On integrating Software-Defined Networking within existing routing systems

Laurent Vanbever

Princeton University

Google, Mountain View

November, 15 2013

On integrating Software-Defined Networking within existing routing systems

1 **SDN-controlled routers**
don’t trash, recycle

2 **SDN-controlled IGP**
fine-grained traffic-engineering

3 **SDN-controlled BGP**
inter domain bonanza

On integrating Software-Defined Networking within existing routing systems

1 SDN-controlled routers

don't trash, recycle

SDN-controlled IGP

fine-grained traffic-engineering

SDN-controlled BGP

inter domain bonanza

Today's networks are managed indirectly

Given network-wide forwarding requirements

Traffic from *i* to *j* should flow along path *P1*

Traffic from *k* to *l* should flow along path *P2*

…

operators' job

Configure each equipment such that
they compute (locally) compatible forwarding entries

Today's networks are managed indirectly, device-by-device

Given network-wide forwarding requirements

Traffic from *i* to *j* should flow along path *P1*

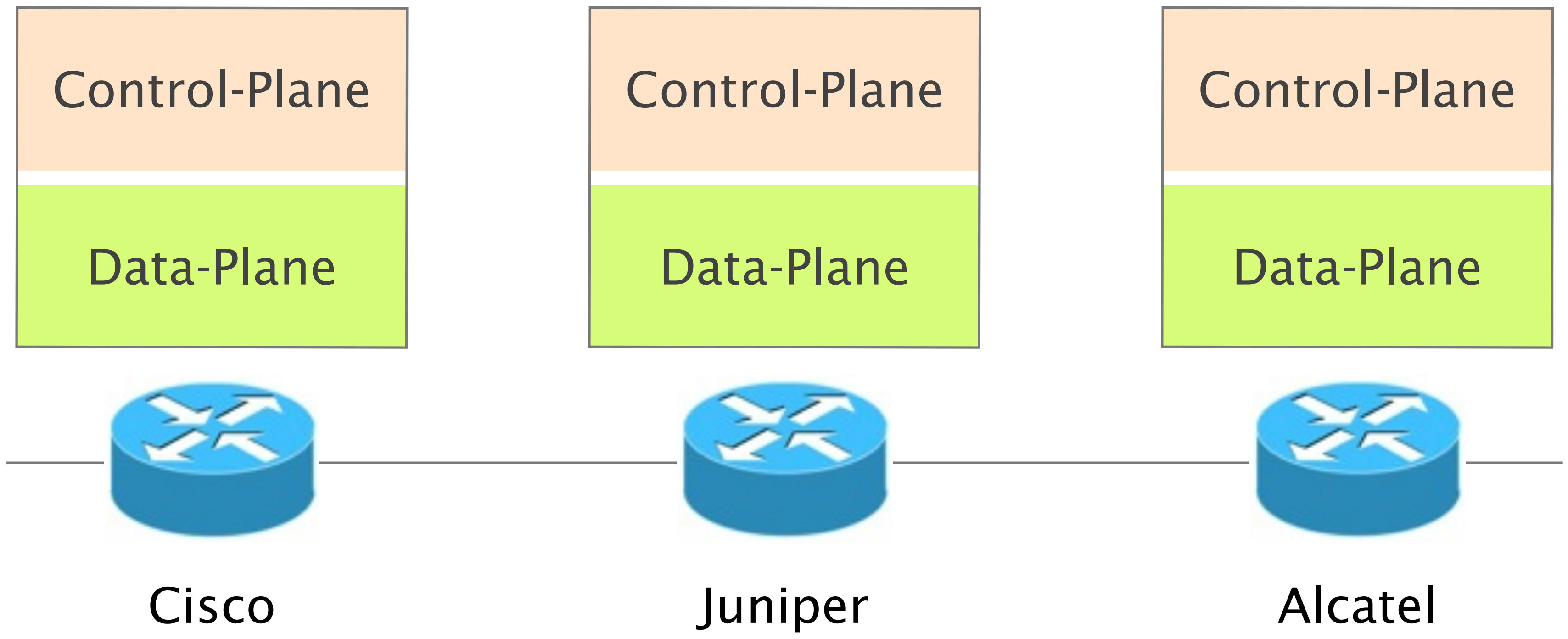
Traffic from *k* to *l* should flow along path *P2*

…

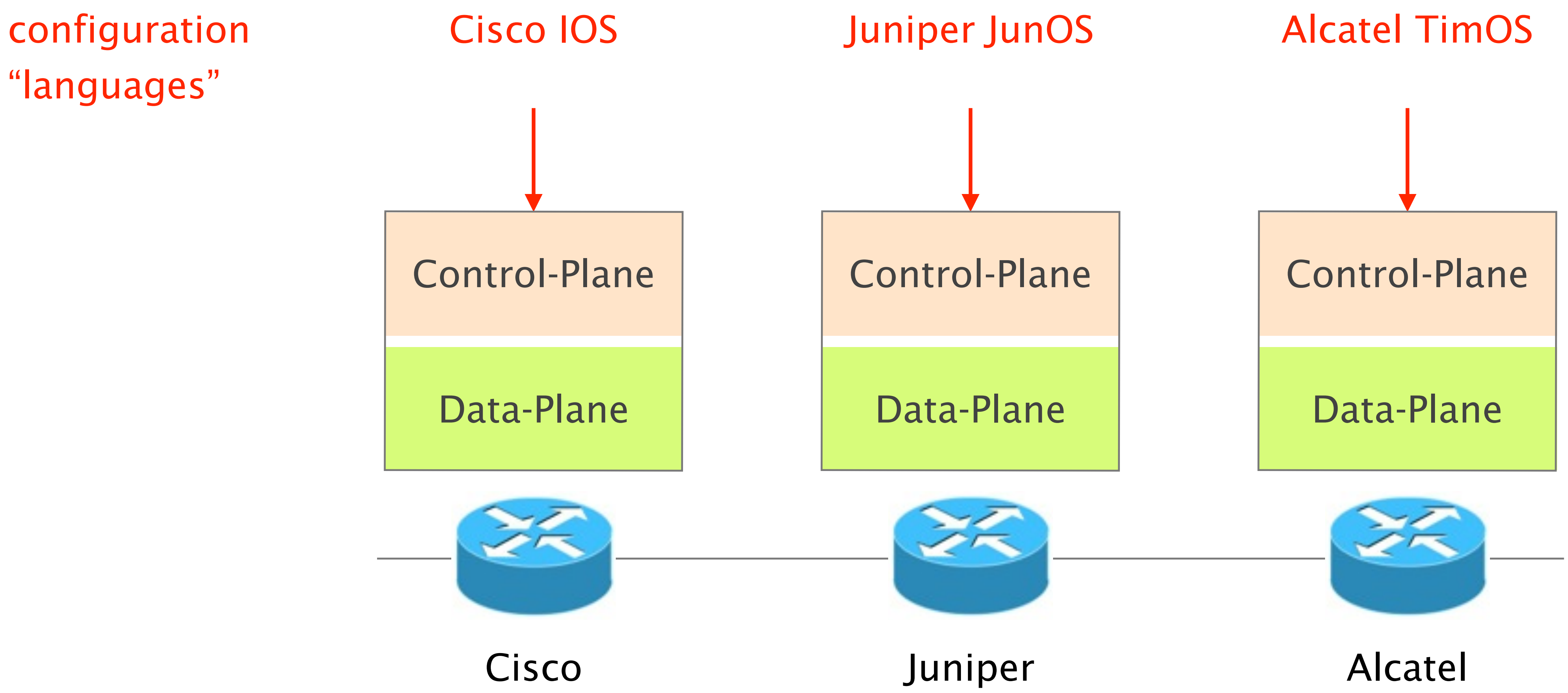
operators' job

Configure each equipment such that they compute (locally) compatible forwarding entries

Today's networks are managed indirectly, device-by-device, using arcane configuration languages

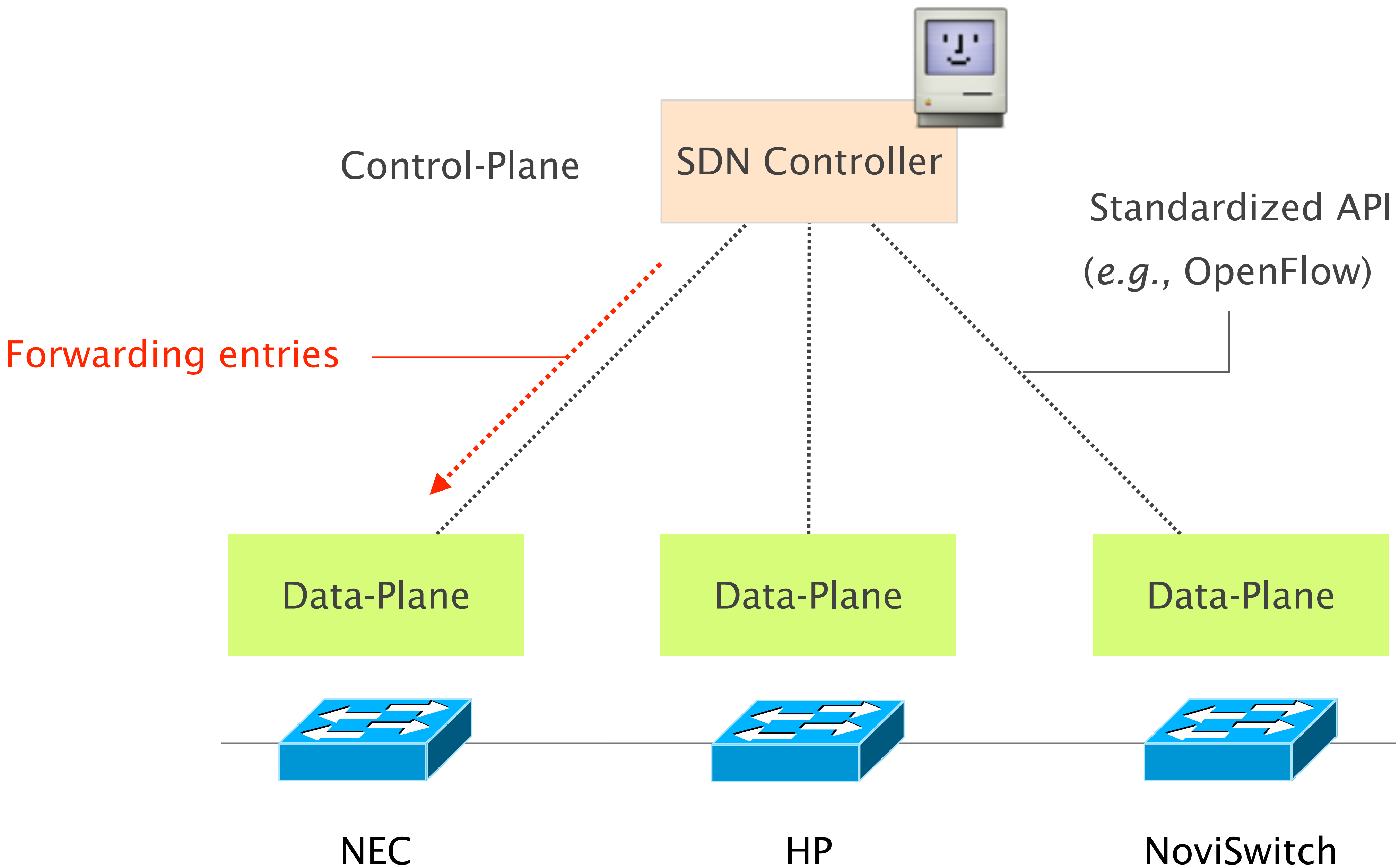


Today's networks are managed indirectly, device-by-device, using arcane configuration languages



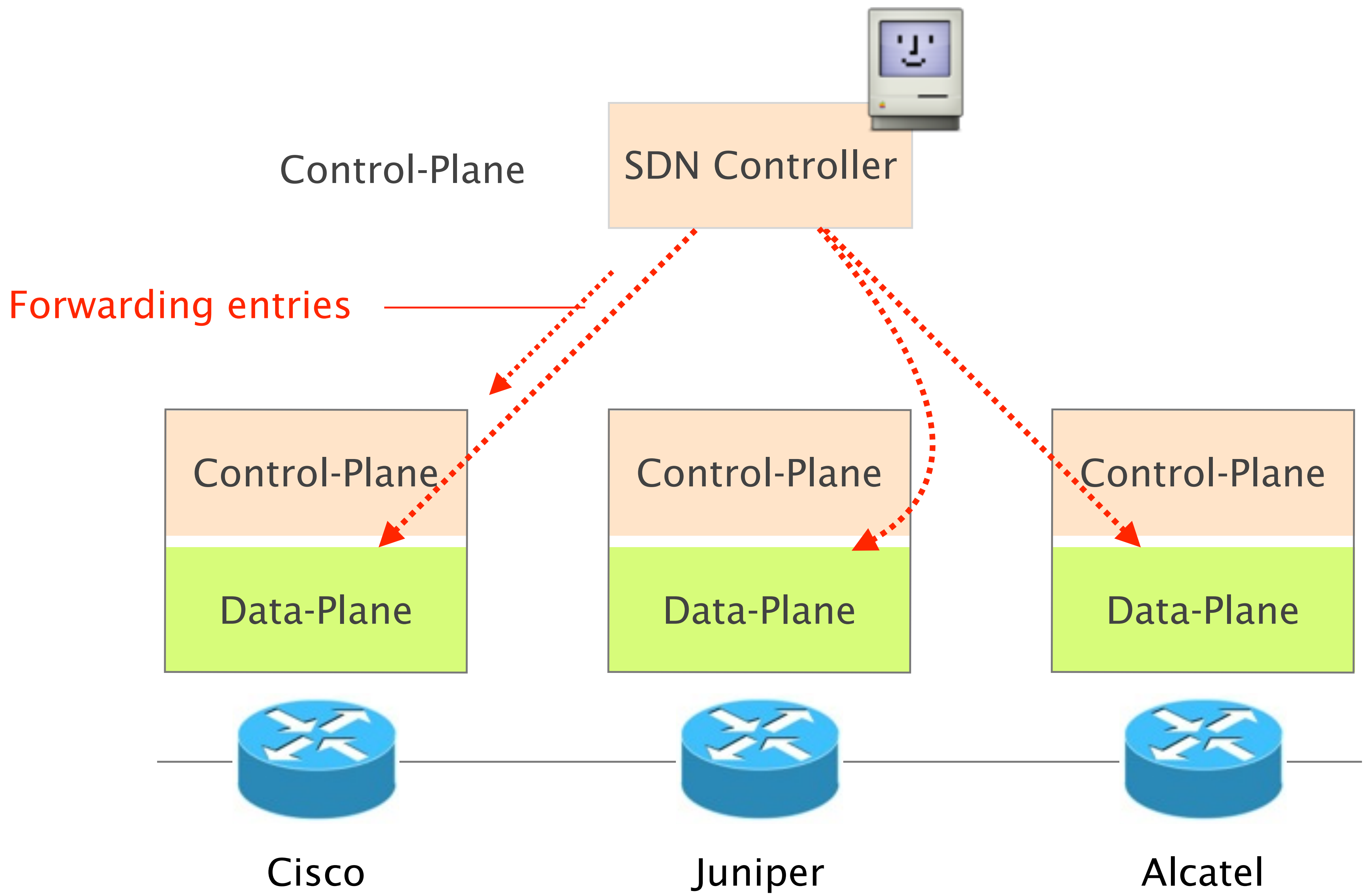
In contrast, SDN simplifies network management…

…by directly programming forwarding entries, using a logically-centralized controller and an open API

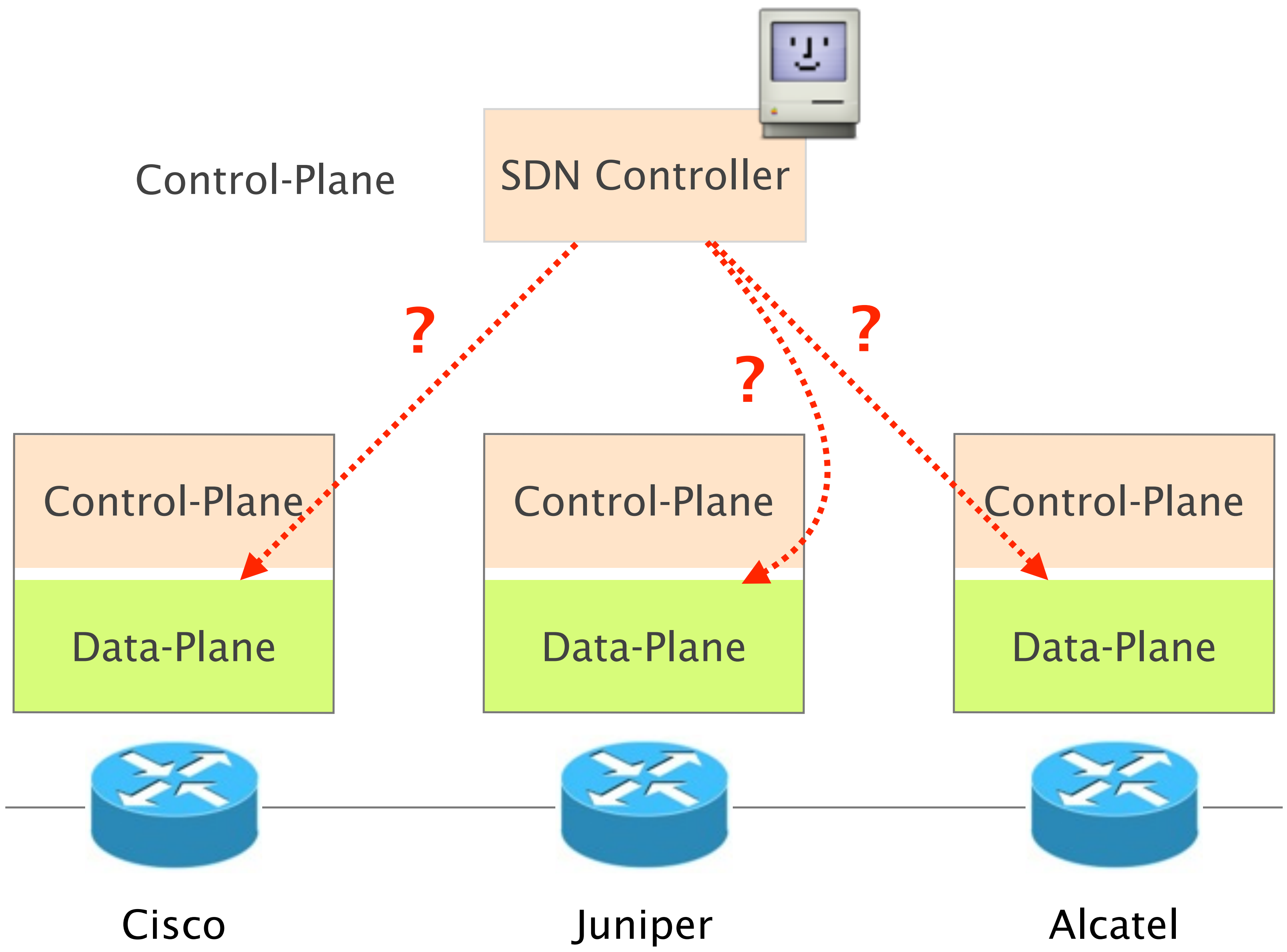


The bad news is that SDN requires compatible devices…

Wouldn’t it be great to manage an *existing* network “à la SDN”?



To do that, we need an open API to program forwarding entries in a router



Routing protocols are good candidates to act as API

Routing protocols

- messages are standardized

  all boxes must speak the same language

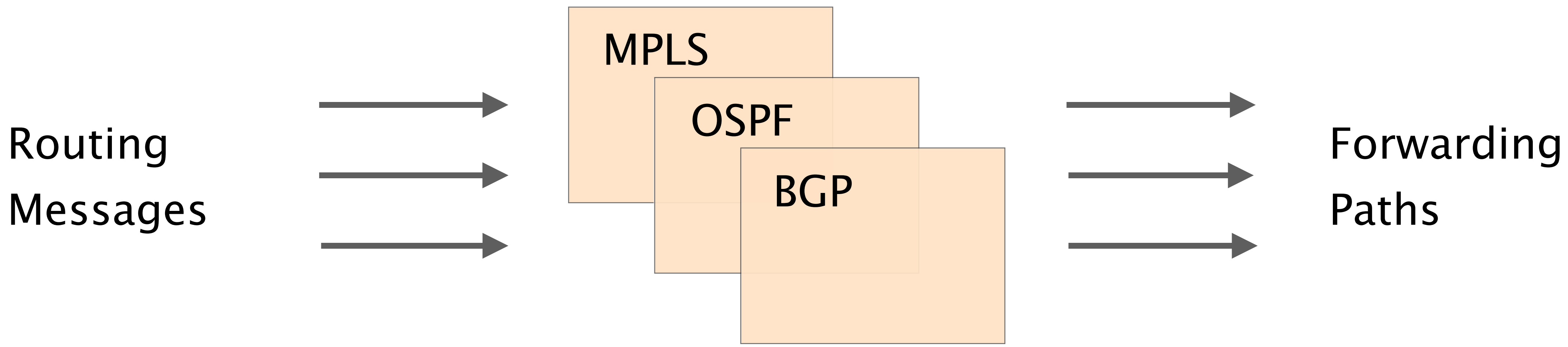
- behaviors are well-defined and understood

  e.g., shortest-path routing

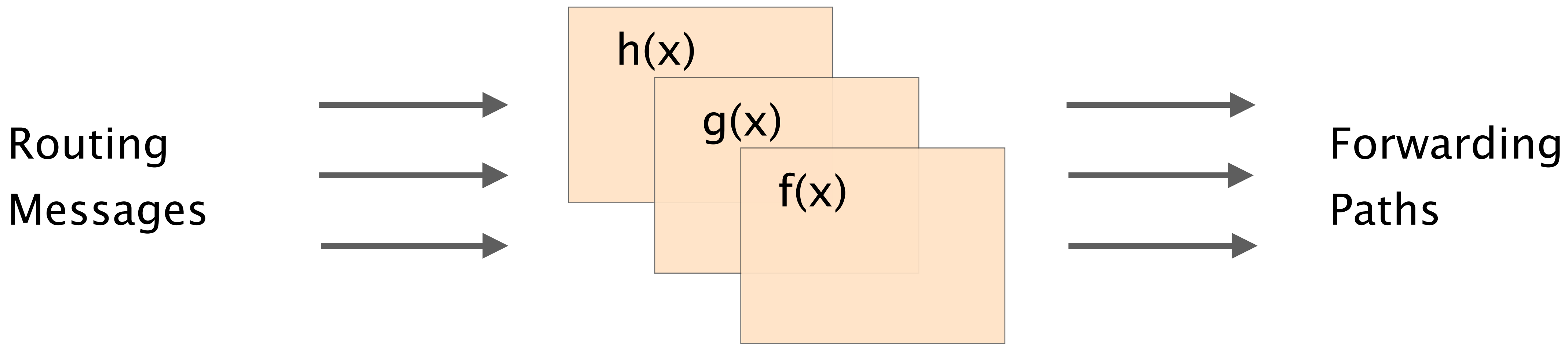
- implementations are widely available

  a vast majority (if not all) Cisco boxes supports OSPF

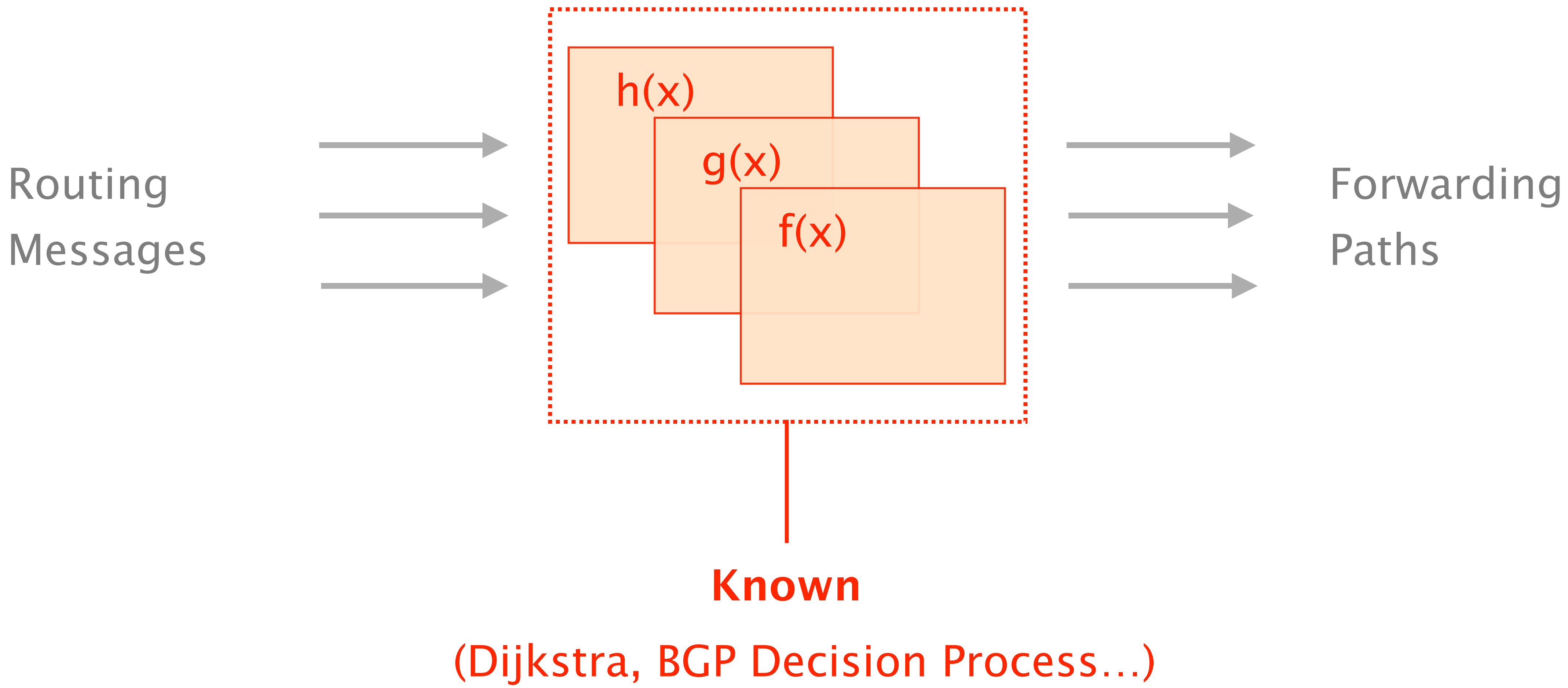
A routing protocol takes routing messages as input and computes forwarding paths as output



