Aalto University
School of Science

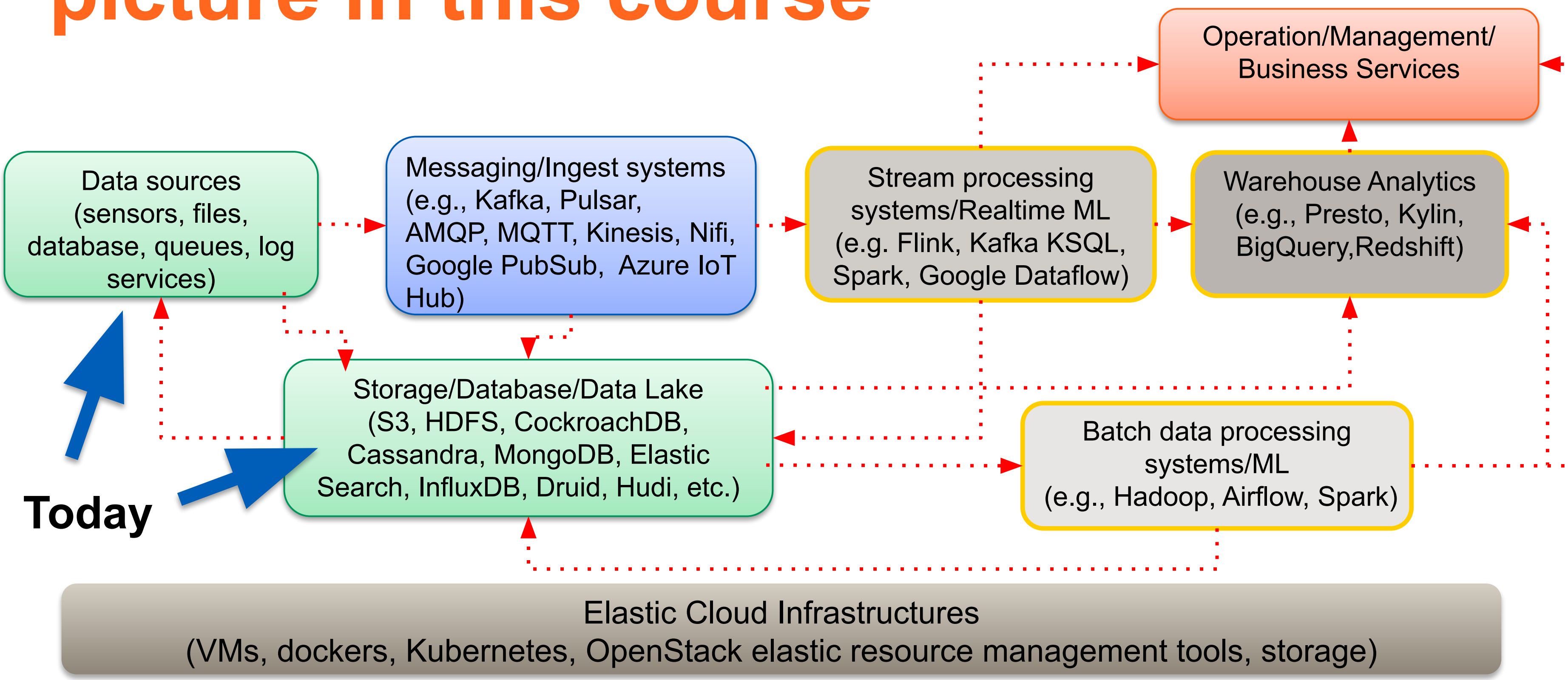
Big Data Storage and Database Services

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

Learning objectives

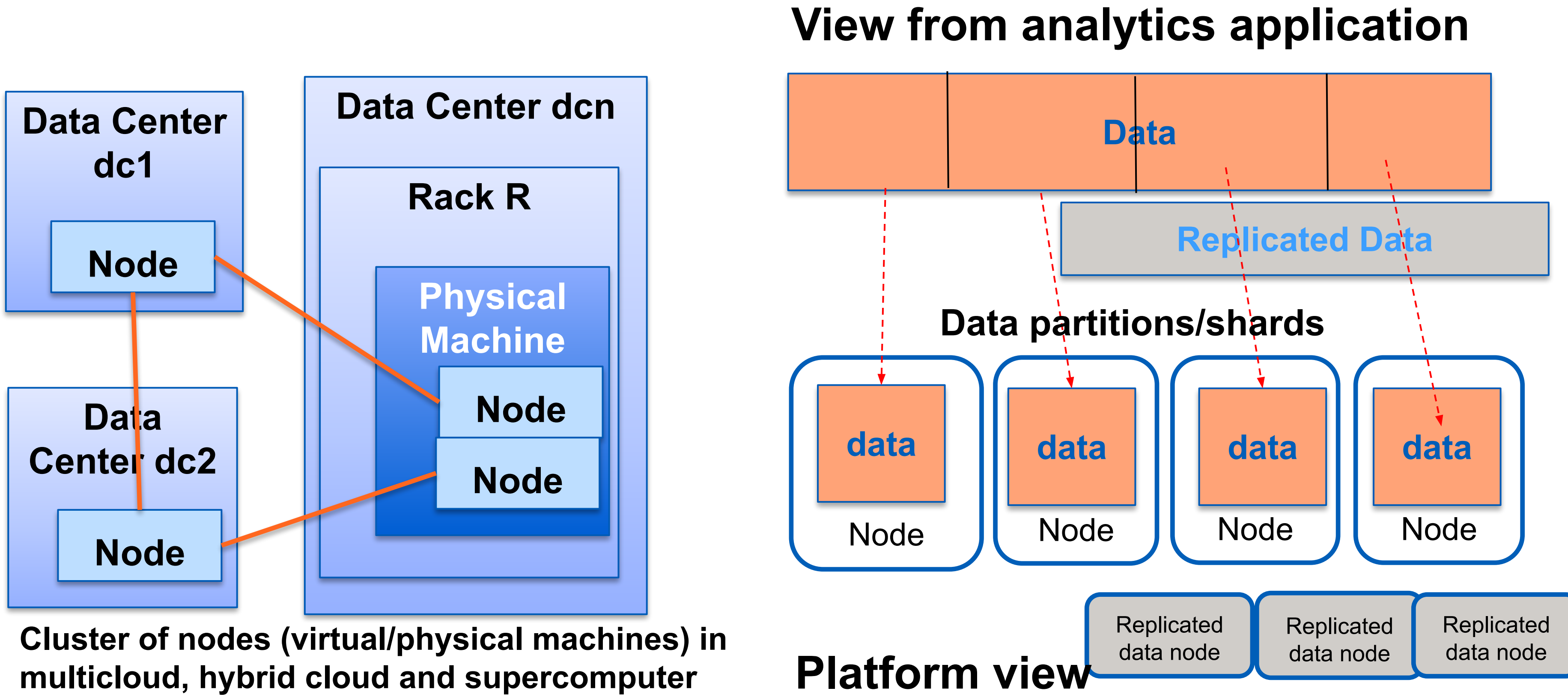
- Understand consistency, availability and partition tolerance issues in design and programming
- Study common data models and data management
- Understand the need of polyglot persistence and metadata management

