Getting Started with Observability for Chaos Engineering

Illustrations by @emilywithcurls!

honeycomb.io

Shelby Spees

Developer Advocate at Honeycomb.io

@shelbyspees

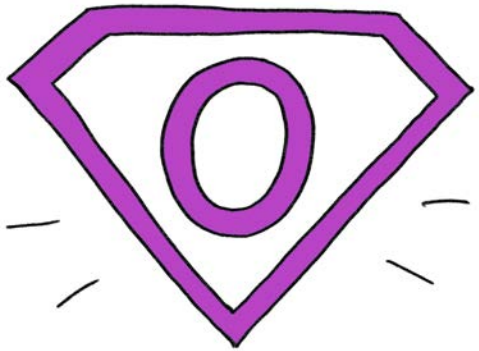
Chaos at 3pm, not 3am

Experimenting in the dark is risky

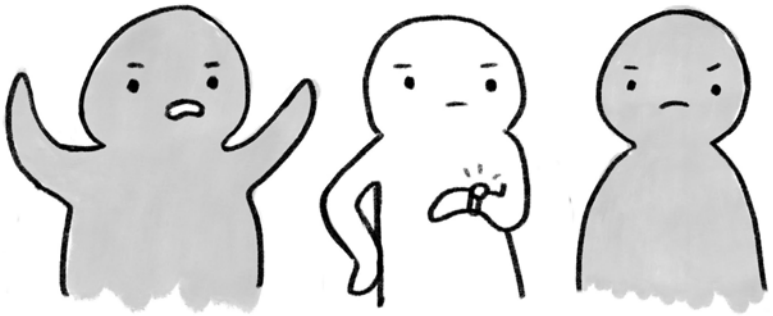
We need observability!

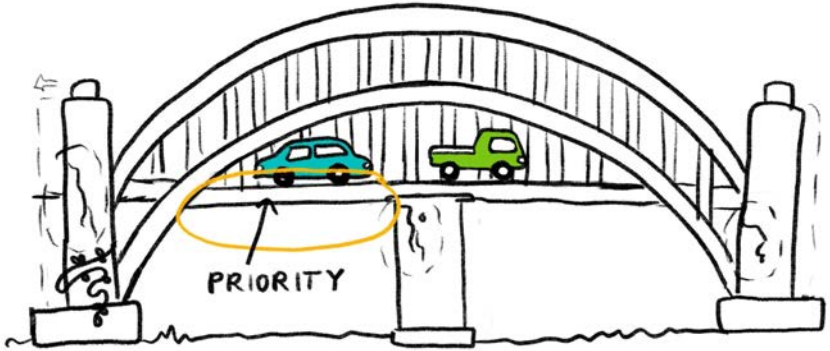
What is **observability**?

> in software, the ability to understand and explain any state a system can get into, no matter how novel or bizarre, without deploying new code



Learn more from chaos

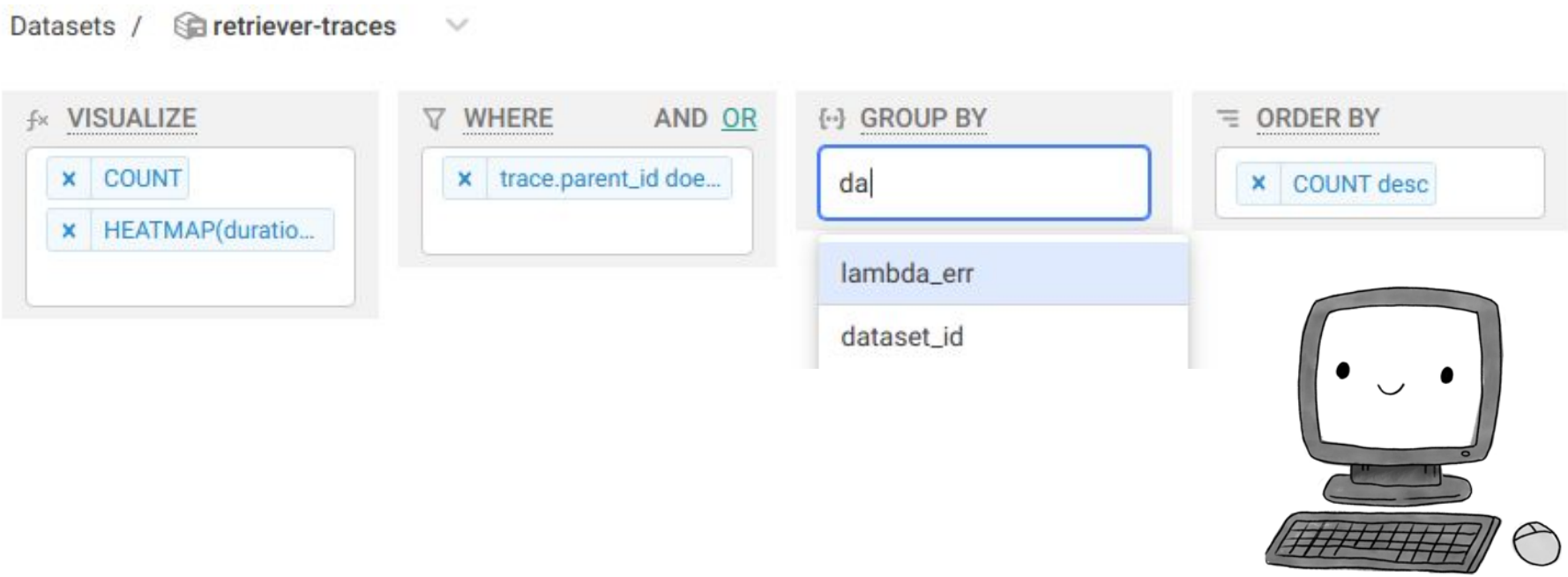




Better account for business risks

Lots of observability tooling out there

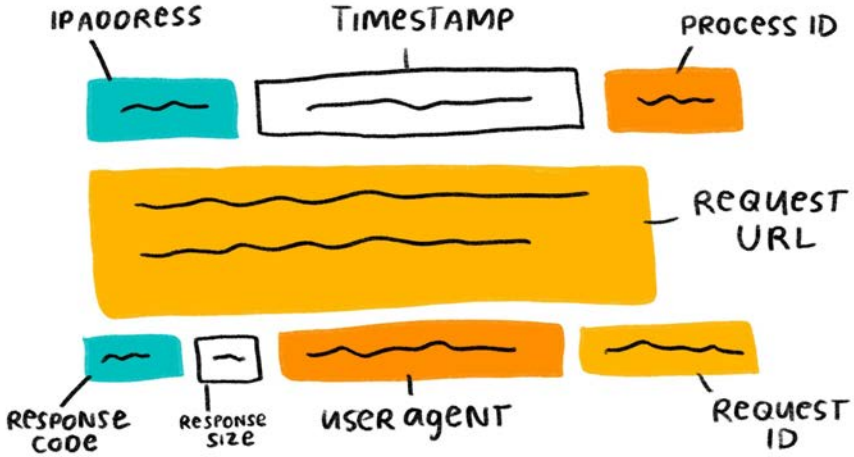
Honeycomb is a system analytics tool



It works by ingesting your system telemetry

What is **telemetry**?

