Scaling Telemetry Systems

Reliability best practices for streaming data

Liz Fong-Jones & Terra Field

Field CTO
@lizthegrey

Staff Platform Engineer
@rainofterra

Scaling Telemetry Systems

Reliability lessons for streaming data

Liz Fong-Jones & Terra Field

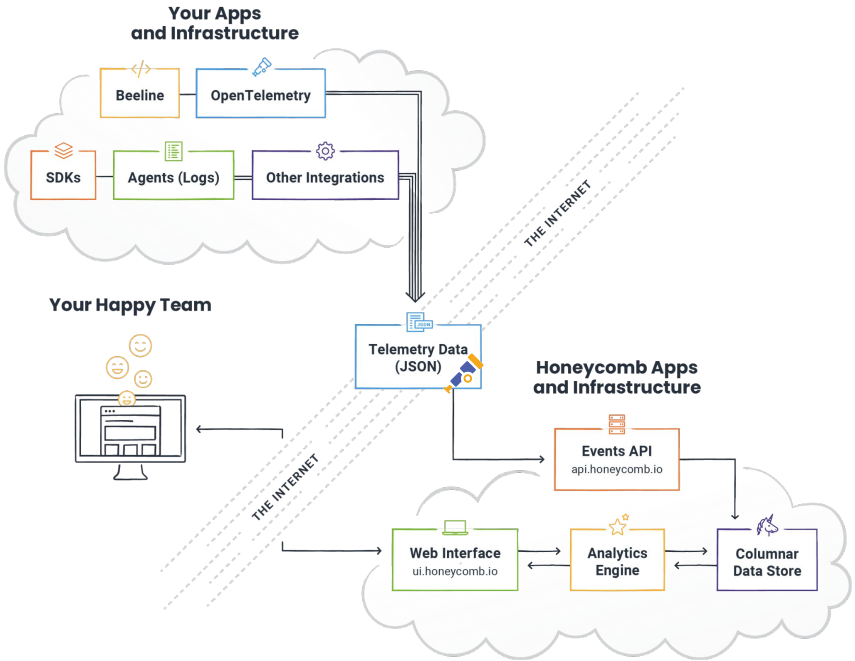
Field CTO
@lizthegrey

Staff Platform Engineer
@rainofterra

Our data ingest, storage, & analytics use case

What Honeycomb does

- Ingests customer's telemetry
- Indexes on every column
- Enables near-real-time querying on newly ingested data



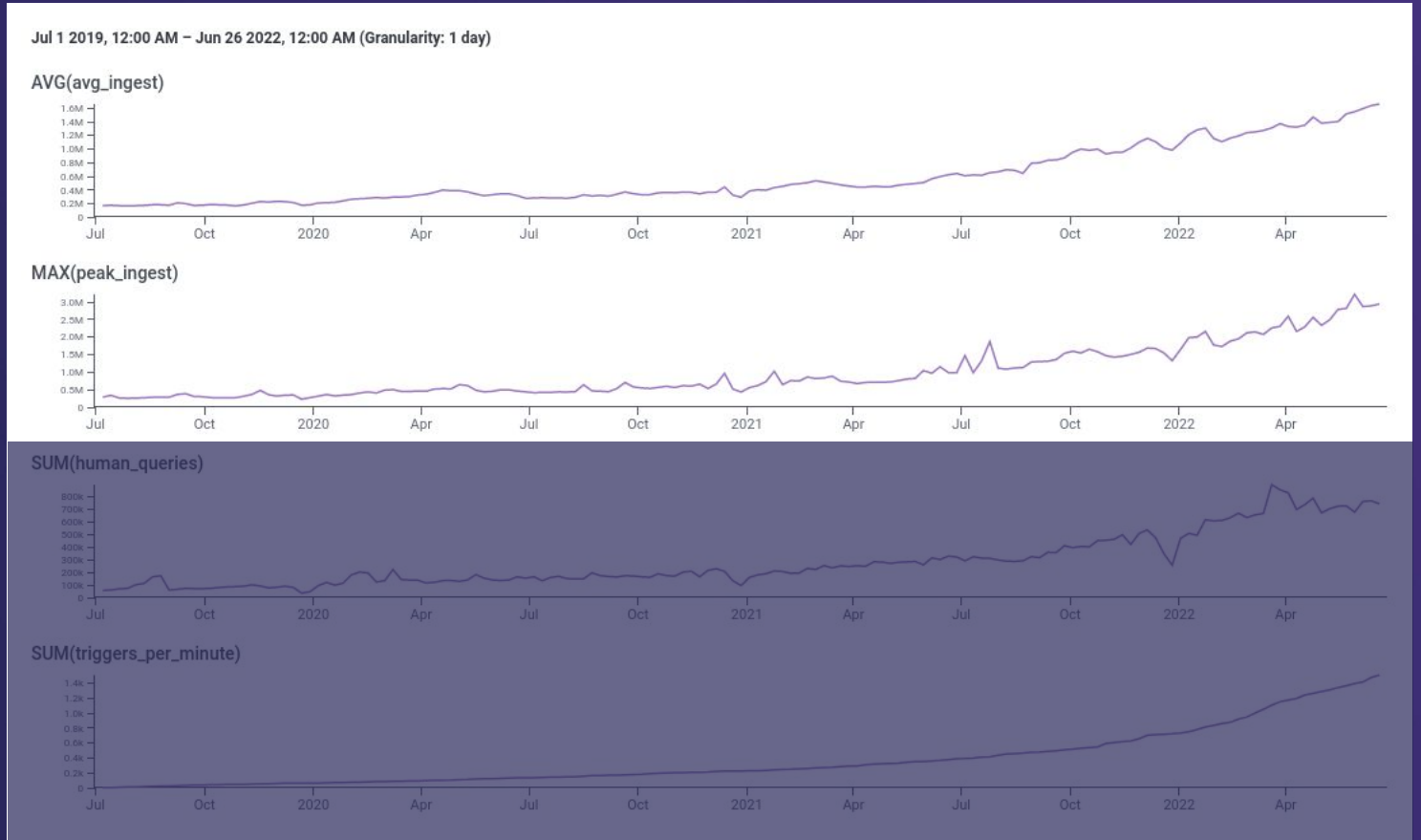
Our data ingest, storage, & analytics use case

What Honeycomb *really* does

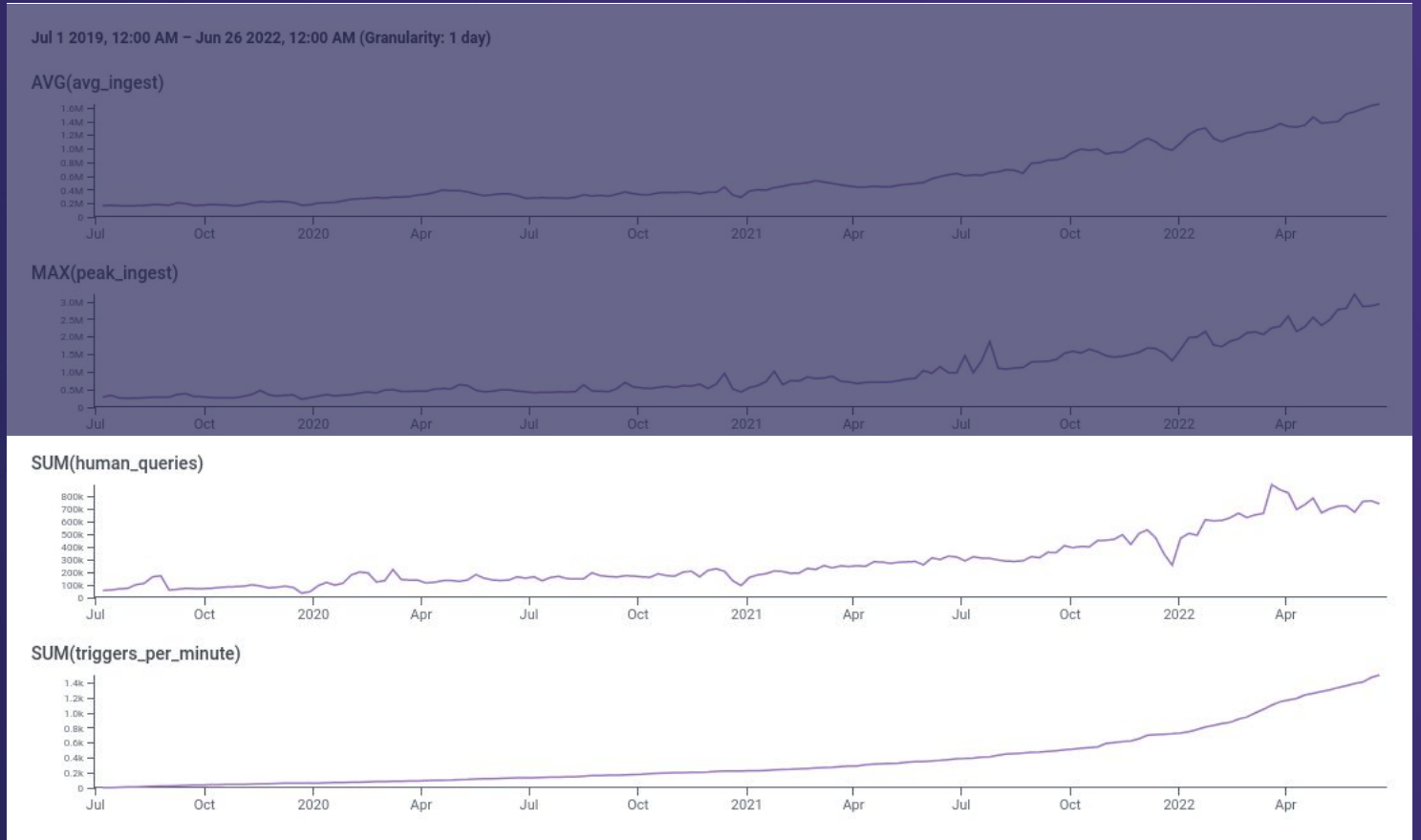
Kafka is the beating heart of Honeycomb.

