VERIFLOW

Network Verification: From Algorithms to the Real World

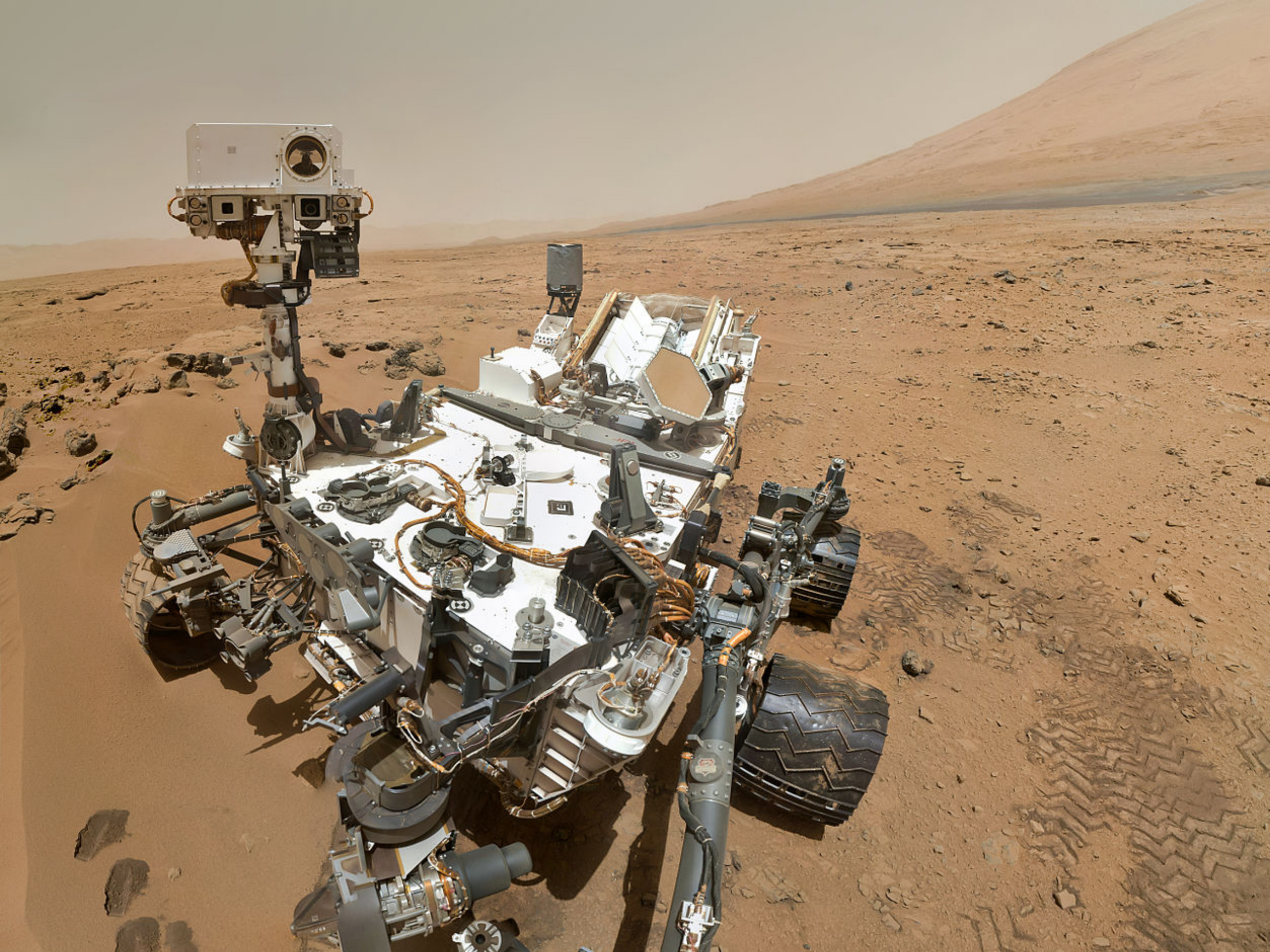
Wenxuan Zhou
Veriflow & UIUC

A simple idea about complexity…

Networks are so complex it's hard to know they're doing the right thing.

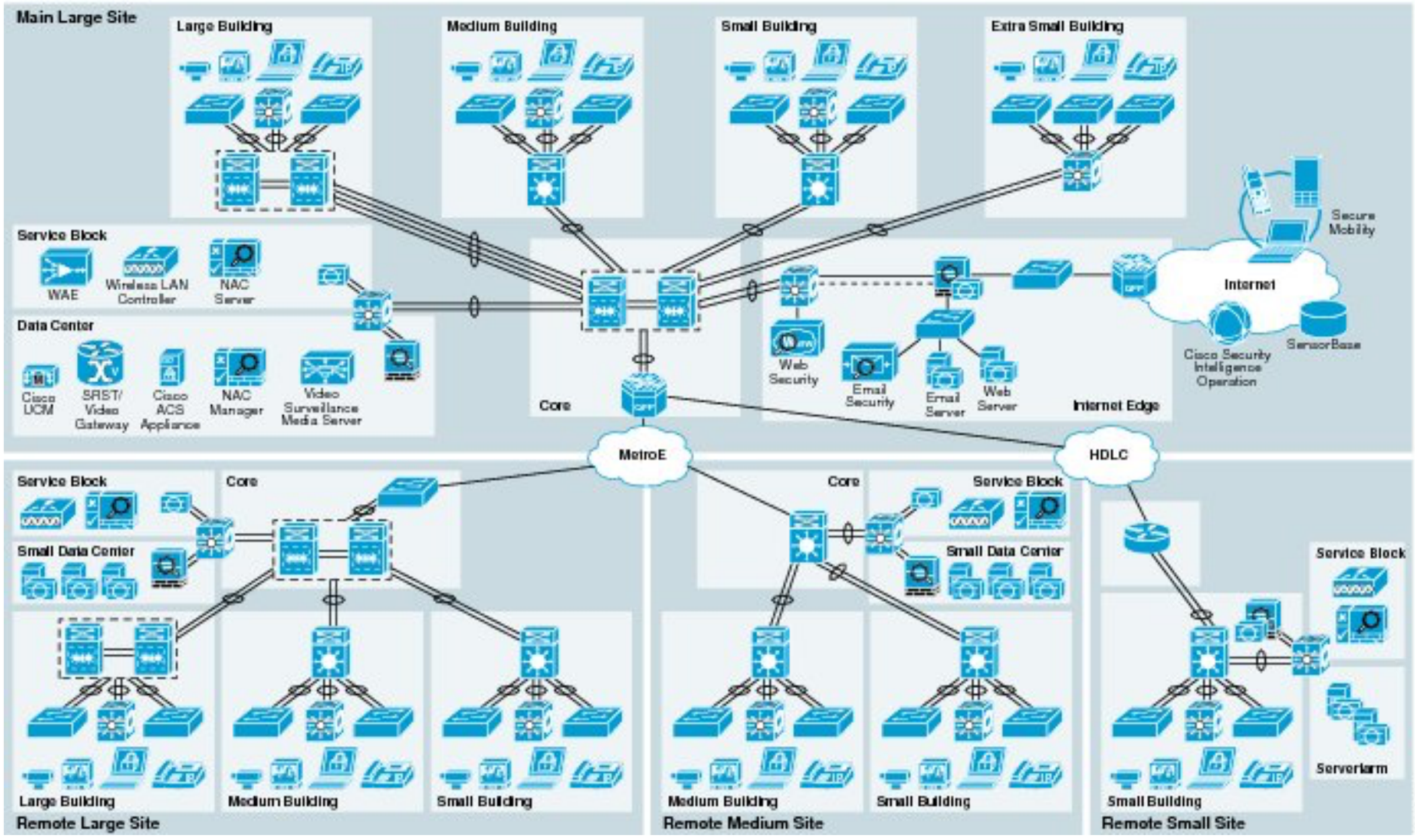
Let's automate.

VERIFLOW



NETWORKING BACKGROUND

Inside a typical enterprise network



Source: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Medium_Enterprise_Design_Profile/MEDP/chap5.html

VERIFLOW

Configs use many protocols & features

Layer 1 protocols (physical layer)

USB Physical layer

Ethernet physical layer including 10 BASE T, 100 BASE T,100 BASE TX,100 BASE FX, 1000 BASE T and other variants

varieties of 802.11 Wi-Fi physical layers

DSL

List of protocols commonly encountered by CCNAs
https://learningnetwork.cisco.com/docs/DOC-25649

VERIFLOW

Configs use many protocols & features

MPLS Multi-protocol label switching
NAT Network Address translation
OSPF Open Shortest Path First
VRRP Virtual Router Redundancy Protocol

Layer 4 (transport layer or Host-to-Host layer)

AH
TCP
UDP

Layer 5 (session layer or application layer in DoD)

NetBIOS File sharing and name resolution protocol -the basis of file sharing in windows
NFS Network File System

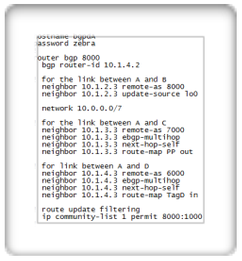
Layer 7 (application layer)

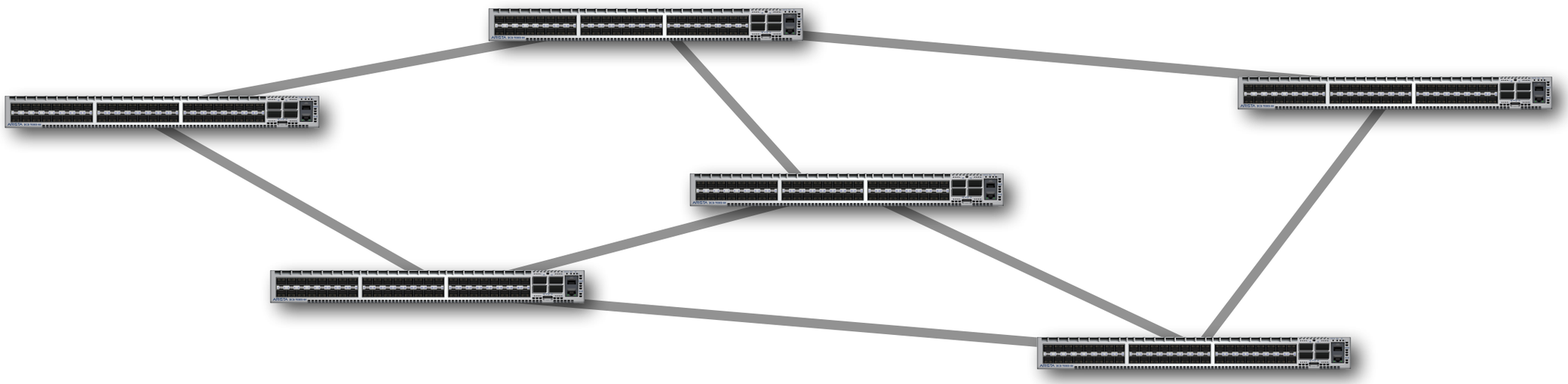
BitTorrent
BGP
DNS
DHCP
FTP
HTTP
HTTPS
IRC
NTP
POP3
RTP
SSH
SMTP
SNMP
Telnet
TFTP
URL

List of protocols commonly encountered by CCNAs
https://learningnetwork.cisco.com/docs/DOC-25649

VERIFLOW

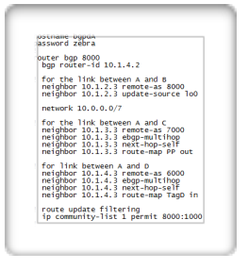
Distributed route computation

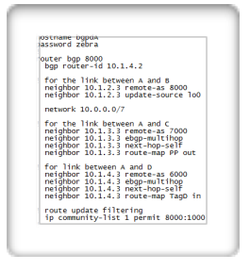


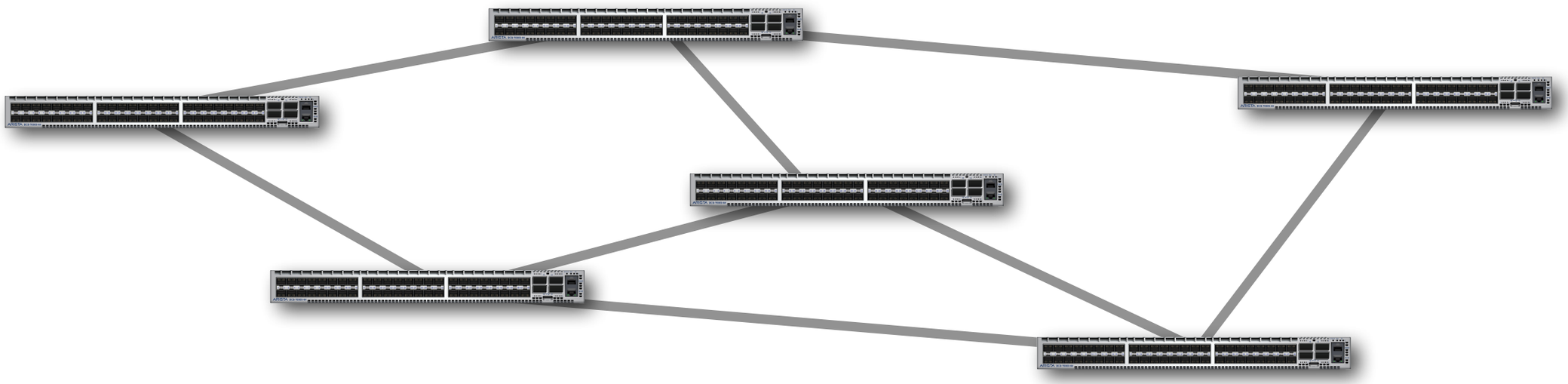


VERIFLOW

Distributed route computation

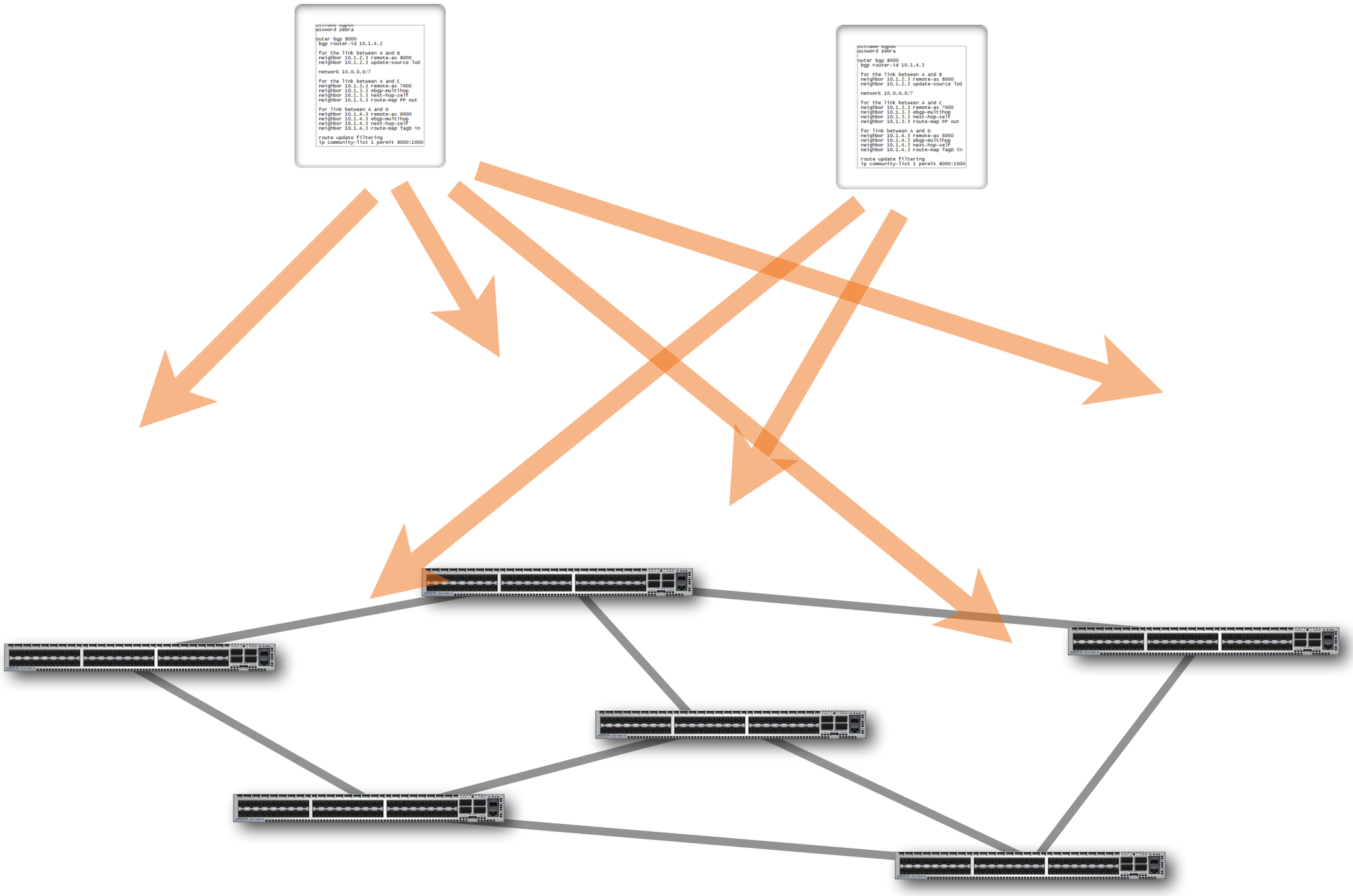




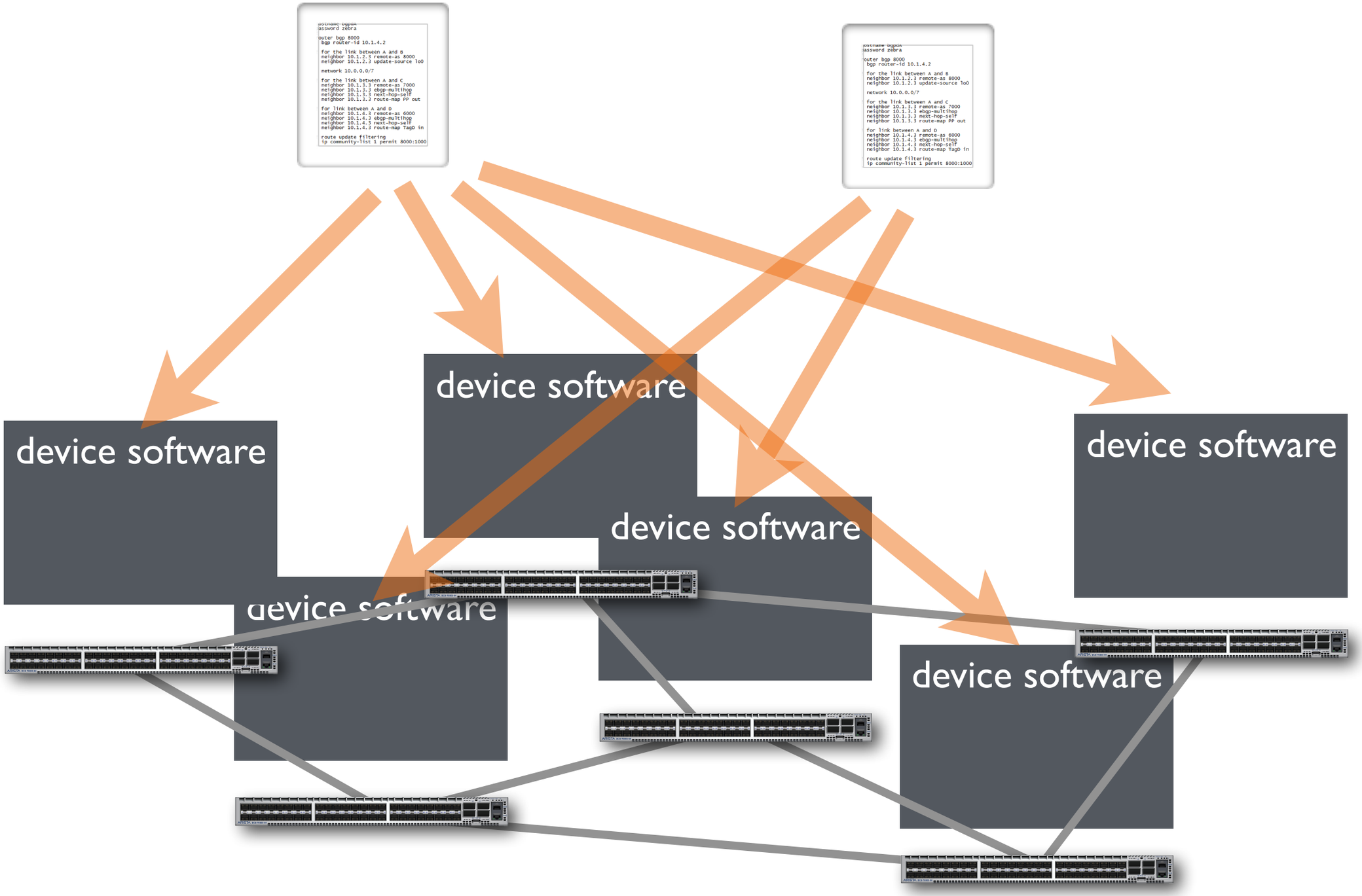


VERIFLOW

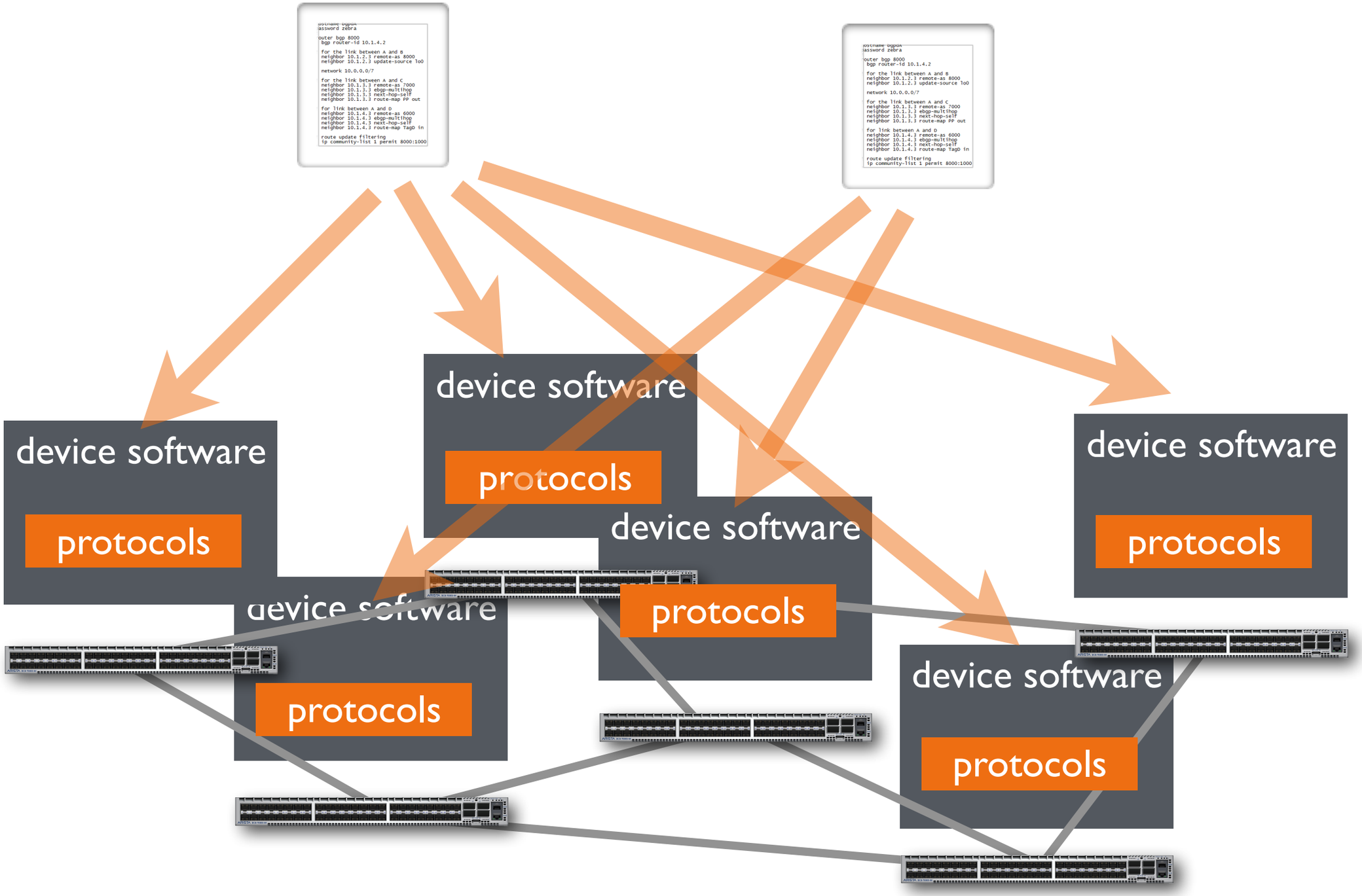
Distributed route computation



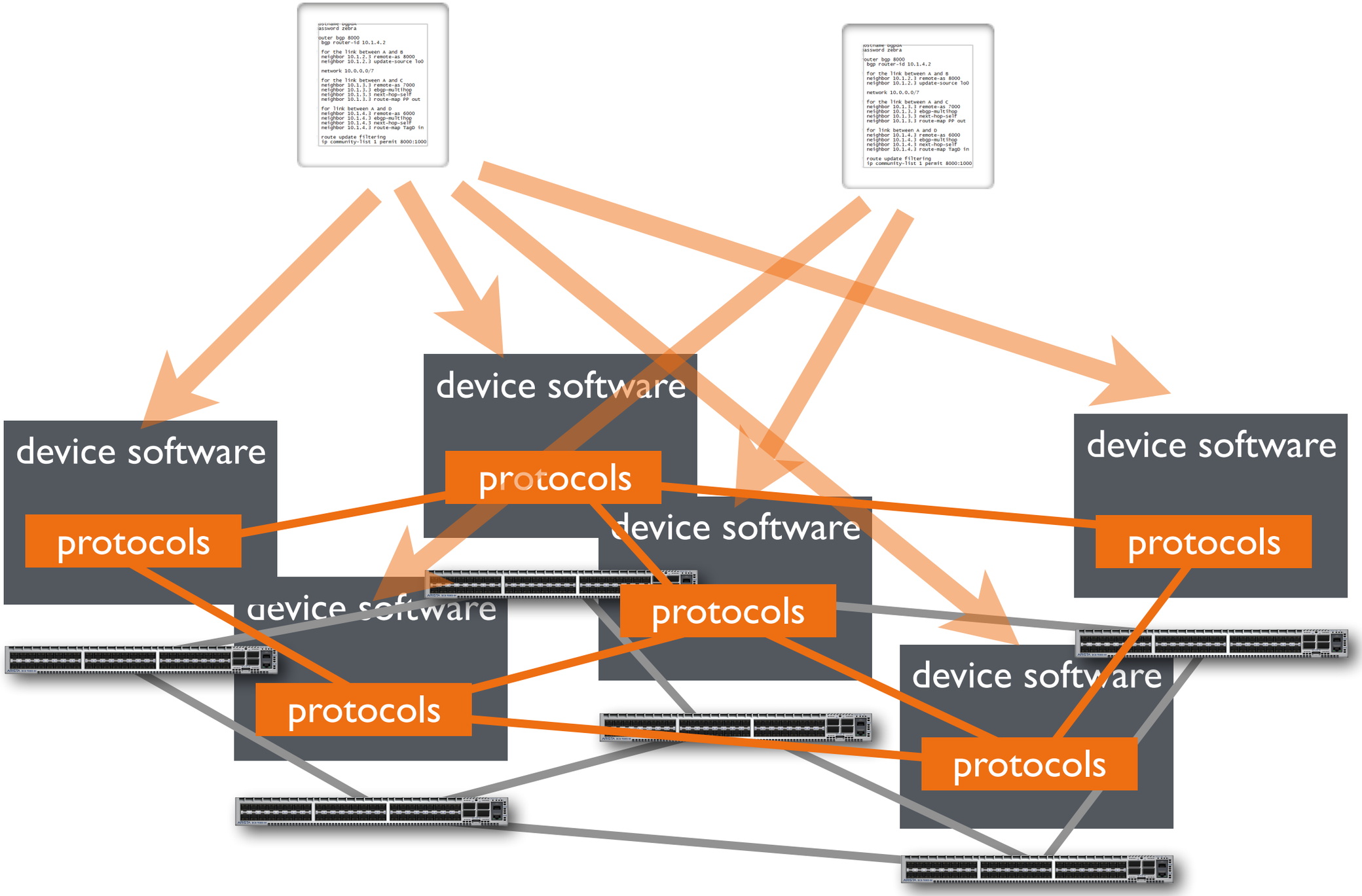
Distributed route computation



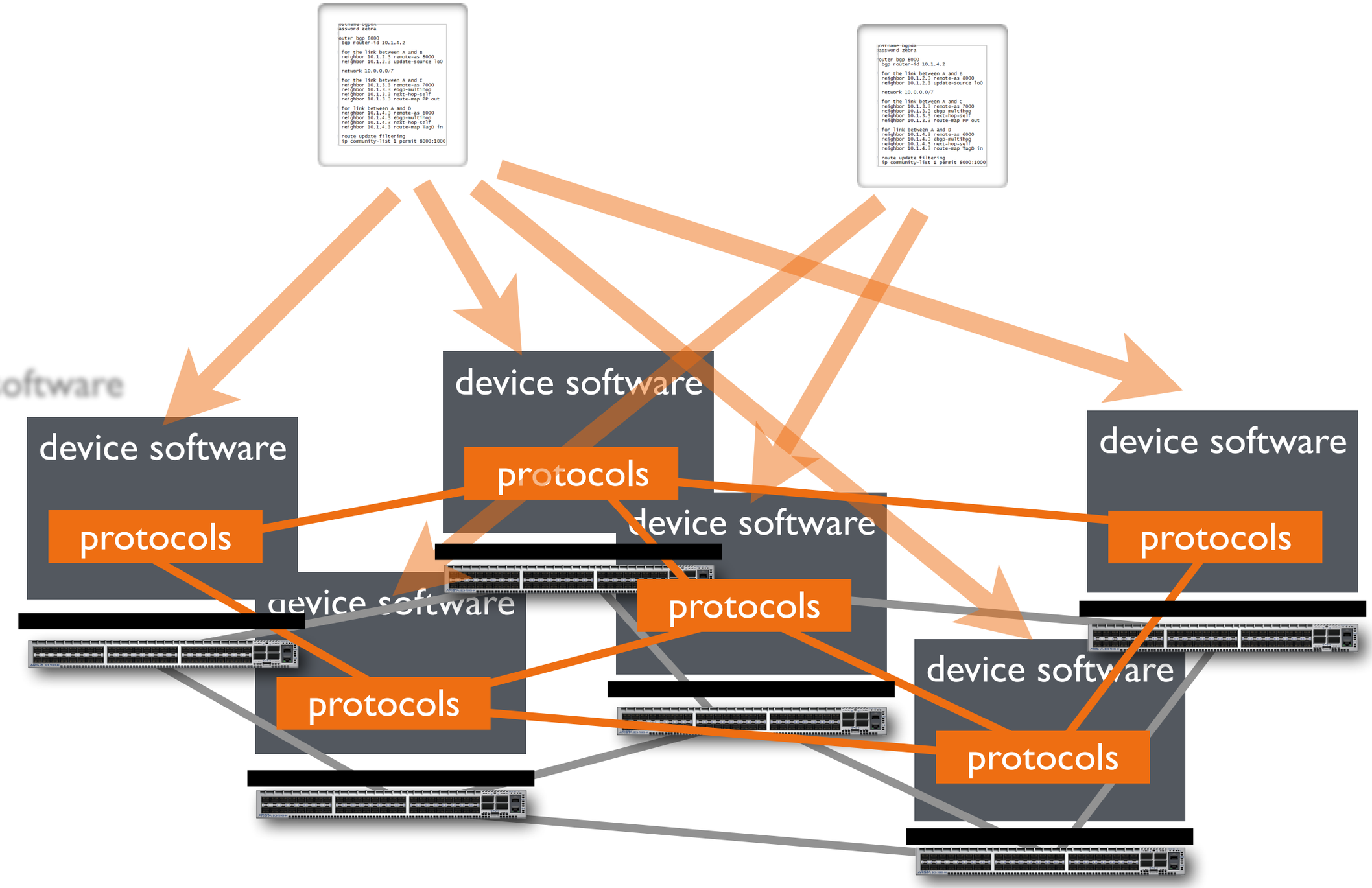
Distributed route computation



Distributed route computation

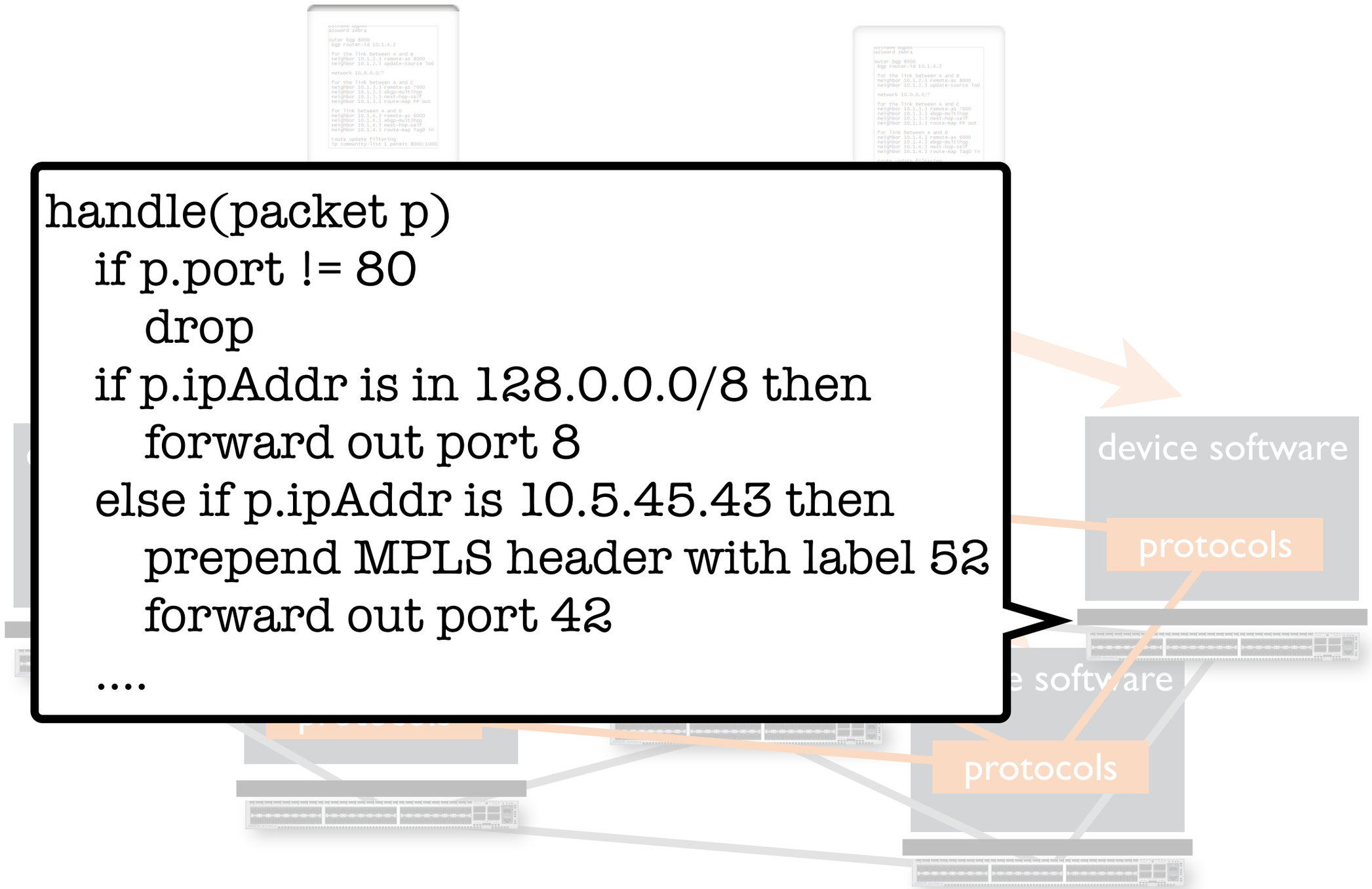


Distributed route computation



Result: data plane state

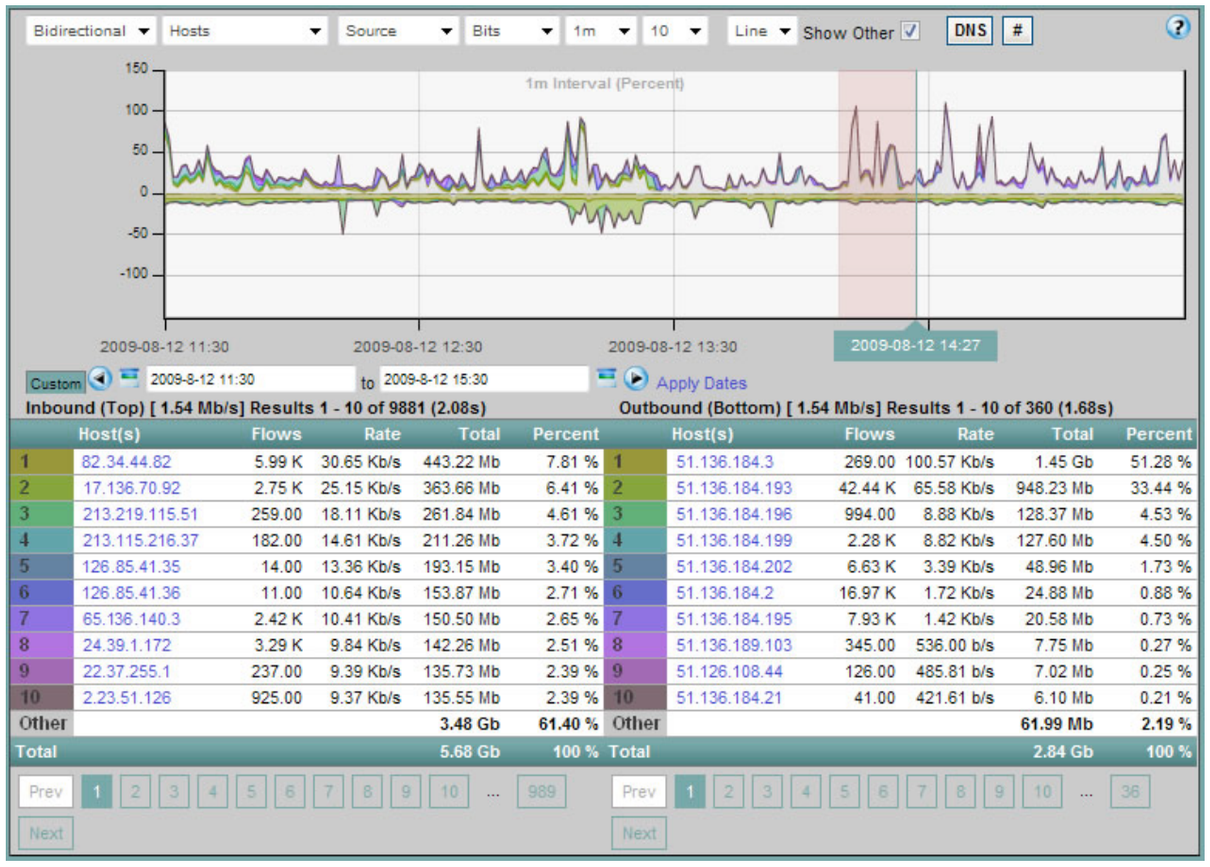
```
handle(packet p)
    if p.port != 80
        drop
    if p.ipAddr is in 128.0.0.0/8 then
        forward out port 8
    else if p.ipAddr is 10.5.45.43 then
        prepend MPLS header with label 52
        forward out port 42
    ....
```



Ensuring correct operations today

Manual spot-checking (pings, traceroutes)

Monitoring of events & flows



	Host(s)	Flows	Rate	Total	Percent
1	82.34.44.82	5.99 K	30.65 Kb/s	443.22 Mb	7.81 %
2	17.136.70.92	2.75 K	25.15 Kb/s	363.66 Mb	6.41 %
3	213.219.115.51	259.00	18.11 Kb/s	261.84 Mb	4.61 %
4	213.115.216.37	182.00	14.61 Kb/s	211.26 Mb	3.72 %
5	126.85.41.35	14.00	13.36 Kb/s	193.15 Mb	3.40 %
6	126.85.41.36	11.00	10.64 Kb/s	153.87 Mb	2.71 %
7	65.136.140.3	2.42 K	10.41 Kb/s	150.50 Mb	2.65 %
8	24.39.1.172	3.29 K	9.84 Kb/s	142.26 Mb	2.51 %
9	22.37.255.1	237.00	9.39 Kb/s	135.73 Mb	2.39 %
10	2.23.51.126	925.00	9.37 Kb/s	135.55 Mb	2.39 %
Other				3.48 Gb	61.40 %
Total				5.68 Gb	100 %

	Host(s)	Flows	Rate	Total	Percent
1	51.136.184.3	269.00	100.57 Kb/s	1.45 Gb	51.28 %
2	51.136.184.193	42.44 K	65.58 Kb/s	948.23 Mb	33.44 %
3	51.136.184.196	994.00	8.88 Kb/s	128.37 Mb	4.53 %
4	51.136.184.199	2.28 K	8.82 Kb/s	127.60 Mb	4.50 %
5	51.136.184.202	6.63 K	3.39 Kb/s	48.96 Mb	1.73 %
6	51.136.184.2	16.97 K	1.72 Kb/s	24.88 Mb	0.88 %
7	51.136.184.195	7.93 K	1.42 Kb/s	20.58 Mb	0.73 %
8	51.136.189.103	345.00	536.00 b/s	7.75 Mb	0.27 %
9	51.126.108.44	126.00	485.81 b/s	7.02 Mb	0.25 %
10	51.136.184.21	41.00	421.61 b/s	6.10 Mb	0.21 %
Other				61.99 Mb	2.19 %
Total				2.84 Gb	100 %

Screenshot from Scrutinizer NetFlow & sFlow analyzer, snmp.co.uk/scrutinizer/

VERIFLOW

Networks are complex

– Survey of network operators
[Kim, Reich, Gupta, Shahbaz, Feamster, Clark, USENIX NSDI 2015]

Networks are complex

89%
of operators never sure that config changes are bug-free

– Survey of network operators
[Kim, Reich, Gupta, Shahbaz, Feamster, Clark, USENIX NSDI 2015]

Networks are complex

89%
of operators never sure that config changes are bug-free

82%
concerned that changes would cause problems with existing functionality

– Survey of network operators
[Kim, Reich, Gupta, Shahbaz, Feamster, Clark, USENIX NSDI 2015]

VERIFLOW

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

VERIFLOW

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

"Host A should be connected to host B."