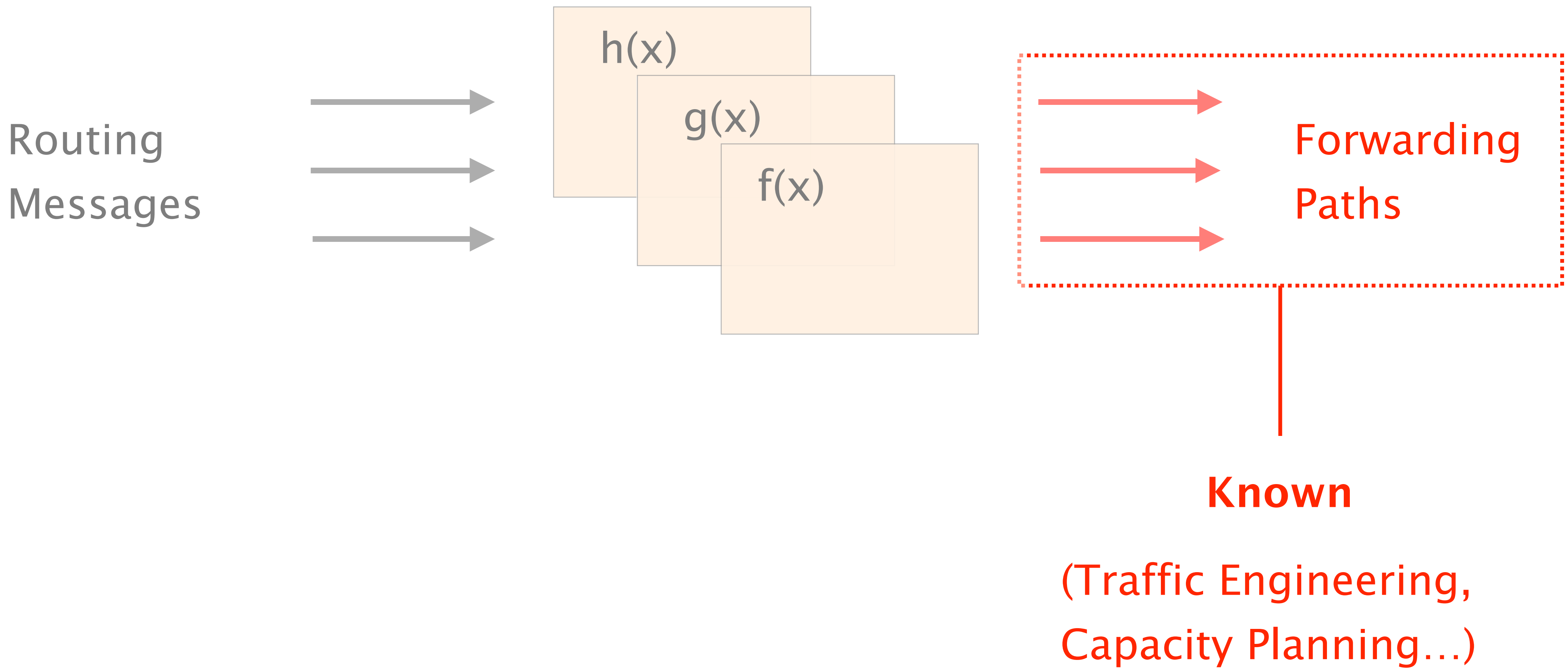
A routing protocol is thus a function from input messages to forwarding paths



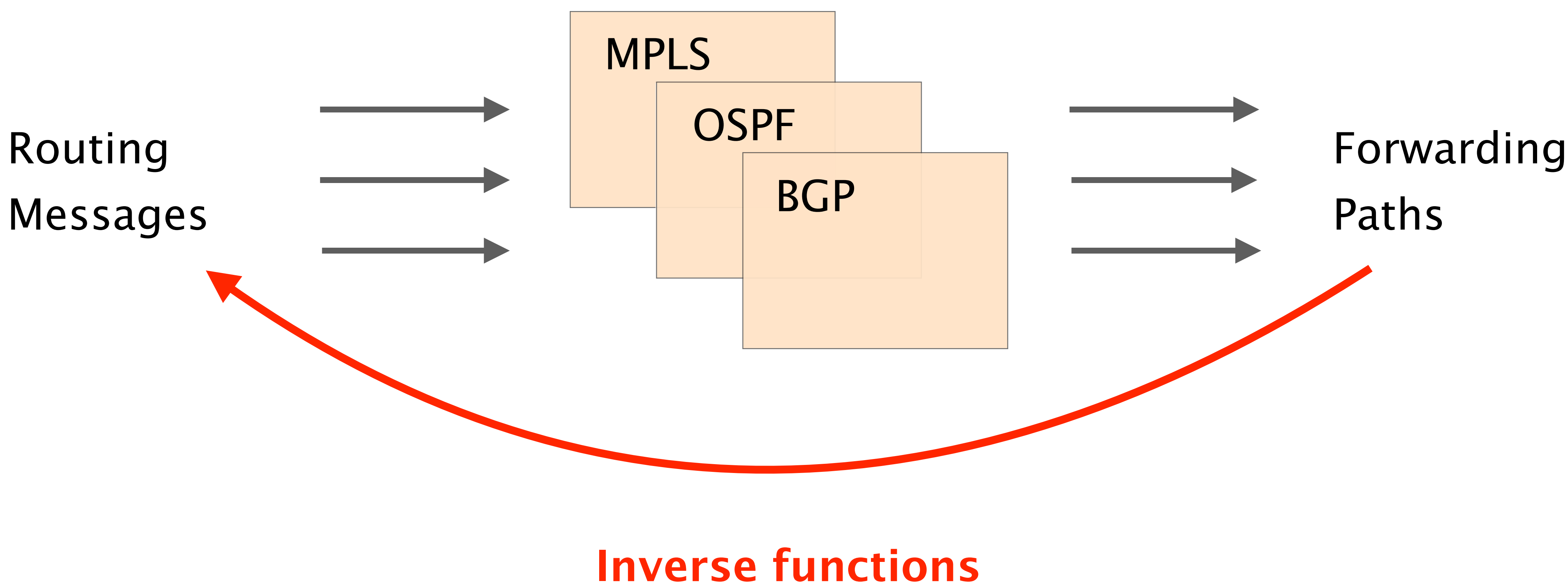
Functions are well known



Forwarding paths are also known, from network-wide requirements



Given a forwarding path and a function (*i.e.*, protocol), can we automatically find the corresponding input?



The type of input to be computed depends on the routing protocol

	Type	Algorithm	Input
IGP	Link-State	Dijkstra	Network topology
BGP	Path-Vector	Decision Process	Received routes

On integrating Software-Defined Networking within existing routing systems

SDN-controlled routers

don't trash, recycle

2 SDN-controlled IGP

fine-grained traffic-engineering

SDN-controlled BGP

inter domain bonanza

Joint work with

Stefano Vissicchio, Olivier Bonaventure, Jennifer Rexford

Traffic Engineering techniques differ in terms of ease of use, functionality and support

	IGP (link reweight)	MPLS (RSVP-TE)	SDN
signaling			
expressiveness			
device support			

Traffic Engineering techniques differ in terms of ease of use, functionality and support

	IGP (link reweight)	MPLS (RSVP-TE)	SDN
signaling	no	yes	no
expressiveness	low shortest path	high	high
device support	excellent	good require MPLS	poor new hardware

In a SDN-controlled IGP, a controller presents a virtual topology to the routers to force them to use given paths

Given a set of forwarding paths,

augment an IGP topology with virtual

- nodes
- links and weights
- destinations

such that routers compute compatible paths

SDN-controlled IGP combines the benefits of each technique

SDN-controlled IGP

signaling

expressiveness

device support

SDN-controlled IGP combines the benefits of each technique

	SDN-controlled IGP
signaling	no
expressiveness	high
device support	excellent

SDN-controlled IGP combines the benefits of each technique

	SDN-controlled IGP
signaling	no
expressiveness	**high**
device support	excellent

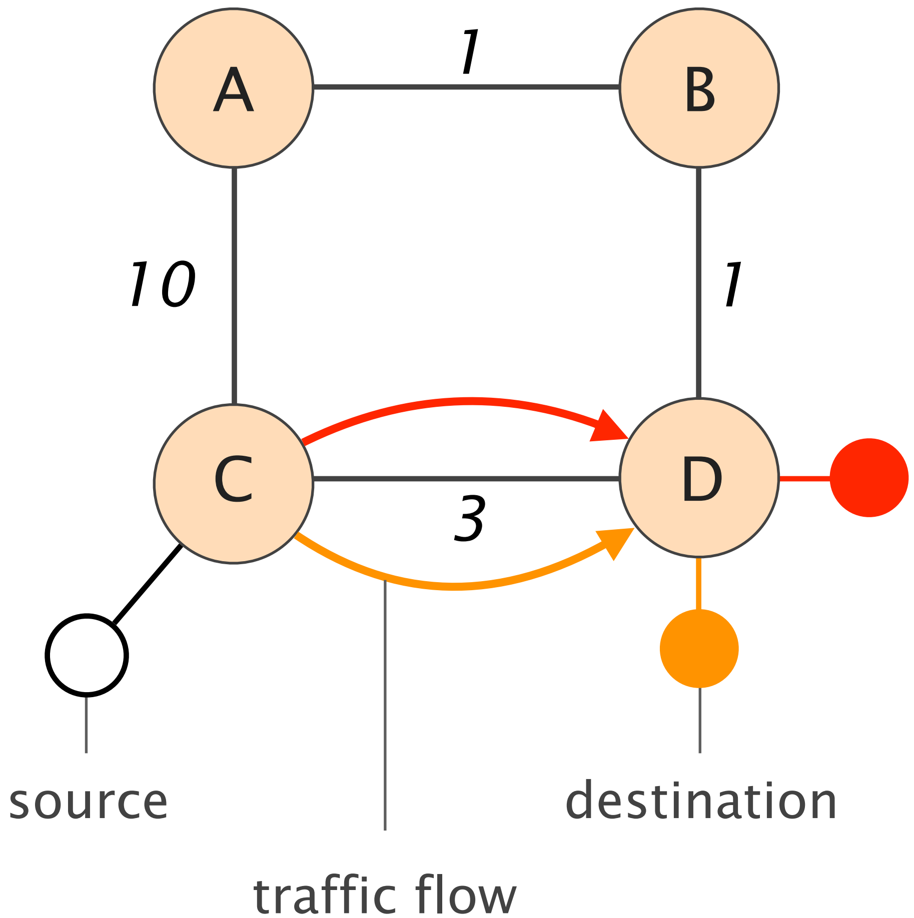
SDN-controlled IGP enables fine-grained IP traffic control

SDN-controlled IGP enables to

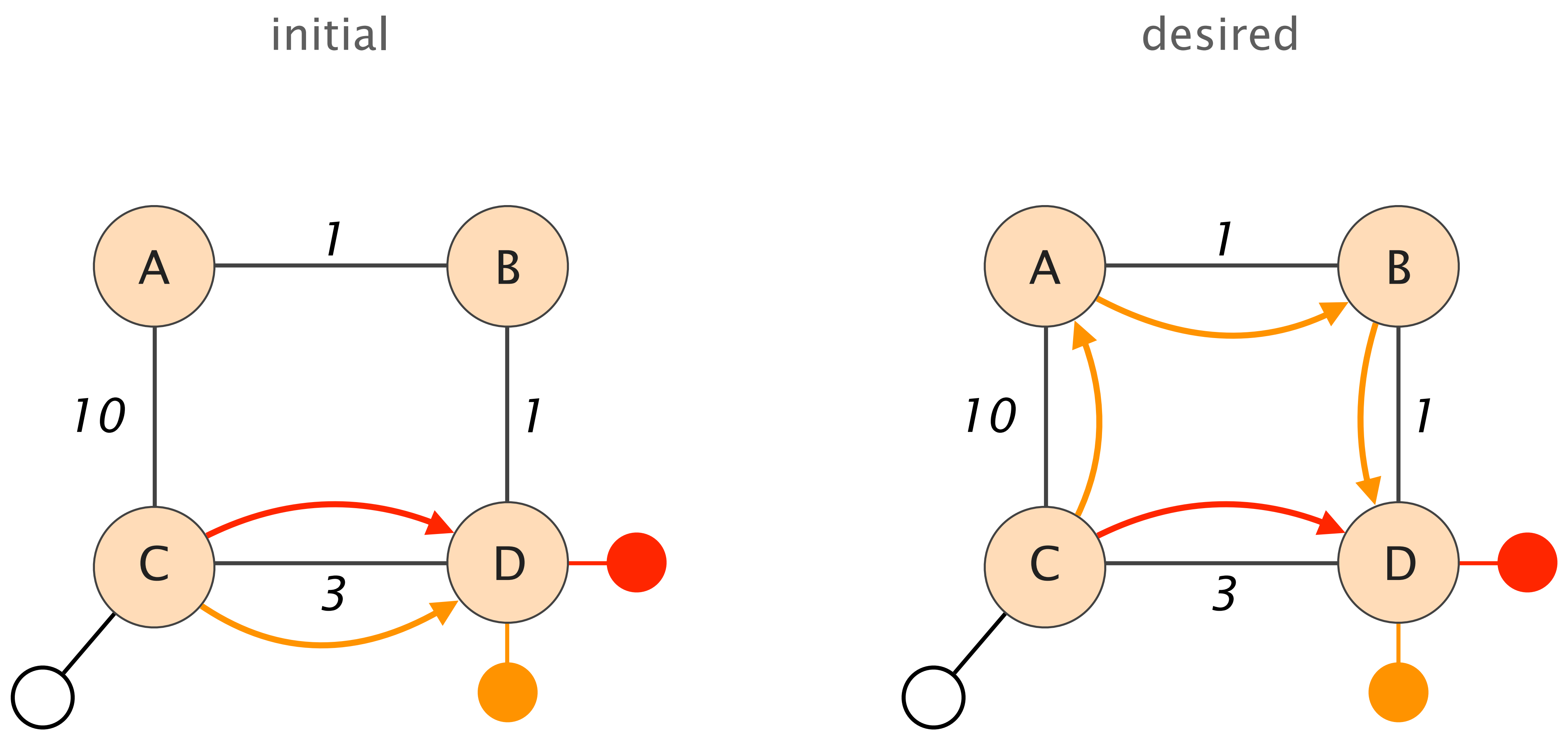
- steer traffic on non-shortest paths
- create ECMP paths (on a per-destination basis)
- provision backup paths

in a centralized manner, on existing network

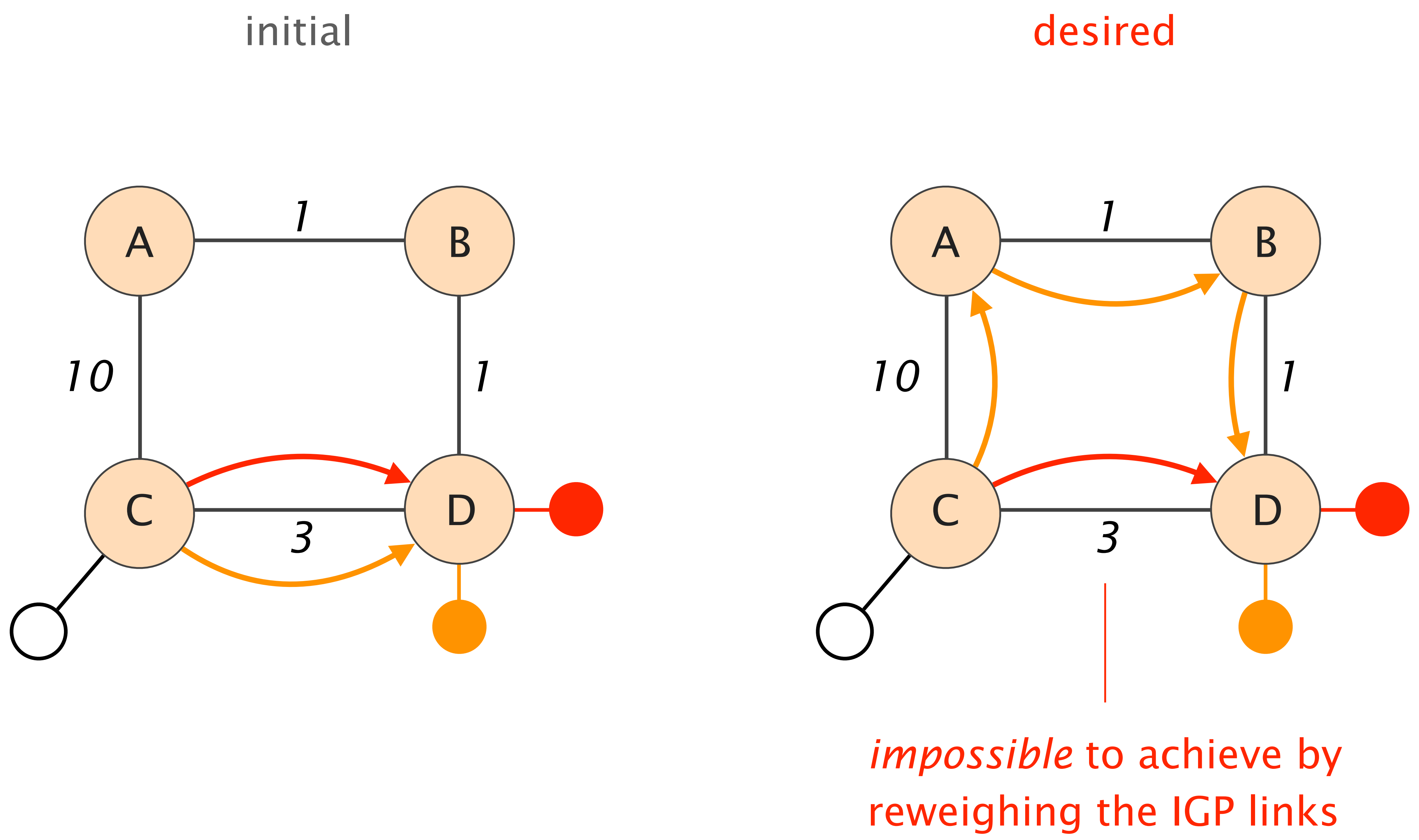
Consider this network where a source sends traffic to 2 destinations



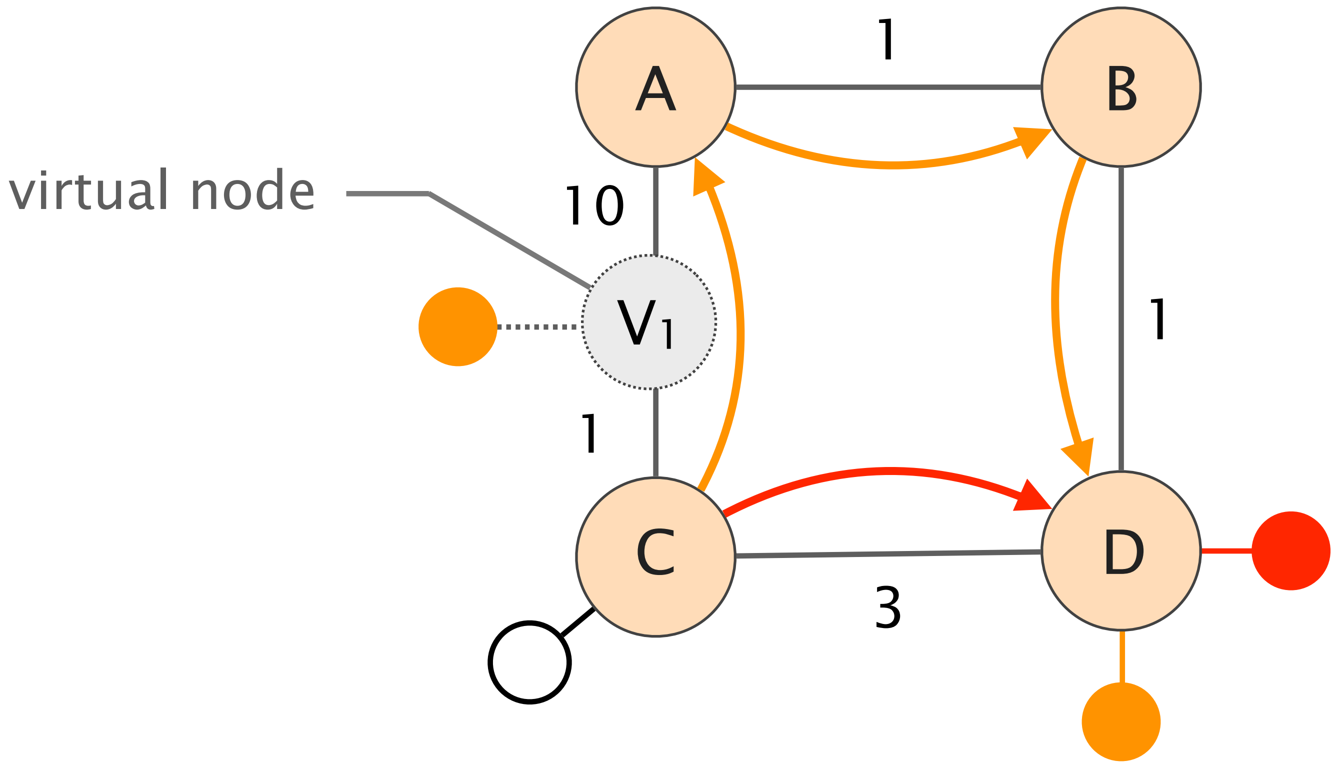
As congestion appears on the *(C,D)* link, operators might want to move away the orange flow to A



Moving only the orange flow to A is impossible with an IGP as both destinations are connected to D



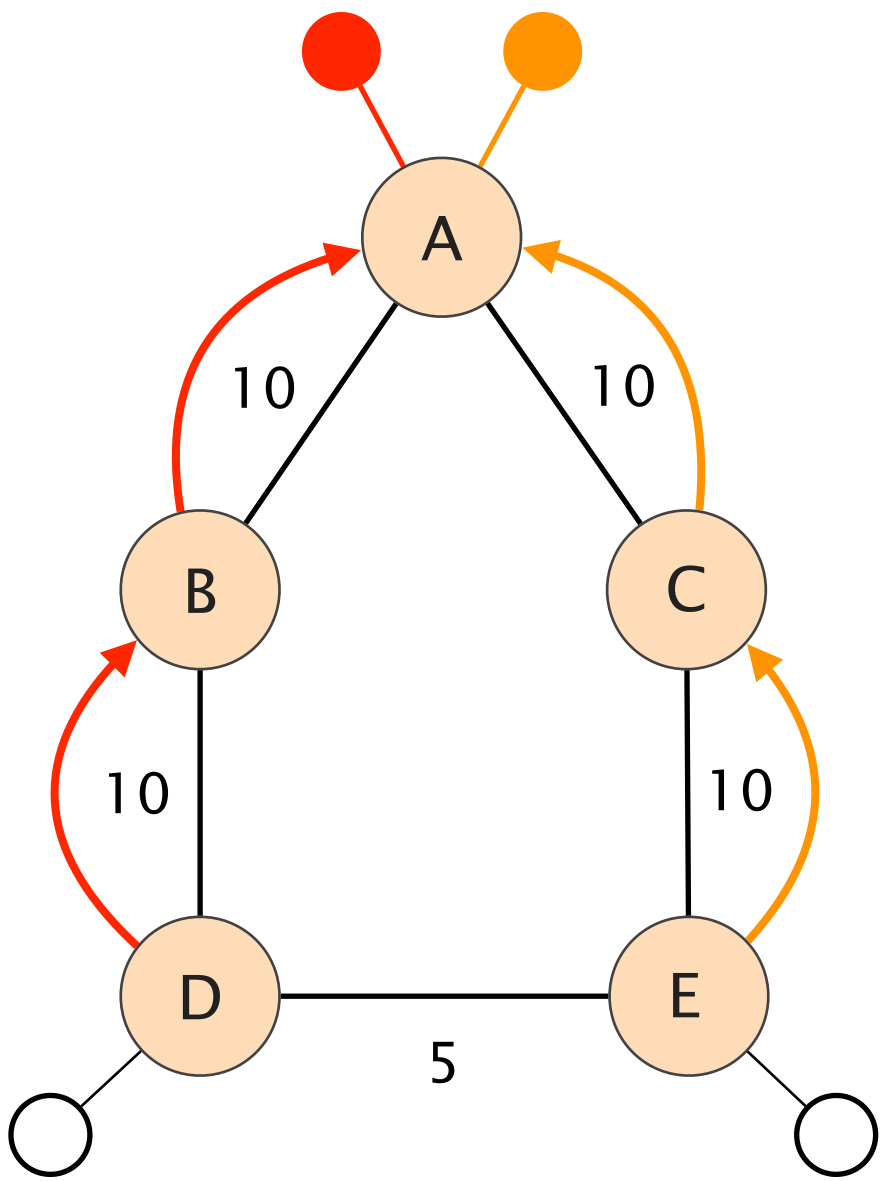
We can attract the orange flow from C by adding a virtual node announcing the orange destination



