

# Hardware-Accelerated Network Control Planes

**Edgar Costa Molero<sup>(1)</sup>,**

Stefano Vissicchio<sup>(2)</sup>, Laurent Vanbever<sup>(1)</sup>

(1)

**ETH** zürich

(2)



Modern networks architectures are split in (at least) two planes

Modern networks architectures are  
split in (at least) two planes

data plane

control plane

# Network planes can be implemented in both software or hardware

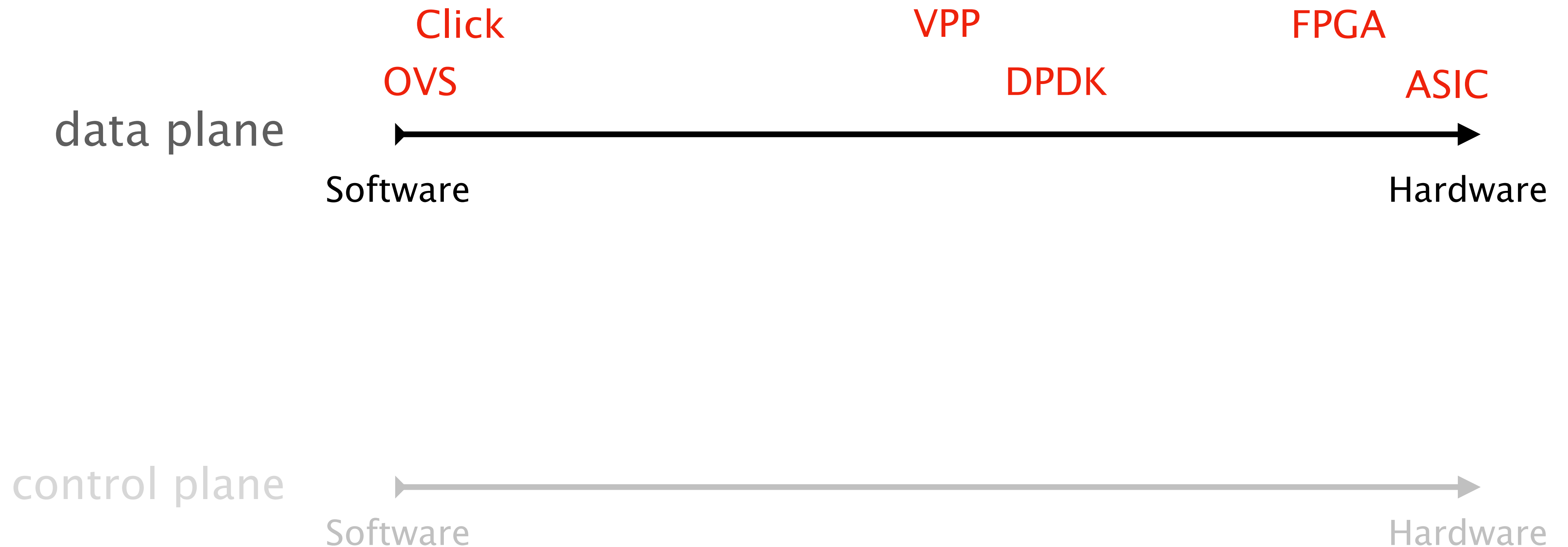
data plane



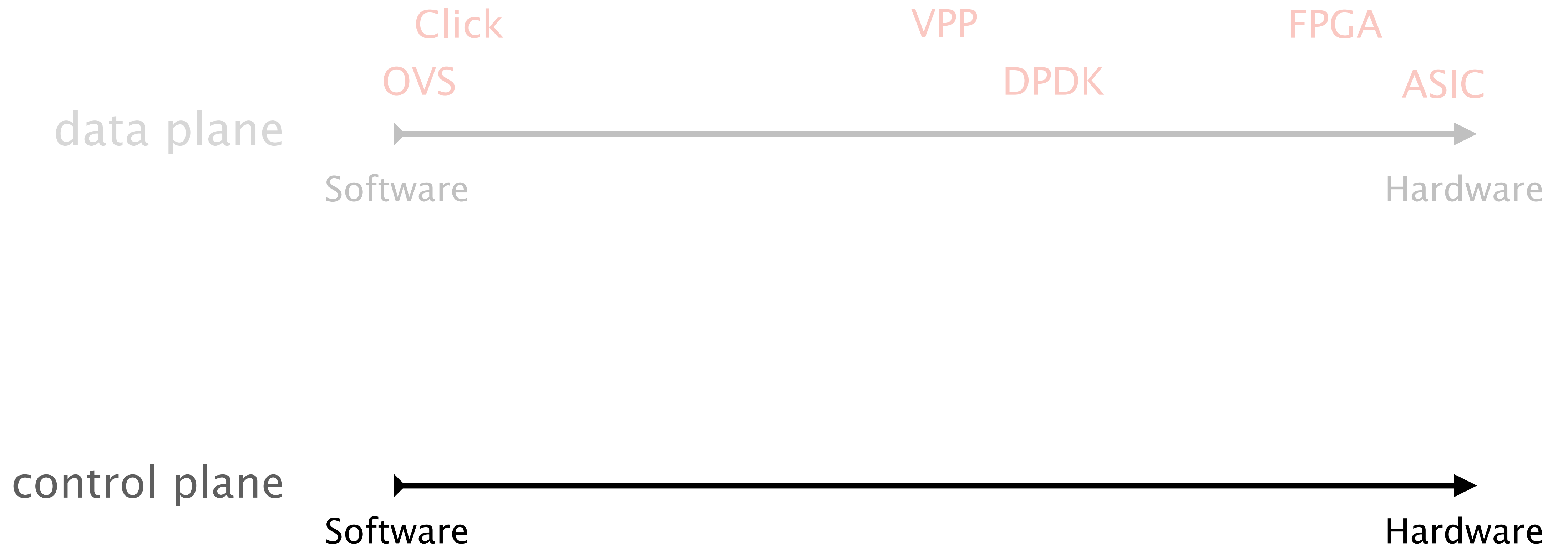
control plane



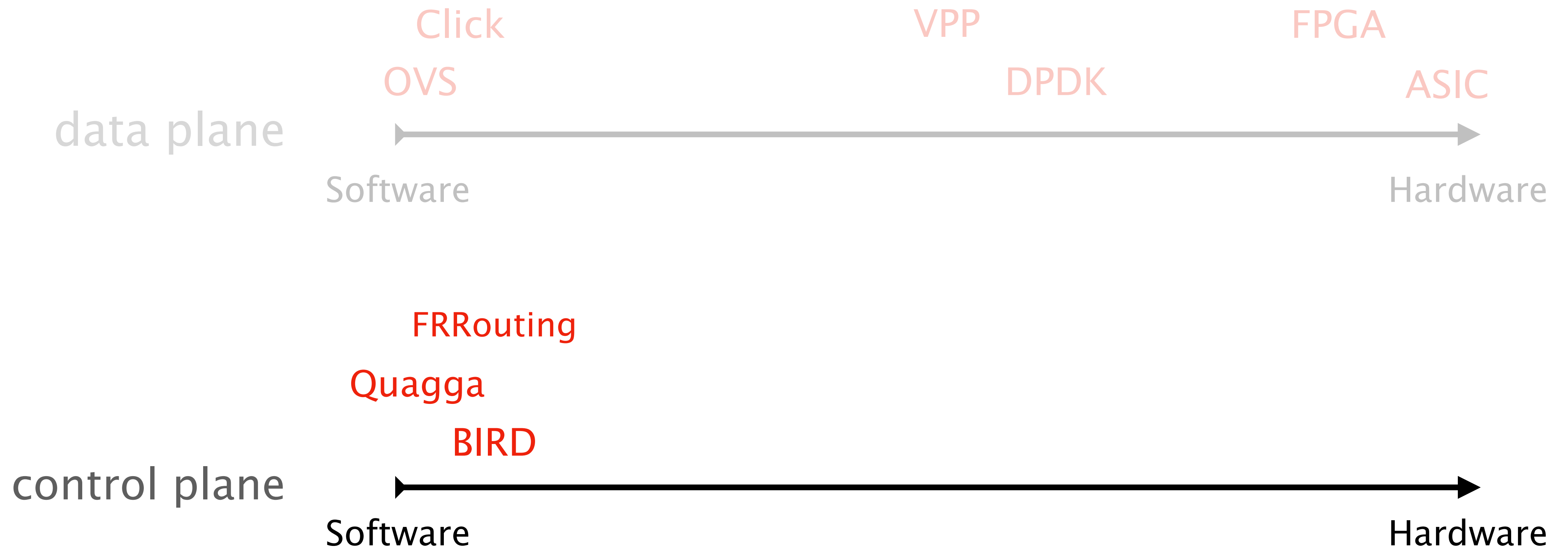
# Existing data plane implementations cover the entire software/hardware **spectrum**

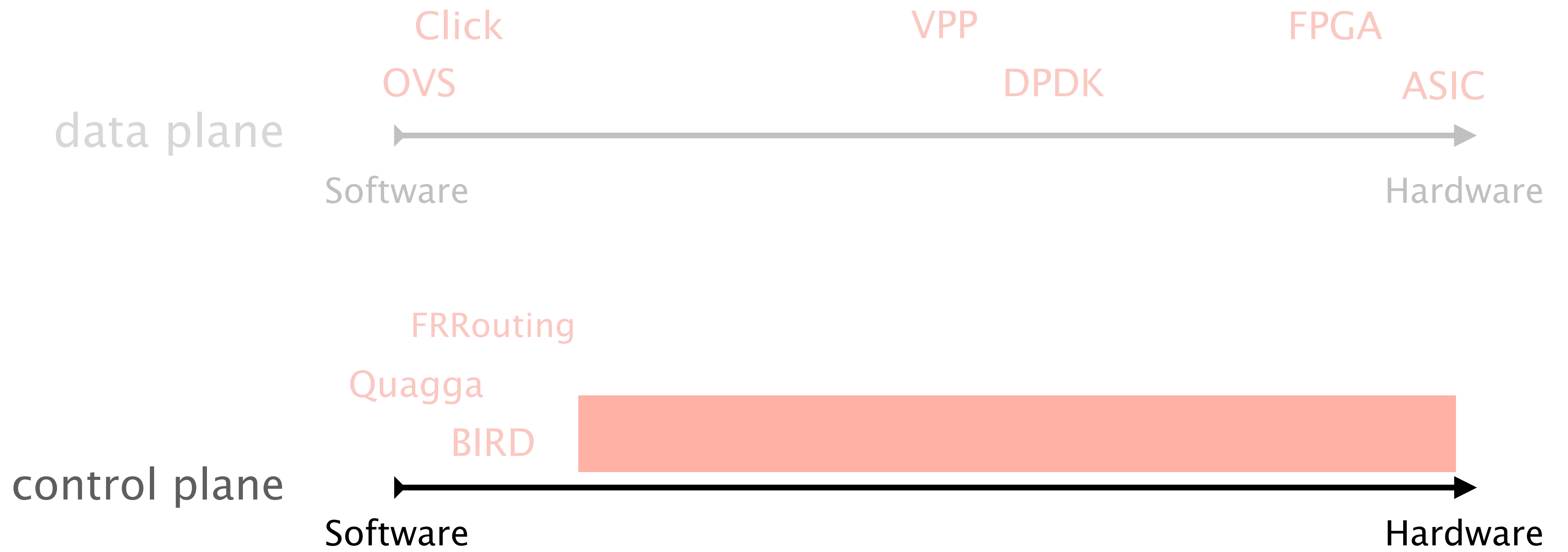


# What about the control plane ?

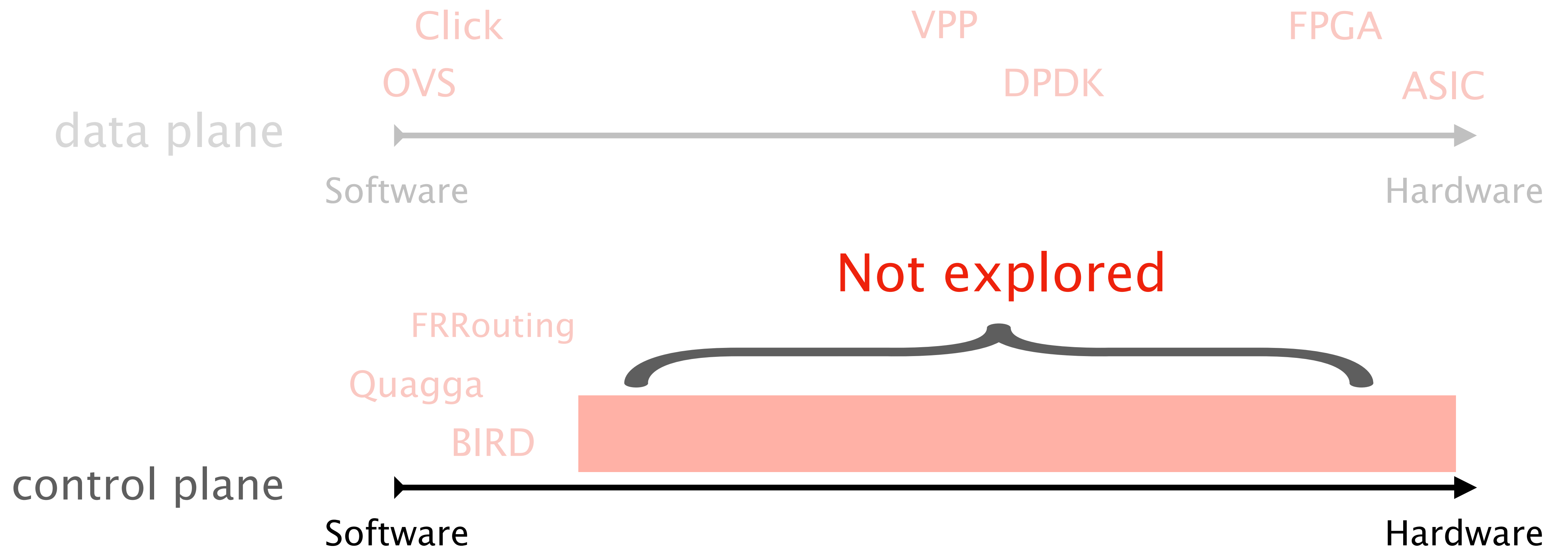


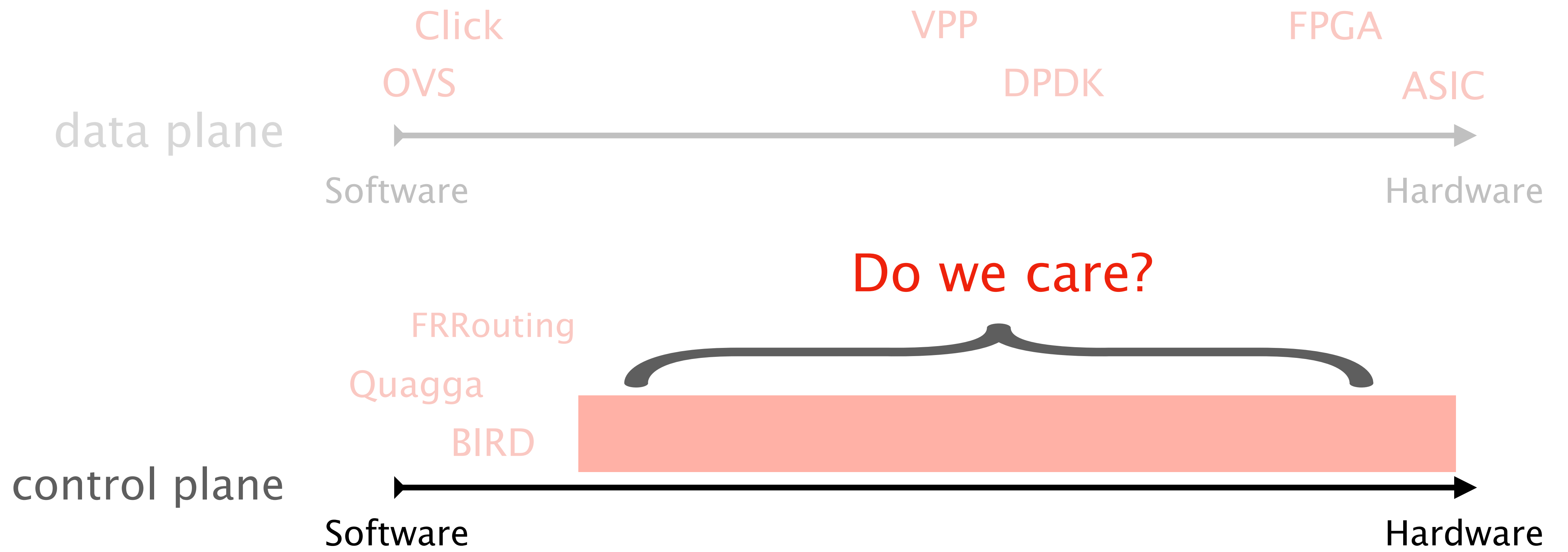
# Control plane implementations make seldom use of the **hardware** resources











Even state-of-the-art software control planes  
have room for **improvement**

Even state-of-the-art software control planes  
have room for **improvement**

1 **React**

It can take up to a minute  
to detect normal failures

Even state-of-the-art software control planes  
have room for **improvement**

1 React

2 **Compute**

~1.5 minutes to converge the  
control plane of an IXP route server

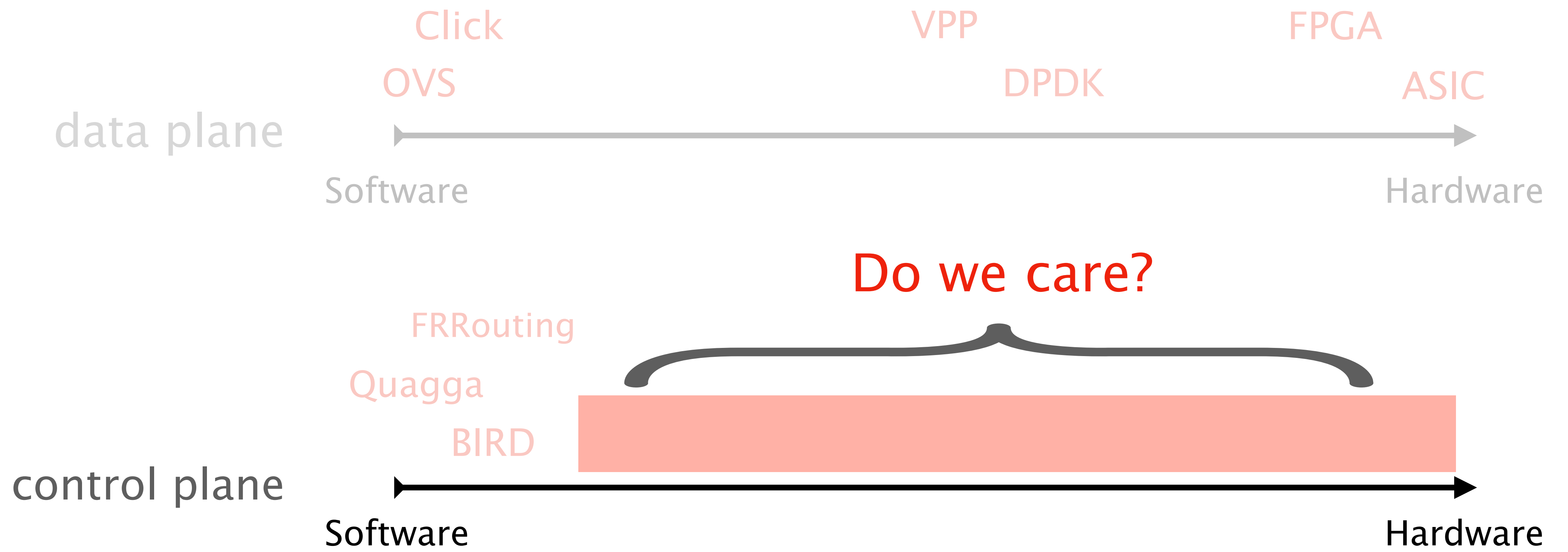
Even state-of-the-art software control planes  
have room for **improvement**

