Google Cloud GO-JEK

GO-TALK SERIES

GO-JEK's Data Architecture

SPEAKER

Naoya Moritani - Solution Engineer, Google Cloud
Johanes Alexander - Solution Architect, GO-JEK

Thursday, 13 July 2017 | 18.30 PM- 21.00 PM

GO-LEARN Audiotorium | GO-JEK Office HQ, Pasaraya Blok M 7th Floor

GOJEK

Welcome

Google Cloud

Today’s discussion

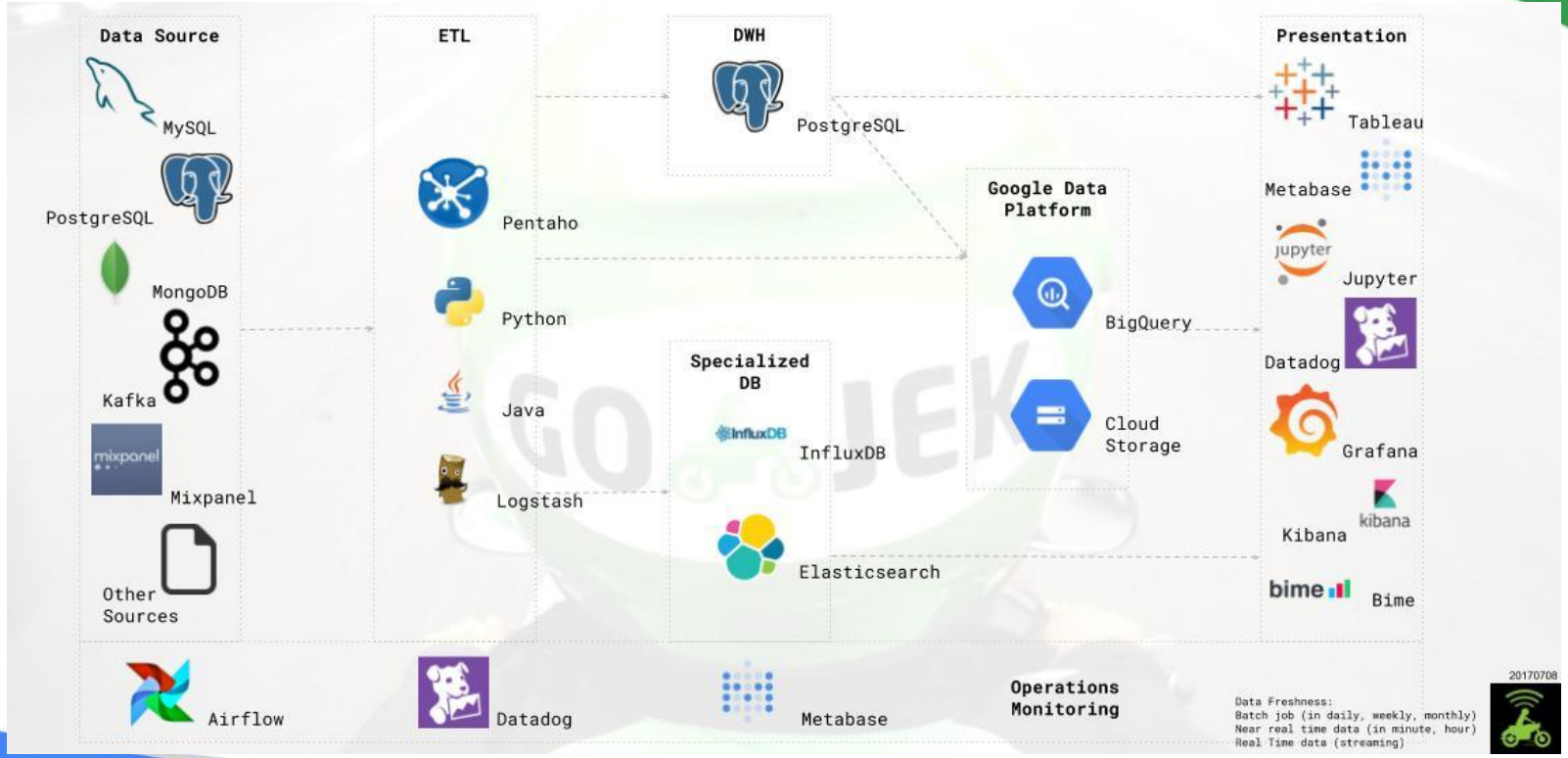
- How data warehouse is designed at GO-JEK
- What business challenges we had with current design
- Open up new possibilities with Google Cloud Data Platform
- How we are remodeling our data to fulfill business needs
- How we made a difference with a well-designed data model

GO JEK

How we design it?

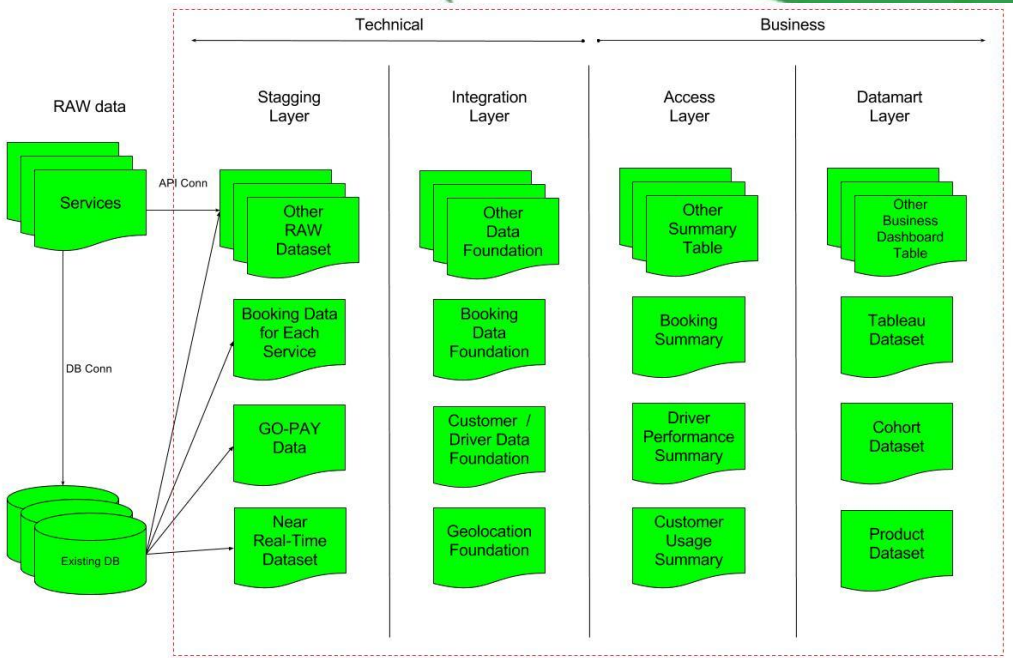
Google Cloud

BI Infrastructure Stack



Current Data Architecture

Staging Layer
RAW Dataset
Integration Layer
Fact / Dimension Dataset
Access Layer
Summary and roll-up data
Datamart Layer
Product-specialized dataset



GO JEK

It’s just not good enough.

Google Cloud

The Need to Change

Why?

~27%

Growing Data Volume per Month

**This is only business metrics data collected by BI.*

> 100

Low Resolution Dataset

**Contains reusable summary and roll-up dataset.*

> 4000

Metabase Cards & Tableau Sheets

> 400

Average Daily Metabase & Tableau Users

**Everyone just loves data!*

The Need to Change

- ***What we want***
  - High performance, scalable, minimum operation maintenance.
  - Full resolution dataset.
  - Easy data discovery process.
- ***Challenges***
  - Need a powerful data warehouse.
  - Need to remodel the entire data and process.
  - Need versatile data discovery tools.

GOJEK

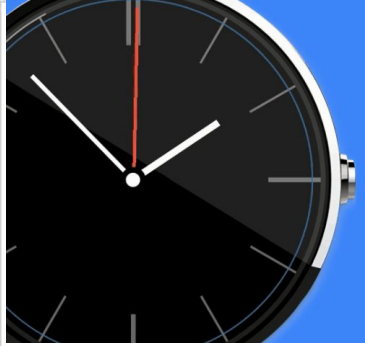
At Google we have a special term for Big Data.

