Google Big Query and ElasticSearch

Relational Databases at Scale?

- How do traditional relational databases handle record growth at scale?
- Traditionally, very large tables are hard to scan & compute

Organization Details

Company ID	Company Name
161218560	NY Association Inc.
...	...
...	...
...	...
10 Billion Row Table	

Relational Databases at Scale?

- Indexes (pre-sorted) help with common queries:

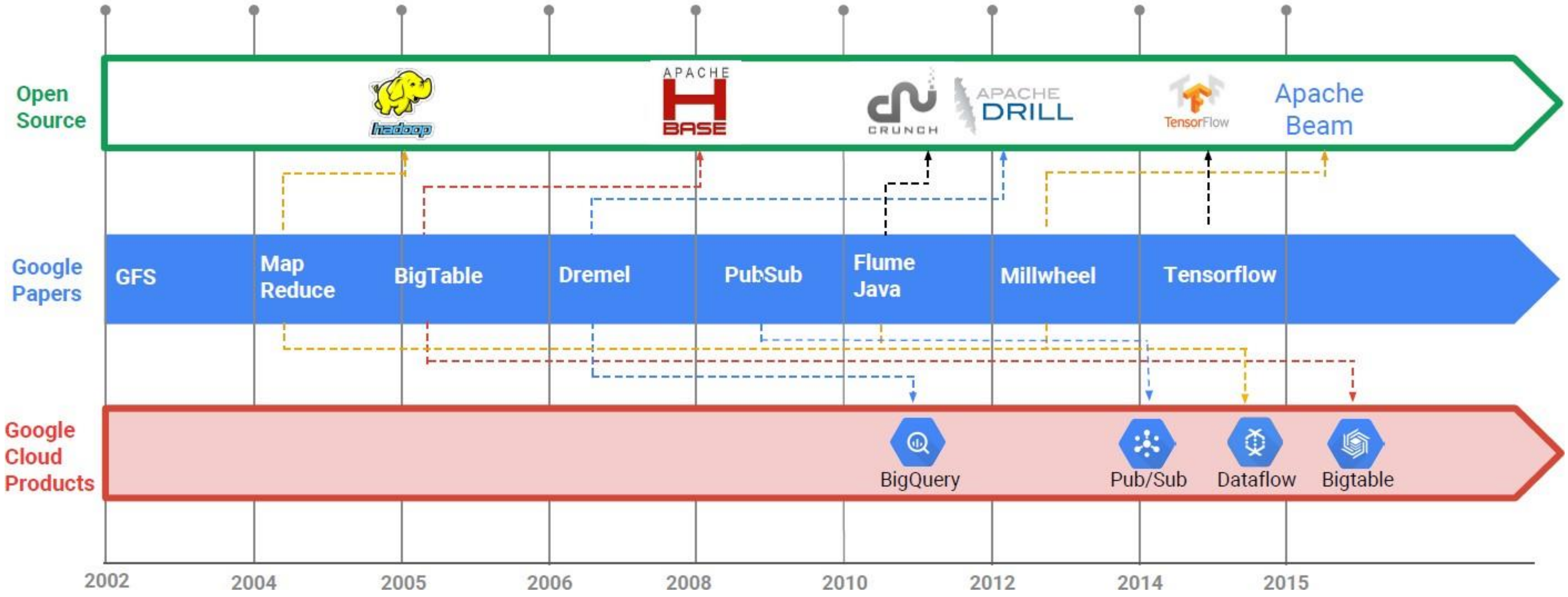
Organization Details

Company ID	Company Name
161218560	NY Association Inc.
...	...
...	...
...	...
10 Billion Row Table	

Index

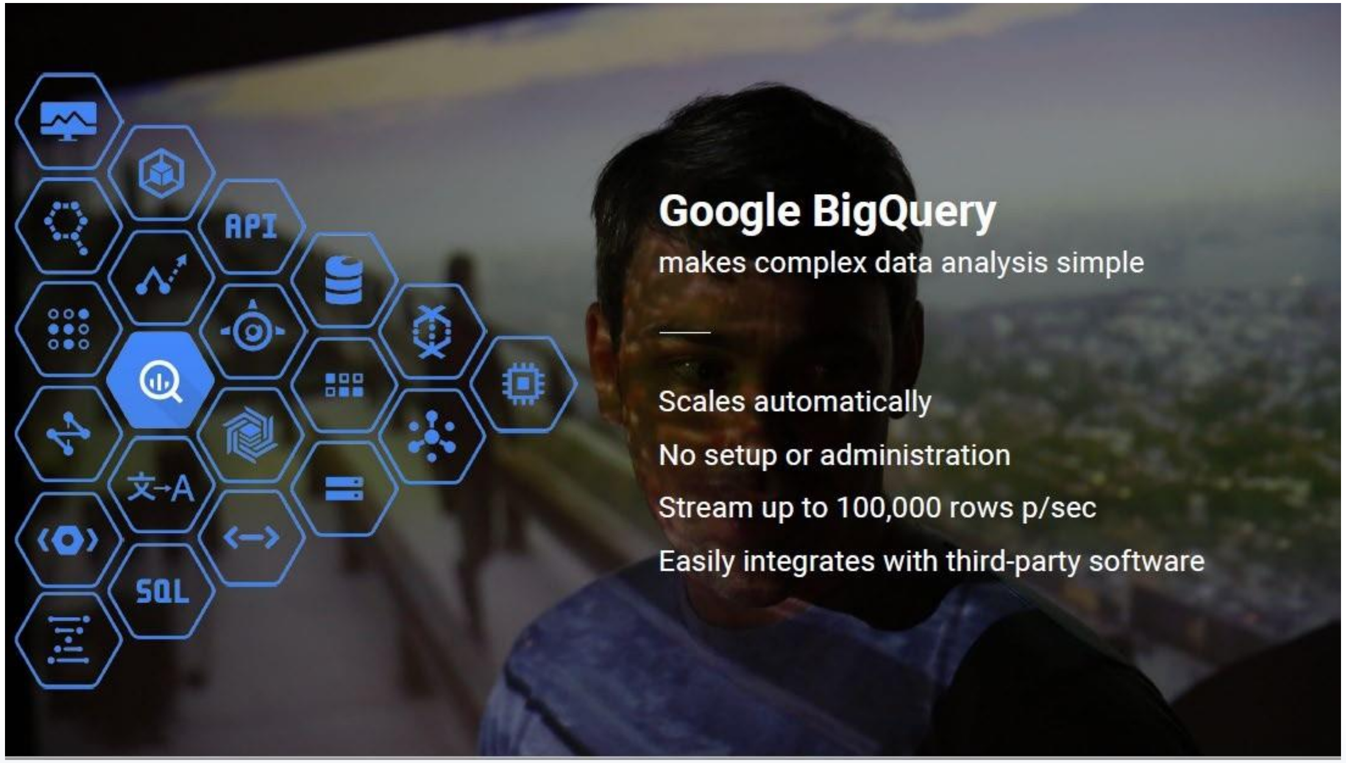
Company Name	Ranked Order
ACME Inc.	1
...	...
...	
NY Association Inc.	900,000
...	