> data gathered about the state of your system at runtime, often sent to external tooling for monitoring or analysis
>
> *tele-* (distance) + *-metry* (measure)



In the form of structured events

Capture

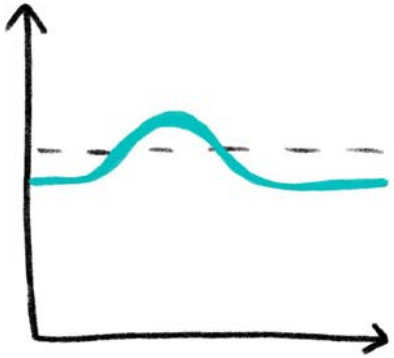
- runtime context
- benchmarking data

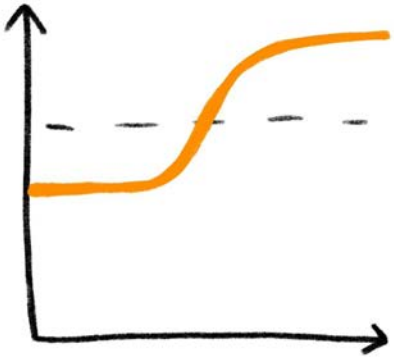
Enable

- trace visualizations
- asking novel questions


```
{
  "timestamp": "2018-03-20T00:47:25.339Z",
  "app.interesting_thing": "banana",
  "duration_ms": 772.446625,
  "handler.name": "main.hello",
  "handler.pattern": "/hello/",
  "handler.type": "http.HandlerFunc",
  "meta.beeline_version": "0.2.0",
  "meta.local_hostname": "cobbler.local",
  "meta.span_type": "root",
  "meta.type": "http_request",
  "name": "main.hello",
  "request.content_length": 0,
  "request.header.user_agent": "curl/7.54.0",
  "request.host": "localhost:8080",
  "request.http_version": "HTTP/1.1",
  "request.method": "GET",
  "request.path": "/hello/",
  "request.remote_addr": "127.0.0.1:60379",
  "response.status_code": 200,
  "service_name": "sample app",
  "trace.span_id": "9e4fe697-3ea9-48c9-b673-72d7ddf118a6",
  "trace.trace_id": "b64c89a9-7671-4732-bef1-9ef75ab831f6"
}
```

We index on individual fields for fast querying



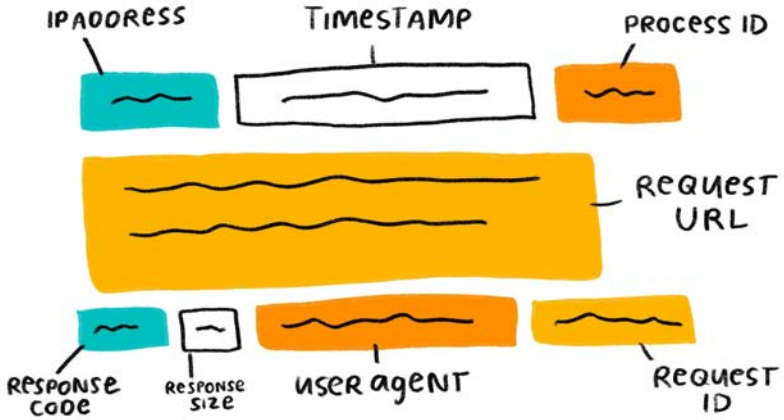


How do we know we're doing a good job?

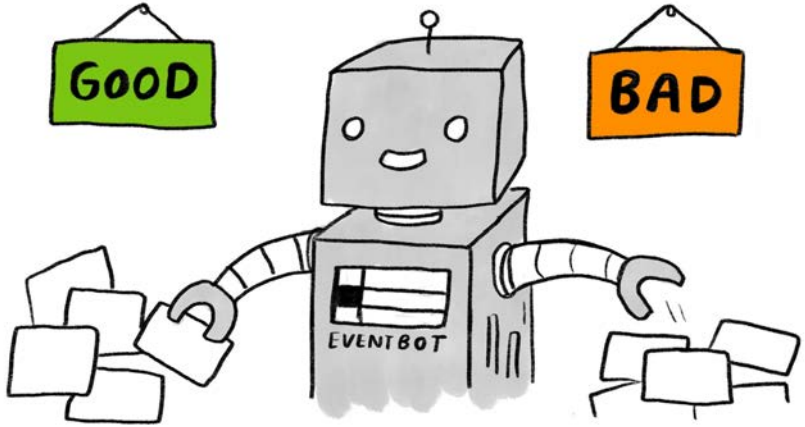
Service Level Objectives (SLOs)

The API for your engineering team

SLOs are a common language



Think in terms of events in context



SLI: for each event, good or bad?

Use a window and target percentage

of all eligible events in 30-day window
X% should meet the threshold for “good”

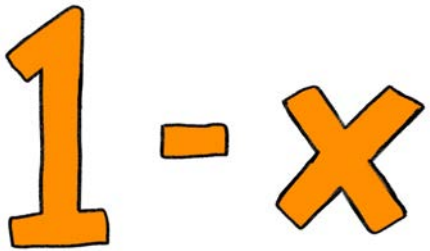
Example SLO

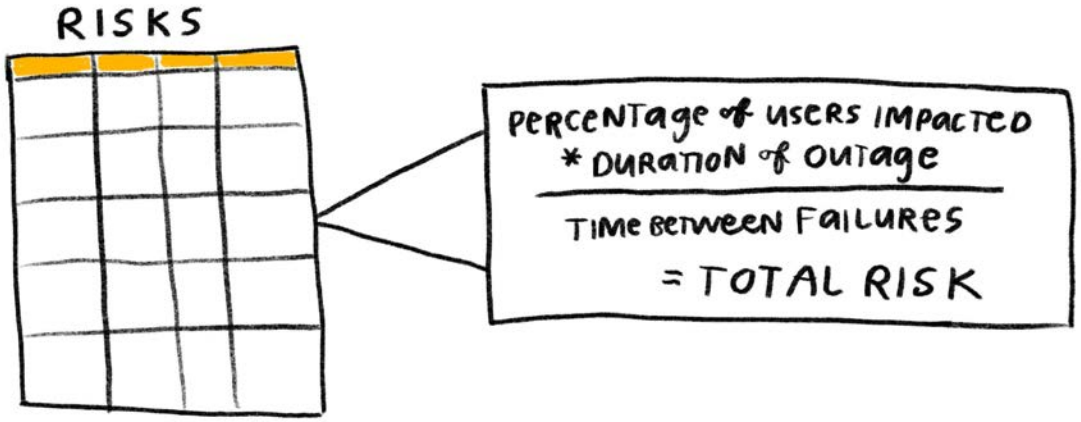
99.9% of Home Page loads in the past 30 days were fast enough

A good SLO barely keeps users happy

Error budget: allowed unavailability

99.9% SLO target × 1 million requests/month
= 1000 requests can fail/month





When is it okay to take risks?

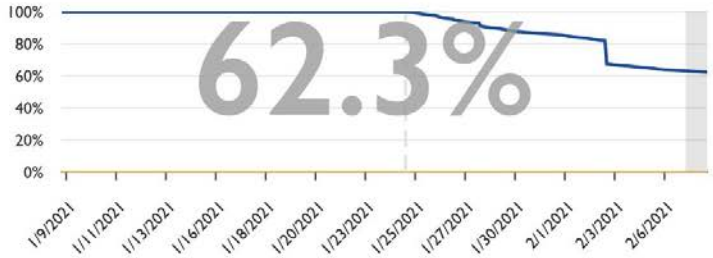
When is it *not* okay?

Historical SLO Compliance

For each day of the past 30, how often this SLI has succeeded over the preceding 30 days.

Budget Burndown

How much of the error budget remains after the last 30 days. Starts at 100% and burns down.



Ingest Latency SLO and Error Budget

Defining an SLO

Better data for better goal-setting

Instrumenting ingest code

```
func (a *App) postBatch(r *Request, w http.ResponseWriter) error {
    ctx := r.Context()

    instr.AddField(ctx, "batch", true)
    instr.AddFieldToTrace(ctx, "event_handler", "http")
    . . .
```

Instrumenting ingest code

```
. . .
instr.AddField(ctx, "raw_size_bytes", size)
instr.AddField(ctx, "batch_total_datapoints", len(batchedEvents))
. . .
```

Their fault or our fault?

```
. . .
if len(batchedEvents) == 0 {
    instr.AddField(ctx, "dropped", "their fault")
    instr.AddField(ctx, "drop_reason", "empty batch")
    return apierr.MsgRequestBodyEmpty
}
. . .
```

Capturing error context

```
. . .
instr.AddField(ctx, "dropped", "our fault")
instr.AddField(ctx, "drop_reason", errResp.Error())
instr.AddField(ctx, "status", errResp.Status)
. . .
```

Defining our SLI: eligible events

```
NOT(STARTS_WITH($request.header.user_agent, "collectd")),
NOT(STARTS_WITH($app.err, "deprecated endpoint")),
NOT(EQUALS($response.status_code, 401)),
NOT(EQUALS($response.status_code, 403)),
NOT(EQUALS($app.dropped, "their fault")),
EQUALS($request.endpoint, "batch"),
EQUALS($request.method, "POST"),
```