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

“Host A should be connected to host B.”

“Host A should not be able to reach service B on any server.”

VERIFLOW

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

“Host A should be connected to host B.”

“Host A should not be able to reach service B on any server.”

“No packet should fall into a loop.”

VERIFLOW

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

"Host A should be connected to host B."

"Host A should not be able to reach service B on any server."

"No packet should fall into a loop."

"All packets should follow shortest paths."

VERIFLOW

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

VERIFLOW

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

Configuration

VERIFLOW

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

Configuration

Control software

VERIFLOW

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

- Configuration
- Control software
- Data plane state

VERIFLOW

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

- Configuration
- Control software
- Data plane state
- Packet processing

VERIFLOW

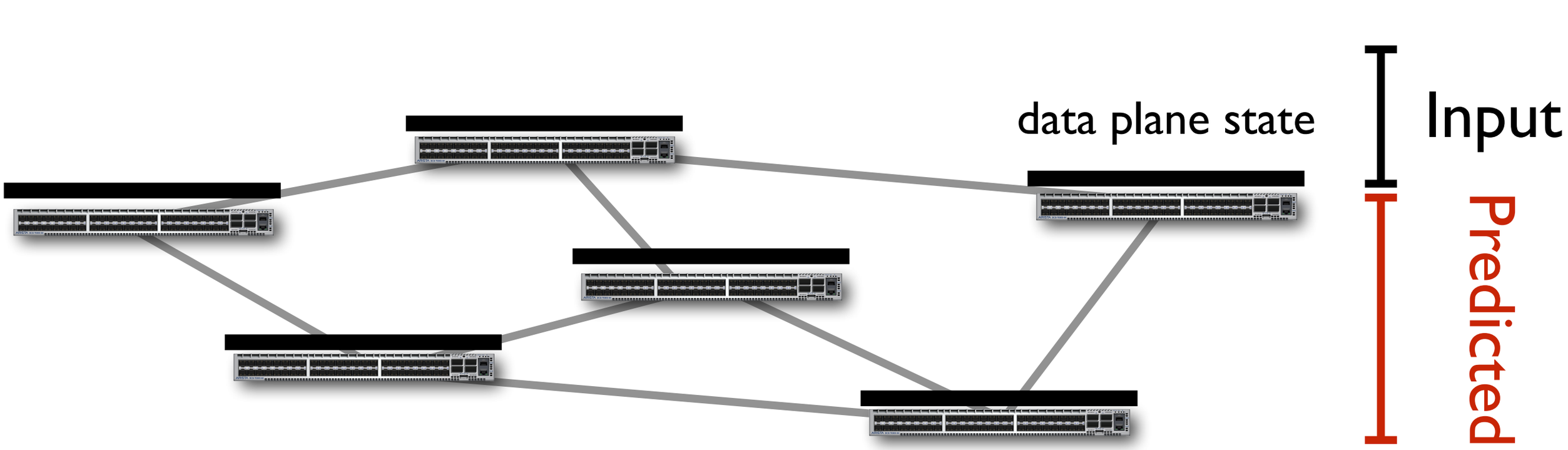
Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

Configuration	Configuration verification
Control software	Controller verification & verifiable control languages
Data plane state	Data plane verification
Packet processing	Software switch verification

VERIFLOW

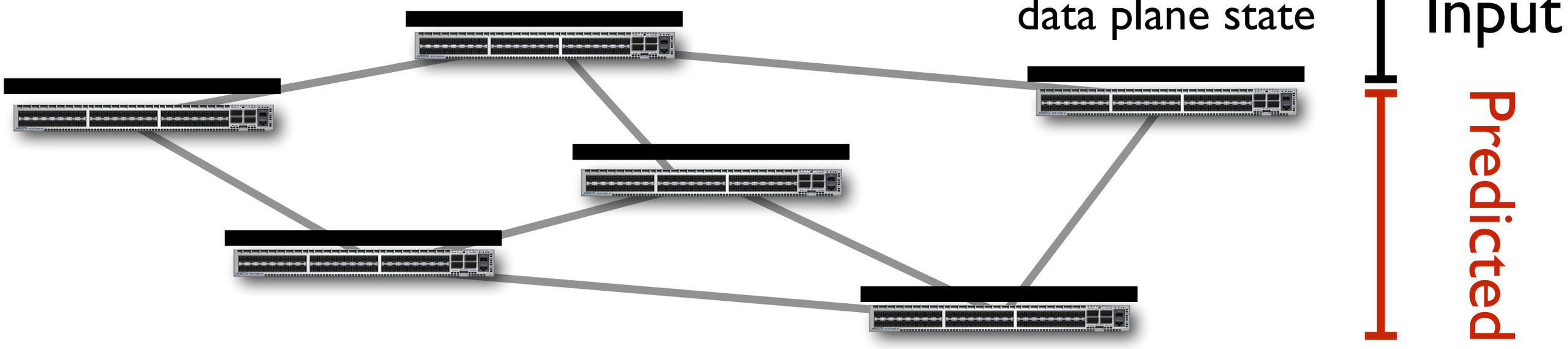
Data plane verification



VERIFLOW

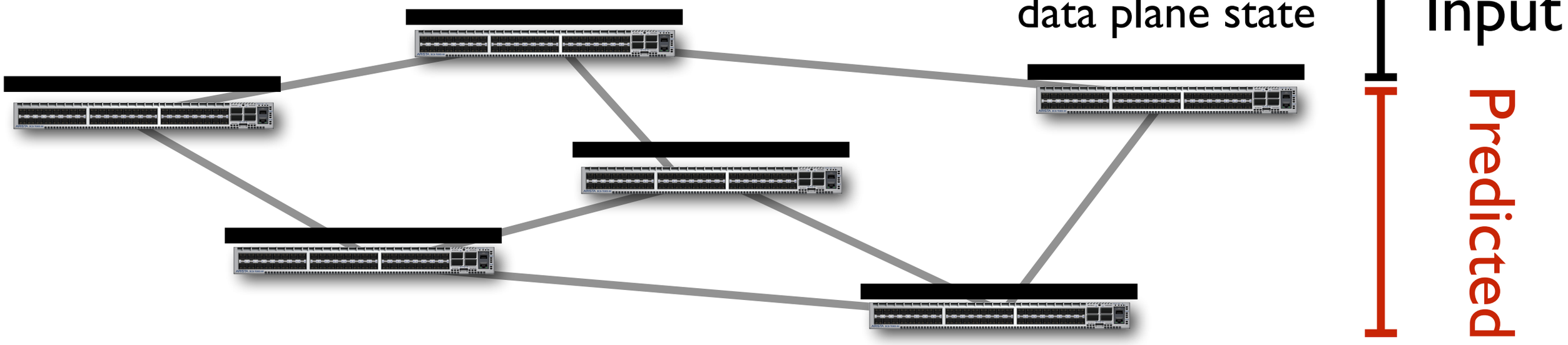
Data plane verification

Verify the network
as close as possible
to its actual
behavior



Data plane verification

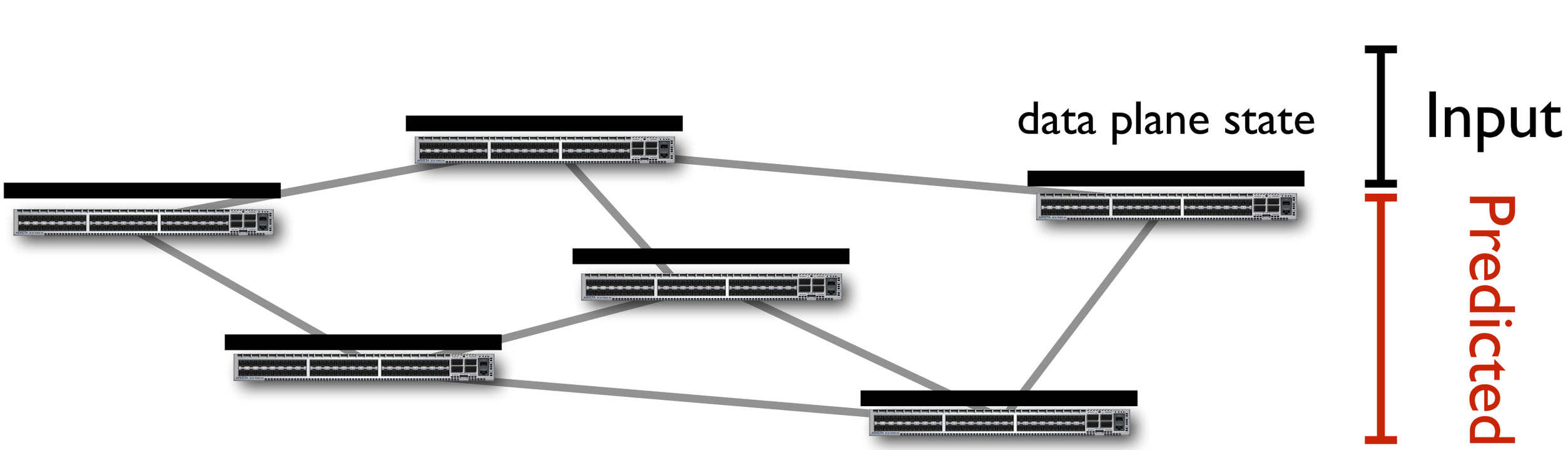
Verify the network as close as possible to its actual behavior



Data plane verification

Verify the network as close as possible to its actual behavior

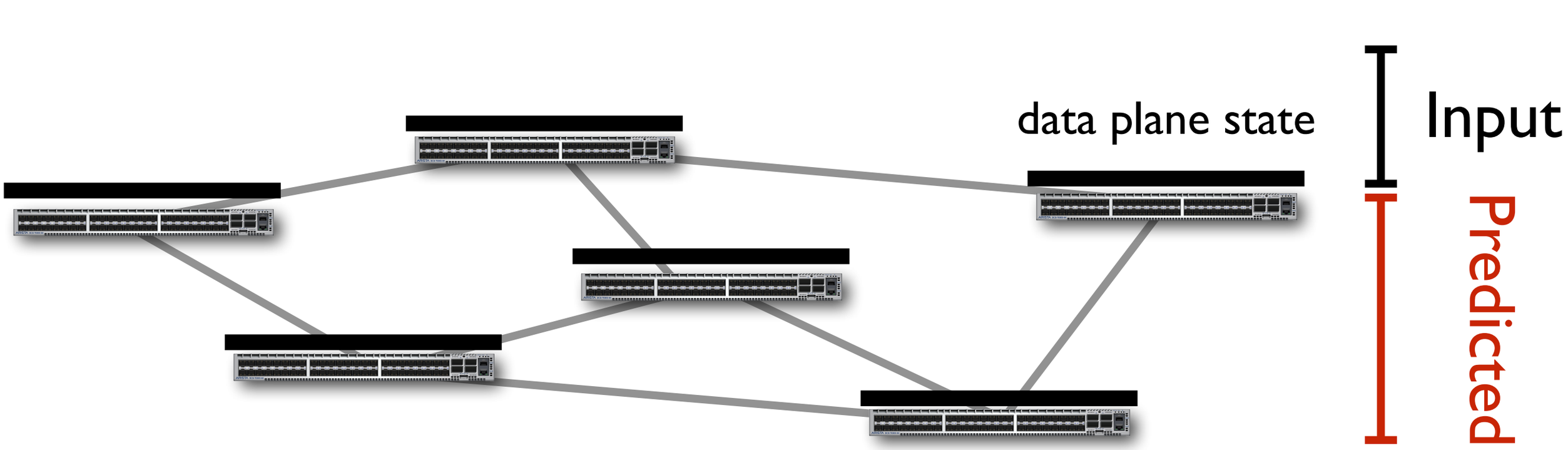
- Insensitive to control protocols



Data plane verification

Verify the network as close as possible to its actual behavior

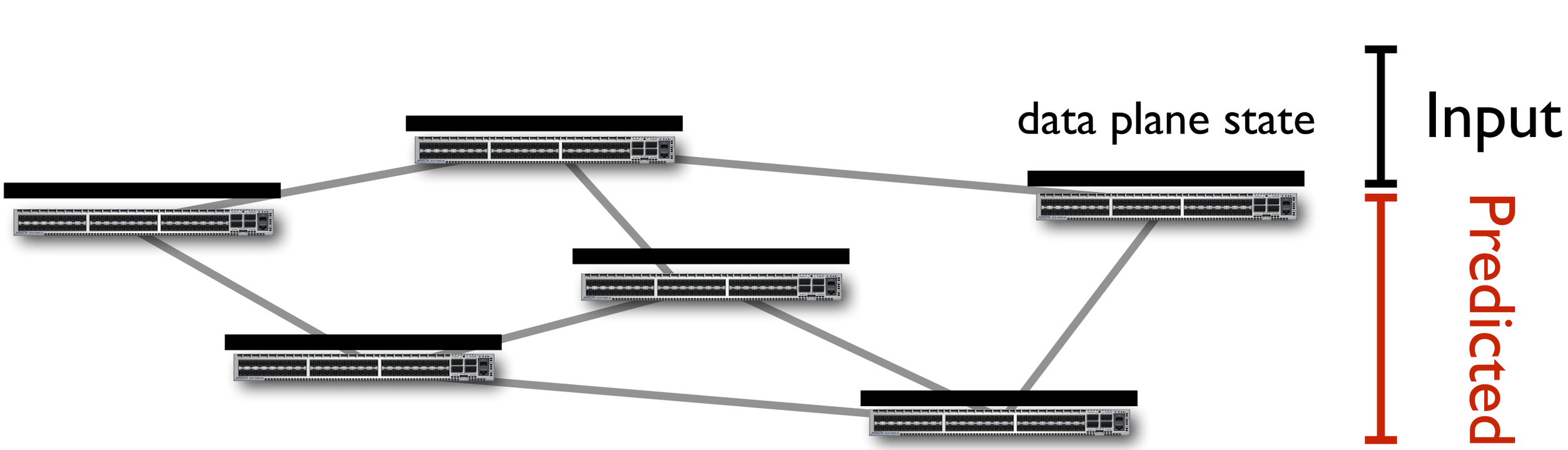
- Insensitive to control protocols
- Accurate model



Data plane verification

Verify the network as close as possible to its actual behavior

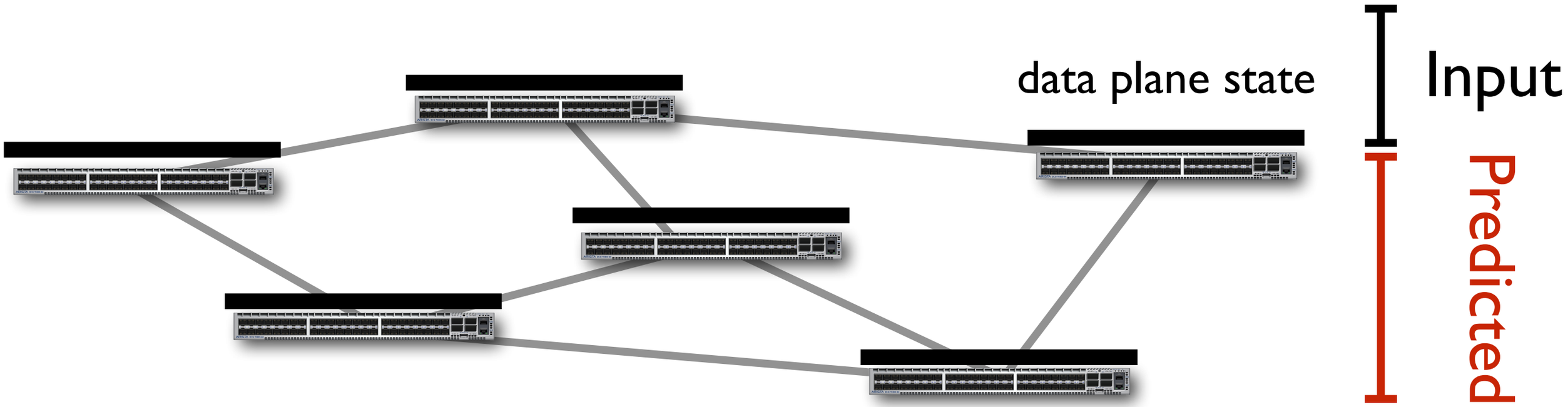
- Insensitive to control protocols
- Accurate model
- *Checks current snapshot*



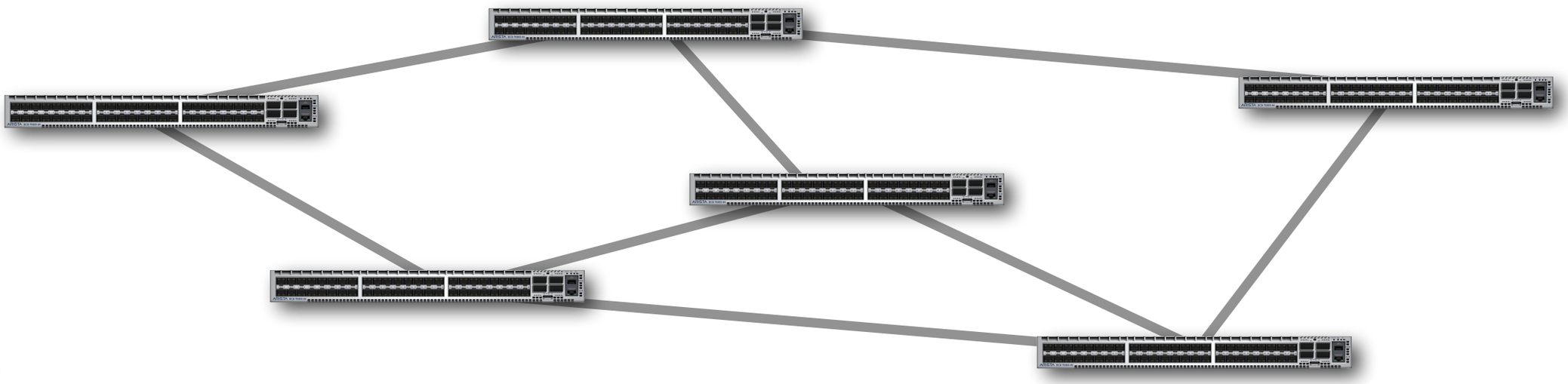
Data plane verification

Verify the network as close as possible to its actual behavior

- Insensitive to control protocols
- Accurate model
- *Checks current snapshot*
  - but can be foundation for config verification



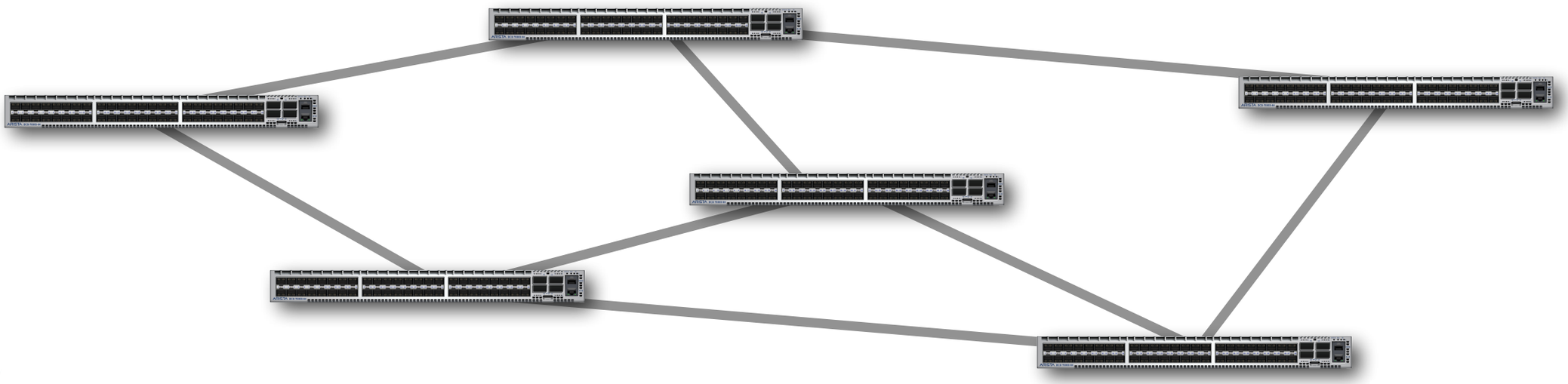
Data plane verification architecture



VERIFLOW

Data plane verification architecture

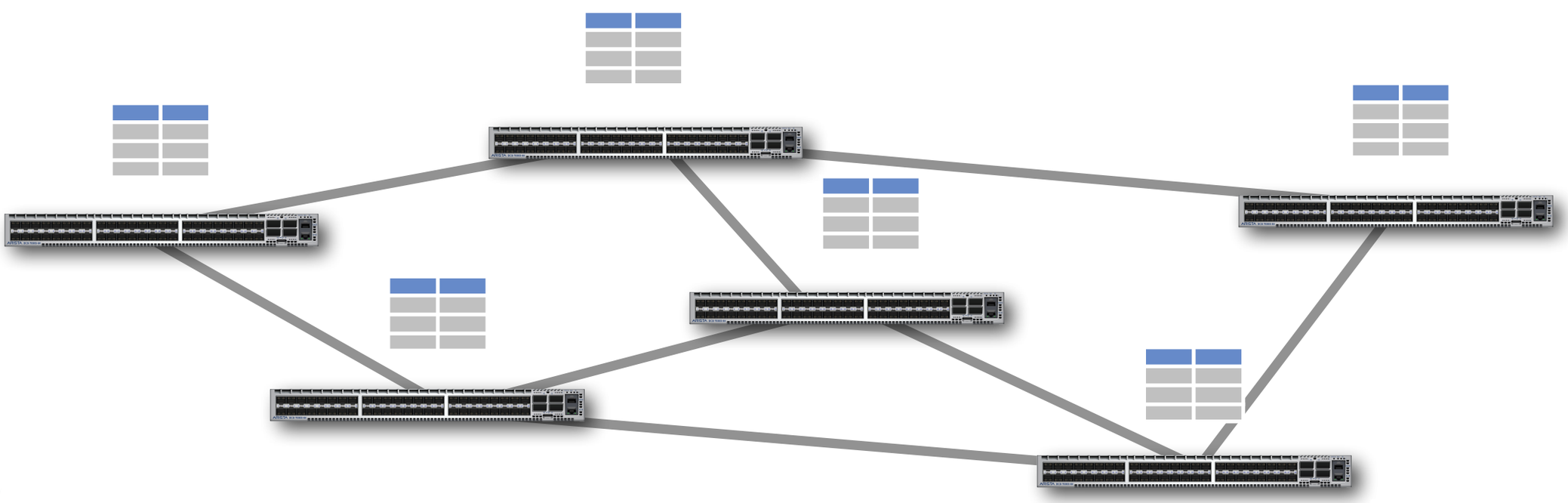
Verifier



VERIFLOW

Data plane verification architecture

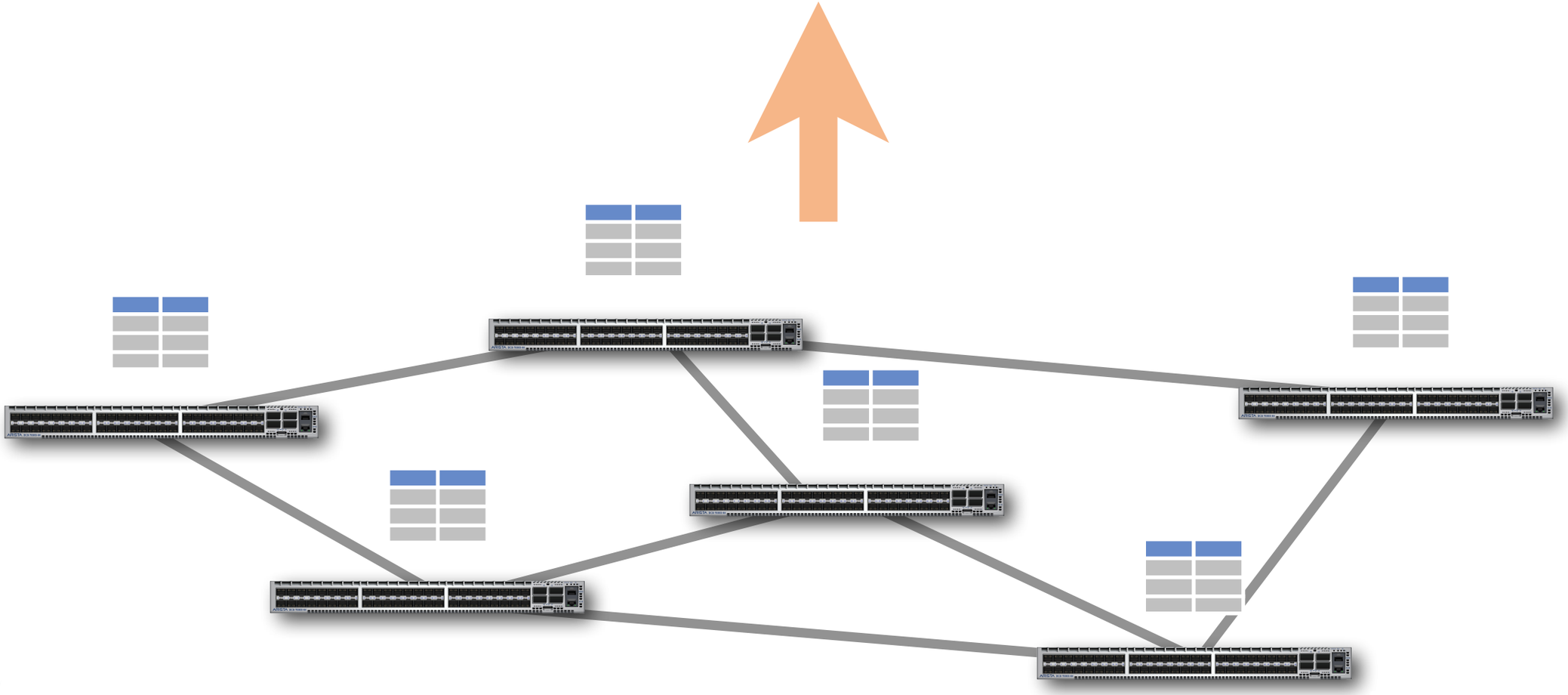
Verifier



VERIFLOW

Data plane verification architecture

Verifier

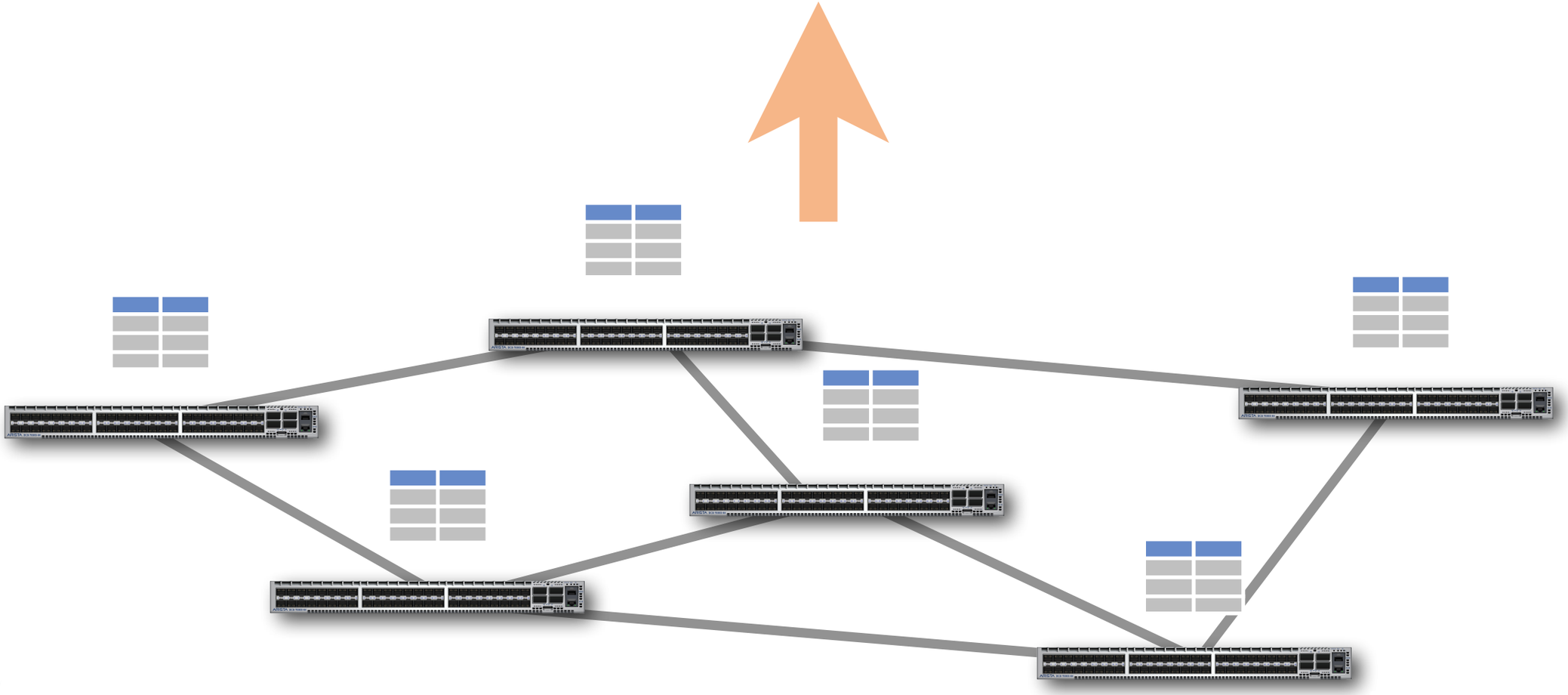


VERIFLOW

Data plane verification architecture

"Can any packet starting at A reach B?"

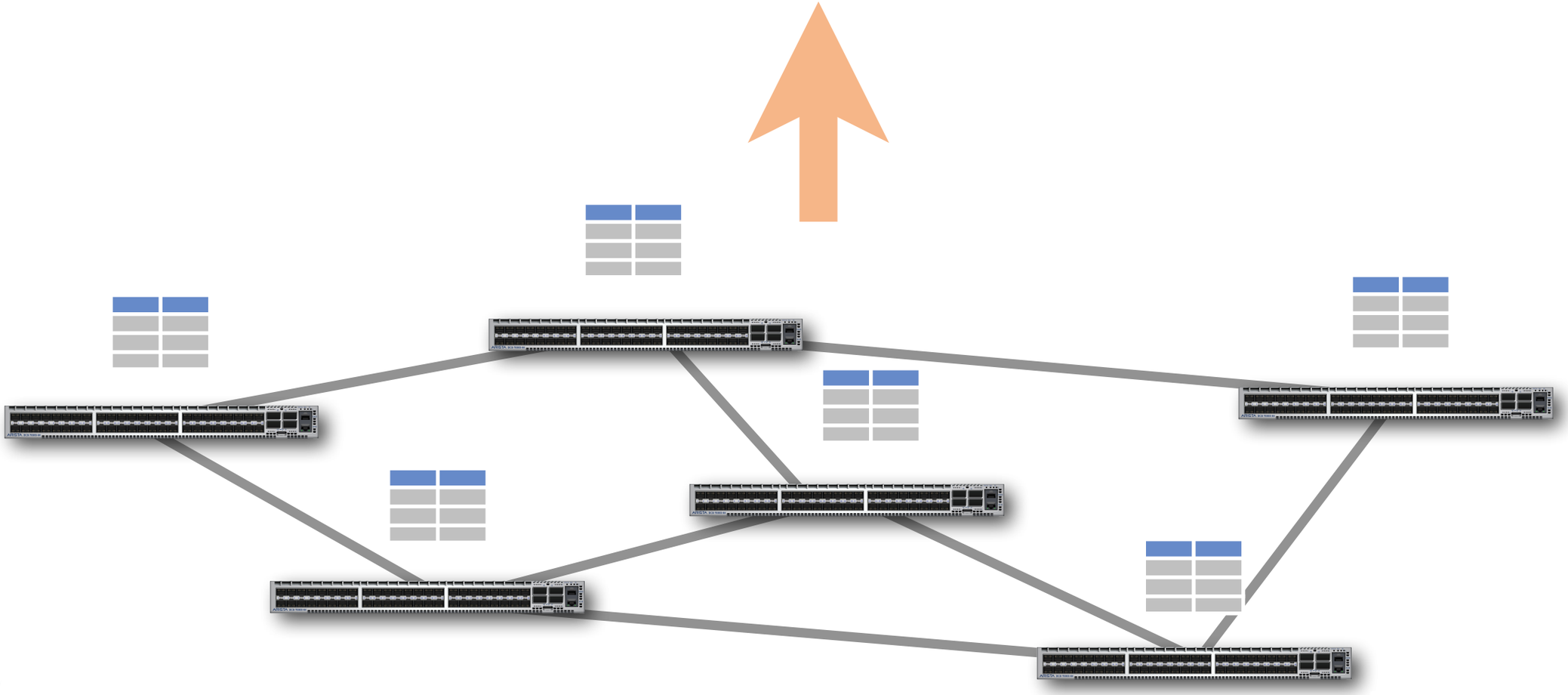
Verifier



Data plane verification architecture

"Can any packet starting at A reach B?"

Verifier

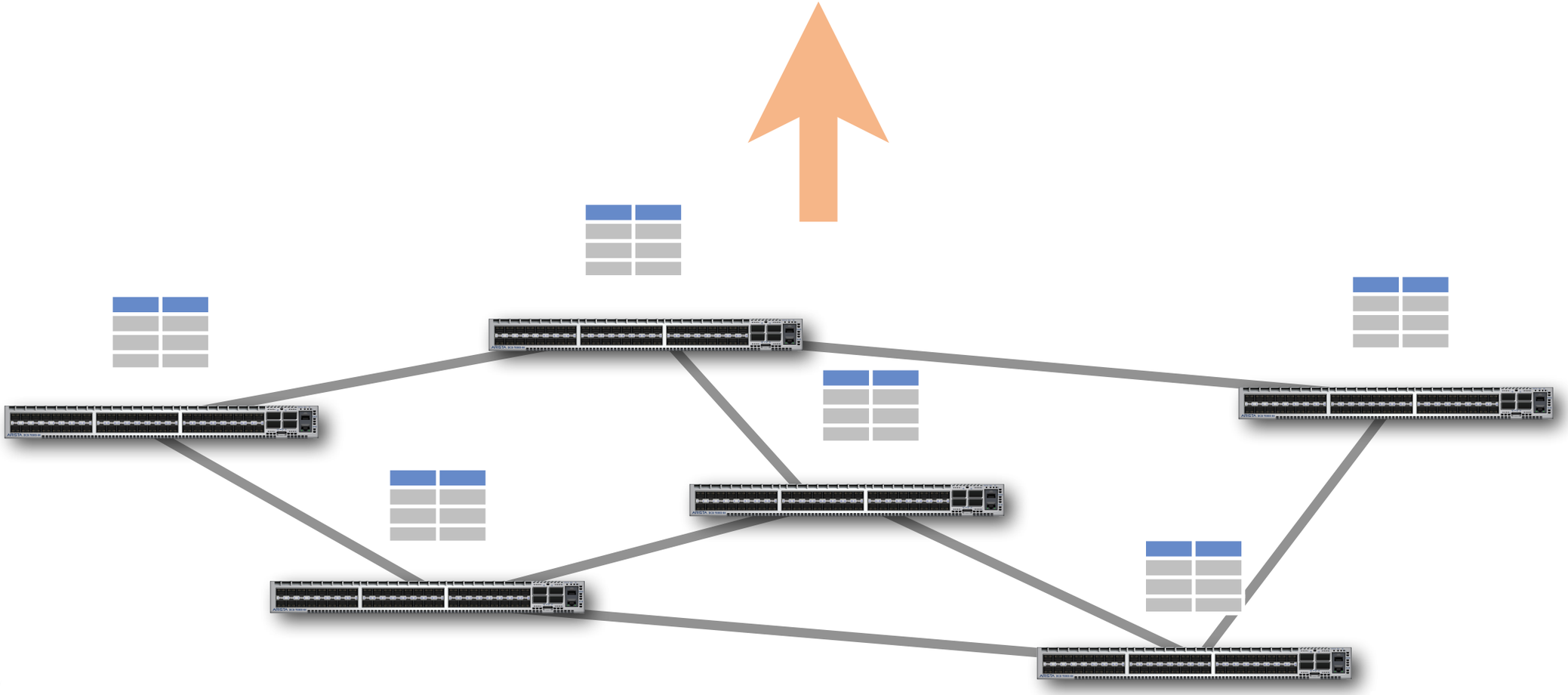


Data plane verification architecture

"Can any packet starting at A reach B?"

