

# Network Virtualization

Lecture 6

Srinivas Narayana

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

# Building Blocks of Containers

# What goes into a container?

- Not a natively hardware-supported abstraction like privilege rings, which enable OSes (and virtual machines)
- Instead, use software mechanisms built into OS kernels
- Containers: a loose conglomeration of kernel-level mechanisms
  - Access isolation of global resources (**namespaces**)
  - Resource/Performance isolation of global resources (**control groups**)
  - Sharing data on filesystem for efficiency (**union filesystems**)
    - (need to add on isolation of unique data)
  - Security mechanisms: appArmor, capabilities
- Kludgy, but essential since hard to get it right from scratch

# Namespaces

- **Access isolation**
- Show an instance of a global resource as available to all processes inside a namespace
- Changes visible to other processes within namespace
  - But invisible outside the namespace
- Show different “copies” of resources associated with the kind of namespace
  - Network, IPC, mount, PID, ...
- Every process starts in init namespace, change with `setns`
- Network: (software/hardware) network device; routing rules; port numbers. `veth` pair connects two network namespaces

# Control groups

- **Resource/Performance isolation**
- **Subsystem: a specific kind of resource**
  - CPU time, memory, network bandwidth, block device access, priority, CPU and memory (numa) node assignment. Many configurable parameters per subsystem
- **Control group or cgroup: a set of processes**
  - If fork(), inherit a bunch of attributes including parent's cgroup
- **Hierarchy: a tree where each node is a cgroup**
  - Many hierarchies can exist, unlike process hierarchy
- **Each subsystem “mounted” onto one hierarchy**
  - Possible to use a single hierarchy for multiple subsystems (resources)
- **Every process has exactly one reservation per resource**

# Union FS: ~~“container images too big”~~

- Directory structures on disk are typically “mounted” at some point in the virtual filesystem (/, /home/users/name, etc.)
- Processes in containers want mostly the same files, with a small number of modifications per process or container
  - Think: common third-party packages and shared library images
  - (while supporting the need for distinct libraries/versions across containers)
  - Similar use cases in the past: data on a read-only medium which needed a small number of updates and refresh into new medium
- **Union filesystem:** maintain a stack of filesystems at each mount point. Only the latest one is writable. Lower layers are read-only.
- Write fresh to the top. Update by **copy up**. Deletion requires a special mechanism to record a file that isn't there (whiteout). Cache heavily.
- Virtual Filesystem layer accomplishes this with minimal changes to underlying filesystem.

# Orchestrating Containers

# Components you need?

- The machines (nodes), pods (container-ish), images
- **Controllers** and mechanization (“choreography”)
  - Provisioning pods and nodes with desired resources (**kube-scheduler + kubelet**)
  - Replicating according to system metrics or demand (autoscaling controller)
  - Detecting and reacting to failures (replicaSet controller)
- Maintaining the cluster’s desired and observed state
  - Persistent data store (**etcd**; consensus protocol -- RAFT)
  - How should everyone see and access this? **api server** (versioning, etc.)
- Desired state: **declaratively** specified. **Label selectors** to group.
  - ... even when we say kubectl do this and that
- Naming and connecting to remote entities
  - Pods shouldn’t have to know physical addresses; IP address management for applications connecting from within container network namespaces
  - Routing between nodes; within a node from/to a pod on the node
  - **Container Network Interface**



# Network Virtualization

# Virtualizing Networking on a Single Machine

# How to virtualize I/O?

- How device I/O works in general:
  - Registers. Interrupts and polling. Shared memory. DMA.
- Full virtualization: trap and emulate any I/O data operation
  - e.g., moving each byte of guest data through VMM memory is too expensive (not a “zero copy” solution)
- Xen’s initial approach (SOSP’03)
  - Descriptor rings: async I/O over memory shared between hypervisor & guest
- Hypervisor responsibilities for virtualization:
  - Validate data pages pointed from guest-enqueued descriptors
  - Remap data pages (avoid time-of-check to time-of-use), even if not copy
  - (incoming) find which VM to signal? Send event notification to guest OS
- Hypervisor intervention to check every descriptor is bad for perf

# Direct I/O

- Key idea: enable memory protections in hardware
  - Processors have a memory management unit (MMU) for segmentation and paging related memory protections
- IOMMU: a hardware page table to translate and validate addresses for I/O accesses through DMA (on processor)
- Device features to also support:
  - Separate interrupts going to each guest (rather than shared across guests or with hypervisor)
  - Separate address space identifier (“tagged” page table) per guest
  - Sensitive registers on device cannot be controlled directly by guest
  - Must classify incoming packets at NIC to the appropriate guest’s memory region
  - But do all this while allowing device to maintain all this info

# Single Root I/O virtualization (SR-IOV)

- PCIe capability: expose a single physical device (“physical function”) as multiple virtual devices (“virtual functions”), dedicated to each guest OS
- Layer-2 switching to classify packets to each guest’s VF
  - Corresponding IOMMU, interrupts invoked
- Mainly performance-critical registers replicated per VF
- Share “insensitive” per-device resources across guests
  - MAC/PHY logic, packet classification logic
- Sensitive configuration or register access mediated through a driver component in domain0

# Single-Root IO Virtualization (SR-IOV)

- Exposing device as dedicated device provides several benefits
- Configure, monitor, view using “typical” tools (iproute2, wireshark, ...)
- VFs are now “Virtual NICs” (vNICs) with their own MAC address, IP Address, ingress/egress forwarding rules, etc.
  - Decoupled from corresponding physical device parameters

How about virtualizing  
networking across a  
shared compute cluster?

A detour...

# Software/hardware layering at hosts



Communication functions broken up and “stacked”

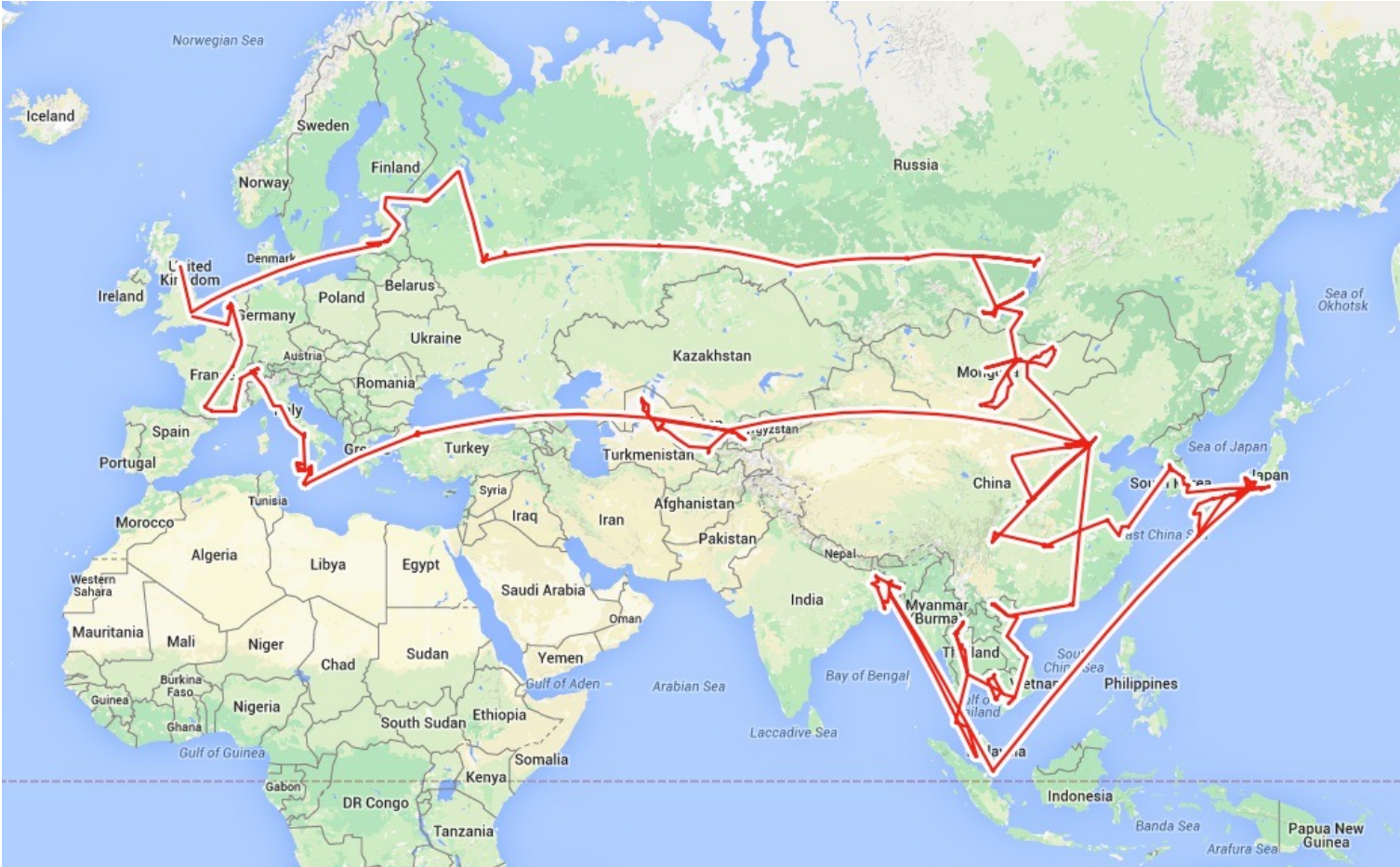
Each layer depends on the one below it.

