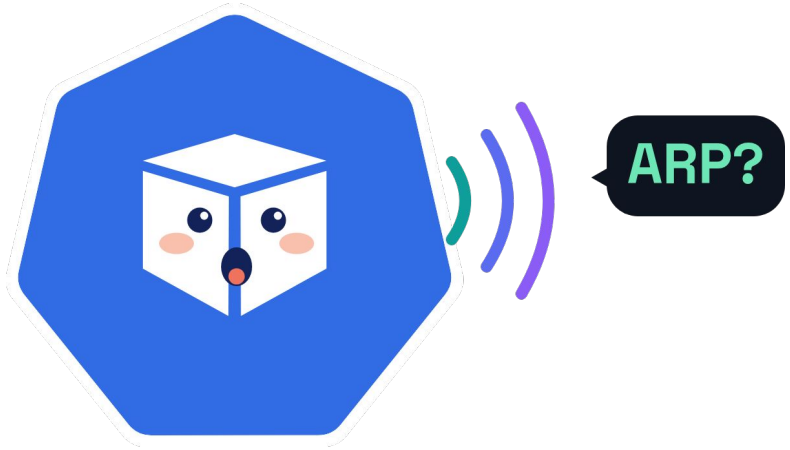


CLOUD NATIVE NOVICE · NOV 11

From Pod Creation to First Packet

What actually happens in the two seconds between `kubectl apply` Pod and till that Pod can send & receive traffic.

📱 https://partici.fi/33510940

TRACK
Cloud Native Novice

FORMAT
25 min + 5 Q&A

CNI FOR DEMOS
kindnet · live on kind

Hi, I'm Surya 👋

Red Hat · Kubernetes networking

- I work on Kubernetes networking at Red Hat, focussed on **OVN-Kubernetes** & SIG-Network Policy API.
- Day to day: how a Pod gets connected and everything that can go wrong.
- Why this talk: the **"~2 seconds"** before a Pod's first packet is where a lot of us get stuck.



