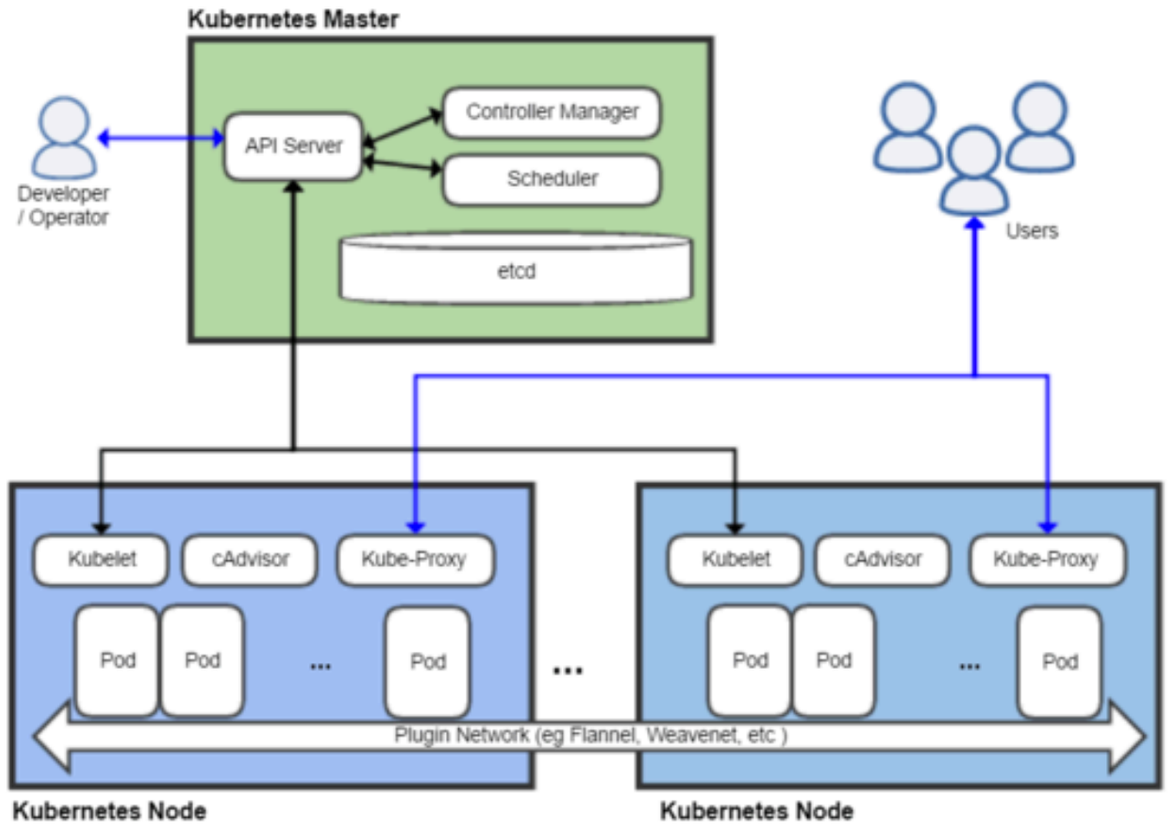
Intro to Clover

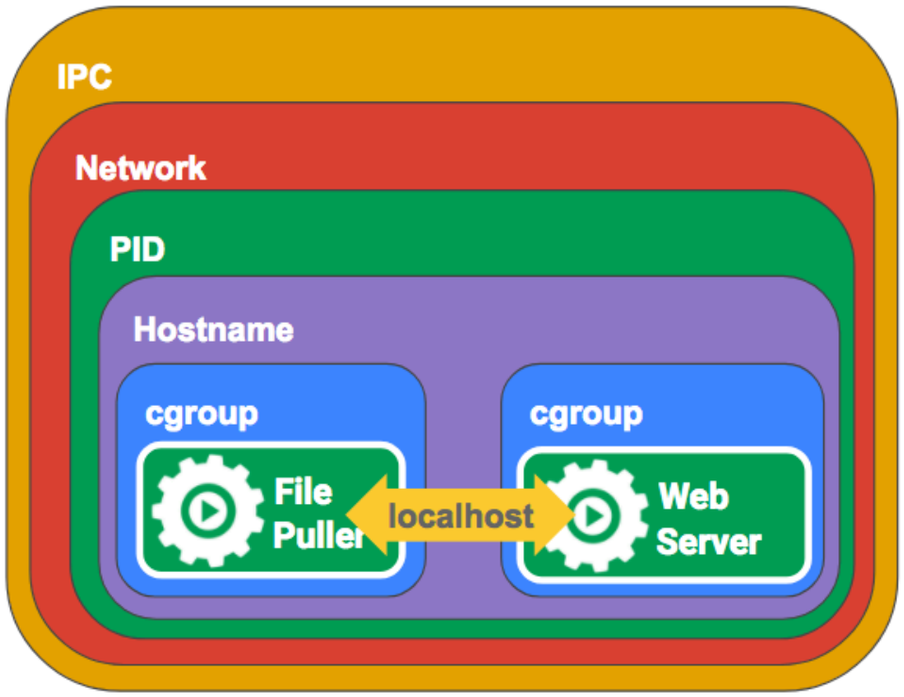
Fraser to Gambia (and beyond)

Brief Intro to Kubernetes

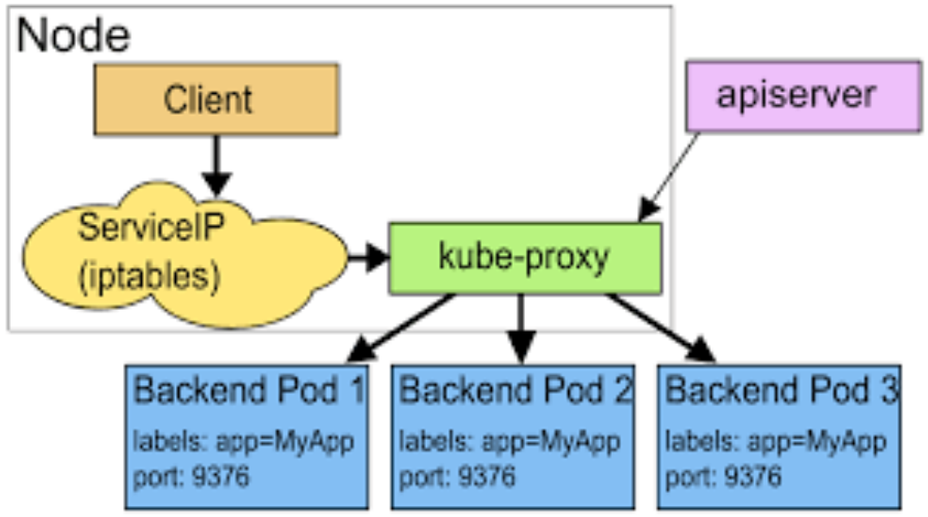


- Basic objects: nodes, cluster, kube-master
- Docker orchestration platform conforming to cloud native application concept for infrastructure
  - One master (and one or more standby kube-master if desired), minion nodes join cluster as they come
  - A Linux process name “kubelet” will run on every node for local management, health probe, setting up environment ….etc
  - Core to Kubernetes is its scheduler, which (by default) schedules (and balances) workload on all the worker nodes (master node can be a worker node also)

Brief Intro to Kubernetes (Pod & Service)