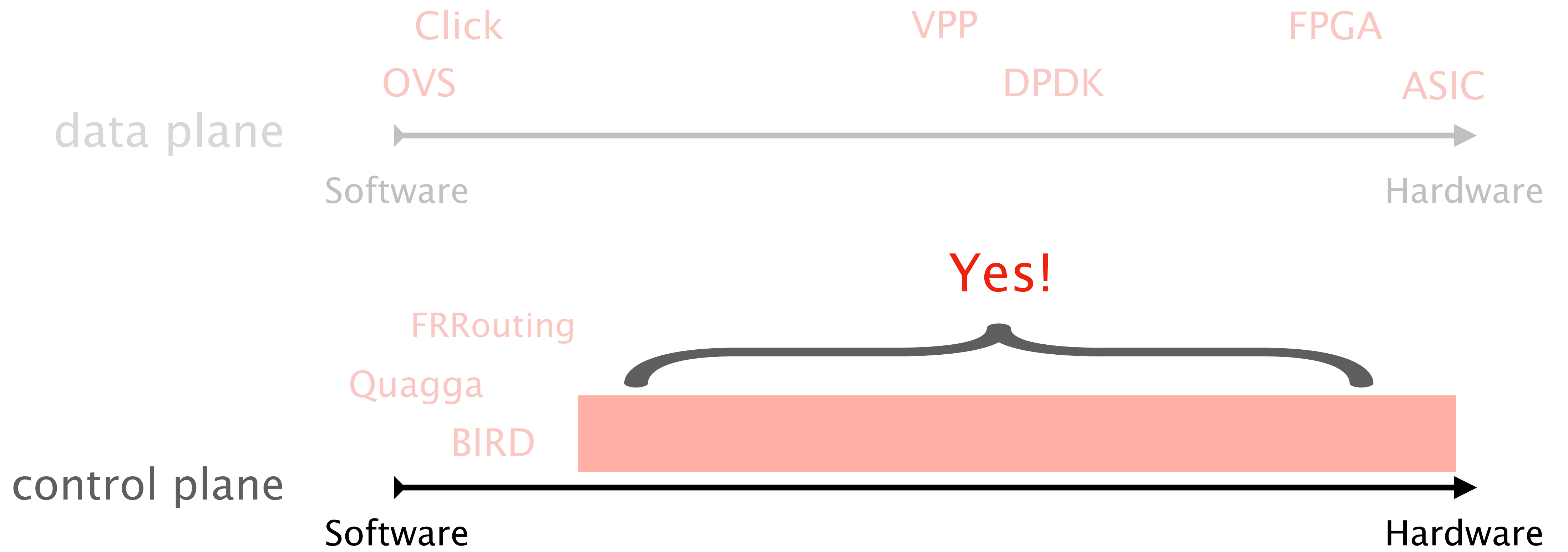
1 React

2 Compute

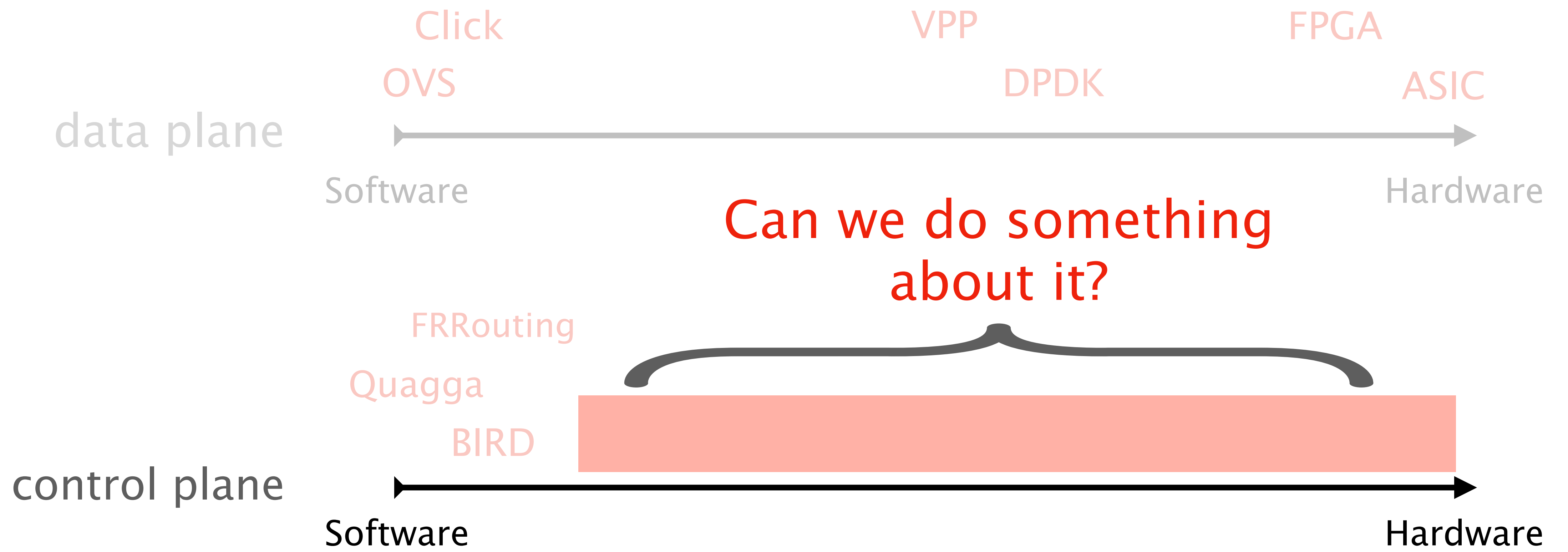
**3 Update**

$O(100\mu\text{s})$  to update a forwarding  
entry





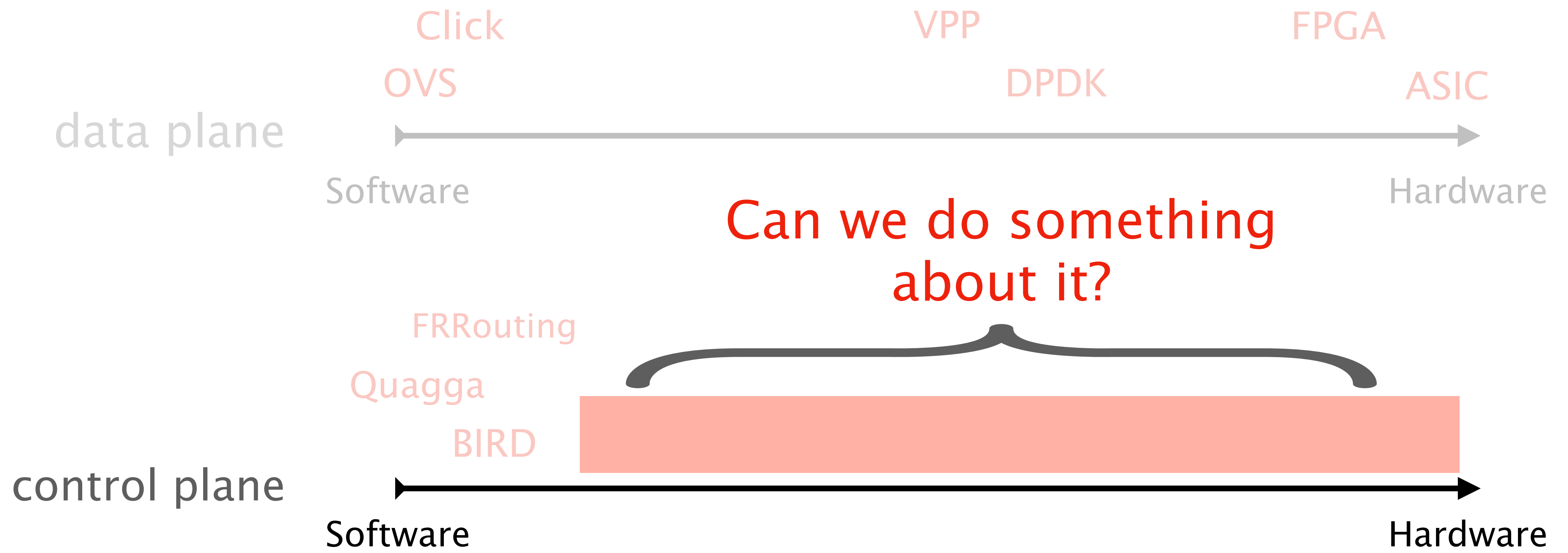


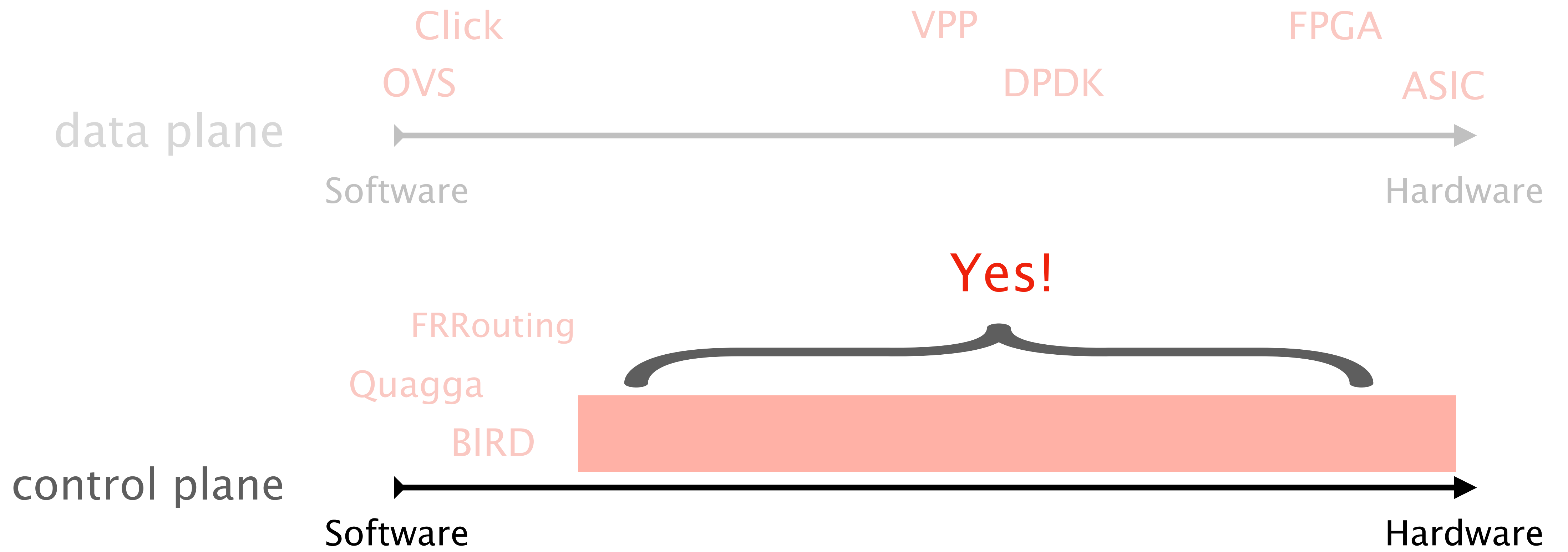


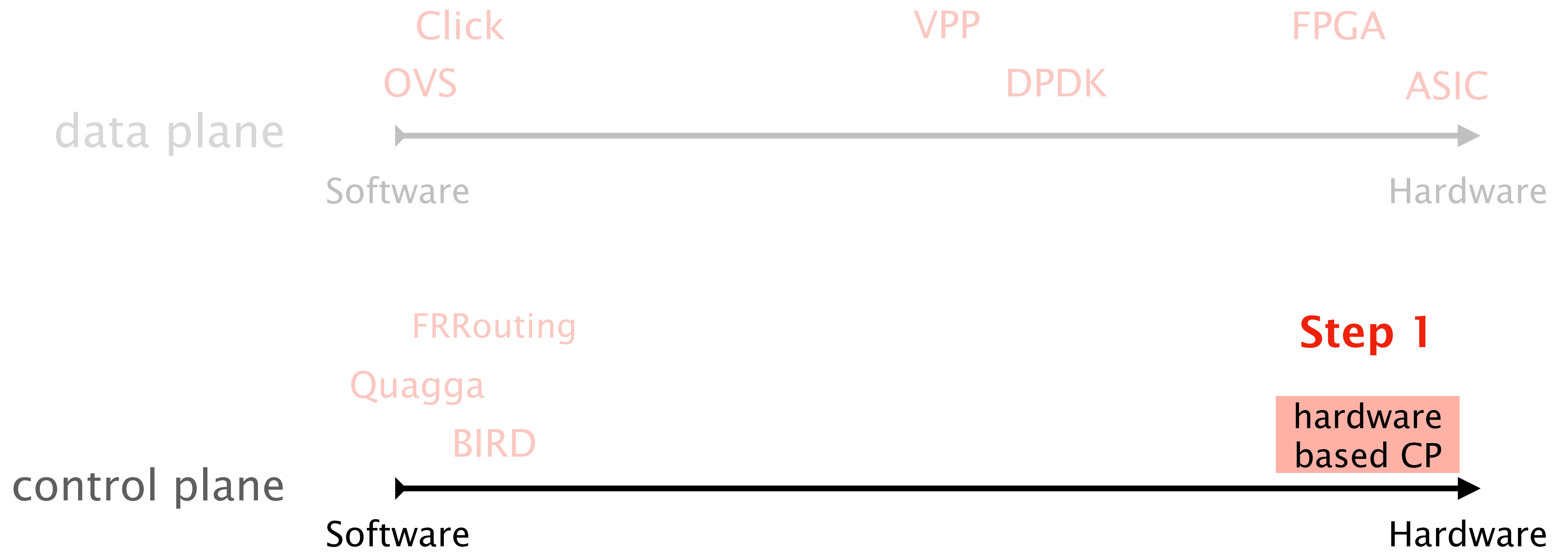
Modern programmable devices can perform computations on billions of packets per second

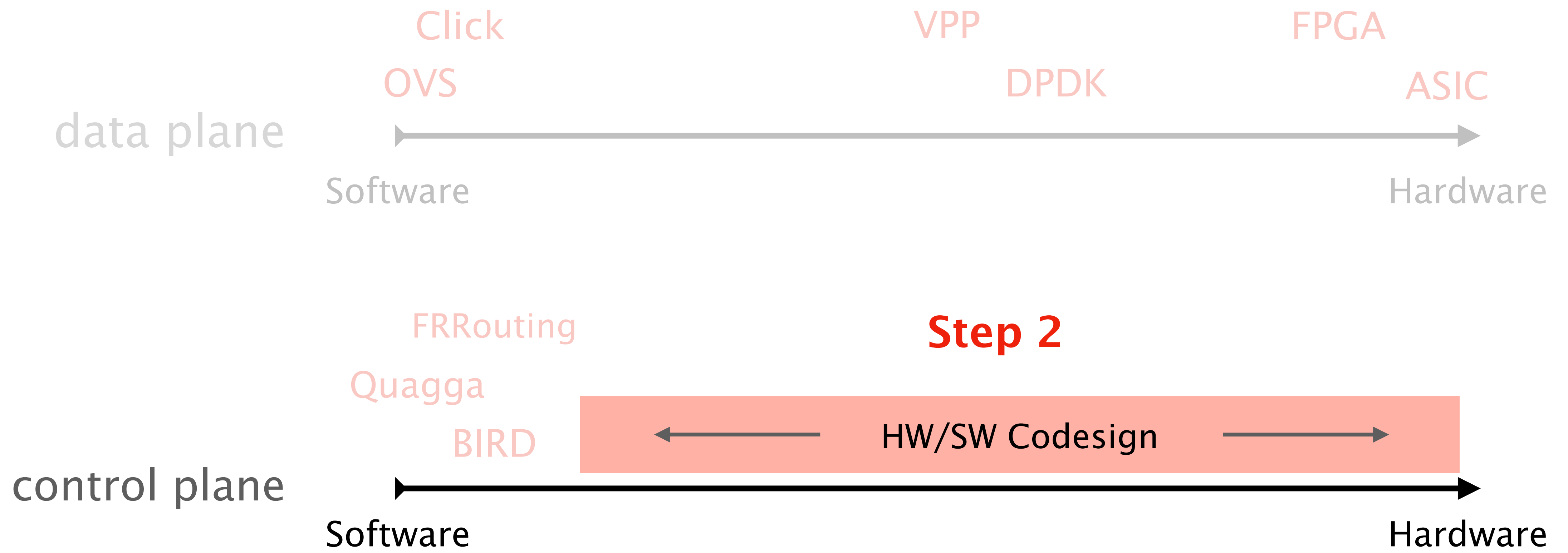
# Modern programmable devices can perform computations on billions of packets per second

- **Read & modify packet headers**  
e.g. to update network state
- **Basic operations**  
e.g. min & max
- **Add or remove custom headers**  
e.g. to carry routing information
- **Keep state**  
e.g. to save best paths









Main **tasks** to compute forwarding state are...

1 **Sensing**

monitors network  
to detect changes



Main **tasks** to compute forwarding state are...