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



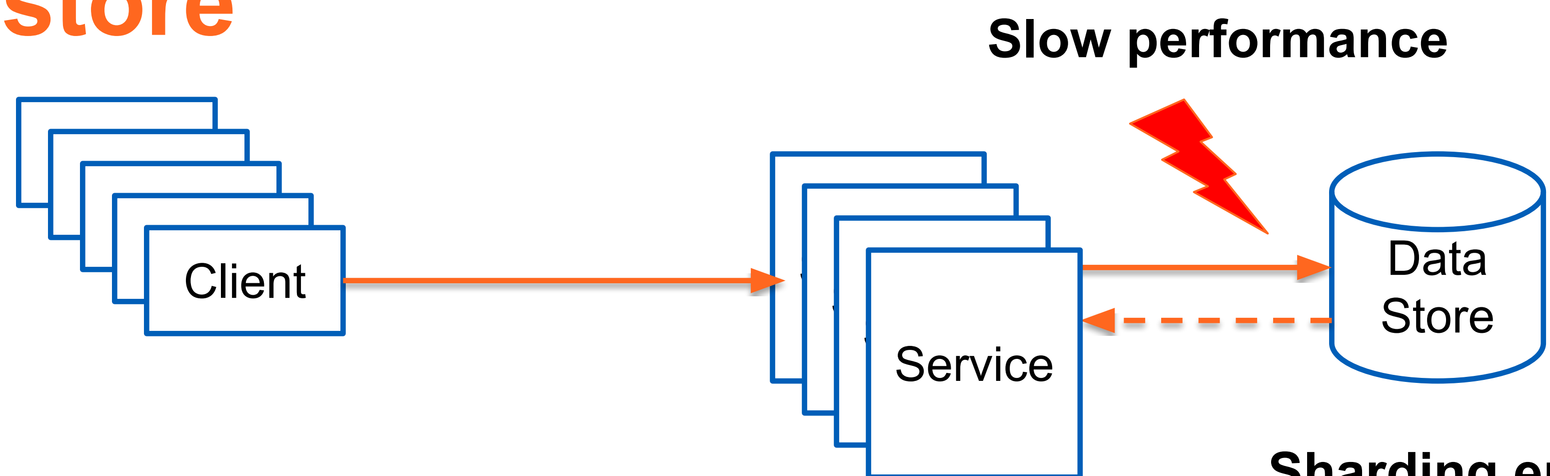
Consistency, Availability and Partition Tolerance

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

Performance problems between service serving request and data store

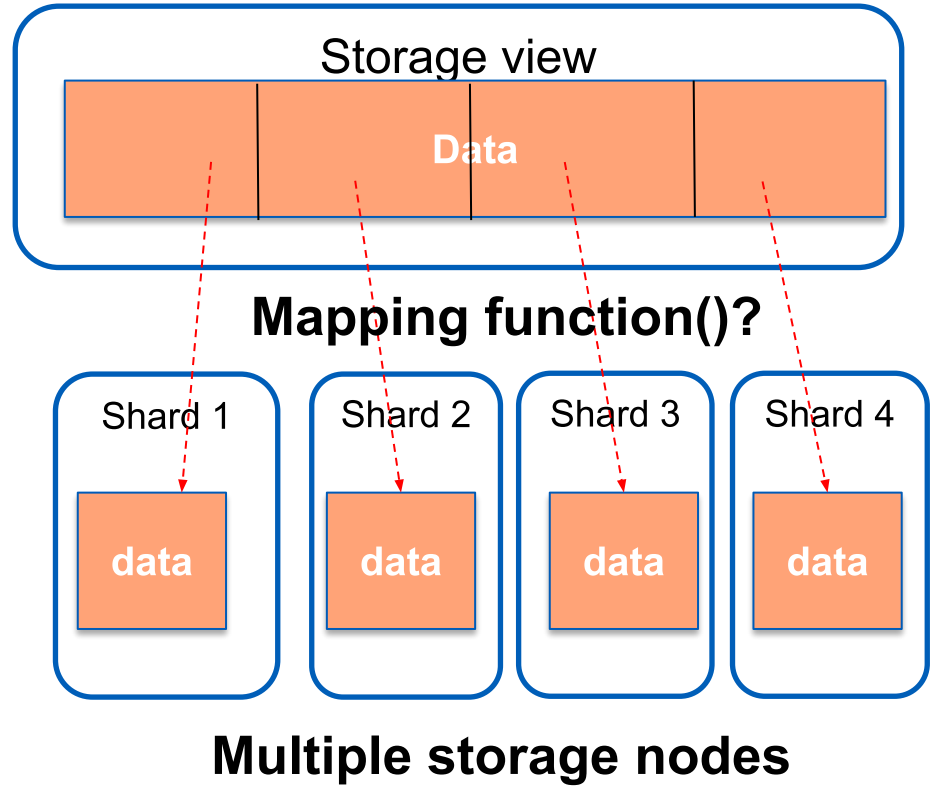


Sharding enables parallel access (data parallelism)

- 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!



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

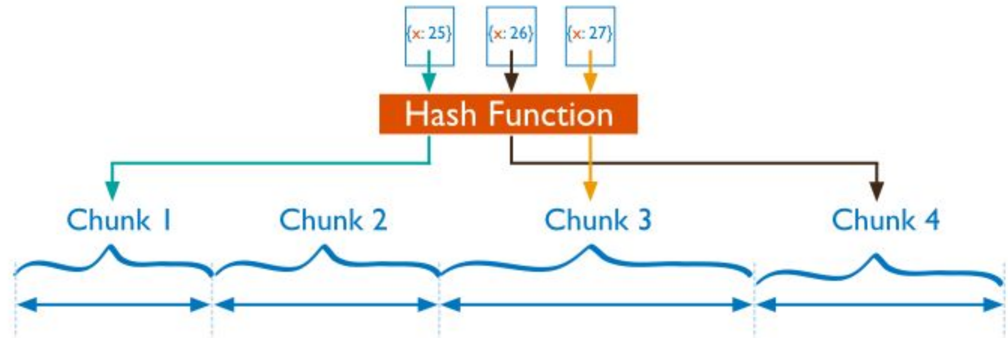
Different, common strategies

- 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

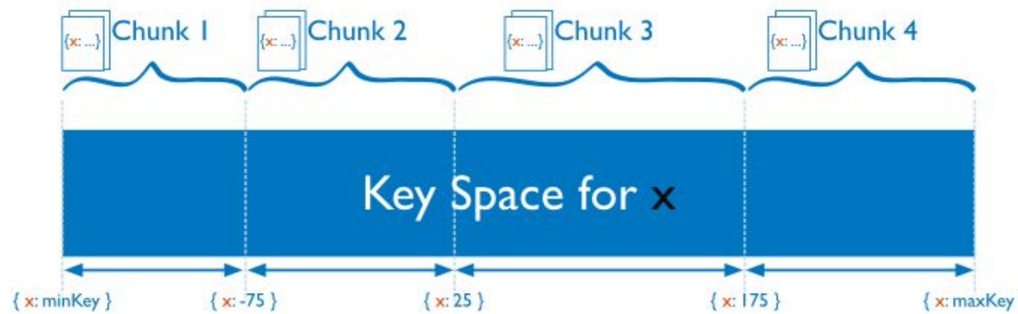
Sharding patterns/strategies reading: https://msdn.microsoft.com/en-us/library/dn589797.aspx

