Software Synthesis for Networks

Hossein Hojjat
Rochester Institute of Technology

R·I·T

Khatam University
Pasargad Summer School on Networks and Systems
August 14, 2018

University of Tehran

TU Eindhoven

Formal Methods & Programming Languages

EPFL

Cornell

Rochester Institute of Technology

It is a perfect time for the formal methods and programming languages communities to get more involved in networking research

Jedidiah McClurgh

UNM

Pavol Cerny

Nate Foster

CORNELL UNIVERSITY FOUNDED A.D. 1865

Philipp Rümmer

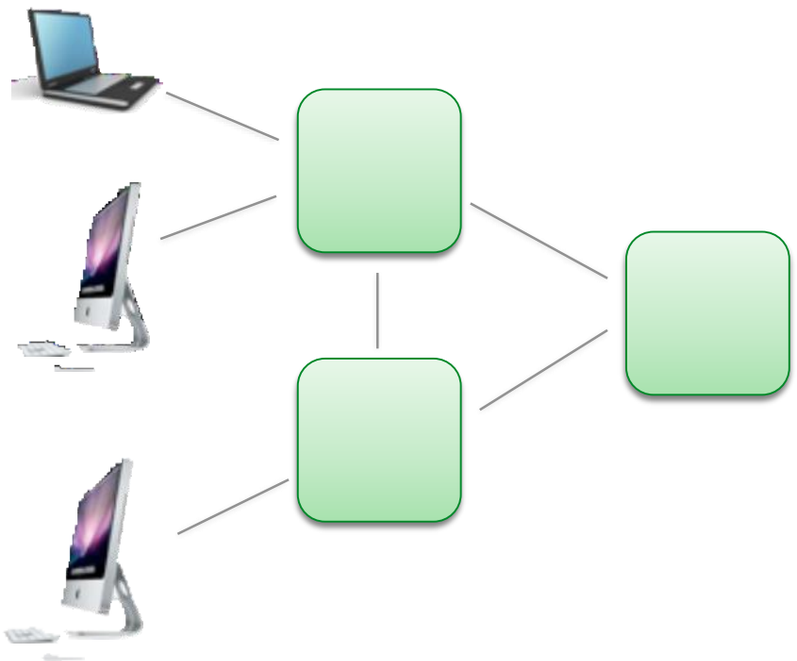
UBSALIENSIS S·ACADEMIAE GRATIAE VERITAS NATURAE

Conventional Networking

There are hosts...

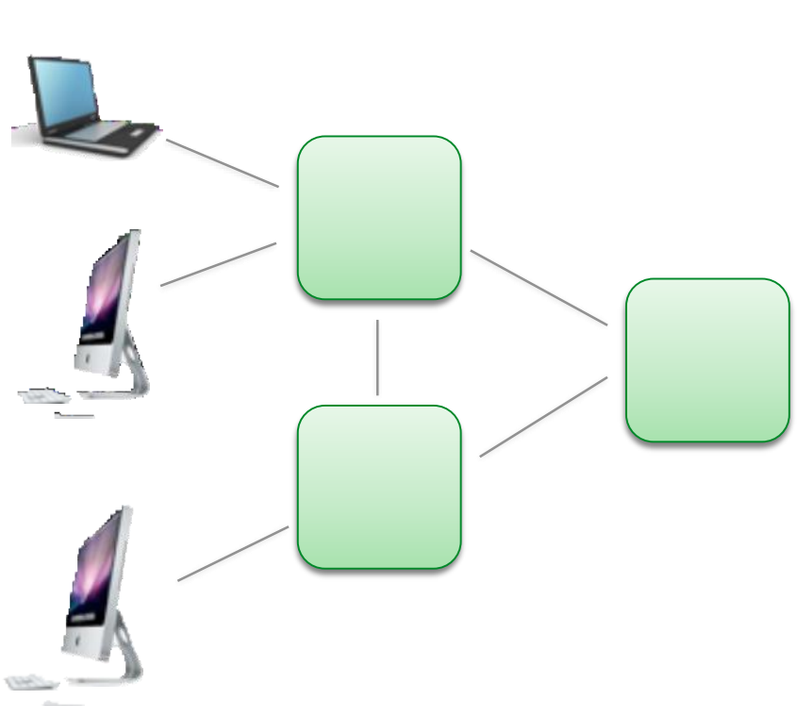
Conventional Networking

Connected by switches…



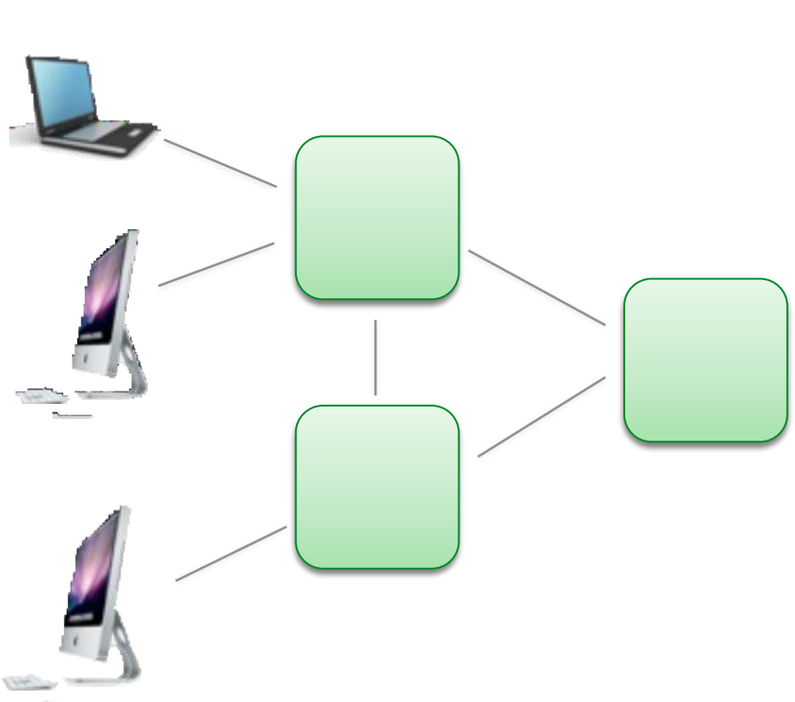
Conventional Networking

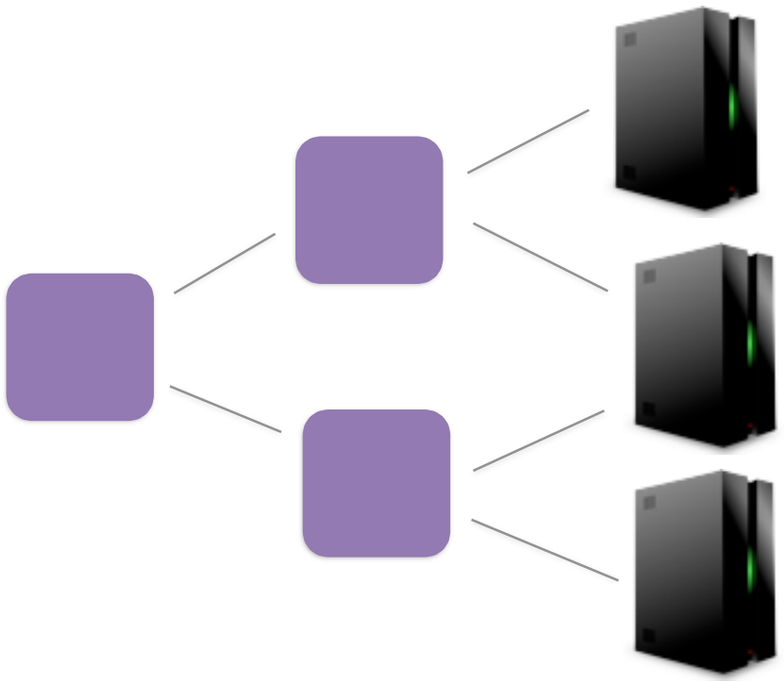
There are also servers...



Conventional Networking

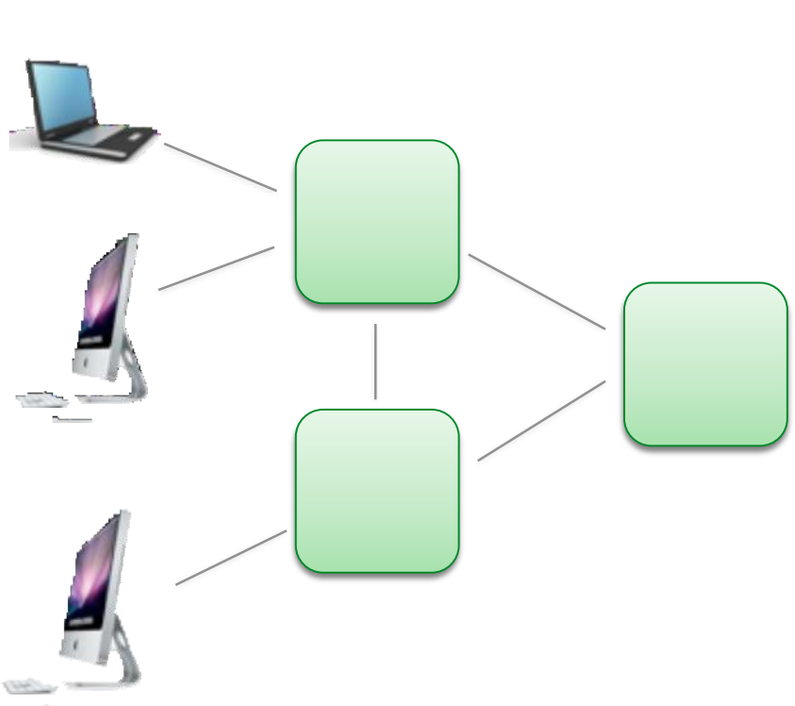
Connected by routers…

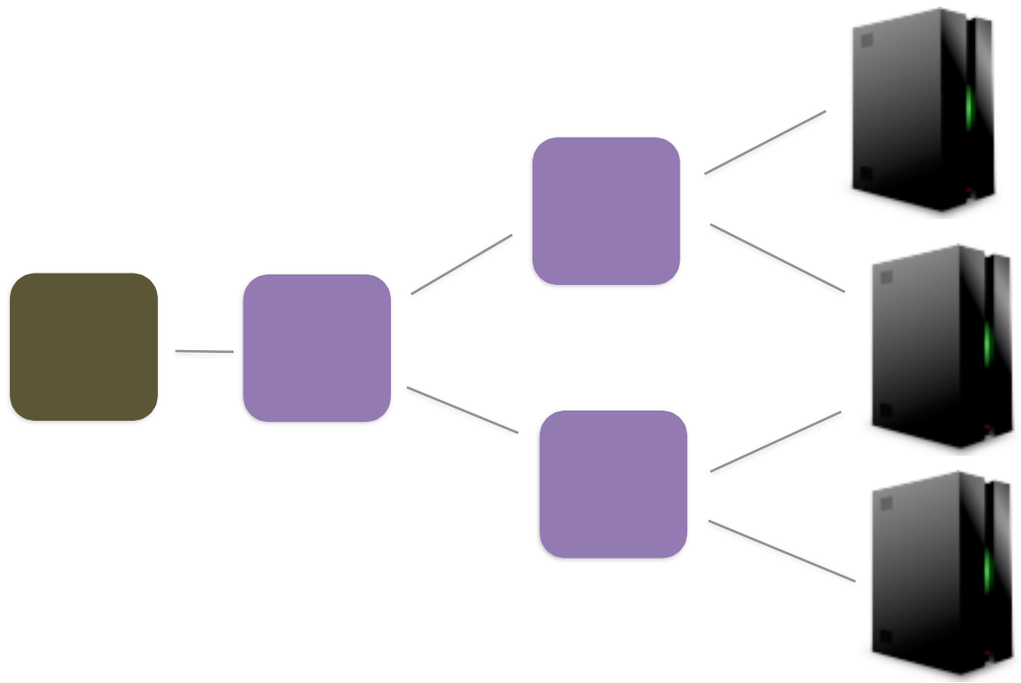




Conventional Networking

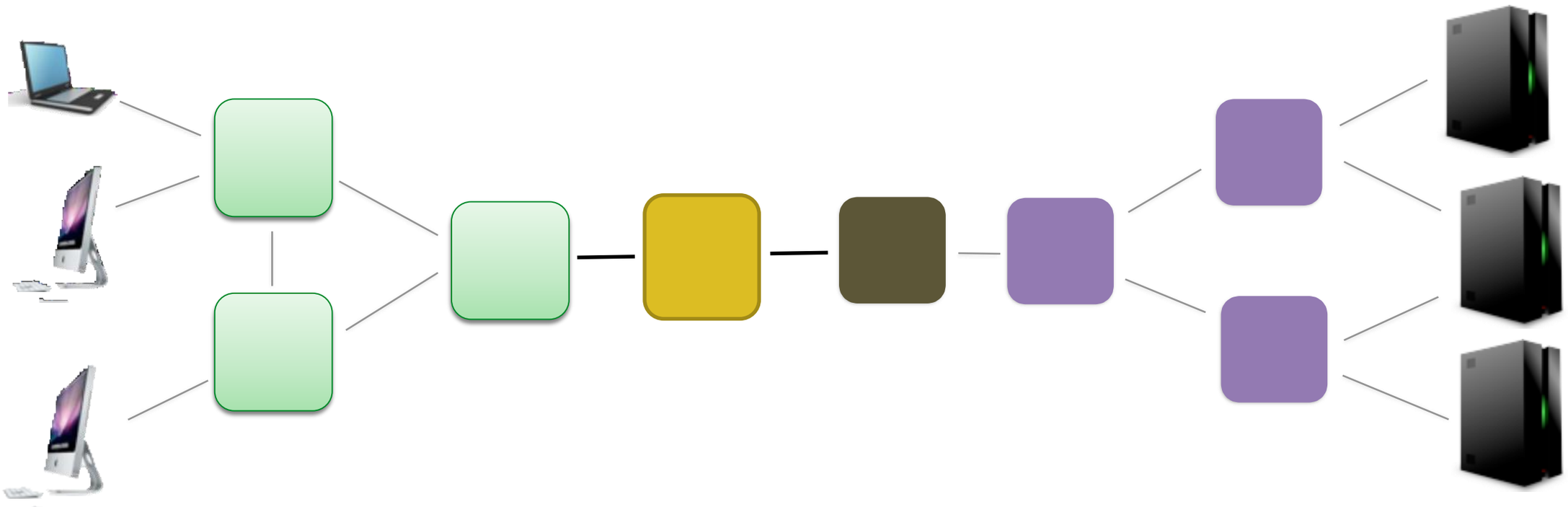
And a load balancer…





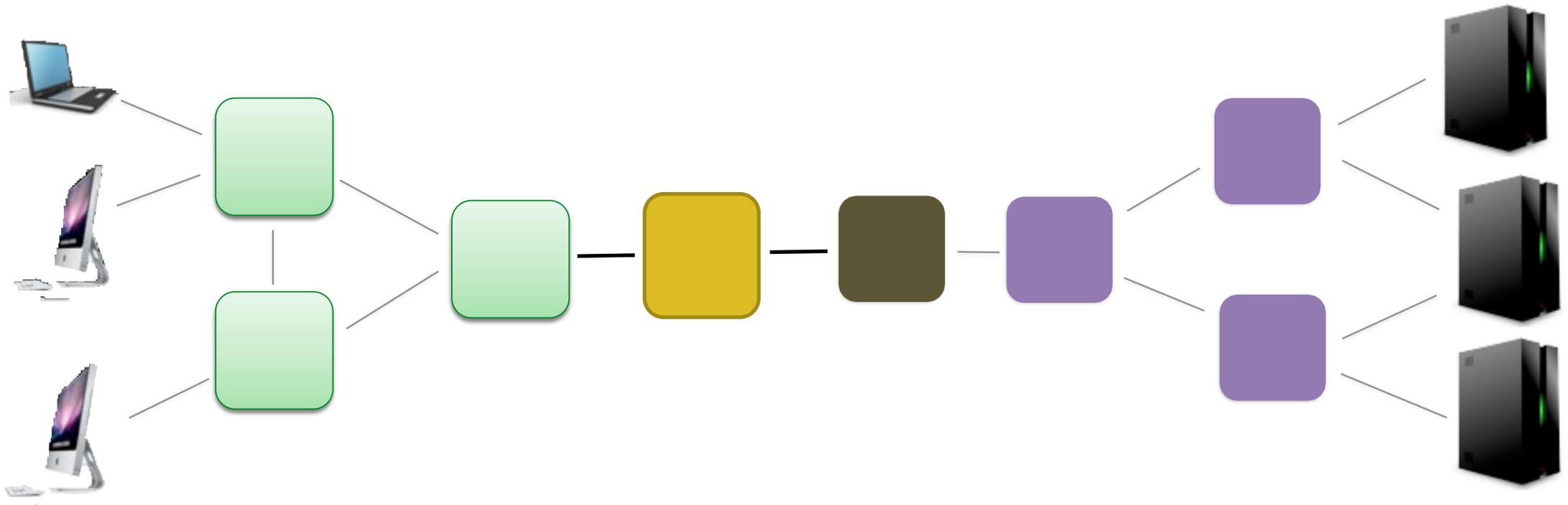
Conventional Networking

And a gateway router…



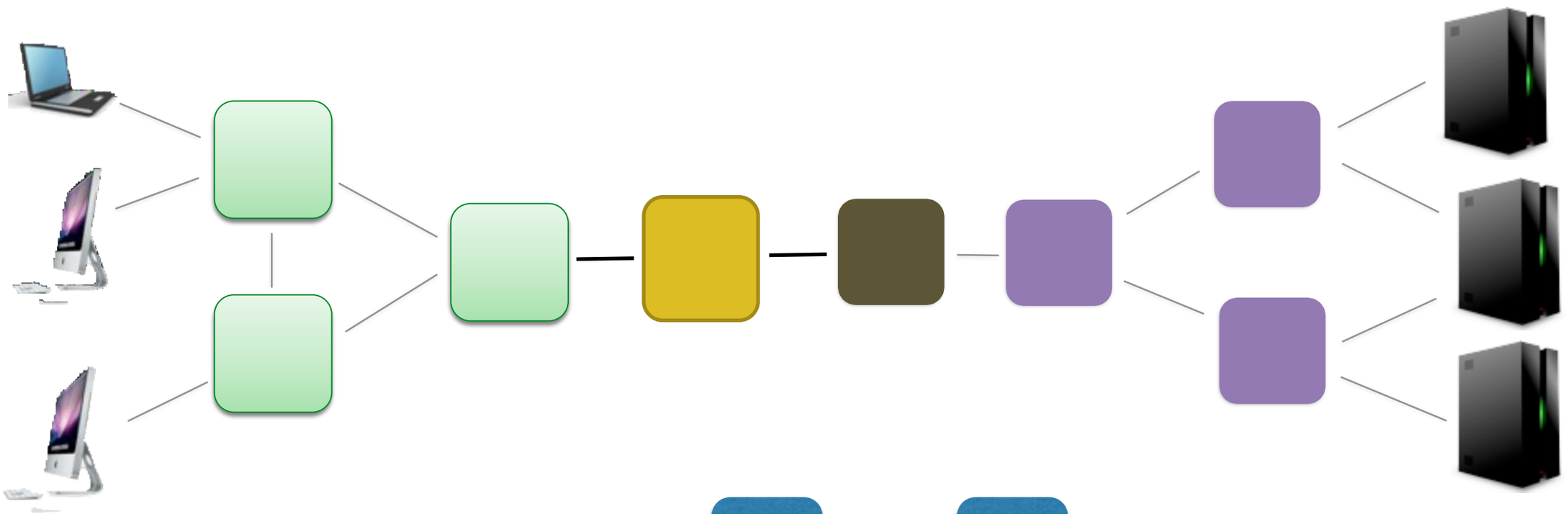
Conventional Networking

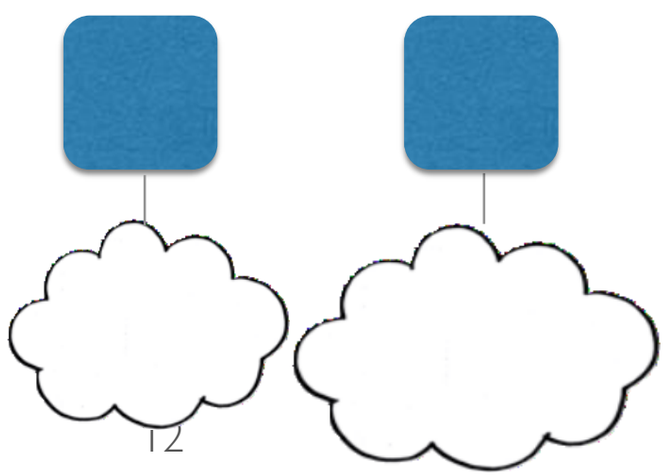
There are other ISPs…



Conventional Networking

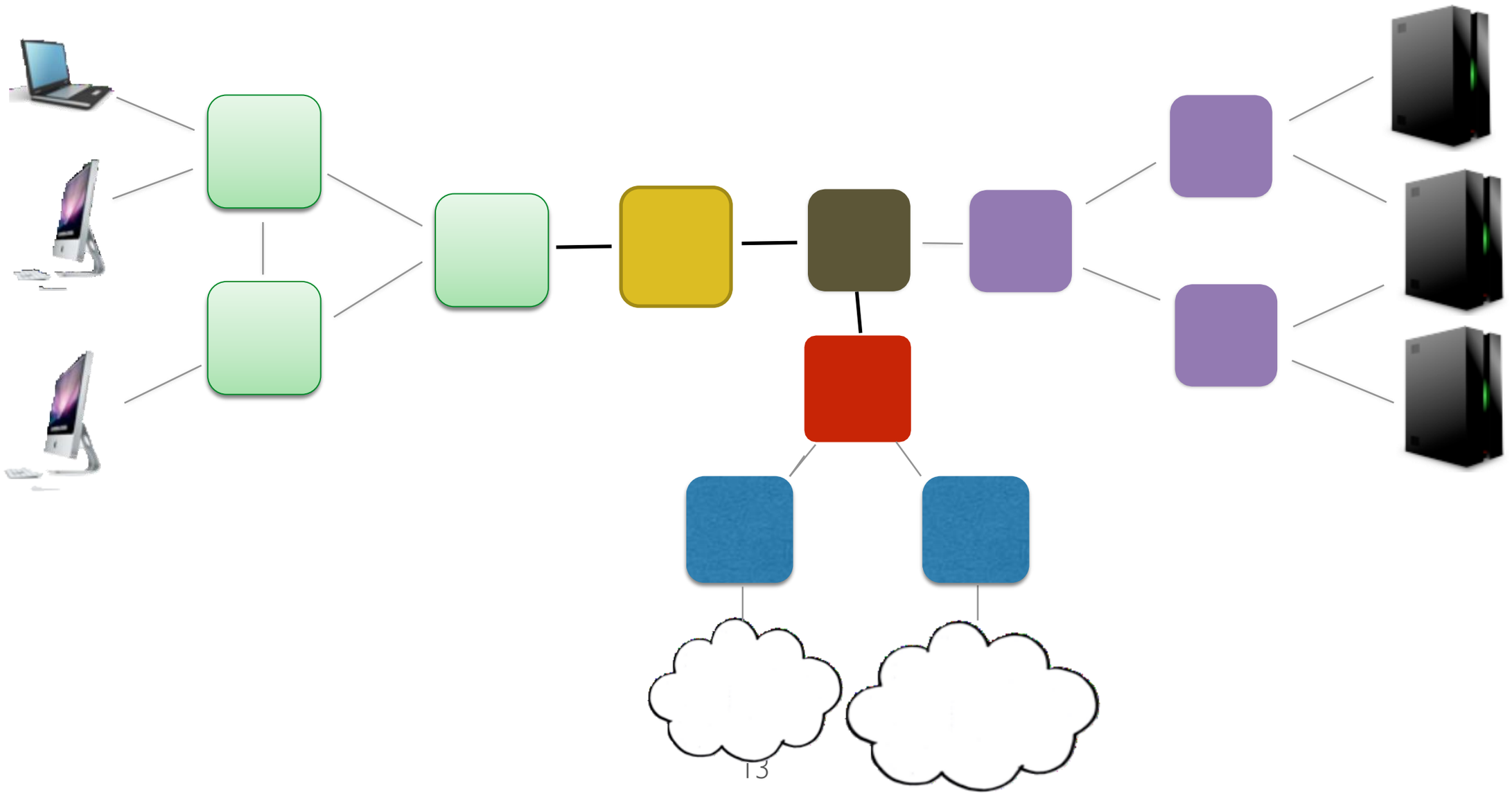
So we need to run
BGP…





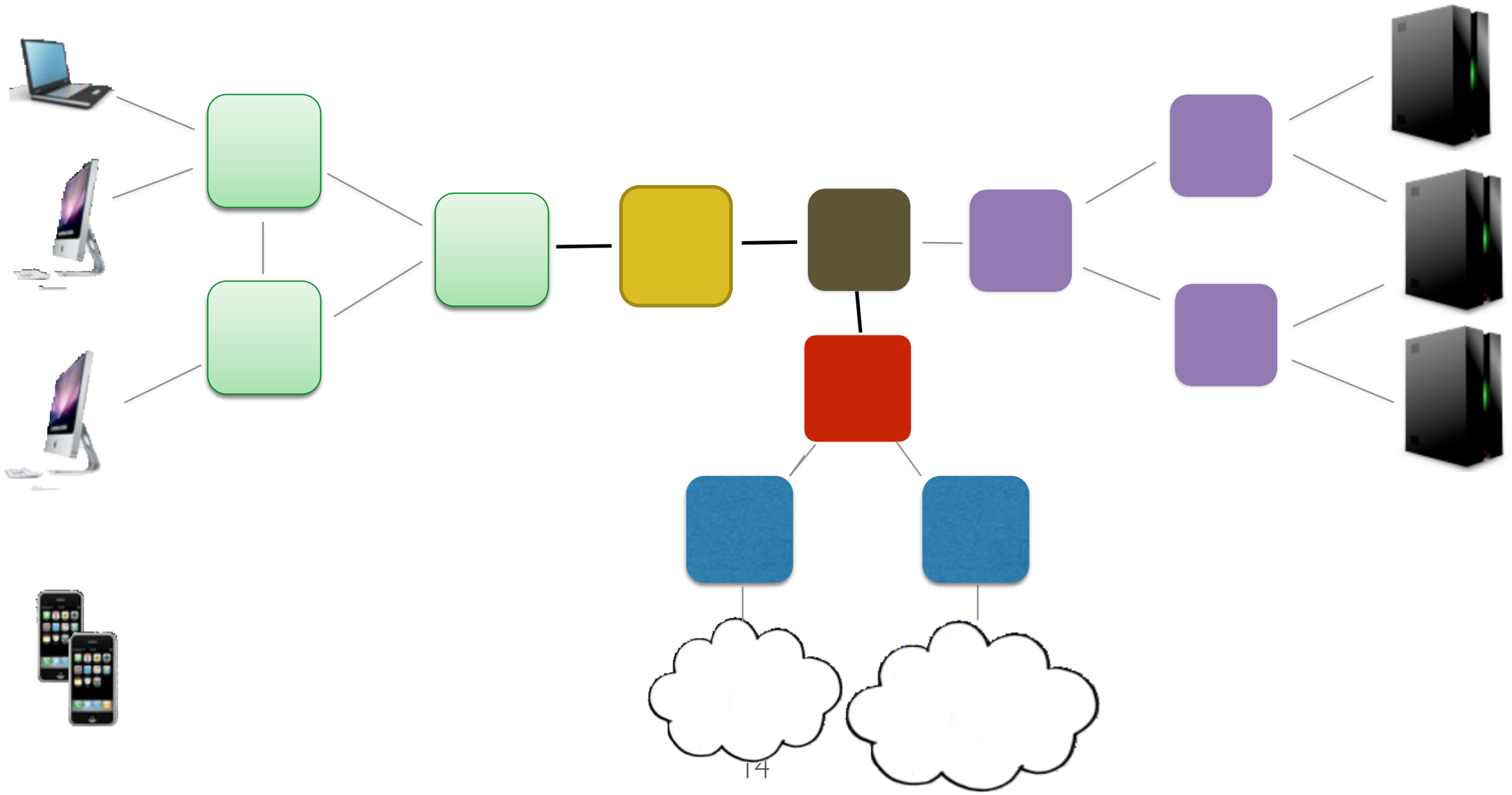
Conventional Networking

And we need a firewall to filter incoming traffic…



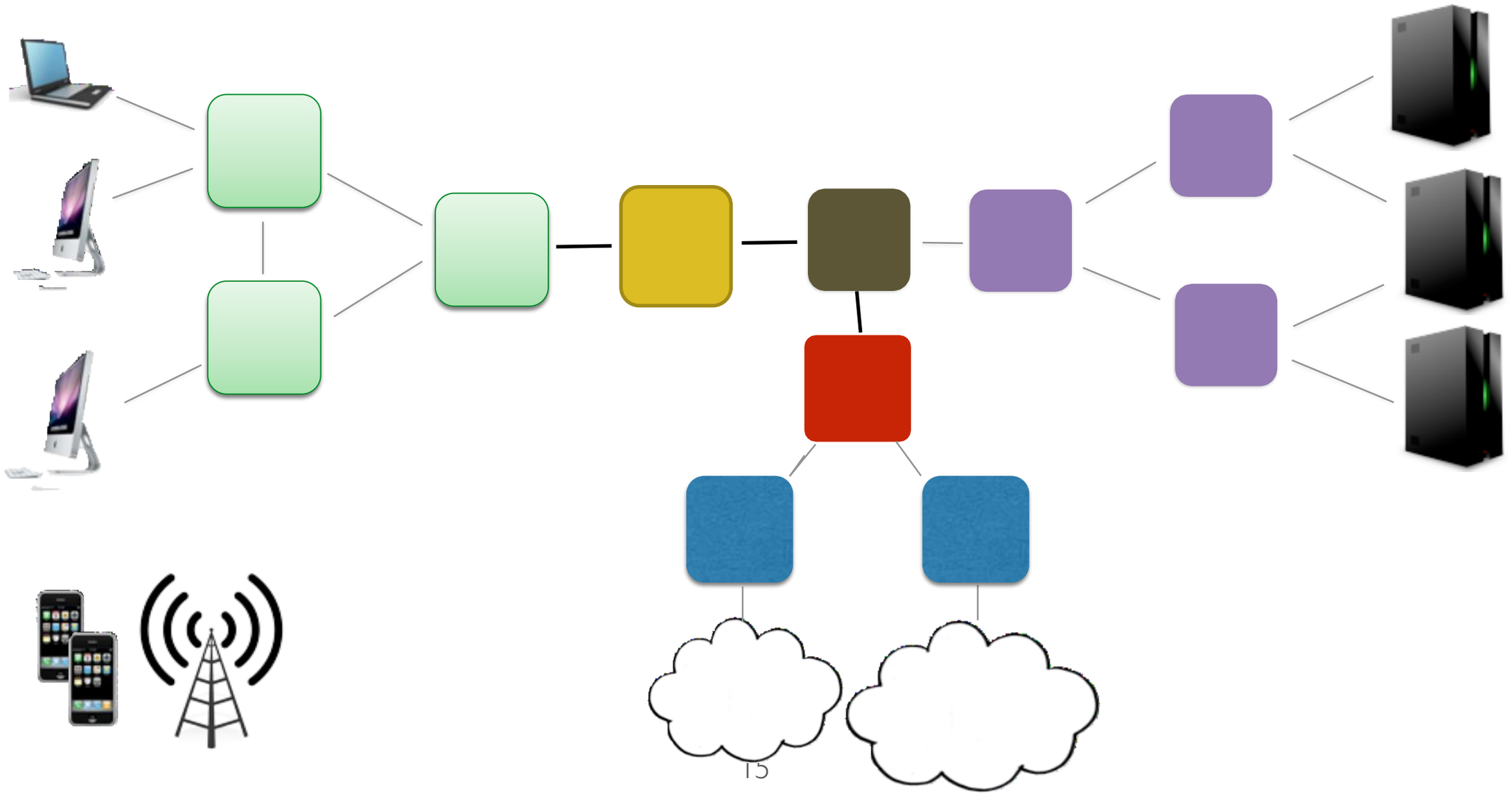
Conventional Networking

There are also wireless hosts…



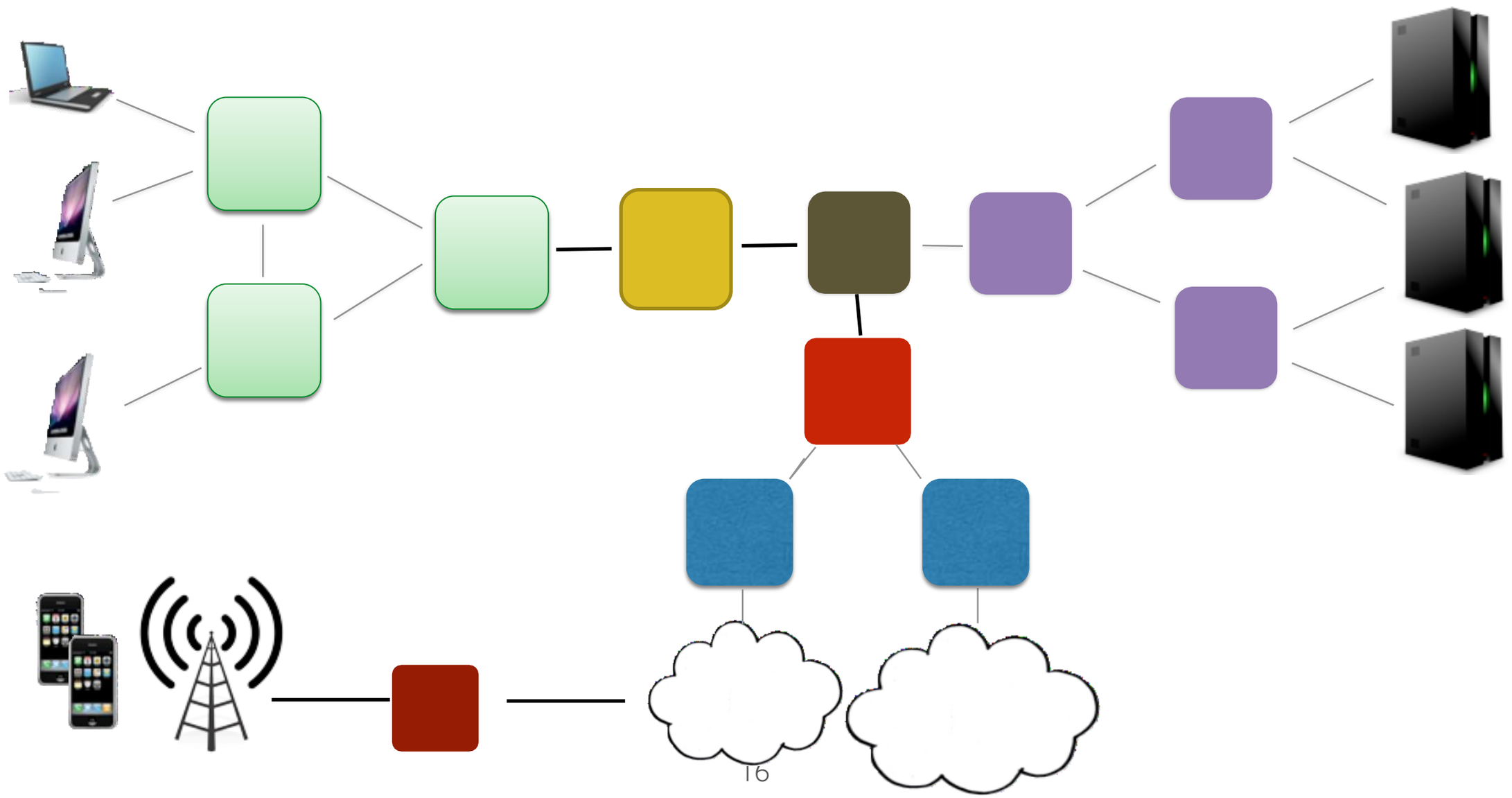
Conventional Networking

So we need wireless gateways...



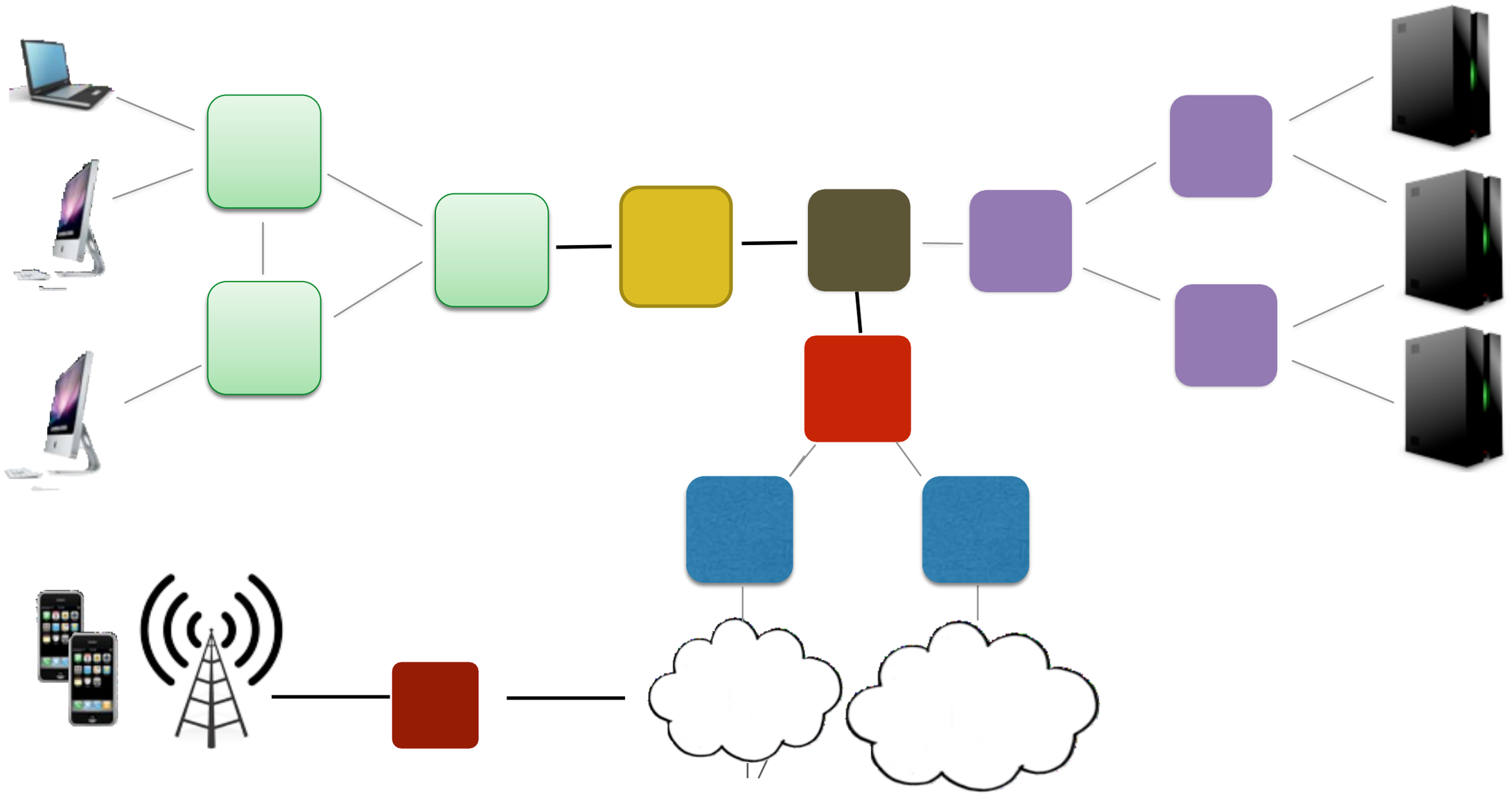
Conventional Networking

And yet more middleboxes for lawful intercept…



Conventional Networking

Each color represents a different set of control plane protocols and algorithms...

