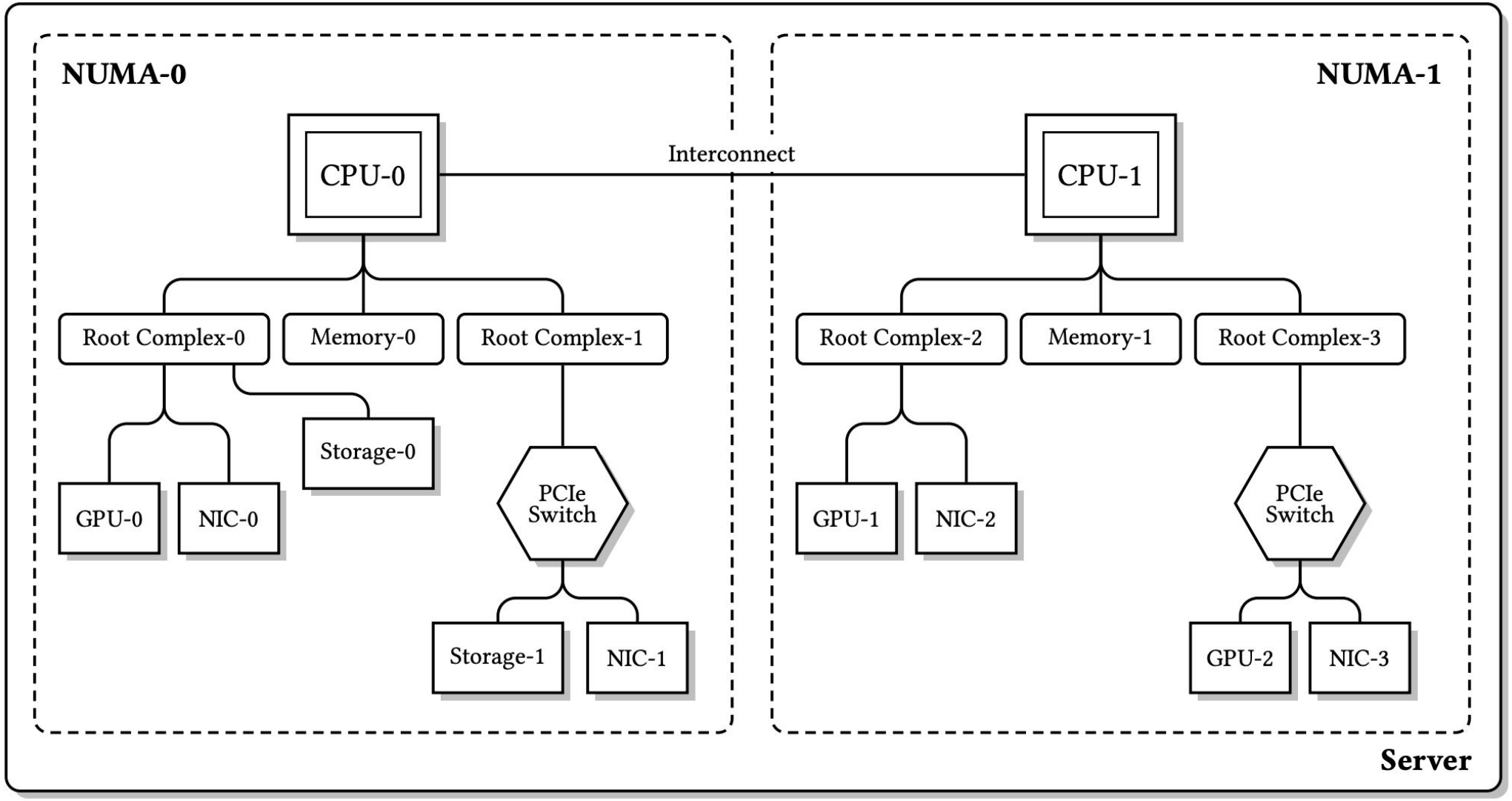
Here be DRAgons: Bringing Native Zero-Copy Networking to Pods with Cilium and netkit

Daniel Borkmann, Aaron Campbell, Bernardo Soares

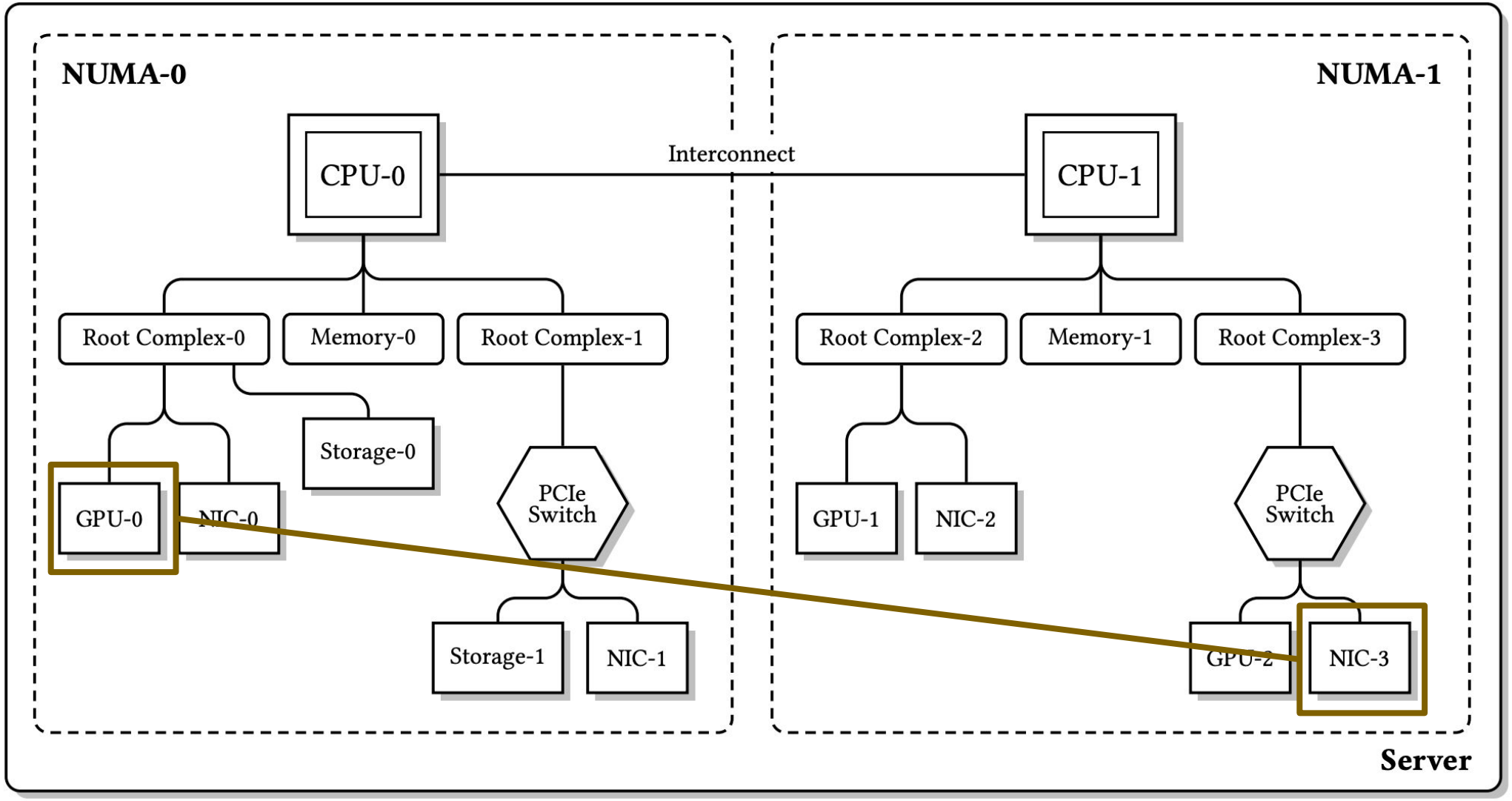
KubeCon North America 2026

ISOVALENT
now part of CISCO

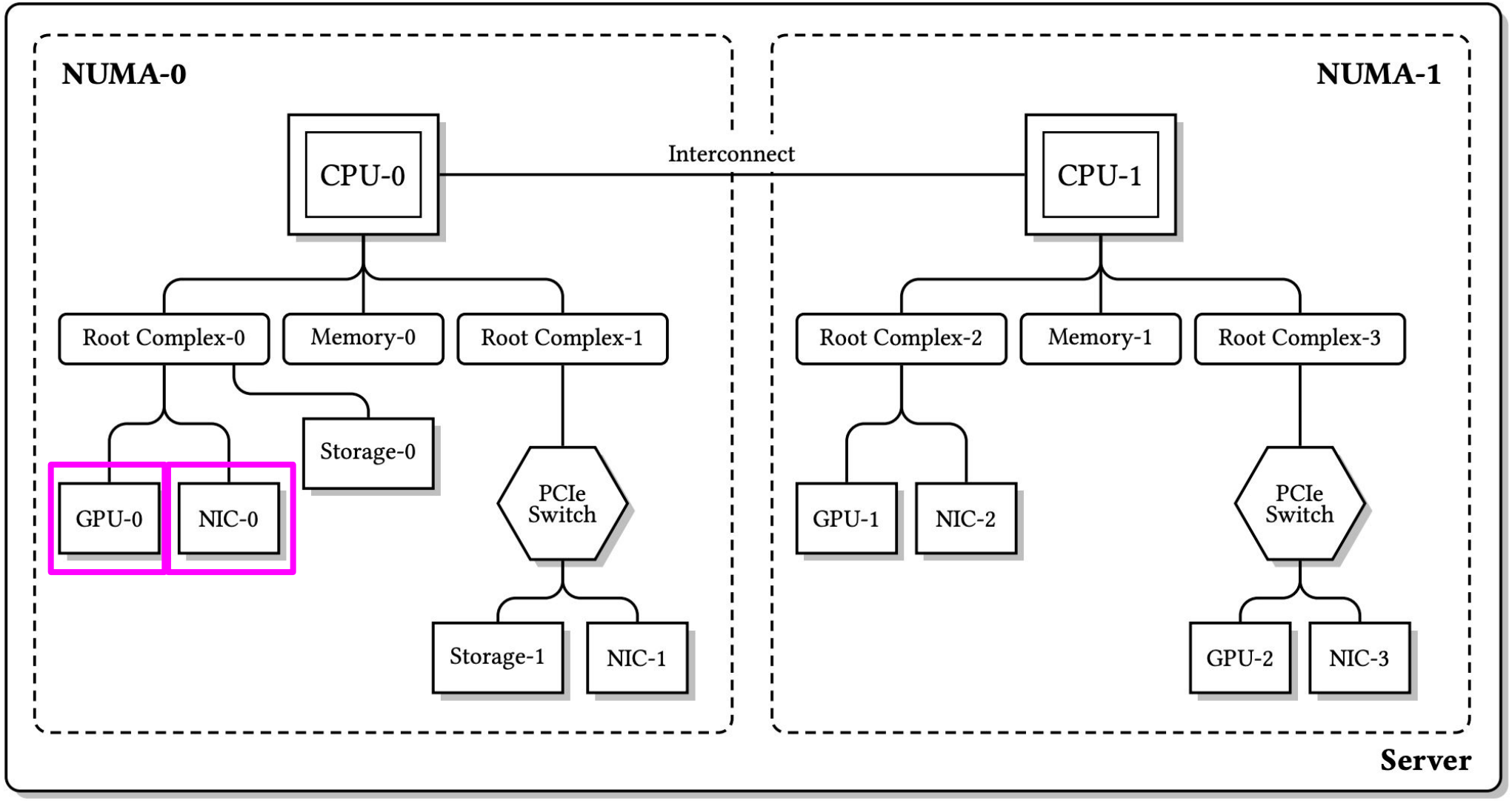




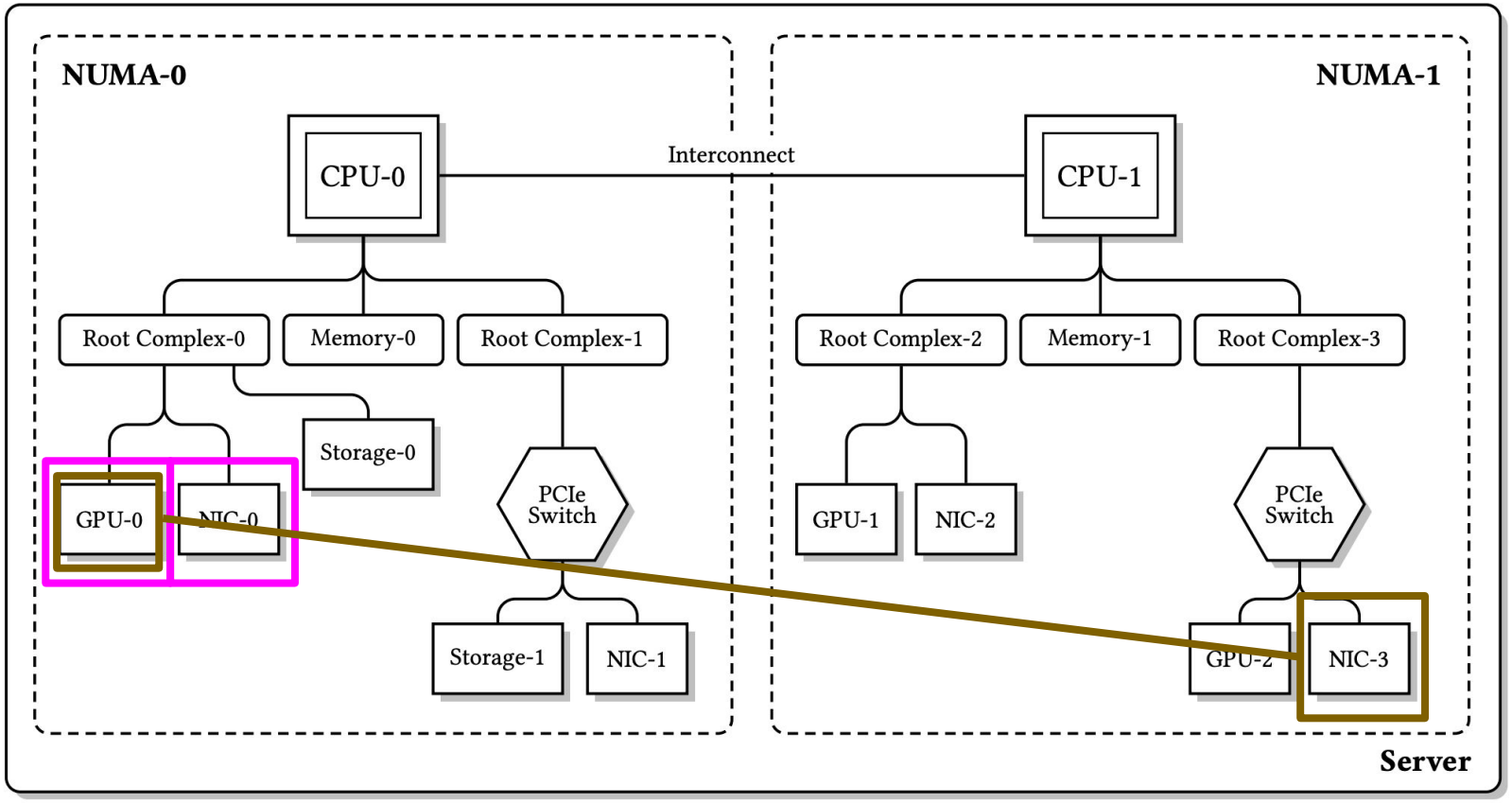
Scenario: YOUR expensive server node



Pod workload A



Pod workload B ... which HW placement is more efficient?



