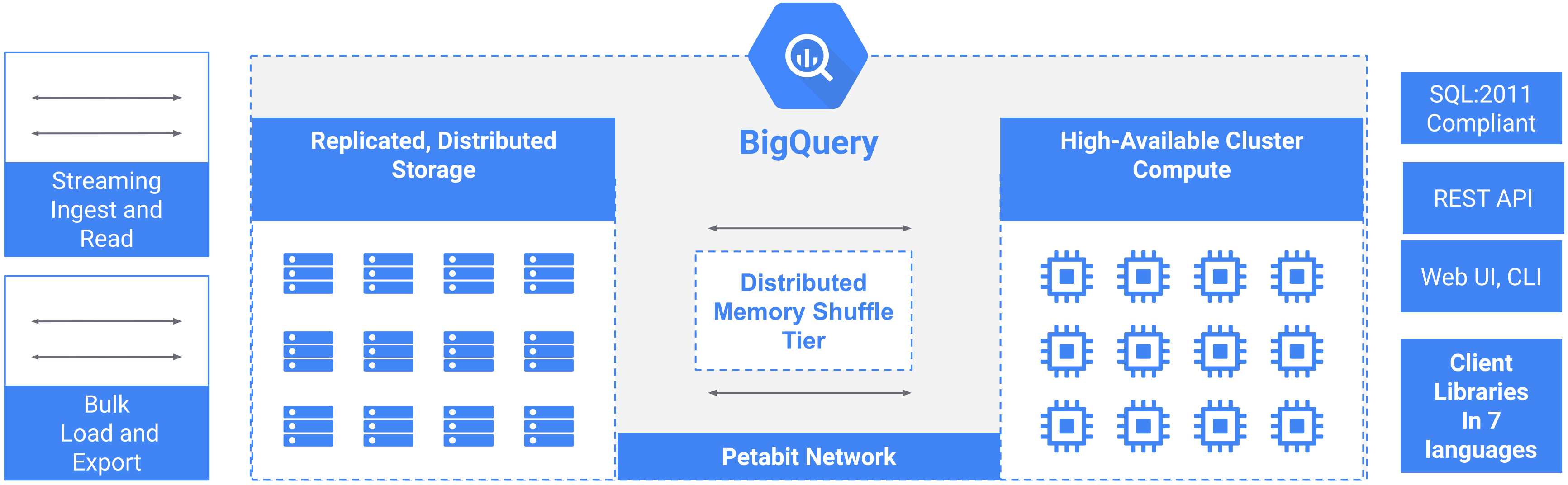
Integrating Unstructured Data Into a Cloud Data Warehouse

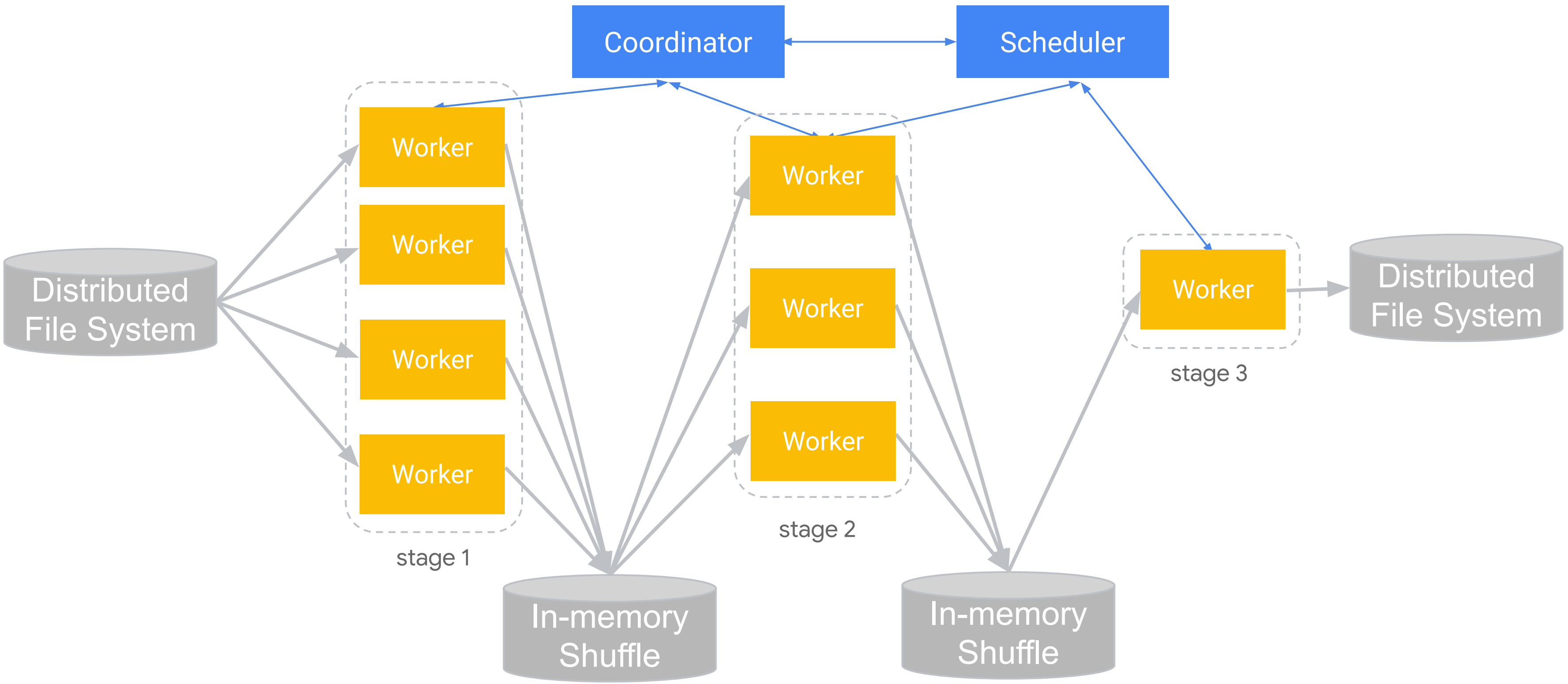
Justin Levandoski | UC Berkeley EPIC Lab Offsite

Google Cloud

BigQuery Architecture



Dremel: BigQuery’s Query Processing Engine

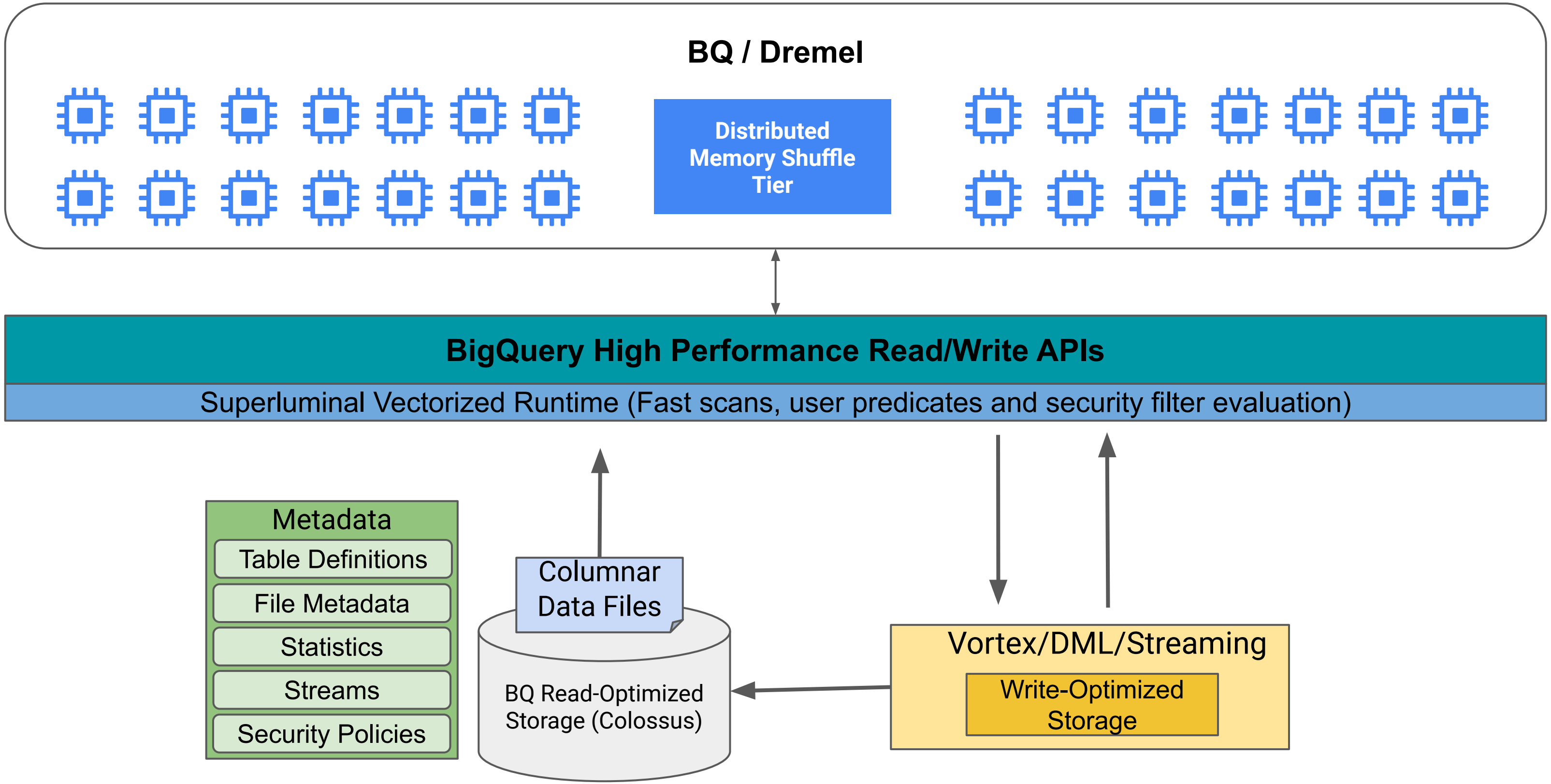


“Serverless” Design Principles and Advantages

- **Disaggregation of compute, storage, memory / shuffle**
  - On-demand scaling of each resource
  - On-demand sharing of resources
  - Adapts well to ***multi-tenant*** usage at lower cost

