Aalto University
School of Science

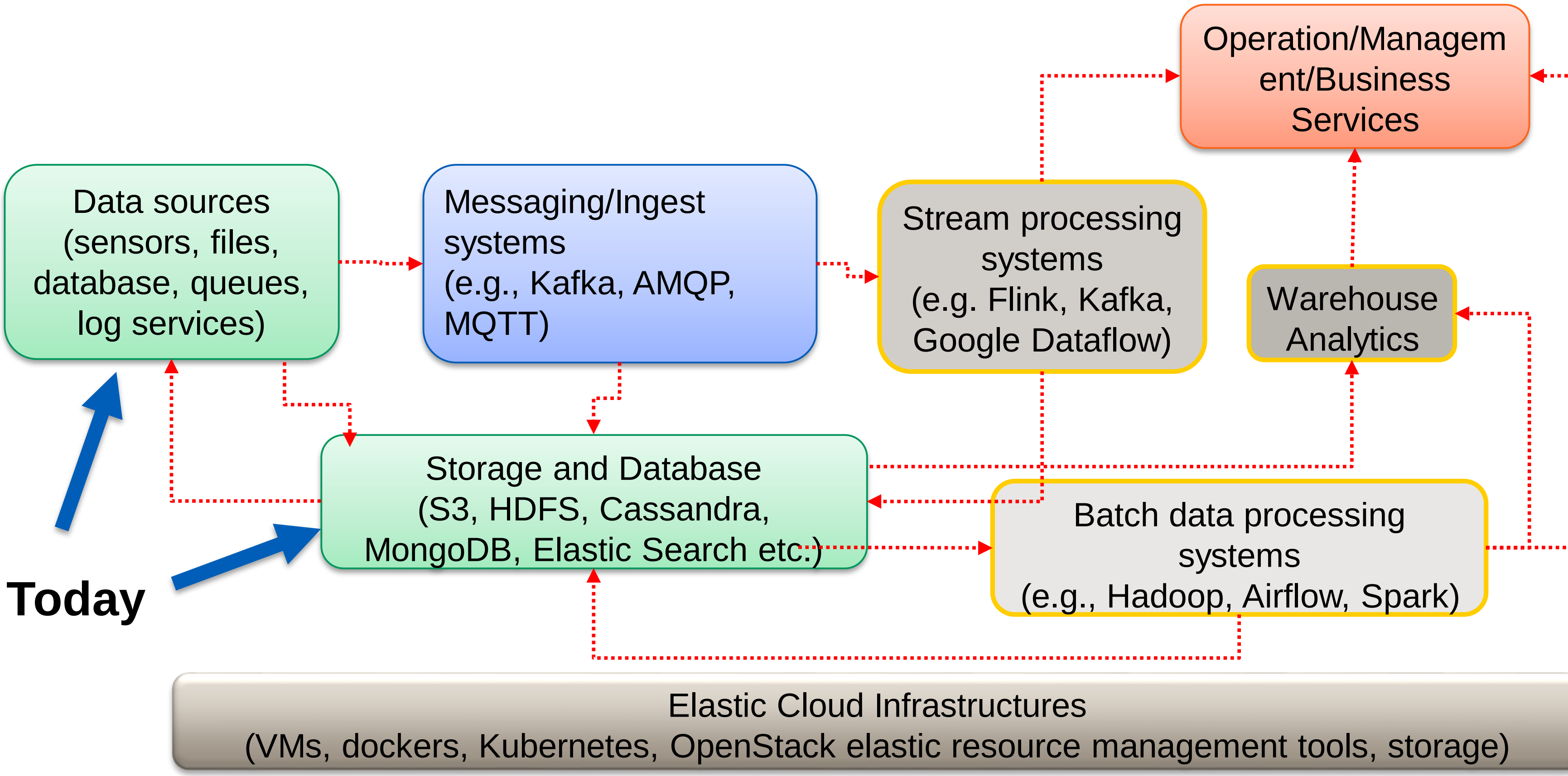
Big Data Storage and Database Services

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

Schedule

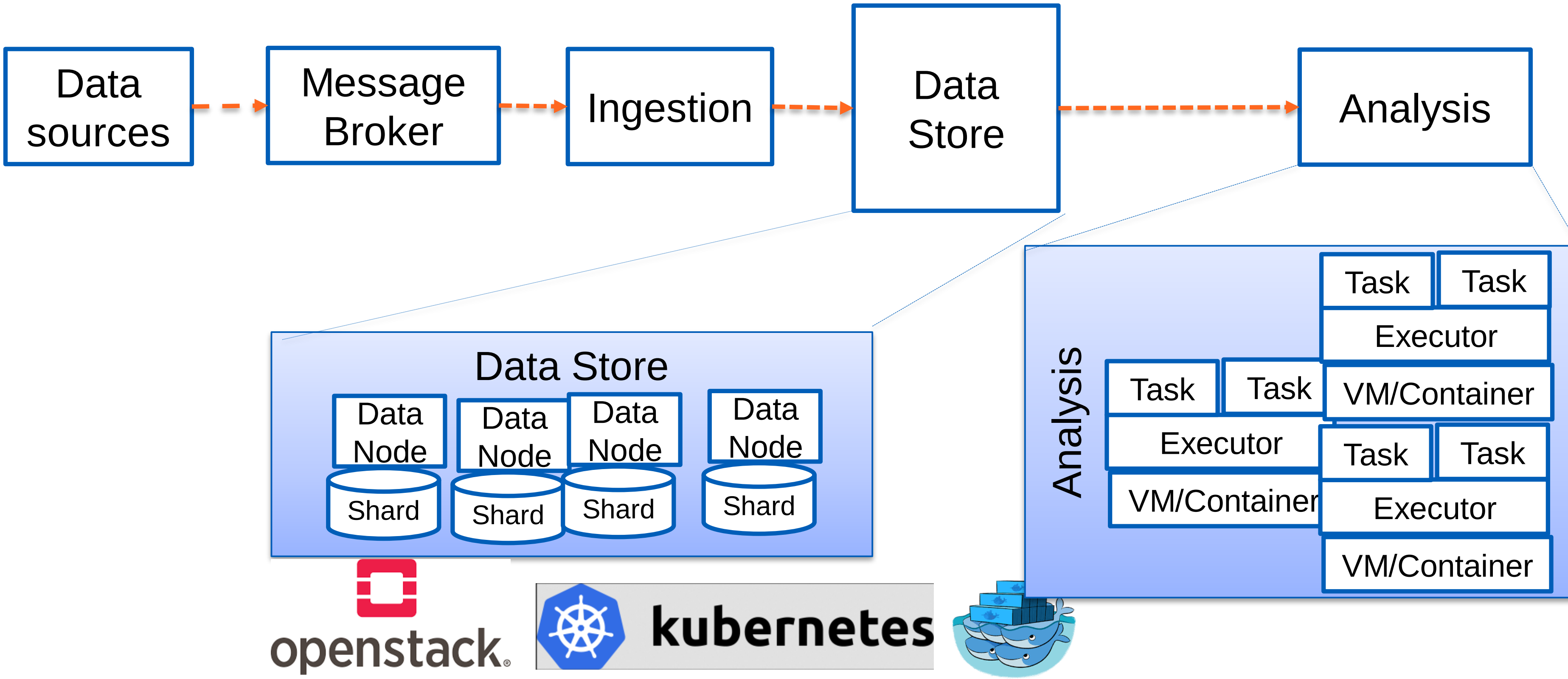
- **Consistency, availability and partition tolerance**
- **Data models and data management**
- **Polyglot persistence and making data available across systems**

