Network Virtualization

Lecture 8

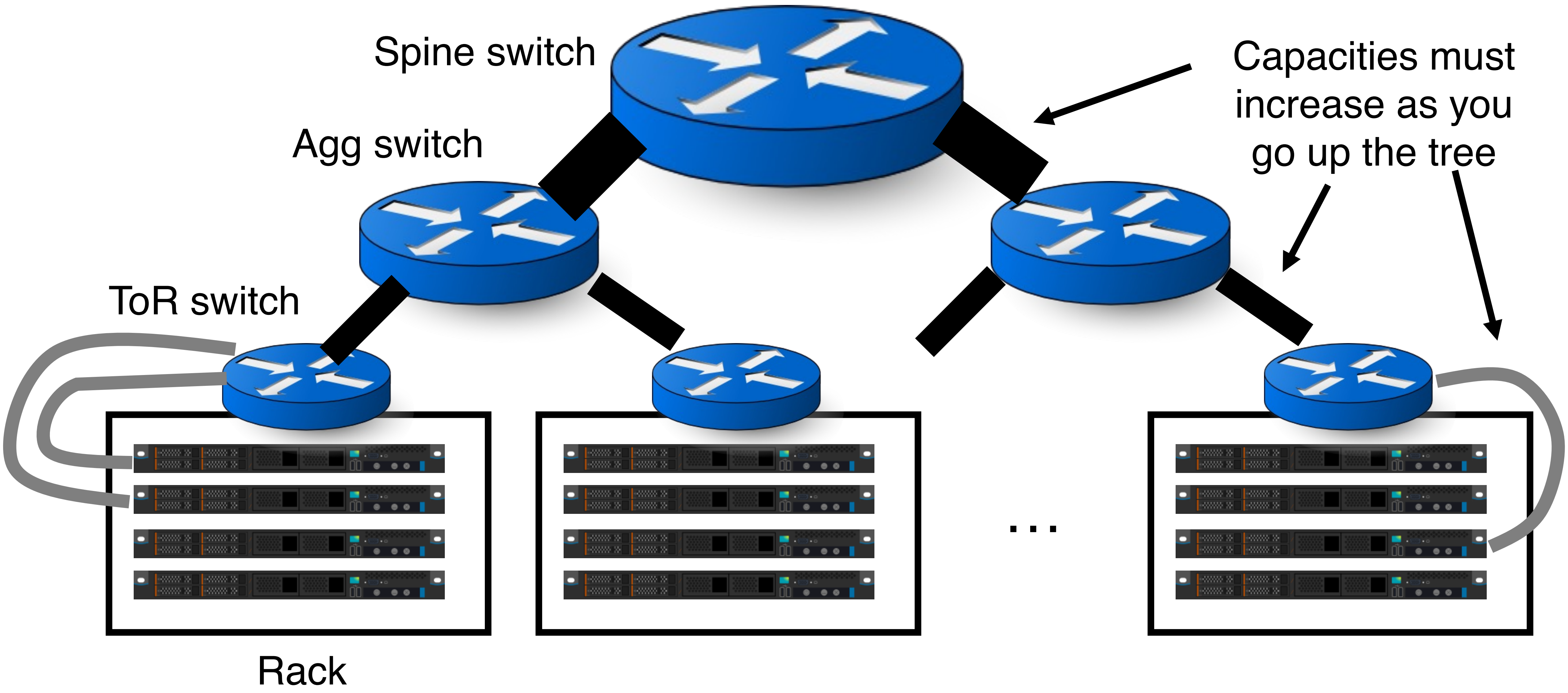
Srinivas Narayana

http://www.cs.rutgers.edu/~sn624/553-S23

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Virtualizing Networking in a Shared Cluster

Typical network structure: Fat Trees



Goals

- Terminology:
  - tenant/customer and provider
  - Virtual NIC (vNIC): network interface exposed with SR-IOV or network namespaces
- (1) Place tenant workloads on any physical machine
- (2) Scale or migrate tenant workload across physical machines at any time
- (3) “Simplify configuration” for everyone involved
  - Views of tenant addresses and interfaces
  - Tenant apps using load balancing, DNS-based IP discovery, etc.
  - Provider’s ability to plumb tenant workloads together
  - Migration from on-premise compute cluster to shared cloud

Design Choice (1): L2 or L3?