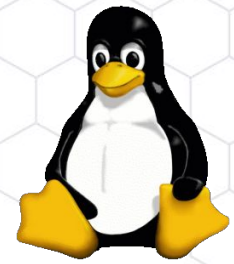
source: DRAmatically simple

- Misaligned GPUs/NICs can cause 10-60% performance drops in AI workloads

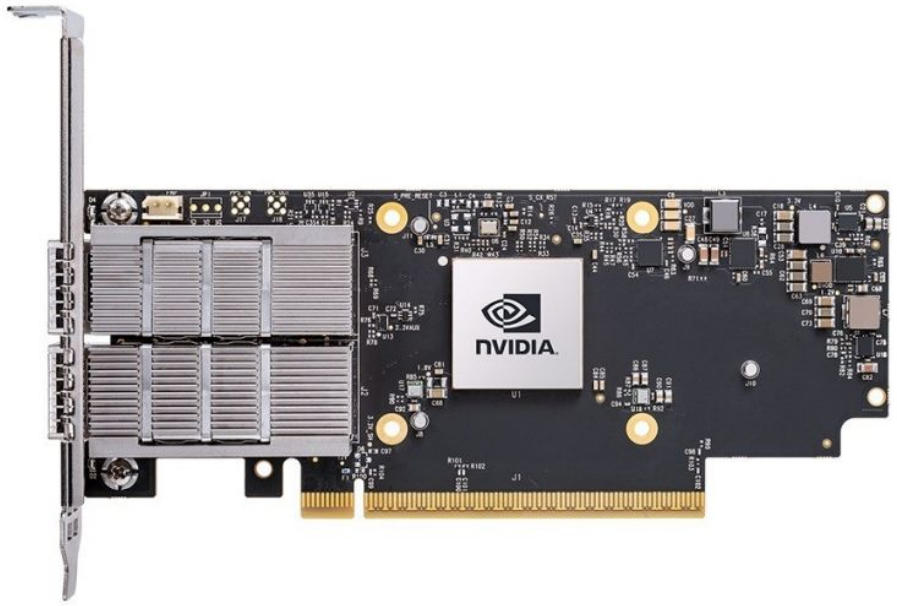
- Missing out on significant performance gains when HW offload is not leveraged
  → **almost 3x in this talk's topic**

The hard question is how to give everyone isolated access without stepping on each other ...

GPUs and Networking go hand in hand

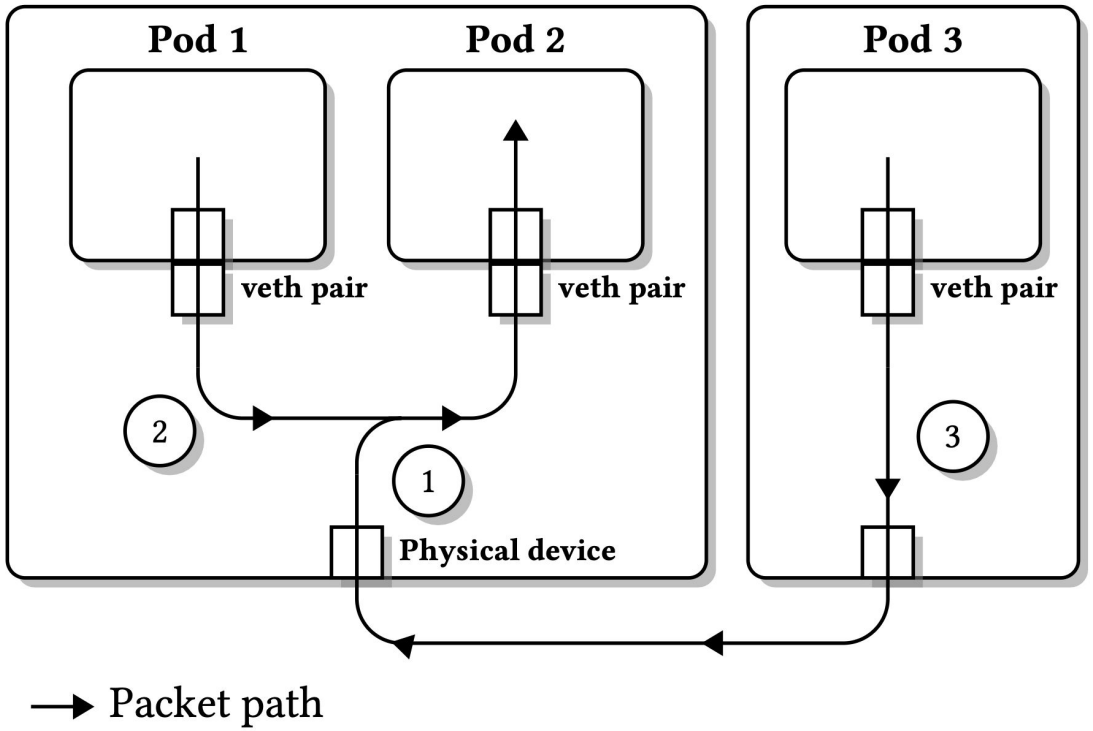


- Speeds of 800+Gb/s per port
- Complex offloads (e.g. crypto)
- Header/data split for native zero-copy via Linux kernel memory provider
- SRIOV with eSwitch functionality
- DPU with ARM cores



Meanwhile: Kubernetes Pod Networking Model

- Basic N/S and E/W connectivity for {node,Pod} <-> {node,Pod}
- K8s service model for load-balancing
  - ClusterIP
  - NodePort
  - LoadBalancer
- Gateway API
- Linux kernel: netns, veth, (iptables or BPF)



Meanwhile: Kubernetes Pod Networking Model



- No awareness of underlying HW in the Pod spec, both good and bad
- Multus CNI to obtain multiple netdevs inside Pods (SRIOV, macvlan, ipvlan, ...)
- Still further building blocks needed on top
- Operational complexity given multiple CNI components (e.g. up/downgrades)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
    - name: nginx-container
      image: nginx:stable-alpine
      ports:
        - containerPort: 80
```

Rescue: Kubernetes Dynamic Resource Allocation (DRA)



- DRA is a k8s feature for generically requesting and sharing resources among Pods
- They can be attached devices like hardware accelerators/GPUs, NICs, etc
- Replaces the need for Multus CNI, and in contrast to Multus has topology-awareness*
- For networking DRAnet is a reference implementation, primarily driven by RDMA / AI-ML use cases

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
spec:
  resourceClaims:
    - name: net
      resourceClaimTemplateName: sriov-claim
  containers:
    - name: app
      image: my-dpdk-app:latest
      resources:
        claims:
          - name: net
```

* e.g. matching device attributes in the K8s scheduler like placing GPU and NIC under the same PCIe root

Rescue: Kubernetes Dynamic Resource Allocation (DRA)



- **Cilium v1.21** ships with native DRA support
- Focus on DRA driver integration is to leverage NIC offloading and specialized networking functionality
- Integration with feature-rich building blocks from Cilium such as IPAM, BGP, DNS, etc
- Reduces operational complexity given all under a single CNI platform

