



DATADOG

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

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

SACON '20

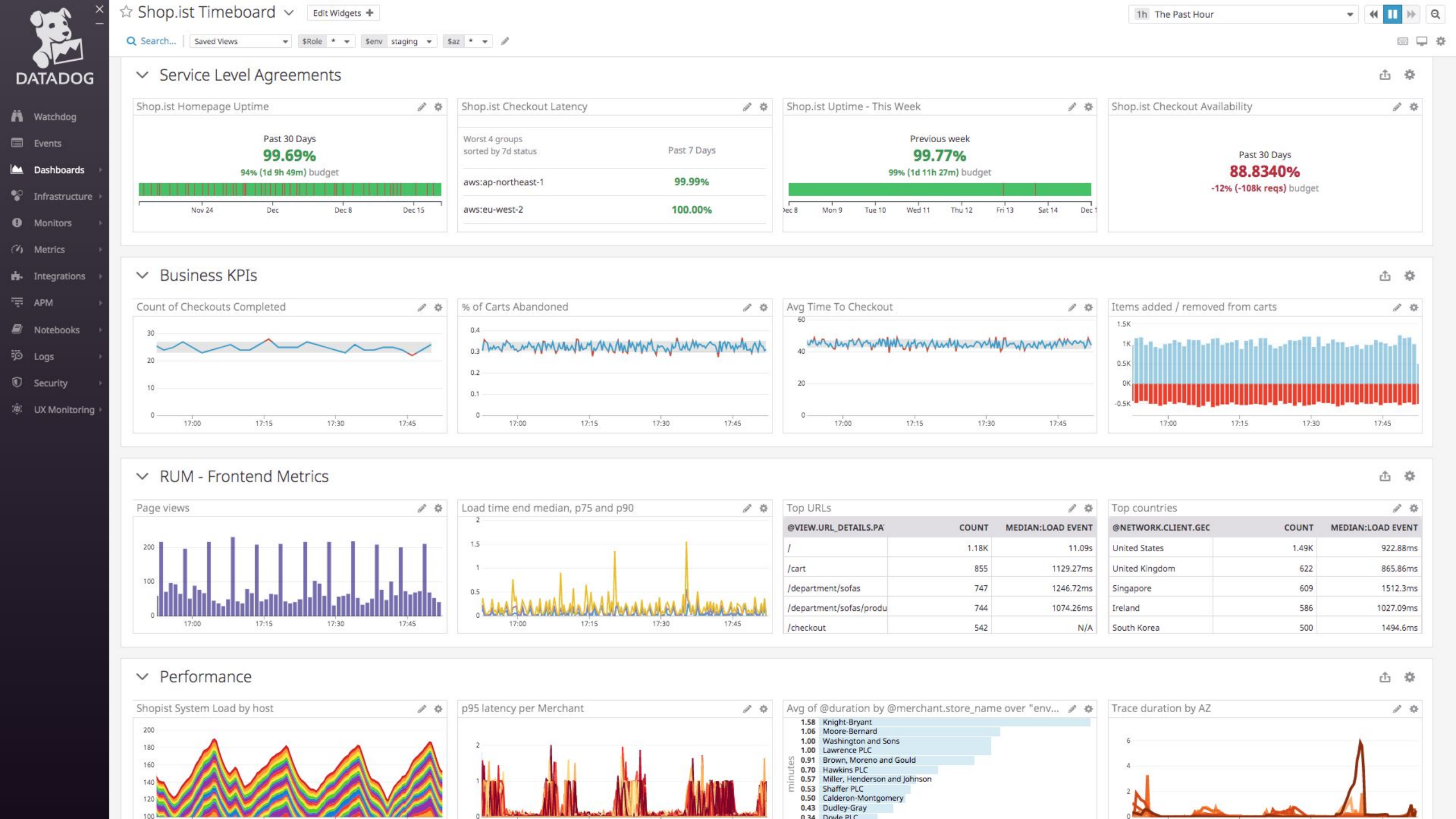


Trillions of points per day

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)

$$10^4 \times 10^3 \times 10^5 = 10^{12}$$





Decreasing Infrastructure Lifecycle

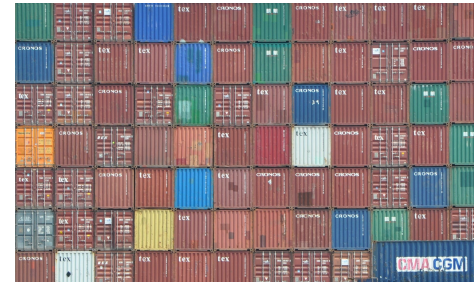
Datacenter



Cloud/VM



Containers



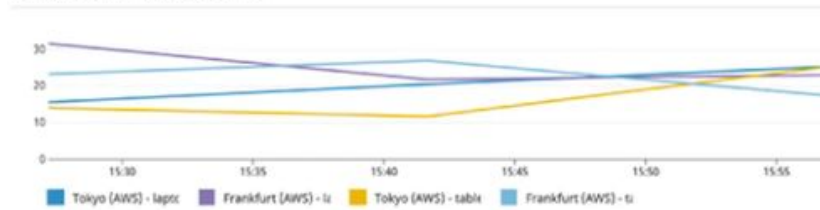
Months/years

Seconds



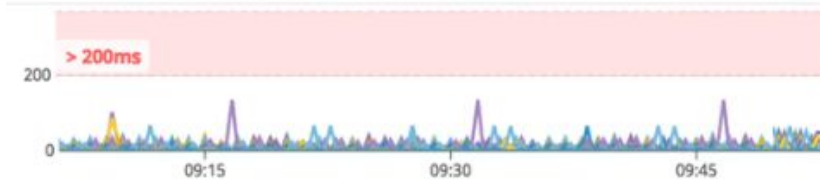
Increasing Granularity

Test duration by location & device



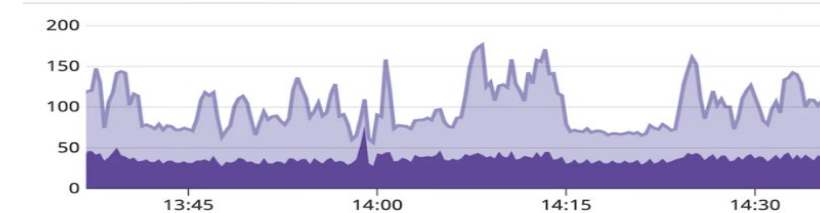
Per User Device

[SLI] Batch processing latency



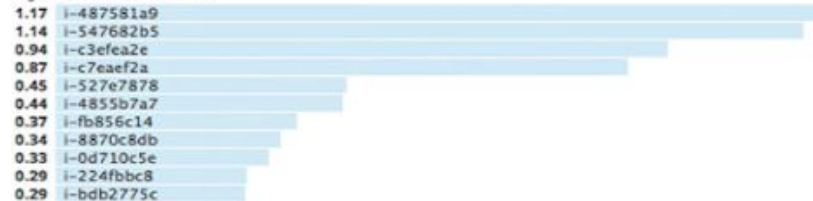
SLIs

Write requests (per second)

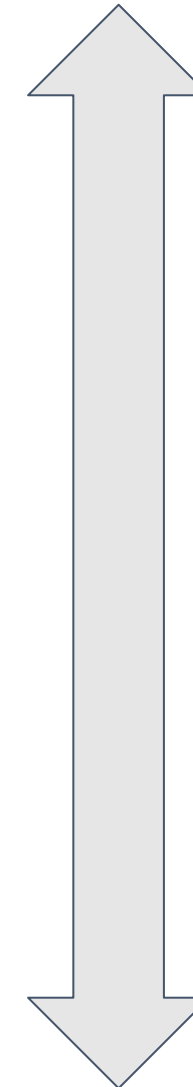


Application

system.load.1 - 1h



System



10,000's

100's



Tackling performance challenges

- 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. Our Architecture
2. Deep Dive On Our Datastores
3. Handling Synchronization
4. Approximation For Deeper Insights
5. Enabling Flexible Architecture



Talk Plan

1. **Our Architecture**
2. Deep Dive On Our Datastores
3. Handling Synchronization
4. Approximation For Deeper Insights
5. Enabling Flexible Architecture

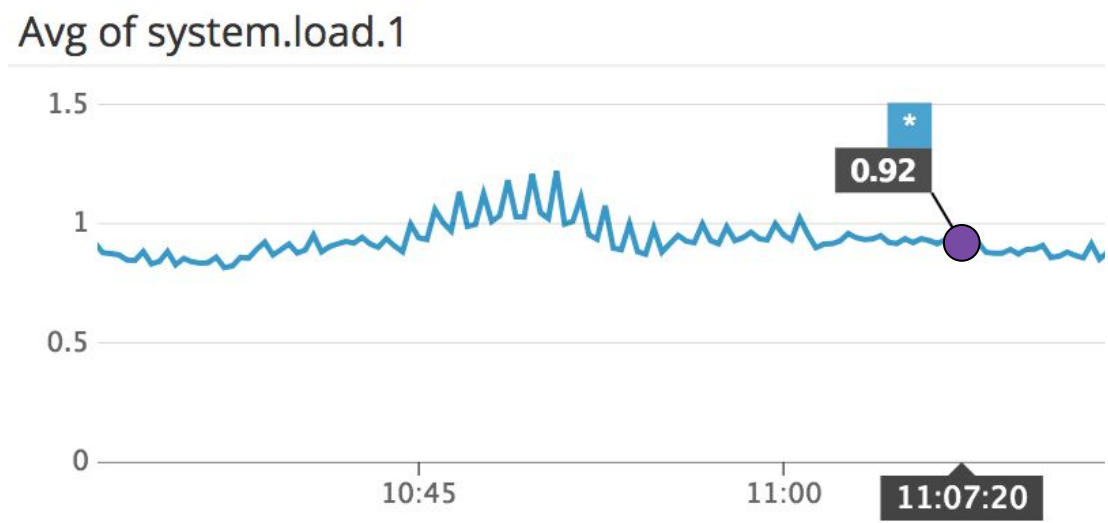


Example Metrics Query 1

“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,...