10+ Years of Tackling Big Data Problems

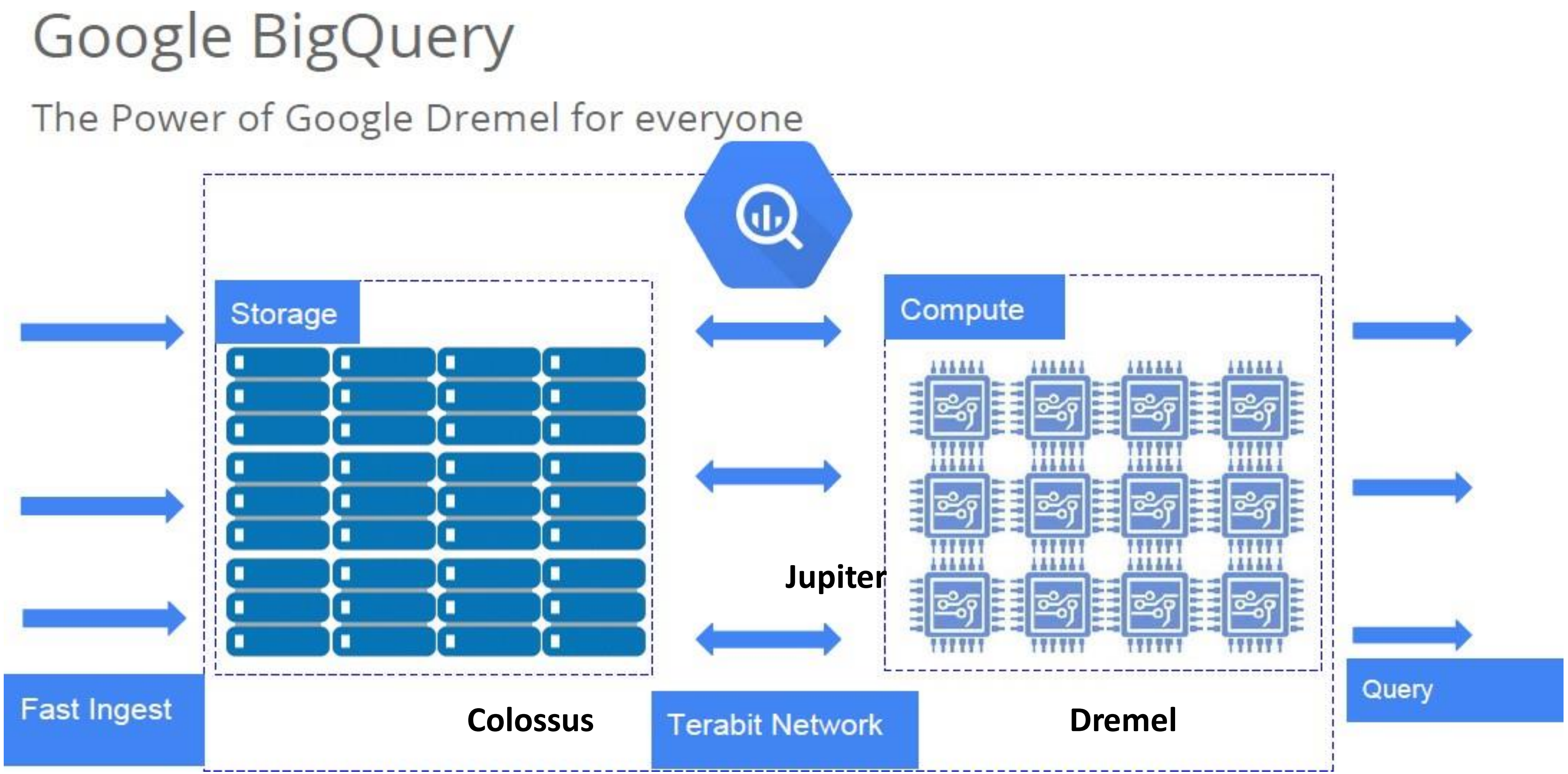


What is Google BigQuery?

- BigQuery is a ***hybrid*** system that allows you to store data in columns
- A serverless, highly scalable, and cost-effective multi-cloud data warehouse designed for business agility
- A "hybrid SQL-NoSQL" database:
  - NoSQL (JSON like) storage (under the hood)
  - SQL query interface
- Extended to support machine learning
- Ideal for big data application



A Distributed System for Querying Very Large Datasets



Dremel **turns SQL queries into execution trees**

Google BigQuery Performance

Running an *inefficient* regular expression over 100 billion rows in

less than 60 seconds

Big Data at Scale!!!

https://cloud.google.com/blog/big-data/2016/01/anatomy-of-a-bigquery-query

Why is BigQuery so fast?

BigQuery is designed to query structured and semi-structured data **using standard SQL**.
It is highly optimized for query performance and provides extremely high cost effectiveness.
BigQuery is a cloud-based fully-managed service

Columnar Storage:
BigQuery stores data in a proprietary columnar format called Capacitor
Data is stored in a columnar storage fashion which makes possible to achieve a very high compression ratio and scan throughput.
Tree Architecture is used for dispatching queries and aggregating results across thousands of machines in a few seconds.

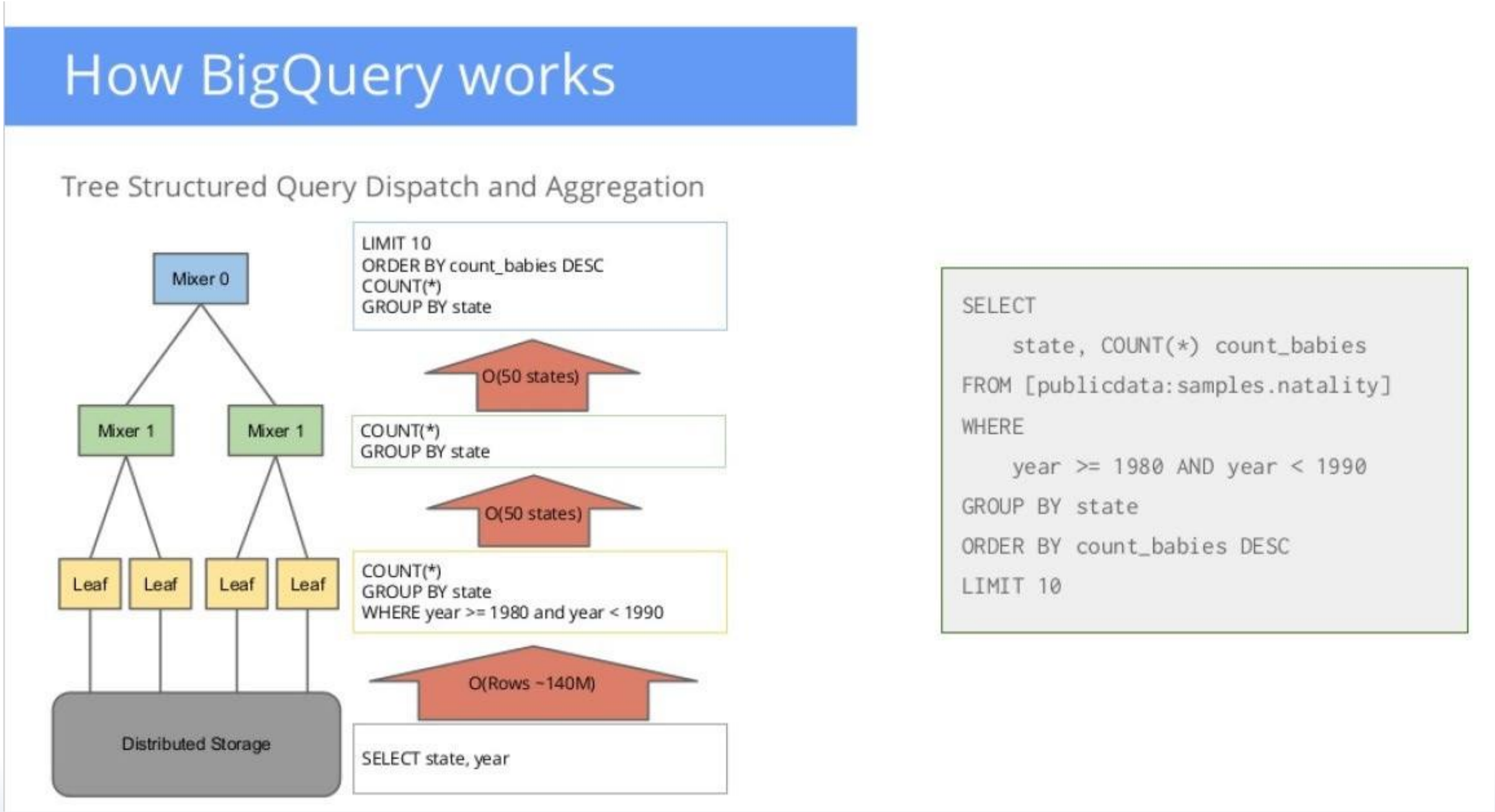
Google BigQuery vs Relational Databases

	Relational DBMS	BigQuery
SQL queries	Yes	Yes
Data	Structured	Structured + Nested Data
Index-based?	Index	Non-Index
Storage	Row based	Column based
ACID Transactions?	ACID	No

