Observability Day
EUROPE

Unpacking Your OTel Baggage

Dan Gomez Blanco
Principal Observability Architect, New Relic
OpenTelemetry End-User SIG Maintainer

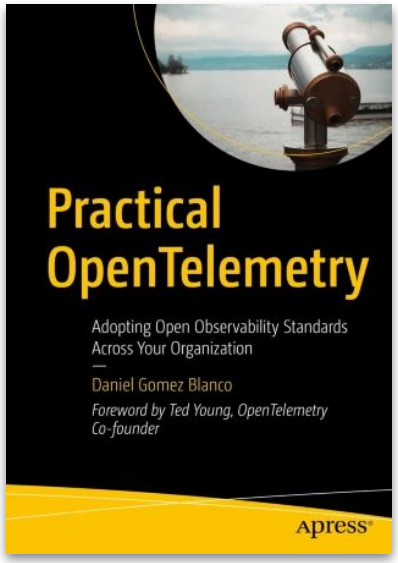
John Clark
Senior Software Engineer, Skyscanner
Istio Maintainer

Speakers

Observability Day EUROPE

Dan Gomez Blanco

- Principal Observability Architect @ New Relic
- OTel SIG End-User Maintainer
- https://dangb.me
- Author of Practical OpenTelemetry

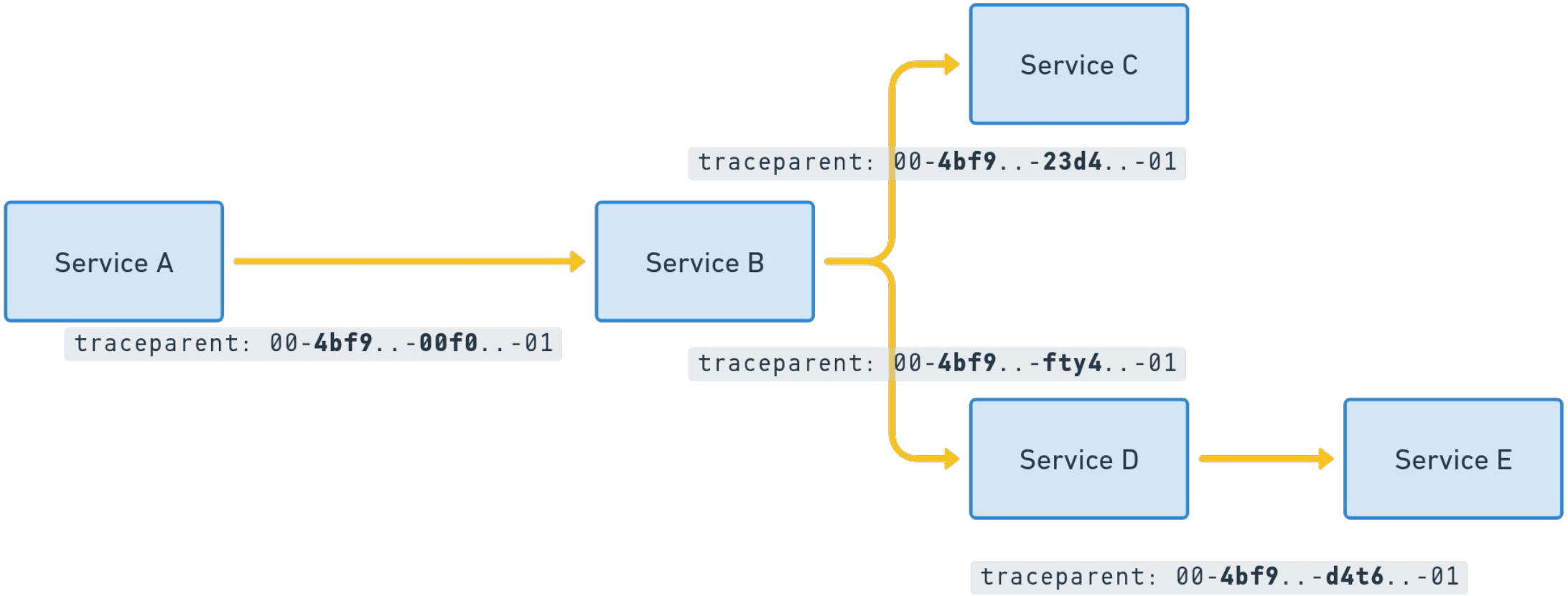


John Clark

- Senior Software Engineer @ Skyscanner
- Focus on Kubernetes and Istio
- Istio Maintainer (just)
- No book :(

Context Propagation

You are already familiar with **OTel Context** and
W3C Trace Context (`traceparent`, `tracestate`)



We Need Domain-Specific Context

In the real world, **Trace Context is not enough**.

