Kubernetes Networking

Pod ↔️ Pod • Cluster Model • Services • Ingress • DNS • Policies • Mesh

Audience: Advanced / Intermediate (beginner-accessible)

Focus: Theory-first, data paths, and control semantics

> Ziel: Architektur- und Konzeptverständnis von K8s-Netzwerken: Pod-Kommunikation, Services, Ingress, DNS, Policies und Service Mesh. Keine Hands-on-Demos.

Agenda

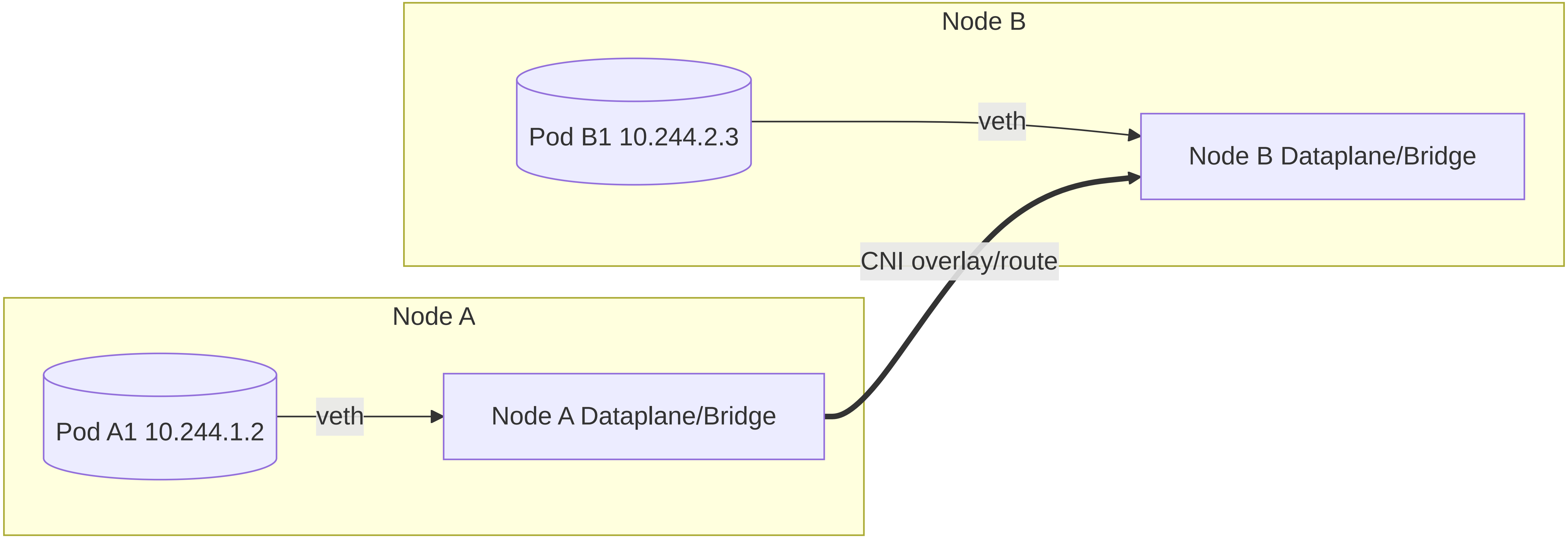
- Pod-to-Pod Communication
- Cluster Networking Model (flat network, no NAT)
- Services Overview
- ClusterIP, NodePort, LoadBalancer
- ExternalName Services
- Ingress & Ingress Controllers
- DNS in Kubernetes (CoreDNS)
- Network Policies (pod-level firewall)
- Service Mesh Overview (Istio, Linkerd)

> Agenda-Überblick. Wir folgen dem Datenpfad von Pod → Service/Ingress → CoreDNS/Policy → Mesh.

1) Pod-to-Pod Communication

Identity & Reachability

- **Pod IP:** Each pod gets an IP routable within the cluster.
- **Same Node:** veth pair connects pod netns to node bridge/dataplane.
- **Cross Node:** CNI provides routing/encapsulation (e.g., VXLAN, BGP, Geneve, eBPF).
- **No NAT within cluster:** Source/dest IPs are preserved across nodes.


Node B
Pod B1 10.244.2.3
veth
Node B Dataplane/Bridge
CNI overlay/route
Node A
Pod A1 10.244.1.2
veth
Node A Dataplane/Bridge

