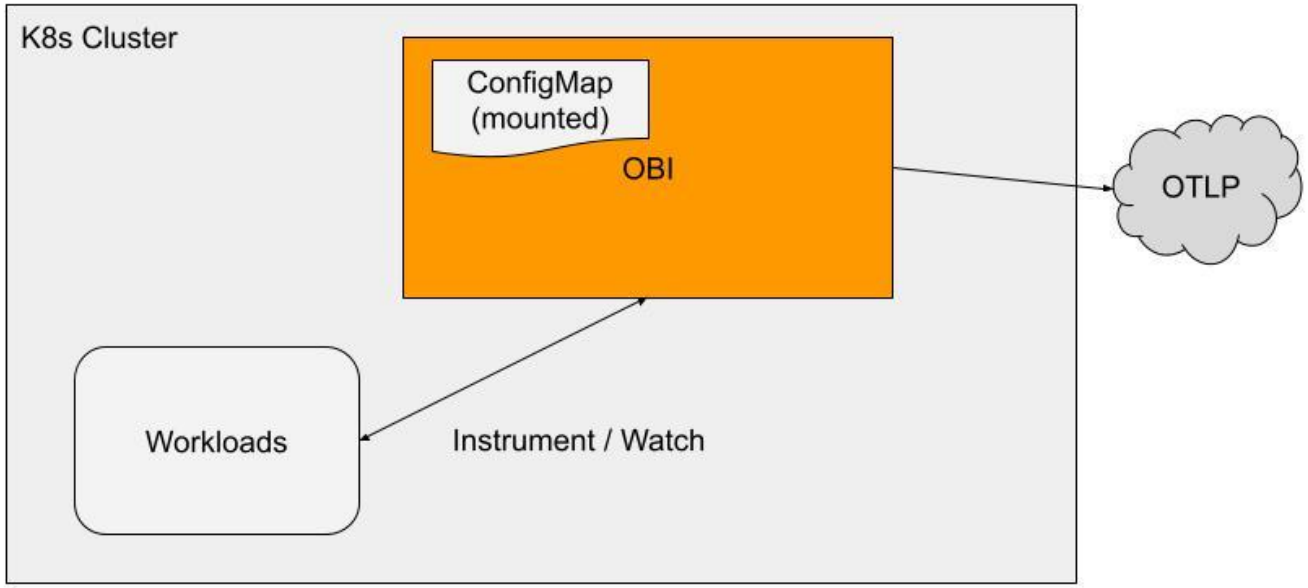
Observability Day
NORTH AMERICA

Building Observability Control Planes with OpenTelemetry, eBPF, and OBI

Mike Dame, Odigos
Rafael Roquetto, Isovalent

OBI as you've seen it

- Auto-instrumentation with eBPF
- Running OBI as an agent, DaemonSet, image, etc
- Single static config surface
- Off the shelf solution similar to collector-contrib
- But what if you need more/custom?



OBI as an Instrumentation Engine

Observability Day
NORTH AMERICA

- What if you want:
  - Distributed config / Instrumentation as Code (GitOps, Argo, etc)
  - Your own API
  - Custom discovery (operators, AI)
  - No restarts

- OBI already manages eBPF programs and instrumentation primitives – don't need to reinvent

- Discovery + Selection are not necessarily the engine's responsibility

What it looks like

OBI as an Instrumentation Engine

Bring your own control plane. Let OBI handle the instrumentation.

PLATFORM / CONTROL PLANE

- **Kubernetes** (Operators, Controllers)
- **Internal Platform** (Developer Platforms)
- **Observability Vendor** (Products, SaaS)

Decides what to instrument based on your domain: workloads, policies, SLOs, user requests, and more.

↓

Instrumentation Intent
(PIDs, configuration, policies)

↓

OpenTelemetry | **OBI** INSTRUMENTATION ENGINE

Discovery, instrumentation, protocol detection, telemetry generation

Takes intent and handles the hard parts: eBPF instrumentation, protocol detection, and OpenTelemetry output.

↓

eBPF
Kernel instrumentation

Safe, low-overhead instrumentation with eBPF.

