qaware.de

QA|WARE
SOFTWARE ENGINEERING

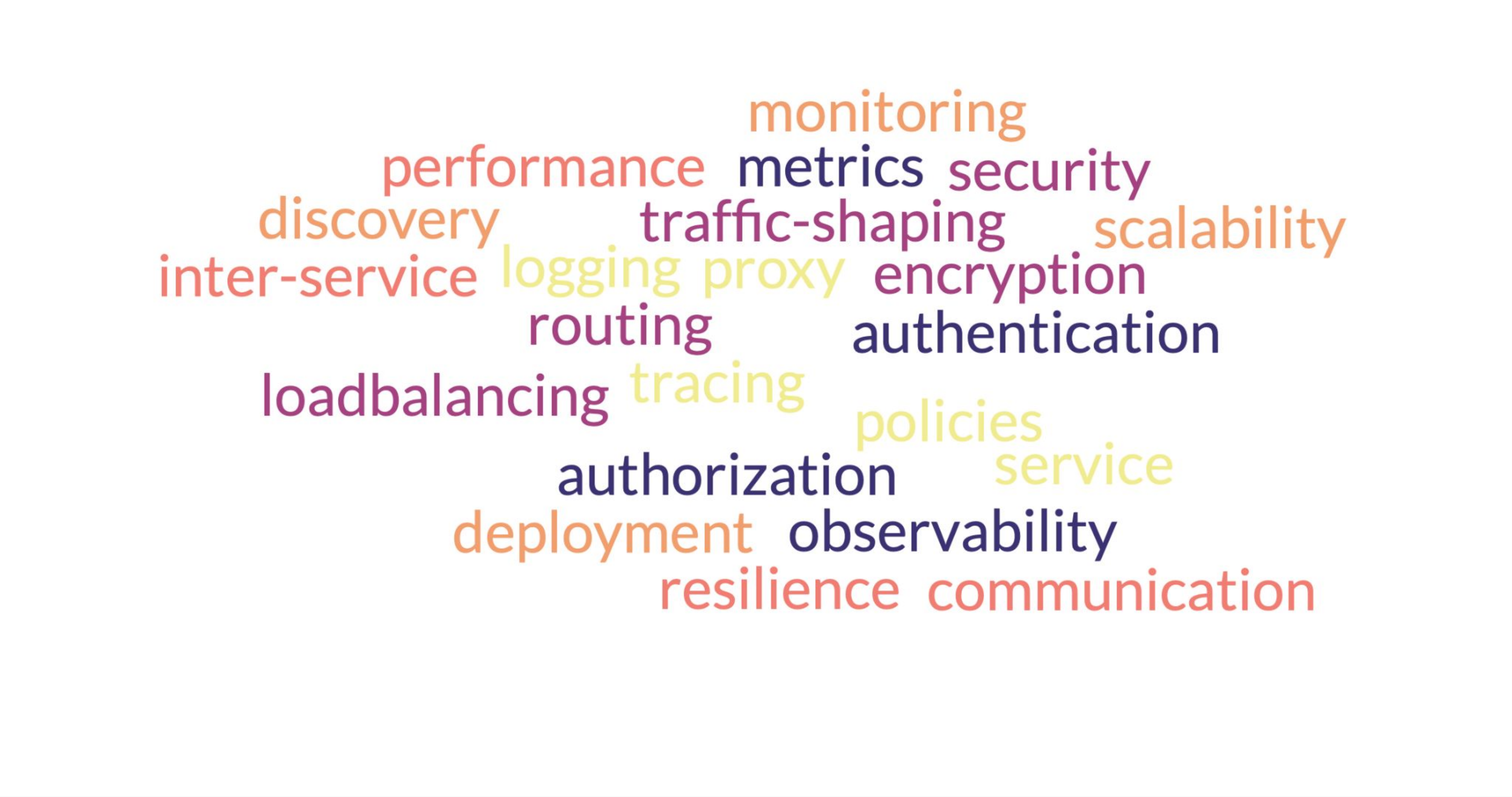
Service Meshes in modern Microservice Architectures