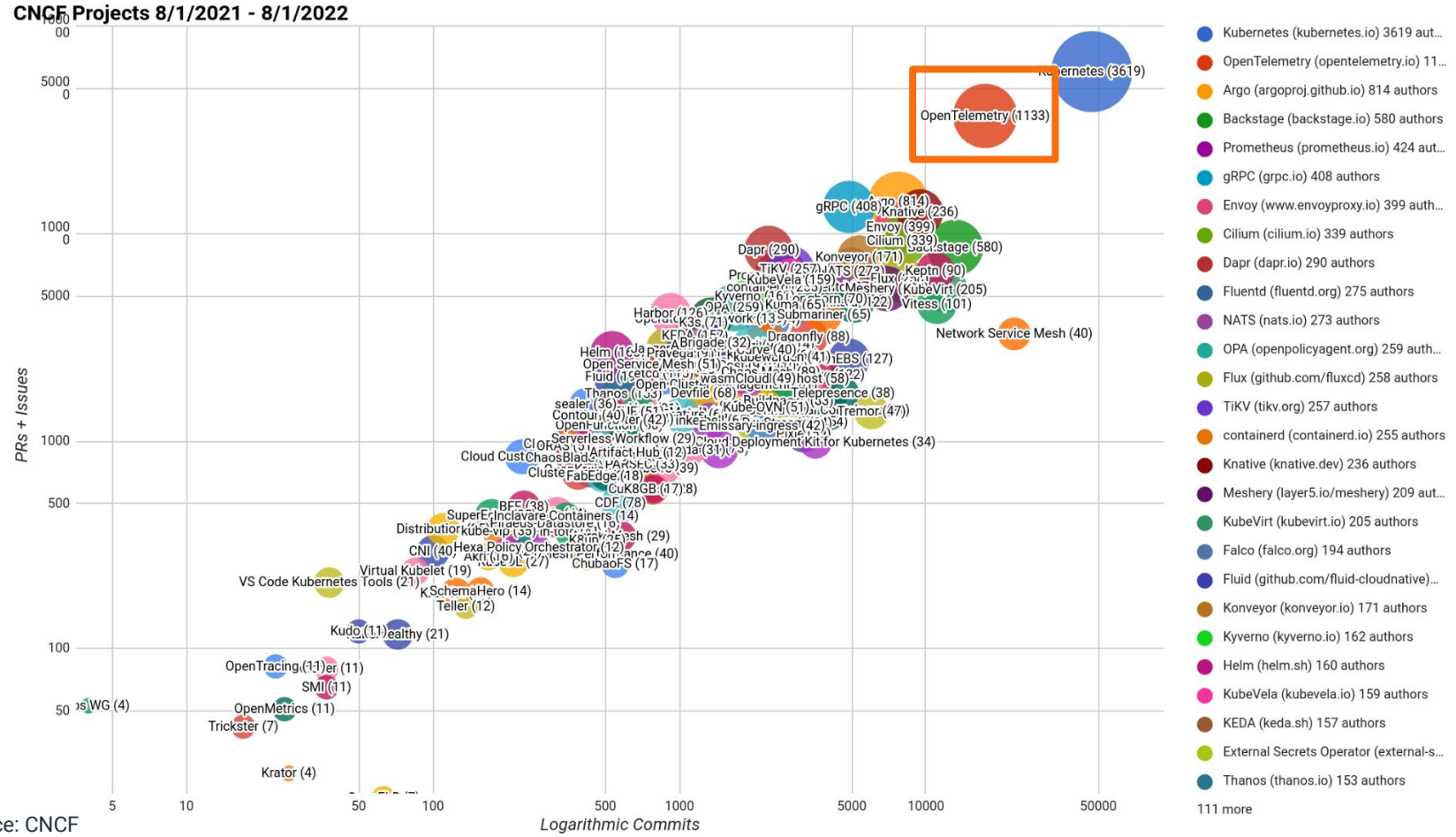
Ben Hartshorne, first employee at Honeycomb

10x growth in three years

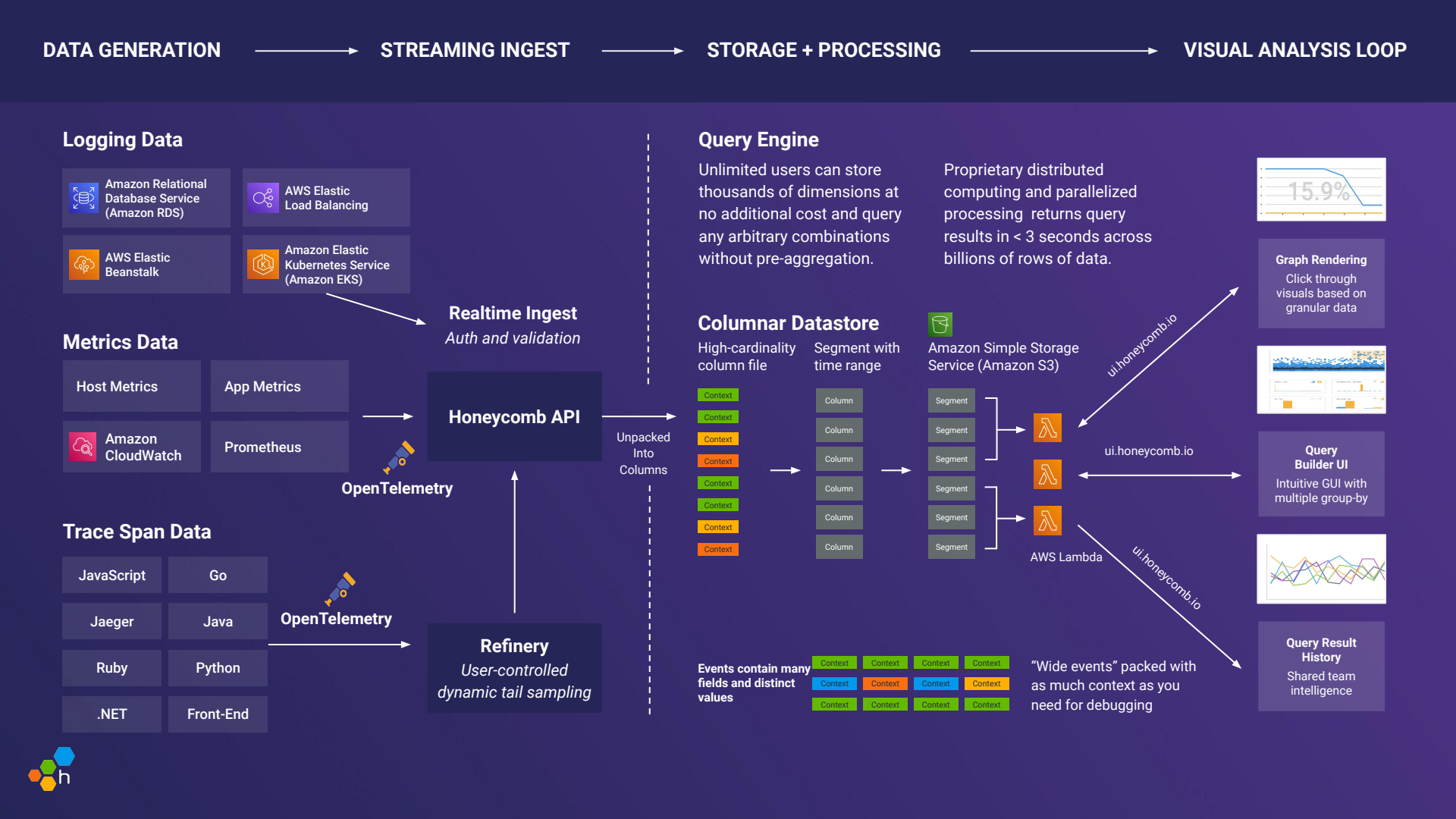


10x growth in three years





Source: CNCF


DATA GENERATION
STREAMING INGEST
STORAGE + PROCESSING
VISUAL ANALYSIS LOOP
Logging Data
Amazon Relational Database Service (Amazon RDS)
AWS Elastic Load Balancing
AWS Elastic Beanstalk
Amazon Elastic Kubernetes Service (Amazon EKS)
Metrics Data
Host Metrics
App Metrics
Amazon CloudWatch
Prometheus
OpenTelemetry
Trace Span Data
JavaScript
Go
Jaeger
Java
Ruby
Python
.NET
Front-End
OpenTelemetry
Realtime Ingest
Auth and validation
Honeycomb API
Refinery
User-controlled dynamic tail sampling
Unpacked Into Columns
Query Engine
Unlimited users can store thousands of dimensions at no additional cost and query any arbitrary combinations without pre-aggregation.
Proprietary distributed computing and parallelized processing returns query results in < 3 seconds across billions of rows of data.
Columnar Datastore
High-cardinality column file
Segment with time range
Amazon Simple Storage Service (Amazon S3)
Context
Column
Segment
AWS Lambda
ui.honeycomb.io
ui.honeycomb.io
ui.honeycomb.io
Events contain many fields and distinct values
"Wide events" packed with as much context as you need for debugging
15.9%
Graph Rendering
Click through visuals based on granular data
Query Builder UI
Intuitive GUI with multiple group-by
Query Result History
Shared team intelligence

Agenda

When to use streaming systems
e.g. Honeycomb's use case (and yours?)

Bottlenecks to streaming
Producers? Brokers? Consumers?

