OTel and Linkerd: A Match Made in Observability Heaven

Relationship status: it's complicated

Julia Furst Morgado
Principal Developer Advocate at Dash0

Flynn
Technical Evangelist at Buoyant



Meet the couple



Linkerd

LIKES

mTLS by default, Rust-powered proxies, negligible overhead.

DISLIKES

Mind-reading, carrying your app baggage.





OpenTelemetry

LIKES

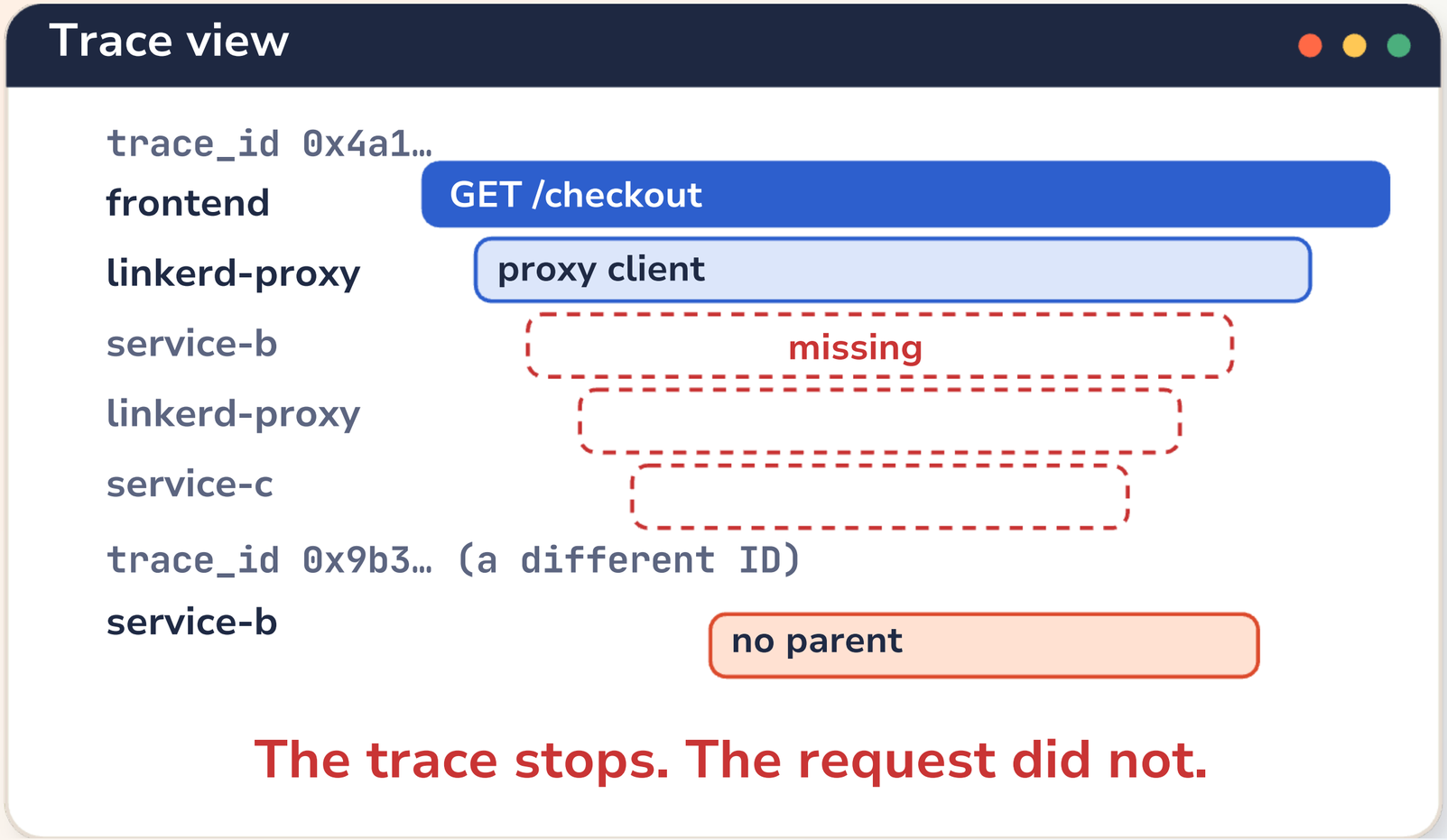
Standard schemas, vendor-neutral signals, telling the whole story.

