Aalto University
School of Science

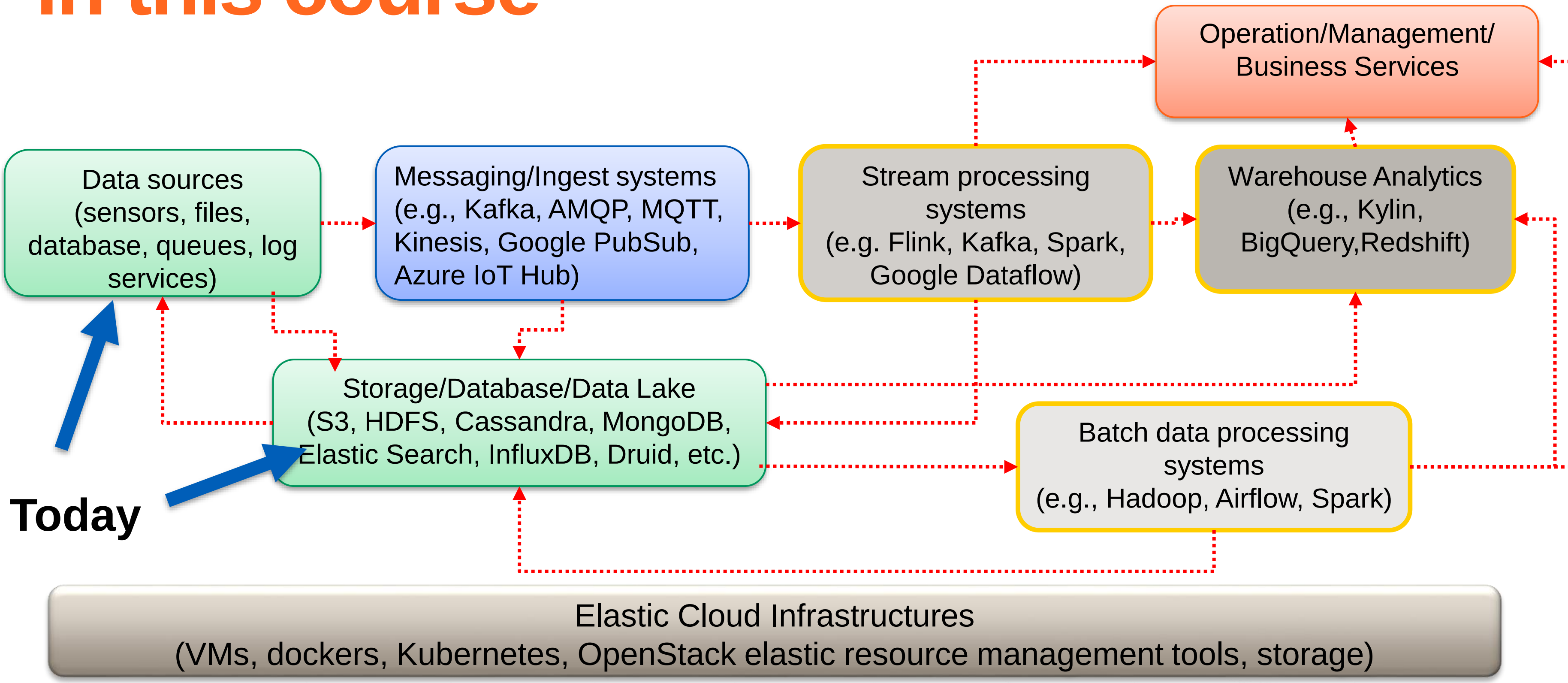
Big Data Storage and Database Services

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

Learning objectives

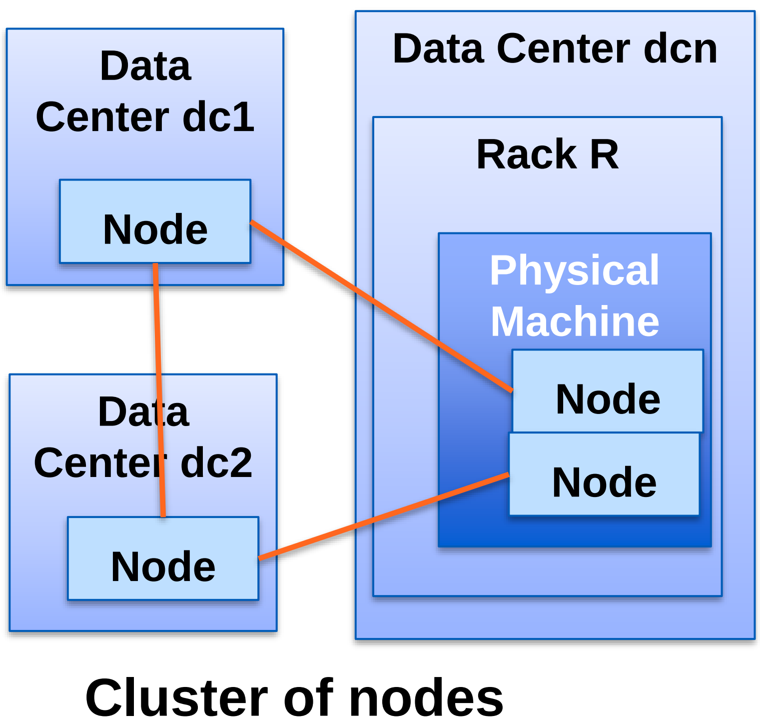
- Understand consistency, availability and partition tolerance issues in design and programming
- Study common data models and data management
- Understand the need of polyglot persistence and metadata management

