Fundamentals of Data Engineering

Scenario

Every stakeholder interacts with the data ecosystem differently: Business executives want results and trends. Data scientists need structured data to build models. Developers need usable interfaces (APIs).Customers want value and privacy. You, the data engineer, ensure the whole data flow works efficiently and ethically.

Data Engineer

Wasting time & resources!

System Requirements

Functional Requirements	Nonfunctional Requirements

Technology Choices

- Ingestion tool
- Storage system
- Transformation tool

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Module-3

1. Common source systems
2. Databases, object storage
3. Streaming sources
4. Databases, object storage, and streaming sources
5. Setting up ingestion from source systems
6. DataOps undercurrent
7. Automating some of your pipeline tasks
8. Orchestration, monitoring, and automating data pipelines
9. Apache Kafka and Apache Airflow.

Structured Data

Data organized as tables of rows and columns

ID	Last	First	Card
14	Barry	John	XXX878
25	Goode	Cynthia	XXX980
14	Barry	John	XXX978
25	Goode	Cynthia	XXX990

Rows

Columns

Video by Adobe Stock (paid license)

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

```
import csv
with open('eggs.csv', newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
    for row in spamreader:
        print(', '.join(row))
```

Structured Data

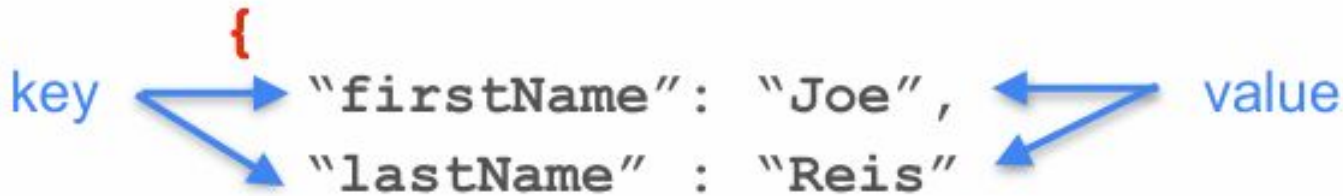
Data organized as tables of rows and columns

Semi-Structured Data

Data that is not in tabular form but still has some structure

JavaScript Object Notation (JSON)

A series of key-value pairs




```
{
  "firstName": "Joe",
  "lastName" : "Reis"
  "age": 10 ,
  "languages":["Python", "JavaScript", "SQL"],
  "address": {
        "city": "Los Angeles",
        "postalCode": 90024,
        "country": "USA"
  }
```

Structured Data — Data organized as tables of rows and columns

Semi-Structured Data — Data that is not in tabular form but still has some structure

JavaScript Object Notation (JSON)

A series of key-value pairs

Nested JSON format

Structured Data — Data organized as tables of rows and columns

Semi-Structured Data — Data that is not in tabular form but still has some structure

Unstructured Data — Data that does not have any predefined structure

Text

Video

Audio

Images

- dimensions
- pixel colors

Databases

Structured data
Semi-structured data

Files

Streaming Systems

Semi-structured data

Databases

Store data in an organized way

Structured data

Semi-structured data

Create

Read

Update

Delete

Files

Streaming Systems

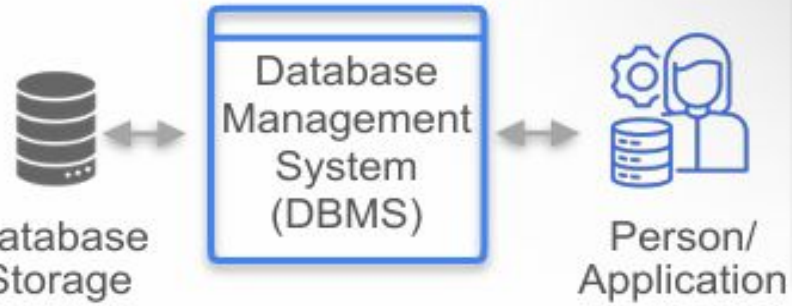
Semi-structured data

Databases

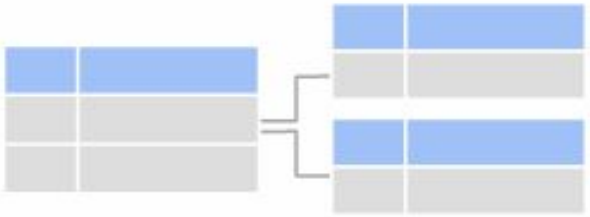
Store data in an organized way

Structured data
Semi-structured data

Create
Read
Update
Delete



Relational databases



Tables with rows and columns

Non-relational (NoSQL) databases

Non-tabular data

Databases

Store data in an organized way

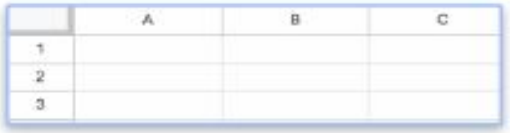
Structured data

Semi-structured data

Create
Read
Update
Delete

Files

Sequence of bytes representing information




```
{
 "firstName": "Joe",
 "lastName" : "Reis",
 "languages":["R", "SQL"],
}
```

Streaming Systems

Semi-structured data

Databases

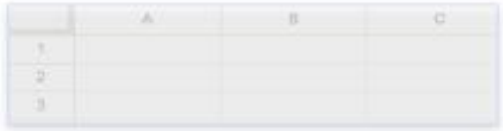
Store data in an organized way

Structured data
Semi-structured data

Create
Read
Update
Delete

Files

Sequence of bytes representing information



Streaming Systems

Continuous flow of data

Semi-structured data

Databases

Store data in an organized way

Structured data

Semi-structured data

Create
Read
Update
Delete

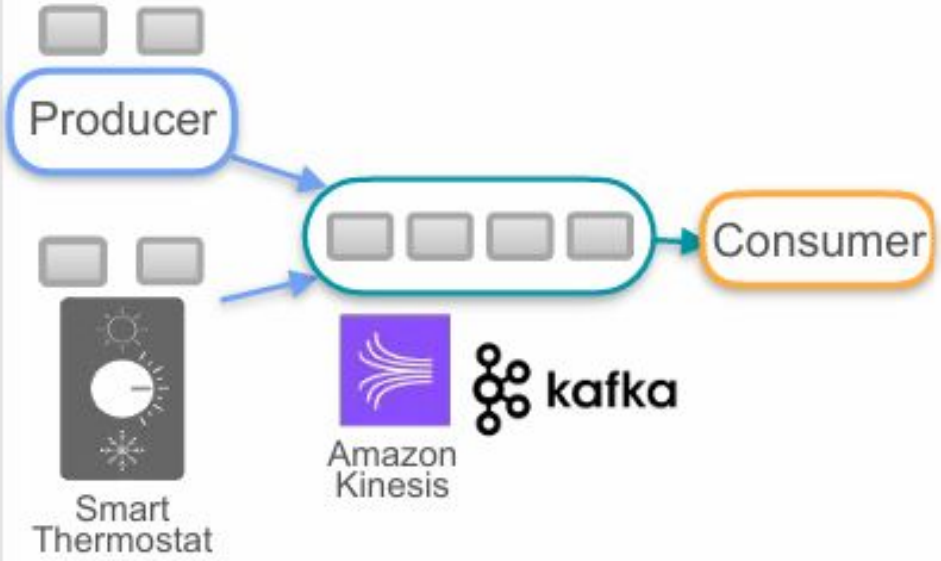
Files

Sequence of bytes representing information

Streaming Systems

Continuous flow of data

Semi-structured data



Databases

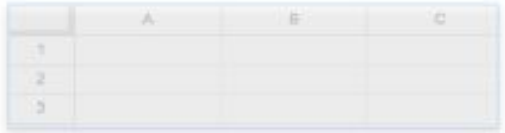
Store data in an organized way

Structured data
Semi-structured data

Create
Read
Update
Delete

Files

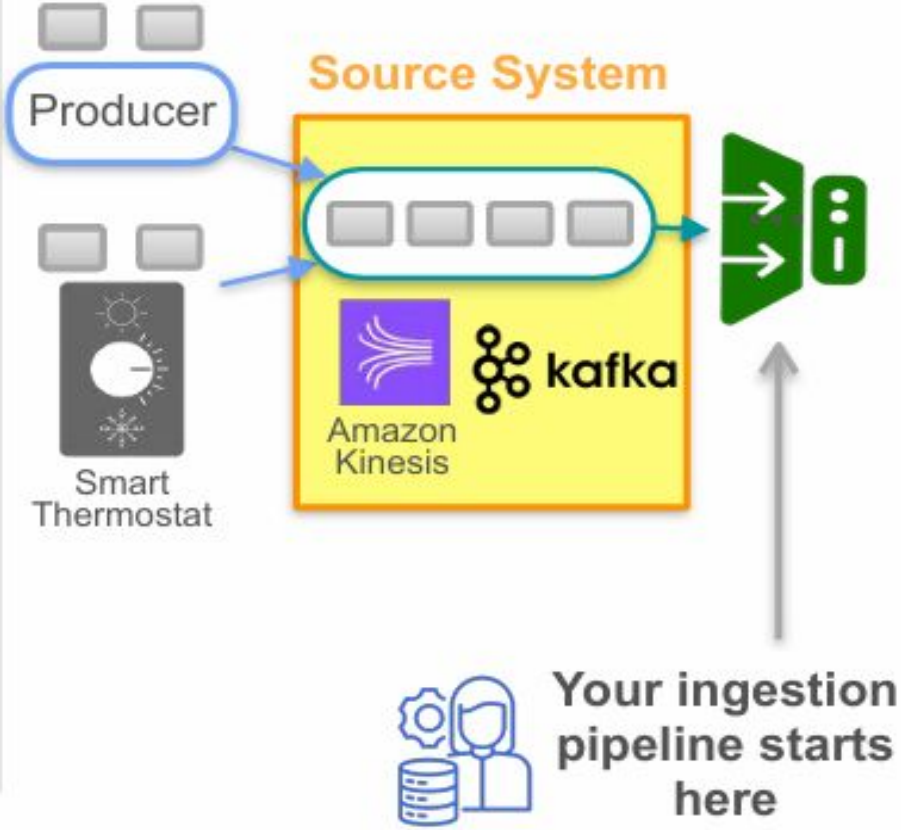
Sequence of bytes representing information

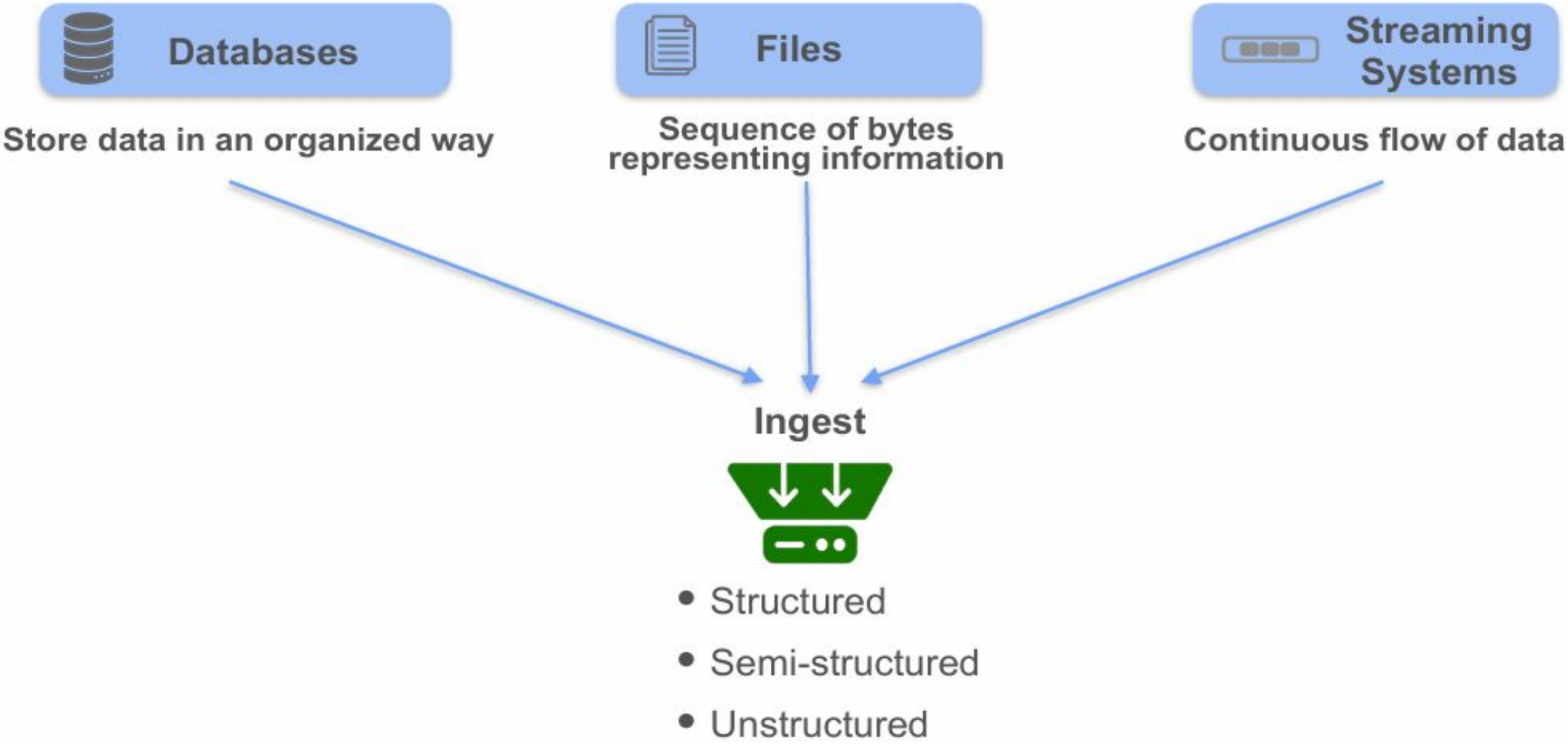


Streaming Systems

Continuous flow of data

Semi-structured data

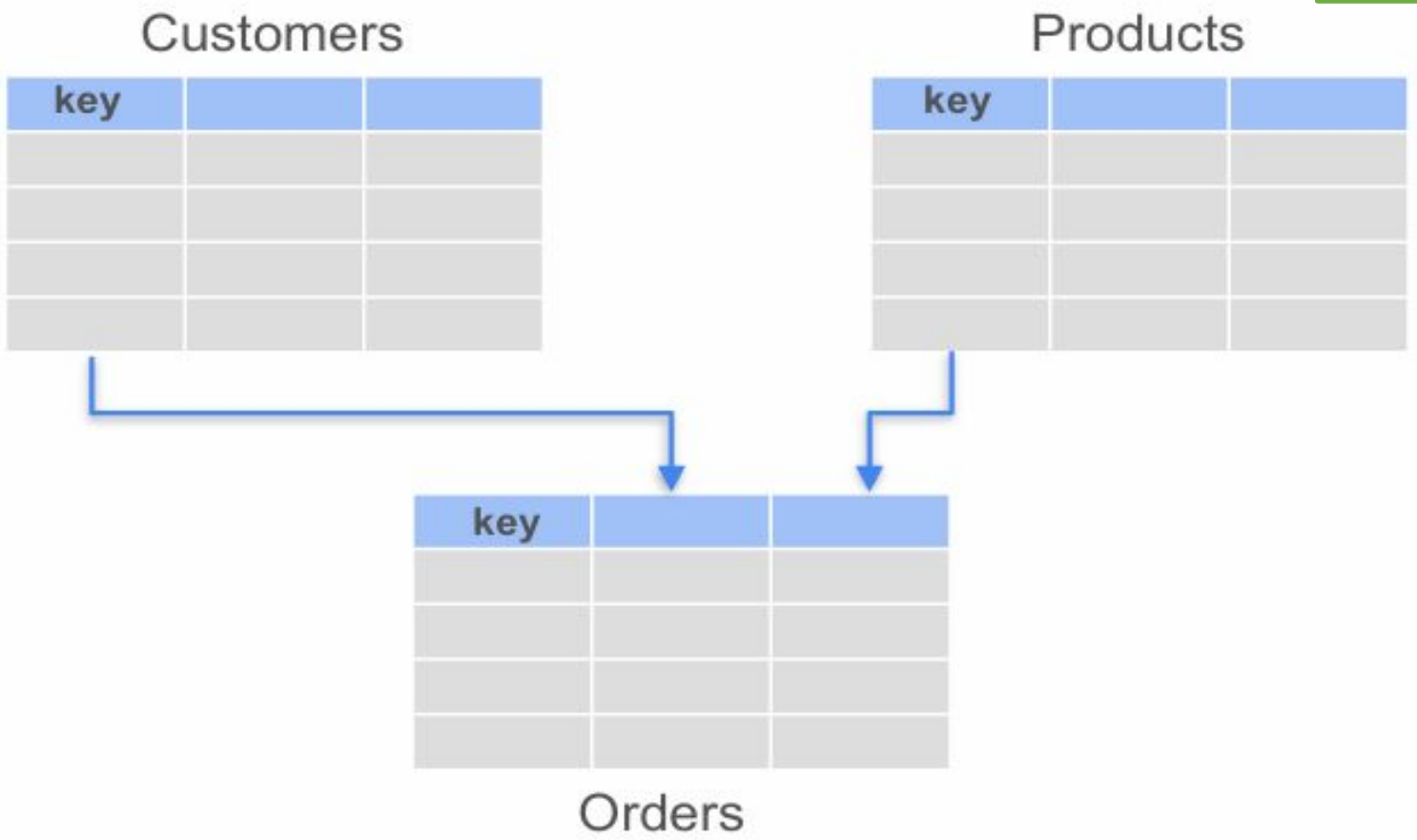



Databases
Store data in an organized way
Files
Sequence of bytes representing information
Streaming Systems
Continuous flow of data
Ingest
Structured
Semi-structured
Unstructured

Relational Databases

Most common source system for data engineers because:
Used widely in web/mobile apps.
Common in corporate systems like CRM, HR, ERP.
Power **OLTP (Online Transaction Processing)** systems such as banking or booking platforms.

Relational Databases



Data is stored across related tables connected by keys.
Example in e-commerce:
Customers table → customer info.
Products table → product info.
Orders table → order info.
This avoids duplication (e.g., customer address or product details repeated many times).

- Reduce redundancy
- Make data easier to manage

Relational Databases

One customer place the order there are three entry for the address.
If you maintain one big table ,data Redundancy and harder to maintain the table.

One big table for everything!

name	address	phone	date_time	amount	brand	SKU	description
Jane Doe	74th Street	12345678	12/08/2024	700	ABC	B32	Blender
Jane Doe	74th Street	12345678	12/08/2024	99	XYZ	i56	Iron
Jane Doe	74th Street	12345678	12/08/2024	100	GHJ	k70	Kettle

Jane Doe

Relational Databases

Three different customers place the order for same product again it three entry

One big table for everything!

name	address	phone	date_time	amount	brand	SKU	description
Jane Doe	74th Street	12345678	12/08/2024	700	ABC	B32	Blender
Jane Doe	74th Street	12345678	12/08/2024	99	XYZ	i56	Iron
Jane Doe	74th Street	12345678	12/08/2024	100	GHJ	k70	Kettle
Mary Ann	19th Avenue	98765432	13/08/2024	899	STU	w40	Washer
John Ken	1st Link	36891623	14/08/2024	899	STU	w40	Washer
Ivy Tan	67th Street	98639513	15/08/2024	899	STU	w40	Washer

Relational Databases

One big table for everything!

Inconsistency

name	address	phone	date_time	amount	brand	SKU	description
Jane Doe	11th Avenue	12345678	12/08/2024	700	ABC	B32	Blender
Jane Doe	11th Avenue	12345678	12/08/2024	99	XYZ	i56	Iron
Jane Doe	74th Street	12345678	12/08/2024	100	GHJ	k70	Kettle
Mary Ann	19th Avenue	98765432	13/08/2024	899	STU	w31	Washer
John Ken	1st Link	36891623	14/08/2024	899	STU	w31	Washer
Ivy Tan	67th Street	98639513	15/08/2024	899	STU	w40	Washer

Jane Doe
now lives on 11th Avenue

SKU
now w31

Inconsistency

Relational Databases

Single customer

Single product

Customers

id	first_name	last_name	age	address
1	Jane	Doe	24	11th Ave.
2	Mary	Ann	65	19th Ave.
3	John	Ken	27	1st Link
4	Ivy	Tan	18	67th St.

Products

id	brand	SKU	description
1	ABC	b32	Blender
2	XYZ	i56	Iron
3	GHJ	k70	Kettle
4	STU	w31	Washer

Orders

id	customer_id	product_id	date_time	purchase_amount

Database schema

Relational Databases

Keys

Primary key:
uniquely identifies each row in a table

Customers

id	first_name	last_name	age	address
1	Jane	Doe	24	11th Ave.
2	Mary	Ann	65	19th Ave.
3	John	Ken	27	1st Link
4	Ivy	Tan	18	67th St.

Products

id	brand	SKU	description
1	ABC	b32	Blender
2	XYZ	i56	Iron
3	GHJ	k70	Kettle
4	STU	w31	Washer

Orders

id	customer_id	product_id	date_time	purchase_amount
1	1	1	12/08/2024	700
2	1	2	12/08/2024	99
3	1	3	12/08/2024	100
4	2	4	13/08/2024	899
5	3	4	14/08/2024	899

Database schema

Foreign key:
references the primary key of the customer table

Relational Databases

Customers

id	first_name	last_name	age	address
1	Jane	Doe	24	11th Ave.
2	Mary	Ann	65	19th Ave.
3	John	Ken	27	1st Link
4	Ivy	Tan	18	67th St.

Products

id	brand	SKU	description
1	ABC	b32	Blender
2	XYZ	i56	Iron
3	GHJ	k70	Kettle
4	STU	w31	Washer

Orders

id	customer_id	product_id	date_time	purchase_amount
1	1	1	12/08/2024	700
2	1	2	12/08/2024	99
3	1	3	12/08/2024	100
4	2	4	13/08/2024	899
5	3	4	14/08/2024	899

Database schema

Each row in a table has to follow the same column structure:
same sequence of columns and data types

Relational Databases

Customers

id	first_name	last_name	age	address
1	Jane	Doe	24	11th Ave.
2	Mary	Ann	65	19th Ave.
3	John	Ken	27	1st Link
4	Ivy	Tan	18	67th St.

Products

id	brand	SKU	description
1	ABC	b32	Blender
2	XYZ	i56	Iron
3	GHJ	k70	Kettle
4	STU	w31	Washer

Orders

id	customer_id	product_id	date_time	purchase_amount
1	1	1	12/08/2024	700
2	1	2	12/08/2024	99
3	1	3	12/08/2024	100
4	2	4	13/08/2024	899
5	3	4	14/08/2024	899
6	1	4	15/08/2024	899

Relational Databases

normalized databases may be slower for querying because joins are required. Some systems use One Big Table (OBT) design for faster processing.

One big table for everything!

name	address	phone	date_time	amount	brand	SKU	description
Jane Doe	74th Street	12345678	12/08/2024	700	ABC	B32	Blender
Jane Doe	74th Street	12345678	12/08/2024	99	XYZ	i56	Iron
Jane Doe	74th Street	12345678	12/08/2024	100	GHJ	k70	Kettle
Mary Ann	19th Avenue	98765432	13/08/2024	899	STU	w40	Washer
John Ken	1st Link	36891623	14/08/2024	899	STU	w40	Washer
Ivy Tan	67th Street	98639513	15/08/2024	899	STU	w40	Washer

One Big Table (OBT) approach: use cases that need faster processing

Relational Databases

Relational Database Management System (RDBMS)

Software layer that sits on top of a relational database to manage and interact with the data.

MySQL

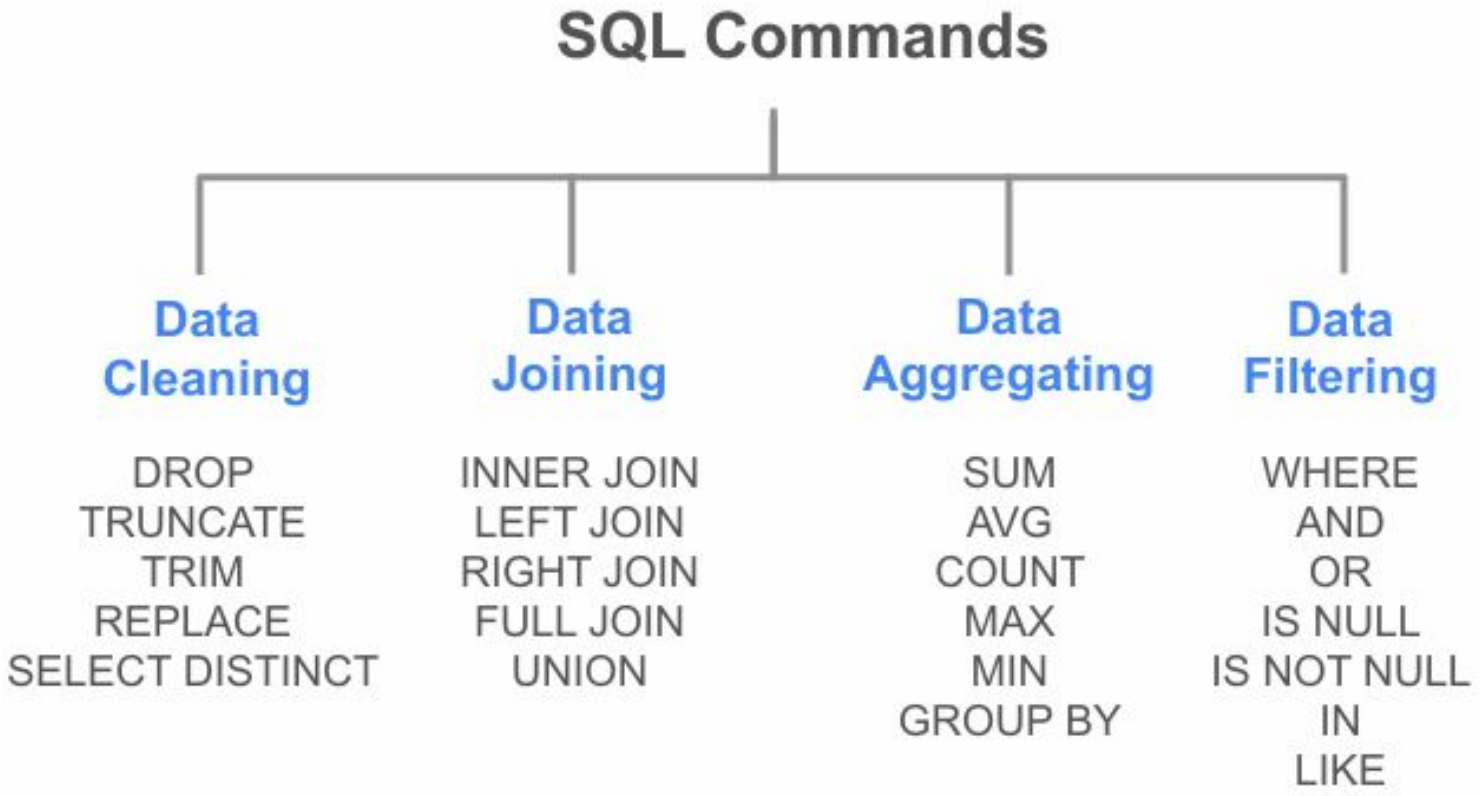
ORACLE DATABASE

PostgreSQL

Microsoft SQL Server

Structured Query Language (SQL)

Relational Databases

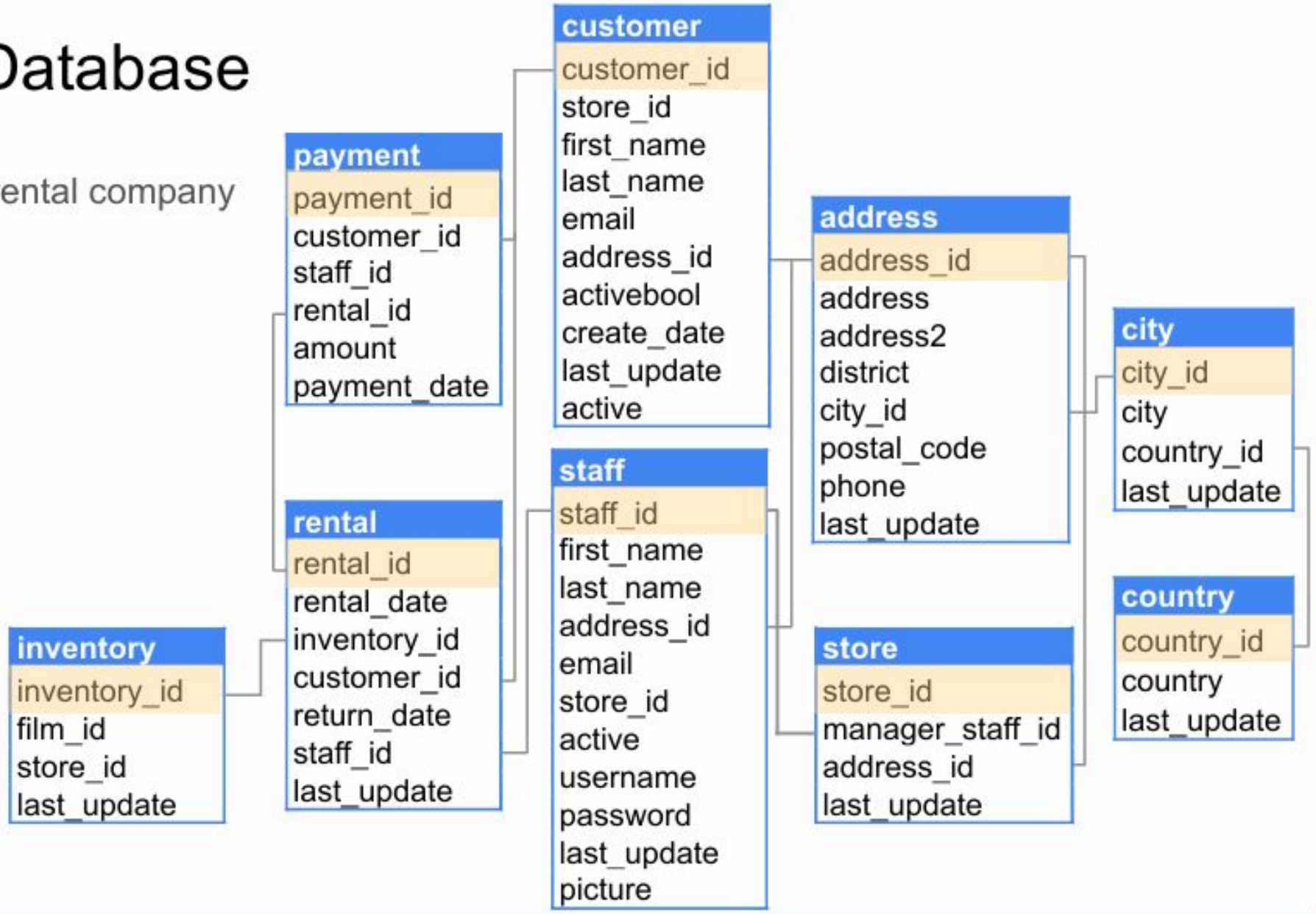


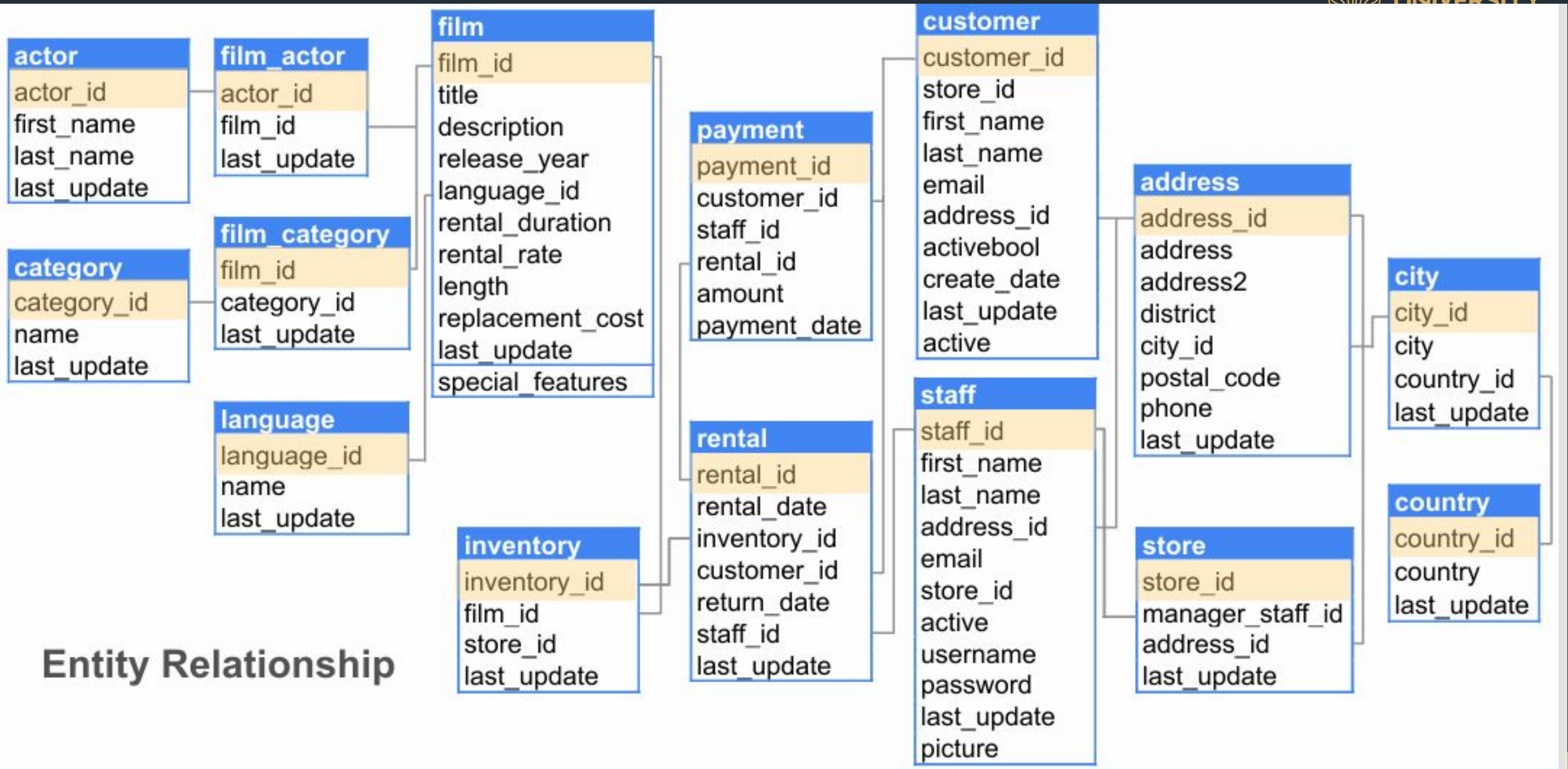
The Relational Database

- Database for a fictitious DVD rental company called **Rentio**
- Database schema

SQL queries

Answer business questions



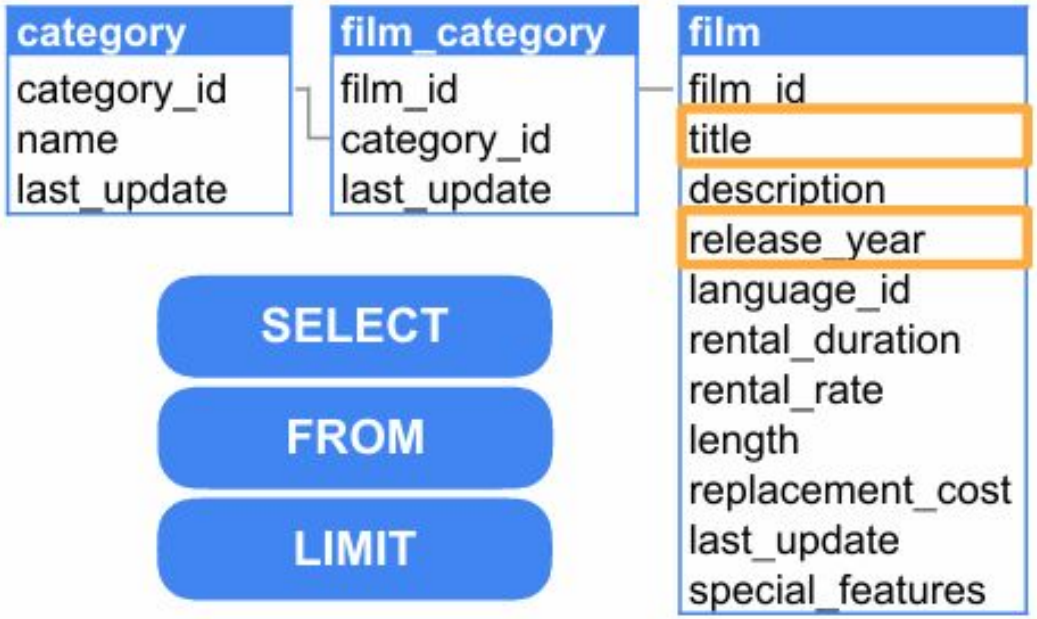


Entity Relationship


```
In [1]: %load_ext sql
        %sql mysql+pymysql://root:adminpwrd@localhost:3306/sakila

In [ ]: %%sql

In [ ]:
```



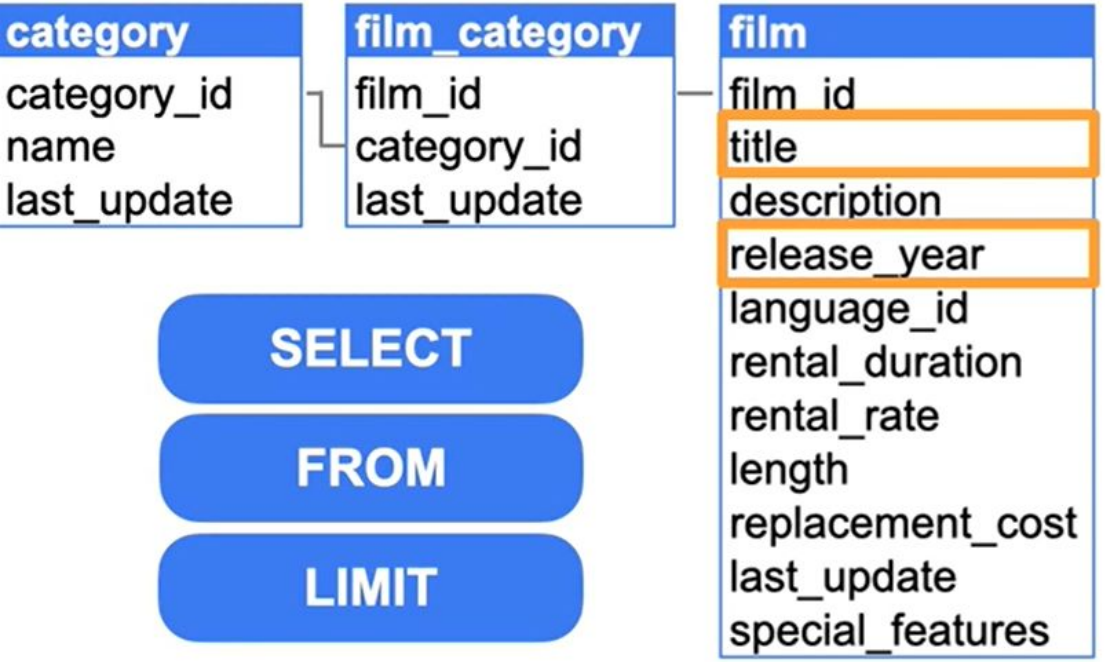

```
In [1]: %load_ext sql
        %sql mysql+pymysql://root:adminpwrd@localhost:3306/sakila

In [2]: %%sql
        SELECT title, release_year
        FROM film
```

```
 * mysql+pymysql://root:***@localhost:3306/sakila
1000 rows affected.
```

Out[2]:

title	release_year
ACADEMY DINOSAUR	2006
ACE GOLDFINGER	2006
ADAPTATION HOLES	2006
AFFAIR PREJUDICE	2006
AFRICAN EGG	2006
AGENT TRUMAN	2006



SELECT

FROM

LIMIT

```
In [1]: %load_ext sql
        %sql mysql+pymysql://root:adminpwrd@localhost:3306/sakila

In [2]: %%sql
        SELECT title, release_year
        FROM film
        LIMIT 10
```

```
 * mysql+pymysql://root:***@localhost:3306/sakila
1000 rows affected.
```

Out[2]:

title	release_year
ACADEMY DINOSAUR	2006
ACE GOLDFINGER	2006
ADAPTATION HOLES	2006
AFFAIR PREJUDICE	2006
AFRICAN EGG	2006

category
category_id
name
last_update

film_category
film_id
category_id
last_update

film
film_id
title
description
release_year
language_id
rental_duration
rental_rate
length
replacement_cost
last_update
special_features

SELECT

FROM

WHERE

Exploring the films that are less than 60 minutes long.

category
category_id
name
last_update

film_category
film_id
category_id
last_update

film
film_id
title
description
release_year
language_id
rental_duration
rental_rate
length
replacement_cost
last_update
special_features

SELECT

FROM

LIMIT

AIRPLANE SIERRA	2006
AIRPORT POLLOCK	2006
ALABAMA DEVIL	2006
ALADDIN CALENDAR	2006

In [4]:
```
%%sql
SELECT *
FROM film
```

```
 * mysql+pymysql://root:***@localhost:3306/sakila
1000 rows affected.
```

Out[4]:

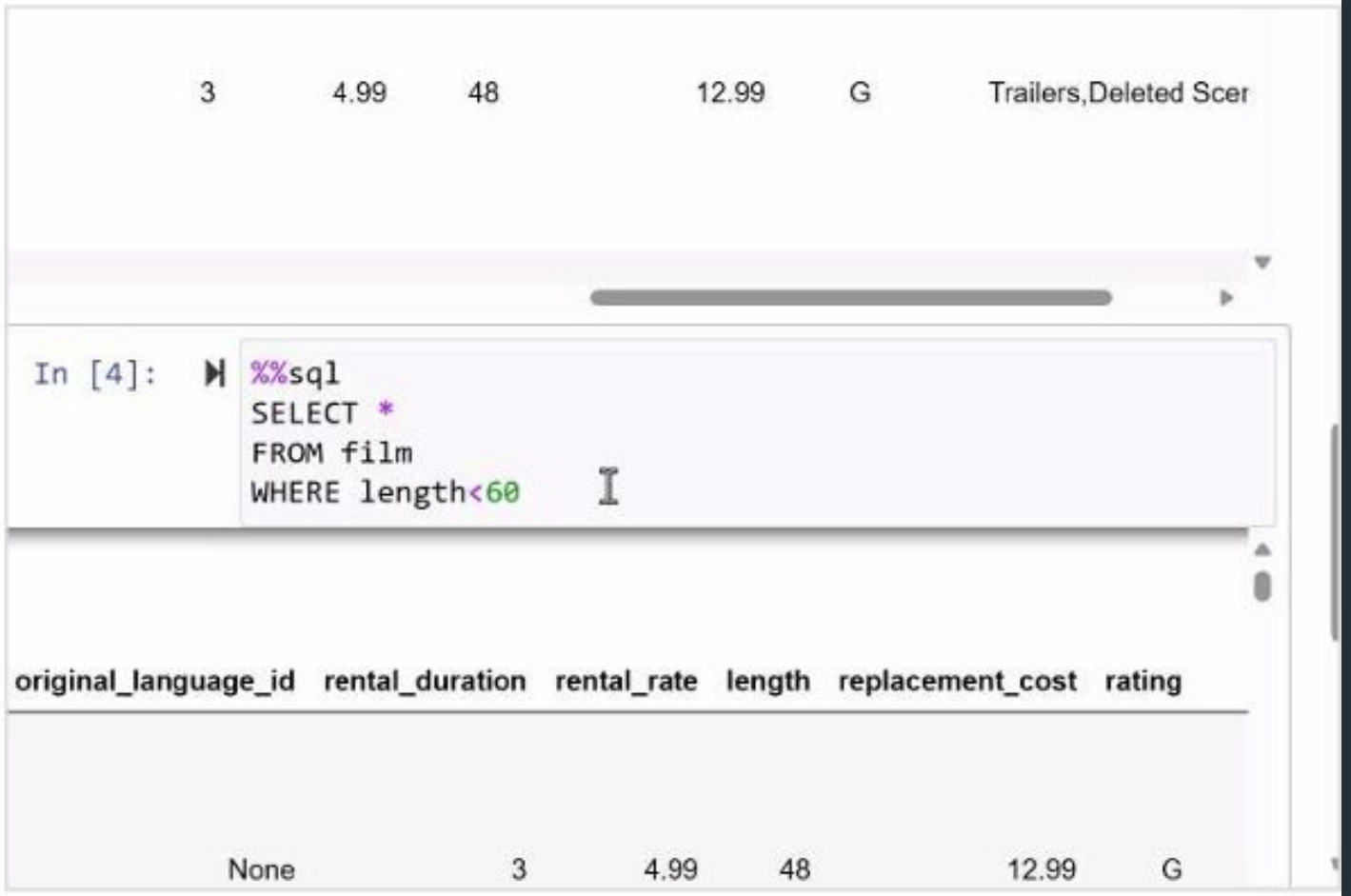
film_id	title	description	release_year	language_id
1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian	2006	1

category
category_id
name
last_update

film_category
film_id
category_id
last_update

film
film_id
title
description
release_year
language_id
rental_duration
rental_rate
length
replacement_cost
last_update
special_features