Big data at large-scale

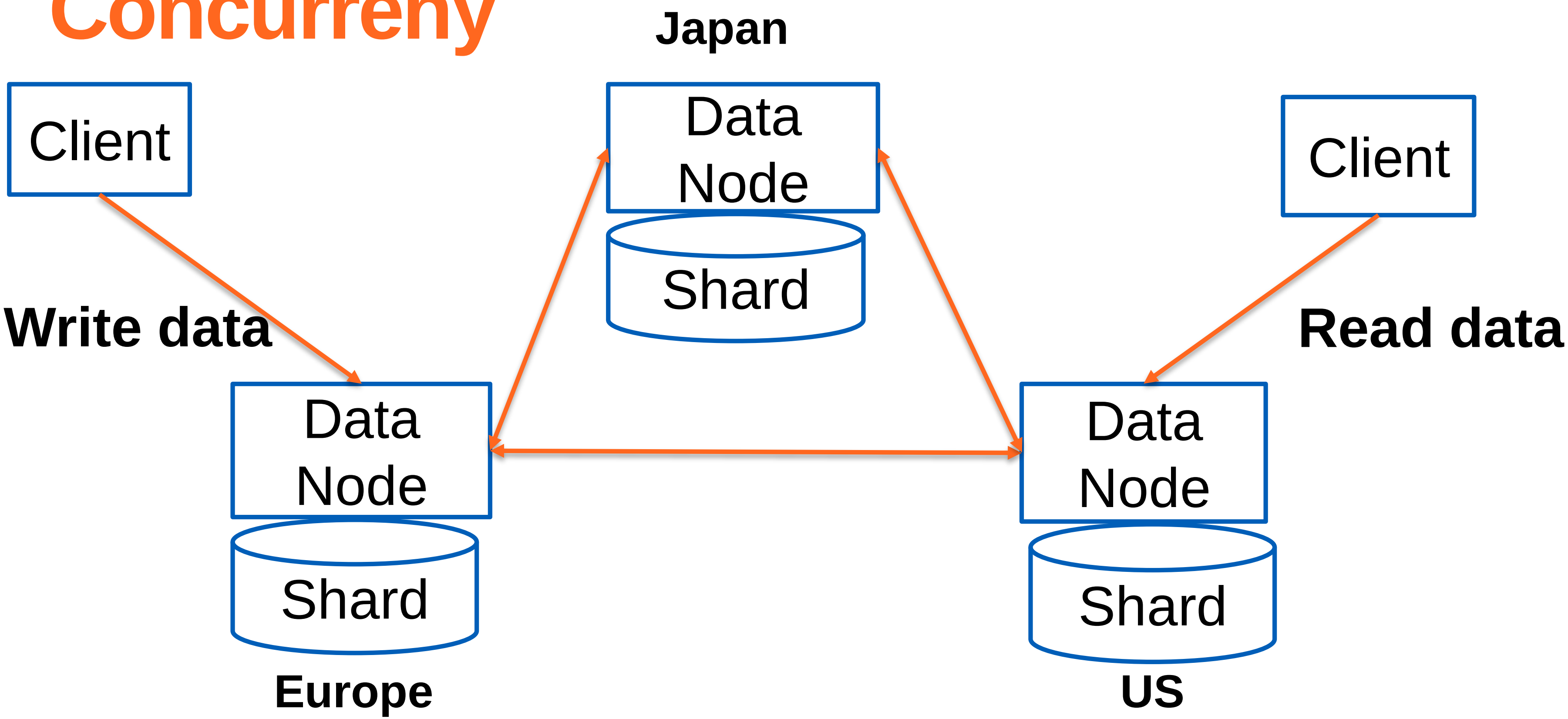


Consistency, Availability and Partition Tolerance

Runtime view of some components



Distribution, Replication and Concurreny



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

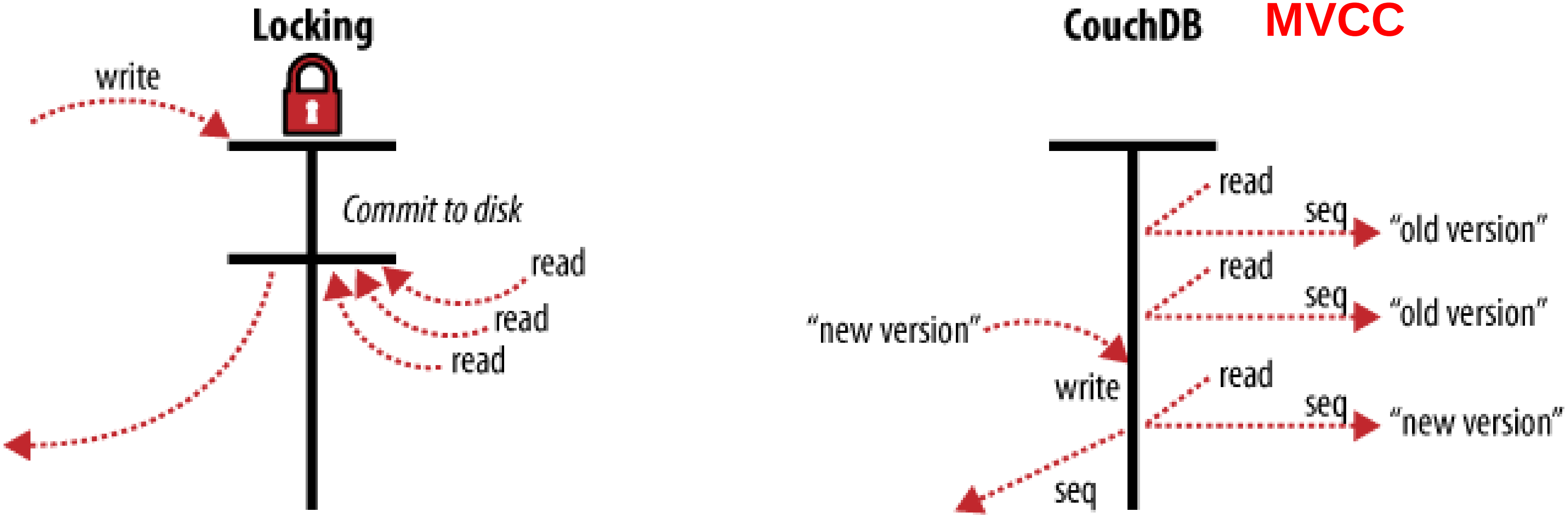


Figure source: https://docs.couchdb.org/en/stable/intro/consistency.html

Why is it hard to ensure ACID in big data platforms?