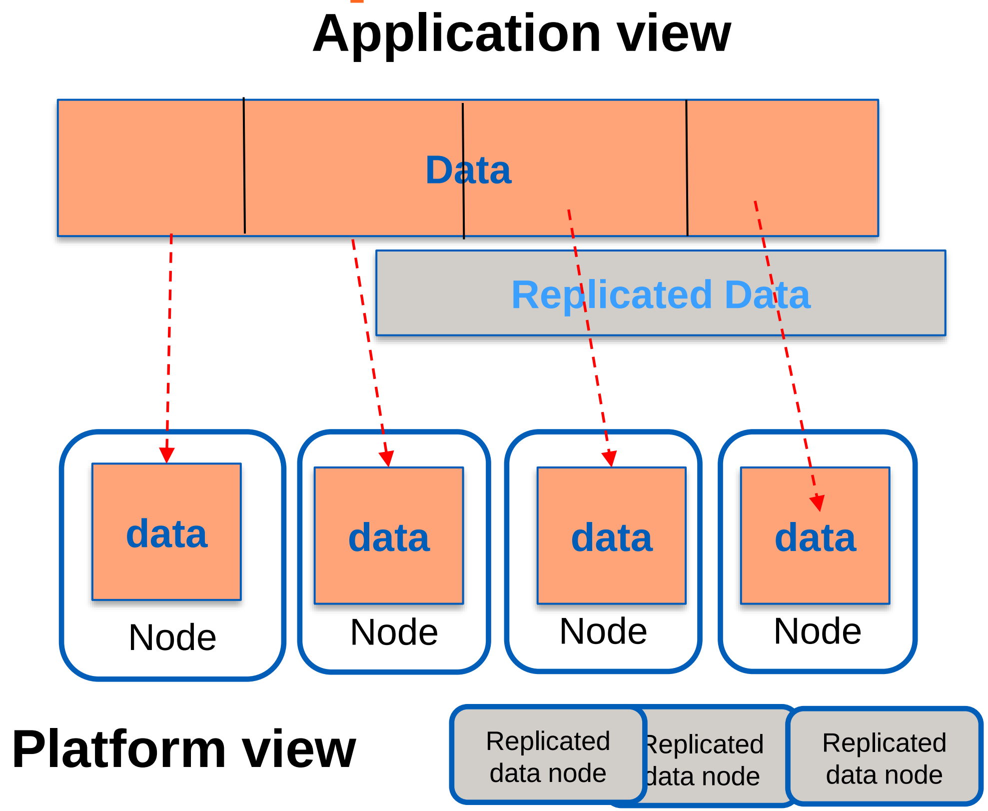
Big data at large-scale: the big picture in this course



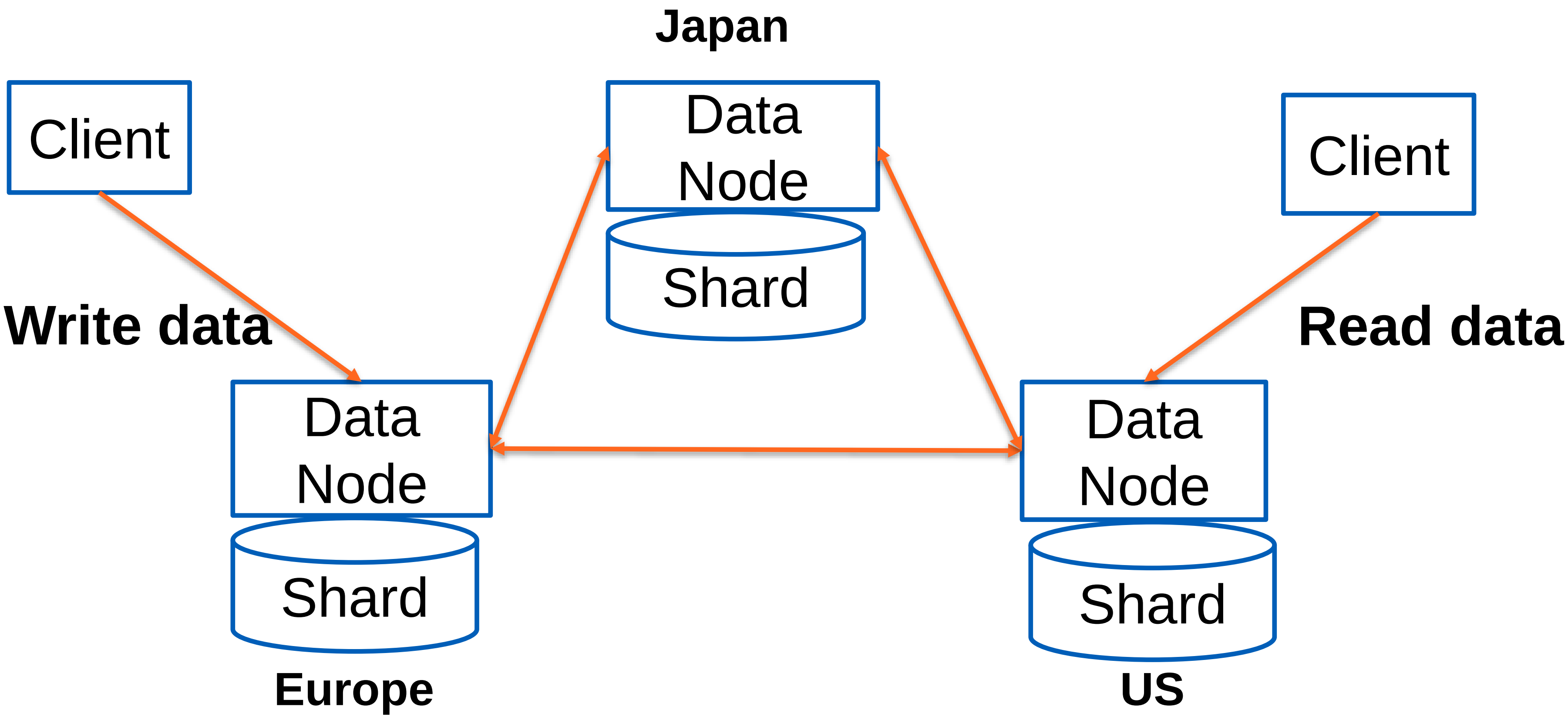
Consistency, Availability and Partition Tolerance