We need to propagate more context:

- ➔ Session IDs
- ➔ Experiment variants
- ➔ Quality of Service constraints
- ➔ Correlation IDs *(maybe not…)*
- ➔ Many others…

Propagating Domain-Specific Context

Observability Day EUROPE

"*I'm going to **build my own tooling**, using my own headers and middleware, to propagate these custom properties. How hard can it be?*"

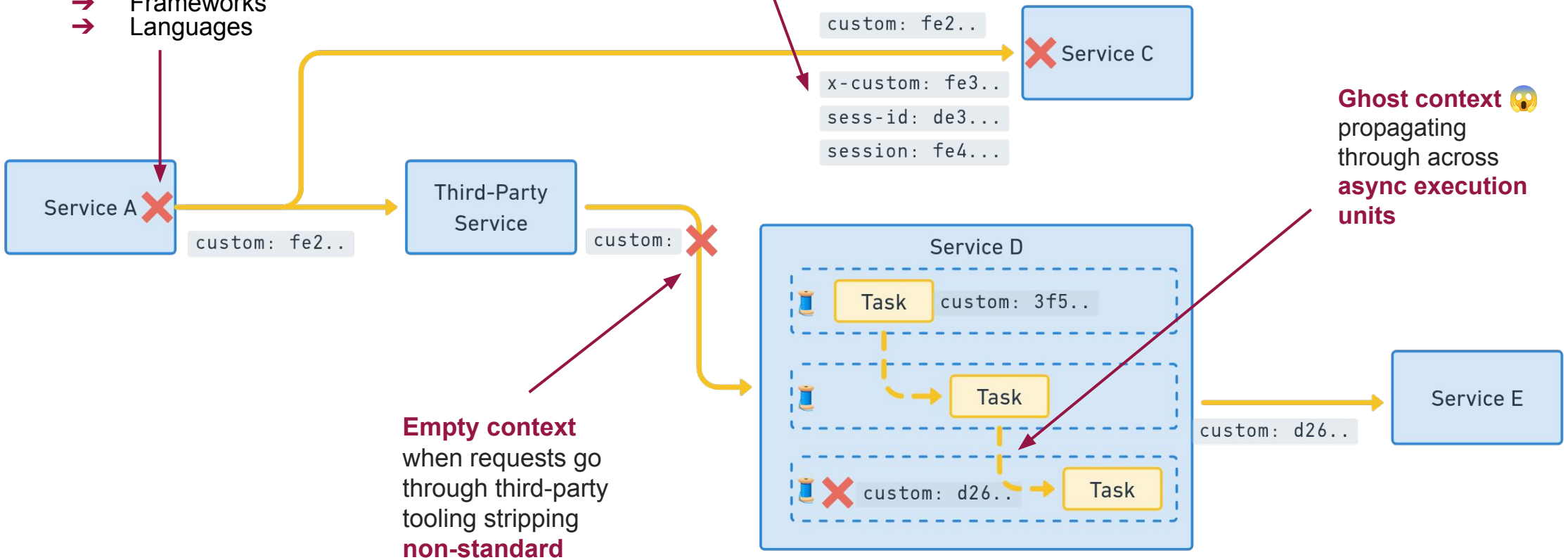
- Some engineer (*e.g. me*) before OTel Baggage

Where Things Start to Break

Maintenance toil
especially with new…

- Clients
- Protocols
- Frameworks
- Languages

Onboarding friction
and **cognitive load**…
"*What are these incoming headers?*"



Ghost context 😱
propagating through across **async execution units**

Empty context
when requests go through third-party tooling stripping **non-standard headers**

