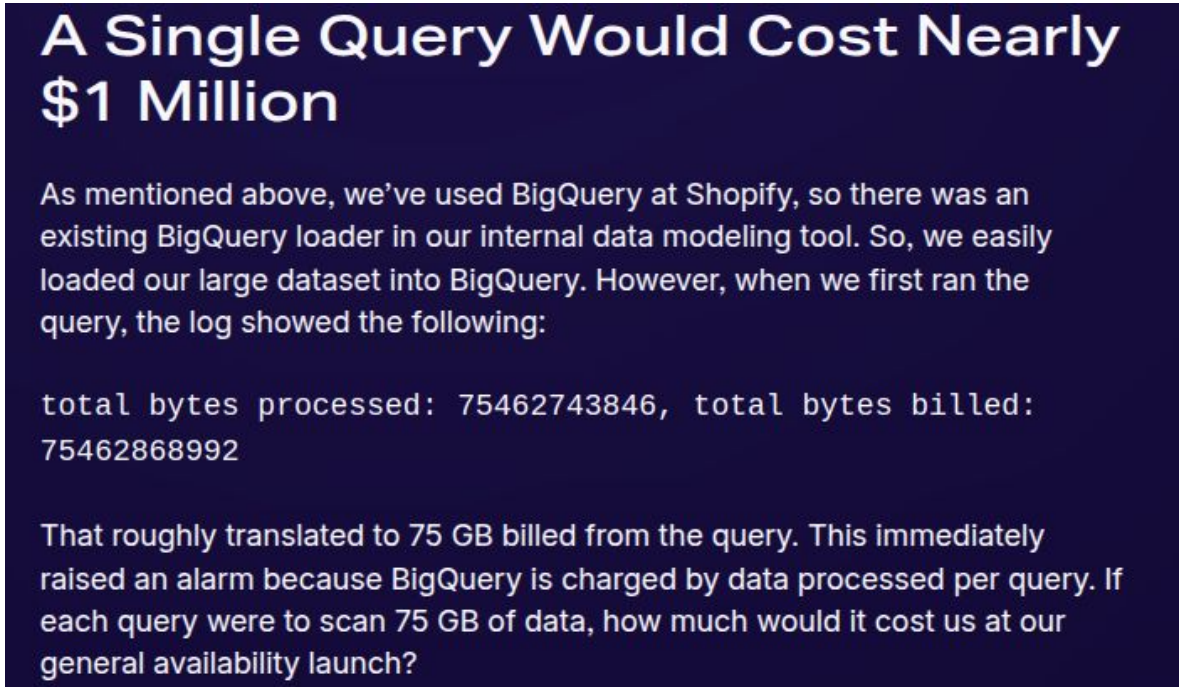
Indicate partition info

```
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



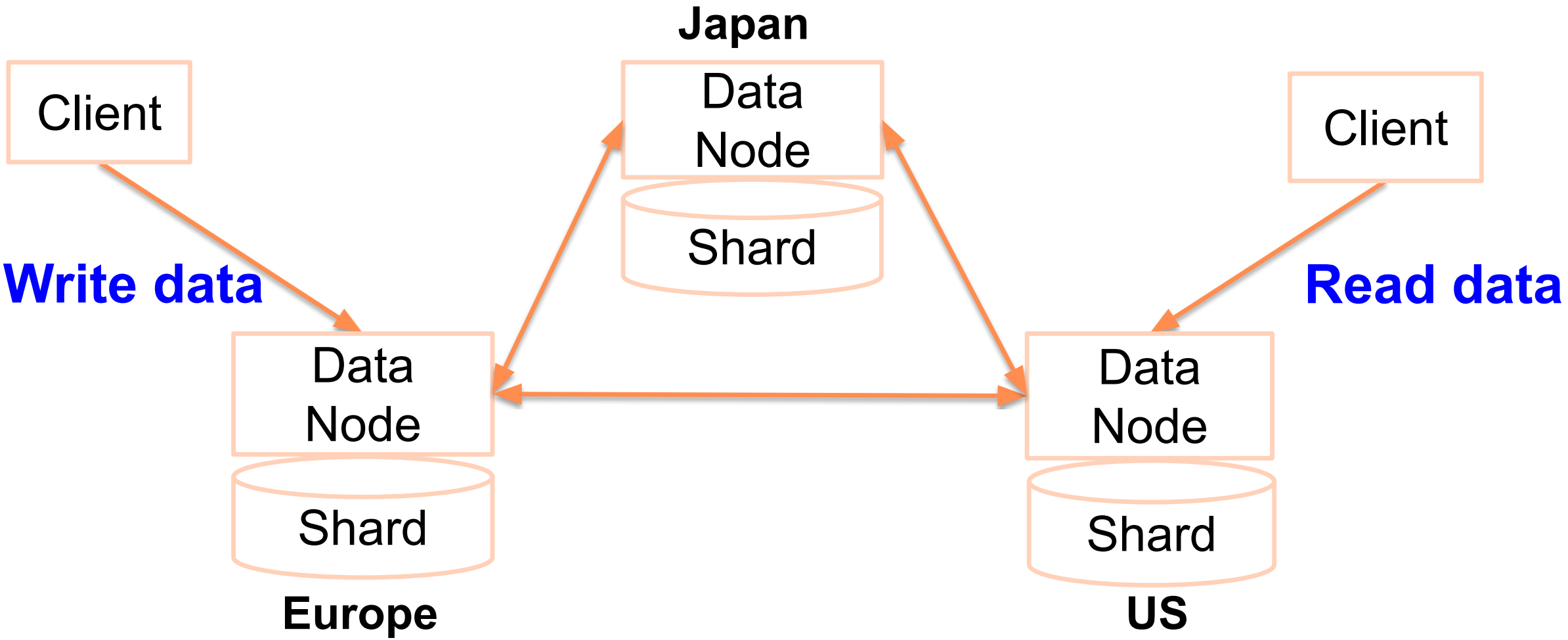
A Single Query Would Cost Nearly $1 Million

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

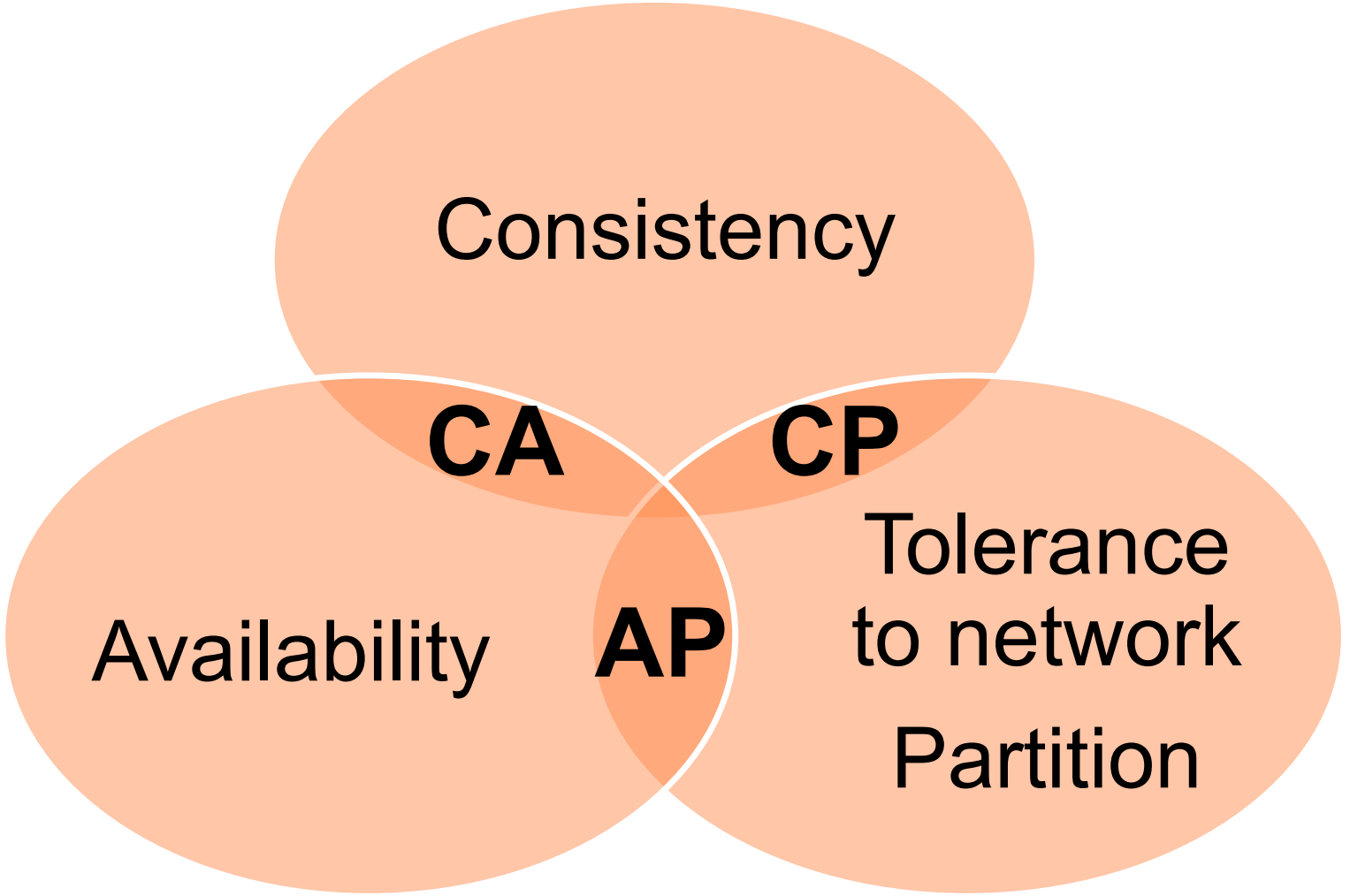
We must carefully study how such properties are supported in big data storage/databases

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)

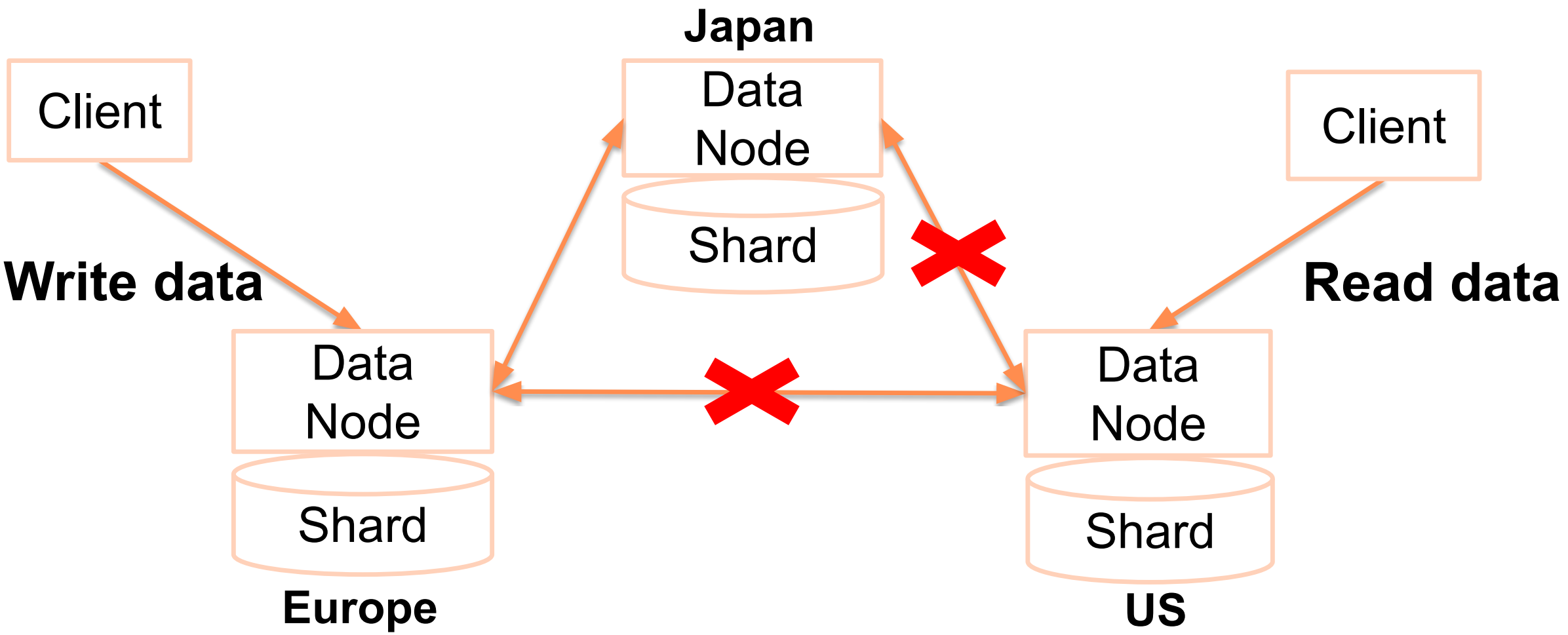
CAP Theorem

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

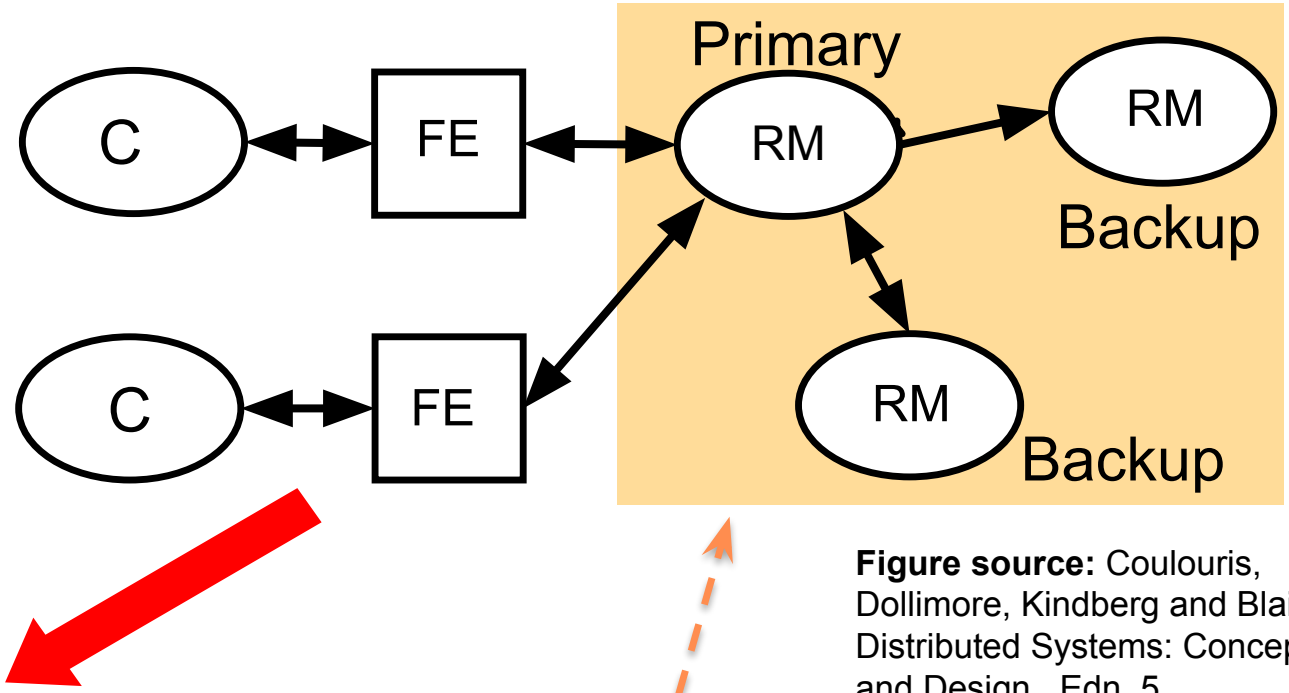


Figure source: Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5

For causal consistency

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 (1)

- 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 and background)
- Support the right ones when you design and implement big data systems
  - based on your data/use cases/applications

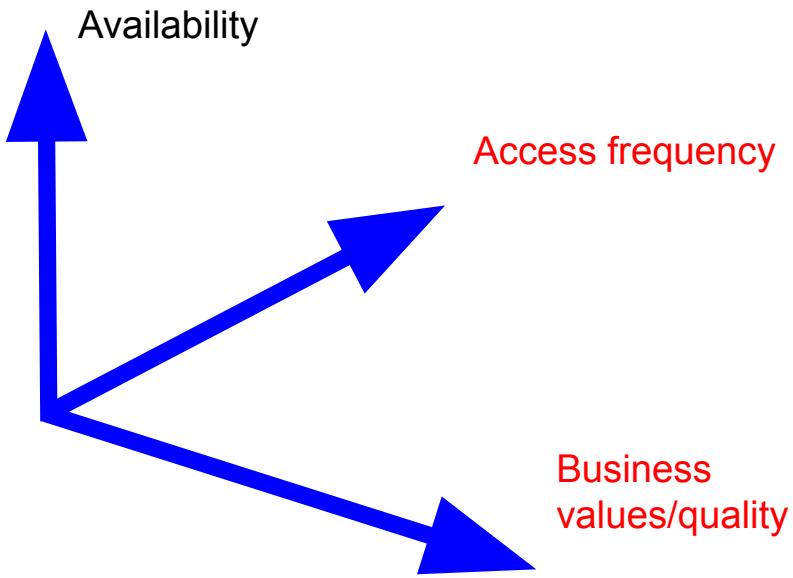
Key expectations for designing big data services (2)

- 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 trade offs w.r.t. performance and consistency!

Data services: Data organization based on access frequency and data values

Data sources and domains

- Domain Data
  - Social media data