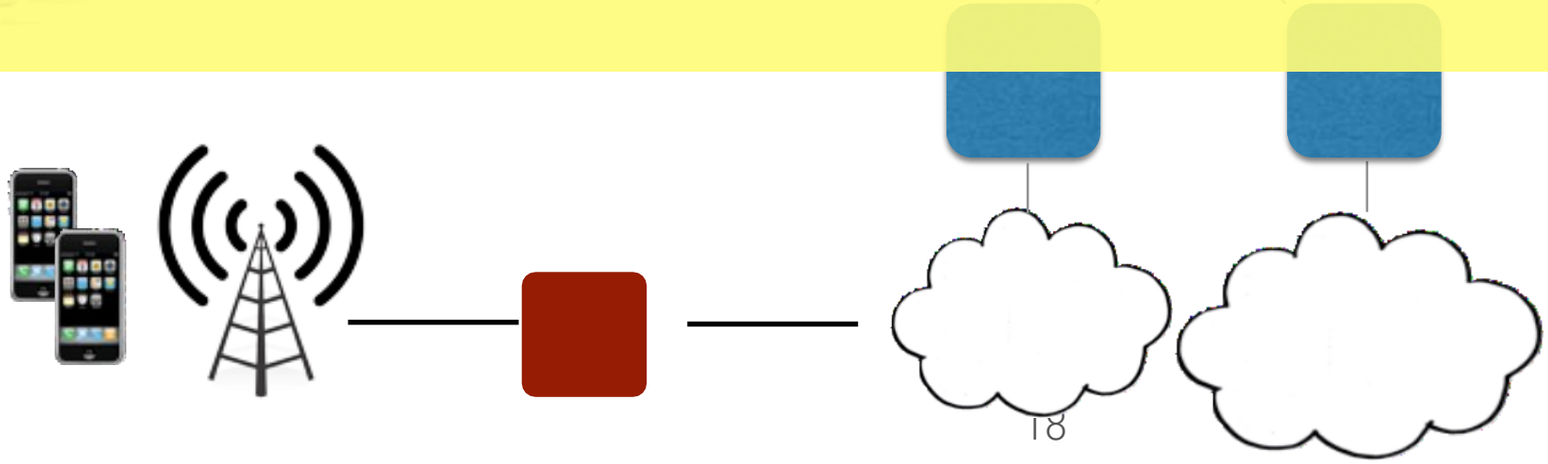


Conventional Networking

Reasoning about network behavior is *extremely* difficult

Does correctness matter? The Internet is best effort…

…the end-to-end principle says that hosts are best equipped to deal with failures!



Example: Outages

We **discovered a misconfiguration** on this pair of switches that caused what's called a "bridge loop" in the network

A network **change was [...] executed incorrectly** [...] more "stuck" volumes and added more requests to the re-mirroring storm

Even technically sophisticated companies are struggling to build networks that provide reliable service to users

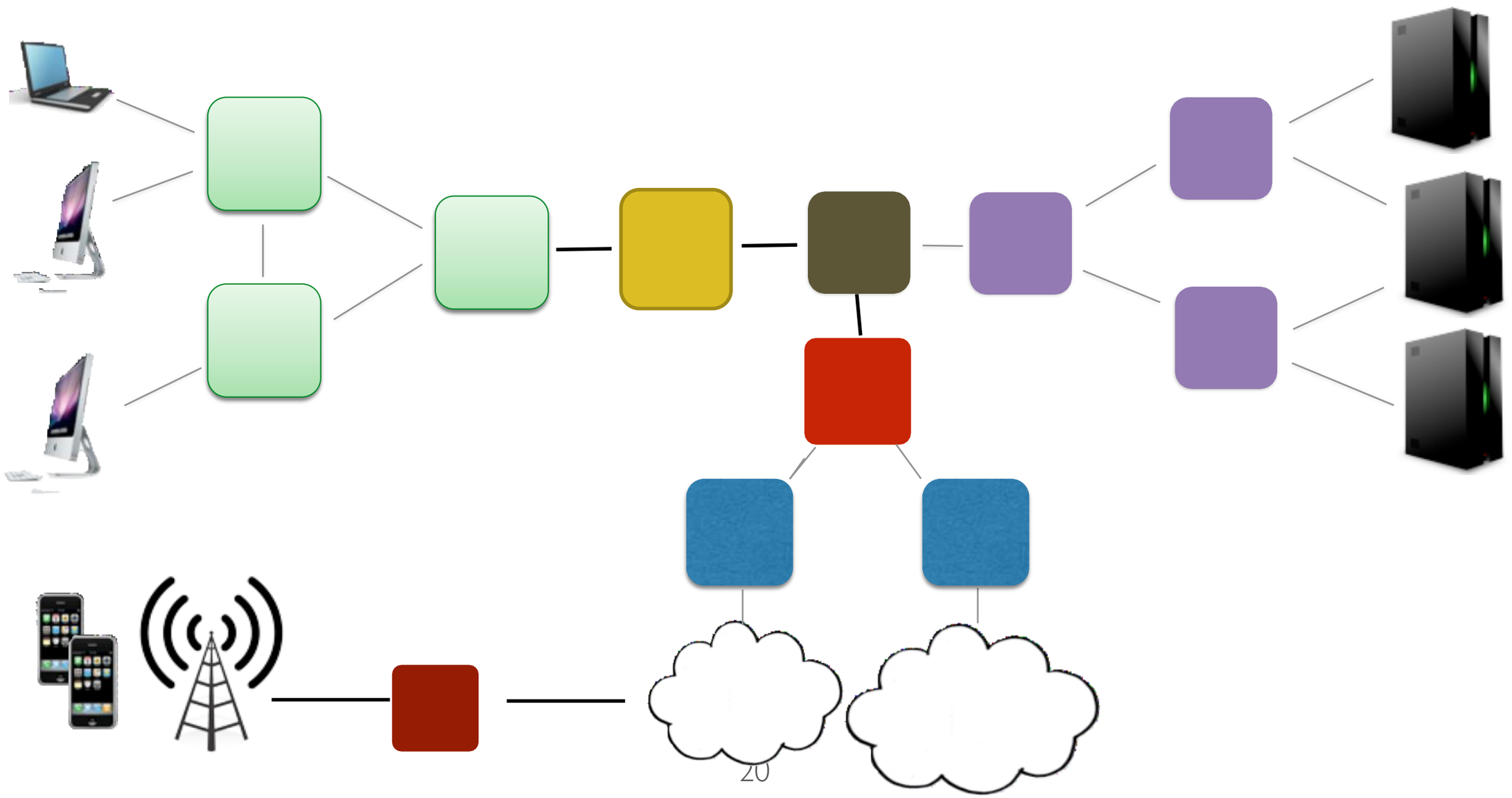
interrupted the airline's flight departures, airport processing and reservations systems

یکی از روترها [...] به دلیل باگی که در نرم افزار مودم وجود داشت به روزرسانی نشد و یک فرد از این آسیب پذیری برای نفوذ استفاده کرد

Software-Defined Networking

A clean-slate architecture based on two key ideas:

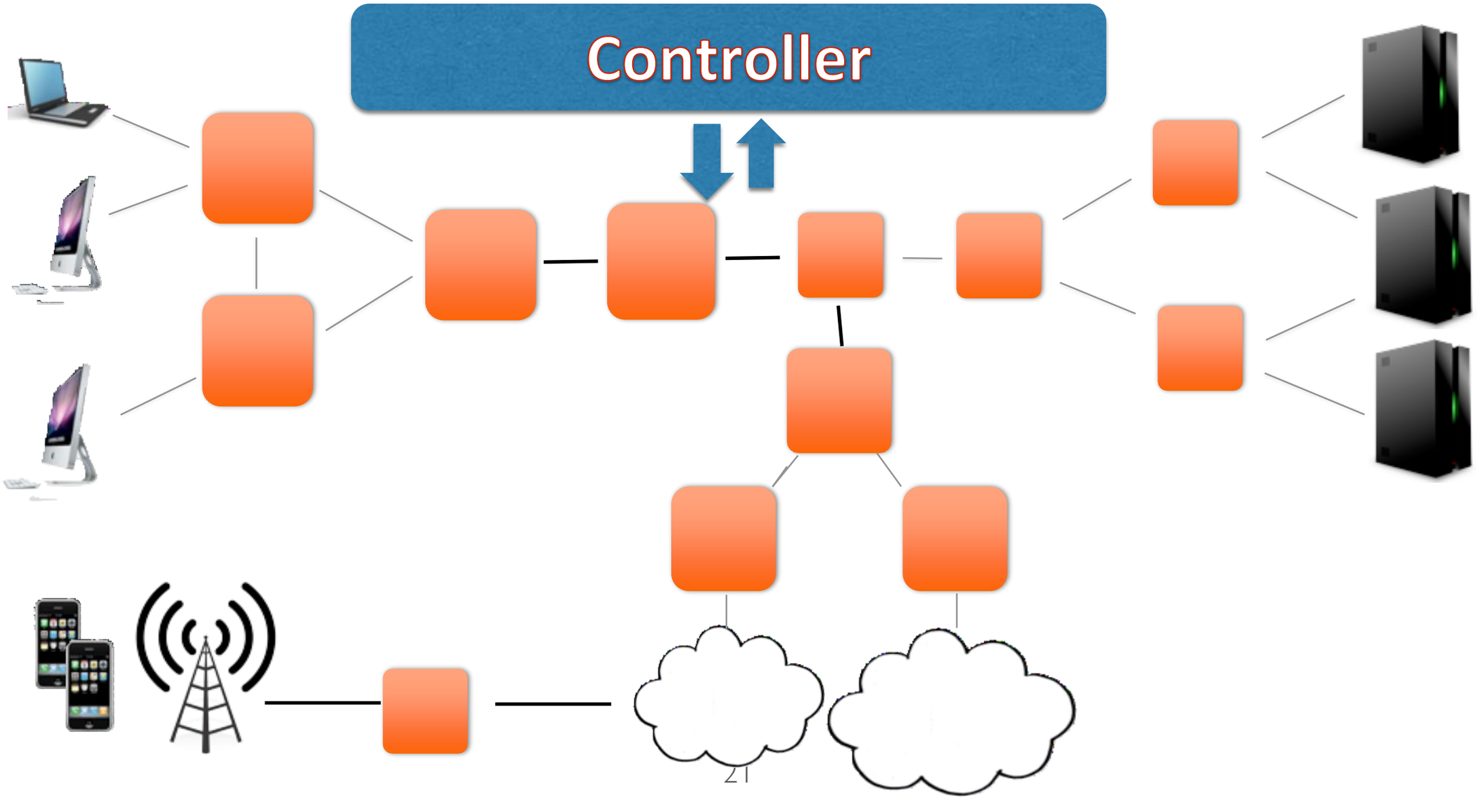
- Generalize network devices
- Separate control and forwarding



Software-Defined Networking

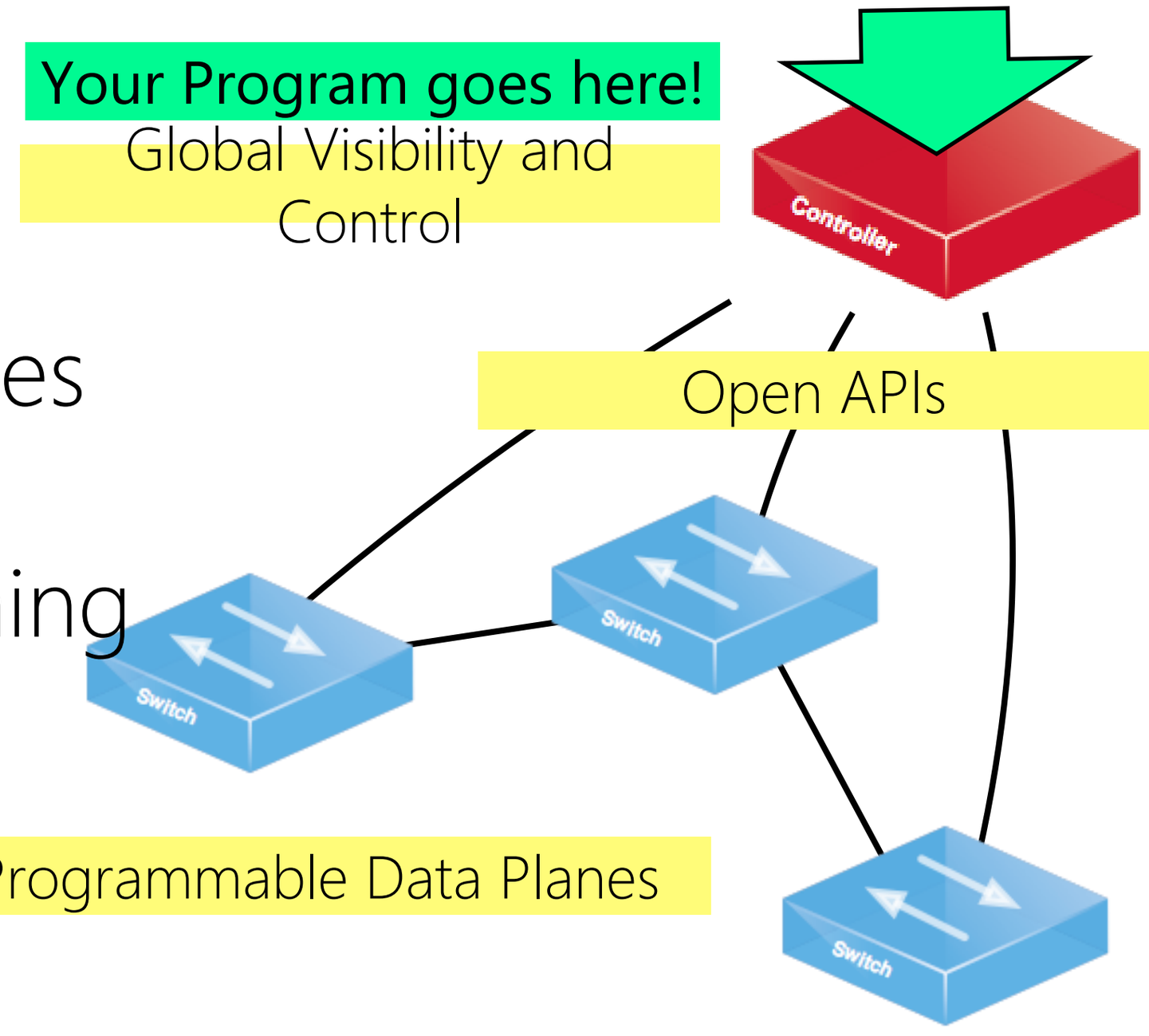
A clean-slate architecture based on two key ideas:

- Generalize network devices
- Separate control and forwarding



Software-Defined Networking

Enabling use of reasoning techniques typically associated with the programming languages and verification communities



But how do we write all of this software?

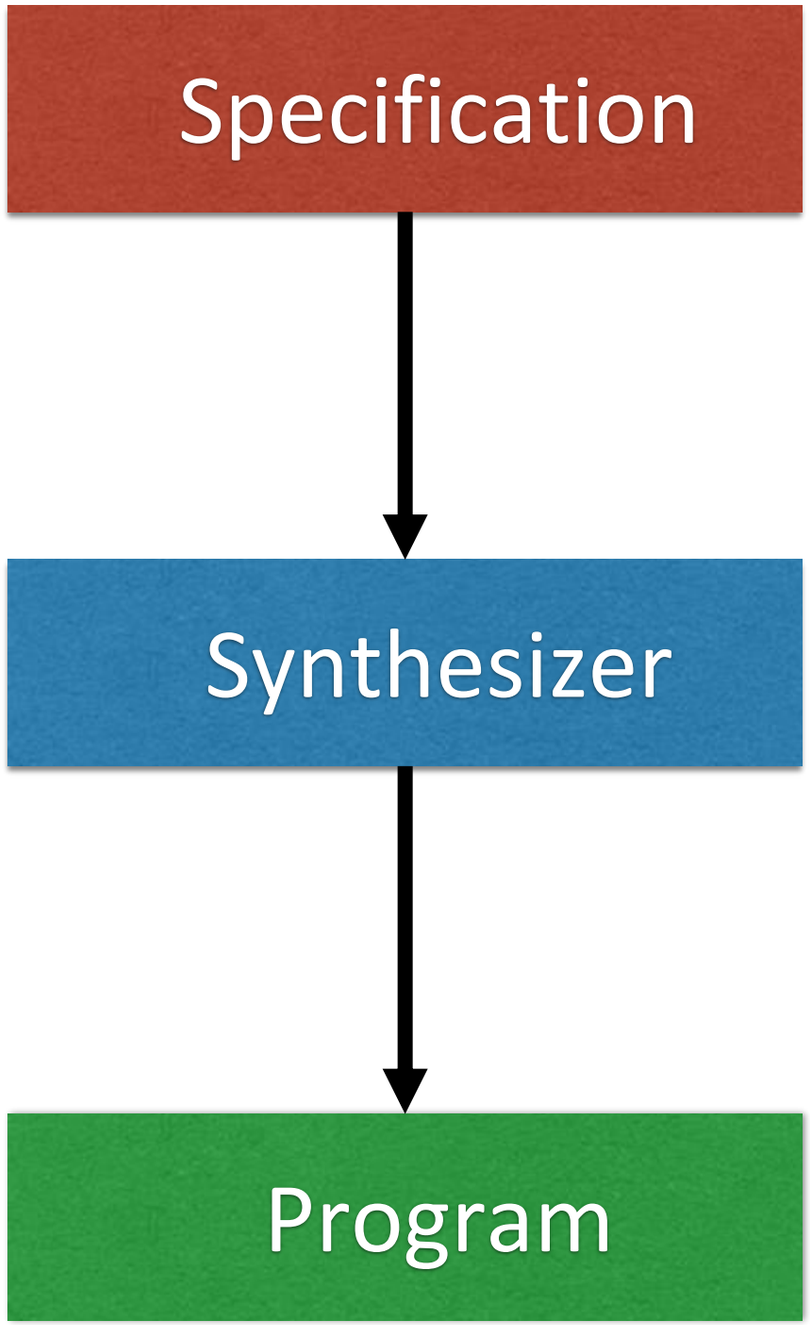
Software Synthesis

What if programmers could…

- **Sketch** the structure of their program…
- Give **examples** and **scenarios**…
- Specify functional **behavior**…
- Write down high-level **requirements**…
- Express resource **constraints**…

…and a tool **automatically synthesized** a correct and efficient implementation?

Software Synthesis



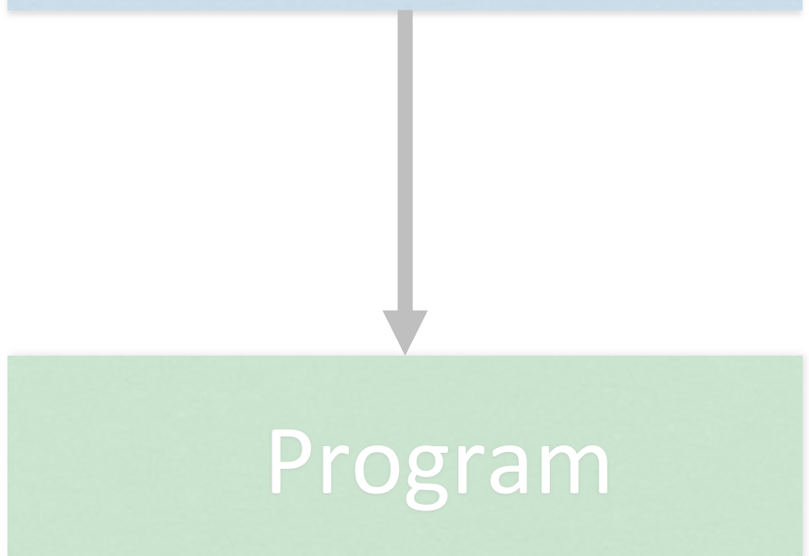
Software Synthesis

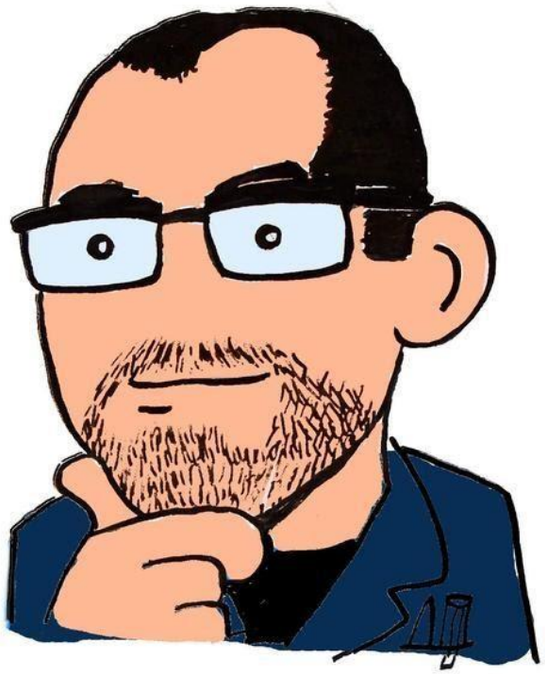
Partial Program

Logical Formula

Input-Output Examples

Programmers can express their insights in a wide variety of ways, not just in standard code!





- Does software synthesis really work?
- Answer: yes - for certain domains

Synthesis for Networks

- Programs are large, but simple and highly structured—e.g., loop free!
- The desired behavior of the network is often clear (at least at an intuitive level)
- Most difficult aspects of network programming stem from limited resources and inherent concurrency

This Tutorial

Synthesis is an effective means for automating some of the trickiest aspects of network programming

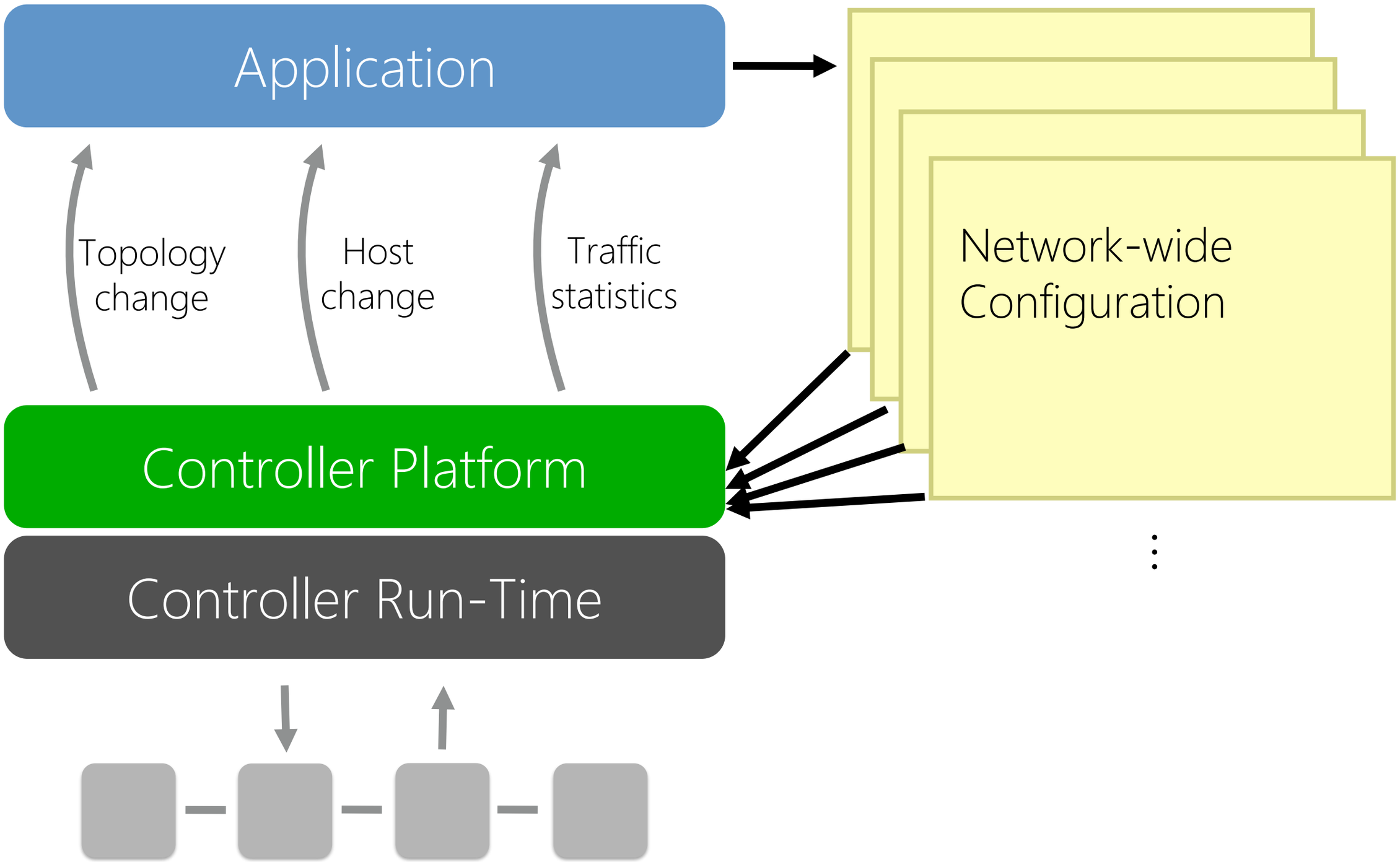
Outline:

- Network Update Synthesis
- Synchronization for Network Programs
- Optimizing Horn Solvers for Network Repair

Efficient Synthesis of Network Updates

[SIGCOMM '12, PLDI '15]

Dynamic SDN Applications

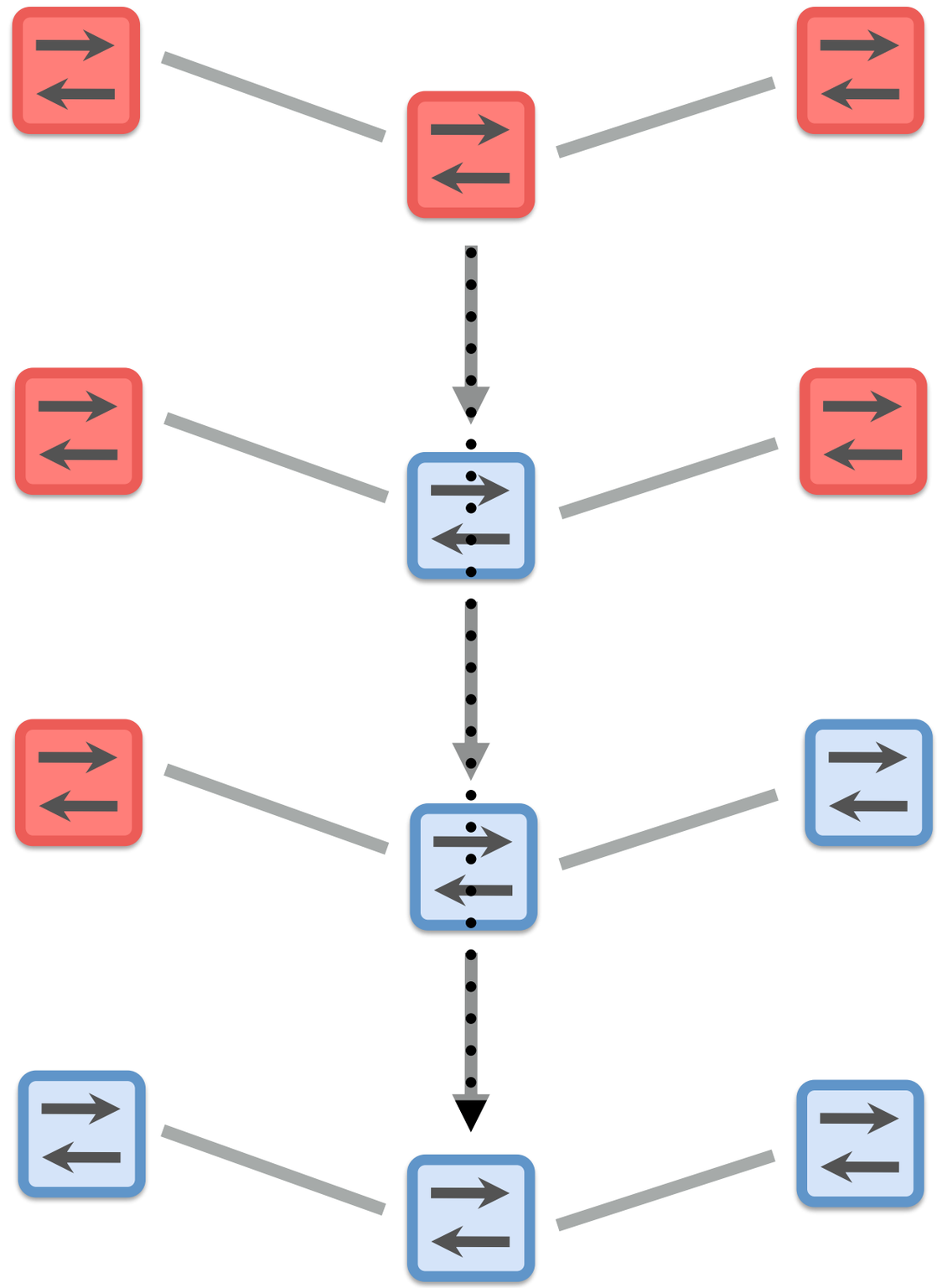


Network Updates

How can we transition between global states?

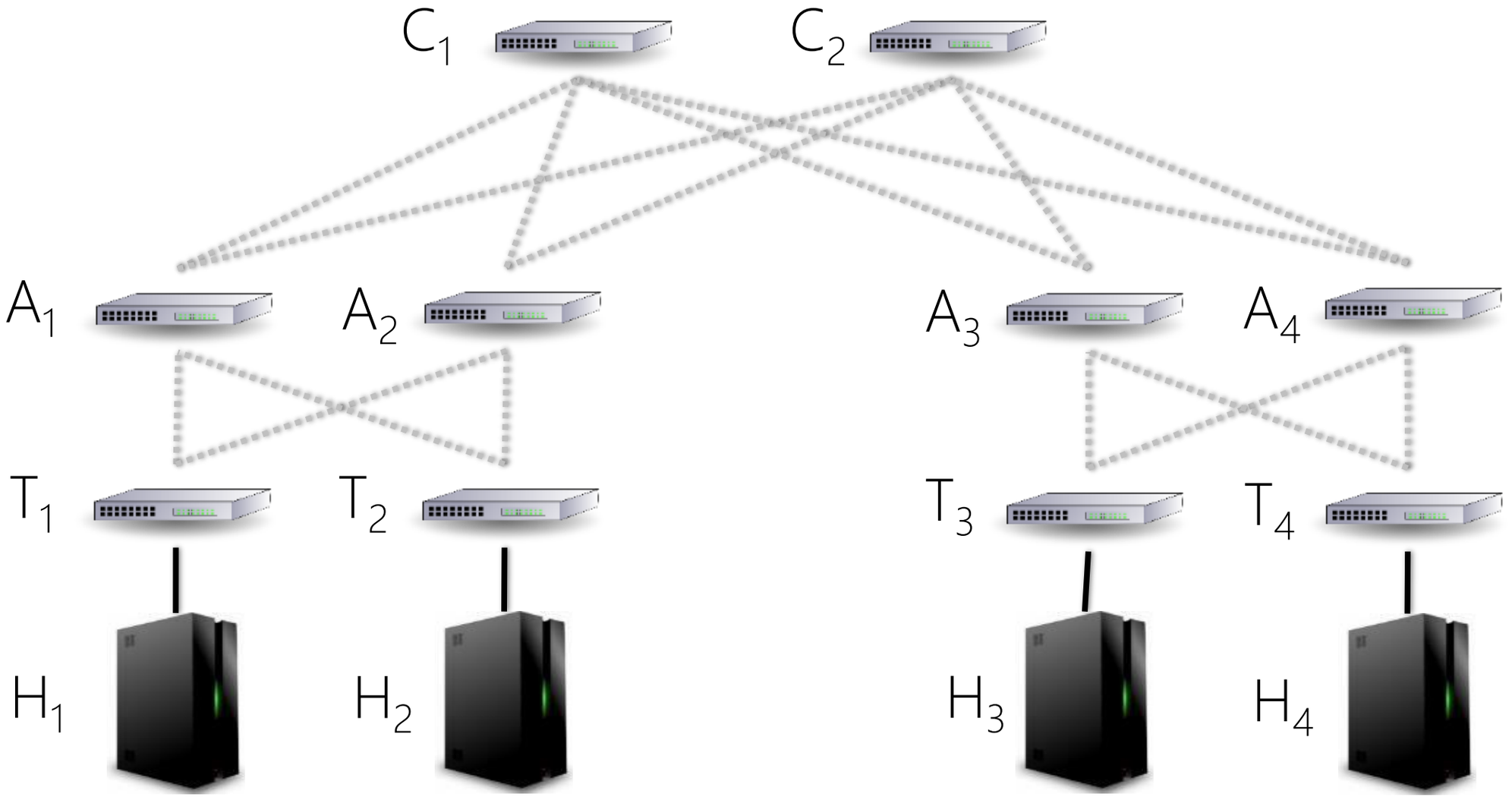
Initial State

Target State

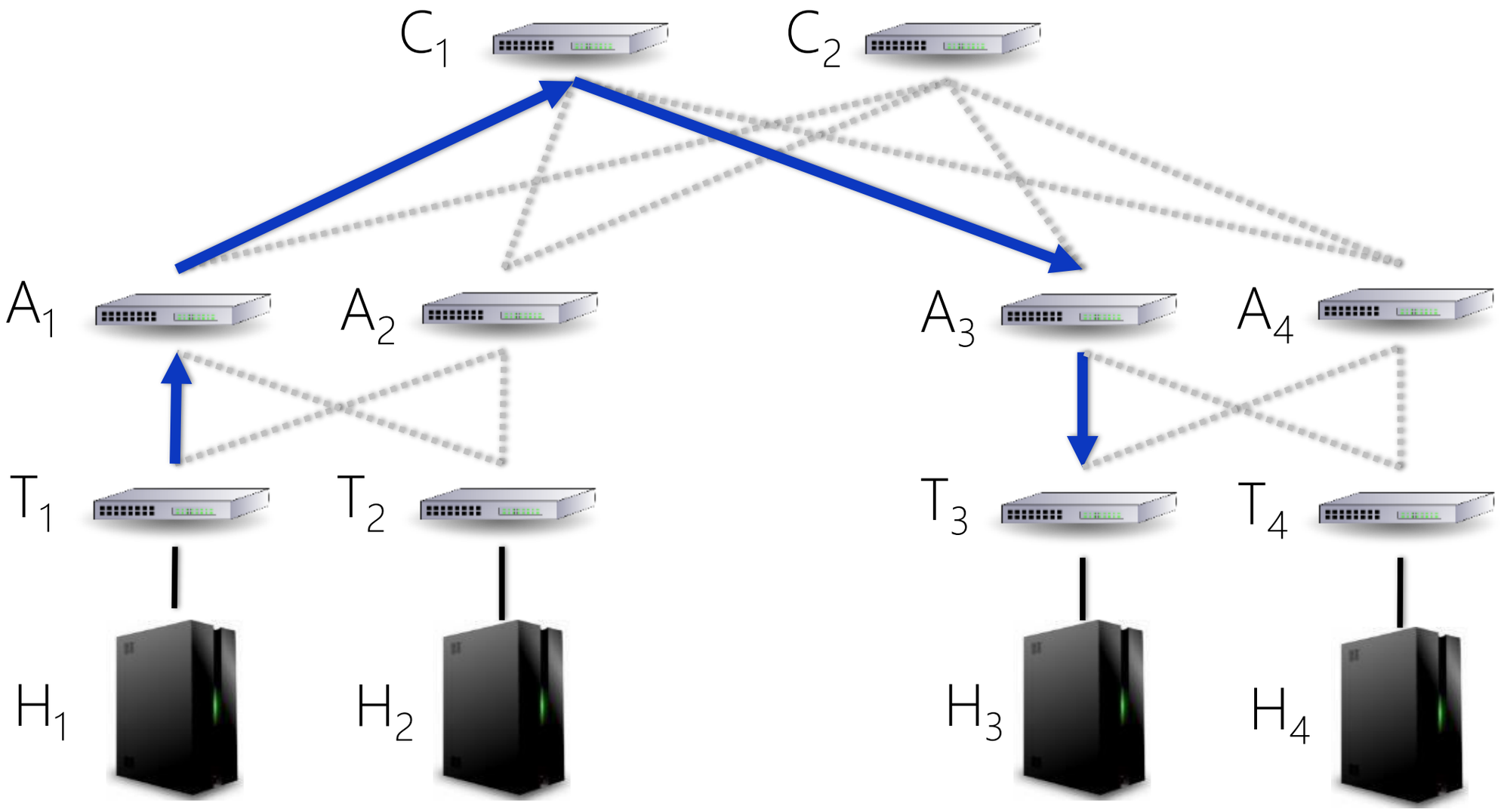


Problem: naive updates can break important invariants!