- SELECT
- FROM
- WHERE
- ORDER BY



category
category_id
name
last_update

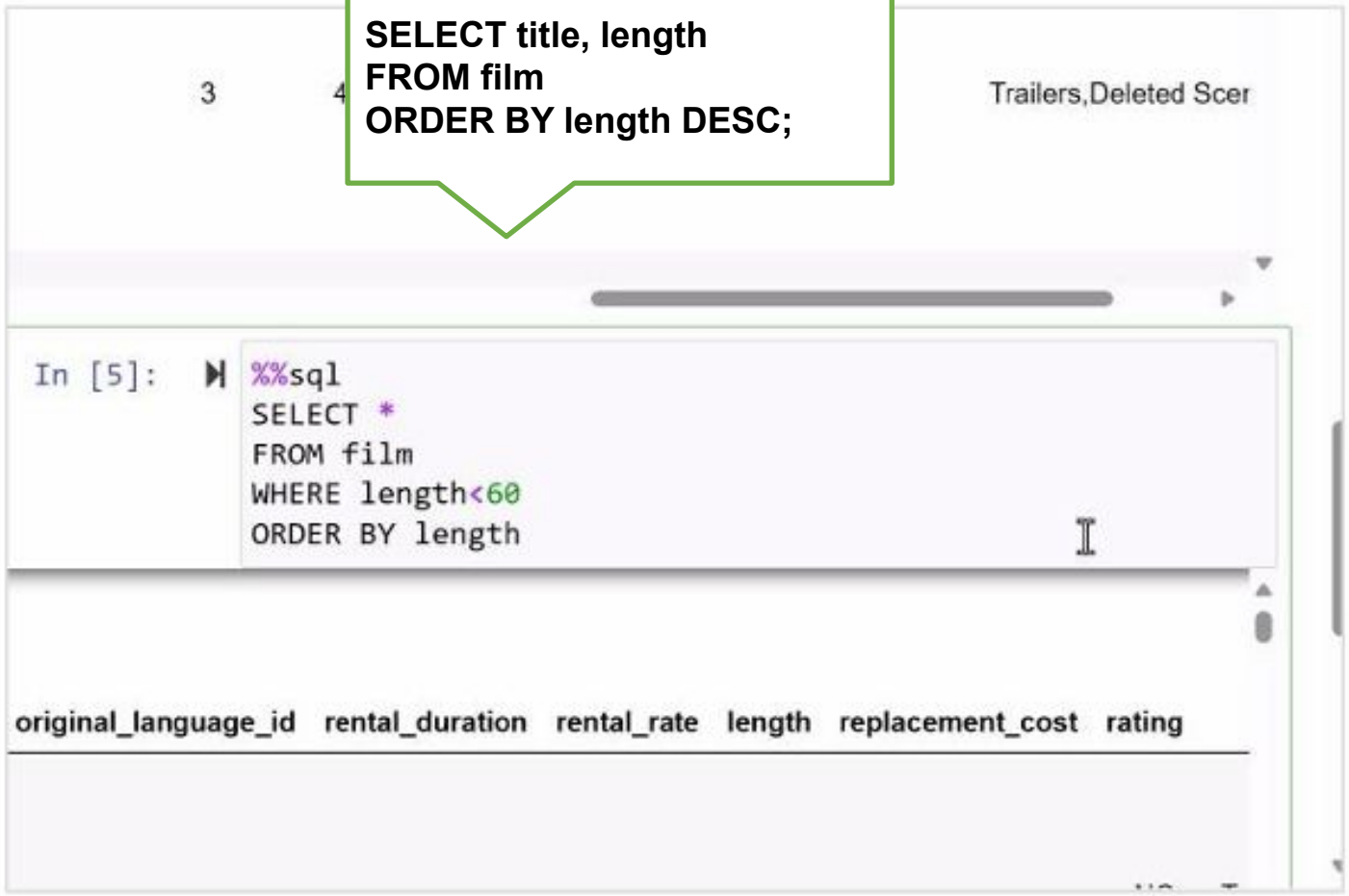
film_category
film_id
category_id
last_update

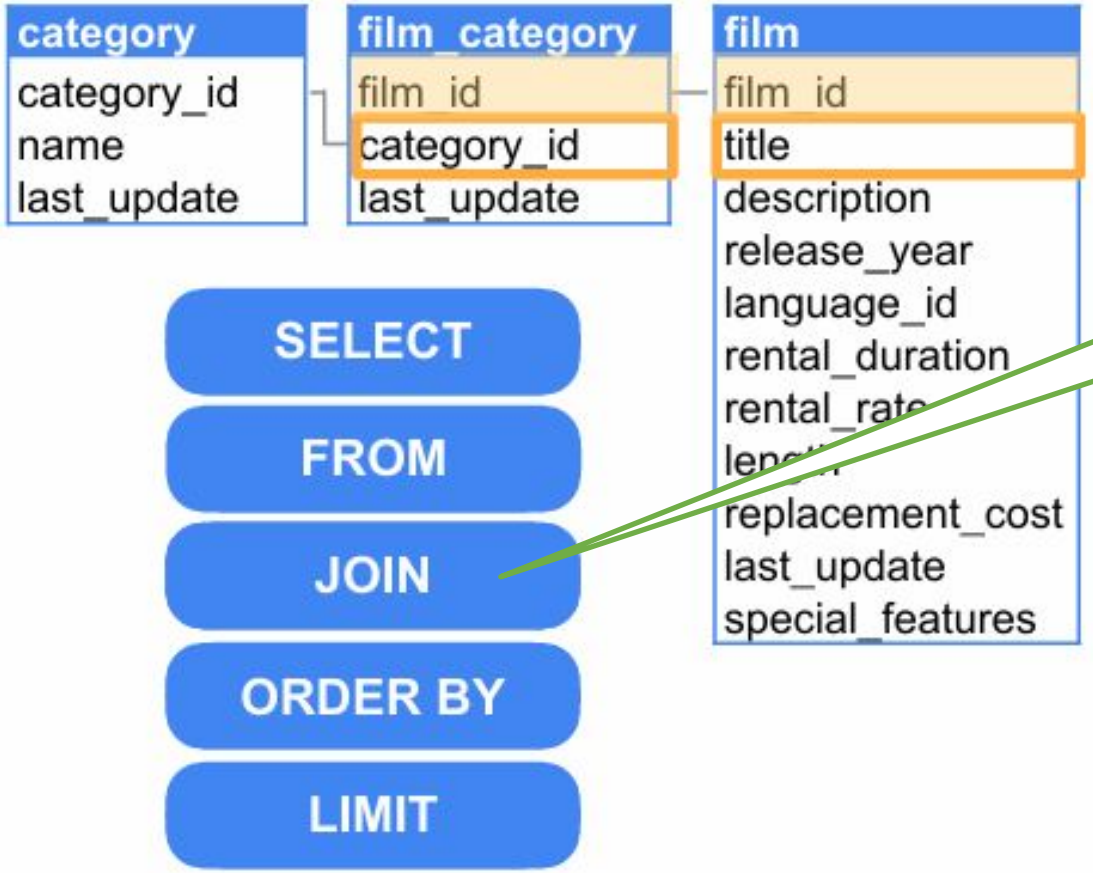
film
film_id
title
description
release_year
language_id
rental_duration
rental_rate
length
replacement_cost
last_update
special_features

- SELECT
- FROM
- WHERE
- ORDER BY
- LIMIT

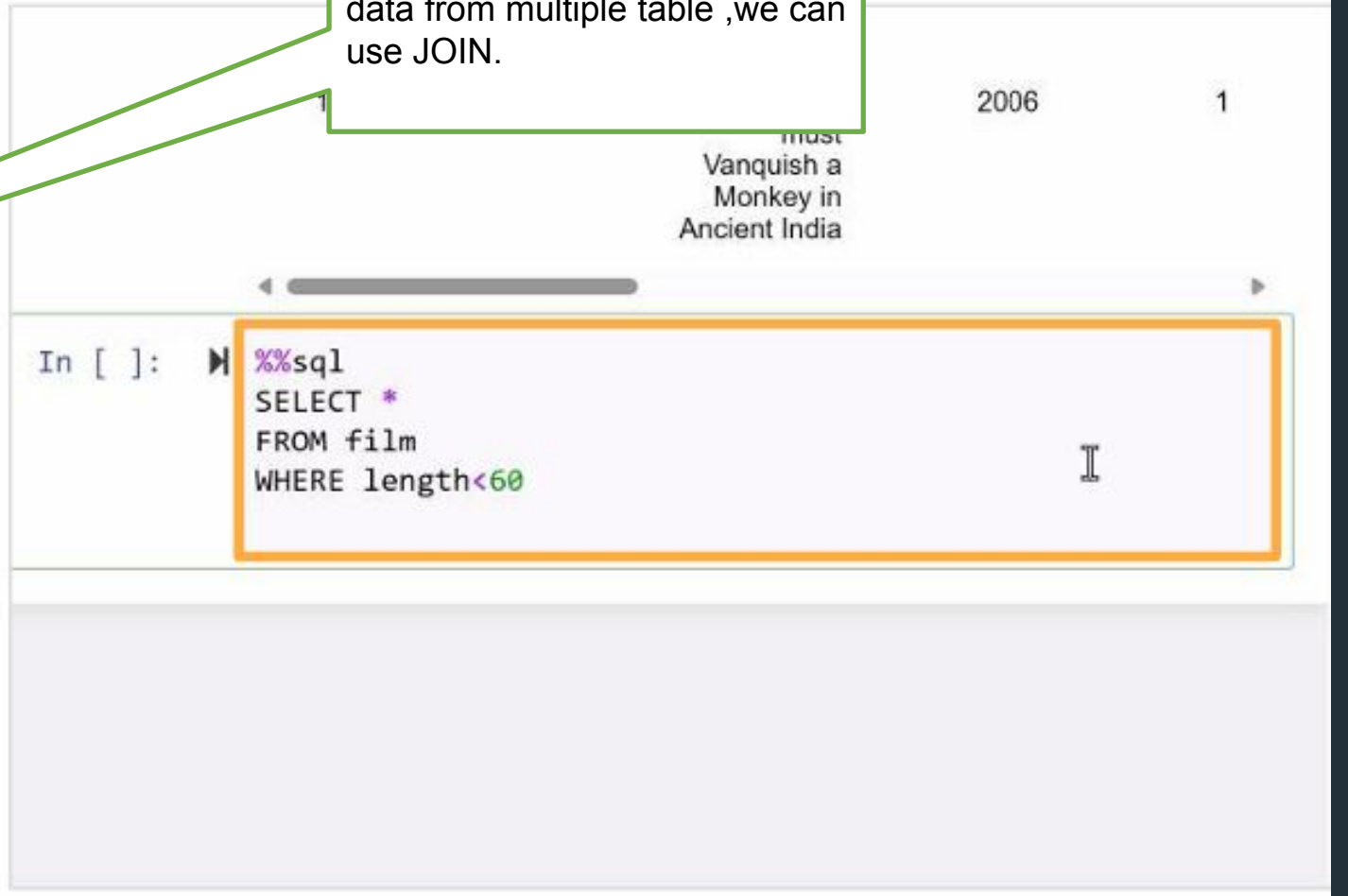
Order by is used to sort a collection of items or a result set, typically in ascending or descending order

SELECT title, length
FROM film
ORDER BY length DESC;





What if you want to retrieve data from multiple table ,we can use JOIN.



- SELECT
- FROM
- JOIN
- WHERE
- ORDER BY
- LIMIT

Monkey in
Ancient India

```
In [5]:
%%sql
SELECT *
FROM film
JOIN film_category
ON film.film_id = film_category.film_id
WHERE length<60
```

```
ysql+pymysql://root:***@localhost:3306/sakila
ows affected.
```

_id	title	description	release_year	language_id	original_language_id
2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrator And a Explorer who must Find a	2006	1	None

- SELECT
- FROM
- JOIN
- WHERE
- ORDER BY
- LIMIT

Monkey in
Ancient India

```
In [5]: %%sql
        SELECT *
        FROM film
        JOIN film_category
        ON film.film_id = film_category.film_id
        WHERE length<60
```

rating	special_features	last_update	film_id_1	category_id	last_update_1
G	Trailers,Deleted Scenes	2006-02-15 05:03:42	2	11	2006-02-15 05:07:09

category
category_id
name
last_update

film_category
film_id
category_id
last_update

film
film_id
title
description
release_year
language_id
rental_duration
rental_rate
length
replacement_cost
last_update
special_features

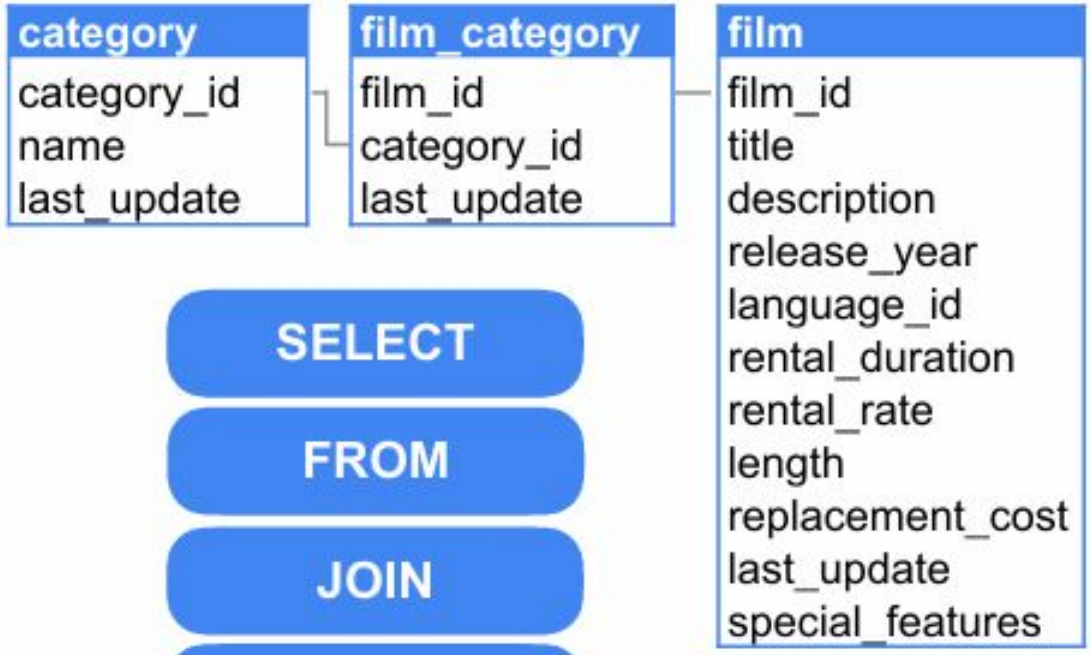
- SELECT
- FROM
- JOIN
- WHERE
- ORDER BY
- LIMIT

Monkey in
Ancient India

In [6]:

```
%%sql
SELECT *
FROM film
JOIN film_category
ON film.film_id = film_category.film_id
JOIN category
ON film_category.category_id = category.category_id
WHERE length<60
```

date	film_id_1	category_id	last_update_1	category_id_1	name	last_update_2
2-15	2	11	2006-02-15	11	Horror	2006-02-15



SELECT

FROM

JOIN

WHERE

ORDER BY

LIMIT

INNER JOIN

JOIN: combine the records from both tables that have a matching column value specified in the ON statement.

film has a row with film_id = 123
film_category does not have a row with film_id= 123

The row with film_id = 123 will not be in the join results

category
category_id
name
last_update

film_category
film_id
category_id
last_update

film
film_id
title
description
release_year
language_id
rental_duration
rental_rate
length
replacement_cost
last_update
special_features

SELECT

FROM

JOIN

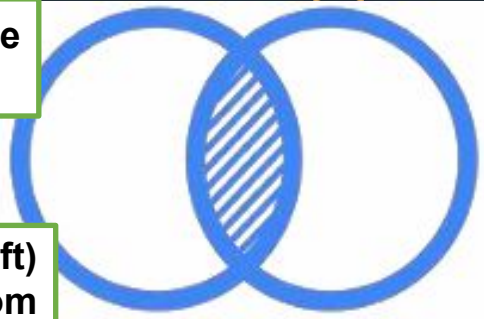
WHERE

ORDER BY

LIMIT

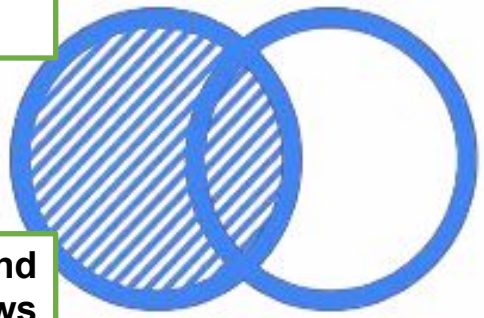
Returns only the rows that have matching values in both tables.

INNER JOIN



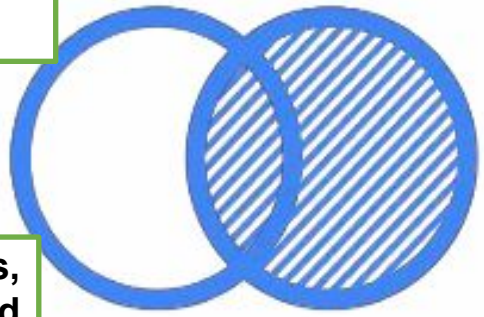
Returns all rows from the first (left) table, and the matching rows from the second (right) table

LEFT JOIN



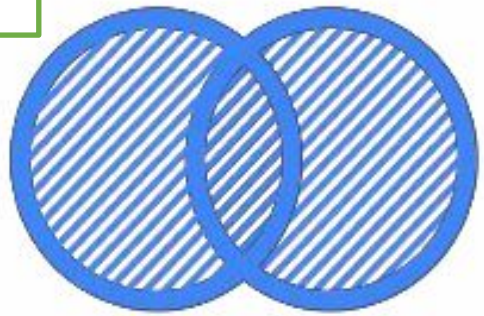
Returns all rows from the second (right) table, and the matching rows from the first (left) table.

RIGHT JOIN



Returns all rows from both tables, combining matches where available and filling with NULL where there is no match.

FULL JOIN



category
category_id
name
last_update

film_category
film_id
category_id
last_update

film
film_id
title
description
release_year
language_id
rental_duration
rental_rate
length
replacement_cost
last_update
special_features

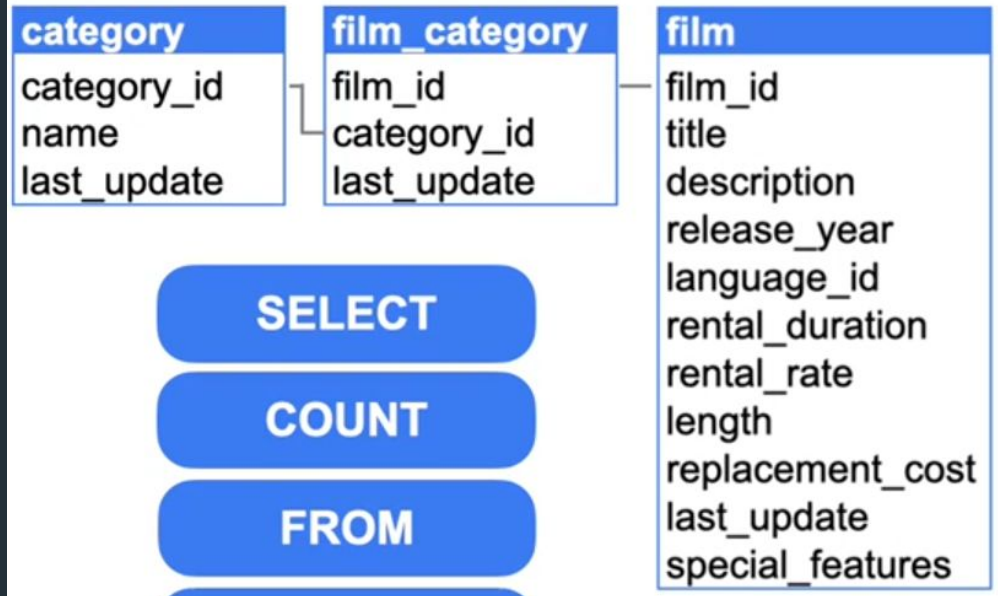
- SELECT
- COUNT
- FROM
- JOIN
- WHERE
- GROUP BY
- ORDER BY
- LIMIT

GROUP BY is used to group rows that have the same values in one or more columns and then apply an aggregate function (like COUNT, SUM, AVG, MAX, MIN) to each group.

In [7]:

```
%%sql
SELECT film.title, category.name
FROM film
JOIN film_category
ON film.film_id = film_category.film_id
JOIN category
ON film_category.category_id = category.category_id
WHERE length<60
```

DAWN POND	Games
DEEP CRUSADE	Documentary
DESTINY SATURDAY	New
DIVORCE SHINING	Sports
DOCTOR GRAIL	Children
DOORS PRESIDENT	Animation



- SELECT
- COUNT
- FROM
- JOIN
- WHERE
- GROUP BY
- ORDER BY
- LIMIT

In [8]:
```
%%sql
SELECT category.name, COUNT(*) AS film_count
FROM film
JOIN film_category
ON film.film_id = film_category.film_id
JOIN category
ON film_category.category_id = category.category_id
WHERE length<60
GROUP BY category.name
ORDER BY film_count DESC
```

```
 * mysql+pymysql://root:***@localhost:3306/sakila
16 rows affected.
```

Out[8]:

name	film_count
Documentary	11
Action	10
Children	9
Family	9

Common SQL Commands

- SELECT
- COUNT
- FROM
- JOIN
- WHERE
- GROUP BY
- ORDER BY
- LIMIT

Data Manipulation Operations

- CREATE
- INSERT INTO
- UPDATE
- DELETE

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

NoSQL Databases

NoSQL Databases

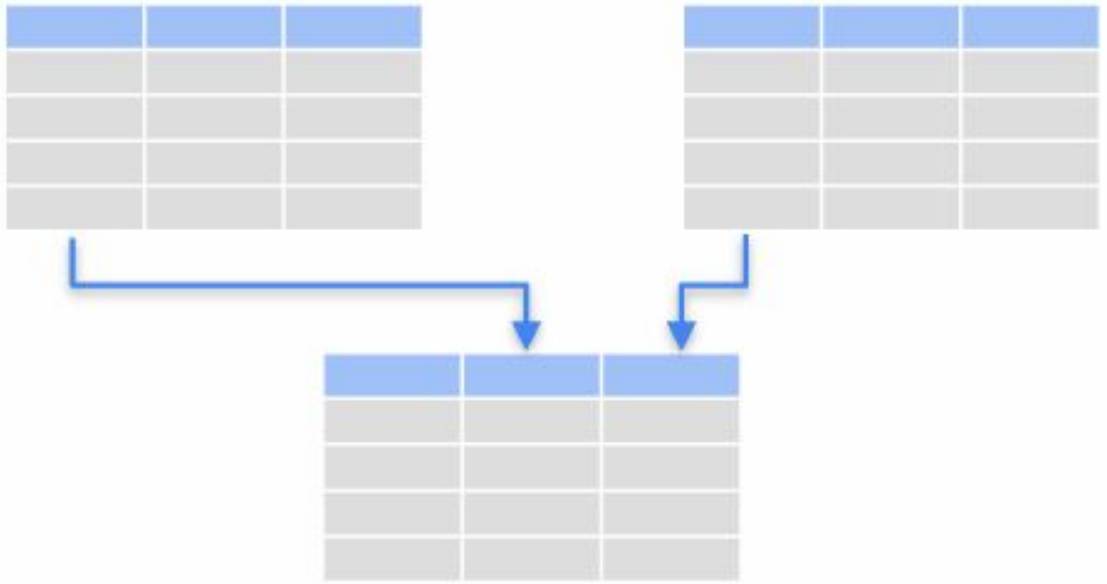
NoSQL was created to handle large, distributed, and flexible data more efficiently. Non-tabular structures (not just rows & columns).

Not Only SQL

Non-Relational Databases

It can still support SQL or SQL-like query languages.

Relational Databases

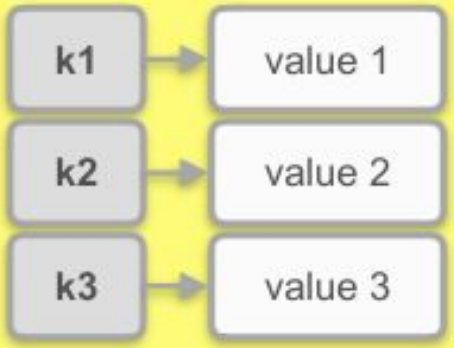


No fixed schema → more flexibility.
Horizontally scalable → spreads data across multiple servers/nodes.

NoSQL Databases

Non-tabular structures

Key-Value

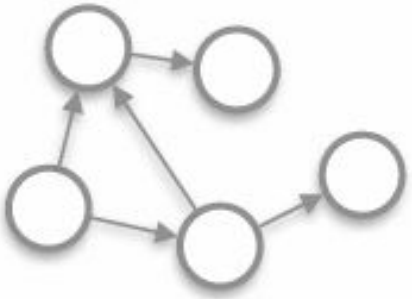


Document

```
{
  "firstName": "Joe",
  "lastName" : "Reis",
  "age": 10,
  "address": {
     "city": "Los Angeles",
     "postalCode": 90024,
     "country": "USA"
  }
}
```

Wide-Column

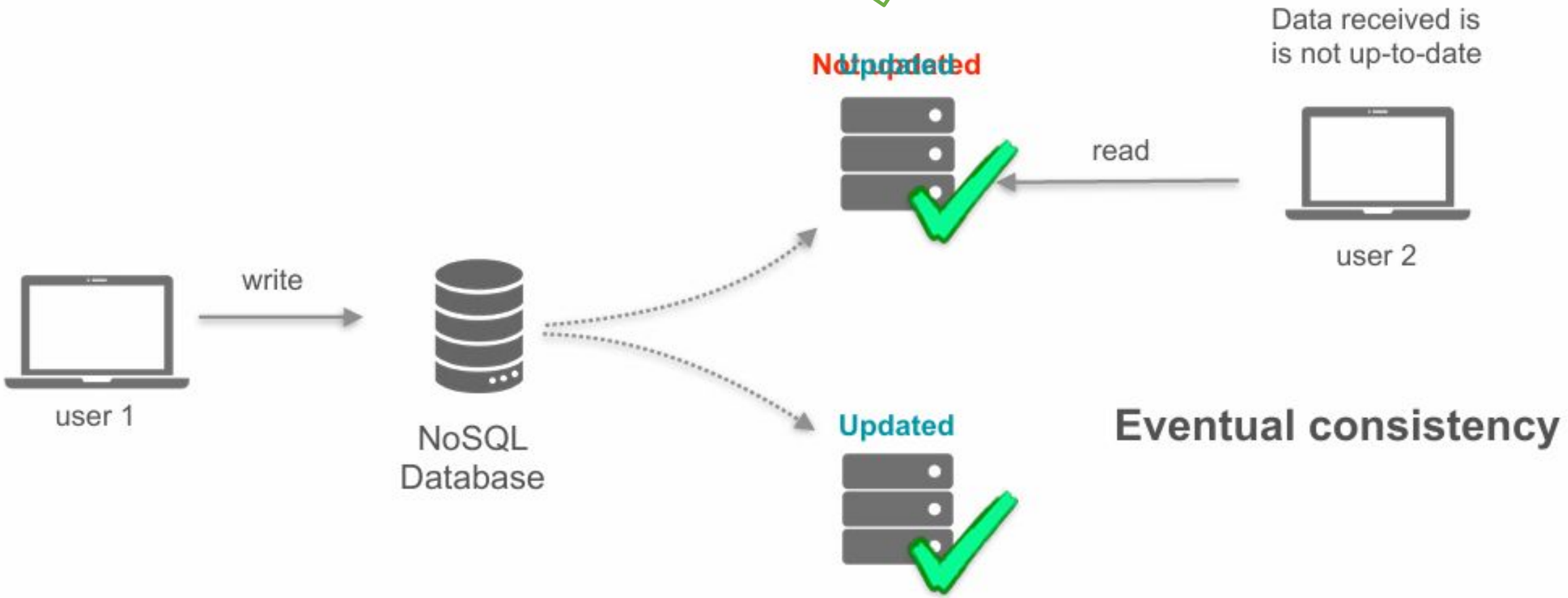
Graph



- No predefined schemas
- More flexibility when storing your data

Horizontal Scaling

Strong consistency = correctness first (but slower).
Eventual consistency = speed and scale first (but may show outdated data).



Consistency

NoSQL Databases	Relational Databases
Eventual Consistency	Strong Consistency
• Speed is prioritized • System availability and scalability are important	• Read data only when all nodes have been updated

NoSQL Databases

Not all NoSQL databases guarantee:

ACID compliance

Atomicity

Consistency

Isolation

Durability

Many NoSQL systems **relaxed one or more ACID properties** to achieve better speed and fault tolerance across distributed nodes.

Instead, they often follow BASE:

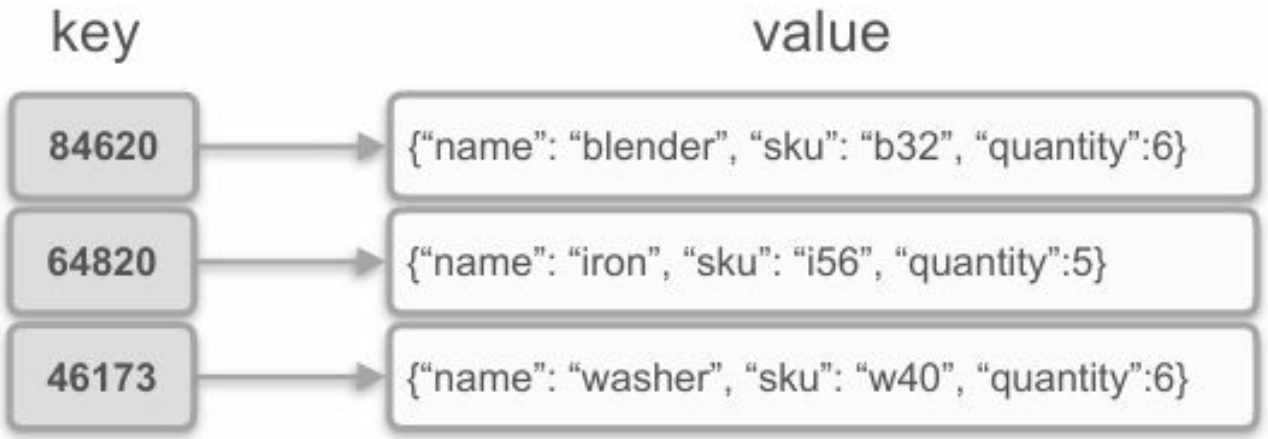
Basically Available – system is always available.

Soft state – state may change over time (not immediately consistent).

Eventually consistent – data will become consistent given enough time.

Types of No-SQL Databases

Key-Value

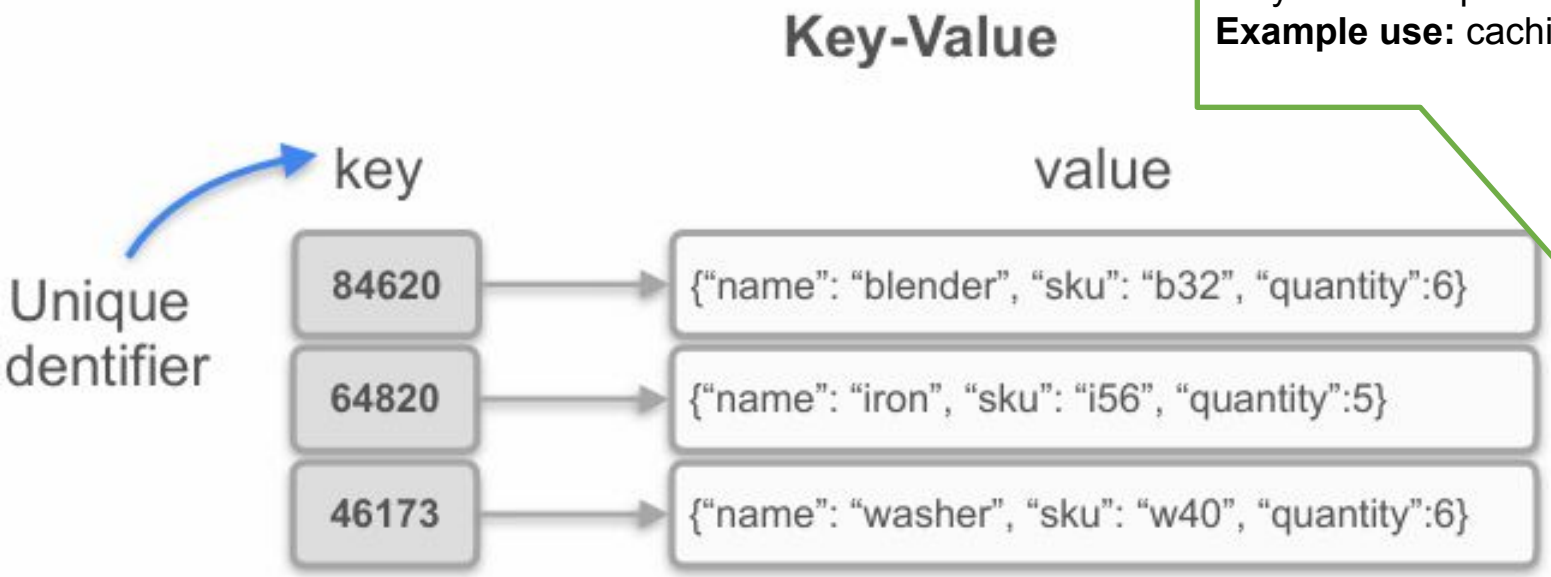


Document

```
{
  “firstName”: “Joe”,
  “lastName” : “Reis”,
  “age”: 10,
  “address”: {
     “city”: “Los Angeles”,
     “postalCode”: 90024,
     “country”: “USA”
  }
}
```

Key-Value Database

Store data as key-value pairs (like Python dictionaries or JSON).
Very fast lookups.
Example use: caching user session data in apps.



Fast lookup: such as caching user session data

- viewing different products
- adding items to the shopping cart
- checking out

Document Database

Data stored in JSON-like documents inside collections (like tables).Easier to fetch all info about one entity (e.g., a user).No joins supported → combining data across documents is harder.

Collection *(Like a table)*

keys

```
{
 "users" : [
    {
    "id": 1234,
    "name": {
              "first":"Joe",
              "last":"Reis"
              },
    "favorite_bands" : ["AC/DC", "Slayer", "WuTang Clan", "Action Bronson" ]
    },
    {
     "id":1235,
     "name": {
               "first": "Matt",
               "last":"Housley"
              },
     "favorite_bands" : ["Dave Matthews Band", "Creed", "Nickelback"]
    }
 ]
}
```

Single users

Documents

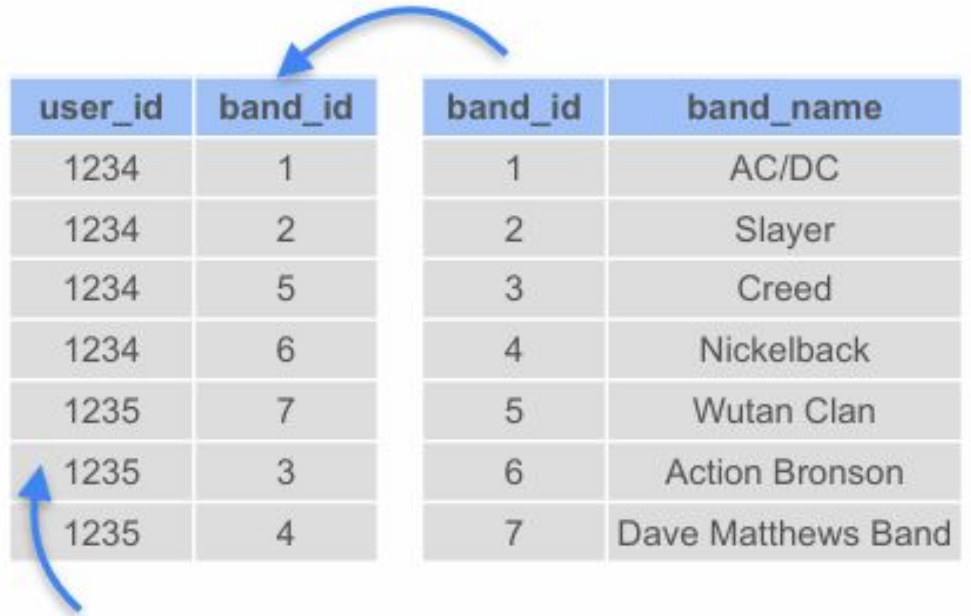
(Like a row)

Document Database

```
{
 "users" : [
          {
           "id": 1234,
           "name": {
                    "first":"Joe",
                    "last":"Reis"
                    },
           "favorite_bands" : ["AC/DC", "Slayer", "WuTang Clan", "Action Bronson" ]
           },

           {
            "id":1235,
            "name": {
                     "first": "Matt",
                     "last":"Housley"
                    },
            "favorite_bands" : ["Dave Matthews Band", "Creed", "Nickelback"]
           }
        ]
}
```

- Easy to retrieve all the information about a user (locality)
- Document stores don't support joins
- Flexible schema



user_id	band_id
1234	1
1234	2
1234	5
1234	6
1235	7
1235	3
1235	4

band_id	band_name
1	AC/DC
2	Slayer
3	Creed
4	Nickelback
5	Wutan Clan
6	Action Bronson
7	Dave Matthews Band

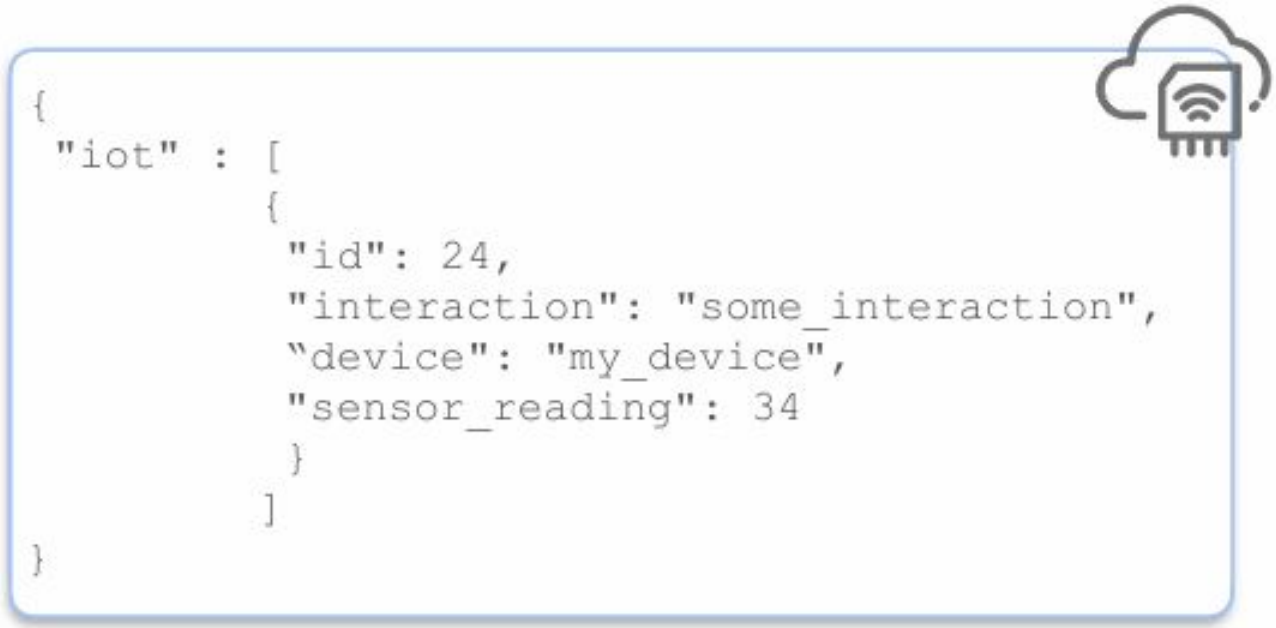
user_id	first_name	last_name
1234	Joe	Reis
1235	Matt	Housely

Fixed schema

Document Database

Use cases

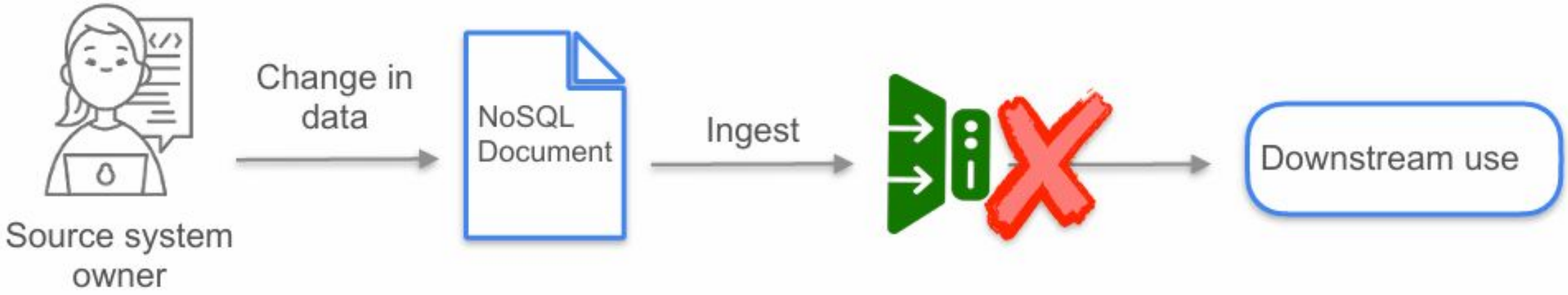
- Content management
- Catalogs
- Sensor readings



Flexible Schema

Document Database

Document databases become absolute nightmares to manage and query.



RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Database ACID Compliance

OLTP Systems

Online Transaction Processing

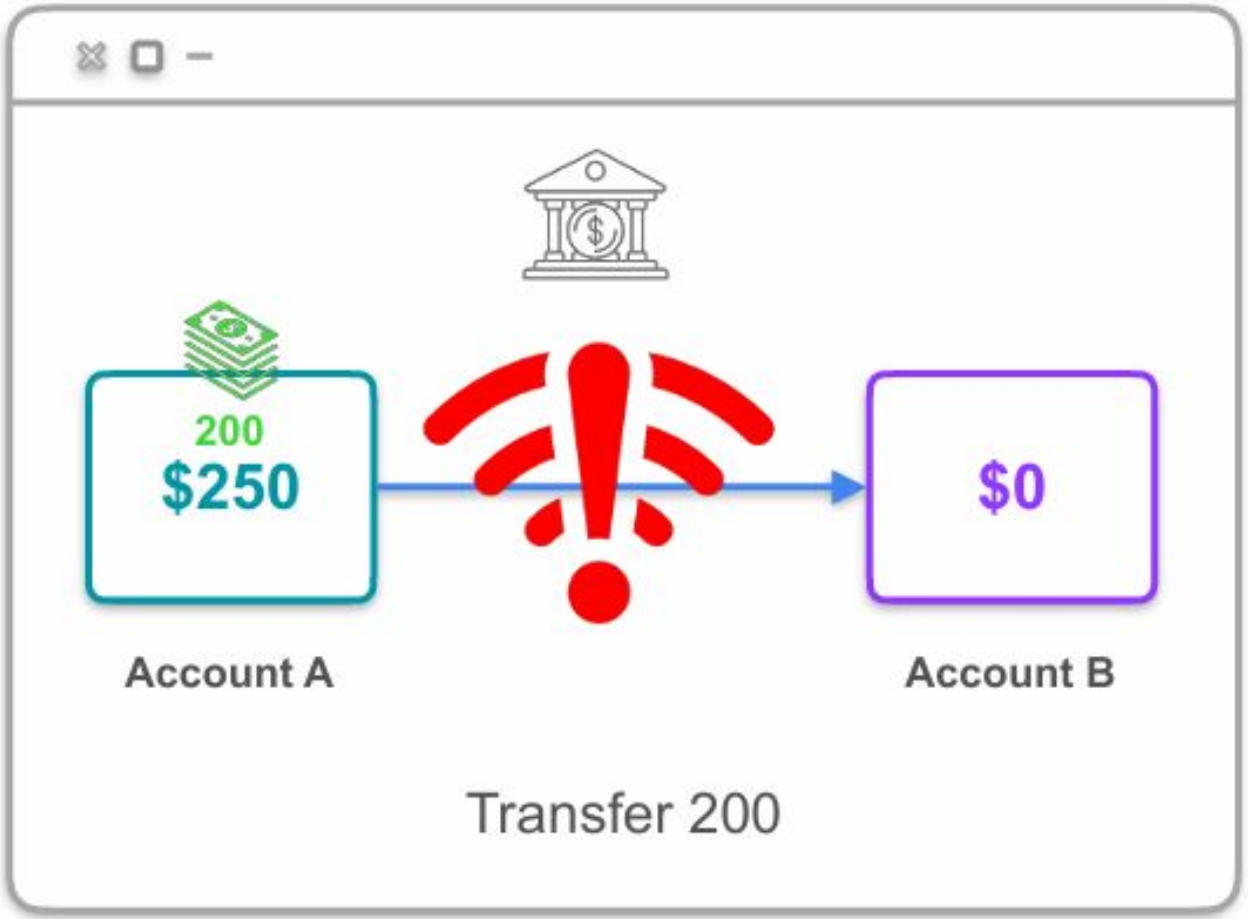
Support very high transaction rates (bank account balances, online orders)

ACID Compliance

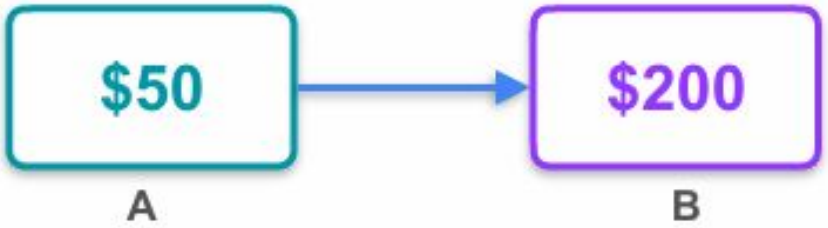
Relational Databases	NoSQL Databases
ACID compliant	Not ACID compliant by default
Atomicity	
Consistency	
Isolation	
Durability	

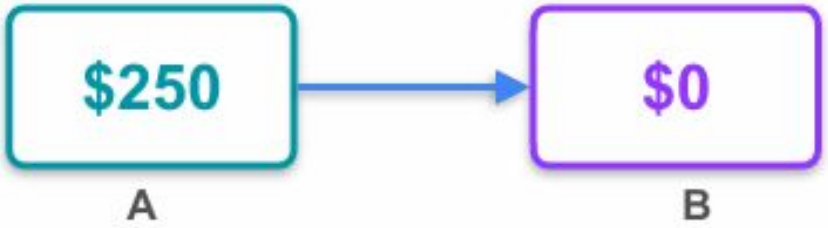
They help ensure transactions are processed reliably and accurately in an OLTP system.