Google Cloud

GOJEK

Data

Google Cloud



Data is surging **every minute**

How are you using it?

142,361,111
Emails sent and received

2,083,333
Minutes used on Skype calls

347,222
Tweets posted

$203,579
In Amazon sales generated

1,389
Uber rides taken

50,200
Mobile apps downloaded

400
Hours of video uploaded on YouTube

120
LinkedIn accounts created

2.4 Million
Google searches made

216,000
Photos posted to Instagram

Google Cloud

source: www.arbornetworks.com, VidCon 2015


Google Cloud

GO JEK

Google Cloud

GO JEK

Jupiter

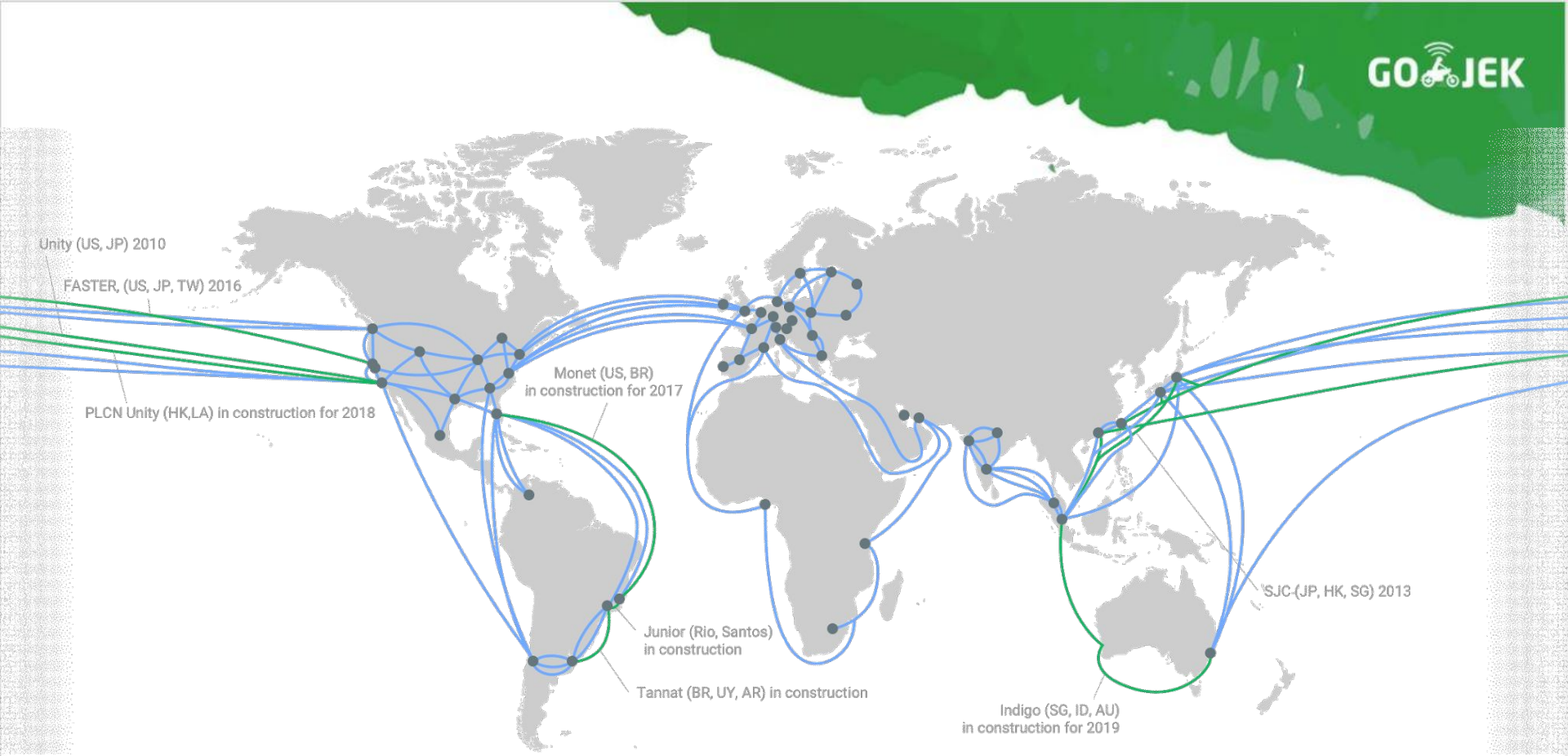
Network fabric hardware

designed by Google

10G x 100K ports = **1 Pbps**

Consolidates servers with

microsecond latency

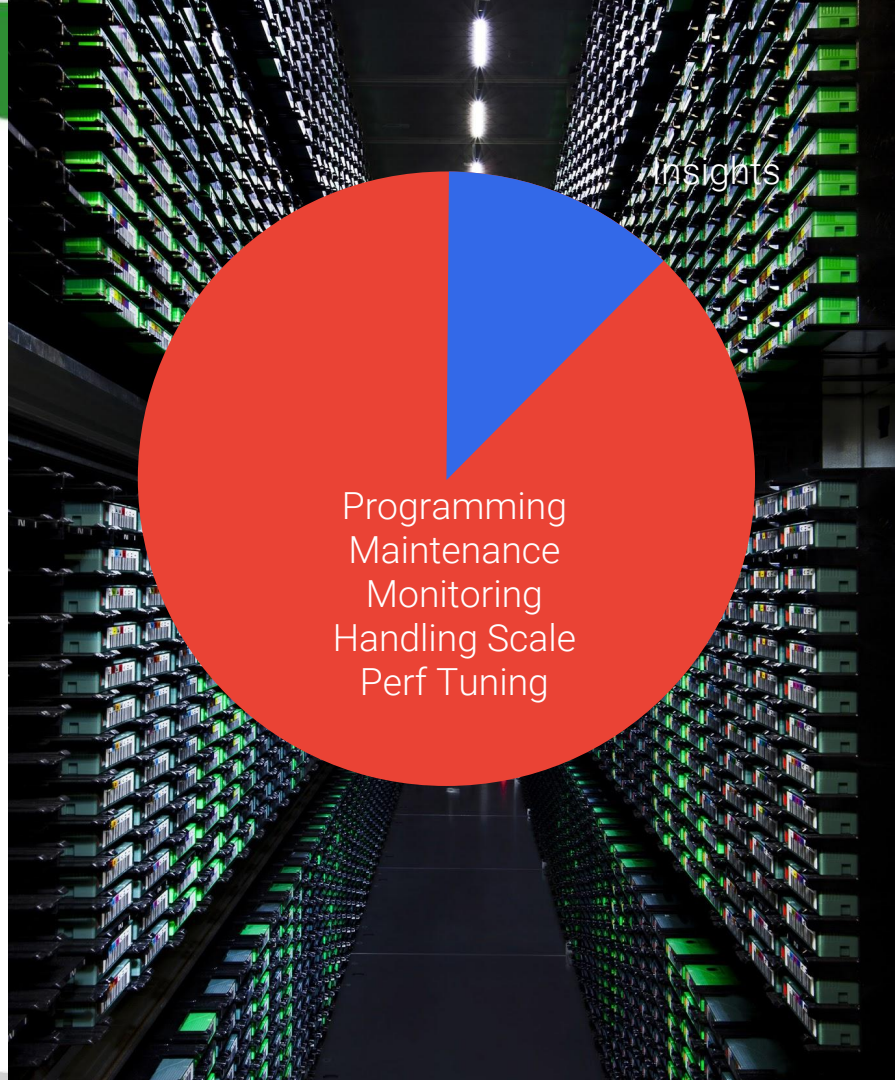

GOJEK
Unity (US, JP) 2010
FASTER, (US, JP, TW) 2016
PLCN Unity (HK,LA) in construction for 2018
Monet (US, BR)
in construction for 2017
Junior (Rio, Santos)
in construction
Tannat (BR, UY, AR) in construction
SJC (JP, HK, SG) 2013
Indigo (SG, ID, AU)
in construction for 2019