Diagnosis

Verifier



A little calculation…

A little calculation…

theoretical packets

A little calculation…

theoretical packets

$$= 2^{(\#\text{header bits})} \times \#\text{injection points}$$

A little calculation…

theoretical packets

$= 2^{(\#\text{header bits})} \times \#\text{injection points}$

$= 2^{(18 \text{ byte ethernet} + 20 \text{ byte IPv4})} \times 10{,}000 \text{ ports}$

$= 3.25 \times 10^{95}$ possible packets

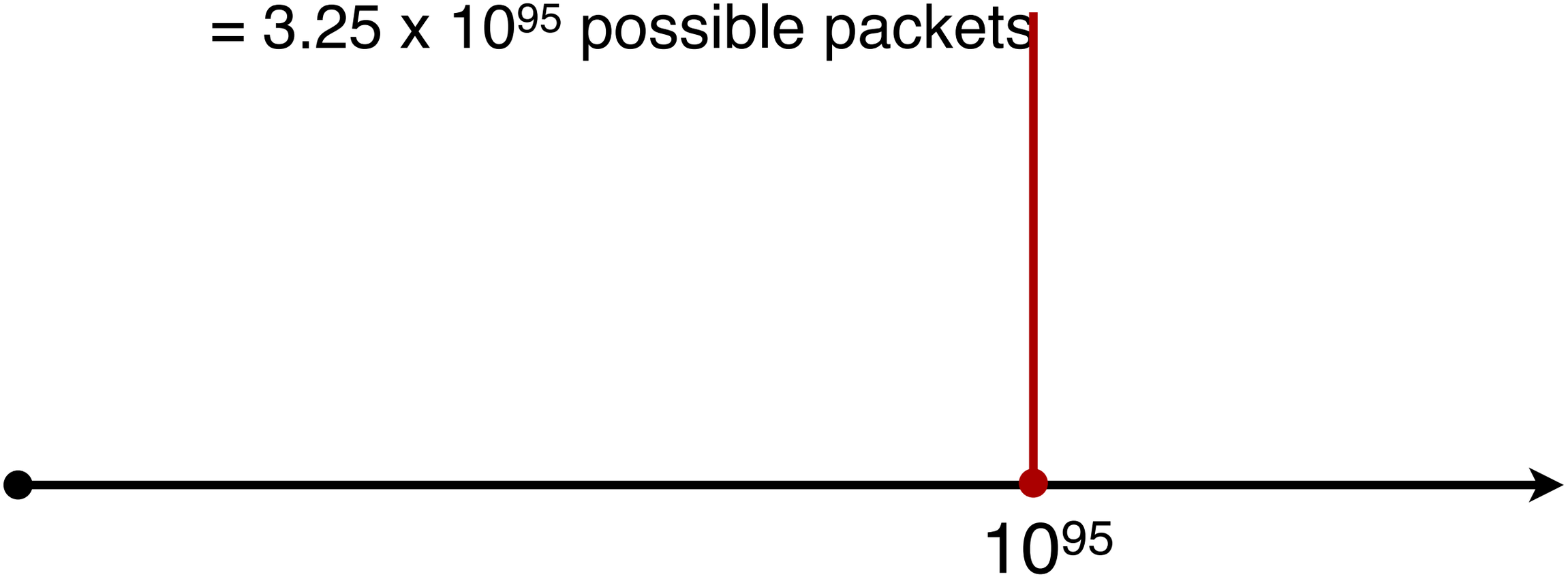
A little calculation…

theoretical packets

$= 2^{(\#\text{header bits})} \times \#\text{injection points}$

$= 2^{(18 \text{ byte ethernet} + 20 \text{ byte IPv4})} \times 10{,}000 \text{ ports}$

$= 3.25 \times 10^{95}$ possible packets



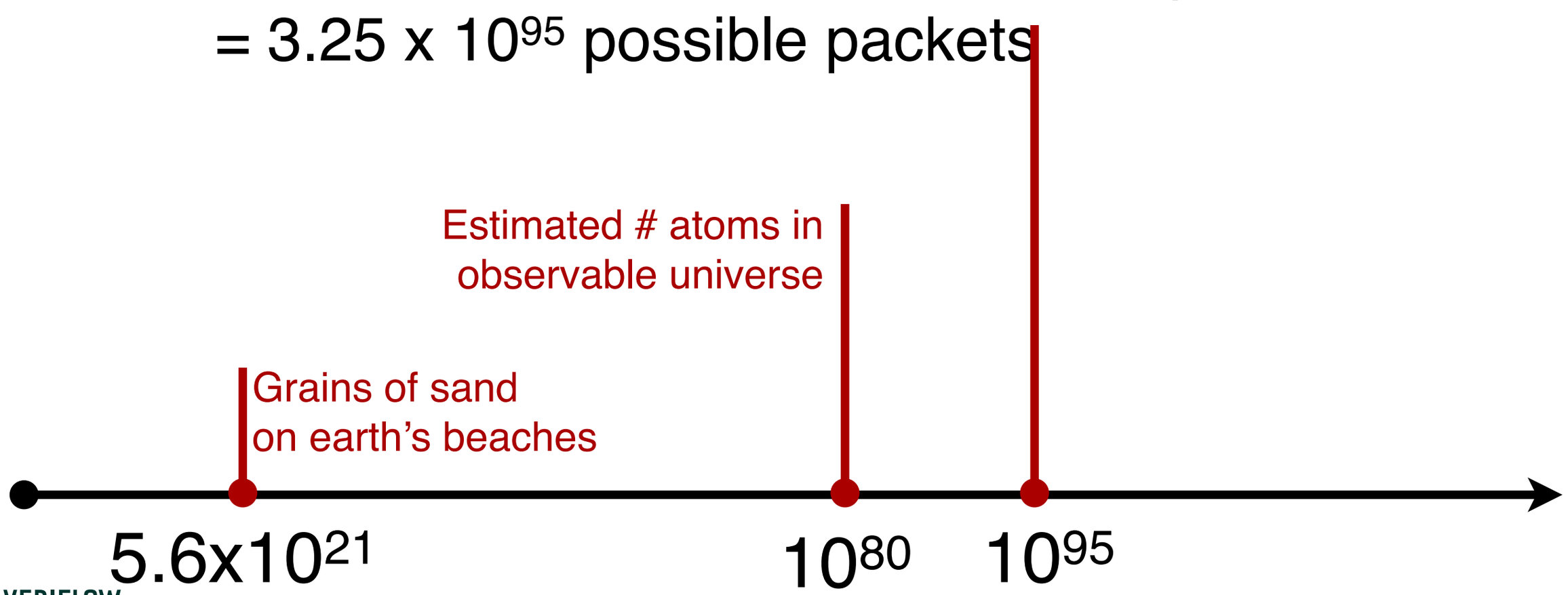
A little calculation…

theoretical packets

$= 2^{(\#\text{header bits})} \times \#\text{injection points}$

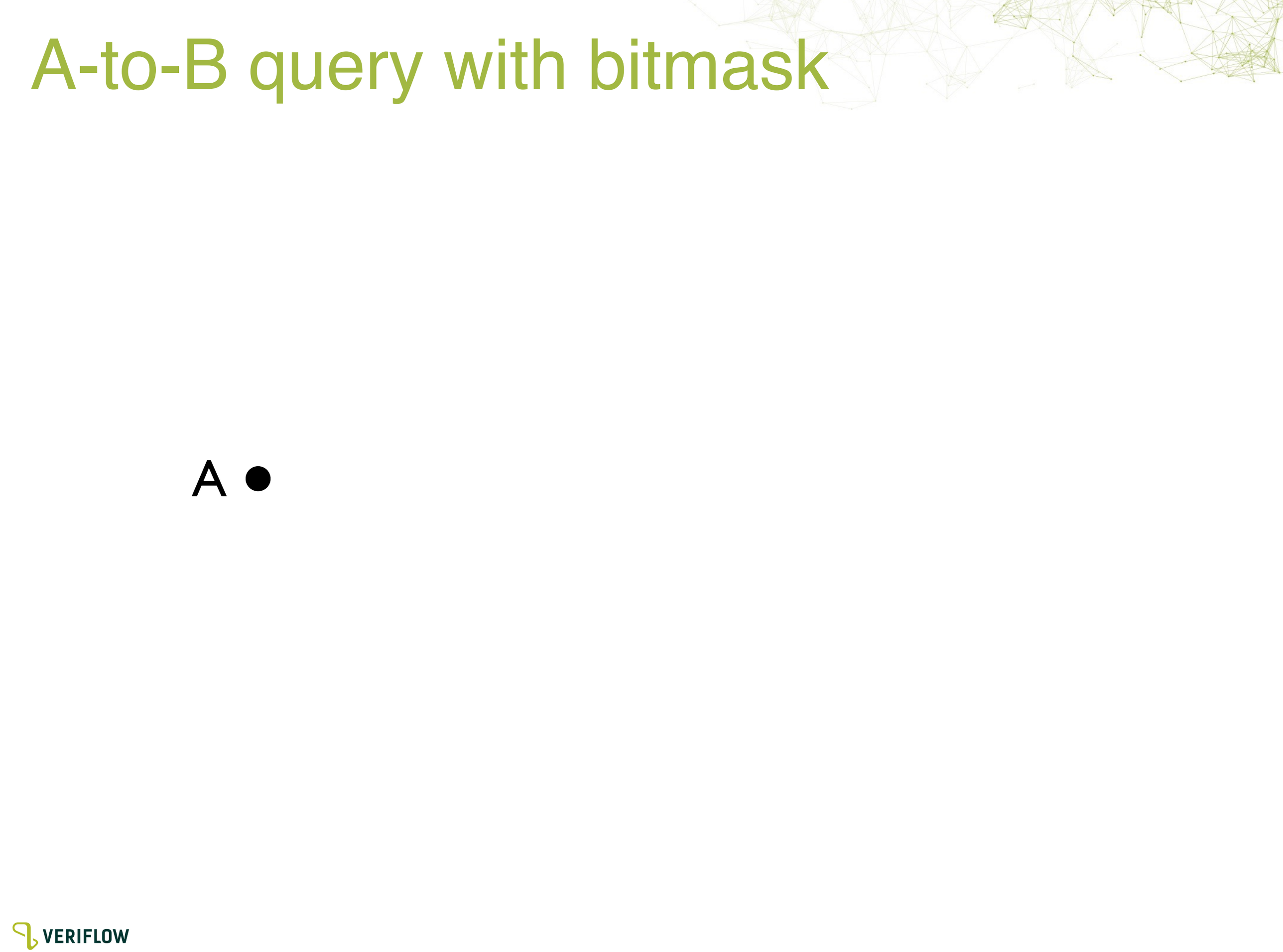
$= 2^{(18 \text{ byte ethernet} + 20 \text{ byte IPv4})} \times 10{,}000 \text{ ports}$

$= 3.25 \times 10^{95}$ possible packets



A-to-B query with bitmask

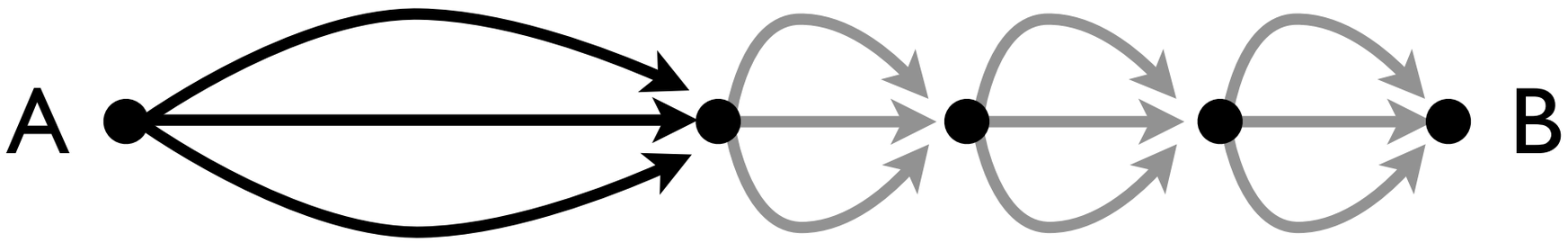
A-to-B query with bitmask



A-to-B query with bitmask

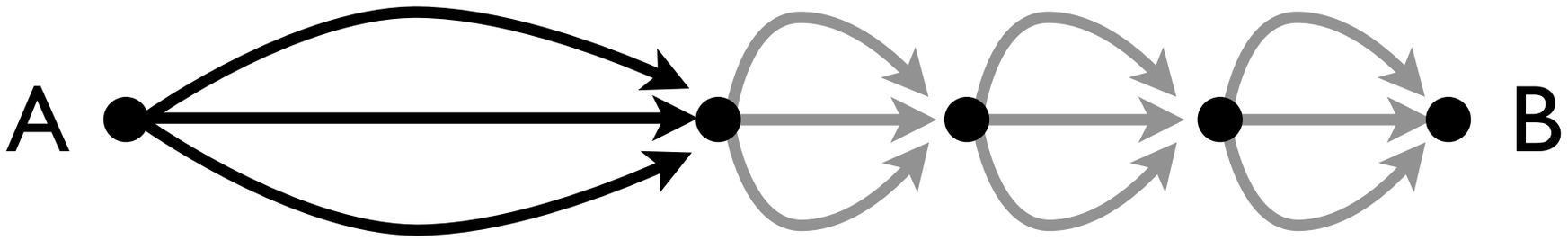
A ● ● B

A-to-B query with bitmask



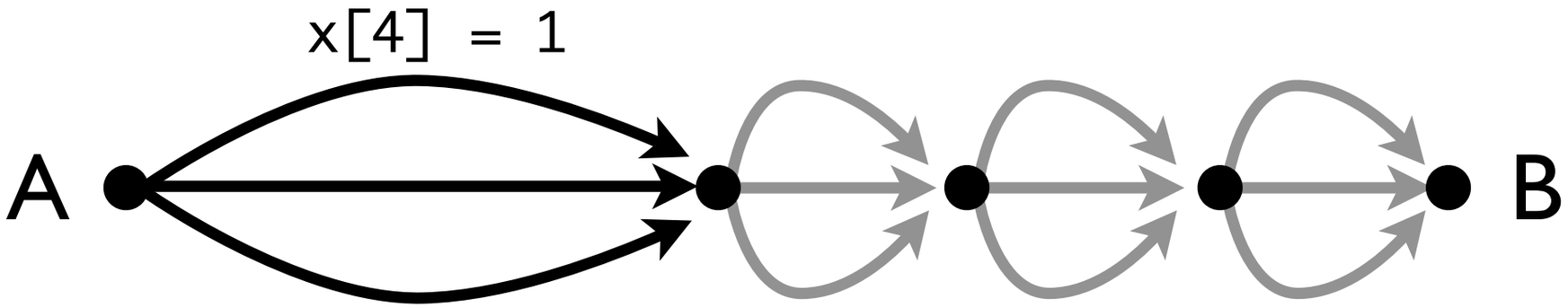
A-to-B query with bitmask

Packet: `x[0] x[1] x[2] … x[n]`



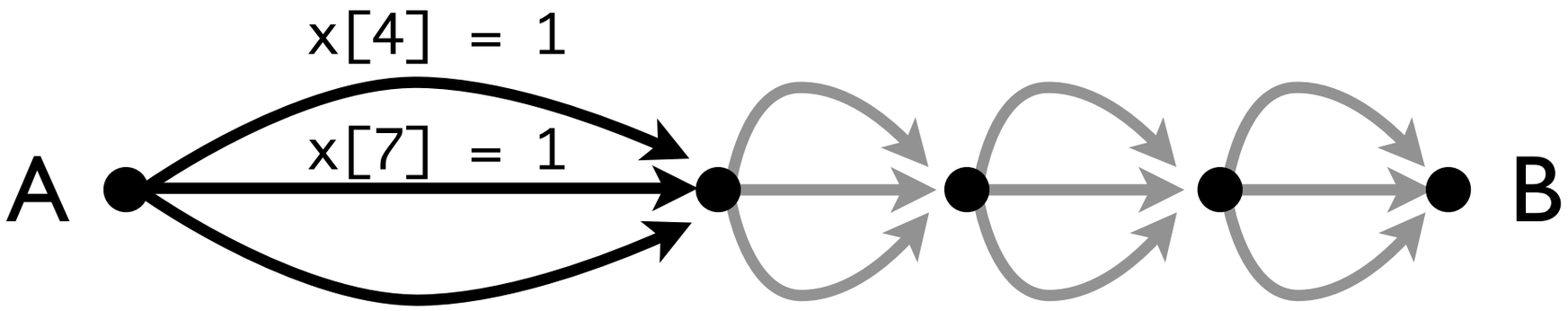
A-to-B query with bitmask

Packet: `x[0] x[1] x[2] … x[n]`



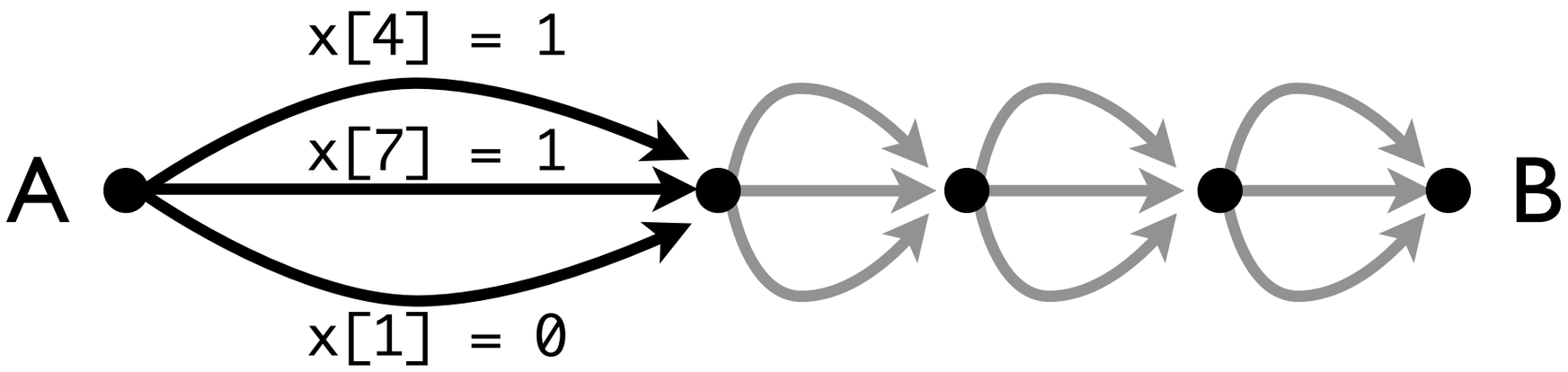
A-to-B query with bitmask

Packet: `x[0] x[1] x[2] … x[n]`



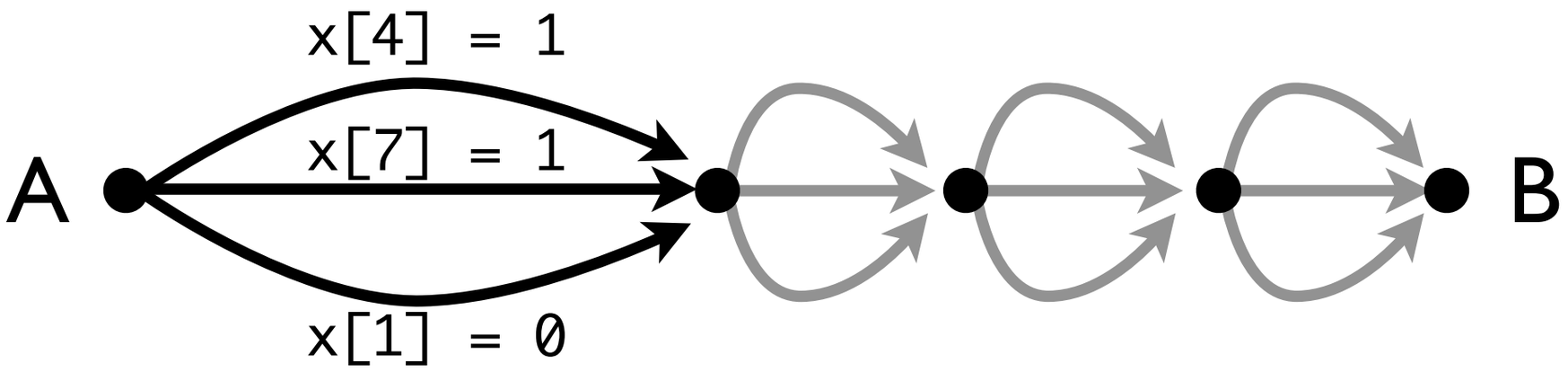
A-to-B query with bitmask

Packet: `x[0] x[1] x[2] … x[n]`



A-to-B query with bitmask

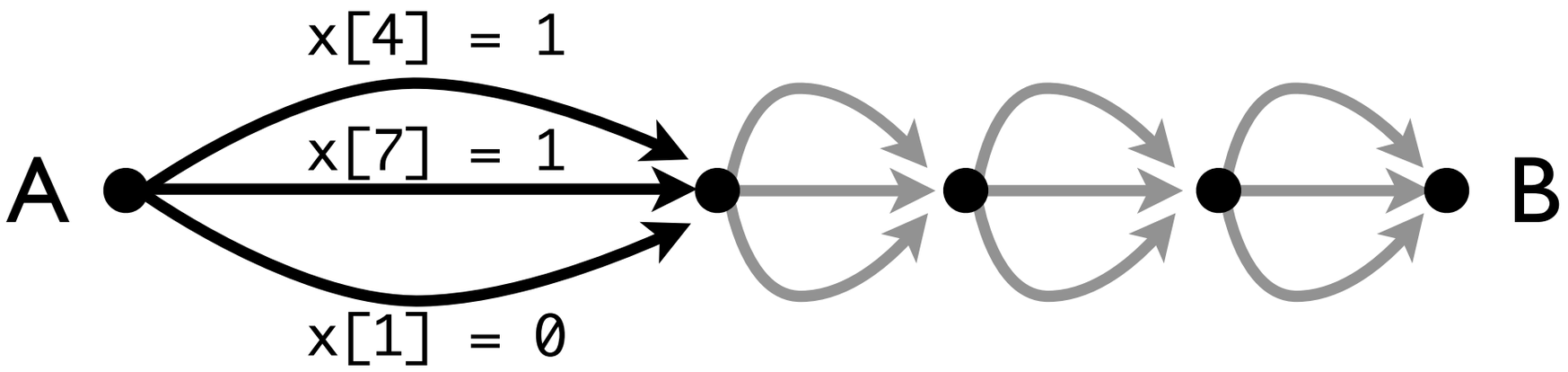
Packet: `x[0] x[1] x[2] … x[n]`



$$(x_4 \vee x_7 \vee \bar{x_1}) \wedge (\ldots) \wedge (\ldots) \wedge (\ldots)$$

A-to-B query with bitmask

Packet: `x[0] x[1] x[2] … x[n]`



$$(x_4 \vee x_7 \vee \bar{x_1}) \wedge (\ldots) \wedge (\ldots) \wedge (\ldots)$$

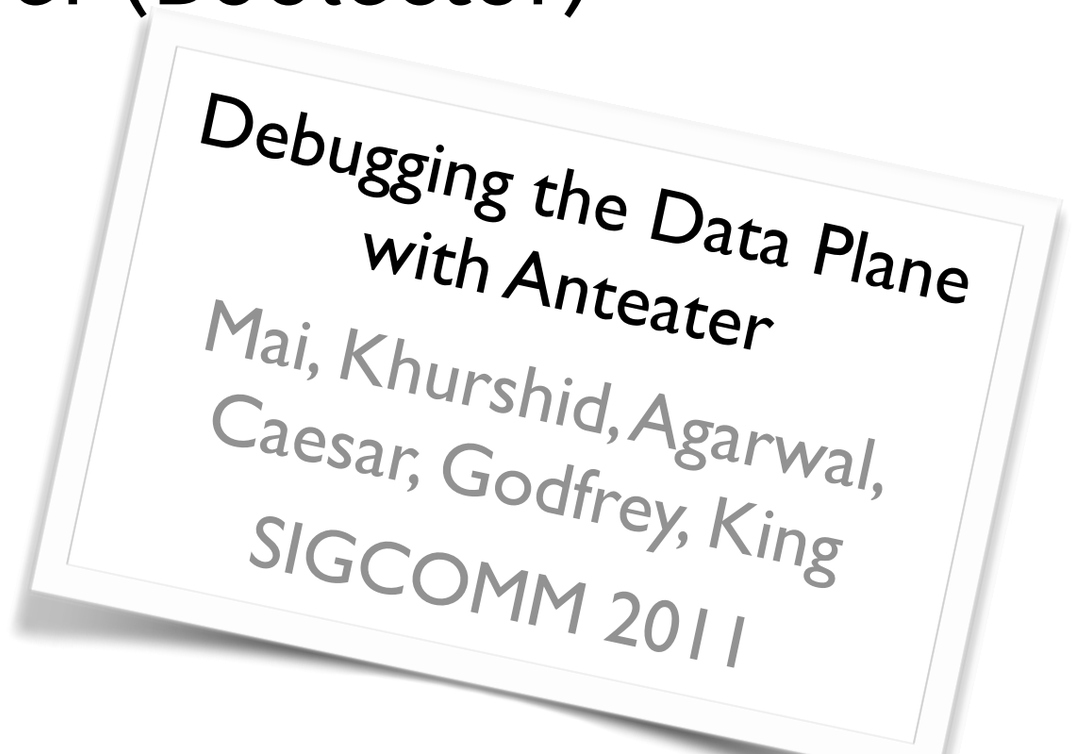
NP-complete!

Anteater's solution

Express data plane and invariants as SAT

- …up to some max # hops
- Dynamic programming to deal with exponential number of paths
- Model packet transformations with vector of packet "versions" & constraints across versions

Check with off-the-shelf SAT solver (Boolector)



Experiences with real network

Evaluated Anteater with operational network

- ~178 routers supporting >70,000 machines
- Predominantly OSPF, also uses BGP and static routing
- 1,627 FIB entries per router (mean)
- State collected using operator's SNMP scripts

Revealed 23 violations of 3 invariants in 2 hours

	Loop	Packet loss	Consistency
Being fixed	9	0	0
Stale config.	0	13	1
Total alerts	9	17	2

CAN WE VERIFY NETWORKS IN REAL TIME?

VeriFlow: Verifying Network-Wide Invariants in Real Time

Khurshid, Zou, Zhou, Caesar, Godfrey

HotSDN'12 best paper, NSDI'13

Not so simple

Not so simple

Challenge #1: Obtaining real time view

Not so simple

Challenge #1: Obtaining real time view

Challenge #2: Verify quickly

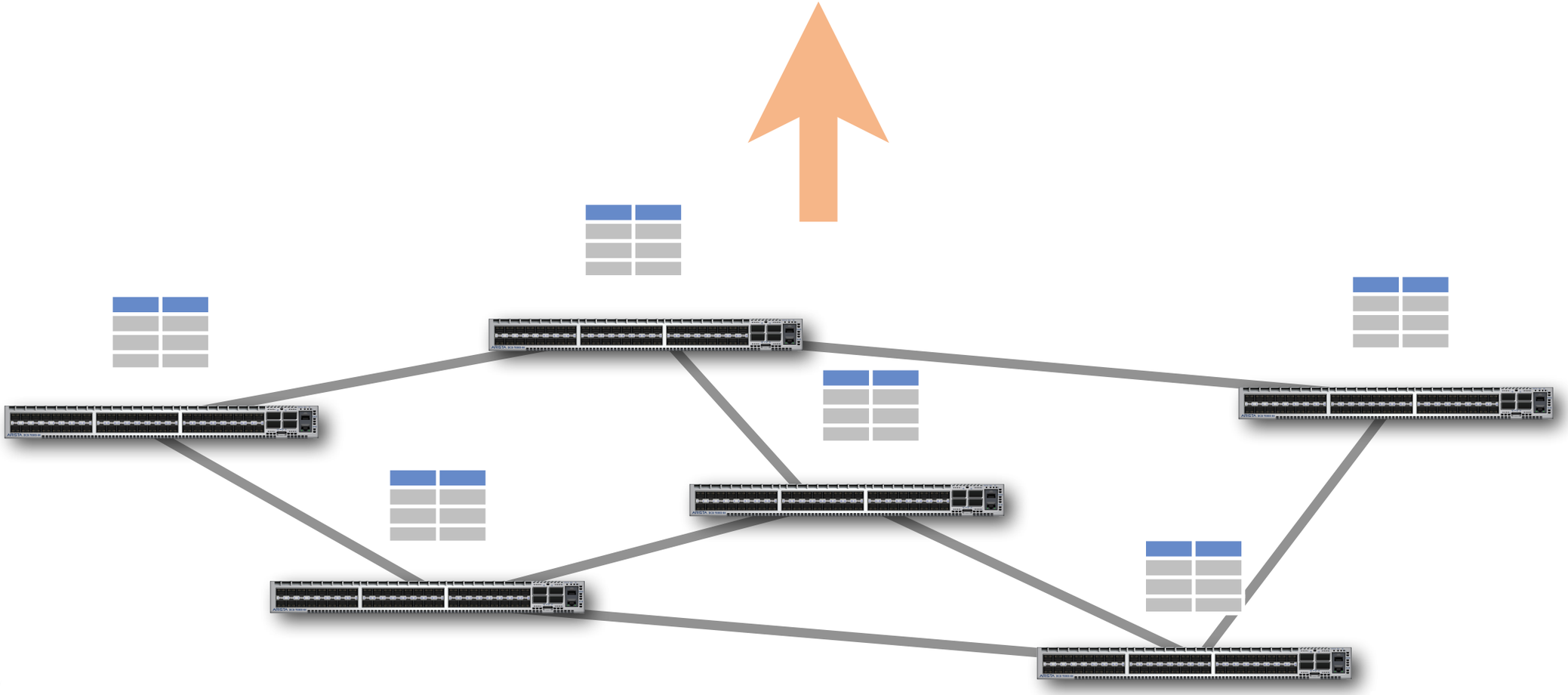
Architecture

"Service S reachable only through firewall?"

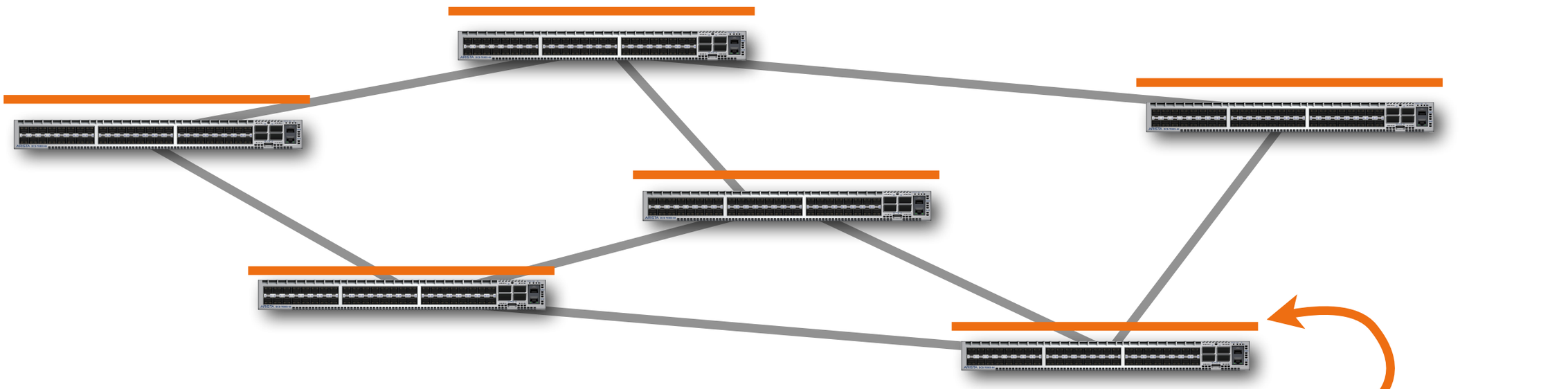
"Is segment isolated?"

Verifier

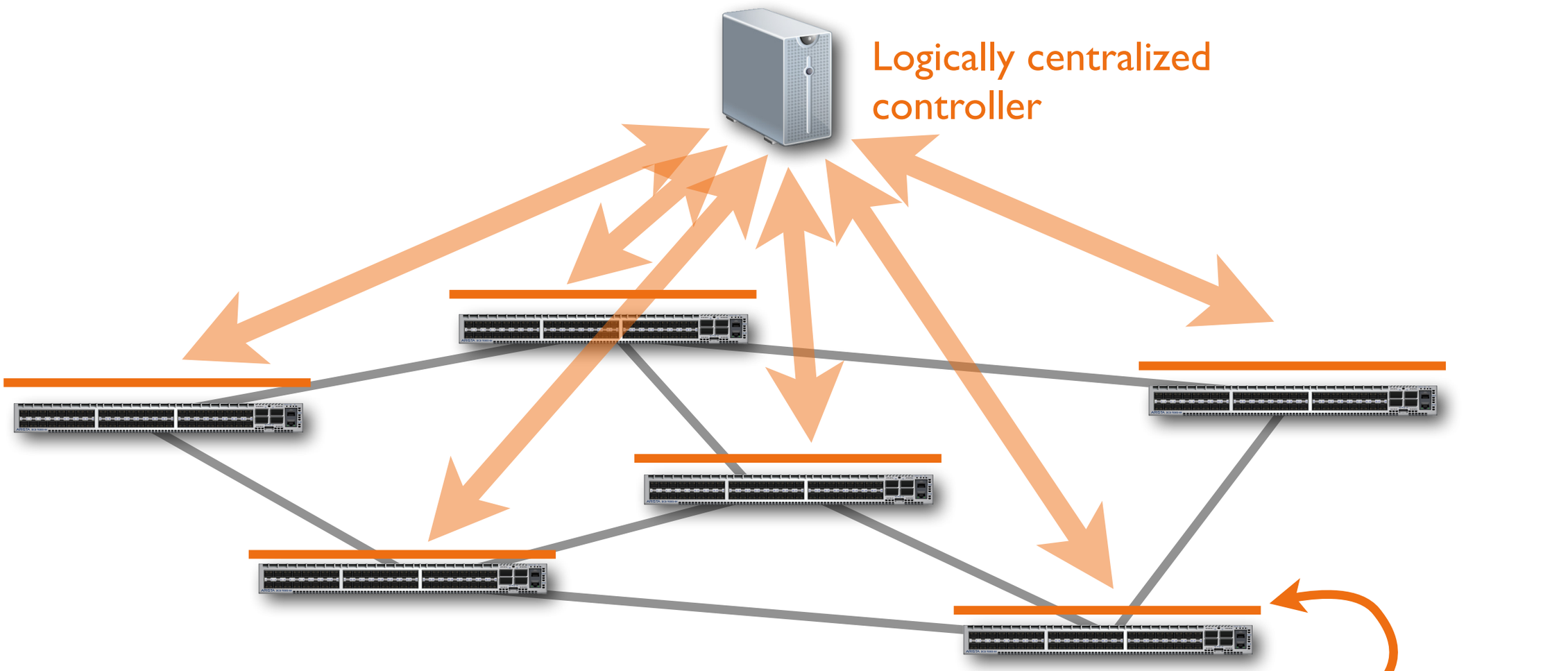
Diagnosis



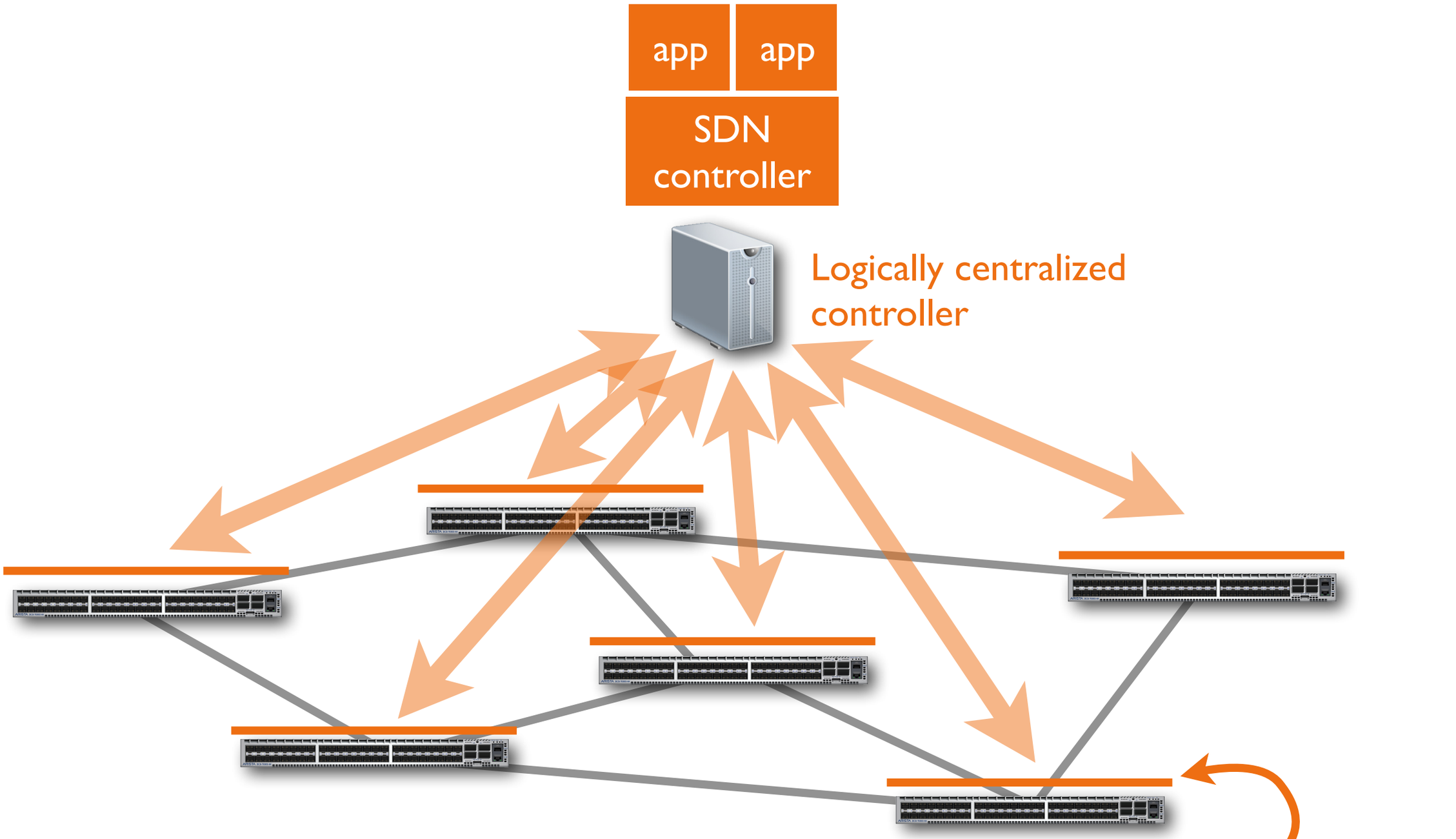
VeriFlow [NSDI'13] architecture



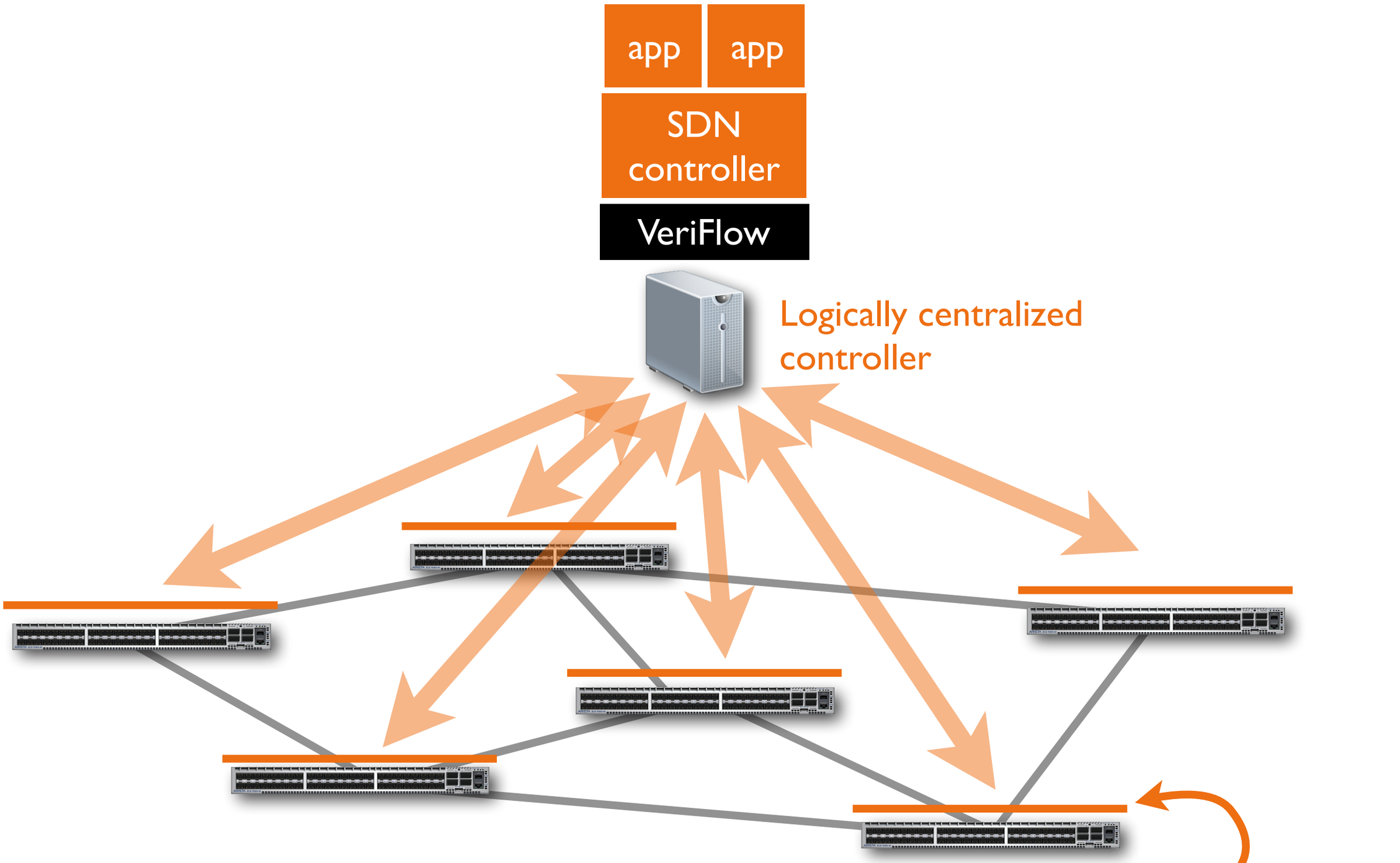
VeriFlow [NSDI'13] architecture



VeriFlow [NSDI'13] architecture



VeriFlow [NSDI'13] architecture



Verifying invariants quickly

Overview:

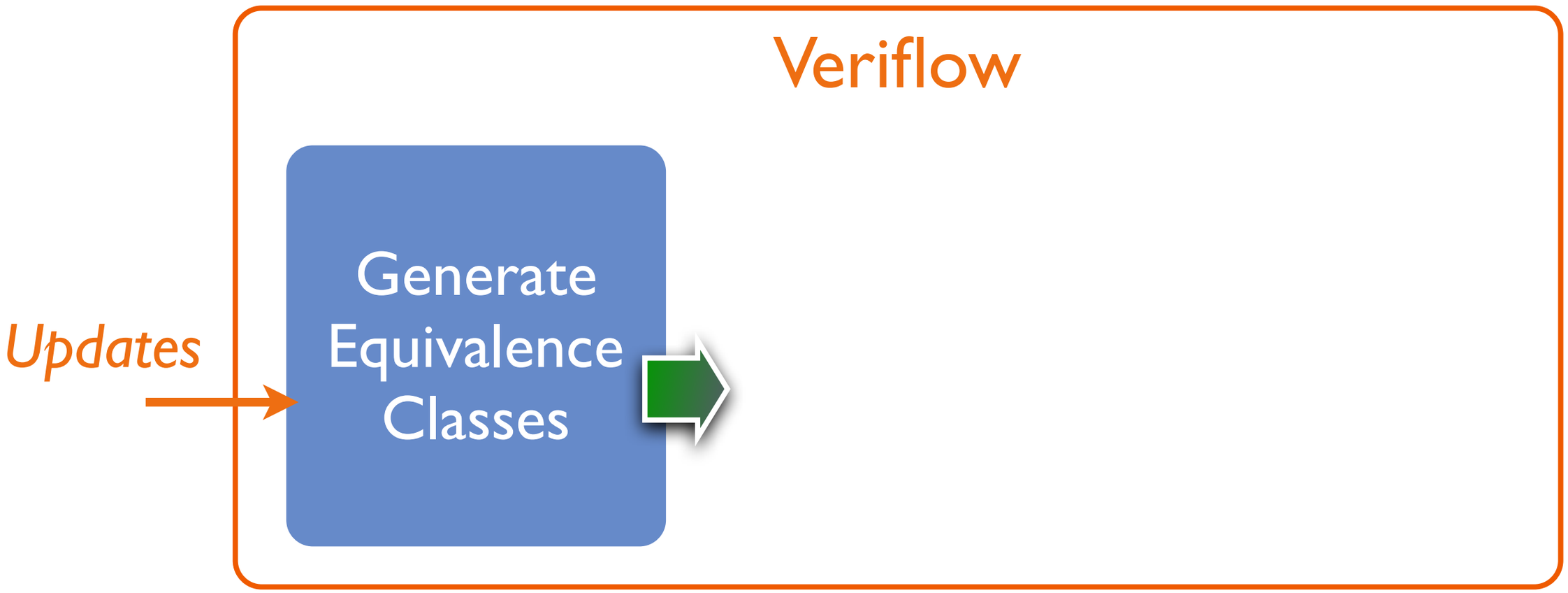
1. Limit the search space
2. Represent forwarding behaviors using graphs
3. Run Light-weight graph-based algorithm to check *reachability properties*