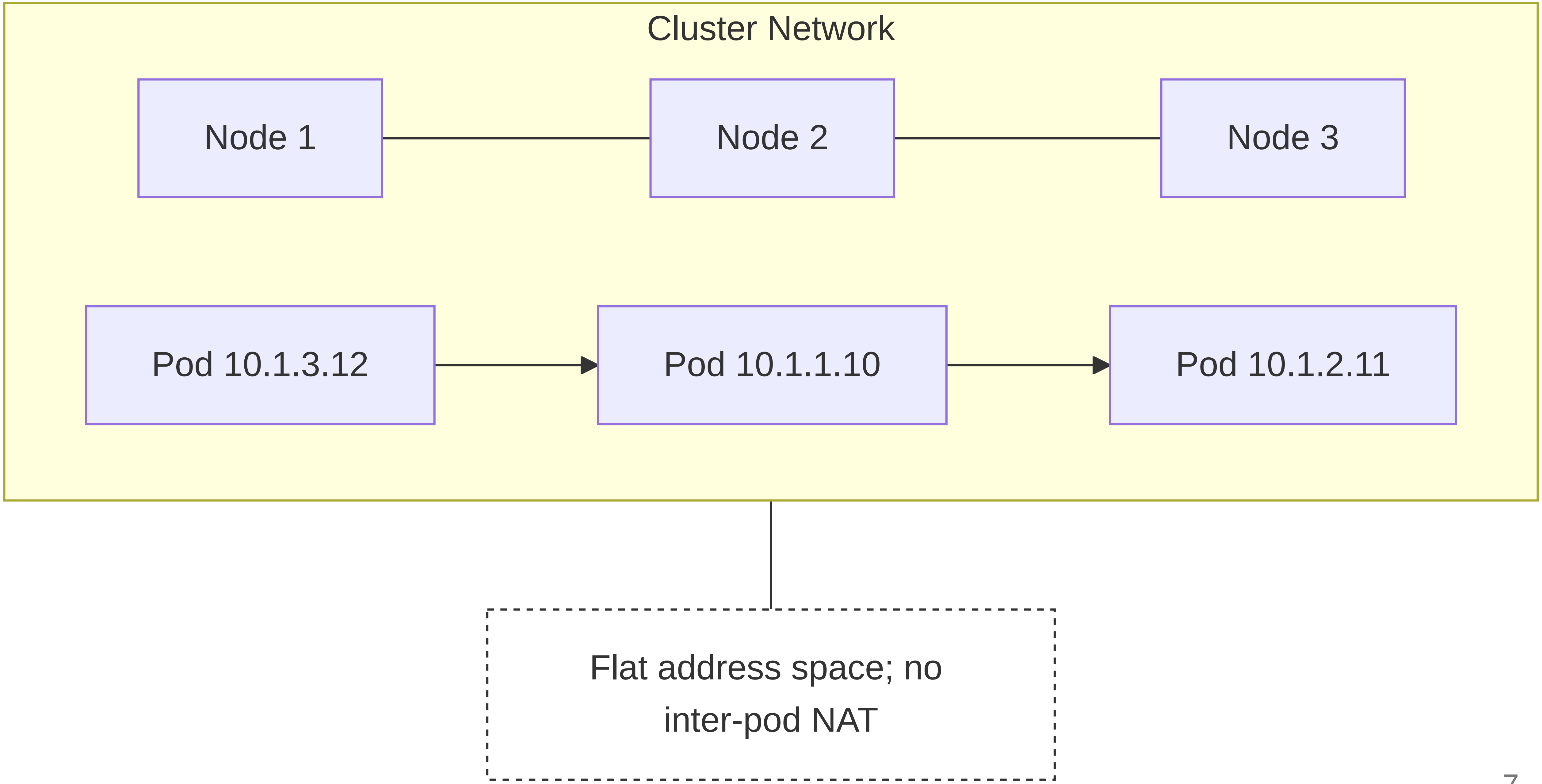
Service discovery: Usually via DNS → Service; direct Pod IP addressing is fragile.

> Pods haben eigenständige IPs. CNI sorgt für Node-übergreifendes Routing/Tunneling. Direkte Pod-IP-Nutzung ist instabil (Ephemerität).

2) Cluster Networking Model

Flat Network, No NAT (Core Assumptions)

- Every pod can communicate with every other pod without NAT.
- Nodes & pods see each other’s IPs; packets are not masqueraded by default.
- CNI plugin implements this model: e.g., Calico (routing), Flannel (VXLAN), Cilium (eBPF).
- Service abstraction provides stable virtual IPs/endpoints on top.