github.com/**@tssurya**

It starts with ~8 lines of YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: demo
spec:
  containers:
  - name: app
    image: agnhost
```

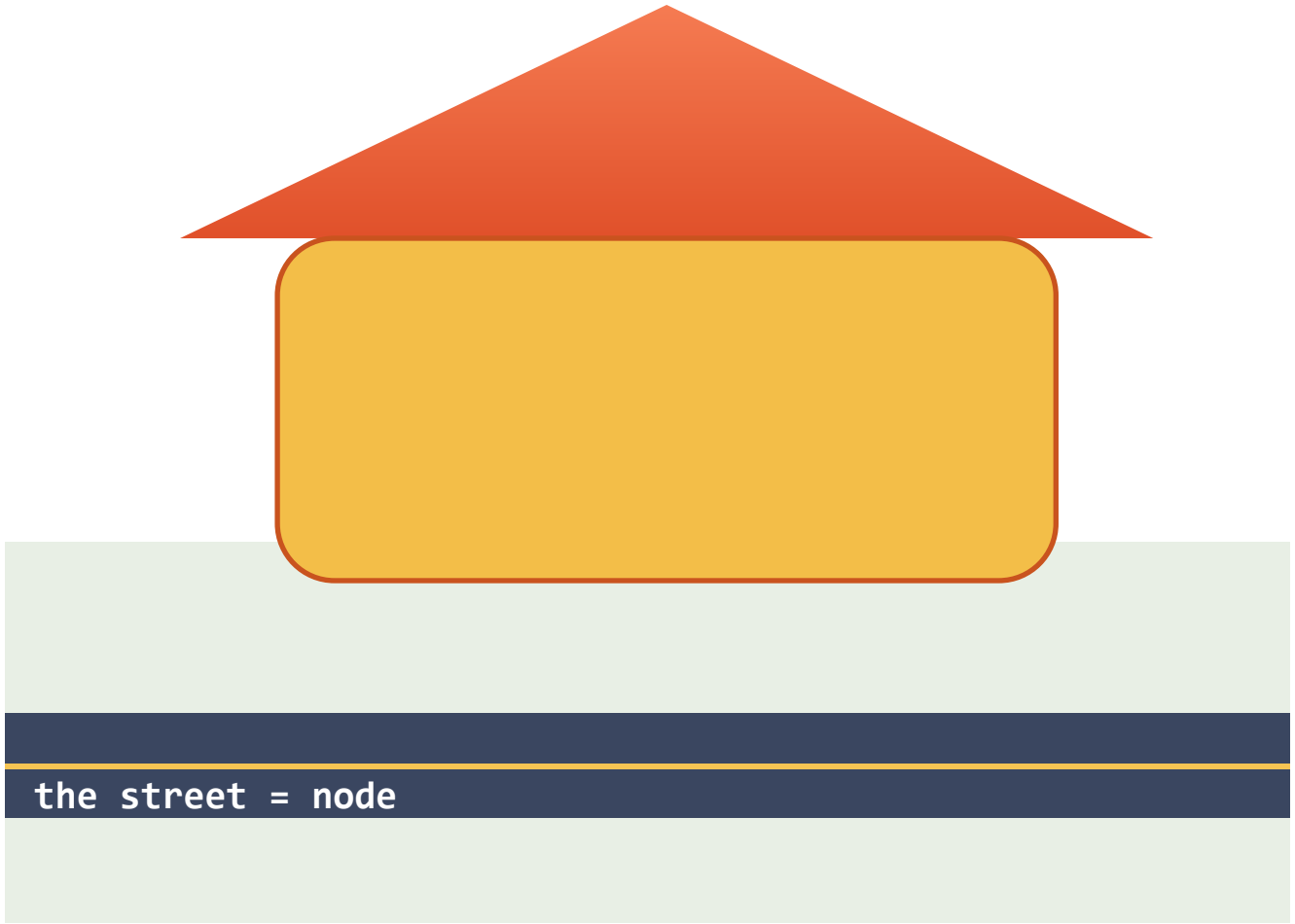
```
$ kubectl apply -f pod.yaml
```

Pending → ContainerCreating → Running

… in ~2 seconds.

What actually happened in those 2 seconds from a networking standpoint?

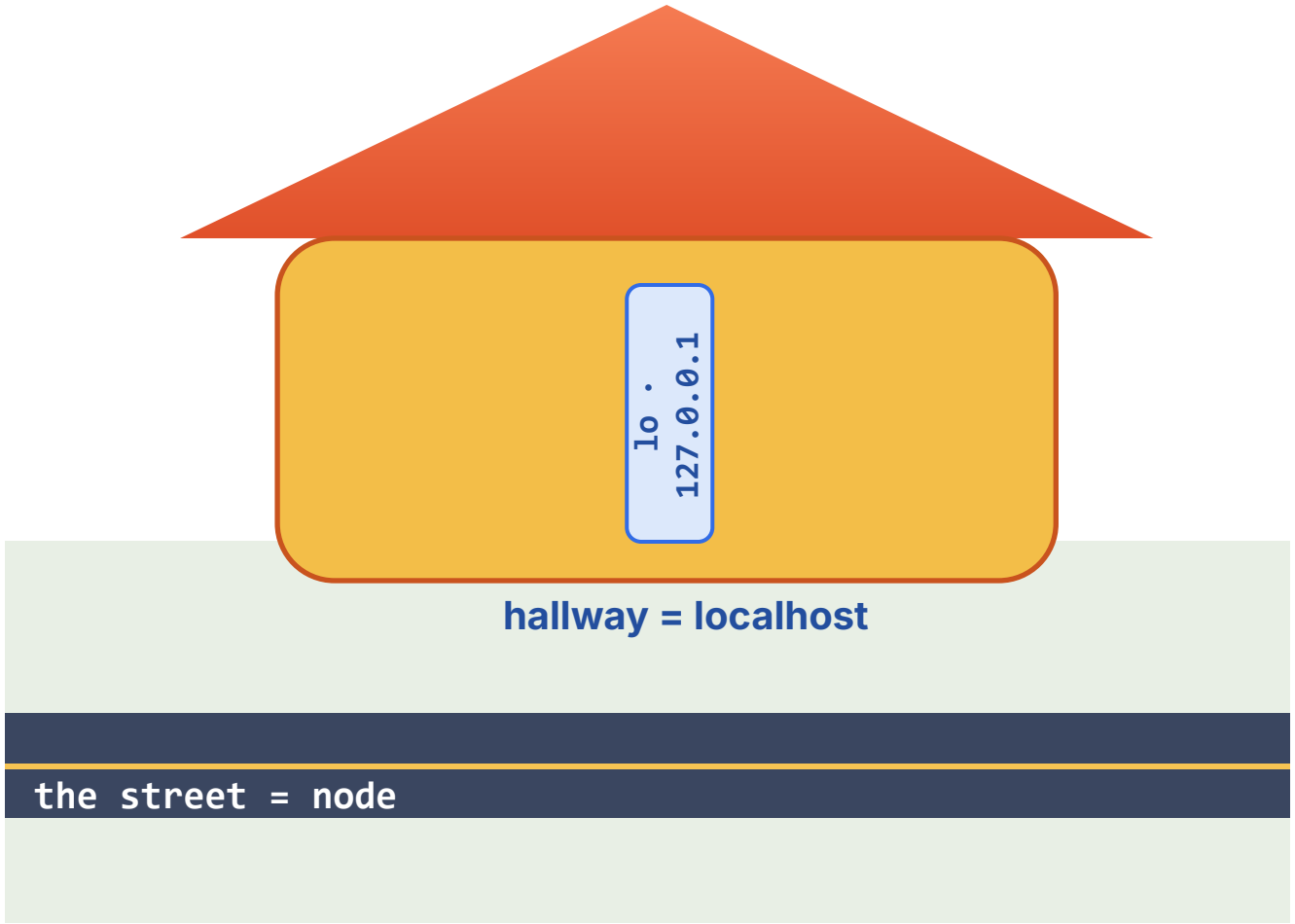
A Pod is a house



🏠 The whole house

One Pod = One network namespace.

A Pod is a house



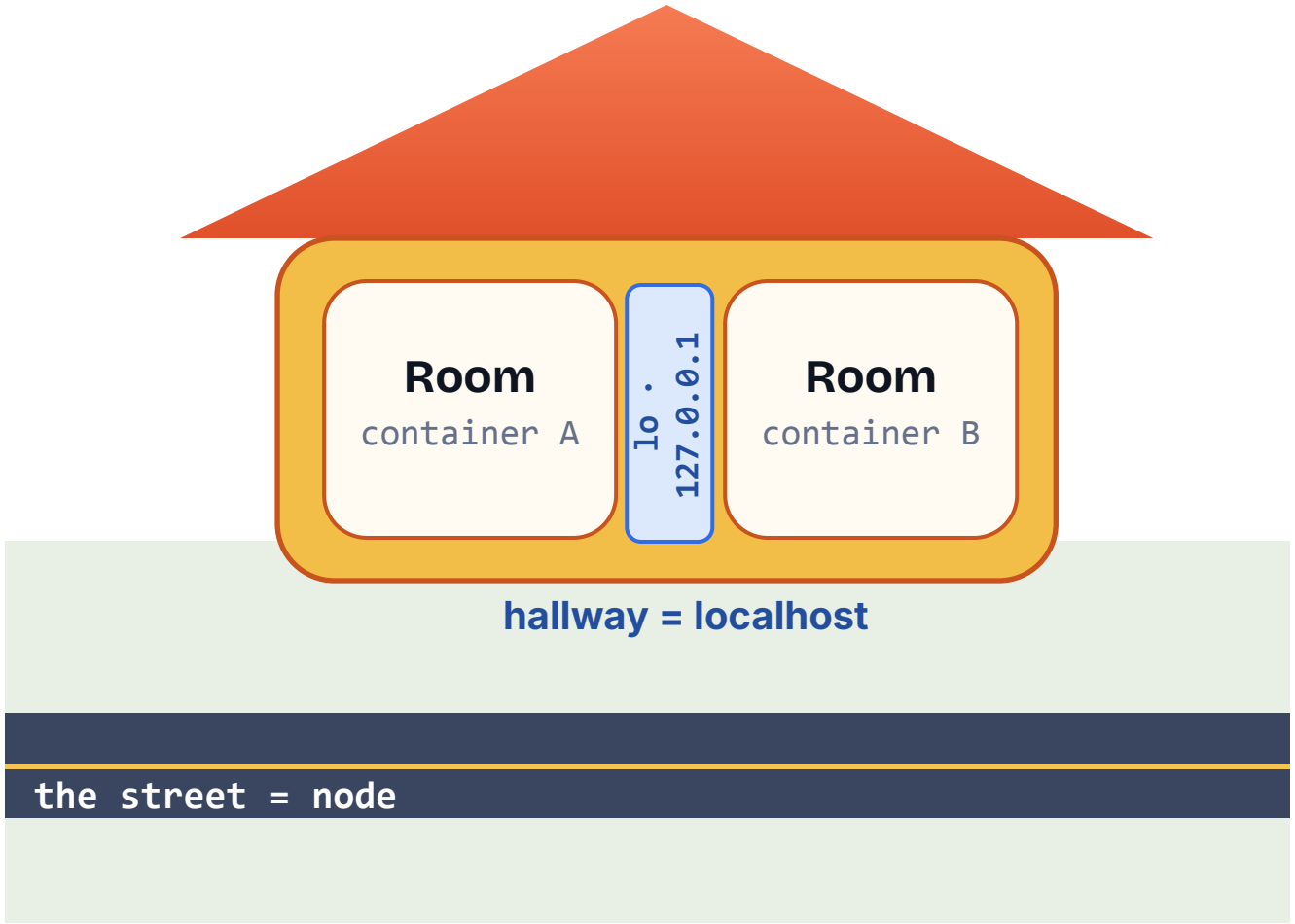
🏠 The whole house

One Pod = One network namespace.

🪜 Hallway = localhost

Sidecars reach the app over lo — traffic loops back, never leaves the house, never touches a veth.

A Pod is a house



🏠 The whole house

One Pod = One network namespace. Every room shares the same IP address.

🪜 Hallway = localhost

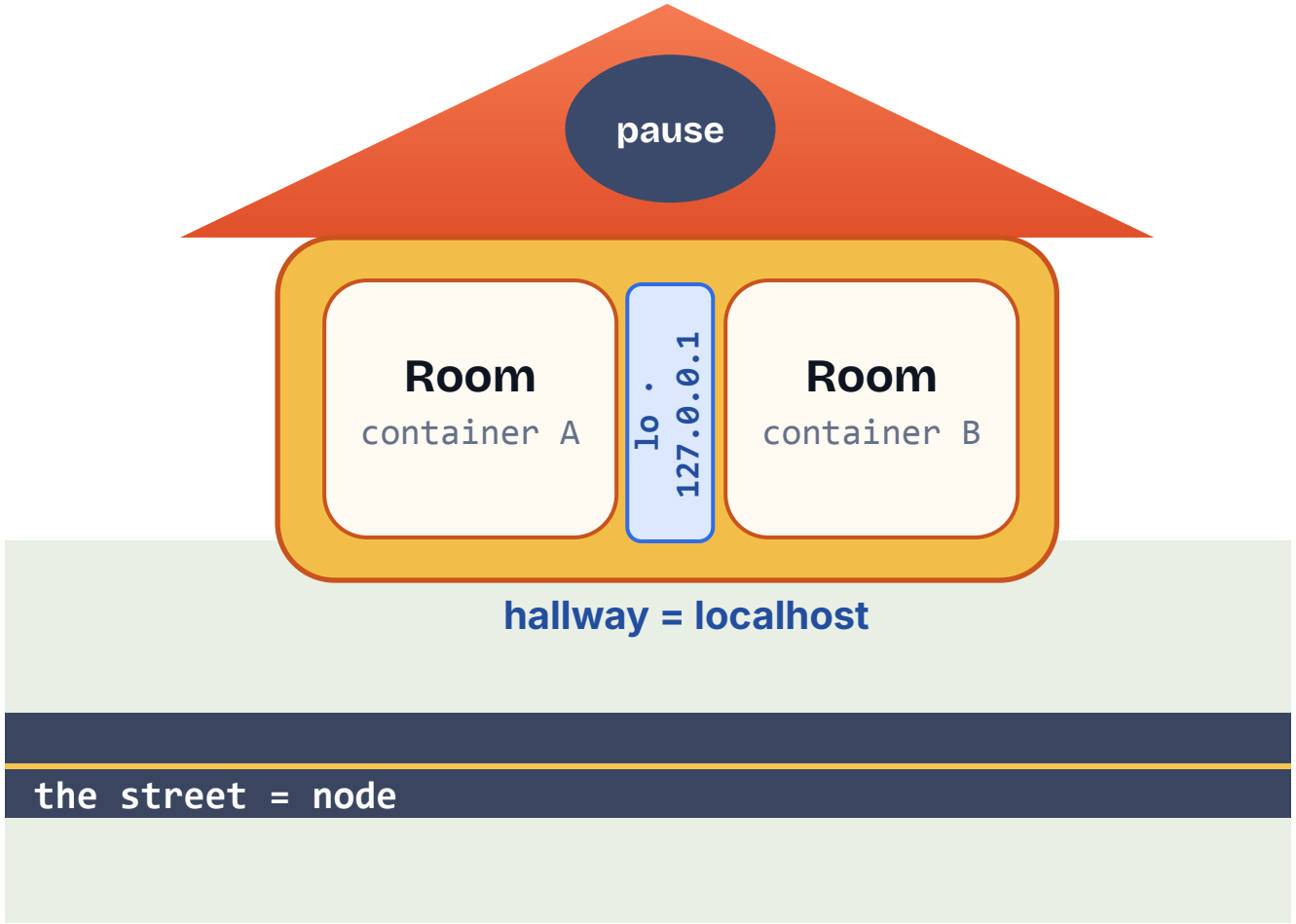
Sidecars reach the app over lo — traffic loops back, never leaves the house, never touches a veth.

Every Pod secretly runs a container you never asked for. What is it?



📱 https://partici.fi/33510940

The caretaker holds the deed



🔑 The caretaker holds the deed

pause = a do-nothing sandbox container, created first, that HOLDS the **netns**. The IP survives container restarts

