Grafana Labs

From PIDs to Pods: the life cycle of an eBPF-autoinstrumented application

Marc Tuduri
Senior Software Engineer

About me

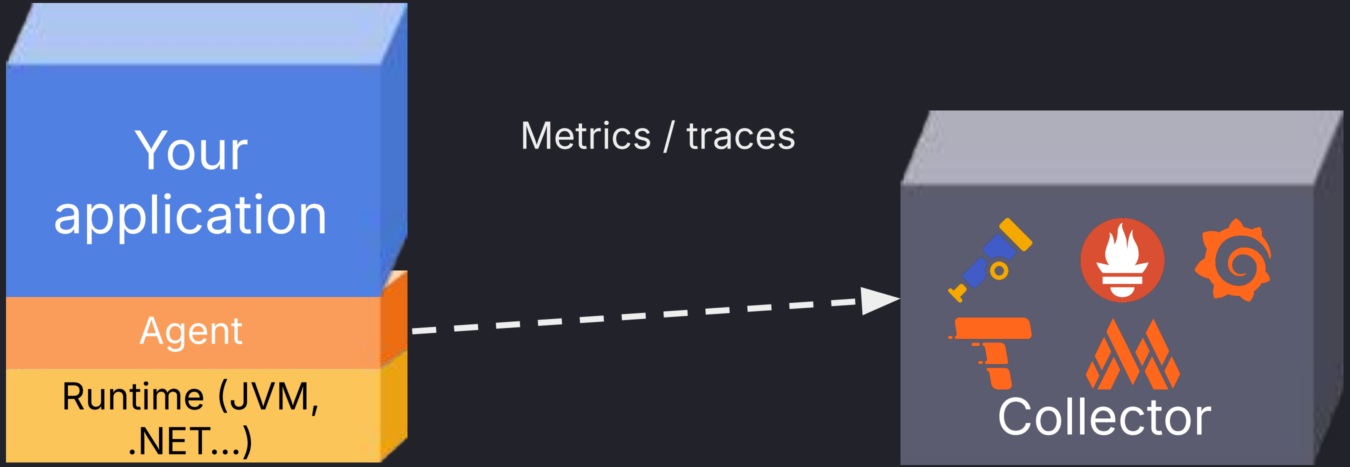
- Working as Software Engineer at Grafana Labs on Beyla project
- Prometheus contributor and OpenTelemetry member
- Currently based in Berlin
- Focused on drumming (but also ex-guitarist and home brewer)

Overview

- Auto-instrumentation with eBPF
- What’s eBPF?
- Instrumenting Kubernetes Applications with eBPF
- The Journey from a PID to a Pod
- Demo
- Future
- Conclusions

Auto-instrumentation with eBPF

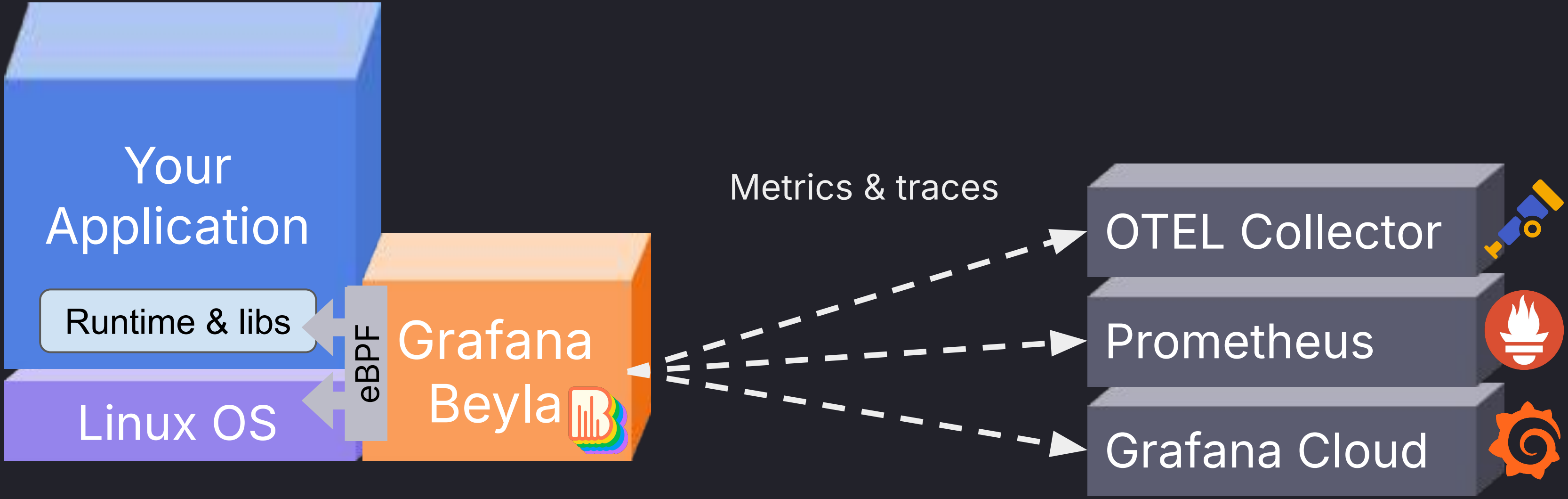
Context: agent-based instrumentation



Agent-based/manual instrumentation: what if...?

- ... my runtime is too old?
- ... too much instrumentation overhead?
- ... my application is a compiled binary?
- ... I don’t want to mess my up code?
- ... I just want instant visibility?

