



Building a Data Warehouse

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYS and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

A data warehouse should consolidate data from many sources



The data in a warehouse should have quality, consistency, and accuracy



A data warehouse should be optimized for simplicity of access and high-speed query performance



A modern data warehouse

- Gigabytes to petabytes

A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops,
including ad hoc queries

A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops,
including ad hoc queries
- Ecosystem of visualization and
reporting tools

A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops,
including ad hoc queries
- Ecosystem of visualization and
reporting tools
- Ecosystem of ETL and data
processing tools

A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops,
including ad hoc queries
- Ecosystem of visualization and
reporting tools
- Ecosystem of ETL and data
processing tools
- Up-to-the-minute data

A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops,
including ad hoc queries
- Ecosystem of visualization and
reporting tools
- Ecosystem of ETL and data
processing tools
- Up-to-the-minute data
- Machine learning

A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops,
including ad hoc queries
- Ecosystem of visualization and
reporting tools
- Ecosystem of ETL and data
processing tools
- Up-to-the-minute data
- Machine learning
- Security and collaboration

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYS and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration



BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration



BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration



BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration



BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration



BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration



BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration



BigQuery has many capabilities that make it an ideal data warehouse

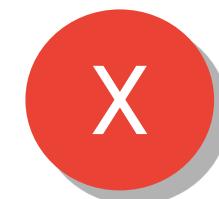
- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration



Demo

Query TB+ of data in seconds

BigQuery is serverless. You don't need to worry about:



Data aging



Query engine
optimization



Backups



Hardware



Storage management



Updates



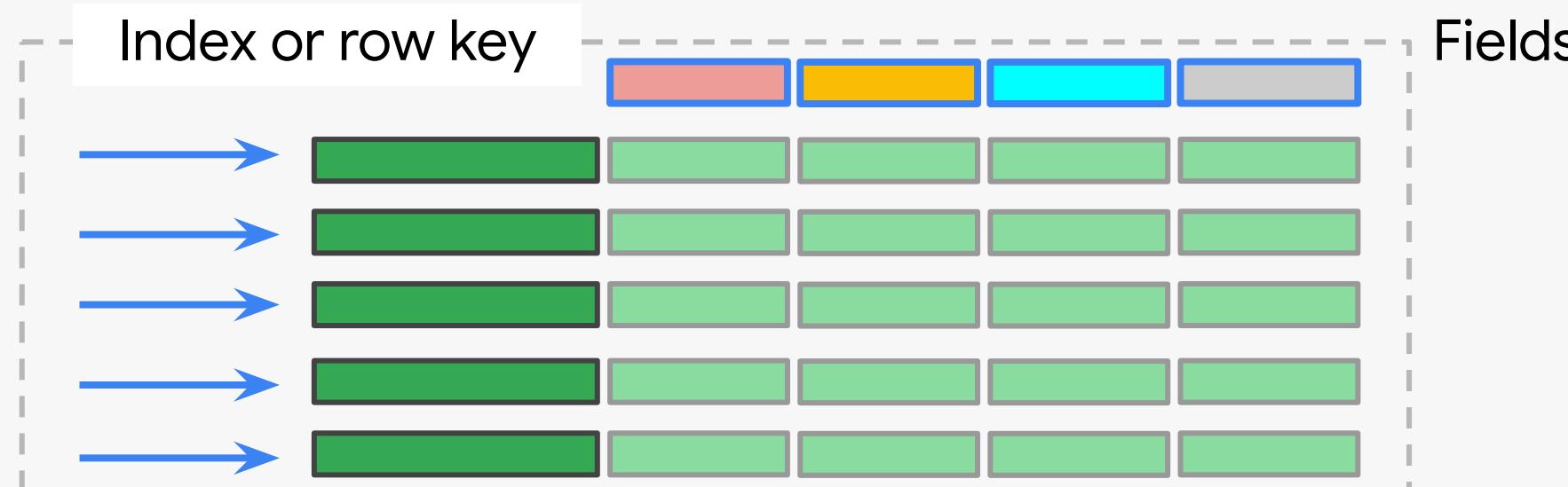
Fault recovery

Free up real people-hours
by not having to worry
about common tasks

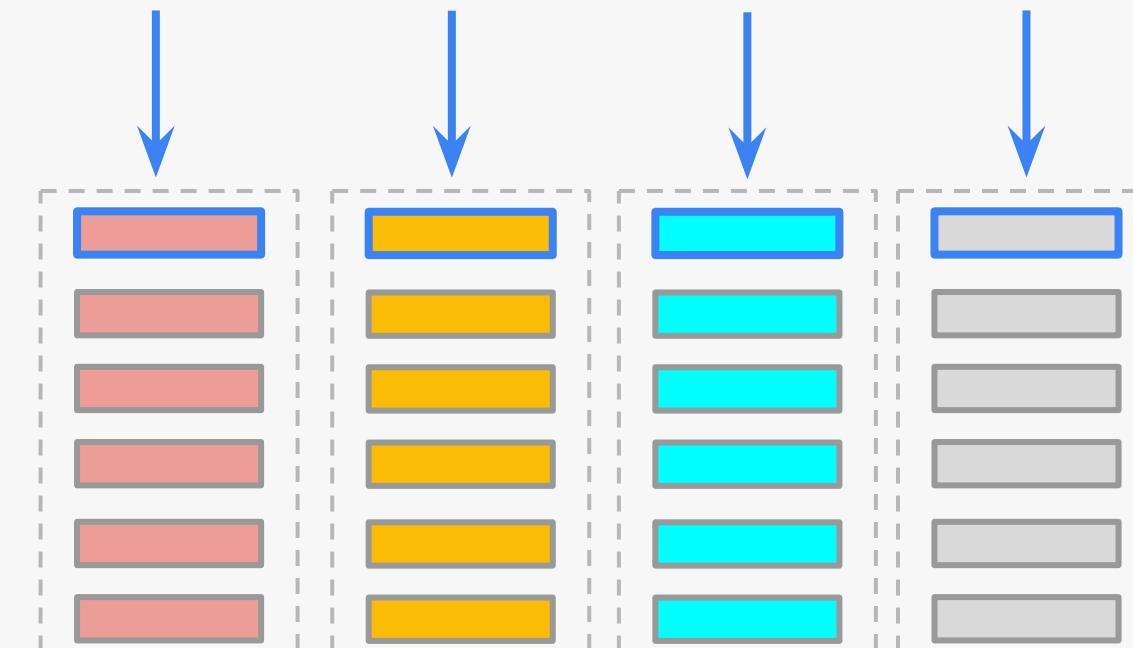


What makes BigQuery fast?

BigQuery tables are
column-oriented



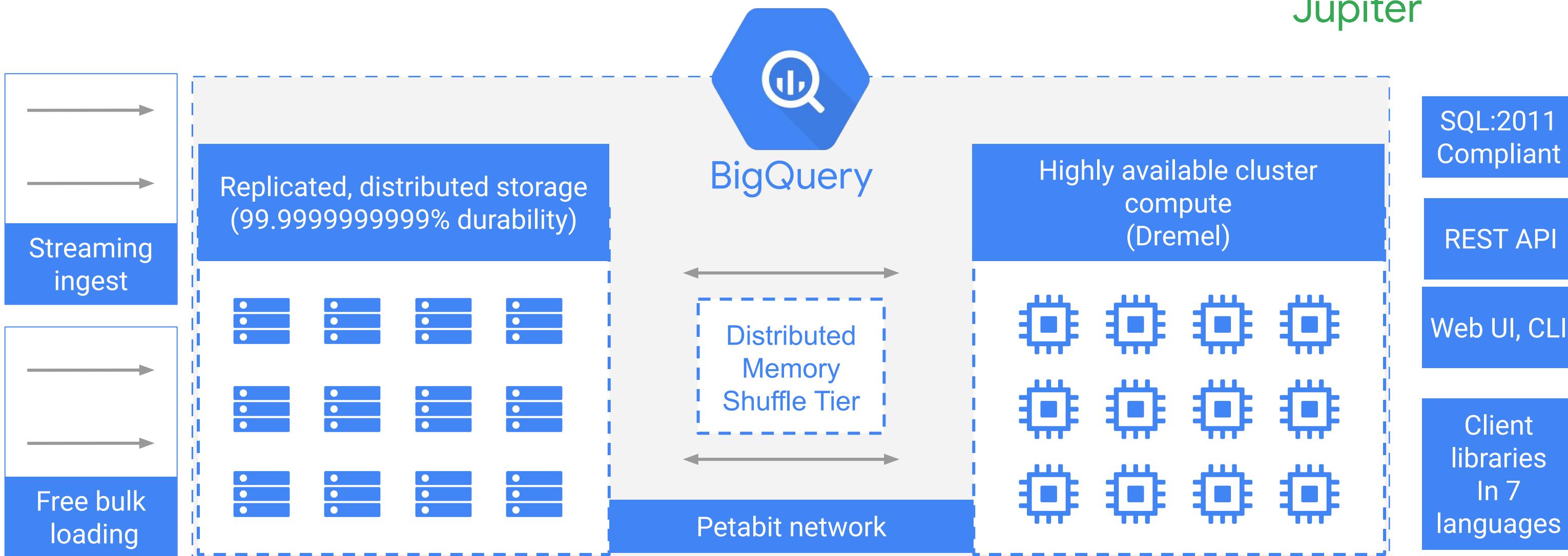
Relational Database tables are row-oriented



Columns

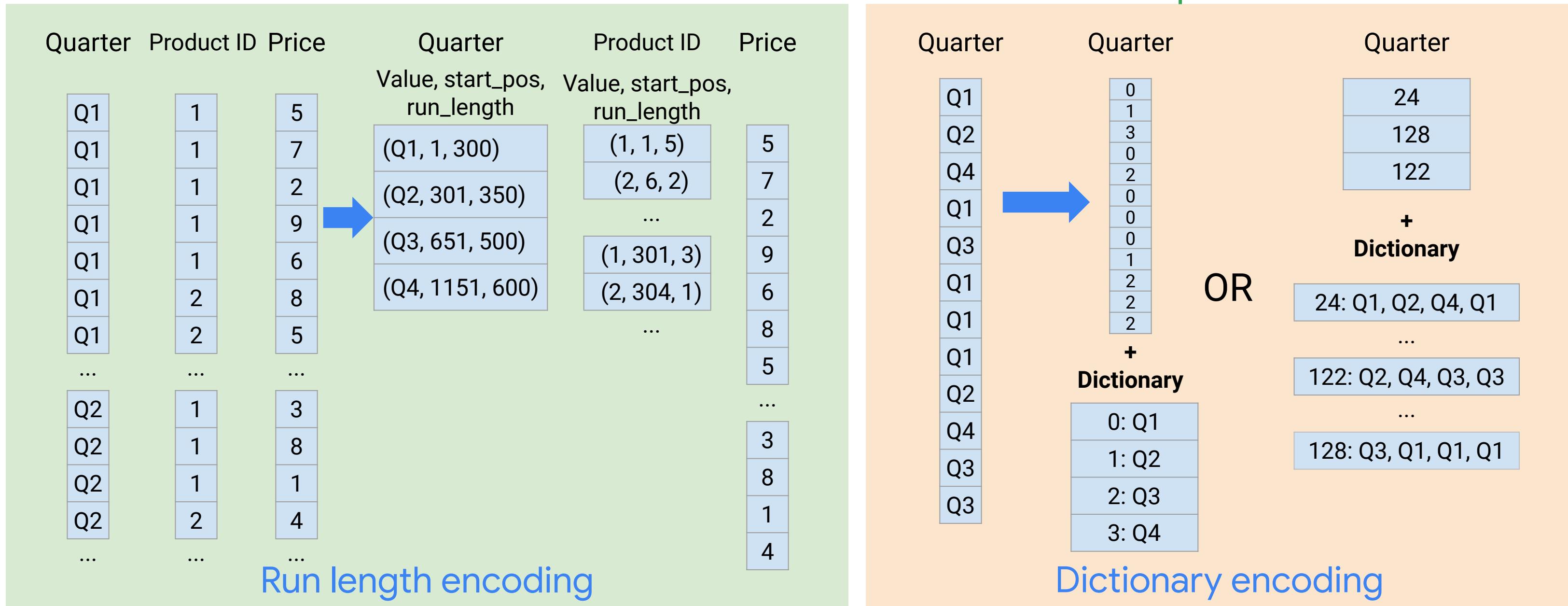
The data is physically stored in a redundant way separate from the compute cluster

Colossus
Jupiter



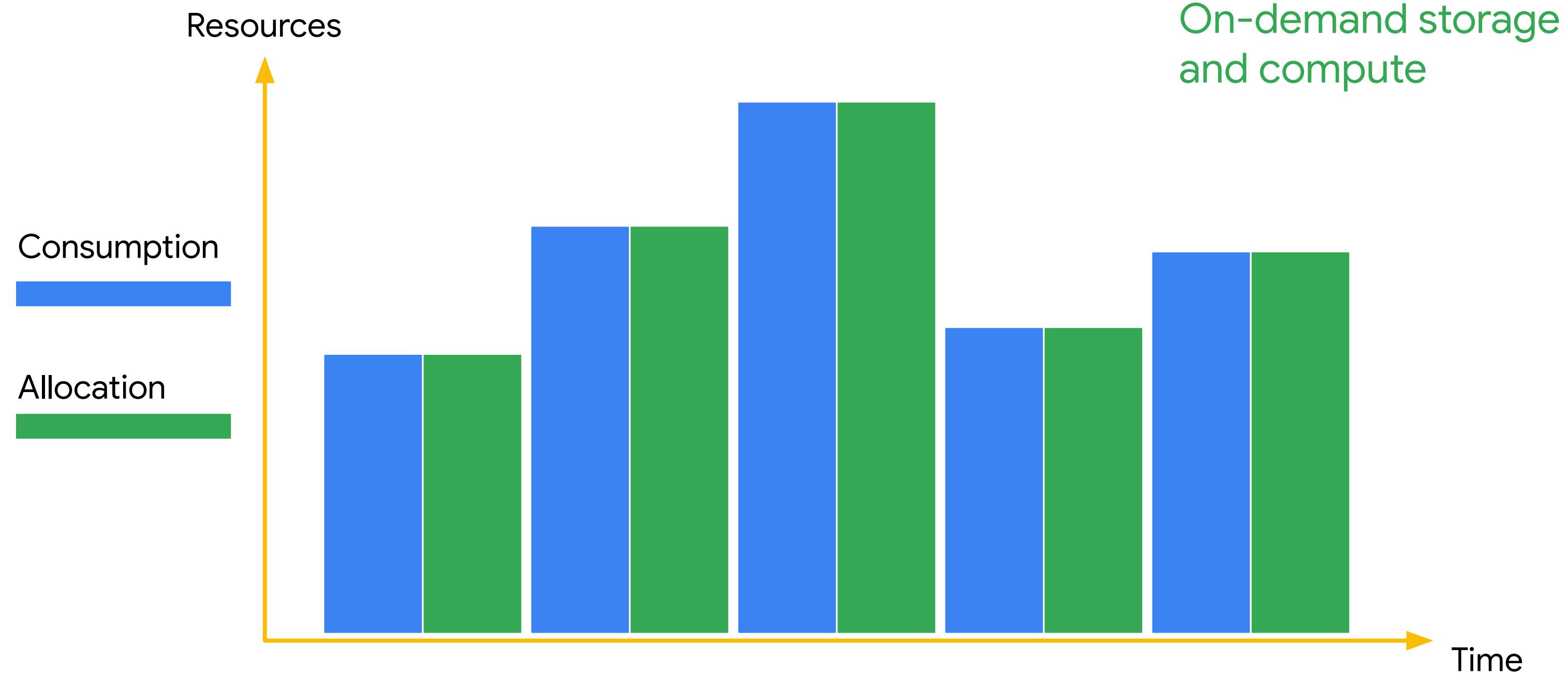
The data are also run length-encoded and dictionary-encoded

Capacitor

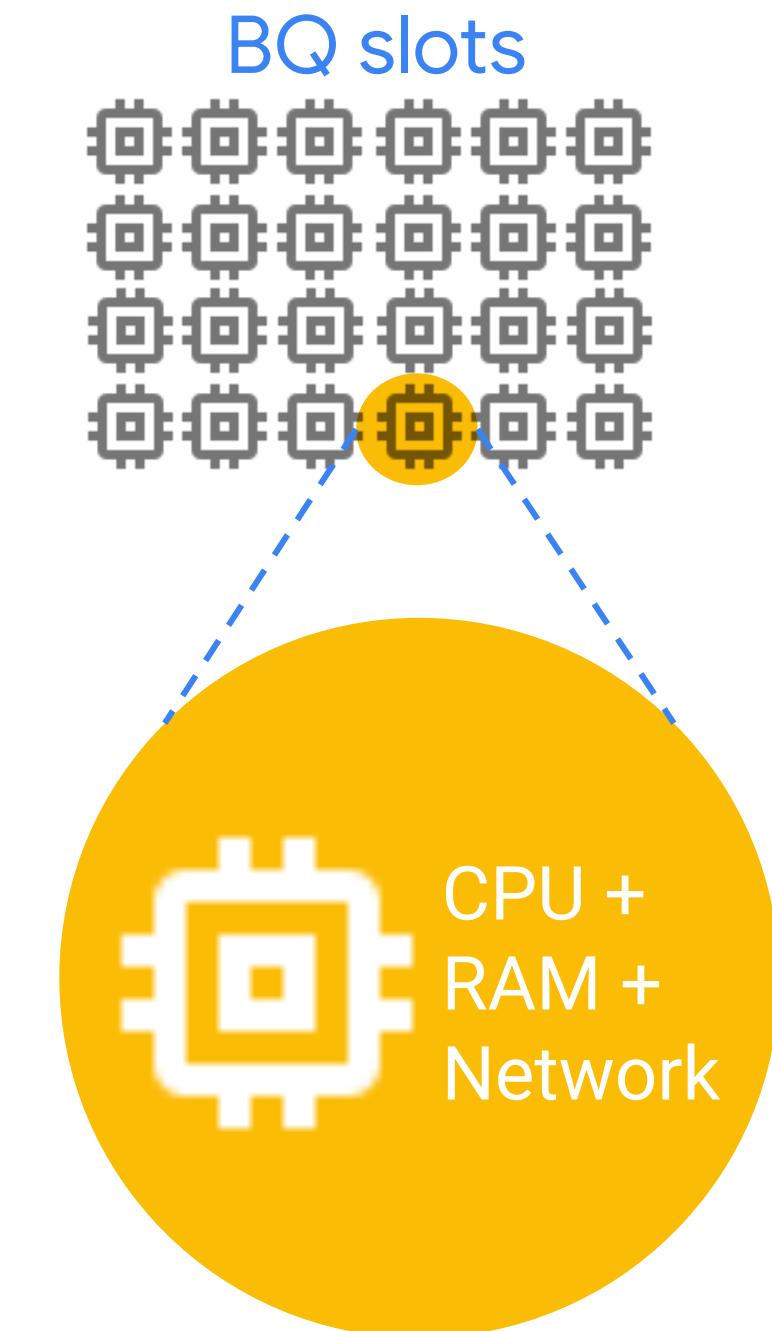


Examples taken from VLDB 2009 tutorial on Column Oriented Database Systems

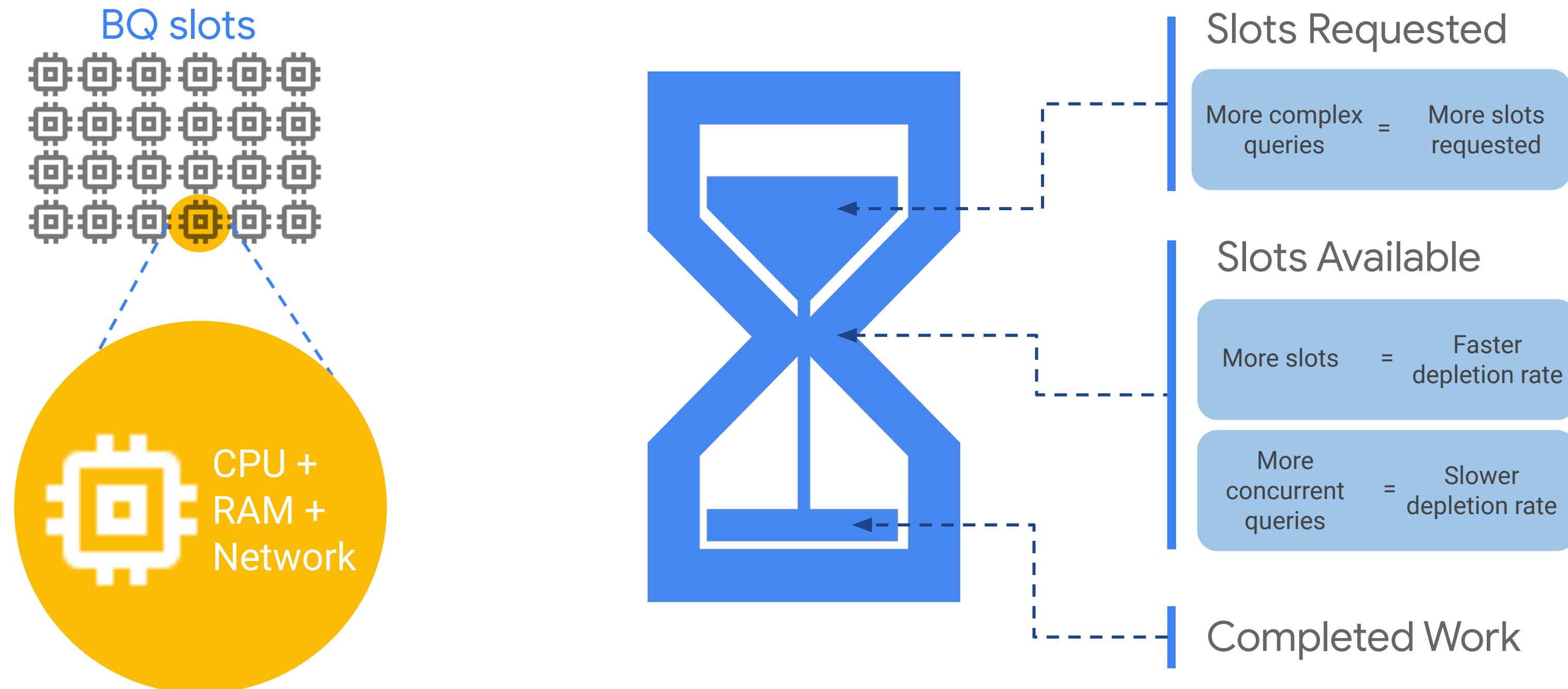
You don't need to provision resources before using BigQuery