Lukas Buchner
lukas.buchner@qaware.de

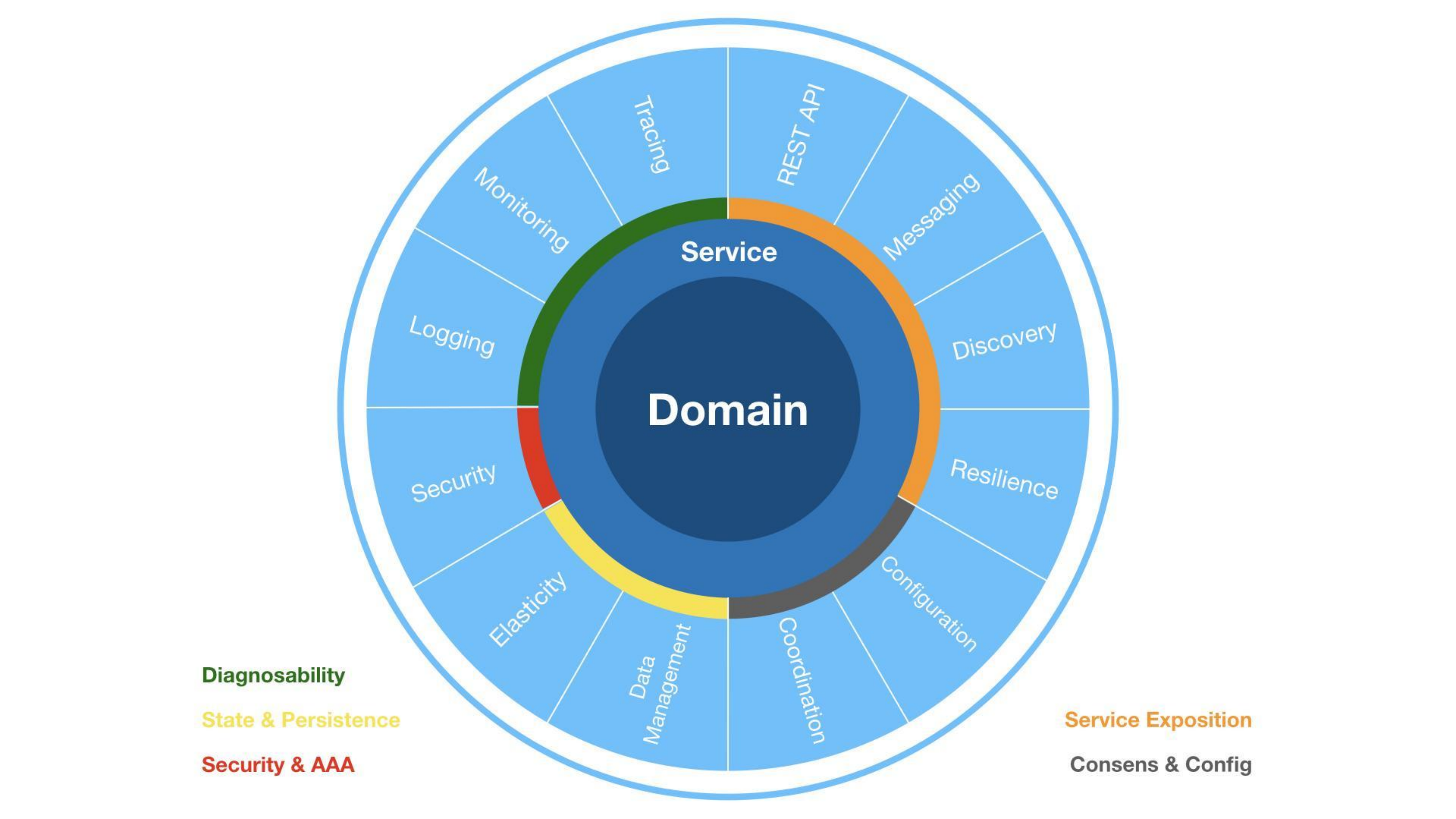
Recap

You’ve built your app…

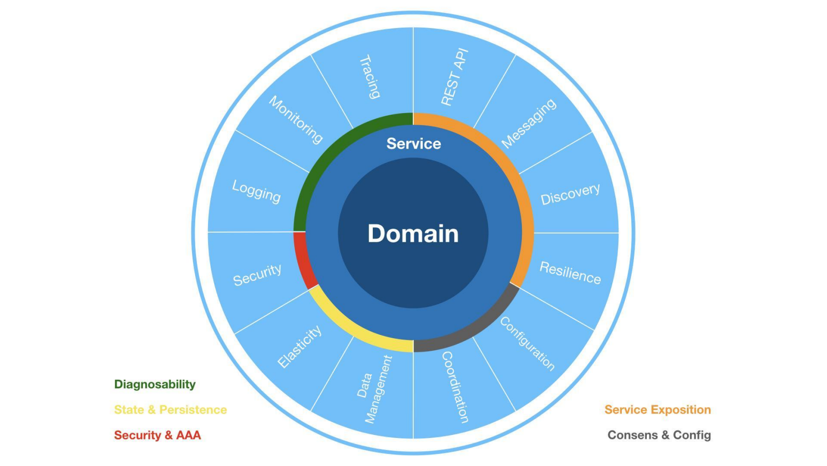
But did you consider?