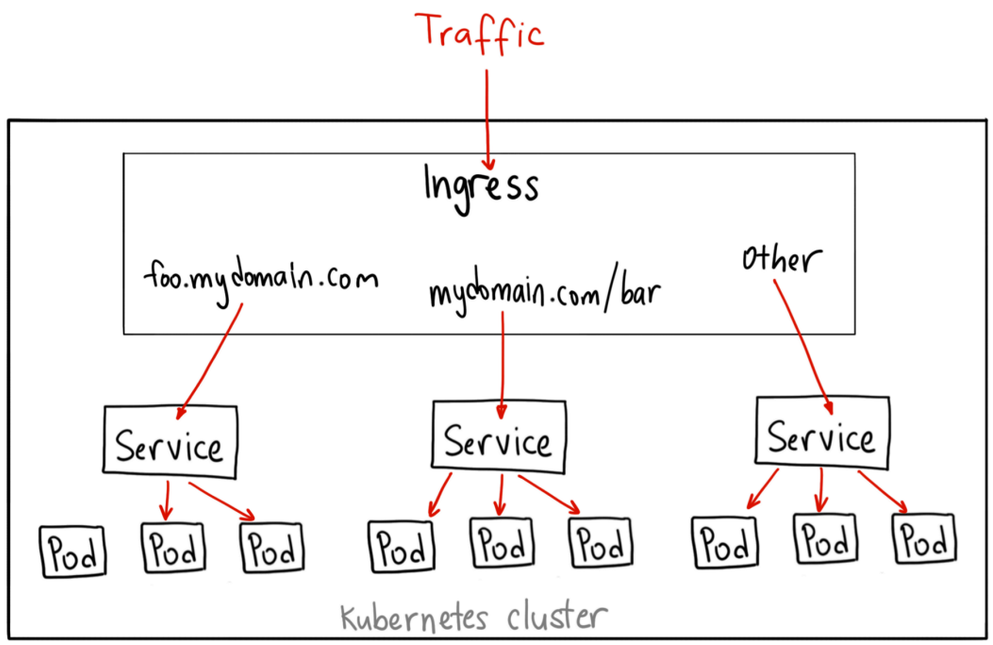


- Pod is the finest granular object in k8s (and it is the unit of scaling up)
- Pod can have one or more containers running inside (different cgroup, same namespace)
  - Same network namespace means containers within a pod can communicate with each other via localhost
  - Typical pod only has one interface (eth0 addressable inside the container, veth pair on host) —- one IP address per pod



- Service is what is exposed as an accessible entity (“endpoint”) in k8s, backed by pods. Basically a “microservice”
- Service maps to the set of pods via labels
- Service names == domain name entry in DNS server maintained by k8s
- A service moreover can expose one or more ports (i.e., TCP port). kube-proxy, which runs on each node, maintains the mapping of the external service:port to a backend pod (on a node). When multiple ports are exposed for a service, port needs to be named

Brief Intro to Kubernetes (external access to Services)

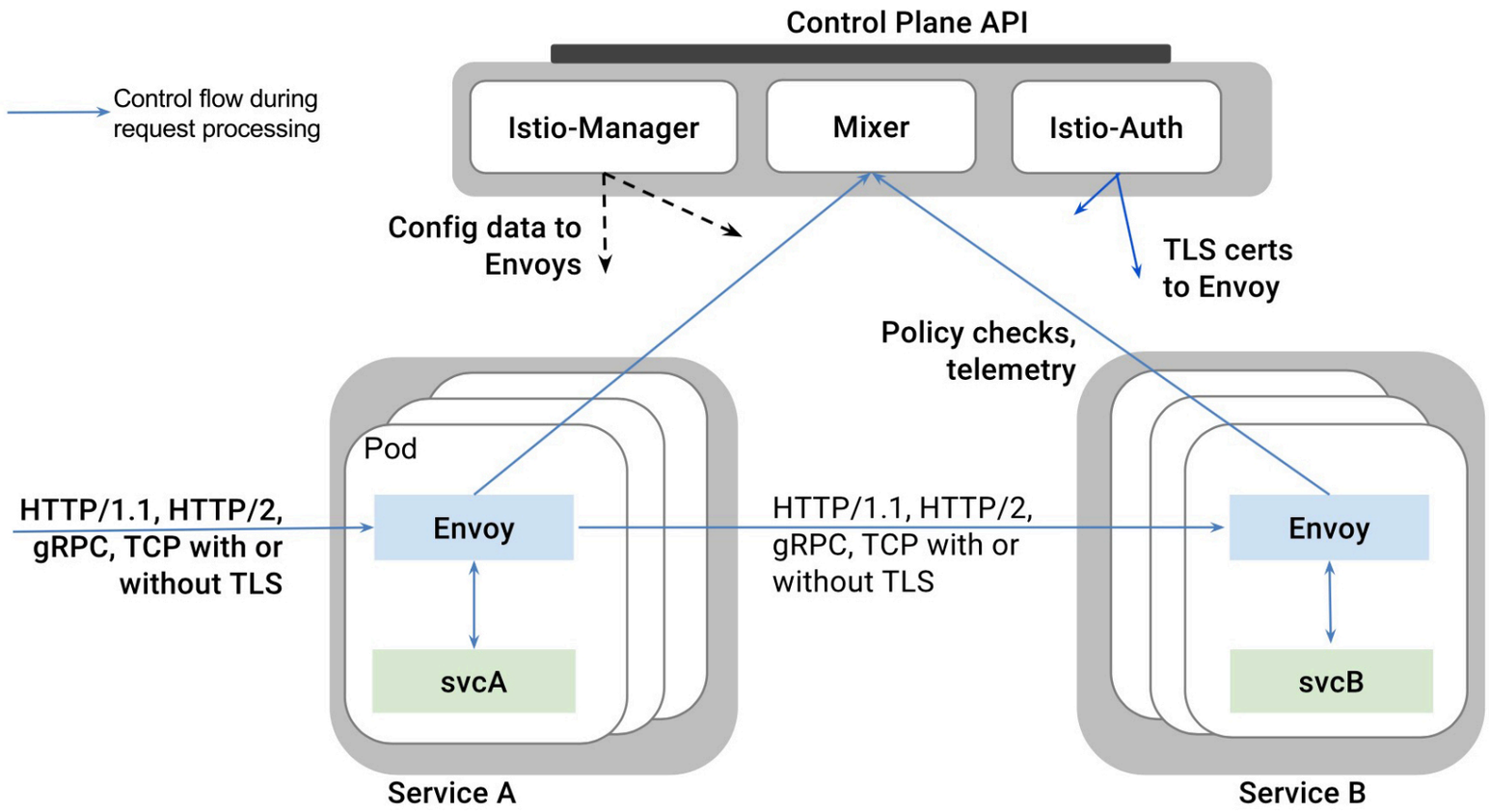


- There are at least three mechanisms to expose services to external users: NodePort, LoadBalancer (needs to map to a cloud provider), and via an ingress controller. Of these, only the ingress controller is relevant to our discussion

- Ingress controller is basically having an HTTP load balancer sitting in front of your cluster

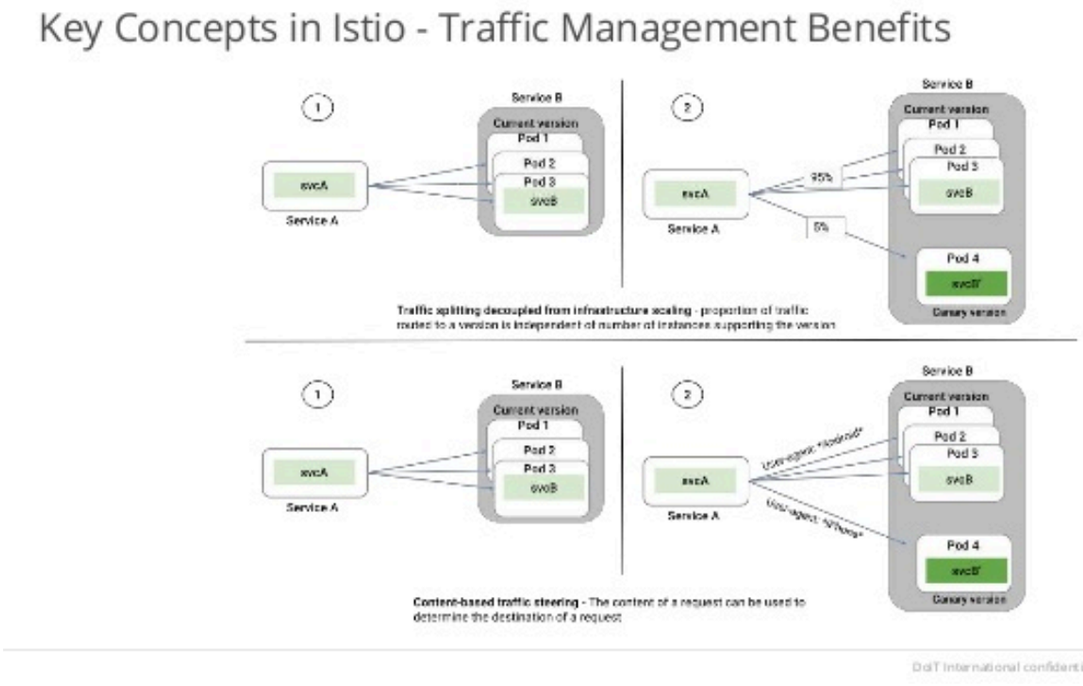
    - Allowing cluster to expose different paths (HTTP) and map that to different micro service entry point