A BigQuery slot is a combination of CPU, memory, and networking resources



The actual number of slots allotted to a query depends on query complexity and project quota



Agenda

The modern data warehouse

Intro to BigQuery

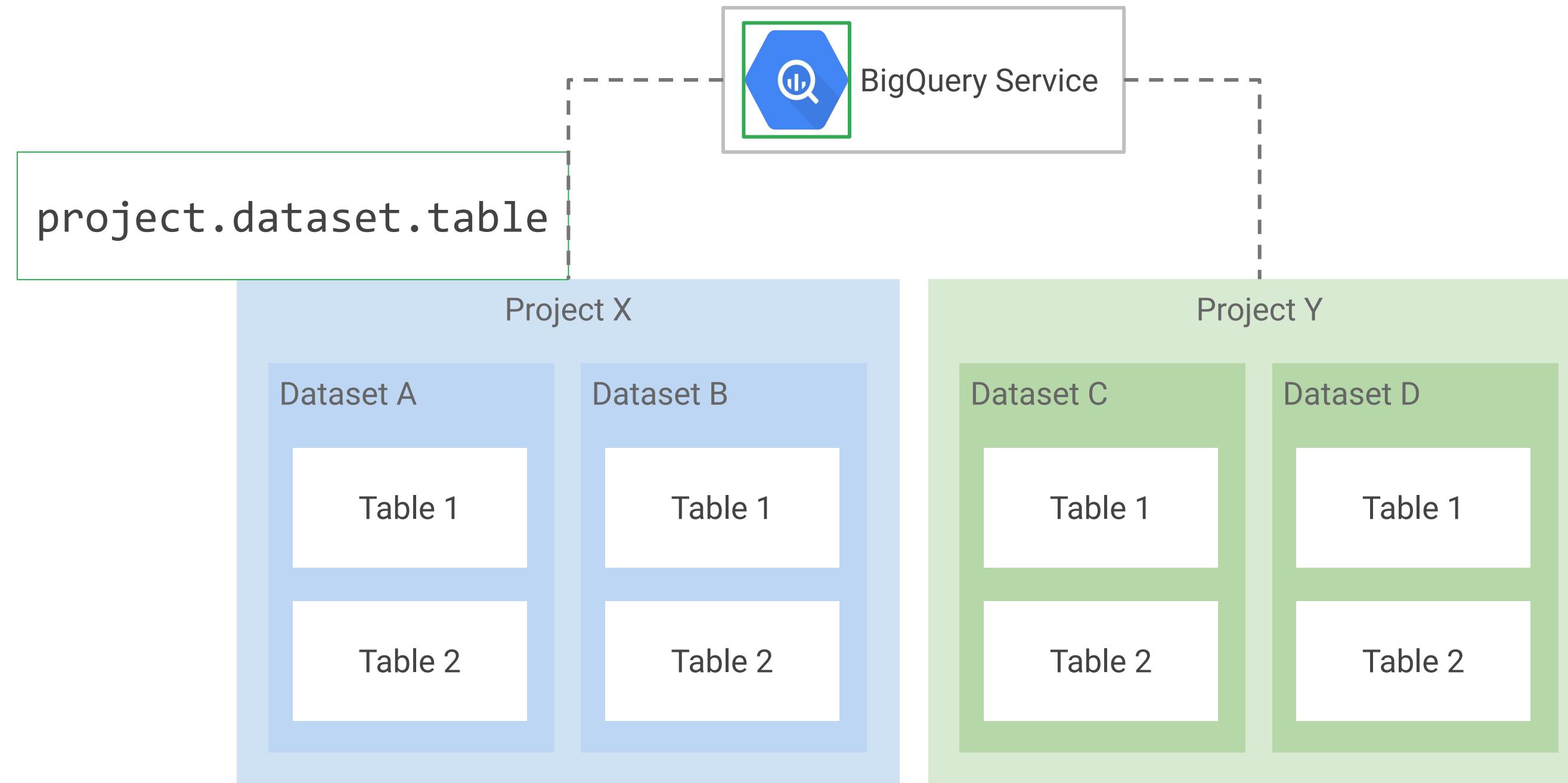
- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

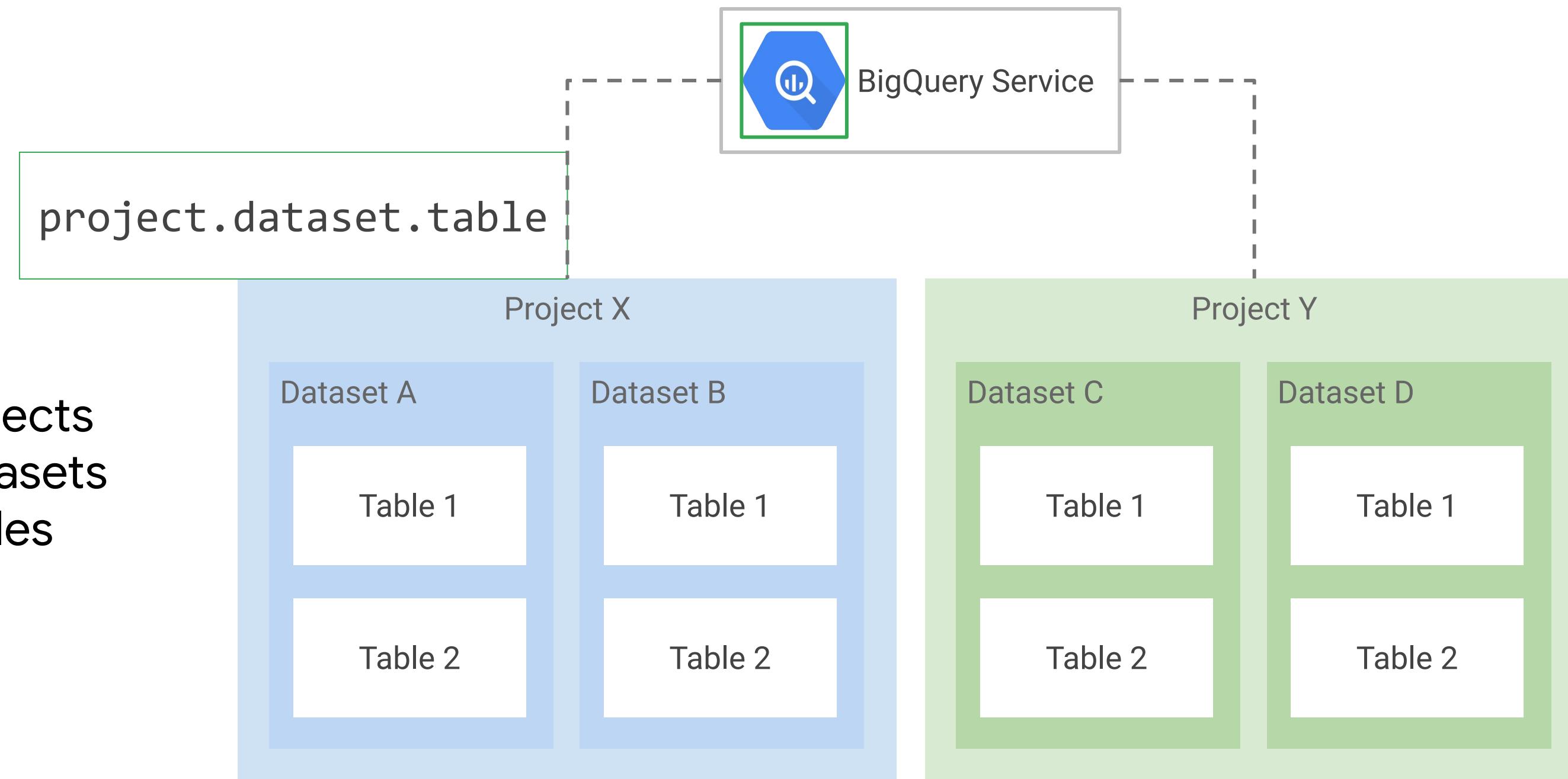
- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYS and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

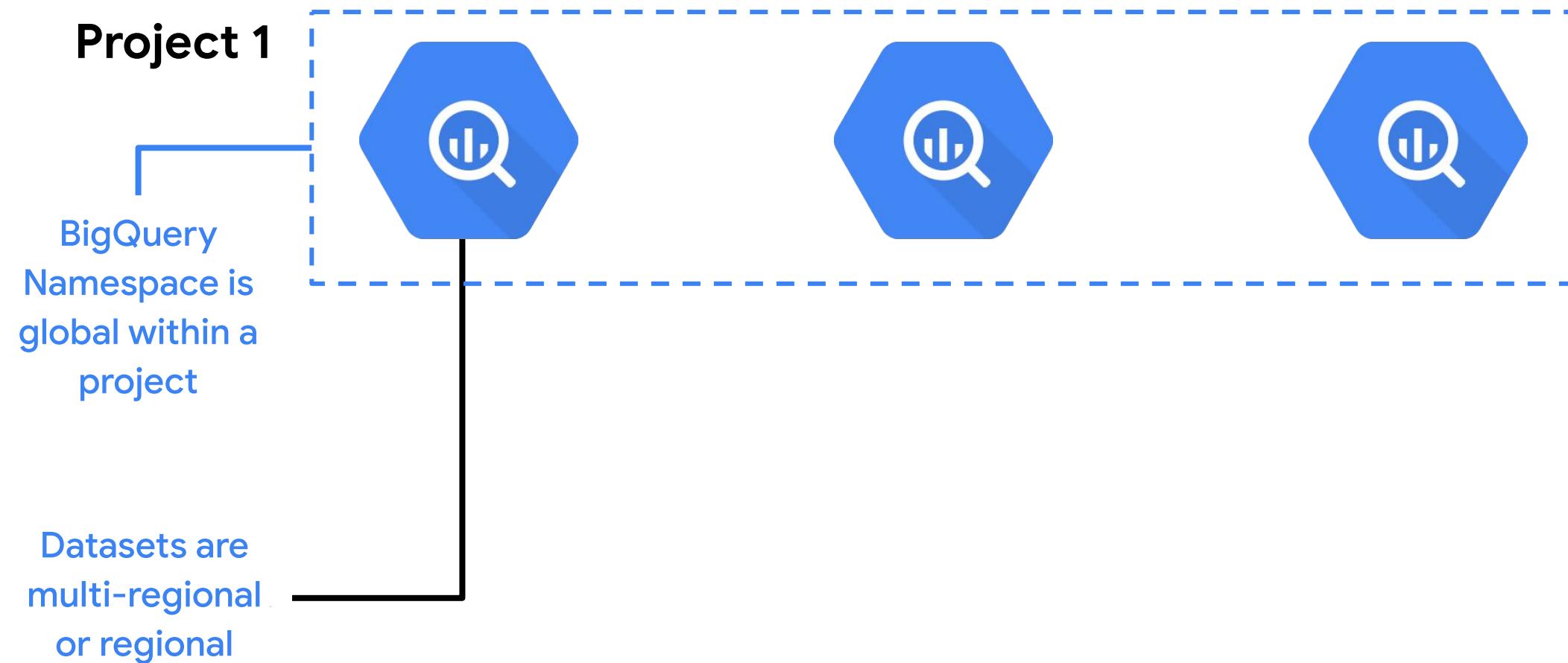
BigQuery organizes data tables into units called datasets



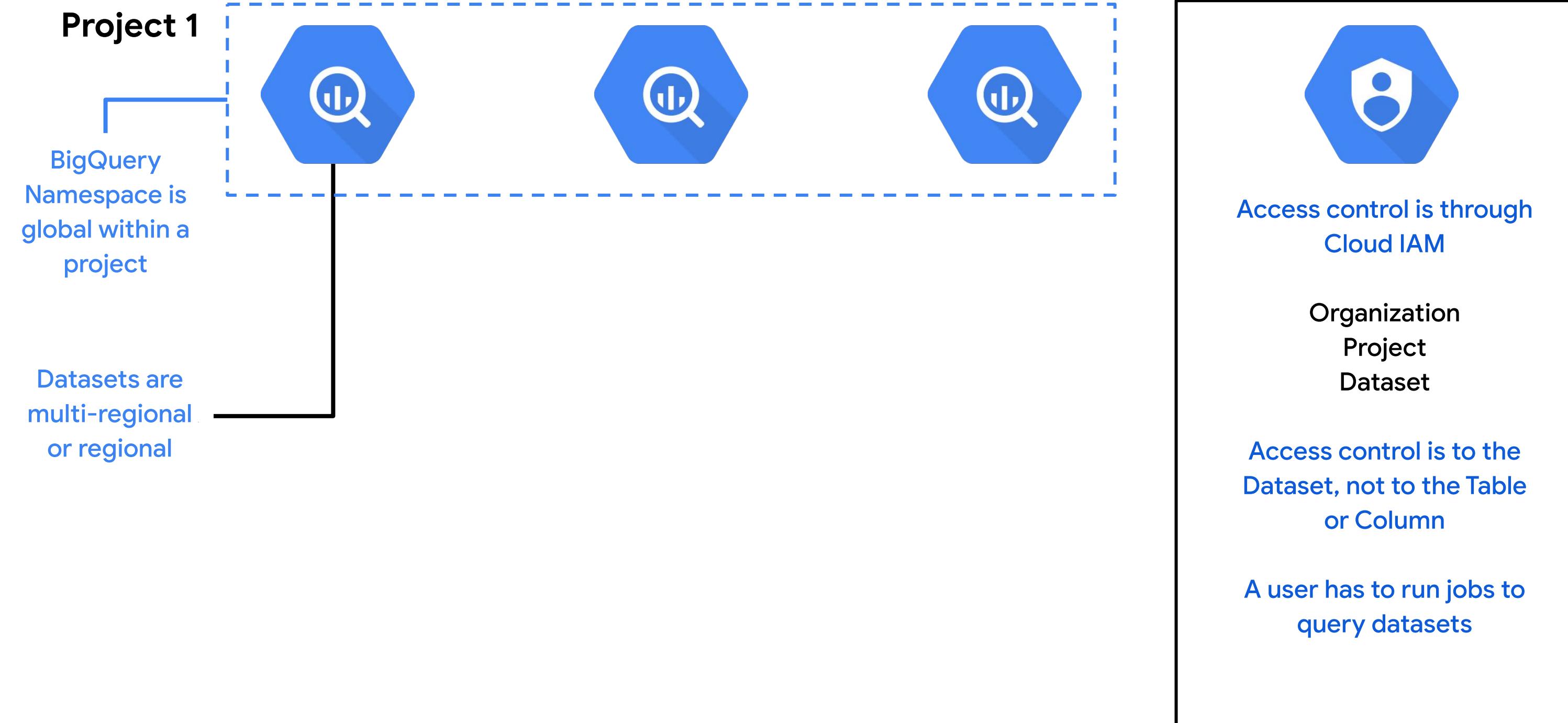
What are some reasons to structure your information?