Don’t Let Me Stop You Though…

Observability Day EUROPE

What Baggage is

Observability Day EUROPE

OpenTelemetry Signal

The Baggage API provides an in-memory key-value store for arbitrary data

Implicitly propagated with Context (if language allows it)

```
baggage.Set('sessID', '2f3...');
baggage.Set('sessId', '8sg...');
baggage.Get('sessID'); // 2f3...
baggage.Get('sessId'); // 8sg…
Baggage.empty().makeCurrent()
```

Propagated via W3C Baggage

Standard for representing and propagating baggage via the `baggage` header

```
baggage:
     sessID=2f3...,
     sessId=8sg...,
     experiment42=A
```

Integrated with OTel Tooling

Supported by instrumentation libraries

Injected/extracted using Propagators API

```
propagator:
     composite:
          - tracecontext:
          - baggage:
          - your_custom_propagator:
```

What Baggage is not

Observability Day EUROPE

Not Auto-Exported or Attached

There's no Baggage exporter, and not part of OTLP.

To add Baggage to data:

- ➔ Manual API
- ➔ SDK Span/Log Processors
- ➔ Logging framework instrumentation (e.g. MDC)

Not Always Safe in Public Networks

Baggage may contain sensitive information.

Sanitise before propagating:

- ➔ Clear Baggage (e.g. set `Baggage.empty()` in the current context)
- ➔ Custom Propagator
- ➔ Instrumentation hooks

Not Best for Core Business Logic

Context propagation can be complex.

Side-effect of missing or inconsistent Baggage should not break core logic.

Observability Day
EUROPE

Baggage at Skyscanner

(no, not that type of baggage)

Skyscanner

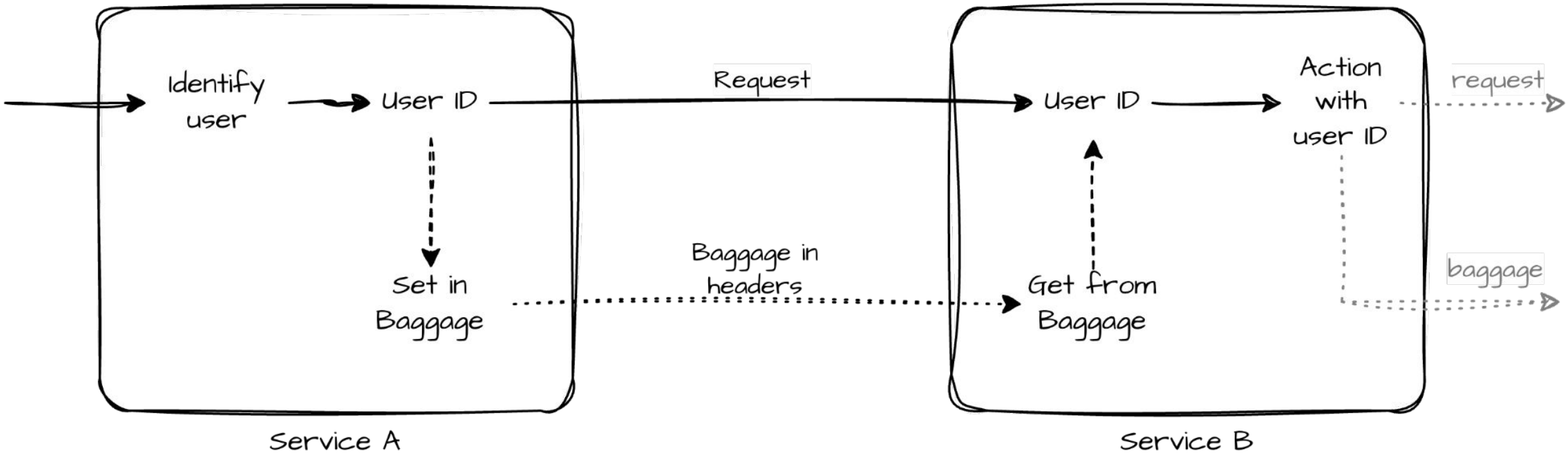
- ➔ Production workloads running on EKS
- ➔ 4 regions, 24 primary clusters
- ➔ 600+ microservices, 35k pods
- ➔ 60 million requests per minute
- ➔ Connected worldwide with Istio*
- ➔ Resilient architecture (failover, retries)

