MARVELL

MACsec-Protected RDMA on DPUs:

From Linux Netdev State to Workload Scheduling

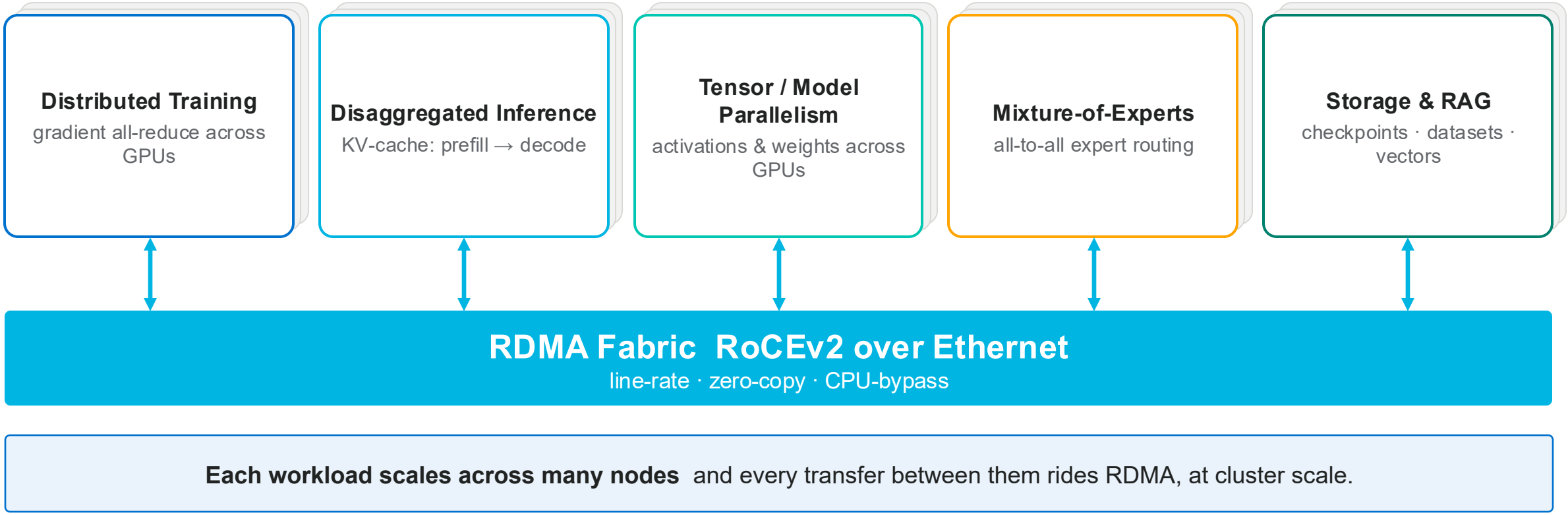
Alkama Hasan

Vijay Ram Inavolu

NetDevConf 2026

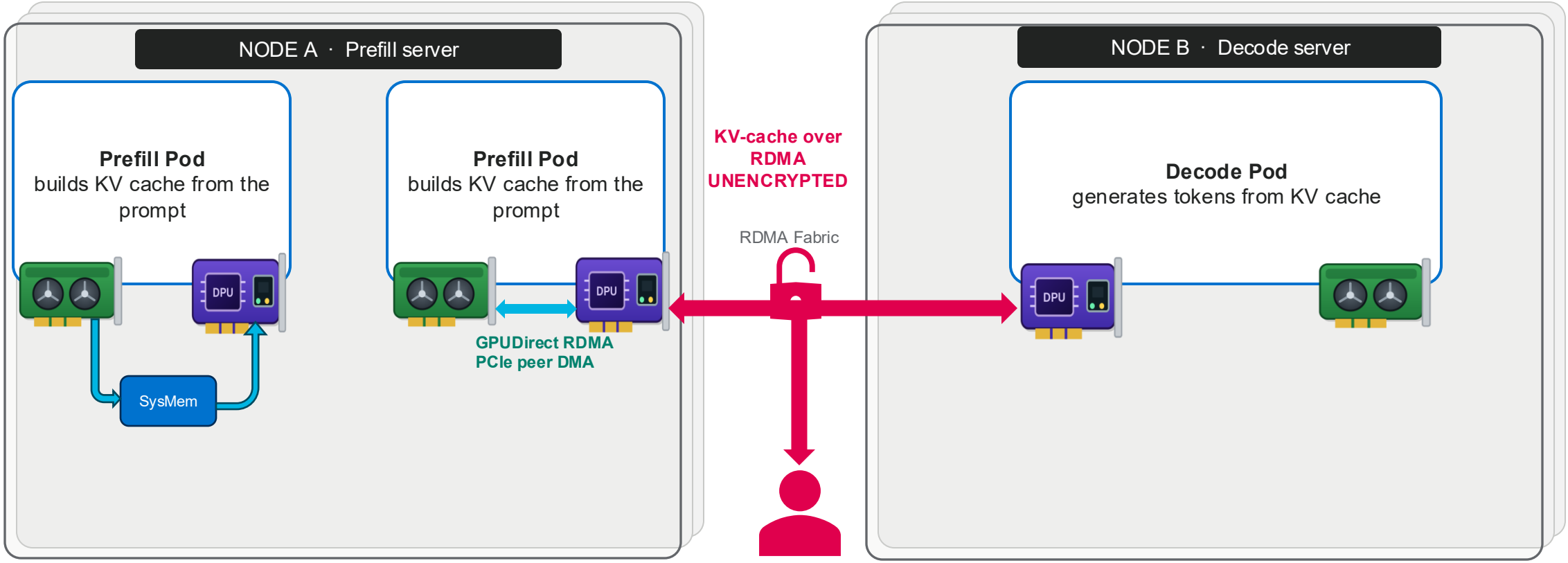
RDMA: The Universal Fabric for AI Workloads

From training to inference to retrieval AI's core workloads all move their data over RDMA.



