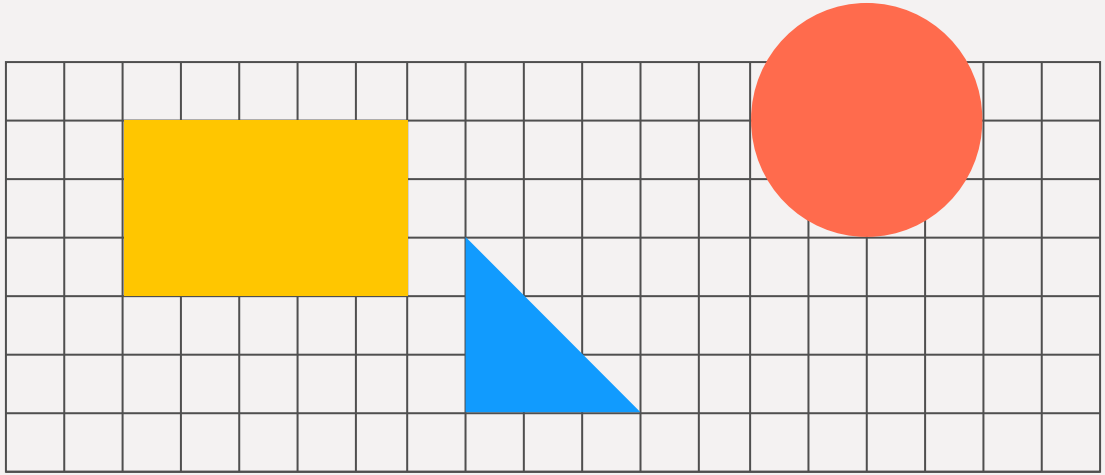
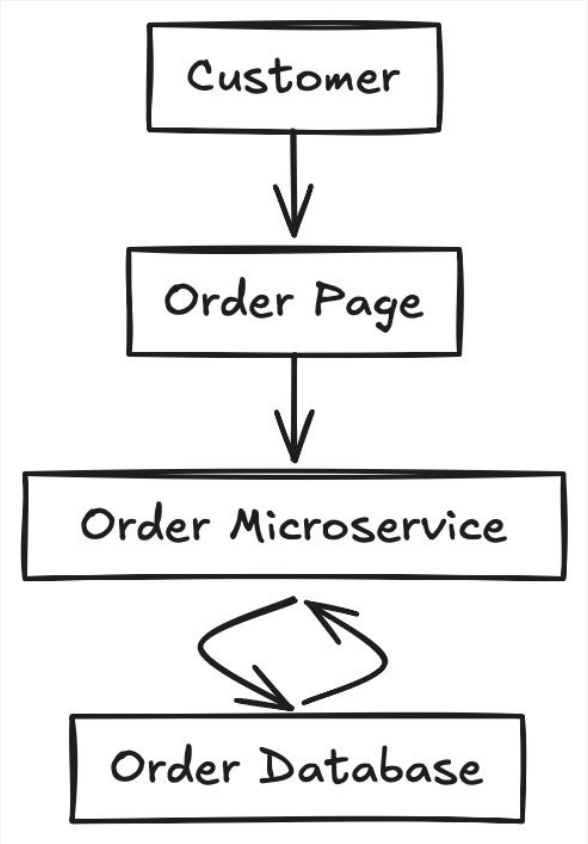
Schema-Driven Telemetry



A scenario, outside of Observability. . .

M&M bakery has a single software engineering team responsible for the order microservice. Their architecture looks something like the following.