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)
…;
```

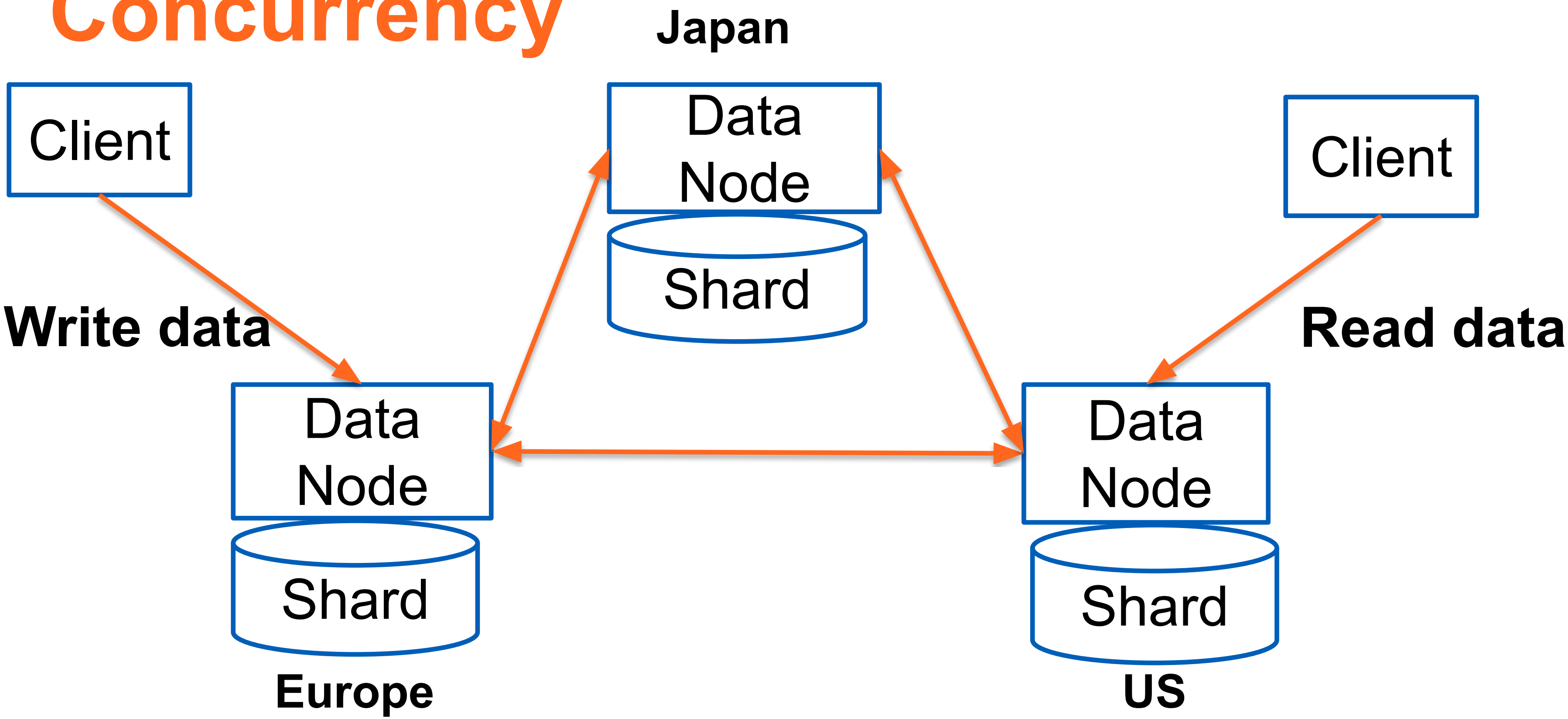
Indicate partition info

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
```

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

Examples of ACID Implementation

Locking, multi-version concurrency control (MVCC), two-phase commit protocol (2PC), etc.