Each workload scales across many nodes and every transfer between them rides RDMA, at cluster scale.

An Example RDMA Use Case: KV-Cache Travels Unencrypted



RDMA verbs bypass the kernel TLS / kTLS / in-kernel IPsec can't see this path. The KV-cache crosses the fabric OPEN: exposed to a passive tap, rogue switch, or co-tenant.

DPUs Are Being Widely Adopted

DPU one platform · many jobs running at once

Infrastructure Offload

via Red Hat DPU Operator + Cilium CNI Offload

Full CNI / eBPF networking stack on the DPU pod networking to L7 policy; host CPU freed.

Dpu Operator managed Network Function Acceleration on DPU

MCP Offload

On Openshift

Application load balancing & Model Context Protocol Load balancing and TLS termination offload to DPU using NGINX+

AI Inference Gateway Offload

via CNI Offload

LLM-d inference Gateway offload including Envoy, EPP and full datapath offload to DPU

Secured RDMA Offload

via DRA NetSec

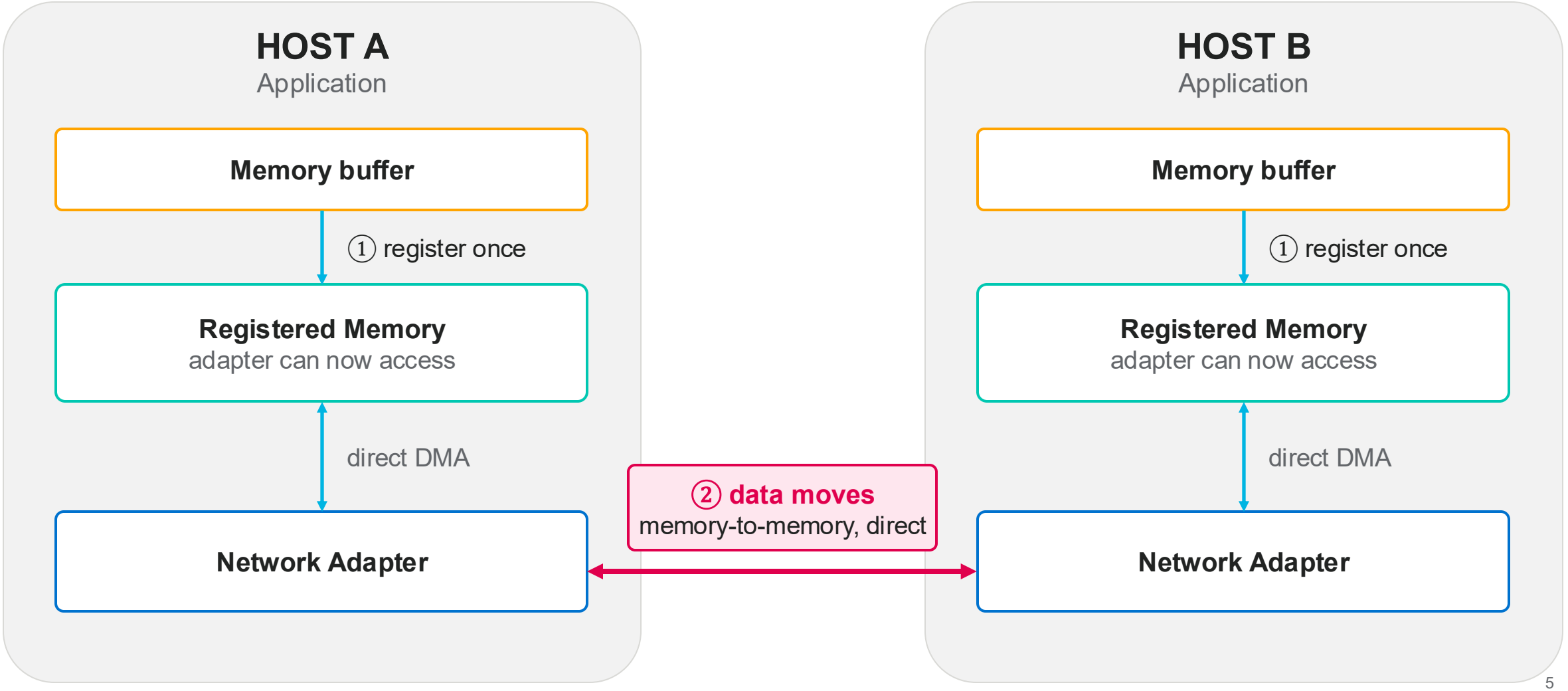
Offloads RDMA to the DPU + inline hardware MACsec encryption