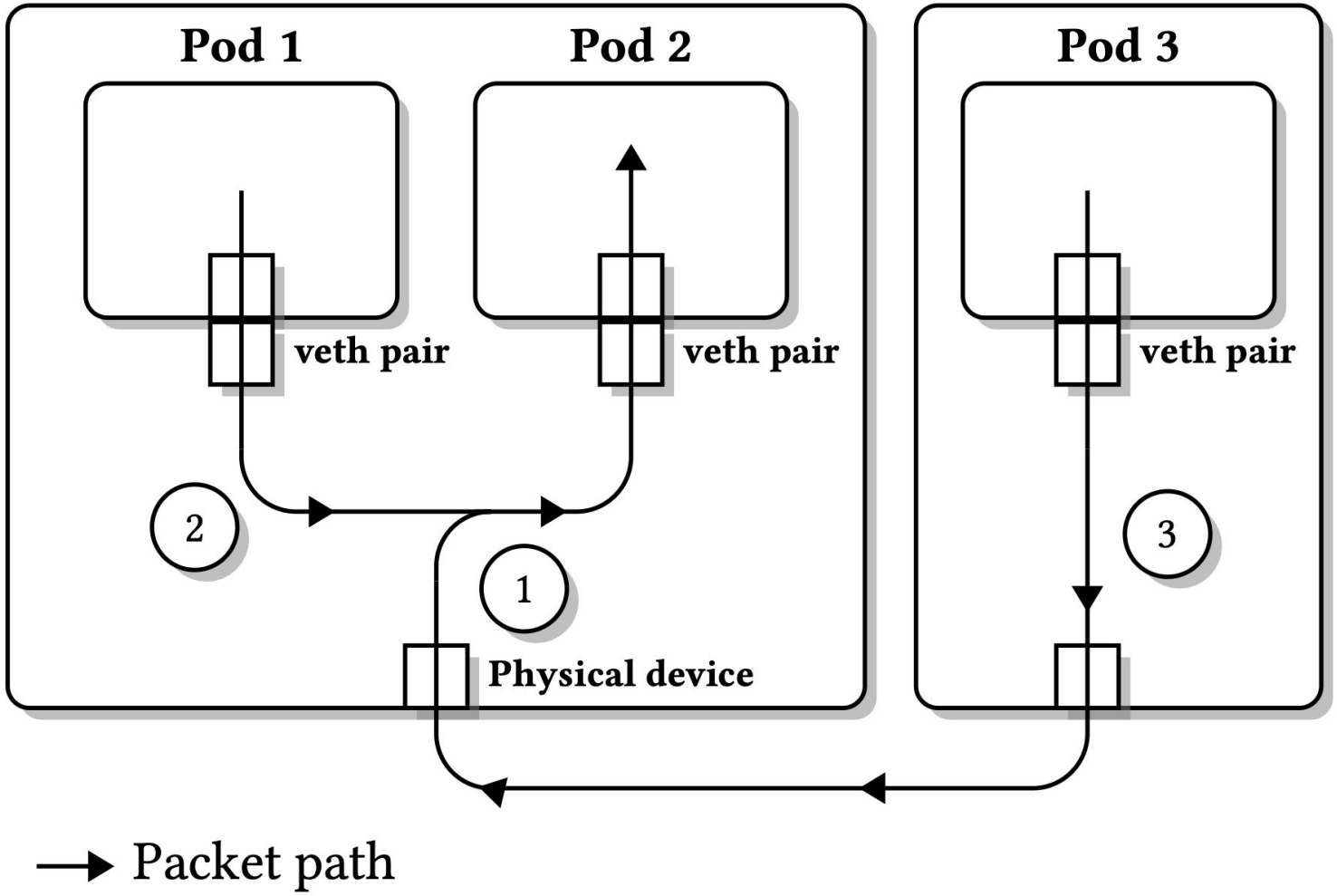
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
spec:
  resourceClaims:
    - name: net
      resourceClaimTemplateName: sriov-claim
  containers:
    - name: app
      image: my-dpdk-app:latest
      resources:
        claims:
          - name: net
```



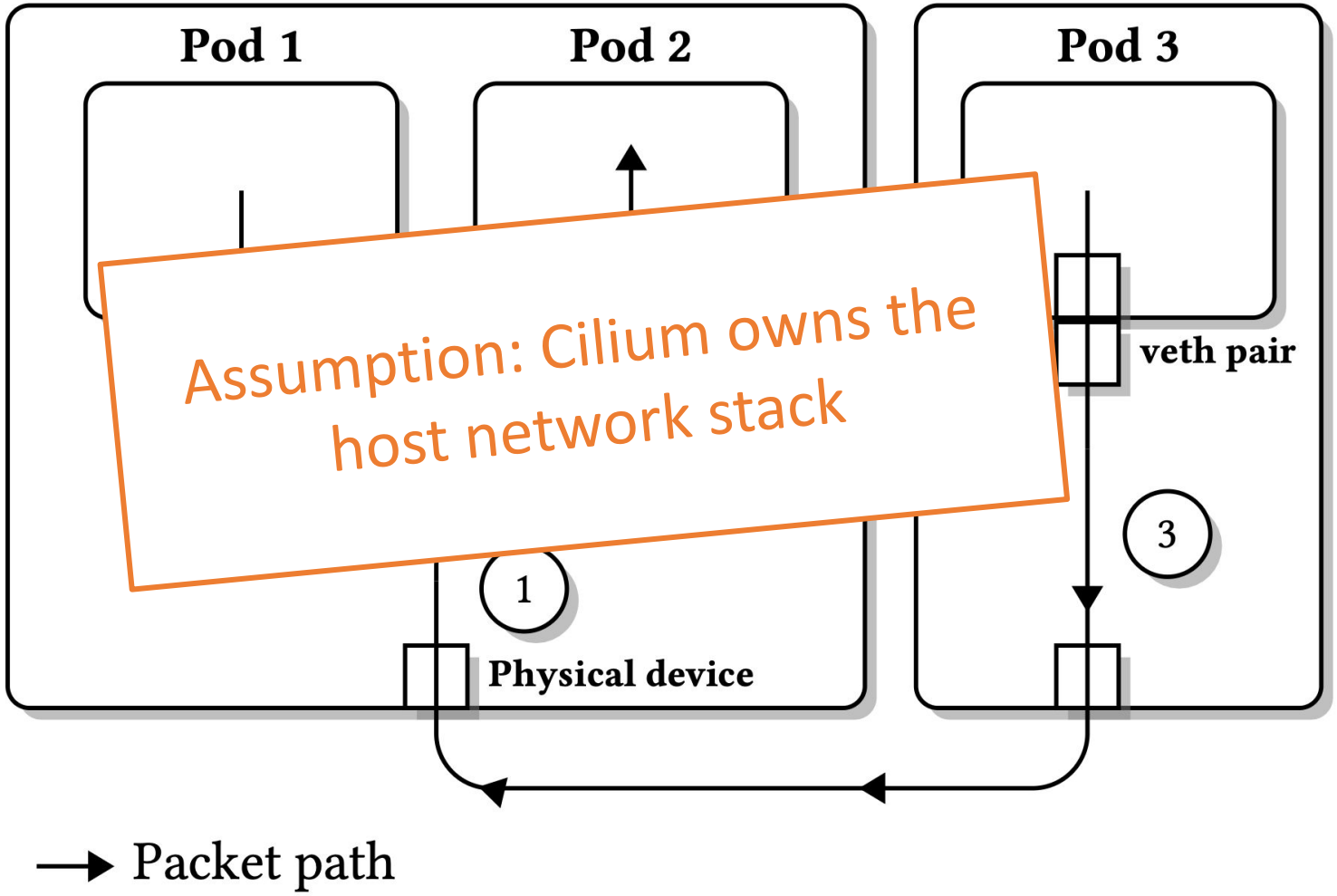
Turning Pod networking performance to 11:

The road from veth → netkit → netkit with DRA for queue leasing ...

Basic Container Networking



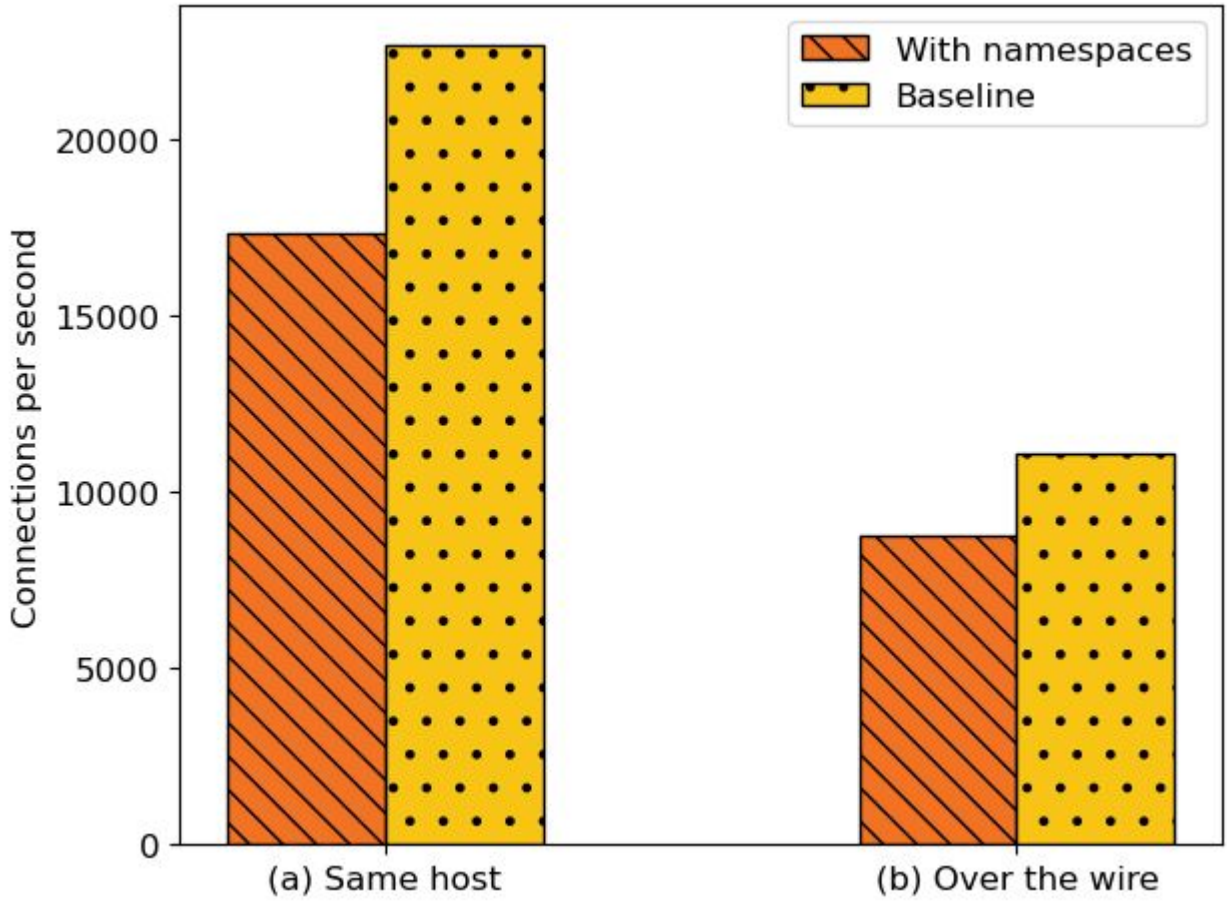
Basic Container Networking



Motivation: Gap to Baseline

- netperf TCP_CRR
- Over loopback and wire
- With and without namespaces

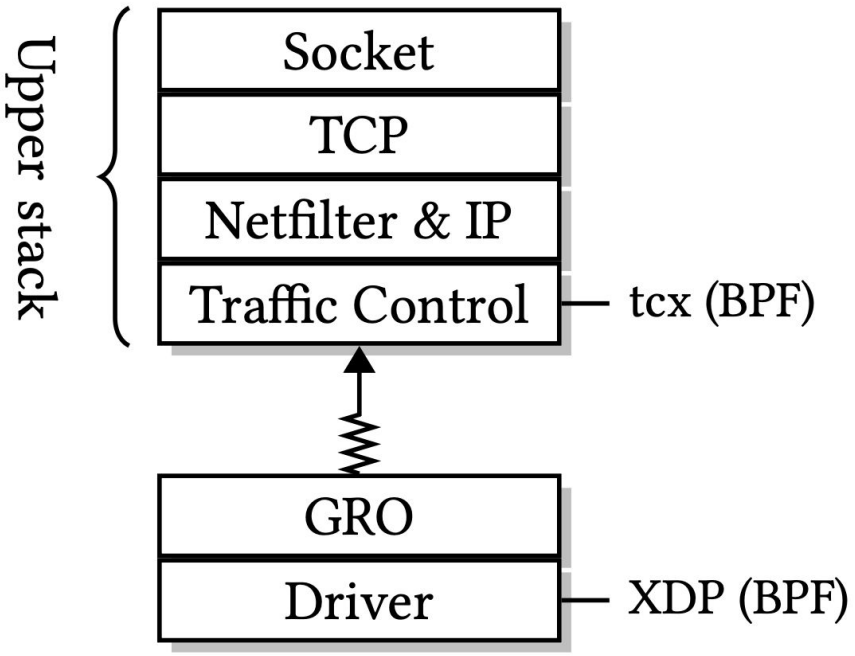
Unexpected. Network namespaces are logical rather than physical boundaries.



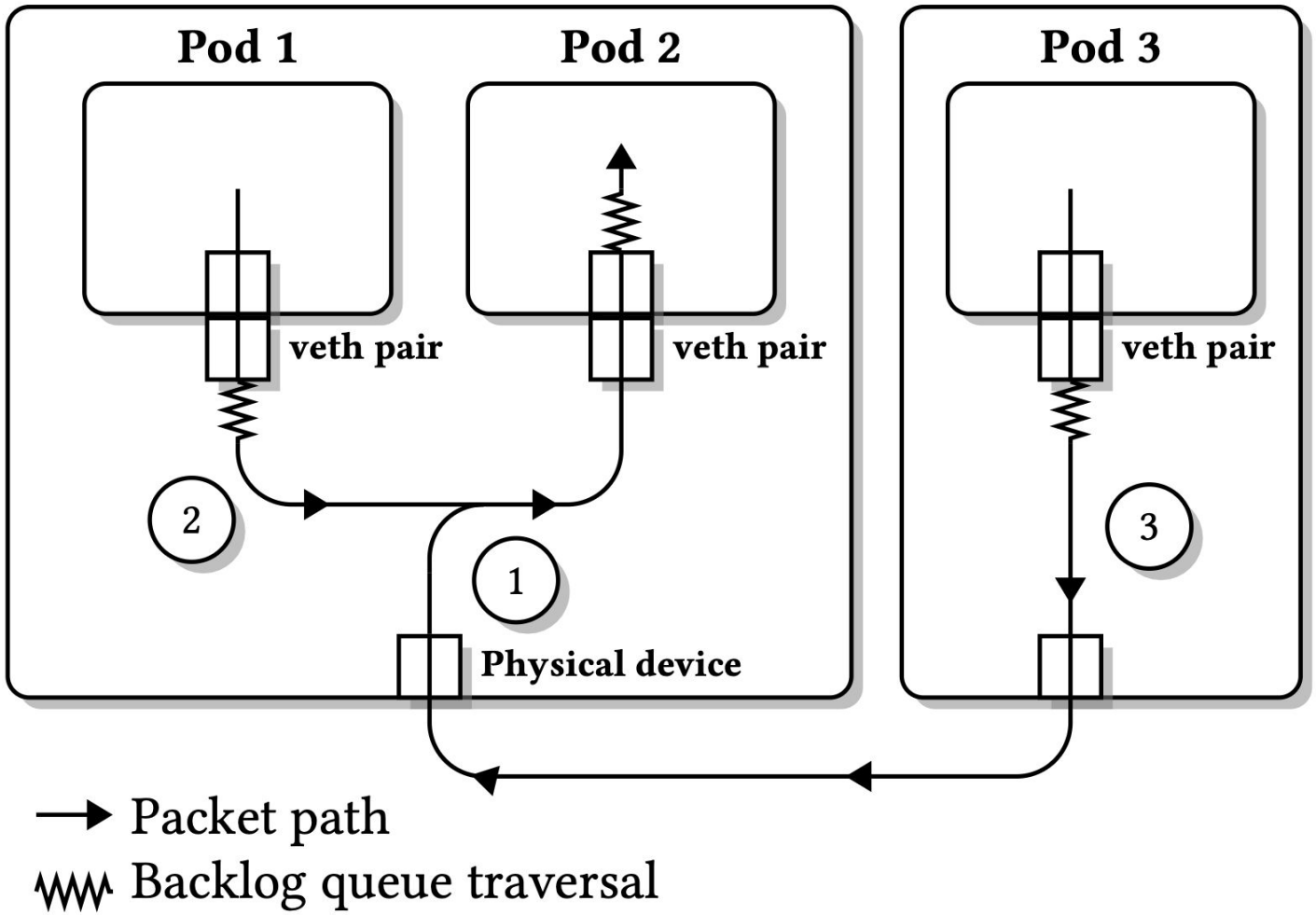
Background: Backlog Queue



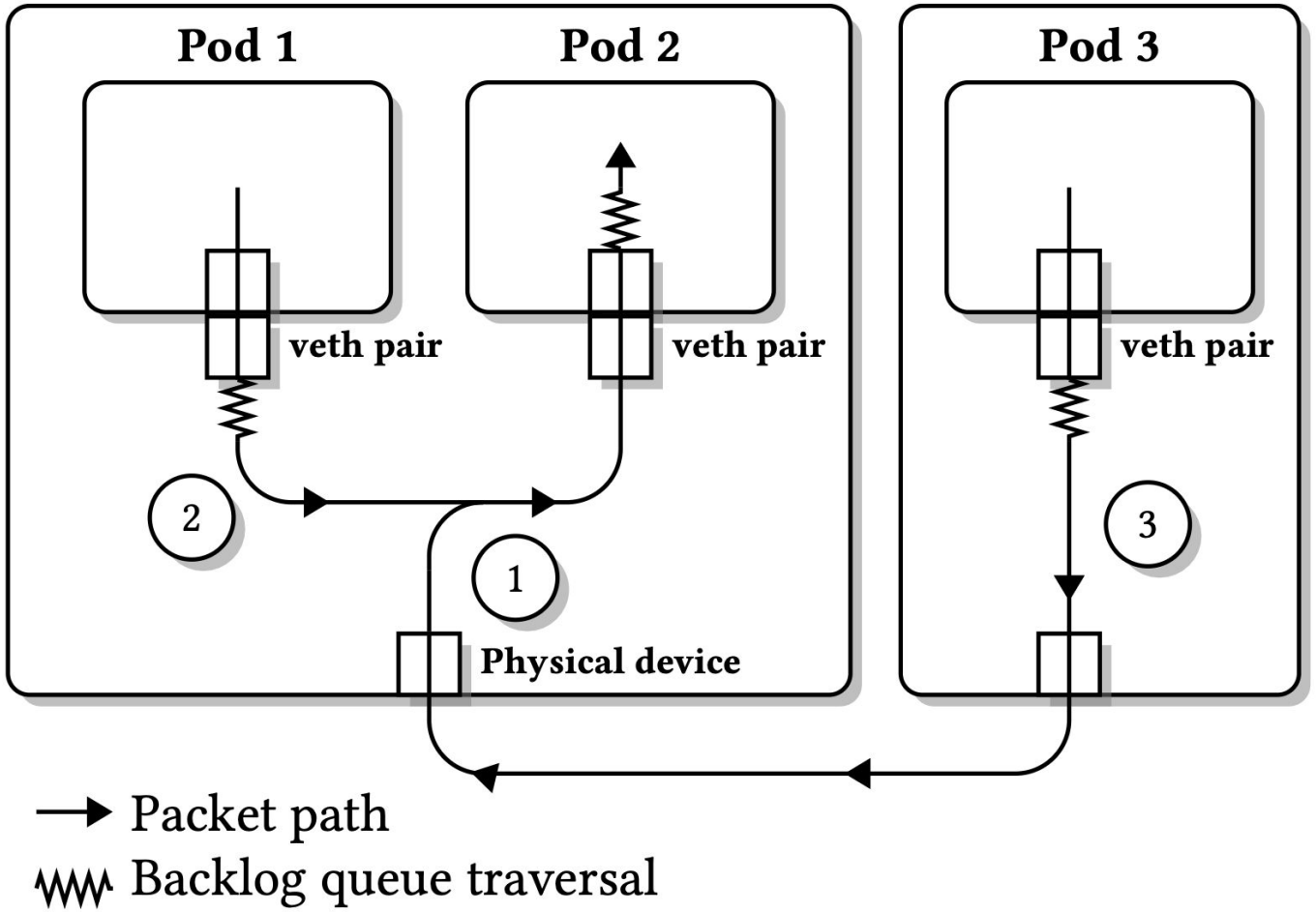
- Per-CPU backlog queue between network driver and upper stack
- Backlog queues used to scale across cores (e.g. RPS); not enabled by default
- Always used for Virtual Ethernet devices (veth) - not always desirable!

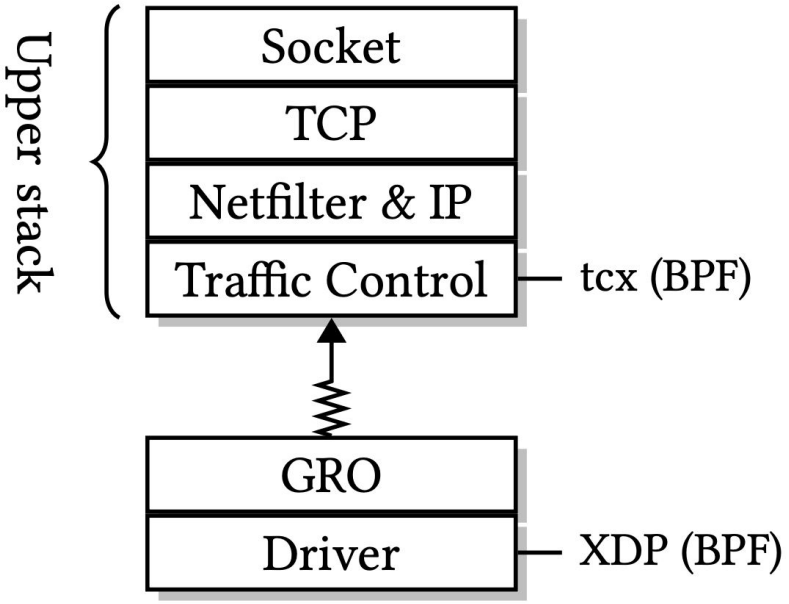


Background: Backlog Queue

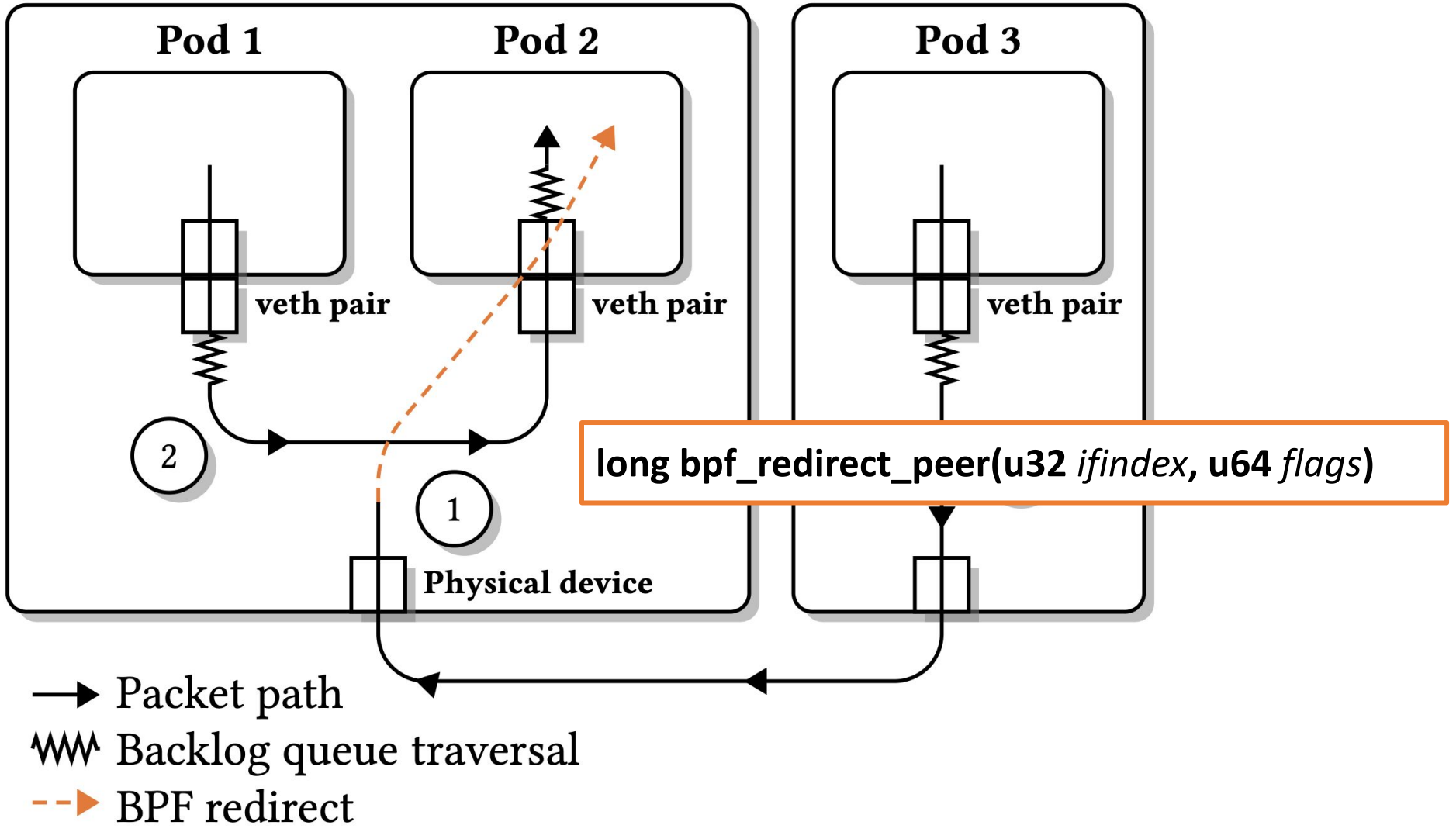


netkit: Eliminating Backlog Queue Traversals

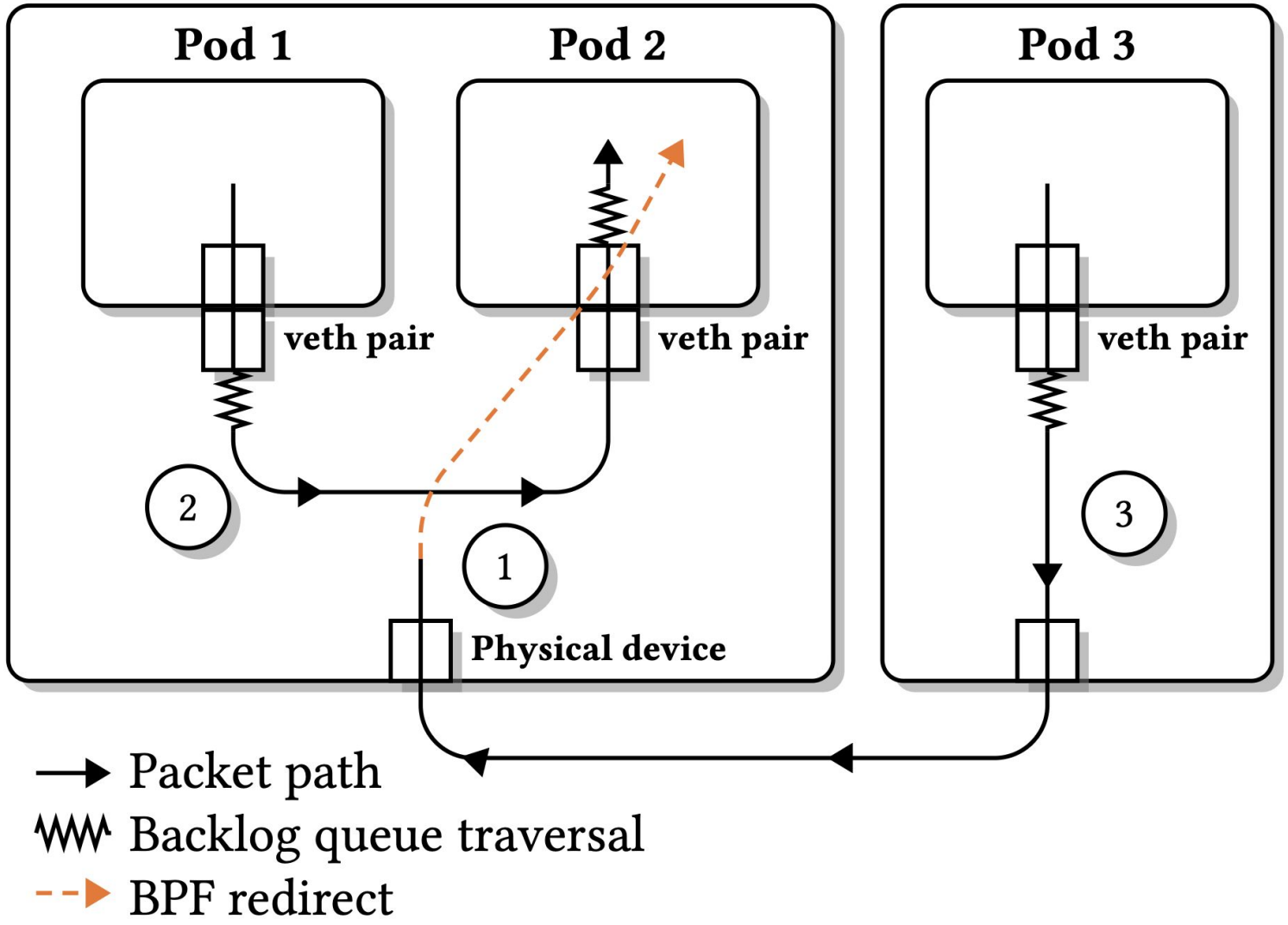




Namespace-aware BPF redirect



The netkit Device Pairs



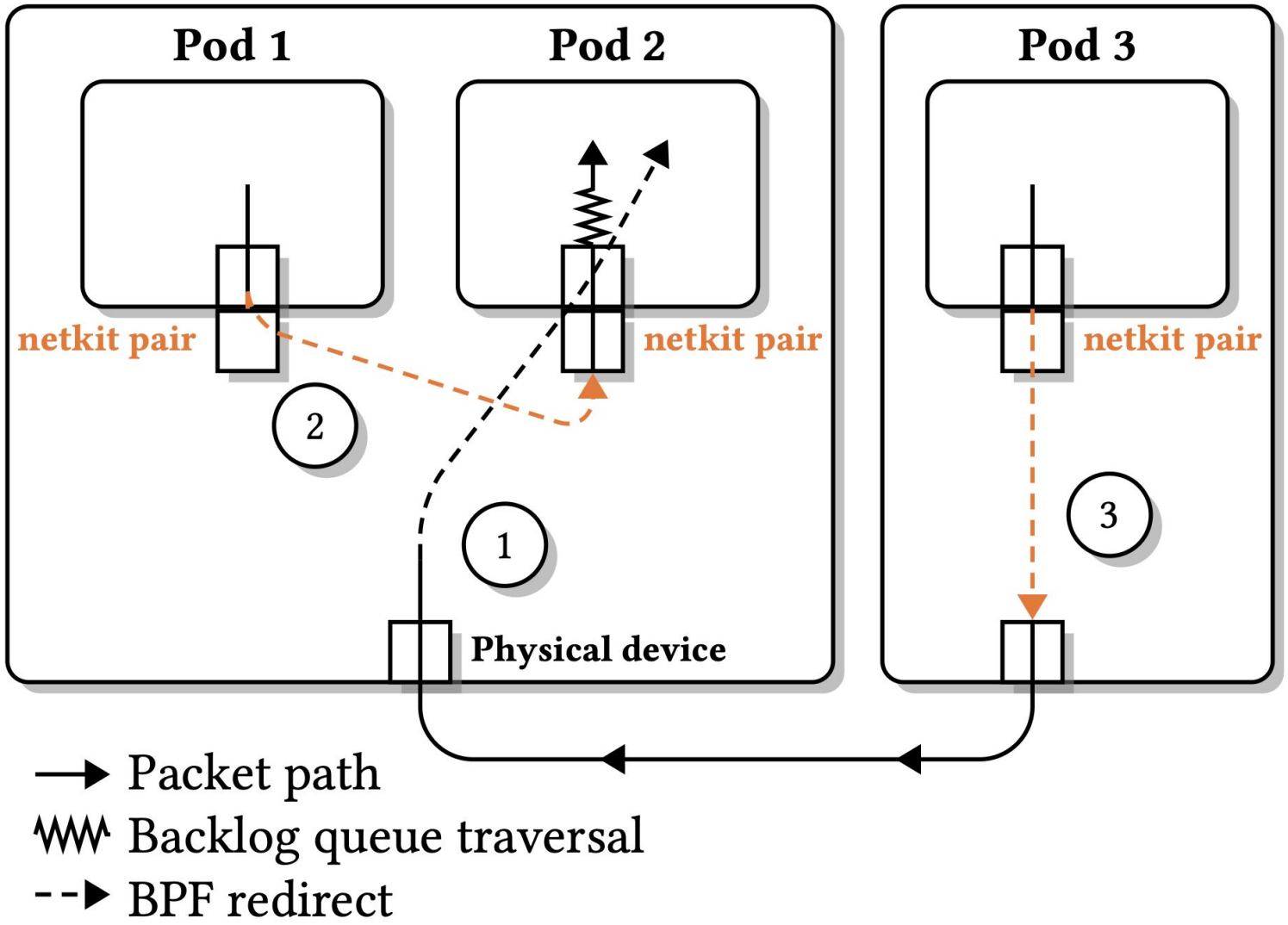
The netkit Device Pairs



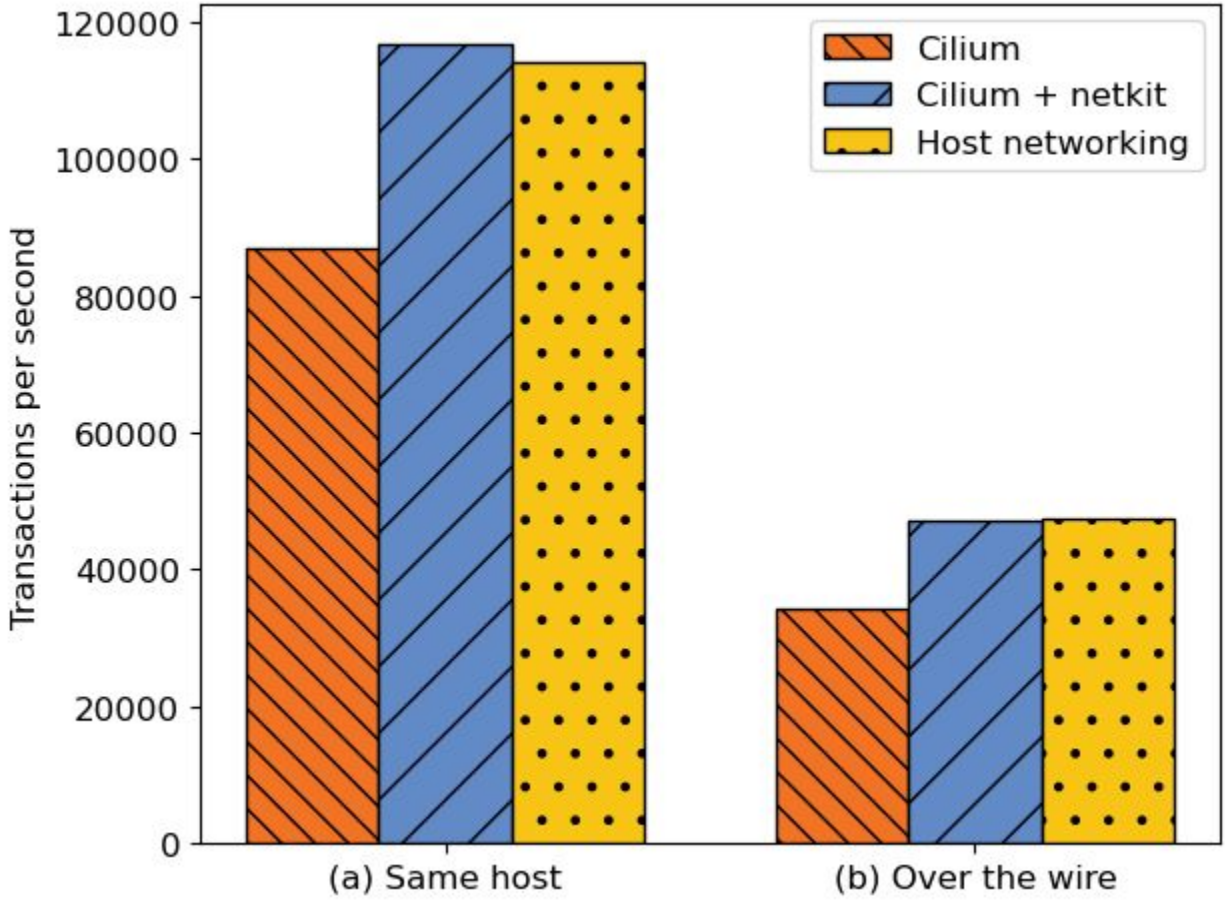
- “Pod-aware” device pair: primary device in host, peer device in network namespace
- Attached BPF programs executed on TX path, immediately after namespace switch
- BPF programs for both sides attached through the primary device

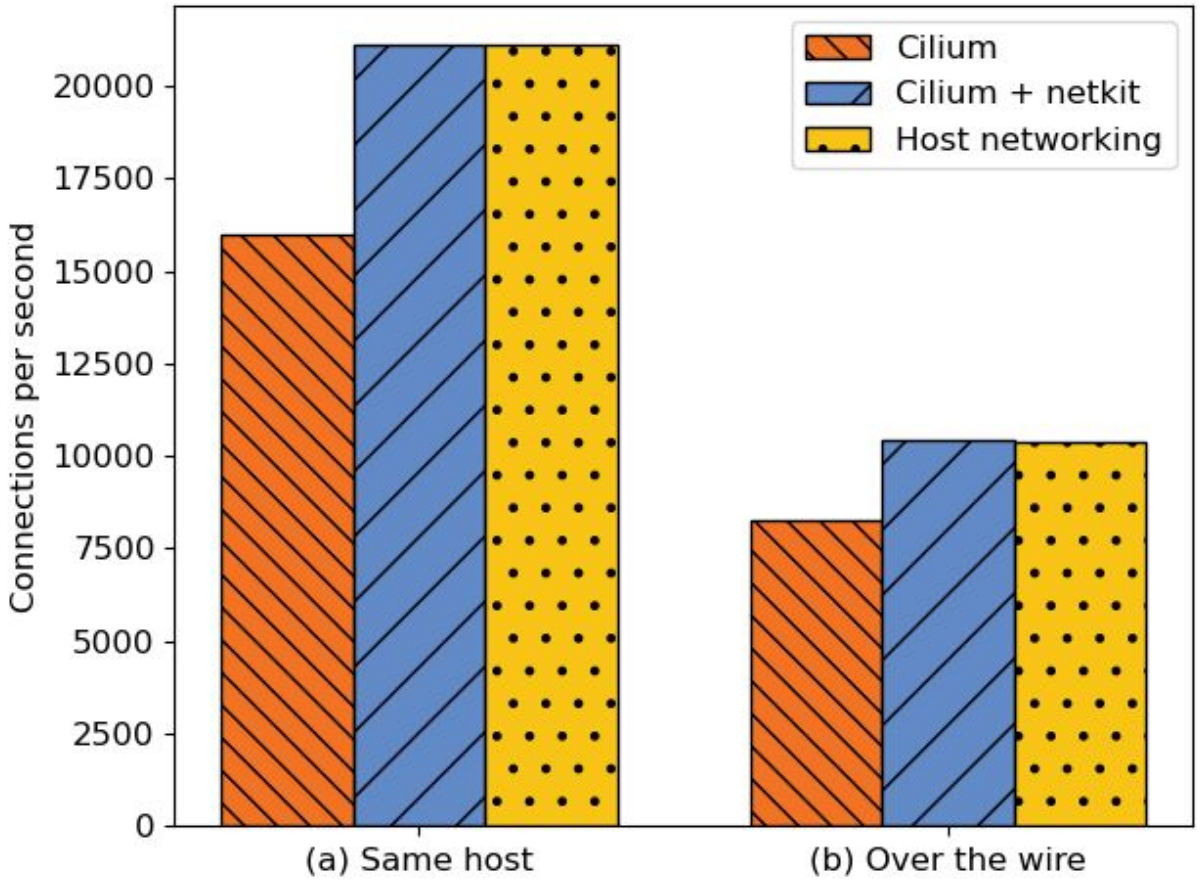
Result: 1) Applications can’t detach peer BPF programs
2) Peer BPF programs can access host information
3) netkit blackholes traffic when no BPF attached

The netkit Device Pairs

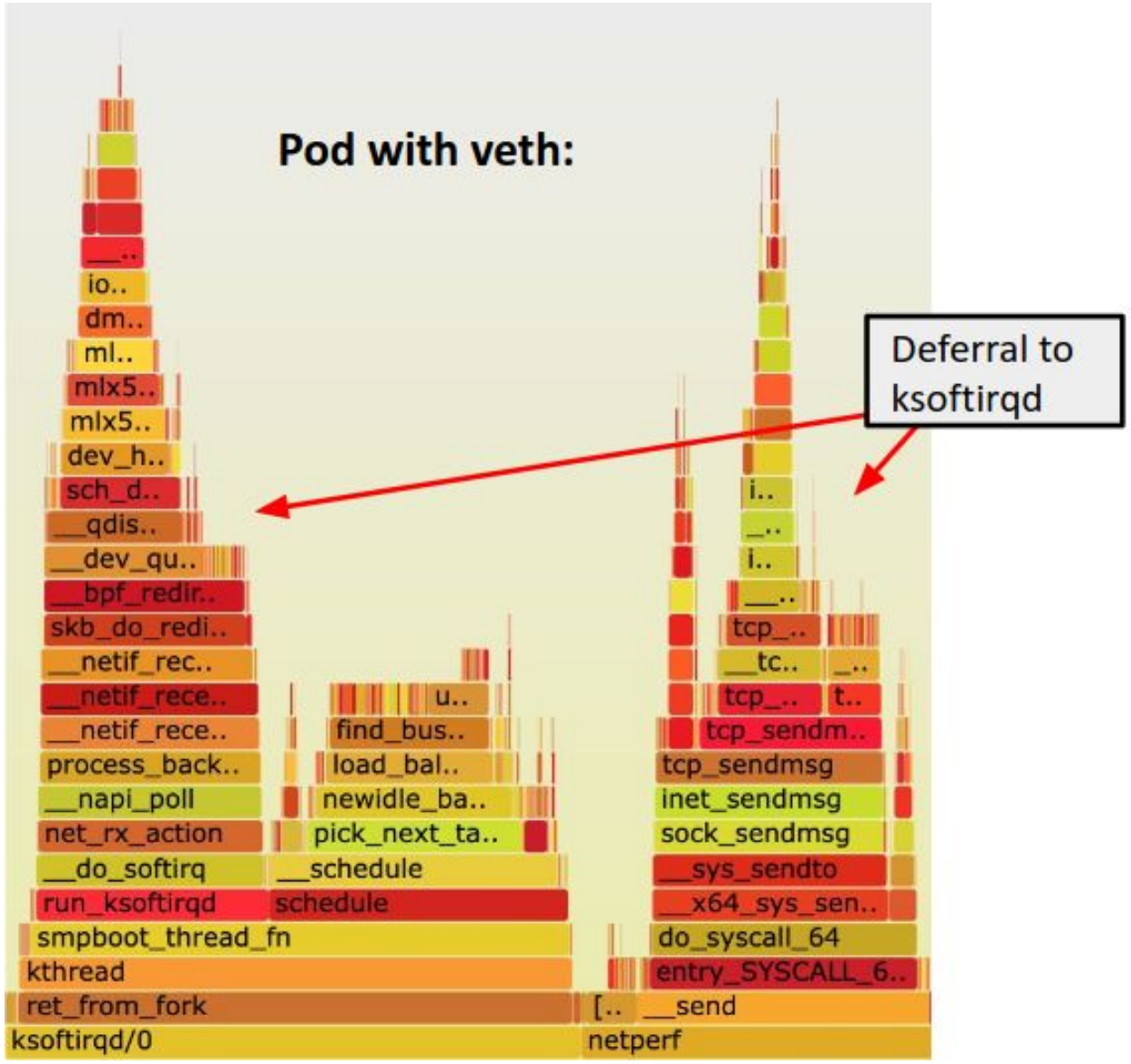


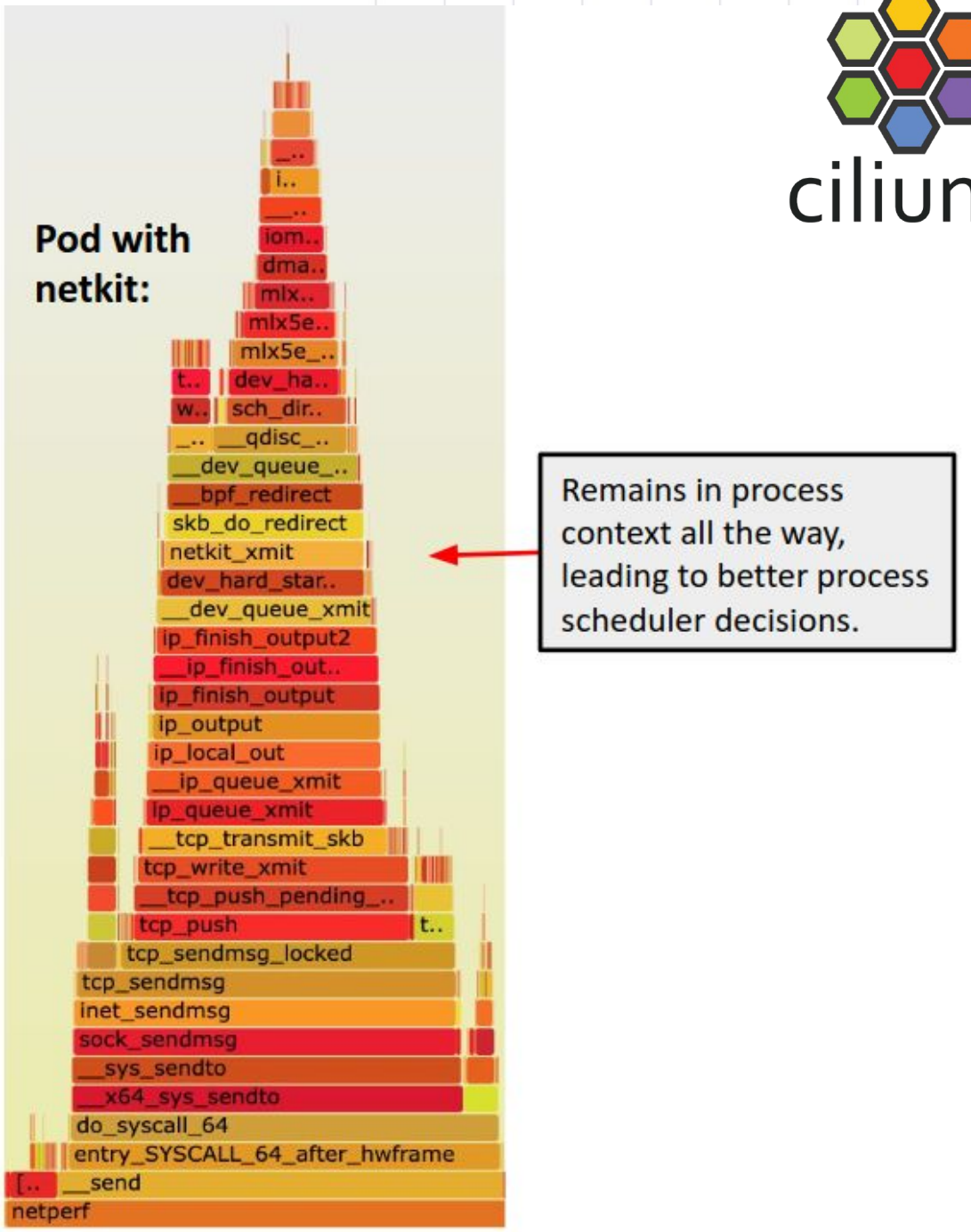
Evaluations: TCP_RR and TCP_CRR





Evaluations: CPU Accountability



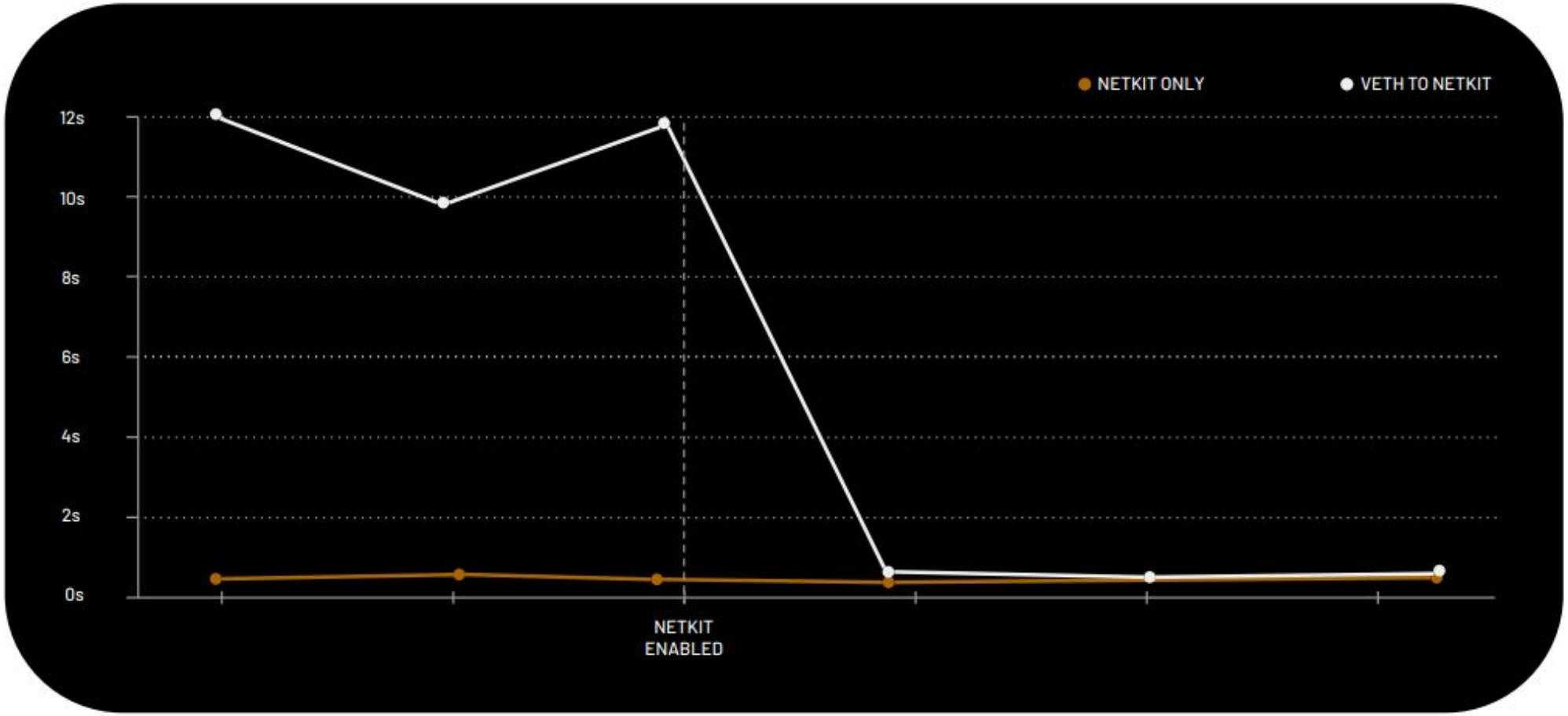


Evaluations: CPU Accountability



- **veth normal case:** Backlog queue drains inline right after softirq raise, borrowing sender's context
- **veth pressure case:** ksoftirqd is a per-CPU, CPU-bound SCHED_NORMAL kthread which competes with applications and cannot migrate CPU. Time-spent accounting against ksoftirqd thread.
- Past netdev_max_backlog (1000) packets are dropped, and TCP retransmits make the tail latency situation even worse
