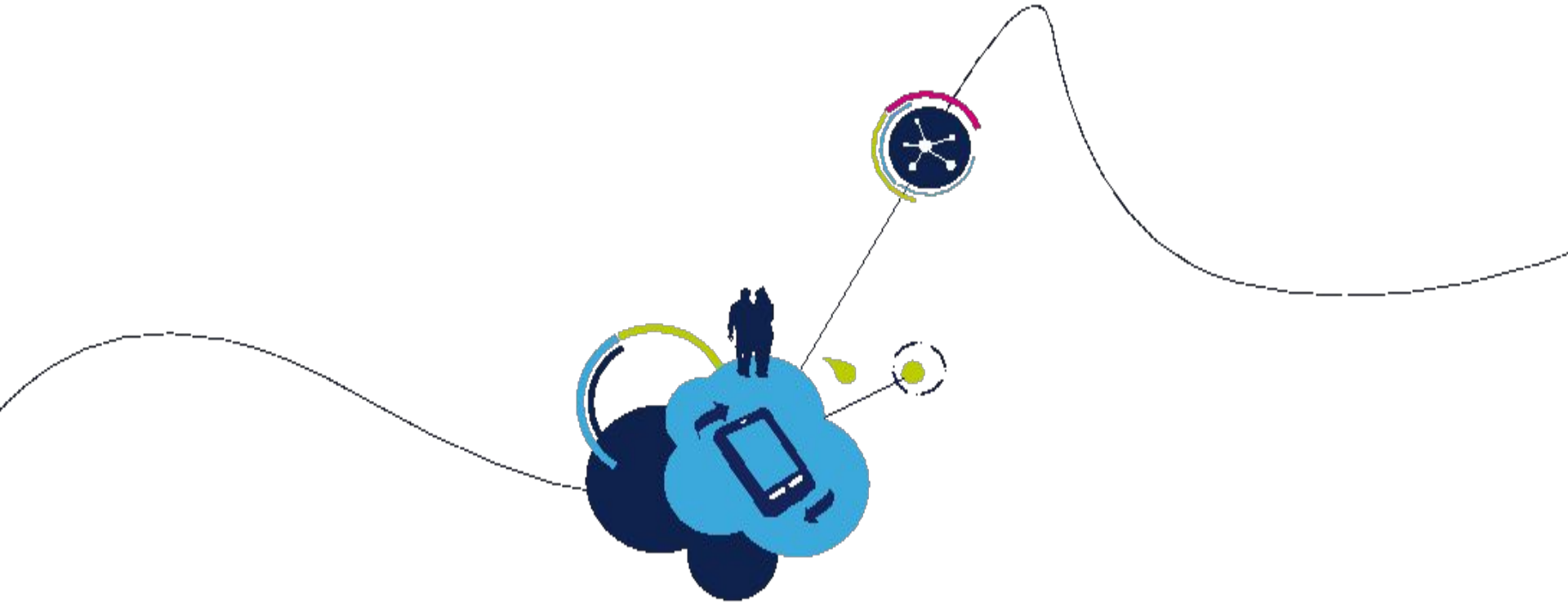
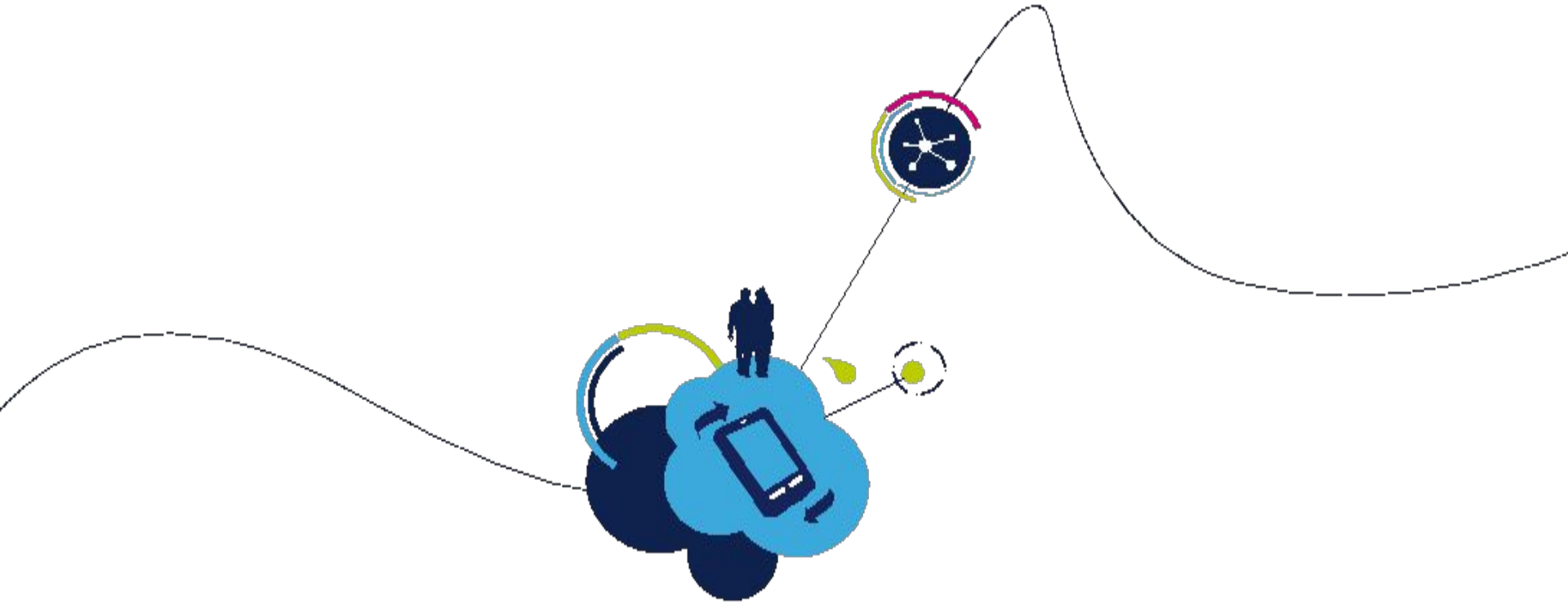


GCP for Data Engineers

Agenda

1. **Data Processing Fundamentals**
2. **Big Data Ecosystem**
3. **Real Time Messaging with Pub/Sub**
4. **Pipelines with Cloud Dataflow**
5. **Managed Spark with Cloud Dataproc**
6. **NoSQL Data with Cloud Bigtable**
7. **Data Analytics with BigQuery**
8. **Exploration with Cloud Datalab**
9. **Visualization with Cloud Data Studio**
10. **Orchestration with Cloud Composer**



Data Processing Fundamentals

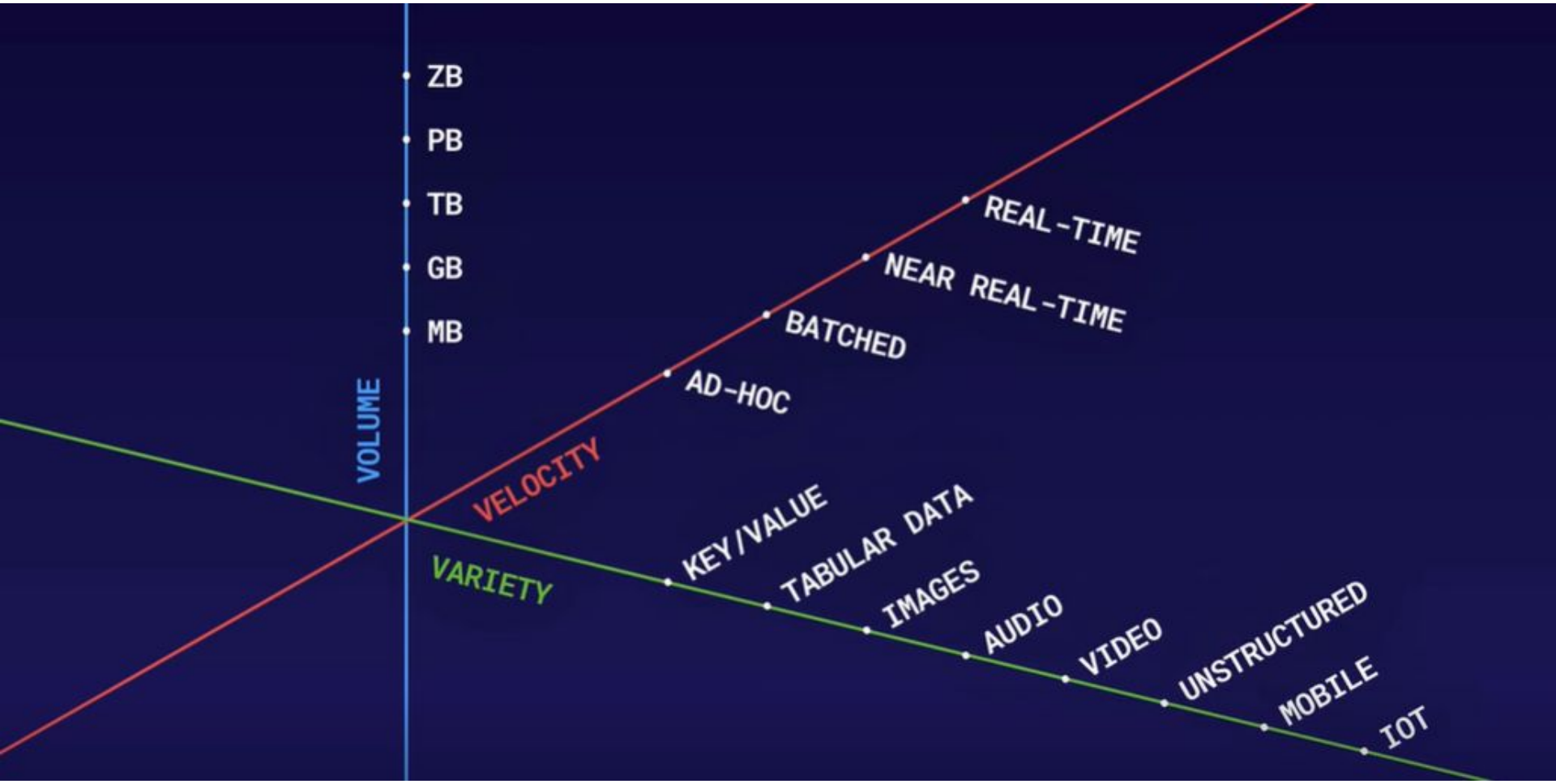
What is BigData?

Large Datasets

Technology

What is BigData?

What is BigData?



Defining BigData Terms

Data Warehouses and Data Lakes

OLTP vs OLAP

SQL vs NoSQL

Batch vs Streaming

Defining BigData Terms

Data Warehouse

1. **Structured and/or Processed**

   Data is organized, may have been transformed, and is stored in a structured way.

2. **Ready to Use**

   Data exists in the warehouse for a defined purpose, and in a format where it is ready to be consumed.

3. **Rigid**

   Data may be easier to understand, but less up-to-date. Structures are hard to change.

What is BigData?

Data Lake

1. **Raw and/or Unstructured**

   The data lake contains all raw, unprocessed data, before any kind of transformation or organization.

2. **Ready to Analyze**

   Data is more up to date, but may require more advanced tools for analysis.

3. **Flexible**

   No structure is enforced, so new types of data can be added at any time.

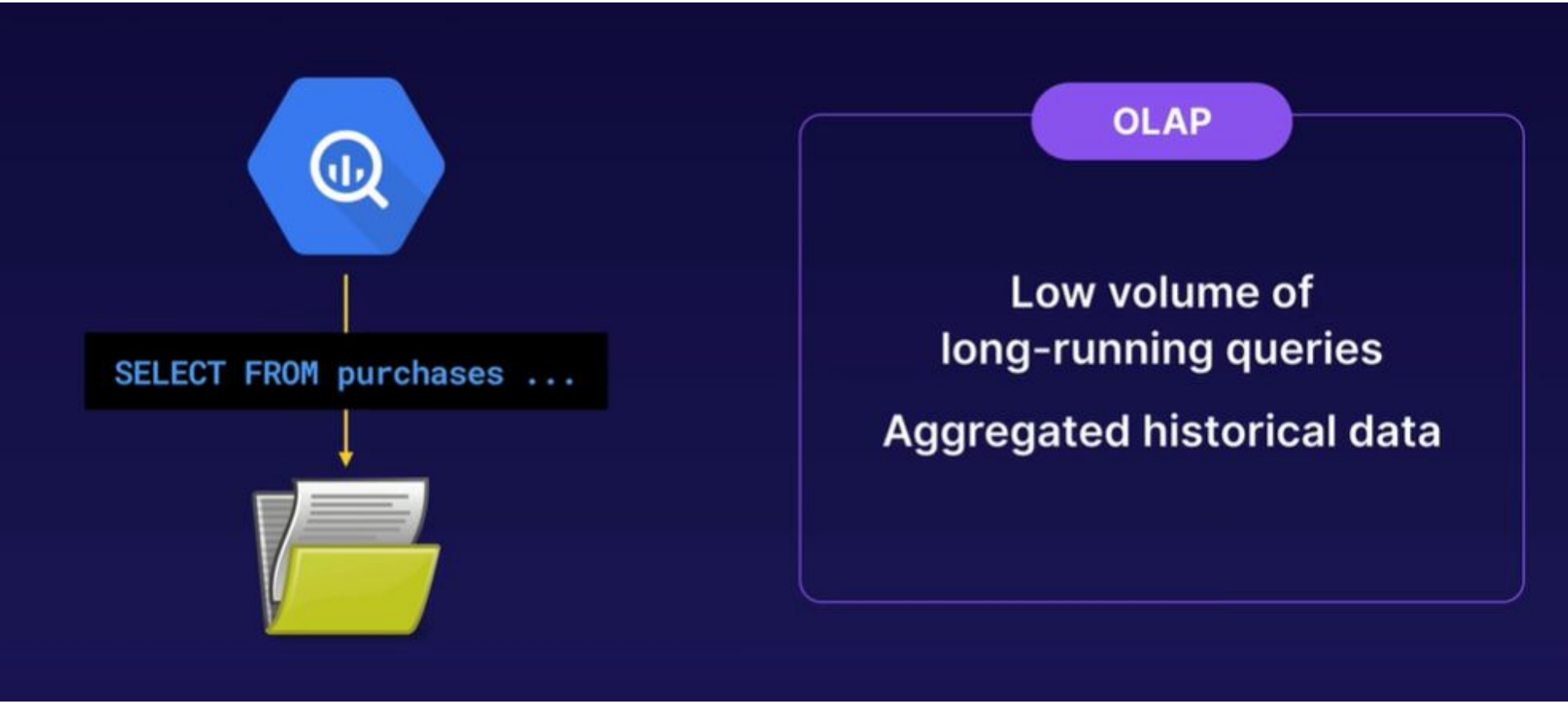
OLAP vs OLTP


On-Line Transactional Processing
On-Line Analytical Processing

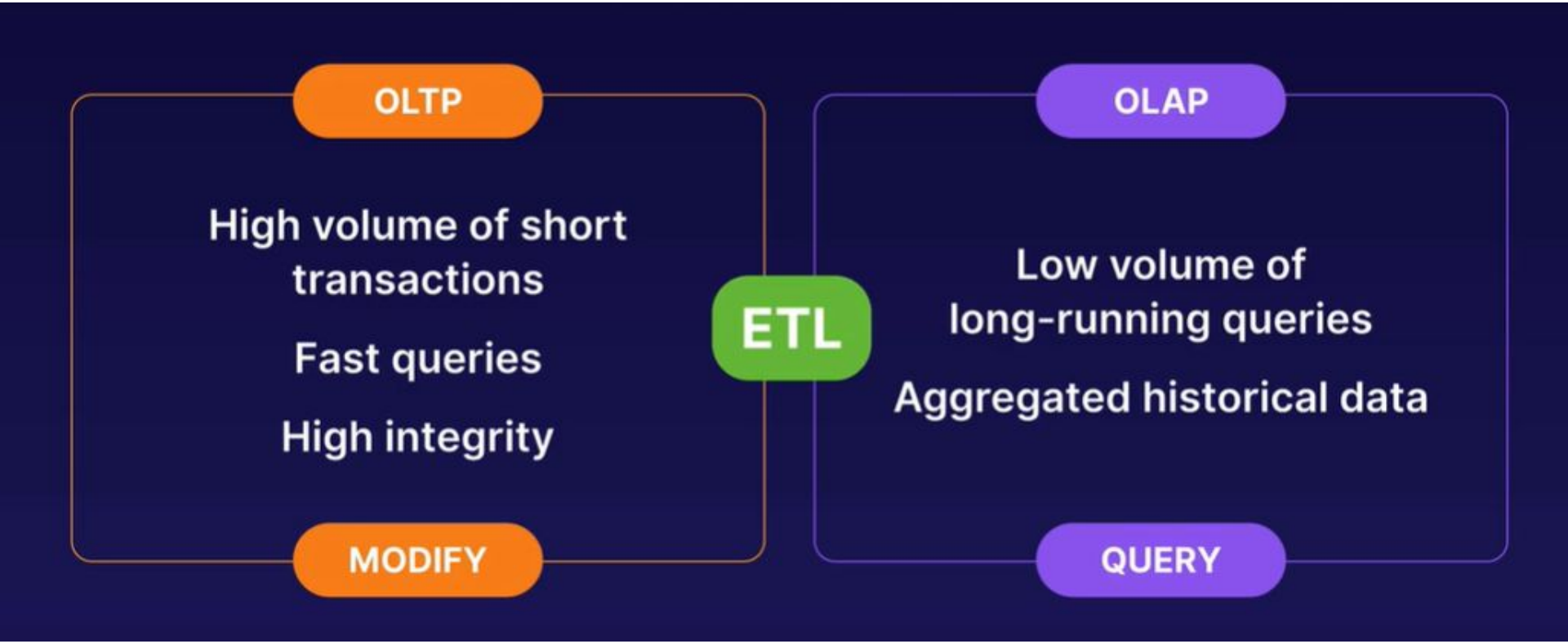
OLAP vs OLTP


OLTP
High volume of short transactions
Fast queries
High integrity
INSERT INTO purchases ...

OLAP vs OLTP


SELECT FROM purchases ...
OLAP
Low volume of
long-running queries
Aggregated historical data

OLAP vs OLTP

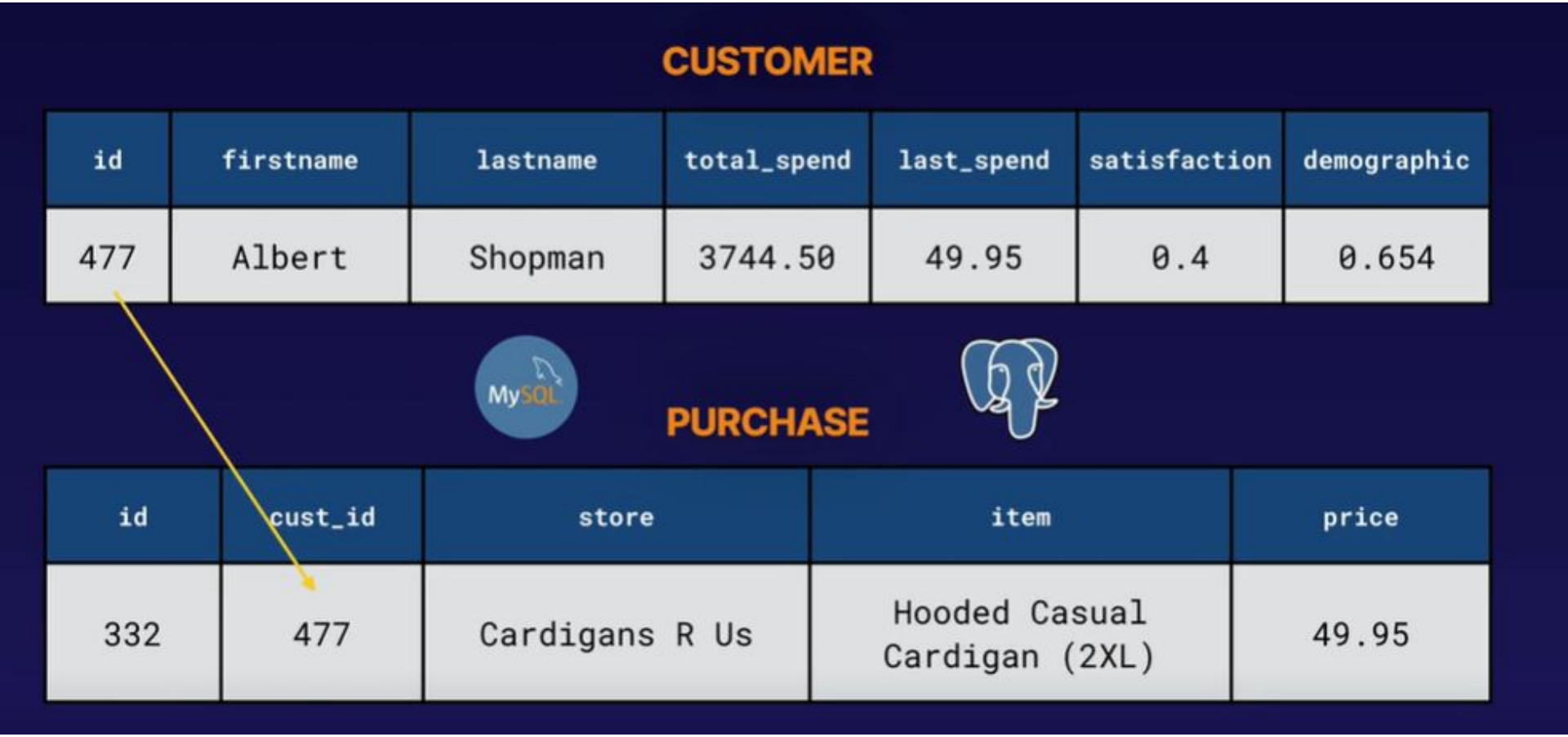

OLTP
High volume of short transactions
Fast queries
High integrity
MODIFY
ETL
OLAP
Low volume of long-running queries
Aggregated historical data
QUERY

SQL Databases

CUSTOMER

id	firstname	lastname	total_spend	last_spend	satisfaction	demographic
477	Albert	Shopman	3744.50	49.95	0.4	0.654

SQL Databases



NoSQL Databases

mongoDB.

cassandra

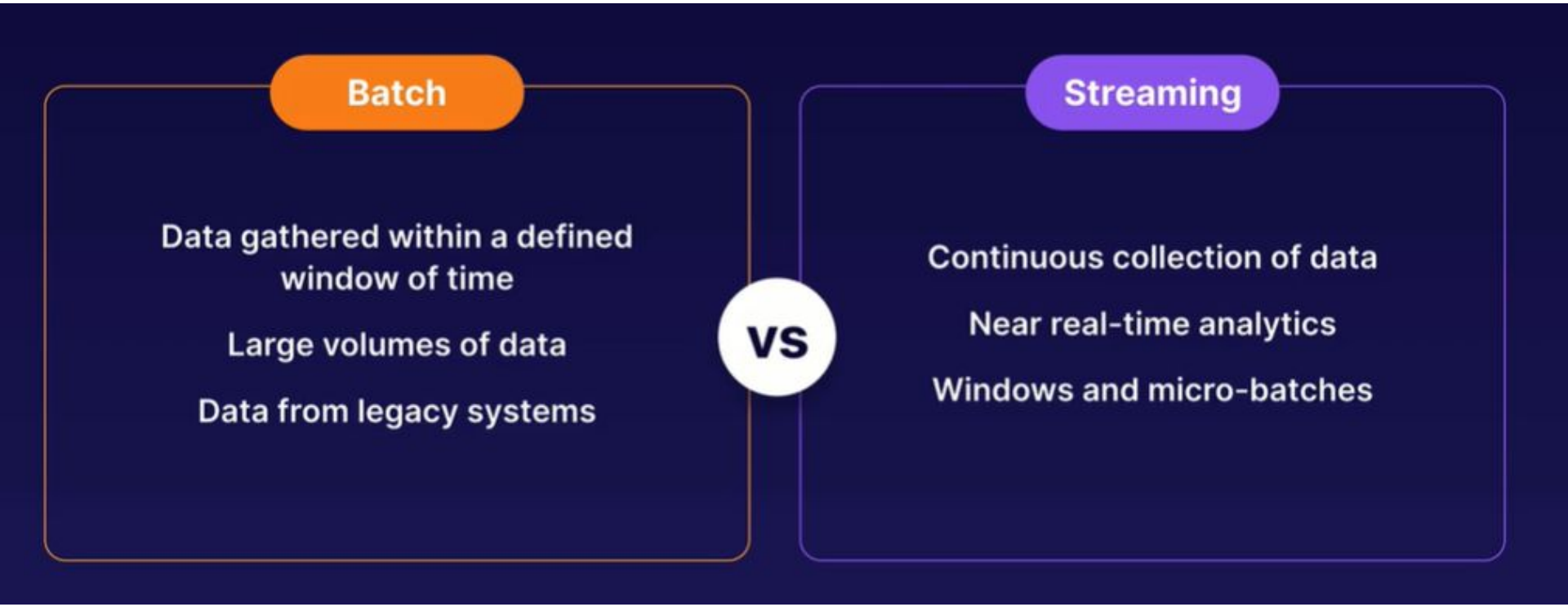
KEY VALUE STORES

```
total_purchases = 3
checked_out     = True
most_recent     = "Cardigan"
```

JSON DOCUMENT STORES

```
{ "name": "Albert Shopman",
  "age": 56,
  "favorite_shops": ["Cardigans R Us",
                     "Frank's Beard Supplies",
                     "Whole Foods"]
}
```

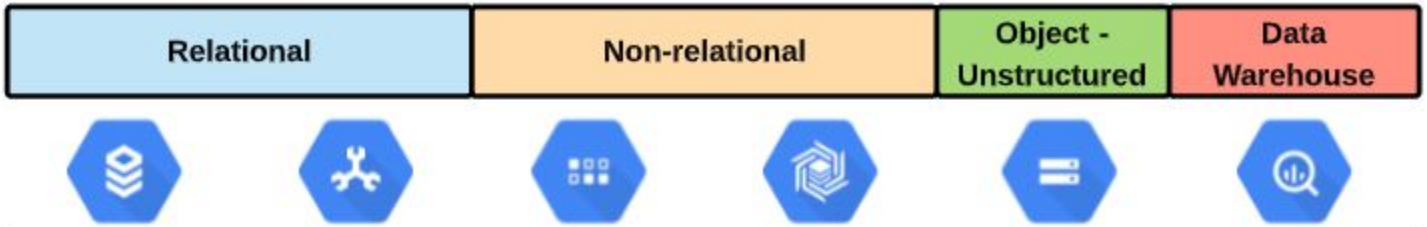
Batch vs Streaming



Choosing a Managed Database

Big picture perspective:

- At minimum, know which managed database is the best solution for any given use case:
  - Relational, non-relational?
  - Transactional, analytics?
  - Scalability?
  - Lift and shift?



	Relational		Non-relational		Object - Unstructured	Data Warehouse
	Cloud SQL	Cloud Spanner	Cloud Datastore	Cloud Bigtable	Cloud Storage	BigQuery
Use Case	Structured data Web framework	RDBMS+scale High transactions	Semi-structured Key-value data	High throughput analytics	Unstructured data Holds everything	Mission critical apps Scale+consistency
e.g.	Medical records Blogs	Global supply chain Retail	Product catalog Game state	Graphs IoT Finance	Multimedia Analytics Disaster recovery	Large data analytics Processing using SQL

Choosing a Managed Database

Decision tree criteria:

- **Structured (database) or unstructured?**
- **Analytical or transactional?**
- **Relational (SQL) or non-relational (NoSQL)?**
- **Scalability/availability/size requirements?**

Cloud SQL Basics

What is Cloud SQL?

- Direct lift and shift of traditional MySQL/PostgreSQL workloads with the maintenance stack managed for you

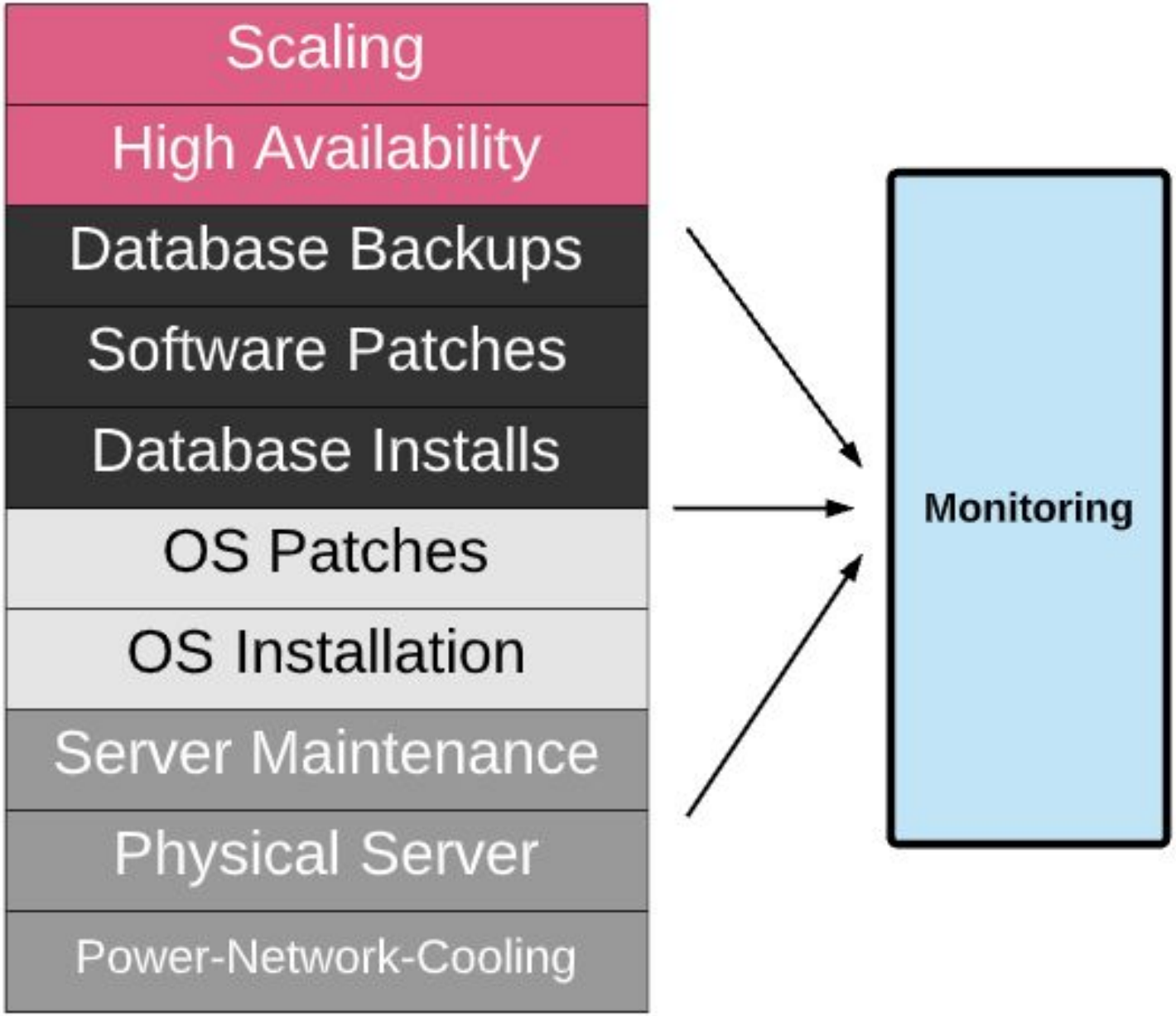
What is managed?