* other service meshes available

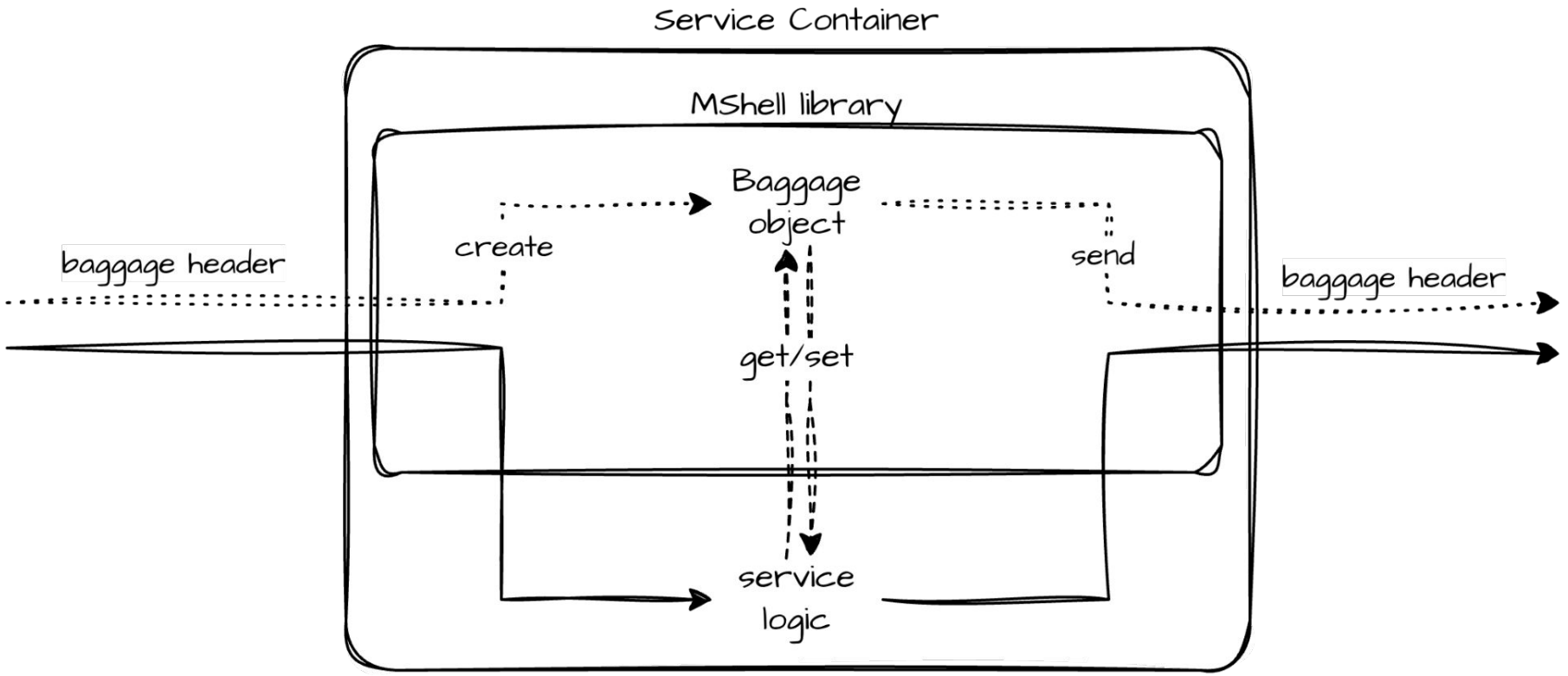
Service-Level Baggage

- Common library APIs that use Baggage across multiple services
  - Traveller/user IDs
  - Correlation IDs
  - Experimentation references
- Custom key/values and APIs built on Baggage
- Internal-only - with custom propagator

Service-to-Service



Public APIs


Service Container
MShell library
Baggage object
create
send
baggage header
baggage header
get/set
service logic

Quality of Service Asks

Observability Day EUROPE

What Service Owners Asked For

How can we better protect our critical streams if another is killing our service?