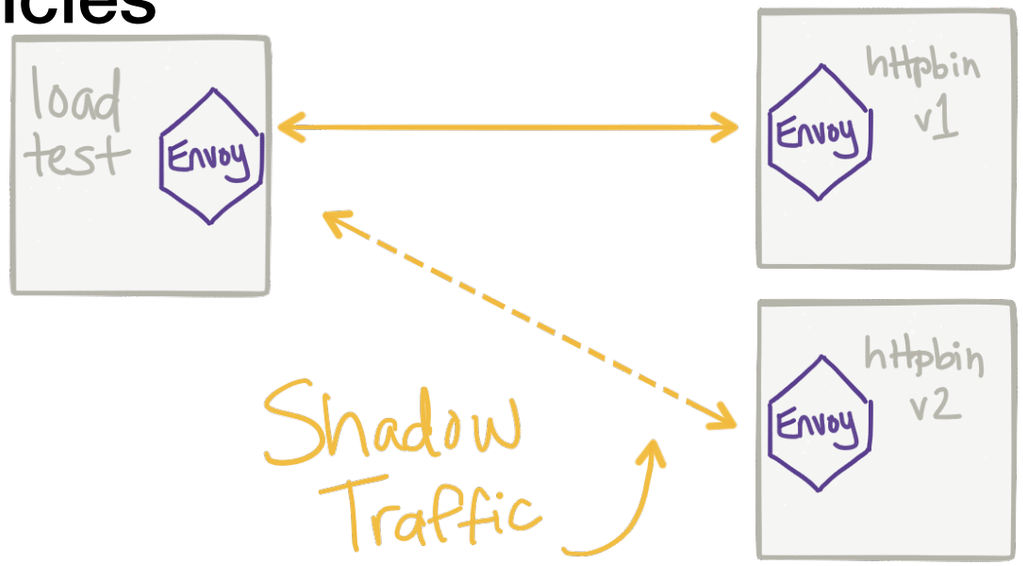
Brief Intro to Istio

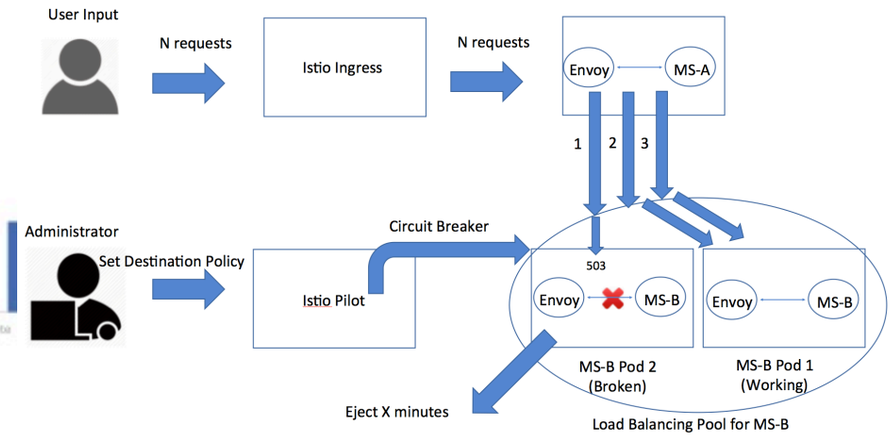


- Need for service mesh in k8s setup:
    - decouple infrastructure logic (IP, where pod is scheduled…etc) from application development
    - Allowing declarative language from application to express infrastructure needs
- Basic: sidecar (service proxy in each pod), mixer maintains policy definition, and pilot (Istio-manager now in 1.0) configures the service proxy
- Traffic flow:
    - From external to cluster: istio-ingress
    - From pod to pod: all inbound and outbound traffic goes through the sidecar Envoy

Brief Intro to Istio: Traffic Policies







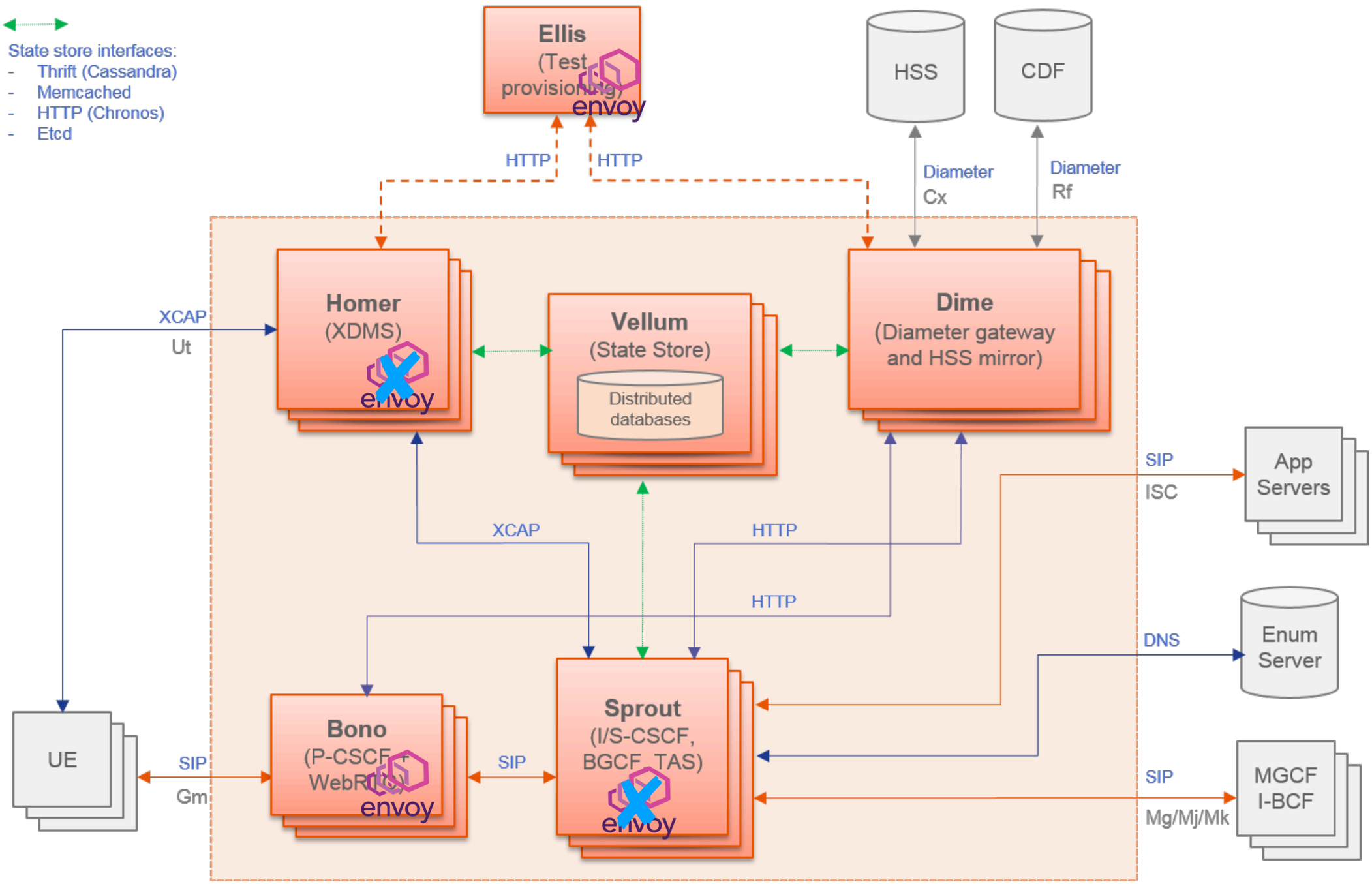
- The core Istio “network” policies include:
  - Traffic shifting: assign % of traffic to a matching label, assign traffic to a destination label based on HTTP header info (agent, cookie…etc)
  - Traffic Mirroring: mirror traffic to two (or more) destination labels
  - Circuit Breaking: disconnect traffic to a destination label based on HTTP header fields
  - Rate limiting: limit how many connections / sessions any clients can have per a time quanta (e.g.: one second)
  - Fault injection: inject delay to any connection to pods associated with a destination label
- Mostly operational, fault isolation type policies