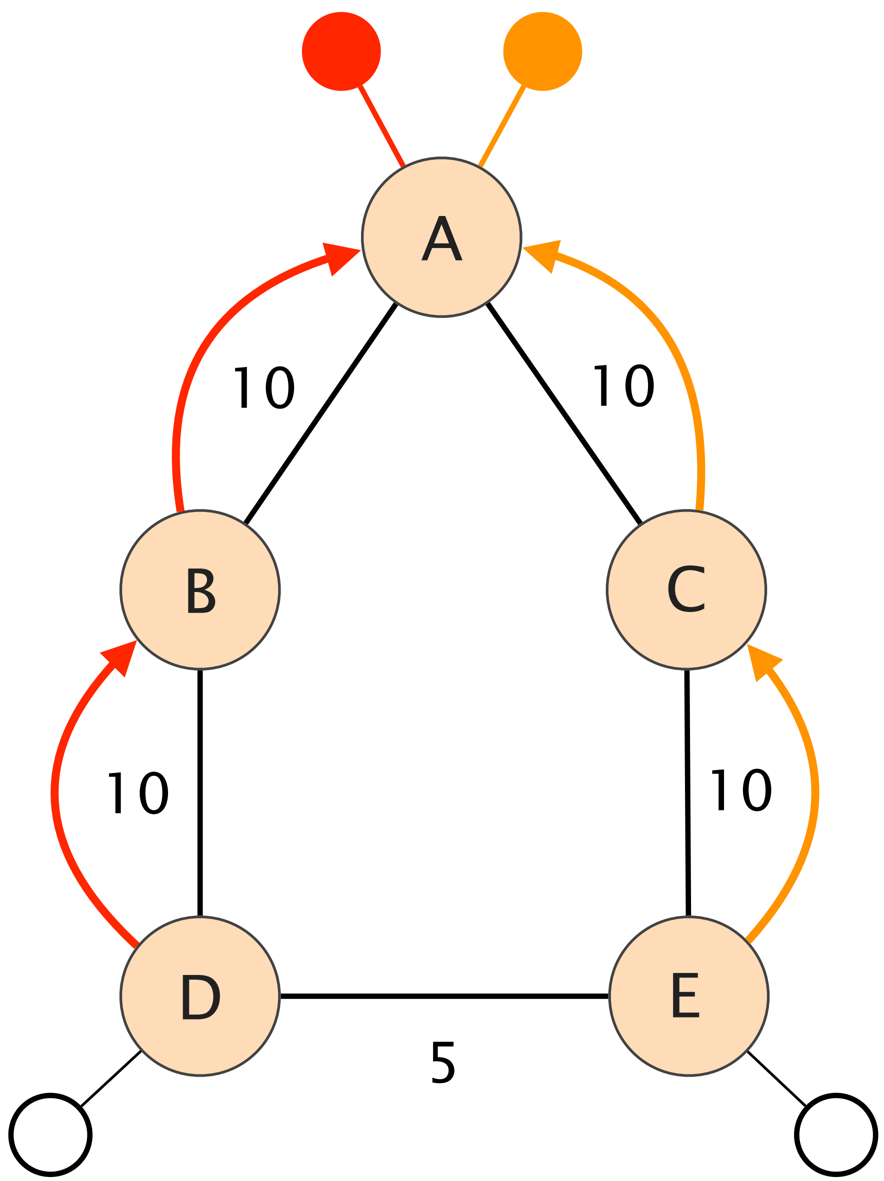
Traffic to V_1 is physically sent to A

Consider another network

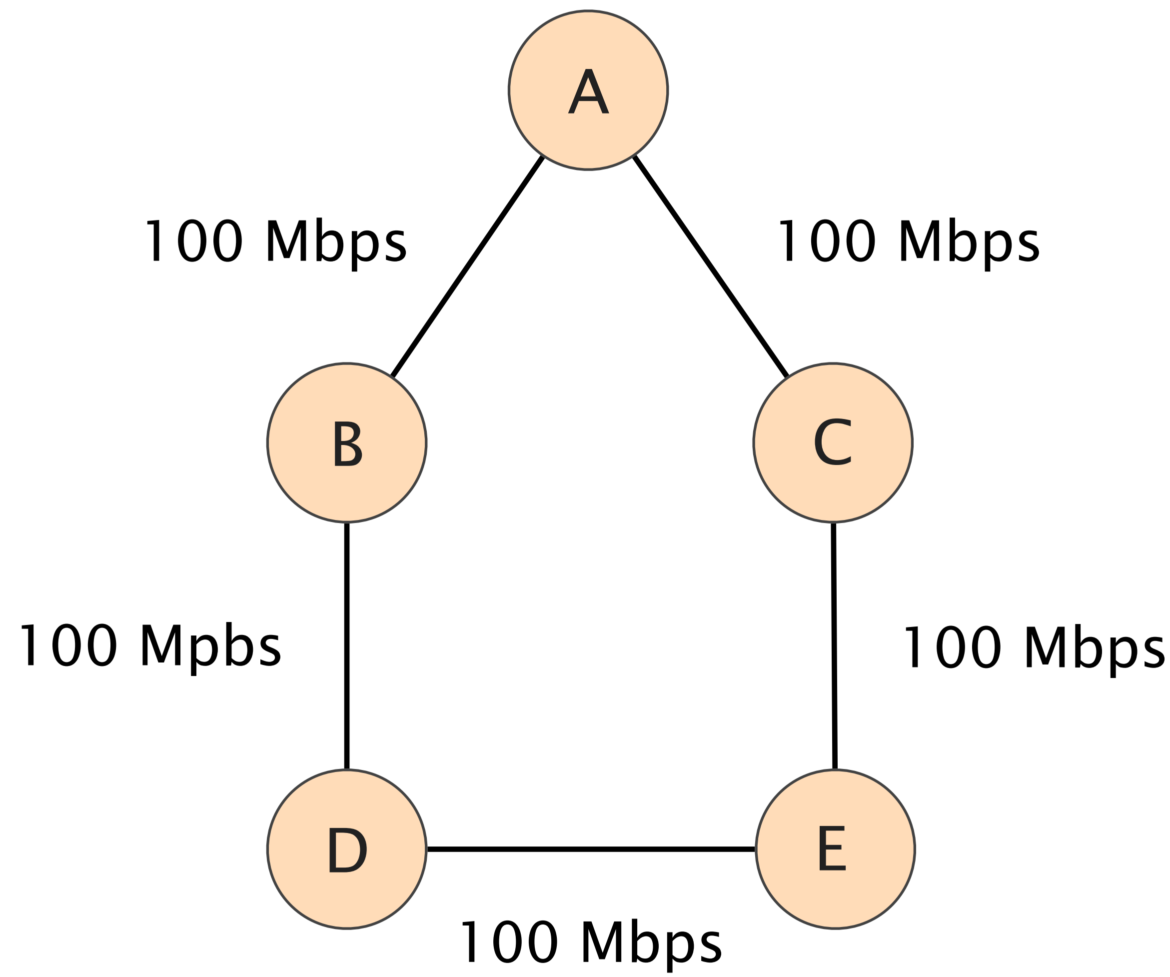
with 2 sources and destinations



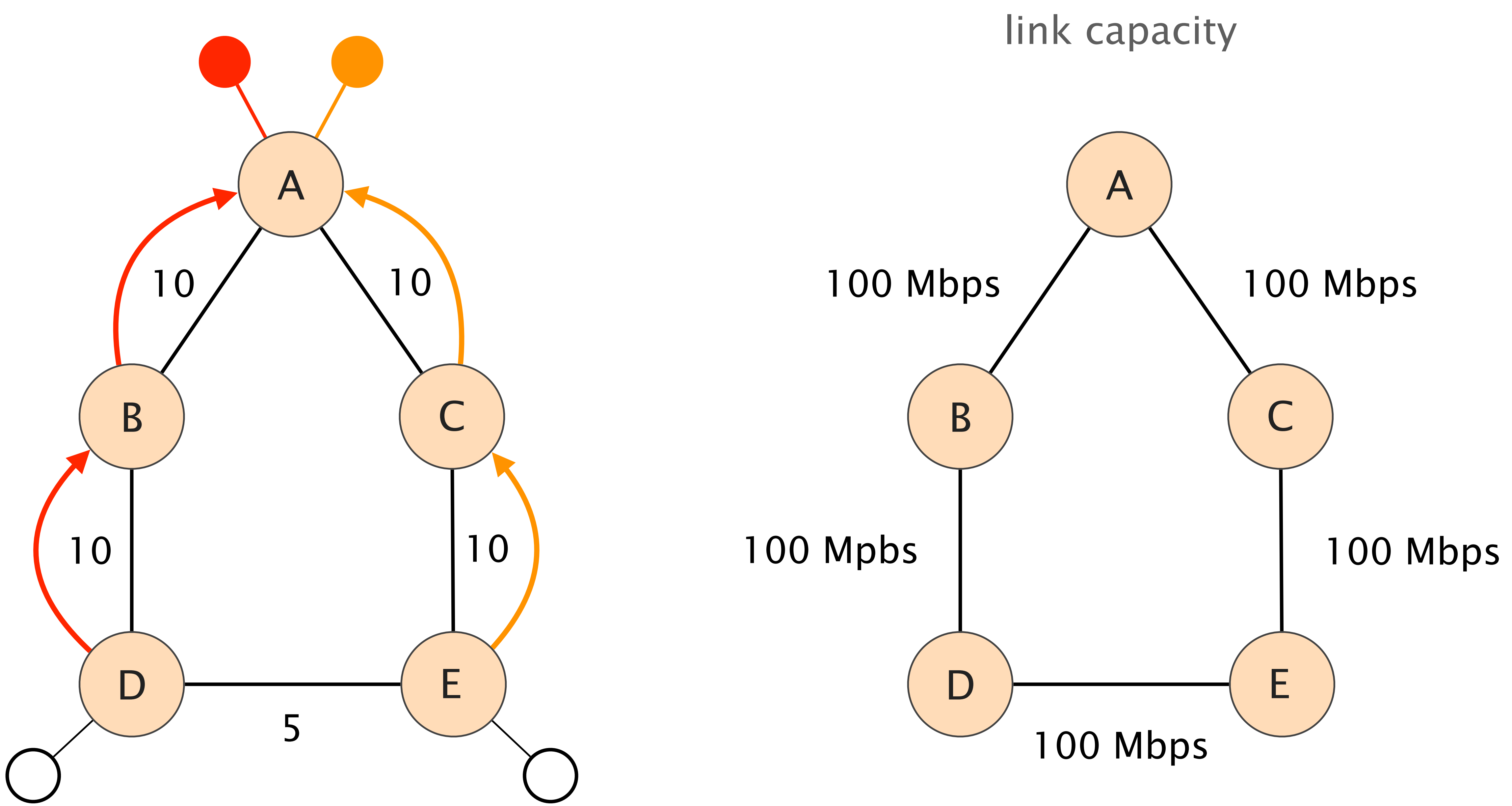
Consider another network with 2 sources and destinations



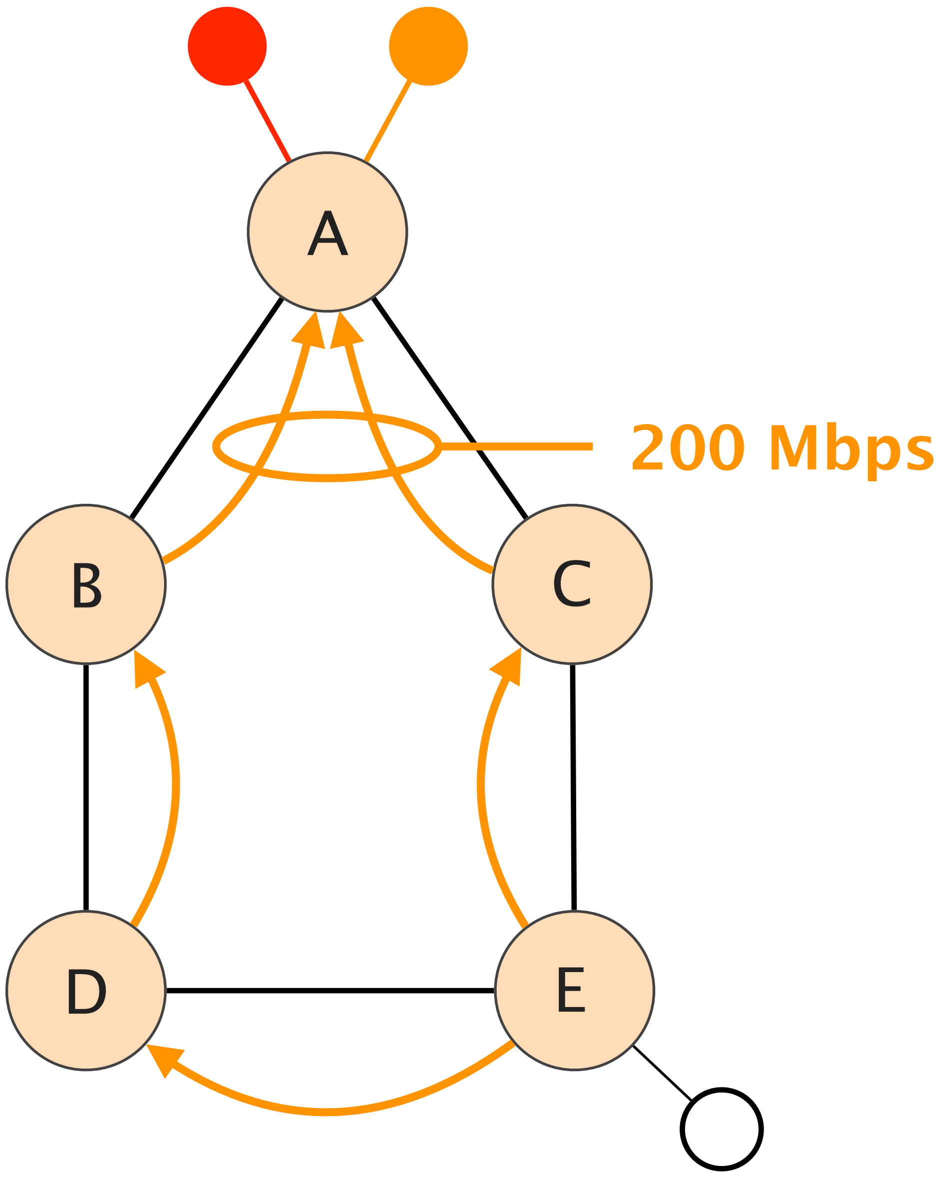
link capacity



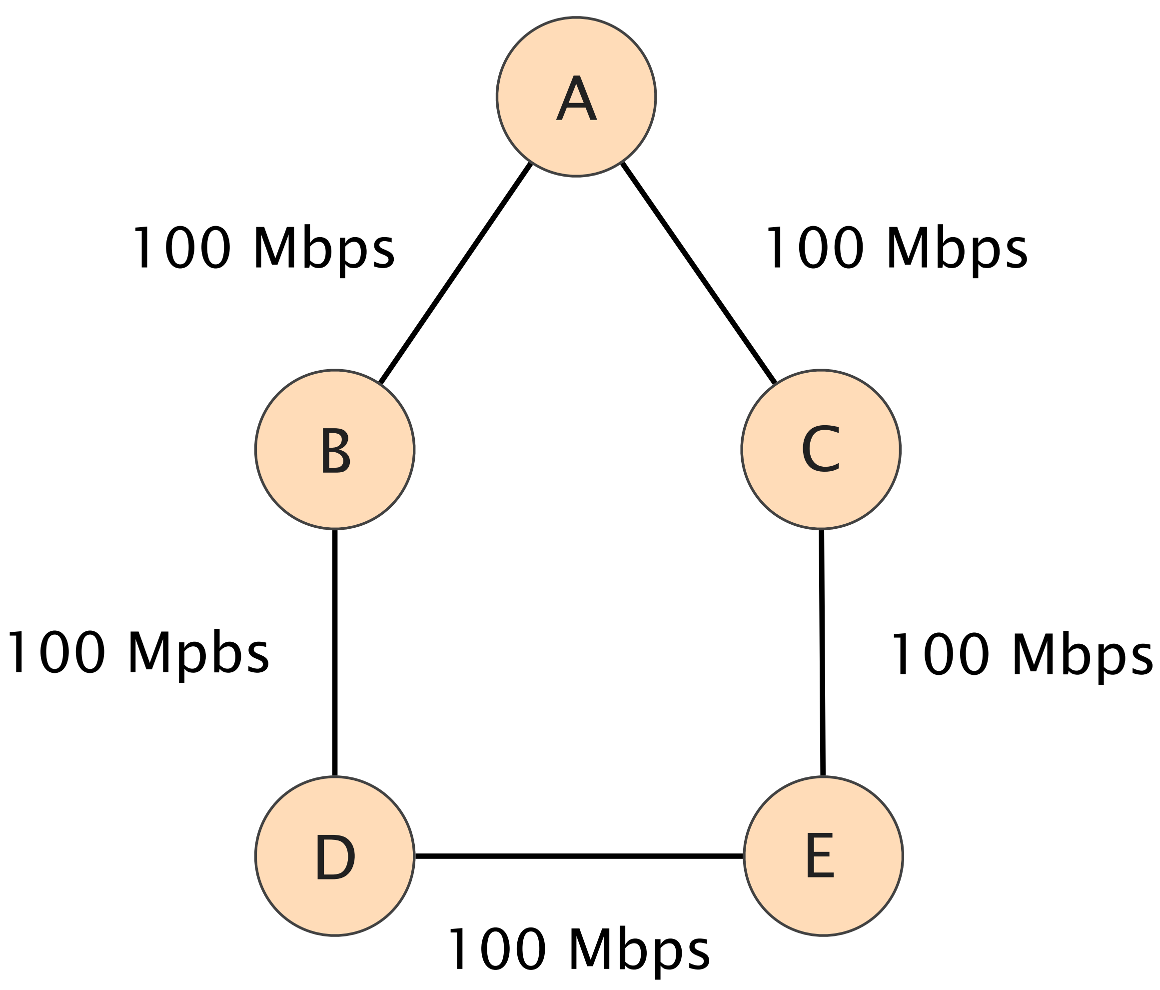
The red and orange flows are limited to 100Mbps



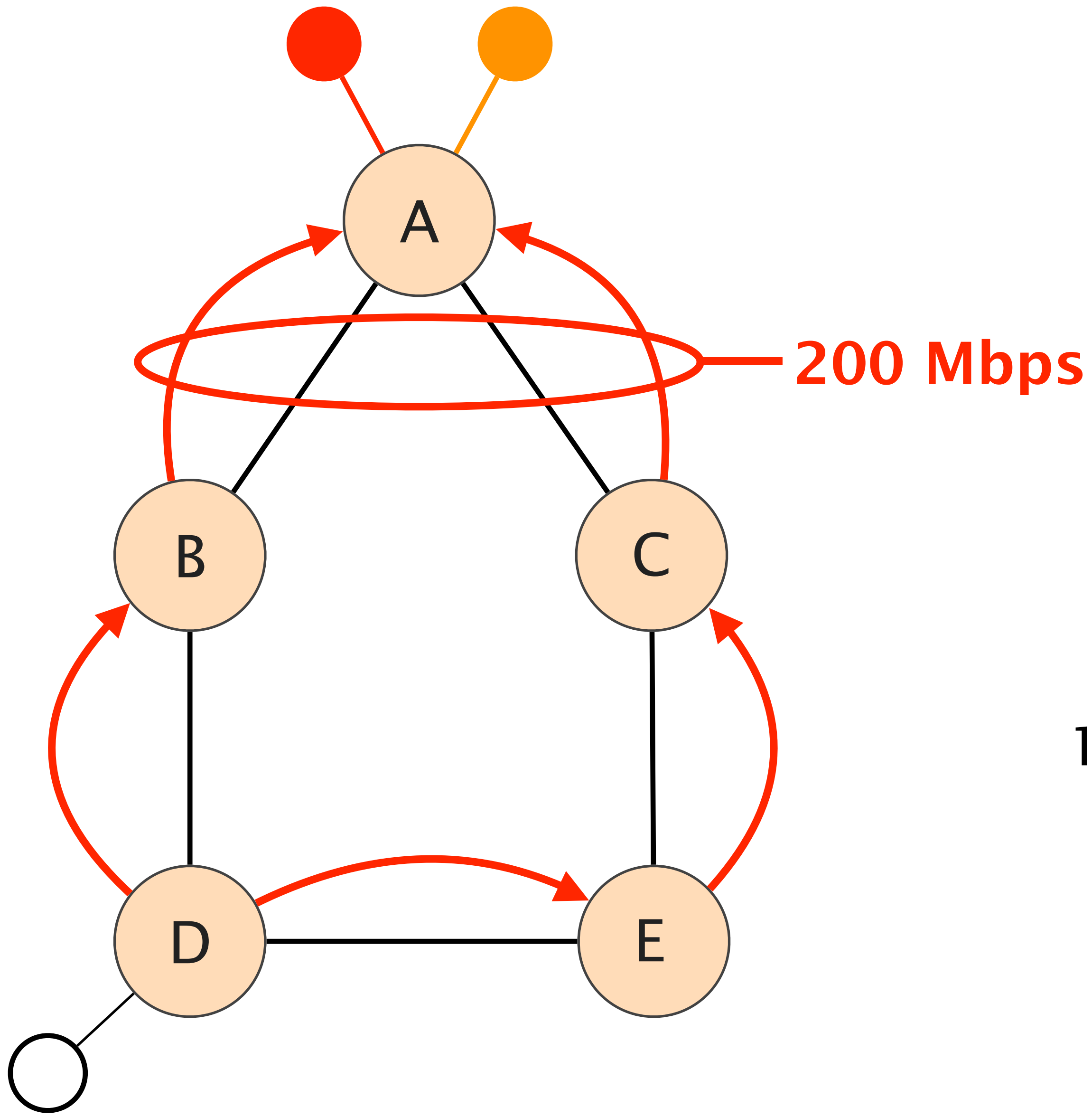
If the two flows do not overlap all the time, using ECMP would enable each flow to use 200Mbps



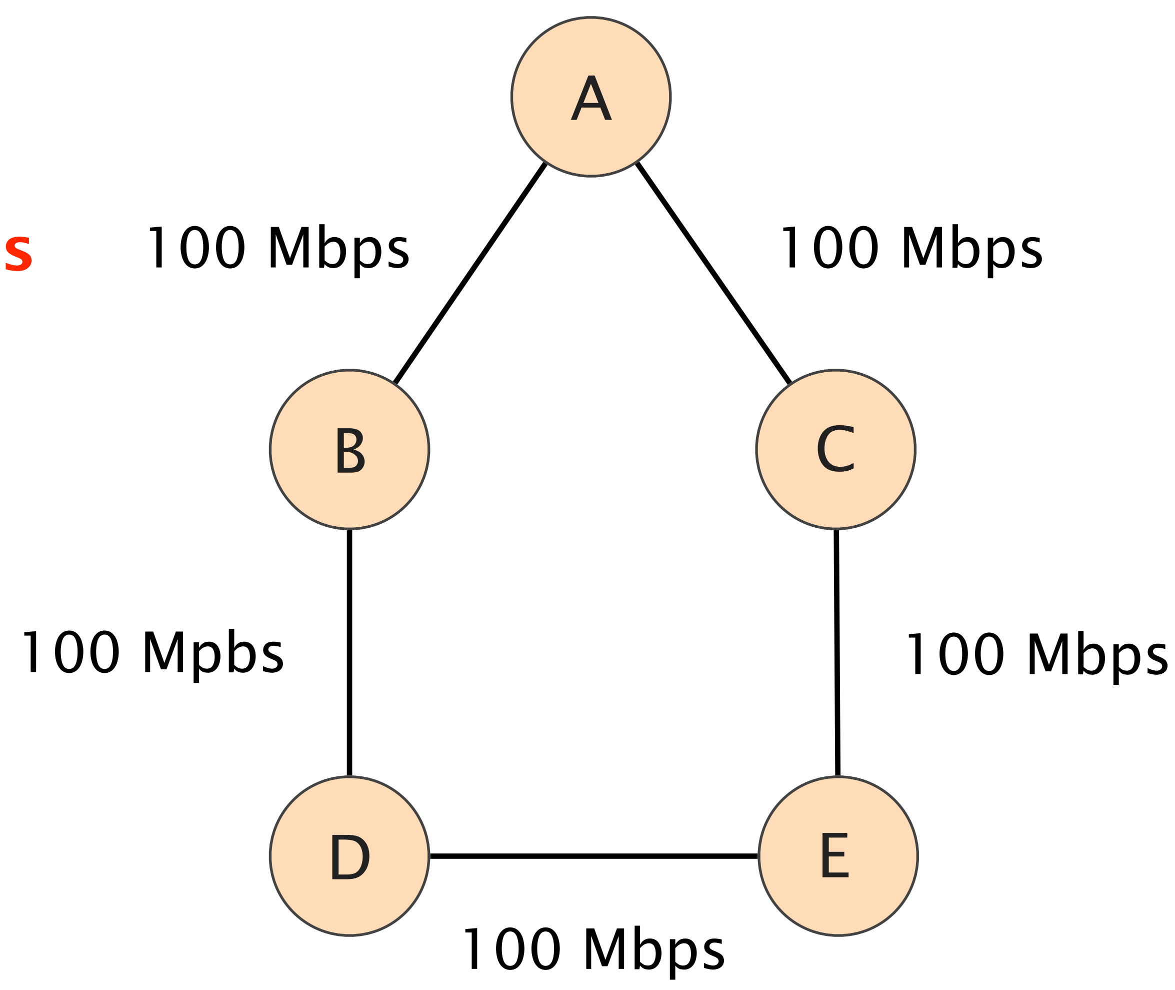
link capacity



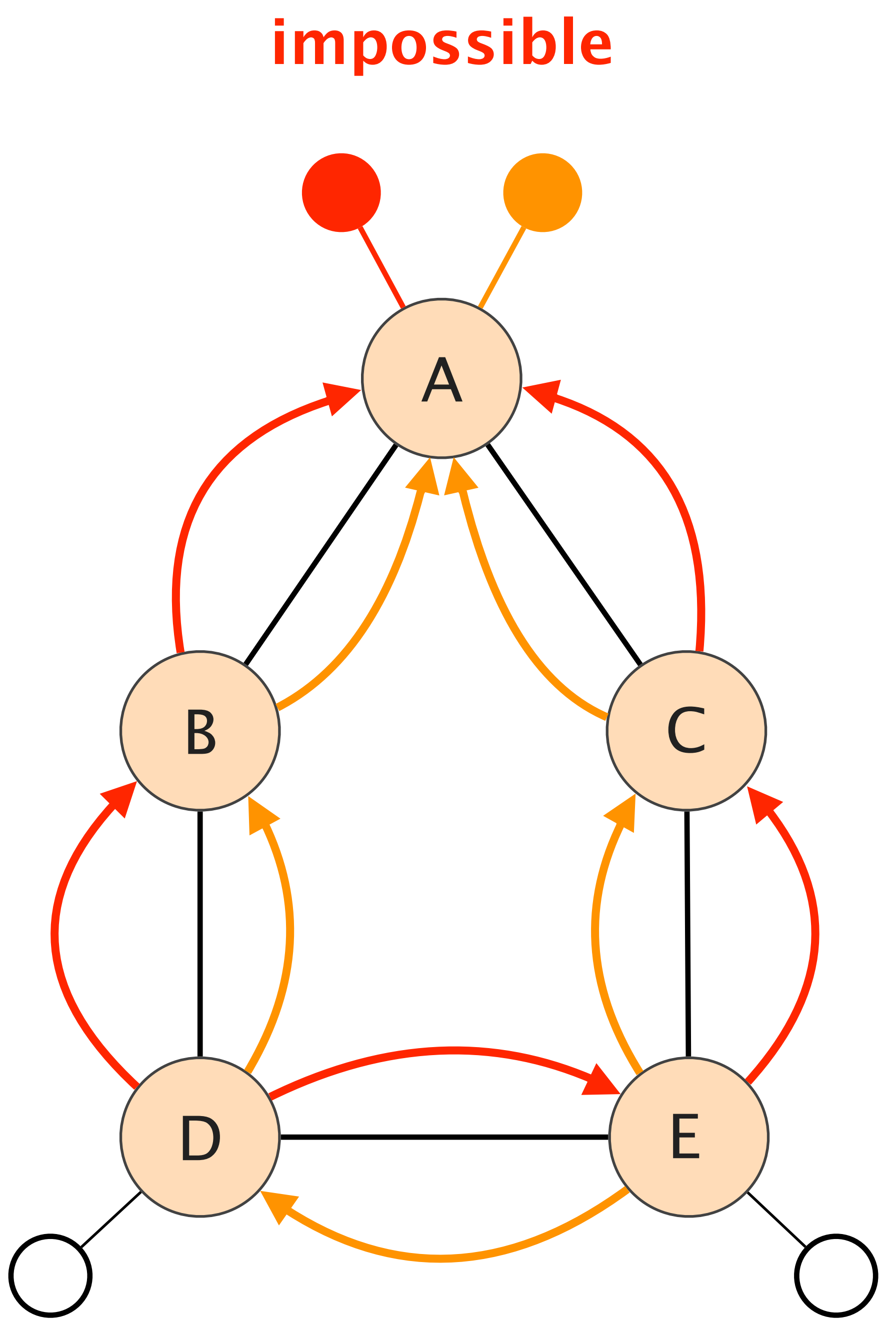
If the two flows do not overlap all the time,
using ECMP would enable each flow to use 200Mbps