  - Internet of Things (IoT)/Machine-to-Machine (M2M)/Industry 4.0
  - Advanced sciences data
  - Personal and disease information
  - Business-related customer data
  - Asset management and lodging
  - Software systems Data



- How to support data organization according the domain requirements?

Data access frequency and business values/quality

- Access frequency
  - organize data into **hot, warm and cold data**
  - “hot”, “warm”, “cold”: an abstraction reflecting the access frequency of data
- Business values/quality
  - organize data into **bronze, silver, and gold**
  - “bronze”, “silver”, “gold”: an abstraction reflecting the values and quality
- Both require suitable storage/data service architectures and appropriate data movement functions

⇒ Question 1: Any conflict? can gold but cold data?

⇒ Question 2: Do we need to know the content/semantics of data for such organizations?

Hot/cold storage: example from the service viewpoint

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

Europe (eu) | Hourly | ✓ Monthly

Standard storage	Nearline storage	Coldline storage	Archive storage
$0.026 / 1 gibibyte month	$0.015 / 1 gibibyte month	$0.007 / 1 gibibyte month	$0.0024 / 1 gibibyte month

Minimum storage duration

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

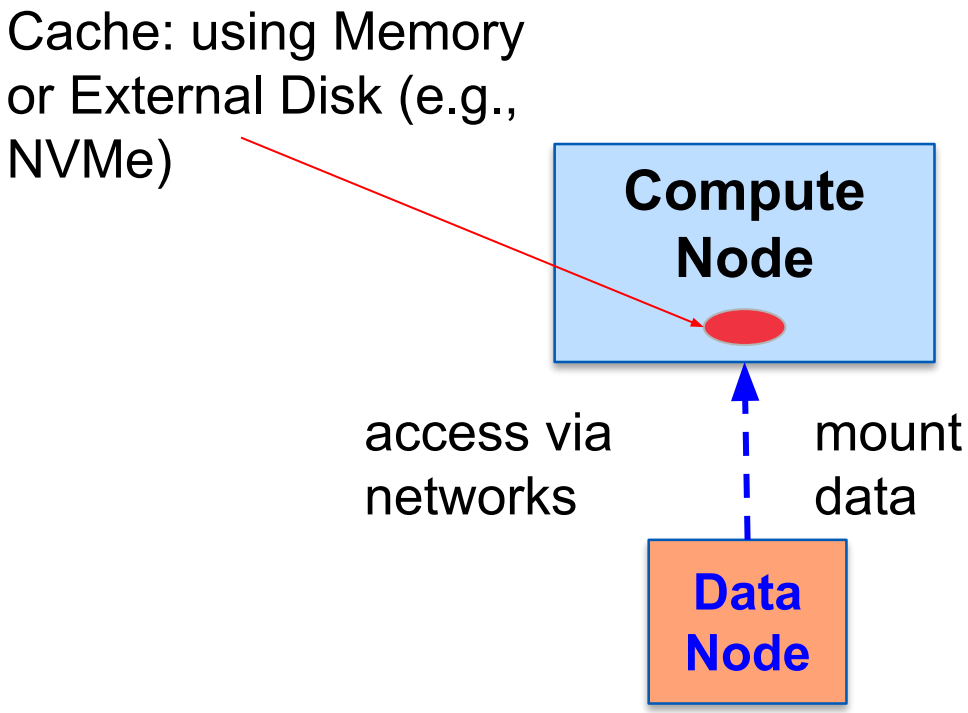
The following table shows the minimum storage duration for each storage class:

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

⇒ Question 2: Do we need to know the content/semantics of data for such organizations?

Caching Data

- Compute nodes separated from data nodes but access data frequently
  - loading big data takes time
- Cache data in compute nodes
  - memory for high performance
  - disk for high throughput
- Hardware plays an important role
  - networks between Compute and Data
  - memory and local hard disk
- Challenges
  - data consistency
  - application workloads: e.g., file read/write vs SQL



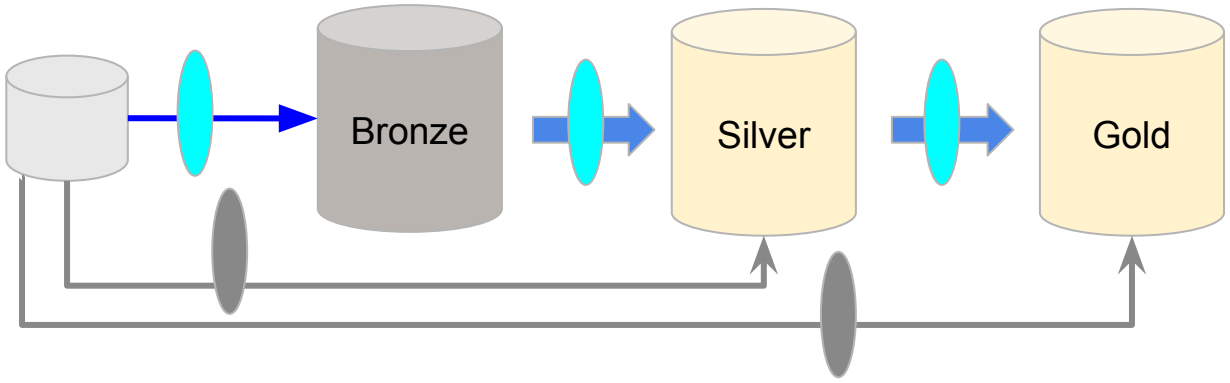
⇒ Question 2: Do we need to know the content/semantics of data for such organizations?

Hot/warm/cold design and implementation in the application view

- Define by applications
  - e.g., based on ingesting time or the age of data
- With underlying data systems support or not
  - data storage systems will automatically carry out the data movement tasks
  - one needs to implement complex functions or easy configurations
- Logic
  - defining constraints/characteristics of “hot”/“cold” data
  - developing/specifying data movement processes