Typical Data & Analytics

Difficult to deploy & maintain

Too much time spent taking care of the system

Not enough time spent getting insights

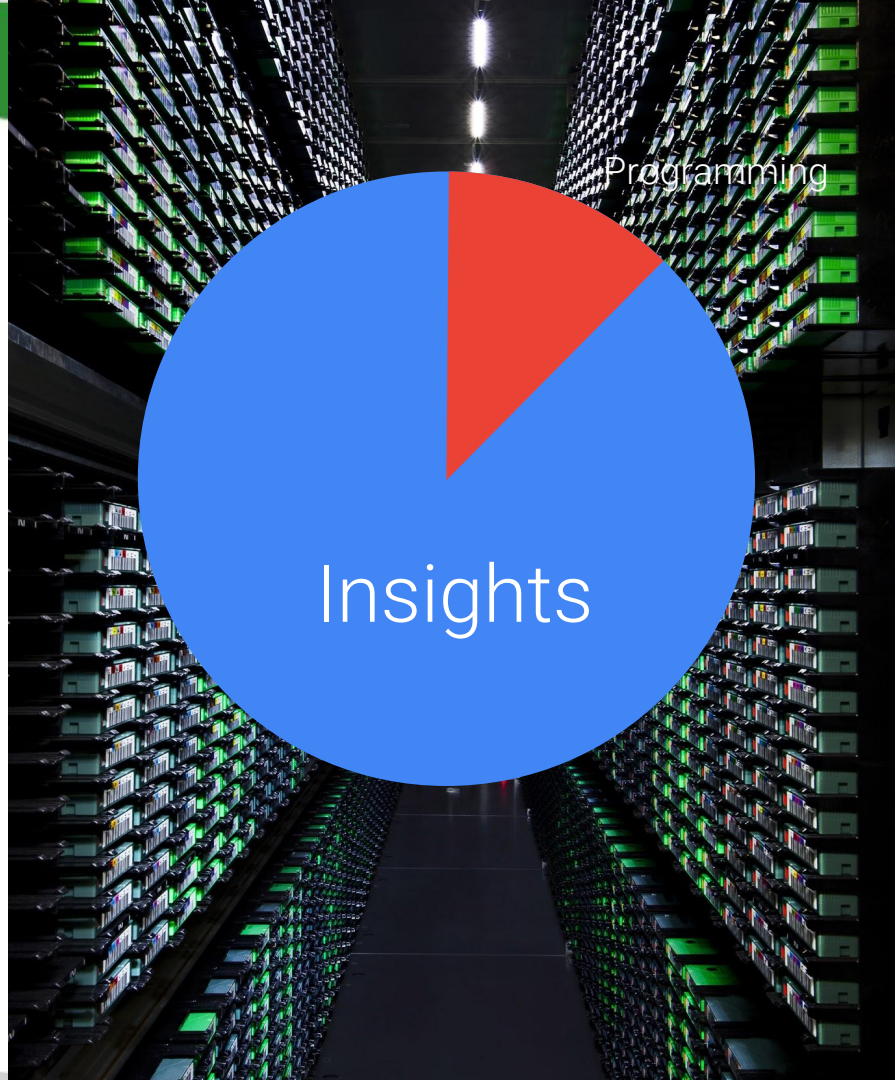


Google Data & Analytics

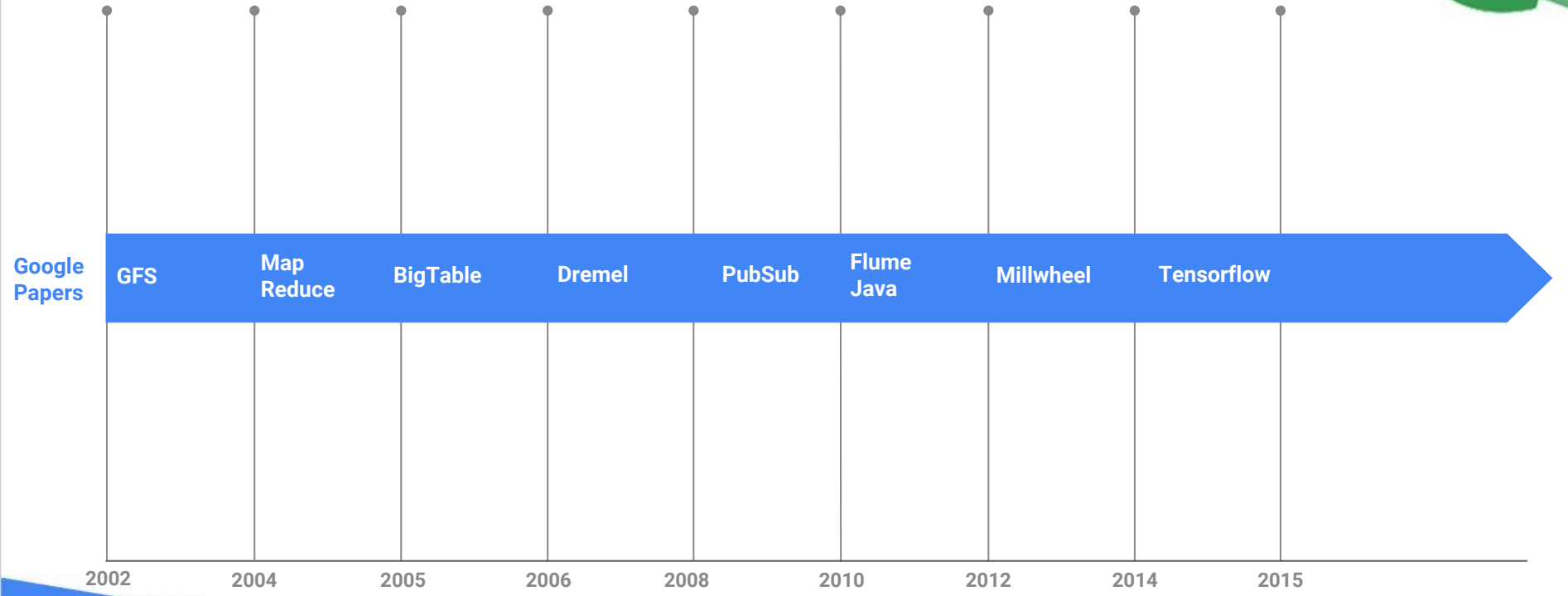
Focus on insights, not infrastructure

From batch to streaming

Analytics, Transactions, and Data Warehousing.



10+ Years of Tackling Big Data Problems



Advancing the state of the art...

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

1 Introduction

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.

The major contributions of this work are a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

Section 2 describes the basic programming model and gives several examples. Section 3 describes an implementation of the MapReduce interface tailored towards our cluster-based computing environment. Section 4 describes several refinements of the programming model that we have found useful. Section 5 has performance measurements of our implementation for a variety of tasks. Section 6 explores the use of MapReduce within Google including our experiences in using it as the basis

To appear in OSDI 2004

2004

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

1 Introduction

Over the last two and a half years we have designed, implemented, and deployed a distributed storage system for managing structured data at Google called Bigtable. Bigtable is designed to reliably scale to petabytes of data and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users. The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data.

In many ways, Bigtable resembles a database: it shares many implementation strategies with databases. Parallel databases [14] and main-memory databases [13] have achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Section 4 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 5 describes the fundamentals of the Bigtable implementation, and Section 6 describes some of the refinements that we made to improve Bigtable's performance. Section 7 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 8, and discuss some lessons we learned in designing and supporting Bigtable in Section 9. Finally, Section 10 describes related work, and Section 11 presents our conclusions.