Views on data in big data platforms: node, shard/partition and replication





Distribution, Replication & Concurrency

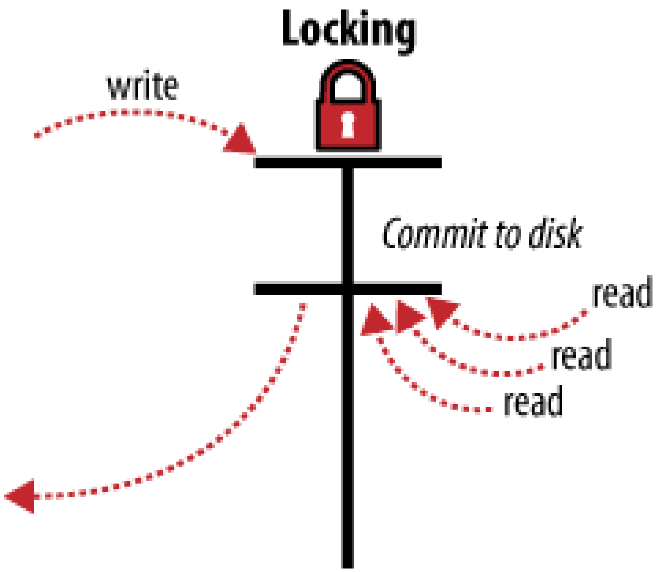


Well-known ACID properties for transactional systems

- **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

Examples of ACID Implementation

Locking, multi-version concurrency control (MVCC), two-phase committed protocols, etc.