OPNFV Clover

- Examine how cloud native technologies / open source projects can potentially be used for NFV use cases
- Assuming Kubernetes with Docker as container runtime
- Main Areas of Emphasis:

1. Use of service mesh —- beyond just performance impact, whether the operational model makes sense for CNF
    - Istio
2. Examine use of cloud native tools for visibility, traceability, operability, and debuggability
    - Jaeger —- tracer for OpenTracing (trace collection)
    - OpenTracing —- client API to invoke traces
    - Prometheus —- time series data base for metrics / monitoring
    - fluentd —- log collection
3. Examine continuous delivery framework for cloud native applications
    - Spinnaker (CI/CD framework)
4. Target deployment environments
    - Baremetal (Pharos)
    - GKE

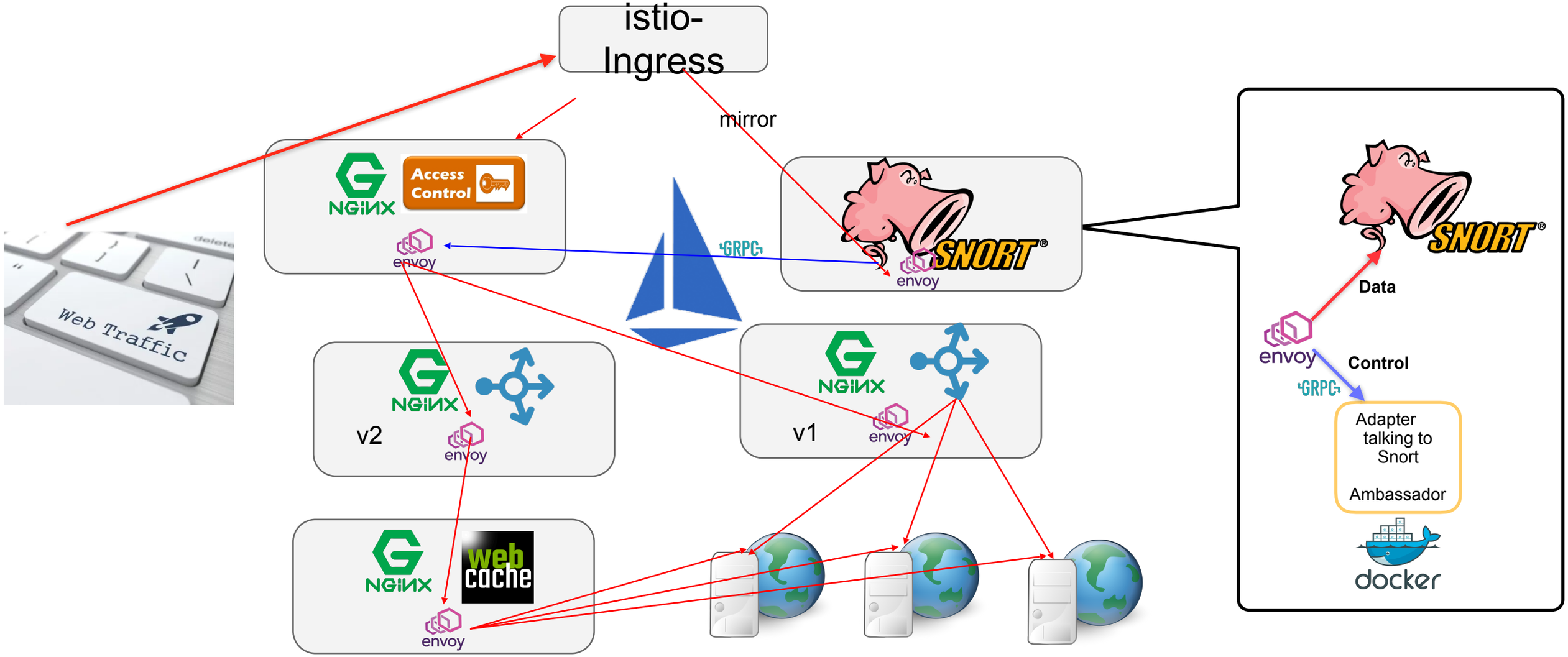
Clover Fraser
Clearwater (IMS) on Istio



- Poor result: most of the components don't really work (traffic blackhole or even Envoy crashing)

