DATADOG

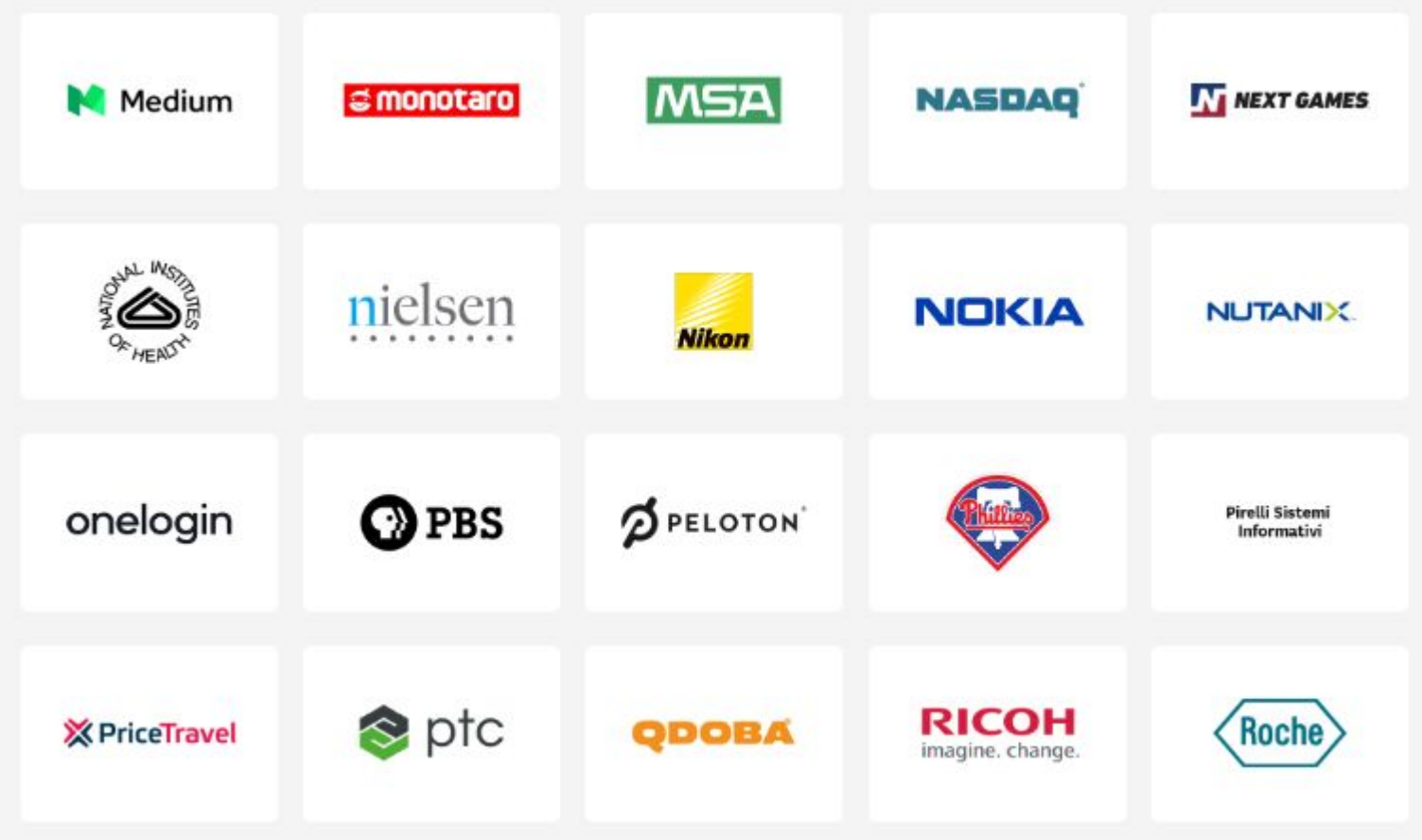
Datadog: A Real-Time Metrics Database for Trillions of Points/Day

Ian NOWLAND (https://twitter.com/inowland)
VP, Metrics and Monitors

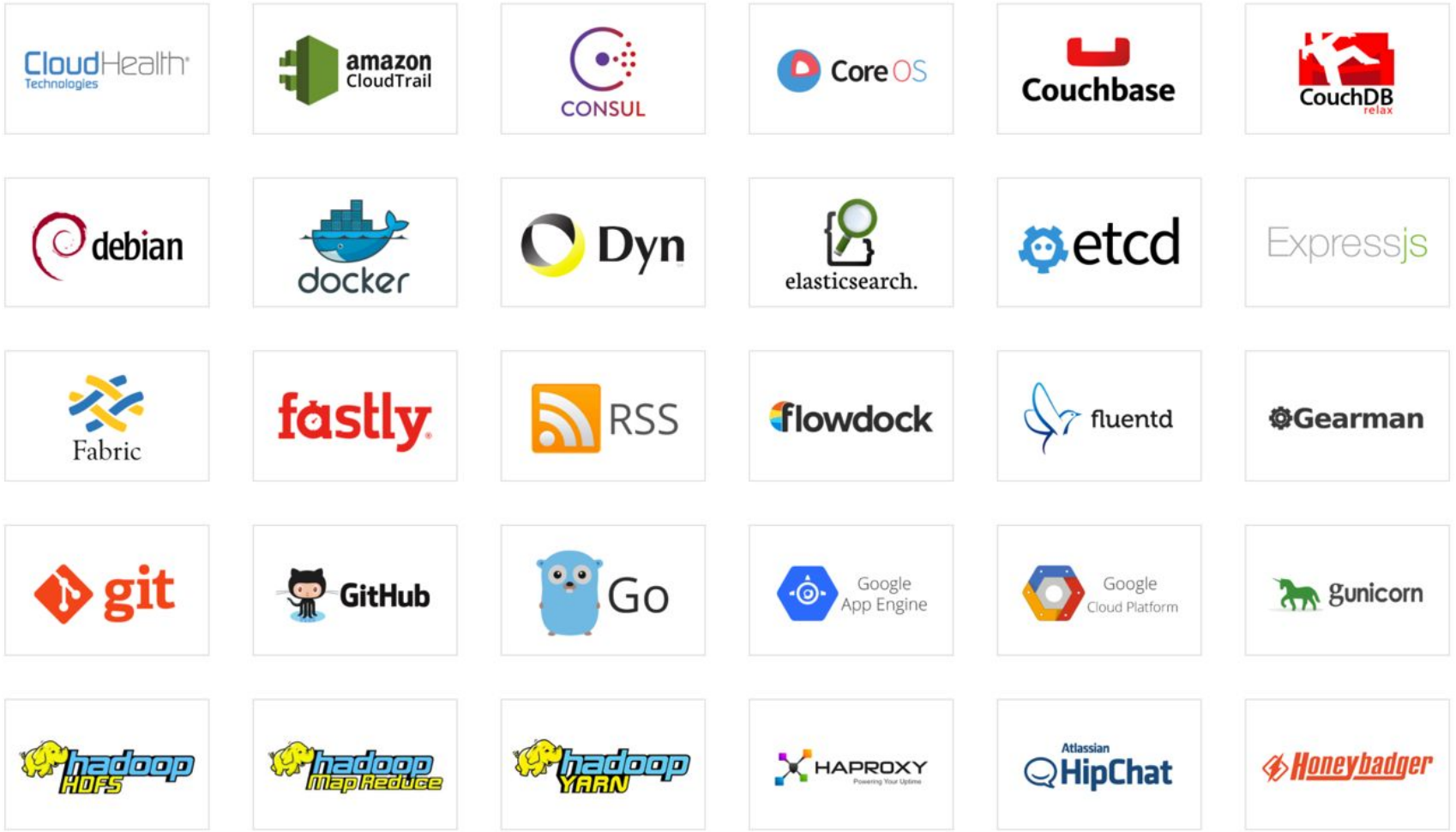
Joel BARCIAUSKAS (https://twitter.com/JoelBarciauskas)
Director, Aggregation Metrics

QCon NYC '19

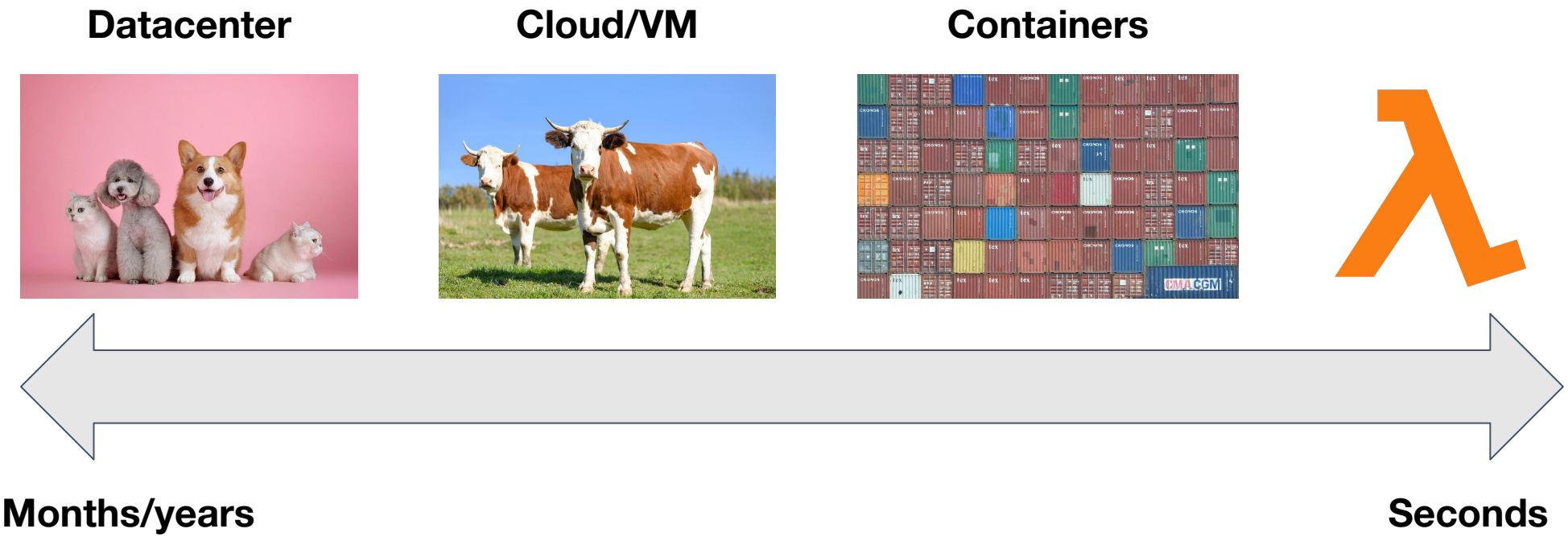
Some of Our Customers



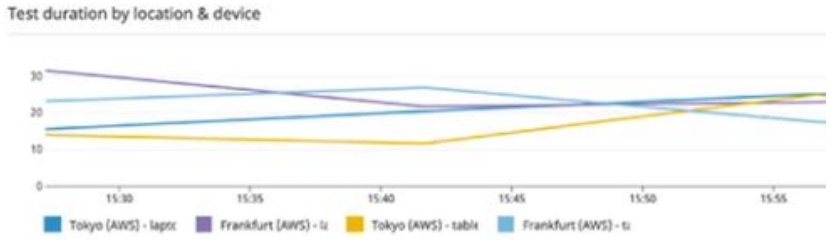
Some of What We Store



Changing Source Lifecycle

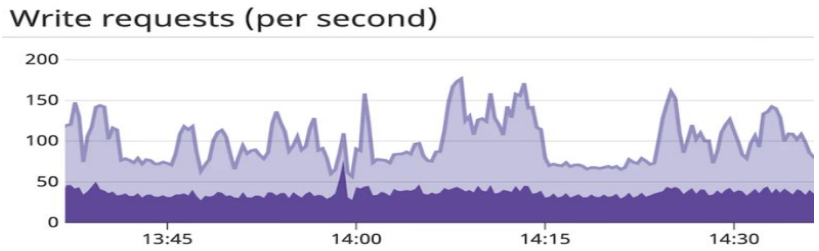


Changing Data Volume

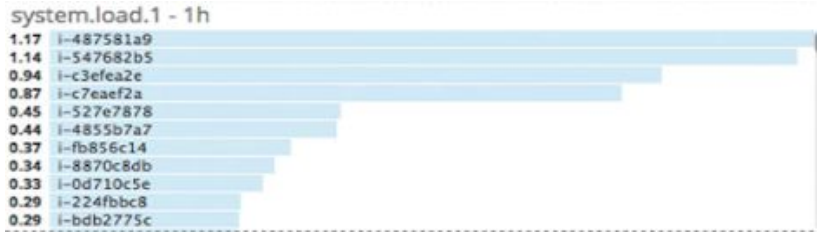


Per User Device

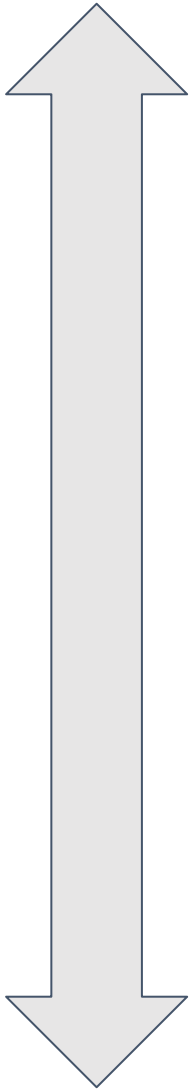
SLIs



Application



System



10,000's

100's

Applying Performance Mantras

- Don't do it
- Do it, but don't do it again
- Do it less
- Do it later
- Do it when they're not looking
- Do it concurrently
- Do it cheaper