- **netkit:** BPF runs inside netkit_xmit and bpf_redirect_neigh goes straight to the phys NIC's xmit. No backlog enqueue, no ksoftirqd handoff, and time-spent accounted against application.

Evaluations: CPU Accountability



FOSDEM'25 talk from Mike Willard (Meta) with 4 copies of a task stacked on one host:

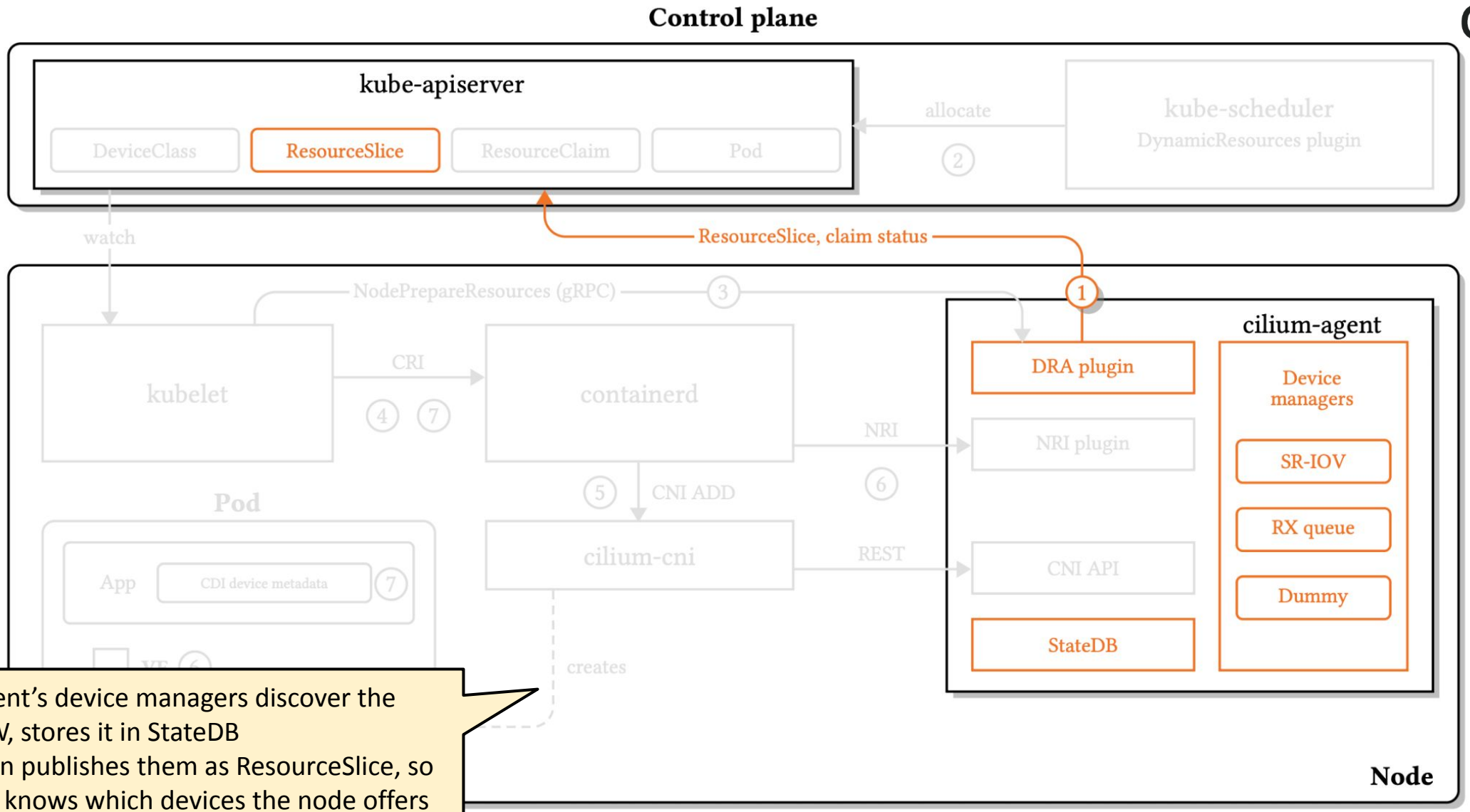
P99 reduction from **>12s to 0.1s**. netkit now part of Meta's container platform (Twine)

Performance++: Offload enablement for Pods via Cilium DRA



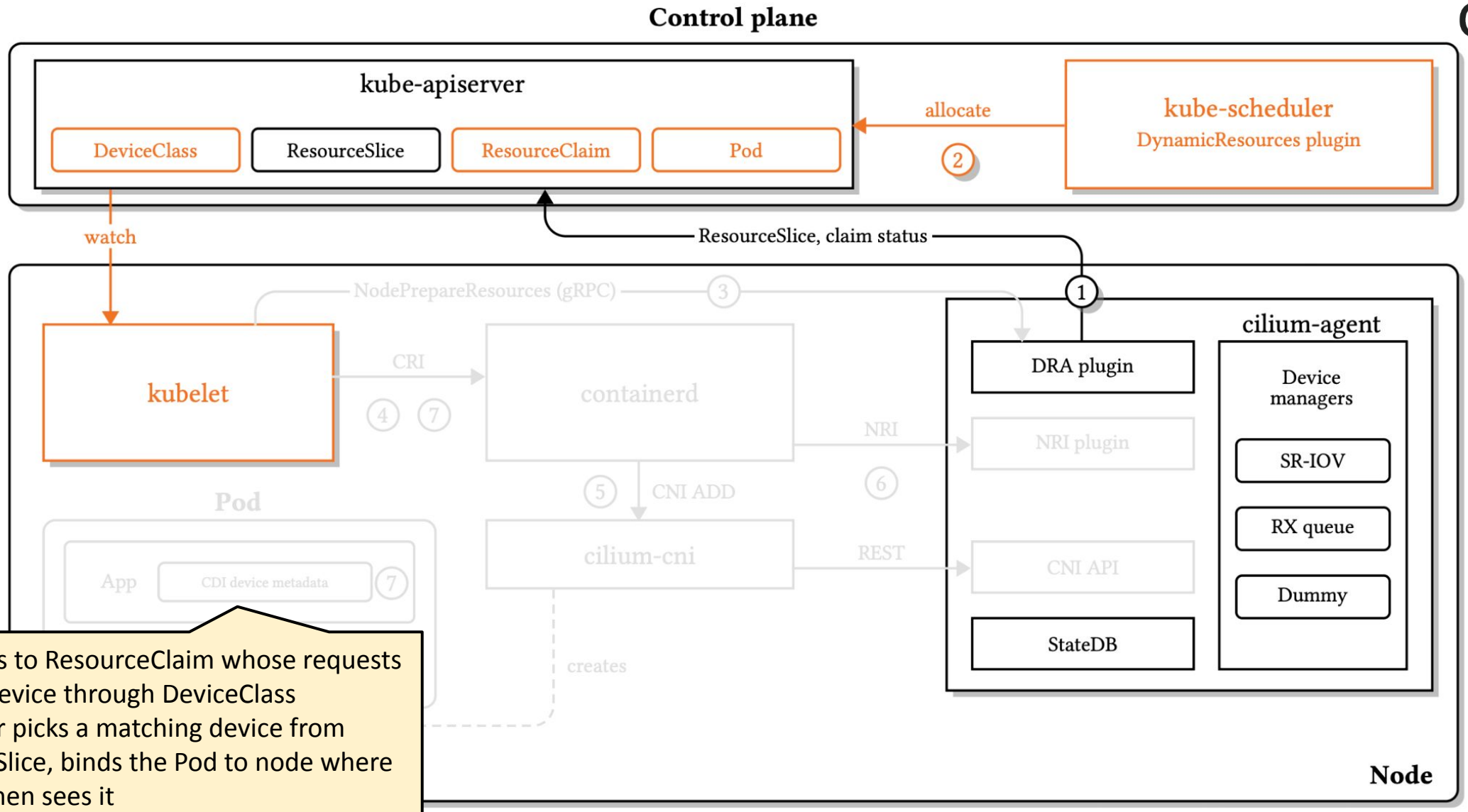
- Cilium "network driver" exposes host network devices and resources to Pods through standard K8s DRA (ResourceSlice, DeviceClass, ResourceClaim)
- 2 flavors implemented as starting point: SRIOV & netkit queue leasing
- cilium-agent embeds a DRA kubelet plugin as well as a NRI plugin for containerd/CRI-O
- CiliumNetworkDriverNodeConfig per node

Cilium's DRA integration



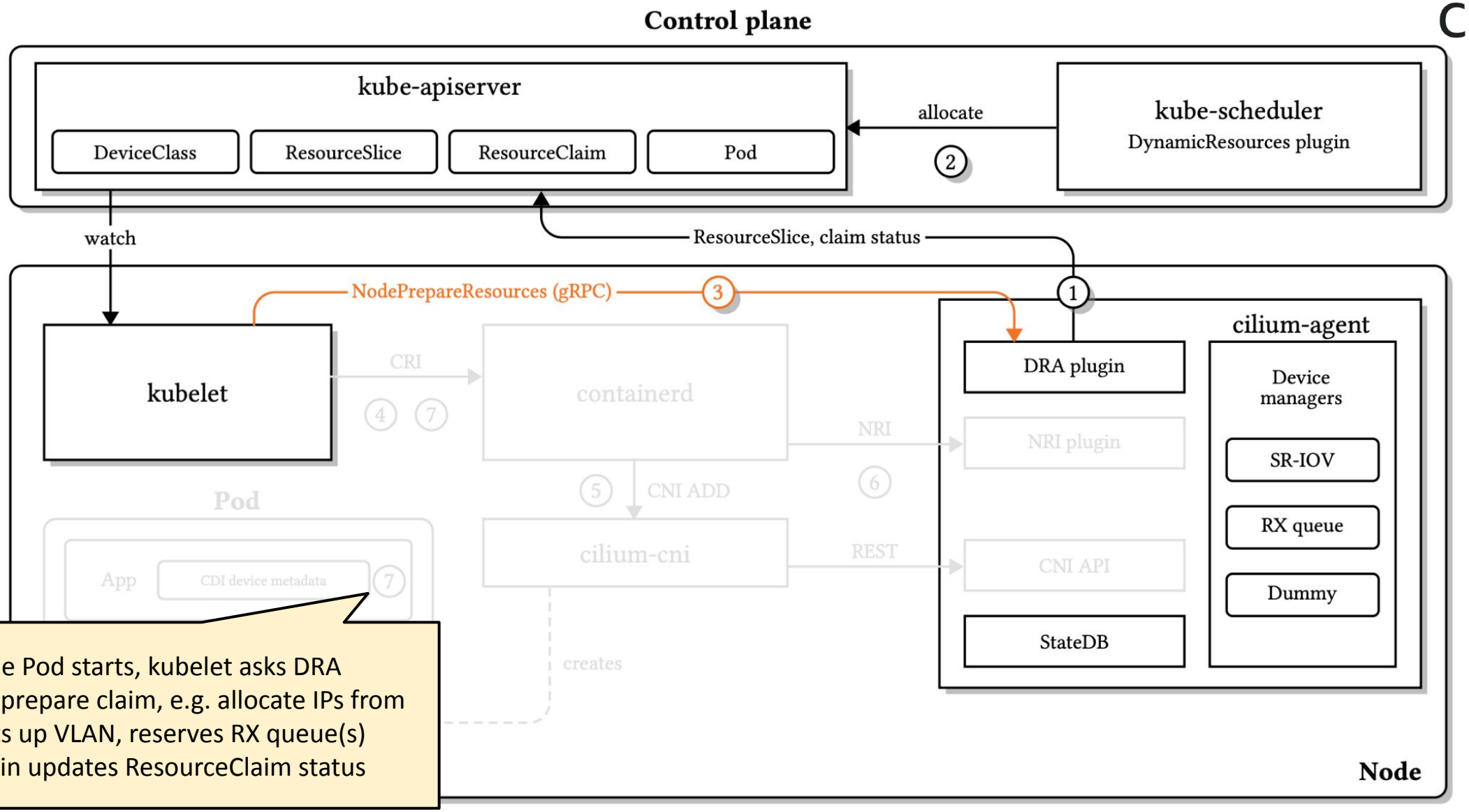
- cilium-agent's device managers discover the node's HW, stores it in StateDB
- DRA plugin publishes them as ResourceSlice, so scheduler knows which devices the node offers

Cilium's DRA integration



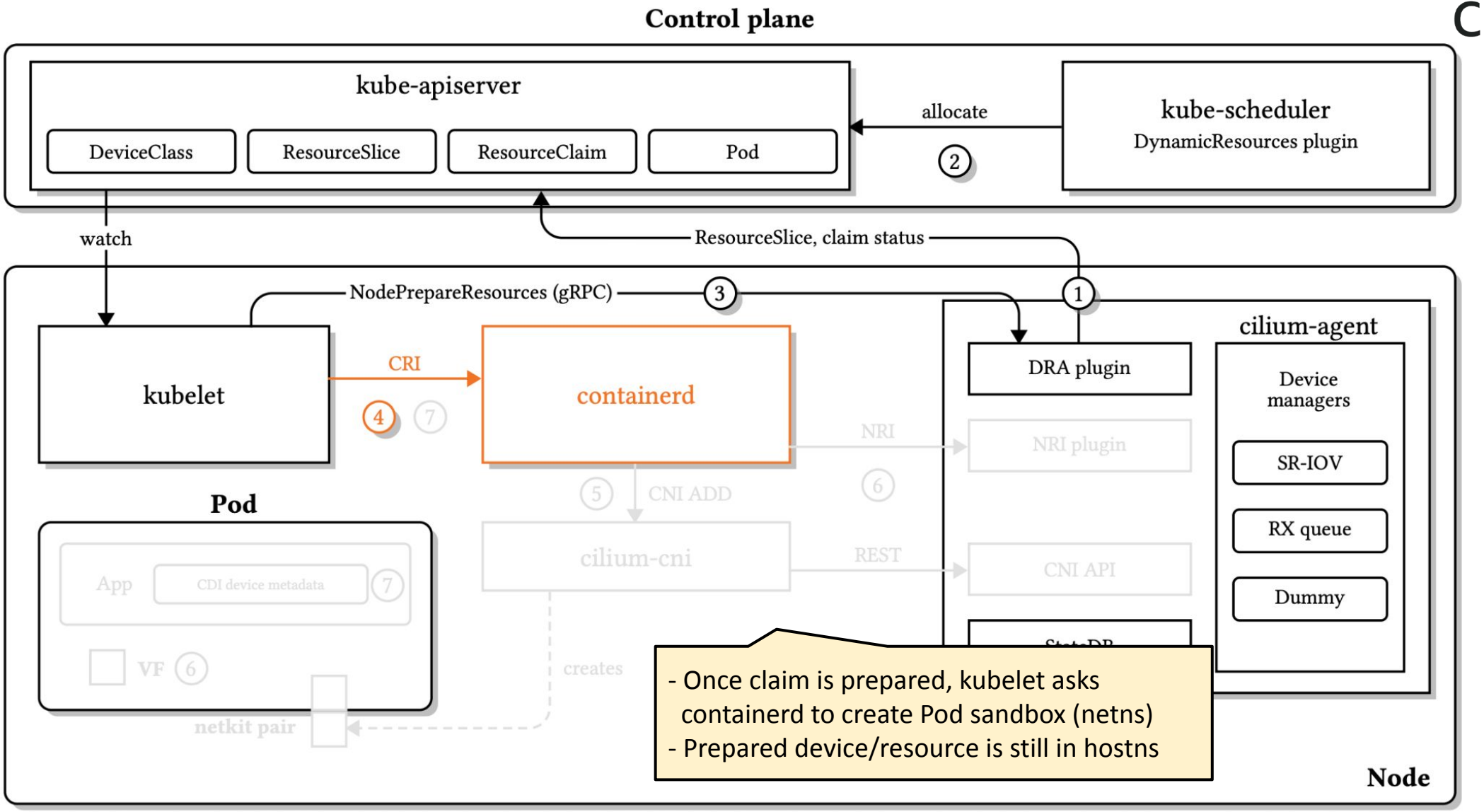
- Pod refers to ResourceClaim whose requests select a device through DeviceClass
- Scheduler picks a matching device from ResourceSlice, binds the Pod to node where kubelet then sees it

Cilium's DRA integration

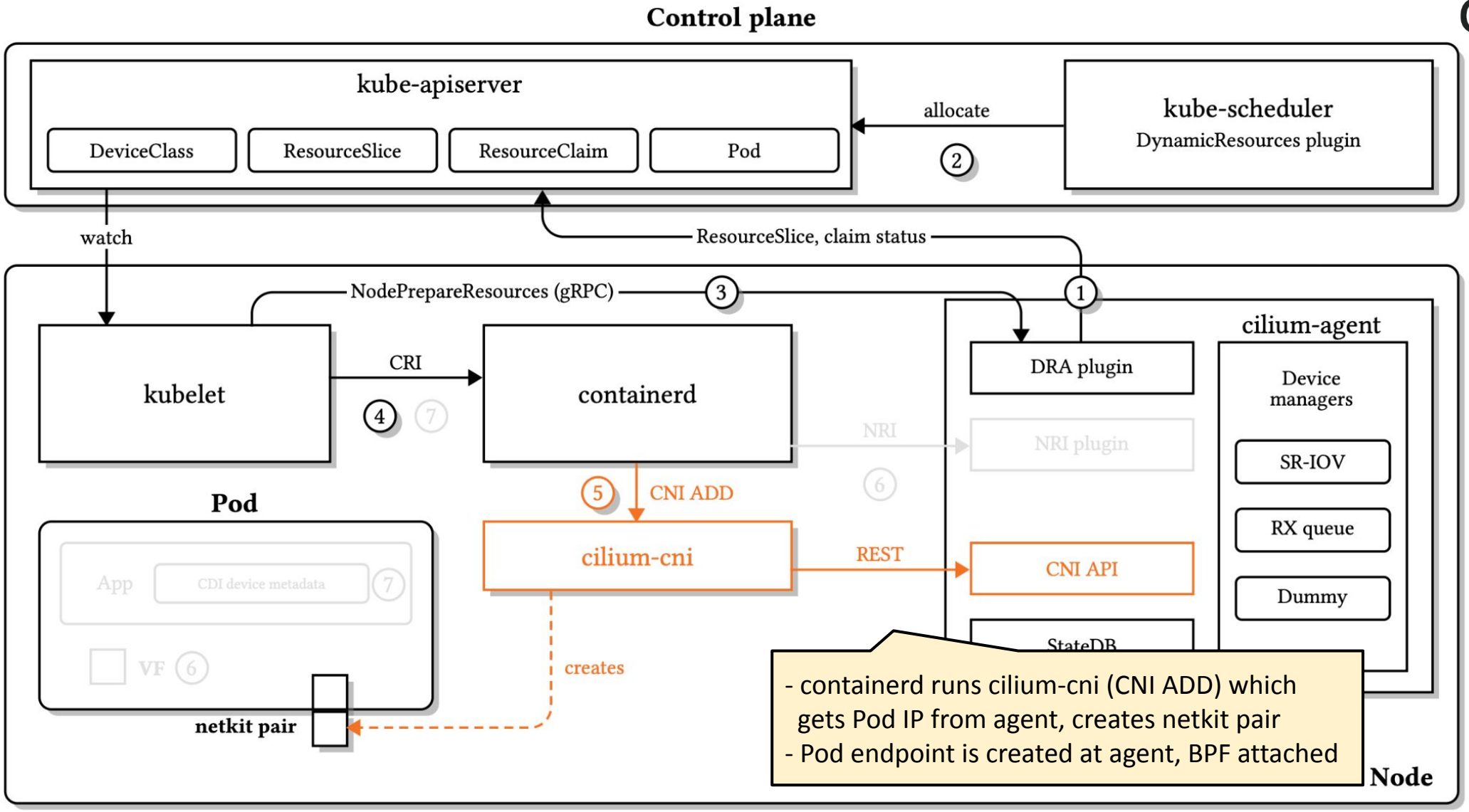


- Before the Pod starts, kubelet asks DRA plugin to prepare claim, e.g. allocate IPs from IPAM, sets up VLAN, reserves RX queue(s)
- DRA plugin updates ResourceClaim status

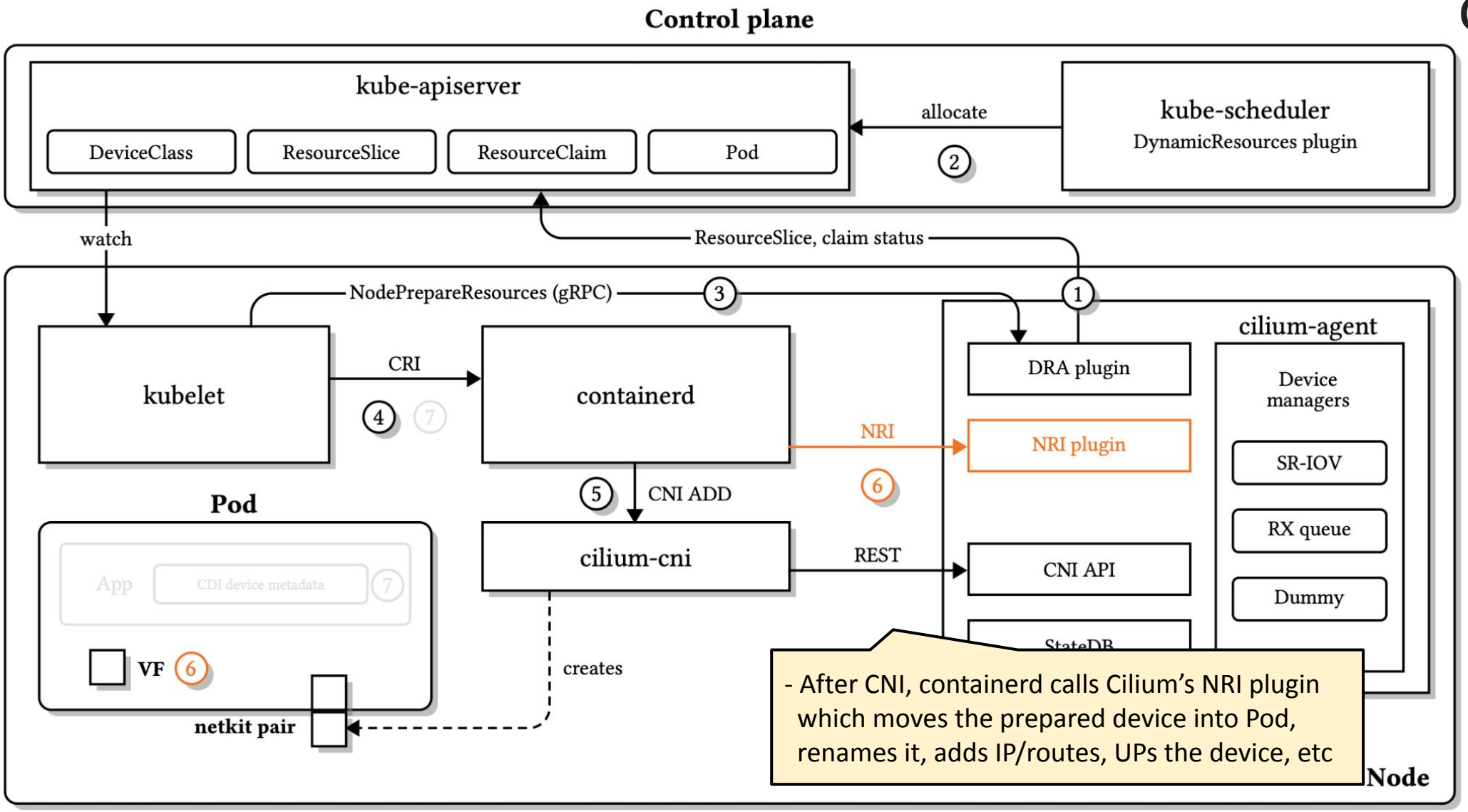
Cilium's DRA integration



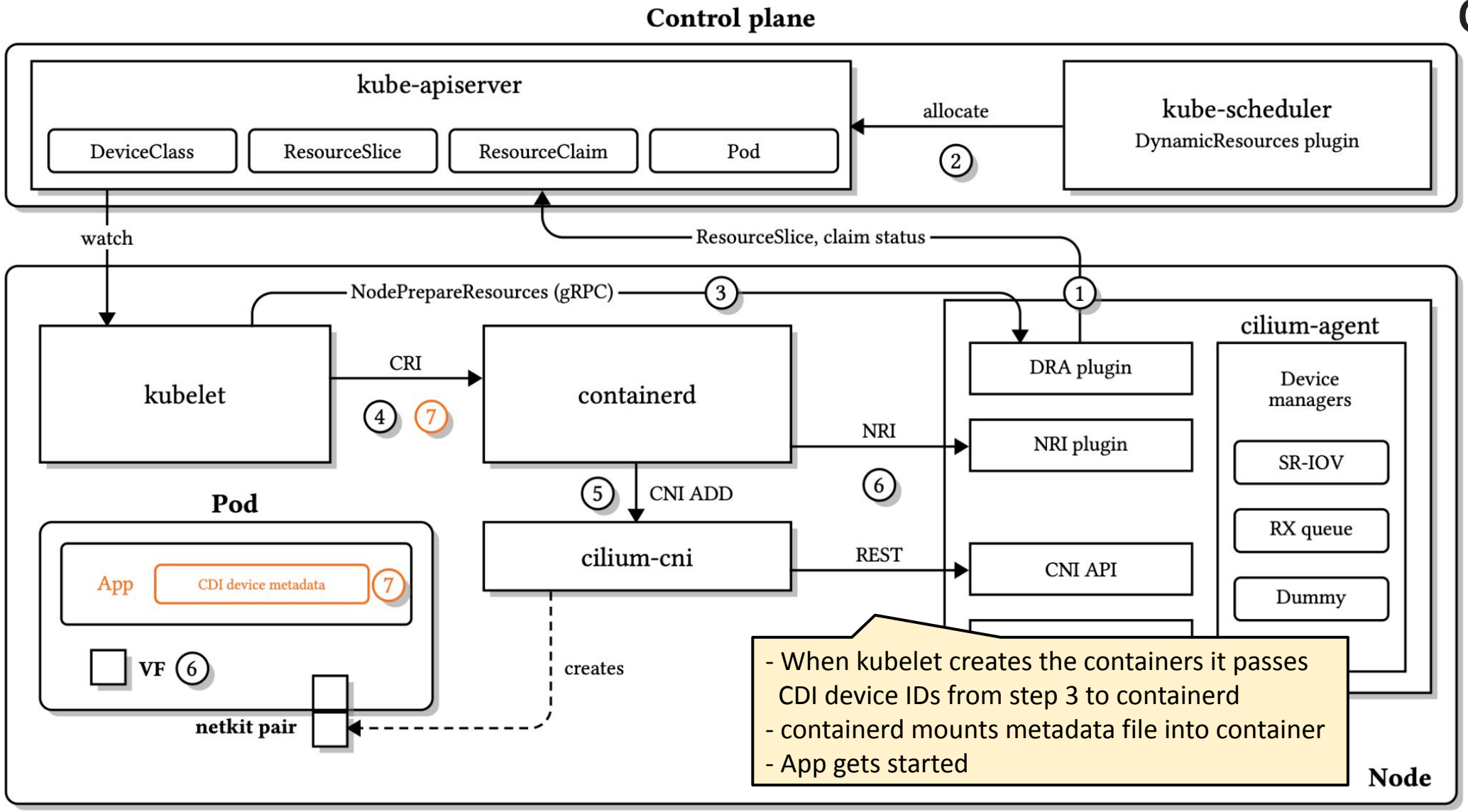
Cilium's DRA integration



Cilium's DRA integration



Cilium's DRA integration



Cilium DRA Driver for SRIOV



- Full HW NIC instance per Pod with its own queues, rings and IRQs
- NIC-internal eSwitch steers straight to the VF port
- Allocation: Changing sriov_numvfs in sysfs means tearing down all VFs on the PF
- Every VF is a full driver instance with its own MSI-X vectors, queues and firmware interface
  mlx5 splits a shared VF MSI-X pool, so more VFs means fewer queues for each VF

Cilium DRA Driver for SRIOV vs virtual netkit

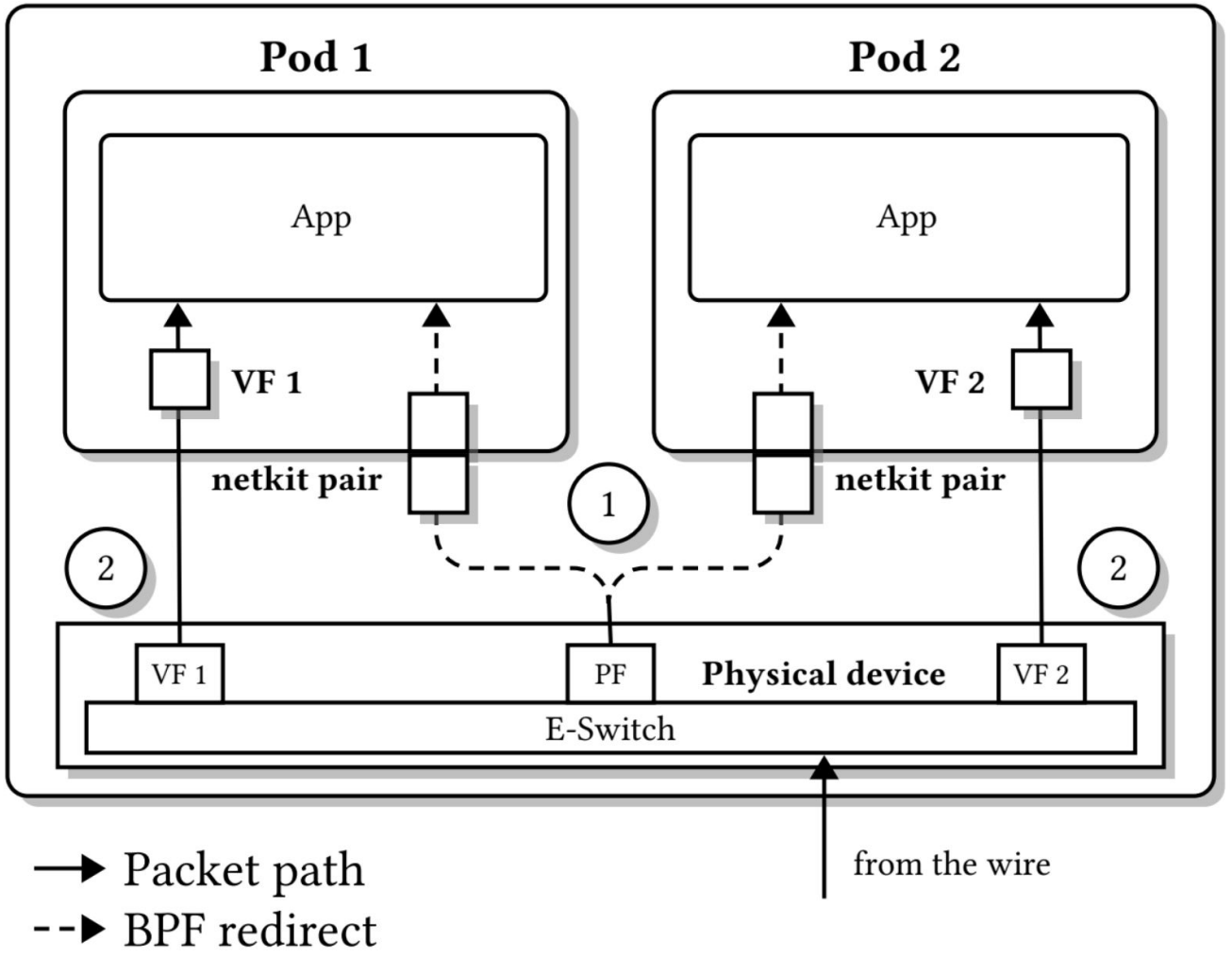
Metric	netkit	SRIOV
Per-packet CPU cost	Kernel TCP + one recv copy + Cilium BPF; host-netns parity	Kernel TCP + one recv copy, no BPF; slightly cheaper
