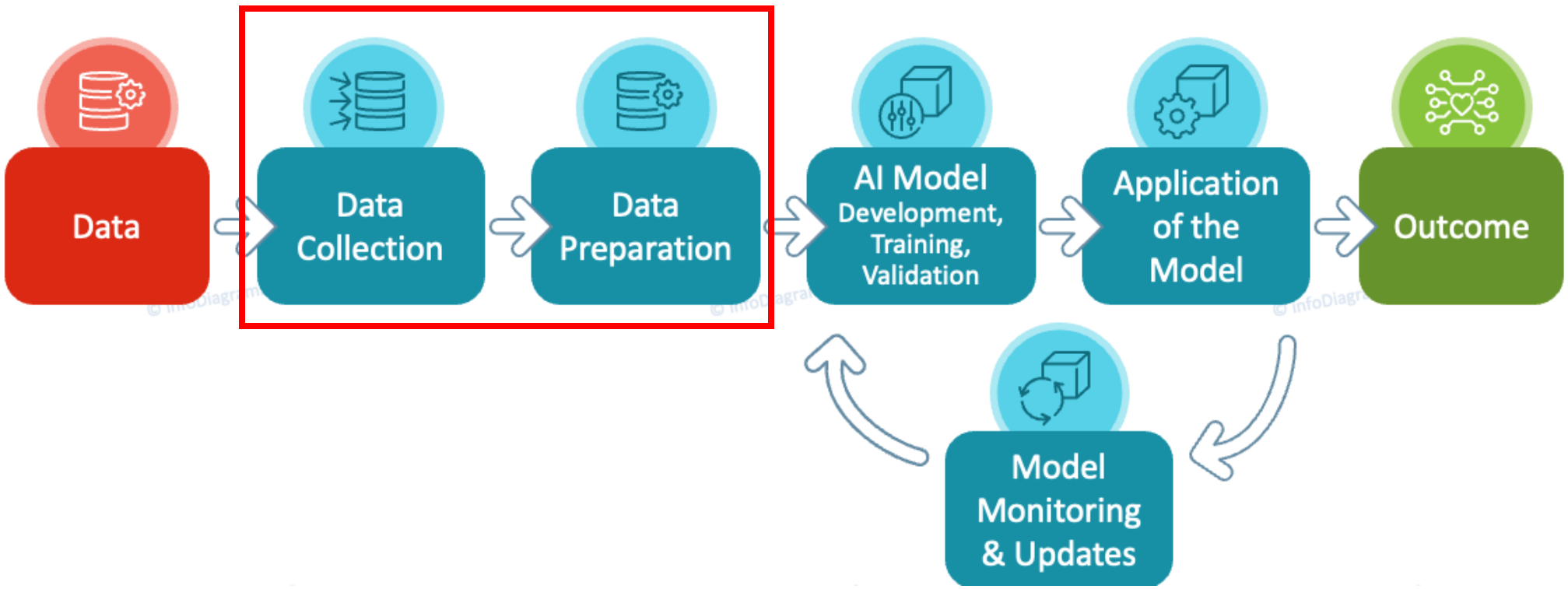
IMEN891K/IDSC501 Data Engineering

14. **Big Data III**: BigQuery & Airflow

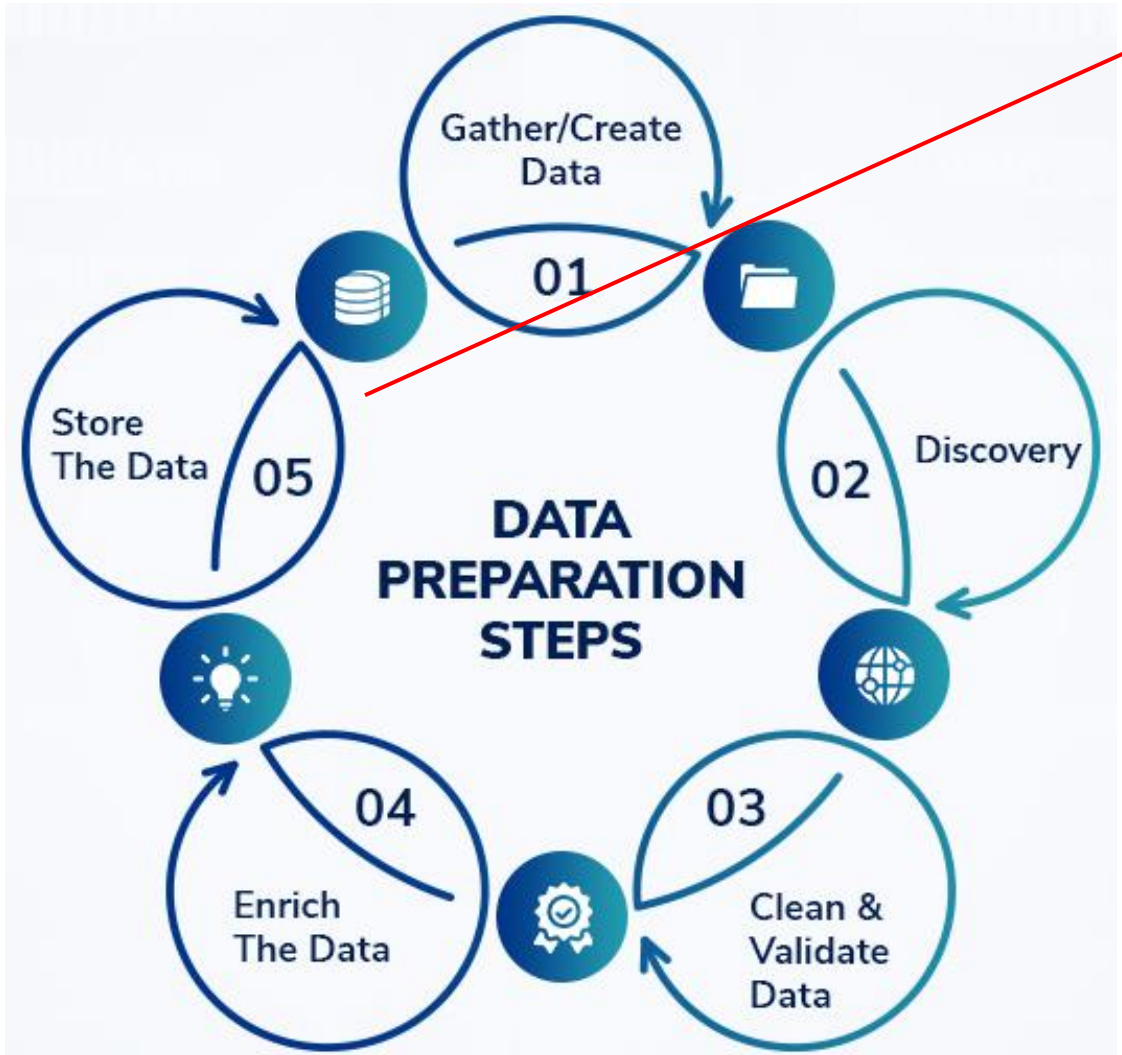
Fall 2025

Instructor: **Sangwoo Mo**

Recap: Course Overview

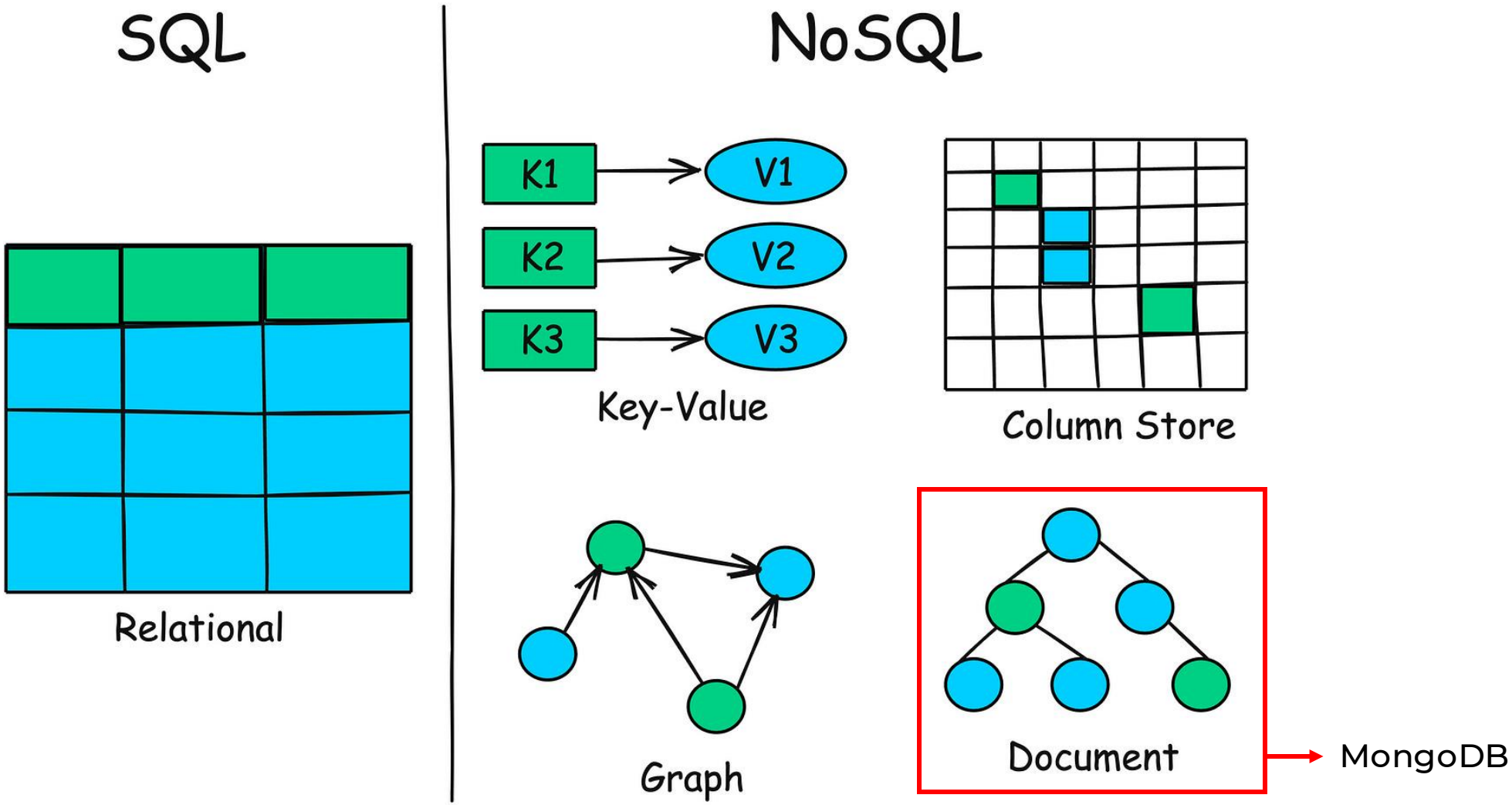


Recap: Course Overview



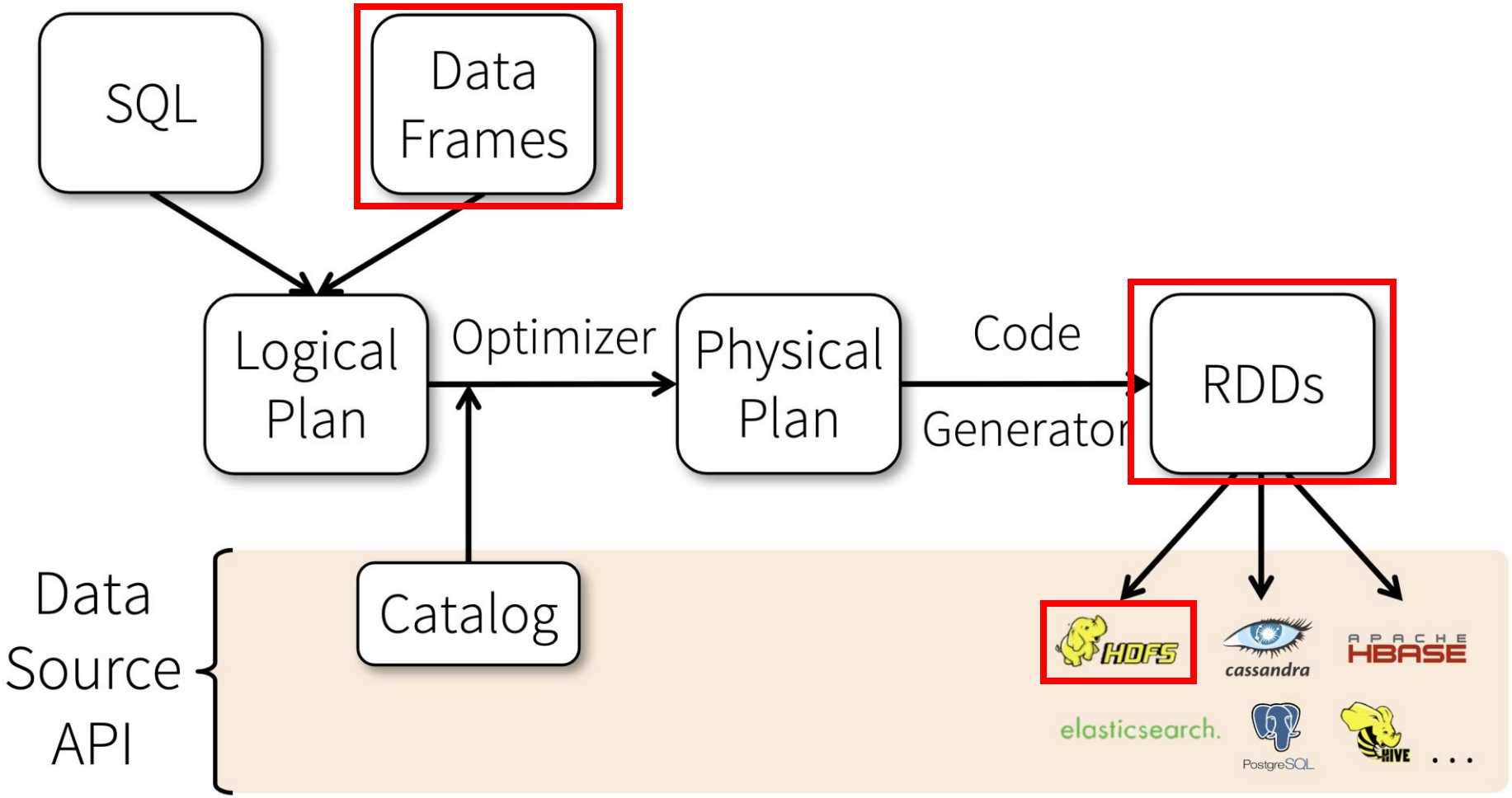
Lecture 7-15

Recap: Database Systems

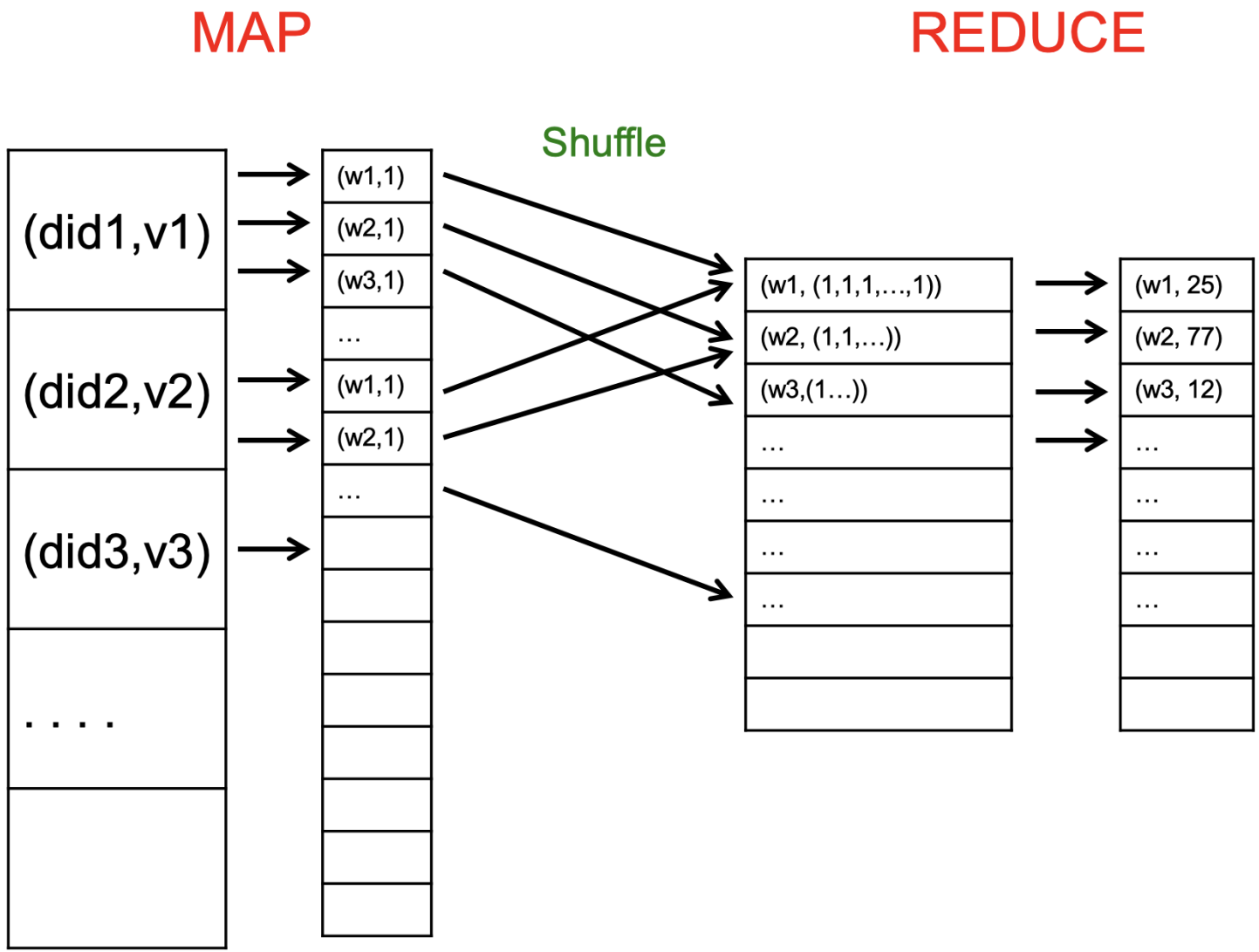


https://blog.algomaster.io/p/sql-vs-nosql-7-key-differences

Recap: Large-scale Data Processing



Recap: Programming in MapReduce



Recap: Programming in MapReduce

- Counting the number of occurrences of each word in a large collection of documents
- Each Document
  - The key = document id (did)
  - The value = set of words (word)