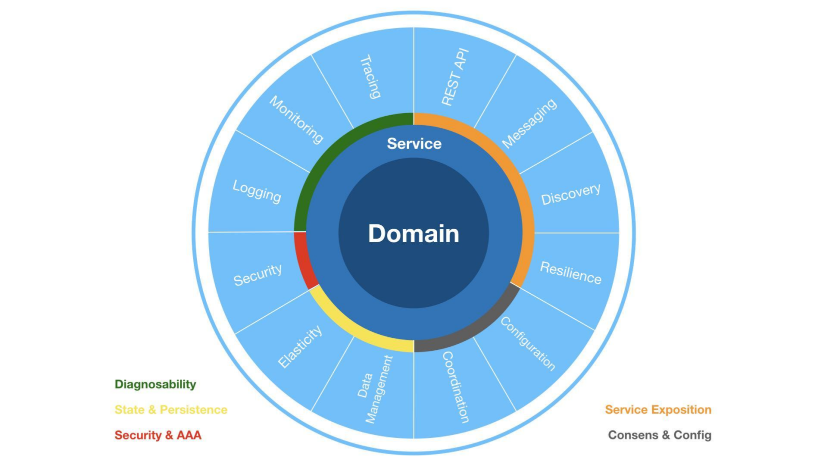


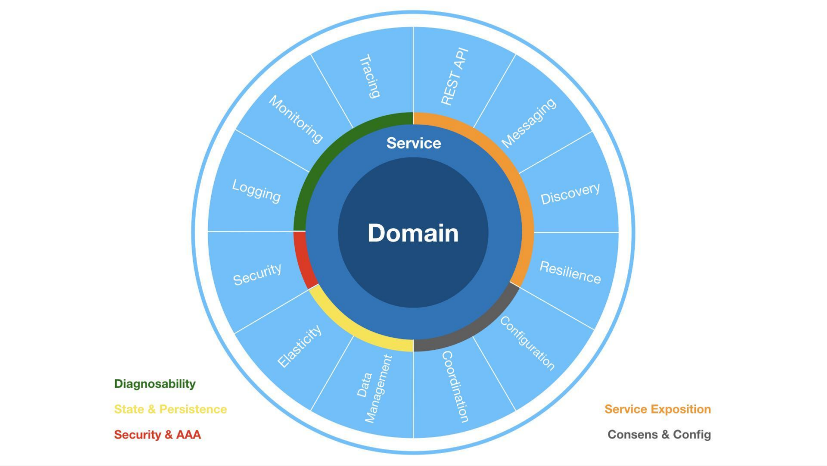
Technical aspects of microservices

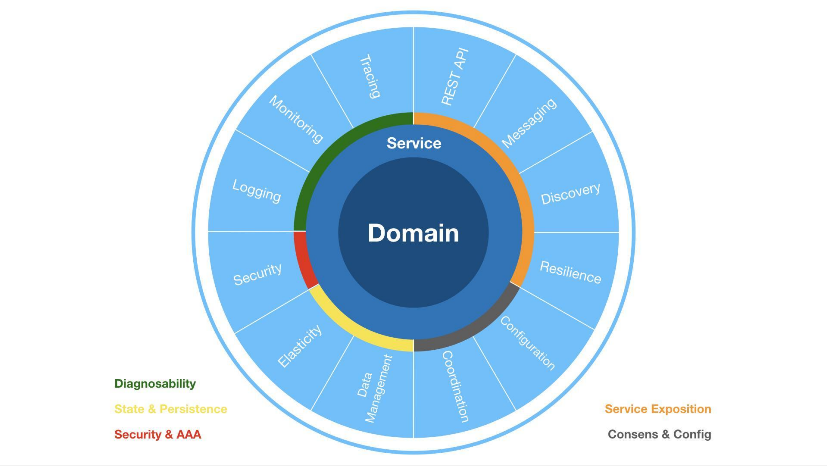


The technical aspects grow bigger with the number of services









Service Meshes

What is a Service Mesh?

In software architecture, a ***service mesh*** *is a dedicated infrastructure layer for facilitating service-to-service communications between services or microservices using a proxy.*

A dedicated communication layer can provide numerous benefits, such as providing observability into communications, providing secure connections or automating retries and backoff for failed requests.

A Service Mesh offers…

Traffic Management

- Load Balancing
- Service Discovery
- Routing
- Retries
- Timeouts
- Circuit Breaker

Security

- Encryption
- Authentication and Authorization
- Rate Limiting

Observability

- Metrics
- Tracing
- (Logging)

The major Service Meshes for Kubernetes

- Istio
- Linkerd
- Cilium
- OpenServiceMesh
- AppMesh (AWS only)

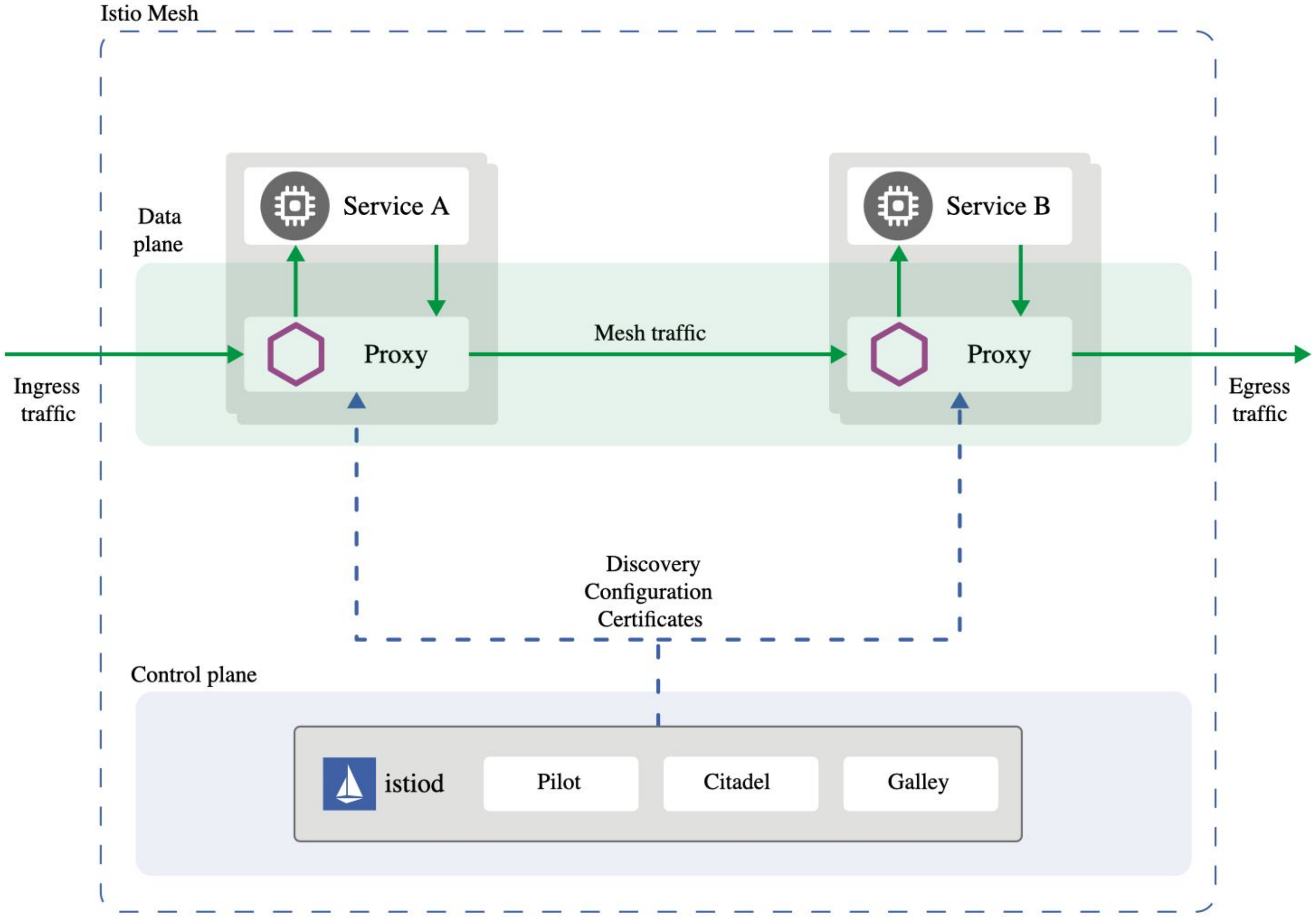
High Level architecture of Service Meshes

Control Plane:

- contains management components, acts as the “brain” of the Service Meshes.
- typical components include Service Discovery & Traffic rules, Configuration Stores & APIs, Identity & Credential Management

