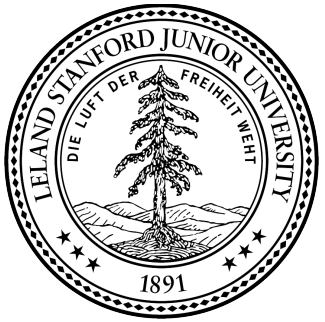
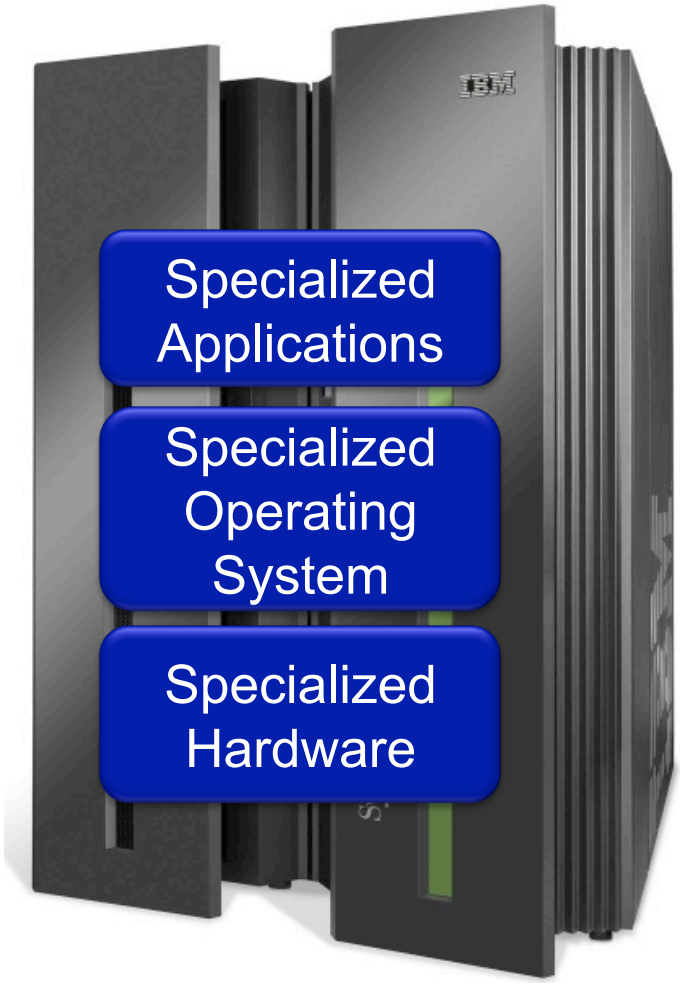
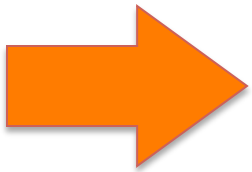


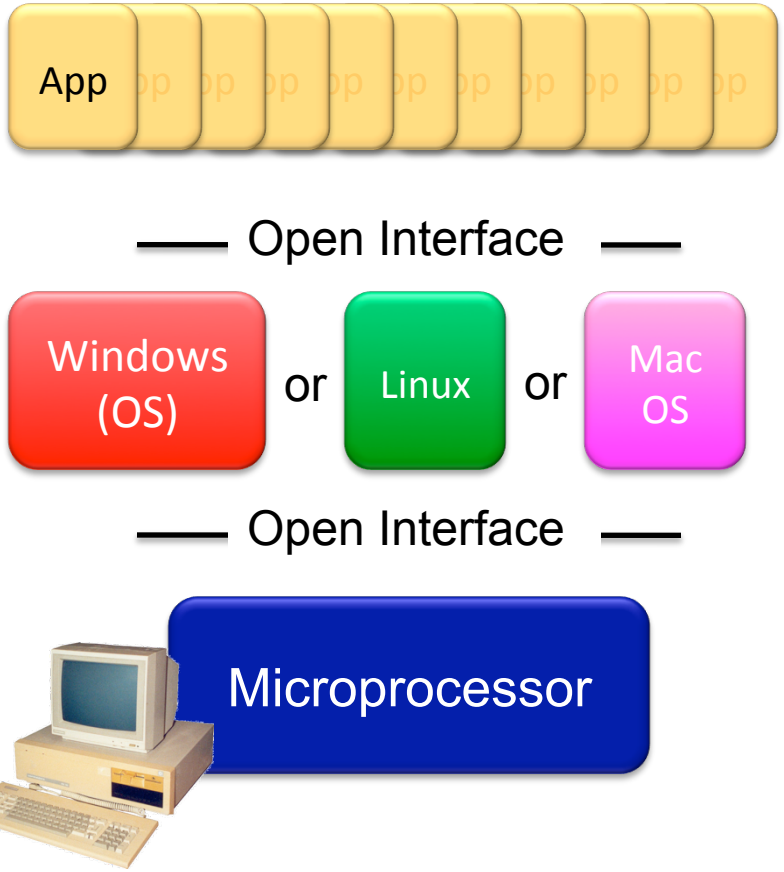
Forwarding Plane Correctness

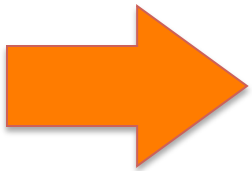
Nick McKeown
Stanford University



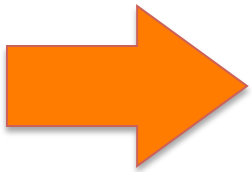
Vertically integrated
Closed, proprietary
Slow innovation
Small industry

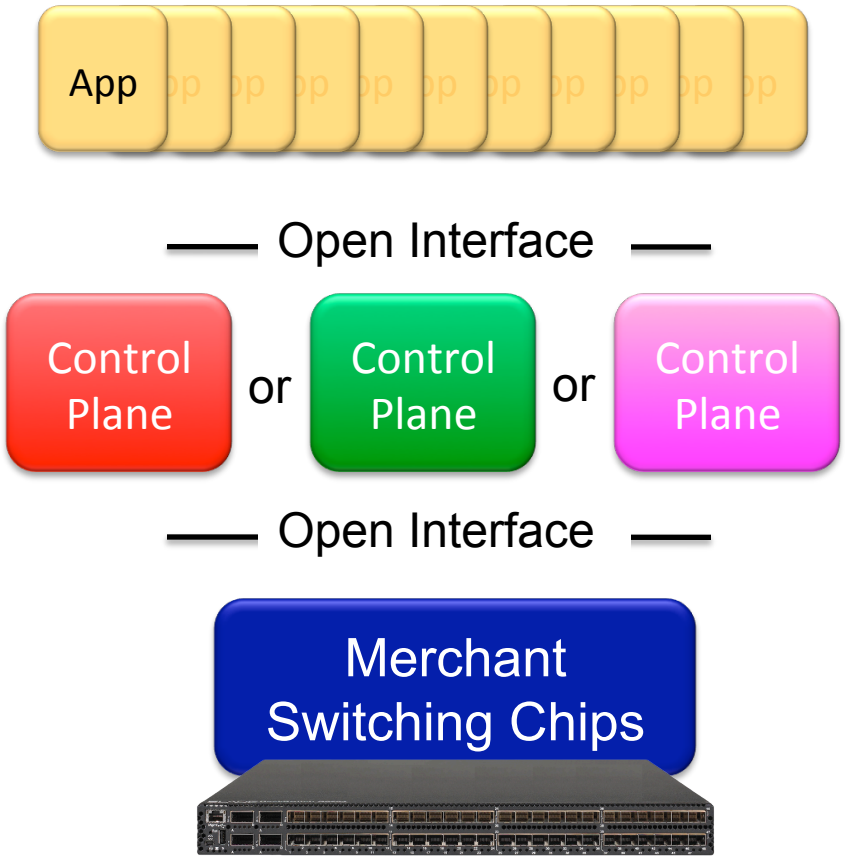




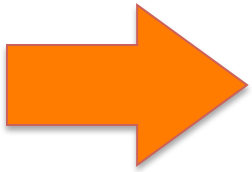


Horizontal
Open interfaces
Rapid innovation
Huge industry





Vertically integrated
Closed, proprietary
Slow innovation



Horizontal
Open interfaces
Rapid innovation

What is SDN?

(when we clear away all the hype)

A network in which the control plane is physically separate from the forwarding plane.

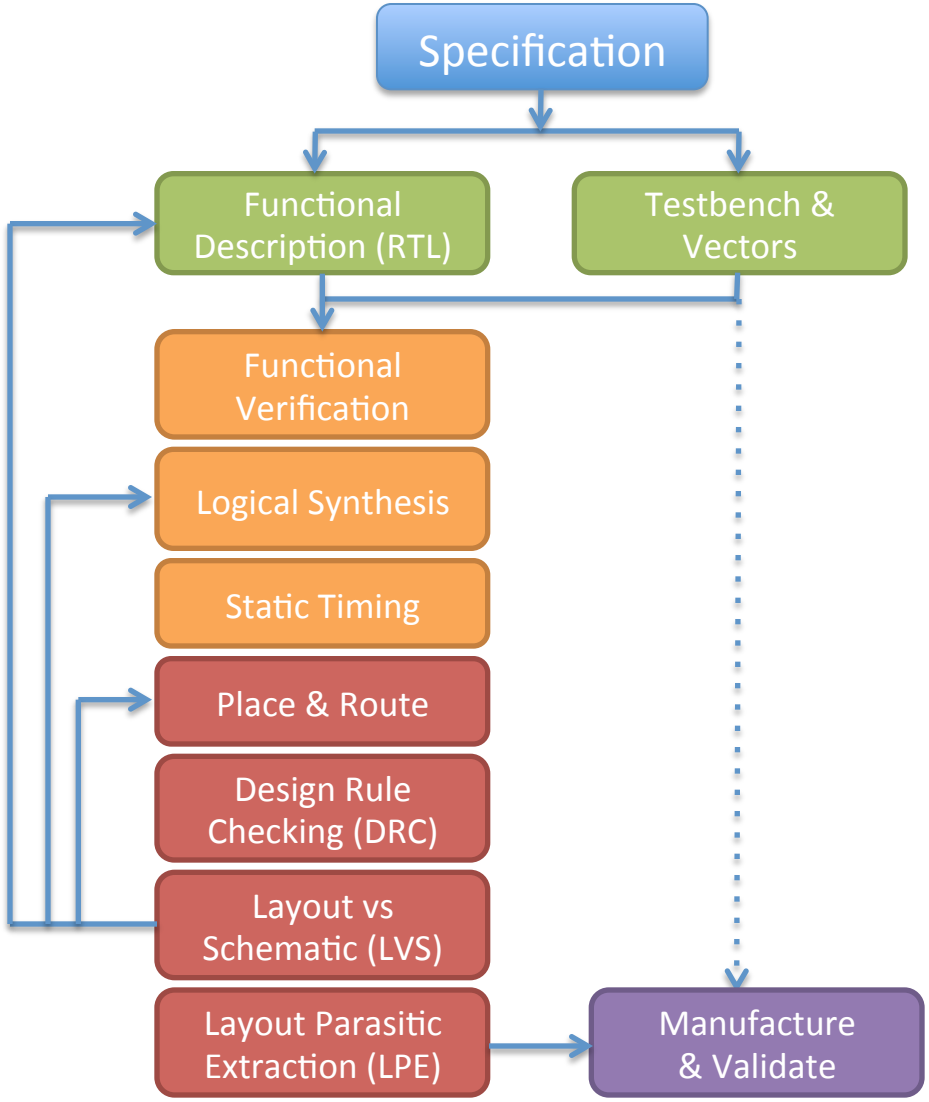
and

A single control plane controls several forwarding devices.

(That's it)

How do other industries check for correctness?

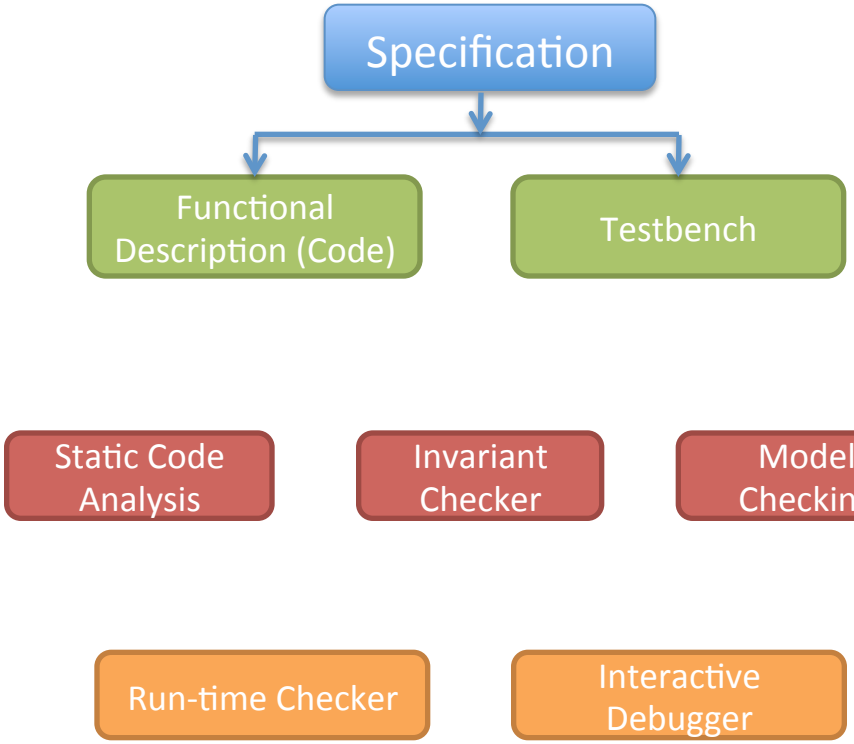
Making ASICs Work



$10B tool business supports a $250B chip industry

100s of Books
>10,000 Papers
10s of Classes

Making Software Work



$10B tool business supports a $300B S/W industry

100s of Books

>100,000 Papers

10s of Classes

Making Networks Work (Today)

`traceroute, ping, tcpdump, SNMP, Netflow`

…. er, that's about it.