Tags for all the dimensions



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

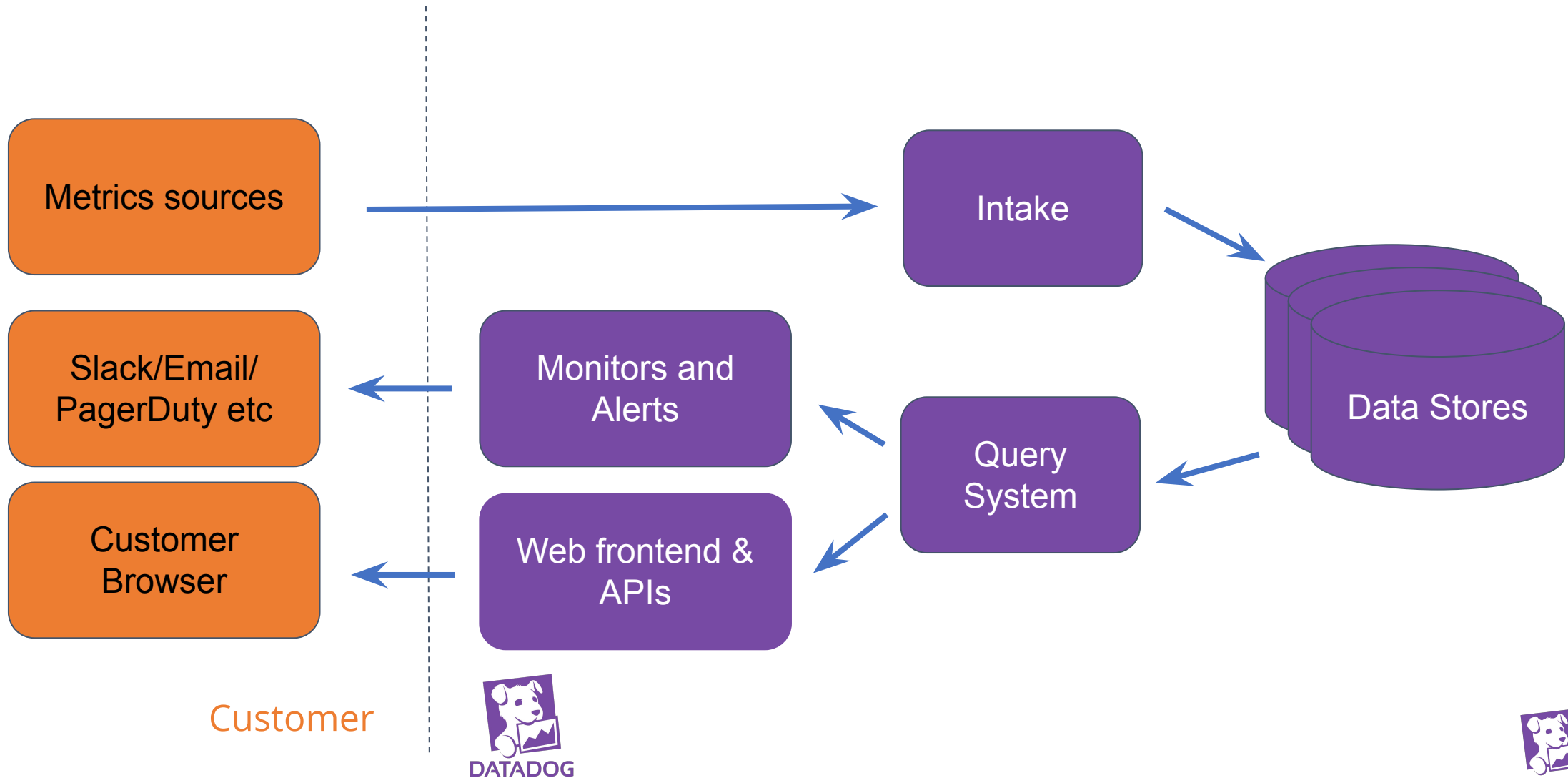
Application: internal cache hit rates, timers **by module**

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

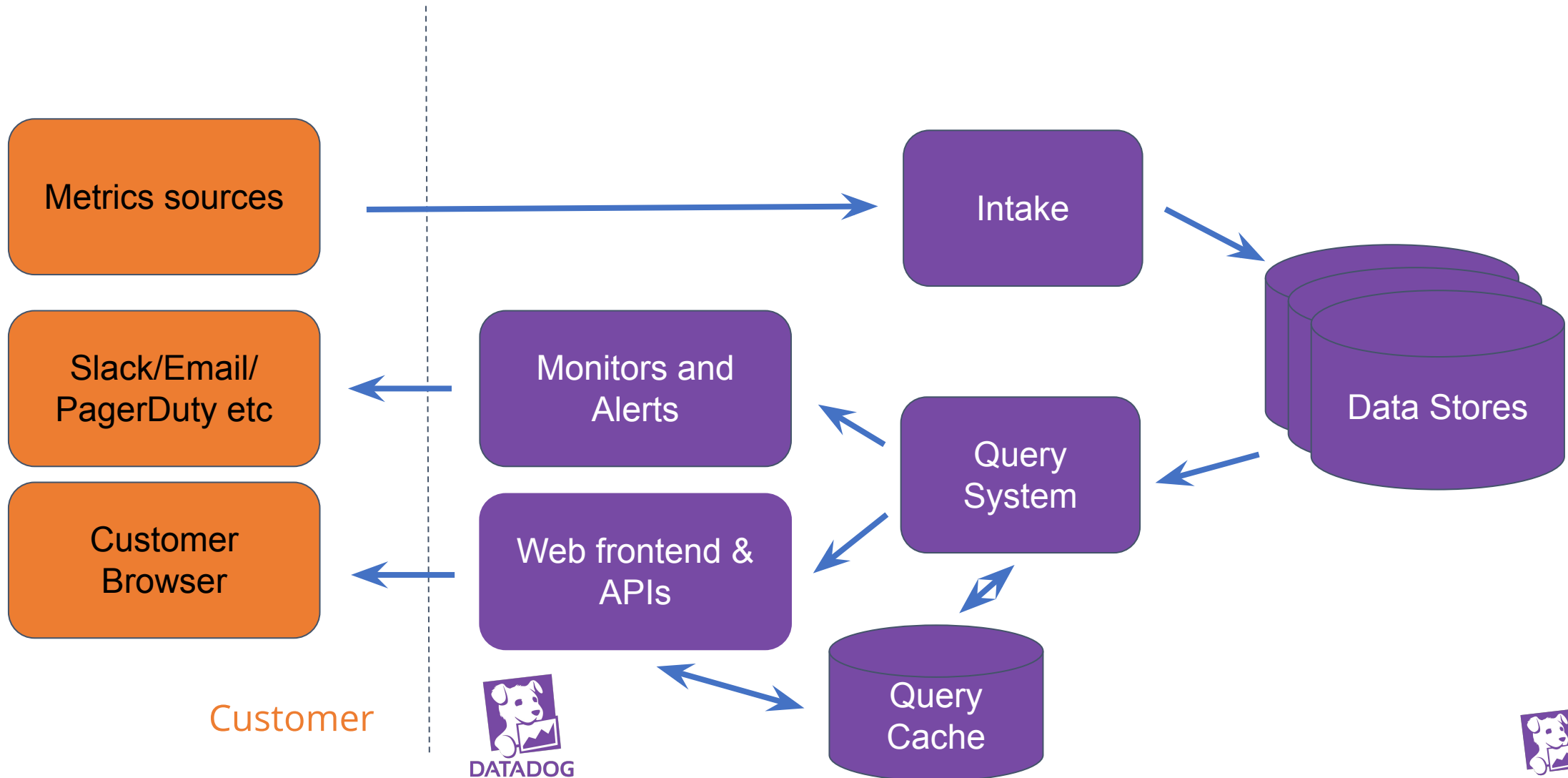
Business: # of orders processed, \$'s per second by **customer ID**