- OS installation/management
- Database installation/management
- Backups
- Scaling - disk space
- Availability:
  - Failover
  - Read replicas
- Monitoring
- Authorize network connections/proxy/use SSL

Limitations:

- Read replicas limited to the same region as the master:
  - Limited global availability
- Max disk size of 10 TB
- If > 10 TB is needed, or global availability in RDBMS, use Spanner

Cloud SQL Basics



Importing Data

Importing data into Cloud SQL:

- Cloud Storage as a staging ground
- SQL dump/CSV file format

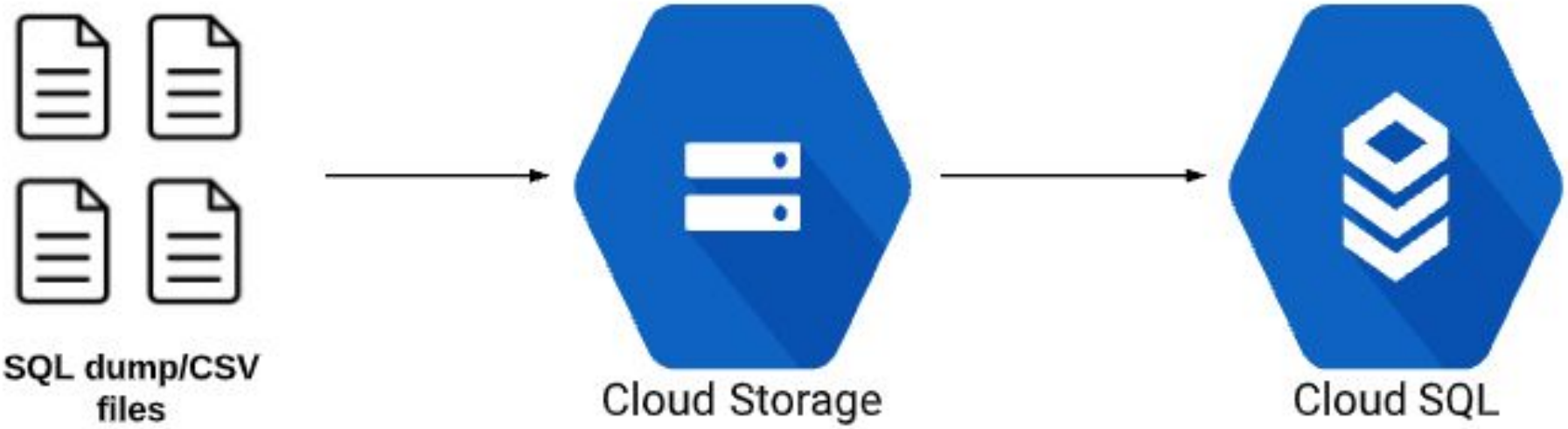
Export/Import process:

- Export SQL dump/CSV file:
  - SQL dump file cannot contain triggers, views, stored procedures
- Get dump/CSV file into Cloud Storage
- Import from Cloud Storage into Cloud SQL instance

Best Practices:

- Use correct flags for dump file (--'flag_name'):
  - Databases, hex-blob, skip-triggers, set-gtid-purged=OFF, ignore-table
- Compress data to reduce costs:
  - Cloud SQL can import compressed .gz files
- Use InnoDB for Second Generation instances

Importing Data



SQL Query Best Practices

SQL Query Best Practices

General SQL efficiency best practices:

- More, smaller tables better than fewer, large tables:
  - Normalization of tables
- Define your SELECT fields instead of using SELECT *:
  - SELECT * acts as a 'select all'
- When joining tables, use INNER JOIN instead of WHERE:
  - WHERE creates more variable combinations = more work

Cloud Datastore Overview

What is Cloud Datastore?

- No Ops:
  - No provisioning of instances, compute, storage, etc.
  - Compute layer is abstracted away
- Highly scalable:
  - Multi-region access available
  - Sharding/replication handled automatically
- NoSQL/non-relational database:
  - Flexible structure/relationship between objects

Use Datastore for:

- Applications that need highly available structured data, at scale
- Product catalogs - real-time inventory
- User profiles - mobile apps
- Game save states
- ACID transactions - e.g., transferring funds between accounts


Cloud Datastore

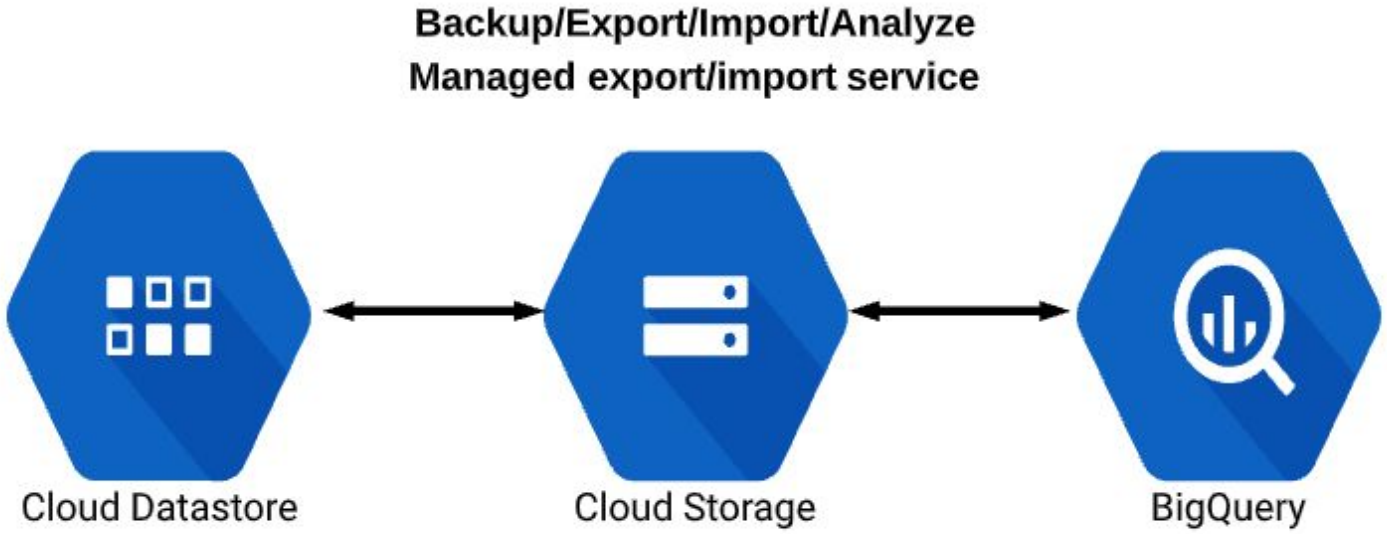
Do not use Datastore for:

- Analytics (full SQL semantics):
  - Use BigQuery/Cloud Spanner
- Extreme scale (10M+ read/writes per second):
  - Use Bigtable
- Don't need ACID transactions/data not highly structured:
  - Use Bigtable
- Lift and shift (existing MySQL):
  - Use Cloud SQL
- Near zero latency (sub-10ms):
  - Use in-memory database (Redis)

Cloud Datastore Overview

Other important facts:

- Single Datastore database per project
- Multi-regional for wide access, single region for lower latency and for single location
- Datastore is a transactional database
- Bigtable is an analytical database
- IAM roles:
  - Primitive and predefined
  - Owner, user, viewer, import/export admin, index admin



Cloud Datastore Overview

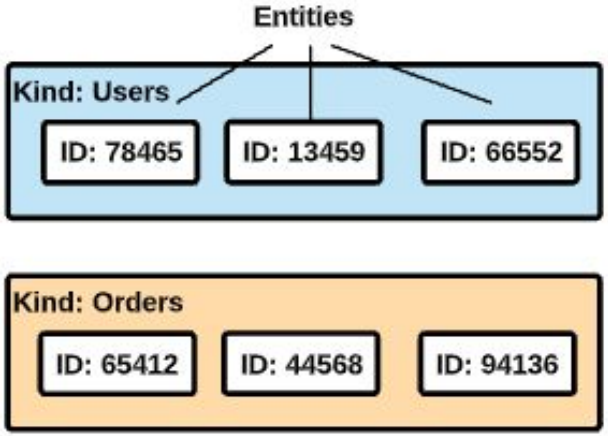
Short version:

- **Entities grouped by kind (category)**
- **Entities can be hierarchical (nested)**
- **Each entity has one or more properties**
- **Properties have a value assigned**

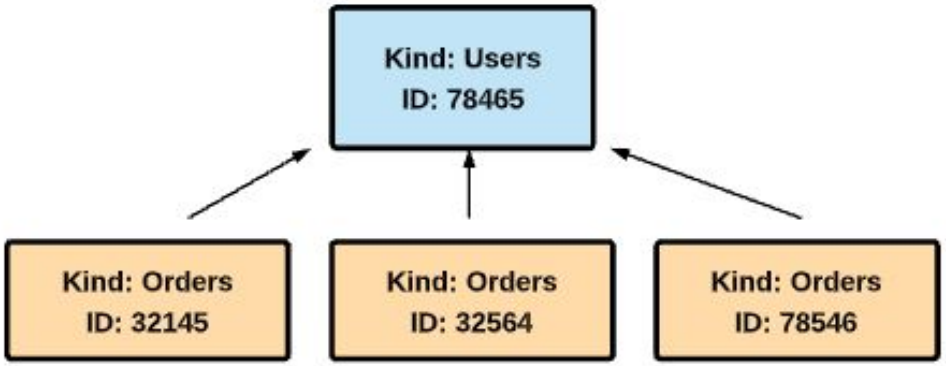
Concept	Relational Database	Datastore
Category of object	Table	Kind
Single Object	Row	Entity
Individual data for an object	Column	Property
Unique ID for an object	Primary key	Key

Data Organization

Simple Collections of Entities



Hierarchies (Entity Groups)



Queries and Indexing

Query:

- Retrieve an entity from Datastore that meets a set of conditions
- Query includes:
  - Entity kind
  - Filters
  - Sort order
- Query methods:
  - Programmatic
  - Web console
  - Google Query Language (GQL)

```
indexes:
- kind: Task
  properties:
  - name: tags
  - name: created
- kind: Task
  properties:
  - name: collaborators
  - name: created
```

Indexing:

- Queries gets results from indexes:
  - Contain entity keys specified by index properties
  - Updated to reflect changes
  - Correct query results available with no additional computation needed

Index types:

- Built-in - default option:
  - Allows single property queries
- Composite - specified with an index configuration file (index.yaml):
  - gcloud datastore create-indexes index.yaml

Data Consistency

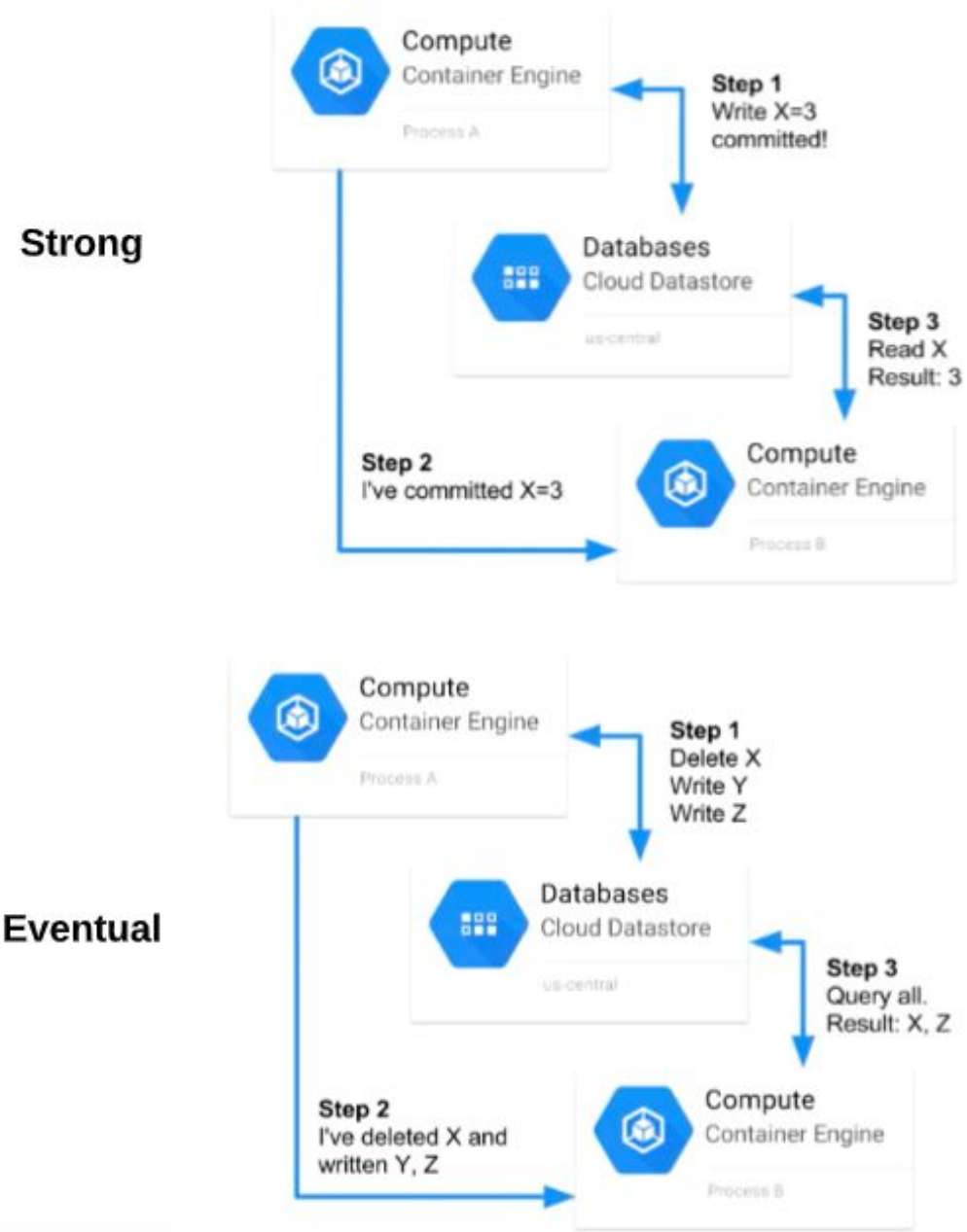
What is data consistency in queries?

- "How up to date are these results?"
- "Does the order matter?"
- Strongly consistent = Parallel processes see changes in same order:
  - Query is guaranteed up to date but may take longer to complete
- Eventually consistent = Parallel process can see changes out of order, will eventually see accurate end state:
  - Faster query, but may *sometimes* return stale results
- Performance vs. accuracy
- Ancestor query/key-value operations = strong
- Global queries/projections = eventual

Use cases:

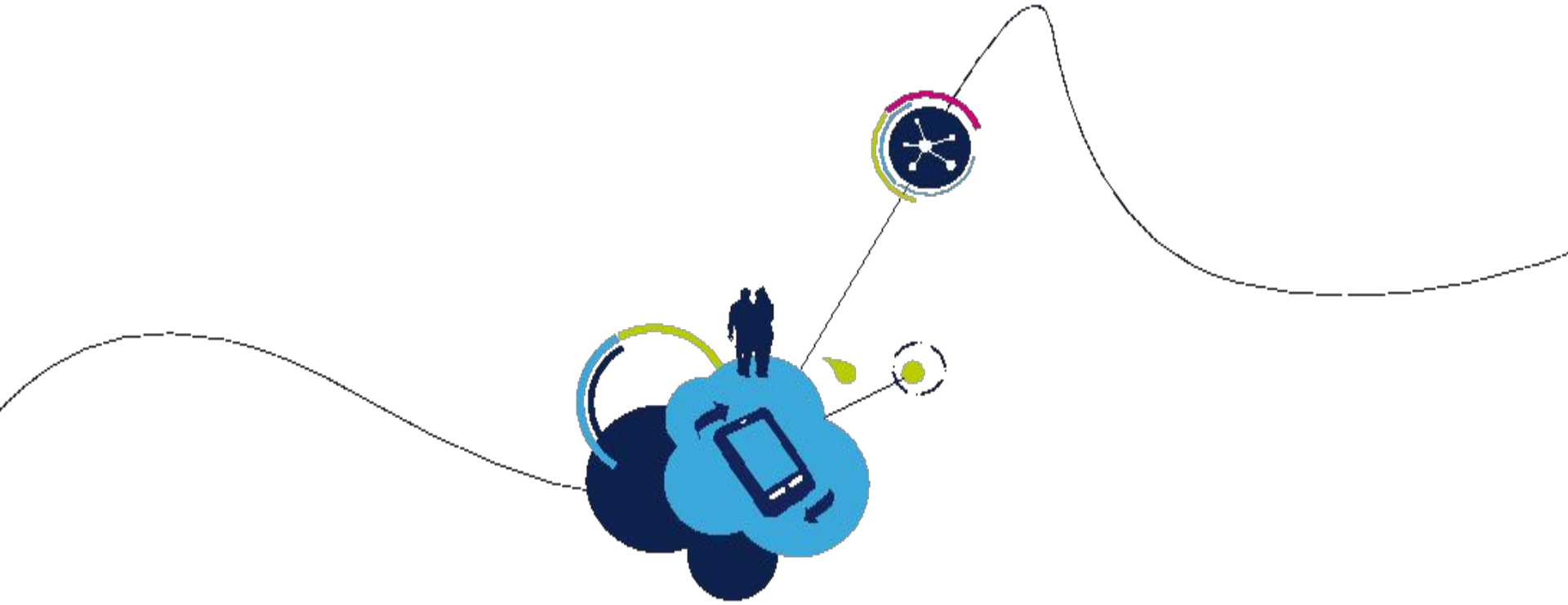
- Strong - financial transaction:
  - Make deposit -- check balance
- Eventual - census population:
  - Order not as important, as long as you get eventual result

Data Consistency



What’s next?

Ingestion > Storage > Processing > Visualization

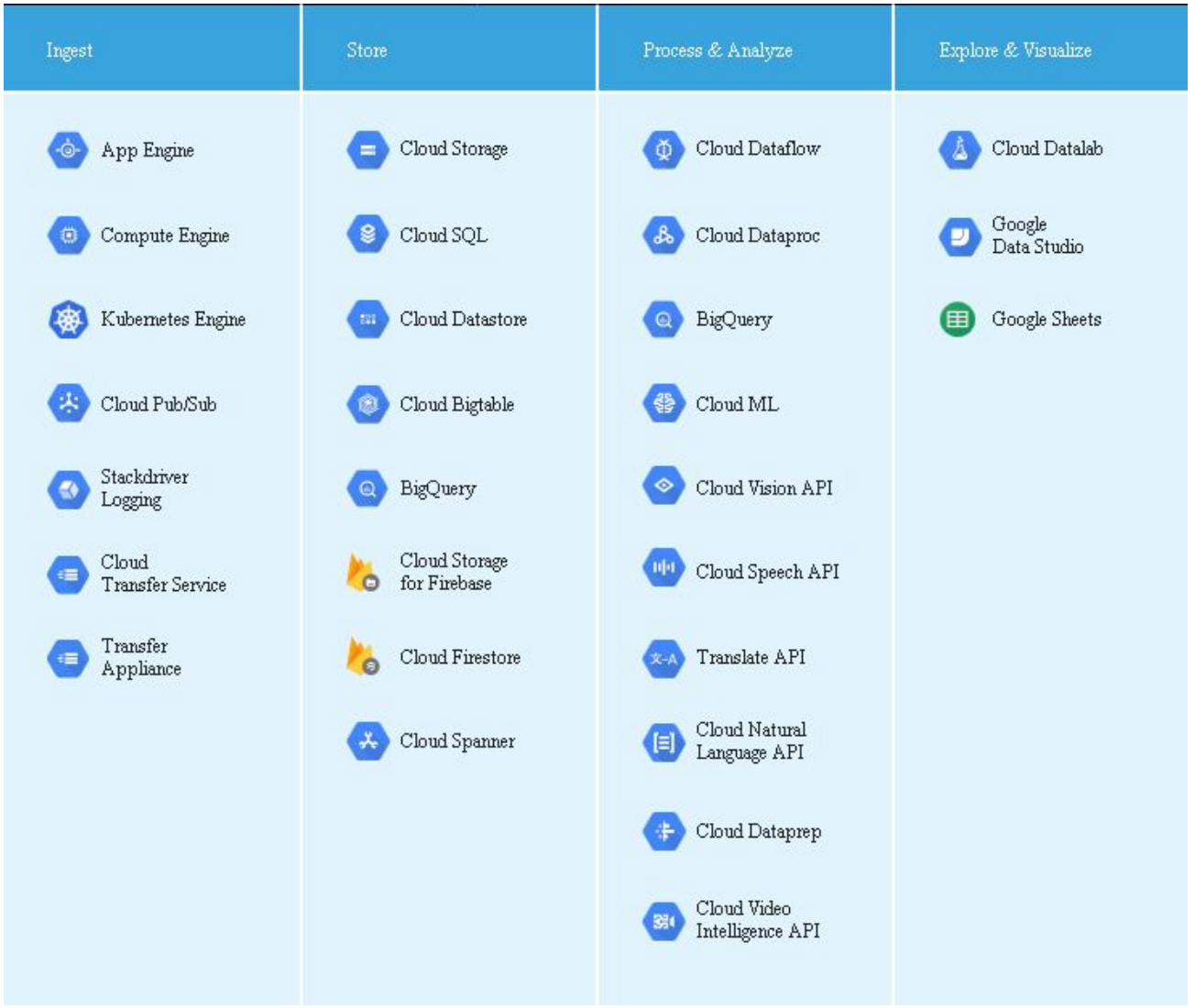


Big Data Ecosystem

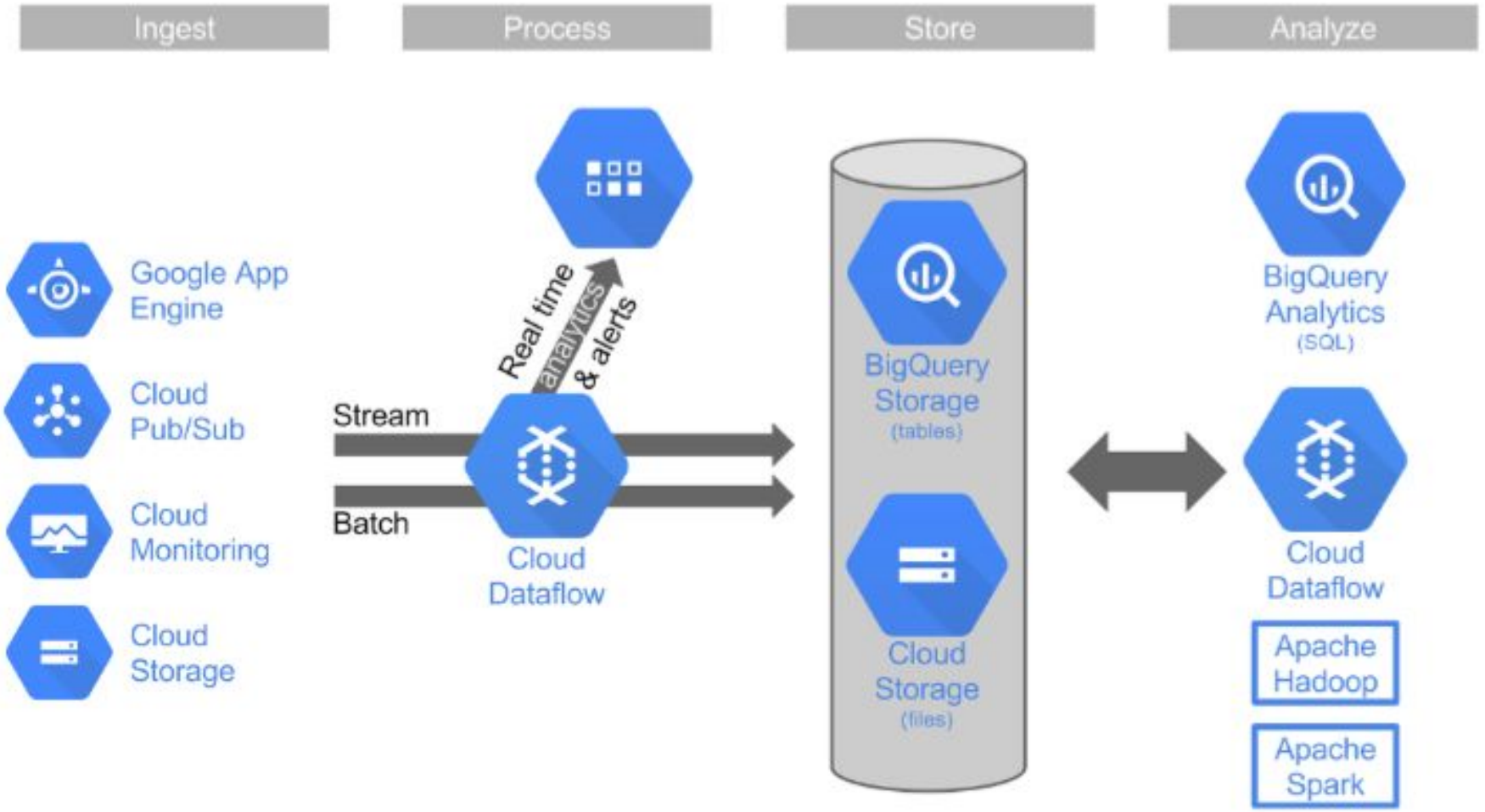
Data lifecycle

- Think of data as a tangible object to be collected, stored, and processed
- Lifecycle from initial collection to final visualization
- Needs to be familiar with the lifecycle steps, what GCP services are associated with each step, and how they connect together
- Data Lifecycle steps:
  - Ingest - Pull in the raw data:
    - Streaming/real-time data from devices
    - On-premises batch data
    - Application logs
    - Mobile-app user events and analytics
  - Store - data needs to be stored in a format and location that is both reliable and accessible
  - Process and analyze - Where the magic happens. Transform data from raw format to actionable information
  - Explore and visualize - "Make it look pretty"
    - The final stage is to convert the results of the analysis into a format that is easy to draw insights from and to share with colleagues and peers

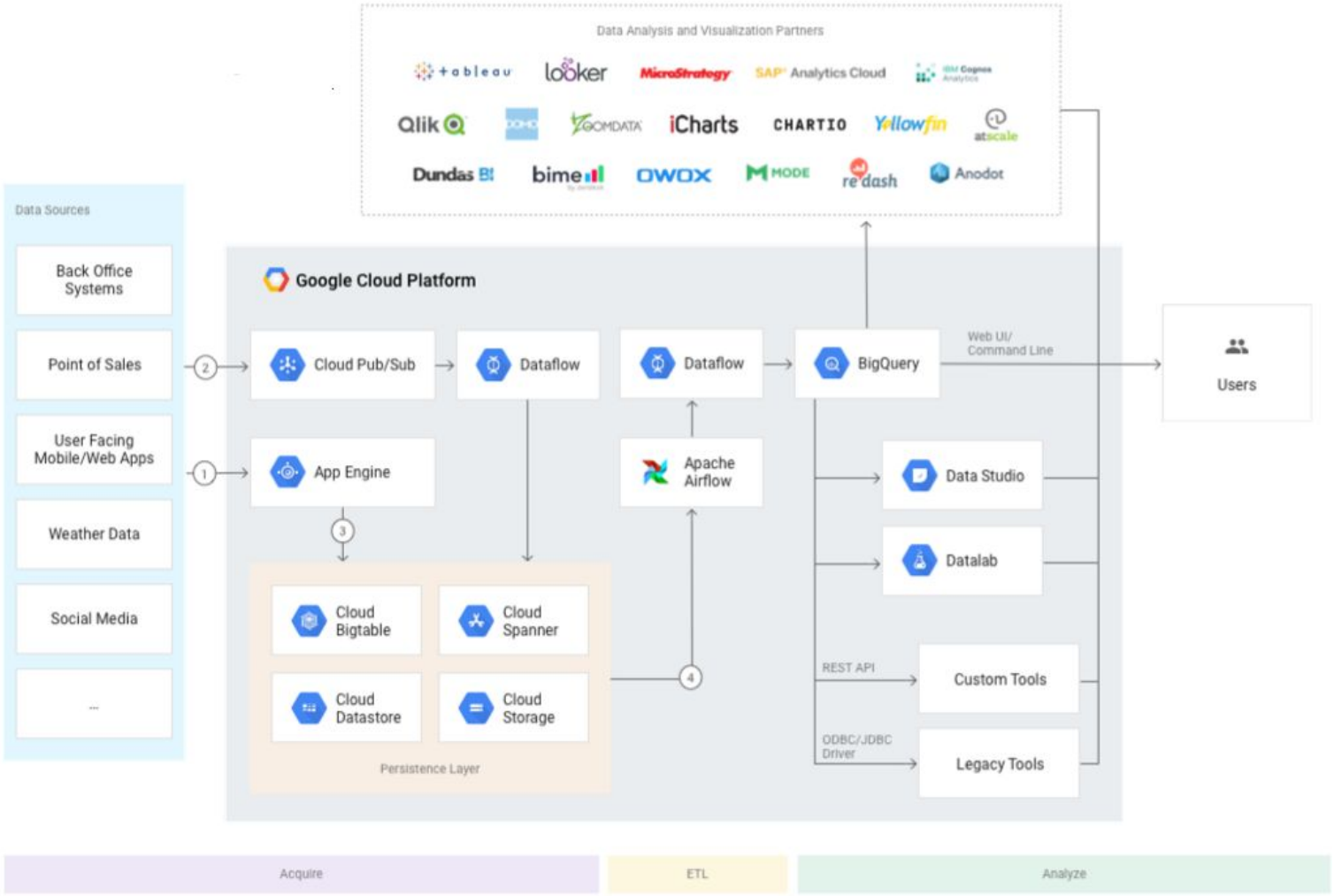
Data lifecycle and associated services



Data Lifecycle is not a Set Order



Increasing Complexity of Data Flow

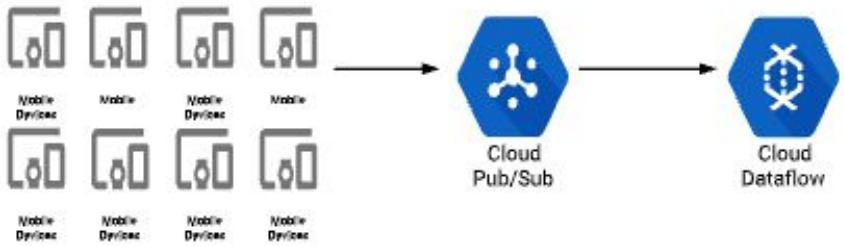


Streaming and Batch Data

Data Lifecycle = Data Ingest

Streaming (or real-time) data:

- Generated and transmitted continuously by many data sources
- Thousands of data inputs, sent simultaneously, in small sizes (KB)
- Commonly used for telemetry - collecting data from a high number of geographically dispersed devices as it's generated
- Examples:
  - Sensors in transportation vehicles - detecting performance and potential issues
  - Financial institution tracks stock market changes
- Data is processed in small pieces as it comes in
- Requires low latency
- Typically paired with Pub/Sub for the streaming data ingest and Dataflow for real-time processing



Batch (or bulk) data:

- Large sets of data that 'pool' up over time
- Transferring from a small number of sources (usually 1)
- Examples:
  - On-premise database migration to GCP
  - Importing legacy data into Cloud Storage
  - Importing large datasets for machine learning analysis
  - gsutil cp [storage_location] gs://[BUCKET] is an example of batch data import
- Low latency is not as important
- Often stored in storage services such as cloud storage, CloudSQL, BigQuery, etc.