→ made schedulable to Pods via DRA

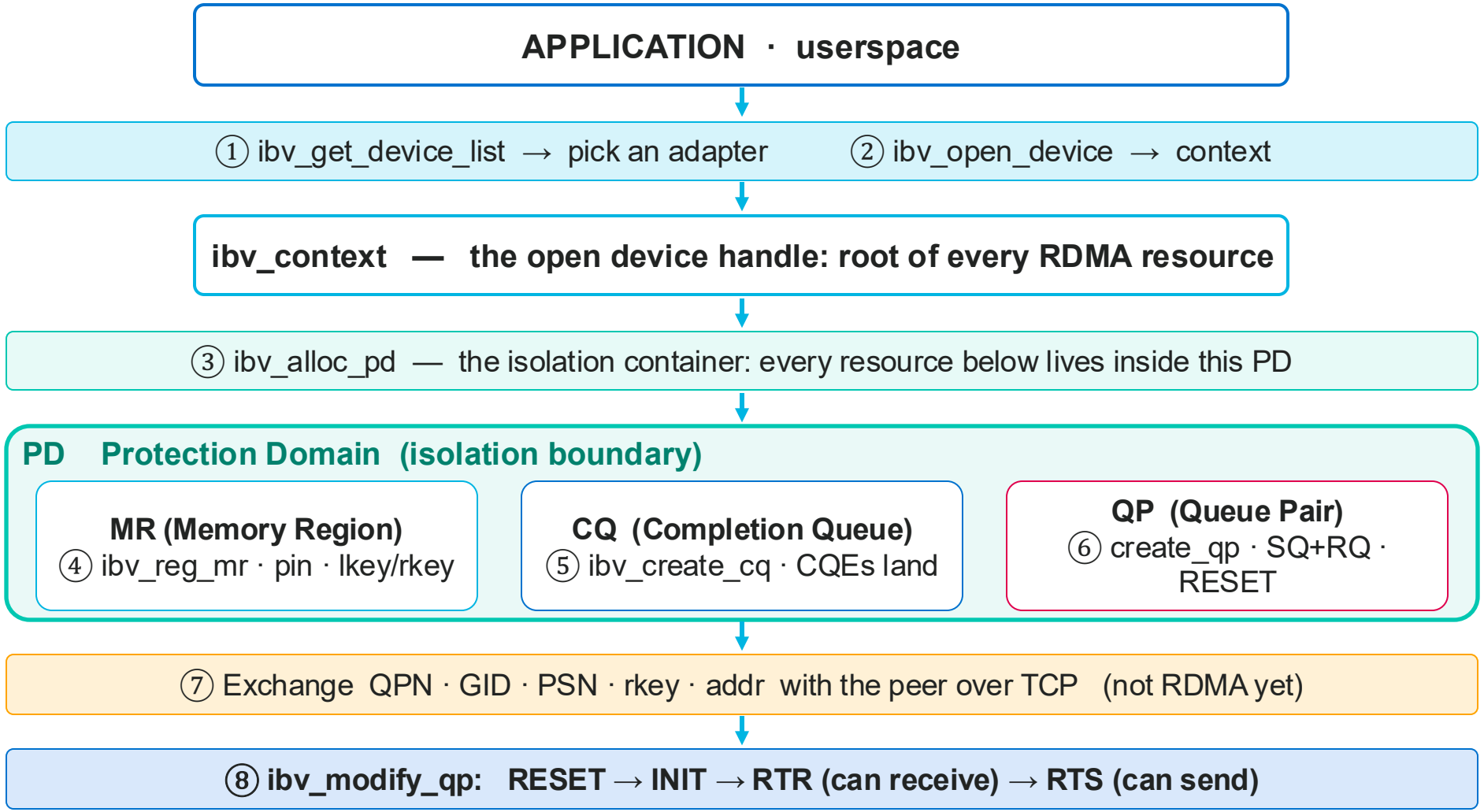
THIS TALK

Already on every node, running many jobs at once **Secured RDMA is just one more offload, no new hardware.**