Cluster Network

Node 1

Node 2

Node 3

Pod 10.1.3.12

Pod 10.1.1.10

Pod 10.1.2.11

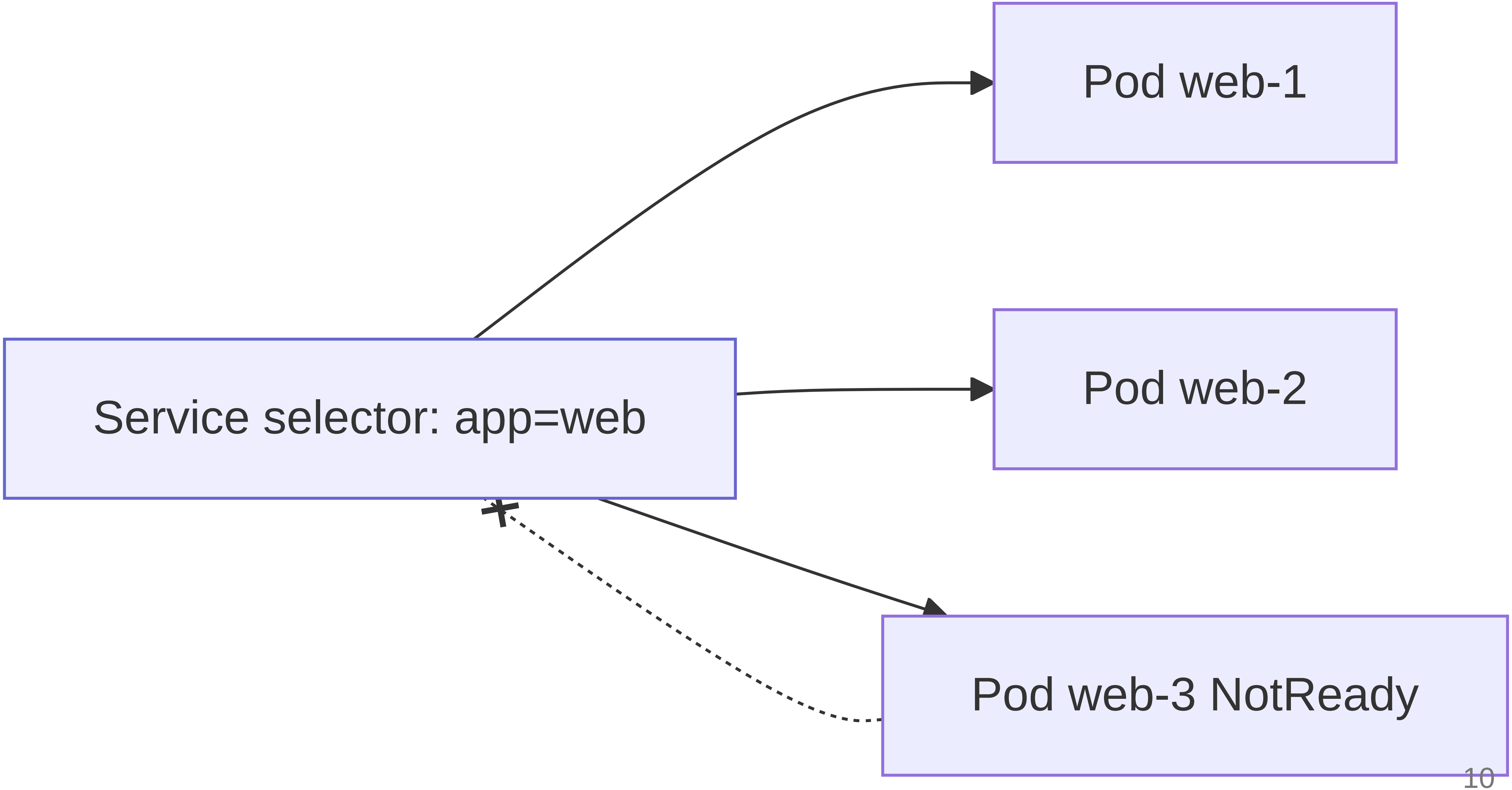
Flat address space; no inter-pod NAT

> „Flaches“ Netz: keine NAT zwischen Pods. CNI implementiert Overlay oder Routing, um dieses Modell herzustellen.

3) Services Overview

Stable Virtual Endpoints

- **Why Services?** Pods are ephemeral; Services provide stable names/IPs and load-balance across healthy endpoints.
- **Selectors:** Match pod labels → build EndpointSlice list dynamically.
- **Types:** ClusterIP (internal), NodePort (node-exposed), LoadBalancer (cloud LB), ExternalName (DNS alias).
- **Session affinity:** ClientIP or None (default).
- **Health:** Only Ready endpoints receive traffic.

