

Where's the “Auto” in Auto-Instrumentation?

Josh Lee • Apr 30 2025 • OpenSearchCon Europe

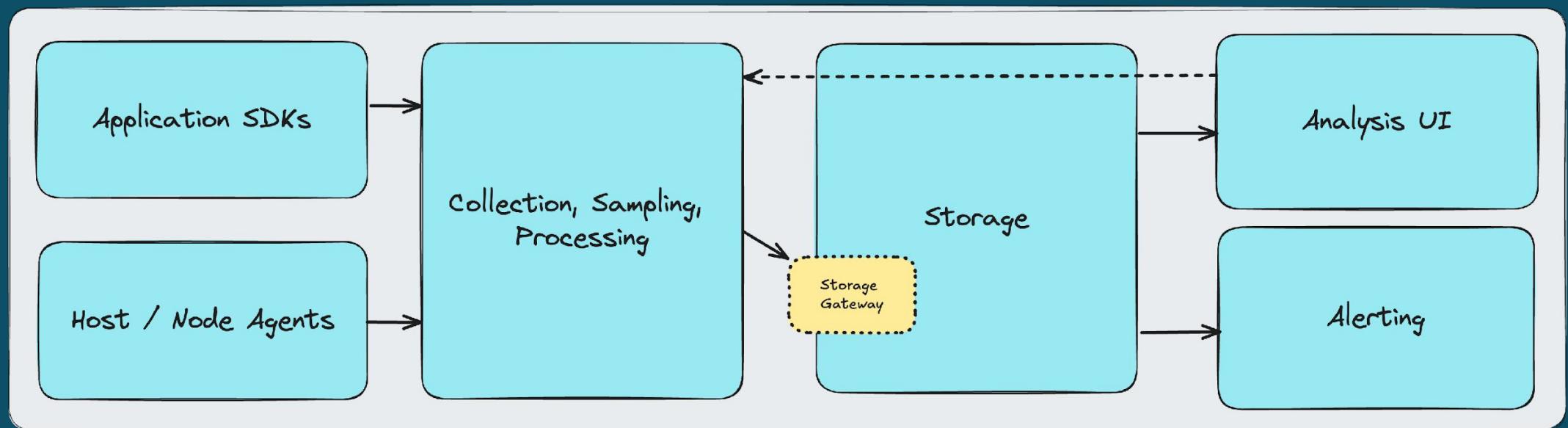


Josh Lee
*Open Source
Advocate @ Altinity*

Altinity® is a Registered Trademark of Altinity, Inc. ClickHouse® is a registered trademark of ClickHouse, Inc.; Altinity is not affiliated with or associated with ClickHouse, Inc.

Observability is our
ability to understand
a system from its
outputs alone

A complete observability solution



Questions about Auto-Instrumentation

Do I need to
modify my
application
code?

Do I need to use
specific libraries?

Will it work
without
Kubernetes?

It depends...