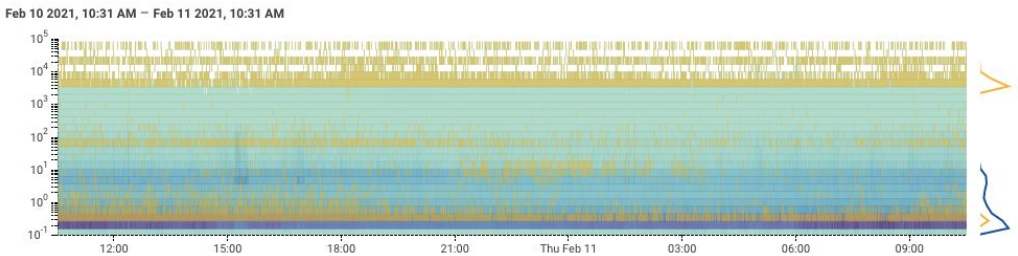
Defining our SLI: good events

```
EQUALS($response.status_code, 200)

normalized_duration =

    duration_ms / (batch_size > 500 ? batch_size : 500)


IF(normalized_duration < 5)
```

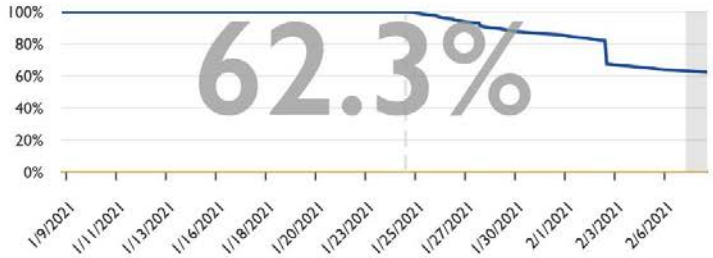


Historical SLO Compliance

For each day of the past 30, how often this SLI has succeeded over the preceding 30 days.

Budget Burndown

How much of the error budget remains after the last 30 days. Starts at 100% and burns down.



Ingest Latency SLO and Error Budget

Get Started with Observability

Prepare yourself for chaos

Choose an observability tool

Use OpenTelemetry!

Vendor-neutral OSS instrumentation framework

Java | C# | Go | JavaScript | Python | Rust | C++ | Erlang/Elixir | (and more!)

Auto-instrumentation for HTTP and gRPC

visit **OpenTelemetry.io**

Dip a toe in the water

Start with auto-instrumentation
(especially for tracing!)

Choose one app or service in the critical path

Send from your dev environment

Deploy a canary branch to a subset of prod

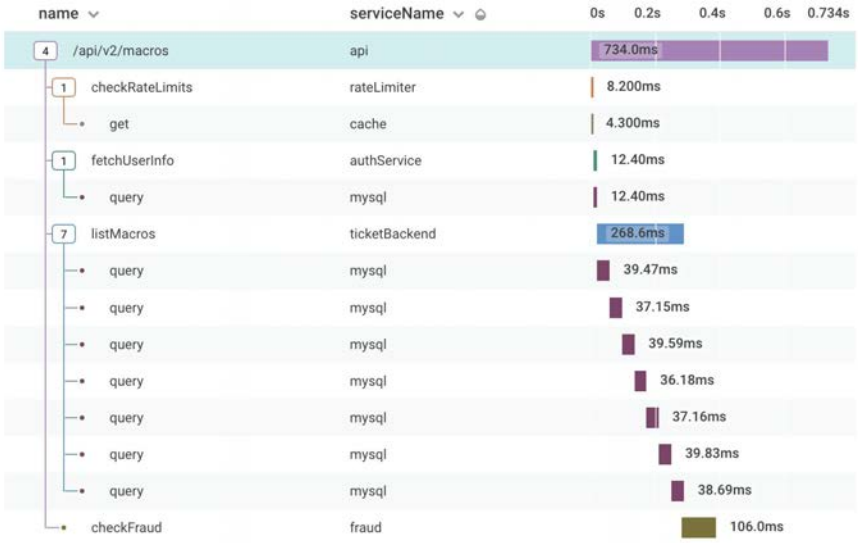
Iterating your instrumentation

Build on auto-instrumentation

Start adding custom fields in your code

Instrument where there's risk (read: change)

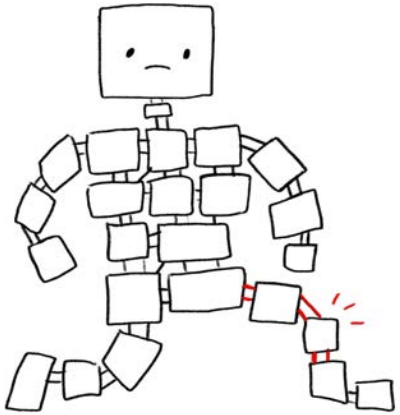
Set up distributed tracing



Experiment with Observability

Designing your experiment

- What's my hypothesis about the system?
- Can I retrace the failure and resolution?
- Test the telemetry too!



Celebrate successes *and* failures.

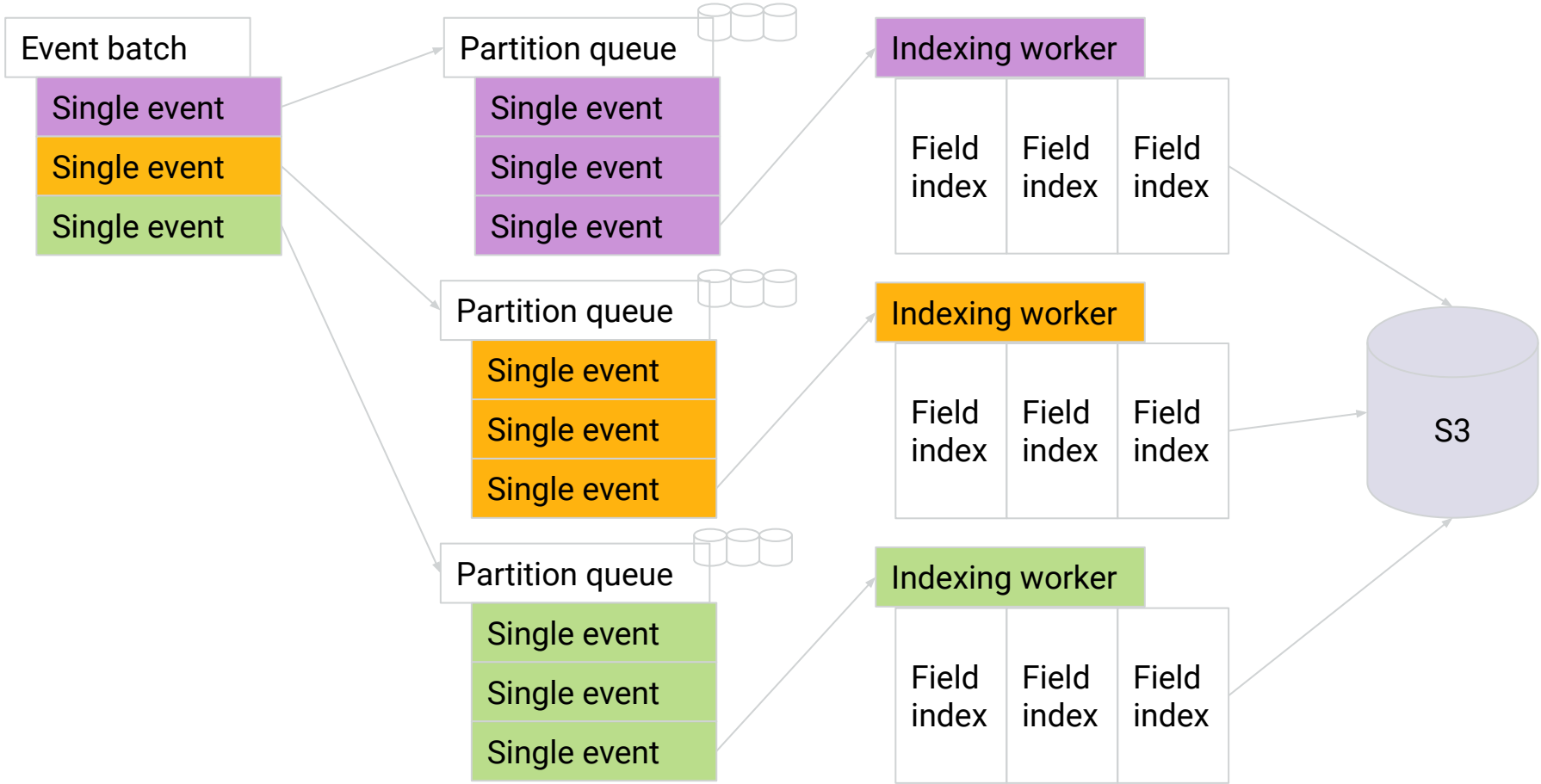
Reach out!