- L2: zero configuration, but doesn't scale due to broadcast
  - Support seamless migration within L2 network
  - Broadcast storms: learning switch, ARPs
  - Scale broadcast by isolating physical machines into VLANs. However, this requires configuring physical switch ports. Complicates migration.
- L3: no broadcast, but configuration needed to run routing protocols
  - Configure both endpoints (e.g. gateways) and routers (IP prefixes)
  - Migration is not cheap but doable with a little work
  - e.g., with BGP, we know how to propagate reachability information changing over time in a scalable fashion

Design Choice (2): CA's or PA's?

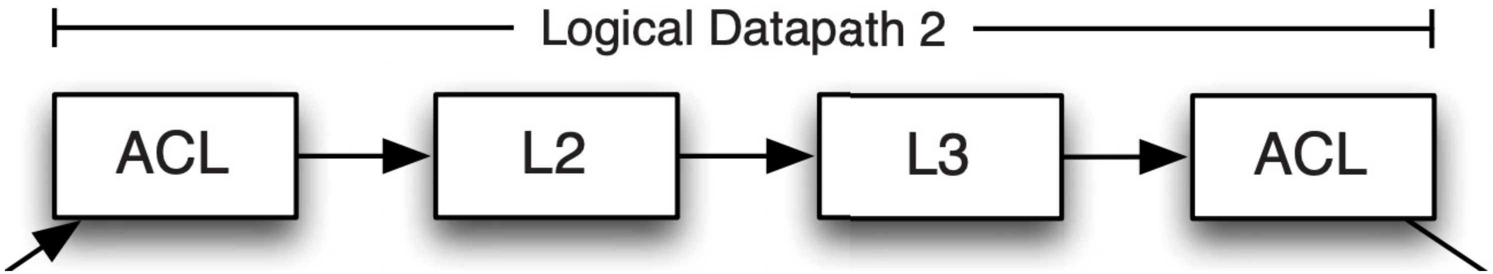
- Do VMs/pods use their own "customer addresses" (CA's) or use the infrastructure's "provider addresses" (PA's)?
- PA's: supporting routing is "business as usual"
  - But one tenant's ports affected by other tenants on same machine
  - Need static allocation of ports to tenants, or dynamic port discovery
  - Reduced isolation, more complex configuration, app changes
- CA's: dedicated IP per VM/pod, visible to applications
  - Clean and backwards compatible. e.g. DNS
  - If VM/pod A sees its own address to be X, any VM/pod B talking to A also thinks that A has address X. A is reachable with CA address X.
  - However, need to design networking to route between CA's,
  - Example: migrate VMs/pods across PA's with unchanging CA

Networking in a multi-tenant data center

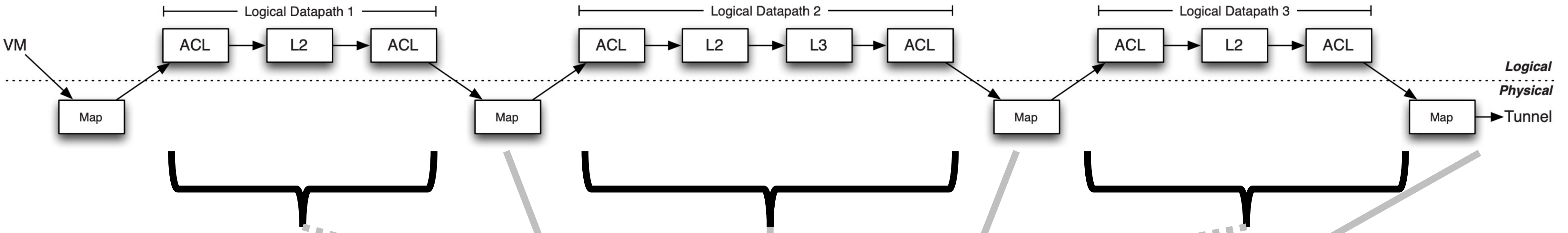
- Address virtualization: VMs/pods use own addresses (CA's)
  - Physical network does not know how to route CA's
  - Additional software to translate CA's between PA's: Tunneling
  - Tunneling endpoint (TEP): software tun/tap interface, NIC hardware, or software switch in a hypervisor. Overlay.
  - TEP encapsulates and decapsulates packet headers (VXLAN, GRE)
- Topology virtualization: Tenants should be able to bring own custom network topologies or assume "one big switch"
  - Facilitate migration into public cloud, consistent view for tenant's monitoring and maintenance tools, etc.
- Supporting virtualized service models
  - e.g. rate limits and isolation across tenants sharing a physical machine

