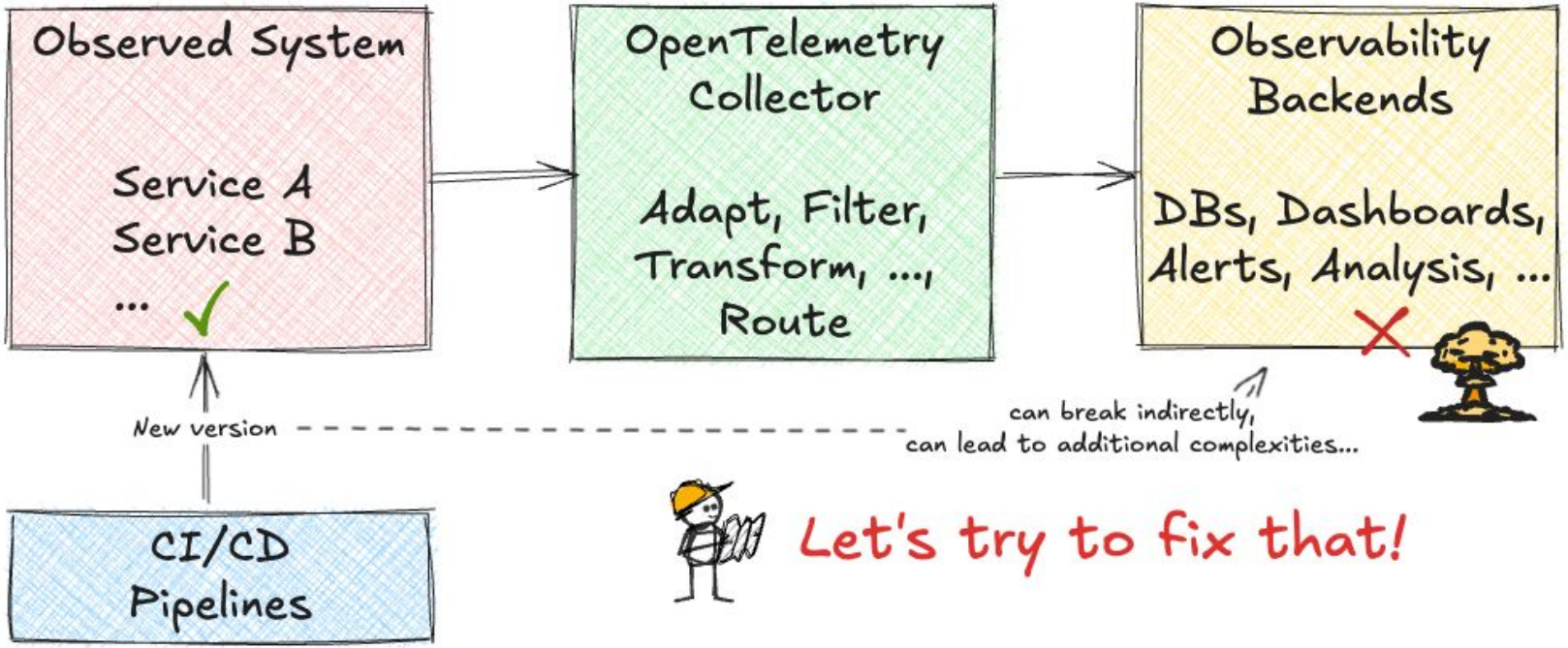
OpenTelemetry Semantic Conventions and How to Avoid Broken Observability