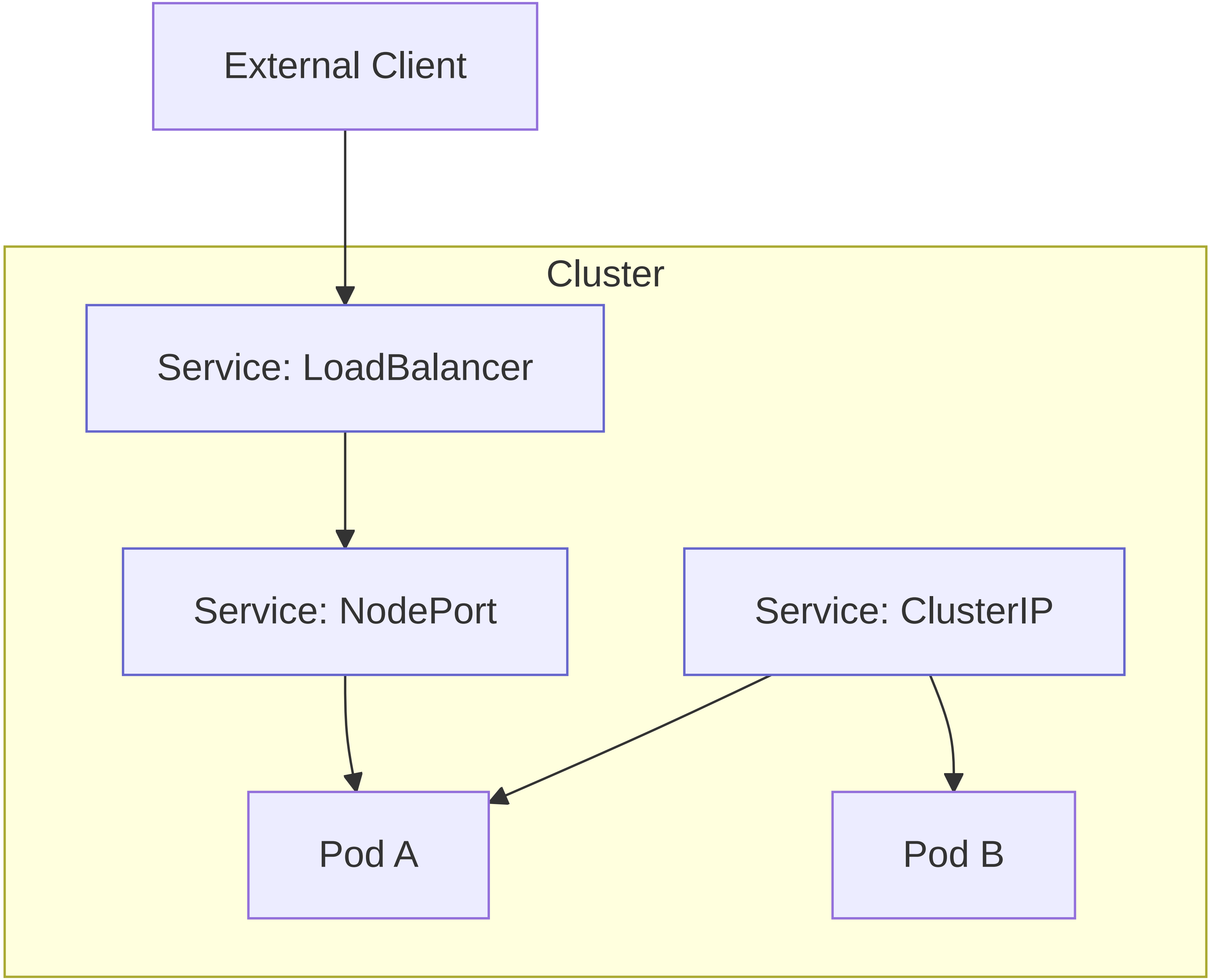

Service selector: app=web
Pod web-1
Pod web-2
Pod web-3 NotReady

> Service = stabiler Zugangspunkt (Name/IP). Endpoints werden aus Labels abgeleitet; nur „Ready“ wird berücksichtigt.

4) ClusterIP, NodePort, LoadBalancer

Data Paths & Use Cases

- **ClusterIP:** Virtual IP only inside cluster; typical for east–west traffic.
- **NodePort:** Exposes a fixed TCP/UDP port on every node; external clients hit any node’s IP:port.
- **LoadBalancer:** Integrates cloud/network LB; LB → NodePort/ClusterIP → Pods.


External Client
Cluster
Service: LoadBalancer
Service: NodePort
Service: ClusterIP
Pod A
Pod B

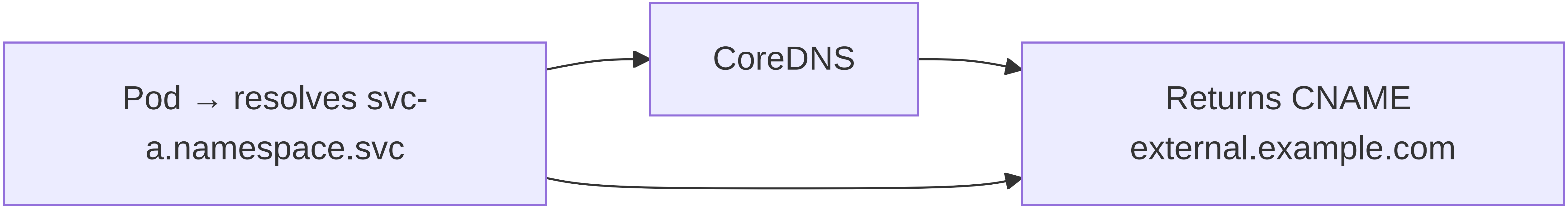
Implementation: kube-proxy programs iptables/IPVS; some CNIs (eBPF) implement service load balancing natively.

> ClusterIP = intern. NodePort öffnet Node-Ports. LoadBalancer integriert externes LB. Datenpfad variiert je nach Proxy/CNI.

5) ExternalName Services

DNS-Level Indirection

- **Purpose:** Map a service name to an external DNS name (CNAME) without creating VIPs or endpoints.
- **Use:** Gradual migration to external/backing services; avoid hardcoding external hostnames in apps.
- **Limits:** Works at DNS layer only (no ports/protocol mapping, no LB health).