Pipeline Architecture



Caching timeseries data

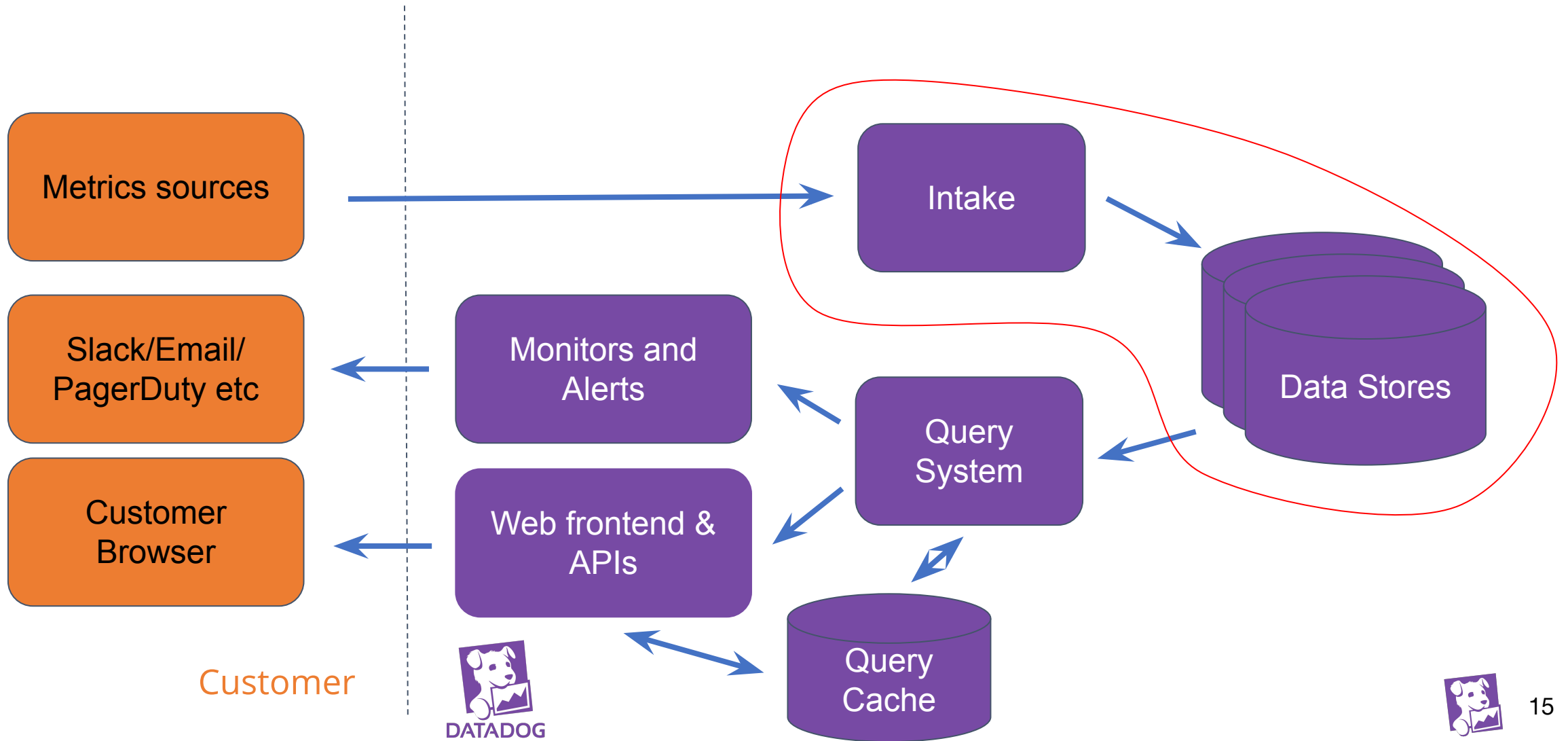


Performance mantras

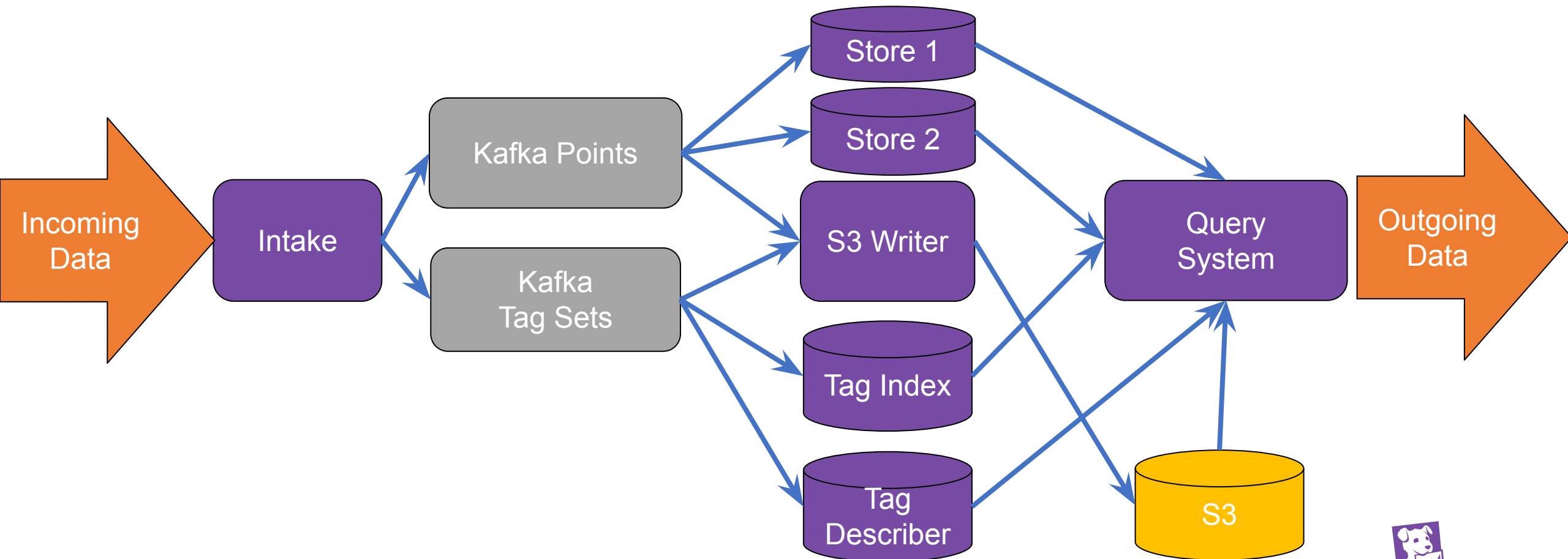
- Don't do it
- **Do it, but don't do it again - cache as much as you can**
- Do it less
- Do it later
- Do it when they're not looking
- Do it concurrently
- Do it cheaper



Zooming in



Kafka for Independent Storage Systems



Performance mantras

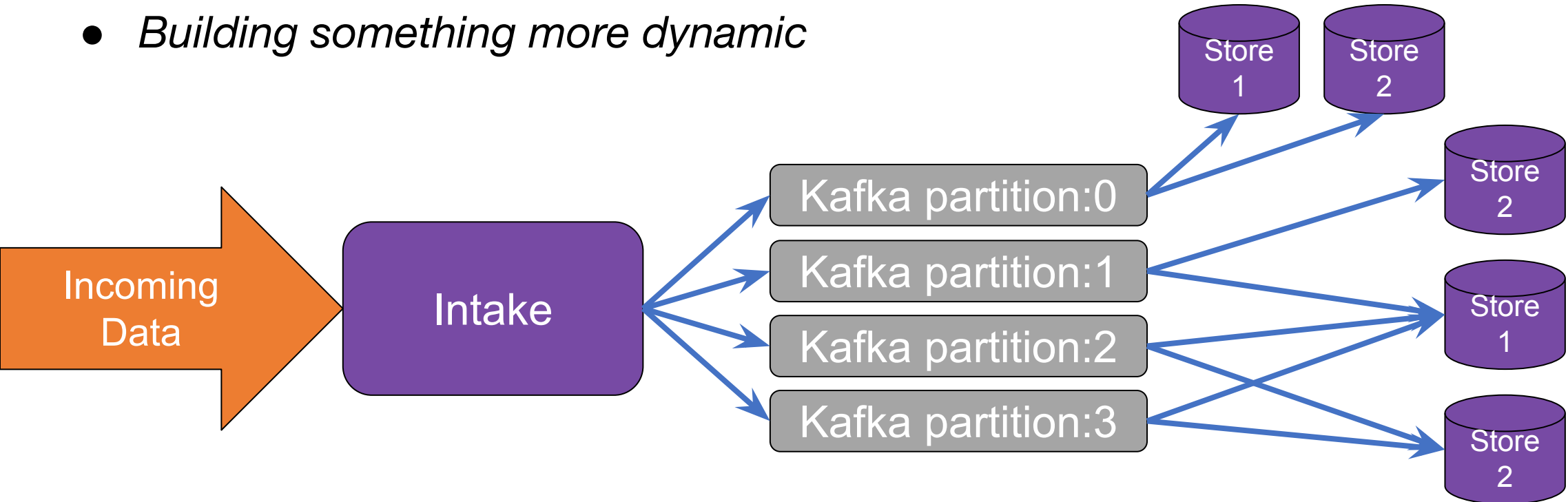
- Don't do it
- Do it, but don't do it again - cache as much as you can
- Do it less
- **Do it later - minimize upfront processing**
- Do it when they're not looking
- Do it concurrently
- Do it cheaper



Scaling through Kafka

Partition by customer, metric, tag set

- **Isolate** by customer
- **Scale concurrently** by metric
- *Building something more dynamic*



Performance mantras

- Don't do it
- Do it, but don't do it again - cache as much as you can
- Do it less
- Do it later - minimize upfront processing
- Do it when they're not looking
- **Do it concurrently - spread data across independent, scalable data stores**
- Do it cheaper



Talk Plan

1. Our Architecture
- 2. Deep Dive On Our Datastores**
3. Handling Synchronization
4. Approximation For Deeper Insights
5. Enabling Flexible Architecture



Trillions of points per day

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)

$$10^4 \times 10^3 \times 10^5 = 10^{12}$$



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 Customer



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



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



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 upfront processing
- Do it when they're not looking
- Do it concurrently - use independent horizontally scalable data stores
- Do it cheaper



Hybrid Data Storage

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



Hybrid Data Storage

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

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



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 upfront processing
- Do it when they're not looking
- Do it concurrently - use independent horizontally scalable data stores
- **Do it cheaper - match data latency requirements to cost**



Talk Plan

1. Our Architecture
2. Deep Dive On Our Datastores
- 3. Handling Synchronization**
4. Approximation For Deeper Insights
5. Enabling Flexible Architecture