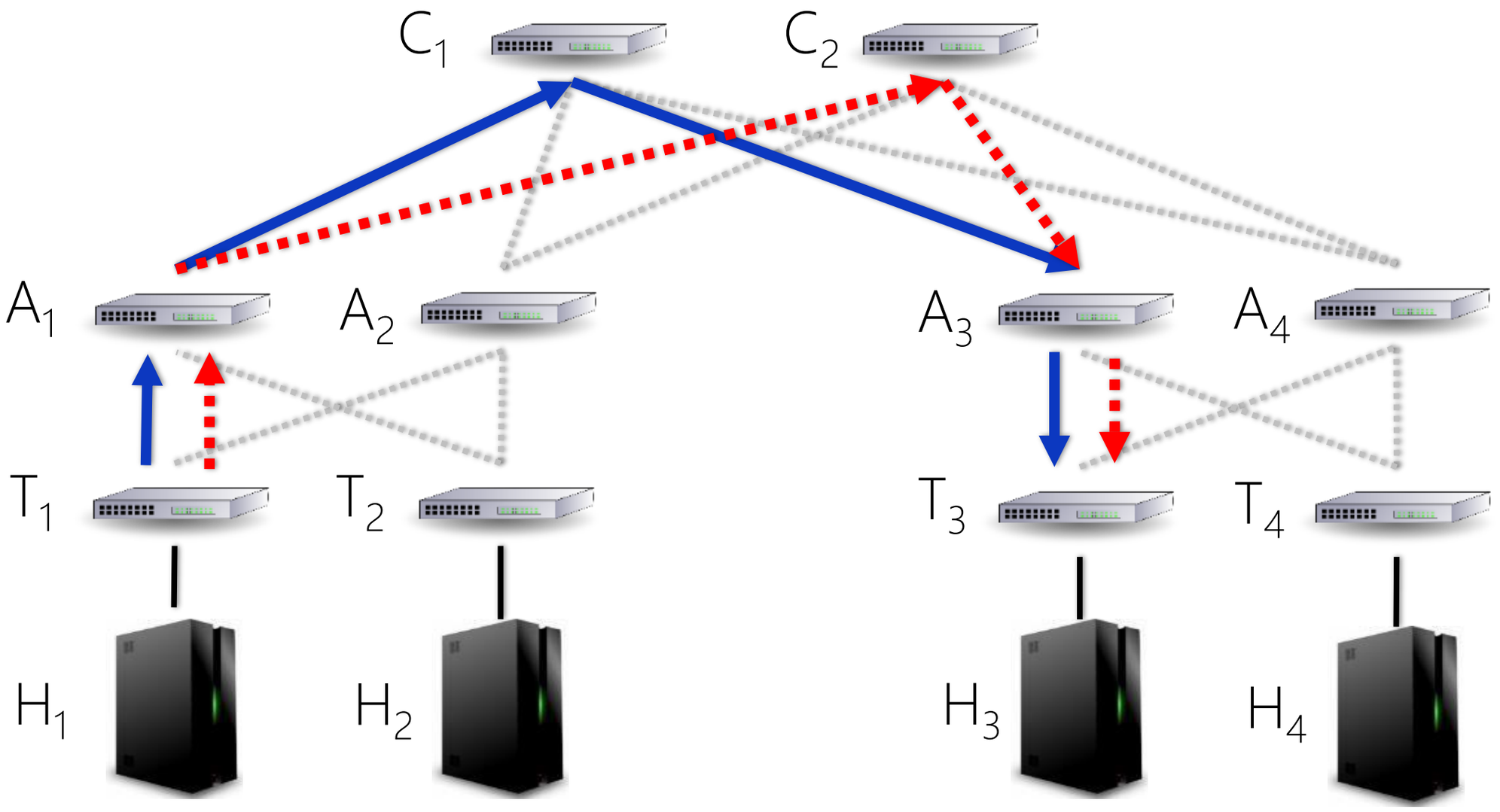
Example: Data Center



Network Configuration

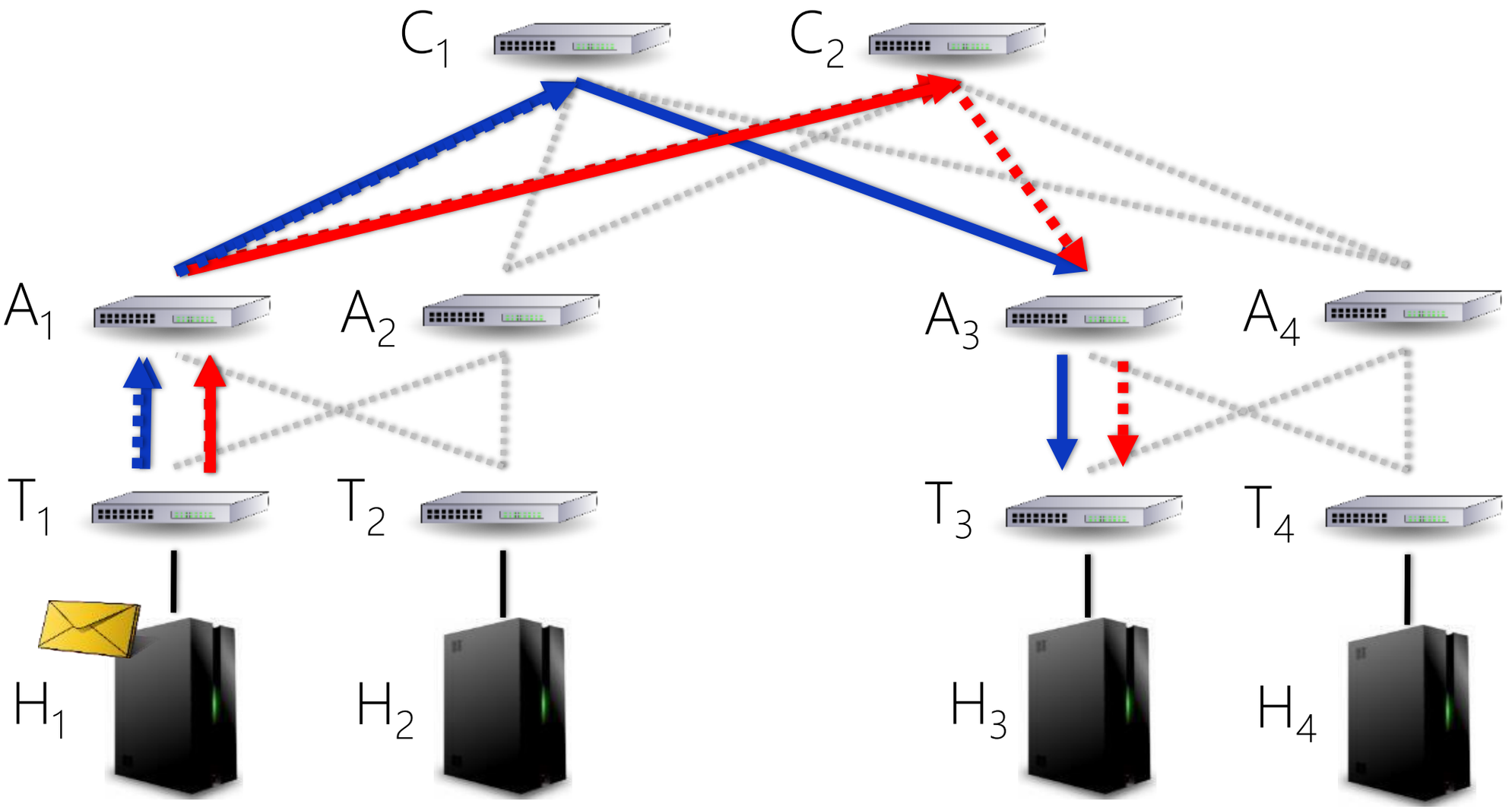


Network Update



- Update program:
  upd T1; upd C2; upd A3; upd A1

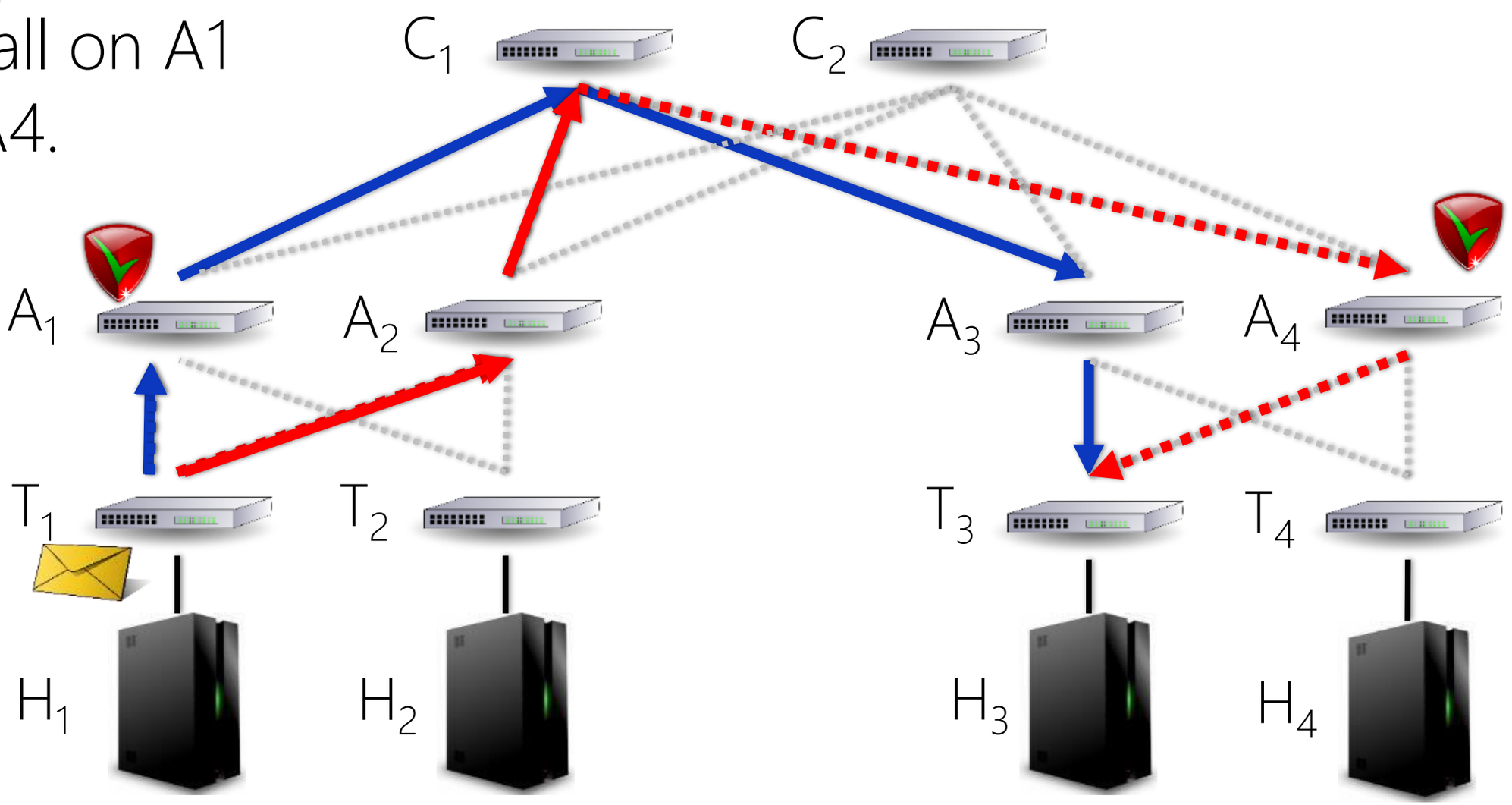
Naïve Update



Possible problem: black holes

Naïve Update

Example:
Firewall on A1 and A4.



Possible problem: access control violation

Is This Really a Problem?

At 12:47 AM PDT on April 21st, a network change was performed as part of our normal scaling activities…

During the change, one of the steps is to shift traffic off of one of the redundant routers…

The traffic shift was executed incorrectly and the traffic was routed onto the lower capacity redundant network.

This led to a "re-mirroring storm"…

During this re-mirroring storm, the volume of connection attempts was extremely high and nodes began to fail, resulting in more volumes left needing to re-mirror. This added more requests to the re-mirroring storm…

The trigger for this event was a **network configuration change**.

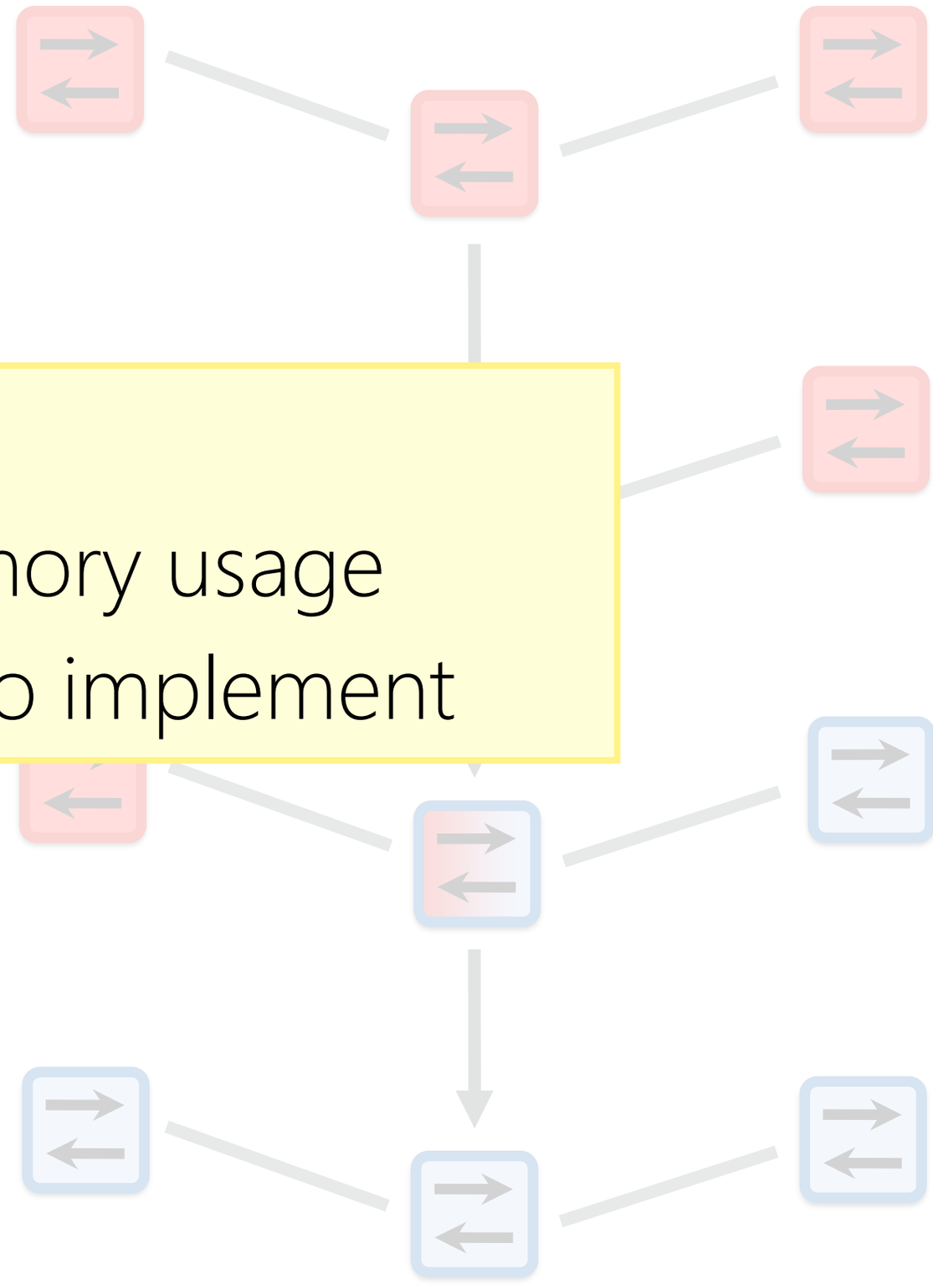
Outages Cost a Lot

- Aug 13, 2013, Amazon was down for roughly 40 minutes
- It lost **$1,104** in net sales per second, on average

https://www.buzzfeednews.com/article/mattlynley/the-high-cost-of-an-amazon-outage

Per-Packet Consistency

Consistency Guarantee: every packet (or flow) in the network "sees" a single policy version



Two-Phase Update:

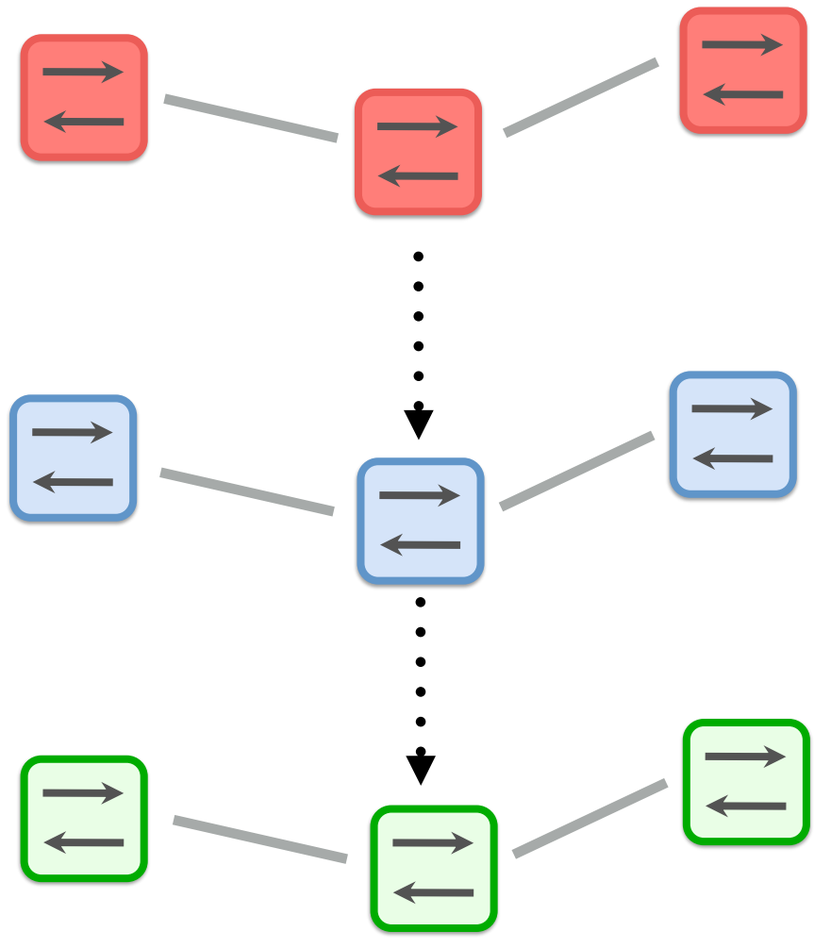
- Tag confi
  versions
- Stamp inc
- Install new configuration in core
- Install new configuration at edge
- Wait for in-flight packets to exit
- Delete old configurations

Limitations:

- Doubles peak memory usage
- Updates are slow to implement

Per-Packet Consistent Updates

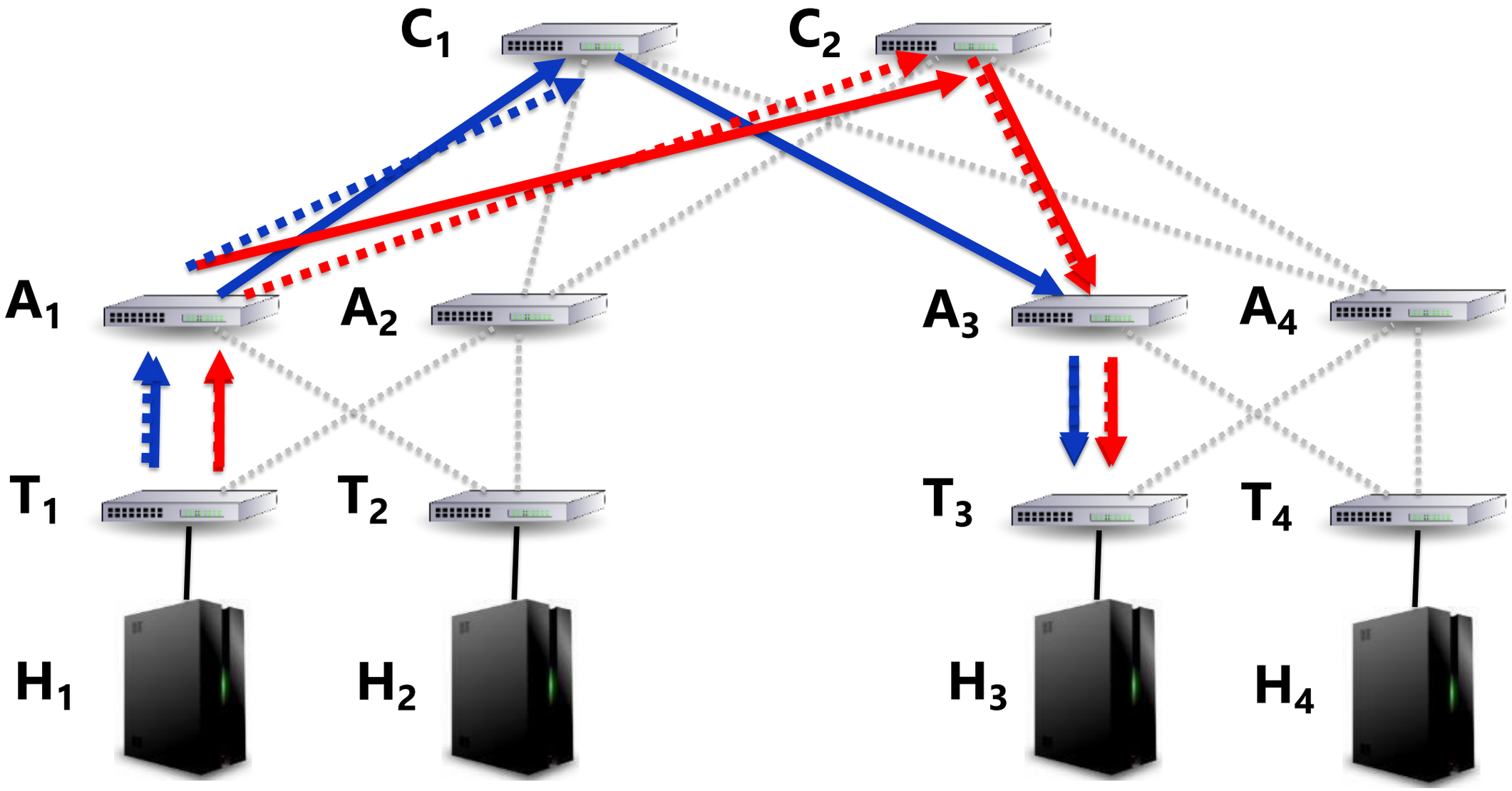
Theorem (Universal Property Preservation): a network update is per-packet consistent if and only if it preserves all safety properties.



Questions:

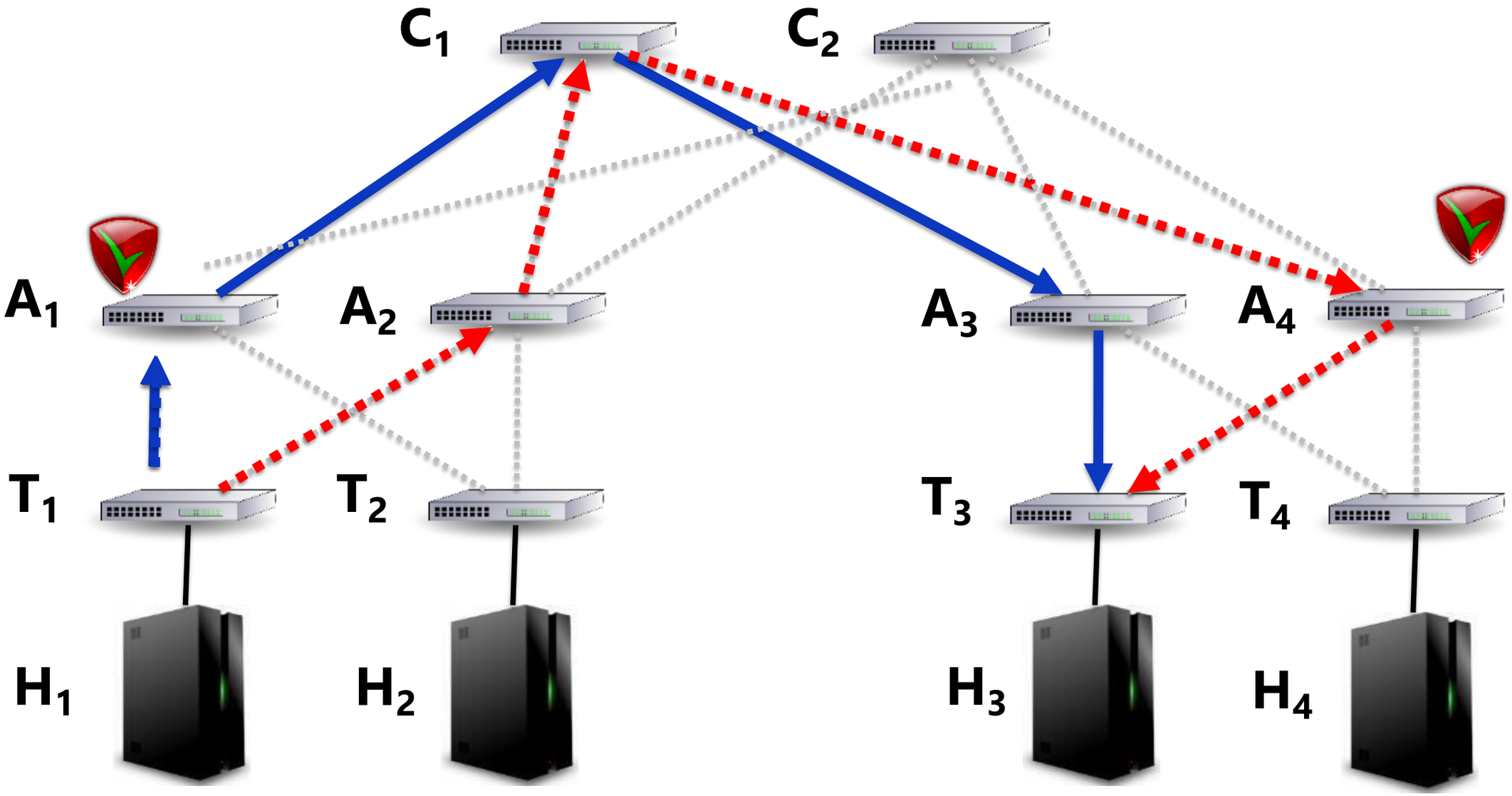
- Can we implement a per-packet consistent update by simply updating switches in the right order?
- If not, can we relax the requirements in a reasonable way to obtain an efficient

Example: Data Center



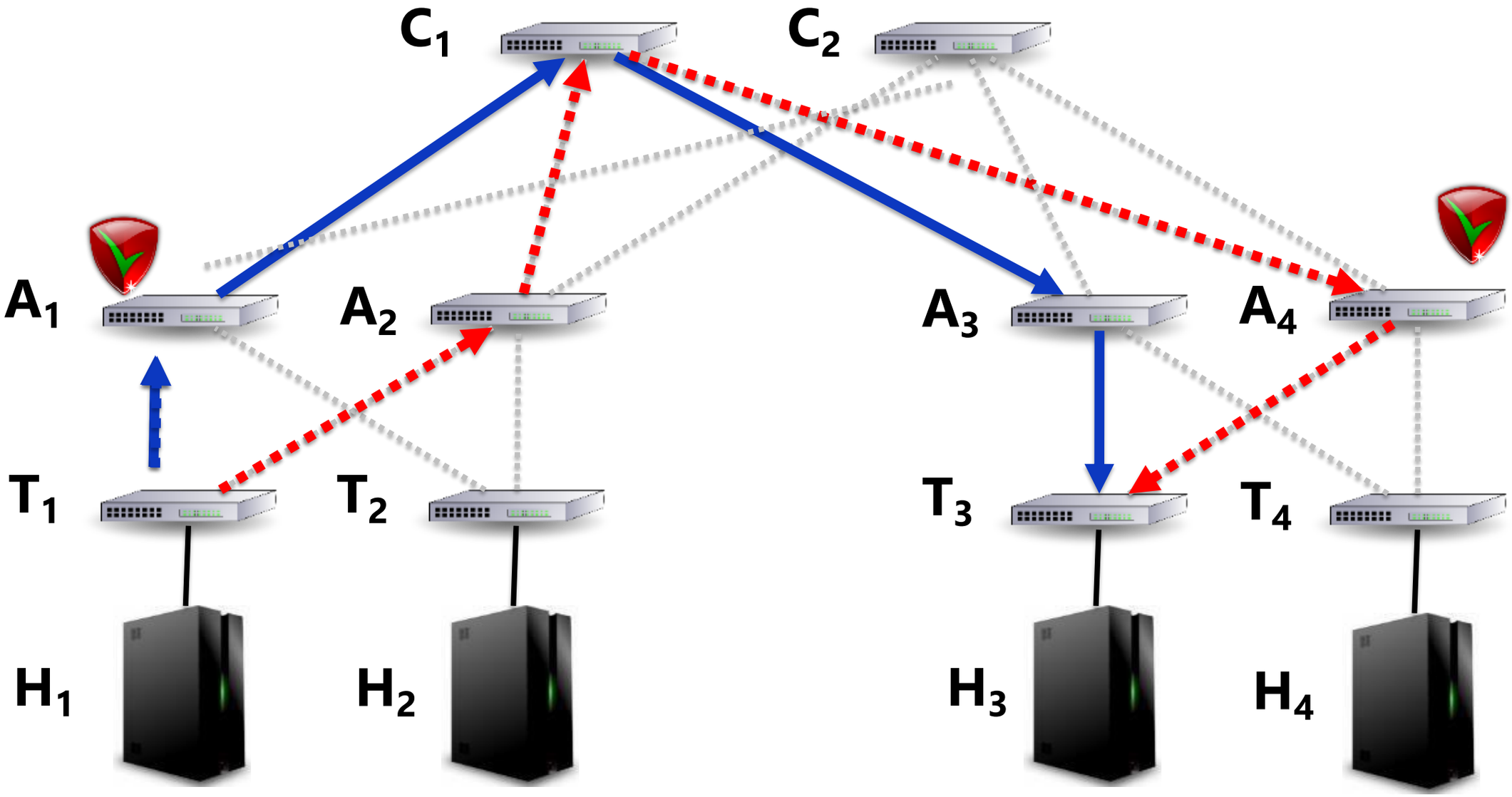
Update: `upd T1; upd C2; upd A3; upd A1`

Naive Update



- **Update:** `upd A2; upd A4; upd T1; upd C1` ✗
- **Update:** `upd A2; upd A4; upd C1; upd T1` ✗
- There is **no update** that ensures per-packet consistency

Relaxing Per-Packet Consistency

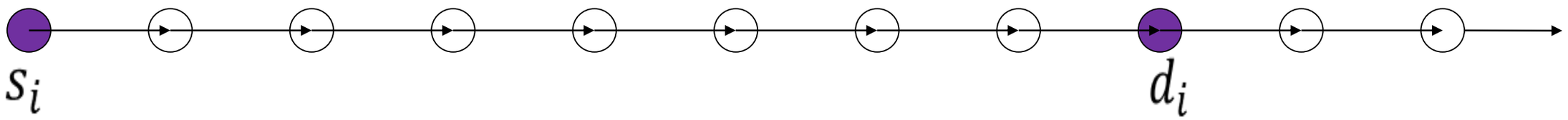


Idea: all packets eventually delivered via A_1 or A_4

- **Update:** `upd A2; upd A4; upd T1; upd C1` ✗
- **Update:** `upd A2; upd A4; upd C1; upd T1`
- ✓

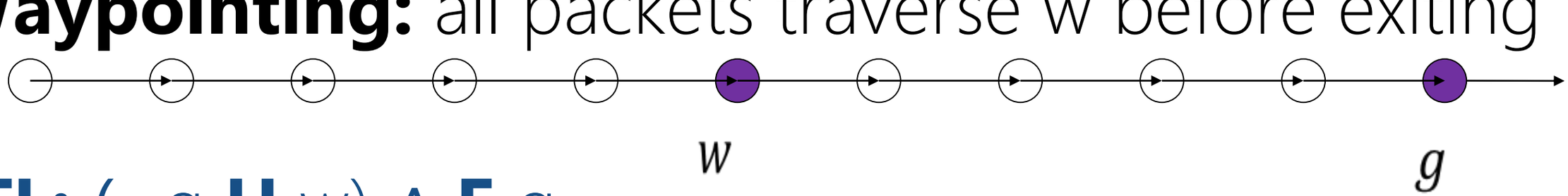
How to Specify Properties?

Reachability: every packet that starts at s_i reaches d_i



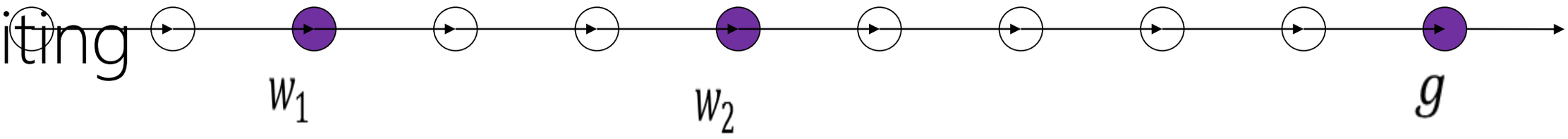
LTL: (s_i →**F** d_i)

Waypointing: all packets traverse w before exiting



LTL: (¬g **U** w) ∧ **F** g

Chaining: all packets traverse w_1 and w_2 before exiting



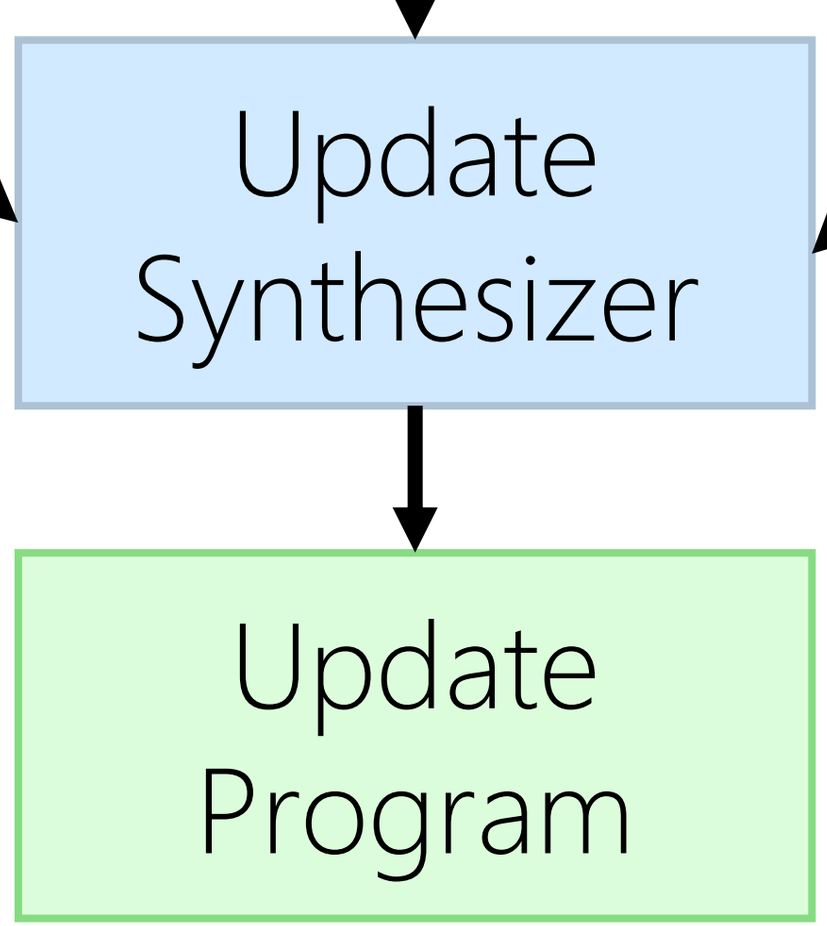
LTL: (¬g **U** w_2)∧ (¬w_2 **U** w_1)∧ **F** g

Network Update Synthesis

Initial and Final Configuration

LTL Specification

Update at most once

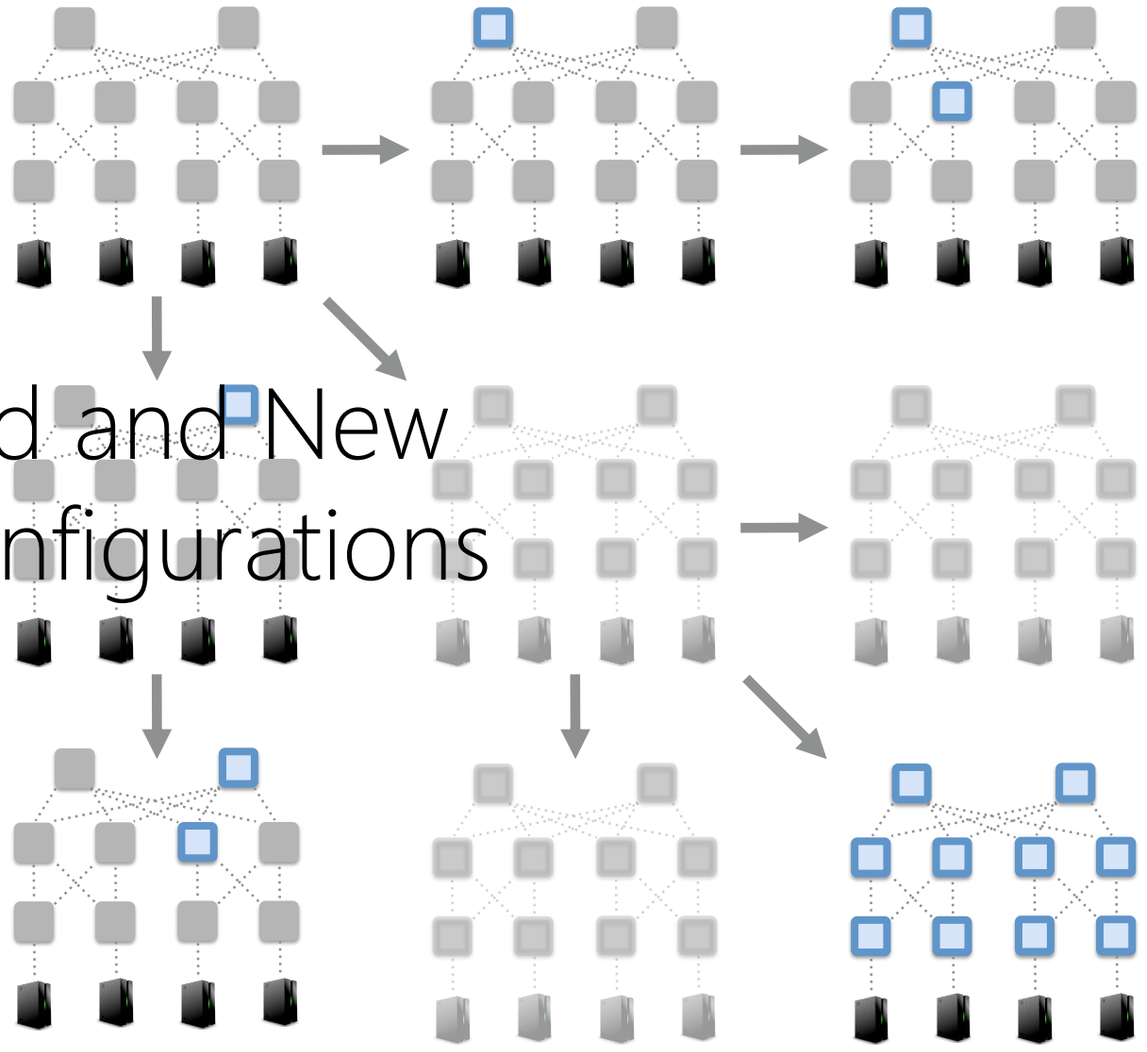


Synthesis Algorithm

φ

LTL Specification

Old and New Configurations



Synthesis Algorithm

Depth-First Search:

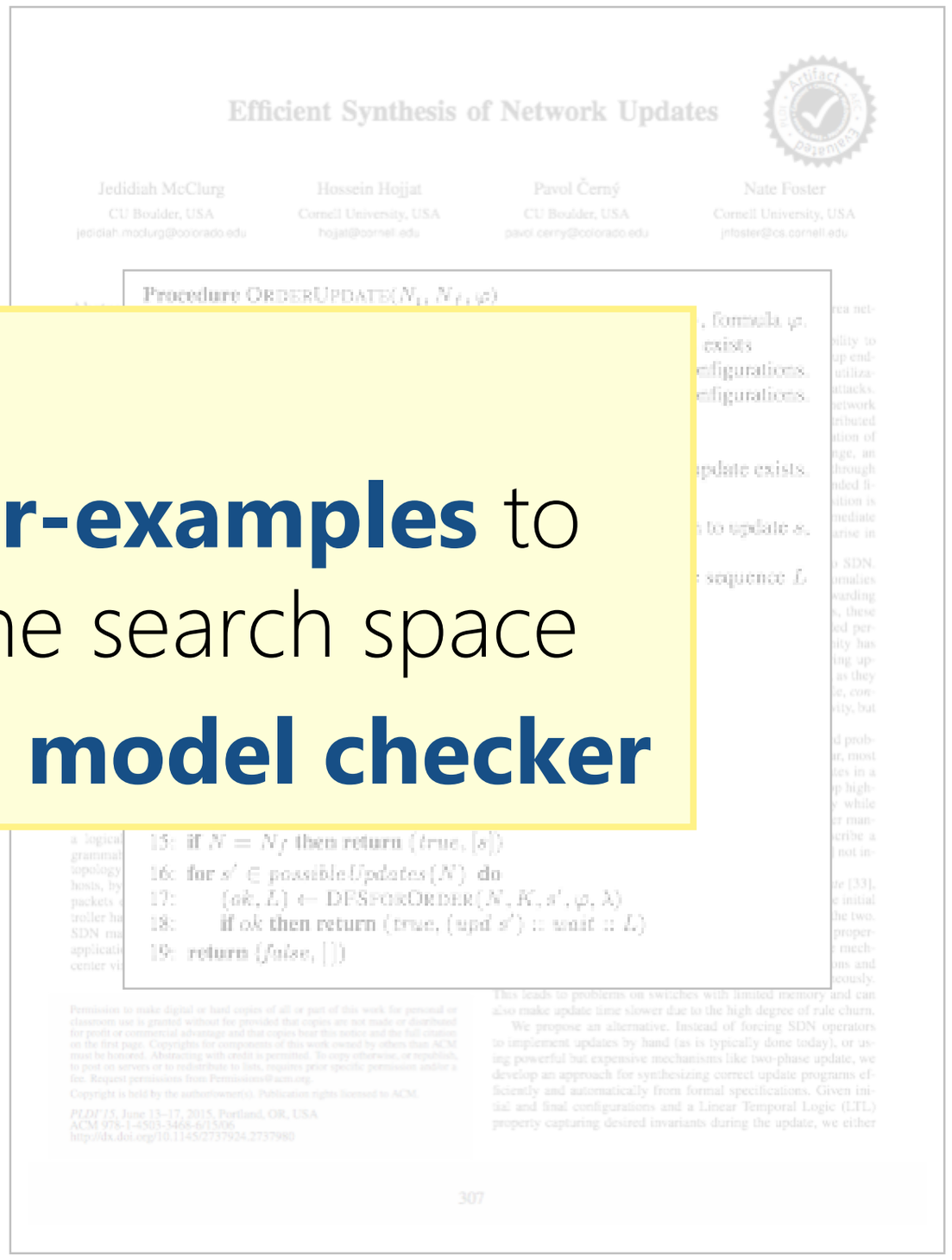
- Attempt to update the switches one-by-one
- Backt
bad c
reach

Two main ideas:

- **Learn from counter-examples** to aggressively prune the search space
- Use an **incremental model checker**

Challe

- Searc
- Checking a configuration means solving an LTL model checking problem (PSPACE-complete)!