- **Architecture advantages**
  - Separation of concerns across “components” or “micro-services”
  - Re-use / evolve components as needed without large “blast radius”

BigQuery Storage Read/Write APIs



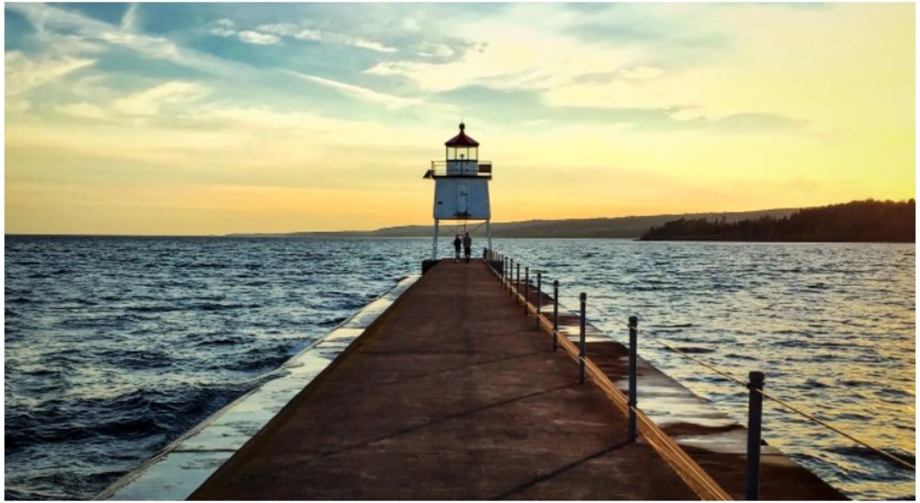
BigLake and Omni

Extending BigQuery's Reach

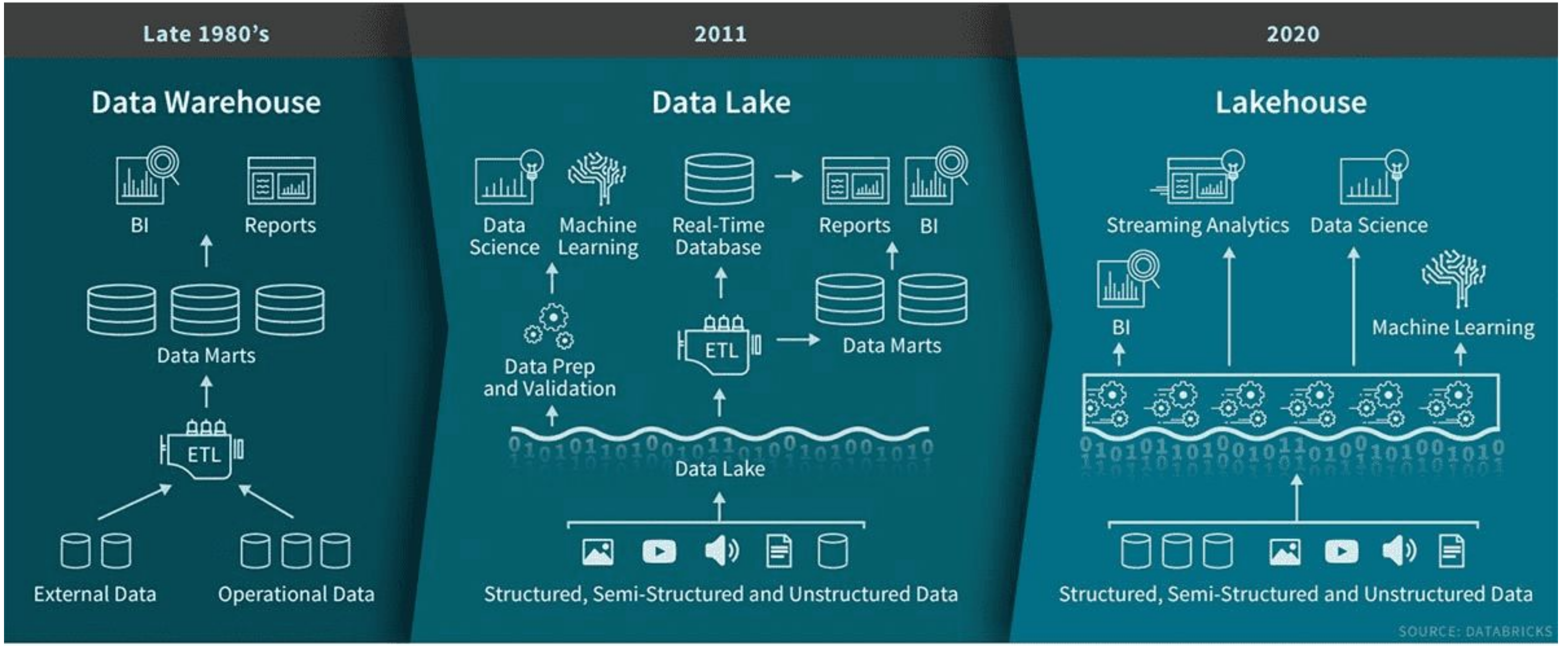
Google Cloud launches BigLake, a new cross-platform data storage engine

Frederic Lardinois @fredericl / 10:00 PM PDT • April 5, 2022

Comment



Toward the "Lakehouse"