https://db-engines.com/en/system/Google+BigQuery%3BMySQL

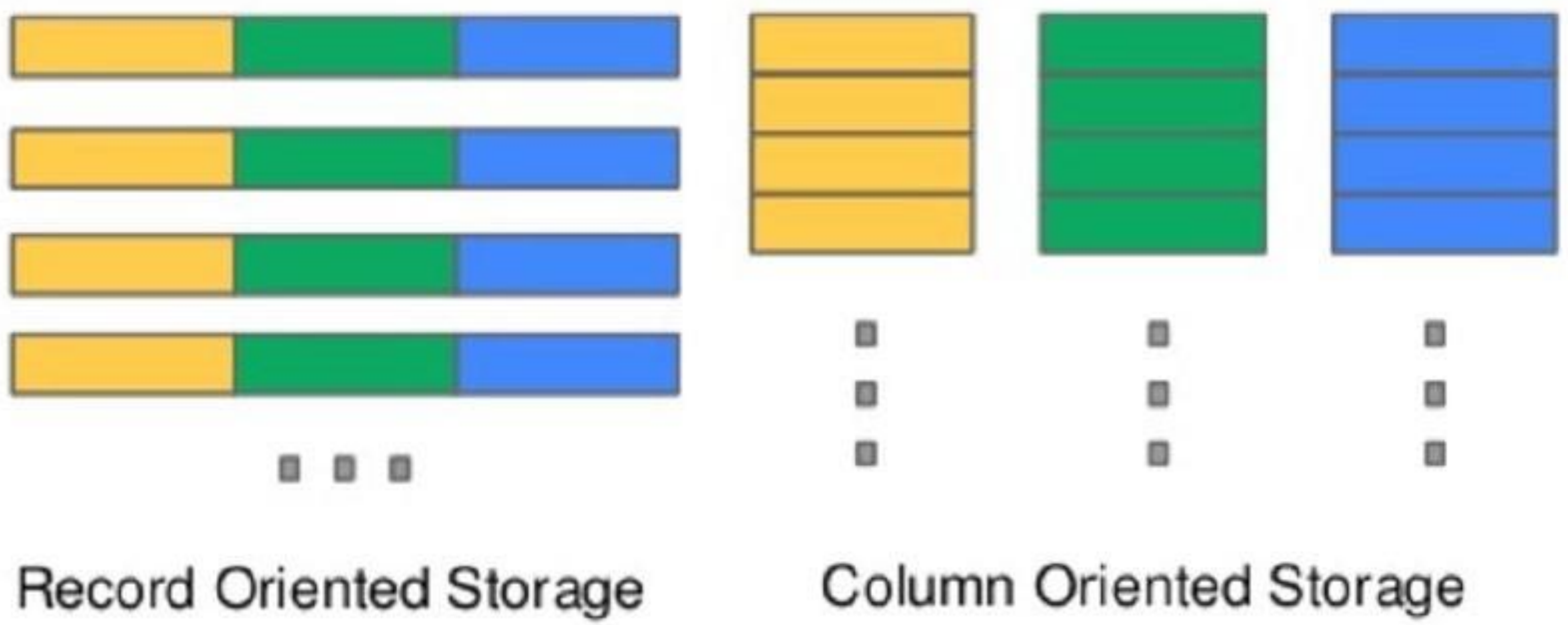
Google BigQuery

- Google BigQuery (GBQ) introduces several key innovations for scalability
- **Column-based** data storage
- **Break Apart Tables** into pieces
- **Nested Fields** within a table



An indepth look at Google BigQuery Architecture by Felipe Hoffa of Google

BigQuery Column-Oriented Storage



- Storing related values (faster to loop through at execution time)
- Columns can be individually compressed
- Access values from a few columns without reading every one

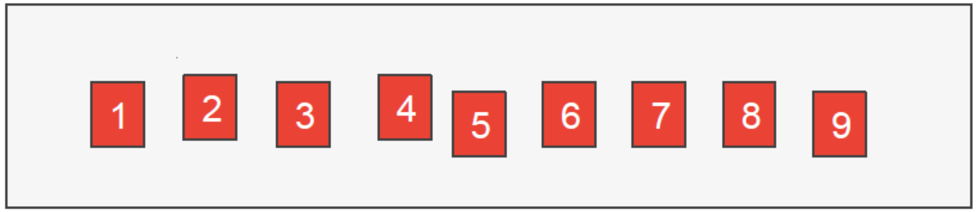
BigQuery Data Sharding

- BigQuery automatically breaks apart data into smaller shards

Organization Details

Company ID	Company Name
161218560	NY Association Inc.
...	...
...	...
...	...
10 Billion Row Table	