2 Data Model

A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

(row:string, column:string, time:int64) → string

To appear in OSDI 2006

2005

Dremel: Interactive Analysis of Web-Scale Datasets

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer,
Shiva Shivakumar, Matt Tolton, Theo Vassilakis
Google, Inc.
{melnik,andrey,jlong,gromer,shiva,mtolton,theov}@google.com

ABSTRACT

Dremel is a scalable, interactive ad-hoc query system for analysis of read-only nested data. By combining multi-level execution trees and columnar data layout, it is capable of running aggregation queries over trillion-row tables in seconds. The system scales to thousands of CPUs and petabytes of data, and has thousands of users at Google. In this paper, we describe the architecture and implementation of Dremel, and explain how it complements MapReduce-based computing. We present a novel columnar storage representation for nested records and discuss experiments on few-thousand node instances of the system.

1. INTRODUCTION

Large-scale analytical data processing has become widespread in web companies and across industries, not least due to low-cost storage that enabled collecting vast amounts of business-critical data. Putting this data at the fingertips of analysts and engineers has grown increasingly important; interactive response times often make a qualitative difference in data exploration, monitoring, online customer support, rapid prototyping, debugging of data pipelines, and other tasks.

Performing interactive data analysis at scale demands a high degree of parallelism. For example, reading one terabyte of compressed data in one second using today's commodity disks would require tens of thousands of disks. Similarly, CPU-intensive queries may need to run on thousands of cores to complete within seconds. At Google, massively parallel computing is done using shared clusters of commodity machines [5]. A cluster typically hosts a multitude of distributed applications that share resources, have widely varying workloads, and run on machines with different hardware parameters. An individual worker in a distributed application may take much longer to execute a given task than others, or may never complete due to failures or preemption by the cluster management system. Hence, dealing with stragglers and failures is essential for achieving fast execution and fault tolerance [10].

The data used in web and scientific computing is often non-relational. Hence, a flexible data model is essential in these domains. Data structures used in programming languages, messages exchanged by distributed systems, structured documents, etc. lend themselves naturally to a *nested* representation. Normalizing and recombining such data at web scale is usually prohibitive. A nested data model underlies most of structured data processing at Google [21] and reportedly at other major web companies.

This paper describes a system called Dremel[1] that supports interactive analysis of very large datasets over shared clusters of commodity machines. Unlike traditional databases, it is capable of operating on *in situ* nested data. *In situ* refers to the ability to access data "in place", e.g., in a distributed file system (like GFS [14]) or another storage layer (e.g., Bigtable [8]). Dremel can execute many queries over such data that would ordinarily require a sequence of MapReduce (MR [12]) jobs, but at a fraction of the execution time. Dremel is not intended as a replacement for MR and is often used in conjunction with it to analyze outputs of MR pipelines or rapidly prototype larger computations.

Dremel has been in production since 2006 and has thousands of users within Google. Multiple instances of Dremel are deployed in the company, ranging from tens to thousands of nodes. Examples of using the system include:

- Analysis of crawled web documents.
- Tracking install data for applications on Android Market.
- Crash reporting for Google products.
- OCR results from Google Books.
- Spam analysis.
- Debugging of map tiles on Google Maps.
- Tablet migrations in managed Bigtable instances.
- Results of tests run on Google's distributed build system.
- Disk I/O statistics for hundreds of thousands of disks.
- Resource monitoring for jobs run in Google's data centers.
- Symbols and dependencies in Google's codebase.

Dremel builds on ideas from web search and parallel DBMSs. First, its architecture borrows the concept of a serving tree used in distributed search engines [11]. Just like a web search request, a query gets pushed down the tree and is rewritten at each step. The result of the query is assembled by aggregating the replies received from lower levels of the tree. Second, Dremel provides a high-level, SQL-like language to express ad hoc queries. In contrast to layers such as Pig [18] and Hive [16], it executes queries natively without translating them into MR jobs.

Lastly, and importantly, Dremel uses a column-striped storage representation, which enables it to read less data from secondary

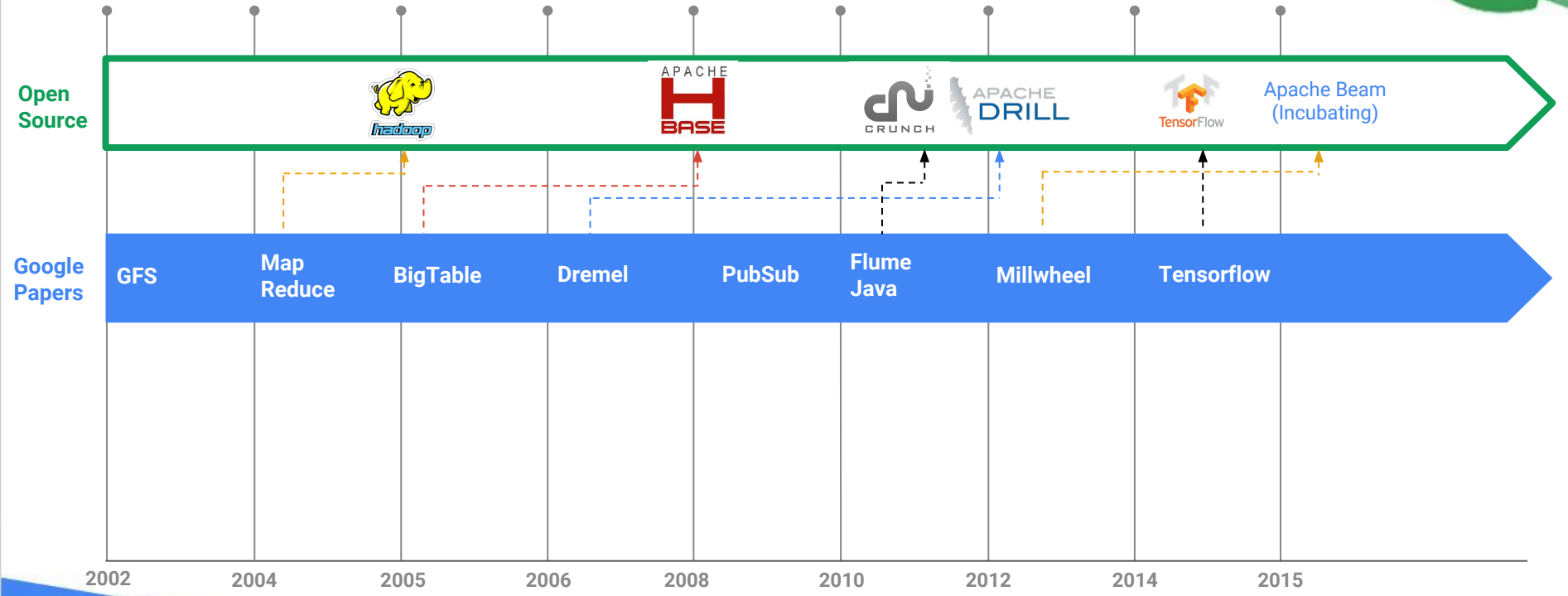
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were presented at The 36th International Conference on Very Large Data Bases, September 13-17, 2010, Singapore.
Proceedings of the VLDB Endowment, Vol. 3, No. 1
Copyright 2010 VLDB Endowment 2150-8097/10/09... $ 10.00.

[1] Dremel is a brand of power tools that primarily rely on their speed as opposed to torque. We use this name for an internal project only.

