Grafana Labs

Zero-code application metrics with eBPF and Prometheus

An introduction to Grafana Beyla

Mario Macías Lloret
Nikola Grcevski

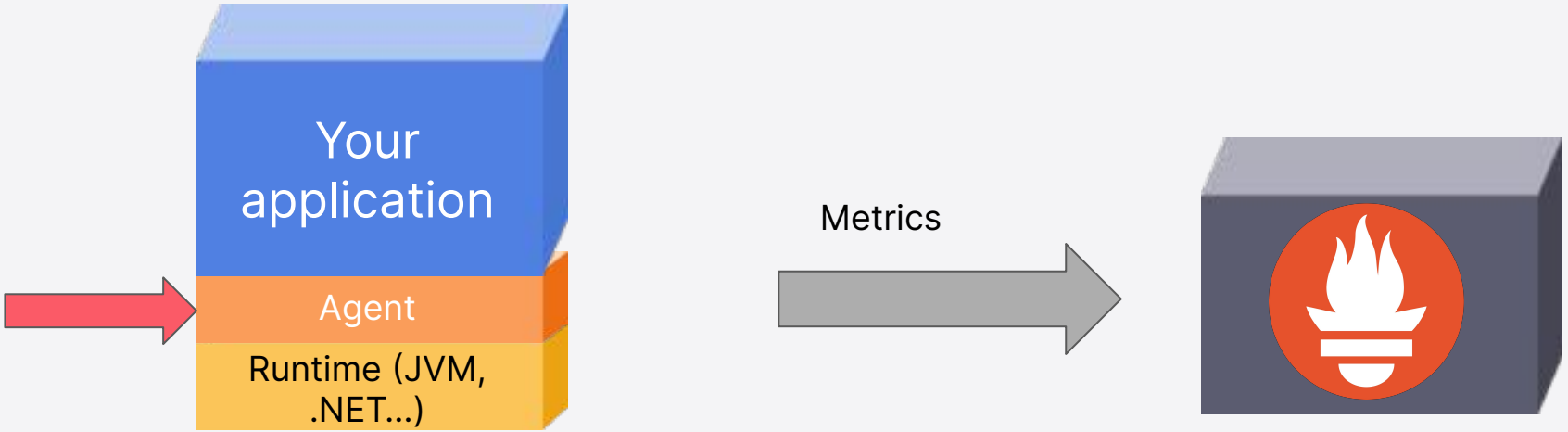
Promcon 2023
Berlin, Germany

Grafana Labs

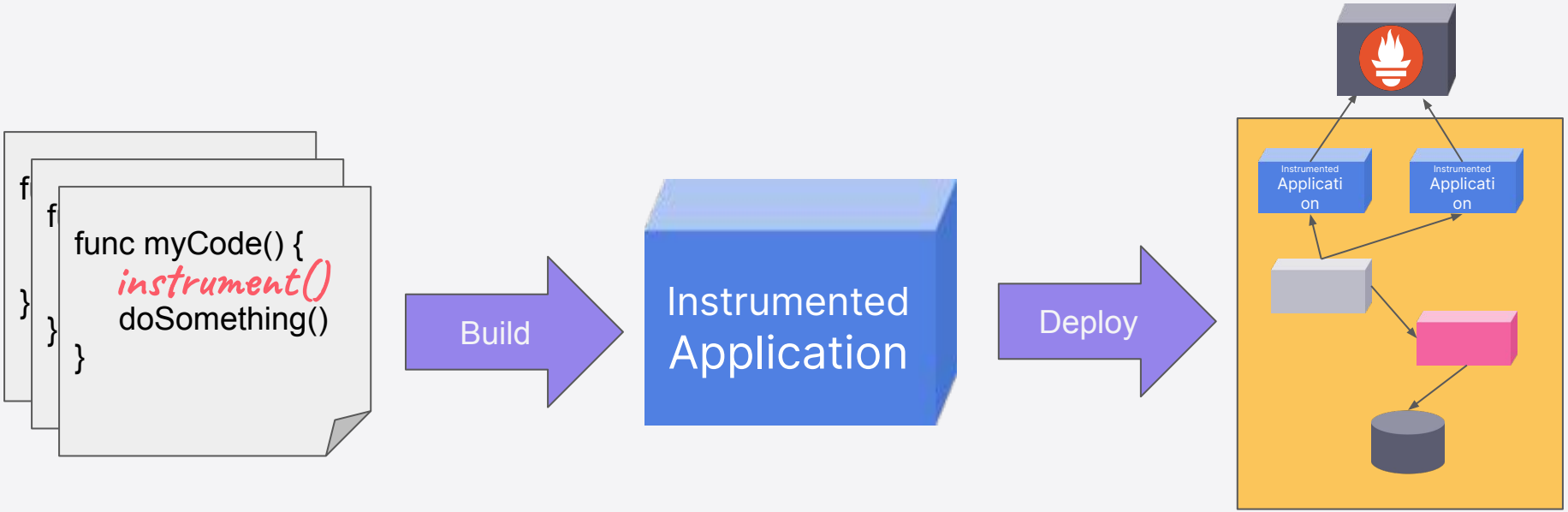
Introduction

Instrumenting your uninstrumented application