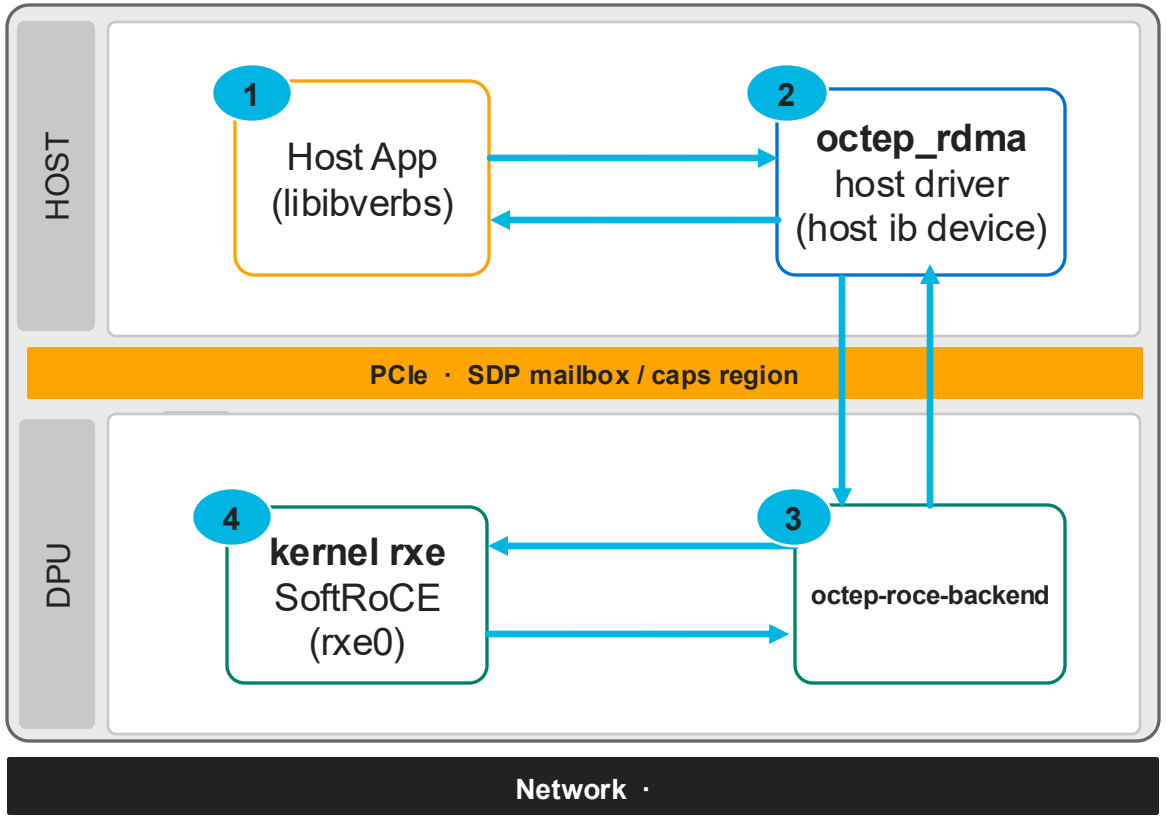
What is RDMA: Direct Memory to Memory



The RDMA Control Path: From Device to Ready QP



DPU to Host: RDMA via the PCIe Mailbox



Setting Up RDMA Control Plane

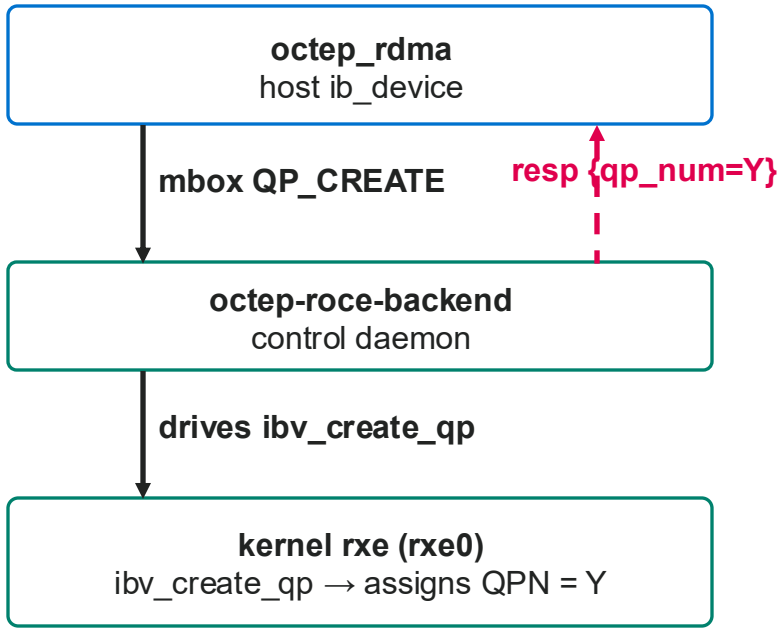
1. **octep_rdma :**
   creates host side PD,CQ,QP,MR + rings/Buffers
   mbox QUERY_DEVICE_CAPABILITY using BAR4
   Set MACsec + RDMA HW Capability to netdev of host device
2. **octep-roce-backend:**
   Detect MACSec & RDMA Capabilities and return to Host
3. **Kernel rxe:** Does the actual rdma creates rdma packet and send to skb

PD Alloc: octep_rdma =mbox PD_ADD==> octep-roce-backend -> ibv_alloc_pd(rxe0) -> pdn
octep_rdma <=mbox rc (0xbeef)==== octep-roce-backend

CQ Create:
octep_rdma =mbox CQ_CREATE==========> octep-roce-backend -> ibv_create_cq(rxe0) (keep host CQ-ring map)
octep_rdma <=mbox rc================== octep-roce-backend

Inside the Control Plane: QP Create & MR Register

① QP CREATE : rxe owns the QPN (inverted)