Laurent Querel
Dinesh Gurumurthy

Who are we

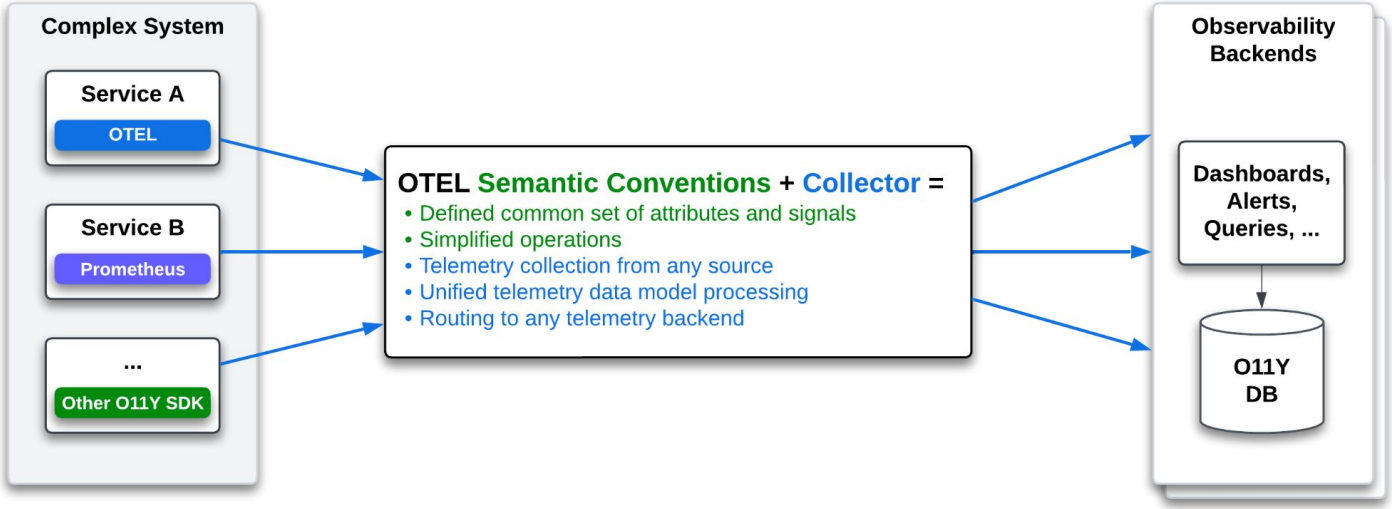
Laurent Querel
Staff Engineer F5
OpenTelemetry Maintainer

Dinesh Gurumurthy
Staff Engineer
Datadog

What is OpenTelemetry?

Vendor-neutral, open source observability framework

Instrument applications in any* language



What is OTEL SemConv Registry?

An **Open Catalog** of telemetry definitions maintained by 9 SIGs.

Ensure consistency and **interoperability** across observability tools.

Collection of Semantic Convention Files (YAML Format) →

900+ attributes and signals organized in 74 domains

```yaml
groups:
  - id: metric.http.server.active_requests
    type: metric
    metric_name: http.server.active_requests
    stability: development
    brief: "Number of active HTTP server requests."
    instrument: updowncounter
    unit: "{request}"
    attributes:
      - ref: http.request.method
        requirement_level: required
      - ref: url.scheme
        requirement_level: required
        examples: ["http", "https"]
      - ref: server.address
        requirement_level: opt_in
        brief: >
          Name of the local HTTP server that received
        note: |
          See Setting `server.address` and `server.po
          > **Warning**
          > Since this attribute is based on HTTP hea
          > to it may allow an attacker to trigger co
          > limits, degrading the usefulness of the m
      - ref: server.port
        requirement_level: opt_in
        brief: >
          Port of the local HTTP server that received
        note: |
          See Setting `server.address` and `server.po
          > **Warning**
          > Since this attribute is based on HTTP hea
          > to it may allow an attacker to trigger co
          > limits, degrading the usefulness of the m
```

But maintaining such a registry isn't that simple!

Have you ever experienced one of these situations?

Deploying an update that **breaks existing alerts or dashboards.**

Writing **overly complex queries:**

```
http_request_duration > 500ms OR http.req.dur > 500ms OR http.request.duration > 500ms
```

Having team members struggling with **unclear metrics.**

Wasting hours debugging a production issue due to **missing/incomplete instrumentation**

Have you ever experienced one of these situations?

Deploying an update that **breaks existing alerts or dashboards.**

Detect breaking changes / Control upgrades and downgrades

Writing **overly complex queries:**

```
http_request_duration > 500ms OR http.req.dur > 500ms OR http.request.duration > 500ms
```

Having team members struggling with **unclear metrics.**

Wasting hours debugging a production issue due to **missing/incomplete instrumentation**

Have you ever experienced one of these situations?