  - scheduling the data movement

Organization in terms of data values

Data organized based on values, conceptually:



- Can be modeled, designed and managed in different ways
- Modern architectures
  - use datalake and lakehouse systems
  - leverage the same software stack for data pipelines
  - support data quality, performance, single-source-of-truth

Concrete example:
Medallion Architecture in Delta Lake

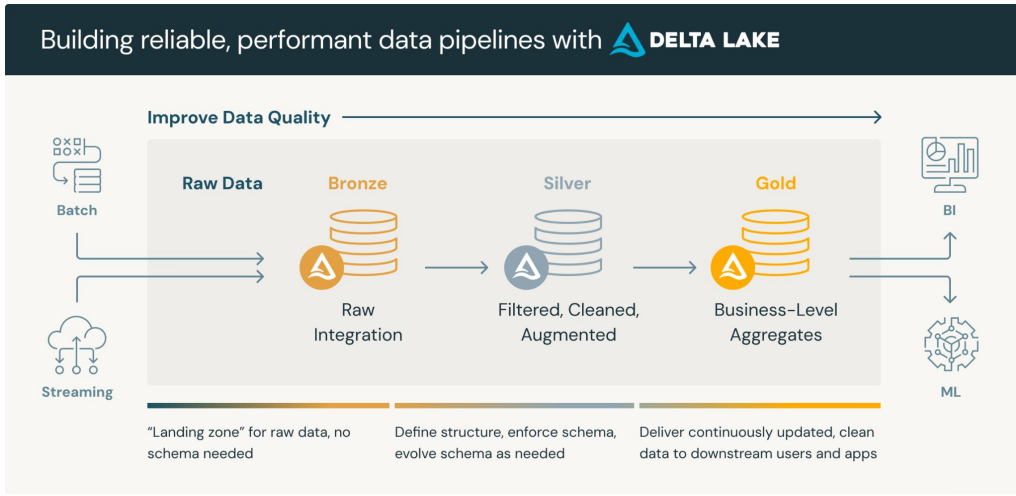


Figure source:
https://www.databricks.com/glossary/medallion-architecture

Data services and Composability

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

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

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

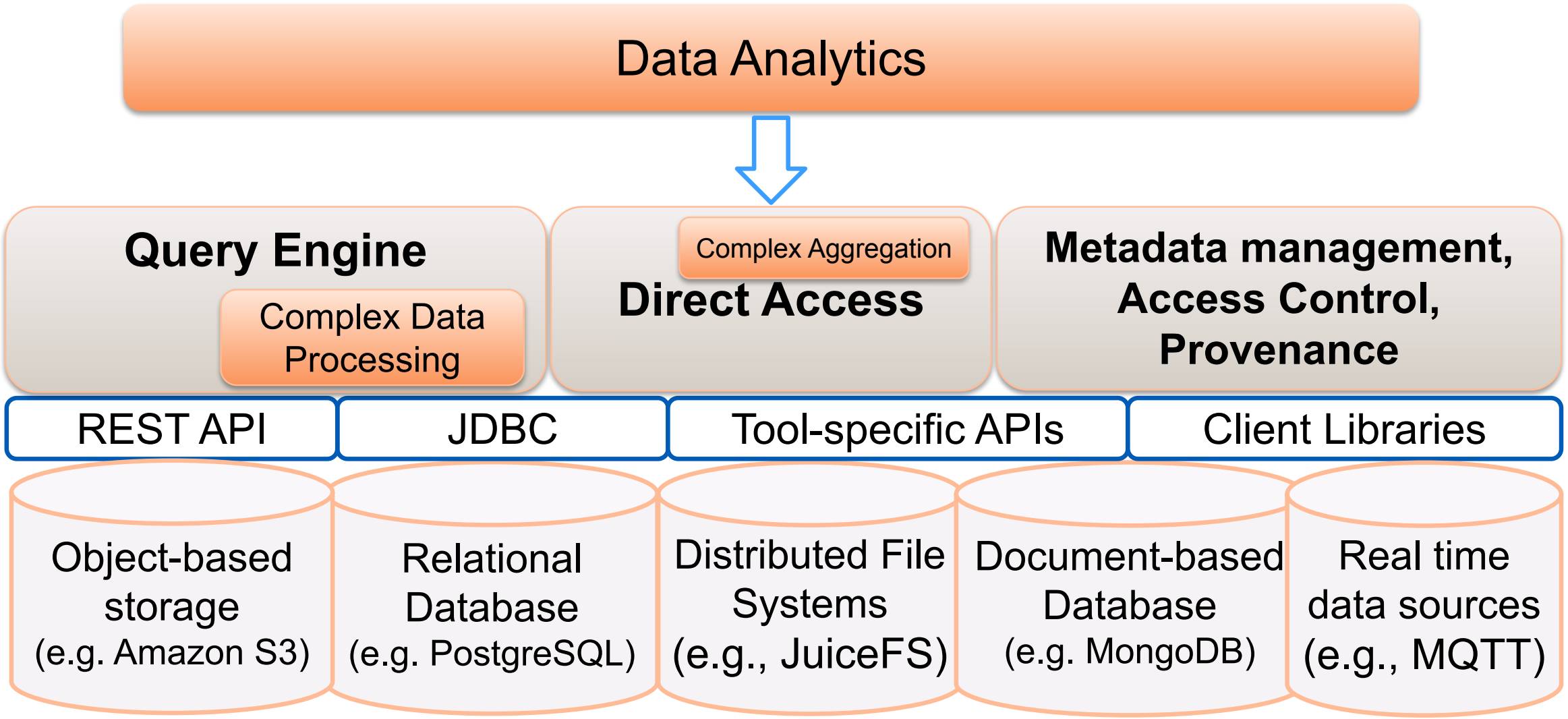
Composability: support multiple types of data

- 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
  - e.g. logs of services, databases for customers, real-time log-based messages

Polyglot persistence is inevitable for many use cases


Data Analytics
Query Engine
Complex Data Processing
Complex Aggregation
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. PostgreSQL)
Distributed File Systems (e.g., JuiceFS)
Document-based Database (e.g. MongoDB)
Real time data sources (e.g., MQTT)

Query engines