1 Sensing

2 **Notification**

exchanges with network devices  
all the information learnt

# Main **tasks** to compute forwarding state are...

- 1 Sensing
- 2 Notification
- 3 Computation**

Computes forwarding paths when network changes are detected

Updates the data plane accordingly

# Hardware-based network sensing

Goal

Challenges

# Hardware-based network sensing

Goal

Detect both hard and gray failures

Challenges

# Hardware-based network sensing

Goal

Detect both hard and gray failures

e.g. random drop, TCAM bit flips

Challenges

# Hardware-based network sensing

Goal

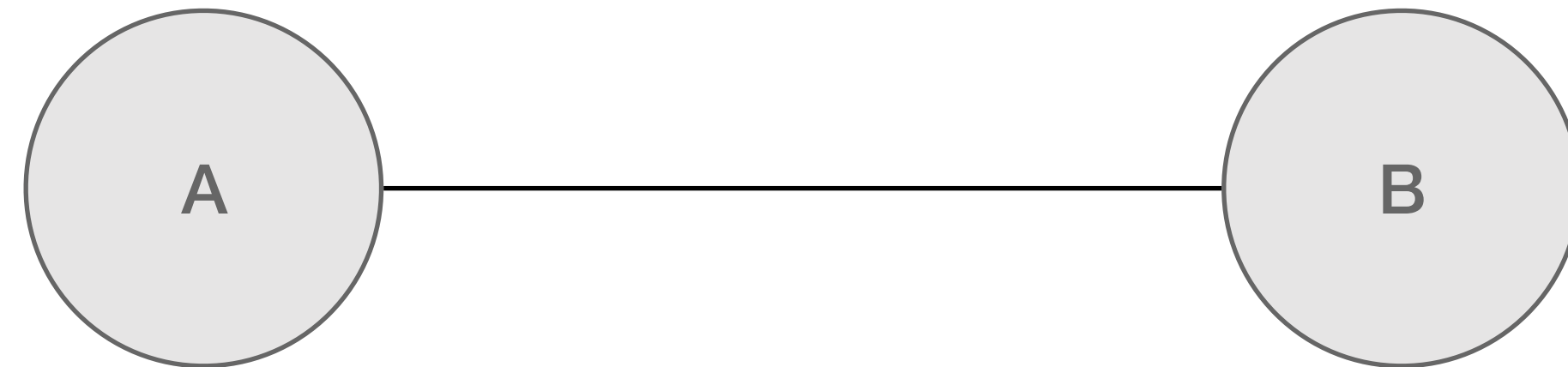
Detect both hard and **gray** failures

e.g. random drop, TCAM bit flips

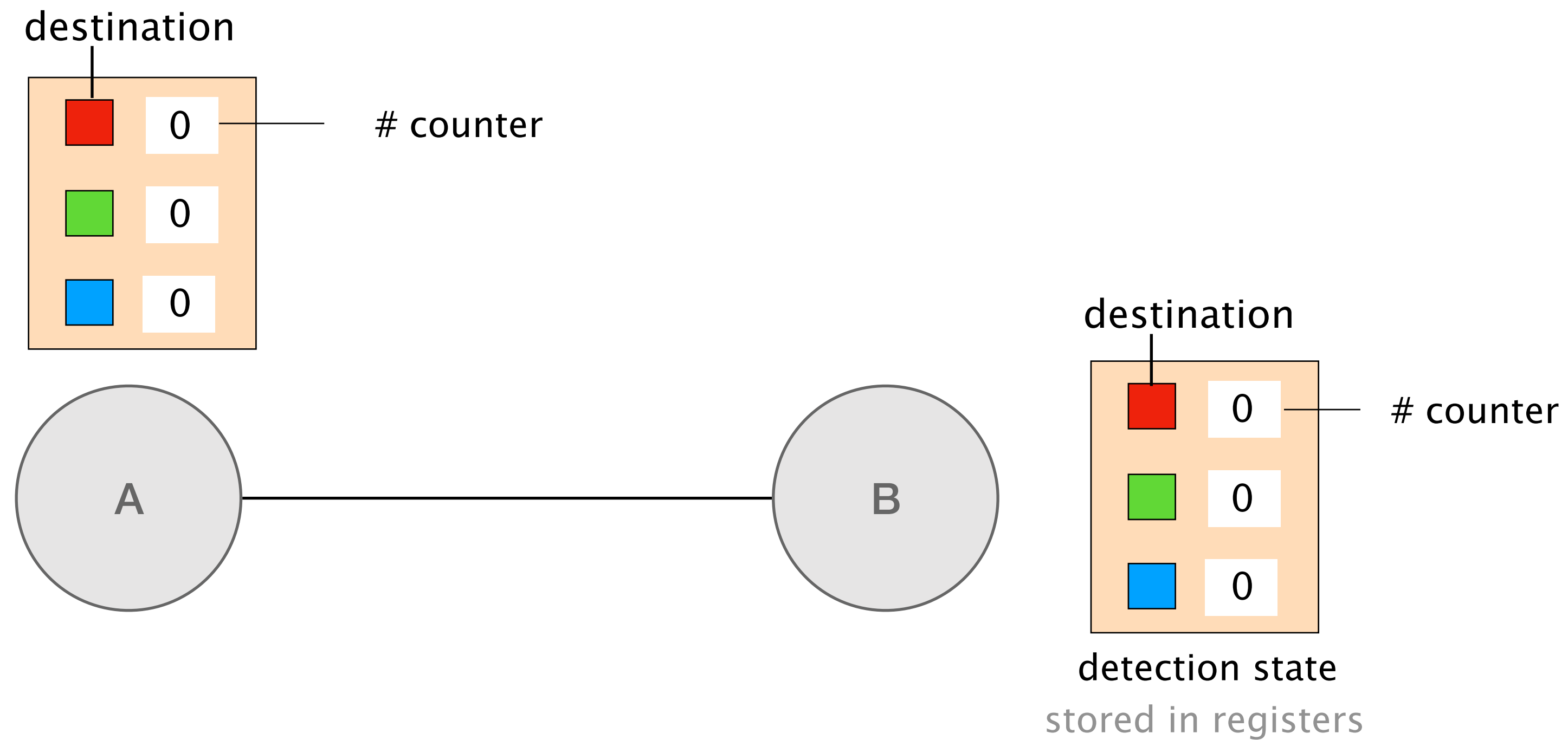
Challenges

Basic hello-based mechanisms are not enough

Switches **synchronously** exchange packet counts

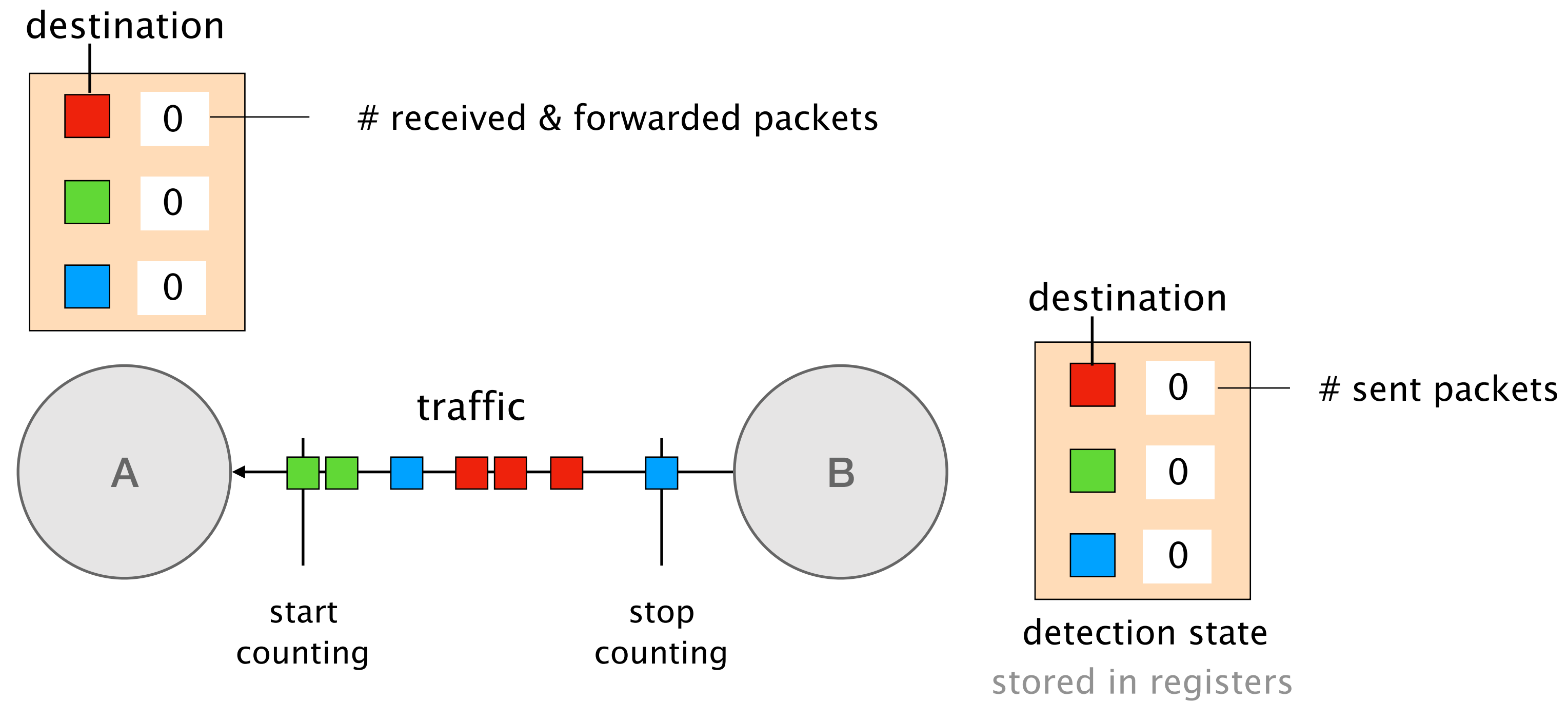


# Switches **synchronously** exchange packet counts

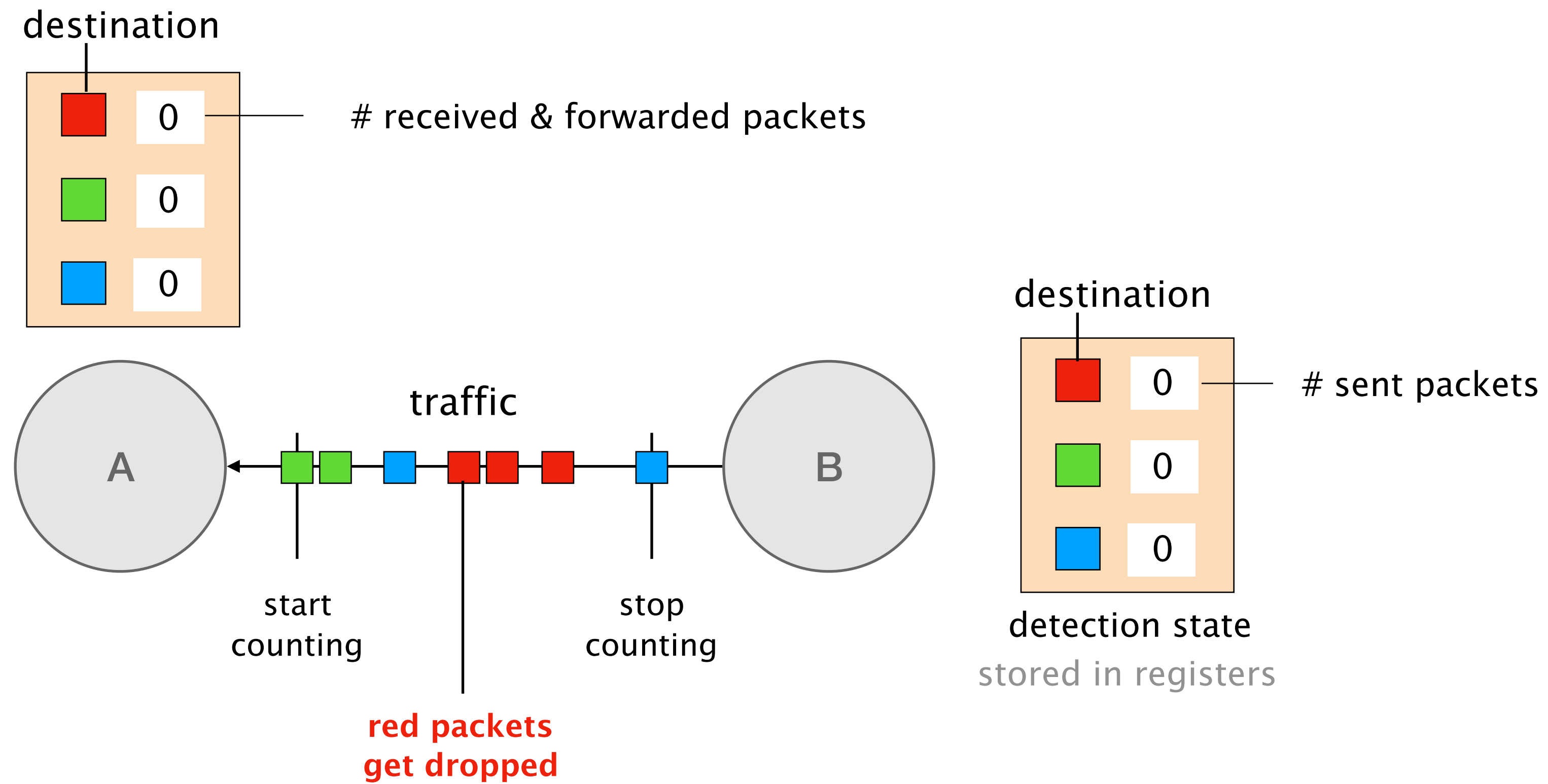




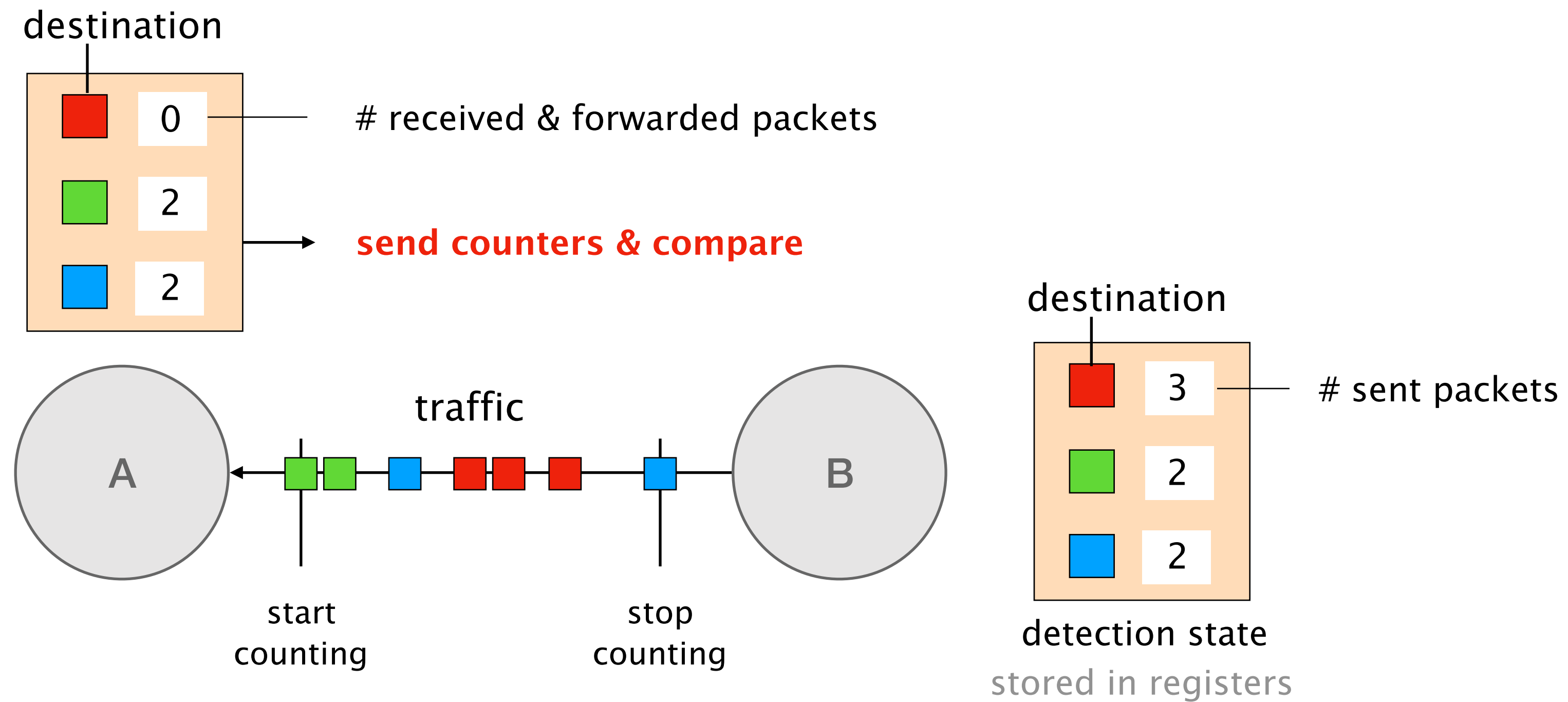
# Upstream switch starts probing campaigns



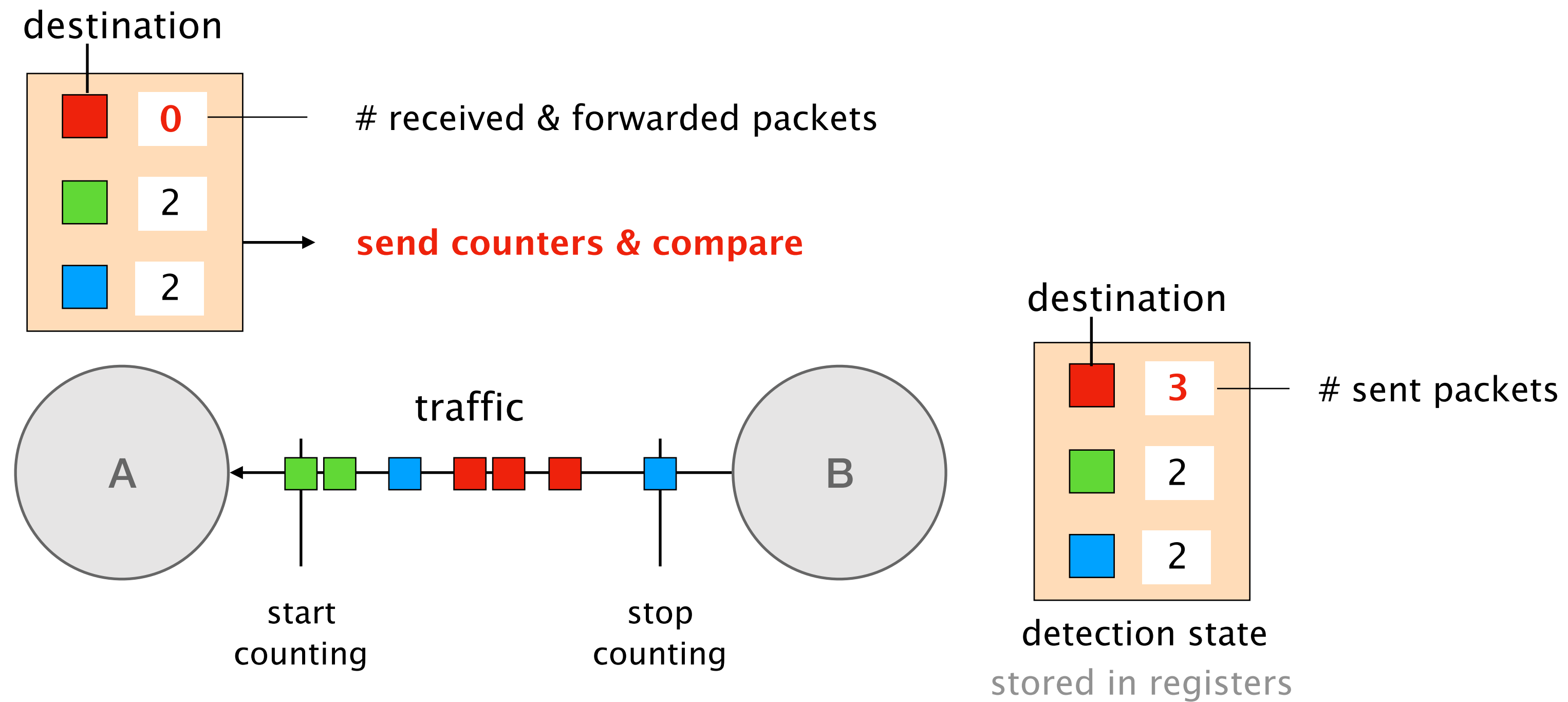
# Traffic for some prefixes gets dropped



# Downstream switch sends counters to upstream



# Upstream switch detects the failure by comparing counters



# Hardware-based notifications

Goal

Challenges

# Hardware-based notifications

Goal

Implement a broadcast notification mechanism in hardware

Challenges

# Hardware-based notifications

Goal

Implement a broadcast notification mechanism in hardware

Challenges

Avoid broadcast storms

Require reliable communication

# Hardware-based notifications

## Avoid broadcast storms

- ▶ Use per switch broadcast sequence numbers



# Hardware-based notifications

## Avoid broadcast storms

- ▶ Use per switch broadcast sequence numbers

## Require reliable communication

- ▶ Send notification duplicates
- ▶ Use maximum priority queues

# Hardware-based computation

Goal

Challenges

# Hardware-based computation

Goal

Run distributed routing algorithms in **hardware**  
e.g. path vector

Challenges

# Hardware-based computation

Goal

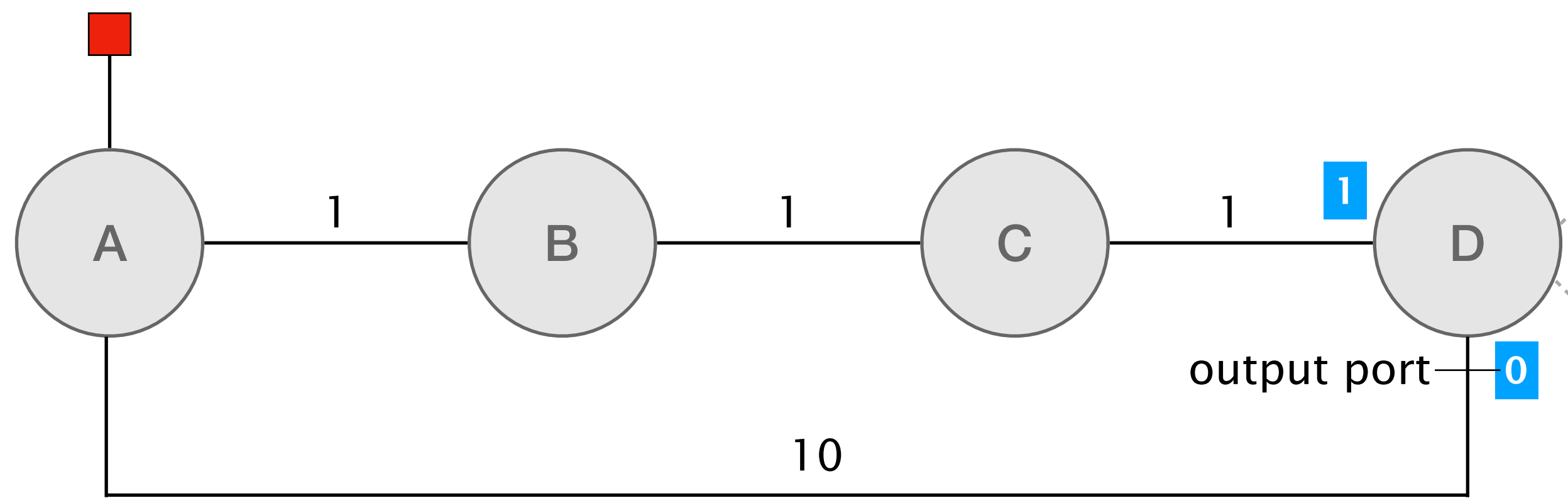
Run distributed routing algorithms in **hardware**

e.g. path vector


Challenges

Computation logic is limited

Resources are heavily limited









prefix-to-index

	50
...	

link cost

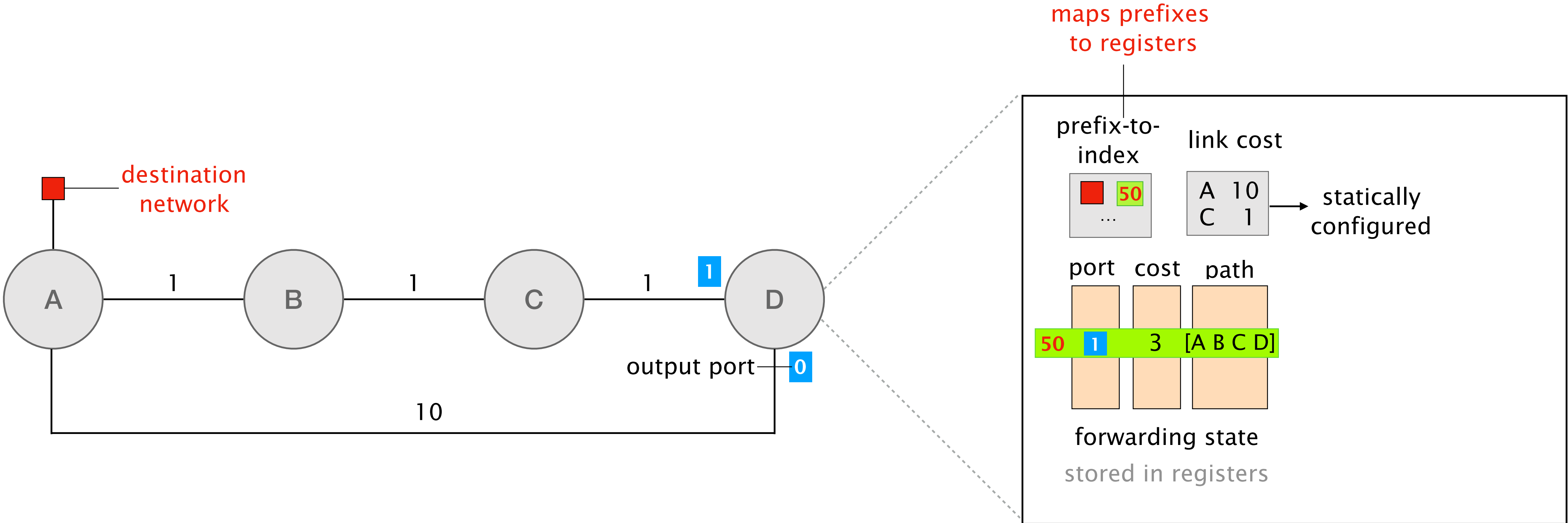
A	10
C	1

port cost path

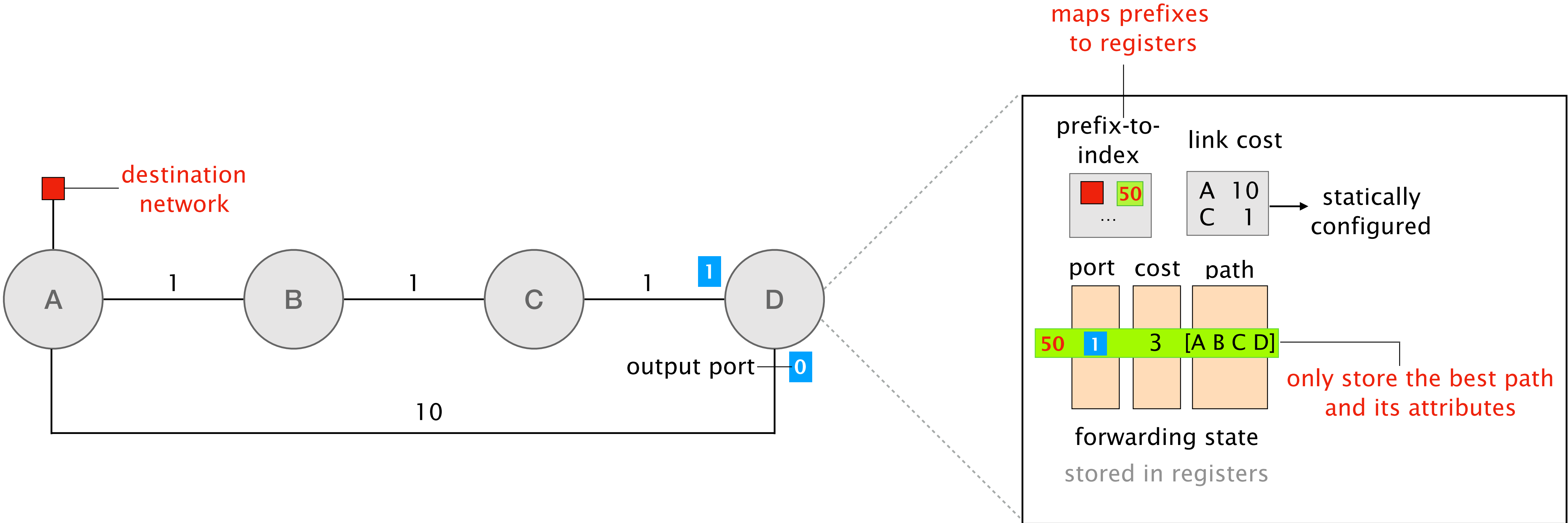
		
50	1	3 [A B C D]
		

forwarding state stored in registers

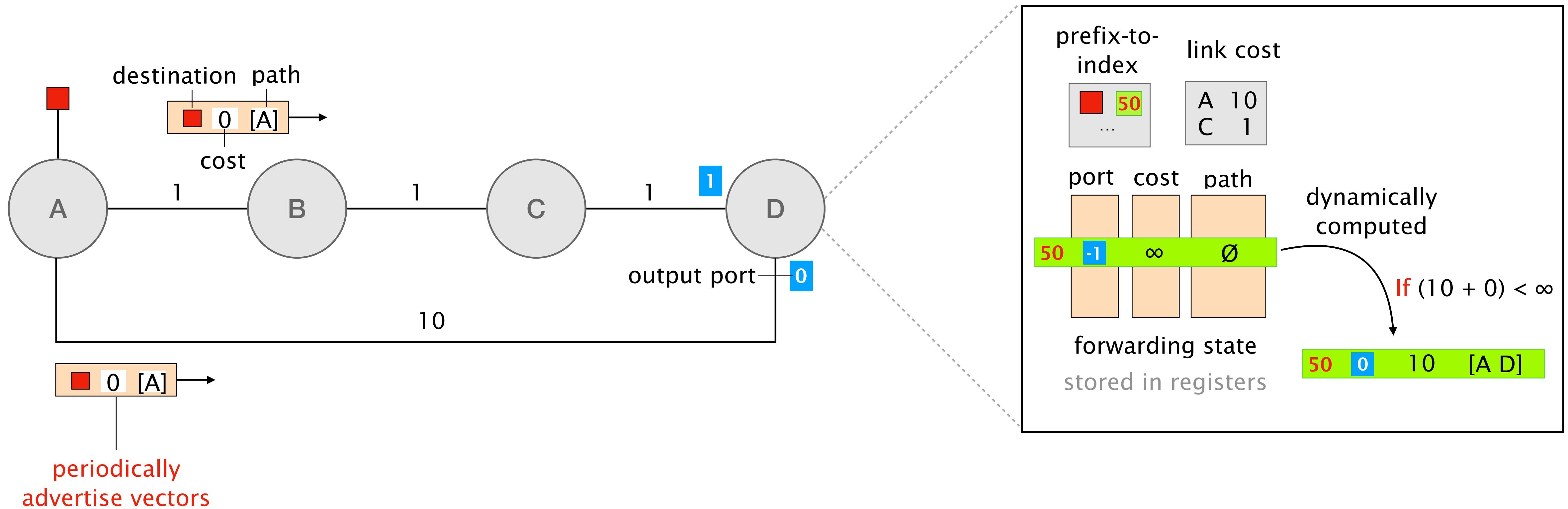
# Statically configured tables map prefixes to registers in memory



