Network Virtualization

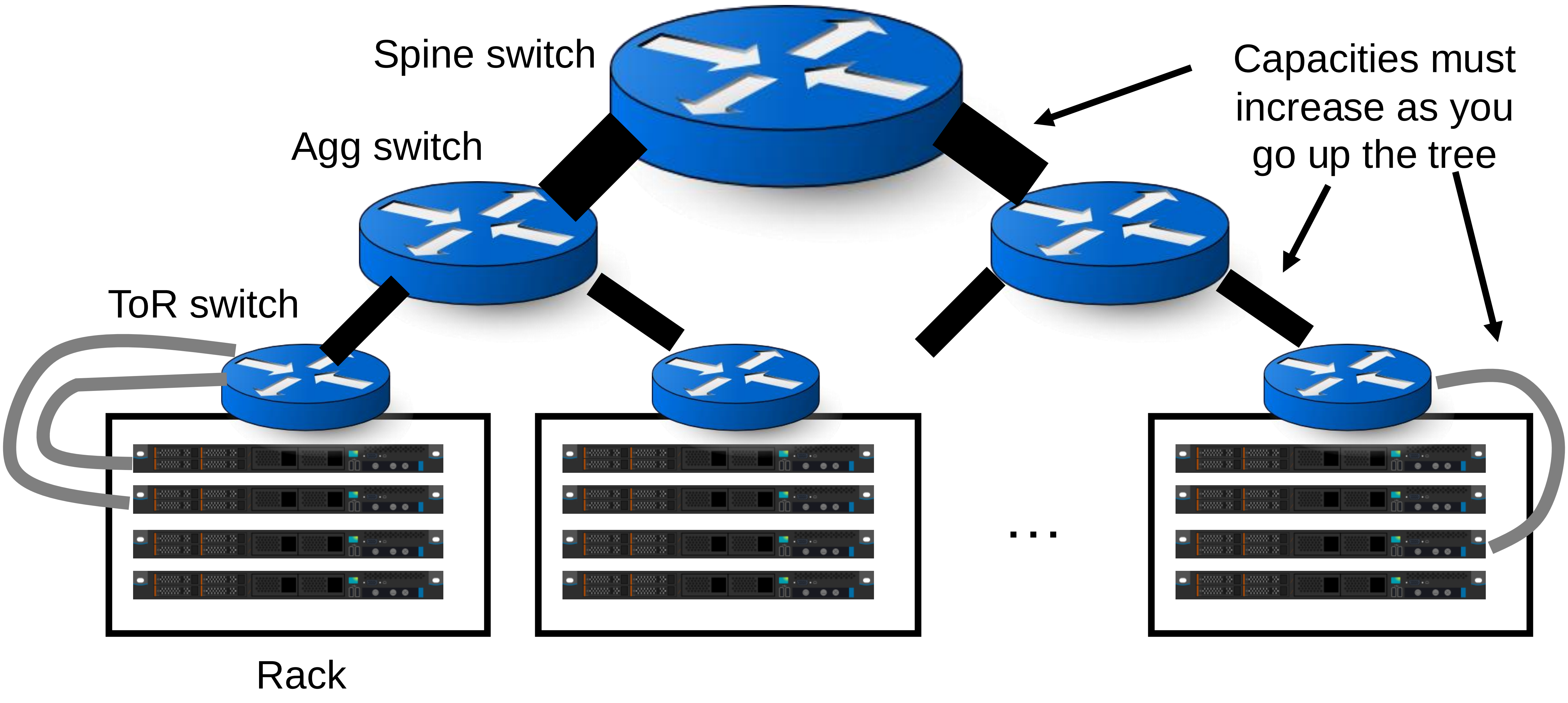
Lecture 12a

Srinivas Narayana

http://www.cs.rutgers.edu/~sn624/553-S25

RUTGERS
UNIVERSITY | NEW BRUNSWICK

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 network connectivity for tenant workloads
  - Migration from on-premise compute cluster to shared cloud