- SQL-based query engines
  - federation of data sources
  - using scalable analytics engines to connect to different data sources
  - analytics using SQL-style queries or workflows

    ⇒ SQL way is familiar with the developer
- API orchestration architectures
  - REST- or GraphQL-based orchestrations
  - workflows

    ⇒ complex merging and correlation of data from different sources

Example of SQL query engine: Presto

- 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

A sample of a setup for a distributed query engine:
https://github.com/rdsea/bigdataplatforms/tree/master/tutorials/basicdisquery

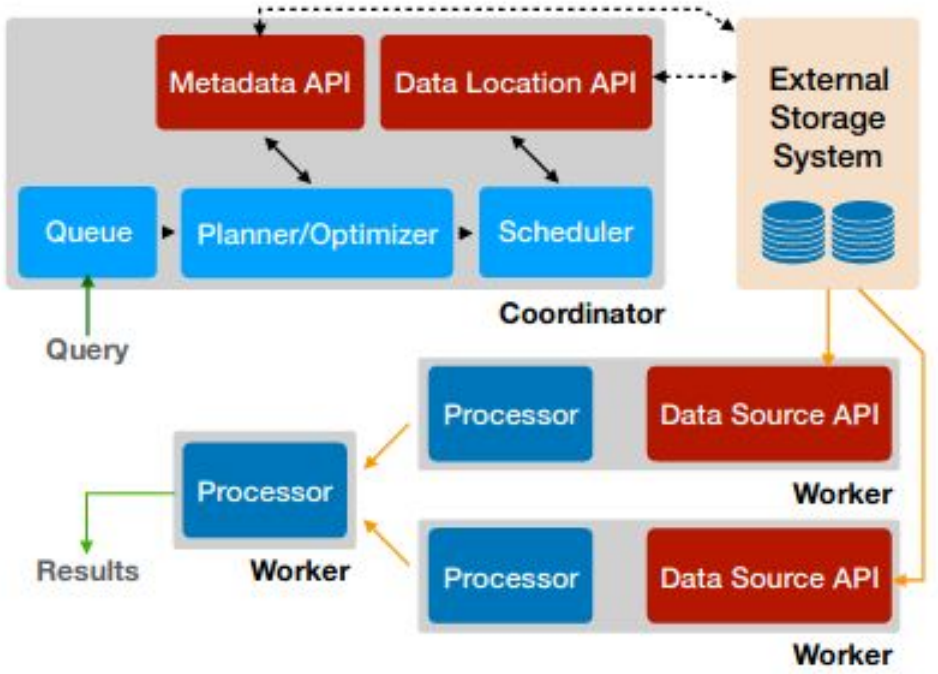


Figure source: *Presto: SQL on Everything https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8731547&tag=1*

Using a combination of different databases/storage

- Pros
  - start with simple, reasonable solutions
  - integrate and include legacy systems
  - incrementally extend the platform according to the need/new requirements
  - use the best features from individual software
- Cons
  - must link different types of data
    - each type requires a different model
  - solve challenges in data synchronization and integration for domain views
  - provide a collection of APIs to work different software