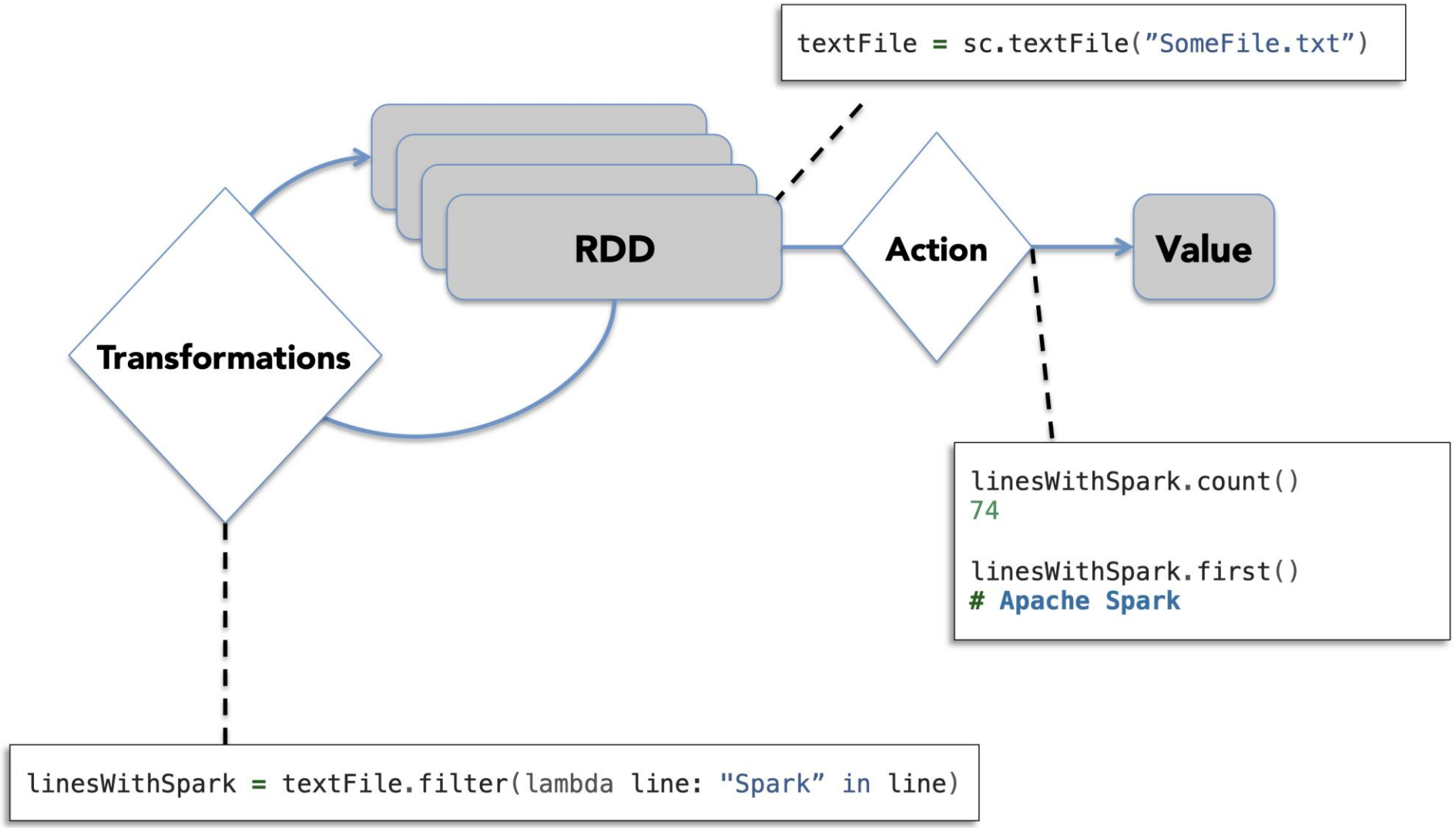
```
map(String key, String value):
   // key: document name
   // value: document contents
   for each word w in value:
        emitIntermediate(w, “1”);
```

```
reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
    for each v in values:
      result += ParseInt(v);
  emit(AsString(result));
```

Recap: Programming in RDD

- A Spark program consists of:
  - Transformations (map, reduceByKey, join…). Lazy
  - Actions (count, reduce, save...). Eager
- Eager: operators are executed immediately
- Lazy: operators are not executed immediately
  - A *operator tree* is constructed in memory instead
  - Similar to a relational algebra tree

Recap: Programming in RDD



Recap: Programming in RDD

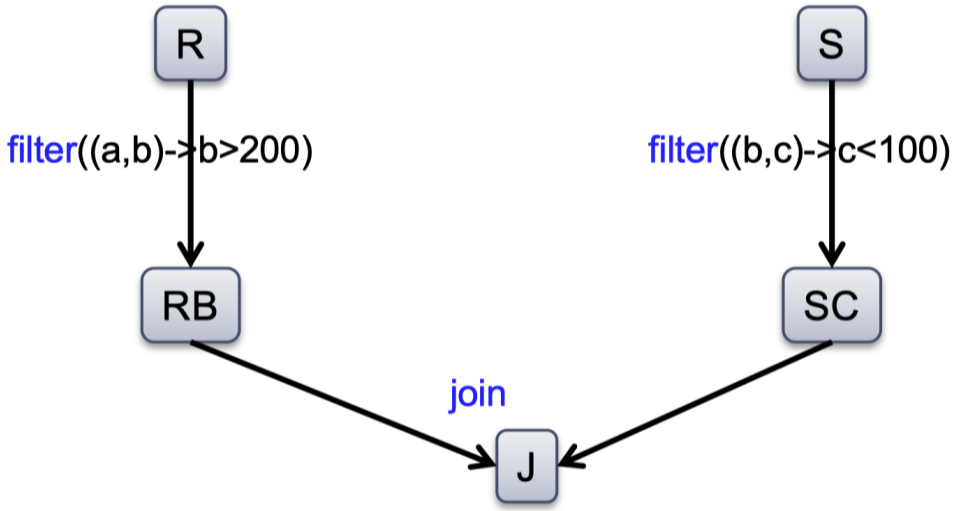
R(A,B)
S(A,C)

```
SELECT count(*)  FROM R, S
WHERE R.B > 200 and S.C < 100  and R.A = S.A
```

```
R = s.read().textFile("R.csv").map(parseRecord).persist();
S = s.read().textFile("S.csv").map(parseRecord).persist();
RB = R.filter(t -> t.b > 200).persist();
SC = S.filter(t -> t.c < 100).persist();
J = RB.join(SC).persist();
J.count();
```

transformations

action



Recap: Programming in DataFrame

Distributed collection of data grouped into named columns (i.e. RDD with schema)

DSL designed for common tasks

- Metadata
- Sampling
- Project, filter, aggregation, join, …
- UDFs

Available in Python, Scala, Java, and R (via SparkR)

Recap: Programming in DataFrame

DataFrames hold rows with a known schema and offer relational operations on them through a DSL

```
val users = spark.sql("select * from users")

val massUsers = users('country === "Canada")

massUsers.count()

massUsers.groupBy("name").avg("age")
```

Expression AST

Motivation

usenix ASSOCIATION Conferences Sign In

SRECON AMERICAS ATTEND PROGRAM PARTICIPATE SPONSORS ABOUT

Why Are Distributed Systems So Hard? → You are NOT alone!

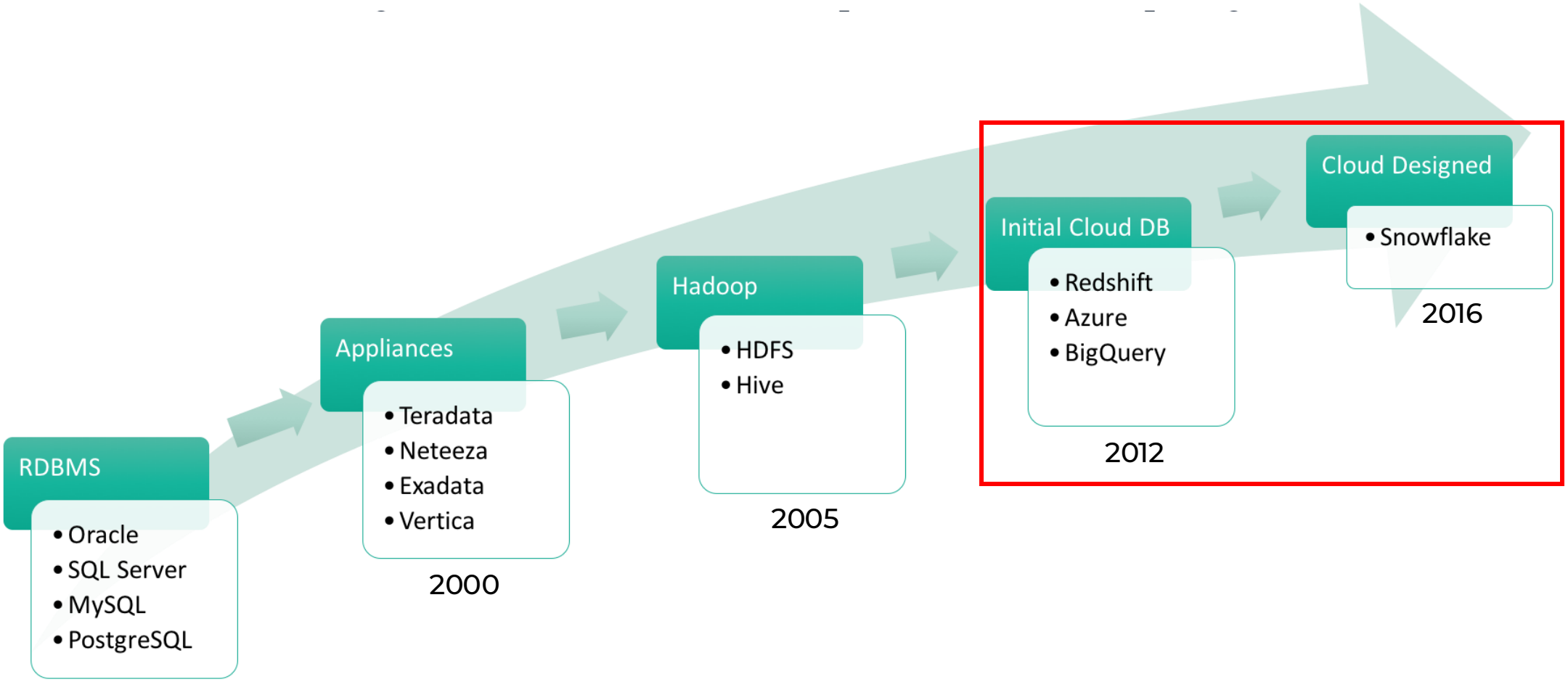
Wednesday, March 27, 2019 - 5:00 pm–5:30 pm
Denise Yu, Pivotal

Abstract:
Distributed systems are known for being notoriously difficult to wrangle. But why? This talk will cover a brief history of distributed databases, clear up some common myths about the CAP theorem, dig into why network partitions are inevitable, and close out by highlighting how a few popular consensus algorithms mitigate the risks of operating in a distributed fashion and the importance of considering human factors to fully understand the systems we build. Almost all slides will contain original illustration featuring mischievous cats masquerading as sysadmins. By the end of this talk you will have a better understanding of the design trade-offs involved in architecting for distributed systems, and hopefully, be inspired to start doodling tech concepts!

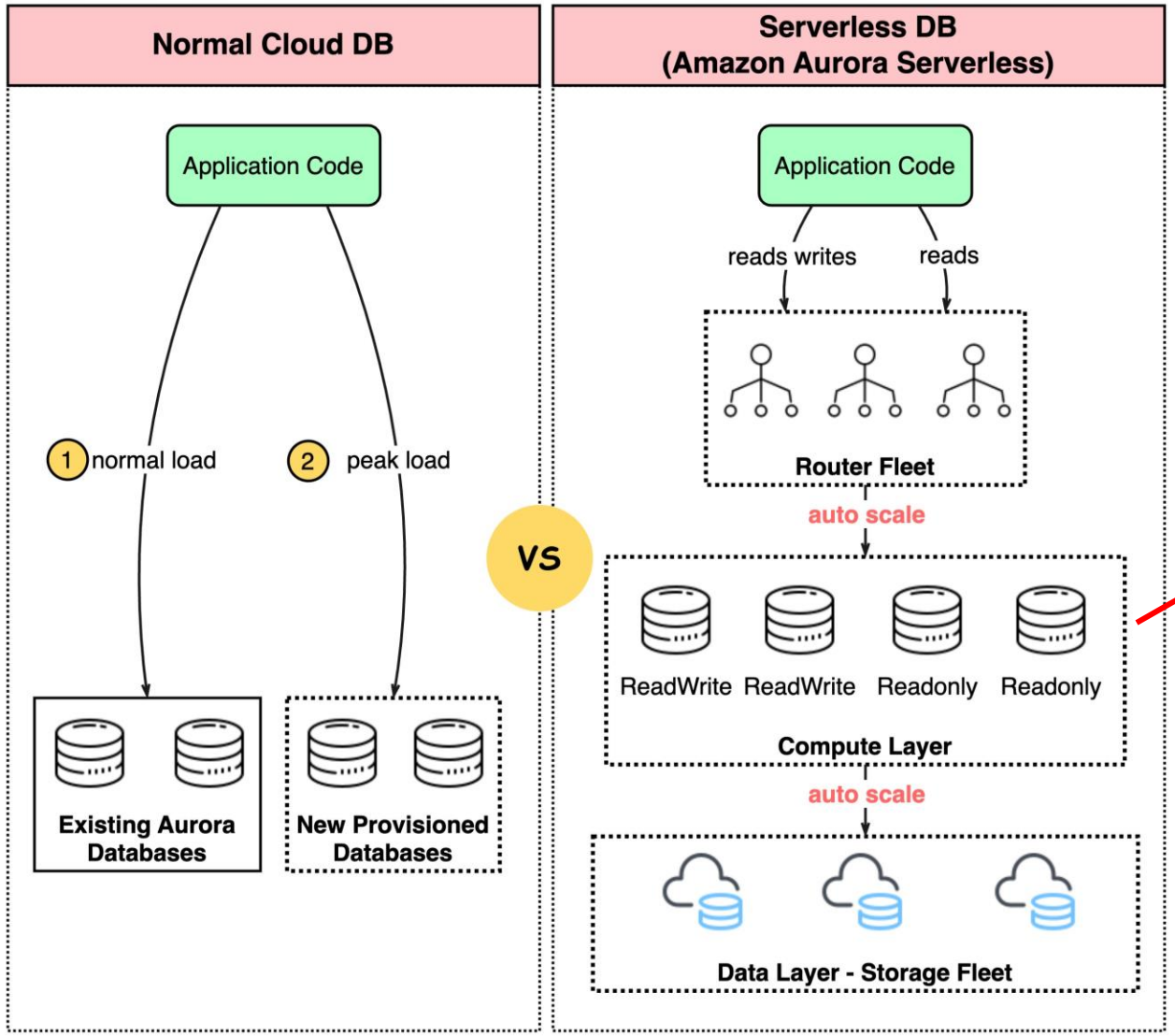


Denise is a software engineer who occasionally wears a product management hat at Pivotal R&D in Toronto. Denise has previously delivered conference talks on topics ranging from continuous delivery to functional programming to scaling company culture. She enjoys learning about distributed systems, release engineering, and low-level Linux kernel programming, and when she's not coding, she is often doodling sketch notes that break down technical concepts into digestible pieces at deniseyu.io/art.