# Registers store best paths and its attributes

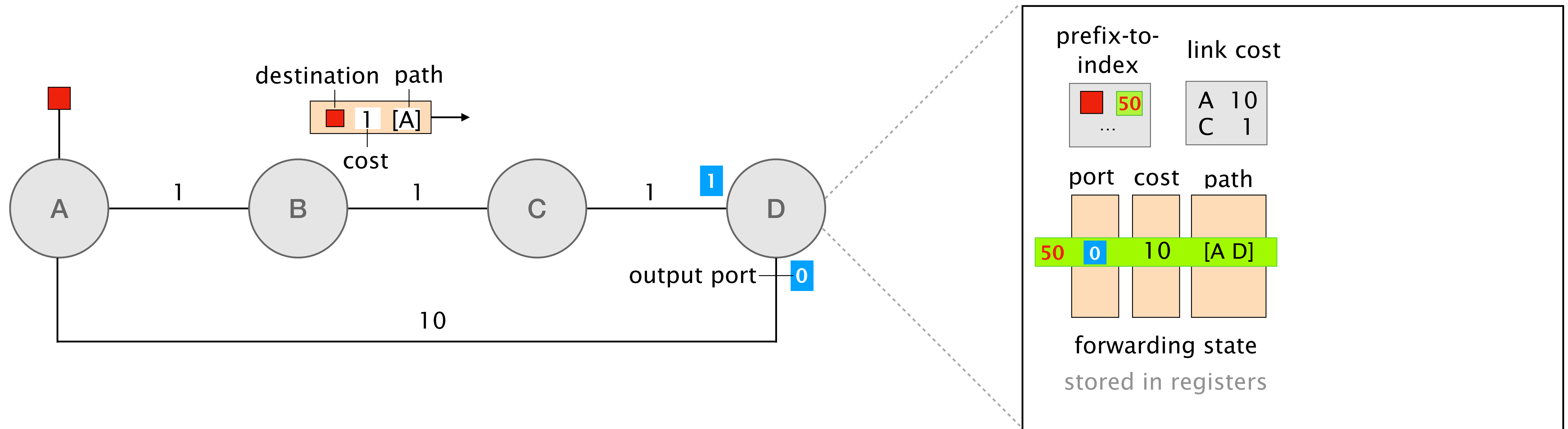


# Switches periodically advertise vectors to neighbors

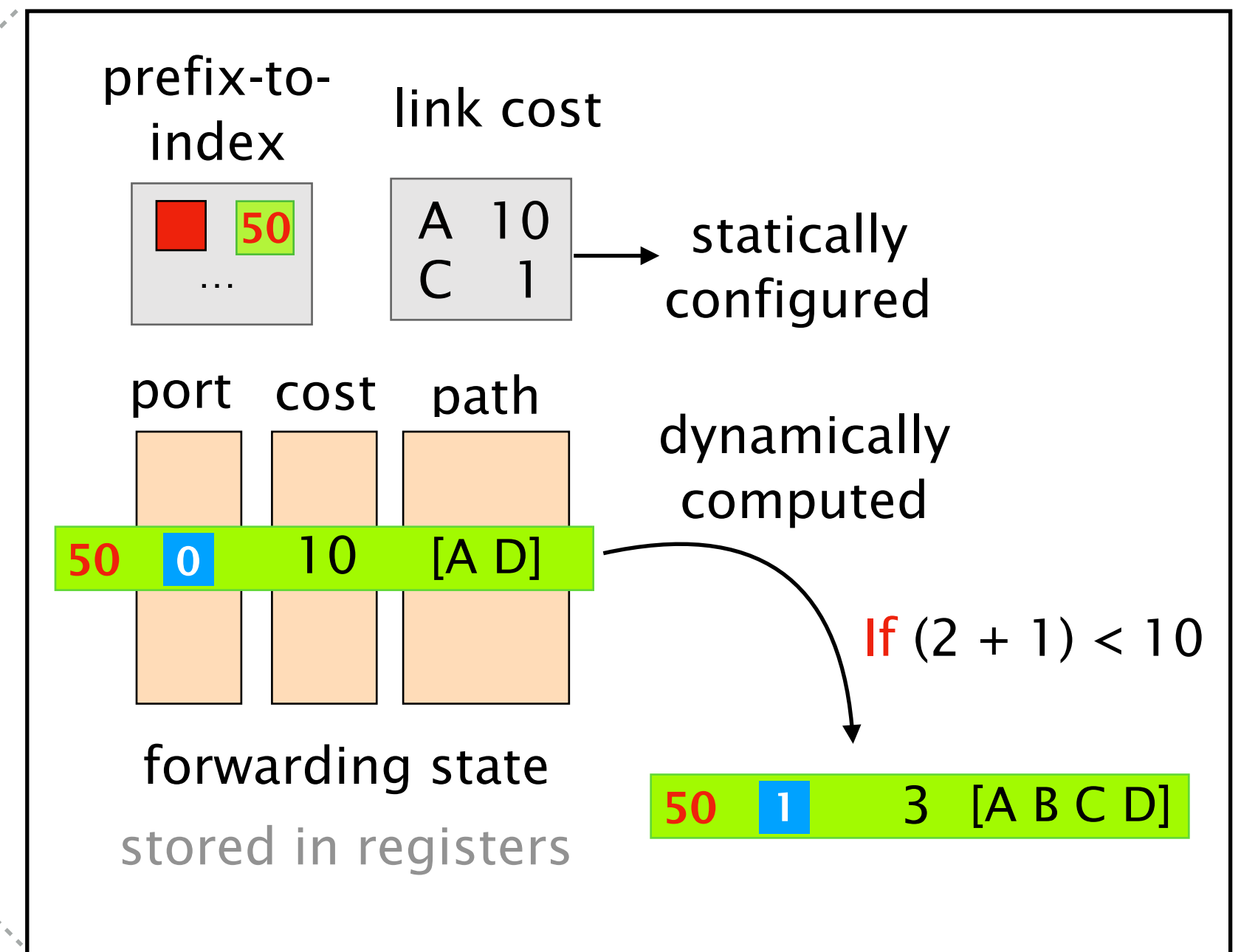
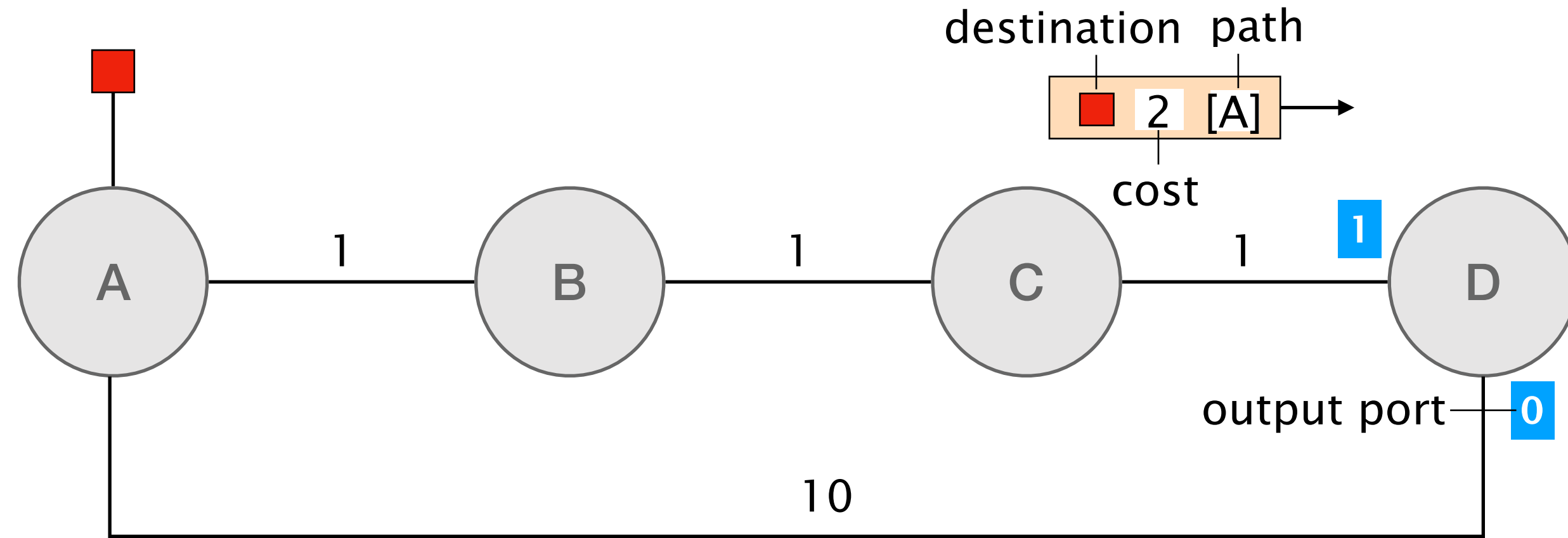




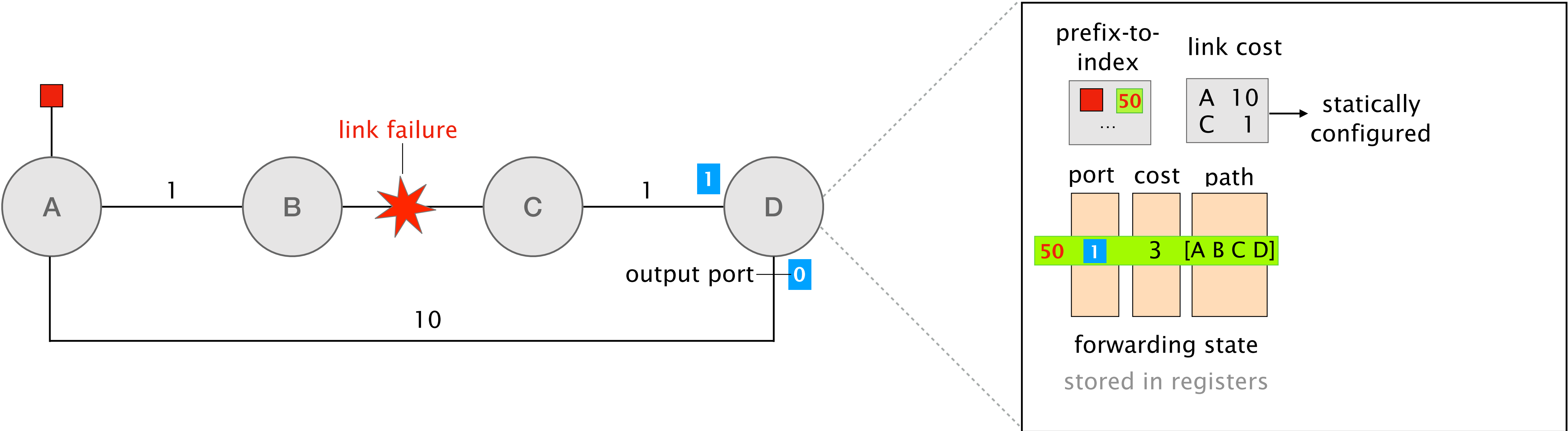
# Switches periodically advertise vectors to neighbors



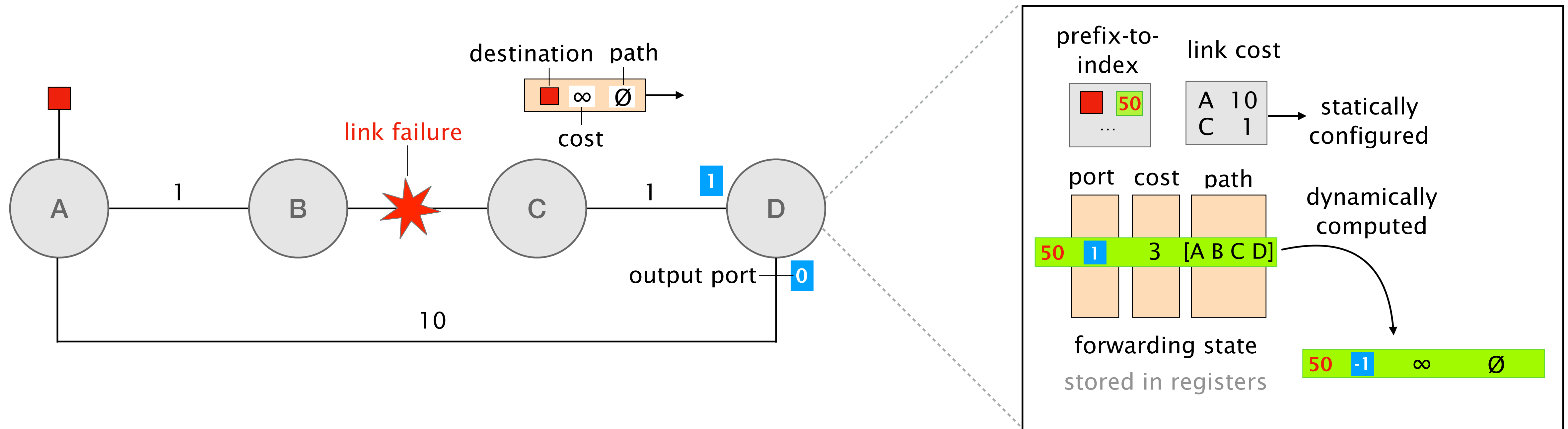
# Switches periodically advertise vectors to neighbors



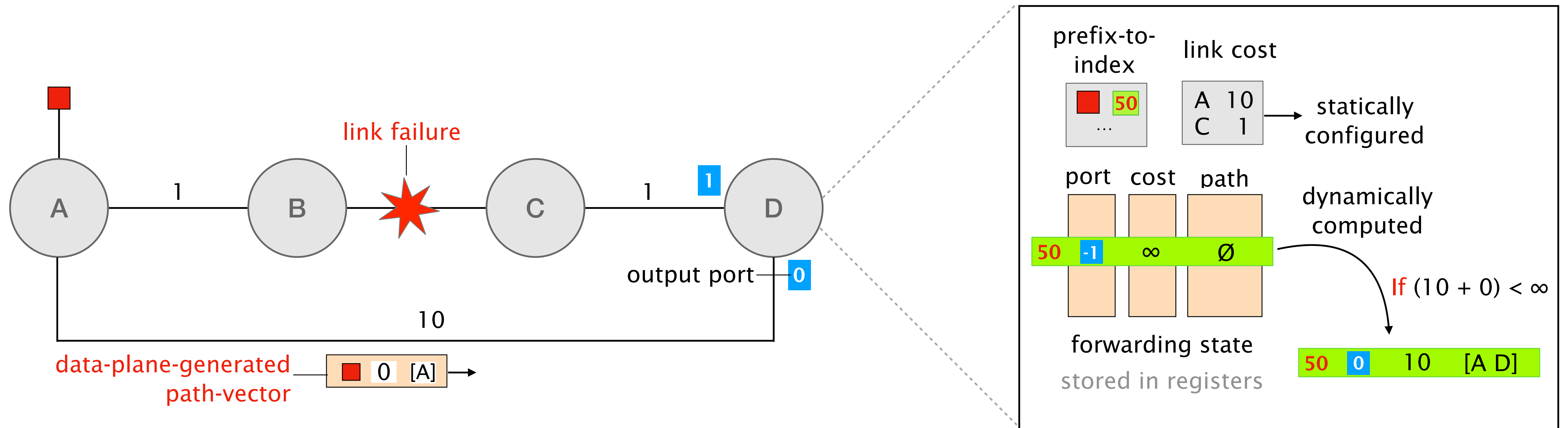
# Computing new forwarding state after a link failure



# Computing new forwarding state after a link failure



# Computing new forwarding state after a link failure



Does it actually work?

Does it actually work?

Yes!

# Hardware-Accelerated P4 prototype

## Implementation

Implemented in P4<sub>16</sub>

Compiled it to bmv2

2000 lines of P4 code

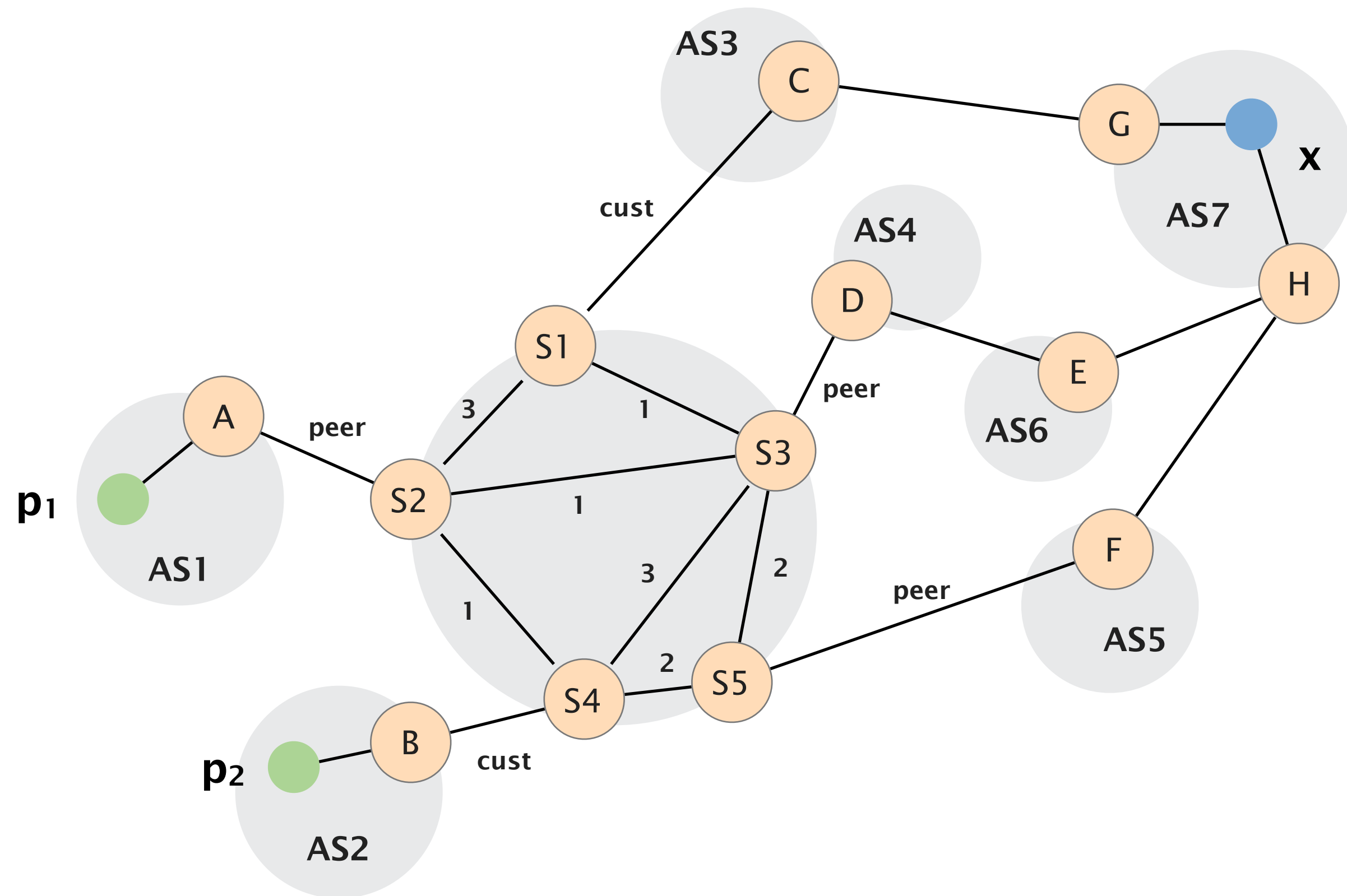
## Capabilities

▶ **Intra**-domain destinations  
path-vector routing

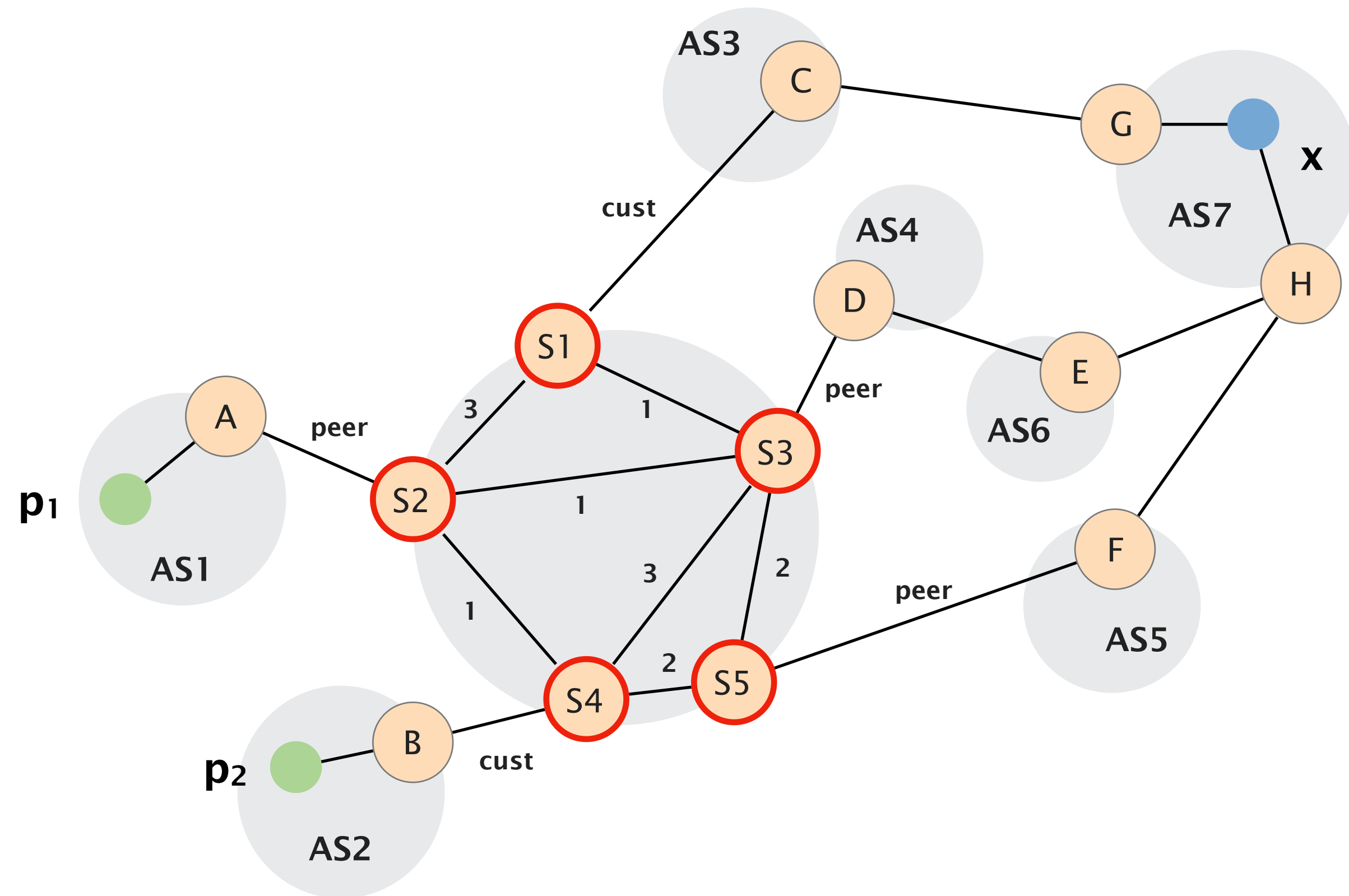
▶ **Inter**-domain destinations  
BGP-like route selection