Example: monitoring/maintenance situations in digital twins, industrial controls

- Subjects to be monitored (e.g., equipment and building)
  - usually in relational/document databases
  - different update/management time scales
- Monitoring data
  - operational data (OT – operational technology), environment monitoring
  - are time series measurements, video, images, etc., updated in near real-time
- Complex analytics

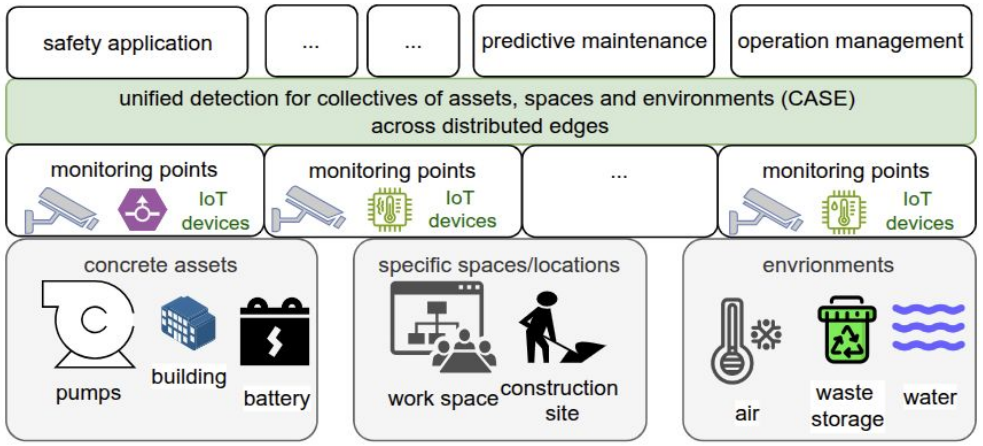


Fig. 1. Unified detection management for collectives of assets, spaces, and environments at the edge. Applications need to analyze multiple related entities in a CASE.

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

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

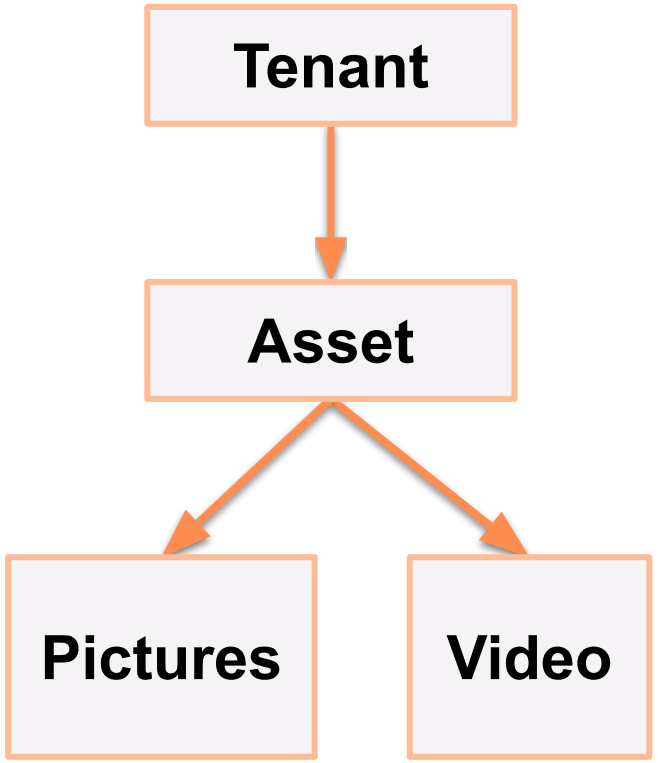
- Pictures/images, video, backup data

Storage

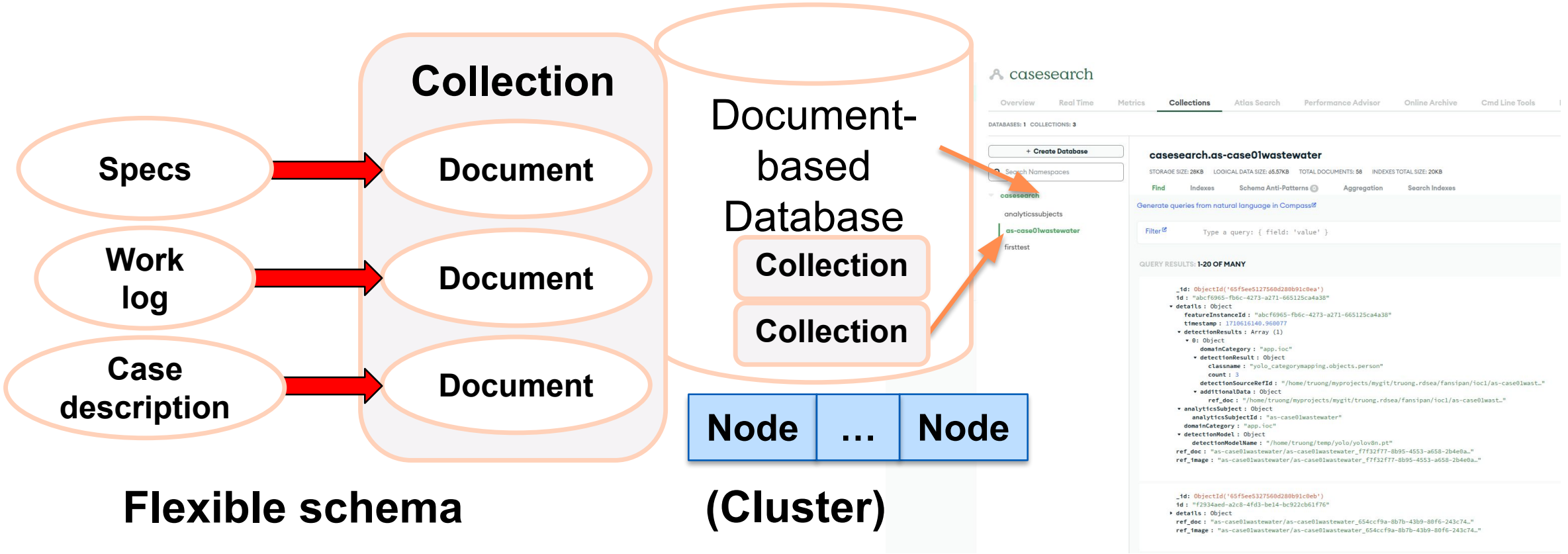
- Distributed file systems or blob/object storage

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)



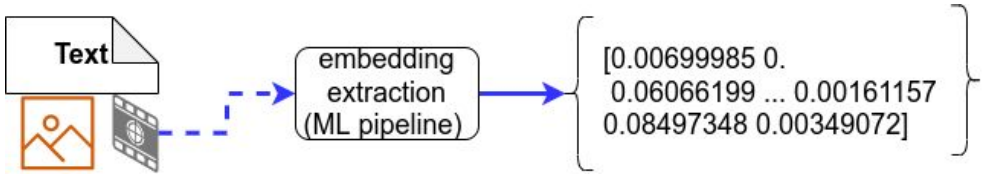
Example: document-oriented model for detailed information about cases, designs, etc.



Many servers hosting database

Example: vector databases for similarity search

- 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

Example of a Weaviate schema