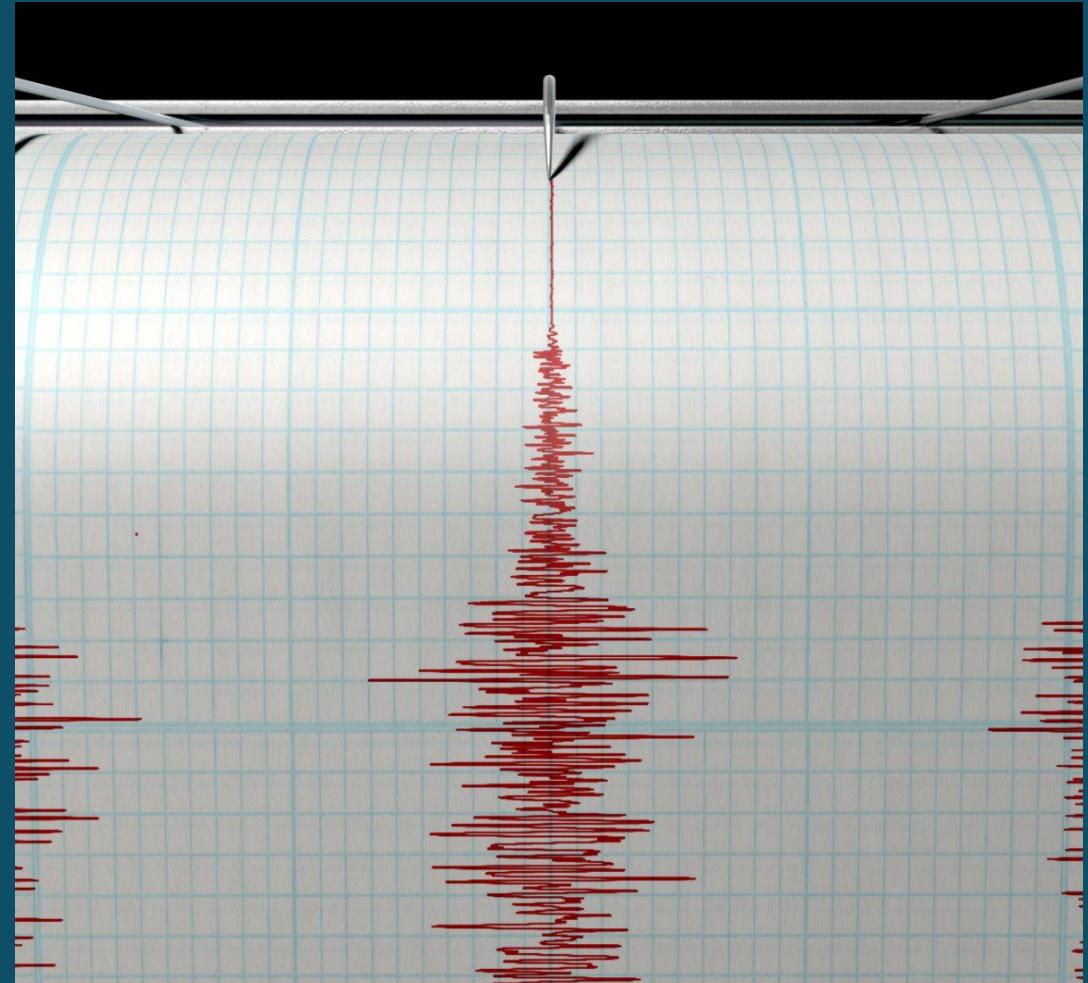
- 01 —— What are we trying to do with instrumentation?
What is “auto-instrumentation”?
Hint: there is more than one right answer

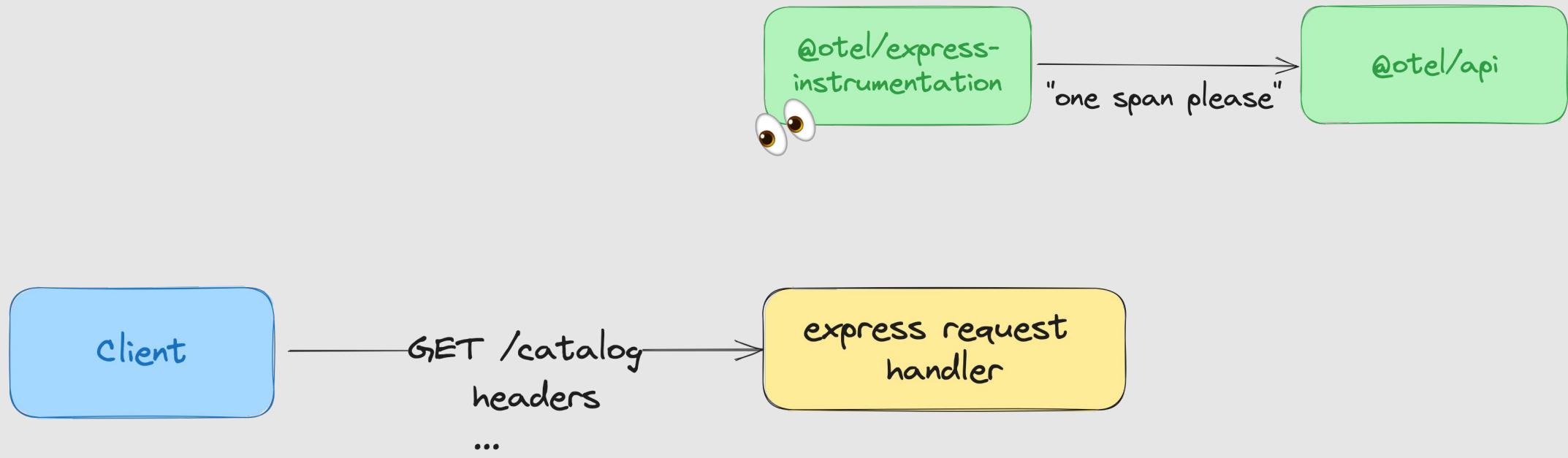
- 02 —— Different kinds of auto-instrumentation
The Kubernetes Operator

- 03 —— When is Auto-Instrumentation not enough?
When is it awesome?