Key issues in big data management

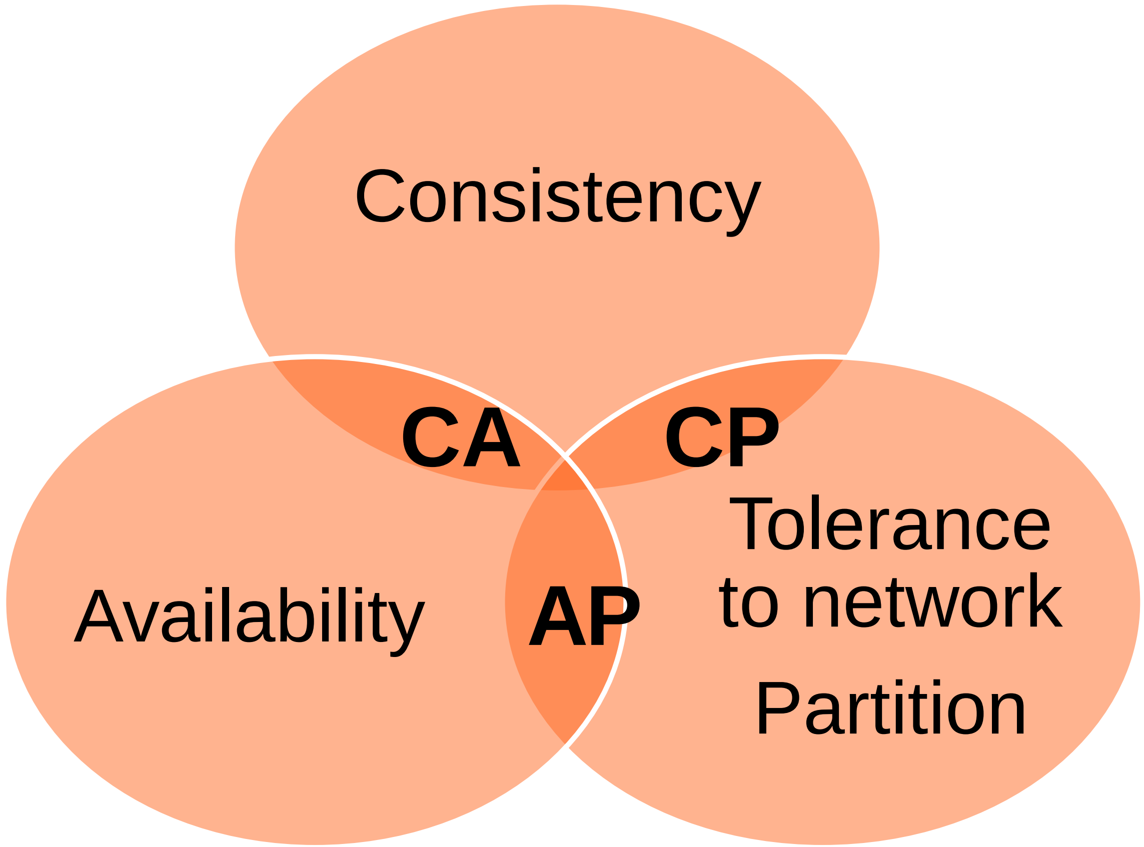
- Can every client see the same data when accessing any node in the platform?
- Can any request always receive a response?
- Can the platform serve clients under network failures?

Key issues in big data management

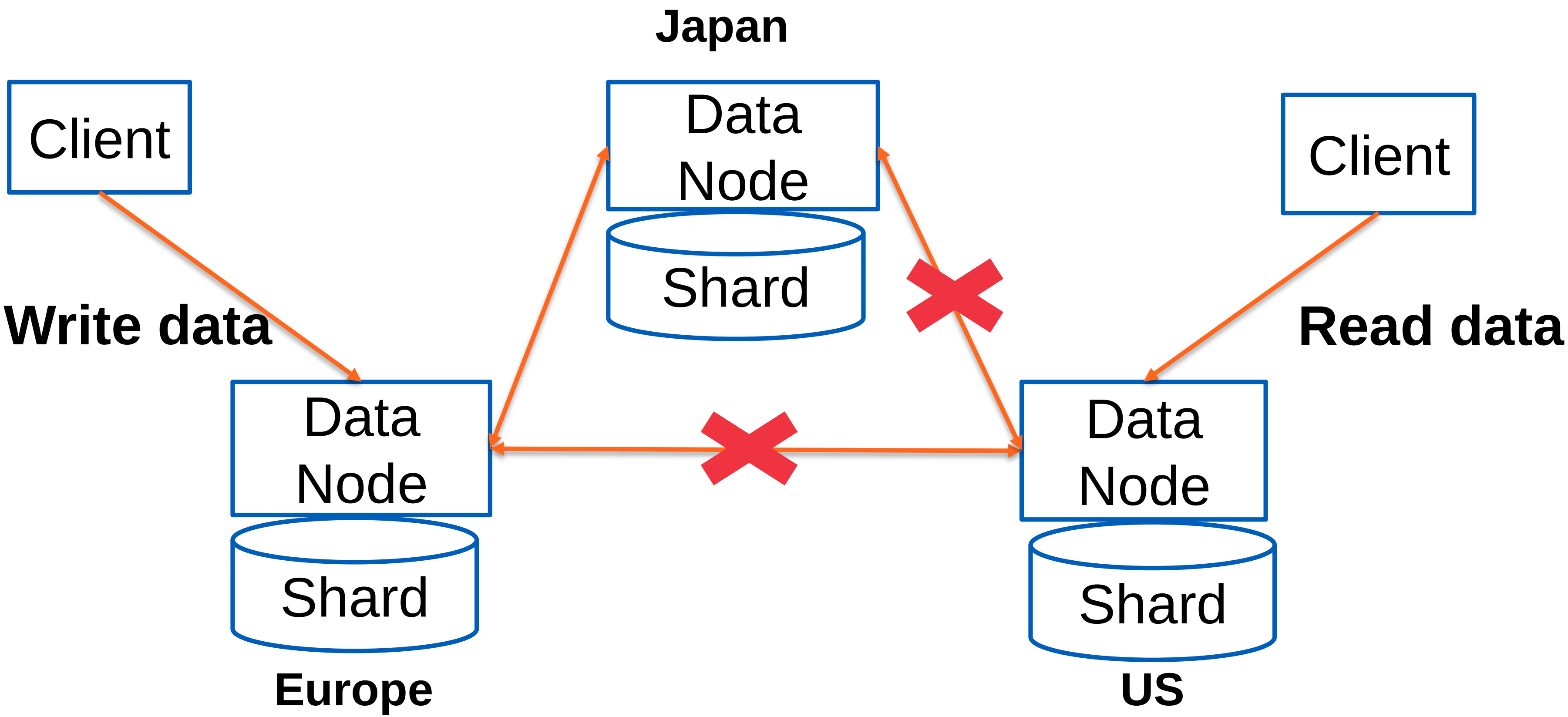
- **Tolerance to Network Partition**
  - if any node fails, the system is still working → a very strong constraint in our big data system design
- **High Consistency**
  - every read from a client must get the most up-to-date result
  - if the network fails, the newest write might not be updated to all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