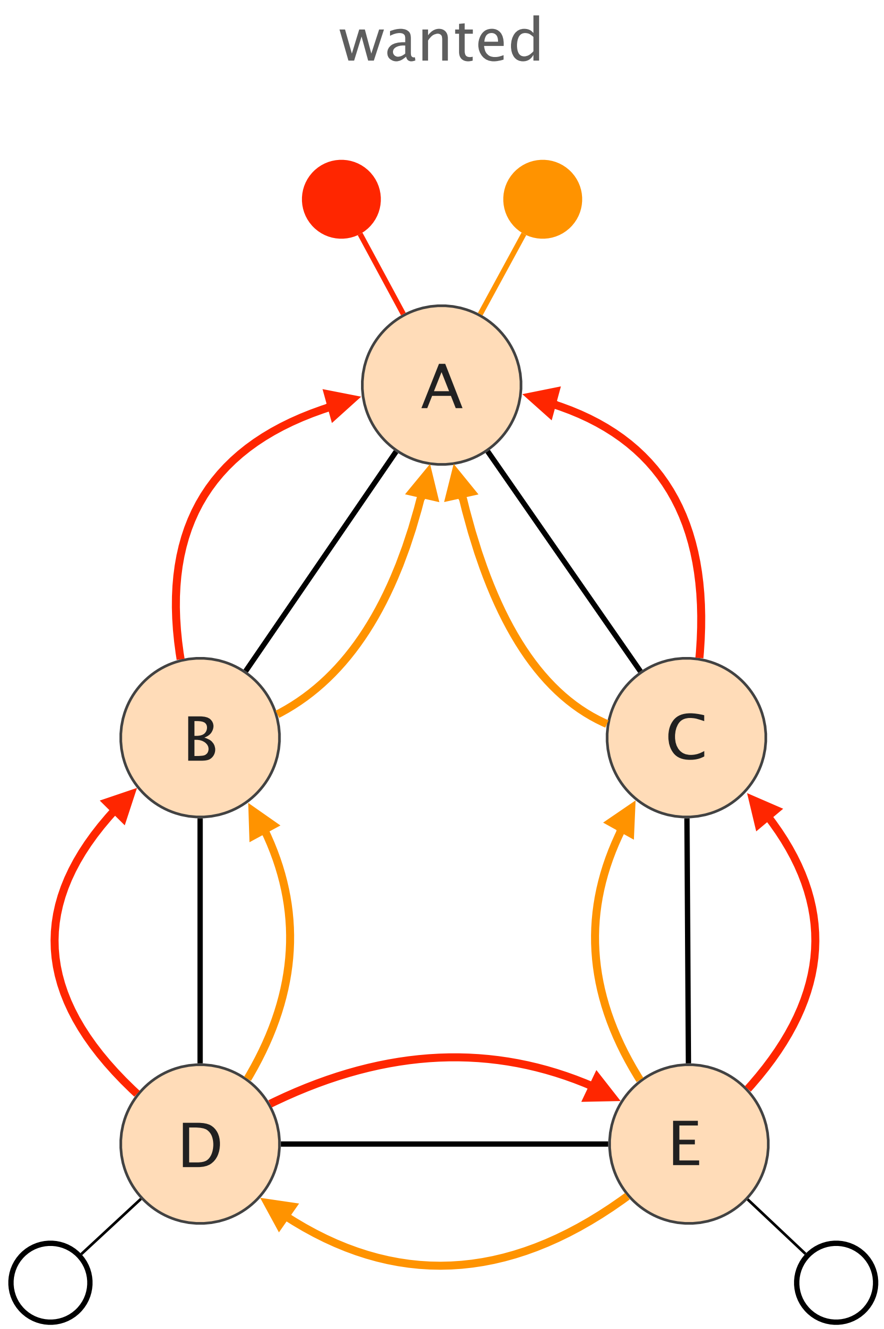
link capacity



Unfortunately, this is impossible to do with an IGP as both destinations are connected to the same node

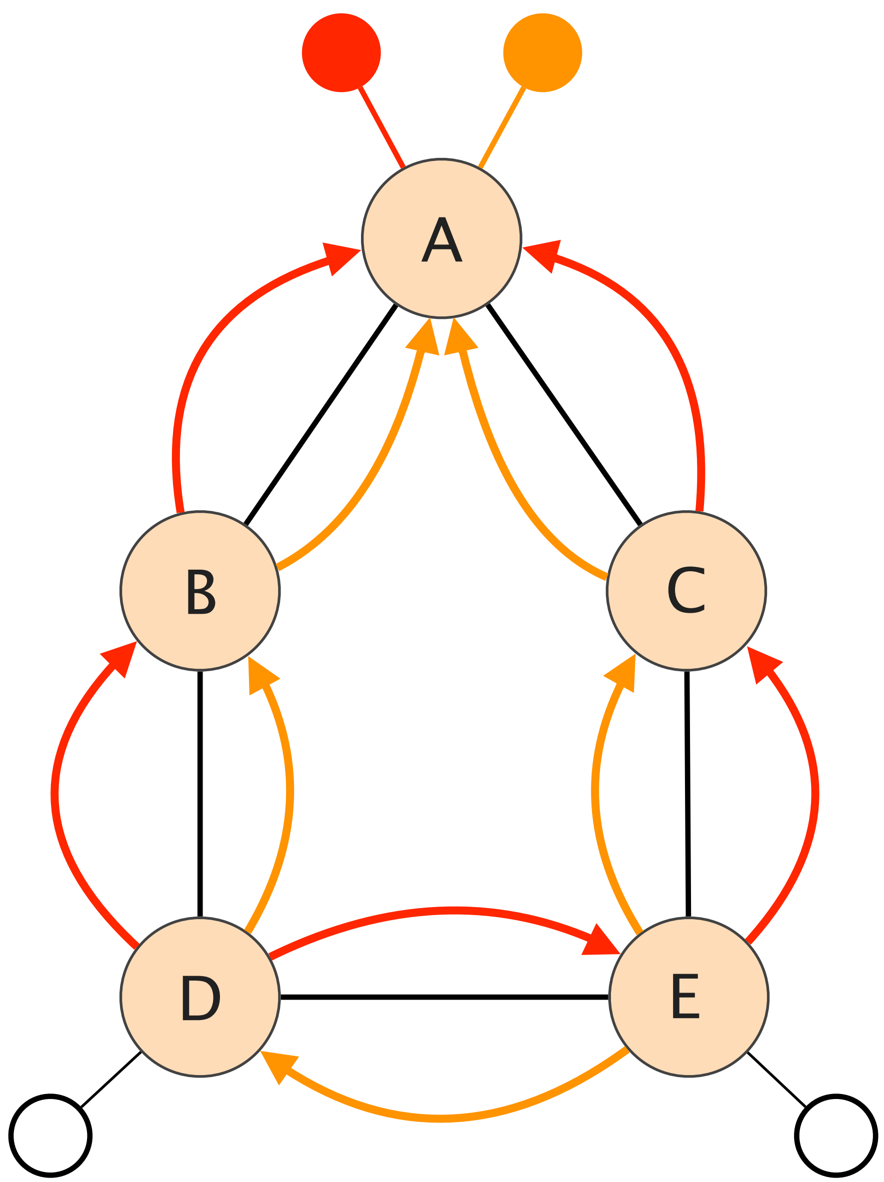


In contrast, SDN-controlled IGP enables
to create ECMP path on a per-destination basis

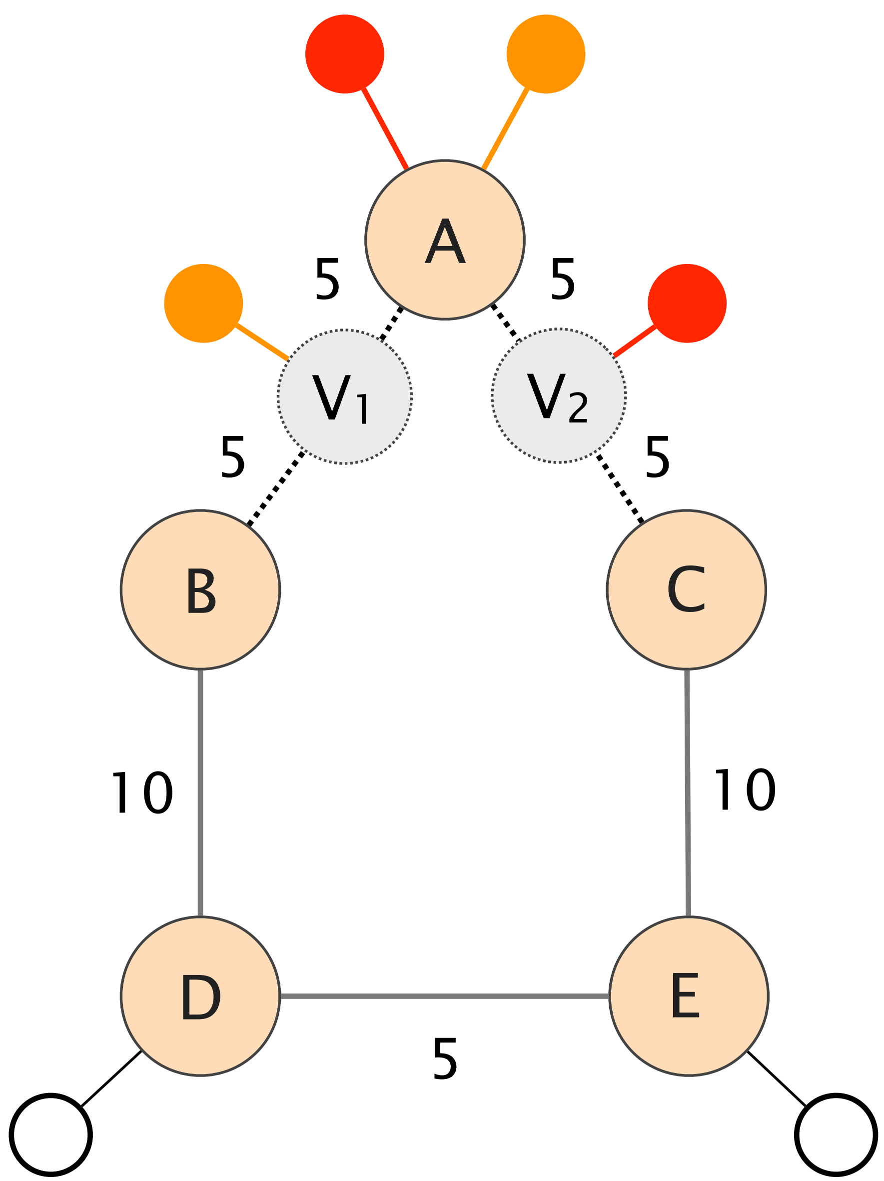


In contrast, SDN-controlled IGP enables to create ECMP path on a per-destination basis

achieved ✔



virtual topology



A SDN-enabled IGP is powerful

Theorem A SDN-enabled IGP can make the routers use any set of non-contradictory paths

A SDN-enabled IGP is powerful

Theorem

A SDN-enabled IGP can make the routers use any set of non-contradictory paths

A SDN-enabled IGP is powerful

Theorem

A SDN-enabled IGP can make the routers use any set of non-contradictory paths

any path is loop-free

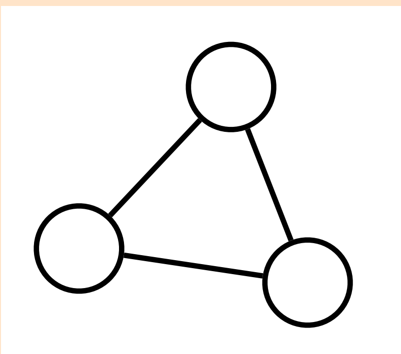
(*e.g.*, [s1, a, b, a, d] is not possible)

paths are consistent

(*e.g.* [s1, a, b, d] and
[s2, b, a, d] are inconsistent)

Given a physical topology and a set of path requirements, a linear program computes a virtual topology

physical topology



+

paths requirements

[RA, RB, RC, RD]

ECMP(
  [RX, RY, RZ],
  [RX, RW, RZ])

[RI, RJ, RK], bckp
[RI, RL, RK]

⇒

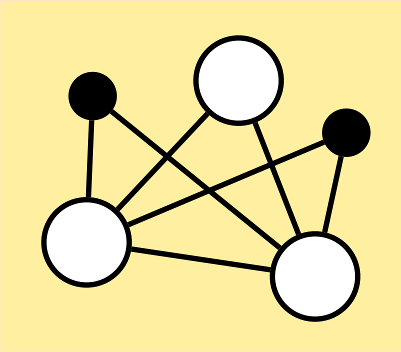
Integer Linear Program

Optimizer

minimize topology size

⇒

virtual topology



SDN-enabled IGP is implementable in practice

SDN-enabled IGP requires to

- listen to the IGP traffic

  simple, just establish an IGP adjacency

- inject fake IGP packets over an adjacency

  effectively, executing a "controlled" IGP attack

- map virtual nodes to physical link

  simple protocol change or use a few SDN-enabled devices

On integrating Software-Defined Networking within existing routing systems

SDN-controlled routers

don’t trash, recycle

SDN-controlled IGP

fine-grained traffic-engineering

3 SDN-controlled BGP

inter domain bonanza

Joint work with

Arpit Gupta, Muhammad Shahbaz, Hyojoon Kim,
Russ Clark, Nick Feamster, Jennifer Rexford and Scott Shenker

BGP can be (and is already) used as a centralized provisioning interface

Three examples of SDN-enabled BGP initiatives

- Route Control Platform [NSDI05]
- BGP Route Injection [LINX69]
- A BGP-Only SDN Controller for Large-Scale Data Centers [NANOG58]

So far, existing initiatives have focused on iBGP

iBGP	Route Control Platform
iBGP	BGP Route Injection
iBGP	A BGP-Only SDN Controller for Large-Scale Data Centers

Managing eBGP is also painful and would also benefit from SDN-like mechanisms

Inflexible (control-plane and data-plane)

BGP decision process and destination-based fwd

Non-deterministic

one can only “influence” remote decisions

Geographically-limited

one can only “do” something *where* it has an eBGP session

We combine BGP with SDN-enabled devices at Internet eXchange Points (IXP)

Augment the IXP data-plane with SDN capabilities

keeping default forwarding and routing behavior

Enable fine-grained inter domain policies

bringing new features while simplifying operations

We combine BGP with SDN-enabled devices at Internet eXchange Points (IXP)

Augment the IXP data-plane with SDN capabilities

keeping default forwarding and routing behavior

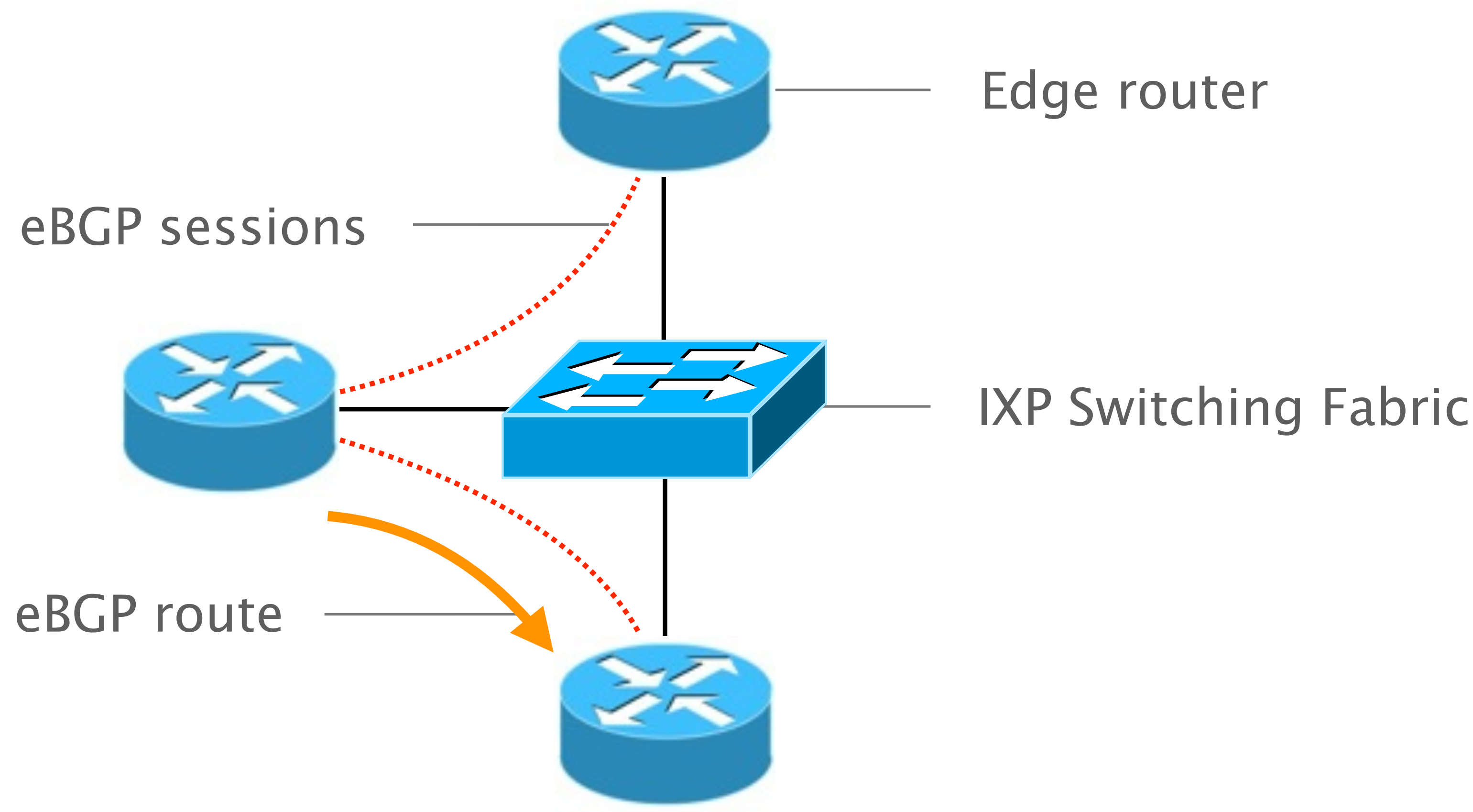
Enable fine-grained inter domain policies

bringing new features while simplifying operations

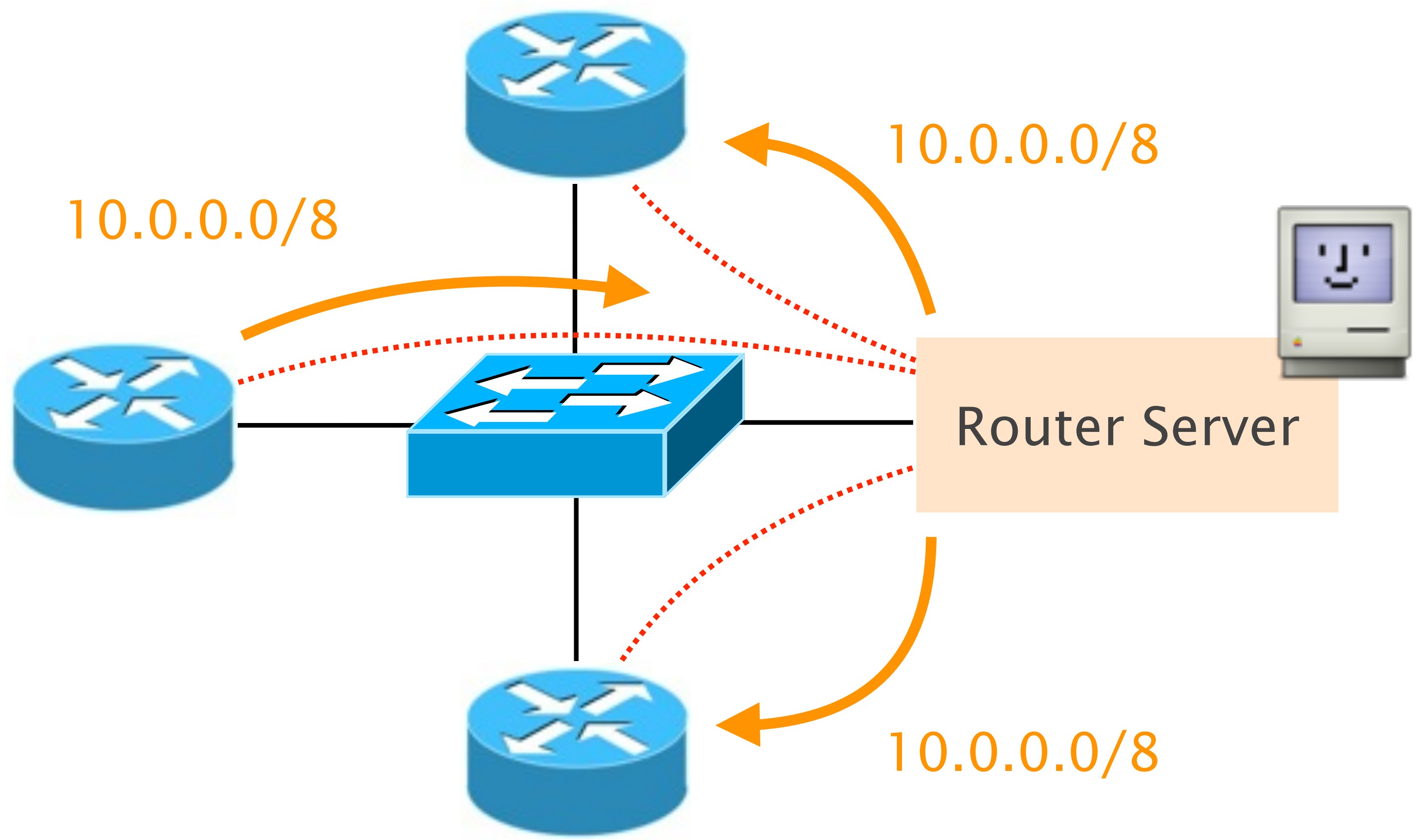
… with scalability in mind

supporting the load of a large IXP

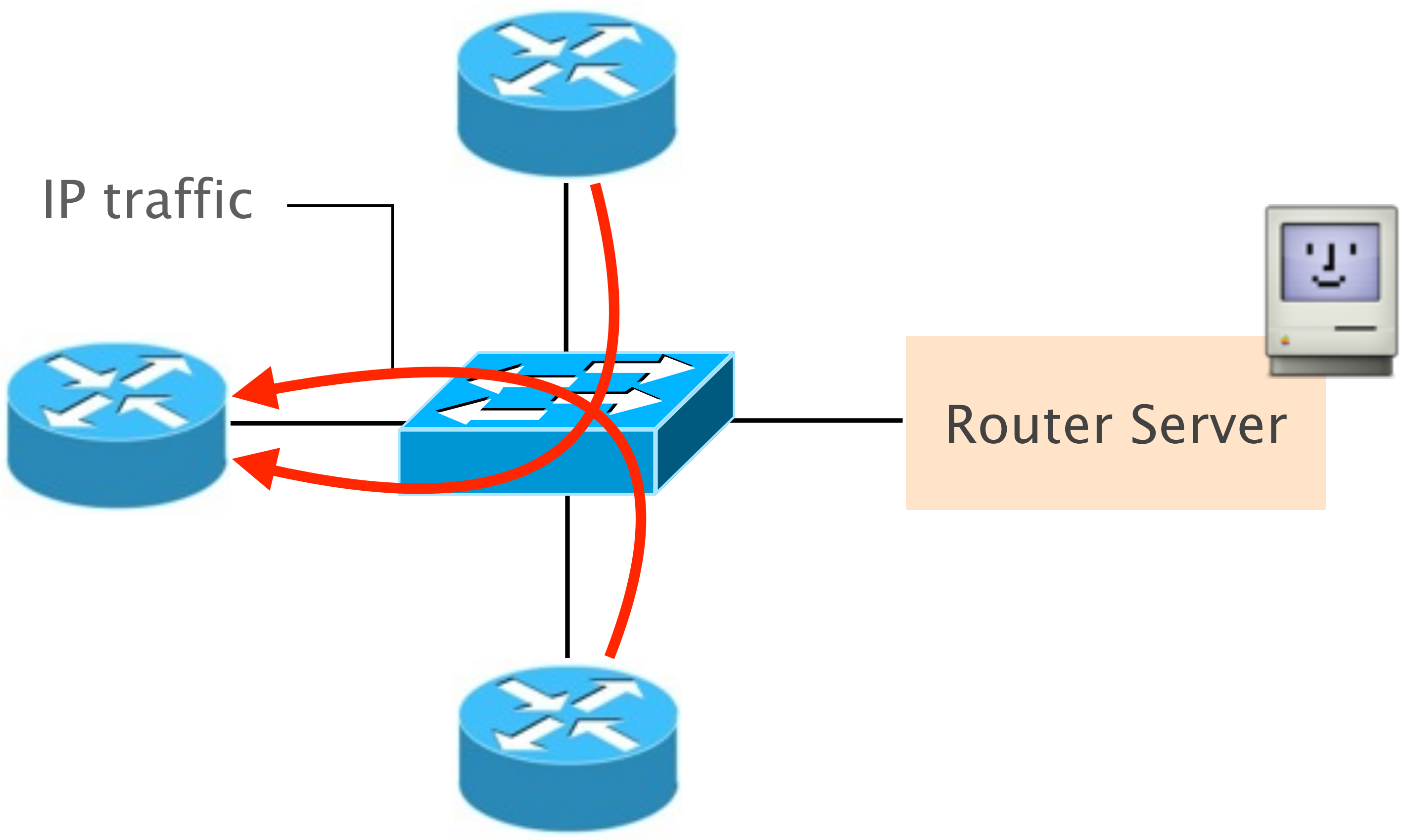
An IXP is a large L2 domain where
participant routers exchange routes using BGP