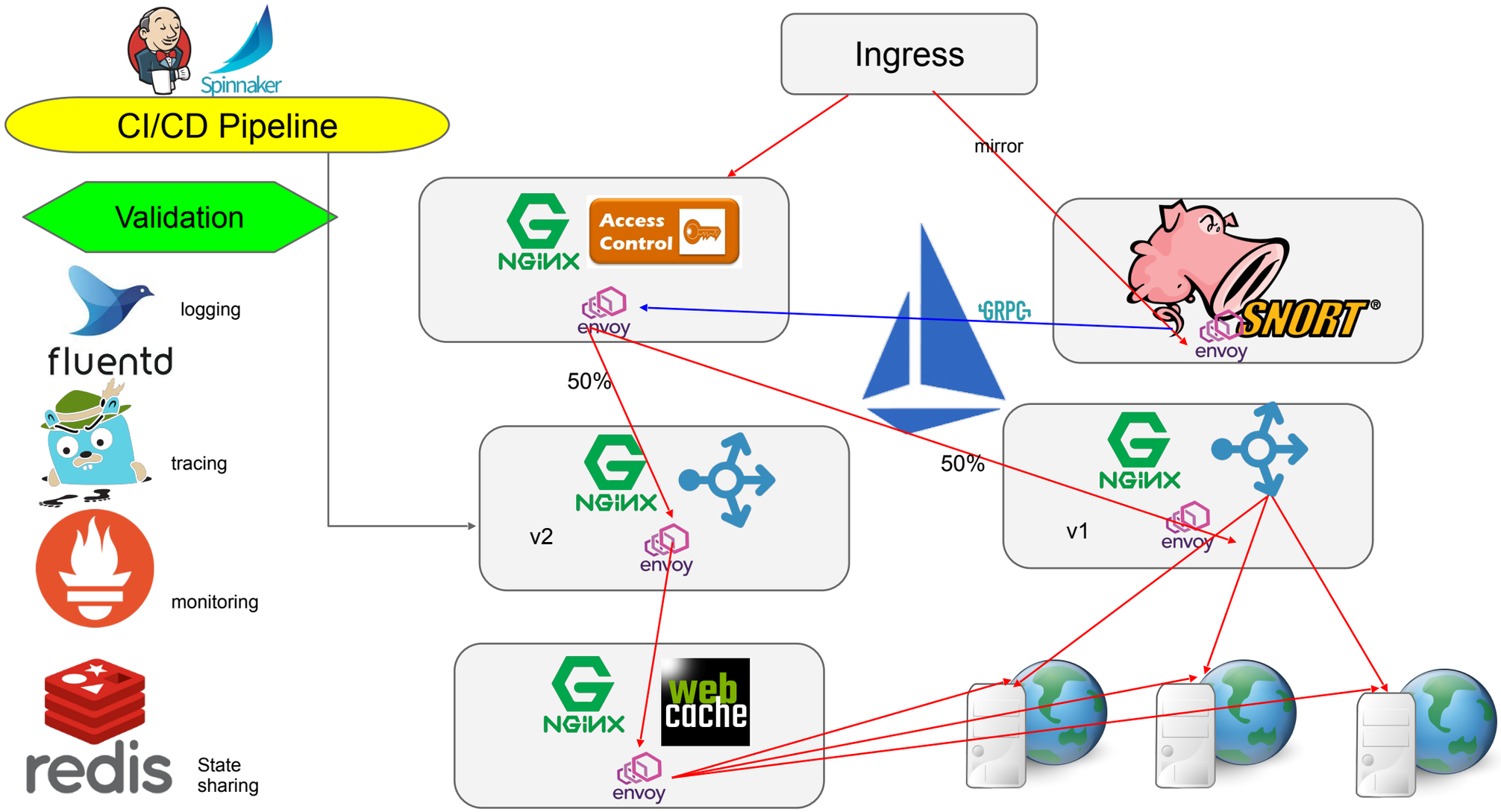
- It does work with Bono and Ellis

Clover Fraser
Sample Network Function



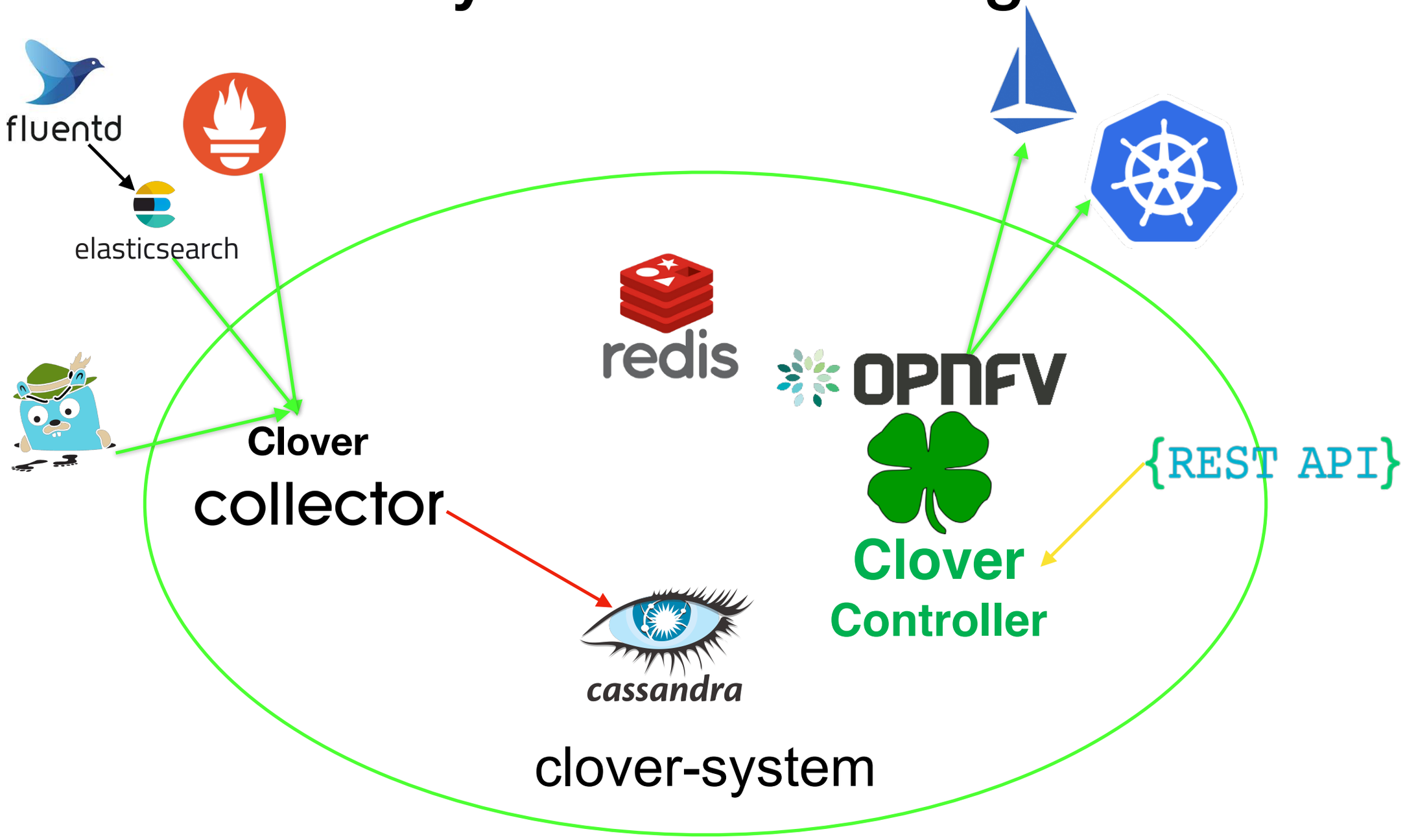
1. A web service gateway with a variety of essential HTTP based network services —- each of which is open-source, battled tested, hight-performance, and performs its specific network function extremely well (in this demo, the majority of the micro-services were built with specific configuration of Nginx to allow it to serve different role —- which exemplifies the micro-service concept: a software module that does ONE function very well)
2. As both data and control traffic are HTTP based, both data and control traffic will traverse through the service mesh
3. Takeaway: HTTP based traffic works well with Istio / Envoy —- but also in the context of Kubernetes also. Pods send requests to other pods via dynamic (per request) look up of domain name, and therefore the IP address of the next micro service is not resolved until the time the request is able to be sent out, which correspond well with the dynamic nature of Kubernetes pod scheduling
4. Potential use of HTTP tunneling for pod to pod data path, and use control to do chaining

Clover Fraser
A-B Testing via Tracing Data from Envoy / Jaeger



1. CI/CD deploys L7 proxy version 2
2. Istio policy applies for 50% traffic to control (v1) and 50% to variant (v2)
3. Clover software gathers logging / tracing / monitoring and state info to validate “success” or “fail” during a time of traffic
4. If success, Istio policy of moving 100% traffic to v2 is applied
5. Clover software gathers info to validate 100% traffic to v2, and results met “success” criteria
6. Fault injected delay to v2 path to simulate failure, and confirm “rollback” case

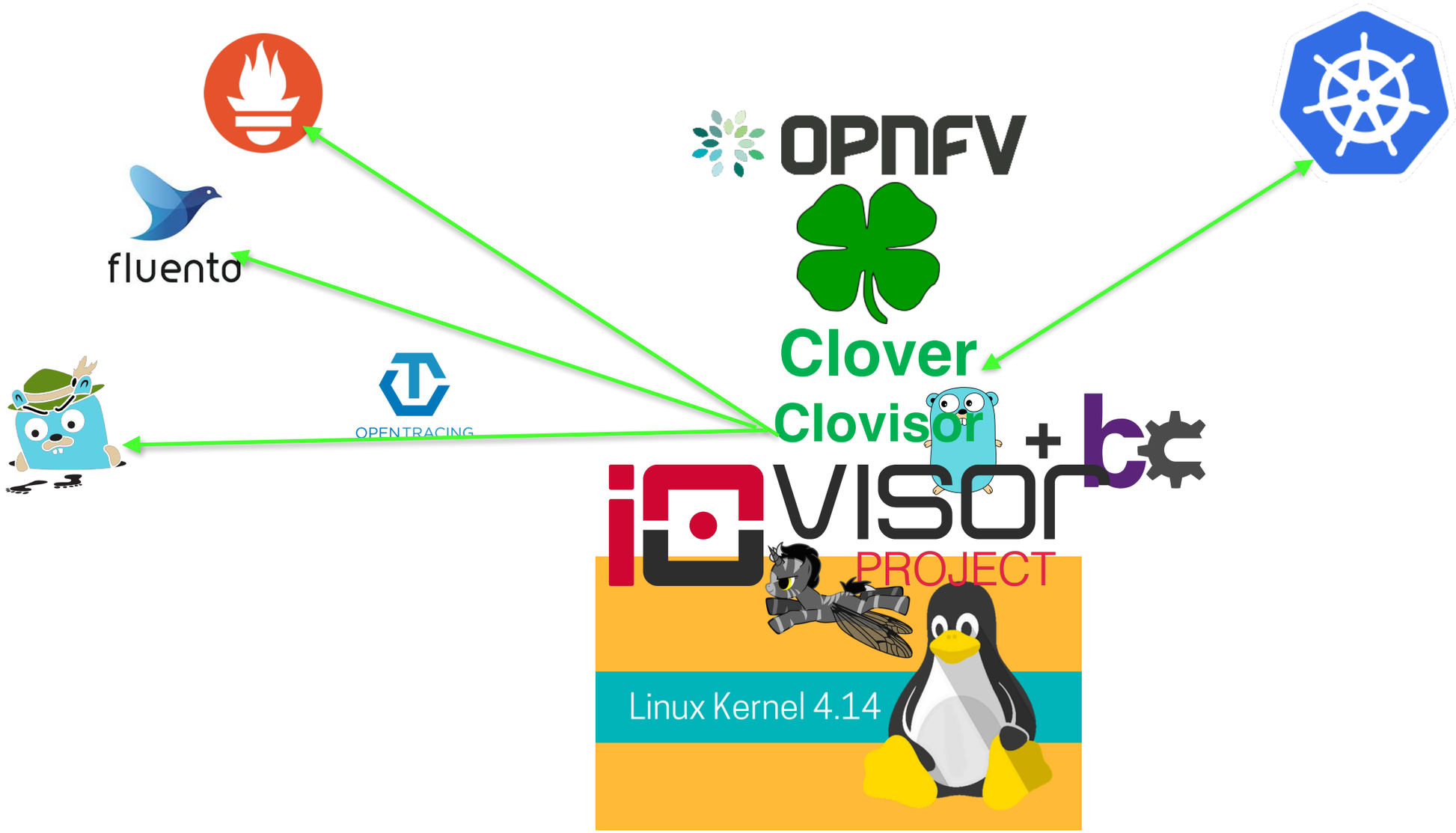
Clover Gambia
Visibility and Control Engine