MVCC

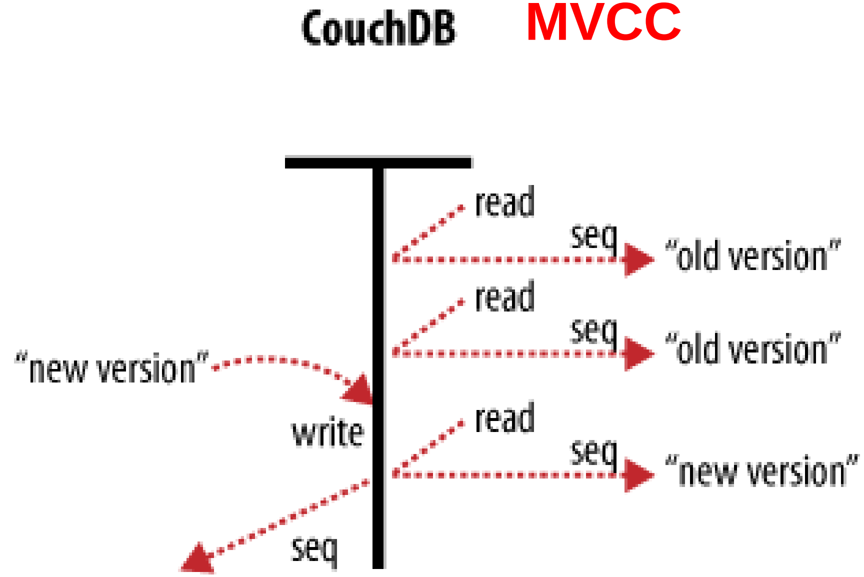


Figure source: https://docs.couchdb.org/en/stable/intro/consistency.html

Key issues in big data management

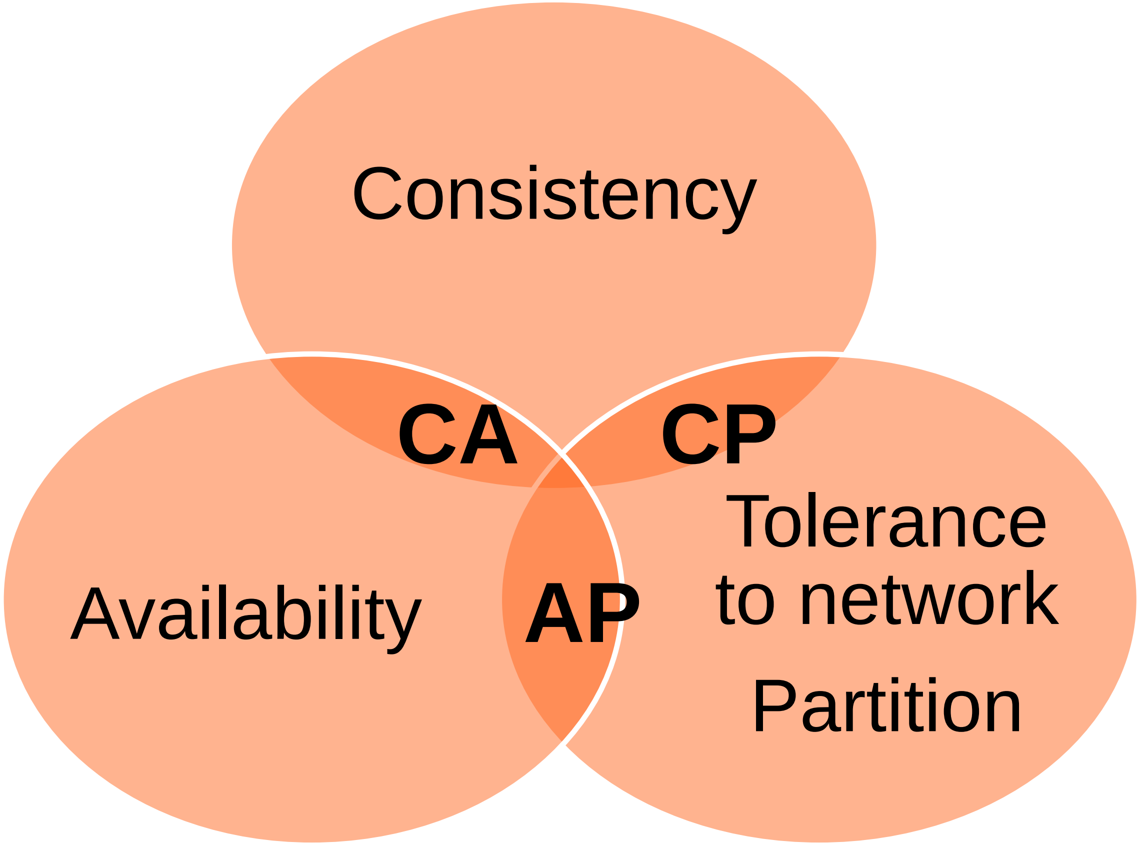
- Can every client see the same data when accessing any node in the platform?
- Can any request always receive a response?
- Can the platform serve clients under network failures?

Key issues in big data management

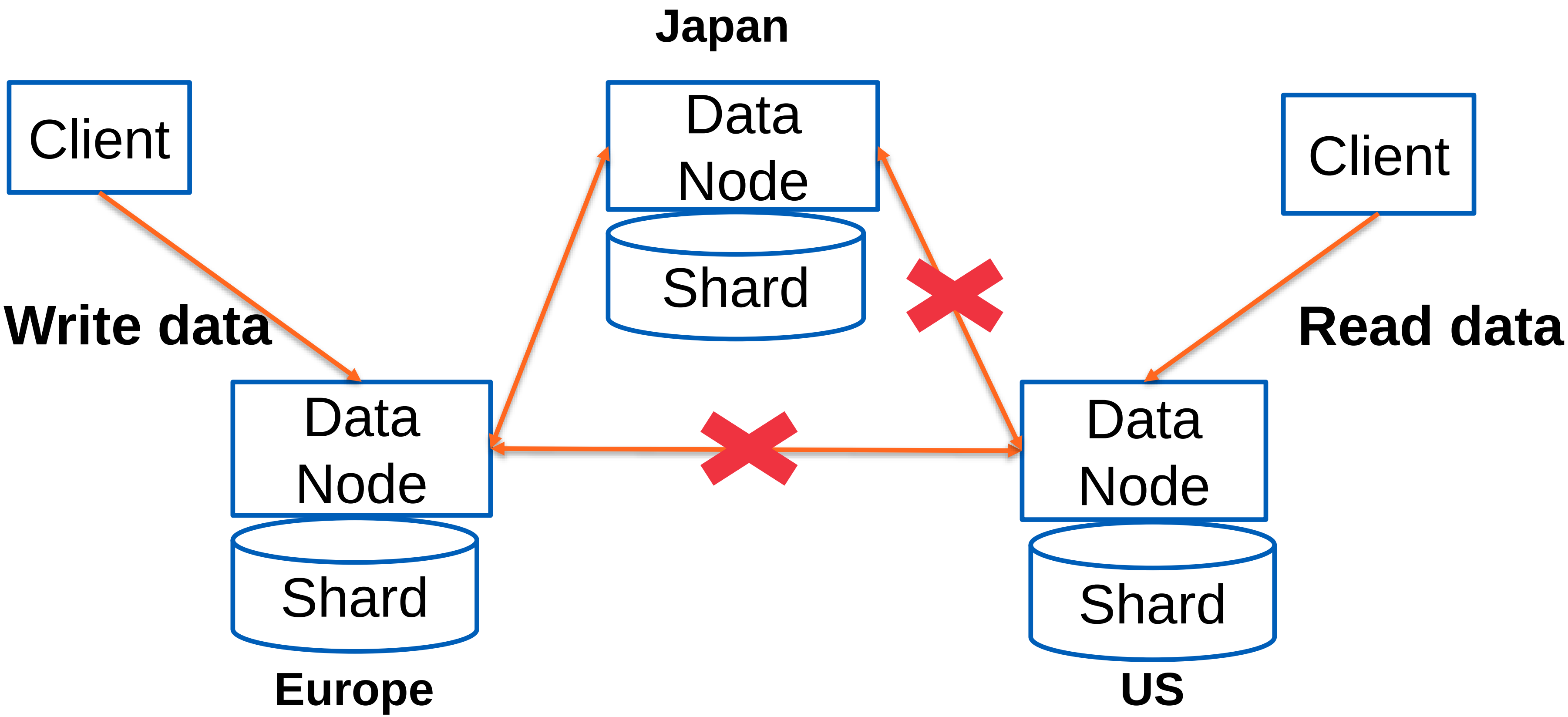
- **Tolerance to Network Partition**
  - if any node fails, the system is still working → a very strong constraint in our big data system design
- **High Consistency**
  - every read from a client must get the most up-to-date result
  - if the network fails, the newest write might not be updated to all 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"