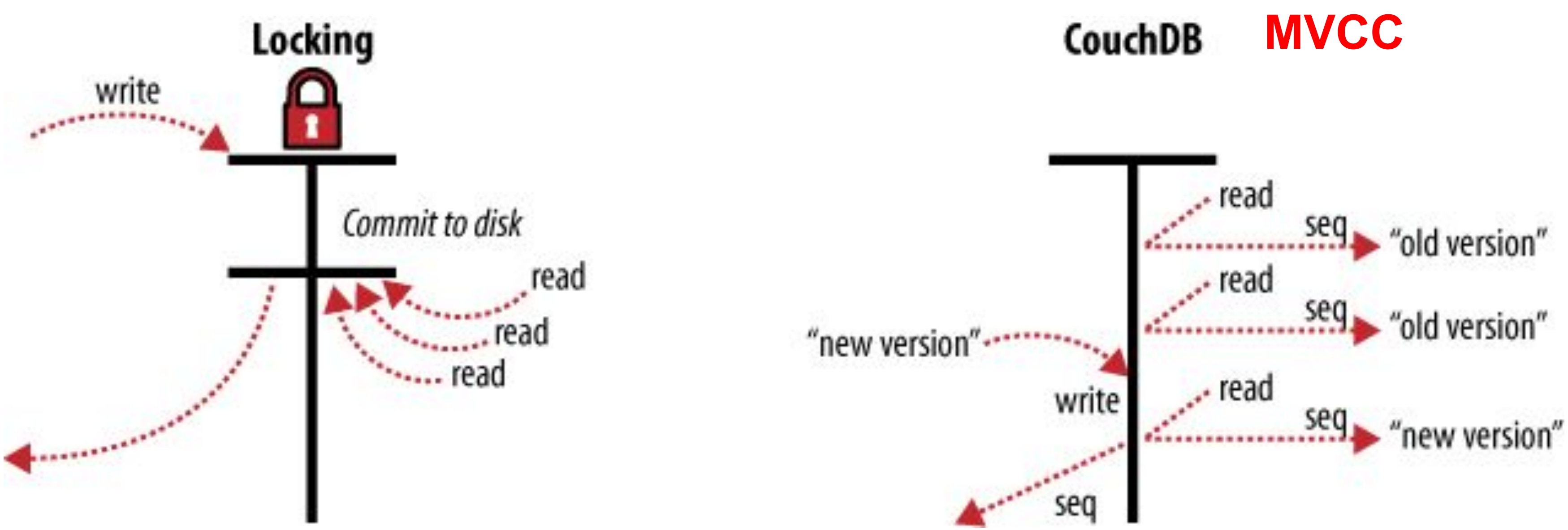


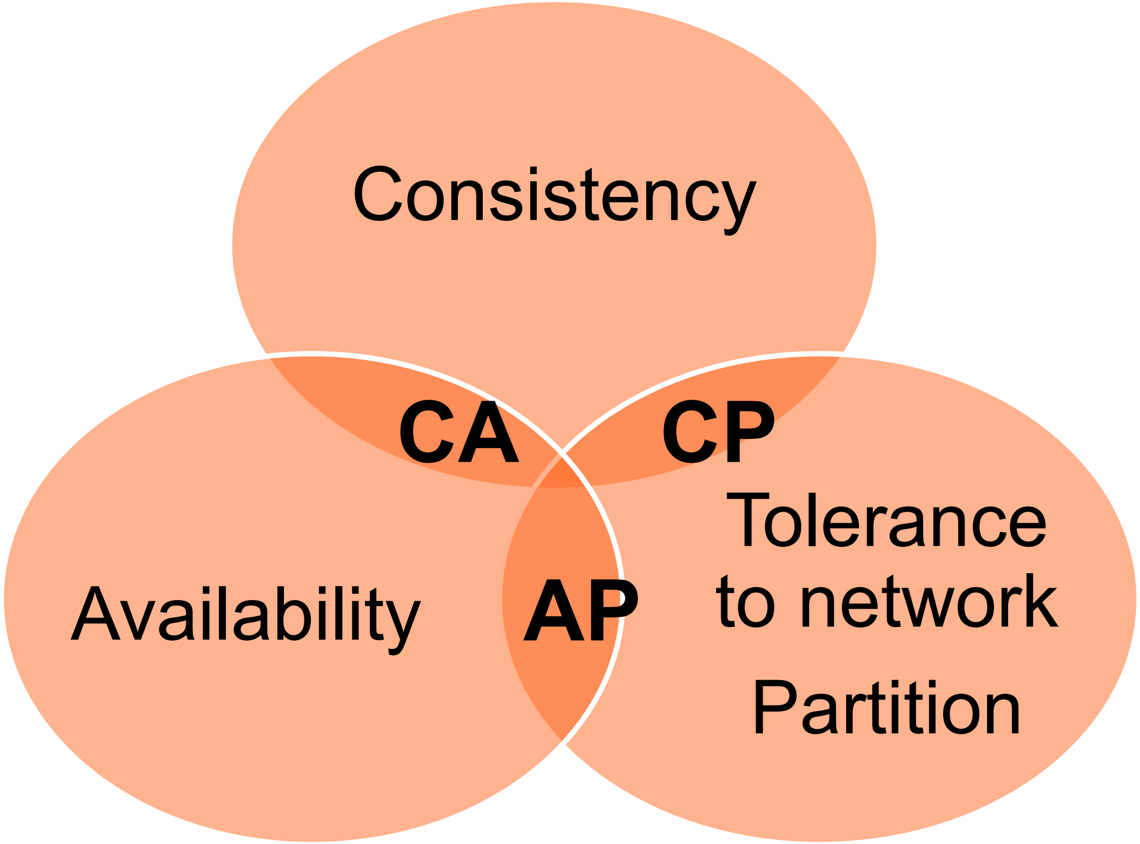
Figure source: https://docs.couchdb.org/en/stable/intro/consistency.html

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)