1.The core software module runs on clover-system namespace, and is responsible for installing Clover related software packages
2.At its core is the Clover Collector —- which reads from various data sources and organize / giving structure (i.e., allowing SQL like query) to the incoming raw data
3.A client program cloverctl is implemented, and provides CLI to all core Clover functionality

Visibility Demo

Clover Gambia
Clovisor



1.Lightweight, low latency network tracing module
2.Utilizes IOVisor (bcc, gobpf) and eBPF to insert bytecode for both ingress / egress direction of a pod
3.In cluster client to automate process of monitoring and service port / protocol info
4.Stream trace / stats / metrics / logs to respective tracer / collector modules

Clovisor Demo

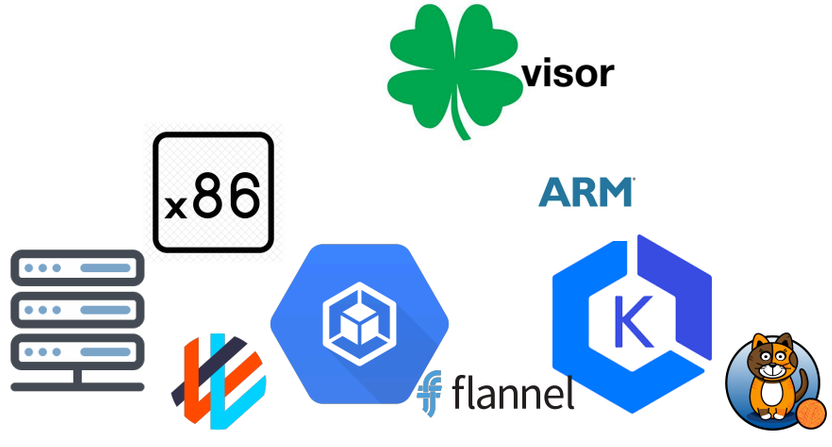
Clovisor: Network Tracing… the Cloud Native Way

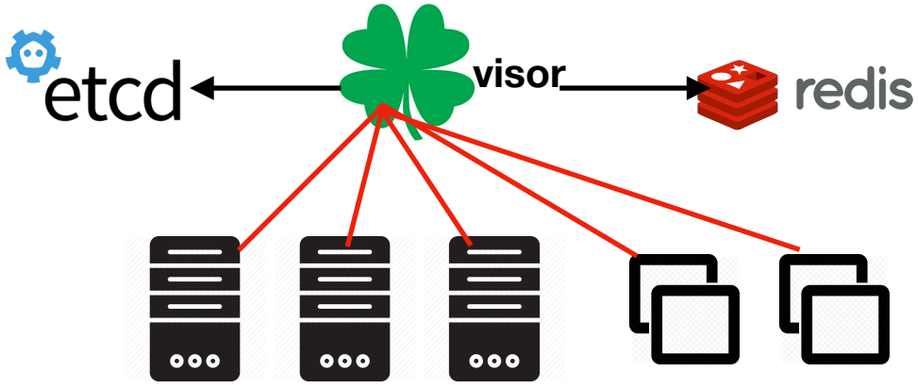
1. Cloud Native:

a) Cloud Provider Independent
- Bare-metal servers, GKE, EKS…etc

b) CNI Plugin Agnostic
- All CNI plugins should work unless such plugin does kernel bypass

c) CPU Architecture Independent
- Any architecture supported by Linux (x86, ARM…etc), code currently tested with kernel versions 4.14 and 4.15





2. Implemented with Cloud Native Design Methodologies:

a) Config Decoupled from Compute
- Config store in backing store or through environment variables

b) Relatively Stateless
- TCP connection/session tracking only dynamic states

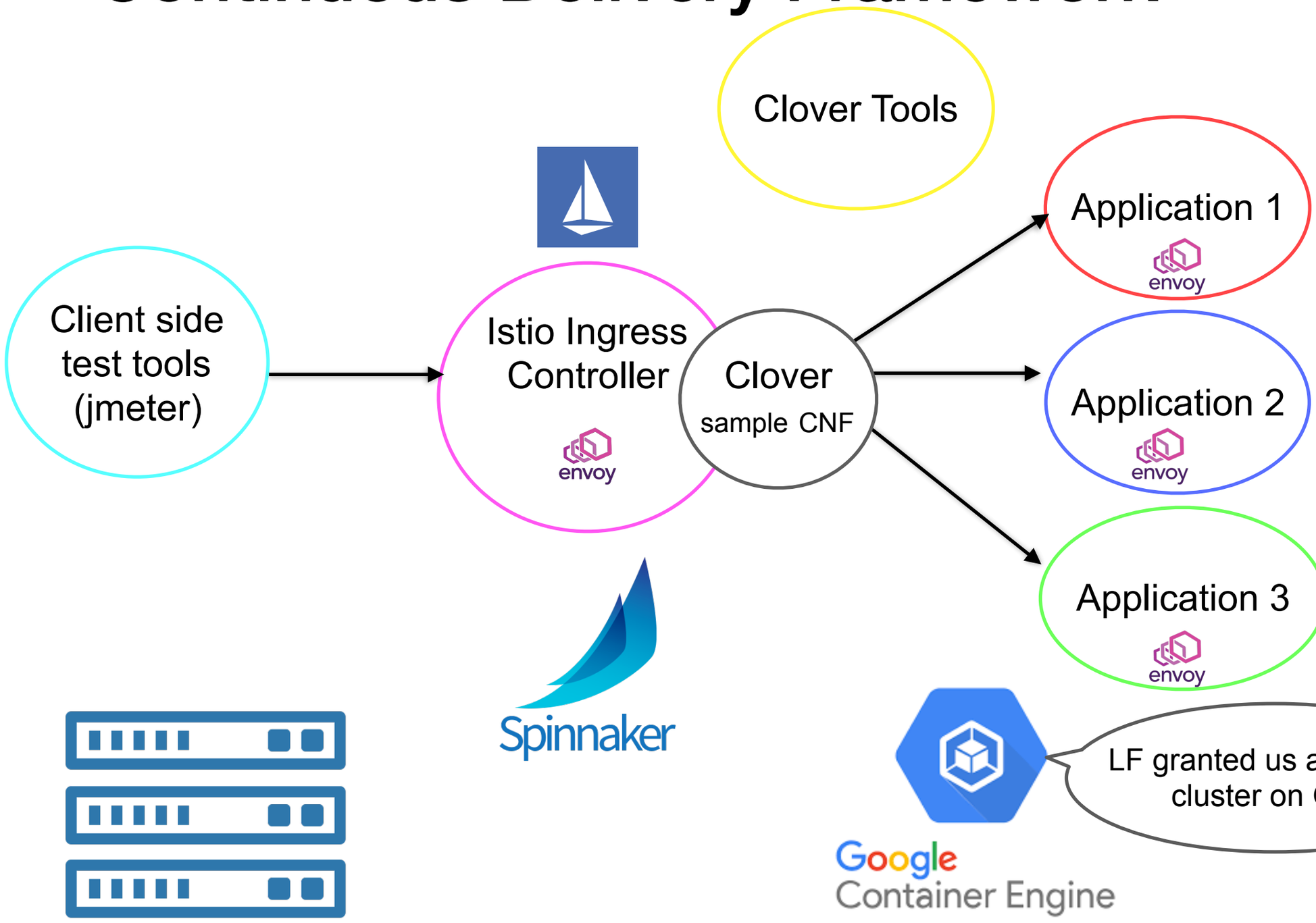
c) Scale-out Architecture
- Pod monitoring partitioning via election from datastore
- DaemonSet —- linearly scale on each node in cluster