2006

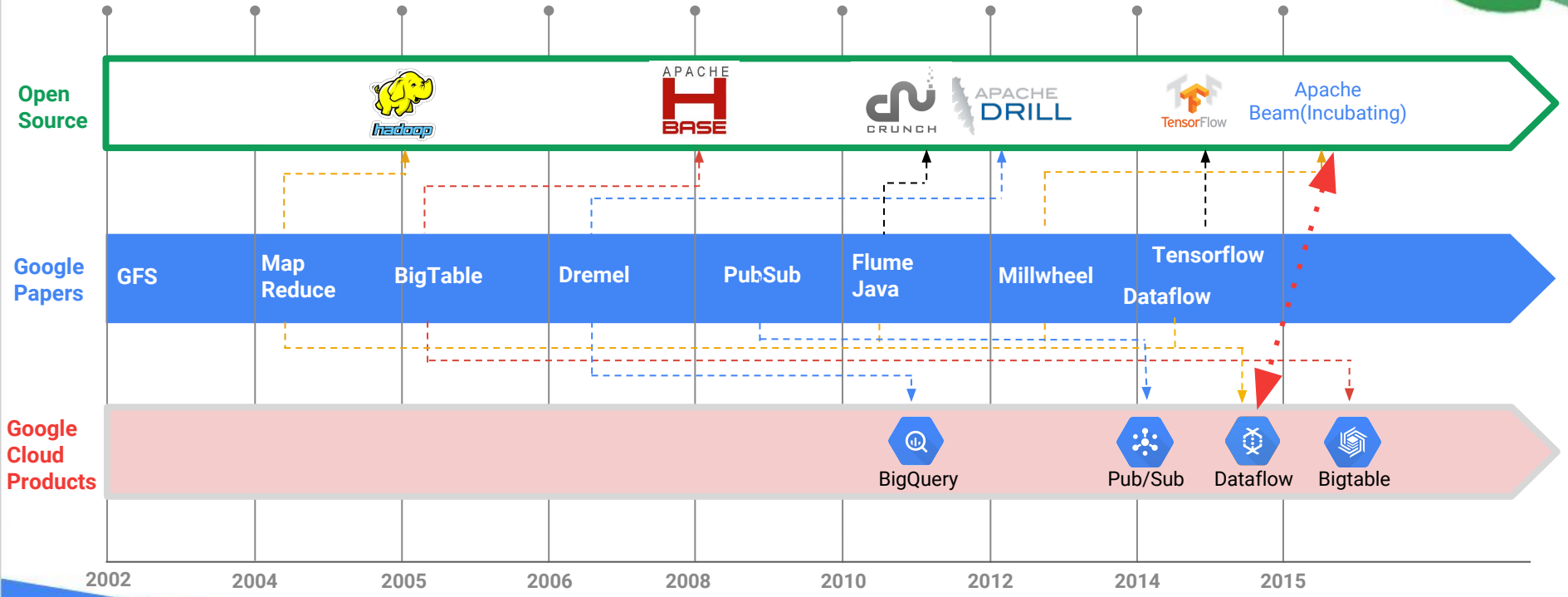
Google Cloud

10+ Years of Tackling Big Data Problems



GO JEK

10+ Years of Tackling Big Data Problems



Google Cloud's "Data" Platform

GO-JEK

Mobile apps

Web apps

Sensors and devices

Databases

- Cloud SQL
- Cloud Datastore
- Cloud Bigtable
- Spanner

Storage

- Cloud Storage
- Google BigQuery

Data Processing & Ingestion

- Cloud Dataflow
- Cloud Dataproc
- Cloud Pub/Sub

Analytics

- Google BigQuery
- Cloud Datalab
- Cloud Dataproc

Advanced Analytics & Intelligence

- Cloud Machine Learning
- Speech API
- Vision API
- Translate API

Business analysts

Data scientists

Developers

Connectors

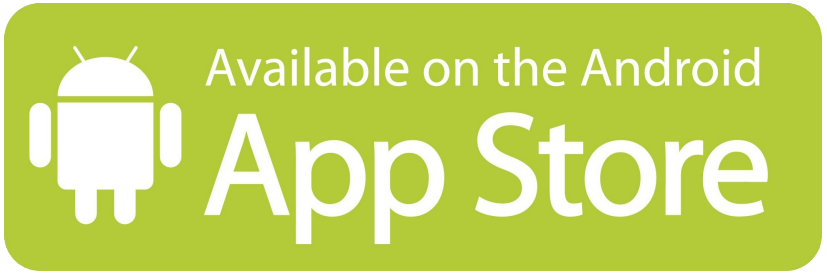
What is BigQuery?

- Fully managed, **no-ops** data warehouse
- Processes **terabytes** of data in **tens of seconds**
- The **economy** of cloud
- **Ease of access**: anyone in enterprise can run **homebrew SQLs**

GOJEK

Google Usage of BigQuery

How can a business analyst find Top 20 Apps in mater of seconds?



Google Usage of BigQuery

How can a business analyst find Top 20 Apps in mater of seconds?

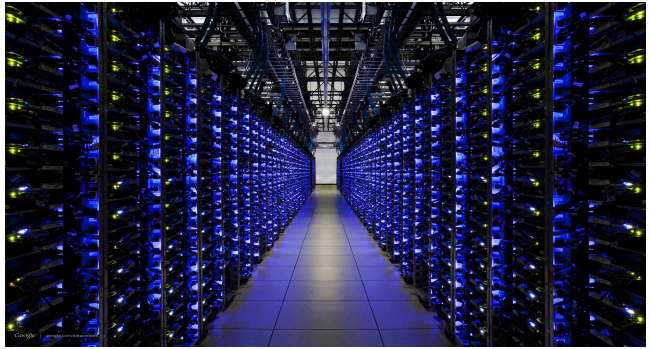

```
SELECT
  top(appId, 20) AS app,
  count(*) AS count
FROM installlog.2012;
ORDER BY
  count DESC
```

Result in ~20 seconds!

Google Usage of BigQuery

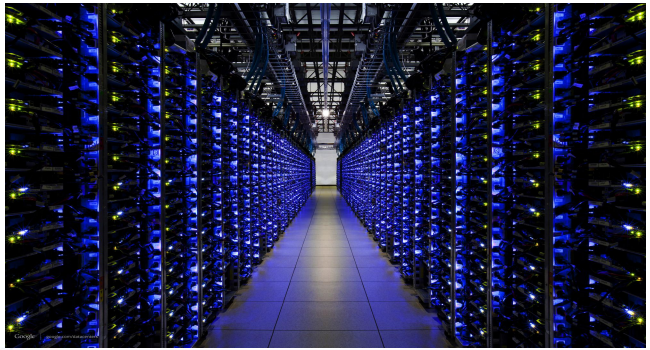
How do you find slow running servers from billions of log entries - in seconds?

Google | Data Centers



Google Usage of BigQuery

How do you find slow running servers from billions of log entries - in seconds?



```
SELECT
  count(*) AS count, source_machine AS
machine
FROM product.product_log.live
WHERE
  elapsed_time > 4000
GROUP BY
  source_machine
ORDER BY
  count DESC
```

Result in ~10 seconds!

Google Use of BigQuery

Find root cause why ad was or was not delivered in the last 30 days.

Query Stats

Result Size:	[44 rows, 2 columns]
Start Time:	Jan 23, 2015, 3:57:26 PM
End Time:	Jan 23, 2015, 3:57:31 PM
Num scanned rows:	1217870544
Total time:	4.812 seconds
Execution time:	2.7 seconds
CPU time:	4.158066666666667 minutes
IO time:	38.863033333333334 minutes
Num bytes read:	2.7394265672191978 GB
Num bytes transferred:	75.19854354858398 MB