Cloud Storage as Staging Ground

Cloud Storage

Storage 'swiss army knife':

- GCS holds all data types:
  - All database transfer types, raw data, any format
- Globally available:
  - Multi-regional buckets provide fast access across regions
  - Regional buckets provide fast access for single regions
  - Edge caching for increased performance
- Durable and reliable:
  - Versioning and redundancy
- Lower cost than persistent disk
- Control access:
  - Project, bucket, or object level
  - Useful for ingest, transform, and publish workflows
  - Option for Public read access

Data Engineering perspective:

- Migrating existing workloads:
  - Migrate databases/data into Cloud Storage for import
- Common first step of data lifecycle - get data to GCS
- Staging area for analysis/processing/machine learning import:
  - 'Data lake'

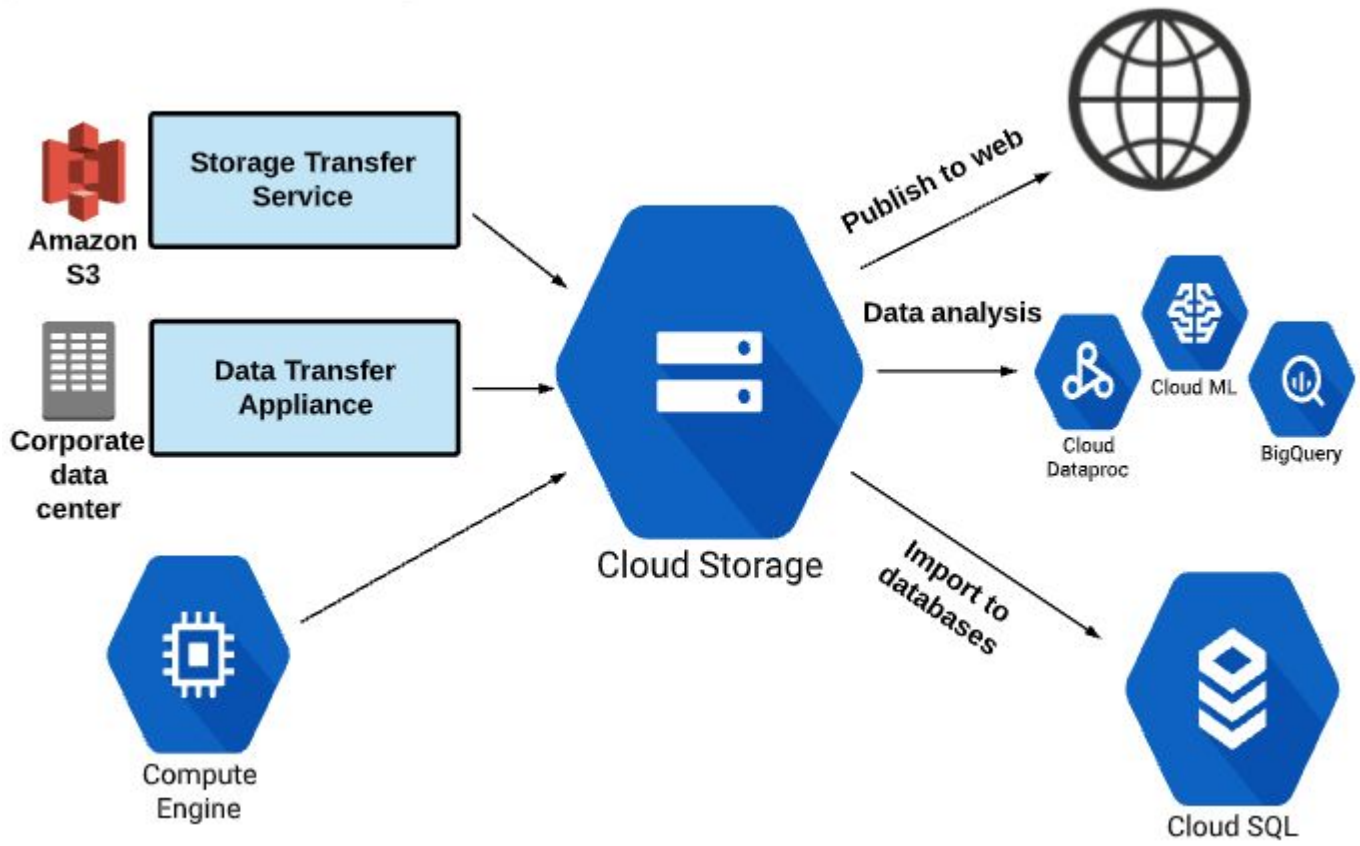
Getting data in and out of Cloud Storage

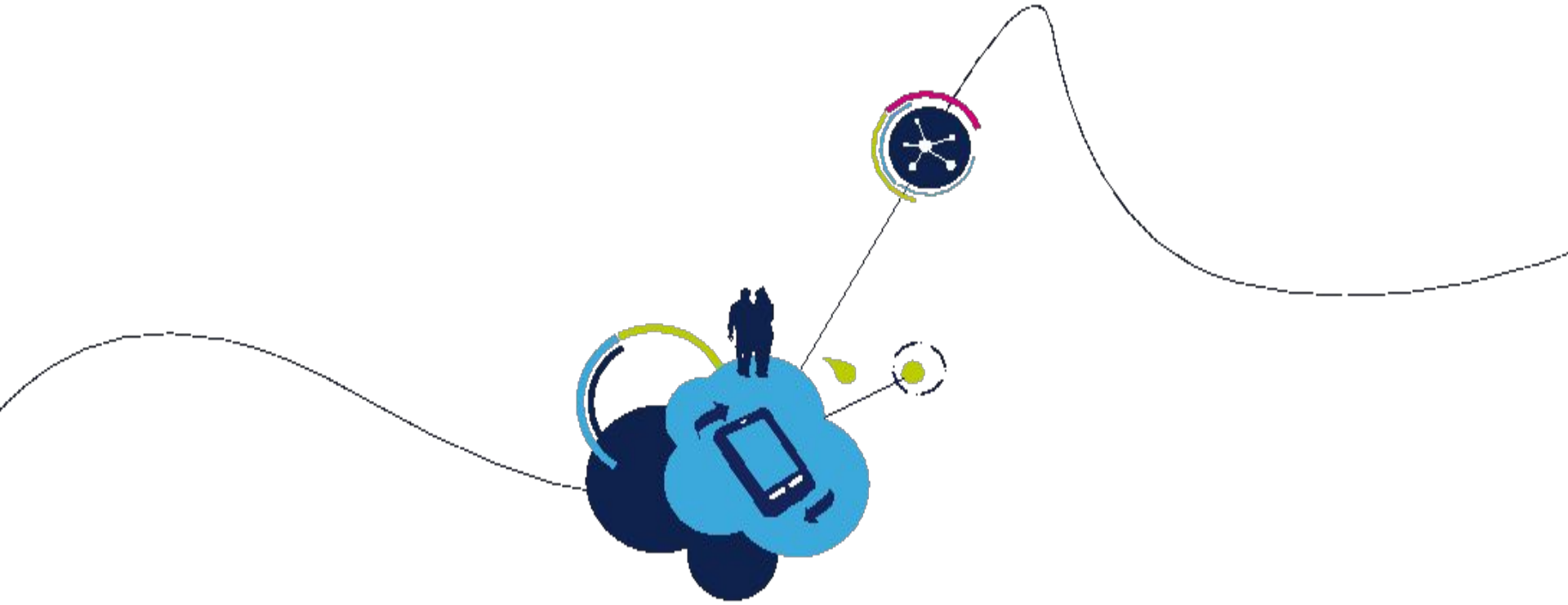
Storage Transfer Service - S3, GCS, HTTP --> GCS:

- One time transfer, periodic sync

Data Transfer Appliance - physically shipped appliance:

- Load up to 1 petabyte, ship to GCP, loaded into bucket
- gsutil, JSON API - "gsutil cp ..."





Real Time Messaging with

Pub/Sub

Streaming Data Challenges

What is Streaming Data?

- "Unbounded" data
- Infinite, never completes, always flowing

Examples

Traffic Sensors

Credit Card Transactions

Mobile Gaming

Fast action is often necessary

- Quickly collect data, gain insights, and take action
- Sending to storage can add latency
- Use cases:
  - Credit card fraud detection
  - Predicting highway traffic

Streaming Data Challenges

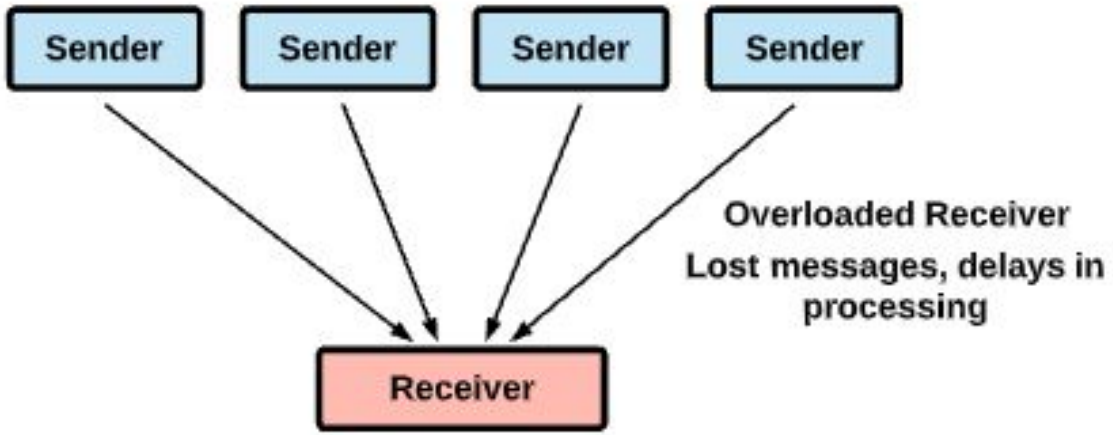
Tight vs. Loose Coupling in Systems

- Tightly (direct) coupled systems more likely to fail
- Loosely coupled systems with 'buffer' scale have better fault tolerance

Tight vs. Loose Coupling in Systems

- Tightly (direct) coupled systems more likely to fail
- Loosely coupled systems with 'buffer' scale have better fault tolerance

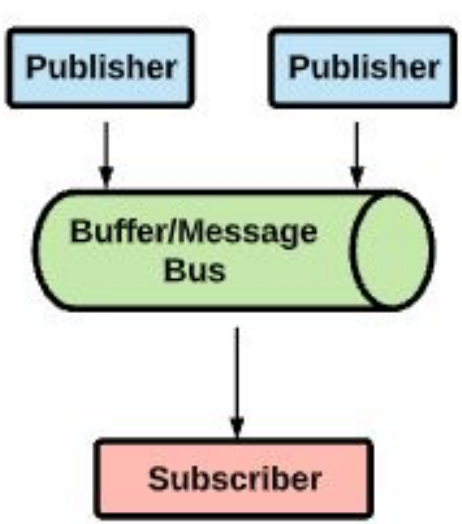
Tightly-Coupled System

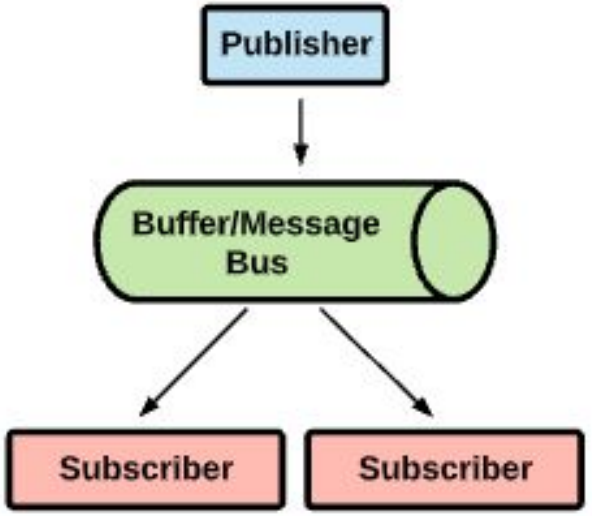


Streaming Data Challenges

Loosely-Coupled System

- **Fault tolerance**
- **Scalability**
- **Message queuing**



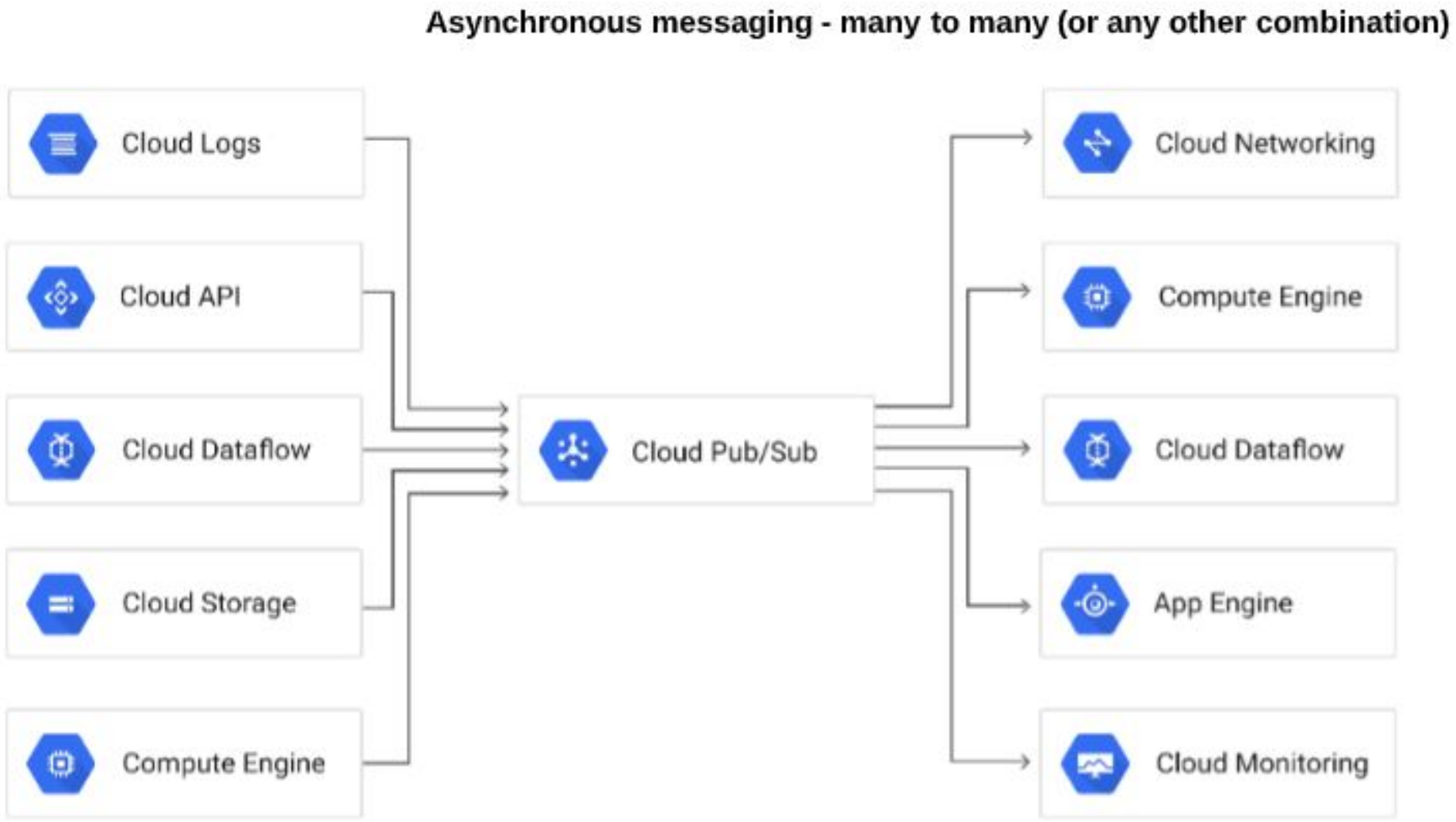


Cloud Pub/Sub Overview

What is Cloud Pub/Sub?

- Global-scale messaging buffer/coupler
- NoOps, global availability, auto-scaling
- Decouples senders and receivers
- Streaming data ingest:
  - Also connects other data pipeline services
- Equivalent to Apache Kafka (open source)
- Guaranteed at-least-once delivery

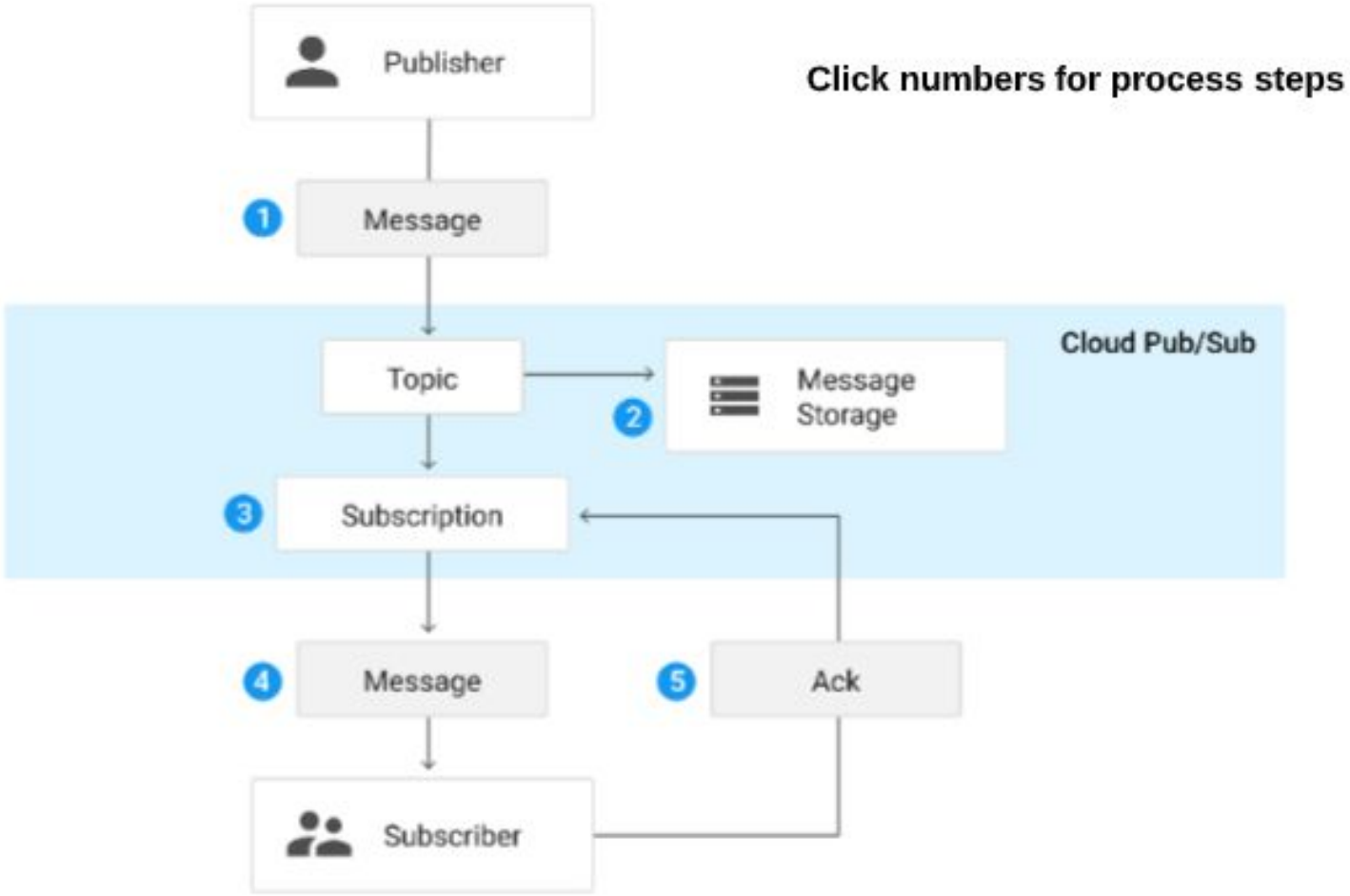
Cloud Pub/Sub Overview



Streaming Data Challenges

How It Works: Terminology

- **Topics, Messages, Publishers, Subscribers, Message Store**



Streaming Data Challenges

Push and Pull

- Pub/Sub can either *push* messages to subscribers, or subscribers can *pull* messages from Pub/Sub.
- Push = lower latency, more real-time.
- Push subscribers must be Webhook endpoints that accept POST over HTTPS.
- Pull is ideal for large volume of messages, and uses batch delivery

IAM

- Allows for controlling access at project, topic, or subscription level
- Admin, Editor, Publisher, Subscriber
- Service accounts are best practice

Pricing

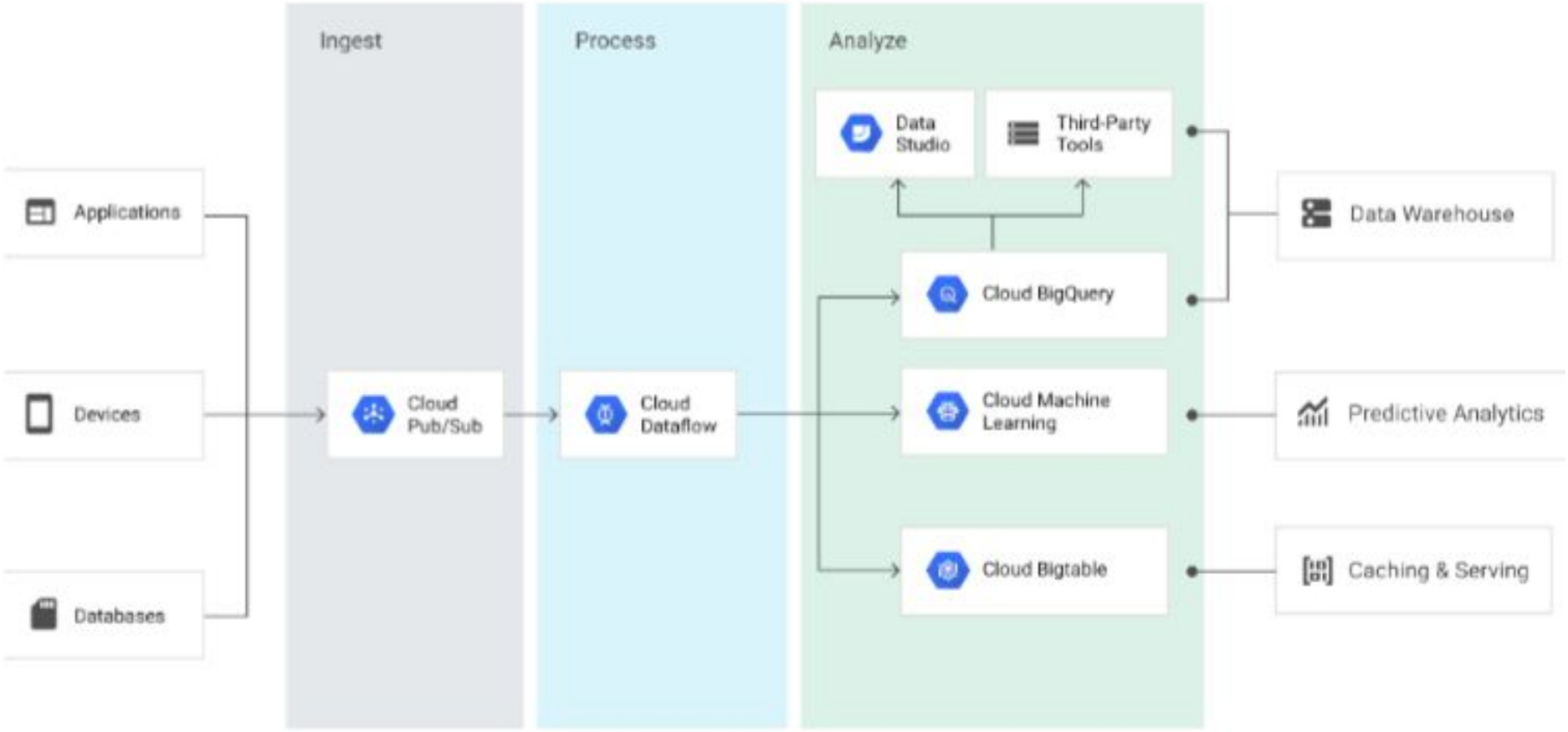
- Data volume used per month (per GB)

Monthly data	Price Per GB
First 10 GB	$0.00
Next 50 TB	$0.06
Next 100 TB	$0.05
Beyond 150 TB	$0.04

Out of order messaging

- Messages may arrive from multiple sources out of order.
- Pub/Sub does not care about message ordering.
- Dataflow is where out of order messages are processed/resolved.
- It's possible to add message attributes to help with ordering.

Big Picture:
Data Lifecycle for Streaming Data Ingest



pub/sub hands On

The Steps

- Create a topic
- Create a subscription
- Publish messages
- Retrieve messages

Simple topic/subscription/publish via `gcloud`

Create a topic called *my-topic*:

- `gcloud pubsub topics create my-topic`

Create subscription to topic *my-topic*:

- `gcloud pubsub subscriptions create --topic my-topic mySub1`

Publish a message to your topic:

- `gcloud pubsub topics publish my-topic --message "hello"`

Retrieve message with your subscription, acknowledge receipt, and remove message from queue:

- `gcloud pubsub subscriptions pull --auto-ack mySub1`

Cancel subscription:

- `gcloud pubsub subscriptions delete mySub1`

pub/sub hands On

Traffic Data Exercise

- **Clone GitHub**
- **Copy data points**
- **Simulate traffic data**
- **Pull messages**

Clone GitHub data to Cloud Shell (or other SDK environment), and browse to publish folder:

```
cd ~
git clone https://github.com/linuxacademy/googledataengineer
cd ~/googledataengineer/courses/streaming/publish
```

Create a topic called *sandiego*:

```
gcloud pubsub topics create sandiego
```

Create subscription to topic *sandiego*:

```
gcloud pubsub subscriptions create --topic sandiego mySub1
```

Run script to download sensor data:

```
./download_data.sh
```

May need to authenticate shell to ensure we have the right permissions:

```
gcloud auth application-default login
```

View script info:

```
vim ./send_sensor_data.py or use viewer of your choice
```

Run python script to simulate one hour of data per minute:

```
./send_sensor_data.py --speedFactor=60 \
--project=YOUR-PROJECT-ID
```

If you receive error: *google.cloud.pubsub can not be found* or an *ImportError: No module named iterator*, run this `pip` command to install components, then try again:

```
sudo pip install -U google-cloud-pubsub
```

Open new Cloud Shell tab (using + symbol)

Pull message using subscription *mySub1*:

```
gcloud pubsub subscriptions pull --auto-ack mySub1
```

Create a new subscription and pull messages with it:

```
gcloud pubsub subscriptions create --topic sandiego mySub2
gcloud pubsub subscriptions pull --auto-ack mySub2
```

Connecting Kafka to GCP

Does Pub/Sub Replace Kafka?