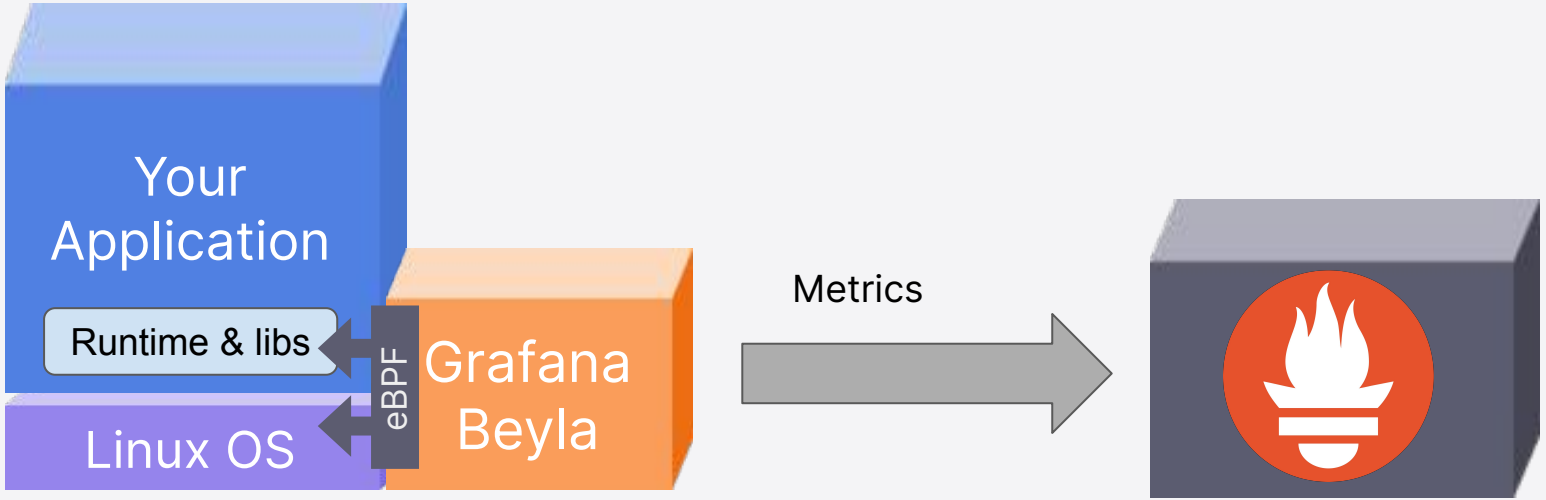
Agent-based instrumentation



Manual instrumentation



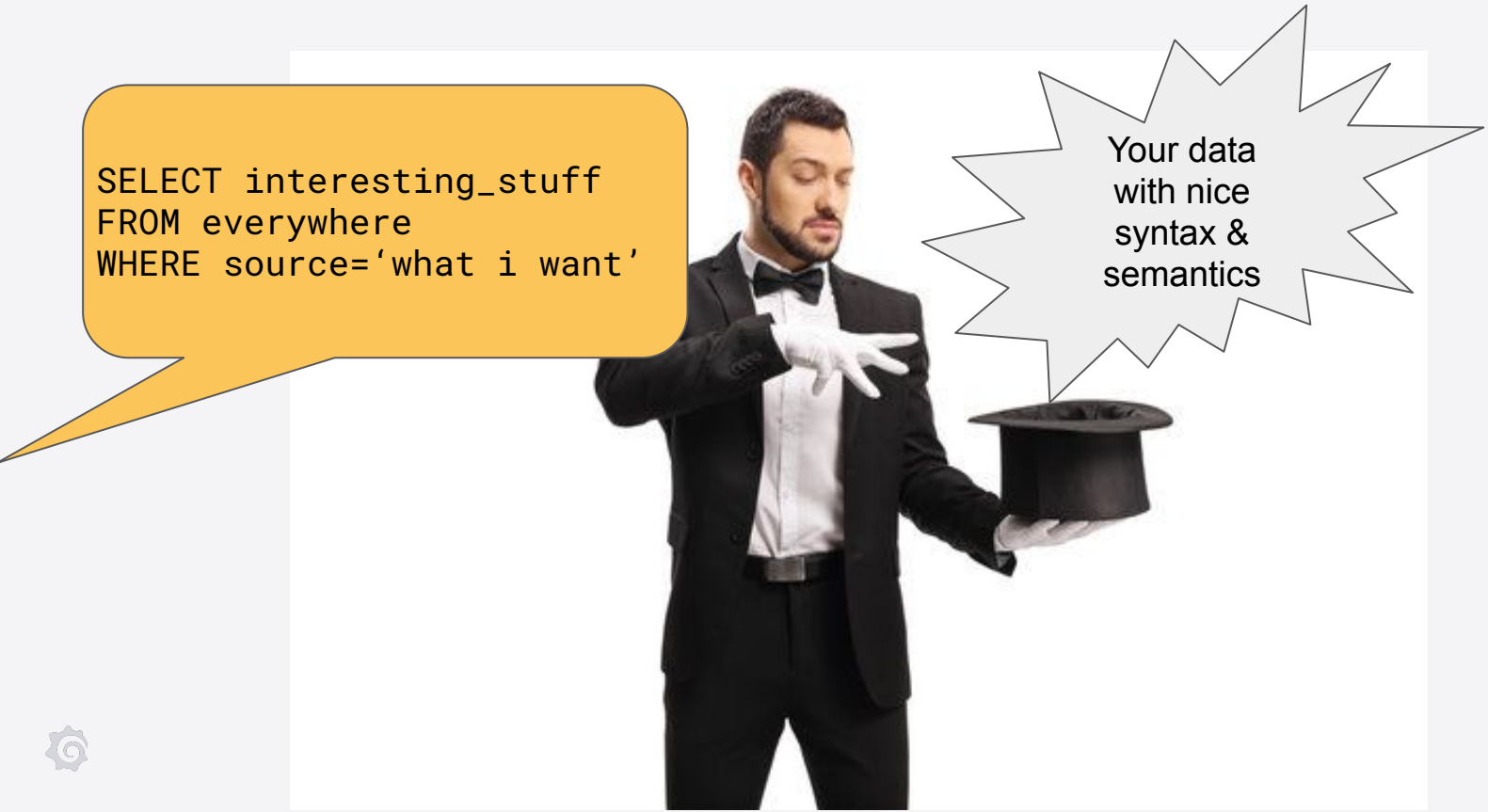