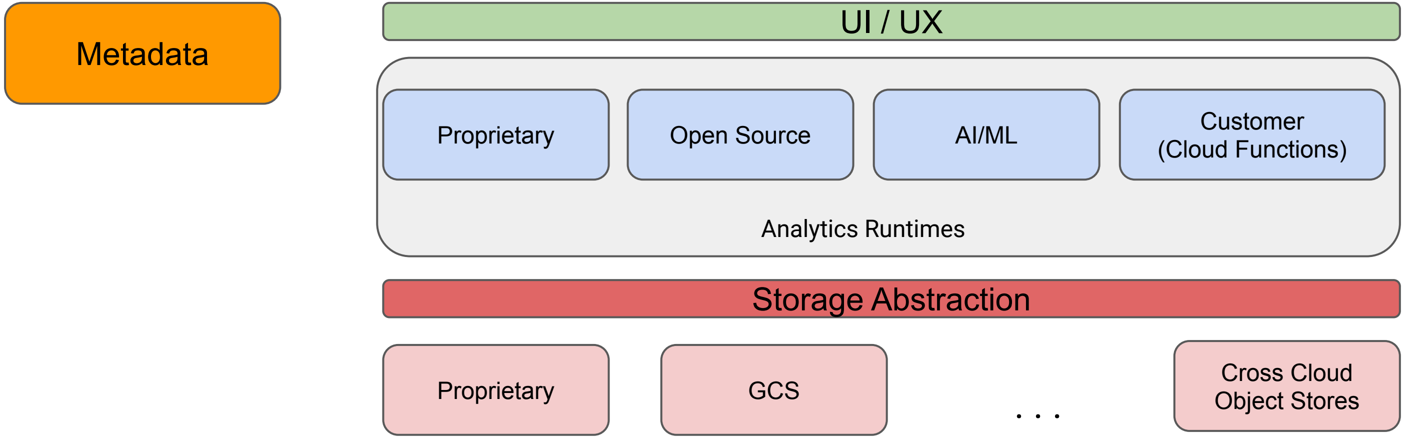


https://databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html

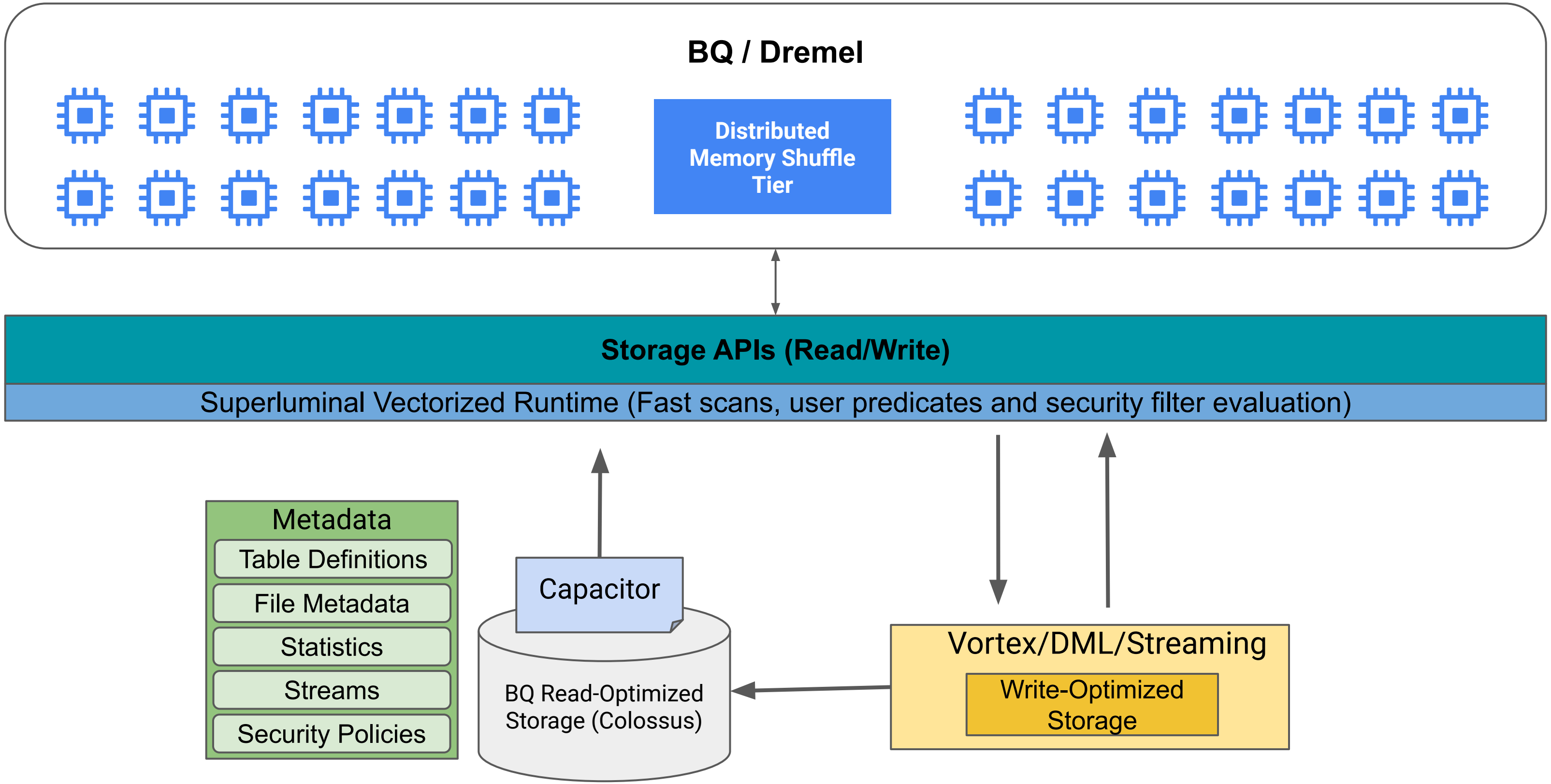
Toward a “Lakehouse”

- **Lakehouse features**
  - Support for diverse workloads and analytics engines
  - Diverse data types spanning structure, semi-structured, and unstructured
  - Support for open formats on object storage – with object storage as the “hub”
  - Common governance support
  - Support for streaming / ACID transactions

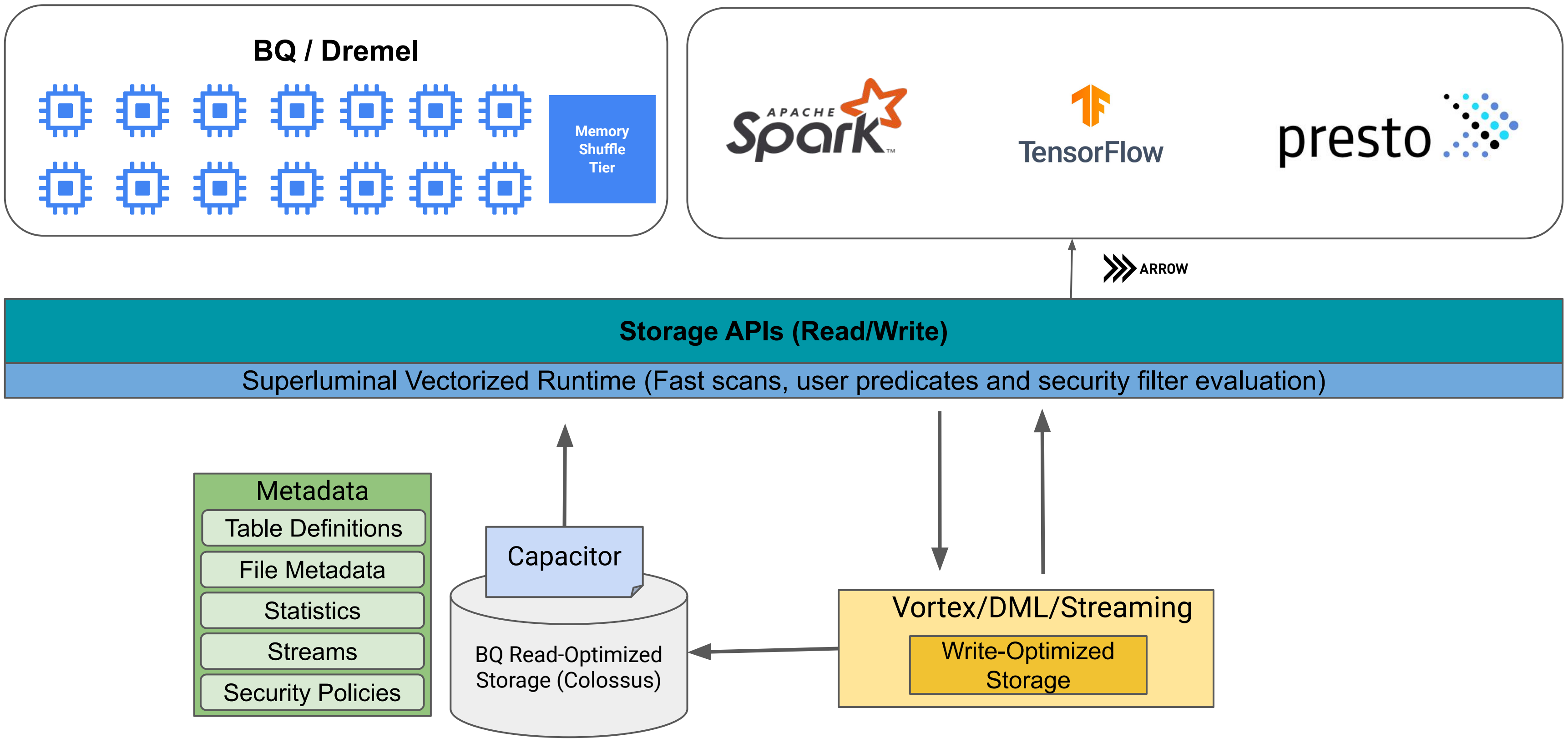
- **A coupling of systems bound together by common data management principles**
  - Choice in data analytics engine usage, storage format and deployment in how they build stacks
  - *Single story* for core data management issues in analytics (the difficult stuff): governance, performance, consistency/transactions, etc.



