Aalto University
School of Science

# Big Data Storage and Database Services – common systems & integration problems

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

Aalto University
School of Science

# Common data models and data storage/database systems

# Common data models

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

# Special file/table formats in big data

- **For analytical big data, remember the patterns about data**
  - write once, read many
  - analytics queries often access data based on “columns” (e.g., sum of all “trip payments”)
- **File/table formats**
  - compression, columnar representation for column-based queries/accesses, encryption
  - suitable with big data analytics (e.g., Spark, Hadoop)
- **Examples**
  - Apache ORC (https://orc.apache.org/)
  - Apache Parquet (https://parquet.apache.org/)
  - Apache Iceberg (https://iceberg.apache.org/spec/)
- **They are the file/table formats under many big data systems**

# Blob data

**Big files:**

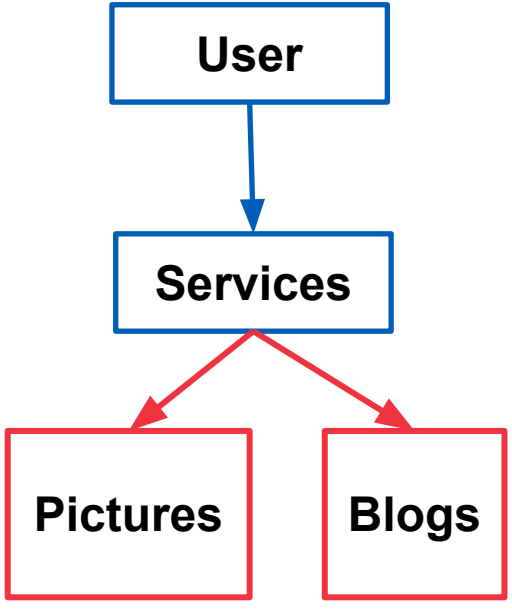
- Pictures, documents, big log files, images, video, backup data

**Storage**

- File systems or blob storage

**Implementations**

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



# Example - Amazon S3

**Store blob files and their metadata**

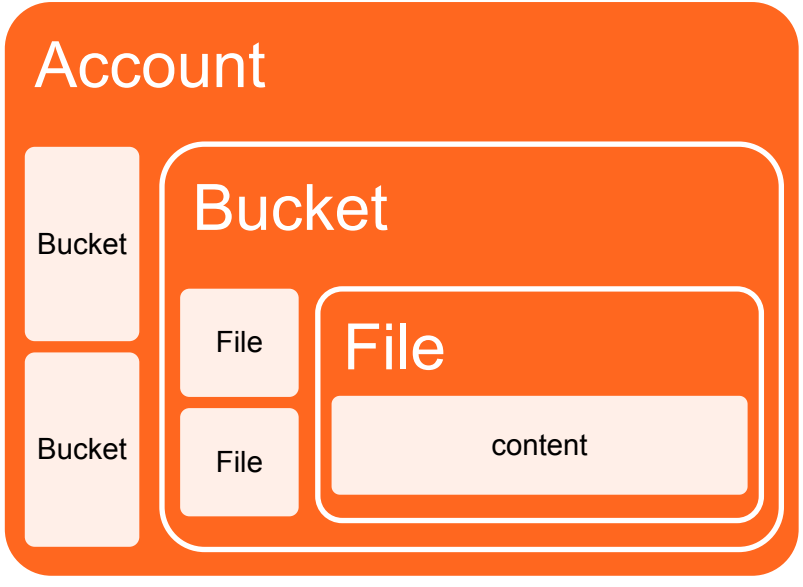
- Max 5TB per file
- A file is identified by a key

