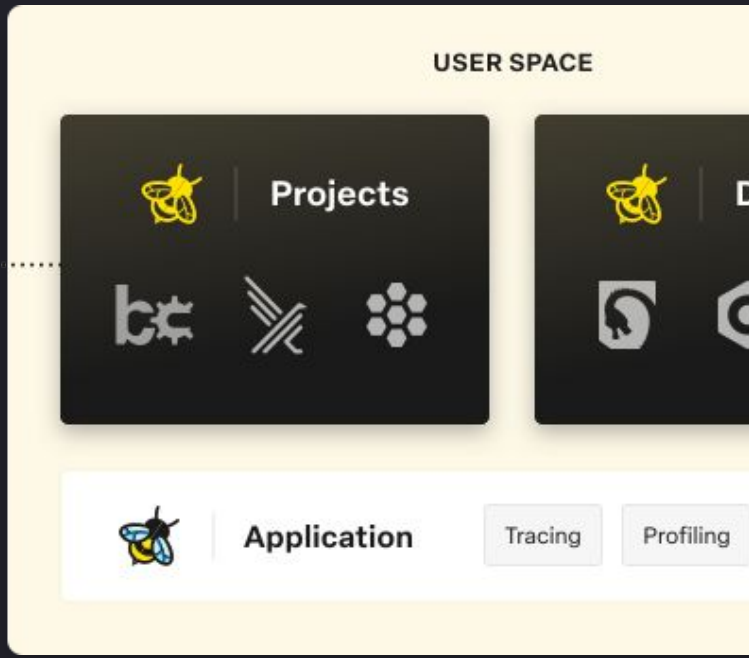
Grafana Labs

Automagic Observability with eBPF

Dominik Süß
DevEx Engineer @ Grafana Labs



Obstacles

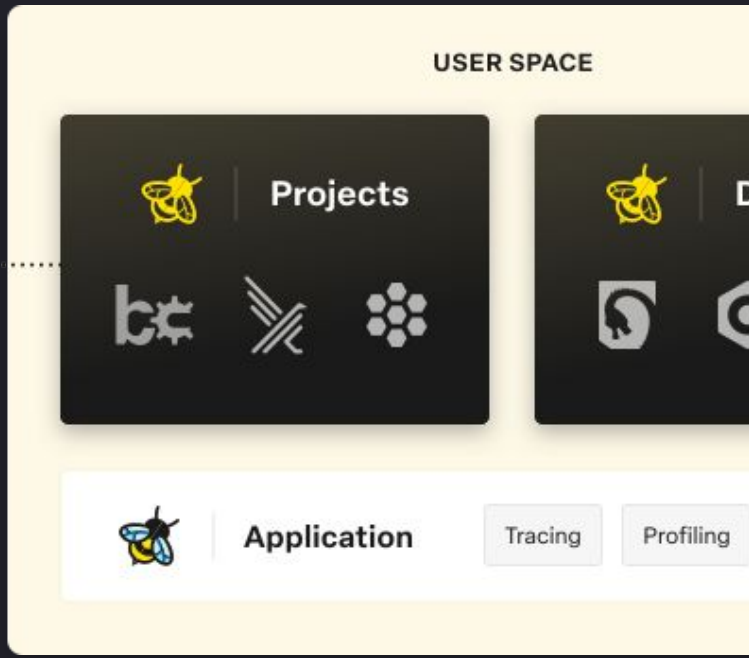

NJ 490
in 2 min
Next stop
415
10:35 arr
Hamburg Hbf
Platform 11
08:47 arr
10:42 arr
Hamburg Dammtor
10:51 arr
Hamburg-Altona