Design Choice: 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

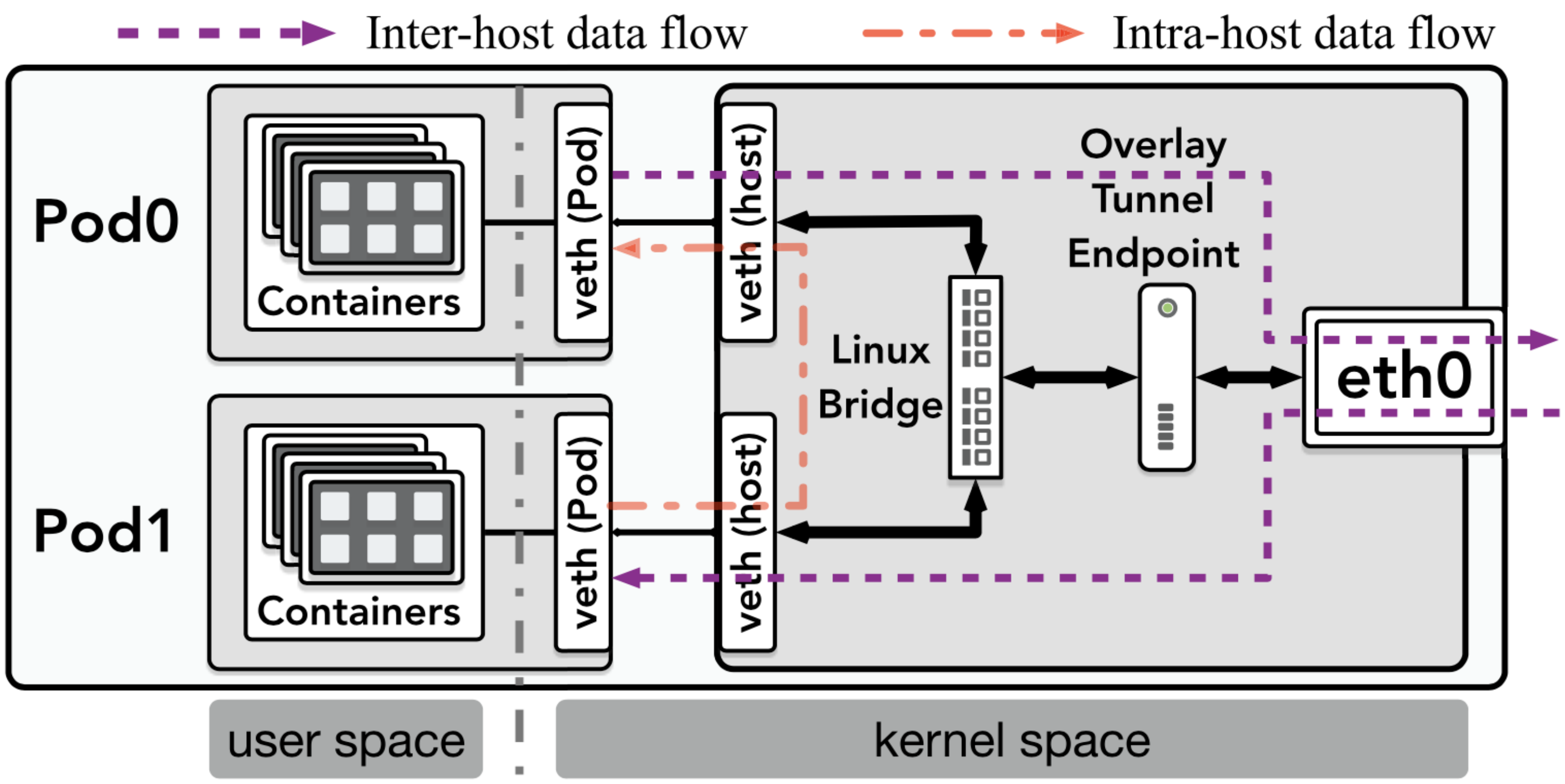
- Address virtualization: VMs/pods use own addresses (CA's)
  - Physical network does not know how to route CA's
  - Additional software to translate CA's between PA's: Tunneling
  - Tunneling endpoint (TEP): software tun/tap interface, NIC hardware, or software switch within a hypervisor. Overlay.
  - 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 service models for the network
  - e.g., rate limits and isolation across tenants sharing a physical machine

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

(Also applies when “machines” substituted by “networks”)

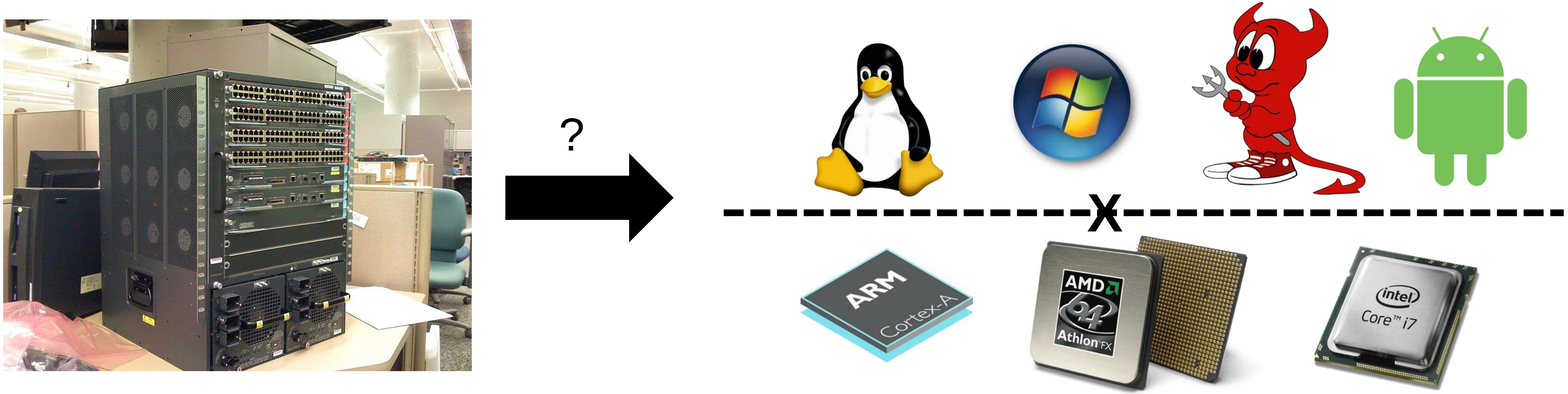
Ex: Network Virtualization in Kubernetes