Think about CAP with this simple model



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

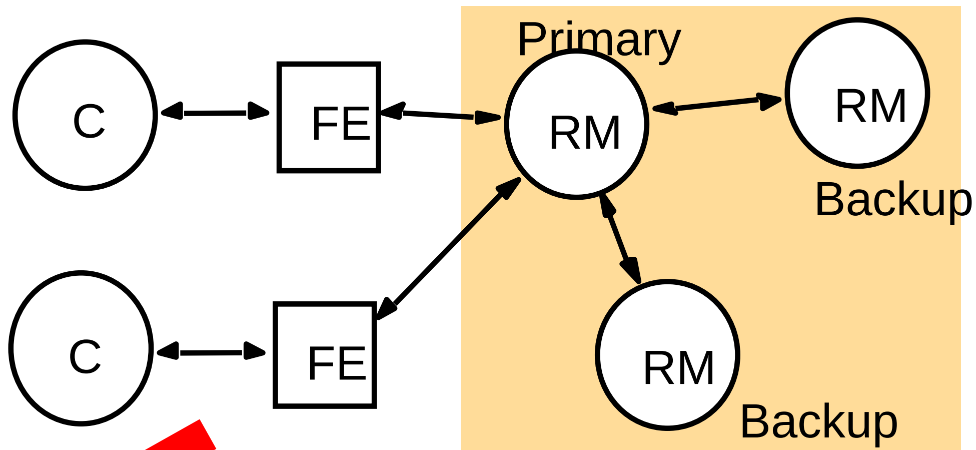
Programming consistency levels

- **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

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

Understanding different levels of consistency in different systems

- **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 Models

Understanding developer concerns

- **Identifying data models**
  - We 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* you need to support?
- **Identifying data management technologies**
  - Based on "multi-dimensional service properties" a technology for data management is selected
  - How would you design & provide your data management solutions?

Data models versus data management systems

- **Data models**
  - how we model and organize data elements of big data
- **Data management systems**
  - which techniques are used to manage big data
- **The combination of both is very important for big data platforms**

Common data models

- File
- Relational data model
- Key-Value data model
- Document-oriented model
- Column family model
- Graph model

Some important aspects when designing 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

Blob data

Big files:

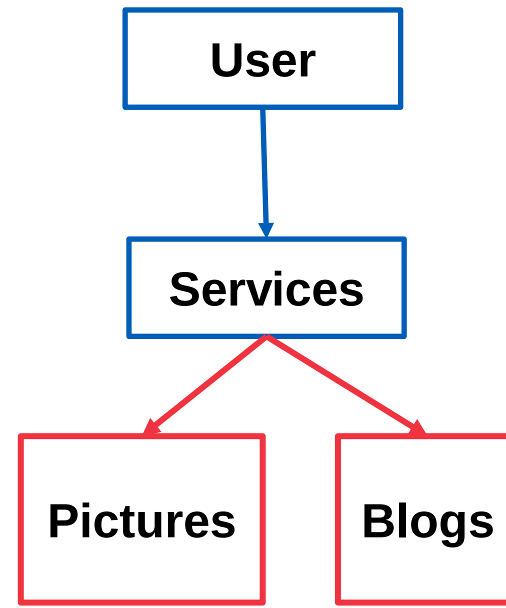
- Pictures, documents, big log files, images, video, backup data

Storage

- File systems or blob storage

Implementations

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



Relational Model

- **Well-known, long history**
- **Tables with rows and columns**
  - Strict schema requirements
- **Powerful querying & strong consistency support**
  - E.g.: Oracle Database, MySQL Server, PostgreSQL, MariaDB