- ➔ Services can configure retries, failover etc.
- ➔ This applies globally **to all clients and requests**
- ➔ Cannot be differentiated for high-value requests

Differentiating Requests

We can't just say "Client A is more critical than Client B"

- ➔ Services send different types of requests
- ➔ Client A might send a critical request that needs highest priority
- ➔ It might also send the lowest priority type

Other Options We Considered

Istio (and other Service Meshes) tend to have rate limiting solutions:

- ➔ Global/regional rate limiting (Redis etc.)
- ➔ Pod level rate limiting

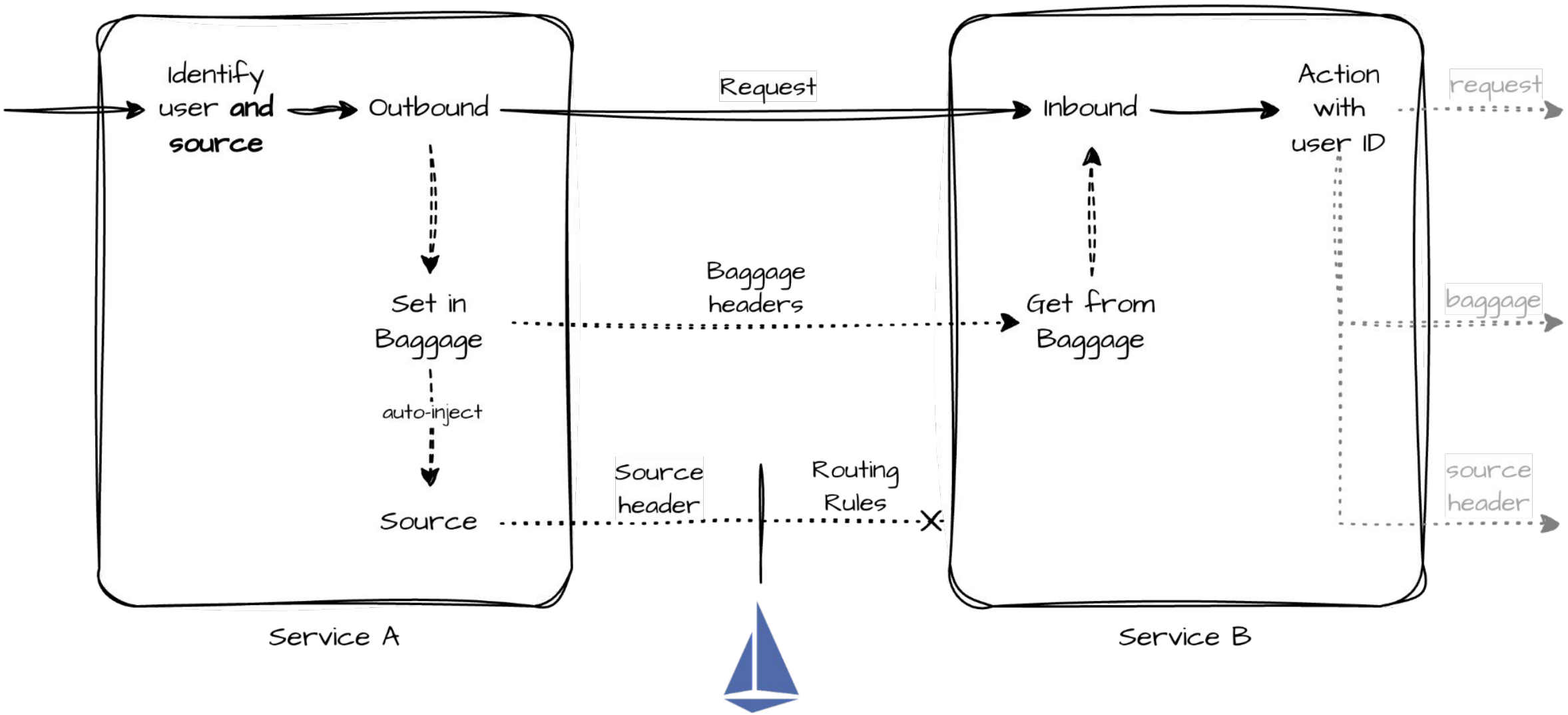
Heavy, complex, and tend to find people don't actually want "rate limiting"