- Example with L2+L3 overlay

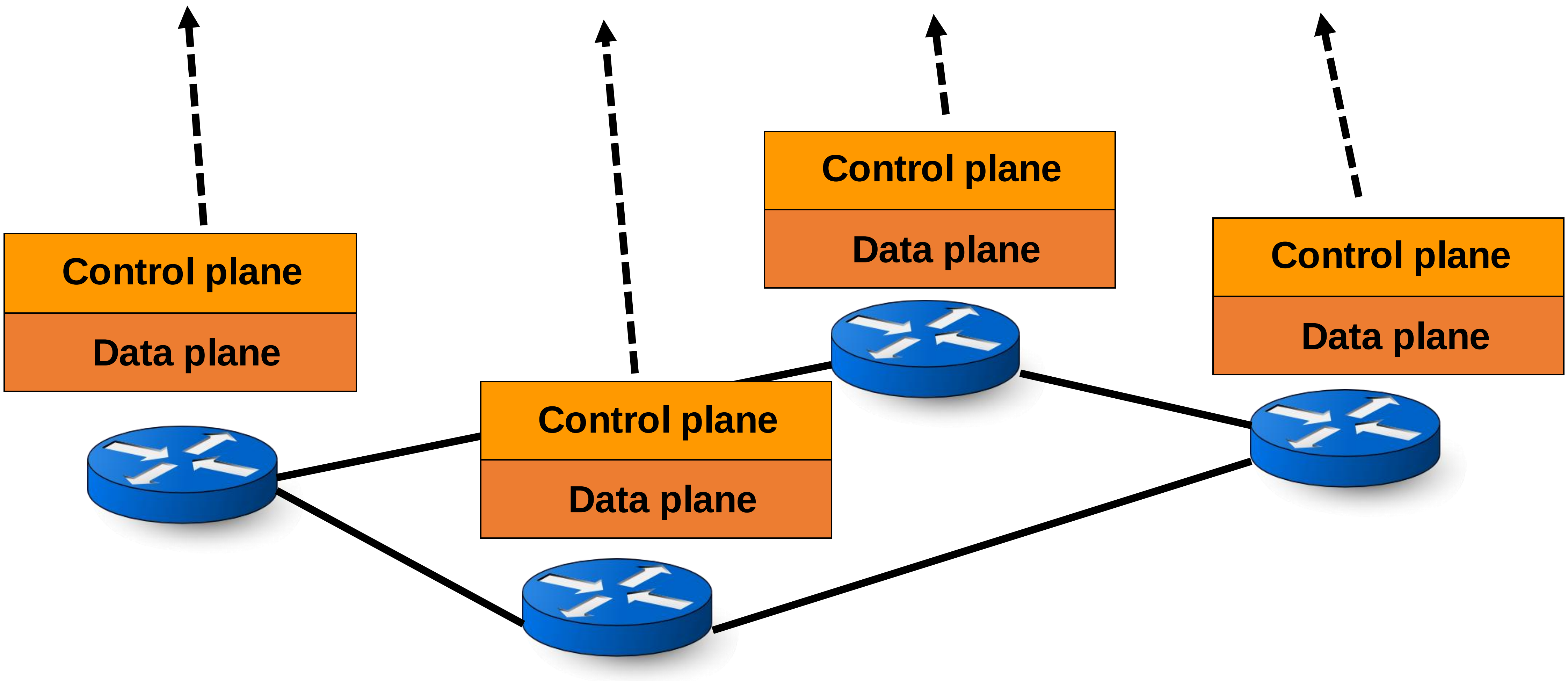


Network control is typically distributed

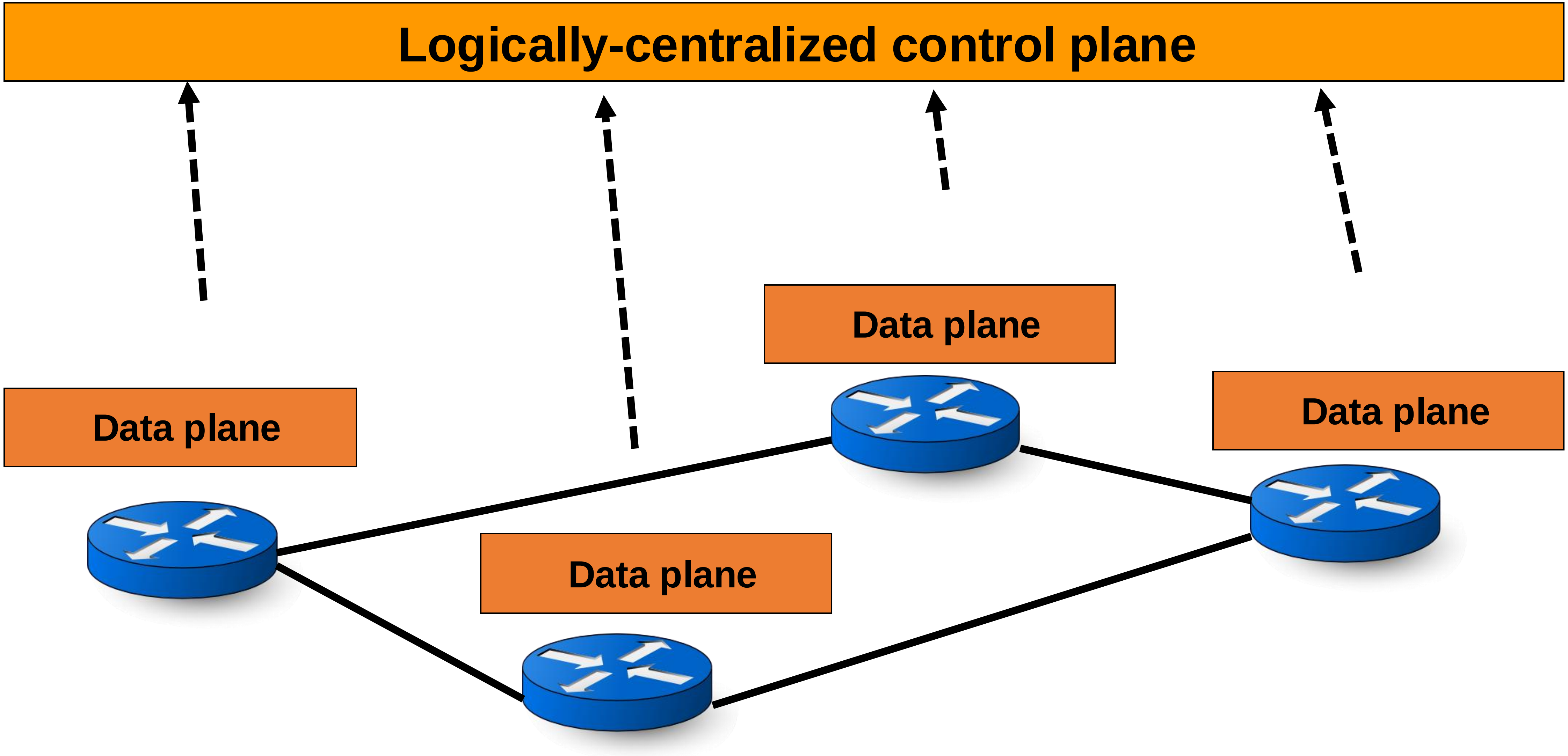
- Traditional IP network: Management tied to distributed protocols
  - Ex: Set OSPF link weights to force traffic through a desired path
  - Ex: Non-deterministic network state after a link failure
- Data and control plane controlled by vendors: proprietary interfaces