Beyla native eBPF auto-instrumentation



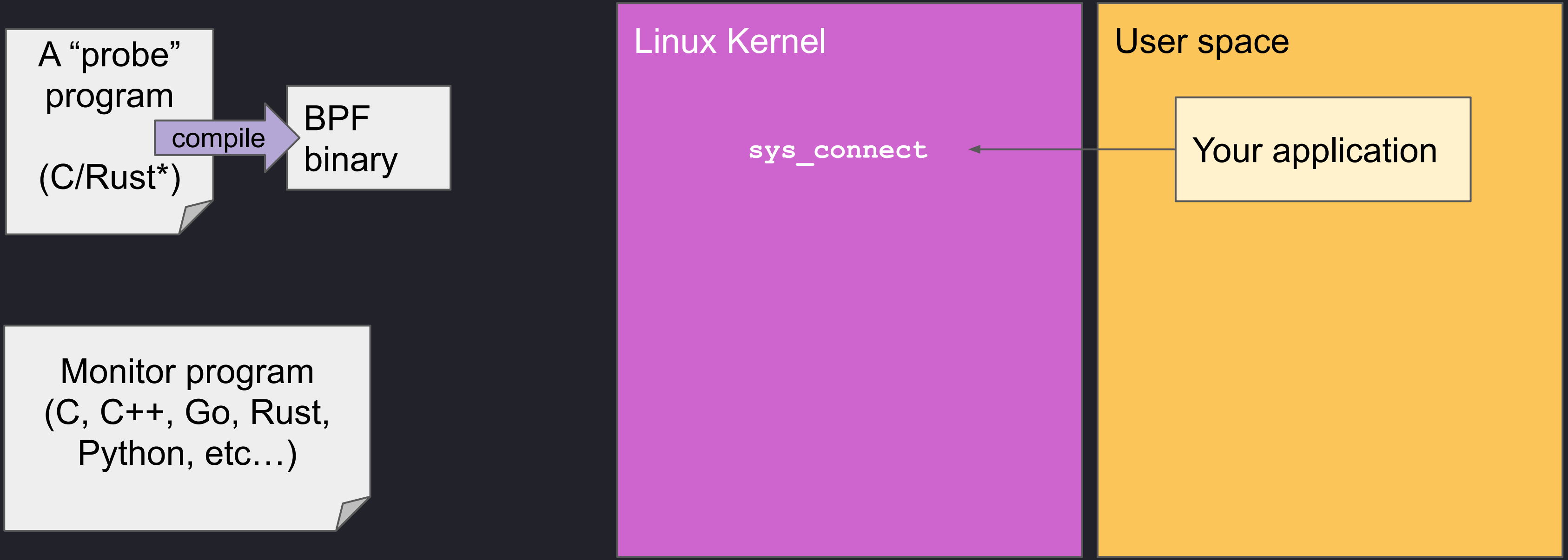
E… B… P… what?

eBPF

- ~~Extended Berkeley Packet Filter~~
  - Virtual Machine built into the Linux Kernel
  - Event-driven programming: “hook” programs into kernel functions and user space programs.
- It requires how the memory is laid out (low-level)
  - Function call arguments
  - Local variables and return values

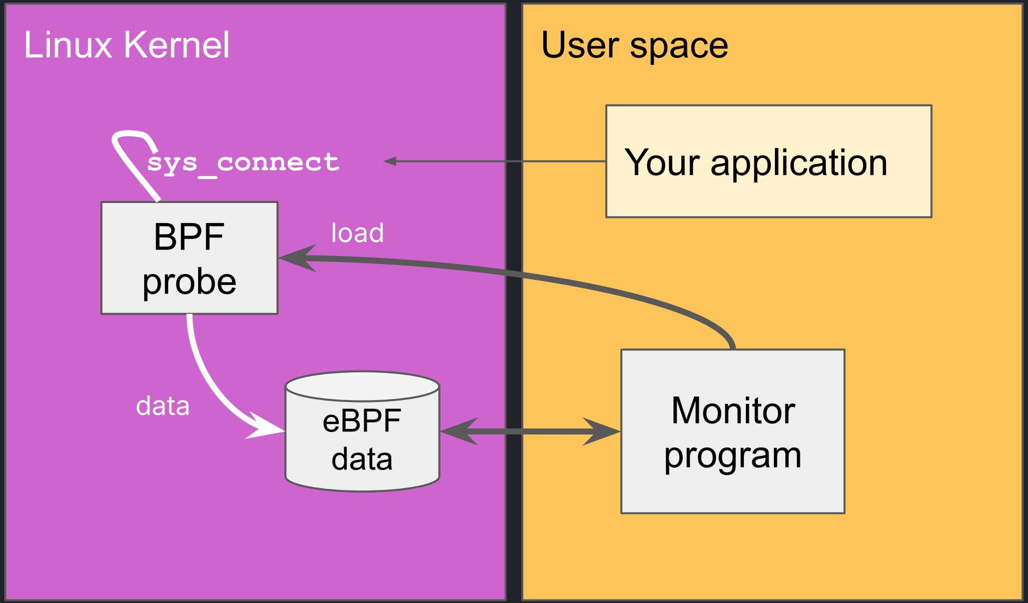
Example: track a new client TCP connection

```
int sys_connect(int fd, struct sockaddr *uservaddr…);
```



Example: track a new client TCP connection

```
int sys_connect(int fd, struct sockaddr *uservaddr…);
```