**From Craig Hanson and Pat Crain, and the performance engineering community - see http://www.brendangregg.com/methodology.html*

Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture

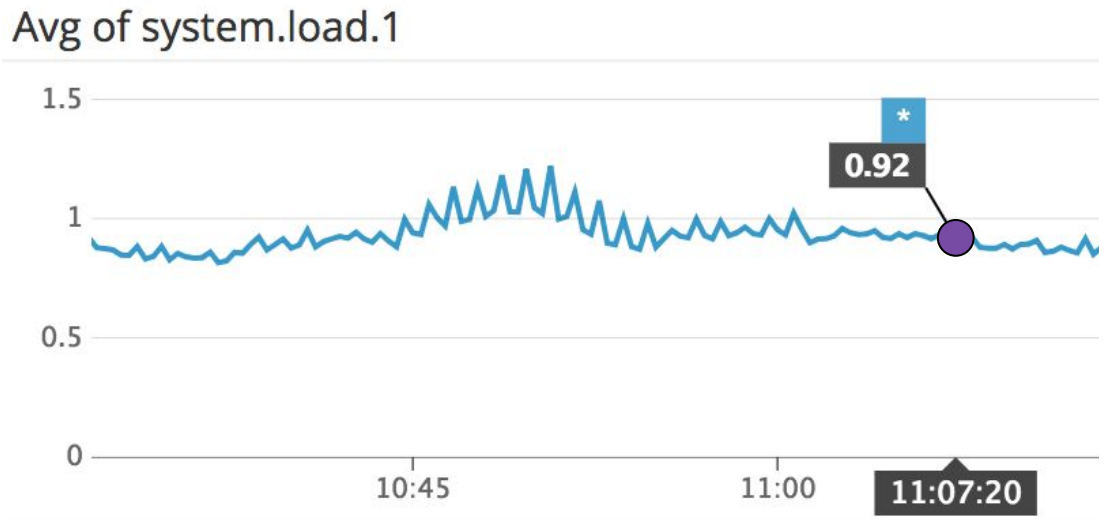
Talk Plan

1. **What Are Metrics Databases?**
2. Our Architecture
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture

Example Metrics Query 1

“What is the system load on instance i-xyz across the last 30 minutes”

A Time Series



metric	system.load.1
timestamp	1526382440
value	0.92
tags	host:i-xyz,env:dev,...

Example Metrics Query 2

“Alert when the system load, averaged across our fleet in us-east-1a for a 5 minute interval, goes above 90%”

Example Metrics Query 2

“Alert when the system load, averaged across my fleet in us-east-1a for a 5 minute interval, goes above 90%”

Take Action

Aggregate

Dimension

Metrics Name and Tags

Name: single string defining what you are measuring, e.g.

system.cpu.user

aws.elb.latency

dd.frontend.internal.ajax.queue.length.total

Tags: list of k:v strings, used to qualify metric and add dimensions to filter/aggregate over, e.g.

['host:server-1', 'availability-zone:us-east-1a', 'kernel_version:4.4.0']

['host:server-2', 'availability-zone:us-east-1a', 'kernel_version:2.6.32']

['host:server-3', 'availability-zone:us-east-1b', 'kernel_version:2.6.32']

Tags for all the dimensions

Host / container: system metrics **by host**

Application: internal cache hit rates, timers **by module**

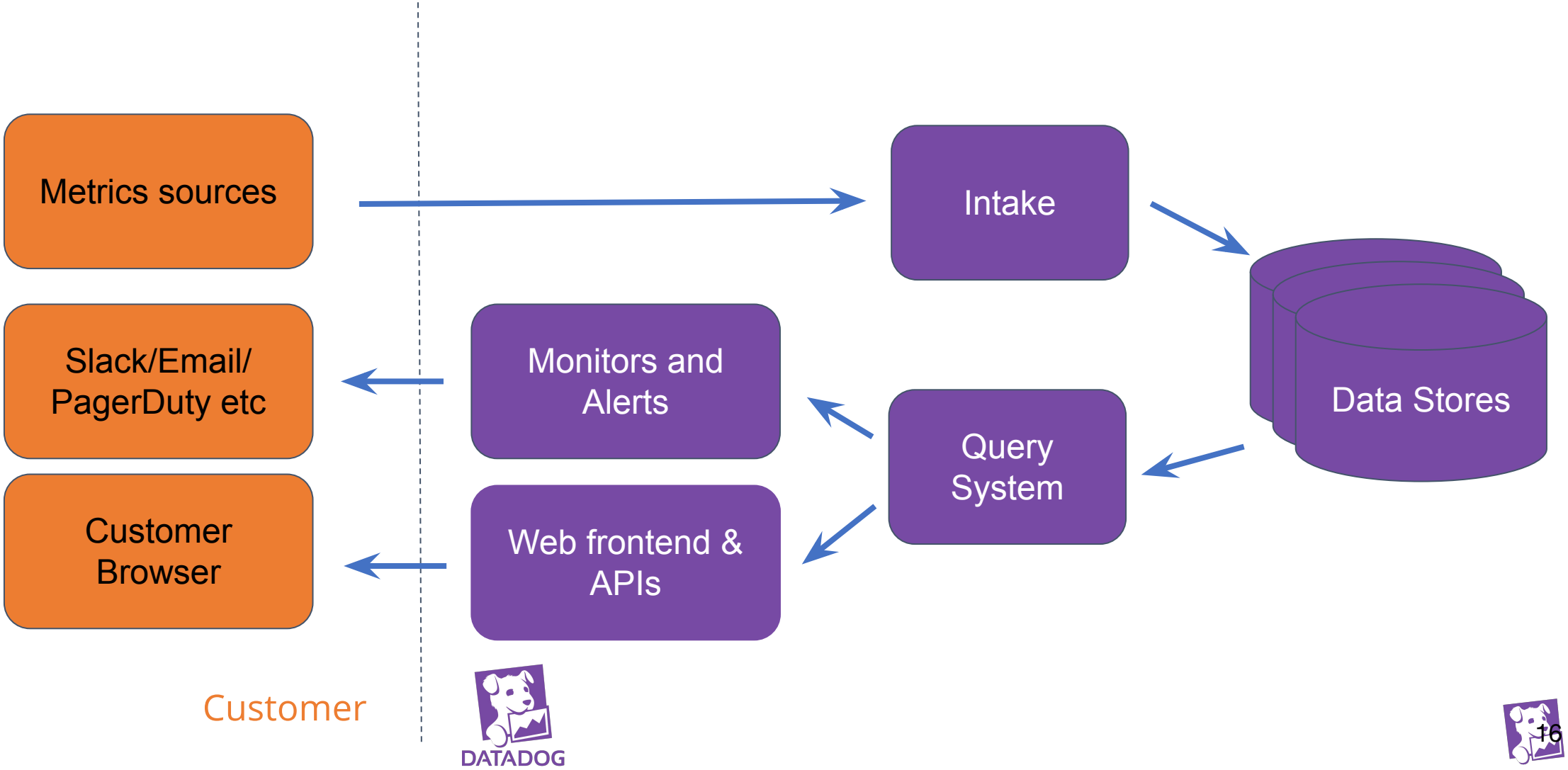
Service: hits, latencies or errors/s by **path** and/or **response** code

Business: # of orders processed, $'s per second by **customer** ID

Talk Plan

1. What Are Metrics Databases?
2. **Our Architecture**
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture

Pipeline Architecture



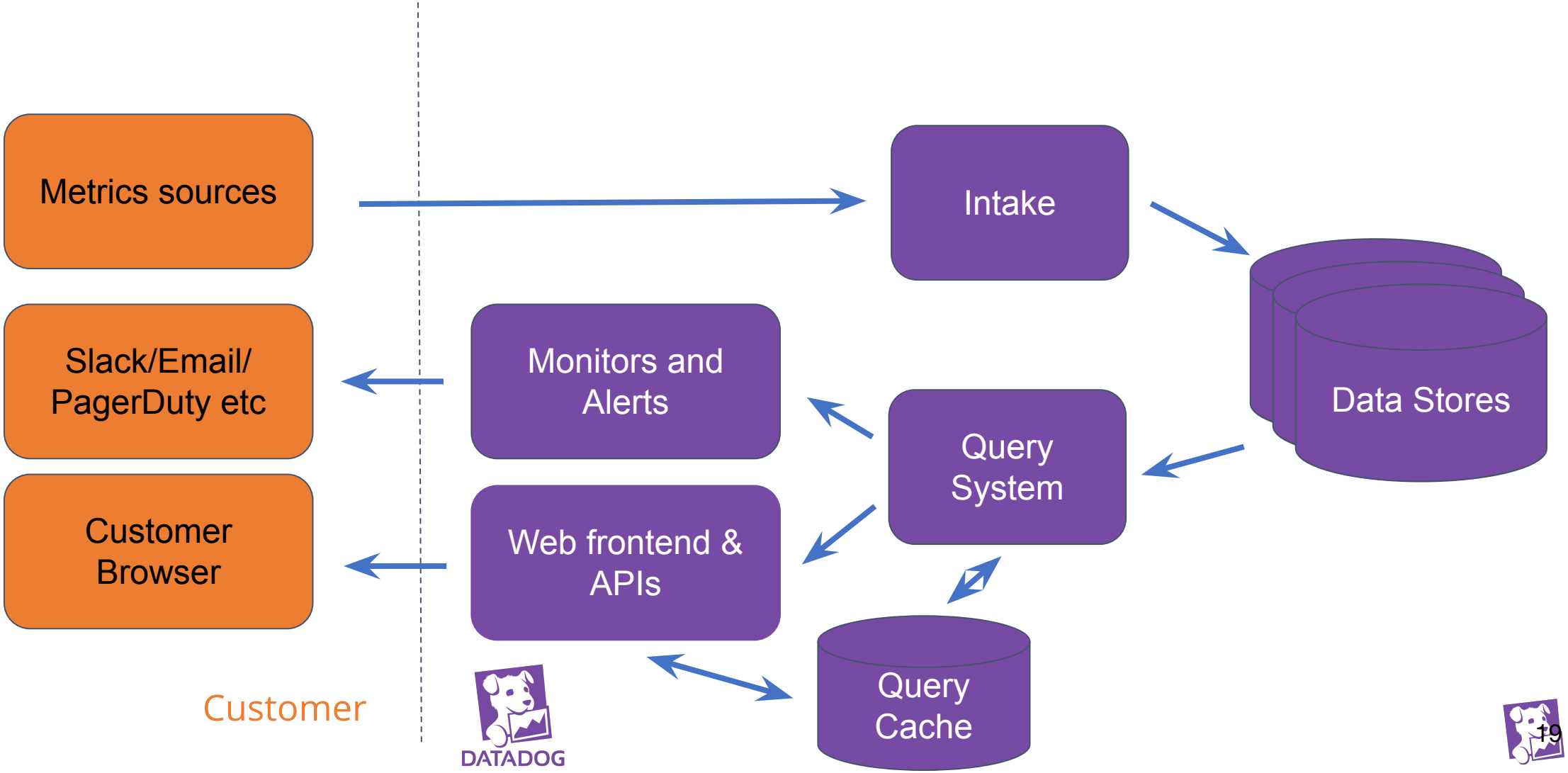
Performance mantras

- Don't do it
- **Do it, but don't do it again**
- Do it less
- Do it later
- Do it when they're not looking
- Do it concurrently
- Do it cheaper

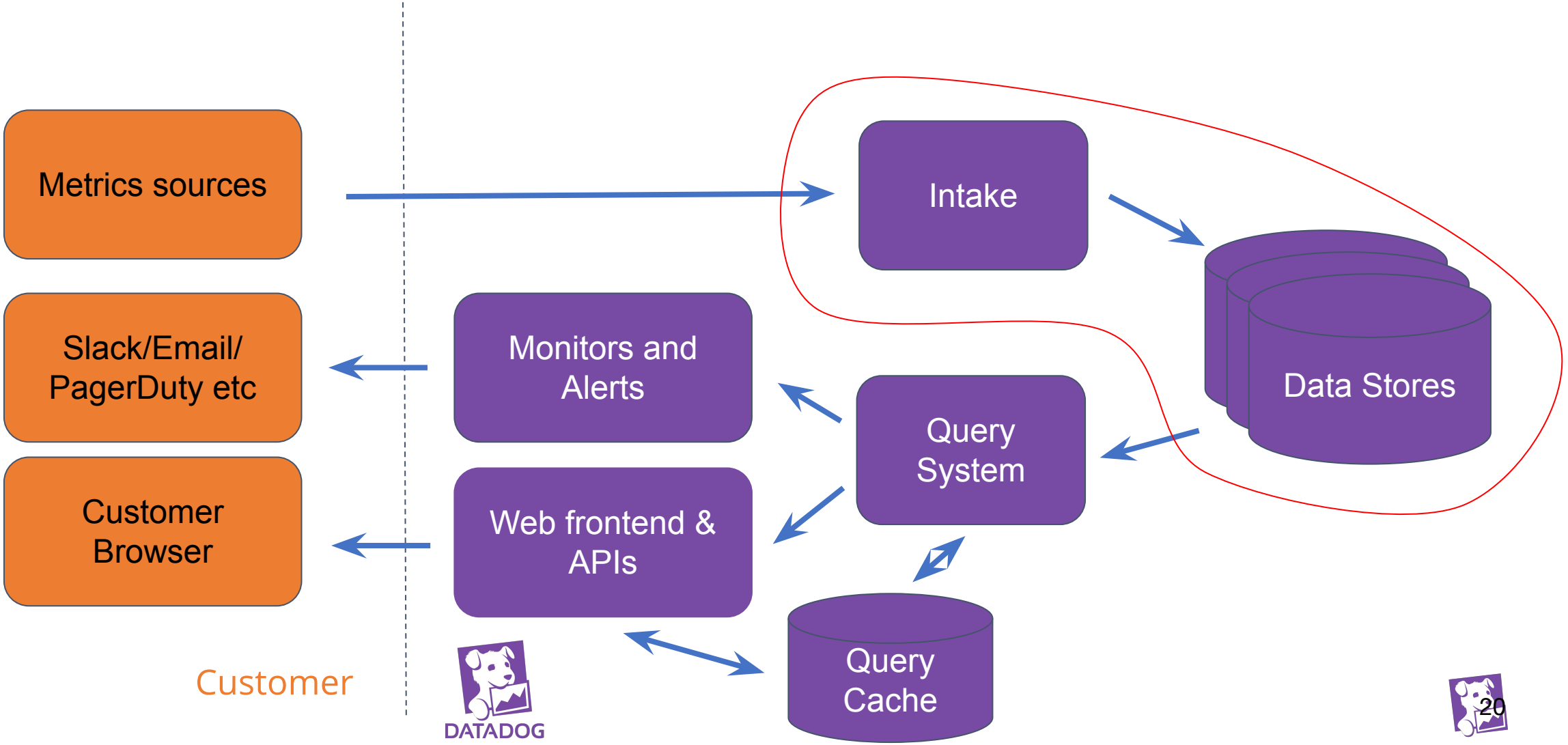
Performance mantras

- Don't do it
- **Do it, but don't do it again - query caching**
- Do it less
- Do it later
- Do it when they're not looking
- Do it concurrently
- Do it cheaper