Each layer supports the one above it.

The interfaces between layers are well-defined and standardized.



# Routing

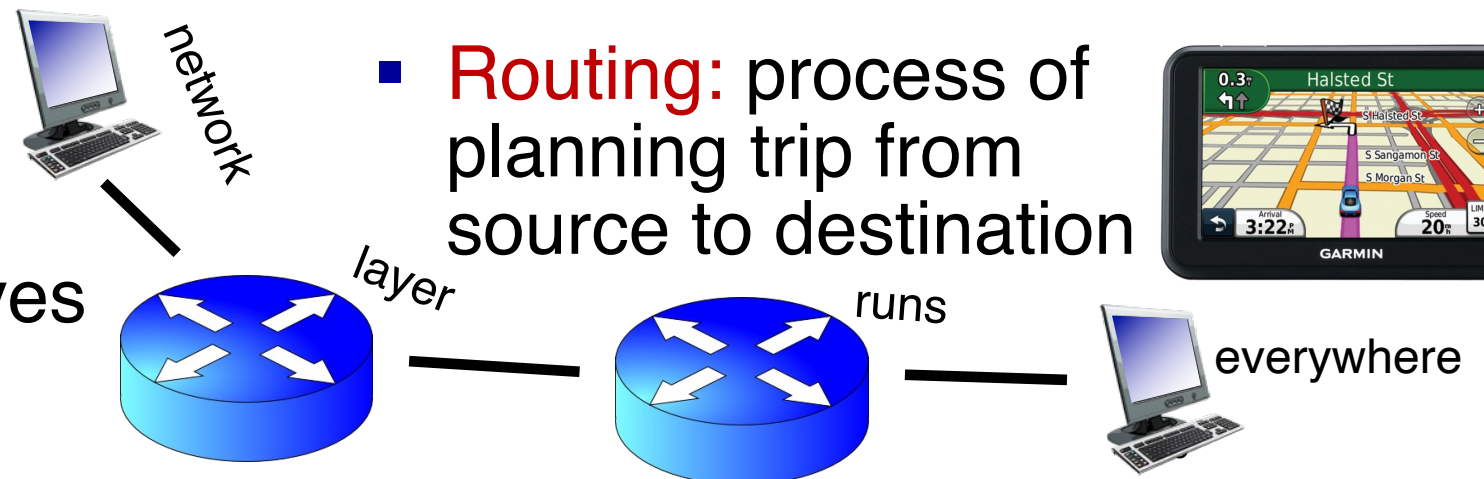


# Two key network-layer functions

- **Forwarding:** move packets from router's input to appropriate router output
- **Routing:** determine route taken by packets from source to destination
  - routing algorithms
- The network layer solves the routing problem.

Analogy: taking a road trip

- **Forwarding:** process of getting through single exit
- **Routing:** process of planning trip from source to destination



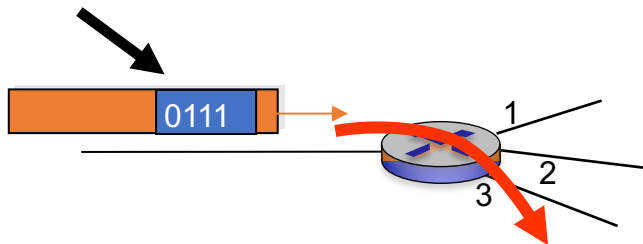
# Distributed Routing

# Control/Data Planes

## Data plane = Forwarding

- local, per-router function
- determines how datagram arriving on router input port is forwarded to router output port

values in arriving  
packet header



## Control plane = Routing

- network-wide logic
- determines how datagram is routed along end-to-end path from source to destination endpoint
- two control-plane approaches:
  - **Distributed routing** algorithm running on each router
  - **Centralized routing** algorithm running on a (logically) centralized machine

# Distributed routing

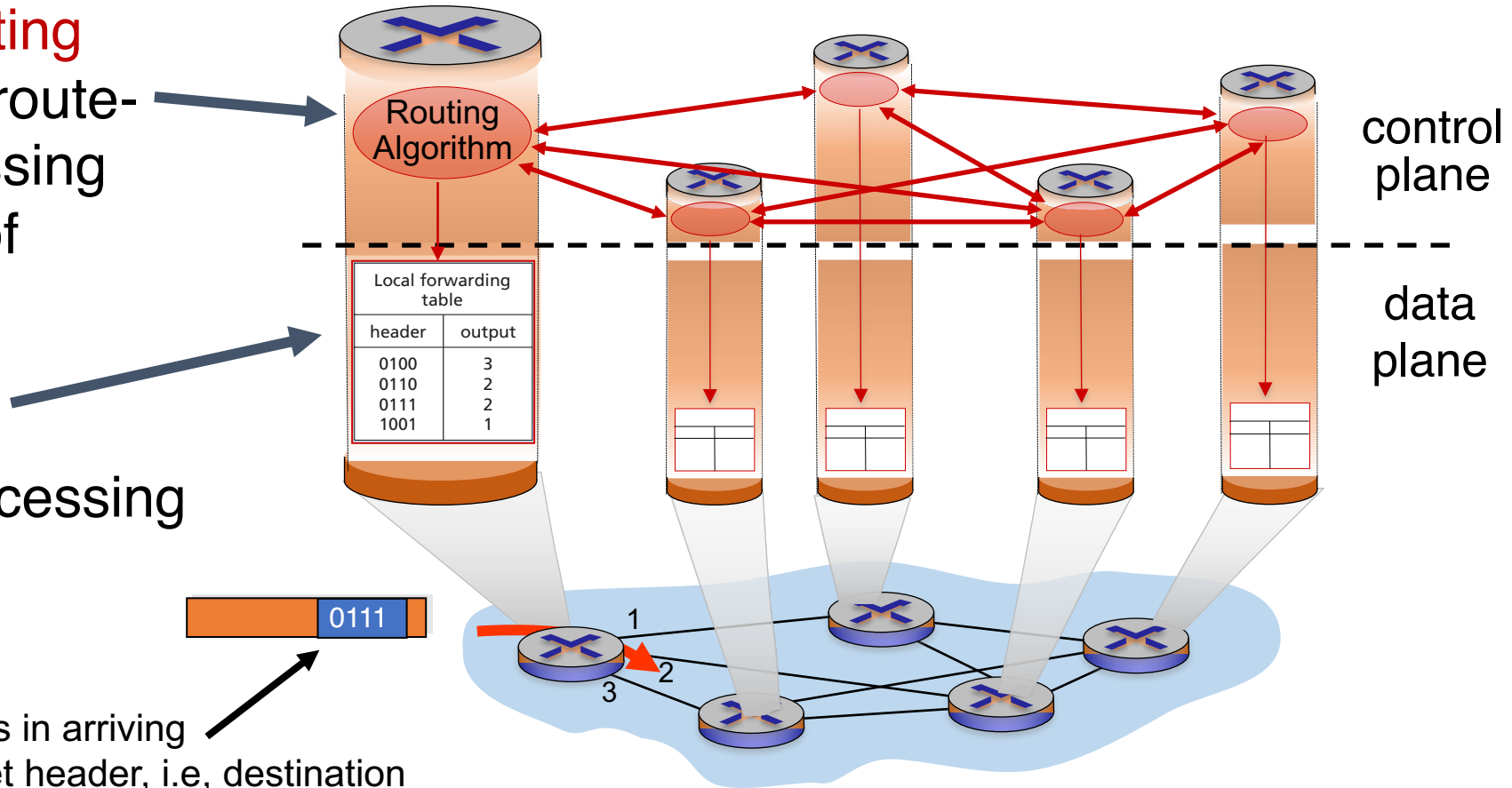
## Control plane

Traditional **routing protocols**: per route-change processing (~ a few tens of seconds)

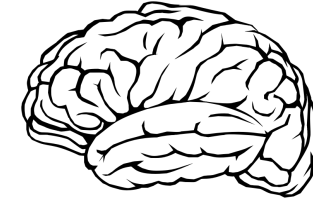
## Data plane

per-packet processing (~ tens of nanoseconds)