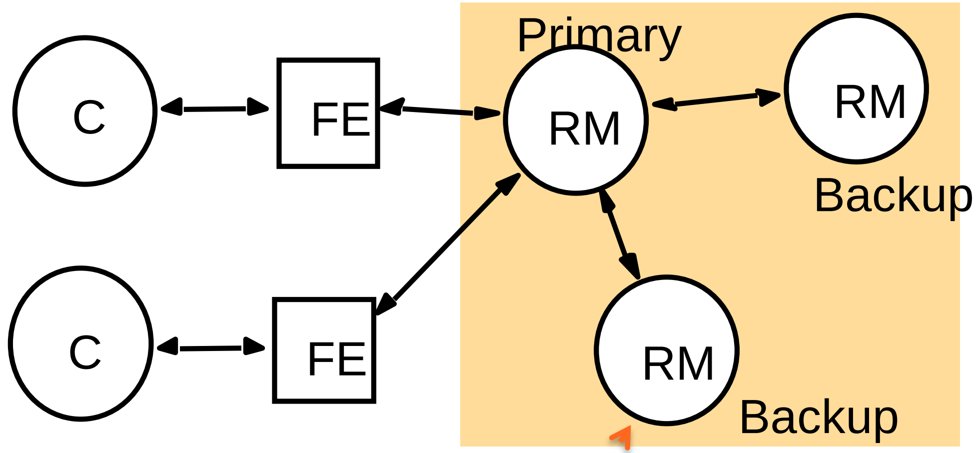
Programming consistency levels

- **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 systems and designs

Remember this model?

Passive (Primary backup) model

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



Single-leader replication architecture

Easy to deploy, globalize, manage and replace using cloud resources

Example: 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

Summary: 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**

Summary: key expectations for designing big data services

- **Designers: which one do you support?**
  - ACID or BASE?
  - Support programmable consistency guarantees?
- **Programmers**
  - How 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!**

Tomorrow tutorial: we play with Cassandra and consistency

Have you investigated consistency in your “mysimbdp”?

Data Models and Data Management

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**

Blob data

Big files:

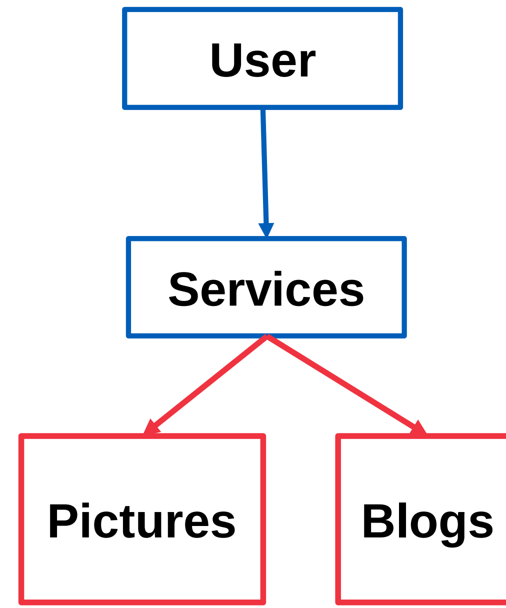
- Pictures, documents, big log files, images, video, backup data

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



Example - Amazon S3

Store blob files and their metadata

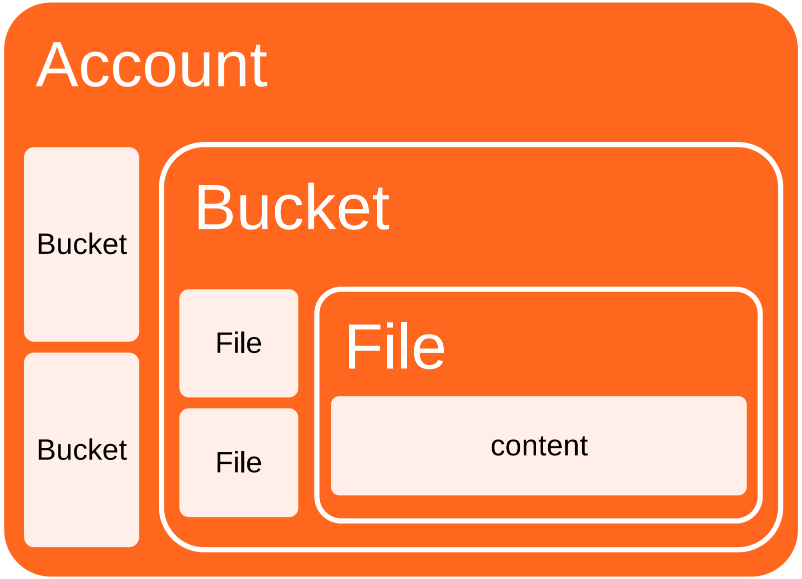
- Max 5TB per file
- A File is identified by a key