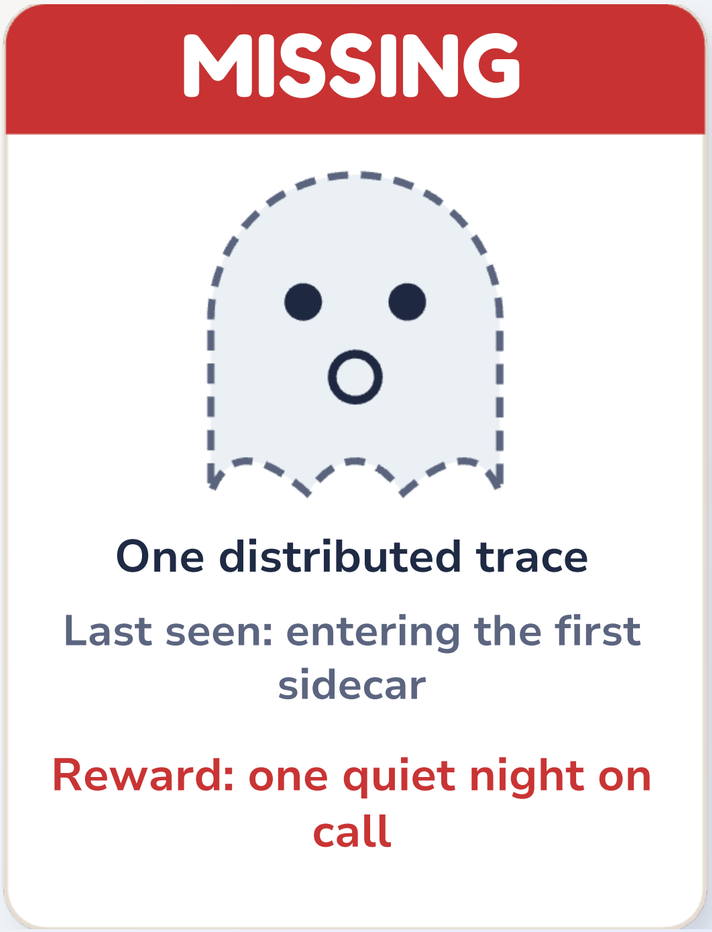
DISLIKES

Silent drops, severed relationships.

It's a match! Two CNCF projects swipe right. Then the traces stop talking.

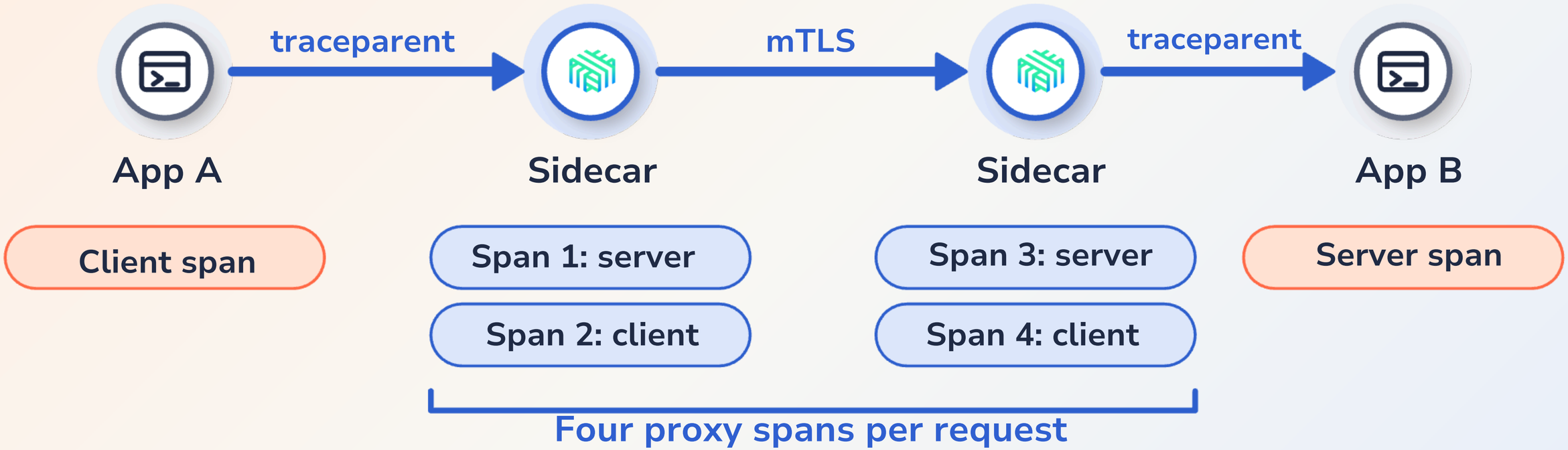
Ghosted at the first hop





Dashed spans are the ones that never arrived.