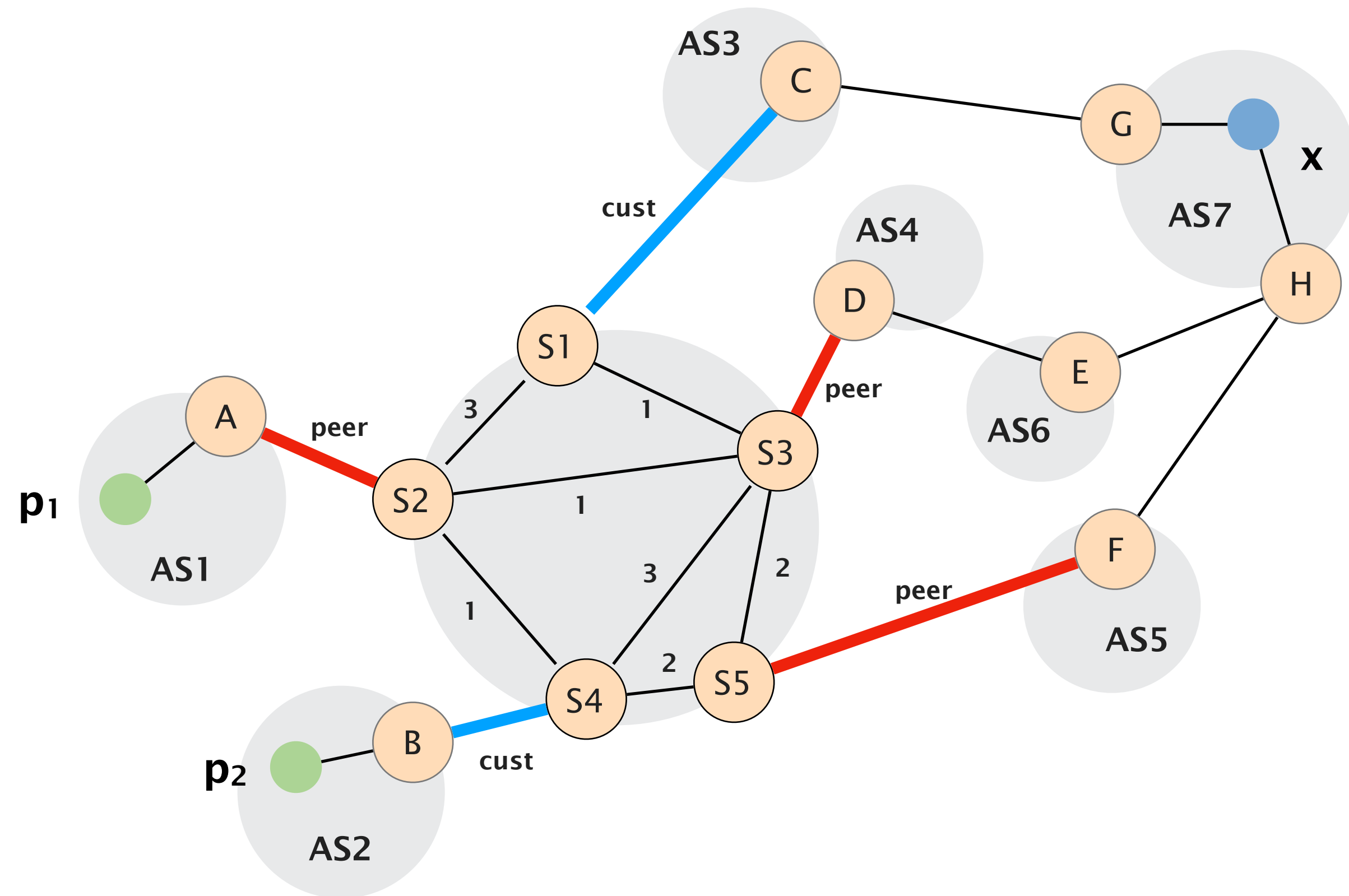
# We tested our implementation in a real case study



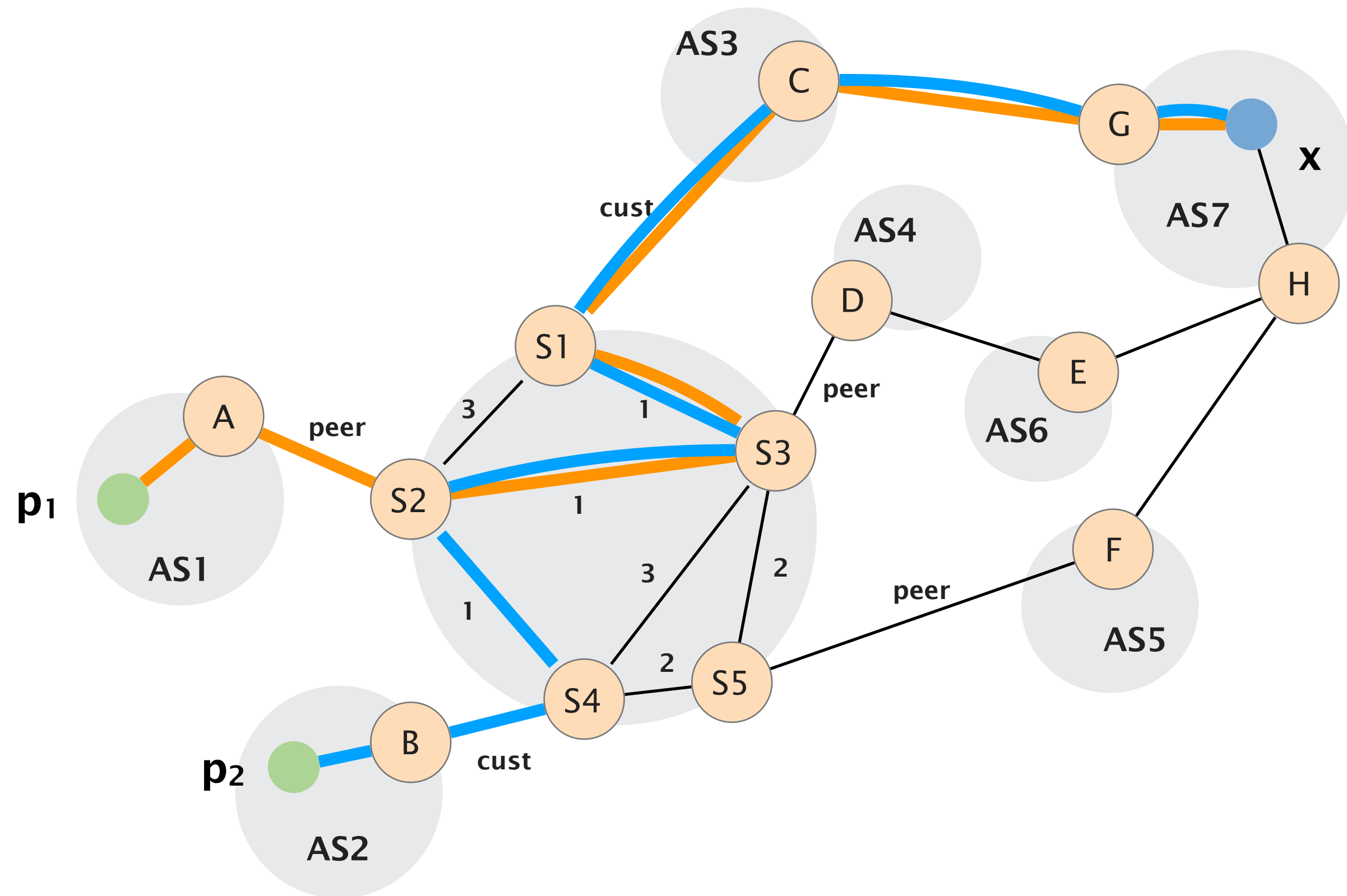
# Only the internal switches run the hardware-based control plane



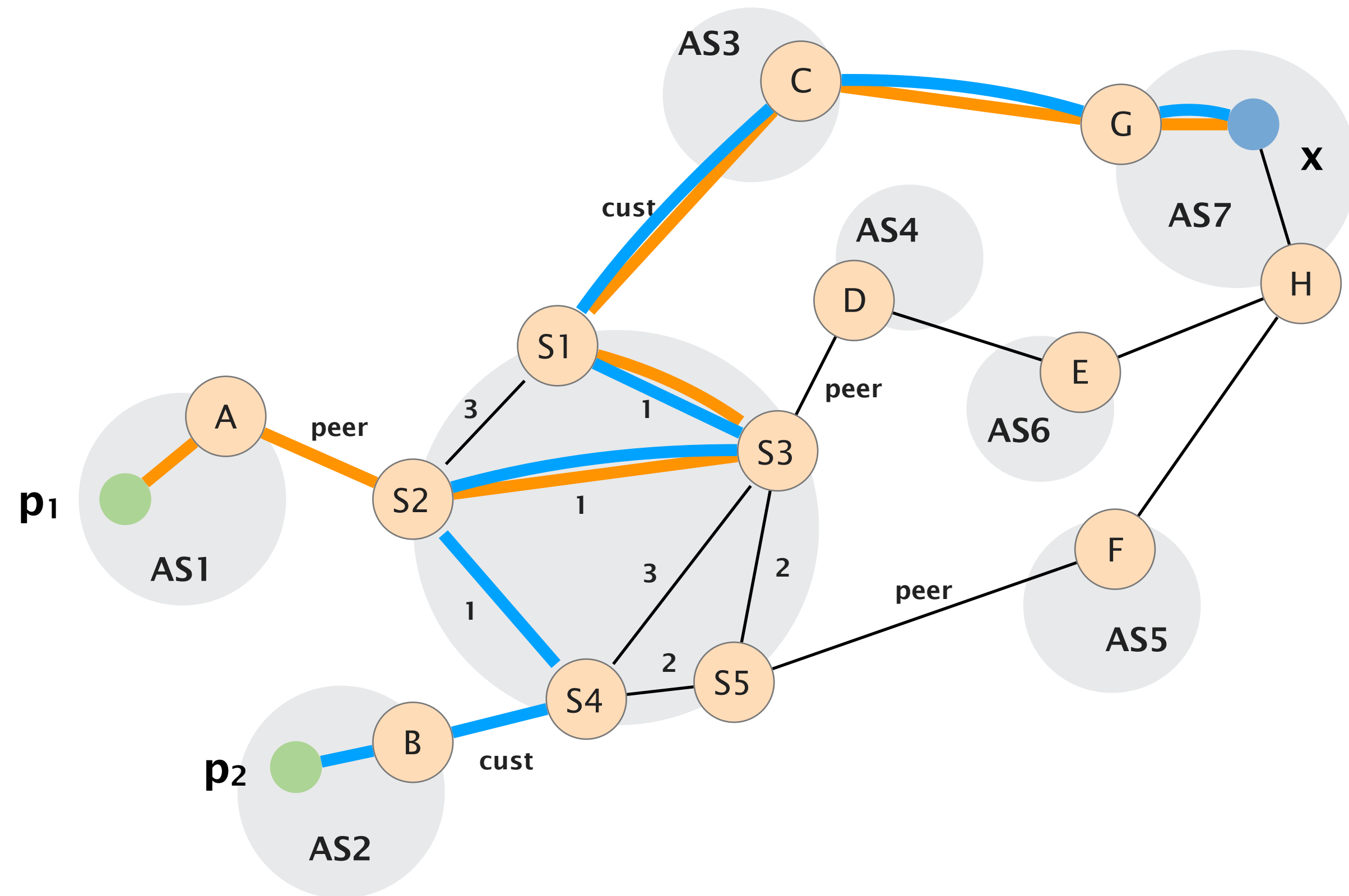
Each switch is connected to an external  
**peer** or **customer**



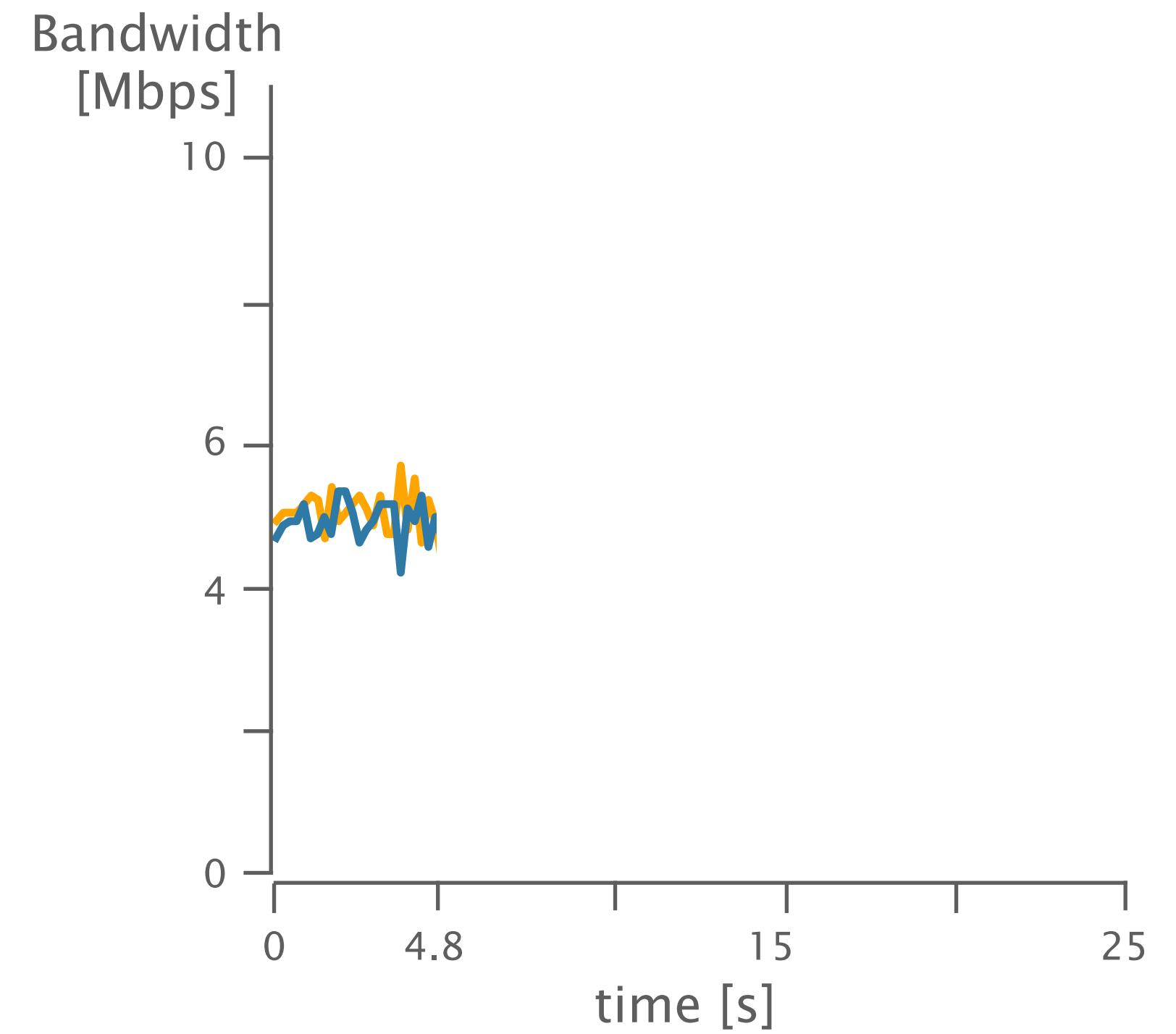
We generate two TCP flows from **AS1** and **AS2**



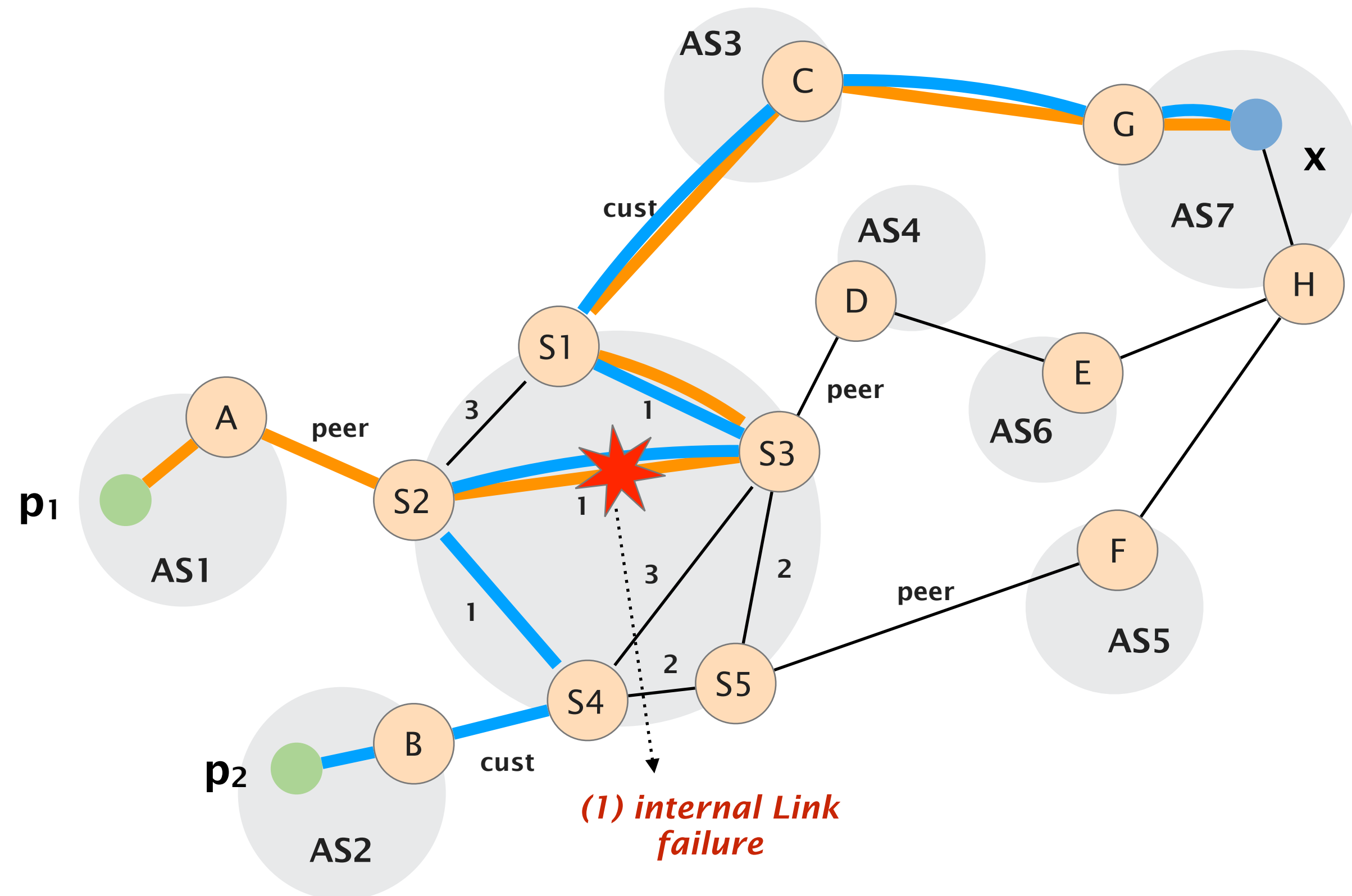
# Monitor traffic before the failure



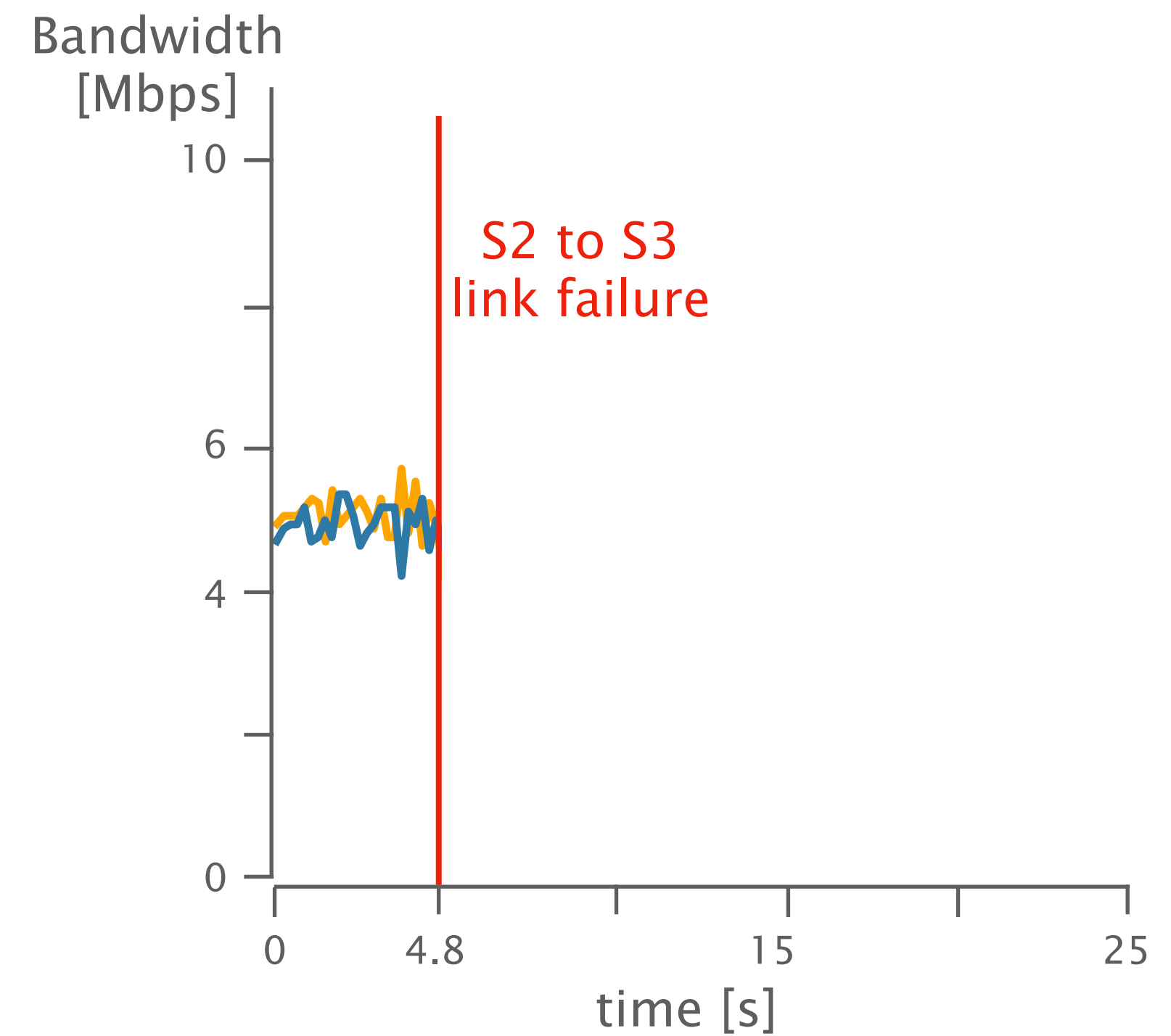
## Traffic S1- AS3



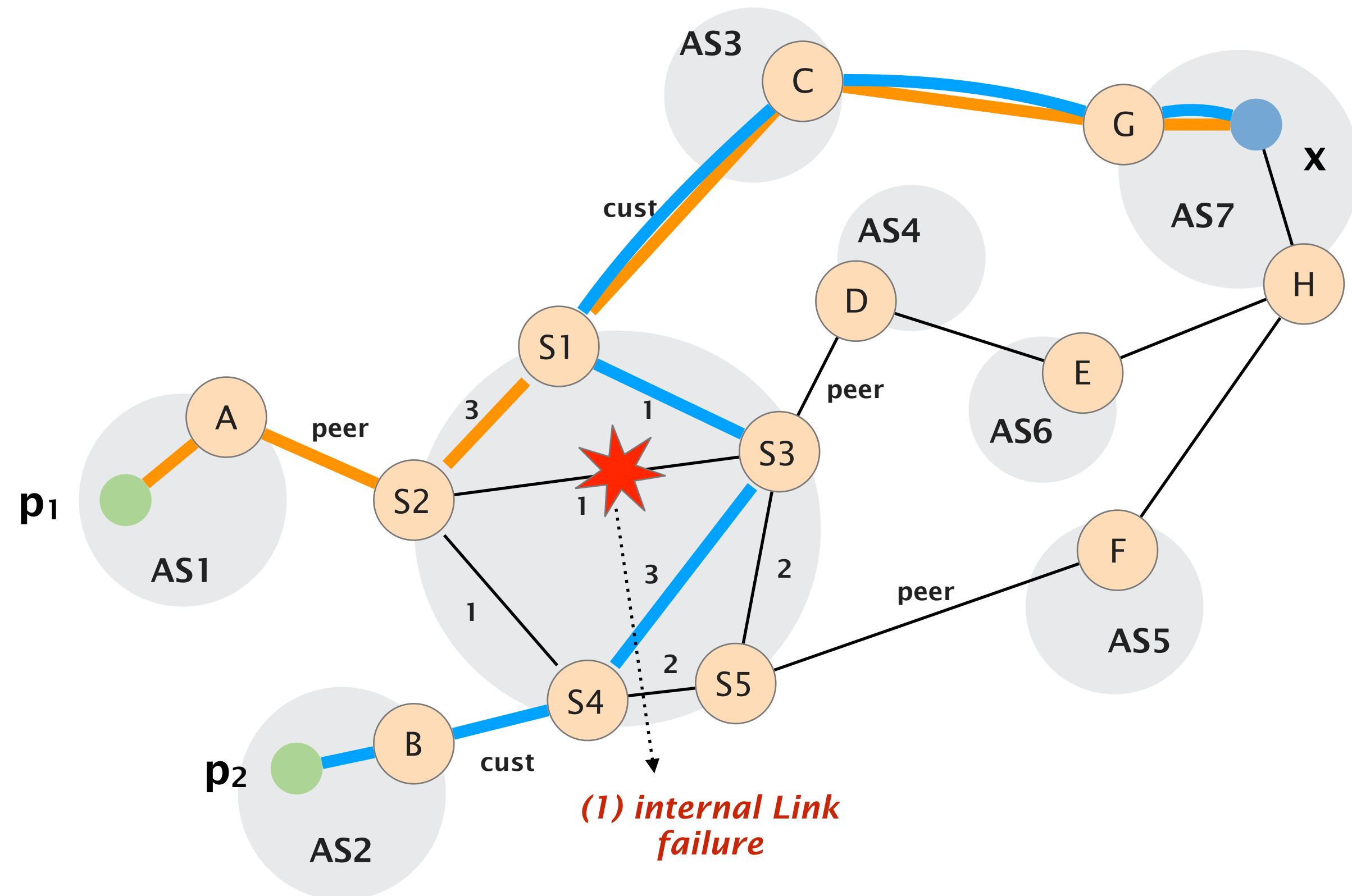
# Internal link fails and triggers the path-vector algorithm