**Structure**

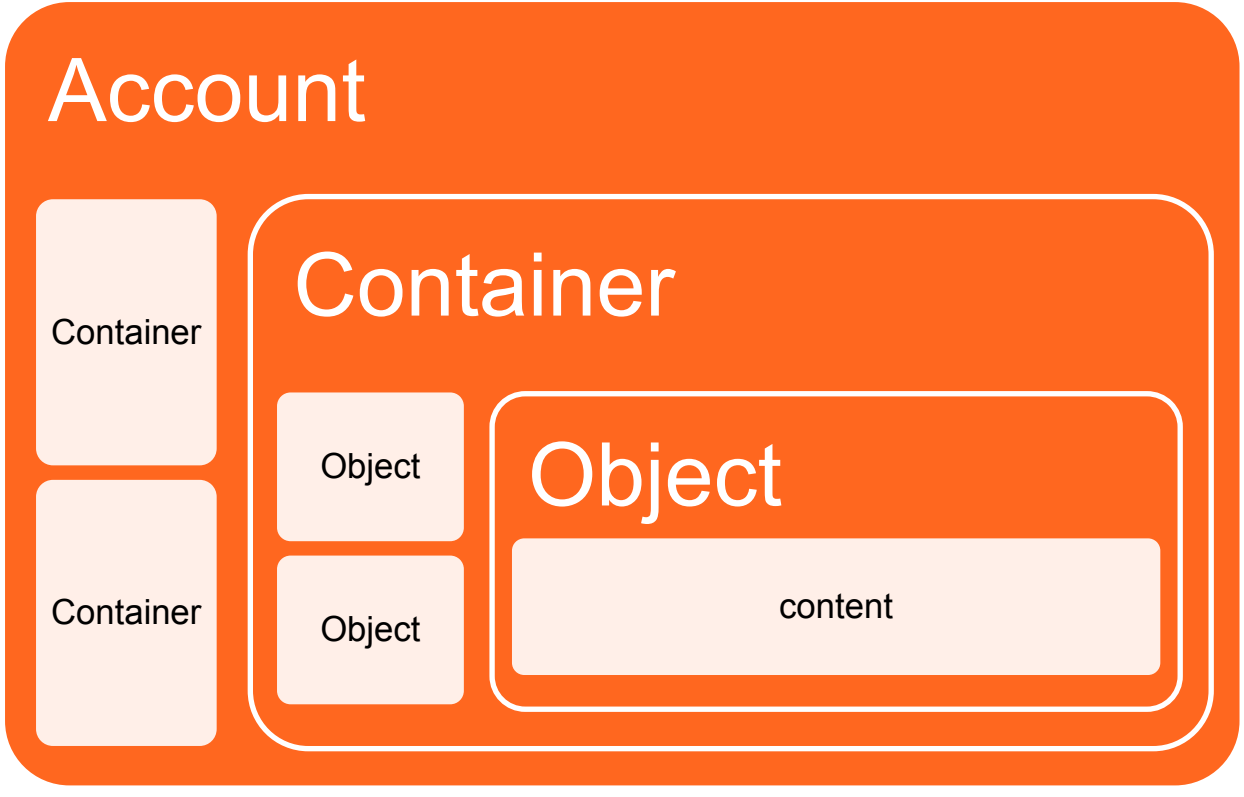
- File = Object
- Object: name and metadata
- Objects are organized into Buckets

**Simple APIs**

- REST



# OpenStack Swift




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

# Minio

- Check https://min.io/
- For different deployment models: Kubernetes, VMs, edge-cloud
- S3 compatibility

# 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

# Example: Alarm in BigQuery

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 |

stationparameters

Schema | Details | Preview

| Field name | Type | Mode |
|---|---|---|
| reading_time | TIMESTAMP | NULLABLE |
| value | FLOAT | NULLABLE |
| station_id | | |
| parameter_id | | |

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 |

# 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

# Relational Databases for big data scenarios

- **Relational database at very large-scale**
  - Amazon Aurora, CockroachDB, Microsoft Azure SQL Data Warehouse
- **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, Valkey, Apache Accumulo