Patterns for scaling
How to address each bottleneck

Patterns for observing
Because you can't run it unless you can see it

TL;DR: tricky to get right, but more supportable than request/response.

Liz Fong-Jones

Field CTO, honeycomb.io

Terra Field

Staff Platform Engineer, honeycomb.io

What streaming is good for

How streaming powers Honeycomb's telemetry pipeline

Streaming data decouples systems

Separation of stateless & stateful

Update producers & consumers on-demand without dropping data. (mostly! SLO != 100%)

Keep one single record of truth.

Multiple fan-out on the event bus

This need came later, but was incredibly helpful.

- Originally: one producer, one consumer.
- Now: two producers, three consumers

Now all the state lives in one place…

This is both good…

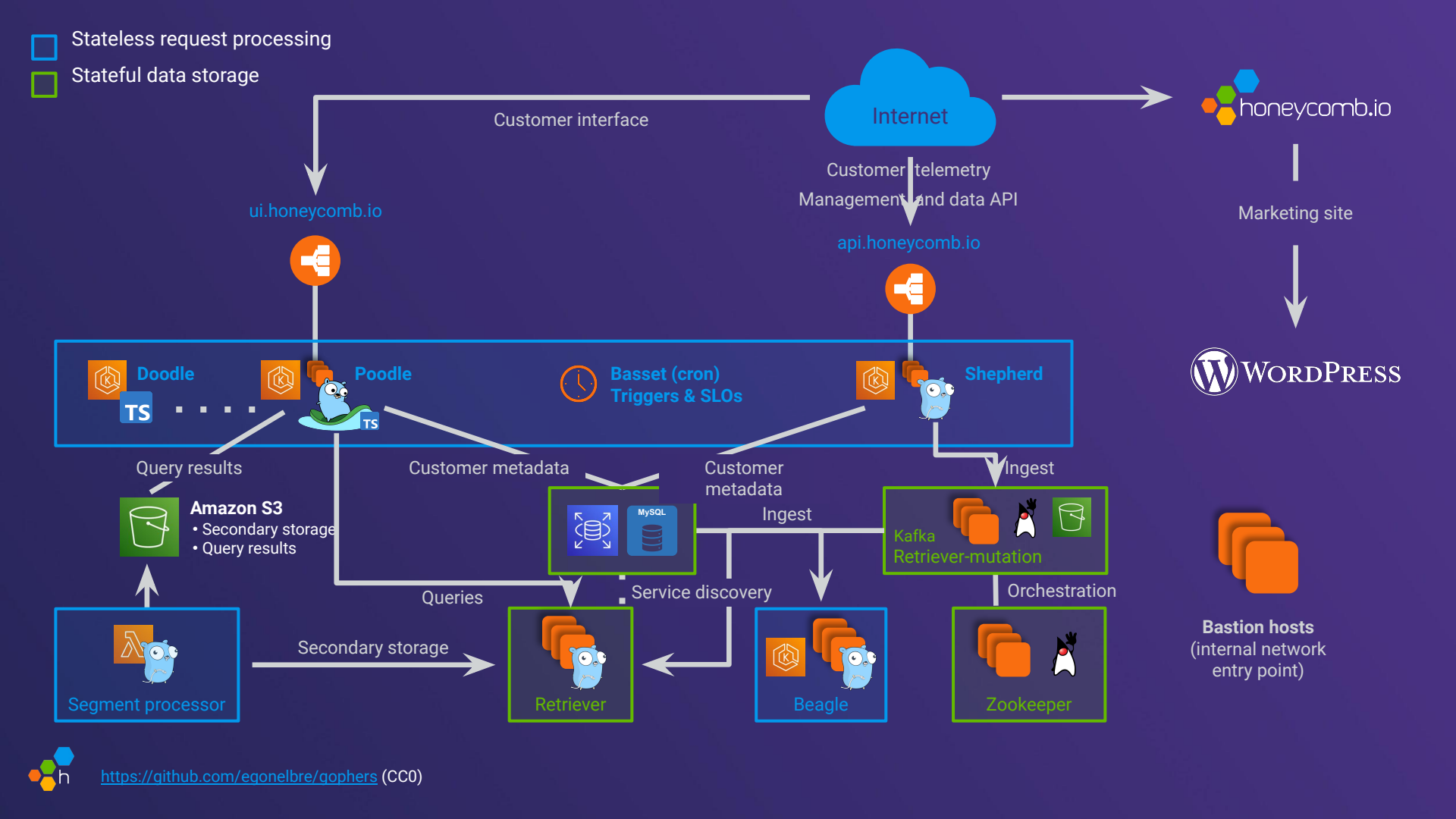
- Rolling restarts of everything else.
- Replay in case of incorrect consumer behaviour

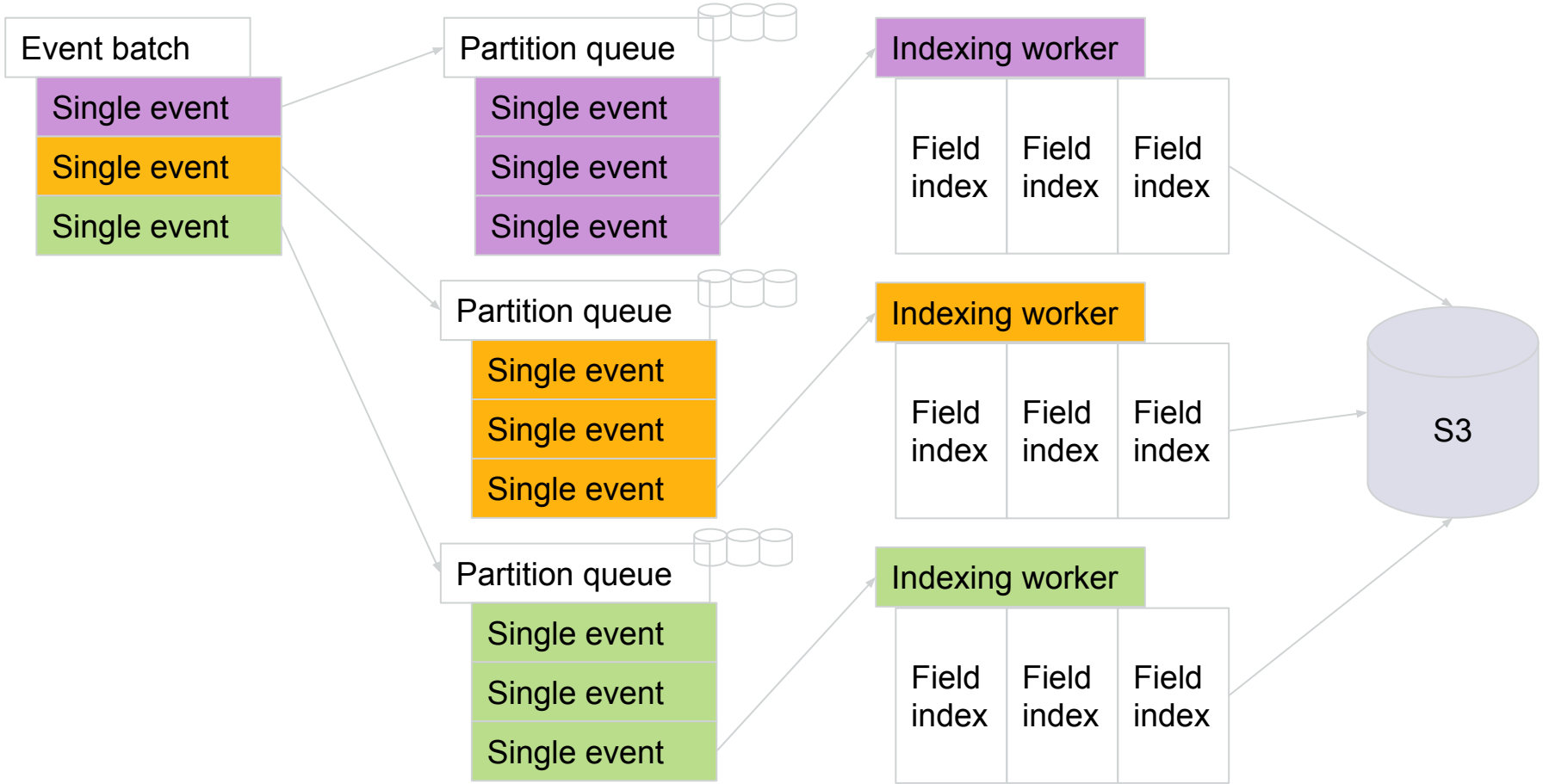
… and scary.