View all stats

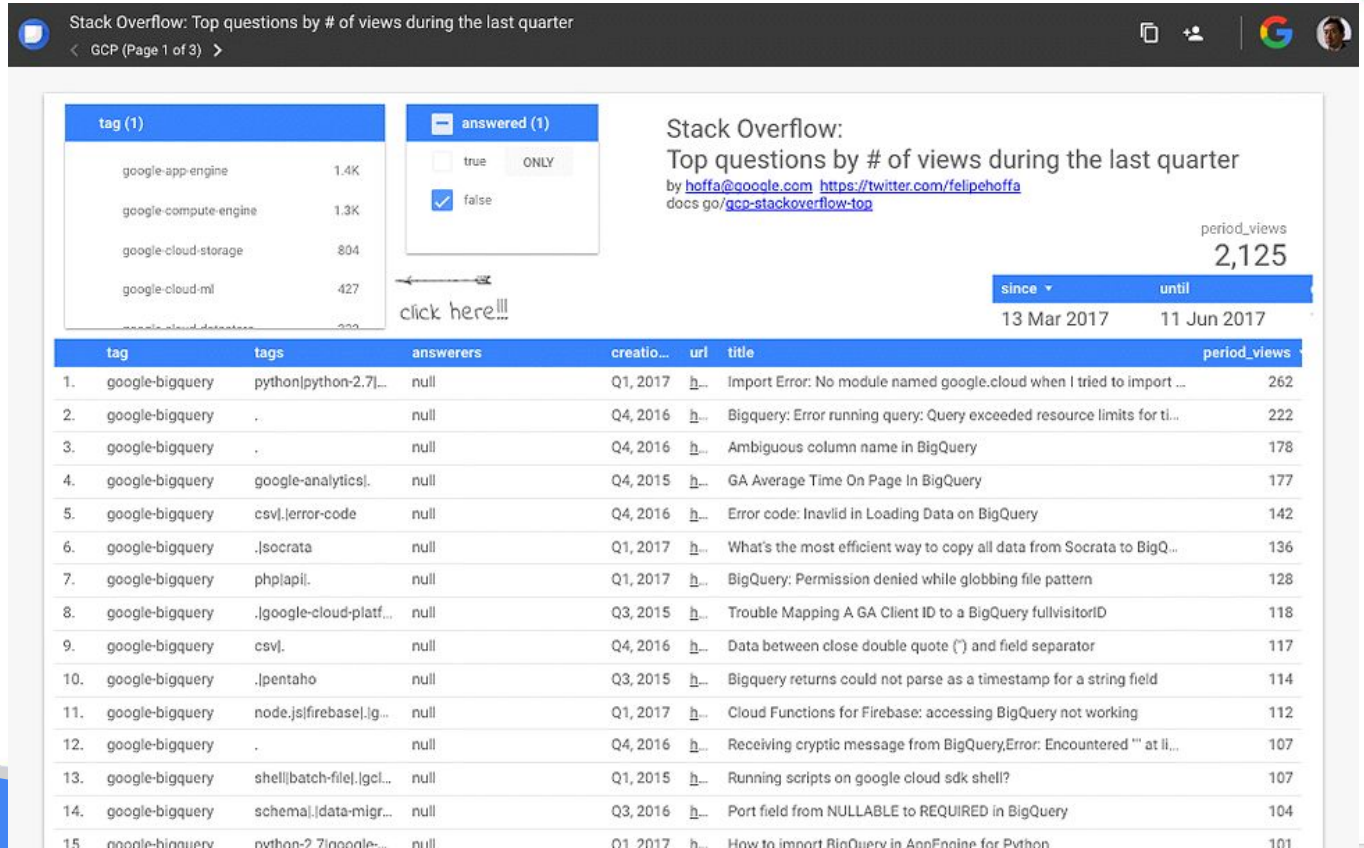
```
select date,
rejection_reason, count(*)
from
line_item_table.last30days
where line_item_id=56781234
```

doubleclick by Google

1.2B Rows scanned
Result in ~5 seconds!

Google Cloud

Who's asking what? StackOverflow Questions



	tag	tags	answerers	creatio...	url	title	period_views
1.	google-bigquery	python|python-2.7|...	null	Q1, 2017	h...	Import Error: No module named google.cloud when I tried to import ...	262
2.	google-bigquery	.	null	Q4, 2016	h...	Bigquery: Error running query: Query exceeded resource limits for ti...	222
3.	google-bigquery	.	null	Q4, 2016	h...	Ambiguous column name in BigQuery	178
4.	google-bigquery	google-analytics|.	null	Q4, 2015	h...	GA Average Time On Page In BigQuery	177
5.	google-bigquery	csv|.|error-code	null	Q4, 2016	h...	Error code: Inavlid in Loading Data on BigQuery	142
6.	google-bigquery	.|socrata	null	Q1, 2017	h...	What's the most efficient way to copy all data from Socrata to BigQ...	136
7.	google-bigquery	php|api|.	null	Q1, 2017	h...	BigQuery: Permission denied while globbing file pattern	128
8.	google-bigquery	.|google-cloud-platf...	null	Q3, 2015	h...	Trouble Mapping A GA Client ID to a BigQuery fullvisitorID	118
9.	google-bigquery	csv|.	null	Q4, 2016	h...	Data between close double quote (") and field separator	117
10.	google-bigquery	.|pentaho	null	Q3, 2015	h...	Bigquery returns could not parse as a timestamp for a string field	114
11.	google-bigquery	node.js|firebase|.|g...	null	Q1, 2017	h...	Cloud Functions for Firebase: accessing BigQuery not working	112
12.	google-bigquery	.	null	Q4, 2016	h...	Receiving cryptic message from BigQuery,Error: Encountered "' at li...	107
13.	google-bigquery	shell|batch-file|.|gcl...	null	Q1, 2015	h...	Running scripts on google cloud sdk shell?	107
14.	google-bigquery	schema|.|data-migr...	null	Q3, 2016	h...	Port field from NULLABLE to REQUIRED in BigQuery	104
15.	google-bigquery	python-2.7|google-...	null	Q1, 2017	h...	How to import BigQuery in AppEngine for Python	101

Google Cloud

GOJEK

Powerful Backend?
Check.
What's next?

Google Cloud

Current Data Model

- Star Schema

Customer Dimension		
id	*nama*	*nomor_telepon*
123	Jo	628112345678

Merchant Dimension		
id	*nama*	*kategori_merchant*
1	Warung Bu Iis	TRADISIONAL

Driver Dimension		
id	*nama*	*jenis_kelamin*
456	Asep	M
457	Doni	M
458	Siti	F

Order Fact			
id	*id_customer*	*id_driver*	id_merchant
10001	123	458	1

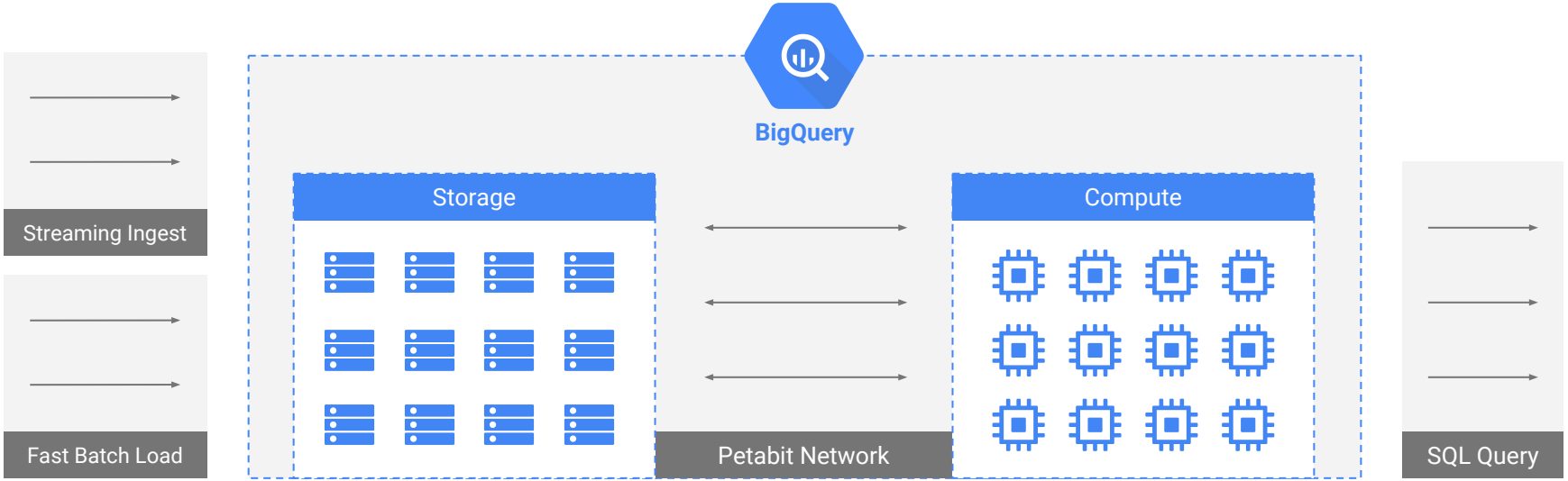
Item Fact			
id	*id_order*	*nama_item*	*harga*
101	10001	Nasi Goreng	30000
102	10001	Es Teh Manis	5000

Driver Search Fact			
id	*id_driver*	*nama*	*status*
1	456	Asep	Rejected
2	457	Doni	Rejected
3	458	Siti	Accepted