Instrumentation is
the process of
translating
interesting things
into telemetry
signals





Interesting events

Requests,
Queries,
& Messages

Errors,
Exceptions,
& Events

Function Calls
(with arguments)

Contextualizing metadata

User Context

Who are they and
what are they
trying to do?

Infrastructure Context

What is the state
of the resources?

Organizational Context

Who is
responsible?

Instrumentation in OpenTelemetry

- > Direct API Calls
- > Instrumentation Libraries
- > Auto-Instrumentation...

```
1 // Copyright The OpenTelemetry Authors
2 // SPDX-License-Identifier: Apache-2.0
3
4 const opentelemetry = require('@opentelemetry/sdk-node');
5 const {getNodeAutoInstrumentations} = require('@opentelemetry/auto-instrumentations-node');
6 const {OTLPTraceExporter} = require('@opentelemetry/exporter-trace-otlp-grpc');
7 const {OTLPMetricExporter} = require('@opentelemetry/exporter-metrics-otlp-grpc');
8 const {PeriodicExportingMetricReader} = require('@opentelemetry/sdk-metrics');
9 ...
10
11 const sdk = new opentelemetry.NodeSDK({
12   traceExporter: new OTLPTraceExporter(),
13   instrumentations: [
14     getNodeAutoInstrumentations(...)
15   ],
16   metricReader: new PeriodicExportingMetricReader({
17     exporter: new OTLPMetricExporter(),
18   }),
19   resourceDetectors: [
20     ...
21   ],
22 });
23
24 sdk.start();
```

Instrumentation in OpenTelemetry

- > OpenTelemetry SDK
- > OpenTelemetry API
- > Instrumentation Libraries

```
1 // Copyright The OpenTelemetry Authors
2 // SPDX-License-Identifier: Apache-2.0
3
4 const opentelemetry = require('@opentelemetry/sdk-node');
5 const {getNodeAutoInstrumentations} = require('@opentelemetry/auto-instrumentations-node');
6 const {OTLPTraceExporter} = require('@opentelemetry/exporter-trace-otlp-grpc');
7 const {OTLPMetricExporter} = require('@opentelemetry/exporter-metrics-otlp-grpc');
8 const {PeriodicExportingMetricReader} = require('@opentelemetry/sdk-metrics');
9 ...
10
11 const sdk = new opentelemetry.NodeSDK({
12   traceExporter: new OTLPTraceExporter(),
13   instrumentations: [
14     getNodeAutoInstrumentations(...)
15   ],
16   metricReader: new PeriodicExportingMetricReader({
17     exporter: new OTLPMetricExporter(),
18   }),
19   resourceDetectors: [
20     ...
21   ],
22 });
23
24 sdk.start();
```

... Isn't that Auto-Instrumentation?

Instrumentation
Libraries *target*
specific libraries and
modify them (or
observe them) to call
the OTel APIs,
creating telemetry

opentelemetry-instrumentation-bunyan	chore: release main (#2497)
opentelemetry-instrumentation-cassandra	chore: release main (#2497)
opentelemetry-instrumentation-connect	chore: release main (#2497)
opentelemetry-instrumentation-dns	chore: release main (#2497)
opentelemetry-instrumentation-express	chore: release main (#2497)
opentelemetry-instrumentation-fastify	chore: release main (#2497)
opentelemetry-instrumentation-generic-p...	chore: release main (#2497)
opentelemetry-instrumentation-graphql	chore: release main (#2497)
opentelemetry-instrumentation-hapi	chore: release main (#2497)
opentelemetry-instrumentation-ioredis	chore: release main (#2497)
opentelemetry-instrumentation-knex	chore: release main (#2497)
opentelemetry-instrumentation-koa	chore: release main (#2497)
opentelemetry-instrumentation-memcached	chore: release main (#2497)
opentelemetry-instrumentation-mongodb	chore: release main (#2497)
opentelemetry-instrumentation-mysql	chore: release main (#2497)
opentelemetry-instrumentation-mysal2	chore: release main (#2497)

What is Auto-Instrumentation?

“Auto-Instrumentation”

Meta Packages

Packages that can be configured to automatically include relevant instrumentation libraries based on the presence of other libraries.

No-Code Instrumentation

Agents + Extensions

Mechanisms for adding instrumentation to an application package after it has already been compiled or bundled.