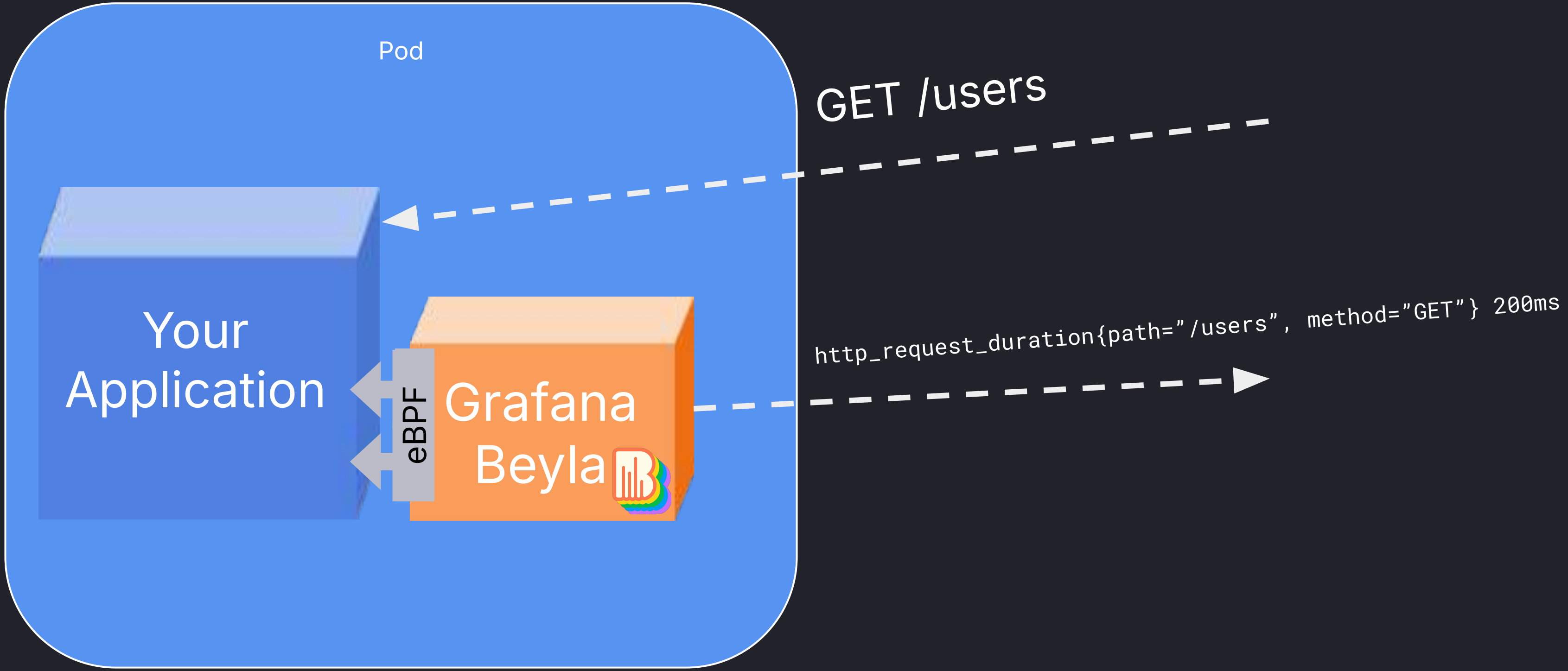


eBPF Pros and Cons

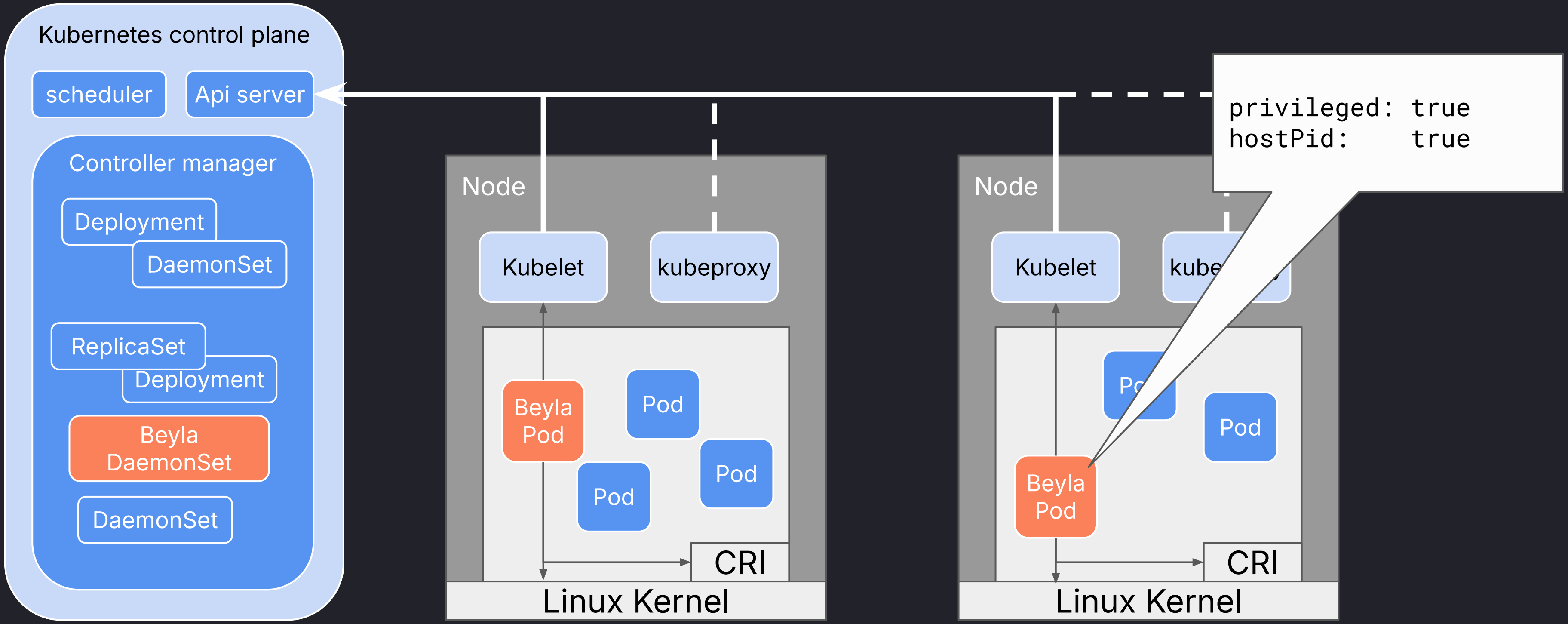
- Pros
  - Fast, JIT compiled probe programs.
  - Safe, all programs are verified at load time by the Kernel.
  - Easy cleanup, once the monitor terminates, all resources are automatically deallocated.
- Cons
  - Hard to debug and write.
  - Architecture dependent.
  - Depending on the used eBPF functions, it requires elevated permissions.