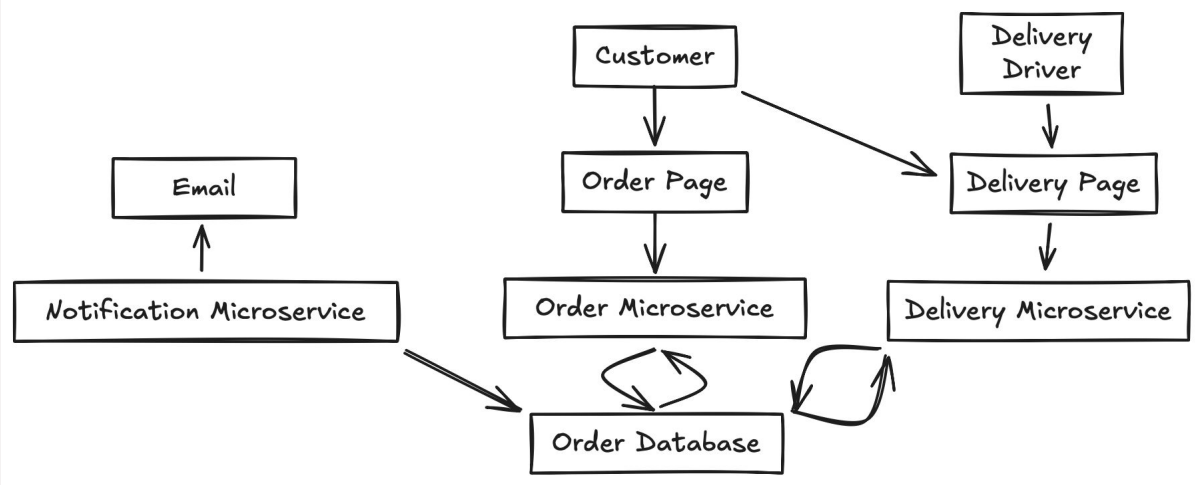


Scaling leads to complex interdependence

As the bakery grows, the owners task new teams with developing a **delivery service** & **notification service**.

The delivery & notification service **depend** on the **order database**, introducing cross-team dependencies.

Now, each time any of the three teams needs to update the schema of the database, they need to **communicate relevant changes** to all teams.



Dependency management

Traditional data engineering tools make managing these cross-team dependencies much easier.

Tools include

- Data cataloging to assist in discovery
- CI/CD automation to surface/block breaking changes
- Auto-generated language clients for DB interaction (improving devex)
- Contracts around database schema and versioning
- Automated DB migration tooling
- Privacy & security enforcement
- Knowledge graph enrichment (to give AI context)

All of these tools depend on the existence of a well-defined **schema.** Without schema(s), these tools cannot exist, and **sprawl** abounds.

Observability is the same!

Observability is no different, except that it operates at the scale of **every service in the company**.

What enterprise would allow every service/team in the company **unrestricted write access** to a **schema-less** database?

Data contracts (i.e. schemas) between the observability platform and the observed services *must* be clarified.

This will improve developer experience by:

- **Auto-generating** schema **documentation** & type-safe **language clients** for languages used by developers
- **Clarifying** the contract of **updating the enterprise-level schema** (think Elastic index templates, etc.)
- Providing a ground truth for **auto-migration capabilities** of dashboards, monitors, or any other assets that depend on telemetry data
- **Improving** the **discoverability** of telemetry data, especially for less-experienced developers