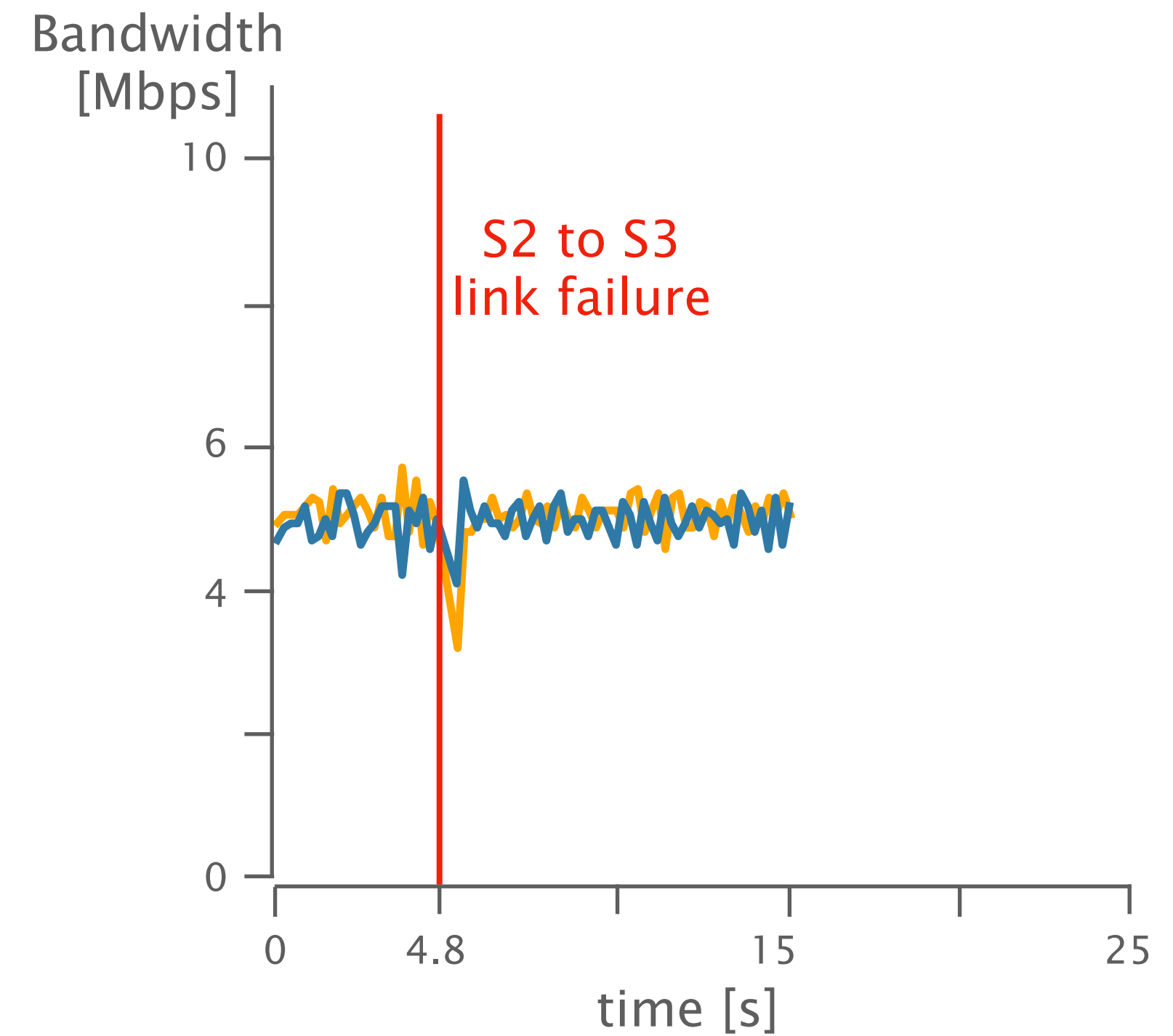
## Traffic S1- AS3



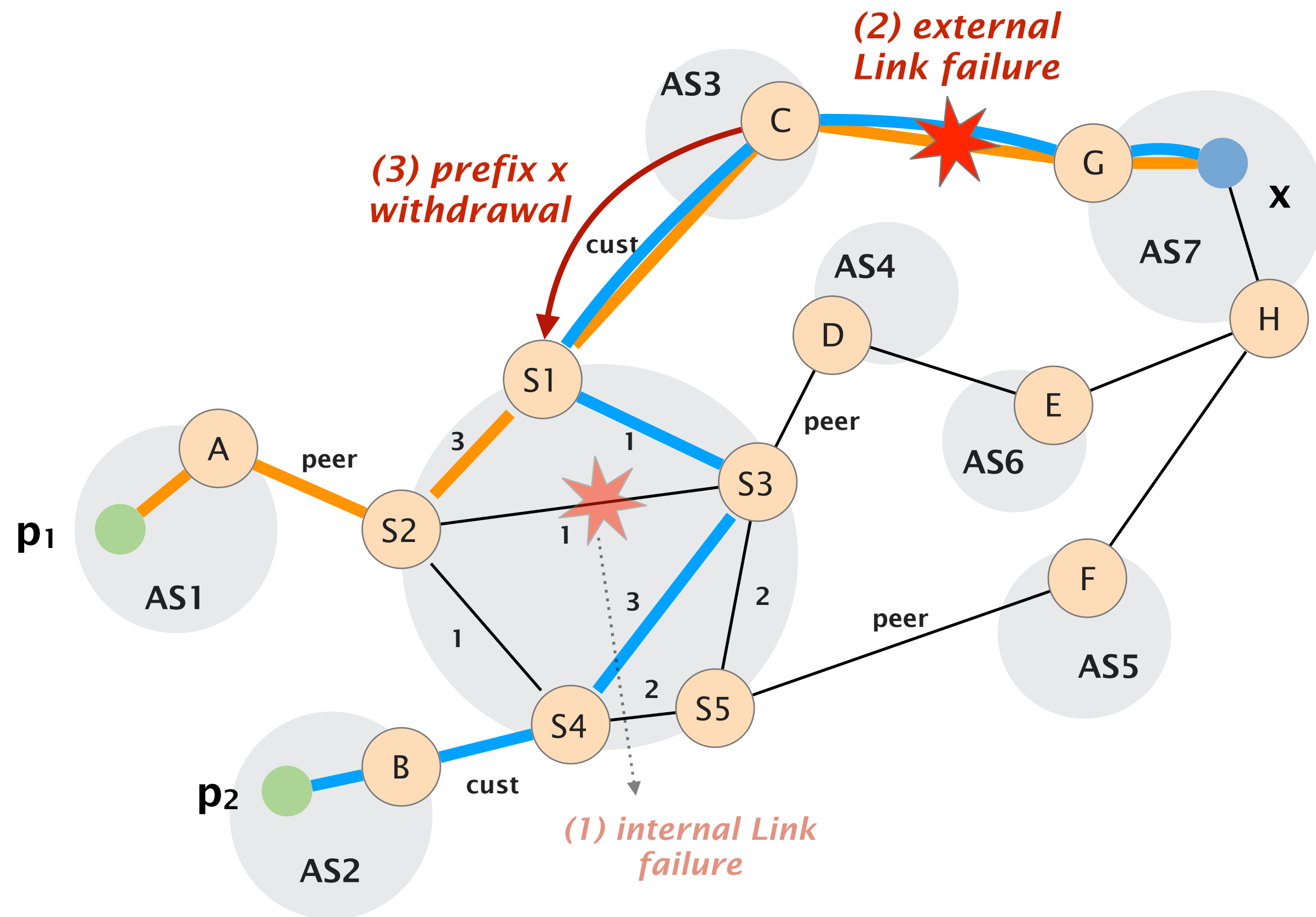
# Internal link fails and triggers the path-vector algorithm



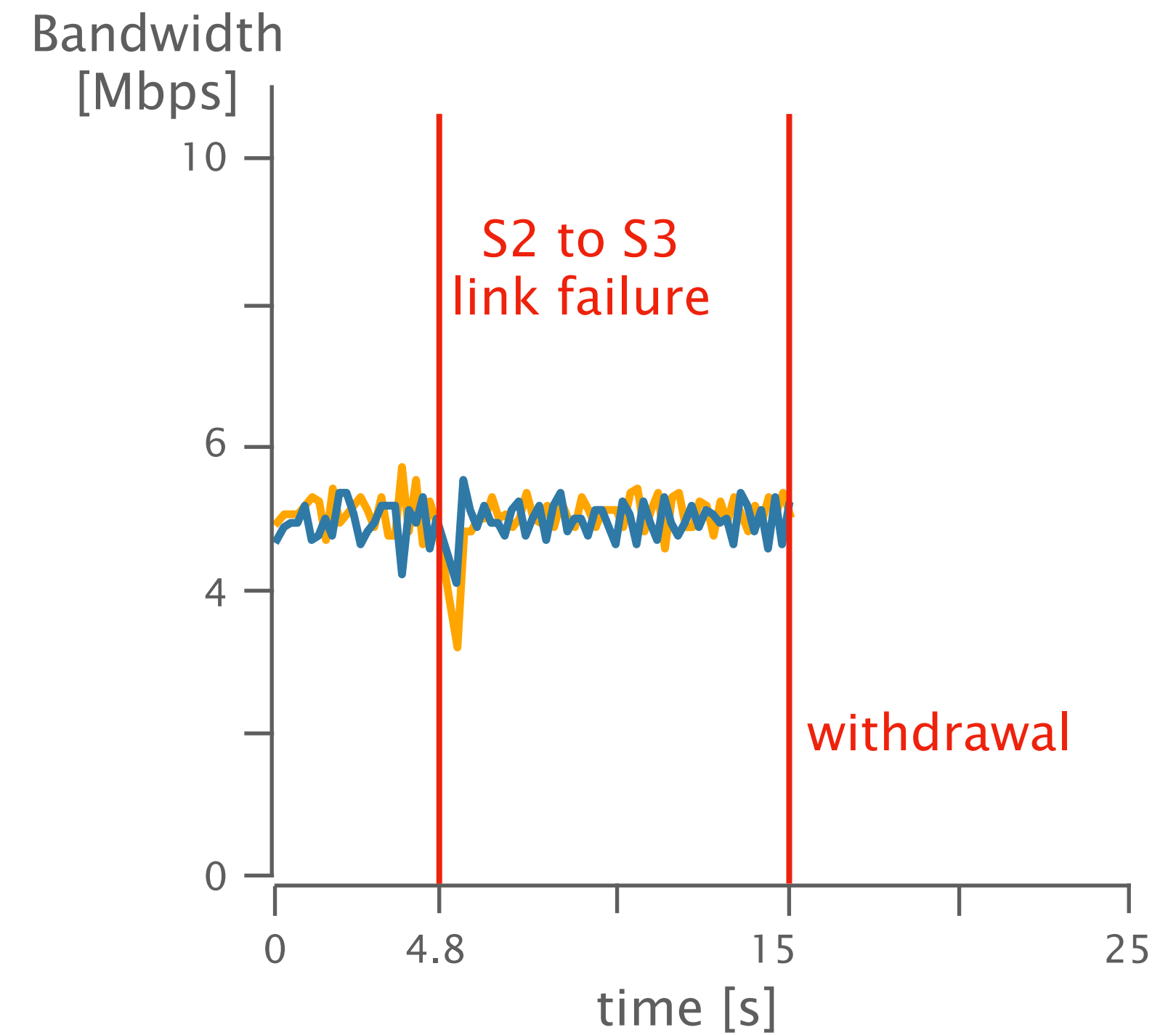
## Traffic S1- AS3



# External link failure triggers a prefix withdrawal

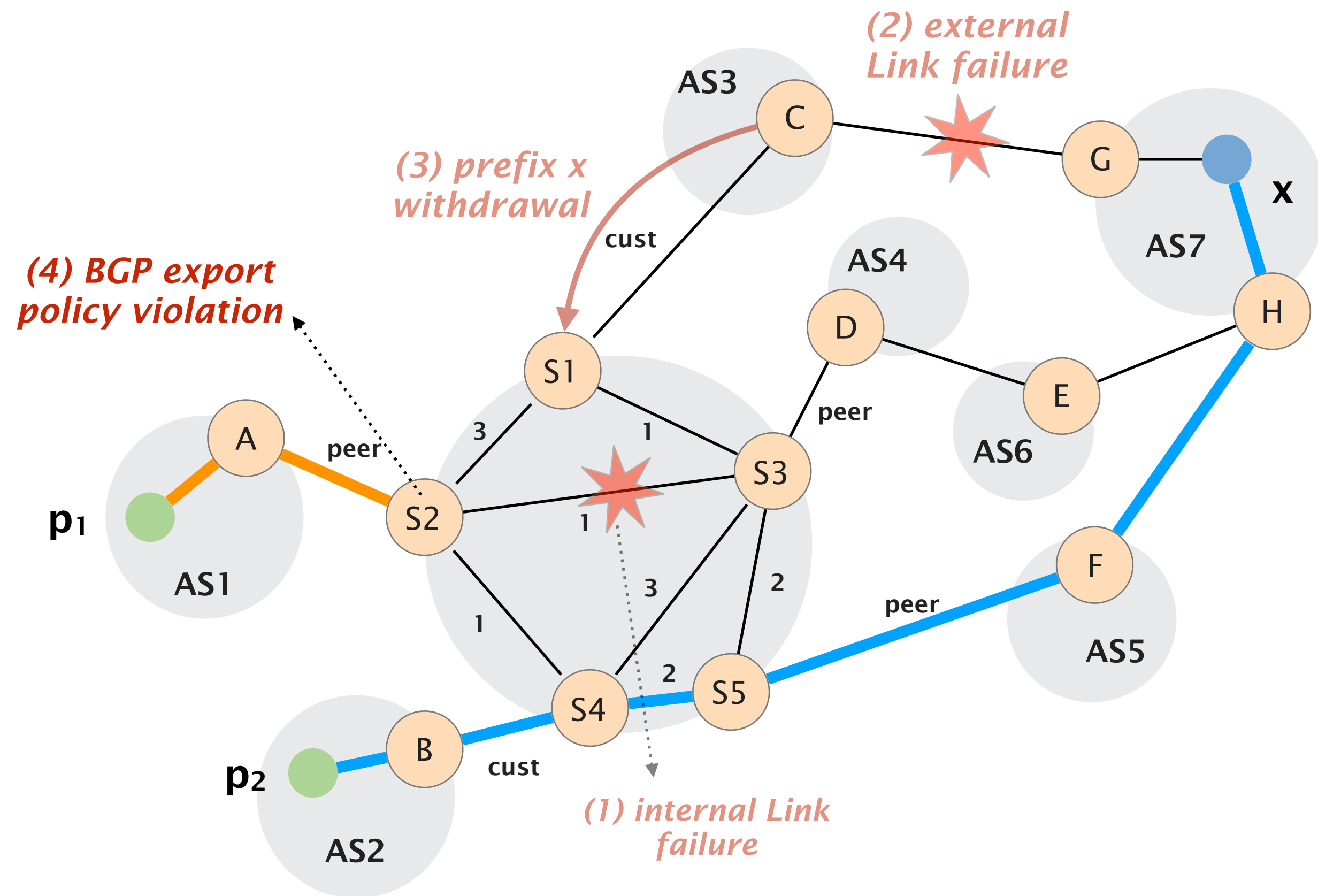


## Traffic S1- AS3

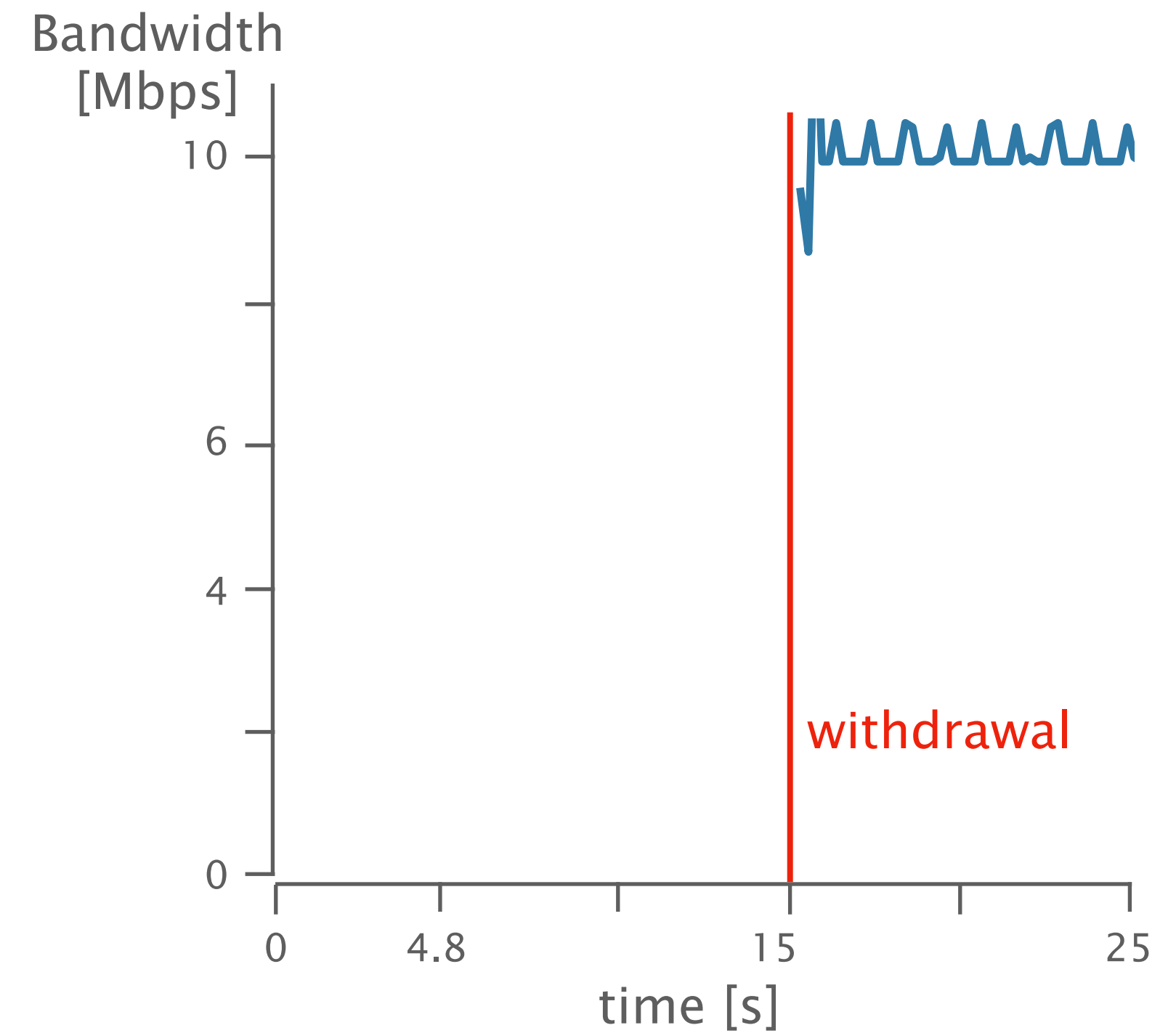


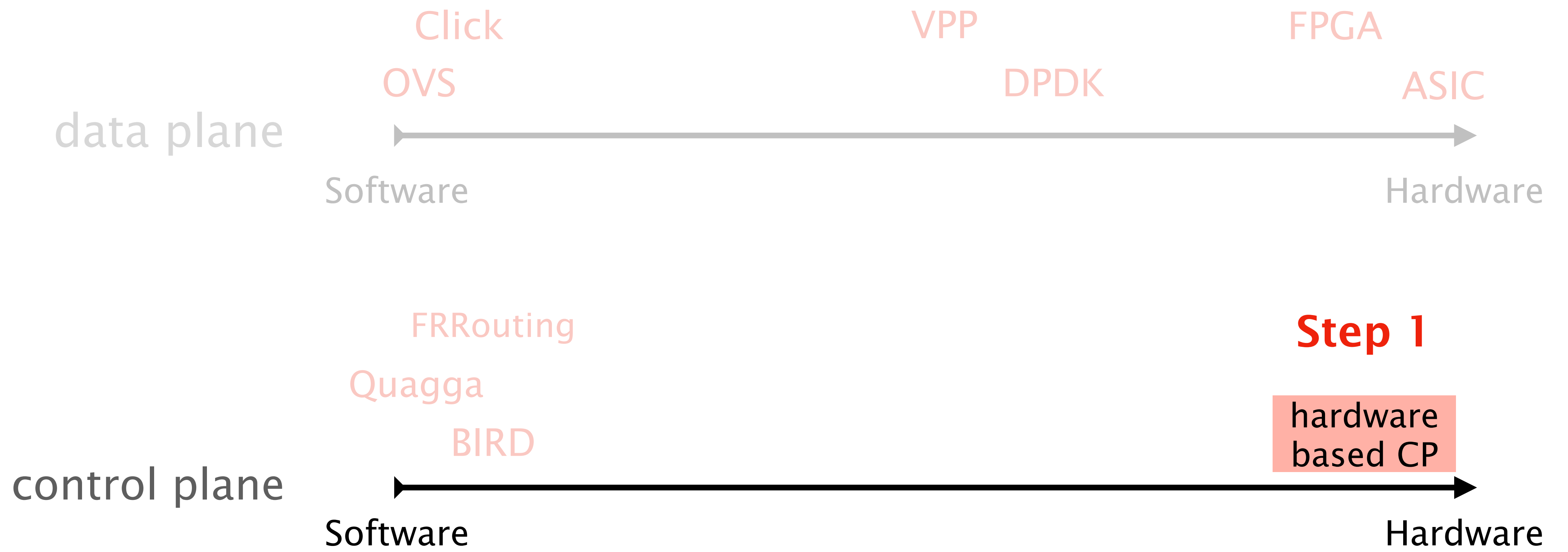


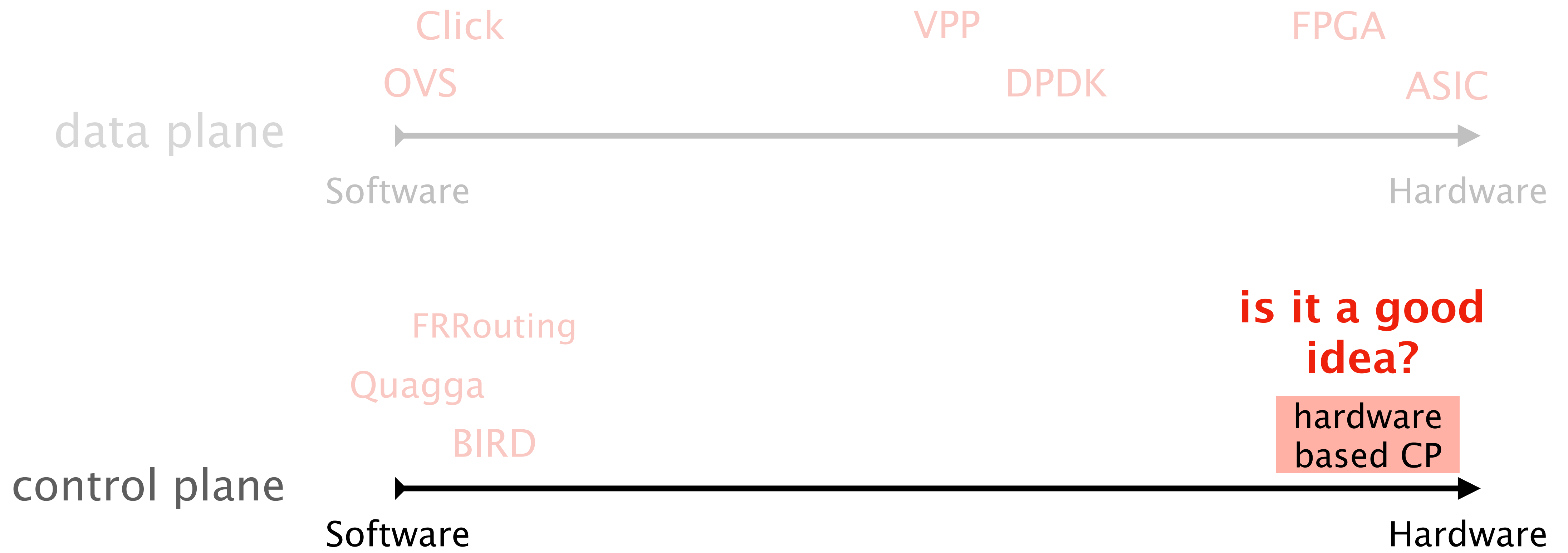
# Network computes new egress and applies new policies



## Traffic S5- AS5







Programmable hardware is great but...  
not **limitless**

Programmable hardware is great but...  
not **limitless**

Some tasks **cannot** be offloaded  
Others might not be even **desirable** !