PHILIPS
Weiß zu begeistern.
couchette

Grafana Labs

Automagic Observability with eBPF

Dominik Süß
DevEx Engineer @ Grafana Labs



The issue


no

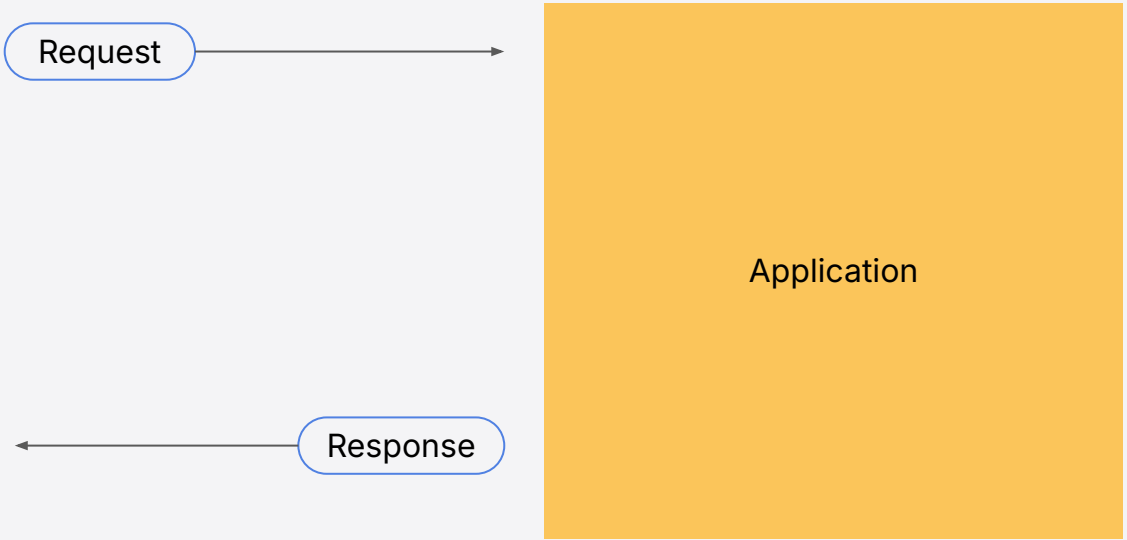
The issue

Proxy

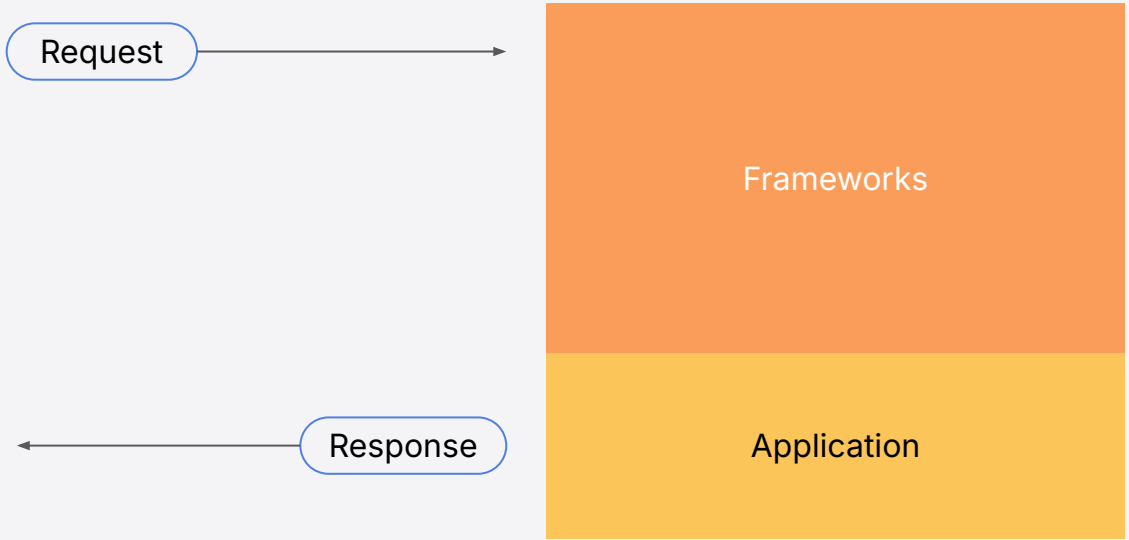
Patch

Pray

Layers upon Layers

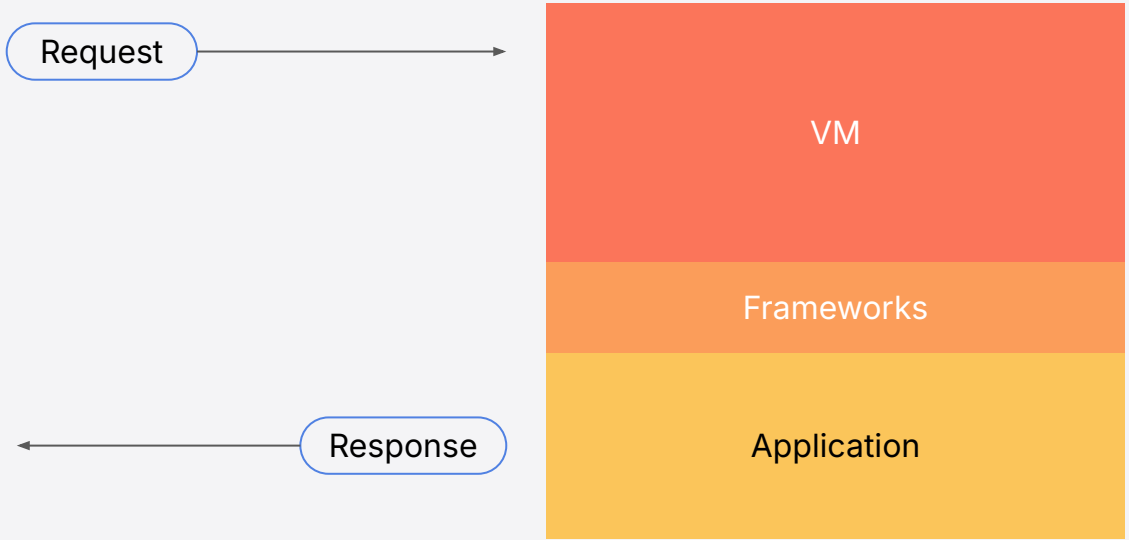


Layers upon Layers



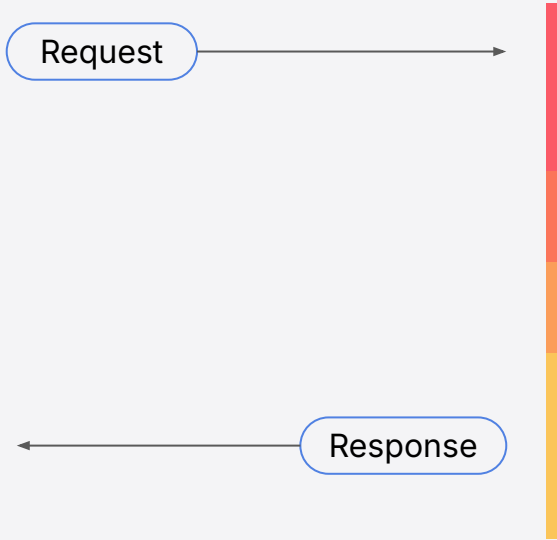
- FastAPI
- Flask
- Django
- Bottle
- Tornado
- Pyramid
- Sanic
- Falcon
- CherryPy
- Hug