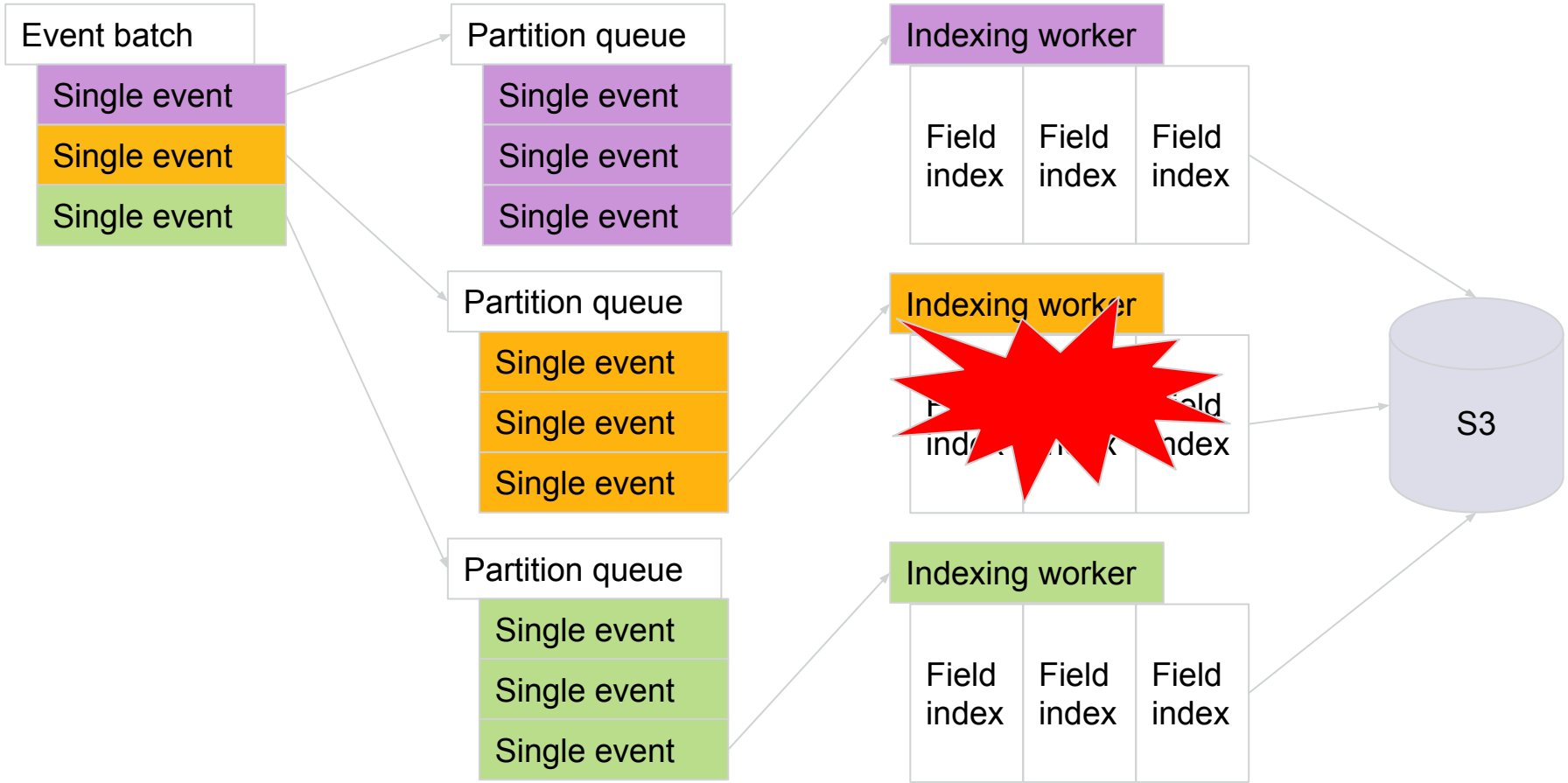
- If it breaks, everything breaks.
- Running third party software = harder to debug/understand

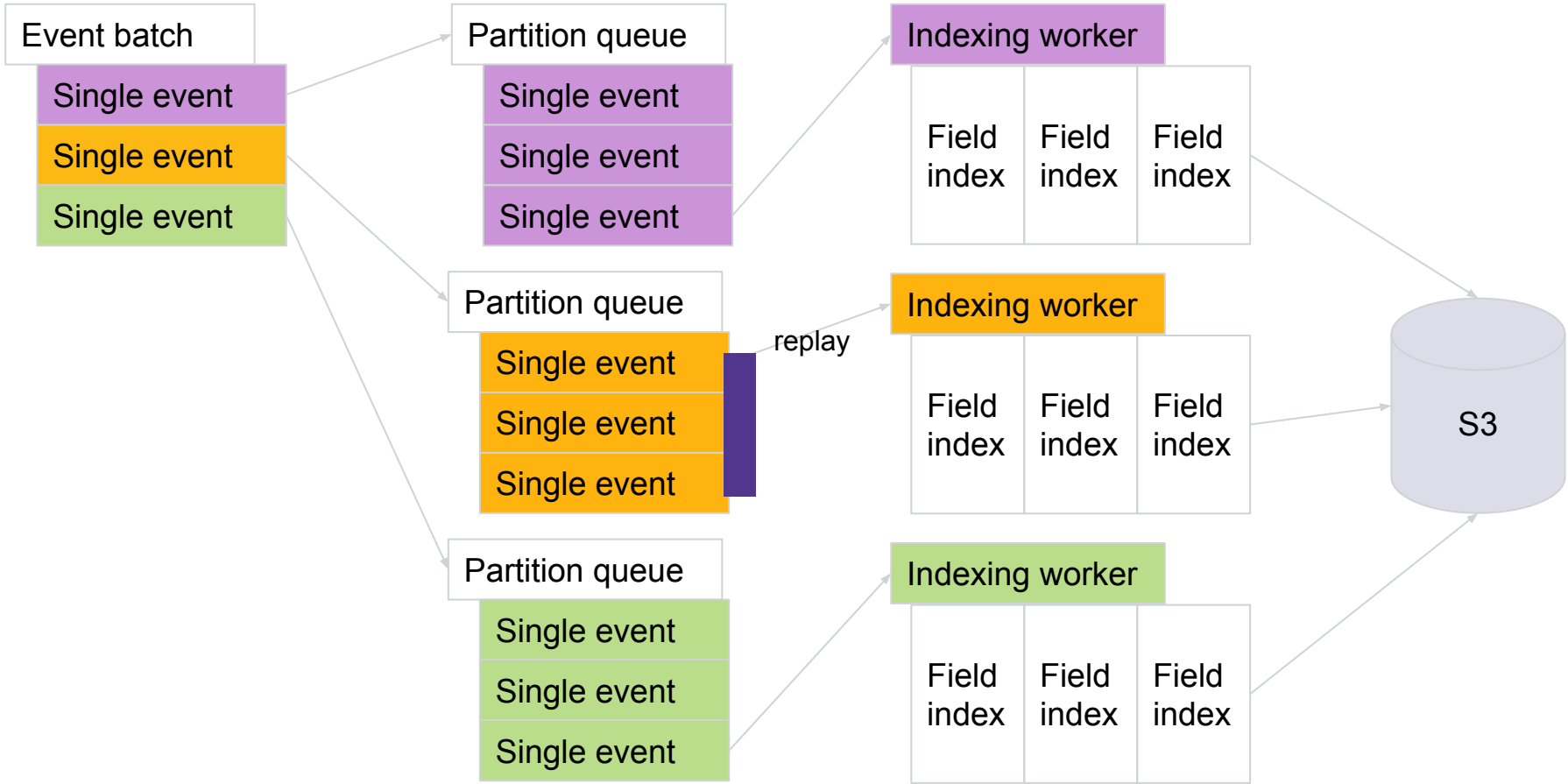
But here's how our use case is atypical

- We don't keep weeks or months of history
- We only typically read the last hour
- 1-2 main topics, not hundreds
- Self-managed partition allocation (~100 partitions)
- High throughput per partition (50k msgs/sec/partition)
- No ksql
- No librdkafka; pure Go Shopify Sarama (Shopify team, we owe you a drink)


Stateless request processing
Stateful data storage
Internet
Customer interface
Customer telemetry
Management and data API
honeycomb.io
Marketing site
ui.honeycomb.io
api.honeycomb.io
WordPress
Doodle
Poodle
Basset (cron)
Triggers & SLOs
Shepherd
Query results
Customer metadata
Customer metadata
Ingest
Ingest
Amazon S3
• Secondary storage
• Query results
MySQL
Kafka
Retriever-mutation
Queries
Service discovery
Orchestration
Secondary storage
Segment processor
Retriever
Beagle
Zookeeper
Bastion hosts
(internal network entry point)
https://github.com/egonelbre/gophers (CC0)


Event batch
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Indexing worker
Field index
Field index
Field index
Indexing worker
Field index
Field index
Field index
Indexing worker
Field index
Field index
Field index
S3


Event batch
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Indexing worker
Field index
Field index
Field index
Indexing worker
Indexing worker
Field index
Field index
Field index
S3


Event batch
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
replay
Indexing worker
Field index
Field index
Field index
Indexing worker
Field index
Field index
Field index
Indexing worker
Field index
Field index
Field index
S3

Identifying & solving bottlenecks

What happens when the system exceeds constraints?

Per-producer limits

- Least outstanding requests LB for ingest
- Tune batch sizes (MB, seconds), queue depths, etc
- Guard against OOMs
- Avoid persistently bad partitions
- Allocate load between partitions

Broker limits

- CPU
  - Use Graviton (ideally im4gn/is4gen)
- On-disk storage
  - Use smaller NVMe for predictable latency
  - Use tiered storage for bulk, less frequently accessed
    - DIY tiering w/ writeback cache does not work. We tried it.
  - We do *not* recommend use of EBS volumes for scaling out (latency, cost)
- Auto-balancing
- Horizontal scale-out (if needed)

Per-consumer limits

- Annoying: Kafka limits consumer-broker BW regardless of distinct partitions.
- Run more brokers (or map consumers:partitions 1:1) if all else fails.
- Watch out for consumer group rebalancing
    - Rolling restarts are a good SRE practice everywhere EXCEPT here
    - (but this advice may change with sticky consumer group assignment)
- You are only as good as your offset commit

