# New Data Architectures For Netflow Analytics
# NANOG 74

Fangjin Yang - Cofounder @ imply

# Overview

**The Problem**

**Comparing technologies**

**Operational analytic databases**

**Try this at home**

# The Problem

Netflow data continues to grow in scale and complexity

This data is critical for network analysts, operators, and monitoring teams to understand performance and troubleshoot issues

Real-time and historical analysis are both important

# The Data

Flows come in many different forms: Netflows, sFlow, IP-Fix, etc
Network data may often be enriched with application/user data as well.

What is in common in all data is 3 components:

- Timestamp: when the flow was created
- Attributes (dimensions): properties that describe the flow
- Measures (metrics): numbers to aggregate (# flows, # packets, bps, etc)

# The Use Case

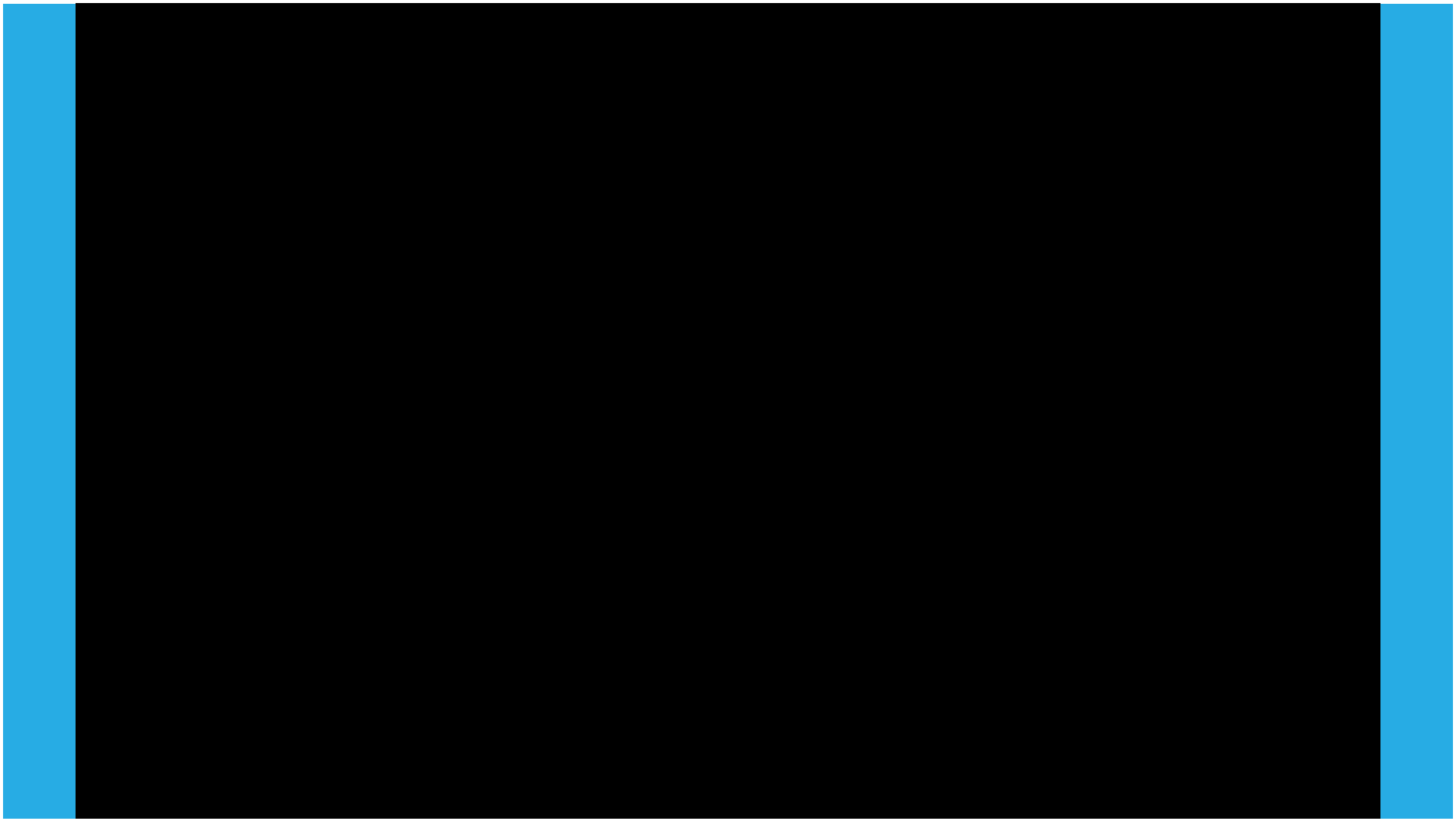
We want to monitor the network and proactively detect issues