🔑 What creates pause container?

the kubelet asks the container runtime (via CRI) and the runtime builds the pause container as the Pod's sandbox (pause + netns)

A Pod is a house

`node p2p-worker` live from the node

1 · pause | 2 · veth | 3 · IP·routes | 4 · packet

POD SANDBOX

sandbox —

netns —

attempt —

🏚 no Pod here yet

`kubectl apply -f pod.yaml`

the node
host netns

WHAT'S HAPPENING

waiting for the first event…

surya@fedora:~/kubecon-talk/demo

surya@fedora:~/kubecon-talk/demo | surya@fedora:~/kubecon-talk/demo

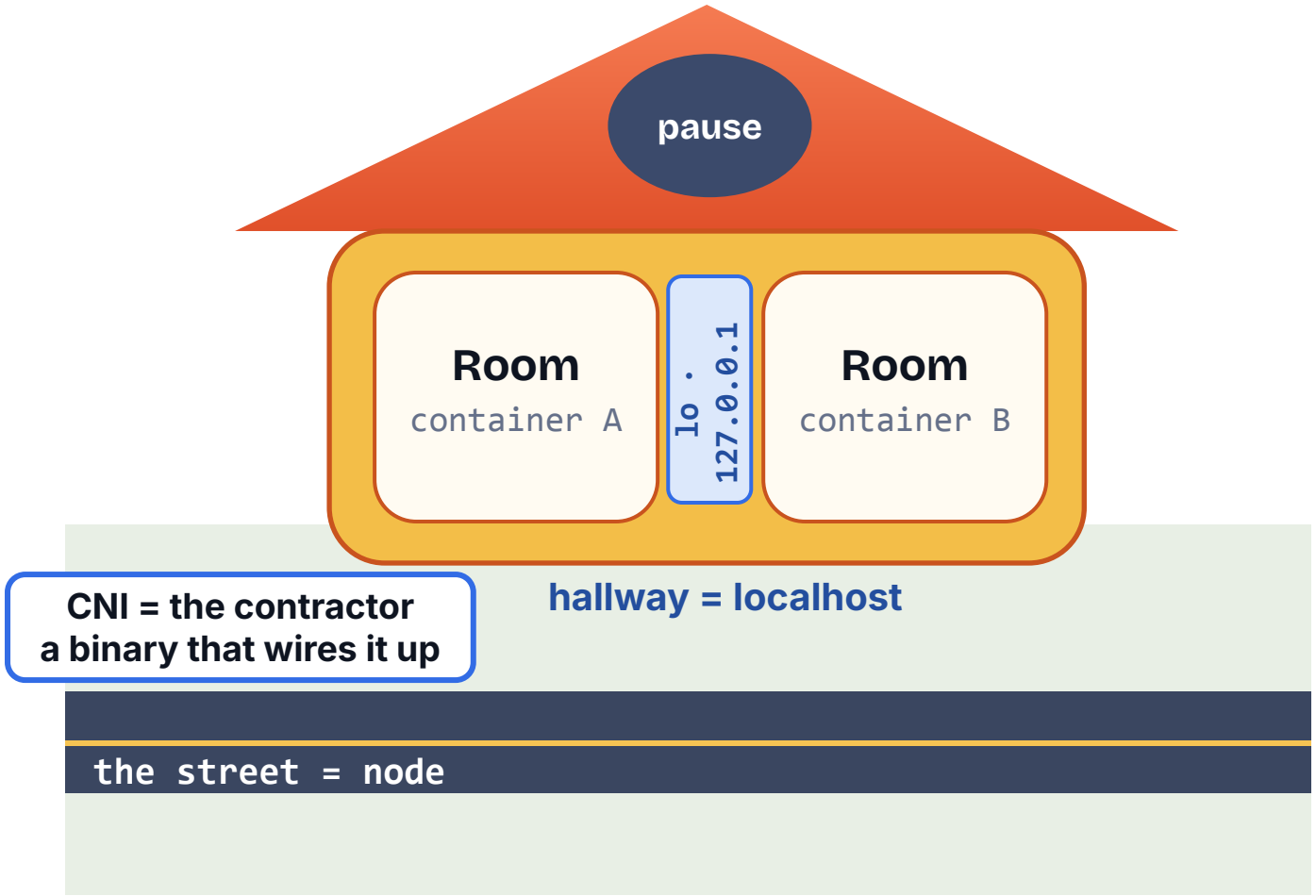
```
surya@fedora:~/kubecon-talk/demo$ ./demo1.sh
```

What is responsible for setting up Pod networking?



📱 https://partici.fi/33510940

CNI: hire the contractor



🛠️ Container Network Interface

Kubernetes doesn't 'do' networking - containerd runs the CNI binary from /opt/cni/bin with JSON on stdin. Here, the CNI is kindnet.

CNIAdd, CNIDel specifications.

🛠️ Examples

- loopback
- **kindnet** (ptp + host-local IPAM)
- cilium (eBPF)
- calico (nftables or eBPF)
- ovn-kubernetes (OVN + OVS)

Is your network plugin ready?

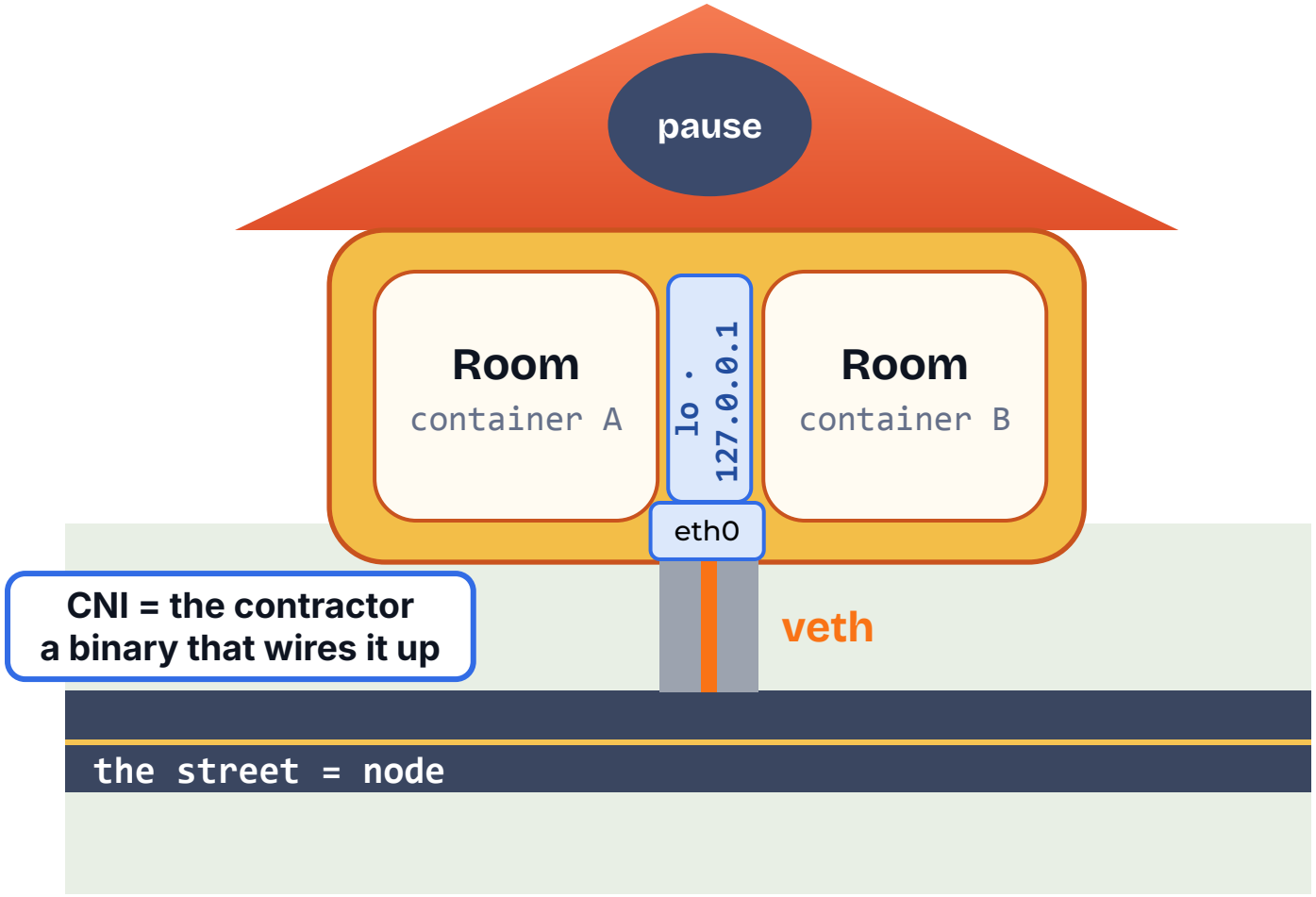
```
$ cat /etc/cni/net.d/10-kindnet.conflist
{
  "cniVersion": "0.3.1",
  "name": "kindnet",