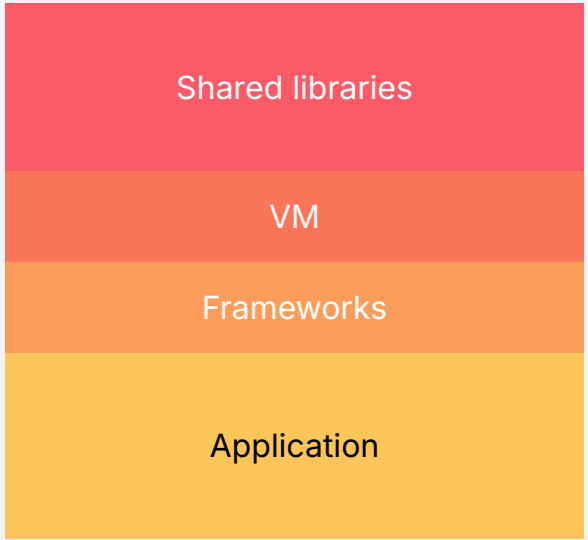
Layers upon Layers



- Implementation differences
- Not every language

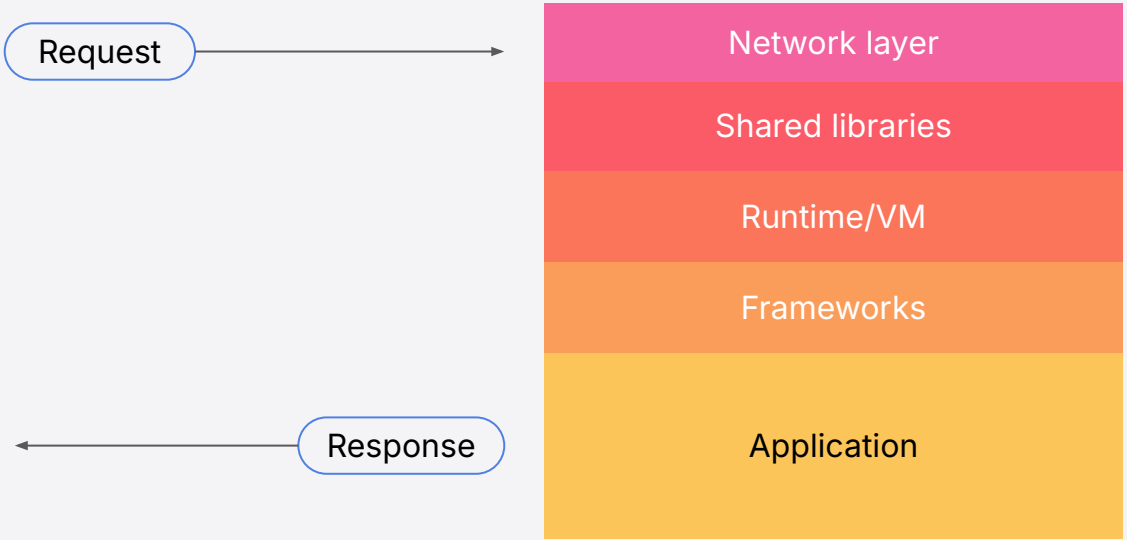
Layers upon Layers





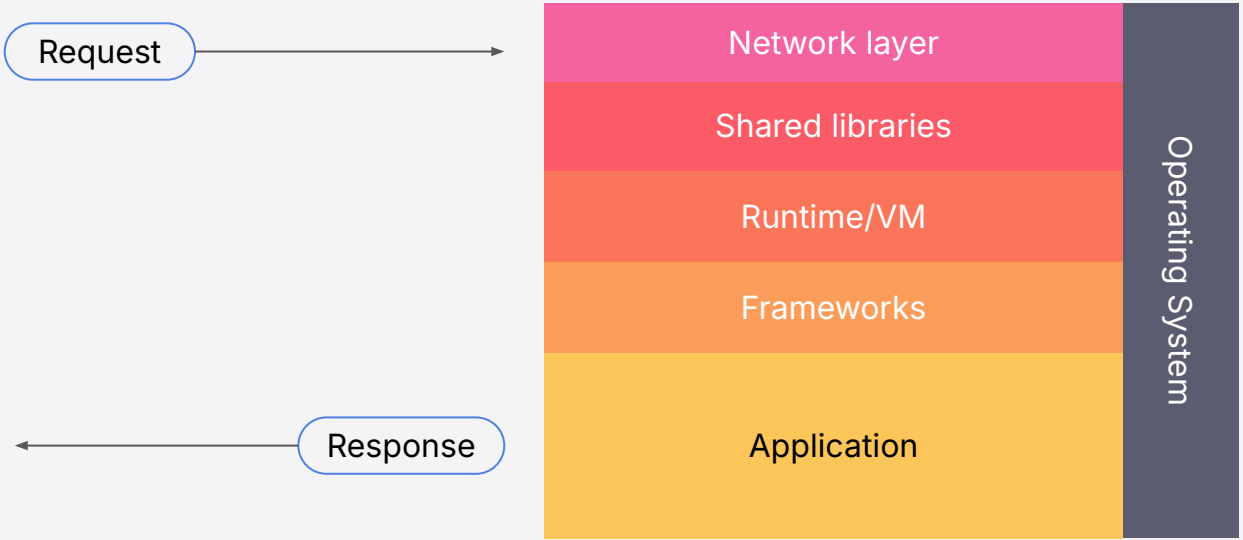
- Very specific
- No correlation

Layers upon Layers



- Not detailed enough

Layers upon Layers




eBPF

Quiz Time

A. Extended Berkley Packet Filters
B. External Binary Plugin Framework
C. Emerging Bin Packing Foundations
D. None of the above