Google File System



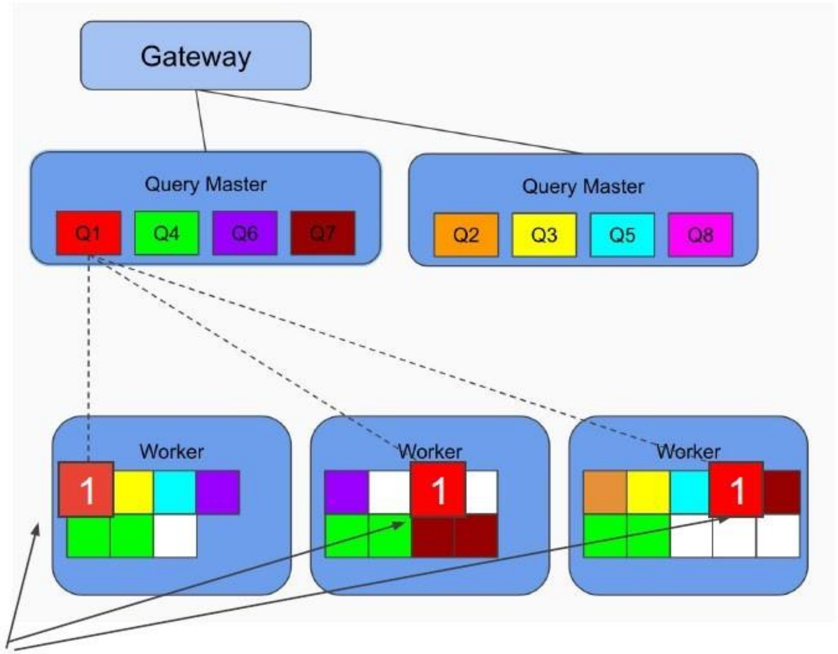
BigQuery Data Sharding

- BigQuery automatically pieces it back together for queries

Organization Details

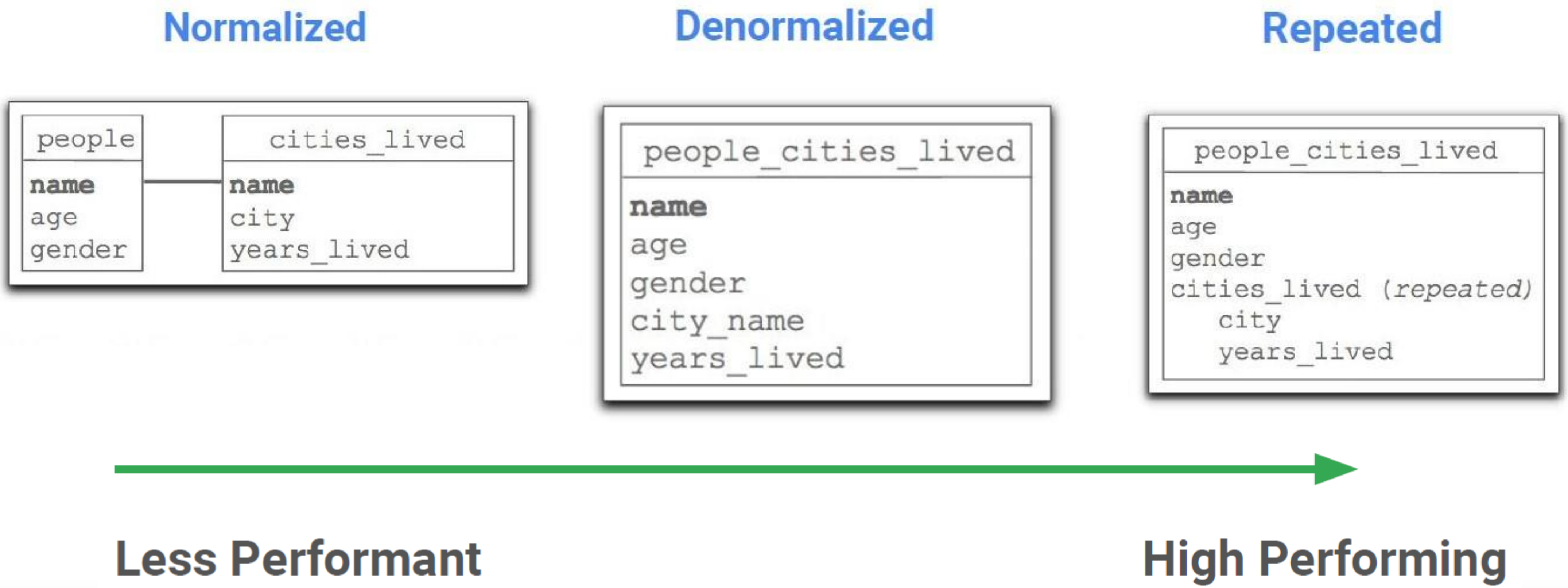
Company ID	Company Name
161218560	NY Association Inc.
...	...
...	...
...	...
10 Billion Row Table	

`SELECT Company Name ORDER BY Company Name`



Shards of data are read and processed in **parallel**

BigQuery: Normalization vs Denormalization vs Nested Data



BigQuery Nested & Repeated Fields

- BigQuery can use **nested schemas** for highly scalable queries

Organization Details with Nested Historical Transactions

NESTED

Company ID	Company Name	Transactions.Amount	Code.Expense
161218560	NY Association Inc.	$10.000	Lobbying
		$5,000	Legal
		$1,000	Insurance
123435560	ACME Co.	$7,000	Travel

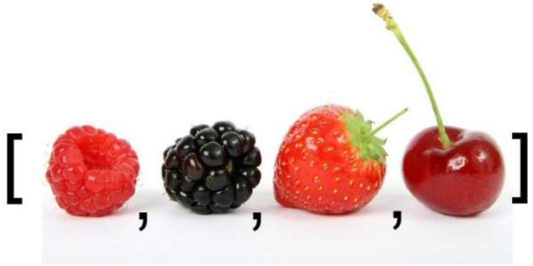
- Avoid costly joins
- No performance slowdown for `SELECT(DISTINCT Company ID)`

Arrays and Structs in BigQuery

- Nested and repeated data are extensions to Standard SQL (SQL 2011 standard) so their supports vary among vendors.
- https://cloud.google.com/bigquery/docs/reference/standard-sql/arrays
- ARRAYs and STRUCTs are powerful concepts
- Nested & repeated records are **ARRAYs of STRUCTs**
- Tables with repeated fields are conceptually like pre-joined tables

Arrays and Structs in BigQuery

- **ARRAYs** are ordered lists of zero or more data values that have the same data type:



- **STRUCTs** are flexible containers of ordered fields each with a type (required) and field name (optional)
  - Can store multiple data types (including ARRAYs) in a STRUCT

[BigQuery Nested and Repeated Fields: Dig Deeper into Data (Cloud Next '18)]()

- Nested records in BigQuery are Arrays of Structs.
- Instead of Joining with a sql_on: expression, **the join relationship is built into the table**.
- UNNESTing a ARRAY of STRUCTs is similar to joining a table.

https://cloud.google.com/bigquery/docs/reference/standard-sql/arrays

https://cloud.google.com/bigquery/docs/reference/standard-sql/dml-syntax

Array Example

build an array literal in BigQuery using brackets ([and]). Each element in an array is separated by a comma. SELECT [1, 2, 3] as numbers; SELECT ["apple", "pear", "orange"] as fruit;

```
SELECT
['raspberry', 'blackberry', 'strawberry', 'cherry']
AS fruit_array
```

Row	fruit_array
1	raspberry
	blackberry
	strawberry
	cherry

```
WITH fruits AS (
SELECT ['raspberry', 'blackberry', 'strawberry', 'cherry']
AS fruit_array
)

SELECT ARRAY_LENGTH(fruit_array) AS array_size
FROM fruits;
```

Row	array_size
1	4

https://cloud.google.com/bigquery/docs/reference/standard-sql/arrays

STRUCT Example

```
SELECT
STRUCT(35 AS age, 'Jacob' AS name)
```

Row	f0_.age	f0_.name
1	35	Jacob

```
SELECT
STRUCT(35 AS age, 'Jacob' AS name) AS customers
```

Row	customers.age	customers.name
1	35	Jacob

STRUCT & ARRAY Example

https://cloud.google.com/bigquery/docs/nested-repeated

- STRUCTs Can Even Contain ARRAY Values

```
SELECT
STRUCT(35 AS age, 'Jacob' AS name,['apple', 'pear', 'peach'] AS
items) AS customers
```

Row	customers.age	customers.name	customers.items
1	35	Jacob	apple
			pear
			peach

STRUCT & ARRAY Example

- ARRAYS can Contain STRUCTs as Values

```
SELECT
[
STRUCT(35 AS age, 'Jacob' AS name, ['apple', 'pear', 'peach'] AS items),
STRUCT(33 AS age, 'Miranda' AS name, ['water', 'pineapple', 'ice cream'] AS items)
] AS customers
```

Row	customers.age	customers.name	customers.items
1	35	Jacob	apple
			pear
			peach
	33	Miranda	water
			pineapple
			ice cream

Example: Data Warehouse Schema

Original Data

OrderId	CustomerId	CustomerName	timestamp	Location	purchasedItems			
					sku	description	quantity	price
1000001	65401	John Doe	12/18/2017 15:0	Faraway	ABC123456	Redwood 8x4	3	36.3
					TBL535522	Sapient Table	1	878.4
					CHR762222	Cherrywood Ch	6	435.6
					sku	description	quantity	price
1000002	74682	Jane Michaels	12/16/2017 11:3	Nearland	GCH635354	Garden chairs	4	345.7
					GRD828822	Ceramic Pots	2	9.5
					sku	description	quantity	price
1000003	63636	Jose Carlos	12/16/2017 13:4	Nearland				

Transaction Fact

Order Id	timestamp	CustomerId	sku	quantity	price
1000001	12/18/2017 15:02:00	65401	ABC123456	3	36.3
1000001	12/18/2017 15:02:00	65401	TBL535522	1	878.4
1000001	12/18/2017 15:02:00	65401	CHR762222	6	435.6
1000002	12/16/2017 11:34:00	74682	GCH635354	4	345.7
1000002	12/16/2017 11:34:00	74682	GRD828822	2	9.5