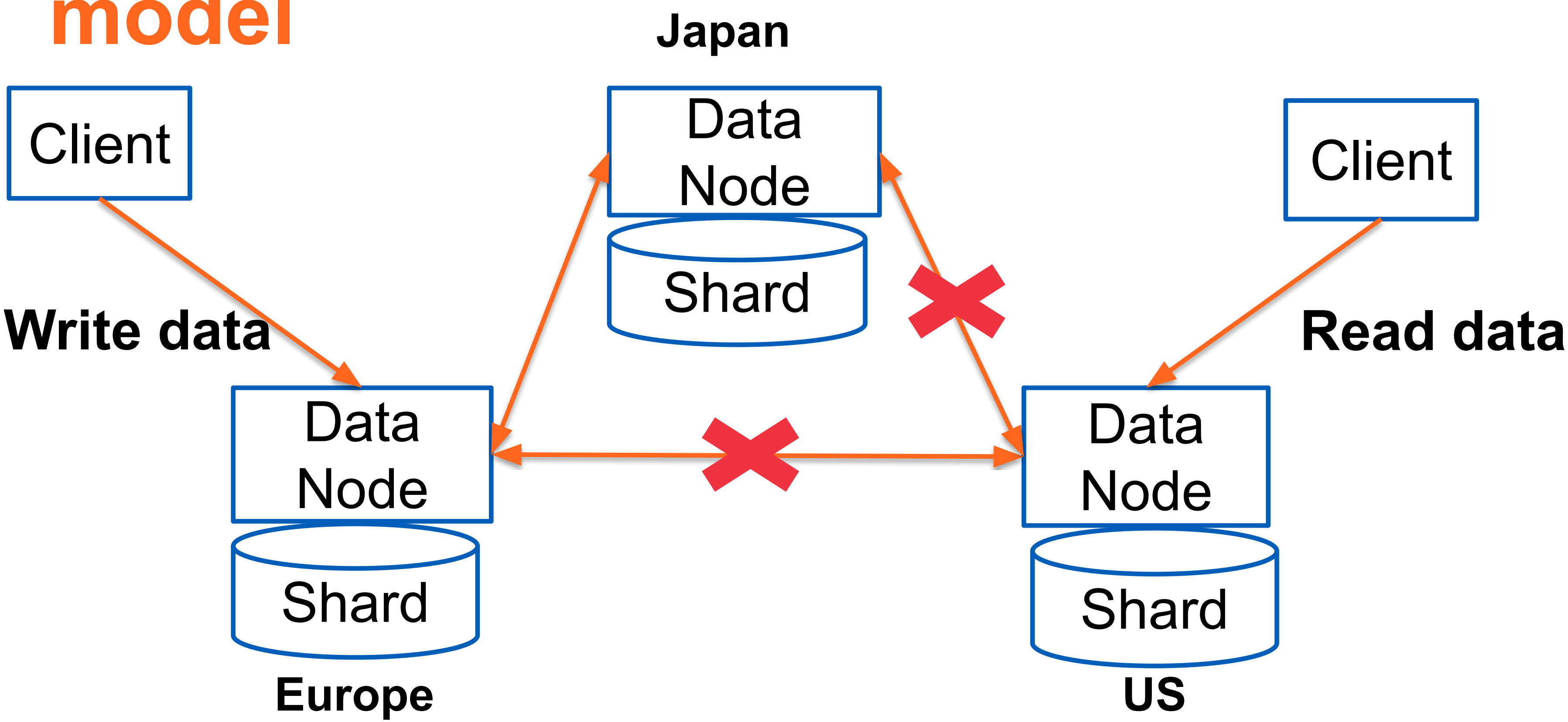
CAP Theorem

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

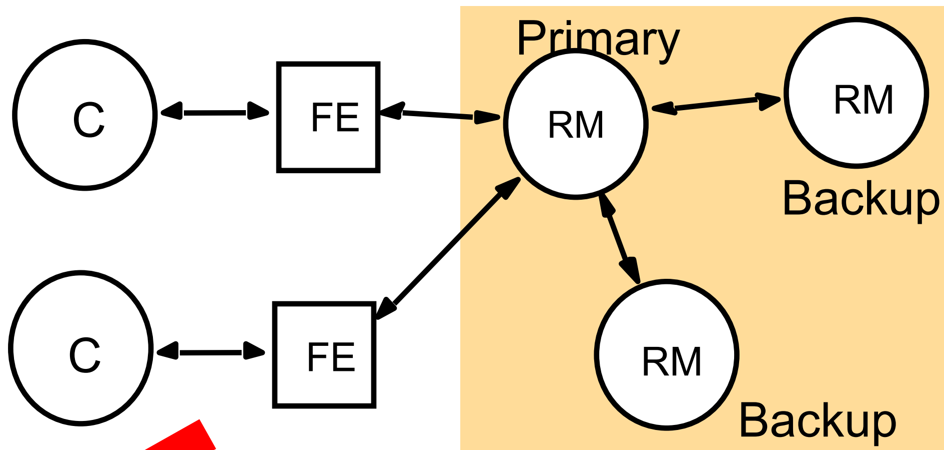
BASE (Basically Available, Soft state, Eventual consistency)

- **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 $\Rightarrow$ 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 Models

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**

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 vs data access technologies

- **Data models explain structure and organization of the data to be analyzed**
  - Very import for deciding technologies and techniques used for data analytics
  - How *many data models* you need to deal with?
    - *Complex analytics might require use to deal with different models*
- **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
- **Some are also seen in "no big data"**
- **Some are *specifically designed to address big data***

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

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

Big data: blob (binary large object)/tabular, text files

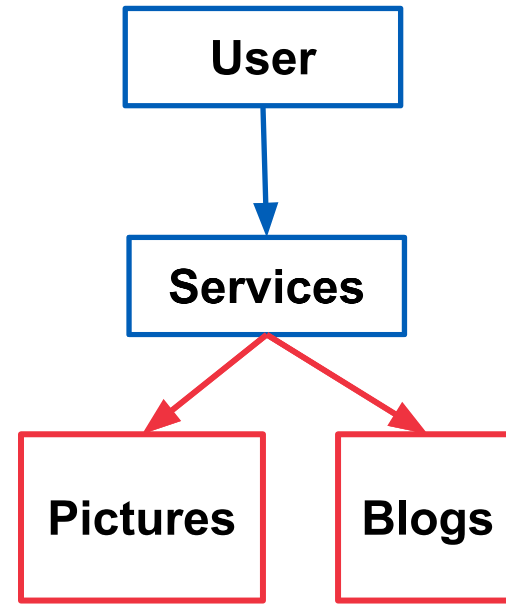
- Pictures, documents, big log files, images, video, backup data

Storage

- Distributed file systems or blob/object storage

Implementations

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



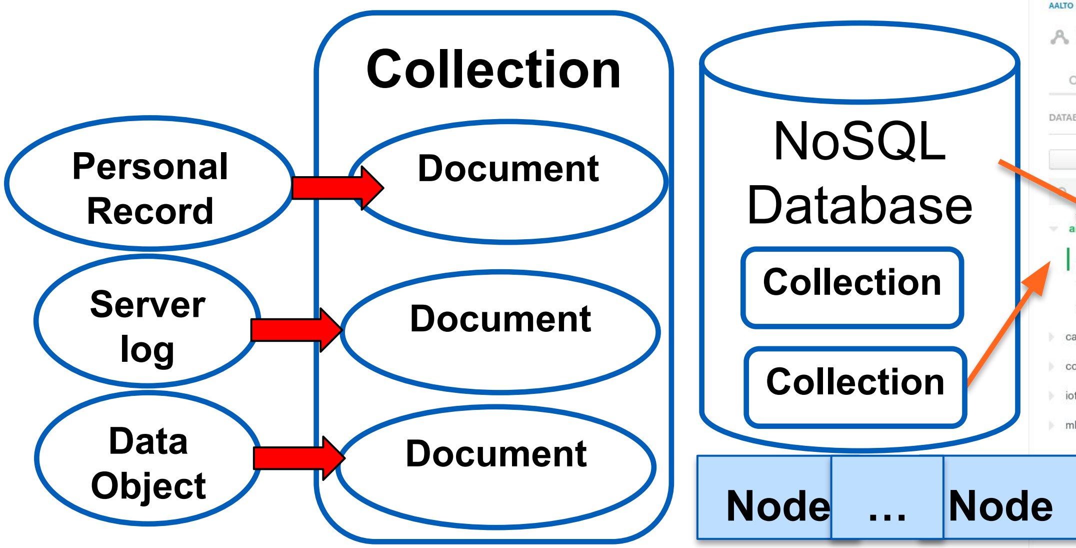
Big data: relational databases

- **Tables with rows and columns**
  - Strict schema requirements, powerful querying & strong consistency support
  - E.g.: Oracle Database, MySQL Server, PostgreSQL, CockroachDB
- **Relational database at very large-scale**
  - Amazon Aurora, Microsoft Azure SQL Data Warehouse
- **ACID (atomicity, consistency, isolation, durability) is hard with big data**
  - relational big database must address replication, distribution, and scalability issues

Key-Value Model

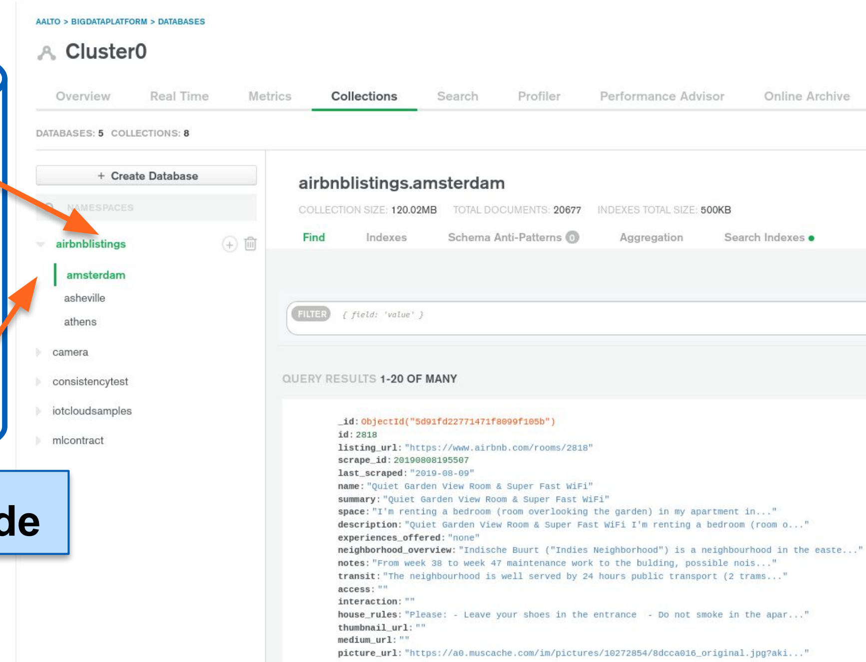
- **Tuple = (key, value)**
  - Values can be base on different structures
- **Scalable and performance**
- **Primary use case: caching (pages, sessions, frequently access data, distributed lock)**
  - Simple, very efficient but limited querying capabilities
- **Implementations:**
  - Memcached, Riak,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