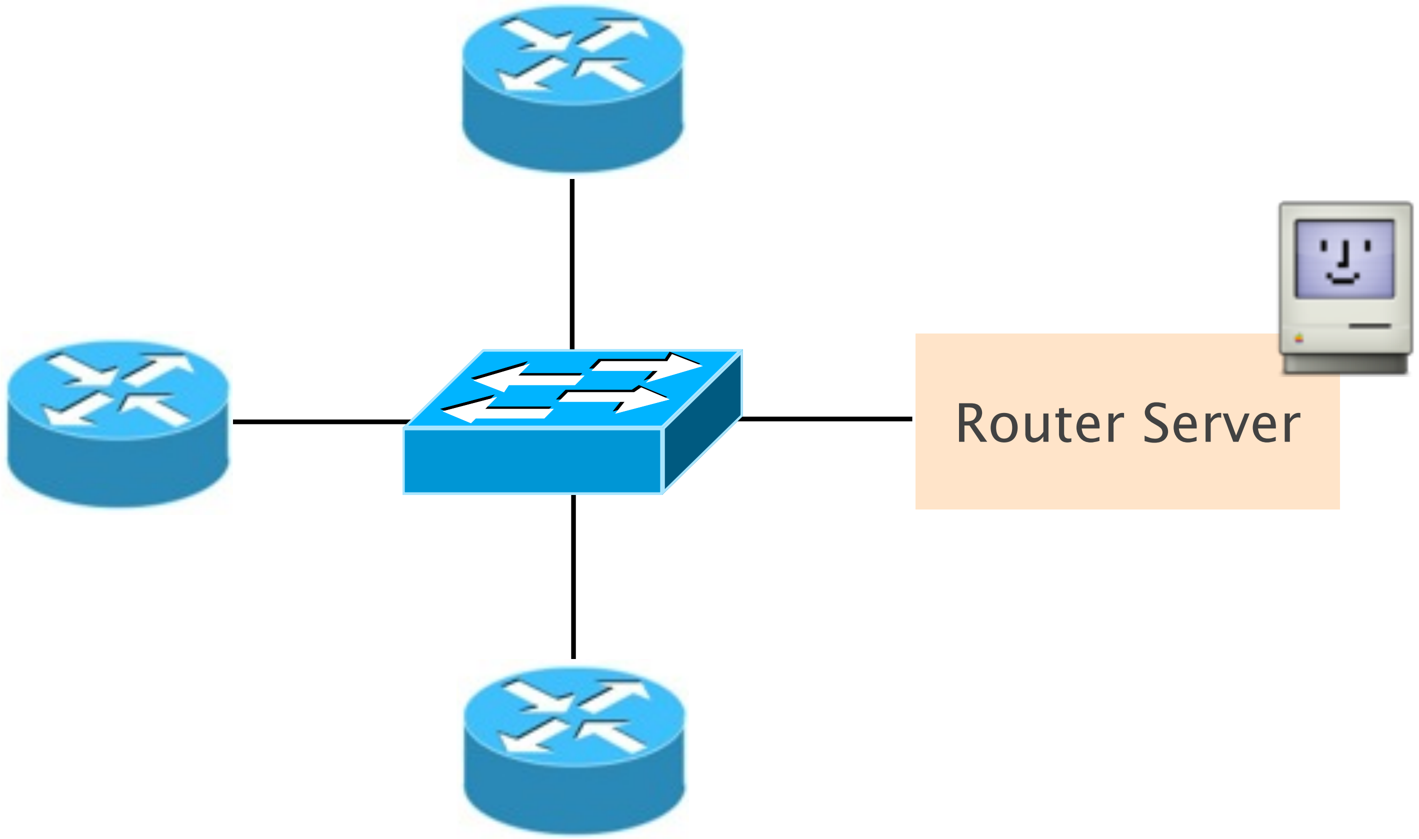
To alleviate the need of establishing eBGP sessions, IXP often provides a Route Server (route multiplexer)



IP traffic is exchanged directly between participants, *i.e.* the IXP is forwarding transparent

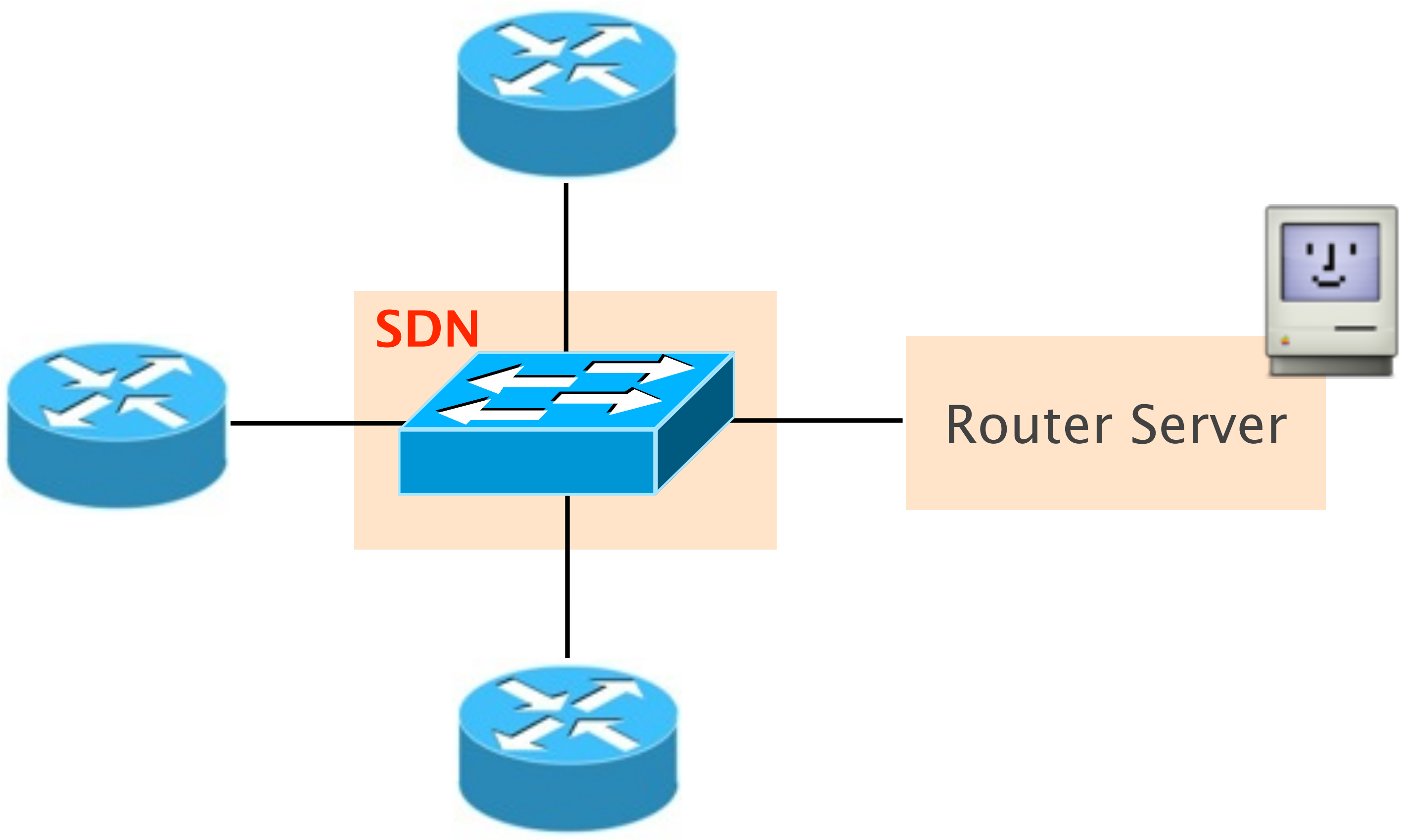


With respect to a traditional IXP, SDN-enabled IXP (SDX)



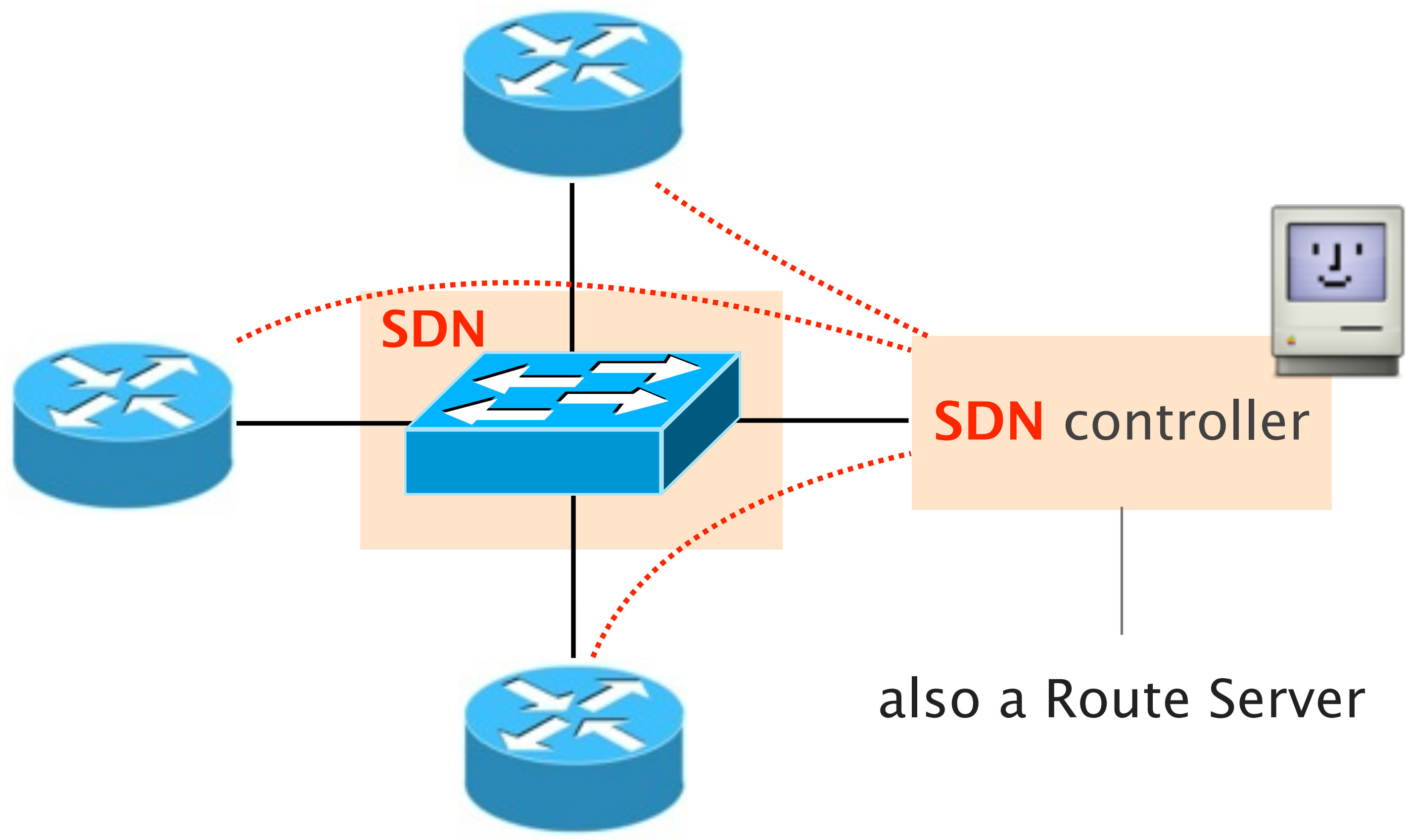
With respect to a traditional IXP, SDN-enabled IXP (SDX)

data-plane relies on SDN-capable devices



With respect to a traditional IXP, SDN-enabled IXP (SDX)

control-plane relies on a SDN controller



SDX participants express their policies
in a high-level language built on top of Pyretic (*)

(*) http://frenetic-lang.org/pyretic/

SDX policies are composed of a *pattern* and some *actions*

```
match (      Pattern       ), then (  Actions  )
```

Pattern selects packets based on any header fields,

Pattern

```
          eth_type
          vlan_id
          srcmac
match (   dstmac    , &&, ||  ), then (  Actions  )
          protocol
          dstip
          tos
          srcip
          srcport
          dstport
```

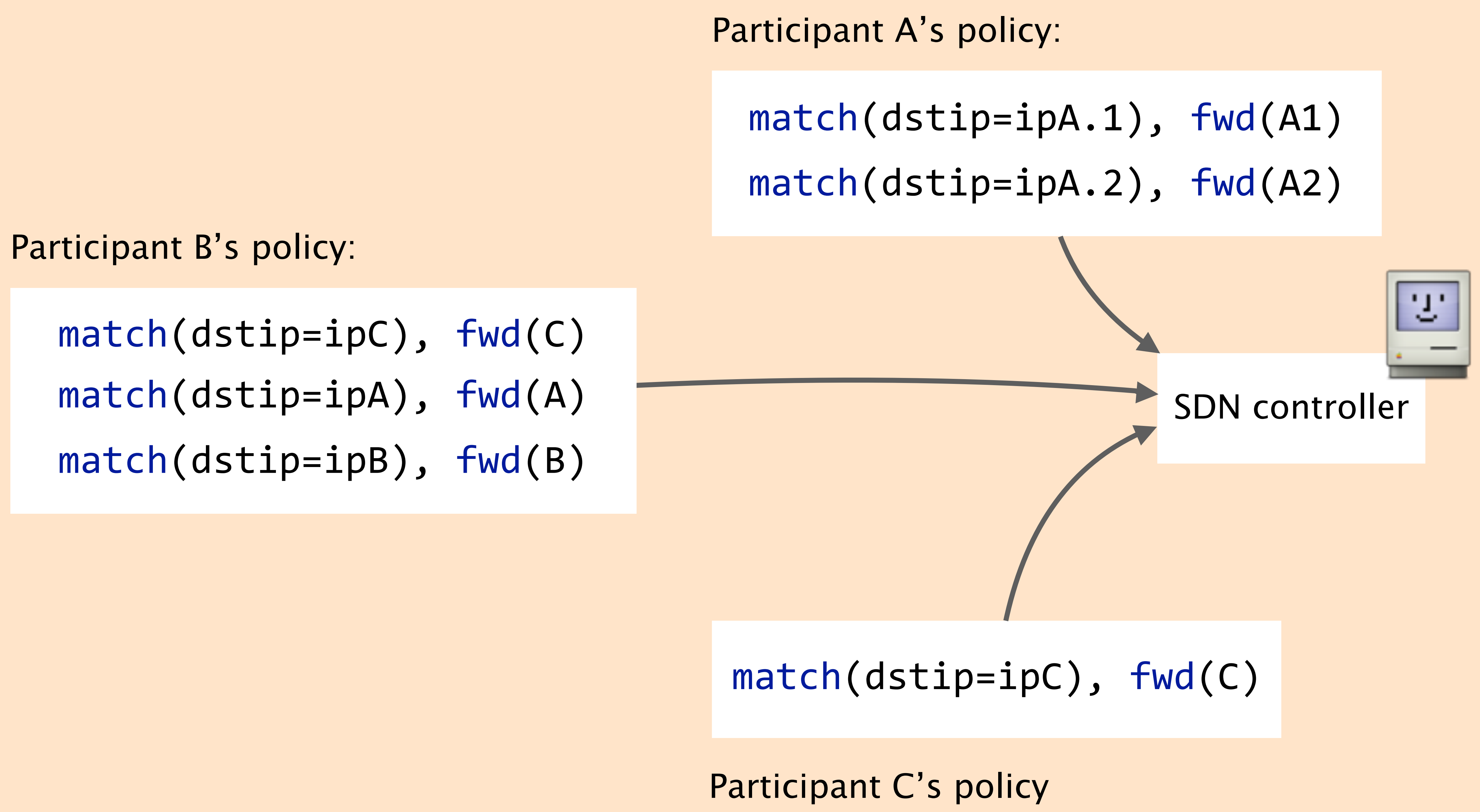
Pattern selects packets based on any header fields,

while actions forward or modify the selected packets

Actions

```
match (      Pattern      ), then ( drop    )
                                    forward
                                    rewrite
```

Each participant writes her policies *independently* and transmits them to the controller



Given the participant policies, the controller compiles them to SDN forwarding entries

Ensuring isolation

Resolving policies conflict

Ensuring scalability

Given the participant policies, the controller compiles them to SDN forwarding entries

Ensuring isolation

Each participant controls one virtual switch

connected to participants it can communicate with

Resolving policies conflict

Ensuring scalability

Given the participant policies, the controller compiles them to SDN forwarding entries

Ensuring isolation

Resolving policies conflict

Participant policies are sequentially composed

in an order that respects business relationships

Ensuring scalability

Given the participant policies, the controller compiles them to SDN forwarding entries

Ensuring isolation

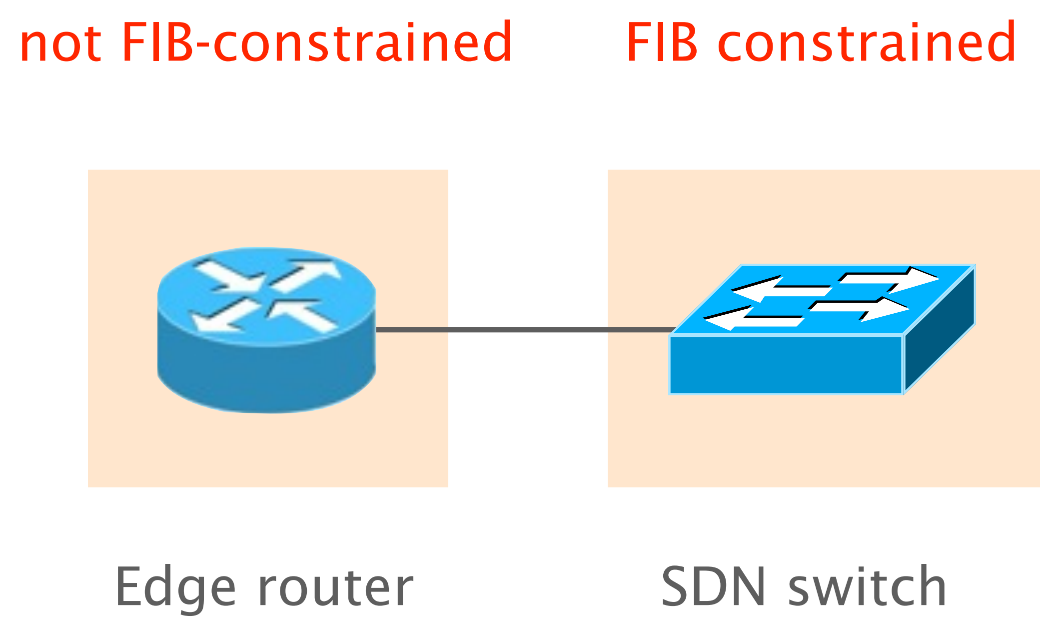
Resolving policies conflict

Ensuring scalability

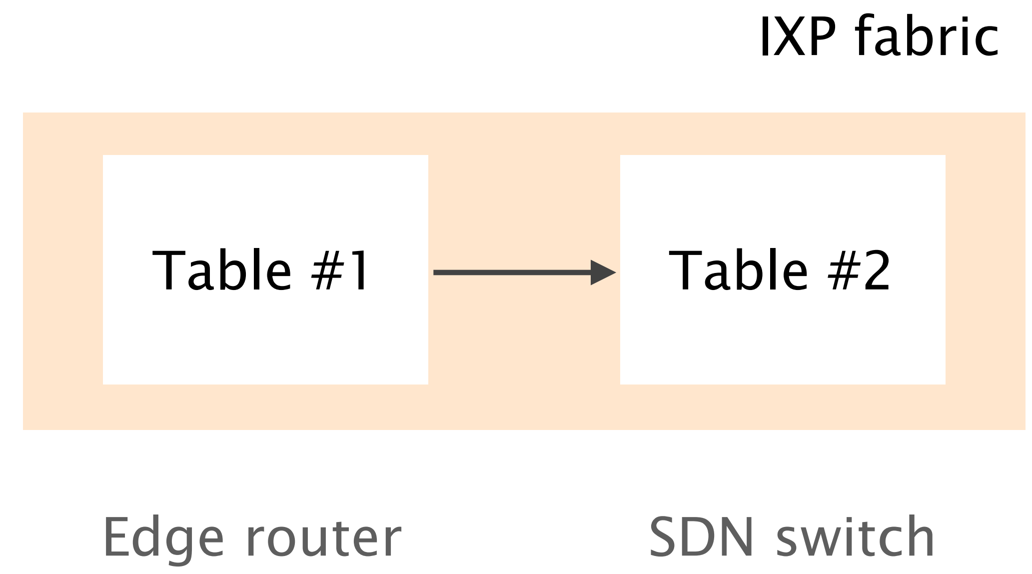
only install the minimum required in the data plane

leverage the existing BGP control plane for the rest

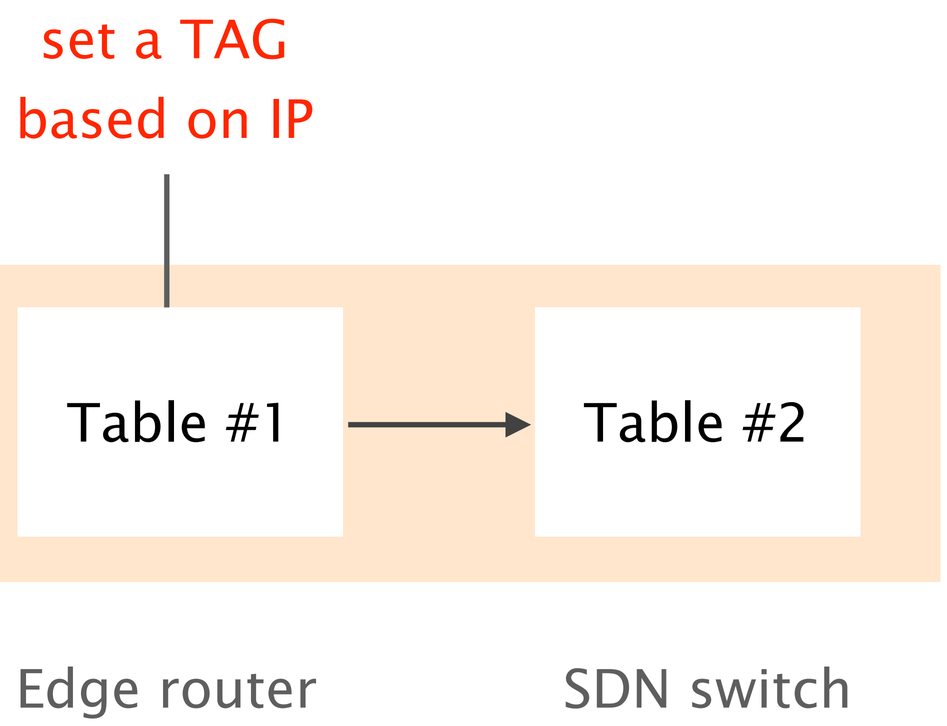
The edge routers, sitting next to the fabric, are tailored to match on numerous IP prefixes



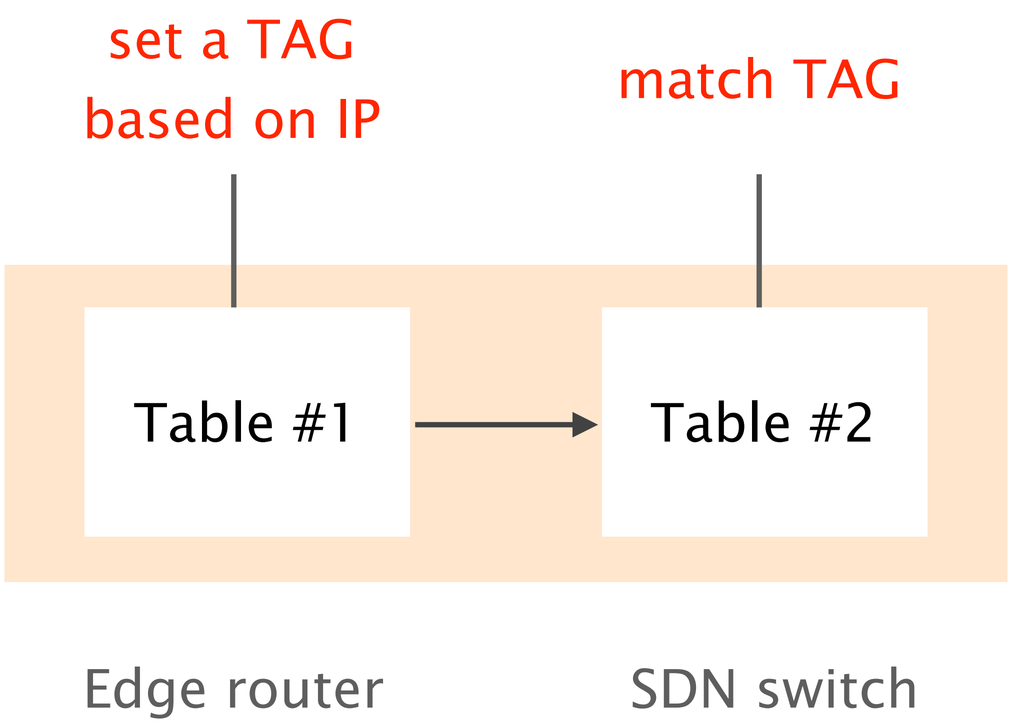
We consider routers FIB as the first stage of a multi-stage FIB



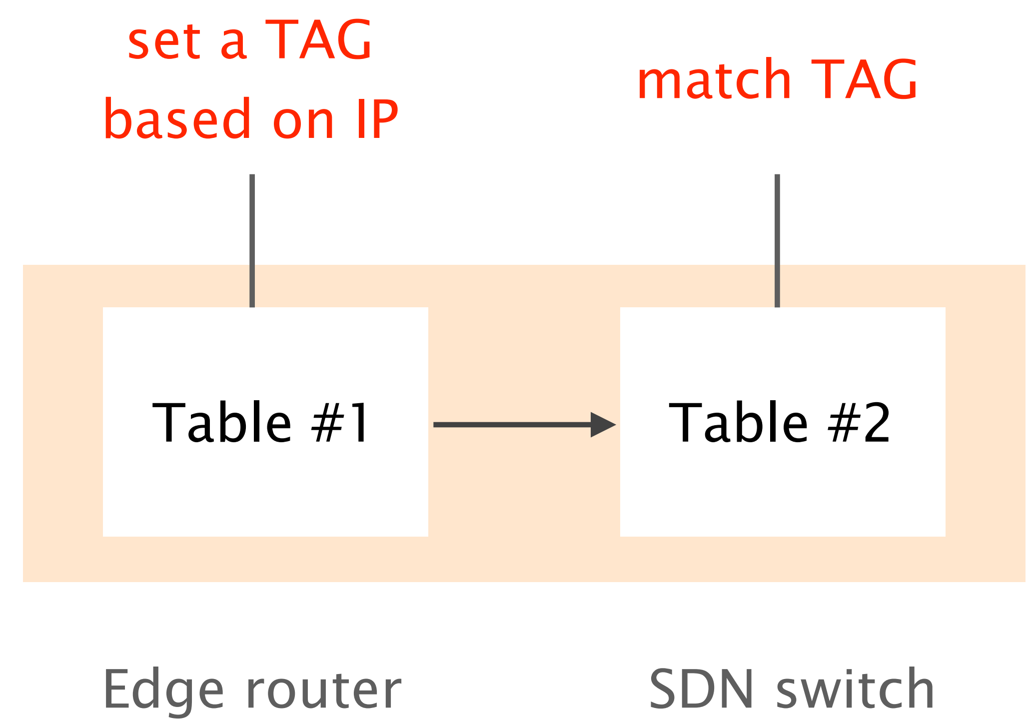
Routers FIB match on the destination prefix and set a tag accordingly



The SDN FIB matches on the tag, not on the IP prefixes



How do we provision tag entries in a router, and what are these tags?