Traditional IP network



Software-defined network

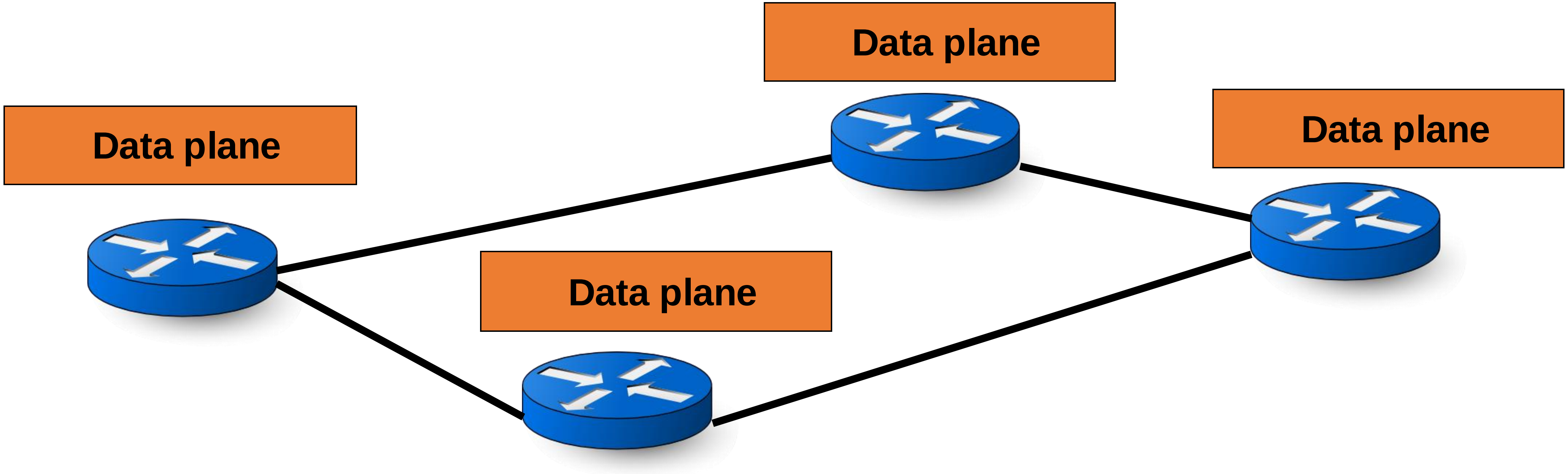


Software-Defined Networking

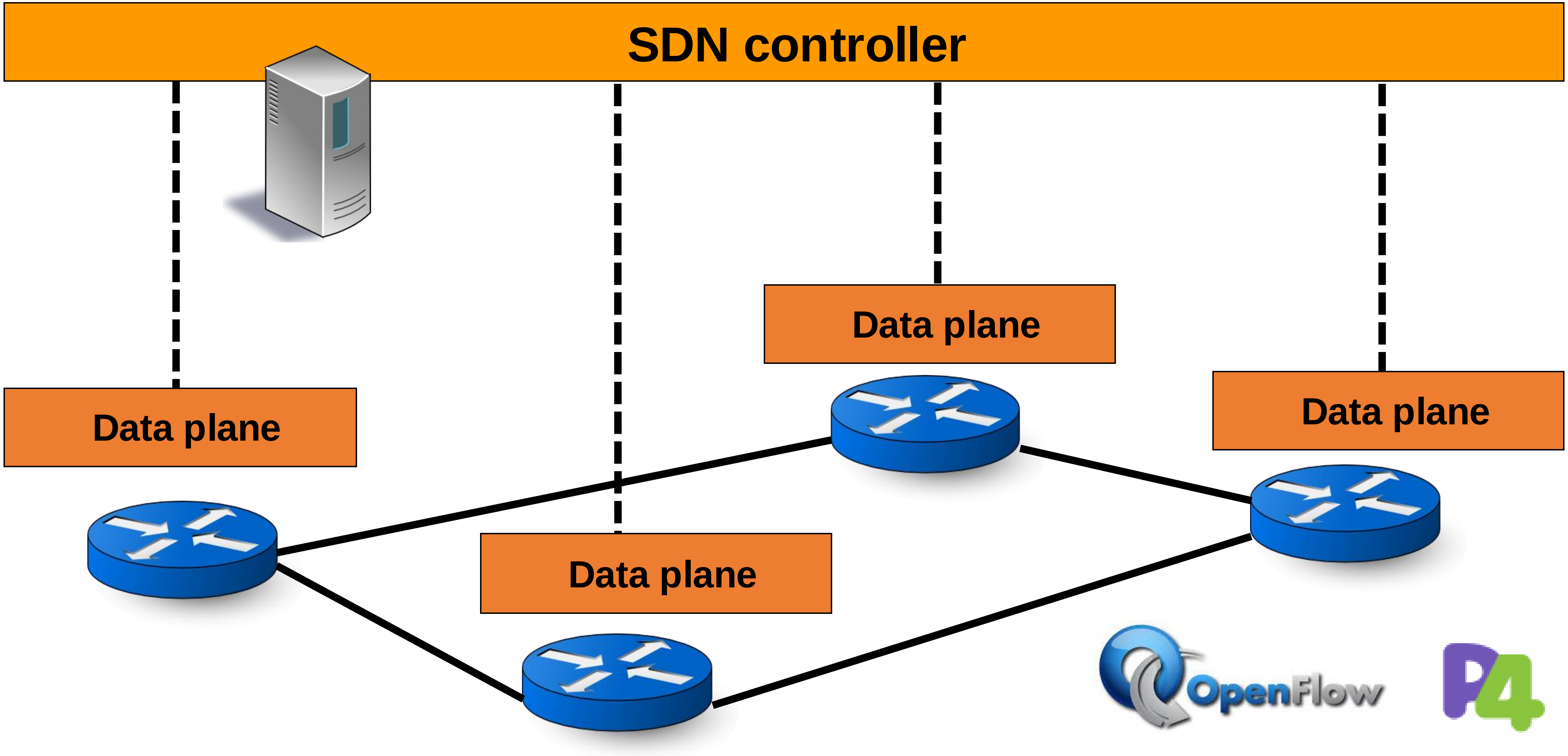
SDN (1/2): Centralized control plane

SDN controller

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