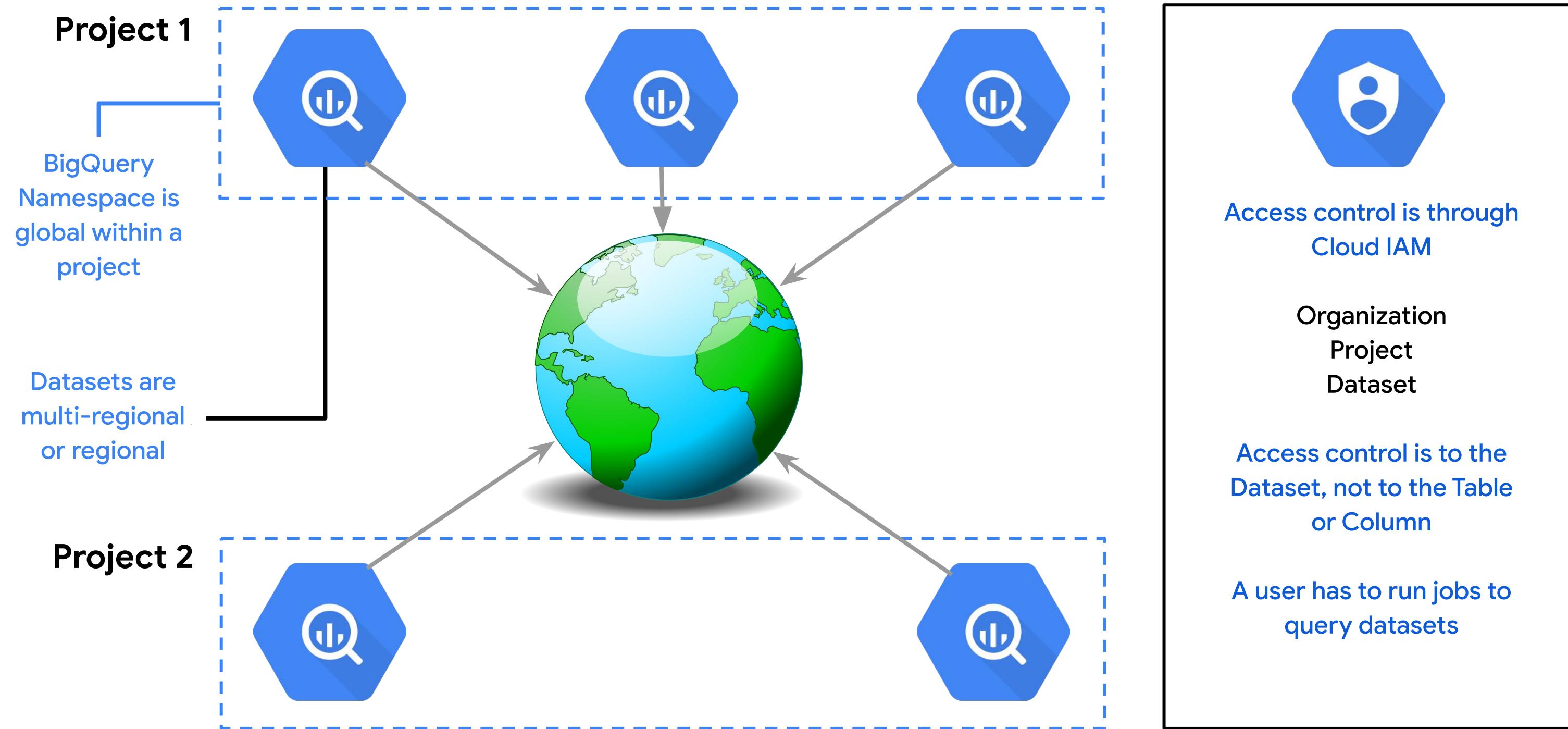
BigQuery datasets belong to a project



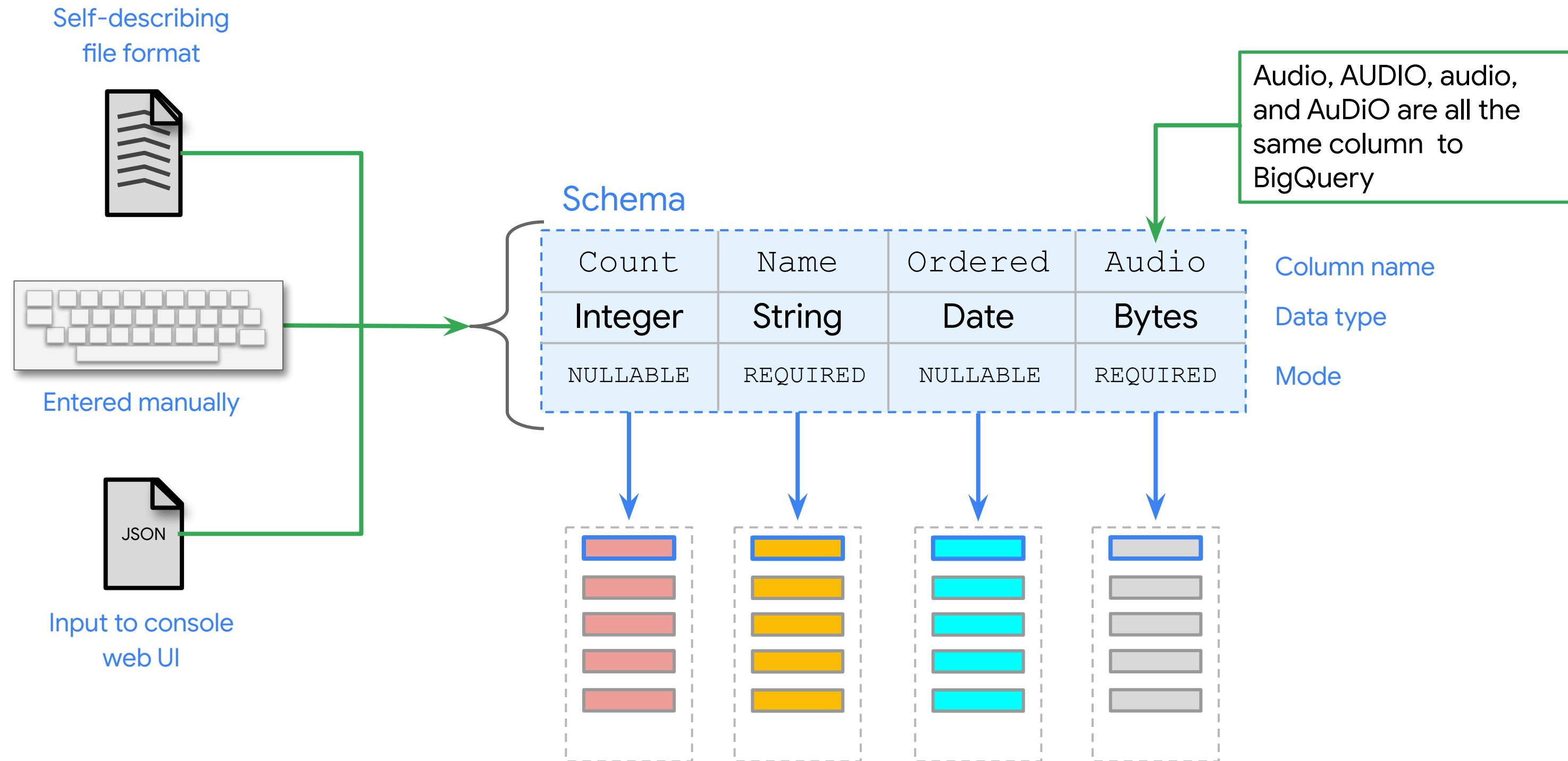
Access control to run a query is via Cloud IAM



BigQuery datasets can be regional or multi-regional



The table schema provides structure to the data



Security, encryption, and auditing for BigQuery

Key Management

Customer-managed
encryption keys



Cloud Audit Logs

Admin activity
System event
Data access

Authentication

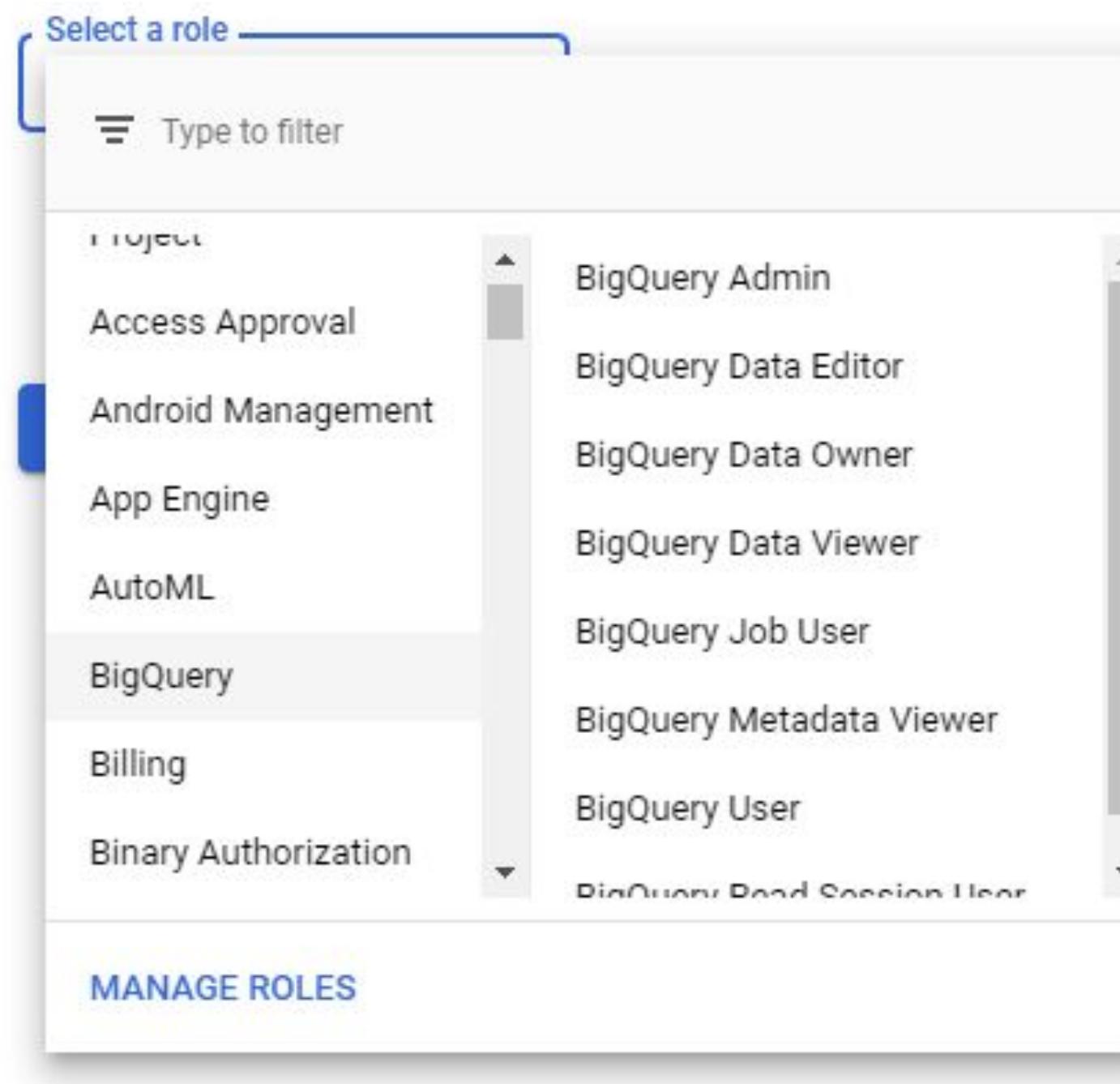
Service accounts
User accounts

Access Control

Cloud IAM roles
Permissions (custom)

Project and Dataset

BigQuery provides predefined roles for controlling access to resources



A screenshot of the Google Cloud IAM interface showing the 'Select a role' dropdown menu. The menu includes a search bar labeled 'Type to filter' and a list of predefined roles under the heading 'PROJECT'. The 'BigQuery' role is selected, highlighted with a blue background. Other visible roles include BigQuery Admin, BigQuery Data Editor, BigQuery Data Owner, BigQuery Data Viewer, BigQuery Job User, BigQuery Metadata Viewer, BigQuery User, and BigQuery Read Session User. At the bottom of the list is a 'BigQuery Read Session User' entry. A 'MANAGE ROLES' button is located at the bottom left of the menu.



Grants

Cloud IAM grants permission to perform specific actions

IAM applies to datasets

Use authorized views to restrict at a finer-grained level

It's easy to share access to datasets with other analysts

The screenshot shows the Google Cloud Platform BigQuery interface. The top navigation bar includes the Google Cloud Platform logo, the project ID "qwiklabs-gcp-c710de4945fa7b9c", and various status icons. The main menu on the left lists "BigQuery", "FEATURES & INFO", "SHORTCUTS", and "+ COMPOSE NEW QUERY". The "Query history", "Saved queries", "Job history", "Transfers", and "Scheduled queries" sections are visible on the left sidebar. The "BI Engine" section is also present. The "Resources" section shows a dataset named "qwiklabs-gcp-c710de4945fa7b9c" which contains a table named "champions_workshop_models". A search bar at the bottom of the sidebar allows searching for tables and datasets. The central "Query editor" pane displays a complex SQL query:

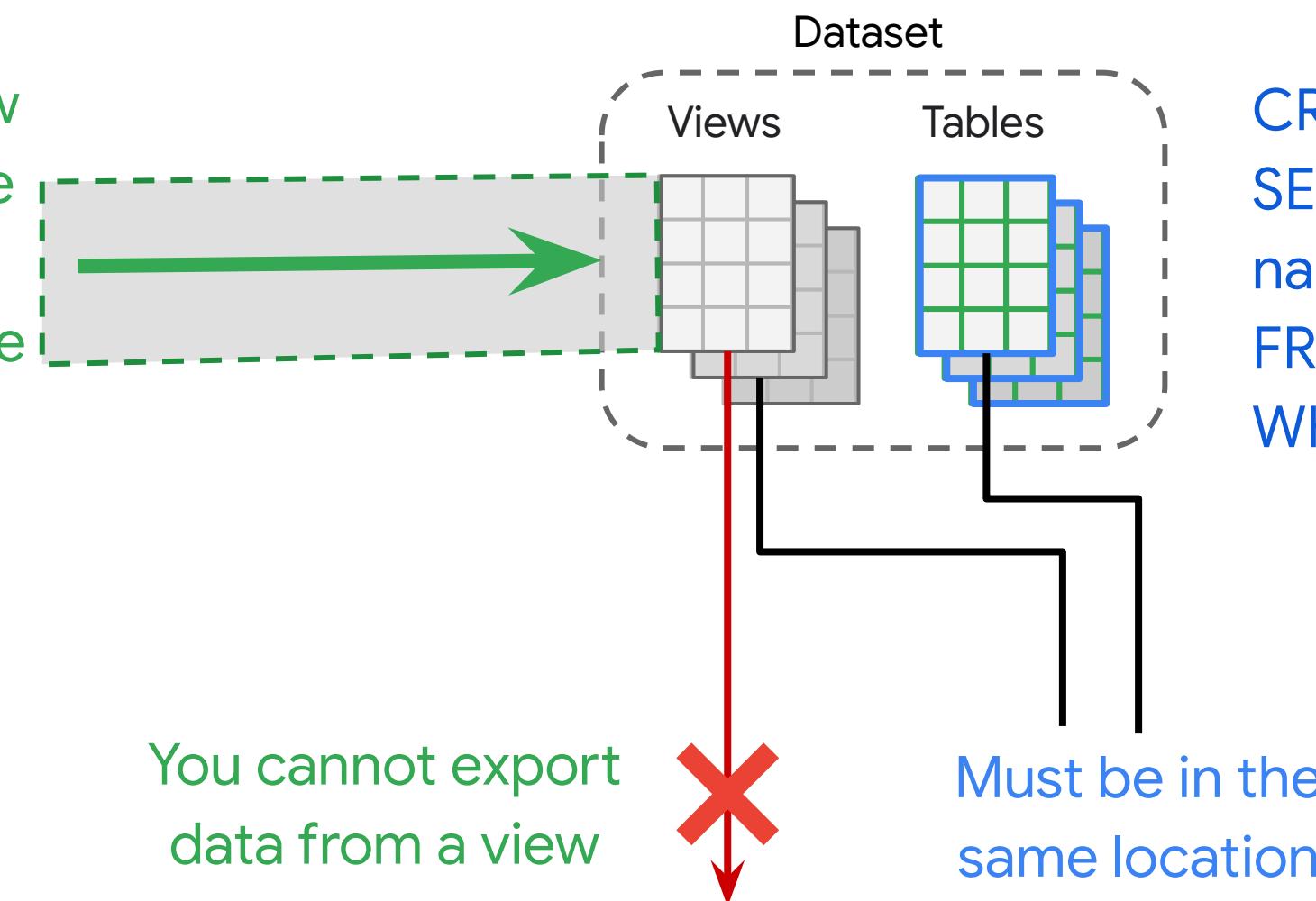
```
1 SELECT
2   MAX (visitNumber) AS visitNumber, #Total Number of User Visits
3   COUNT (*) AS totalHits, #Total Number of Hits (website interactions)
4   MAX (totals.timeOnSite) AS maxTime, #Longest visit
5   #Flag for whether user was on desktop or not
6   MAX (CASE
7     WHEN device.deviceCategory = 'desktop' THEN 1 ELSE 0 END) AS deviceDesktop,
8   #Root session is doomed by GA4GA's session quality dimension
```

Below the query editor are buttons for "Run", "Save query", "Save view", "Schedule query", and "More". A note indicates "This query will process 2.7 MB when run." The dataset details pane shows the dataset name "qwiklabs-gcp-c710de4945fa7b9c", a "SHARE DATASET" button (which is highlighted with a red box), and a "DELETE DATASET" button.

Views add another degree of access control

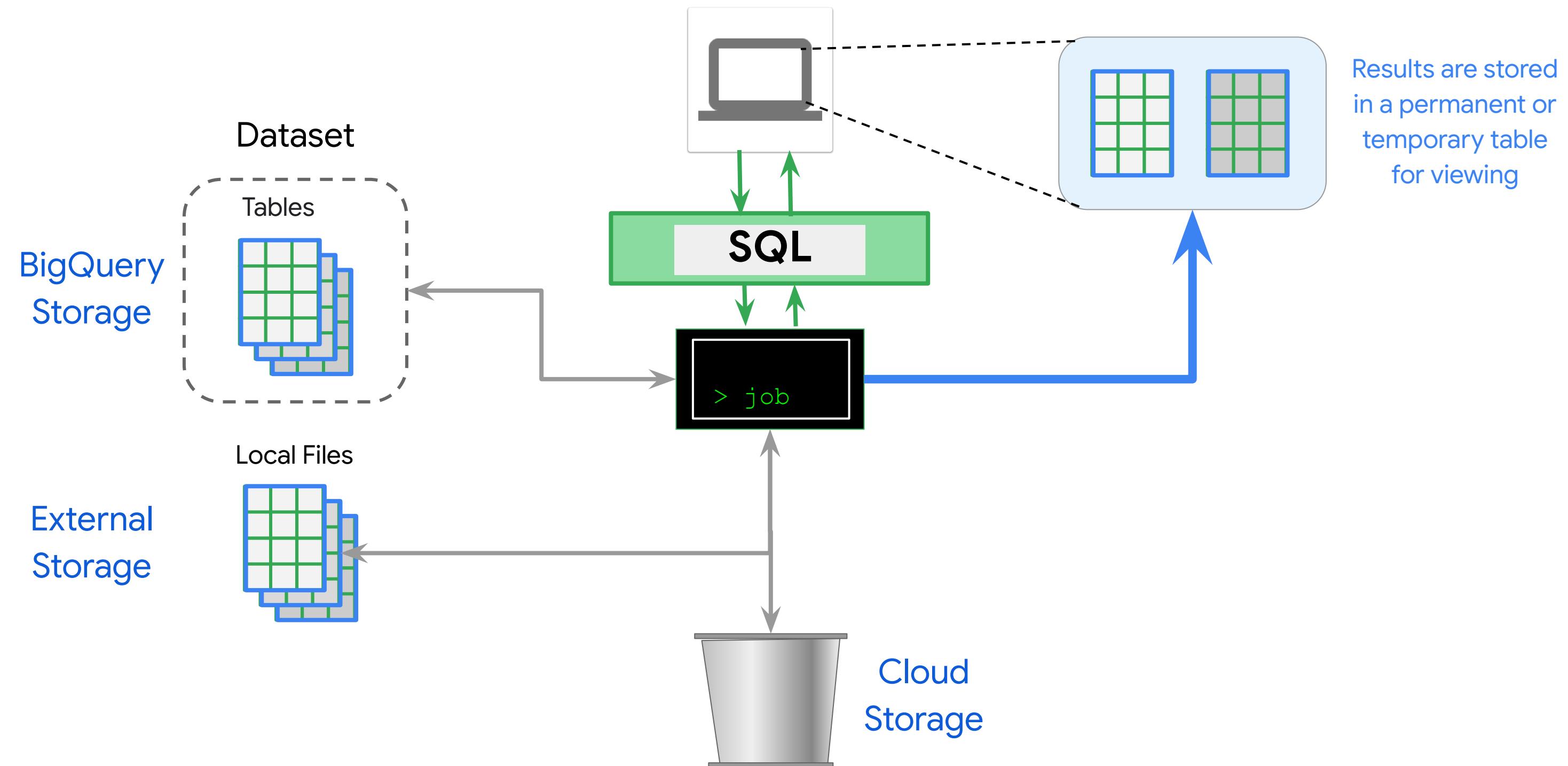
A view is a virtual table
defined by a SQL query

An authorized view
allows you to share
data externally
without sharing the
underlying table



CREATE VIEW dsB.myview AS
SELECT
name, number
FROM dsB.mytable
WHERE year >1950

The life of a BigQuery SQL query



Use query validator with pricing calculator for estimates

The screenshot shows the Google Cloud Platform interface with two main components highlighted:

- Query Validator (Top Bar):** A green box highlights the "Valid." message and the "Run" button. Red boxes labeled "2" and "1" highlight the "193.1 MB" data estimate and the green checkmark icon respectively.
- Pricing Calculator (Main Area):** A blue arrow points from the "Flood Zone Data" section to the "Query Pricing" section. Red boxes labeled "3" highlight the "2" queries input field. The calculator shows a total estimated cost of \$30.72 per month.