Networks are kept working by

“Masters of Complexity”

A handful of books

Almost no papers

No classes

Philosophy of Making Networks Work

YoYo

"You're On Your Own"

Even simple questions are hard

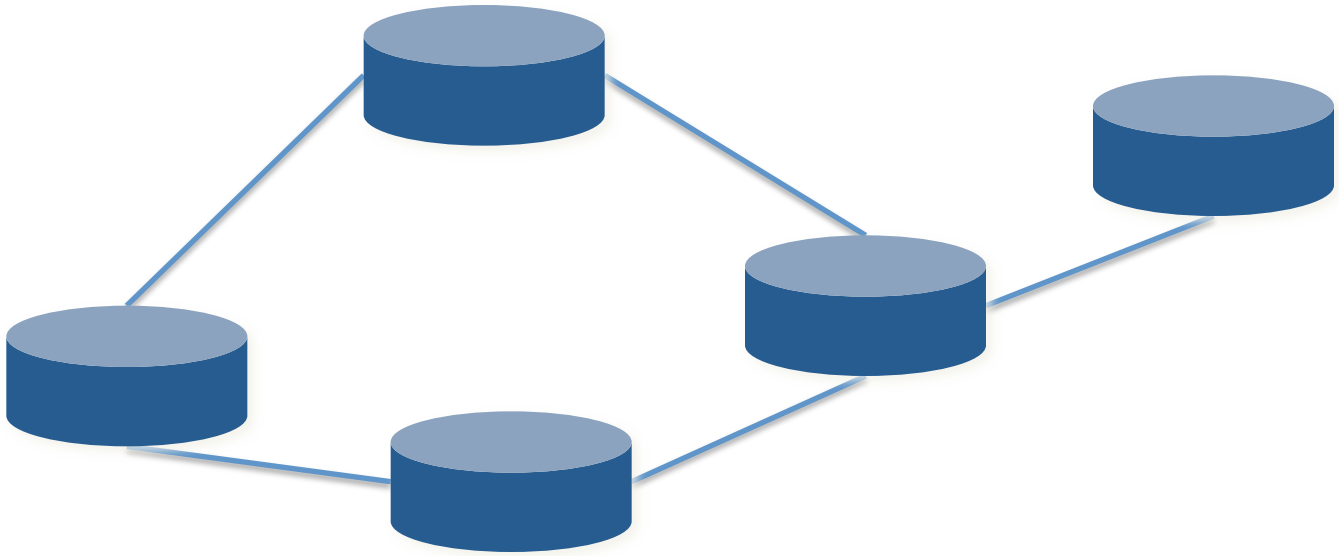
1. Can host A talk to host B?
2. What are all the packet headers from A that can reach B?
3. Are there any loops in the network?
4. Is Group X provably isolated from Group Y?
5. What happens if I remove a line in the config file?

A Bug Story

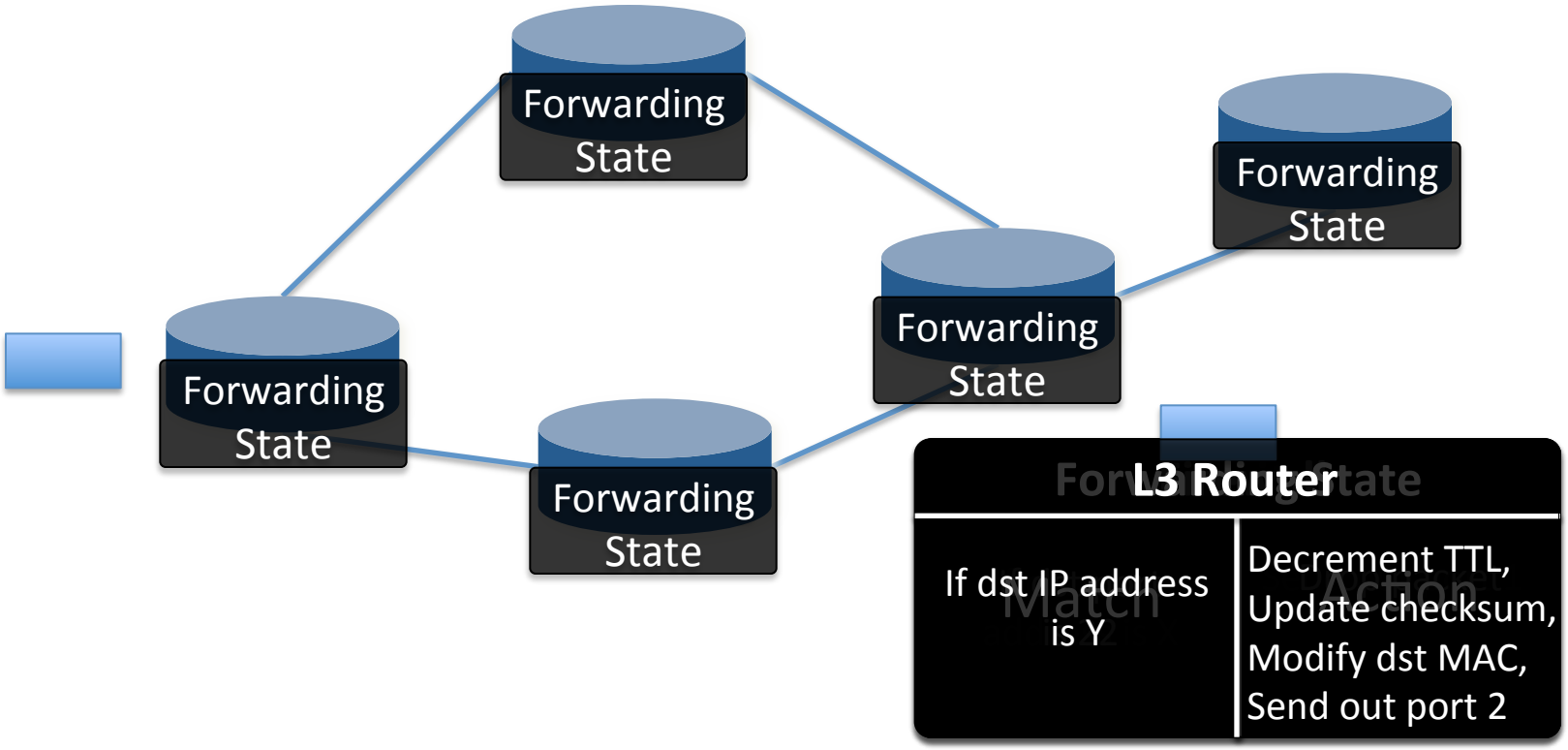
Tue, Oct 2, 2012 at 7:54 PM:

"Between 18:20-19:00 tonight we experienced a complete network outage in the building when a loop was accidentally created by CSD-CF staff. We're investigating the exact circumstances to understand why this caused a problem, since automatic protections are supposed to be in place to prevent loops from disabling the network."

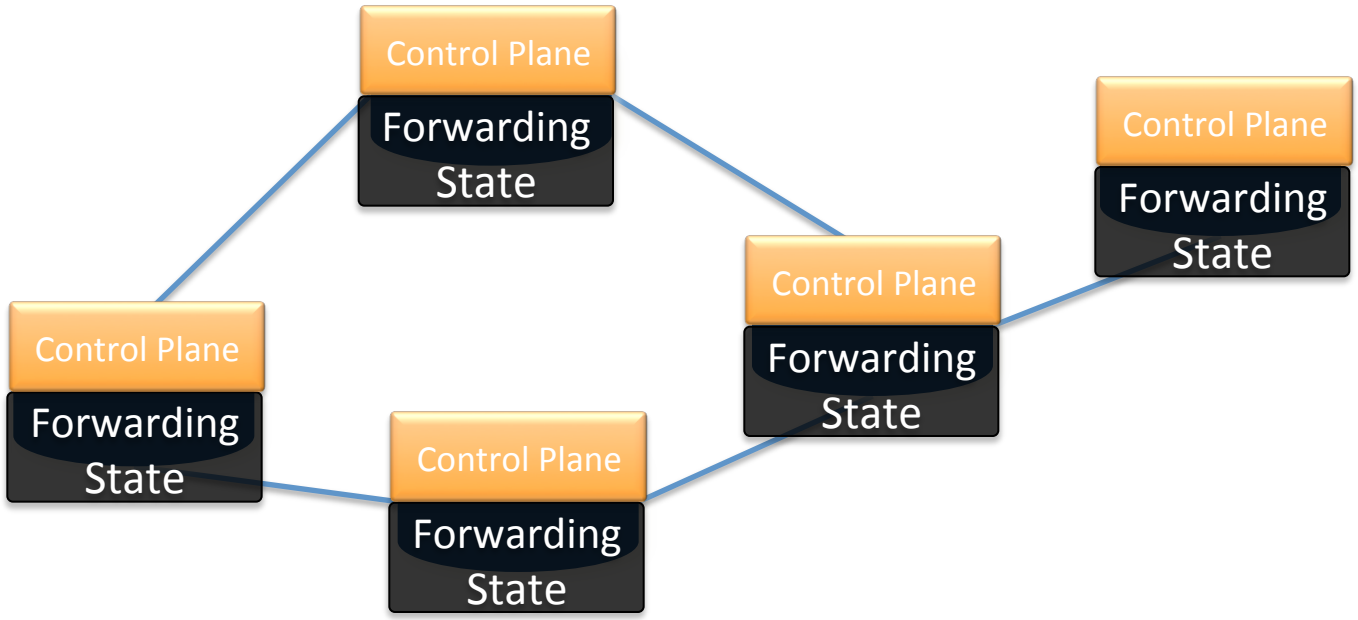
Why Troubleshooting is Hard



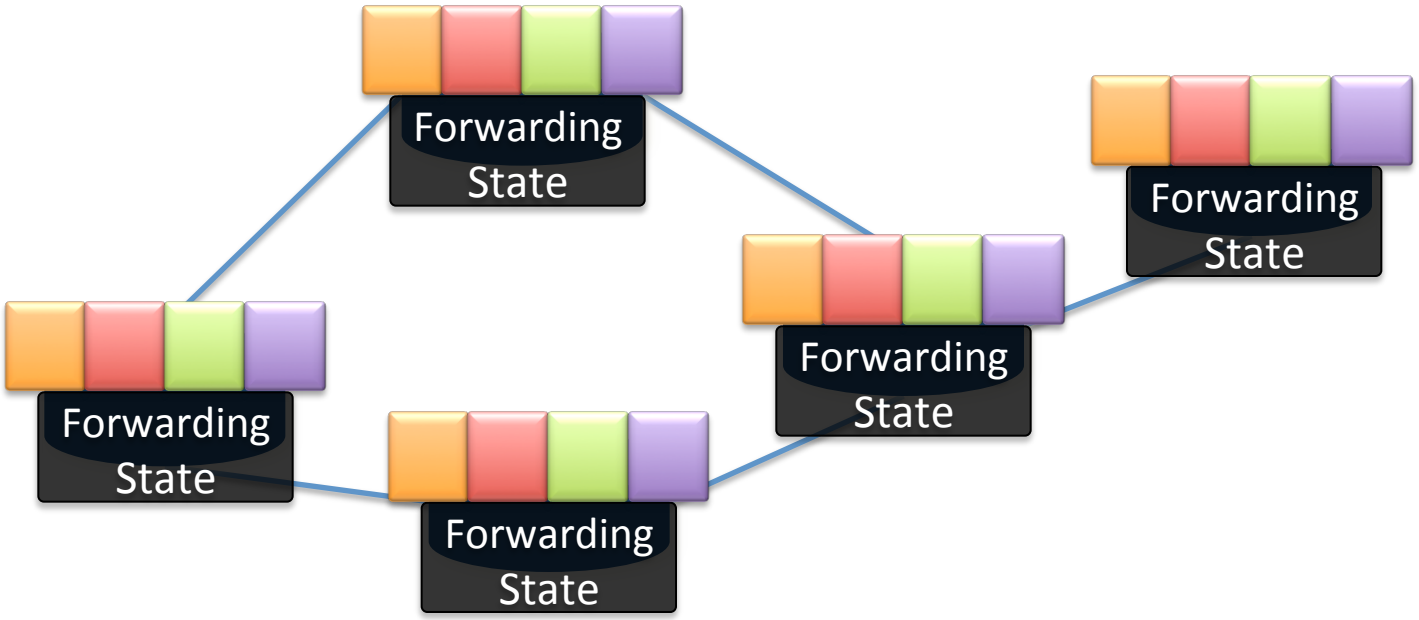
Why Troubleshooting is Hard



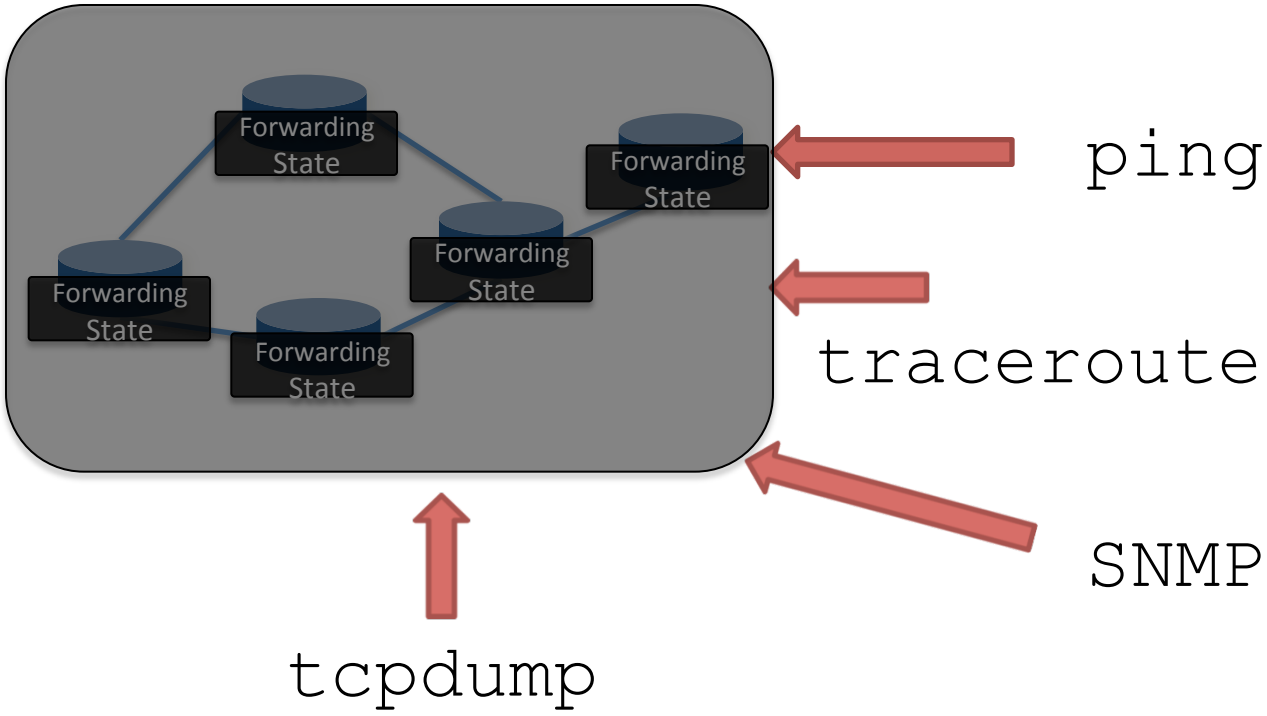
Why Troubleshooting is Hard



Why Troubleshooting is Hard



Troubleshooting Today



- Tedious and ad hoc
- Requires skill and experience
- Not guaranteed to provide helpful answers

Neither observe or control

Complex interaction

- Between multiple protocols on a switch/router.
- Between state on different switches/routers.