# Example: Redis

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

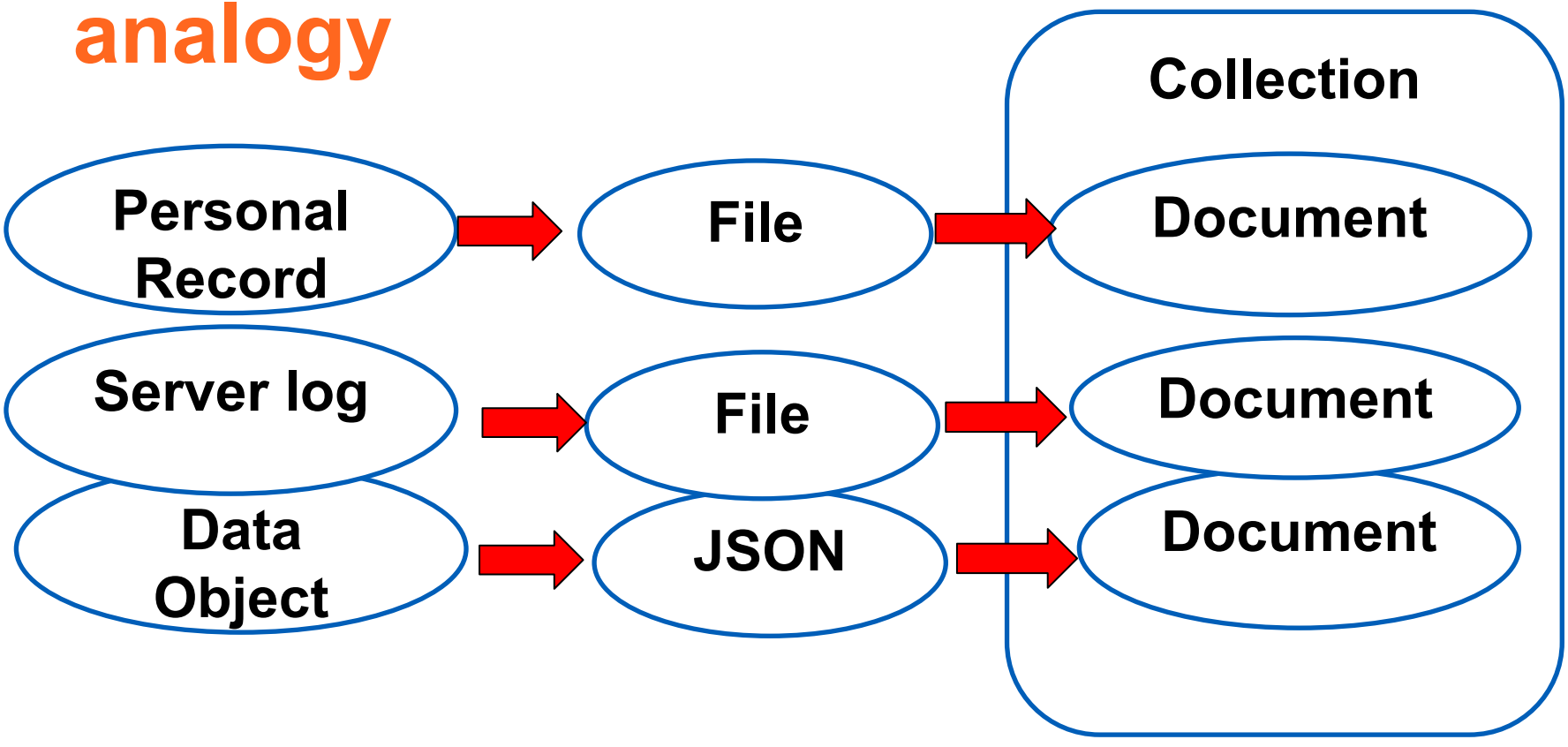
# Example: Redis

http://redis.io/topics/benchmarks

# Document-oriented Model

- **Documents**
  - flexible schema (schemaless) with flexible content
  - data fields can be  complex for 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

# Document-oriented model – simple analogy



# Examples: MongoDB.Atlas

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

# 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)

# 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)

# Working with graph databases

- **Graph databases**
  - Azure CosmosDB, ArangoDB, Titan, TypeDB, Neo4J, OrientDB
- **Query languages:**
  - Gremlin, SPARQL, Cypher
- **Graph computing frameworks (analysis)**
  - Apache TinkerPop, Apache Spark GraphX