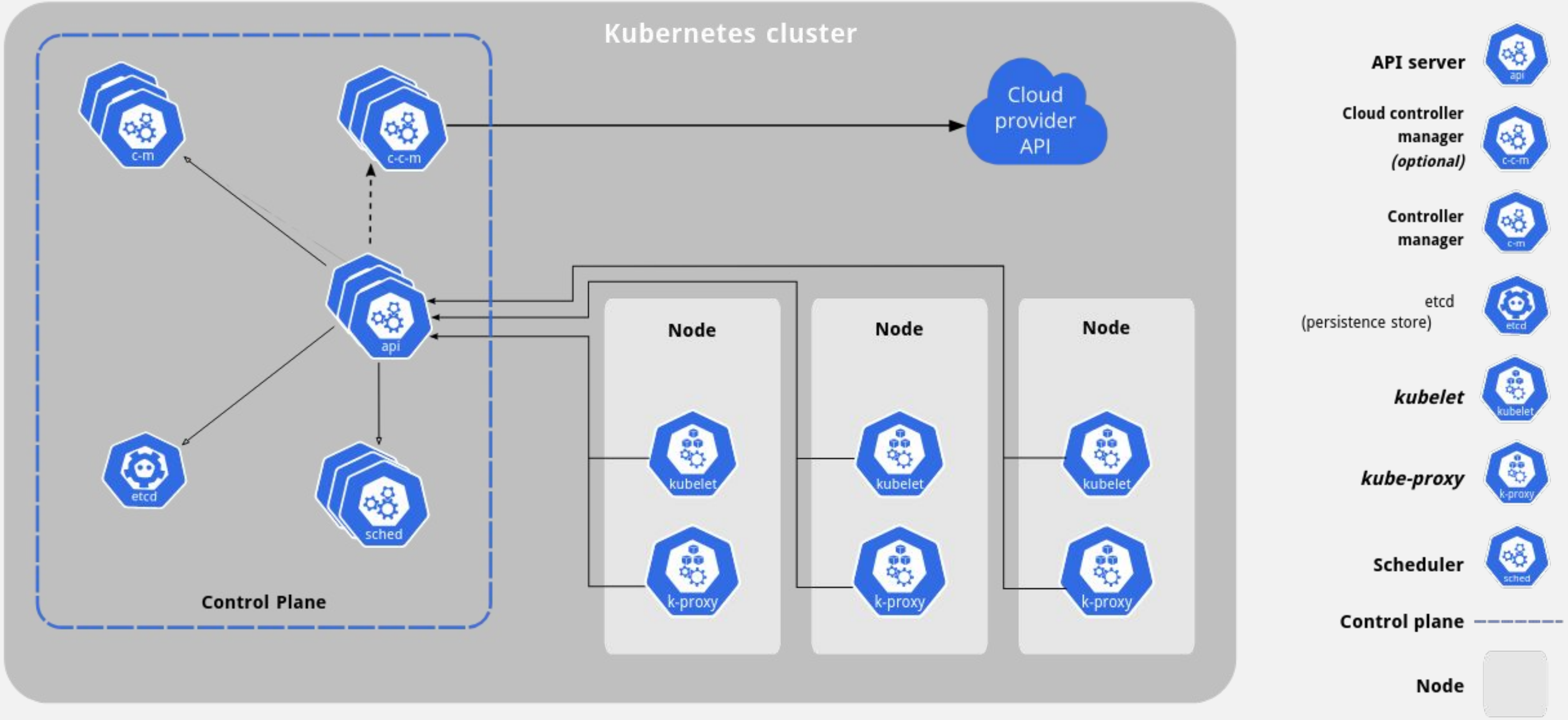
Data Plane:

- contains the actual workloads
- includes a proxy component that implements the service mesh’s features



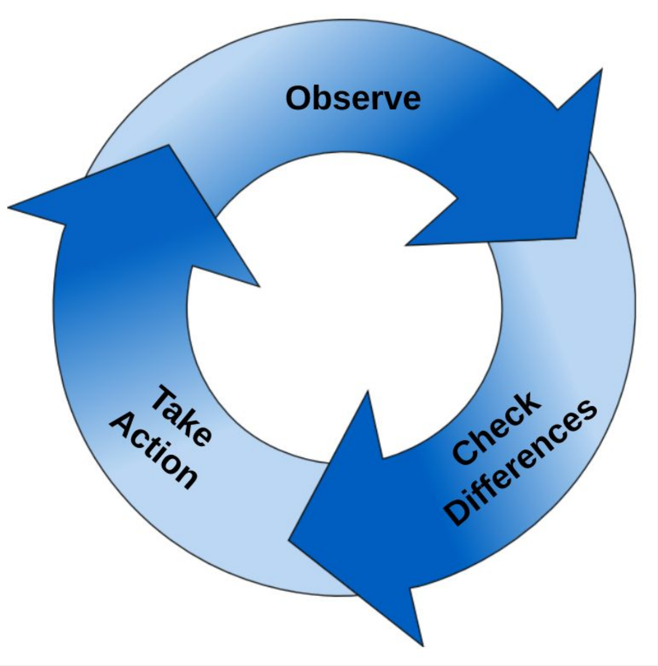
Istio Architecture

Side notes: How to extend Kubernetes?



Side notes: Kubernetes APIs are declarative and describe an object's target state

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```



Kubernetes APIs are not imperative.
For each **kind**, there is one or more responsible controllers in Kubernetes.
Controllers work event-based in a control loop:

- **observe**: what is the current state of my cluster regarding the resource managed by the controller?
- **diff**: are there deviations from the declaratively described target state?
- **take action**: bring the managed resources to the target state.