Efficiency with many Pods	Better: one queue per CPU, batching across Pods	IRQ count grows and queues per VF shrink as VFs grow
Same node Pod to Pod	Clearly better: the packet stays in memory	VF to VF crosses PCIe to the NIC's eSwitch and back
Tail latency, many tenants	Shared rings: one Pod's burst delays the others	Better, *iff* each VF's IRQs are pinned to separate CPUs
Multicast	- No NIC filter - Needs expensive SW replication	- HW MC filter + eSwitch replication - IGMP/MLD reach the fabric; works with native routing
Hardware VLAN, and other offload features	No	Yes
Cilium load-balancing, policy, and Hubble visibility	Yes	No, VF traffic bypasses Cilium

Cilium DRA Driver for SRIOV vs virtual netkit

Metric	netkit	SRIOV
Per-packet CPU cost	Kernel TCP + one recv copy + Cilium BPF; host-netns parity	Kernel TCP + one recv copy, no BPF; slightly cheaper
Efficiency with many Pods	Better: one queue per CPU, batching across Pods	IRQ count grows and queues per VF shrink as VFs grow
Same node Pod to Pod	Clearly better: the packet stays in memory	VF to VF crosses PCIe to the NIC's eSwitch and back
Tail latency, many tenants	Shared rings: one Pod's burst delays the others	Better, *iff* each VF's IRQs are pinned to separate CPUs
Multicast	- No NIC filter - Needs expensive SW replication	- HW MC filter + eSwitch replication - IGMP/MLD reach the fabric; works with native routing
Hardware VLAN, and other offload features	No	
Cilium load-balancing, policy, and Hubble visibility	Yes	

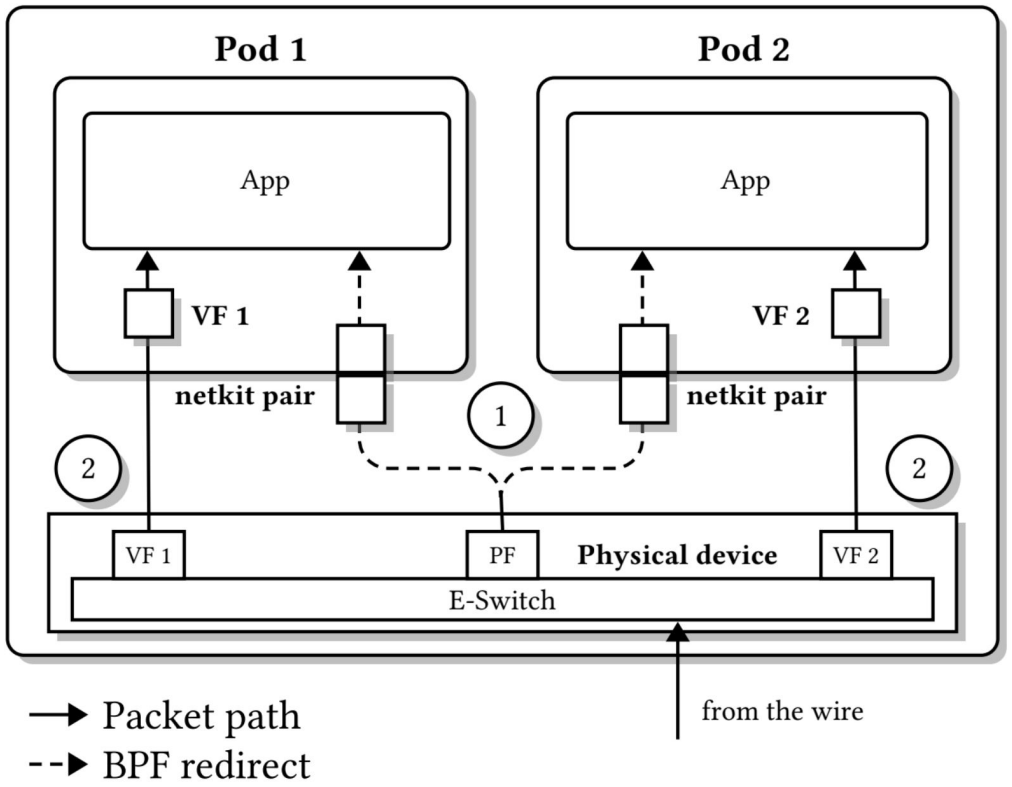
netkit is better in many scenarios, but if multicast matters, why not take a hybrid approach?

Cilium DRA Driver for SRIOV



Cilium DRA Driver for SRIOV

- Use cases: High-frequency trading, dedicated multicast processing or other low-latency workloads
- Bad: VF traffic is not visible/enforceable given it is received and processed directly inside the Pod
- Good: VFs expose their offload capabilities directly into Pod to consume for workloads
- DRA driver has special IPAM & route ingestion features
- Combinable with regular netkit interfaces going through Cilium's datapath



Cilium DRA Driver for Queue Leasing



- Queue leasing: phys netdevice in host namespace dynamically “carves out” a receive queue and hands it to virtual netdevice in Pod such that applications can bind against it

- Payload buffers (“memory providers”) live in Pod memory and are accounted to the application Pod, no access to the host netns is needed

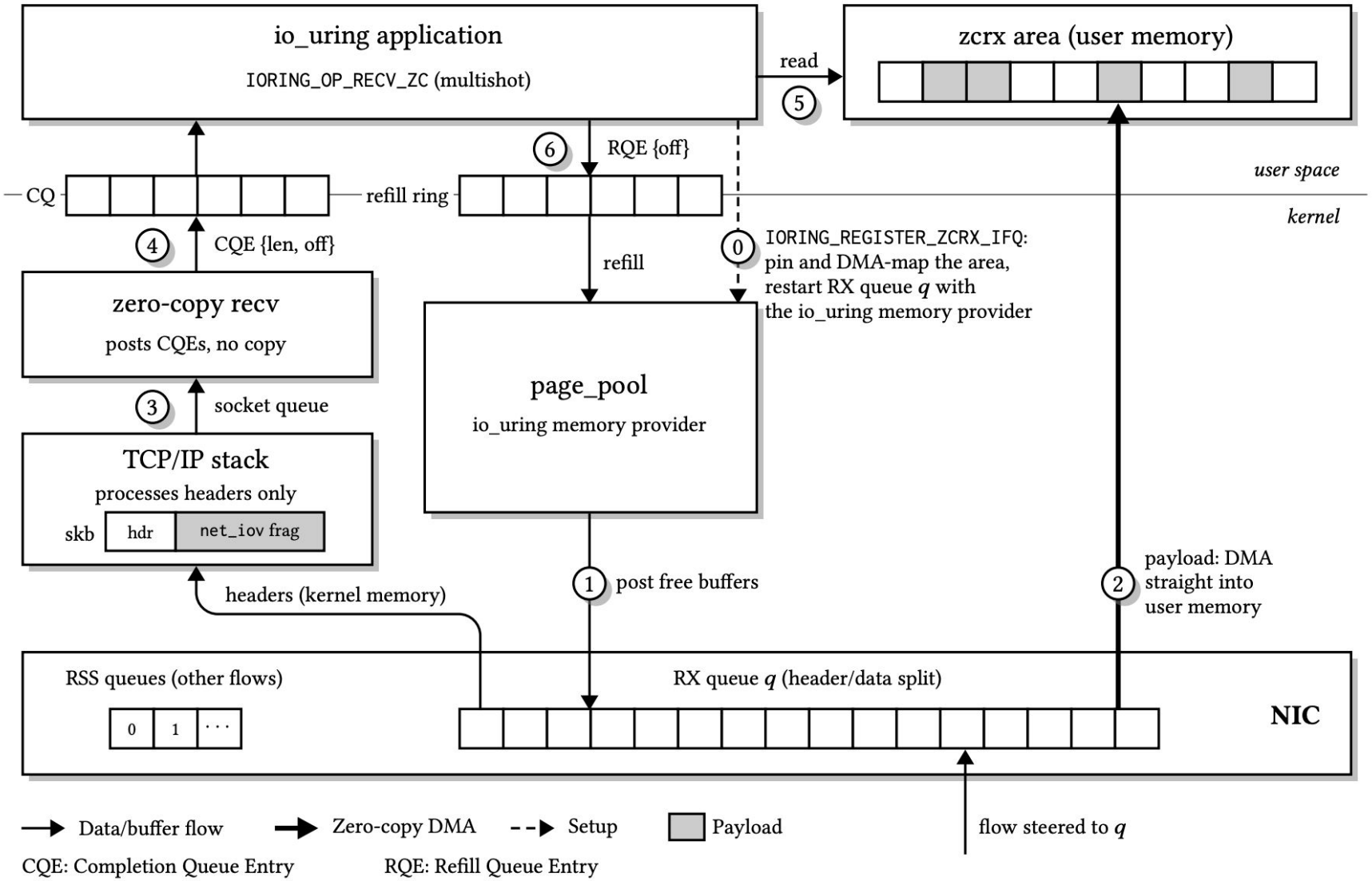
- Merged in Linux kernel v7.1 for netkit, developed specifically for Cilium/k8s Pod use-case