Optimizations to consider

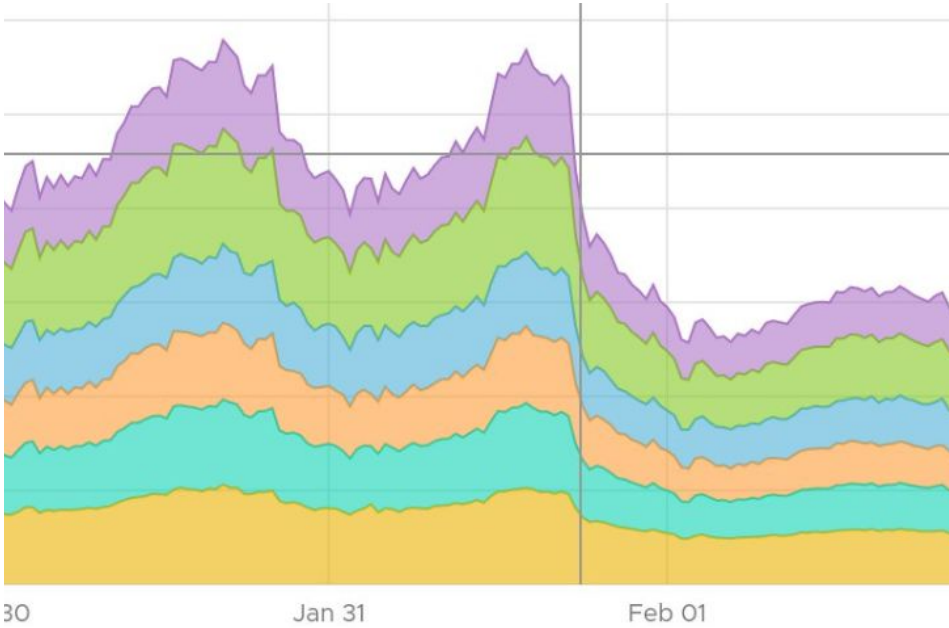
How to get the most out of your cluster

Consider not running brokers yourself

- Confluent Cloud, Google Cloud Pub/Sub, Amazon Kinesis, etc.
- **Run. Less. Software.**

Use zstd. Seriously.

- CPU is ~cheaper than network.
- 20%+ savings on bandwidth vs snappy



Fully utilise Kafka's replication

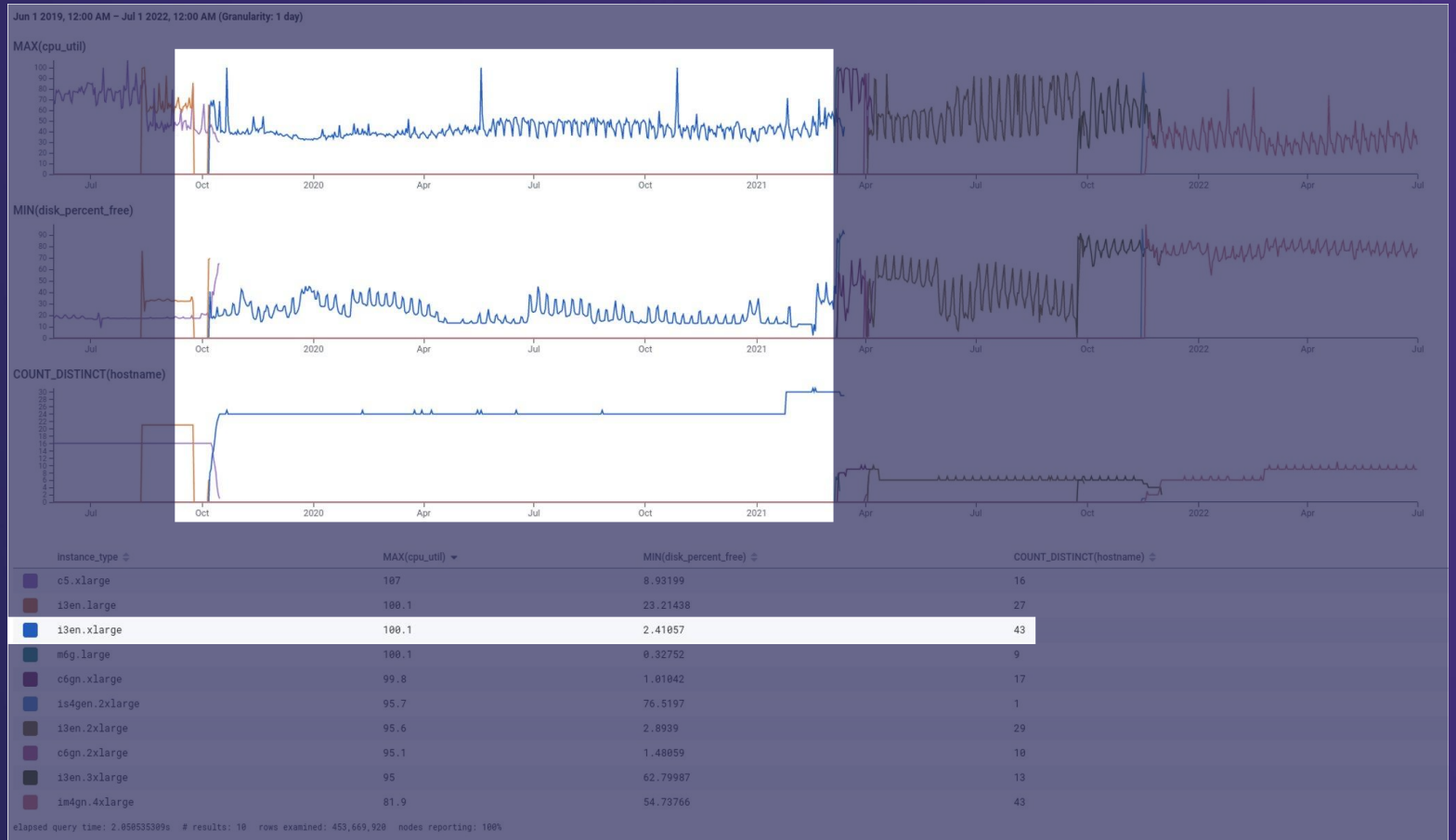
- DTAZ = $$$$
- Don't re-copy data between AZs; read from followers
- Don't pay for more durability than you need (Kafka already provides R=3)

Use the most efficient base you can

Hardware

- Be careful of unknown unknowns
    - There may be dimensions you're unaware of.
- Burst balances create metastable systems
    - There may be dimensions you're unaware of.

Finding the right way to migrate Kafka



instance_type	MAX(cpu_util)	MIN(disk_percent_free)	COUNT_DISTINCT(hostname)
c5.xlarge	107	8.93199	16
i3en.large	100.1	23.21438	27
i3en.xlarge	100.1	2.41057	43
m6g.large	100.1	0.32752	9
c6gn.xlarge	99.8	1.01042	17
is4gen.2xlarge	95.7	76.5197	1
i3en.2xlarge	95.6	2.8939	29
c6gn.2xlarge	95.1	1.48059	10
i3en.3xlarge	95	62.79987	13
im4gn.4xlarge	81.9	54.73766	43

elapsed query time: 2.050535309s # results: 10 rows examined: 453,669,928 nodes reporting: 100%

Our month of Kafka pain

Longtime Confluent Kafka users

First to use Kafka on Graviton2 at scale

Changed multiple variables at once

- move to tiered storage
- i3en → c6gn
- AWS Nitro

Read more: go.hny.co/kafka-lessons

Unexpected constraints

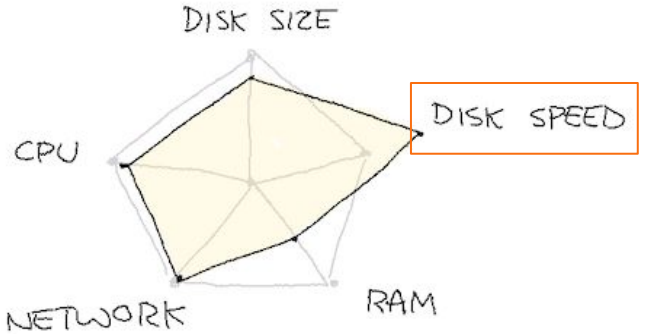
We thrashed multiple dimensions.

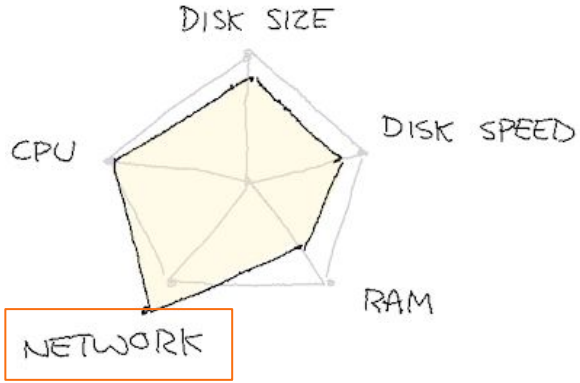
We tickled hypervisor bugs.

We tickled EBS bugs.

Burning our people out wasn't worth it.

But we were finally able to move forward in Dec 2021 with im4gn!