Verifying invariants quickly

Overview:

1. Limit the search space
2. Represent forwarding behaviors using graphs
3. Run Light-weight graph-based algorithm to check *reachability properties*

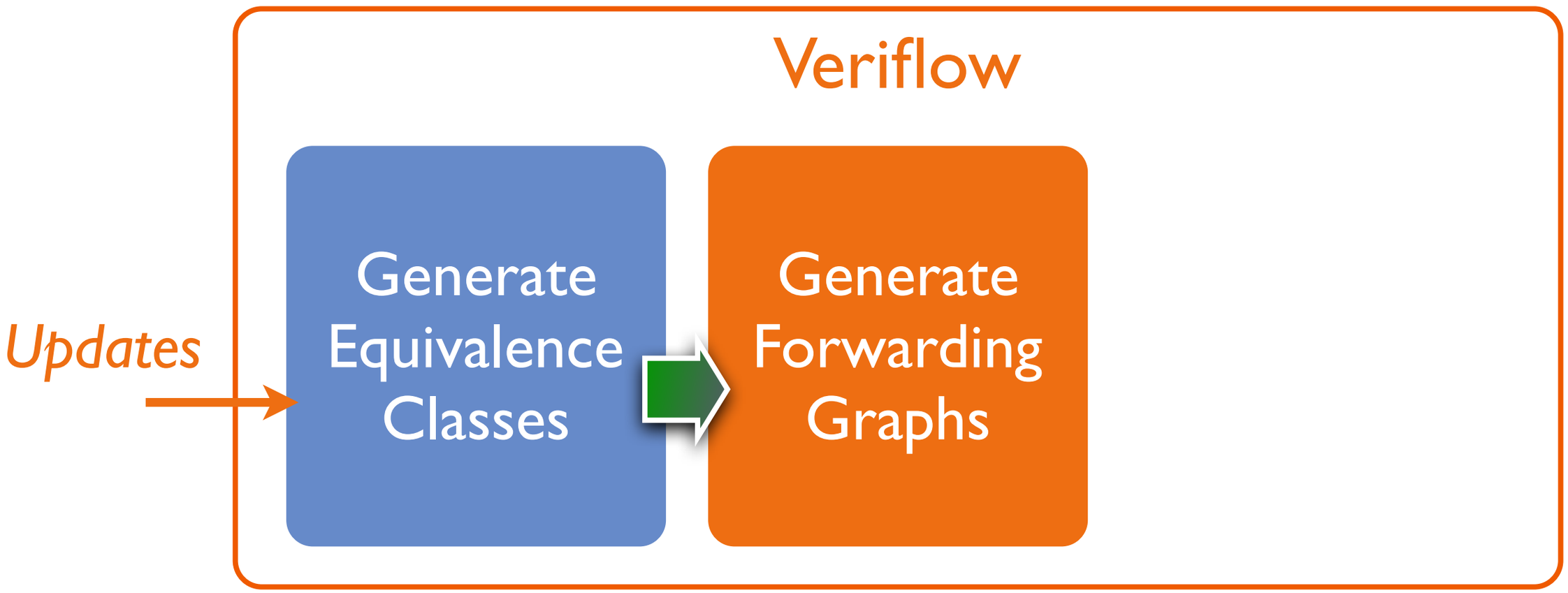
Verifying invariants quickly



Overview:

1. Limit the search space
2. Represent forwarding behaviors using graphs
3. Run Light-weight graph-based algorithm to check *reachability properties*

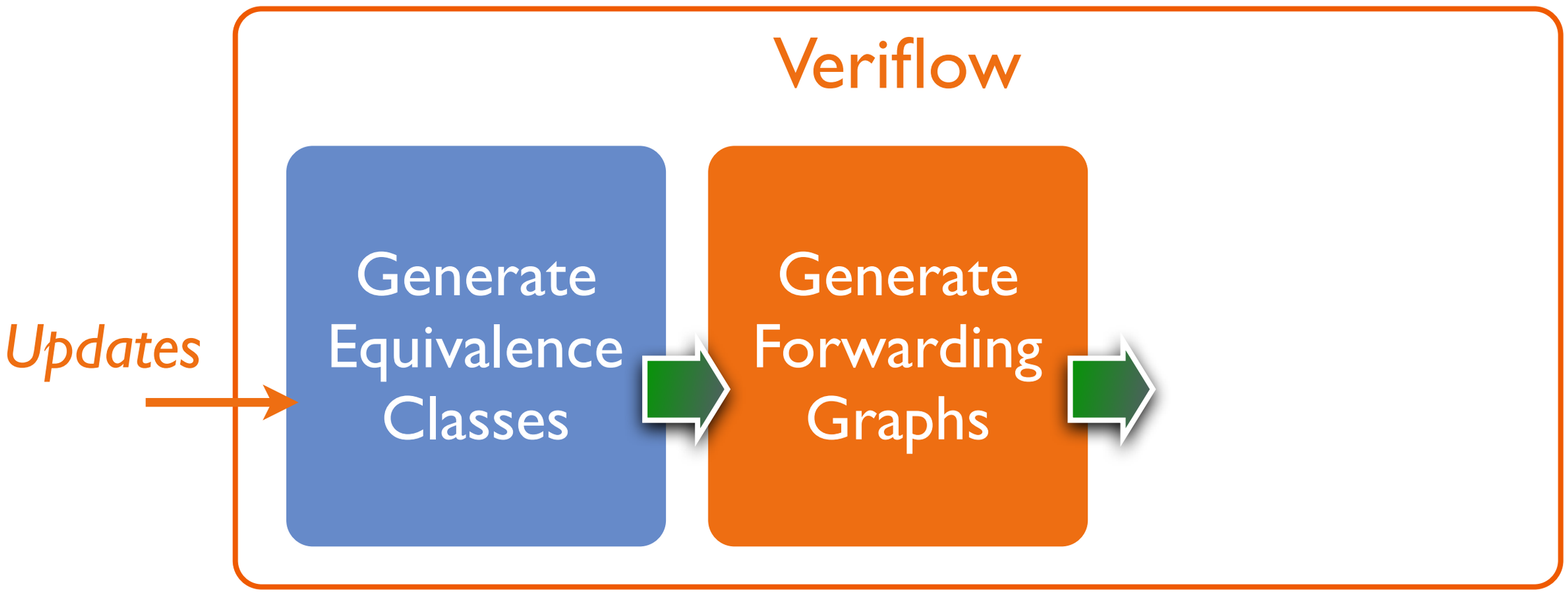
Verifying invariants quickly



Overview:

1. Limit the search space
2. Represent forwarding behaviors using graphs
3. Run Light-weight graph-based algorithm to check *reachability properties*

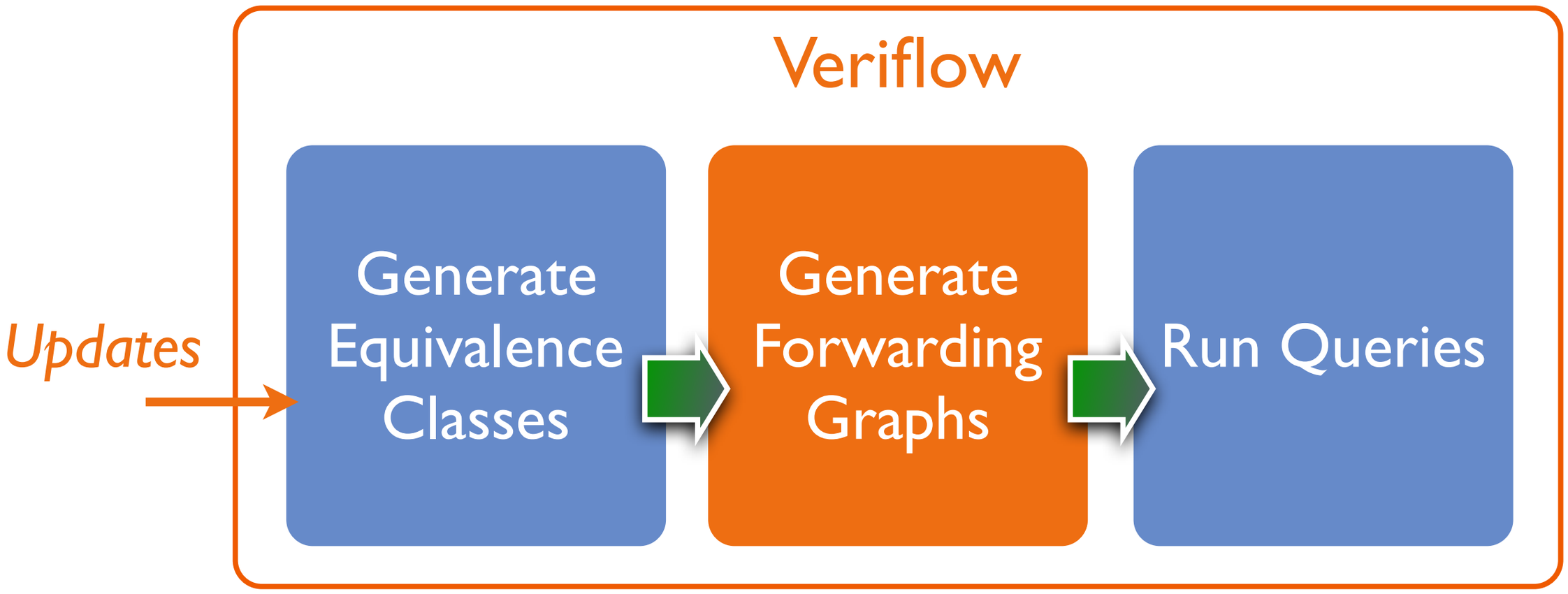
Verifying invariants quickly



Overview:

1. Limit the search space
2. Represent forwarding behaviors using graphs
3. Run Light-weight graph-based algorithm to check *reachability properties*

Verifying invariants quickly



Overview:

1. Limit the search space
2. Represent forwarding behaviors using graphs
3. Run Light-weight graph-based algorithm to check *reachability properties*

MODELING DYNAMIC NETWORKS

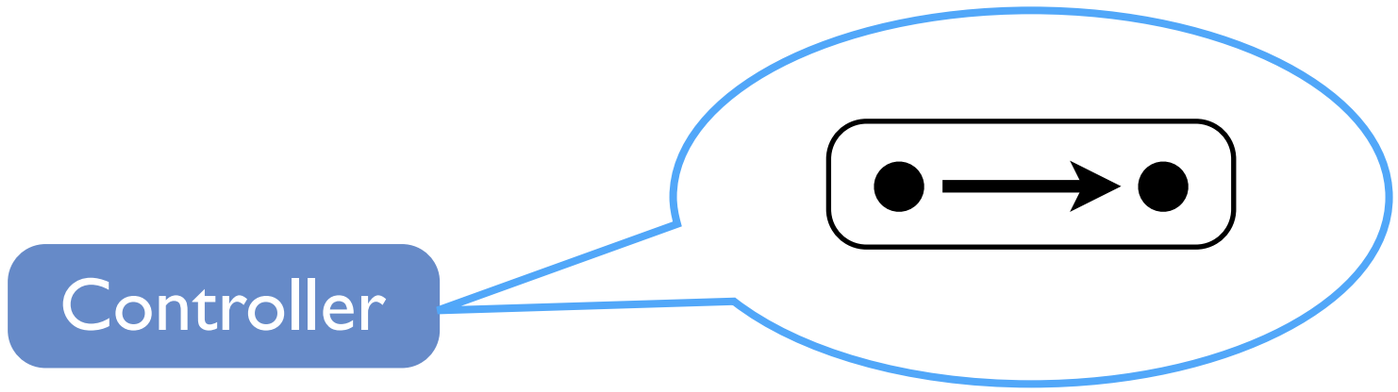
Enforcing Customizable Consistency Properties in Software-Defined Networks

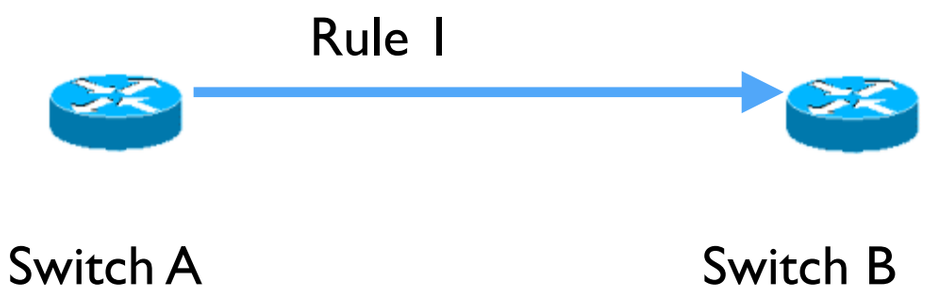
Zhou, Jin, Croft, Caesar, Godfrey

NSDI 2015

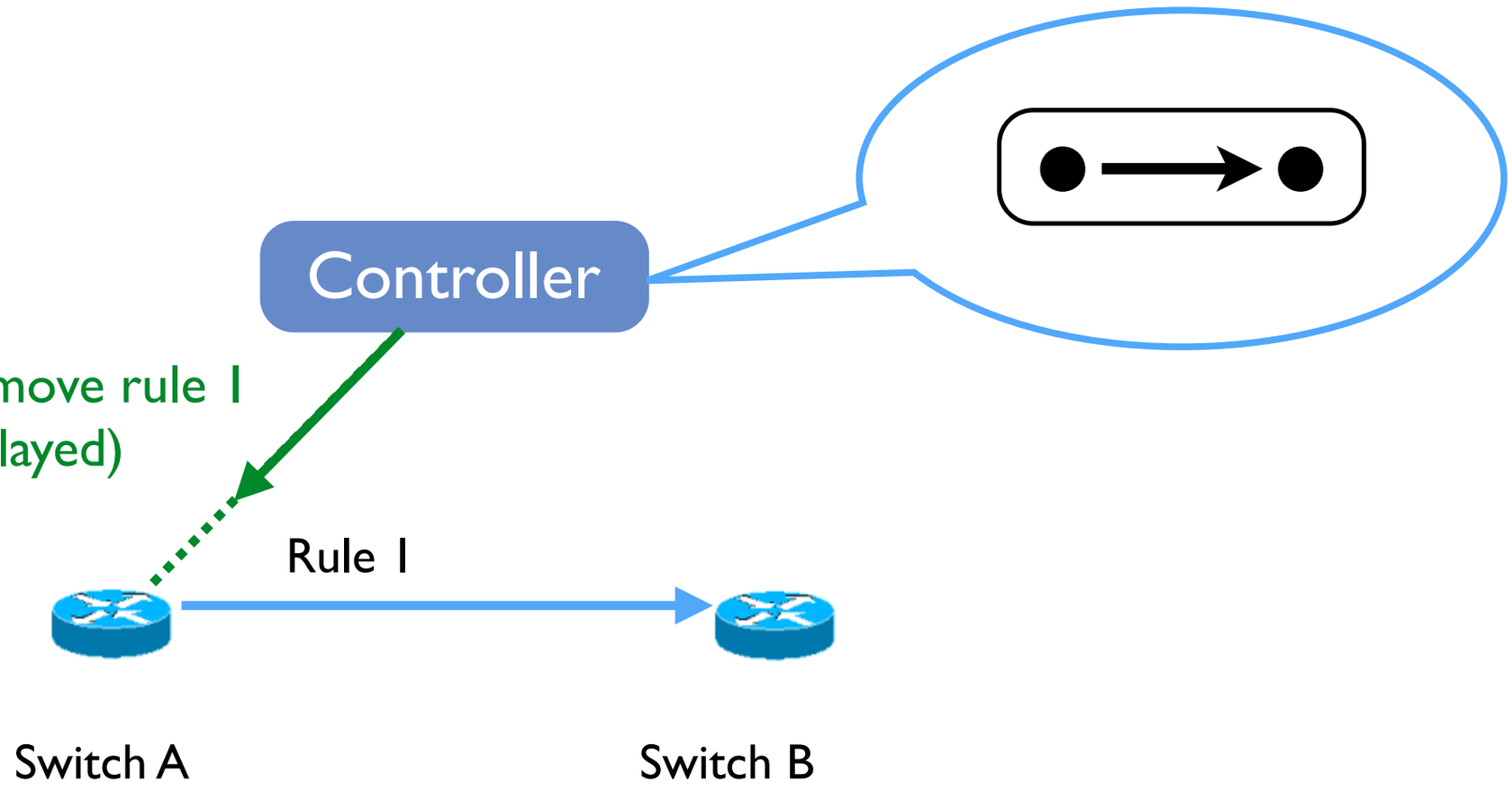
Timing uncertainty

Timing uncertainty

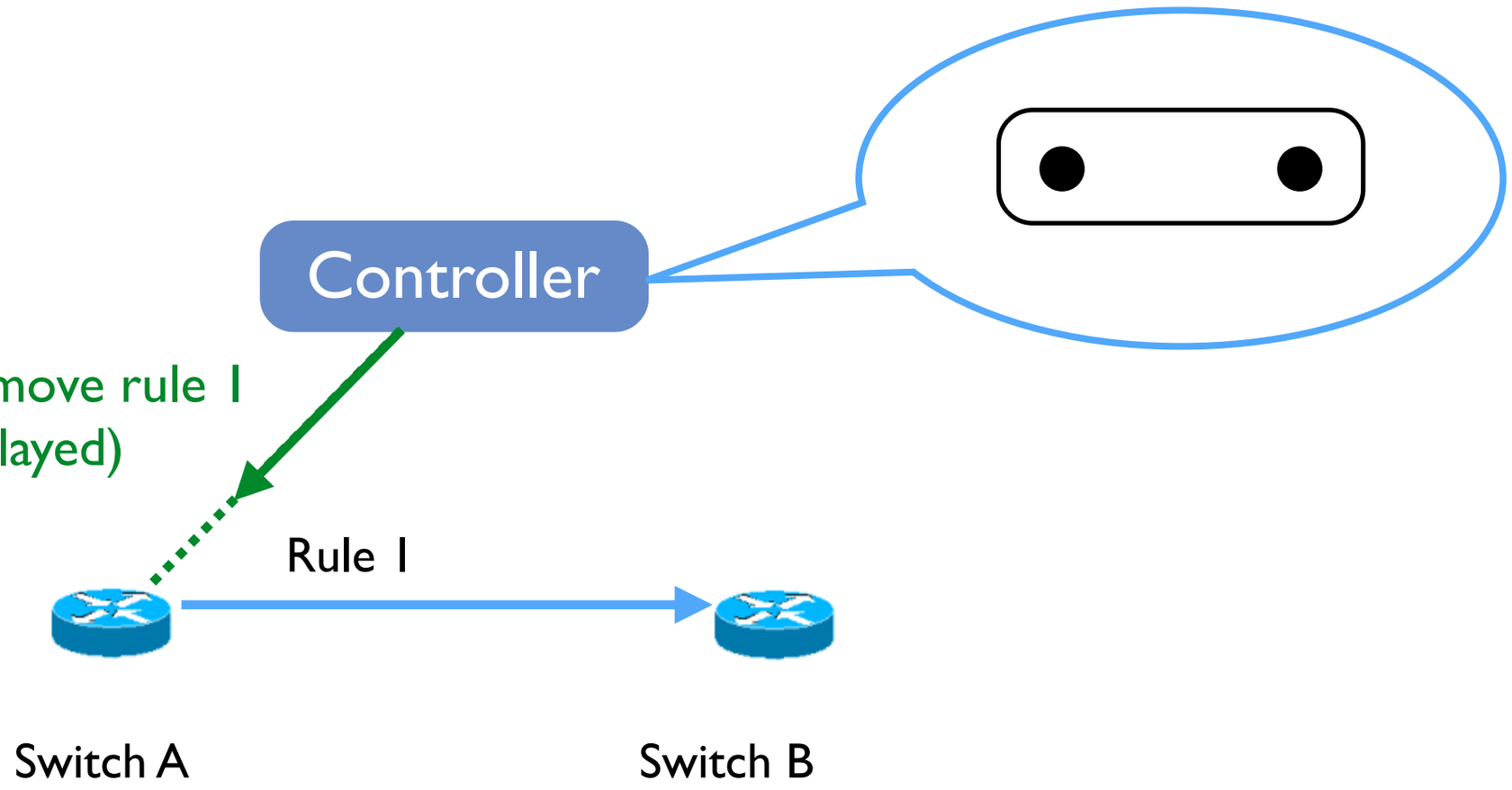




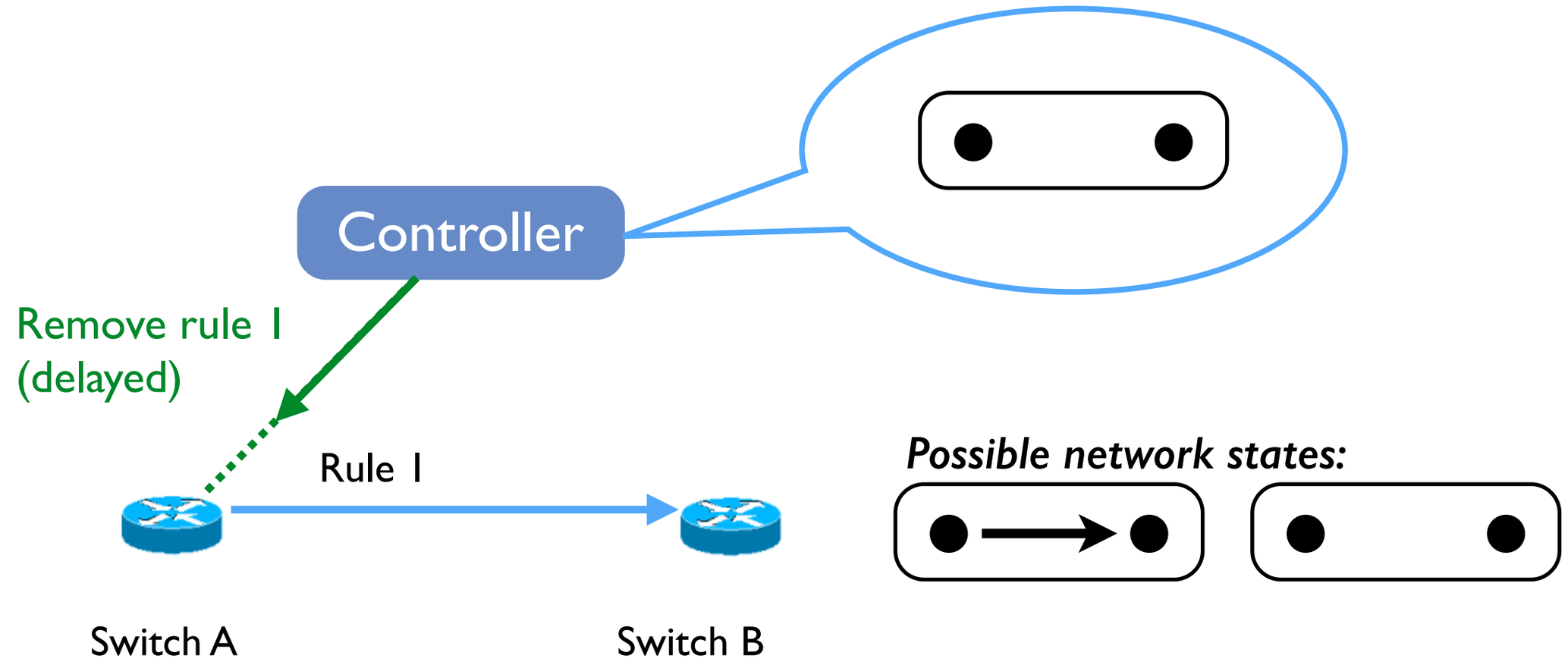
Timing uncertainty



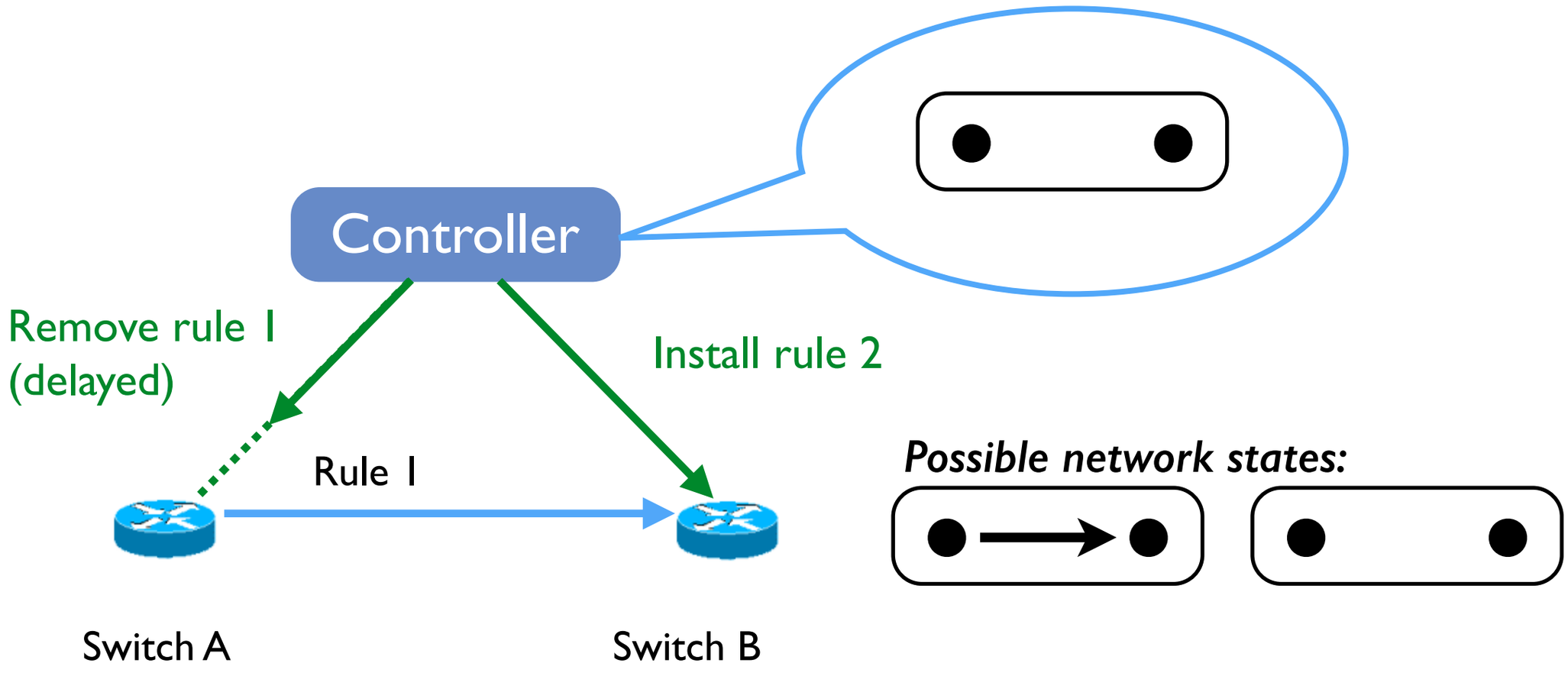
Timing uncertainty



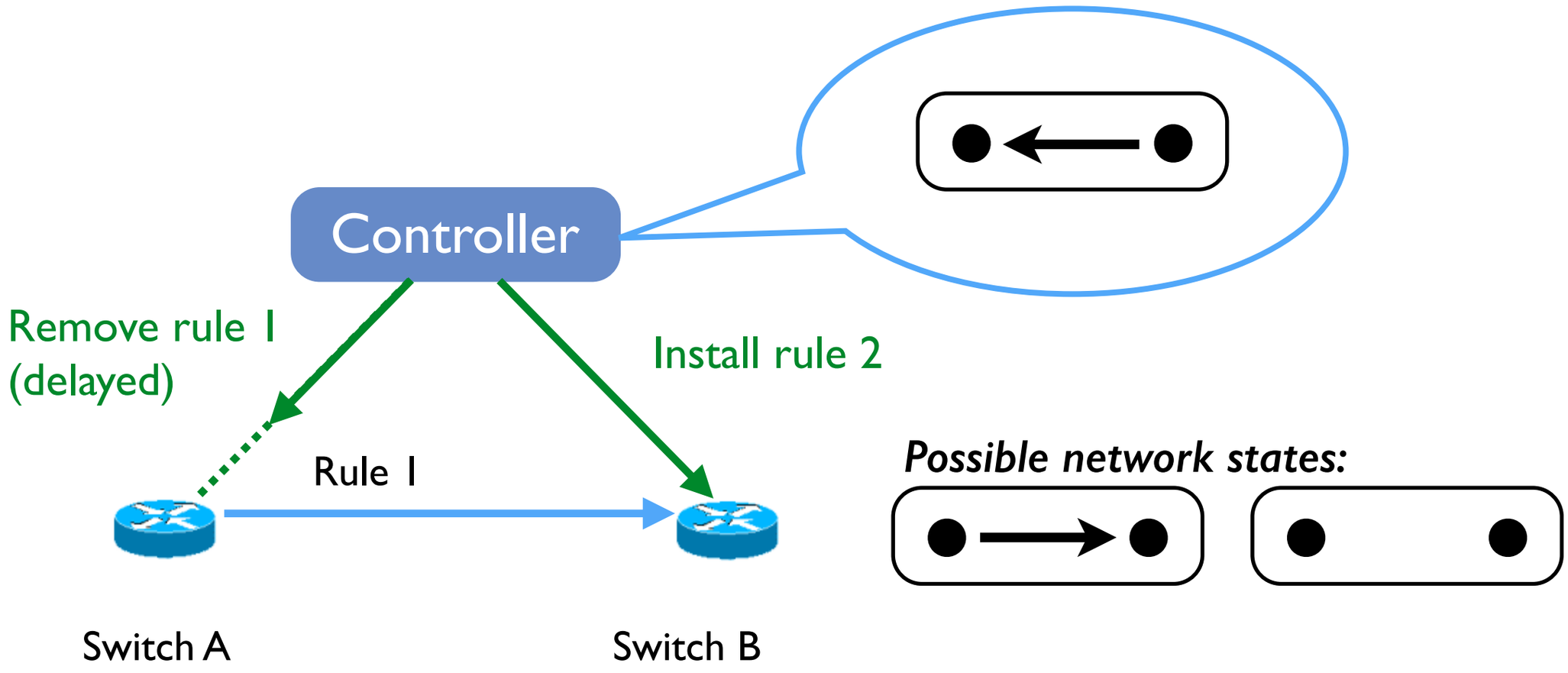
Timing uncertainty



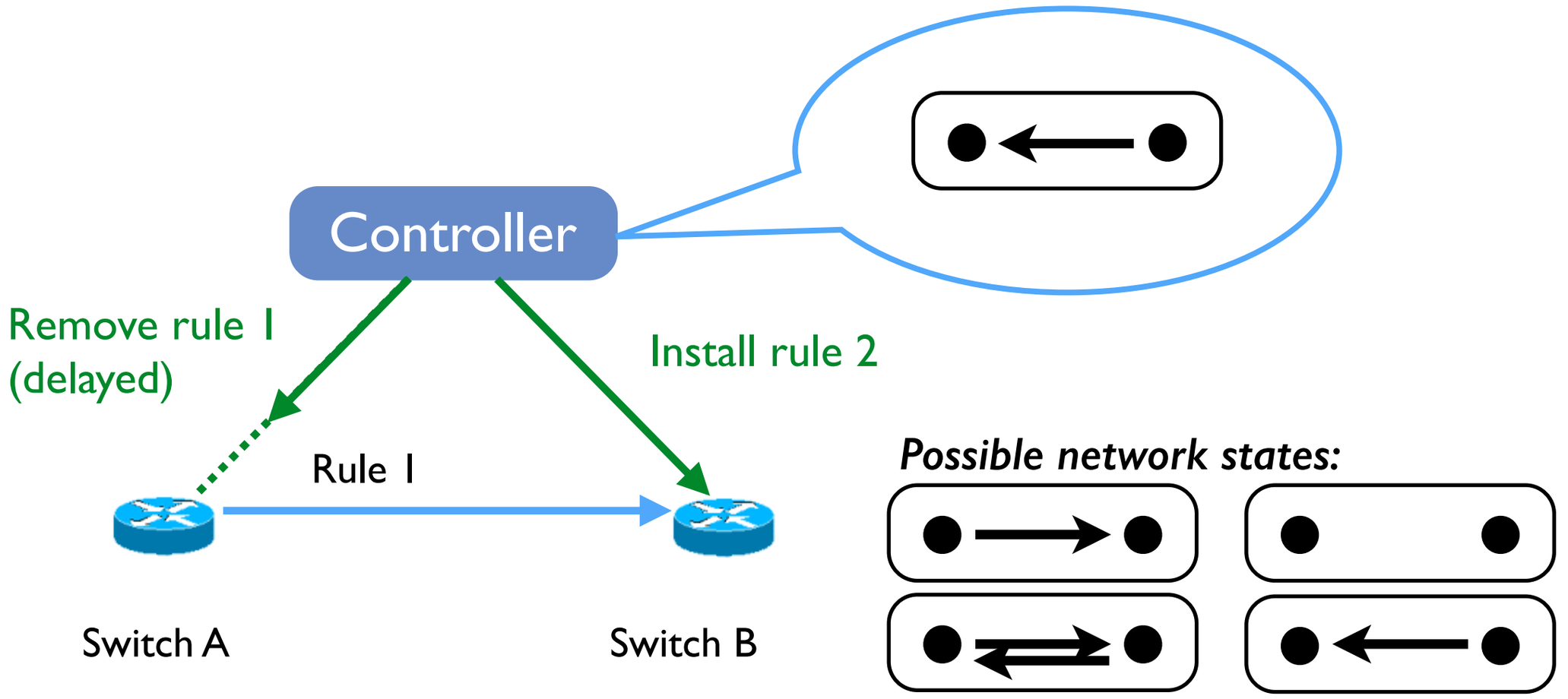
Timing uncertainty



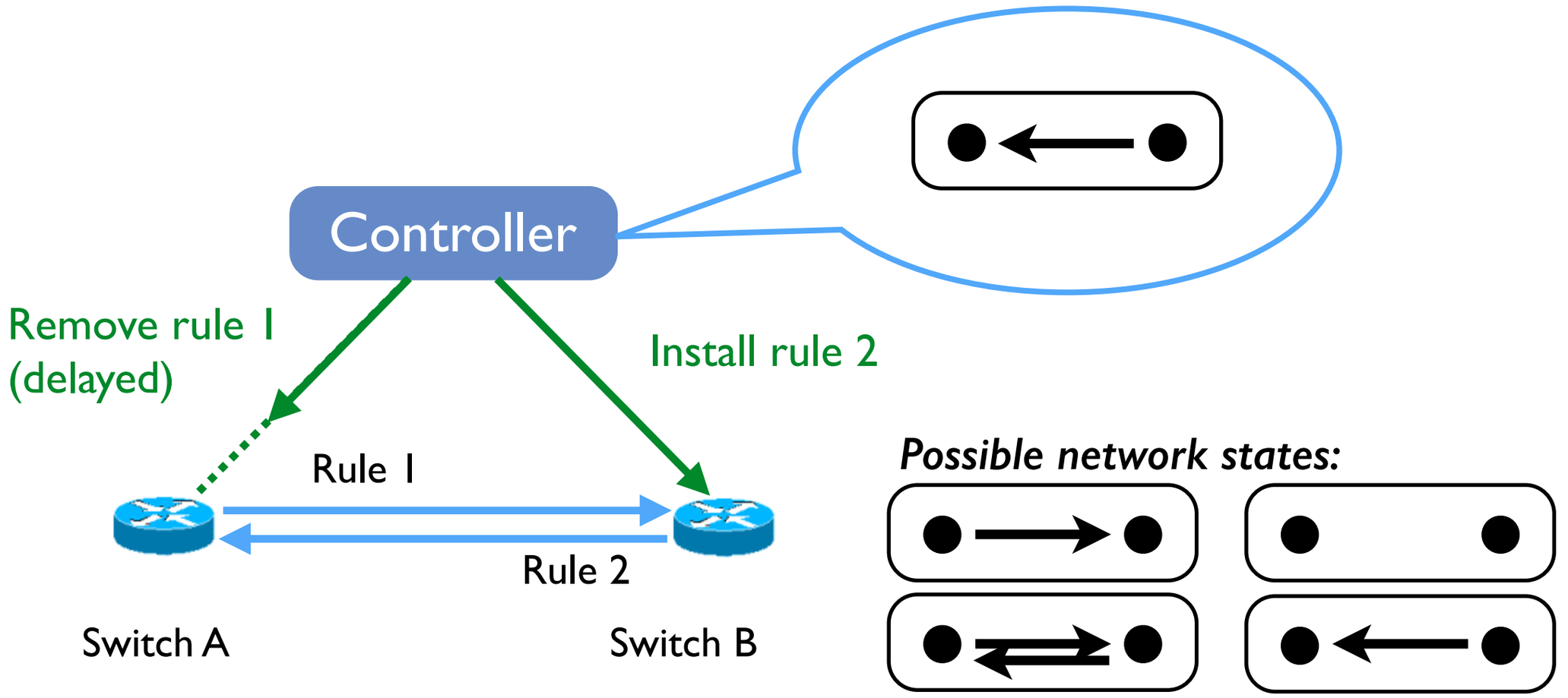
Timing uncertainty



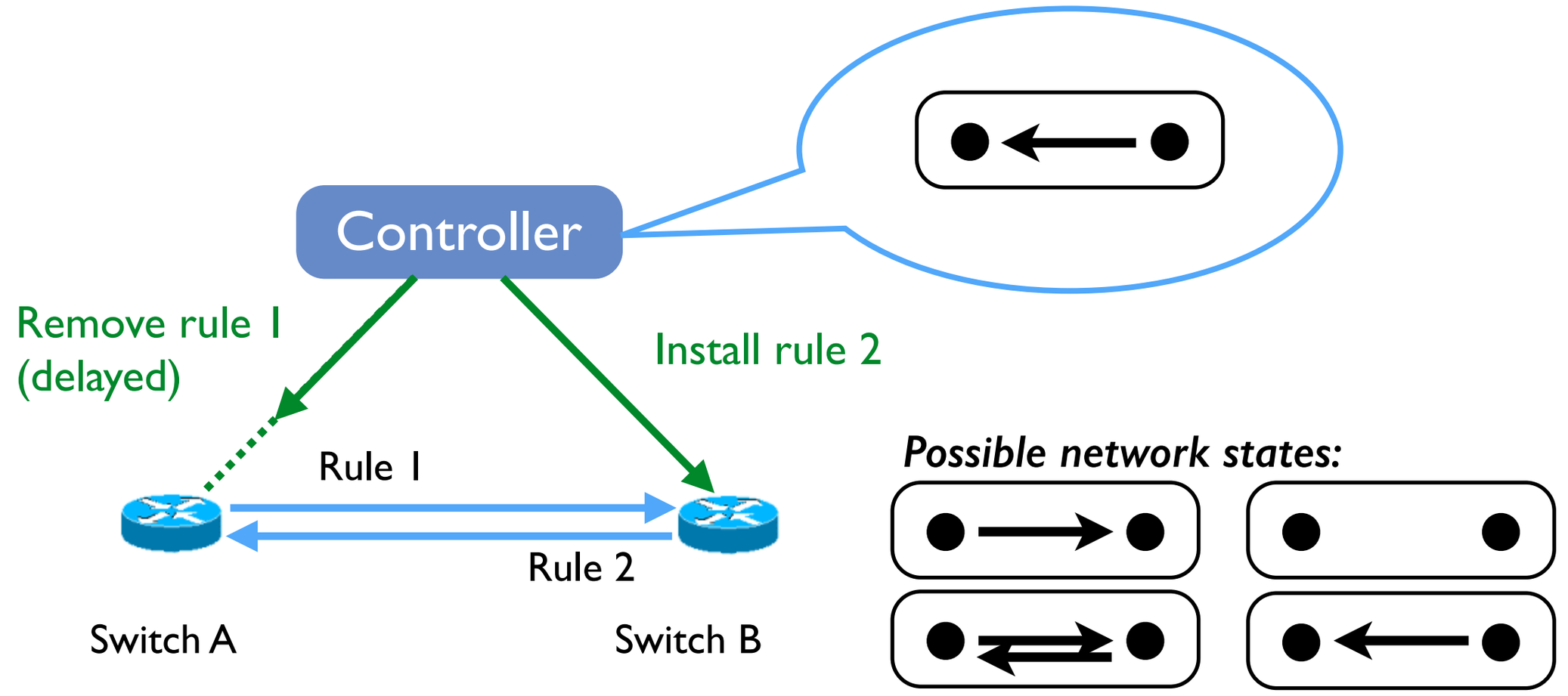
Timing uncertainty



Timing uncertainty



Timing uncertainty

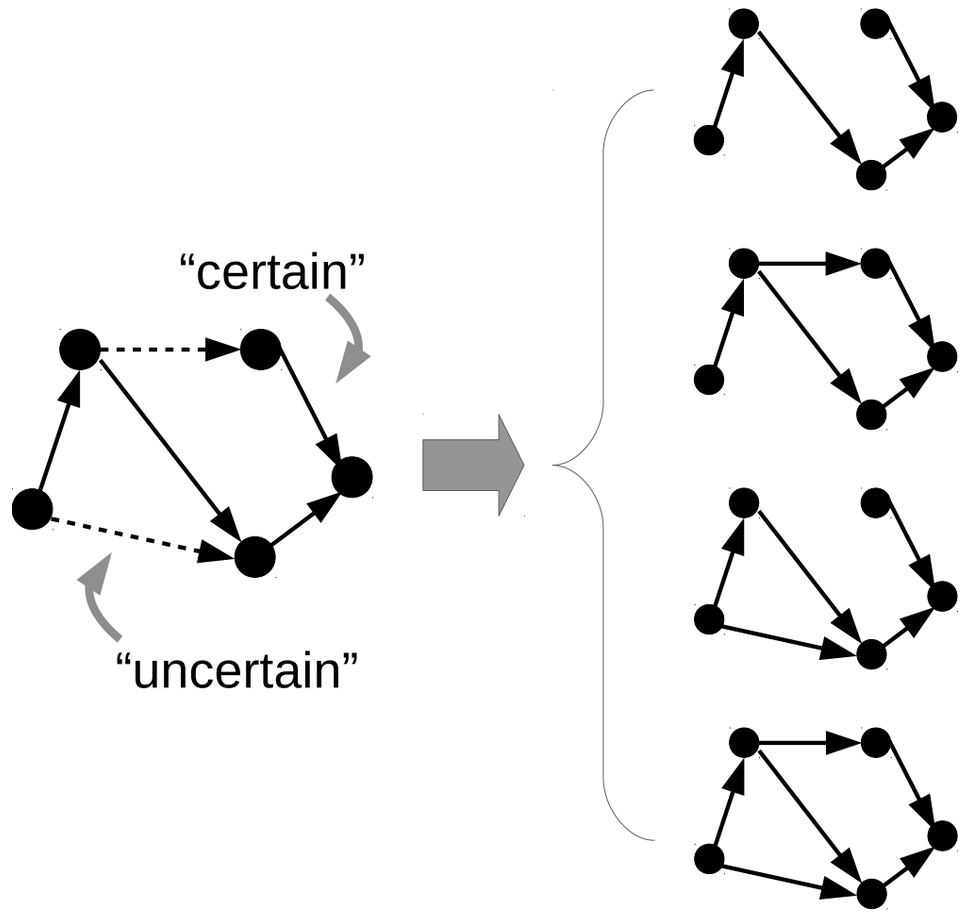


One solution: "consistent updates"