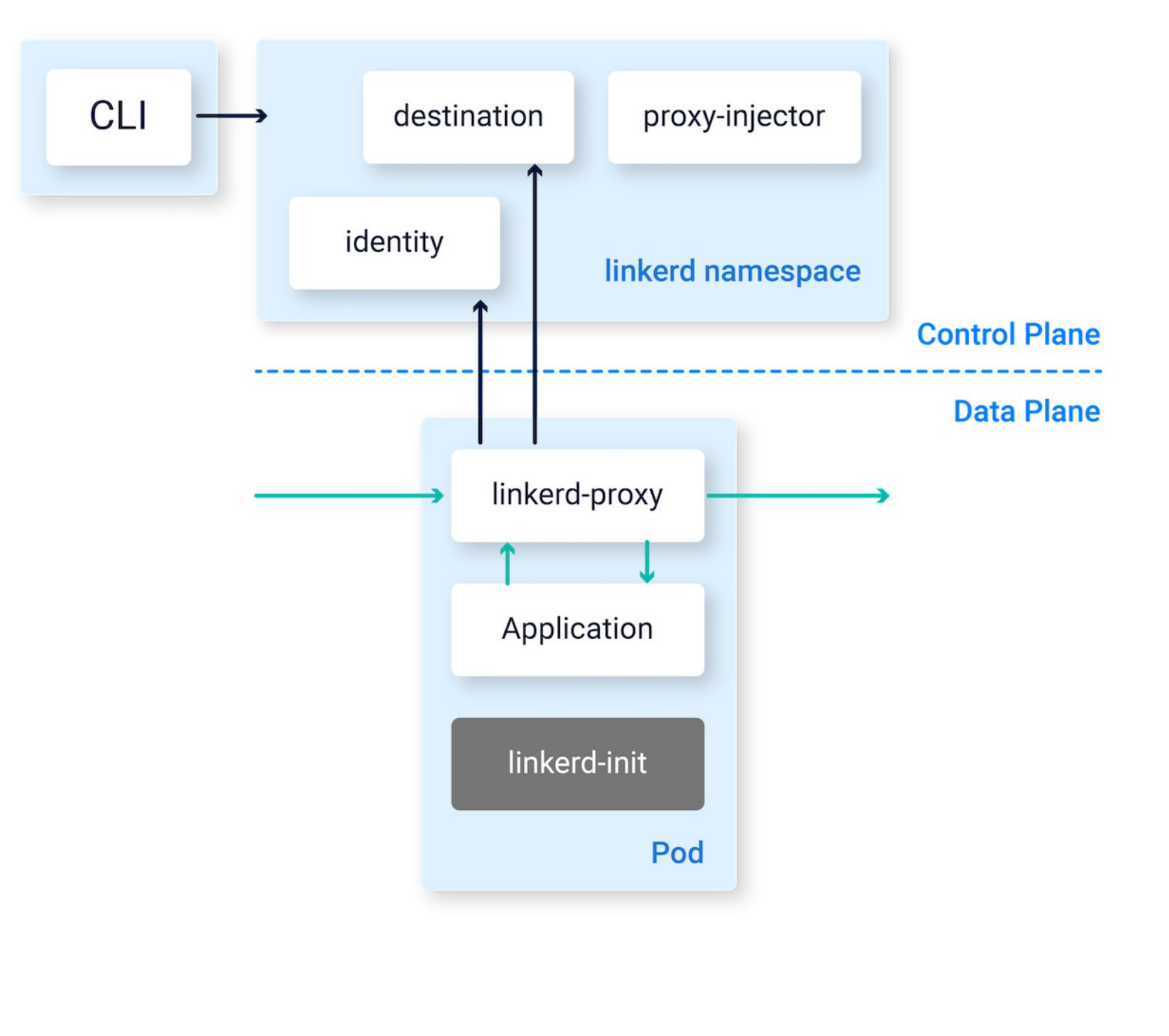
Data plane architectures - Sidecar

Sidecar-Pattern:

Each application pod contains an additional container, the “sidecar”.

Sidecars in Linkerd Service Mesh:

- Each pod in the mesh automatically is injected with a sidecar proxy
- Inbound and outbound traffic is routed through the sidecar proxy via IP tables rules. The rules are set either through the *linkerd-init* startup container or implemented via a CNI plugin.
- The sidecar proxy implements the mesh functionality.



Linkerd’s architecture

Data plane architectures - Sidecar

QA|WARE

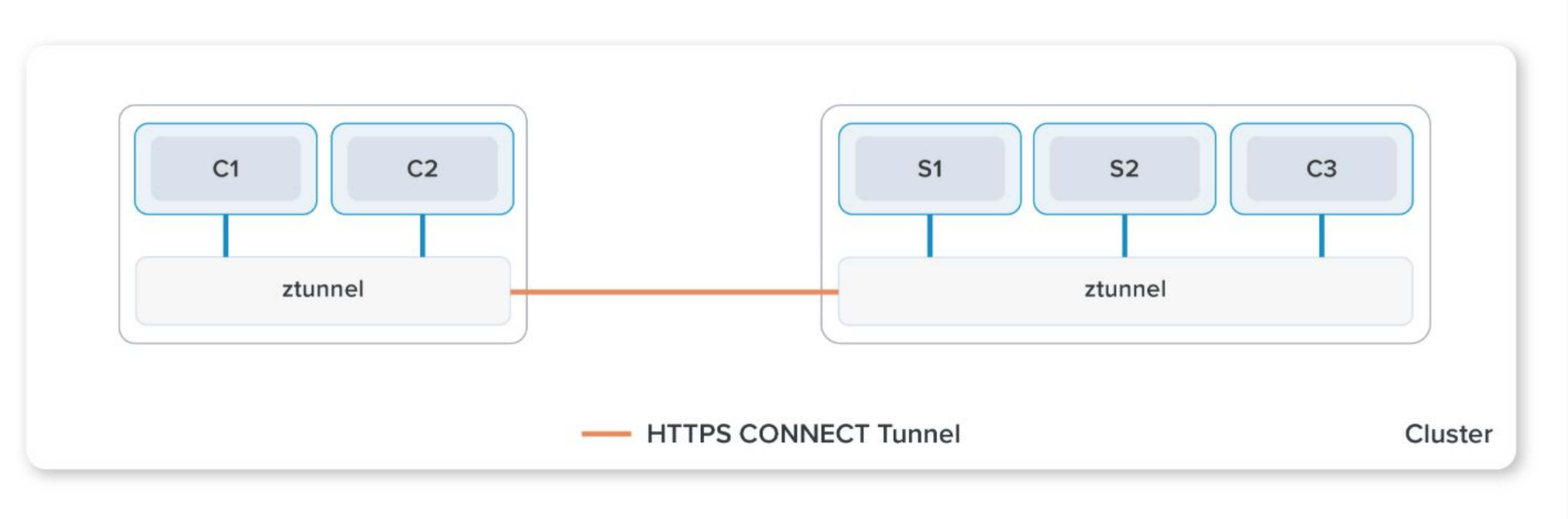
Advantages:

- **Resilience:** No single point of failure – as there is one proxy per application pod
- **Uniformity:** Traffic is routed uniformly through the proxy. This is identical for all applications
- **Multi-tenancy:** There is no shared infrastructure in the data plane between different parties
- **Scalability:** Proxy scales horizontally with the application
- **Zero code changes:** The application can be meshed without code modifications

Disadvantages:

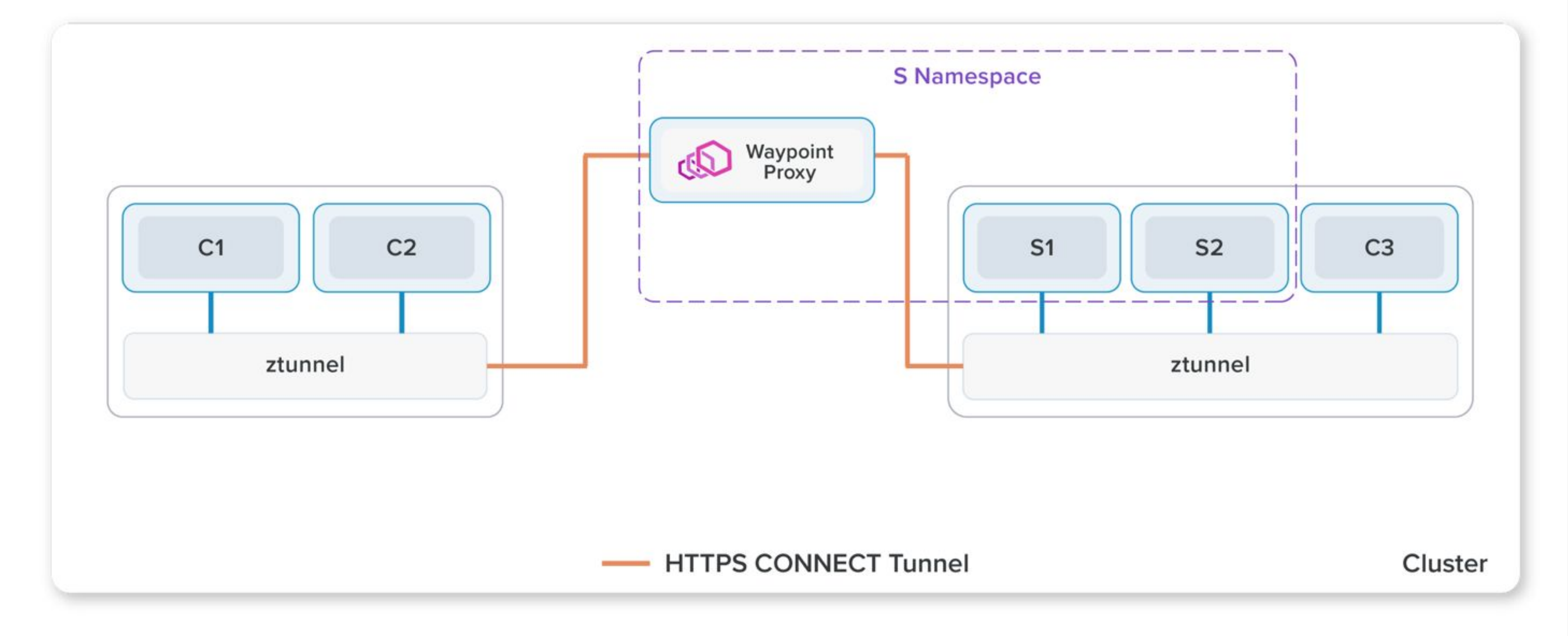
- **Resource Overhead:** Every pod in the mesh automatically gets a proxy injected, which in turn consumes resources like CPU and RAM.
- **Higher Latency:** All traffic is routed through proxies, requiring more hops. Therefore, Latency increases, which can be problematic depending on the application.
- **Resource Management:** The sidecar proxy runs as a container in Kubernetes and is subject to typical resource constraints. These need to be chosen and maintained carefully to ensure optimal resource utilization.
- **Security:** A new infrastructure component is deployed, potentially increasing the attack surface.

Data plane architectures - Ambient Mesh



Only supports Layer 4 features. As a result the implementation is much simpler and way more performant.

Data plane architectures - Ambient Mesh



In case Layer 7 features are required, envoy proxies are deployed.
All Traffic directed via the zTunnels through the proxy..

Data plane architectures - Ambient Mesh

Advantages:

- **Scalability**: Waypoint proxy scales horizontally, independently of applications
- **Zero code changes**: The application can be meshed without code modifications
- **Resource efficiency:** Expensive Envoy proxies are only used when L7 features are needed. In total, fewer proxies are deployed overall, as one waypoint proxy can be used per namespace.
- **Flexibility:** Can theoretically be deployed alongside the sidecar model