  "plugins": [
    {
      "type": "ptp",             # the veth + routes plugin
      "ipMasq": false,
      "ipam": {
        "type": "host-local",    # hands out the IP
        "dataDir": "/run/cni-ipam-state",
        "routes": [ { "dst": "0.0.0.0/0" } ],
        "ranges": [ [ { "subnet": "10.244.2.0/24" } ] ]
      },
      "mtu": 1500
    },
    {
      "type": "portmap",         # hostPort mappings
      "capabilities": { "portMappings": true }
    }
  ]
}
```

What connects the Pod's network namespace to the host’s network namespace in a typical CNI setup?



📱 https://partici.fi/33510940

The driveway (veth pair)



🚗 The driveway (veth pair)

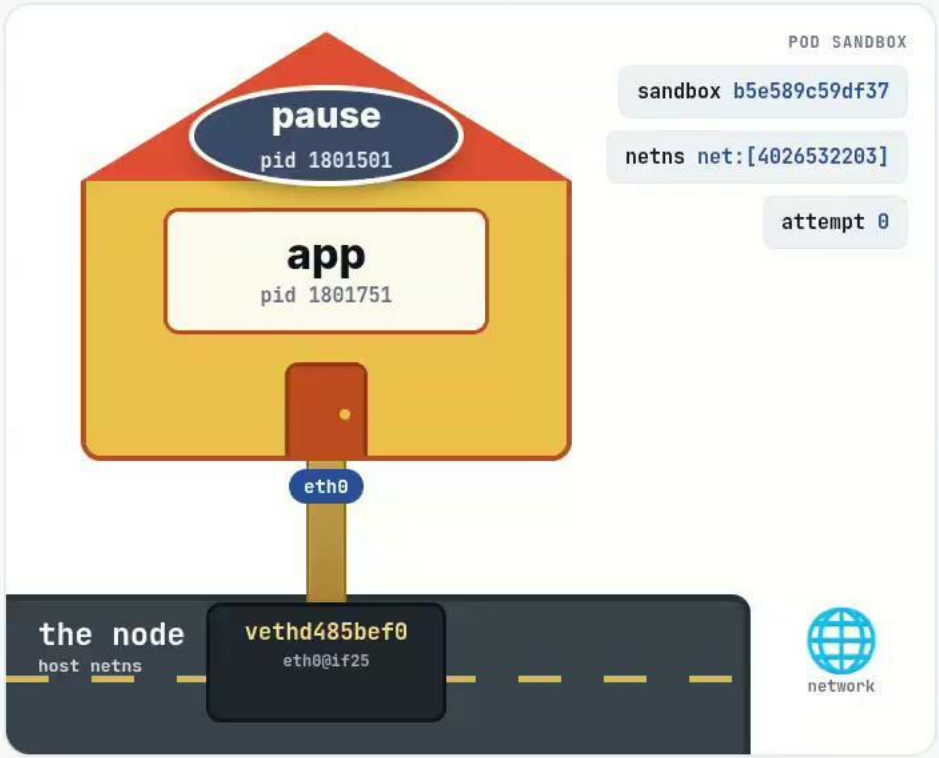
A virtual cable: eth0 inside the Pod ⇄ veth on the host. Point-to-point, no bridge in kindnet. The @ifN indexes match.

A Pod is a house

node p2p-worker

live from the node

1 · pause | 2 · veth | 3 · IP·routes | 4 · packet



WHAT'S HAPPENING

waiting for the first event…

surya@fedora:~/kubecon-talk/demo

surya@fedora:~/kubecon-talk/demo

surya@fedora:~/kubecon-talk/demo

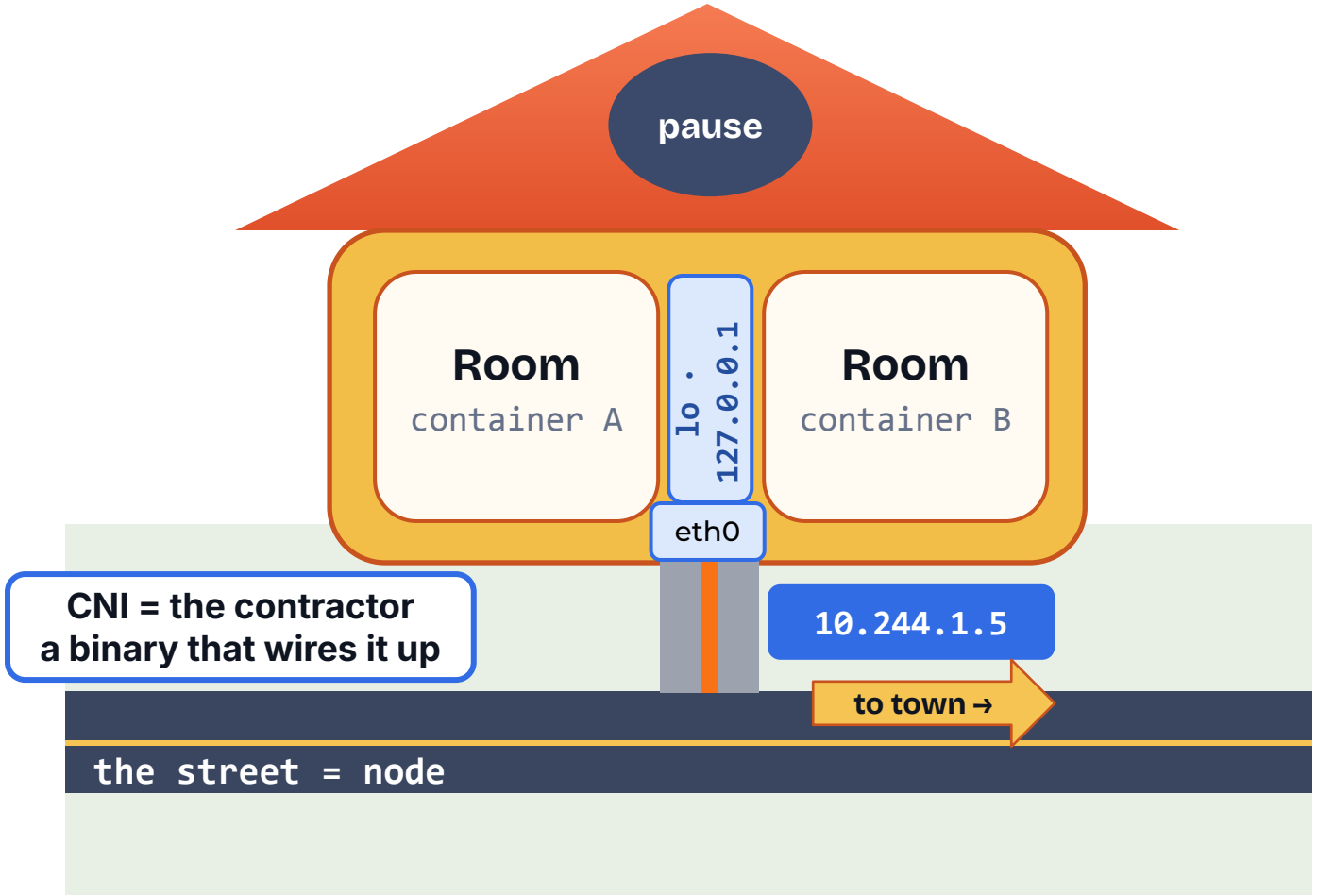
```
surya@fedora:~/kubecon-talk/demo$ ./demo2.sh
```

A Pod has an IP but can't reach anything. What's most likely missing?



📱 https://partici.fi/33510940