Deploying an update that **breaks existing alerts or dashboards.**

Writing **overly complex queries:**

```
http_request_duration > 500ms OR http.req.dur > 500ms OR http.request.duration > 500ms
```

Identify deviations from naming conventions

Having team members struggling with **unclear metrics.**

Wasting hours debugging a production issue due to **missing/incomplete instrumentation**

Have you ever experienced one of these situations?

Deploying an update that **breaks existing alerts or dashboards.**

Writing **overly complex queries:**

```
http_request_duration > 500ms OR http.req.dur > 500ms OR http.request.duration > 500ms
```

Having team members struggling with **unclear metrics.**

Ensure metadata quality and documentation updates

Wasting hours debugging a production issue due to **missing/incomplete instrumentation**

Have you ever experienced one of these situations?

Deploying an update that **breaks existing alerts or dashboards.**

Writing **overly complex queries:**

```
http_request_duration > 500ms OR http.req.dur > 500ms OR http.request.duration > 500ms
```

Having team members struggling with **unclear metrics.**

Wasting hours debugging a production issue due to **missing/incomplete instrumentation**

Analyze instrumentation quality in the CI/CD pipeline

OTEL Weaver to the rescue!

A CLI tool supported by the SemConv project, capable of:

- **Parse/Resolve** semantic conventions
- **Check compliance** with best practices
- **Generate** documentation, code, and schemas
- **Compute differences** between registry versions
- … and more

Available on DockerHub and seamlessly integrates into your CI/CD pipelines.