Customer Dimension

CustomerId	CustomerName	Location
65401	John Doe	Faraway
74682	Jane Michaels	Nearland
63636	Jose Carlos	Nearland

Product Dimension

sku	description
ABC123456	Redwood 8x4
TBL535522	Sapient Table
CHR762222	Cherrywood Chair
GCH635354	Garden chairs
GRD828822	Ceramic Pots

https://cloud.google.com/blog/topics/developers-practitioners/bigquery-explained-working-joins-nested-repeated-data

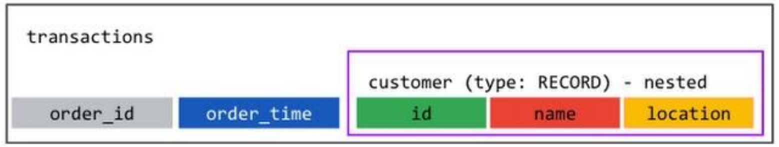
Example: Data Warehouse in BigQuery (Denormalizing data with nested and repeated structures)

Transaction Fact					
Order Id	**timestamp**	**CustomerId**	**sku**	**quantity**	**price**
1000001	12/18/2017 15:02:00	65401	ABC123456	3	36.3
1000001	12/18/2017 15:02:00	65401	TBL535522	1	878.4
1000001	12/18/2017 15:02:00	65401	CHR762222	6	435.6
1000002	12/16/2017 11:34:00	74682	GCH635354	4	345.7
1000002	12/16/2017 11:34:00	74682	GRD828822	2	9.5

Customer Dimension		
CustomerId	**CustomerName**	**Location**
65401	John Doe	Faraway
74682	Jane Michaels	Nearland
63636	Jose Carlos	Nearland

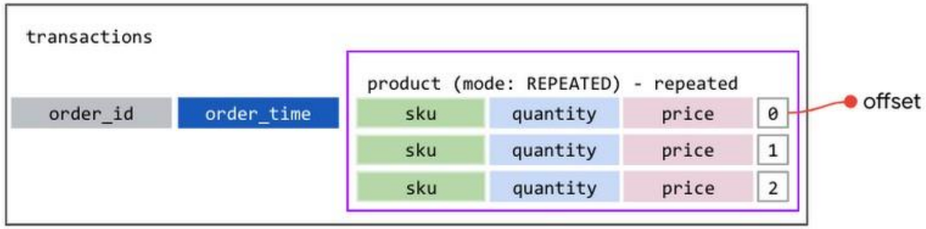
Product Dimension	
sku	**description**
ABC123456	Redwood 8x4
TBL535522	Sapient Table
CHR762222	Cherrywood Chair
GCH635354	Garden chairs
GRD828822	Ceramic Pots

Nested Fields



- **STRUCT/RECORD** data type contains ordered fields with a type and name.
- Use dot notation to query a nested column.
  E.g. **customer.name** refers to **name** field in **customer** column.

Repeated Fields



- **ARRAY** data type is an ordered list of zero or more elements of the same data type.
  For e.g. **product** is an **ARRAY** of **STRUCT** here.
- Use UNNEST() to flatten the repeated data and **OFFSET/ORDINAL** to access individual element

Nested Repeated Fields

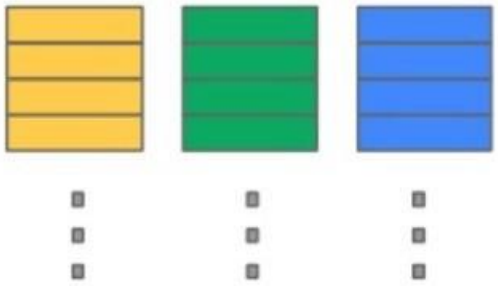
- Combining nested and repeated fields denormalizes a 1:many relationship without joins.
- Use dot notation to query a nested column and UNNEST() to flatten the repeated data.

https://cloud.google.com/blog/topics/developers-practitioners/bigquery-explained-working-joins-nested-repeated-data

Summary

• BigQuery architecture is designed for petabyte-scale querying performance

Tables are broken into pieces, called shards, to allow for scalability



BigQuery uses compressed column-based storage for fast retrieval

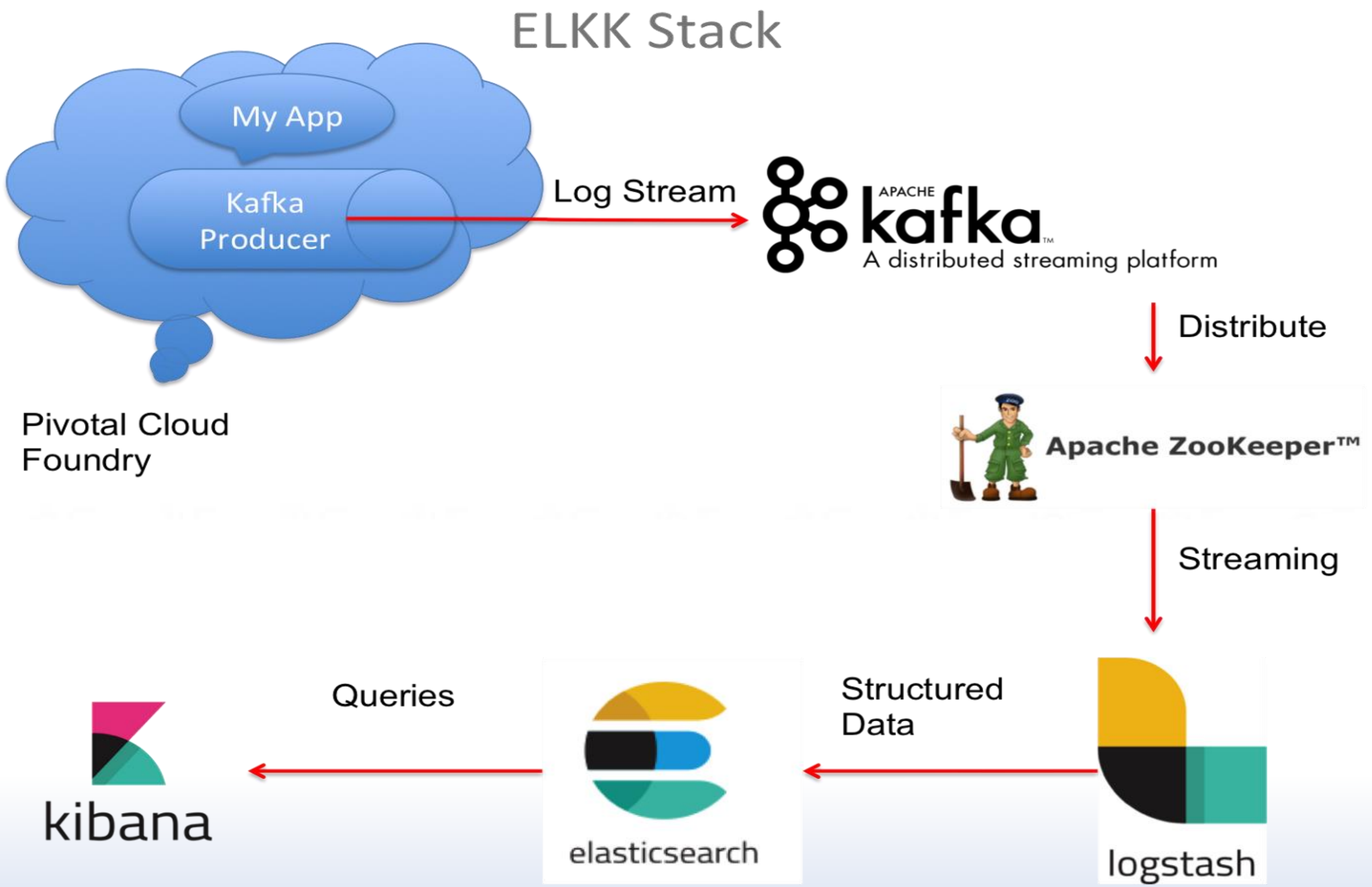
Structs and arrays are data type containers that are foundational to repeated fields