Pipeline Architecture



Pipeline Architecture

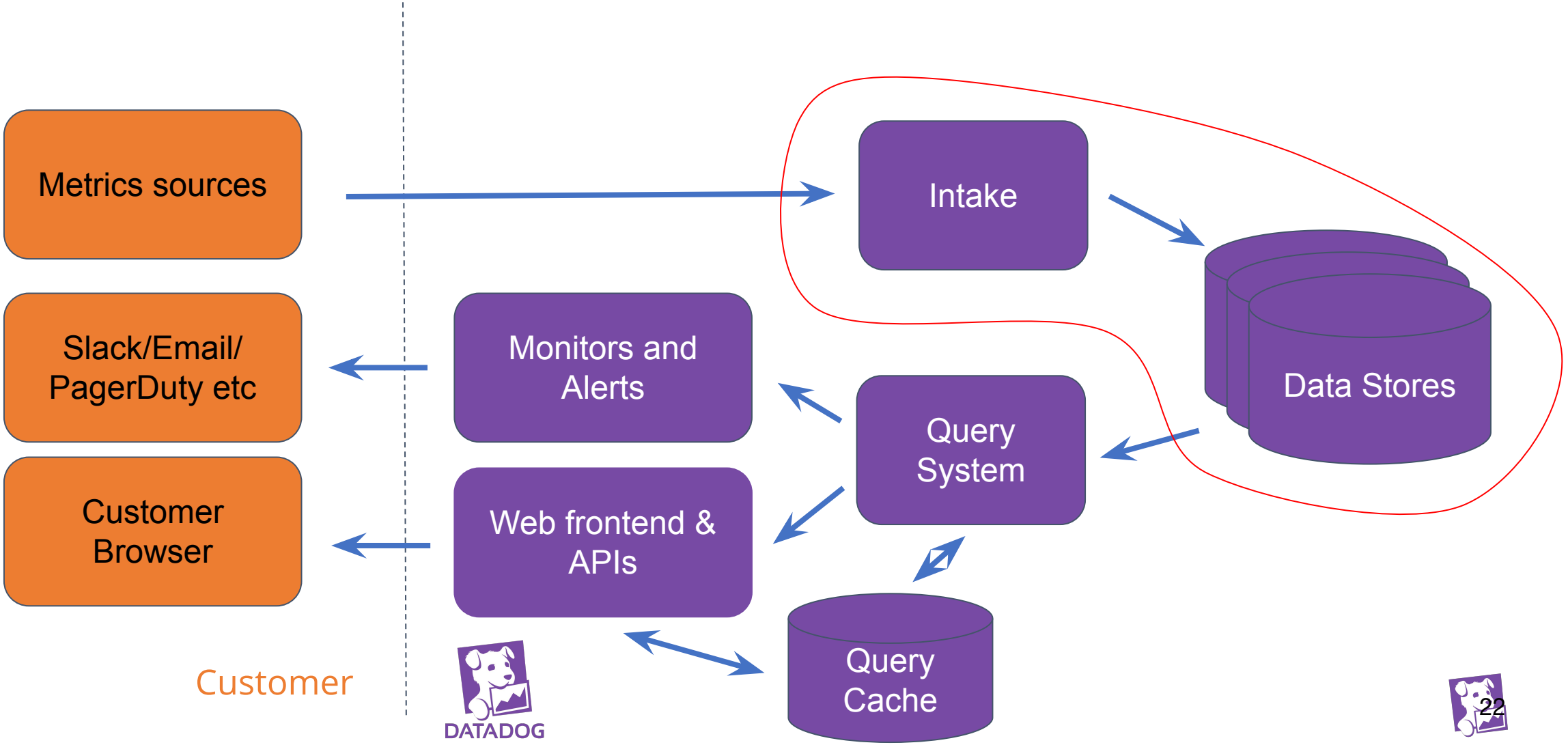


Metrics Store Characteristics

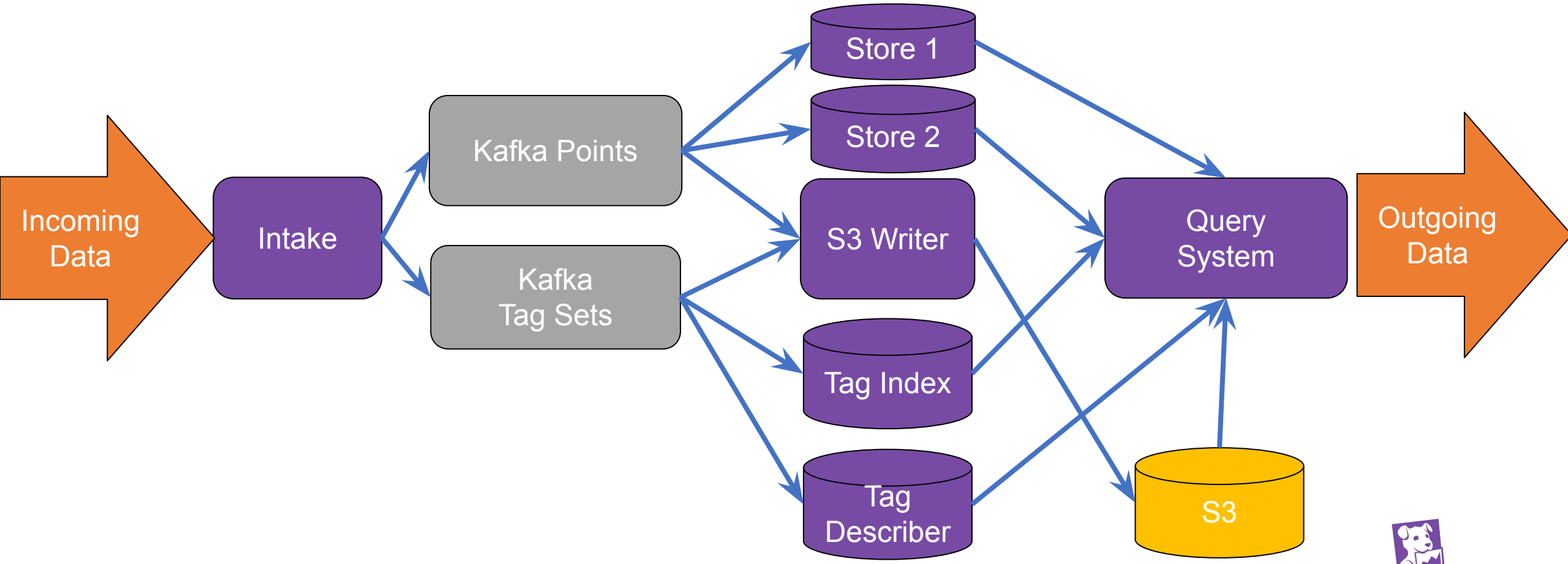
- Most metrics report with a tag set for quite some time

  => Therefore separate tag stores from time series stores

Pipeline Architecture



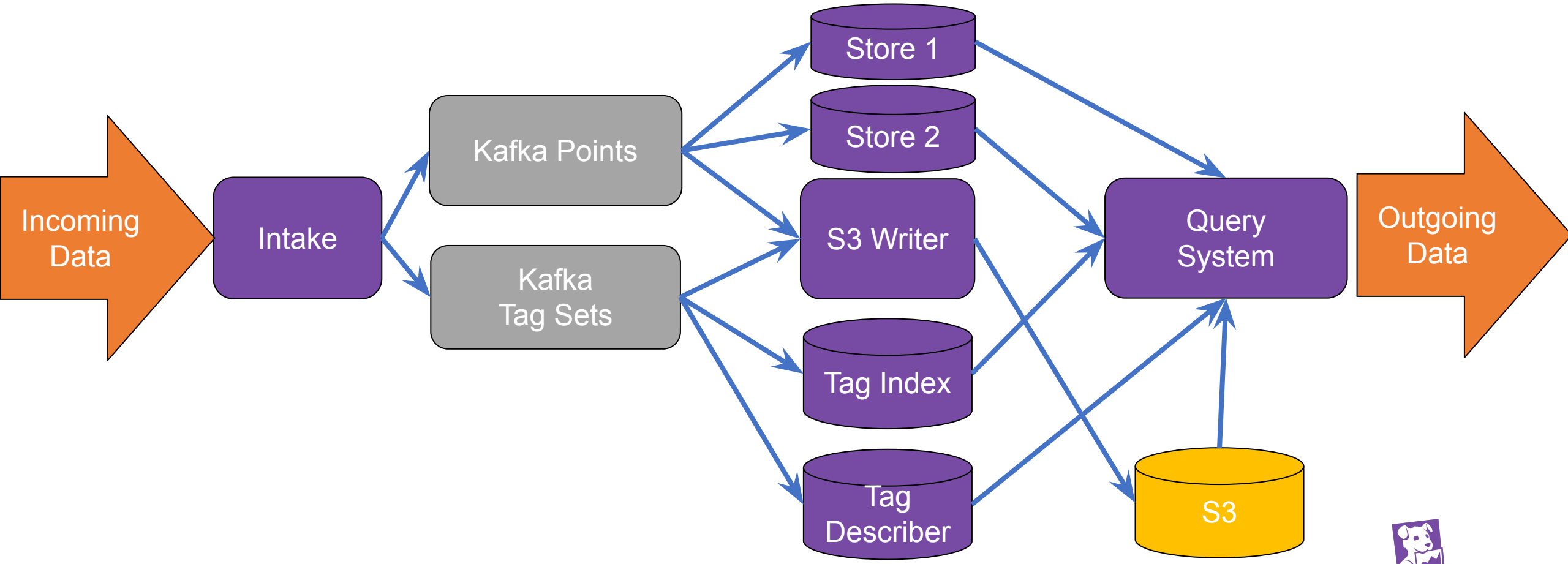
Kafka for Independent Storage Systems



Performance mantras

- Don't do it
- Do it, but don't do it again - query caching
- Do it less
- **Do it later - minimize processing on path to persistence**
- Do it when they're not looking
- Do it concurrently
- Do it cheaper

Kafka for Independent Storage Systems

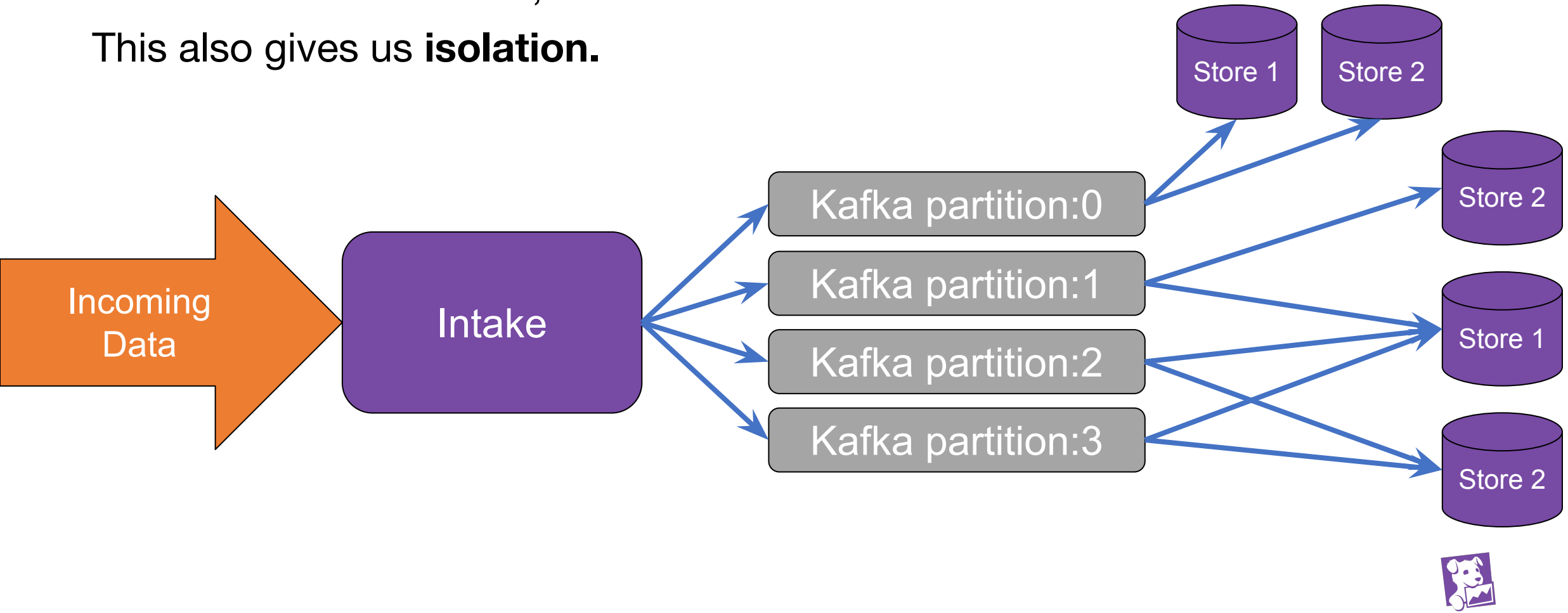


Scaling through Kafka

Data is separated by **partition** to distribute it

Partitions are customers, or a mod hash of their metric name

This also gives us **isolation.**



Performance mantras

- Don't do it
- Do it, but don't do it again - query caching
- Do it less
- Do it later - minimize processing on path to persistence
- Do it when they're not looking
- **Do it concurrently - use independent horizontally scalable data stores**
- Do it cheaper

Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. **Deep Dive On Our Datastores**
4. Handling Synchronization
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture

Per Customer Volume Ballparking

10^4	Number of apps; 1,000's hosts times 10's containers
10^3	Number of metrics emitted from each app/container
10^0	1 point a second per metric
10^5	Seconds in a day (actually 86,400)
10^1	Bytes/point (8 byte float, amortized tags)
$= 10^{13}$	10 Terabytes a Day For One Average Customer

Volume Math

- $210 to store 10 TB in S3 for a month
- $60,000 for a month rolling queryable (300 TB)
- But S3 is not for real time, high throughput queries

Cloud Storage Characteristics

Type	Max Capacity	Bandwidth	Latency	Cost/TB for 1 month	Volatility
DRAM[1]	4 TB	80 GB/s	0.08 us	$1,000	Instance Reboot
SSD[2]	60 TB	12 GB/s	1 us	$60	Instance Failures
EBS io1	432 TB	12 GB/s	40 us	$400	Data Center Failures
S3	Infinite	12 GB/s[3]	100+ ms	$21[4]	11 nines durability
Glacier	Infinite	12 GB/s[3]	hours	$4[4]	11 nines durability

1. X1e.32xlarge, 3 year non convertible, no upfront reserved instance
2. i3en.24xlarge, 3 year non convertible, no upfront reserved instance
3. Assumes can highly parallelize to load network card of 100Gbps instance type. Likely does not scale out.
4. Storage Cost only

Volume Math

- 80 x1e.32xlarge DRAM for a month
- $300,000 to store for a month
- This is with no indexes or overhead
- And people want to query much more than a month.

Performance mantras

- Don't do it
- Do it, but don't do it again - query caching
- **Do it less - only index what you need**
- Do it later - minimize processing on path to persistence
- Do it when they're not looking
- Do it concurrently - use independent horizontally scalable data stores
- Do it cheaper

Returning to an Example Query

“Alert when the system load, averaged across our fleet in us-east-1a for a 5 minute interval, goes above 90%”

Queries We Need to Support

DESCRIBE TAGS	What tags are queryable for this metric?
TAG INDEX	Given a time series id, what tags were used?
TAG INVERTED INDEX	Given some tags and a time range, what were the time series ingested?
POINT STORE	What are the values of a time series between two times?

Performance mantras

- Don't do it
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking
- Do it concurrently - use independent horizontally scalable data stores
- **Do it cheaper**

Performance mantras

- Don't do it
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking
- Do it concurrently - use independent horizontally scalable data stores
- **Do it cheaper - use hybrid data storage types and technologies**

Cloud Storage Characteristics

Type	Max Capacity	Bandwidth	Latency	Cost/TB for 1 month	Volatility
DRAM[1]	4 TB	80 GB/s	0.08 us	$1,000	Instance Reboot
SSD[2]	60 TB	12 GB/s	1 us	$60	Instance Failures
EBS io1	432 TB	12 GB/s	40 us	$400	Data Center Failures
S3	Infinite	12 GB/s[3]	100+ ms	$21[4]	11 nines durability
Glacier	Infinite	12 GB/s[3]	hours	$4[4]	11 nines durability

1. X1e.32xlarge, 3 year non convertible, no upfront reserved instance
2. i3en.24xlarge, 3 year non convertible, no upfront reserved instance
3. Assumes can highly parallelize to load network card of 100Gbps instance type. Likely does not scale out.
4. Storage Cost only

Cloud Storage Characteristics

Type	Max Capacity	Bandwidth	Latency	Cost/TB for 1 month	Volatility
DRAM[1]	4 TB	80 GB/s	0.08 us	$1,000	Instance Reboot
SSD[2]	60 TB	12 GB/s	1 us	$60	Instance Failures
EBS io1	432 TB	12 GB/s	40 us	$400	Data Center Failures
S3	Infinite	12 GB/s[3]	100+ ms	$21[4]	11 nines durability
Glacier	Infinite	12 GB/s[3]	hours	$4[4]	11 nines durability

1. X1e.32xlarge, 3 year non convertible, no upfront reserved instance
2. i3en.24xlarge, 3 year non convertible, no upfront reserved instance
3. Assumes can highly parallelize to load network card of 100Gbps instance type. Likely does not scale out.
4. Storage Cost only

Hybrid Data Storage Types

System
DESCRIBE TAGS
TAG INDEX
TAG INVERTED INDEX
POINT STORE
QUERY RESULTS

Hybrid Data Storage Types

System	Type	Persistence
DESCRIBE TAGS	Local SSD	Years
TAG INDEX	DRAM	Cache (Hours)
	Local SSD	Years
TAG INVERTED INDEX	DRAM	Hours
	On SSD	Days
	S3	Years
POINT STORE	DRAM	Hours
	Local SSD	Days
	S3	Years
QUERY RESULTS	DRAM	Cache (Days)

Hybrid Data Storage Technologies

System	Type	Persistence	Technology	Why?
DESCRIBE TAGS	Local SSD	Years	LevelDB	High performing single node k,v
TAG INDEX	DRAM	Cache (Hours)	Redis	Very high performance, in memory k,v
	Local SSD	Years	Cassandra	Horizontal scaling, persistent k,v
TAG INVERTED INDEX	DRAM	Hours	In house	Very customized index data structures
	On SSD	Days	RocksDB + SQLite	Rich and flexible queries
	S3	Years	Parquet	Flexible Schema over time
POINT STORE	DRAM	Hours	In house	Very customized index data structures
	Local SSD	Days	In house	Very customized index data structures
	S3	Years	Parquet	Flexible Schema over time
QUERY RESULTS	DRAM	Cache (Days)	Redis	Very high performance, in memory k,v

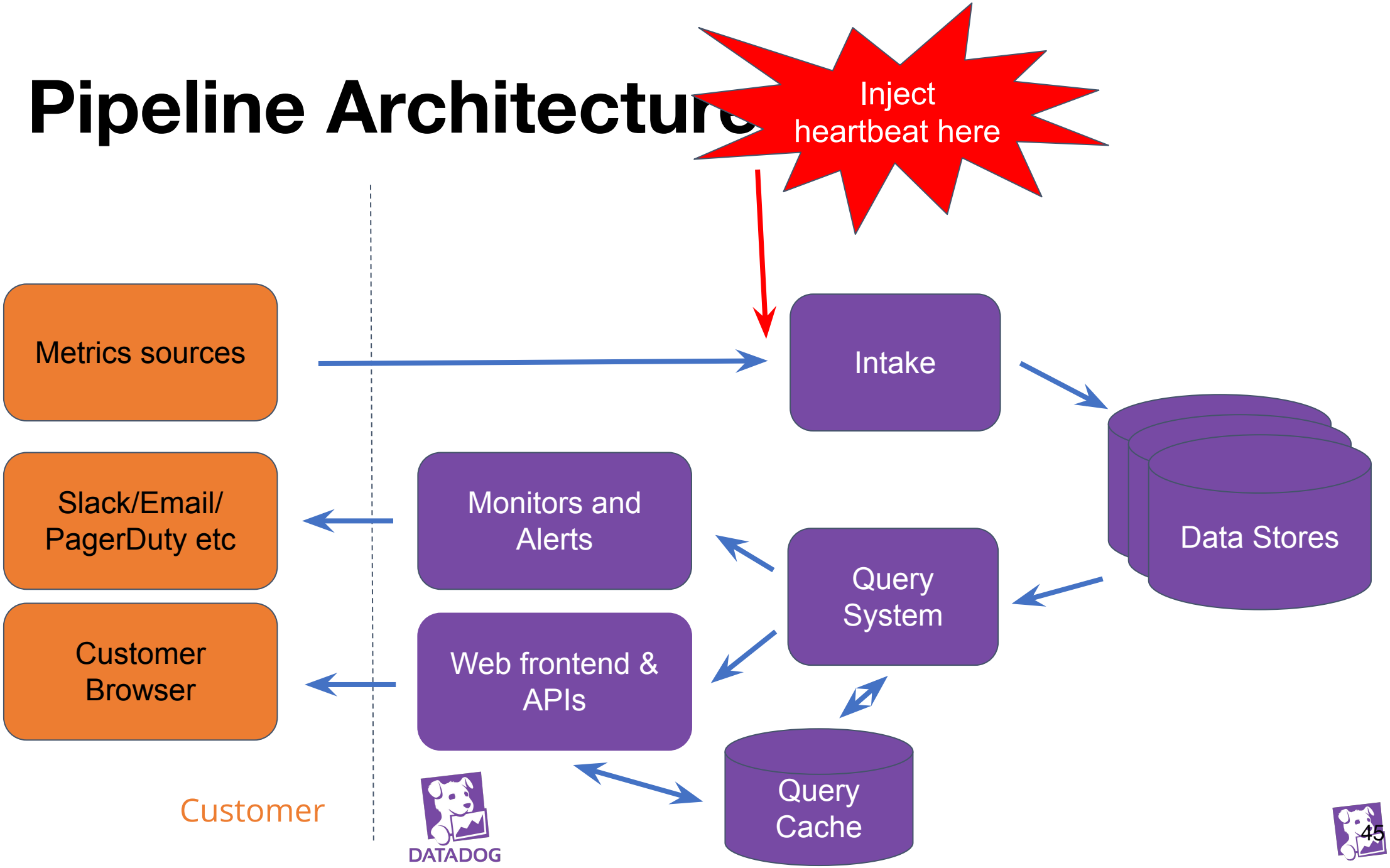
Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. **Handling Synchronization**
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture

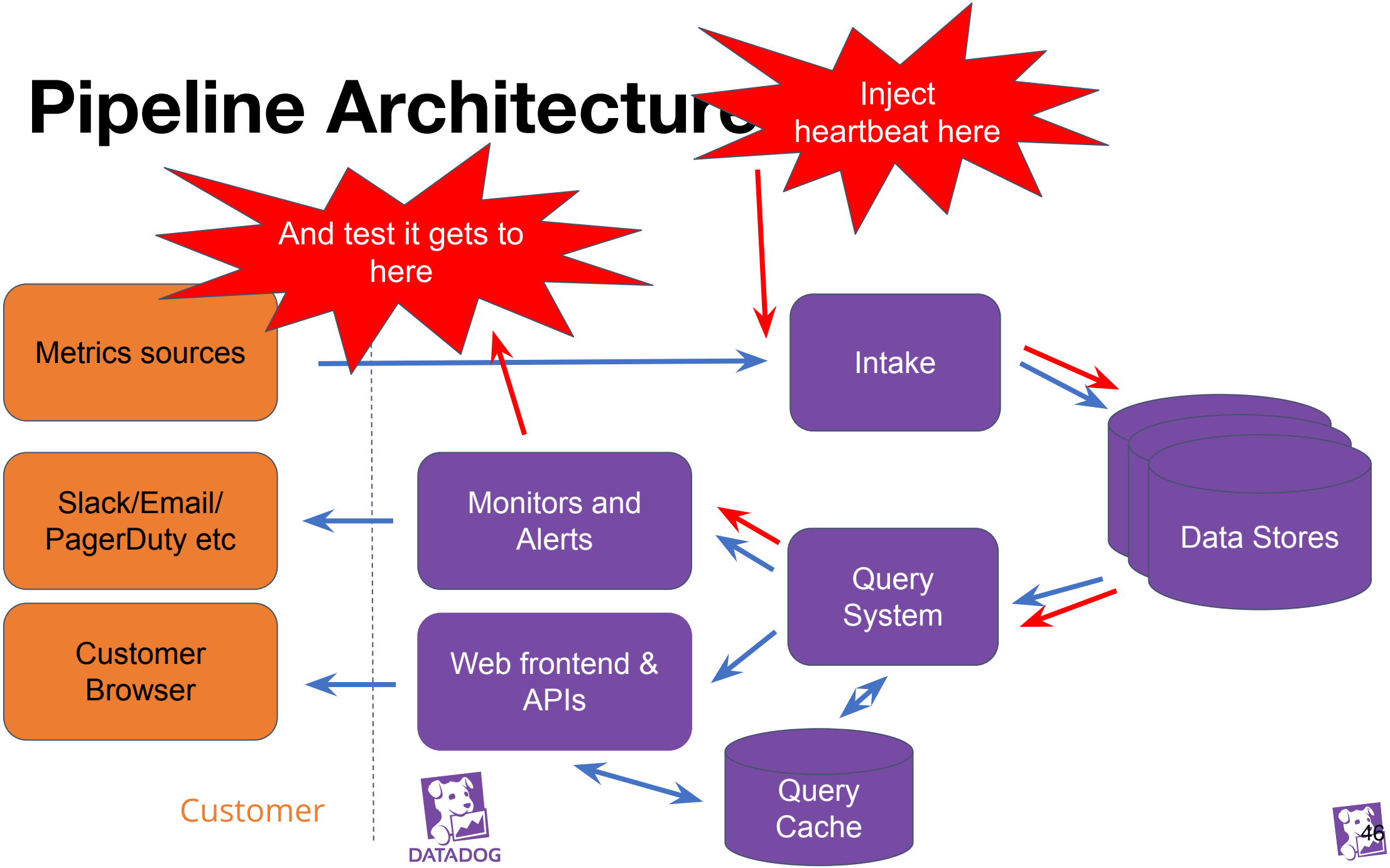
Alerts/Monitors Synchronization

- Level sensitive
  - False positives is almost as important as false negative
- Small delay preferable to evaluating incomplete data
- Synchronization need is to be sure evaluation bucket is filled before processing