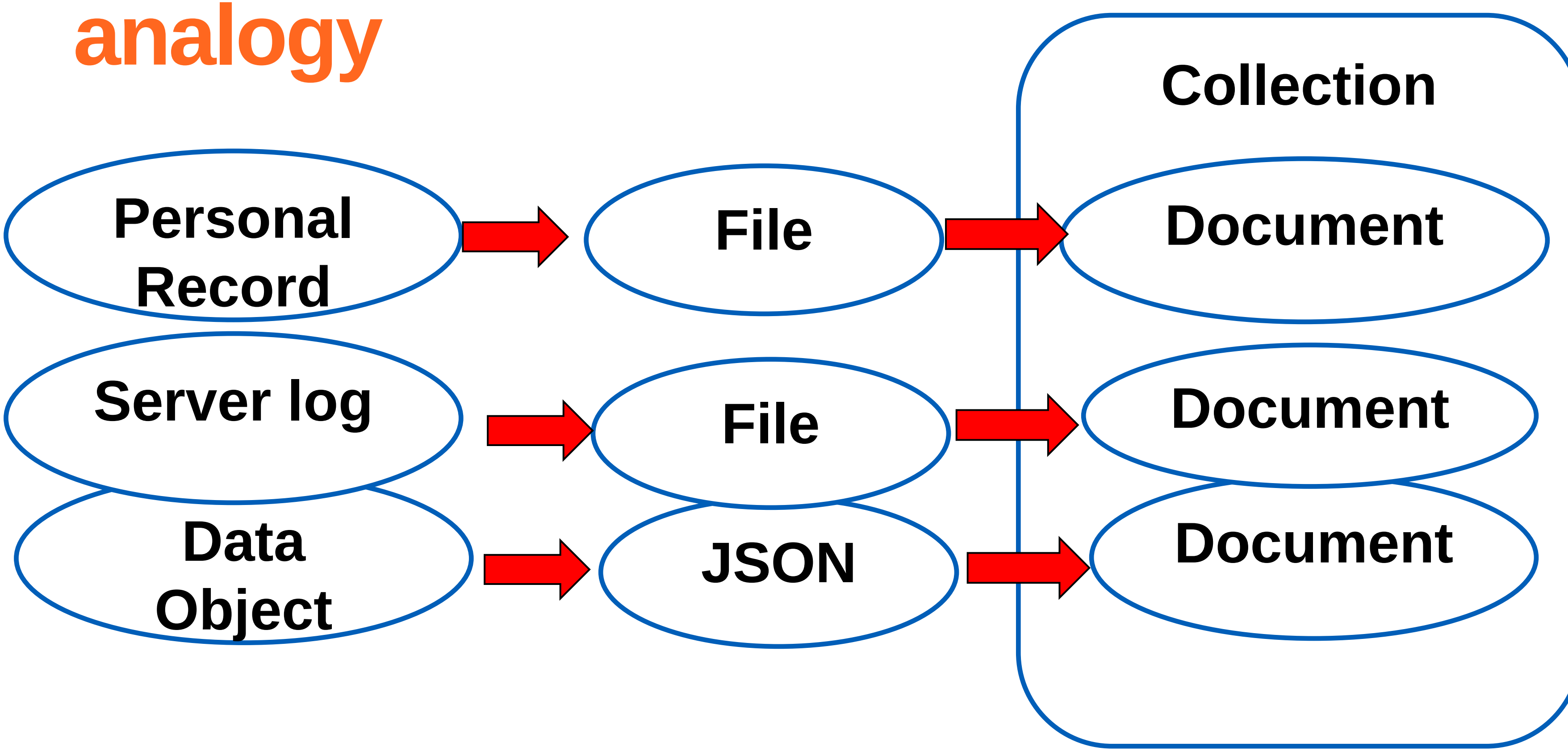
Relational databases for big data scenarios

- **Relational database at very large-scale**
  - Amazon Aurora, Microsoft Azure SQL Data Warehouse
- **We said ACID is hard with big data**
  - relational big database must address replication, distribution, and scalability issues
- **Examples of Amazon Aurora (reading list)**
  - based on MySQL/InnoDB but change the architecture, separate storage from engine, support cloud scale and replication, etc.

Key-Value Model

- **Tuple = (key, value)**
  - Values can be base on different structures
- **Scalable and performance**
- **Primary use case: caching (pages, sessions, frequently access data, distributed lock)**
  - Simple, very efficient but limited querying capabilities
- **Implementation:**
  - Memcached, Riak, Redis

Document-oriented model – simple analogy



Document-oriented Model

- **Documents**
  - flexible schema (schemaless) with flexible content
  - data fields can be complex to describe sub-documents
  - use collections, each collection is a set of documents
- **Primary use cases**
  - large amounts of semi-structured data
  - collection of data with different structures
- **Examples: MongoDB, CouchDB**

Graph-oriented model

- **Data is represented as a graph**
  - nodes or vertices represent objects, an edge describing a relationship between nodes
  - properties associated with nodes and edge provide other information
- **Use cases**
  - when searching data is mainly based on relations (social networks, asset relationship, knowledge graph)
- **Examples:**
  - Azure CosmosDB, ArgangoDB, Titan, Grakn.AI, Neo4J, OrientDB