Multiple uncoordinated writers of state.

Operators can't…

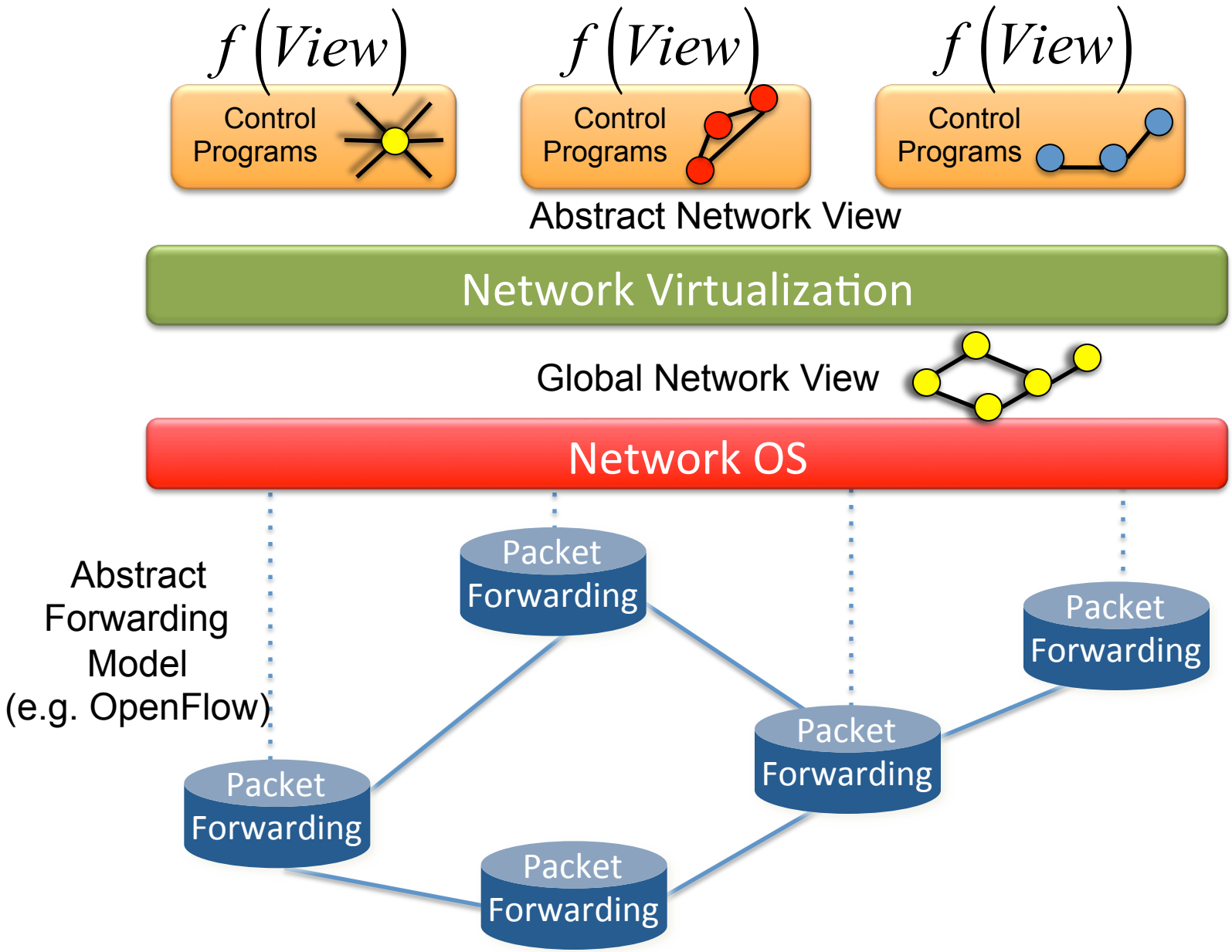
- Observe all state.
- Control all state.

How SDN helps

Scott Shenker at 1st ONS in 2011

"The Future of Networking and the Past of Protocols"

Software Defined Network (SDN)



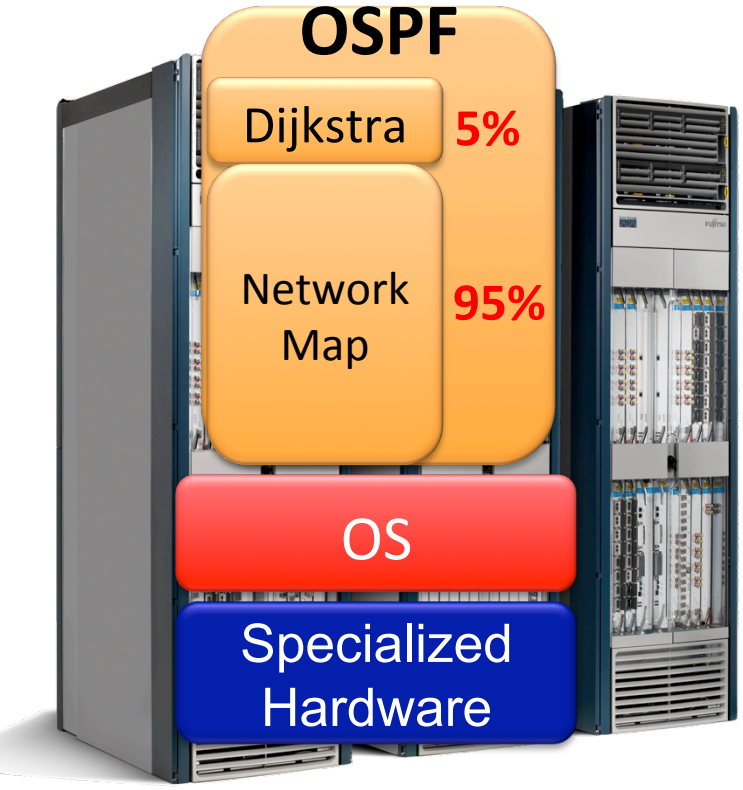
The Technical Benefits of SDN

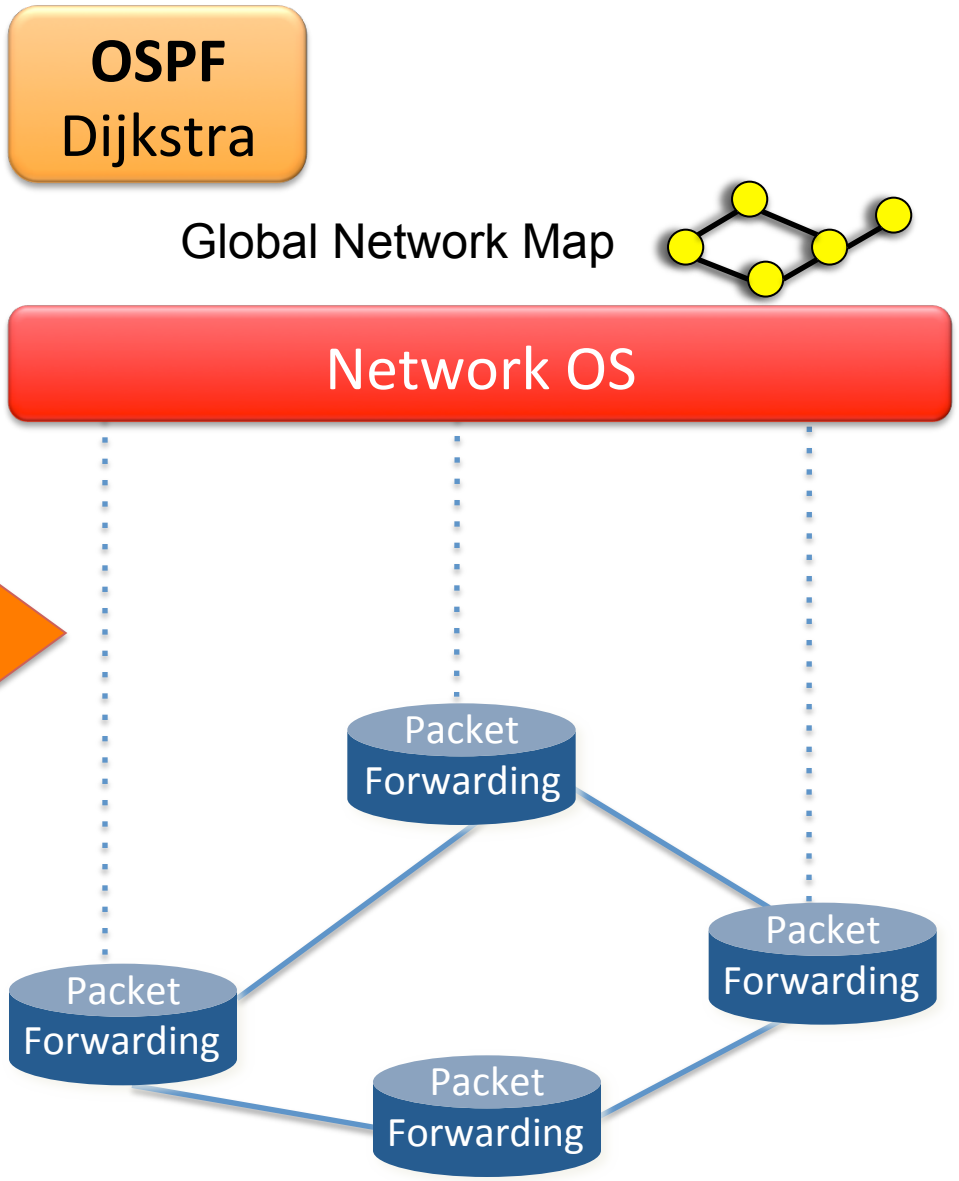
1. Well-defined control abstraction
2. Well-defined forwarding abstraction
3. Well-defined forwarding behavior

The Technical Benefits (1)

Well-defined control abstraction

- Control plane can run on modern servers
- Can adopt software engineering best-practices
- Easier to add new control programs
- …or customize locally
- Solve distributed systems problem once, rather than for every protocol


OSPF
Dijkstra
5%
Network Map
95%
OS
Specialized Hardware


OSPF
Dijkstra
Global Network Map
Network OS
Packet Forwarding
Packet Forwarding
Packet Forwarding
Packet Forwarding

The Technical Benefits of SDN

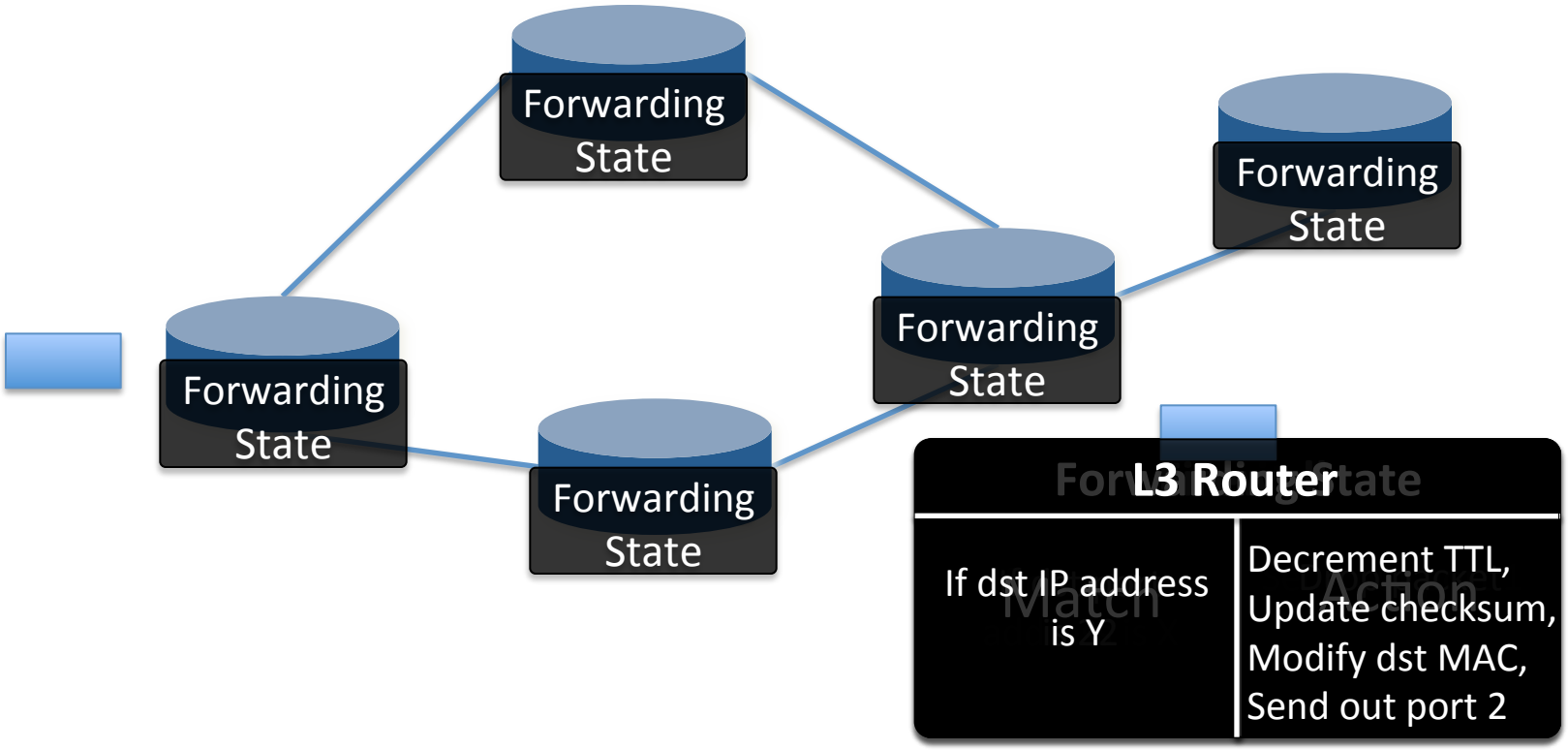
1. Well-defined control abstraction
2. Well-defined forwarding abstraction
3. Well-defined forwarding behavior