Pipeline Architecture



Pipeline Architecture



Heartbeats for Synchronization

Semantics:

- 1 second tick time for metrics
- Last write wins to handle agent concurrency
- Inject fake data as heartbeat through pipeline

Then:

- Monitor evaluator ensure heartbeat gets through before evaluating next period

Challenges:

- With sharding and multiple stores, lots of independent paths to make sure heartbeats go through

Performance mantras

- **Don't do it - build the minimal synchronization needed**
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking
- Do it concurrently - use independent horizontally scalable data stores
- Do it cheaper - use hybrid data storage types and technologies

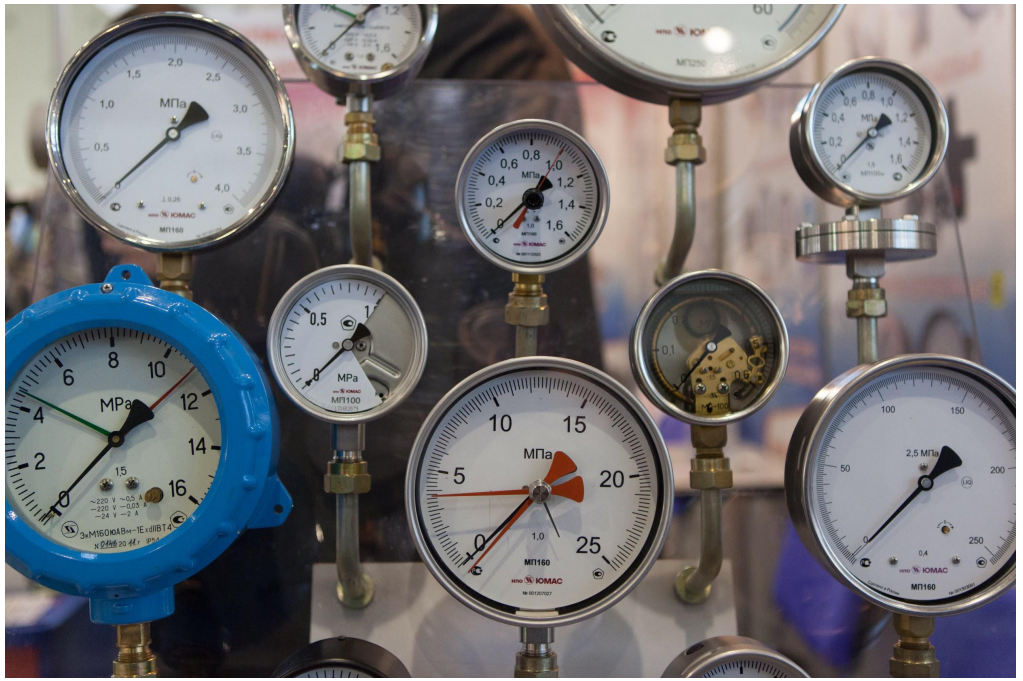
Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. **Introducing Aggregation**
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture

Types of metrics

Counter, aggregate by sum

Ex: Requests, errors/s, total time spent (stopwatch)

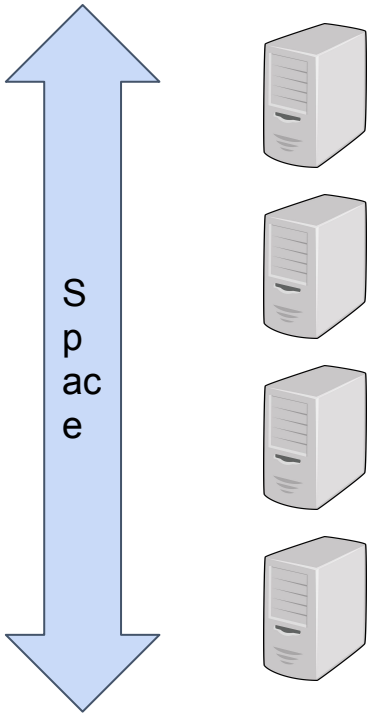


Gauges, aggregate by last or avg

Ex: CPU/network/disk use, queue length

Aggregation

t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9

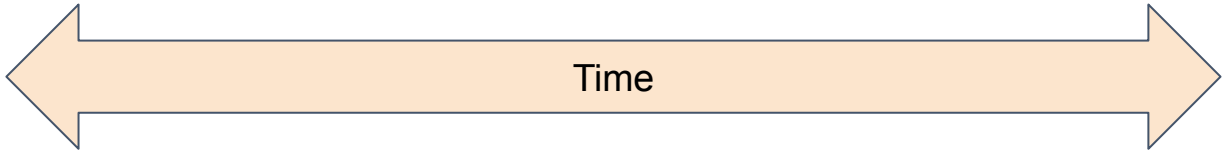


{0, 1, 0, 1, 0, 1, 0, 1, 0, 1}

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

{5, 5, 5, 5, 5, 5, 5, 5, 5, 5}

{0, 2, 4, 8, 16, 32, 64, 128, 256, 512}



Query output

Counters: {5, 40, 50, 1023}

Gauges (average): {0.5, 4, 5, 102.3}

Gauges (last): {1, 9, 5, 512}

Query characteristics

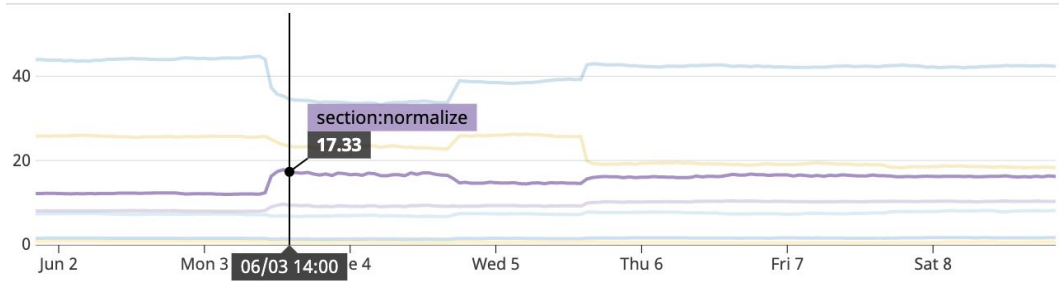
User:

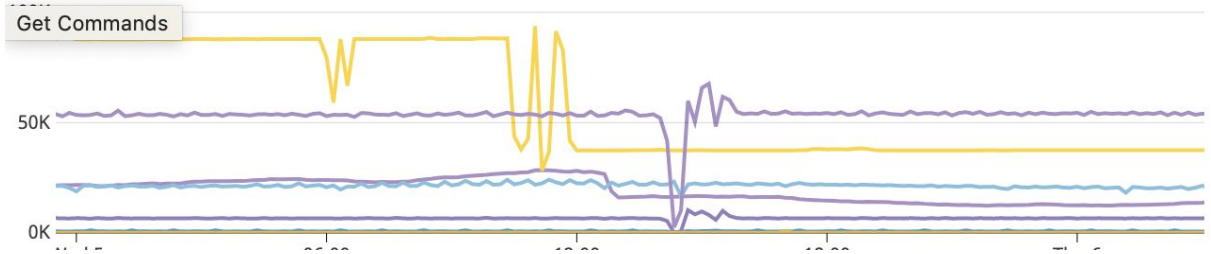
- Bursty and unpredictable
- Latency Sensitive - ideal end user response is 100ms, 1s at most.
- Skews to recent data, but want same latency on old data

Query characteristics

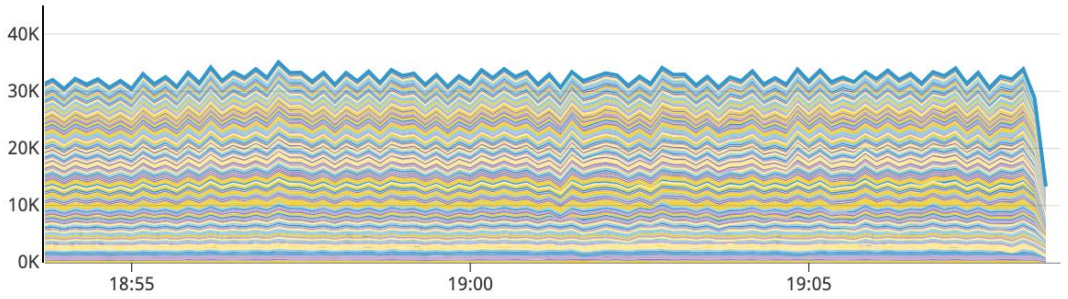
Dashboards:

- Predictable
- Important enough to save
- Looking for step-function changes, e.g. performance regressions, changes in usage

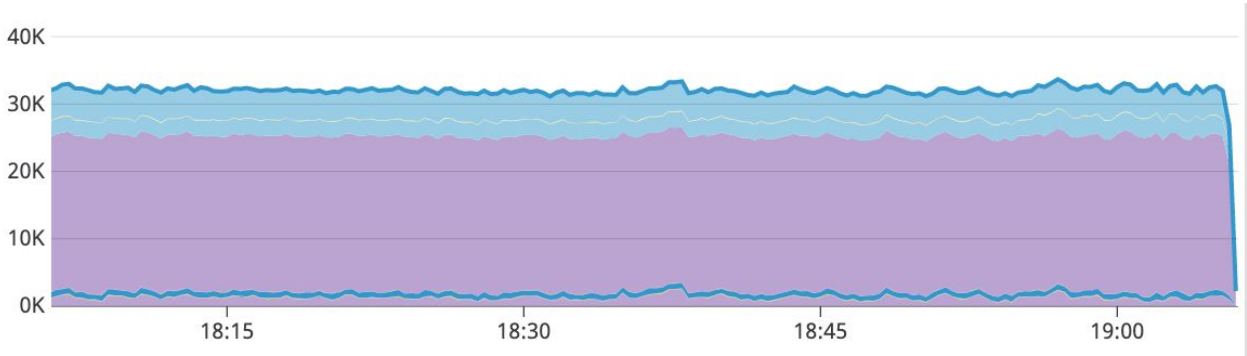




Focus on outputs



Showing 3254 series from 1 queries.



Showing 33 series from 1 queries.

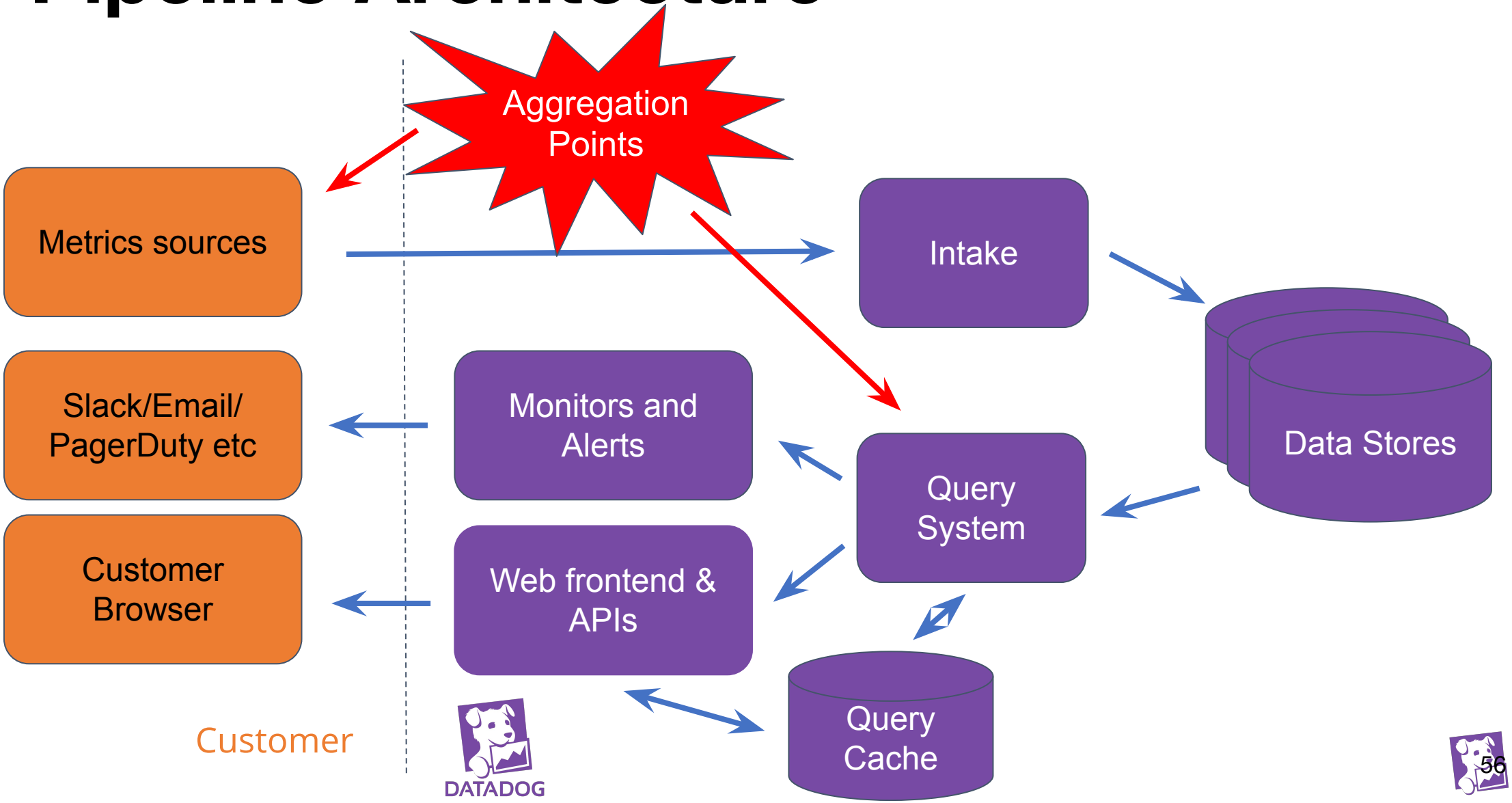
These graphs are *both* aggregating 70k series

Not a lot, but still output 10x to 2000x less than input!