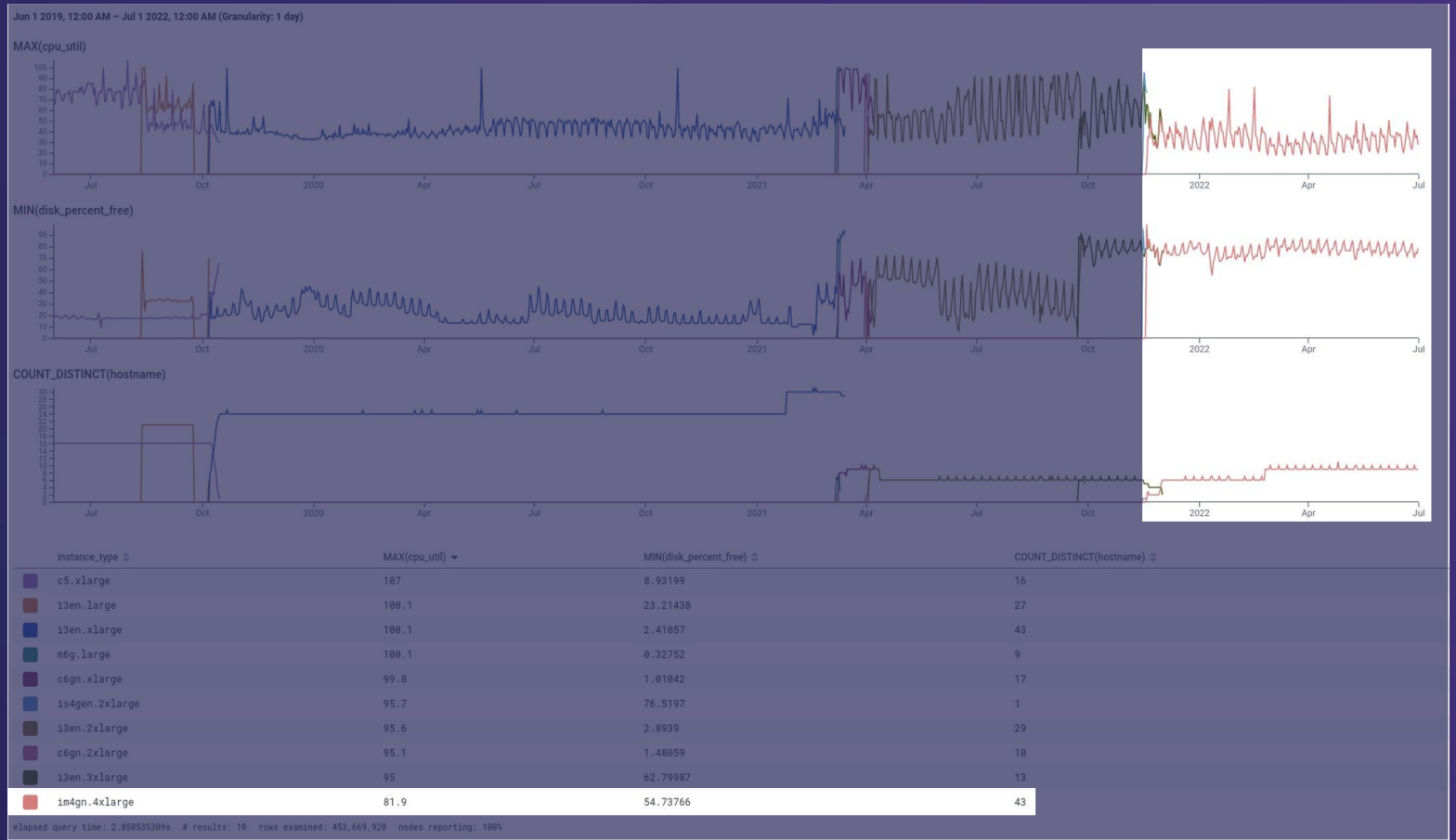
Read more: go.hny.co/kafka-lessons

Finding the right way to migrate Kafka

instance_type	MAX(cpu_util)	MIN(disk_percent_free)	COUNT_DISTINCT(hostname)
c5.xlarge	107	8.93199	16
i3en.large	100.1	23.21438	27
i3en.xlarge	100.1	2.41057	43
m6g.large	100.1	0.32752	9
c6gn.xlarge	99.8	1.01042	17
is4gen.2xlarge	95.7	76.5197	1
i3en.2xlarge	95.6	2.8939	29
c6gn.2xlarge	95.1	1.48059	10
i3en.3xlarge	95	62.79987	13
im4gn.4xlarge	81.9	54.73766	43

elapsed query time: 2.050535309s # results: 10 rows examined: 453,669,920 nodes reporting: 100%

Finding the right way to migrate Kafka



instance_type	MAX(cpu_util)	MIN(disk_percent_free)	COUNT_DISTINCT(hostname)
c5.xlarge	107	8.93199	16
i3en.large	100.1	23.21438	27
i3en.xlarge	100.1	2.41057	43
m6g.large	100.1	0.32752	9
c6gn.xlarge	99.8	1.01042	17
is4gen.2xlarge	95.7	76.5197	1
i3en.2xlarge	95.6	2.8939	29
c6gn.2xlarge	95.1	1.48059	10
i3en.3xlarge	95	62.79987	13
im4gn.4xlarge	81.9	54.73766	43

elapsed query time: 2.050535309s # results: 10 rows examined: 453,669,920 nodes reporting: 100%

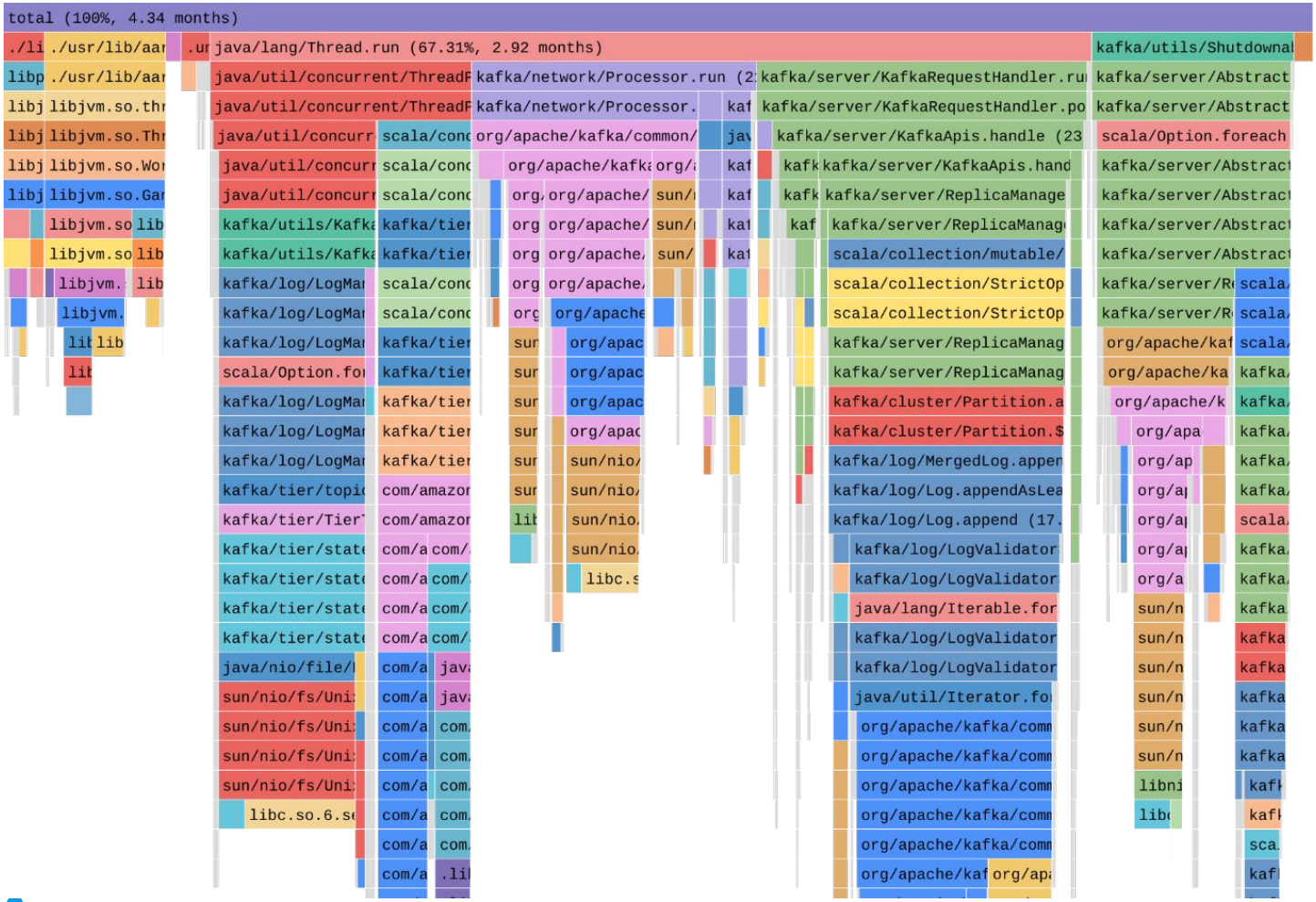
Use the most efficient base you can

Hardware

- Be careful of unknown unknowns
  - There may be dimensions you're unaware of.
- Burst balances create metastable systems
  - There may be dimensions you're unaware of.