Phase 1: Adopt a language-agnostic telemetry standard & pipeline

Two foundational capabilities are required to enable schema-driven telemetry

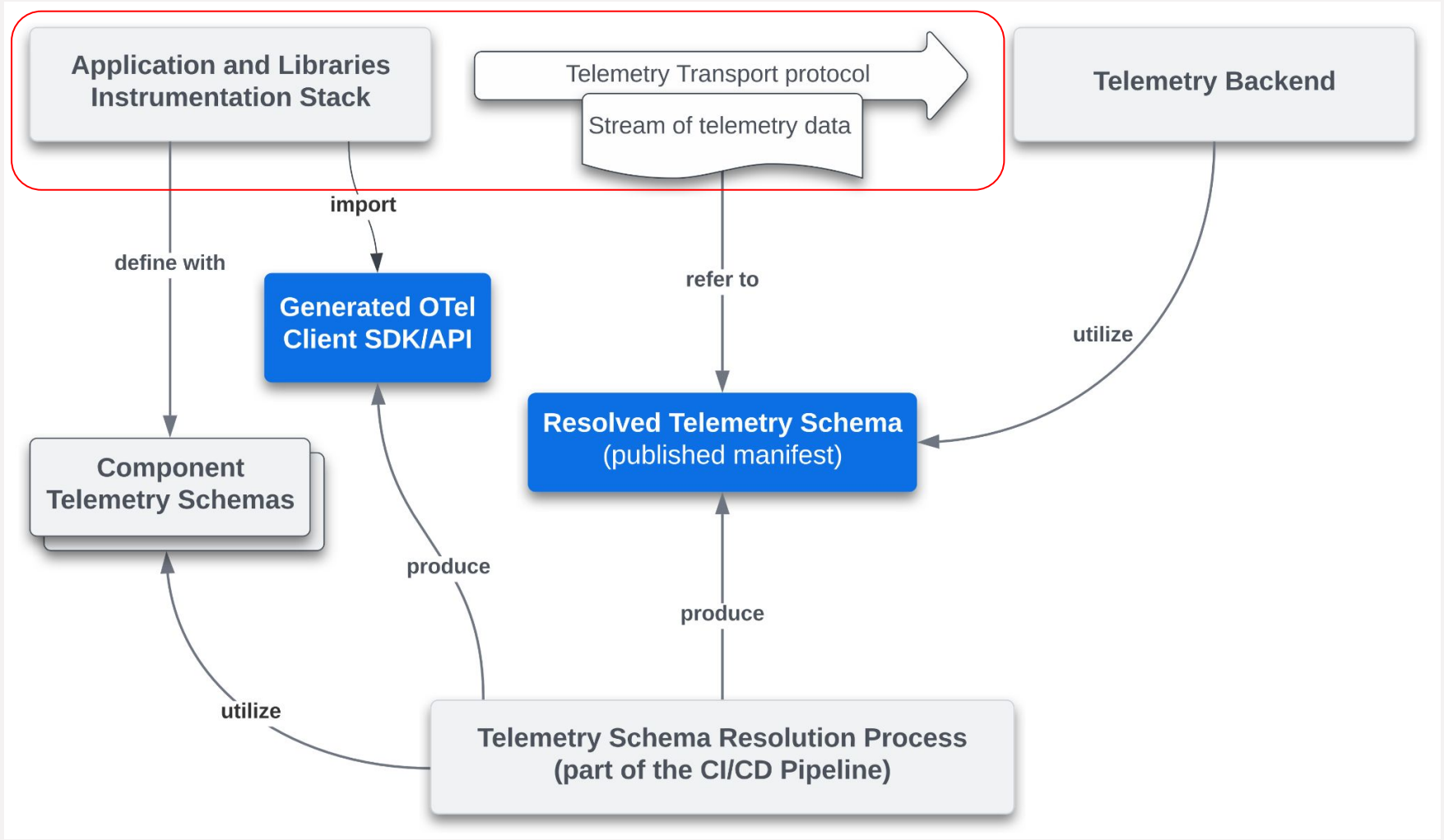
1. A **telemetry SDK/standard** that **spans all languages** used in the enterprise
2. **Observability pipelines** to collect, filter, enrich, and transform telemetry data

But that sounds like a lot of development work! The good news is, **these already exist** under the OpenTelemetry project.

OpenTelemetry instrumentation APIs/SDKs exist for

- Java/Kotlin
- Python
- Javascript/Typescript
- Go

The OpenTelemetry collector can be deployed as a daemonset and/or a gateway to collect and transform telemetry data. In fact, the OpenTelemetry Kubernetes Operator can both **deploy the OTEL collector** and **insert auto-instrumentation** for Java/Kotlin, Python, Node.js, and Go services in the cluster without any code changes.


Application and Libraries Instrumentation Stack
Telemetry Transport protocol
Stream of telemetry data
Telemetry Backend
import
define with
refer to
utilize
Generated OTel Client SDK/API
Resolved Telemetry Schema (published manifest)
Component Telemetry Schemas
produce
produce
utilize
Telemetry Schema Resolution Process (part of the CI/CD Pipeline)

Phase 2: Develop an enterprise-level telemetry schema (a.k.a semantic convention registry)

Once Phase 1 is complete, an enterprise-level telemetry schema can be developed. Again, OpenTelemetry has already created a standard for this, *semantic conventions*.

Semantic conventions specify common **names** for different kinds of telemetry (metrics, spans) and fields/tags (a.k.a attributes in OTEL-speak) An enterprise-level telemetry schema would define span names, metric names, and attributes unique to SoFi.

This would provide

- A **documentation reference** for all teams when implementing instrumentation
    - Has someone already created a custom metric like this before? If so, what did they name it?

- A **contract/SLA** for updating this schema or registry
    - How do I request a new indexed field/tag for my log?