Column-family data model

scalable, distributed storage for multi-dimensional sparse sorted map data

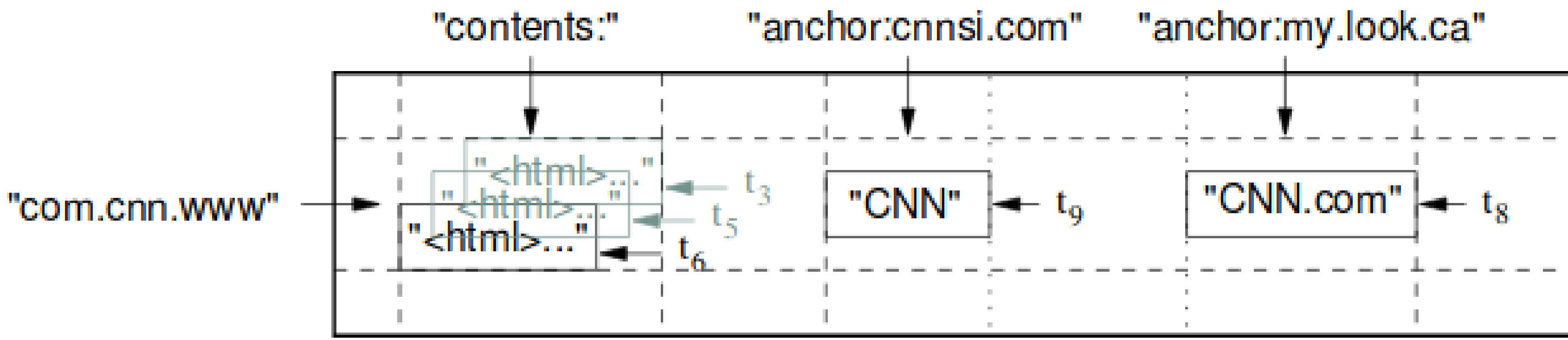


Figure source: Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2006. Bigtable: a distributed storage system for structured data. In Proceedings of the 7th symposium on Operating systems design and implementation (OSDI '06). USENIX Association, Berkeley, CA, USA, 205-218.

Column-family data model

- **Data Model**
  - Table consists of rows
  - Row consists of a key and one or more columns
  - Columns (column name, value, timestamp)
  - Columns are grouped into column families
  - Columns can be different in rows
- **Examples: Cassandra, HBase**

Time Series Database

- **So many types of data in big data are time series**
  - IoT measurements, session data, log, etc.
- **Document/relational models can be used**
  - e.g., Cassandra, ElasticSearch, BigTable
- **Time Series Databases specially designed for time series data**
  - *examples: Riak TS (Time Series), InfluxDB, Apache Druid*

In-memory databases

- **Databases use machine memory for storage**
  - Persist data on disks
  - Require very powerful machines
- **In principle it is not just about data models but also data management, data processing, software and hardware optimization, e.g.,**
  - SAP HANA, VoltDB: in memory relational databases
- **Why are in-memory databases important?**

Polyglot persistence and metadata

Issues

- **State of data**
  - Raw, extracted, high-level info
- **its purposes (data usage and requirements)**
  - Archival, processing, sensitive, test and develops
- **Need Store technologies**
  - Different format, model, access, control, ..
  - Poly, multi purposes

Multiple types of data available across systems

- **Real-world applications need different types of databases!**
  - It is easy to use a single type of database, but it might not work for real projects
- **Strong set of APIs, connectors and client libraries**
  - for providing data to different analytics frameworks

Examples

- **Case 1: Monitoring/maintenance situations**
  - Relational/document models would be good for describing subjects to be monitored (e.g., equipment, house, animal)
  - But time series/column-family models would be good for storing monitoring data (e.g., sensor data, feedback, …)
- **Case 2: financial management/fintech/e-commerce**
  - Relational model could be good for customer records and payment
  - But document/column-family models would be good for product description, activity logs, or transactions records

Polyglot Big Data models/systems

- **A platform might need to provide multiple supports for different types of data**
  - 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,