Instrumenting Kubernetes Applications with eBPF

Basic Idea


Pod
GET /users
Your Application
eBPF
Grafana Beyla
http_request_duration{path="/users", method="GET"} 200ms

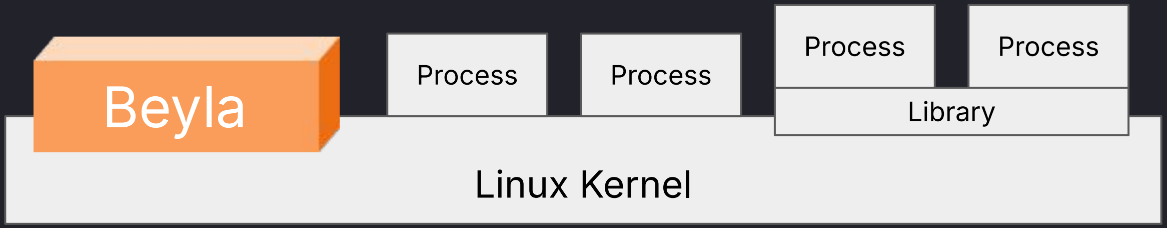
Kubernetes cluster architecture



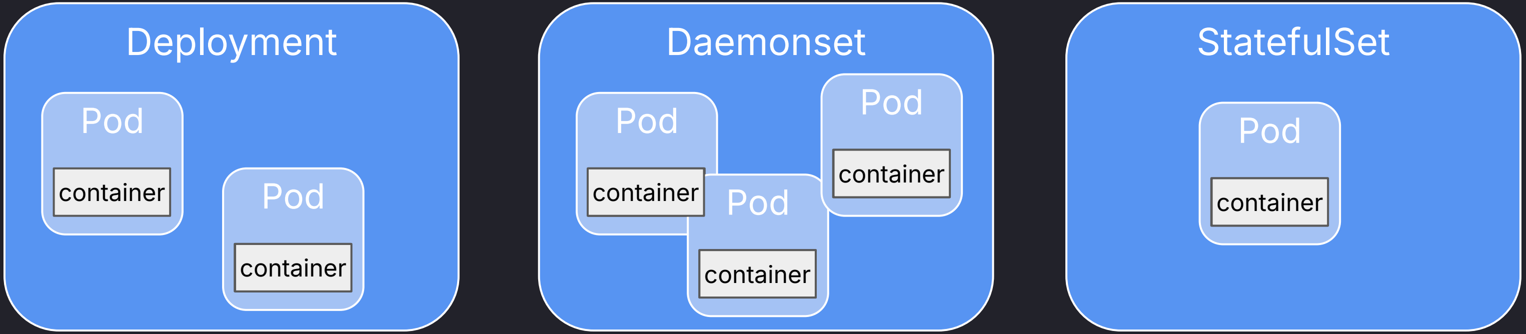
```
privileged: true
hostPid:    true
```

What Beyla directly sees

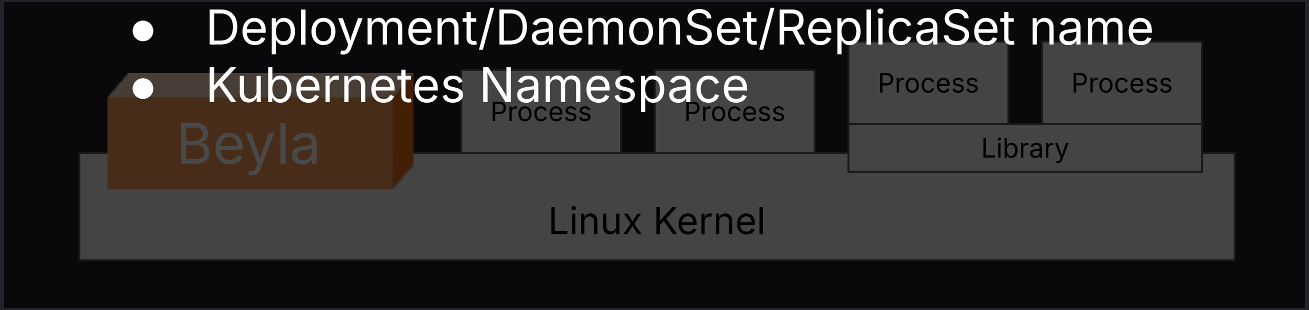
- Command name
- Process ID (e.g. 12145)
- Host Name
- …