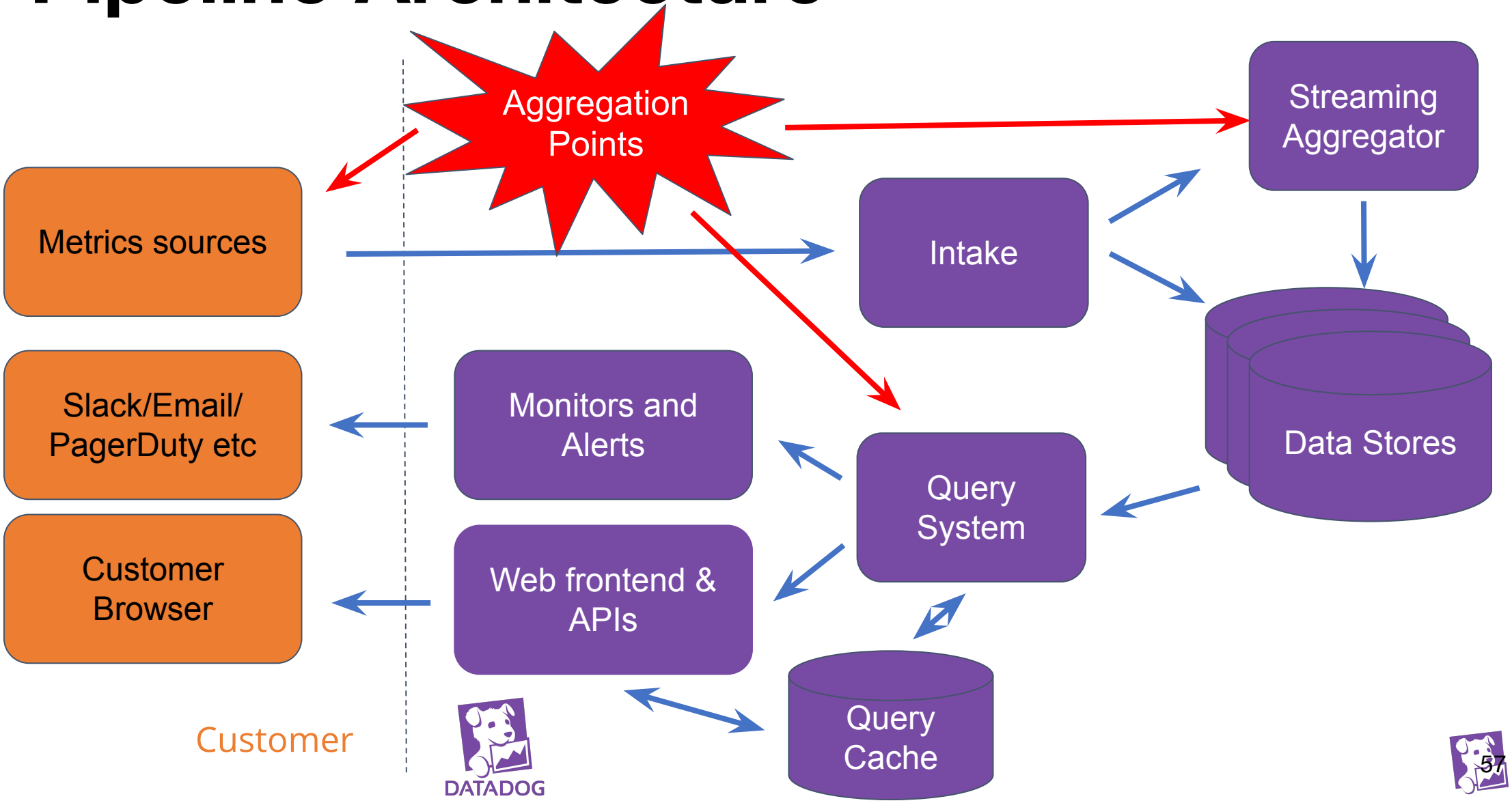
Performance mantras

- Don't do it - build the minimal synchronization needed
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- **Do it when they're not looking?**
- Do it concurrently - use independent horizontally scalable data stores
- Do it cheaper - use hybrid data storage types and technologies

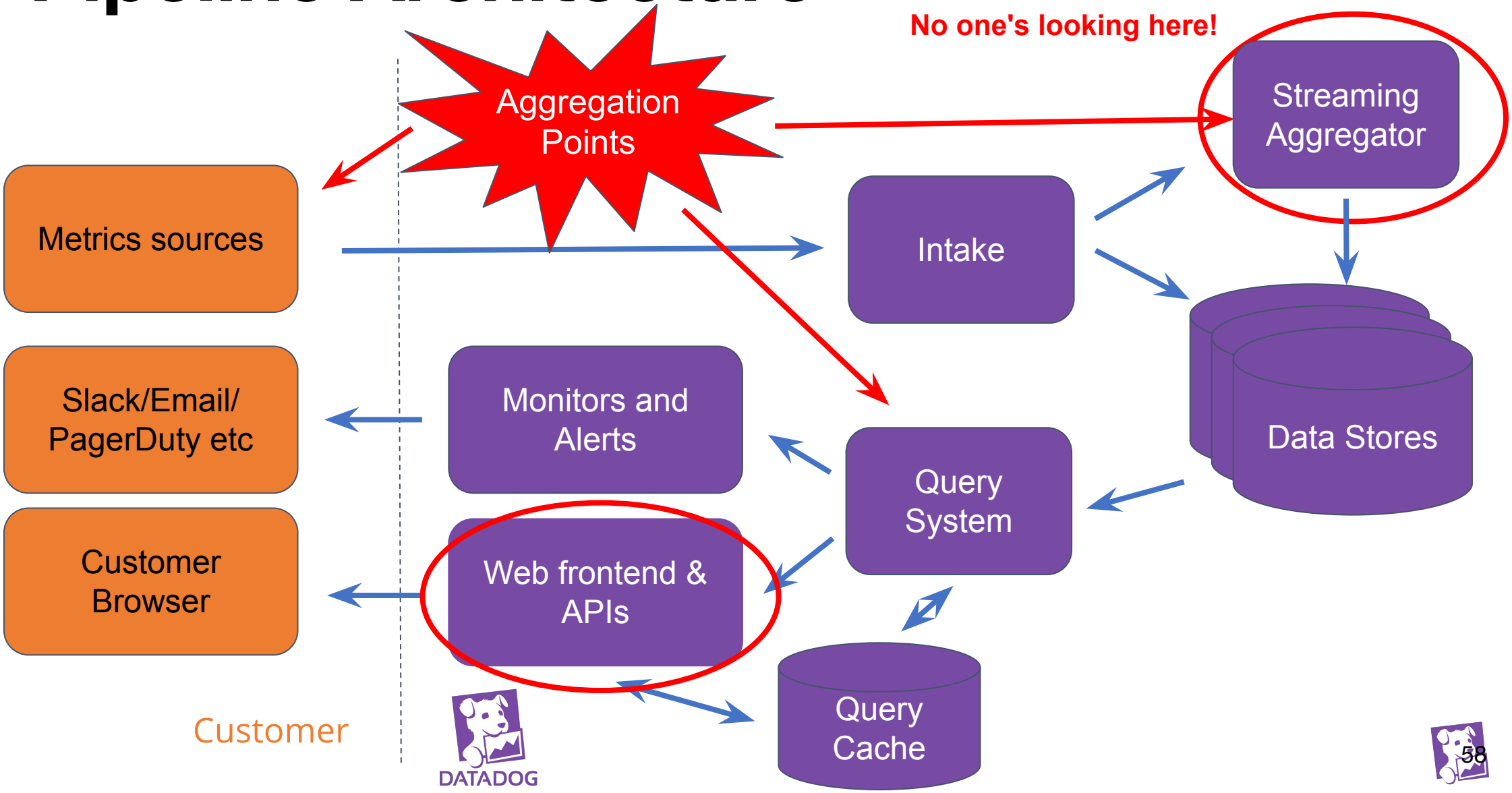
Pipeline Architecture



Pipeline Architecture



Pipeline Architecture



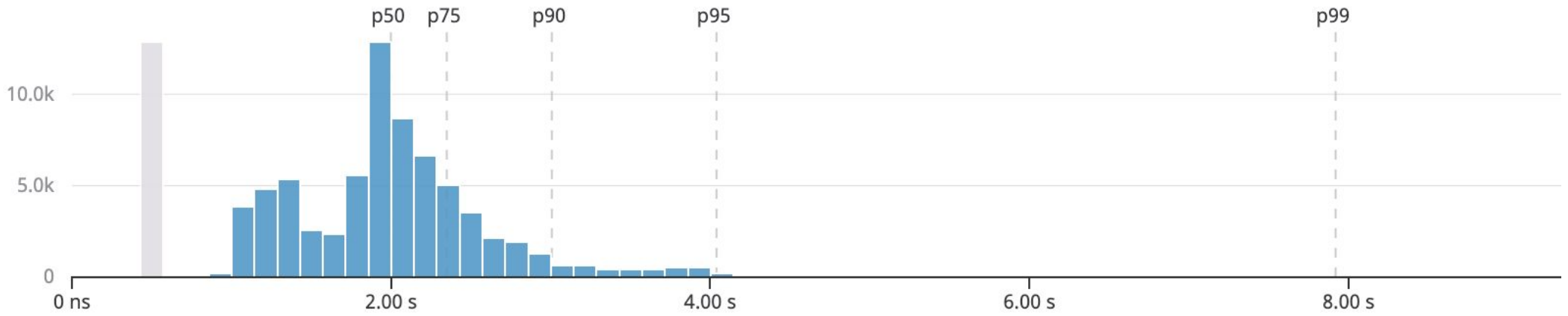
Performance mantras

- Don't do it - build the minimal synchronization needed
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- **Do it when they're not looking - pre-aggregate**
- Do it concurrently - use independent horizontally scalable data stores
- Do it cheaper - use hybrid data storage types and technologies

Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. Introducing Aggregation
6. **Aggregation For Deeper Insights Using Sketches**
7. Sketches Enabling Flexible Architecture

Distributions

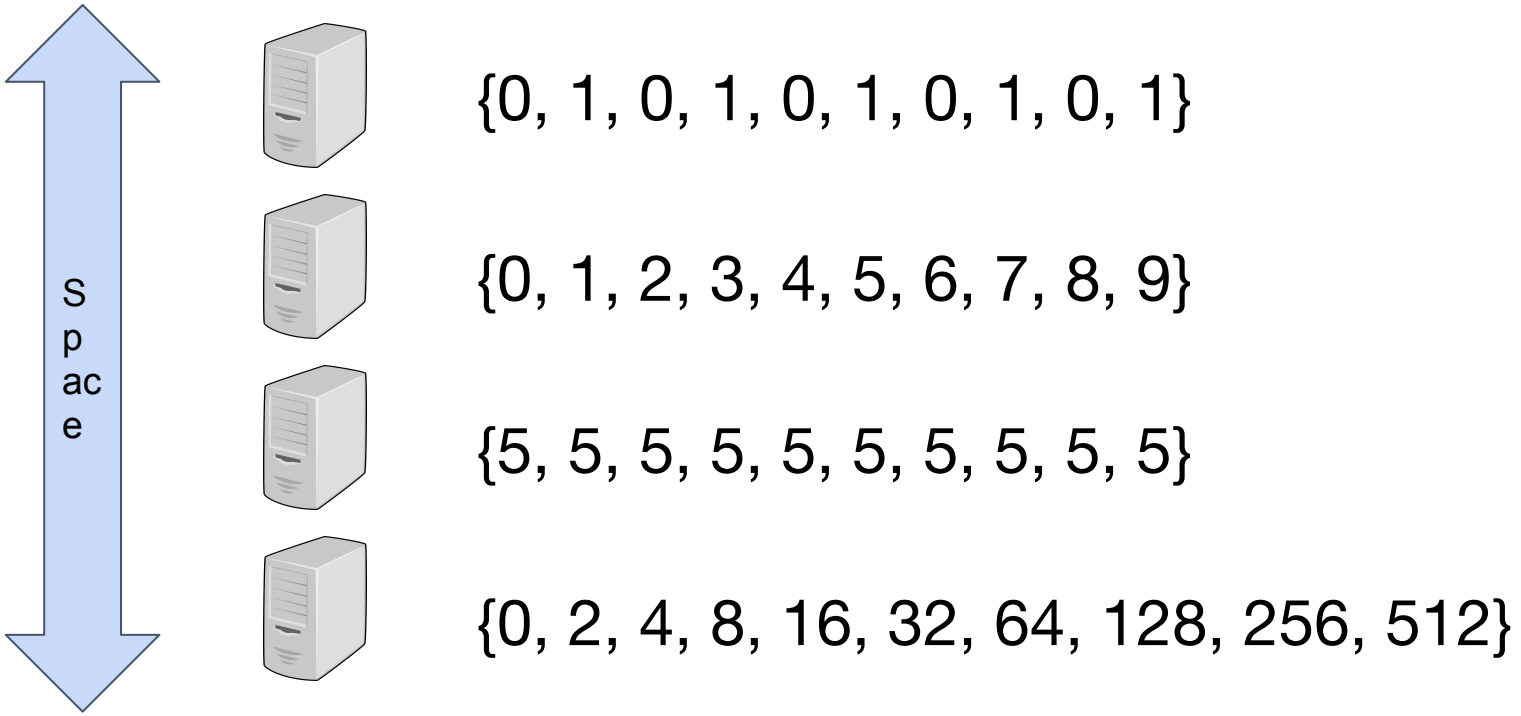


Aggregate by percentile or SLO
(count of values above or below a threshold)