Big data: document-oriented model



Flexible schema

(Cluster)



Many servers hosting database

Big data: column-family data model

Many situations we aggregate and scan few columns of million rows of data ⇒ store big data in columns enable fast scan/retrieval/aggregation

Colum Family = (Column, Column, …): for similar type of data

Column Key =Family: qualifier

Data = (Key, Value) where Key =(Row Key, Column Key, Timestamp)

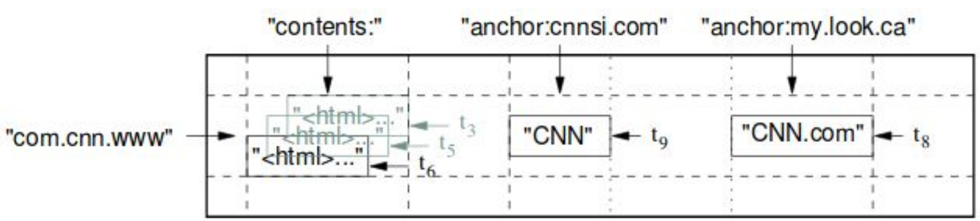


Figure source: Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2006. Bigtable: a distributed storage system for structured data. In Proceedings of the 7th symposium on Operating systems design and implementation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

Examples: Cassandra, HBase

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, TypeDB, Neo4J, OrientDB

Time Series Database

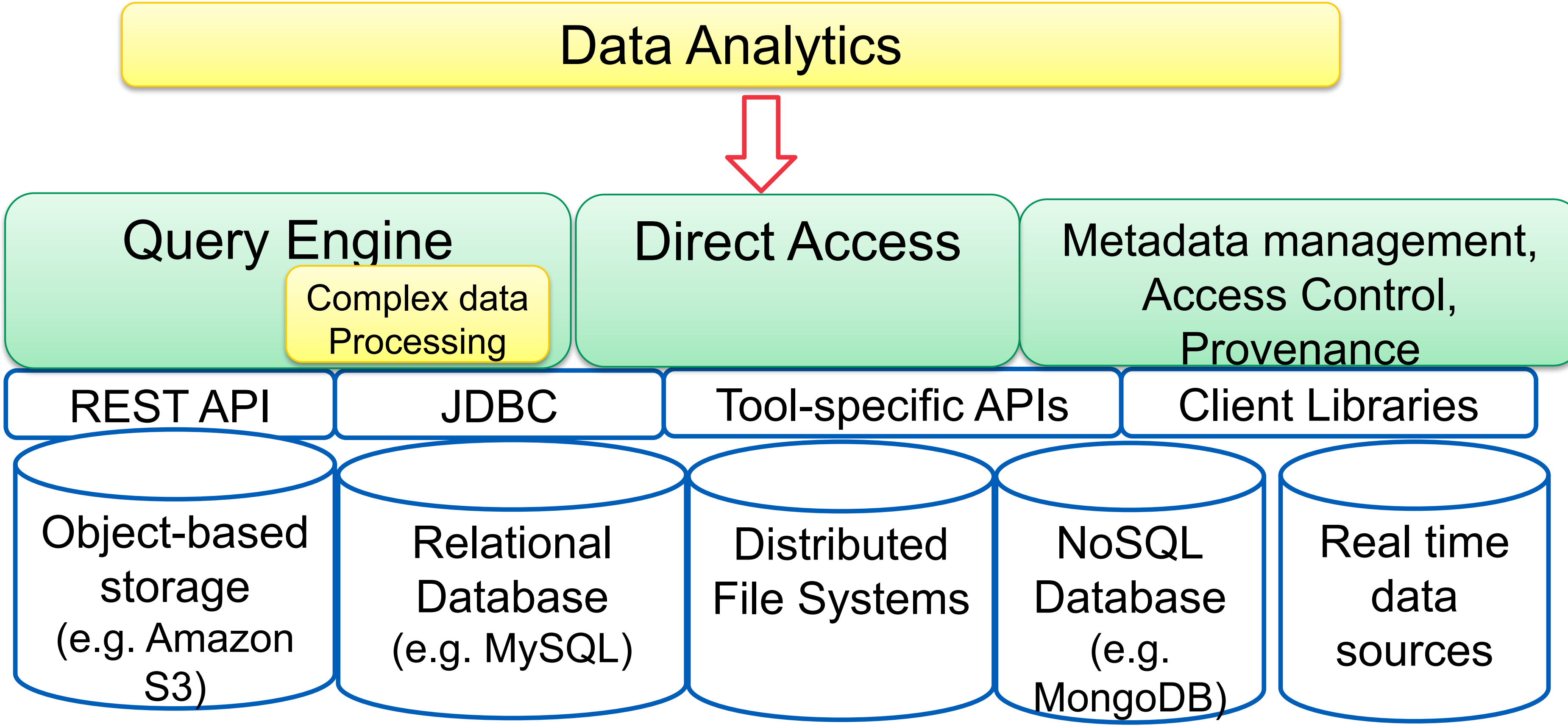
- **So many types of data in big data are time series**
  - IoT measurements, session data, log, etc.
- **Document/relational models can be used**
  - e.g., Cassandra, ElasticSearch, BigTable
- **Time Series Databases specially designed for time series data**
  - *examples: Riak TS (Time Series), InfluxDB,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

High-level analytics 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)


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

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

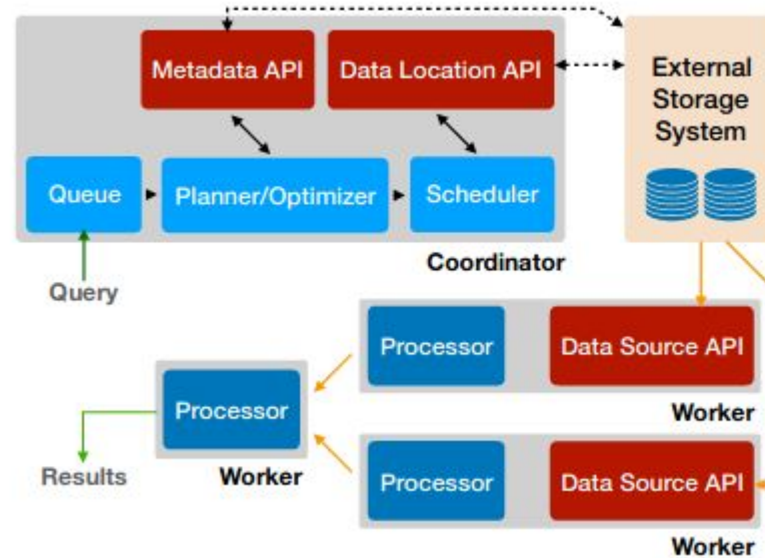


Figure source: *Presto: SQL on Everything https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8731547&tag=1*