ACID Compliance

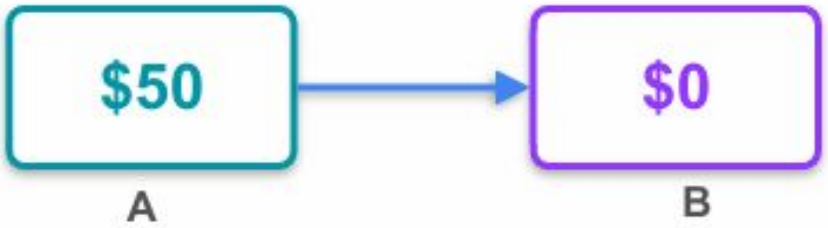


You'd be hoping





But not



Atomicity

Atomicity ensures that transactions are **atomic**, treated as a single, indivisible unit.

A transaction = a single indivisible unit.
Either all operations succeed or none do.

Atomicity

Atomicity ensures that transactions are **atomic**, treated as a single, indivisible unit.

A transaction: placing an order

Both operations must happen as a single transaction

Atomicity

Atomicity ensures that transactions are **atomic**, treated as a single, indivisible unit.

A transaction: placing an order

Both operations must happen as a single transaction

Database must move from one valid state to another (obeying rules/constraints).Example: If schema says stock can't go below 0, then trying to order more items than available fails → database remains valid.

Atomicity Atomicity ensu ransactions are **atomic**, treated as a single, indivisible unit.

Consistency Any changes to the data made within a transaction follow the set of rules or constraints defined by the database schema.

id	product_name	quantity
1	blender	1

Buy 2 blenders

Transaction

id	product_name	quantity
1	blender	-1

Rule: stock level ≥ 0

Atomicity Atomicity ensures that transactions are **atomic**, treated as a single, indivisible unit.

Consistency Any changes to the data made within a transaction follow the set of rules or constraints defined by the database schema.

ACID compliance

Atomicity

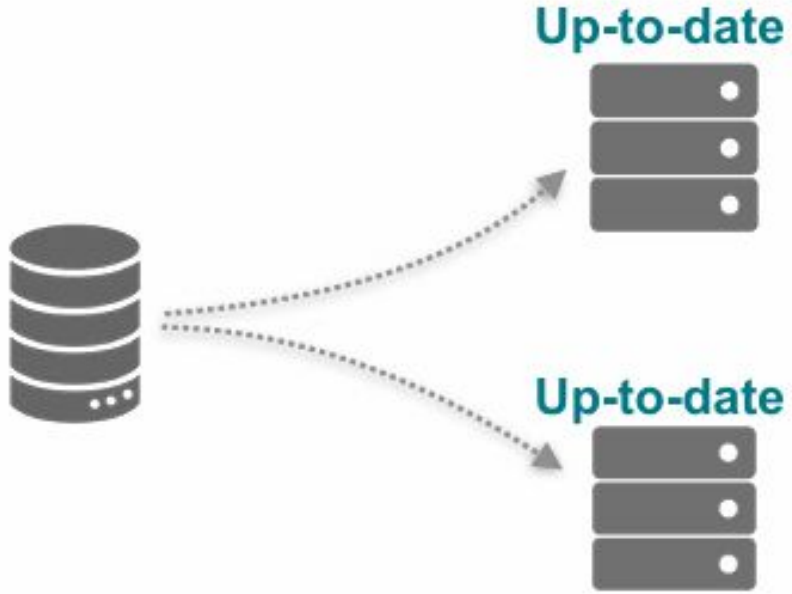
Consistency

Isolation

Durability

Strong Consistency

All nodes provide the same up-to-date

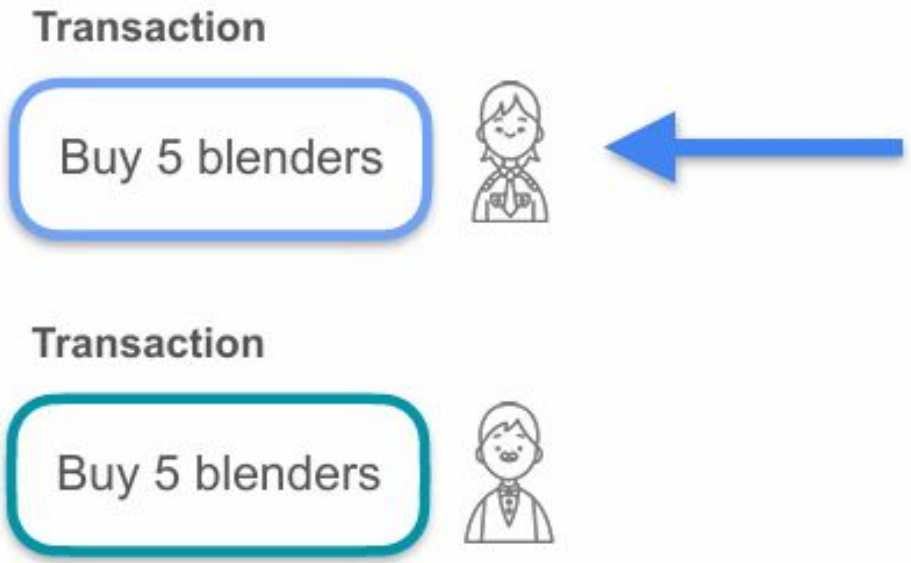


Atomicity Atomicity ensures that transactions are **atomic**, treated as a single, indivisible unit.

Consistency Any changes to the data made within a transaction follow the set of rules or constraints defined by the database schema.

Isolation Each transaction is executed independently in sequential order.

id	product_name	quantity
1	blender	5

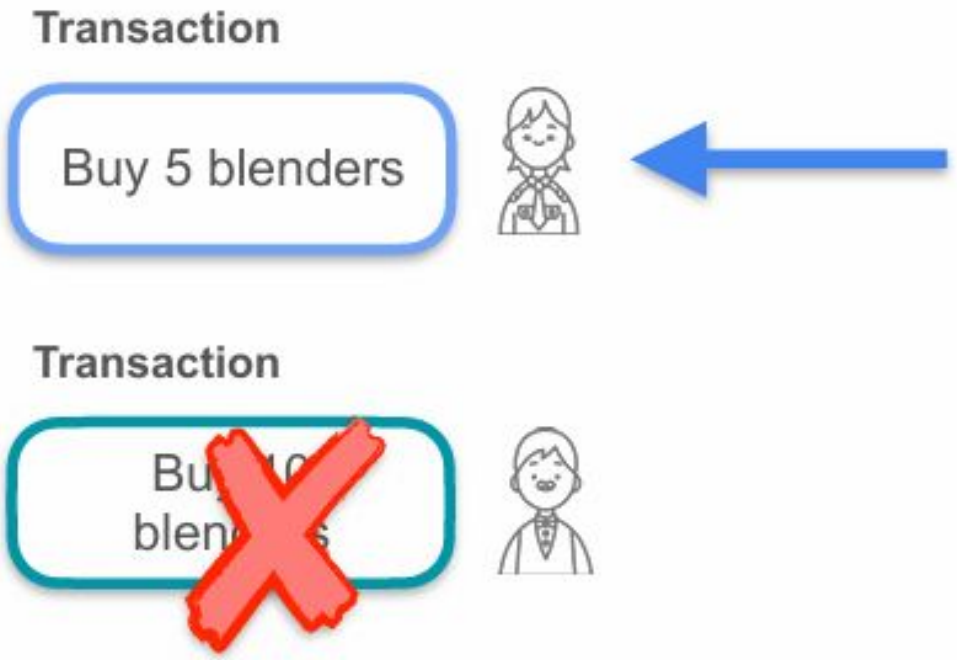


Atomicity Atomicity ensures that transactions are **atomic**, treated as a single, indivisible unit.

Consistency Any changes to the data made within a transaction follow the set of rules or constraints defined by the database schema.

Isolation Each transaction is executed independently in sequential order.

id	product_name	quantity
1	blender	5



Once committed, a transaction is permanent. Even if there's a crash, restart, or power loss, changes are saved (via logs, replication, etc.).

Atomicity Atomicity ensures that trans are **atomic**, treated as a single, indivisible unit.

Consistency Any changes to the data mad in a transaction follow the set of rules or constraints defined by database schema.

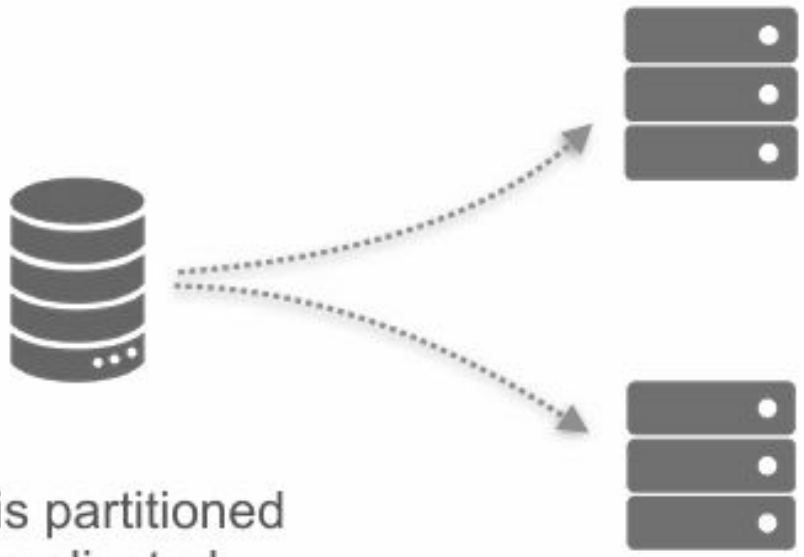
Isolation Each transaction is executed ind pendently in sequential order.

Durability Once a transaction is completed, its effects are permanent and will survive any subsequent system failures.

Essential for maintaining the reliability of the database

ACID Compliance

The ACID principles guarantee that a database will maintain a consistent picture of the world.



Data is partitioned or replicated

Strong Consistency

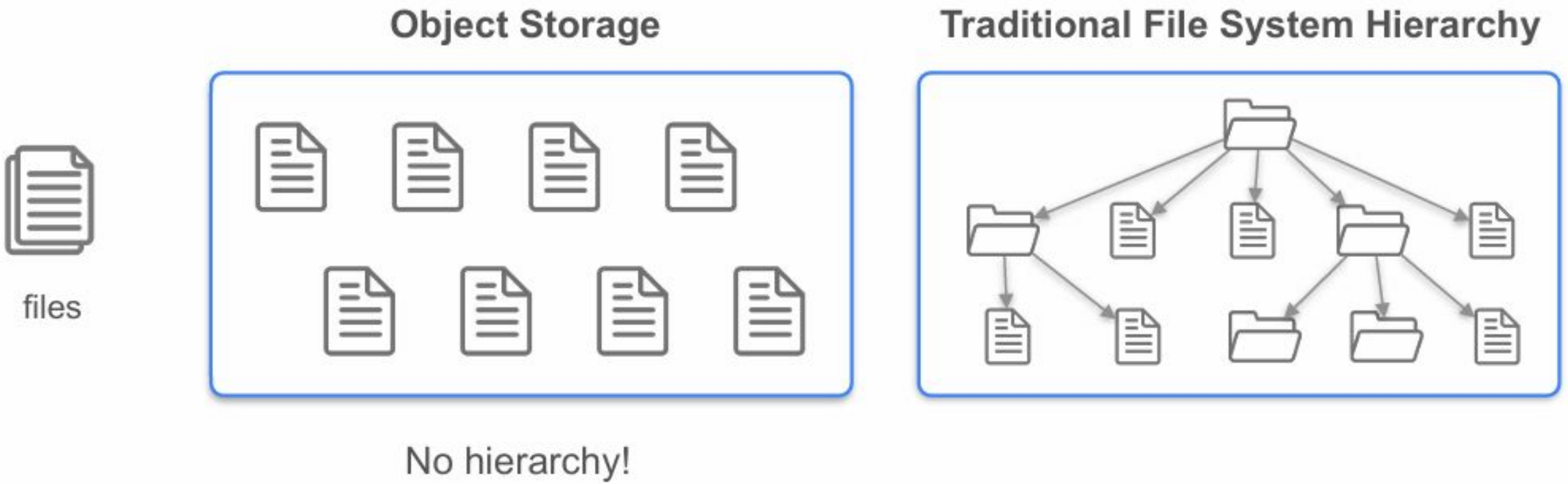
- Data is consistent across the entire network
- Key feature of relational databases that ensures ACID

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Object Storage

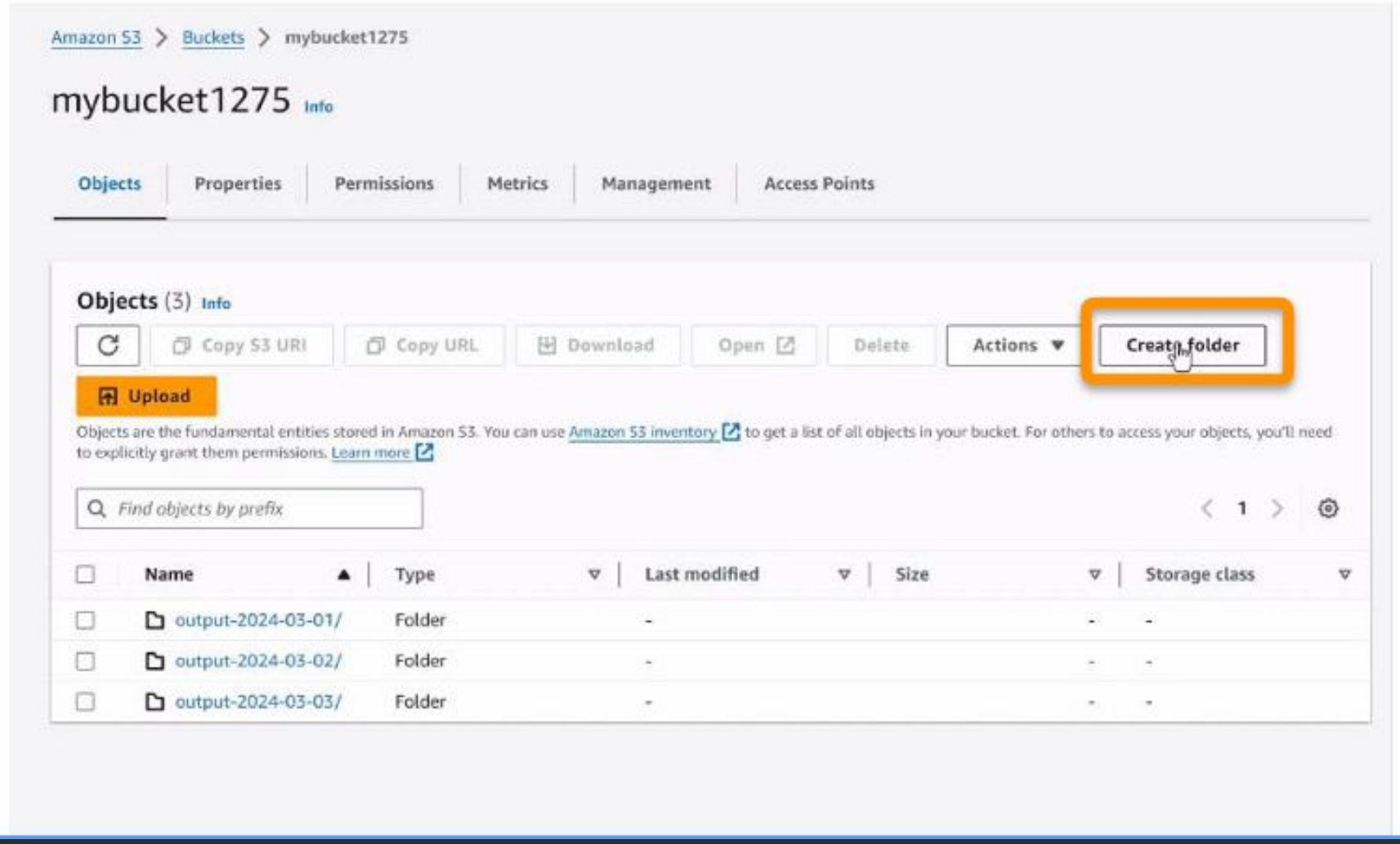
Object Storage

Object storage treats files (objects) as individual items stored in a flat structure, unlike hierarchical file systems with folders/subfolders.
Examples: Amazon S3, Google Cloud Storage, Azure Blob Storage.



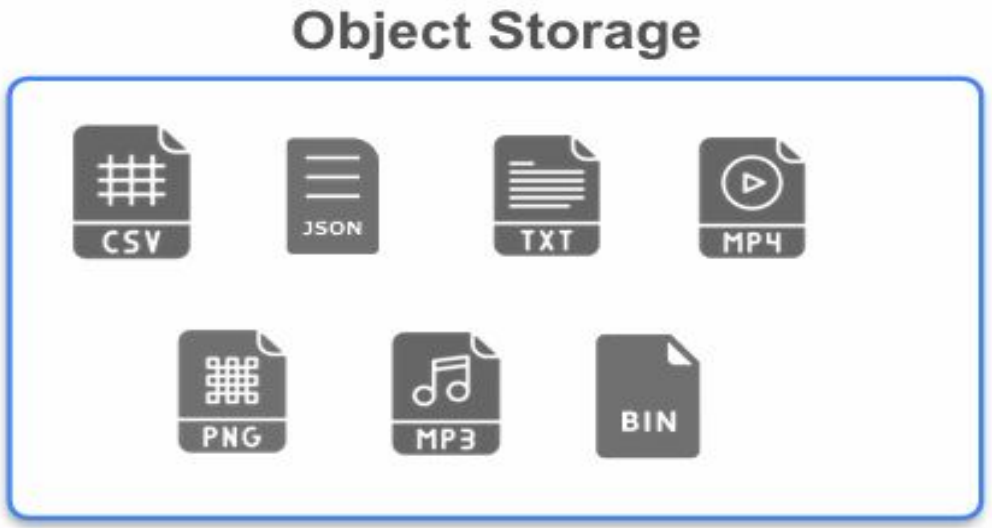
Object Storage

Amazon S3



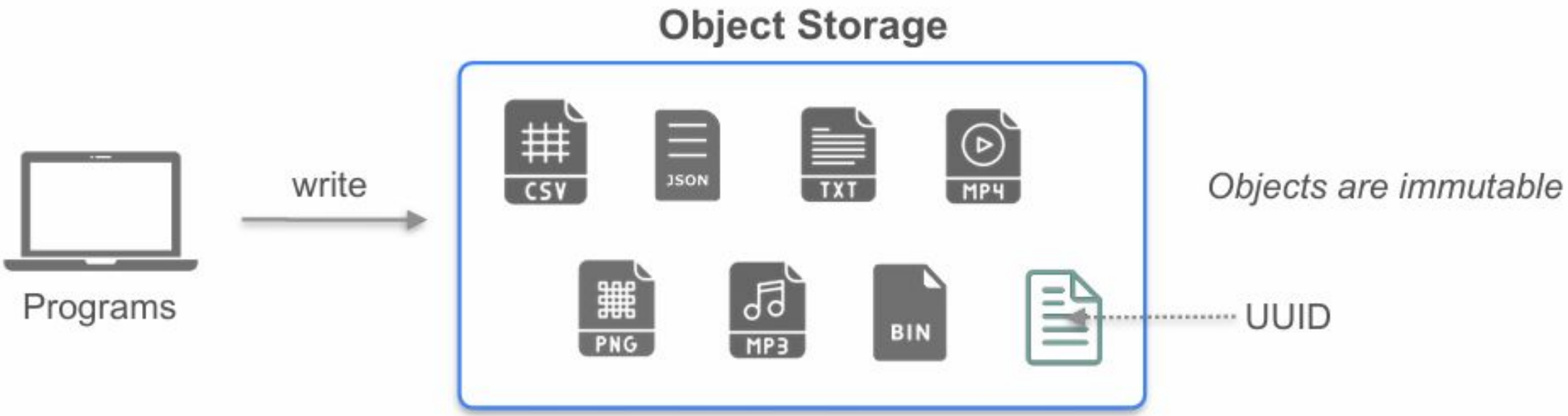
Object Storage

files



- Storing semi-structured and unstructured data
- Serving data for training machine learning models

Object Storage

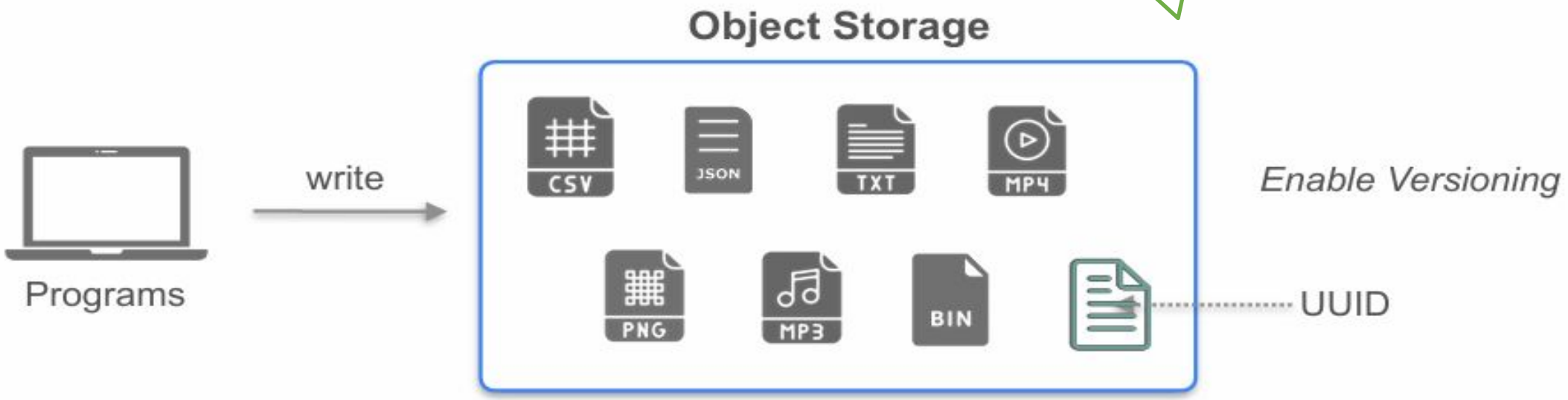


For each object,

- Universal Unique Identifier or UUID (key)
- Metadata: creation date, file type, owner

Each object = data + UUID key + metadata.
objects are immutable → cannot be updated/edited in place.

Object Storage



For each object,

- Universal Unique Identifier or UUID (key)
- Metadata: creation date, file type, owner, version

Why Use Object Storage?

- Store files of various data formats without a specific file system structure
- Easily scale out to provide virtually limitless storage space
- Replicate data across several availability zones

Amazon S3

99.999999999% : data durability

- Cheaper than other storage options

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Logs

A **log** is a record of information about events happening in a system or application.

Logs

Software Application

Logs

01-01-2025:10.30	67945	success	user added a product x to their cart
01-01-2025:10.32	38910	fail	invalid values typed for product quantity
01-01-2025:10.38	17462	fail	customer table corrupted

Exhaust / Byproduct

Monitoring or Debugging a system

- User activity:
  - Signing in
  - navigating to a particular page
- An update to a database
- An error from a procedure

Log An append-only sequence of records ordered by time, capturing information about events that occur in systems.

Rich data source

Downstream use cases

Web Server Logs
Detailed user activity data

Analysis of user behavior patterns

Database System Logs

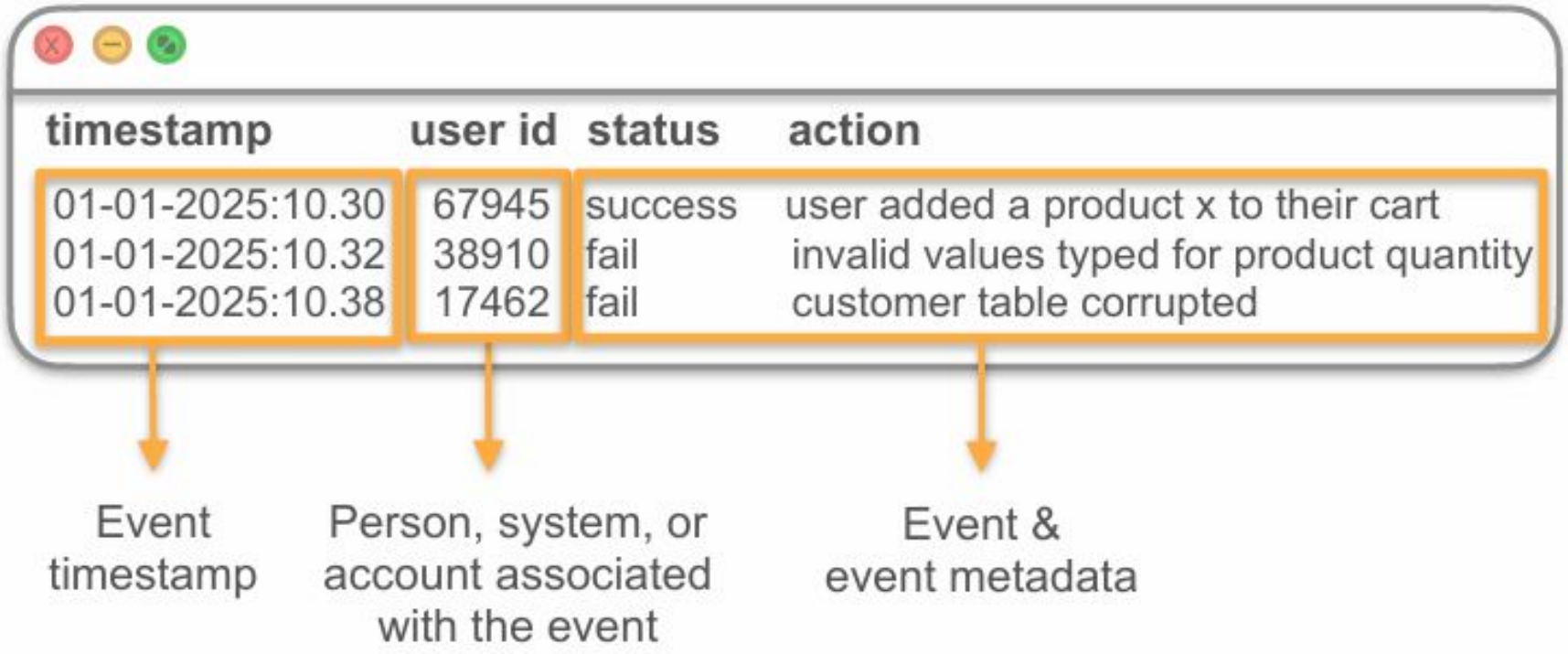
Track changes in source database

Security System Logs

Machine Learning *anomaly detection*

Log

An append-only sequence of records ordered by time, capturing information about events that occur in systems.



Log An append-only sequence of records ordered by time, capturing information about events that occur in systems.

timestamp	user id	status	action
01-01-2025:10.30	67945	success	user added a product x to their cart
01-01-2025:10.32	38910	fail	invalid values typed for product quantity
01-01-2025:10.38	17462	fail	customer table corrupted

Log can be stored as JSON or RDBM

```
{
  "user id": 67945,
  "action": "user added a product x to their cart",
  "status": "success",
  "time-stamp": 01-01-2025:10.30
}
```

[00101011 11000101 11001001 11000101 110001001]

user id	action	status	timestamp
67945	user added a product x to their cart	success	01-01-2025:10.30
38910	invalid values typed for product quantity	fail	01-01-2025:10.32
17462	customer table corrupted	fail	01-01-2025:10.38

Log Levels

A tag to categorize the event (log level)

- “debug”
- “info”
- “warn”
- “error”
- “fatal”

user id	action	status	timestamp	level
67945	user added a product x to their cart	success	01-01-2025:10.30	Info
38910	invalid values typed for product quantity	fail	01-01-2025:10.32	error
17462	customer table corrupted	fail	01-01-2025:10.38	fatal

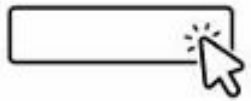
RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Streaming Systems

Terminology

Event

Something that happened in the world or a change to the state of a system.



User clicking on a link

Sensor measuring a temperature change

Message

Stream

Data: record of events

Terminology

Event

Something that happened in the world or a change to the state of a system.

User clicking on a link

Sensor measuring a temperature change

Message

A record of information about an event.

Message
Event Details
Event Metadata
Event Timestamp

Producer

Event | Event | Event | Event

Stream

Terminology

Event

Something that happened in the world or a change to the state of a system.

User clicking on a link

Sensor measuring a temperature change

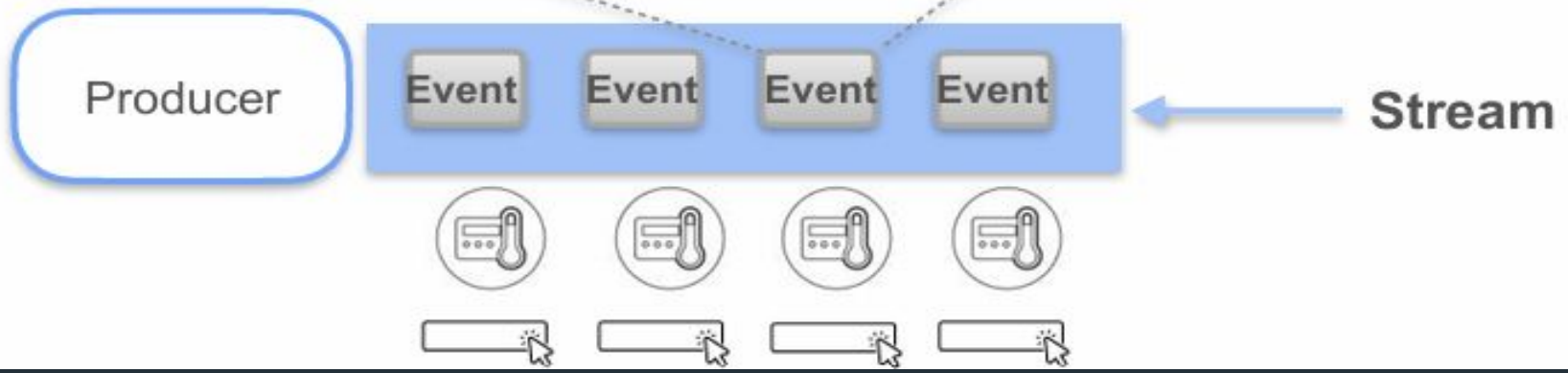
Message

A record of information about an event.

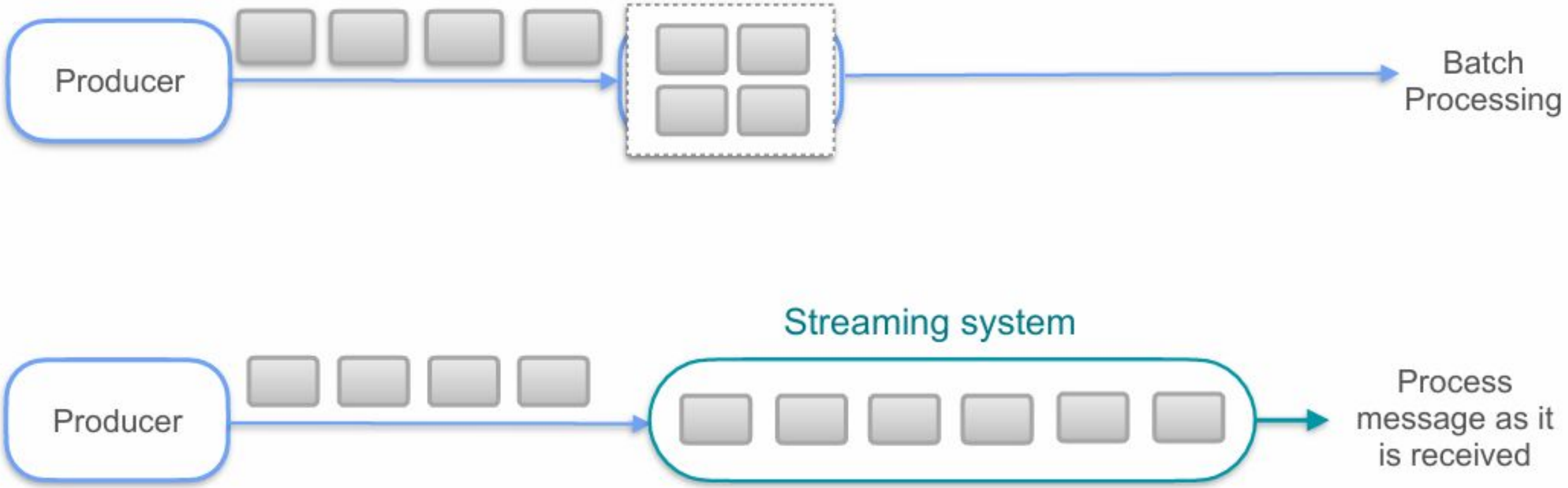
Message
Event Details
Event Metadata
Event Timestamp

Stream

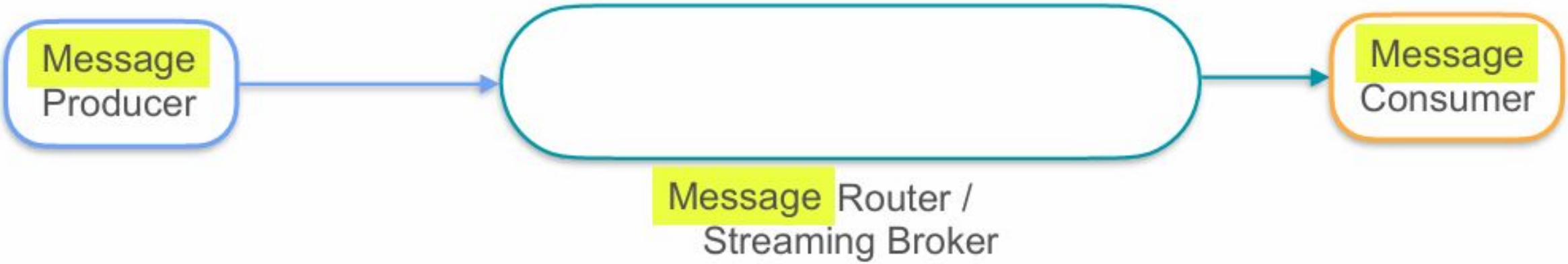
A sequence of messages.



Stream Processing



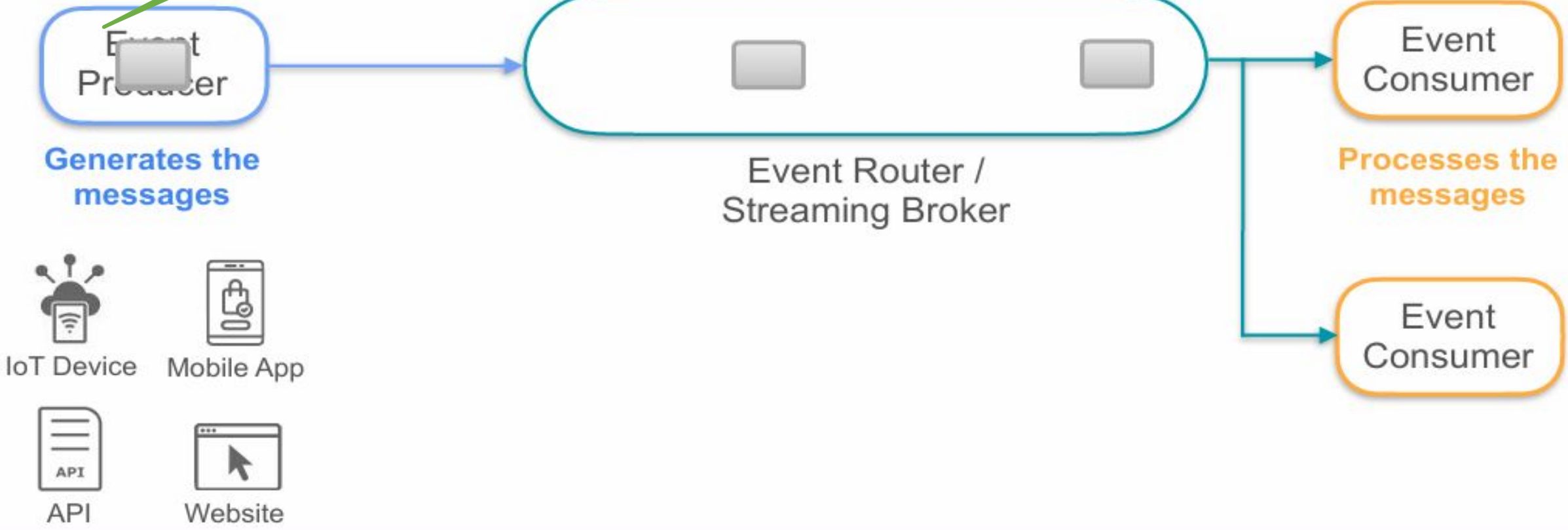
Streaming System



Generates messages (IoT device, app, API, website).

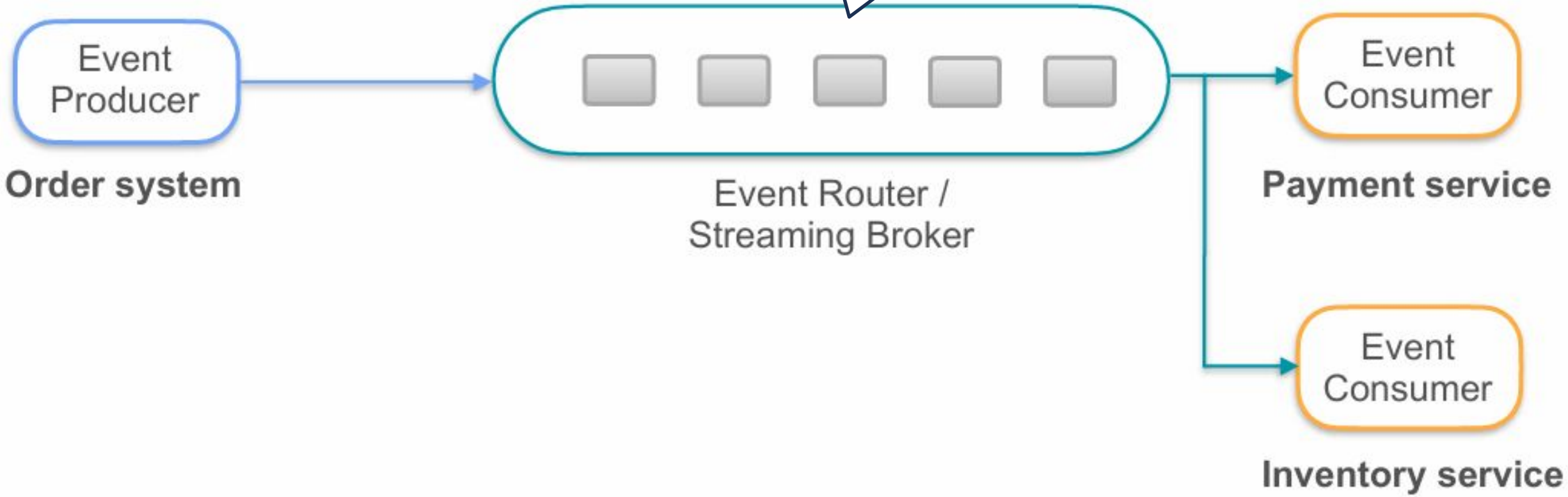
Routes events from producers to consumers. Examples: **Apache Kafka, Amazon Kinesis**.

Streaming System



Streaming System

User places an order → Event is generated.
Payment service consumes the event → Processes payment.
Inventory service consumes the event → Updates stock.
Streaming broker (e.g., Kafka) ensures both services get the right events without the producer waiting.