Ex: Latency, request size

Calculating distributions

t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9



p50

{0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 3, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 7, 8, 8, 9, 16, 32, 64, 128, 256, 512}

p90

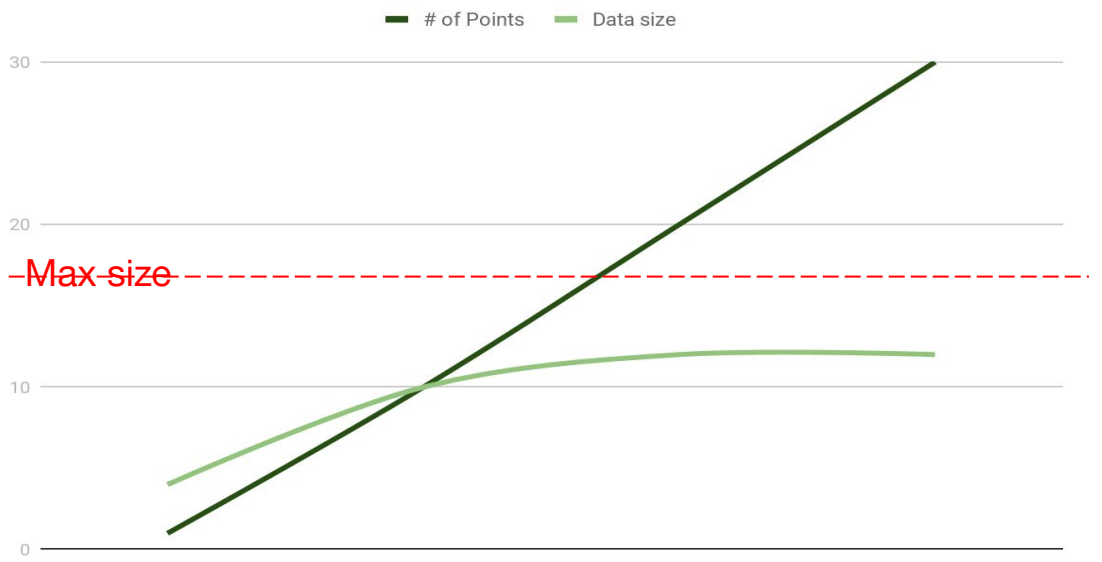
Performance mantras

- Don't do it - build the minimal synchronization needed
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking - pre-aggregate
- Do it concurrently - use independent horizontally scalable data stores
- **Do it cheaper again?**

What are "sketches"?

Data structures designed for operating on streams of data

- Examine each item a limited number of times (ideally once)
- Limited memory usage (logarithmic to the size of the stream, or fixed)



Examples of sketches

HyperLogLog

- Cardinality / unique count estimation
- Used in Redis `PFADD`, `PFCOUNT`, `PFMERGE`

Others: Bloom filters (also for set membership), frequency sketches (top-N lists)

Tradeoffs

Understand the tradeoffs - speed, accuracy, space

What other characteristics do you need?

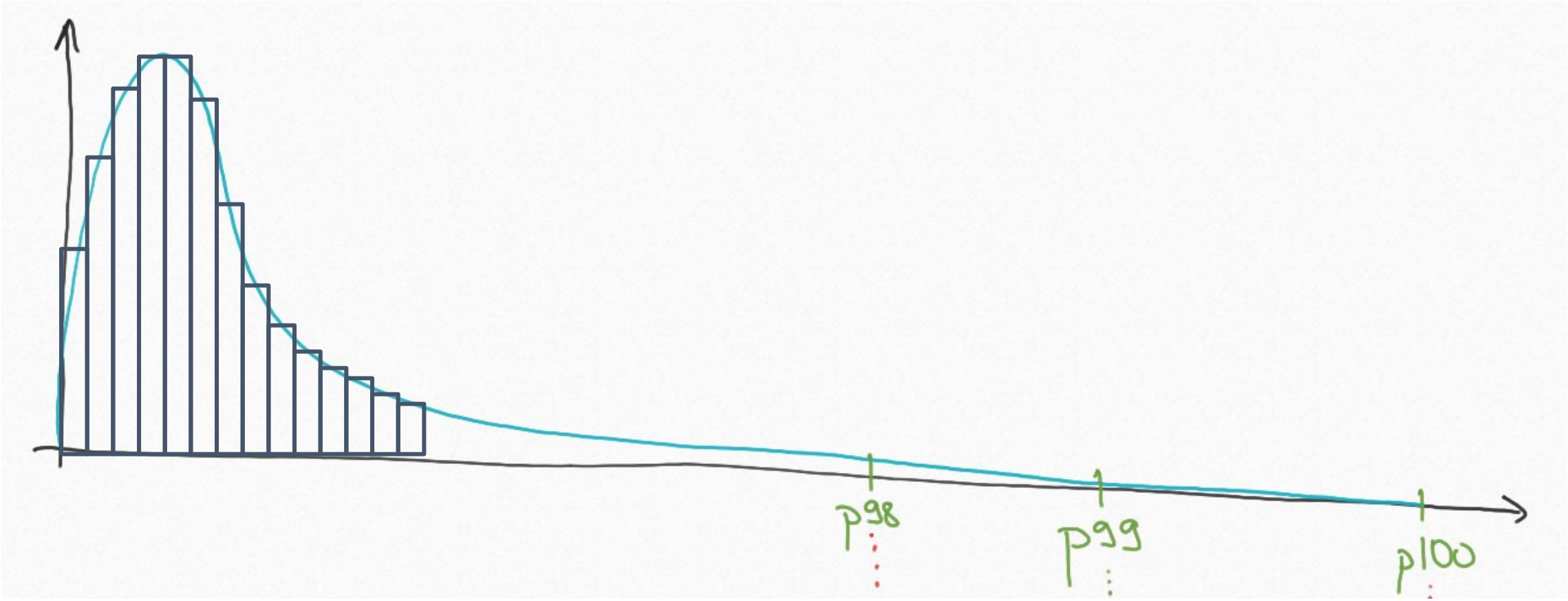
- Well-defined or arbitrary range of inputs?
- What kinds of queries are you answering?

Approximation for distribution metrics

What's important for approximating distribution metrics?

- Bounded error
- Performance - size, speed of inserts
- Aggregation (aka "merging")

How do you compress a distribution



Histograms

Basic example from OpenMetrics / Prometheus

```
# HELP http_request_duration_seconds A histogram of the request duration.
# TYPE http_request_duration_seconds histogram
http_request_duration_seconds_bucket{le="0.05"} 24054
http_request_duration_seconds_bucket{le="0.1"} 33444
http_request_duration_seconds_bucket{le="0.2"} 100392
http_request_duration_seconds_bucket{le="0.5"} 129389
http_request_duration_seconds_bucket{le="1"} 133988
http_request_duration_seconds_bucket{le="+Inf"} 144320
http_request_duration_seconds_sum 53423
http_request_duration_seconds_count 144320
```

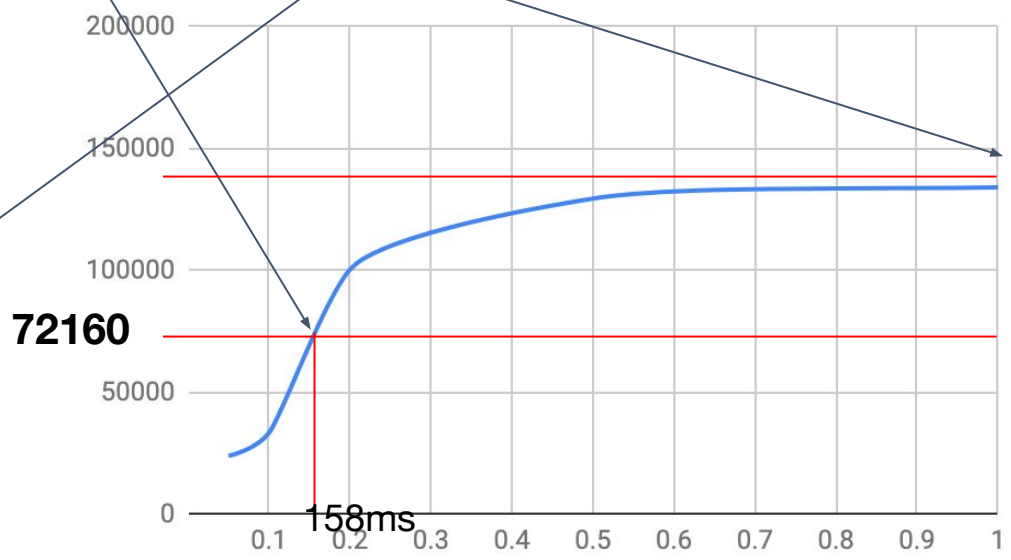
Histograms

Basic example from OpenMetrics / Prometheus

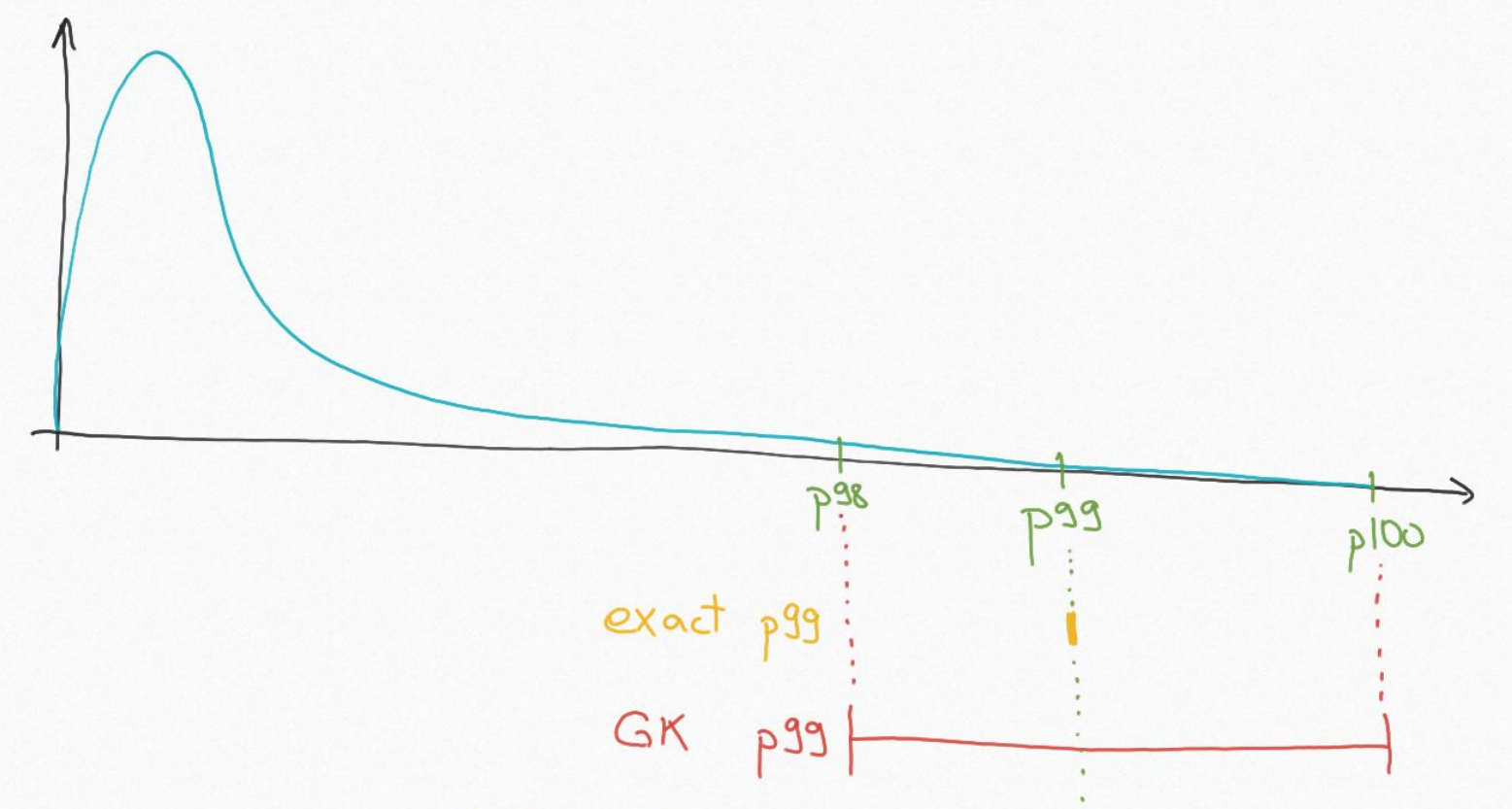
Time spent	Count
<= 0.05 (50ms)	24054
<= 0.1 (100ms)	33444
<= 0.2 (200ms)	100392
<= 0.5 (500ms)	129389
<= 1s	133988
> 1s	144320

median = ~**158ms** (using linear interpolation)