We use BGP as a provisioning interface and the L2 address of the BGP NH as label

When a BGP router receives a route, it

- runs the decision process
- resolves the BGP NH to a L2 NH (if it is a best route)
- installs a FIB entry directing the traffic to the L2 NH

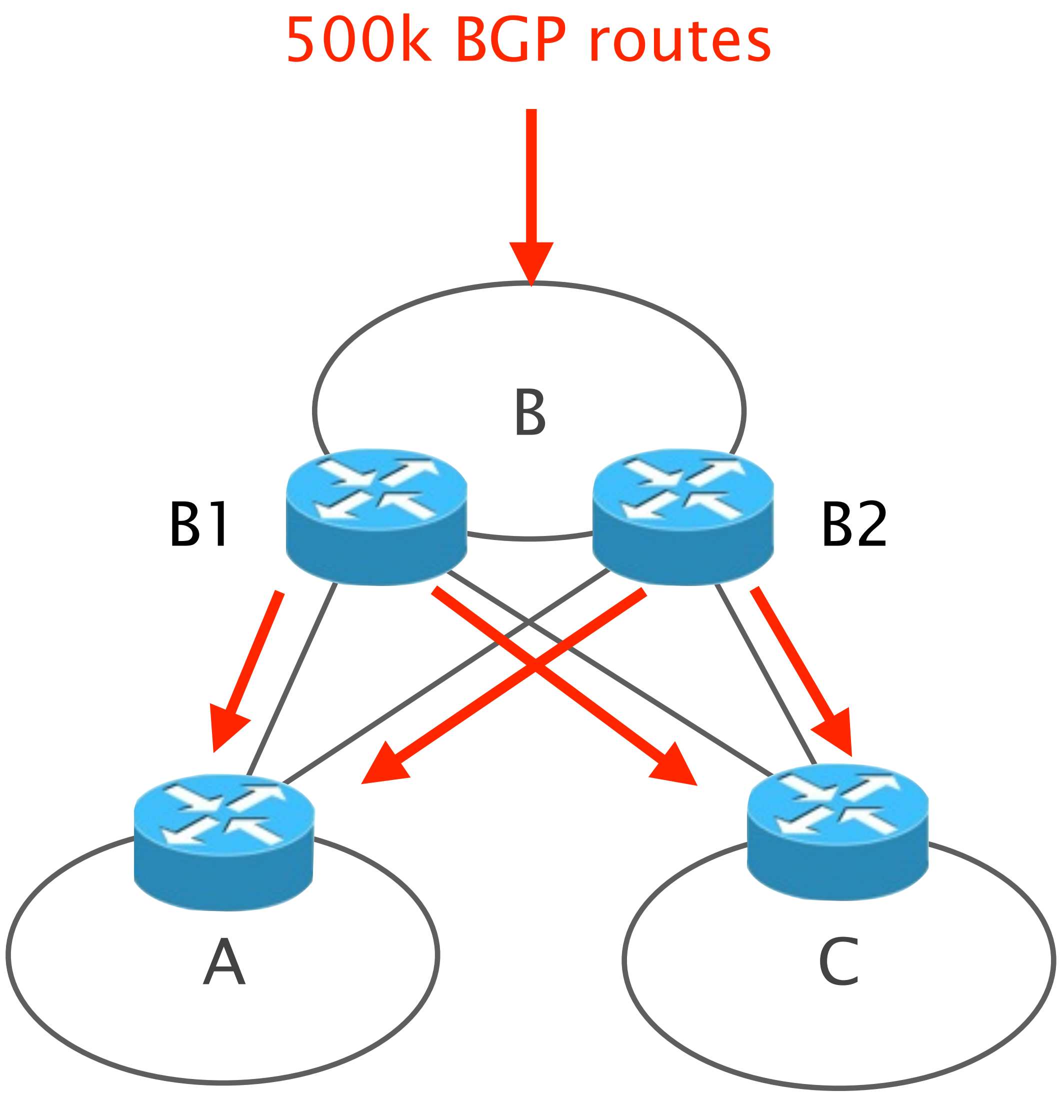
We use BGP as a provisioning interface and the L2 address of the BGP NH as label

When a BGP router receives a route, it

- runs the decision process,
- resolves the BGP NH to a L2 NH (if it is a best route),
- installs a FIB entry directing the traffic to the L2 NH

We can tweak the BGP/L2 NH and use it as a tag

Let's walk through the compilation of a simple inbound TE policy



A, B and C are all connected to the SDX

AS B's SDX policy

match(dstip=0*)	fwd(B1)
match(dstip=1*)	fwd(B2)

The policy is first divided in match and forward actions

match(dstip=0*)	fwd(B1)
match(dstip=1*)	fwd(B2)

The policy is first divided in match and forward actions

match(dstip=0*)	fwd(B1)
match(dstip=1*)	fwd(B2)

A virtual IP/MAC next-hop is associated to each distinct forwarding actions

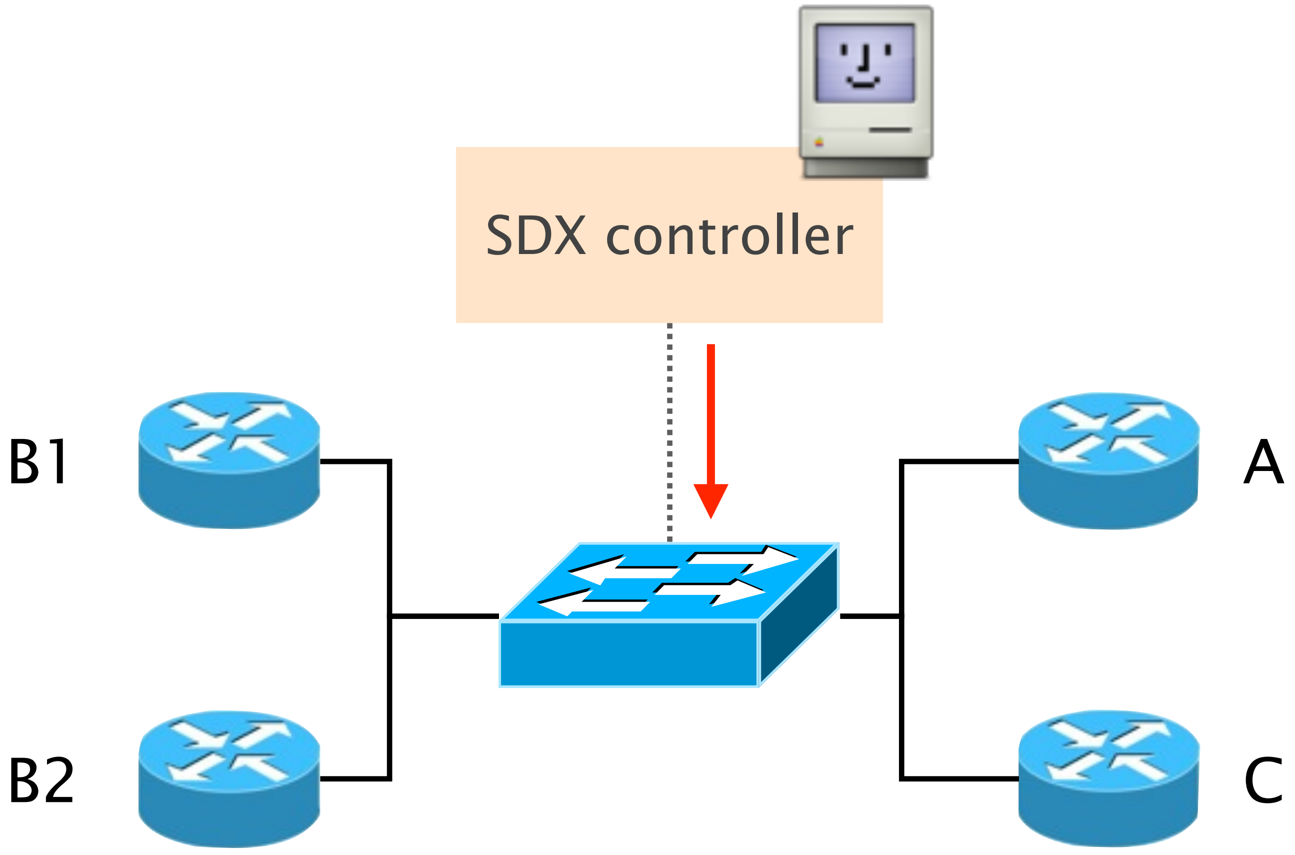
fwd(B1) —— (NH1, MAC1)

fwd(B2) —— (NH2, MAC2)

The SDX controller provisions two data plane rules matching the destination MAC

fwd(B1) —— (NH1, MAC1)

fwd(B2) —— (NH2, MAC2)



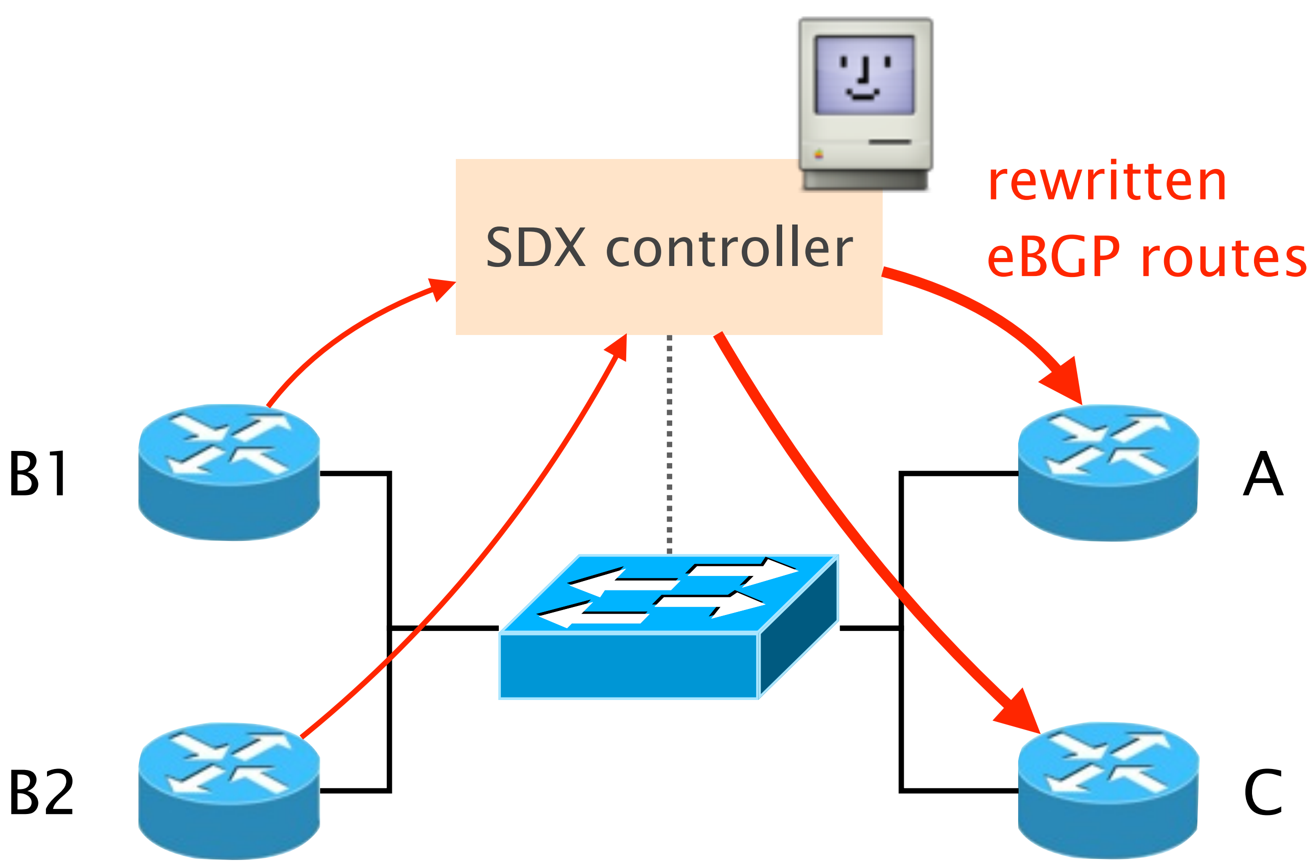
Forwarding rules

match(dst:MAC1), fwd(B1)
match(dst:MAC2), fwd(B2)

The SDX controller rewrite the BGP NH of B’s routes according to the match part of the policy

match(dstip=0*) — (NH1, MAC1)

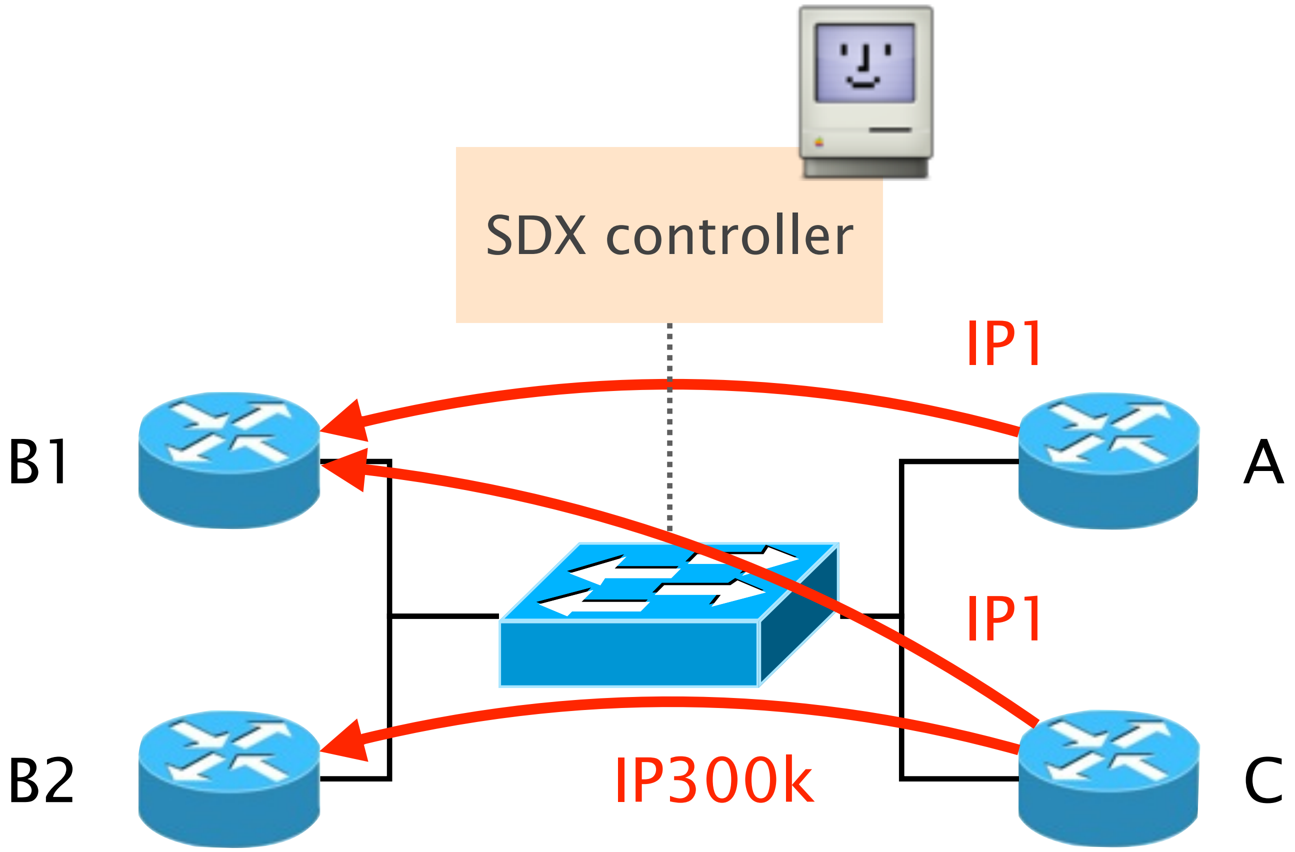
match(dstip=1*) — (NH2, MAC2)



BGP routes sent to A & C

prefix	*NH*
p1	NH1
…	…
p250k	NH1
p250k+1	NH2
…	…
p250k	NH2

Traffic from A and C is splitted on B1 and B2 according to B's policy, with only 2 data-plane rules



What else does SDX enable that was hard or impossible to do before?

SDX enables a wide range of novel applications

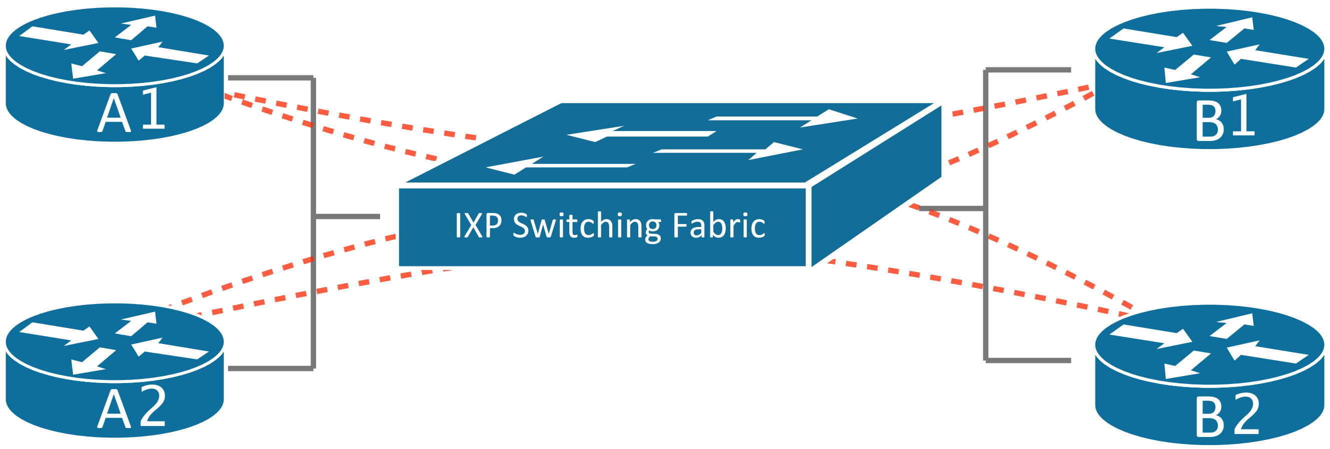
security	Prevent/block policy violation
	Prevent participants communication
forwarding optimization	Middlebox traffic steering
	Traffic offloading
	Inbound Traffic Engineering
	Fast convergence
peering	Application-specific peering
remote-control	Upstream blocking of DoS attacks
	Influence BGP path selection
	Wide-area load balancing

SDX enables a wide range of novel applications

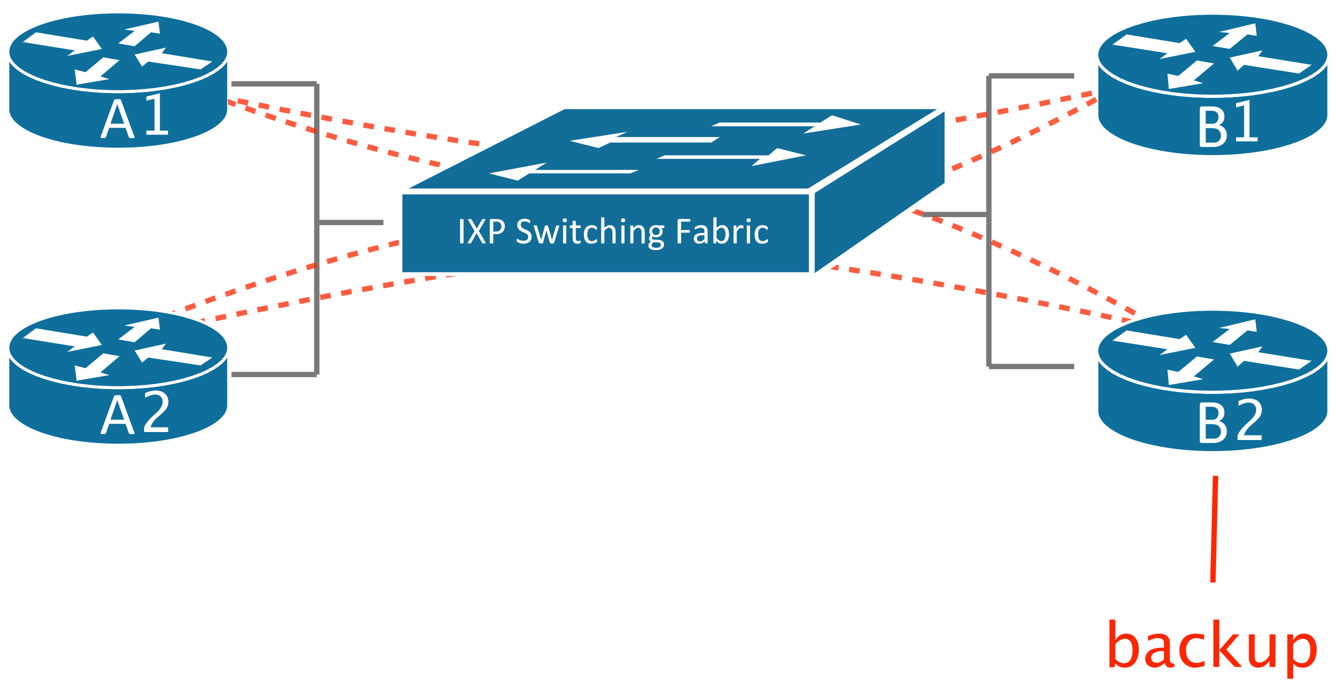
security	Prevent/block policy violation
	Prevent participants communication
forwarding optimization	Middlebox traffic steering
	Traffic offloading
	Inbound Traffic Engineering
	Fast convergence
peering	Application-specific peering
remote-control	Upstream blocking of DoS attacks
	Influence BGP path selection
	Wide-area load balancing