values in arriving packet header, i.e, destination IP address



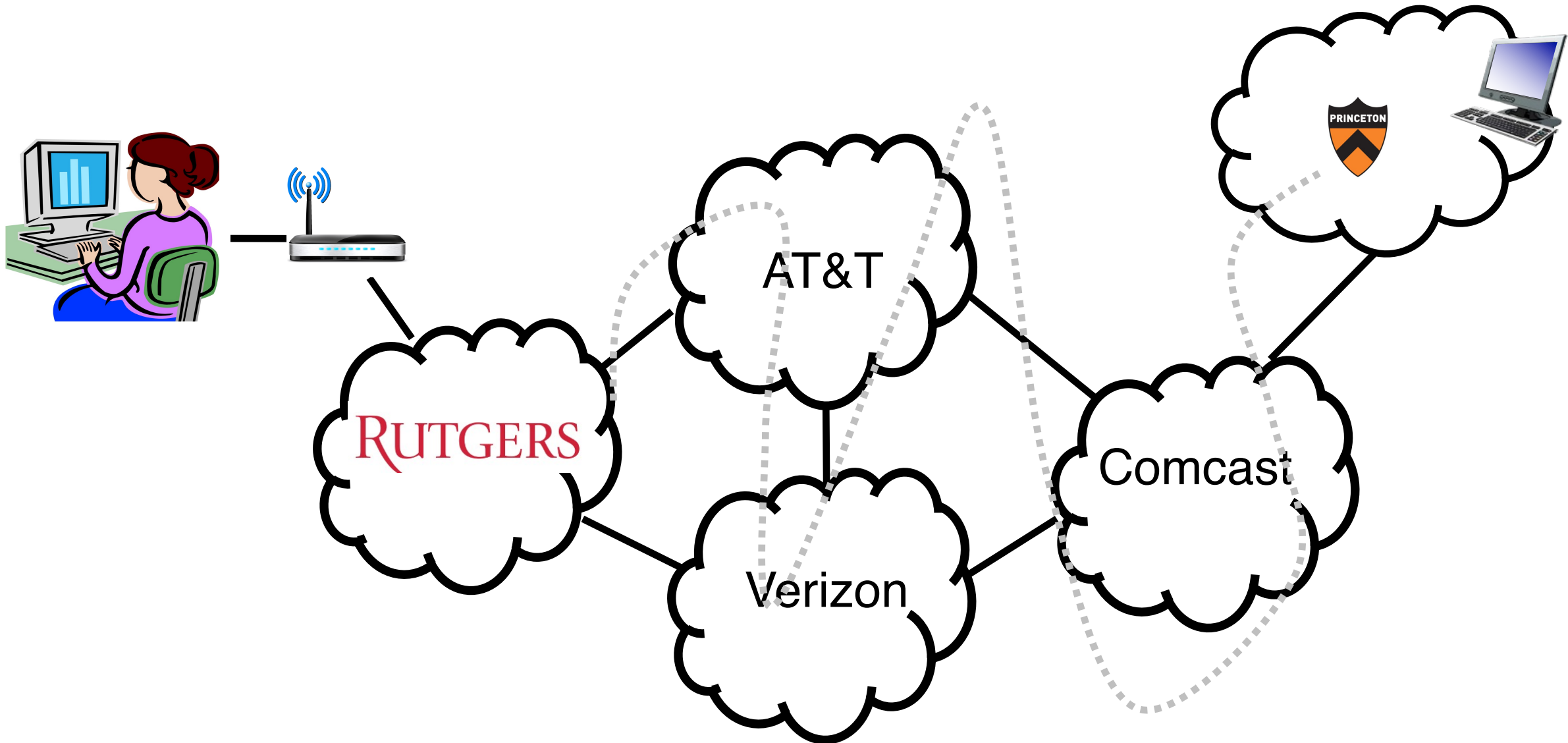
# What should routing optimize?



- Efficiency goals: “good” paths
  - Low latency, low cost, high bandwidth, etc.
  - Often translates to **shortest path** on a suitably modeled graph!
  - Edges: link metrics. Nodes: routers.
- Internet rationale: keep it distributed: avoid failures; scaling.
- Two questions: (1) what messages? (2) what algorithm?
  - What is the algorithm computing? – is already known (shortest path)
  - Link state and distance vector protocols
  - Applicable when



# The Internet is a large federated network

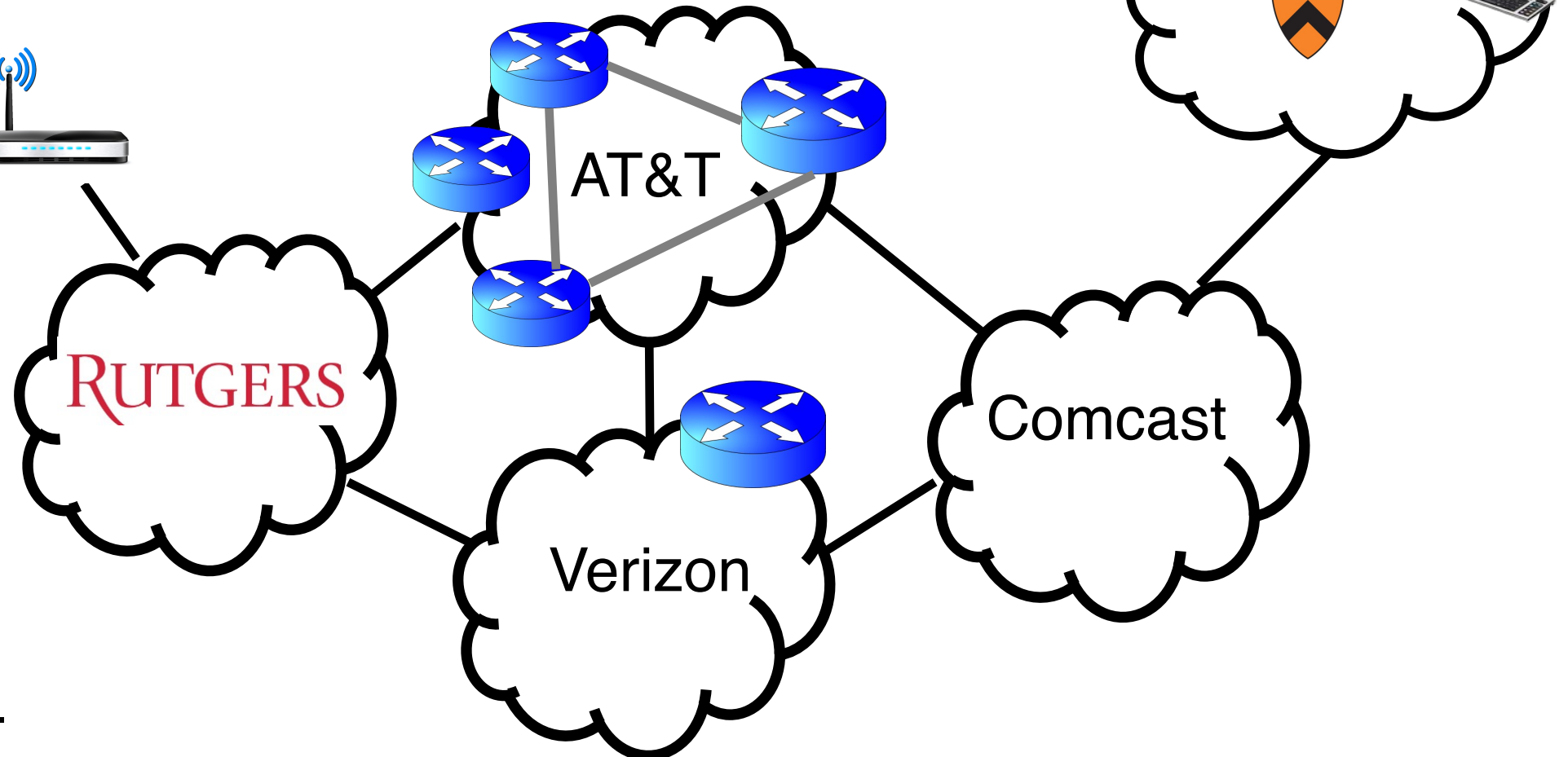




# The Internet is a large **federated** network

Several autonomously run organizations (**AS'es**): No one "boss"

Organizations cooperate, but also **compete**



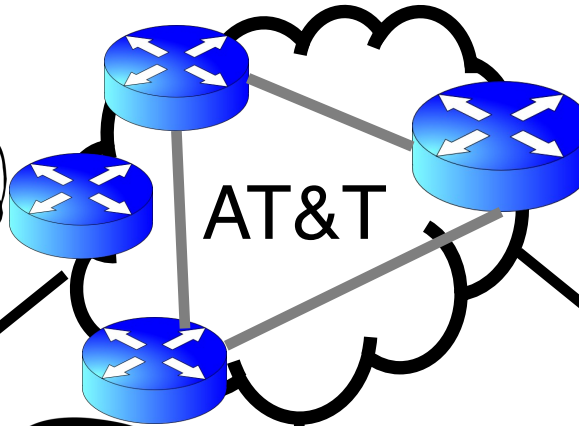
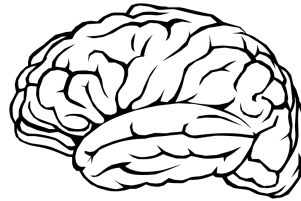
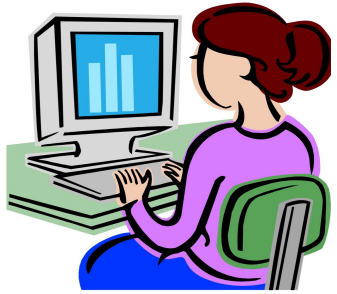
e.g., AT&T has little commercial interest in revealing its internal network structure to Verizon.