Structure

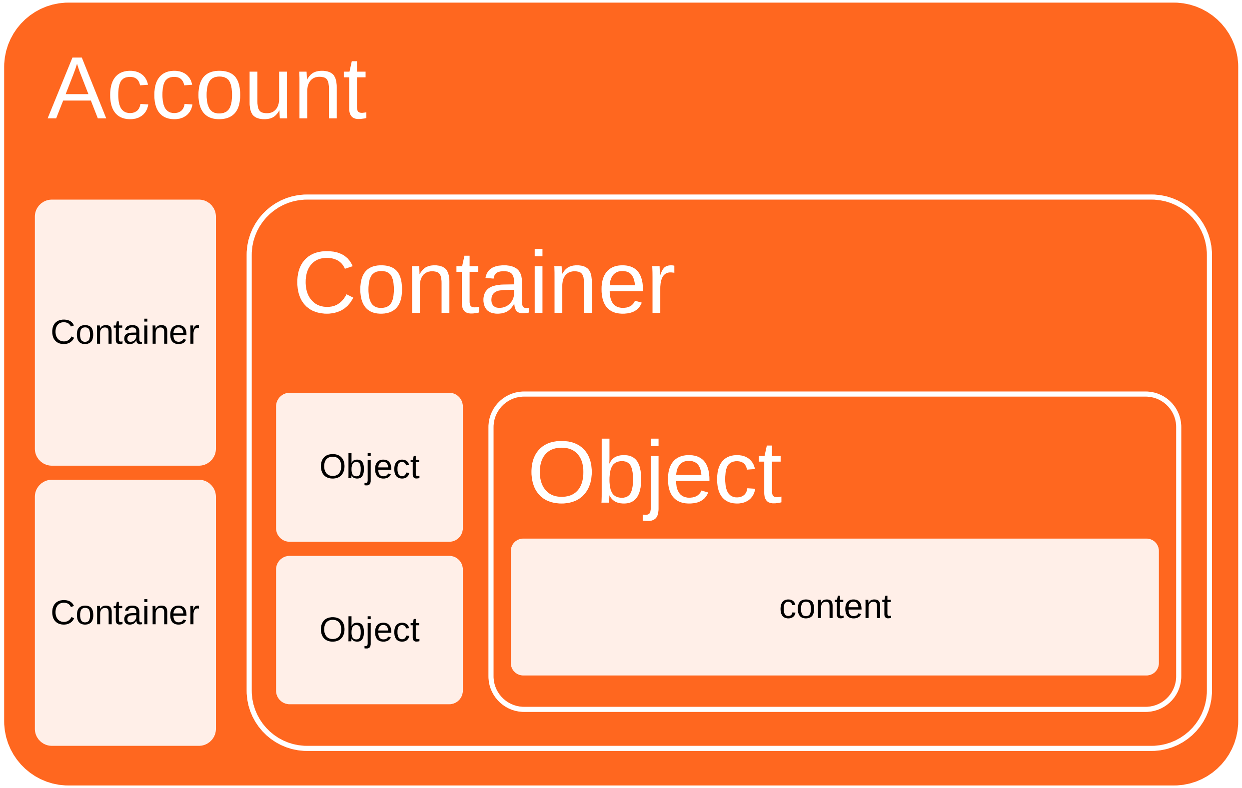
- File = Object
- Object: name and metadata
- Objects are organized into Buckets

Simple APIs

- REST



OpenStack Swift



http://docs.openstack.org/developer/swift/

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

Example: Alarm

stationparameters

Schema | Details | Preview

Field name	Type	Mode
reading_time	TIMESTAMP	NULLABLE
value	FLOAT	NULLABLE
station_id	INTEGER	NULLABLE
parameter_id		

stationdescription

Schema | Details | Preview

Field name	Type	Mode
station_id	INTEGER	NULLABLE
code	STRING	NULLABLE
name	STRING	NULLABLE
address	STRING	NULLABLE
description	STRING	NULLABLE
latitude	STRING	NULLABLE
longitude	STRING	NULLABLE

stationalarms

Schema | Details | Preview

Field name	Type	Mode
station_id	INTEGER	NULLABLE
alarm_id	INTEGER	NULLABLE
parameter_id	INTEGER	NULLABLE
start_time	TIMESTAMP	NULLABLE
end_time	TIMESTAMP	NULLABLE
value	FLOAT	NULLABLE
threshold	INTEGER	NULLABLE