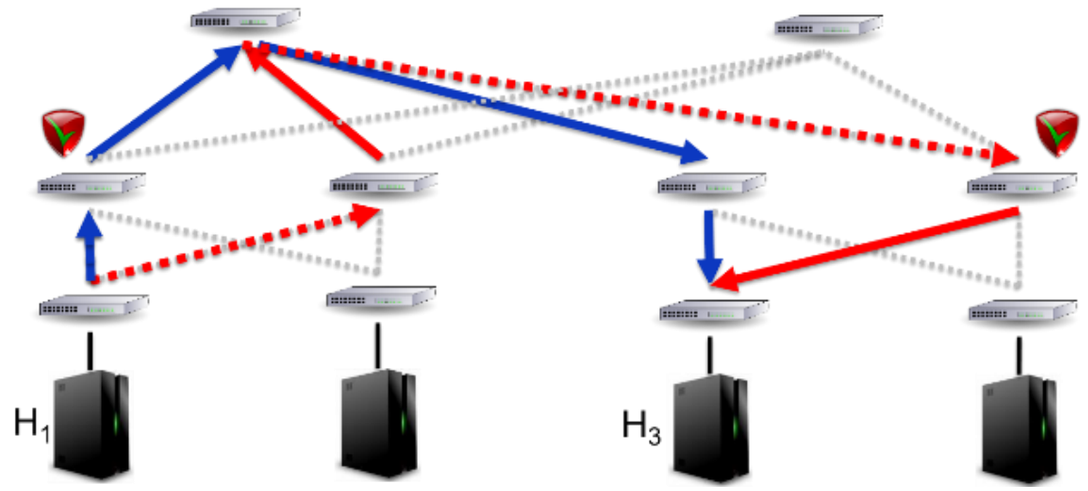

Efficient Synthesis of Network Updates

Jedidiah McClurg, CU Boulder, USA, jedidiah.mcclurg@colorado.edu
Hossein Hojjat, Cornell University, USA, hojjat@cornel.edu
Pavol Černý, CU Boulder, USA, pavol.cerny@colorado.edu
Nate Foster, Cornell University, USA, jnfoster@cs.cornell.edu

Procedure ORDERUPDATE(N_i, N_f, φ)
15: if $N = N_f$ then return $(true, [s])$
16: for $s' \in possibleUpdates(N)$ do
17: $(ok, L) \leftarrow$ DFSFORORDER($N, K, s', \varphi, \lambda$)
18: if ok then return $(true, (upd\ s') :: wait :: L)$
19: return $(false, [])$

Model Checking

Model M:

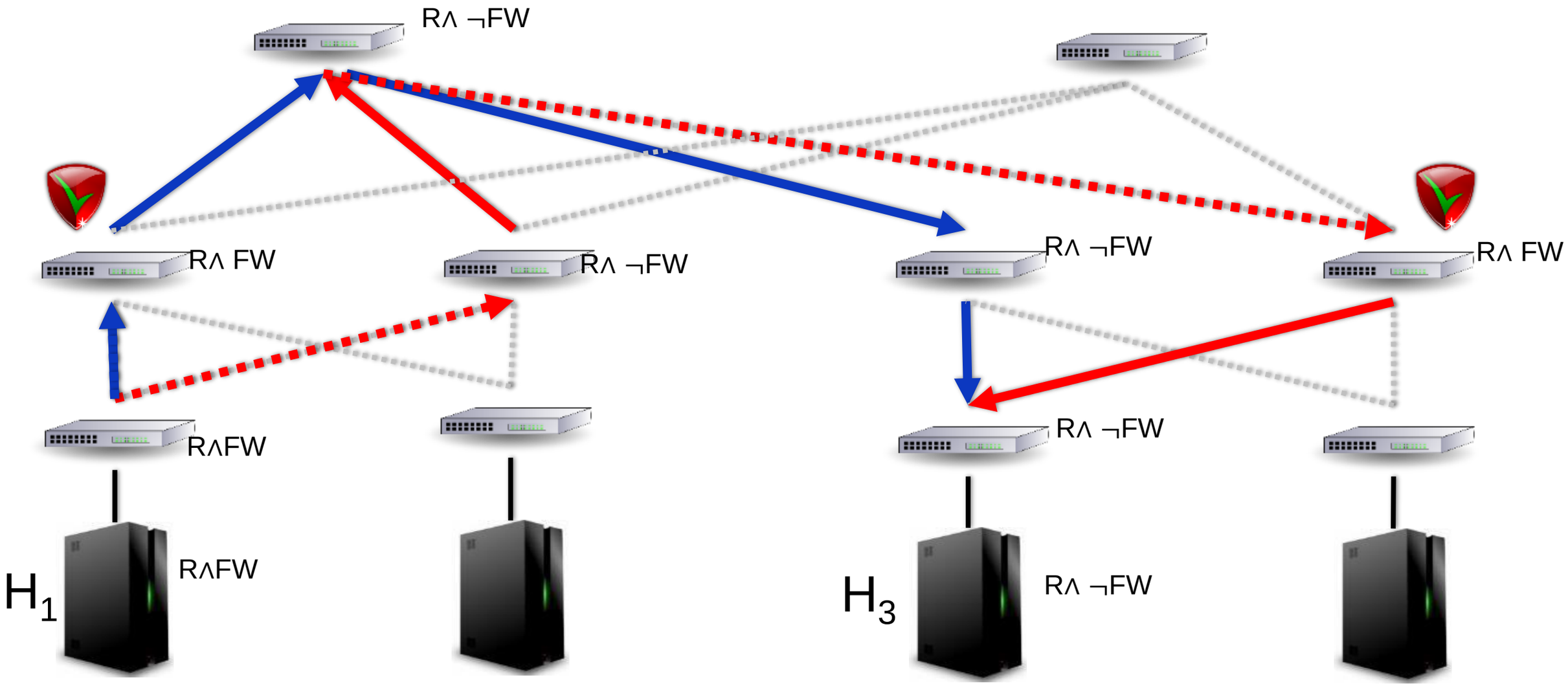


Specification S:

- ❑ all packets reach H3
- ❑ all packets traverse firewall

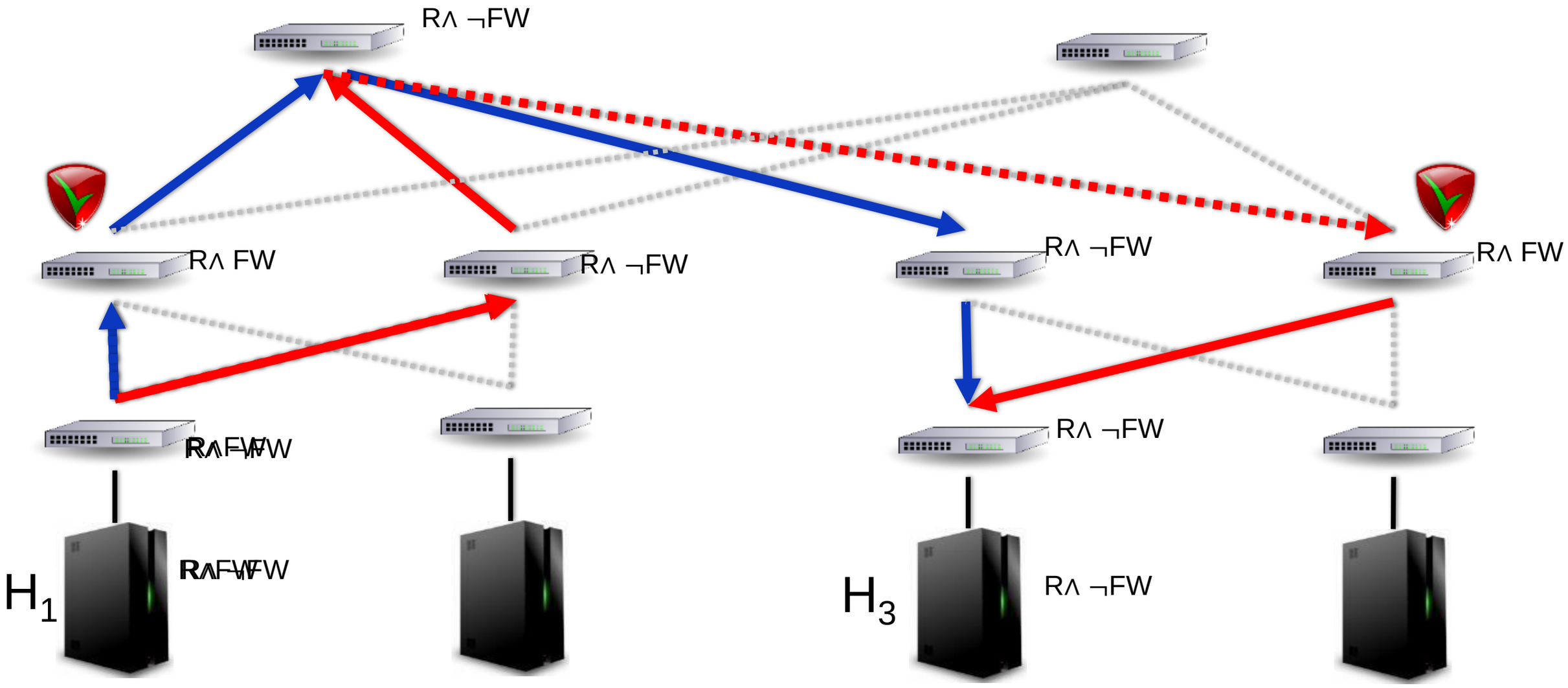
Question: Does M satisfy S?

Model Checking



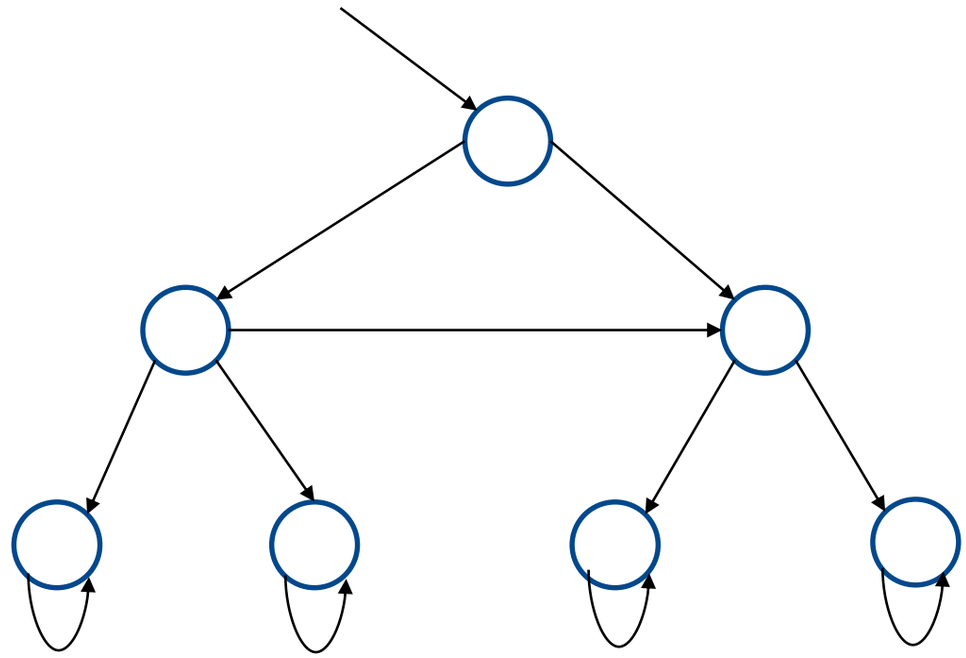
- R holds at a switch s if all packets that traverse s reach H3
- FW holds at a switch s if all packets that traverse s then traverse firewall

Incremental model checking



- R holds at a switch s if all packets that traverse s reach H3
- FW holds at a switch s if all packets that traverse s then traverse firewall

Model checking loop-free structures



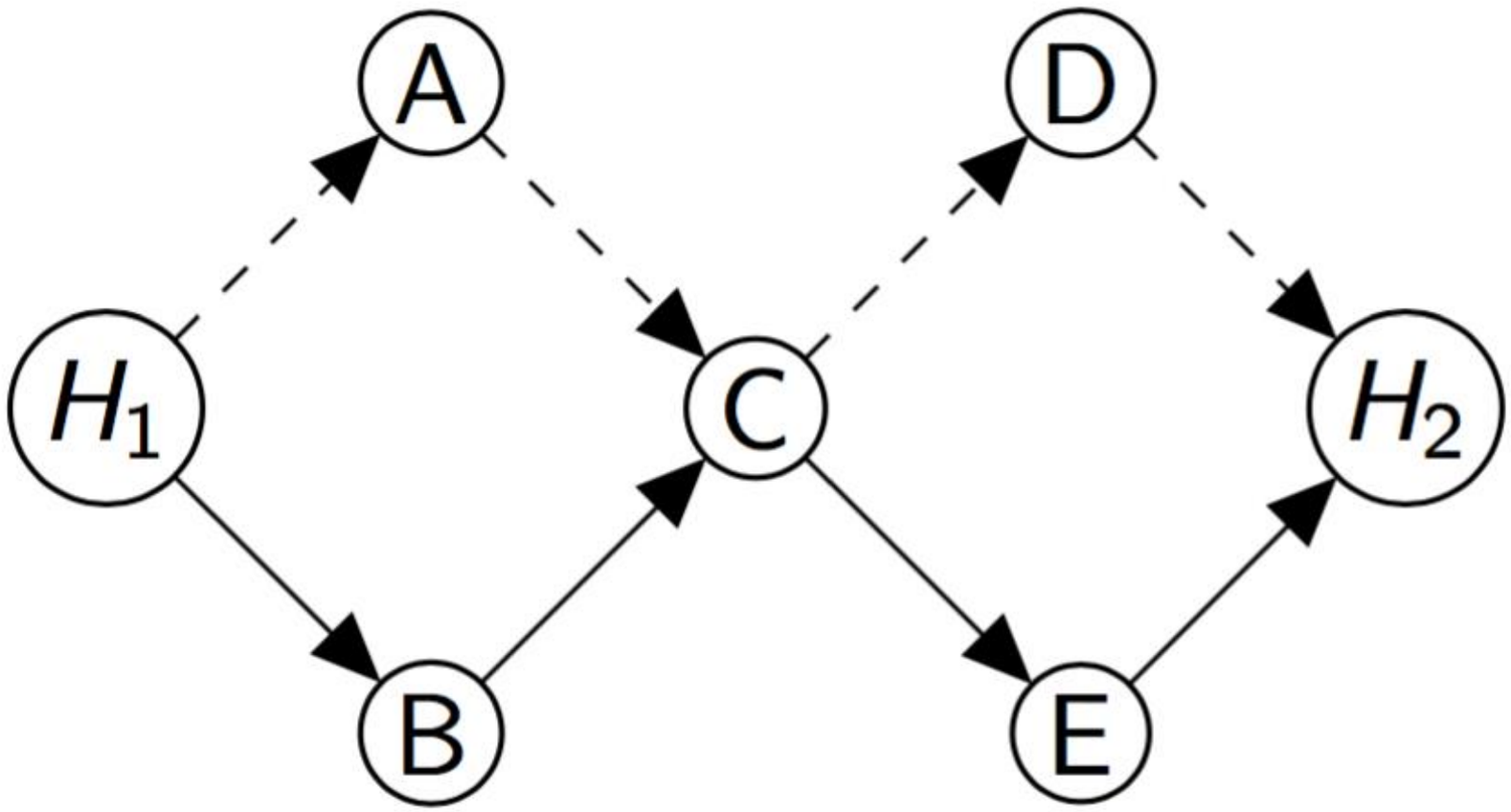
One sentence summary:

The idea is the same as in LTL-to-Büchi construction, but on loop-free structures it is possible to check all constraints locally (no need for the Büchi condition)

Main Limitation

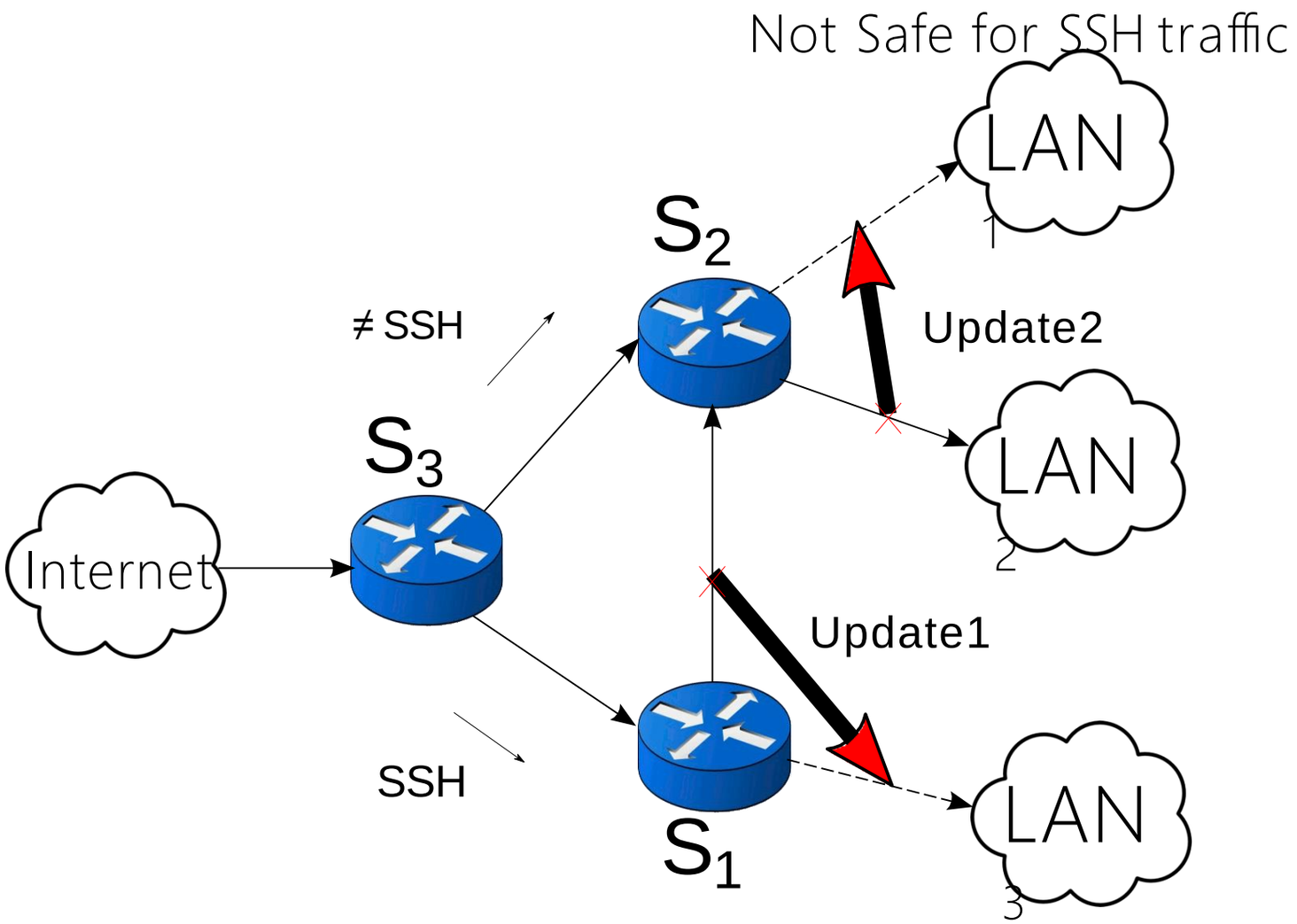
For some topologies, configurations, and specifications, there is no correct ordering we can use

Example: "double diamond" [DISC '16]



Our implementation reverts to a two-phase update…

Waits


Not Safe for SSH traffic
LAN
2
S_2
≠ SSH
Update2
S_3
LAN
3
Internet
Update1
SSH
S_1
LAN
3

Waits

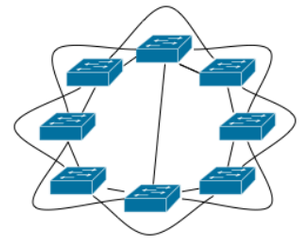
- Correspondence to weak memory systems
- Equivalence of two problems:

1) Finding a correct and efficient placement of fences for a concurrent program under weak memory model
2) Finding minimum number of waits for an update sequence

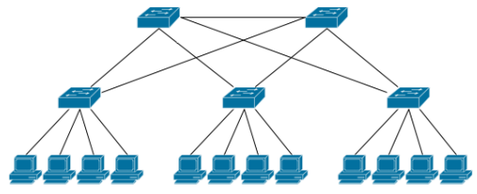
Evaluation

Questions:

- Impact of optimizations:
  - Pruning search space
  - Incremental model checking
- Scalability of approach:
  - Topology
  - Complexity of specifications
  - Total space explored



Small-world

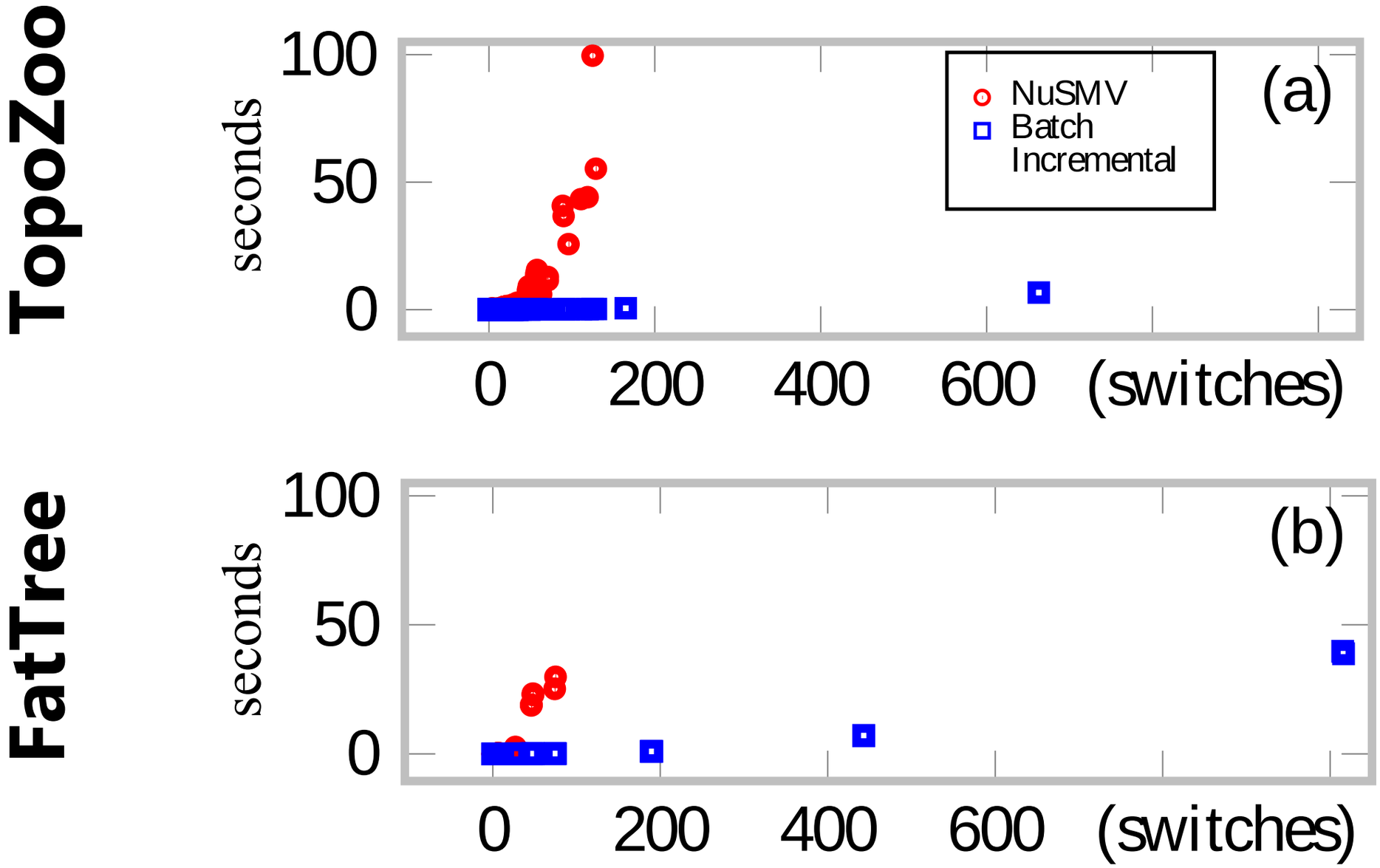


Fattree

Methodology:

- Real-world topologies (TopoZoo, FatTrees, Small World)
- Synthetic configurations (e.g., shortest-path forwarding)

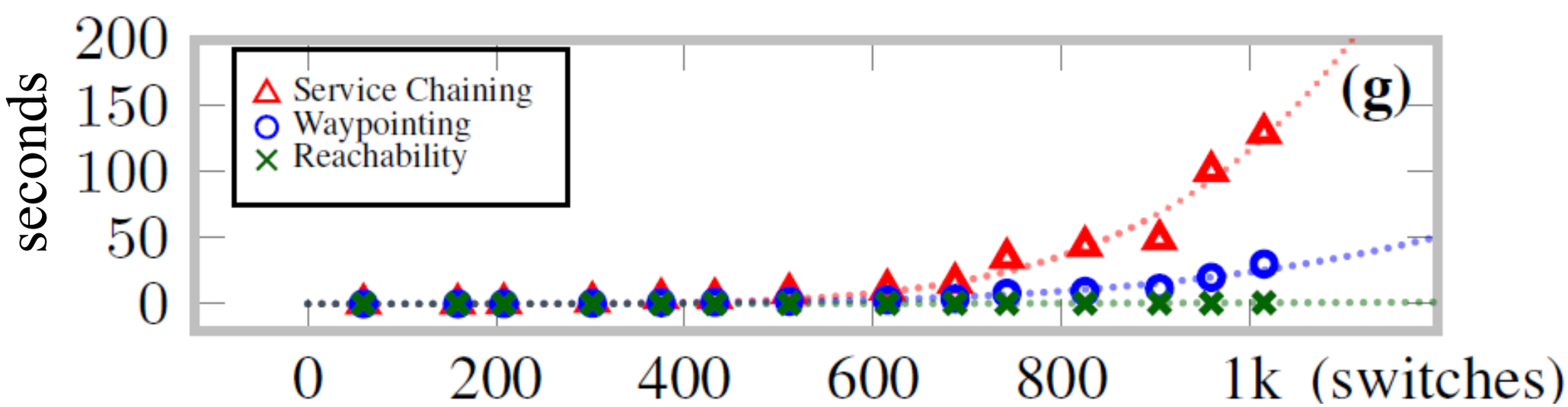
Impact of Optimizations



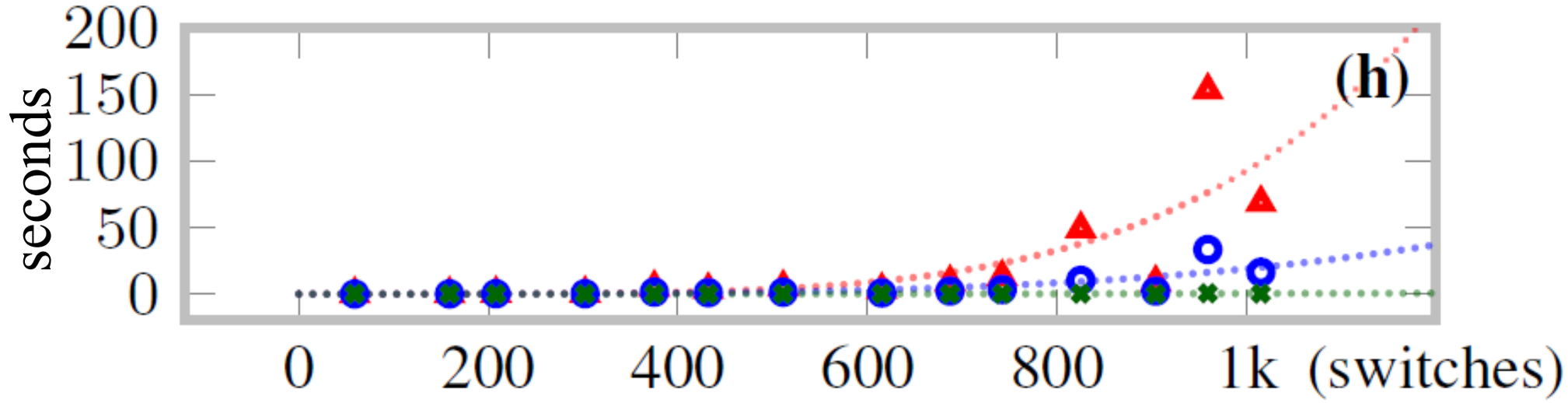
- **Configurations:** shortest-path forwarding

Scalability

Feasible



Infeasible

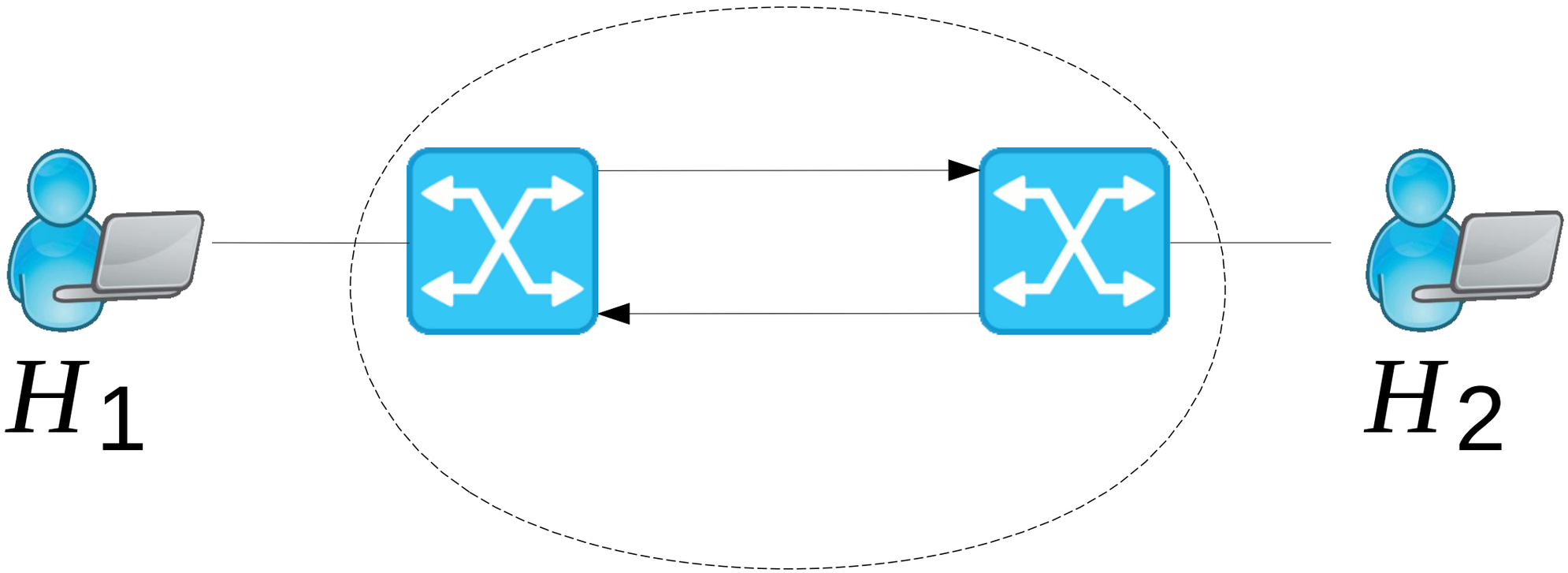


- **Configurations:** "diamond" / "double diamond"
- **Specifications:** reachability, waypointing, chaining

Synchronization for Network Programs

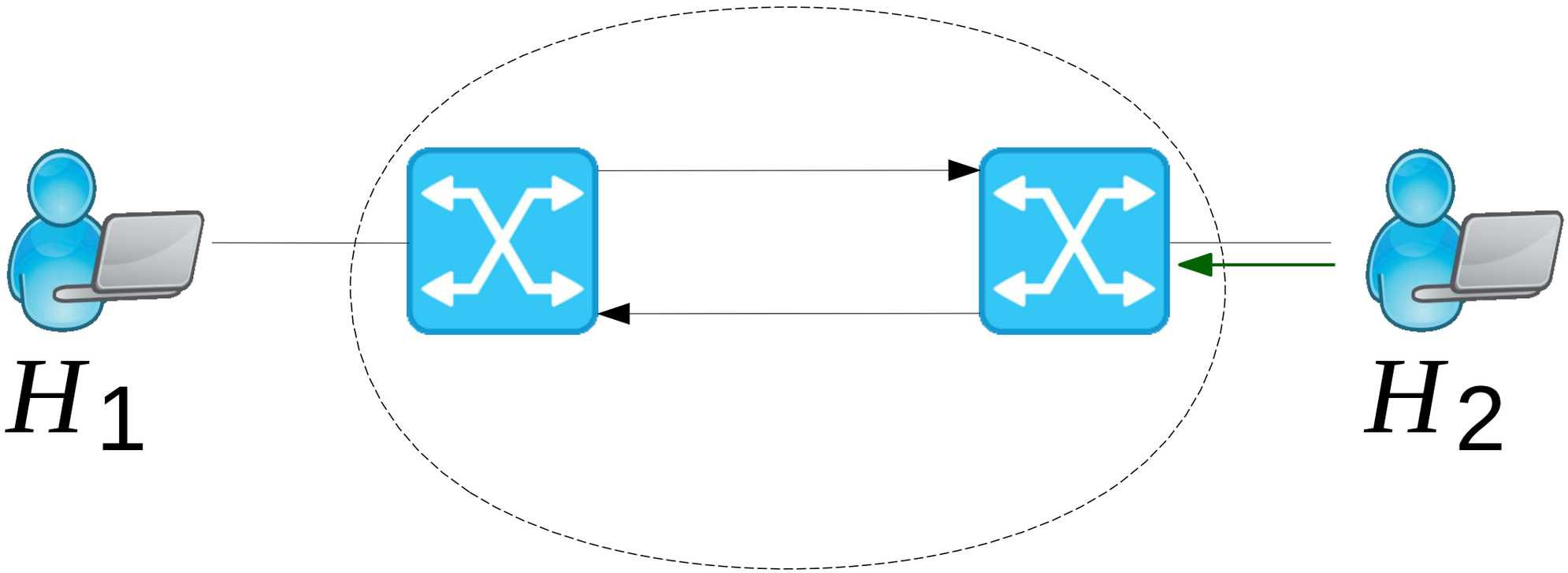
[PLDI'16,CAV '17]

Stateful Firewall



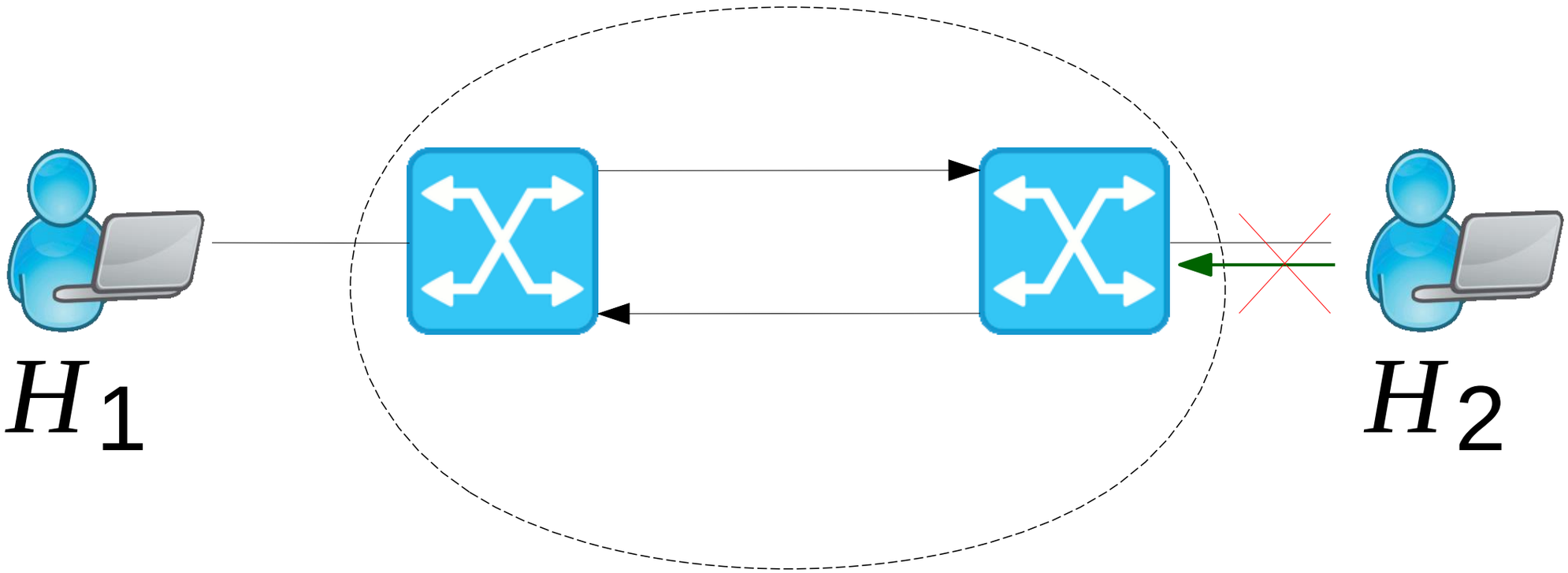
- H_1 should be allowed to communicate with H_2
- H_2 should only be allowed to communicate with H_1 if H_1 has previously initiated a connection