Instrumentation-Injection

(w/ Kubernetes Operator)

The Kubernetes Operator can automatically inject no-code instrumentation into matching workloads.

Mechanisms

Code-based Library-Instrumentation

Observe other modules in the same process without directly modifying their source, and emit telemetry*

No-Code Instrumentation

Modify or observe another program (binary, bytecode, etc.) and emit telemetry*

Code-based
Library-instrumentation

Monkey Patching

Function Wrapping

Middleware Injection

Event Observation

No-Code
Instrumentation Mechanisms

Monkey Patching (JS, Python)

Runtime Agent (Java, .NET)

Interpreter Extension (PHP)

eBPF (Go)

Monkey Patching

```
new InstrumentationNodeModuleDefinition(
  'express',
  ['>=4.0.0 <5'],
  moduleExports => {
    const routerProto = moduleExports.Router as unknown as express.Router;
    // patch express.Router.route
    if (isWrapped(routerProto.route)) {
      this._unwrap(routerProto, 'route');
    }
    this._wrap(routerProto, 'route', this._getRoutePatch());
    ...
    return moduleExports;
  },
  moduleExports => {
    if (moduleExports === undefined) return;
    const routerProto = moduleExports.Router as unknown as express.Router;
    this._unwrap(routerProto, 'route');
    this._unwrap(routerProto, 'use');
    this._unwrap(moduleExports.application, 'use');
  }
),
];
```

Function Wrapping

```
1 private static function _hook(..., ?string $class, string $function, string $name, int $spanKind = SpanKind::KIND_SERVER): void
2 {
3     hook(
4         class: $class,
5         function: $function,
6         pre: static function ($object, ?array $params, ?string $class, ?string $function, ?string $filename, ?int $lineno) use ($instrumentation, $name, $spanKind) {
7             $span = self::builder($instrumentation, $name, $function, $class, $filename, $lineno)
8                 ->setSpanKind($spanKind)
9                 ->startSpan();
10            Context::storage()->attach($span->storeInContext(Context::getCurrent()));
11        },
12        post: static function ($object, ?array $params, mixed $return, ?Throwable $exception) {
13            self::end($exception);
14        }
15    );
16 }
```

Middleware Injection

```
from fastapi import FastAPI
from opentelemetry.instrumentation.fastapi import FastAPIInstrumentor
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import ConsoleSpanExporter, SimpleSpanProcessor

# Setup OpenTelemetry tracing provider
provider = TracerProvider()
provider.add_span_processor(SimpleSpanProcessor(ConsoleSpanExporter()))

# Create FastAPI app
app = FastAPI()

# Inject middleware explicitly
FastAPIInstrumentor.instrument_app(app, tracer_provider=provider)

@app.get("/")
async def read_root():
    return {"Hello": "Middleware injection!"}
```

Event Observation

```
@doc false
def attach_router_start_handler(_opts) do
  :telemetry.attach(
    {__MODULE__, :router_dispatch_start},
    [:phoenix, :router_dispatch, :start],
    &__MODULE__.handle_router_dispatch_start/4,
    %{}
  )
end
```

No-Code Instrumentation Mechanisms

No-Code Instrumentation

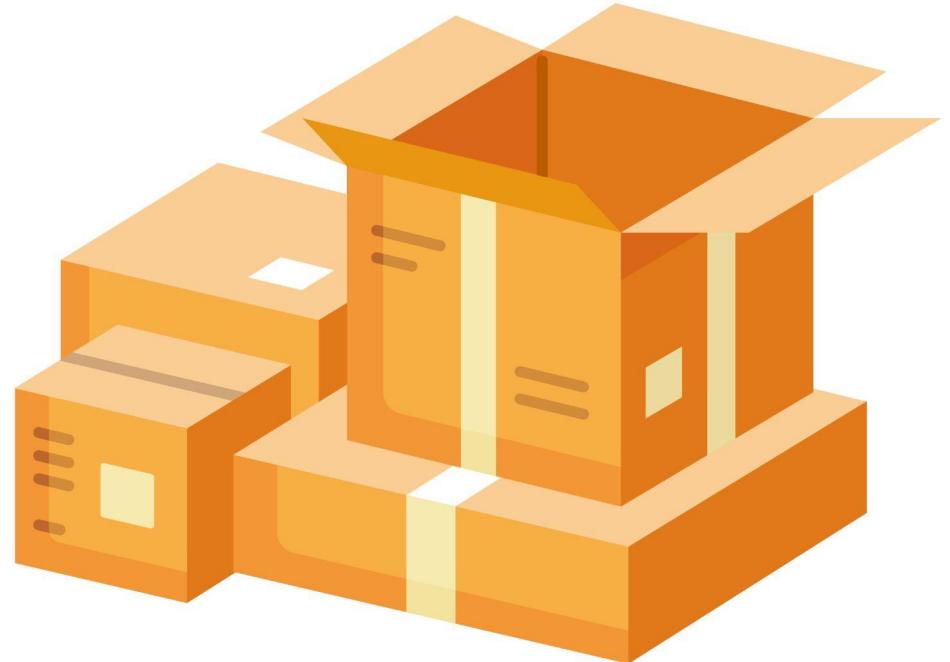
```
export OTEL_SERVICE_NAME=my-service  
export OTEL_TRACES_SAMPLER=parentbased_traceidratio  
export OTEL_TRACES_SAMPLER_ARG=0.5  
export OTEL_INSTRUMENTATION_HTTP_CLIENT_CAPTURE_HEADERS=true
```