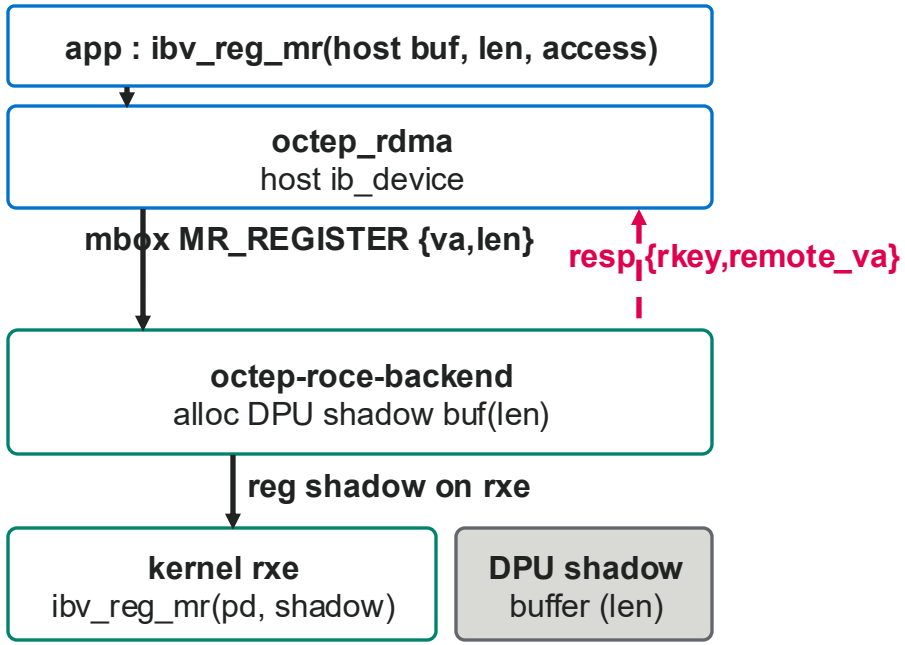


ABI add: octep_rdma_qp_create_resp

host ADOPTS ibqp->qp_num = Y
before app reads it

solid → mbox request dashed → response / adopt

② MR REGISTER : DPU shadow buffer (dao_dma bounce)