Stateful Firewall



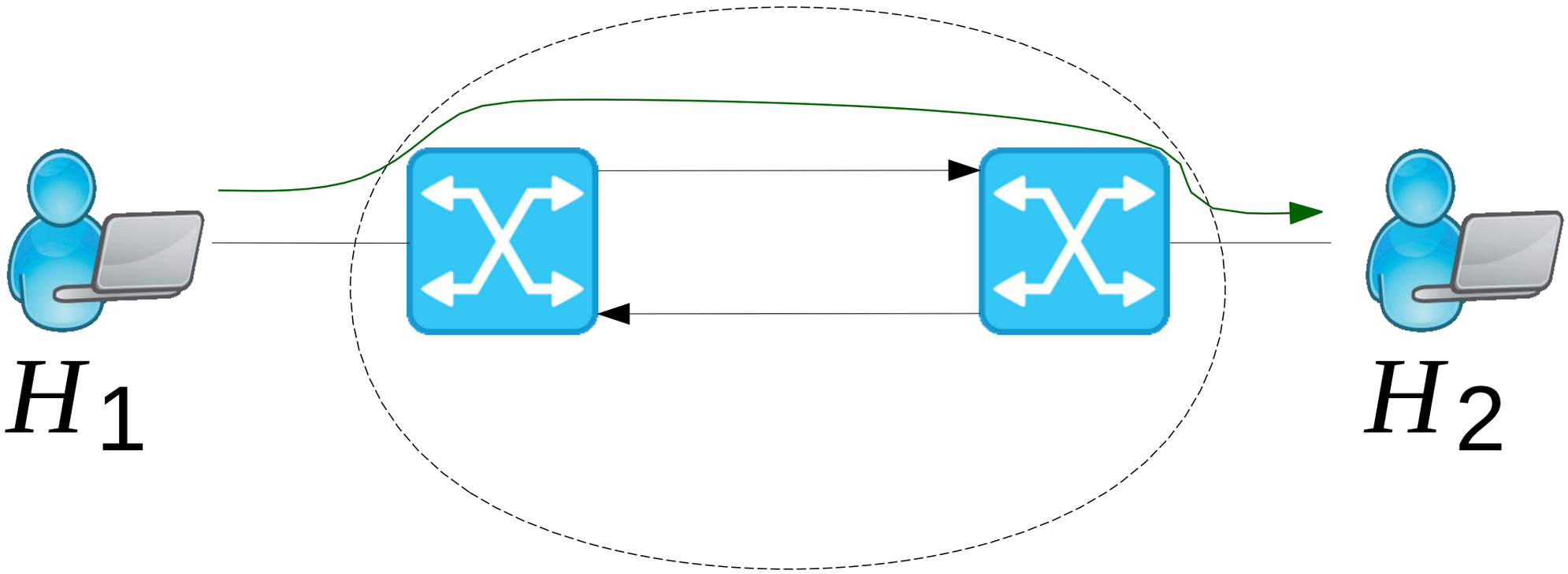
- H_1 should be allowed to communicate with H_2
- H_2 should only be allowed to communicate with H_1 if H_1 has previously initiated a connection

Stateful Firewall



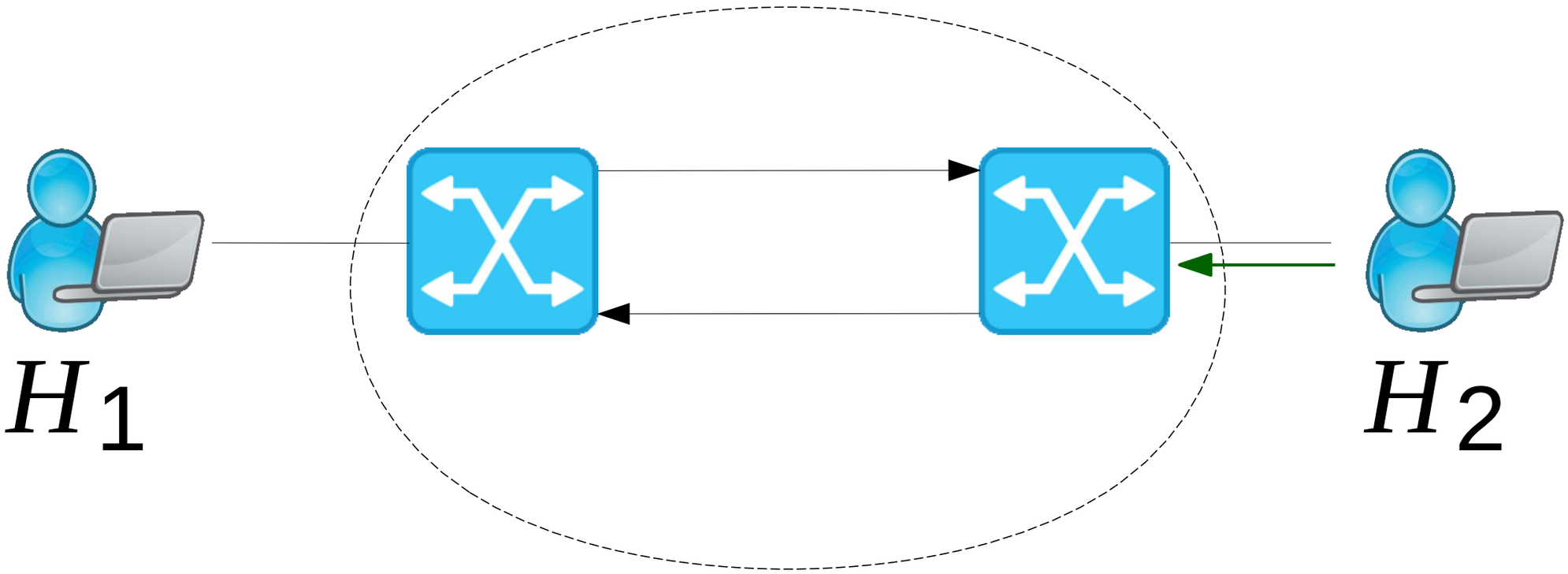
- H_1 should be allowed to communicate with H_2
- H_2 should only be allowed to communicate with H_1 if H_1 has previously initiated a connection

Stateful Firewall



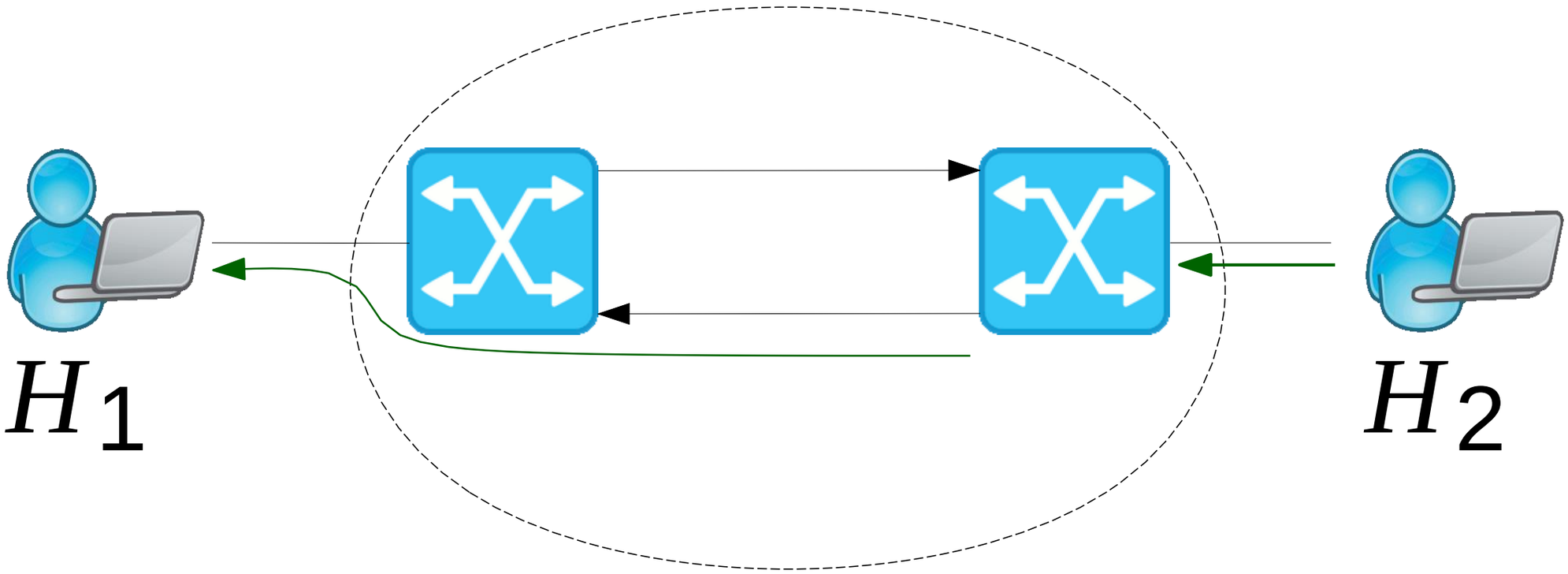
- H_1 should be allowed to communicate with H_2
- H_2 should only be allowed to communicate with H_1 if H_1 has previously initiated a connection

Stateful Firewall



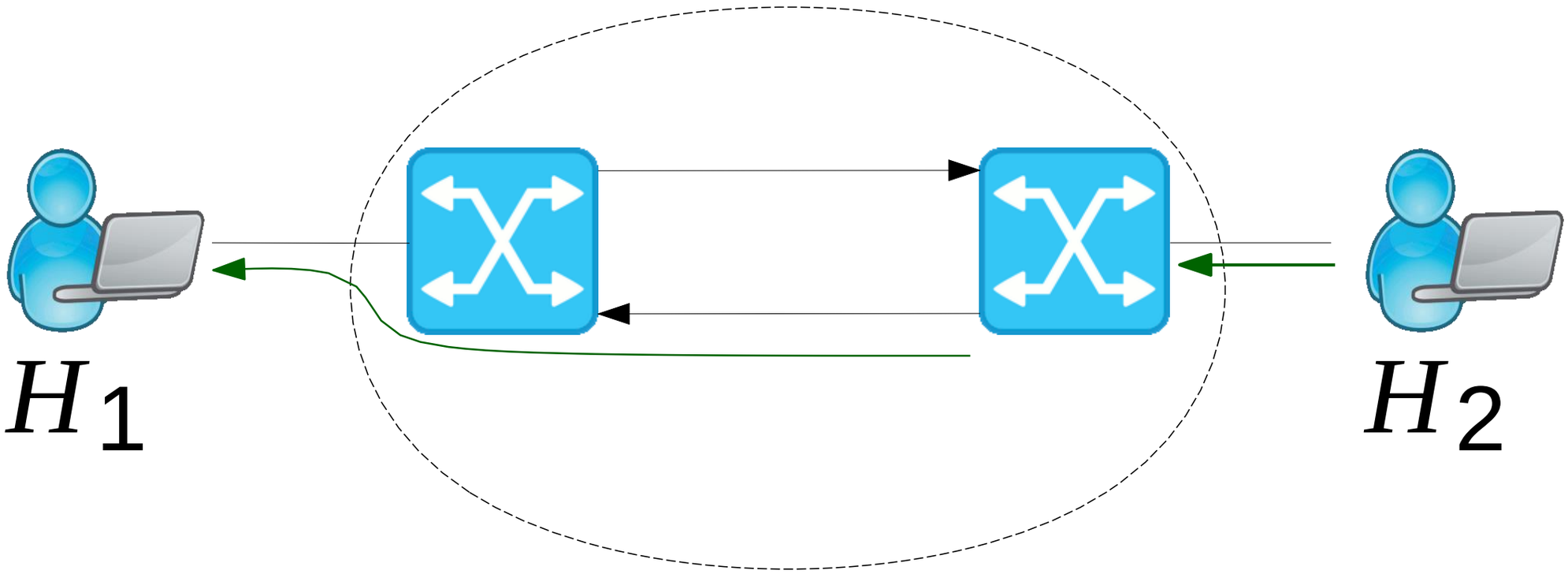
- H_1 should be allowed to communicate with H_2
- H_2 should only be allowed to communicate with H_1 if H_1 has previously initiated a connection

Stateful Firewall

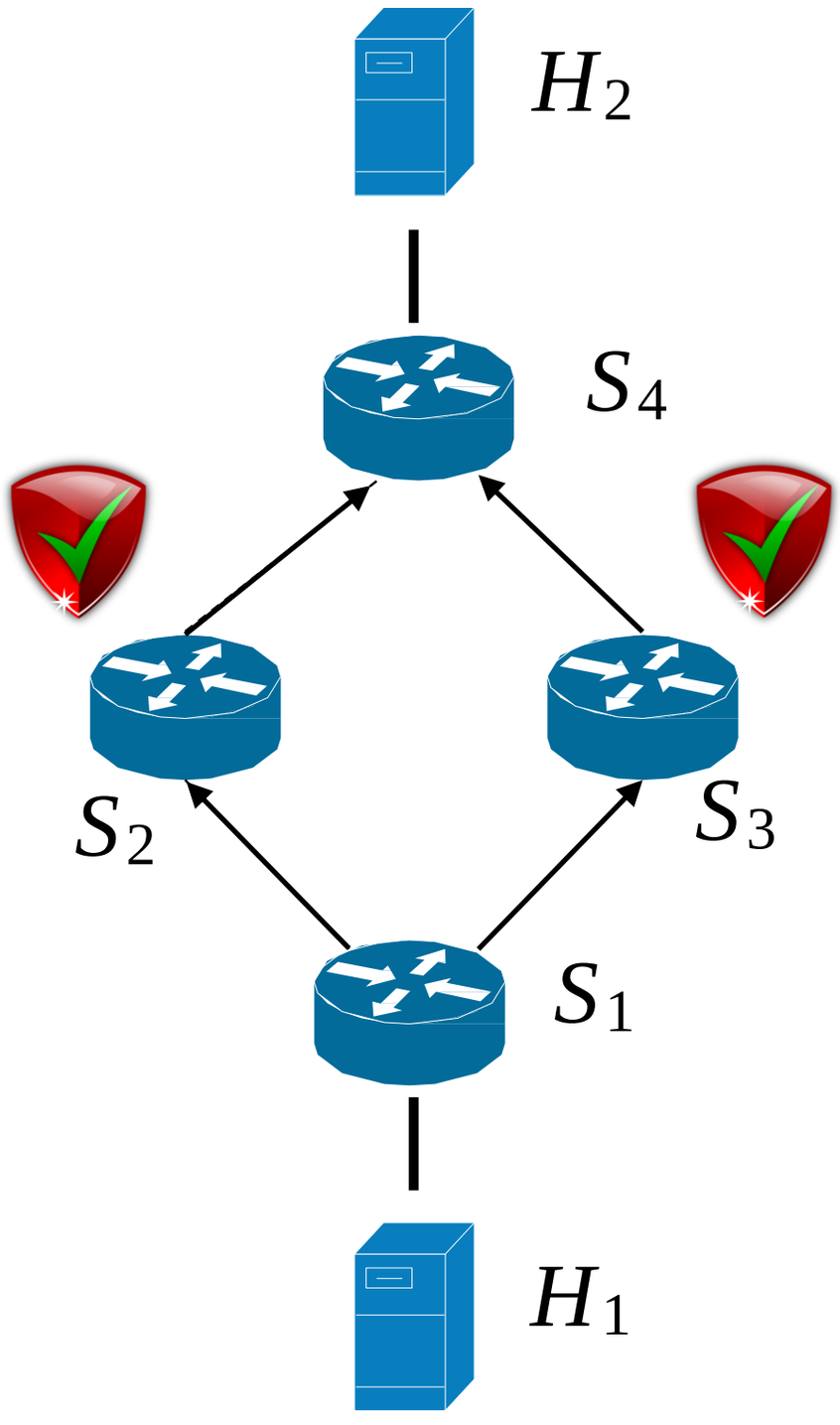


- H_1 should be allowed to communicate with H_2
- H_2 should only be allowed to communicate with H_1 if H_1 has previously initiated a connection

Stateful Firewall



- An **event** can trigger a configuration change
- Bug: packet race – we need guarantees about when configurations change with respect to events
- Don't respond to an event too late (and don't respond too early)!



Initial configuration:

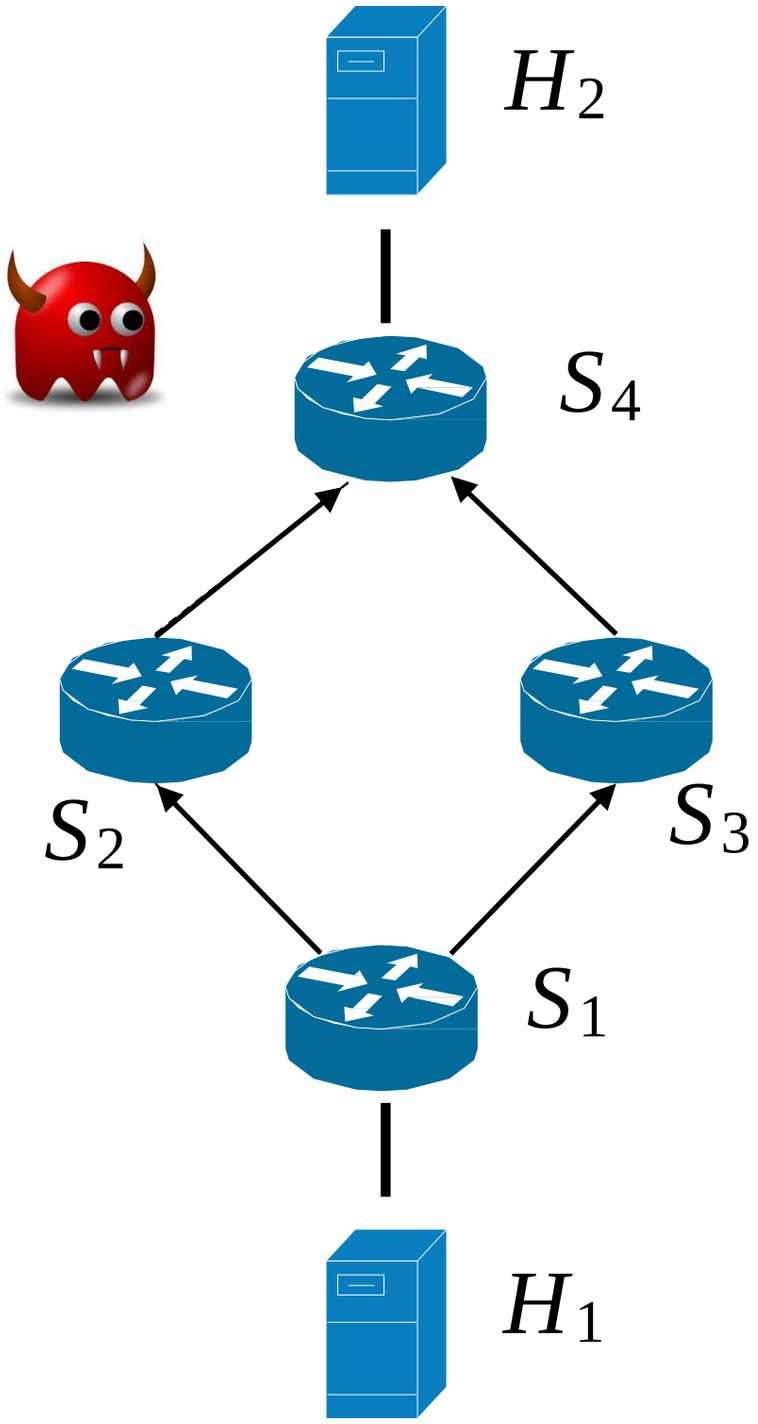
- Forward from H_1 to H_2 via S_1-S_3-S_4
- S_3 has a firewall

Load balancer at S_1:

- Throughput greater than 500: Start load balancing
  - using path through S_2
- Throughput less than 400: Stop load balancing

Firewall on S_2:

Operator can enable/disable firewall rules installed at S_2



Problem:

- Load balancer on and
- Firewall on S_2 off

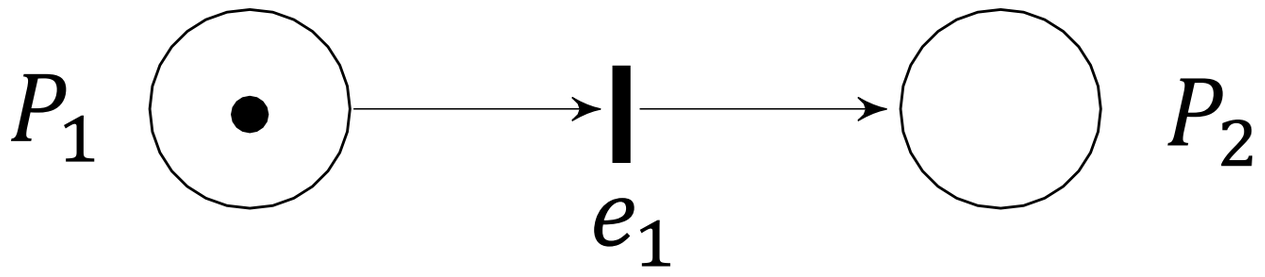
General problem:

- Synchronization for event-driven network programs

Solution

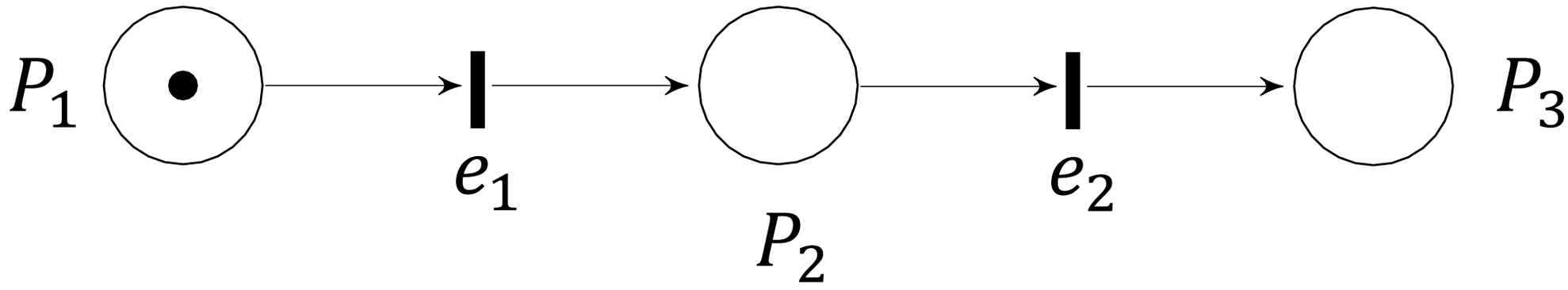
- Programming model: event nets
- Algorithmic synthesis of synchronization

Event nets: One event-update

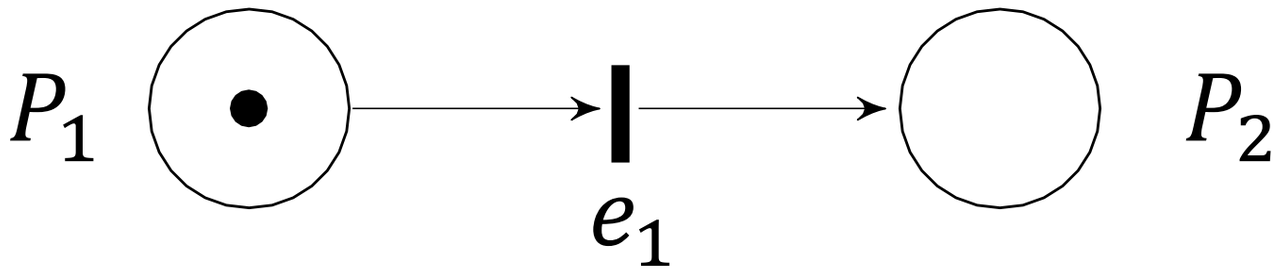


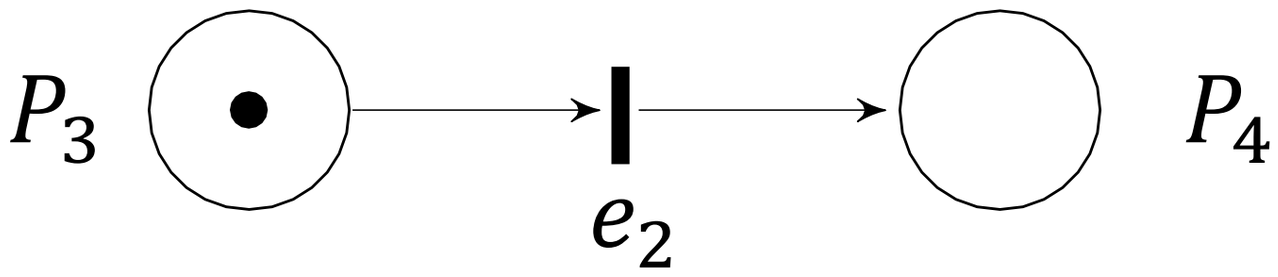
- Places labeled by configurations
- Transitions labeled by events
- 1-safe Petri-nets
- Can be implemented without packet races
  - (first part of the tutorial)
- Logical time bounds on when to change the configuration can be given [PLDI16]

Event nets: Sequential Composition

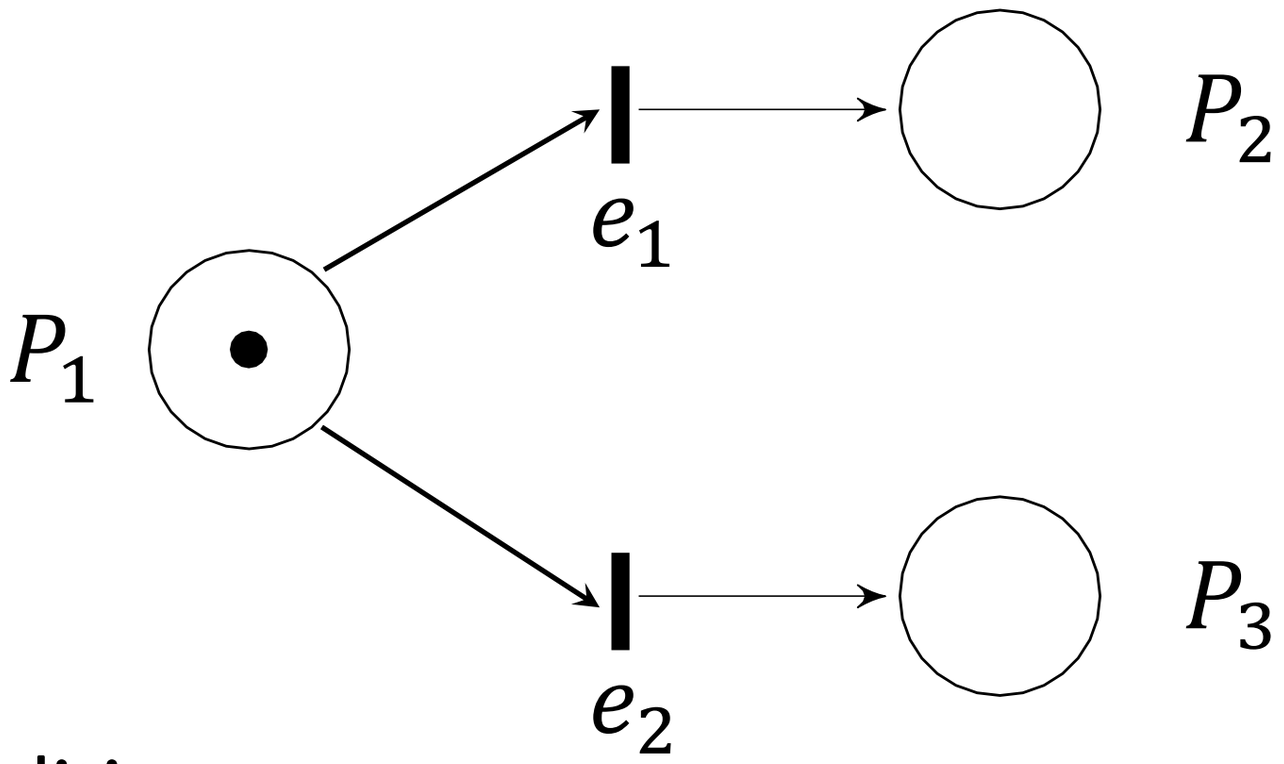


Event nets: Parallel Composition





Event nets: Conflicting event-updates

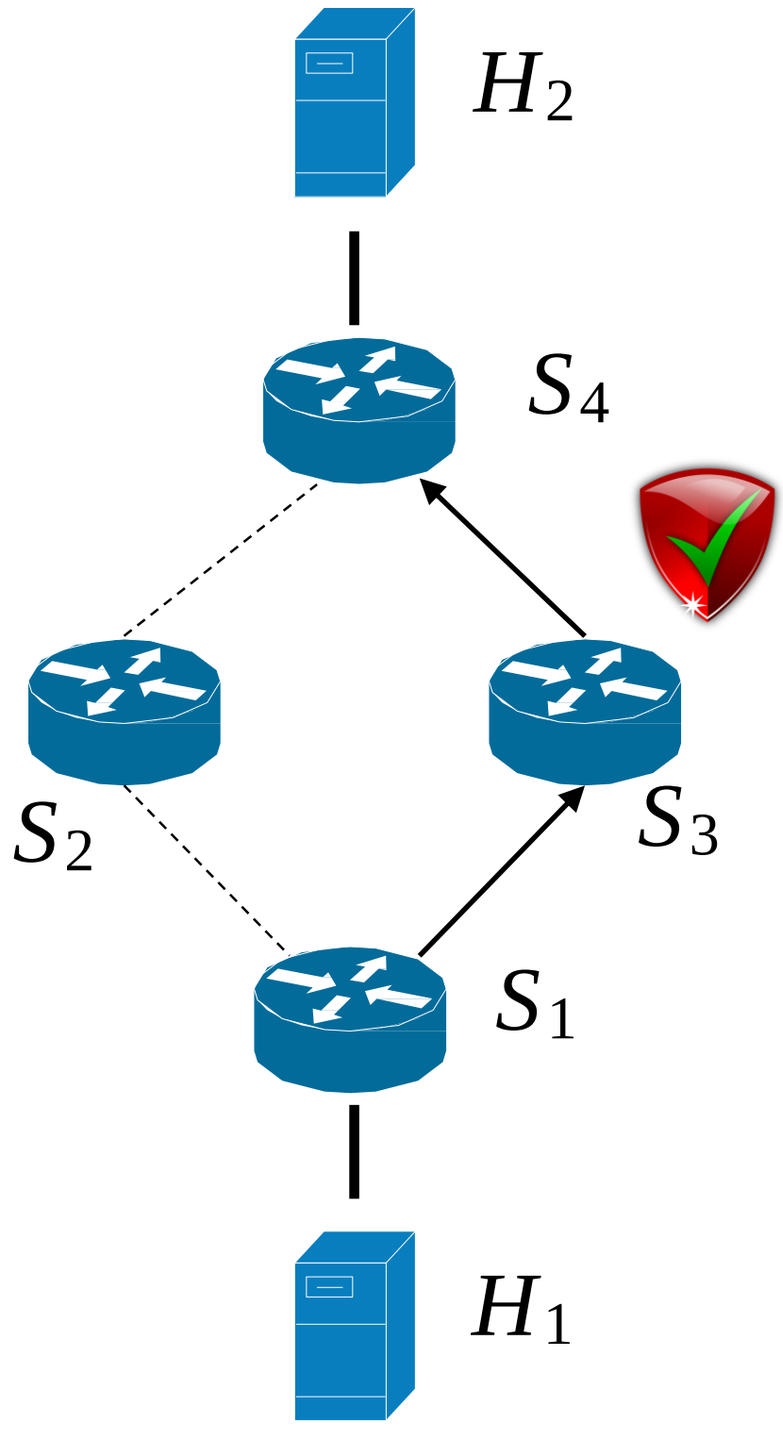


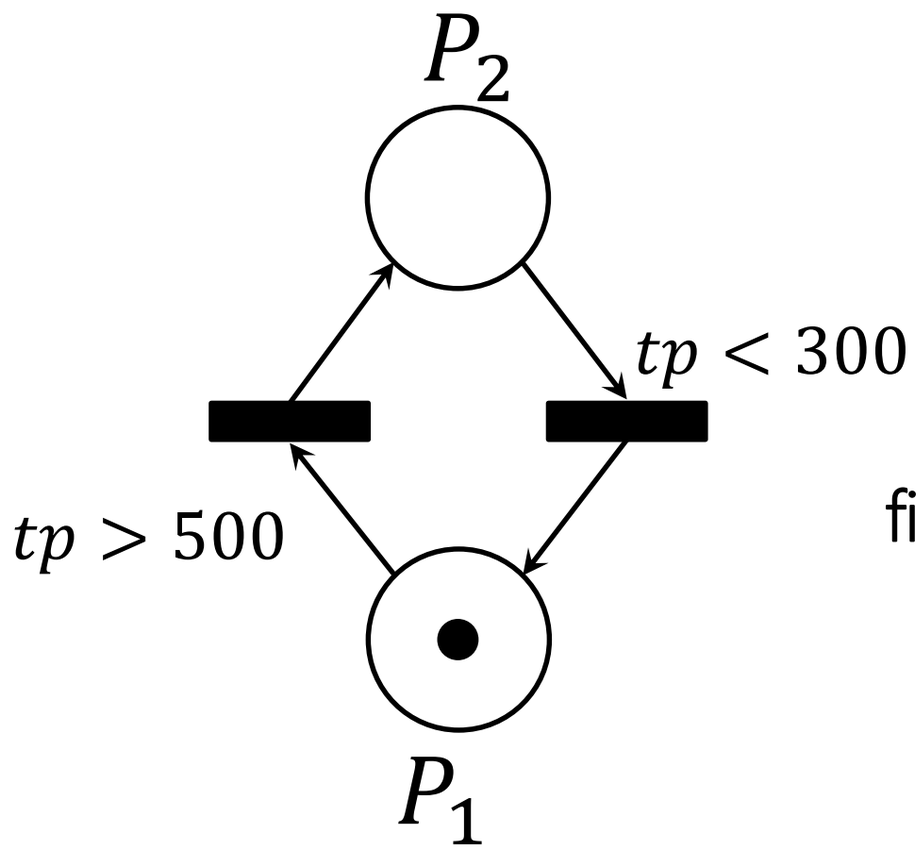
- Locality condition:
  which transition to take (e_1 or e_2) must be decided locally
- Otherwise availability cannot be maintained [PLDI16] (usual Consistency-Availability tension)

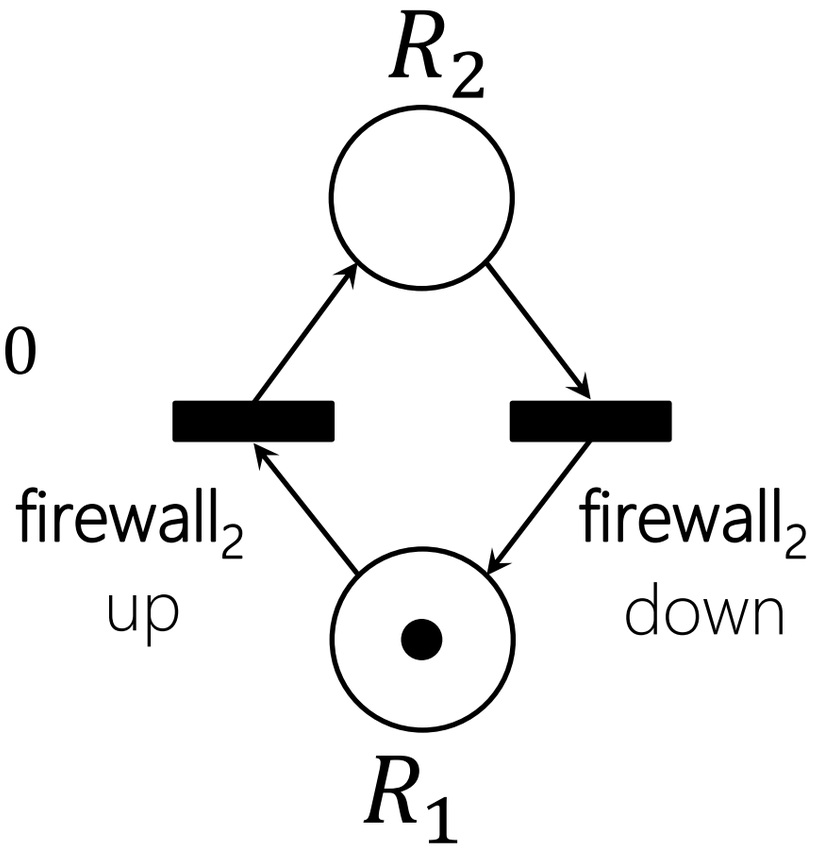
Programming Model: Event nets

- Can be implemented without packet races
- Can be implemented without losing availability (under the locality restriction)
- Synchronization can be added to prevent controller races