The Technical Benefits (2)

Well-defined forwarding abstraction

- e.g. OpenFlow
- Vendor-agnostic interface to forwarding plane
- Simpler, lower-cost, lower-power hardware

Match-Action Forwarding Abstraction

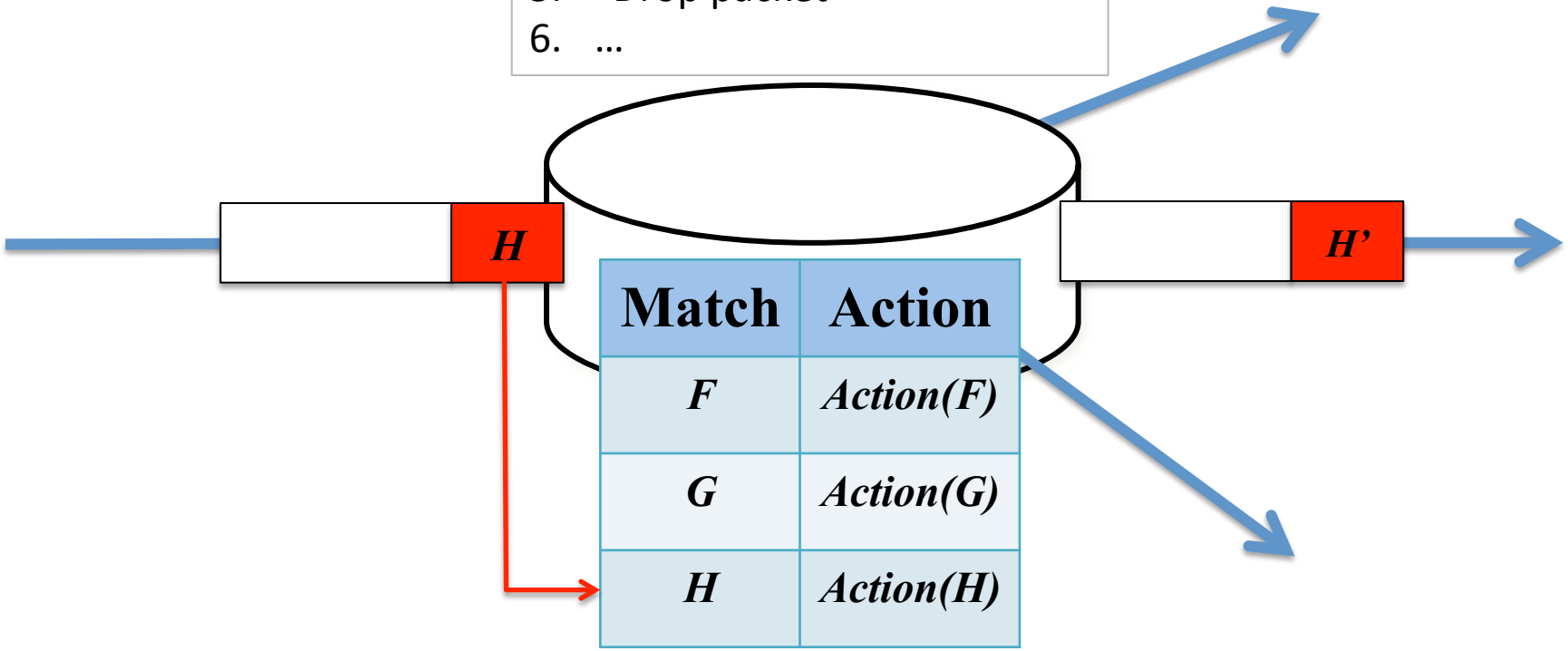


Match-Action Forwarding Abstraction

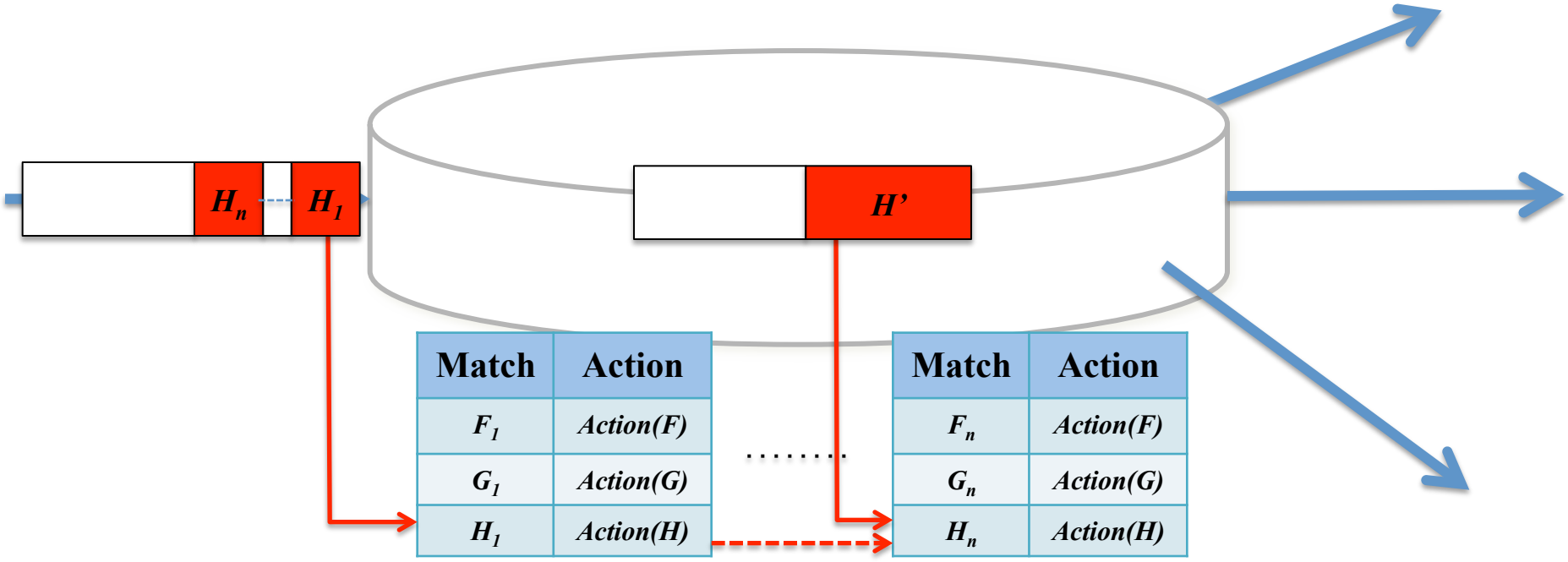
Action Primitives

1. "Forward to ports 4 & 5"
2. "Push header Y after bit 12"
3. "Pop header bits 8-12"
4. "Decrement bits 13-18"
5. "Drop packet"
6. …

"Plumbing primitives"



Multiple Table Match-Action



OpenFlow Philosophy

Long-term, forwarding looking

Match: Very general, not protocol specific.

Action: Small instruction set, not protocol specific.

– Make it easy to add new headers and actions.

– Any network (packet, circuit, radio).

Short-term, backward looking

Match: include well-known header fields.

Action: necessary set for existing protocols.

– Support existing protocols on existing switch chips.

The Technical Benefits of SDN

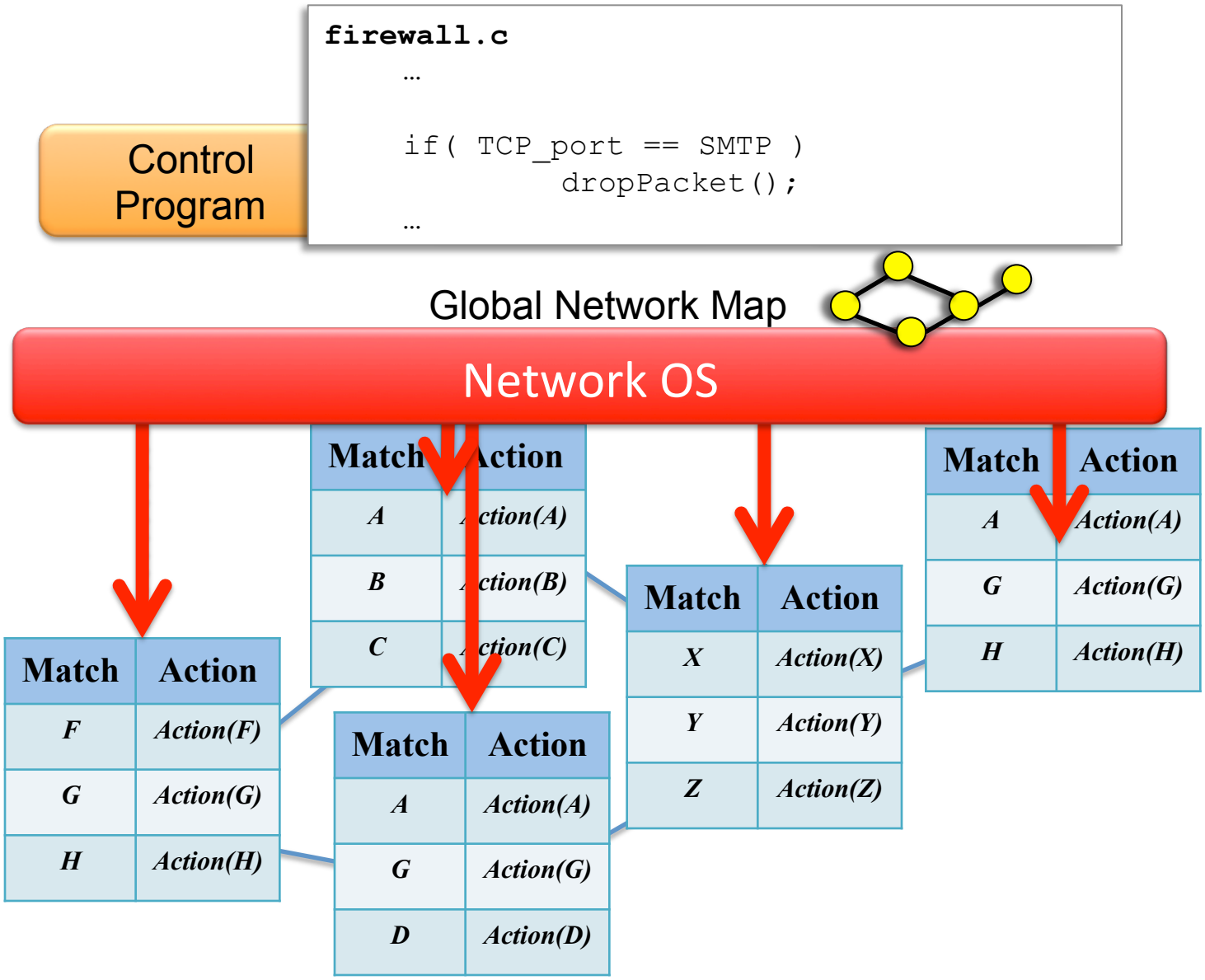
1. Well-defined control abstraction
2. Well-defined forwarding abstraction
3. Well-defined forwarding behavior

The Technical Benefits (3)

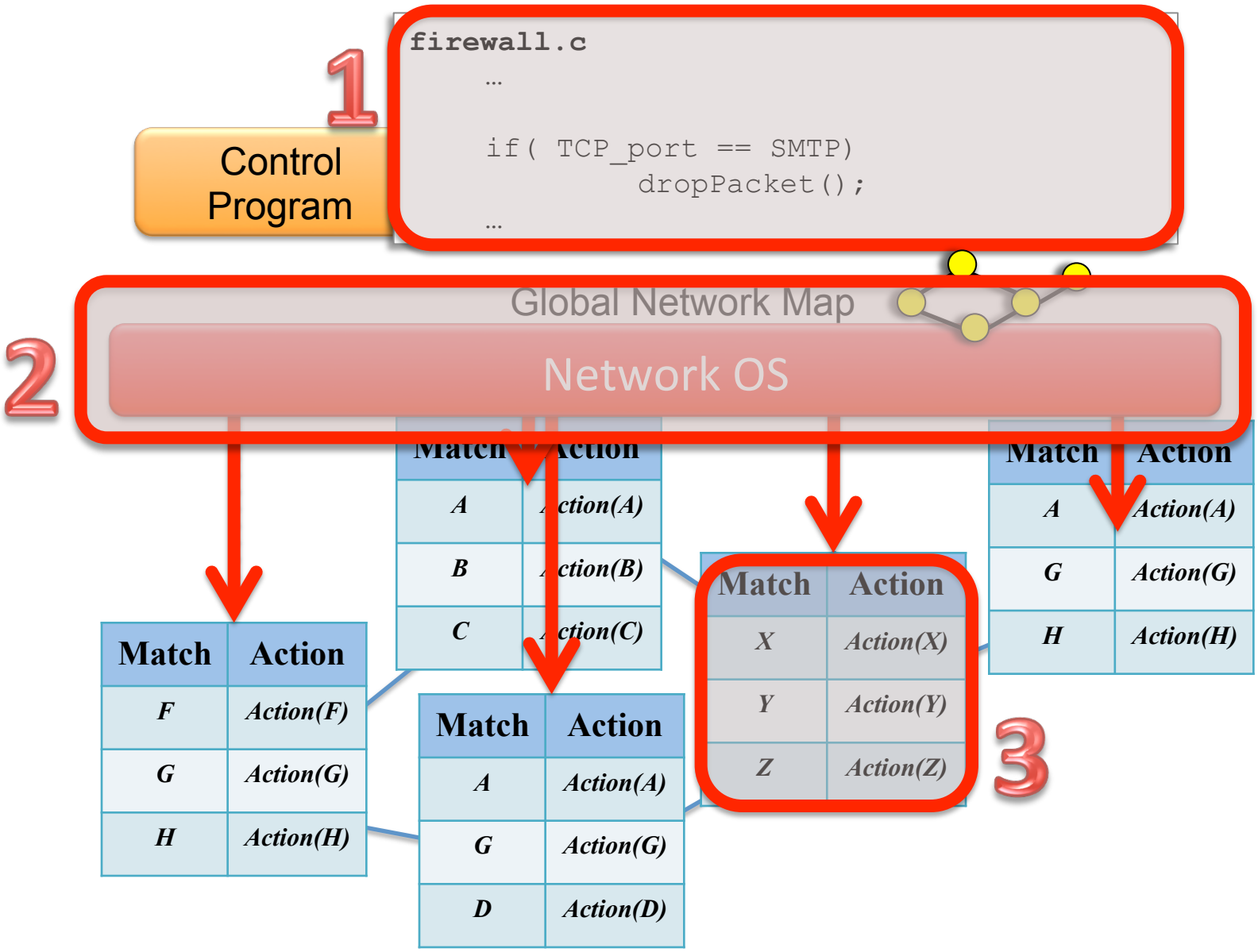
Well-defined forwarding behavior

- The forwarding tables capture the entire forwarding behavior.
- Control plane writes the forwarding state.
- Therefore, we can verify its correctness.