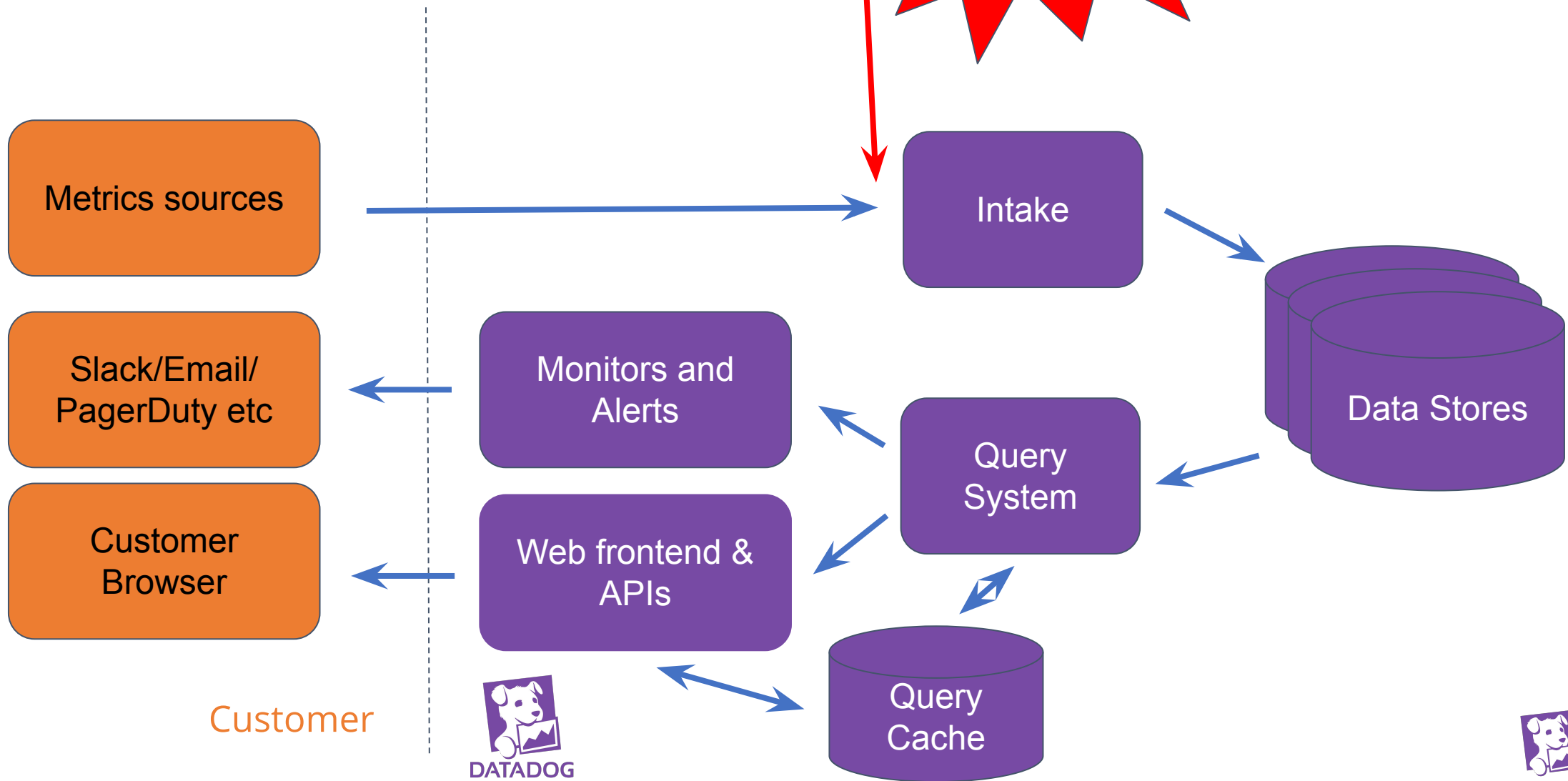
Alerts/Monitors Synchronization

- Required to prevent false positives
- Need all data for the evaluation time period is ready

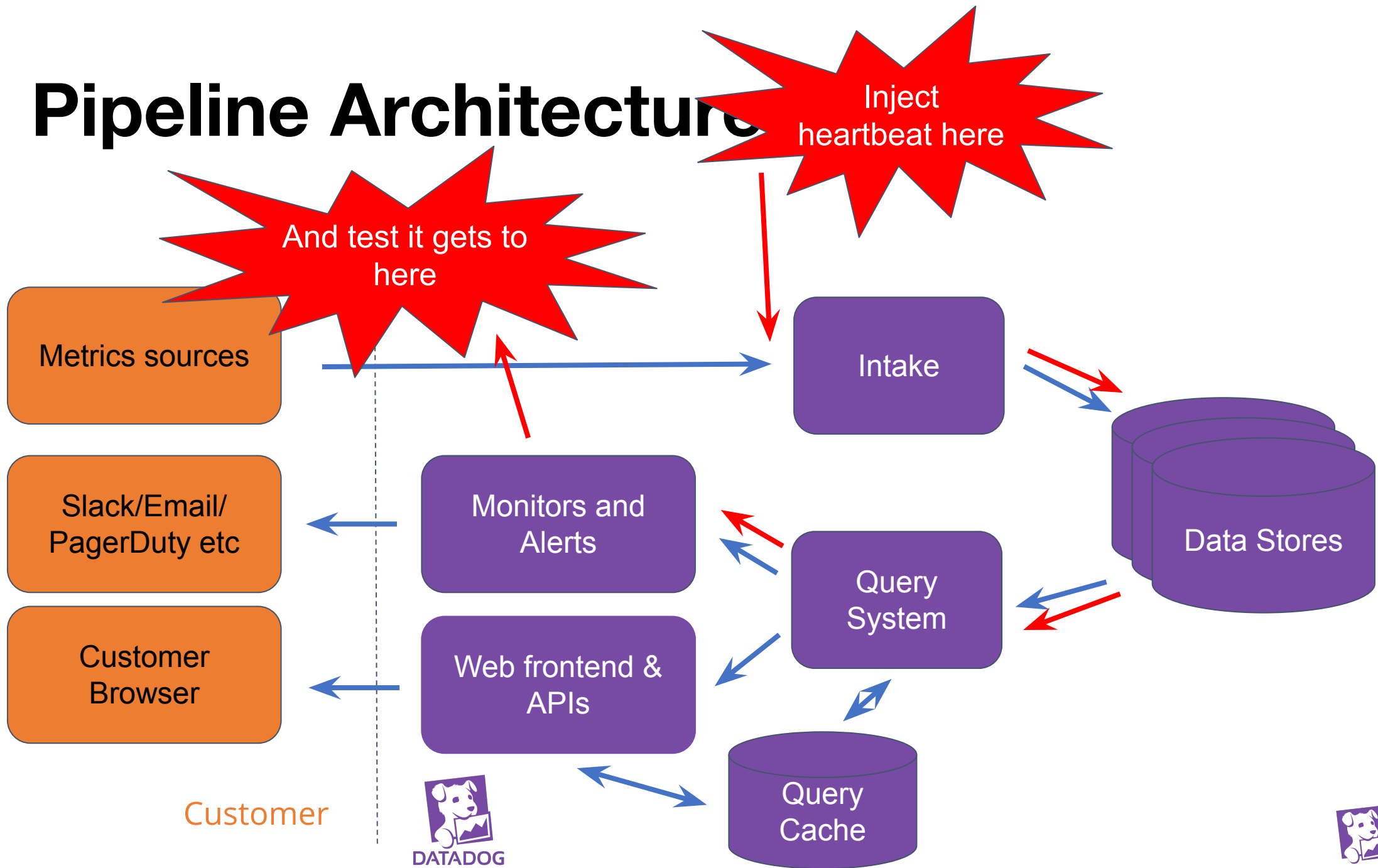


Pipeline Architecture

Inject
heartbeat here



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 upfront processing
- Do it when they're not looking
- Do it concurrently - use independent horizontally scalable data stores
- Do it cheaper - match data latency requirements to cost



Talk Plan

1. Our Architecture
2. Deep Dive On Our Datastores
3. Handling Synchronization
- 4. Approximation For Deeper Insights**
5. Enabling Flexible Architecture



Types of metrics



Counter, aggregate by sum

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



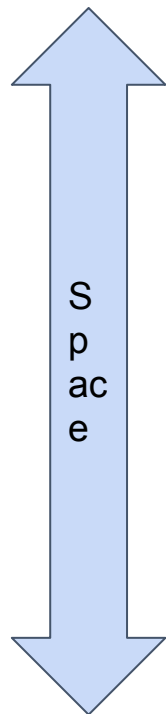
Gauges, aggregate by last or avg

Ex: CPU/network/disk use, queue length



Aggregation for counters and gauges

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}



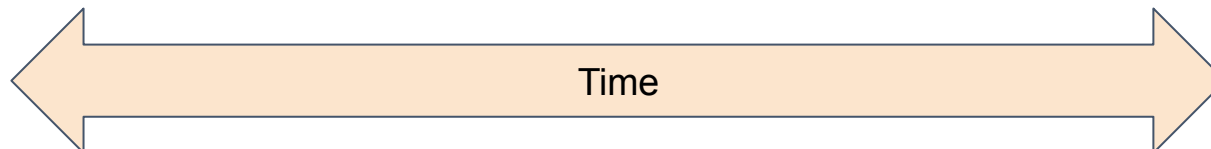
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}



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



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



Query output

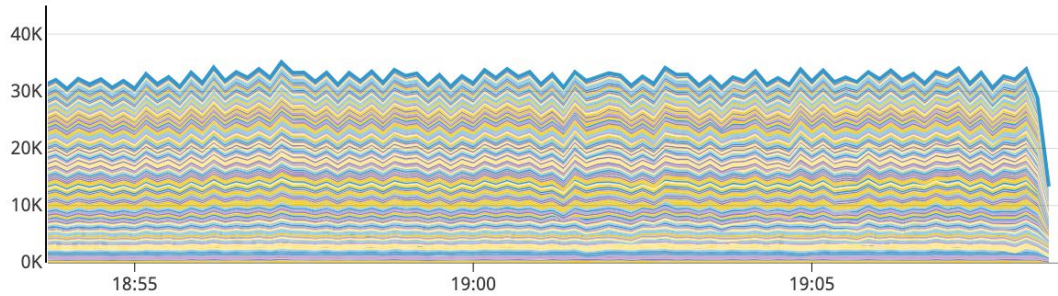
Counters: {5, 40, 50, 1023}

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

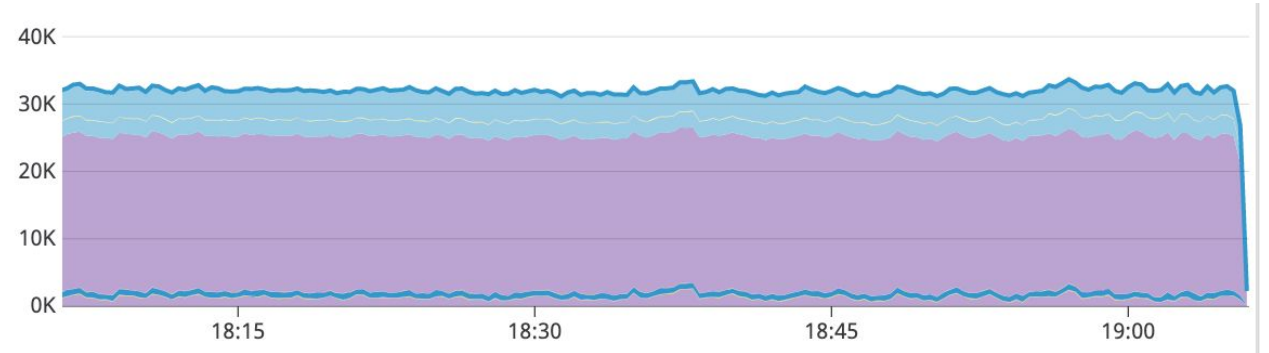
Gauges (last): {1, 9, 5, 512}



Focus on outputs



Showing 3254 series from 1 queries.