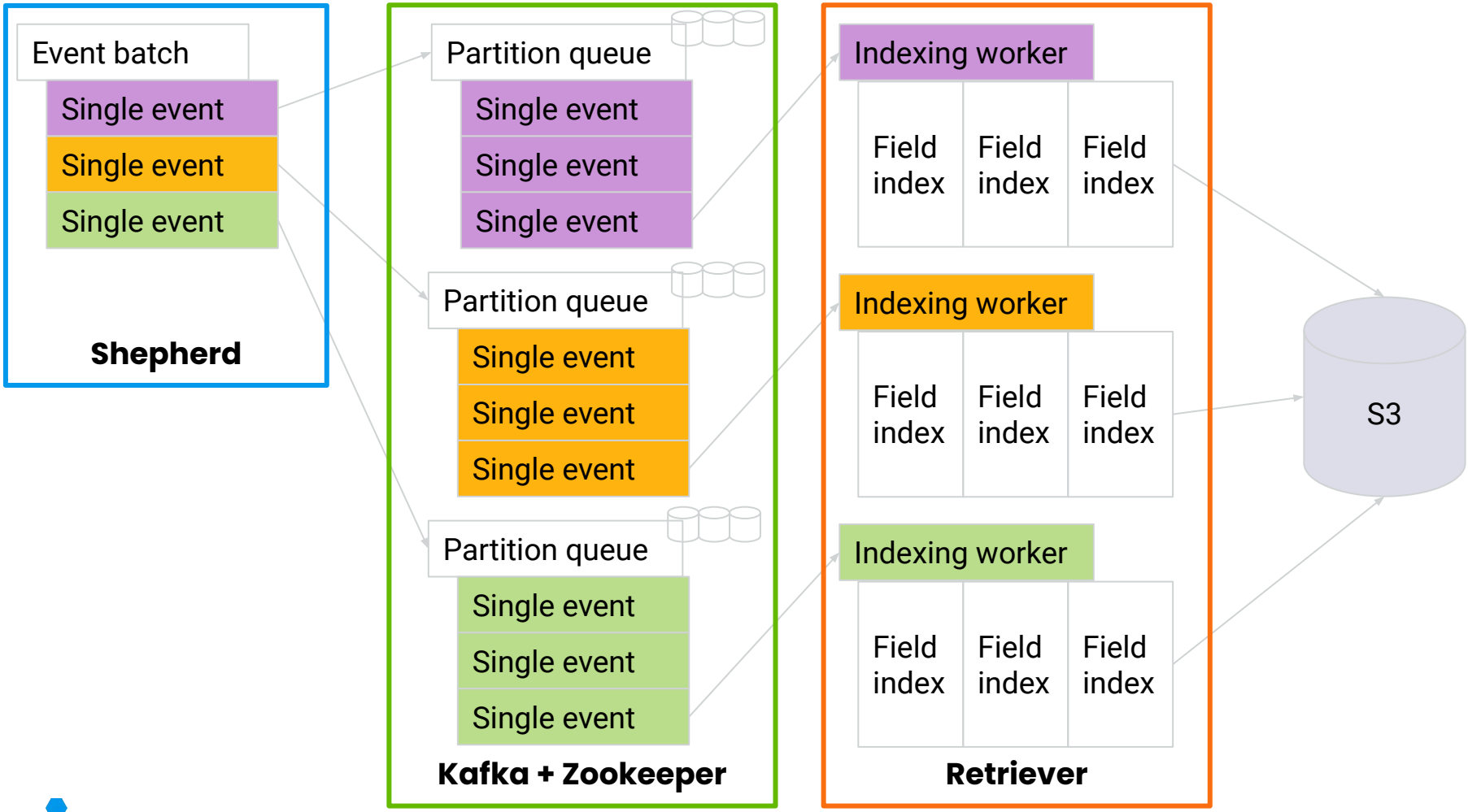
honeycomb.io/shelby

@shelbyspees

Questions?

Experimenting in Prod

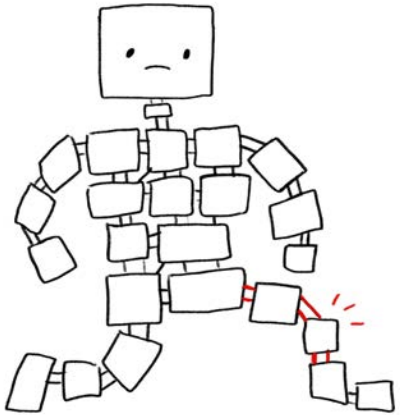




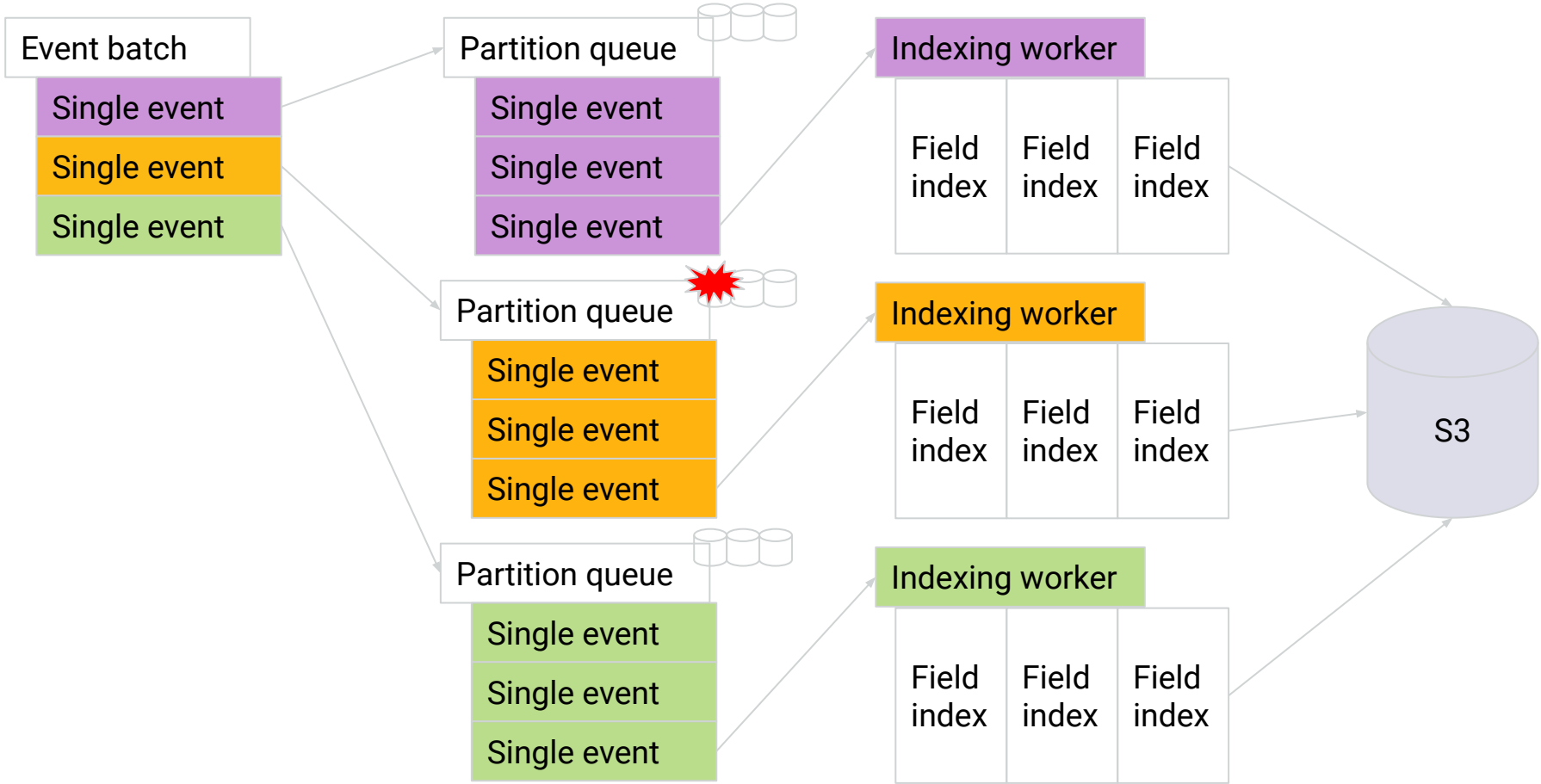
Infrequent changes.