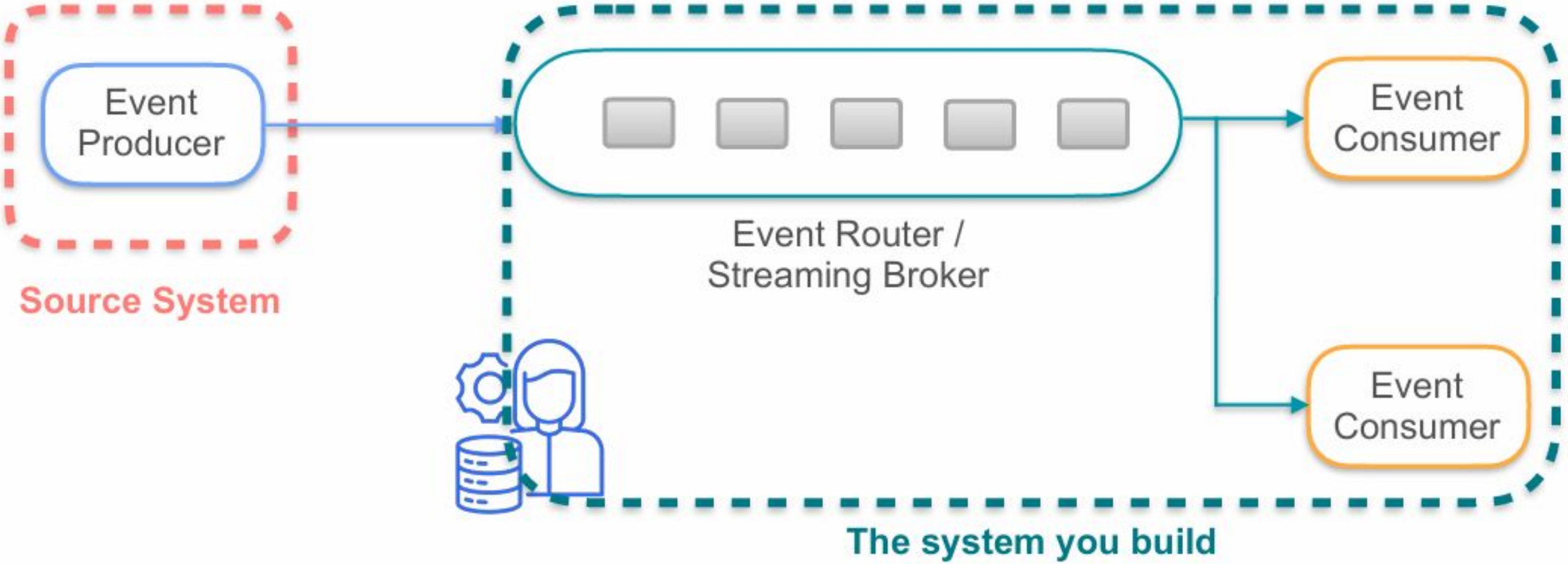
Streaming System

Producer can send messages anytime

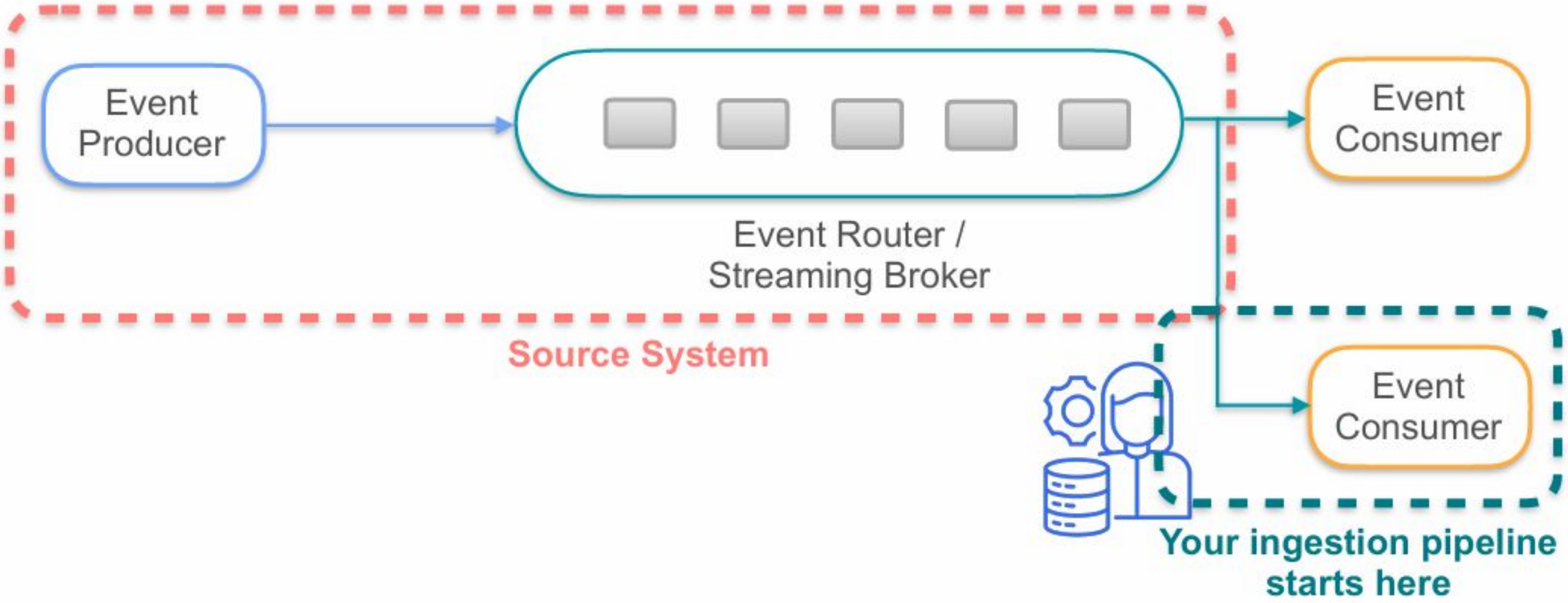
Event Router / Streaming Broker

- Acts as a buffer to filter and distribute the messages
- Decouples producer from consumer
- Prevents message from being lost

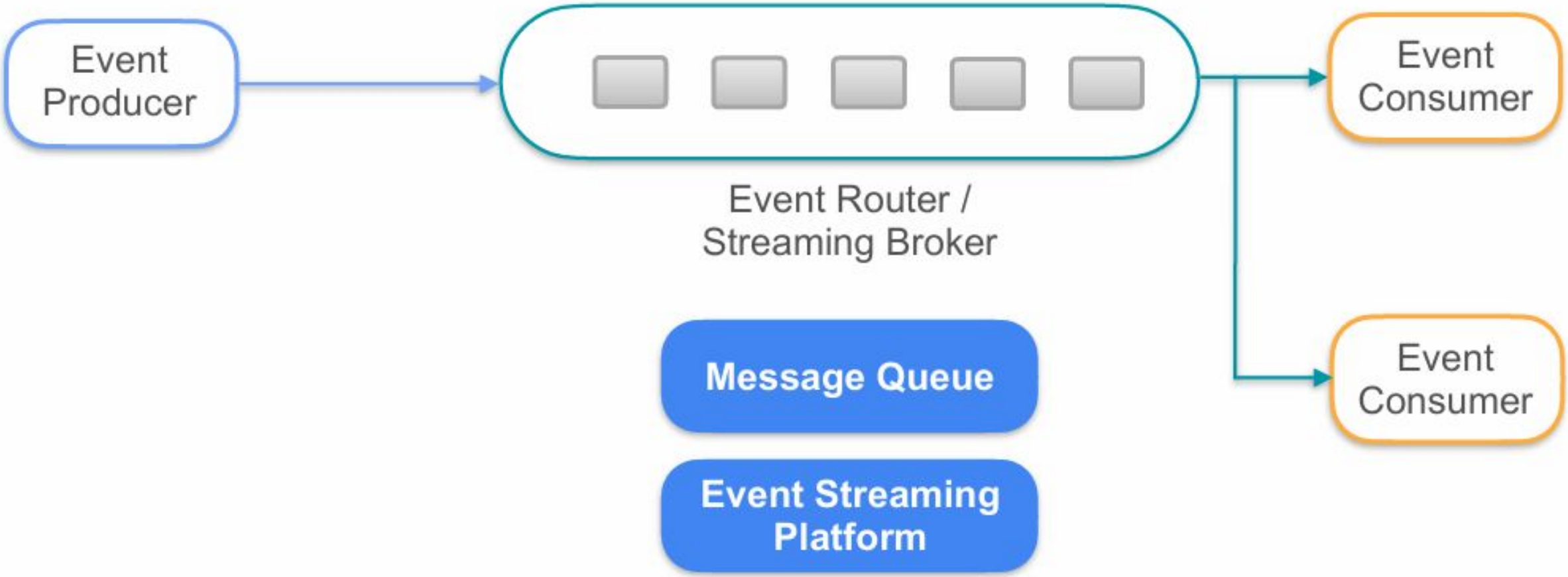
Your Data System



Your Data System



Streaming System



Message Queue

A queue/buffer that accumulates messages

Message Queue

A queue/buffer that accumulates messages
and delivers those messages to consumers asynchronously.

Message Queue

A queue/buffer that accumulates messages
and delivers those messages to consumers asynchronously.

First-in first-out (FIFO) basis

Router = Queue (FIFO: First-In, First-Out).
Once consumed → message deleted.
Acts like temporary storage between producer and consumer.
Example: Amazon SQS.
Commonly used in microservices and serverless applications.

Message Queue

A queue/buffer that accumulates messages and delivers those messages to consumers asynchronously.

First-in first-out (FIFO) basis

Temporary storage

Amazon Simple Queue Service (Amazon SQS)

Event Streaming Platform

Log: Append-only record of events

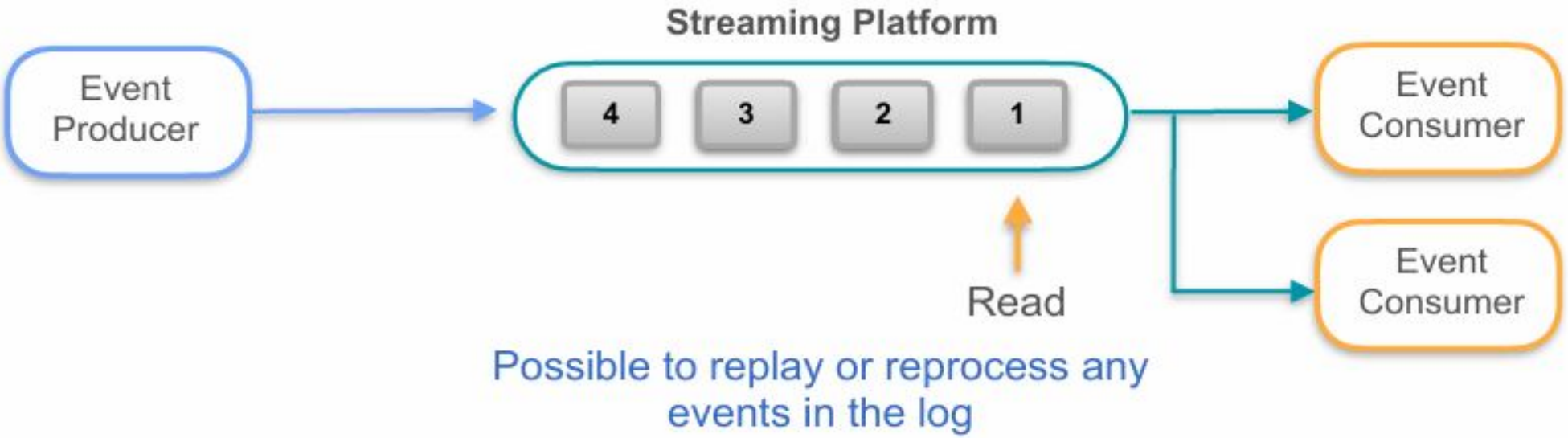
kafka

Amazon Kinesis Data Streams

Event Streaming Platform

Log: Append-only

Stores all messages in a log; multiple consumers can read independently



Possible to replay or reprocess any events in the log

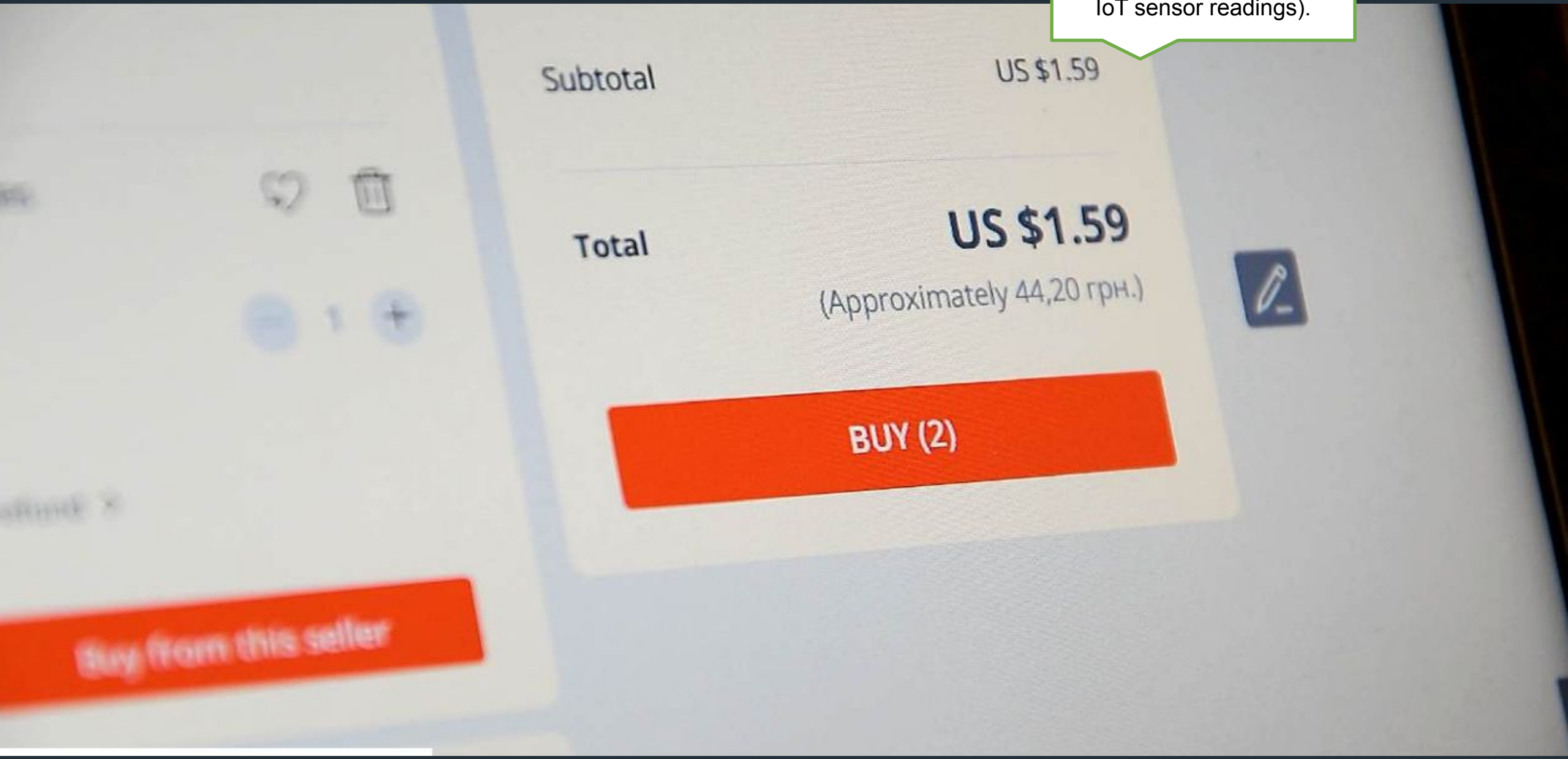
kafka

Amazon Kinesis Data Streams

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Data Ingestion Overview

All data = stream of events at its source (stock price updates, user clicks, IoT sensor readings).



Data Ingestion

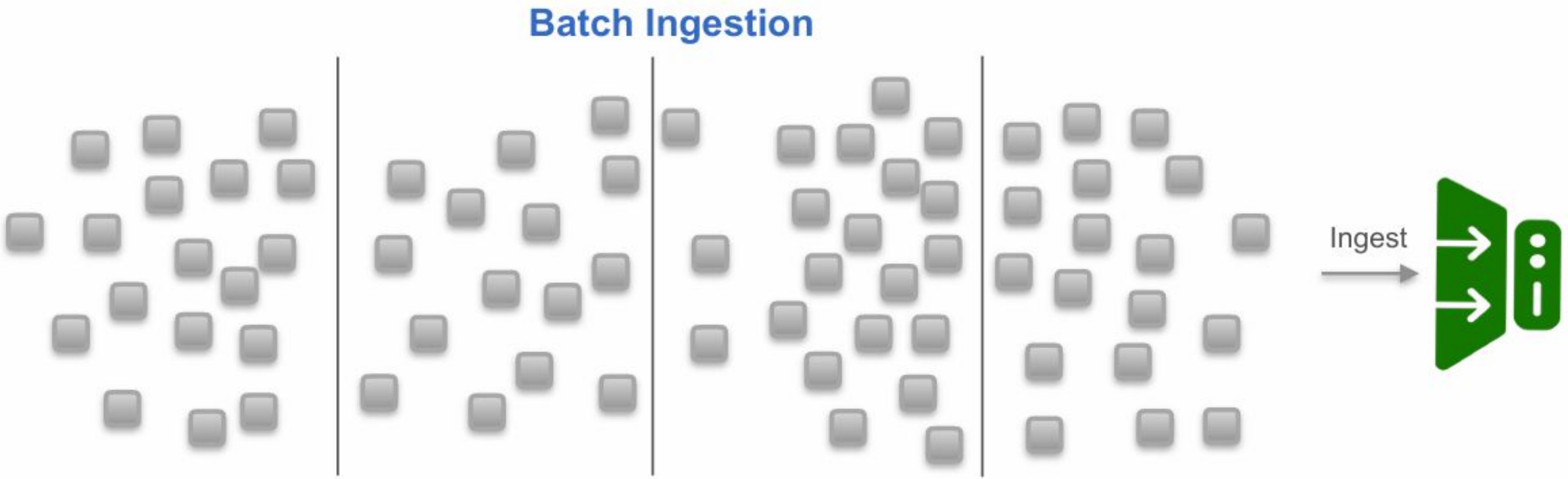
Unbounded Data: continuous stream of events

Data Ingestion

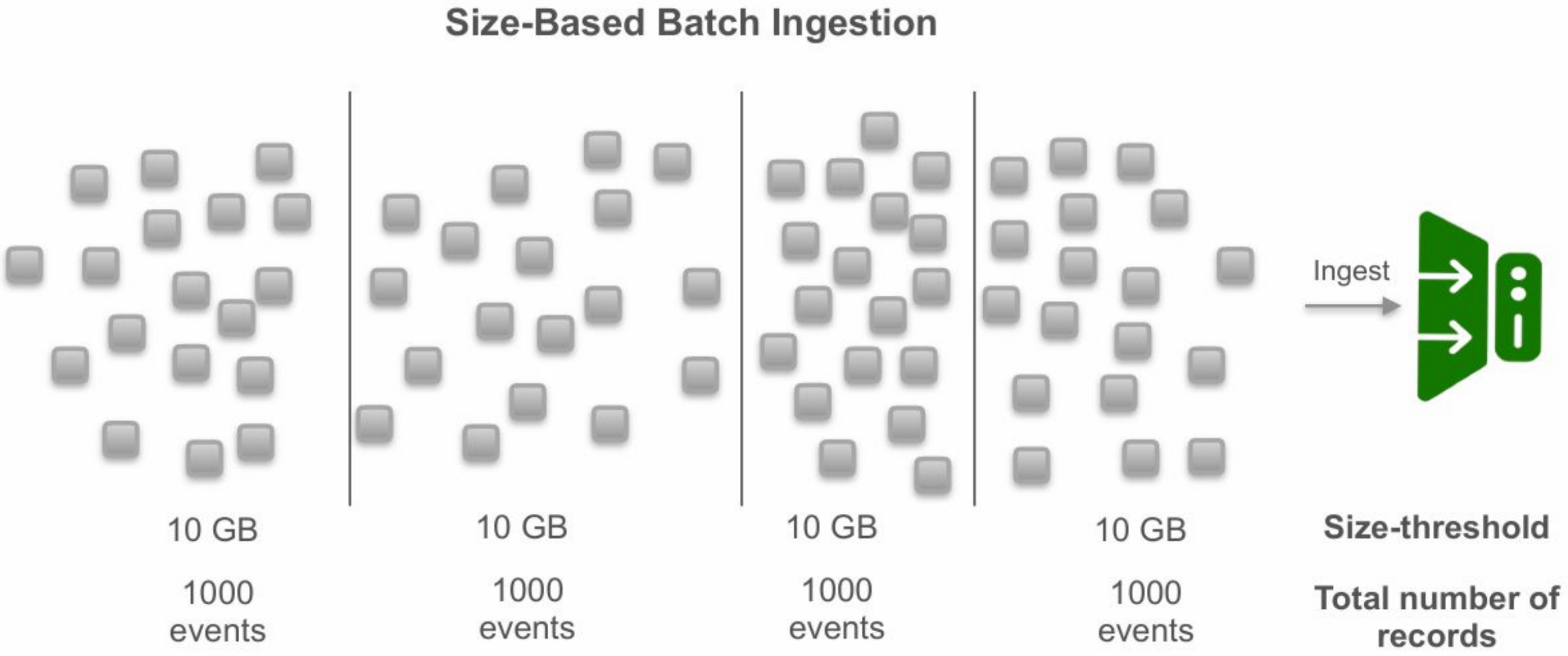
Stream ingestion → Handle events individually, in real time.
Batch ingestion → Collect events into a bounded set (time, size, or count), then process as a unit.

Stream Ingestion

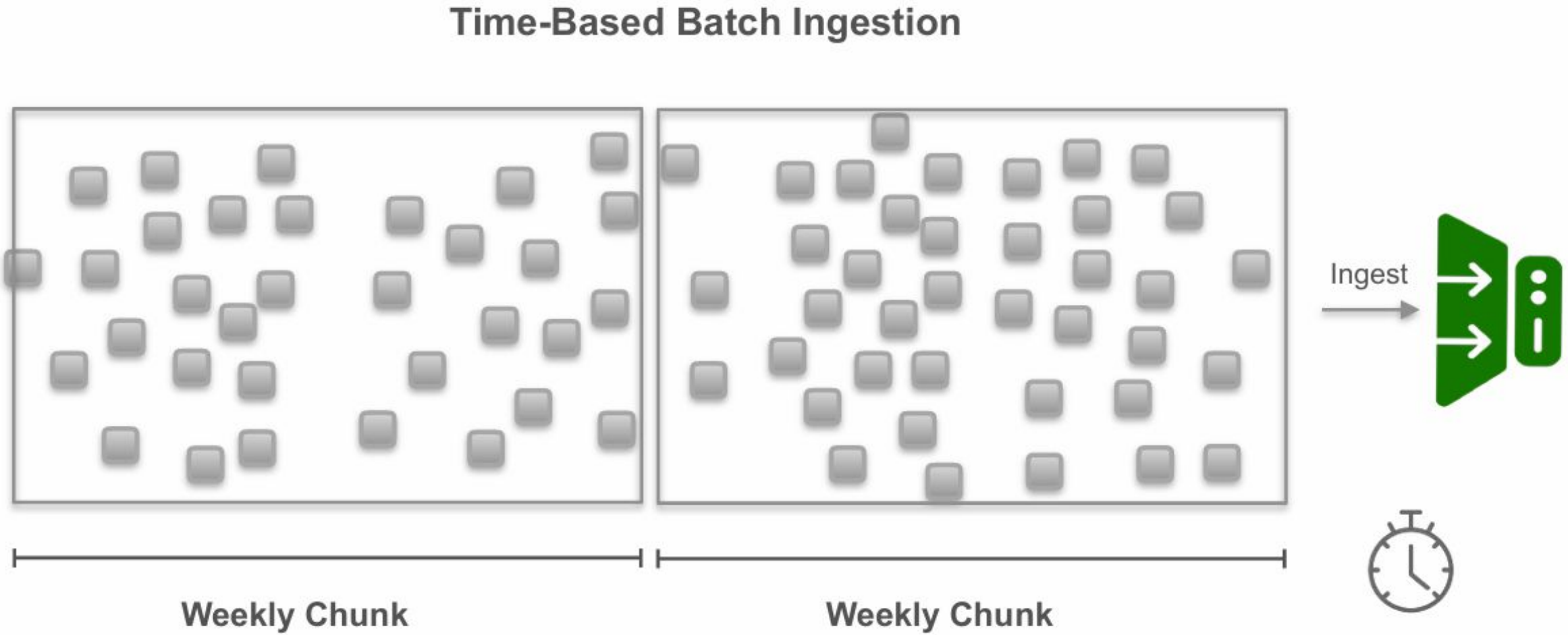
Data Ingestion



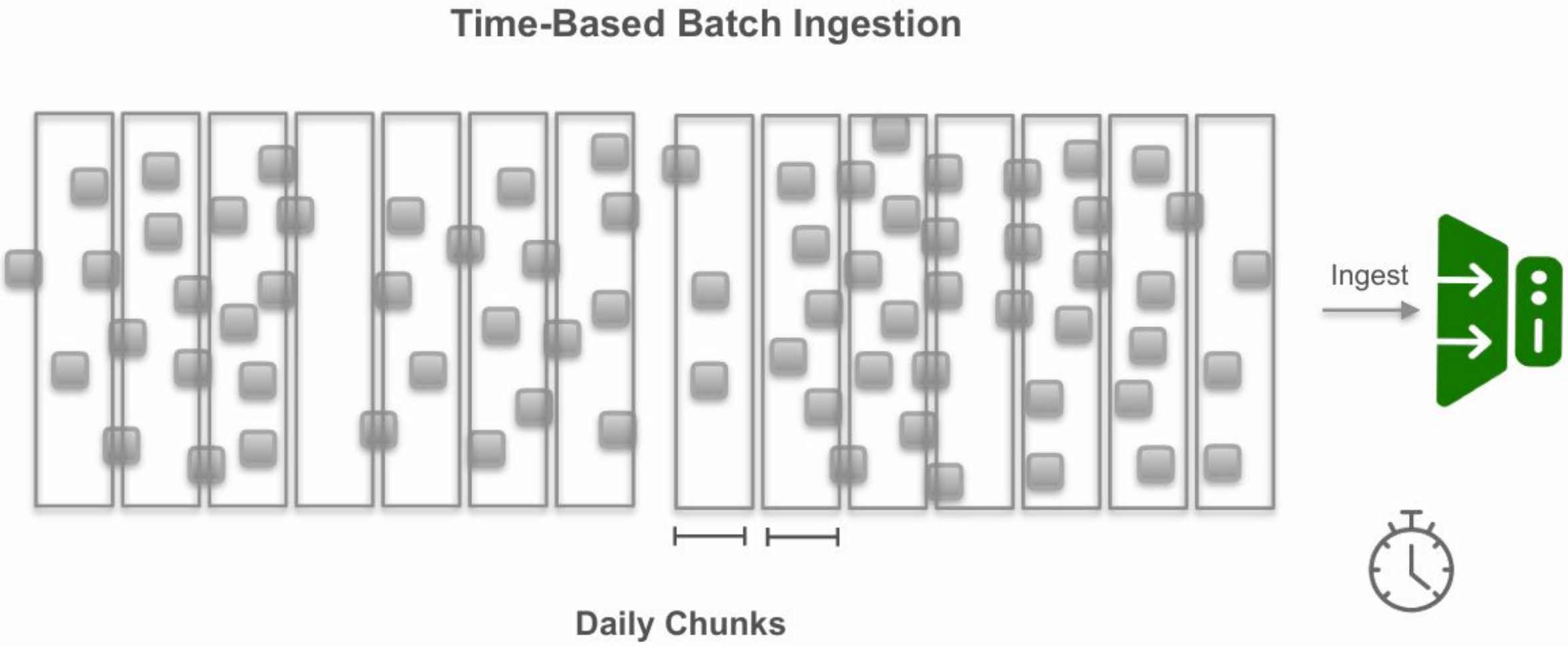
Data Ingestion



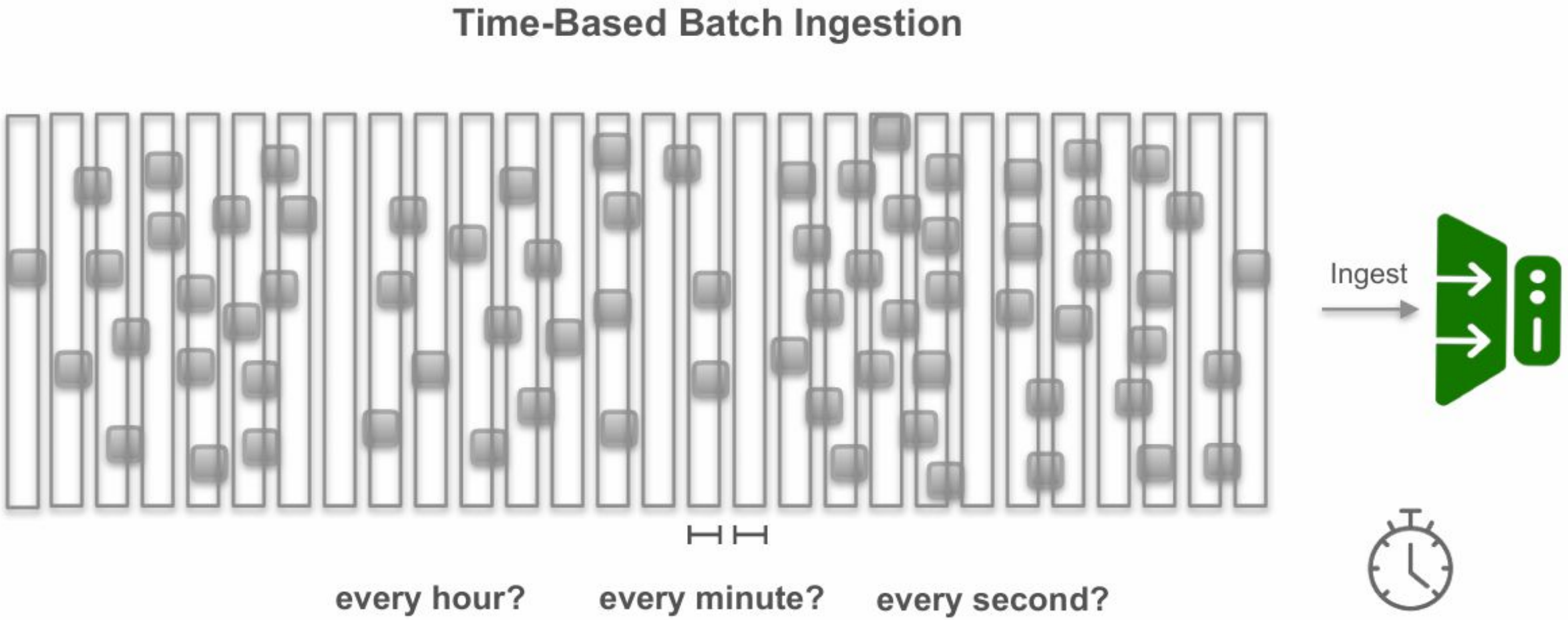
Data Ingestion



Data Ingestion

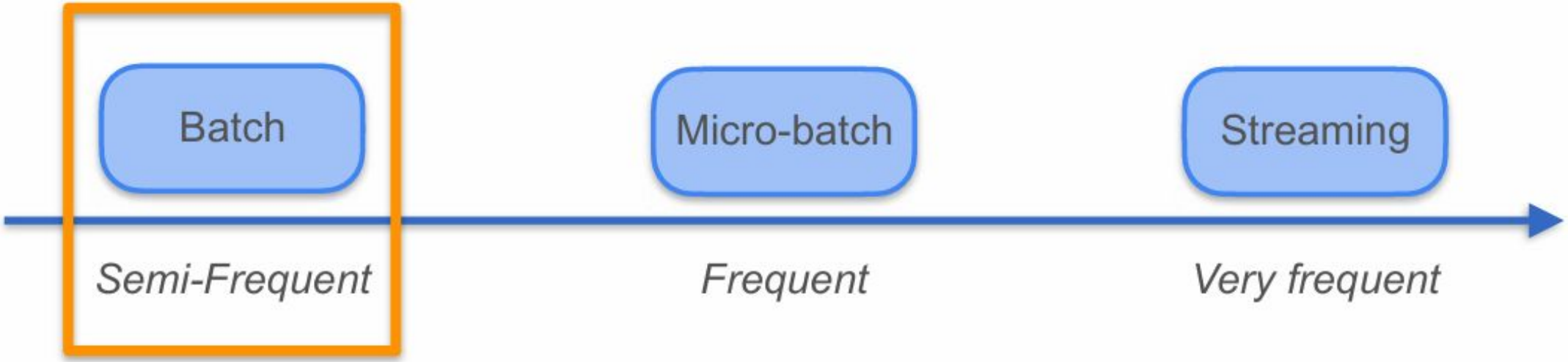


Data Ingestion



Ingestion Frequencies

Process **small batches** of data at **frequent intervals**. (e.g., every 1s, 10s, 1min).

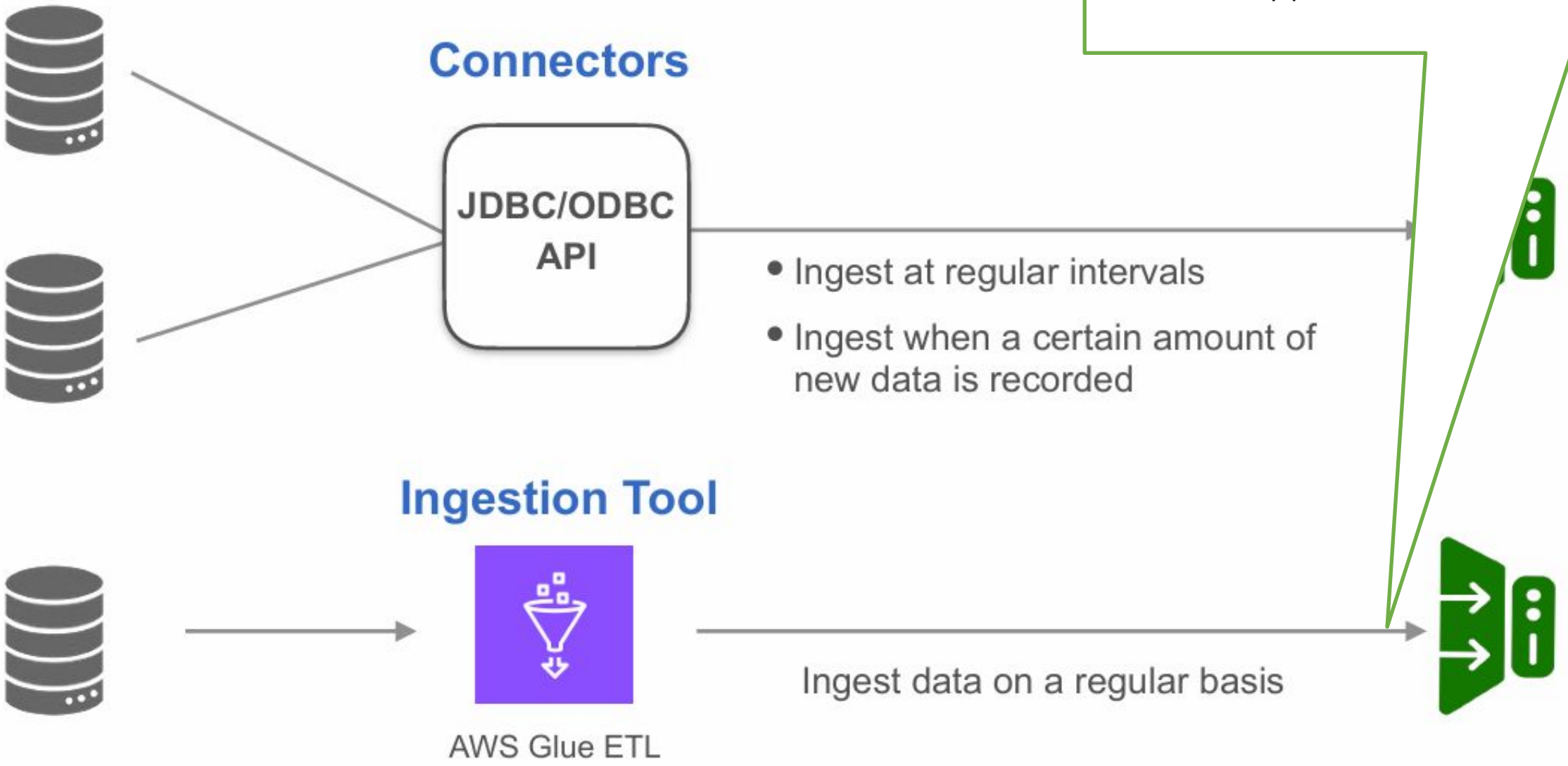


Choice of ingestion frequency depends on:

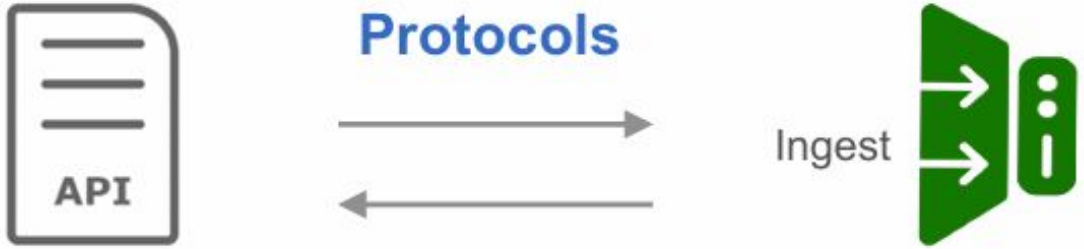
- the source systems you're working with
- the end use case

Ways to Ingest Data from Databases

AWS Glue = **serverless ETL + data catalog** → simplifies building scalable, automated data pipelines on AWS.



Ways to Ingest Data from APIs



- How much can you ingest in one go?
- How frequently can you call the API?

Reading API documentation

Communicating with data owners

Writing custom API connection code

Ways to Ingest Data from Files

Manual File Download

Secure File Transfer

File Transfer Protocols

SFTP: Secure File Transfer Protocol

SCP: Secure Copy Protocol

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Batch Ingestion

Goals of the Marketing Analyst

Frequency of request is limited

Analysis of Historical Trends

No need for real-time analysis

Batch Ingestion Patterns

ETL — **Extract - Transform - Load**

Tradeoffs

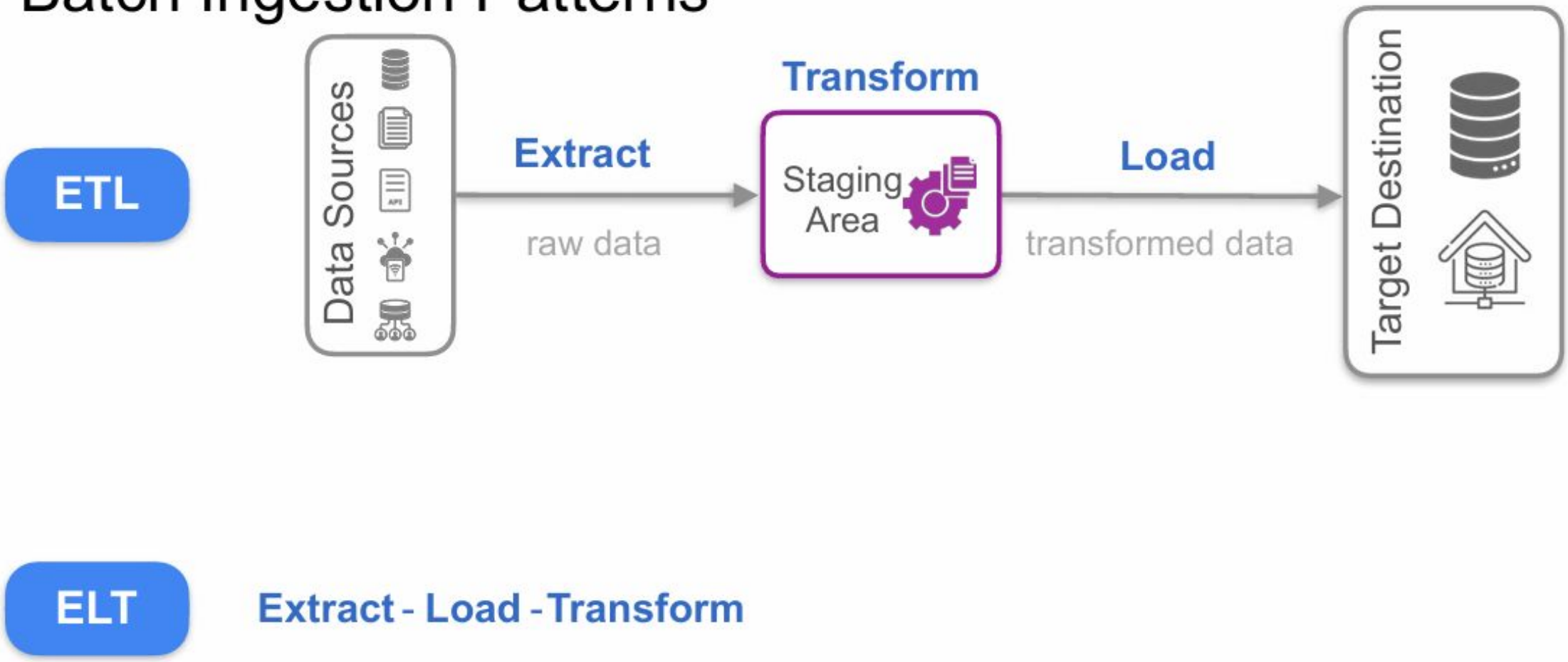
ELT — **Extract - Load - Transform**

Batch Ingestion Patterns

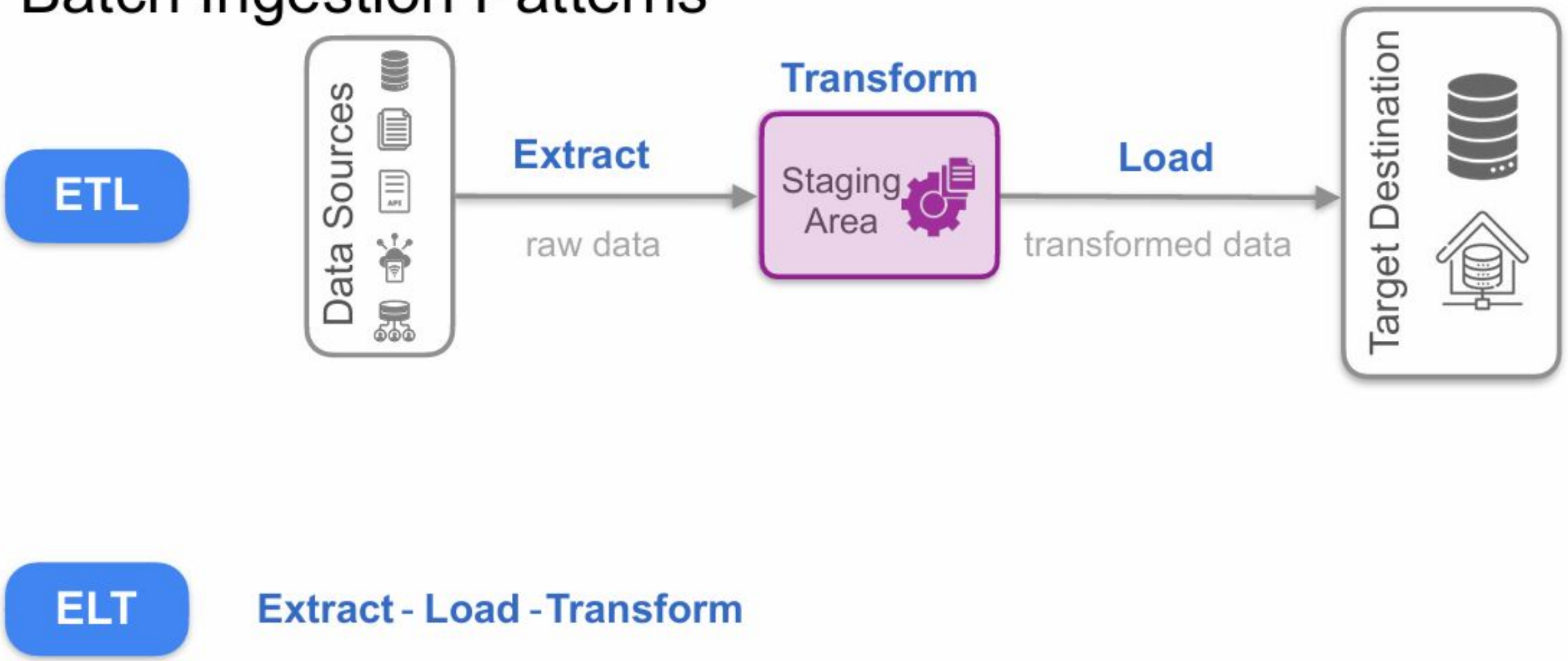
ETL — Extract - Transform - Load

ELT — Extract - Load - Transform

Batch Ingestion Patterns



Batch Ingestion Patterns



Emergence of Cloud Storage Systems

Early 2010s: Highly scalable cloud storage

Data Lake

built on top of object storage

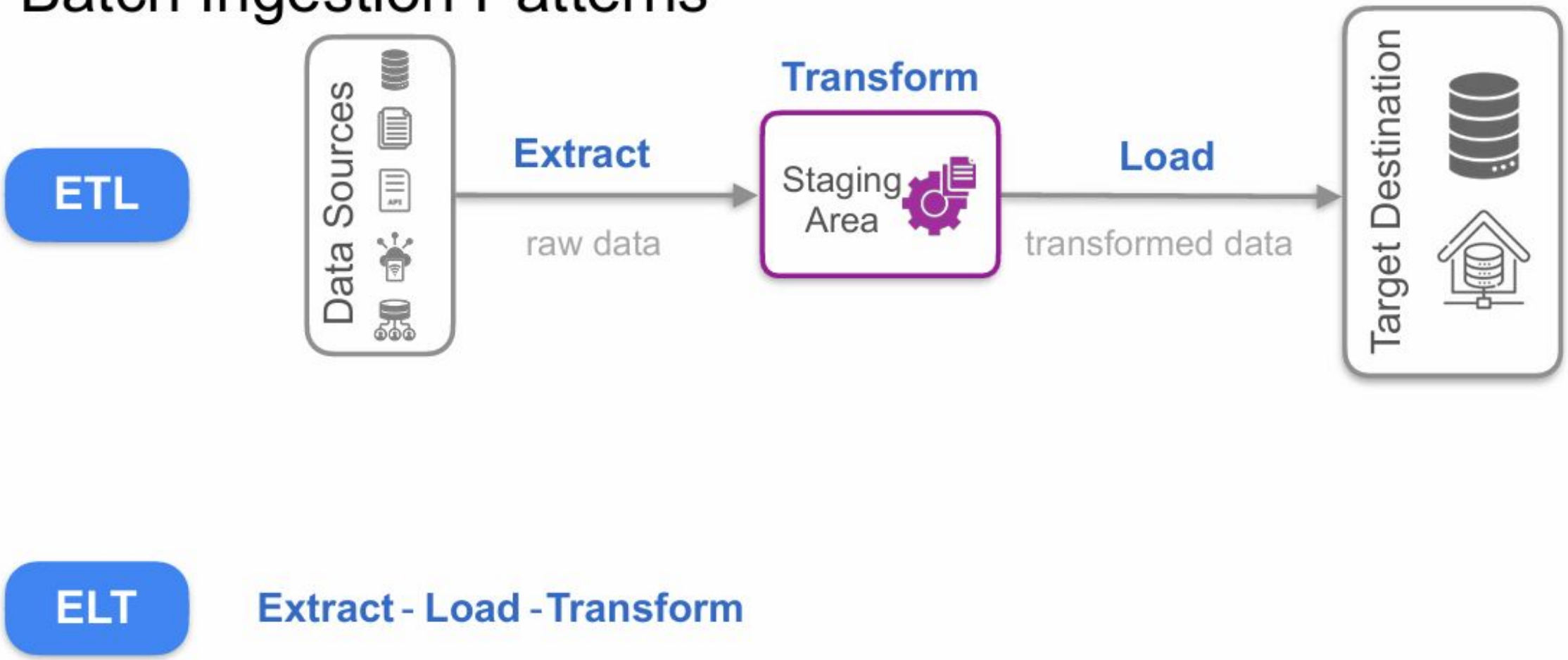
Cloud Data Warehouse

Amazon Redshift

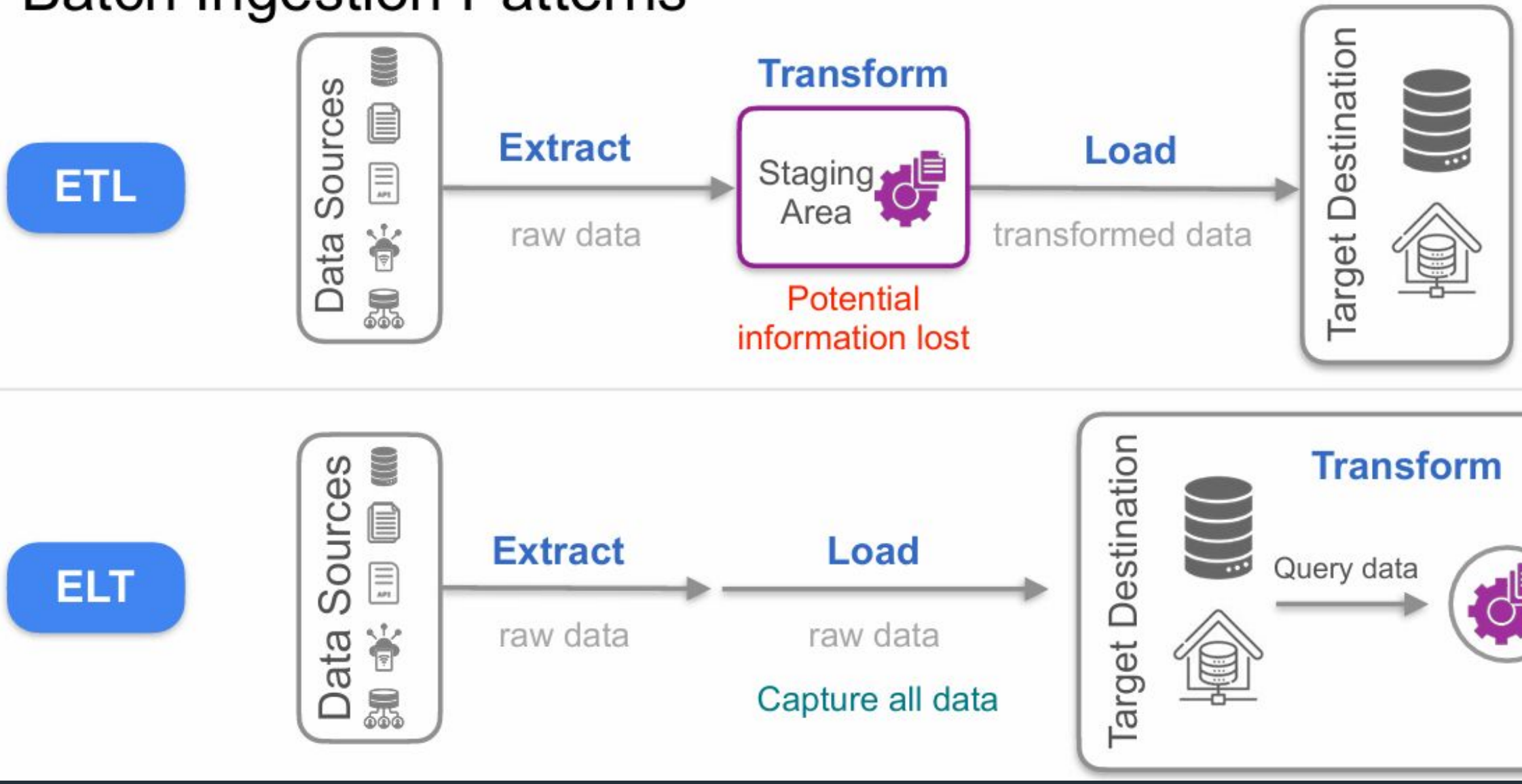
snowflake

- Store enormous amounts of data for relatively cheap
- Perform data transformations directly in the data warehouse

Batch Ingestion Patterns



Batch Ingestion Patterns



Advantages of Extract-Load-Transform

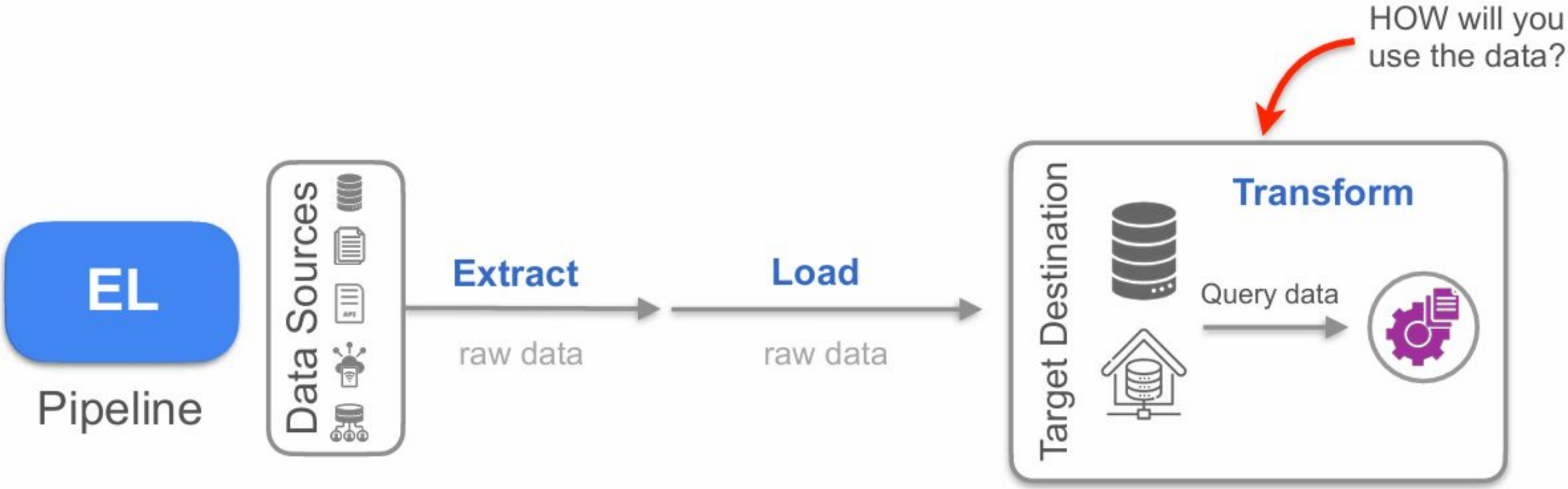
It is faster to implement.