The mesh can't read your mind



Mesh: envelope
Sidecars carry the traffic and add spans.

App: letter
Only the app can copy traceparent to its next call.

The catch
Every span hangs off the header the app passes.

Anatomy of a hop: the proxy is not psychic. The app has to pass the note.

Break 1: the ghosting incident

Drops the context

```
resp, err := http.Get(url)

// no ctx, no header
```

Passes the context

```
client := http.Client{
  Transport: otelhttp.NewTransport(
    http.DefaultTransport)}
req, _ := http.NewRequestWithContext(
  r.Context(), "GET", url, nil)
```

1. The traceparent arrives at the app.
2. The outbound call is built without it, with a bare client or a fresh goroutine.
3. The next proxy sees no header, so there is no parent to join.

I read the note. I forgot to pass it on.

Break 2: different love languages



W3C

`traceparent`



Joins the trace



B3

`b3 or X-B3-*`



Joins the trace



W3C + B3

`both headers`



W3C wins, B3 is ignored



Vendor header

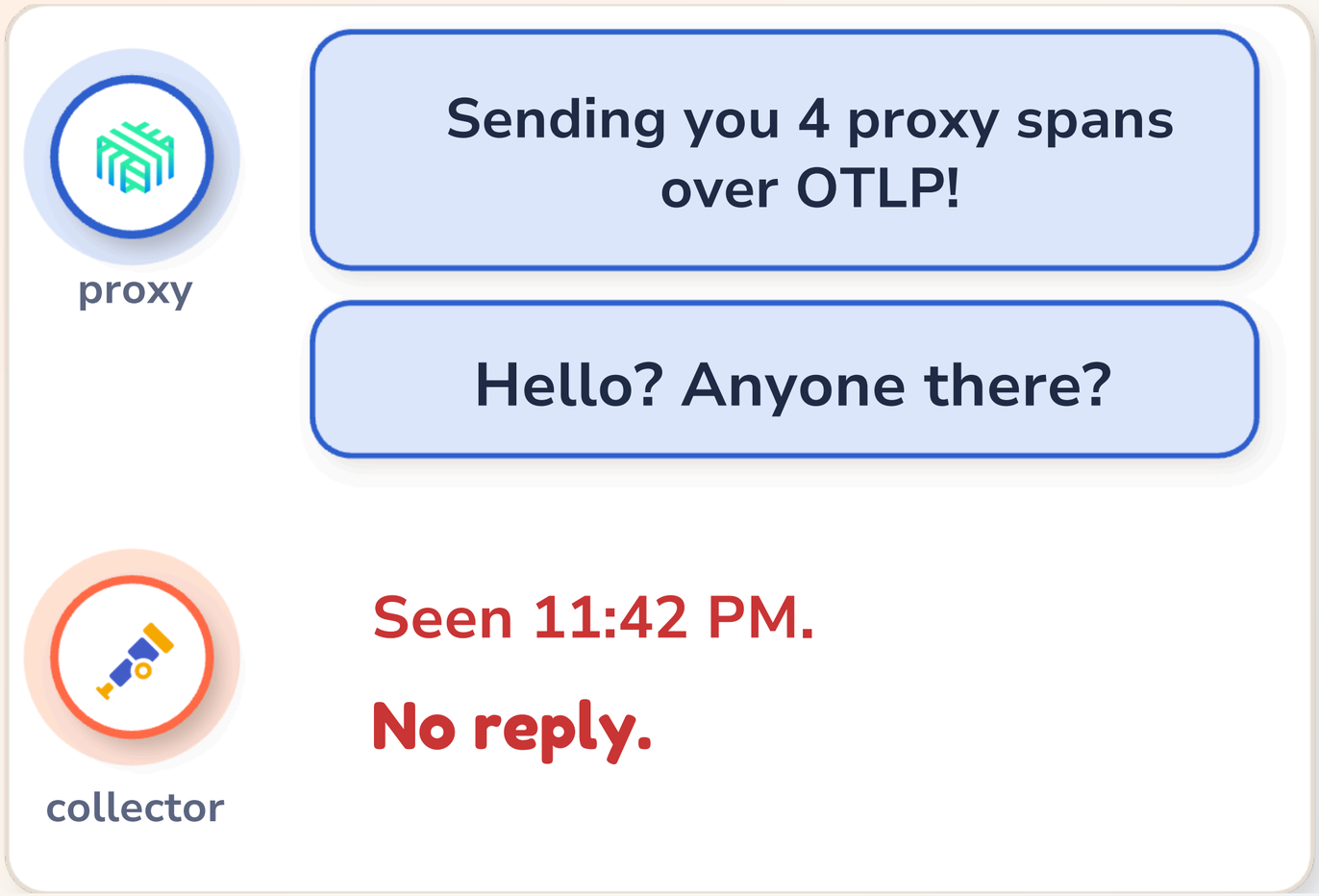
`X-Custom-Trace-Id`



Forwarded, never joined

Set your clients to W3C: OTEL_PROPAGATORS=tracecontext.