BGP is pretty slow to converge upon peering failure

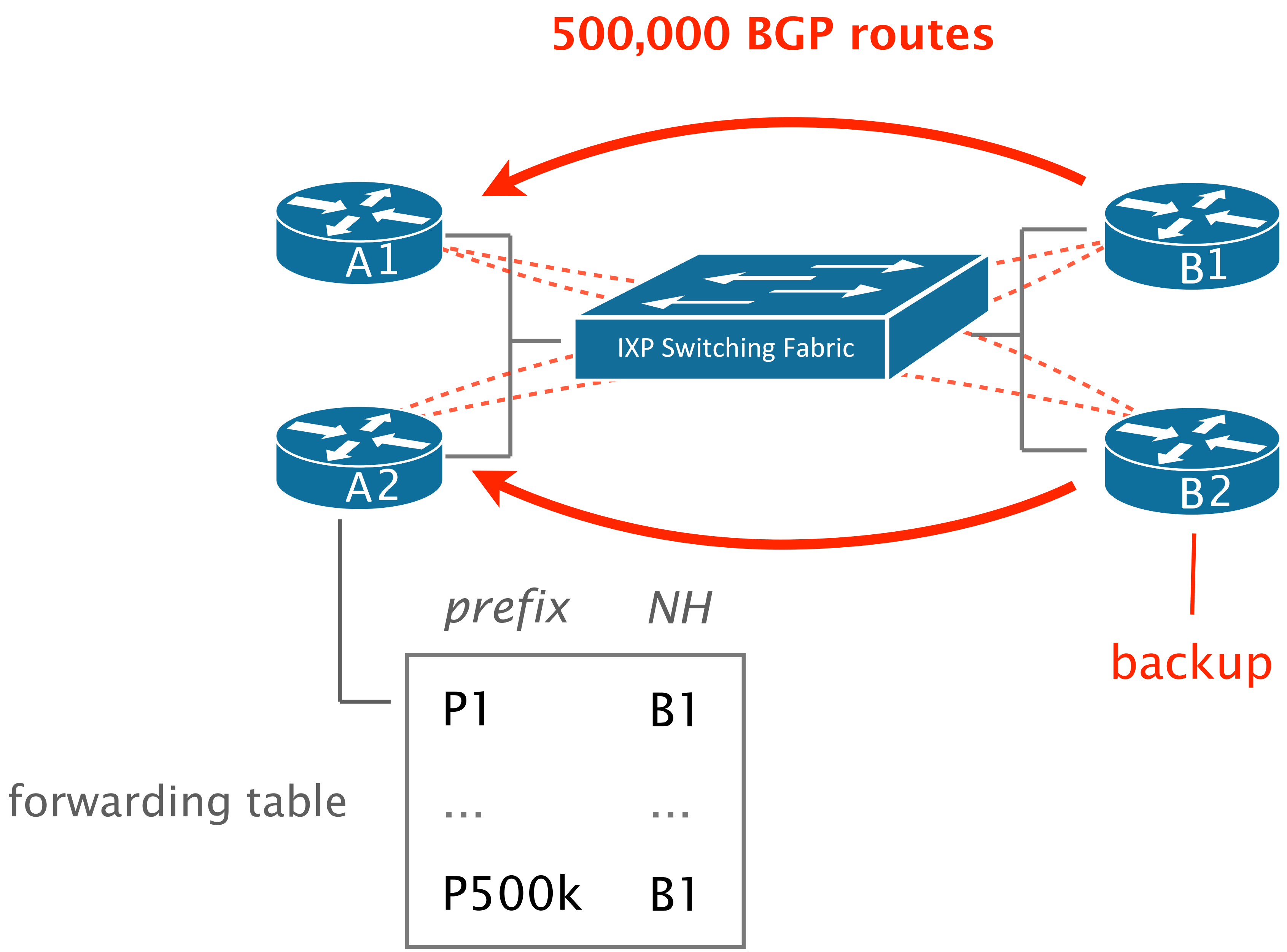
Let's consider a simple example with 2 networks, A and B, with B being the provider of A



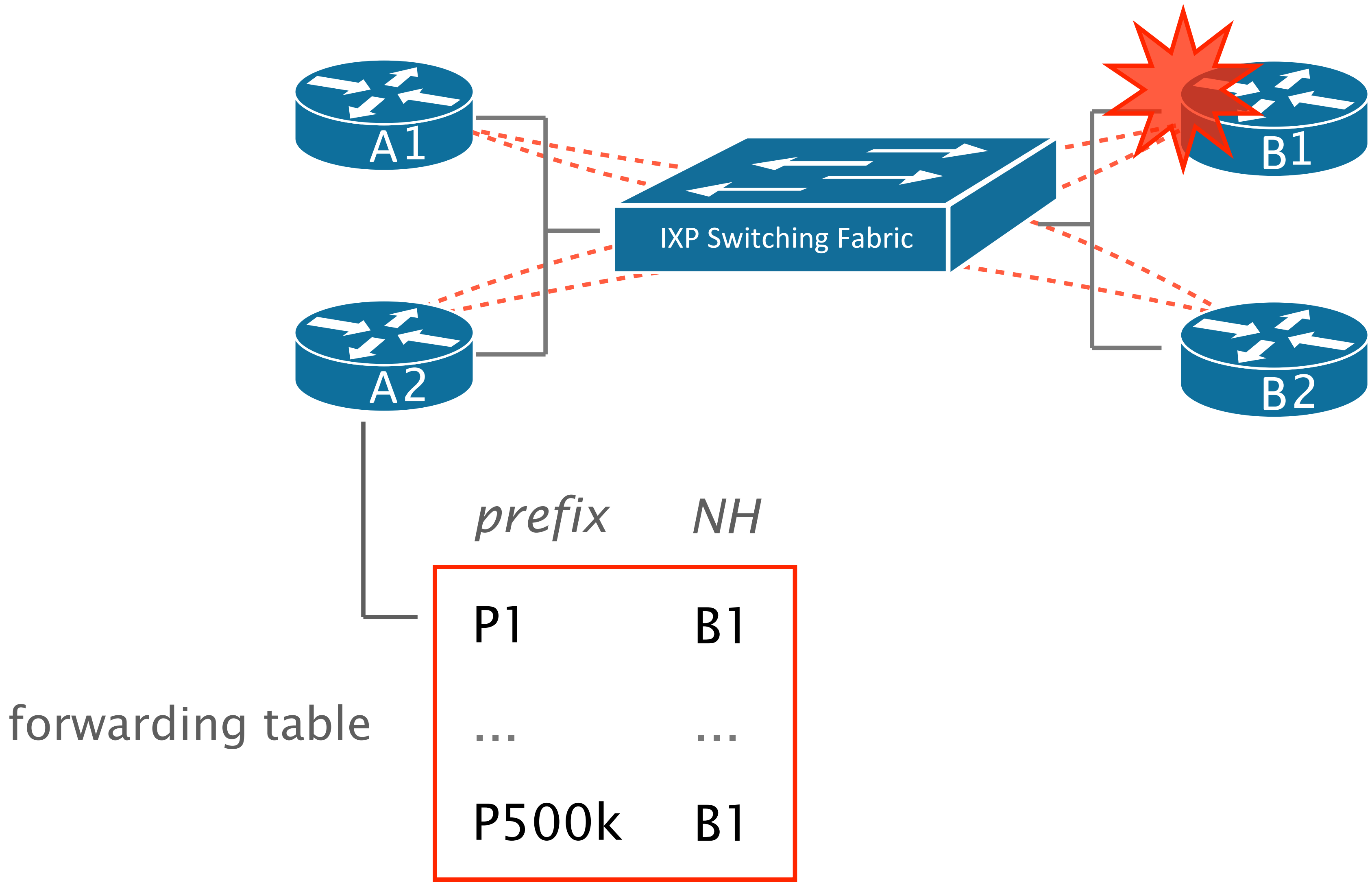
Router B2 is a backup router,
it may be used only upon B1's failure



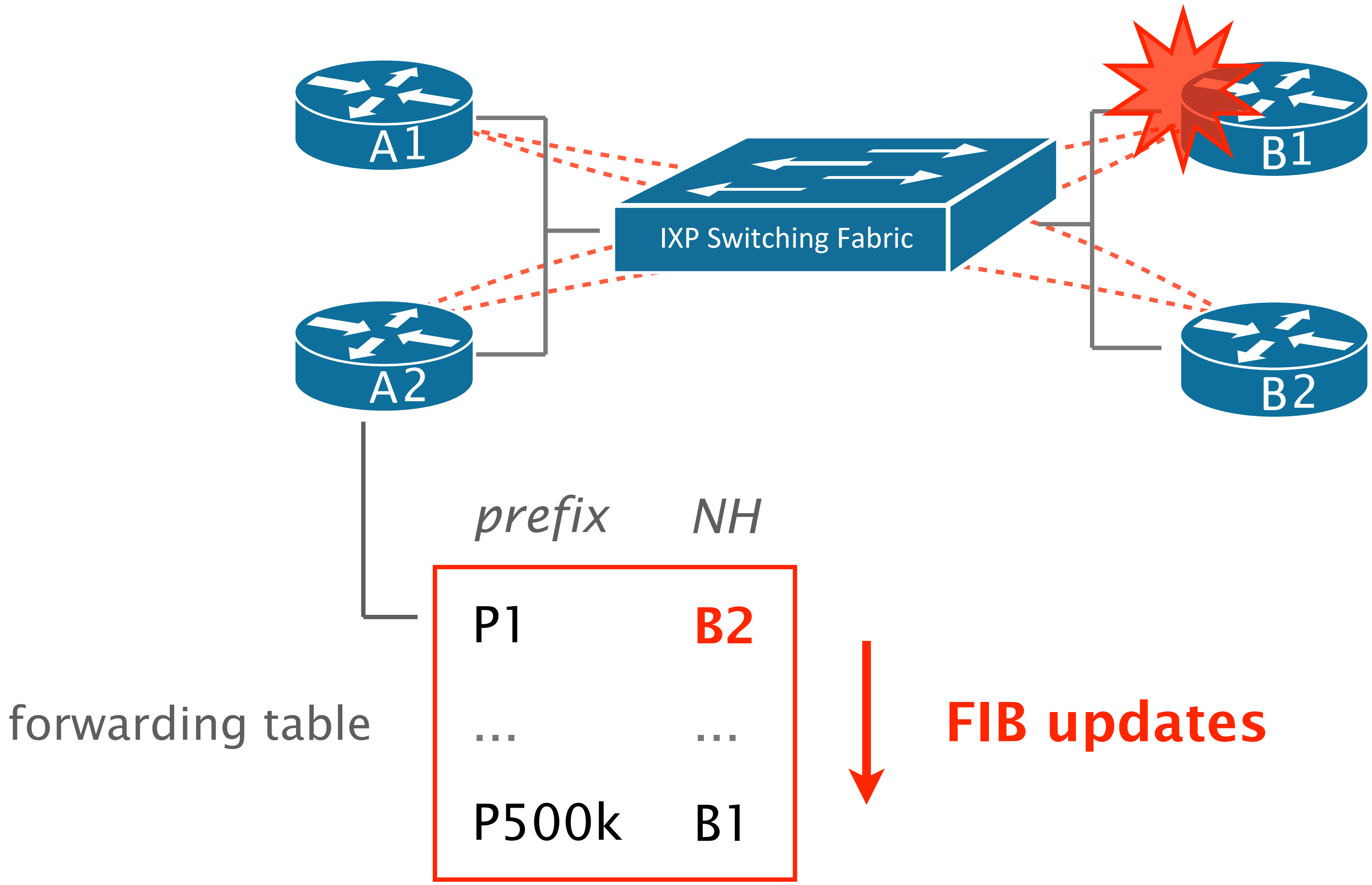
Both A1 and A2 prefer the routes received from B1 and install them in their FIB



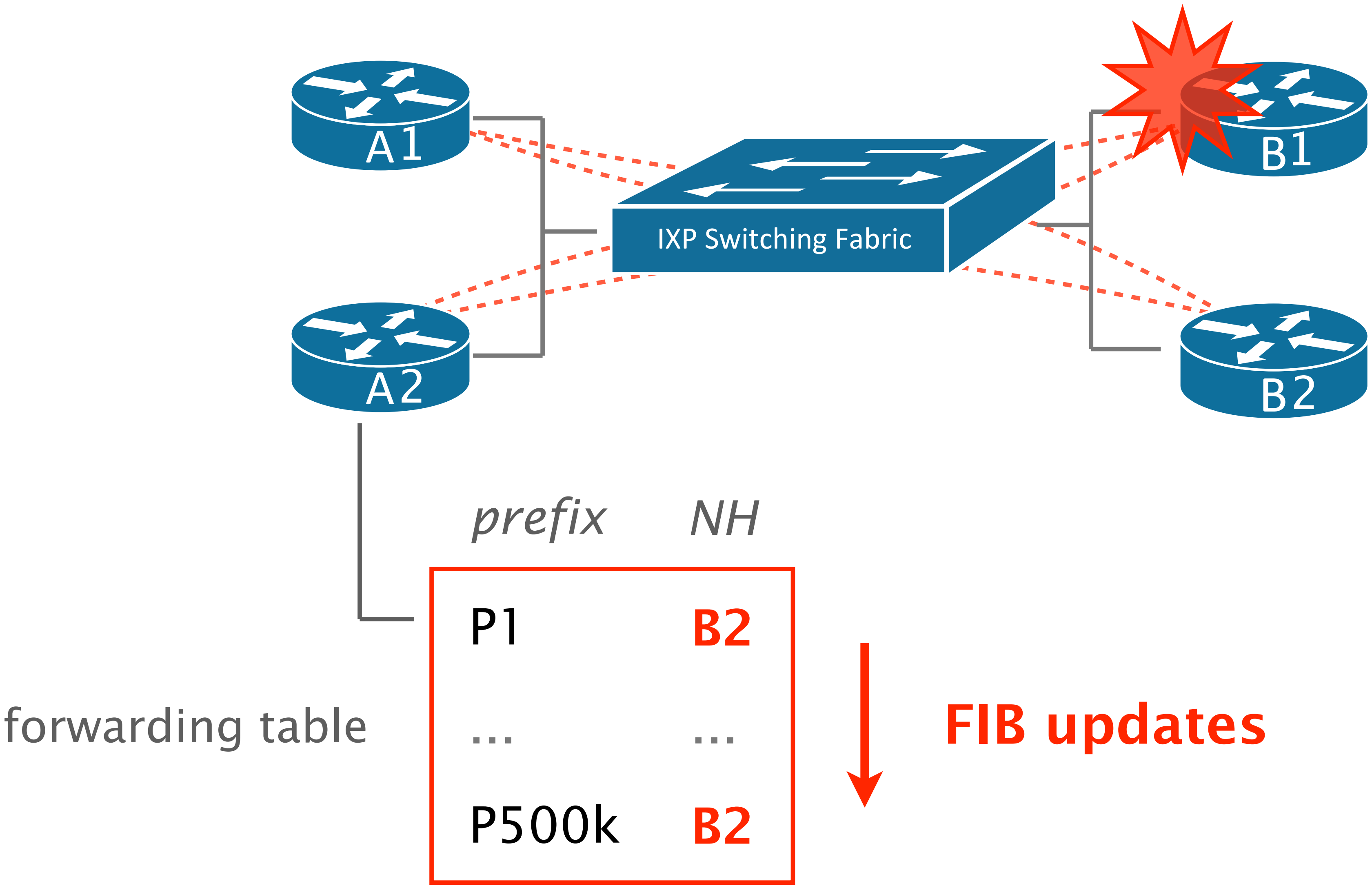
Upon B1's failure, A1 and A2 must update every single entry in their FIB (~500k entries)



Upon B1's failure, A1 and A2 must update every single entry in their FIB (~500k entries)



Upon B1's failure, A1 and A2 must update every single entry in their FIB (~500k entries)



On most routers, FIB updates are performed linearly, entry-by-entry, leading to *slow* BGP convergence

convergence time 500k entries * 150 $\frac{\mu\text{secs}}{\text{entry}}$

average time
to update one entry

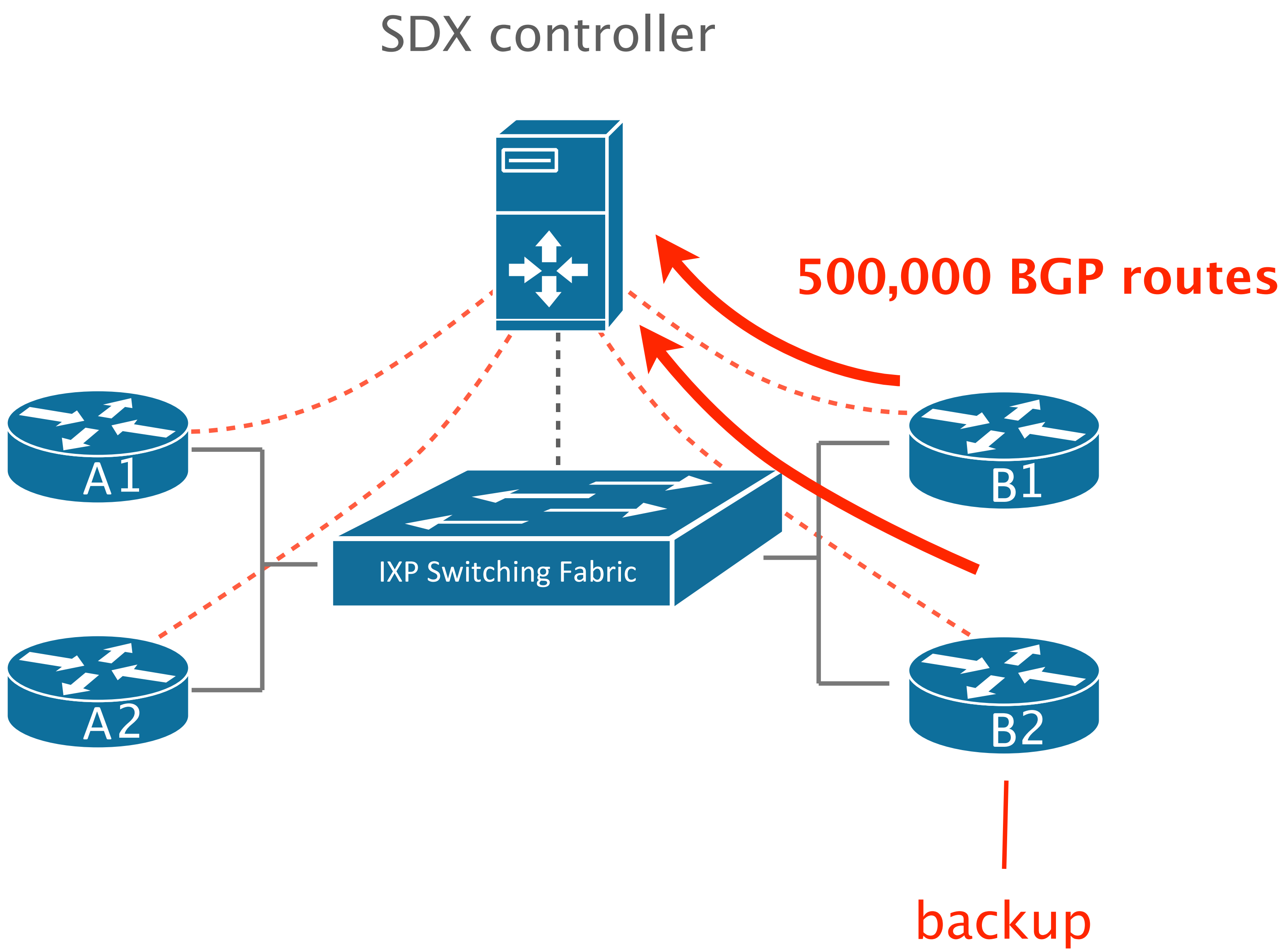
On most routers, FIB updates are performed linearly, entry-by-entry, leading to *slow* BGP convergence

convergence time $\quad$ 500k entries * 150 $\frac{\mu\text{secs}}{\text{entry}}$ = **O(75) seconds**

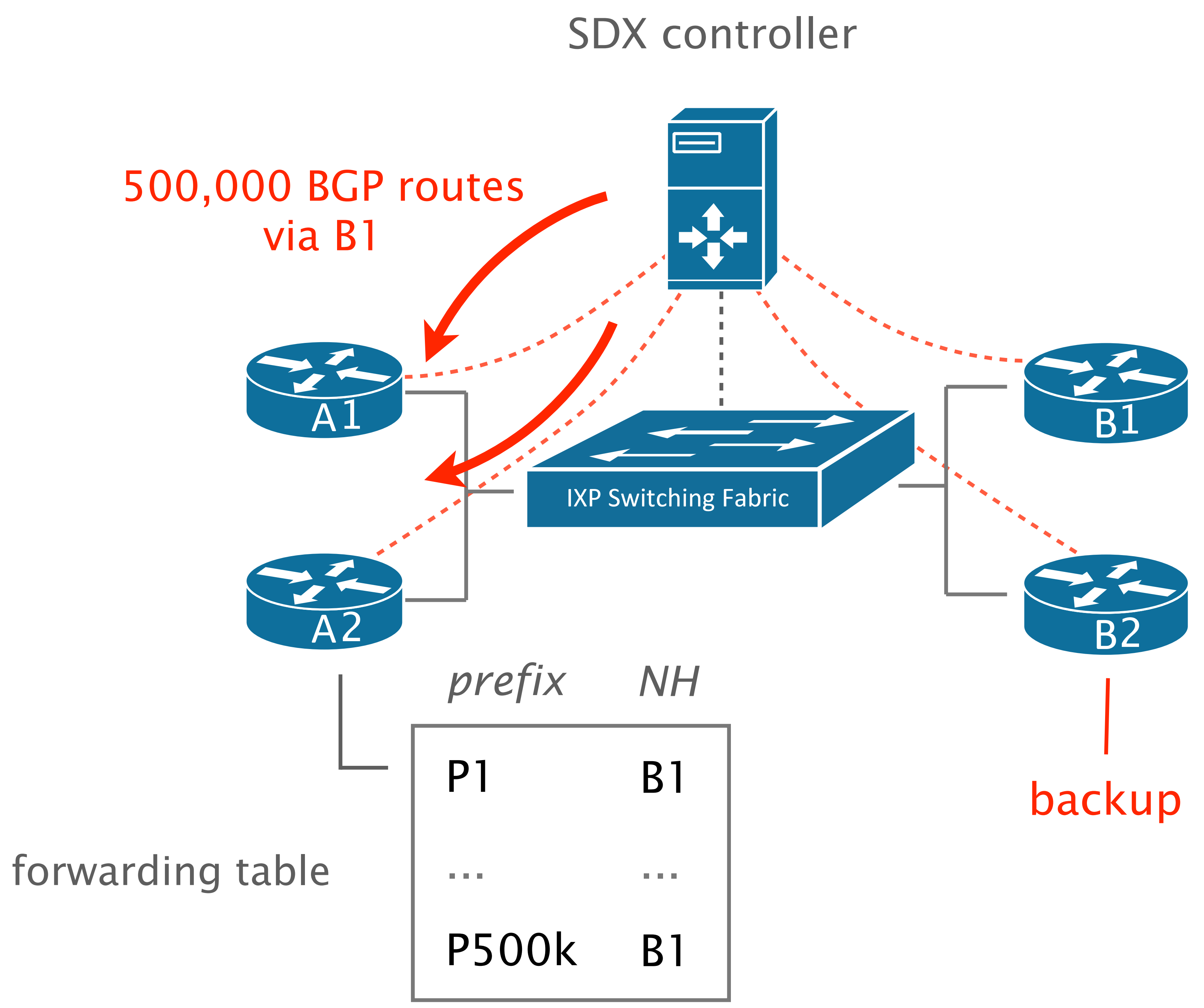
average time
to update one entry

With SDX, sub-second peering convergence can be achieved with any router

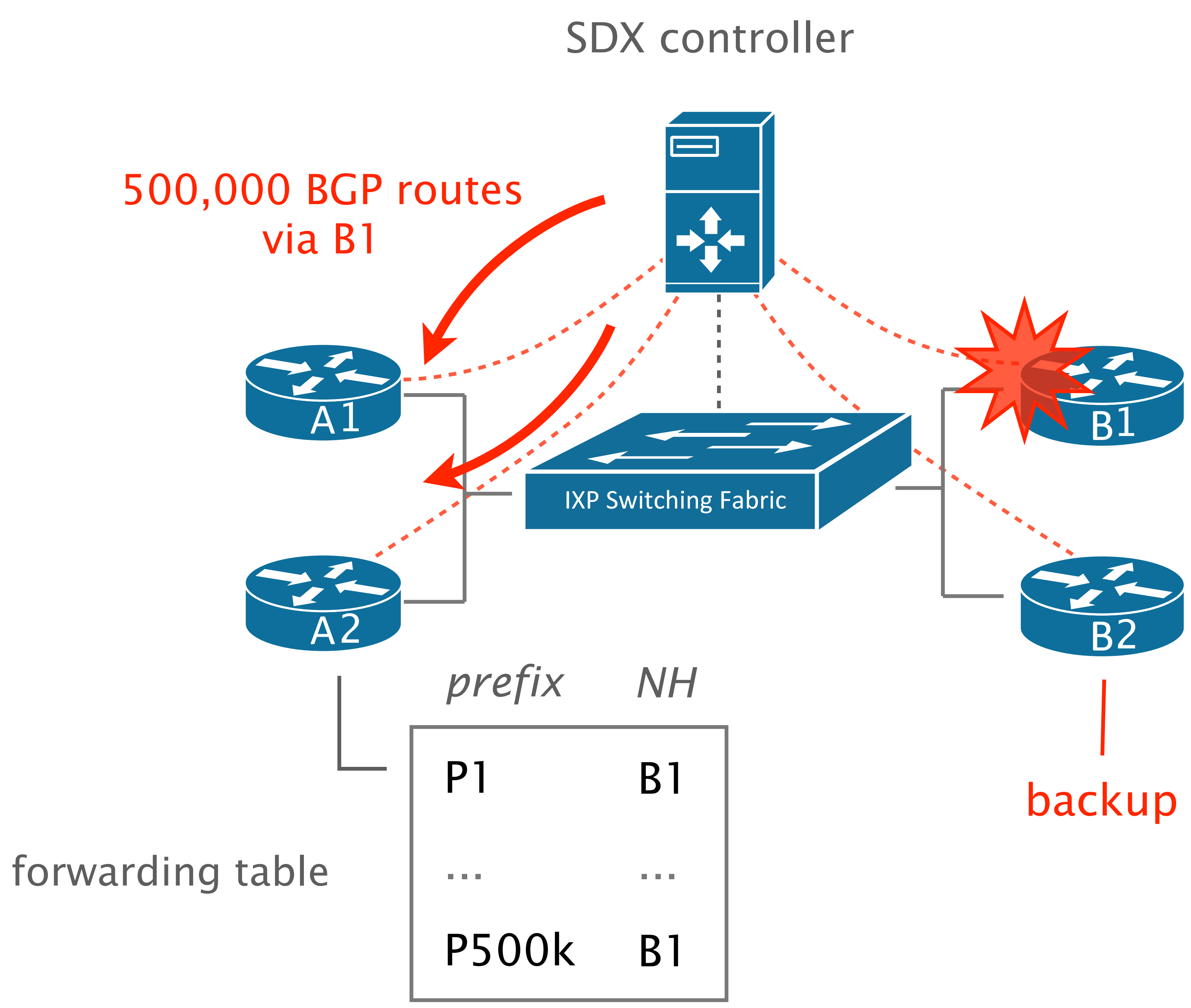
When receiving multiple routes, the SDX controller pre-computes a backup NH for each prefix