3. In-depth Integration with Cloud Native Ecosystem Projects:

a) Built-in Kubernetes Client
- Monitoring k8s pod states

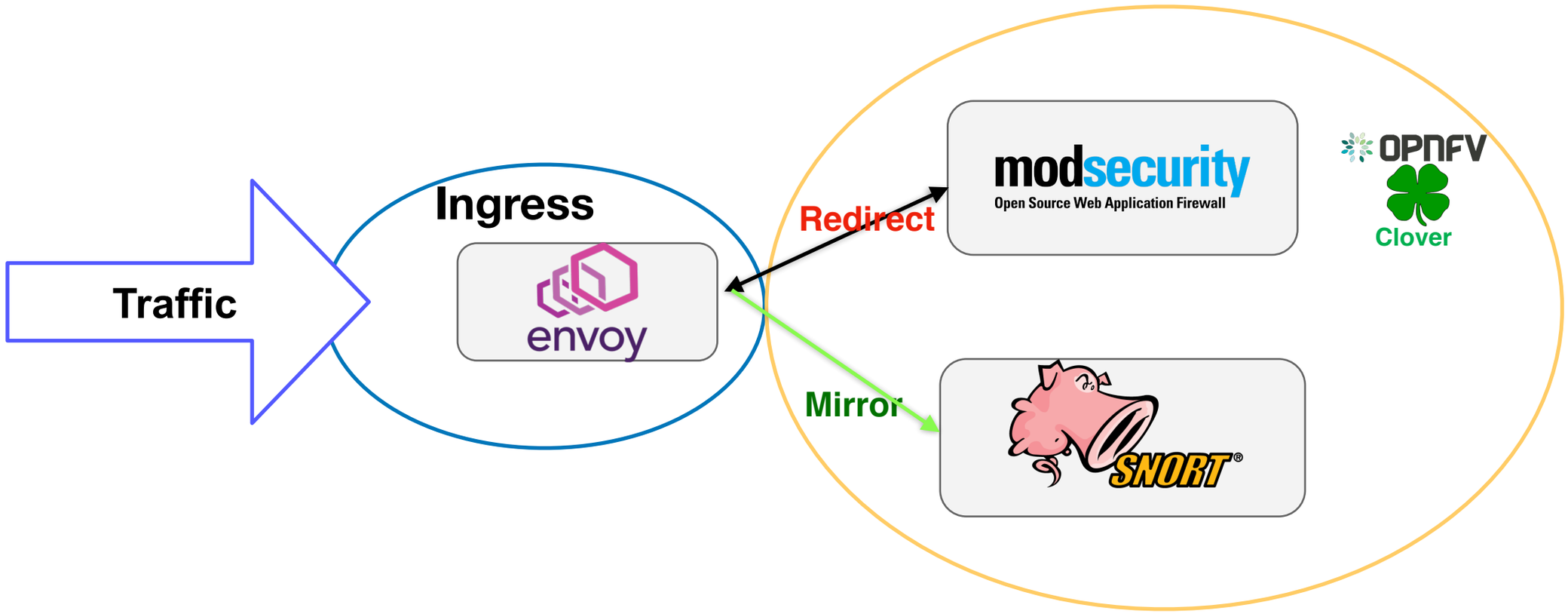
b) Integrate with CNCF Collector Projects
- OpenTracing to Jaeger, metrics to Prometheus

Clover Gambia
Continuous Delivery Framework



1. Goal: Deployment on bare metal and GKE —- and use it as CD platform
2. Spinnaker installed, hooked with OPNFV DockerHub, and whenever a component is updated (on the sample CNF side), the CD pipeline will be executed
3. Long running testbed

Clover Gambia
Add Gateway Network Functions to Istio



1. Network functions in clover-gateway namespace can serve as Istio gateway enhancement
2. Simply providing two main functions: web application firewall (WAF) and intrusion detection
    - Two primary service insertion mechanism: redirect (traffic flow affected by service) or mirror (“read” mode)

Clover Gambia (and beyond)