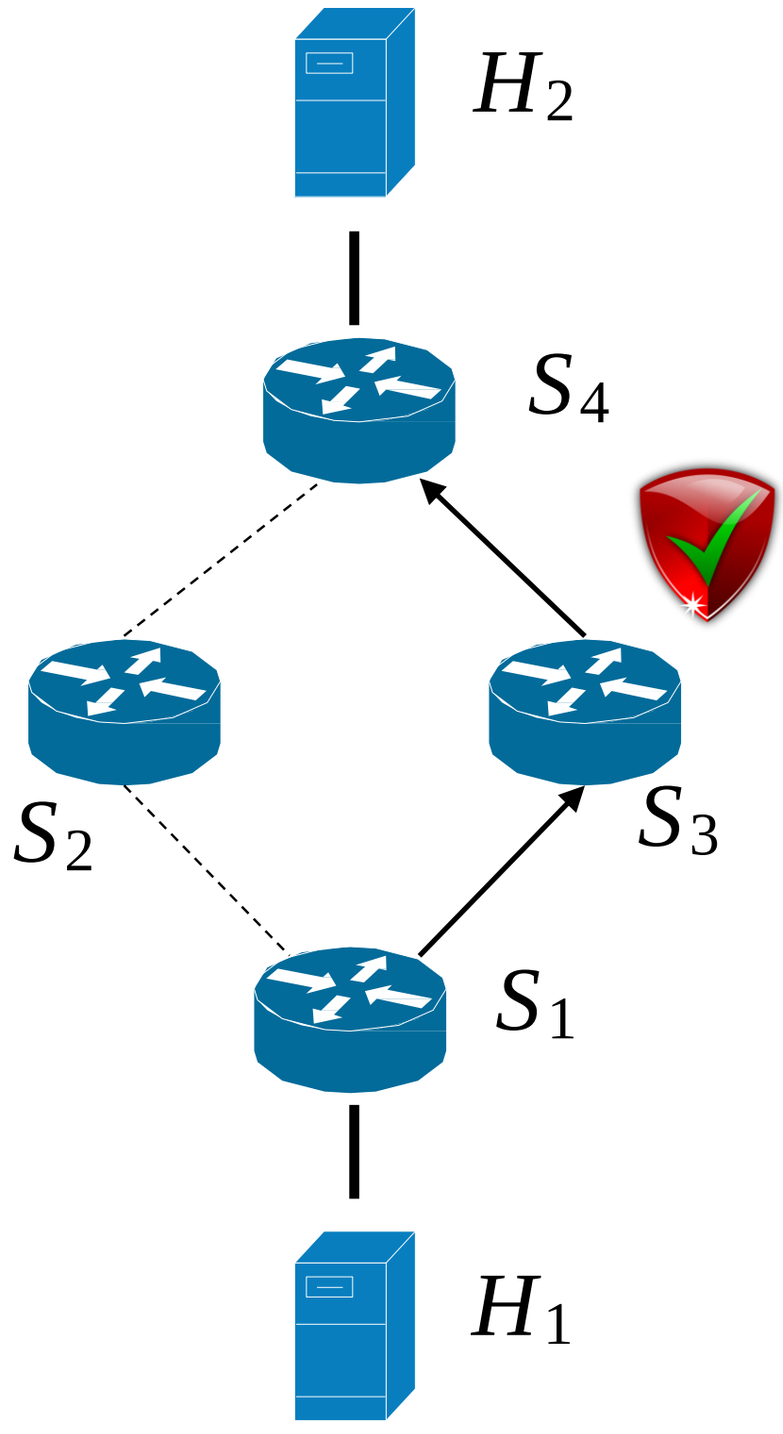
Load Balancing

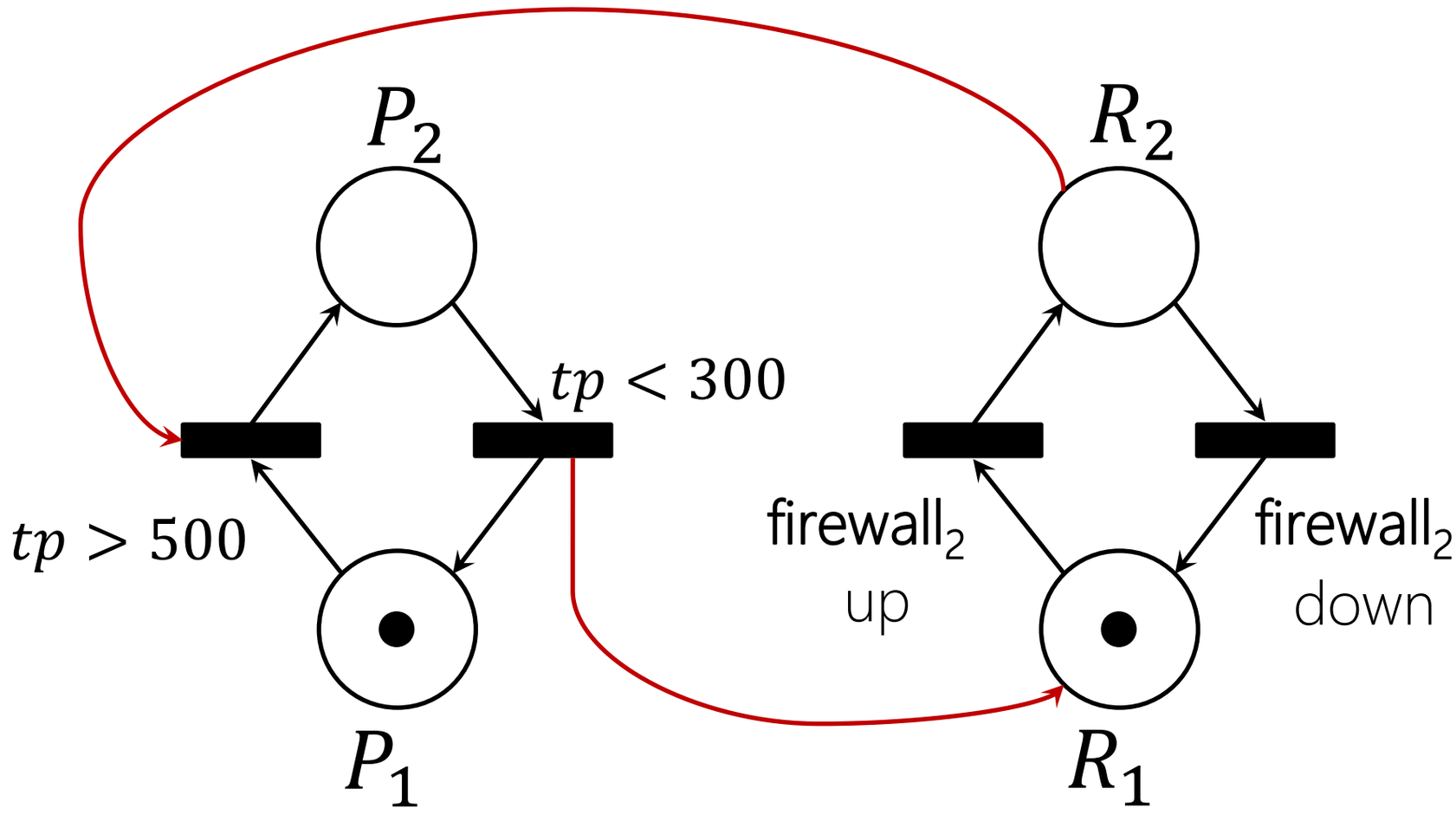






Load Balancing: synchronized

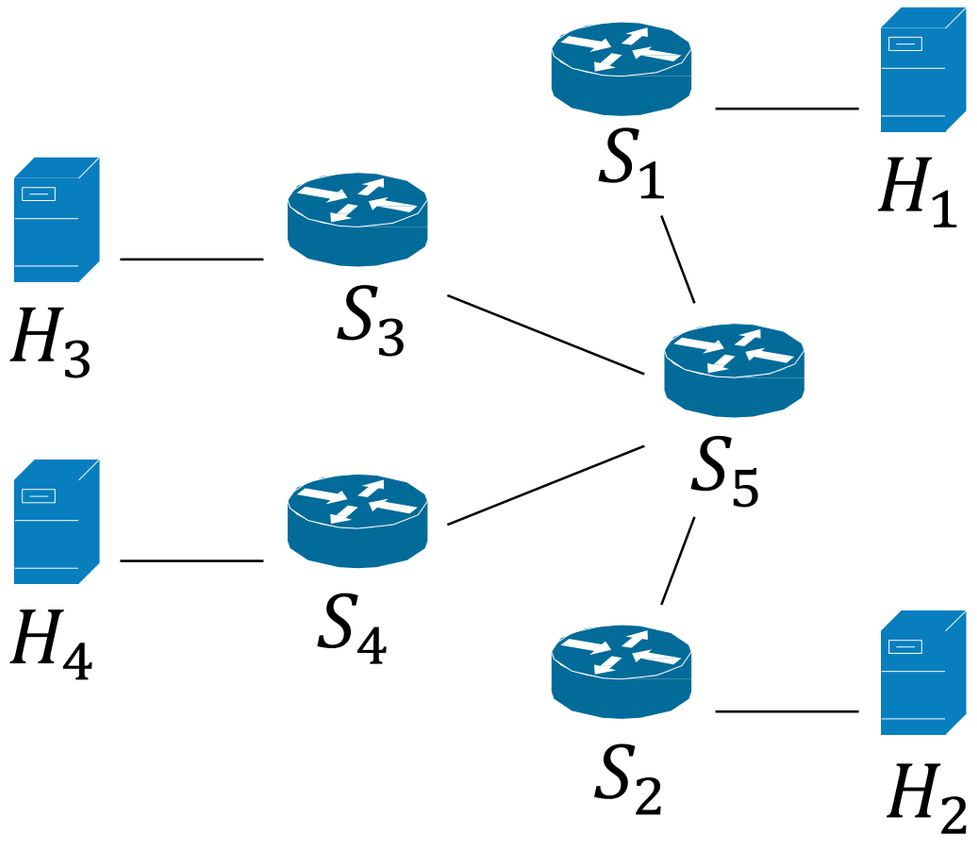




Synchronization Synthesis Eliminating Controller Races

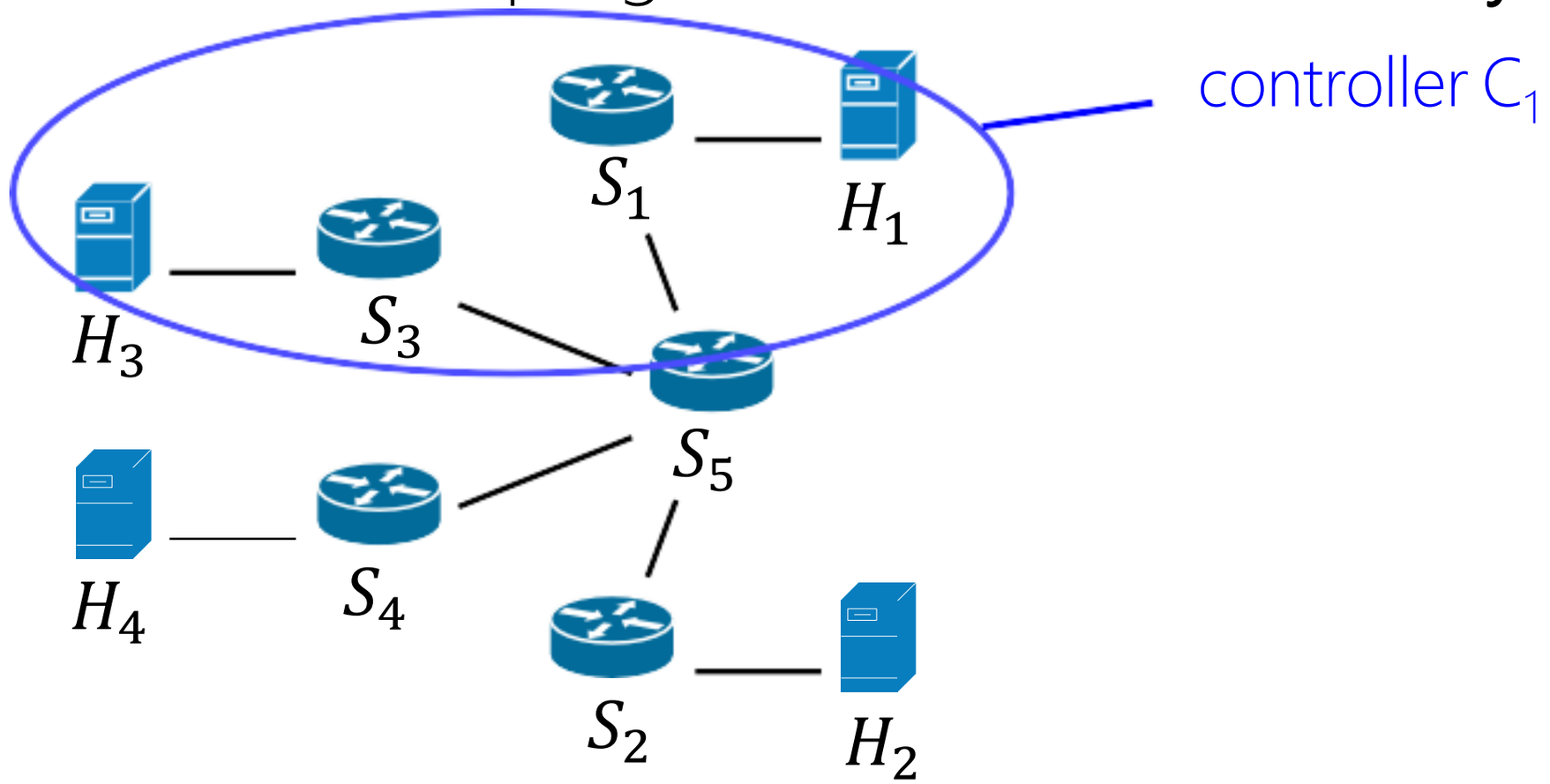
Concurrent network programs

Correctness when network programs execute **concurrently**?



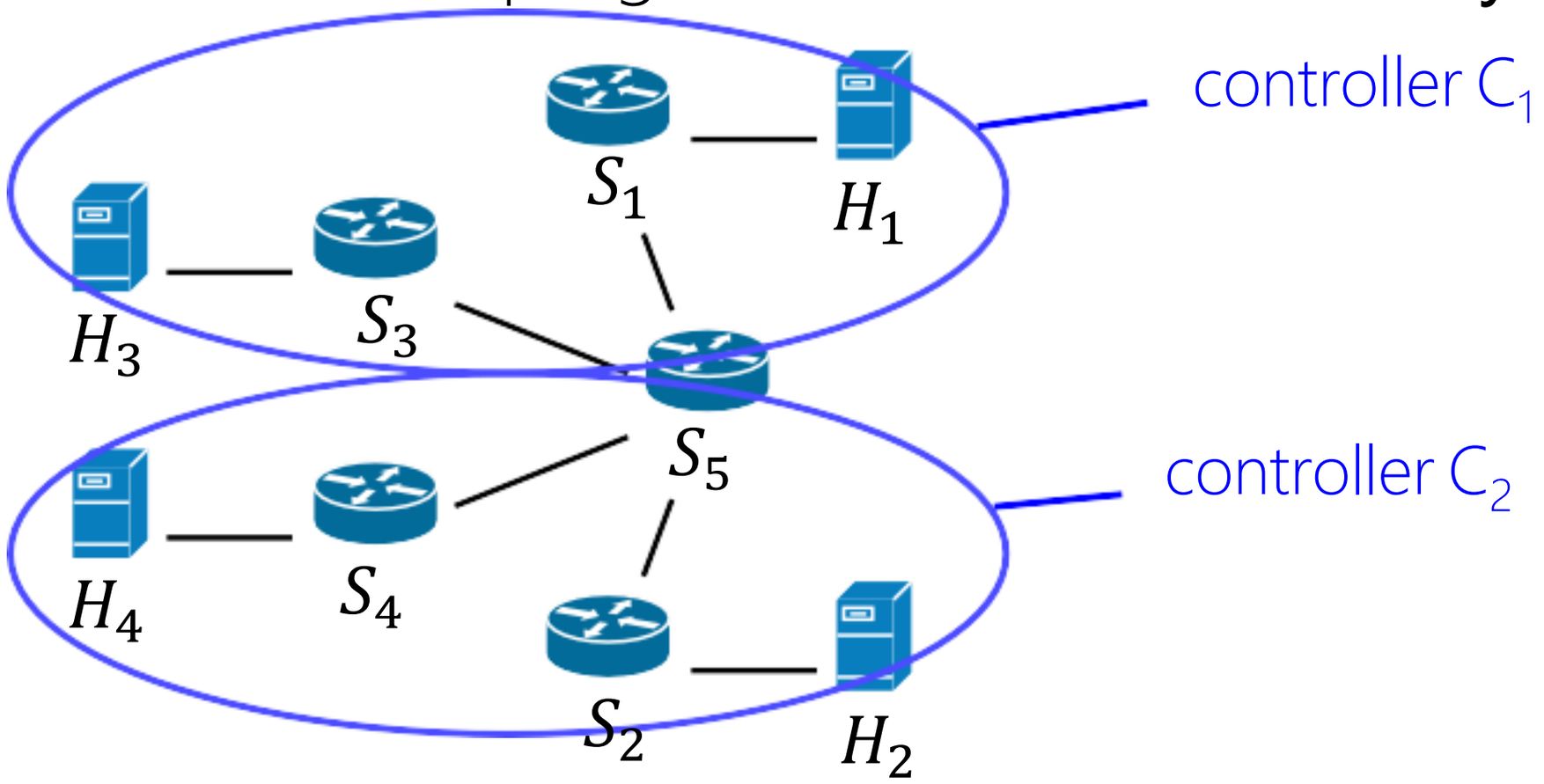
Concurrent network programs

Correctness when network programs execute **concurrently**?



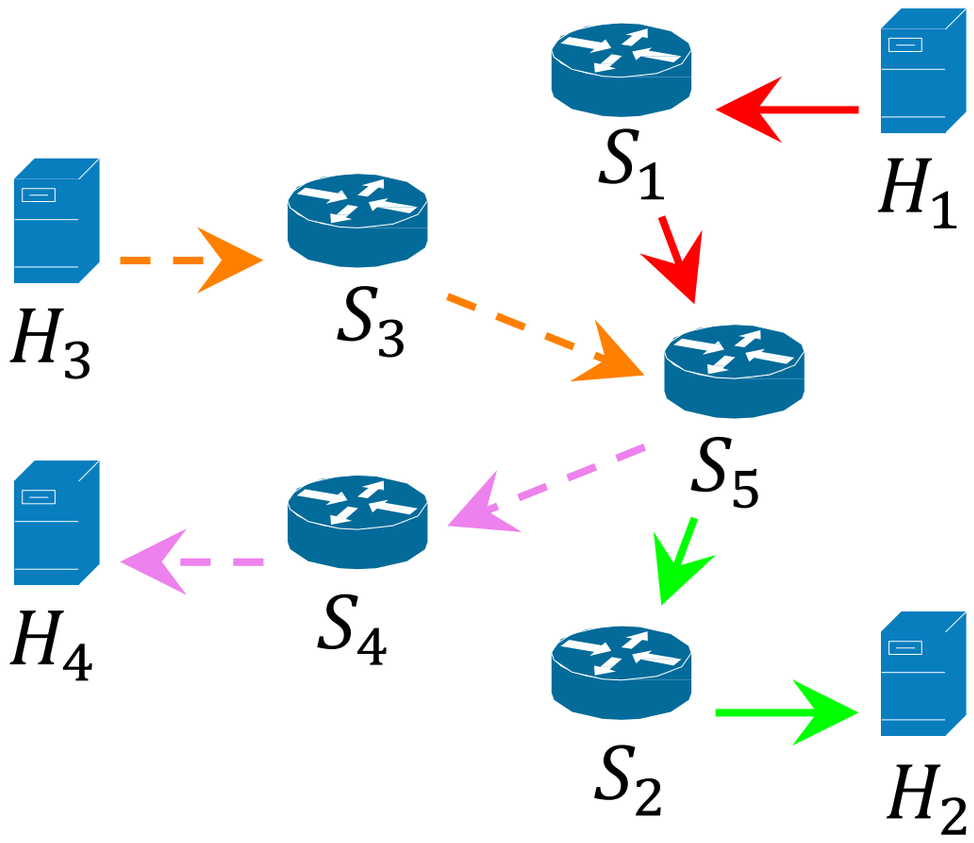
Concurrent network programs

Correctness when network programs execute **concurrently**?



Concurrent network programs

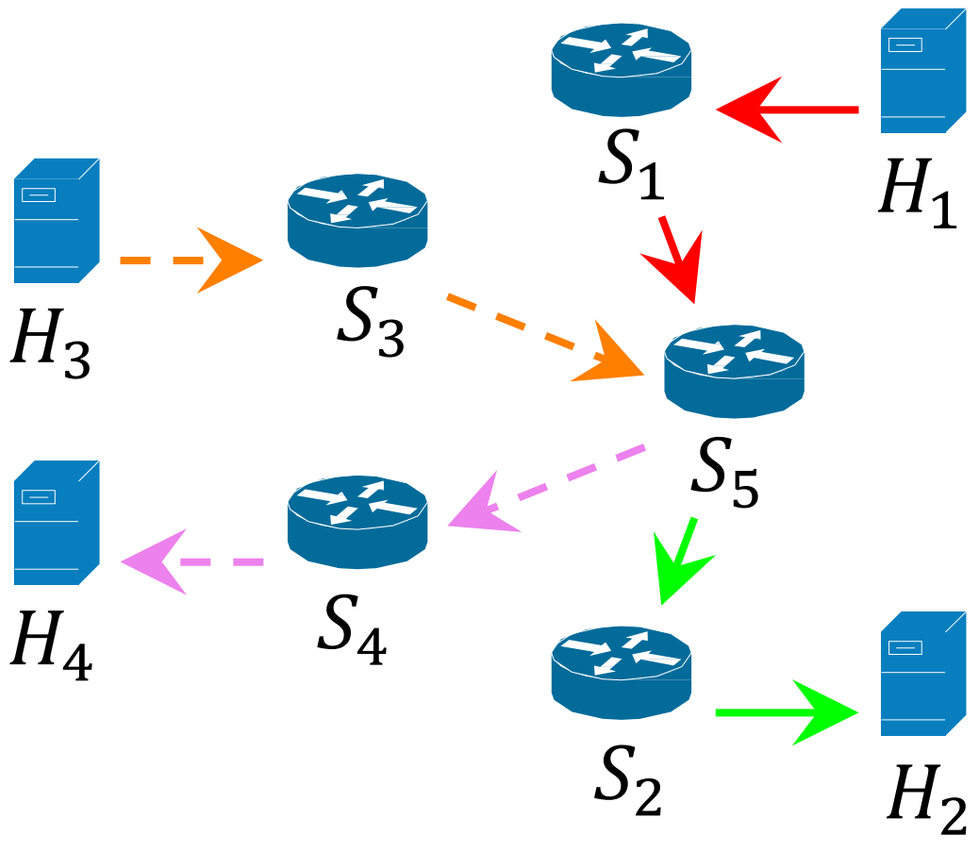
Correctness when network programs execute **concurrently**?



- Network operator wants to take down the $H_1 \rightarrow H_2$ forwarding rules, and install $H_3 \rightarrow H_4$ rules

Concurrent network programs

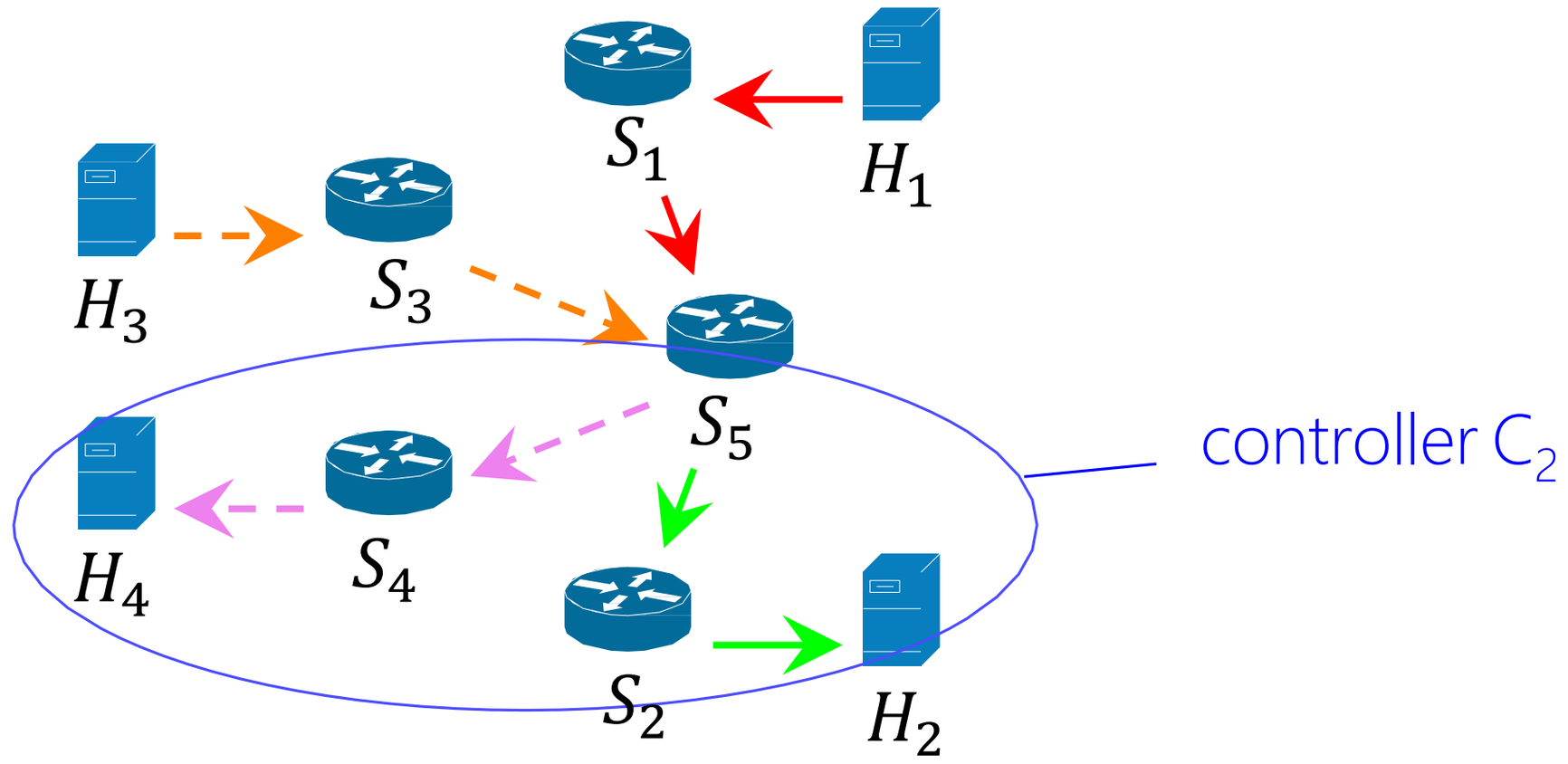
Correctness when network programs execute **concurrently**?



- Network operator wants to take down the $H_1 \rightarrow H_2$ forwarding rules, and install $H_3 \rightarrow H_4$ rules
- Example property: *isolation — all packets entering the network from* H_1 *must exit at* H_2.

Concurrent network programs

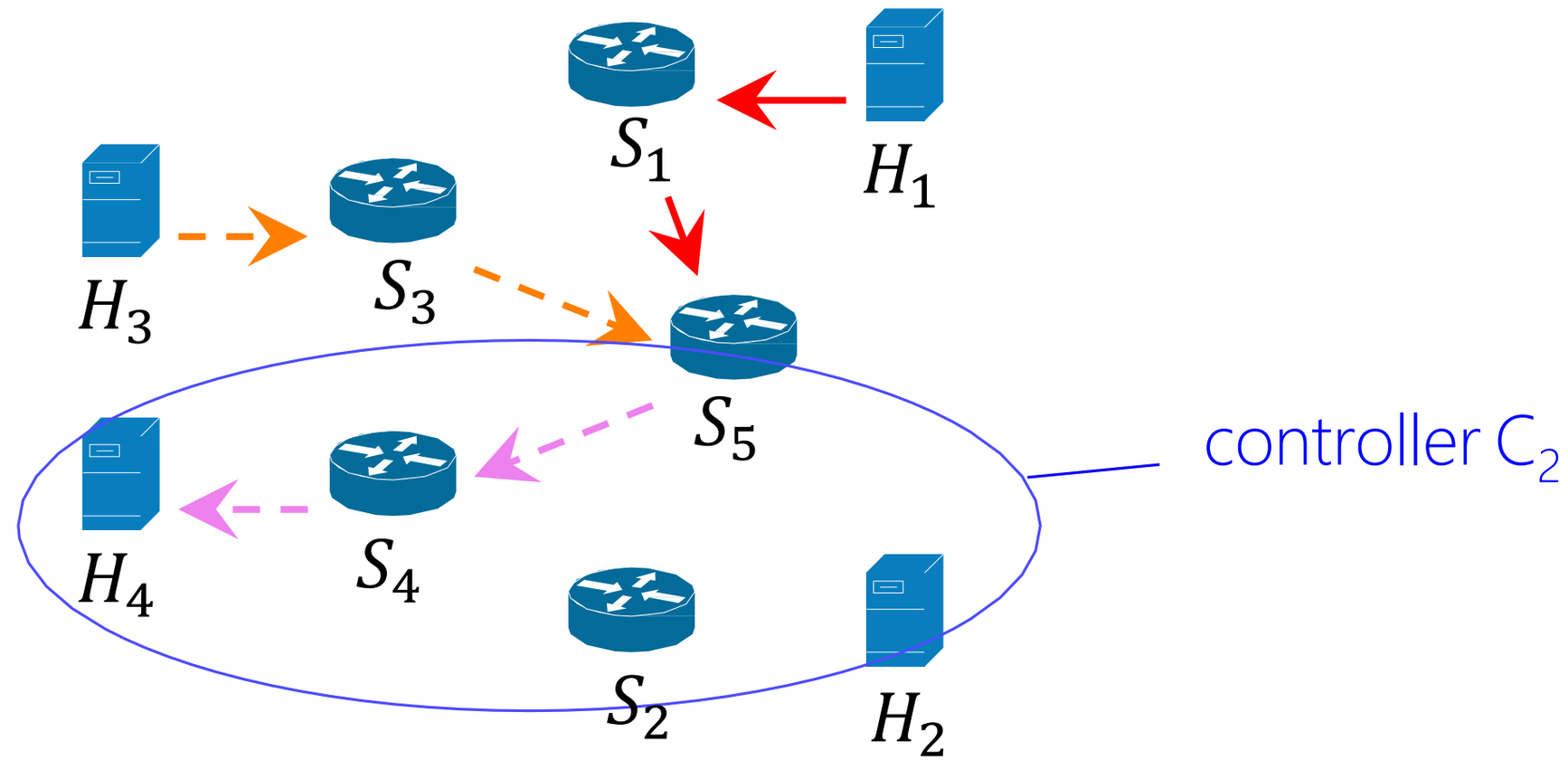
Correctness when network programs execute **concurrently**?



- Network operator wants to take down the $H_1 \rightarrow H_2$ forwarding rules, and install $H_3 \rightarrow H_4$ rules
- Example property: *isolation — all packets entering the network from* H_1 *must exit at* H_2.
- Potential bug: *controller race*

Concurrent network programs

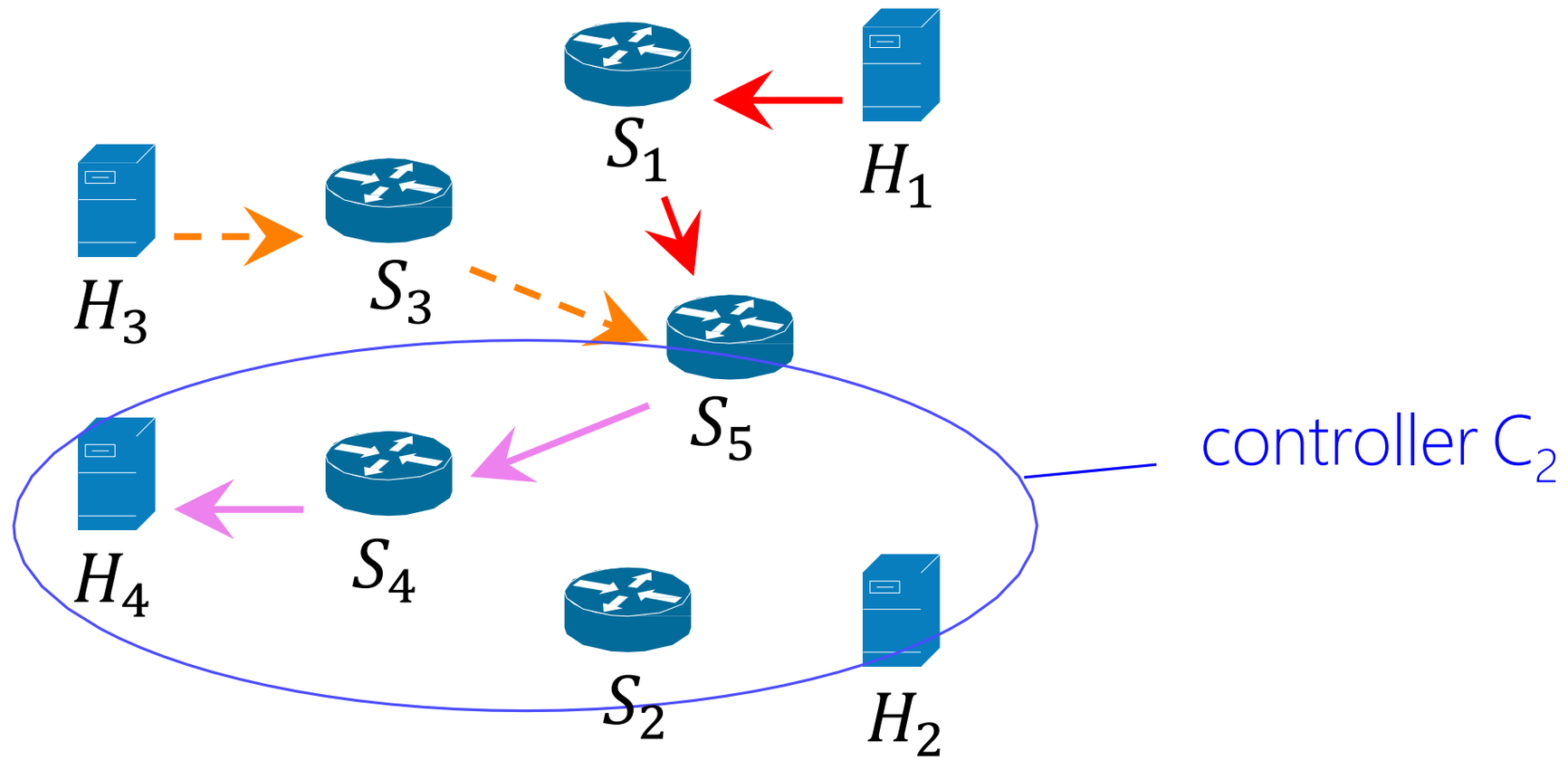
Correctness when network programs execute **concurrently**?



- Network operator wants to take down the $H_1 \rightarrow H_2$ forwarding rules, and install $H_3 \rightarrow H_4$ rules
- Example property: *isolation — all packets entering the network from* H_1 *must exit at* H_2.
- Potential bug: *controller race*

Concurrent network programs

Correctness when network programs execute **concurrently**?