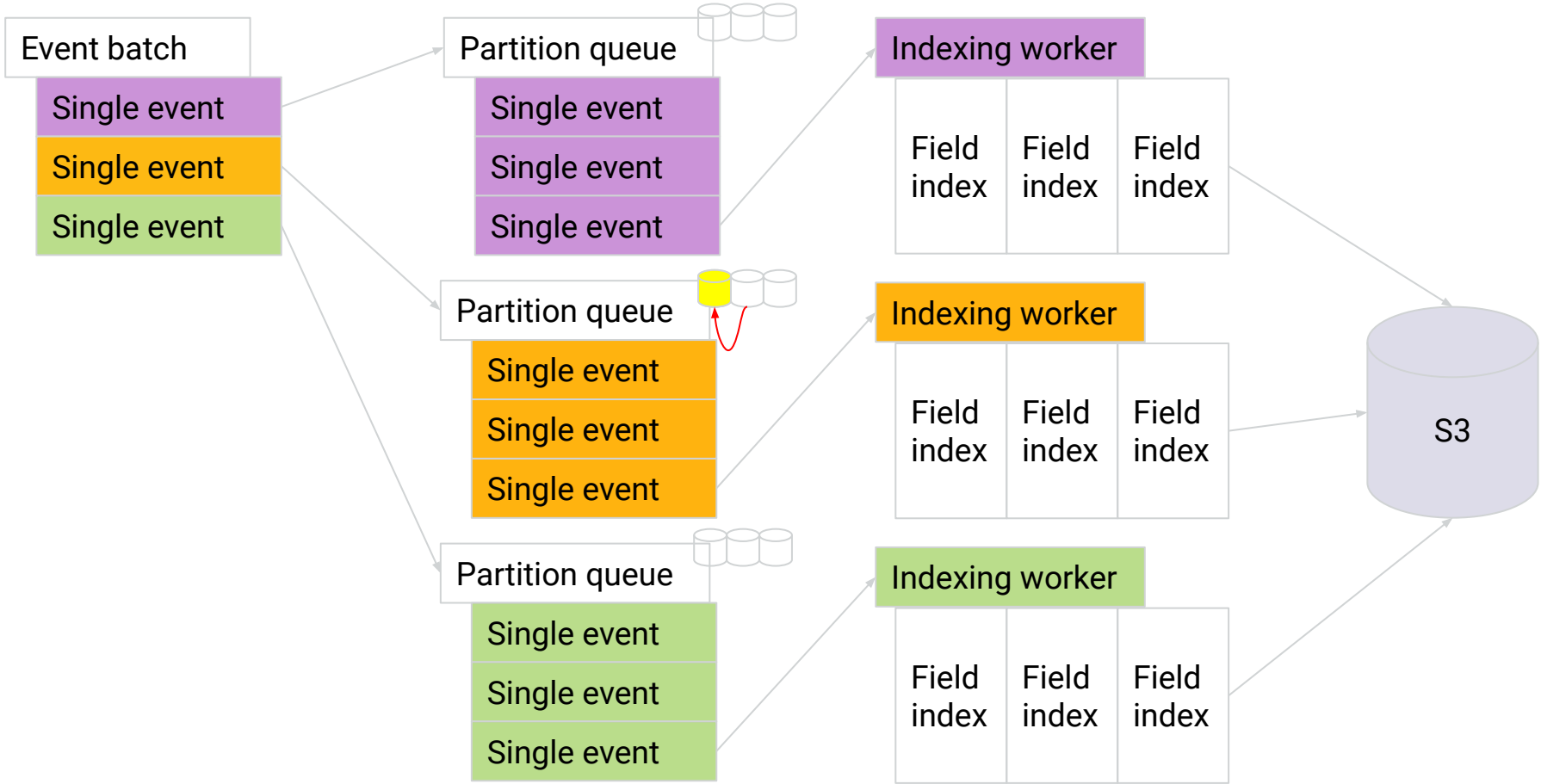
Long-running processes.

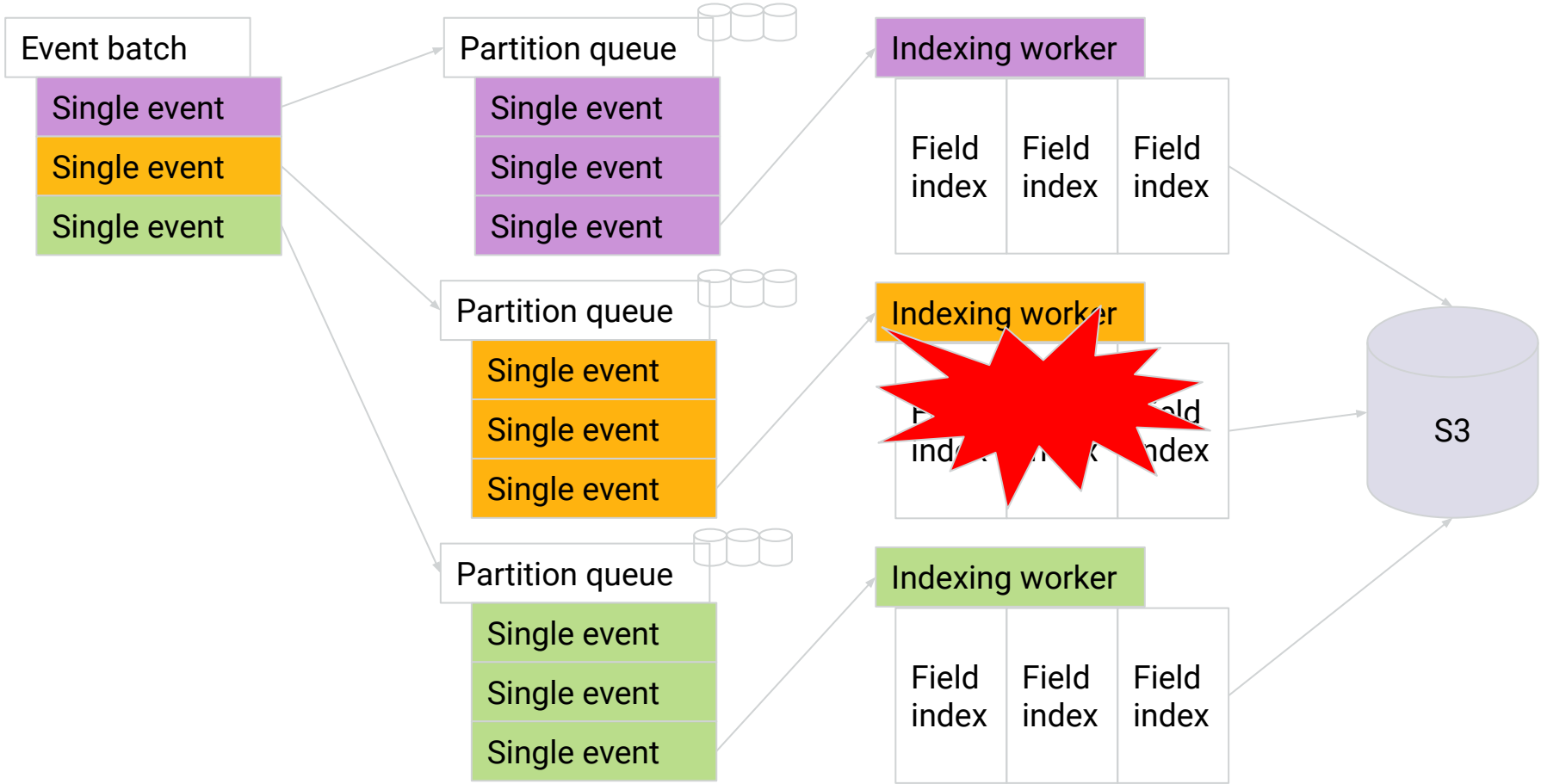
Data integrity and consistency.