```
Subject: [PATCH net-next v11 00/14] netkit: Support for io_uring zero-copy and AF_XDP
Date: Fri,  3 Apr 2026 01:10:17 +0200   [thread overview]
Message-ID: <20260402231031.447597-1-daniel@iogearbox.net> (raw)

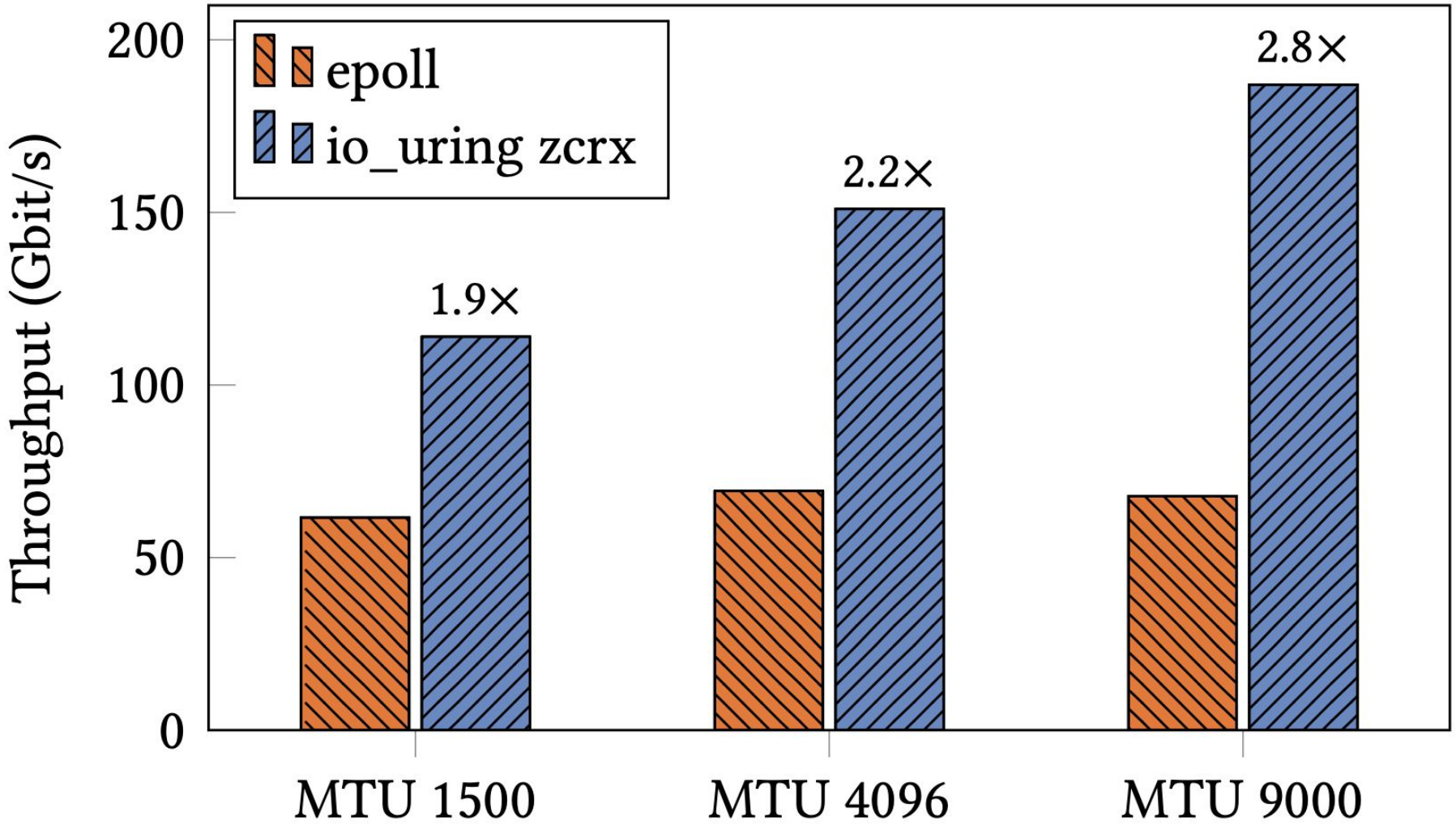
Containers use virtual netdevs to route traffic from a physical netdev
in the host namespace. They do not have access to the physical netdev
in the host and thus can't use memory providers or AF_XDP that require
reconfiguring/restarting queues in the physical netdev.