When issues occur, we want to resolve them as fast as possible

We want our analytics system to be cheap and efficient

# Demo

In case the internet didn’t work,
pretend you saw something cool



# Challenges

Storing and computing the data:

- Scale: millions events/sec (batch and real-time)
- Complexity: high dimensionality & high cardinality
- Structure: semi-structured (nested, evolving schemas, etc.)

Accessing the data:

- UIs: need both reporting and exploring capabilities
- Troubleshooting: need fast drill downs to quickly pinpoint issues
- Multi-tenancy: support up to thousands of concurrent users across different networking teams

# What technology to use?

Logsearch systems (Elasticsearch)

Analytic query engines (Presto, Vertica)

Timeseries databases (InfluxDB, OpenTSDB)

Something new?

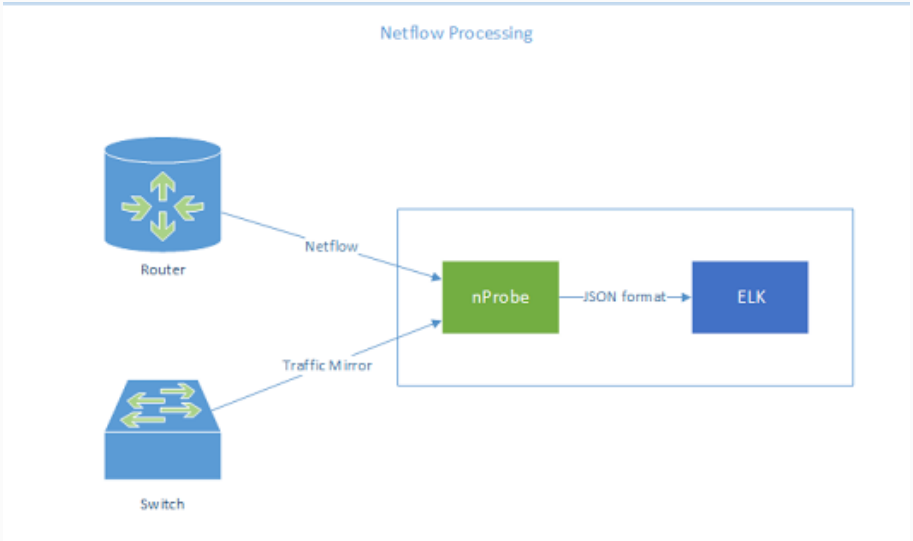
# Logsearch systems

Pros:

- Real-time ingest
- Flexible schemas
- Fast search and filter

Cons:

- Poor performance for high dimensional, high cardinality data
- Not designed for numerical aggregation



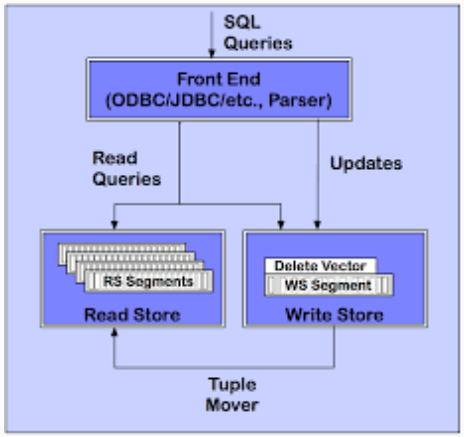
# Analytic query engines

Pros:

- Column-oriented storage: fast aggregations
- Supports large scale groupBys
- Supports complex aggregations

Cons:

- Not designed for real-time ingest
- Inflexible schemas
- Slow search and filter



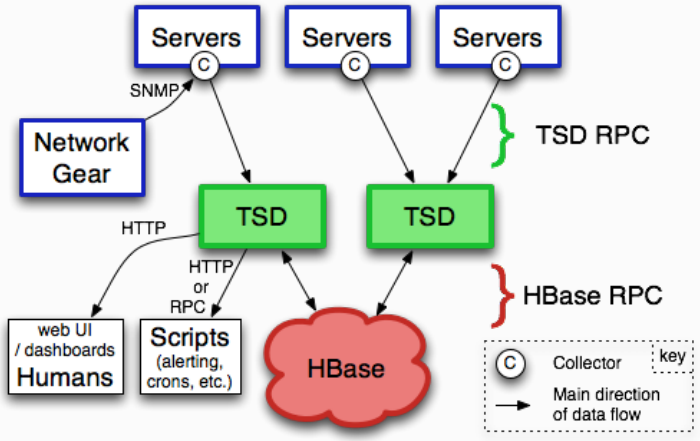
# Time series databases

Pros:

- Time-optimized partitioning and storage
- Able to quickly aggregate time series

Cons:

- Slow for groupings on dimensions that are not time
- Slow search and filter



# Operational Analytics Database

Combines benefits of all 3 classes of systems

Ingest and search/filter capabilities of logsearch

Column-oriented storage and query capabilities of analytic query engines

Time-specific optimizations from time series databases

# Operational Analytics Database

Examples:

- Apache Druid (incubating) (open source community)
- Scuba (from Facebook)
- Pinot (from LinkedIn)
- Palo (from Baidu)
- Clickhouse (from Yandex)

Operational analytic databases have near identical storage formats and similar architecture.

We'll use Druid to explain the architecture.

# Druid in Production

Alibaba: https://strata.oreilly.com.cn/hadoop-big-data-cn/public/schedule/detail/52345

Cisco: http://www.networkworld.com/article/3086250/cisco-subnet/under-the-hood-of-cisco-s-tetration-analytics-platform.html

eBay: https://www.slideshare.net/gianmerlino/druid-for-real-time-monitoring-analytics-at-ebay

Netflix: https://www.youtube.com/watch?v=Dlqj34l2upk

Paypal: https://dataworkssummit.com/san-jose-2018/session/paypal-merchant-ecosystem-using-apache-spark-hive-druid-and-hbase/

Walmart: https://medium.com/walmartlabs/event-stream-analytics-at-walmart-with-druid-dcf1a37ceda7

Wikimedia Foundation: https://conferences.oreilly.com/strata/strata-ny-2017/public/schedule/detail/60986

Airbnb, Lyft, Slack, Snapchat, Target, Tencent, Verizon + many more

http://druid.io/druid-powered.html for more info

# Storage Format

# Raw data

| timestamp | Action | Protocol | Flows |
|---|---|---|---|
| 2011-01-01T00:01:35Z | ACCEPT | TCP | 10 |
| 2011-01-01T00:03:03Z | ACCEPT | TCP | 1 |
| 2011-01-01T00:04:51Z | REJECT | UDP | 10 |
| 2011-01-01T00:05:33Z | REJECT | UDP | 10 |
| 2011-01-01T00:05:53Z | REJECT | TCP | 1 |
| 2011-01-01T00:06:17Z | REJECT | TCP | 10 |
| 2011-01-01T00:23:15Z | ACCEPT | TCP | 1 |
| 2011-01-01T00:38:51Z | REJECT | UDP | 10 |
| 2011-01-01T00:49:33Z | REJECT | TCP | 10 |
| 2011-01-01T00:49:53Z | REJECT | TCP | 1 |

# Rollup

| timestamp | Action | Protocol | Flows |
|---|---|---|---|
| 2011-01-01T00:01:35Z | ACCEPT | TCP | 10 |
| 2011-01-01T00:03:03Z | ACCEPT | TCP | 1 |
| 2011-01-01T00:04:51Z | REJECT | UDP | 10 |
| 2011-01-01T00:05:33Z | REJECT | UDP | 10 |
| 2011-01-01T00:05:53Z | REJECT | TCP | 1 |
| 2011-01-01T00:06:17Z | REJECT | TCP | 10 |
| 2011-01-01T00:23:15Z | ACCEPT | TCP | 1 |
| 2011-01-01T00:38:51Z | REJECT | UDP | 10 |
| 2011-01-01T00:49:33Z | REJECT | TCP | 10 |
| 2011-01-01T00:49:53Z | REJECT | TCP | 1 |

| timestamp | Action | Protocol | Flows |
|---|---|---|---|
| 2011-01-01T00:00:00Z | ACCEPT | TCP | 12 |
| 2011-01-01T00:00:00Z | REJECT | TCP | 22 |
| 2011-01-01T00:00:00Z | REJECT | UDP | 30 |