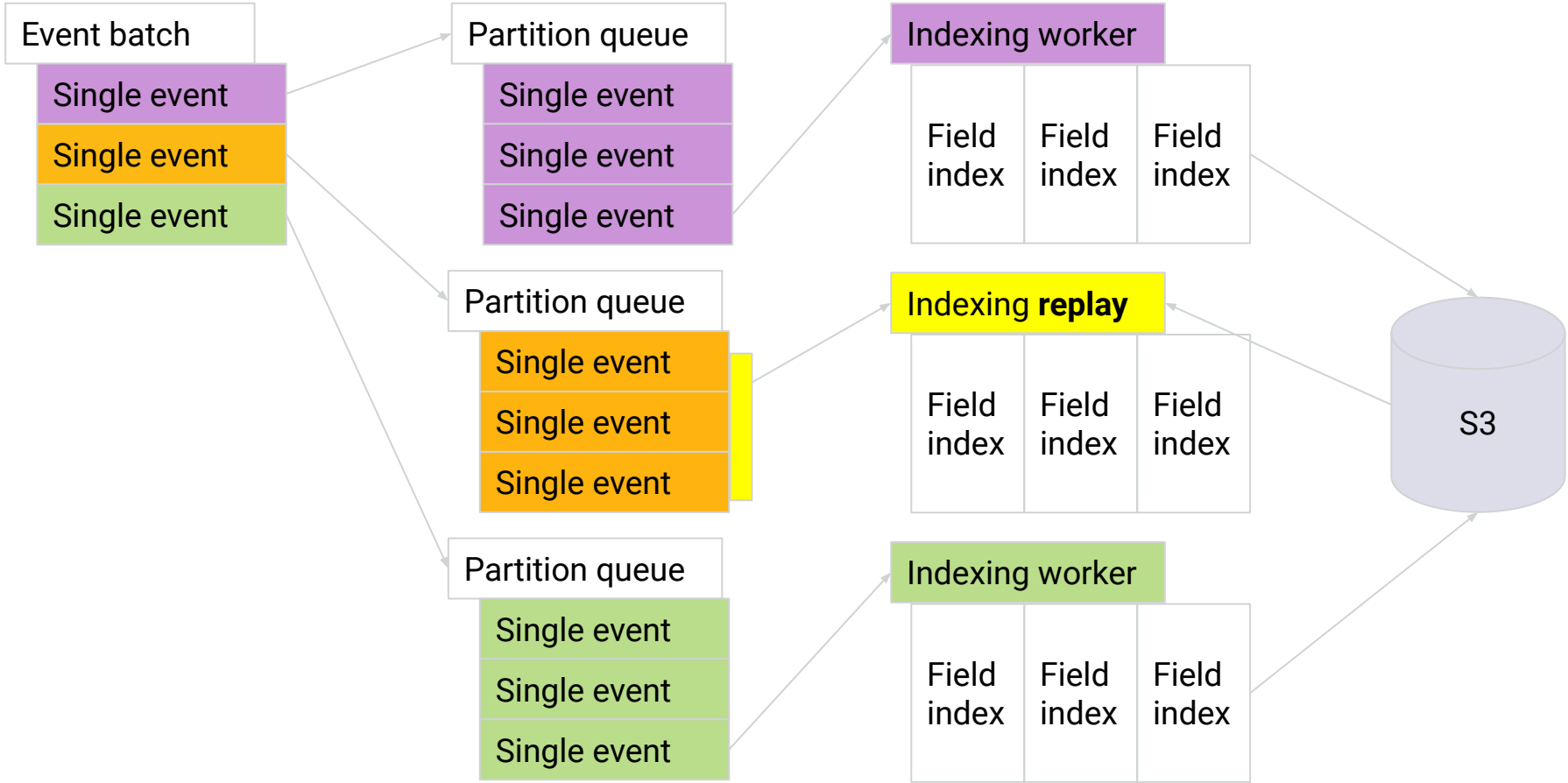


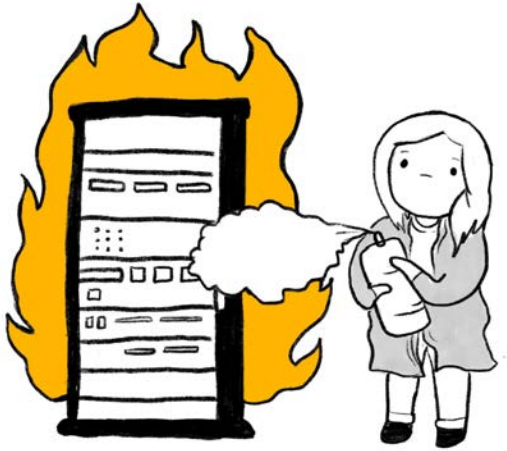
Delicate failover dances



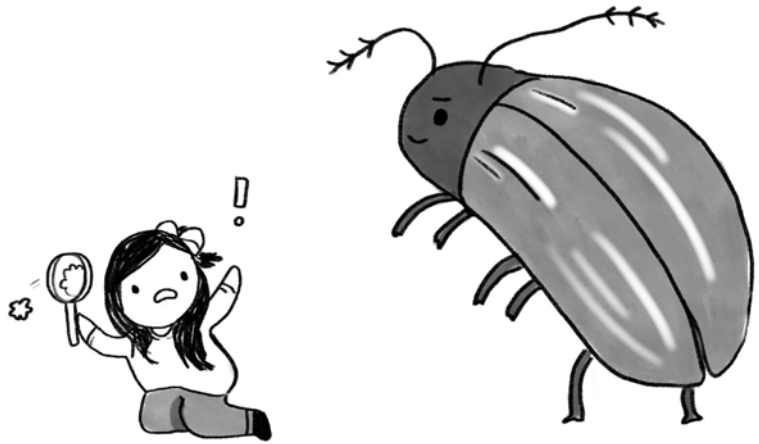