Quiz Time

A. Extended Berkley Packet Filters

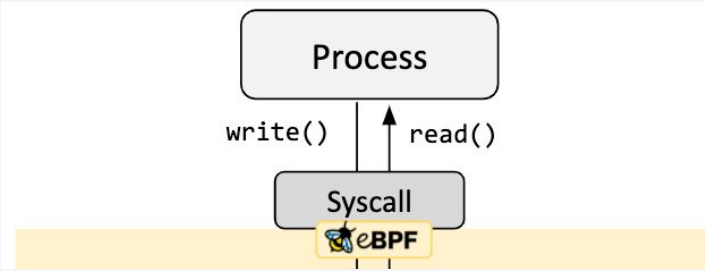
B. External Binary Plugin Framework

C. Emerging Bin Packing Foundations

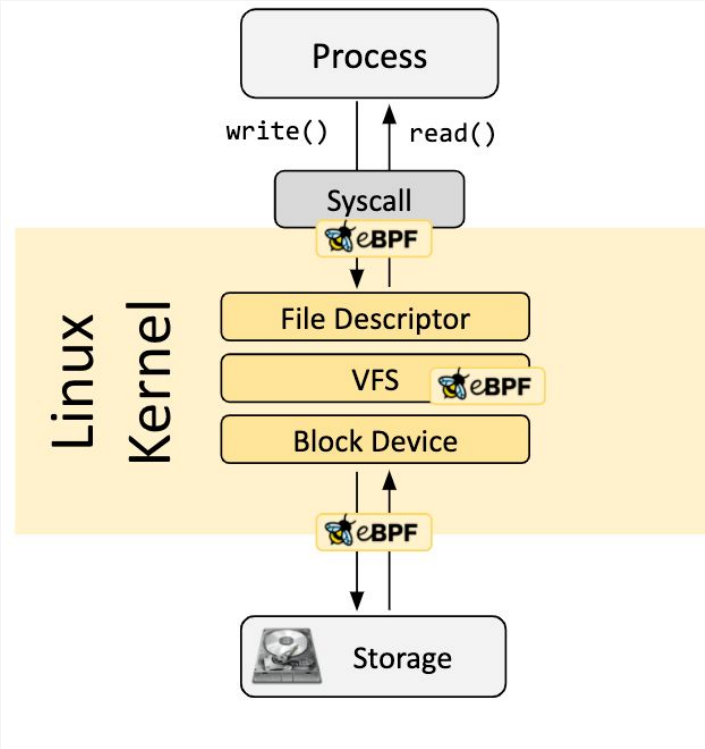
D. None of the above


eBPF

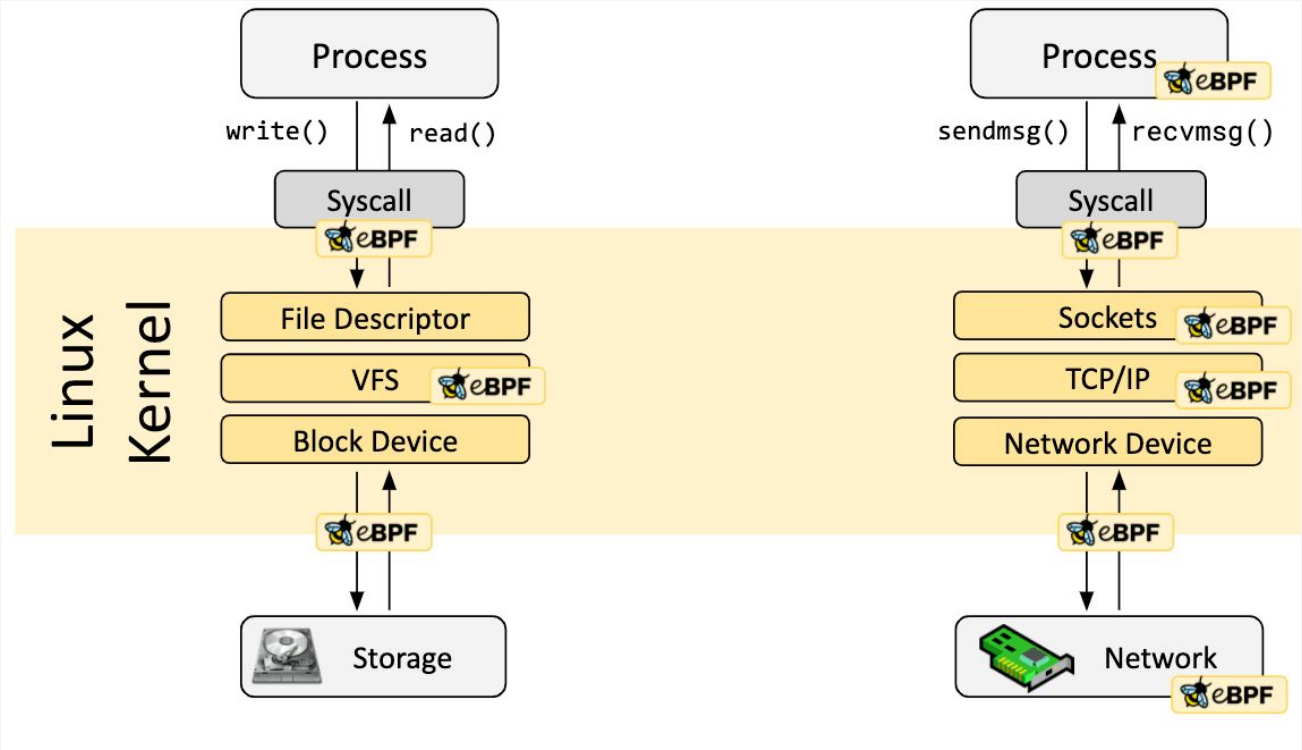
Where to hook?

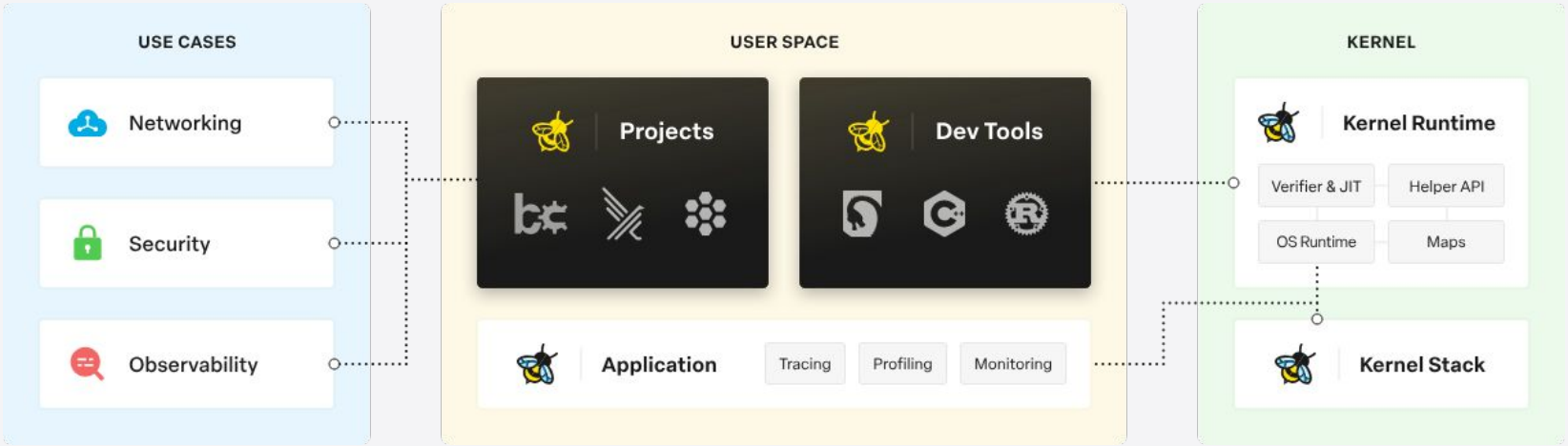


Where to hook?