Pod → resolves svc-a.namespace.svc

CoreDNS

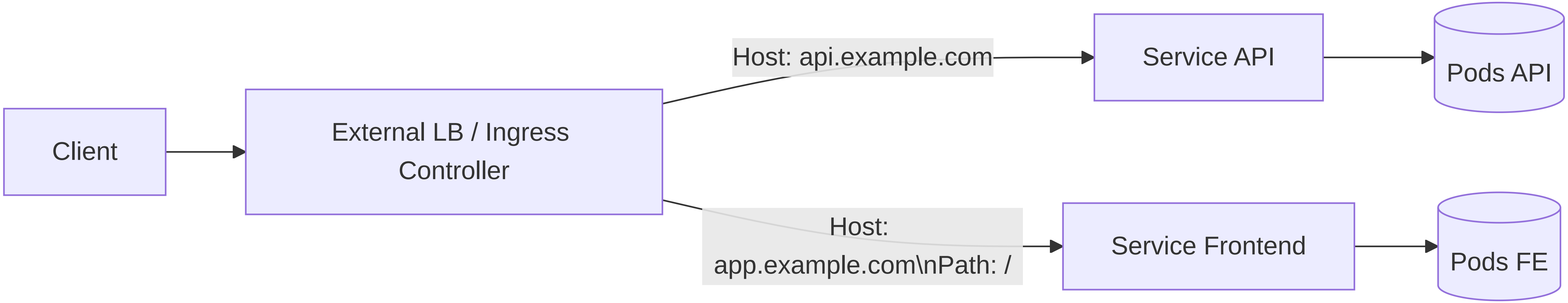
Returns CNAME external.example.com

> ExternalName bietet nur DNS-Alias—keine Lastverteilung oder Health-Checks.
> Ideal für einfache externe Abhängigkeiten.

6) Ingress & Ingress Controllers

North–South HTTP/S Routing

- **Ingress:** API object describing L7 routing (host/path → backend service).
- **Ingress Controller:** Implements the data plane (e.g., NGINX, HAProxy, Envoy, Traefik, cloud L7).
- **TLS:** Termination at edge; certificates via cert-manager; SNI-based routing.
- **Gateway API (evolution):** Decouples listener, route, backend with richer model.


Client
External LB / Ingress Controller
Host: api.example.com
Host: app.example.com\nPath: /
Service API
Service Frontend
Pods API
Pods FE

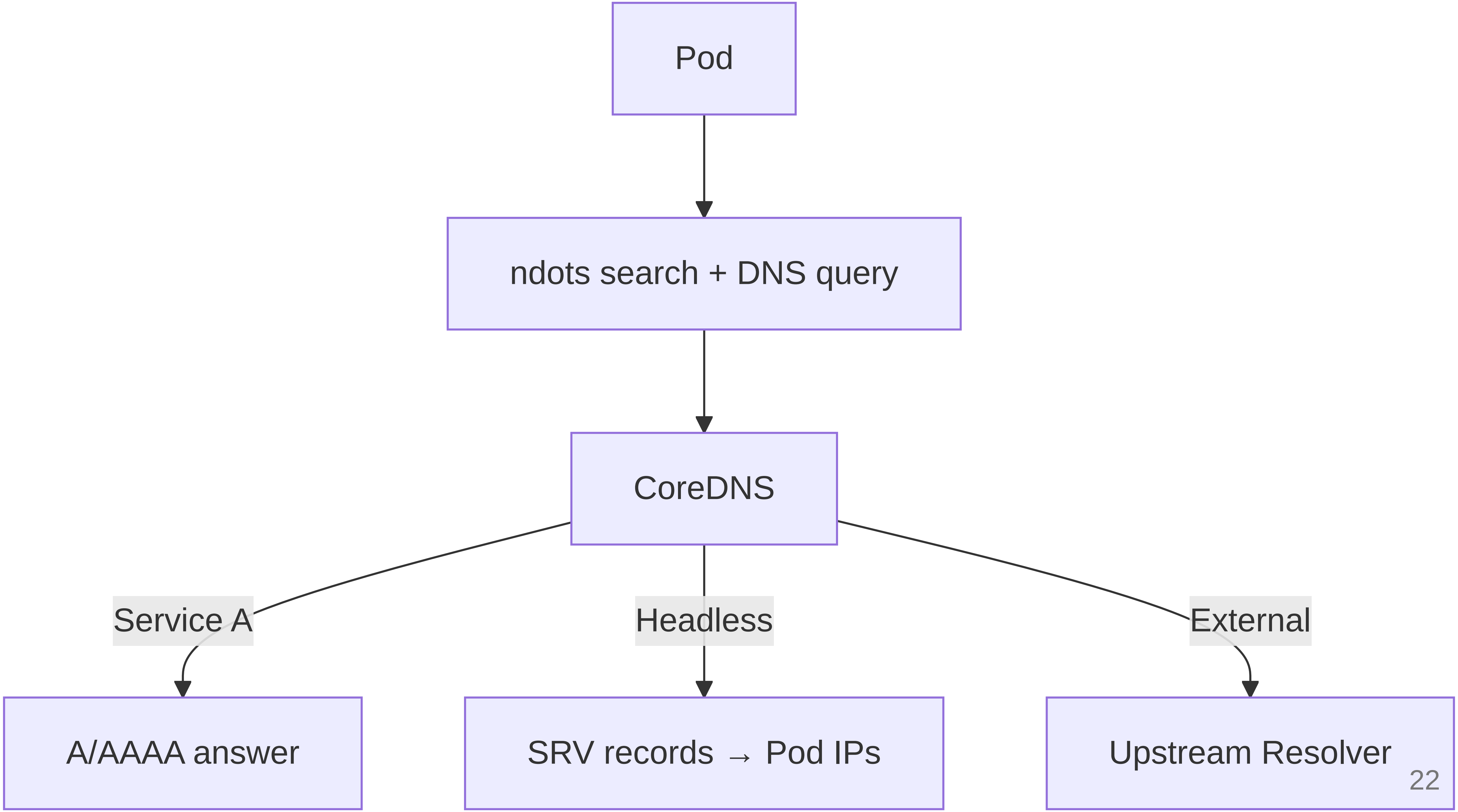
Patterns: Canary by header/path, blue/green routes, WAF/ratelimiting at edge.