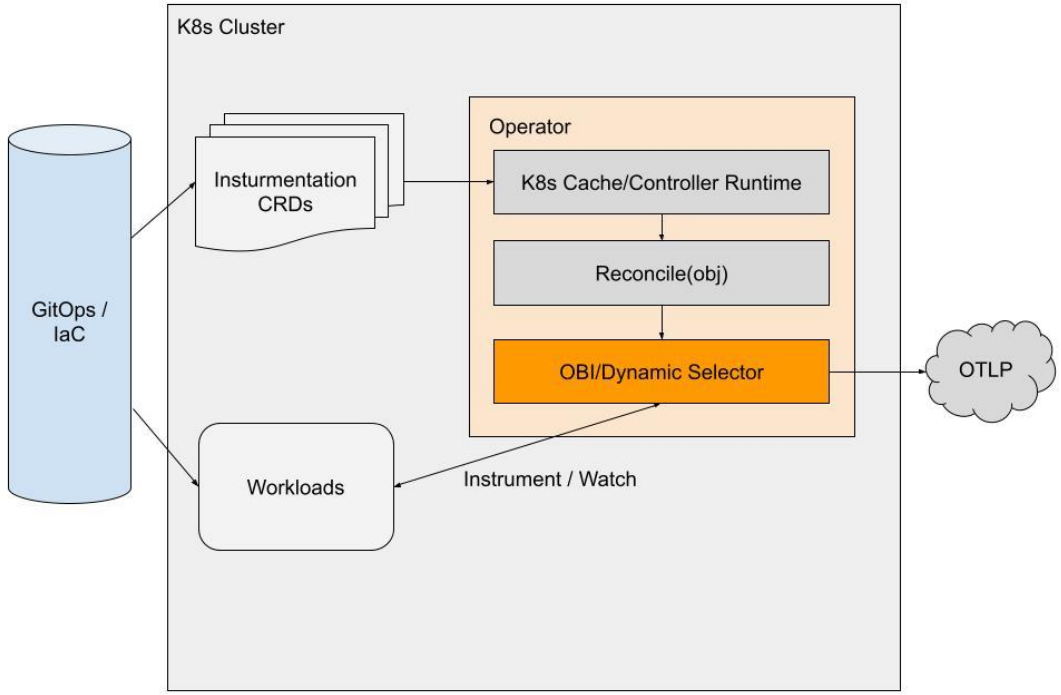
↓

YOUR APPLICATIONS

- Pods
- Services
- Containers
- Applications

Your workloads, automatically instrumented with the right signals, at the right time.

Simple Example



OBI in a nutshell

Configure what to instrument. OBI handles the rest.

Static selection

`YAML / env`

exe · port · PID
workloads / labels

→

`instrumenter.Run(ctx, &config)`

Orchestrates discovery, probes and telemetry

Discover
& select

→

Attach
eBPF

→

Process
& export

→

OTLP

Traces
+ metrics

Static rules ≠ one-time discovery: new matching processes are still found.

Next: WithDynamicSelector() updates targets — the engine stays the same.

Dynamic Selector API

Observability Day
NORTH AMERICA

- Added in OBI v0.7.0
- Import OBI main thread and pass a `WithDynamicSelector()`
- Supports per-signal selection, PID, and K8s workloads

```
selector := discover.NewDynamicSelector()

go instrumenter.Run(ctx, &config,
    instrumenter.WithDynamicSelector(selector),
)

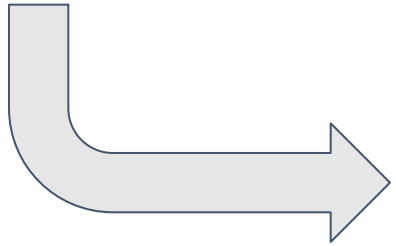
selector.Traces().AddK8sWorkload(workload, opts)
selector.ApplicationMetrics().AddK8sWorkload(workload, opts)
selector.NetworkMetrics().AddK8sWorkload(workload, opts)
```

Dynamic Selection (How it works)