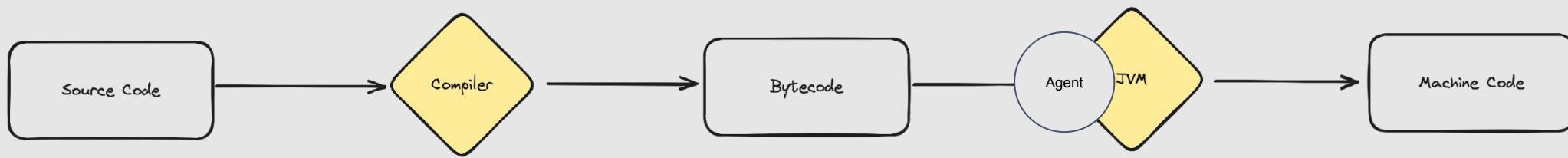
	Instrumentation Libraries	No-Code Instrumentation	Mechanism
Java	✓	✓	Java Agent + Bytecode Injection
Python	✓	✓	Python Agent + Monkey patching
JavaScript	✓	✓	Monkey patching
.NET	✓	✓	.NET Profiler + Bytecode Injection
Go	✓	✓ *	eBPF
PHP	✓	✓	Interpreter Extension + Autoloading
Erlang / Elixir	✓		
Ruby	✓		

Bytecode Injection

Runtime agents can inject
OpenTelemetry SDKs and library
instrumentation at the bytecode level.

Functions can also be wrapped and/or
monkey patched.