Solution

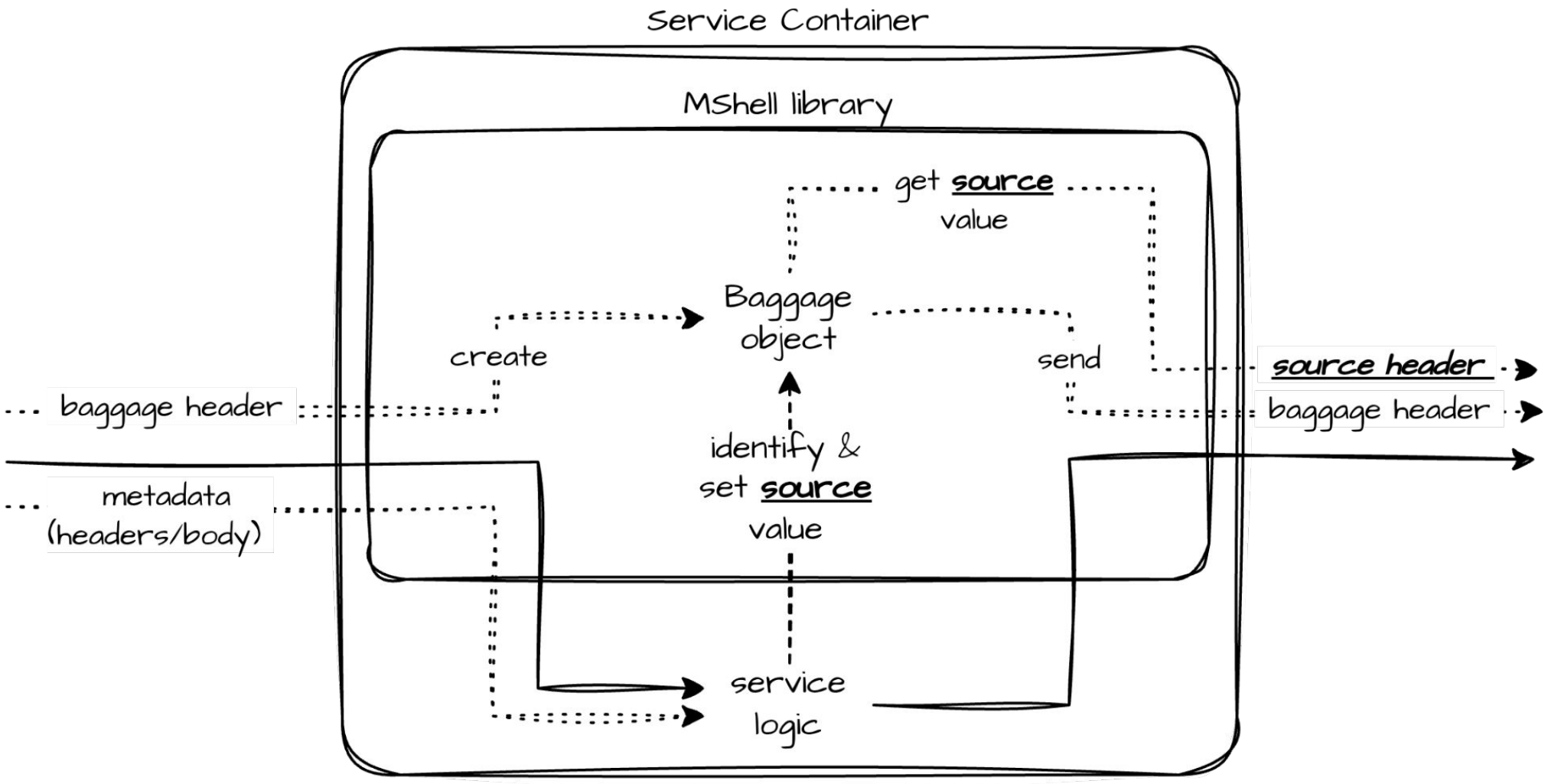
Why not expand with Baggage?

- Let top-level services identify the request “source”
  - App, web, B2B, batch/data, etc.
  - Set in Baggage
- Auto-inject Baggage “source” value to outbound requests
- Tune Istio values based on “source” value
  - “*I want app to get more requests, Batch work to get less*”
- Fire and forget - source is only set once on the flow