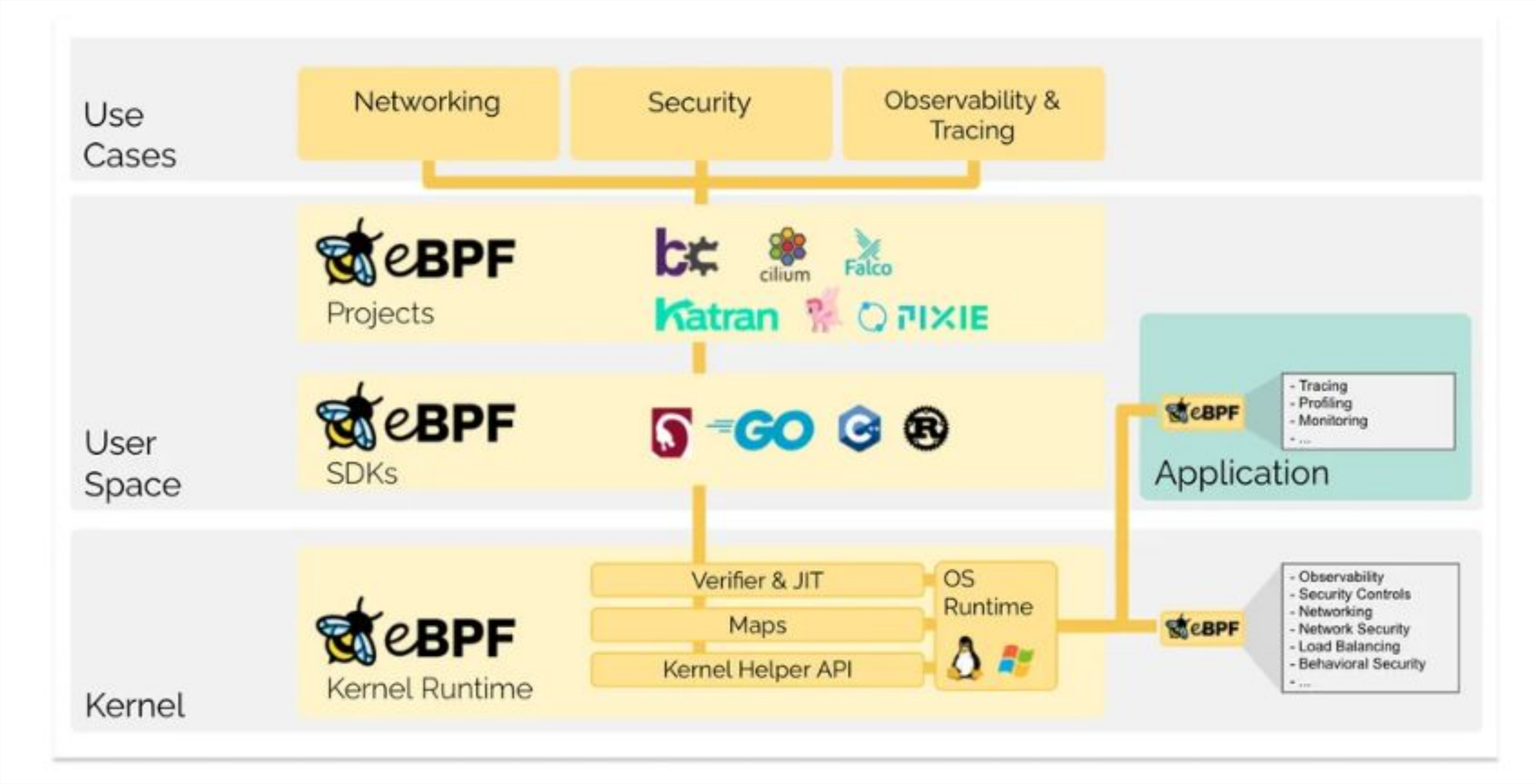
Disadvantages:

- **Potentially even higher latency:** When L7 features are needed, traffic is routed through a waypoint proxy. However, the proxy may no longer be on the same node as the application.

Evolution: extended berkeley packet filter (eBPF)

- Historically derived from the Berkeley Packet Filter (BPF).
- BPF was primarily a network tool.
- eBPF has significantly extended BPF, so the acronym essentially no longer makes sense.
- eBPF allows the user to dynamically extend the kernel with programs.
  - These programs run in a sandbox within the kernel.
  - They are JIT-compiled and must pass a verification engine in the kernel.
  - These programs run event-driven and subscribe to existing hooks in the kernel, e.g., system calls.
  - If there is no existing hook, a program can still be attached to relatively freely chosen points using a kProbe/uProbe.

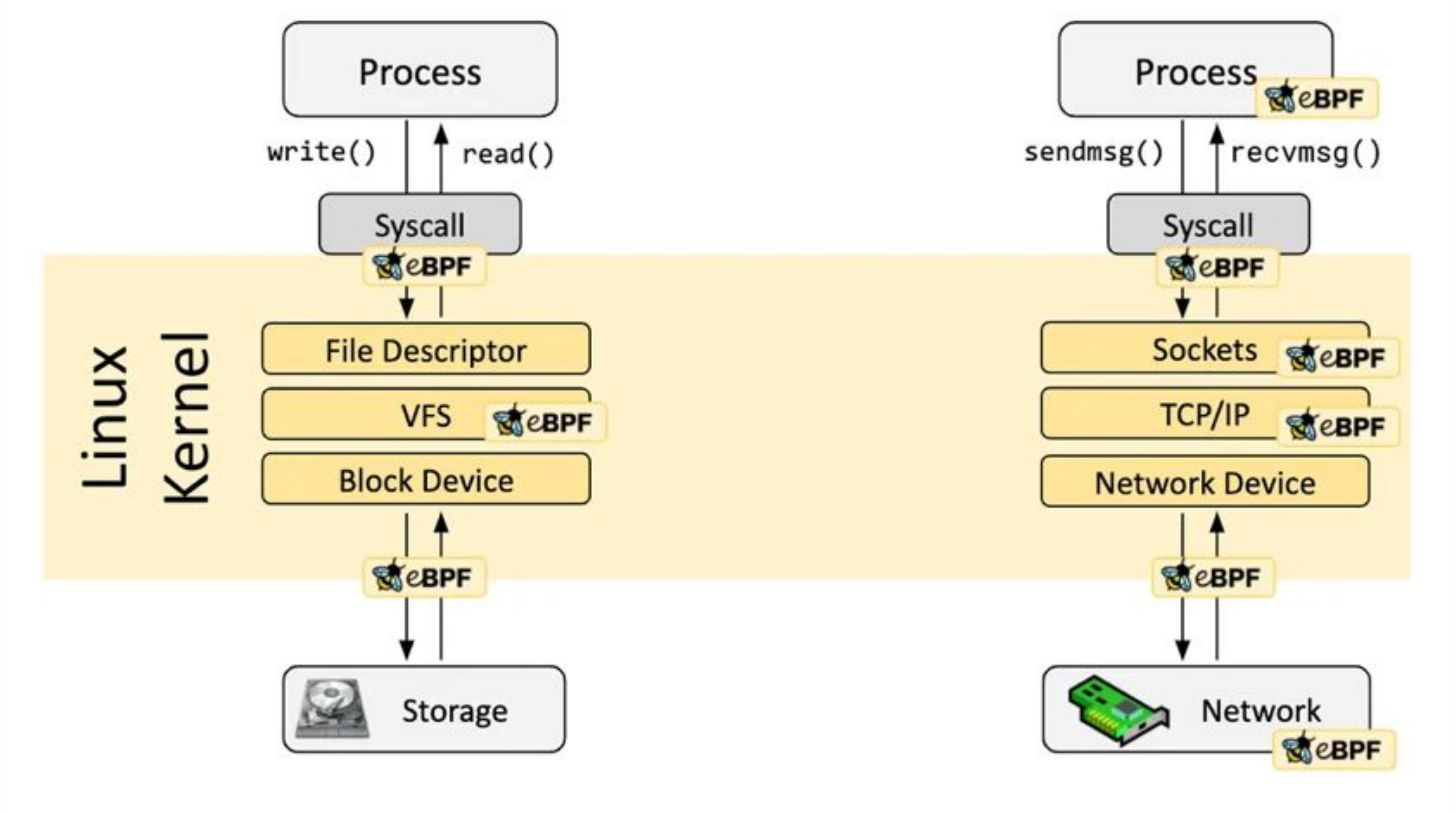
Evolution: extended berkeley packet filter (eBPF)