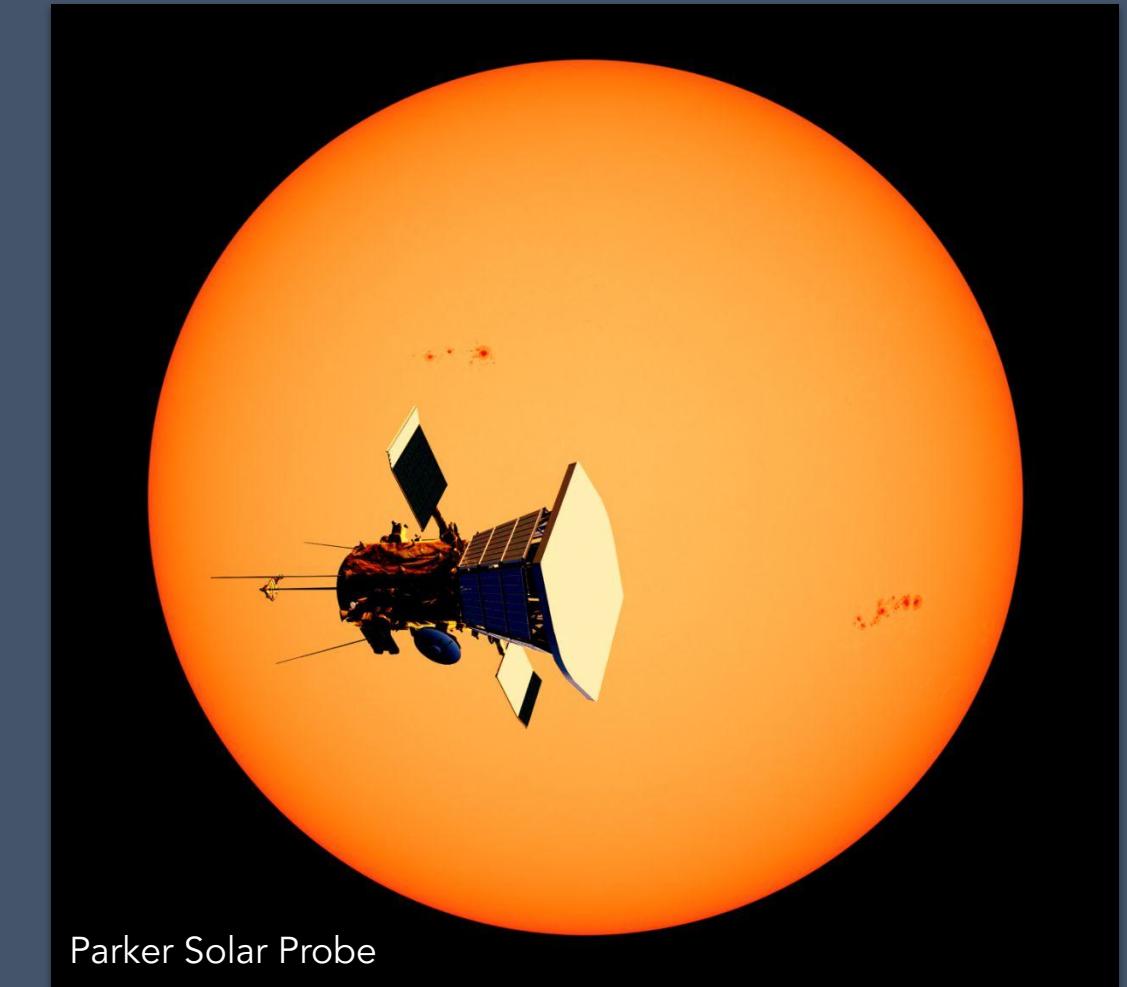




eBPF

Compiled binaries can't be modified with instrumentation.

eBPF + uProbes offer visibility.



Parker Solar Probe

eBPF Example

```
//go:build ignore

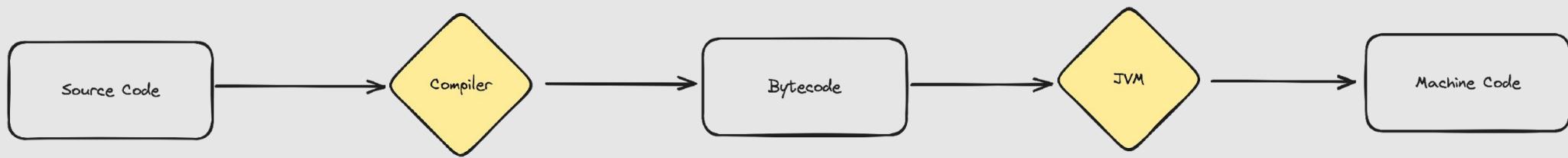
#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>

struct {
    __uint(type, BPF_MAP_TYPE_ARRAY);
    __type(key, __u32);
    __type(value, __u64);
    __uint(max_entries, 1);
} pkt_count SEC(".maps");

// count_packets atomically increases a packet counter on every invocation.
SEC("xdp")
int count_packets() {
    __u32 key      = 0;
    __u64 *count = bpf_map_lookup_elem(&pkt_count, &key);
    if (count) {
        __sync_fetch_and_add(count, 1);
    }

    return XDP_PASS;
}

char __license[] SEC("license") = "Dual MIT/GPL";
```