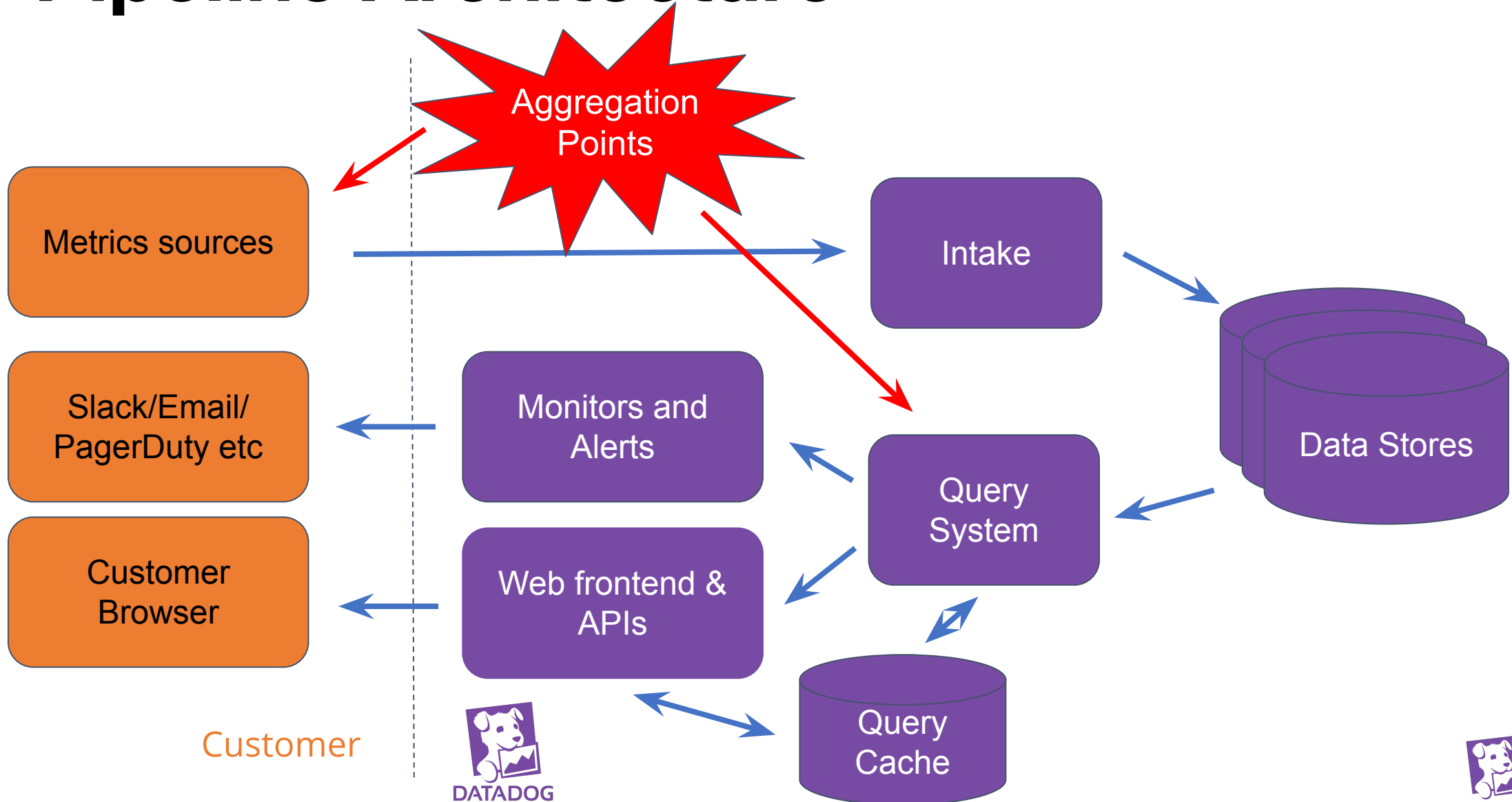
Showing 33 series from 1 queries.

These graphs are *both* aggregating 70k series

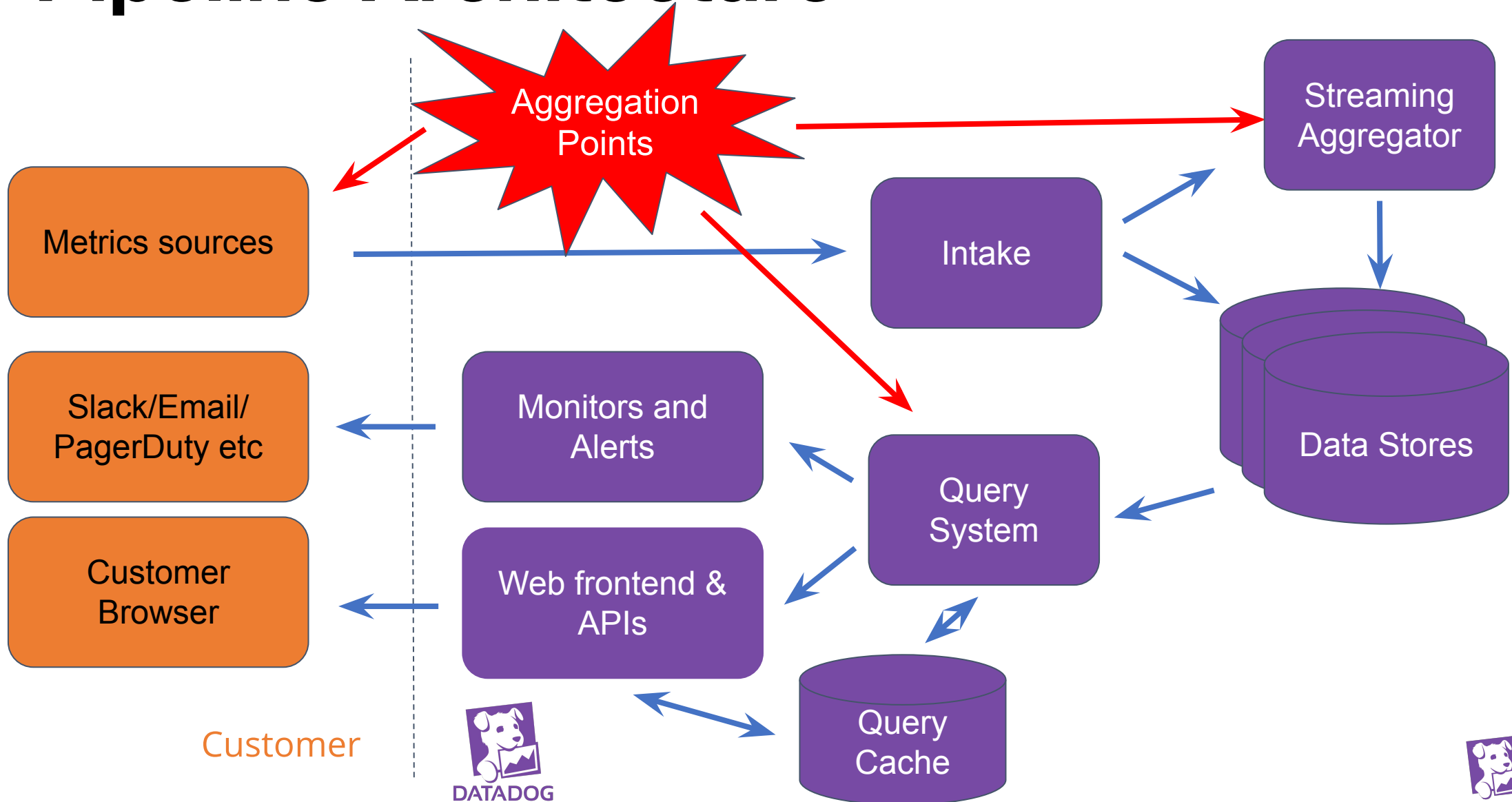
Output 20 to 2000 times less series than input