Google Cloud Platform Pricing Calculator

BigQuery

Flood Zone Data

Storage 1,536 GB

Queries 0.002 TB

\$30.72

Total Estimated Cost: \$30.72 per 1 month

Adjust Estimate Timeframe

1 day 1 week 1 month 1 quarter 1 year 3 years

Query Pricing

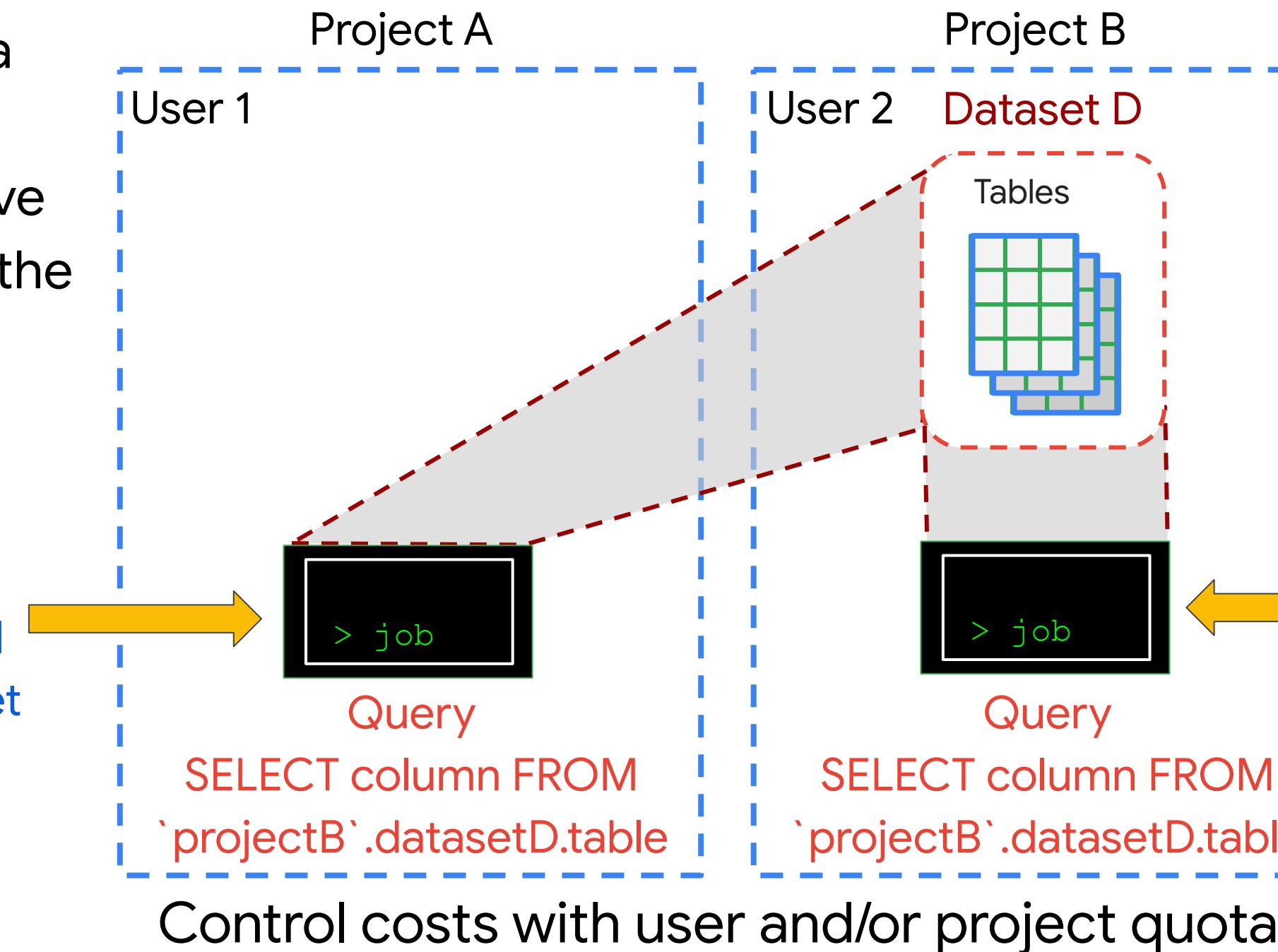
Queries 2

EMAIL ESTIMATE **SAVE ESTIMATE**

You can separate cost of storage and cost of queries

NOTE: The cost of a query is always assigned to the active project from where the query is executed

Project A pays for User 1's queries against data stored in Project B, Dataset D (assuming data access is shared)



Project B pays for storage and queries by User 2 executed from Project B.

With the BigQuery Data Transfer Service, you can copy large datasets from different projects to yours in seconds

The screenshot shows the Google Cloud Platform BigQuery interface. On the left, there's a sidebar with links like 'Query history', 'Saved queries', 'Job history', 'Transfers', 'Scheduled queries', and 'BI Engine'. Below that is a 'Resources' section with a search bar and a list of datasets from various projects. A specific dataset, 'dw-workshop:tpcds_1g_baseline', is selected. At the top right of the main area, there are buttons for 'Run', 'Save query', 'Save view', 'Schedule query', 'More', 'CREATE TABLE', 'SHARE DATASET', 'COPY DATASET' (which is highlighted with a red box), and 'DELETE DATASET'. The 'COPY DATASET' button has a blue icon of two overlapping boxes.

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
 - Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYS and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

The method you use to load data depends on how much transformation is needed



If the data is usable in its original form, just load it



BigQuery Data
Transfer Service (DTS)

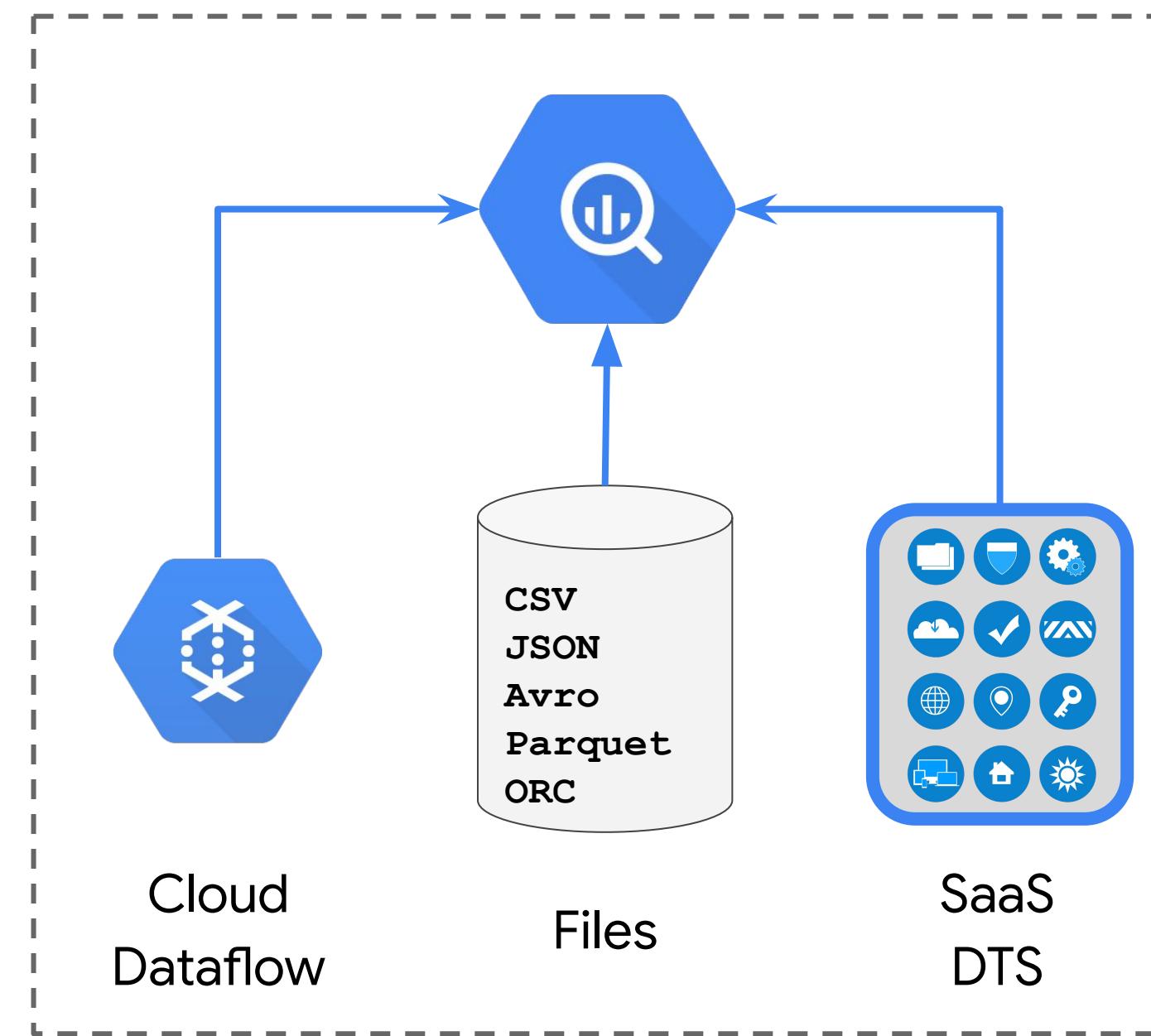


BigQuery

Batch load supports different file formats

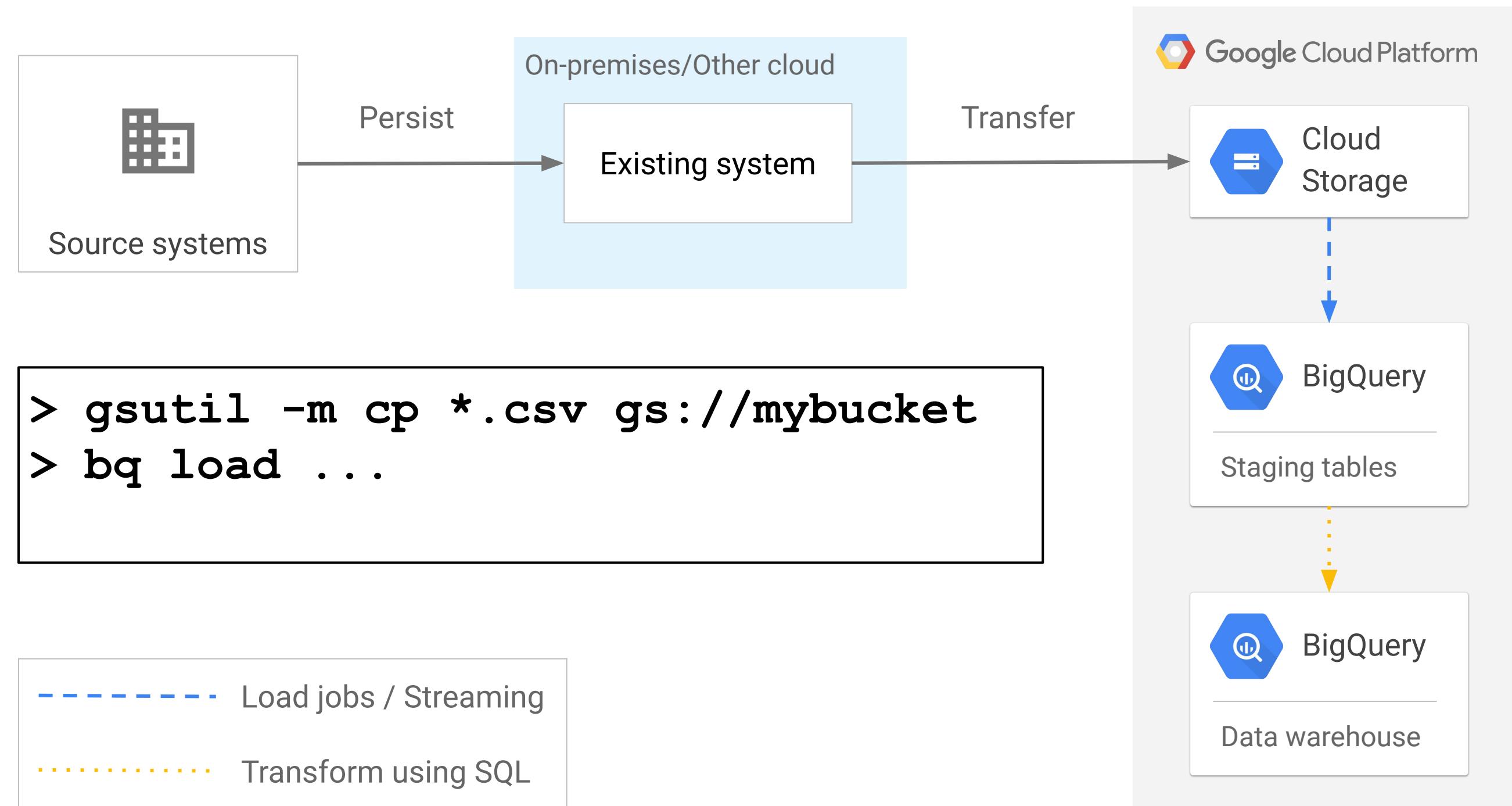
- CSV
- NEWLINE_DELIMITED_JSON
- AVRO
- DATASTORE_BACKUP
- PARQUET
- ORC

Most common is loading data into BigQuery tables (batch, periodic)



Loading data into BigQuery tables (batch, periodic) offers the best performance

Loading data through Cloud Storage



Automate the execution of queries based on a schedule

New scheduled query

Details and schedule

Name for scheduled query

daily_schedule

Schedule options

Repeats

Daily

Start now Schedule start time

End never Schedule end time

⚠ This schedule will run Every day at 17:46 Europe/London

Destination for query results

i A destination table is required to save scheduled query options.

Project name: `qwiklabs-gcp-c710de4945fa7b9c`

Dataset name: `champions_workshop_models`

Table name: `scheduled_query_output`

Destination table write preference

Append to table

Overwrite table

Notification options

Send email notifications

Schedule Cancel

BigQuery addresses backup and disaster recovery at the service level (time travel)

```
CREATE OR REPLACE TABLE ch10eu.restored_cycle_stations AS  
SELECT  
*  
FROM `bigquery-public-data`.london_bicycles.cycle_stations  
FOR SYSTEM_TIME AS OF  
TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 24 HOUR)
```

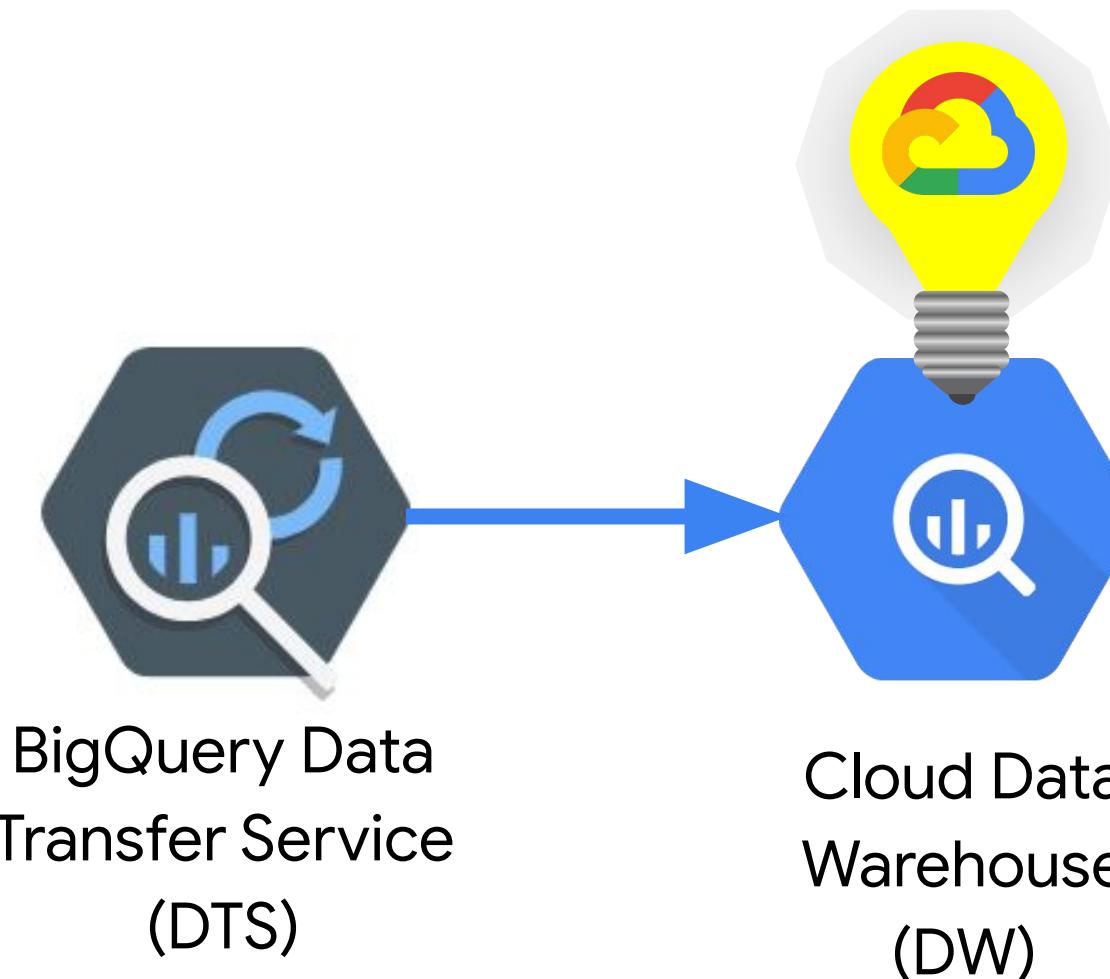


Query a point-in-time snapshot (up to 7 days)
Use it to create a backup

```
NOW=$(date +%s)  
SNAPSHOT=$(echo "($NOW - 120)*1000" | bc)  
bq --location=EU cp \  
ch10eu.restored_cycle_stations@$SNAPSHOT \  
ch10eu.restored_table
```

Restore a 2-min old copy
(Deleted tables are flushed after 2 days or if recreated with same name)

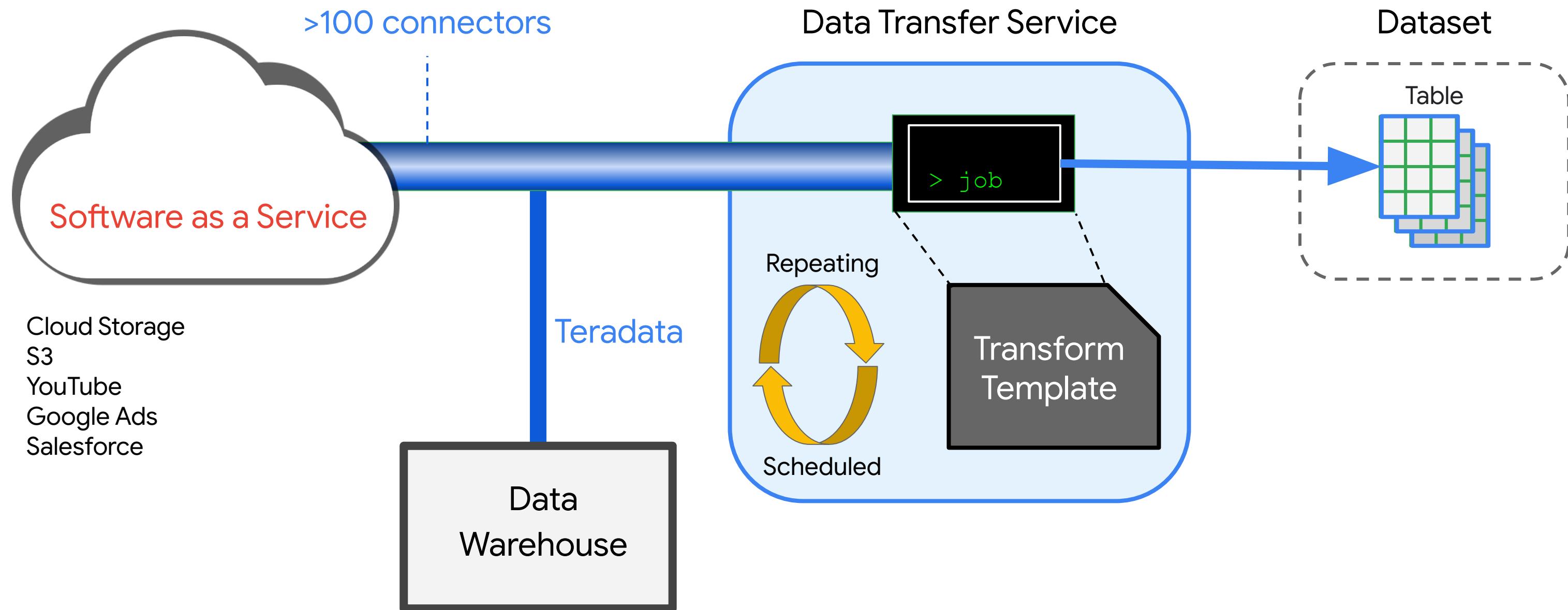
BigQuery Data Transfer Service helps you build and manage your data warehouse



EL

- Managed service
- Automatic transfers
- Scheduled
- Data staging
- Data processing
- Data backfills

Data Transfer Service provides SaaS connectors



BigQuery DTS supports 100+ SaaS applications

The screenshot shows the 'Create transfer' page in the BigQuery Data Transfer Service (DTS) interface. The top navigation bar includes the BigQuery logo and a 'BETA' badge, along with a back arrow and the text 'Create transfer'.