- Not always
- Hybrid workloads:
  - Interact with existing tools and frameworks
  - Don't need global/scaling capabilities with pub/sub
- Can use *both*: Kafka for on-premises and pub/sub for GCP in same data pipeline

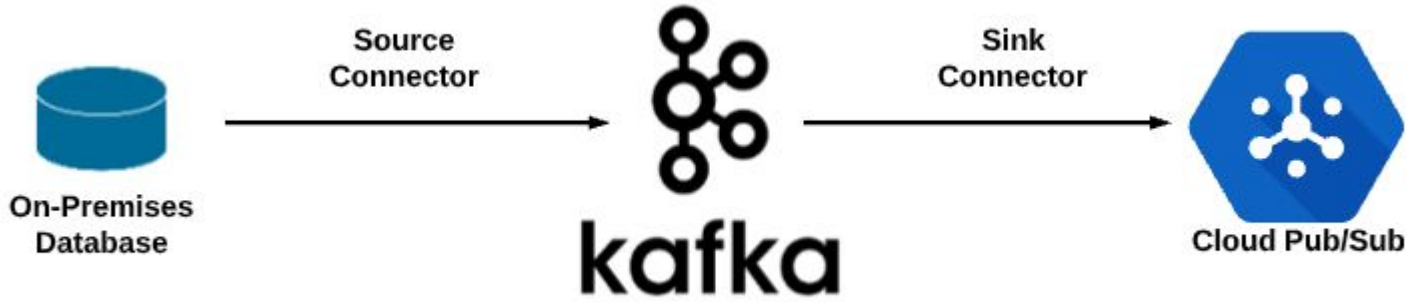
How do we connect Kafka to GCP?

Overview on Connectors:

- Open-source plugins that connect Kafka to GCP
- Kafka Connect: One optional "connector service"
- Exist to connect Kafka directly to pub/sub, Dataflow, and BigQuery (among others)

Additional Terms

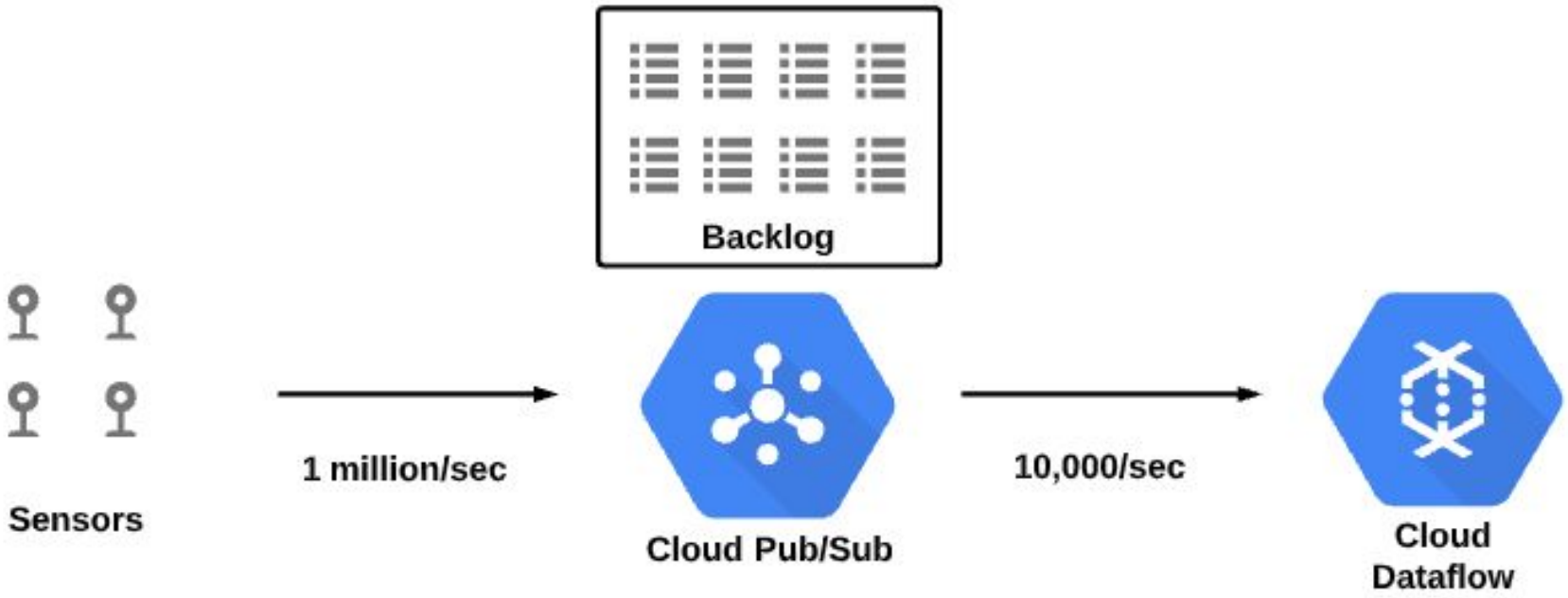
- **Source connector:** An upstream connector:
  - Streams *from* something *to* Kafka
- **Sink connector:** A downstream connector:
  - Streams *from* Kafka *to* something else

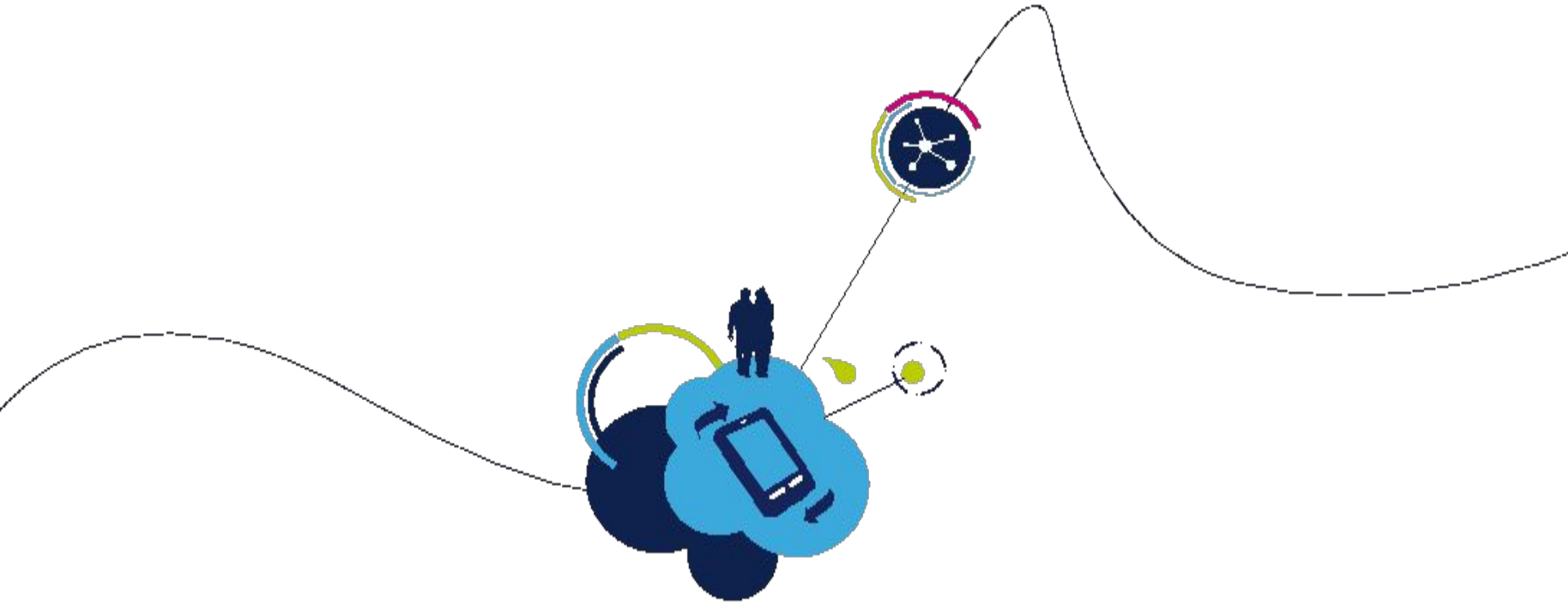


Monitoring Subscriber Health

Troubleshooting Subscriber Health (Backlog)

- Create alerts for (x) backlog threshold
- Subscriber not able to keep up:
  - Under-provisioned
  - Code not optimized
- Not acknowledging message receipt:
  - Pub/Sub doesn't know it's delivered, and keeps trying
  - Subscriber code not properly acknowledging pulled messages
- Check publishers for excessive re-transmits



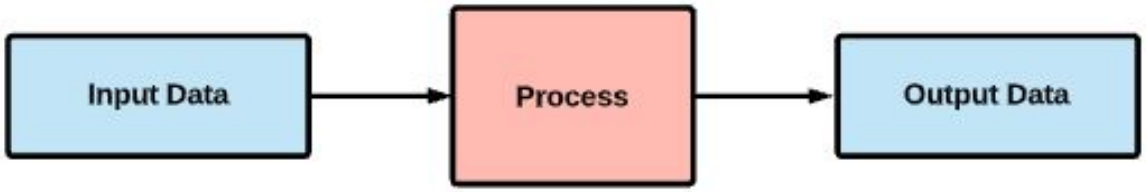


Pipelines with Cloud Dataflow

Data Processing Challenges

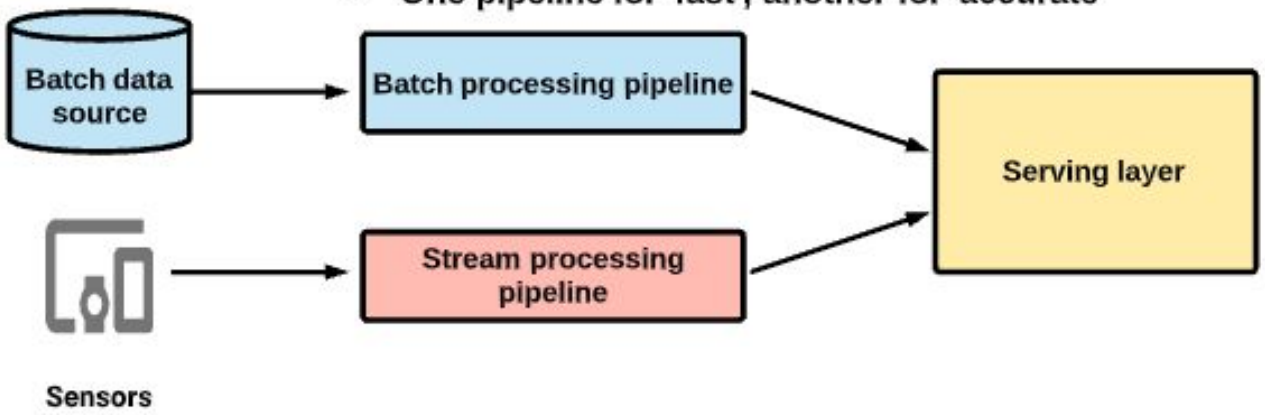
What is Data Processing?

- Read Data (Input)
- Transform it to be relevant - Extract, Transform, and Load (ETL)
- Create output



Challenge: Streaming and Batch data pipelines:

- Until recently, separate pipelines are required for each
- Difficult to compare recent and historical data
- One pipeline for 'fast', another for 'accurate'



Why both?

- Credit card monitoring
- Compare streaming transactions to historical batch data to detect fraud

Data Processing Challenges

Challenge: Complex element processing:

- Element = single data input
- One at a time element ingest from single source = easy
- Combining elements (aggregation) = hard
- Processing data from different sources, streaming, and out of order (composite) = REALLY hard

Solution: Apache Beam + Cloud Dataflow

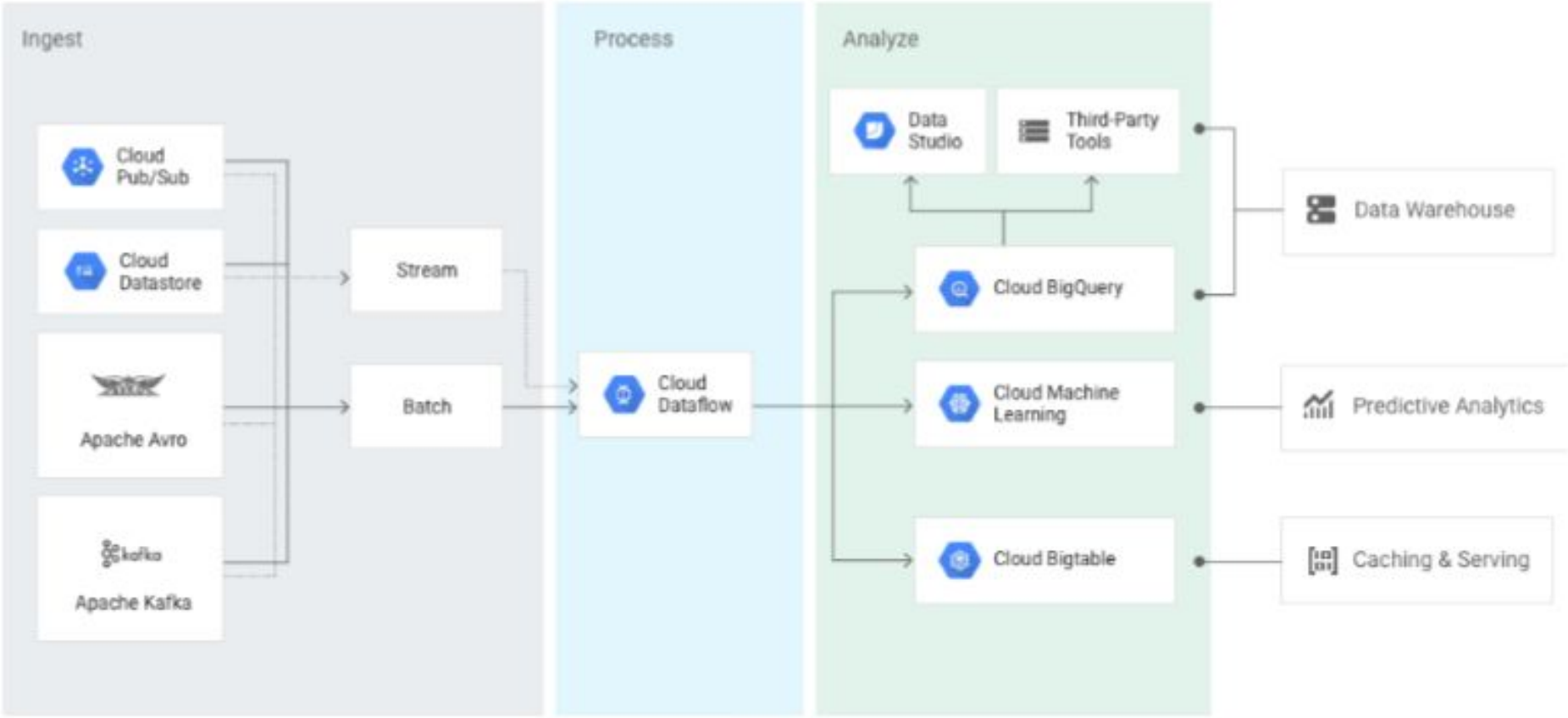
Cloud Datfalow overview

What is it?

- Auto-scaling, No-Ops, Stream, and Batch Processing
- Built on Apache Beam:
  - Documentation refers to Apache Beam site
  - Configuration is 100% code-based
- Integrates with other tools (GCP and external):
  - Natively - Pub/Sub, BigQuery, Cloud ML Engine
  - Connectors - Bigtable, Apache Kafka
- Pipelines are regional-based

Big Picture - Data Transformation

Cloud Datfalow overview



Cloud Datfalow overview

IAM:

- Project-level only - all pipelines in the project (or none)
- Pipeline data access separate from pipeline access
- Dataflow Admin - Full pipeline access plus machine type/storage bucket config access
- Dataflow Developer - Full pipeline access, no machine type/storage bucket access
- Dataflow Viewer - view permissions only
- Dataflow Worker - Specifically for service accounts

Dataflow vs Dataproc?

Beam vs. Hadoop/Spark?

Cloud Datfalow overview

Dataproc:

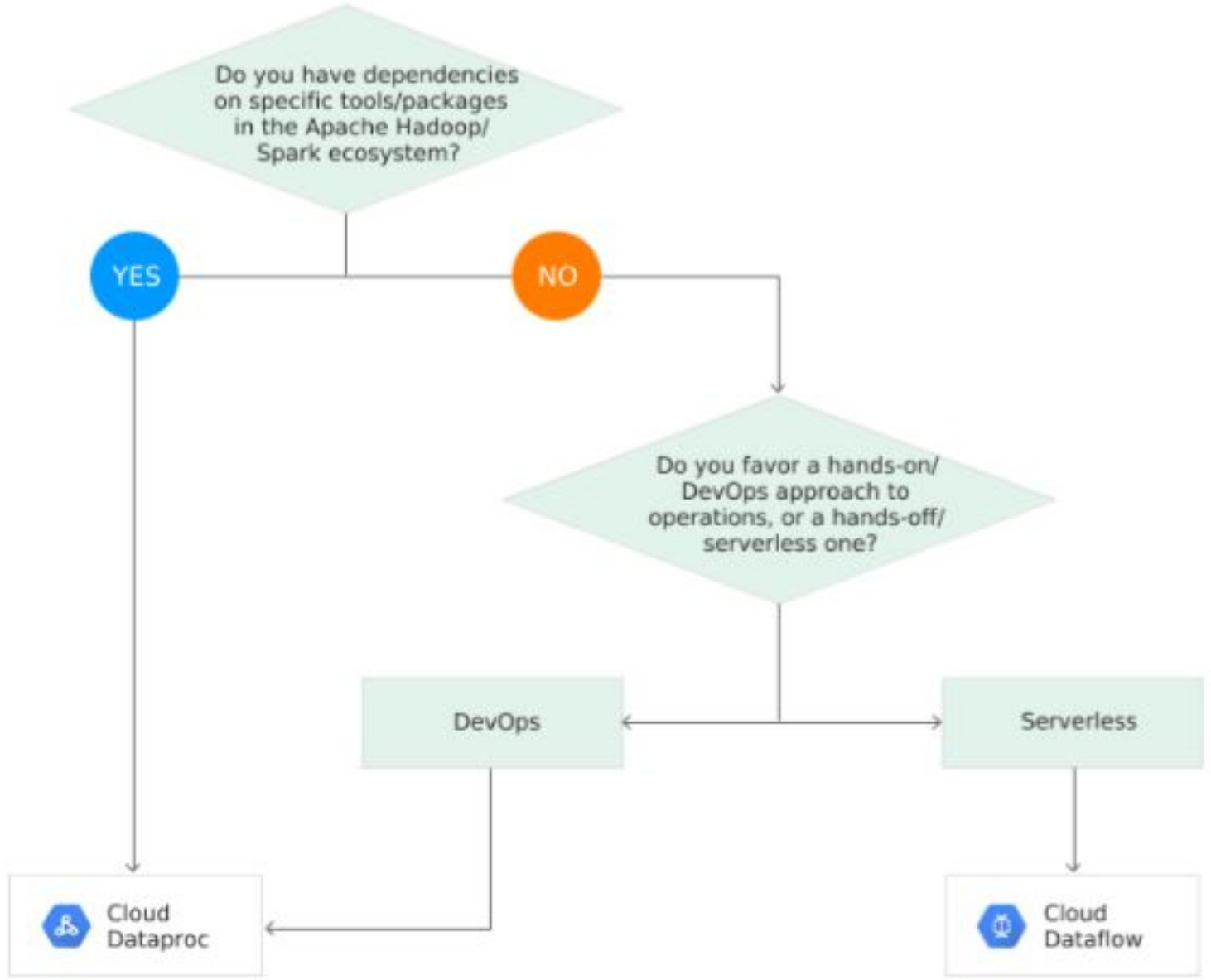
- Familiar tools/packages
- Employee skill sets
- Existing pipelines

Dataflow:

- Less Overhead
- Unified batch and stream processing
- Pipeline portability across Dataflow, Spark, and Flink as runtimes

WORKLOADS	CLOUD DATAPROC	CLOUD DATAFLOW
Stream processing (ETL)		X
Batch processing (ETL)	X	X
Iterative processing and notebooks	X	
Machine learning with Spark ML	X	
Preprocessing for machine learning		X (with Cloud ML Engine)

Dataflow vs. Dataproc decision tree



Key Concepts

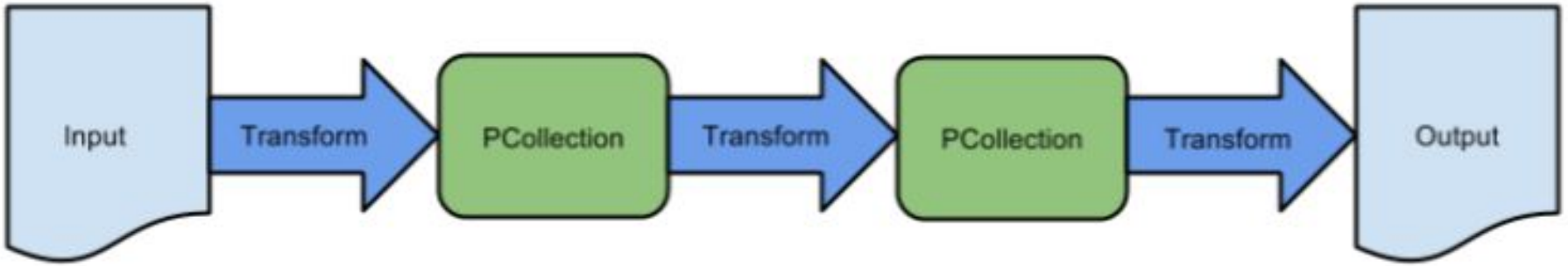
Course/exam perspective:

- Dataflow is very code-heavy
- Exam does not go deep into coding questions
- Some key concepts/terminology will be tested

Key terms:

- Element - single entry of data (e.g., table row)
- PCollection - Distributed data set, data input and output
- Transform - Data processing operation (or step) in pipeline:
  - Uses programming conditionals (for/while loops, etc.)
- ParDo - Type of transform applied to individual elements:
  - Filter out/extract elements from a large group of data

Key Concepts



PCollection and ParDo in example Java code.

One step in a multi-step transformation process.

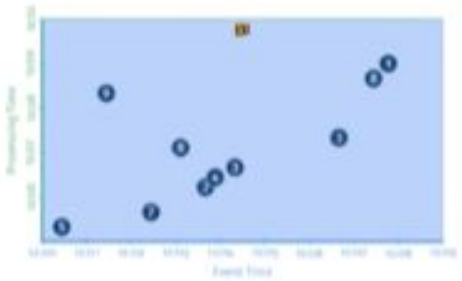
```
PCollection<LaneInfo> currentConditions = p //
    .apply("GetMessages", PubsubIO.readStrings().fromTopic(topic)) //
    .apply("ExtractData", ParDo.of(new DoFn<String, LaneInfo>() {
      @ProcessElement
      public void processElement(ProcessContext c) throws Exception {
        String line = c.element();
        c.output(LaneInfo.newLaneInfo(line));
      }
    }));
```

Key Concepts

Dealing with late/out of order data:

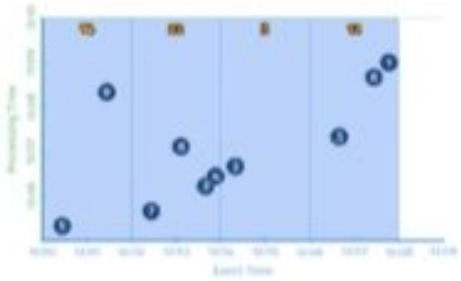
- Latency is to be expected (network latency, processing time, etc.)
- Pub/Sub does not care about late data, that is resolved in Dataflow
- Resolved with Windows, Watermarks, and Triggers
- Windows = logically divides element groups by time span
- Watermarks = 'timestamp':
  - Event time = when data was generated
  - Processing time = when data processed anywhere in the processing pipeline
  - Can use Pub/Sub-provided watermark or source-generated
- Trigger = determine when results in window are emitted (submitted as complete):
  - Allow late-arriving data in allowed time window to re-aggregate previously submitted results
  - Timestamps, element count, combinations of both

Key Concepts



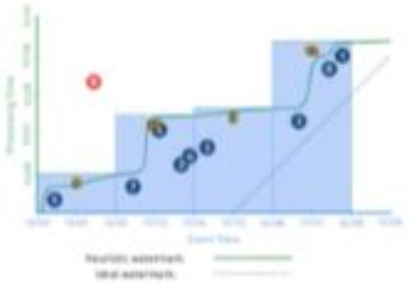
1

Classic Batch



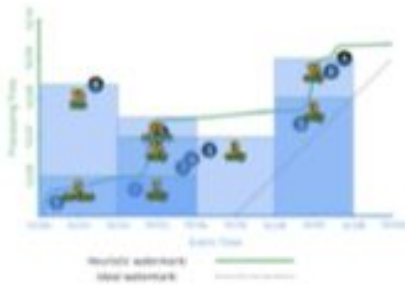
2

Windowed Batch



3

Streaming w/Discarding

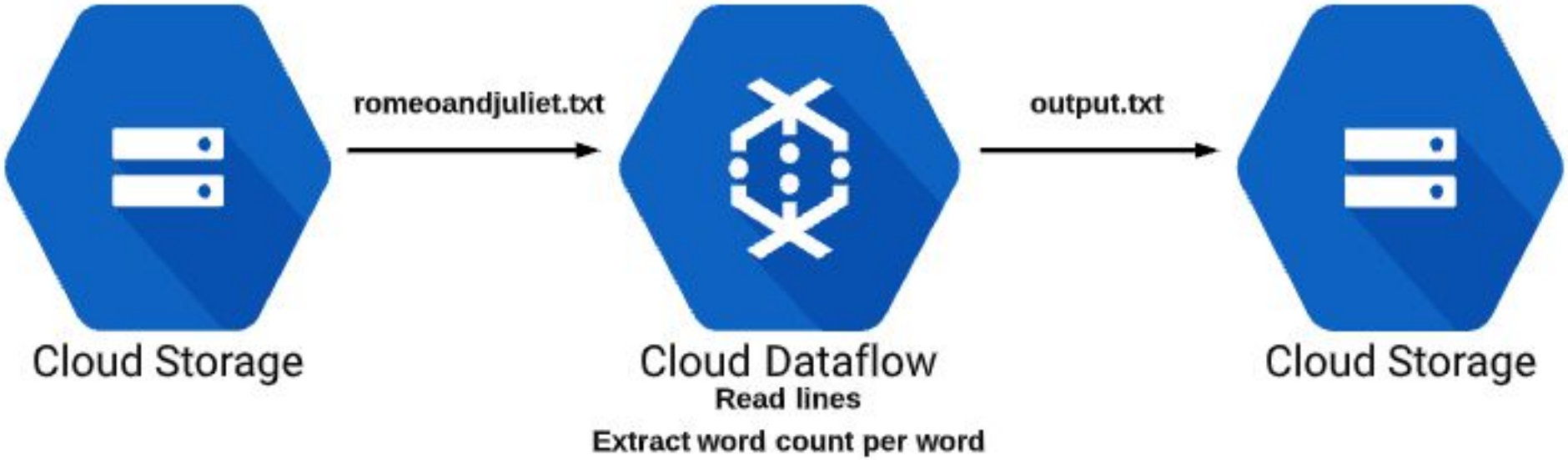


4

Streaming + Accumulation

Template Hands on

- **Google-provided templates**
- **Simple word count extraction**



Streaming Ingest Pipeline Hands On

- Take San Diego traffic data
- Ingest through Pub/Sub
- Process with Dataflow
- Analyze results with BigQuery
- First: Enable Dataflow API from API's and Services

Traffic data

Data ingest

Published streaming sensor data

Topic: sandiego

Subscription pulls messages

Cloud Dataflow

Transform data to calculate average speed.

Output to BigQuery.