# The Internet is a large **federated** network

Several autonomously run organizations: No one “boss”

Organizations cooperate, but also **compete**



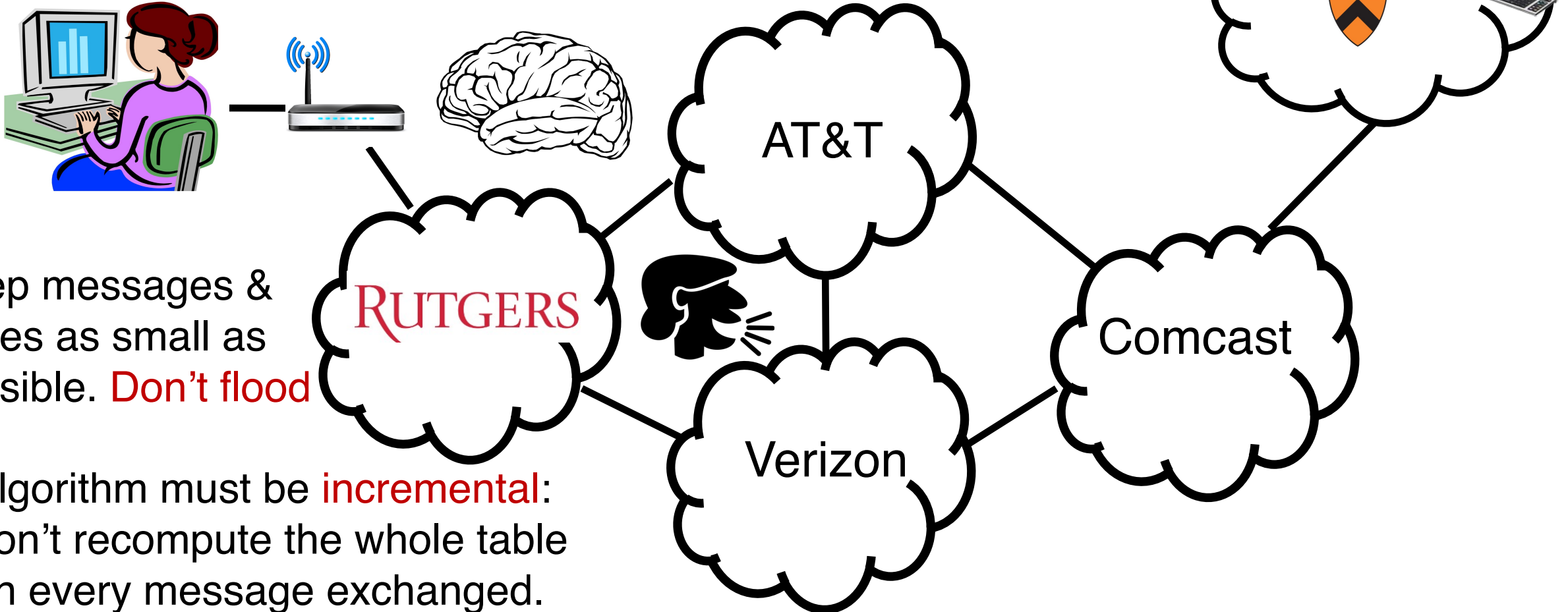
Message exchanges must not reveal internal network details.

Algorithm must work with “incomplete” information about its neighbors’ internal topology.

# The Internet is a **large** federated network

Internet today: > 70,000 unique autonomous networks

Internet routers: > 800,000 forwarding table entries

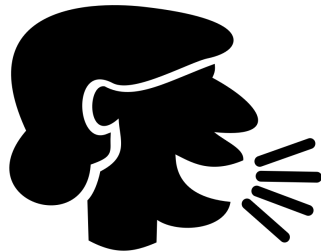


Keep messages & tables as small as possible. **Don't flood**

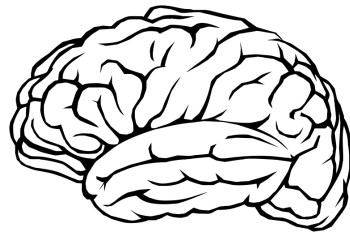
Algorithm must be **incremental**: don't recompute the whole table on every message exchanged.

# Inter-domain Routing

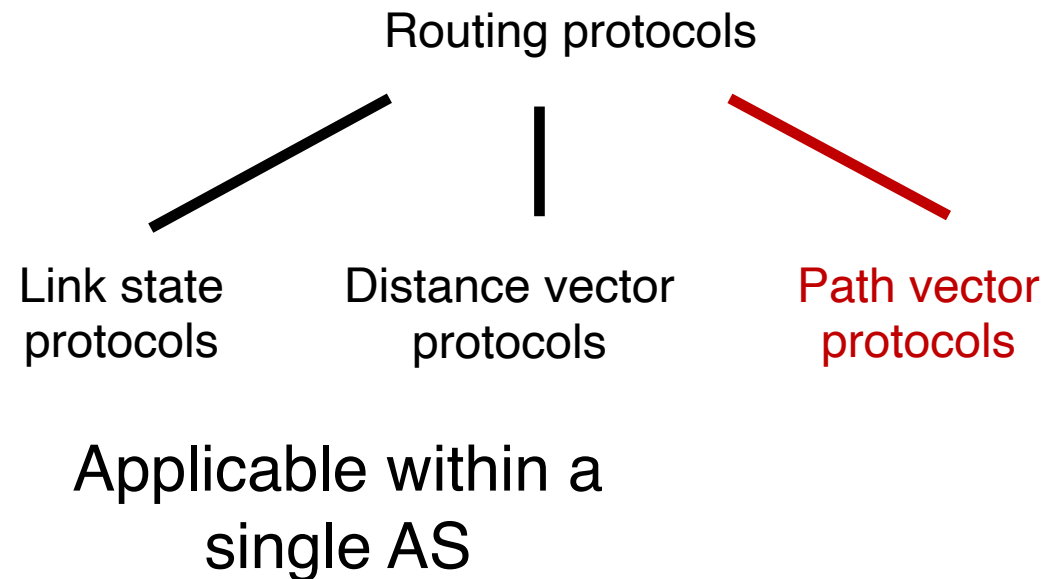
- The Internet uses **Border Gateway Protocol (BGP)**
- **All AS'es speak BGP.** It is the glue that holds the Internet together
- BGP is a **path vector protocol**



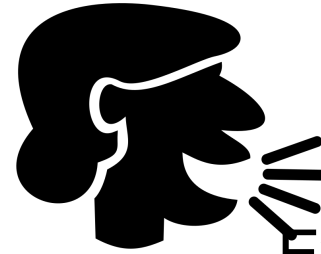
Messages?



Algorithm?



# (1) BGP Messages



Loop detection is easy  
(no “count to infinity”)

Exchange paths: **path vector**

- Routing **Announcements** or **Advertisements** No link metrics, distances!

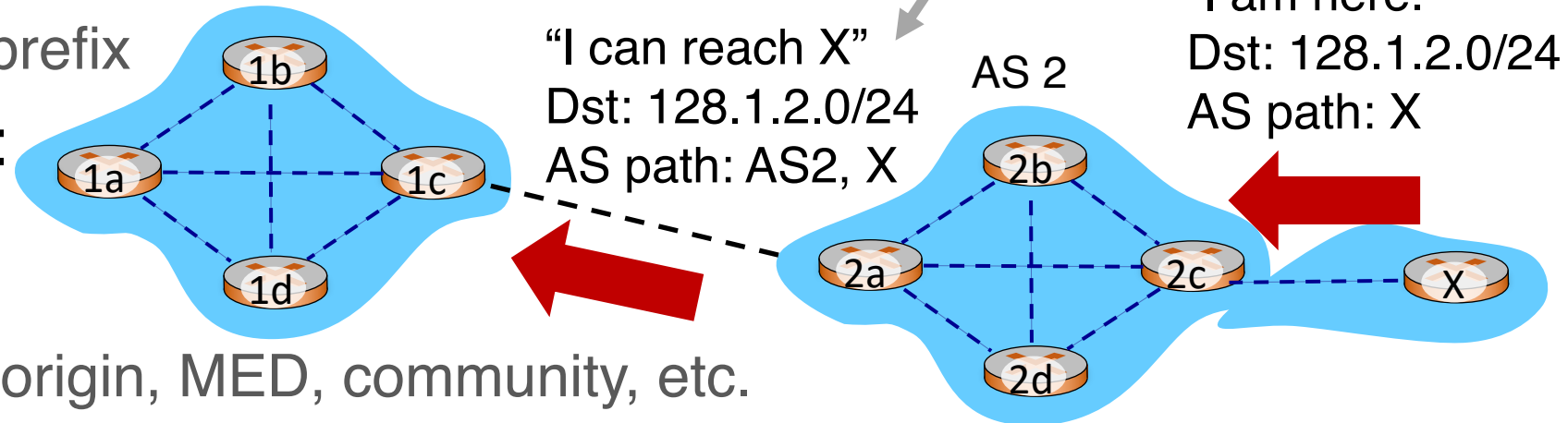
- “I am here” or “I can reach here”
- Occur over a TCP connection (**BGP session**) between routers

- Route announcement = destination + attributes

- Destination: IP prefix

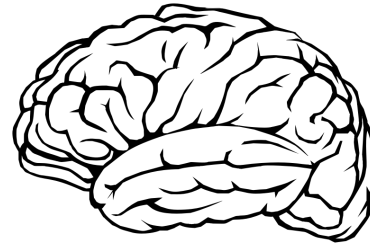
- Route Attributes:

- **AS-level path**
- Next hop
- Several others: origin, MED, community, etc.