It makes data available more quickly to end users.

Transformations can still be done efficiently.

You can decide later to adopt different transformations.

Downsides of Extract-Load-Transform



Data Swamp

Data has become unorganized, unmanageable, and essentially useless.

Downsides of Extract-Load-Transform

Data Swamp

Video by Adobe Stock (paid license)

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

REST API

What is an API?

API

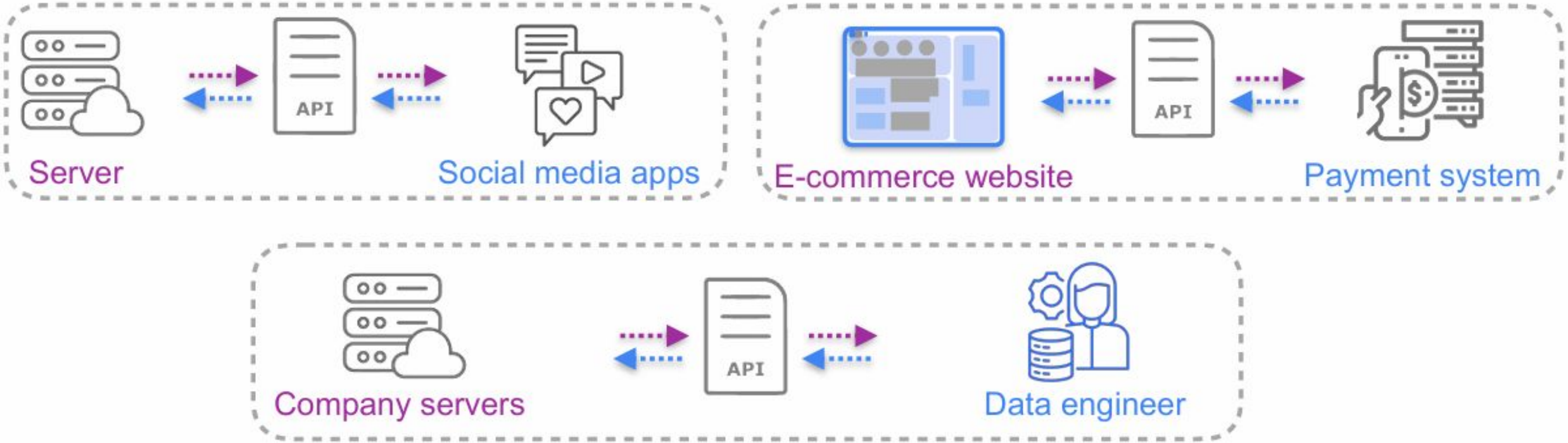
A set of rules and specifications that allows you to programmatically communicate and exchange data with an application.

Built into a wide range of software applications

What is an API?

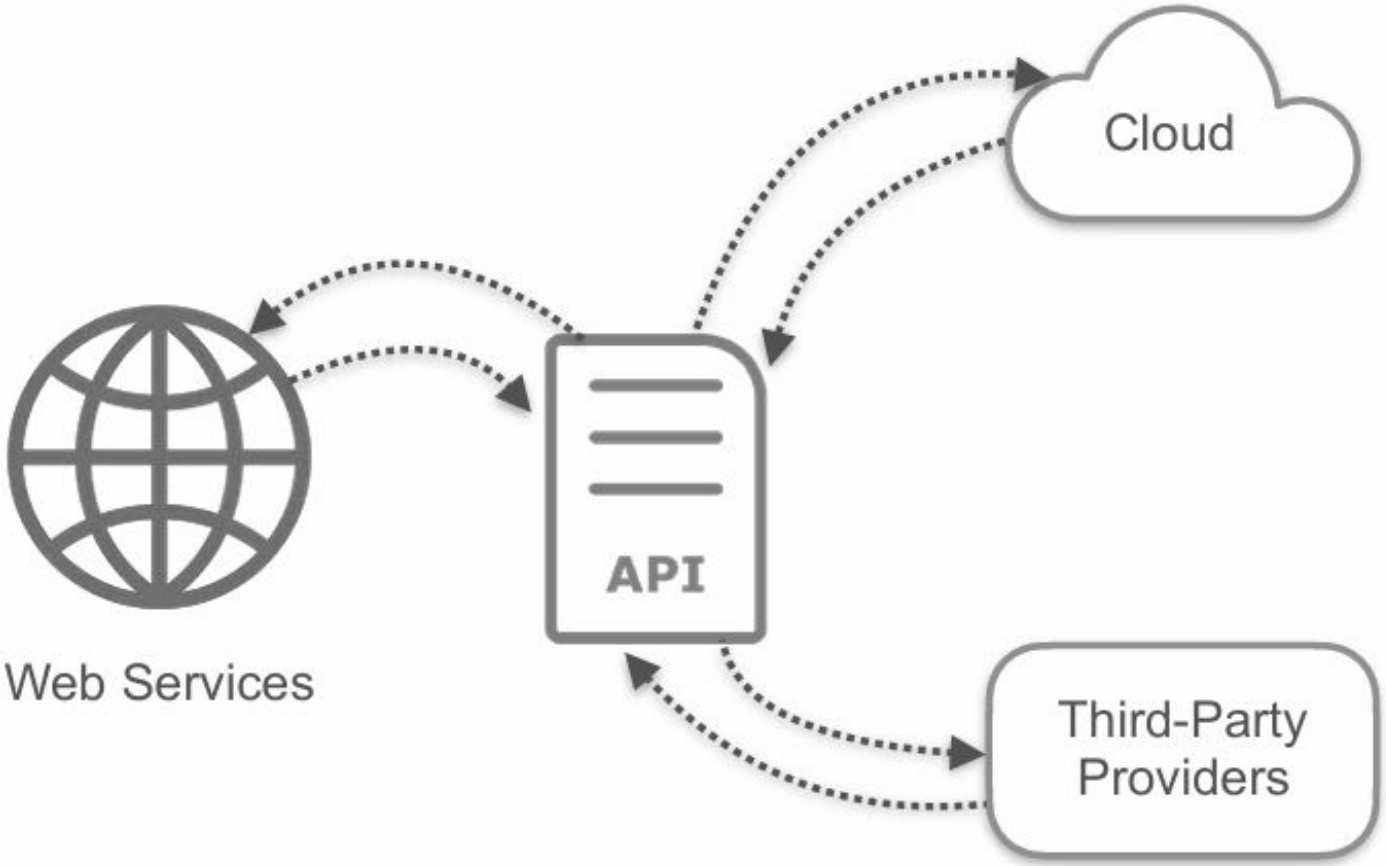
API

A set of rules and specifications that allows you to programmatically communicate and exchange data with an application.



What is an API?

Extract data from 3rd-party platforms (Spotify, Twitter, Google Analytics).Integrate with cloud services (AWS, GCP, Azure).



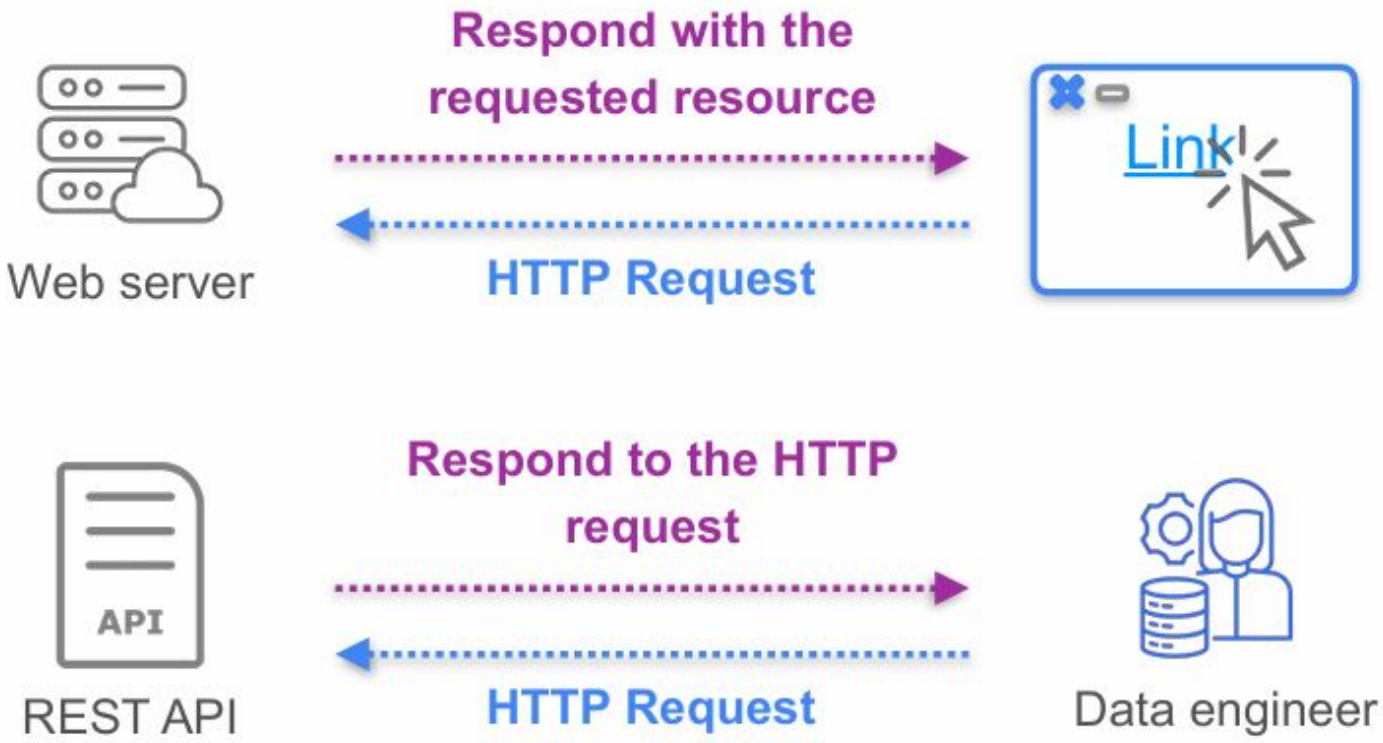
API Features

- Metadata
- Documentation
- Authentication
- Error handling

REST API

REST API

Representational State Transfer API
Use Hypertext Transfer Protocol (HTTP) as the basis for communication



RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Batch Processing to Get Data From an API

Batch Processing from an API

Upcoming Lab

- Extract data from the Spotify API
- Explore what pagination means
- Send an API request that requires authorization

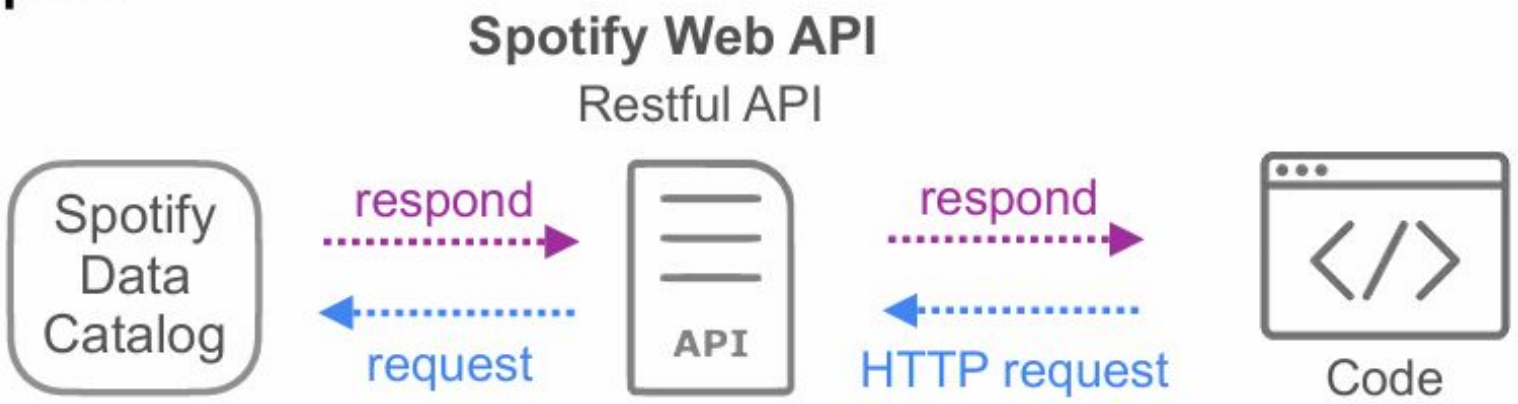
In this video,

- Go through some API concepts
- Give you an overview of the lab tasks

What you need

- Spotify account https://developer.spotify.com/
  When working with an API, it's very common that you'll have to sign up for an account
- Spotify Documentation https://developer.spotify.com/documentation/web-api

API Concepts



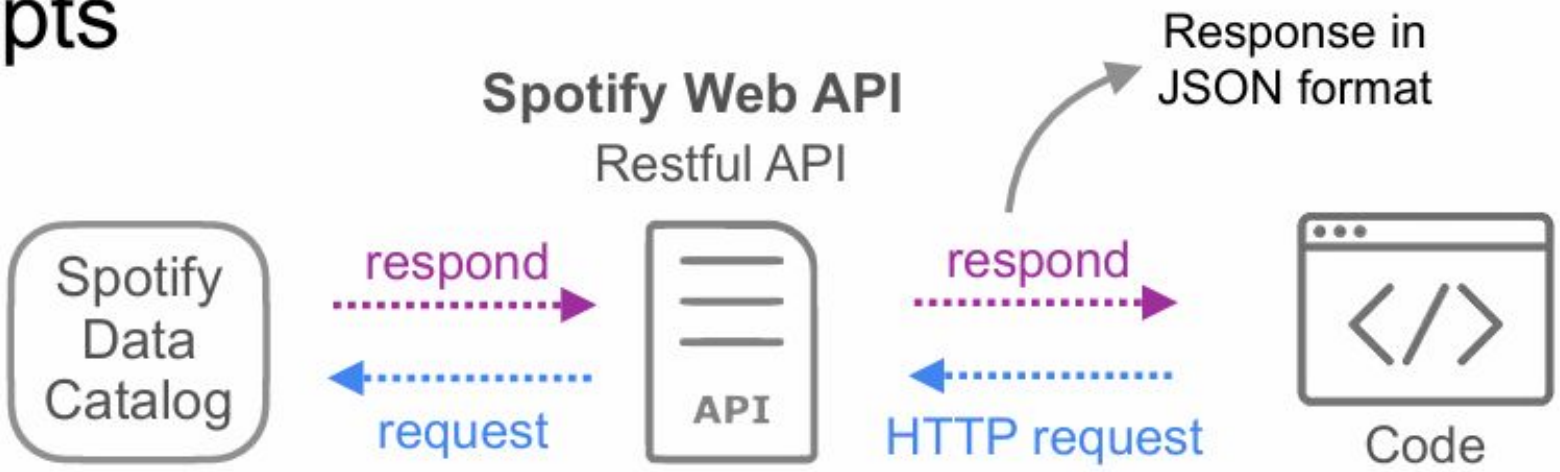
Each resource is represented by an **endpoint**.

Resource

- Music
- Artists
- Albums
- Tracks
- Playlist

HTTP Request	Action
GET	Retrieve a resource
POST	Create a resource
PUT	Change/Replace a resource
Delete	Delete a resource

API Concepts



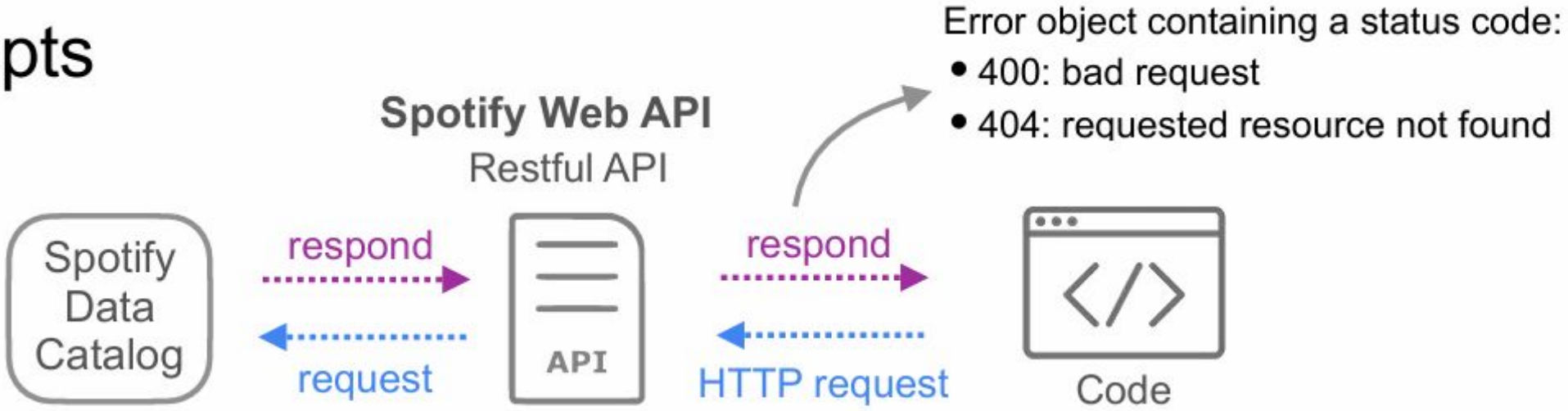
Each resource is represented by an **endpoint**.

Resource

- Music
- Artists
- Albums
- Tracks
- Playlist

HTTP Request	Action
GET	Retrieve a resource
POST	Create a resource
PUT	Change/Replace a resource
Delete	Delete a resource

API Concepts



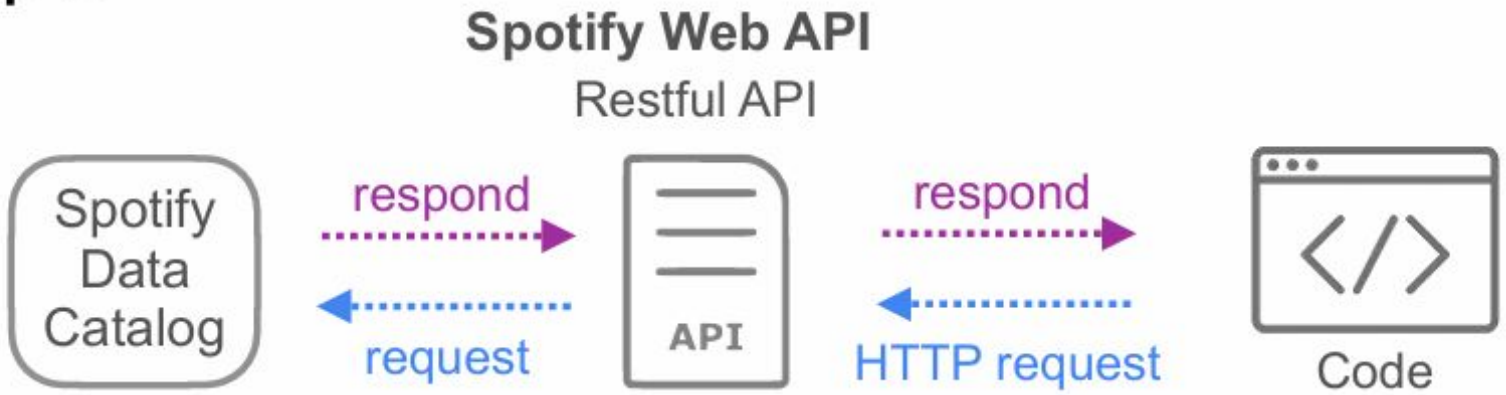
Each resource is represented by an **endpoint**.

Resource

- Music
- Artists
- Albums
- Tracks
- Playlist

HTTP Request	Action
GET	Retrieve a resource
POST	Create a resource
PUT	Change/Replace a resource
Delete	Delete a resource

API Concepts



: Endpoint + Access token

Each resource is represented by an **endpoint**.

Resource
- Music
- Artists
- Albums
- Tracks
- Playlist

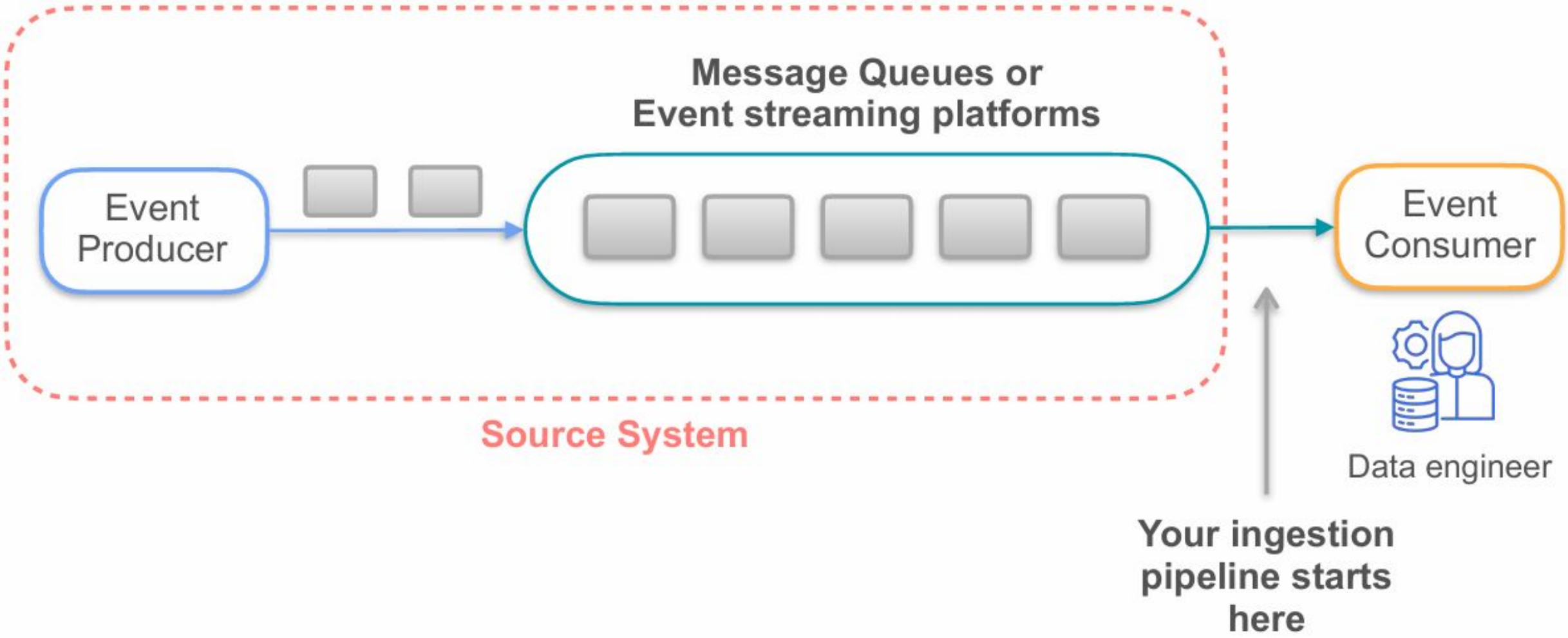
Access token: string that contains the permissions to acces a given resource. *(valid for 1 hour)*
- create a Spotify account
- get a client ID and a client secret and use them to generat the access token *(provided with the code)*

https://developer.spotify.com/documentation/web-api/concepts/authorization

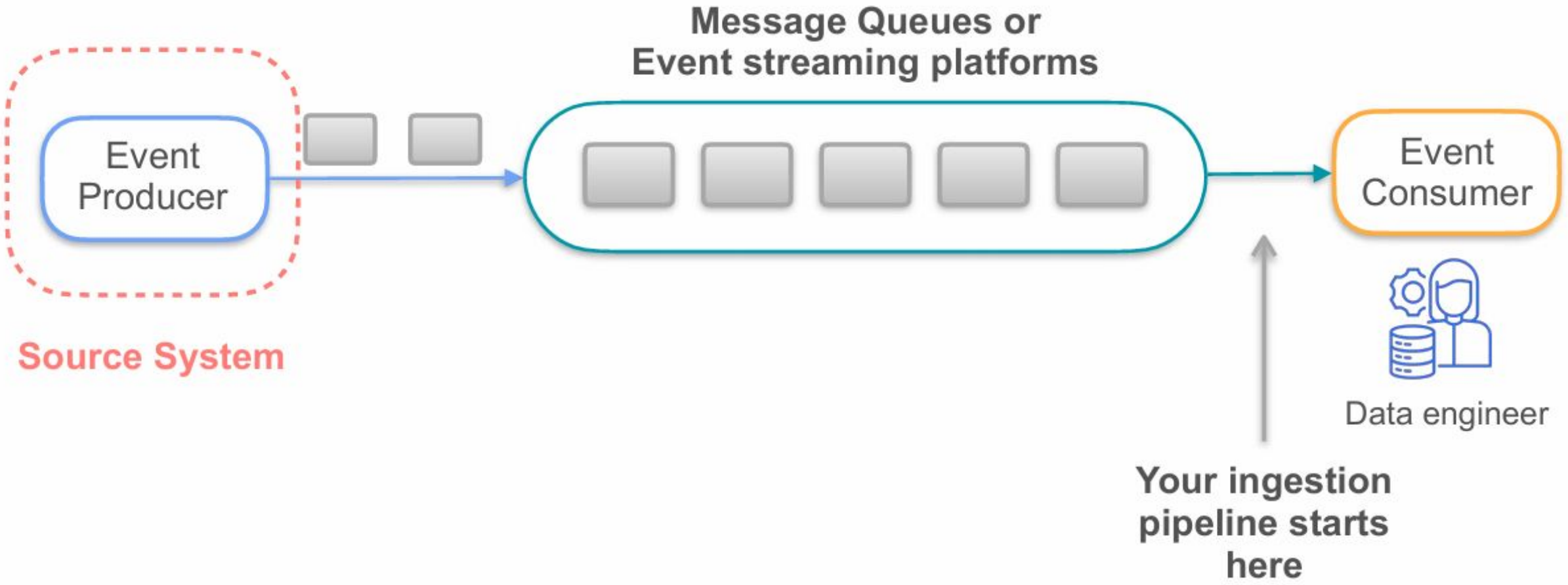
RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Streaming Ingestion Details

Streaming Systems



Streaming Systems



Message Queue

Event Streaming Platform

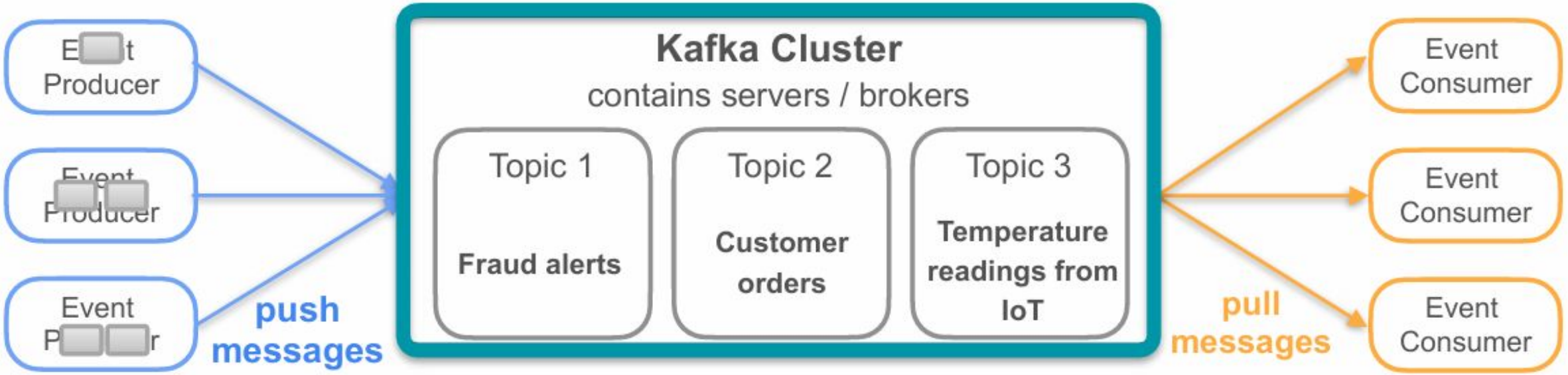
Message Queue A buffer used to deliver messages asynchronously

Event Streaming Platform

Message Queue A buffer used to deliver messages asynchronously

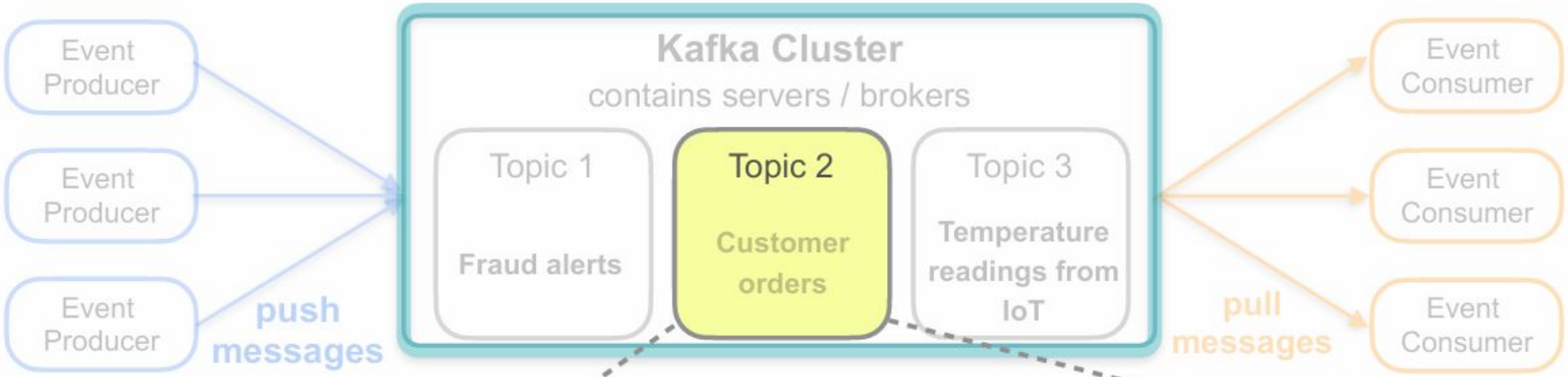
Event Streaming Platform Append-only persistent log

kafka Open-source event streaming platform



Topics:
Categories to hold related events

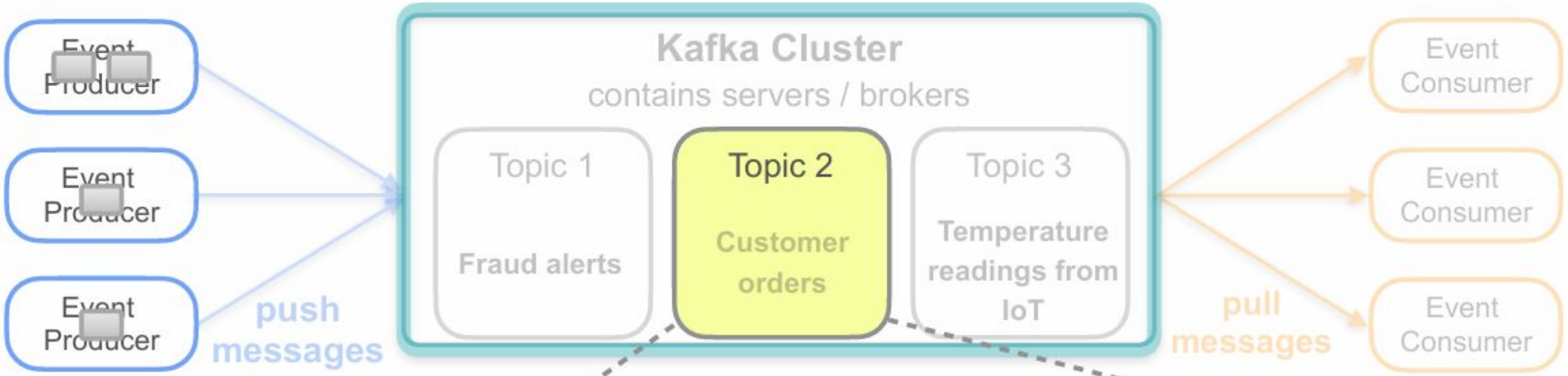
kafka Open-source event streaming platform



Topics:
Categories to hold related events

Partitions (logs):
Ordered immutable sequences of messages

kafka Open-source event streaming platform



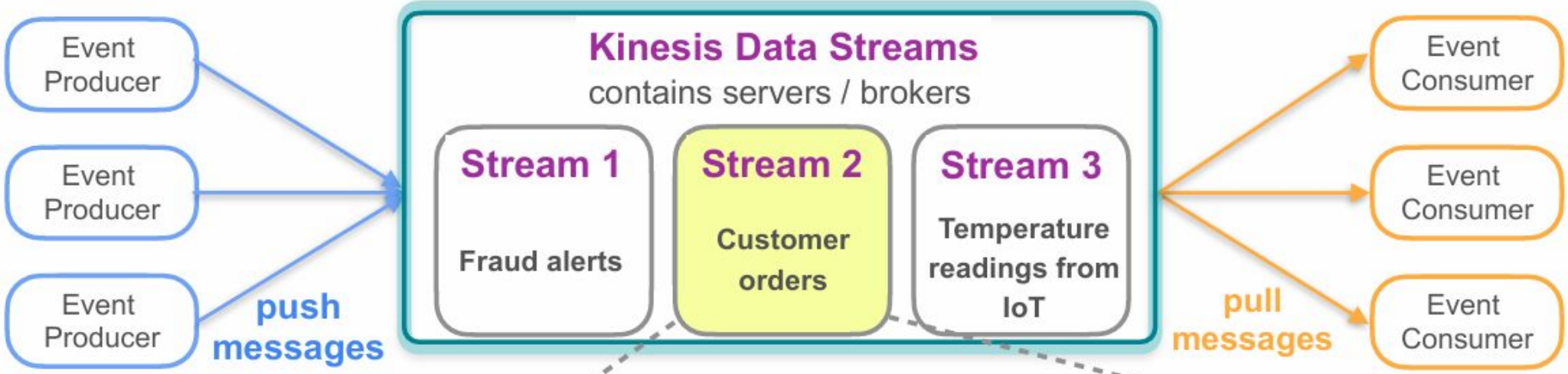
Topics:
Categories to hold related events

Partitions (logs):
Ordered immutable sequences of messages

Partition decision:

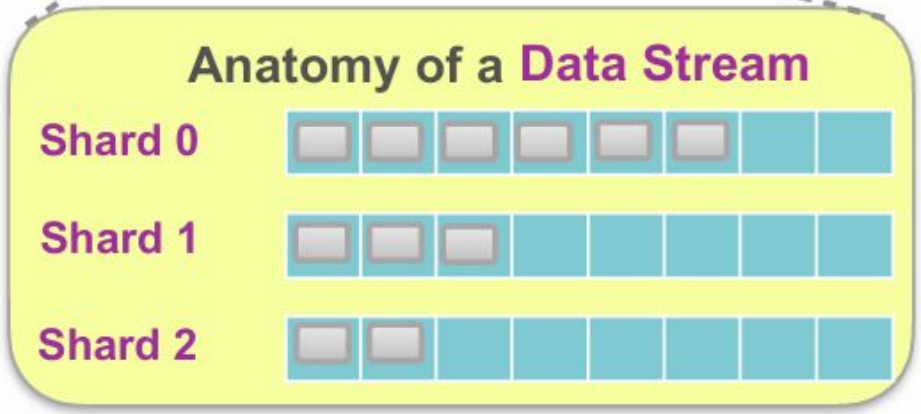
- Round-robin strategy
- Message key

Amazon Kinesis Data Stream

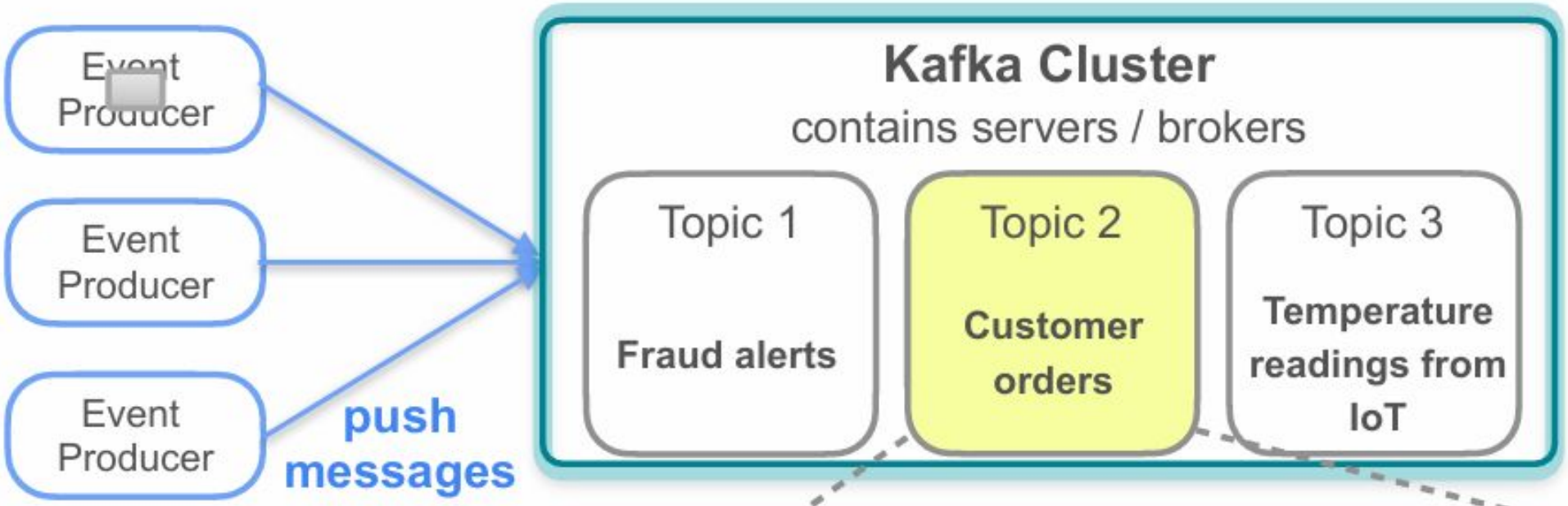


Stream:
Categories to hold related events

Shard (logs):
Ordered immutable sequences of messages



kafka Open-source event streaming platform



Topics:
Categories to hold related events

Partitions (logs):
Ordered immutable sequences of messages

A Consumer
subscribed to

Eve Consu
Eve Consu
Eve Consu

Kafka clu
retain
messa

Conversation with the Software Engineer

Software Engineers

Data Engineer

User Id: 7945
IP address: 127.168.10.32
Action: User added a product x to their cart
Status: Success
Time Stamp: 01-01-2025:10.30

Web-Server Log

Event Producer

Amazon Kinesis Data Streams

kafka

Ingestion pipeline starts here

Conversation with the Software Engineer

Software Engineers

Data Enginee

Ingestion pipeline starts here

User Id: 7945
IP address: 127.168.10.32
Action: User added a product x to their cart
Status: Success
Time Stamp: 01-01-2025:10.30