# Example

**https://github.com/vaticle/typedb**

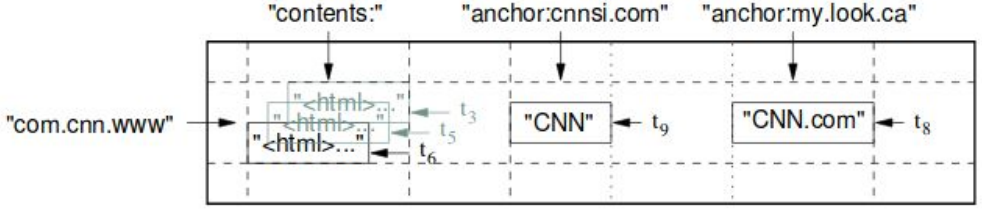
# Column-family data model

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

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

**Column Key =Family: qualifier**

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



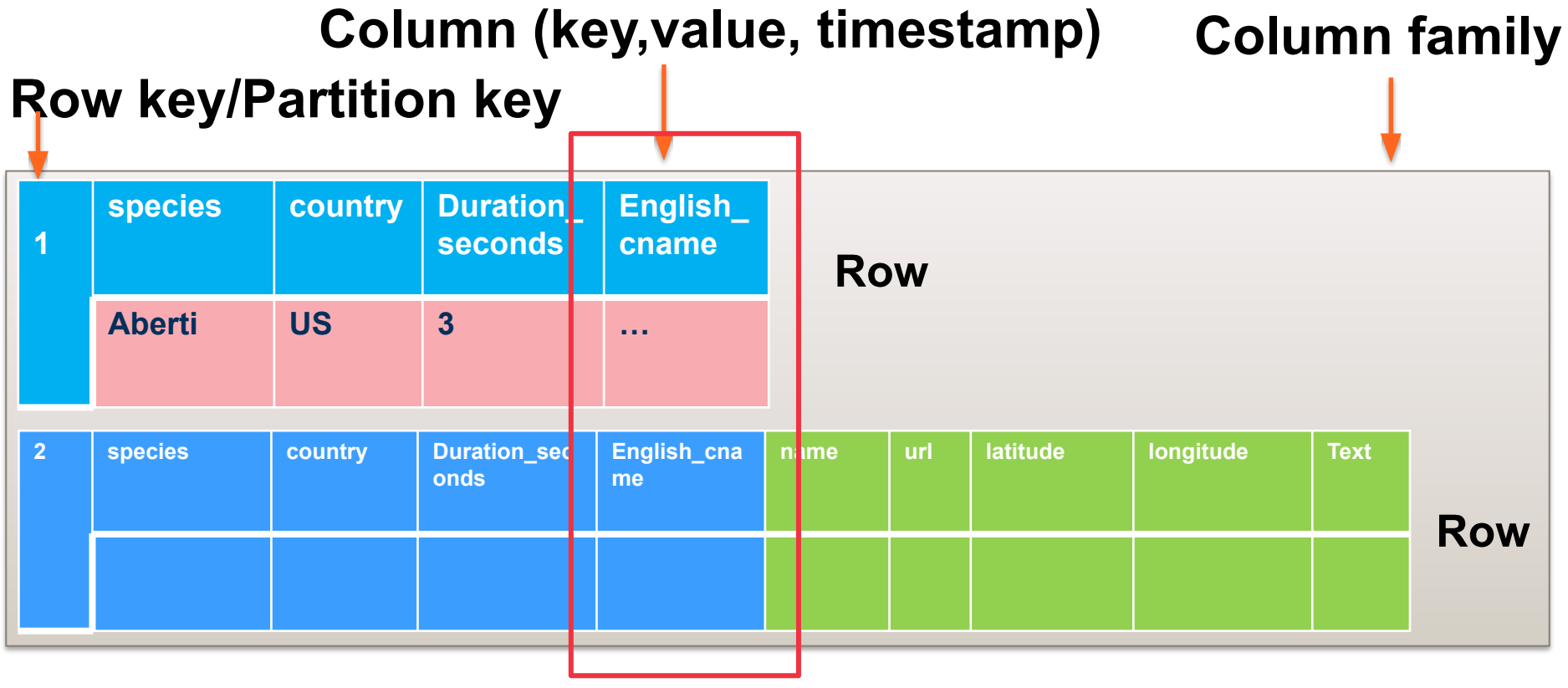

Examples: Cassandra, HBase

**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
    - flexible, wide columns → save spaces

# 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

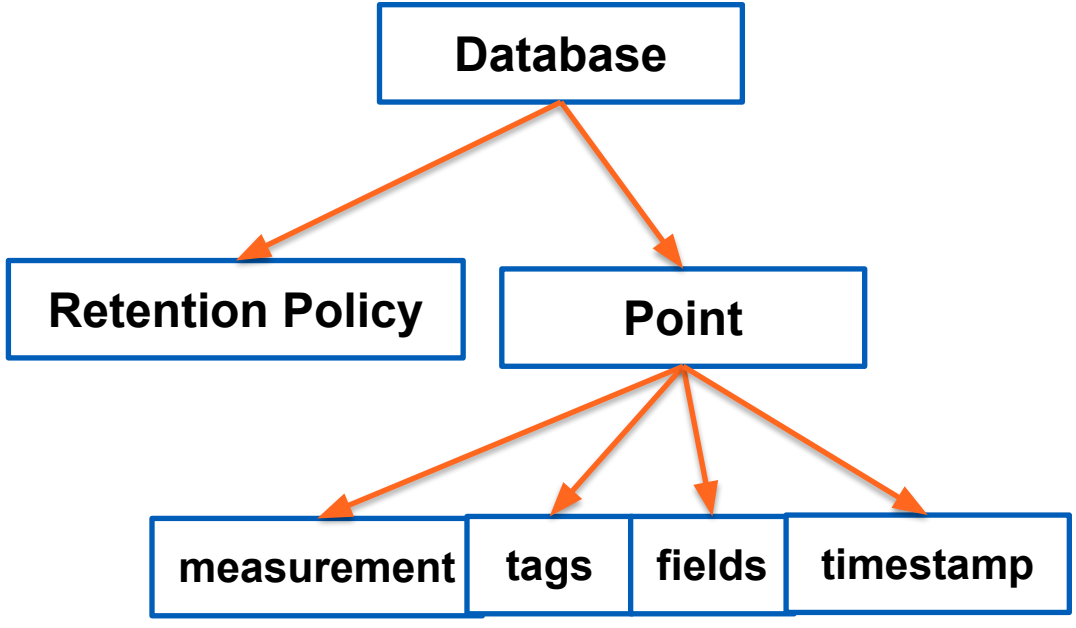
```
cassandra@cqlsh> select * from tutorial12345.bird2;