BigQuery

Streaming Ingest Pipeline Hands On

Quick command line setup (Cloud Shell)

- Create BigQuery dataset for processing pipeline output:
  - bq mk --dataset $DEVSHELL_PROJECT_ID:demos
- Create Cloud Storage bucket for Dataflow staging:
  - gsutil mb gs://$DEVSHELL_PROJECT_ID
- Create Pub/Sub topic and stream data:
  - cd ~/googledataengineer/courses/streaming/publish
  - gcloud pubsub topics create sandiego
  - ./download_data.sh
  - sudo pip install -U google-cloud-pubsub
  - ./send_sensor_data.py --speedFactor=60 --project=$DEVSHELL_PROJECT_ID

Open a new Cloud Shell tab:

- Execute Dataflow pipeline for calculating average speed:
  - cd ~/googledataengineer/courses/streaming/process/sandiego
  - ./run_oncloud.sh $DEVSHELL_PROJECT_ID $DEVSHELL_PROJECT_ID AverageSpeeds
- Error resolution:
  - Pub/Sub permission denied, re-authenticate
    - gcloud auth application-default login
  - Dataflow workflow failed - enable Dataflow API

Streaming Ingest Pipeline Hands On

View results in BigQuery:

- List first 100 rows:
  - SELECT * FROM [<PROJECTID>:demos.average_speeds] ORDER BY timestamp DESC LIMIT 100
- Show last update to table:
  - SELECT MAX(timestamp) FROM [<PROJECTID>:demos.average_speeds]
- Look at results from the last minute:
  - SELECT * FROM [<PROJECTID>:demos.average_speeds@-60000] ORDER BY timestamp DESC

Shut down pipeline:

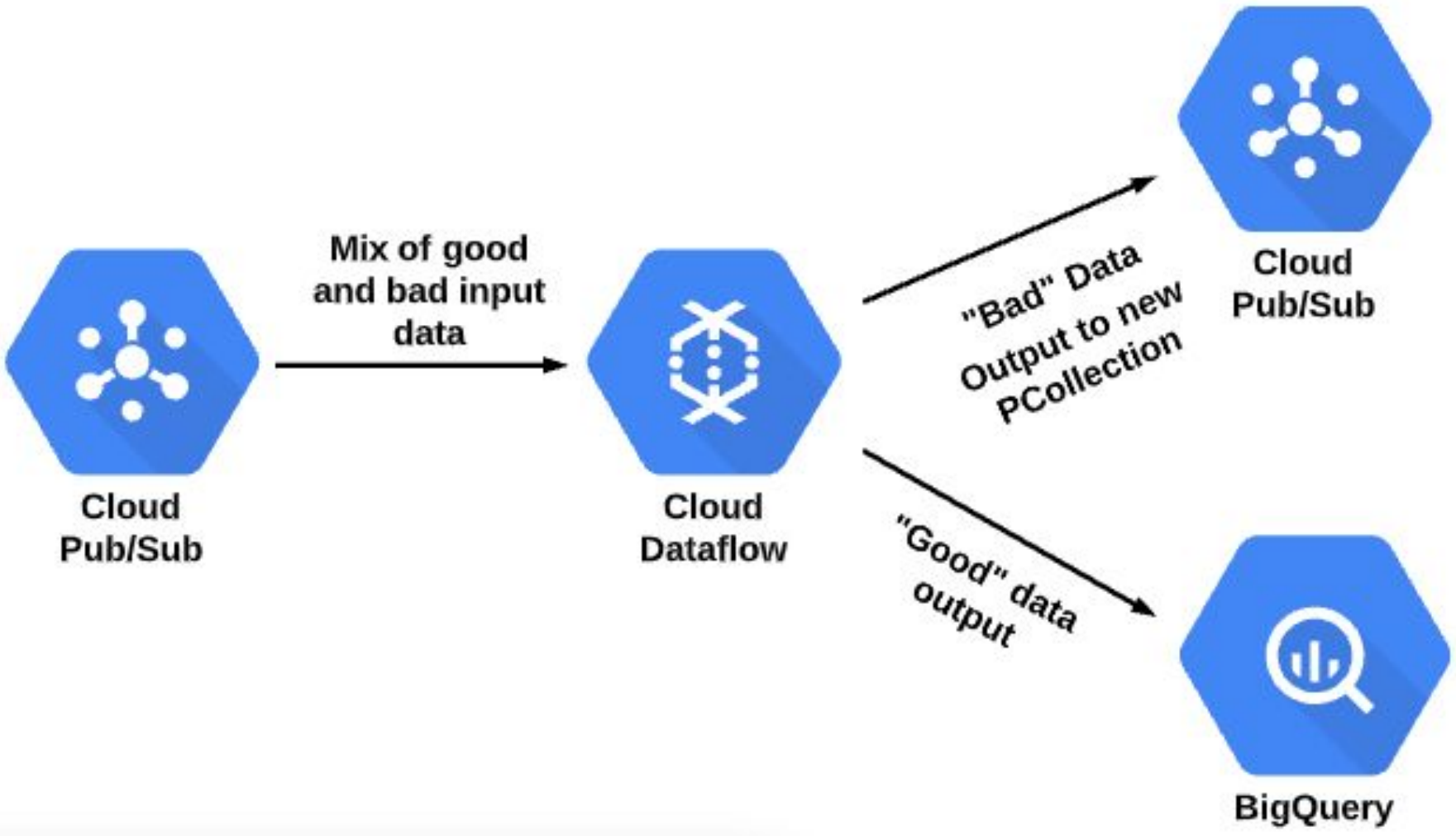
- Drain - finishing processing buffered jobs before shutting down
- Cancel - full stop, cancels existing buffered jobs

Additional best practice

Handling Pipeline Errors

- If you do not have a mechanism in place to handle input data errors in your pipeline, the job can fail. How can we account for this?
- Gracefully catch errors:
  - Create separate output:
    - **Try-catch** block handles errors
    - Output errors to new **PCollection** - Send to **collector** for later analysis (Pub/Sub is a good target)
    - Think of it as *recycling* the *bad* data
- Technique is also valid for troubleshooting missing messages:
  - Scenario: Streaming pipeline missing some messages
  - Solution: Run a batch of the streaming data, and check output:
    - Create additional output to capturing and processing error data.

Additional best practice

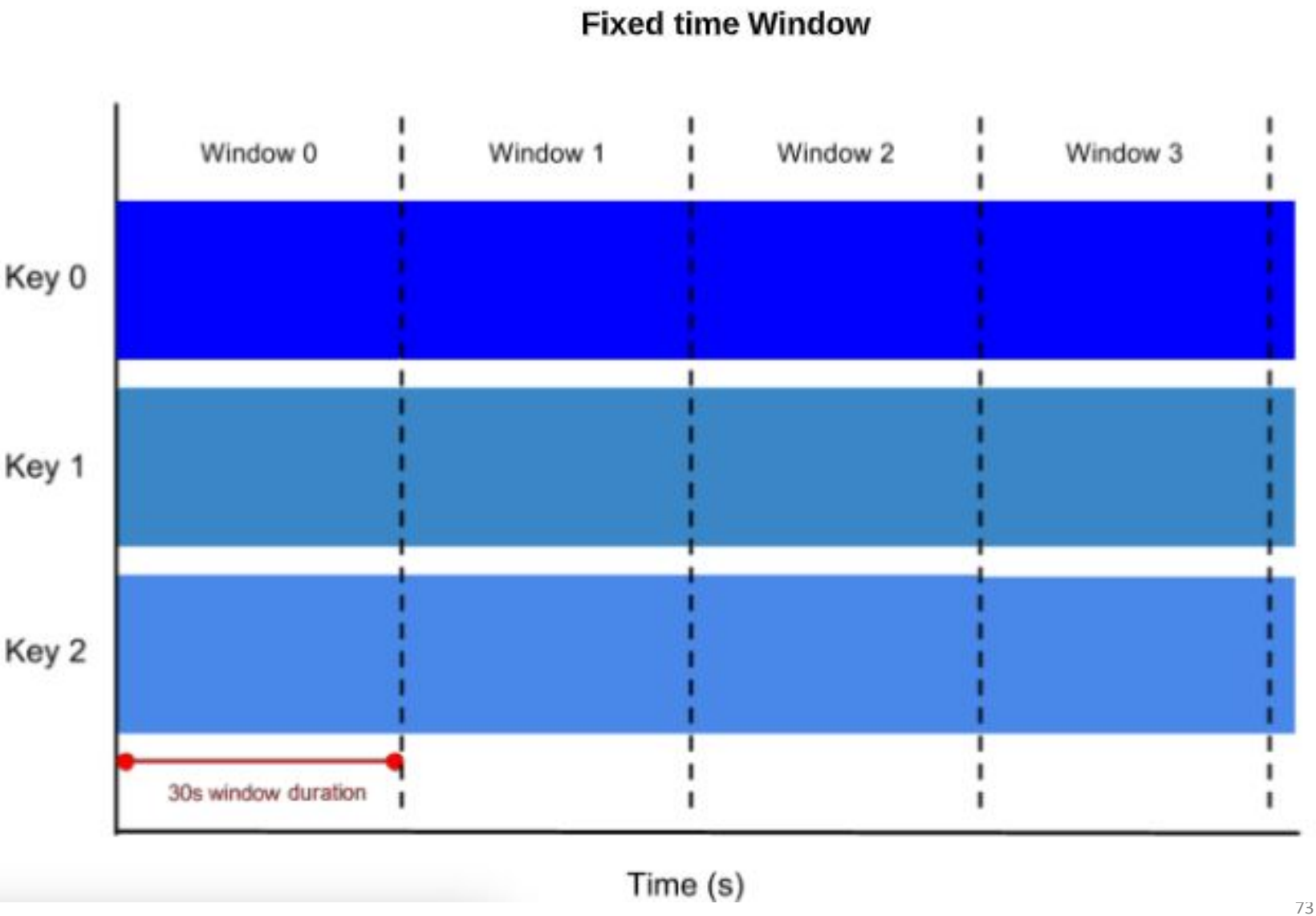


Additional best practice

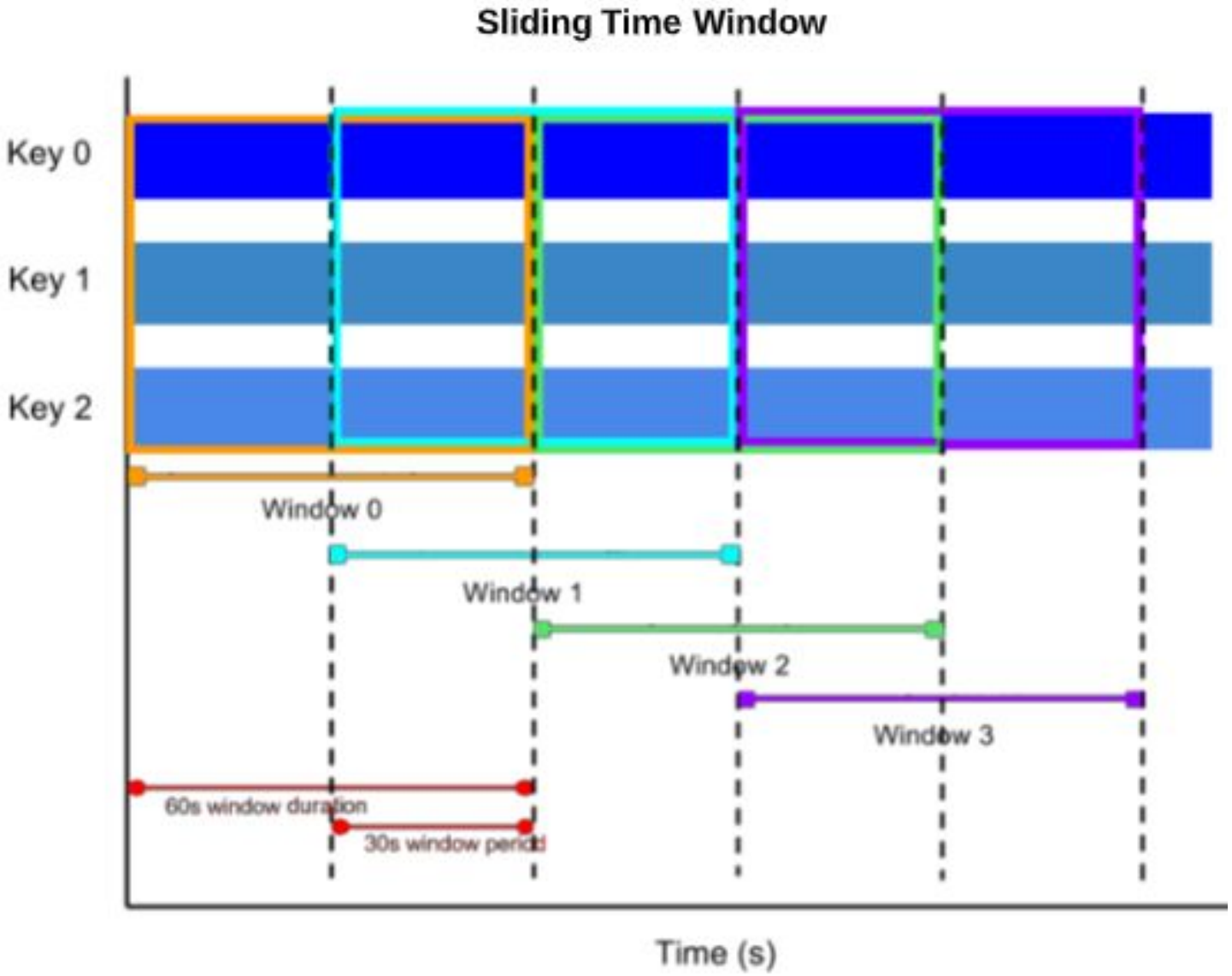
Know your window types

- **Global, Fixed, Sliding, Session**
- **Global** - The default, uses a single window for entire pipeline
- **Fixed time** - Every (x) period of time
  - Every 5 seconds, 10 minutes, etc.
- **Sliding time** - Overlapping time windows
- **Session** - Within certain time of certain elements:
  - For example, *Time since last user/mouse activity*

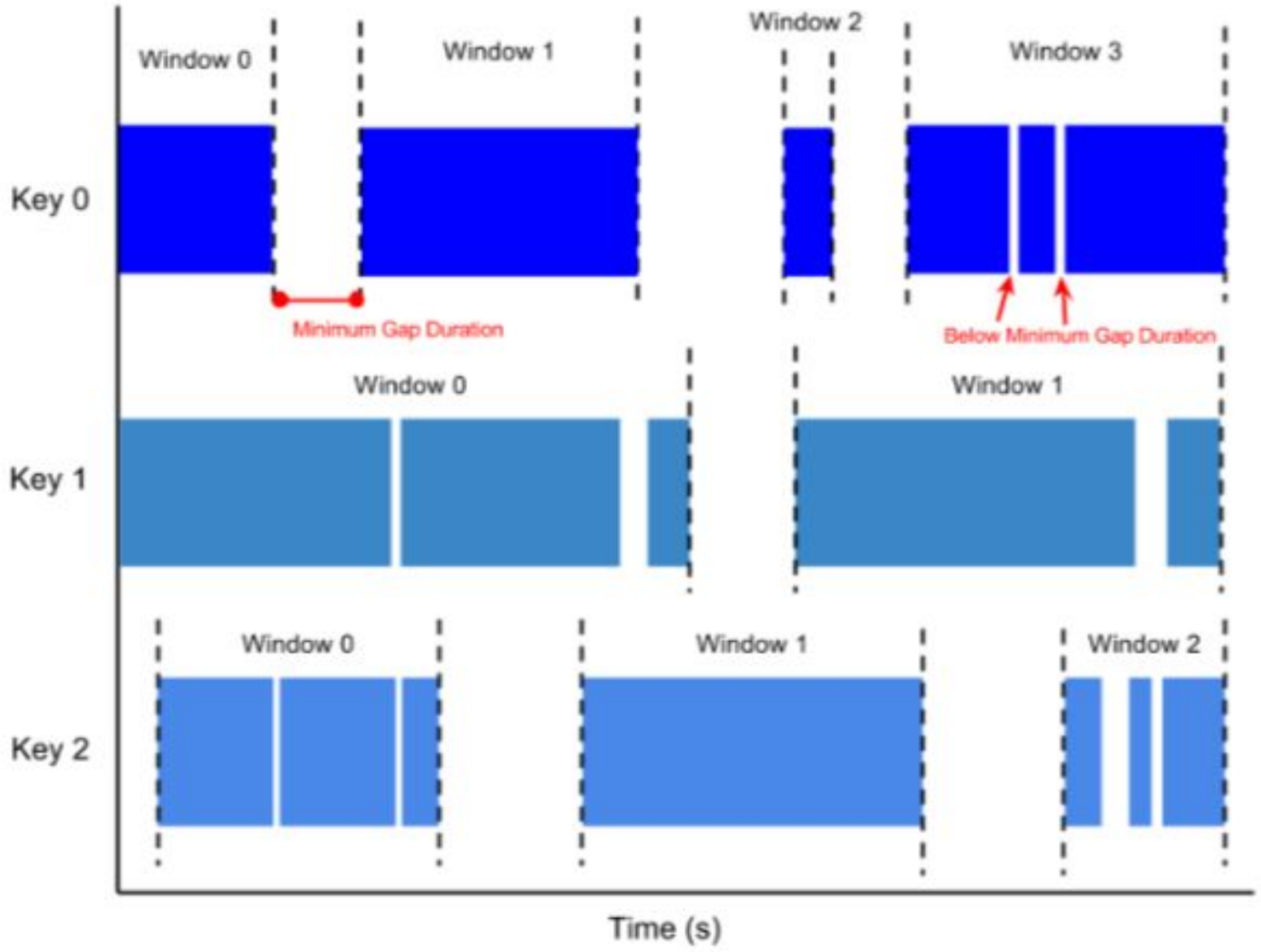
Additional best practice



Additional best practice



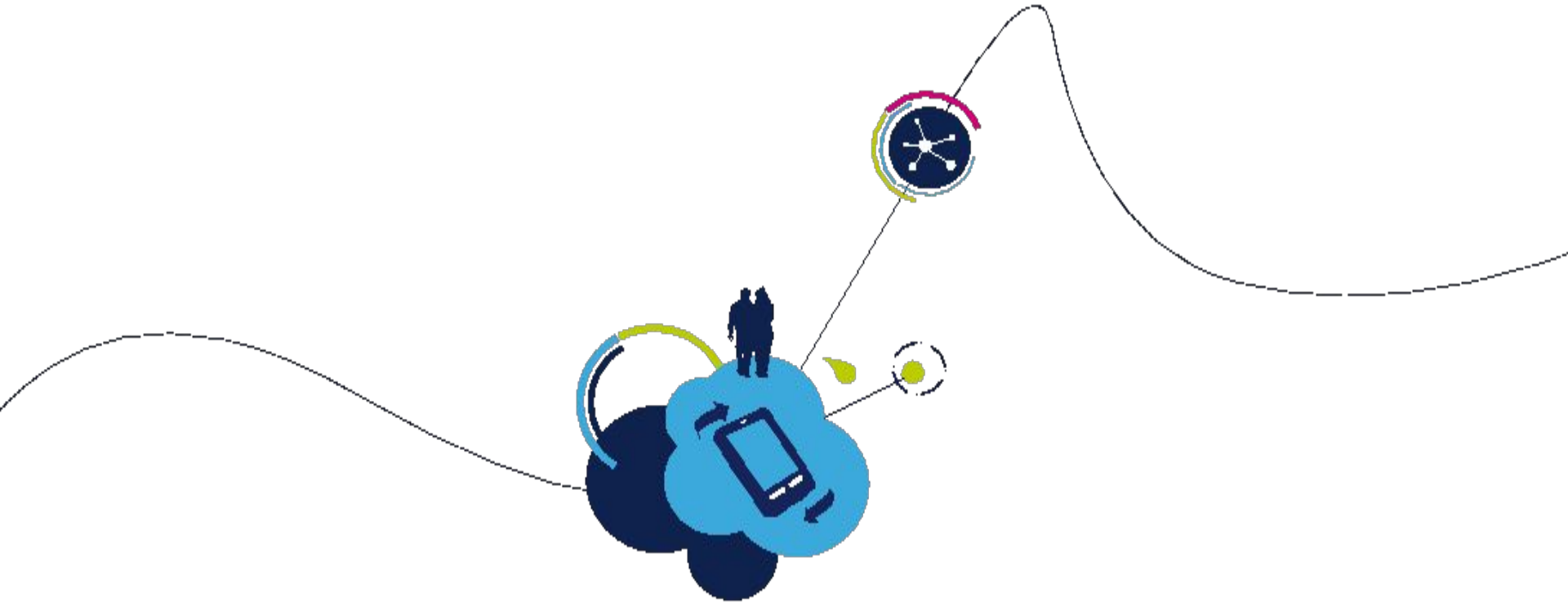
Additional best practice



Additional best practice

Updating Dataflow Pipelines

- **Scenario:** Update streaming Dataflow pipeline with new code:
  - New code = new pipeline not compatible with current version
  - Need data to *switch over* to new job/pipeline without losing anything in the process
- **Solution**: Update job:
  - Creates new job with same name/new *jobID*
- Compatibility between old/new jobs:
  - Map old to new job transforms with **transform mapping**
    - "Bridge" between old and new code base
  - After compatibility check:
    - Buffered data transferred to new job, using transform mapping to translate changes



Managed Spark with Cloud Dataproc

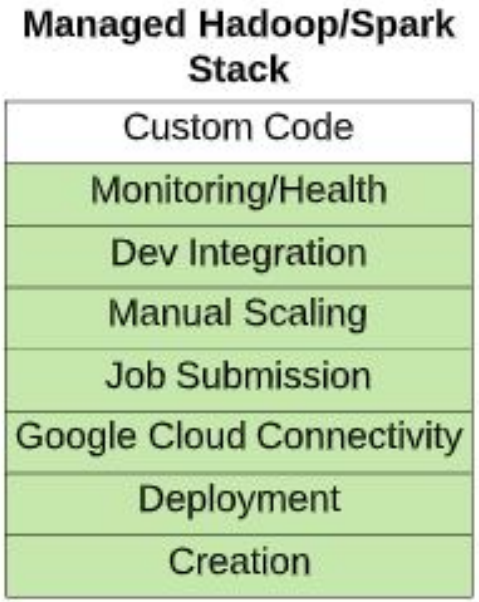
Dataproc Overview

What is Cloud Dataproc?

Hadoop ecosystem:

- **Hadoop, Spark, Pig, Hive**
- **Lift and shift to GCP**

Dataproc Overview



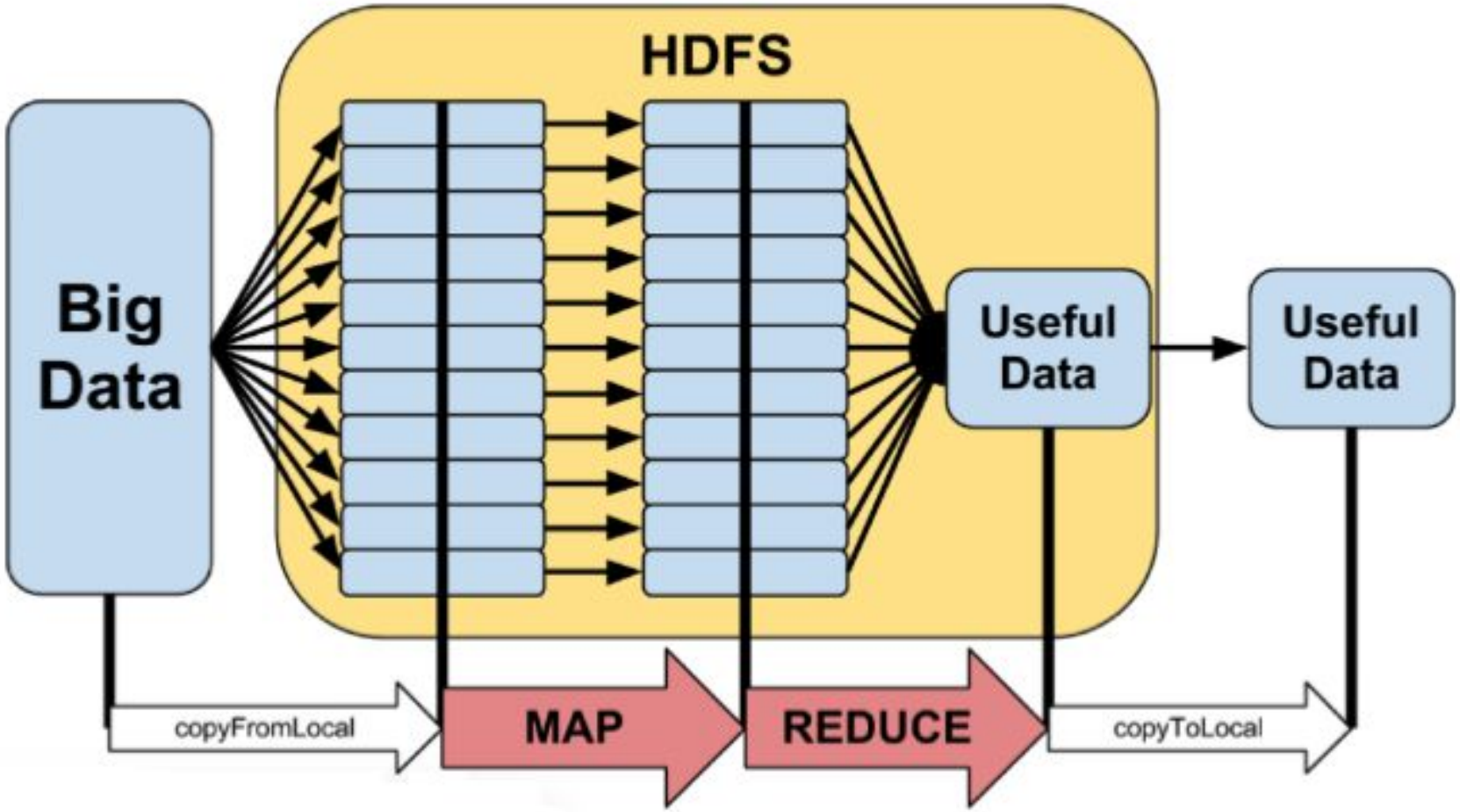
Dataproc facts:

- On-demand, managed Hadoop and Spark clusters
- Managed, but not no-ops:
  - Must configure cluster, not auto-scaling
  - Greatly reduces administrative overhead
- Integrates with other Google Cloud services:
  - Separate data from the cluster - save costs
- Familiar Hadoop/Spark ecosystem environment:
  - Easy to move existing projects
- Based on Apache Bigtop distribution:
  - Hadoop, Spark, Hive, Pig
- HDFS available (but maybe not optimal)
- Other ecosystem tools can be installed as well via initialization actions

Dataproc Overview

What is MapReduce?