Address (IPAM) + signpost (routes)



🏷️ Street address = Pod IP

kindnet does IPAM - hands out the IP from the Pod CIDR. kindnet is running host-local IPAM mode. ptp applies that IP to the pod interface

Default route = the signpost

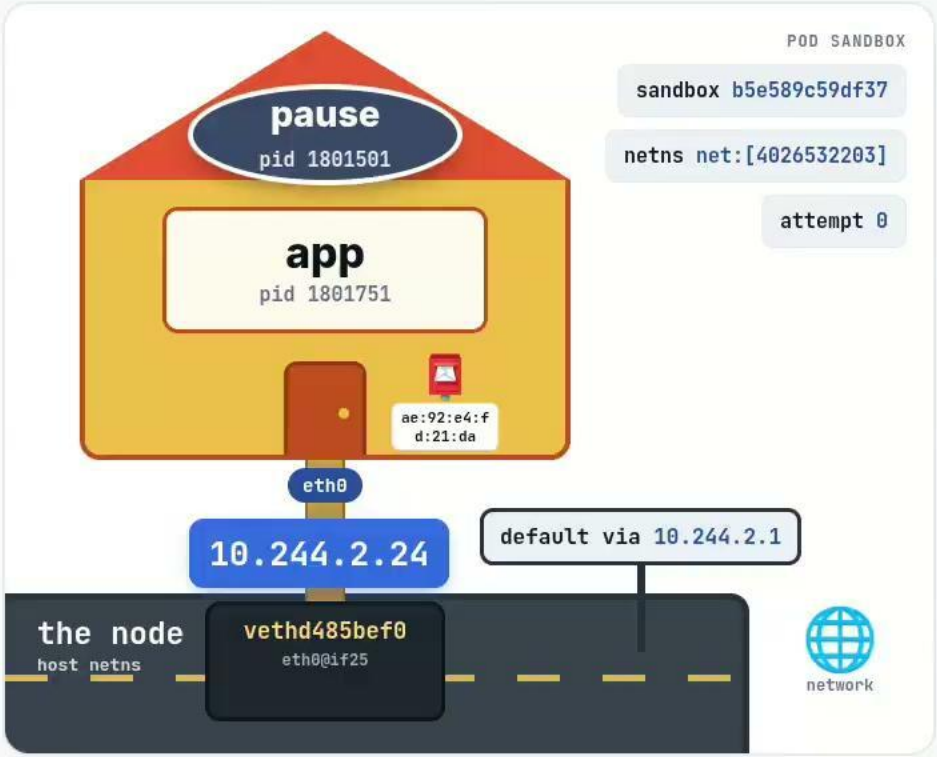
An IP says who you are; the route says how to leave. No default route = goes nowhere.

A Pod is a house

node p2p-worker

● live from the node

1 · pause | 2 · veth | 3 · IP·routes | 4 · packet



WHAT'S HAPPENING

waiting for the first event…

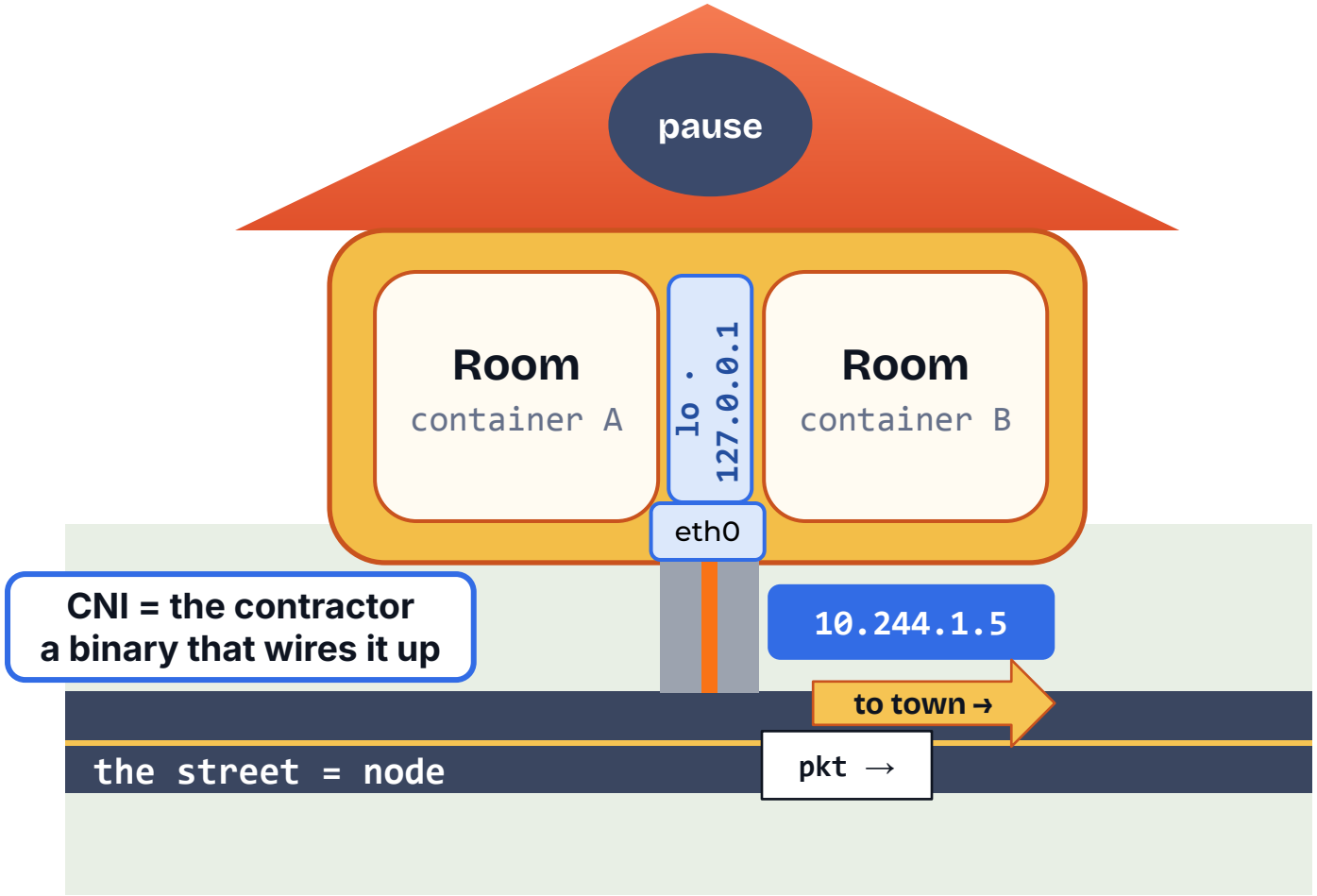
surya@fedora:~/kubecon-talk/demo

surya@fedora:~/kubecon-talk/demo

surya@fedora:~/kubecon-talk/demo

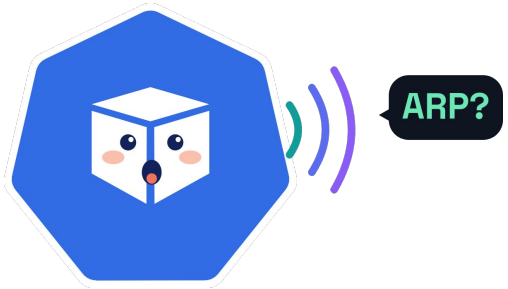
```
surya@fedora:~/kubecon-talk/demo$ ./demo3.sh
```

Mail a letter - who is my gateway?



📣 First packet = ARP

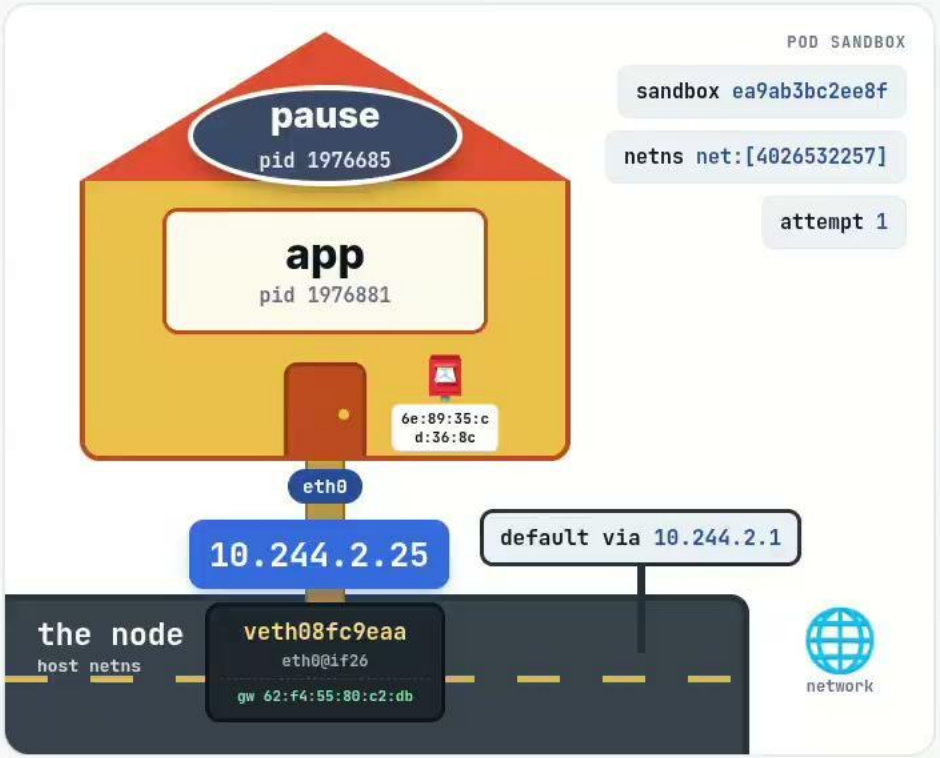
To send anywhere, the Pod first ARPs for its gateway's MAC. IP = the envelope (end-to-end); MAC = which door, changes every hop.



A Pod is a house

`node p2p-worker` · live from the node

1 · pause | 2 · veth | 3 · IP·routes | 4 · packet



WHAT'S HAPPENING

waiting for the first event…

surya@fedora:~/kubecon-talk/demo

surya@fedora:~/kubecon-talk/demo | surya@fedora:~/kubecon-talk/demo