@ 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?**

Aalto University
School of Science

# Interfaces between a data storage/databases system and its external analysis systems

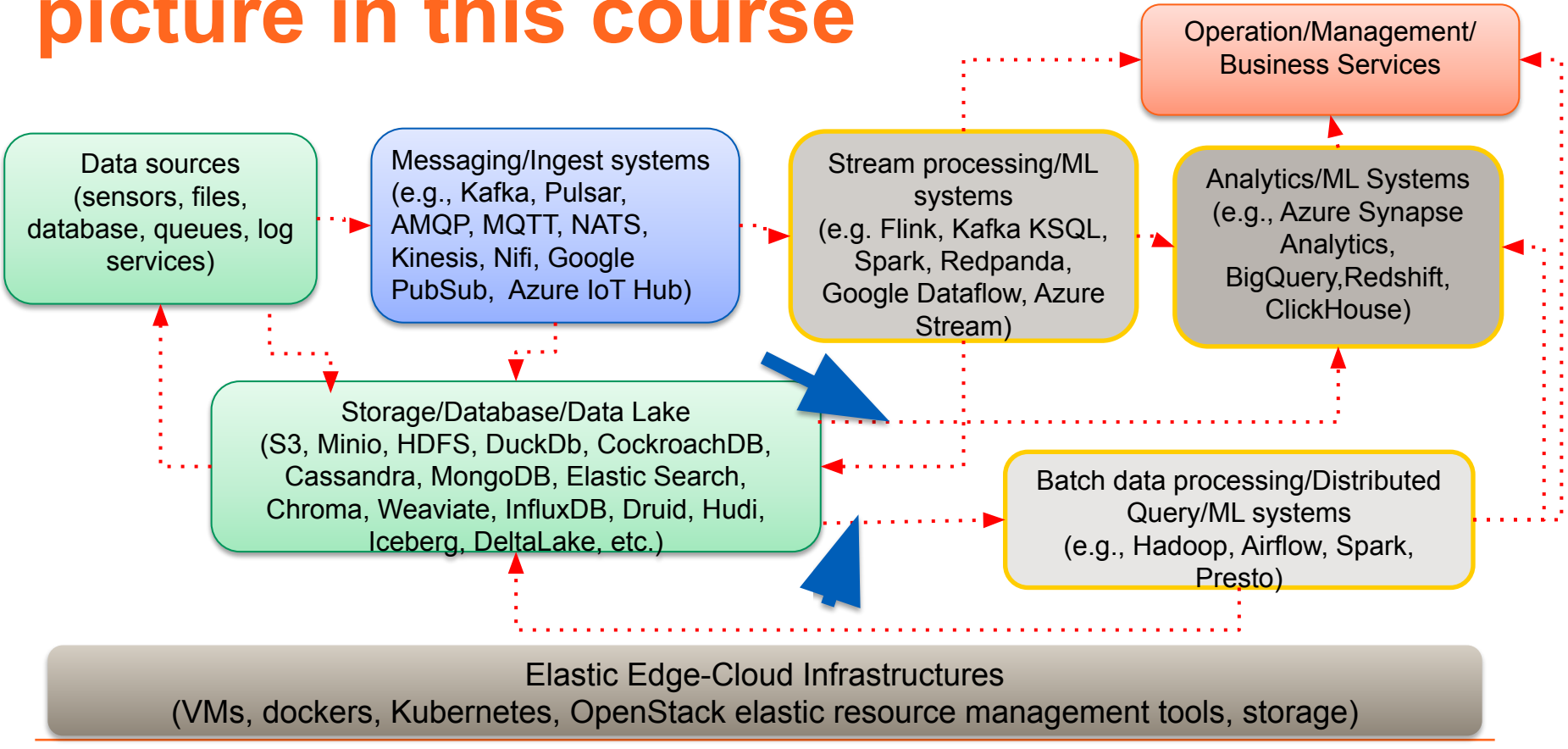
Aalto University
School of Science

# In many cases: the data in data storage/database service must be made available for large-scale analysis:

## large-scale analytics and data are managed by different systems

## an important consideration in big data platforms design!

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



# Making data available to the analytics

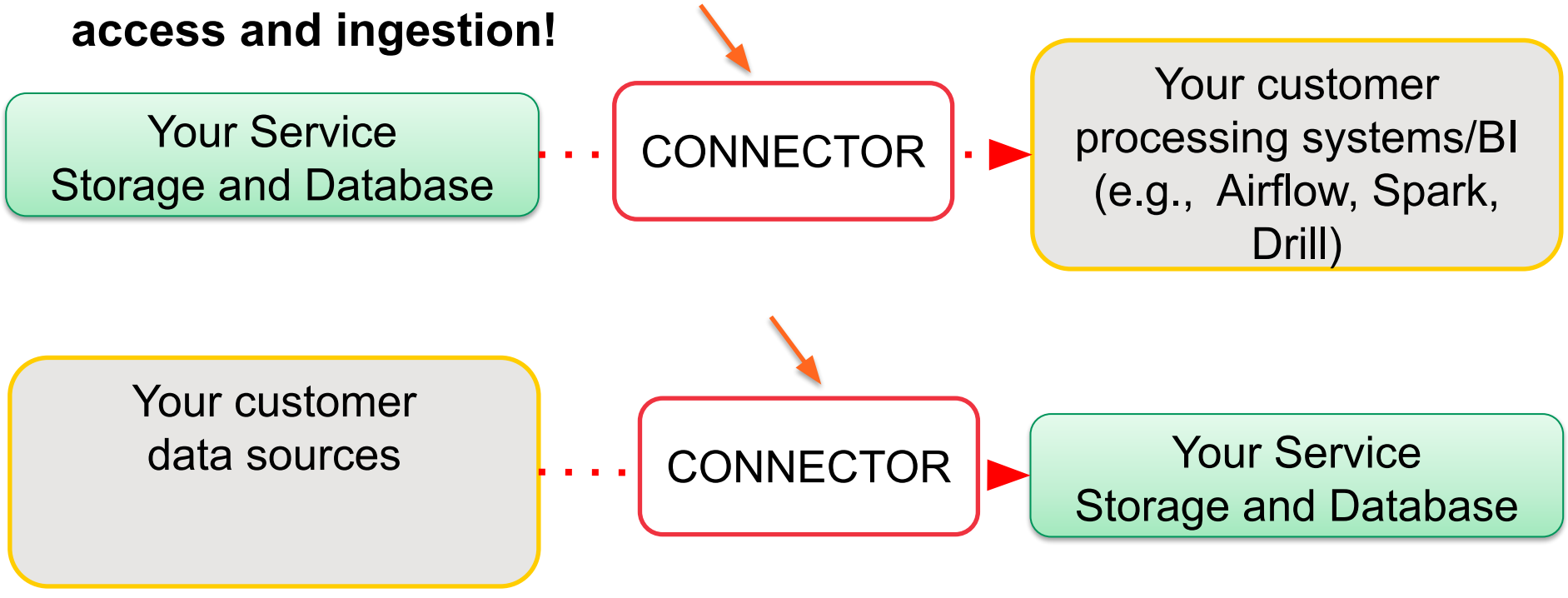
- **Data layer must map/provide data to processing layer**