- **Simple definition:**
  - **Take big data, distribute it to many workers (map)**
  - **Combine results of many pieces (reduce)**
- **Distributed/parallel computing**



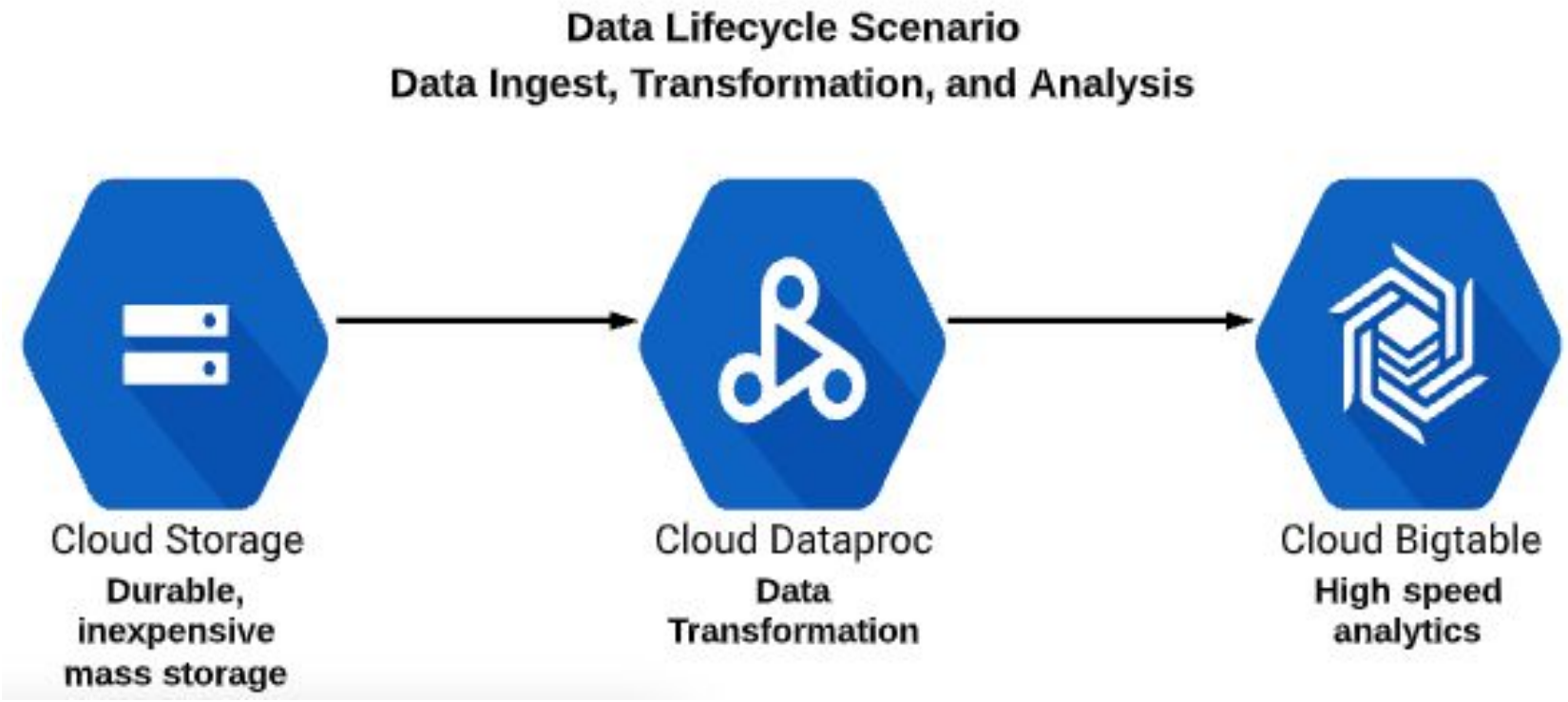
Dataproc Overview

Pricing:

- **Standard Compute Engine machine type pricing + managed Dataproc premium**
- **Premium = $0.01 per vCPU core/hour**

Machine type	Virtual CPUs	Memory	Dataproc
n1-highcpu-2	2	1.80GB	$0.020
n1-highcpu-4	4	3.60GB	$0.040
n1-highcpu-8	8	7.20GB	$0.080
n1-highcpu-16	16	14.40GB	$0.160
n1-highcpu-32	32	28.80GB	$0.320
n1-highcpu-64	64	57.60GB	$0.640

Dataproc Overview



Dataproc Overview

Identity and Access Management (IAM):

- Project level only (primitive and predefined roles)
- Cloud Dataproc Editor, Viewer, Worker
- Editor - Full access to create/delete/edit clusters/jobs/workflows
- Viewer - View access only
- Worker - Assigned to service accounts:
  - Read/write GCS, write to Cloud Logging

Configure Dataproc Cluster

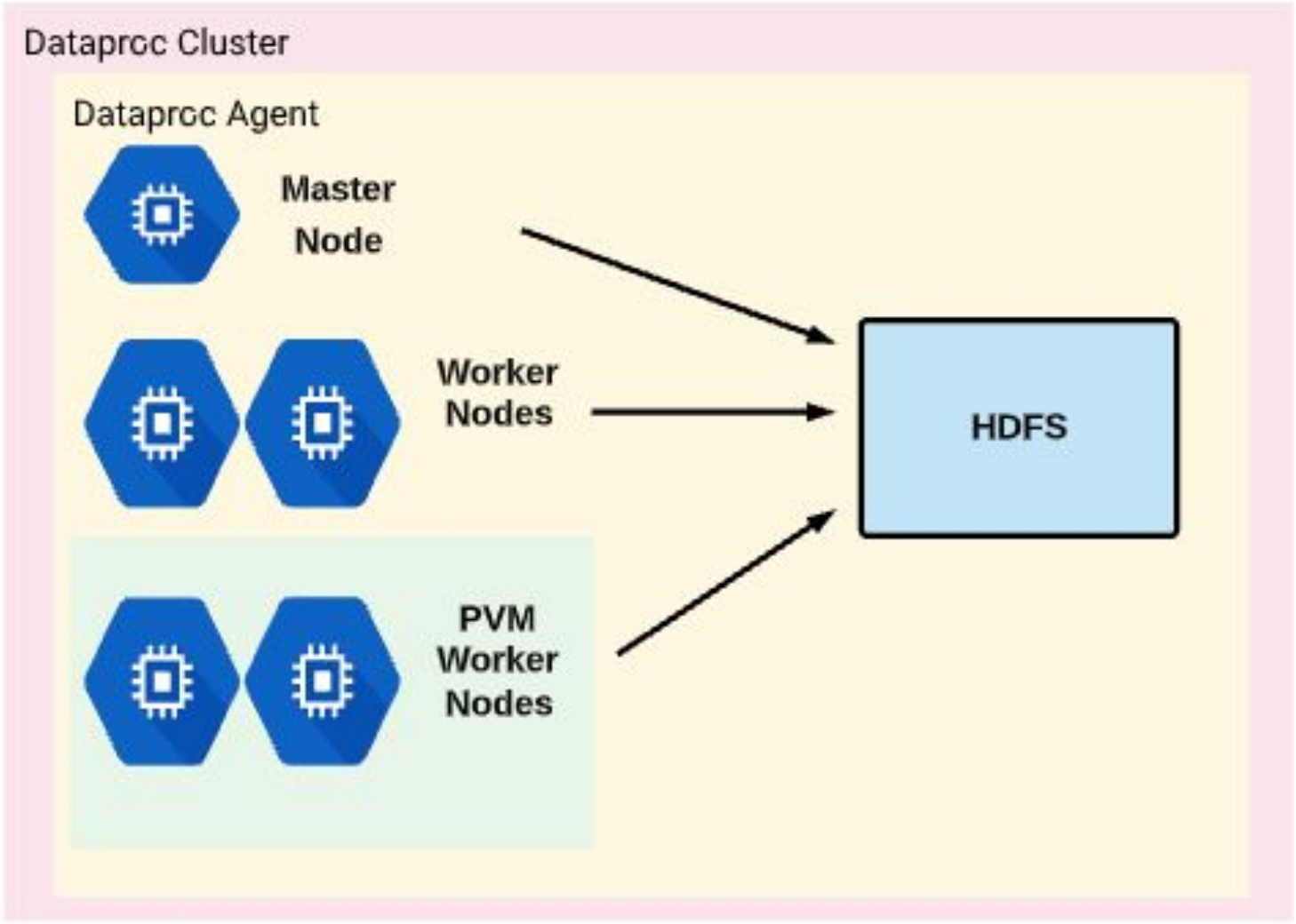
Create cluster:

- gcloud dataproc clusters create [cluster_name] --zone [zone_name]
- Configure master node, worker nodes:
  - Master contains YARN resource manager
  - YARN = Yet Another Resource Negotiator

Updating clusters:

- Can only change # workers/preemptible VM's/labels/toggle graceful decommission
- Automatically reshards data for you
- gcloud dataproc clusters update [cluster_name] --num-workers [#] --num-preemptible-workers [#]

Configure Dataproc Cluster



Configure Dataproc Cluster

Preemptible VM's on Dataproc:

- Excellent low-cost worker nodes
- Dataproc manages the entire leave/join process:
  - No need to configure startup/shutdown scripts
  - Just add PVM's...and that's it
- No assigned disks for HDFS (only disk for caching)
- Want a mix of standard + PVM worker nodes

Configure Dataproc Cluster

Access your cluster:

- SSH into master - same as any compute engine instance
- gcloud compute ssh [master_node_name]

Access via web - 2 options:

- Open firewall ports to your network (8088, 9870)
- Use SOCKS proxy - does not expose firewall ports

SOCKS proxy configuration:

- SSH to master to enable port forwarding:
  - gcloud compute ssh *master-host-name* --project=*project-id* --zone=*master-host-zone* -- -D 1080 -N
- Open new terminal window - launch web browser with parameters (varies by OS/browser):
  - "/Applications/Google Chrome.app/Contents/MacOS/Google Chrome" --proxy-server="socks5://localhost:1080" --host-resolver-rules="MAP * 0.0.0.0 , EXCLUDE localhost" --user-data-dir=/tmp/*cluster1-m*
- Browse to http://[master]:port:
  - 8088 - Hadoop
  - 9870 - HDFS

Using Cloud Shell (must use for each port):

- gcloud compute ssh *master-host-name* --project=*project-id* --zone *master-host-zone* -- -4 -N -L *port1*:*master-host-name*:*port2*
- Use Web Preview to choose port (8088/9870)

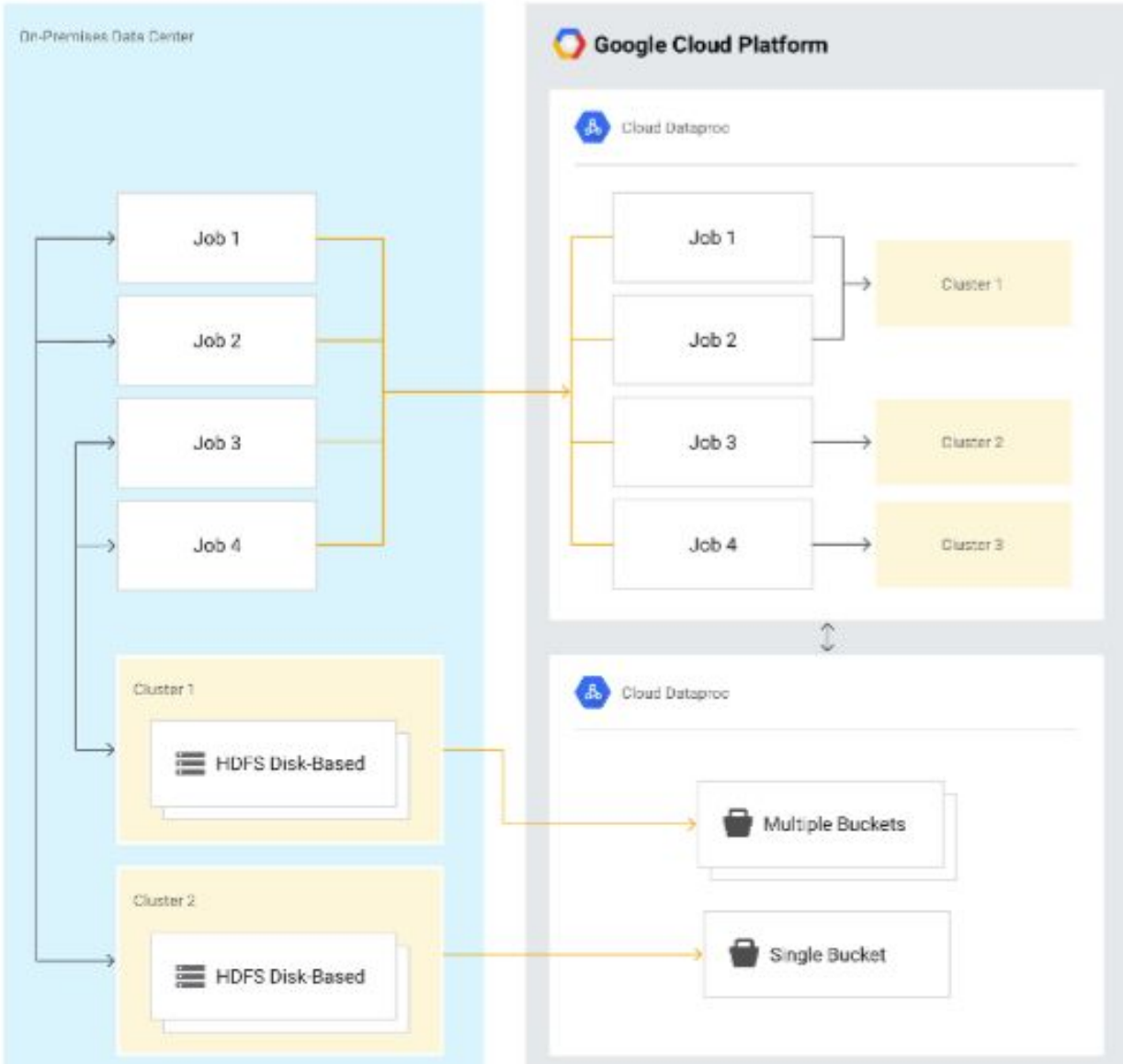
Migrating and optimizing for Google Cloud

Migrating to Cloud Dataproc

What are we moving/optimizing?

- **Data (from HDFS)**
- **Jobs (pointing to Google Cloud locations)**
- **Treating clusters as ephemeral (temporary) rather than permanent entities**

Install Cloud Storage connector to connect to GCS (Google Cloud Storage).



Migrating and optimizing for Google Cloud

Migration Best Practices:

- Move data first (generally Cloud Storage buckets):
  - Possible exceptions:
    - Apache HBase data to Bigtable
    - Apache Impala to BigQuery
    - Can still choose to move to GCS if Bigtable/BQ features not needed
- Small-scale experimentation (proof of concept):
  - Use a subset of data to test
- Think of it in terms of ephemeral clusters
- Use GCP tools to optimize and save costs

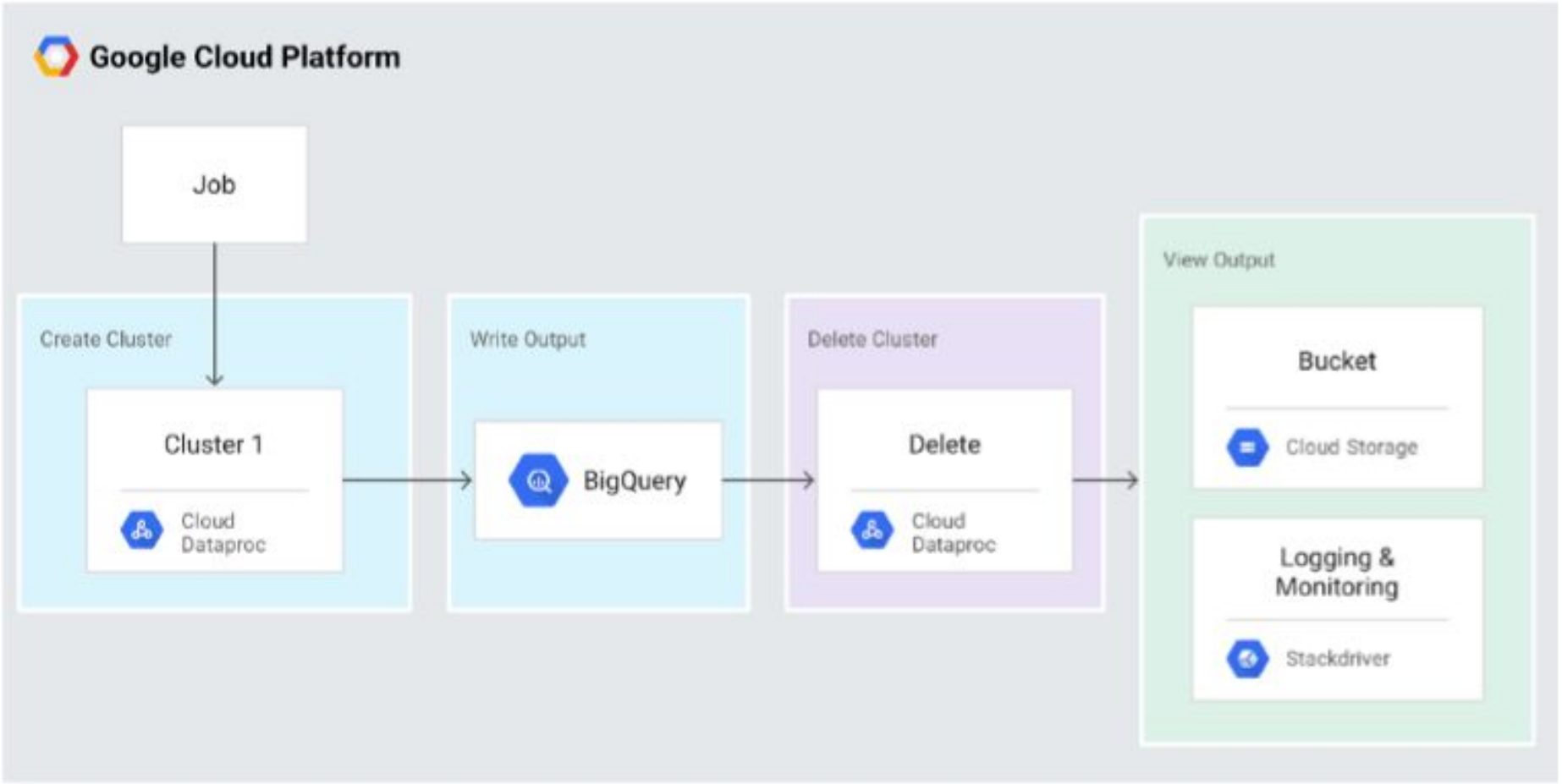
Migrating and optimizing for Google Cloud

Optimize for the Cloud ("Lift and Leverage")

Separate storage and compute (cluster):

- Save on costs:
  - No need to keep clusters to keep/access data
- Simplify workloads:
  - No shaping workloads to fit hardware
  - Simplify storage capacity
- HDFS --> Google Cloud Storage
- Hive --> BigQuery
- HBase --> Bigtable

Migrating and optimizing for Google Cloud



Migrating and optimizing for Google Cloud

Converting from HDFS to Google Cloud Storage:

1. Copy data to GCS:
   - Install connector or copy manually
2. Update file prefix in scripts:
   - From hdfs:// to gs://
3. Use Dataproc, and run against/output to GCS

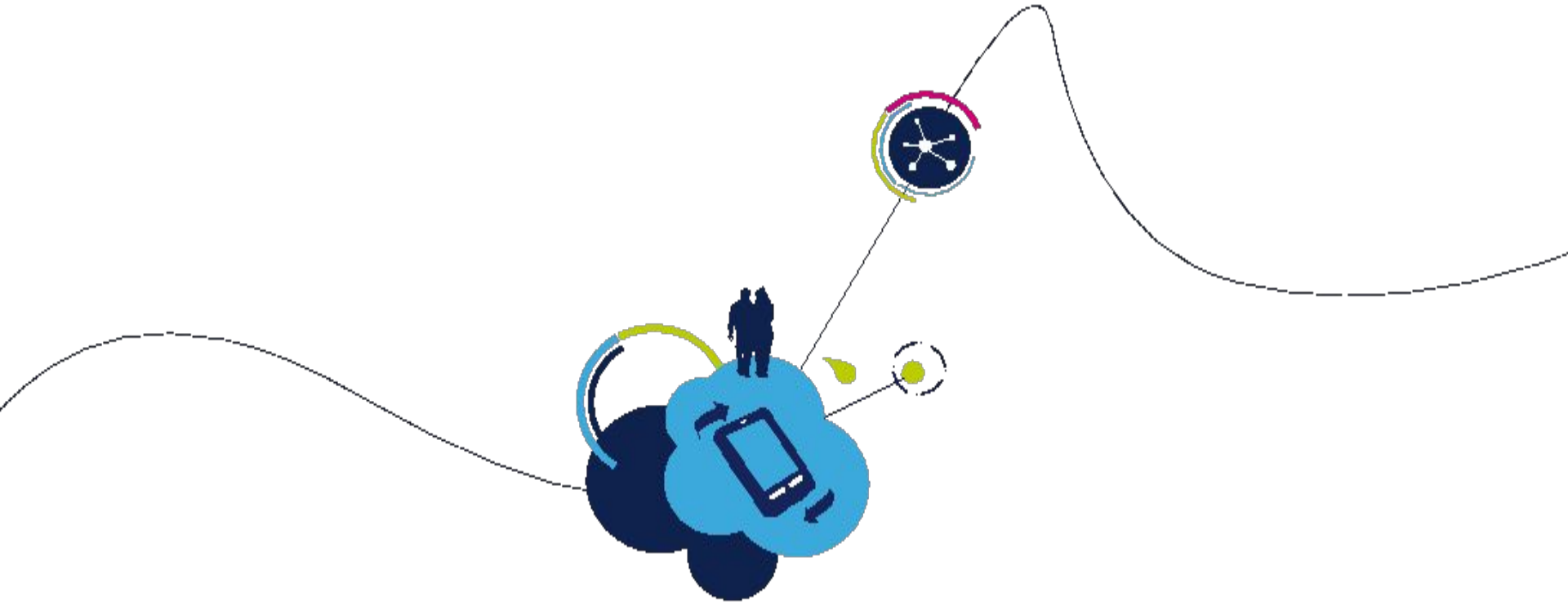
The end goal should be to eventually move toward a cloud-native and serverless architecture (Dataflow, BigQuery, etc.).

Best Practices for Cluster Performance

Dataproc Performance Optimization

(GCP-specific)

- Keep data close to your cluster
  - Place Dataproc cluster in the same region as storage bucket
- Larger persistent disk = better performance
  - Consider using SSD over HDD – slightly higher cost
- Allocate more VM's
  - Use preemptible VM's to save on costs
  - More VM's will come at a higher cost than larger disks if more disk throughput is needed



NoSQL Data with Cloud Bigtable

Cloud Bigtable Overview

What is Cloud Bigtable?

- High performance, massively scalable NoSQL database
- Ideal for large analytical workloads

History of Bigtable

- Considered one of the originators for a NoSQL industry
- Developed by Google in 2004
  - Existing database solutions were too slow
  - Needed real-time access to petabytes of data
- Powers Gmail, YouTube, Google Maps, and others

What is it used for?

- High throughput analytics
- Huge datasets

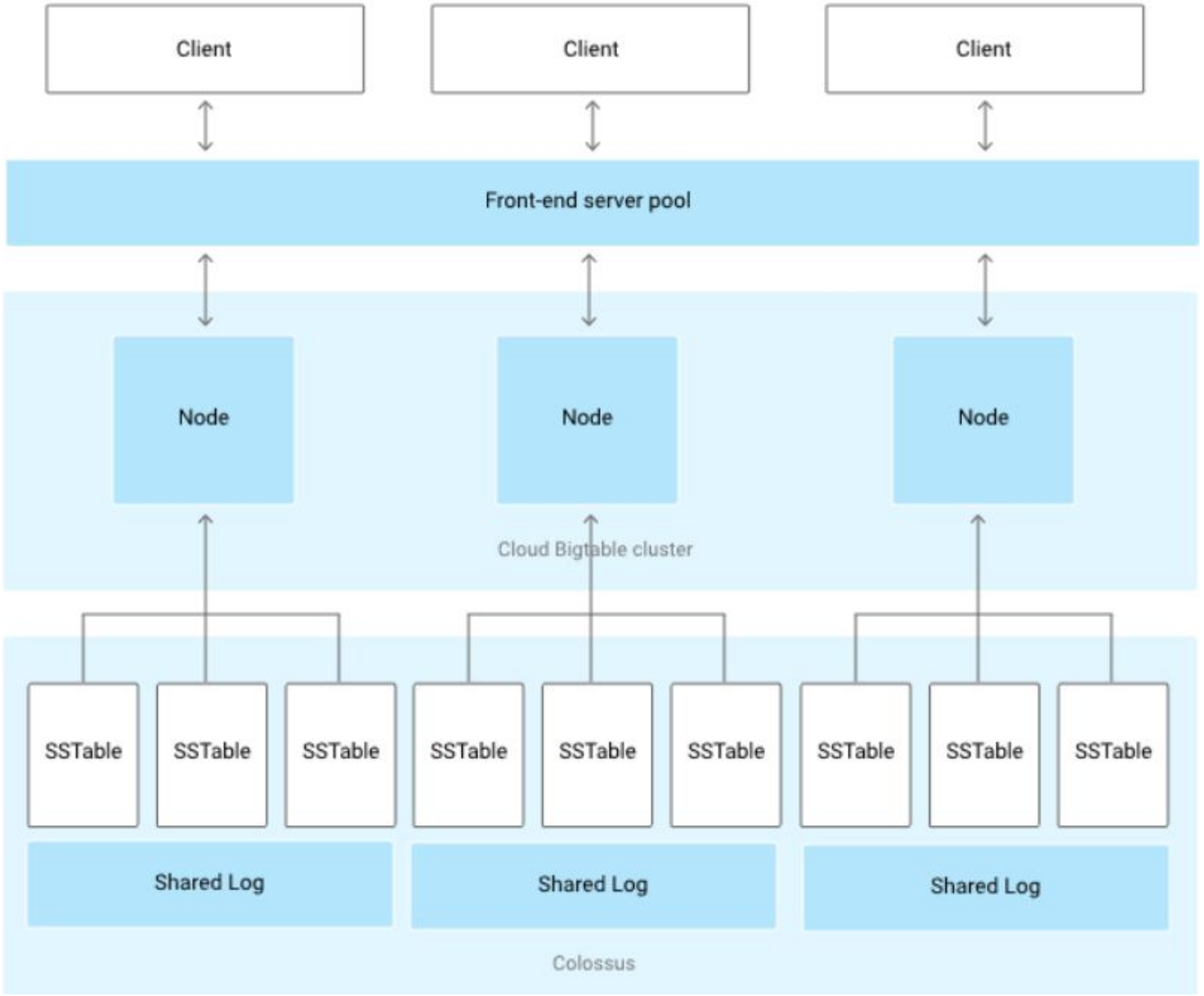
Use Cases

- Financial data – stock prices
- IoT data
- Marketing data – purchase histories

Access Control

- Project wide or instance level
- Read/Write/Manage

Cloud Bigtable Infrastructure



Cloud Bigtable : Instance Configuration

Instance basics

- Not no-ops
  - Must configure nodes
- Entire Bigtable project called 'instance'
  - All nodes and clusters
- Nodes grouped into clusters
  - 1 or more clusters per instance
- Auto-scaling storage
- Instance types
  - Development - low cost, single node
    - No replication
  - Production - 3+ nodes per cluster
    - Replication available, throughput guarantee

Replication and Changes

- Synchronize data between clusters
  - One additional cluster, total
  - (Beta) available cross-region
- Resizing
  - Add and remove nodes and clusters with no downtime
- Changing disk type (e.g. HDD to SSD) requires new instance

Interacting with Bigtable

- Command line - cbt tool or HBase shell
  - cbt tool is simpler and preferred option

Cloud Bigtable : Instance Configuration

Bigtable interaction using cbt

- Install the cbt command in Google SDK
  - sudo gcloud components update
  - gcloud components install cbt
- Configure cbt to use your project and instance via .cbtrc file'
  - echo -e "project = [PROJECT_ID]\ninstance = [INSTANCE_ID]" > ~/.cbtrc
- Create table
  - cbt createtable my-table
- List table
  - cbt ls
- Add column family
  - cbt createfamily my-table cf1
- List column family
  - cbt ls my-table
- Add value to row 1, using column family cf1 and column qualifier c1
  - cbt set my-table r1 cf1:c1=test-value
- Delete table (if not deleting instance)
  - cbt deletetable my-table
- Read the contents of your table
  - cbt read my-table

Get help with cbt command using 'cbt --help'

Cloud Bigtable : Data Organization

Data Organization

- One big table (hence the name Bigtable)
- Table can be thousands of columns/billions of rows
- Table is sharded across tablets

Table components

- Row Key
  - First column
- Columns grouped into column families

	Column-Family-1		Column-Family-2	
Row Key	*Column-Qualifier-1*	*Column-Qualifier-2*	*Column-Qualifier-1*	*Column-Qualifier-2*
r1	r1, cf1:cq1	r1, cf1:cq2	r1, cf1:cq1	r1, cf1:cq2
r2	r2, cf1:cq1	r2, cf1:cq2	r2, cf1:cq1	r2, cf1:cq2

Indexing and Queries

- Only the row key is indexed
- Schema design is necessary for efficient queries!
- Field promotion - move fields from column data to row key

Cloud Bigtable : Data Organization

Row Key
memusage+user+timestamp
20-mattu-201805082048

Schema Design

- Per table – Row key is the only indexed item
- Keep all entity info in a single row
- Related entities should be in adjacent rows
  - More efficient reads
- Tables are sparse – empty columns take no space

Schema Efficiency

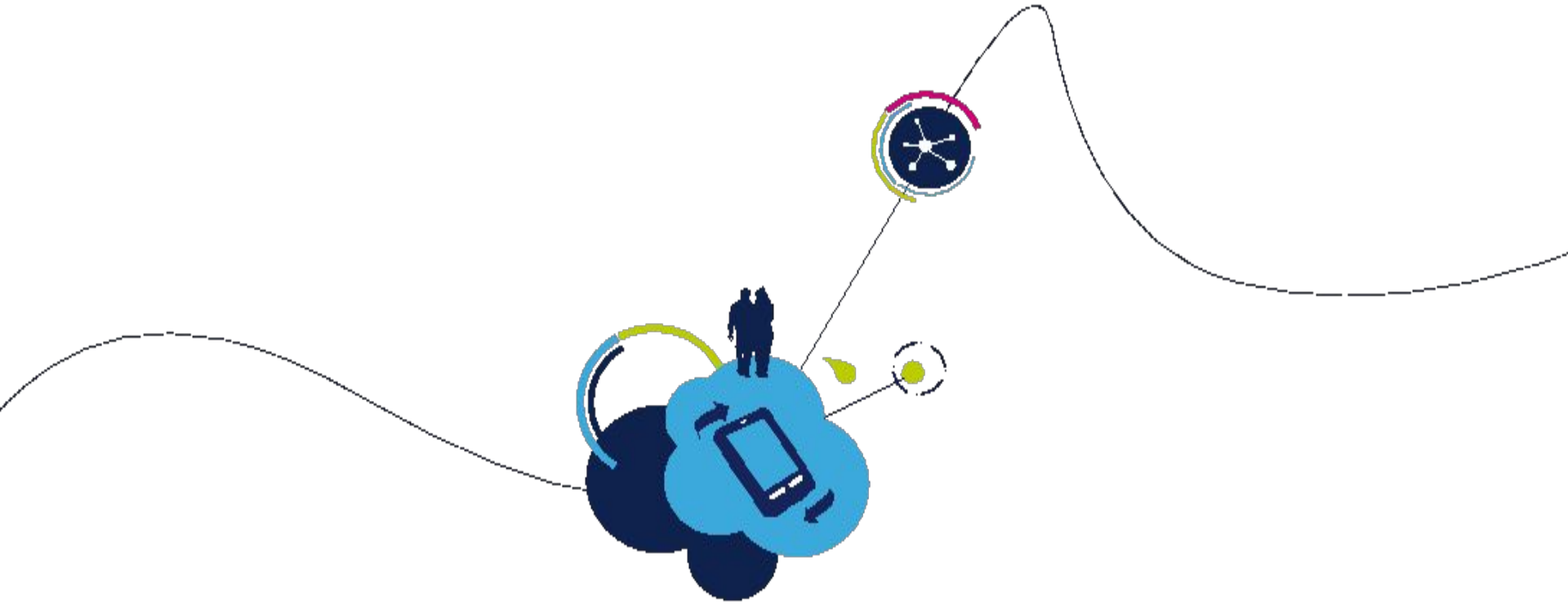
- Well-defined row keys = less work
  - Multiple values in row key
- Row key (or prefix) should be sufficient for a search
- Goal = spread loads over multiple nodes
  - All on one node = hotspotting

Row Key Best Practices

- Good row keys = distributed load
  - Reverse domain names (com.linuxacademy.support)
  - String identifiers (mattu)
  - Timestamps (reverse, NOT at front/or only identifier)
- Poor row keys = hotspotting
  - Domain names (support.linuxacademy.com)
  - Sequential ID's
  - Timestamps alone/at front

Table Design - Time Series Data

- For time series data, use tall and narrow tables (one event per row)
  - Easier to run queries against data



Cloud Spanner Overview

Cloud Spanner

What is Cloud Spanner?

- Fully managed, highly scalable/available, relational database
- Similar architecture to Bigtable
- "NewSQL"

What is it used for?