This patchset adds the concept of queue leasing to virtual netdevs that
allow containers to use memory providers and AF_XDP at native speed.
Leased queues are bound to a real queue in a physical netdev and act
as a proxy.
```

io_uring zero-copy overview

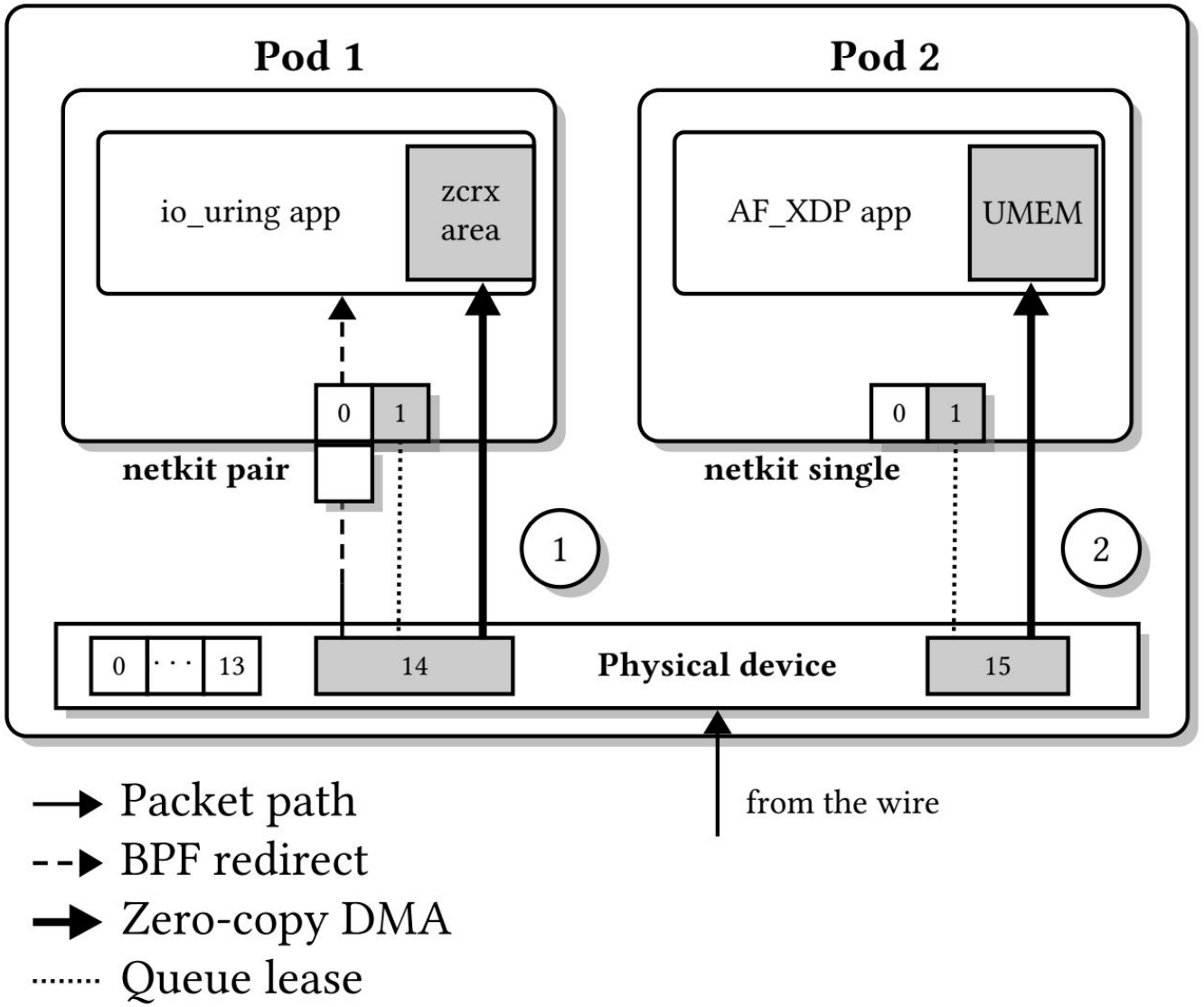


Evaluations: Single TCP flow over the wire



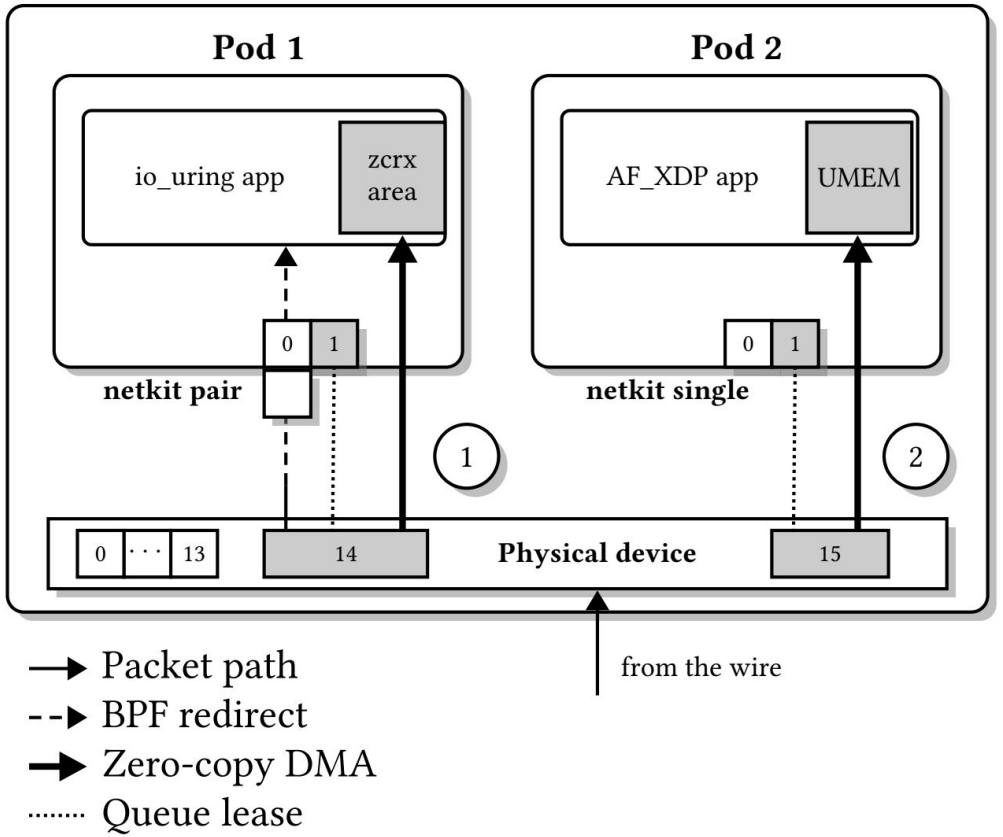
source: CX7 + io_uring zcrx

Cilium DRA Driver for Queue Leasing



Cilium DRA Driver for Queue Leasing

- Use cases: Native zero-copy for Pods via memory providers (io_uring, devmem TCP, AF_XDP/DPDK) storage/GPU workloads, KubeVirt in future
- Such traffic is visible/enforceable/load-balanceable given it takes the regular Cilium datapath
- Memory providers: NICs perform header/data split and payload sits in memory mapped DMA buffers
- Combinable with BIG TCP in GRO engine



Cilium DRA Demo: Pod to run native io_uring zero-copy