```
surya@fedora:~/kubecon-talk/demo$ ./demo4.sh
```

FULL CIRCLE

The Pod got its IP

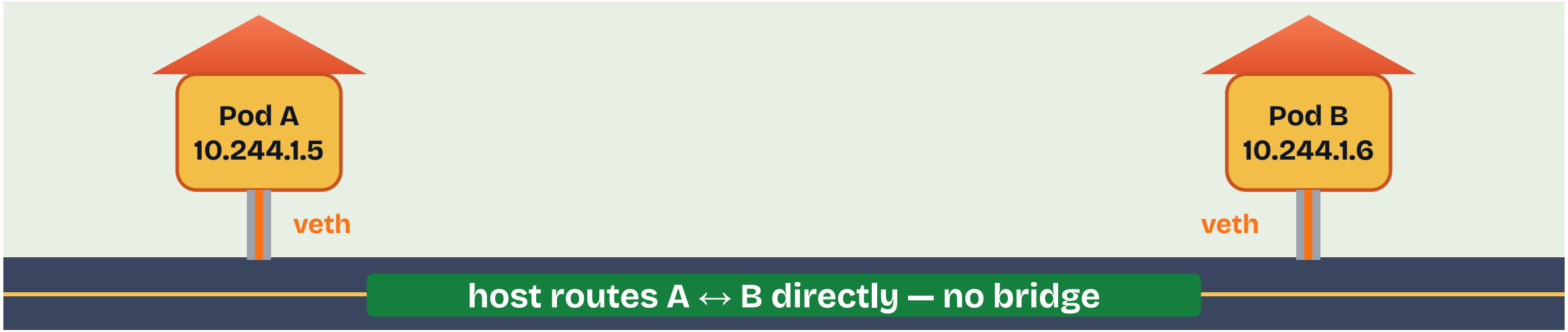