- Mission critical, relational databases that need strong transactional consistency (ACID compliance)
- Wide scale availability
- Higher workloads than Cloud SQL can support
- Standard SQL format (ANSI 2011)

Horizontal vs. vertical scaling

- Vertical = more compute on single instance (CPU/RAM)
- Horizontal = more instances (nodes) sharing the load

Compared to Cloud SQL

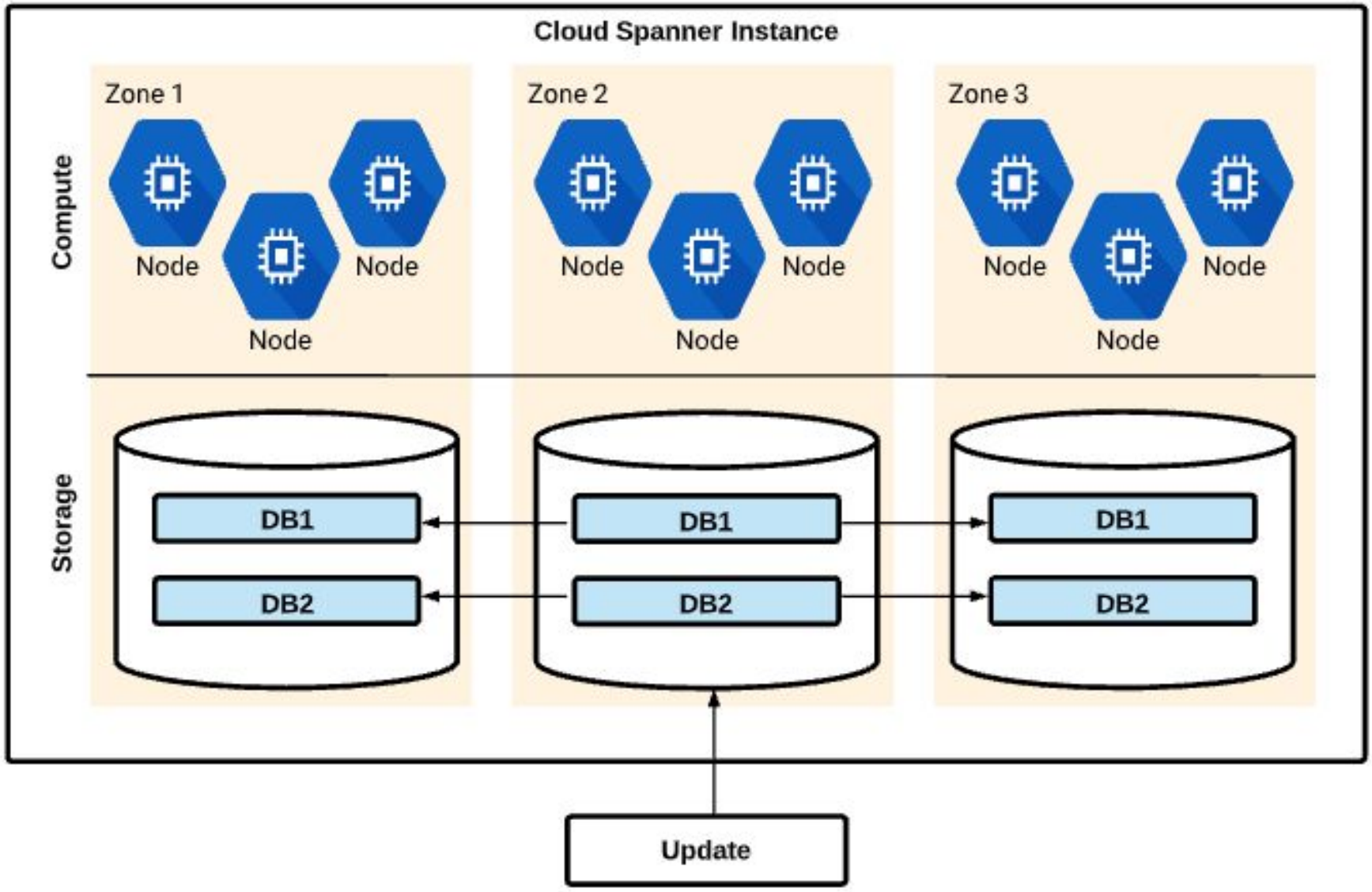
- Cloud SQL = Cloud incarnation of *on-premises* MySQL database
- Spanner = designed from the ground up for the cloud
- Spanner is not a 'drop in' replacement for MySQL
  - Not MySQL/PostreSQL compatible
  - Work required to migrate
  - However, when making transition, don't need to choose between consistency and scalability

Transactional Consistency vs. Scalability
Why not both?

	Cloud Spanner	Traditional Relational	Traditional Non-relational
Schema	Yes	Yes	No
SQL	Yes	Yes	No
Consistency	Strong	Strong	Eventual
Availability	High	Failover	High
Scalability	Horizontal	Vertical	Horizontal
Replication	Automatic	Configurable	Configurable

Primary purpose of Cloud Spanner:
No compromises relational database

Cloud Spanner Architecture
(similar to Bigtable)



Cloud Spanner

Identity and Access Management (IAM)

- Project, Instance, or Database level
- roles/spanner._____
- Admin - Full access to all Spanner resources
- Database Admin - Create/edit/delete databases, grant access to databases
- Database Reader - read/execute database/schema
- Viewer - view instances and databases
  - Cannot modify or read from database

Cloud Spanner

Organization

- RDBMS = tables
- Supports SQL joins, queries, etc
- Same SQL dialect as BigQuery
- Tables are handled differently
  - Parent/child tables
  - Interleave Data Layout

Typical Relational Database
Two sets of related data = Two tables

SingerId	SingerName
1	Beatles
2	U2
3	Pink Floyd

SingerId	AlbumId	AlbumName
1	1	Help!
1	2	Abbey Road
3	1	The Wall

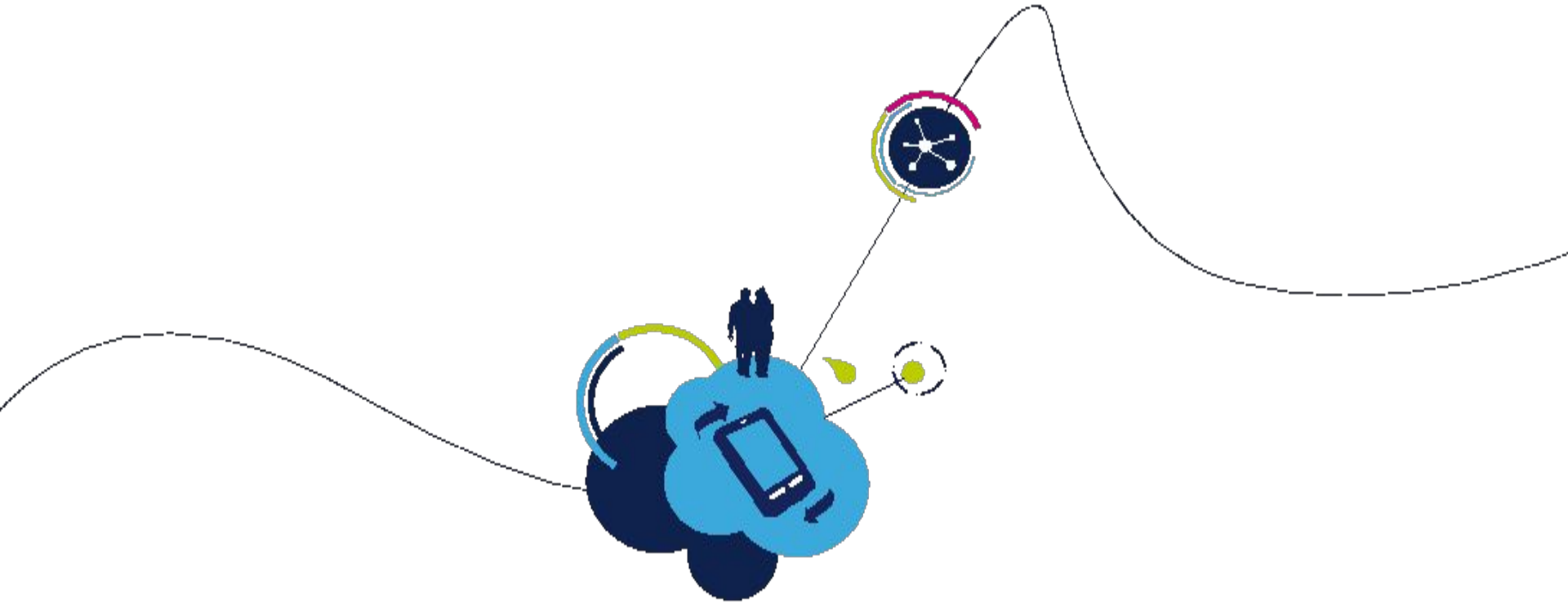
Spanner
Interleave Tables

Singers(1)	"Marc"	"Richards"	<Bytes>		
Albums(1, 1)				"Total Junk"	
Albums(1, 2)				"Go, Go, Go"	
Songs(1, 2, 1)					"42"
Songs(1, 2, 2)					"Nothing Is The Same"
Singers(2)	"Catalina"	"Smith"	<Bytes>		
Albums(2, 1)				"Green"	
Songs(2, 1, 1)					"Let's Get Back Together"
Songs(2, 1, 2)					"Starting Again"
Songs(2, 1, 3)					"I Knew You Were Magic"
Albums(2, 2)				"Forever Hold Your Peace"	
Albums(2, 3)				"Terrified"	
Songs(2, 3, 1)					"Fight Story"

Data Organization and Schema

Primary keys and Schema

- How to tell which child tables to store with which parent tables
- Usually a natural fit
  - 'Customer ID'
  - 'Invoice ID'
- Avoid hotspotting
  - No sequential numbers
  - No timestamps (also sequential)
    - Use descending order if timestamps required

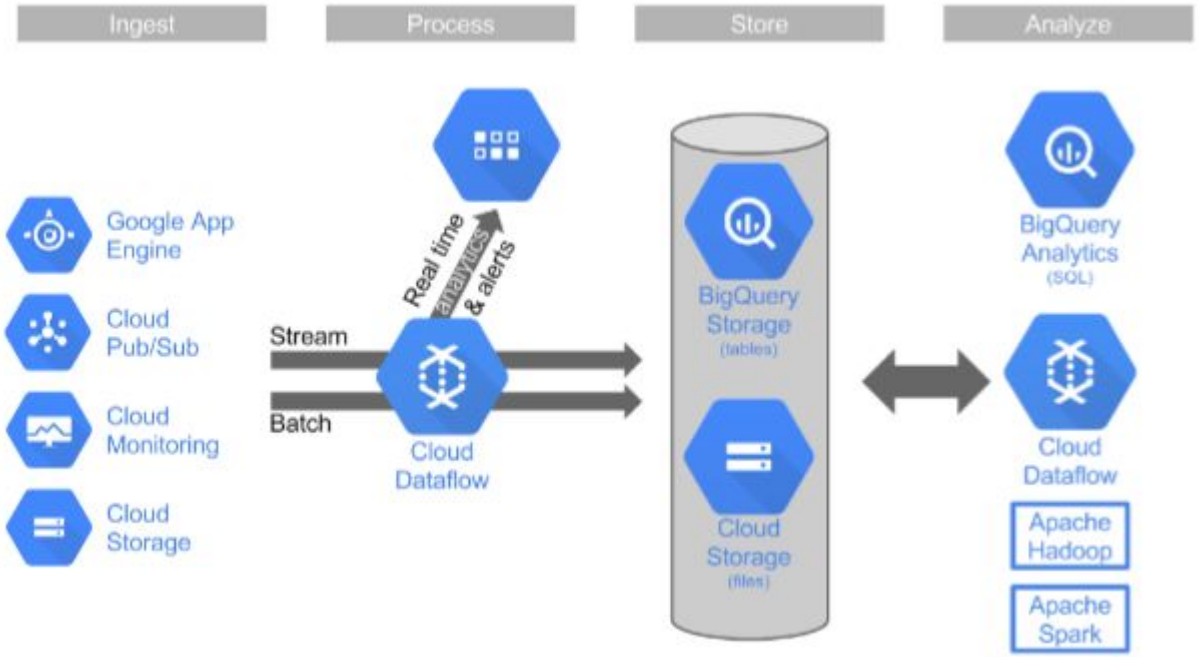


Data Analytics with BigQuery

BigQuery Overview

What is BigQuery?

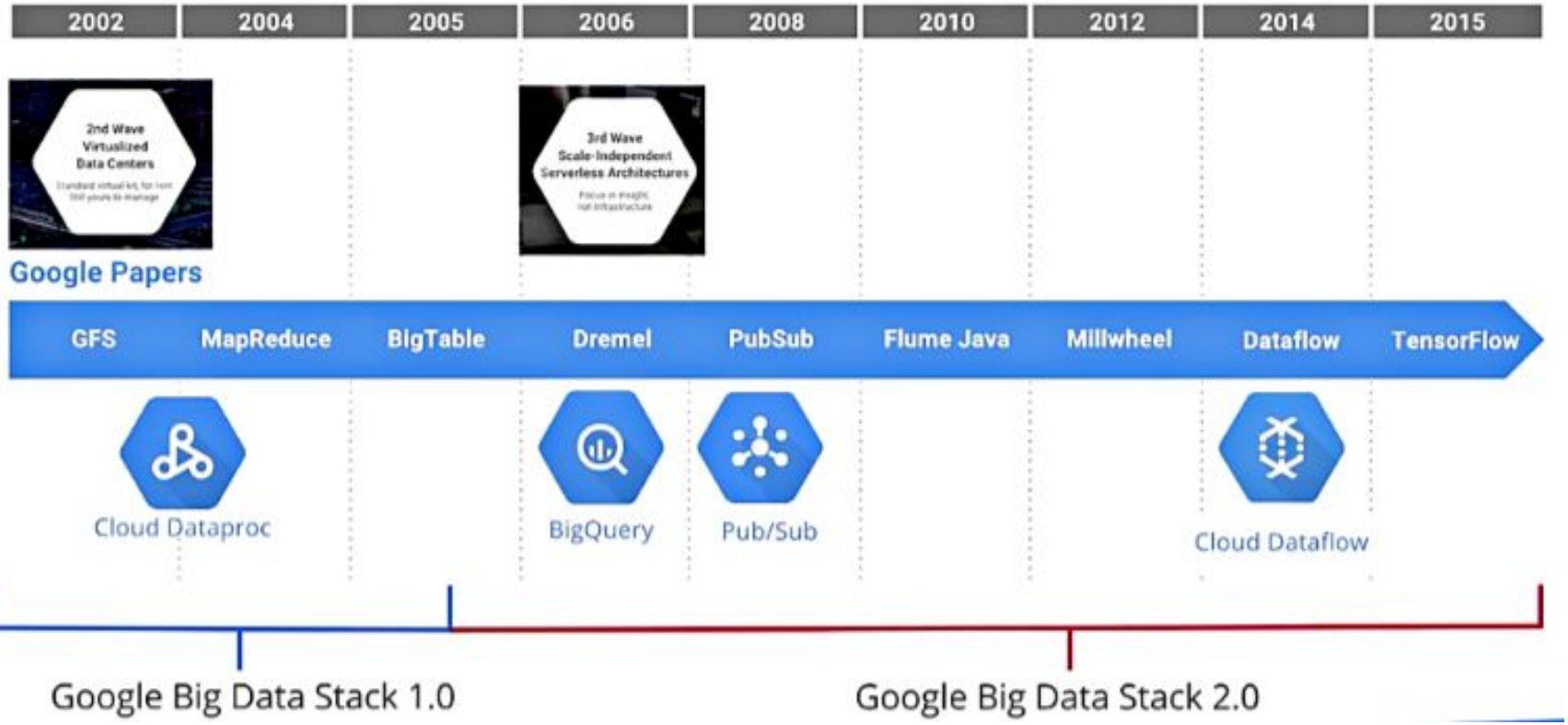
- Fully Managed Data warehousing
  - Near-real time analysis of petabyte scale databases
- Serverless (no-ops)
- Auto-scaling to petabyte range
- Both storage and analysis
- Accepts batch and streaming loads
- Locations = multi-regional (US, EU), Regional (asia-northeast1)
- Replicated, durable
- Interact primarily with standard SQL (also Legacy SQL)
  - SQL Primer course



BigQuery Overview

How BigQuery works

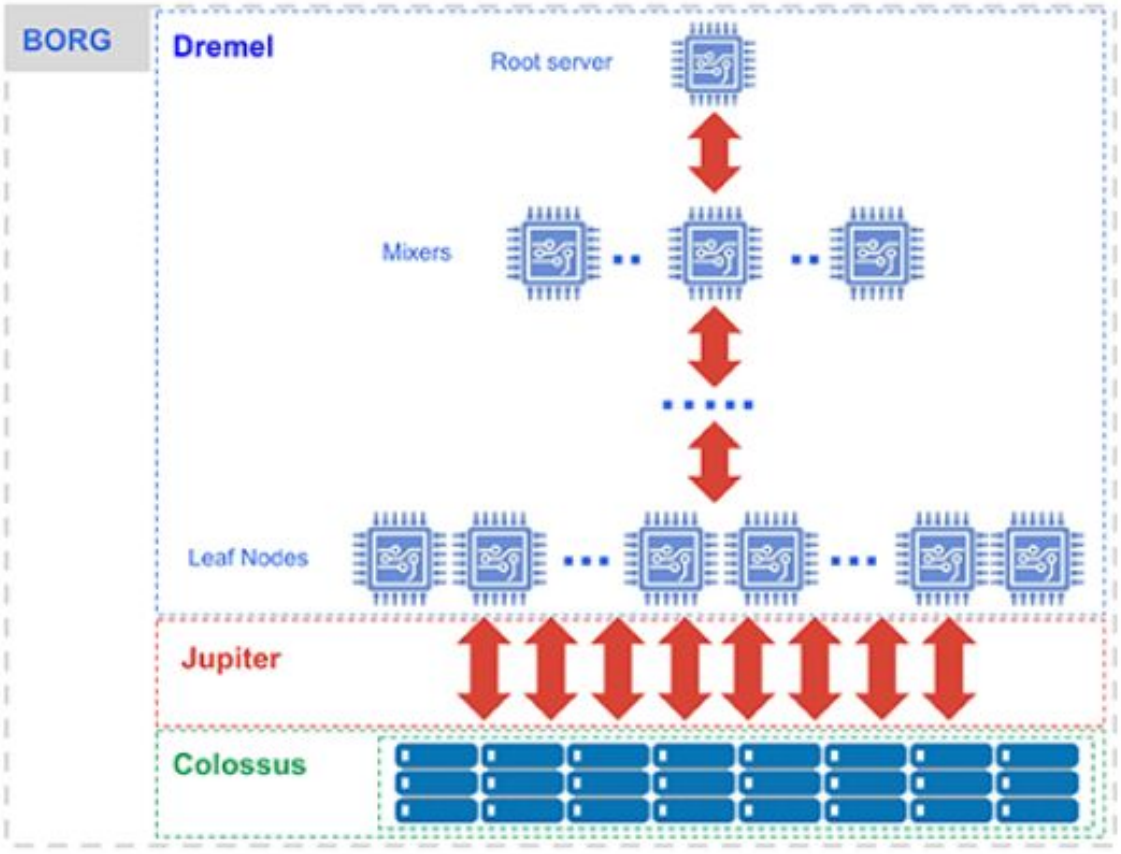
- Part of the "3rd wave" of cloud computing
  - Google Big Data Stack 2.0
- Focus on serverless compute, real time insights, machine learning...
  - ...instead of data placement, cluster configuration
  - No managing of infrastructure, nodes, clusters, etc



BigQuery Overview

How BigQuery works (cont)

- Jobs (queries) can scale up to thousands of CPU's across many nodes, but the process is completely invisible to end user
- Storage and compute are separated, connected by petabit network

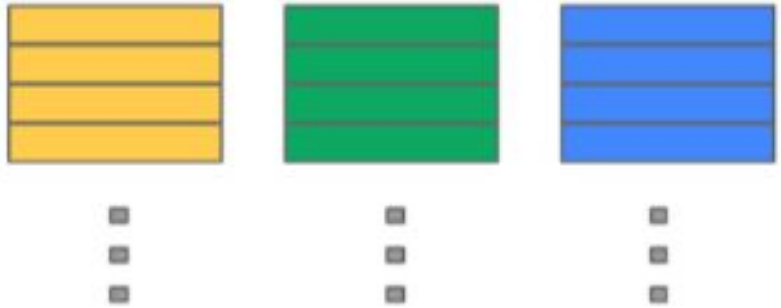


BigQuery Overview

How BigQuery works (cont)

- Columnar data store
  - Separates records into column values, stores each value on different storage volume
  - Traditional RDBMS stores whole record on one volume
  - Extremely fast read performance, poor write (update) performance - BigQuery does not update existing records
  - Not transactional

Record Oriented Storage

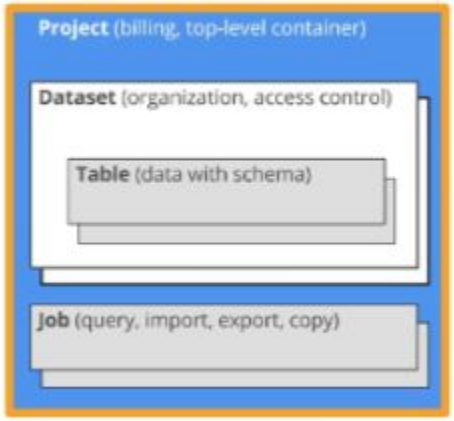


Column Oriented Storage

BigQuery Overview

BigQuery structure

- Dataset - contains tables/views
- Table = collection of columns
- Job = long running action/query



Identity and Access Management (IAM)

- Control by project, dataset, view
- Cannot control at table level
  - But can control by views via datasets as alternative (virtual table defined by SQL query)
- Predefined roles - BigQuery...
  - Admin - full access
  - Data Owner - full dataset access
  - Data Editor - edit dataset tables
  - Data Viewer - view datasets and tables
  - Job User - run jobs
  - User - run queries and create datasets (but not tables)
- Roles comparison matrix
- Sharing datasets
  - Make public with All Authenticated Users

BigQuery Overview

Pricing

- Storage, Queries, Streaming insert
- Storage = $0.02/GB/mo (first 10GB/mo free)
  - Long term storage (not edited for 90 days) = $0.01/GB/mo
- Queries = $5/TB (first TB/mo free)
- Streaming = $0.01/200 MB
- Pay as you go, with high end flat-rate query pricing
- Flat rate - starts at $40K per month with 2000 slots

Interacting with BigQuery

Interaction methods

- Web UI
- Command line (bq commands)
  - bq query --arguments 'QUERY'
- Programmatic (REST API, client libraries)
- Interact via queries

Querying tables

- FROM `project.dataset.table` (Standard SQL)
- FROM [project:dataset.table] (Legacy SQL)

Searching multiple tables with wildcards

Query across multiple, similarly named tables

- FROM `project.dataset.table_prefix*`

Filter further in WHERE clause

- AND _TABLE_SUFFIX BETWEEN 'table003' and 'table050'

Advanced SQL queries are allowed

- JOINS, sub queries, CONCAT

Interacting with BigQuery

Views

- Virtual table defined by query
- 'Querying a query'
- Contains data only from query that contains view
- Useful for limiting table data to others

Cached queries

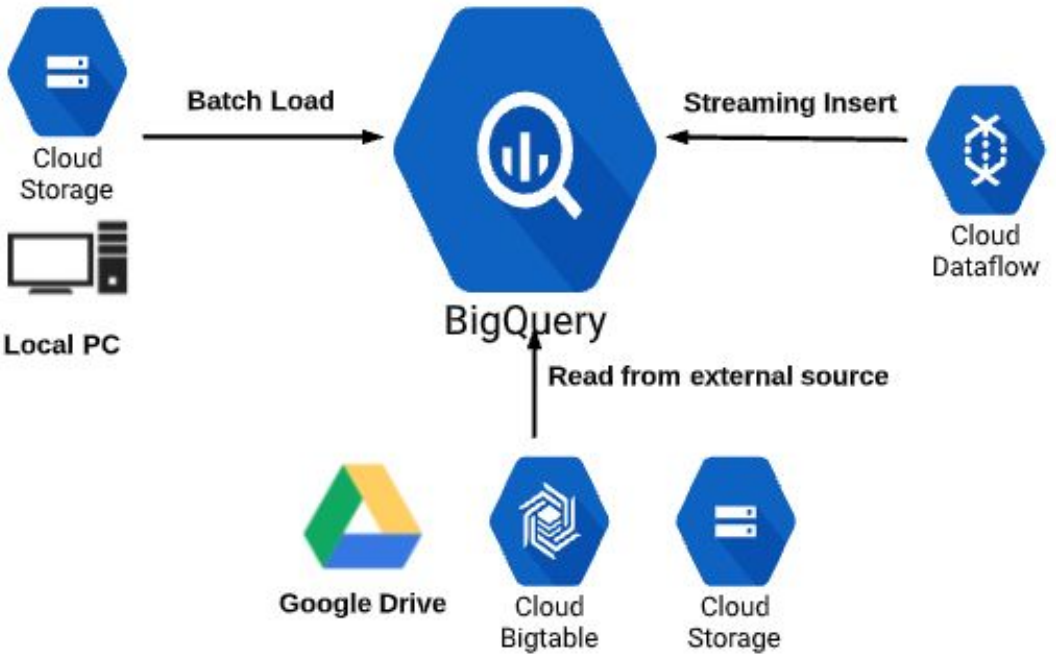
- Queries cost money
- Previous queries are cached to avoid charges if ran again
- command line to disable cached results
  - bq query --nouse_cache '(QUERY)'
- Caching is per user only

User Defined Functions (UDF)

- Combine SQL code with JavaScript/SQL functions
- Combine SQL queries with programming logic
- Allow much more complex operations (loops, complex conditionals)
- WebUI only usable with Legacy SQL

Load and Export Data

Loading and reading sources



Data formats:

Load

- CSV
- JSON (Newline delimited)
- Avro - best for compressed files
- Parquet
- Datastore backups

Read

- CSV
- JSON (Newline delimited)
- Avro
- Parquet

Why use external sources?

- Load and clean data in one pass from external, then write to BigQuery
- Small amount of frequently changing data to join to other tables

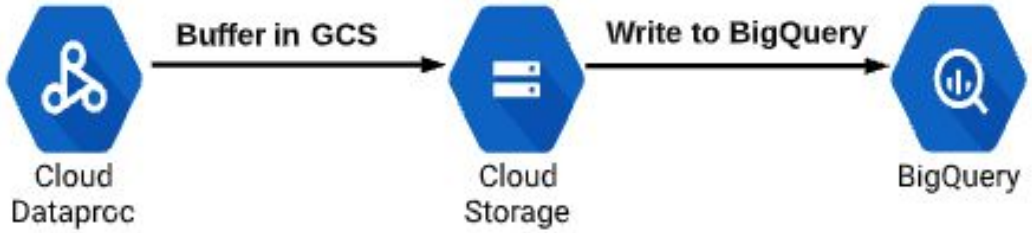
Loading data with command line

- bq load --source_format=[format] [dataset].[table] [source_path] [schema]
- Can load multiple files with command line (not WebUI)

Load and Export Data

Connecting to/from other Google Cloud services

- Dataproc - Use BigQuery connector (installed by default), job uses Cloud Storage for staging



Exporting tables

- Can only export to Cloud Storage
- Can copy table to another BigQuery dataset
- Export formats: CSV, JSON, Avro
- Can export multiple tables with command line
- Can only export up to 1GB per file, but can split into multiple files with wildcards
- Command line
  - bq extract 'projectid:dataset.table' gs://bucket_name/folder/object_name
  - Can drop 'project' if exporting from same project
  - Default is CSV, specify other format with --destination_format
  - --destination_format=NEWLINE_DELIMITED_JSON

BigQuery Transfer Service

- Import data to BigQuery from other Google advertising SaaS applications
- Google AdWords
- DoubleClick
- YouTube reports

Optimize for performance and Costs

Performance and costs are complementary

- Less work = faster query = less costs
- What is 'work'?
  - I/O - how many bytes read?
  - Shuffle - how much passed to next stage
  - How many bytes written?
  - CPU work in functions

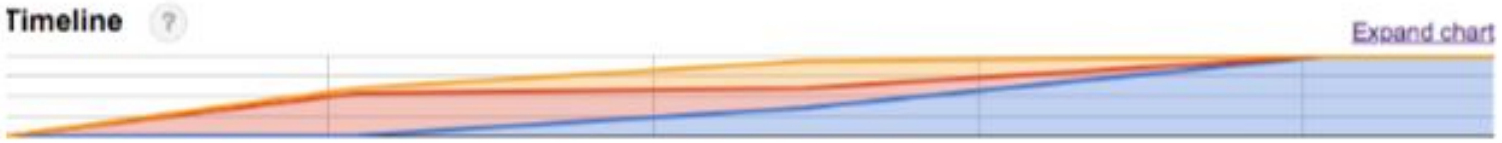
General best practices

- Avoid using SELECT *
- Denormalize data when possible
  - Grouping data into single table
  - Often with nested/repeated data
  - Good for read performance, not for write (transactional) performance
- Filter early and big with WHERE clause
- Do biggest joins first, and filter pre-JOIN
- LIMIT does not affect cost
- Partition data by date
  - Partition by ingest time
  - Parition by specified data columns

Optimize for performance and Costs

Monitoring query performance

- Understand color codes
- Understand 'skew' in difference between average and max time



Execution Plan

		Stage timing					Rows	
		Wait	Read	Compute	Write	Parallel Inputs	Input	Output
S00: Input	✓					105	122 M	5.34 K (78.2 KB)
S02: Coalesce	✓					100	5.34 K	5.34 K (78.2 KB)
S03: Join+	✓					74	22.1 M	37 (925 B)
S04: Aggregate+	✓					17	37	19 (646 B)
S05: Output	✓					1	19	19 (532 B)

Streaming Insert Example

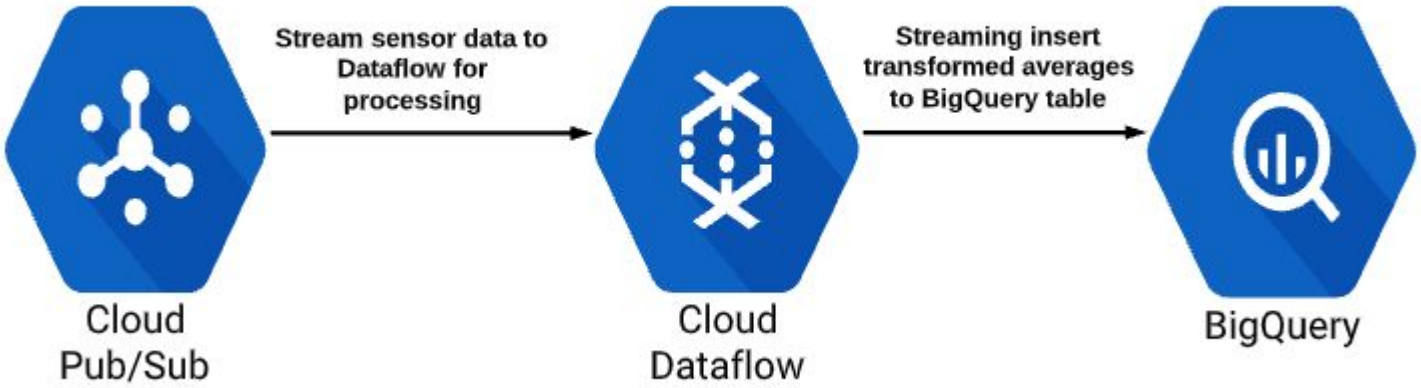
Quick setup

```
cd
gsutil cp -r gs://gcp-course-exercise-scripts/data-engineer/* .
bash streaming-insert.sh
```

Clean up

```
bash streaming-cleanup.sh
```

Manually stop Dataflow job



BigQuery Logging and Monitoring

Stackdriver Monitoring and Logging Differences

- Monitoring = performance/resources
- Logging = who is doing what
  - History of actions

Monitoring BigQuery Performance/Resources

- Monitoring = metrics, performance, resource capacity/usage (slots)
  - Query count, query times, slot utilization
  - Number of tables, stored and uploaded bytes over time
  - Alerts on metrics e.g., long query times
    - Example: Alerts when queries take more than one minute
- No data on **who** is doing what, or query details

Stackdriver Logging: "A Paper Trail"

- Logging = who is doing what
- Record of jobs and queries associated with accounts

BigQuery Best Practices

Data Format for Import

- Best performance = Avro format
- Scenario: Import multi-TB databases with millions of rows

Faster
Avro - Compressed
Avro - Uncompressed
Parquet
CSV
JSON
CSV - Compressed
JSON - Compressed
Slower

BigQuery Best Practices

Partitioned Tables

What is a partitioned table?