When receiving multiple routes, the SDX controller pre-computes a backup NH for each prefix

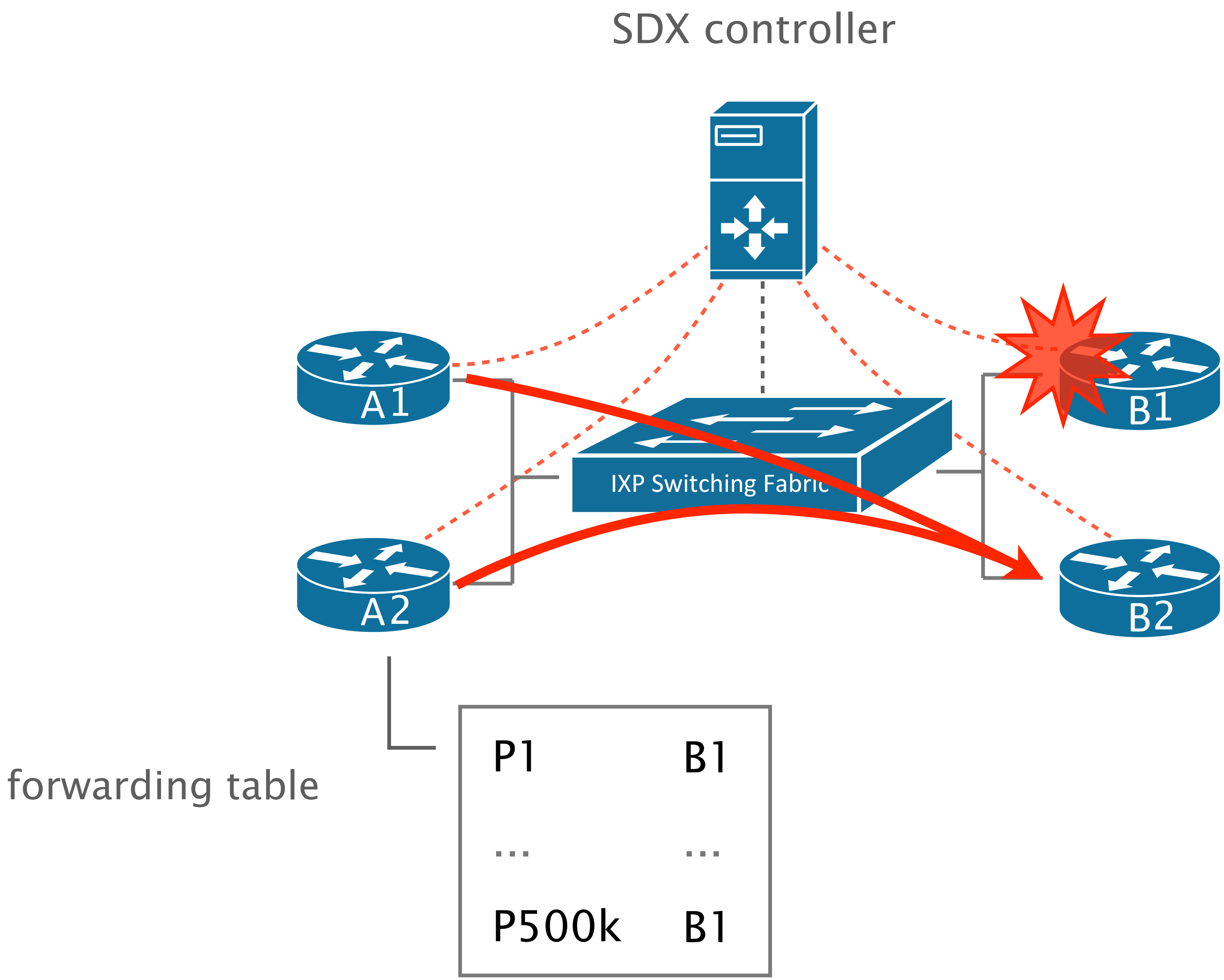


Upon a peer failure, the SDX controller directly pushes next-hop rewrite rules



```
match(srcmac:A1, dstmac:B1), rewrite(dstmac:B2), fwd(B2)
match(srcmac:A2, dstmac:B1), rewrite(dstmac:B2), fwd(B2)
```

All BGP traffic immediately moves from B1 to B2, independently of the number of FIB updates



SDX data-plane can enable sub-second, prefix-independent BGP convergence

controller
communication time

$$\text{convergence time} \quad \#\,\text{edge entries} \; * \; 150 \; \frac{\mu\text{secs}}{\text{entry}} \; + \; 30\text{\textasciitilde}50\ \text{ms}$$

average update time per entry

SDX data-plane can enable sub-second, prefix-independent BGP convergence

$$\text{convergence time} = \#\,\text{edge entries} * 150\ \frac{\mu\text{secs}}{\text{entry}} + 30\sim50\ \text{ms}$$

$$= \mathbf{O(30\sim50)\ ms}$$

SDX enables a wide range of novel applications

security	Prevent/block policy violation
	Prevent participants communication
forwarding optimization	Middlebox traffic steering
	Traffic offloading
	Inbound Traffic Engineering
	Fast convergence
peering	Application-specific peering
remote-control	Upstream blocking of DoS attacks
	Influence BGP path selection
	Wide-area load balancing

DNS-based wide-area load balancing has several limitations

High TTL values lead to slow recovery when a device fails

due to caching by local DNS servers and browsers

Low TTL values lead to higher delay for DNS resolution

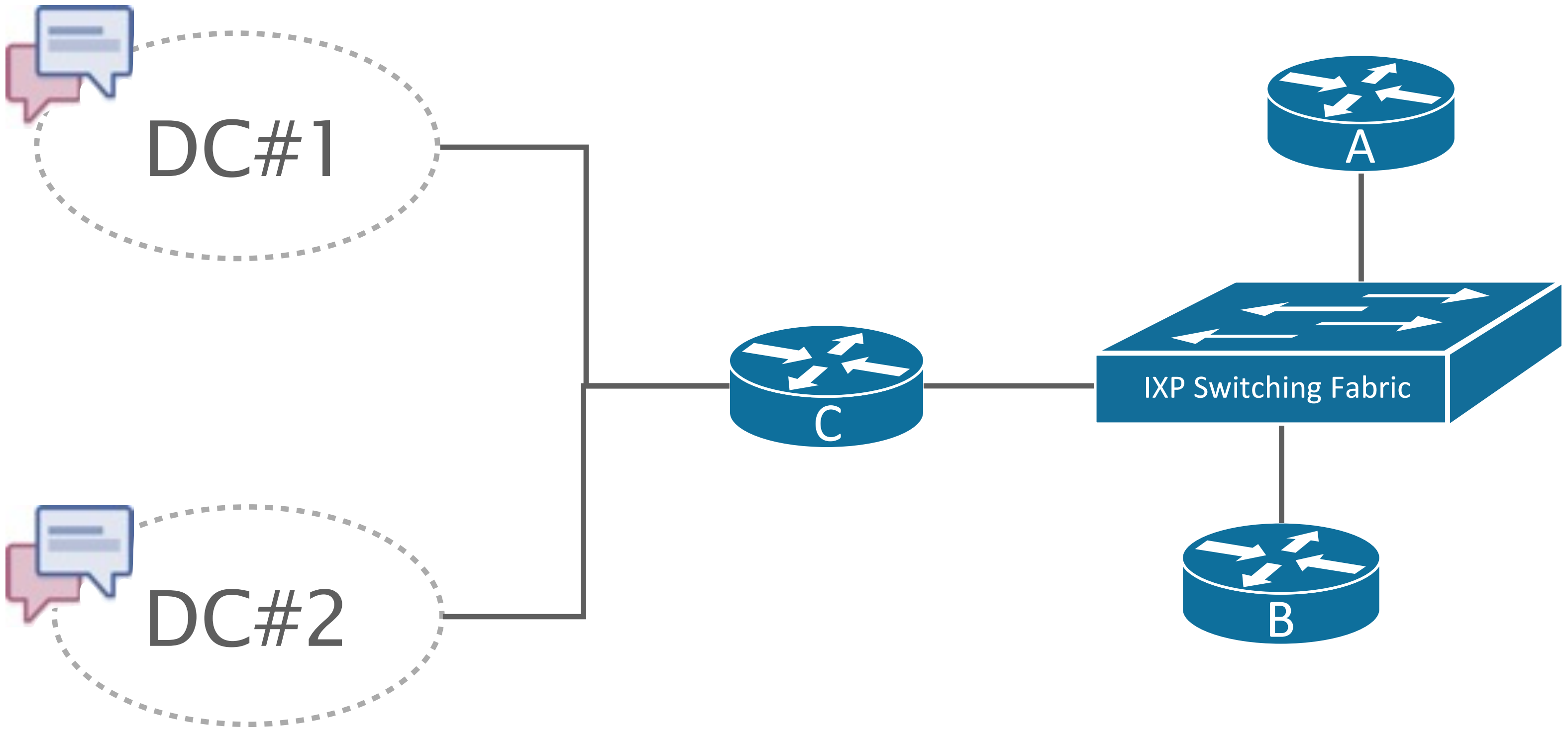
due to cache misses

Load-balancing is not based on the client IP address

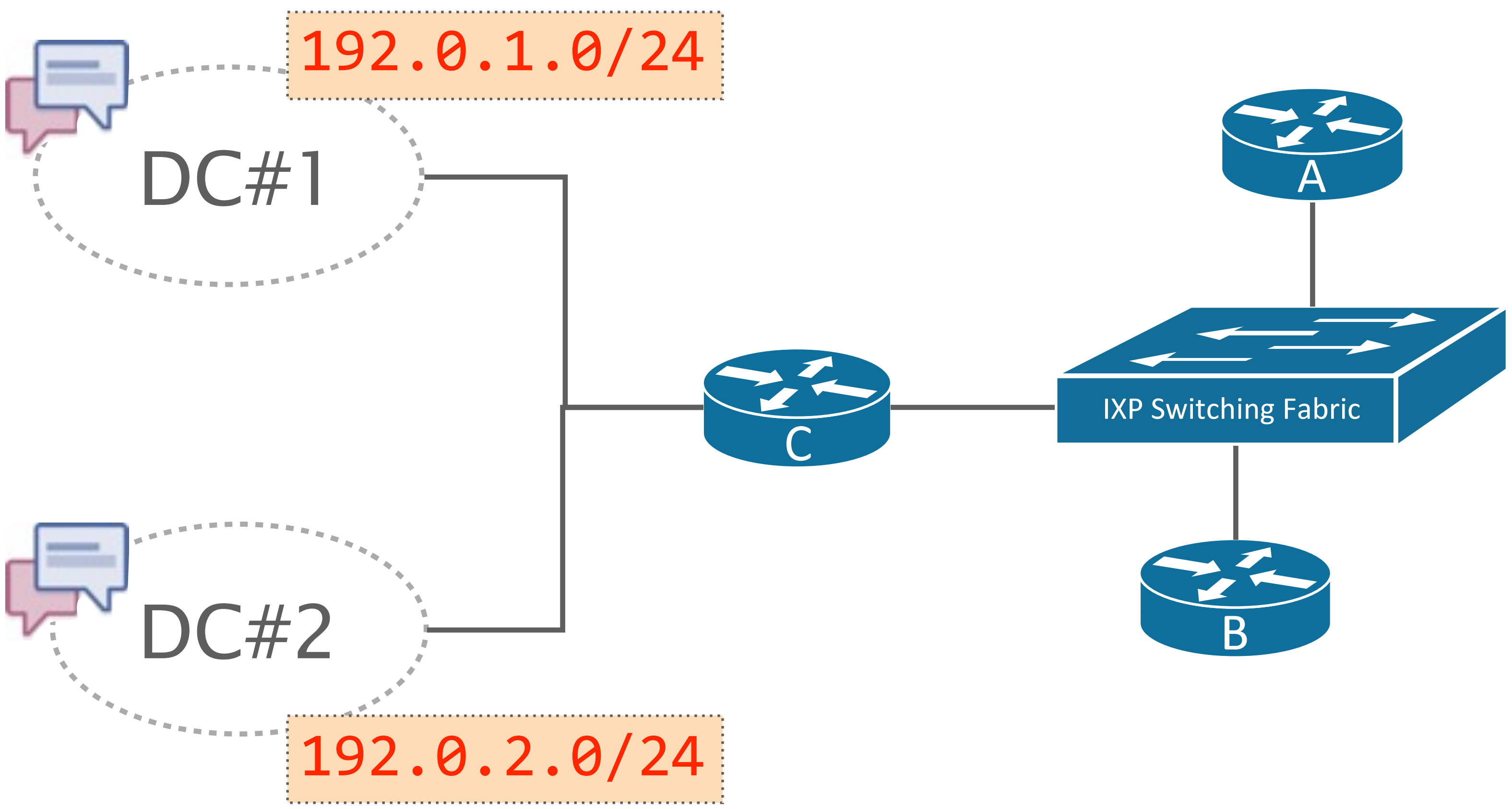
but on the DNS resolver IP address (*e.g.*, 8.8.8.8)

SDX enable direct and quick control of traffic redirection

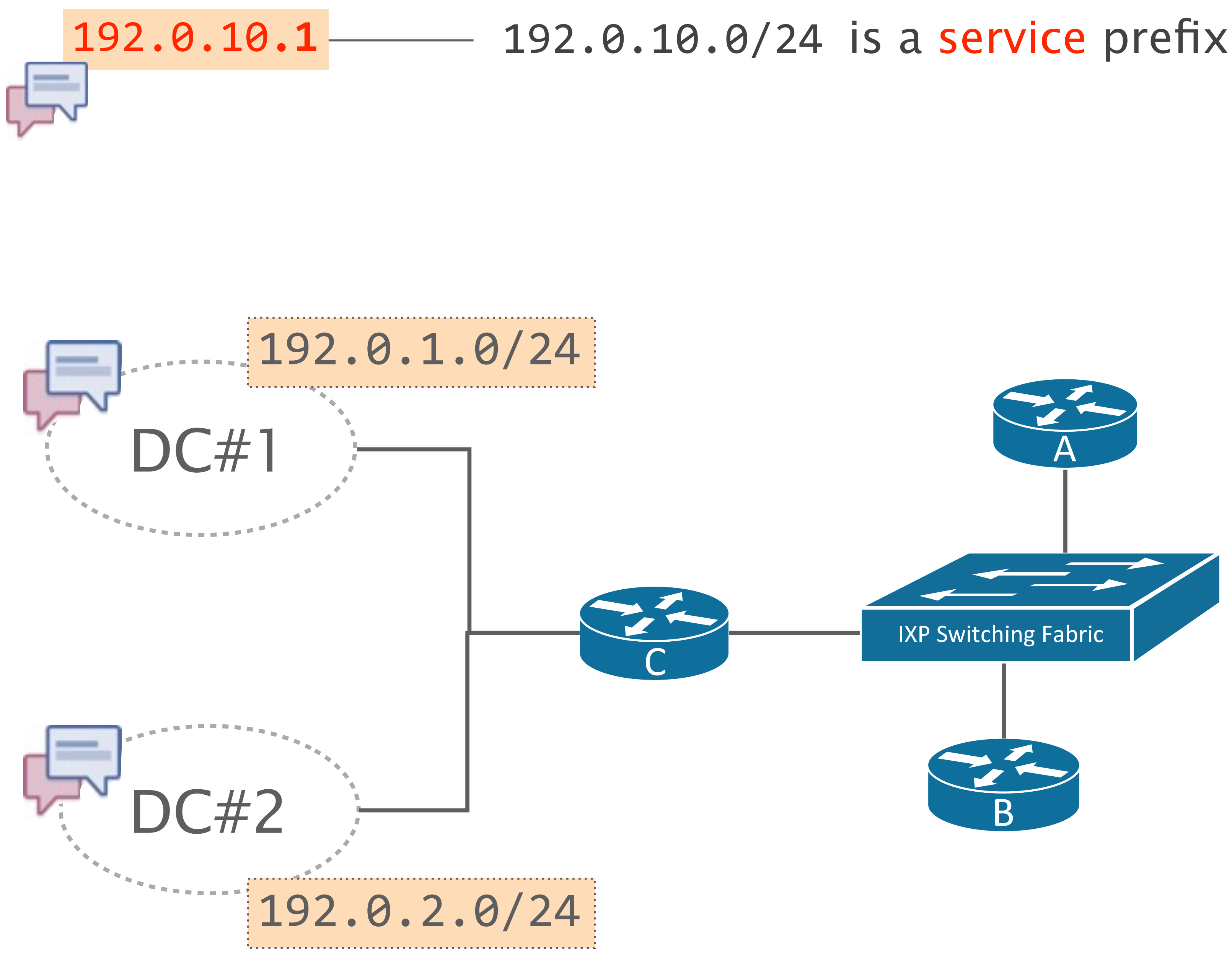
Let's consider a CDN *C* that provides one service at two Data Centers (DC)



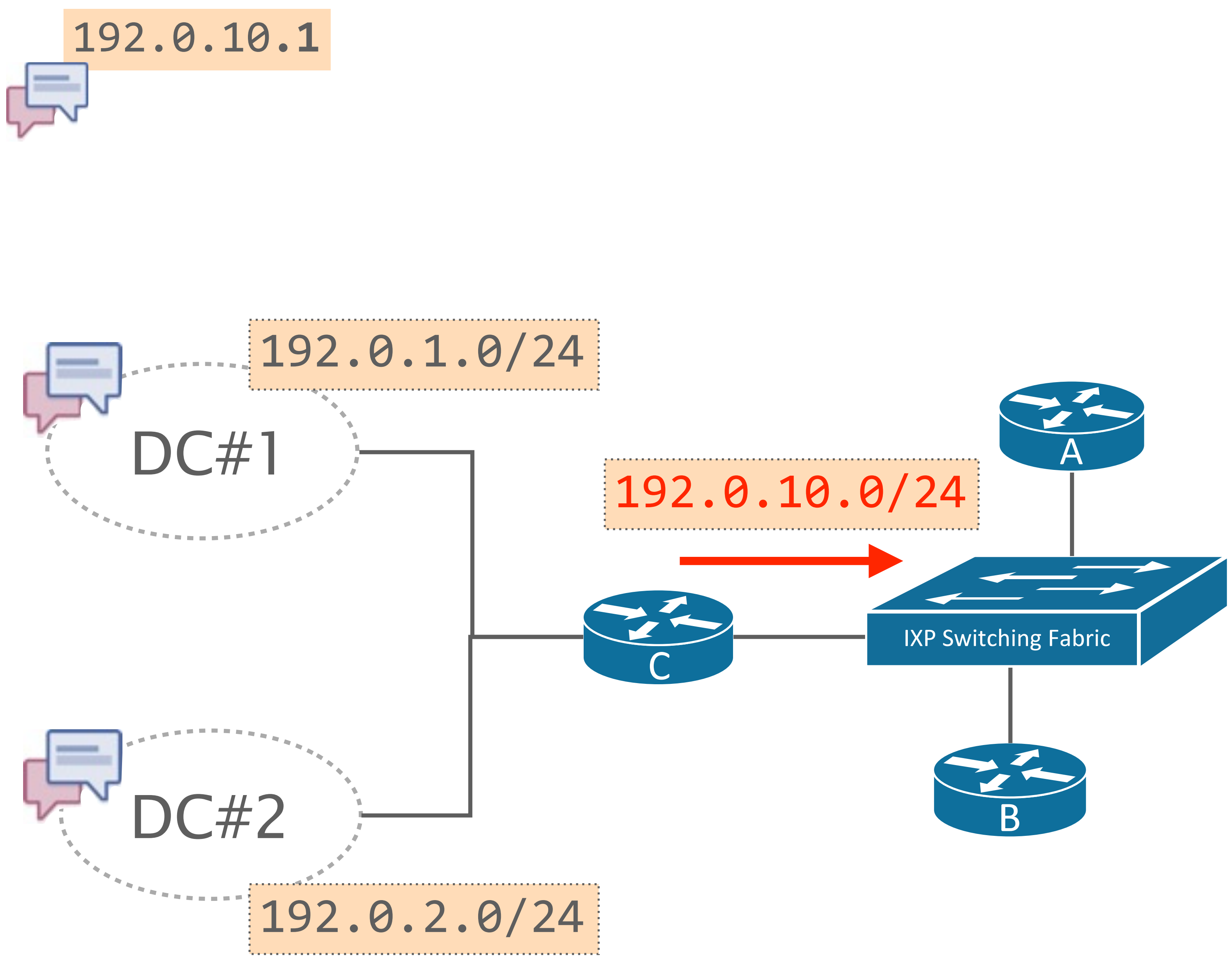
C assigns one IP prefixes per DC



C assigns one IP address identifying the service



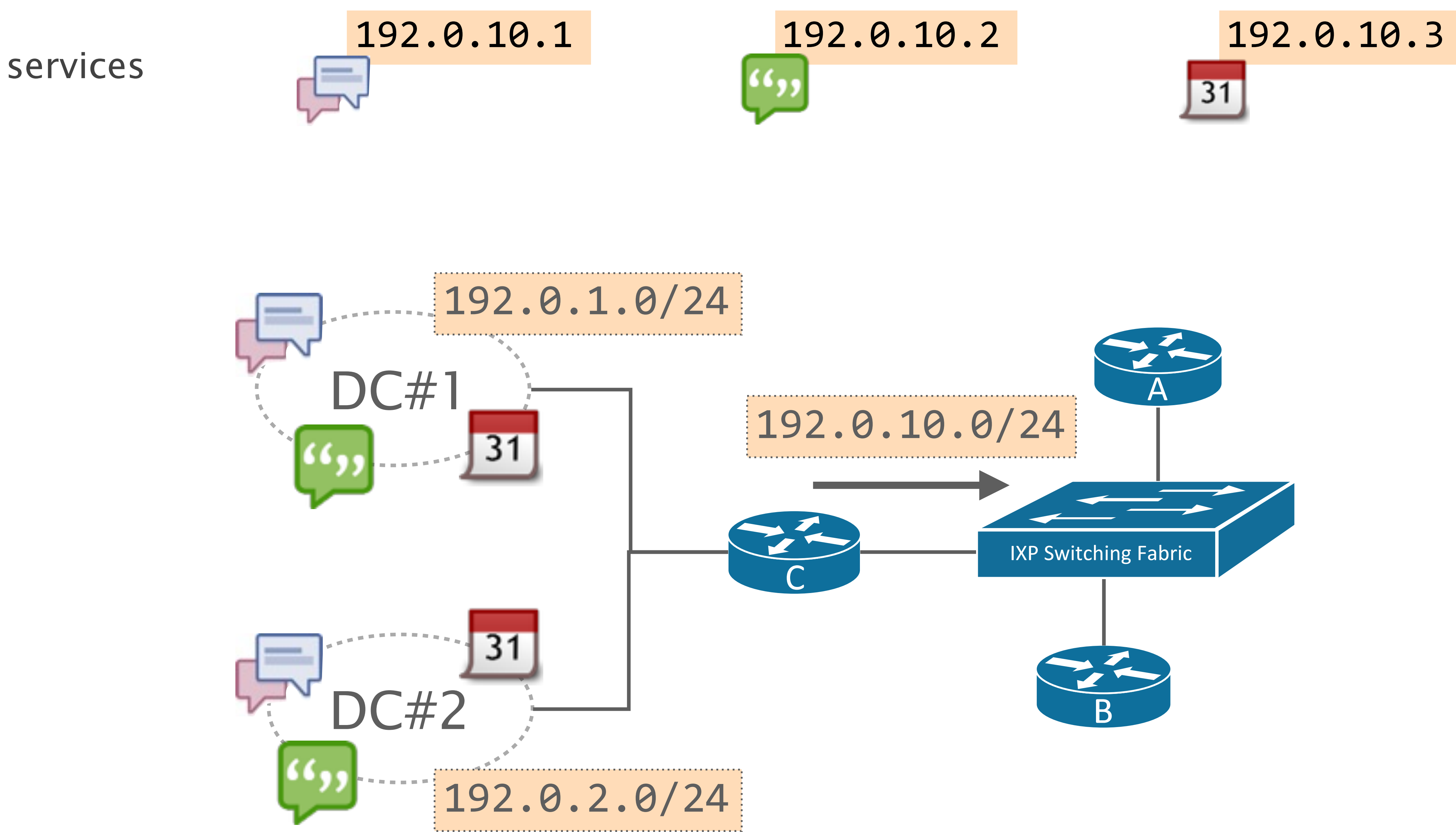
C announces the service prefix at the IXP



C directs the service traffic to the appropriate DC based on the client's IP address

```
match(dstip=192.0.10.1) then
  (match(srcip=0.0.0.0/1) then
    mod(dstip=192.0.1.161)) and       // forward to DC1
  (match(srcip=128.0.0.0/1) then
    mod(dstip=192.0.2.139))           // forward to DC2
```

SDX based wide-area load-balancing works for any number of services and data centers



SDX enables direct and quick control of traffic redirection

SDX-based load-balancing is

fast	no DNS caching problem
flexible	use of any load-balancing algorithm
efficient	based on the actual client IP address

On integrating Software-Defined Networking within existing routing systems

SDN-controlled routers

don’t trash, recycle

SDN-controlled IGP

fine-grained traffic engineering

SDN-controlled BGP

inter domain bonanza

SDN-controlled routing enables to realize parts of the SDN promises today, on an existing network

Facilitate a complete transition to SDN

provide *one interface to rule them all*

Simplify the controller implementation

most of the work is still done by the routers

Maintain operators' mental model

same good old protocols running, easier troubleshooting

On integrating Software-Defined Networking within existing routing systems

Laurent Vanbever

Google, Mountain View

www.vanbever.eu

vanbever@cs.princeton.edu