Serverless Computing

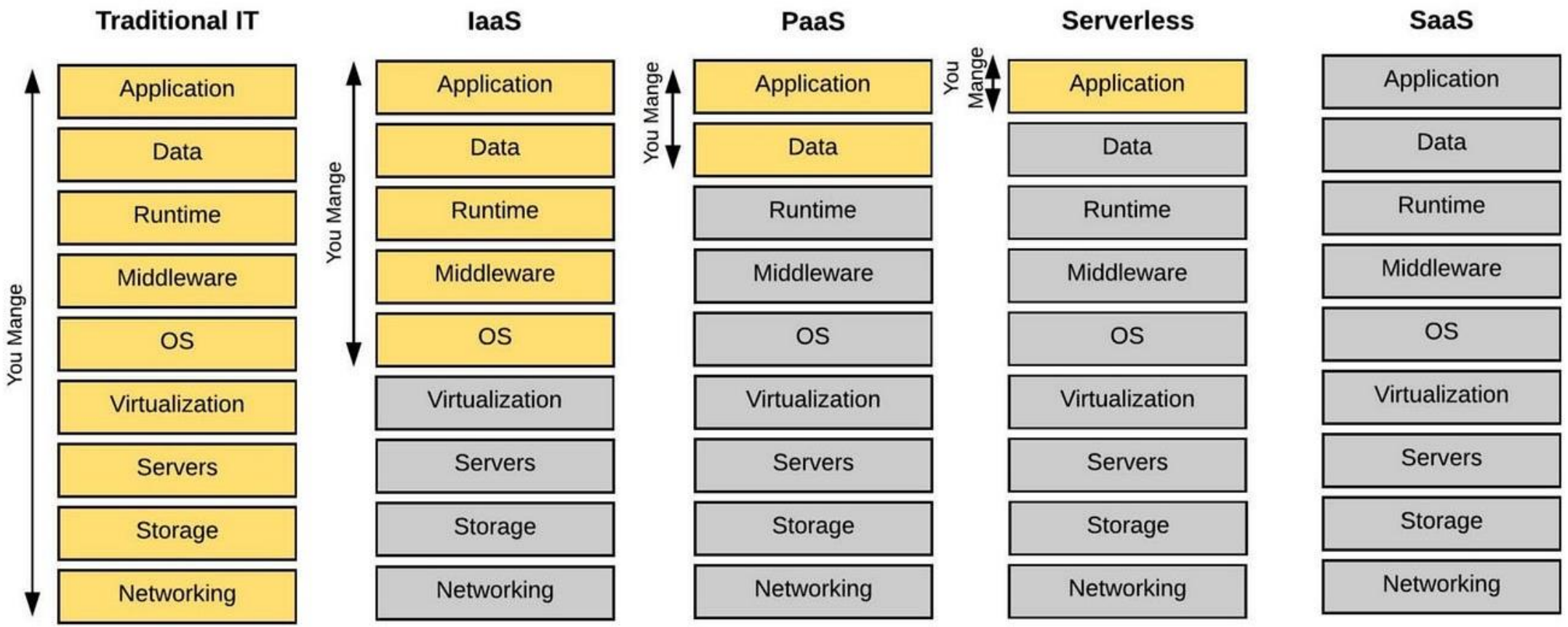


Serverless Computing

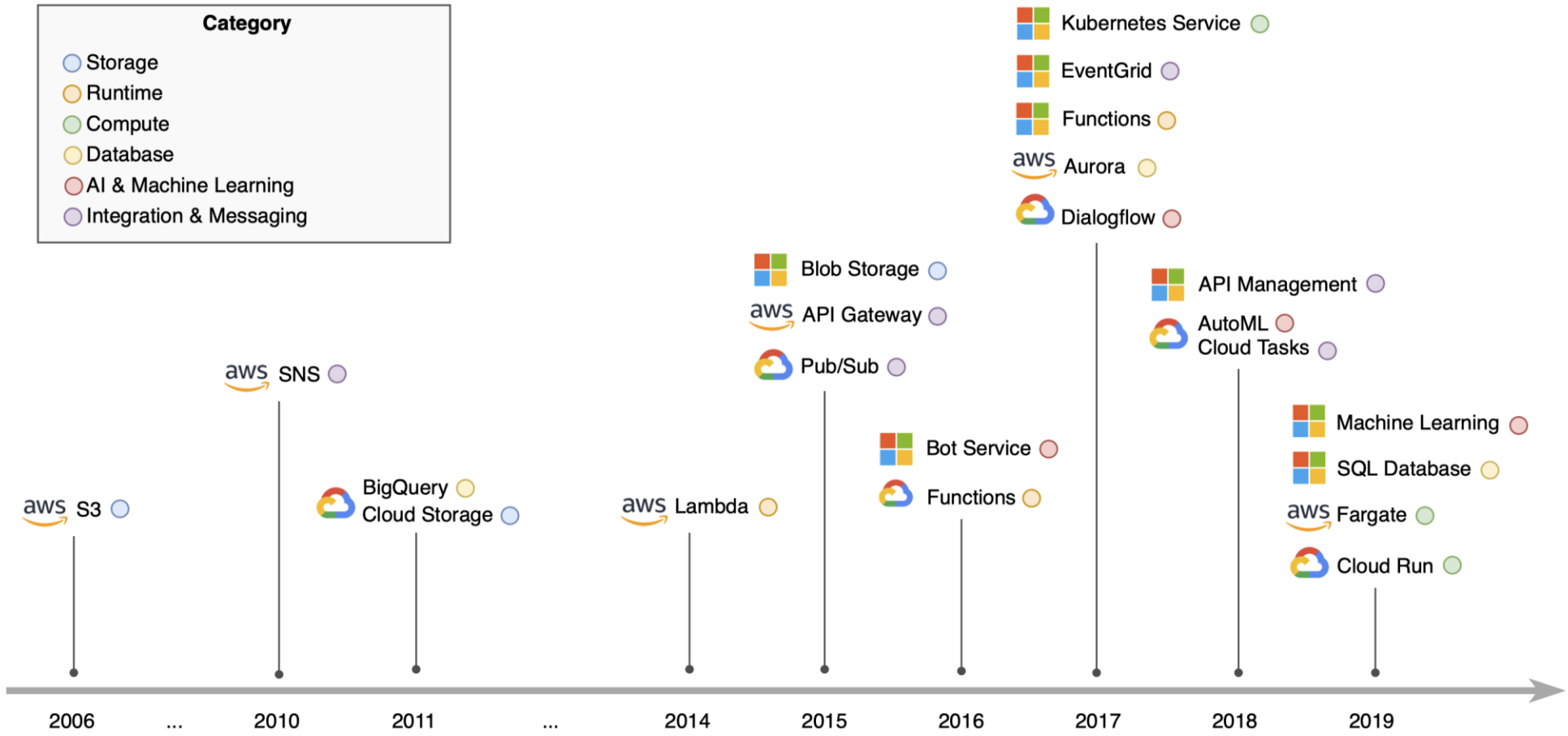


Abstraction:
No need to care about the servers

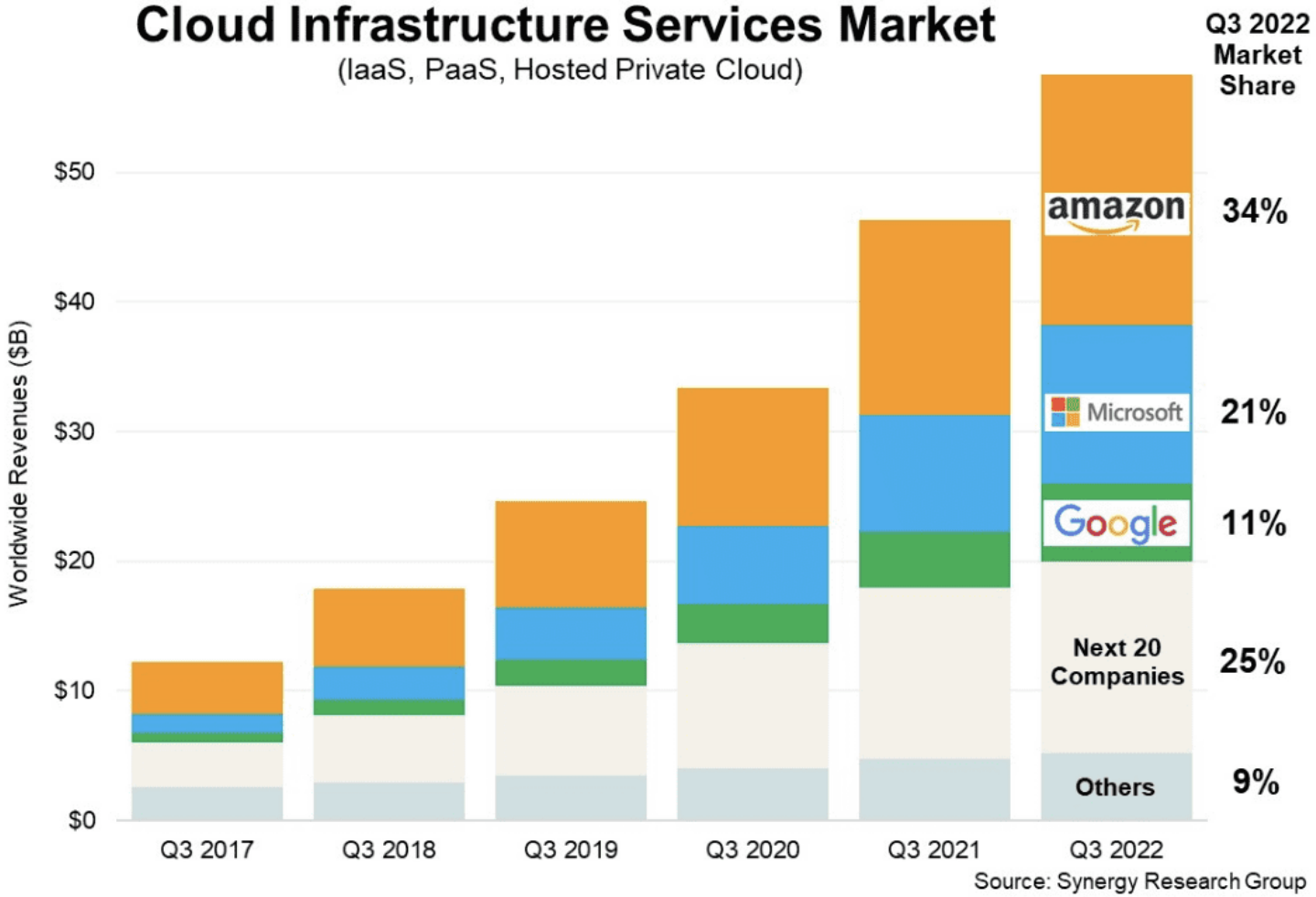
Serverless Computing



Serverless Computing



Serverless Computing



Evolution of Distributed Systems

Generation	Core Concept	Key Innovation	Main Limitation / Motivation for Next Gen	Representative Systems
1️⃣ **MapReduce Era (2000s)**	Disk-based batch processing on large clusters	Parallelize computation across distributed nodes; fault-tolerant via replication	High I/O cost (read/write to disk); complex to program; slow for iterative jobs → need for faster, more expressive frameworks	Hadoop MapReduce, Hive, Pig
2️⃣ **Spark Era (2010s)**	In-memory distributed data processing (DAG execution)	Unified APIs for batch, streaming, and ML; optimized DAG scheduler; efficient caching	Manual cluster setup and tuning; operational overhead remains → desire for fully managed, elastic systems	Apache Spark, Flink, Beam
3️⃣ **Serverless Era (2017→)**	Cloud-native, fully managed distributed analytics	Serverless architecture — automatic scaling, pay-per-query, separation of storage & compute	Limited low-level control; mostly SQL-based → trade-off between abstraction and flexibility	Google BigQuery, Snowflake, AWS Athena, Azure Synapse

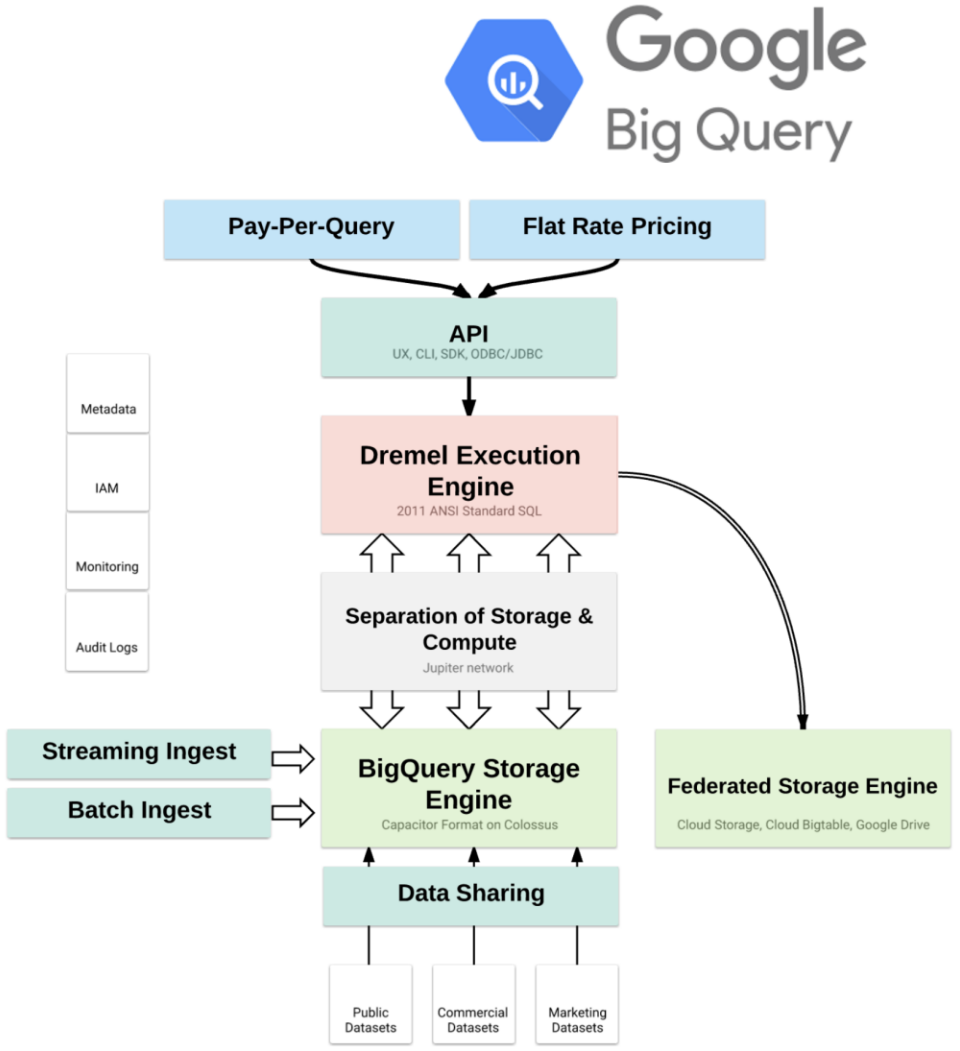
What is BigQuery?

Fully managed Data Warehouse

- “Arbitrarily” large data and queries
- Pay per bytes being processed
- No concept of cluster

Other similar systems

- AWS Athena
- Snowflake
- …



What is BigQuery?