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

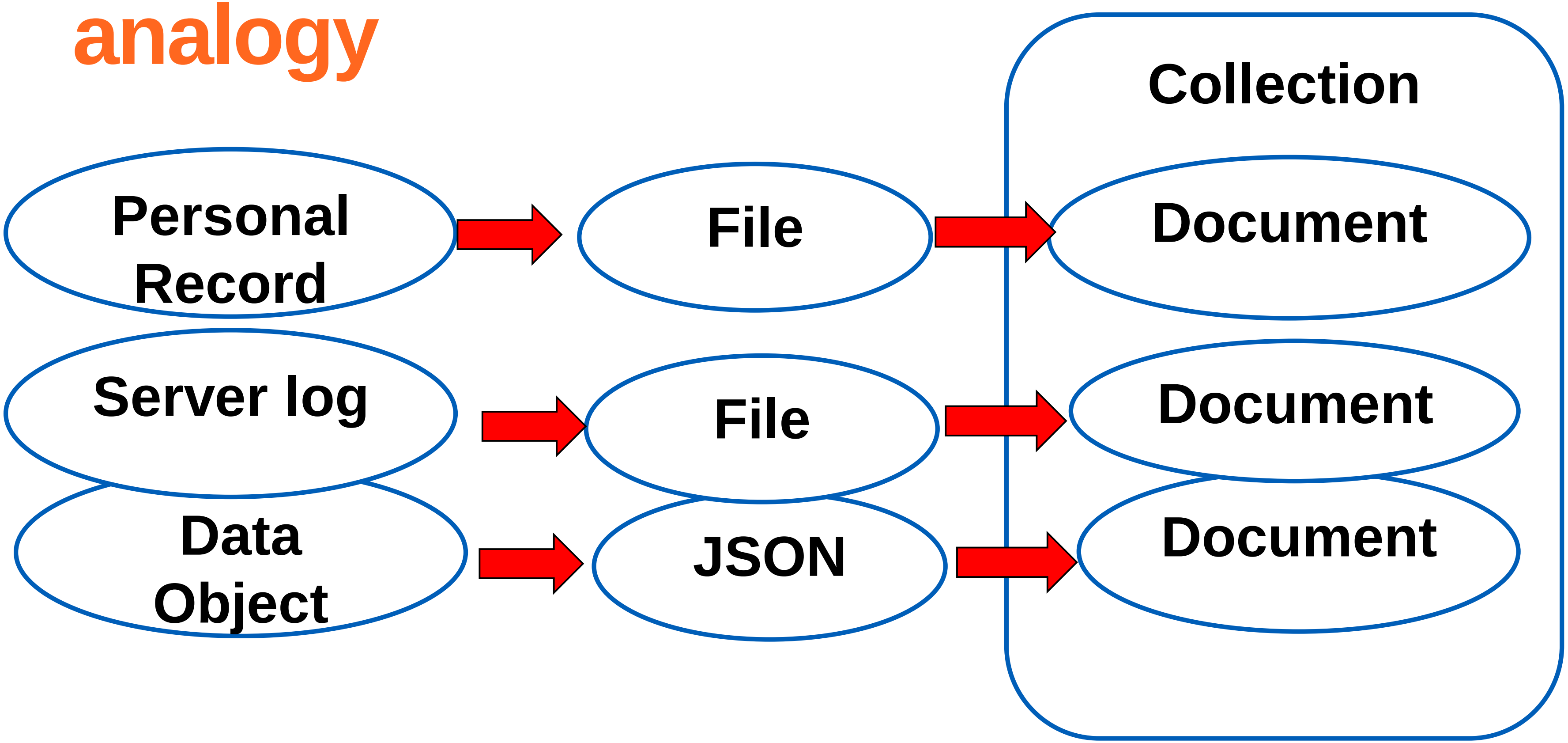
Example: Redis

- **http://redis.io/**
- **In-memory cache service**
  - *Store (key,value) tuples in memory but persistent back to database*
- Simple APIs
  - Well support with many programming languages
  - Widely used in big data ecosystems
- **Learning**
  - *https://redislabs.com/ provides a free account*

Example: Redis

http://redis.io/topics/benchmarks

Document-oriented model – simple analogy



Document-oriented Model

- **Documents**
  - flexible schema (schemaless) with flexible content
  - all values are schema-free and typically complex
  - 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.Atlas

https://www.mongodb.com/cloud/atlas

Graph-oriented model

- **Data is represented as a graph**
  - nodes or vertices represent objects
  - an edge describes 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)

Working with graph databases

- **Graph databases**
  - Auze CosmosDB, ArgangoDB, Titan, Grakn.AI, Neo4J, OrientDB
- **Query languages:**
  - Gremlin, SPARQL, Cypher
- **Graph computing frameworks**
  - Apache TinkerPop, Apache Spark GraphX

https://grakn.ai/

Column-family data model

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

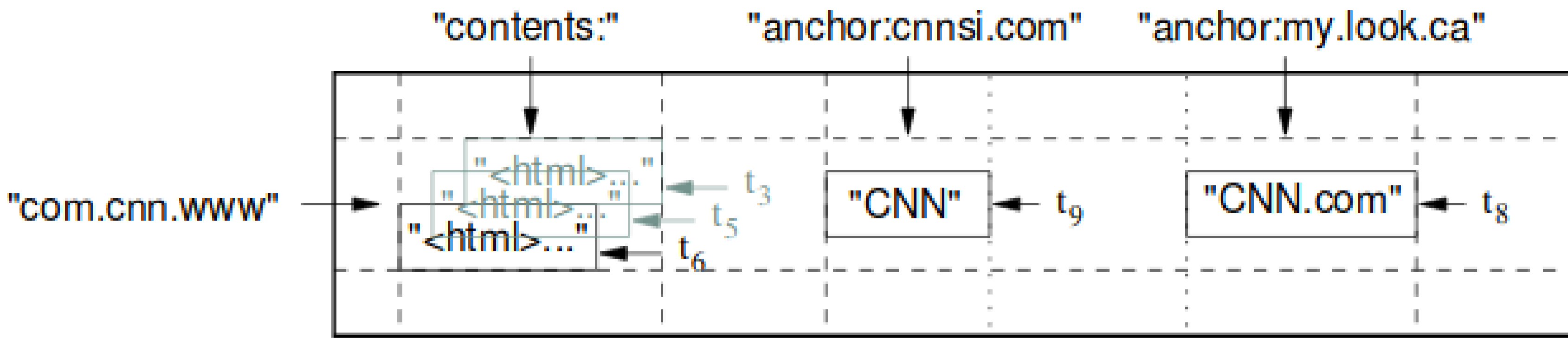
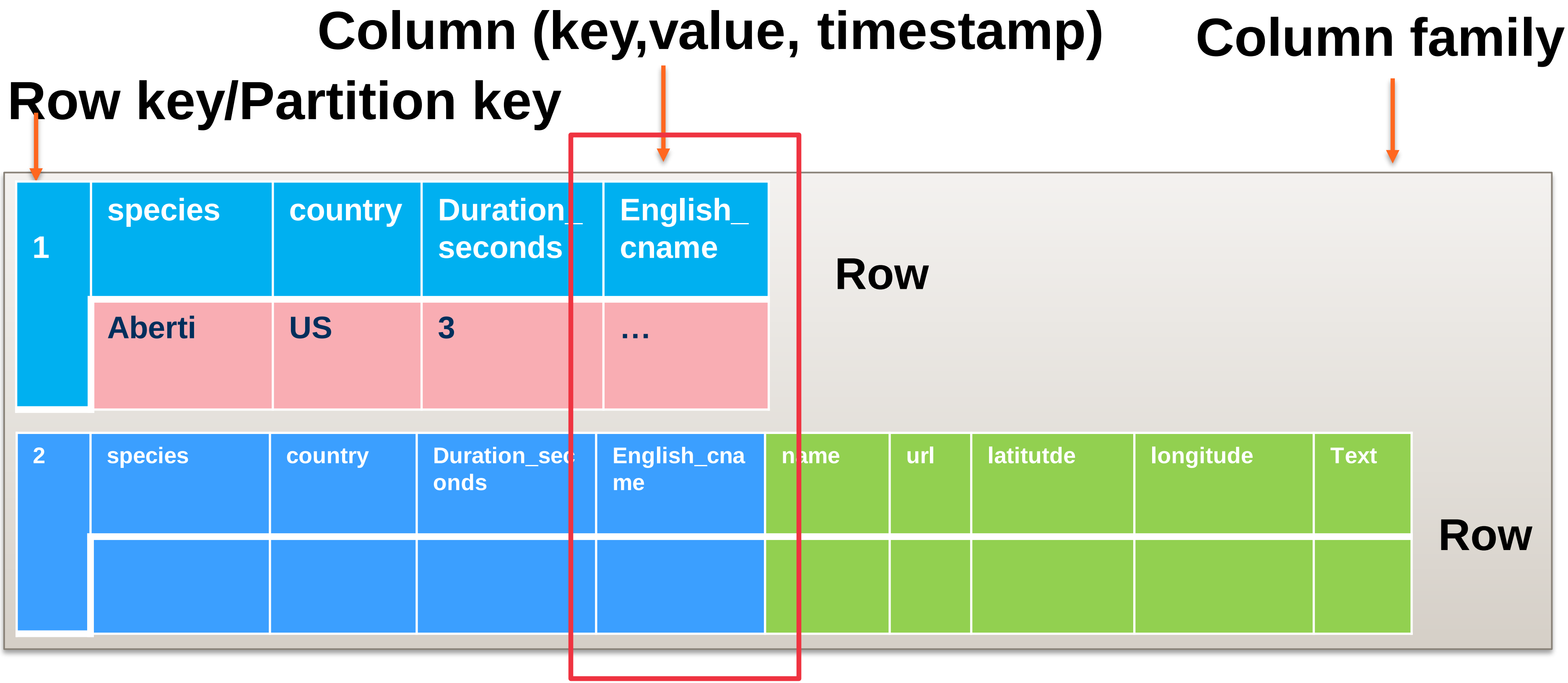