```
$ kubectl get pod demo -o wide

NAME    READY    STATUS     IP             NODE
demo    1/1      Running    10.244.2.25    kind-worker
```

Remember “Running in ~2 seconds”? That's everything we just built - pause · CNI · veth · IP · routes · first packet.

Same street = same node

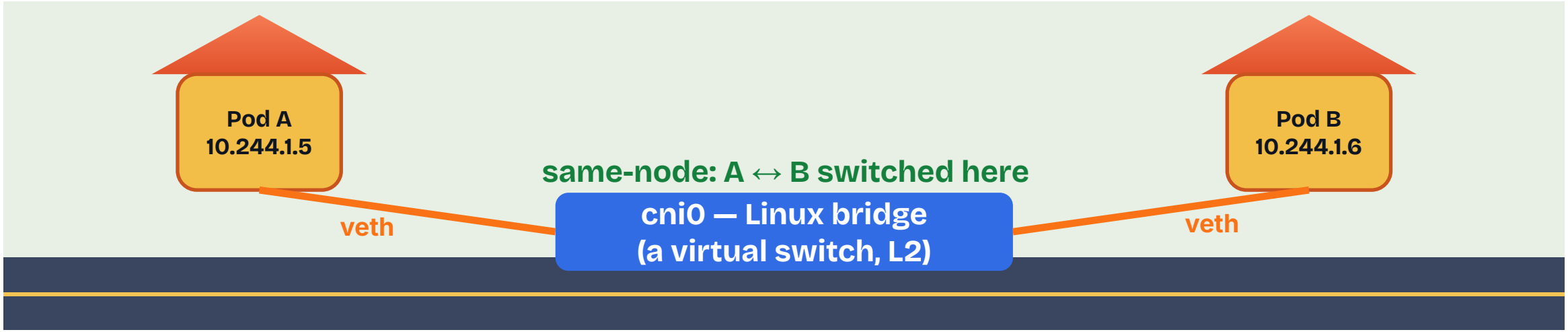


THE STREET = one NODE

kindnet: the host holds a /32 route to each Pod's veth, so it routes A ↔ B directly between the veths.

Point-to-point - NO bridge. (Bridge CNIs would switch them at a bridge instead.)

Same node, other CNIs: bridge



THE STREET = one NODE

Flannel, OVN-Kubernetes, etc. bridge per node / use a virtual switch (Flannel: cni0 · OVN-Kubernetes: OVS br-int); same-node traffic is switched there.

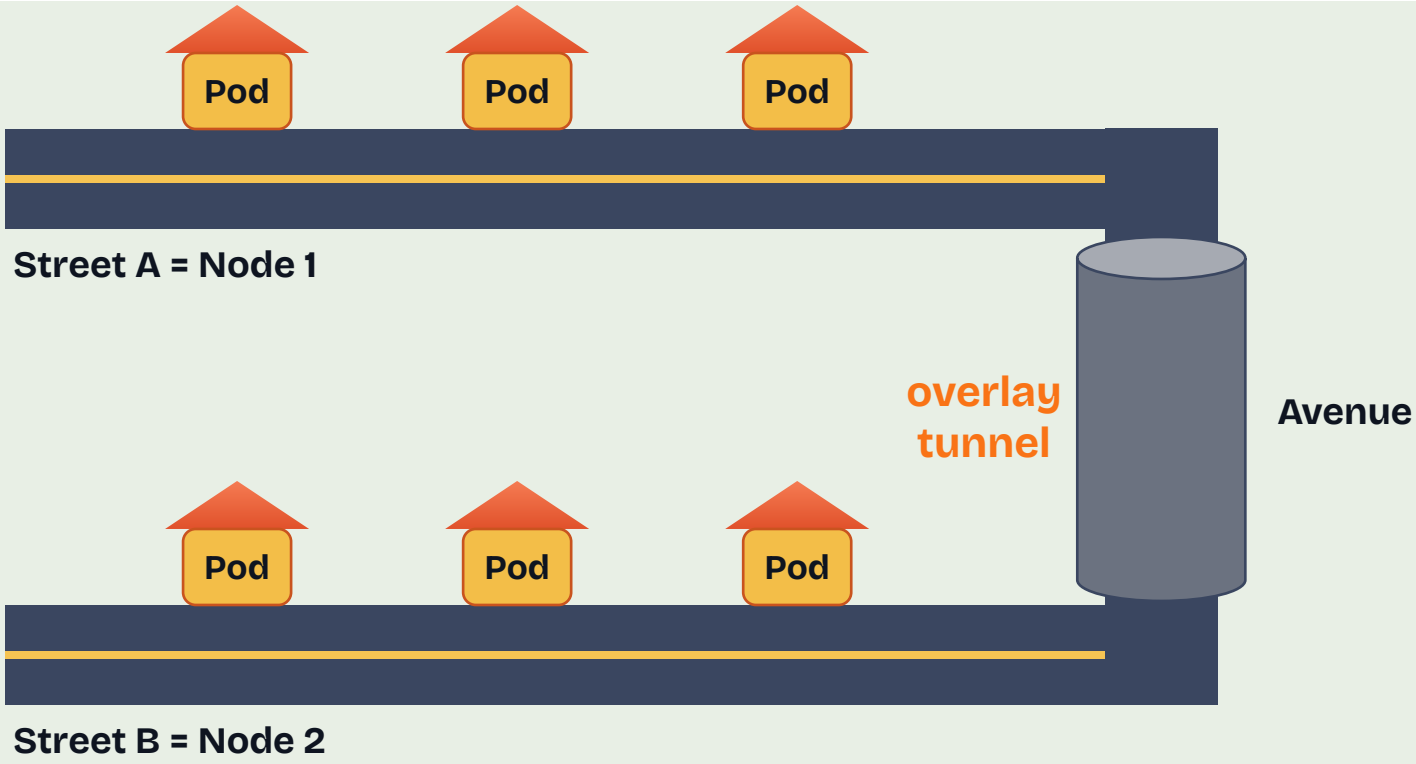
kindnet is point-to-point - NO bridge; it uses host /32 routes between the veths instead.

In a word or two: how does a packet reach a Pod on ANOTHER node?



📱 https://partici.fi/33510940

Crossing to another street



Same street = same node (local). Different streets = cross-node.

Avenue = UNDERLAY

Direct route - kindnet lays plain kernel routes to the other node's PodCIDR via its node IP. Real Pod IPs on the wire. eg. BGP

The tube = OVERLAY

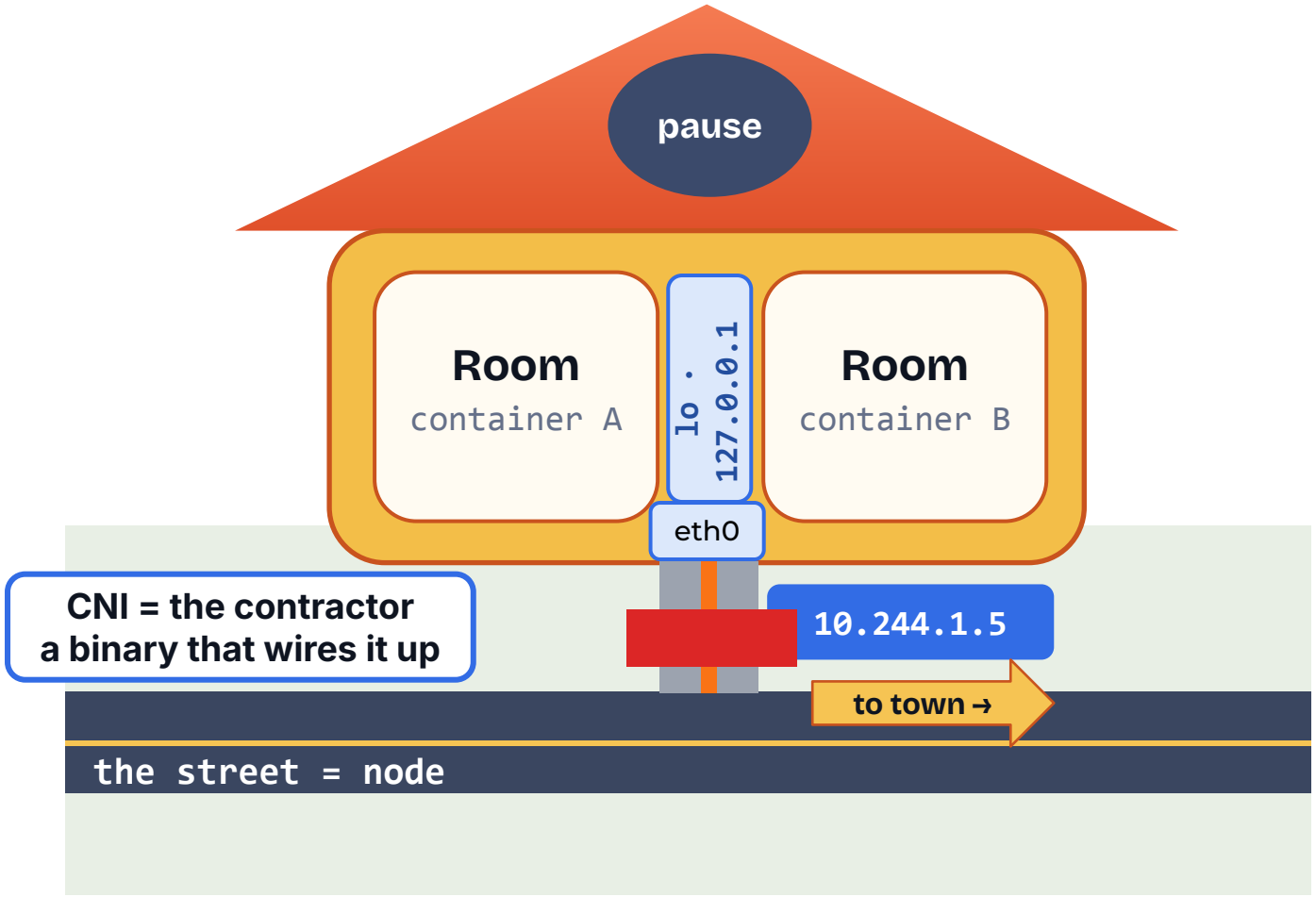
Other CNIs (Flannel / Calico / OVN-Kubernetes) wrap the packet (VXLAN / Geneve) node→node; Pod IPs stay private.

Overlay vs underlay - how different network plugins do it

CNI	Mode	Crossing to another node
kindnet (our demo)	underlay	plain routes to other node's PodCIDR — no wrapping
Flannel (vxlan)	overlay	wraps the packet in VXLAN, Node→Node
Calico	underlay (BGP) / overlay	routes Pod IPs natively, or wraps (IPIP/VXLAN)
Cilium (eBPF)	underlay / overlay	routes Pod IPs natively, or wraps in encap
OVN-Kubernetes	underlay (BGP)/overlay (Geneve)	Geneve tunnels by default; or advertise routes via BGP

The one question every CNI answers: is my Pod IP real on the wire, or does it need an envelope?

Blocked ≠ Broken



🛡️ NetworkPolicy = the gate guard

Apply a default-deny policy and the same ping hangs — but veth, IP and route are all intact. It's not broken, it's BLOCKED.

Your debug checklist:

No IP at all?	→ CNI / IPAM — the contractor never came
Has IP, can't leave?	→ veth or the default route — no driveway / no signpost
Reaches node, not other Pods?	→ cross-node routes (underlay) / overlay / host side forwarding settings on interfaces
Was reachable, now dead, plumbing intact?	→ maybe NetworkPolicy — blocked, not broken needs tcpdump, nft dump, pwru, frr, ipsec etc, MTU?

CNI errors worth recognising

“cni config uninitialized / network plugin not ready”
→ no config in /etc/cni/net.d (CNI pod down) → stuck ContainerCreating

“failed to find plugin … in /opt/cni/bin”
→ CNI binary missing or version mismatch

“failed to allocate IP / no IP addresses available”
→ IPAM pool exhausted (PodCIDR too small)

“failed to set up sandbox … plugin … failed (add)”
→ generic CNI ADD failure; read the network plugin logs

“small pings OK, big transfers hang”
→ MTU mismatch (overlay header not accounted for), PMTUD, fragmentation settings

Can two Pods ever have the SAME IP in the same cluster?



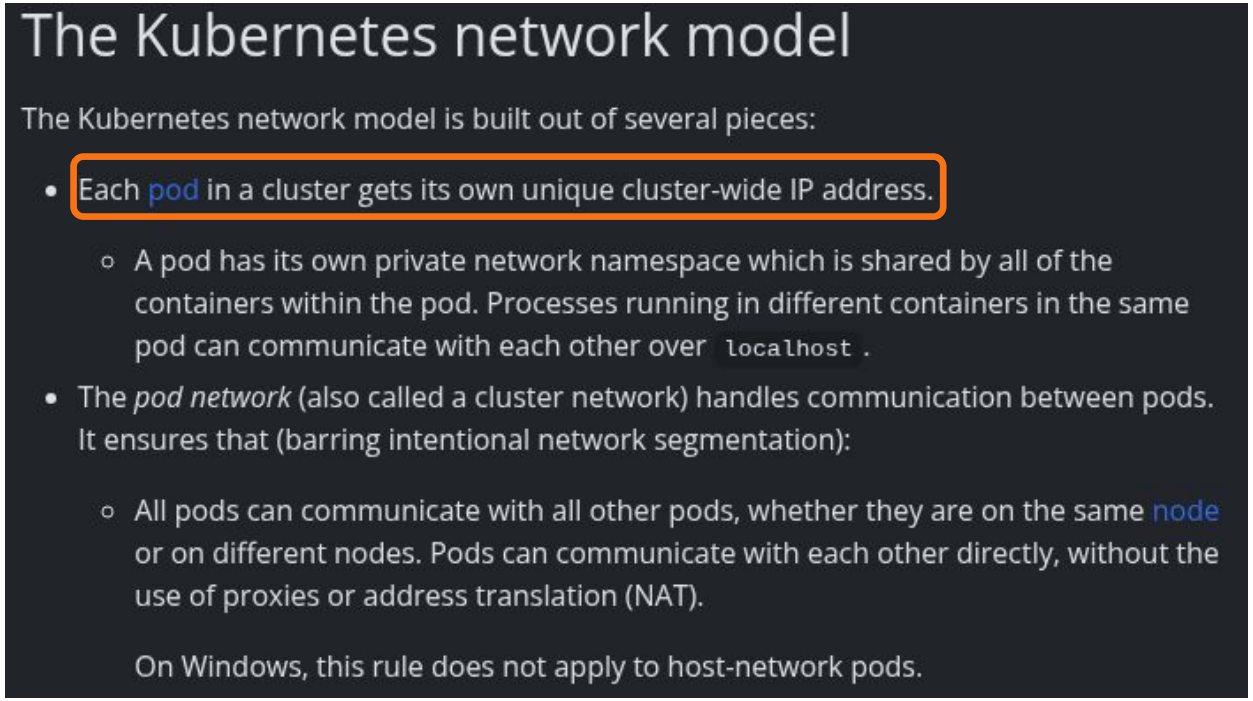
📱 https://partici.fi/33510940

Select every case where a Pod KEEPS its IP:



📱 https://partici.fi/33510940

One IP per Pod? Mostly…



The Kubernetes network model

The Kubernetes network model is built out of several pieces:

- Each pod in a cluster gets its own unique cluster-wide IP address.
  - A pod has its own private network namespace which is shared by all of the containers within the pod. Processes running in different containers in the same pod can communicate with each other over `localhost`.
- The *pod network* (also called a cluster network) handles communication between pods. It ensures that (barring intentional network segmentation):
  - All pods can communicate with all other pods, whether they are on the same node or on different nodes. Pods can communicate with each other directly, without the use of proxies or address translation (NAT).