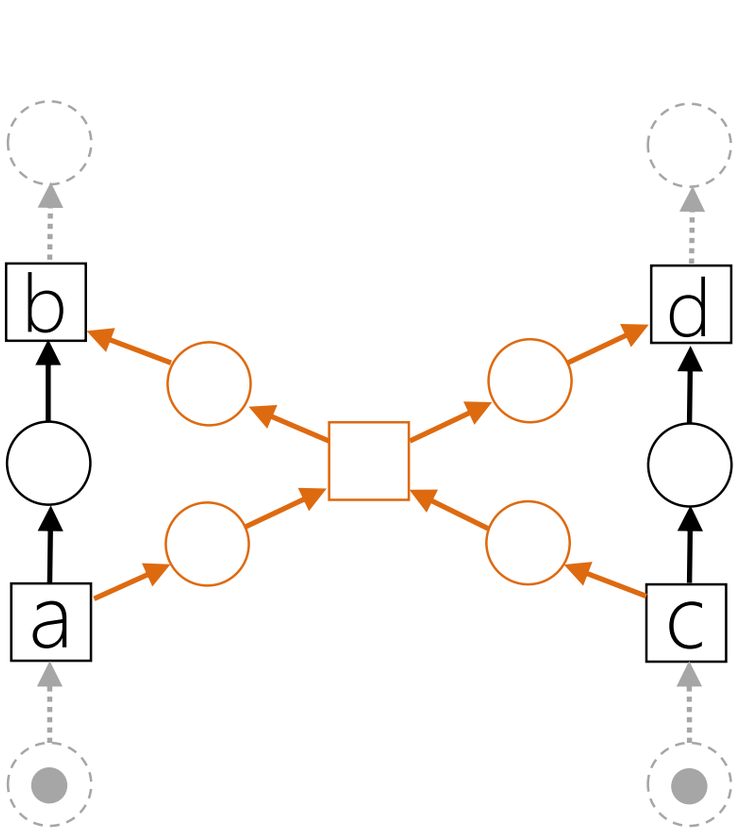


- Network operator wants to take down the $H_1 \rightarrow H_2$ forwarding rules, and install $H_3 \rightarrow H_4$ rules
- Example property: *isolation — all packets entering the network from* H_1 *must exit at* H_2.
- Potential bug: *controller race*

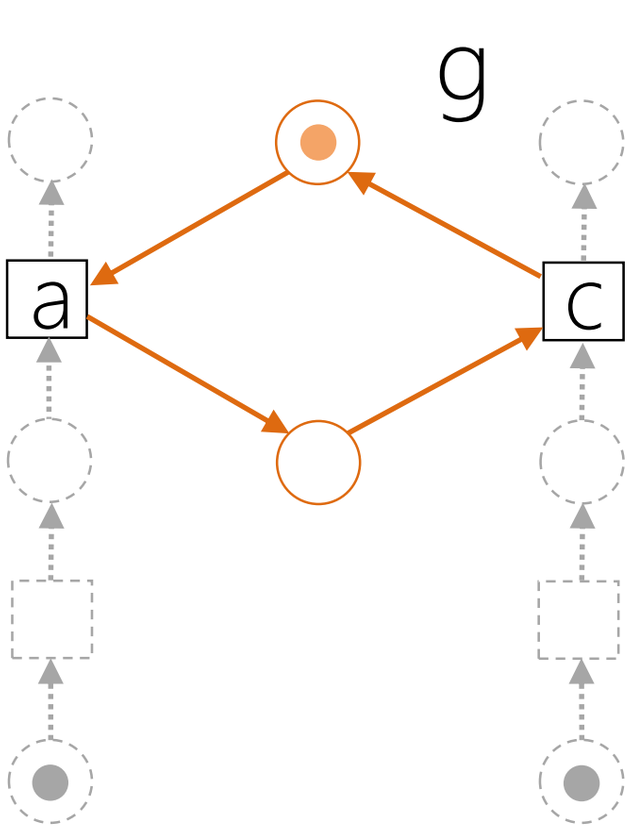
Adding synchronization

- How can we model *synchronization constructs*?
- Synchronization skeletons:

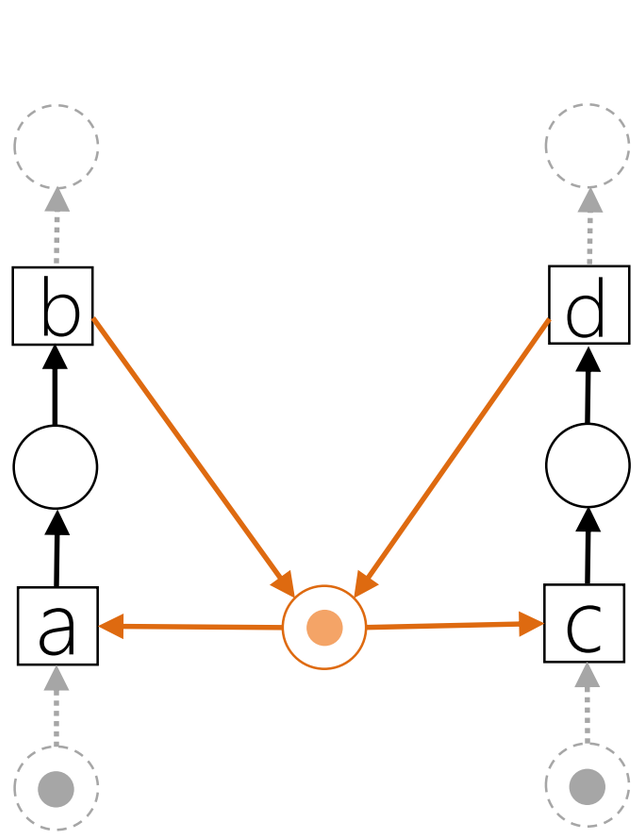
barrier



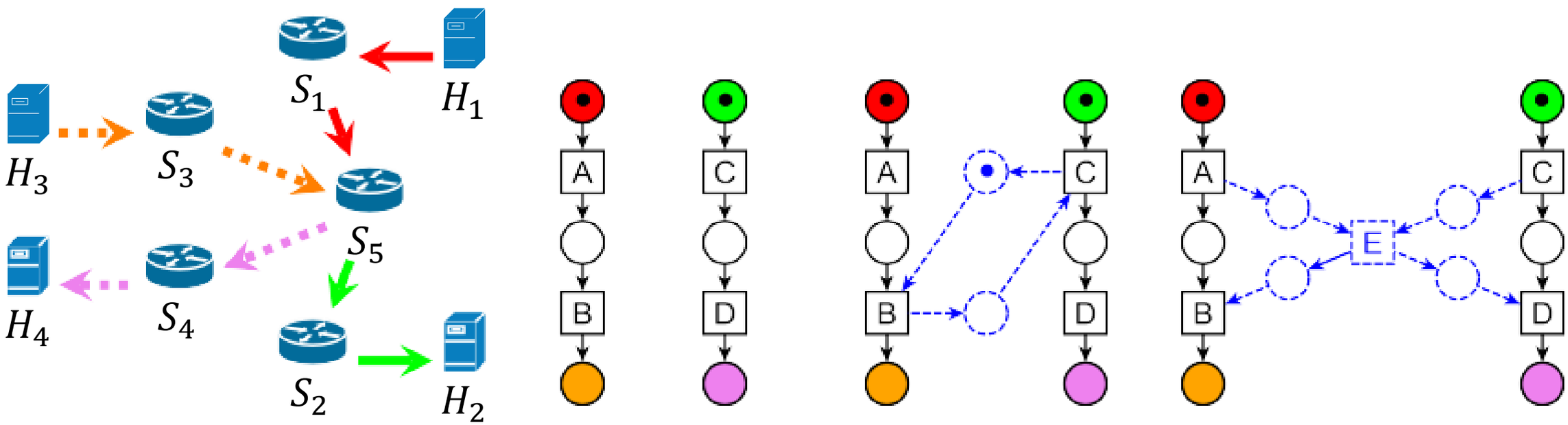
orderin g



mutex



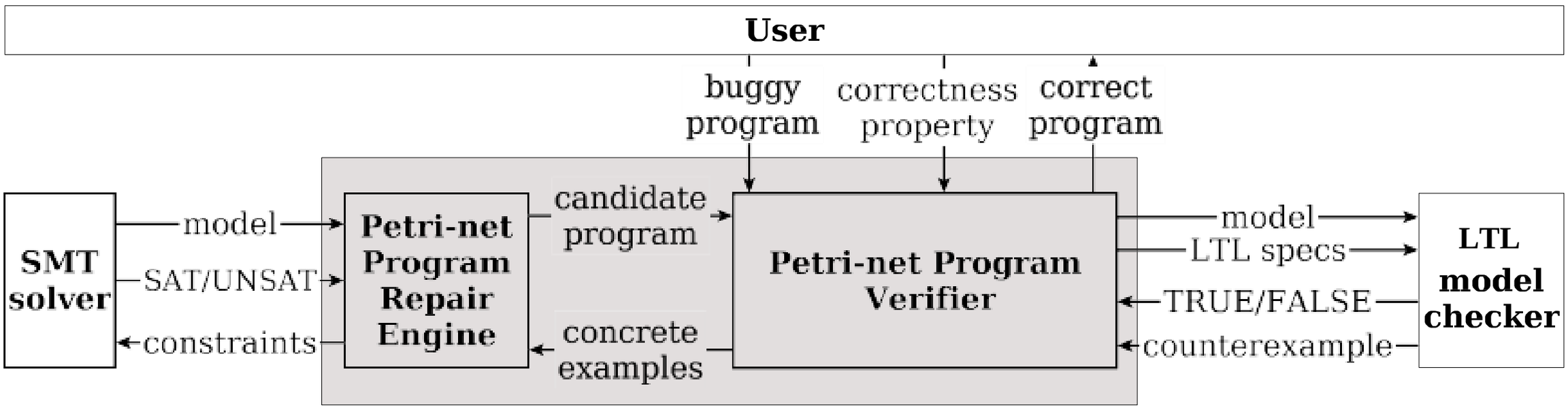
Synthesizing petri-net programs



- φ_1: *no packet originating at* $\boldsymbol{H_1}$ *should arrive at* $\boldsymbol{H_4}$
- φ_2: *no packet originating at* $\boldsymbol{H_3}$ *should arrive at* $\boldsymbol{H_2}$

- First counterexample: $[\boldsymbol{C}, \boldsymbol{D}]$, because { } violates the spec
- Second counterexample: $[\boldsymbol{A}, \boldsymbol{B}]$, because { } violates the spec

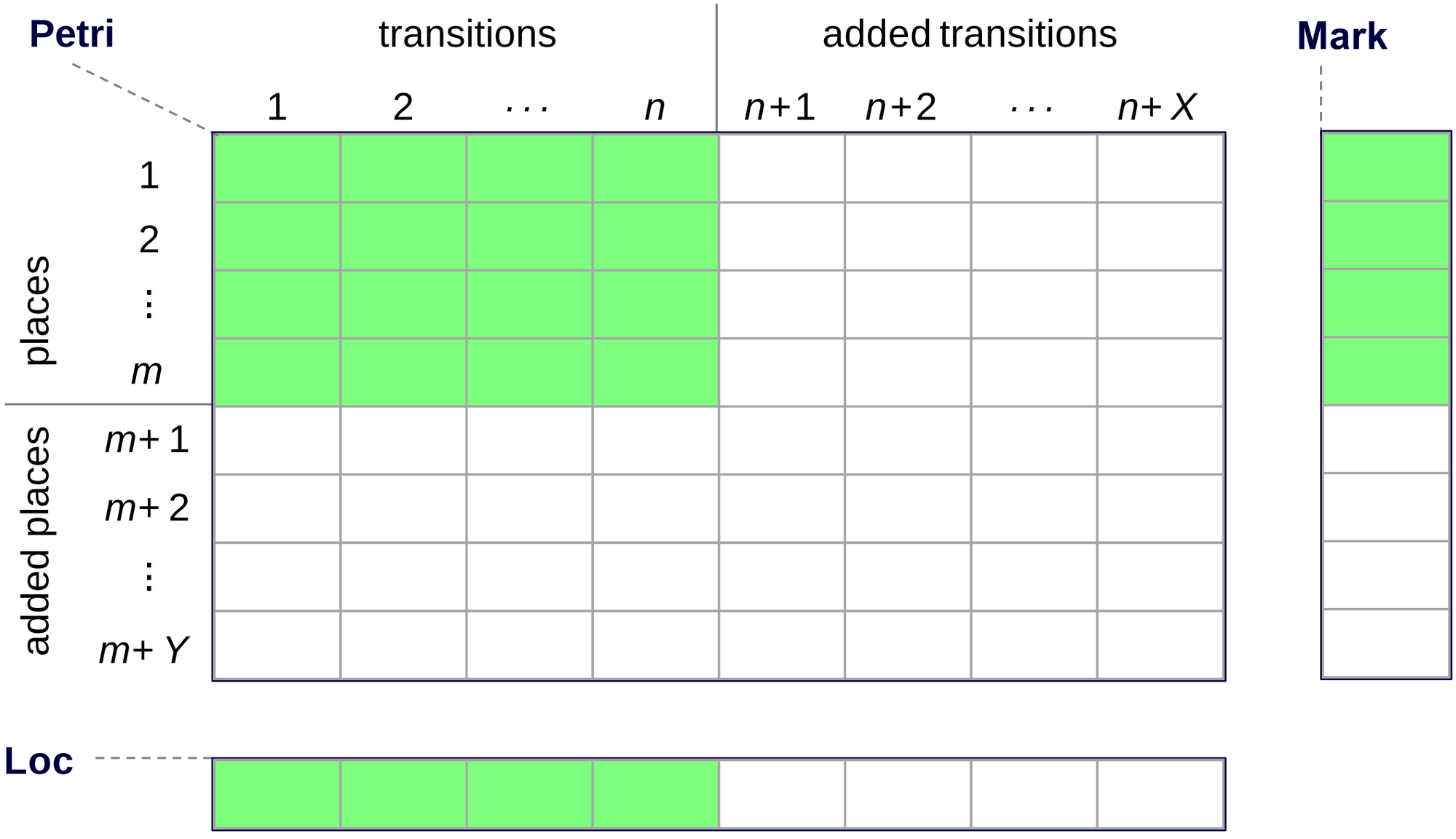
Synthesizer Architecture



- LTL model checker (SPIN) returns *trace* (sequence of events) which leads to a network configuration in which the property is violated
    - (also checks 1-safety)
- Synthesizer (Z3) produces *Petri-net program* containing none of the buggy traces so far

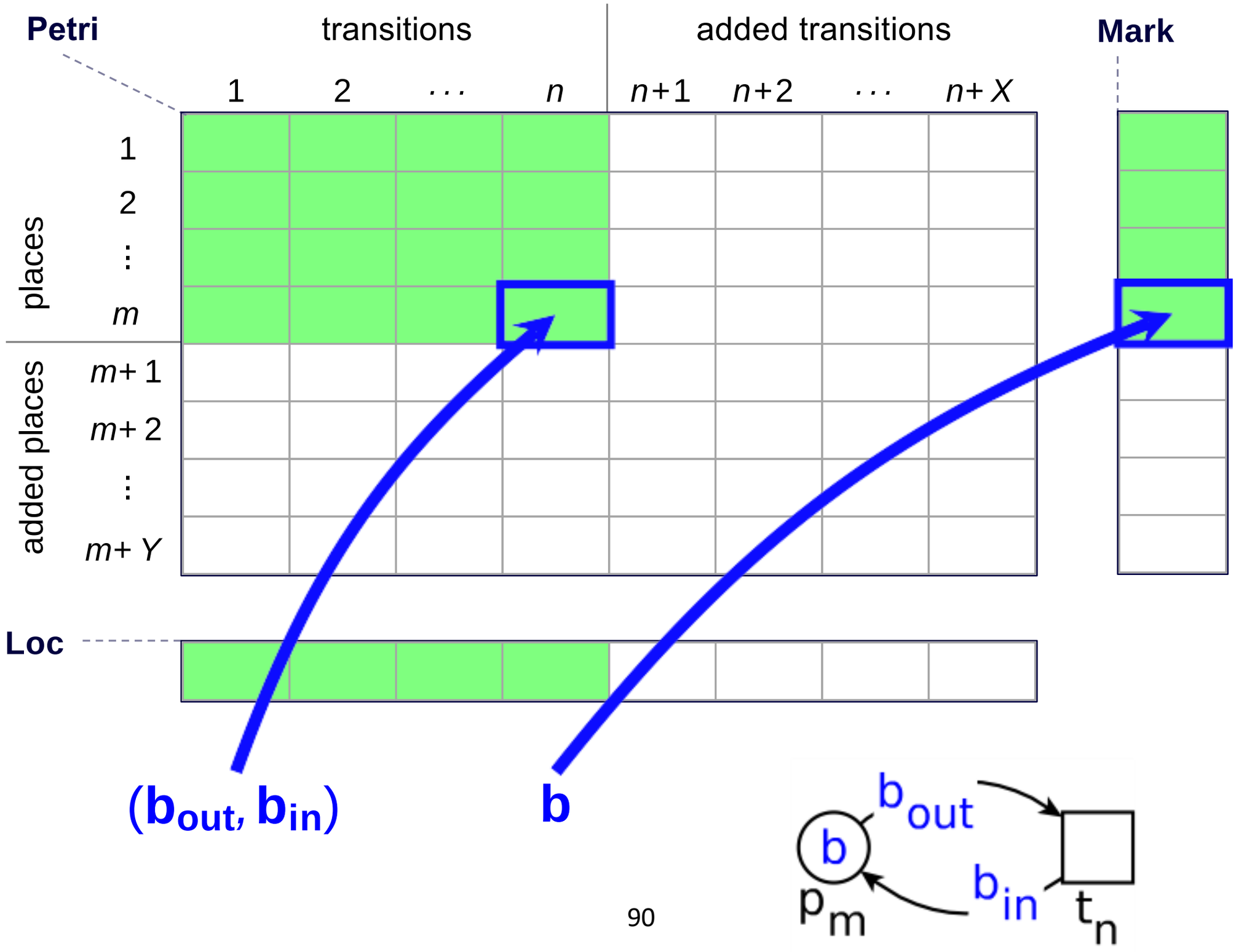
Petri-net Synthesis Engine

SMT encoding for Petri-net programs:



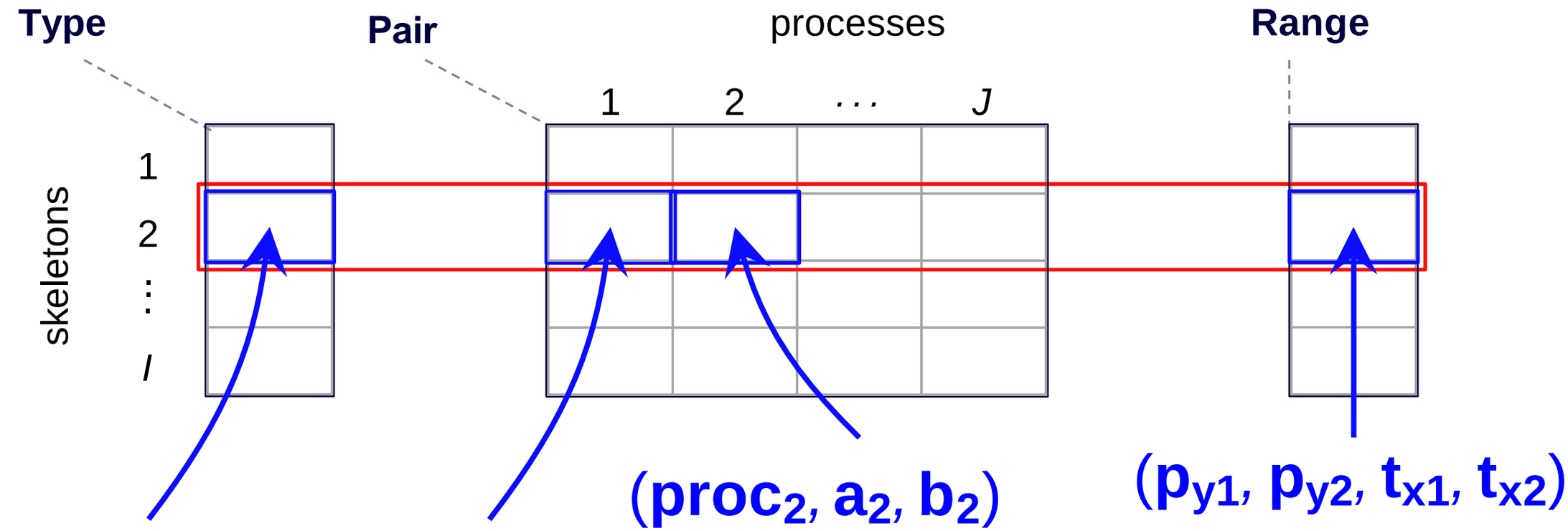
Petri-net Synthesis Engine

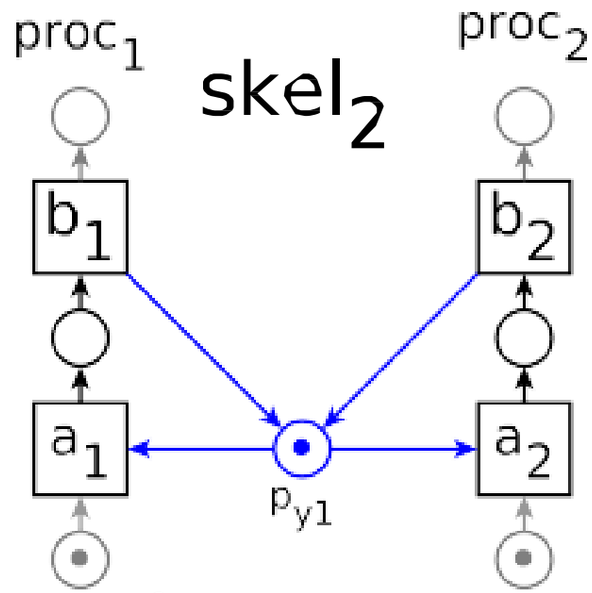
SMT encoding for Petri-net programs:



Petri-net Synthesis Engine

SMT encoding for synchronization skeletons:





We add constraints based on the finite set of counterexample traces

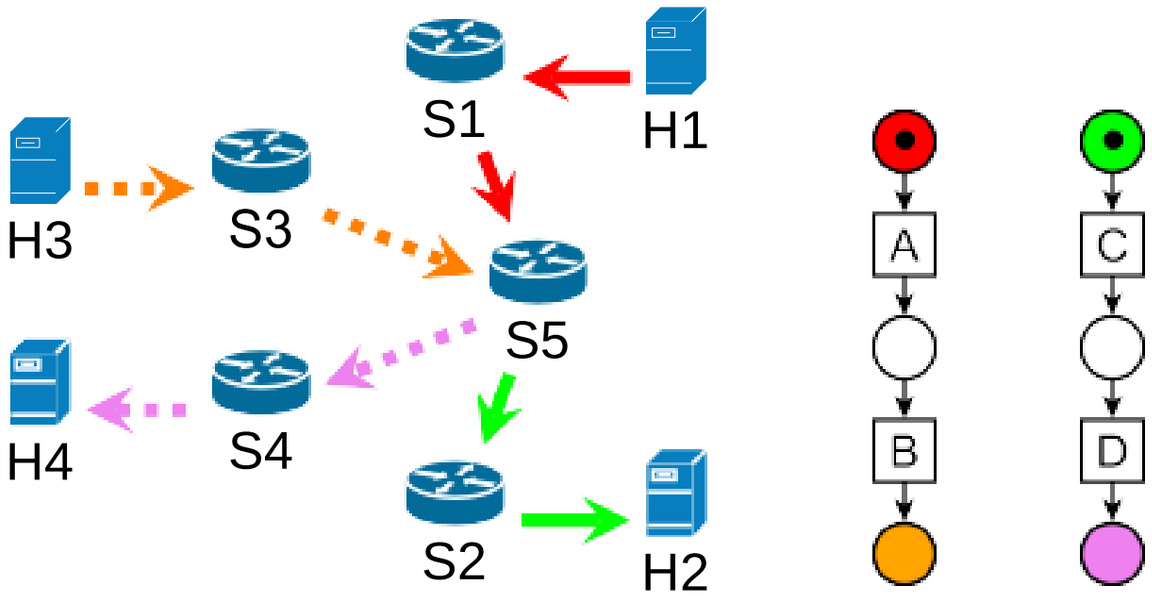
Experimental results - expressiveness

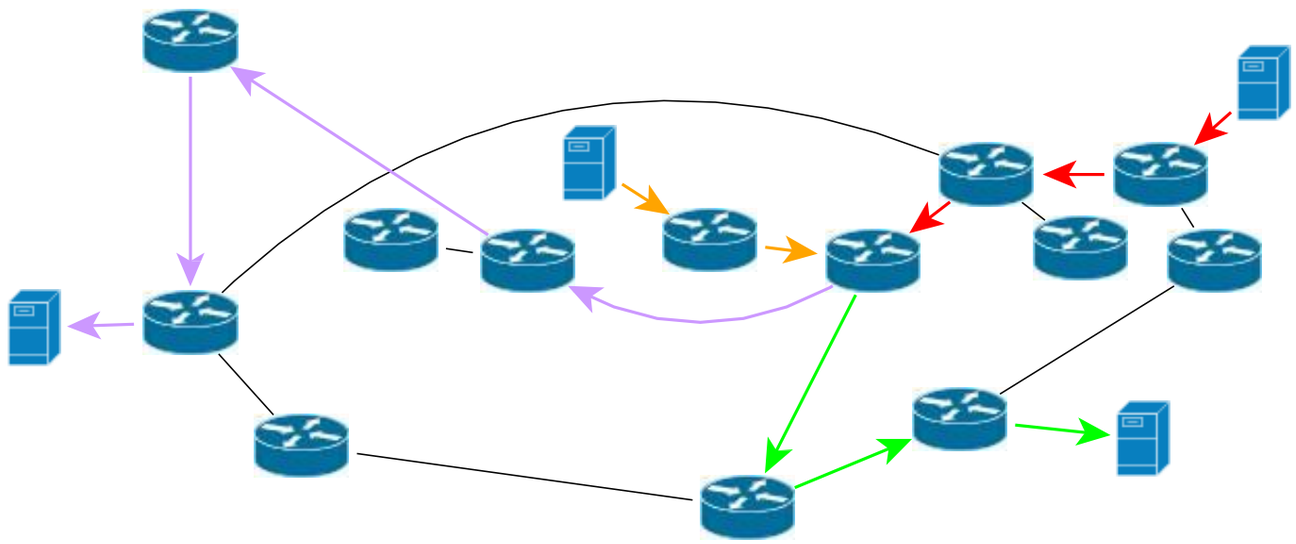
Conflicting Controller modules:

- Discovery vs Forwarding Modules, POX controller
  [El Hassany et al]
- Discovery vs Forwarding Modules, NOX controller
  [Scott et al]
- HTTP traffic monitoring vs Waypoint Enforcement
  [Canini et al]
- Update vs Update
  [Peresini et al]

Experimental results - scalability

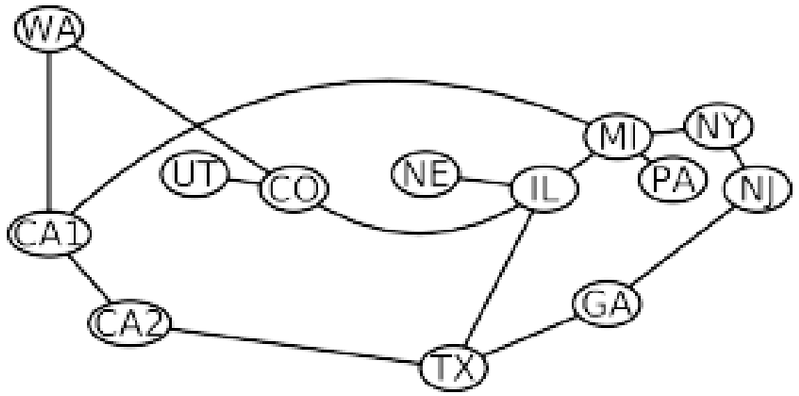
We scaled up the topology on the previously-discussed **Isolation** example

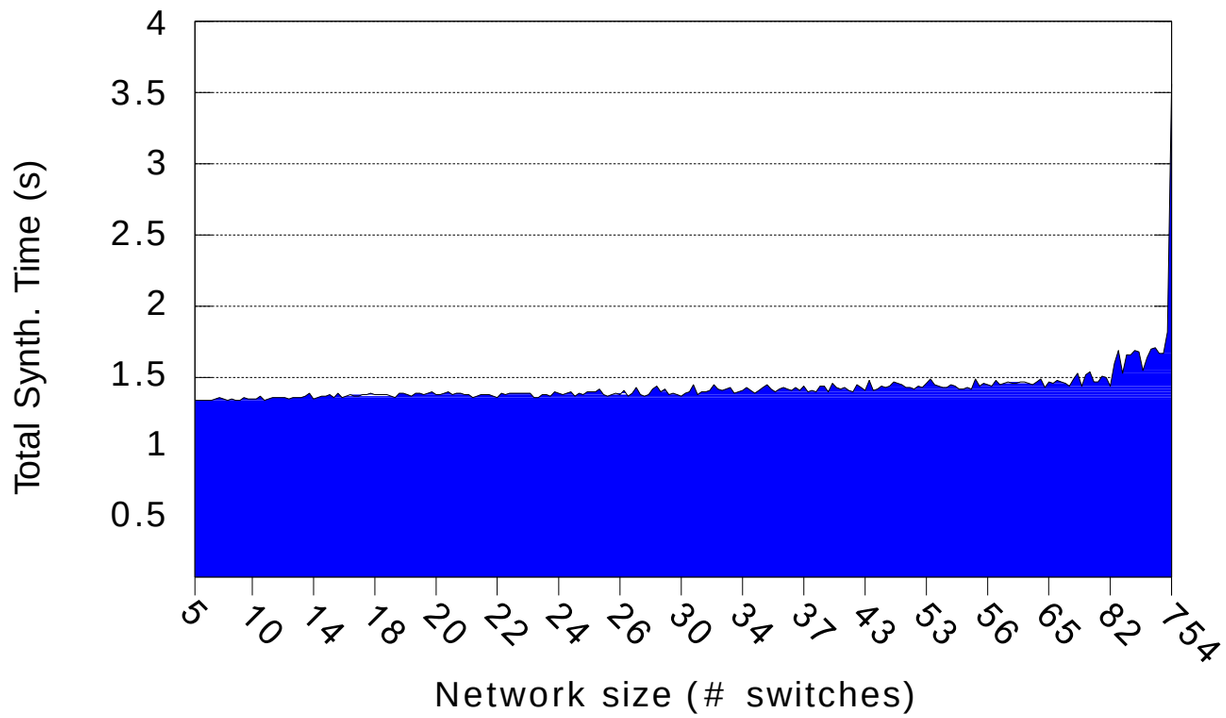




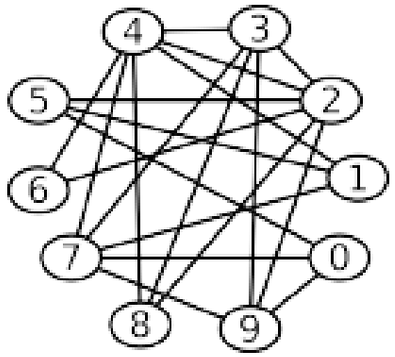
We then measured total synthesizer runtime versus topology size

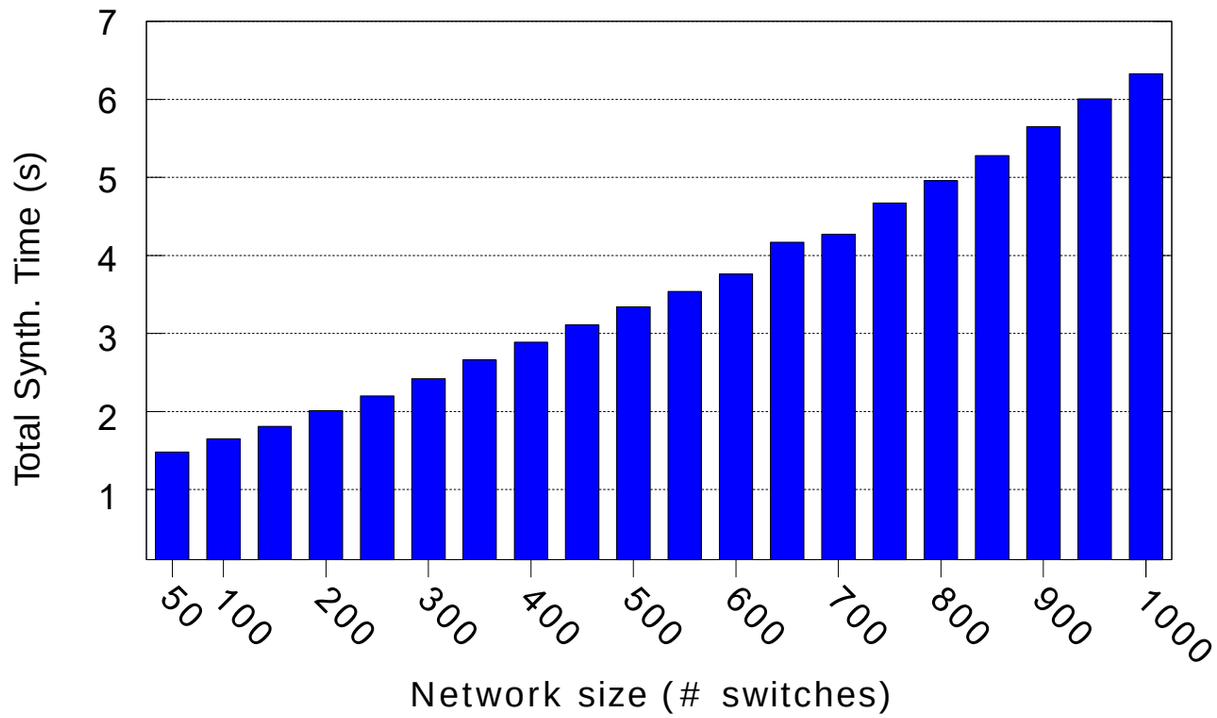
Experimental results – Topology Zoo



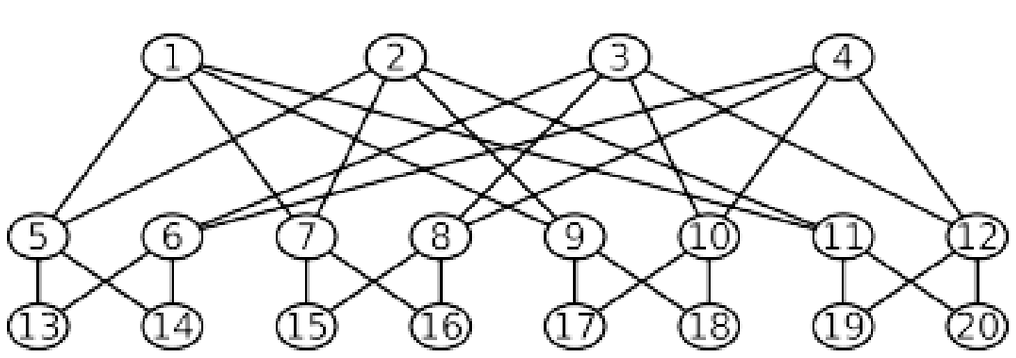


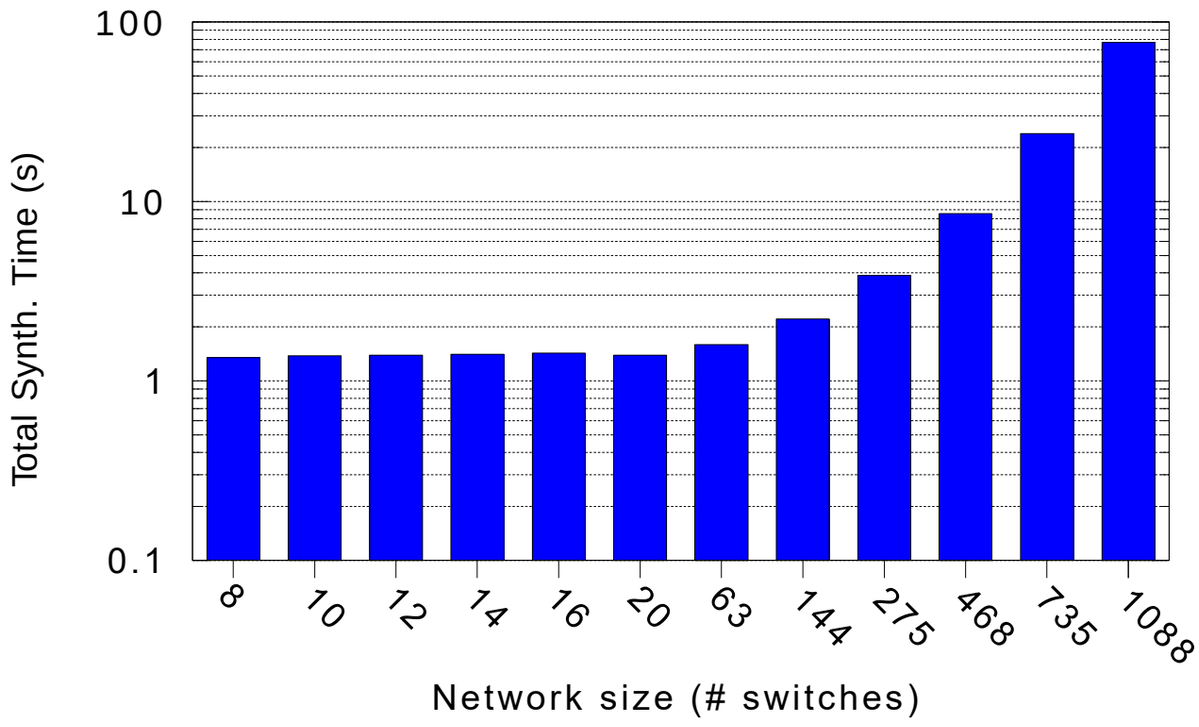
Experimental results – Small World





Experimental results – FatTree





Optimizing Horn Solvers for Network Repair

[FMCAD ‘16,18]

Software Synthesis

Buggy Partial Program

Logical Formula

Input-Output Examples

Software Repair

Synthesizer

Program

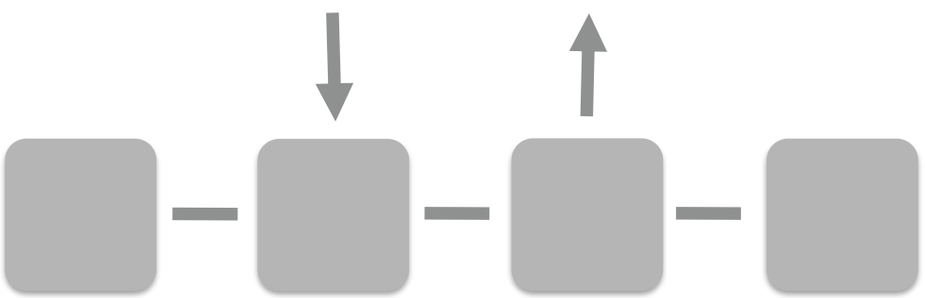
SDN with Buggy Cofiguration

Application

Network-wide Configuration

Controller Platform

Controller Run-Time

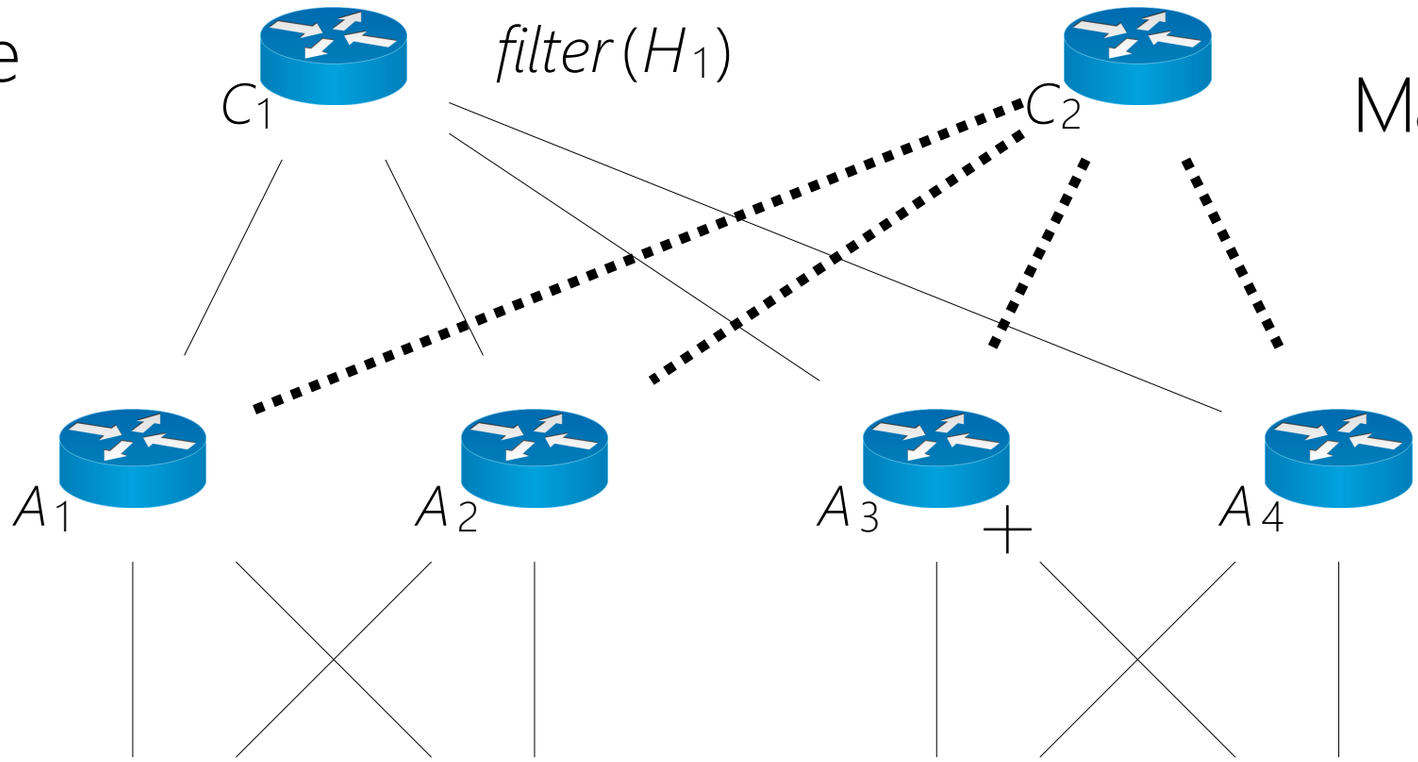


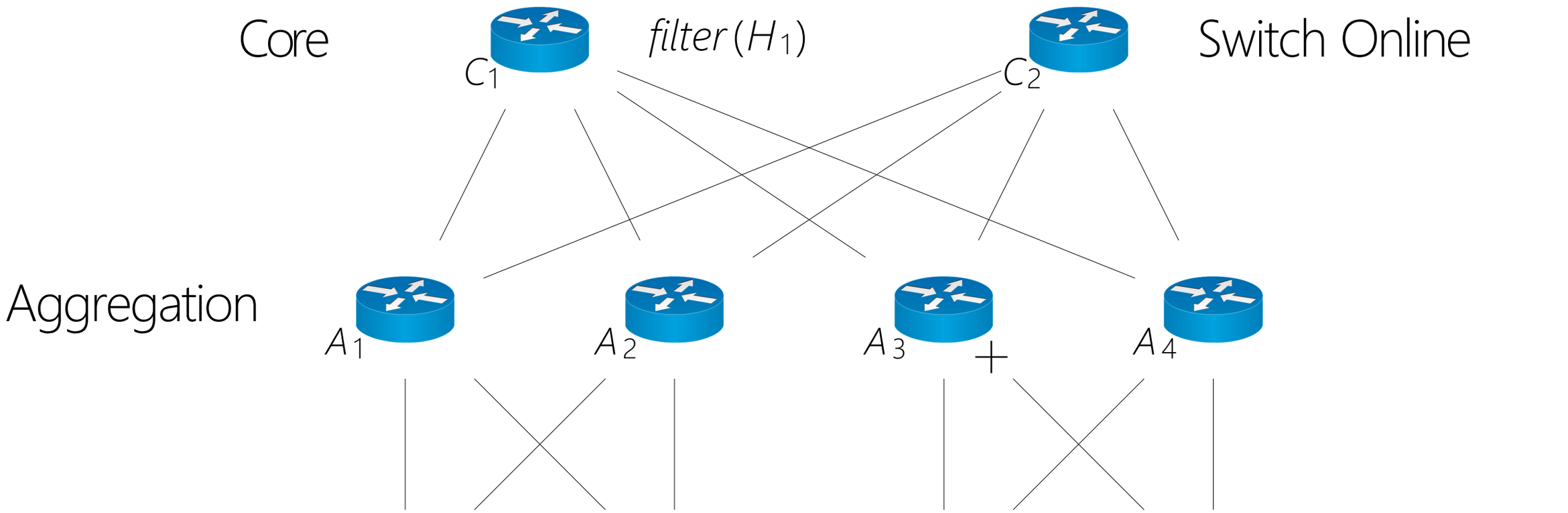
Core

Aggregation

ToR

Host



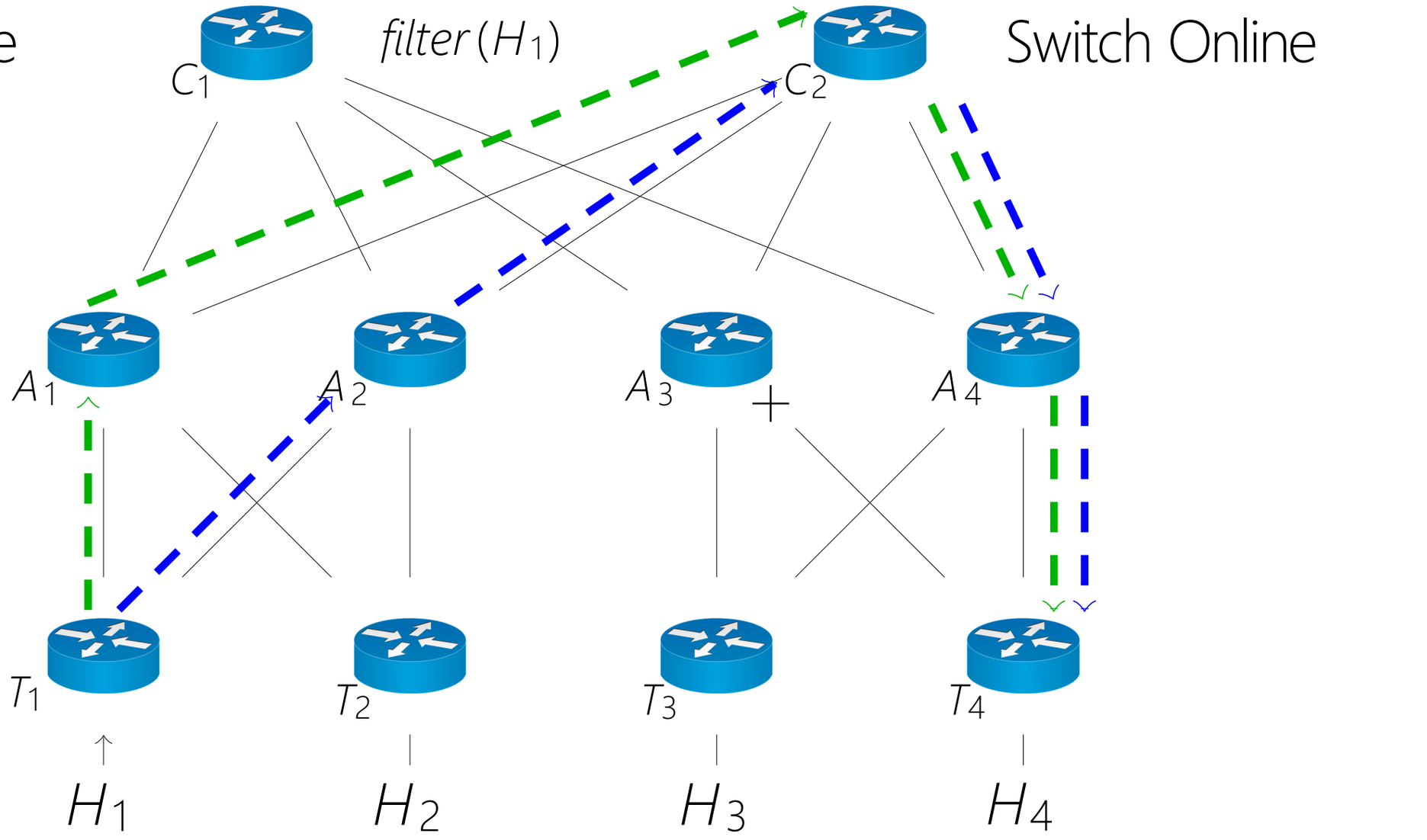

Core
C_1
$filter(H_1)$
C_2
Switch Online
Aggregation
A_1
A_2
A_3
A_4
ToR
T_1
T_2
T_3
T_4
Host
H_1
H_2
H_3
H_4
not safe for H_1 traffic