Figure source: Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2006. Bigtable: a distributed storage system for structured data. In Proceedings of the 7th symposium on Operating systems design and implementation (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
  - Column families can be grouped into a row

Example of a data model in Cassandra



Examples

Column (name, value, timestamp)

```
 english_cname              | writetime(english_cname)
----------------------------+--------------------------
 Black-tailed Gnatcatcher   |         1569966171073228

(1 rows)
```

Examples of rows

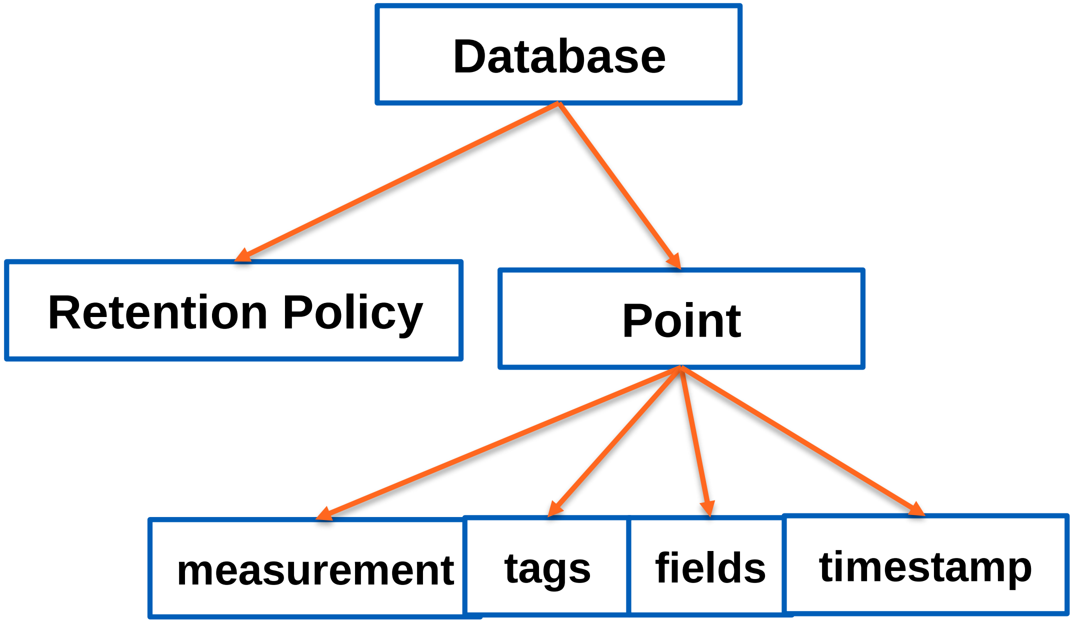
```
cassandra@cqlsh> select * from tutorial12345.bird2;

@ Row 1
------------------+--------------------------
 species          | melanura
 country          | Mexico
 duration_seconds | 29
 english_cname    | Black-tailed Gnatcatcher
 file_id          | 71907
 latitude         | 32.156
 longitude        | -115.793

@ Row 2
------------------+--------------------------
 species          | melanura
 country          | United States
 duration_seconds | 29
 english_cname    | Black-tailed Gnatcatcher
 file_id          | 358907
 latitude         | 33.7329
 longitude        | -115.8023
```

Time Series Database

- **So many types of data in big data are time series**
  - IoT measurements, session data, log, etc.
- **Of course you can also use other databases**
  - e.g., Cassandra, ElasticSearch, BigTable
- **Time Series Databases specially designed for time series data**
  - *examples: Riak TS (Time Series), InfluxDB, Apache Druid*

Example: InfluxDB

- https://www.influxdata.com/
- High-level query, SQL-alike Language
- Retention policy for data storage, sharding and replication



An example of InfluxDB

```
> show measurements
name: measurements
name
----
stationalarm
stationaparameter
> select * from stationalarm;
name: stationalarm
time             alarm_id   datapoint_id      station_id   value    valueThreshold
----             --------   ------------      ----------   -----    --------------
1487444343000000 308        121               1161115016   240      240
```

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?**

Summary: 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 change
- **Normalization and denormalization**
  - do we have to normalize data when storage is cheap?
  - but data consistency maybe a problem!
- **Mapping into large-scale computing infrastructure?**
  - data is for analytics

Polyglot persistence and making data available across systems

Polyglot Big Data models/systems

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

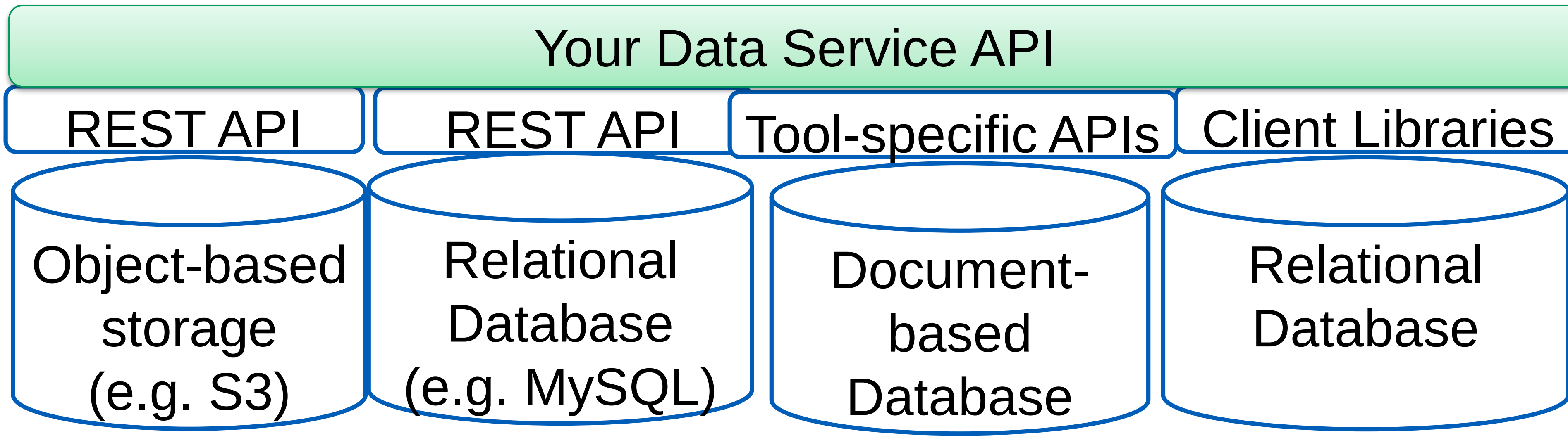
Polyglot persistence is inevitable for many use cases