Source type
Choose a data source from the list below

Source * Type to filter

- Amazon S3
- Campaign Manager (formerly DCM)
- Google Ad Manager (formerly DFP)
- Google Ads (formerly AdWords)
- Google Cloud Storage
- Google Play
- Migration: Teradata
- YouTube Channel

EXPLORE DATA SOURCES

Marketplace > "Kafka"

Data sources

Filter by CATEGORY

4 results

Connector	Name	Description
	Heroku Kafka Connector by Fivetran	Fivetran Effortlessly replicate all your Heroku Kafka data into BigQuery.
	Apache Kafka Connector by Fivetran	Fivetran Effortlessly replicate all your Apache Kafka data into BigQuery.

How DTS transfers work

You provide the dataset

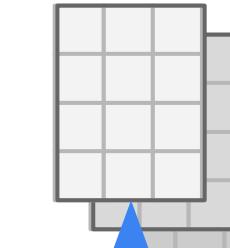
DTS runs BigQuery jobs that
create the Tables and Views



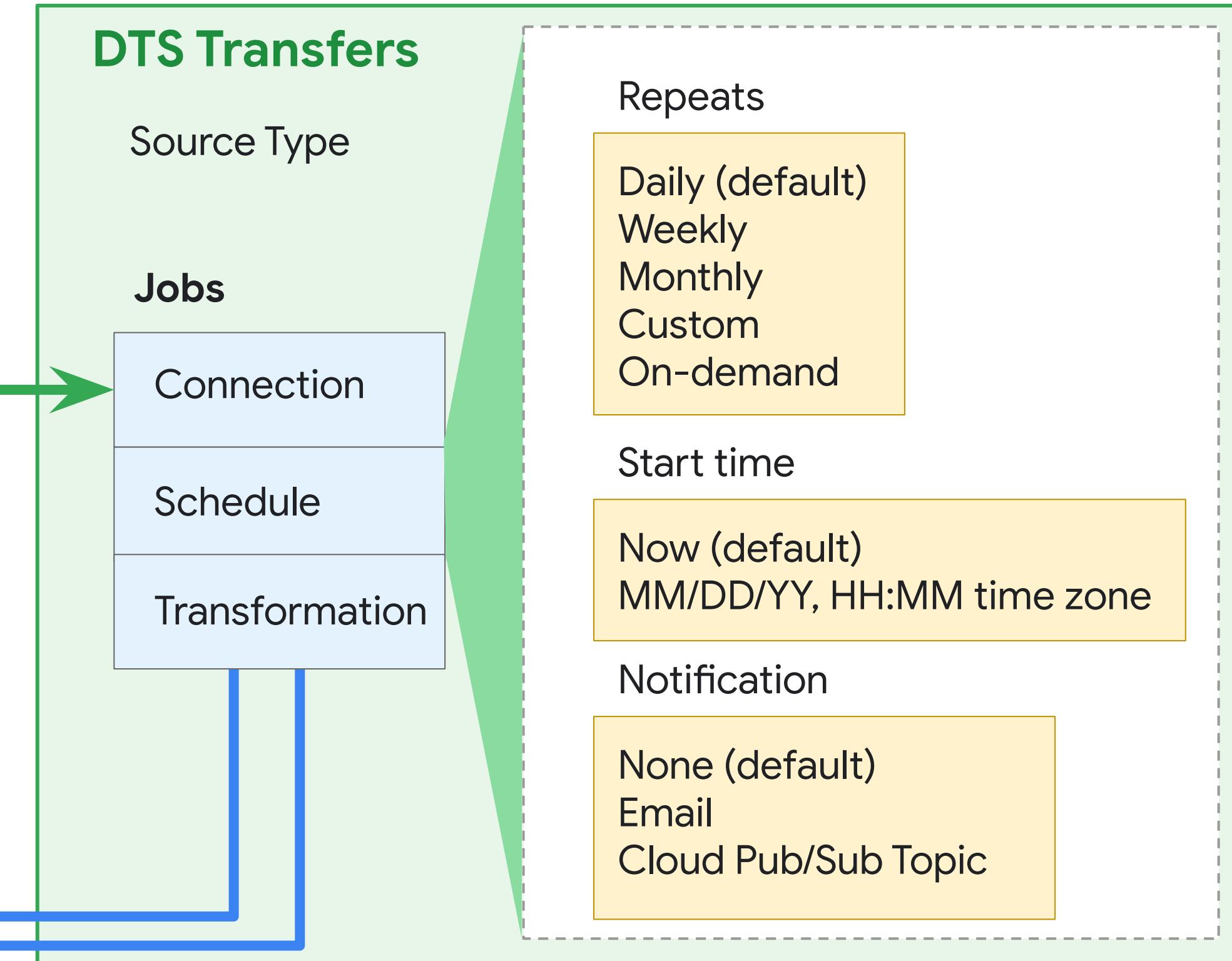
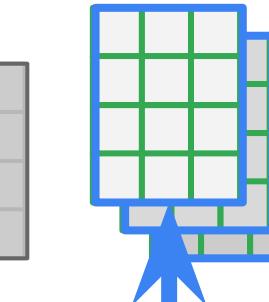
Pull

Datasets

Views



Tables



If the data requires simple transformations, such as scaling, maybe it can be handled in SQL



BigQuery



Modify table data with standard DML statements

INSERT, UPDATE, DELETE, MERGE records into tables

```
UPDATE table_A  
SET  
    y = table_B.y,  
    z = table_B.z + 1  
FROM table_B  
WHERE table_A.x = table_B.x  
    AND table_A.y IS NULL;
```

INSERT INTO table VALUES (1,2,3), (4,5,6), (7,8,9);

DELETE FROM table WHERE TRUE;

Create new tables from data with SQL DDL

```
SELECT
  *
FROM
  movielens.movies_raw
WHERE
  movieId < 5;
```

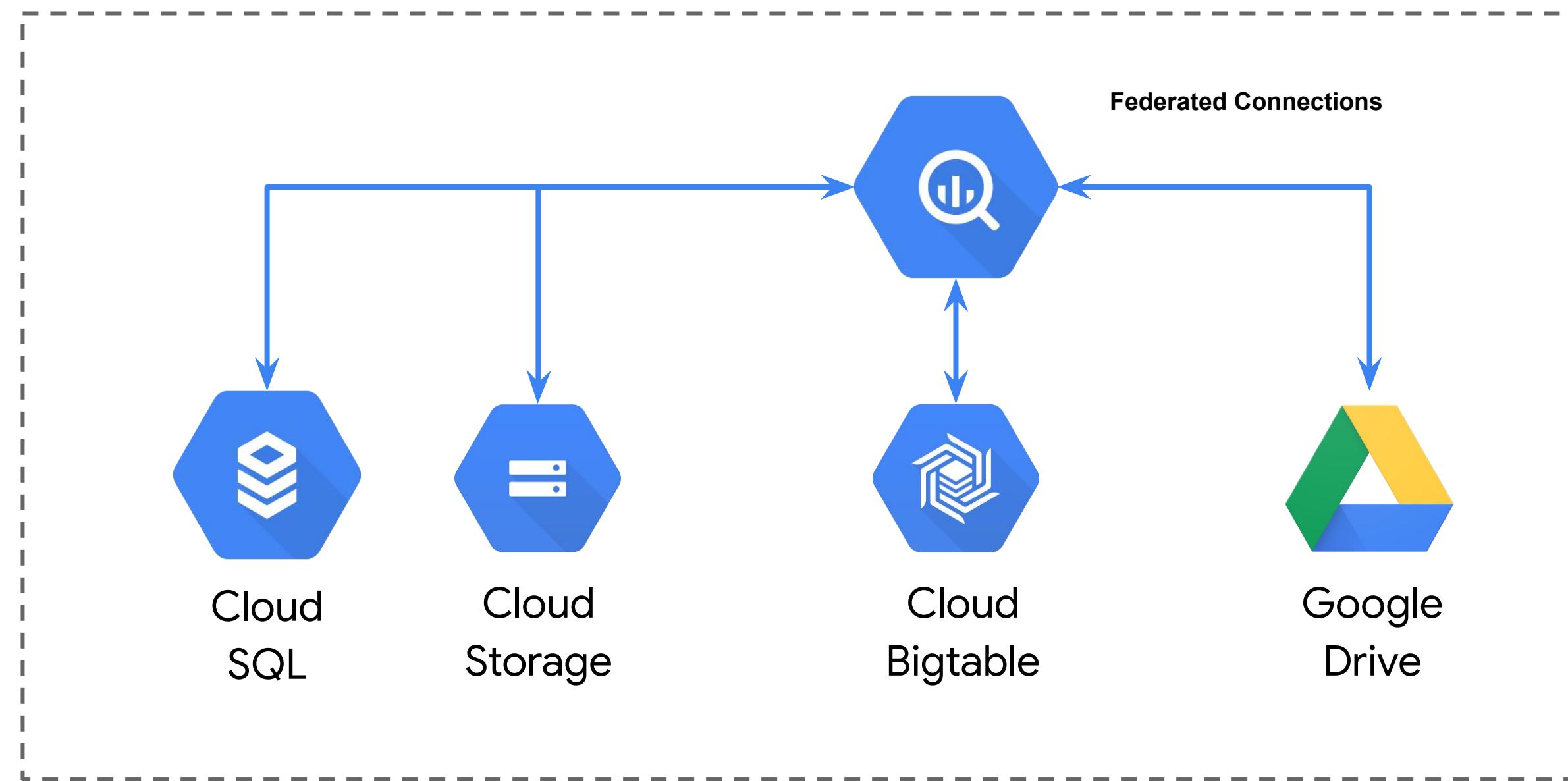
	movieId	title	genres
0	3	Grumpier Old Men (1995)	Comedy Romance
1	4	Waiting to Exhale (1995)	Comedy Drama Romance
2	2	Jumanji (1995)	Adventure Children Fantasy
3	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy

```
CREATE OR REPLACE TABLE
movielens.movies AS
SELECT
  * REPLACE(SPLIT(genres, "|")
AS genres)
FROM
  Movielens.movies_raw;
-- Execute multiple statements.
SELECT * FROM
  movielens.movies;
```

	movieId	title	genres
0	4	Waiting to Exhale (1995)	[Comedy, Drama, Romance]
1	3	Grumpier Old Men (1995)	[Comedy, Romance]
2	2	Jumanji (1995)	[Adventure, Children, Fantasy]
3	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]

Question: What's the difference between CREATE OR REPLACE TABLE and CREATE TABLE IF NOT EXISTS ? When would you use each?

Some data can be used in place without importing into BigQuery tables using external data sources



Video Demo: Querying Cloud SQL directly from BigQuery

The screenshot shows the Google Cloud BigQuery web interface. On the left, there's a sidebar with navigation links: 'BigQuery', 'Saved queries', 'Jobs history', 'Transactions', 'Scheduled queries', 'All views', 'Resources', and 'ADD DATA'. Below these are sections for 'Query Information' and 'External connections'. A specific connection named 'cloudsql-instances' is selected, highlighted with a blue background.

The main area is titled 'Unsaved query: Edit'. It contains a code editor with the following SQL query:

```
SELECT
    1 AS ID,
    1 AS table_id,
    1 AS column_id,
    1 AS offset_in_result,
    average_value,
    variable_current_value
FROM
    EXTERNAL_QUERY('bigquery:information_schema.information_schema')
WHERE
    table_name = 'COLUMNS' AND column_name = 'TABLE_NAME'
```

Below the code editor, there are tabs for 'Table', 'Query', 'Save view', 'Schedule query', and 'View'. The 'Query' tab is selected. The results section shows a table with the following data:

Row	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE	ENGINE	VERSION
1	inf	information_schema	CHARACTER_SETS	SYSTEM VIEW	MEMORY	10
2	inf	information_schema	COLLATIONS	SYSTEM VIEW	MEMORY	10
3	inf	information_schema	COLLATION_CHARACTER_SET_APPLICABILITY	SYSTEM VIEW	MEMORY	10
4	inf	information_schema	COLUMNS	SYSTEM VIEW	InnoDB	10
5	inf	information_schema	COLUMNS_PRIVILEGES	SYSTEM VIEW	MEMORY	10
6	inf	information_schema	TABLES	SYSTEM VIEW	MEMORY	10
7	inf	information_schema	EVENTS	SYSTEM VIEW	InnoDB	10
8	inf	information_schema	FILES	SYSTEM VIEW	MEMORY	10
9	inf	information_schema	GLOBAL_STATUS	SYSTEM VIEW	MEMORY	10

Custom transformations? BigQuery supports user-defined functions in SQL, JavaScript, and scripting

Query editor

```
1 CREATE TEMP FUNCTION multiplyInputs(x FLOAT64, y FLOAT64)
2 RETURNS FLOAT64
3 LANGUAGE js AS """
4   return x*y;
5 """
6 WITH numbers AS
7   (SELECT 1 AS x, 5 as y
8    UNION ALL
9    SELECT 2 AS x, 10 as y
10   UNION ALL
11   SELECT 3 as x, 15 as y)
12  SELECT x, y, multiplyInputs(x, y) as product
13 FROM numbers;
```

Query complete (2.5 sec elapsed, 0 B processed)

Job information **Results** JSON Execution details

Row	x	y	product
1	1	5	5.0
2	2	10	20.0
3	3	15	45.0

Run ▾ Save query Save view Schedule query ▾ More ▾

You can persist and share your UDF objects with other team members or publically

The screenshot shows a GitHub repository page for `GoogleCloudPlatform/bigquery-utils`. The repository has 2 issues and 5 pull requests. The current branch is `master`, which contains a single commit by `ryanmcdowell` adding a `random_value` function. Below the commit, there is a list of SQL files with their commit details:

File	Commit Description
<code>README.md</code>	Add random_value function (#9)
<code>int.sql</code>	Initial commit
<code>median.sql</code>	Initial commit
<code>multiply_full_scale.sql</code>	Add multiply_full_scale function.
<code>nlp_compromise_number.sql</code>	Initial commit
<code>nlp_compromise_people.sql</code>	Initial commit
<code>radians.sql</code>	Initial commit
<code>random_int.sql</code>	Fix project-id for random_int UDF

The BigQuery team has a public GitHub repo for common User Defined Functions

<https://github.com/GoogleCloudPlatform/bigquery-utils/tree/master/udfs/community>

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYS and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data



Lab

Loading data into BigQuery

Objectives

- Loading data into BigQuery from various sources
- Loading data into BigQuery using the CLI and Console
- Using DDL to create tables

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYS and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

Designing schemas that scale is a core job of data engineers -- let's explore BigQuery Public Dataset schemas



Public datasets include flights, taxi cab logs, weather recordings, and many more

<https://cloud.google.com/bigquery/public-data/>

Demo

Exploring BigQuery Public Datasets with SQL using **INFORMATION_SCHEMA**

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYS and STRUCTs
- Optimizing with Partitioning and Clustering

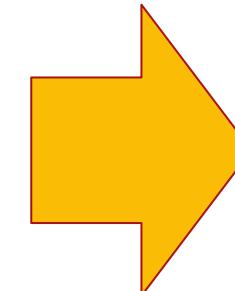
Preview: Transforming Batch and Streaming Data

Transactional databases often use normal form

Original data

Customer	OrderID	Date	Items	
			Product	Quantity
Doug	1600p	8/20/19	Caulk	3 boxes
			Soffit	34 meters
			Sealant	2 liters
Tom	221b	10/29/19	Sealant	1 liter
			Soffit	17 meters
			Caulk	4 tubes

Normalized data



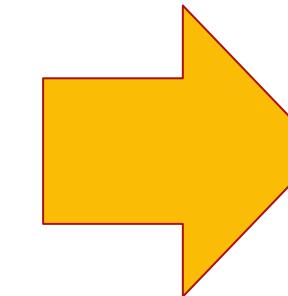
Orders		
Date	OrderID	
8/20/2019	1600p	
10/29/2018	221b	

Order_Items		
OrderID	Product	Quantity
1600p	Caulk	3 boxes
221b	Sealant	1 liter
1600p	Soffit	34 meters
221b	Soffit	17 meters
221b	Caulk	4 tubes
1600p	Sealant	2 liters

Data warehouses often denormalize

Normalized data

Orders	
Date	OrderID
08/20/2019	1600p
10/29/2018	221b



Order_Items		
OrderID	Product	Quantity
1600p	Caulk	3 boxes
221b	Sealant	1 liter
1600p	Soffit	34 meters
221b	Soffit	17 meters
221b	Caulk	4 tubes
1600p	Sealant	2 liters

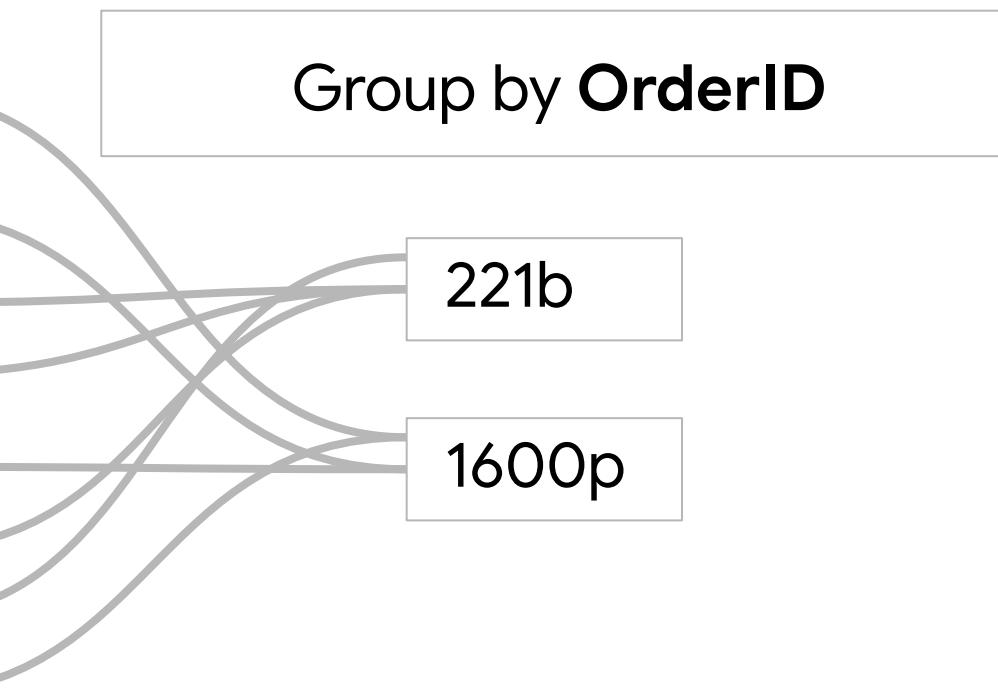
Denormalized flattened data

Customer	OrderID	Date	Product	Quantity
Doug	1600p	08/20/2019	Siding	3 boxes
Doug	1600p	08/20/2019	Caulk	12 tubes
Tom	221b	10/29/2019	Soffit	17 meters
Tom	221b	10/29/2019	Sealant	1 liter
Doug	1600p	08/20/2019	Soffit	34 meters
Tom	221b	10/29/2019	Siding	2 boxes
Tom	221b	10/29/2019	Caulk	4 tubes
Doug	1600p	08/20/2019	Sealant	2 liters

Grouping on a 1-to-many field in flattened data can cause shuffling of data over the network

Denormalized flattened table

Customer	OrderID	Date	Product	Quantity
Doug	1600p	08/20/2019	Siding	3 boxes
Doug	1600p	08/20/2019	Caulk	12 tubes
Tom	221b	10/29/2019	Soffit	17 meters
Tom	221b	10/29/2019	Sealant	1 liter
Doug	1600p	09/09/2018	Soffit	34 meters
Tom	221b	10/29/2019	Siding	2 boxes
Tom	221b	10/29/2019	Caulk	4 tubes
Doug	1600p	08/20/2018	Sealant	2 liters



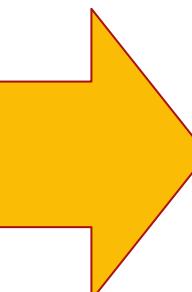
Nested and repeated columns improve the efficiency of BigQuery with relational source data

Denormalized flattened table

Customer	OrderID	Date	Product	Quantity
Doug	1600p	08/20/2019	Siding	3 boxes
Doug	1600p	08/20/2019	Caulk	12 tubes
Tom	221b	10/29/2019	Soffit	17 meters
Tom	221b	10/29/2019	Sealant	1 liter
Doug	1600p	8/20/2019	Soffit	34 meters
Tom	221b	10/29/2019	Siding	2 boxes
Tom	221b	10/29/2019	Caulk	4 tubes
Doug	1600p	08/20/2019	Sealant	2 liters

Denormalized with nested and repeated data

Order.ID	Order.Date	Order.Product	Order.Quantity
1600p	08/20/2019	Siding	3 boxes
		Caulk	12 tubes
		Soffit	34 meters
		Sealant	2 liters
221b	10/29/2019	Soffit	17 meters
		Sealant	1 liter
		Siding	2 boxes
		Caulk	4 tubes



Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYS and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data



GO-JEK is a ride booking service in Indonesia running on GCP

GO-JEK has 13+PB of data queried each month

Orders

Events

Pick-ups

Drop-offs

- Each ride is stored as an order
- Each ride has a **single** pickup and drop-off
- Each ride can have **one-to-many** events:
 - Ride confirmed
 - Driver en route
 - Pick-up
 - Drop-off

How do you structure your data warehouse for scale?
Four separate and large tables that we join together?