- An AS promises to use advertised path to reach destination
- Only route changes are advertised after BGP session established

## (2) BGP algorithm



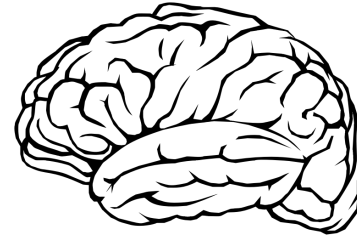
- A BGP router does *not* consider every routing advertisement it receives by default to make routing decisions!
    - An **import policy** determines whether a route is even considered a candidate
  - Once imported, the router performs **route selection**
  - A BGP router does *not* propagate its chosen path to a destination to all other AS'es by default!
    - An **export policy** determines whether a (chosen) path can be advertised to other AS'es and routers
- Programmed by network operator

**Business policy considerations drive BGP.  
NOT efficiency considerations.**

# Policy arises from business relationships

- Customer-provider relationships:
  - E.g., Rutgers is a customer of AT&T
- Peer-peer relationships:
  - E.g., Verizon is a peer of AT&T
- Business relationships depend on **where** connectivity occurs
  - “Where”, also called a “point of presence” (PoP)
  - e.g., customers at one PoP but peers at another
  - Internet-eXchange Points (IXPs) are large PoPs where ISPs come together to connect with each other (often for free)

## Q2. BGP Route Selection



- When a router imports more than one route to a destination IP prefix, it selects route based on:
  1. **local preference value** attribute (import policy decision -- set by network admin)
  2. shortest AS-PATH
  3. closest NEXT-HOP router
  4. Several additional criteria: You can read up on the full, complex, list of criteria, e.g., at <https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html>

# Problems with BGP

- Not designed for efficiency

1. **local preference value** attribute (import policy decision -- set by network admin)
2. shortest AS-PATH
3. closest NEXT-HOP router

- Only a single path per destination
- Slow to converge after a change
- Vulnerable to bugs & malice

Approaches to bring flexibility:  
Flexible control logic for path selection  
(Google, Facebook)  
Detour/overlay routing (Akamai)

Nothing to do with  
path length, delay, or  
available capacity.





# So far: layer-3. But “layer-2” exists too

- **Switch**: move packets based on link layer addresses
- Provide an illusion of a single link connecting many endpoints
  - Without every endpoint necessarily hearing every other endpoint
- **Learning switch**: zero configuration or control plane.
  - All endpoints in the same IP network
  - Flood packets when dest MAC address unknown
  - Use source MAC of incoming packets and associate with the incoming switch port: use later for forwarding
- Works even if endpoints move, so long as they are in the same IP prefix

# Centralized Routing

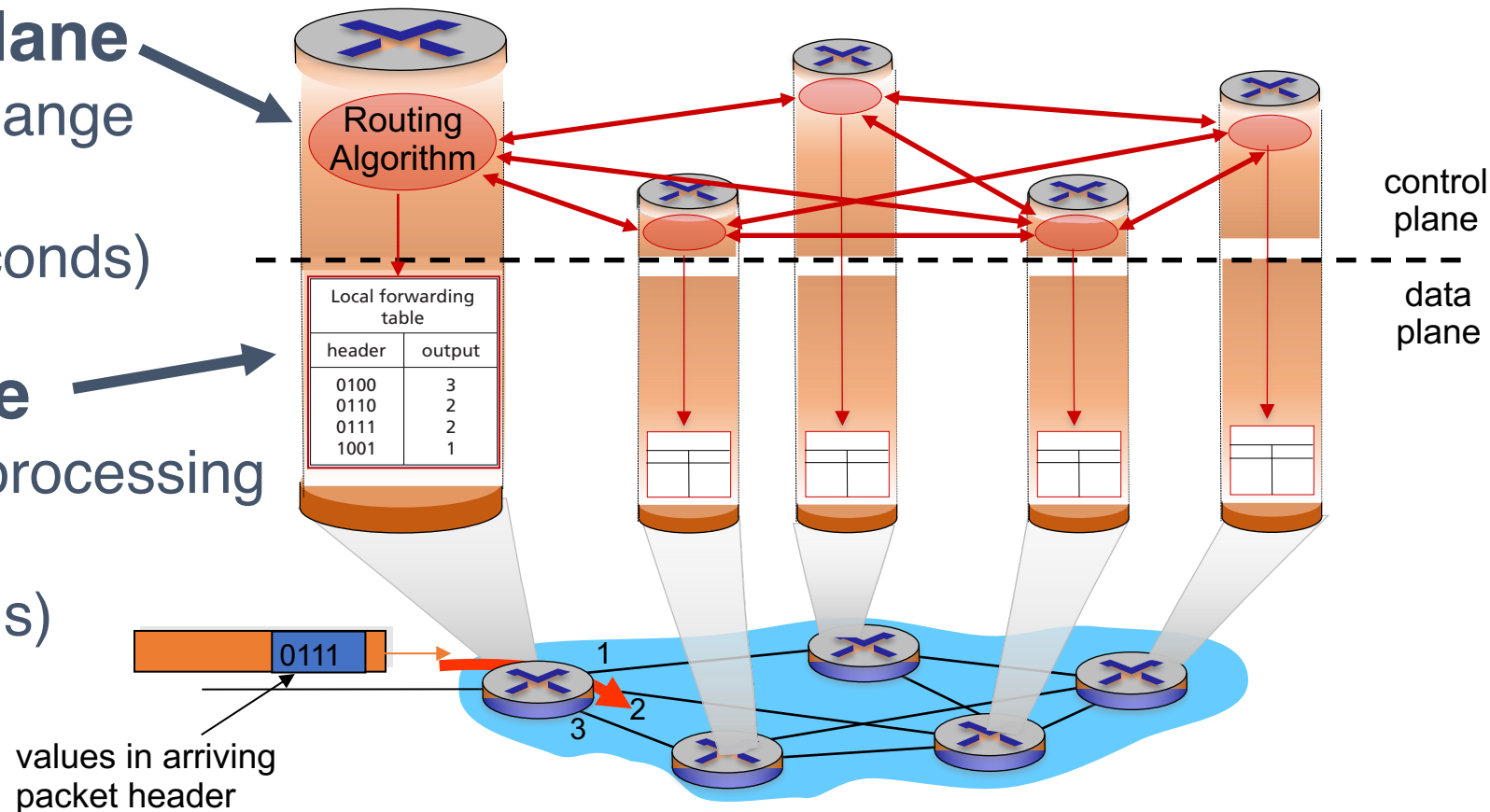
# Control & Data Planes inside a router

Traditionally:

Individual routing algorithm components *in each and every router* interact in the control plane

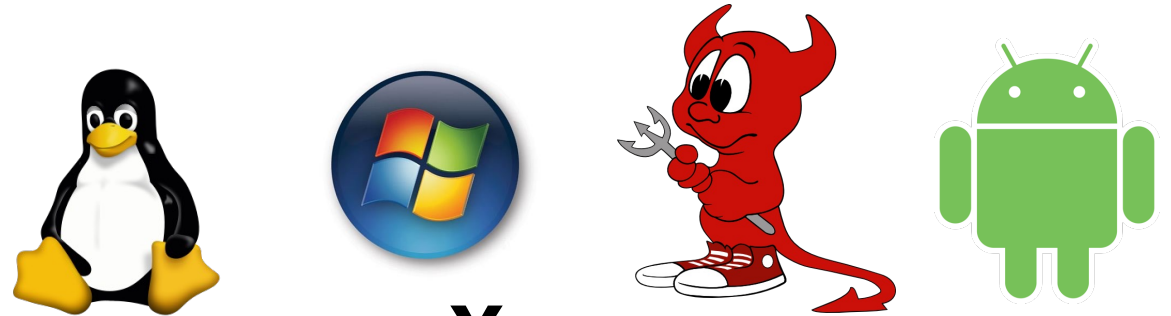
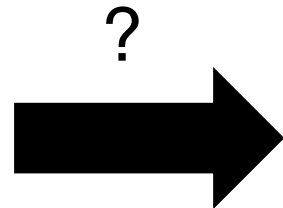
**Control plane**  
per route-change processing  
(~ a few seconds)

**Data plane**  
per-packet processing  
(~ tens of nanoseconds)

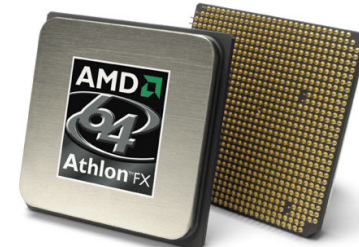
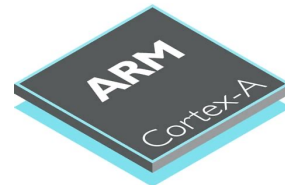