Row	date	top_articles.title
1	2010-08-23	Why GNU grep is Fast
		Readme Driven Development
2	2010-04-26	Police raid Gizmodo editor's house
		Not even in South Park?
3	2009-09-15	Learning Advanced JavaScript
		Sub-pixel re-workings of YouTube and BBC favicons

Tables with repeated fields are conceptually like pre-joined tables

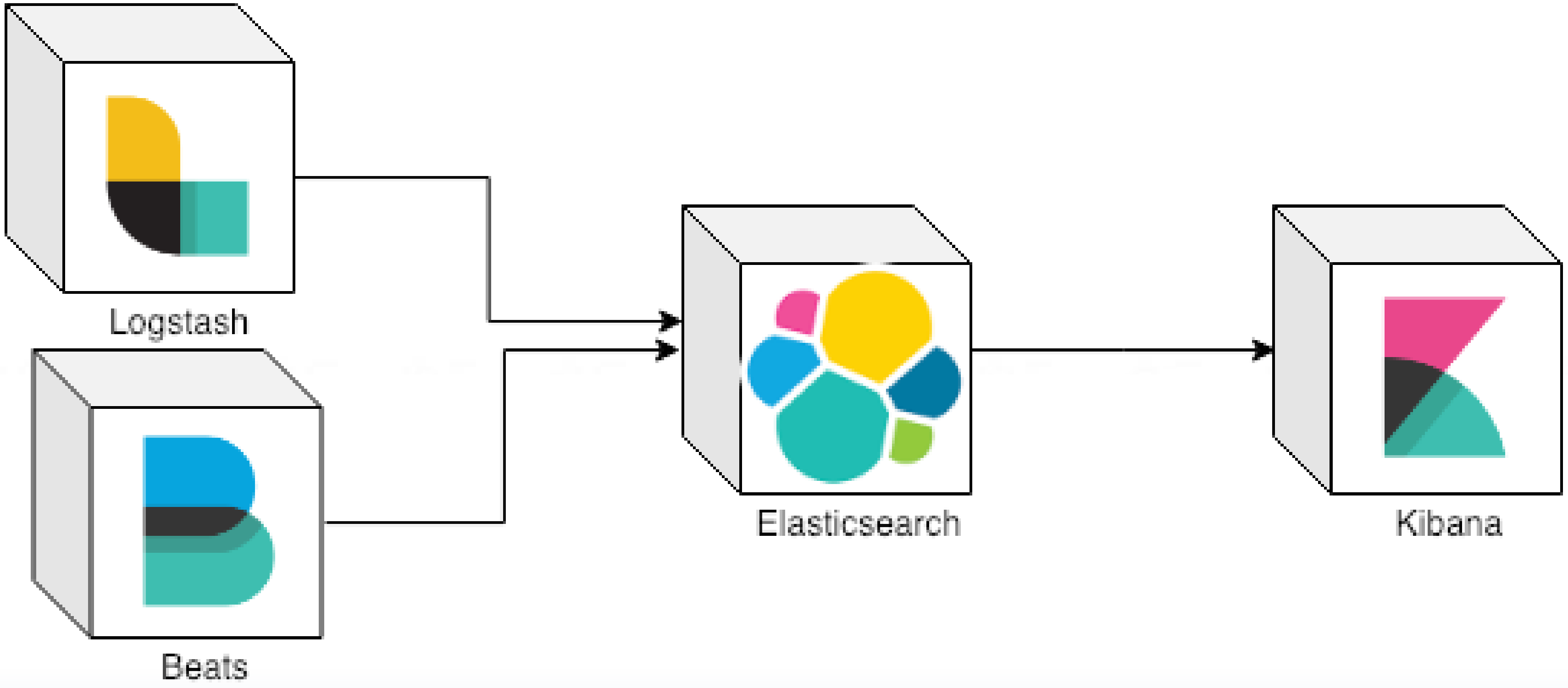
Introduction to ElasticSearch

Real-time Data Monitoring using Kafka, Logstash, Elasticsearch and Kibana

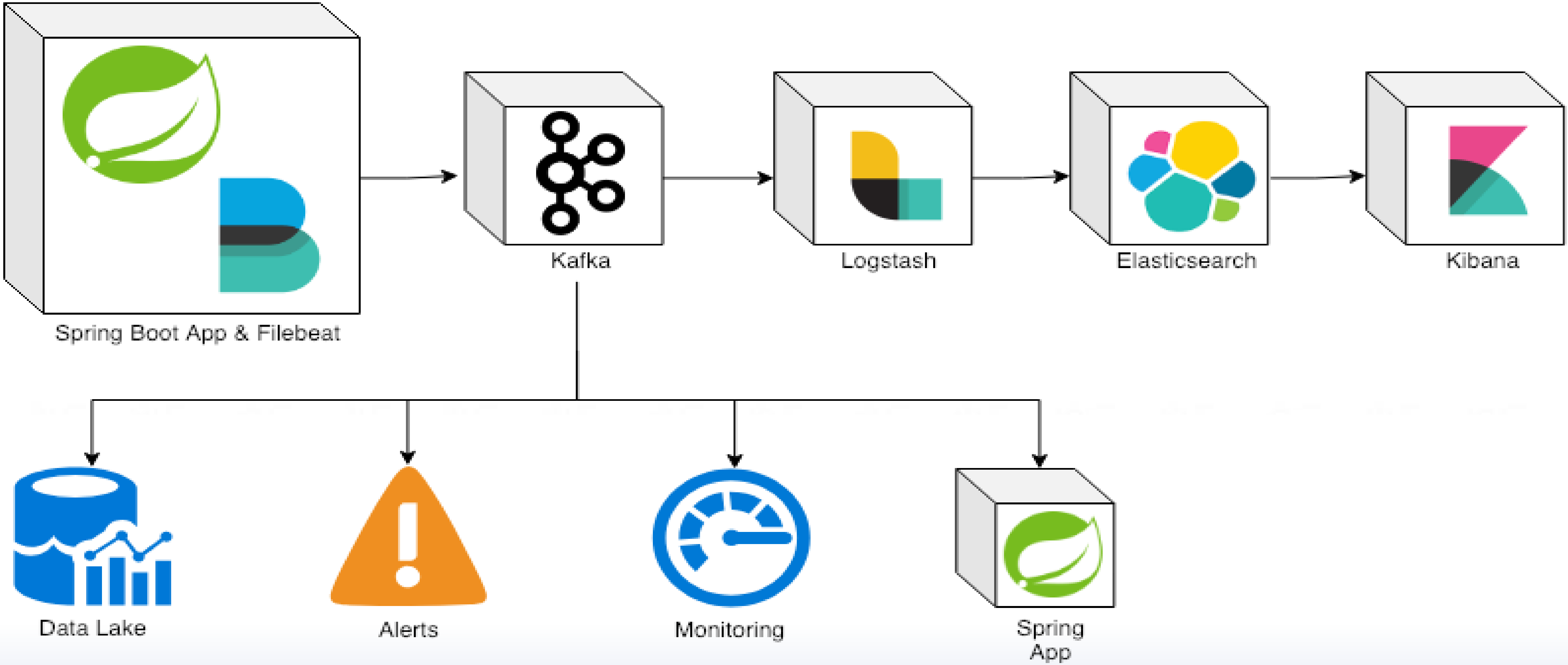


ELK

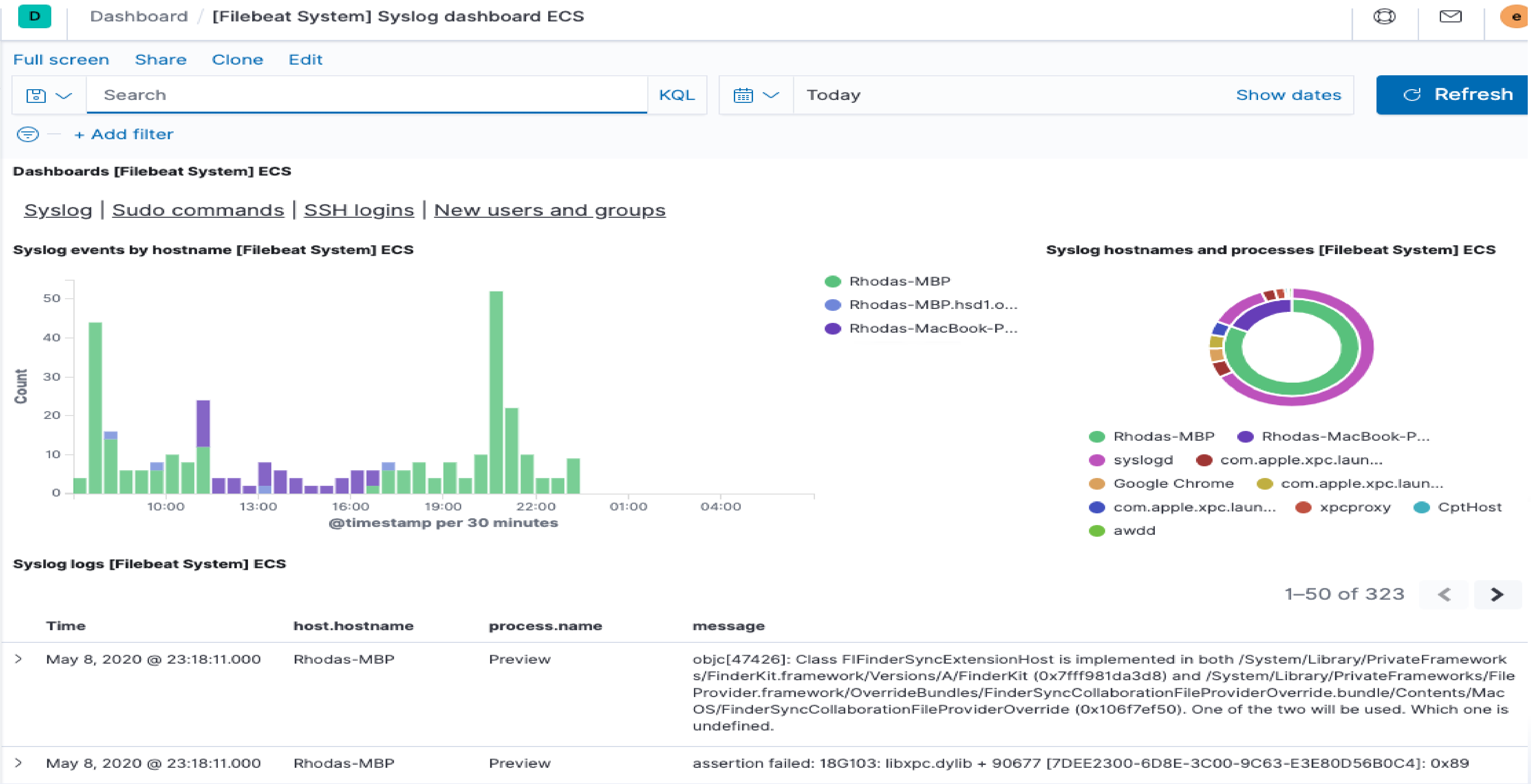
- Elasticsearch
- Logstash
- Kibana



Full log monitoring ecosystem



Kibana


D
Dashboard / [Filebeat System] Syslog dashboard ECS
e
Full screen
Share
Clone
Edit
Search
KQL
Today
Show dates
Refresh
+ Add filter
Dashboards [Filebeat System] ECS
Syslog | Sudo commands | SSH logins | New users and groups
Syslog events by hostname [Filebeat System] ECS
50
40
30
20
10
0
Count
10:00
13:00
16:00
19:00
22:00
01:00
04:00
@timestamp per 30 minutes
Rhodas-MBP
Rhodas-MBP.hsd1.o...
Rhodas-MacBook-P...
Syslog hostnames and processes [Filebeat System] ECS
Rhodas-MBP
Rhodas-MacBook-P...
syslogd
com.apple.xpc.laun...
Google Chrome
com.apple.xpc.laun...
com.apple.xpc.laun...
xpcproxy
CptHost
awdd
Syslog logs [Filebeat System] ECS
1–50 of 323
Time
host.hostname
process.name
message
May 8, 2020 @ 23:18:11.000
Rhodas-MBP
Preview