Reporting Approach: Should we normalize or denormalize?

Orders

Events

Pick-ups

Drop-offs

VS

Orders

Many tables

One big table

Reporting Approach: Should we normalize or denormalize?

Orders

Events

Pick-ups

Drop-offs

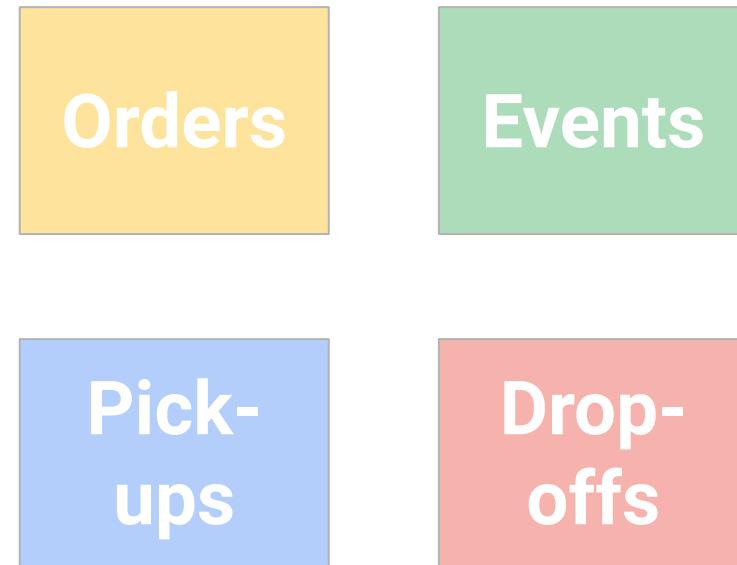
VS

Orders

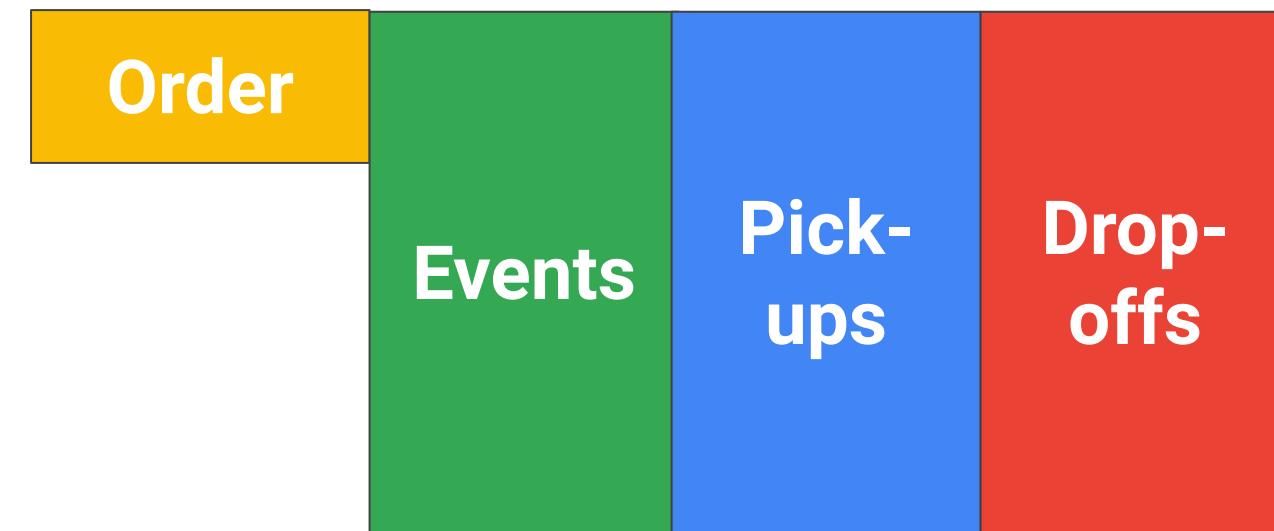
JOINs are costly

Data is repeated

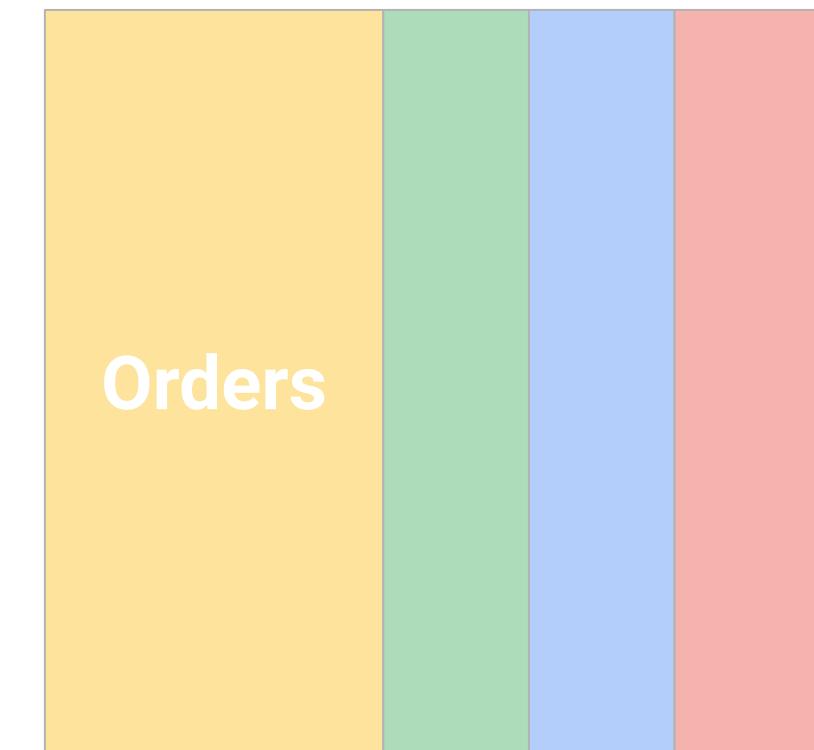
Nested and Repeated Fields allow you to have multiple levels of data granularity



JOINS are costly



Data is nested
and repeated



Data is repeated

Store complex data with nested fields (ARRAYS)

Row	order_id	service_type	payment_method	event.status	event.time	pickup.latitude	pickup.longitude	destination.latitude	destination.longitude	total_distance_km	pricing_type
151	FD-5117	GO_FOOD	GOPAY	CREATED	2018-12-31 04:44:02.545210 UTC	-7.75105	110.410561	-7.7430367	110.4046433	1.56	regular
				COMPLETED	2018-12-31 05:06:27.897769 UTC						
				PICKED_UP	2018-12-31 04:48:25.945331 UTC						
				DRIVER_FOUND	2018-12-31 04:44:06.869010 UTC						
152	FD-6834	GO_FOOD	CASH	PICKED_UP	2018-12-31 12:49:52.518880 UTC	1.121272	104.049739	1.1368655	104.03322	4.84	surge
				DRIVER_FOUND	2018-12-31 12:40:14.214843 UTC						
				COMPLETED	2018-12-31 13:04:00.291780 UTC						
				CREATED	2018-12-31 12:40:13.431094 UTC						
153	FD-6293	GO_FOOD	PARTIAL_PAYMENT	PICKED_UP	2018-12-31 04:33:11.856445 UTC	-7.9657554	112.6247491	-7.9384084	112.6227862	4.68	regular
				COMPLETED	2018-12-31 04:56:05.885521 UTC						
				CREATED	2018-12-31 04:16:24.356539 UTC						
				DRIVER_FOUND	2018-12-31 04:16:25.643766 UTC						
154	FD-7817	GO_FOOD	CASH	COMPLETED	2018-12-31 09:14:44.897136 UTC	-6.353915	106.247312	-6.368896	106.25787	3.51	regular
				PICKED_UP	2018-12-31 09:01:11.471274 UTC						
				CREATED	2018-12-31 08:40:31.821796 UTC						
				DRIVER_FOUND	2018-12-31 08:40:32.910319 UTC						

[Table](#) [JSON](#)

[First](#) [< Prev](#) Rows 151 - 154 of 1137 [Next >](#) [Last](#)

Report on all data in once place with STRUCTS

Row	order_id	service_type	payment_method	event.status	event.time	pickup.latitude	pickup.longitude	destination.latitude	destination.longitude	total_distance_km	pricing_type
151	FD-5117	GO_FOOD	GOPAY	CREATED	2018-12-31 04:44:02.545210 UTC	-7.75105	110.410561	-7.7430367	110.4046433	1.56	regular
				COMPLETED	2018-12-31 05:06:27.897769 UTC						
				PICKED_UP	2018-12-31 04:48:25.945331 UTC						
				DRIVER_FOUND	2018-12-31 04:44:06.869010 UTC						
152	FD-6834	GO_FOOD	CASH	PICKED_UP	2018-12-31 12:49:52.518880 UTC	1.121272	104.049739	1.1368655	104.03322	4.84	surge
				DRIVER_FOUND	2018-12-31 12:40:14.214843 UTC						
				COMPLETED	2018-12-31 13:04:00.291780 UTC						
				CREATED	2018-12-31 12:40:13.431094 UTC						
153	FD-6293	GO_FOOD	PARTIAL_PAYMENT	PICKED_UP	2018-12-31 04:33:11.856445 UTC	-7.9657554	112.6247491	-7.9384084	112.6227862	4.68	regular
				COMPLETED	2018-12-31 04:56:05.885521 UTC						
				CREATED	2018-12-31 04:16:24.356539 UTC						
				DRIVER_FOUND	2018-12-31 04:16:25.643766 UTC						
154	FD-7817	GO_FOOD	CASH	COMPLETED	2018-12-31 09:14:44.897136 UTC	-6.353915	106.247312	-6.368896	106.25787	3.51	regular
				PICKED_UP	2018-12-31 09:01:11.471274 UTC						
				CREATED	2018-12-31 08:40:31.821796 UTC						
				DRIVER_FOUND	2018-12-31 08:40:32.910319 UTC						

[Table](#) [JSON](#)

[First](#) [< Prev](#) Rows 151 - 154 of 1137 [Next >](#) [Last](#)

Nested ARRAY fields and STRUCT fields allow for differing data granularity in the same table

Row	order_id	service_type	payment_method	event.status	event.time	pickup.latitude	pickup.longitude	destination.latitude	destination.longitude	total_distance_km	pricing_type
151	FD-5117	GO_FOOD	GOPAY	CREATED	2018-12-31 04:44:02.545210 UTC	-7.75105	110.410561	-7.7430367	110.4046433	1.56	regular
				COMPLETED	2018-12-31 05:06:27.897769 UTC						
				PICKED_UP	2018-12-31 04:48:25.945331 UTC						
				DRIVER_FOUND	2018-12-31 04:44:06.869010 UTC						
152	FD-6834	GO_FOOD	CASH	PICKED_UP	2018-12-31 12:49:52.518880 UTC	1.121272	104.049739	1.1368655	104.03322	4.84	surge
				DRIVER_FOUND	2018-12-31 12:40:14.214843 UTC						
				COMPLETED	2018-12-31 13:04:00.291780 UTC						
				CREATED	2018-12-31 12:40:13.431094 UTC						
153	FD-6293	GO_FOOD	PARTIAL_PAYMENT	PICKED_UP	2018-12-31 04:33:11.856445 UTC	-7.9657554	112.6247491	-7.9384084	112.6227862	4.68	regular
				COMPLETED	2018-12-31 04:56:05.885521 UTC						
				CREATED	2018-12-31 04:16:24.356539 UTC						
				DRIVER_FOUND	2018-12-31 04:16:25.643766 UTC						
154	FD-7817	GO_FOOD	CASH	COMPLETED	2018-12-31 09:14:44.897136 UTC	-6.353915	106.247312	-6.368896	106.25787	3.51	regular
				PICKED_UP	2018-12-31 09:01:11.471274 UTC						
				CREATED	2018-12-31 08:40:31.821796 UTC						
				DRIVER_FOUND	2018-12-31 08:40:32.910319 UTC						

Table JSON

First < Prev Rows 151 - 154 of 1137 Next > Last

Your turn

- Practice reading the new schema
- Spot the STRUCTS
- Type RECORD = STRUCTS

booking

Schema Details Preview

Field name	Type	Mode
order_id	STRING	NULLABLE
service_type	STRING	NULLABLE
payment_method	STRING	NULLABLE
event	RECORD	REPEATED
event.status	STRING	NULLABLE
event.time	TIMESTAMP	NULLABLE
pickup	RECORD	NULLABLE
pickup.latitude	FLOAT	NULLABLE
pickup.longitude	FLOAT	NULLABLE
destination	RECORD	NULLABLE
destination.latitude	FLOAT	NULLABLE
destination.longitude	FLOAT	NULLABLE
total_distance_km	FLOAT	NULLABLE
pricing_type	STRING	NULLABLE
duration	RECORD	NULLABLE
duration.booking_to_dispatch	FLOAT	NULLABLE
duration.booking_to_pickup	FLOAT	NULLABLE

Practice reading the new schema

- Practice reading the new schema
- Spot the STRUCTS
- Type RECORD = STRUCTS

booking		
Schema	Details	Preview
Field name	Type	Mode
order_id	STRING	NULLABLE
service_type	STRING	NULLABLE
payment_method	STRING	NULLABLE
event	RECORD	REPEATED
event.status	STRING	NULLABLE
event.time	TIMESTAMP	NULLABLE
pickup	RECORD	NULLABLE
pickup.latitude	FLOAT	NULLABLE
pickup.longitude	FLOAT	NULLABLE
destination	RECORD	NULLABLE
destination.latitude	FLOAT	NULLABLE
destination.longitude	FLOAT	NULLABLE
total_distance_km	FLOAT	NULLABLE
pricing_type	STRING	NULLABLE
duration	RECORD	NULLABLE
duration.booking_to_dispatch	FLOAT	NULLABLE
duration.booking_to_pickup	FLOAT	NULLABLE

Events

Pick-ups

Destination

Duration

Your turn

- Practice reading the new schema
- Spot the ARRAYS
- Hint: Look at Mode

booking

Schema	Details	Preview
Field name	Type	Mode
order_id	STRING	NULLABLE
service_type	STRING	NULLABLE
payment_method	STRING	NULLABLE
event	RECORD	REPEATED
event. status	STRING	NULLABLE
event. time	TIMESTAMP	NULLABLE
pickup	RECORD	NULLABLE

Practice reading the new schema

- Practice reading the new schema
- Spot the ARRAYS
- REPEATED = ARRAY

booking

Schema Details Preview

Field name	Type	Mode
order_id	STRING	NULLABLE
service_type	STRING	NULLABLE
payment_method	STRING	NULLABLE
event	RECORD	REPEATED
event.status	STRING	NULLABLE
event.time	TIMESTAMP	NULLABLE

Events

pickup RECORD NULLABLE

Row	order_id	service_type	payment_method	event.status	event.time
151	FD-5117	GO_FOOD	GOPAY	CREATED	2018-12-31 04:44:02.545210 UTC
				COMPLETED	2018-12-31 05:06:27.897769 UTC

Status and Time in an ARRAY of Event STRUCTs

Recap

- STRUCTS (RECORD)
- ARRAYS (REPEATED)
- ARRAYS can be part of regular fields or STRUCTS
- A single table can have many STRUCTS

booking

Schema

Details

Preview

Field name	Type	Mode
order_id	STRING	NULLABLE
service_type	STRING	NULLABLE
payment_method	STRING	NULLABLE
event	RECORD	REPEATED
event.status	STRING	NULLABLE
event.time	TIMESTAMP	NULLABLE
pickup	RECORD	NULLABLE
pickup.latitude	FLOAT	NULLABLE
pickup.longitude	FLOAT	NULLABLE
destination	RECORD	NULLABLE
destination.latitude	FLOAT	NULLABLE
destination.longitude	FLOAT	NULLABLE
total_distance_km	FLOAT	NULLABLE
pricing_type	STRING	NULLABLE
duration	RECORD	NULLABLE
duration.booking_to_dispatch	FLOAT	NULLABLE
duration.booking_to_pickup	FLOAT	NULLABLE

BigQuery stores repeated, nested fields in a columnar format

DocId: 10	r₁
Links	
Forward: 20	
Forward: 40	
Forward: 60	
Name	
Language	
Code: 'en-us'	
Country: 'us'	
Language	
Code: 'en'	
Url: 'http://A'	
Name	
Url: 'http://B'	
Name	
Language	
Code: 'en-gb'	
Country: 'gb'	

```
message Document {  
  required int64 DocId;  
  optional group Links {  
    repeated int64 Backward;  
    repeated int64 Forward; }  
  repeated group Name {  
    repeated group Language {  
      required string Code;  
      optional string Country; }  
    optional string Url; }}
```

DocId: 20	r₂
Links	
Backward: 10	
Backward: 30	
Forward: 80	
Name	
Url: 'http://C'	

DocID			Name.Url			Links.Forward			Links.Backward		
value	r	d	value	r	d	value	r	d	value	r	d
10	0	0	http://A	0	2	20	0	2	NULL	0	1
20	0	0	http://B	1	2	40	1	2	10	0	2
			NULL	1	1	60	1	2	30	1	2
			http://C	0	2	80	0	2			

Name.Language.Code			Name.Language.Country		
value	r	d	value	r	d
en-us	0	2	us	0	3
en	2	2	NULL	2	2
NULL	1	1	NULL	1	1
en-gb	1	2	gb	1	3
NULL	0	1	NULL	0	1

Dremel

Demo

Nested and repeated fields in
BigQuery

General guidelines to design the optimal schema for BigQuery



Instead of joins, take advantage of nested and repeated fields in denormalized tables.

General guidelines to design the optimal schema for BigQuery



Instead of joins, take advantage of nested and repeated fields in denormalized tables.



Keep a dimension table smaller than 10 gigabytes normalized, unless the table rarely goes through UPDATE and DELETE operations.

General guidelines to design the optimal schema for BigQuery



Instead of joins, take advantage of nested and repeated fields in denormalized tables.



Keep a dimension table smaller than 10 gigabytes normalized, unless the table rarely goes through UPDATE and DELETE operations.



Denormalize a dimension table larger than 10 gigabytes, unless data manipulation or costs outweigh benefits of optimal queries.