What users actually need

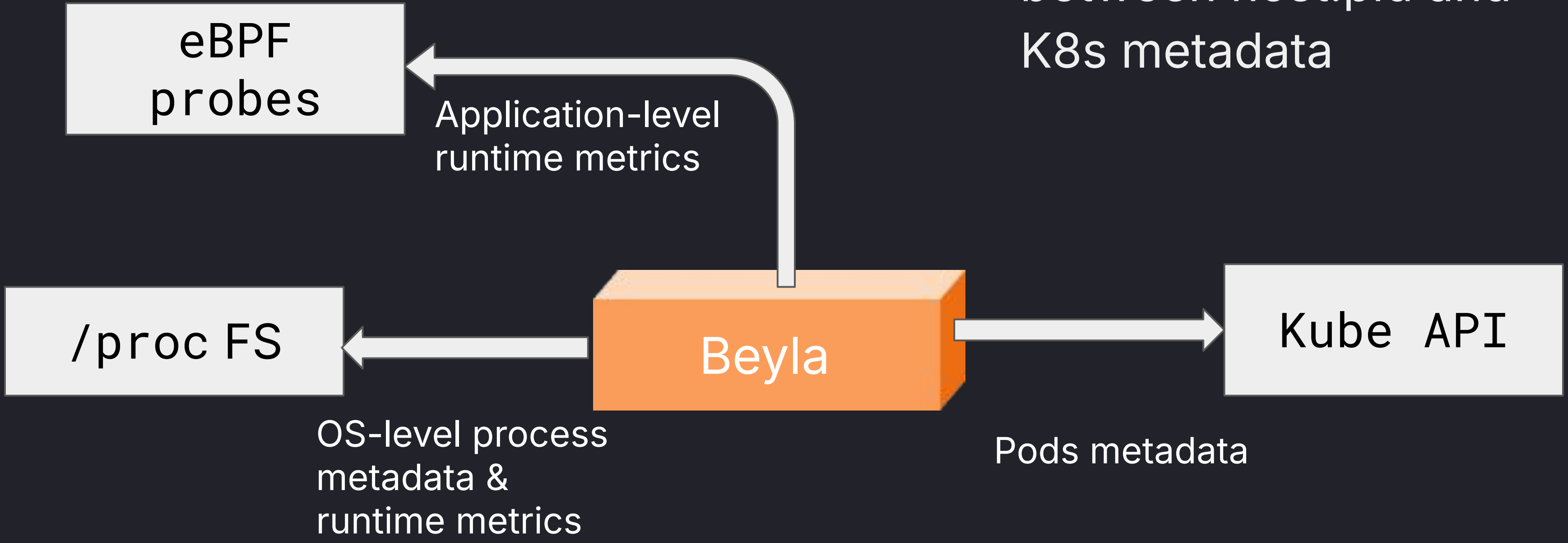


- Pod name & metadata
- Node name
- Deployment/DaemonSet/ReplicaSet name
- Kubernetes Namespace



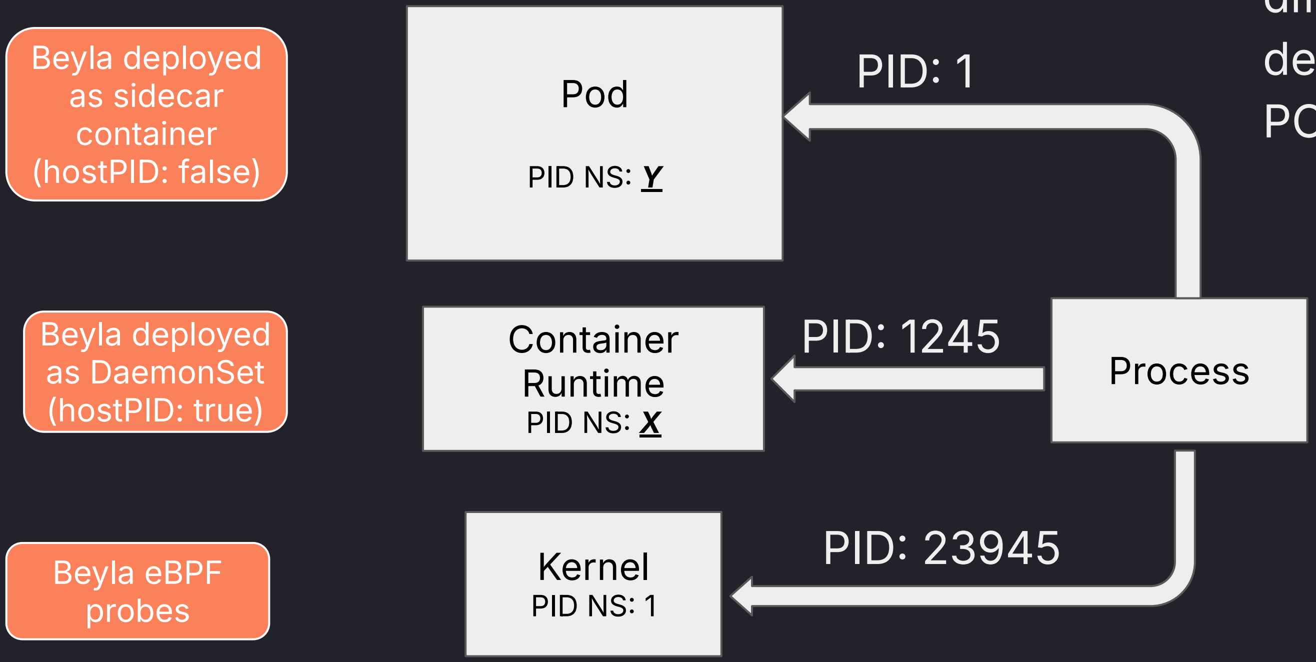
The Journey from a PID to a Pod

Matching processes with Kubernetes metadata


eBPF probes
Application-level runtime metrics
/proc FS
Beyla
Kube API
OS-level process metadata & runtime metrics
Pods metadata