# Problems with traditional control planes

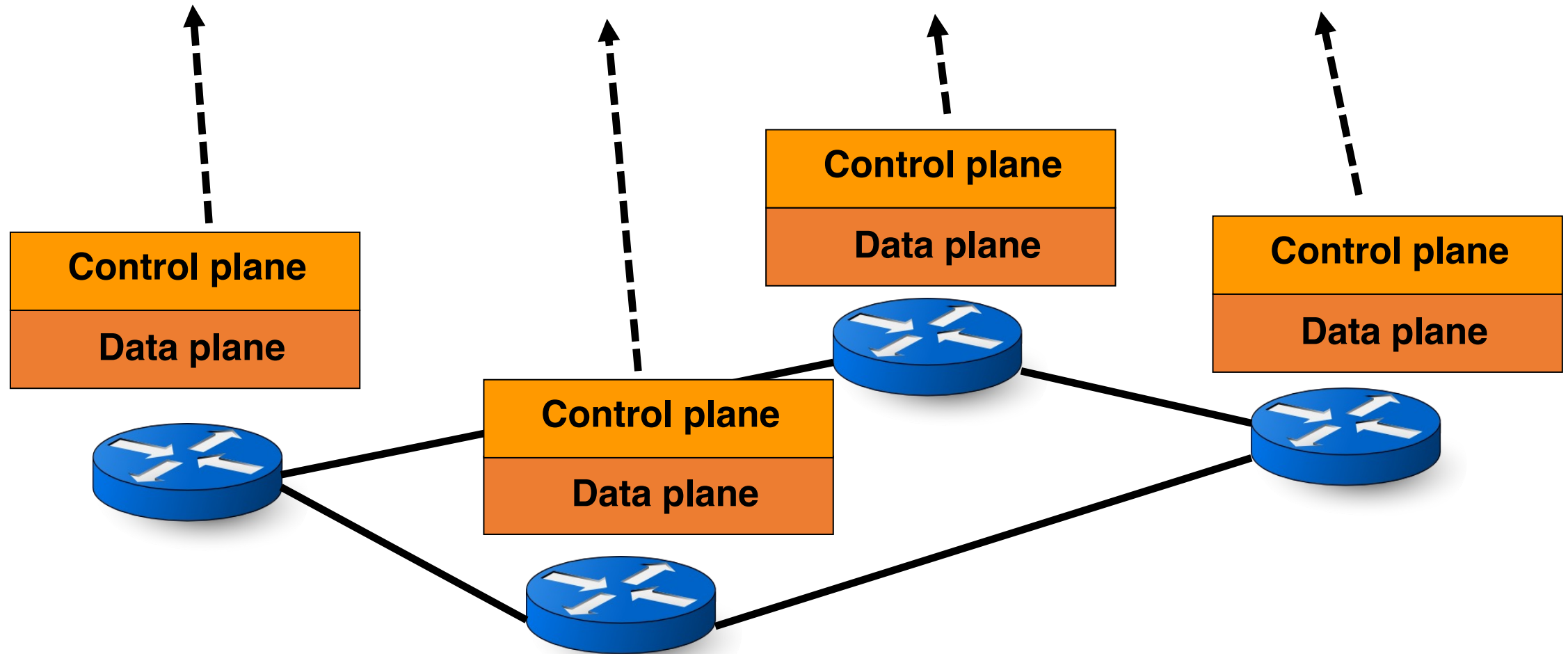
- Management decisions tied to distributed protocols
  - Ex: Set OSPF link weights to force traffic through desired path
  - Ex: Non-deterministic network state after a link failure
- Data and control plane controlled by vendors: proprietary interfaces