Beyla native eBPF auto-instrumentation



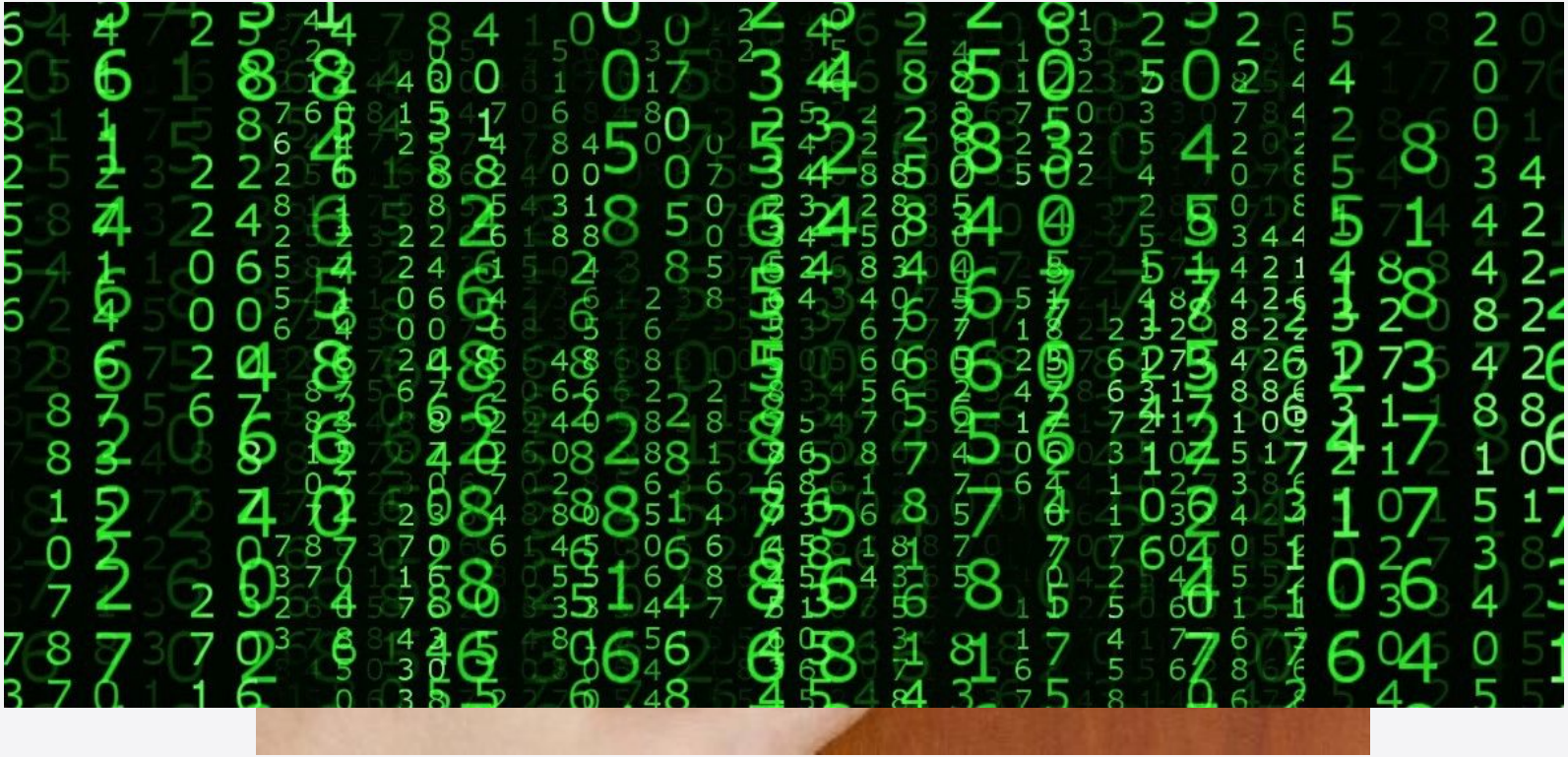
Grafana Labs

E... B... P... what?

What’s not eBPF



What eBPF looks like?