BigQuery Architecture: Separation of Compute and Storage

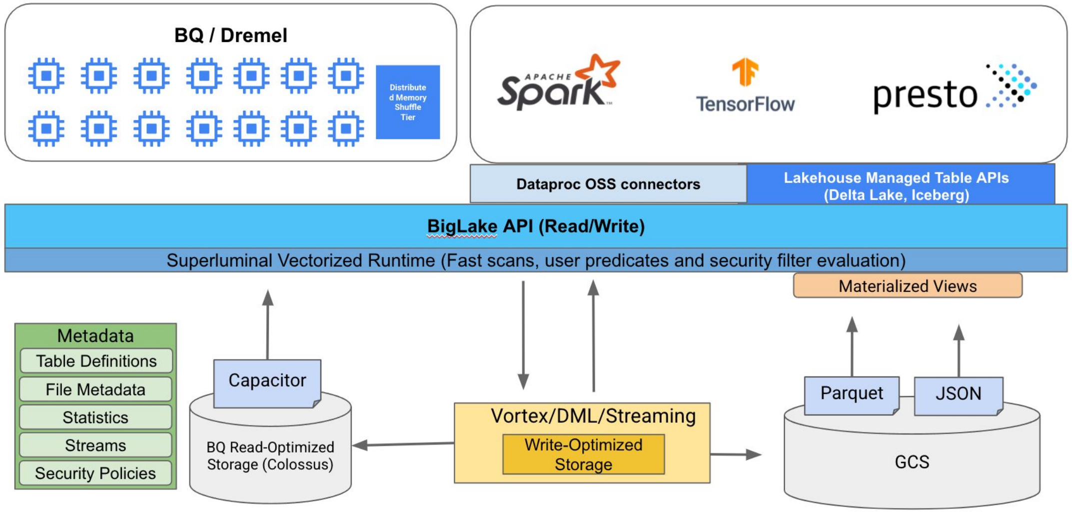


BigQuery Architecture with BigQuery Storage API for OSS Engines



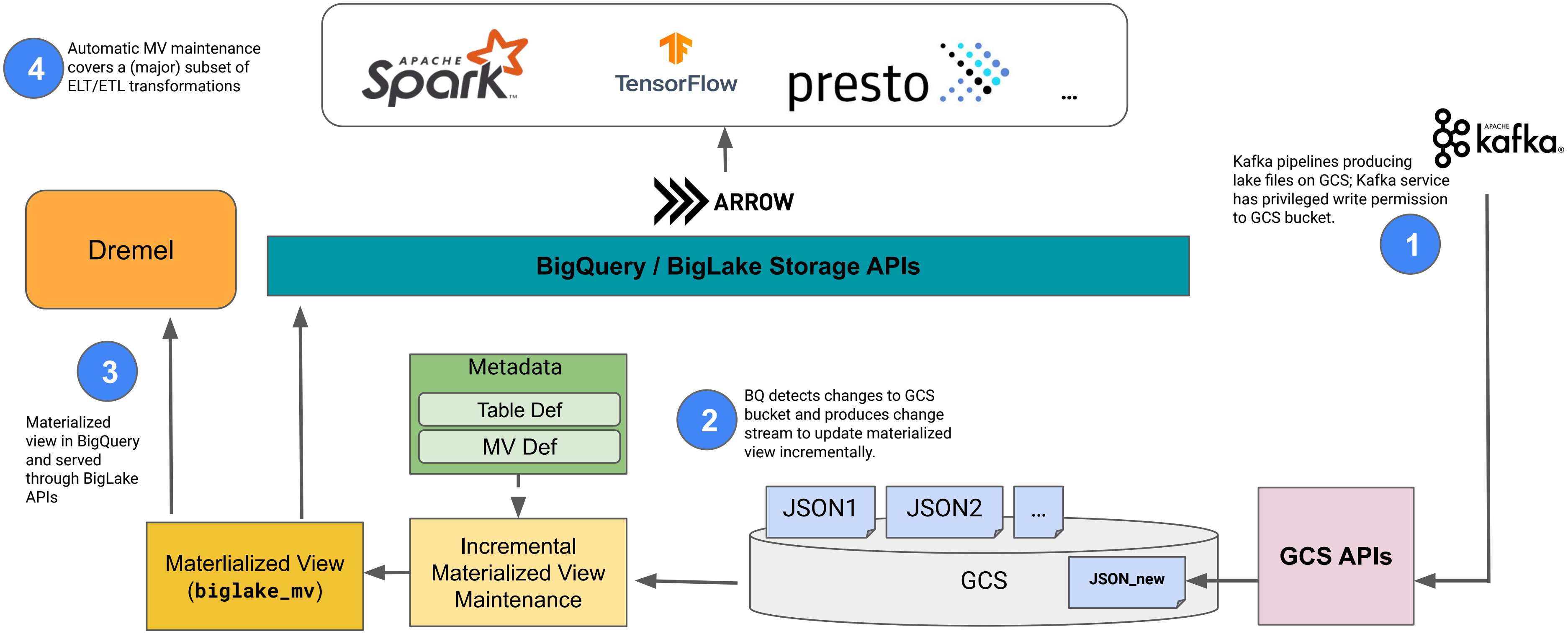
BigLake - Expanding BigQuery Capabilities to GCS and other Object Stores

- **Vectorized runtime** with columnar scanners efficiently evaluate user predicates
- Data served in **Apache Arrow** format with security enforced prior to egress
- **Vortex stream ingest** for efficient streaming and (eventually) DML
- **Materialized views** on BQ and BigLake tables served through API
- Compatibility with major data lake managed table APIs: Databricks **Delta Lake and Apache Iceberg**
- Flexibility in usage modes, e.g., GCS APIs for ingress