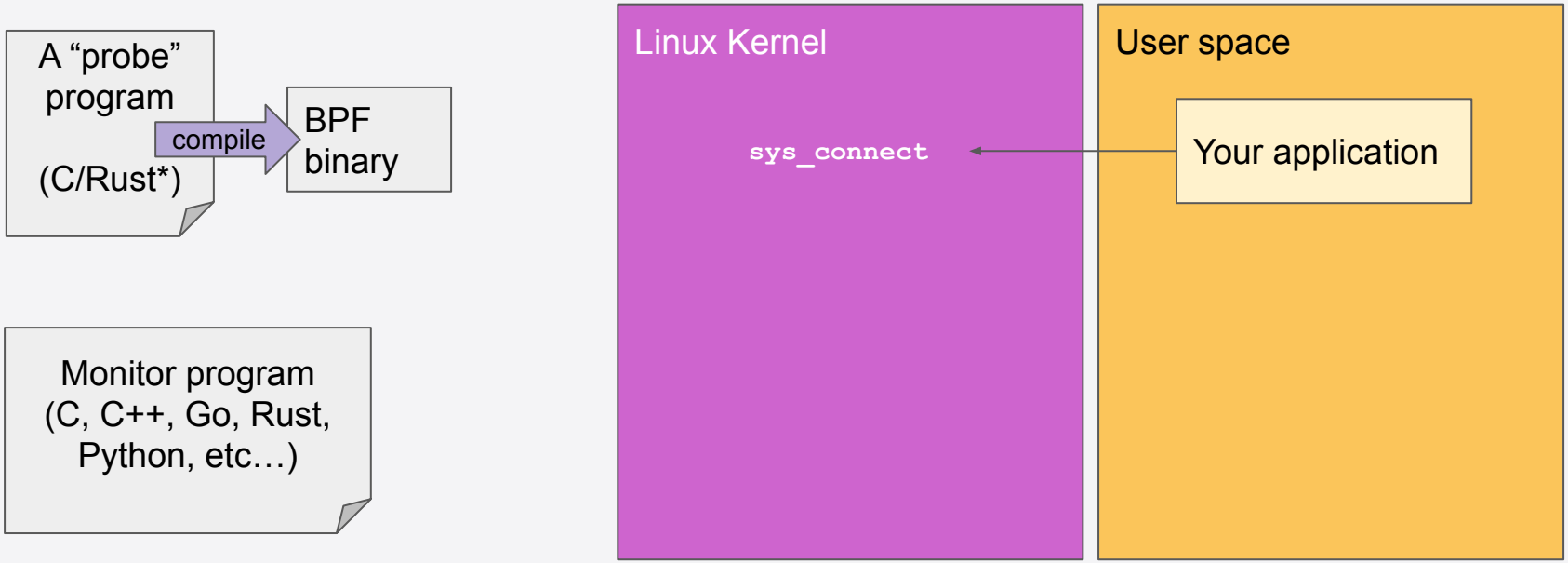
Where to hook?




USE CASES
Networking
Security
Observability
USER SPACE
Projects
Dev Tools
Application
Tracing
Profiling
Monitoring
KERNEL
Kernel Runtime
Verifier & JIT
Helper API
OS Runtime
Maps
Kernel Stack

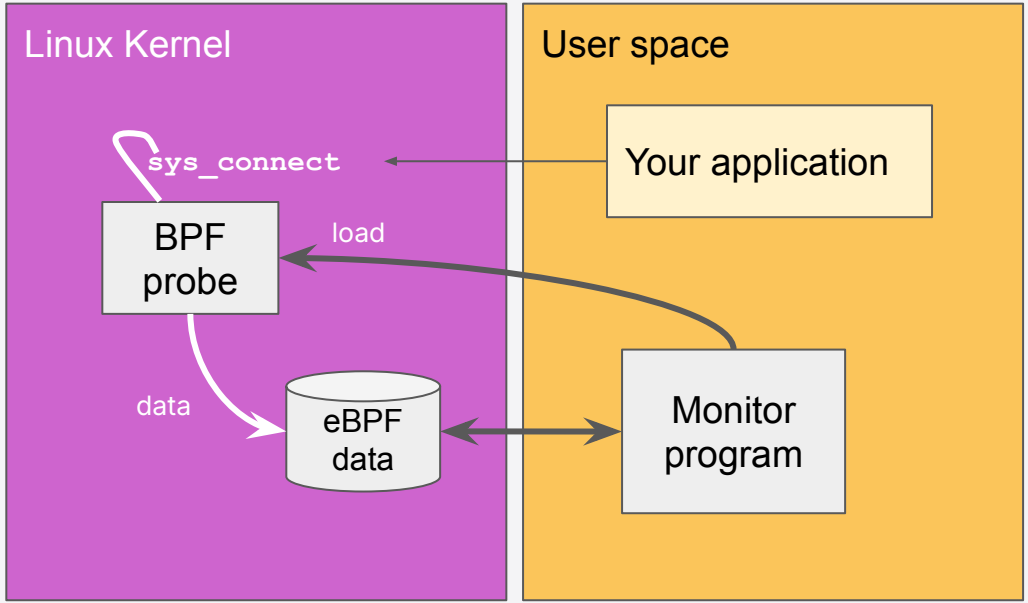
Example: track a new client TCP connection

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



Example: track a new client TCP connection

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



ebpf.c

```
/* HTTP Server */

// This instrumentation attaches uprobe to the following function:
// func (mux *ServeMux) ServeHTTP(w ResponseWriter, r *Request)
// or other functions sharing the same signature (e.g
http.Handler.ServeHTTP)
SEC("uprobe/ServeHTTP")
int obi_uprobe_ServeHTTP(struct pt_regs *ctx) {
    bpf_dbg_printk("=== uprobe/ServeHTTP === ");
    void *goroutine_addr = GOROUTINE_PTR(ctx);

    bpf_dbg_printk("goroutine_addr %lx", goroutine_addr);
    void *req = GO_PARAM4(ctx);
    go_addr_key_t g_key = {};
    go_addr_key_from_id(&g_key, goroutine_addr);
```