# Programmable hardware is great but... not **limitless**

Some tasks **cannot** be offloaded

Others might not be even **desirable** !

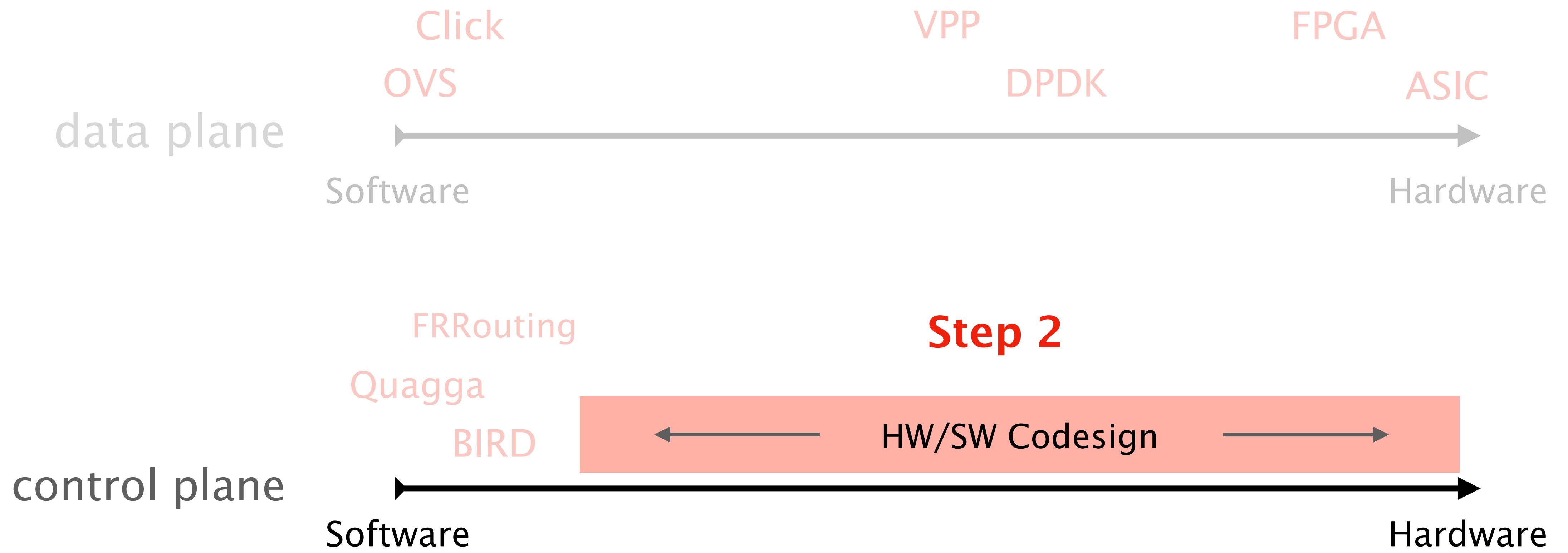
- Reliable protocols  
e.g. TCP would require too many resources !
- Poor scalability of control plane tasks  
hardware memory is scarce and expensive

Can we have the best  
of both worlds?

Can we have the best  
of both worlds?