Web-Server Log

Event Producer

Ingest

Amazon Kinesis Data Streams

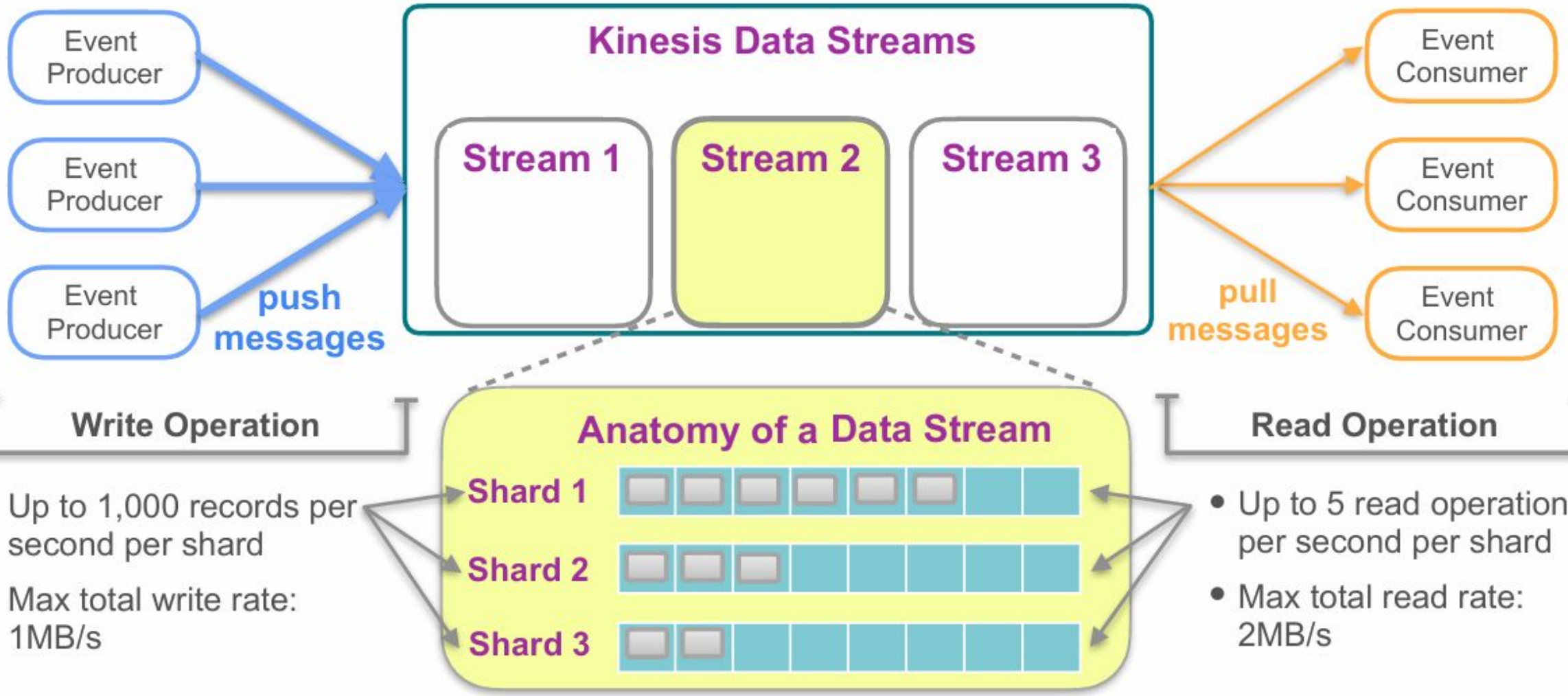
kafka

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

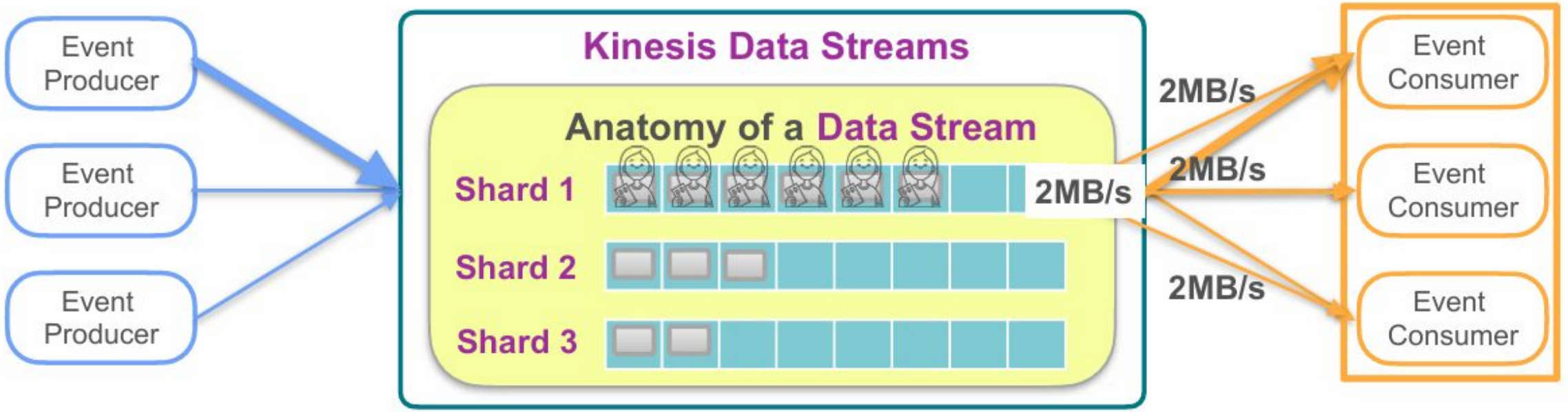
Amazon Kinesis Data Streams Details

Kinesis in “on-demand” Mode	Kinesis in “Provisioned” Mode

• Automatically manage the scaling of the shards up or down as needed
• Only charged for what you use
• More convenient from an operational perspective
	• Specify the number of shards necessary for your application based on the expected write and read request rate
	• Manually add more shards or re-shard when needed
	• A good fit if…
	  ◦ you have predictable application traffic
	  ◦ you are able to control your costs more carefully



Size and rate of write and read operations



Data Record

Partition Key — customerID *12567910*

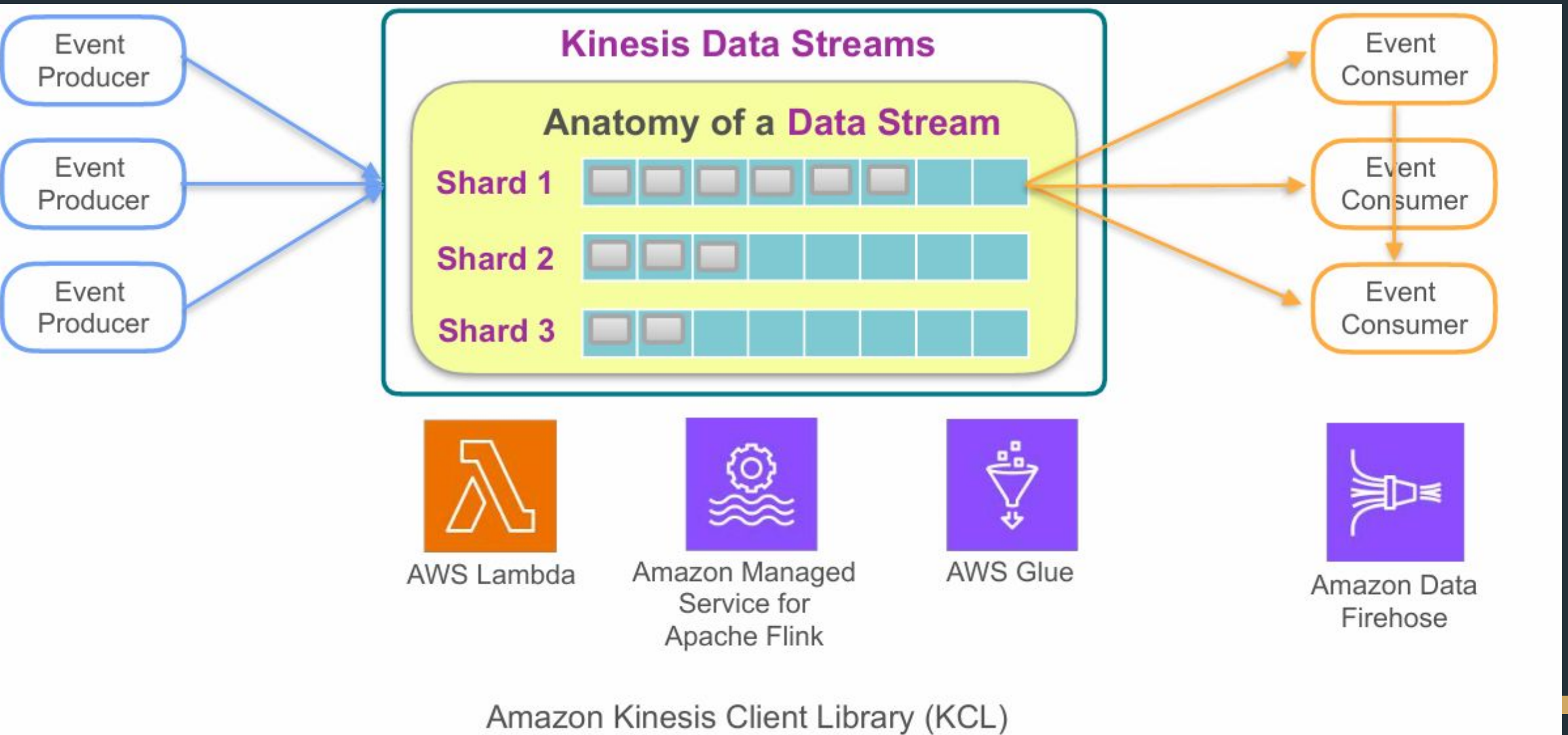
Sequence Number

Binary Large Object (BLOB)

Used to determine which shard the data record is placed into

Shared Fan-Out — When consumers share a shard's read capacity

Enhanced Fan-Out — When consumers are able to read at the full read capacity of the shard



Amazon Kinesis Client Library (KCL)

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

DataOps

DataOps

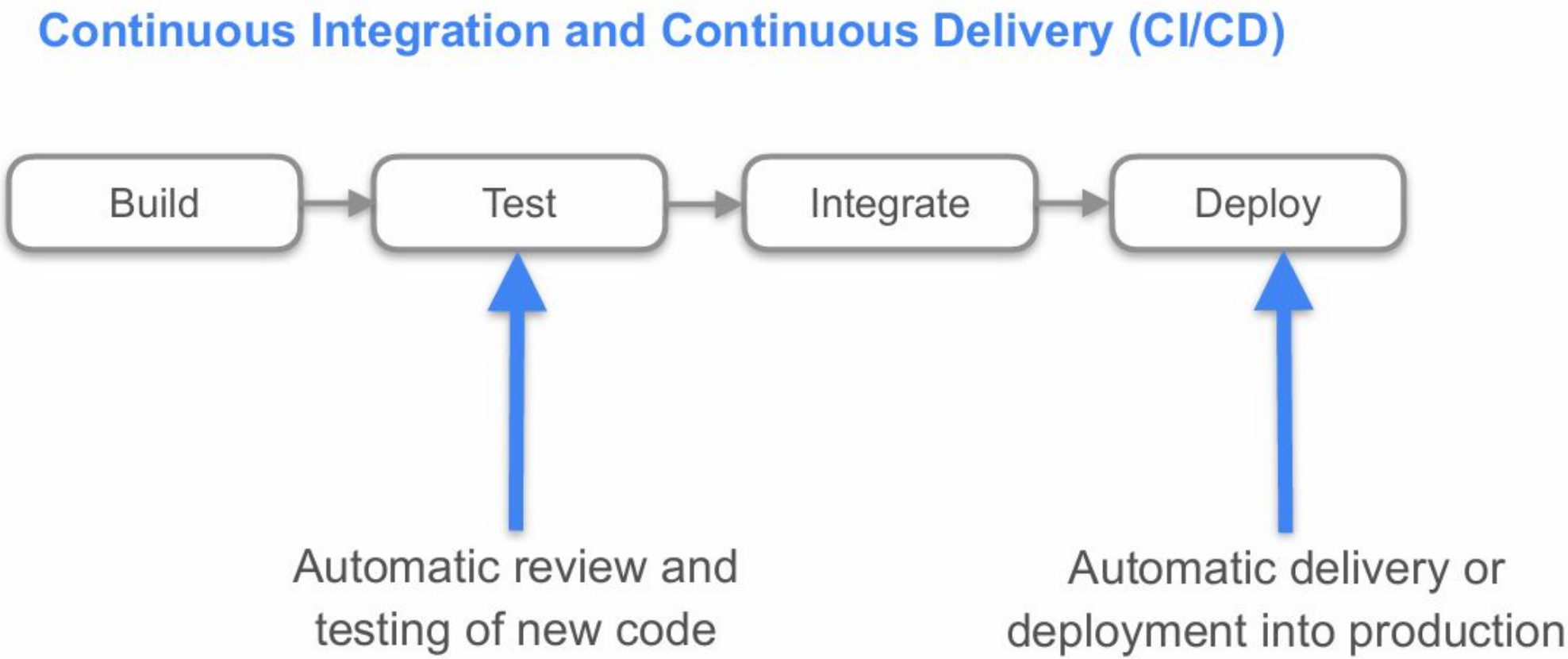
DataOps

Set of practices and cultural habits centered around building robust data systems and delivering high quality data products.

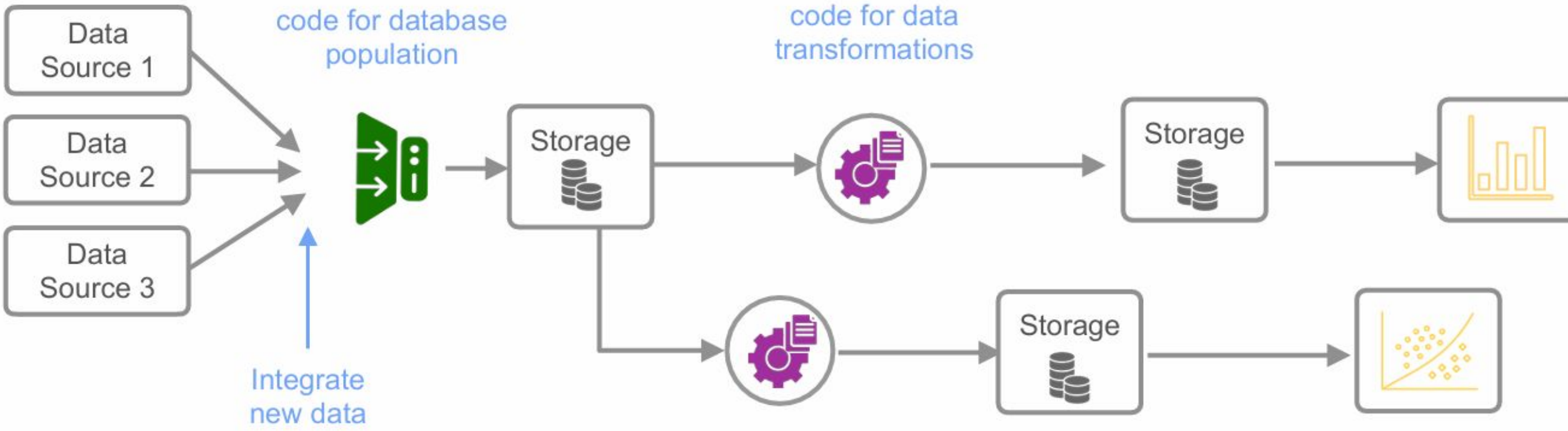
DevOps

Set of practices and cultural habits that allow software engineers to efficiently deliver and maintain high quality software products.

DevOps Automation



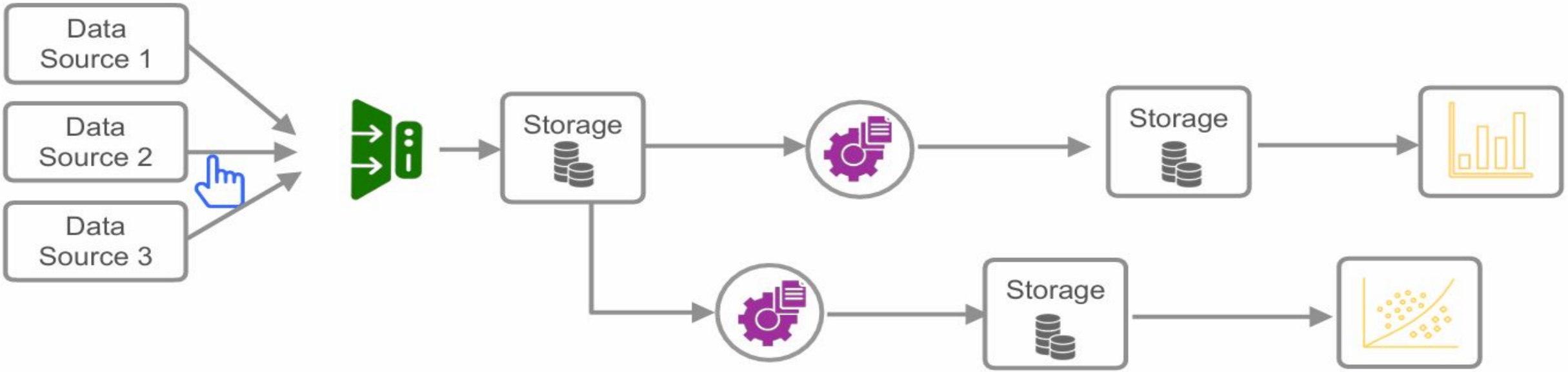
CI/CD can be applied directly to code and data within your data pipeline



DataOps Automation

Automation of data pipeline running

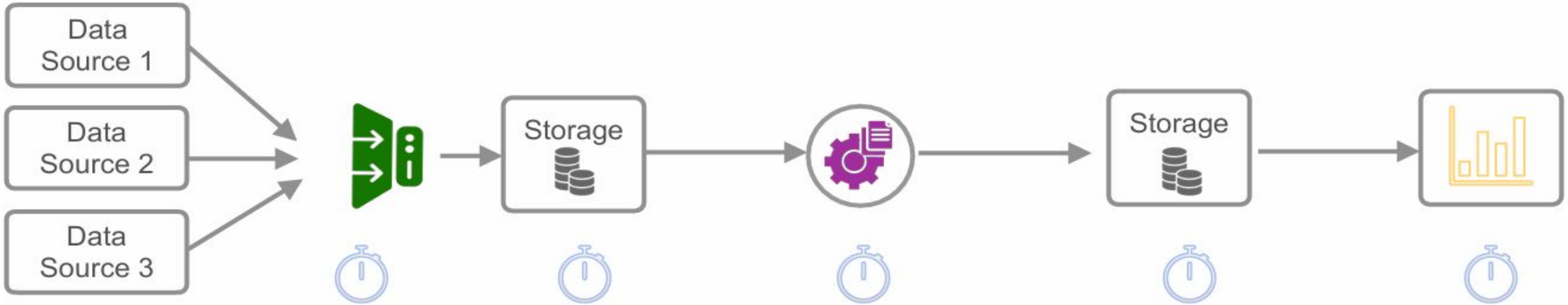
No automation: run all processes manually



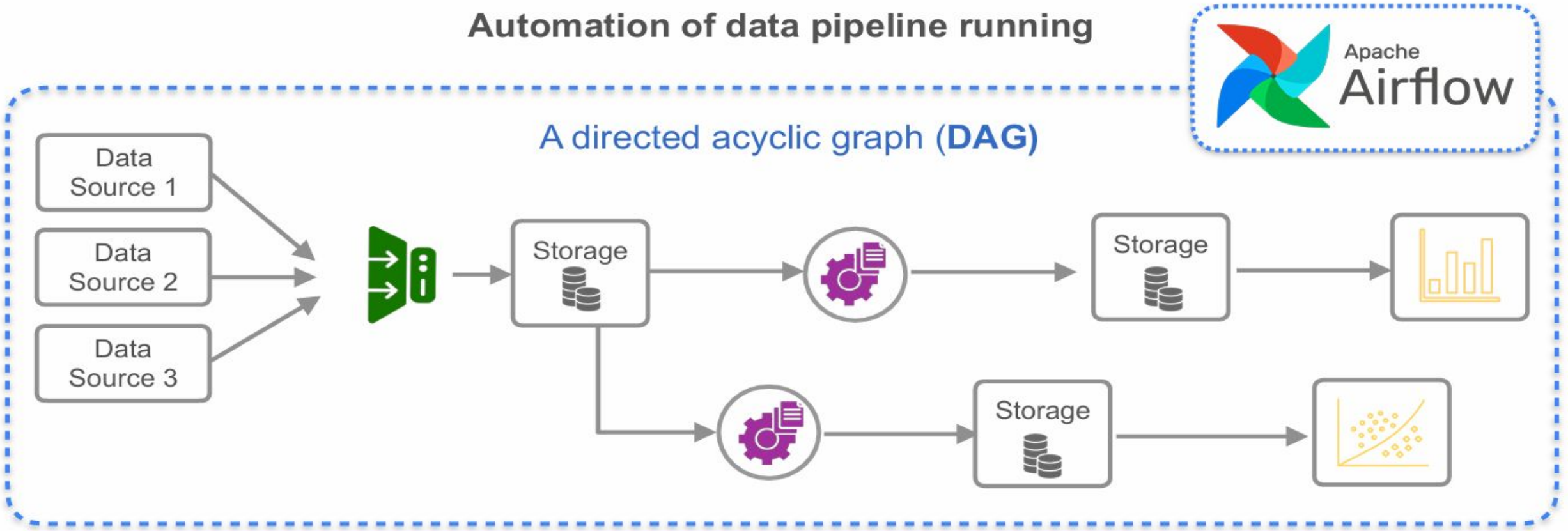
DataOps Automation

Automation of data pipeline running

Pure scheduling: run stages of your pipeline according to a schedule



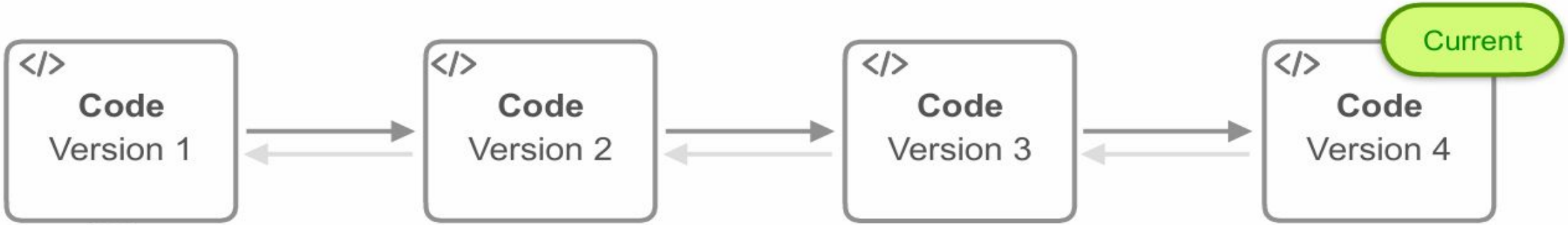
DataOps Automation



DataOps Automation

Key Underpinning CI/CD : Version control

Track changes in the **code**



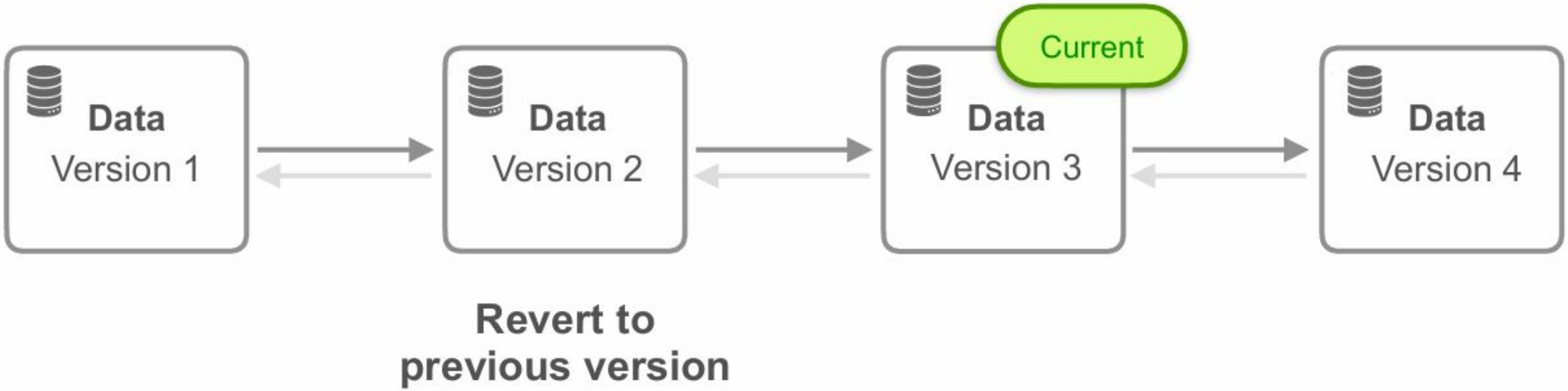
GitHub

Revert to a previous version

DataOps Automation

Key Underpinning CI/CD : Version control

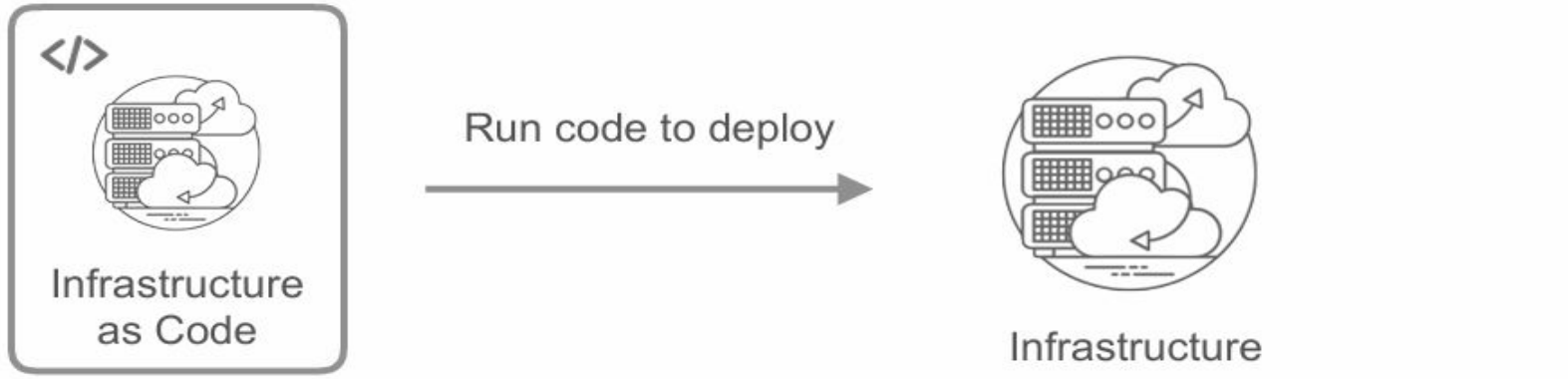
Track changes in the **code** and the **data**



DataOps Automation

Key Automation Aspect: Infrastructure as Code

Maintain the design of your infrastructure as a codebase

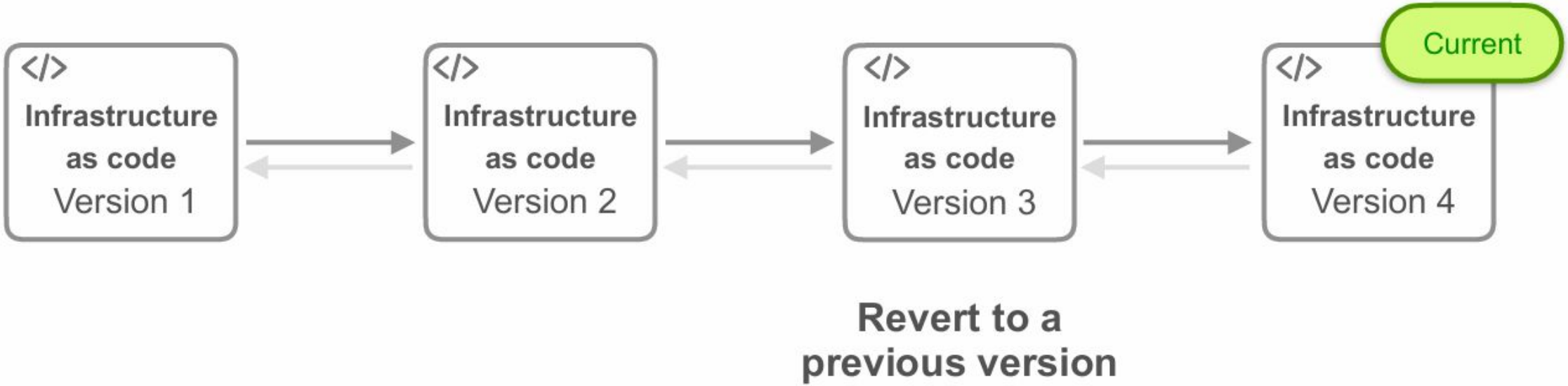


Modify this code to redefine your infrastructure

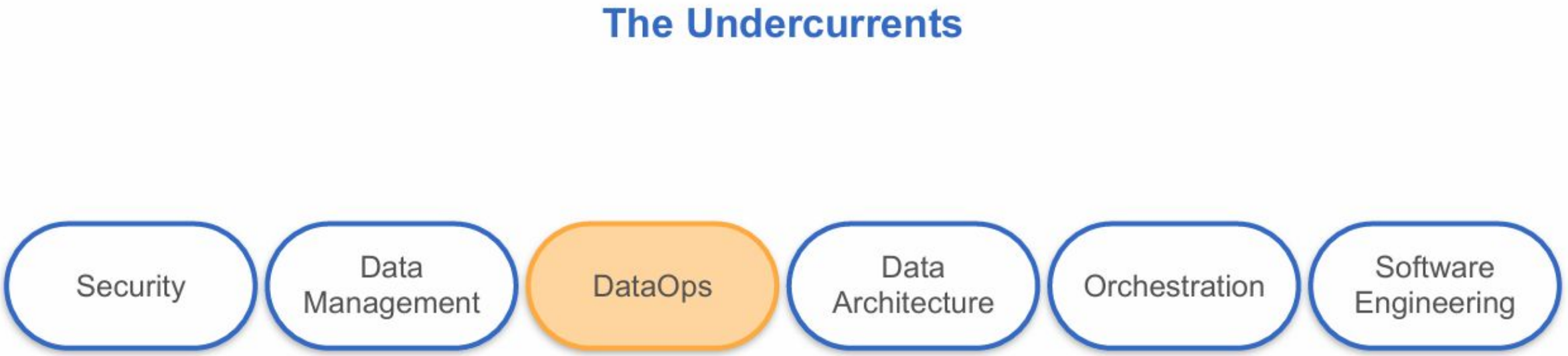
DataOps Automation

Version control over your entire infrastructure

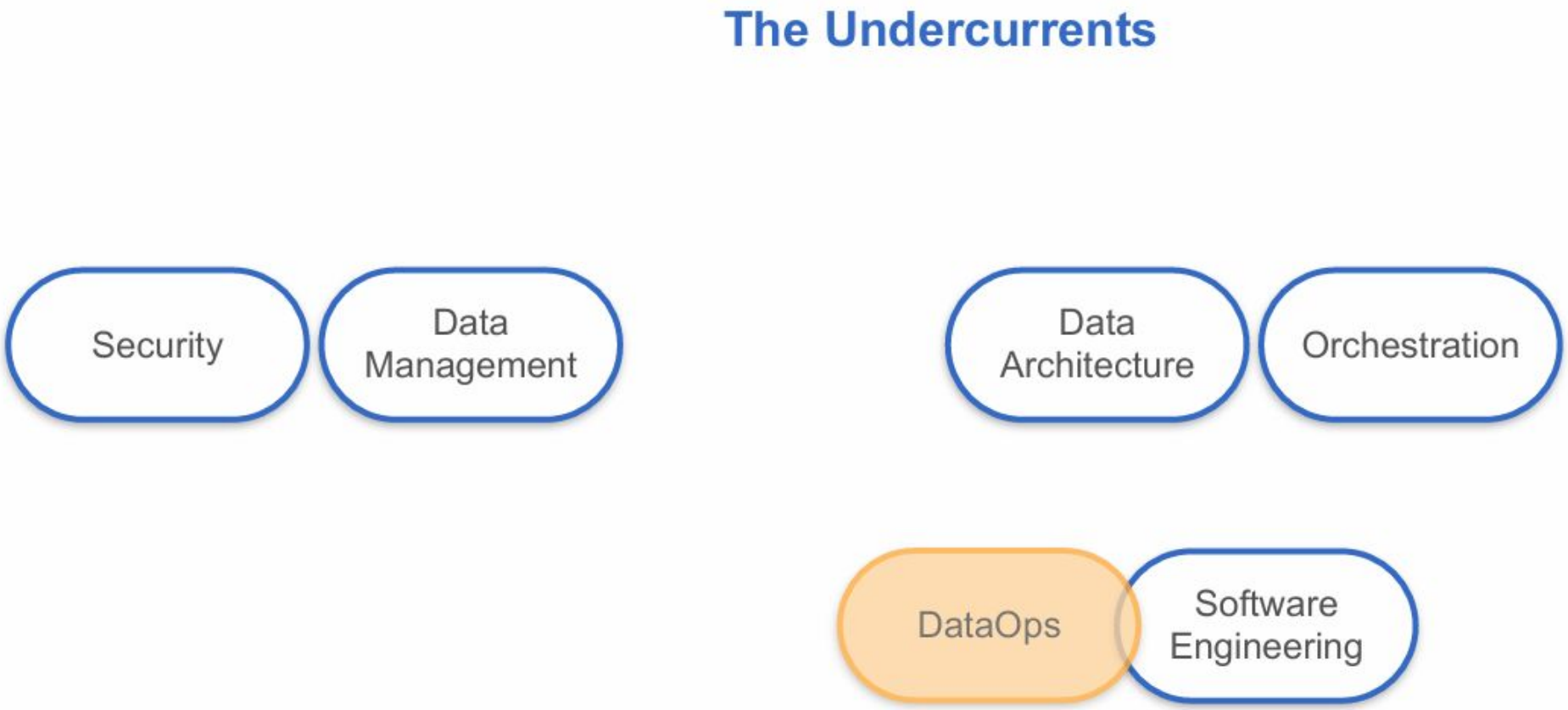
Track changes in the **code** defining your infrastructure



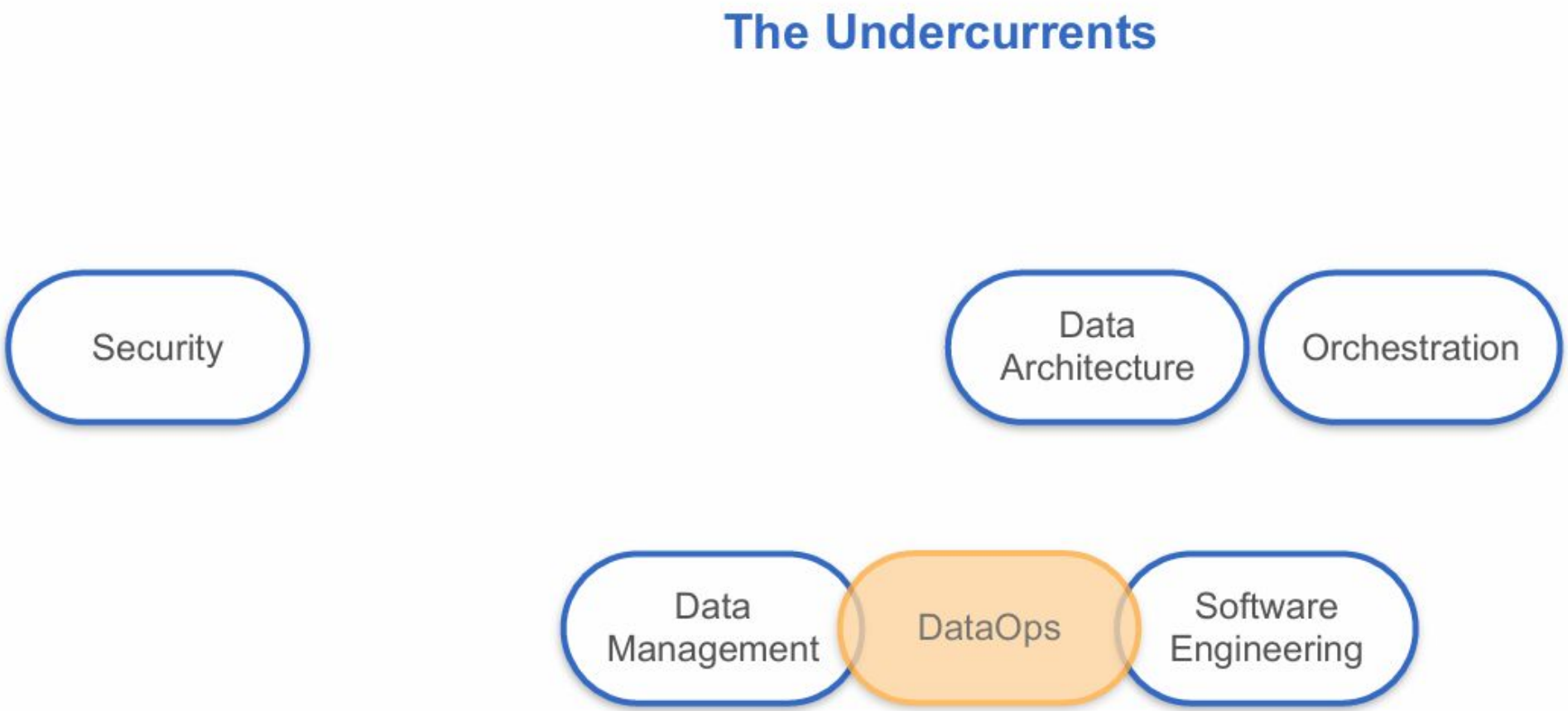
DataOps


The Undercurrents
Security
Data Management
DataOps
Data Architecture
Orchestration
Software Engineering

DataOps


The Undercurrents
Security
Data Management
Data Architecture
Orchestration
DataOps
Software Engineering

DataOps


The Undercurrents
Security
Data Architecture
Orchestration
Data Management
DataOps
Software Engineering

Infrastructure as Code

1970s

Configuration Management

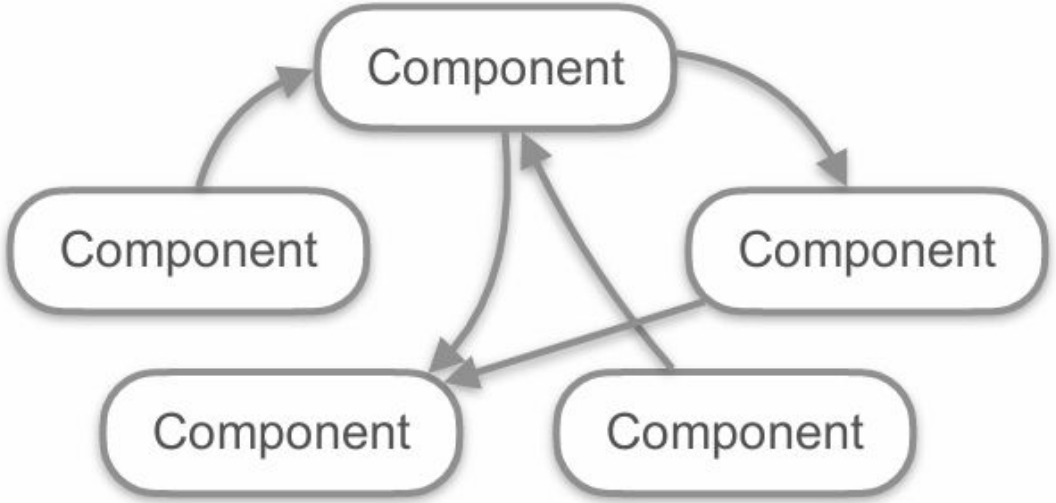
Challenge : configuration of a series of physical machines

To automate some configuration tasks

Infrastructure as Code

Configuration Management

Amazon Elastic Compute Cloud (Amazon EC2)



Infrastructure as Code

1970s

Configuration Management

2006

Amazon Elastic Compute Cloud (Amazon EC2)

2010s

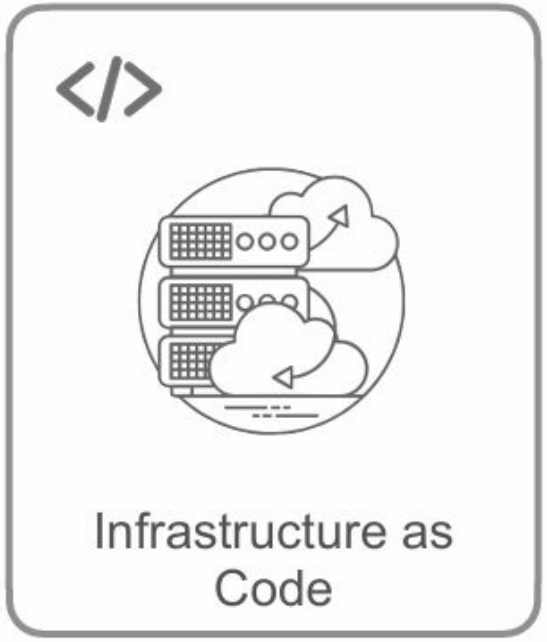
Development of Infrastructure as Code Tools

HashiCorp Terraform

AWS CloudFormation

ANSIBLE

Infrastructure as Code



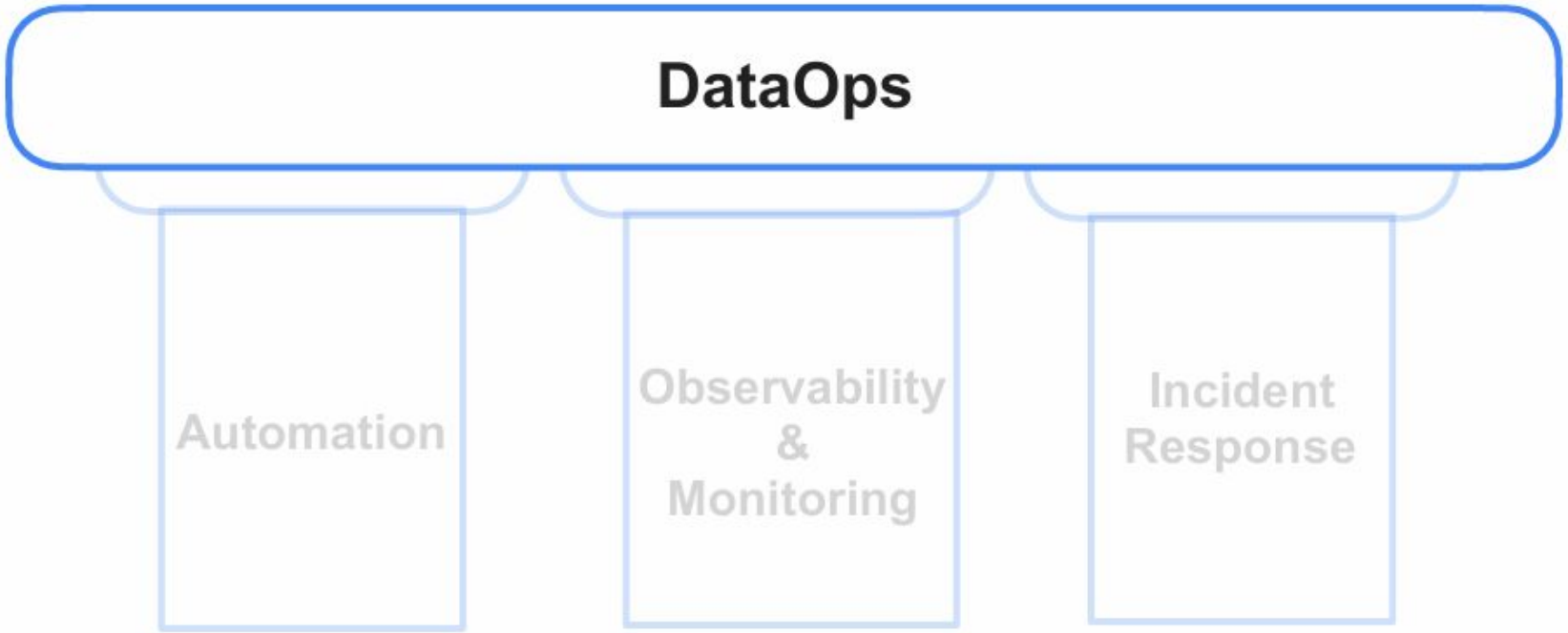
AWS CloudFormation

HashiCorp Terraform

ANSIBLE

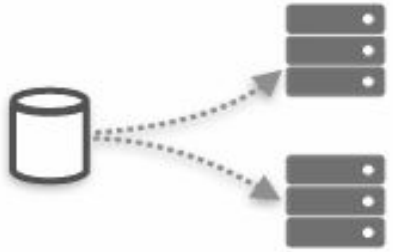
- Manage infrastructure resources on the cloud using code
- No manual clicking or writing bash scripts

Observability and Monitoring



DevOps Observability

Advent of the Cloud

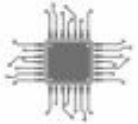


Move toward Distributed Systems

Observability tools

Gain visibility into systems' health

Metrics



CPU and RAM usage

Response time

- Quickly detect anomalies
- Identify problems
- Prevent downtime
- Ensure reliable software products

Data Observability

Monitor the health of data systems

Observability tools

Monitor the health and quality of data

Health/quality of your data

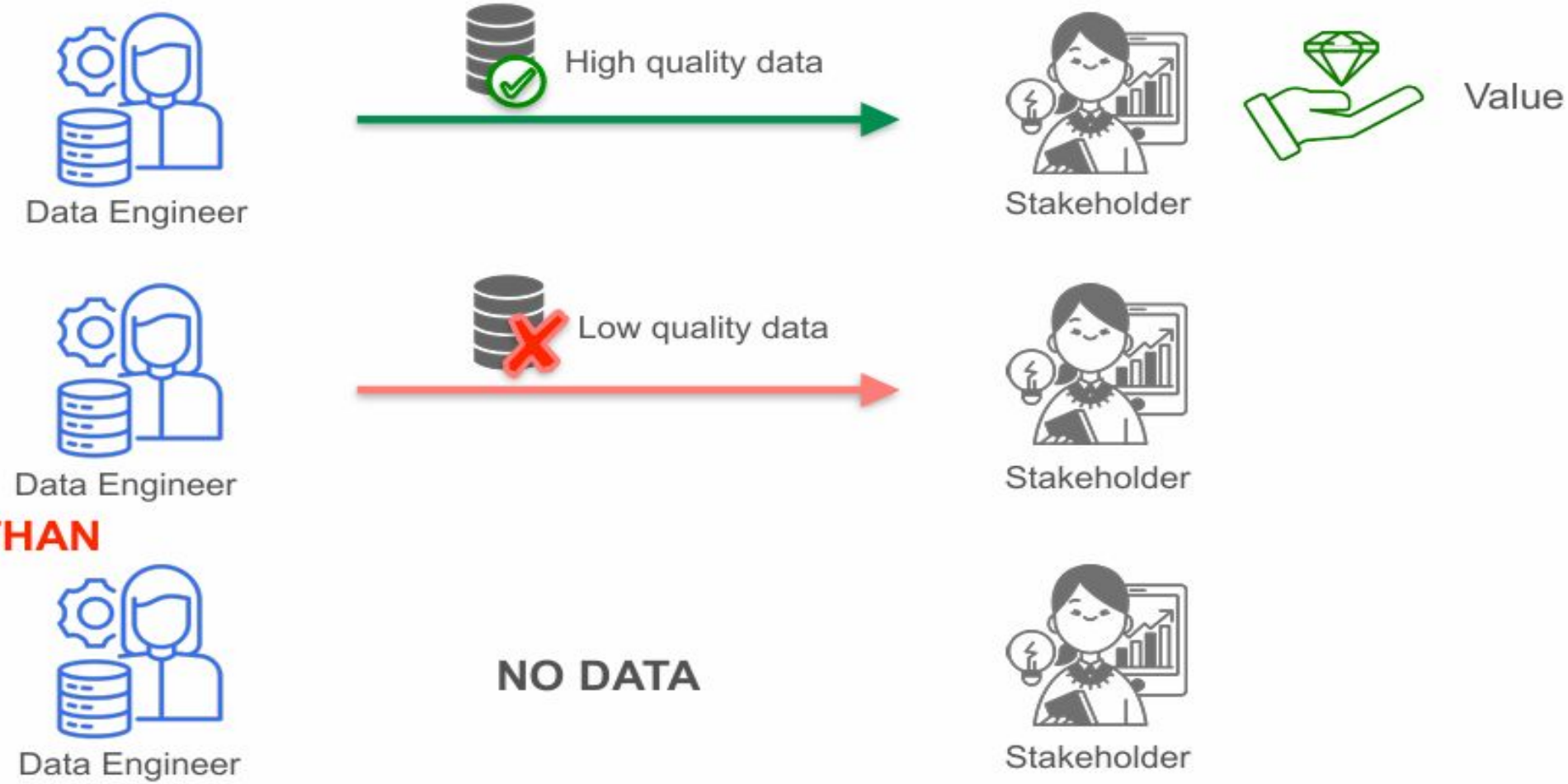
What is High Quality Data?