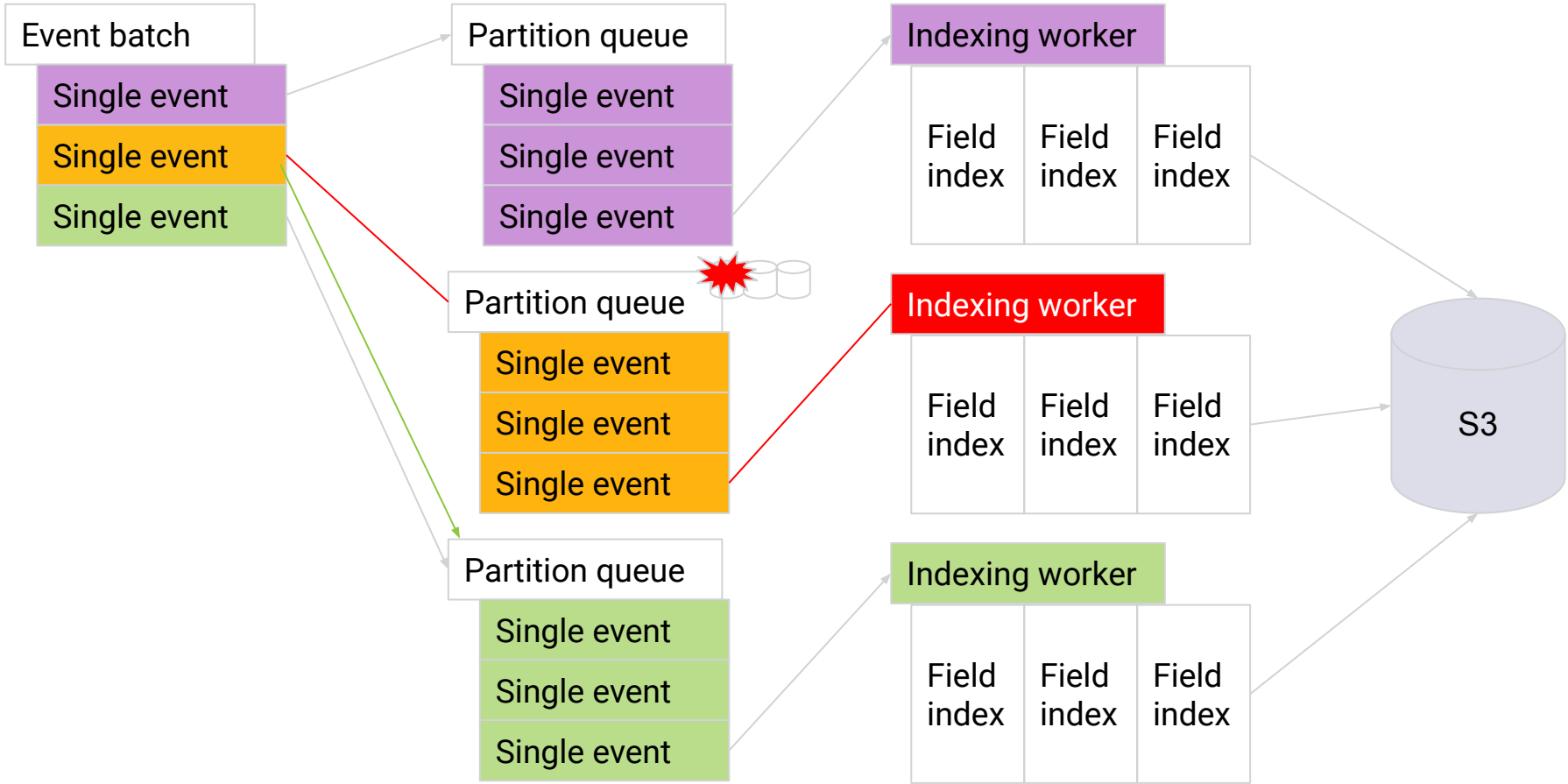


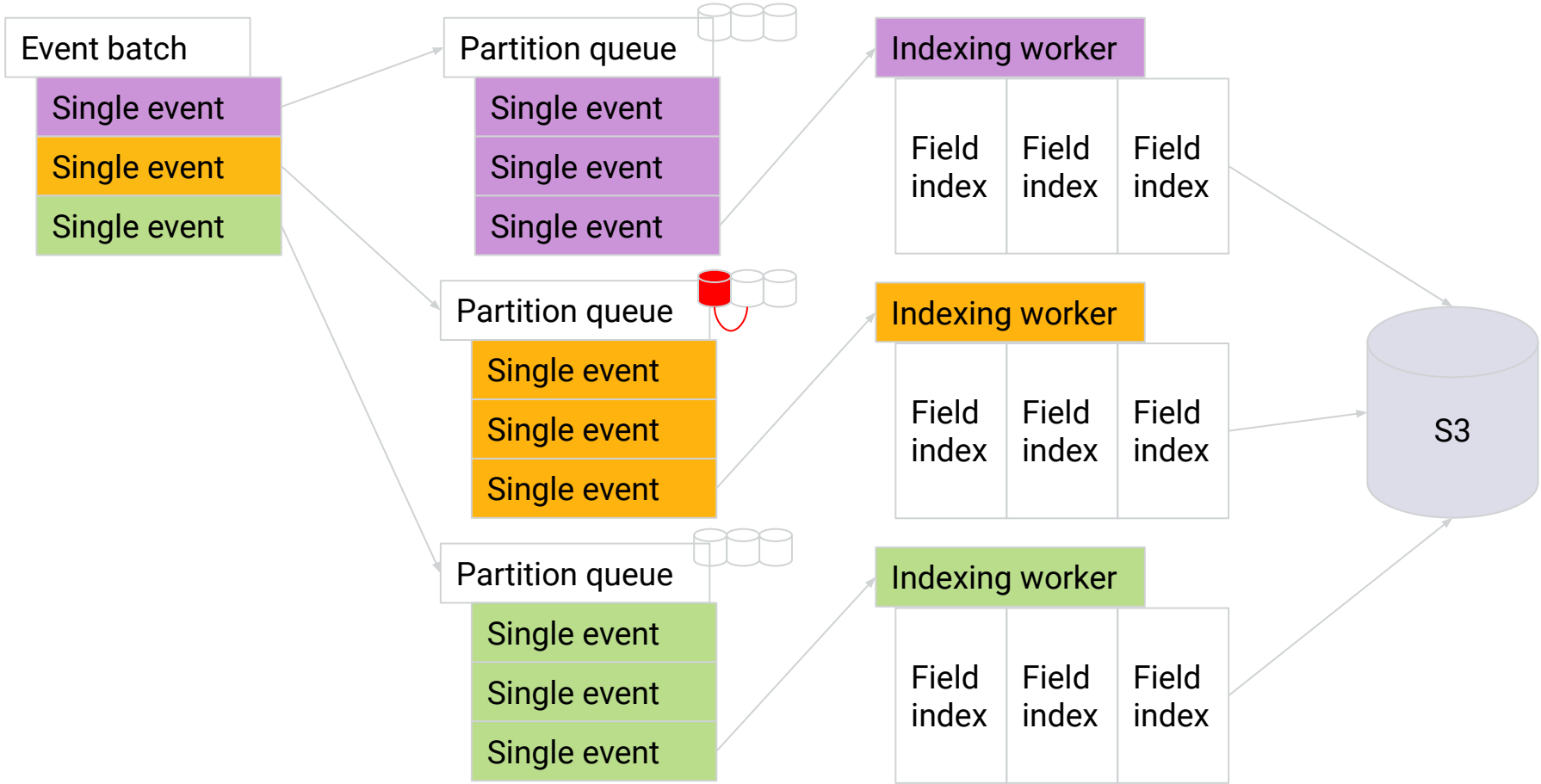


Restart one server & service at a time.



Monitor for changes using SLIs.