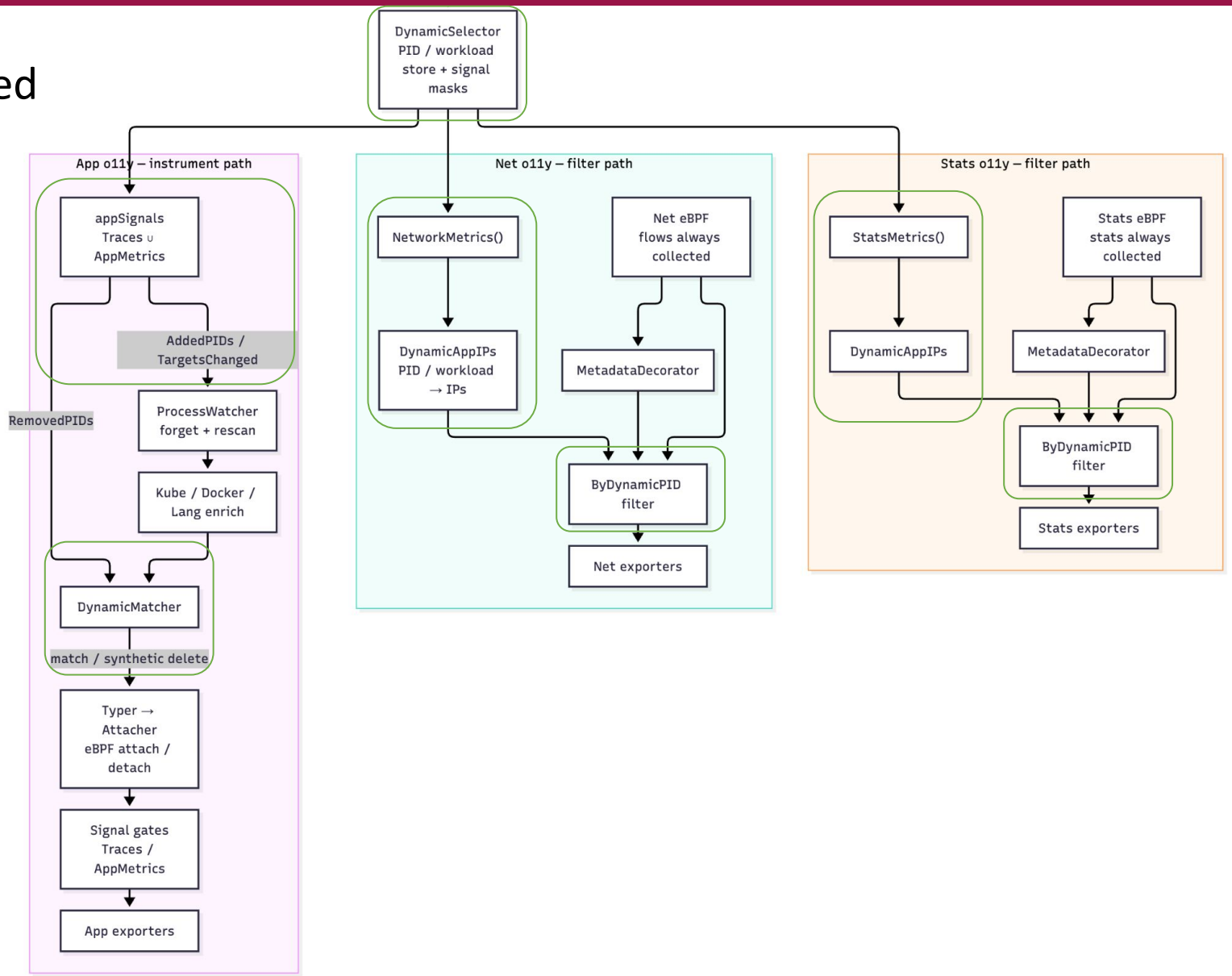
Observability Day
NORTH AMERICA

- Uses existing OBI signal PID/IP-based pipelines+discovery with synthetic async updates as called by API

- Learn more in our talk tomorrow:
  From Bytes to Traces: A Deep Dive into OpenTelemetry eBPF Instrumentation Internals







Vendoring OBI (3 easy steps)

Observability Day
NORTH AMERICA

1. **Import Go packages**
   a. Instrumenter + Dynamic Selector API

2. **Update manifests**
   a. Permissions, mounts, (match OBI)