Ingest

- Batch & Streaming
- Structured & Semi-structured



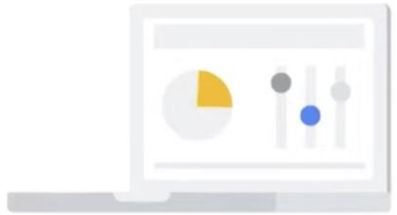
Store

- Up to Petabytes of data
- Datasets & tables
- Federate data



Analyze

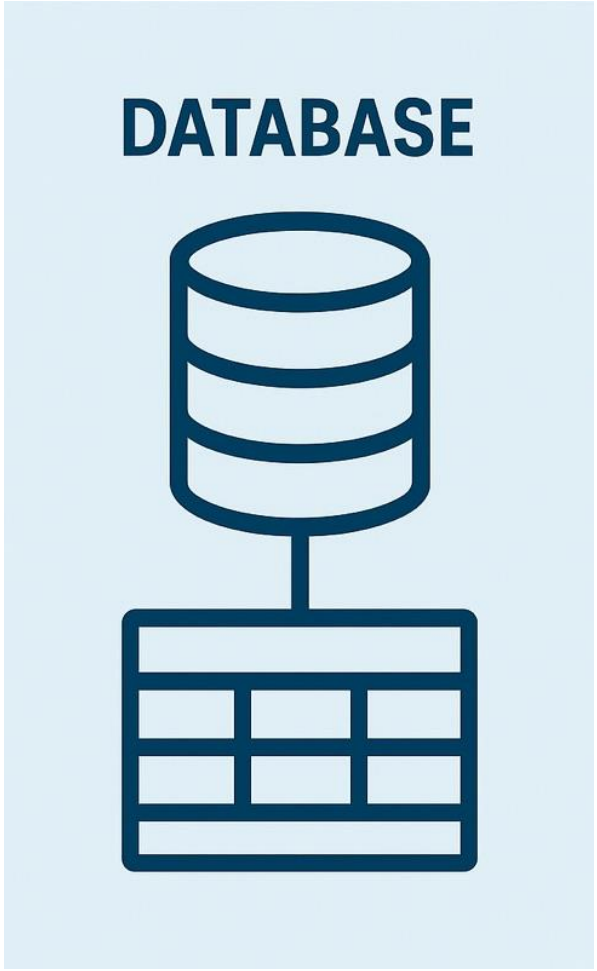
- Super-fast SQL queries
- Aggregate
- Transform with DML



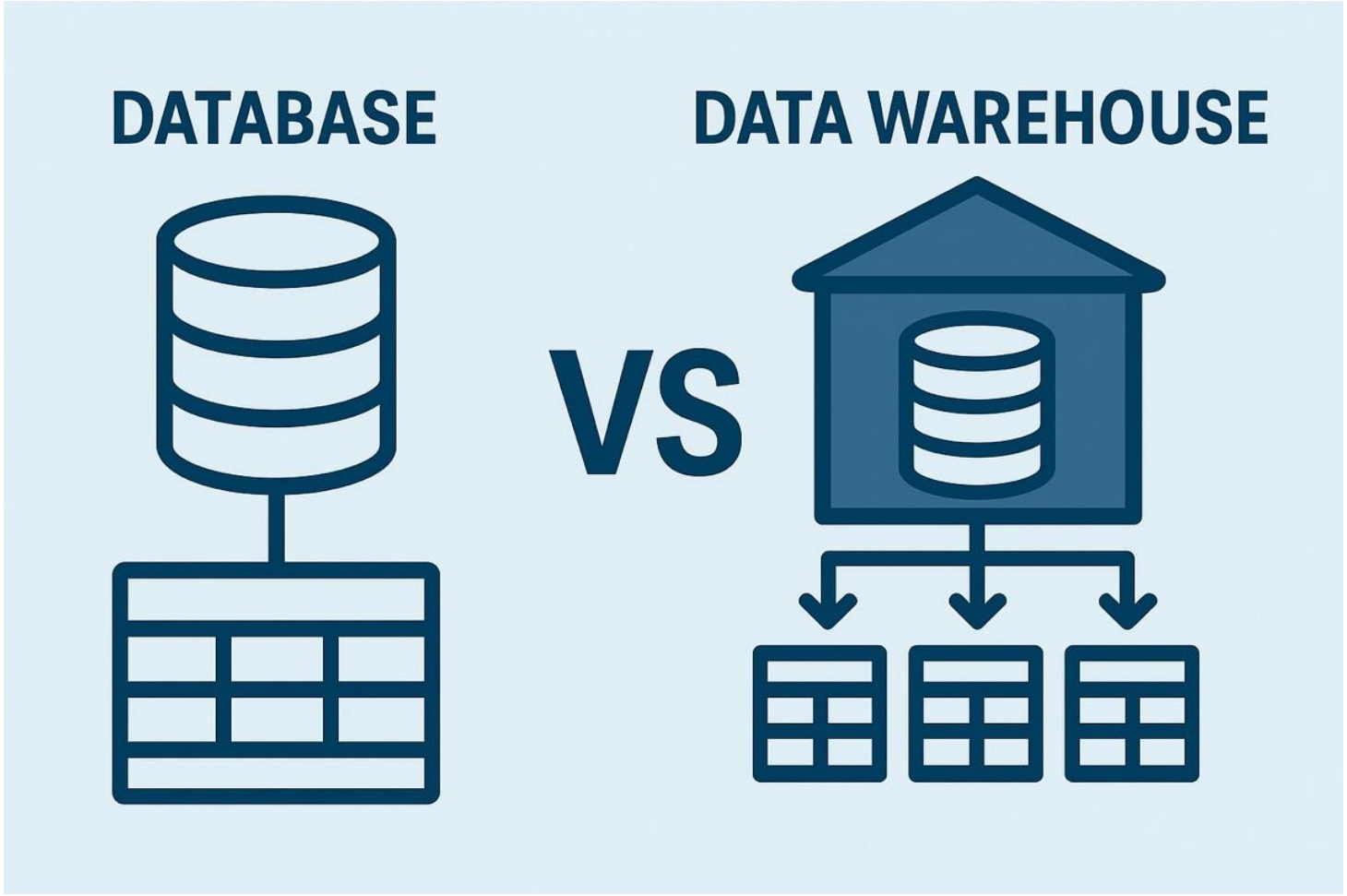
Visualize

- Export and connect to other tools more easily

What is Data Warehouse?

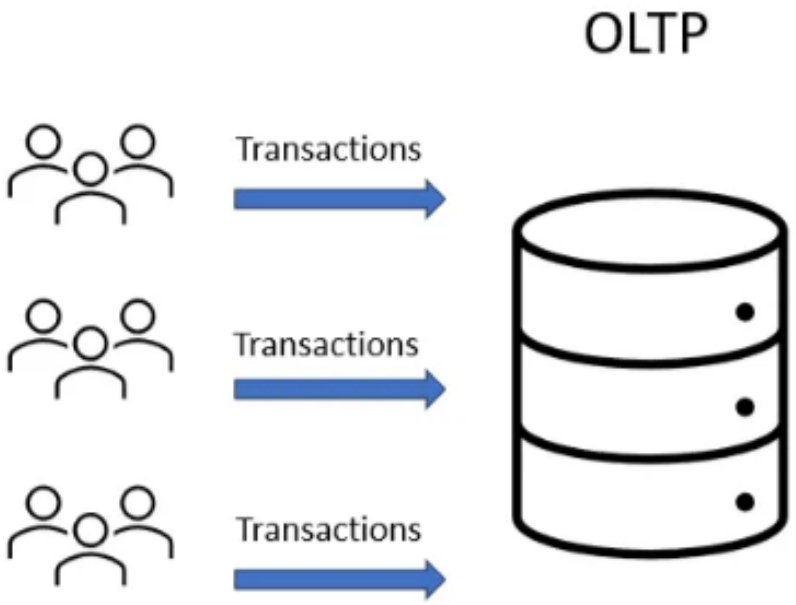


What is Data Warehouse?



What is Data Warehouse?

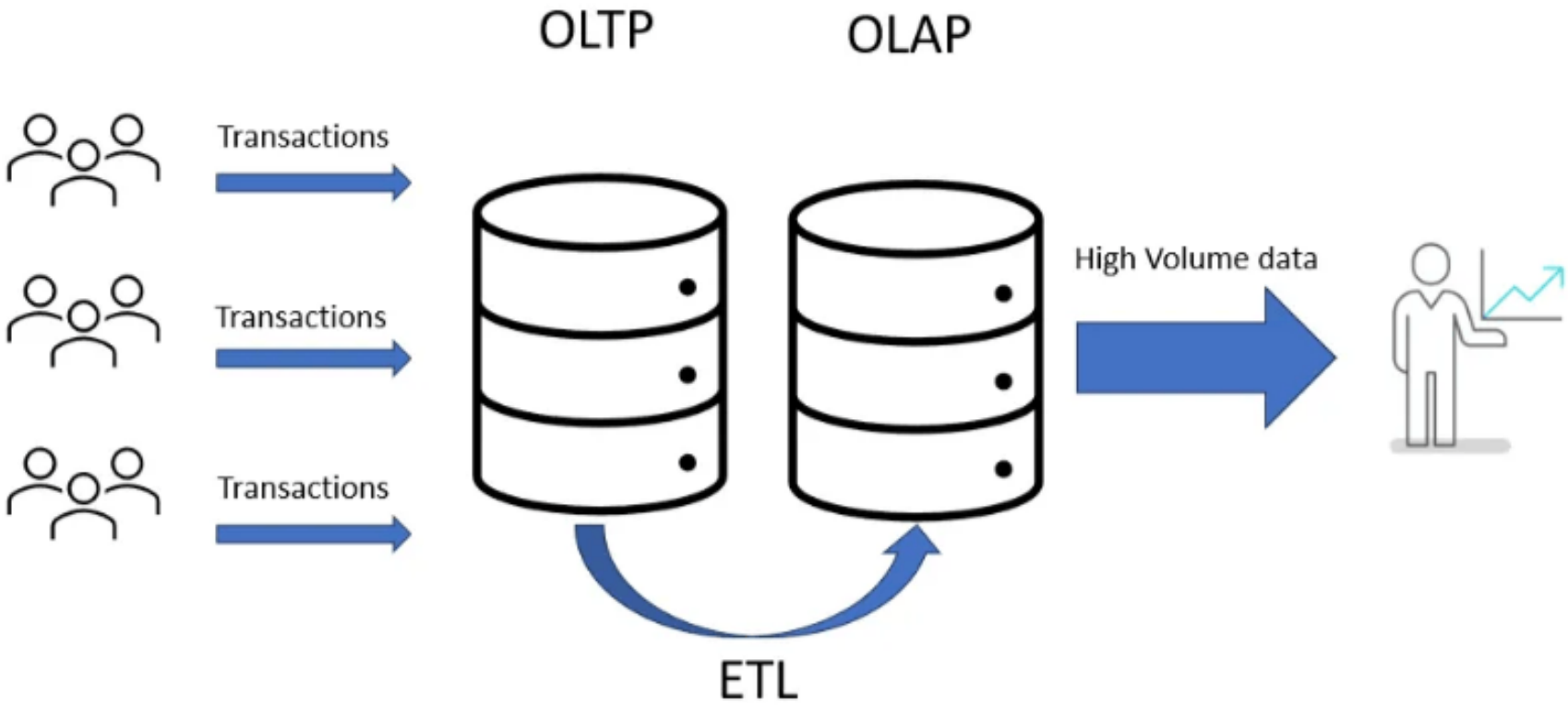
Online **Transaction** Processing
(Database)



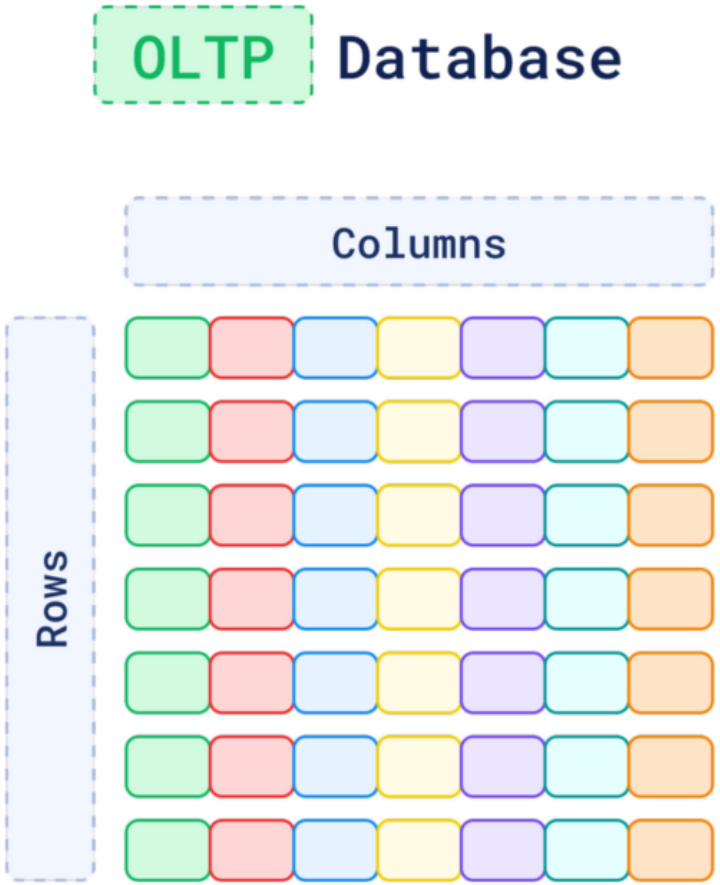
What is Data Warehouse?

Online **Transaction** Processing
(Database)

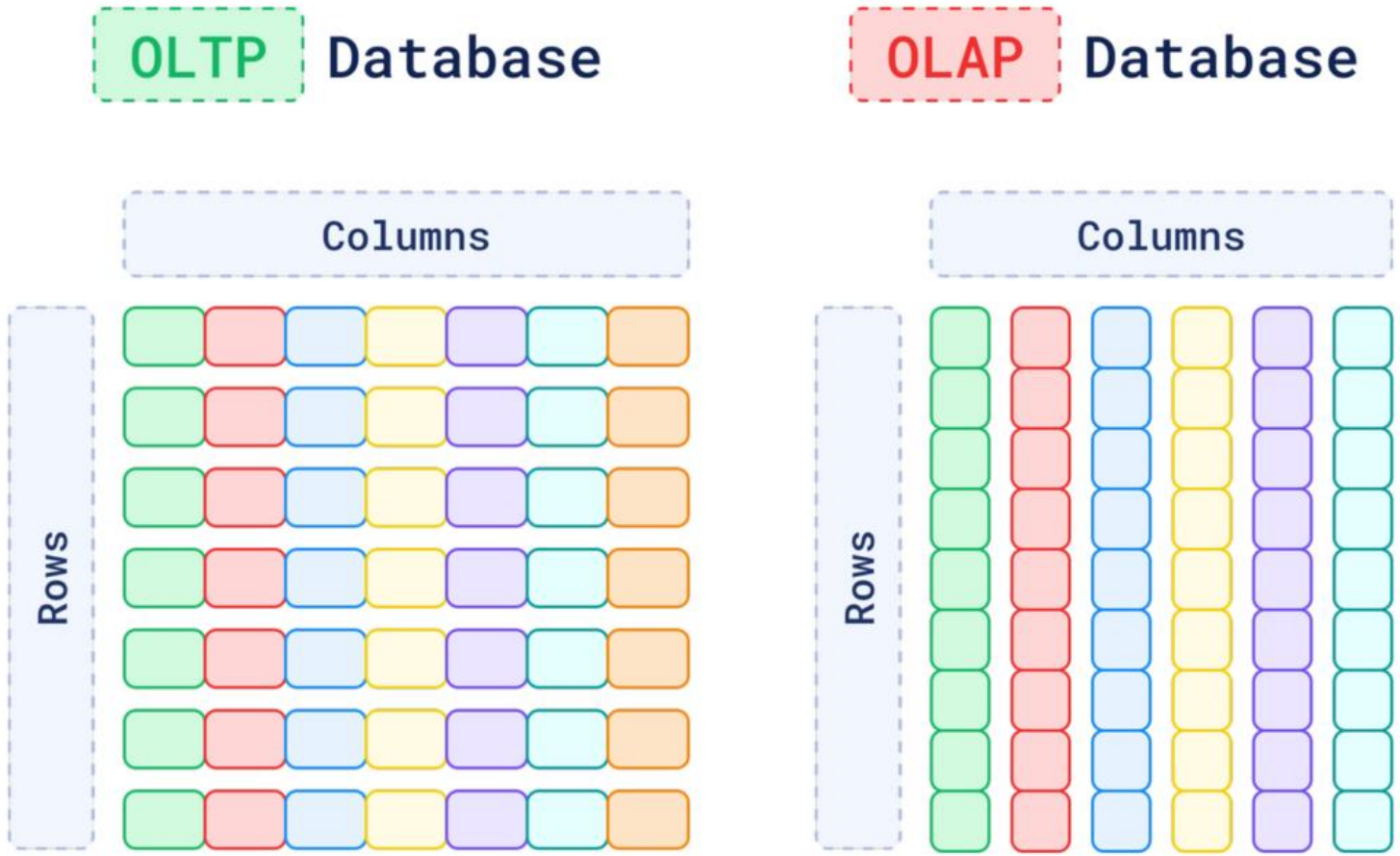
Online **Analytical** Processing
(Data Warehouse)



OLTP vs OLAP



OLTP vs OLAP



OLTP vs OLAP

Table

	Country	Product	Sales
Row 1	India	Chocolate	1000
Row 2	India	Ice-cream	2000
Row 3	Germany	Chocolate	4000
Row 4	US	Noodle	500

Row Store

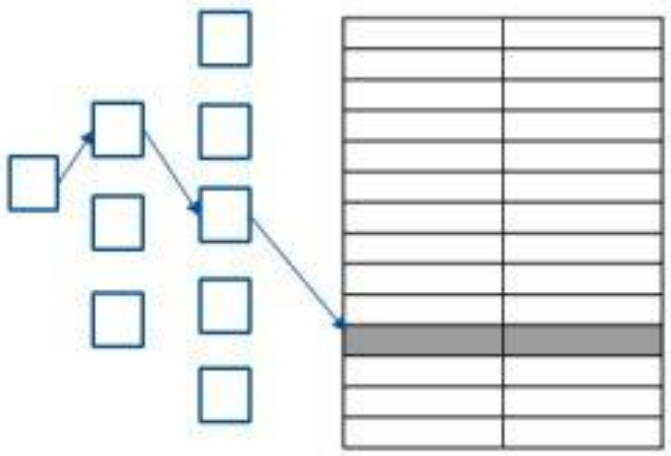
Row 1	India
	Chocolate
	1000
Row 2	India
	Ice-cream
	2000
Row 3	Germany
	Chocolate
	4000
Row 4	US
	Noodle
	500

Column Store

Country	India
	India
	Germany
	US
Product	Chocolate
	Ice-cream
	Chocolate
	Noodle
Sales	1000
	2000
	4000
	500