Example 1: Nicira Virtualization Platform

- NVP: Motivated by migration of on-premise cloud workloads as seamlessly as possible to cloud
- Address virtualization: VM's see and use CA's
- Topology virtualization (bring your own topology)
  - packets processed through logical switch/router tenant topology
  - Tables populated by classic routing protocols (e.g. OSPF, BGP)
- Edge: logical datapaths and TEPs (vNIC → hypervisor OVS)
- Network core is a simple pipe that routes between TEPs



Topology and Address Virtualization



Topology virtualization

Address virtualization (separation of tenant and provider addresses through tunneling)

Effect: Move packets from source vNIC to dest vNIC

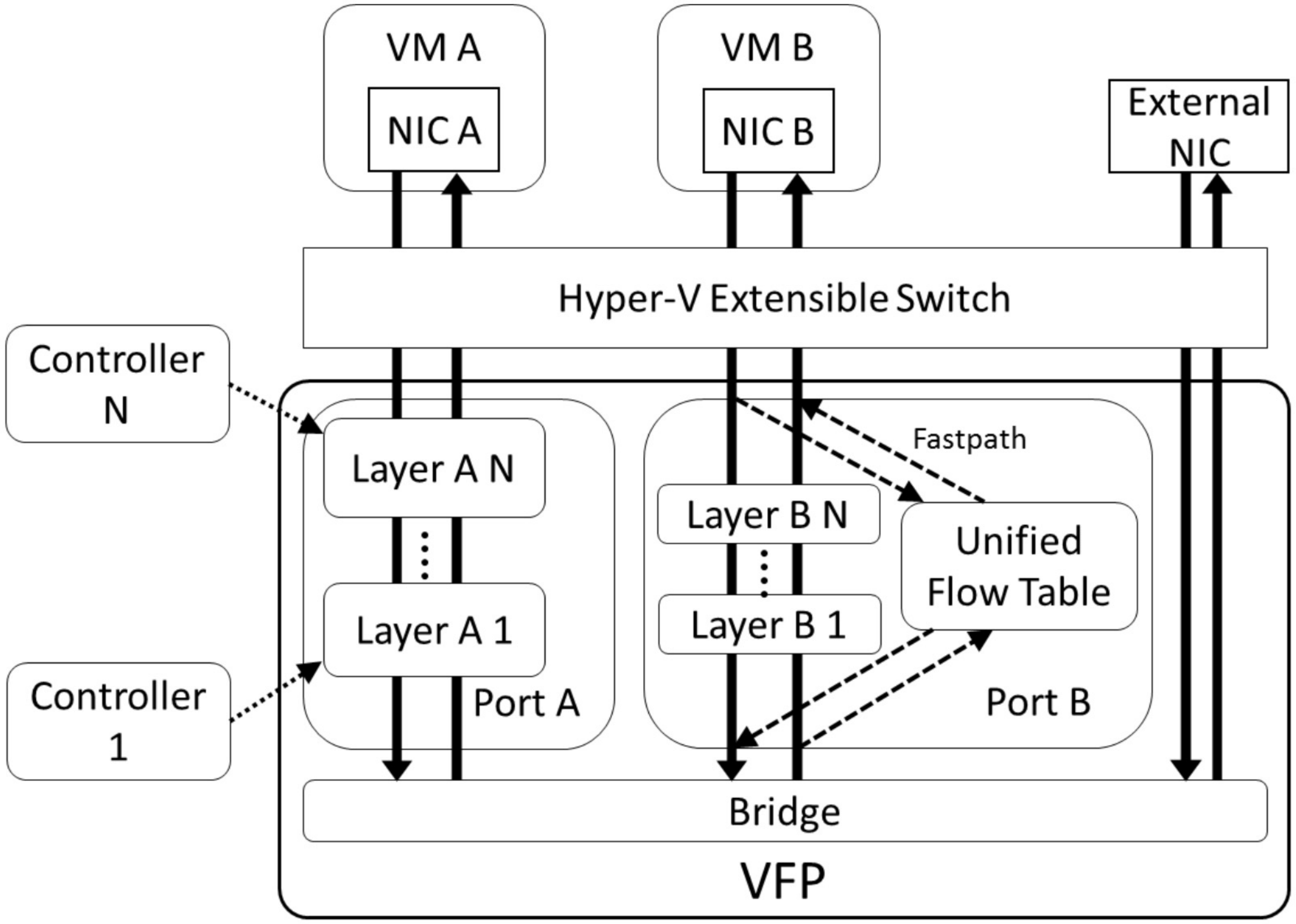
Performance: Caching

SDN in NVP: Controller design

- Declarative design: language to specify tuples of rules/relations
  - No need to implement a state machine to transition rule sets
  - Use a compiler to emit correct, up to date logical datapaths (tuples)
- Shared-nothing parallelism to scale
  - Different logical datapaths easily distributed
  - “Template” rules output from logical datapaths may be independently specialized to specific hypervisors and VMs
- Controller availability maintained using standard leader election mechanisms
- Control and data paths fail independently
  - Existing OVS hypervisor rules can process packets even if controller fails
  - Fast failover through precomputed failover installed in the data path

Example 2: Azure VFP

- Tenants use CA-space addresses
- One big switch
- Multiple controllers, each programming distinct layer(s)
- Layer implements a part of the policy: NAT, etc.
- The TEP itself is a MAT
- Stateful actions (e.g. NAT) are first-class citizens
- Unified flow tables (caching)