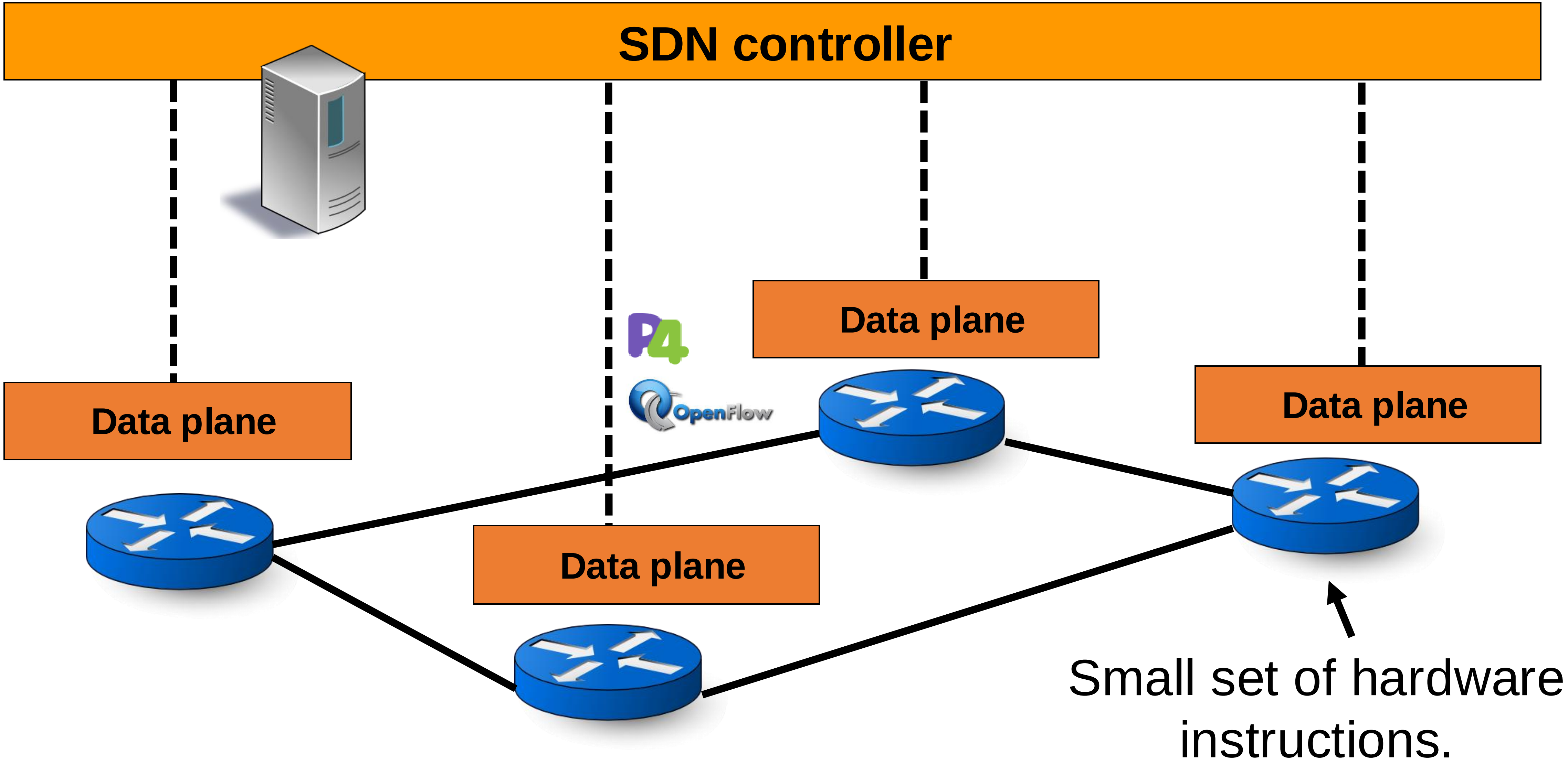


SDN (2/2): Open interface to data plane



Some immediate consequences

(1) Simpler switches



Data plane primitive: Match-action rules

- Match arbitrary bits in the packet header

Match: 1000x01xx01001x

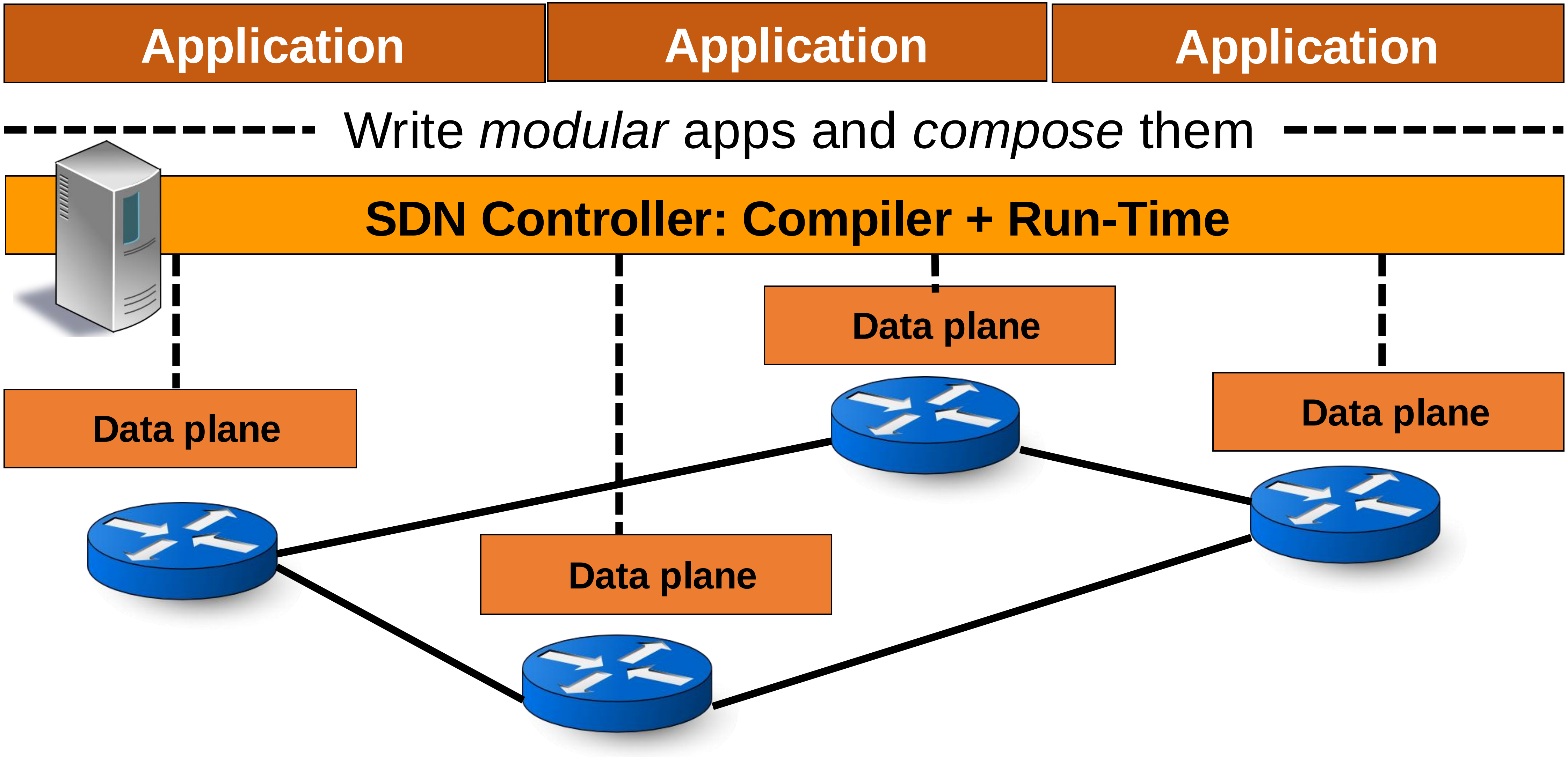
  - Match on any header, or new header
  - Match exact, a subset (ternary), or over a range
  - Allows any flow granularity
- Actions
  - Forward to port(s), drop, send to controller, count,
  - Overwrite header with mask, push or pop, …
  - Forward at specific bit-rate

Action: fwd(port 2)