  - maximize the analytics possibilities
- **Key issues**
  - avoid data movement as much as possible
  - avoid contention between the data management and the data analytics system
- **Techniques**
  - “mount”, specific connectors/drivers, copy-process-remove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

# Mount/"Fuse"

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

# Connectors

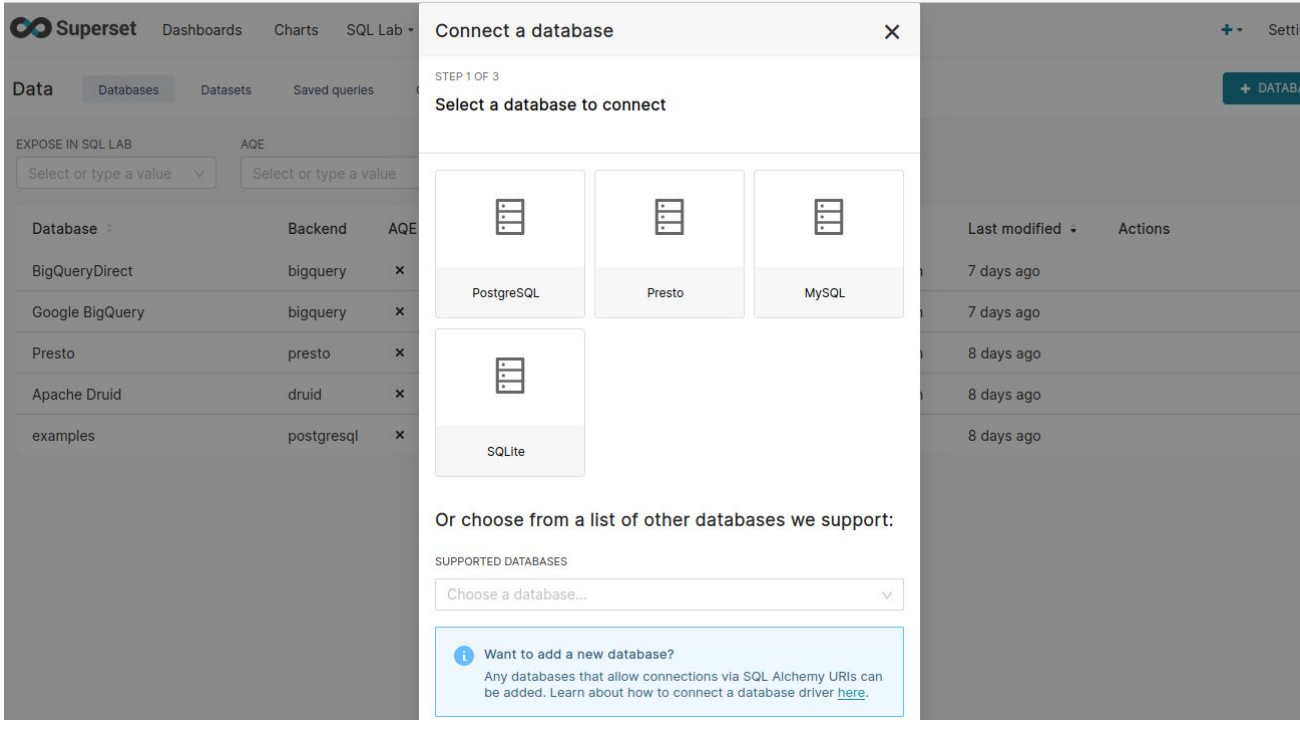
**ODBC or other specific protocol connectors to enable data access and ingestion!**



# Example: Superset connectors

https://superset.apache.org/

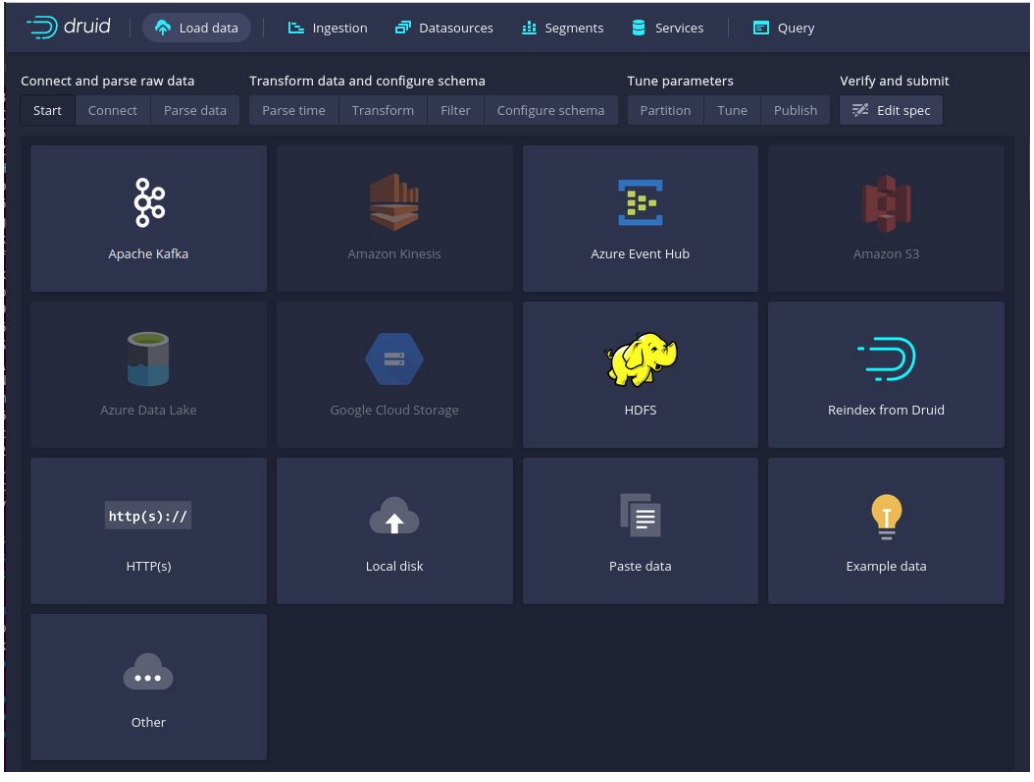
**Connectors to different types of databases/datasets to retrieve and analyze data**



# Example: Druid

**https://druid.apache.org/**

**Different types of connectors (e.g., Kafka, Files, S3, etc.) to allow data ingestion into the database**



# Analytics and Cloud Storage

- **Various connectors for making data in cloud storages available for analytics**
- **Apache Hadoop/Spark (data analysis) can work with Amazon S3, OpenStack Swift, Google Cloud Storage**
- **Examples:**
  - https://github.com/GoogleCloudDataproc/hadoop-connectors
  - https://spark.apache.org/docs/latest/cloud-integration.html

# “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)
```