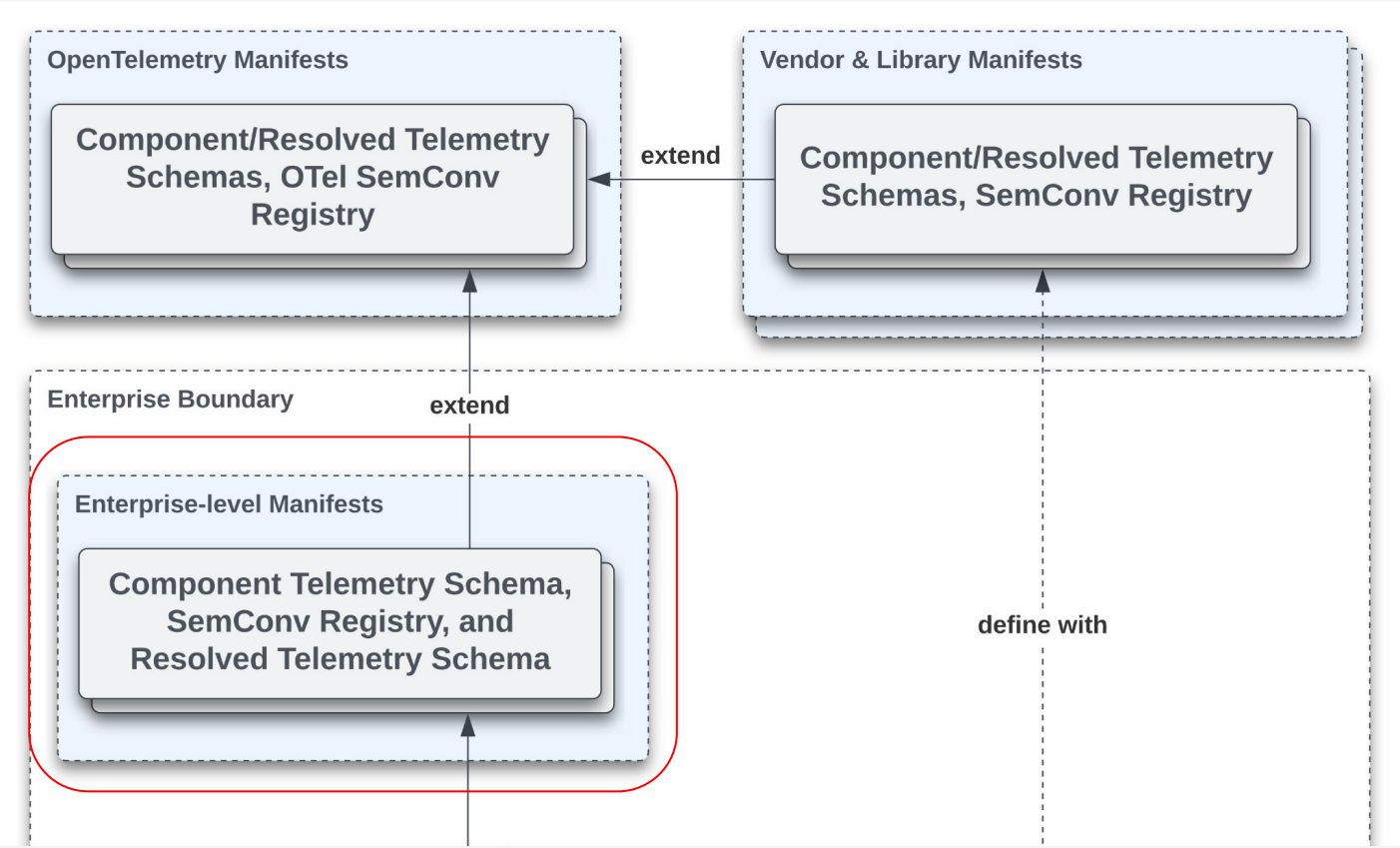
- **Automation** can be built **to implement this schema** in relevant backends
    - For example, automatically sync the fields/tags from the log schema to index templates in Elastic

- **Auto-generated instrumentation clients**/SDKs for each language
    - A library for teams to import and reference type-safe fields (w/ migration via library versioning)

Use case 1: Publishing a semantic convention registry

These two use cases yield the same outcome: a Resolved Telemetry Schema that is easy to publish and consume.

Use case 2: Publishing an app/lib telemetry schema


OpenTelemetry Manifests
Component/Resolved Telemetry Schemas, OTel SemConv Registry
extend
Vendor & Library Manifests
Component/Resolved Telemetry Schemas, SemConv Registry
Enterprise Boundary
extend
Enterprise-level Manifests
Component Telemetry Schema, SemConv Registry, and Resolved Telemetry Schema
define with

Phase 3: Implement application-level telemetry schemas

Once an enterprise-level telemetry schema is developed, individual application telemetry schemas can be implemented.