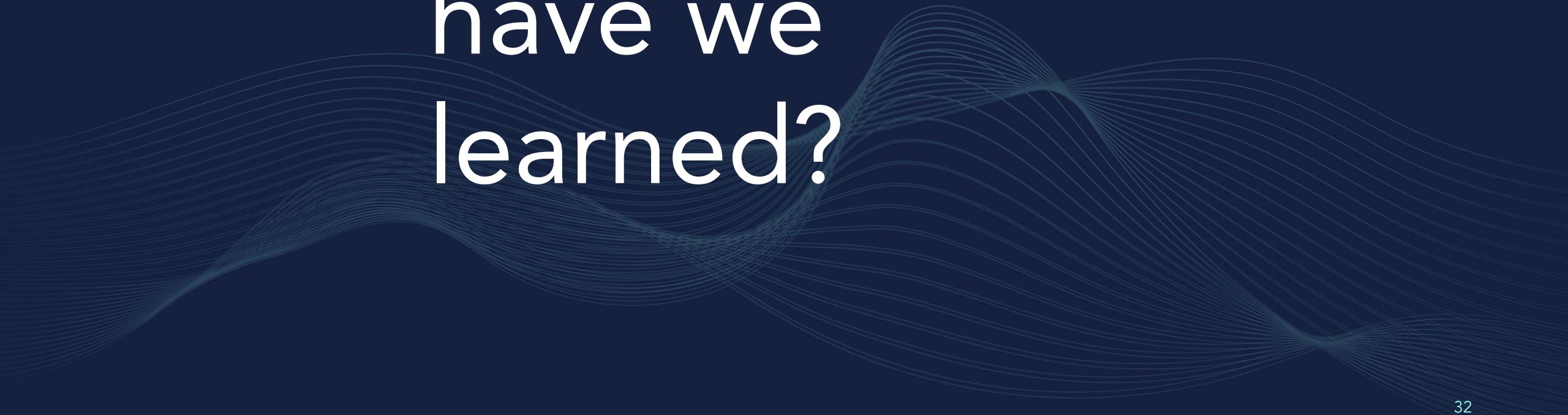


Injecting Instrumentation

Kubernetes Operator

```
kubectl apply -f - <<EOF
apiVersion: opentelemetry.io/v1alpha1
kind: Instrumentation
metadata:
  name: demo-instrumentation
spec:
  exporter:
    endpoint: http://demo-collector:4318
  propagators:
    - tracecontext
    - baggage
  sampler:
    type: parentbased_traceidratio
    argument: "1"
EOF
```

What have we learned?



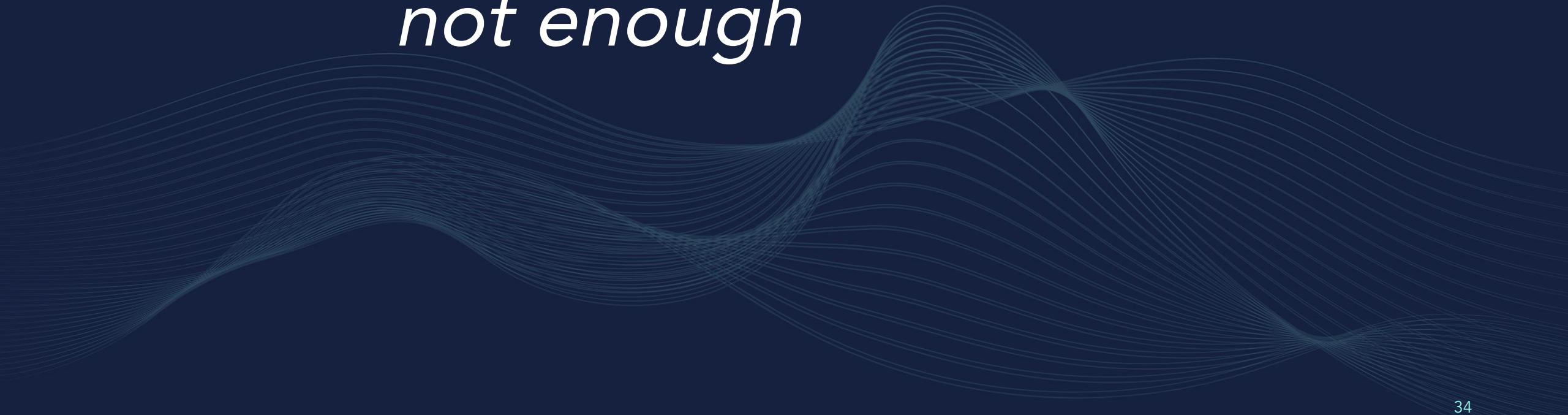
Questions about Auto-Instrumentation

Do I need to
modify my
application
code?

Do I need to use
specific libraries?

Will it work
without
Kubernetes?

When is
Auto-instrumentation
not enough



Essential Custom Spans

"Stack" trace
information

Incompatible
frameworks or
libraries

Especially for
monolithic
services

Essential Manual Annotation

User Experience & Client Details

Details about the user (who they are or what they are trying to do) from the Application or Client contexts.

Business Metrics & Dimensions

Runtime details about business operations (e.g. regions, departments, revenue, cost)

Team / Business Unit Ownership

Especially if it isn't defined by infrastructure context.
Extra useful in monolith-like services.