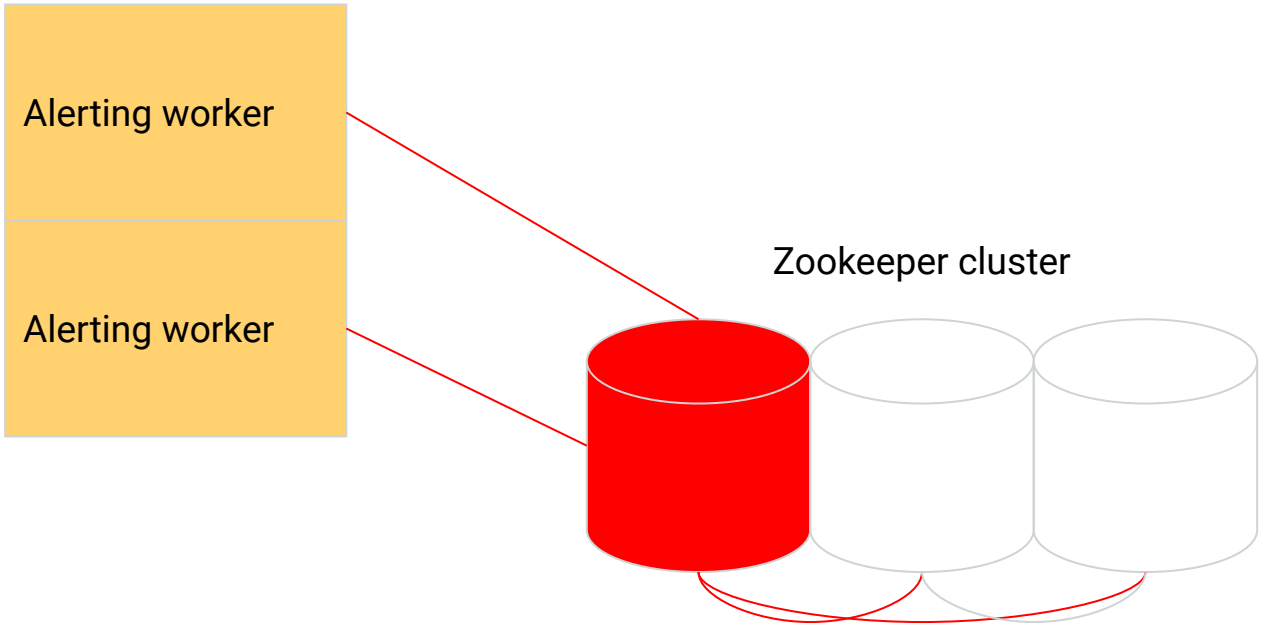


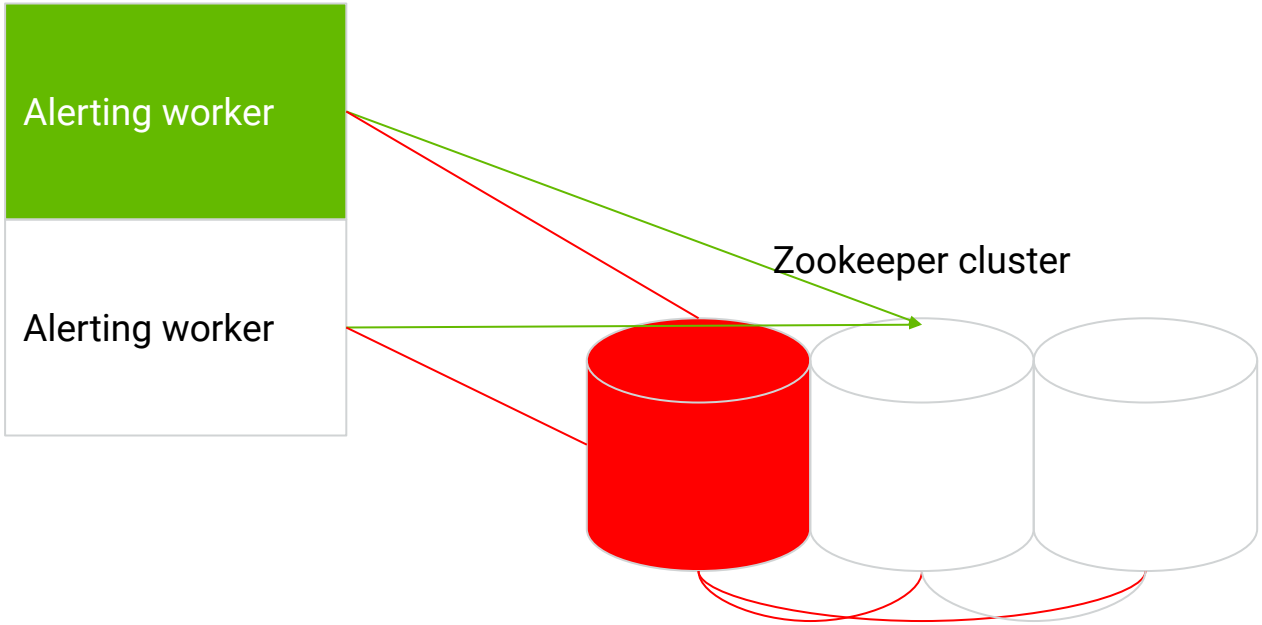


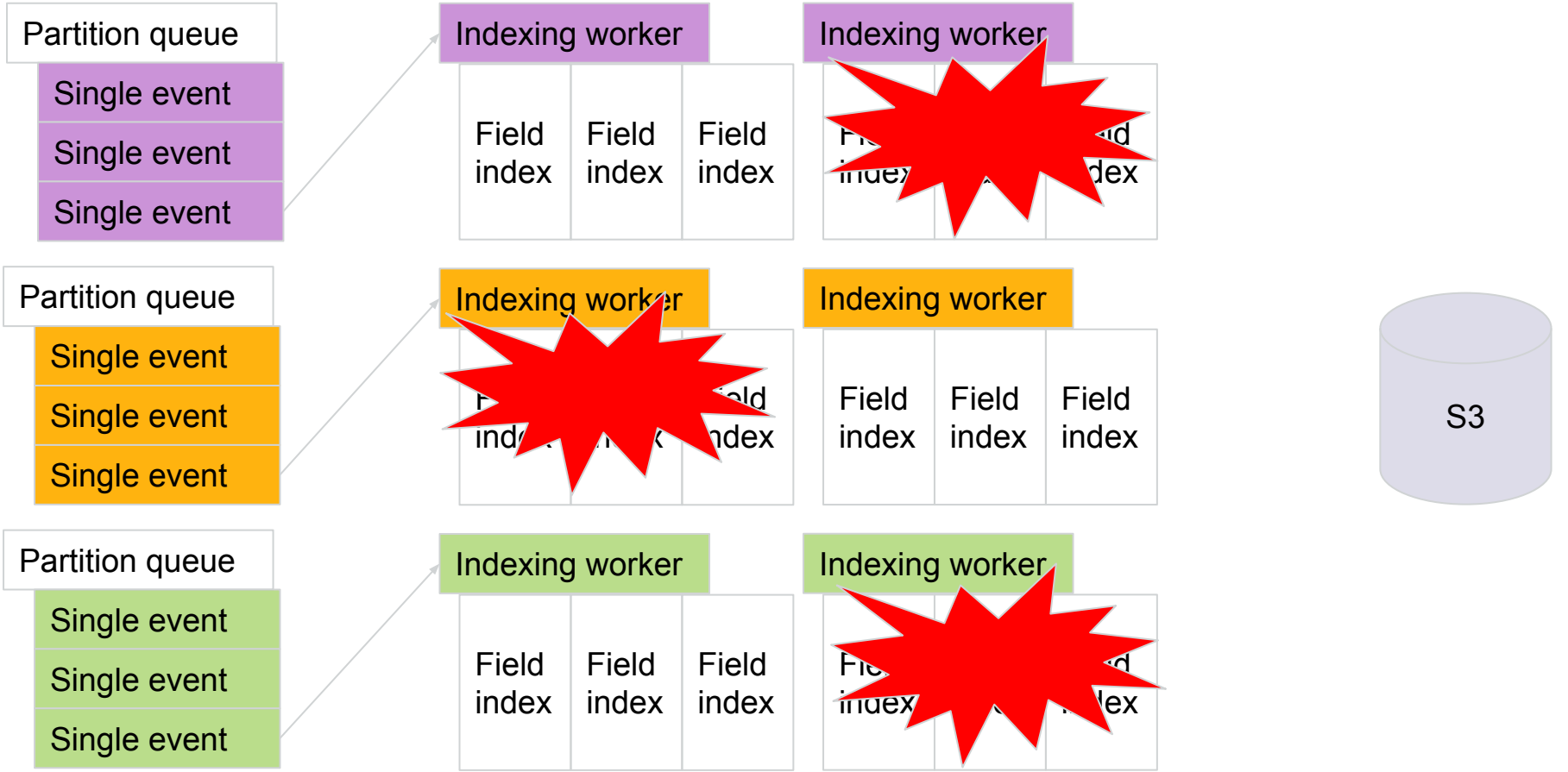
Alerting worker

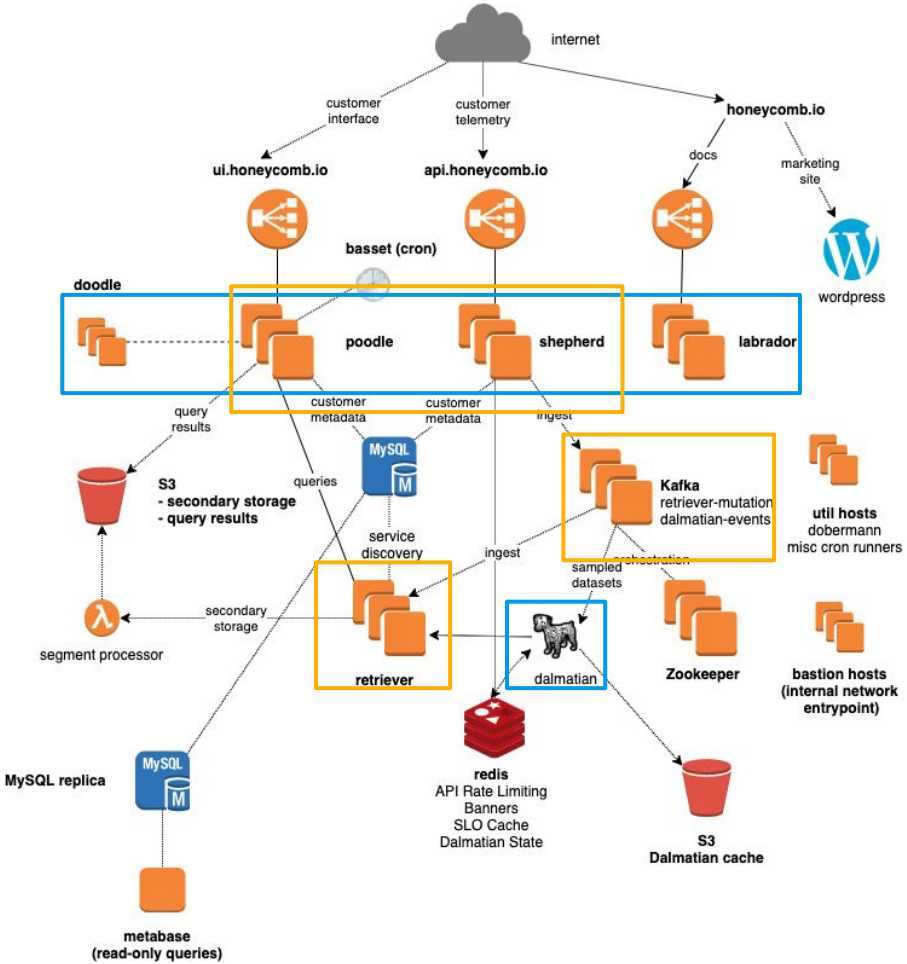
Alerting worker

Zookeeper cluster









ARM64 hosts
Spot instances

Reach out!

honeycomb.io/shelby

@shelbyspees

Questions?