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data



Lab

Working with JSON and Array Data in BigQuery

Objectives

- Load semi-structured JSON into BigQuery
- Create and query arrays
- Create and query structs
- Query nested and repeated fields

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYS and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

Reduce cost and amount of data read by partitioning your tables

c1	c2	c3	eventDate	c5
			2019-01-01	
			2019-01-02	
			2019-01-03	
			2019-01-04	
			2019-01-05	

Partitioned tables

```
SELECT c1, c3 FROM ...
WHERE eventDate BETWEEN
"2019-01-03" AND "2019-01-04"
```

BigQuery supports three ways of partitioning tables

Ingestion time

```
bq query --destination_table mydataset.mytable  
--time_partitioning_type=DAY  
...
```

Any column that is of type
DATE or TIMESTAMP

```
bq mk --table --schema a:STRING,tm:TIMESTAMP  
--time_partitioning_field tm
```

Integer-typed column

```
bq mk --table --schema "customer_id:integer,value:integer"  
--range_partitioning=customer_id,0,100,10 my_dataset.my_table
```

Partitioning can improve query cost and performance by reducing data being queried

```
SELECT  
    field1  
FROM  
    mydataset.table1  
WHERE  
    _PARTITIONTIME > TIMESTAMP_SUB(TIMESTAMP('2016-04-15'), INTERVAL 5 DAY)
```

Isolate the partition field in the left-hand side of the query expression!

```
bq query \  
--destination_table mydataset.mytable  
--time_partitioning_type=DAY --require_partition_filter  
...
```

BigQuery automatically sorts the data based on values in the clustering columns

c1	c2	c3	eventDate	c5
			2019-01-01	
			2019-01-02	
			2019-01-03	
			2019-01-04	
			2019-01-05	

```
SELECT c1, c3 FROM ...
WHERE eventDate BETWEEN "2019-01-03" AND
"2019-01-04"
```

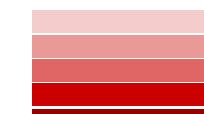
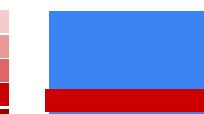
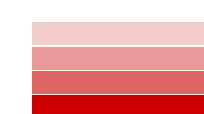
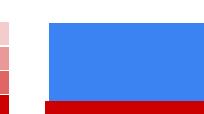
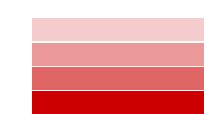
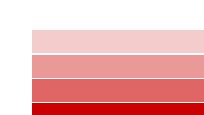
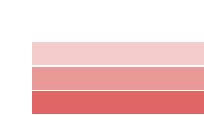
Partitioned tables

c1	userId	c3	eventDate	c5
			2019-01-01	
			2019-01-02	
			2019-01-03	
			2019-01-04	
			2019-01-05	

```
SELECT c1, c3 FROM ... WHERE userId BETWEEN 52
and 63 AND eventDate BETWEEN "2019-01-03" AND
"2019-01-04"
```

Clustered tables

Set up clustering at table creation time

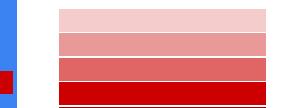
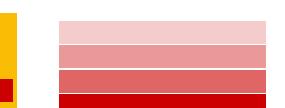
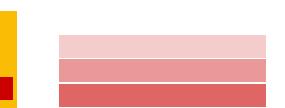
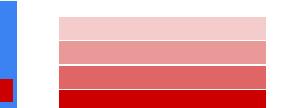
c1	userId	c3	eventDate	c5
			2019-01-01	
			2019-01-02	
			2019-01-03	
			2019-01-04	
			2019-01-05	

```
CREATE TABLE mydataset.myclusteredtable
(
  c1 NUMERIC,
  userId STRING,
  c3 STRING,
  eventDate TIMESTAMP,
  C5 GEOGRAPHY
)
PARTITION BY DATE(eventDate)
CLUSTER BY userId
OPTIONS (
  partition_expiration_days=3,
  description="cluster")
AS SELECT * FROM mydataset.myothertable
```

In streaming tables, the sorting fails over time, and so BigQuery has to recluster

```
UPDATE ds.table  
SET c1 = 300  
WHERE c1 = 300  
AND eventDate > TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 1 DAY)
```

Can force a recluster using DML on necessary partition

c1	userId	c3	eventDate	c5
			2019-01-01	
			2019-01-02	
			2019-01-03	
			2019-01-04	
			2019-01-05	

BigQuery will automatically recluster your data

Automatic re-clustering

free

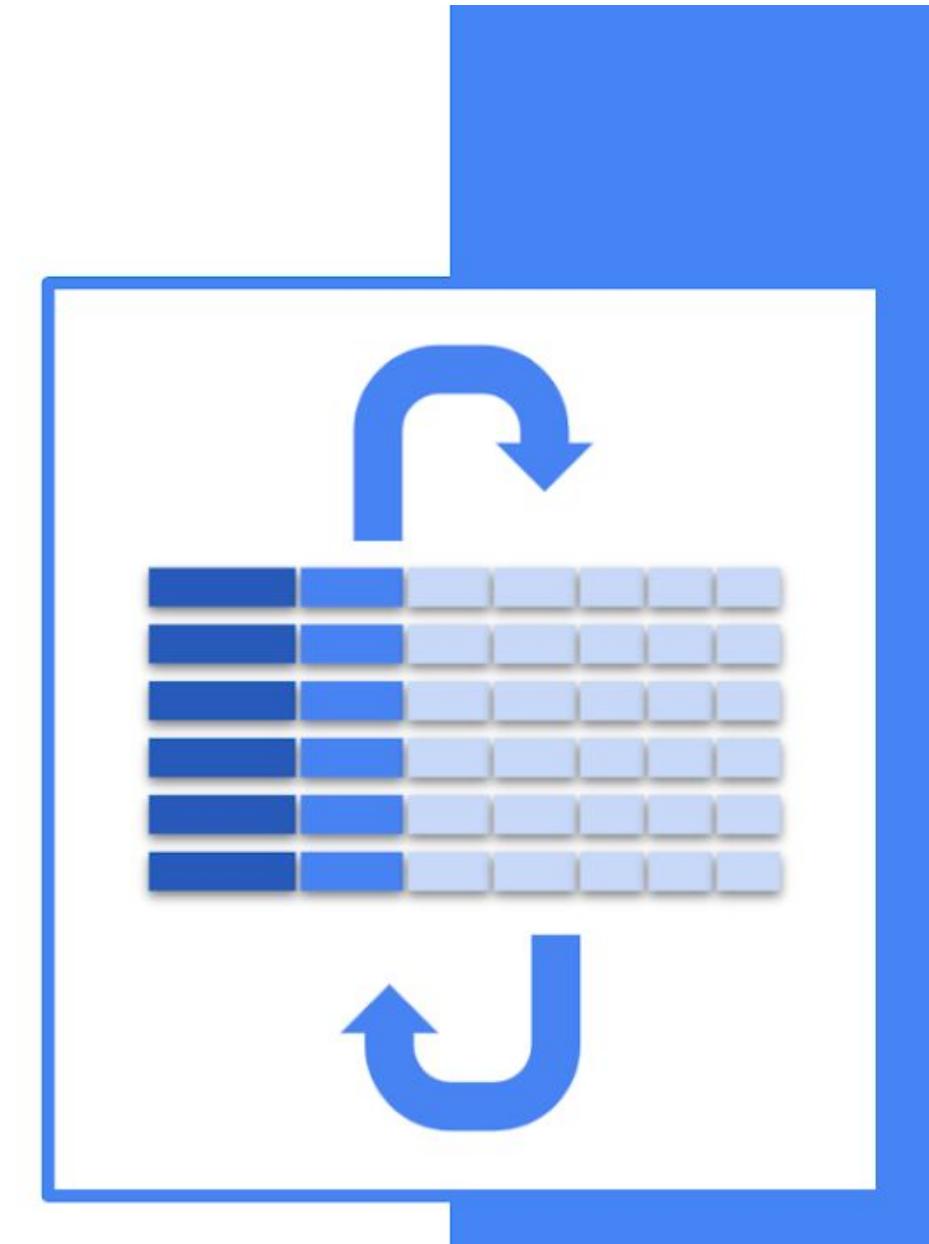
Doesn't consume your query resources

maintenance-free

Requires no setup or maintenance

autonomous

Automatically happens in the background



Organize data through managed tables

Partitioning

Filtering storage before query execution begins to reduce costs.

Reduces a full table scan to the partitions specified.

A single column results in lower cardinality (e.g., thousands of partitions).

- Time partitioning (Pseudocolumn)
- Time partitioning (User Date/Time column)
- Integer range partitioning

Clustering

Storage optimization within columnar segments to improve filtering and record colocation.

Clustering performance and cost savings can't be assessed before query begins.

Prioritized clustering of up to 4 columns, on more diverse types (but no nested columns).

Demo

Partitioned and Clustered Tables in BigQuery

When to use clustering



Your data is already partitioned on a DATE or TIMESTAMP or Integer Range



You commonly use filters or aggregation against particular columns in your queries.

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYS and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

What if your data is not usable in its original form?



Data Processing



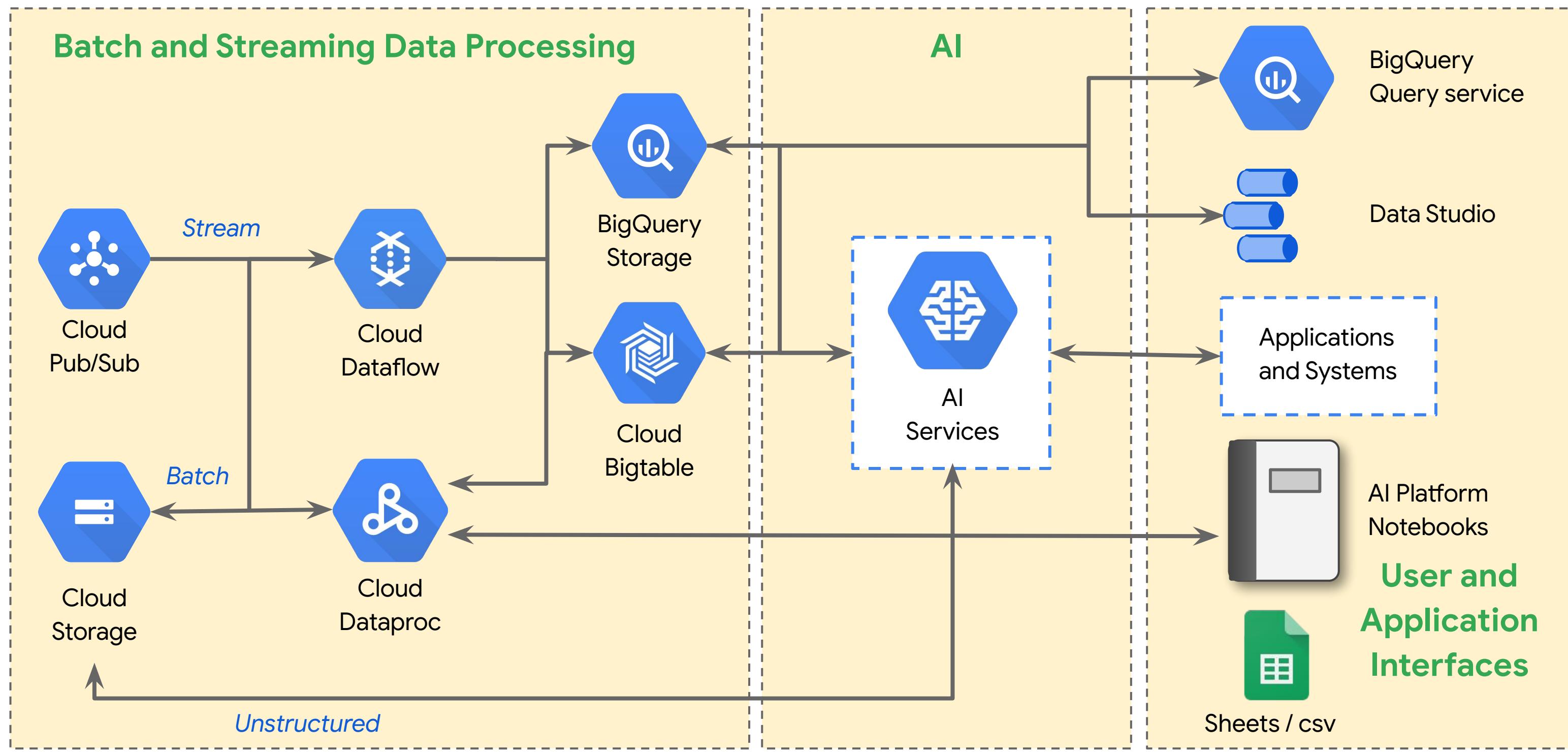
Cloud Dataproc



Cloud Dataflow

Data is processed in an intermediate system before it is loaded into the analytics warehouse.

A typical pipeline of GCP products that support ETL



What if your data arrives continuously and endlessly?



Streaming Data Processing



Cloud
Pub/Sub



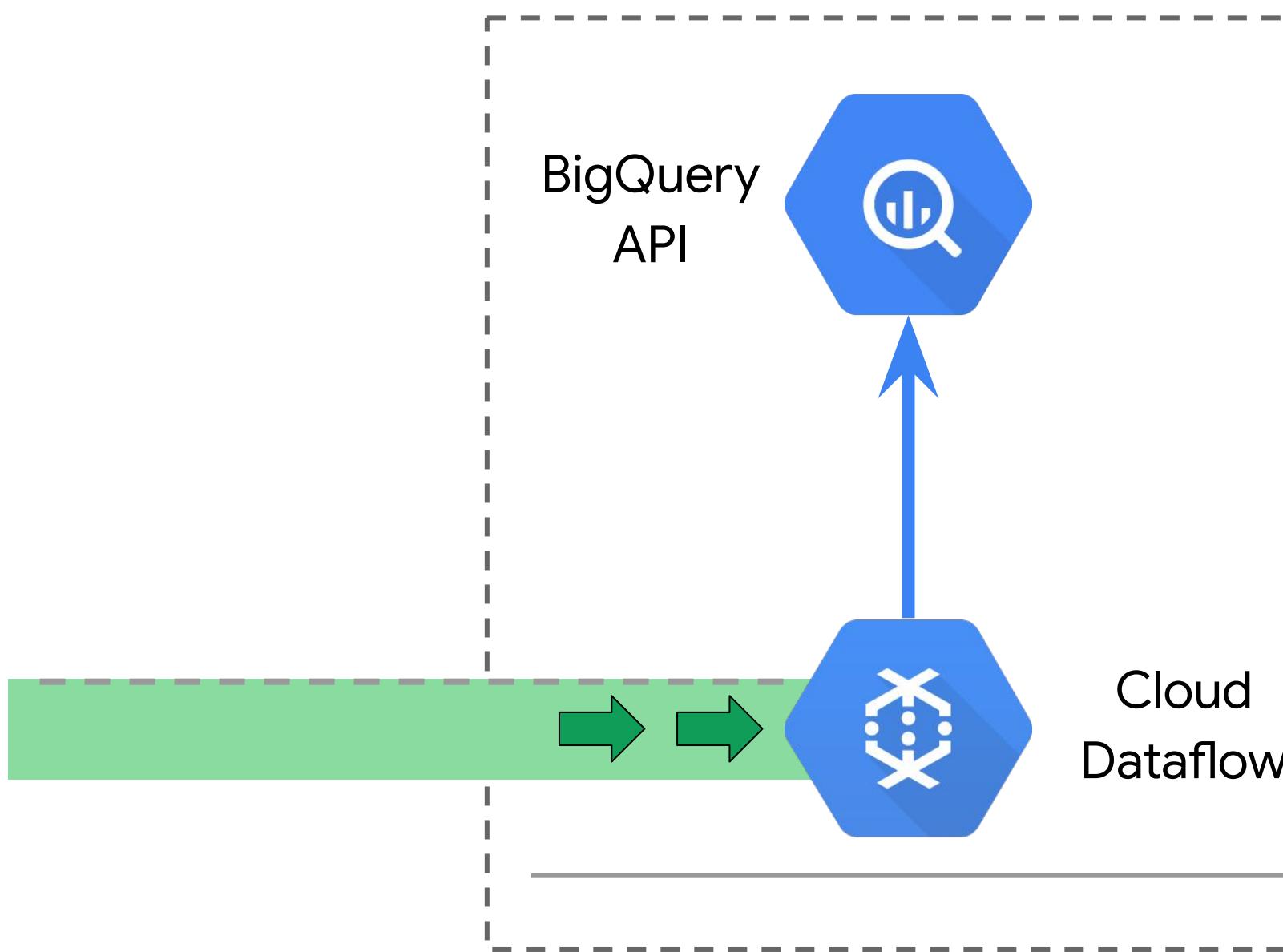
Cloud
Dataflow



BigQuery

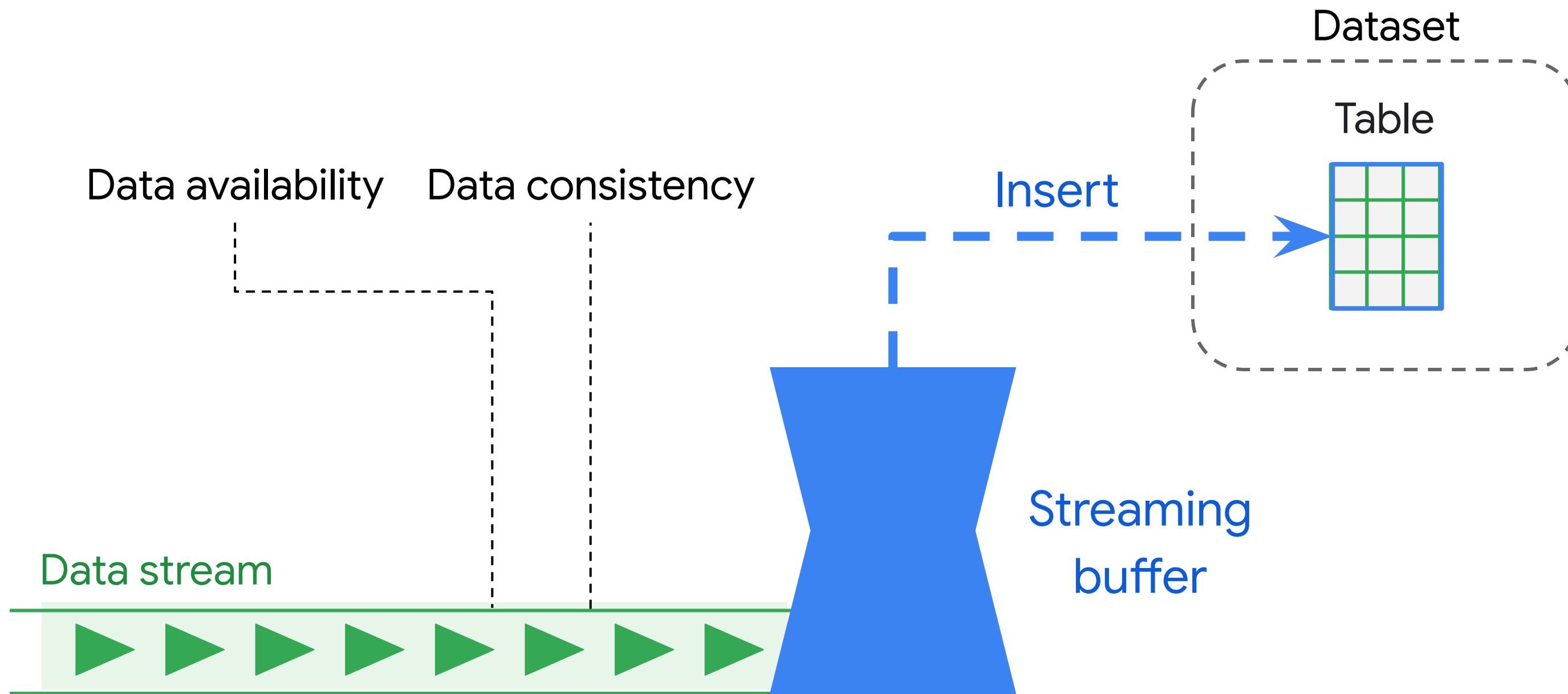
Data flows are buffered and processed rapidly in pipelines.