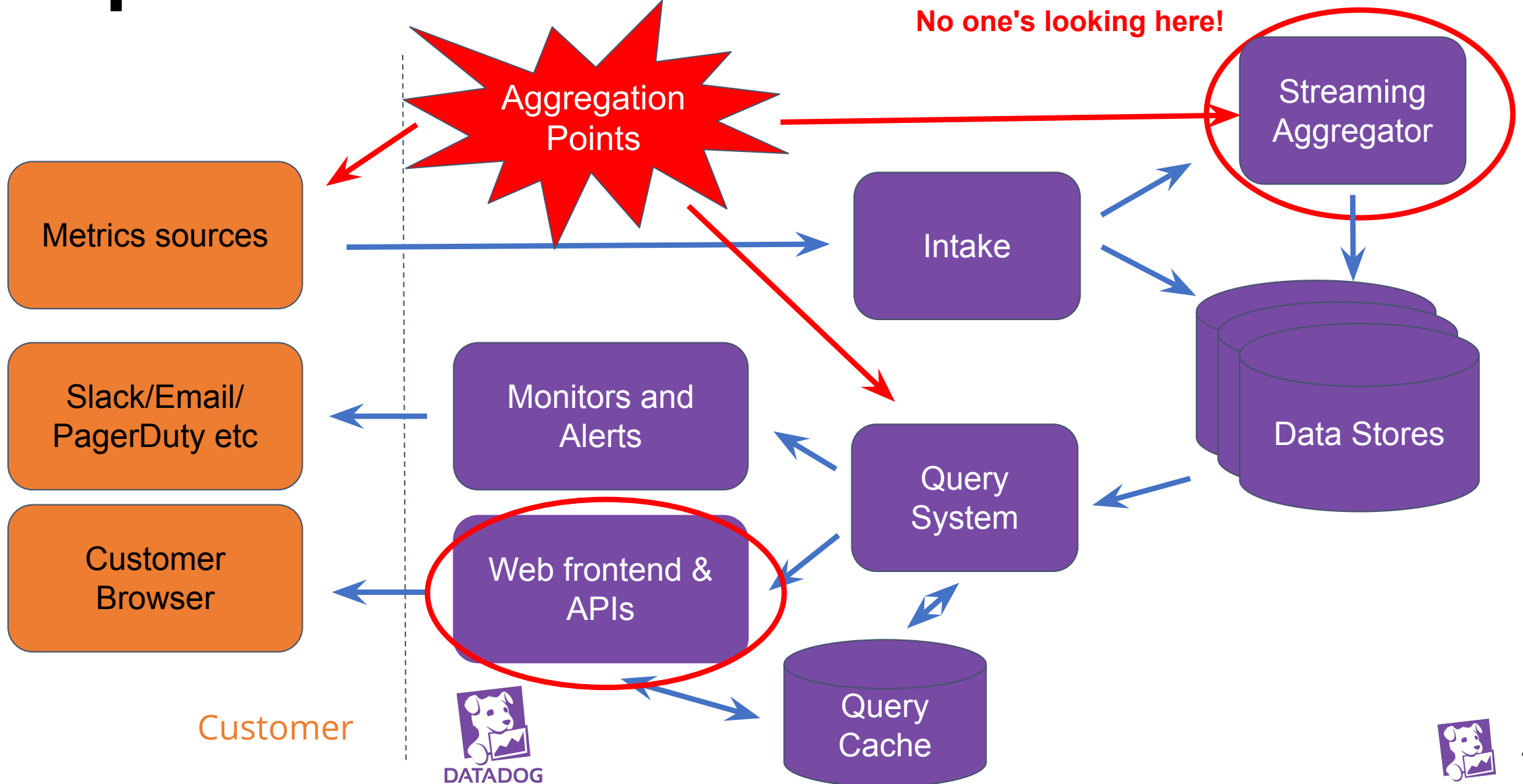
Pipeline Architecture



Pipeline Architecture



Pipeline Architecture

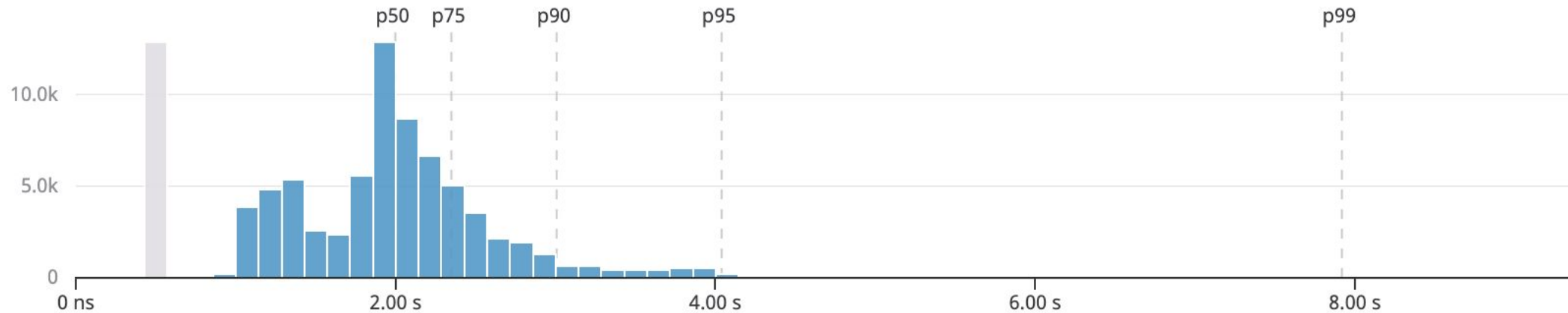


Performance mantras

- 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



Distributions



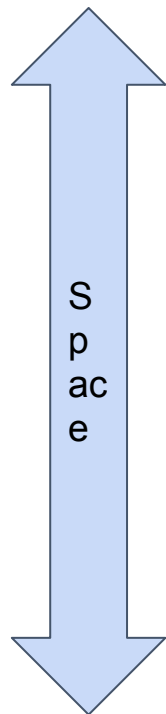
Aggregate by percentile or SLO
(count of values above or below a threshold)

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
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------



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



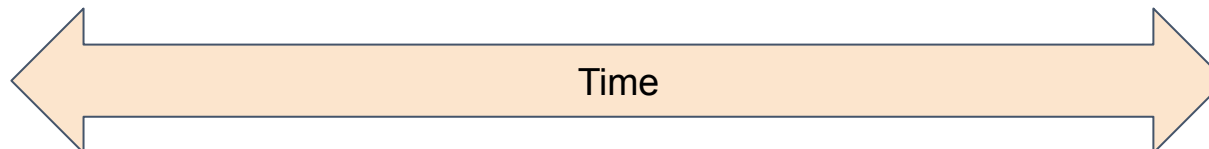
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}



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



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



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, 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 upfront processing
- Do it when they're not looking
- Do it concurrently - use independent horizontally scalable data stores
- **Do it cheaper again?**

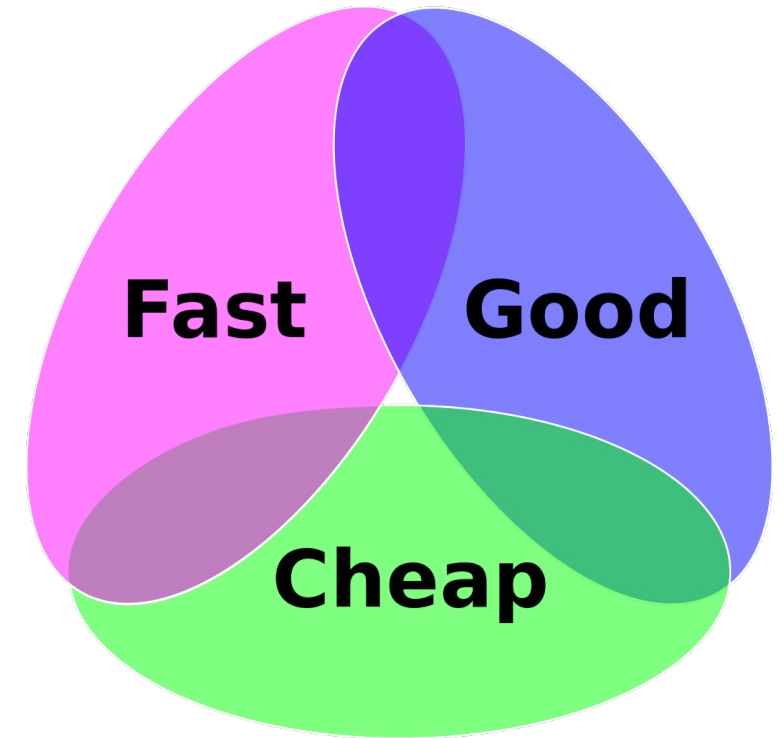


Tradeoffs

Engineering triangle - fast, good or cheap

What's the universe of valid values? (inputs)

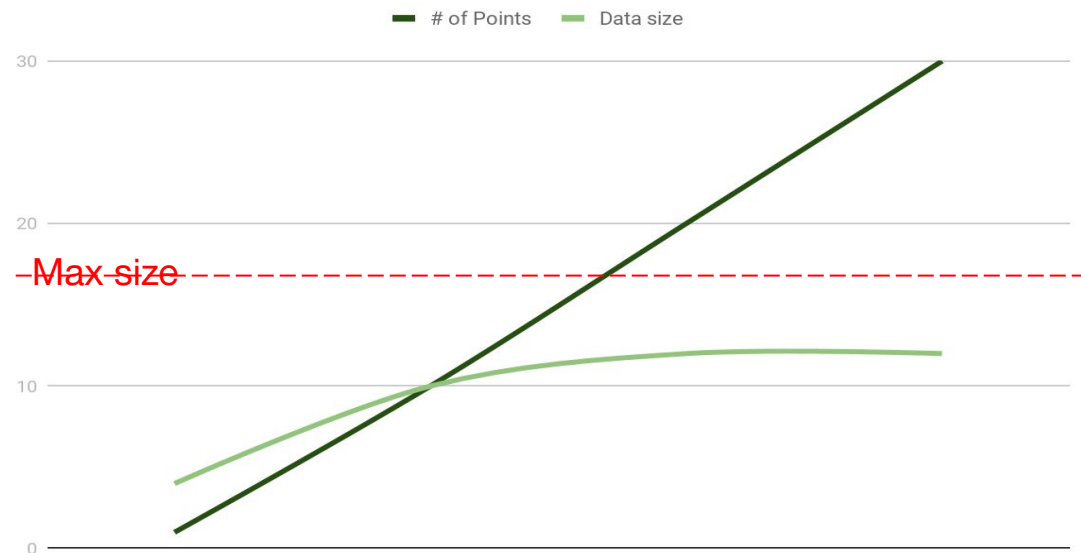
What are common queries? (outputs)



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)



You may know these sketches

HyperLogLog

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

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



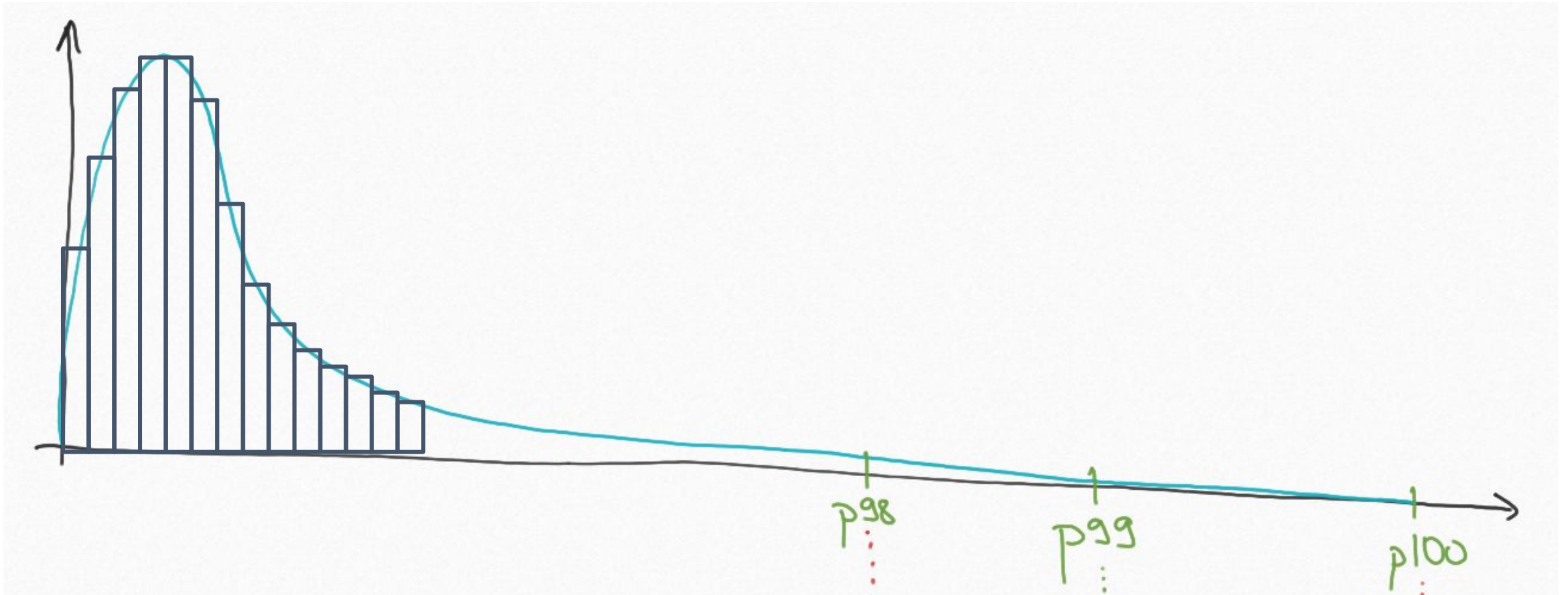
Approximation for distribution metrics

What's important for approximating distribution metrics?