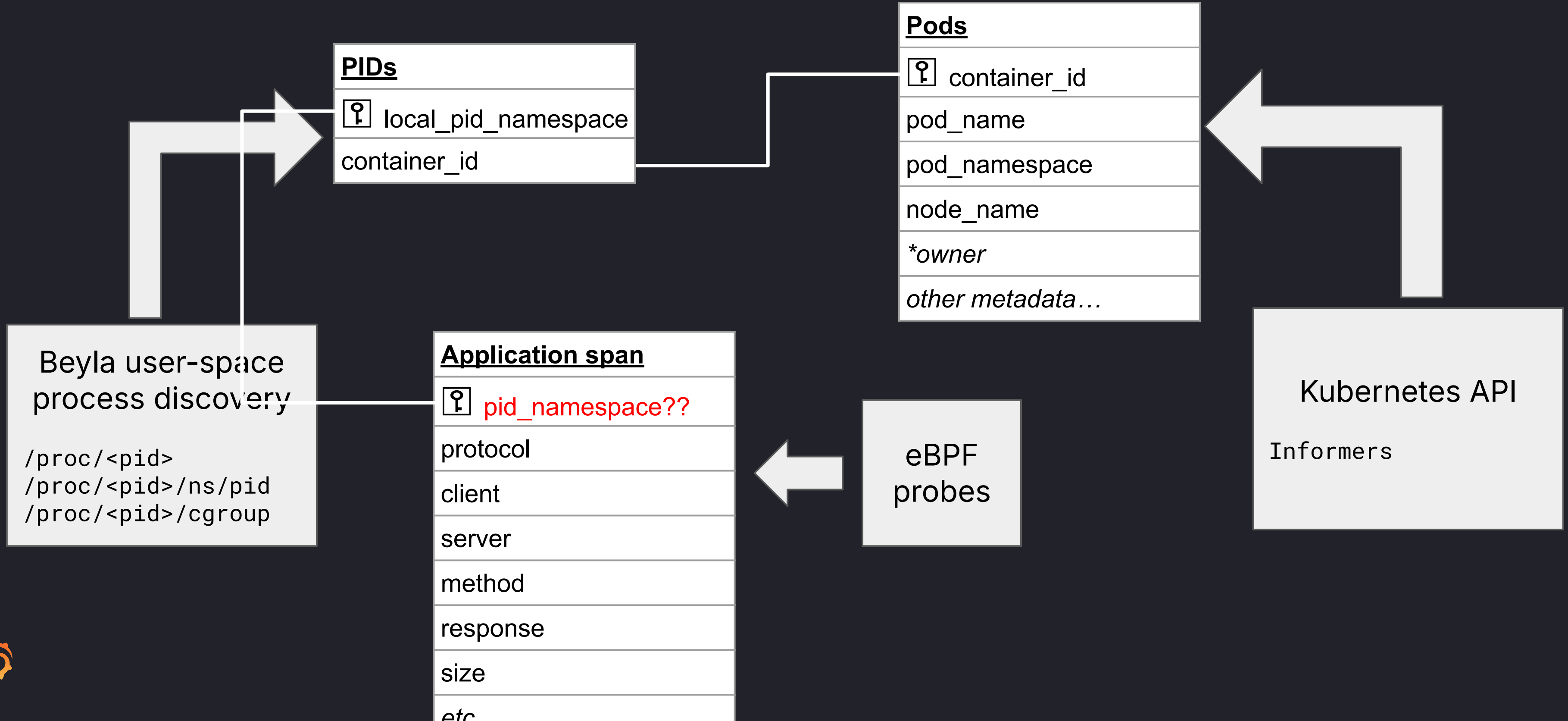
⚠ No direct mapping between host:pid and K8s metadata

Playing in god mode: PID namespaces



Same process, different PIDs depending on the POV

Matching all together



PIDs
🔑 local_pid_namespace
container_id

Pods
🔑 container_id
pod_name
pod_namespace
node_name
**owner*
other metadata…

Application span
🔑 pid_namespace??
protocol
client
server
method
response
size
etc

Beyla user-space process discovery

```
/proc/<pid>
/proc/<pid>/ns/pid
/proc/<pid>/cgroup
```