Break 3: left on read



Before you panic, check:

1. Tracing on?
   `proxy.tracing.enabled: true`
2. Collector in the mesh?
   `linkerd.io/inject=enabled`
3. Workloads restarted?
   `kubectl rollout restart`

Symptom: app spans show up, but no linkerd-proxy spans at all.

Break 4: bad timing



T = 5 ms

The leaf span ends and flushes first. Jaeger shows a child with no parent.



T = 180 ms

The root span ends after its database work and flushes.



T = 200 ms

The backend assembles the trace as spans land. It was whole all along.

Spans export when they end, so children arrive before parents. Judge a trace after the batch window.

Kids arrive before the parents. Do not file the P1 yet.

Couples therapy: which break is it?

	WHAT YOU SEE	WHO DROPPED THE BALL	THE FIX
1	Proxy span ends, a new trace starts in the app	The app forgot the context	Pass ctx on every outbound call
2	App spans and proxy spans never join	Header dialect mismatch	Enforce W3C everywhere
3	App spans, zero proxy spans	Tracing off, collector not meshed, or no restart	Enable, mesh the collector, restart
4	Orphan spans that heal after seconds	Normal asynchronous batching	Wait for the batch window

At 2 AM, do not guess. Find the symptom, find who dropped the ball.

Eight lines of YAML to say "I do"

```
proxy:
  tracing:
    enabled: true
    collector:
      endpoint: otel-collector.observability:4317
      meshIdentity:
        serviceAccountName: otel-collector
        namespace: observability
```

1 **Label the namespace**

```
kubectl label ns observability
linkerd.io/inject=enabled
```

2 **Apply the values**

```
helm upgrade linkerd-control-plane
linkerd/linkerd-control-plane
-f values.yaml
```

3 **Restart the workloads**

```
kubectl rollout restart deployment
-n <workload-ns>
```

Eight lines, a namespace label, and a restart. That is the commitment.

Sampling: whole traces or none

Head sampling: the front door decides

Ingress keeps 10% → Flag rides in traceparent → Everyone follows

Tail sampling: decide once the whole trace is in

all spans → **Tier 1: route**

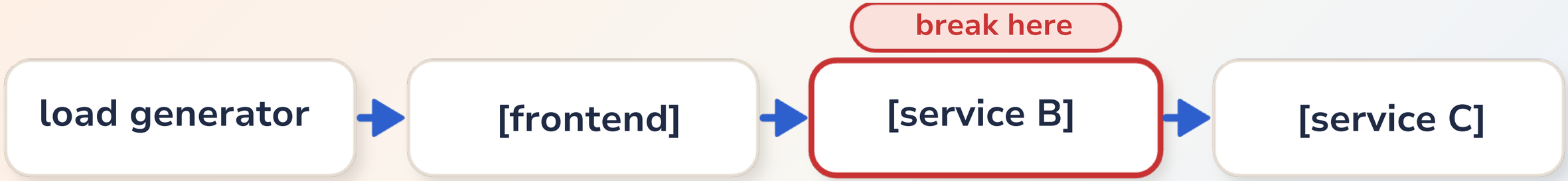
Same trace ID goes to the same collector → **Tier 2: decide**

Wait for the whole trace, keep errors and slow ones

10% kept independently at each of 4 hops = 0.01% chance of the whole trace.
A lottery ticket.

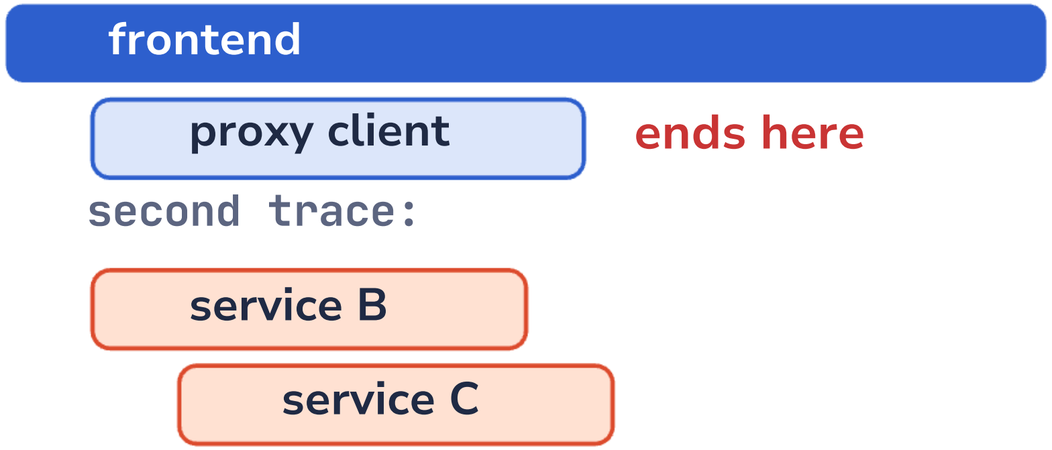
You cannot love half a trace.