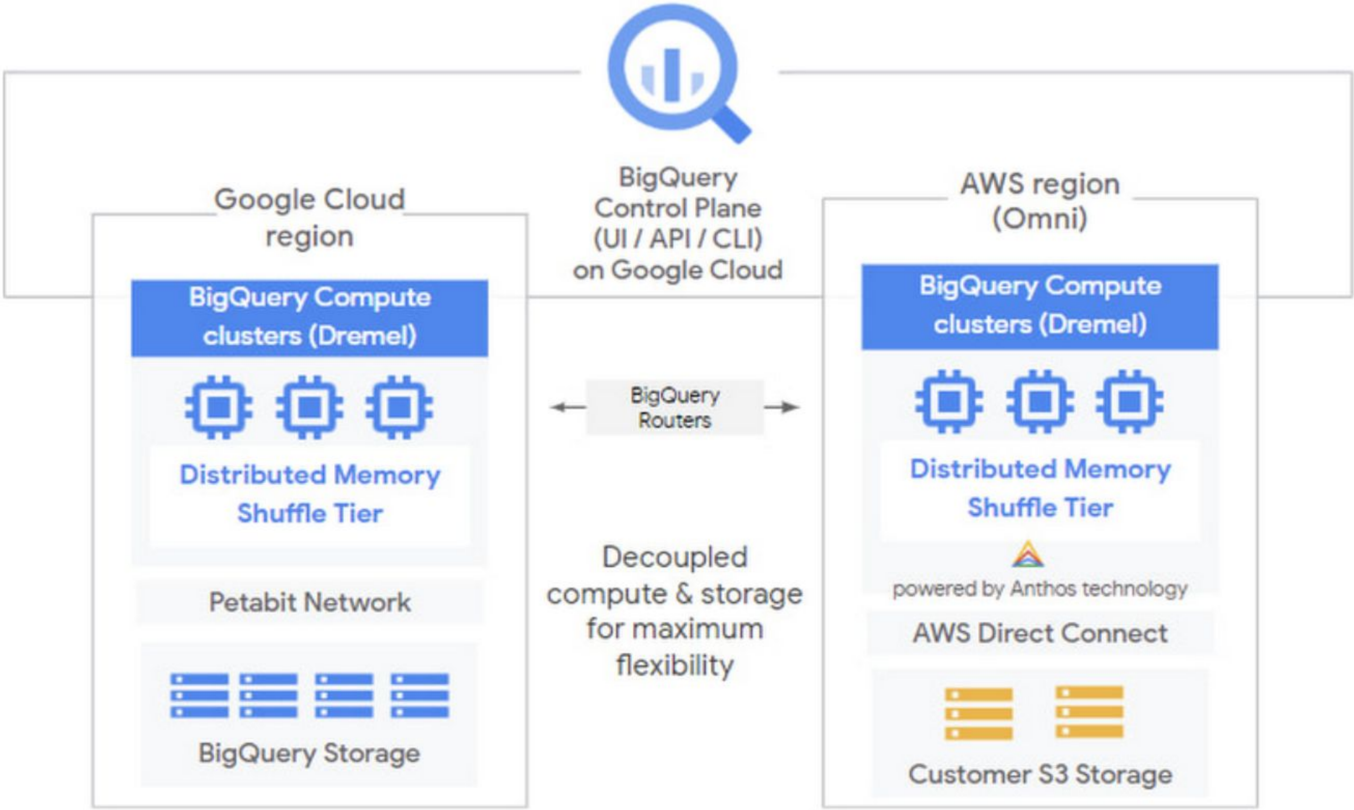


BigLake: Materialized Views / ELT

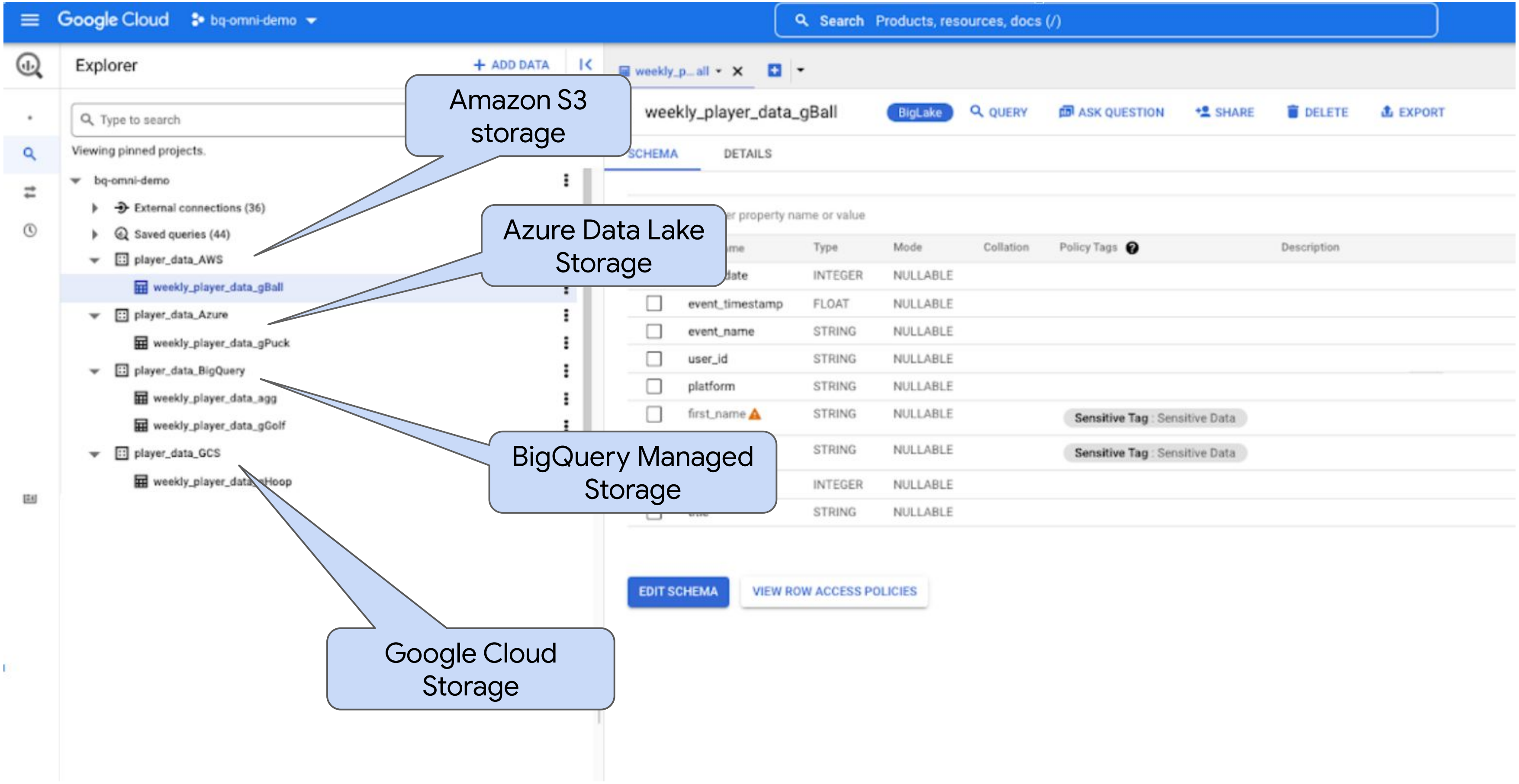
```
CREATE MATERIALIZED VIEW biglake_mv
AS "SELECT COL1 SUM(COL2)
    FROM biglake_table
    GROUP BY COL1";
```



BigQuery Omni: Extending BigQuery to AWS and Azure



Querying a Multi-Cloud Lakehouse

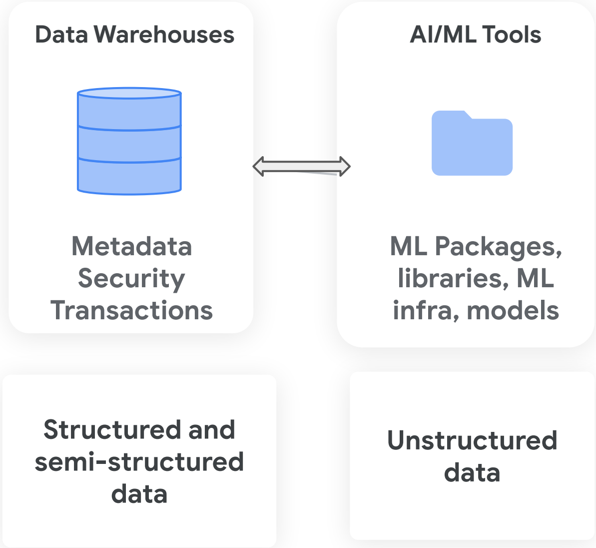


Unstructured Data

In a Cloud Data Warehouse

Unstructured Data and Data Warehousing: The Next Frontier

- Unstructured data analytics generally requires ML tools, separate from data warehouses
- Bridging this gap requires infra, specialized teams and introduces security challenges
- This constraints unstructured data use cases that can be prioritized
- Our solution: BigQuery management of unstructured data as a first-class citizen



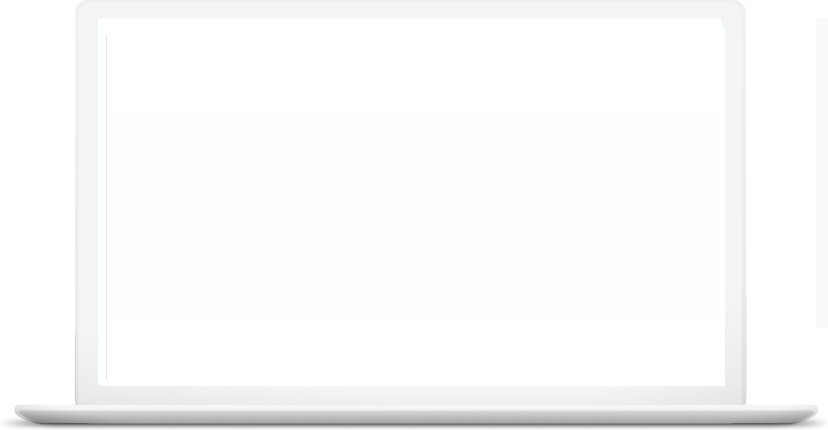
BigQuery Object Tables: Unstructured Data Management

- A SQL interface to object store metadata.
- BigQuery's window into the world of unstructured data
- Delegated access to files: Access to a row === access that file content.

```
CREATE EXTERNAL TABLE dataset1.images
WITH CONNECTION 'us.biglake1'
OPTIONS (uris=['gs://mybucket/*'],
         object_metadata='DIRECTORY')
```

uri	create_time	generation	...
bucket/image1.jpg	2021-11-04	2rba7gbp0	
bucket/image2.jpg	2021-11-05	gbp02rba7	
bucket/image3.jpg	2021-11-06	p02rbgbgb	

BigQuery ML: Native ML support in a data warehouse



Classification

- Logistic regression
- DNN classifier (TensorFlow)
- Boosted trees using XGBoost
- AutoML Tables

Regression

- Linear regression
- DNN regressor (TensorFlow)
- Boosted trees using XGBoost
- AutoML Tables

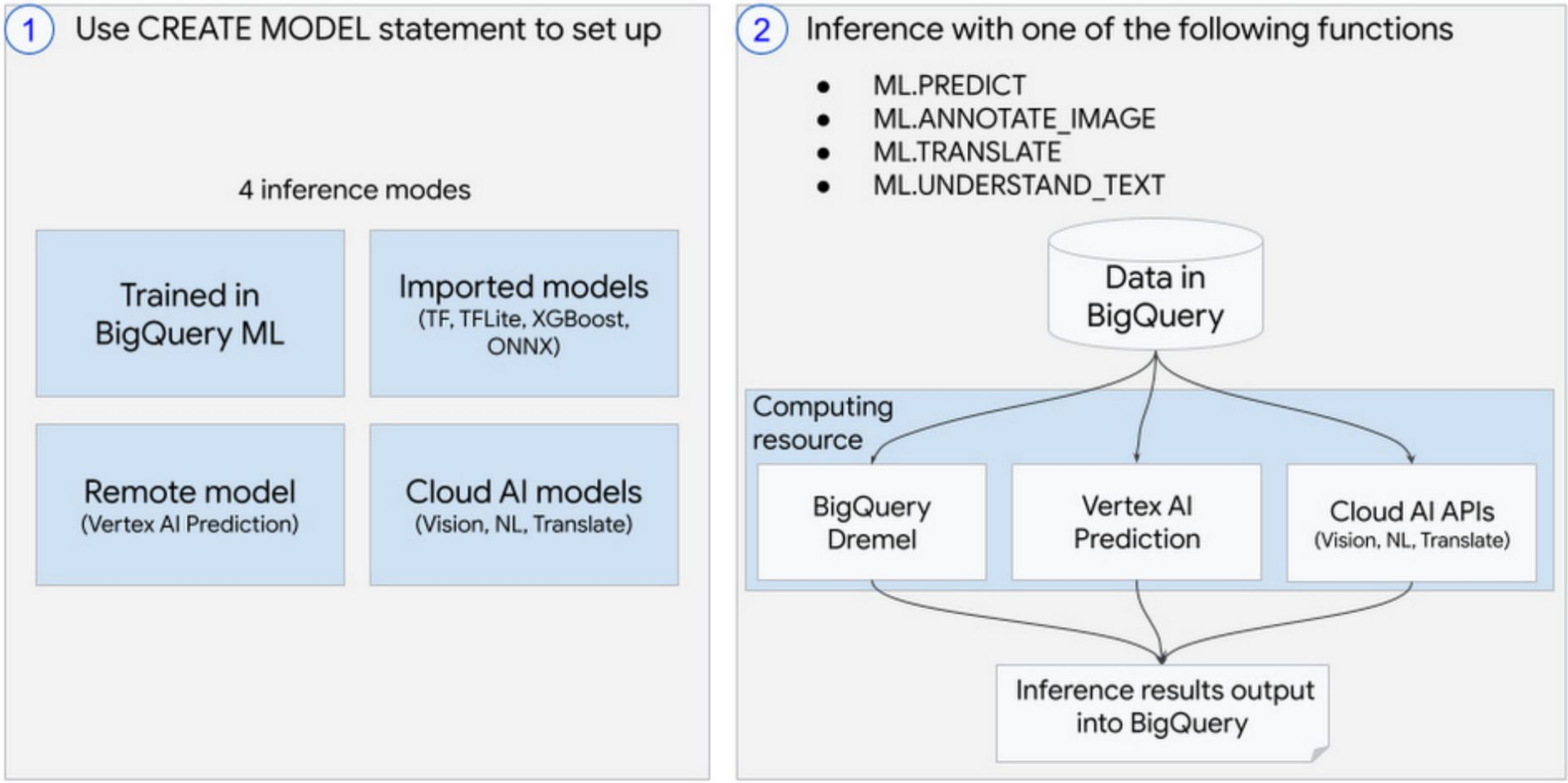
Other Models

- k-means clustering
- Time series forecasting
- Recommendation: Matrix factorization