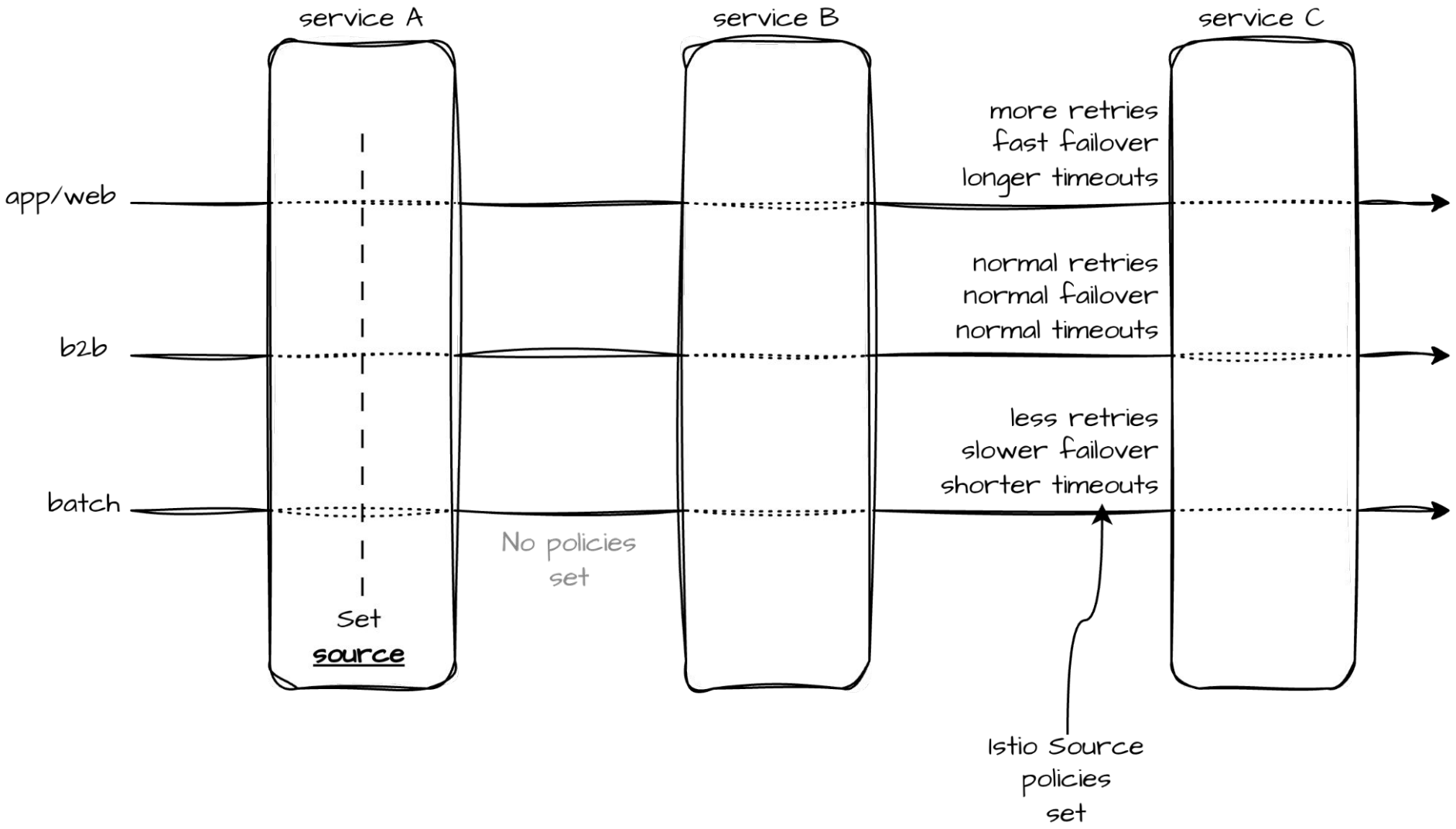
Routing Baggage



New Public API



Cross-Service Routing



Outcome

We've provided a new platform capability which **is enabled by OpenTelemetry Baggage**

- ➔ Session "source" is passed around services
- ➔ Gets set once in Baggage at the earliest possible point
- ➔ Auto-injected from Baggage into internal request headers
- ➔ Service owners decide what "sources" get priority
- ➔ No new infrastructure, fully compatible by extending Istio routing objects

Main Takeaways

Observability Day EUROPE

- Stop building your own tooling for context propagation, use Baggage
- Baggage != Span\Log Attributes
- The power of Baggage goes beyond observability!

Observability Day
EUROPE

Give us your feedback!

https://sched.co/2DY2T

(slides available soon)