Istio Network Functions Extensions

Add relevant network functions to enhance security and visibility of Istio mesh

Visibility & Traceability

Core network tracing / monitoring

Expose data to backend datastore

Application logic for correlating data and Istio exposed data

Deployment

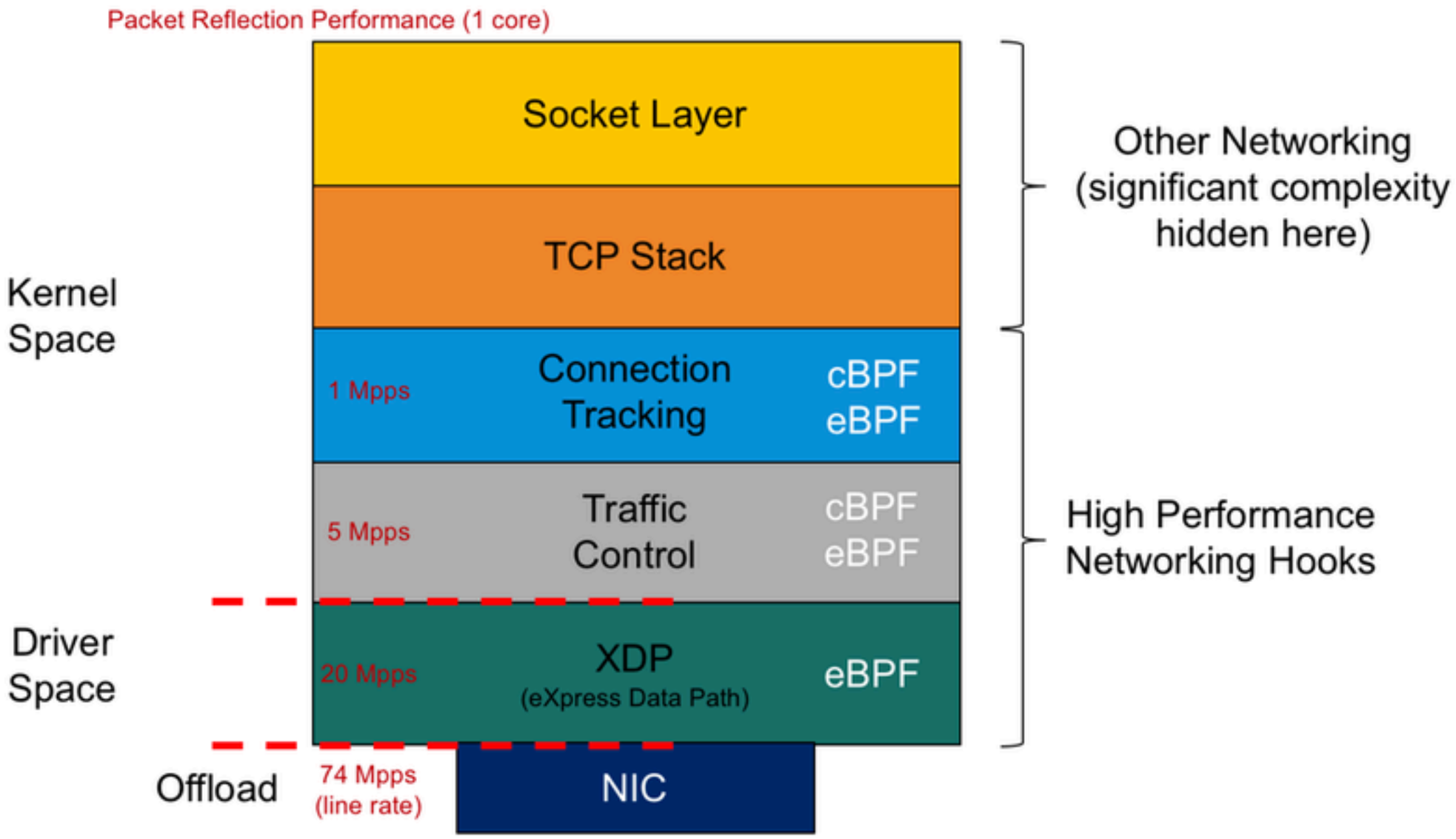
CD pipeline

Target environment and web applications to test ingress

Application specific SYSTEM and FUNCTIONAL tests

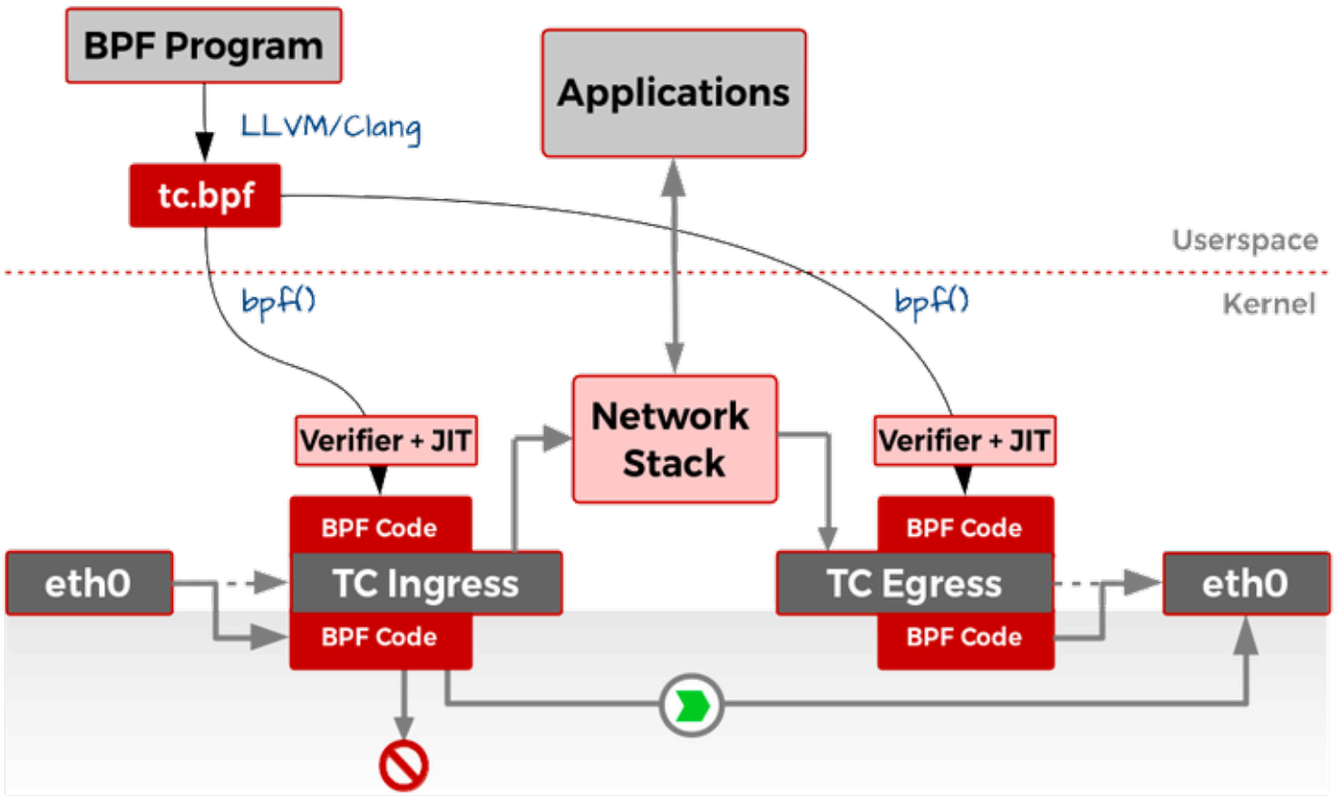
Backup Slides

eBPF: Where is the packet being intercepted?



IO VISOR PROJECT and

Enhanced BPF in Linux

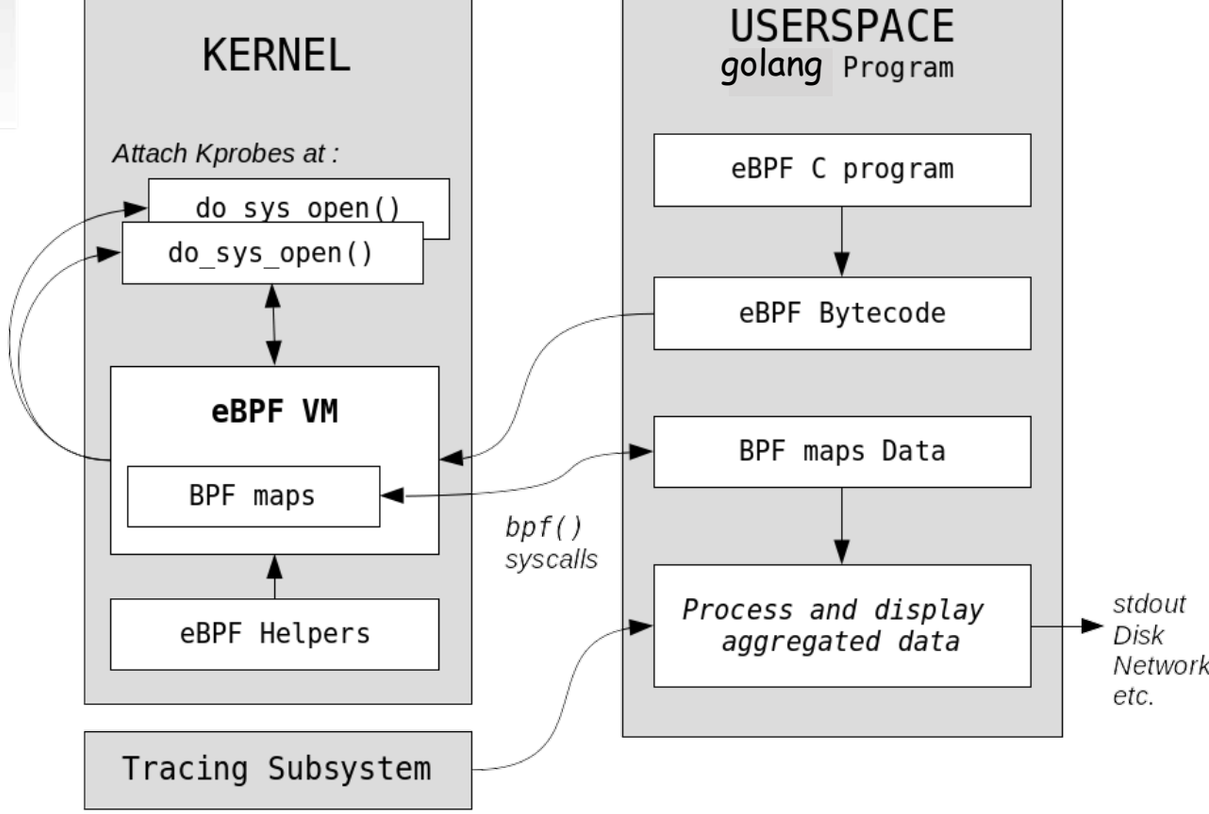


1.eBPF:

a) Inject bytecodes to kernel trace points / probes
- Event driven model

b) Networking: tc
- Utilizes Linux **tc** (traffic control) to inject bytecode on ingress and egress direction of a network interface

c) Verifier / JIT (just-in-time compiler)
- Verifier ensures bytecode does NOT crash kernel



2.IOVisor bcc:

a) Ease of eBPF Development
- Helper functions, kernel API wrappers…etc

b) Dynamic Validation and Compilation
- Userspace eBPF code written in ‘C’ is dynamically verified (static analysis) and compiled

c) gobpf
- Golang interface for userspace code —- much more performant than Python