objc[47426]: Class FIFinderSyncExtensionHost is implemented in both /System/Library/PrivateFrameworks/FinderKit.framework/Versions/A/FinderKit (0x7fff981da3d8) and /System/Library/PrivateFrameworks/FileProvider.framework/OverrideBundles/FinderSyncCollaborationFileProviderOverride.bundle/Contents/MacOS/FinderSyncCollaborationFileProviderOverride (0x106f7ef50). One of the two will be used. Which one is undefined.
May 8, 2020 @ 23:18:11.000
Rhodas-MBP
Preview
assertion failed: 18G103: libxpc.dylib + 90677 [7DEE2300-6D8E-3C00-9C63-E3E80D56B0C4]: 0x89

What is ElasticSearch

- Document (Json) oriented search engine
- Distributed
- Horizontally scalable and highly available
- RESTful API
- Built on Lucene search engine library

- It is used for full-text search, analytics.

ElasticSearch

- ES has become de facto ***fast search*** solution

- Examples
  - GitHub uses ES to query 130 billion lines of code
  - Wikipedia uses ES to provide full-text search with highlighted search snippets
  - StackOverflow combines full-text search with geolocation queries and uses more-like-this to find related questions and answers

History

Shay Benon @kimchy

Elasticsearch released in February 2010.
Worked on this for 6 years (started with compass)
Now part of http://elastic.co commercial offerings

Doug Cutting @cutting

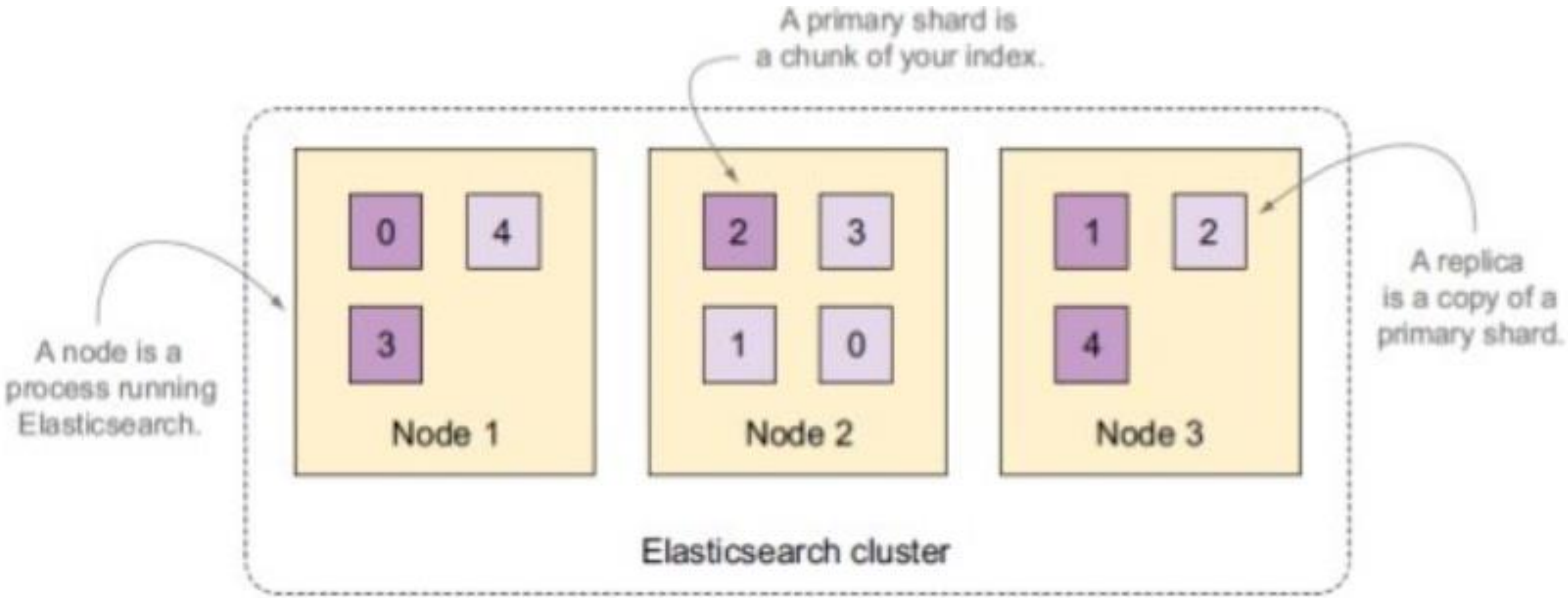
Started **Lucene** in 1999, released under apache in 2005.
Now part of cloudera supporting rival solution solr and commercial offerings