Quelle: https://ebpf.io/what-is-ebpf/#what-is-ebpf

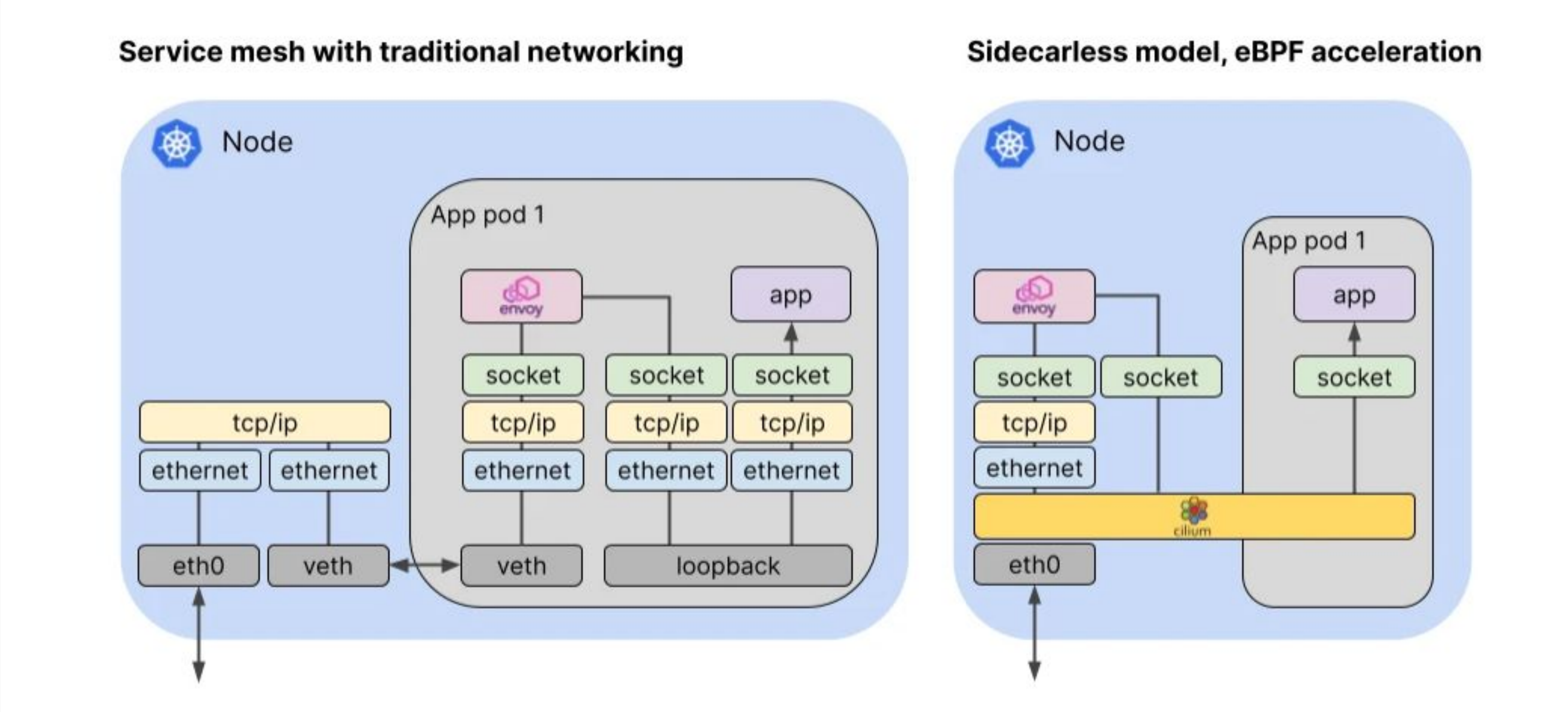
Evolution: extended berkeley packet filter (eBPF)

QA|WARE



Quelle: https://ebpf.io/what-is-ebpf/#hook-overview

Data plane architectures - eBPF based



Quelle: https://cilium.io/blog/2021/12/01/cilium-service-mesh-beta/

Data plane architectures - eBPF based

Advantages:

- **Zero code changes:** The application can be meshed without code modifications
- **Resource efficiency:** Performance is superior to the previous model
- **Flexibility:** Many features can be implemented in eBPF. Only some L7 features still require an L7 proxy

Disadvantages:

- **Complexity:** New, challenging technology. How do I debug this as a user?

QA|WARE

LinkerD in action!