High Quality Data	Low Quality Data
Accurate	Inaccurate
Complete	Incomplete
Discoverable	Hard to find
Available in a timely manner	Late

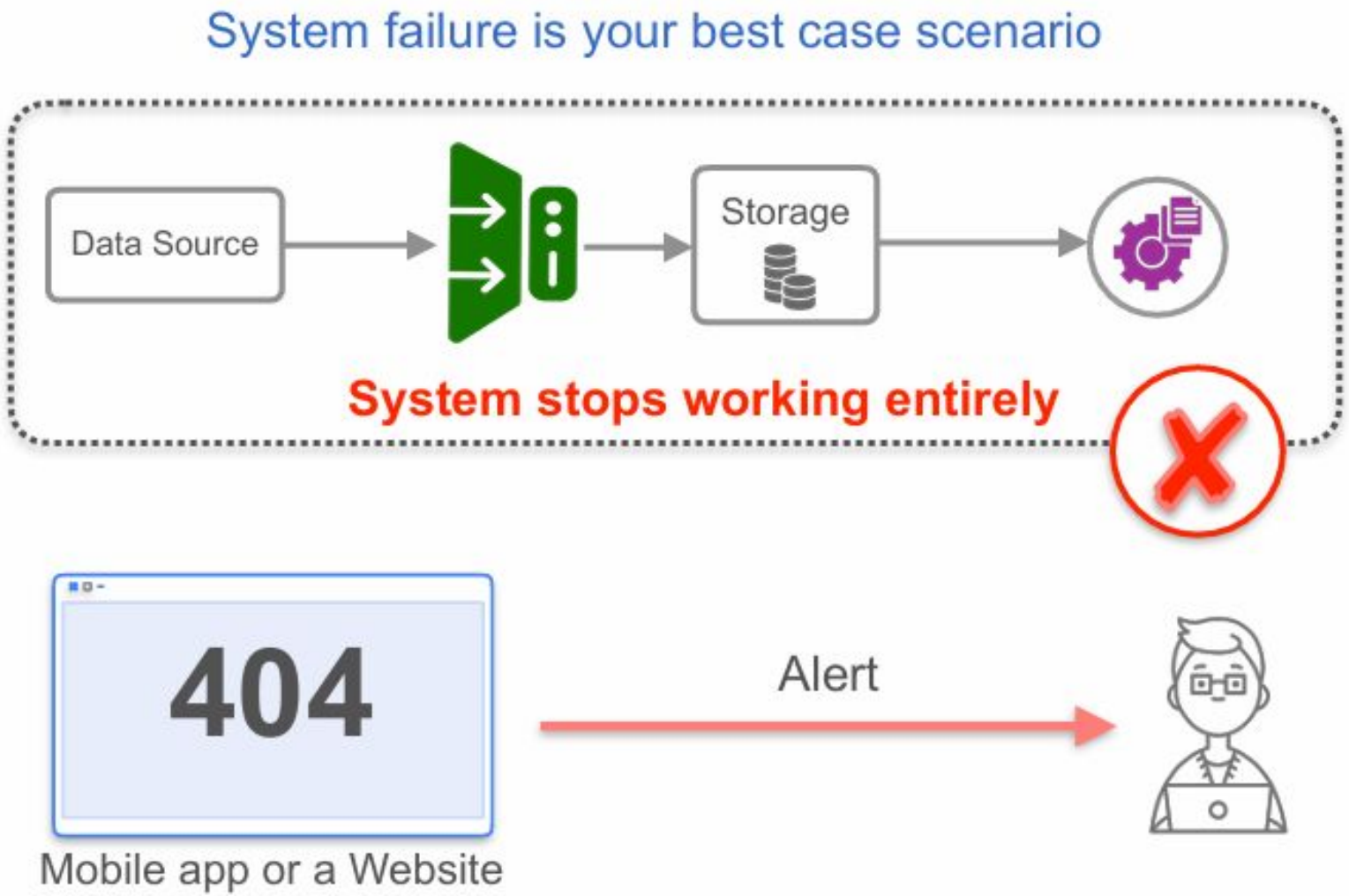
Represents exactly what stakeholders expect:

- well-defined schema
- data definitions

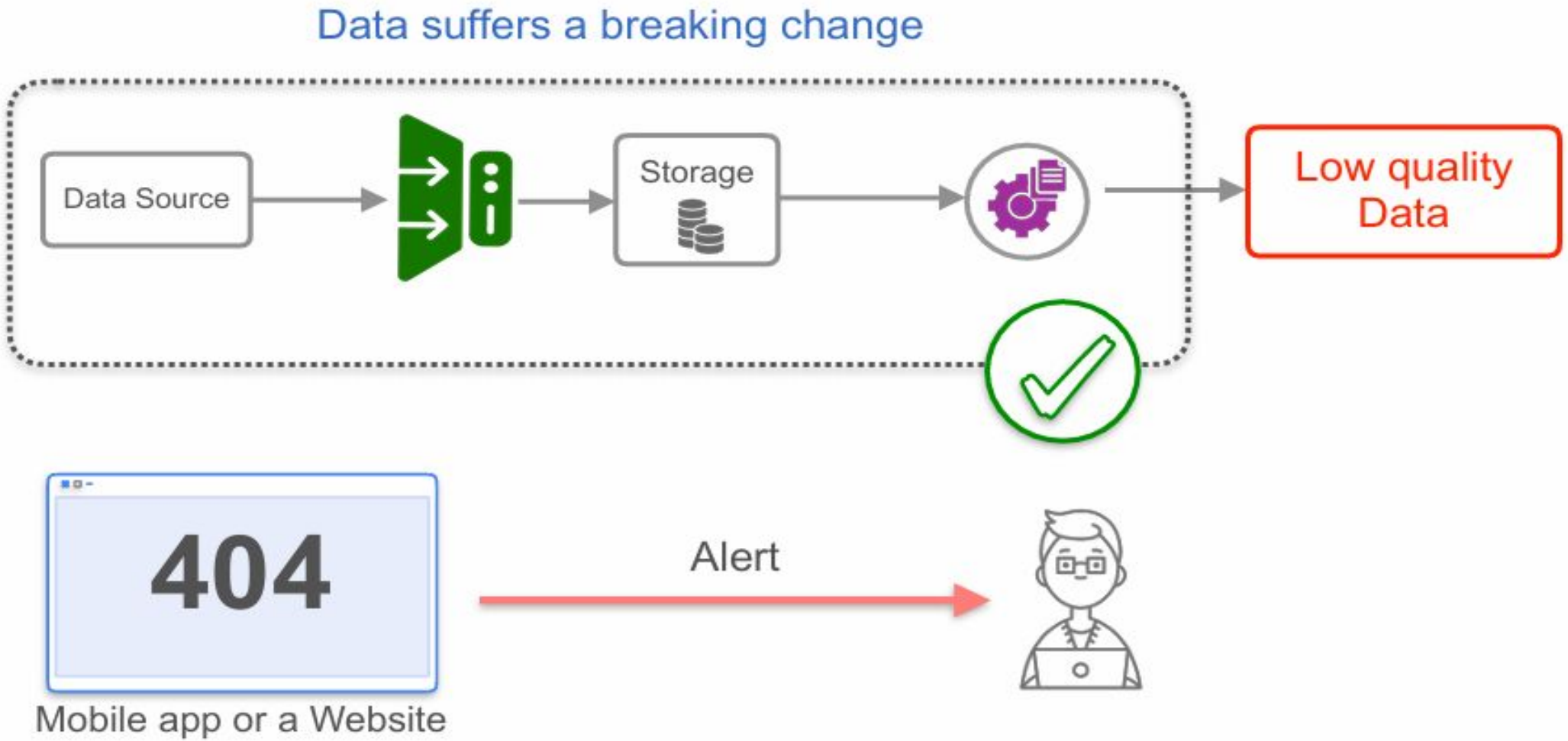
Quality of Data



Data Observability



Data Observability



Scenario

U.S. based company

Sell products in Europe

sale id	sale price (USD)
1	10
2	20
3	50

Total: 80

sale id	sale price (EUR)
1	9.15
2	18.31
3	45.77

Total:73.23

It looks like there was a sudden Drop in Revenue

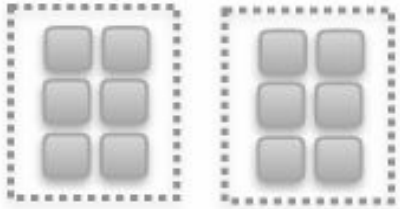
RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

DataOps - Observability & Monitoring

Monitoring Data Quality

Data Quality Metrics

- **Volume**: Total number of records ingested in each batch or over some time interval



- **Distribution**: Range of values in a particular column stays within some thresholds

col 1	col 2	col 3
1	10	100
2	20	300
3	50	600

- **Null values**: Total number of null values in a table

col 1	col 2	col 3
1	10	100
2	null	null
null	50	600

- **Freshness**: The *difference* in time between *now* and the *timestamp of the most recent* record in your data

- Identify and focus on the most important metrics
- Avoid creating confusion and "alert fatigue"

Data Quality Metrics

Stakeholders

What do stakeholders care about most for this use case?

Stakeholder Requirement	What to monitor?
Current data: no more than 24 hours old	"Freshness" of the data: measure when the latest records were ingested

Data Quality Metrics

Stakeholders interested in product sales revenue

Sales

sale id	product id	purchase amount
1	2	10
2	3	20
3	4	null

Product

product id	product name	SKU
1	product 1	yk3
2	product 2	yk3
3	product 3	sj6

What to monitor?

- Verify that all sales records were successfully ingested
- No null values in sales record

What may be less important?

- Product SKU numbers match product descriptions
- Each customer's postal code was recorded correctly

Testing Ingested Data

Source Owners

Data Engineer

- Build in checks or tests to verify that the schema and data types stay consistent.
- Identify problems as early as possible before they propagate further down your pipelines.

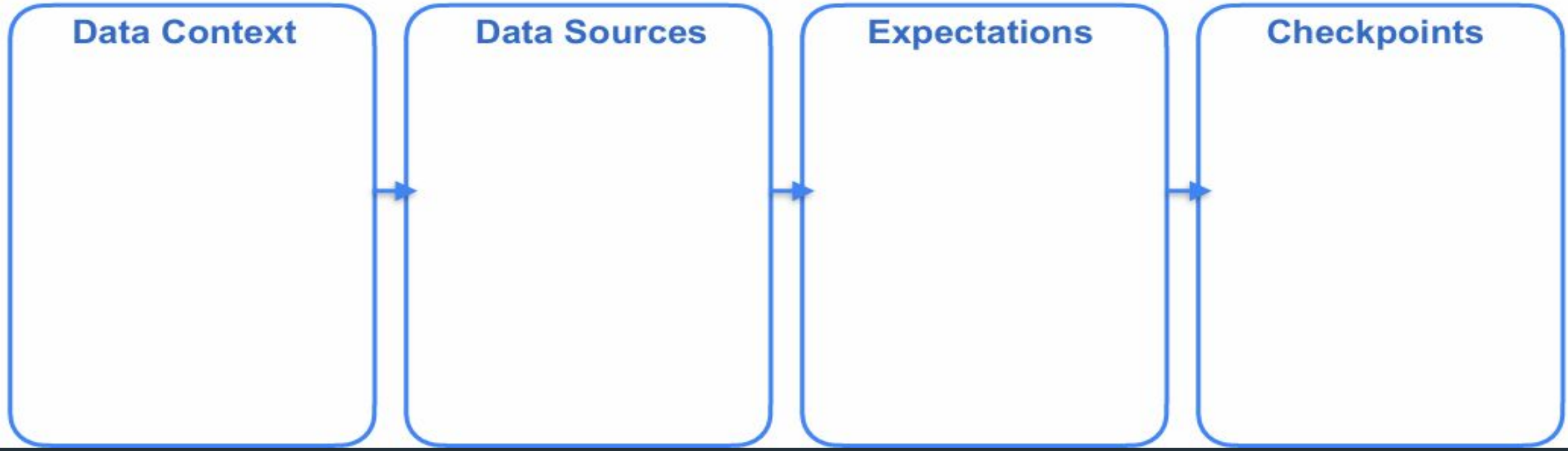
RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

DataOps - Observability & Monitoring

Great Expectations - Core Components

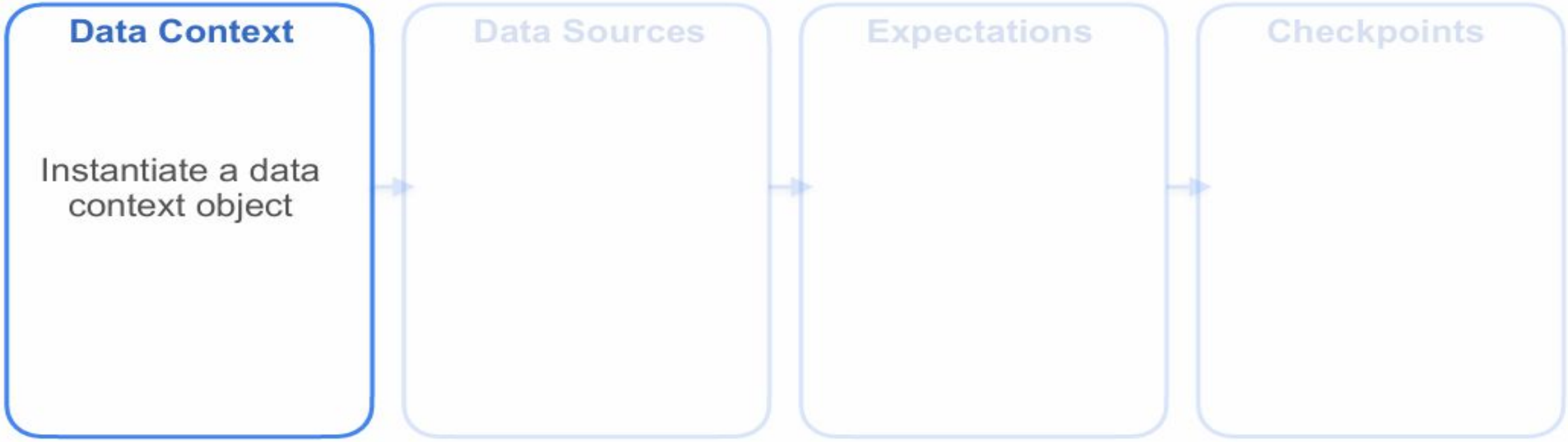
great expectations

1. Specify the data
2. Define your expectations
3. Validate your data against the expectations



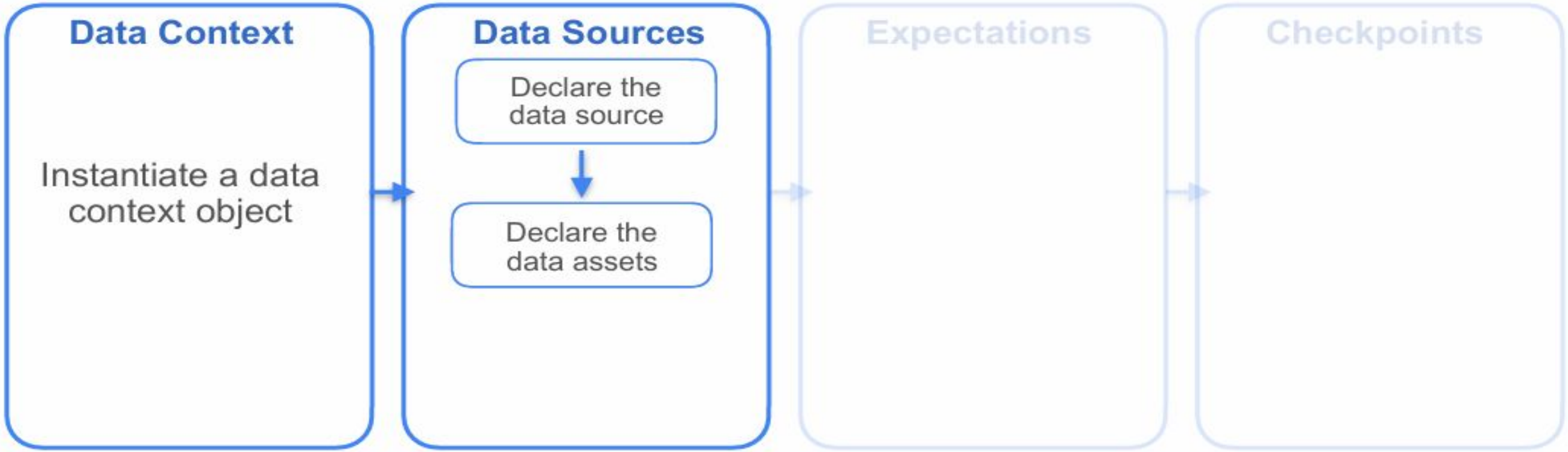
Great Expectations Workflow

- **Data Context**: entry point for the Great Expectations API
- **Great Expectations API**: classes and methods that allow you to connect data sources, create expectations and validate your data



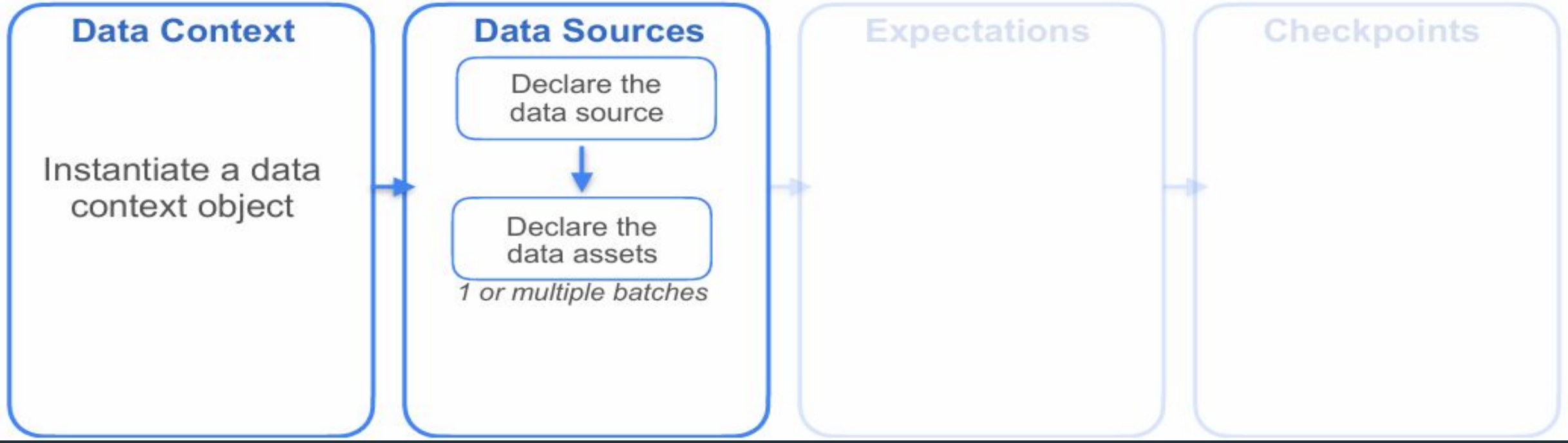
Great Expectations Workflow

- **Data source:** SQL Database, a local file system, an S3 bucket, a Pandas DataFrame
- **Data asset**: collection of records within a data source
  - Table in a SQL database
  - File in a file system
  - Join query asset
  - Collections of files matching a pattern



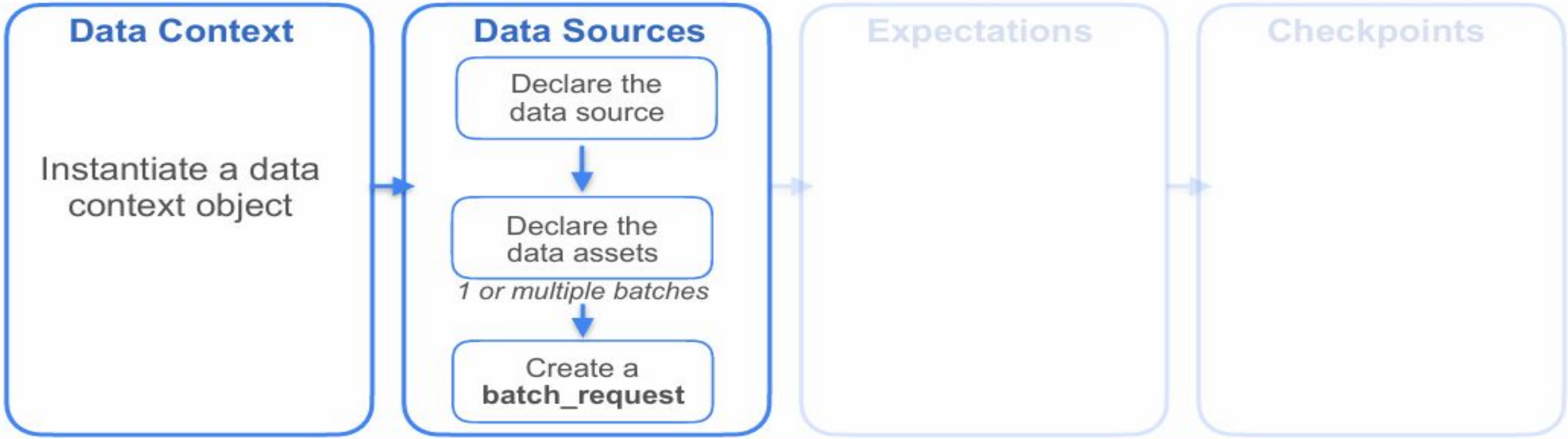
Great Expectations Workflow

- **Data source:** SQL Database, a local file system, an S3 bucket, a Pandas DataFrame
- **Data asset**: collection of records within a data source
- **Batches**: partitions from your data asset
  - partitions by date
  - partitions by column values



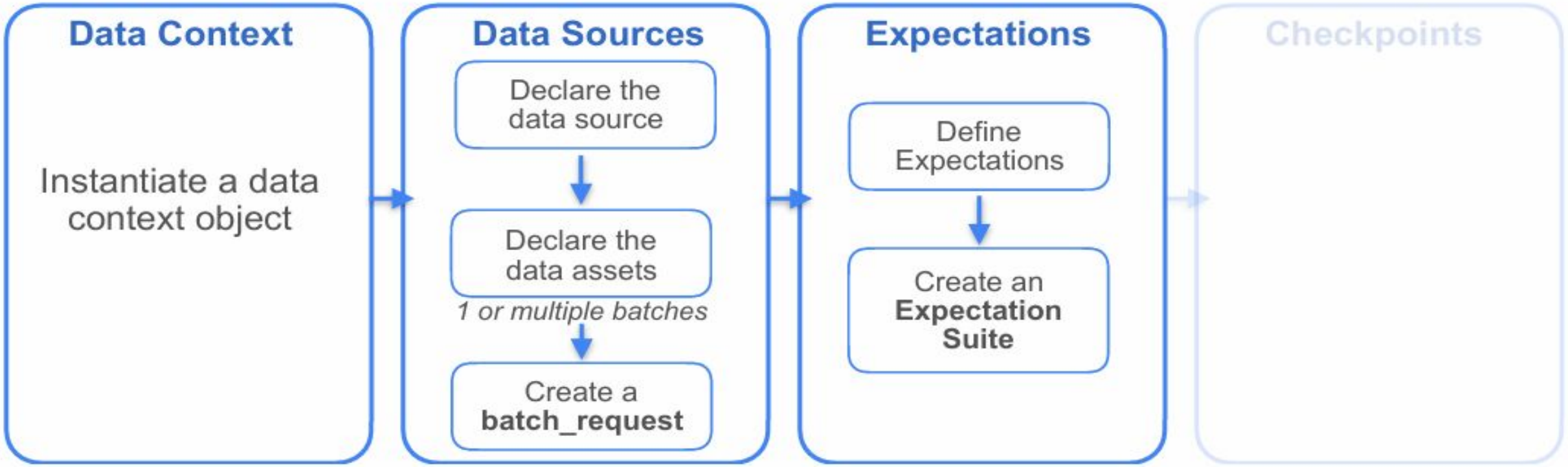
Great Expectations Workflow

- **Data source:** SQL Database, a local file system, an S3 bucket, a Pandas DataFrame
- **Data asset**: collection of records within a data source
- **Batches**: partitions from your data asset
- **Batch_request**: primary way to retrieve data from the data asset



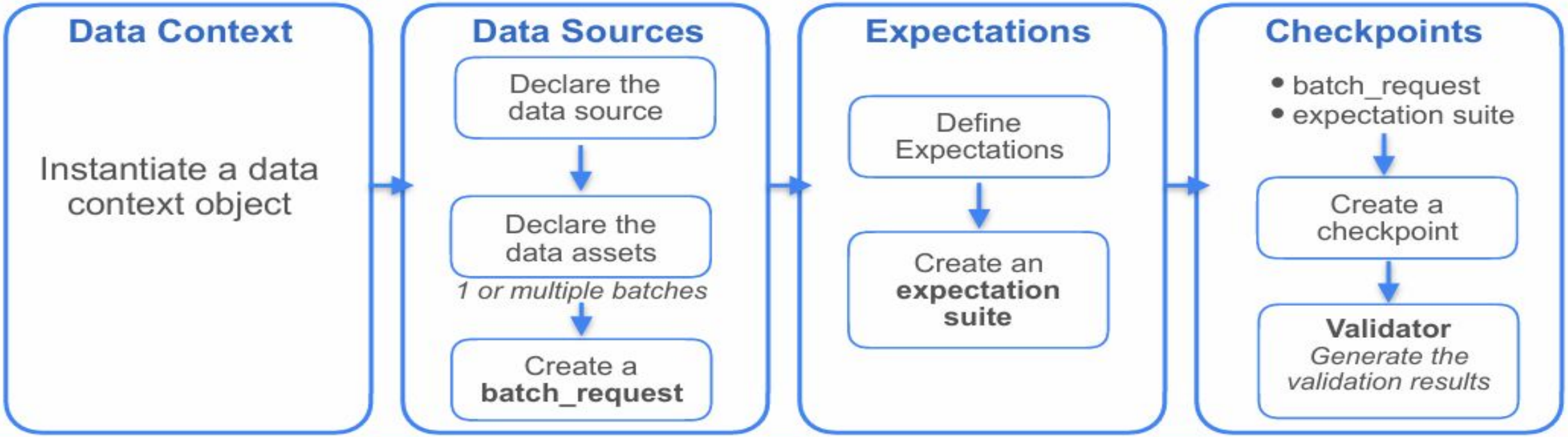
Great Expectations Workflow

- **Expectation:** statement that you can use to verify if your data meets a certain condition
  - expect_column_min_to_be_between
  - expect_column_values_to_be_unique
  - expect_column_values_to_be_null



Great Expectations Workflow

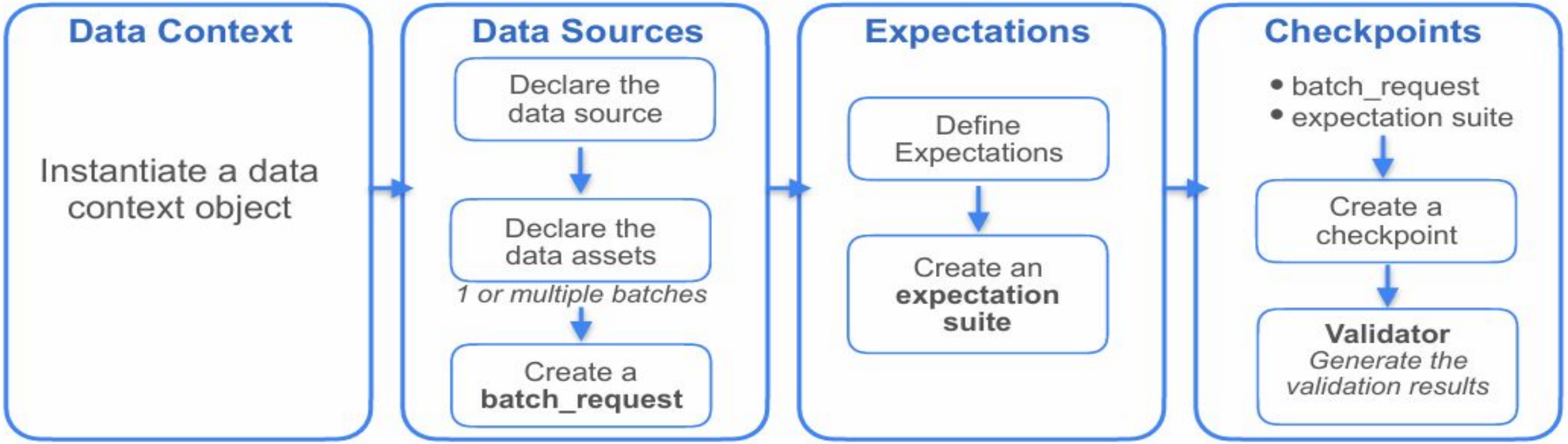
- **Validator:** expects a batch_request and its corresponding expectation suite
  - Manual interaction
  - Or automating the validation process using a checkpoint object



Great Expectations Workflow

Metadata is saved in backend stores:

- Expectation Store
- Validation Store
- Checkpoint Store
- Data docs



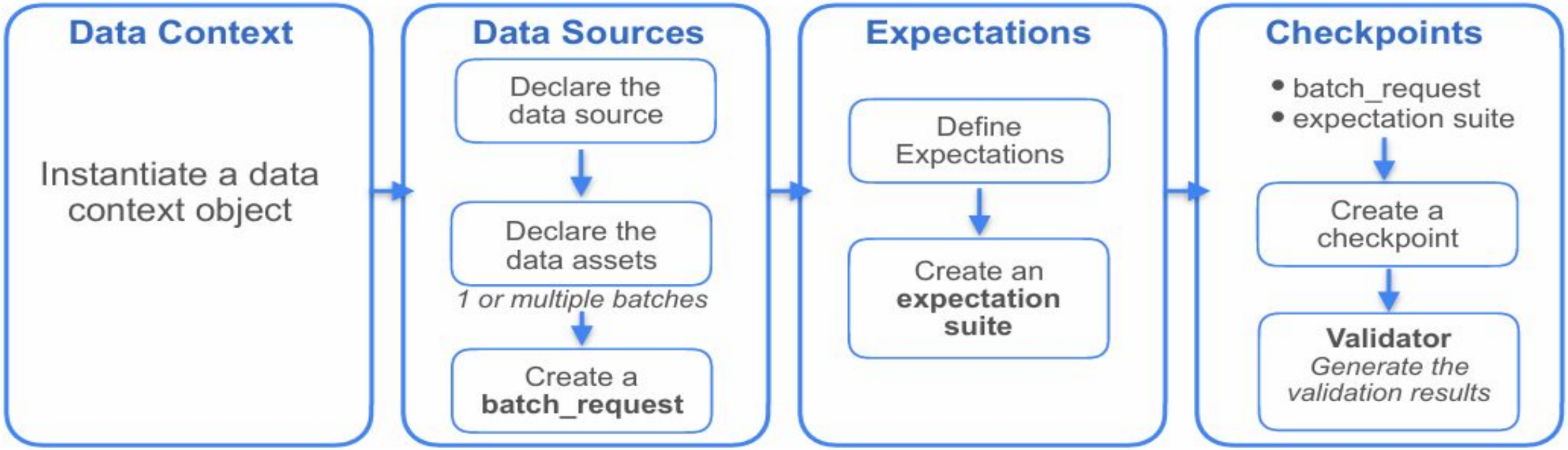
Workflow Example

DVD rental Database

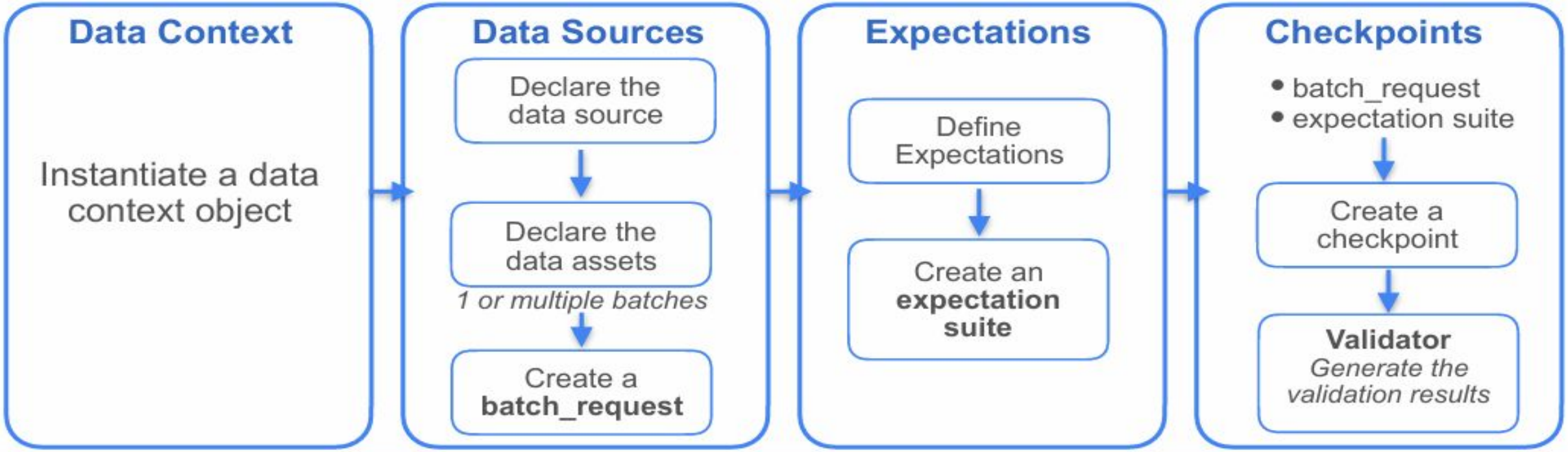
Local postgresSQL database

payment
payment_id
customer_id
staff_id
rental_id
amount
payment_date

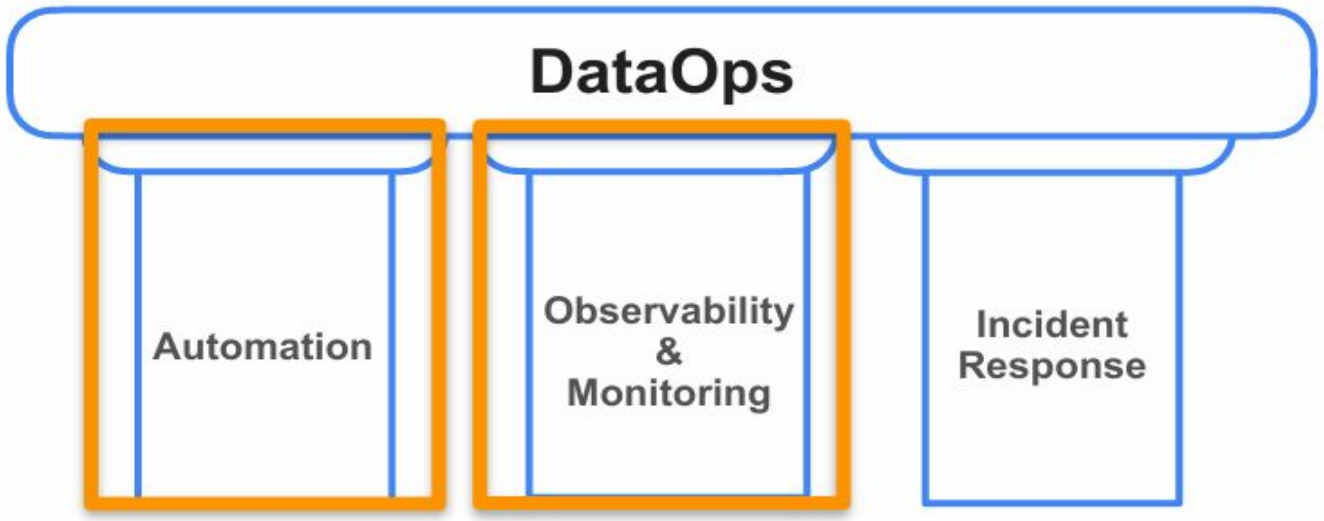
- payment_id contains unique IDs
- customer_id does not contain null values
- all the values in the amount column are non-negative



Workflow Example



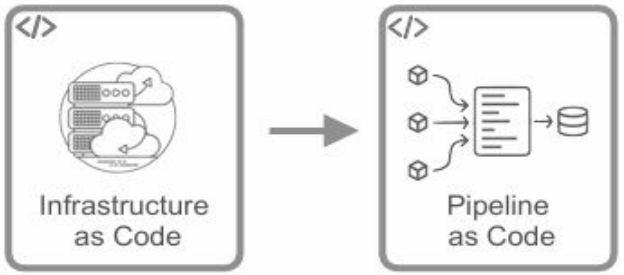
The Three Pillars of DataOps



HashiCorp Terraform

great expectations

Amazon CloudWatch



- Automate individual tasks in your data pipeline
- Build in data quality tests and monitoring

Apache Airflow

Orchestration, Monitoring, and Automating Data Pipelines

Before Orchestration

Cron

- A command line utility introduced in the 1970s
- Used to execute a particular command at a specified date and time.