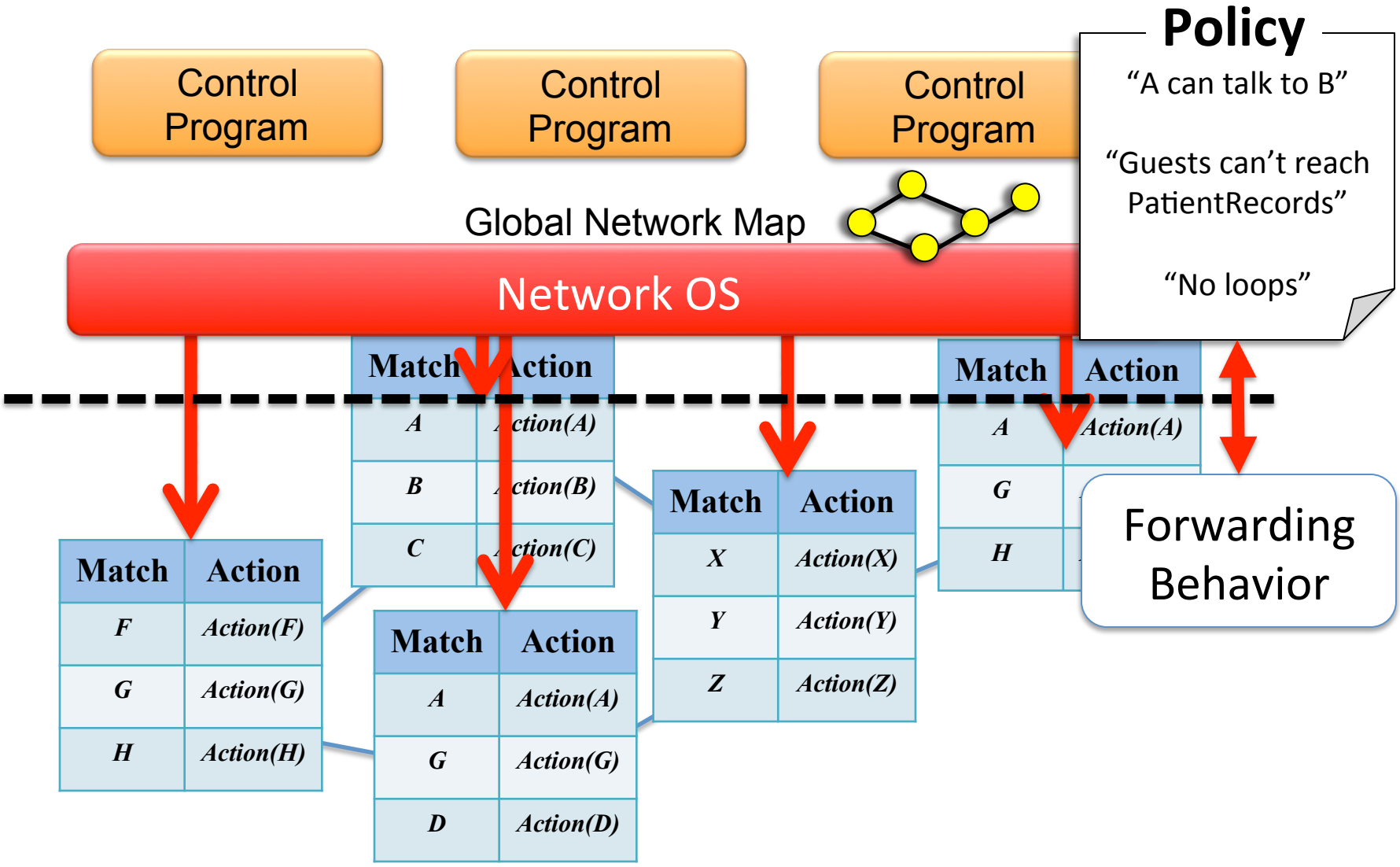
Software Defined Network (SDN)



Software Defined Network (SDN)



Software Defined Network (SDN)



Methods for Forwarding Correctness

1. Static Checking (e.g. HSA, Anteater)
   "Independently checking correctness"

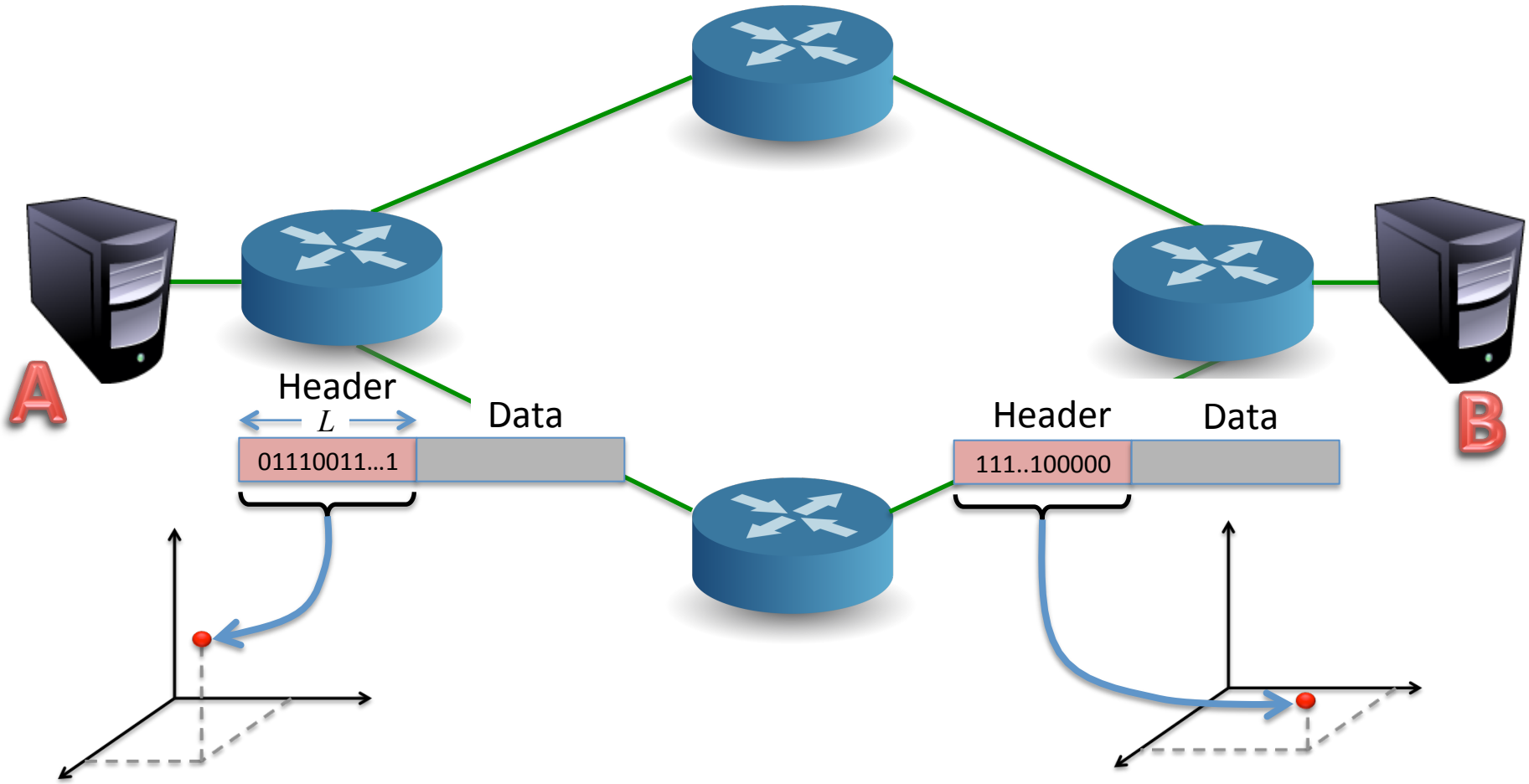
2. Dynamic Checking (e.g. NetPlumber, Veriflow)
   "Checking new state before it is added"

3. Automatic Testing (e.g. ATPG)
   "Is the datapath behaving correctly?"

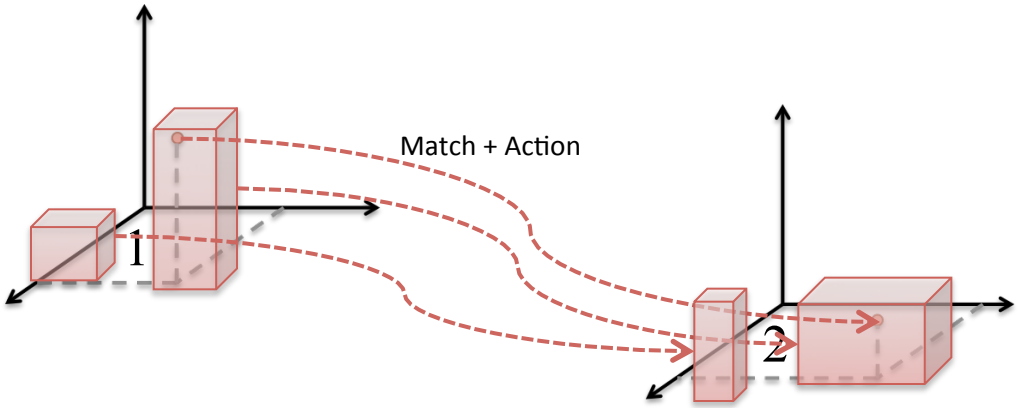
4. Interactive Debugging (e.g. NetSight)
   "Finding bugs, and their root cause,
   in an operational network"

Header Space Analysis

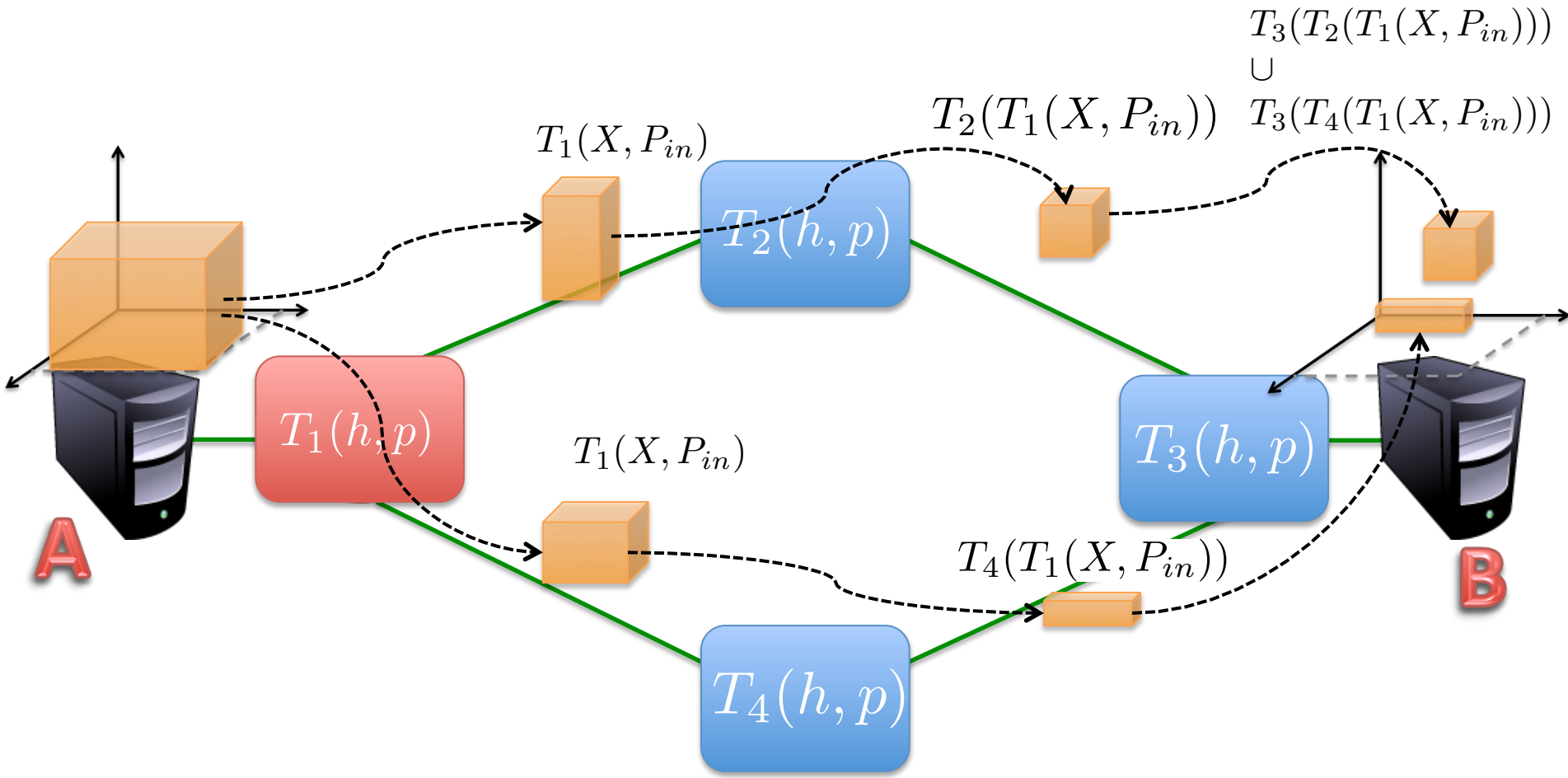
[NSDI '12]