Quick overview of the default OTel policies

No attributes outside registry

Definitions require stability

No requirement levels on attributes

Names must follow format rules

IDs must match naming patterns

Attributes fully qualified

No constant name collisions

No namespace collisions

No duplicate attributes in group

Experimental attributes in stable groups must be opt-in

No removal of elements

No stability downgrades

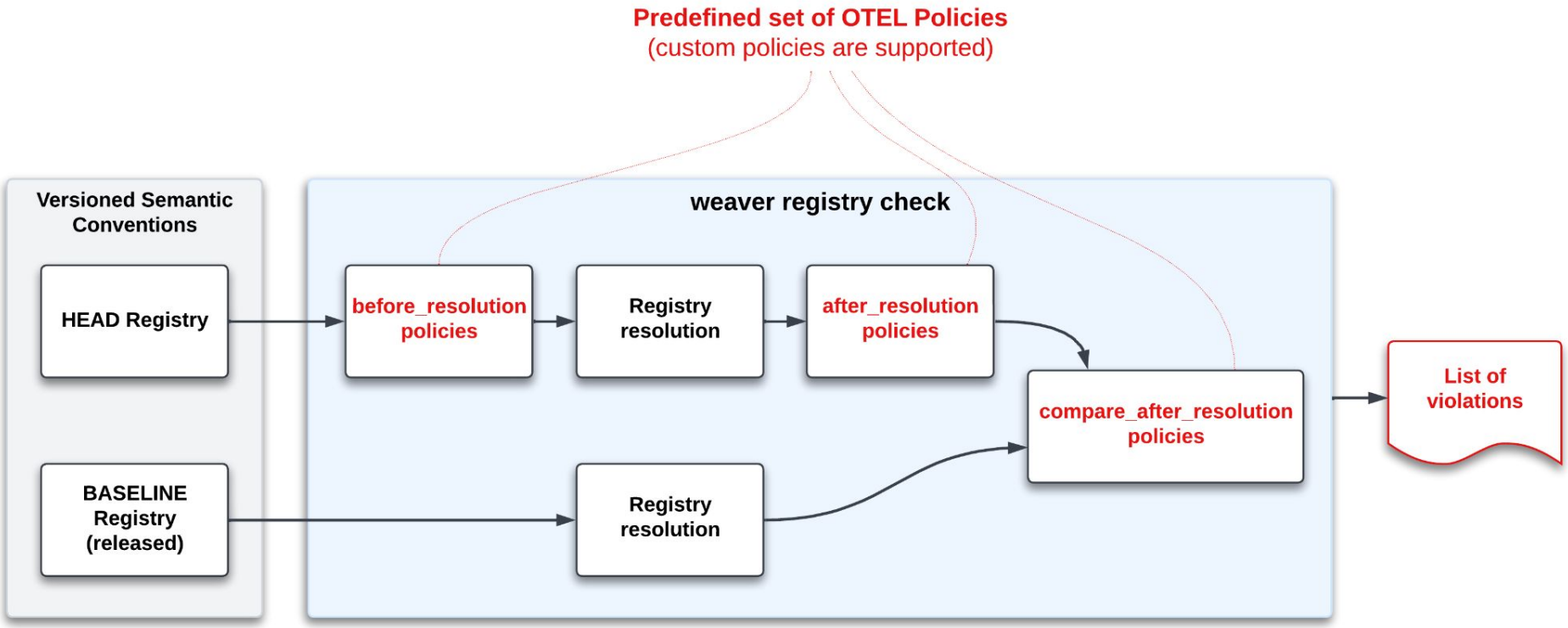
No type or unit changes

Attribute sets immutable

Enum values immutable

You can define your own policies!