OLTP vs OLAP

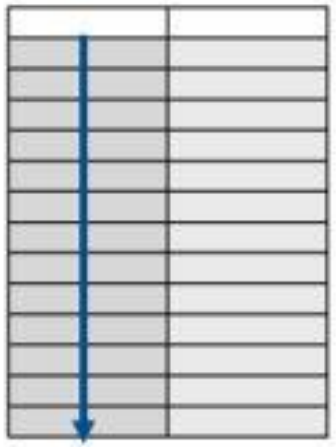
Row-oriented storage & execution



```
SELECT * FROM orders
WHERE orderid = $1
```

At scale: Fast on OLTP, Slow on OLAP

Column-oriented storage & execution



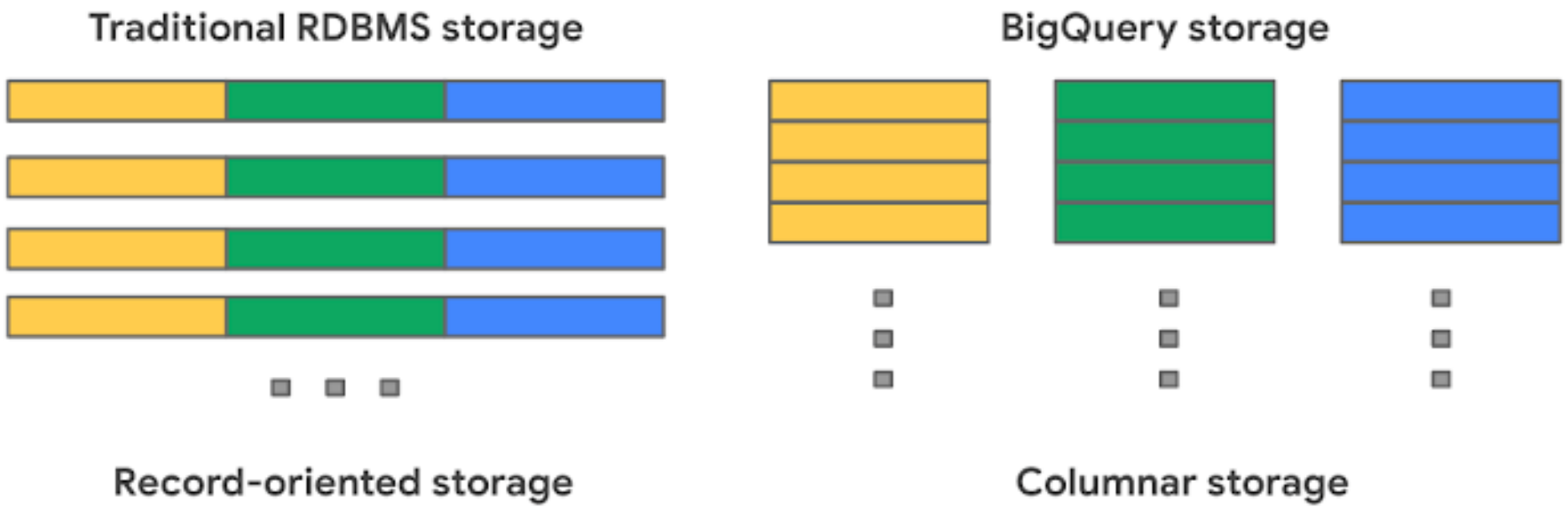
```
SELECT productid, count(*) FROM orders
GROUP BY 1 ORDER BY 2 DESC LIMIT 10
```

At scale: Slow on OLTP, Fast on OLAP

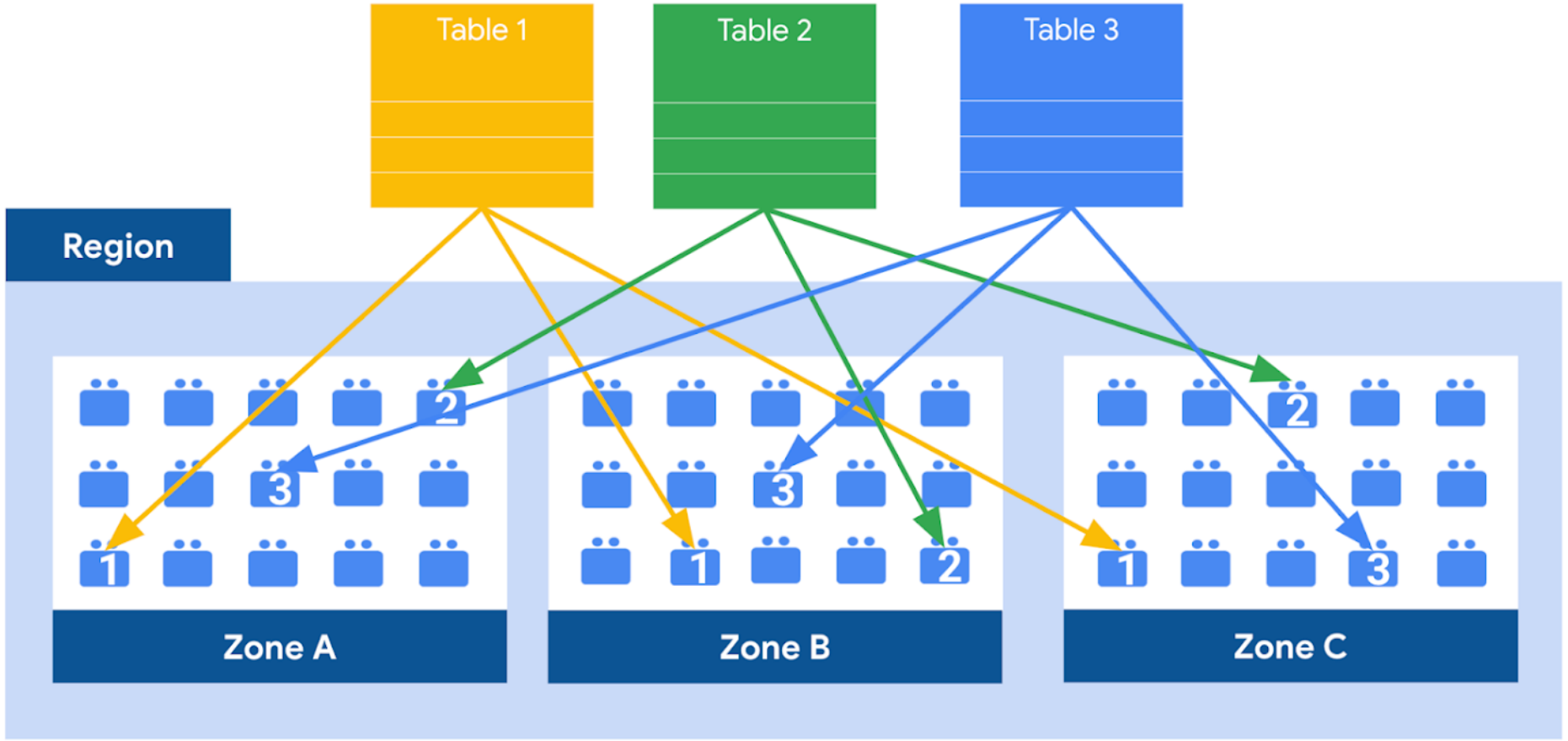
OLTP vs OLAP

Feature	OLTP	OLAP
Full Form	Online Transaction Processing	Online Analytical Processing
Purpose	Manages daily transactions in real time	Analyzes large volumes of historical data
Operations	Frequent data modifications: Insert, Update, Delete	Complex queries: SELECT, Aggregations
Storage	Row-based (stores complete records)	Column-based (stores by field/column)
Optimization	Write-optimized	Read-optimized
Response Time	Milliseconds ⚡	Seconds or Minutes ⏳
Systems	Databases: MySQL, PostgreSQL, Oracle, SQL Server	Data Warehouses: Snowflake, Redshift, BigQuery, Azure Synapse
Real-world Usage	Banking 💳, E-commerce 🛒, Apps 📱	Reports 📈, Dashboards 📊, Business Insights

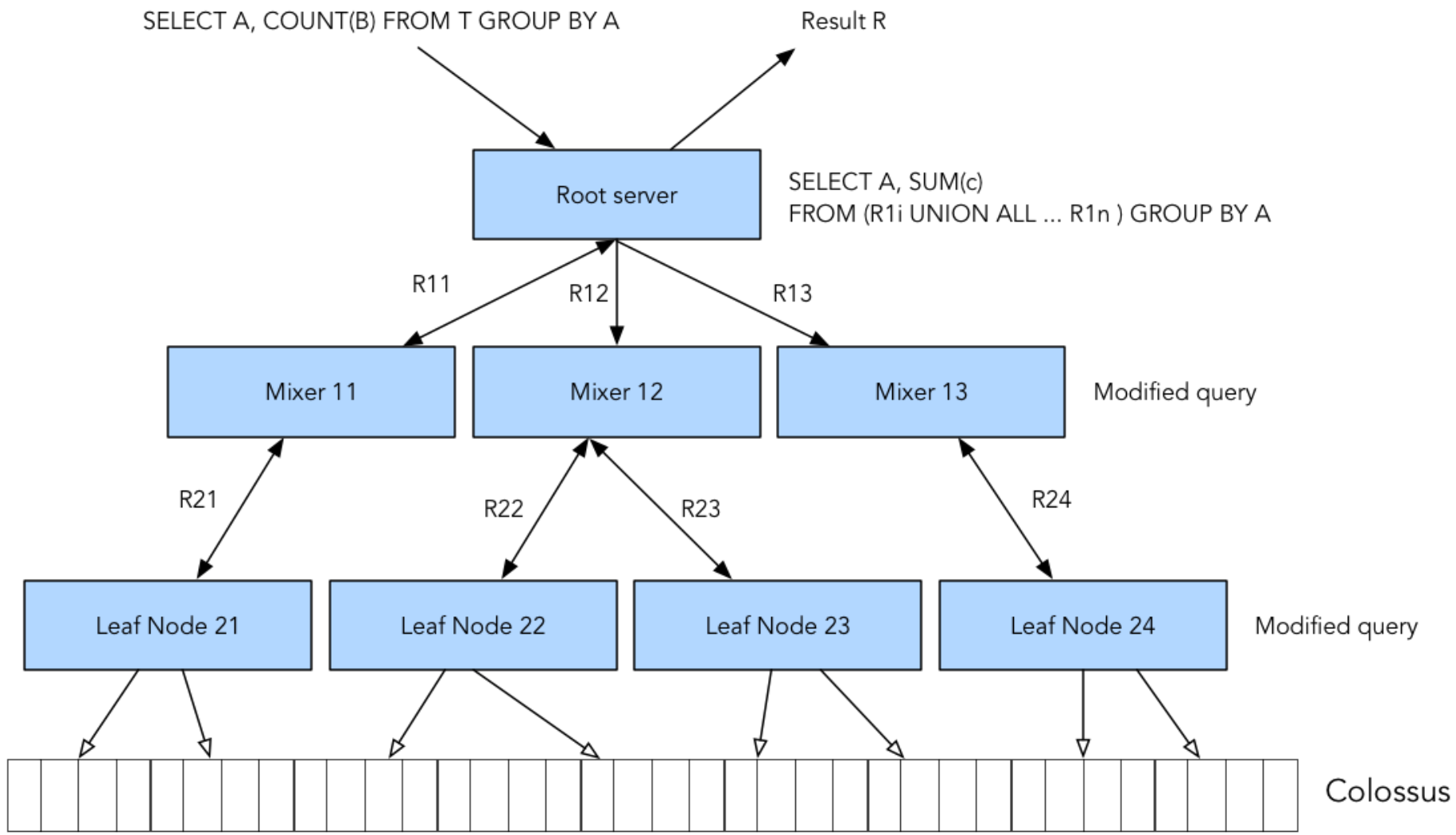
BigQuery = Serverless Data Warehouse



BigQuery Internals (Fault Tolerance)



BigQuery Internals (Dremel Engine)



Programming in BigQuery

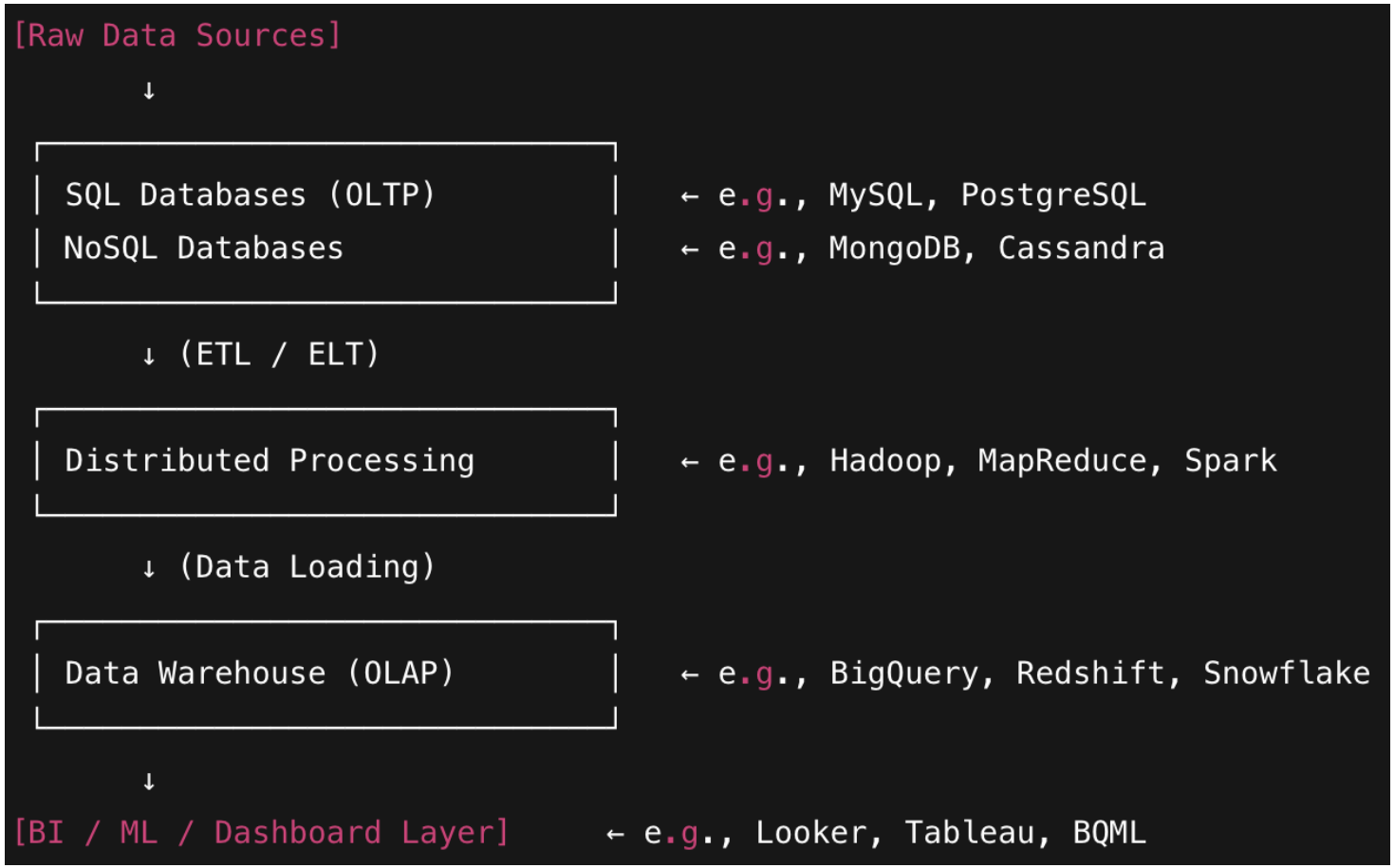
Programming in BigQuery = SQL

- Basic form (there are many many more bells and whistles)