Building Blocks

Term	Description (~analogy with relational database)
Cluster	~Database cluster Group of nodes
Node	~Instance of database A JVM process, usually a machine
Index	~Database schema Hosts mapping types and their definitions contains many shards
Mapping Type	~Database Table Field description, indexing requirements
Document	~Database row Json document.
Shard	A Lucene index. Scalable unit and heart of search engine (*primary and replica*)

Relational DB	Elasticsearch 6
Table	Index
Row	Document
Column	Field

Physical Layout



Inverted Index

- **Step 1** - we need tokenize those docs into terms with a tokenizer. So, let's say, we use Tokenizer A and get following results:
  - Doc1: Harry, Potter, And, The, Half,
  - Doc2: Harry, Potter, And, The, Deathly
- **Step 2** - Build inverted index

  Harry -> Doc1, Doc2
  Potter -> Doc1, Doc2
  And -> Doc1, Doc2
  The -> Doc1, Doc2
  Half -> Doc1
  Blood -> Doc1
  Prince -> Doc1
  Deathly -> Doc2
  Hallows -> Doc2

To query/search

- **Step 1** - We also need tokenize search words at first. Such as, our search words are Harry Potter.
  And you have two tokenizer to choose.
  Tokenizer A is same with the one we use during indexing, will tokenize our words into two terms: Harry and Potter.

- **Step 2** - Do query
  If you choose Tokenizer A,
  you get Harry and Potter,
  both of them are in our inverted index,
  then you can get search results:
  Doc1 and Doc2.

Lucene Inverted Index

A shard is a Lucene index.

get-together0 shard

Inverted index

Term	Document	Frequency
elasticsearch	id1	1 occurrence: id1->1 time
denver	id1,id3	3 occurrences: id1->1 time, id3->2 times
clojure	id2,id3	5 occurrences: id2->2 times, id3->3 times
data	id2	2 occurrences: id2->2 times

ES: Lucene +

- Distributed
- Transaction Log
    - The translog **stores all operations that are not yet safely persisted in Lucene**. Although these operations are available for reads, they will need to be replayed if the shard was stopped and had to be recovered.

Simplifies shared relocation/recovery

- Query DSL
    - Provides set of grammar for search syntax

Index>Type>Document>Field

Vehicles(index)
- Car(document 1)
- Car (document 2)
- Car (document n)

```
{
        “type”: “car”,
        “documents”: [
        {
                “id”: 23134,
                “make”: “Honda”,
                “color”: “red”,
                “mileage”: 8000
        },
        {
                “id”: 12334,
                “make”: “Ford”,
                “color”: “white”,
                “mileage”: 5000
        }
        ]
}
```

index docs

Inserting = indexing

```
PUT /{index}/{type}/{id}
{
   "field1": "value1",
   "filed2": "value2",
…
}
```

```
PUT /vehicles/car/123
{
   "make": "Ford",
   "mileage": 5000,
   "color": "red"
}
```

Search Types

Count

Returns no hits, only total count matching the query

Scan

Allows to iterate over large amounts of data using a cursor to paginate and memory efficient

Search

General search

Query DSL (Domain Specific Language)

```
Get /courses/_search
{
        "query": {
                "match_all": {}
    }
}
```

```
Get /courses/_search
{
        "query": {
                "match":
{"name": "computer" }
    }
}
```

```
{
  "took": 0,
  "timed_out": false,
  "_shards": {
     "total": 1,
     "successful": 1,
     "failed": 0
},
 "hits": {
   "total": 1,
   "max_score": 0.81233,
   "hits" : [
       {
          "_inde": "courses",
          "_type": "classroom"
          "_id": "3",
          "_score": 0.8243,
          "_source": {
          "name": "Computer Eng 101",
          "room": "ENG202"
          "student enrolled":  23
       }
   ]
}
```

Match query

The match query is the standard query for performing a full-text search.

```
GET /_search
{
  "query": {
    "match": {
      "message": {
        "query": "this is a test"
      }
    }
  }
}
```

Getting a document

Request:

```
GET test/cities/1?pretty
```

Response:

```
{
    "_index": "test",
    "_type": "cities",
    "_id": "1",
    "_version": 1,
    "found": true,
    "_source": {
        "rank": 3,
        "city": "Hyderabad",
        "state": "Telangana",
        "population2014": 7750000,
        "land_area": 625,
        "location": {
            "lat": 17.37,
            "lon": 78.48
        },
        "abbreviation": "Hyd"
    }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-query.html

Document Metadata Fields

- ***_id*** - The id of the document
- ***_type*** - The document type
- ***_source*** - enabled Stores the original document that was indexed
- ***_all*** enabled Indexes all values of all document fields
- ***_timestamp*** disabled timestamp associated with the document
- ***_ttl*** disabled optionally defines an expiration time
- ***_size*** disabled indexes the size of the uncompressed

Query DSL (Domain Specific Language)

```
% curl 'localhost:9200/get-together/_search?sort=date:asc&_source=title,date'
{
   _"index": "get-together",
   _"type": "event",
   _"id": "114",
   _"score": null,
   _"source": {
        "date": "2013-09-09T18:30",
        "title": "Using Hadoop with Elasticsearch"
   },
   "sort": [
        1378751400000
   ]
},
```

Show one hit of the response.

Request matching all documents but return the default first 10 of all results ordered by date in ascending order. You want only two fields in the response: title and date.

The filtered _source document now contains only filtered fields.

The score is null; you're using a sort and therefore no score is calculated.

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html

Aggregation

- An aggregation summarizes your data as metrics, statistics, or other analytics. Aggregations help you answer questions like:
- What's the average load time for my website?

```
GET /my-index-000001/_search
{
"aggs": {
"my-agg-name": {
"terms": {
"field": "my-field"
}
}
}
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations.html

demo

Web application with backend DB as ES, autocomplete.
https://www.youtube.com/watch?v=hVSC4ZNiVdA

Identifying anomalies and forecasting with ES
https://www.youtube.com/watch?v=wJVgh5knV4E

the ELK Stack: Elasticsearch, Kibana, Beats, and Logstash
https://www.youtube.com/watch?v=DRQJHOOstY0

Application Performance Monitoring (APM) with Elasticsearch, Elastic Stack
https://www.youtube.com/watch?v=2sdOvuLiBb8

Any volunteer to build a **full stack ML** with ELK, node.js and React.js?