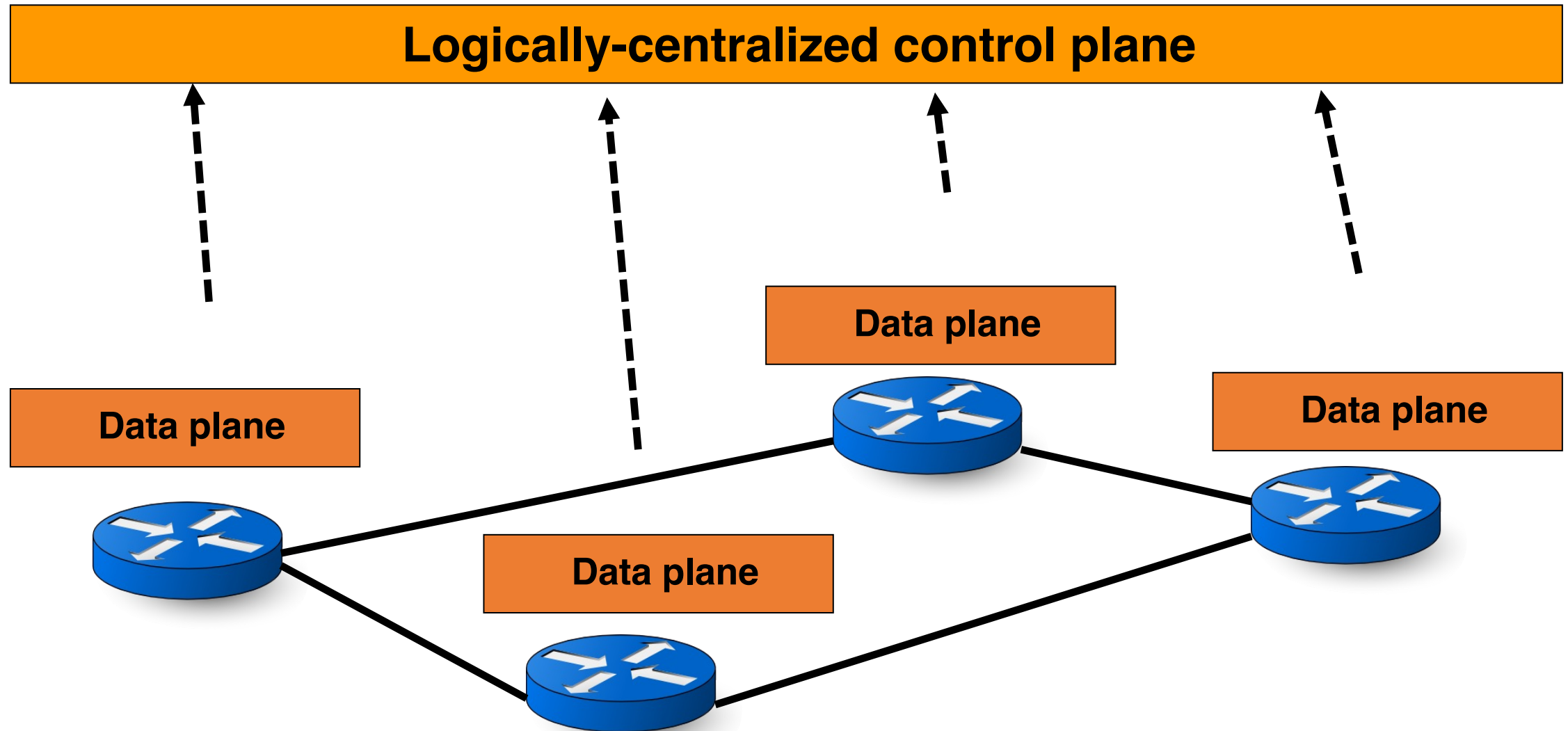
X



# Traditional IP network



# Software-defined network



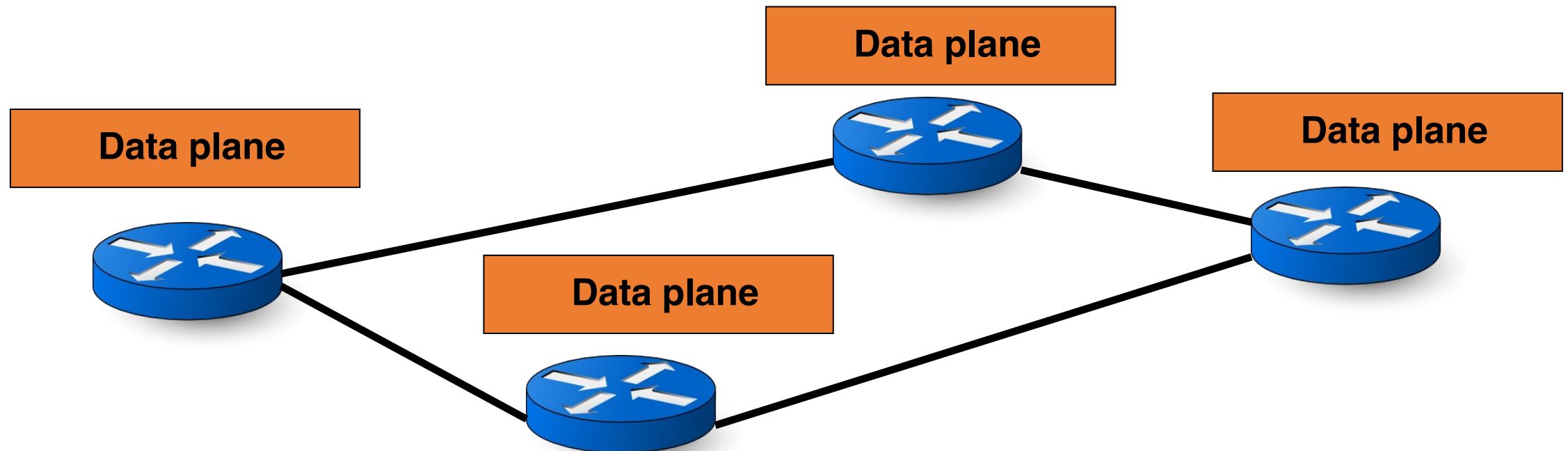
# Software-Defined Networking

# SDN (1/2): Centralized control plane

## SDN controller

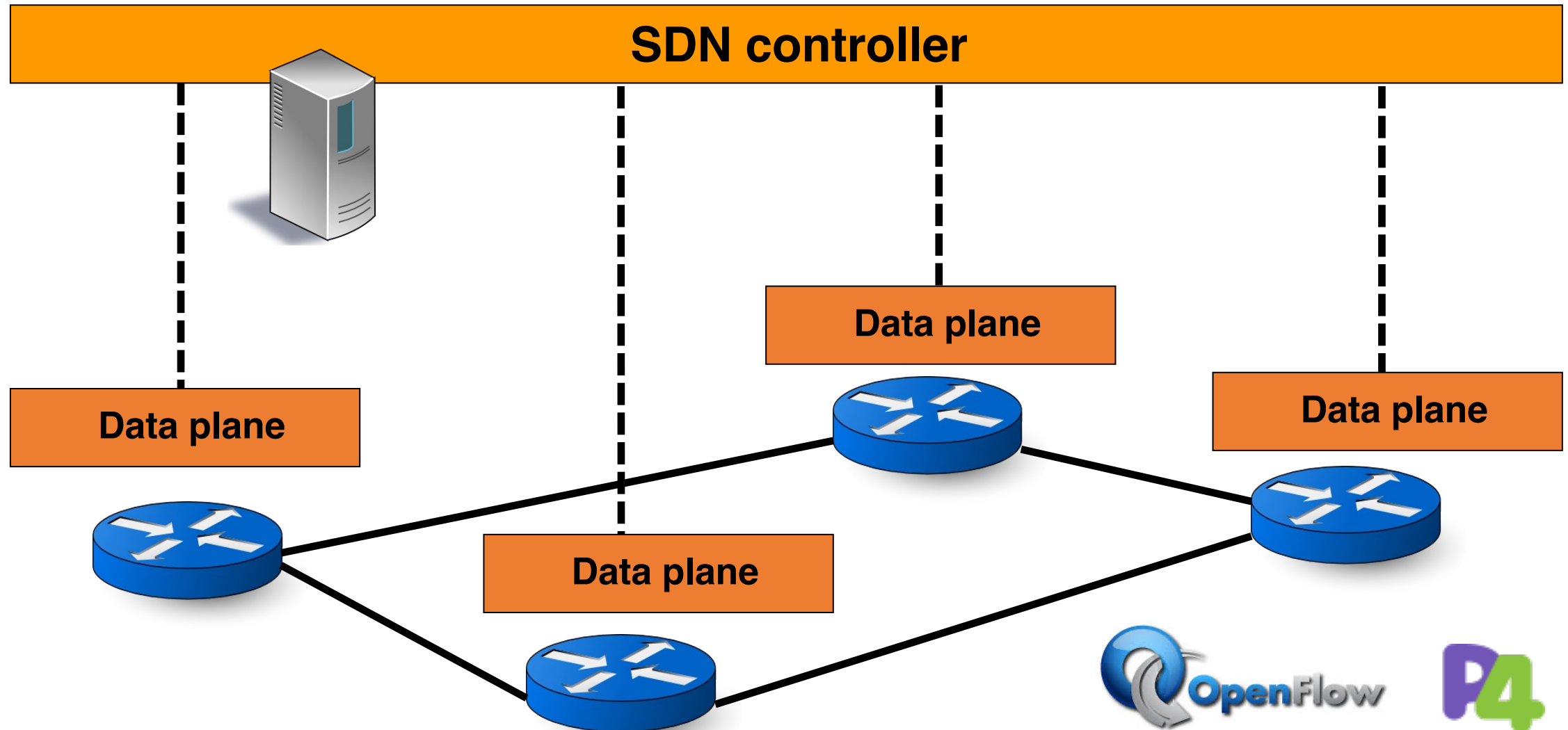


Control planes lifted from switches  
... into a logically centralized *controller*  
... running in a compute cluster



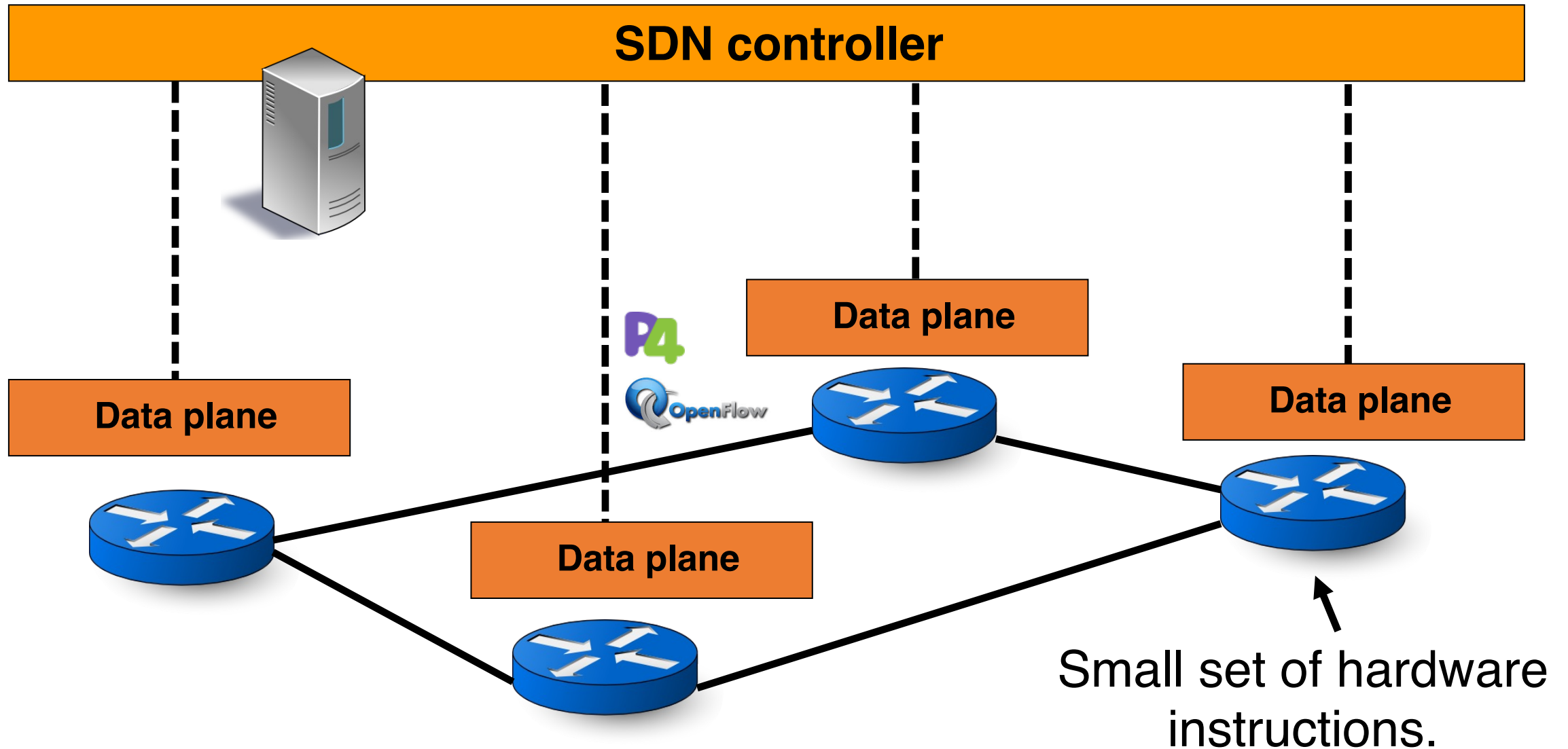


# SDN (2/2): Open interface to data plane



Some immediate consequences

# (1) Simpler switches




# Data plane primitive: Match-action rules

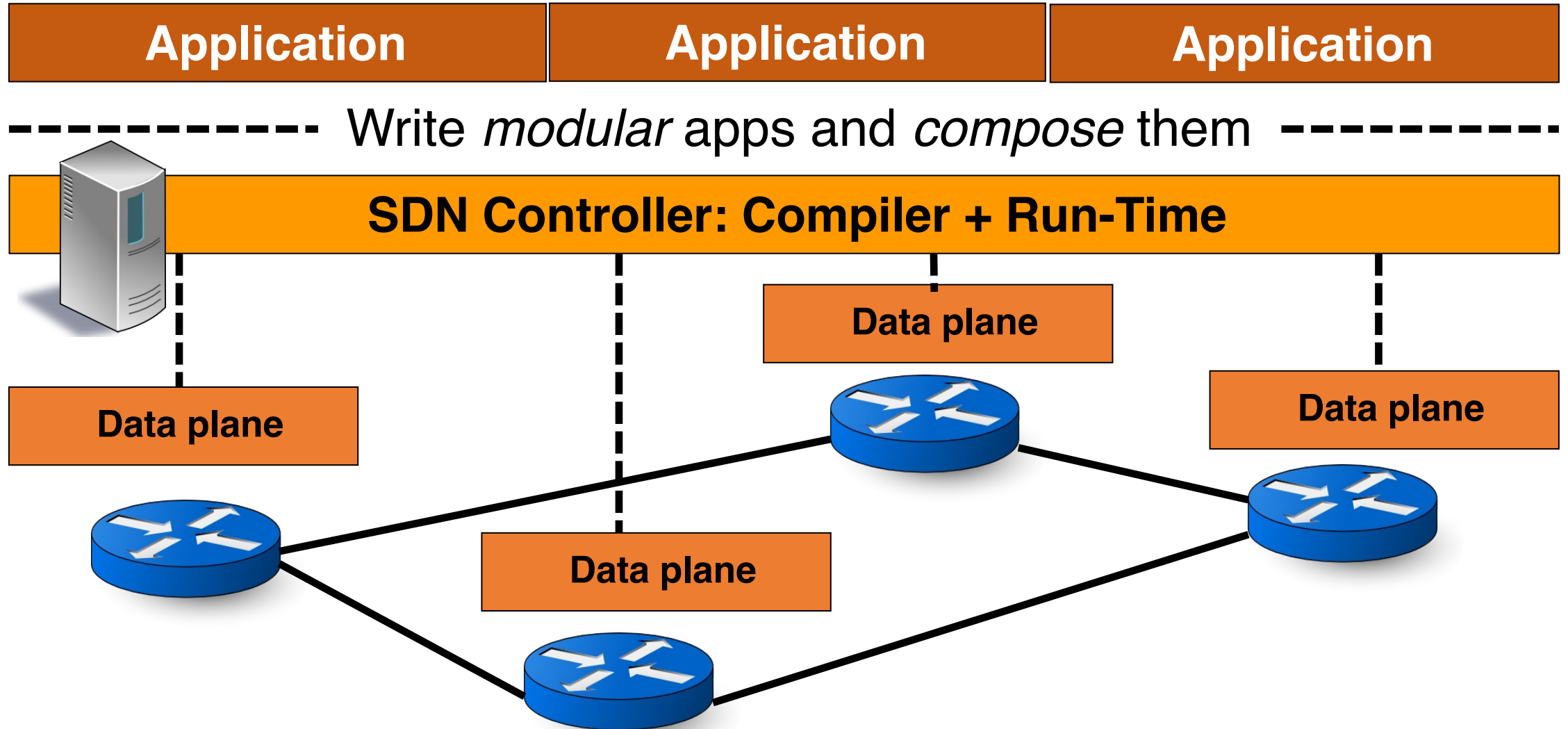
- Match arbitrary bits in the packet header