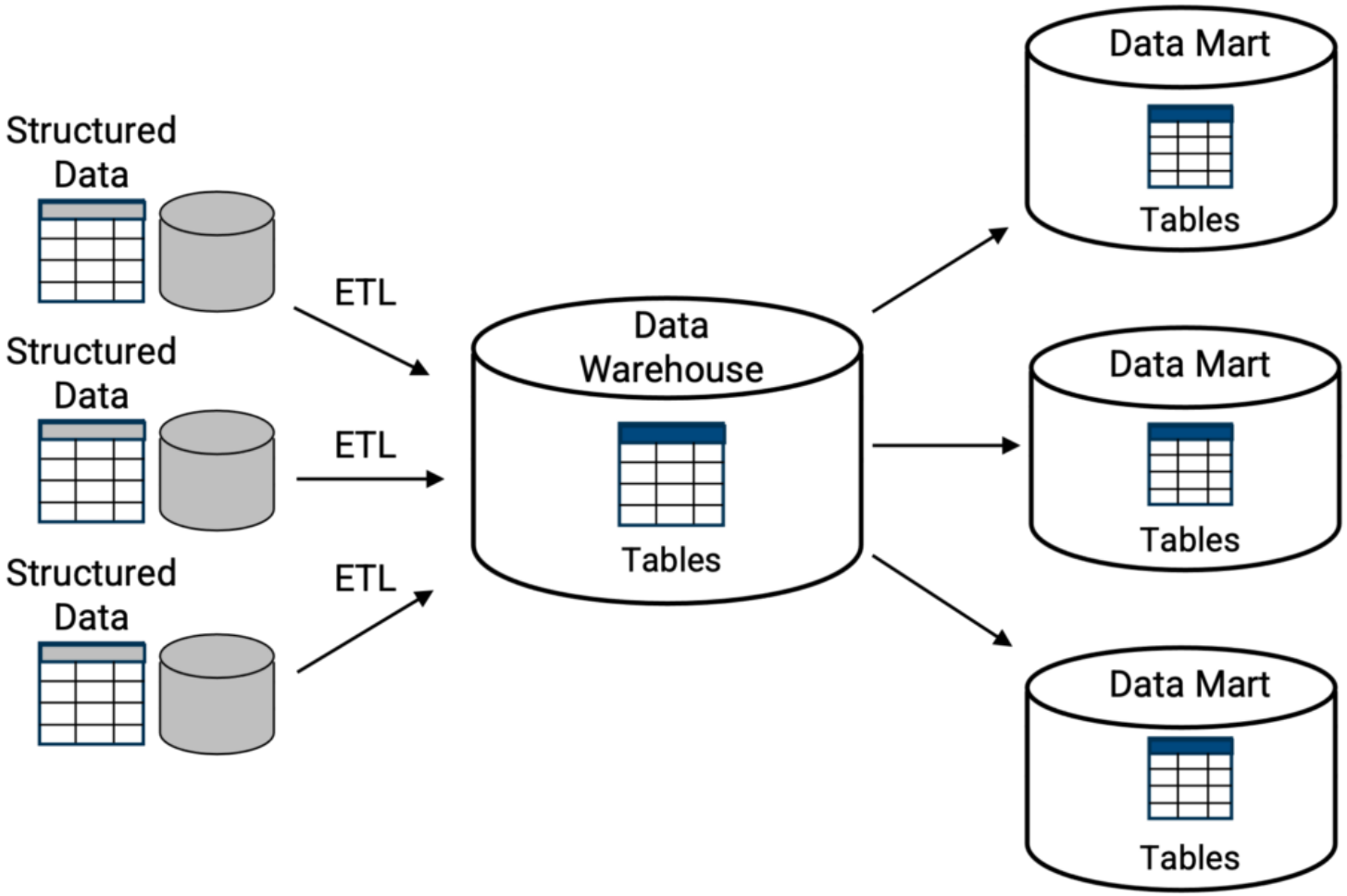
```
SELECT <attributes>
FROM   <one or more relations>
WHERE  <conditions>
```

Call this a SFW query.

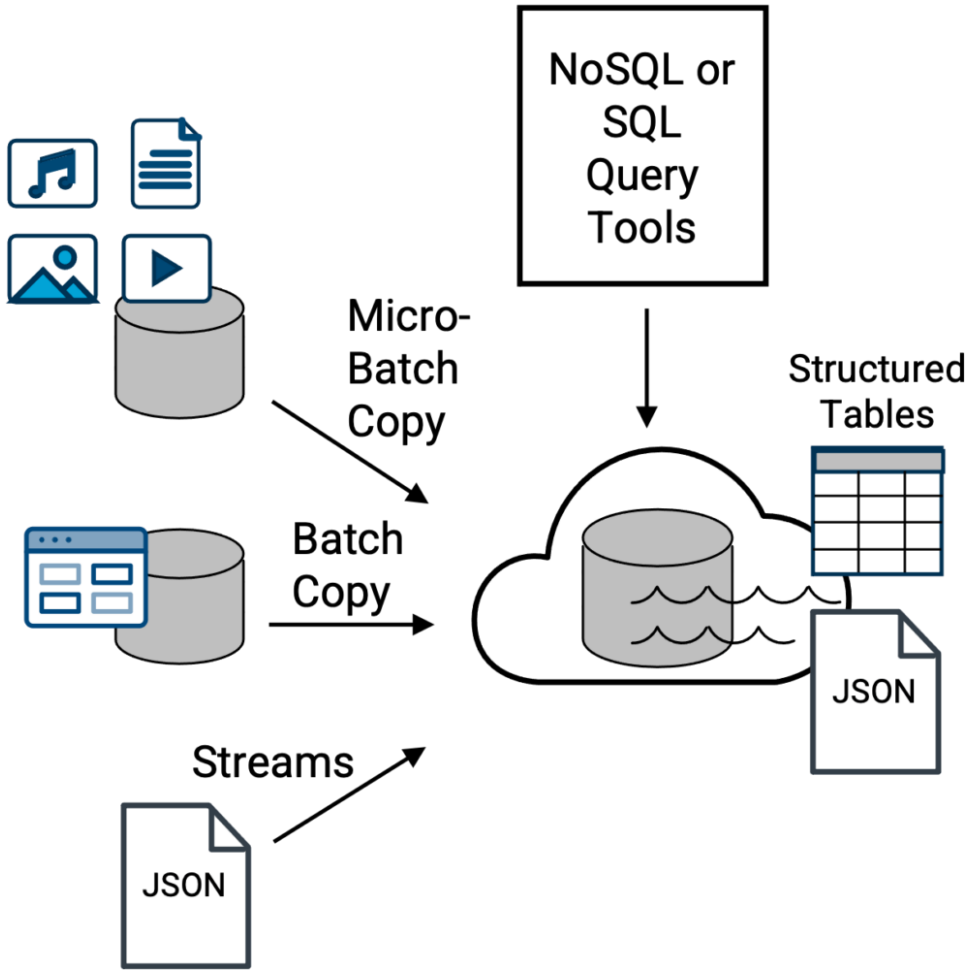
Summary: What We've Learned



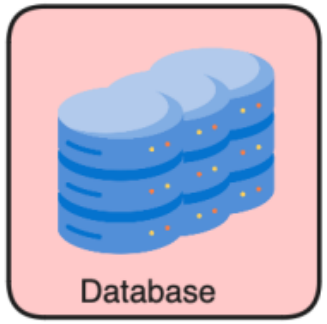
Recap: Data Warehouse



Data Lake


NoSQL or
SQL
Query
Tools
Micro-
Batch
Copy
Batch
Copy
Streams
JSON
Structured
Tables
JSON

Difference Between Data Storages



VS



VS

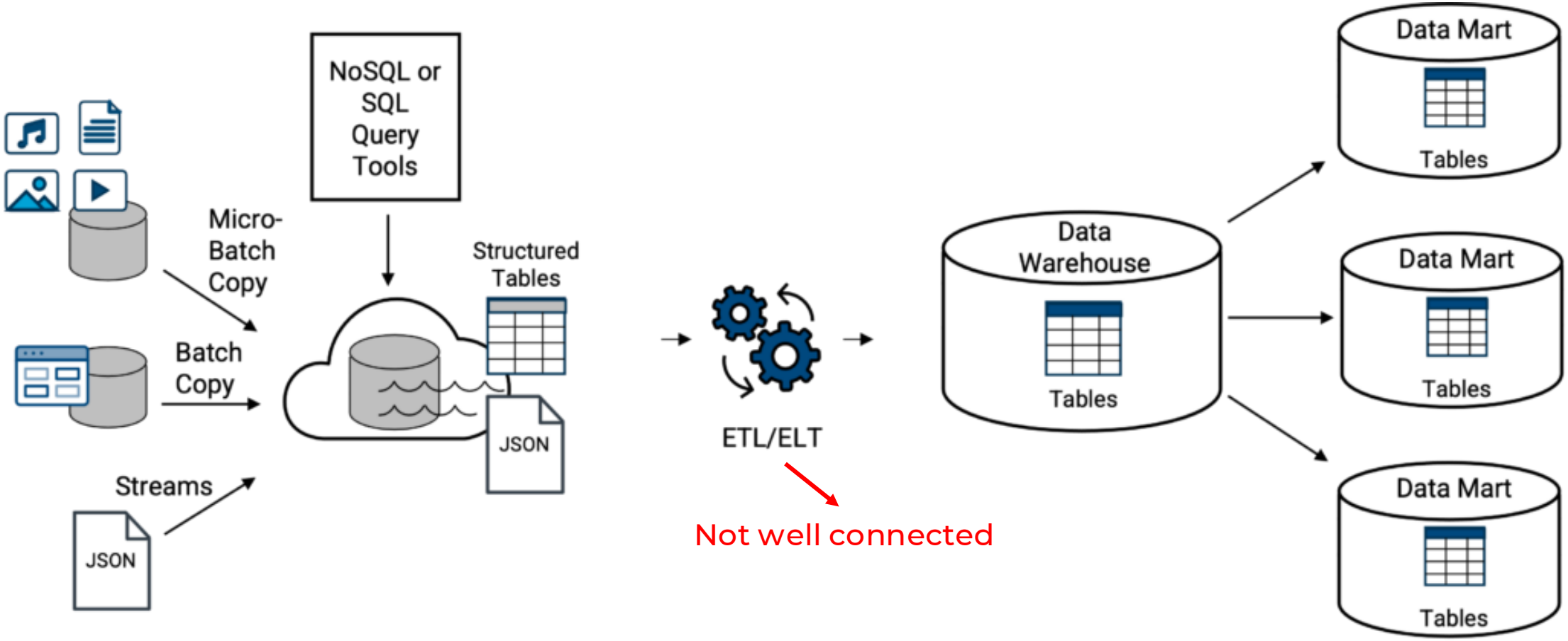


VS

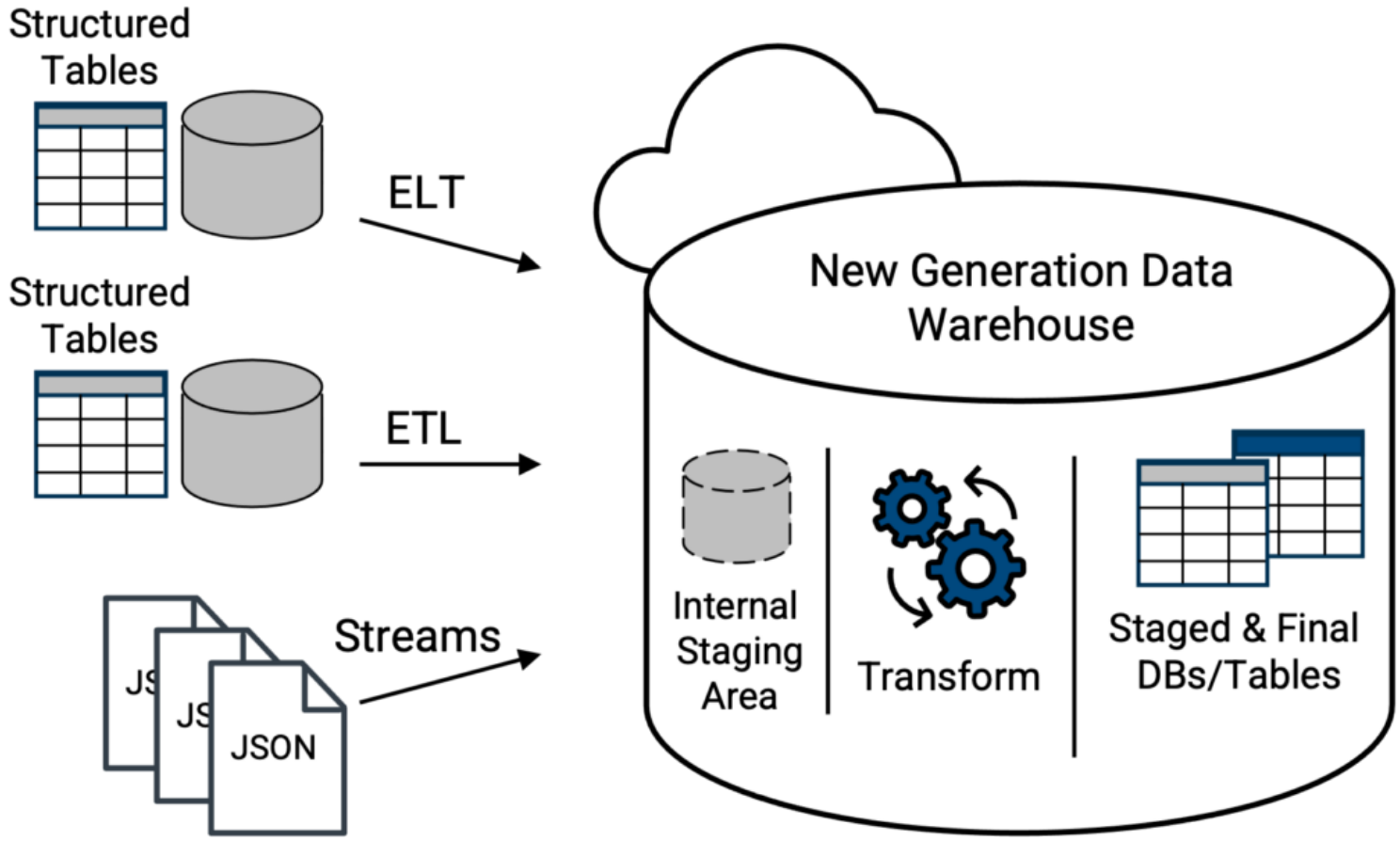


	Database	Data Warehouse	Data mart	Data lake
Scope	Application-specific	Organization-wide, structured data.	Department-specific, structured data.	Organization-wide, any type of data
Data Type	Structured	Structured	Structured	Structured, semi-structured, unstructured.
Structure	Predefined schema	Schema on write	Schema on write (inherited from data warehouse)	Schema on read
Use Case	Operational applications(OLTP)	Business intelligence, historical analysis(OLAP).	Specific business function analysis	Big data analytics, data exploration.