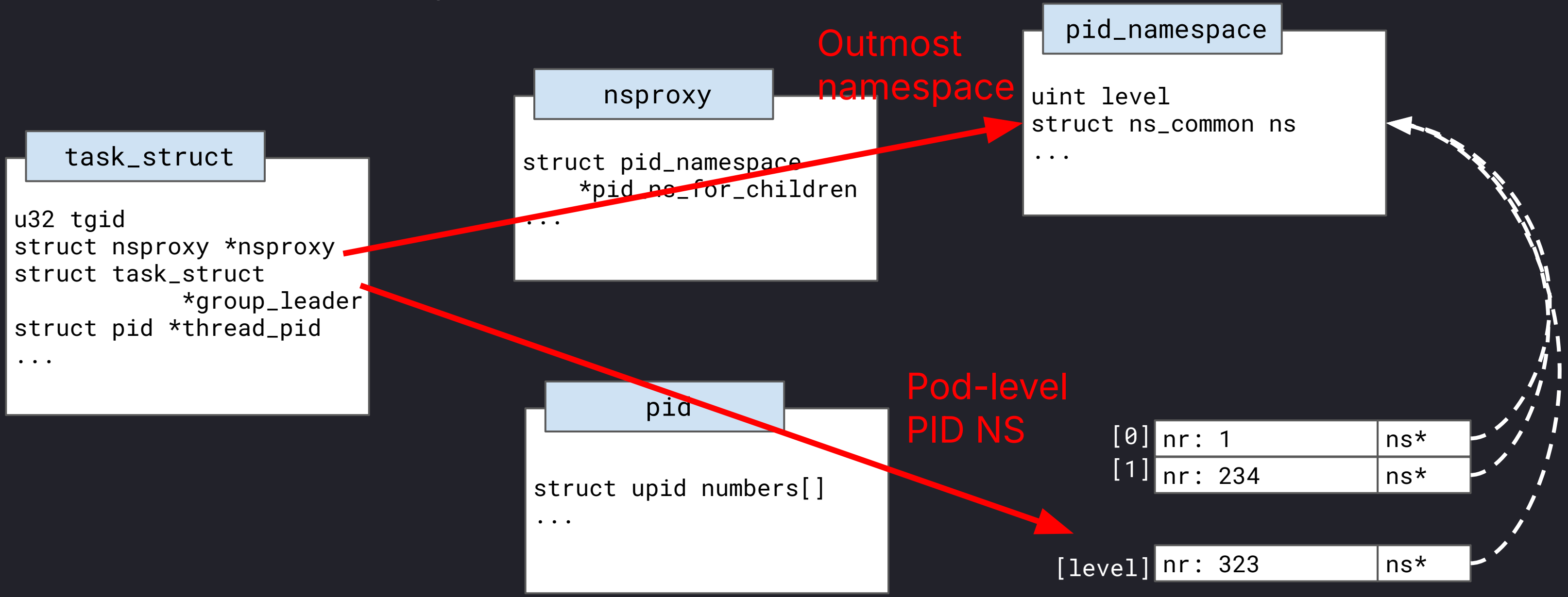
eBPF probes

Kubernetes API

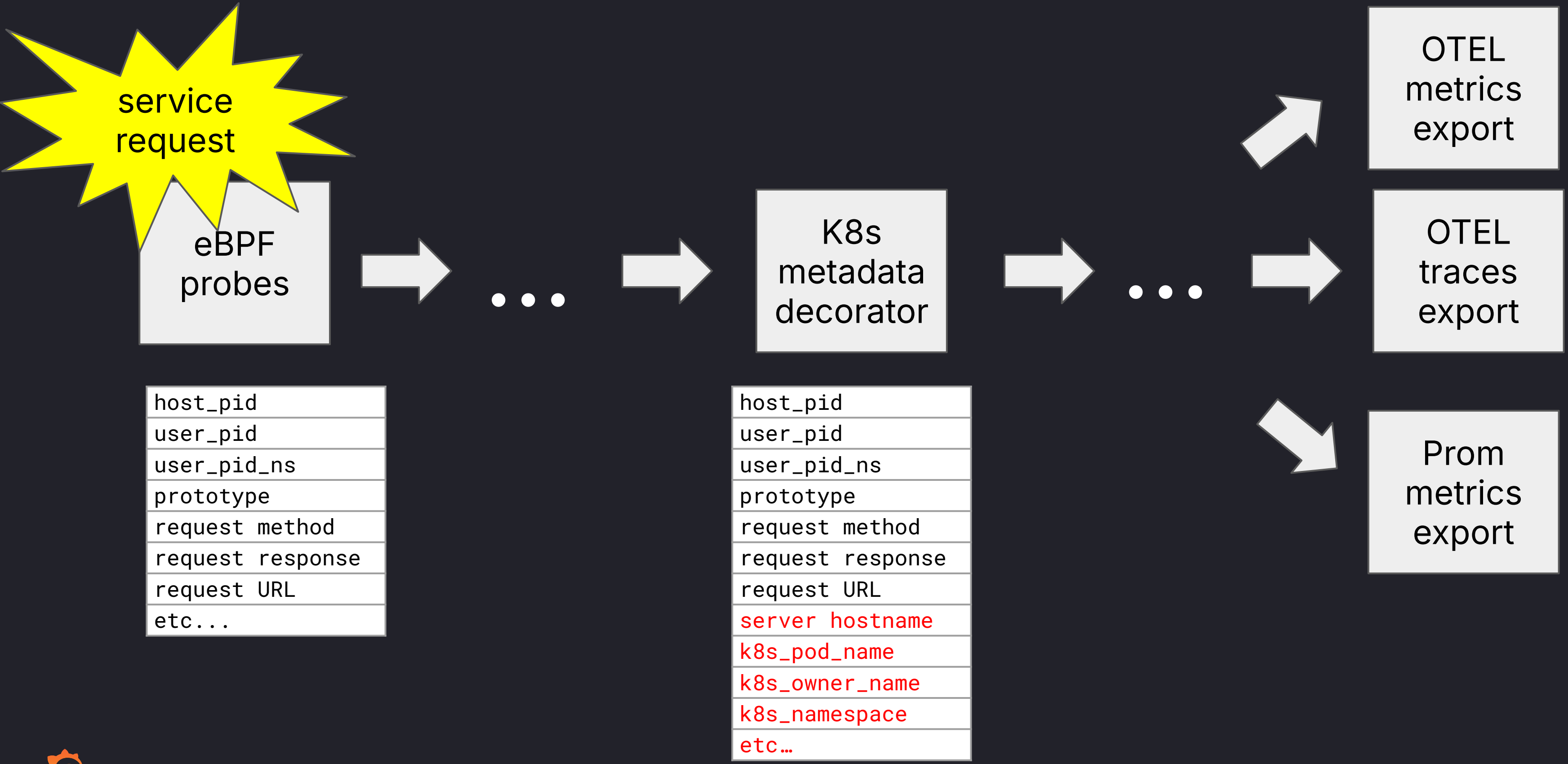
```
Informers
```

Getting the PID as seen by Beyla

- u64 bpf_get_current_pid_tgid()
  - Returns the PID as seen from the Kernel (Namespace: 1) != PID as seen from Beyla
- struct task_struct* bpf_get_current_task()



The journey of an application trace



host_pid
user_pid
user_pid_ns
prototype
request method
request response
request URL
etc...