Cron Job


No restriction
* * * * * command to be executed
Week (0 = Sun … 6 = Saturday)
Month (1 - 12)
Day (1 - 31)
Hour (0 - 23)
Minute (0 - 59)

Cron Job

```
0 0 1 1 *  echo "Happy New Year"
```

- Any day of the week
- 1st Month
- 1st Day
- 0th hour
- 0th minute

Every year at midnight on January 1st

```
01/01/2006 0:00:00 Happy New Year
01/01/2007 0:00:00 Happy New Year
01/01/2008 0:00:00 Happy New Year
01/01/2009 0:00:00 Happy New Year
```

Scheduling Data Pipeline with Cron

Every night at midnight

```
00*** python ingest_from_rest_api.py
```

Ingest from a REST API

Scheduling Data Pipeline with Cron

Every night at midnight

```
00*** python ingest_from_rest_api.py
```

1 AM every night

```
01*** python transform_api_data.py
```

Scheduling Data Pipeline with Cron

Every night at midnight

```
00*** python ingest_from_rest_api.py
```

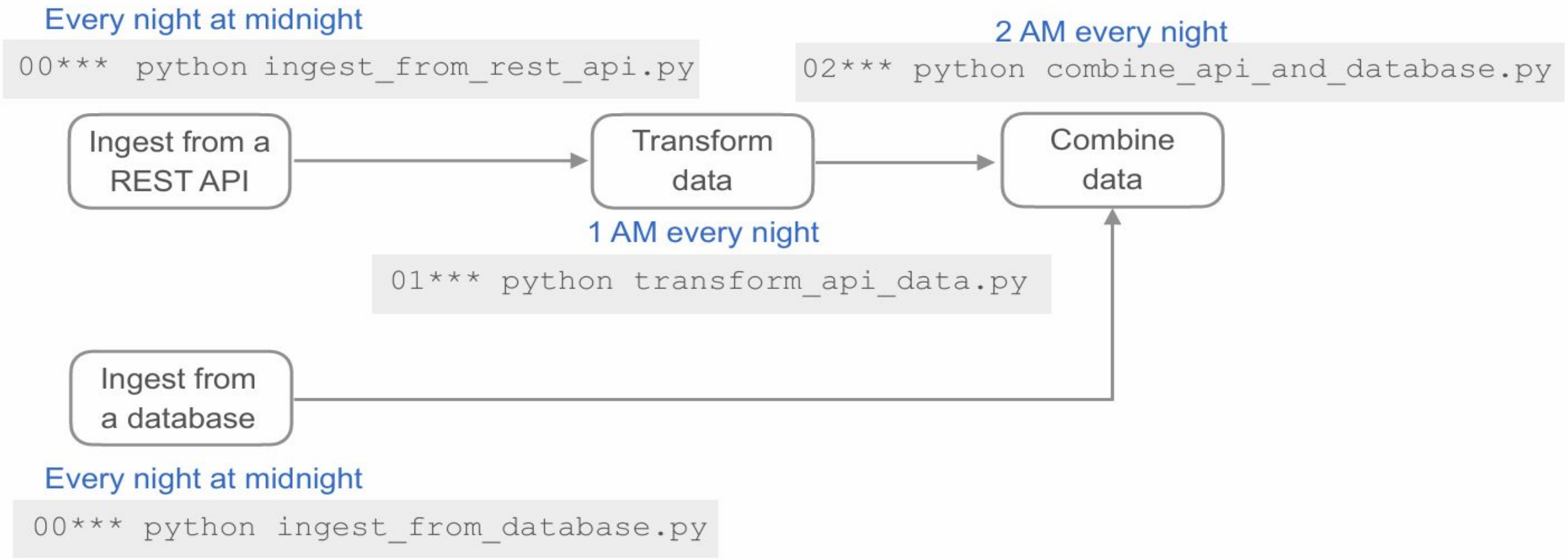
1 AM every night

```
01*** python transform_api_data.py
```

Every night at midnight

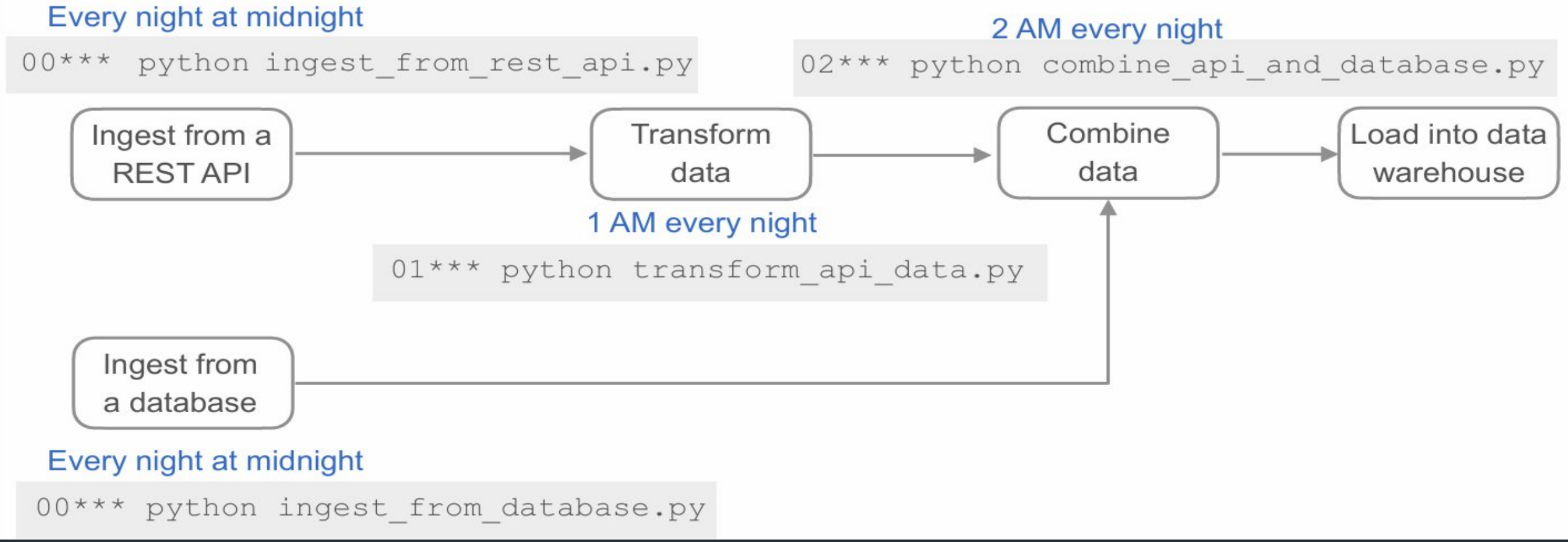
```
00*** python ingest_from_database.py
```

Scheduling Data Pipeline with Cron



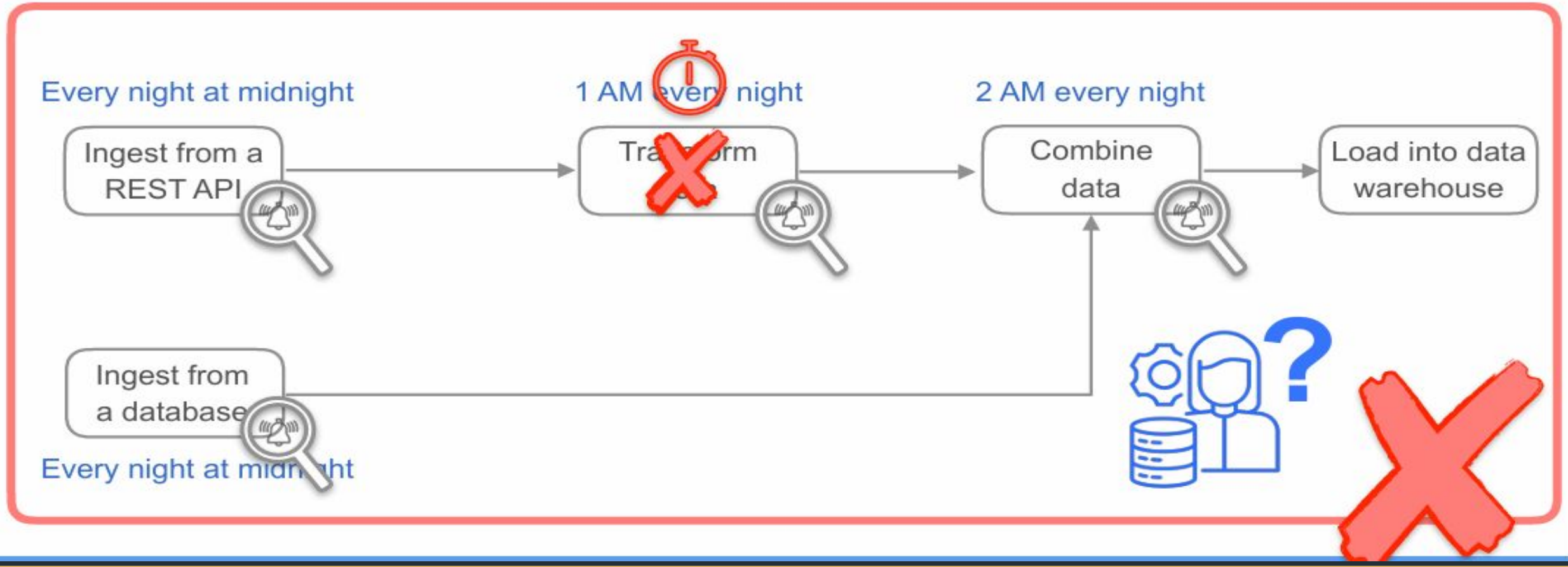
Scheduling Data Pipeline with Cron

Pure Scheduling Approach



Scheduling Data Pipeline with Cron

Pure Scheduling Approach



When To Use Cron?

- To schedule simple and repetitive tasks:

  Example: Regular data downloads

- In the prototyping phase

  Example: Testing aspects of your data pipeline

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Evolution of Orchestration Tools

Orchestration Tools

late 2000s

Dataswarm

facebook

Orchestration Tools

late 2000s

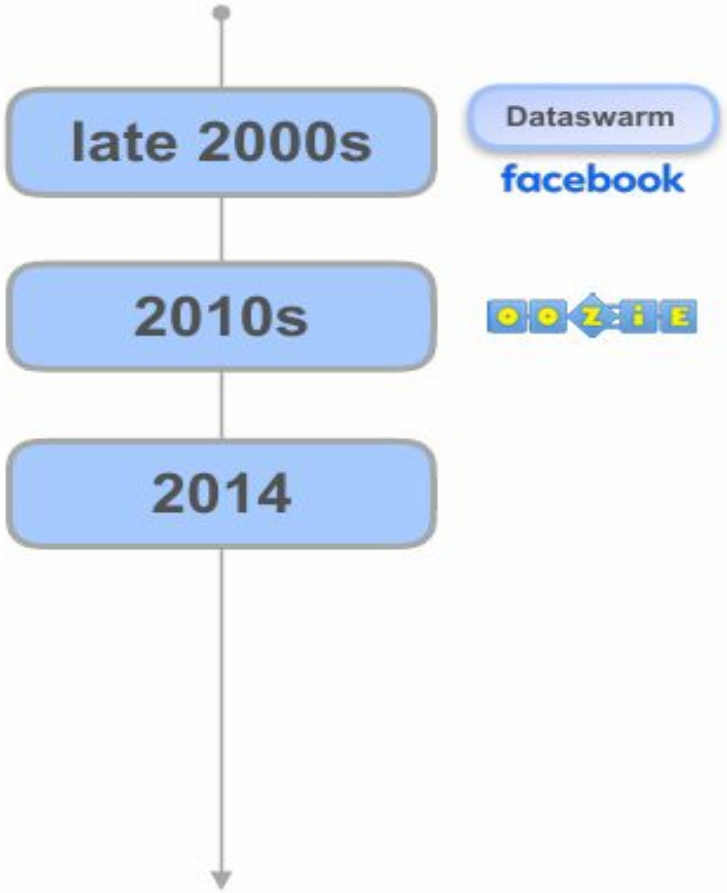
Dataswarm

facebook

2010s

- Designed to work within a Hadoop cluster
- Difficult to use in a more heterogeneous environment

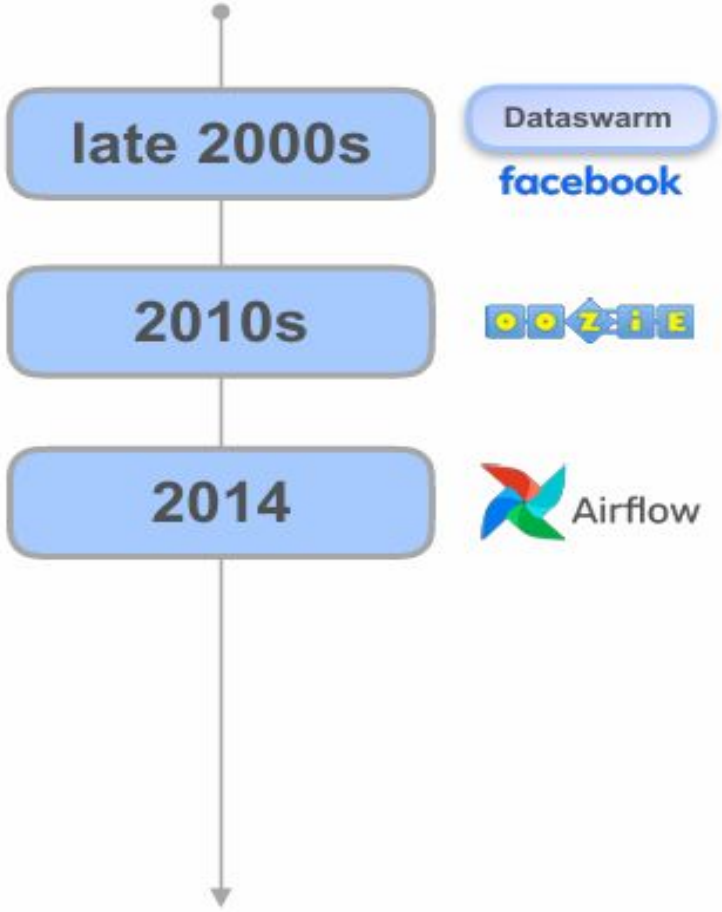
Orchestration Tools



airbnb introduced

Airflow

Orchestration Tools



Orchestration Tools

late 2000s — Dataswarm, facebook

2010s — Oozie

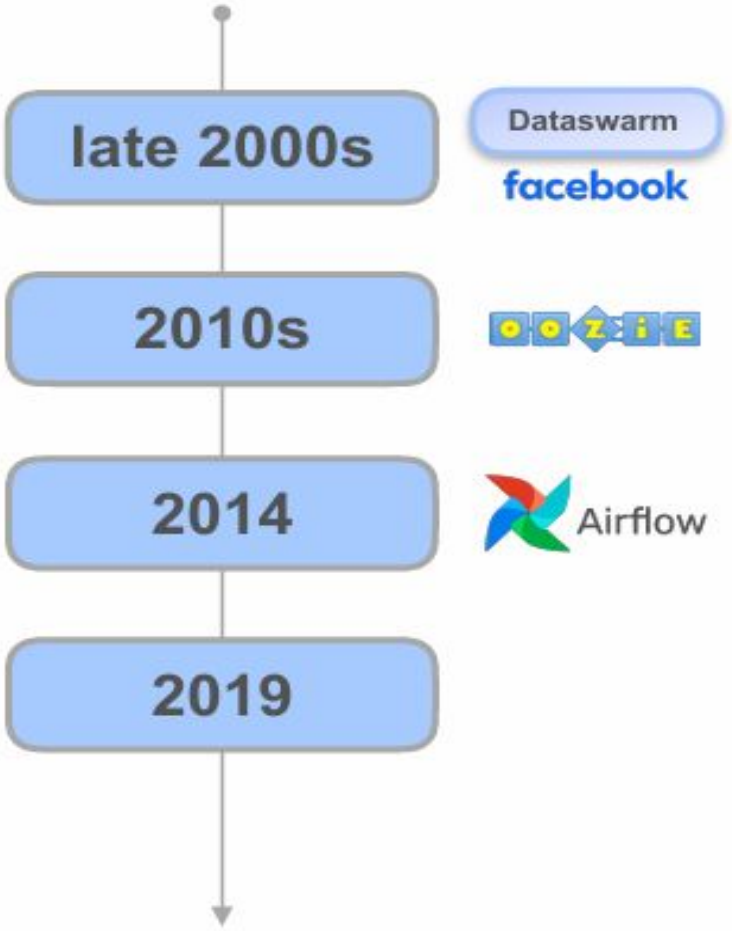
2014 — Airflow

Maxime Beauchemin

airbnb

Open source project

Orchestration Tools



Advantages and Challenges of Airflow

Advantages	Challenges
• Airflow is written in Python • The Airflow open source project is very active • Airflow is available as a managed service	• Scalability challenge • Ensuring data integrity • No support for streaming pipelines

aws

Google Cloud Platform

ASTRONOMER

Apache Airflow

Other Open-Source Orchestration Tools

Other Open-Source Orchestration Tools

PREFECT

More scalable orchestration solutions than Airflow

dagster

Built-in capabilities for data quality testing

MAGE

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

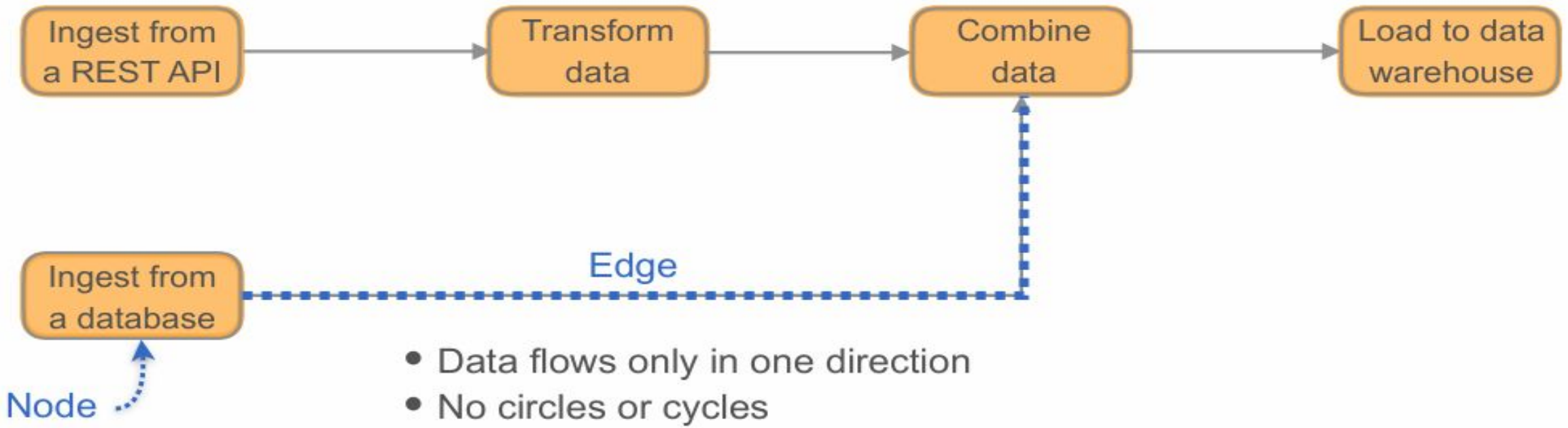
Orchestration Basics

Orchestration

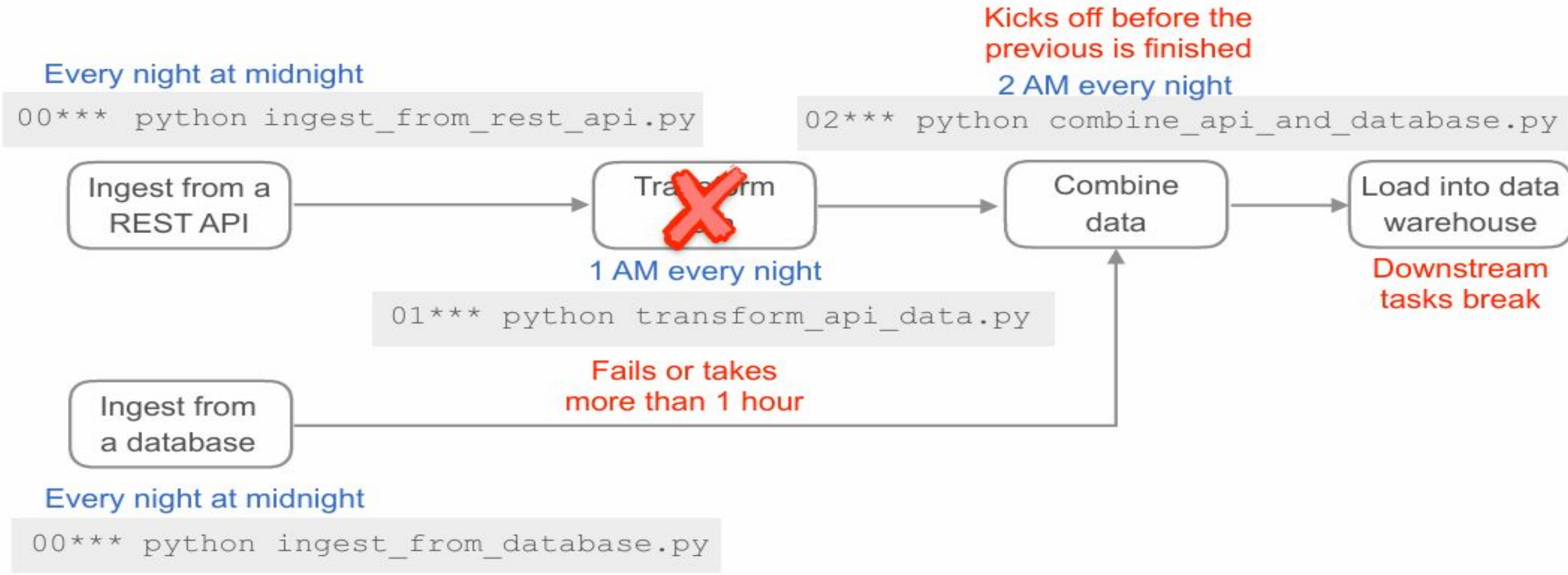
PROS	CONS
• Set up dependencies • Monitor tasks • Get alerts • Create fallback plans	• More operational overhead than simple Cron scheduling

Orchestration

Directed Acyclic Graph (DAG)

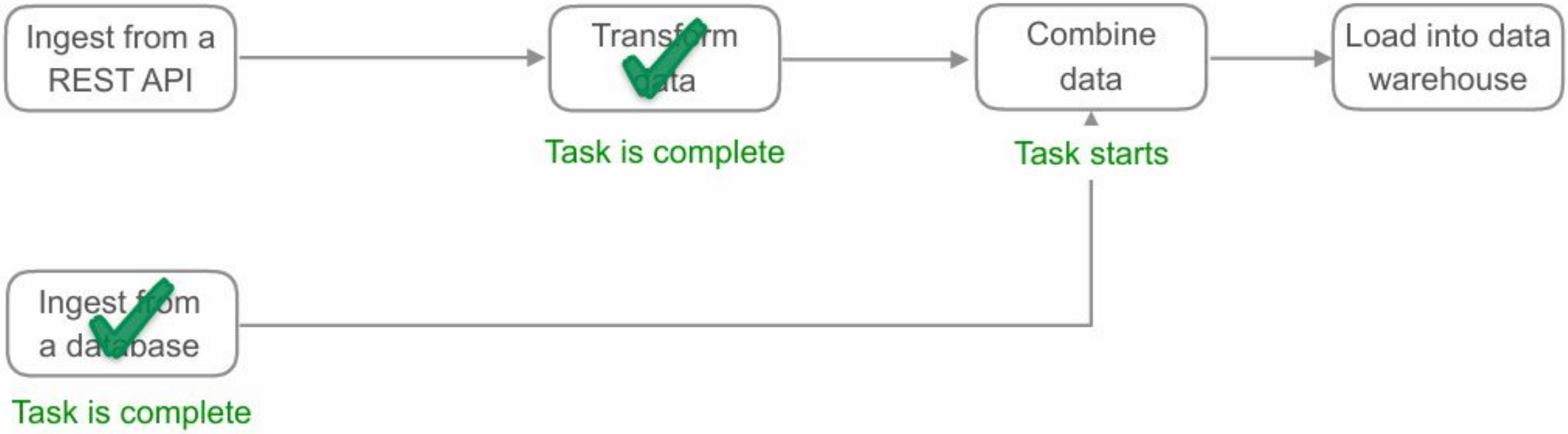


Scheduling Tasks with Cron



Orchestration

You could build in dependencies between tasks.



Orchestration in Airflow

```
with DAG(dag_id="dag_etl_example", start_date=datetime( year: 2024,  month: 3,  day: 23), schedule='@weekly'):

    task_ingest_api = PythonOperator(task_id='ingest_from_API', python_callable=ingest_from_rest_api)
    task_ingest_database = PythonOperator(task_id='ingest_from_database', python_callable=ingest_from_database)
    task_transform_api = PythonOperator(task_id = 'transform_data', python_callable=transform_api_data)
    task_combine_data = PythonOperator(task_id = 'combine_data', python_callable=combine_api_and_database)
    task_load_warehouse = PythonOperator(task_id = 'load_warehouse', python_callable=load_to_warehouse)
```

Tasks

```
[task_ingest_api >> task_transform_api, task_ingest_database] >> task_combine_data >> task_load_warehouse
```

Dependencies

Orchestration in Airflow

You can set up the conditions on which the DAG should run:

time-based conditions

event-based conditions

Orchestration in Airflow

Trigger the DAG based on a schedule

```
with DAG(dag_id="my_first_dag",
         start_date=datetime( year: 2024,  month: 3,  day: 13),
         description="First DAG",
         tags=["data_engineering_team"],
         schedule='@daily',
         catchup=False):
```

Orchestration in Airflow

Trigger the DAG based on an event

Example: when a dataset has been updated by another DAG

```
dataset = Dataset("s3://dataset-bucket/example.csv")

with DAG(dag_id="dag_etl_example",
         start_date=datetime( year: 2024,  month: 3,  day: 23),
         schedule=[dataset],
         catchup= False):
```

Orchestration in Airflow

Trigger a portion of the DAG based on an event

Example: presence of a file in an S3 bucket

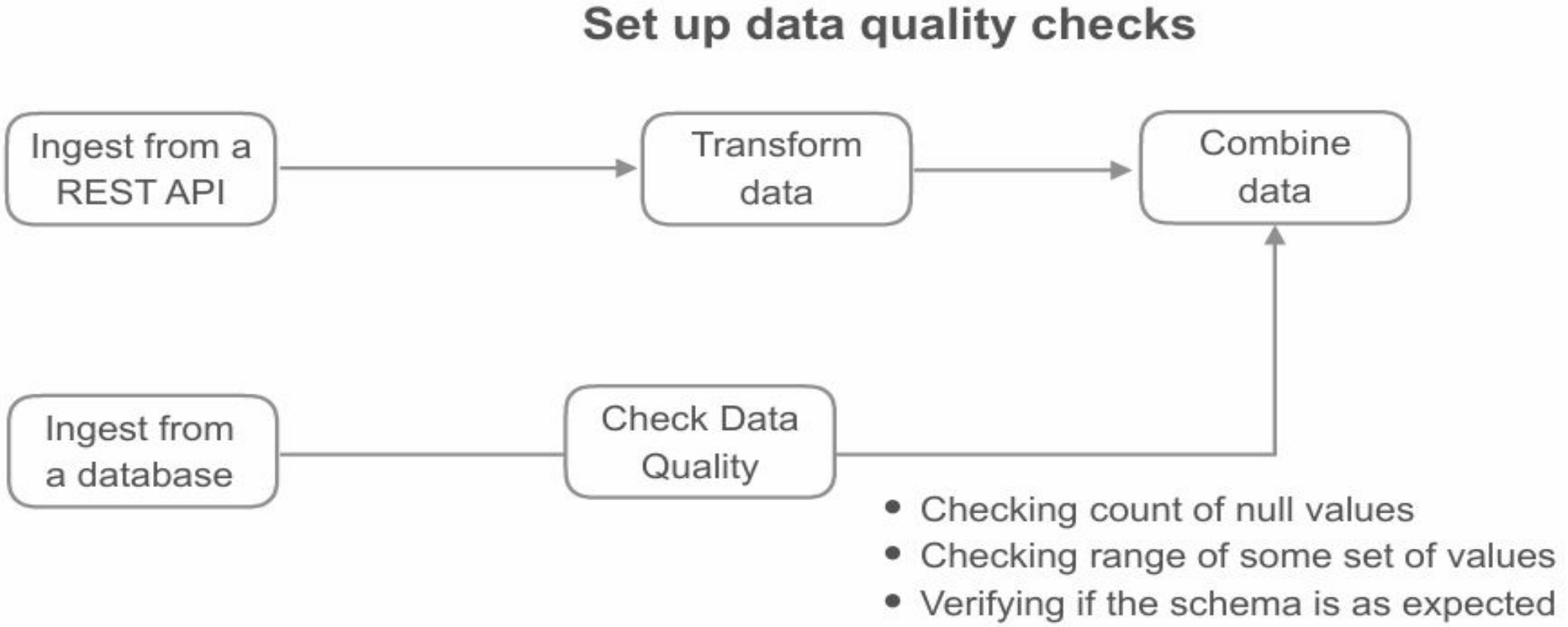
```
s3_sensor = S3KeySensor(task_id='s3_file_check',
                        bucket_key='my_file.csv',
                        bucket_name='my_bucket_name',
                        aws_conn_id='my_aws_connection')
```

Orchestration in Airflow

Set up monitoring, logging and alerts

```
t1 = PythonOperator(
    task_id="ingest_data",
    python_callable=extract_data,
    email=['someone@deeplearning.ai'],
    email_on_failure=True,
    email_on_retry=True
)
```

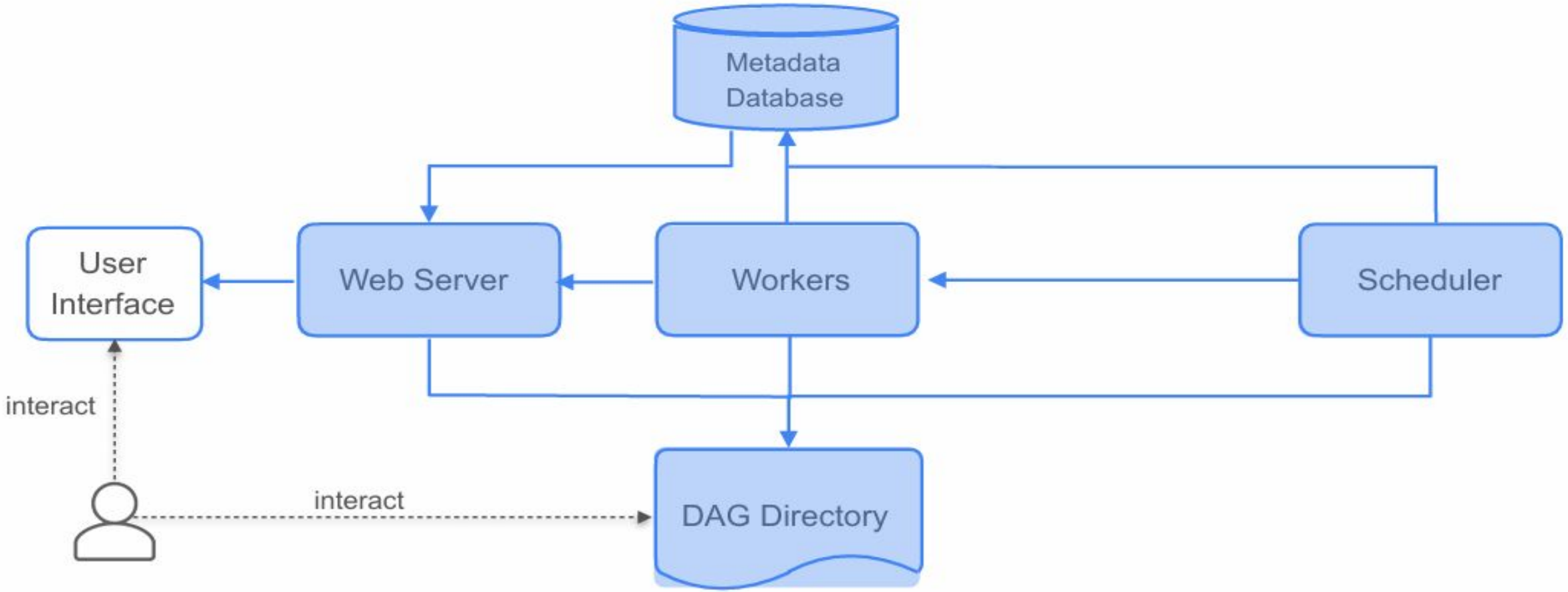
Orchestration in Airflow



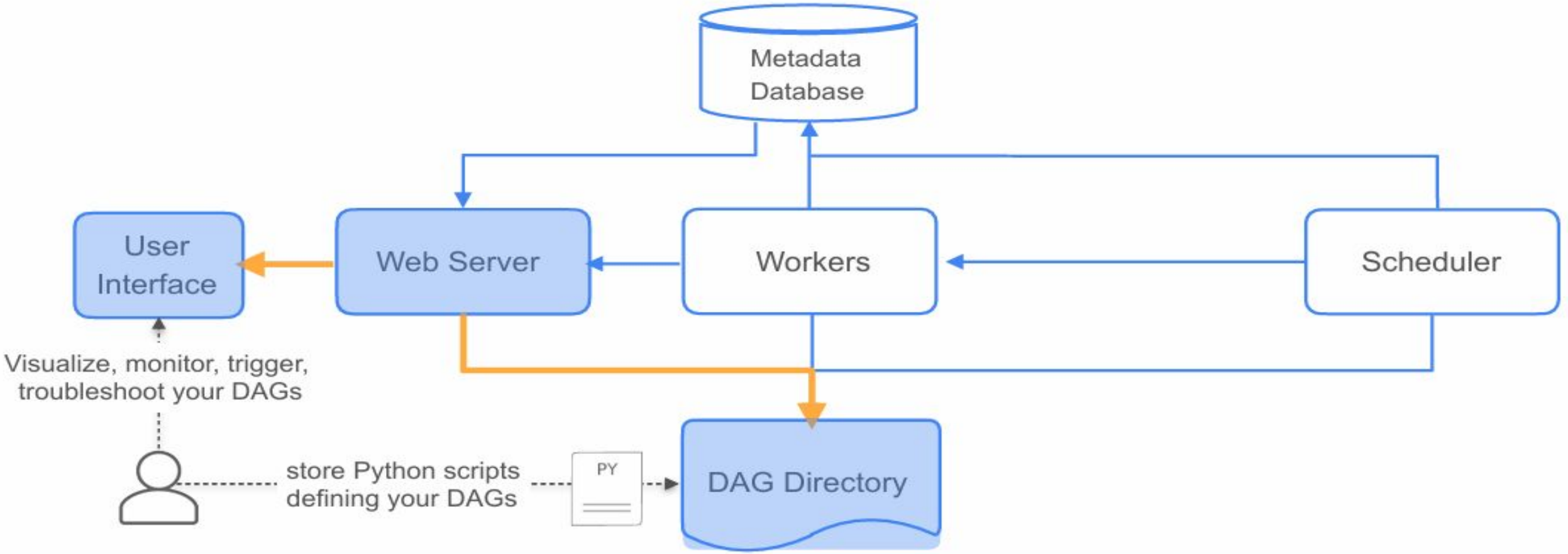
RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Airflow – Core Components

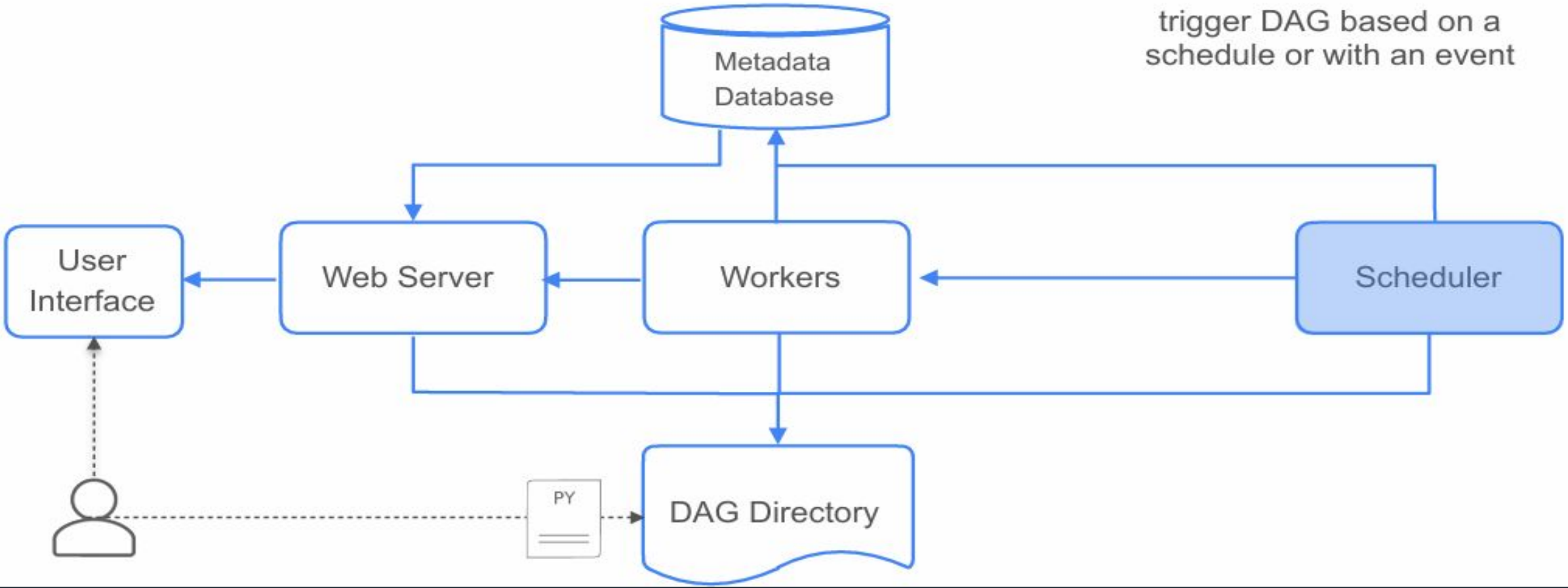
Airflow Components



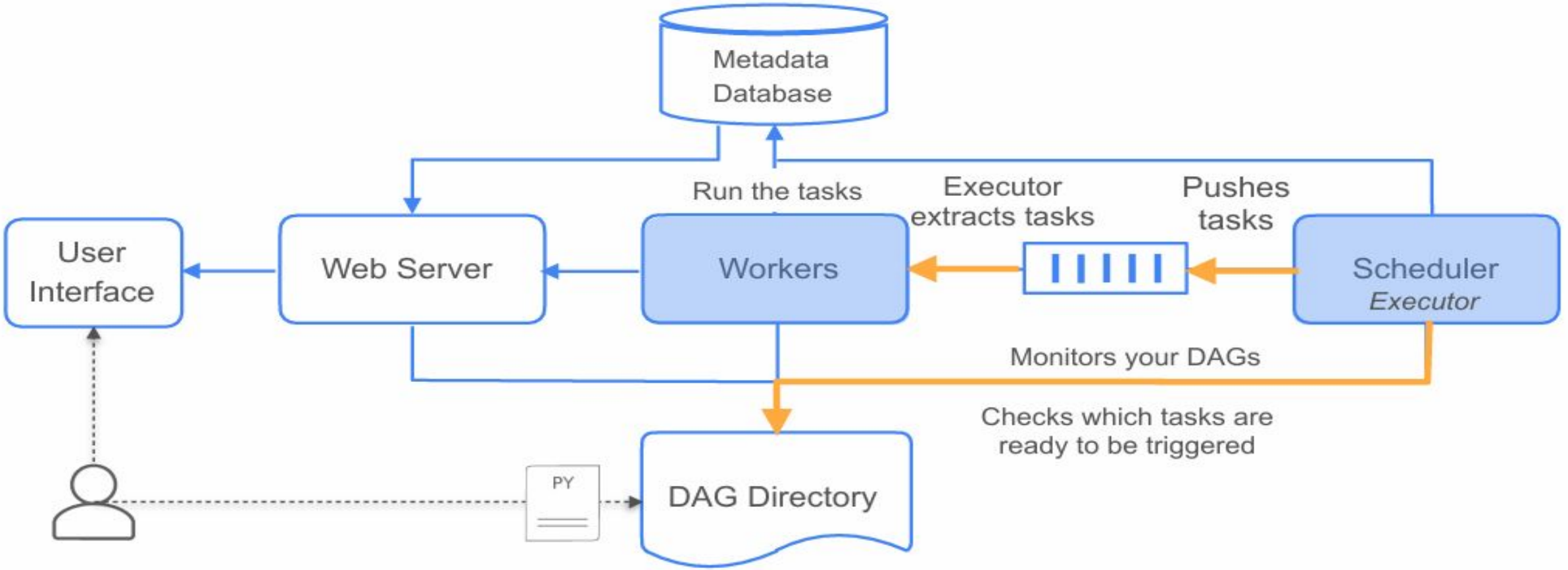
Airflow Components



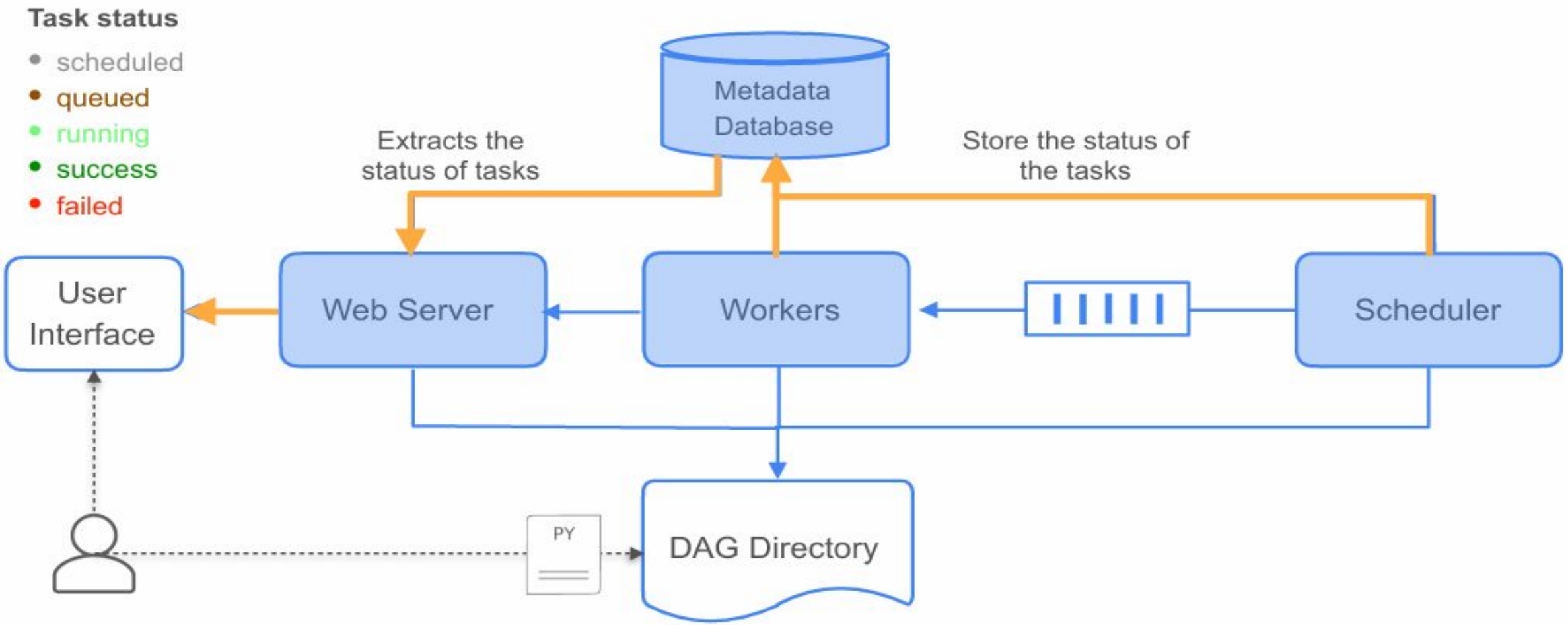
Airflow Components



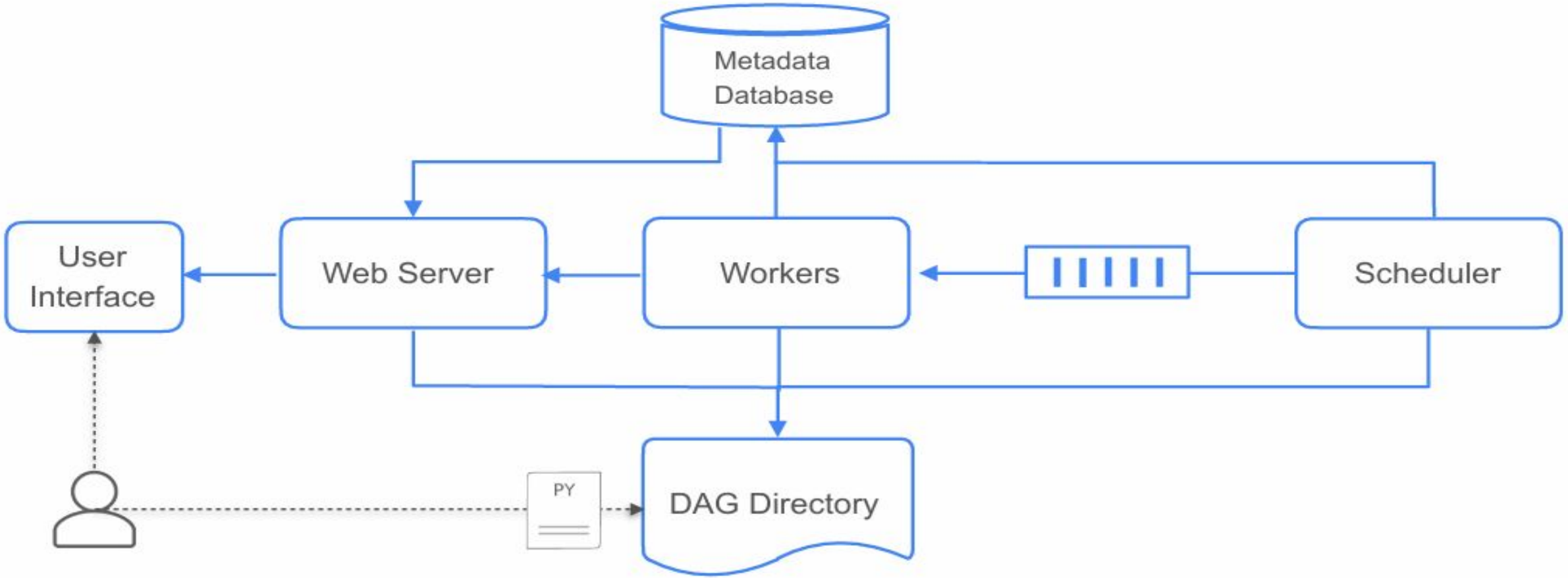
Airflow Components



Airflow Components



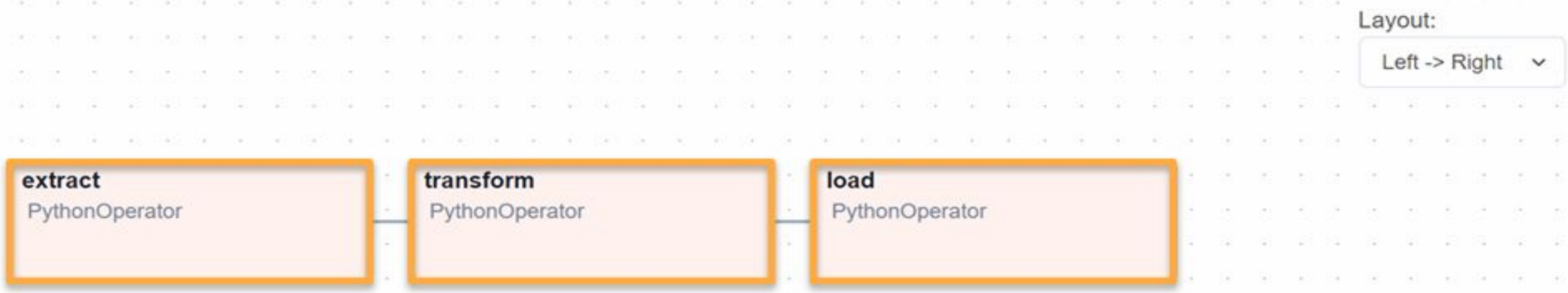
Airflow Components



RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Airflow – Creating a DAG

DAG Example


Layout:
Left -> Right
extract
PythonOperator
transform
PythonOperator
load
PythonOperator

Airflow Operators

- Operators are Python classes used to encapsulate the logic of the tasks.
- You can import Operators into your code.
- Each task is an instance of an Airflow Operator.

Airflow Operators

Types of Operators:

- `PythonOperator`: execute a python script
- `BashOperator`: execute bash commands
- `EmptyOperator`: organize the DAGs
- `EmailOperator`: send you notification via an email
- `Sensor`: special type of Operator used to make your DAGs event-driven

Airflow Operators