    On Windows, this rule does not apply to host-network pods.

kubernetes.io/docs/concepts/services-networking

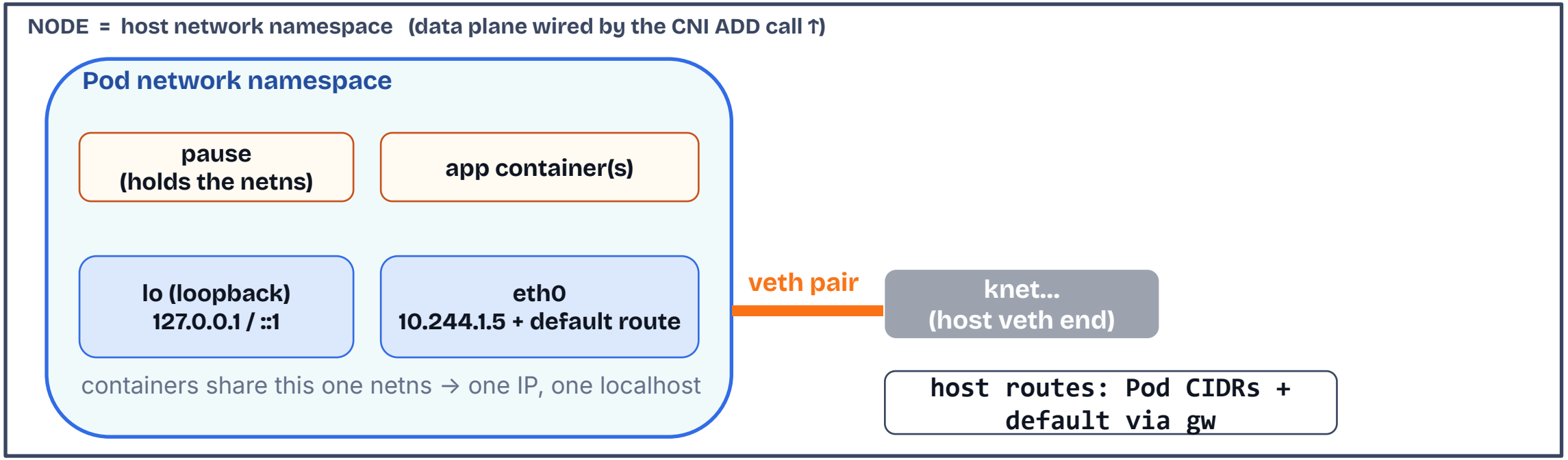
- **…but hostNetwork Pods share the node's IP, so two CAN match**
- Multus: a Pod can have many NICs & IPs - one house, many streets
- KubeVirt: a whole VM runs inside a Pod (VM-as-a-container)
- Live-migrate that VM and OVN-Kubernetes / Calico KEEP its IP - the address outlives the Pod and crosses nodes
- Overlapping IPs
- **The model is a floor, not a ceiling - the ecosystem keeps bending it**

The whole journey — the real stack

kubelet → CRI → container runtime (containerd) → CNI ADD → CNI plugin: veth + IP + routes *

The container runtime creates the sandbox (pause + netns), then calls the CNI (CNI ADD) to wire it ↓

* IPAM can run before / in parallel to veth creation; many CNIs (incl. kindnet) allocate the IP first.



first packet: eth0 → veth → host routing → out the NIC

FYI - KEP-6313 (ALPHA - SUBJECT TO CHANGE) `defaultNetwork: None`

KEP-6313: Network-Isolated Pod Sandboxes

IMPLEMENTATION HISTORY

ALPHA | Implementable

Created	2026-09-02
Latest	v1.38

MILESTONES

Alpha	v1.38
Beta	v1.41
Stable	v1.43

RELATED ENHANCEMENTS

- KEP-5328: Node Declared Features
- KEP-4817: Resource Claim Status With Possible Standardized Network Interface Data

OWNERSHIP

OWNING SIG

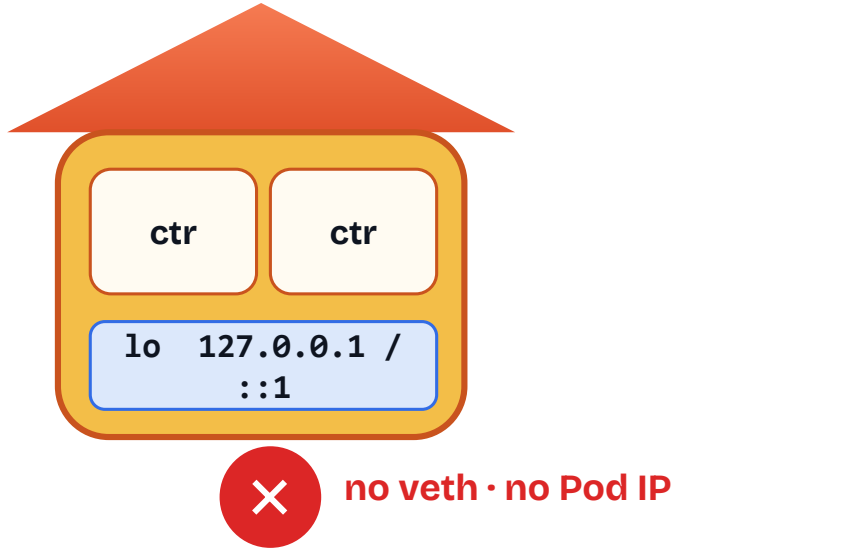
SIG Network

PARTICIPATING SIGS

SIG Network | SIG Node

PRIMARY AUTHORS

@aojea @MikeZappa87

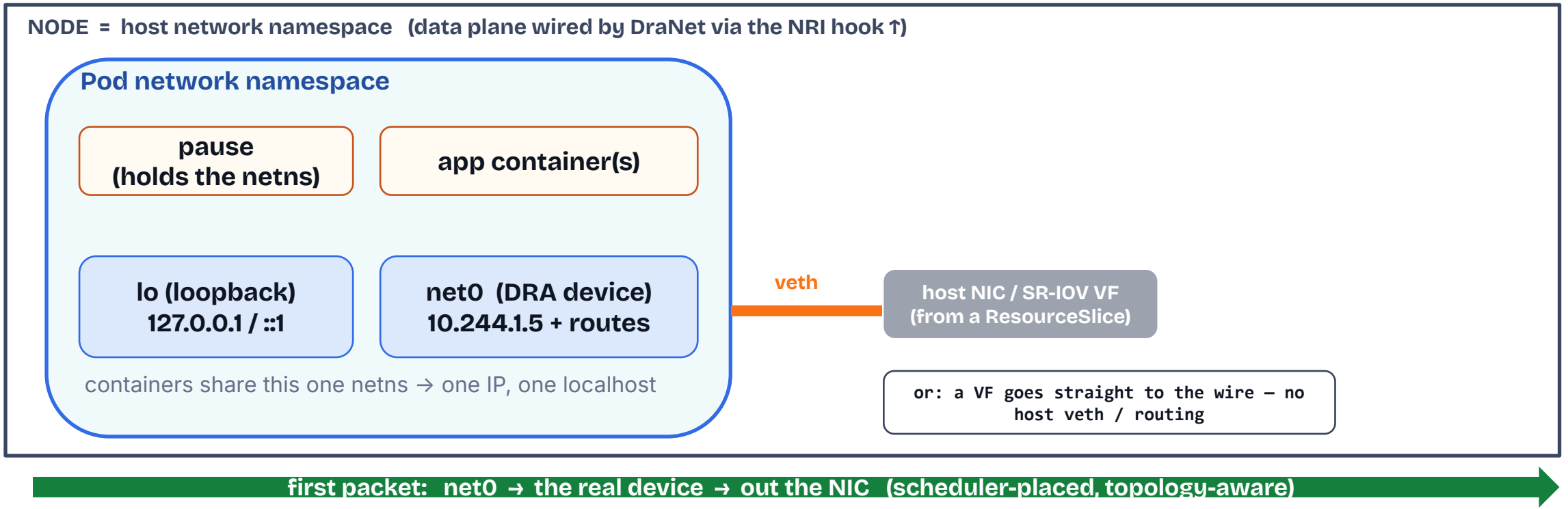


- defaultNetwork: Pod | Host | None (new PodSpec field)
- None = a Pod with ONLY lo; the pod-network CNI ADD is skipped
- For DRA / RDMA fabrics, provable isolation, VM networking

The same stack — without CNI

kubelet → DRA → DraNet (DRA driver): moves netdev + IP ⟷ NRI → container runtime (containerd)

Scheduler + DRA allocate the device first (ResourceClaim ⇄ ResourceSlice); kubelet calls DraNet's NodePrepareResources.
defaultNetwork: None ⇒ NO CNI — the runtime makes the sandbox+netns, then NRI calls DraNet to attach the device ↓

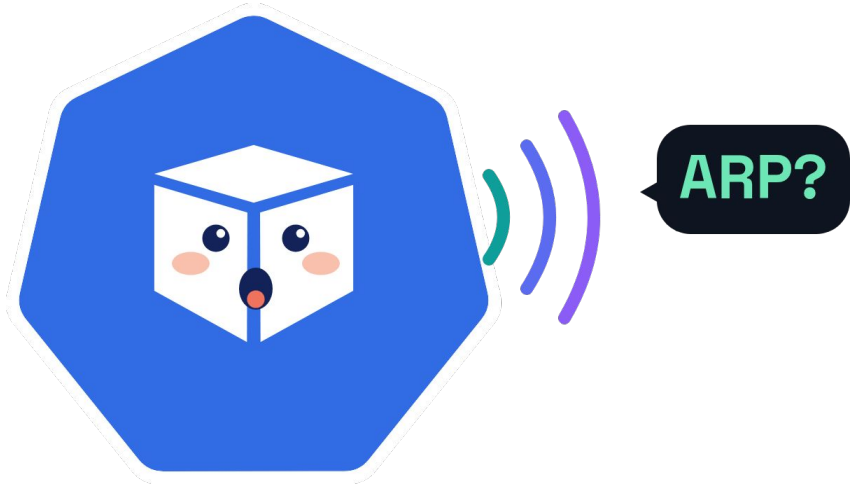


THAT'S THE JOURNEY

Thanks!

Questions?

Pod creation → first packet



From CNI to composable network drivers

The one default network uses CNI. For Pods that need more — many NICs, RDMA, GPUs side-by-side — a scheduler-aware, composable model is emerging.

CRI

Container Runtime Interface. kubelet → runtime: build the sandbox (pause + netns), then start the containers.

CNI

Container Network Interface. runtime → plugin: veth + IP + routes — the one default network we demoed.

DRA

Dynamic Resource Allocation. Request NICs / RDMA / GPUs in the PodSpec; the scheduler places by topology. Core GA 1.34.

NRI

Node Resource Interface. Parallel, context-aware runtime hooks — attach prepared interfaces; they even see the Pod's IPs.

How they fit: DRA picks the node & prepares the device → NRI hooks the runtime to attach it → OCI moves the interface into the Pod's netns.

The emerging Kubernetes Network Driver model (reference driver: DraNet) makes Pod networking composable & scheduler-aware — and with defaultNetwork: None (KEP-6313) a Pod can skip the default CNI entirely. No more serial CNI chaining.

Antonio Ojea et al. · “The Kubernetes Network Driver Model” (arXiv 2506.23628) · Kubernetes SIG-Network

TAKEAWAYS

The journey: Pod creation → first packet

- A Pod is a **house** 🏠
- 1 · **pause** holds the deed (the netns)
- 2 · **CNI**, the contractor, wires it up
- 3 · **veth**, the driveway to the street
- 4 · **IPAM + routes** = address + signpost
- 5 · **cross-node**: underlay vs overlay
- 6. First packet → then: **blocked vs broken** (NetworkPolicy)
- 7 · **common failures**

LIVE DEMO - 2

Demo — the veth match game

LIVE DEMO

```
$ kubectl exec <pod> -- ip -o link show eth0
$ ip -o link show type veth | grep knet
```

👀 **Watch for:** eth0@ifN inside the Pod pairing with knet...@ifM on the host — same index.