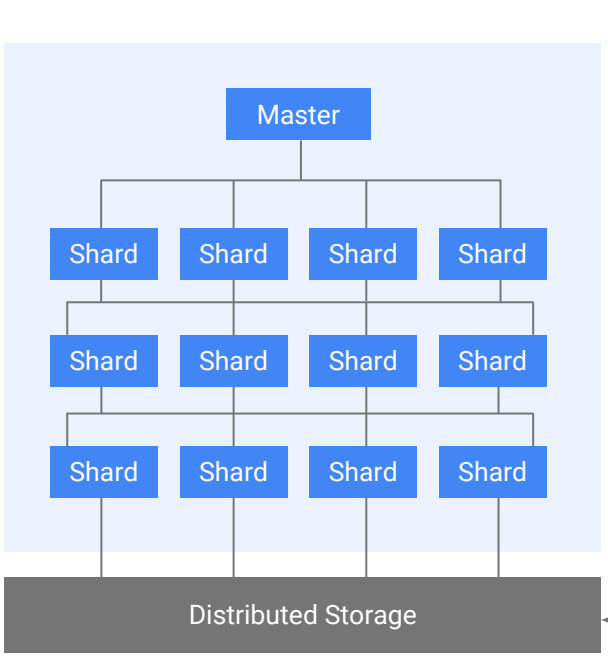
- *Advantages*
  - Clean and structured model.
- *Disadvantages*
  - Difficult to do data discovery.
  - Need a lot of join, resulting high computational resource; especially in MPP environment.

BigQuery = **Massively Parallel Processing** query

with the petabit network and thousands of servers



BigQuery Compute: Dremel



Dynamic Serving Tree

Columnar Storage

Dynamically organized tree, sized for each query

Adapt to queries on normalized and denormalized schemas

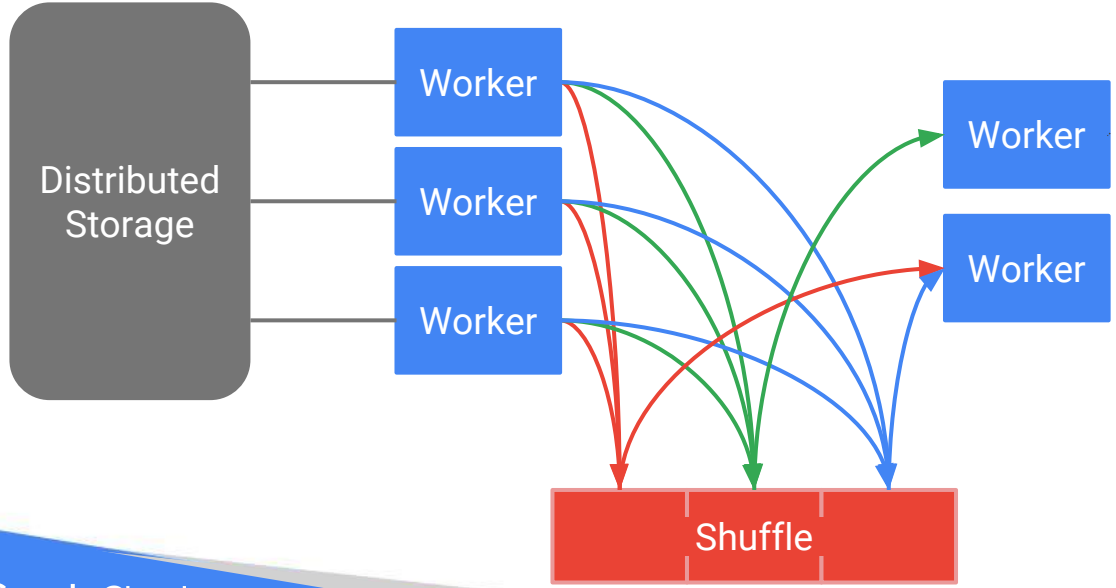
Multi-stage queries several times faster

BigQuery Remote Memory Shuffle

SELECT state

WHERE year...
SHUFFLE BY
state

GROUP BY state
COUNT(*)



Shuffle does not block future stages

BigQuery uses dynamic partitioning to distribute shuffle optimally

Substantial key-skew can still impact performance

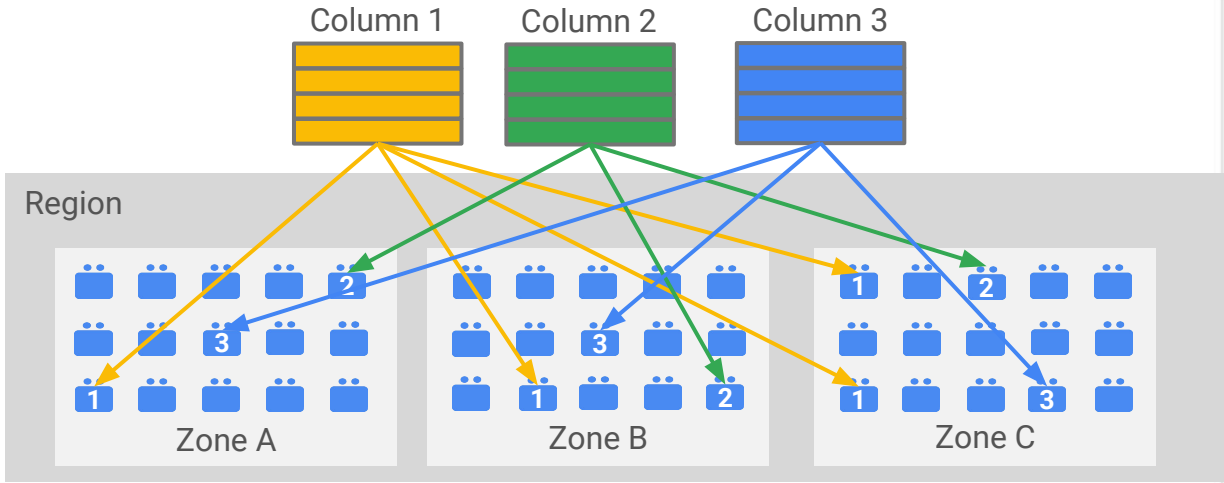
BigQuery Managed Storage

Each column is stored in its own file

Each file is compressed and encrypted on disk

Storage is durable

Each file is replicated across datacenters



Demo

Full scan on 10B rows
with BigQuery

Some BigQuery Stats

10.5 Trillion	Largest query (rows)
2.1 petabytes	Largest query (data size)
62 petabytes	Largest storage customer
4.5 million rows/sec	Peak ingestion rate

GOJEK

So, denormalized?

Google Cloud

Remodeling The Data: Part One

- Denormalized

Denormalized Dataset												
id_order	*id_customer*	*nama_customer*	*nomor_telepon*	*id_merchant*	*nama_merchant*	*id_item*	*nama_item*	*harga_item*	*id_bid*	*id_driver*	*nama_driver*	*jenis_kelamin_pengendara*
10001	123	Jo	628112345678	1	Warung Bu Iis	101	Nasi Goreng	30000	1	456	Asep	M
10001	123	Jo	628112345678	1	Warung Bu Iis	102	Es Teh Manis	5000	1	456	Asep	M
10001	123	Jo	628112345678	1	Warung Bu Iis	101	Nasi Goreng	30000	2	457	Doni	M
10001	123	Jo	628112345678	1	Warung Bu Iis	102	Es Teh Manis	5000	2	457	Doni	M
10001	123	Jo	628112345678	1	Warung Bu Iis	101	Nasi Goreng	30000	3	458	Siti	F
10001	123	Jo	628112345678	1	Warung Bu Iis	102	Es Teh Manis	5000	3	458	Siti	F

- *Advantages*
  - Everything it's in one place.
- *Disadvantages*
  - A lot of duplicates data; need to understand the model to build the correct logic.

What about conceptually nested data?