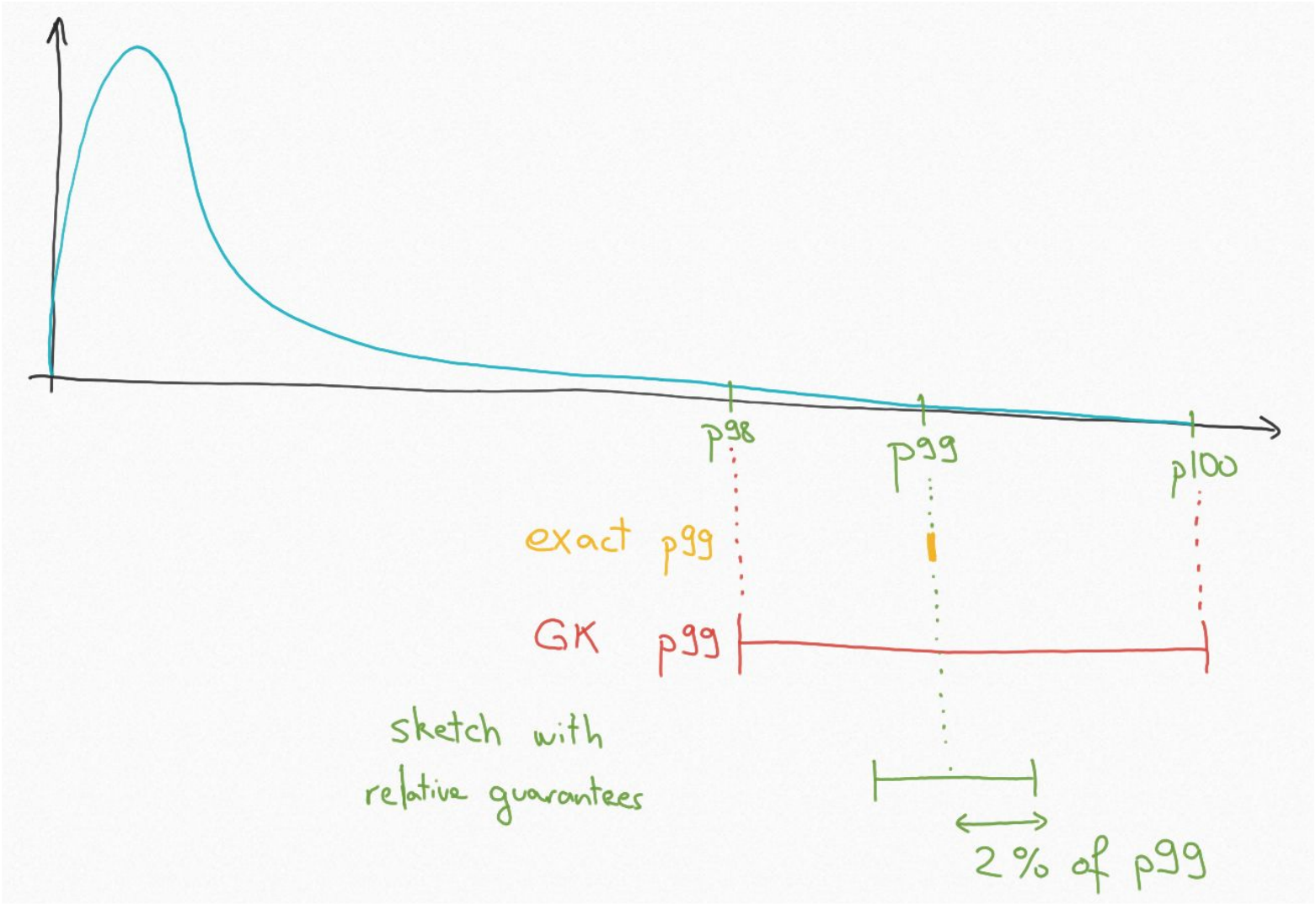
p99 = **?!**



Rank and relative error



Rank and relative error



Relative error

In metrics, specifically latency metrics, we care about about both the distribution of data as well as **specific values**

E.g., for an SLO, I want to know, is my p99 500ms or less?

Relative error bounds mean we can answer this: Yes, within 99% of requests are <= 500ms +/- 1%

Otherwise stated: 99% of requests are **guaranteed** <= 505ms

Fast insertion

Each insertion is just two operations - find the bucket, increase the count (sometimes there's an allocation)

Fixed Size - how?

With certain distributions, we may reach the maximum number of buckets (in our case, 4000)

- Roll up lower buckets - lower percentiles are generally not as interesting!*

**Note that we've yet to find a data set that actually needs this in practice*

Aggregation and merging

"a binary operation is **commutative** if *changing the order* of the operands does not change the result"

Why is this important?

Talk Plan

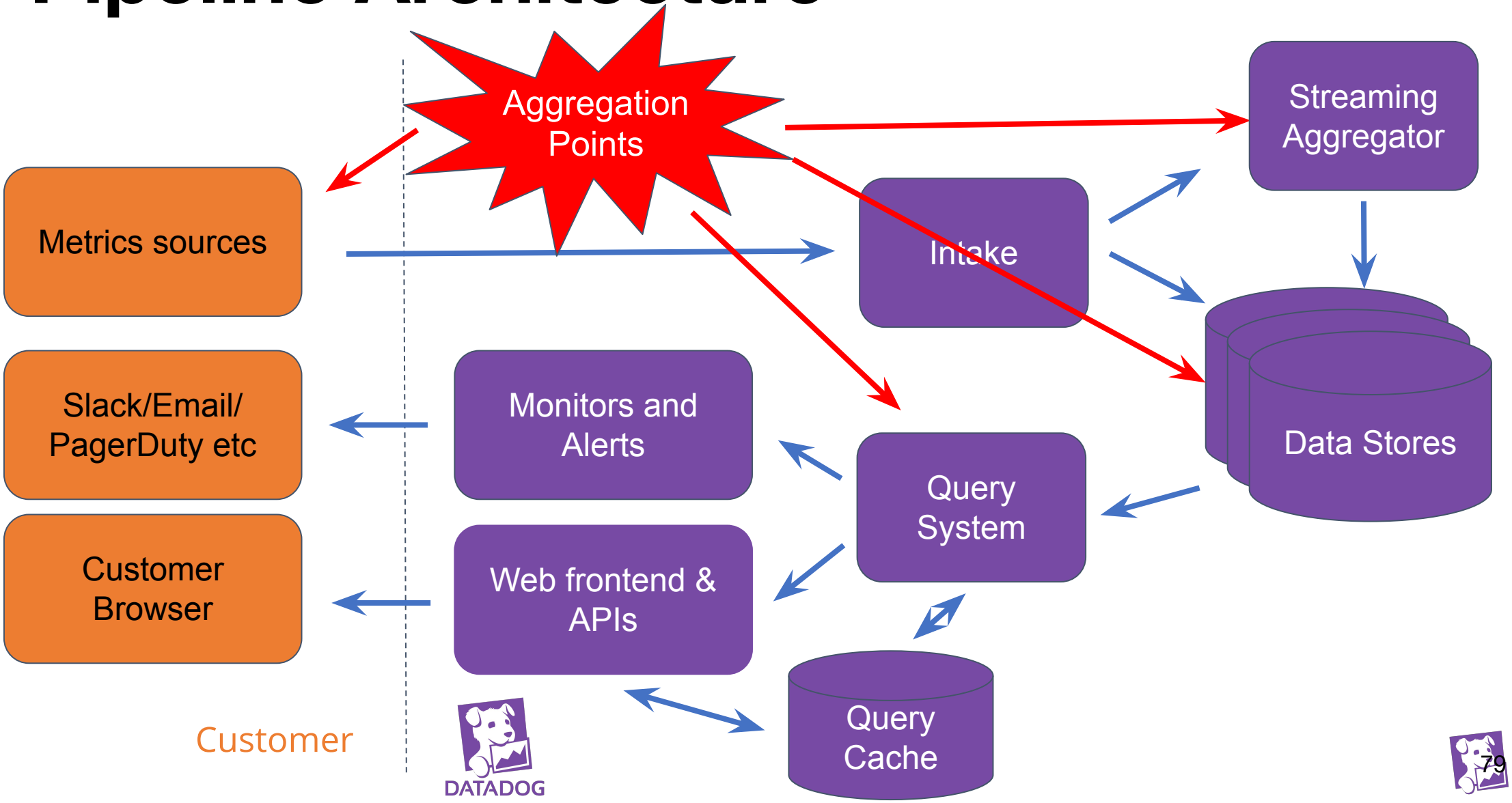
1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. **Sketches Enabling Flexible Architecture**

Before, during, save for later

If we have two-way mergeable sketches, we can re-aggregate the aggregations

- Agent
- Streaming during ingestion
- At query time
- In the data store (saving partial results)

Pipeline Architecture



DDSketch

DDSketch (Distributed Distribution Sketch) is open source (part of the agent today)

- Presenting at VLDB2019 in August
- Open-sourcing standalone versions in several languages

Performance mantras

- Don't do it - build the minimal synchronization needed
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking - pre-aggregate
- Do it concurrently - use independent horizontally scalable data stores
- **Do it cheaper - use hybrid data storage types and technologies**

Performance mantras

- Don't do it - build the minimal synchronization needed
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking - pre-aggregate
- Do it concurrently - use independent horizontally scalable data stores
- **Do it cheaper - use hybrid data storage types and technologies, and use compression techniques based on what customers really need**

Summary

- **Don't do it -** build the bare minimal synchronization needed
- **Do it, but don't do it again -** use query caching
- **Do it less** - only index what you need
- **Do it later** - minimize processing on path to persistence
- **Do it when they're not looking -** pre-aggregate where is cost effective
- **Do it concurrently -** use independent horizontally scaleable data stores
- **Do it cheaper -** use hybrid data storage types and technologies, and use compression techniques based on what customers really need

Thank You

Challenges and opportunities of aggregation

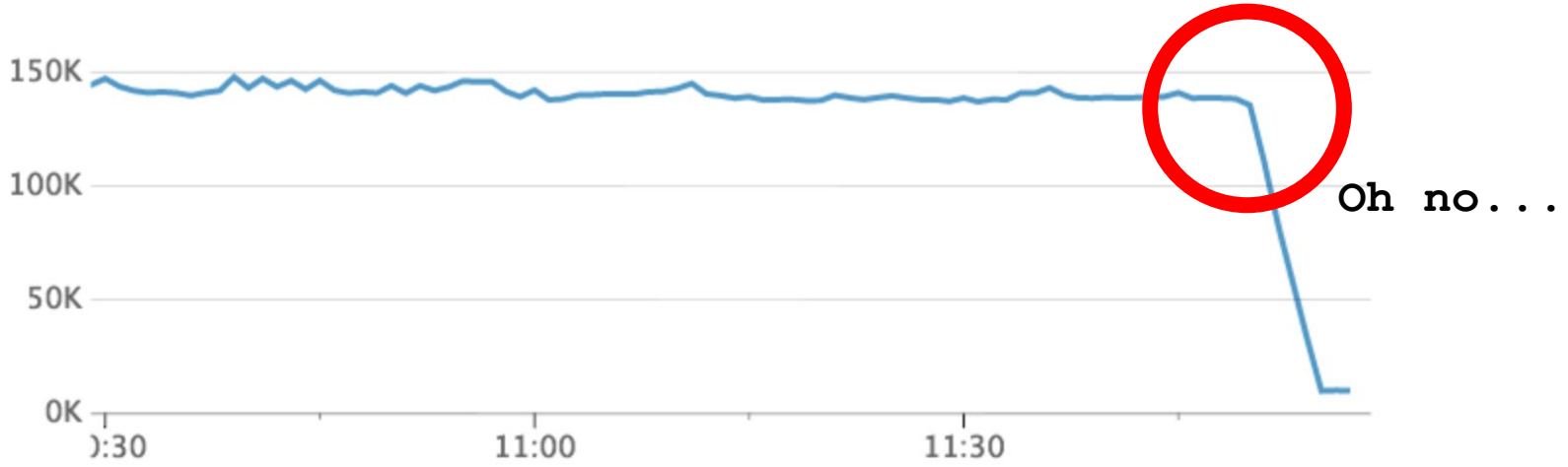
- Challenges:
  - Accuracy
  - Latency
- Opportunity:
  - Orders of magnitude performance improvement on common and highly visible queries

Human factors and dashboards

- Human-latency sensitive - high visibility

  Late-arriving data makes people nervous

- Human granularity - how many lines can you reason about on a dashboard?



Where aggregation happens

At the metric source (agent/lambda/etc)

- Counts by sum
- Gauges by last

At query time

- Arbitrary user selection (avg/sum/min/max)
- Impacts user experience

Adding a new metric type

Counters, gauges, **distributions**!

Used gauges for latency, etc, but aggregate by last is not what you want

Need to update the agent, libraries, integrations

We're learning and building on what we have today

Building blocks

We have a way to move data around (Kafka)

We have ways to index that data (tagsets)

We know **how** to separate recent and historical data

Plan for the **future**

[Lego / puzzle with gaps]

Connect the dots