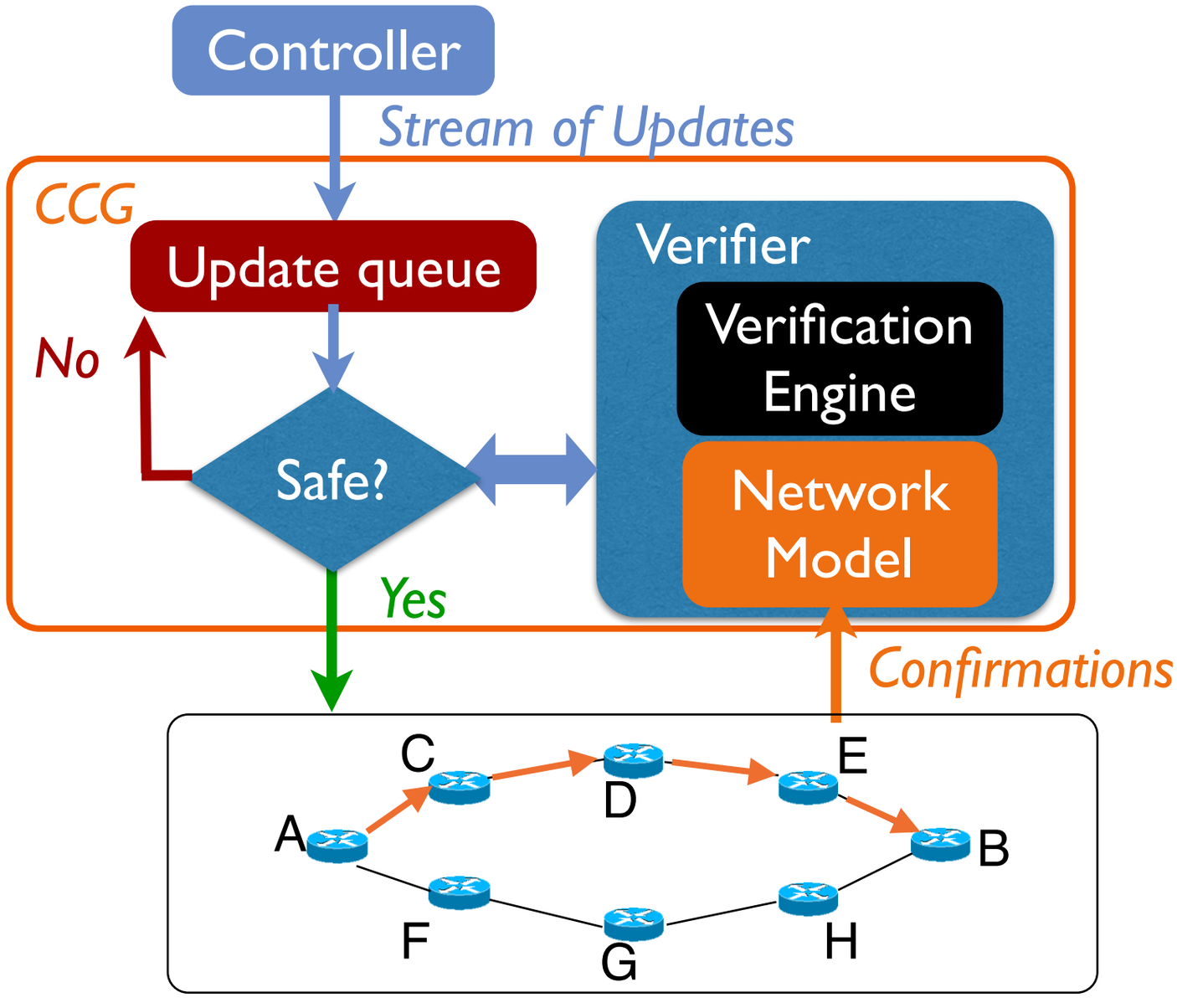
[Reitblatt, Foster, Rexford, Schlesinger, Walker, "Abstractions for Network Update", SIGCOMM 2012]

Uncertainty-aware verification

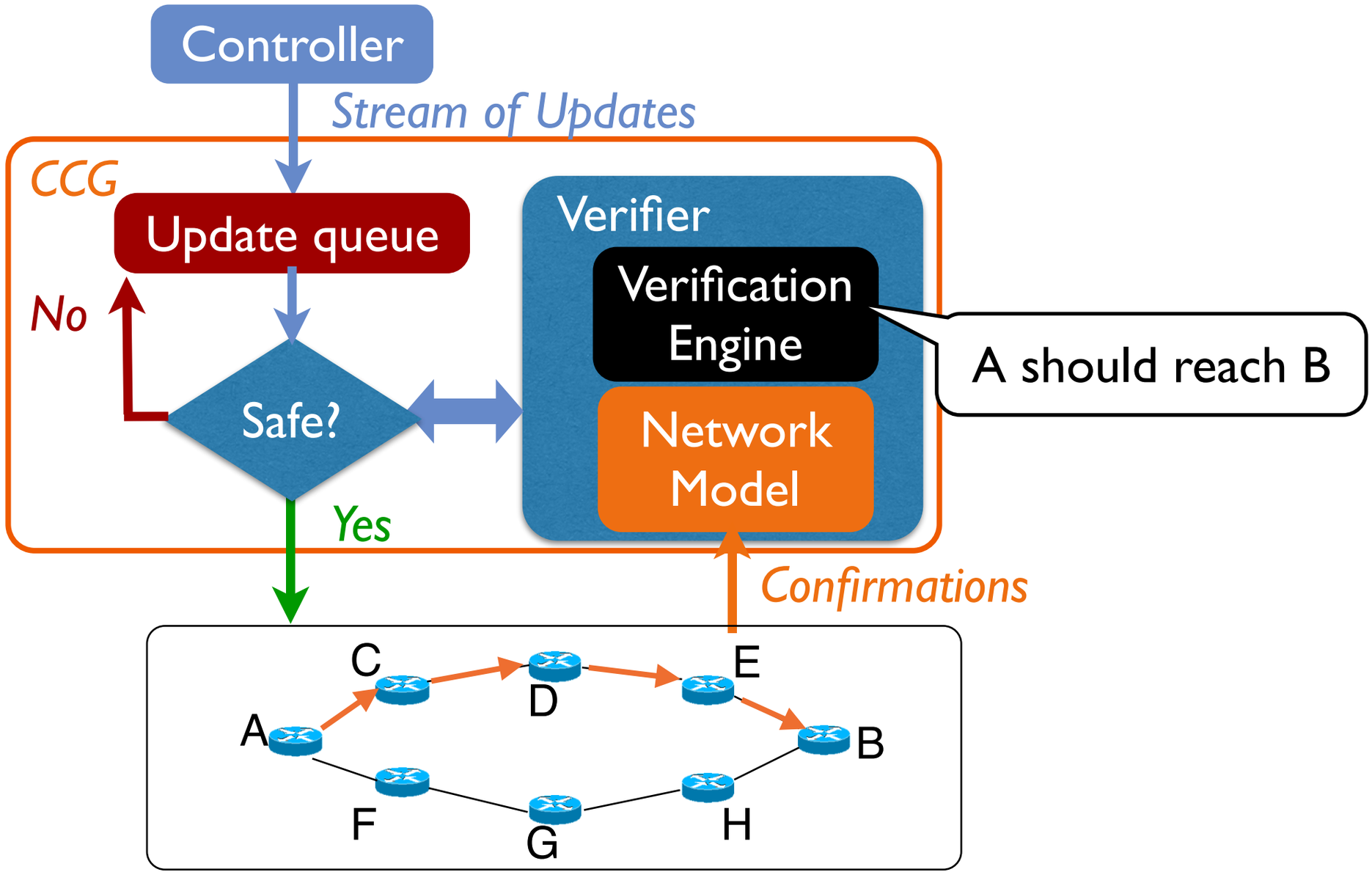
Uncertainty-aware verification



Update synthesis via verification

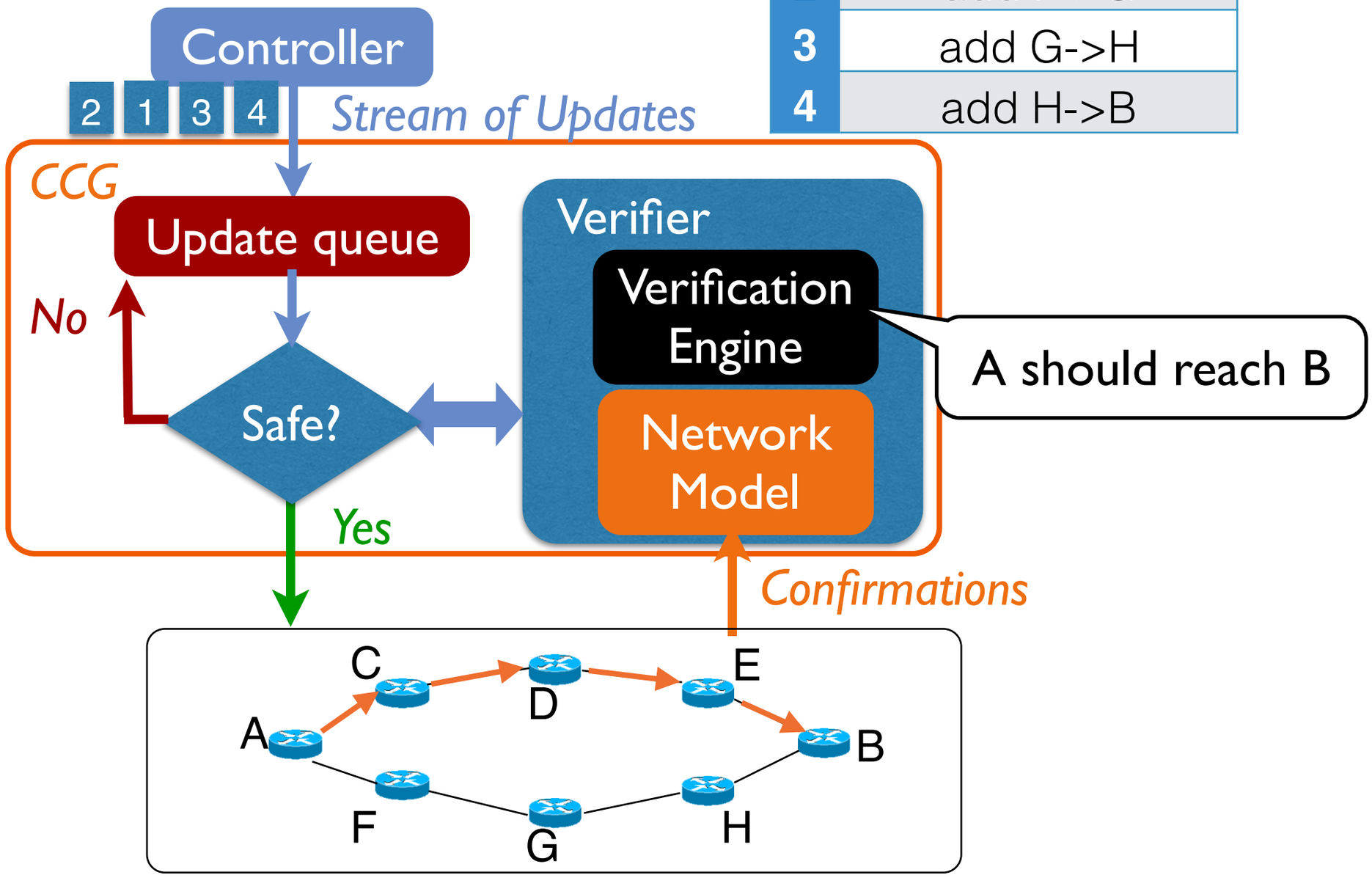


Update synthesis via verification



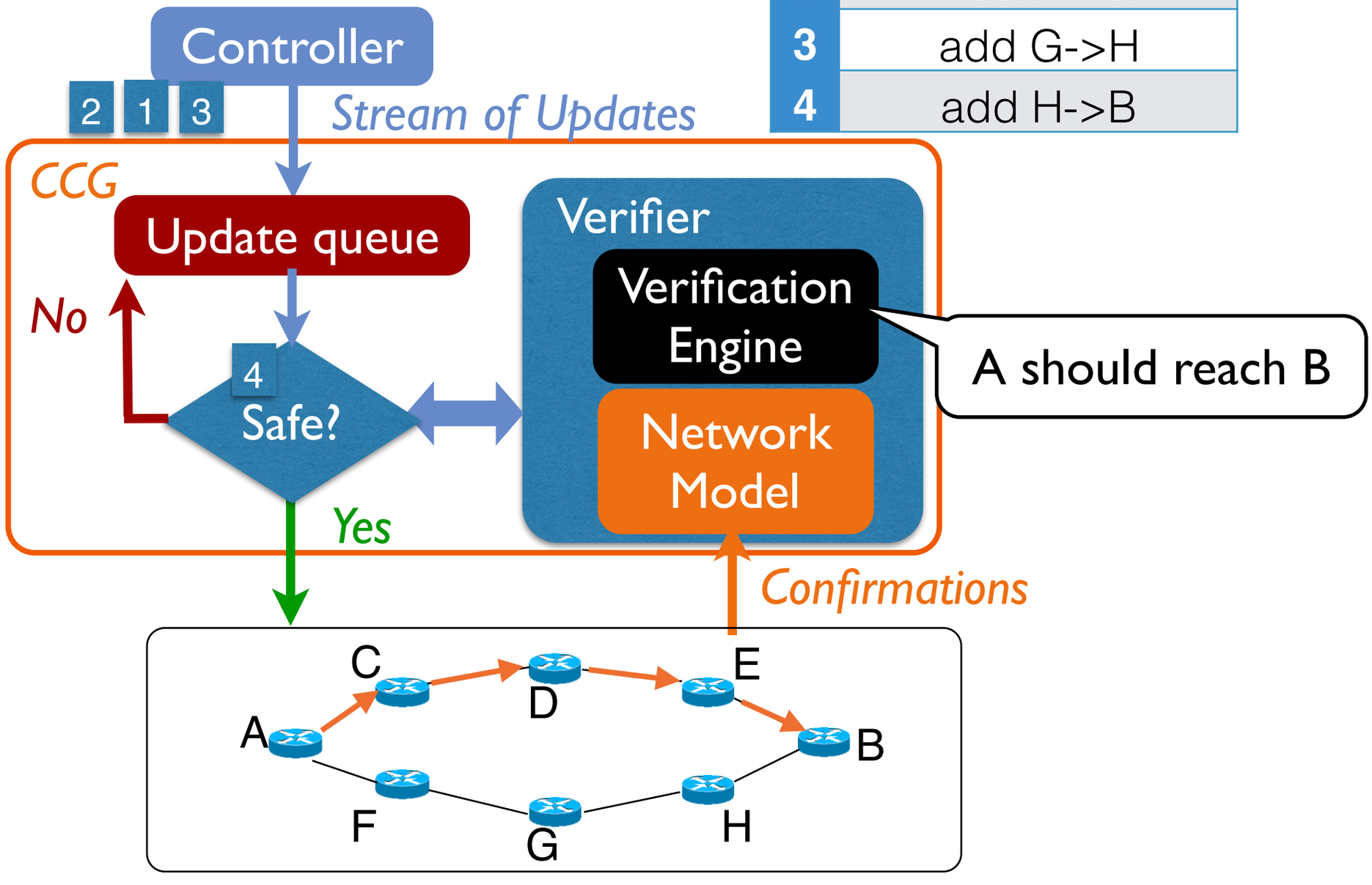
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B



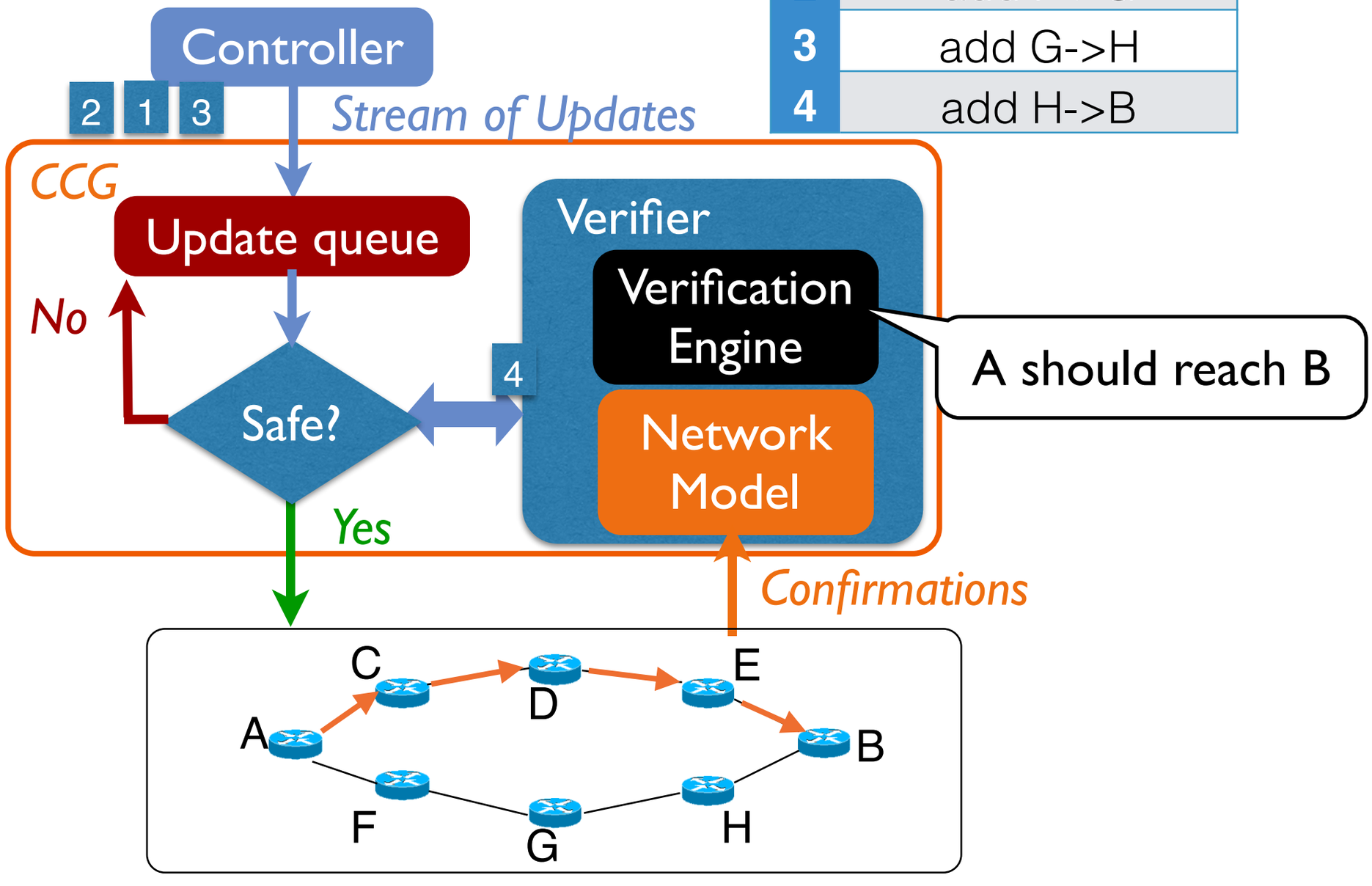
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B



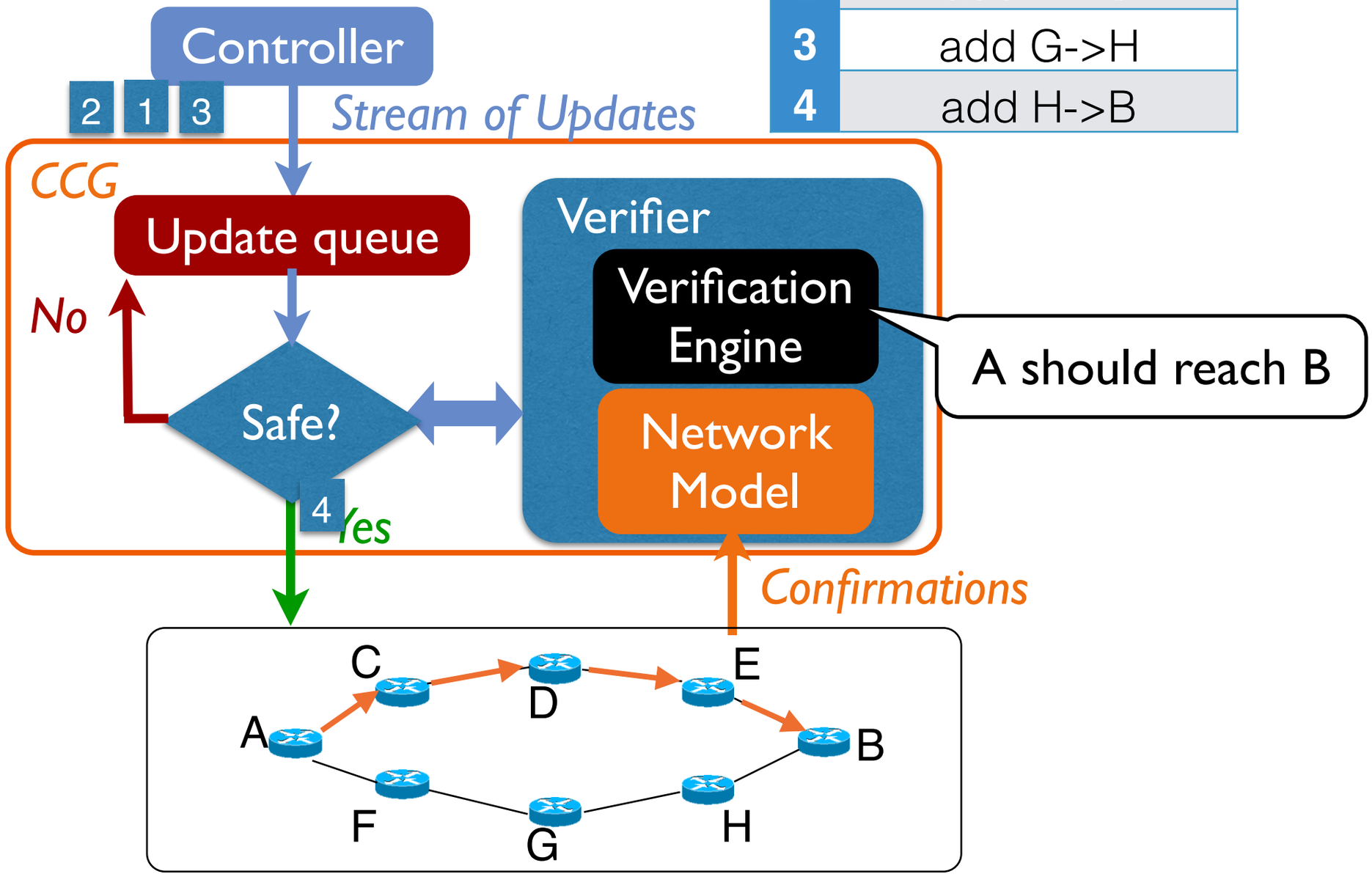
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B



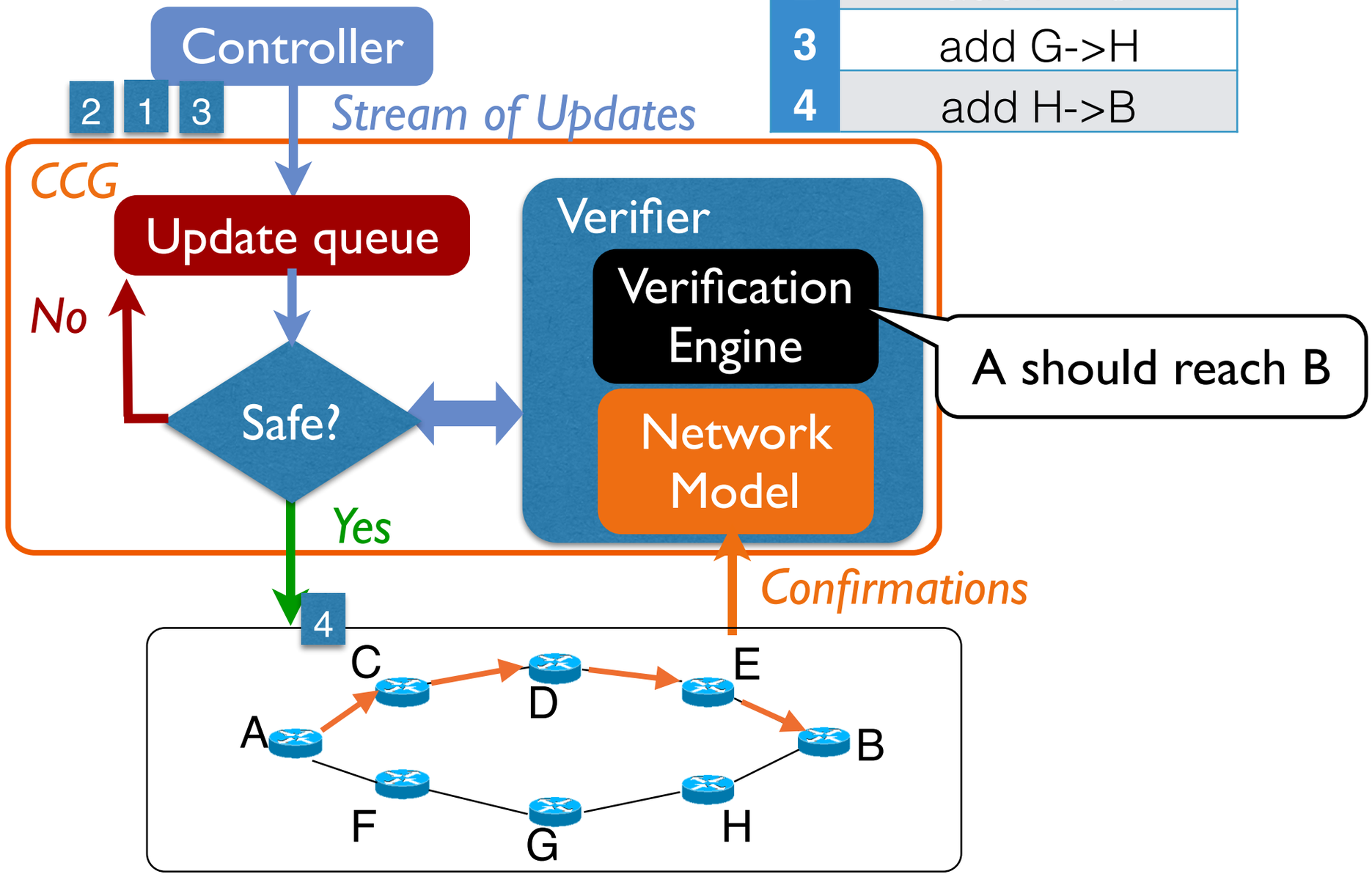
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B

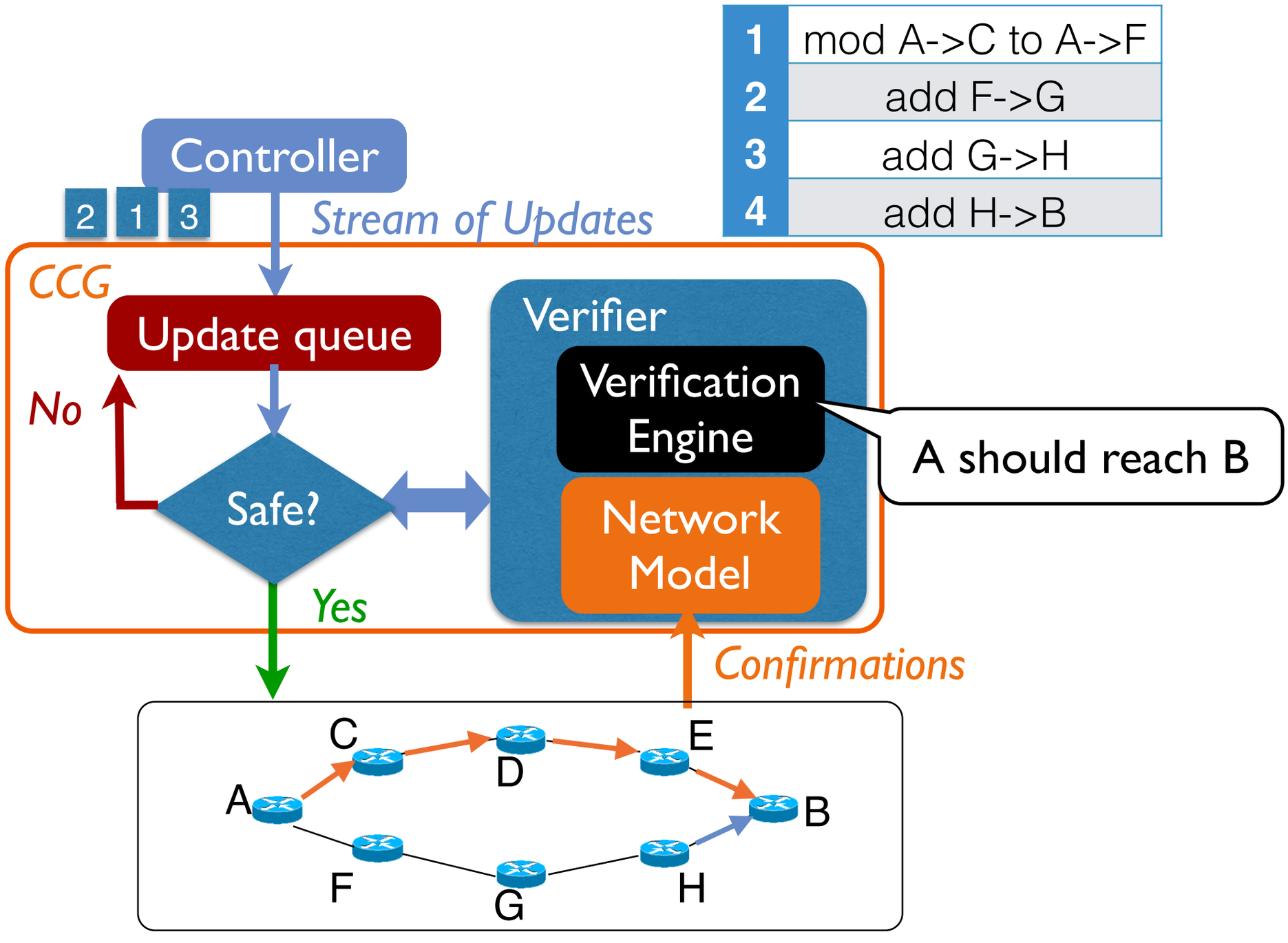


Update synthesis via verification

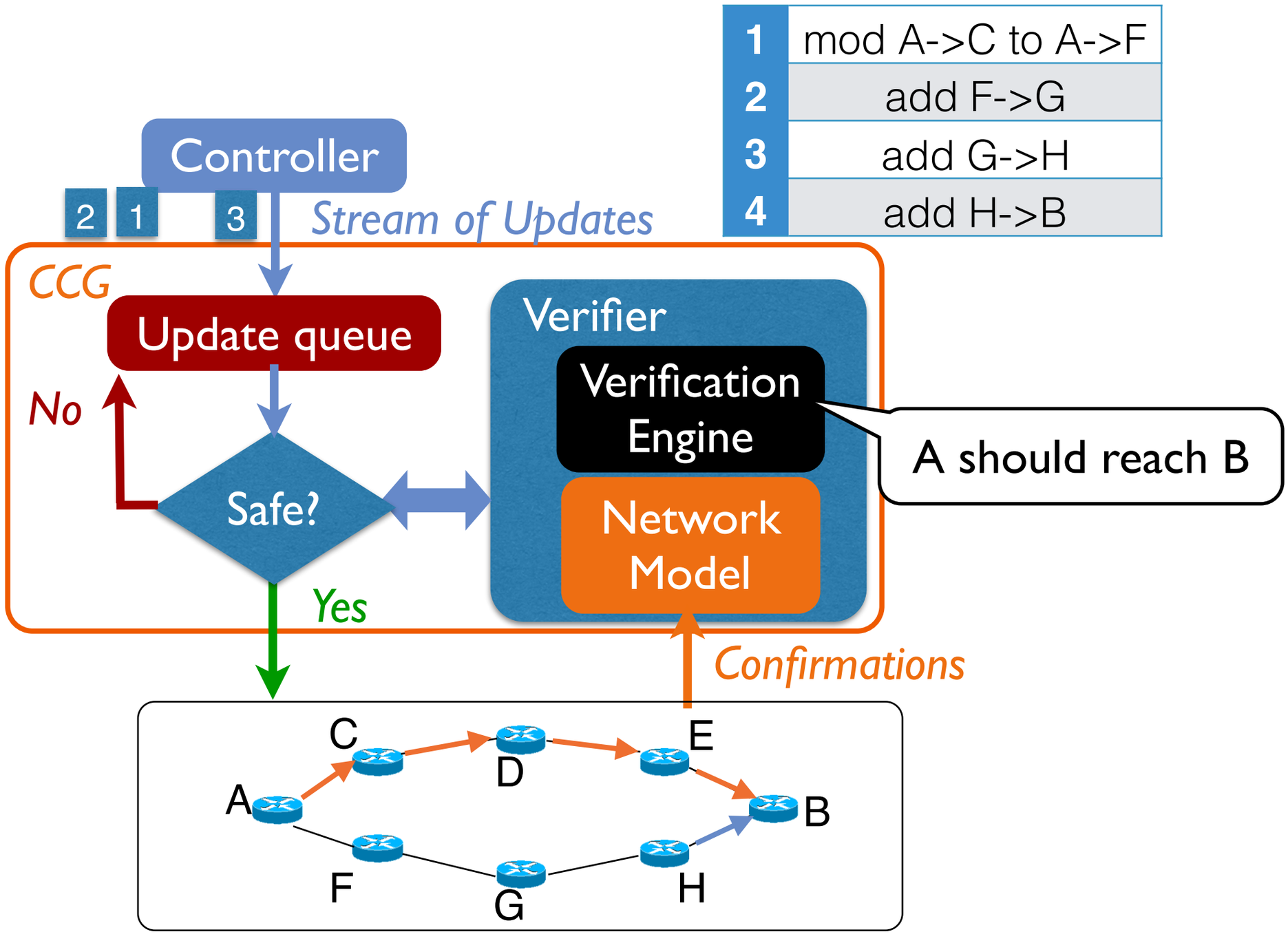
1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B



Update synthesis via verification

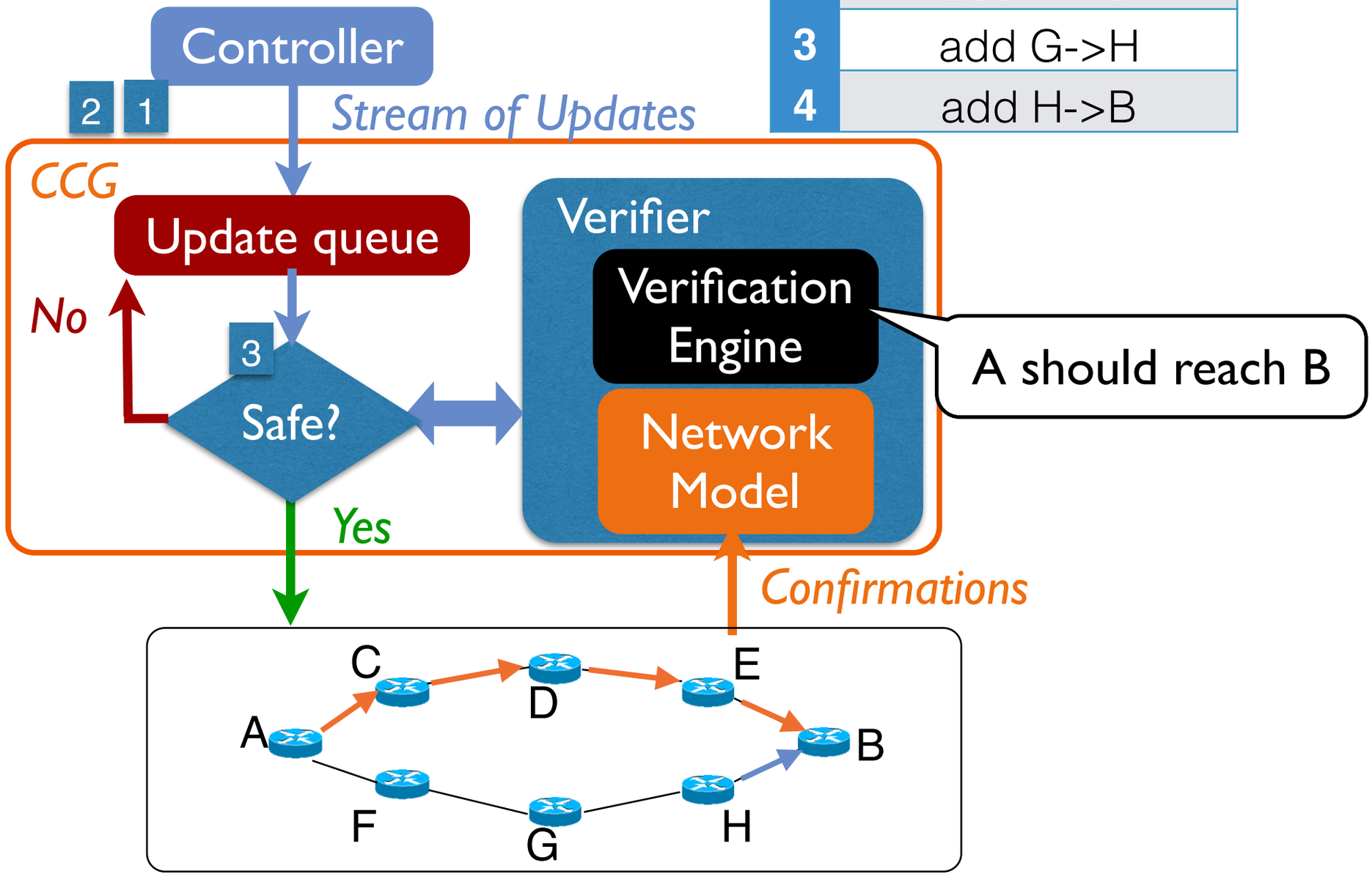


Update synthesis via verification



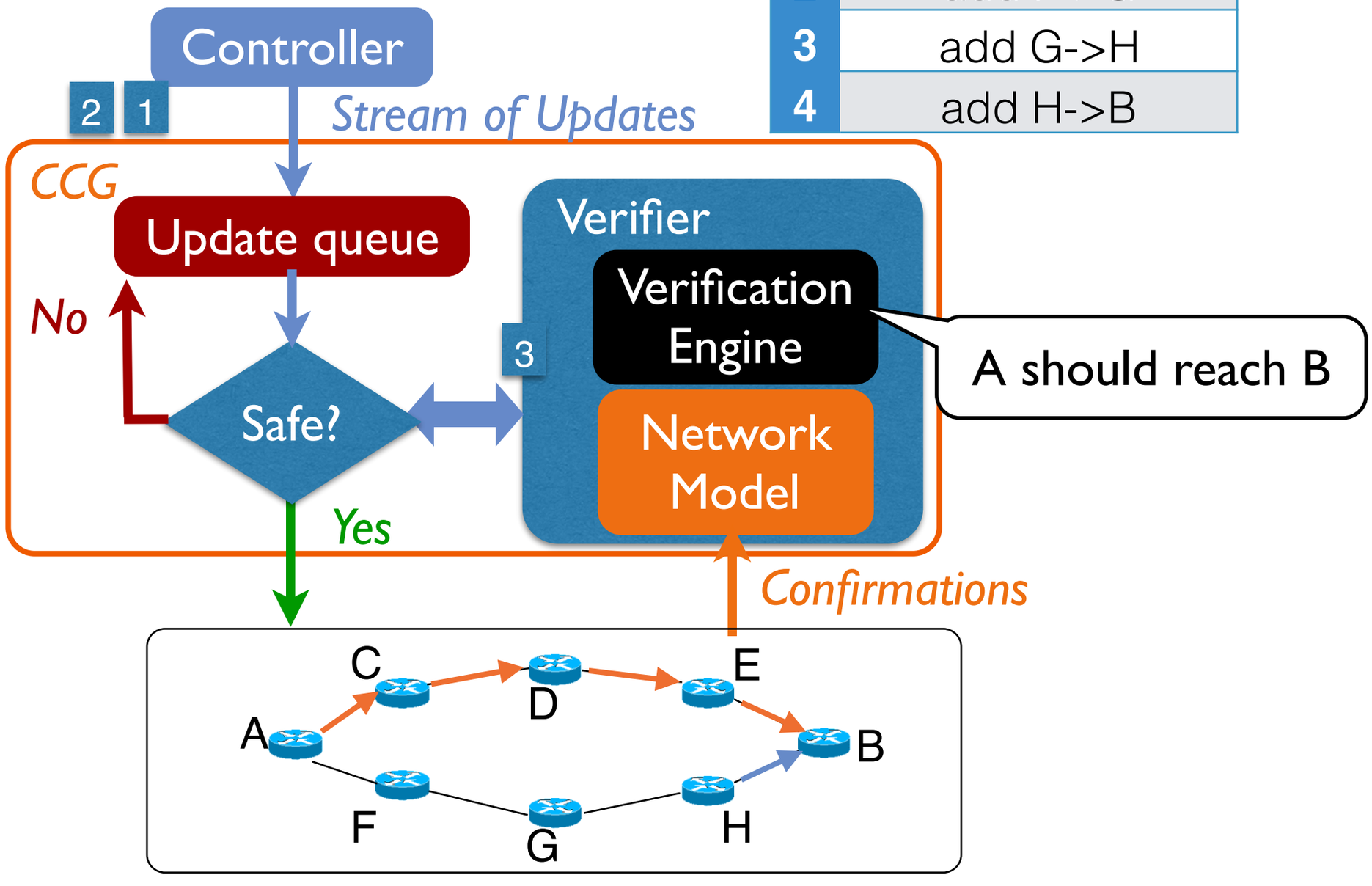
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B