Model Import/Export

- TensorFlow models for batch and online prediction

BigQuery ML Inference Engine



Object Tables and **BQML**

```
CREATE EXTERNAL TABLE dataset1.images
WITH CONNECTION 'us.biglake1'
OPTIONS (uris=['gs://mybucket/*'],
         object_metadata='DIRECTORY')
```

uri	create_time	generation	...
bucket/image1.jpg	2021-11-04	2rba7gbp0	
bucket/image2.jpg	2021-11-05	gbp02rba7	
bucket/image3.jpg	2021-11-06	p02rbgbgb	

```
SELECT * FROM
  ML.PREDICT(MODEL cat_detector,
    SELECT uri FROM dataset1.images
    WHERE ENDSWITH(uri, 'jpg')
    AND create_time > TIMESTAMP('2021-1-1')
  )
```

In the query engine, we

1. Load images from object store
2. Preprocess them (decode, resize)
3. Execute TF inference in across BigQuery's query processing shards

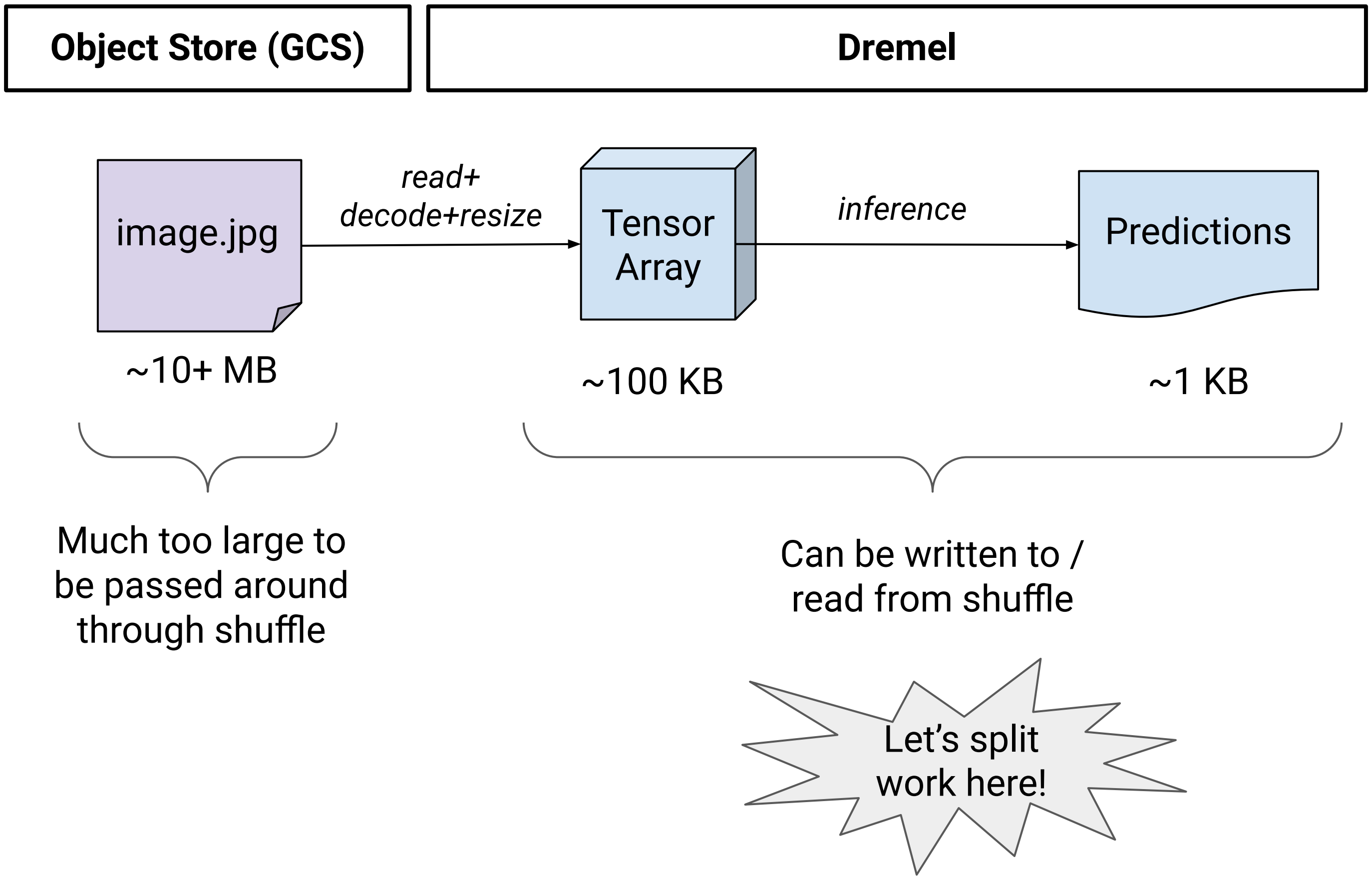
Processing Unstructured Data in Dremel

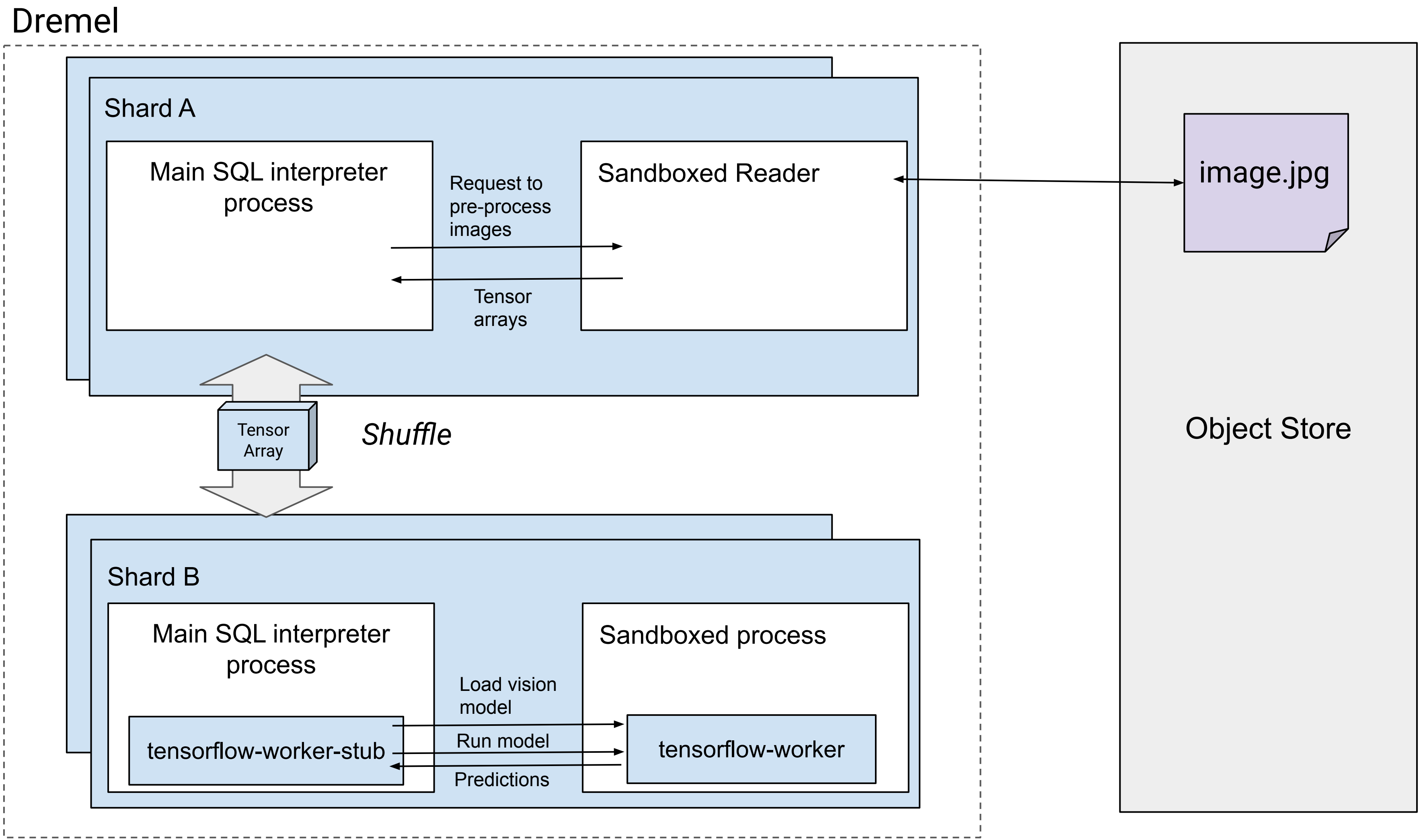
Challenges:

- Unstructured data and associated models typically larger than for structured data.
- Dremel scales only horizontally!
  - Dremel workers are tiny: 8GB of ram (less than your phone?)
- Dremel is serverless: clusters are shared across all users.
  - Scheduling fairly & efficiently is a huge challenge. Introducing special high-compute/memory workers would be a very difficult lift: Requires a complete rethink of scheduling.
  - Shared resources like shuffle are scarce.