> Ingress beschreibt, Controller implementiert. TLS/Host/Path-Routing erlaubt feingranulare Steuerung. Gateway API wird Ingress mittelfristig ergänzen/ersetzen.

7) DNS in Kubernetes (CoreDNS)

Service & Pod Name Resolution

- **CoreDNS** runs as a Deployment/DaemonSet inside the cluster.
- **Search domains:** svc.cluster.local by default; FQDN structure: service.namespace.svc.cluster.local.
- **A/AAAA records:** For ClusterIP; SRV records for headless services map to pod IPs.
- **Stub domains & upstreams:** Forward unknown queries to upstream DNS; can stub cloud/service zones.


Pod
ndots search + DNS query
CoreDNS
Service A
Headless
External
A/AAAA answer
SRV records → Pod IPs
Upstream Resolver

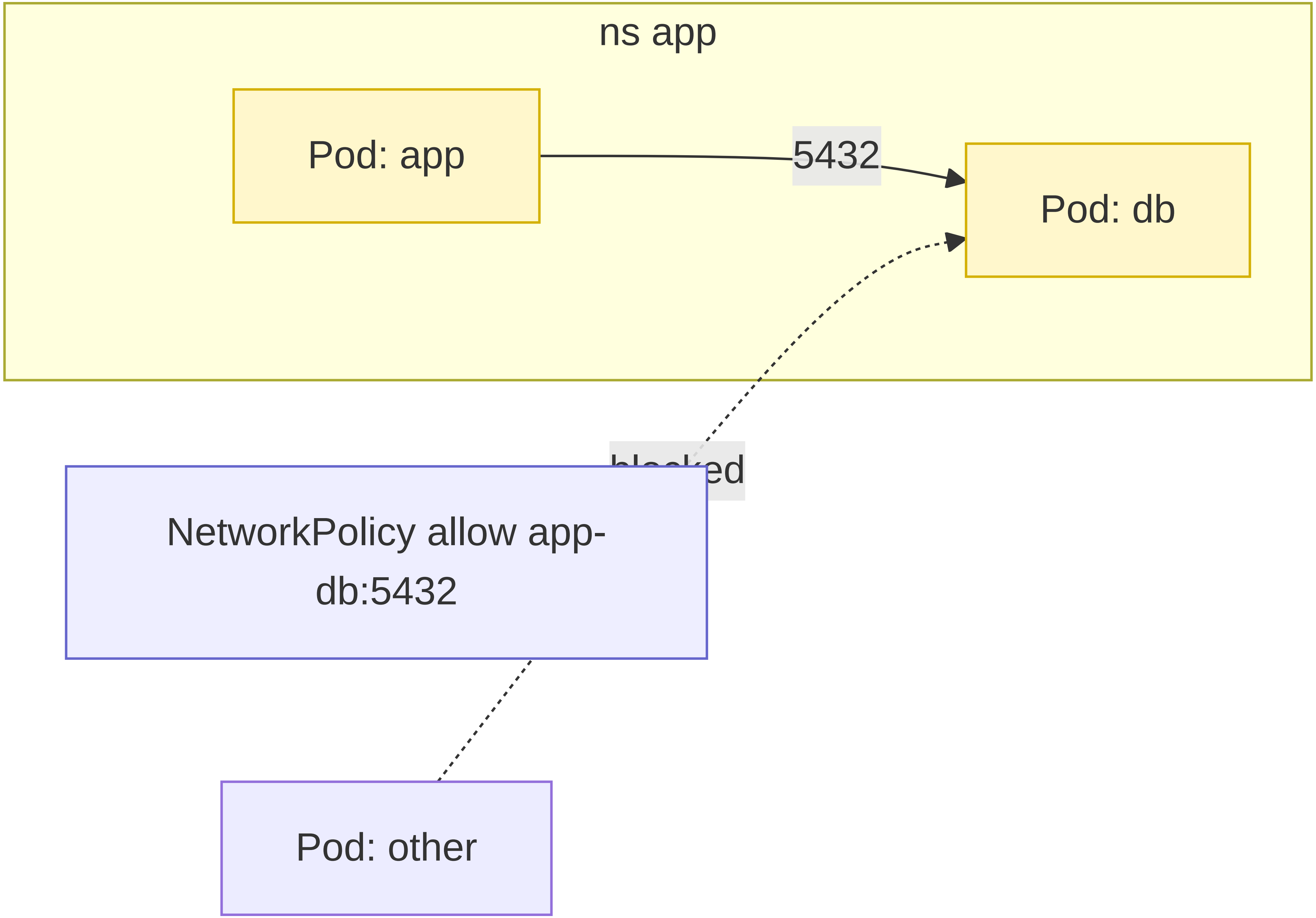
Headless (ClusterIP: None): Direct pod endpoints (stateful discovery).

> CoreDNS liefert Service-/Pod-Auflösung. Headless-Services sind wichtig für StatefulSets (stabile, direkte Endpunkte).