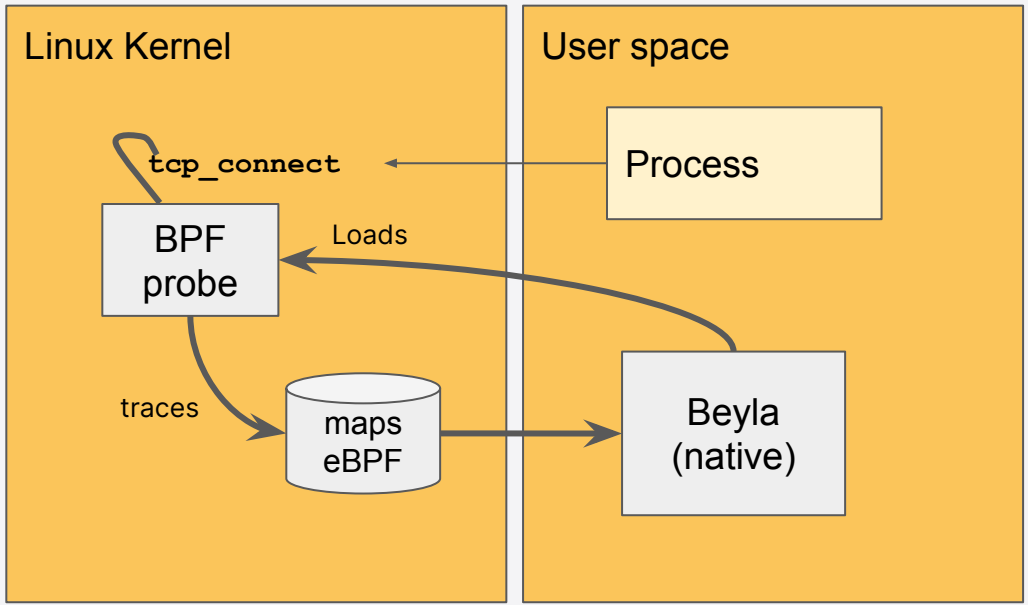


eBPF

- Extended Berkeley Packet Filter
- JIT Virtual Machine at the Linux Kernel
- Can hook your probe programs to multiple events of the Kernel, libraries and user-space programs
    - Lets you see (and even modify) the runtime memory
- You need to know how the memory is organized at a binary level
    - For each language, compiler, and architecture
        - Arguments
        - Data structures
        - Local variables
        - Return values

Example: trace each new TCP connection

```
int tcp_connect(struct sock *sk);
```



eBPF advantages

- Fast: JIT compilation
- Stable
  - Programs are pre-verified before loading
    - Prevent unallowed memory accesses
    - Prevent memory loops
- Clean
  - Stopping (or crashing) the “monitor” frees the loaded resources.

eBPF Disadvantages

- Hard to debug (kprintf)
- Your probes are often dependent of the implementation details
  - Arguments in stack vs registers
  - Architecture/language/compiler conventions
  - Big endian vs little endian
  - etc...
- Changes in the inspected APIs can break your code
- User-space monitor program requires at least CAP_SYS_ADMIN privileges

Grafana Labs

Grafana Beyla

Grafana Beyla

- Automatic instrumentation of your services and clients
  - Go: HTTP, HTTPS & GRPC
    - Instruments Go executables, inserting User Probes at concrete symbols (e.g. ServeHttp functions)
  - Other languages: HTTP & HTTPS
    - Instruments the Kernel and libraries with Kernel probes
  - More protocols to come...

Grafana Beyla

- Export data as
  - RED metrics (Request, Errors, Duration)
    - Prometheus
    - OpenTelemetry
  - OpenTelemetry traces

Currently supported Prometheus Metrics

```
http_server_duration_seconds_{count, sum, bucket}
http_client_duration_seconds_{count, sum, bucket}

http_server_request_size_bytes_{count, sum, bucket}
http_client_request_size_bytes_{count, sum, bucket}

rpc_server_duration_seconds_{count, sum, bucket}
rpc_client_duration_seconds_{count, sum, bucket}
```

(Also internal performance counters as prometheus metrics)

Grafana Labs

Demo!

Grafana Labs

Thank you for your attention!

https://grafana.com/oss/
https://github.com/grafana/beyla