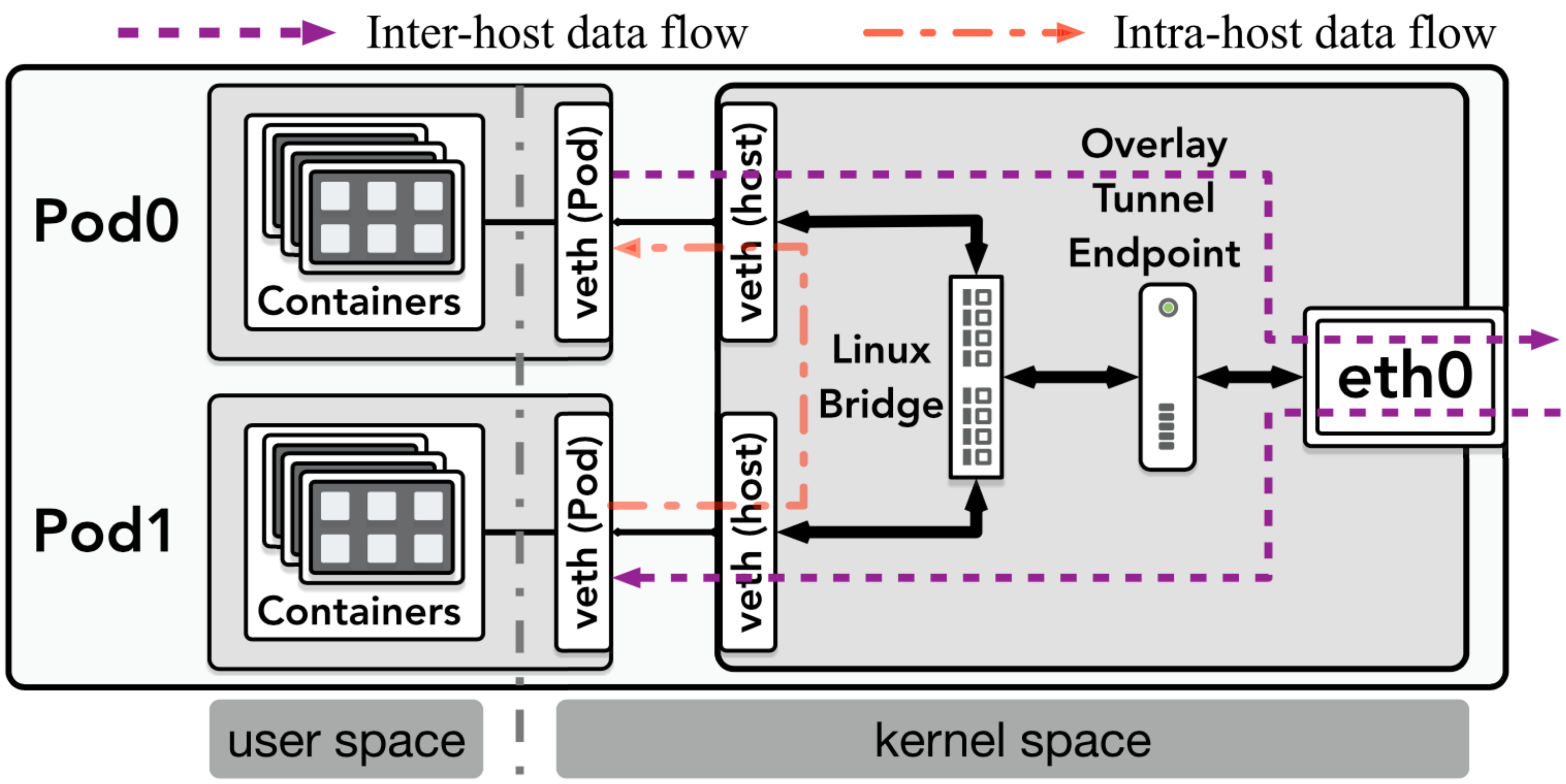


Example 3: Kubernetes/CNIs

- Container Network Interfaces: configuring networking for inter-pod networking
  - Within a pod, use loopback interface (e.g. service mesh)
- Pods use CA-space addresses (overlay); but PA also possible (underlay)
- Topology virtualization: If CA, TEP configured through
  - In-kernel forwarding (L3 forwarding tables, netfilter, iptables)
  - Bridging
  - Tun/tap software interface
  - eBPF
- Can use either L2 or L3 networking to interconnect CAs