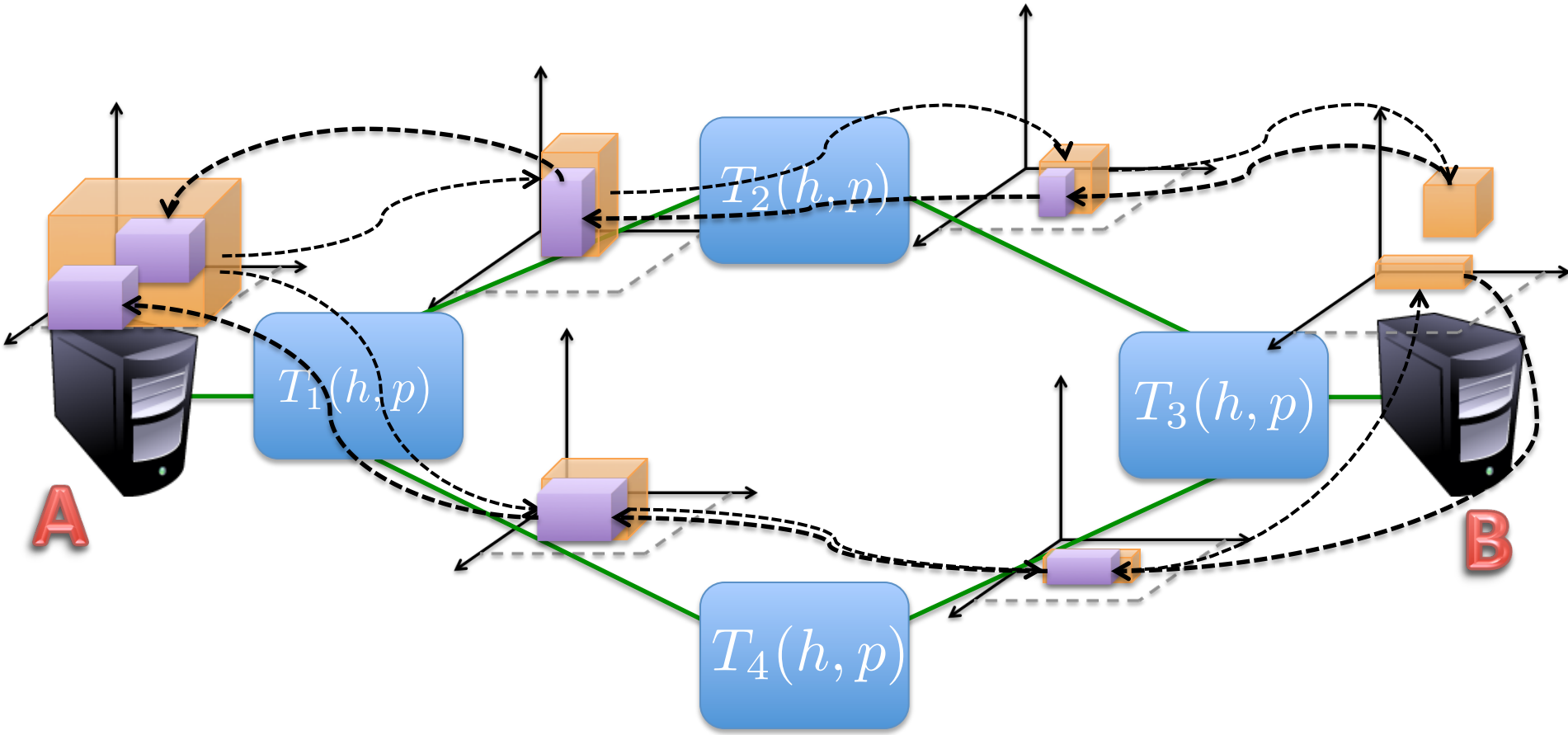
Header Space Analysis



The set of packets from A that can reach B



All packets from A that can reach B



Implications

Short term:

1. Finds all packets from A that can reach B
2. Find loops, regardless of protocol or layer
3. Can prove that two groups are isolated
4. …for (mostly) stateless graph mappings

Longer term:

1. Abstract forwarding model; protocol independent
2. Analagy to Boolean algebra for digital logic design
3. Can be a foundation for a variety of tools

Methods for Forwarding Correctness

1. Static Checking (e.g. HSA, Anteater)
   "Independently checking correctness"

2. Dynamic Checking (e.g. NetPlumber, Veriflow)
   "Checking new state before it is added"

3. Automatic Testing (e.g. ATPG)
   "Is the datapath behaving correctly?"

4. Interactive Debugging (e.g. NetSight)
   "Finding bugs, and their root cause, in an operational network"

Packet Histories

Basic idea:

- Capture forwarding plane events
- Look for errant behavior (real time or offline)

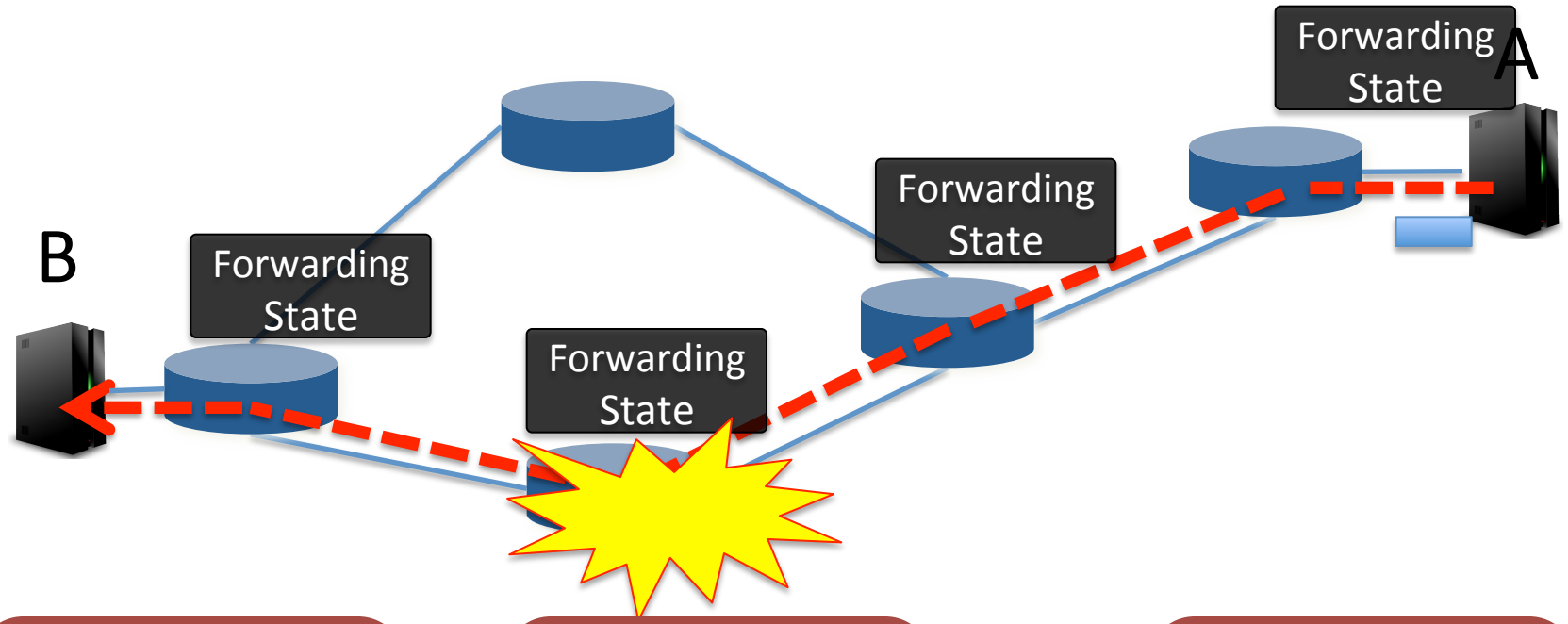
Enormous amounts of data:

- Scalability is essential
- Another example of Big Data

NetSight

A platform to capture and filter packet histories

A Troubleshooting Construct



Symptom: Errant packet behavior in the dataplane

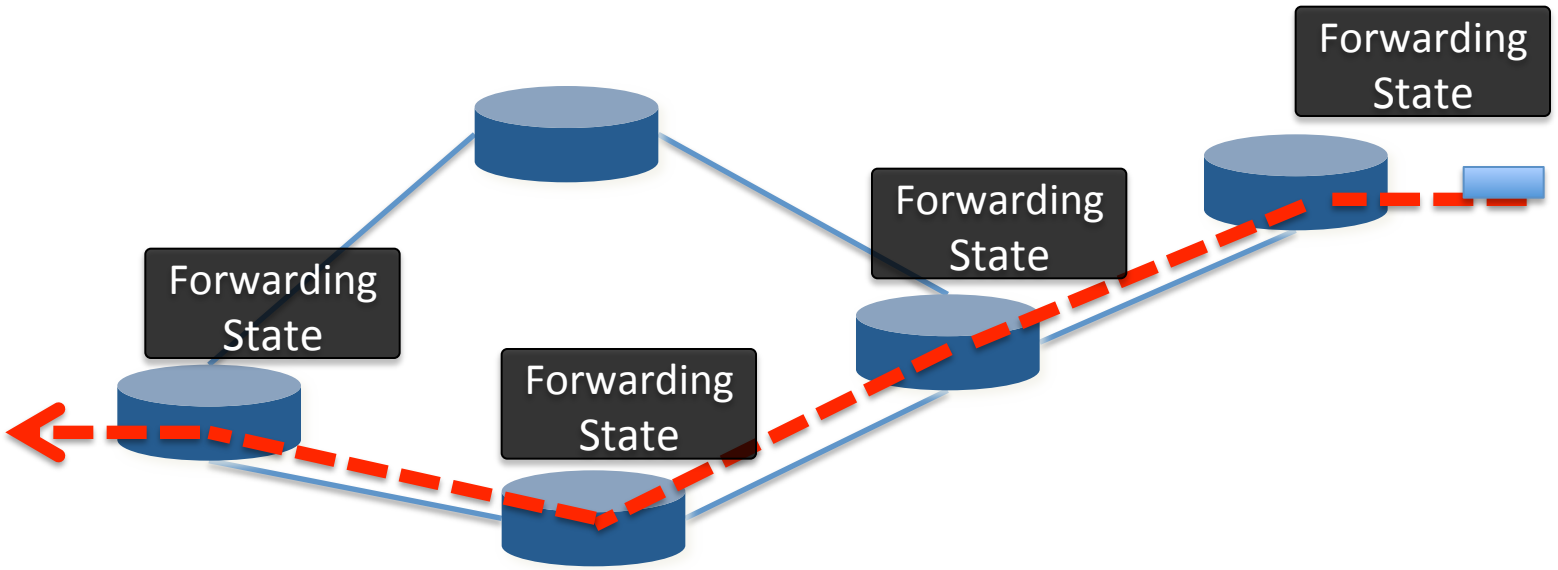
+

Cause: Forwarding State

→

Packet History

Packet History

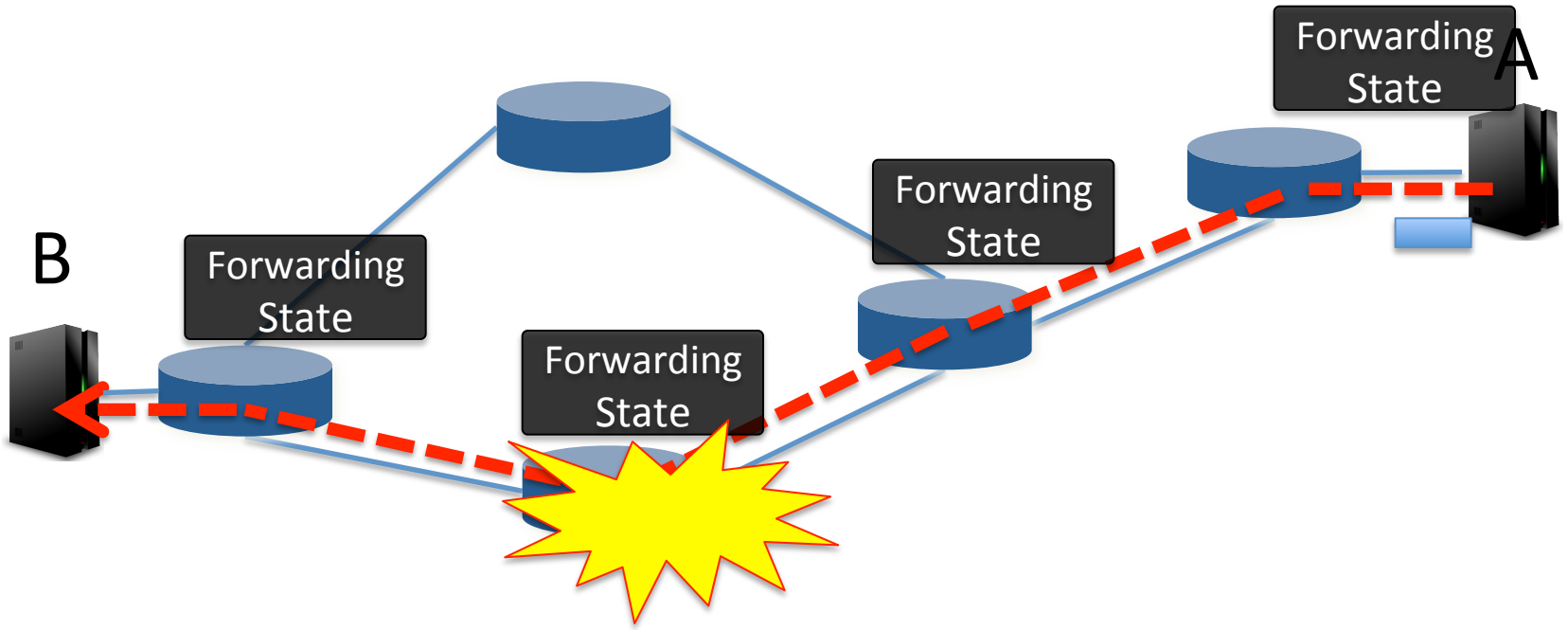


Packet history = Path taken by a packet

+ Header modifications

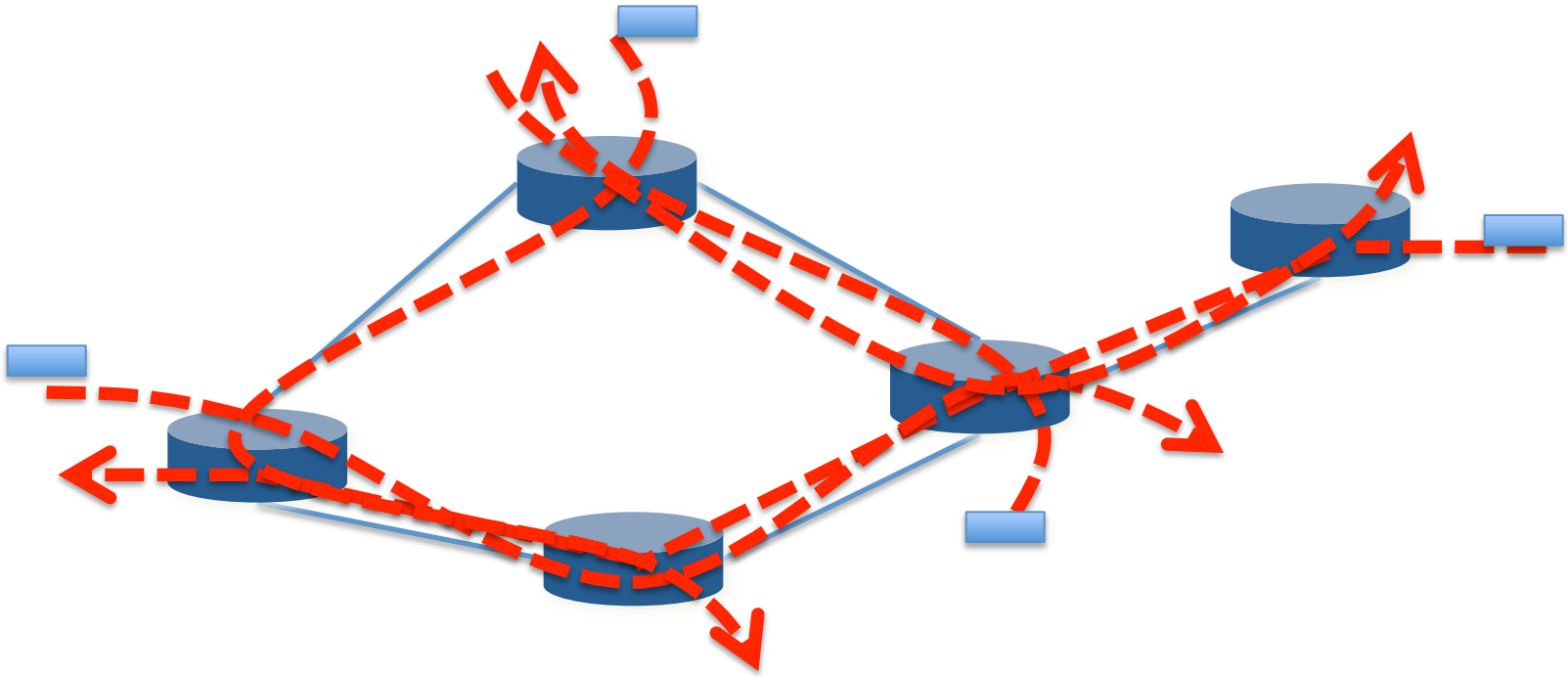
+ Switch state encountered

Troubleshooting Workflow



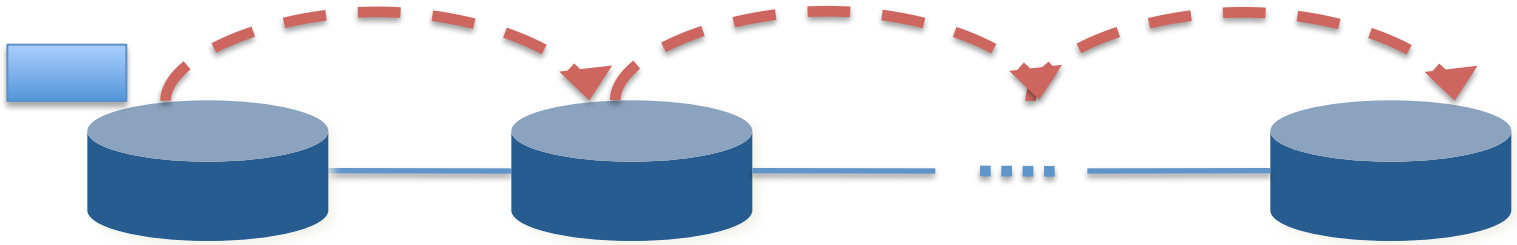
1. Ask for the packet history of errant packets
2. Use the packet history to diagnose the root cause

Packet History



How to specify packet histories of interest?

Querying Packet Histories



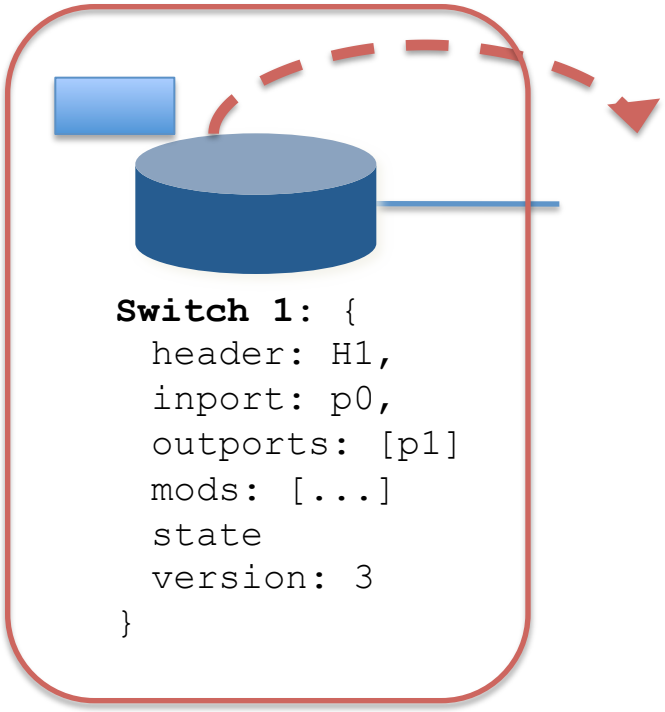
```
Switch 1: {
  header: H1,
  inport: p0,
  outports: [p1]
  mods: [...]
  state
  version: 3
}
```

```
Switch 2: {
  header: H2,
  inport: p0,
  outports: [p3]
  mods: [...]
  state
  version: 5
}
```

```
Switch N: {
  header: HN,
  inport: p3,
  outports: [p2]
  mods: [...]
  state
  version: 13
}
```

Packet History Filter: A regular-expression-like language to specify packet histories of interest

Postcard Filter



Postcard Filter

```
--bpf [not] <BPF>
--dpid [not] <switch ID>
--inport [not] <input port>
--outport [not] <output port>
--version [not] <state version>
```

Example:

```
--bpf "ip src 10.0.1.2" --dpid 5 --inport not 1
```

Building PHFs using Postcard Filters

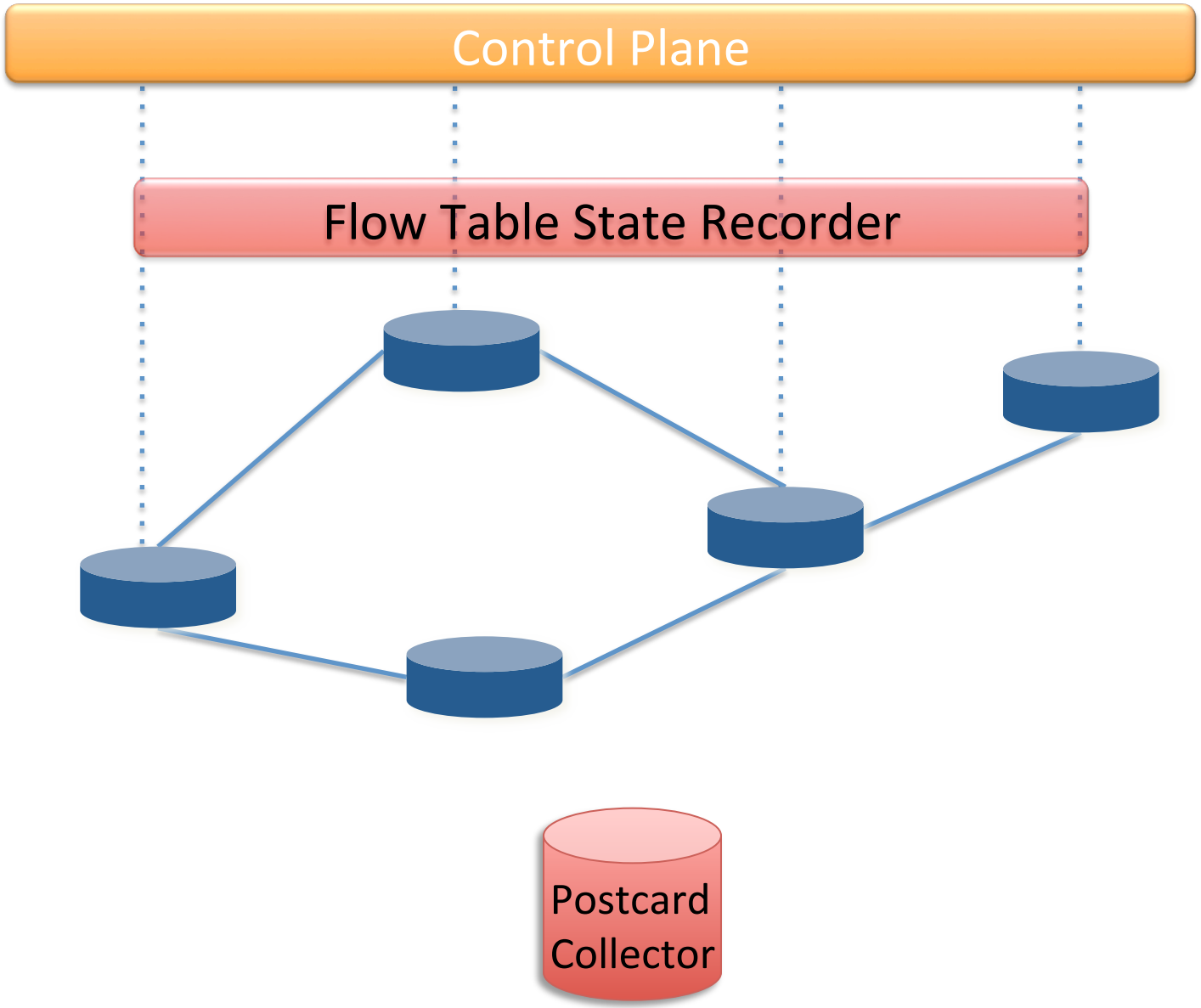
PHF to match packets that:

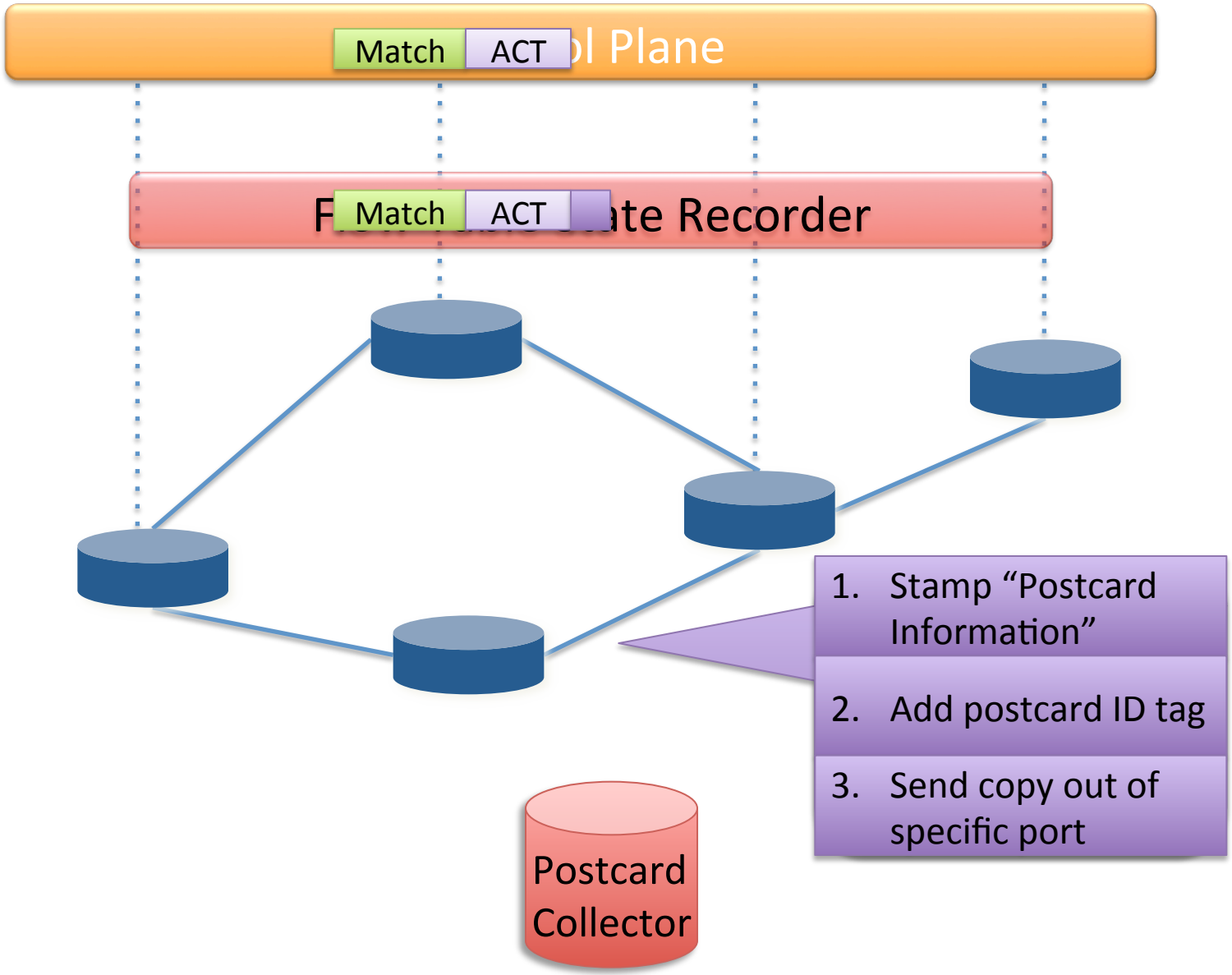
- start at switch X, and end at switch Y:

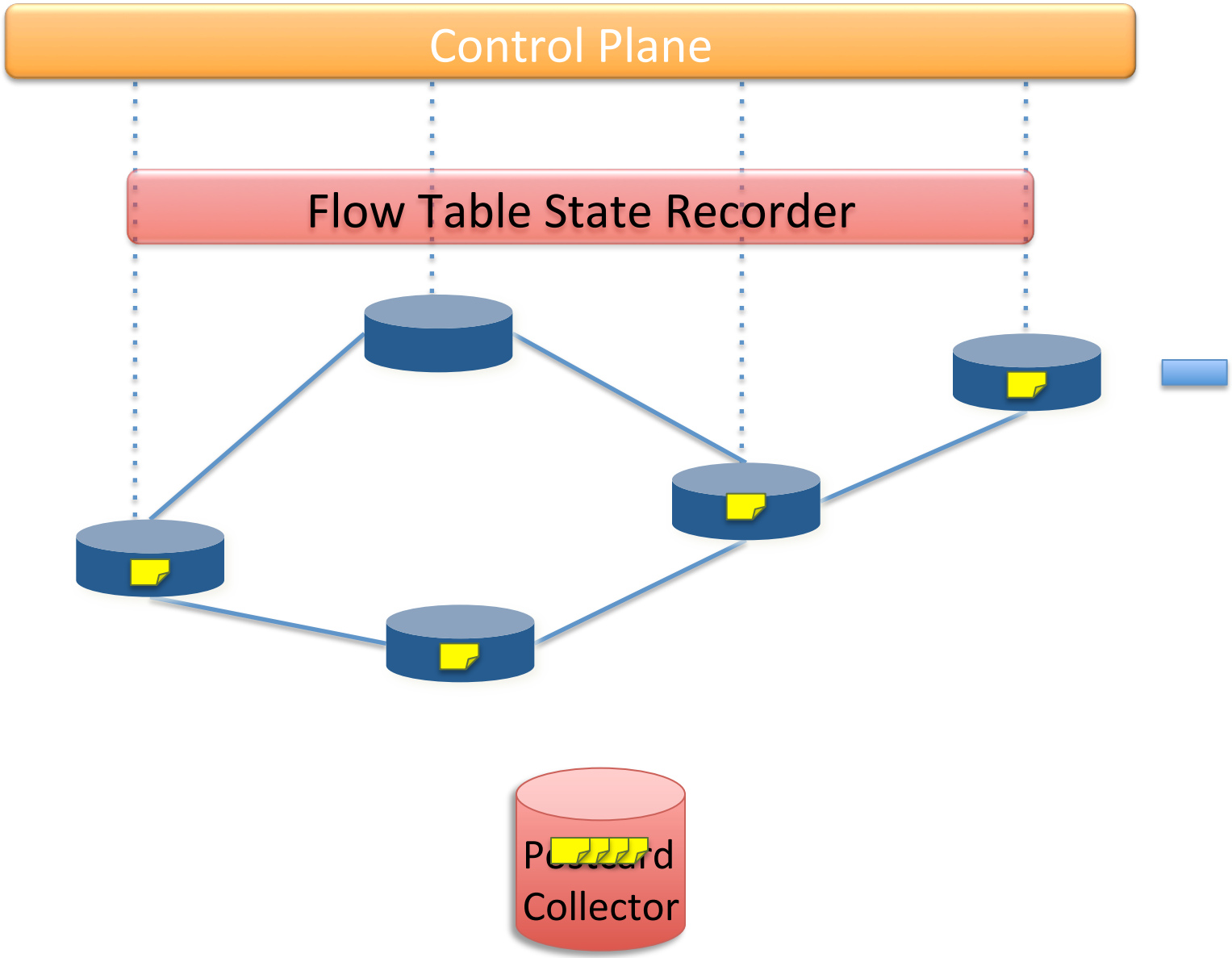
```
{{--dpid X}}.*{{--dpid Y}}$
```

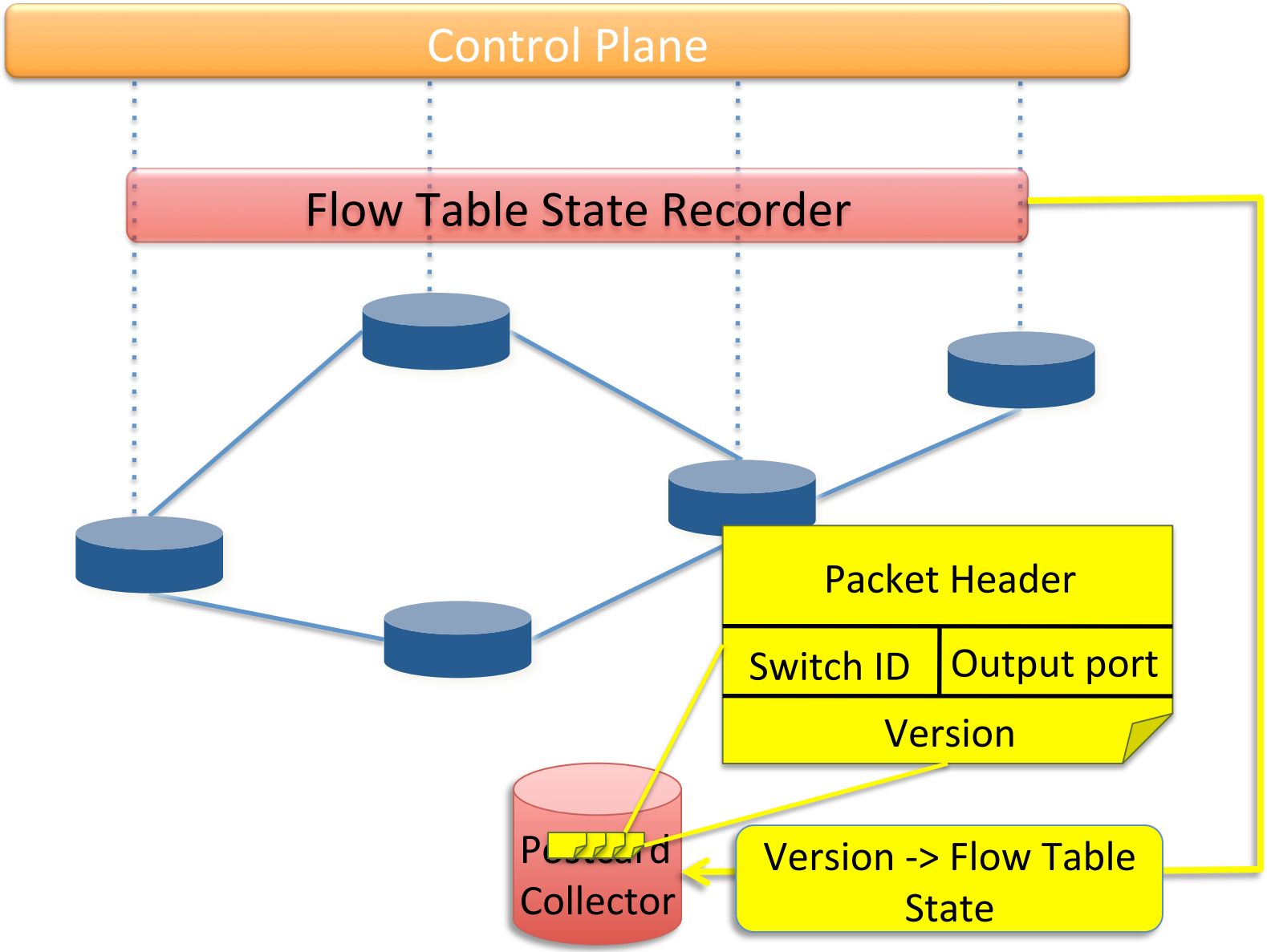
- are of type UDP, start at switch X, never reach Y:

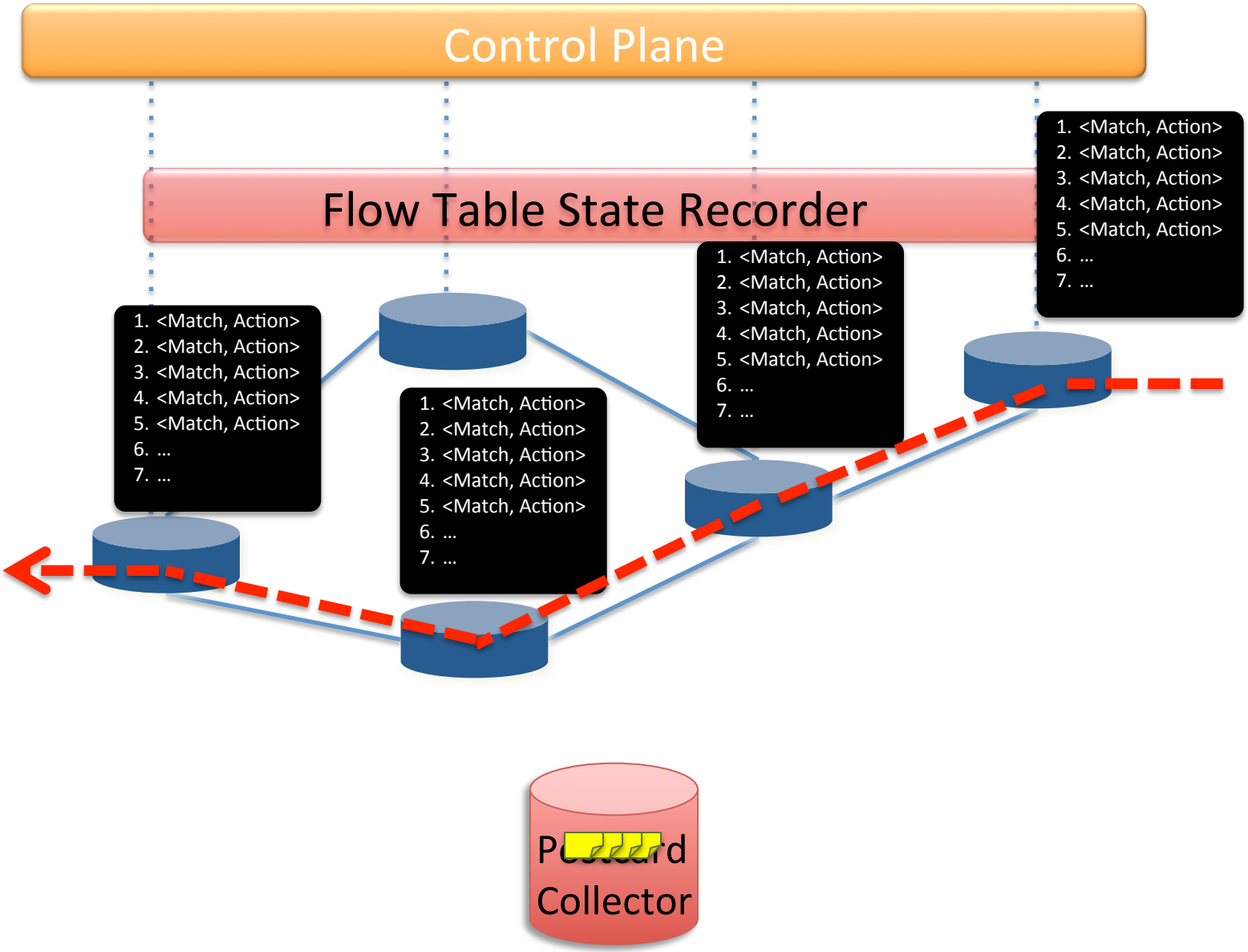
```
{{--bpf "udp" --dpid X}}[^{{--dpid Y}}]*$
```


Control Plane
Flow Table State Recorder
Postcard
Collector


Match
ACT
Match
ACT
1. Stamp "Postcard Information"
2. Add postcard ID tag
3. Send copy out of specific port
Postcard Collector


Control Plane
Flow Table State Recorder
Collector


Control Plane
Flow Table State Recorder
Packet Header
Switch ID
Output port
Version
Collector
Version -> Flow Table State


Control Plane
Flow Table State Recorder
1. <Match, Action>
2. <Match, Action>
3. <Match, Action>
4. <Match, Action>
5. <Match, Action>
6. …
7. …
1. <Match, Action>
2. <Match, Action>
3. <Match, Action>
4. <Match, Action>
5. <Match, Action>
6. …
7. …
1. <Match, Action>
2. <Match, Action>
3. <Match, Action>
4. <Match, Action>
5. <Match, Action>
6. …
7. …
1. <Match, Action>
2. <Match, Action>
3. <Match, Action>
4. <Match, Action>
5. <Match, Action>
6. …
7. …
Collector

Methods for Forwarding Correctness

1. Static Checking (e.g. HSA, Anteater)
   "Independently checking correctness"

2. Dynamic Checking (e.g. NetPlumber, Veriflow)
   "Checking new state before it is added"

3. Automatic Testing (e.g. ATPG)
   "Is the datapath behaving correctly?"

4. Interactive Debugging (e.g. NetSight)
   "Finding bugs, and their root cause,
   in an operational network"

<end>