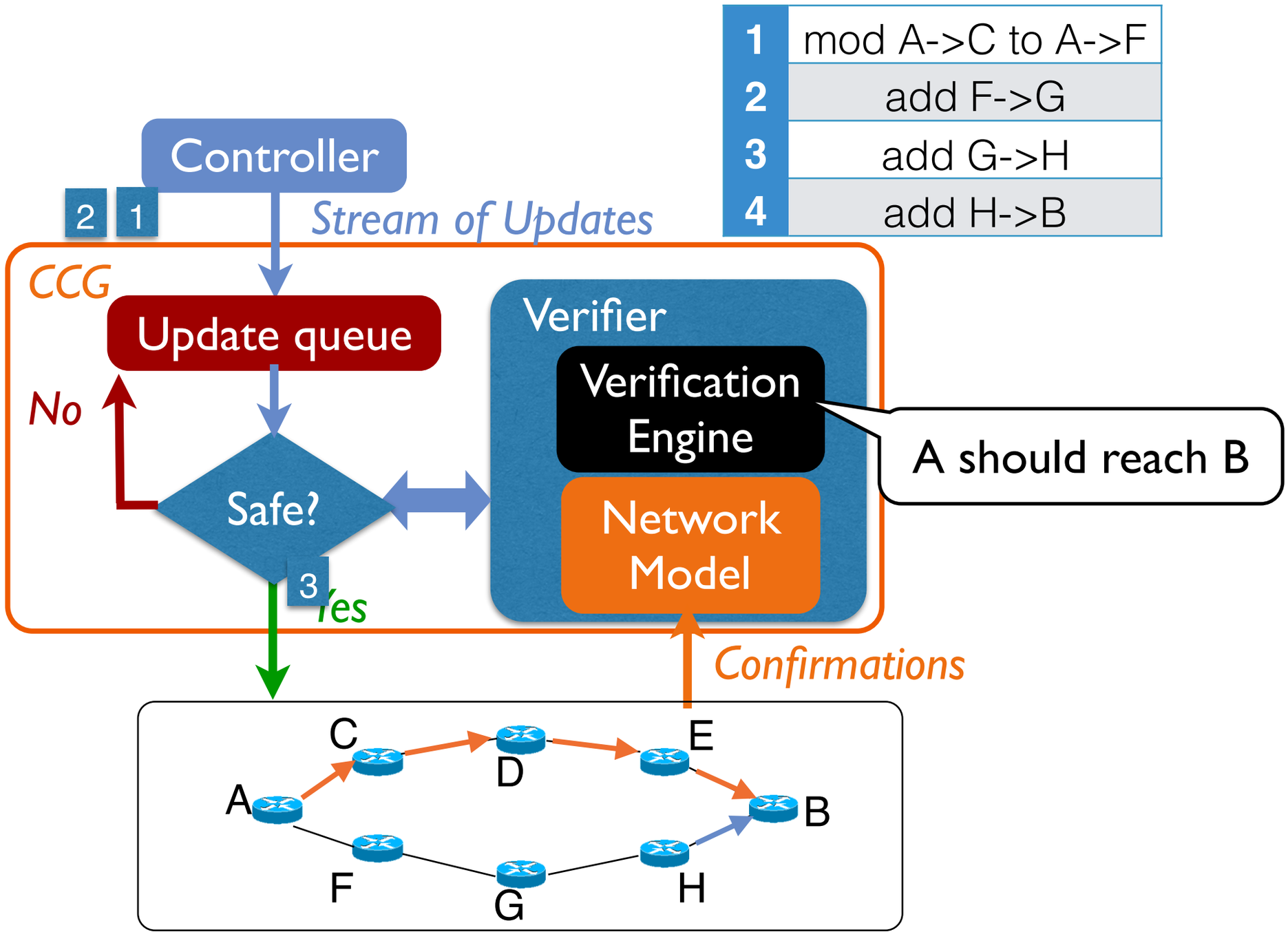


Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B

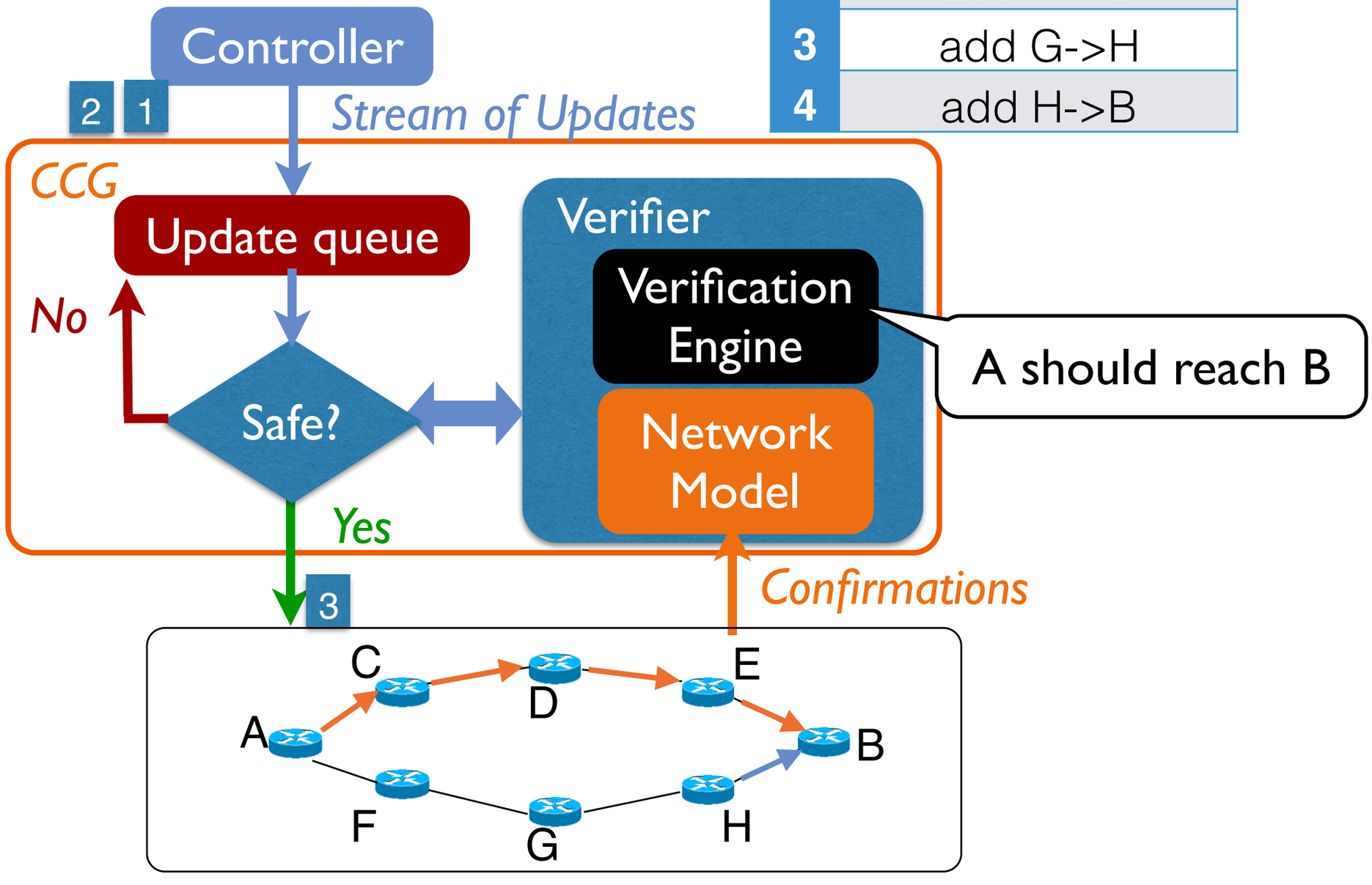


Update synthesis via verification

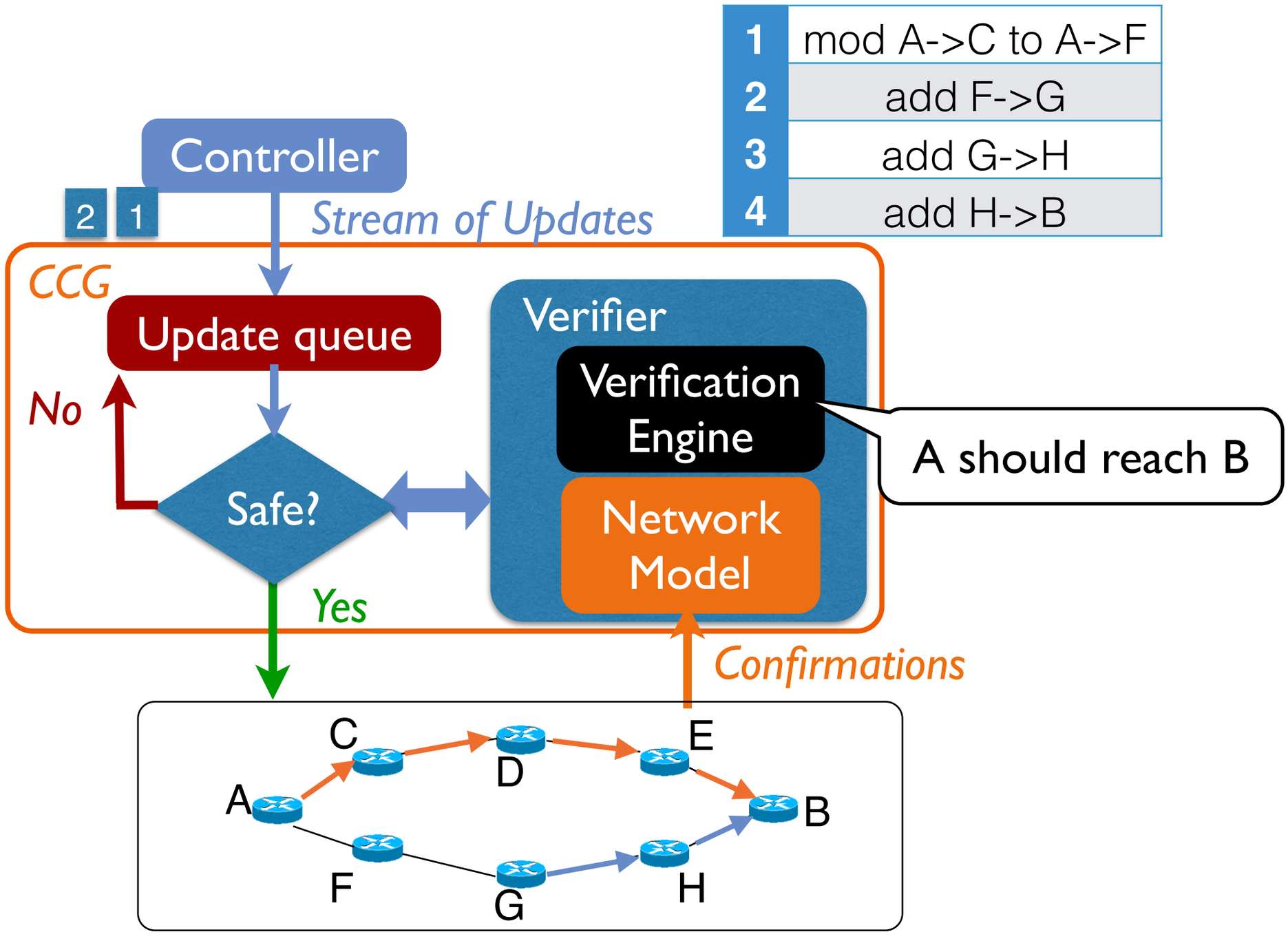


Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B

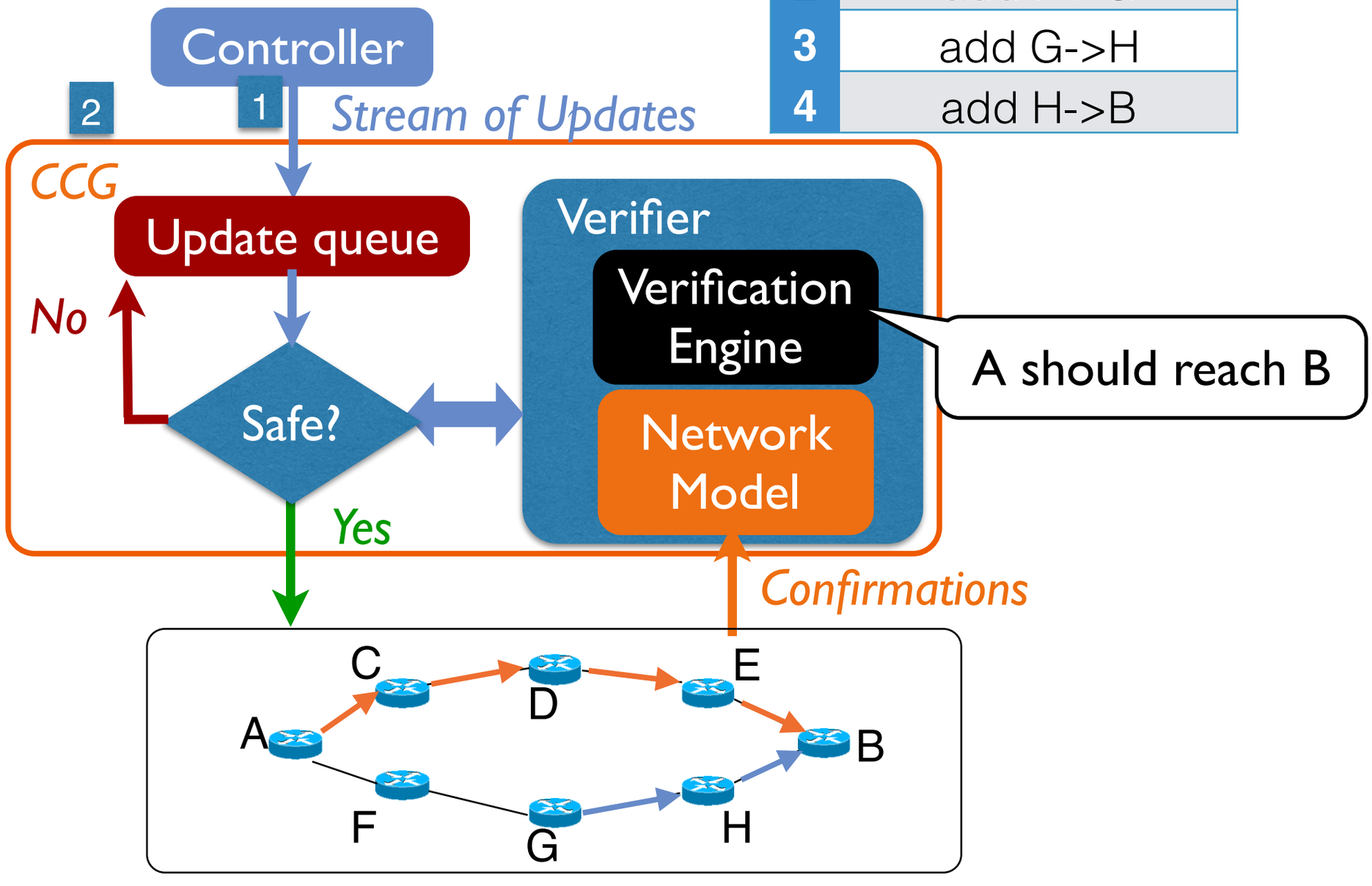


Update synthesis via verification



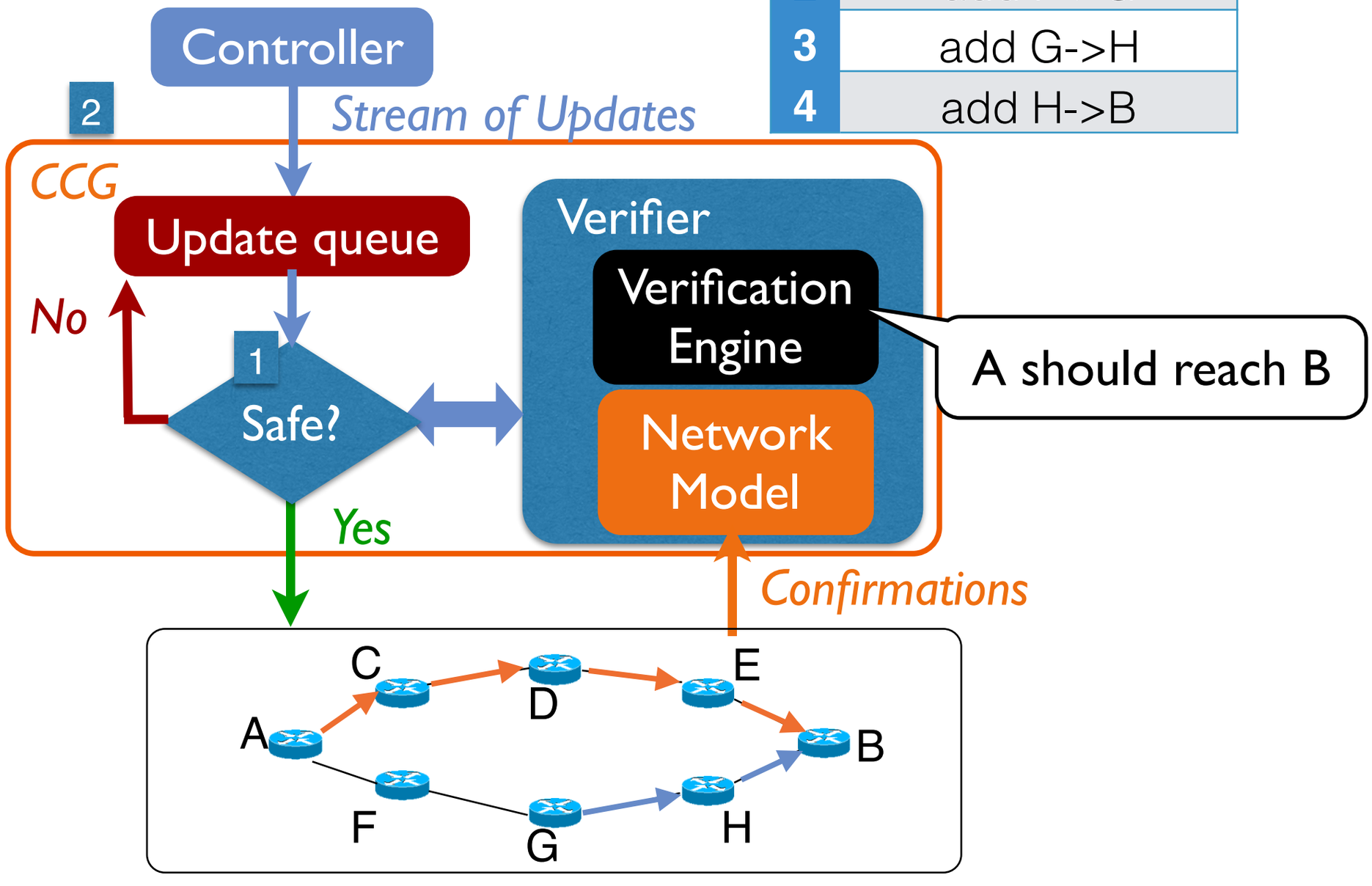
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B

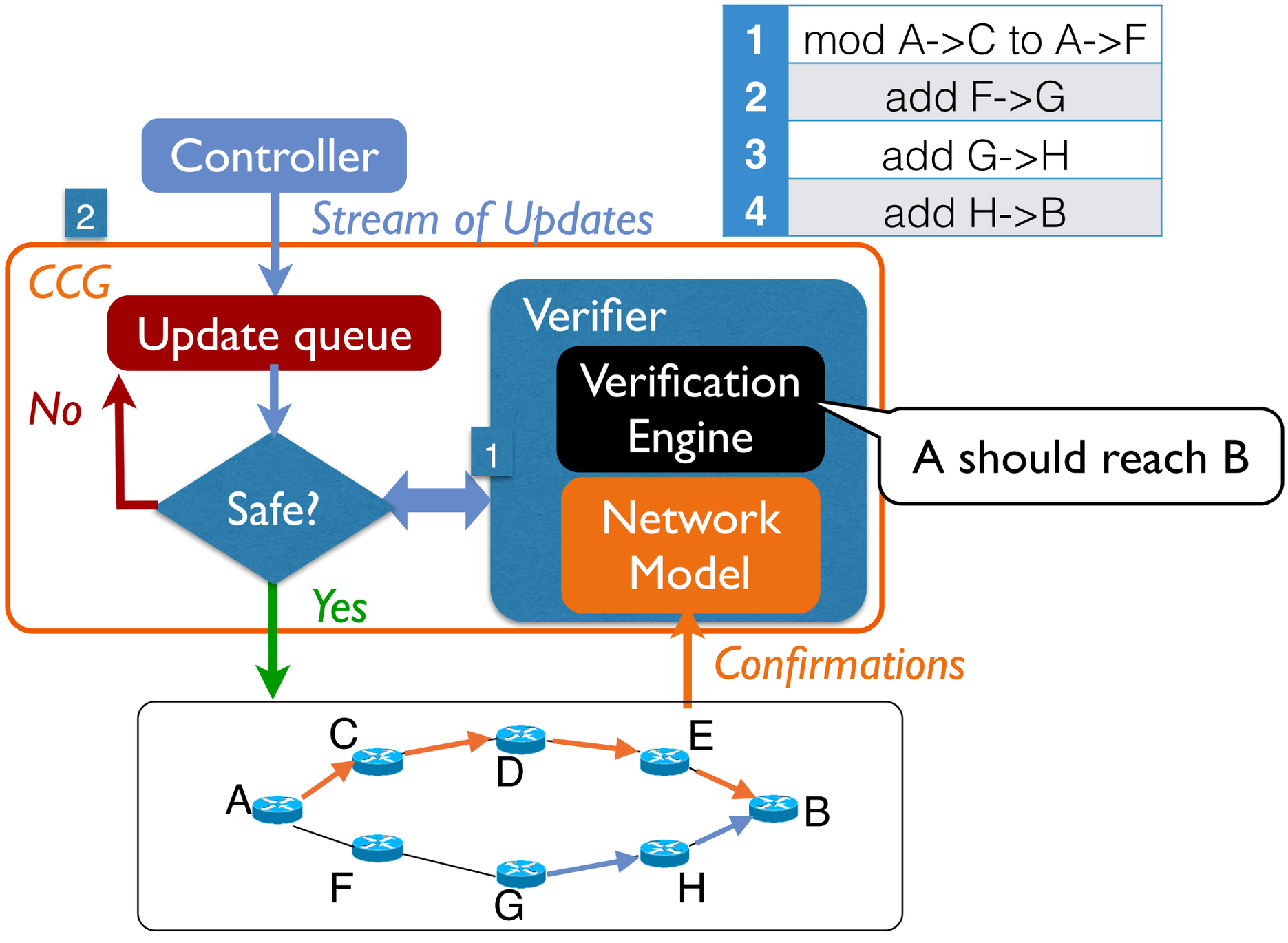


Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B

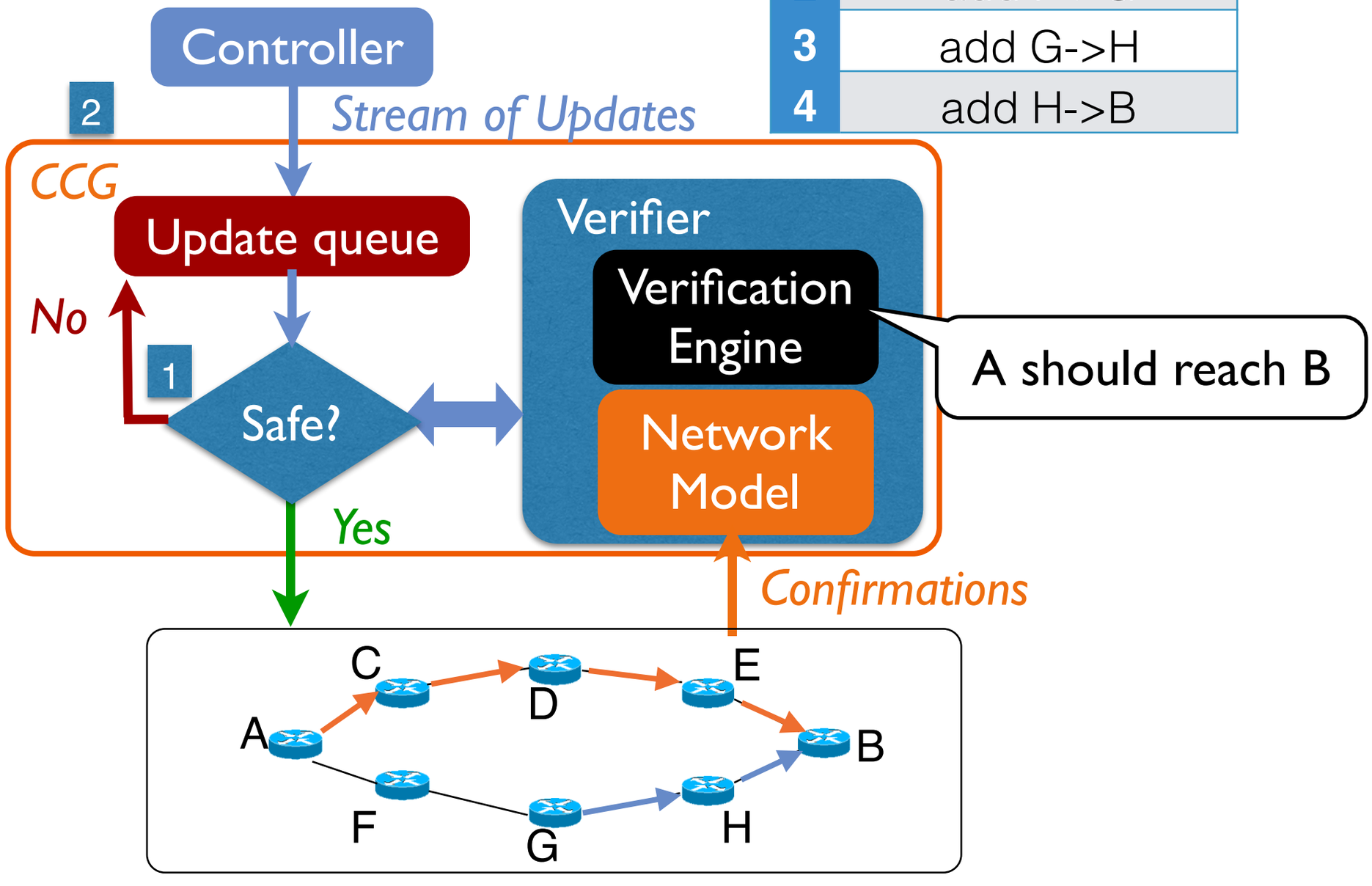


Update synthesis via verification



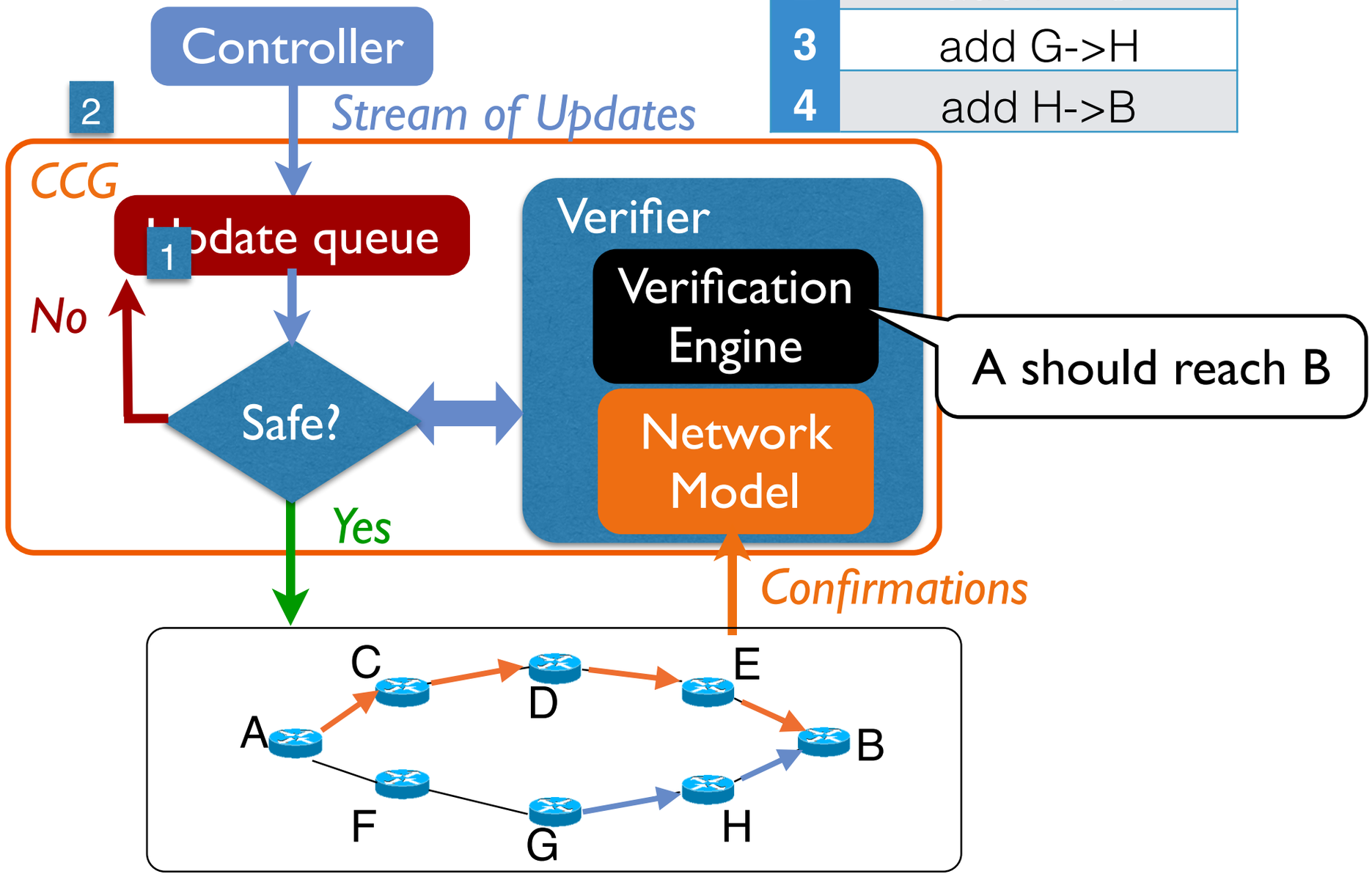
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B

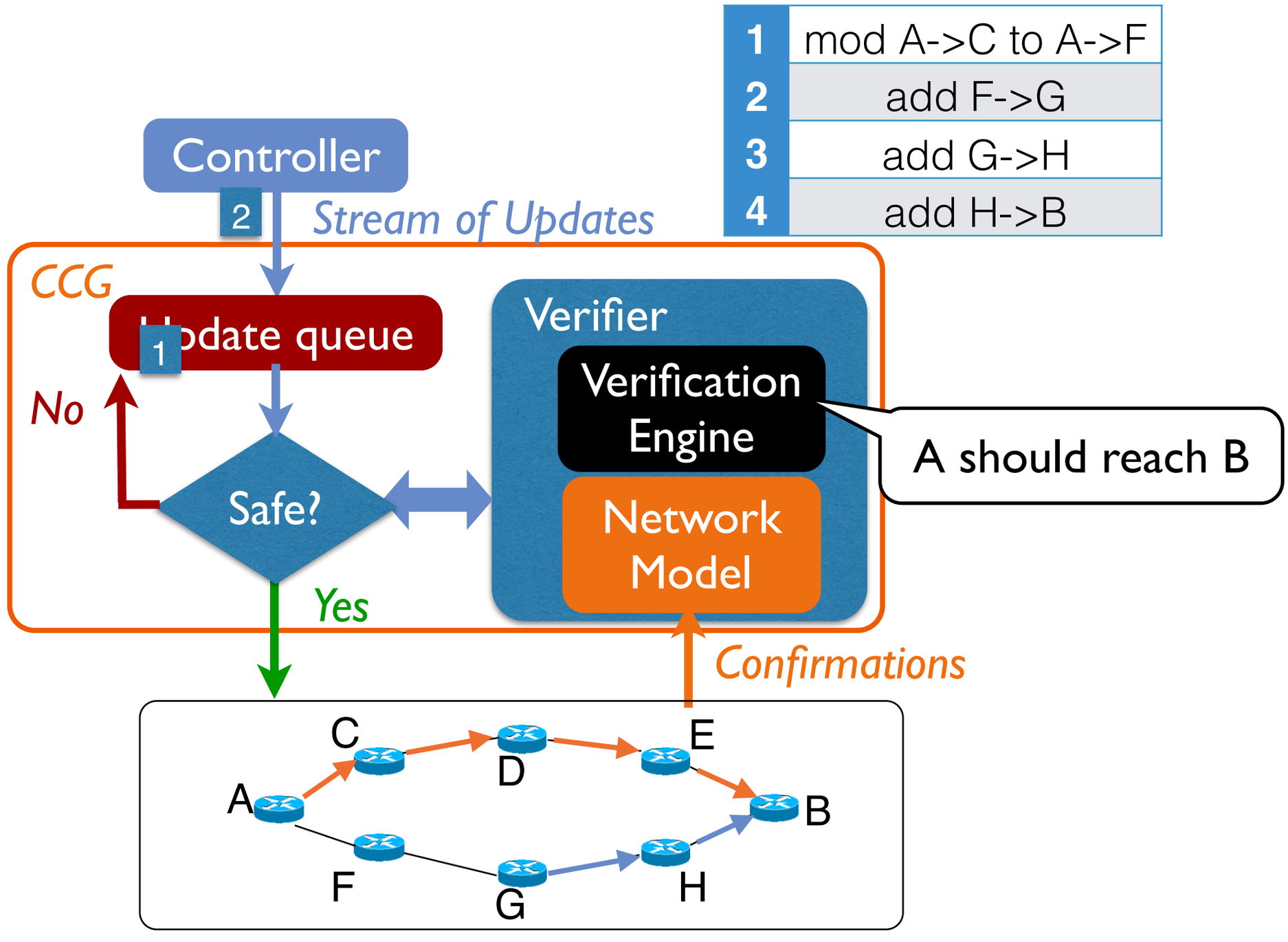


Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B

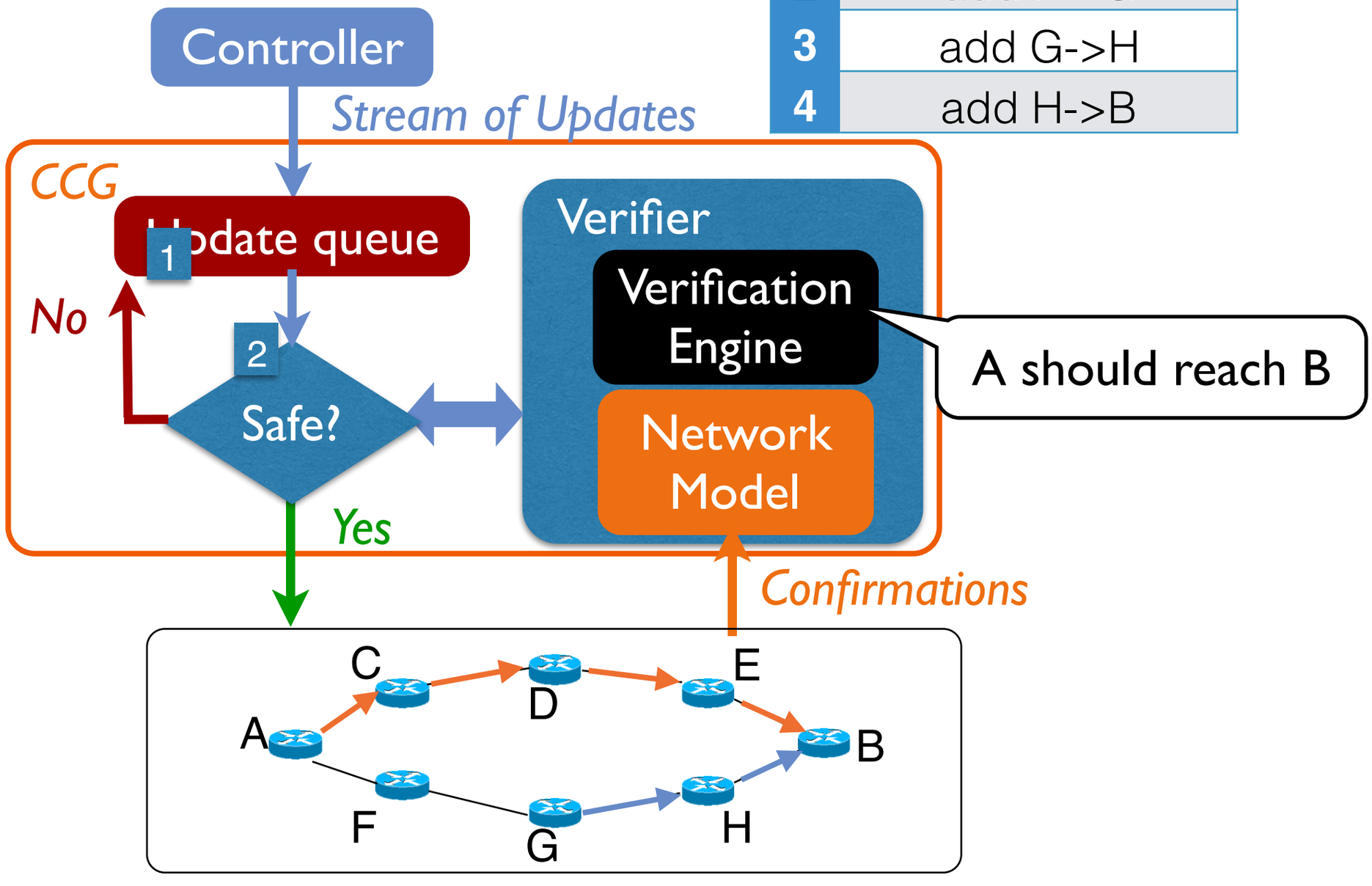


Update synthesis via verification



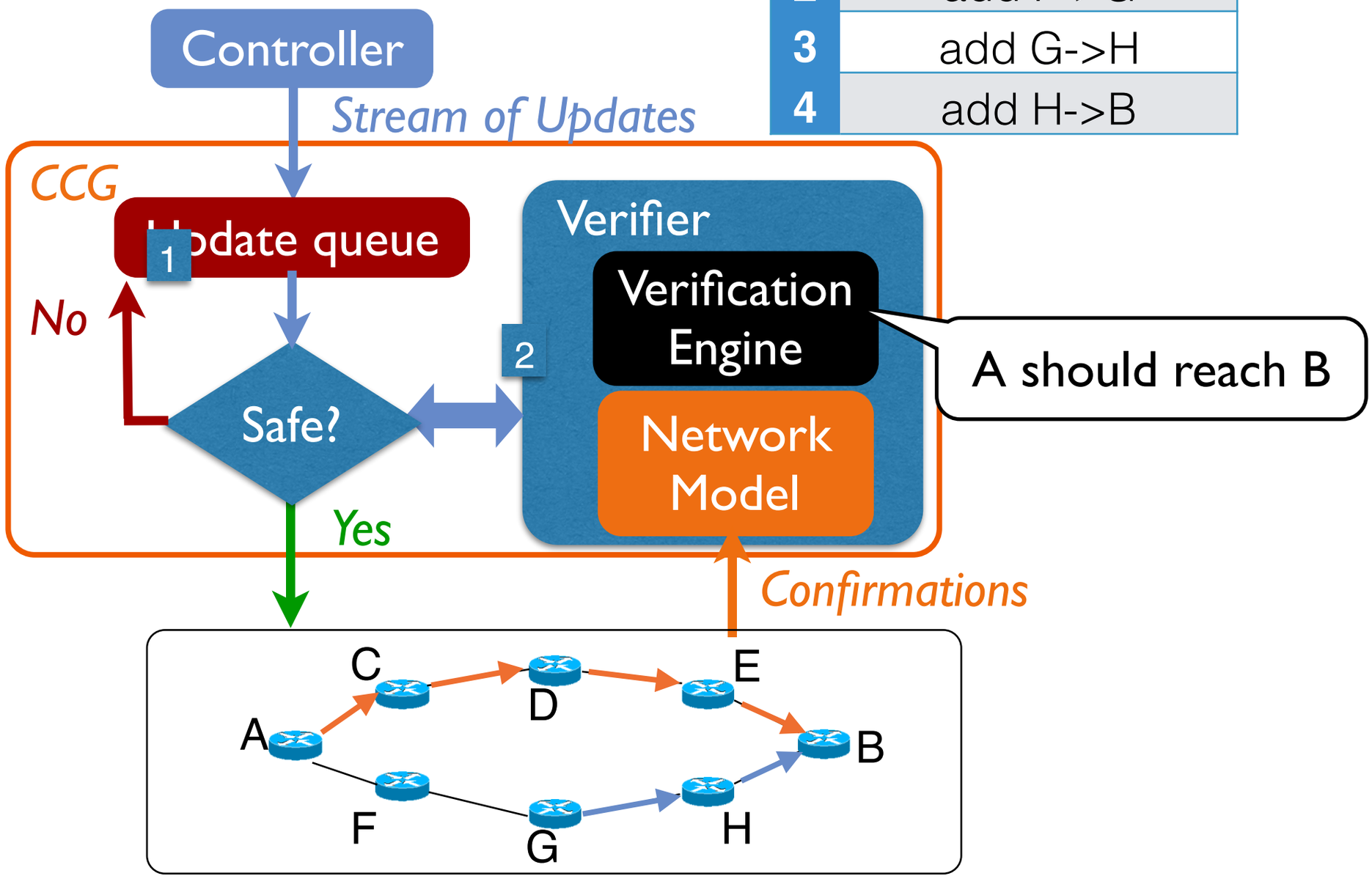
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B