8) Network Policies

Pod-Level Traffic Control

- **Model:** Namespaced, deny-by-default once any NetworkPolicy selects a pod.
- **Selectors:** Pod/namespace selectors + IPBlocks; directions: Ingress and/or Egress.
- **Scope:** L3/L4 (IP/port); some CNIs extend to L7 (e.g., Cilium).
- **Best practice:** Default-deny + allow-only required communications.


ns app
Pod: app
5432
Pod: db
blocked
NetworkPolicy allow app-db:5432
Pod: other

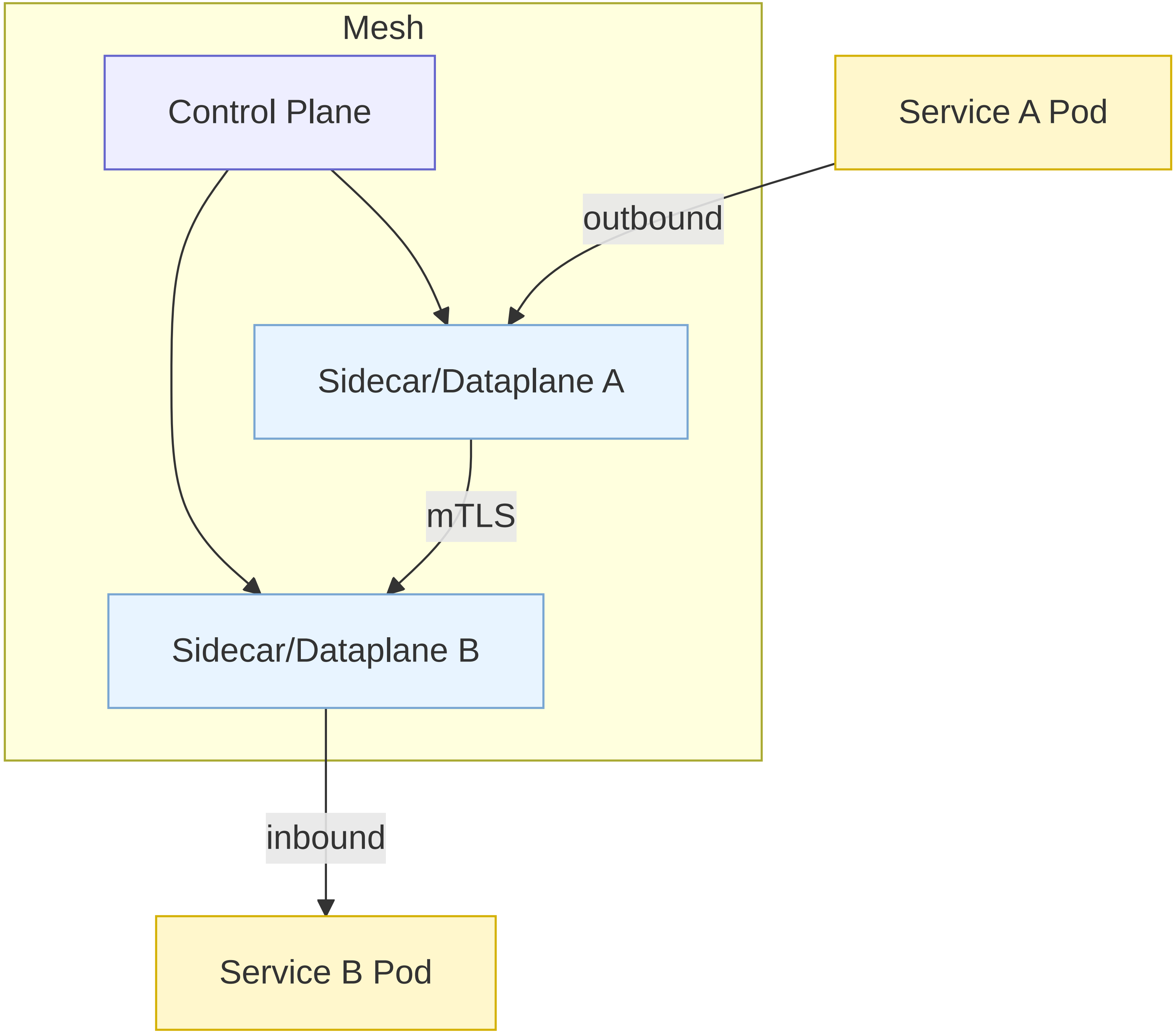
Policy gaps: DNS egress, node metadata access—explicitly allow/deny as needed.

> Sobald eine Policy gilt, werden alle nicht erlaubten Flüsse blockiert (implizites Deny). Policies granular planen und DNS/Egress nicht vergessen.

9) Service Mesh Overview

Istio, Linkerd (Data & Control Planes)

- **Goal:** Uniform mTLS, traffic policy, telemetry, and advanced routing without app changes.
- **Architecture:** Sidecar (or ambient dataplane) + control plane (Istiod/Linkerd control) for config and certs.
- **Capabilities:** mTLS identity (SPIFFE), retries/timeouts, circuit breaking, canary/traffic shifting, metrics/traces.