OBI

/

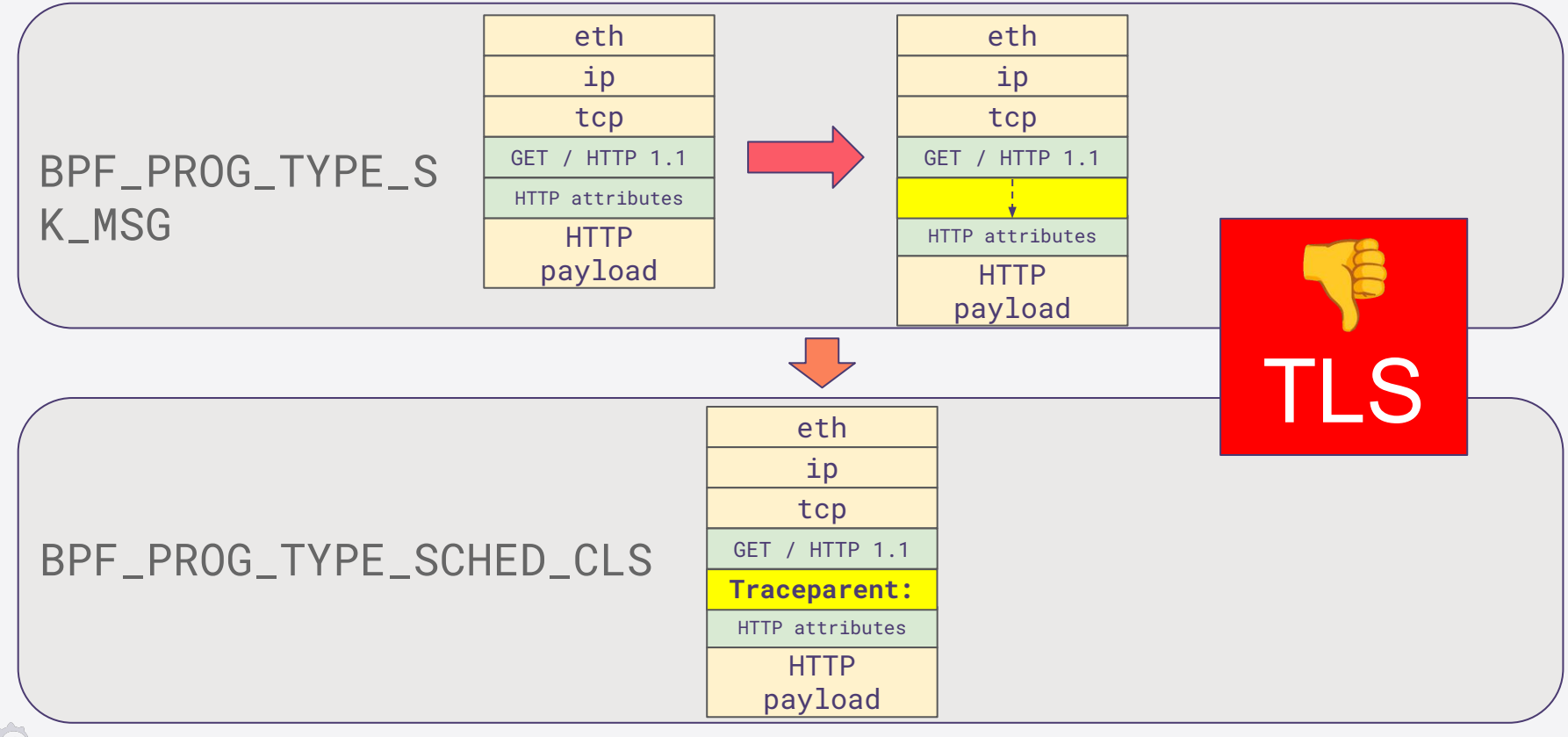
OBI

OpenTelemetry e**B**PF **I**nstrumentation

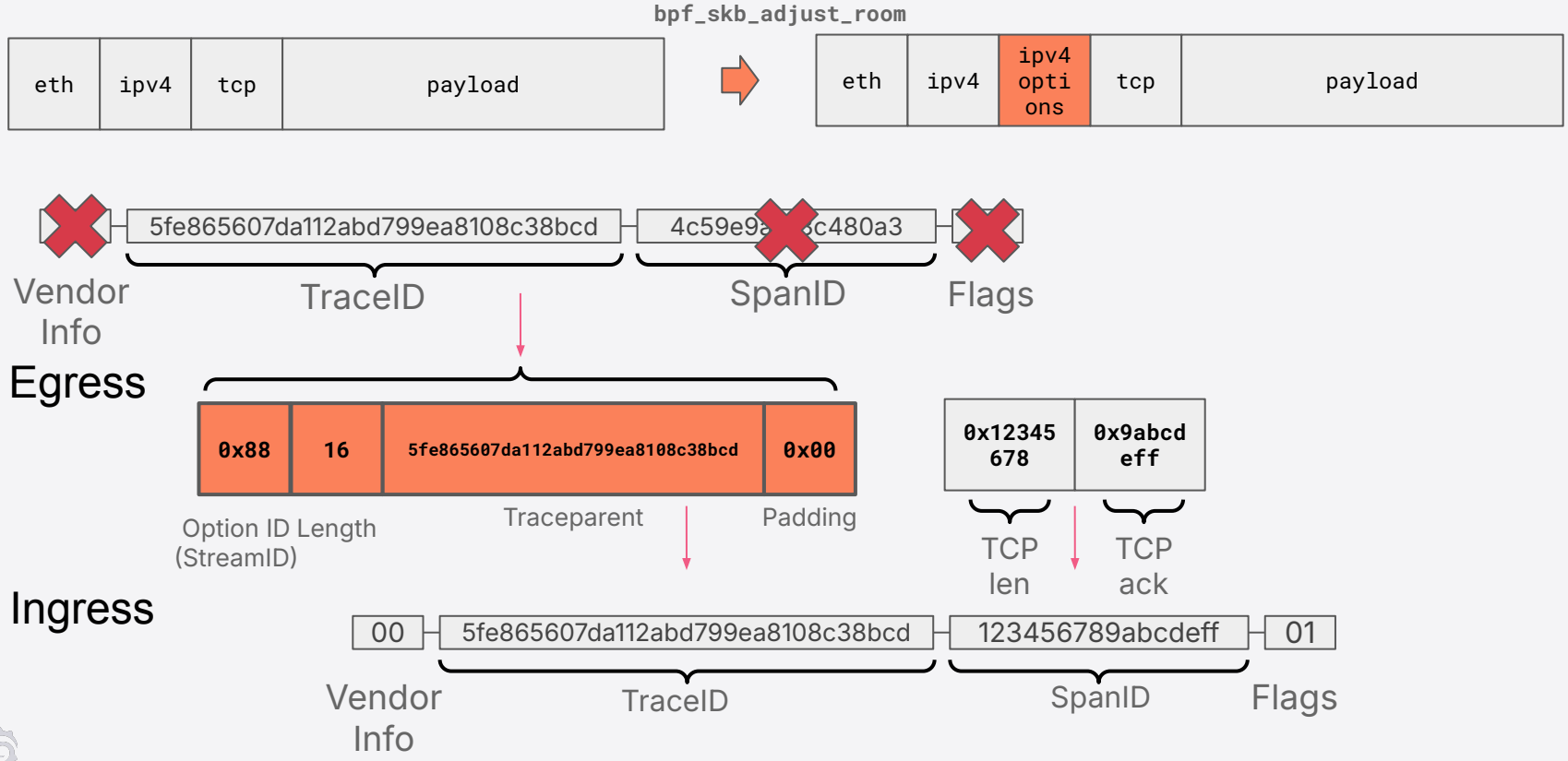
Demo

Distributed tracing deep dive

Context Propagation



Context Propagation



Considerations

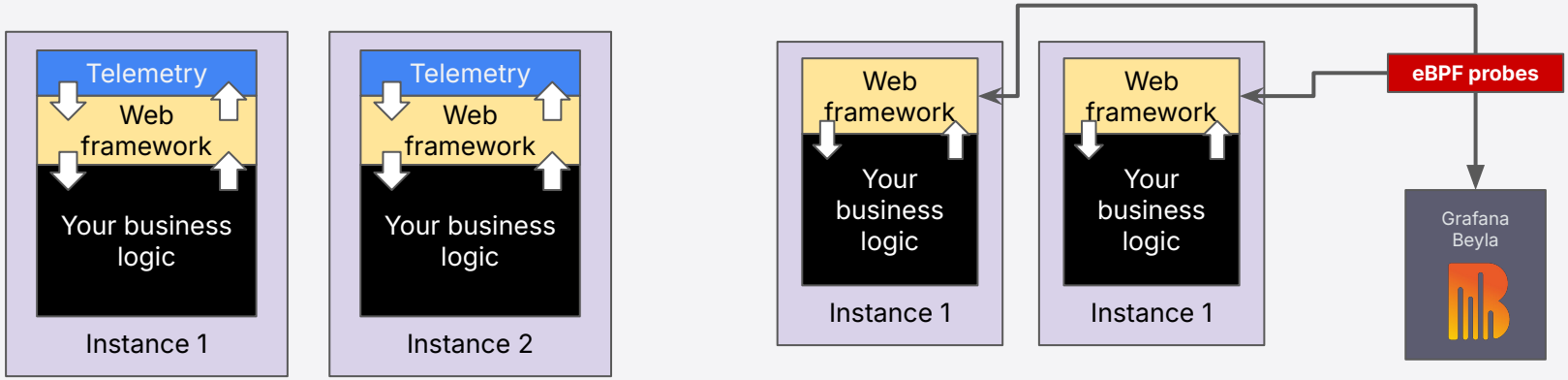
Network O11y Demo

Performance

Maturity

Flexibility

Decoupling trace/metric generation from your workload



In *single-threaded* apps, the telemetry might slow down business logic.

Performance

Maturity

Flexibility

Maturity

- OTel Ruby Metrics SDK isn’t ready yet
- Different SDKs emit different semantic conventions 🤦
- Some SDKs don’t support older versions of the runtime

Performance

Maturity

Flexibility

Flexibility

- You *cannot* add custom attributes to spans.
  - `customer_id`? nope.
  - `order_id`? Nope.
- Metrics will tell you *something* is wrong.
  - Not what.
- What comes from *traces / logs*.
  - Not being able to attach attributes decreases the value.

Summary

- If there is no / poor instrumentation, use Beyla
- For Java and .Net use auto-instrumentation
- Beyla / OBI is *NOT* the end
  - Low fidelity
  - Lack of custom attributes
  - You will very likely need manual instrumentation.

Grafana Labs

Thanks for participating!

Have more questions?

Join us at community.grafana.com

or Grafana public slack #beyla

Get involved:

#beyla

grafana/beyla

community.grafana.com