Match: 1000x01xx01001x

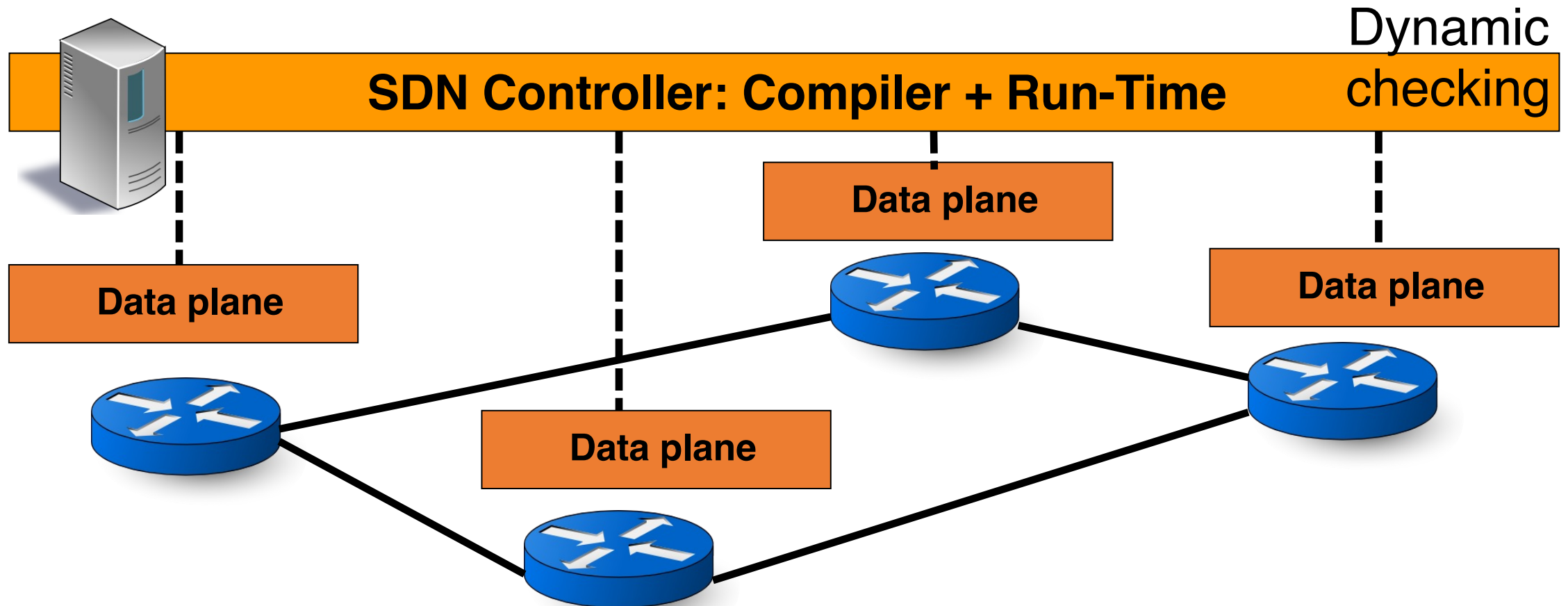
- Match on any header, or new header 
- Match exact, a subset (ternary), or over a range
- Allows any flow granularity
- **Actions**
  - Forward to port(s), drop, send to controller, count,
  - Overwrite header with mask, push or pop, ... Action: fwd(port 2)
  - Forward at specific bit-rate
- **Prioritized list of rules** Priority: 65500

## (2) Network programming abstractions



# (3) Formal verification of Network Policy

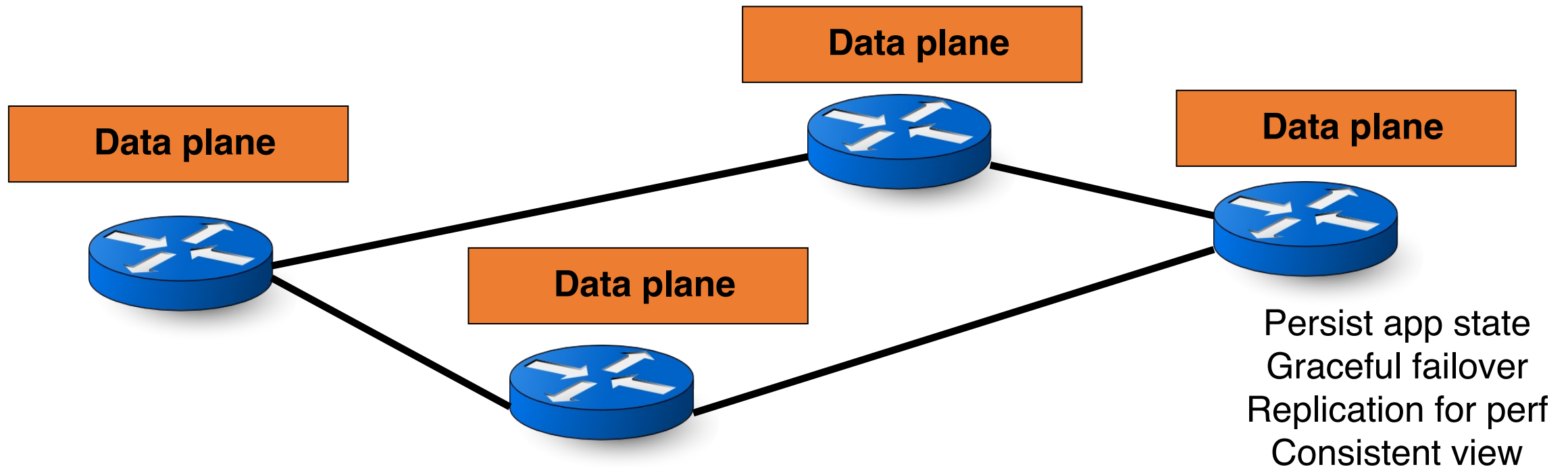
Static checking    Application (specified as code)



# (4) Unified network operating system



Separate distributed system concerns from expressing intent



# Technical challenges of SDN

- Availability: surviving failures of the controller & data plane
- Controller scalability: many routers, many events
  - Response time: Delays between controller and routers
- Consistency: Ensuring multiple controllers behave consistently
- Designing flexible router mechanisms
- Compilation: translating intent to mechanisms
- Verification: ensuring controller policy is faithfully implemented
- Security: entire network owned if the controller is exploited
- Interoperability: legacy routers? neighboring domains?
  - Developing interfaces that are portable across hardware vendors



# Virtualizing Networking in a Shared Cluster

# Networking in a multi-tenant cloud

- L2 and L3 networking: basics
- A typical public cloud network topology: Tree upon ToRs
- Problems: Many tenants, time-varying demands.
  - Want homogeneity across data center on use of compute capacity
  - Where to provision VMs?
  - How to migrate VMs or scale the number of VMs?
- Ideas: VMs get their own network addresses
  - network **address** virtualization
- Ideas: tenants should be able to use custom topologies
  - Needed “in practice” rather than “in principle”. But important to do.

# How cloud network looks to tenant

- Control abstraction: pipeline of lookup tables.
  - Example: L2, ACL, L2.
  - Example: L2, L3, ACL
- Packet abstraction: send to IP addresses of your own
  - Processed through switch/router topology
  - Data plane behavior defined through control plane configuration
- Design of NVP: (nicira virtualization platform):
  - Push all interesting data plane behaviors to the edge (hypervisor, OVS)
  - The core of the network (switches/routers) just moves data using tunnel headers

# Datapath design

- (1) Topology virtualization: Implement tenant control plane policies faithfully
  - Compute match-action forwarding rules inside a pipeline of logical data paths. Plumb them to each other carefully.
- (2) Address virtualization: Get hypervisors to tunnel to each other based on forwarding outcomes from tenant's logical data paths
  - Separate protocol to communicate the hypervisor's provider-address, hosted VM identifiers, and logical port identifiers for each VM to the controller
  - Eventually send packet to the local virtual NIC of the VM
- Use caching heavily to avoid many table lookups

# 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

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