```
acam@bluefield02 ~/zcrx-demo $ cat 04-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: zcrx-receiver
  labels:
    app: zcrx-receiver
spec:
  resourceClaims:
    - name: rxq
      resourceClaimName: zcrx-rxq

  containers:
    - name: receiver
      image: docker.io/library/busybox:1.36.1
      command: ["sleep", "3600"]
      resources:
        claims:
          - name: rxq
acam@bluefield02 ~/zcrx-demo $ kubectl apply -f 04-pod.yaml
pod/zcrx-receiver created
acam@bluefield02 ~/zcrx-demo $ kubectl get pod zcrx-receiver -w
NAME            READY   STATUS              RESTARTS   AGE
zcrx-receiver   0/1     ContainerCreating   0          1s
zcrx-receiver   1/1     Running             0          4s
^C
acam@bluefield02 ~/zcrx-demo $ kubectl get resourceclaim zcrx-rxq
NAME       STATE                AGE
zcrx-rxq   allocated,reserved   48s
acam@bluefield02 ~/zcrx-demo $
```

Conclusion



- BPF to specialize the Linux network stack for containers
- netkit as veth replacement eliminates network namespace overhead
- Cilium’s native DRA support to bring offload capabilities into Pods beyond 100Gbps
- Two merged DRA drivers in **Cilium v1.21**: SRIOV for VFs, netkit queue leasing

Conclusion

- BPF to specialize the Linux network stack for containers
- netkit as veth replacement eliminates network namespace overhead
- Cilium's native DRA support to bring offload capabilities into Pods beyond 100Gbps
- Two merged DRA drivers in **Cilium v1.21**: SRIOV for VFs, netkit queue leasing

Lots more to build in future:

- Combined with AF_XDP to speed up KubeVirt (Plumbers'25)
- Accelerate RDMA (netdev 0x1A)

ISOVALENT
now part of cisco



Thank you! Questions?

github.com/cilium/cilium

cilium.io

ebpf.io

Cilium DRA Network Driver docs/template

Cilium & netkit Tuning Guide

netkit paper @ eBPF workshop'26

netkit Linux kernel upstream driver

K8s DRA docs

Appendix: Comparison for netkit and both DRA driver flavors



	netkit	netkit + queue leasing	SRIOV
Use cases	Default Pod networking, microservices, RPC, web	Bulk receive-heavy Pods e.g. storage/GPU-related	Separate network plane in Pod, e.g. multicast or telco-related
K8s visibility	Full	Full* (except payload itself)	VF traffic is invisible to Cilium/hostns
Performance	Host-netns parity	Removes the RX copy: saves CPU and memory bandwidth	Pod itself still runs kernel TCP and copies on RX, no hostns involved
IRQs / softirq isolation	All Pods share the NIC's RSS queues, IRQs and NAPI	Dedicated HW ring for the steered flow; queue removed from RSS	Own MSI-X vectors, EQs, rings and NAPI instance per VF
Linux kernel	>= v6.8, any NIC	>= v7.1; NIC with queue lease + HDS + ntuple (mlx5, bnxt, gve)	SR-IOV NIC with VFs enabled in firmware; any recent kernel
Cilium	Cilium >= v1.16, bpf.datapathMode=netkit	Cilium >= v1.21, K8s >= 1.34, bpf.datapathMode=netkit, networkDriver.enabled=true	Cilium >= v1.21, K8s >= 1.34, networkDriver.enabled=true
Limits	Software device, hence CPU and memory bandwidth	One phys queue per claim, Each leased queue shrinks RSS for the rest of the node	VF count capped by PCI TotalVFs (firmware), resizing sriov_numvfs tears down all VFs on the PF