```
{
    "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 Data Lake/Lakehouse

- 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

Data Lake/Lakehouse: example

Allows different analytics

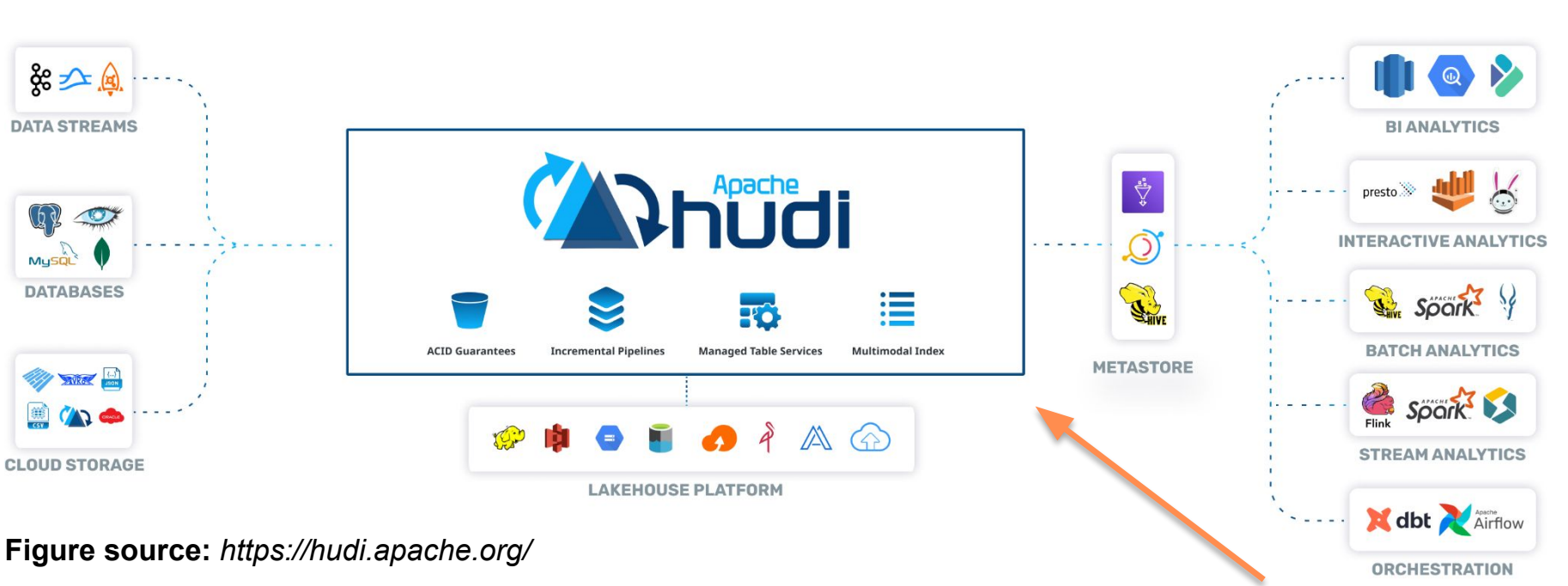


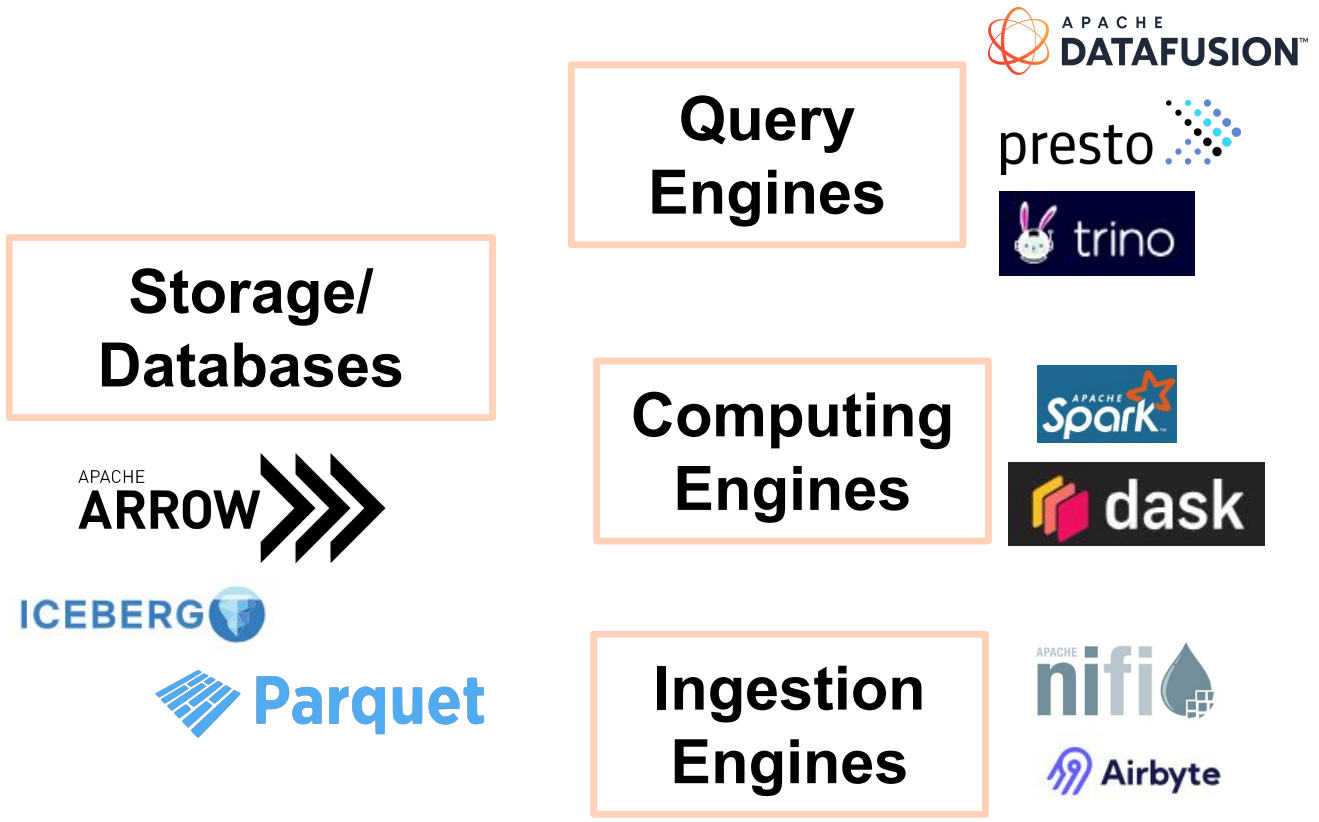
Figure source: *https://hudi.apache.org/*