# Sharding/partitioning data

| timestamp | Action | Protocol | Flows |
|---|---|---|---|
| 2011-01-01T00:00:00Z | ACCEPT | TCP | 12 |
| 2011-01-01T00:00:00Z | REJECT | TCP | 22 |
| ... | | | |

→ 1st hour **segment**

| | | | |
|---|---|---|---|
| 2011-01-01T01:00:00Z | ACCEPT | TCP | 12 |
| 2011-01-01T01:00:00Z | REJECT | TCP | 22 |
| ... | | | |

→ 2nd hour **segment**

| | | | |
|---|---|---|---|
| 2011-01-01T02:00:00Z | ACCEPT | TCP | 12 |
| 2011-01-01T02:00:00Z | REJECT | TCP | 22 |
| ... | | | |

→ 3rd hour **segment**

# Segments

Fundamental storage unit in Druid

Immutable once created

No contention between reads and writes

One thread scans one segment

# Columnar storage - compression

| timestamp | Action | Protocol | Flows |
|---|---|---|---|
| 2011-01-01T00:00:00Z | ACCEPT | TCP | 10 |
| 2011-01-01T00:03:03Z | ACCEPT | TCP | 1 |
| 2011-01-01T00:04:51Z | REJECT | UDP | 10 |
| 2011-01-01T00:05:33Z | REJECT | UDP | 10 |
| 2011-01-01T00:05:53Z | REJECT | TCP | 1 |
| 2011-01-01T00:06:17Z | REJECT | TCP | 10 |

Create IDs

- Accept → 0, Reject → 1
- TCP → 0, UDP → 1

Store

- Action → **[0 0 1 1 1 1]**
- Protocol → **[0 0 1 1 0 0]**

# Columnar storage - fast search and filter

| timestamp | Action | Protocol | Flows |
|---|---|---|---|
| 2011-01-01T00:00:00Z | ACCEPT | TCP | 10 |
| 2011-01-01T00:03:03Z | ACCEPT | TCP | 1 |
| 2011-01-01T00:04:51Z | REJECT | UDP | 10 |
| 2011-01-01T00:05:33Z | REJECT | UDP | 10 |
| 2011-01-01T00:05:53Z | REJECT | TCP | 1 |
| 2011-01-01T00:06:17Z | REJECT | TCP | 10 |

ACCEPT → [0, 1] → [110000]
REJECT → [2, 3, 4, 5] → [001111]

ACCEPT OR REJECT → [111111]

Compression!

# Approximate algorithms

Many aggregations in netflow use cases don't require exactness

- Count distinct
- Histograms and quantiles
- Set analysis

Approximate algorithms are very powerful for fast queries while minimizing storage

# Rollup revisited

| timestamp | Action | Protocol | Flows |
|---|---|---|---|
| 2011-01-01T00:01:35Z | ACCEPT | TCP | 10 |
| 2011-01-01T00:03:03Z | ACCEPT | TCP | 1 |
| 2011-01-01T00:04:51Z | REJECT | UDP | 10 |
| 2011-01-01T00:05:33Z | REJECT | UDP | 10 |
| 2011-01-01T00:05:53Z | REJECT | TCP | 1 |
| 2011-01-01T00:06:17Z | REJECT | TCP | 10 |
| 2011-01-01T00:23:15Z | ACCEPT | TCP | 1 |
| 2011-01-01T00:38:51Z | REJECT | UDP | 10 |
| 2011-01-01T00:49:33Z | REJECT | TCP | 10 |
| 2011-01-01T00:49:53Z | REJECT | TCP | 1 |

| timestamp | Action | Protocol | Flow |
|---|---|---|---|
| 2011-01-01T00:00:00Z | ACCEPT | TCP | 12 |
| 2011-01-01T00:00:00Z | REJECT | TCP | 22 |
| 2011-01-01T00:00:00Z | REJECT | UDP | 30 |