Incompatible or Bespoke System Architecture

Heavy use of non-HTTP messaging, IPC, RPC *
Compiled languages (besides Go)

When is Auto-Instrumentation *awesome*



When is Auto-Instrumentation *awesome*

Combined with Manual
Annotation

Reduce toil with a consistent
starting point for customization

Fill Gaps in E2E
Distributed Tracing

Legacy services
Skill-gap on central observability
team

Quickly Understand
System Topology

Complete tracing can reveal
architecture details

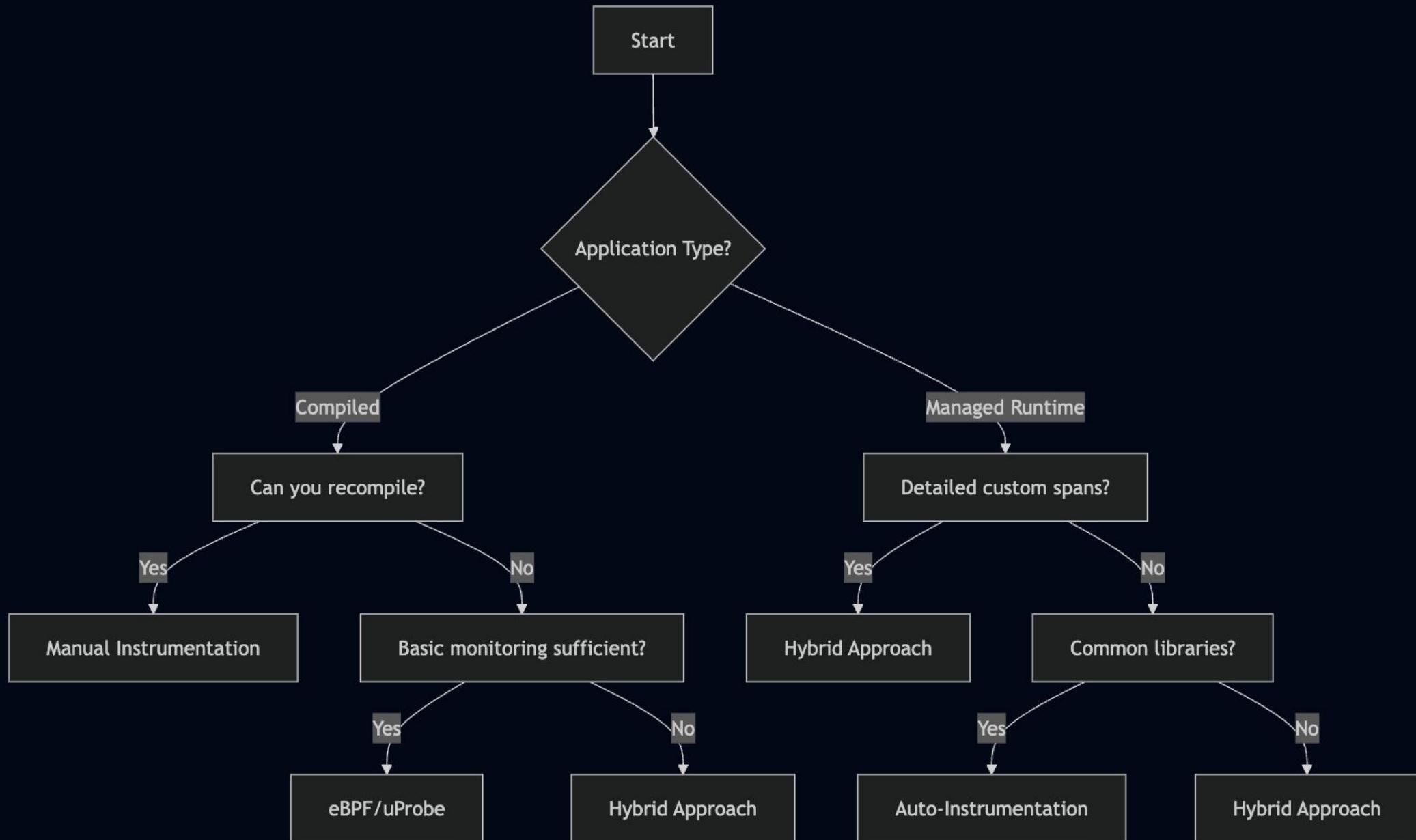
Cannot modify original
source code

Auto-instrumentation is the only
option here

Off-the-shelf OSS components can
be deployed as-is

Auto-Instrumentation Decision Tree





Thank You OpenSearch Community!



Connect with me



We're seeking OSS contributors