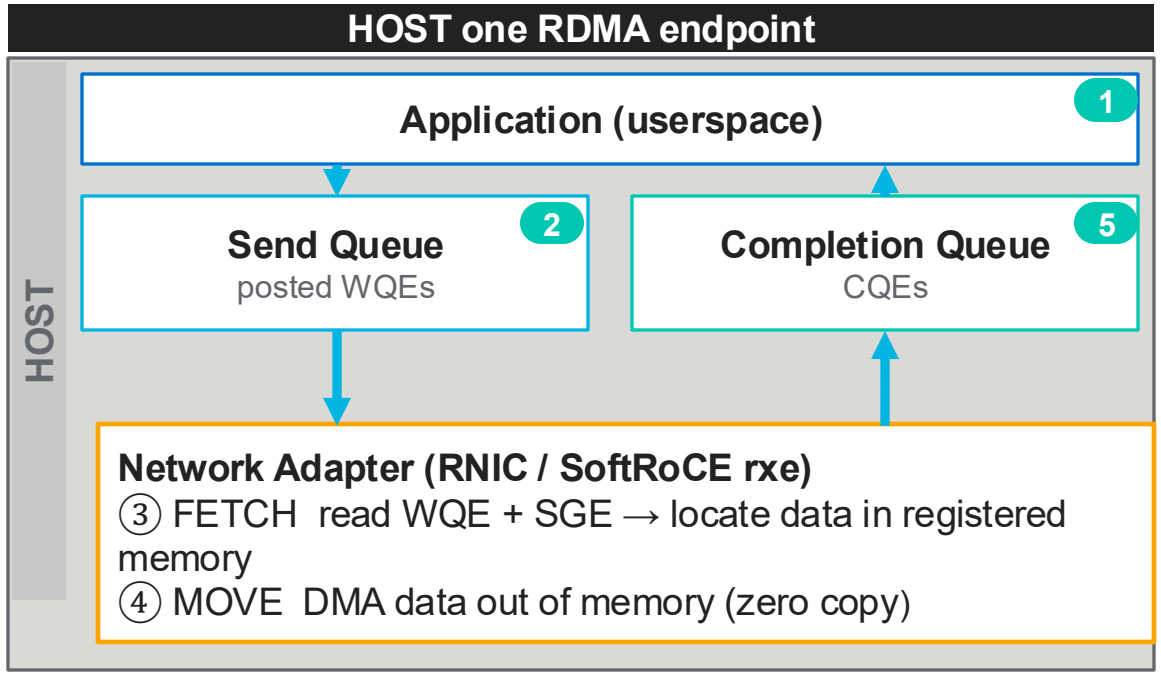


ABI add: octep_rdma_mr_register_resp

host ADOPTS rkey + remote_va
keeps host lkey

payload dao_dma-bounced host ↔ shadow on datapath

The RDMA Data Path: From Verb to Wire



How it works one RDMA operation

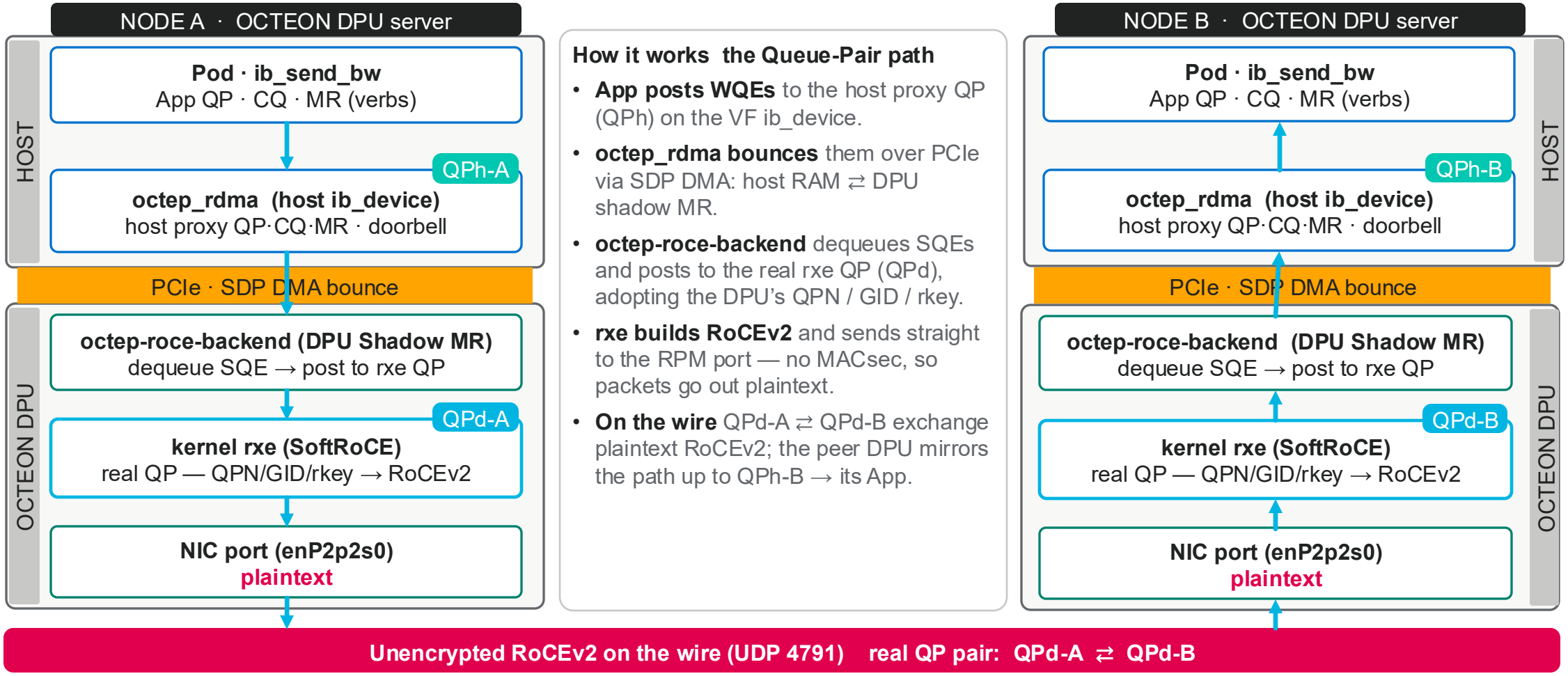
- ① **POST** ibv_post_send(WQE): post "send / RDMA-write here."
- ② **RING** doorbell, one MMIO write: "new work, go!"
- ③ **FETCH** adapter reads WQE + SGE, finds data in registered memory.
- ④ **MOVE** DMA data out of memory, zero-copy, onto the wire.
- ⑤ **COMPLETE** ibv_poll_cq() reads CQE → "done" (status, bytes).

RoCEv2 on the wire → to remote memory · remote host CPU not involved

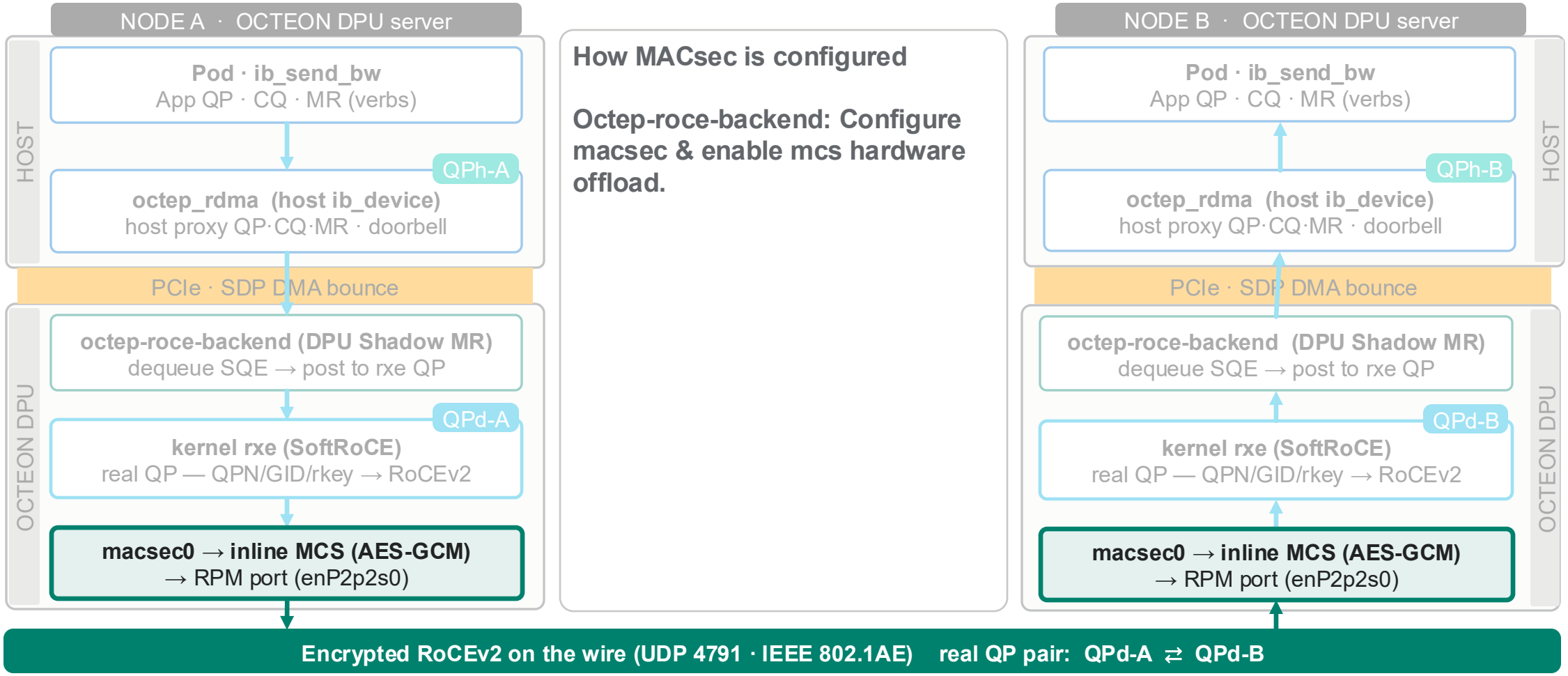
What actually moves on the wire one RoCEv2 packet:

Ethernet L2 · MACs	IP L3 · src/dst	UDP dport 4791	BTH opcode·QP·PSN	RETH VA · rkey · len	PAYLOAD	ICRC IB CRC	FCS Eth CRC

DPU Offloaded RDMA Data Path : Unencrypted



DPU-Offloaded RDMA Data Path with MACsec HW Encryption



Kubernetes: The De-Facto Platform for AI

The AI ecosystem runs on Kubernetes open-source stack + every hyperscaler

Kubeflow	KServe	Ray	vLLM · llm-d	GPU Operator	Kueue · Volcano

Google

Serves Llama 405B and OpenAI gpt-oss on Google Kubernetes Engine(GKE). Built GKE Inference Gateway for LLM-aware routing with prefix KV cache.

Amazon + Anthropic

Built 100,000-node Amazon Elastic Kubernetes Service(EKS) clusters for training. Launched “AI on EKS” as the official inference framework.

Microsoft Azure

KAITO operator makes vLLM the default engine on Azure Kubernetes Service (AKS). DRA + MIG for GPU slicing.

66% of GenAI-hosting organizations use Kubernetes for inference

82% of container users now run Kubernetes in production (up from 66% in 2023)

Sources: CNCF Annual Cloud Native Survey 2025 · Google Cloud · AWS · Microsoft Azure

Why Kubernetes? what makes it the default for AI

Scalable	Portable	Reliable	Extensible
1000s of nodes & GPUs	cloud · on-prem · edge	self-healing · auto-recovery	operators & DRA

Dynamic Resource Allocation(DRA)

DRA you describe the hardware you need, and Kubernetes finds a device that matches.

The category
DeviceClass
names a kind of device and which attributes matter (rdma, macsec…)

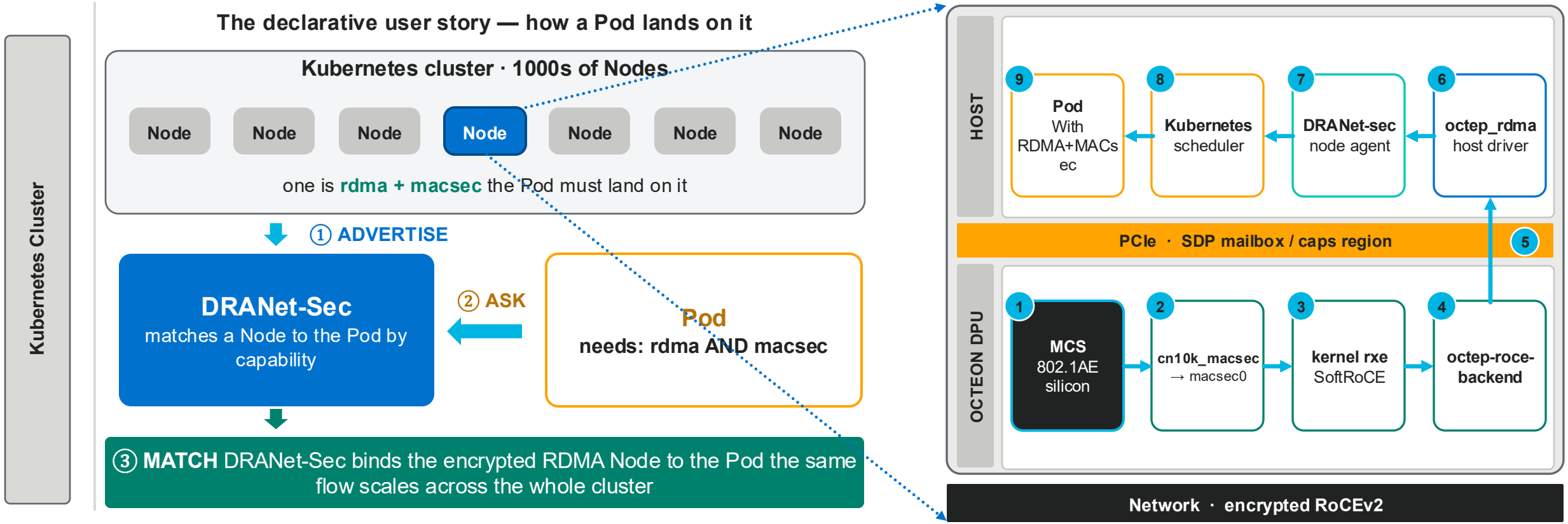
The catalog
ResourceSlice
each node lists the devices it has, with their attributes

The request
ResourceClaim
a Pod asks for a device with the attributes it needs

The matchmaker
Scheduler
picks a device that matches — and attaches it to the Pod