Mesh
Control Plane
Service A Pod
outbound
Sidecar/Dataplane A
mTLS
Sidecar/Dataplane B
inbound
Service B Pod

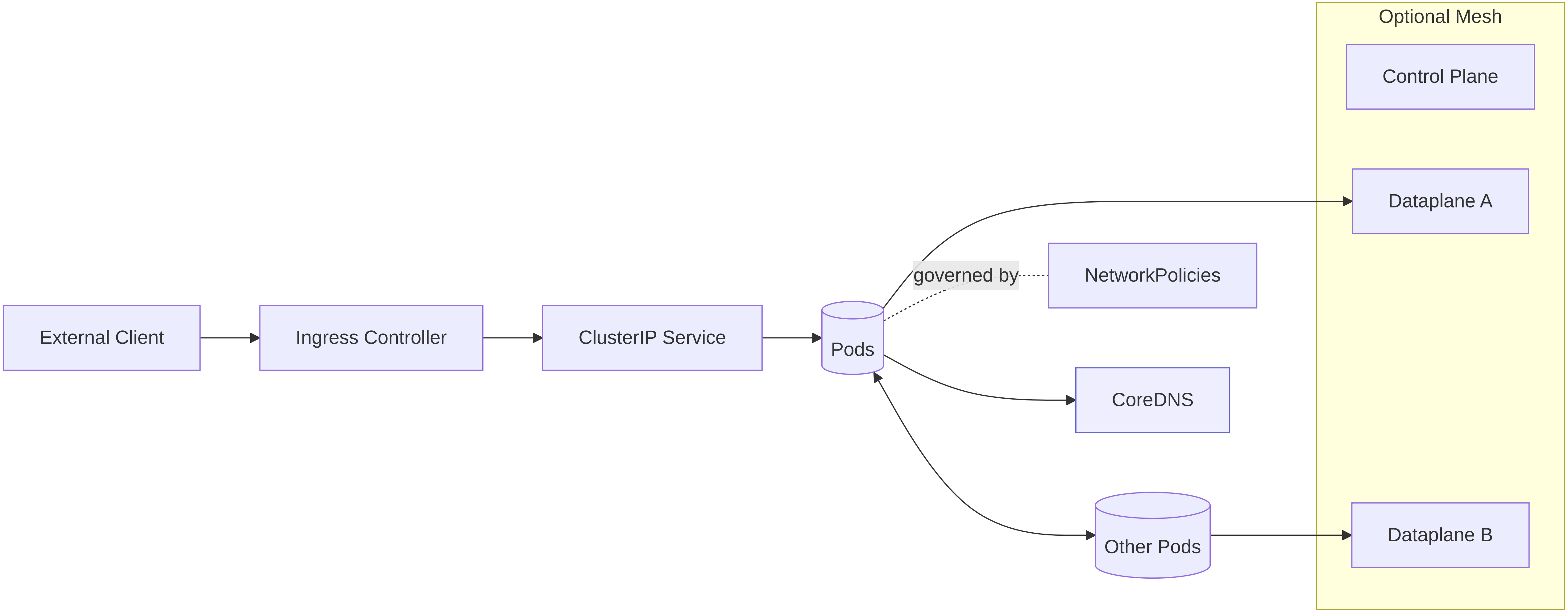
Trade-offs: Operational complexity, resource overhead, policy sprawl.

When to use: Strong identity & encryption needs, fine-grained traffic policy, platform-wide telemetry.

> Service Mesh liefert Identität und L7-Policy als Plattform-Feature. Kosten: Komplexität/Overhead. Auswahl abhängig von Anforderungen (mTLS, Traffic-Steuerung, Telemetrie).

Putting It All Together

End-to-End North–South & East–West


Optional Mesh
Control Plane
Dataplane A
governed by
NetworkPolicies
External Client
Ingress Controller
ClusterIP Service
Pods
CoreDNS
Other Pods
Dataplane B

Flow: Client → Ingress → Service → Pods; pods resolve peers via CoreDNS; NetworkPolicies constrain flows; optional Mesh adds mTLS & routing control.

> Gesamtbild: Services und Ingress definieren Erreichbarkeit, CoreDNS löst Namen, Policies begrenzen Flüsse, Mesh ergänzt Identität/Telemetrie.

Design Takeaways

- Flat IP model with no inter-pod NAT underpins K8s networking.
- Use Services for stable endpoints; choose type by exposure needs.
- Ingress/ Gateway API for HTTP(S) edge routing; terminate TLS at edge.
- CoreDNS is critical plumbing—tune caching/upstreams for reliability.
- NetworkPolicies implement least-privilege connectivity.
- Service Mesh adds mTLS, traffic policy, and rich telemetry—adopt when justified.

> Merksätze: Basismodell verstehen, Service-Typ passend wählen, DNS/Policies korrekt konfigurieren und Mesh gezielt einsetzen.

References & Further Reading

- **Kubernetes Docs:** Services/Networking, Ingress & Gateway API, DNS/CoreDNS
- **CNI Providers:** Calico, Cilium, Flannel; kube-proxy iptables/IPVS
- **NetworkPolicy best practices; EndpointSlice**
- **Service Mesh:** Istio, Linkerd; SPIFFE/SPIRE for identity

> Primärquellen für Vertiefung. Besonders wichtig: CNI-Fähigkeiten prüfen, Gateway API-Entwicklungen verfolgen, Mesh-Overhead gegen Nutzen abwägen.