# High cardinality dimensions impact rollup

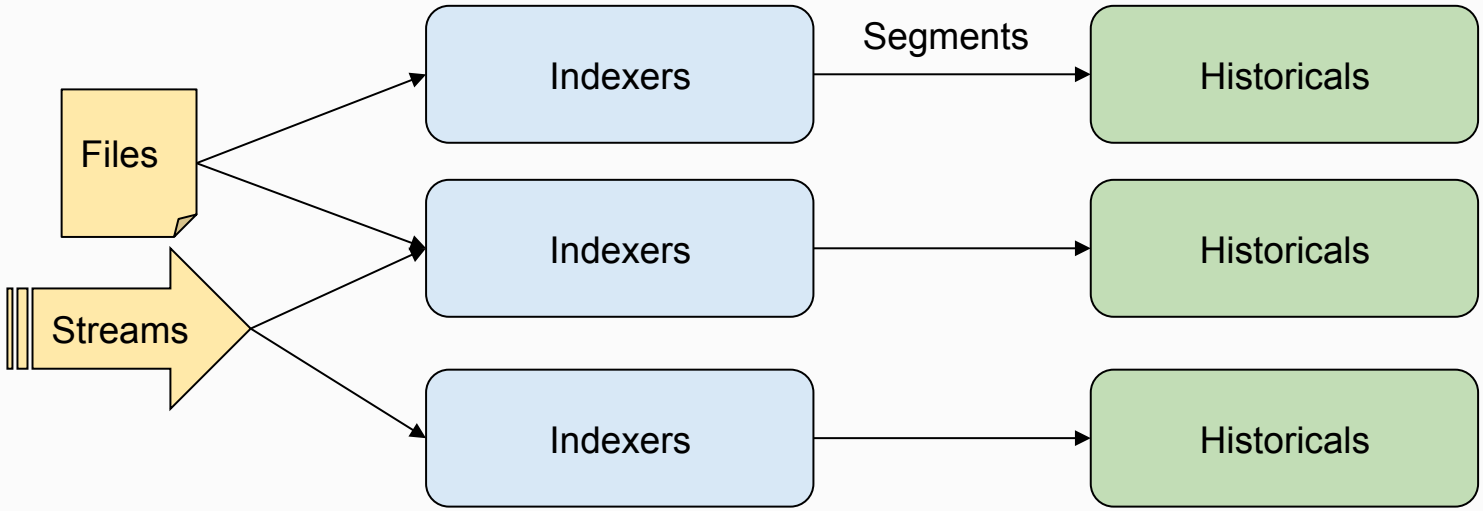
| timestamp | Action | device_id | Protocol | Flows |
|---|---|---|---|---|
| 2011-01-01T00:01:35Z | ACCEPT | 4312345532 | TCP | 10 |
| 2011-01-01T00:03:03Z | ACCEPT | 3484920241 | TCP | 1 |
| 2011-01-01T00:04:51Z | REJECT | 9530174728 | UDP | 10 |
| 2011-01-01T00:05:33Z | REJECT | 4098310573 | UDP | 10 |
| 2011-01-01T00:05:53Z | REJECT | 5832058870 | TCP | 1 |
| 2011-01-01T00:06:17Z | REJECT | 5789283478 | TCP | 10 |
| 2011-01-01T00:23:15Z | ACCEPT | 4730093842 | TCP | 1 |
| 2011-01-01T00:38:51Z | REJECT | 9530174728 | UDP | 10 |
| 2011-01-01T00:49:33Z | REJECT | 4930097162 | TCP | 10 |
| 2011-01-01T00:49:53Z | REJECT | 3081837193 | TCP | 1 |