Nodes list what they have → Pod asks for what it needs → Kubernetes matches them

DRANET-Sec: Silicon Capability to Pod at Cluster-Wide

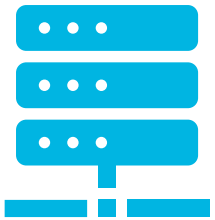


Deploy at Cluster Scale the Easy Way with DRANet-Sec

Kubernetes cluster · scale-out

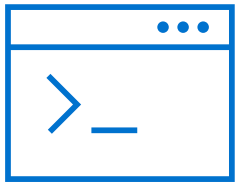
①DRANet-Sec node agent
◆ **Discover & publish**

find VFs with
macsec + rdma caps
publish ResourceSlice



②USER · kubectl apply
◆ **Declare intent**

rdma && macsec
ResourceClaim + Pod
stock RDMA image

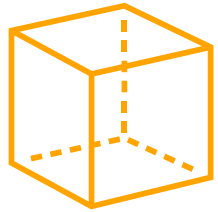


③ k8s scheduler
◆ **Match & allocate**

match rdma + macsec
encrypted VF only
pin pod to node

④ Pod · runtime
◆ **Attach & run**

VF → pod netns
ib_send_bw (stock)
→ rxe → MCS block

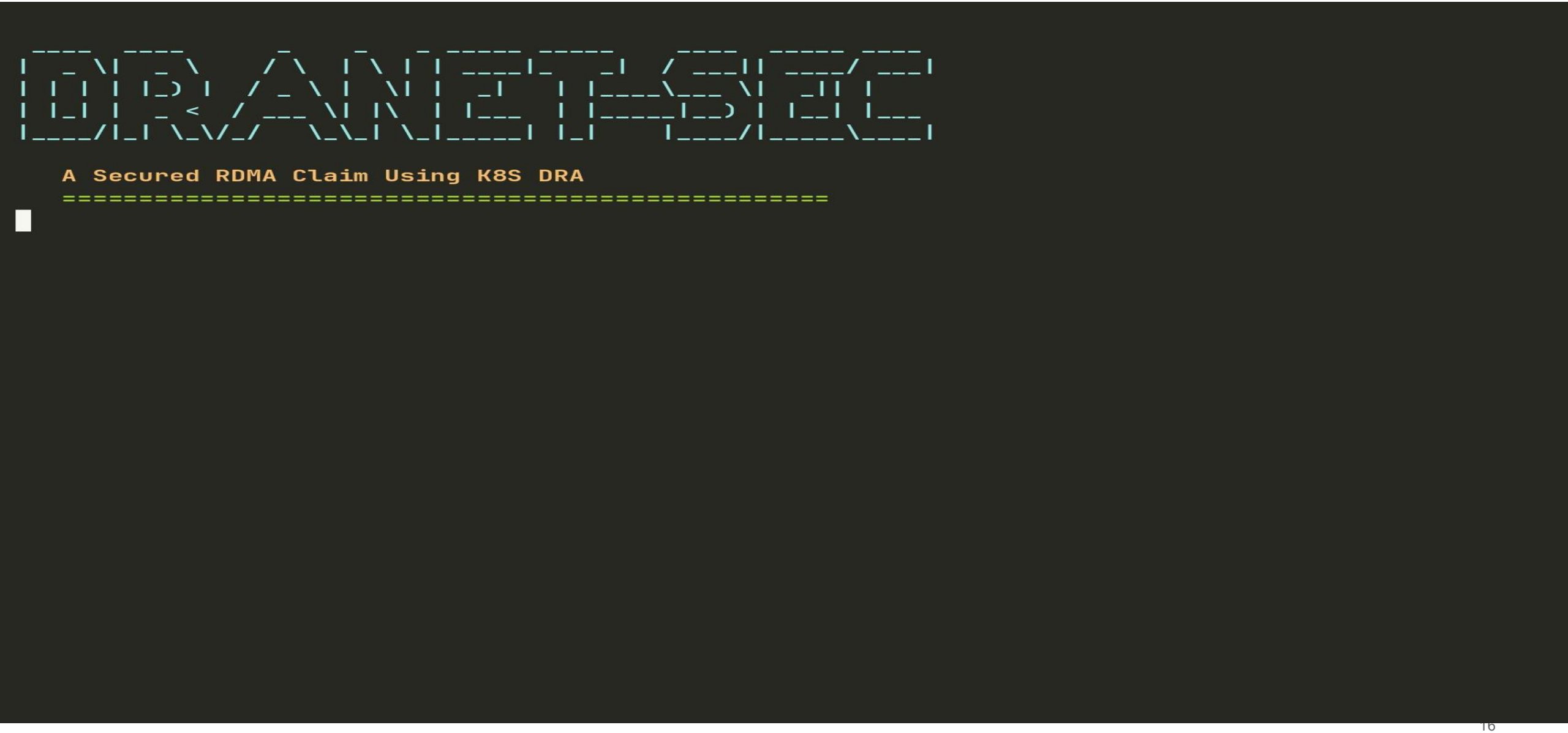


⑤ Outcome
◆ **An Encrypted RDMA**

inline MCS encrypt
802.1AE line-rate with rdma traffic.

Everything you've seen RDMA + inline HW MACsec deploys at cluster scale from one declarative request. That's DRANet-Sec.

Demo



Thank You

Essential technology, done right™