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**
  - the same service that can host different data models

Multi databases/services

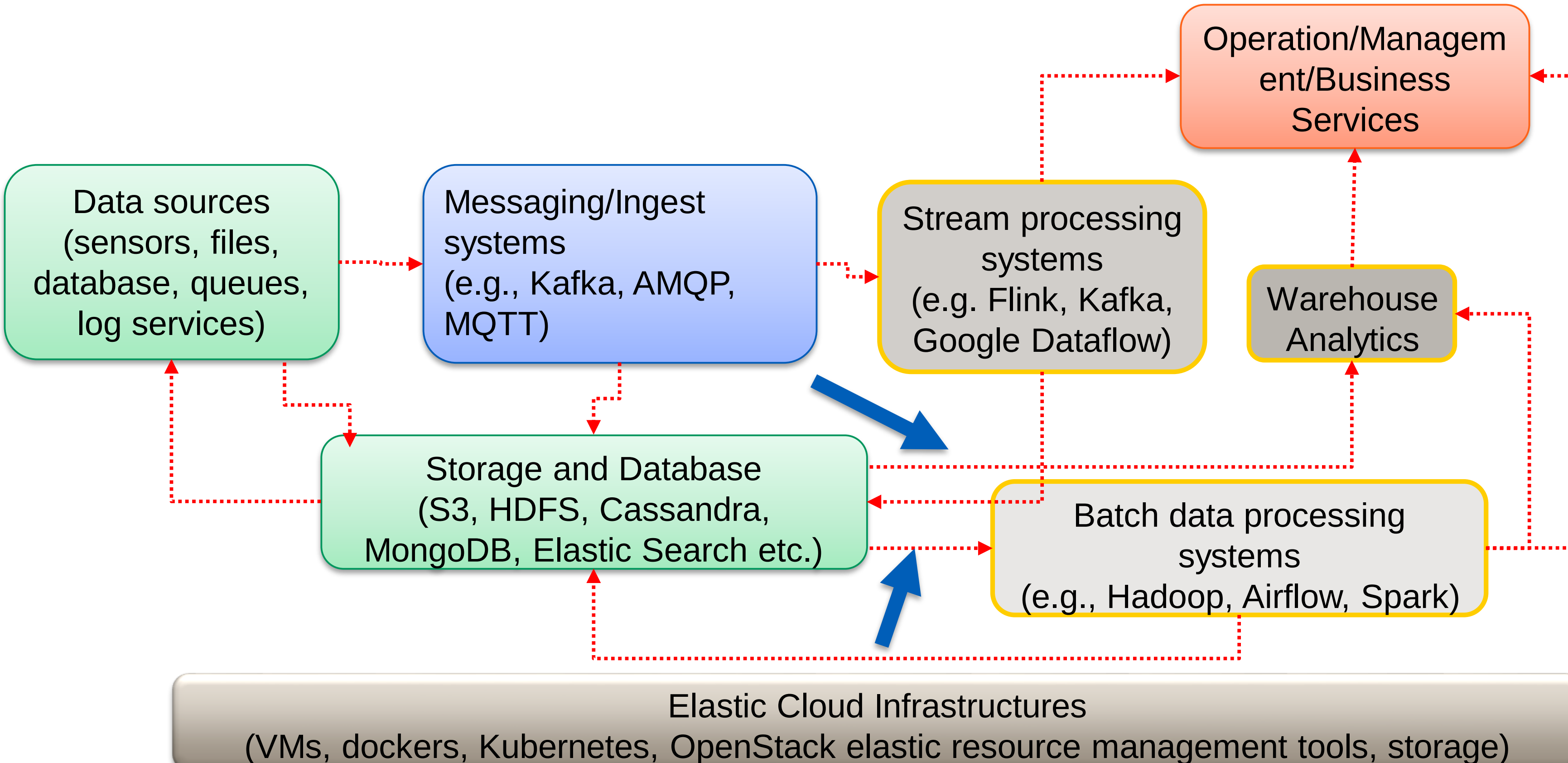
Data access APIs can be built based on well-defined interfaces



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

Big data at large-scale



Mapping to the analytics

- **Data layer must map/provide data to processing layer**
  - maximize the analytics possibilities
- **Key issues**
  - avoid data movement as much as possible
  - avoid contention between data management and analytics
- **Techniques**
  - “mount”, specific connectors/drivers, copy-process-remove activities

Mount/”Fuse”

- **Mapping a remote storage as a local file system**
  - Blobfuse (Microsoft Azure), gcsfuse (Google Storage)
  - the network performance is important

Connectors

ODBC or other specific protocol connectors

Your Service Storage and Database → CONNECTOR → Your customer processing systems/BI (e.g., Airflow, Spark, Drill)

Example

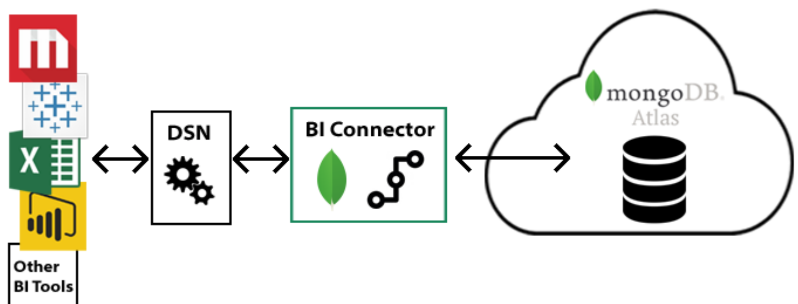


Figure source: https://docs.mongodb.com/bi-connector/master/

“Copy and Process”

Client libraries are used to move data from storages and databases to processing places

Examples:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from cassandra.cluster import Cluster

cluster = Cluster(contact_points=hosts, port=9042,auth_provider=auth_provider)
session = cluster.connect(“tutorial12345”)
sql_query = "SELECT * FROM tutorial12345.bird1234;“
df = pd.DataFrame()
rows= session.execute(sql_query)
df = rows._current_rows
print(df)
```

Summary: 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

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io