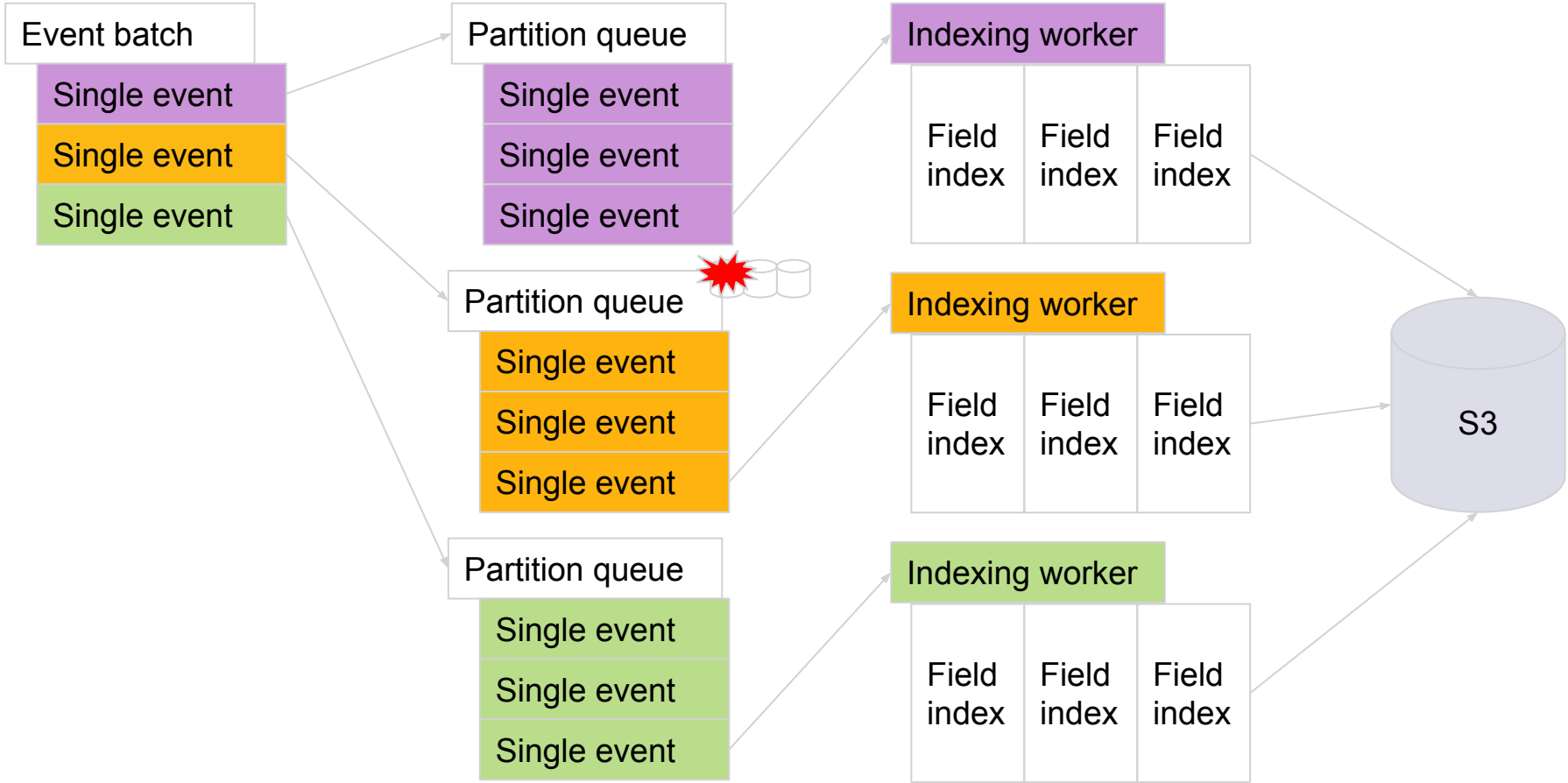
Software

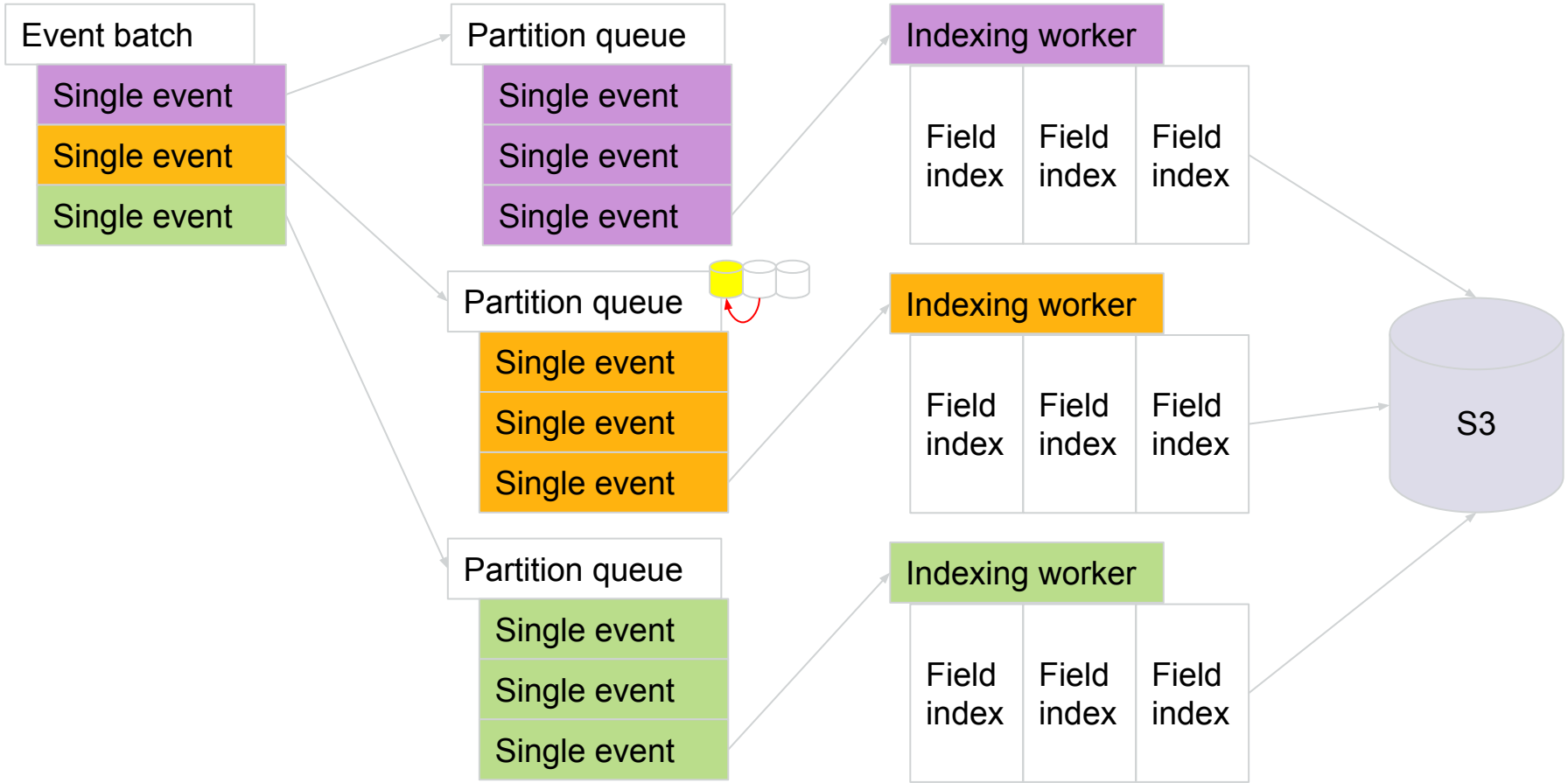
- Profile, profile, profile.
- Use Corretto JVM, not GetOpenJDK
- Use a well-tuned GC algorithm
- Upgrade your JNI deps (eg Zstd)
- Replace Java crypto libraries with AWS Corretto Crypto Provider



Continuously chaos test your DR strategy

- Weekly consumer and broker replacement to verify cold start
- Remember: an untested backup is *not* a restore.
- Leave plenty of room for unexpected scenarios
- In streaming, headroom = *time* before pear-shaped


Event batch
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Indexing worker
Field index
Field index
Field index
Indexing worker
Field index
Field index
Field index
Indexing worker
Field index
Field index
Field index
S3


Event batch
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Partition queue
Single event
Single event
Single event
Indexing worker
Field index
Field index
Field index
Indexing worker
Field index
Field index
Field index
Indexing worker
Field index
Field index
Field index
S3

Observing streaming systems

How we make sure everything is working correctly

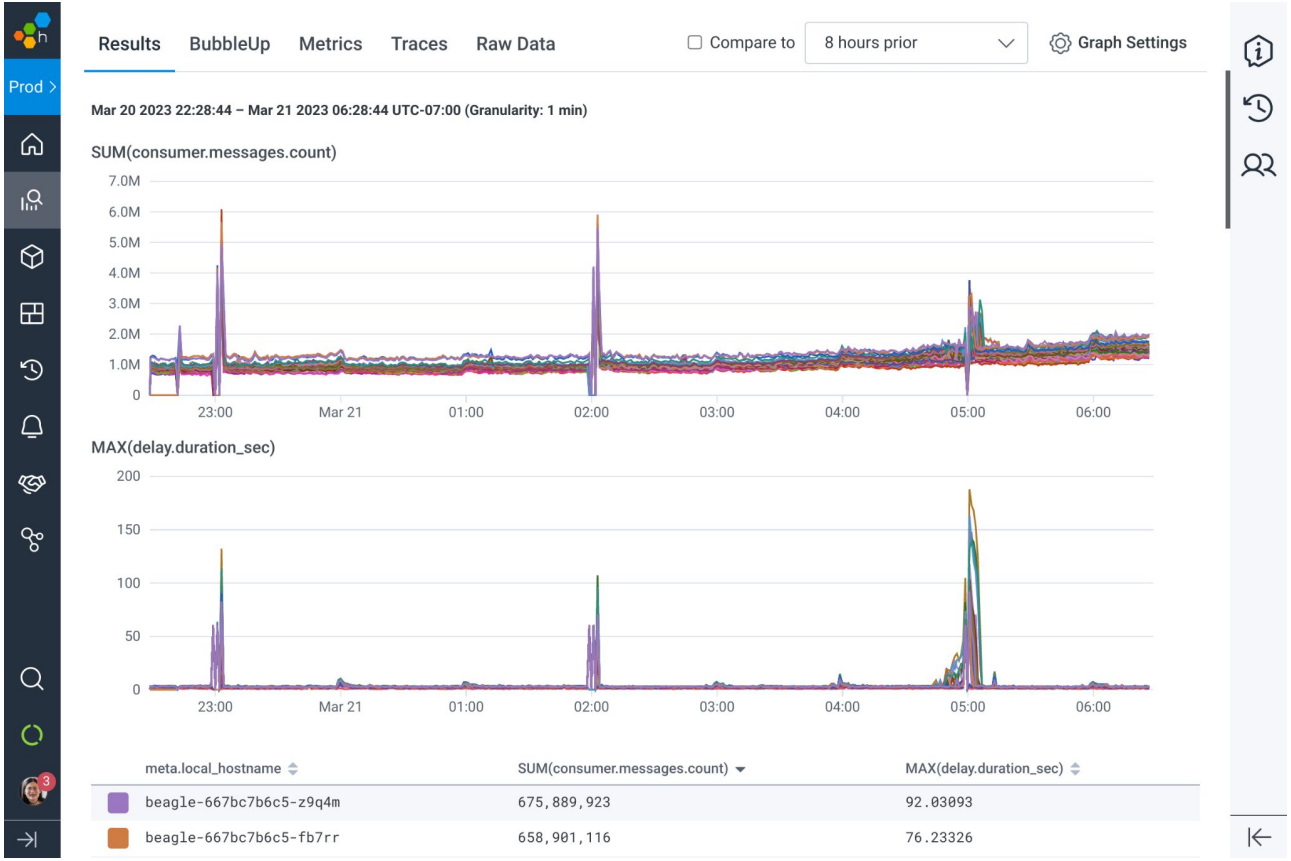
Application vs system observability

App level: trace spans & links

- Periodic trace spans per consumer ("tick")
- Heavily sampled produce requests
- Trace links between consume & produce