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 Iceberg, Parquet, Arrow
- Combine them according to workloads, costs, interoperability, etc.



The Composable Data Management System Manifesto: *https://www.vldb.org/pvldb/vol16/p2679-pedreira.pdf*
Voltron Data. 2023. The Composable Codex. *https://voltrondata.com/codex*

Metadata and Data Resources

Metadata about data resources

- Data resources
  - are datasets, data streams, databases, and data services
  - cover raw data to data assets/products
- Metadata characterizes data resources 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!
  - where and when the metadata is captured
    - dependent on data flows in a platform
  - techniques and models for ingestion, collection and management

Example of Metadata tools

Key design:

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

Example:
https://docs.datahub.com/docs/metadata-modeling/metadata-model

Amundsen

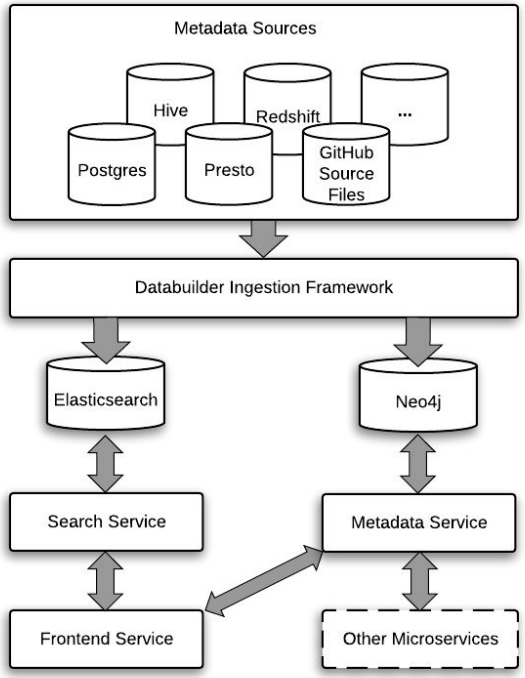


Figure source:
https://www.amundsen.io/amundsen/architecture/

DataHub

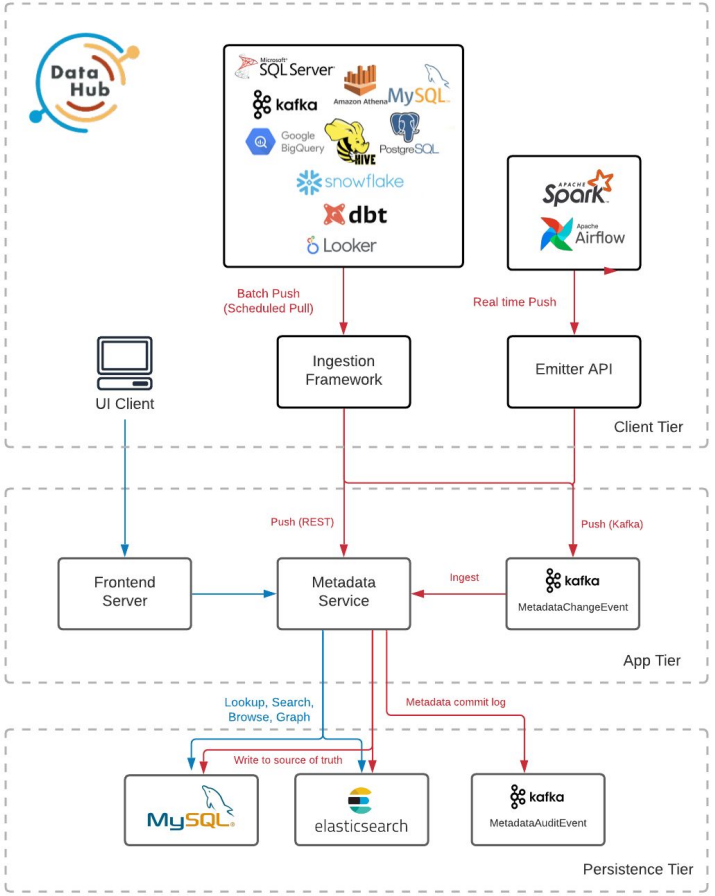


Figure source:
https://datahubproject.io/docs/architecture/architecture

other tools: Google Data Catalog, Apache Atlas, OpenLineage

Using a centralized data discovery

- Approach 3 (lecture 2):
  - 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

Summary: some key points for homework

- Understand data modeling and organization from multiple views
- 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
  - features for programming consistency and its effects
- Work on understanding the relationships in big database provisioning and composition
  - distribution (multiple nodes, data centers, geo-distributed locations), sharding/replication, security, multi-tenancy, availability
- Practice with some key database technologies
  - individual, federated, and data lake/lakehouse
- Implement metadata and lineage
  - metadata capturing: schema/type (what) and process (when and where)

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io

—

Kiitos
aalto.fi