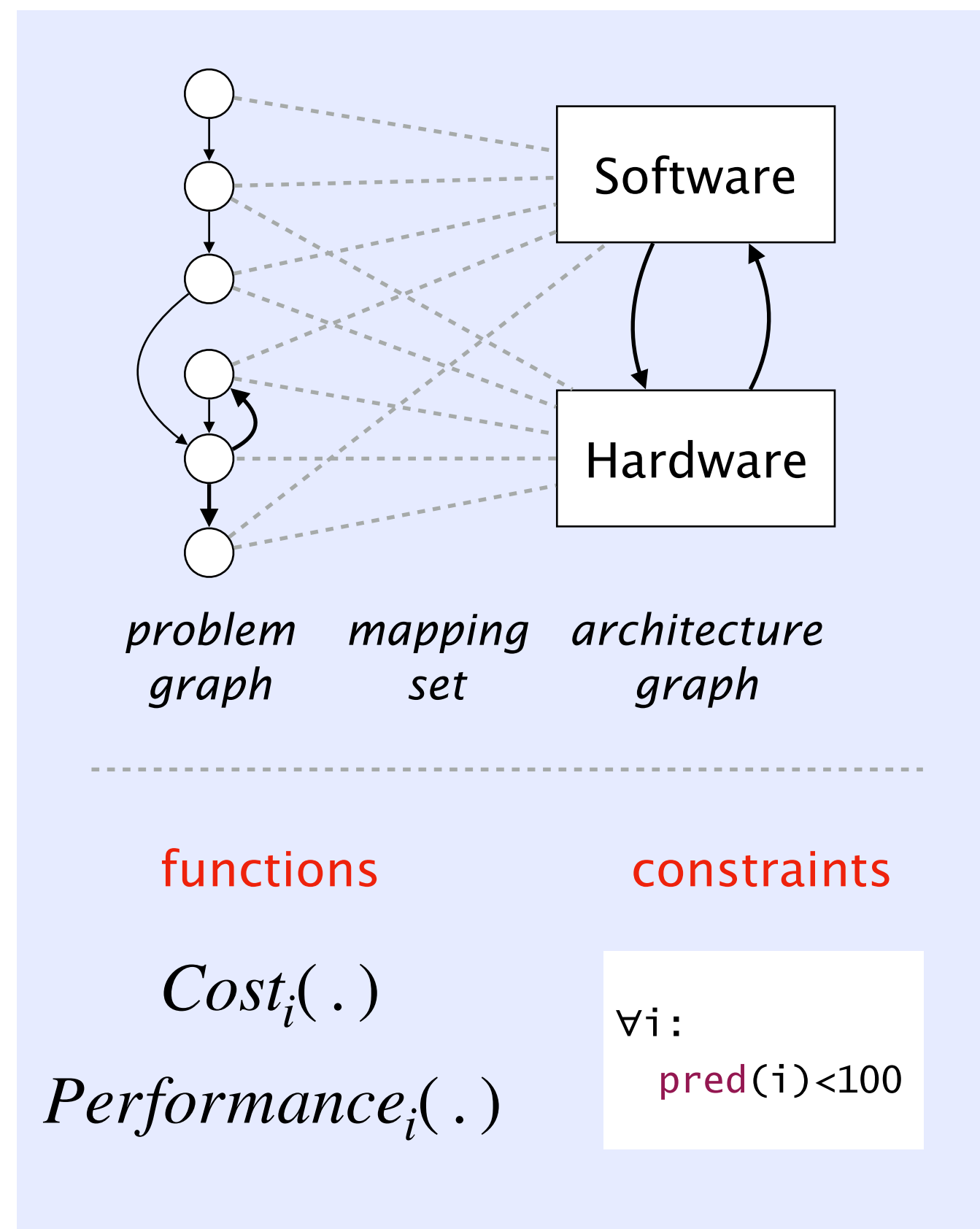
HW/SW  
codesign





# Hardware-software codesign

## Specification

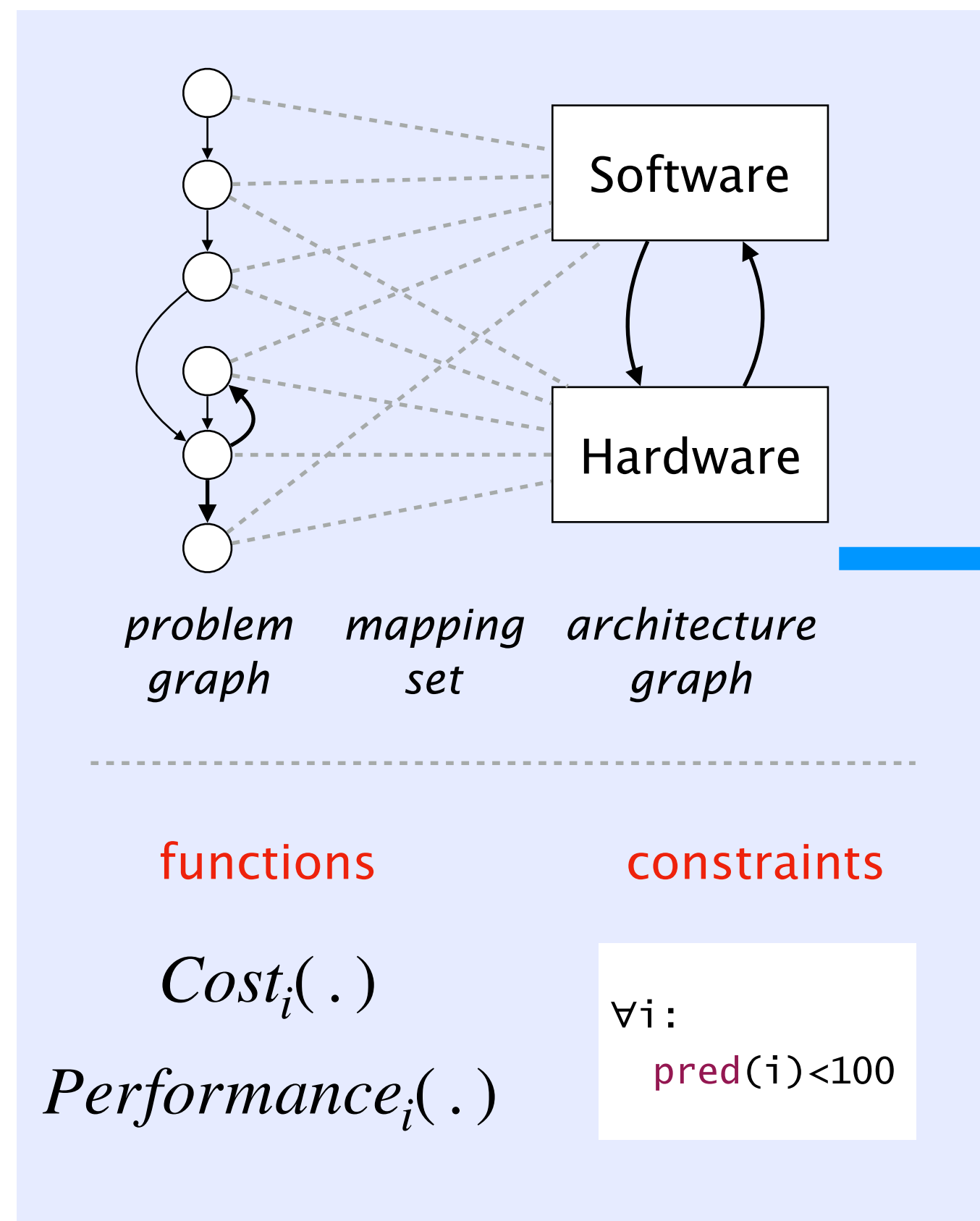


## Optimization

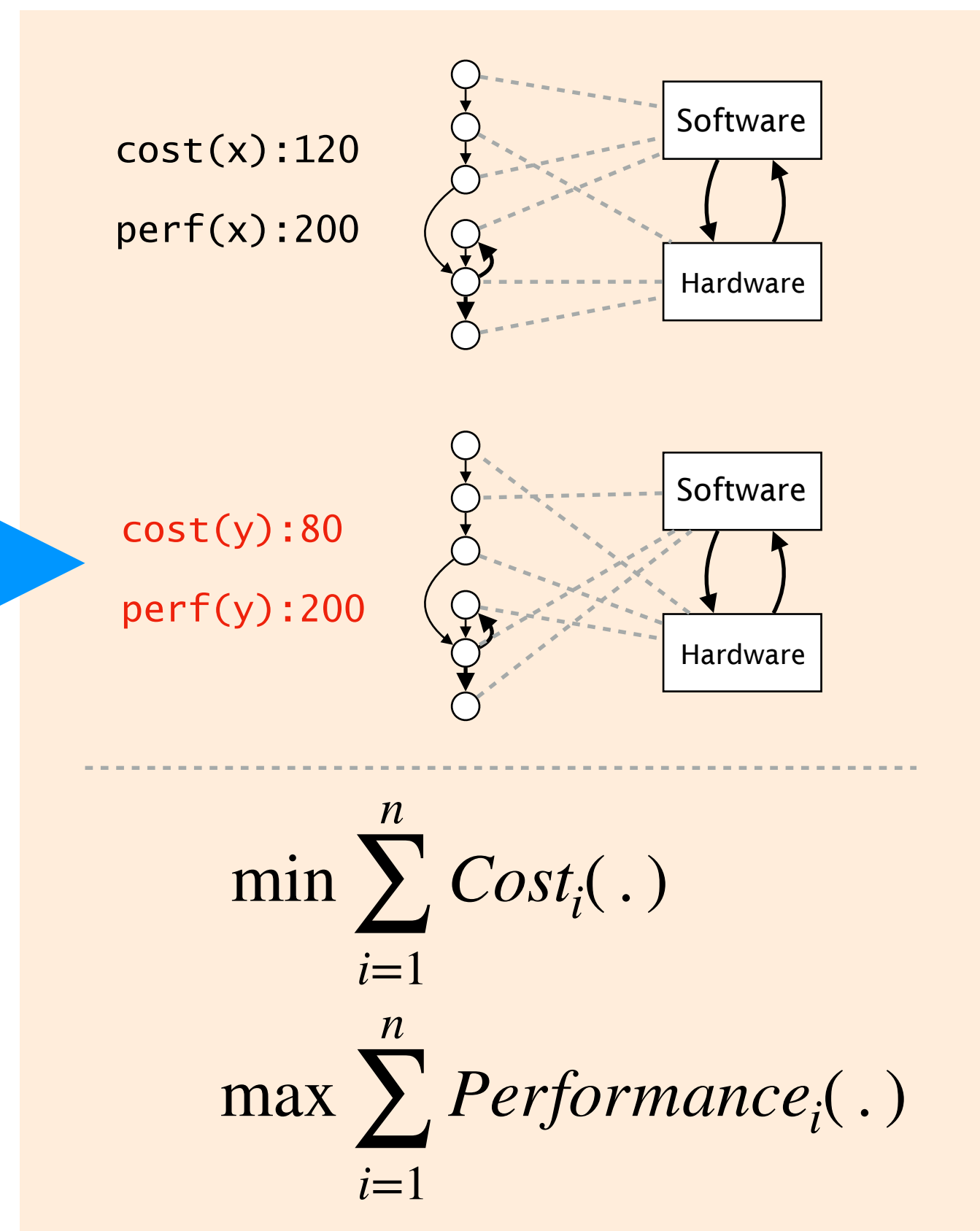
## Synthesis

# Hardware-software codesign

## Specification



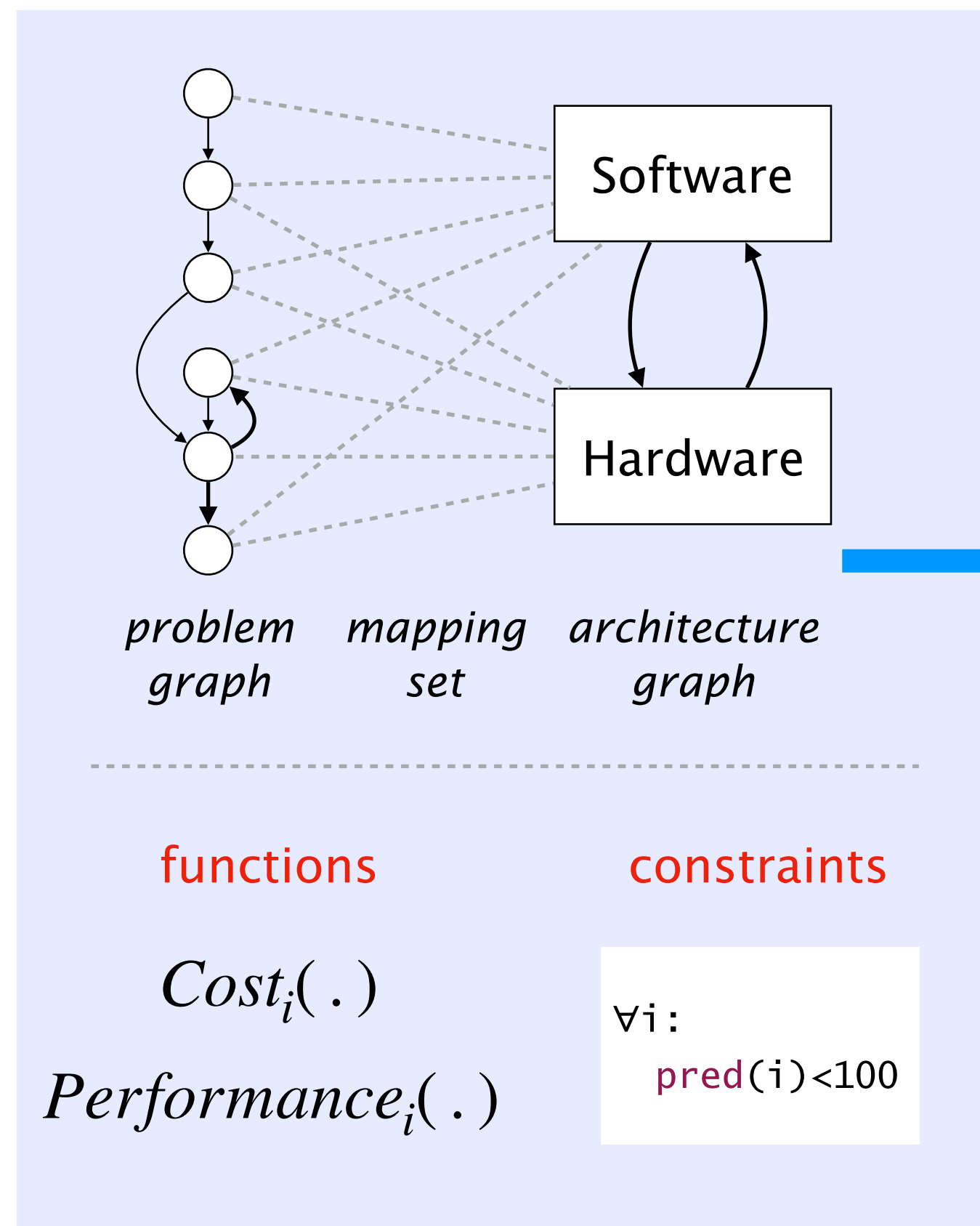
## Optimization



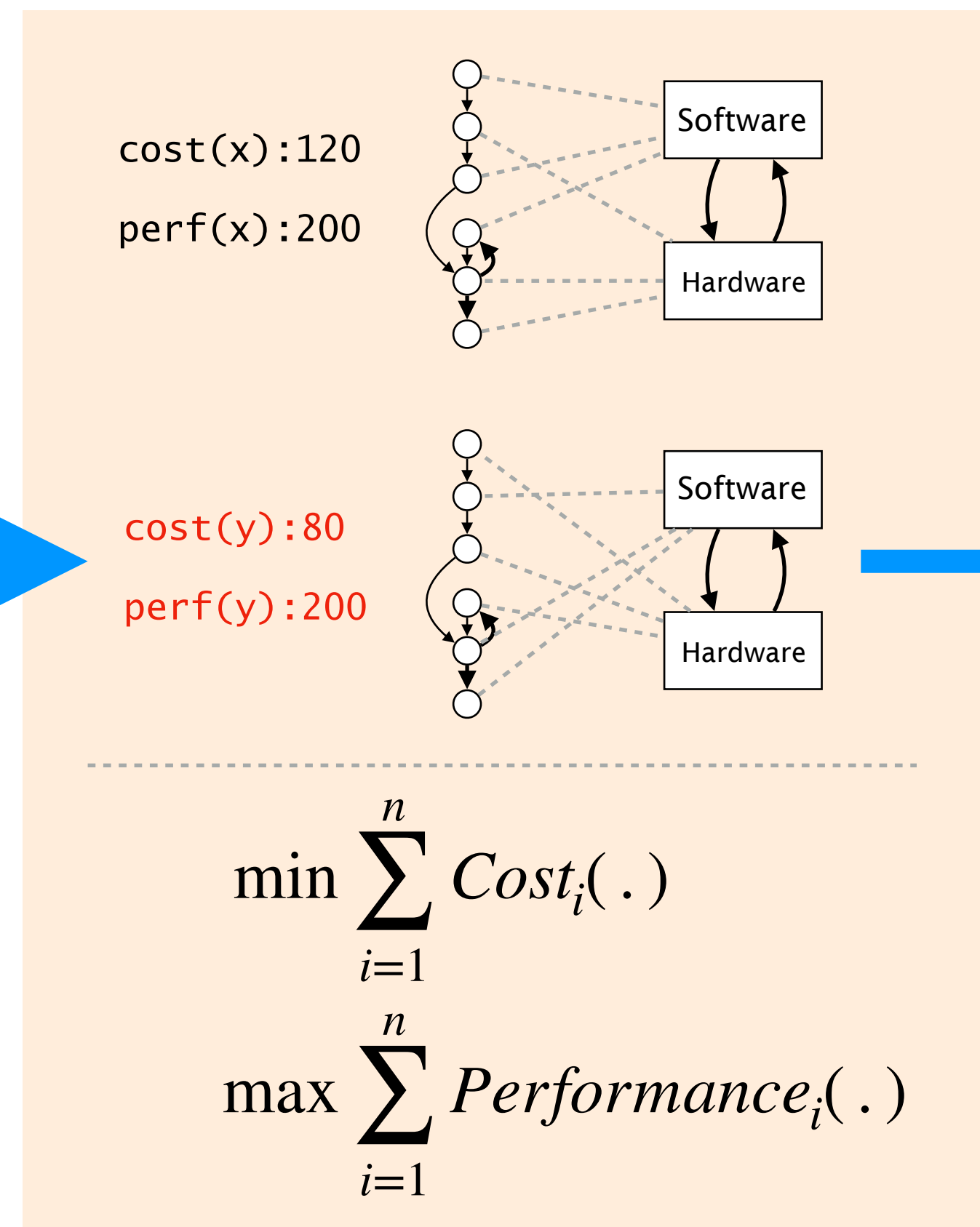
## Synthesis

# Hardware-software codesign

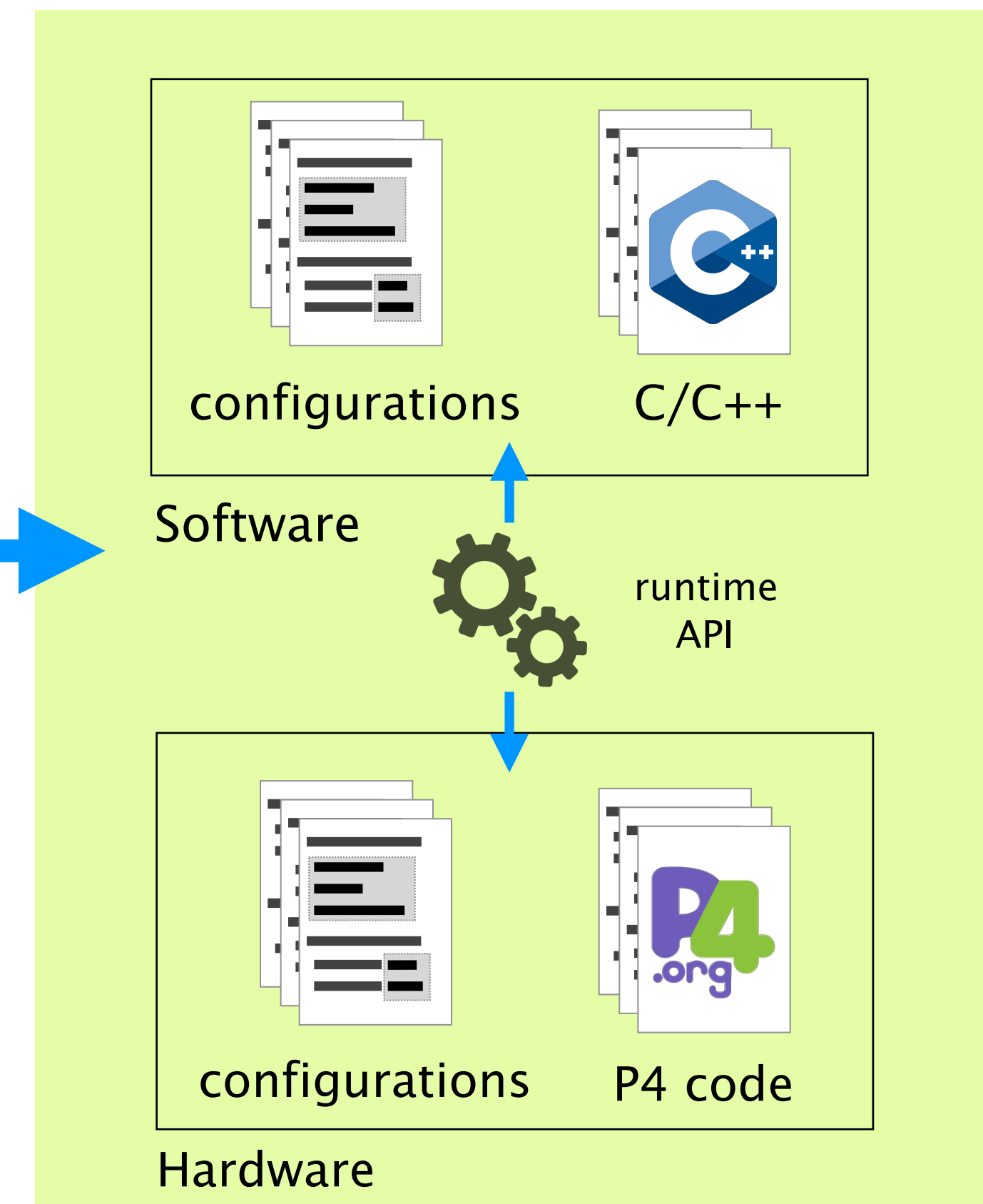
## Specification



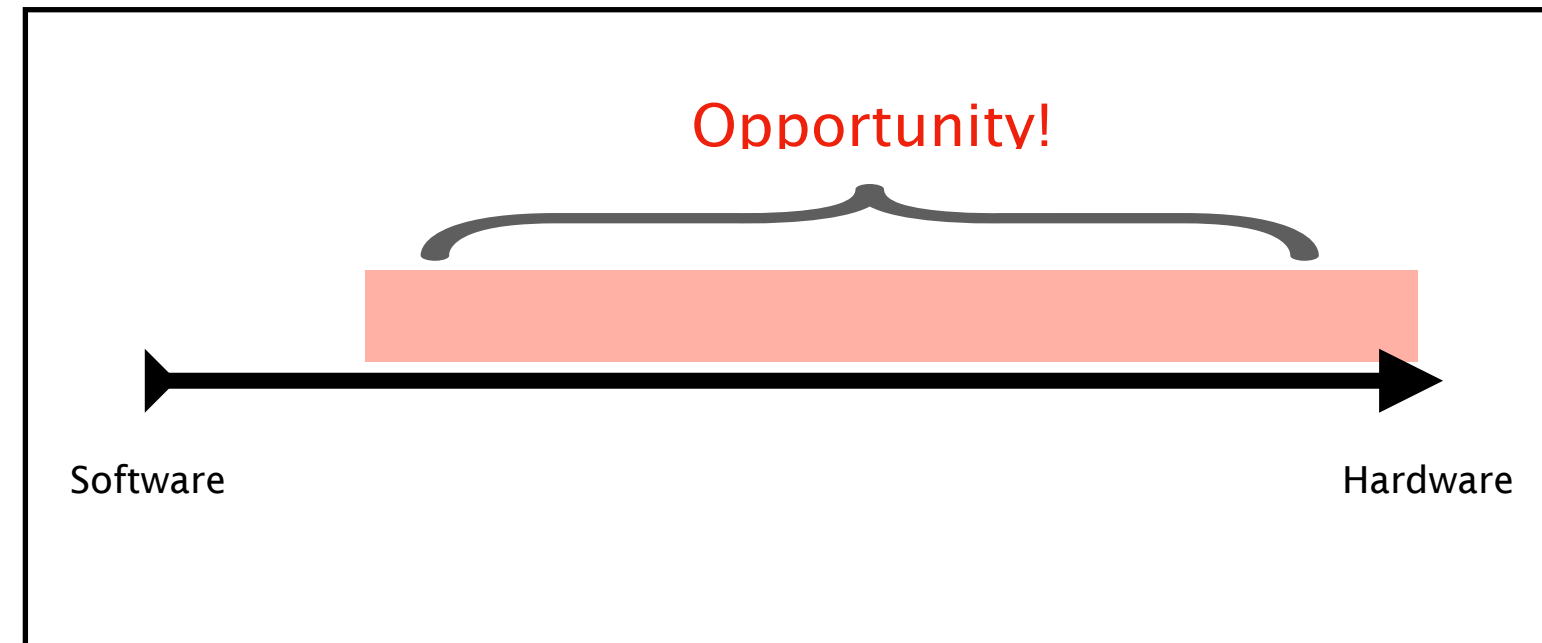
## Optimization



## Synthesis

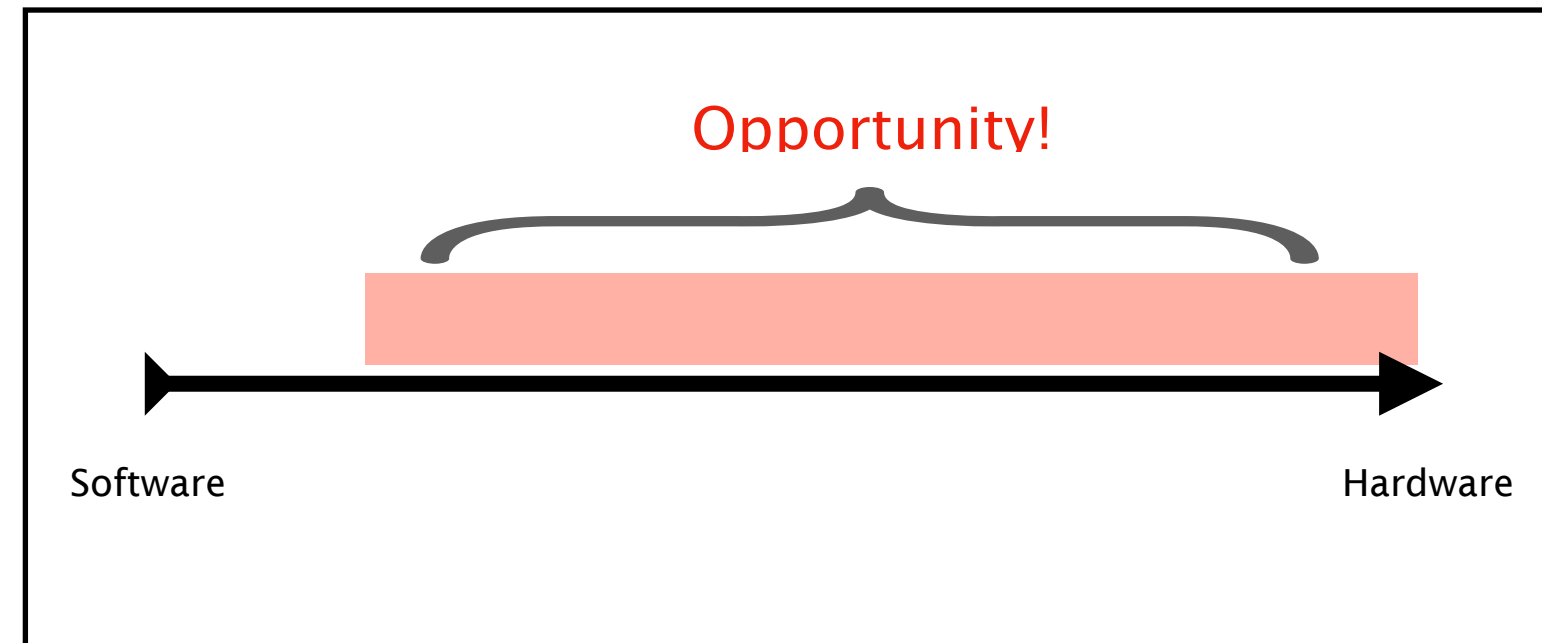


# Summary

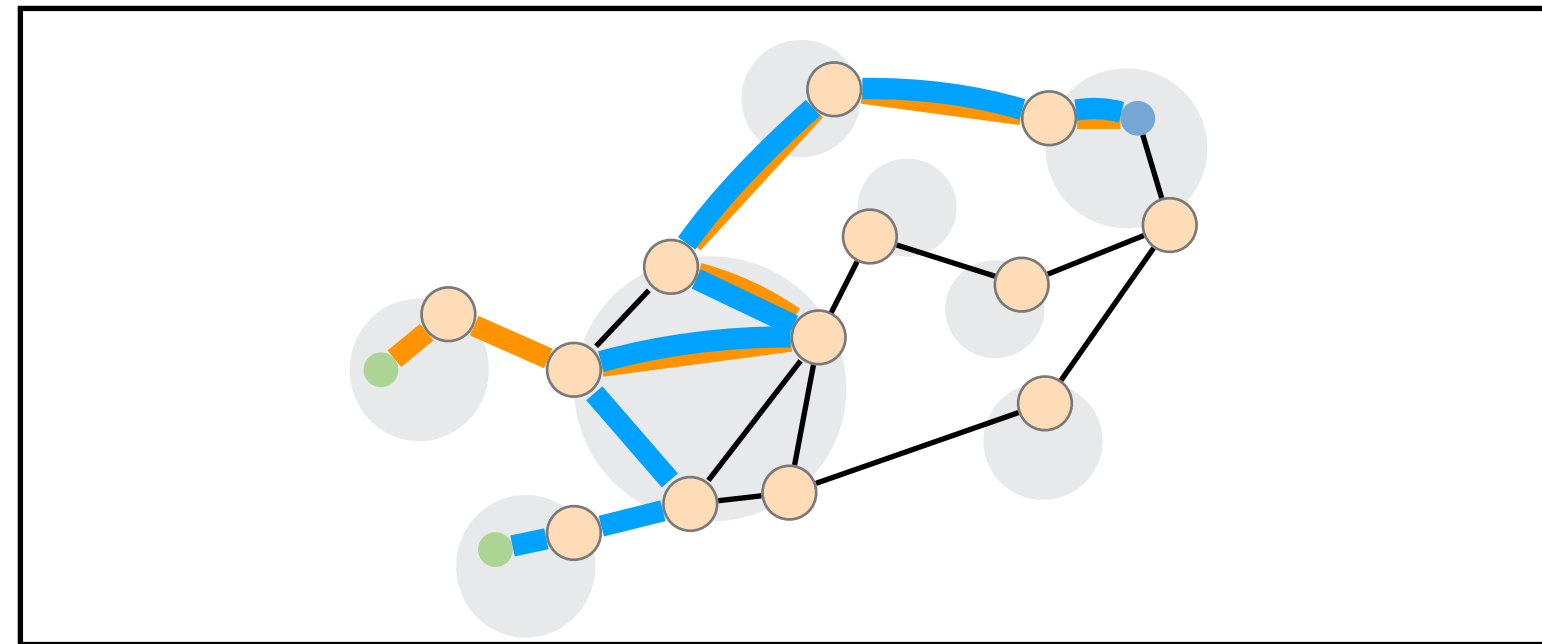


We identified an unexploited opportunity

# Summary

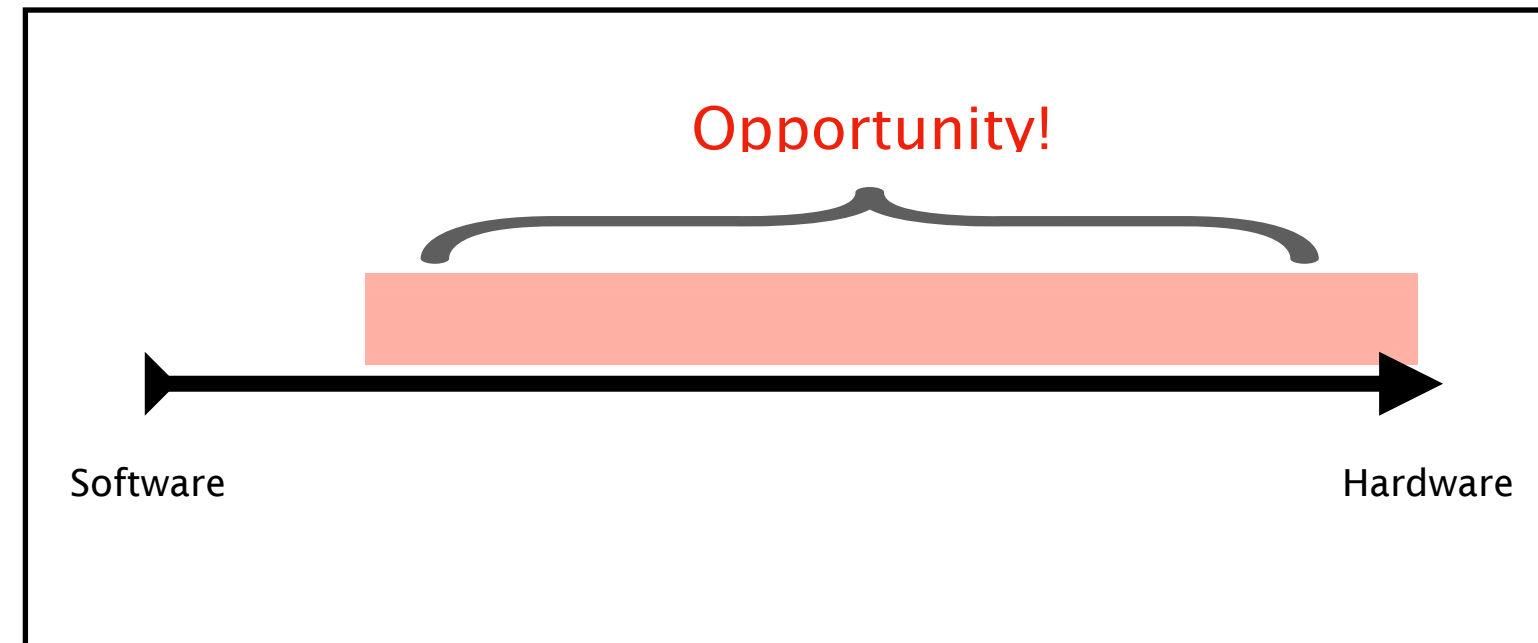


We identified an unexploited opportunity

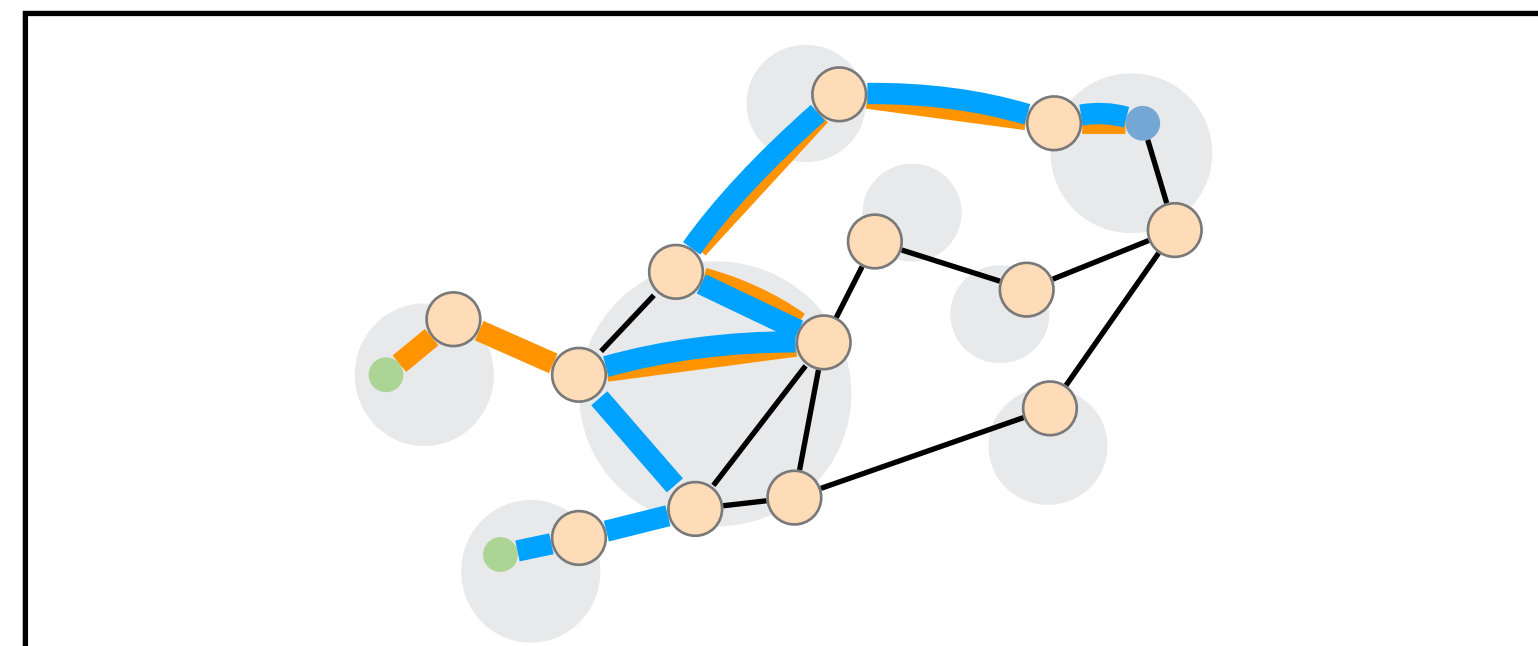


We showed that programmable data planes can run control plane tasks

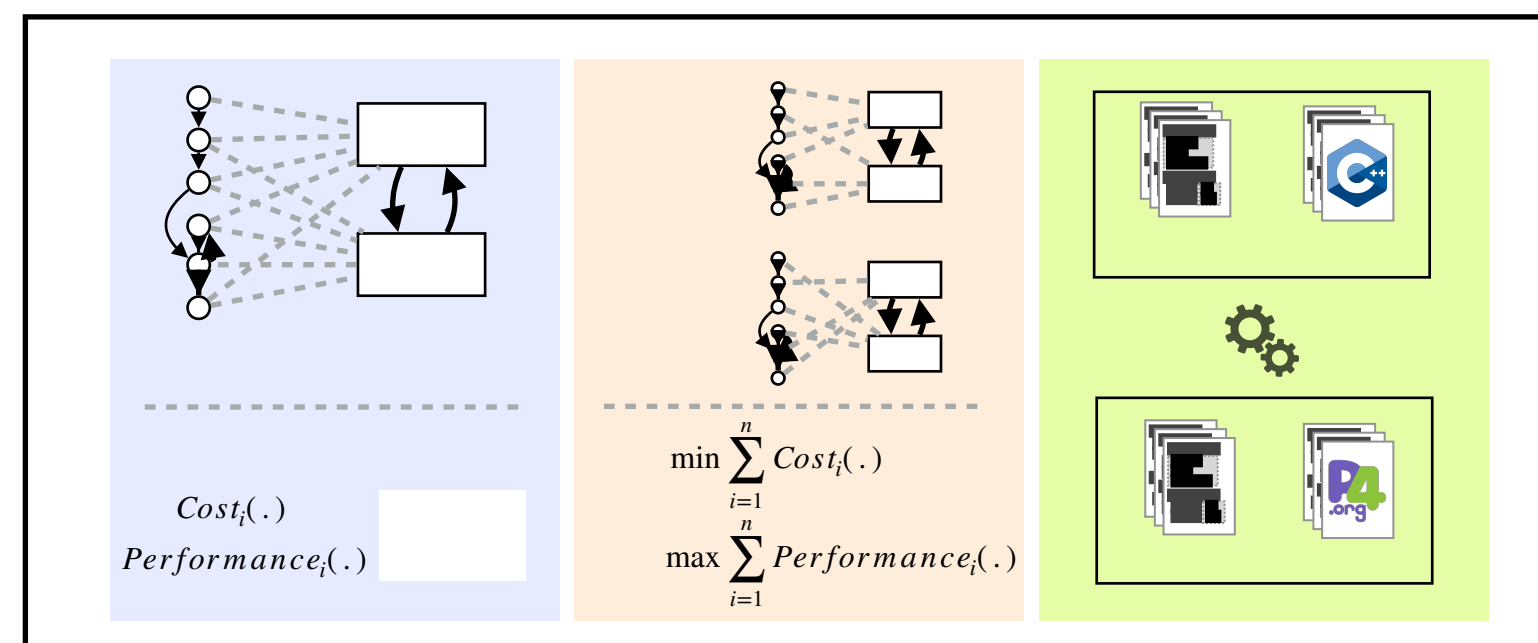
# Summary



We identified an unexploited opportunity



We showed that programmable data planes can run control plane tasks



We plan on leveraging HW/SW codesign to explore design tradeoffs