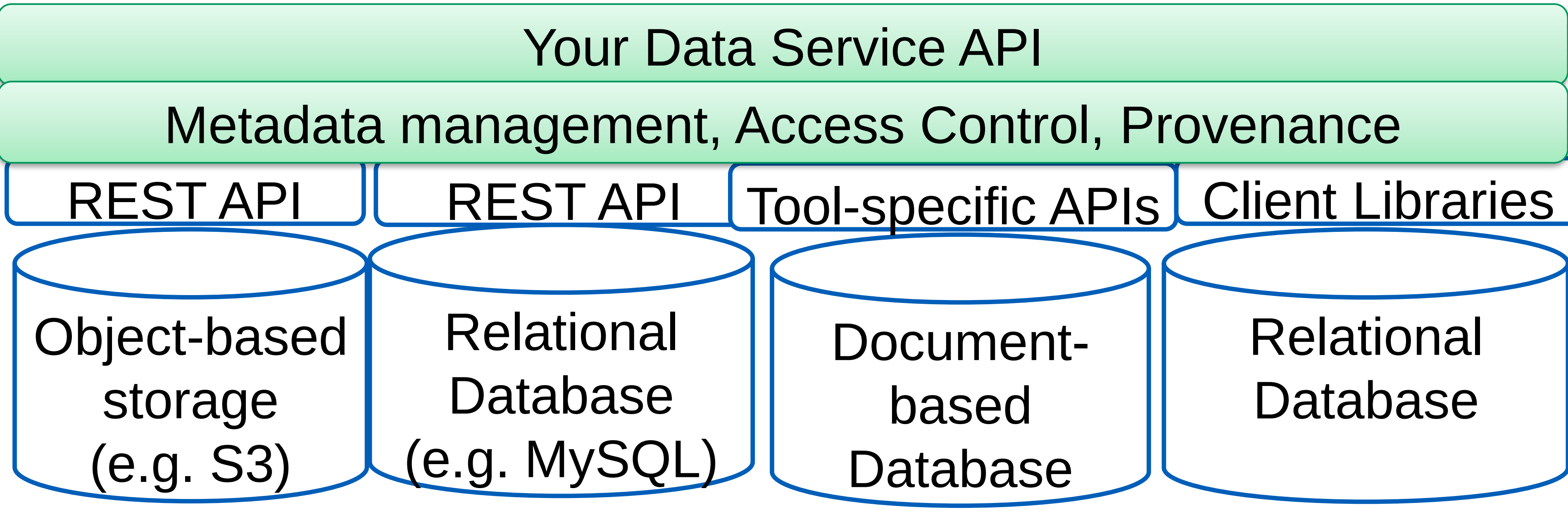
Polyglot persistence is inevitable for many use cases

Design choices

- **Using different databases/storages**
  - different types of data must be linked
    - *Each type requires a different model*
  - provide a collection of APIs
- **Multi-model database services**
  - a data service can host different data models
  - can be a virtual service atop other database services

Multi databases/services

Data access APIs can be built based on well-defined interfaces



Large-scale multi-model database services

- **Able to store different types of data models**
  - Relational tables, documents, graphs, etc.
- **Benefits**
  - the same system (query, storage engine)
- **Example**
  - Microsoft Azure Cosmos, OrientDB, ArangoDB, Virtuoso

Metadata about data resources

- **Metadata characterizes data assets (stored in databases/datasets)**
  - 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!**
  - Ingestion, collection and management
- **Tools:** Google Data Catalog, Apache Atlas, Linkedin DataHub

Example of Metadata

Key design:

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

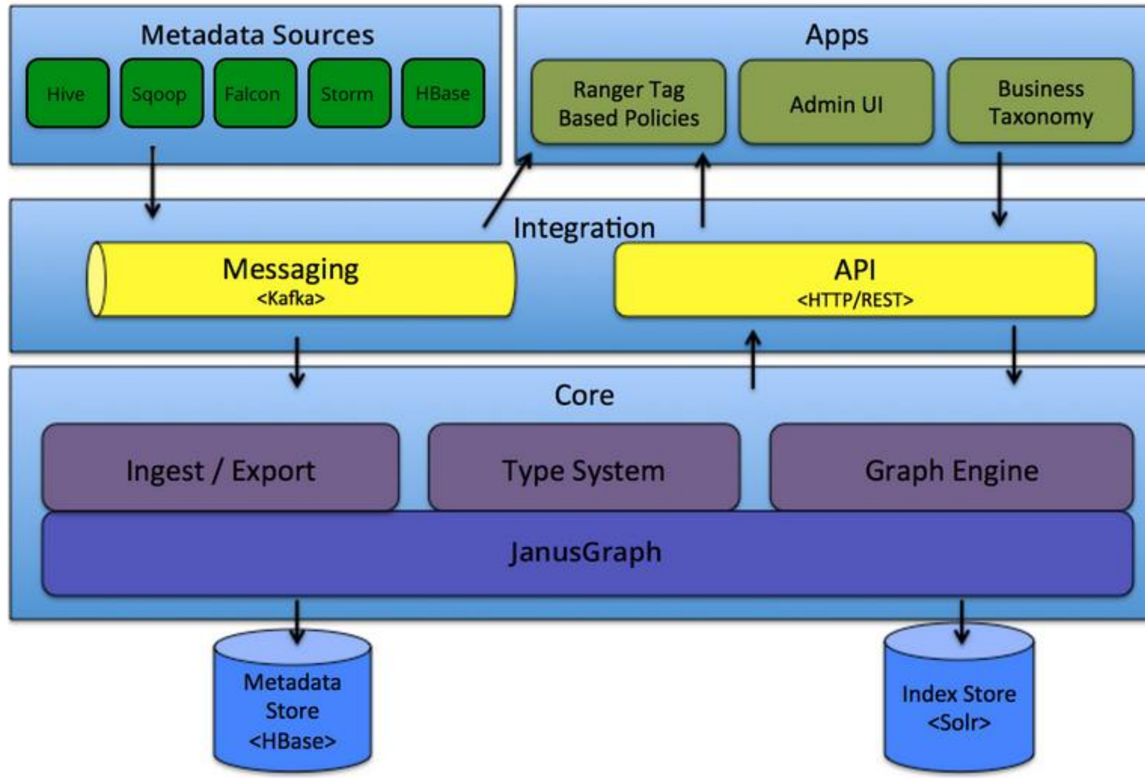


Figure source: https://atlas.apache.org/#/Architecture

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io