- Good: accurate
- Fast: quick insertion & queries
- Cheap: bounded-size storage



Approximating a distribution



Bucketed 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
```



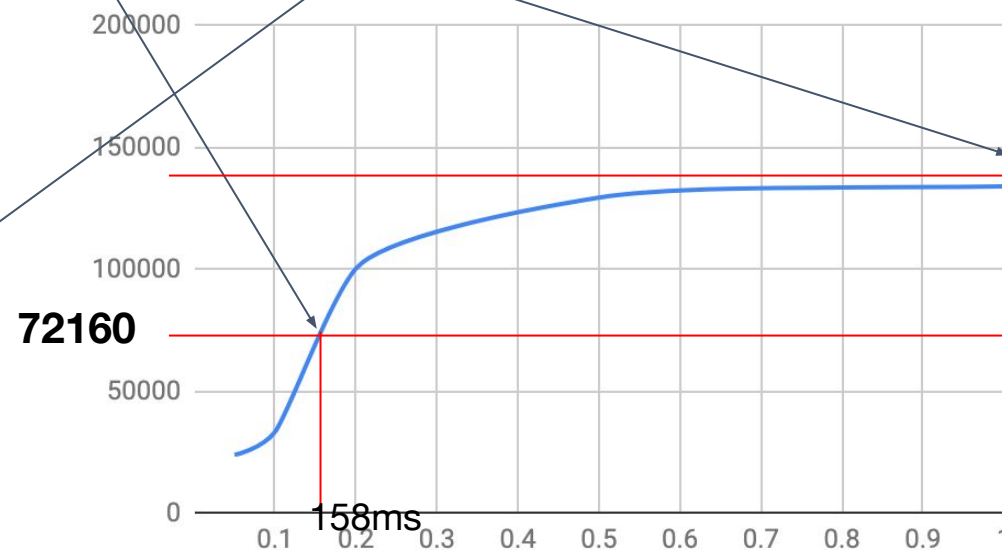
Bucketed histograms

Basic example from OpenMetrics / Prometheus

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

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

p99 = ?!

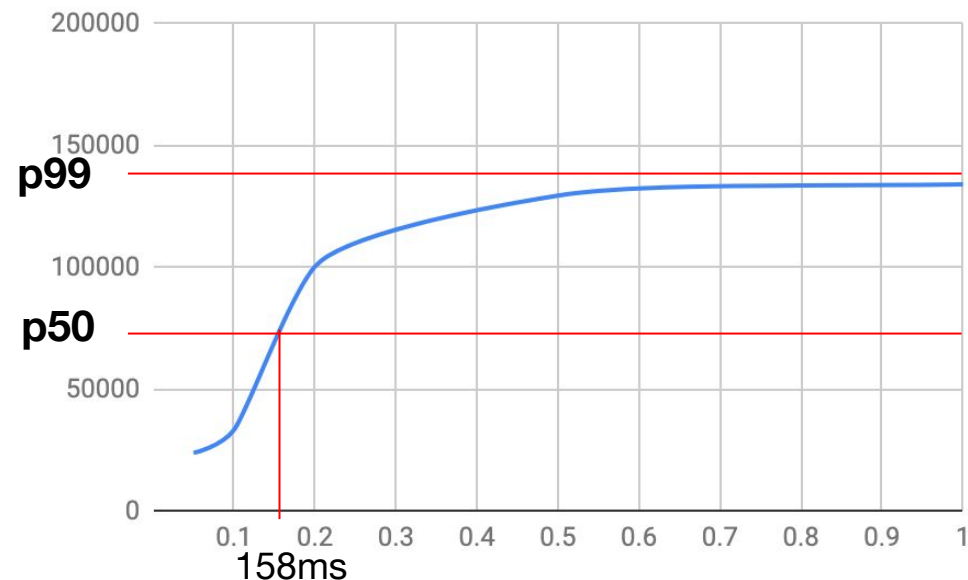


Bucketed histograms

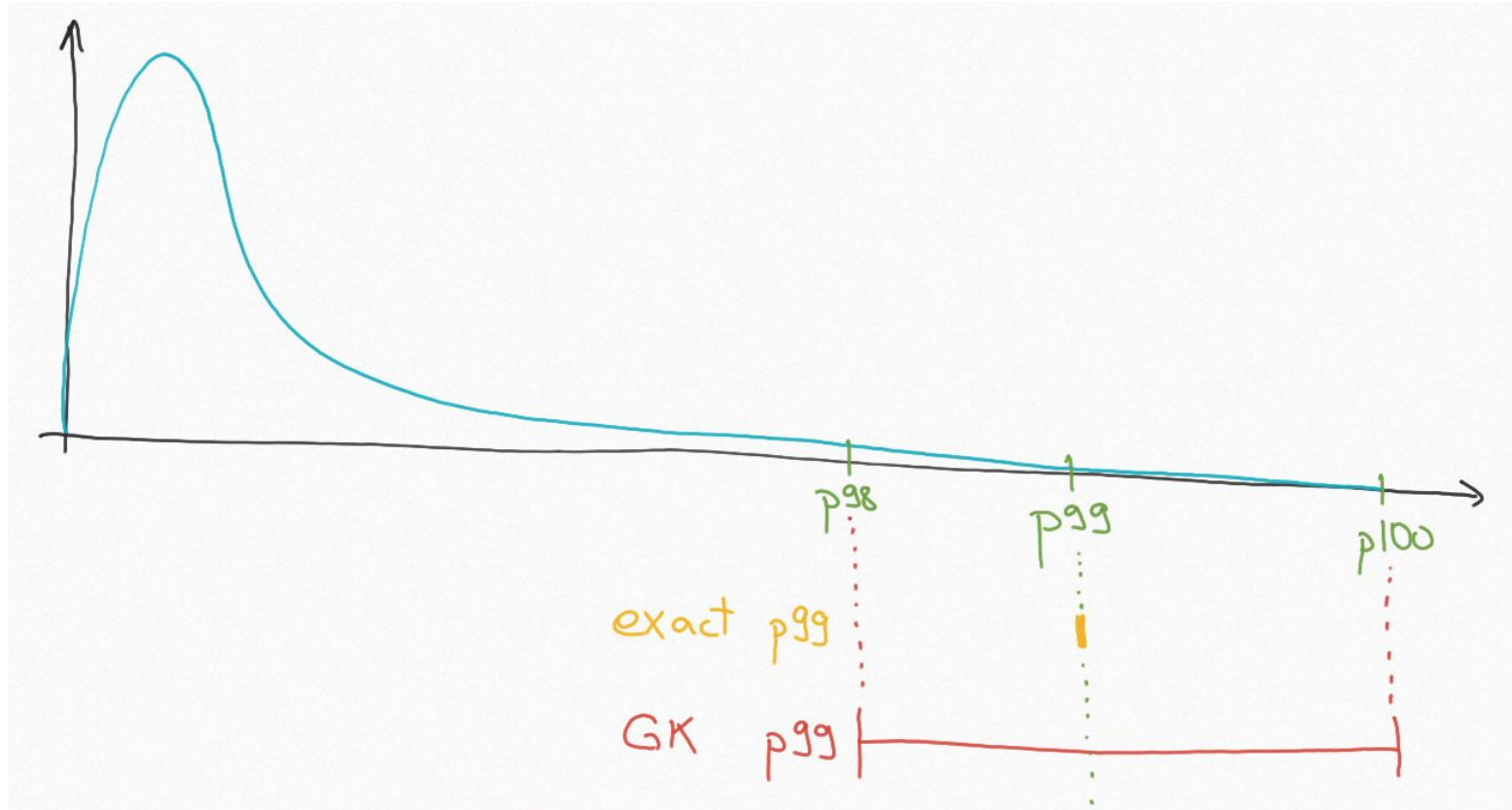
Basic example from OpenMetrics / Prometheus

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

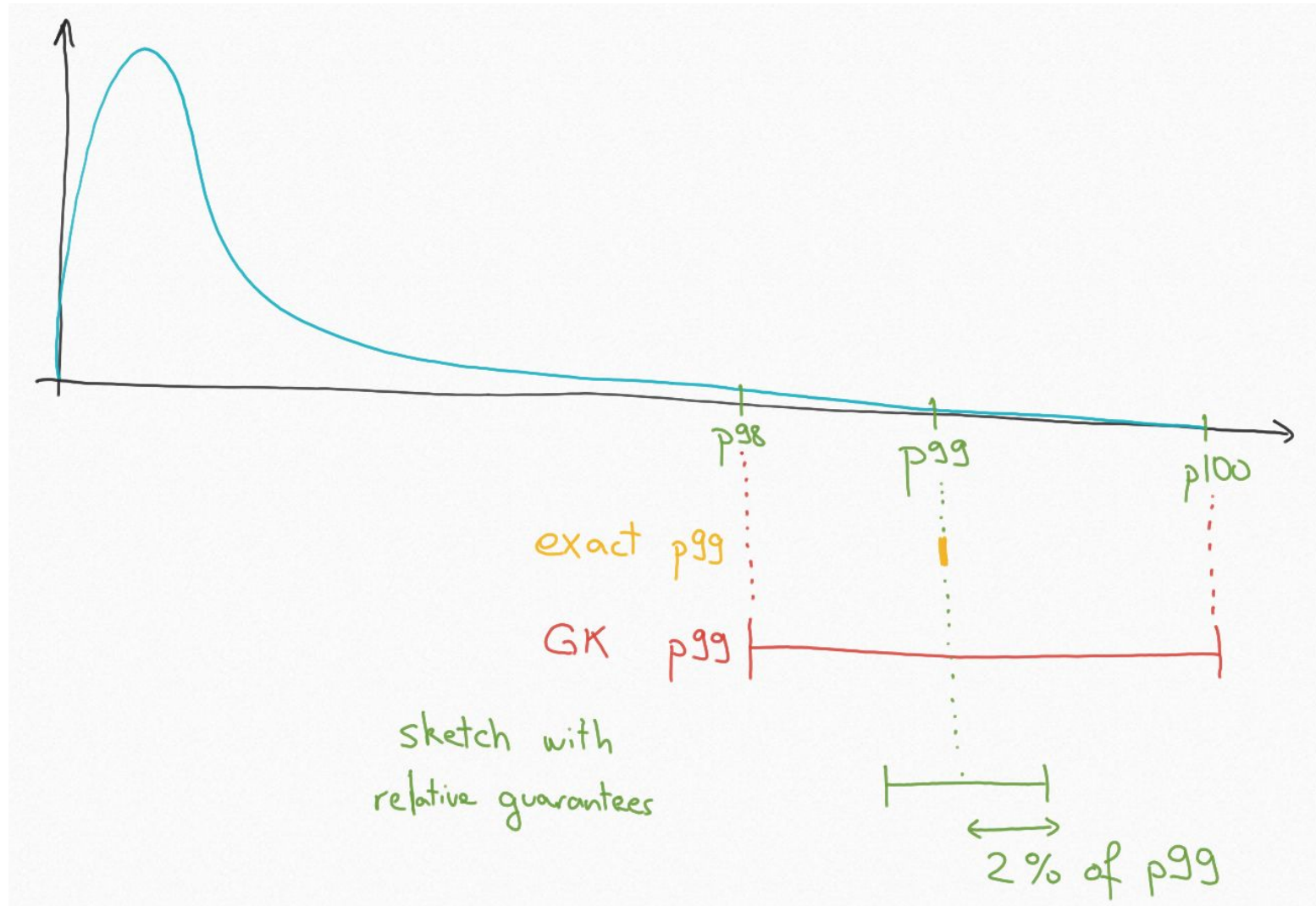
median = ~**158ms** (using linear interpolation)



Rank and relative error



Rank and relative error



Good: relative error



Relative error bounds mean we can answer this: Yes, within 99% of requests are $\leq 500\text{ms} \pm 1\%$

Otherwise stated: 99% of requests are **guaranteed** $\leq 505\text{ms}$



Cheap: fixed storage size



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*



Fast: insertion & query



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

Queries look at the fixed number of buckets



DDSketch

DDSketch (Distributed Distribution Sketch) is open source

- Presented at VLDB2019 in August
- Open-source versions in several languages

Python: github.com/DataDog/sketches-py

Java: github.com/DataDog/sketches-java

Go: github.com/DataDog/sketches-go



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 upfront processing
- Do it when they're not looking
- Do it concurrently - use independent horizontally scalable data stores
- **Do it cheaper - leverage approximation**



Talk Plan

1. Our Architecture
2. Deep Dive On Our Datastores
3. Handling Synchronization
4. Approximation For Deeper Insights
- 5. Enabling Flexible Architecture**



Commutativity

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

Why is this important?



Commutativity

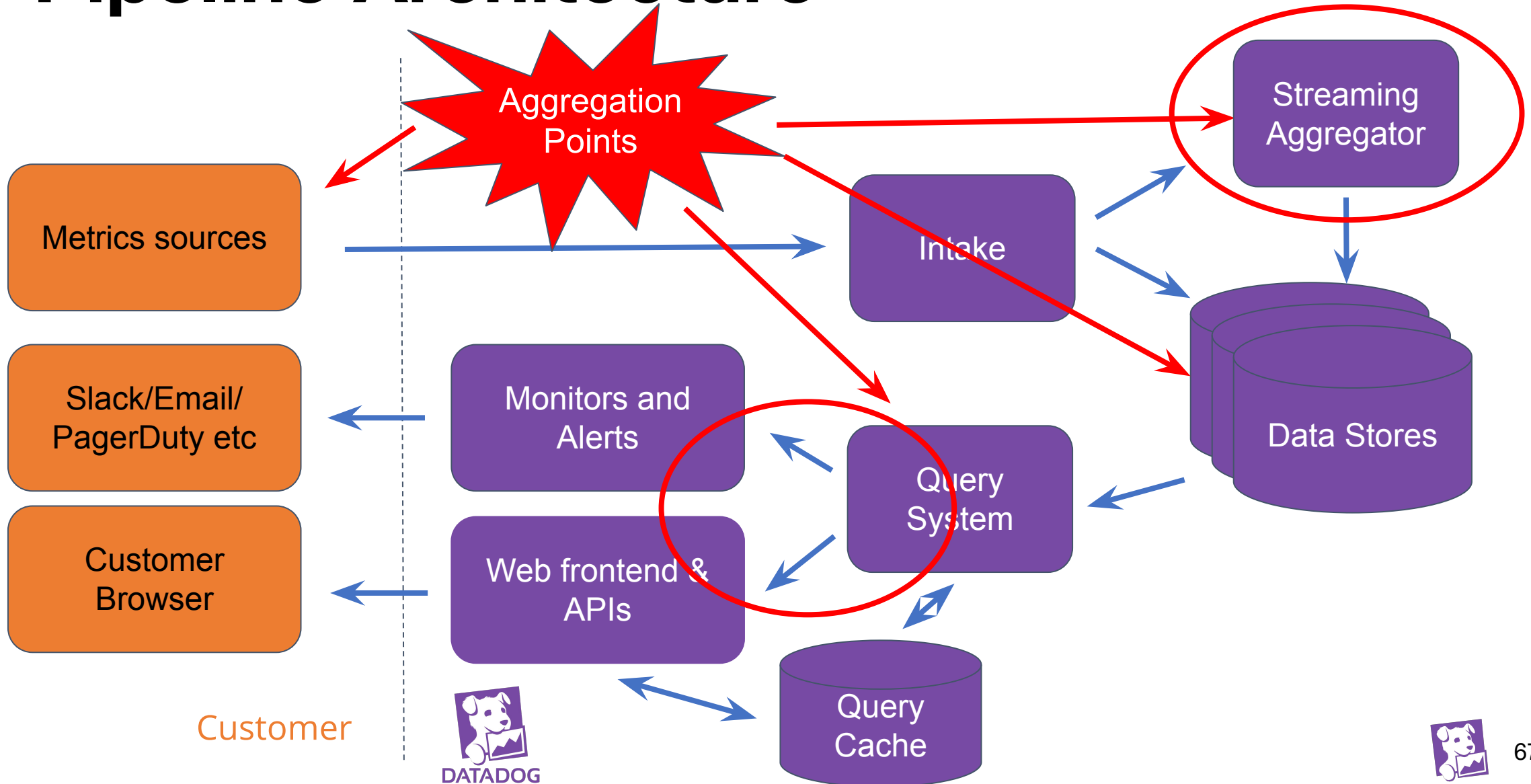
"a binary operation is **commutative** if *changing the order* of the operands does not change the result"

Why is this important?

Distribute aggregation work throughout the pipeline



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 upfront processing
- **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 leverage approximation



- **Don't do it** - build the bare minimal synchronization needed
- **Do it, but don't do it again** - cache as much as you can
- **Do it less** - only index what you need