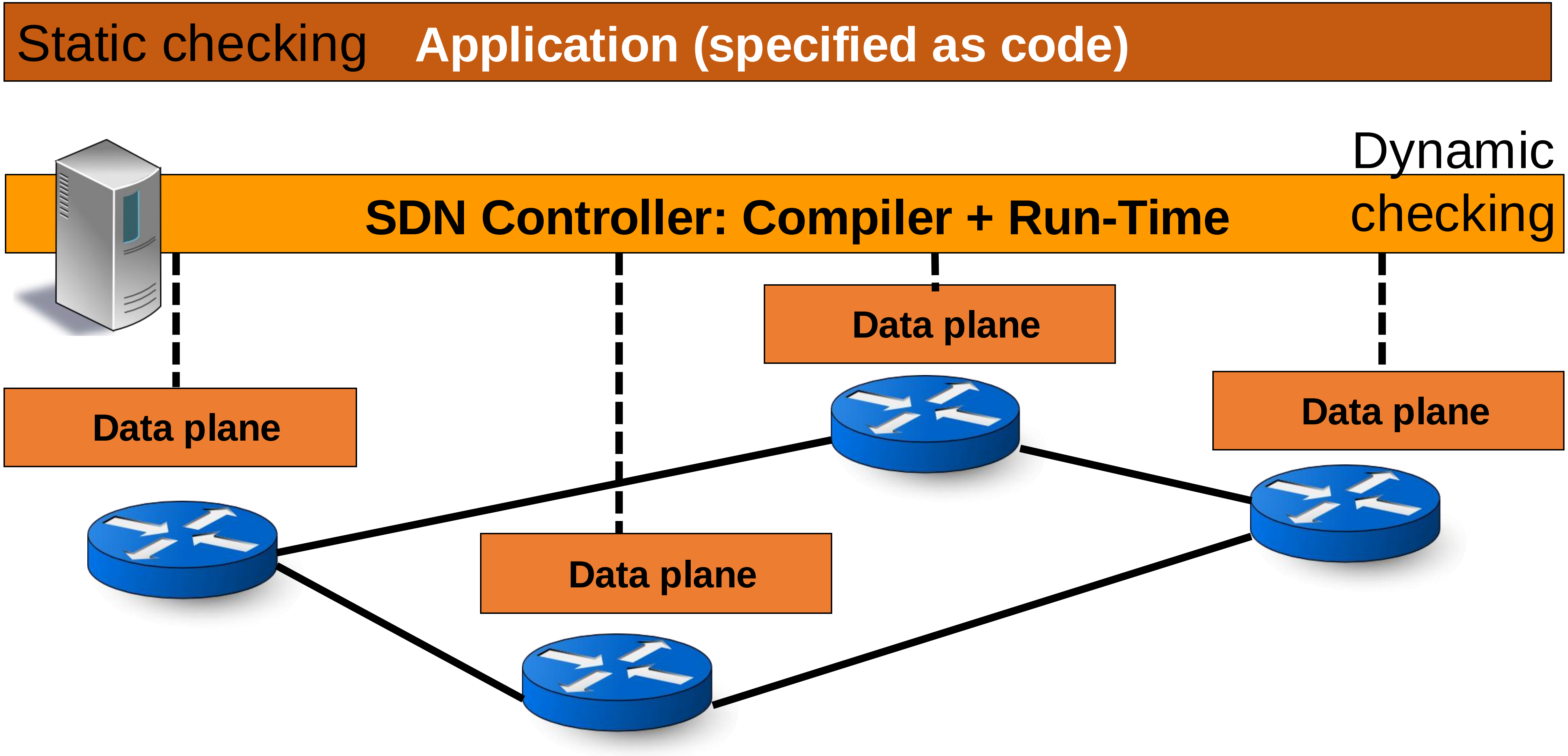
- Prioritized list of rules

Priority: 65500

(2) Network programming abstractions



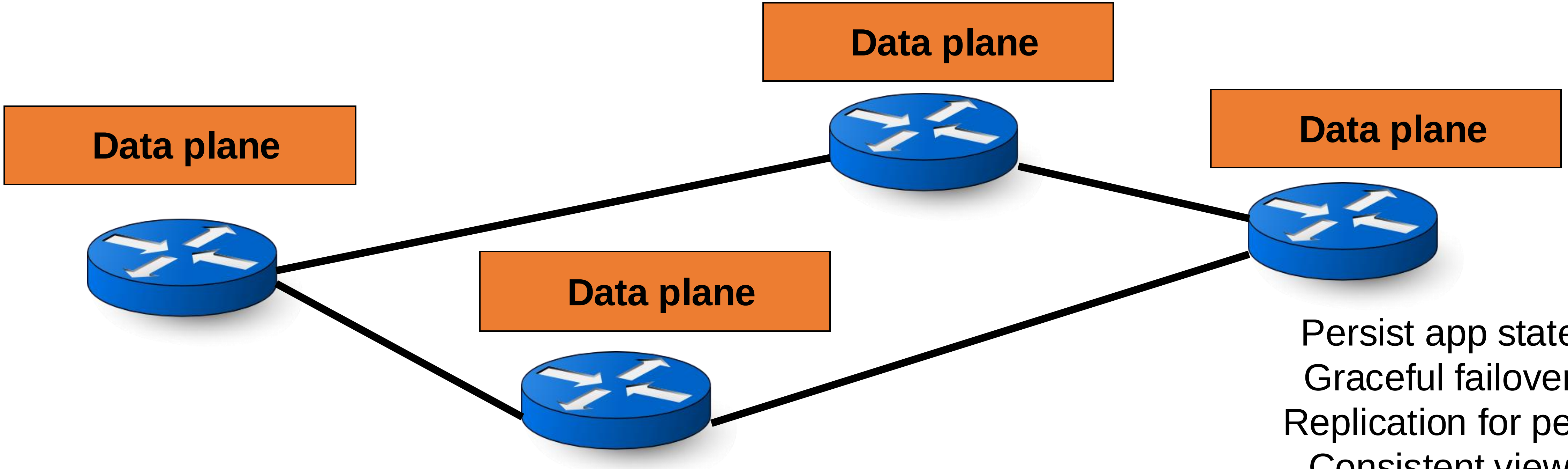
(3) Formal verification of network policy



(4) Unified network operating system

Application	Application	Application
Network Operating System		

Separate distributed system concerns from expressing intent



Persist app state
Graceful failover
Replication for perf
Consistent view

New technical challenges of SDN

- Availability: surviving failures of the controller
- 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; …

Legacy?

- Openflow is just a protocol. The details can change or become irrelevant, but the philosophy is longer-lasting
- Programming software switches: Match-action abstraction common; programmable hardware switches common
- OVS modules available for the Linux kernel
- P4: protocol independence and stateful behavior in switches
  - In-network computing