- Special single table
  - Divided into segments known as “partitions”

Why is this important?

- Query only certain rows (partitions) instead of entire table
  - Limits amount of read data
  - Improves performance
  - Reduces costs
- Partition types
  - Ingests time — when the data/row is created
  - Includes **TIMESTAMP** or **DATE** column
- **Scenario**: A large amount of data gets generated every day, and we need to query for only certain time periods within the same table.

Why not use multiple tables (one for each day) plus wildcards?

- Limited to 1000 tables per dataset
- Substantial performance drop vs. a single table

BigQuery Best Practices

Clustered Tables

- Taking partitioned tables “to the next level”
- Similar to partitioning, divides table reads by a specified columnn field
  - Instead of dividing by date/time, divides by field
- **Scenario**: Logistics company needs to query by tracking ID
  - Cluster by tracking ID columnn = only reading table rows with specified tracking ID’s
- Restriction: only (currently) available for partitioned tables

Slots

- Computational capacity required to run a SQL query
  - Bigger/more complex queries need more slots
- Default, on-demand pricing allocates 2000 slots
  - Only an issue for extremely complex queries, or high number of simultaneous users
  - If more than 2000 slots required, switch to **flat-rate pricing**

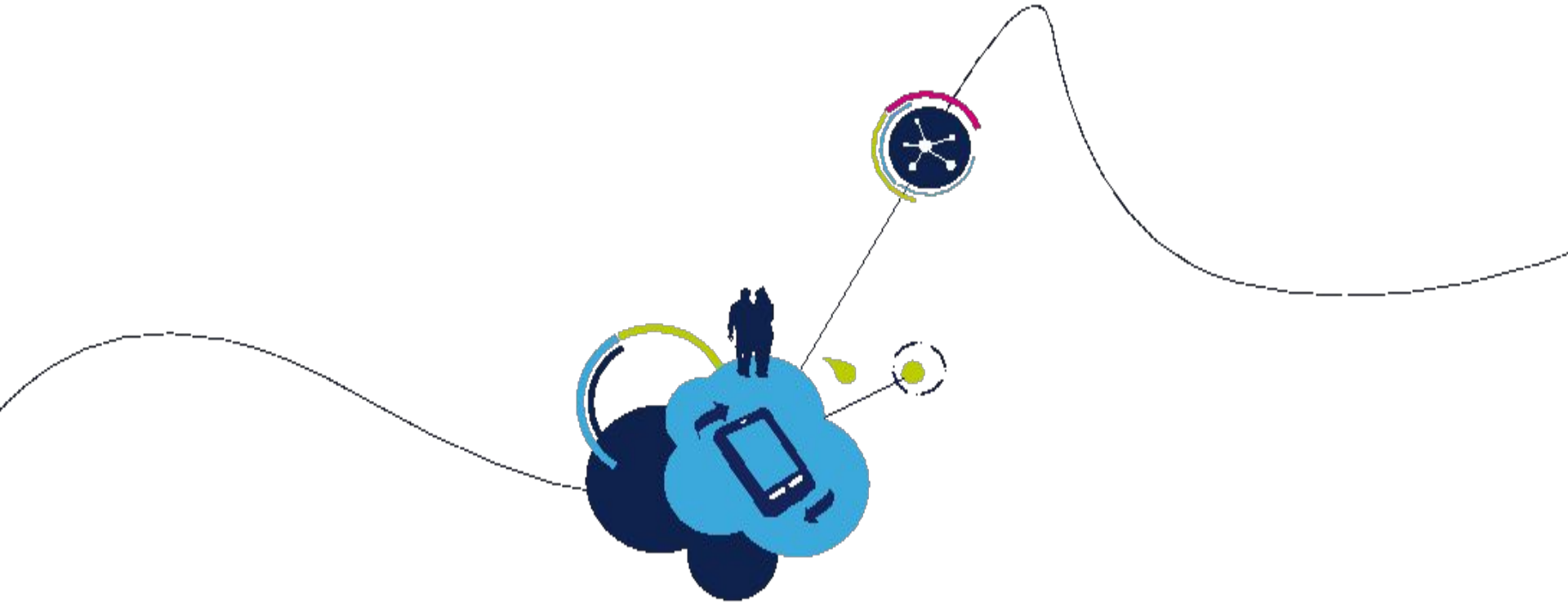
BigQuery Best Practices

Backup and Recovery

- Highly available = multi-regional dataset vs. regional
- Backup/recovery = BigQuery automatically takes continuous snapshots of tables
  - 7 day history, but 2 days if purposely deleted
- Restore to previous point in time using `@(time)`, in milliseconds
- Example: Get snapshot from one hour ago

```
#legacySQL
SELECT * FROM [PROJECT_ID:DATASET.TABLE@-3600000]
```

- Alternatively, export table data to GCS, though not as cost effective



Exploration with Cloud Datalab

Datalab Overview

How It Works

Create and connect to a Datalab instance

datalab create (instance-name) →

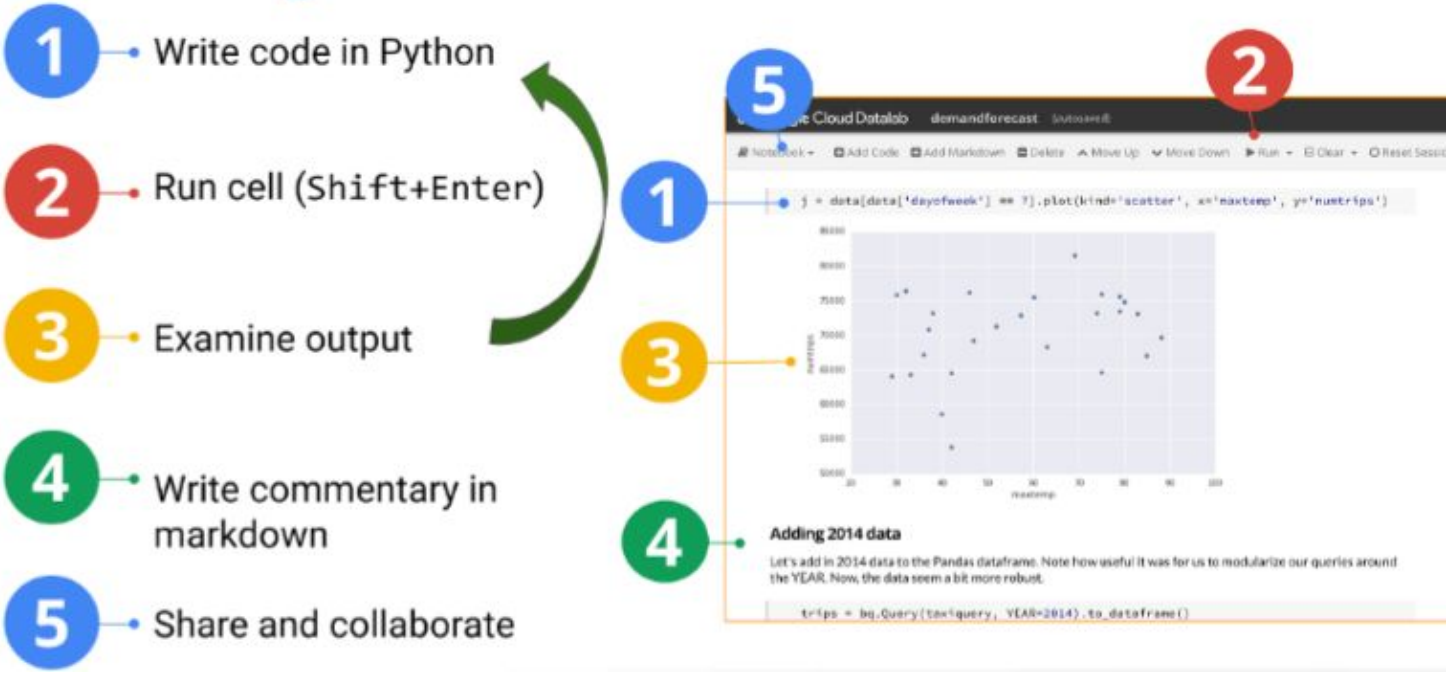
- Connect via SSH and open web preview
- datalab connect (instance-name)
- Open web preview - port 8081

datalab-network

datalab-instance

datalab-notebooks
Source repository

Working with Datalab



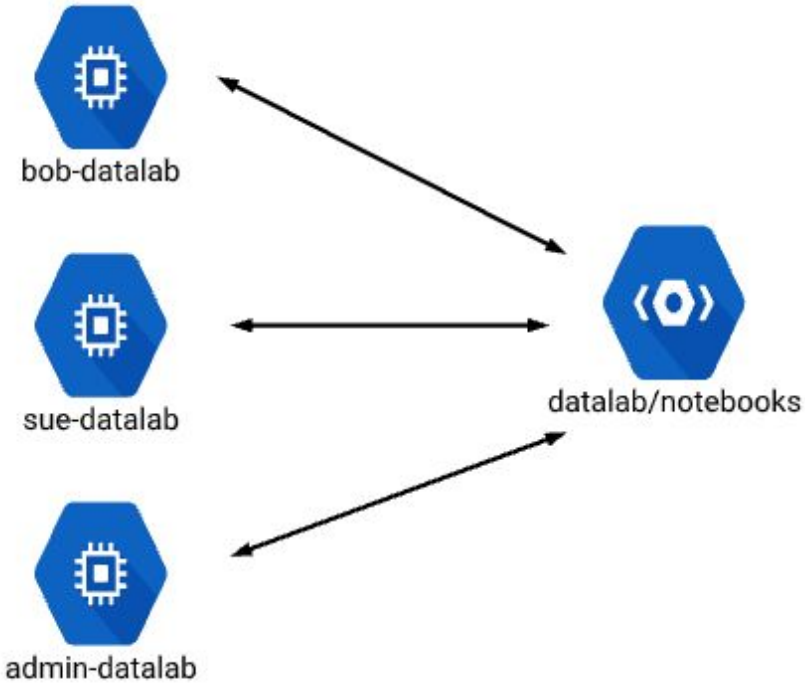
Datalab Overview

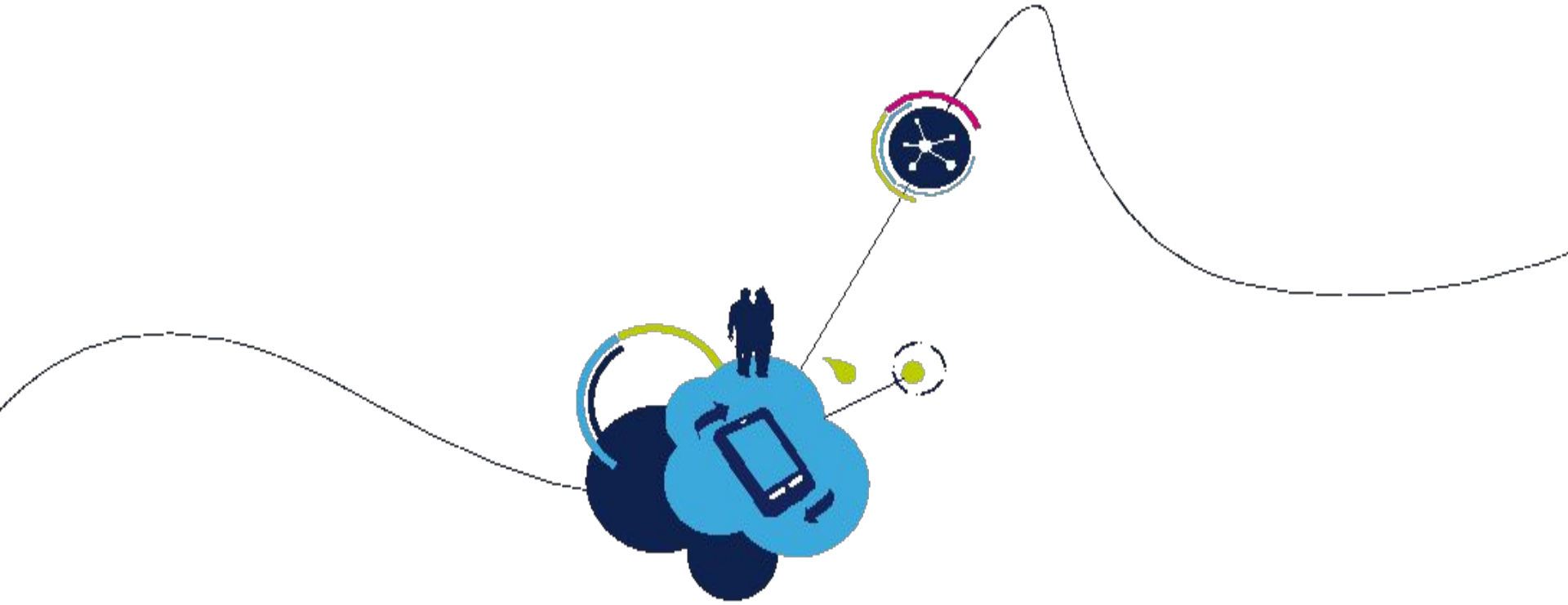
Sharing notebook data:

- GCE access based on GCE IAM roles:
  - Must have Compute Instance Admin and Service Account Actor roles
- Notebook access per user only
- Sharing data performed via shared Cloud Source Repository
- Sharing is at the project level

Creating team notebooks - two options:

- Team lead creates notebooks for users using --for user option:
  - datalab create [instance] --for-user bob@professionalwireless.net
- Each user creates their own datalab instance/notebook
- Everyone accesses same shared repository of datalab/notebooks



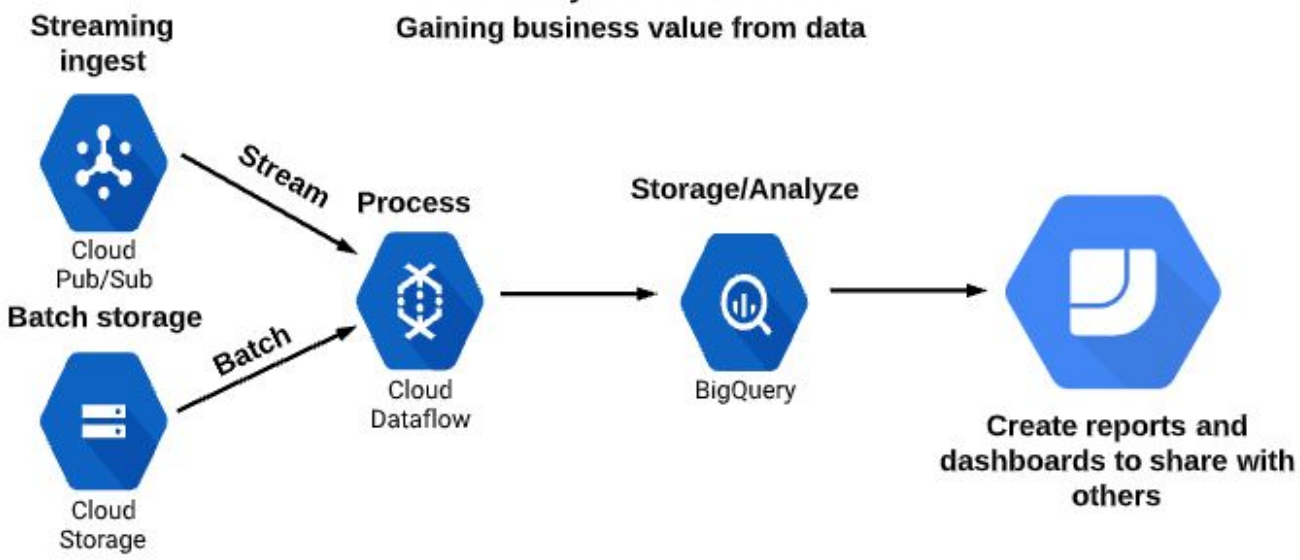


Visualization with Cloud Data Studio

Data Studio Introduction

What is Data Studio?

- Easy to use data visualization and dashboards:
  - Drag and drop report builder
- Part of G Suite, not Google Cloud:
  - Uses G Suite access/sharing permissions, not Google Cloud (no IAM)
  - Google account permissions in GCP will determine data source access
  - Files saved in Google Drive
- Connect to many Google, Google Cloud, and other services:
  - BigQuery, Cloud SQL, GCS, Spanner
  - YouTube Analytics, Sheets, AdWords, local upload
  - Many third party integrations
- Price - *Free*:
  - BigQuery access run normal query costs



Data Studio Introduction

Basic process

- Connect to data source
- Visualize data
- Share with others

Creating charts

- Use combinations of dimensions and metrics
- Create custom fields if needed
- Add date range filters with ease

Caching - options for using cached data performance/costs

Two cache types, query cache and prefetch cache

Query cache:

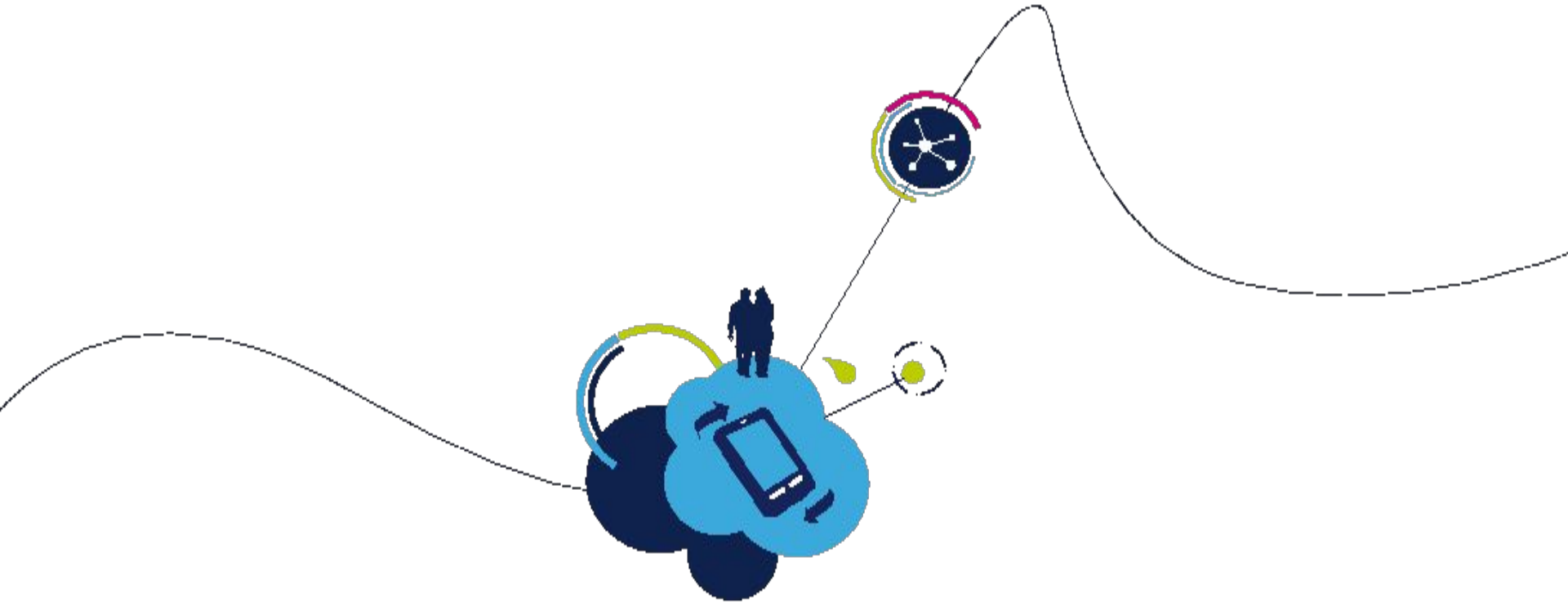
- Remembers queries issues by reports components (i.e. charts)
- When performing same query, pulls from cache
- If query cache cannot help, goes to prefetch cache
- Cannot be turned off

Prefetch cache:

- 'Smart cache' - predicts what 'might' be requested
- If prefetch cache cannot serve data, pulls from live data set
- Only active for data sources that use owner's credentials for data access
- Can be turned off

When to turn caching off:

- Need to view 'fresh data' from rapidly changing data set



Orchestration with Cloud Composer

Cloud Composer overview

What is Cloud Composer?

- Fully managed **Apache Airflow** implementation:
  - Infrastructure/OS handled for you

What is Apache Airflow?

- Programatically create, schedule, and monitor data workflows

Why is this important?

- Automation and monitoring
- Big data pipelines are often a multi-step, complex process:
  - Create resources in multiple services
  - Process and move data from one service to another
  - Remove resources when they complete a task
- Collaborate workflow process with other team members

How Airflow/Composer helps

- Automates the above steps, including scheduling
- Built on open source, using Python as common language
- Easy to work with, and share workflow with others
- Works with non-GCP providers (on-prernises, other clouds)

Cloud Composer overview

How It Works

Behind the scenes:

- GKE cluster with Airflow implemented
- Cloud Storage bucket for workflow files (and other application files)

Workflows?

- Orchestrate data pipelines:
  - Like a walkthrough of tasks to run
- Format = Direct Acyclic Graph (DAG):
  - Written in Python
  - Collection of organized tasks that you want to schedule and run
- **Cloud Composer** creates **workflows** using **DAG** files

The Process

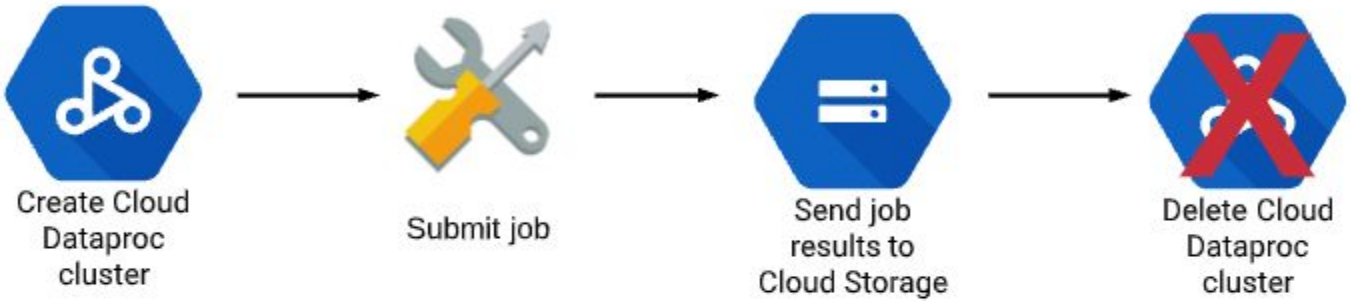
- Create Composer Environment
- Set Composer variables (i.e. project ID, GCS bucket, region)
- Add Workflows (DAG files), which Composer will execute

Hands On - Cloud Composer

The Process:

- Create the Composer environment.
- Then create the GCS bucket for Dataproc output.
- Assign Cloud Composer variables.
- Upload the workflow file to DAG folder.
- View the results.

Automatic processes -- Workflow



Create Composer Environment

- Enable Composer/Dataproc API
- Create environment in closest region:
  - What's happening?
  - Creating GKE cluster + GCS bucket

Create GCS bucket to output Dataproc results

- `gsutil mb -l us-central1 gs://output-$DEVSHELL_PROJECT_ID`

Hands On - Cloud Composer

Configure Cloud Composer Variables

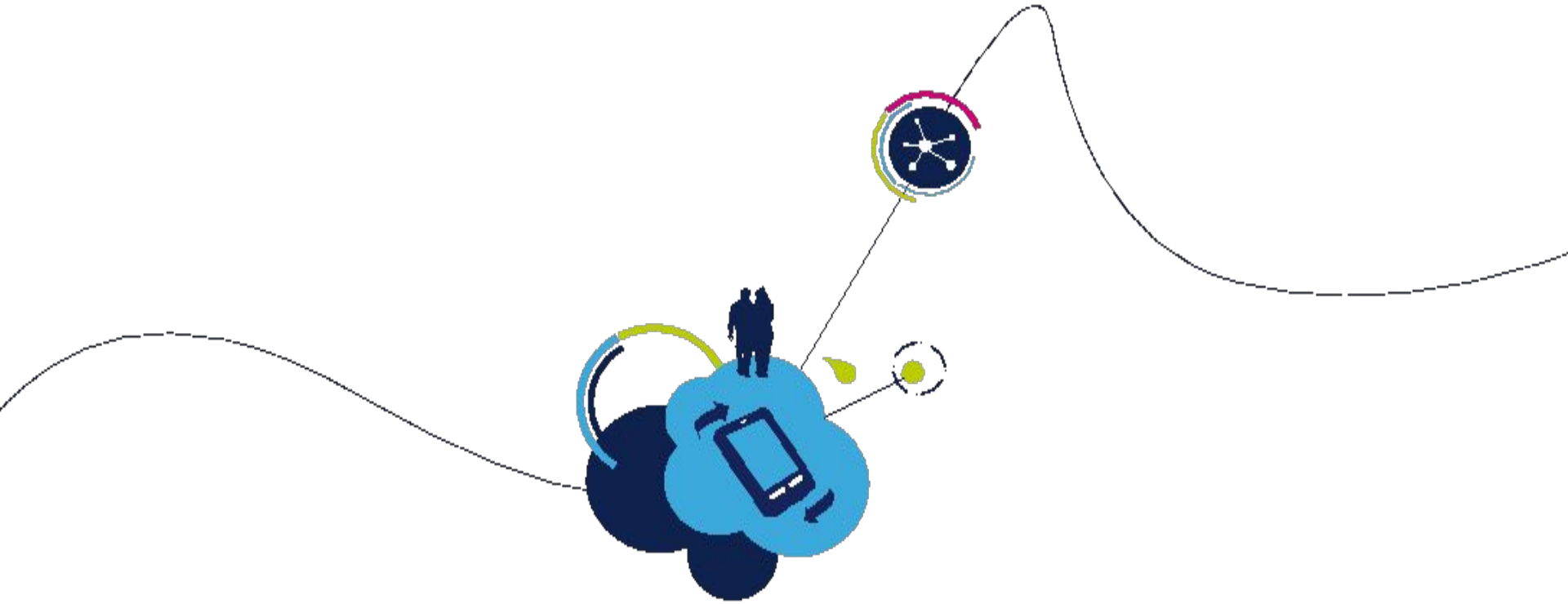
- Format
  - **gcloud composer environments run (ENVIRONMENT_NAME) --location (LOCATION) variables -- --set (KEY VALUE)**
- gcloud composer environments run my-environment --location us-central1 variables -- --set gcp_project (PROJECT-ID)
- gcloud composer environments run my-environment --location us-central1 variables -- --set gcs_bucket gs://output-(PROJECT-ID)
- gcloud composer environments run my-environment --location us-central1 variables -- --set gce_zone us-central1-c

Add workflow file (Python) to Composer DAG folder:

- github link

Next step? There is none! Cloud Composer will take it from here...

https://github.com/GoogleCloudPlatform/python-docs-samples/blob/b80895ed88ba86fce223df27a48bf481007ca708/composer/workflows/quickstart.py



Thank You