3. **Build your image**
   a. Generated files required for eBPF

1. Import+Use Go packages

Observability Day
NORTH AMERICA

```
import (
    "go.opentelemetry.io/obi/pkg/obi"
    "go.opentelemetry.io/obi/pkg/instrumenter"
    "go.opentelemetry.io/obi/pkg/appolly/discover"
    "go.opentelemetry.io/obi/pkg/selection"
    // ...
)

func main() {
    config := obi.DefaultConfig
    selector := discover.NewDynamicSelector()

    go instrumenter.Run(ctx, &config,
        instrumenter.WithDynamicSelector(selector),
    )

    // ... Kubernetes controller reconciles workloads ...

    checkout := &appsv1.Deployment{
        ObjectMeta: metav1.ObjectMeta{
            Namespace: "payments",
            Name:      "checkout",
        },
    }

    selector.AddK8sWorkload(checkout, selection.DynamicOptions{
        ServiceName:      "checkout-deployment",
        ServiceNamespace: "payments",
    })

    // ... Workload no longer selected ...

    selector.RemoveK8sWorkload(checkout)
}
```

Dynamic Selector handle

Main OBI instrumentation thread

Select workload

2. Update manifests

Linux capabilities: which operation are we authorizing?

Load & attach eBPF

`CAP_BPF + CAP_PERFMON`

BPF maps, tracing probes

Inspect processes

`CAP_SYS_PTRACE` · `CAP_DAC_READ_SEARCH`

`CAP_CHECKPOINT_RESTORE`

/proc, executables, symbols

Network hooks

`CAP_NET_RAW` → socket filters

`CAP_NET_ADMIN` → TC / propagation

Conditional / platform-specific

`CAP_SYS_ADMIN` → Go user-memory writes

`CAP_SYS_RESOURCE` → old memlock limits

Separate from Linux capabilities

Host visibility & mounts

hostPID · hostNetwork · cgroup · tracefs

Kubernetes API access

ServiceAccount RBAC to watch workloads

3. Build your image

- Need to include compiled **.o* binaries and generated Go code in your module path
- Generator image: *ghcr.io/open-telemetry/obi-generator*
- OBI releases also publish pre-generated artifacts

```
# 1) Checkout a released OBI tag and generate *_bpfel.go / *.o
FROM ghcr.io/open-telemetry/obi-generator:0.2.17 AS obi
ARG OBI_VERSION=v0.14.0
RUN git clone --depth 1 --branch ${OBI_VERSION} \
      https://github.com/open-telemetry/opentelemetry-ebpf-instrumentation /src
WORKDIR /src
RUN make generate    # entrypoint wires BPF2GO / clang for you

# 2) Build your agent against that tree (module cache alone is not enough)
FROM golang:1.25 AS builder
COPY --from=obi /src /deps/obi
COPY . .
RUN go mod edit -replace go.opentelemetry.io/obi=/deps/obi \
 && CGO_ENABLED=0 go build -o /agent ./cmd/...
```

1. Generate

2. Copy+replace

3.2 - Building with submodules

Beyla: pin OBI source, generate artifacts, then vendor

1 Pin source

.obi-src/
Submodule pins OBI commit

→

2 Generate

make vendor-obi
BPF objects + vendor/

→

3 Compile

go build -mod vendor
Beyla executable

```
go.mod   replace go.opentelemetry.io/obi => ./.obi-src
```

✓ Pinned source + matching generated BPF artifacts

↳ Cost: submodule updates, code generation, vendoring and CI

Tangent: eBPF Artifacts

- See https://github.com/open-telemetry/opentelemetry-go-instrumentation/issues/1233
- **tl;dr** eBPF code + generated Go doesn't ship well through Git like other idiomatic Go packages
- We don't want to commit binaries to Git, but want Go to be usable out of the box
- OBI uniquely sits somewhere between "standard Go project" and "standard eBPF project"

Summary

- Automatic instrumentation isn't one approach fits all
- Automatic instrumentation can be part of programmable infra
- OBI does what it does well, but that shouldn't stop you from trying to build more on top of it!
- Dynamic Selector API is young, subject to change, and we encourage new use cases and contributions:
  - VM, runtime config (beyond just workload selection), instrumentation

Observability Day
NORTH AMERICA

Thank you!