In OpenTelemetry, the concept is called **App Telemetry Schema**s, and is currently on the roadmap. An app telemetry schema defines *exactly* what telemetry data is surfaced by the service. It can import schema definitions from OpenTelemetry's schema and the enterprise-level schema.

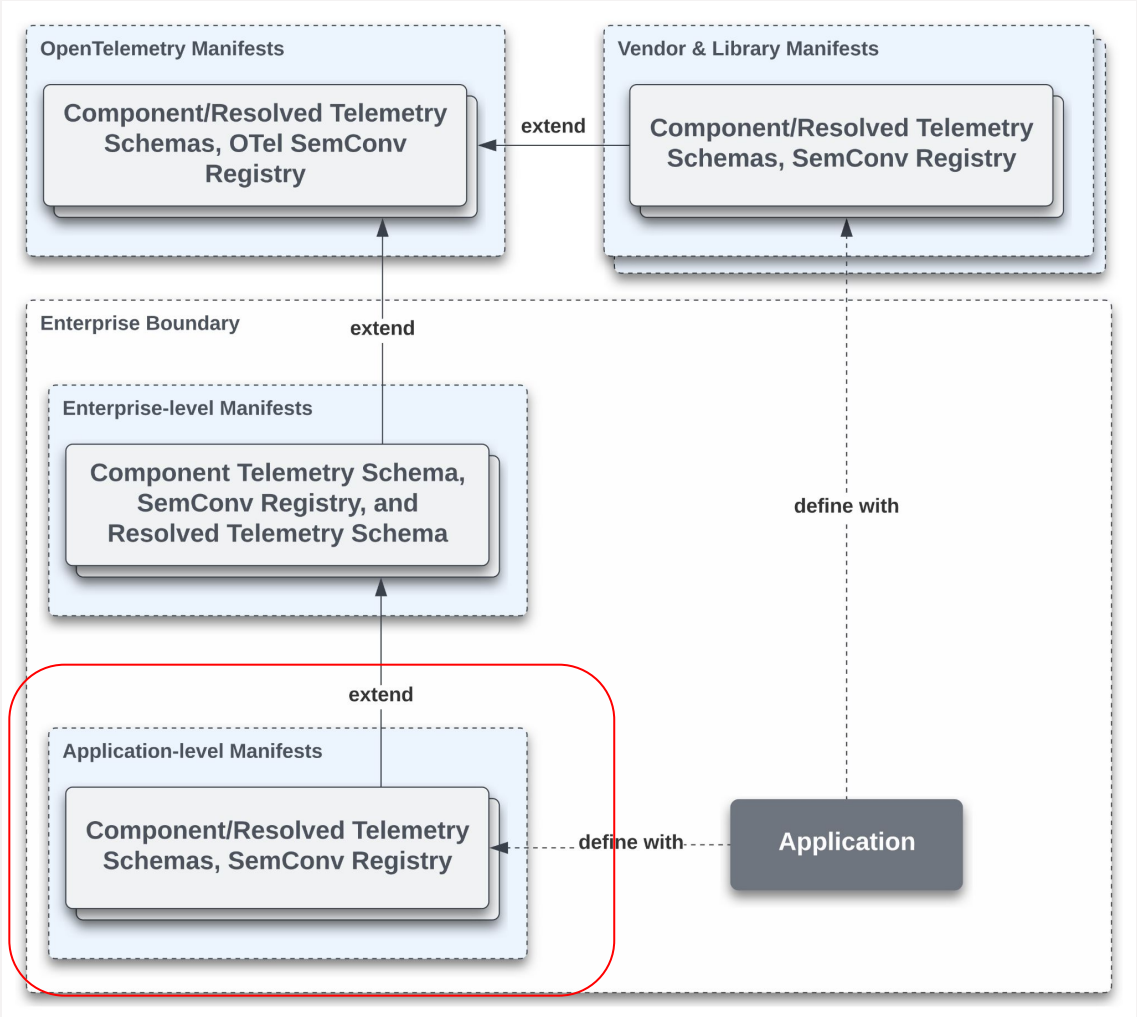
This would provide

- An accurate and up-to-date **documentation reference** or **data catalog** for discovery of telemetry data