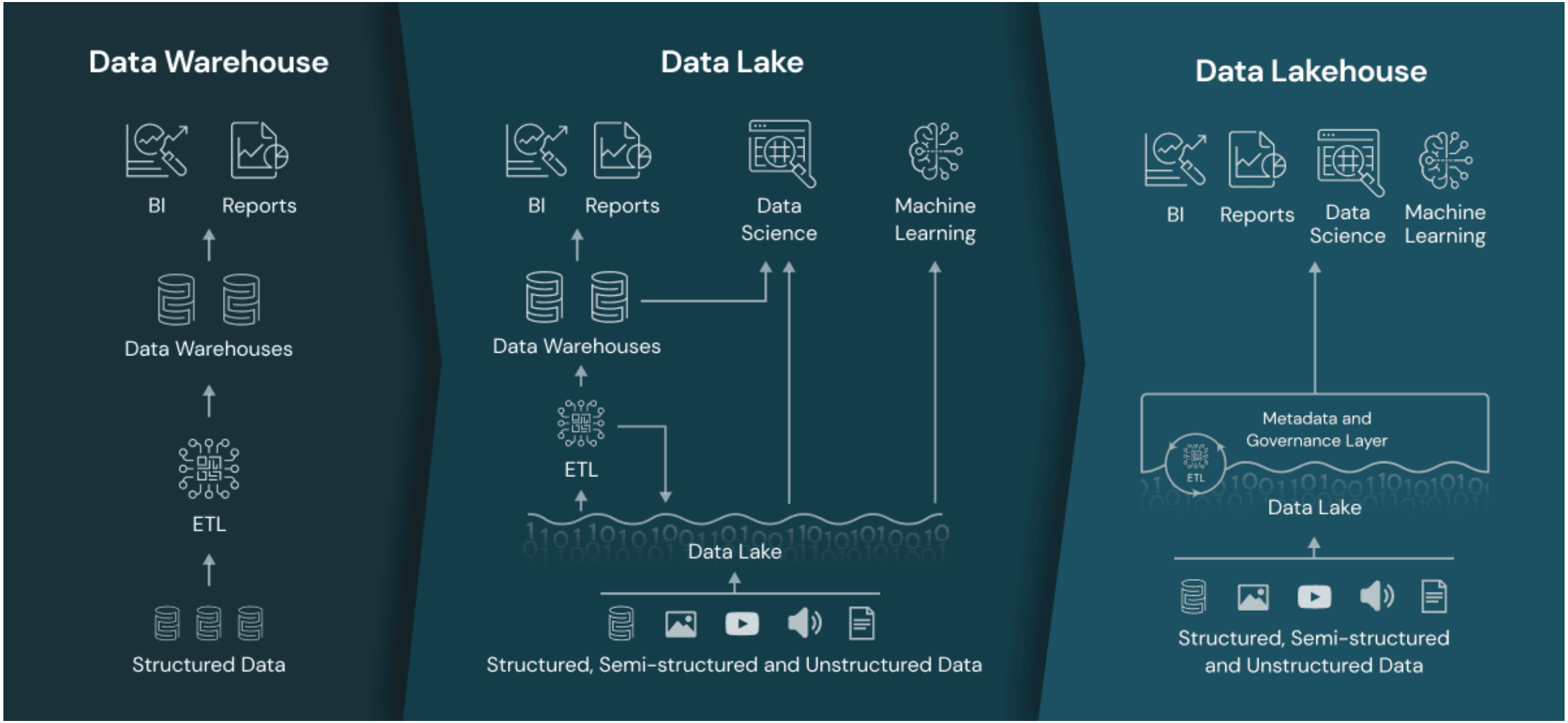
Data Lake → Warehouse



Data Lakehouse



Data Lakehouse

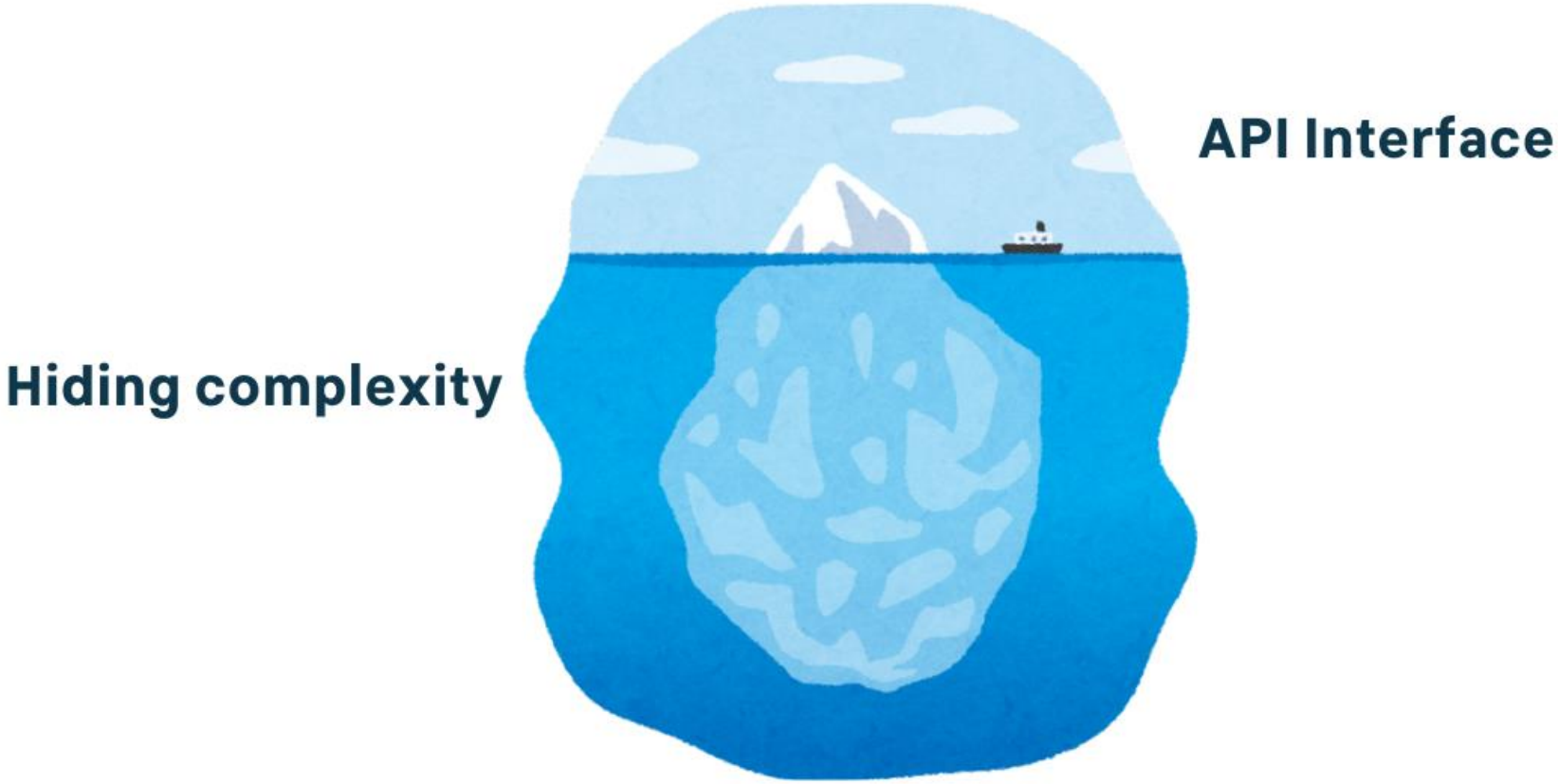


Lakehouse Internals

1. Storage layer: add transactions, versioning & more
2. New query engine designs: great SQL performance on data lake storage systems and file formats
3. Optimized access for data science & ML

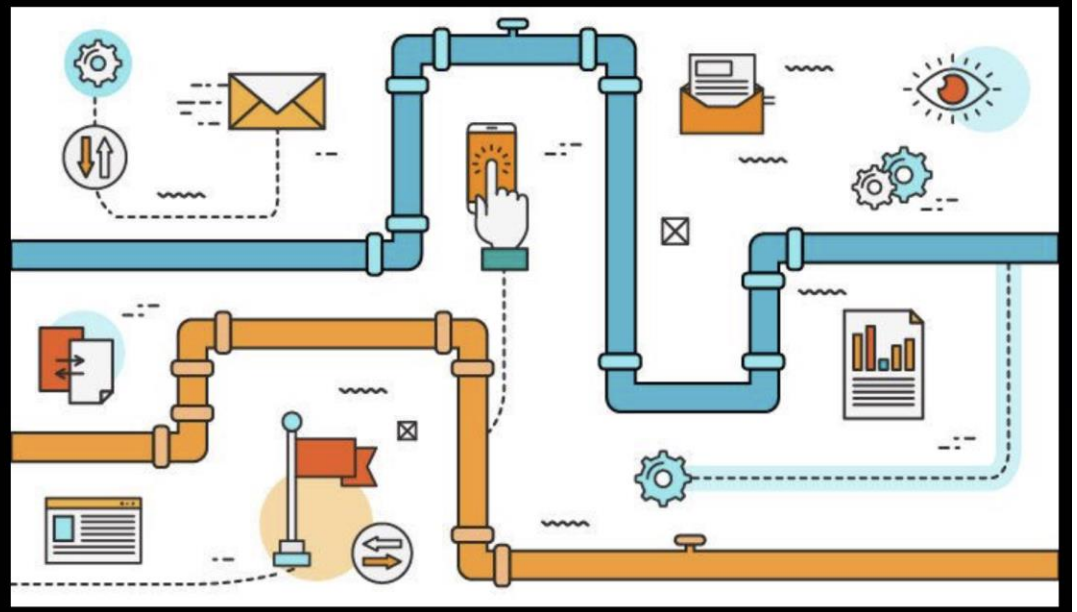
https://web.stanford.edu/class/cs245/slides/LakehouseGuestTalk.pdf

Abstraction = Hiding Complexity

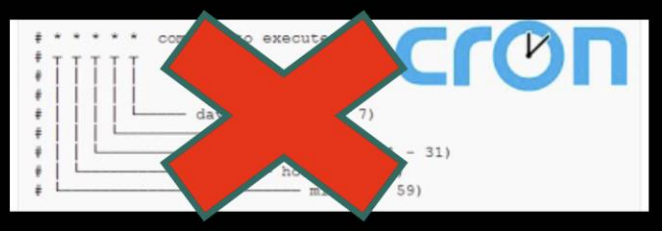


Data Orchestration

Data **orchestration** is the process of taking siloed data from multiple data storage locations, combining and organizing it, and making it available for data analysis tools. It can also include tasks like provisioning resources and monitoring.

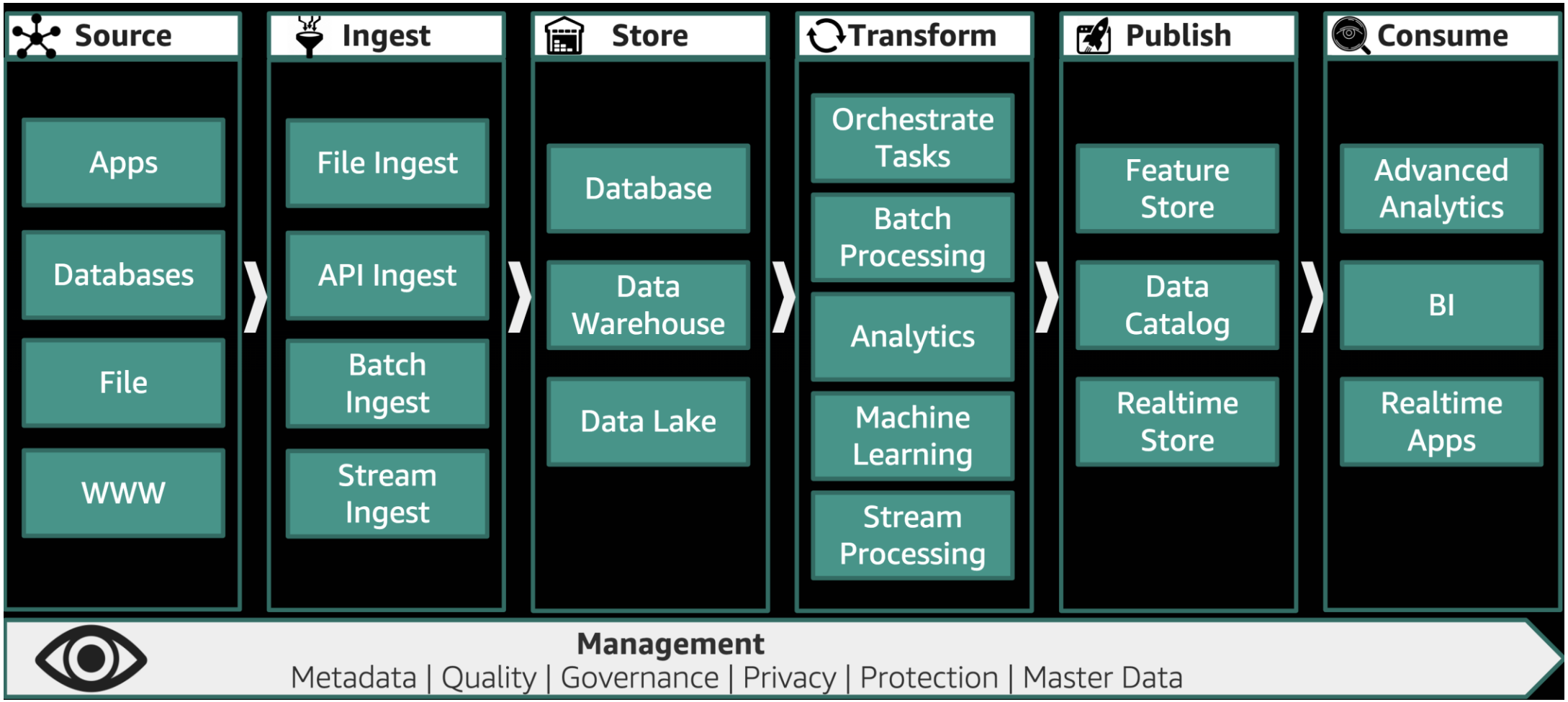


Data Pipelines, Workflows, Jobs

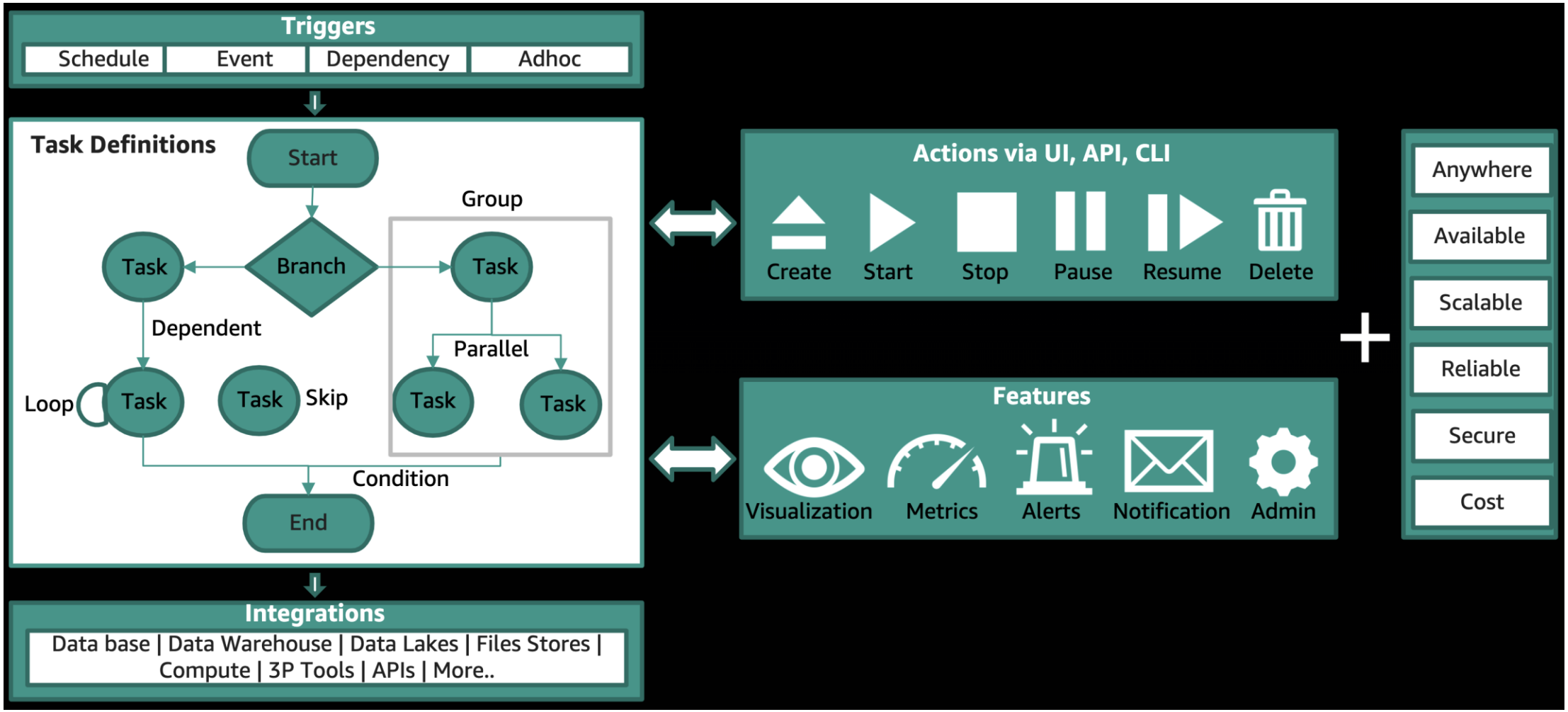


https://airflowsummit.org/slides/2022/b1-choosing-airflow-v2.pdf

Data Pipeline



The Tool Should

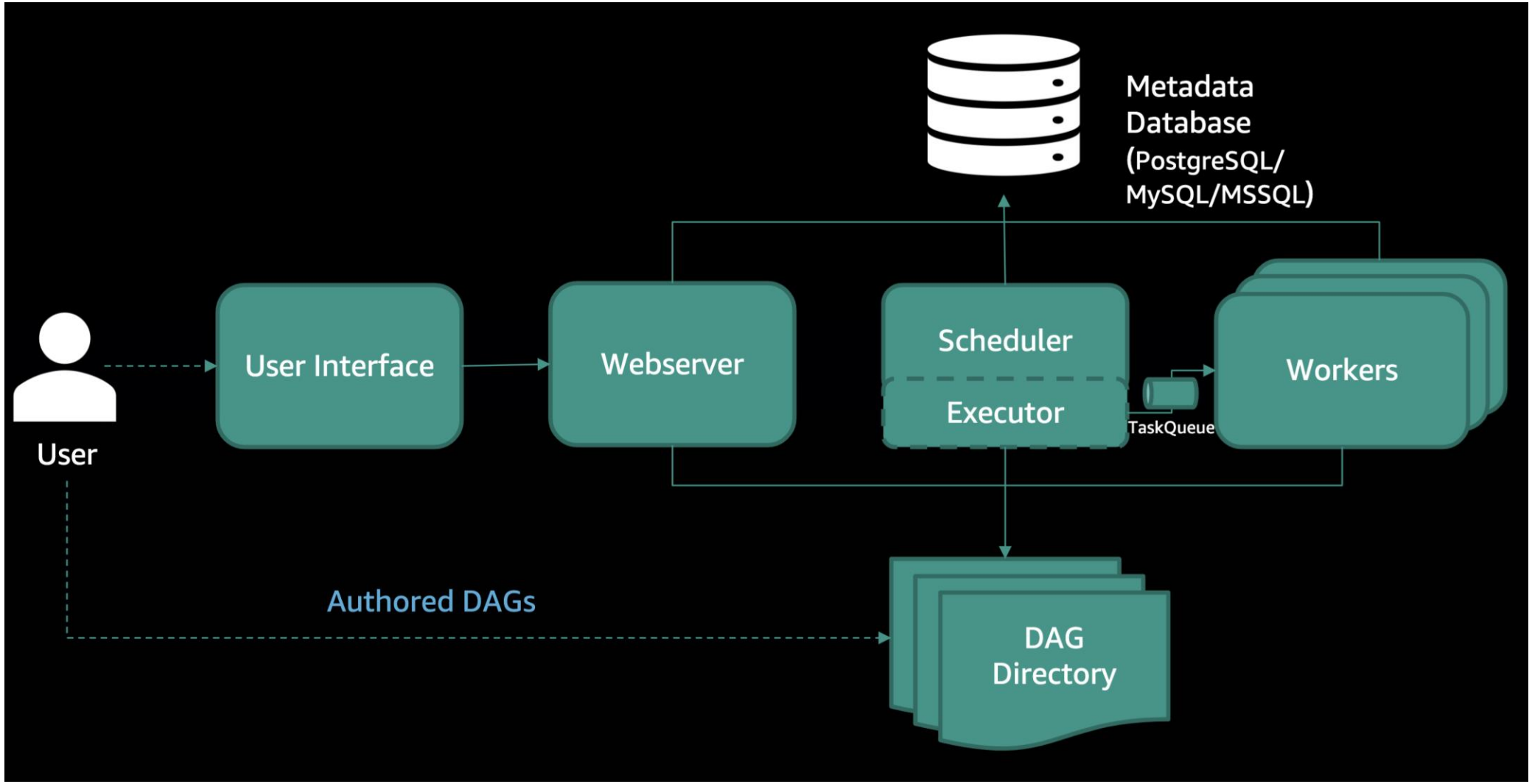


Airflow: Brief History

- Started in October 2014 by Maxime Beauchemin at Airbnb.
- Open source from the very first commit and officially brought under the Airbnb GitHub and announced in June 2015
- Joined the Apache Software Foundation's Incubator program in March 2016