System level: broker metrics

- Basics: Msgs/sec, CPU, URP, Disk
- Advanced: GC, Network, Controller, Rebalancing



← Trace 3224b822936253f854e89a84bdf7bd4e at 2023-03-21 03:27:40 Rerun

Search spans

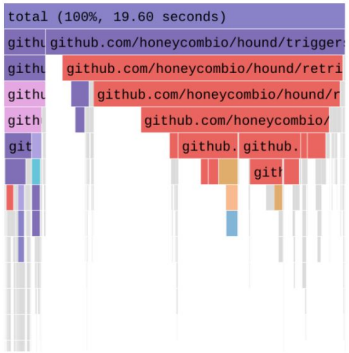
0 spans with errors Fields

name	service.name	0s 20s 40s 60.75s
1K+ tick	beagle	60.753s
1 getSLOsForDataset	beagle	1.900ms
11 getSLOsForDataset	beagle	12.97ms
15 getSLOsForDataset	beagle	17.96ms
worker	beagle	58.113s
5K+ getSLOsForDataset	beagle	644.1ms
worker	beagle	58.095s
worker	beagle	57.442s
worker	beagle	57.415s
worker	beagle	57.414s
worker	beagle	57.404s
worker	beagle	57.396s
worker	beagle	57.388s
worker	beagle	57.380s
worker	beagle	57.354s
worker	beagle	57.354s
worker	beagle	57.354s

beagle >

tick

Latencies | Profile Self | Profile Children



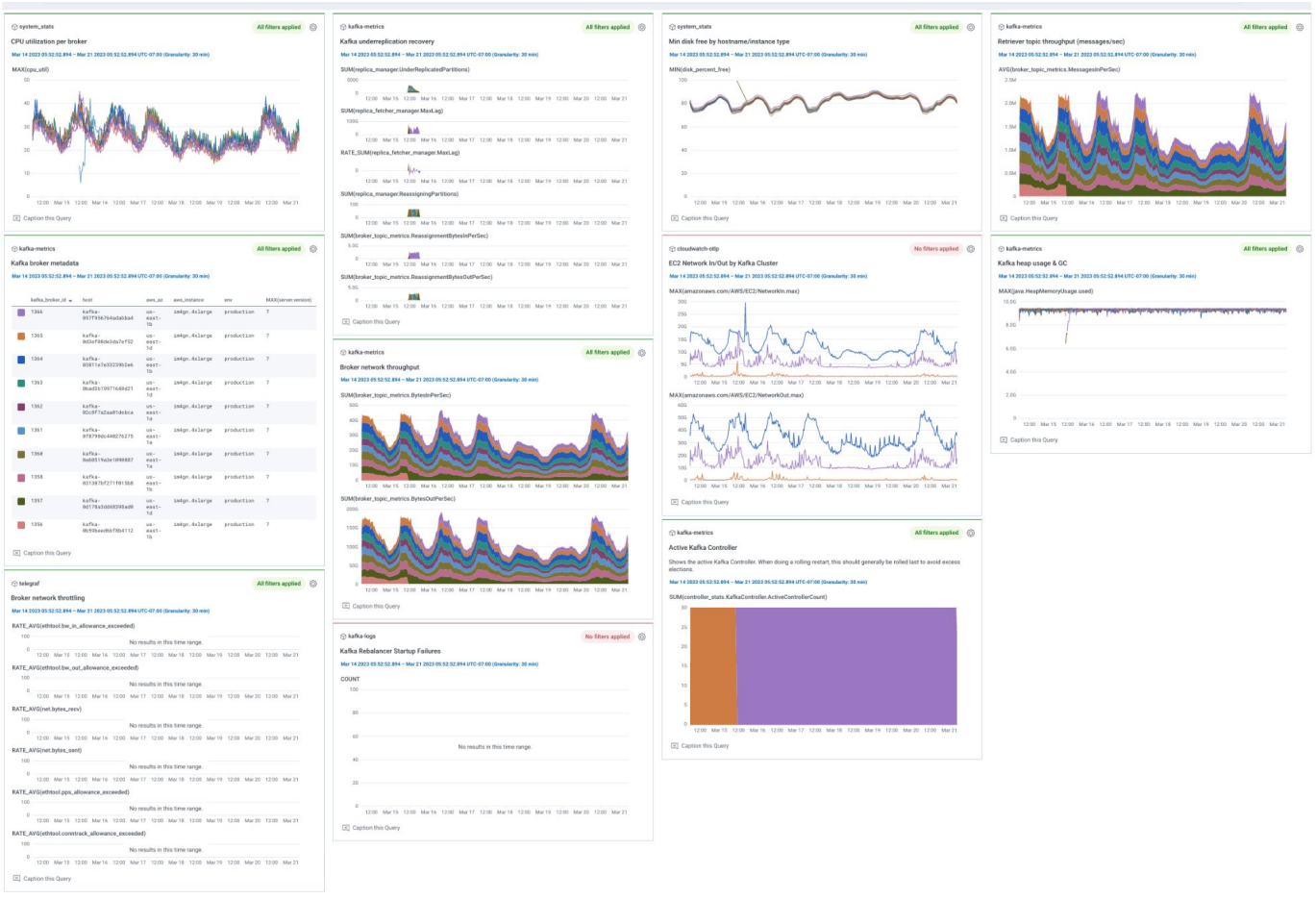
Fields | Span events (128) | Links (0)

Filter fields and values in span

str Timestamp

2023-03-21T10:27:40.010006877Z

flt consumer.generation_duration_sec



Application vs system SLOs

App level: many SLOs

- SLOs on producer write success/latency
- SLOs on consumer freshness per-consumer
- Implied SLO for durability (~never lose data)

System level: No SLOs.

- Because we use the producer/consumer views instead!

Future work

Where we want to go next

tl;dr better balancing & auto-healing

Broker self-balancing sometimes sticks

Leadership imbalance causes CPU anomalies/perf pain

Long-term partition imbalance causes operational pain

Short-term spikes on specific partitions from new customers cause pain

Kafka is the beating heart, but should not produce toil.

Non-goal: k8s. None of these problems are things k8s would solve for us.

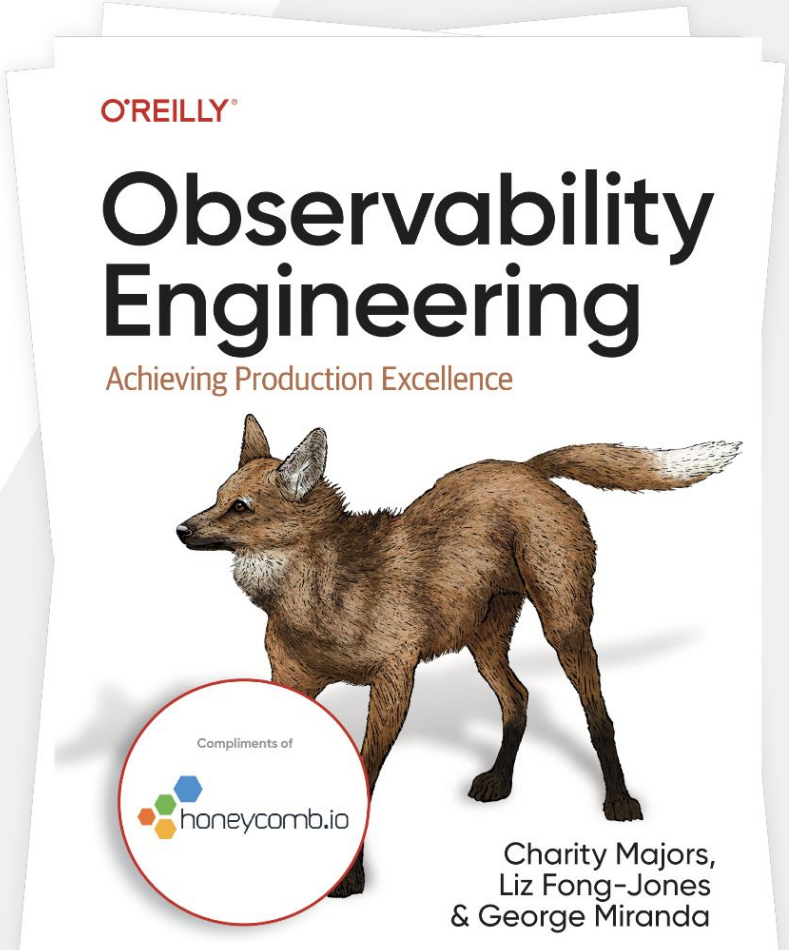
Visit our booth!

hny.co/srecon23-americas

@lizthegrey & @rainofterra

Observability Engineering

Get our new book, free!



@lizthegrey

Questions?

www.honeycomb.io