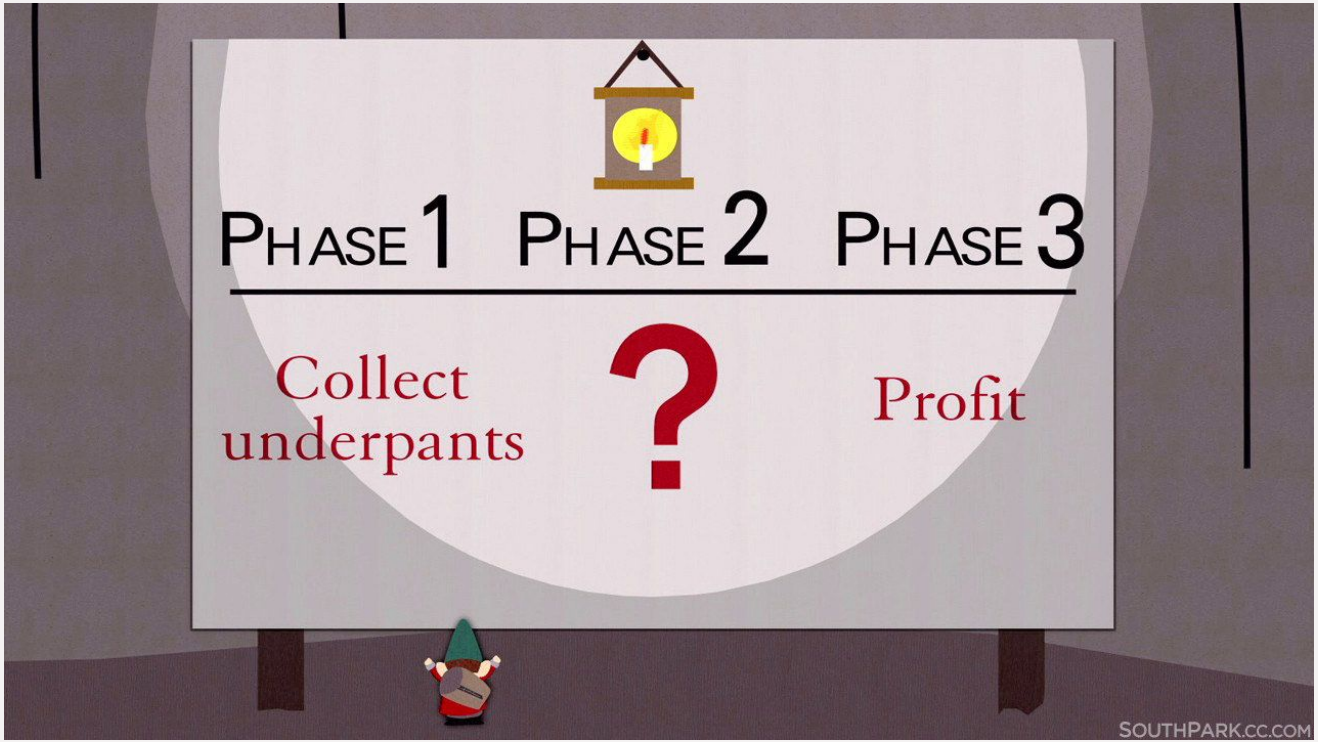
- **Automatic migration of assets** associated with the application's telemetry data
    - As span, metric, and attribute names update, dashboards and monitors can be automatically updated by traversing the schema history

- The possibility to define **compliant and non-compliant telemetry** at the service-level for improved storage efficiency

- The capability to produce a **knowledge graph for AI consumption**
    - Picture asking an LLM "Write me a <X vendor language> query to show <Y metric>", and it already has the context to create an accurate query using SoFi's semantics.


OpenTelemetry Manifests
Component/Resolved Telemetry Schemas, OTel SemConv Registry
extend
Vendor & Library Manifests
Component/Resolved Telemetry Schemas, SemConv Registry
Enterprise Boundary
extend
define with
Enterprise-level Manifests
Component Telemetry Schema, SemConv Registry, and Resolved Telemetry Schema
extend
Application-level Manifests
Component/Resolved Telemetry Schemas, SemConv Registry
define with
Application

Phase 4: Profit



Resources

OpenTelemetry Semantic Conventions

OTEP 243: App Telemetry Schema

Meta’s Schema-driven Telemetry