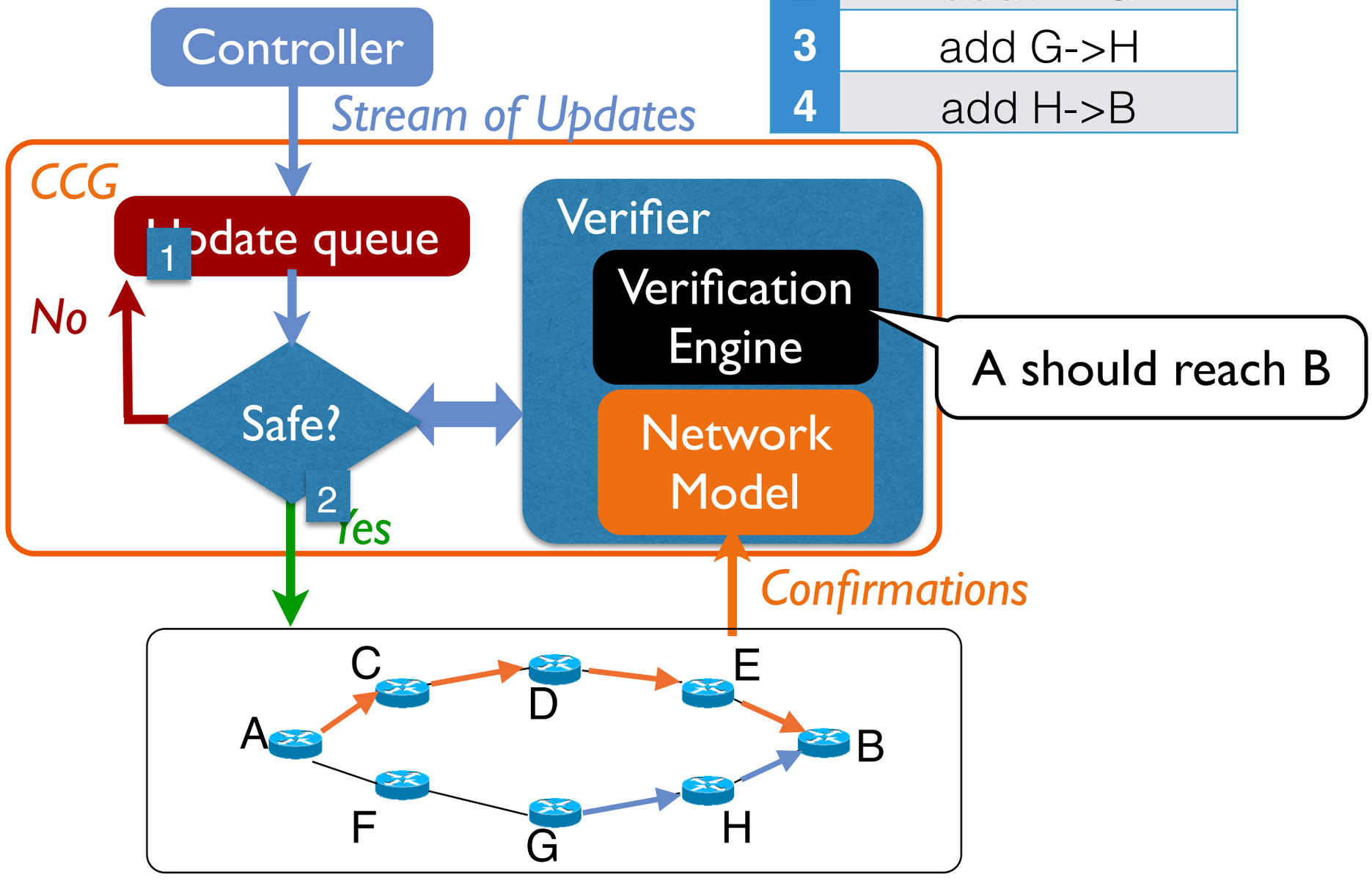
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B



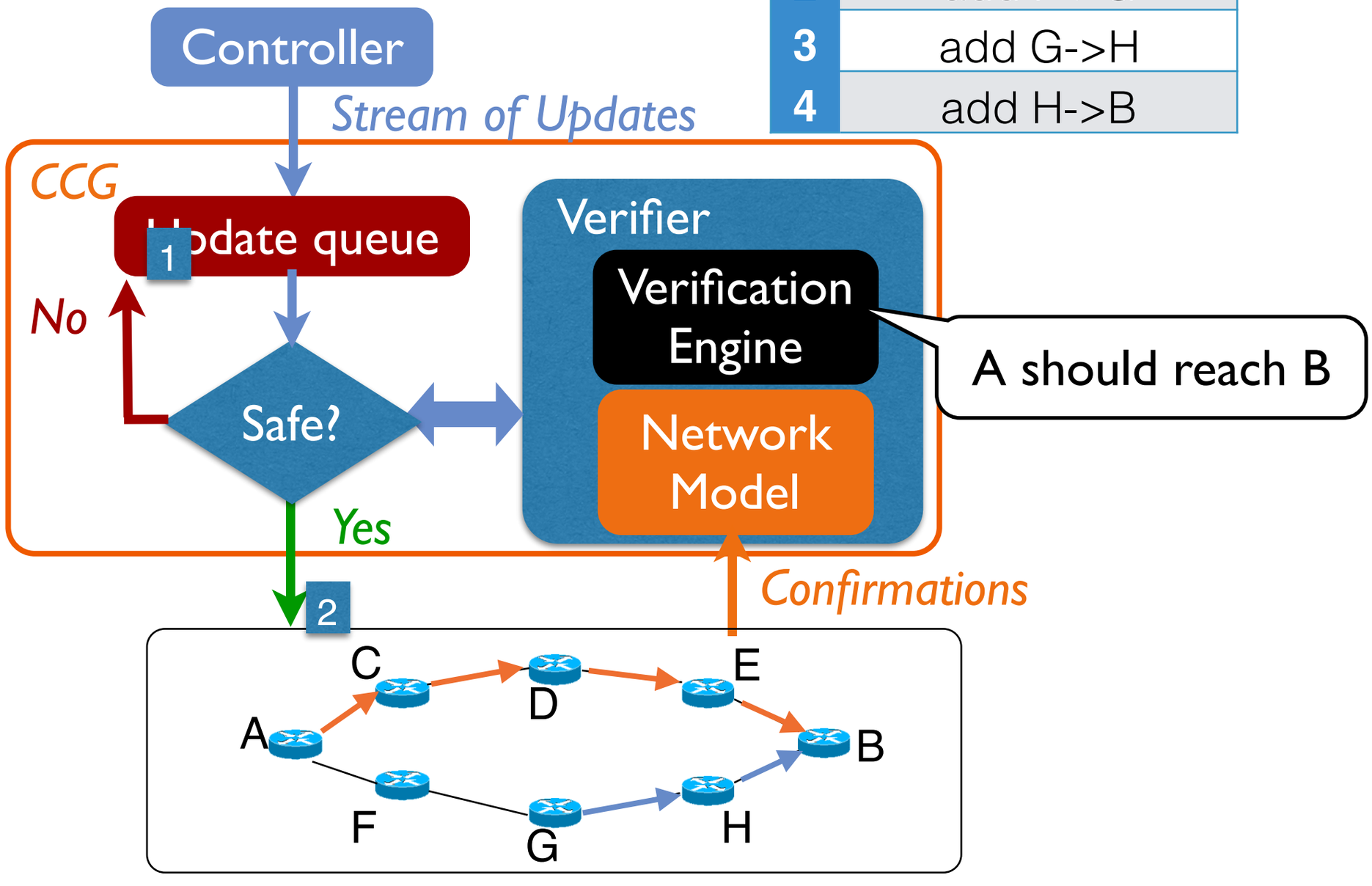
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B



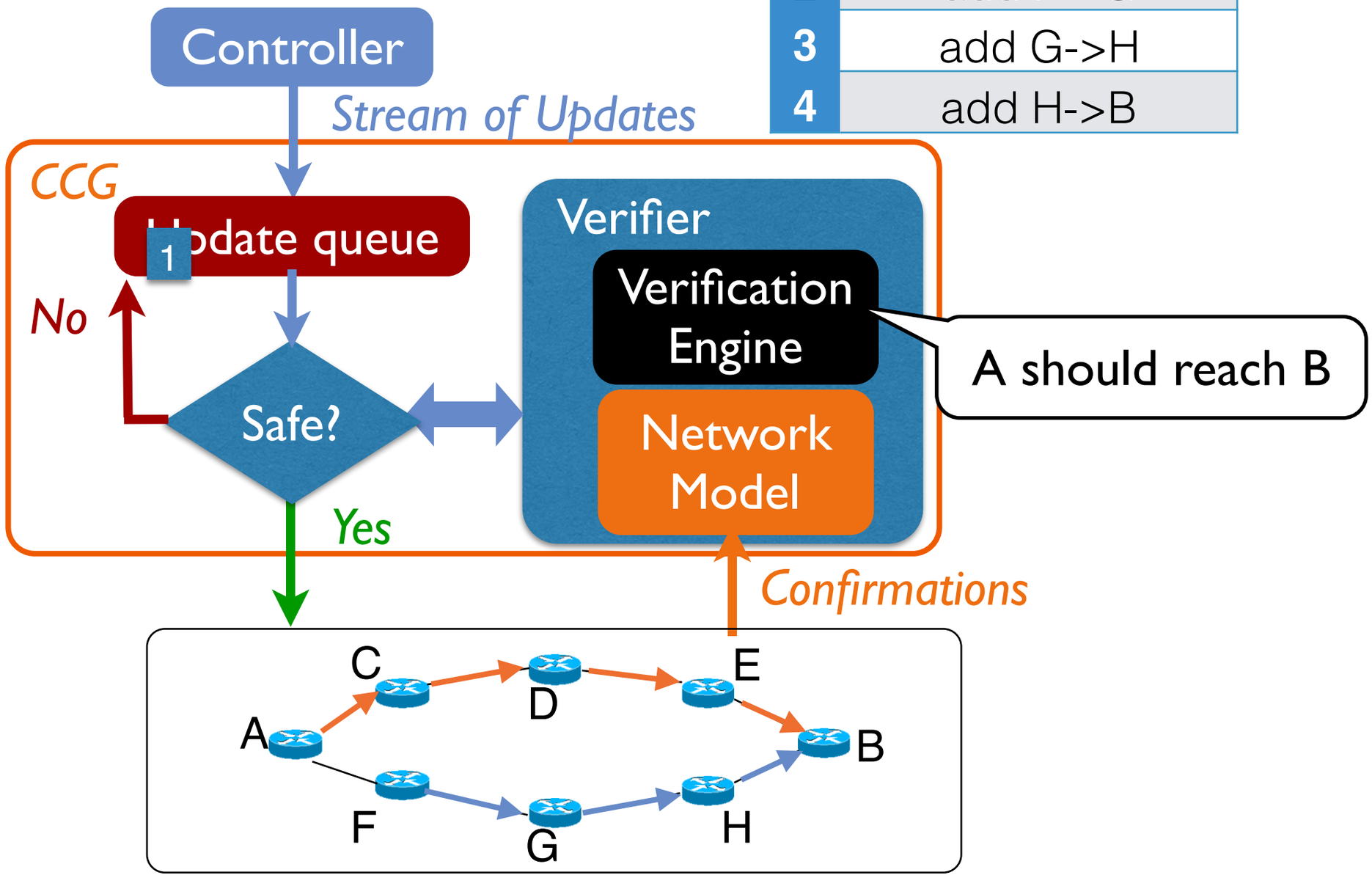
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B



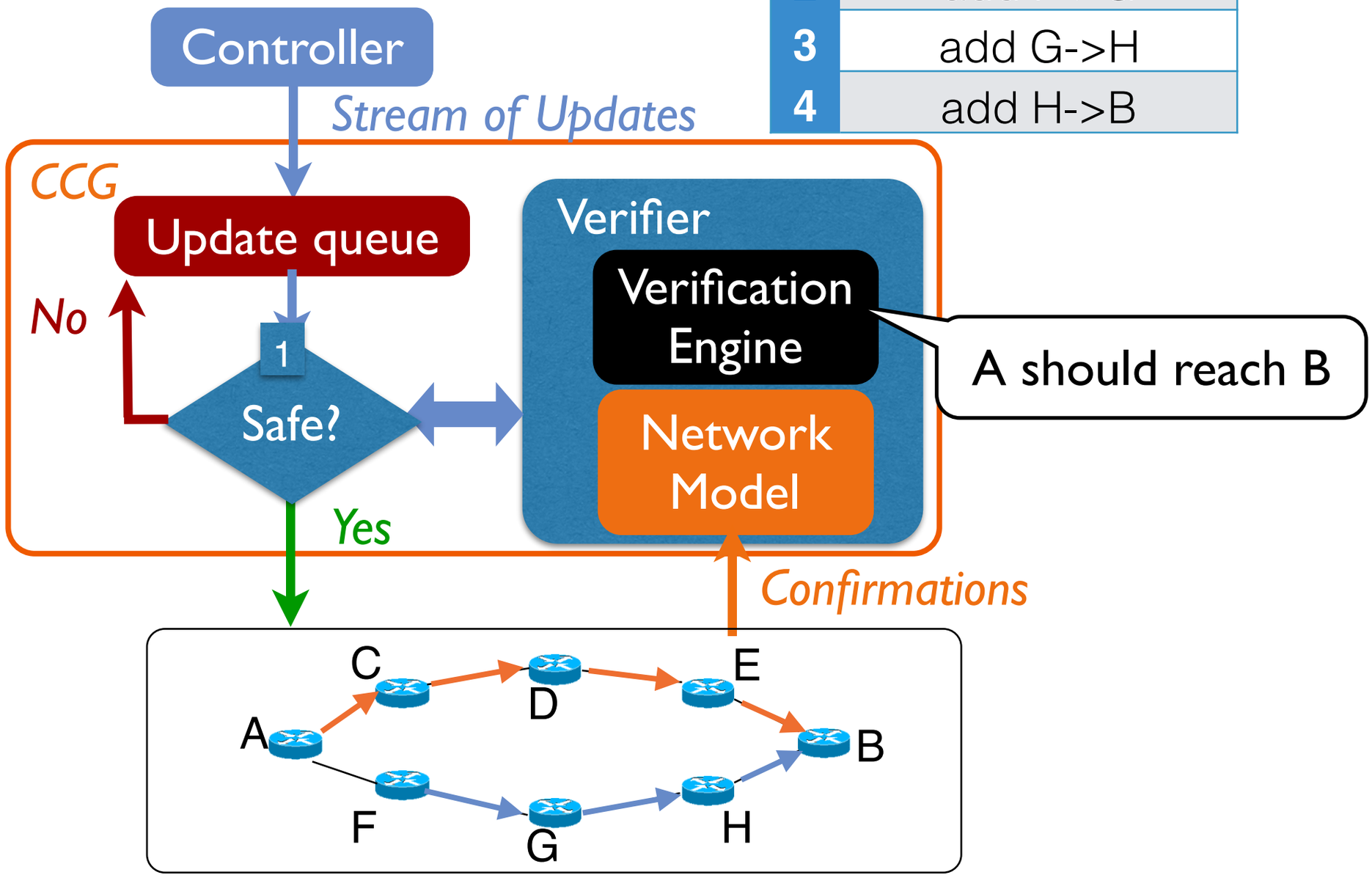
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B

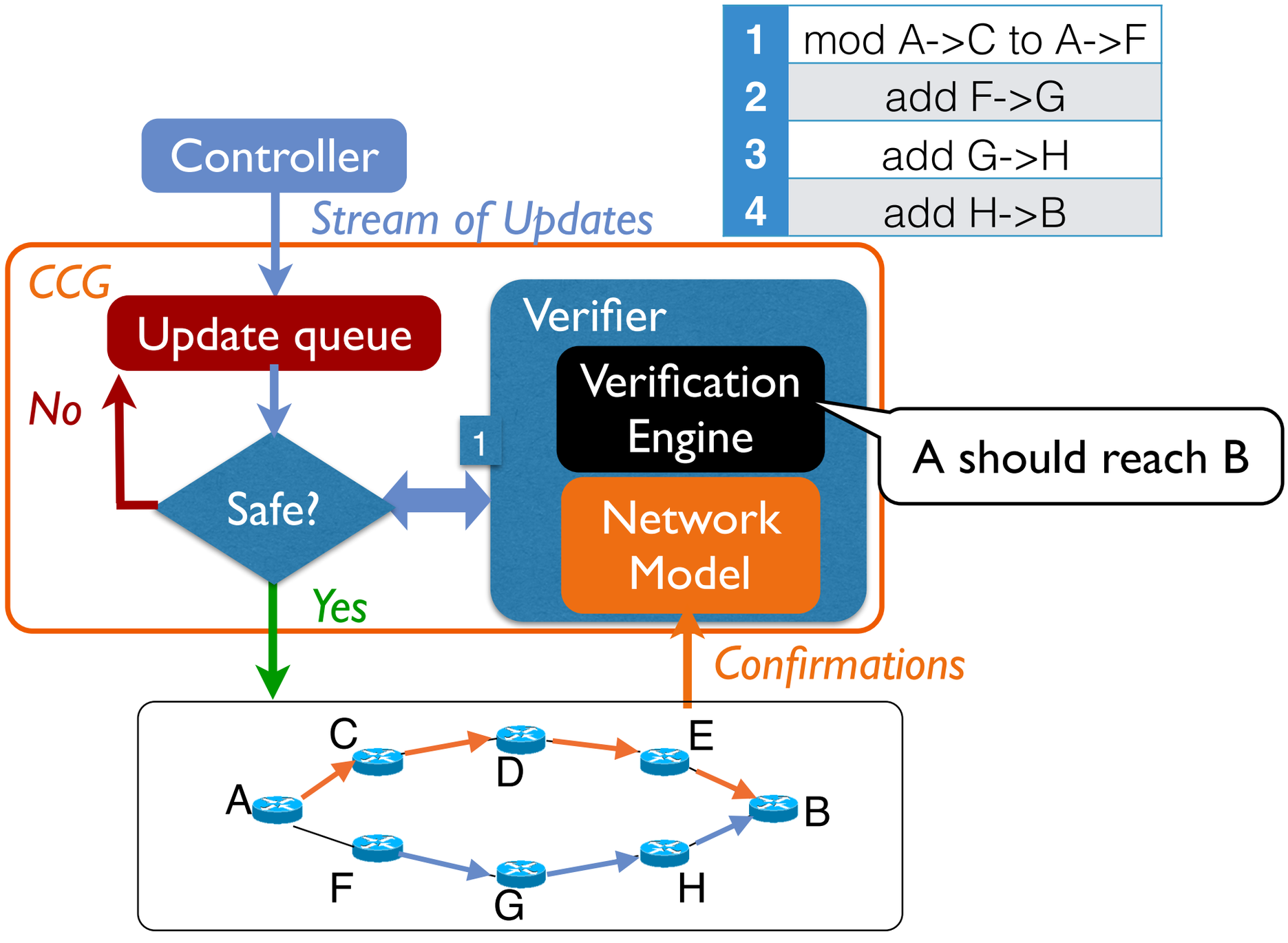


Update synthesis via verification

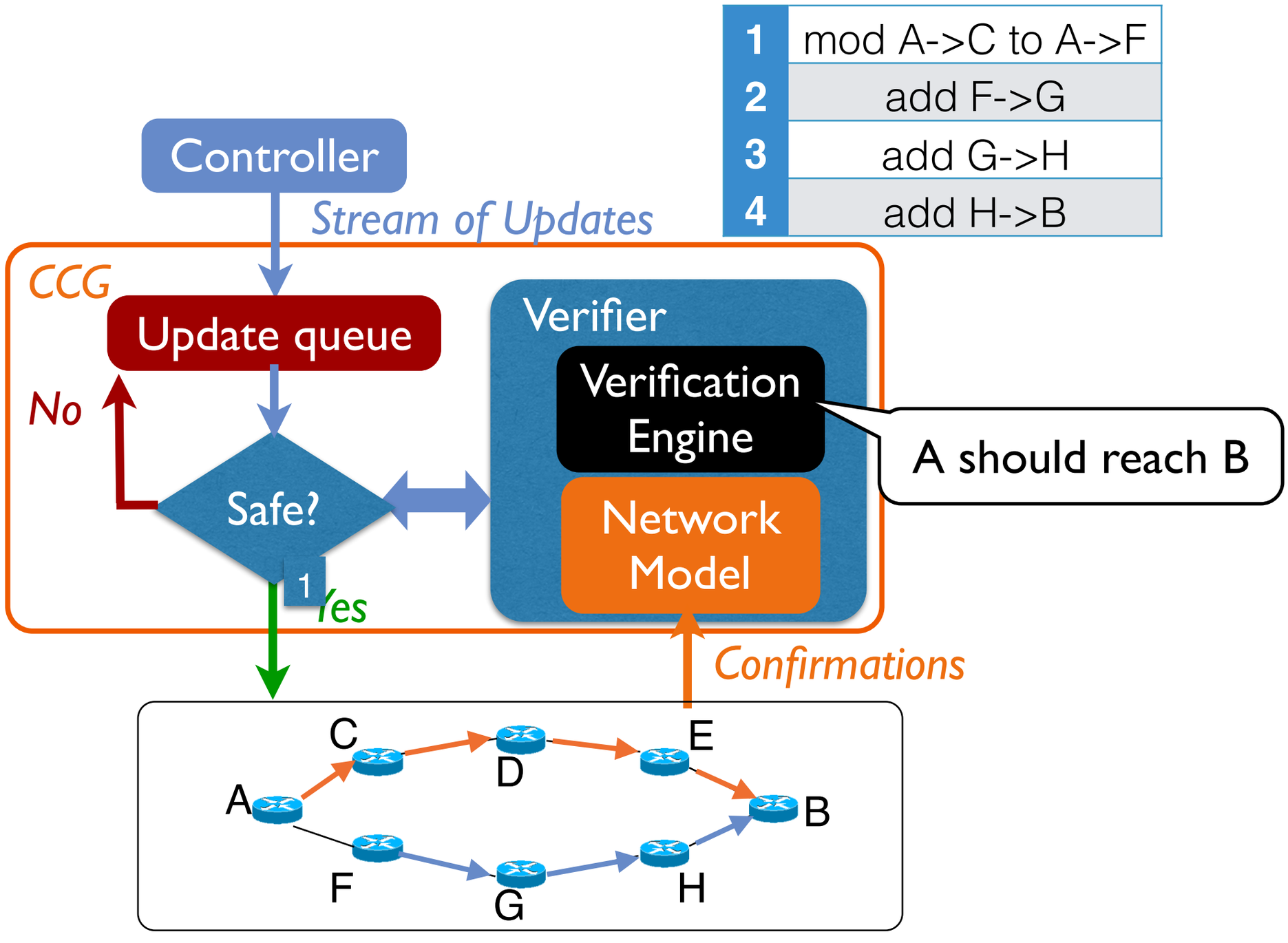
1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B



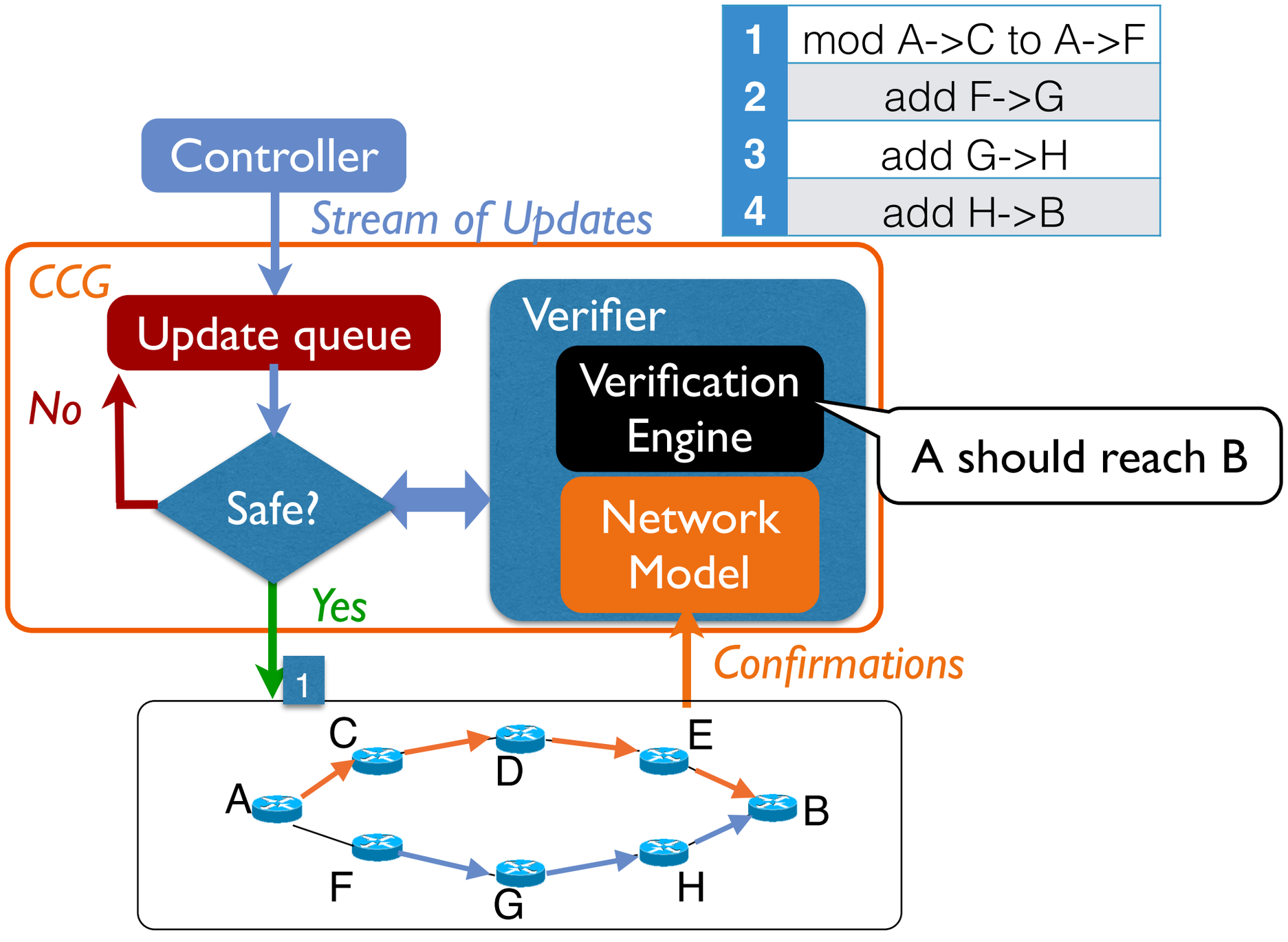
Update synthesis via verification



Update synthesis via verification

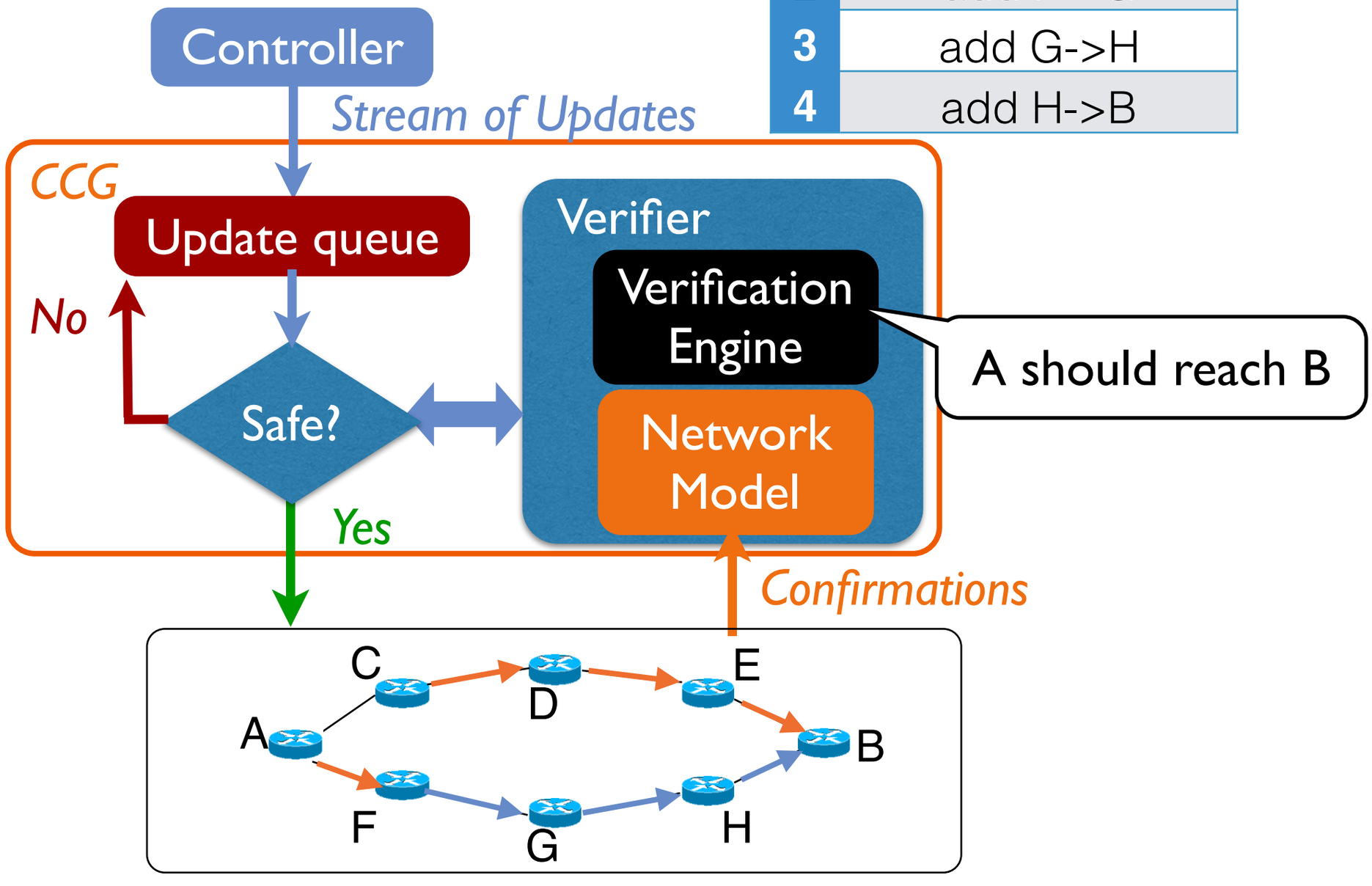


Update synthesis via verification



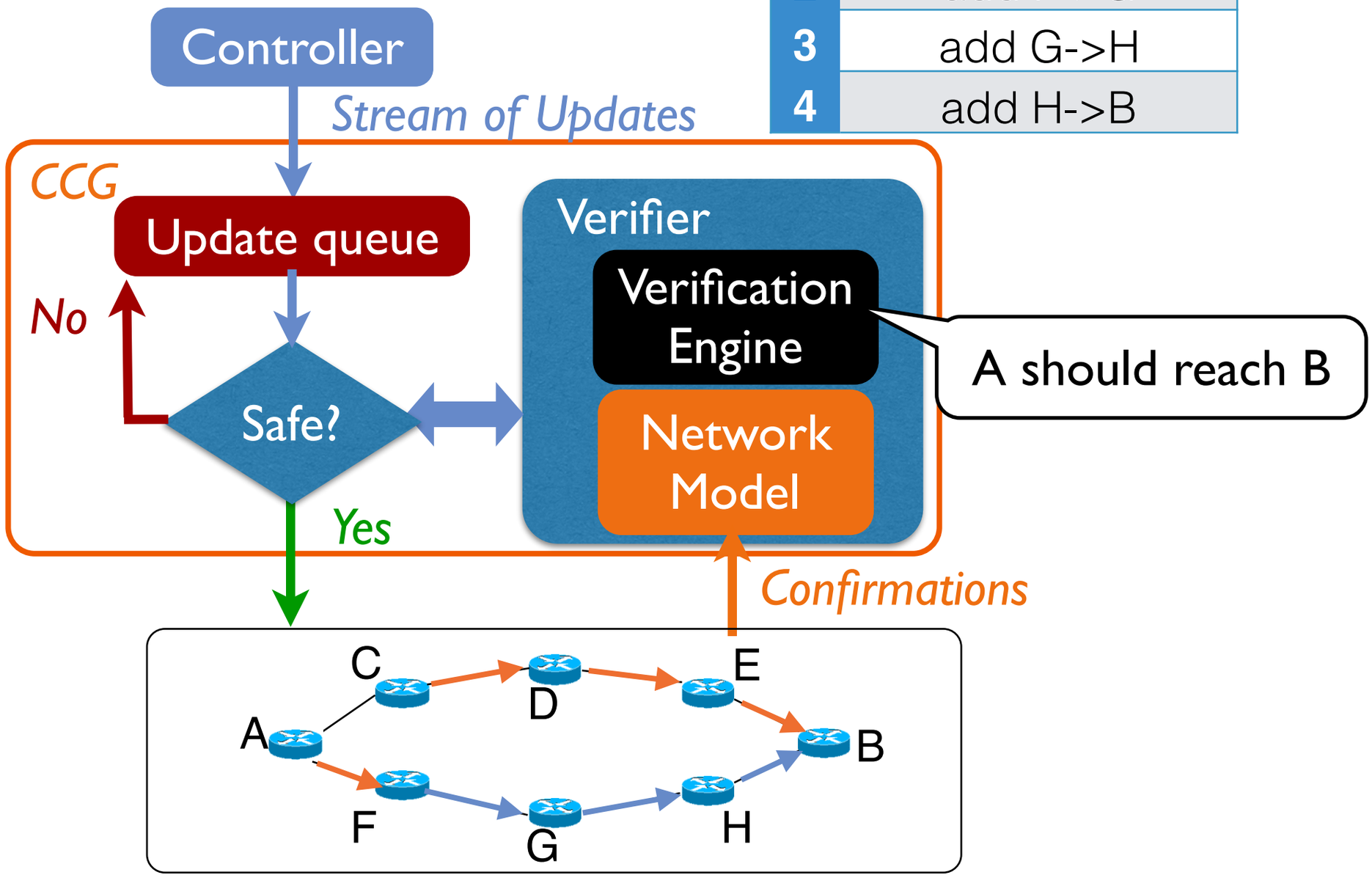
Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B



Update synthesis via verification

1	mod A->C to A->F
2	add F->G
3	add G->H
4	add H->B



Enforcing dynamic correctness with heuristically maximized parallelism

VERIFICATION IN THE REAL WORLD:

WHAT DID WE LEARN?

Industry efforts

Three startups pursuing general-purpose network verification for enterprises

Special purpose efforts

- Hyperscale clouds
- Major network device manufacturer

Gartner grouping verification in “intent-based networking” category

Industry efforts

Three startups pursuing general-purpose network verification for enterprises

Special purpose efforts

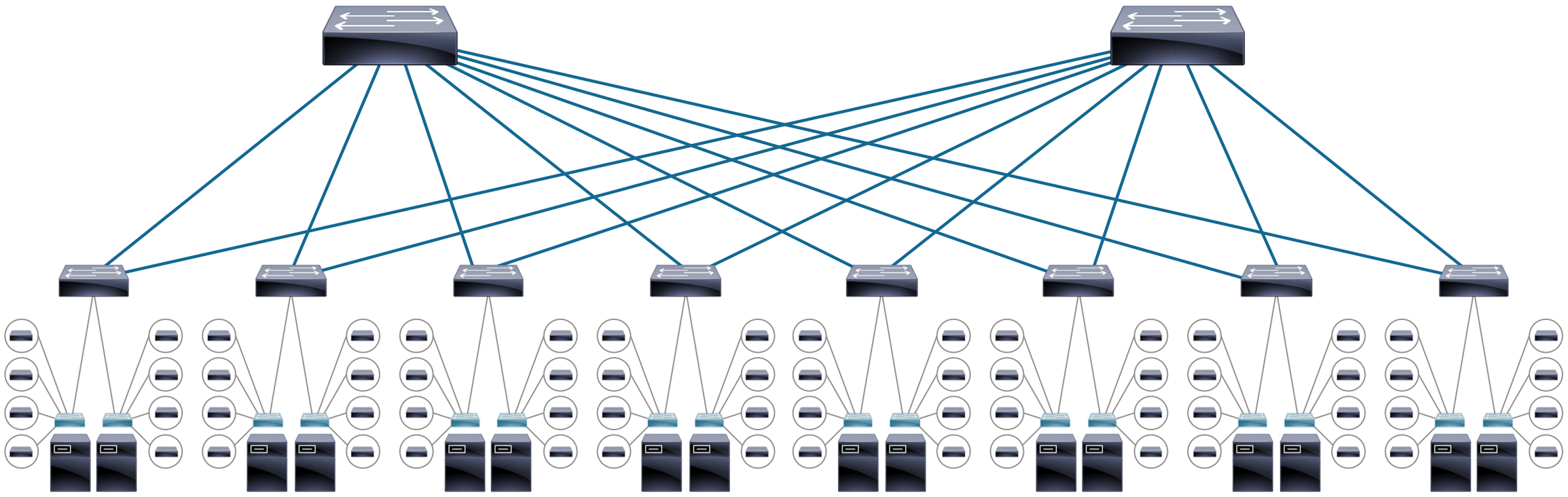
- Hyperscale clouds
- Major network device manufacturer

Gartner grouping verification in “intent-based networking” category

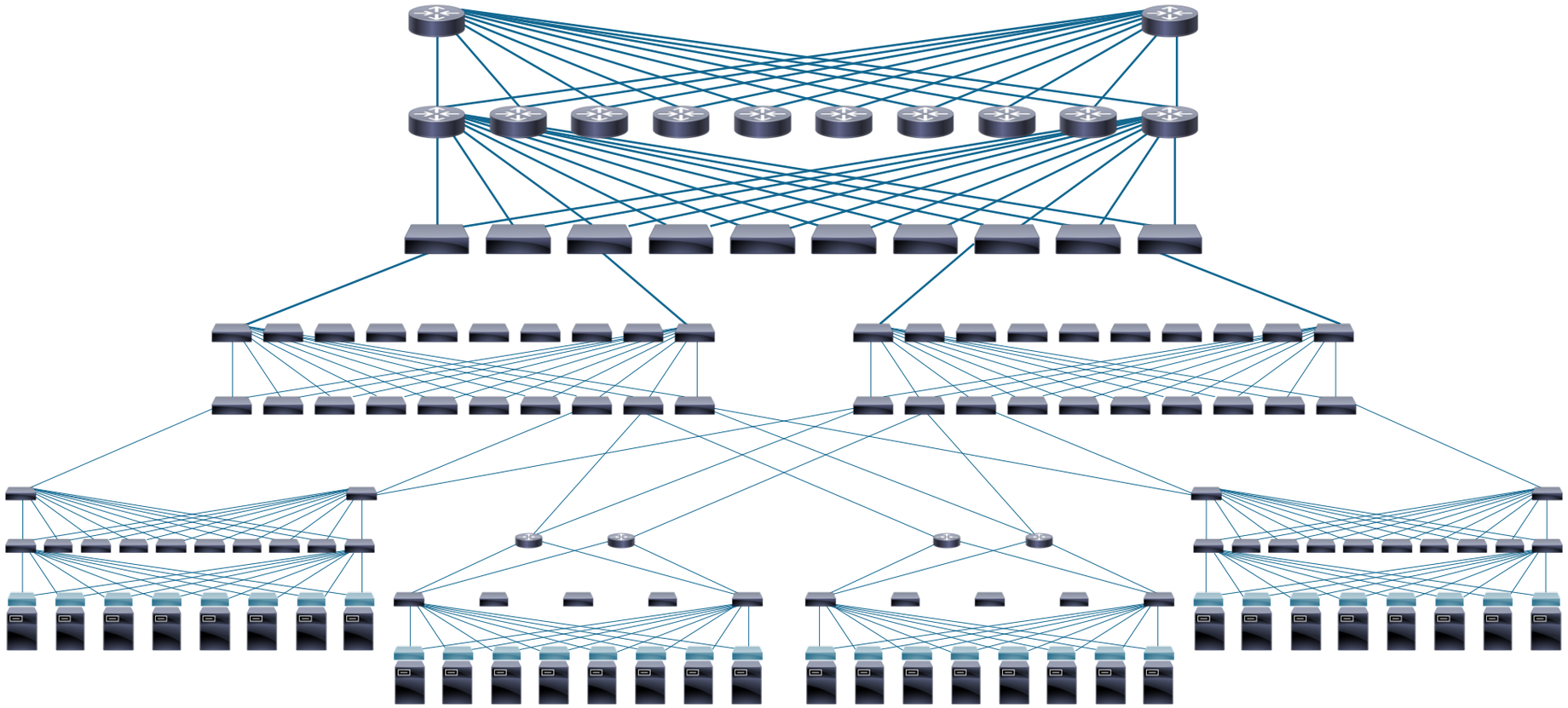
What have we learned?