The grand gesture: break it, then fix it

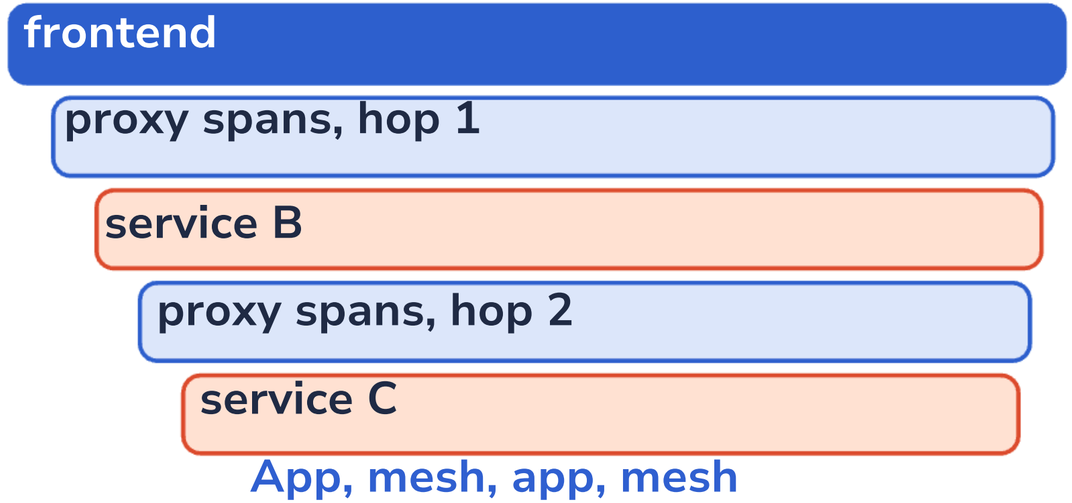


Every pod has a sidecar. All spans go to one collector, then Jaeger.

Before: the trace falls apart

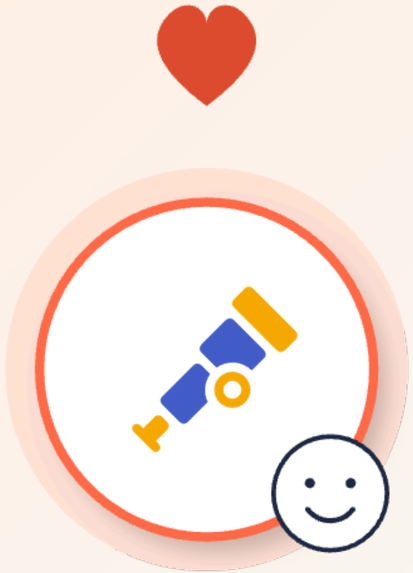


After: one unbroken waterfall

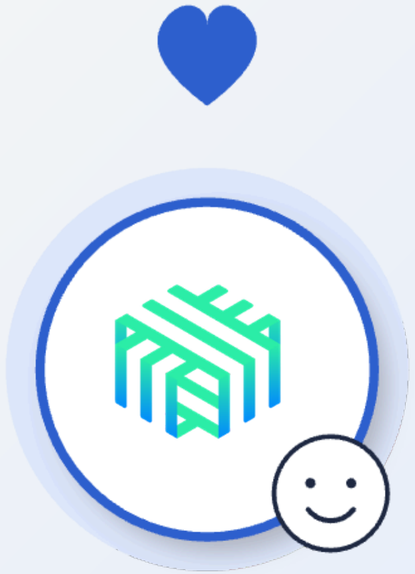


Service B stops passing the note. Watch service C show up as an orphan.

Our five vows for cloud-native bliss



1. I will pass the context on every outgoing call.
2. I will speak W3C and retire custom headers.
3. I will keep the collector reachable and meshed.
4. I will keep whole traces together when I sample.
5. I will be patient and let the slowest span report.



They lived happily ever after… until the next Helm upgrade.