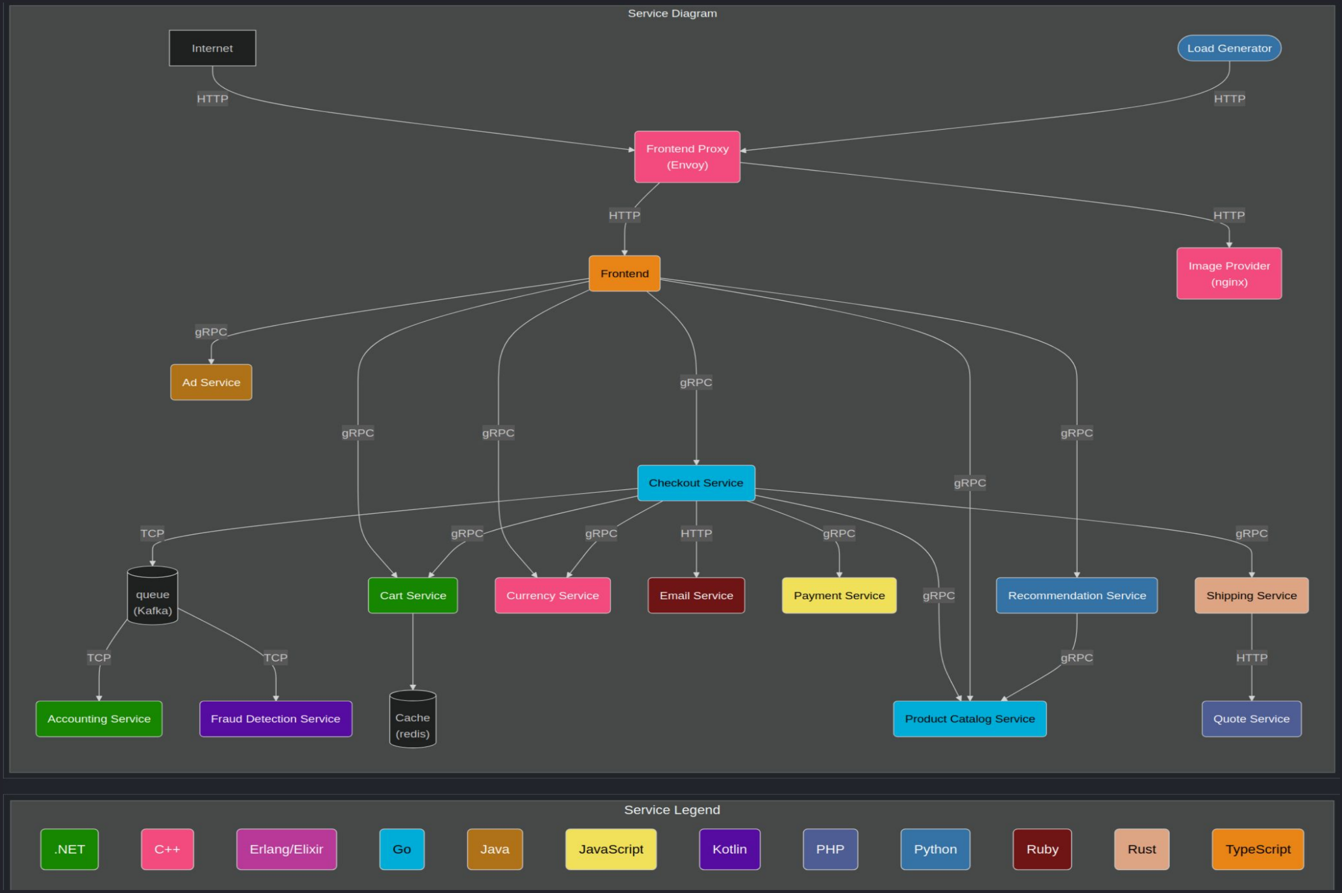
host_pid
user_pid
user_pid_ns
prototype
request method
request response
request URL
server hostname
k8s_pod_name
k8s_owner_name
k8s_namespace
etc…

Demo Time

Config (values.yml)

```
config:
  data:
    attributes:
      kubernetes:
        enable: true
    prometheus_export:
      port: 9090
      path: /metrics
    discovery:
      services:
        - k8s_namespace: default
          k8s_deployment_name: .
        - k8s_namespace: default
          k8s_daemonset_name: .
```

OpenTelemetry demo



Service Diagram

Internet

Load Generator

HTTP

HTTP

Frontend Proxy (Envoy)

HTTP

HTTP

Frontend

Image Provider (nginx)

gRPC

Ad Service

gRPC

gRPC

gRPC

gRPC

gRPC

Checkout Service

TCP

gRPC

gRPC

HTTP

gRPC

gRPC

queue (Kafka)

Cart Service

Currency Service

Email Service

Payment Service

gRPC

Recommendation Service

Shipping Service

TCP

TCP

gRPC

HTTP

Accounting Service

Fraud Detection Service

Cache (redis)

Product Catalog Service

Quote Service

Service Legend

.NET

C++

Erlang/Elixir

Go

Java

JavaScript

Kotlin

PHP

Python

Ruby

Rust

TypeScript

(near) Future

(near) Future

- Reduce privileges required to run Beyla
  - Currently depending on BPFS to mount maps
  - Working on required only few capabilities
- Improve performance of Kubernetes informers
  - Currently fetches all metadata all Pods in the node
  - Working on a centralised cache of objects metadata

Conclusions

Conclusions

- eBPF is a powerful tool
- But at same time hard to master
- Challenges to match Kubernetes abstractions
- Future work

Questions