Some data is real-time and streamed into BigQuery by inserting records into tables via Cloud Dataflow



Insert records into tables from streams (unbounded) using a Dataflow pipeline

You can also use the BigQuery streaming API directly



Recap

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

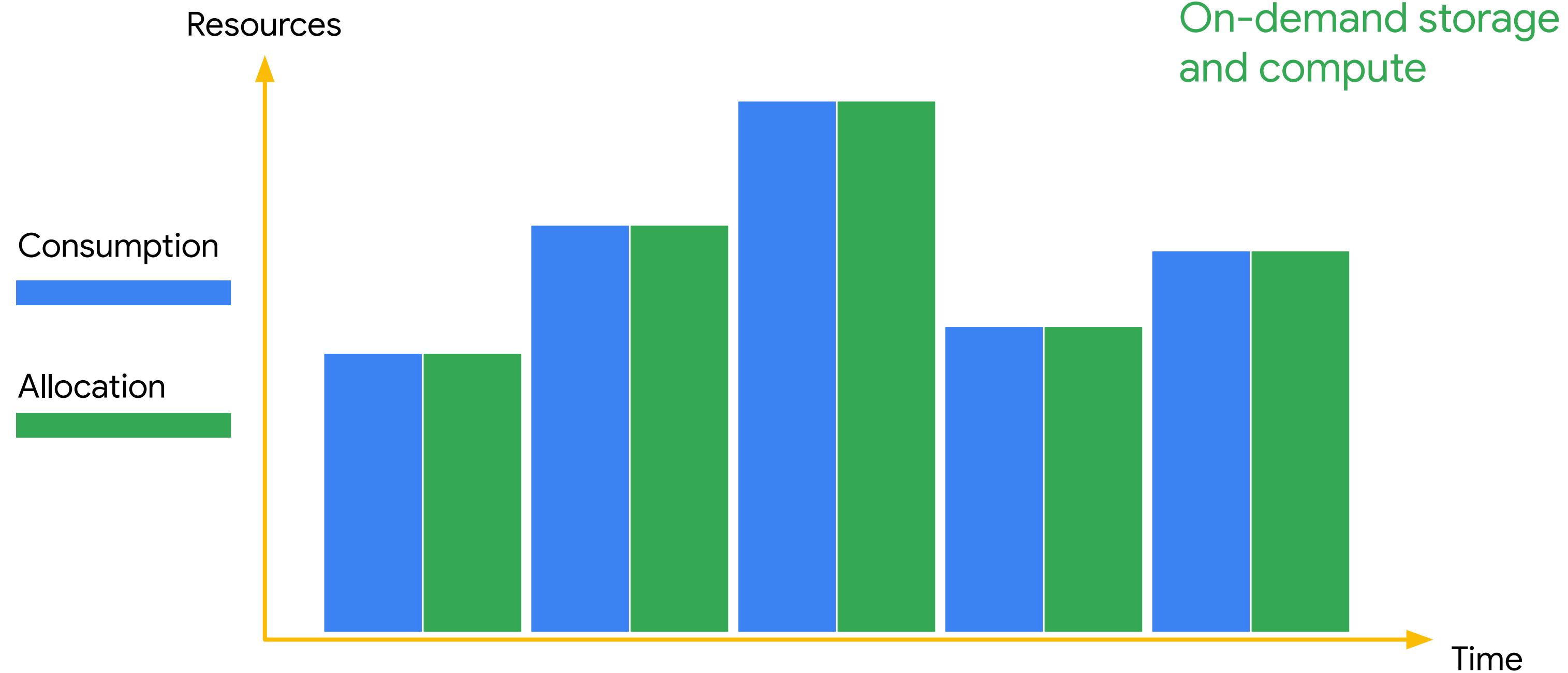
- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYS and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

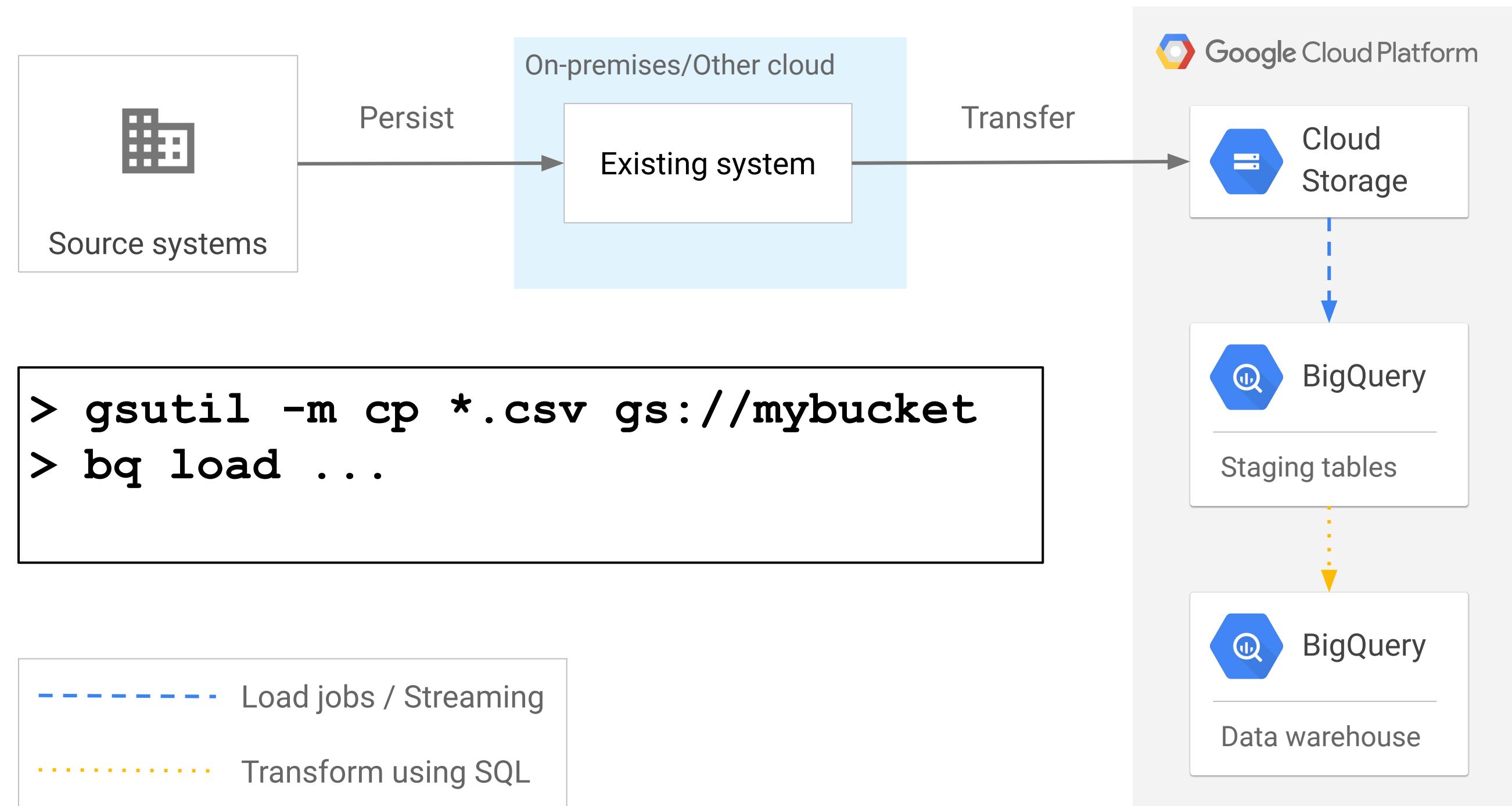
A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops,
including ad hoc queries
- Ecosystem of visualization and
reporting tools
- Ecosystem of ETL and data
processing tools
- Up-to-the-minute data
- Machine learning
- Security and collaboration

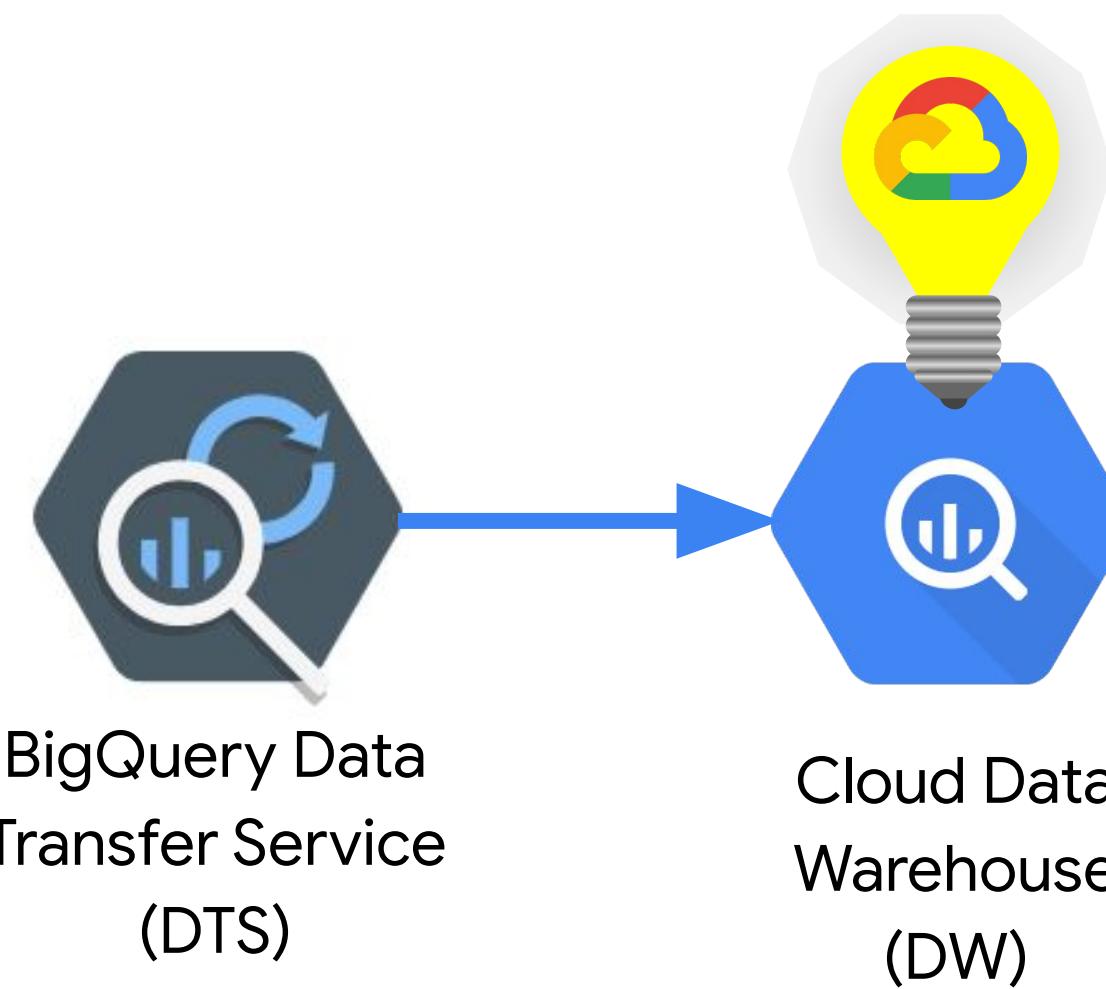
You don't need to provision resources before using BigQuery



Loading data through Cloud Storage



BigQuery Data Transfer Service helps you build and manage your data warehouse

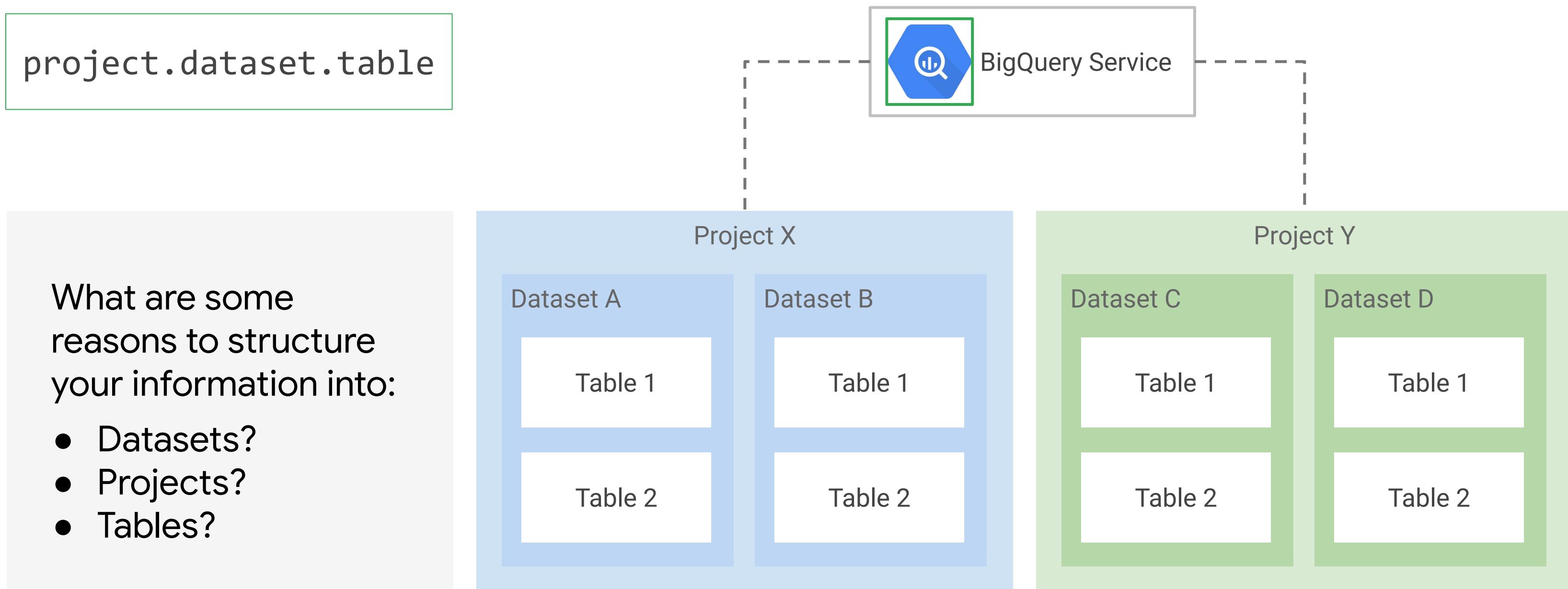


EL

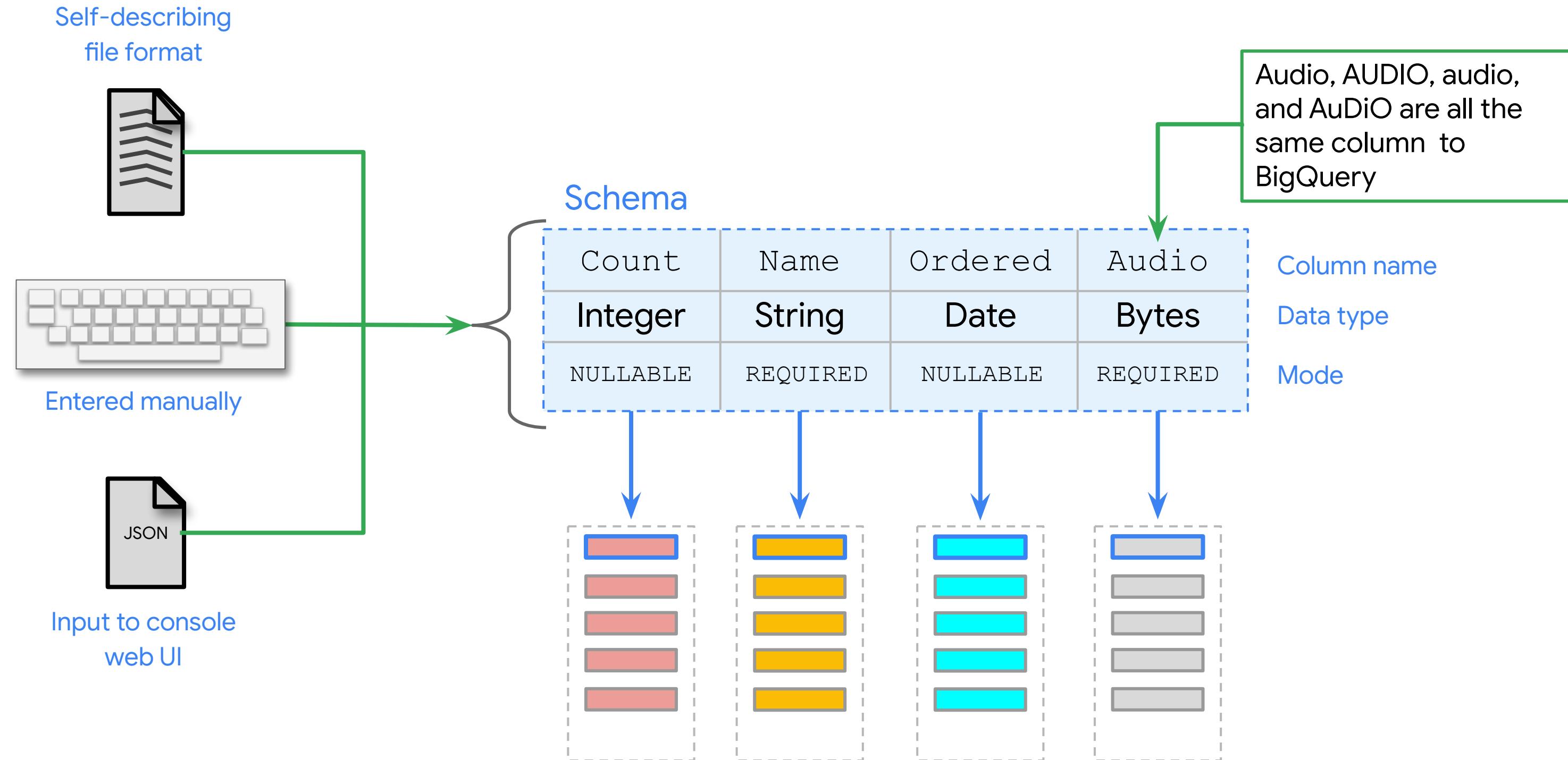
- Managed service
- Automatic transfers
- Scheduled
- Data staging
- Data processing
- Data backfills

BigQuery organizes data tables into units called datasets

project.dataset.table



The table schema provides structure to the data

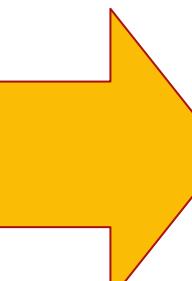


Nested and repeated columns improve the efficiency of BigQuery with relational source data

Denormalized flattened table

Product Name	OrderID	Date	Quantity
Siding	1600p	08/20/2019	3 boxes
Caulk	1600p	08/20/2019	12 tubes
Soffit	221b	10/29/2019	17 meters
Sealant	221b	11/05/2019	1 liter
Soffit	1600p	09/09/2018	34 meters
Siding	221b	08/20/2019	2 boxes
Caulk	221b	09/09/2019	4 tubes
Sealant	1600p	08/20/2018	2 liters

Denormalized with nested and repeated data



Order.ID	Order.Date	Order.Product_Name	Order.Quantity
1600p	08/20/2019	Siding	3 boxes
		Caulk	12 tubes
		Soffit	34 meters
		Sealant	2 liters
		221b	17 meters
		Sealant	1 liter
		Siding	2 boxes
		Caulk	4 tubes

BigQuery automatically sorts the data based on values in the clustering columns

c1	c2	c3	eventDate	c5
			2019-01-01	
			2019-01-02	
			2019-01-03	
			2019-01-04	
			2019-01-05	

```
SELECT c1, c3 FROM ...
WHERE eventDate BETWEEN "2019-01-03" AND
"2019-01-04"
```

Partitioned tables

c1	userId	c3	eventDate	c5
			2019-01-01	
			2019-01-02	
			2019-01-03	
			2019-01-04	
			2019-01-05	

```
SELECT c1, c3 FROM ... WHERE userId BETWEEN 52
and 63 AND eventDate BETWEEN "2019-01-03" AND
"2019-01-04"
```

Clustered tables