VERIFLOW

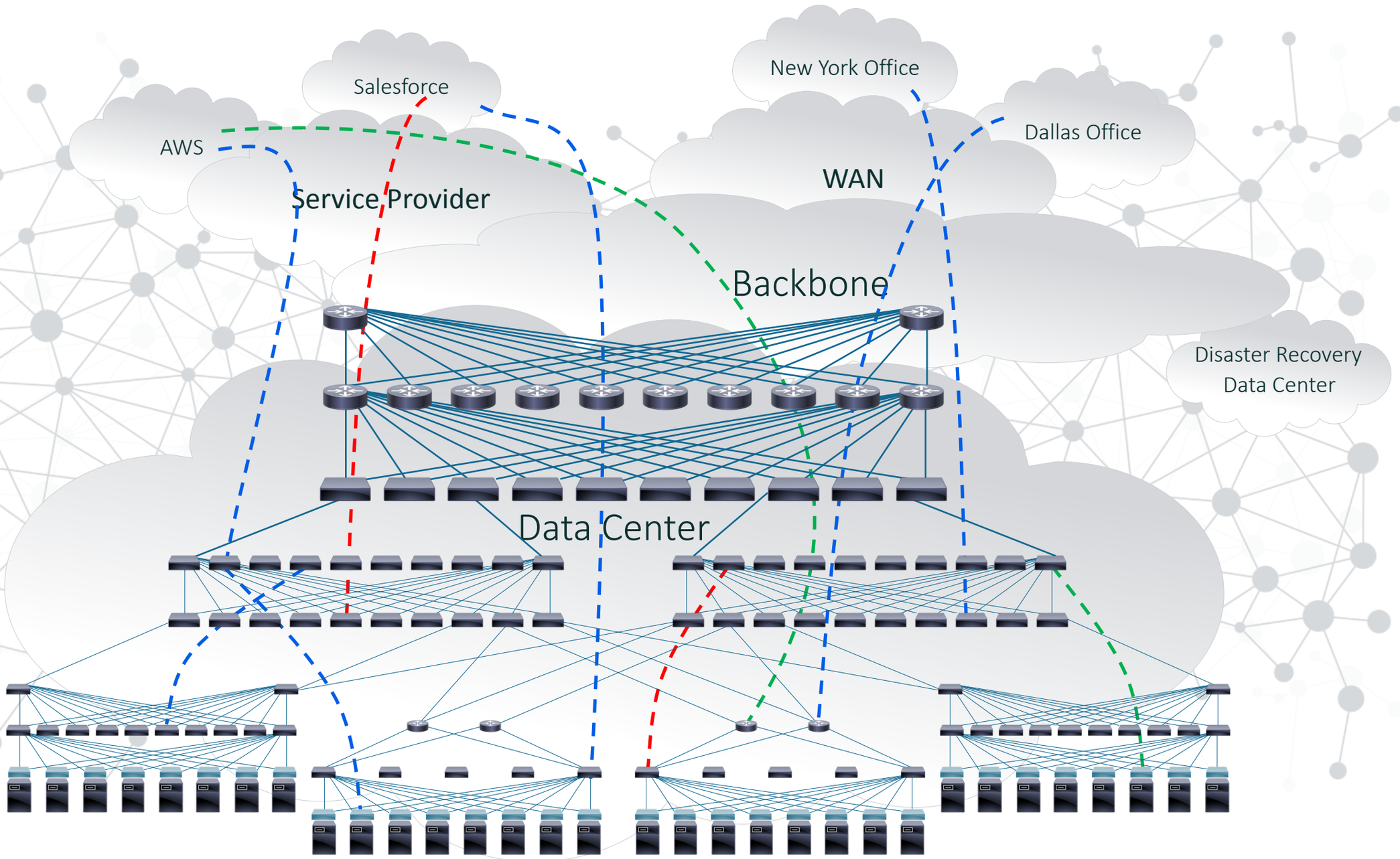
1. The Need is Real



1. The Need is Real



1. The Need is Real



1. The Need is Real

Network

Complexity

VERIFLOW

1. The Need is Real

Network Complexity

59% say growth in complexity has led to more frequent outages [Dimensional Research]

VERIFLOW

1. The Need is Real

Network Complexity

59% say growth in complexity has led to more frequent outages [Dimensional Research]

Change

1. The Need is Real

Network Complexity

59% say growth in complexity has led to more frequent outages [Dimensional Research]

Change

22,000 changes/mo. at DISA [S. Zabel, 2016]

1. The Need is Real

Network Complexity

59% say growth in complexity has led to more frequent outages [Dimensional Research]

Change

22,000 changes/mo. at DISA [S. Zabel, 2016]

Manual Processes

1. The Need is Real

Network Complexity

59% say growth in complexity has led to more frequent outages [Dimensional Research]

Change

22,000 changes/mo. at DISA [S. Zabel, 2016]

Manual Processes

69% use manual checks (most common technique) [Dimensional Research]

2. How is it actually useful?

2. How is it actually useful?

Availability & Resilience

2. How is it actually useful?

Availability & Resilience

Network Segmentation

VERIFLOW

2. How is it actually useful?

Availability & Resilience

Network Segmentation

Continuous Compliance

VERIFLOW

2. How is it actually useful?

Availability & Resilience

Network Segmentation

Continuous Compliance

Incident Response

VERIFLOW

3. Extracting the abstraction: not easy

3. Extracting the abstraction: not easy

Software verification

```
#include <stdio.h>

int main(int argc, char** argv) {
   if (argv >= 2) {
      printf("Hello world, %s!",
         argv[1]);
   }
   return 0;
}
```

- Given program as input
- Assume formal specification of programming language

VERIFLOW

3. Extracting the abstraction: not easy

Software verification

```
#include <stdio.h>

int main(int argc, char** argv) {
   if (argv >= 2) {
      printf("Hello world, %s!",
         argv[1]);
   }
   return 0;
}
```

- Given program as input
- Assume formal specification of programming language

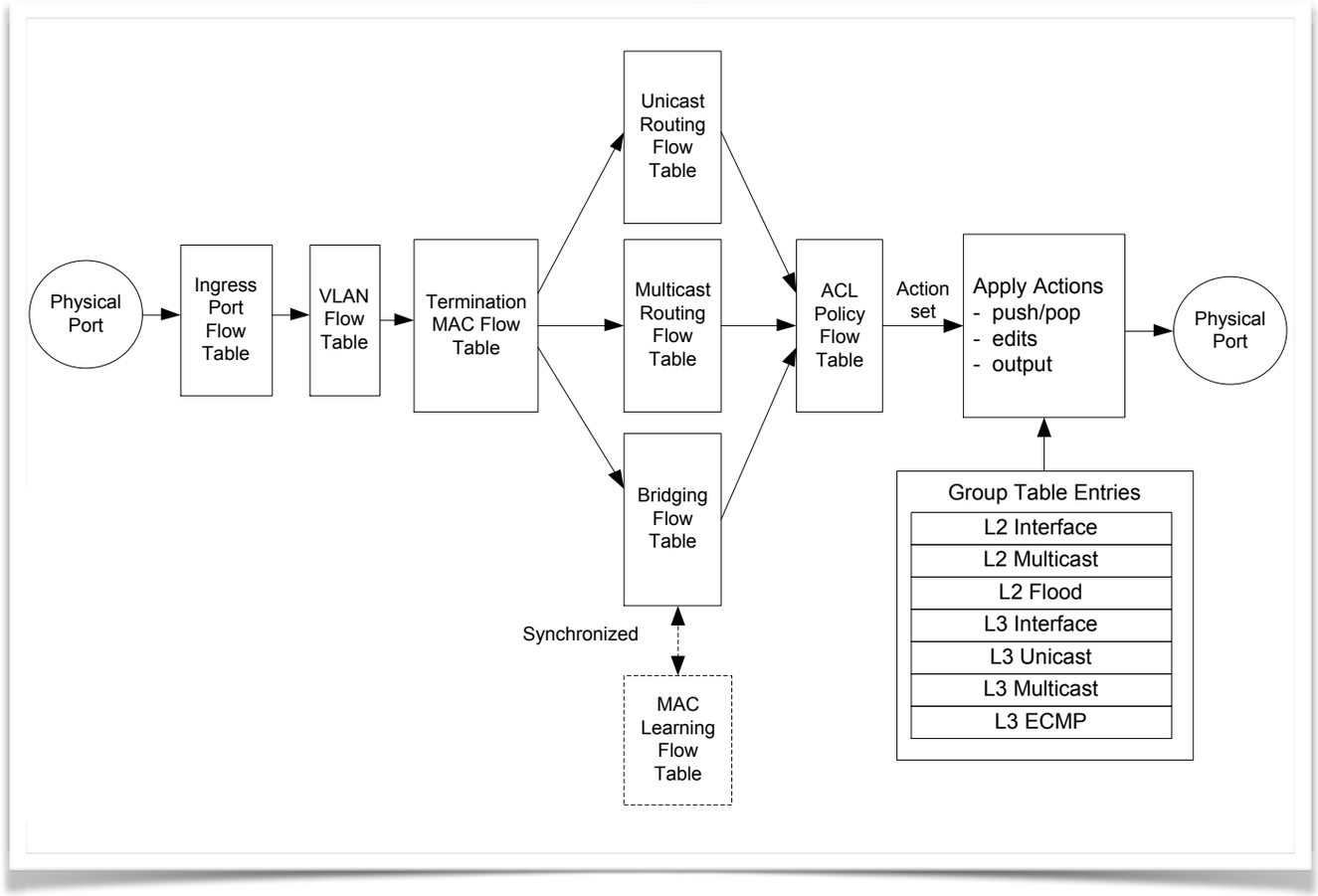
Data plane verification

- No universal API to extract state (LCD: SSH + CLI "show" commands")
- No formal spec of how that state relates to functionality
- Vendor-specific behaviors

3. Extracting the abstraction: not easy

Data plane verification

- No universal API to extract state (LCD: SSH + CLI “show” commands”)
- No formal spec of how that state relates to functionality
- Vendor-specific behaviors

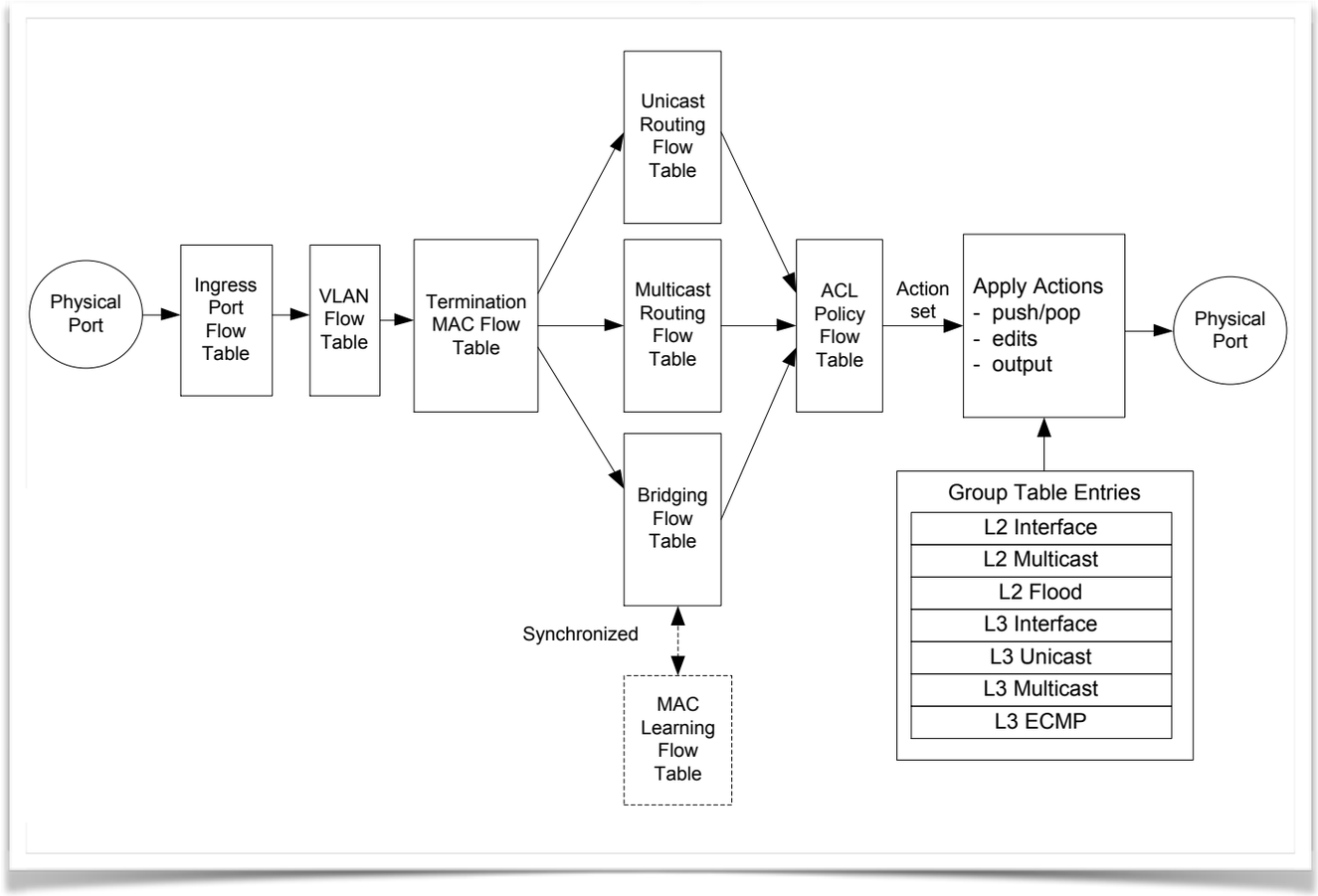


Broadcom’s OF-DPA 1.0 Abstract Switch
https://www.ietf.org/proceedings/90/slides/slides-90-sdnrg-3.pdf

3. Extracting the abstraction: not easy

Data plane verification

- No universal API to extract state (LCD: SSH + CLI “show” commands”)
- No formal spec of how that state relates to functionality
- Vendor-specific behaviors



Broadcom’s OF-DPA 1.0 Abstract Switch
https://www.ietf.org/proceedings/90/slides/slides-90-sdnrg-3.pdf

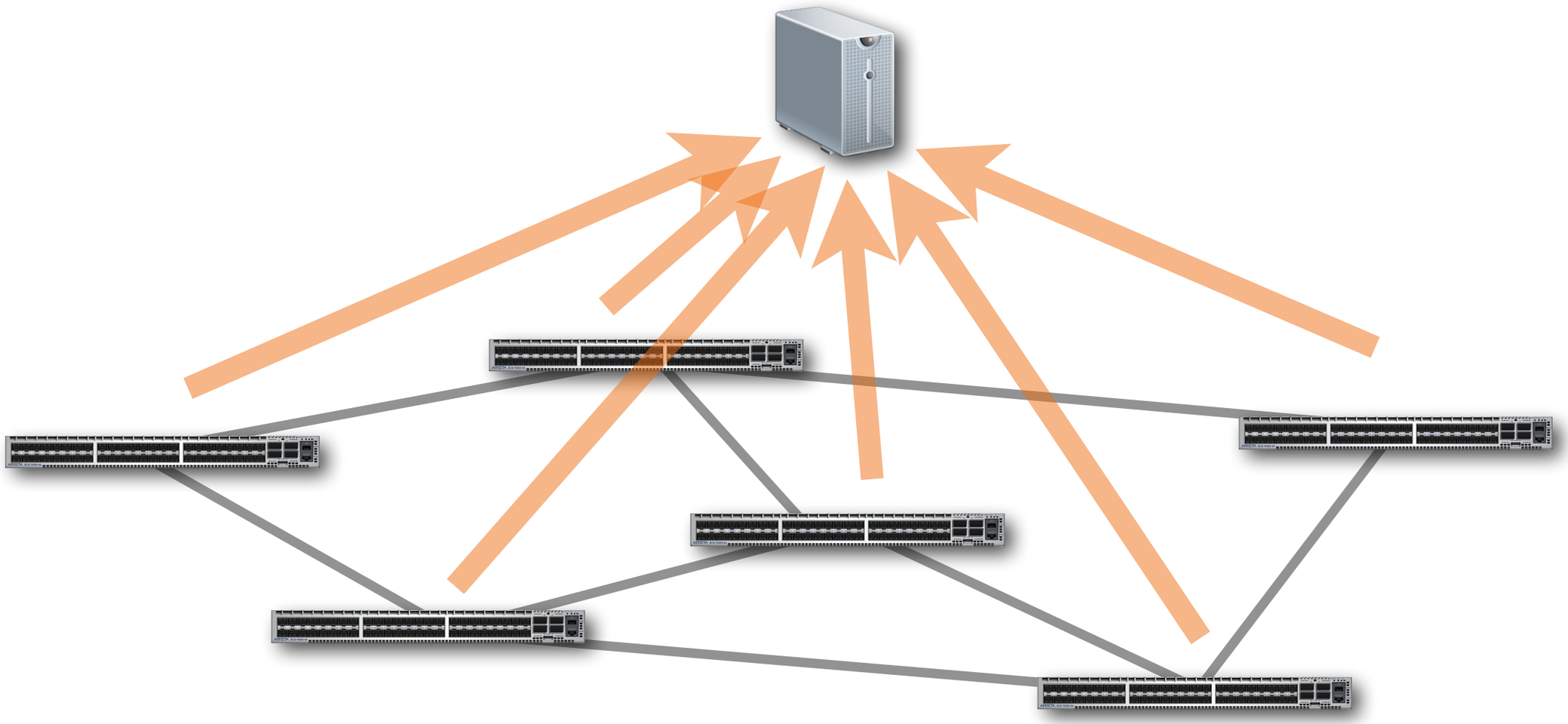
- Some hope: Vendor-specific APIs, OpenConfig

VERIFLOW

4. Model / Verifier separation works

verify

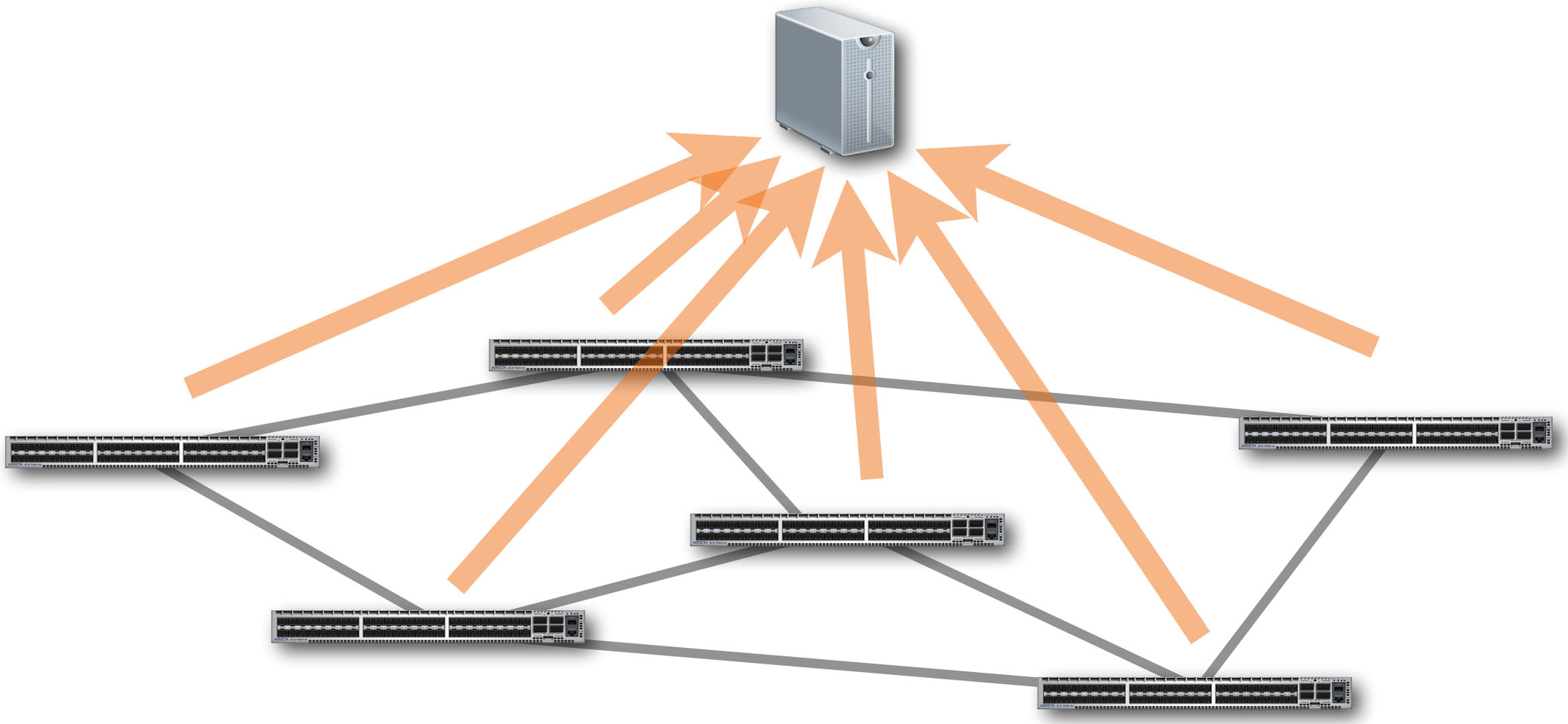
Predictive model
("forwarding graph")



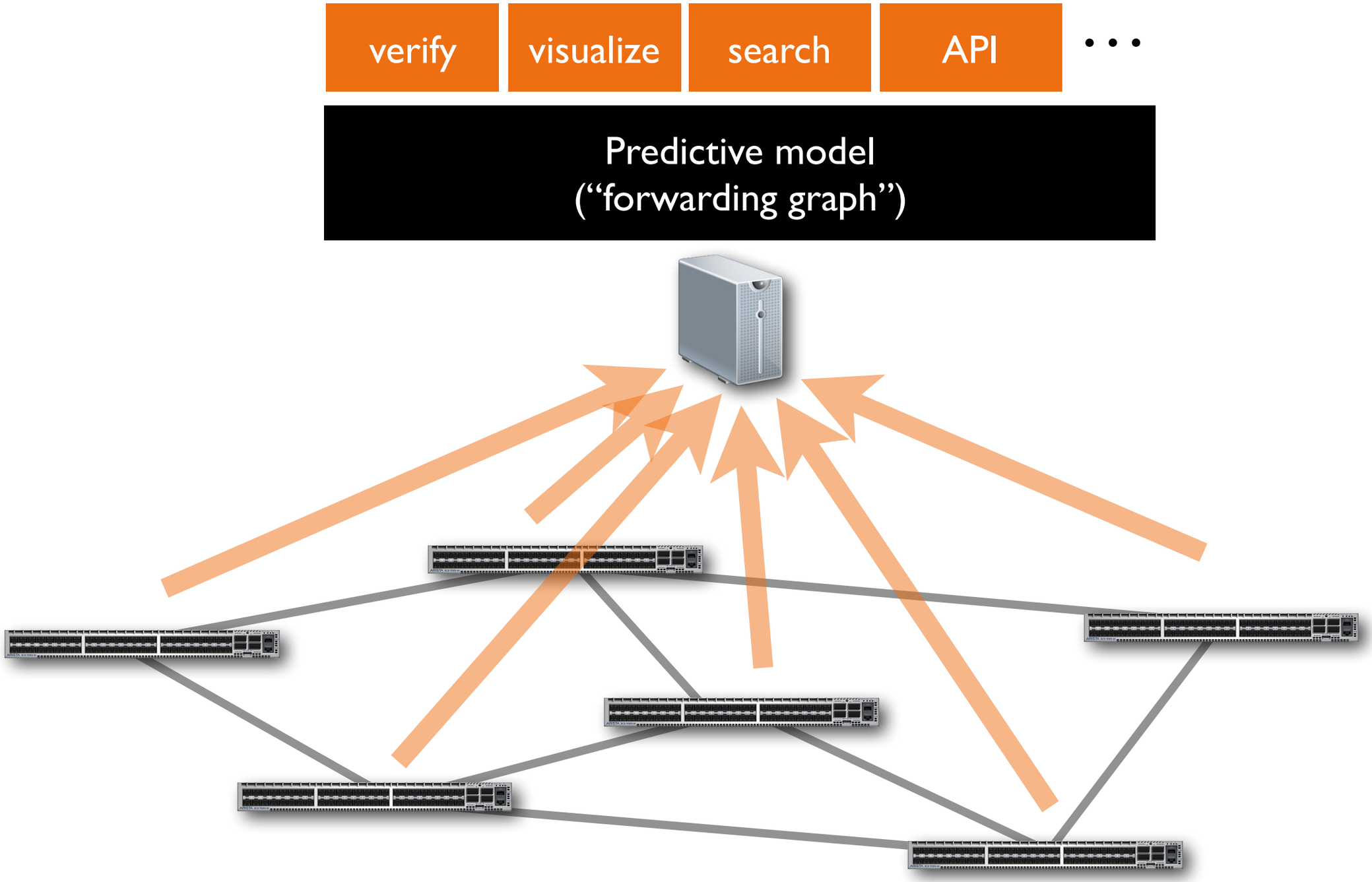
4. Model / Verifier separation works

verify

Predictive model
("forwarding graph")



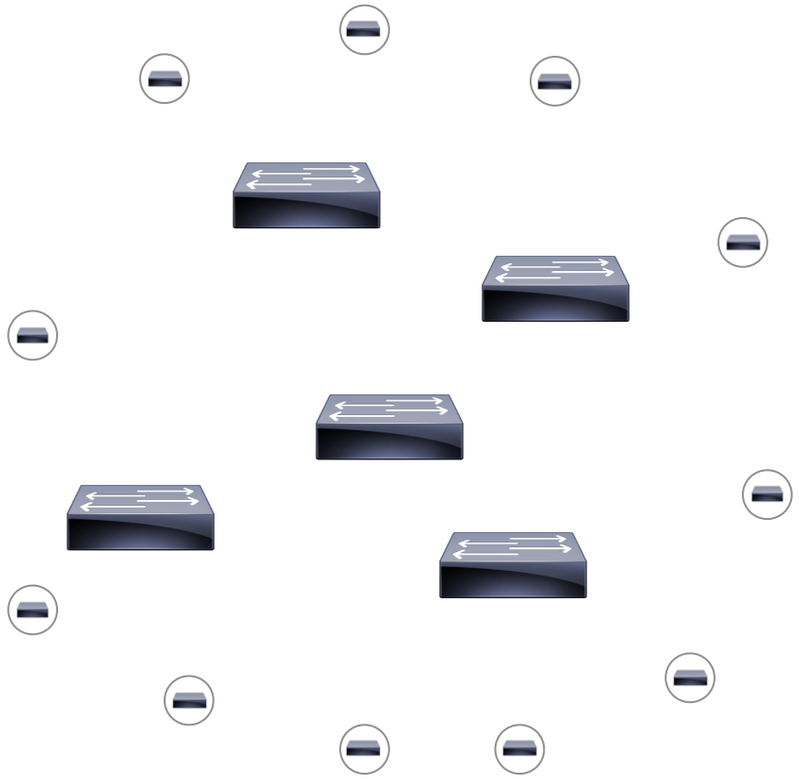
4. Model / Verifier separation works



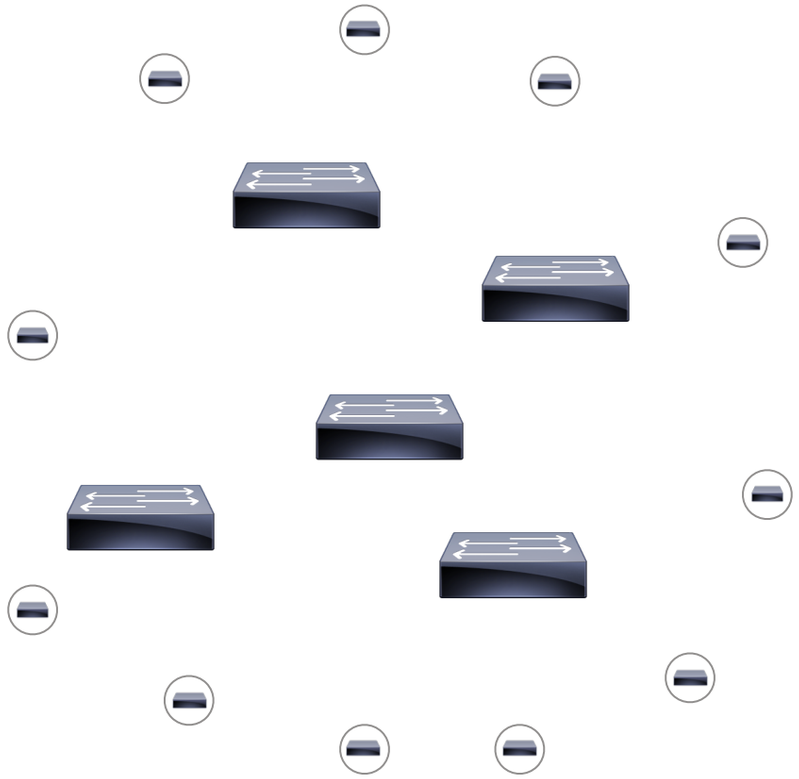
VERIFLOW

5. We need a shift in thought

5. We need a shift in thought



5. We need a shift in thought

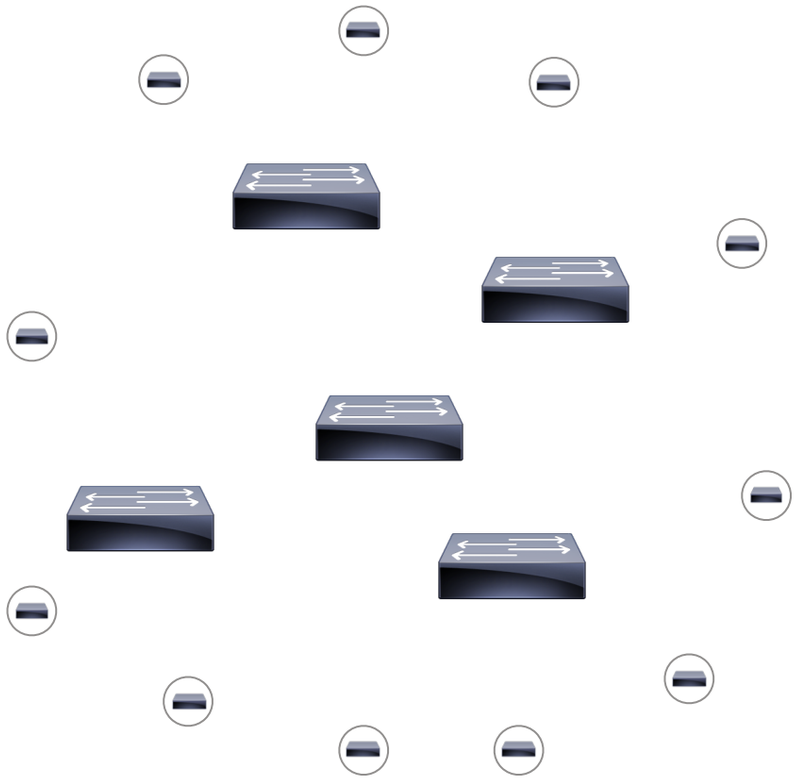


Network as individual devices

Individual config knobs

VERIFLOW

5. We need a shift in thought

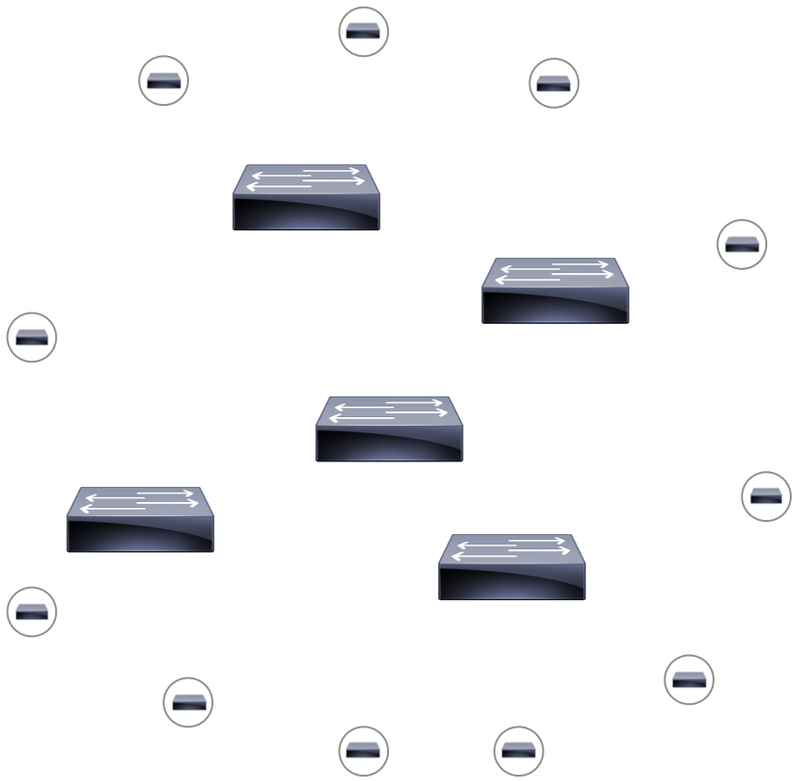


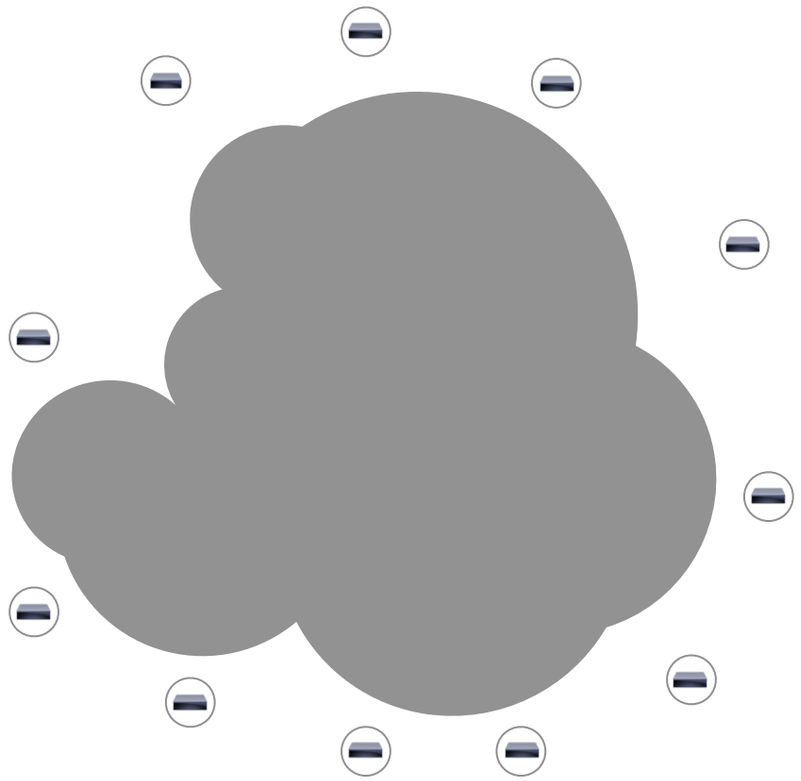
Network as individual devices

Individual config knobs

VERIFLOW

5. We need a shift in thought





Network as individual devices → Network as one system

Individual config knobs → End-to-end intent

VERIFLOW

Data plane verification

Static

- On static reachability in IP networks [Xie, Zhan, Maltz, Zhang, Greenberg, Hjalmtysson, Rexford, INFOCOM ’05]
  - Essentially early form of data plane verification
  - Computed reachable sets with IP forwarding rules
- FlowChecker [Al-Shaer, Al-Haj, SafeConfig ’10]
- Anteater [Mai, Khurshid, Agarwal, Caesar, G., King, SIGCOMM’11]
- Header Space Analysis [Kazemian, Varghese, and McKeown, NSDI ’12]
- Network-Optimized Datalog (NoD) [Lopes, Bjørner, Godefroid, Jayaraman, Varghese, NSDI 2015]

Real time (incremental)

- VeriFlow [Khurshid, Zou, Zhou, Caesar, G., HotSDN’12, NSDI’13]
- NetPlumber [Kazemian, Chang, Zeng, Varghese, McKeown, Whyte, NSDI ’13]
- CCG [Zhou, Jin, Croft, Caesar, G., NSDI’15]

Data plane verification (cont'd)

Optimizations

- Libra: Divide and Conquer to Verify Forwarding Tables in Huge Networks [Zeng, Zhang, Ye, Google, Jeyakumar, Ju, Liu, McKeown, Vahdat, NSDI'14]
- Atomic Predicates [Yang, Lam, ToN'16]
- ddNF [Bjorner, Juniwal, Mahajan, Seshia, Varghese, HVC'16]

Configuration verification

Configuration verification

- RCC (Detecting BGP config faults w/static analysis)
  [Feamster & Balakrishnan, USENIX ’05]
- ConfigAssure [Narain et al, ’08]
- ConfigChecker [Al-Shaer, Marrero, El-Atawy, ICNP ‘09]
- Batfish [Fogel, Fung, Pedrosa, Walraed-Sullivan, Govindan, Mahajan, Millstein, NSDI’15]
- Bagpipe [Weitz, Woos, Torlak, Ernst, Krishnamurthy, Tatlock, NetPL’16 & OOPSLA’16]

Richer verification

Richer data plane models

- Software Dataplane Verification [Dobrescu, Argyraki, NSDI'14]
- SymNet [Stoenescu, Popovici, Negreanu, Raiciu, SIGCOMM'16]
- Mutable datapaths [Panda, Lahav, Argyraki, Sagiv, Shenker, NSDI'17]

Verifiable controllers & control languages

- NICE [Canini, Venzano, Perešíni, Kostić, Rexford, NSDI'12]
- NetKAT [Anderson, Foster, Guha, Jeannin, Kozen, Schlesinger, Walker, POPL'14]
- Kinetic: Verifiable Dynamic Network Control [Kim, Gupta, Shahbaz, Reich, Feamster, Clark, NSDI'15]

FUTURE RESEARCH DIRECTIONS

FOR VERIFICATION

Research Directions

Research Directions

1. Pushing the limit towards richer models

    - Software pipelines
    - Verifiable SDN Controllers
    - Stateful networks
    - Higher layer concepts

Research Directions

1. Pushing the limit towards richer models

   - Software pipelines
   - Verifiable SDN Controllers
   - Stateful networks
   - Higher layer concepts

2. Moving from verification to remediation

Research Directions

1. Pushing the limit towards richer models

   - Software pipelines
   - Verifiable SDN Controllers
   - Stateful networks
   - Higher layer concepts

2. Moving from verification to remediation

3. Making it easy for users to express intent

VERIFLOW