- Should I normalize or denormalize sessions, orders, etc?
- **Should I normalize for space efficiency?**
  Don't bother. Space is cheap
- **So denormalize to avoid joins?**
  Don't bother. There's a better option

GOJEK

Nested Denormalized

Google Cloud

Remodeling The Data: Part Two

- Nested Denormalized

Nested Denormalized Dataset													
id_order	*id_customer*	*nama_customer*	*nomor_telepon*	*id_merchant*	*nama_merchant*	*id_item*	*nama_item*	*harga_item*	*id_bid*	*id_driver*	*nama_driver*	*jenis_kelamin_pengendara*	*status_pencarian_pengendara*
10001	123	Jo	628112345678	1	Warung Bu Iis	101	Nasi Goreng	30000	1	456	Asep	M	Rejected
						102	Es Teh Manis	5000	2	457	Doni	M	Rejected
									3	458	Siti	F	Accepted

- *Advantages*
  - Everything is correctly modeled based on the data flow.
- *Disadvantages*
  - Shift in thinking is needed to build the correct logic.

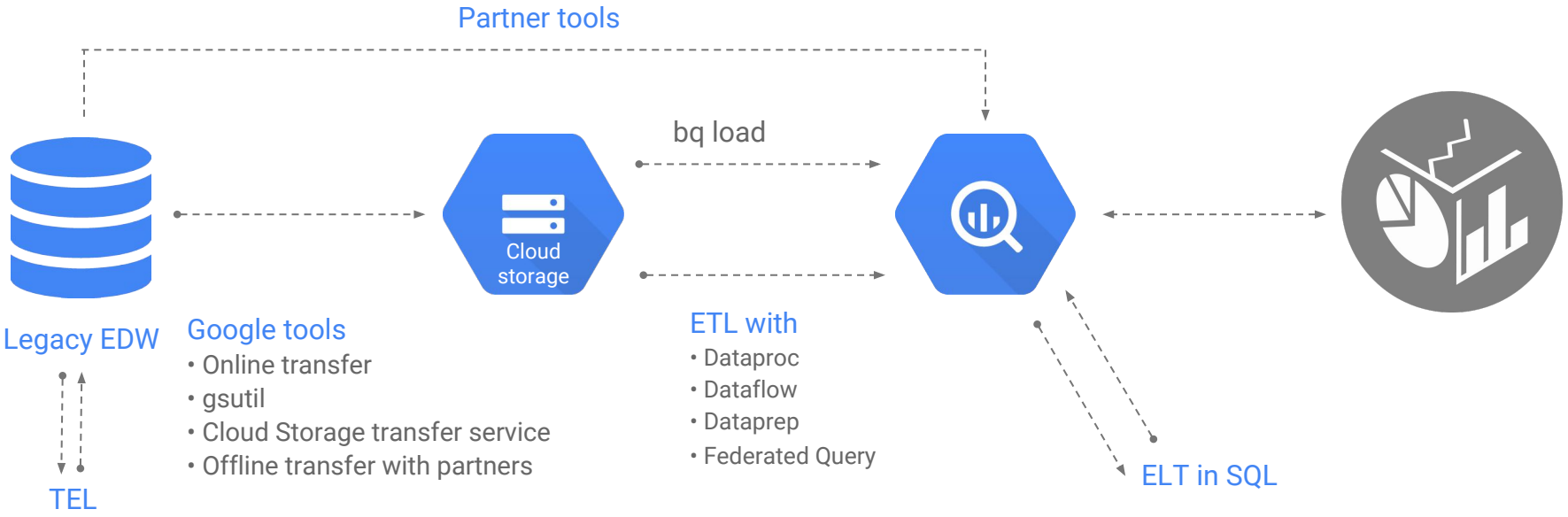
GOJEK

Remodel the Data?
Check.
What's next?

Google Cloud

Migrating to your NEXT EDW

To ETL or to ELT, that is the question



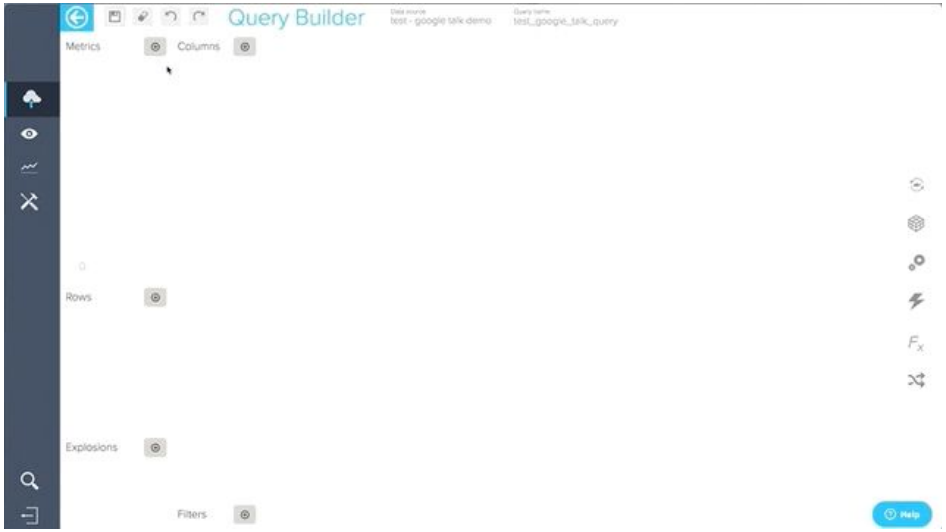
Demo

You have your data

Let's get it into BigQuery

Tools to The Rescue!

- ***What we need?***
  - Versatile data discovery tools.
  - Integrate nicely with BigQuery as well as other system.
  - Easy to use.
- ***Use cases***
  - Data discovery process.
  - Dashboard / Reporting purpose.



GOJEK

How all of this can help the business?

Google Cloud

Be hypothesis-driven

“Always start with a hypothesis, then do analysis to prove/disprove it — minimize exploratory analyses — unless you’re starting from scratch and need to generate a hypothesis” (Raditya Wibowo, VP of Allocation)

Example

“During Ramadan season, especially before Lebaran and before vacation end, GO-FOOD orders from merchants who sell “Oleh-Oleh” will increase in some areas.”

Data Discovery Framework

- Show the order data based on date.
- Filter based on store category.
- See the trend of each area.
- Pick up some sample and see the details.

Dataset Requirement

- Dimension Dataset

Customer Dimension	
id	*nama*

Driver Dimension	
id	*nama*

Merchant Dimension		
id	*nama*	*kategori_merchant*

Area Dimension	
id	*nama*

- Fact Dataset

Item Fact			
id	*id_order*	*nama_item*	*harga*

Order Fact			
id	*id_customer*	*id_driver*	id_merchant

- Denormalized Dataset

Nested Denormalized Dataset							
id_order	nama_customer	nama_driver	nama_merchant	nama_area	kategori_merchant.[]	nama_item.[]	harga_item.[]

GOJEK

Let’s start!

Google Cloud

Data Discovery Process

- Show the data.
- Who sells “Oleh-Oleh”?
- What about the trend for each area?
- Pick up some sample and see the details!

```
SELECT
  nama_area,
  nama_merchant,
  COUNT(1) AS total
FROM

[gojek_google_demo.dummy_gojek_google_dataset]
WHERE
  kategori_merchant IN ('OLEH_OLEH')
  AND nama_area IN ('MEDAN')
  AND DATE(waktu_booking) = DATE('2017-06-23')
GROUP BY
  1,
  2 ;
```

GO JEK

Self Service!

Google Cloud

GO JEK

The Takeaway

Google Cloud

The Takeaway

- **BigQuery:** petabyte scale data warehouse in the cloud.
- **Dremel** has been supporting core business decisions and insights within Google.
- BigQuery is for everyone. **Ease of use**.
- Fully managed, **No-Ops**.
- Start with BigQuery and expand when needed!

The Takeaway

- **Business-needs analysis** is always a best place to start to build or change the way data flow works; and it should be the end as well.
- We should be able to **correlate between the actual process flow and the data model** itself: storing the data is important, so did the representation of it.
- **Effectively designing the data architecture** in certain way will strike a good balance between operation maintenance, performance and accessibility.

GOJEK

Thank You

Google Cloud