# Approximate counts enable rollup again

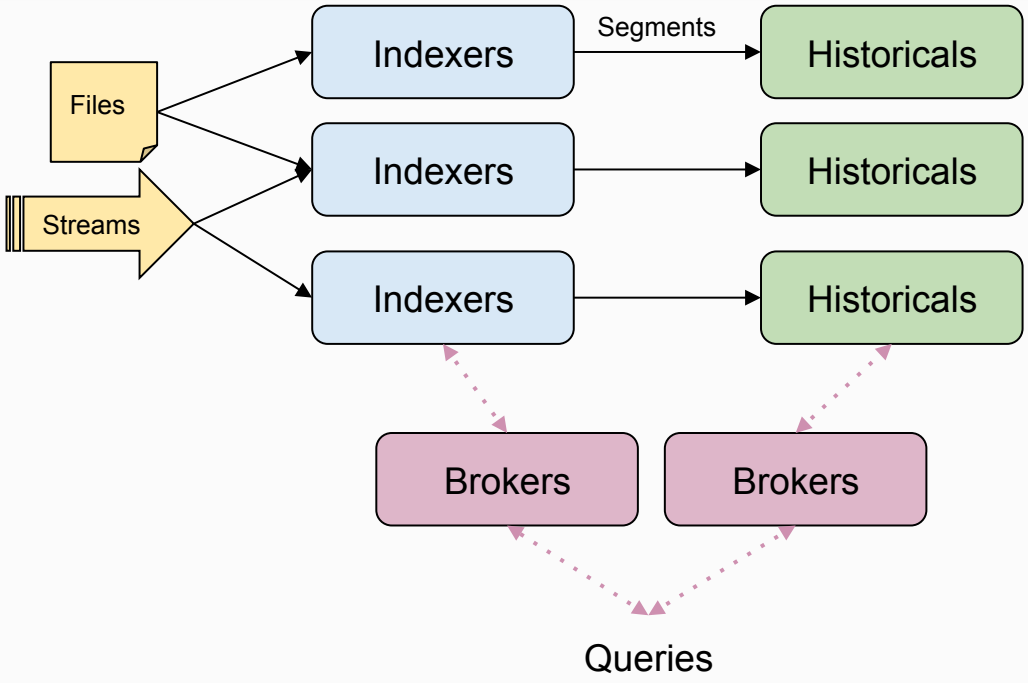
| timestamp | Action | Protocol | Flow | unique_device_count |
|---|---|---|---|---|
| 2011-01-01T00:00:00Z | ACCEPT | TCP | 12 | [sketch] |
| 2011-01-01T00:00:00Z | REJECT | TCP | 22 | [sketch] |
| 2011-01-01T00:00:00Z | REJECT | UDP | 30 | [sketch] |

# Architecture

# Architecture (Ingestion)



# Architecture



# Querying

Query libraries:

- JSON over HTTP
- SQL
- R
- Python
- Ruby

# By the numbers

Supports high scale:

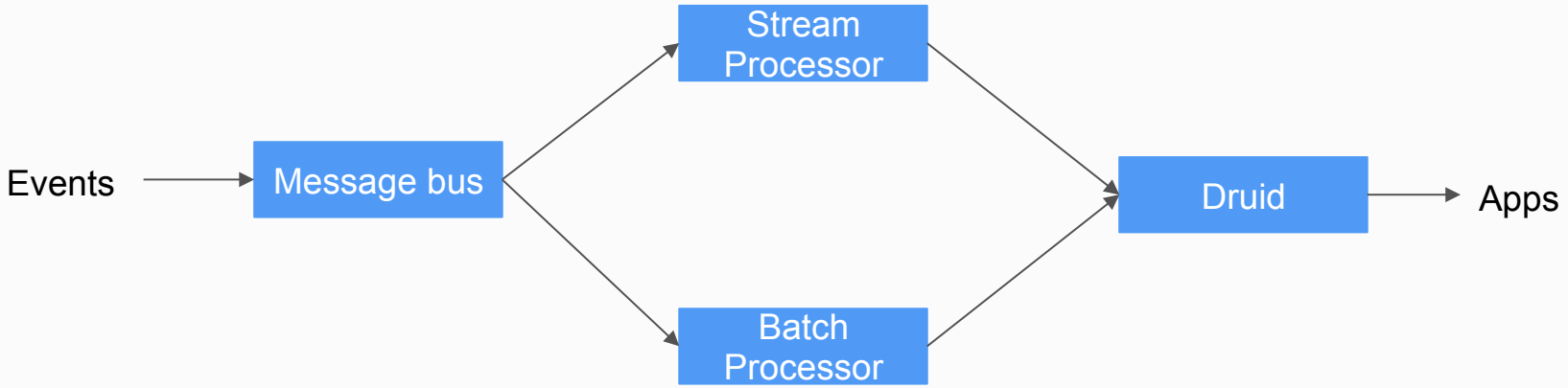
- 300B events daily on a 10K core Druid cluster
- 100 Trillion events in the total database
- From https://metamarkets.com/2017/benefits-of-a-full-stack-solution/

On 10TB of data (~50B rows):

- Avg query latency of 550 ms
- 90% of queries return < 1s, 95% < 2s, and 99% < 10s
- 22,914.43 events/second/core on a datasource with 30 dimensions and 19 metrics, running an Amazon cc2.8xlarge instance.
- From Druid whitepaper (http://static.druid.io/docs/druid.pdf)

# End to end data architecture

# End-to-end data stack



# Connect

Works well with:

- Kafka
- Hadoop
- S3
- Spark Streaming
- Storm
- Samza
- I bet some other things that also start with S

Do try this at home

# Download

http://druid.io/downloads.html

or

https://imply.io/download

# Thanks!

fj@imply.io
@fangjin