Airflow: Architecture



Airflow: Concepts

- **Workflows** are called **DAGs** for Directed Acyclic Graph.

https://speakerdeck.com/artwr/apache-airflow-at-airbnb-introduction-and-lessons-learned

On DAG: airflow_maintenance

schedule: 1 day, 0:00:00

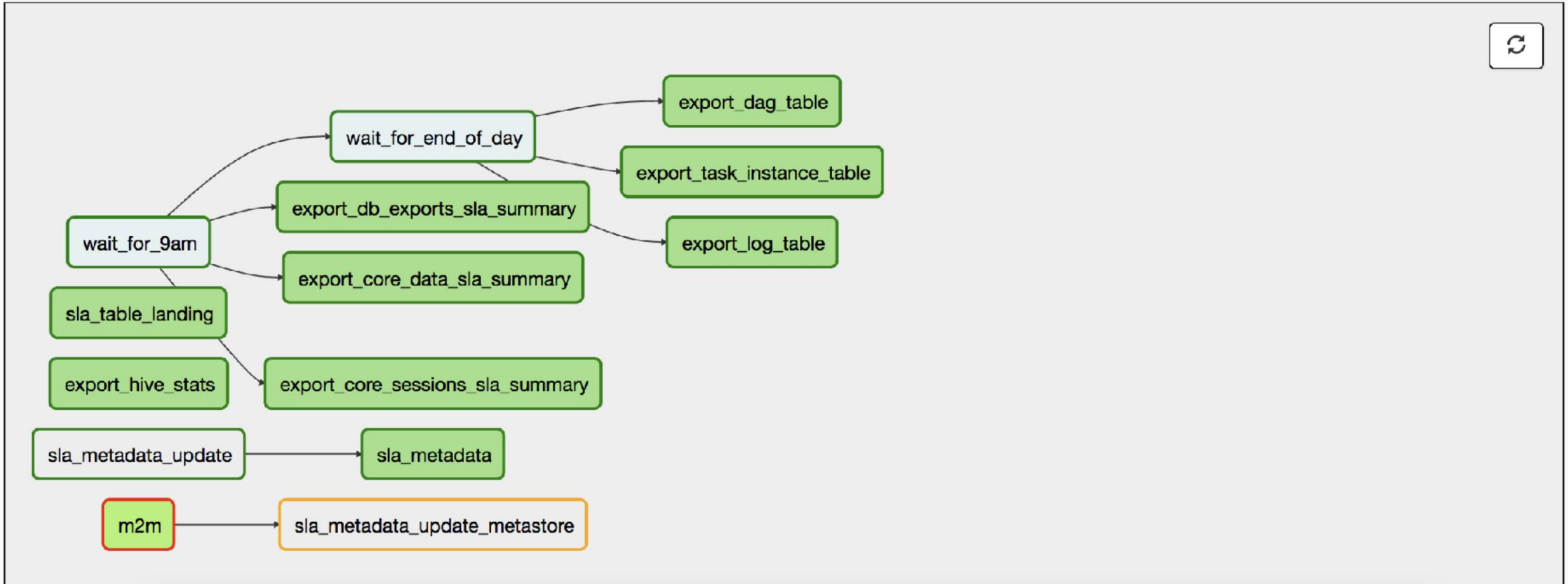
Graph View | Tree View | Task Duration | Task Tries | Landing Times | Gantt | Details | Code | Refresh

failed Run: scheduled__2017-02-15T00:00:00 Layout: Left->Right Go

Search for...

GenericTransfer MySqlOperator MySqlToHiveTransfer TimeSensor

success running failed skipped retry queued no status



```
dag = DAG(
    'tutorial',
    default_args=default_args,
    description='A simple tutorial DAG',
    schedule_interval=timedelta(days=1))
```

Airflow: Concepts

- **Tasks:** Workflows are composed of tasks called **Operators.**
- Operators can do pretty much anything that can be run on the Airflow machine.
- We tend to classify operators in 3 categories : **Sensors, Operators, Transfers.**

```
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag)
```

t2.set_upstream(t1)

Airflow: Code Example

```python
from airflow.contrib.sensors.file_sensor import FileSensor
from airflow.operators.dummy_operator import DummyOperator

import datetime
from datetime import date, timedelta
import airflow

default_args = {
    "depends_on_past": False,
    "start_date": airflow.utils.dates.days_ago(1),
    "retries": 1,
    "retry_delay": datetime.timedelta(hours=5),
}
today = datetime.datetime.today()
yesterday = date.today() - timedelta(days=1)

with airflow.DAG("file_sensor_example", default_args=default_args,
                 schedule_interval= "*/5 * * * *") as dag:

    start_task = DummyOperator(task_id="start")
    stop_task = DummyOperator(task_id="stop")
    sensor_task = FileSensor(task_id="file_sensor_task",
                             poke_interval=30,
                             fs_conn_id= "<path>",
                            filepath= "<file or directory name>")

start_task >> sensor_task >> stop_task
```

Imported Libraries

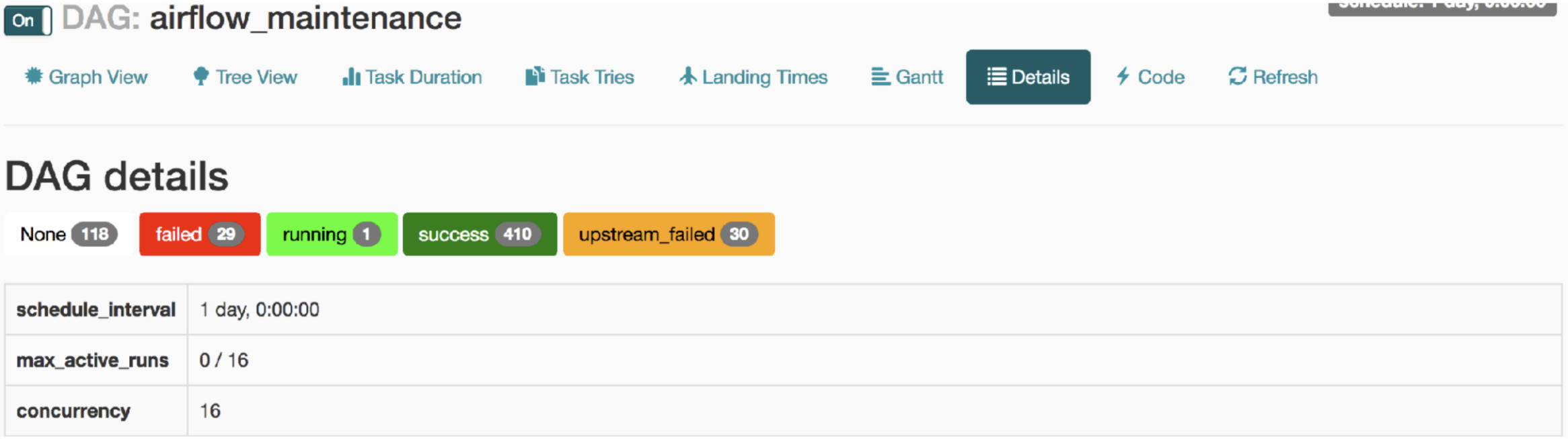
Default Arguments/dictionary of default parameters

Unique identifier + schedule interval

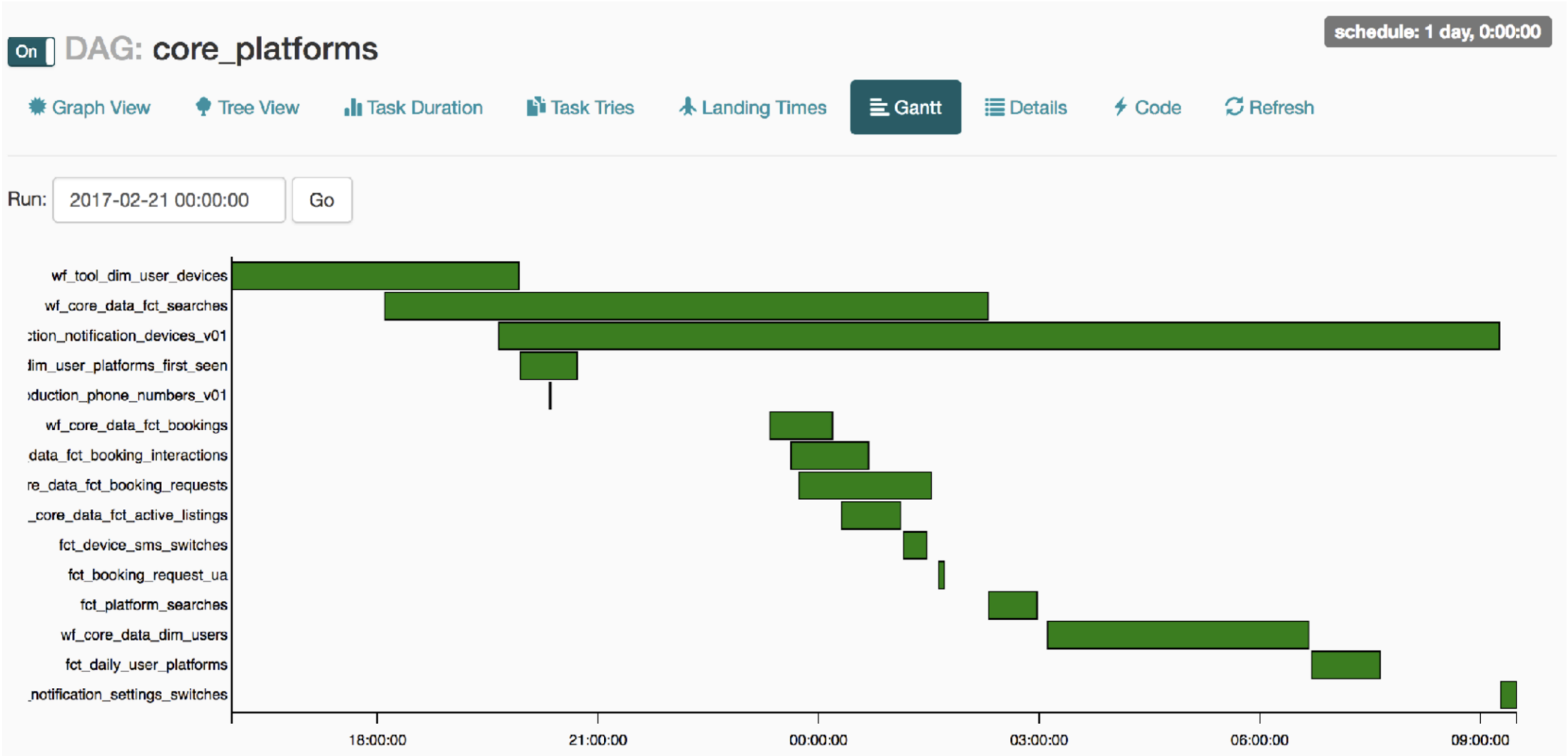
Task definitions

Dependency

Airflow: Monitoring DAG Status

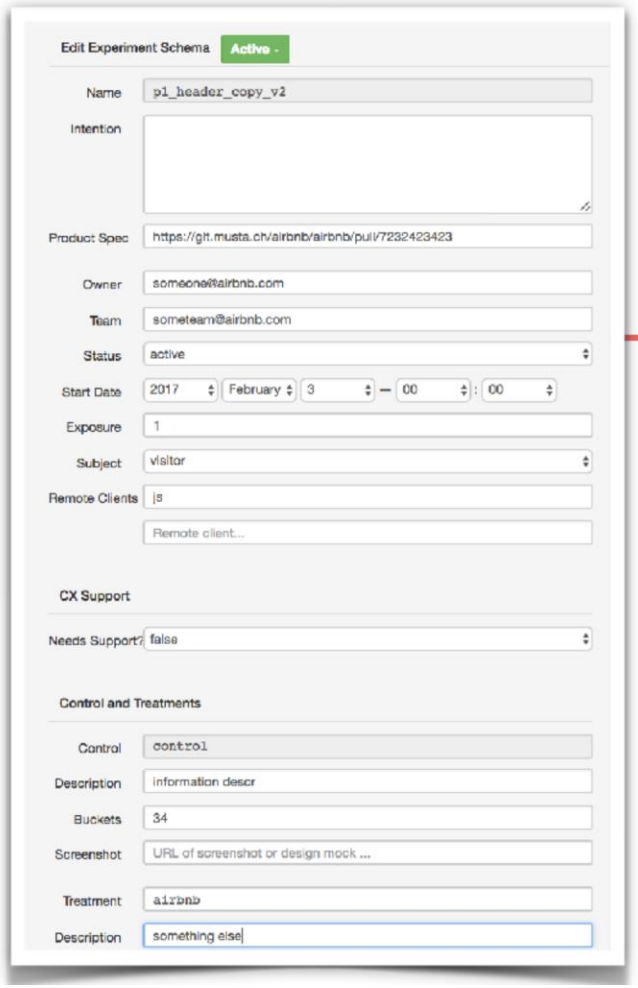


Airflow: Monitoring Gantt Chart

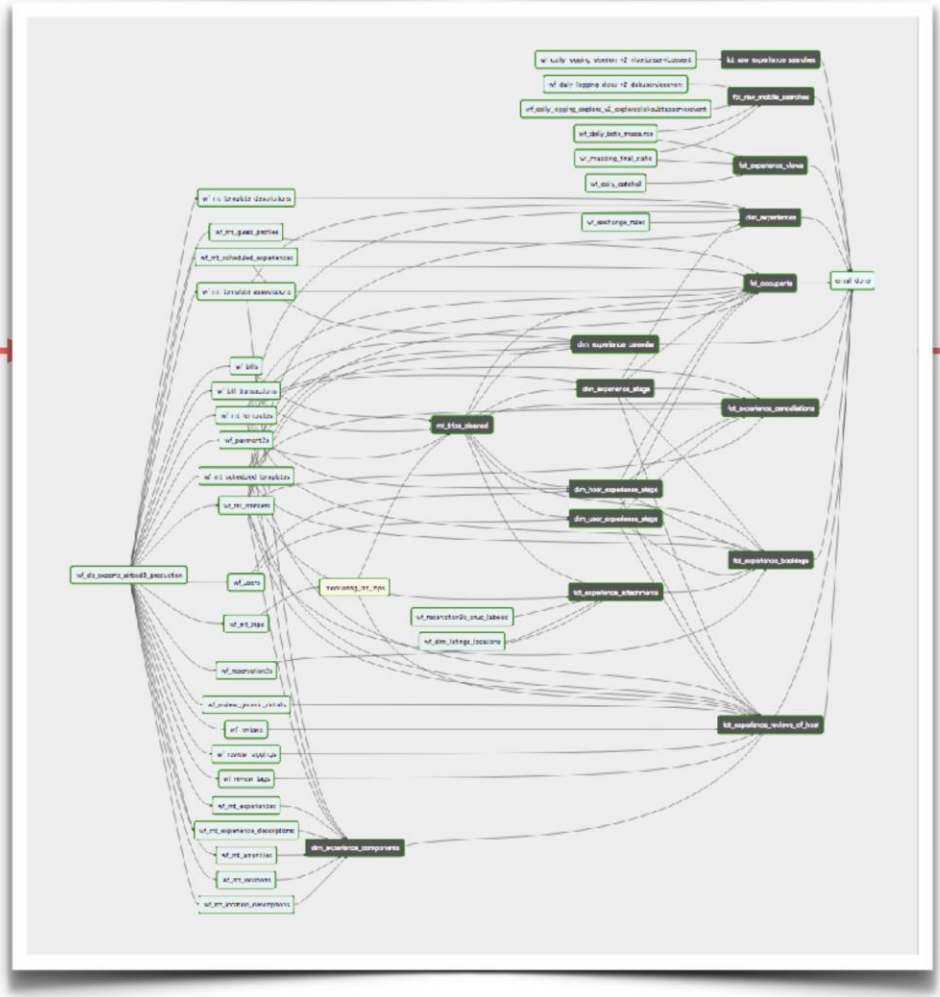


Airflow: Common Pattern

Input



Data Processing



Output