Our solution:

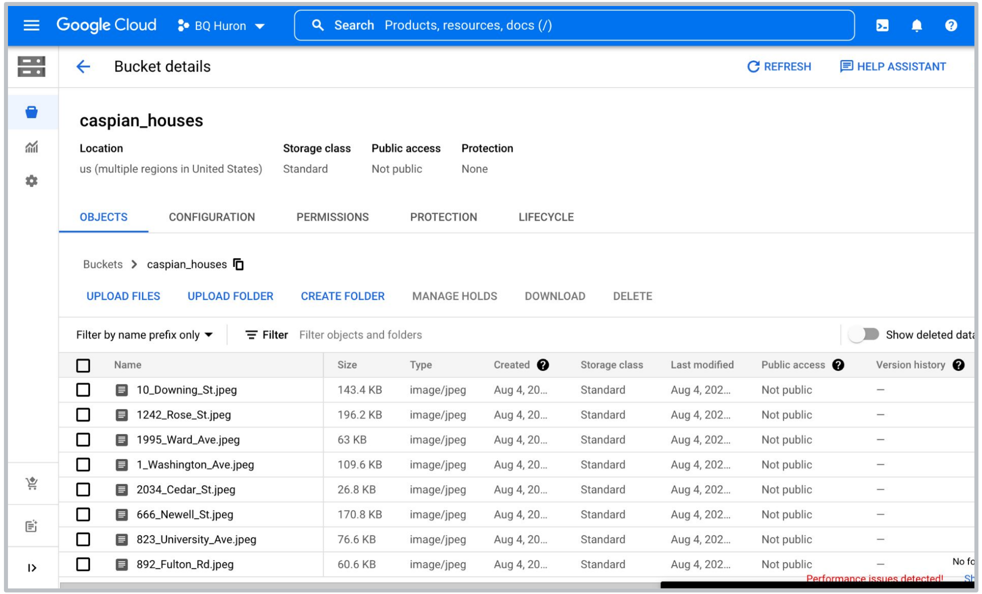
➔ Lean on Dremel's strengths and decompose inference horizontally across workers.


Object Store (GCS)
Dremel
image.jpg
read+
decode+resize
Tensor
Array
inference
Predictions
~10+ MB
~100 KB
~1 KB
Much too large to
be passed around
through shuffle
Can be written to /
read from shuffle
Let’s split
work here!



Dremel
Shard A
Main SQL interpreter process
Request to pre-process images
Tensor arrays
Sandboxed Reader
image.jpg
Object Store
Tensor Array
Shuffle
Shard B
Main SQL interpreter process
tensorflow-worker-stub
Load vision model
Run model
Predictions
Sandboxed process
tensorflow-worker

Example of Object Tables for Images




```sql
CREATE EXTERNAL TABLE dataset.houses
WITH CONNECTION us.demo_lake
OPTIONS (uris=['gs://caspian_houses/*']
        )

SELECT * FROM dataset.houses LIMIT 10
```

uri	generation	content_type	size
gs://caspian_houses/10_Downing_St.jpeg	1659659941032822	image/jpeg	146843
gs://caspian_houses/1242_Rose_St.jpeg	1659659941932150	image/jpeg	200905
gs://caspian_houses/1995_Ward_Ave.jpeg	1659659942122696	image/jpeg	64551
gs://caspian_houses/1_Washington_Ave.jpeg	1659659940828040	image/jpeg	112249
gs://caspian_houses/2034_Cedar_St.jpeg	1659659942231821	image/jpeg	27476
gs://caspian_houses/666_Newell_St.jpeg	1659659941129893	image/jpeg	174938
gs://caspian_houses/823_University_Ave.jpeg	1659659941526406	image/jpeg	78473
gs://caspian_houses/892_Fulton_Rd.jpeg	1659659941628295	image/jpeg	62013

Customer Use Case: Adswerve and Twiddy & Co

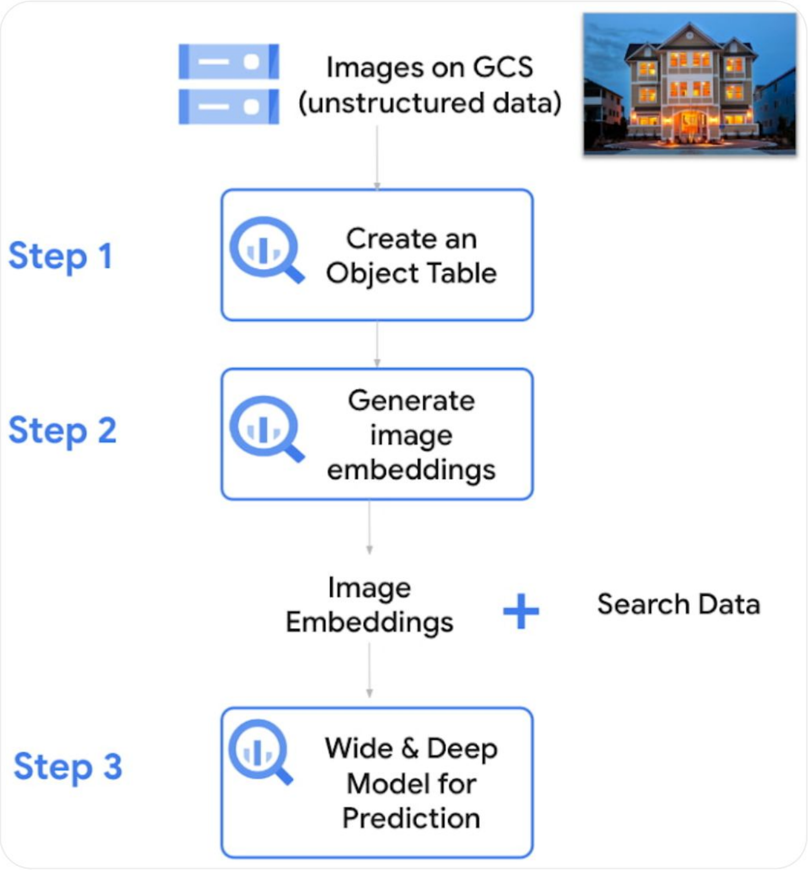
"As a local family vacation rental business specializing in delivering hospitality for nearly 45 years, we've always strived for our vacation home images to convey the unique local experience that our homes offer. BigQuery ML made it really easy for our business analysts to find just the right images by analyzing thousands of potential options and combining them with existing click-through data. This, otherwise, would have taken a lot longer or simply we wouldn't have done it at all."

— Shelley Tolbert, Director of Marketing, Twiddy & Company

Customer Use Case: Adswerve and Twiddy & Co

- **Goal:** build ML models using both website search data and rental listing images to predict the click-through rate of the rental properties.
- **Challenges**
  - Previously only relied on structured data (e.g. location, size) to predict what customers might like
  - The editorial team uses a manual photo selection process
  - Require data science resources to build machine learning pipelines and processing data to resize images is *labor intensive*

Adswerve and Twiddy & Co: 3 Steps to Success



Adswerve and Twiddy & Co: object table creation

```
CREATE OR REPLACE EXTERNAL TABLE
  `images.property_images`
WITH CONNECTION
  `us.datalake`
OPTIONS(uris=["gs://demo/images/*"]
        object_metadata="DIRECTORY",
        max_staleness=INTERVAL 30 MINUTE,
        metadata_cache_mode="AUTOMATIC");
```