- **Do it later** - minimize upfront processing
- **Do it when they're not looking** - pre-aggregate where is cost effective
- **Do it concurrently** - use independent horizontally scalable data stores
- **Do it cheaper** - use hybrid data storage types and technologies and leverage approximation

**Do exactly as much work as needed,
and no more**



Thank You



Rate today's session

Cyberconflict: A new era of war, sabotage, and fear

David Sanger (The New York Times)
9:55am-10:10am Wednesday, March 27, 2019
Location: Ballroom
Secondary topics: Security and Privacy

See passes & pricing

 Add to Your Schedule
 Add Comment or Question

Rate This Session

We're living in a new era of constant sabotage, misinformation, and fear, in which everyone is a target, and you're often the collateral damage in a growing conflict among states. From crippling infrastructure to sowing discord and doubt, cyber is now the weapon of choice for democracies, dictators, and terrorists.

David Sanger explains how the rise of cyberweapons has transformed geopolitics like nothing since the invention of the atomic bomb. Moving from the White House Situation Room to the dens of Chinese, Russian, North Korean, and Iranian hackers to the boardrooms of Silicon Valley, David reveals a world coming face-to-face with the perils of technological revolution—a conflict that the United States helped start when it began using cyberweapons against Iranian nuclear plants and North Korean missile launches. But now we find ourselves in a conflict we're uncertain how to control, as our adversaries exploit vulnerabilities in our hyperconnected nation and we struggle to figure out how to deter these complex, short-of-war attacks.

David Sanger
The New York Times

David E. Sanger is the national security correspondent for the *New York Times* as well as a national security and political contributor for CNN and a frequent guest on *CBS This Morning*, *Face the Nation*, and many PBS shows.



Session page on conference website

✓ Attending

Notes

Remove

Cyberconflict: A new era of war, sabotage, and fear

9:55 AM - 10:10 AM, Wed, Mar 27, 2019

Speakers




David Sanger
National Security Correspondent
The New York Times

📍 Ballroom

Keynotes

David Sanger explains how the rise of cyberweapons has transformed geopolitics like nothing since the invention of the atomic bomb. From crippling infrastructure to sowing discord and doubt, cyber is now the weapon of choice for democracies, dictators, and terrorists.

 SESSION EVALUATION

O'Reilly Events App