# Distributed SQL queries: 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

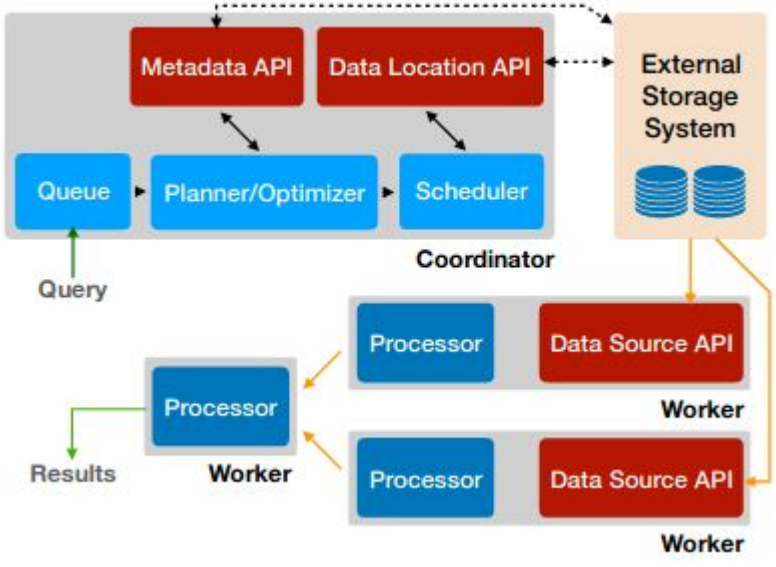
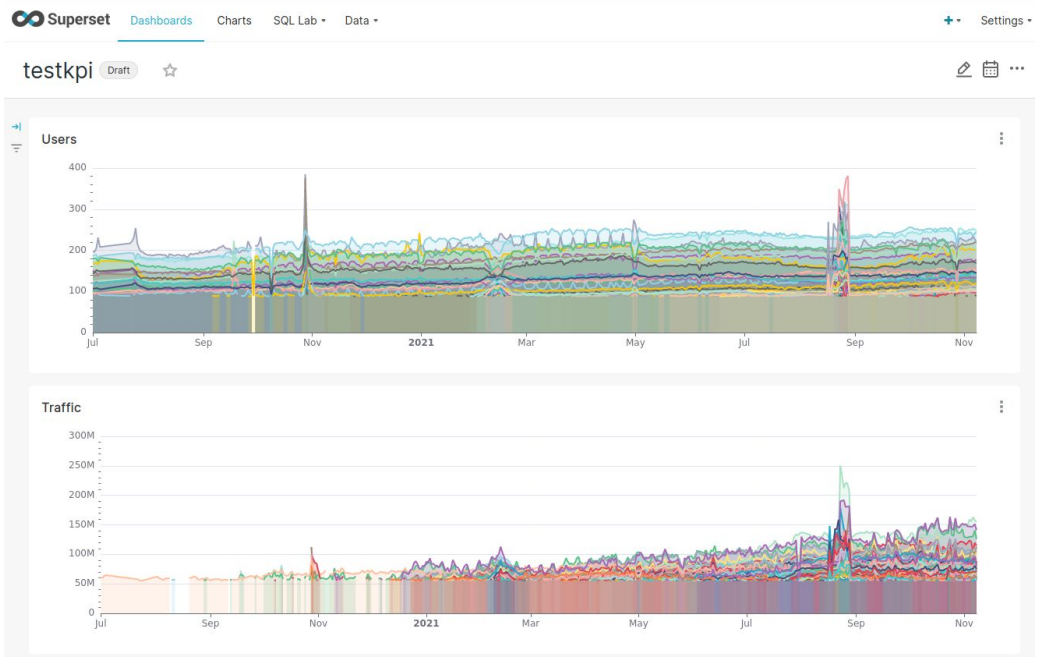



**Figure source:** *Presto: SQL on Everything https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8731547&tag=1*

# 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

# Thanks!


## Hong-Linh Truong
## Department of Computer Science


**rdsea.github.io**