Ensure registry quality and compliance with best practices



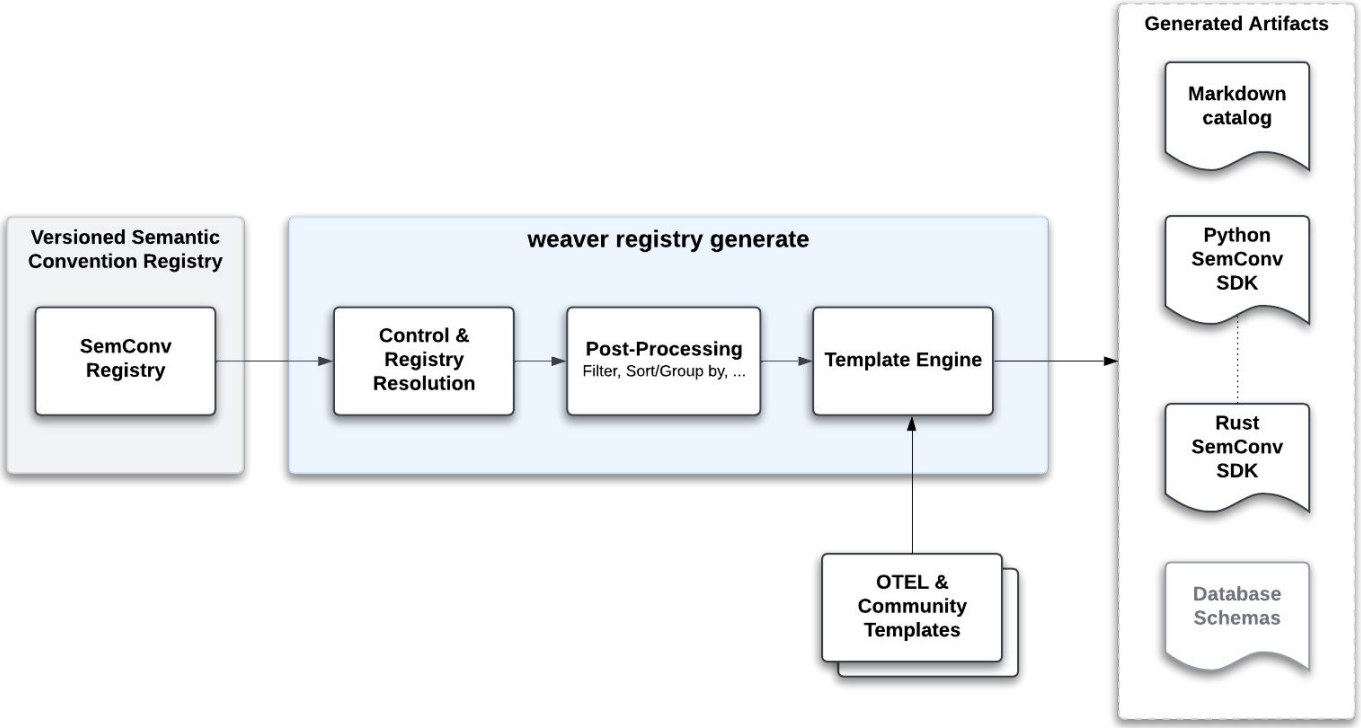
Rego Policy example

Evaluated during the comparison_after_resolution phase

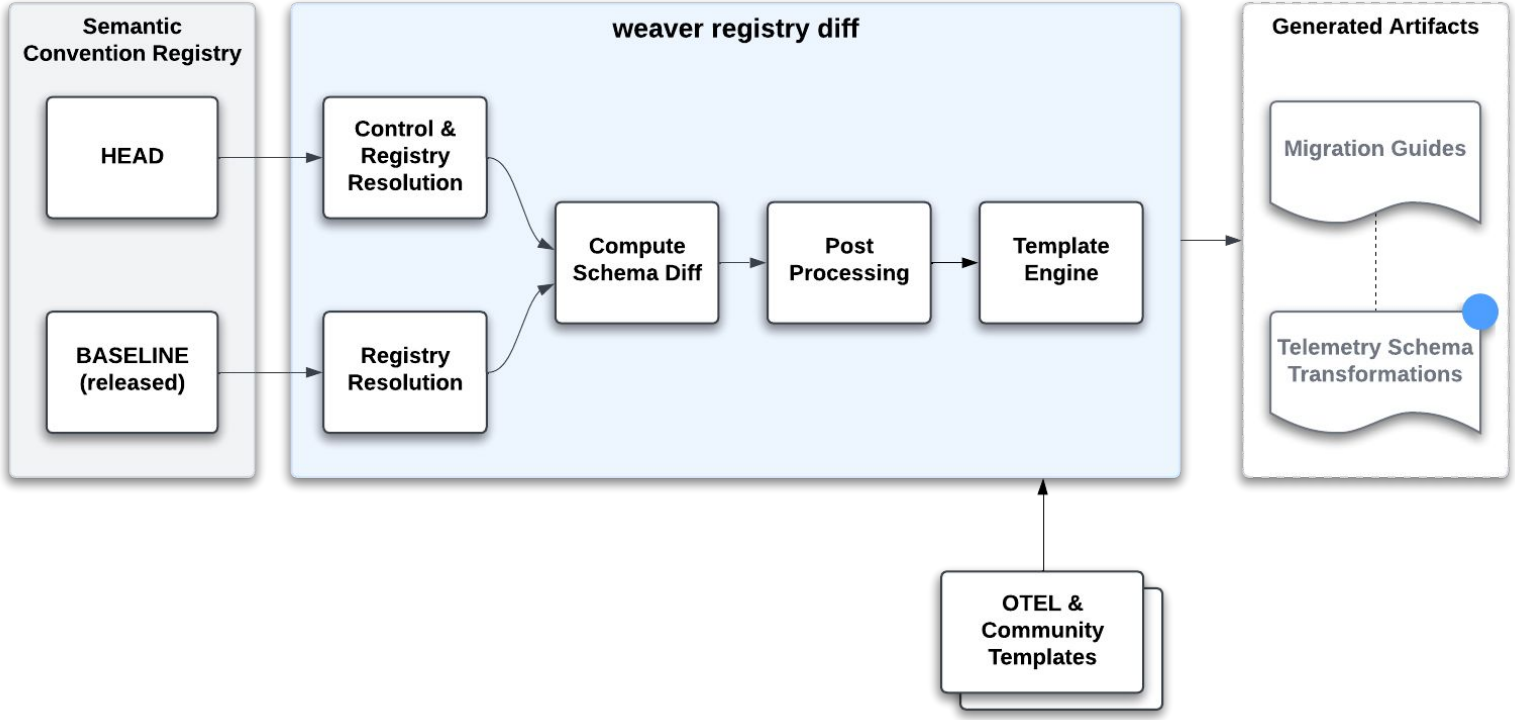
```
# Rule: Detect Removed Attributes
#
# This rule checks for attributes that existed in the baseline registry
# but are no longer present in the current registry. Removing attributes
# is considered a backward compatibility violation.
#
# In other words, we do not allow the removal of an attribute once added
# to the registry. It must exist SOMEWHERE in a group, but may be deprecated.
deny contains back_comp_violation(description, group_id, attr.name) if {
    # Check if an attribute from the baseline is missing in the current registry
    some attr in baseline_attributes
    not registry_attribute_names[attr.name]

    # Generate human readable error.
    group_id := data.semconv.baseline_group_ids_by_attribute[attr.name]
    description := sprintf("Attribute '%s' no longer exists in the attribute registry", [attr.name])
}
```