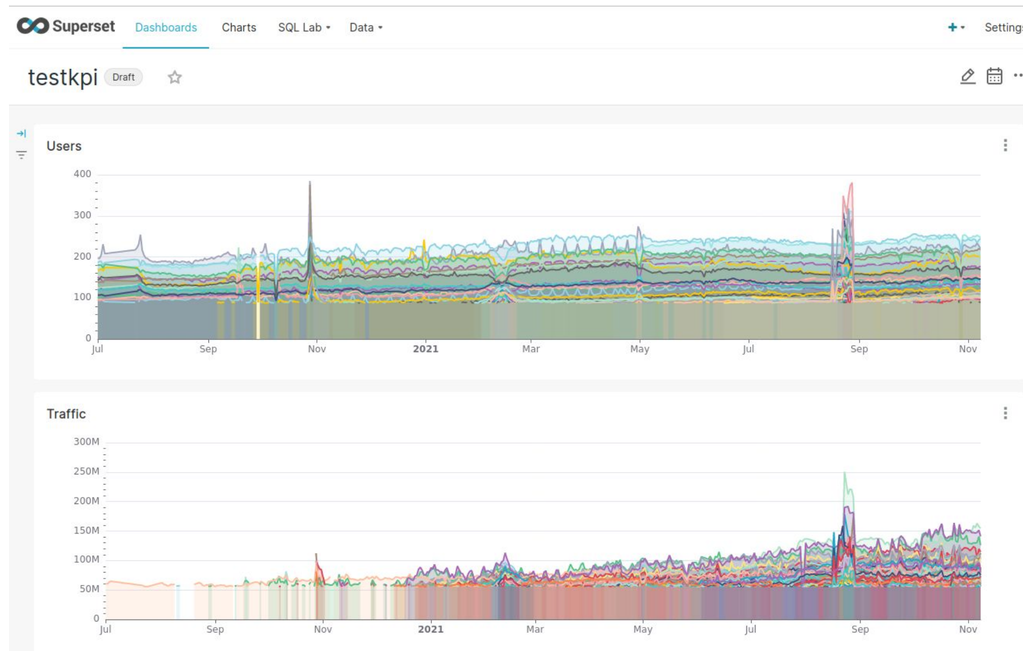
Examples

Analytics: write SQL

```
presto> explain (format text, type logical) SELECT sum(trip_total) FROM chicago_taxi_trips.taxi_trips;

--------------------------------------------------------------------------------------------------------
 - Output[_col0] => [sum:double]
         _col0 := sum (1:44)
     - Aggregate(FINAL) => [sum:double]
             sum := "presto.default.sum"((sum_4)) (1:44)
         - LocalExchange[SINGLE] () => [sum_4:double]
             - RemoteStreamingExchange[GATHER] => [sum_4:double]
                 - Aggregate(PARTIAL) => [sum_4:double]
                         sum_4 := "presto.default.sum"((trip_total)) (1:44)
                     - TableScan[TableHandle {connectorId='bigquery', connectorHandle='BigQueryTableHa
                             Estimates: {rows: ? (?), cpu: ?, memory: 0.00, network: 0.00}
                             LAYOUT: BigQueryTableHandle{projectId=bigquery-public-data, schemaName=ch
                             trip_total := BigQueryColumnHandle{name=trip_total, type=FLOAT, mode=NULL
```

Analytics: data exploration and visualization, e.g. with Apache Superset



Polyglot persistence and metadata

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

Examples

- **Case 1: monitoring/maintenance situations**
  - Subjects to be monitored (e.g., equipment, house, animal) are usually in relational/document databases with different updates/management
  - Their monitoring data are time series, update in real time (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

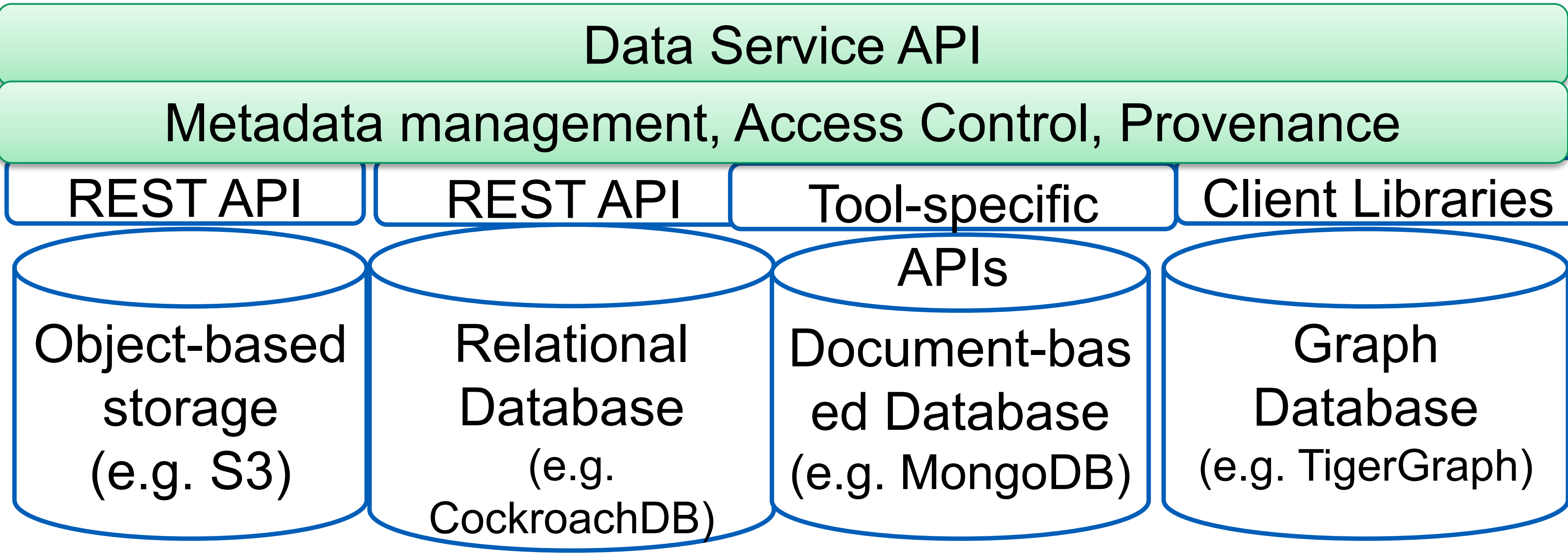
Polyglot persistence is inevitable for many use cases