Example 3: Kubernetes/CNIs

- Example with L2+L3 overlay



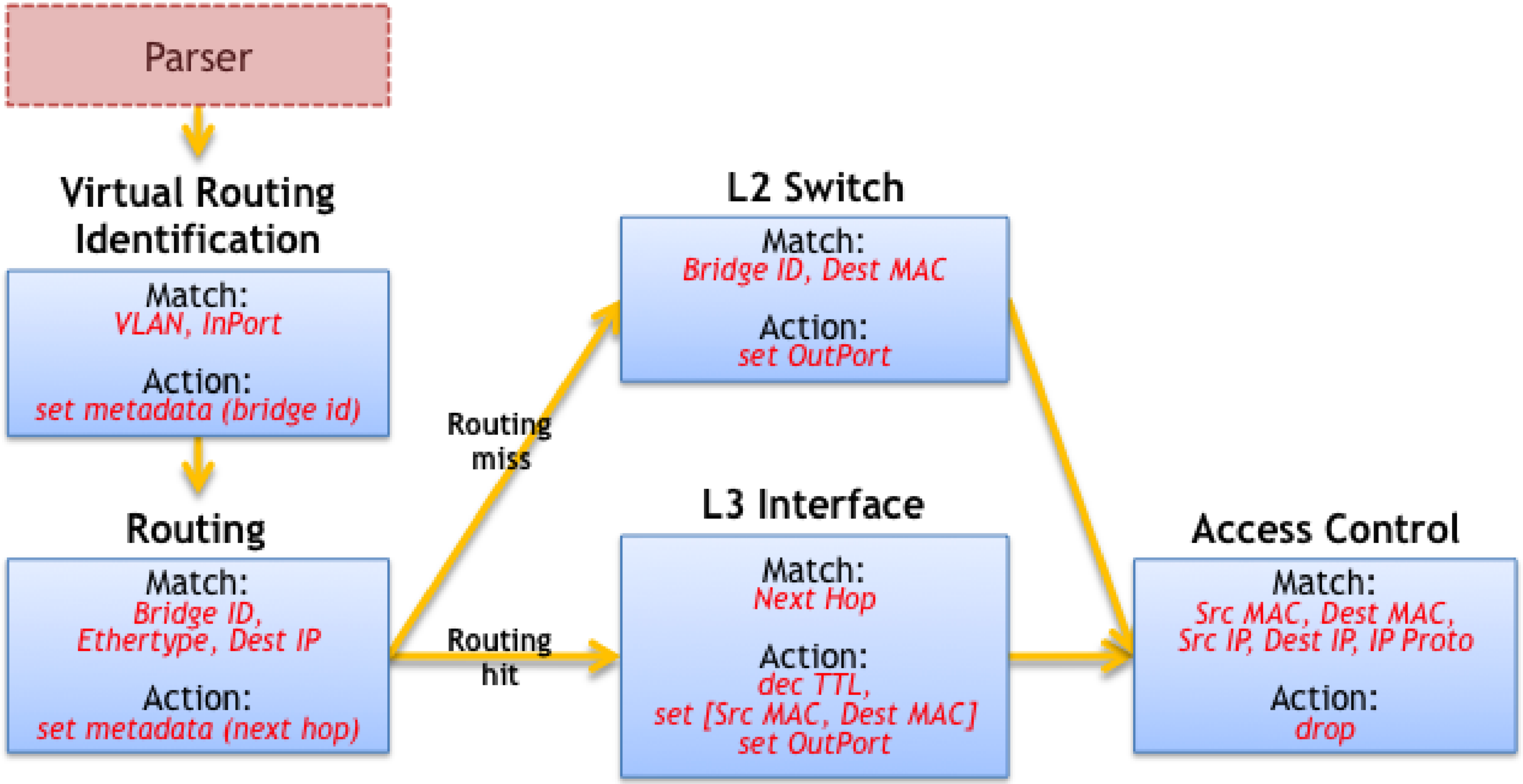
Making old software use new networks usually means making new networks behave like old ones.

Making forwarding more programmable

- Proliferation of fixed table types

Version	Date	Header Fields
OF 1.0	Dec 2009	12 fields (Ethernet, TCP/IPv4)
OF 1.1	Feb 2011	15 fields (MPLS, inter-table metadata)
OF 1.2	Dec 2011	36 fields (ARP, ICMP, IPv6, etc.)
OF 1.3	Jun 2012	40 fields
OF 1.4	Oct 2013	41 fields

P4: Flexible Parsing & Table Dependencies



Packet Header Structure Specification

```
header ethernet {
    fields {
        dst_addr : 48; // width in bits
        src_addr : 48;
        ethertype : 16;
    }
}

header vlan {
    fields {
        pcp : 3;
        cfi : 1;
        vid : 12;
        ethertype : 16;
    }
}
```

```
parser ethernet {
    switch(ethertype) {
        case 0x8100: vlan;
        case 0x9100: vlan;
        case  0x800: ipv4;
        // Other cases
    }
}

parser vlan {
    switch(ethertype) {
        case 0xaaaa: mTag;
        case  0x800: ipv4;
        // Other cases
    }
}
```