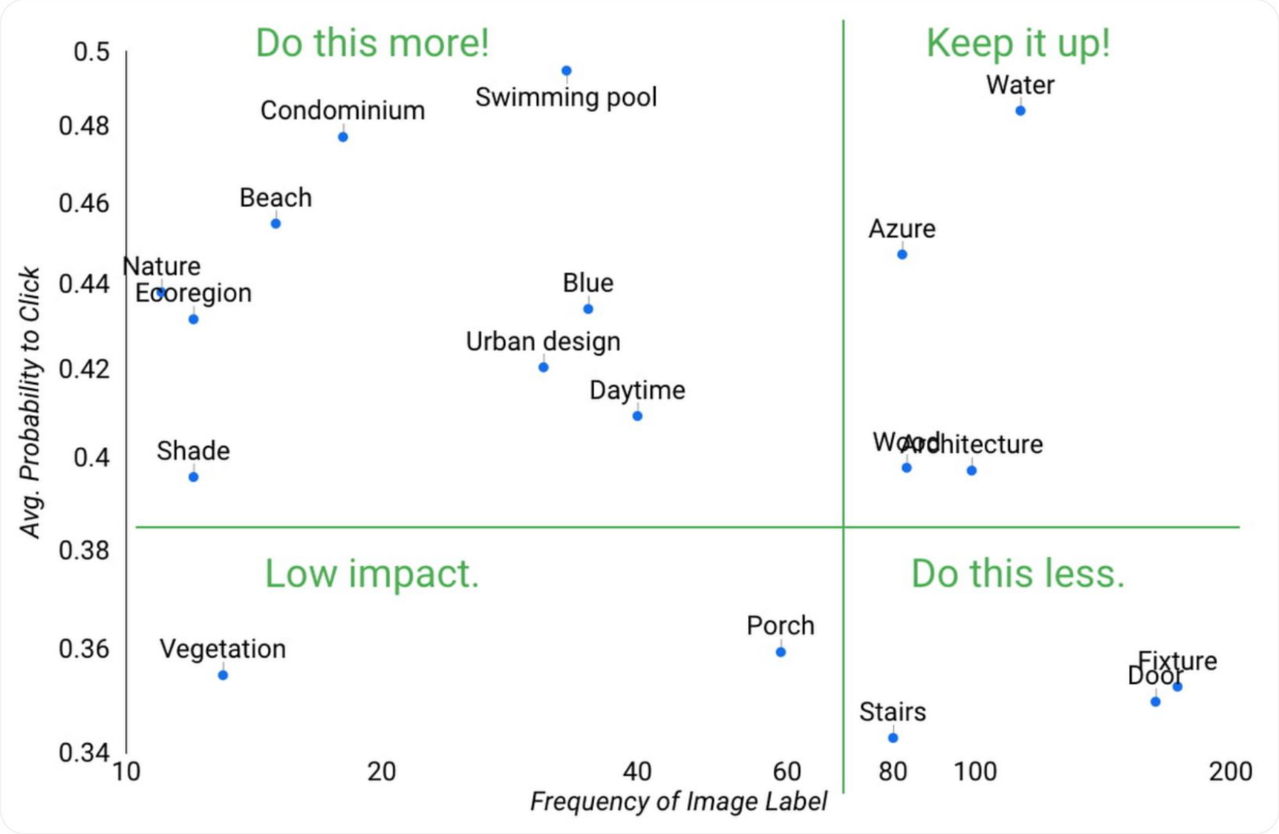
uri	content_type	size
gs://as-bqml-launch-demo/images/ER010-aerialrearext.jpg	image/jpeg	1940969
gs://as-bqml-launch-demo/images/KD1111-rearext.jpg	image/jpeg	1508587
gs://as-bqml-launch-demo/images/ER011-aerialrearext.jpg	image/jpeg	1949278
gs://as-bqml-launch-demo/images/B800-rearext.jpg	image/jpeg	338409
gs://as-bqml-launch-demo/images/b372-rearext.jpg	image/jpeg	1841978
gs://as-bqml-launch-demo/images/J10975-aerial-2.jpg	image/jpeg	1606464

Adswerve and Twiddy & Co: image embedding generation

```
CREATE OR REPLACE MODEL
  `pipeline.resnet_embeddings`
OPTIONS(model_type="TENSORFLOW",
        model_path="gs://resnet-embeddings/*")

SELECT *
FROM ML.PREDICT(
  MODEL `pipeline.resnet_embeddings`,
  (SELECT ML.DECODE_IMAGE(data)
   FROM `images.property_images`)
);
```

img_name	pc1	pc2	pc3
EC1-Bar.jpg	1.8493...	1.3984...	1.9969...
EC1-ext.jpg	-18.16...	-8.0527...	1.3630...
EC1-Bar2.jpg	-1.382...	-0.5212...	-3.135...
EC1-Bar3.jpg	3.8091...	-5.0284...	2.1694...
ec1-pool.jpg	-4.891...	7.1742...	-5.182...

Adswerve and Twiddy & Co: train model and predict listing clickthrough rate, all in SQL!

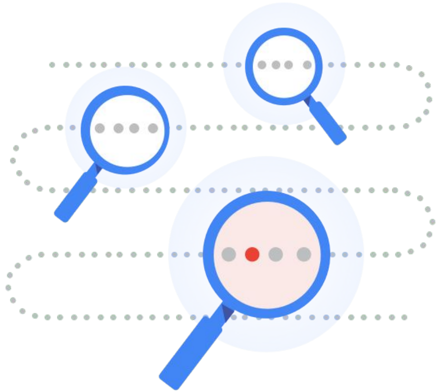


BigSearch on unstructured data inferences

1. Build search indexes on metadata generated from inference. Such as text from PDF docs, objects and entities from images or videos, or speech transcription

2. Run ‘needle in the haystack’ queries for search use cases. Tightly integrated with native JSON, allowing you to get BigQuery performance and storage optimizations on JSON

3. Get signed URLs for search results to retrieve objects from GCS. Use it in ad hoc queries or BI reports



```sql
CREATE SEARCH INDEX my_index ON
pdf_text_extract(ALL COLUMNS);
```

```sql
SELECT * FROM pdf_text_extract WHERE
SEARCH(pdf_text, "Google");
```

Secure and govern cloud storage using row-level security

1. Rows in Object tables map to objects on Google Cloud Storage

2. Users will be able to access signed URLs only for the objects for which you grant them row level access in Object table

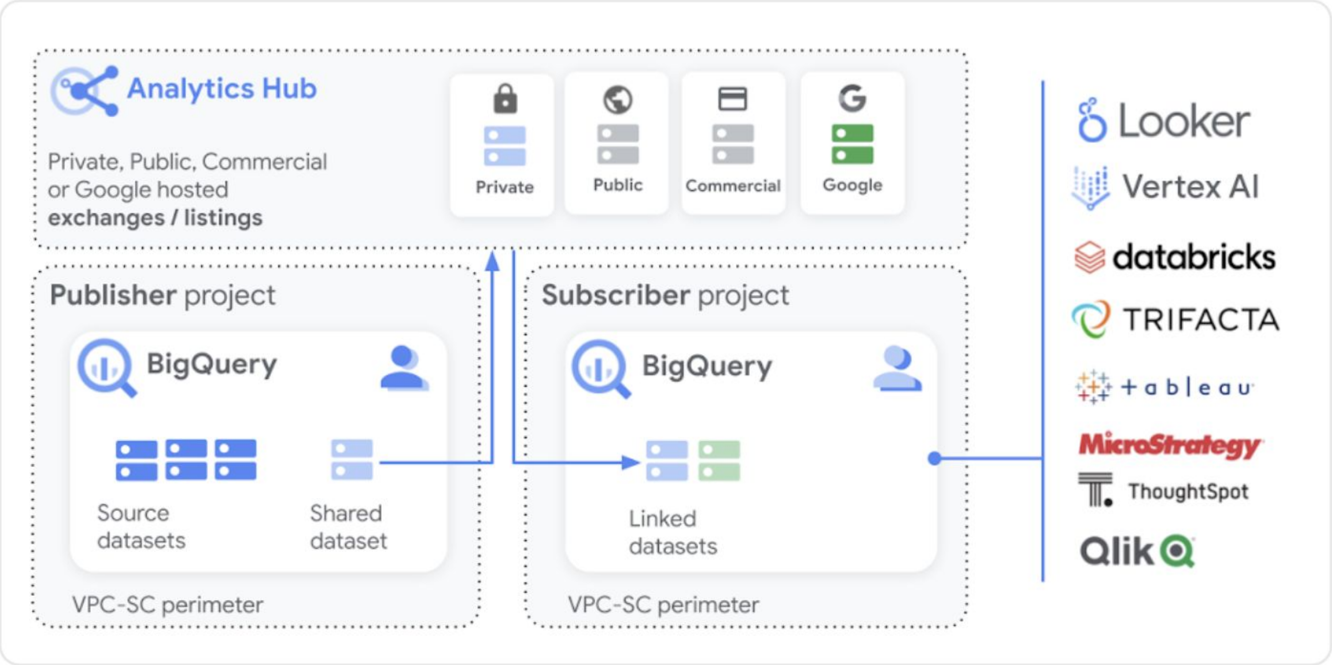
3. Specify these fine-grained access policies using object metadata (e.g., *from structured inference*) for use cases such PII management.

```
CREATE ROW ACCESS POLICY pii_data ON
object_table_images

GRANT TO ("group:admin@example.com")
FILTER USING (

metadata[OFFSET(0)].name="face_detected")
```

Share unstructured + governed data sets

Sharing use case

- Share object tables using analytics hub
- Consumer will only be able to access signed URLs for objects that are shared with them (via rows on object tables)



Open Challenges / Directions

- **Systems-level challenges**
  - Scaling inference within a relational query processing engine
    - Reuse HPC techniques to minimize communication overhead?
  - Heterogeneous query processing
    - Generalized planning combining, e.g., tensor operators and traditional relational algebra
  - Mix and match of hardware profiles under one roof?
    - “Push up” operators in a query plan
- **Usability / experience challenges**
  - Asynchrony in SQL
  - SQL is great, but is it the only interface? (Barbara’s talk on Tuesday)
  - Will we build verticals? Or are there common abstractions?

Questions