OTEL policies: https://github.com/open-telemetry/semantic-conventions/tree/main/policies

Keep documentation and SDK clients up to date



Manage schema evolutions

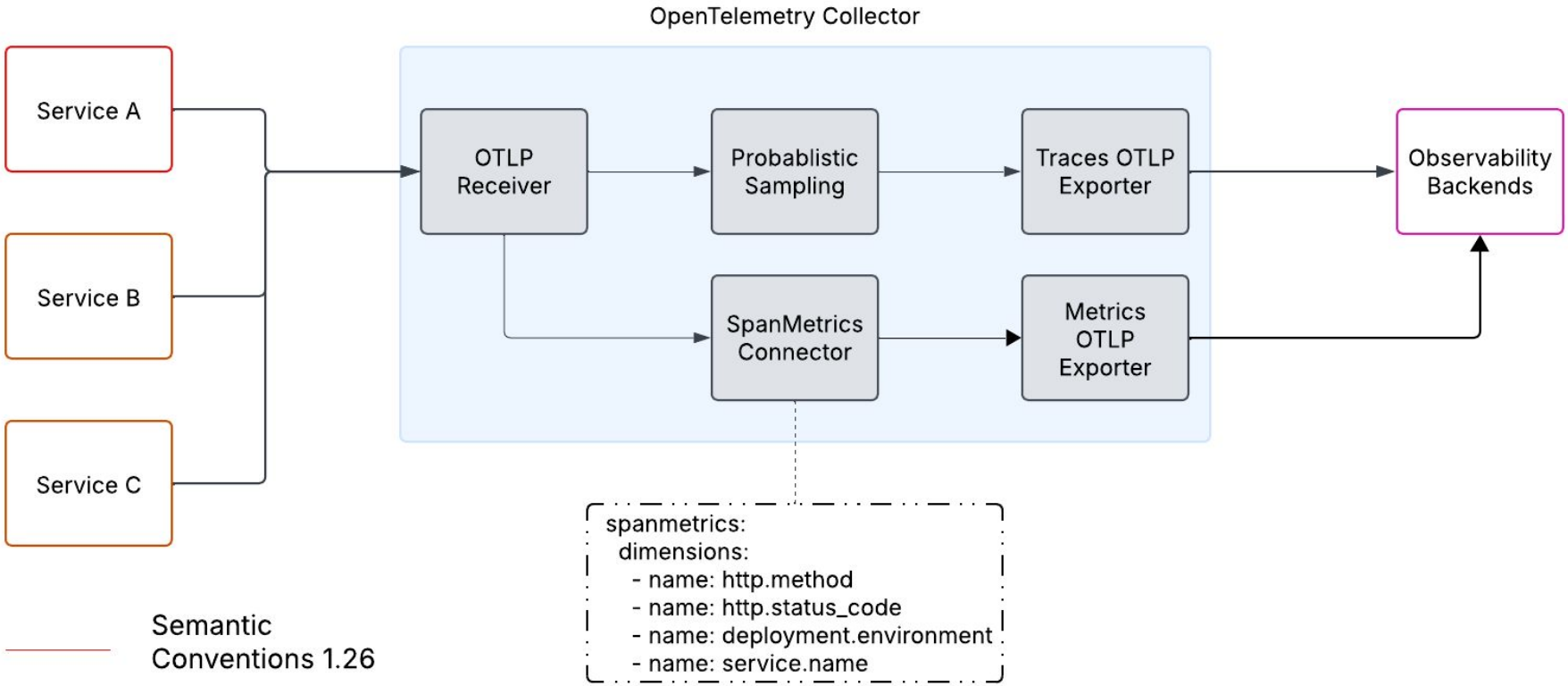


OpenTelemetry Collector

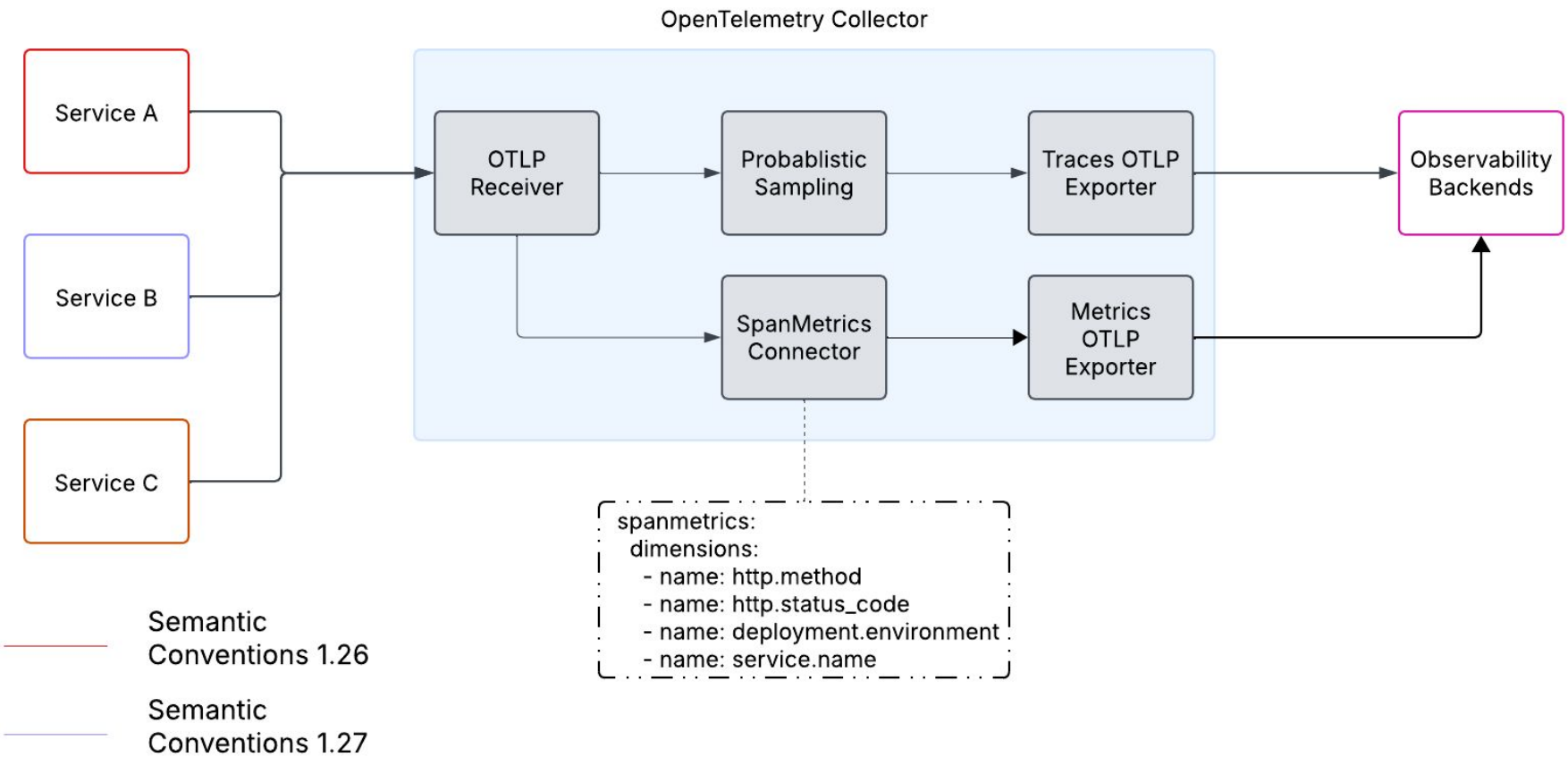
Managing Schema Evolution

OpenTelemetry Collector Pipelines

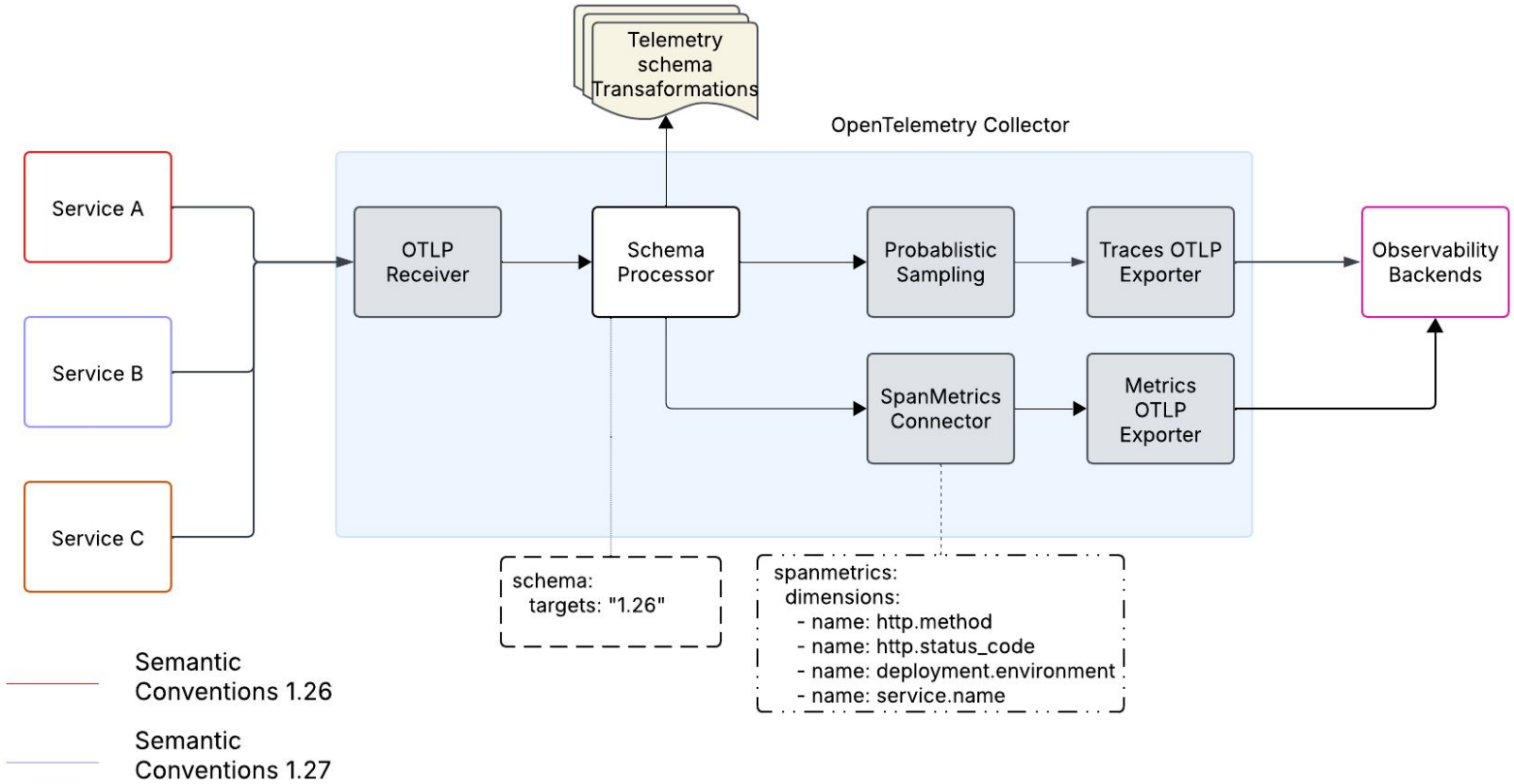
OpenTelemetry Collector pipeline



OpenTelemetry Collector pipeline



Schema Processor



Limitations of schema processor

- Actively in Development
- Schema_url is optional field
- Breaking changes are not supported
- Telemetry Schema File 1.0
  - Doesn’t handle Complex transformations
  - Updated Manually with no checks

Next Steps

What are we working on

What we are working on

- Multi-registry support
- New schema file format (self-contained registry, back/forward transformations)
- Stabilizing schema processor
- OTEL Weaver live-check (compliance and coverage testing)
- Type-safe client SDK generation
- Schema-first approach and Observability by Design

BE A PART OF THE JOURNEY

CNCF Slack:

- #otel-semantic-conventions
- #otel-weaver

Github Repos:

- Semantic Conventions
- Weaver
- Schema Processor

QUESTIONS ??