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
- **Data Lake**

Multi databases/services

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



Large-scale multi-model database services

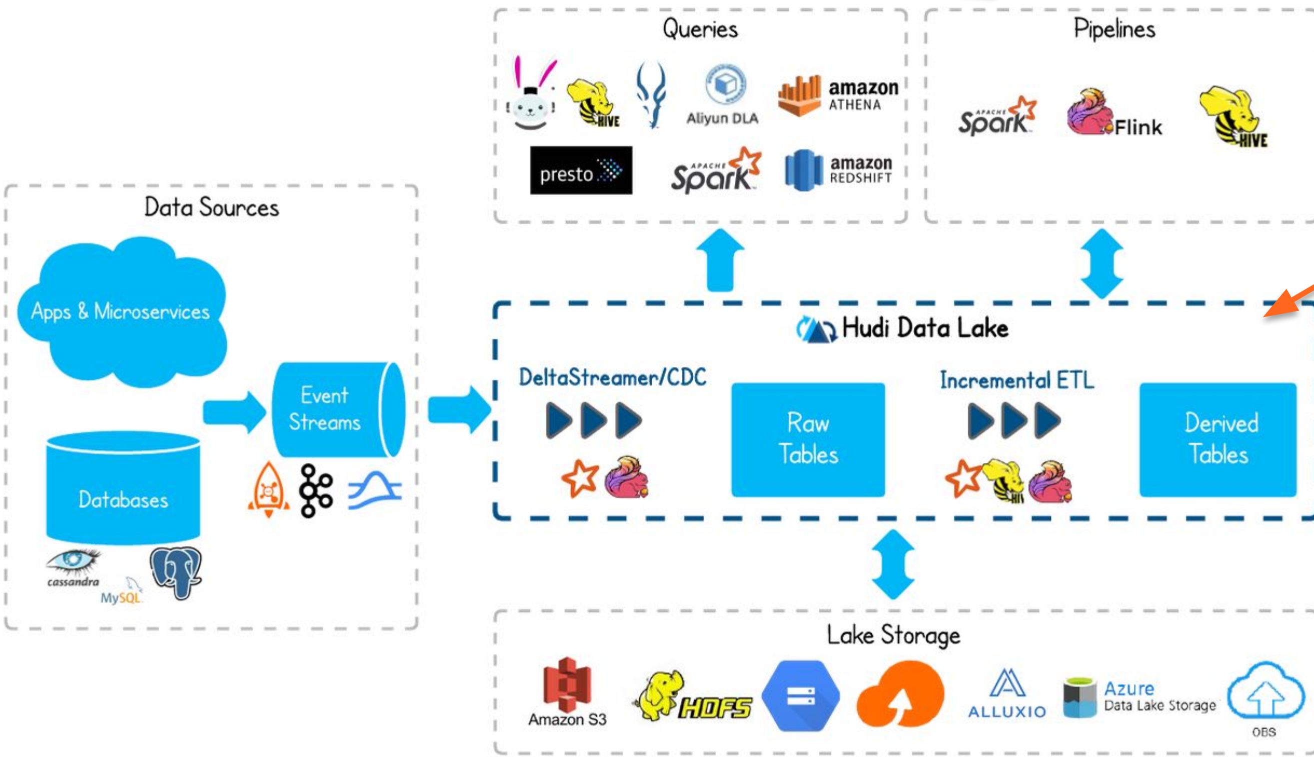
- **Able to store different types of data models**
  - Relational tables, documents, graphs, etc.
- **Benefits**
  - the same system (query, storage engine)
- **Example**
  - Microsoft Azure Cosmos, OrientDB, ArangoDB, Virtuoso

Data Lake

- **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
- **Related concepts**
  - Data mesh (data, infrastructures, services and governance for domain-oriented data products)

Data Lake: example

Allows different analytics



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

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

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

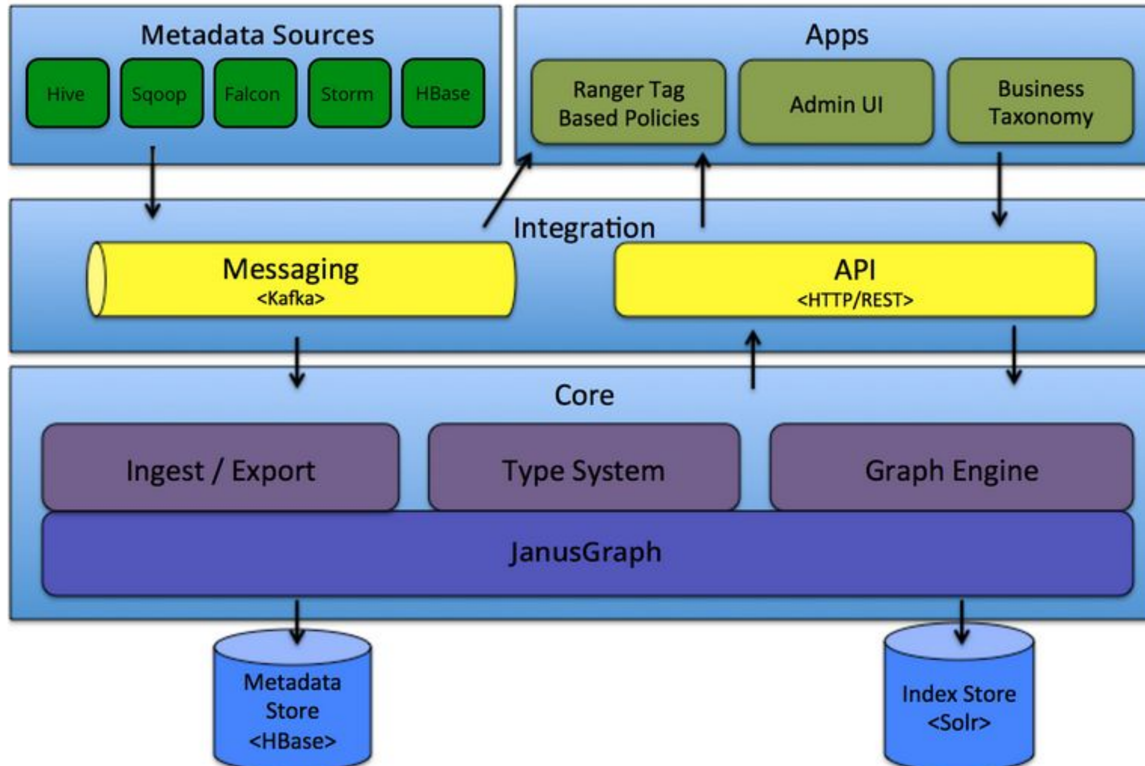


Figure source: https://atlas.apache.org/#/Architecture

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 deployment**
  - deployment (multiple nodes, data centers, geo-distributed locations), sharding, and replication
- **Focus on understanding features for programming consistency**
- **Practice with some key database/store technologies**
  - Individual, federated, multi-model, and lake

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io