Core

Aggregation

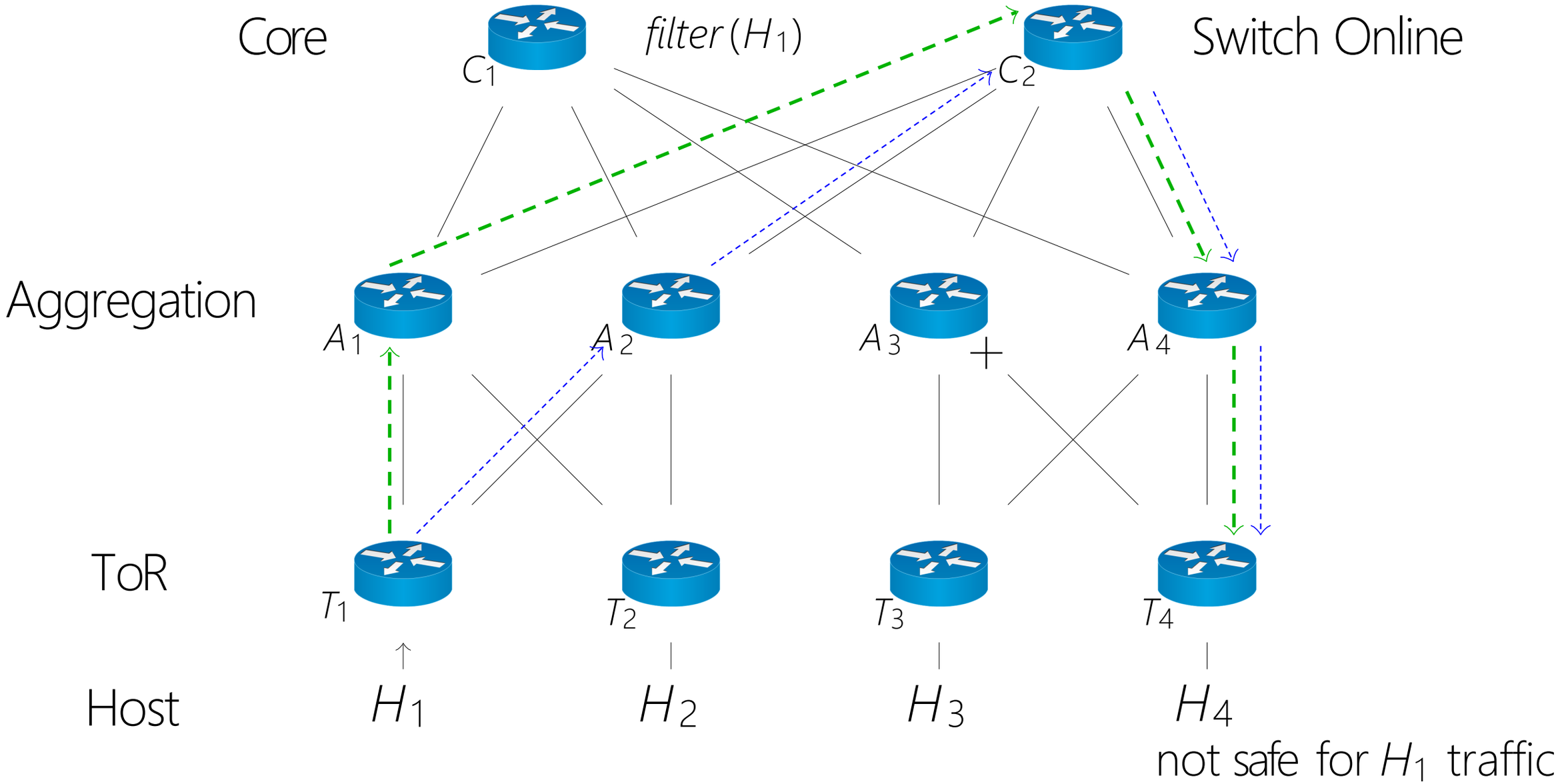
ToR

Host

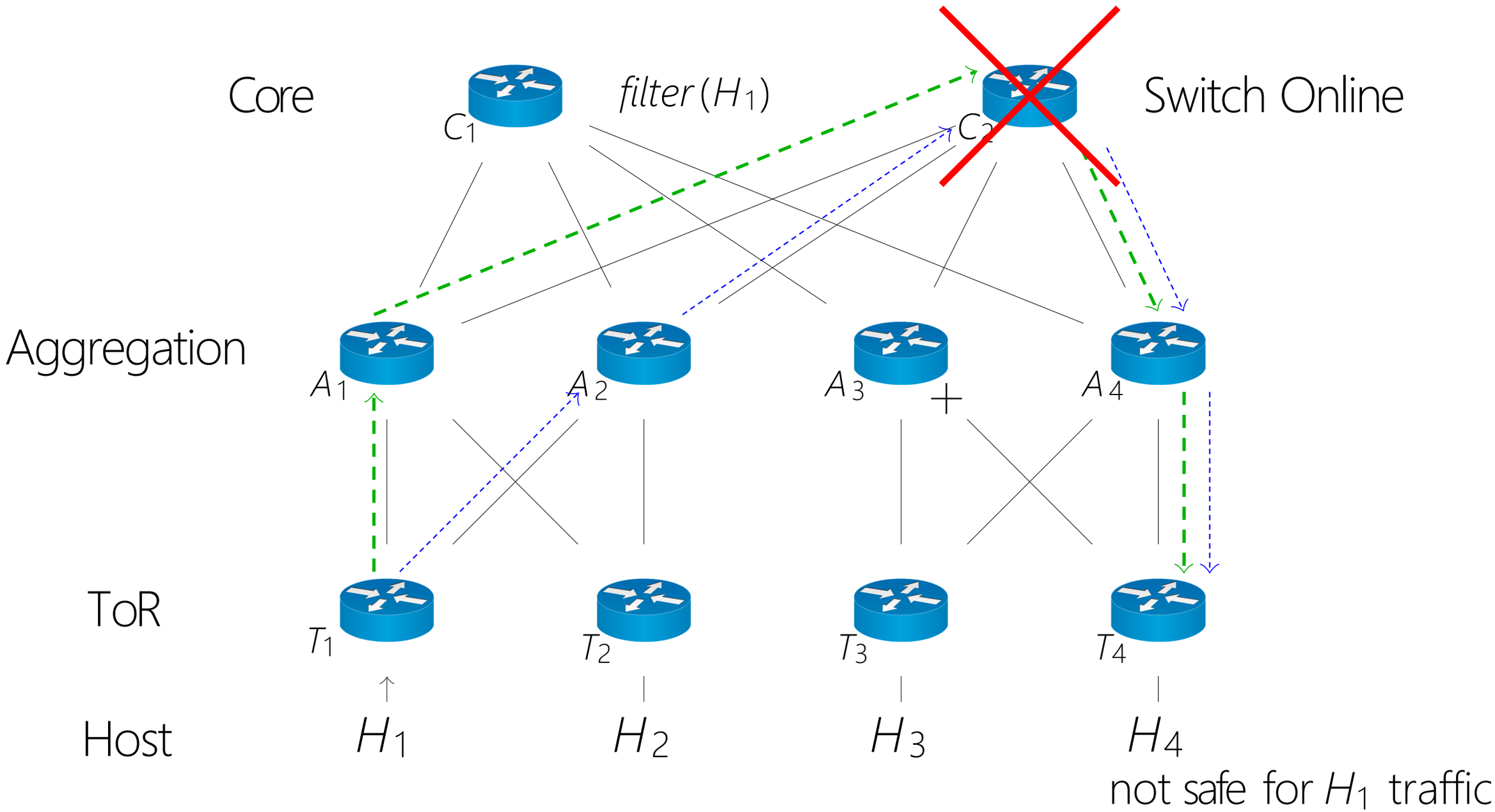


Switch Online

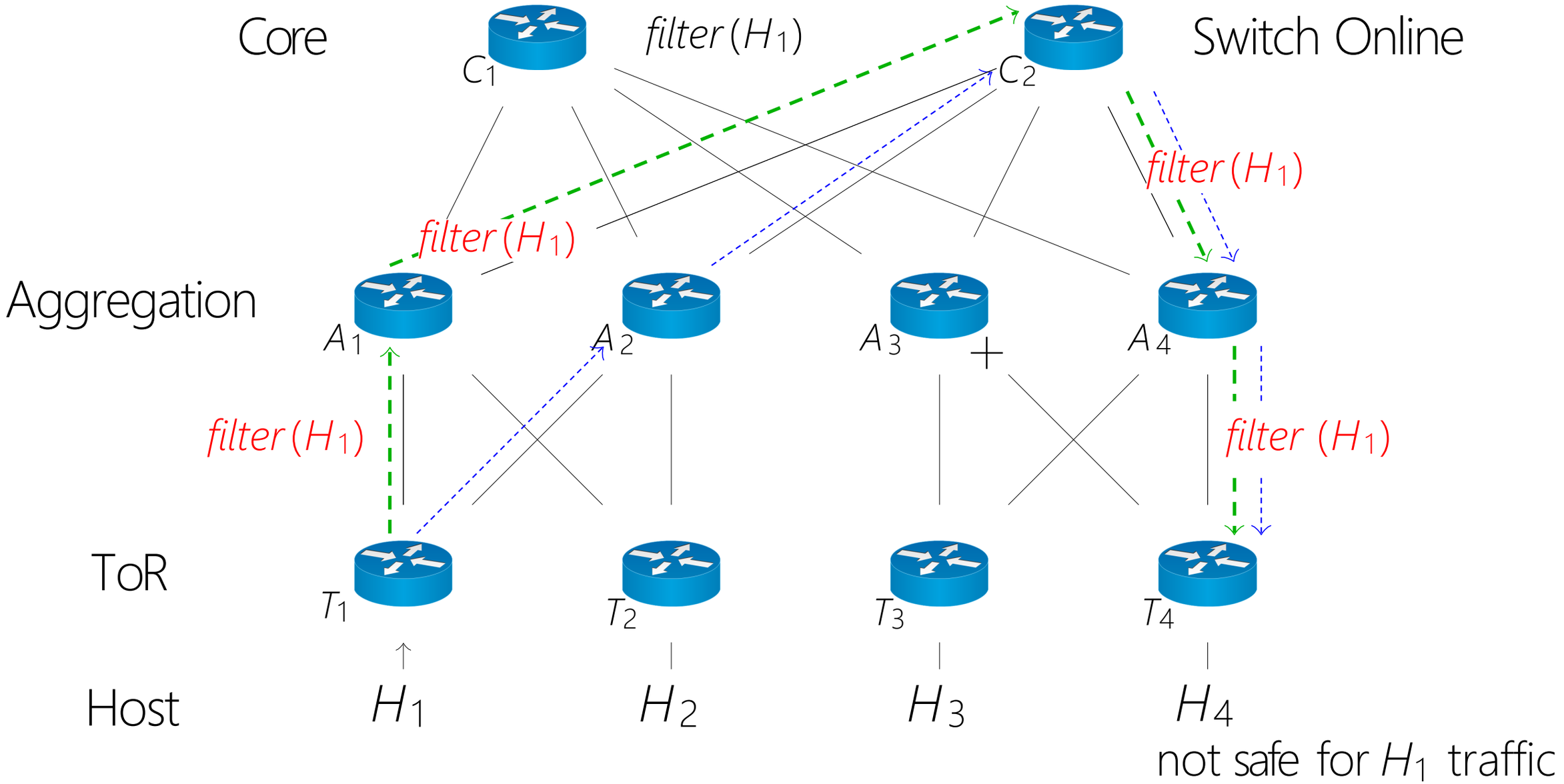
not safe for H_1 traffic



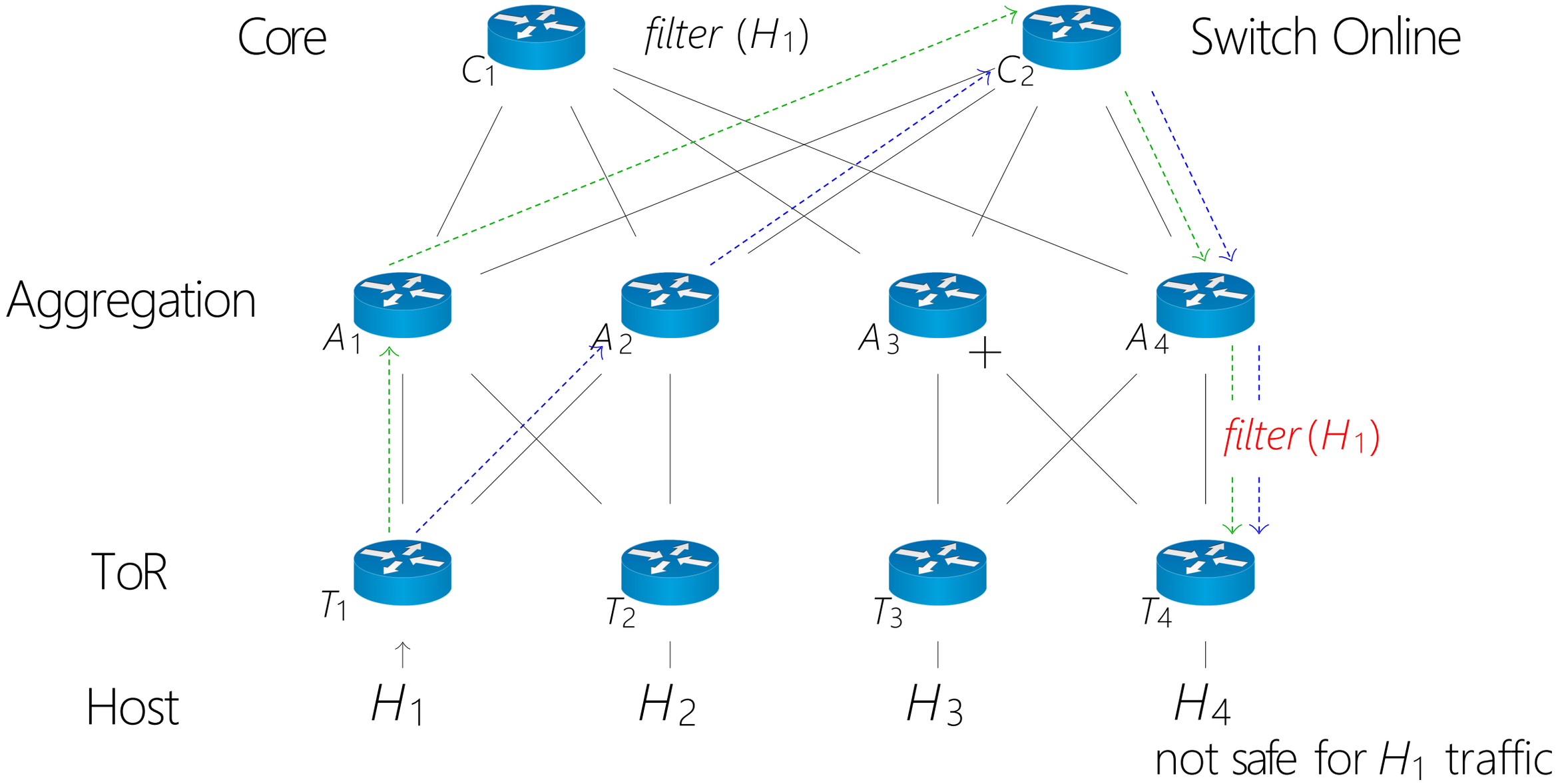
- How can we return back to safety by adding filters on links?
- There are several possible repair solutions
- Interested in **best** solutions;



- How can we return back to safety by adding filters on links?
- There are several possible repair solutions
- Interested in **best** solutions;



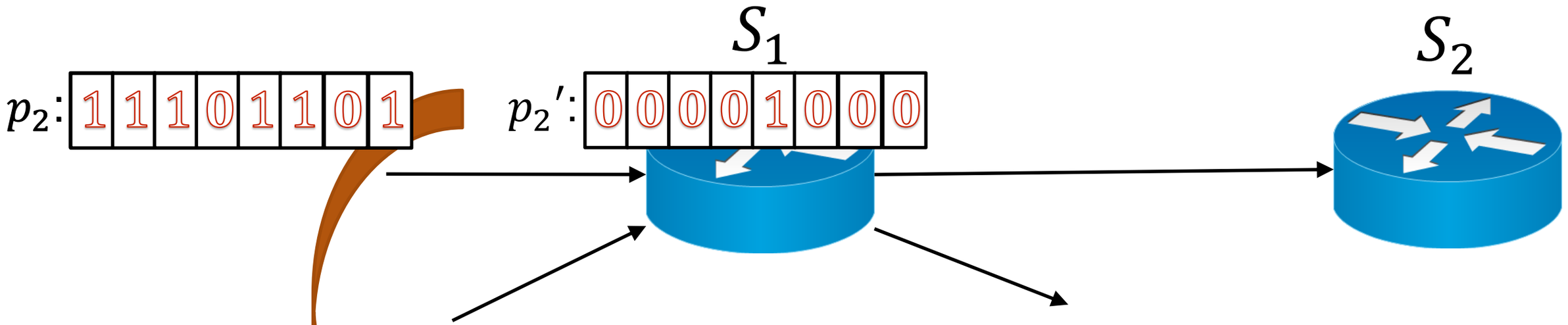
- How can we return back to safety by adding filters on links?
- There are several possible repair solutions
- Interested in **best** solutions;



- How can we return back to safety by adding filters on links?
- There are several possible repair solutions
- Interested in **best** solutions;

Contributions

- Translation of network and its correctness conditions to logic (Horn clauses)
- Repair unsatisfiable Horn clauses
  - (i.e. buggy system violating correctness)
- New lattice-based optimization procedure for Horn clause repair

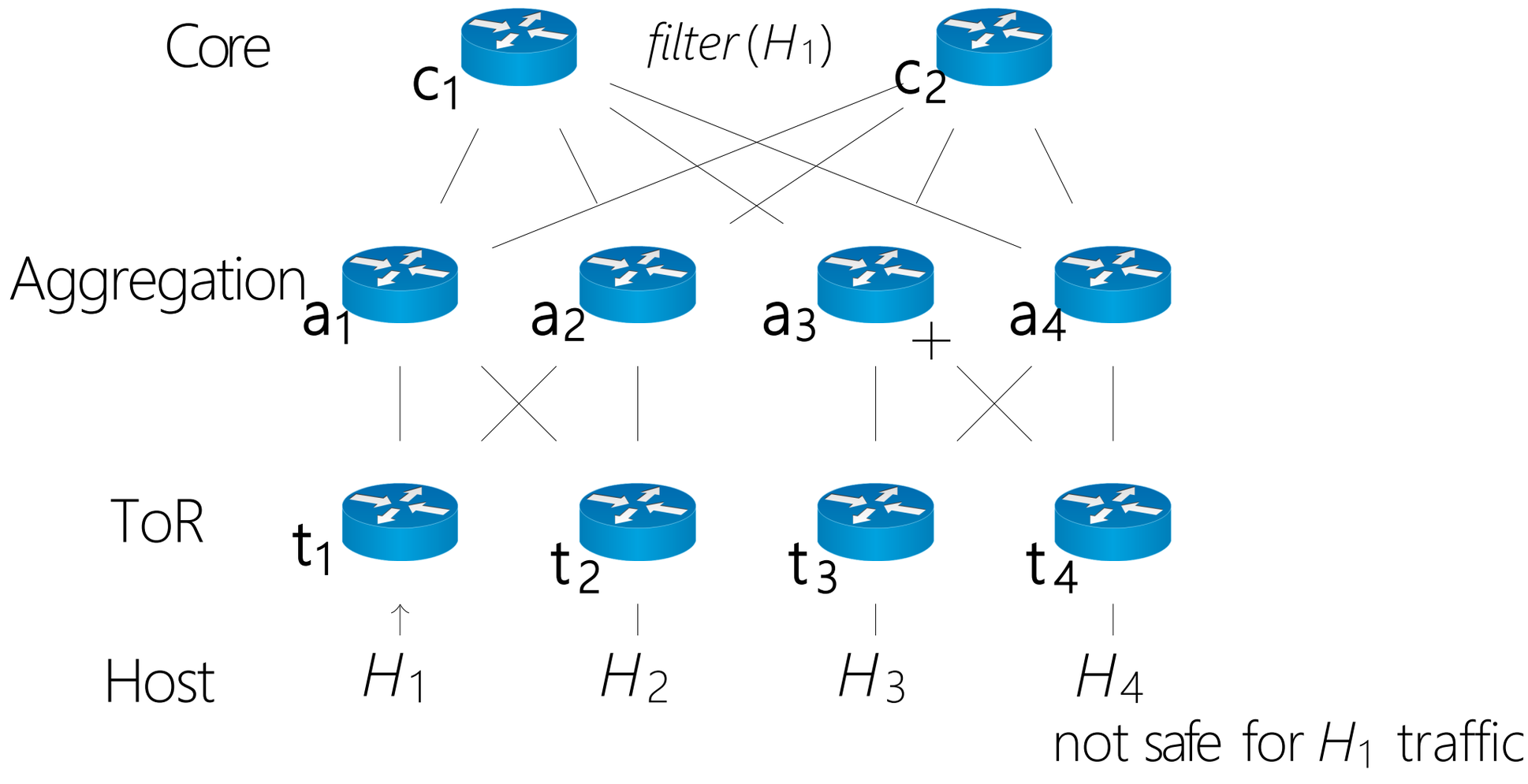


Forwarding Table

pattern	action
p_1	fwd S_2
p_2	rewrite p_2'; fwd S_2
p_3	drop
⋮	⋮

- Assume $S_i(p)$means packet p is at switch S_i
  - $S_1(p) \wedge (p = p_1) \rightarrow S_2(p)$
  - $S_1(p) \wedge (p = p_2) \rightarrow S_2(p_2')$
  - $S_1(p) \wedge (p = p_3) \rightarrow D(p)$
- These formulae are called Horn clauses

Horn Clauses for Network



Ingress. H_1 sends out the special traffic type 0

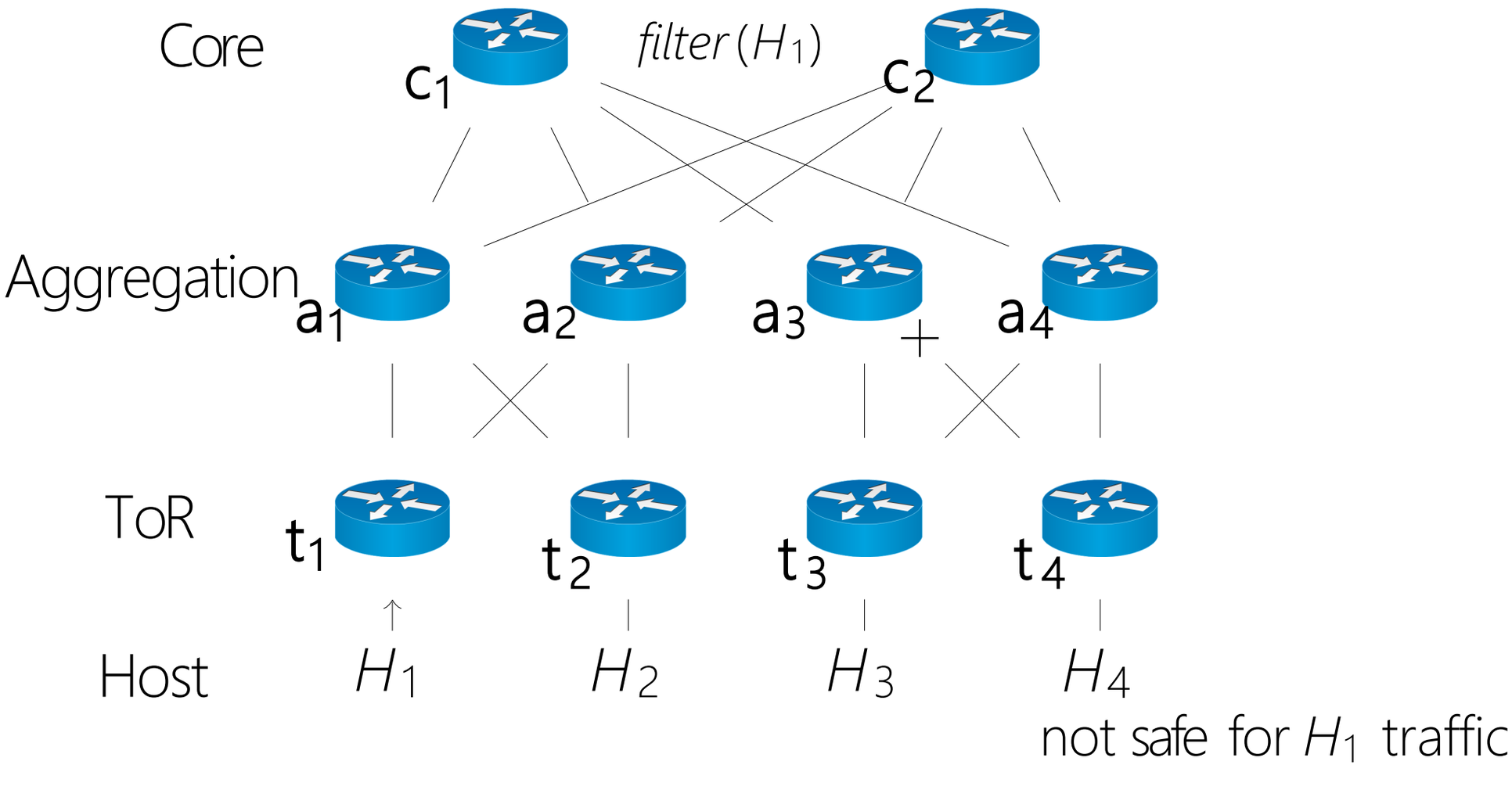
$$(typ = 0 \wedge dst \in \{2, 3, 4\}) \rightarrow \mathbf{t_1}(dst, typ)$$

$$(typ > 0 \wedge typ < 8 \wedge dst \in \{1, 3, 4\}) \rightarrow \mathbf{t_2}(dst, typ)$$

$$(typ > 0 \wedge typ < 8 \wedge dst \in \{1, 2, 4\}) \rightarrow \mathbf{t_3}(dst, typ)$$

$$(typ > 0 \wedge typ < 8 \wedge dst \in \{1, 2, 3\}) \rightarrow \mathbf{t_4}(dst, typ)$$

Horn Clauses for Network



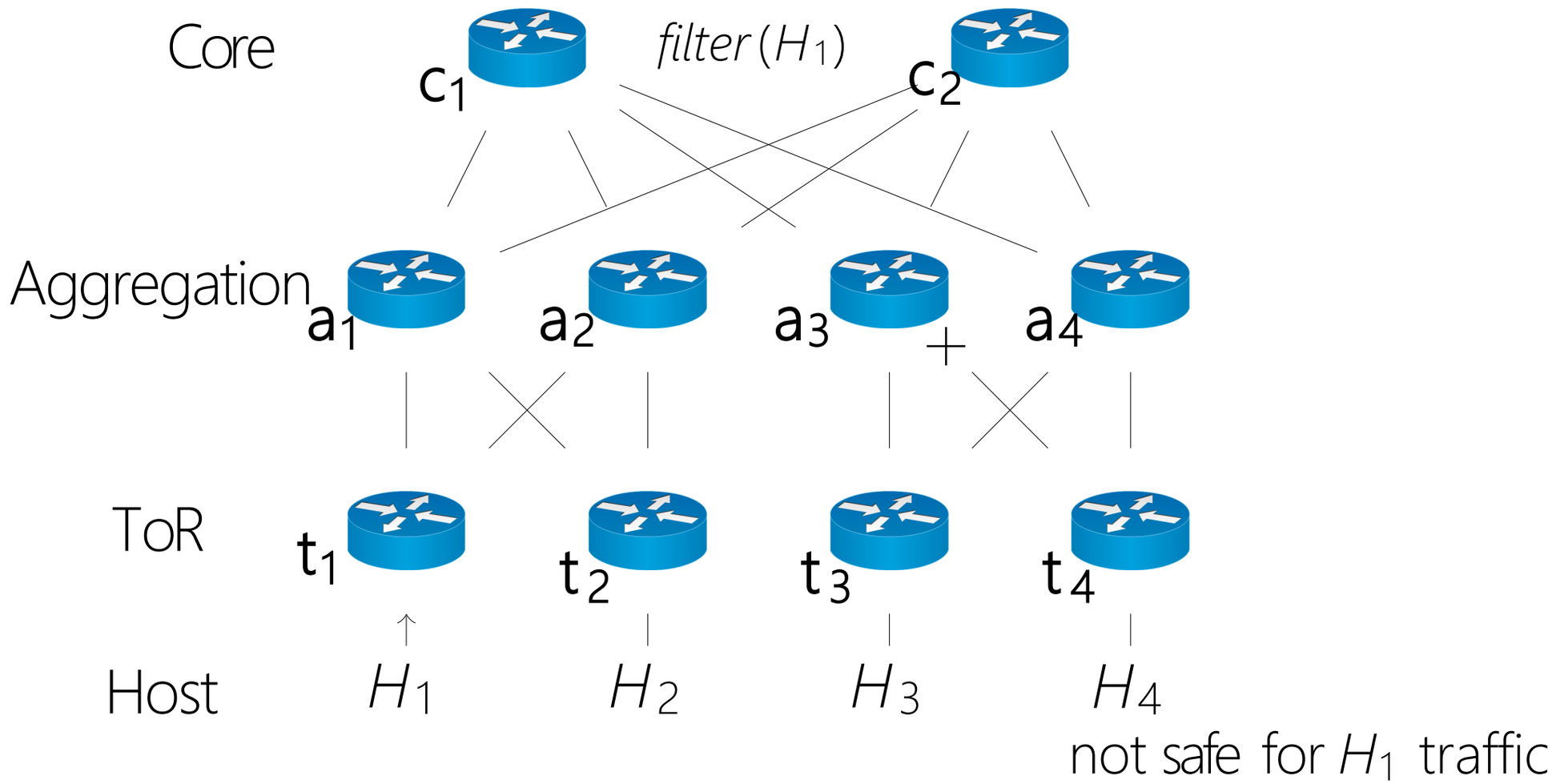
We use a special relation symbol **D** for dropping a packet

$$\mathbf{t}_1(dst, typ) \wedge (dst \neq 1) \rightarrow \mathbf{a}_1(dst, typ)$$

$$\mathbf{t}_1(dst, typ) \wedge (dst \neq 1) \rightarrow \mathbf{a}_2(dst, typ)$$

$$\mathbf{t}_1(dst, typ) \wedge \neg\ (\ (dst \geq 1) \wedge (dst \leq 4) \wedge (typ \geq 0) \wedge (typ \leq 7)\ ) \rightarrow \mathbf{D}(dst, typ)$$

Horn Clauses for Network



Properties. Flow 0 should not reach destination 4 or the drop state

$$\mathbf{t}_4(dst, typ) \wedge (typ = 0) \rightarrow false$$
$$\mathbf{D}(dst, typ) \wedge (typ = 0) \rightarrow false$$

$$(typ = 0 \ \wedge \ dst \ \in \{2, 3, 4\}) \ \rightarrow \mathbf{t_1}(dst, \ typ)$$

…

$$\mathbf{t_1}(dst, typ) \wedge \ (dst \neq 1) \rightarrow \mathbf{a_1}(dst, \ typ)$$

…

$$\mathbf{a_1}(dst, typ) \ \wedge \ (dst \neq 1) \wedge (dst \neq 2) \ \rightarrow \mathbf{c_2}(dst, typ)$$

…

$$\mathbf{c_2}(dst, typ) \wedge (dst = 3 \vee dst = 4) \rightarrow \mathbf{a_4}(dst, typ)$$

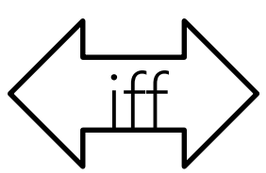
…

$$\mathbf{a_4}(dst, typ) \ \wedge \ (dst = 4) \ \rightarrow \mathbf{t_4}(dst, typ)$$

…

$$\mathbf{t_4}(dst, typ) \ \wedge (typ = 0) \ \rightarrow \text{false}$$

- Set of **Horn** Clauses
- An implication:
  - Conjunction of positive literals in premise
  - Single positive literal in conclusion

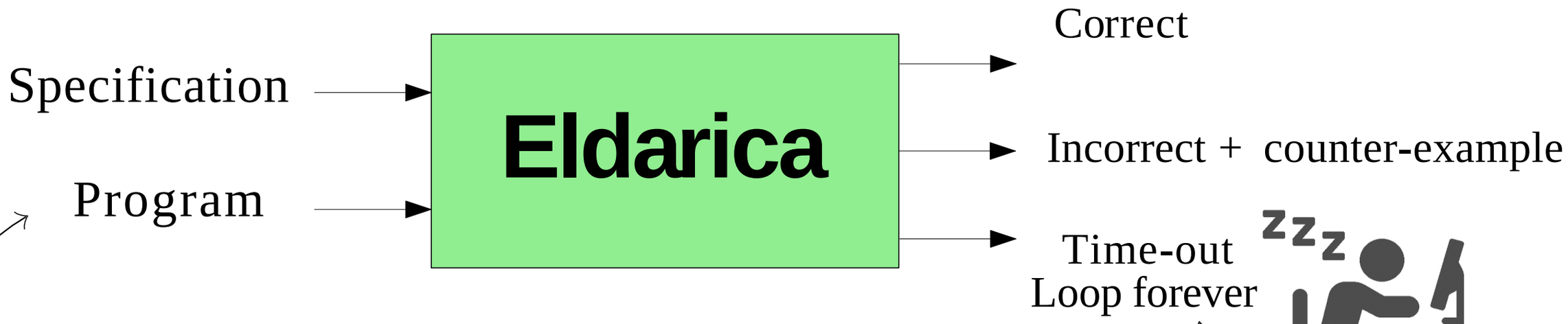
Network is safe  Clauses are valid

- Clauses are invalid here (dst = 4, typ = 0)

Horn Clause Solvers

- Duality
  - http://research.microsoft.com/en-us/projects/duality/
- HSF
  - http://www7.in.tum.de/tools/hsf/
- Eldarica
  - http://lara.epfl.ch/w/eldarica
- PDR implementation in Z3
  - http://z3.codeplex.com/
- SPACER
  - http://spacer.bitbucket.org/

https://github.com/uuverifiers/eldarica



- Novel techniques for abstraction refinement:
  1. Accelerated Intperolants
  2. Disjunctive Interpolants
- Support common representation languages (Horn clauses, Numerical Transition Systems, ...)

[FM 2012] [ATVA 2012] [VSTTE 2013] [CAV 2013] [FMSD 2015]


A Verification Toolkit for Numerical Transition Systems
Tool Paper*


Accelerating Interpolants*

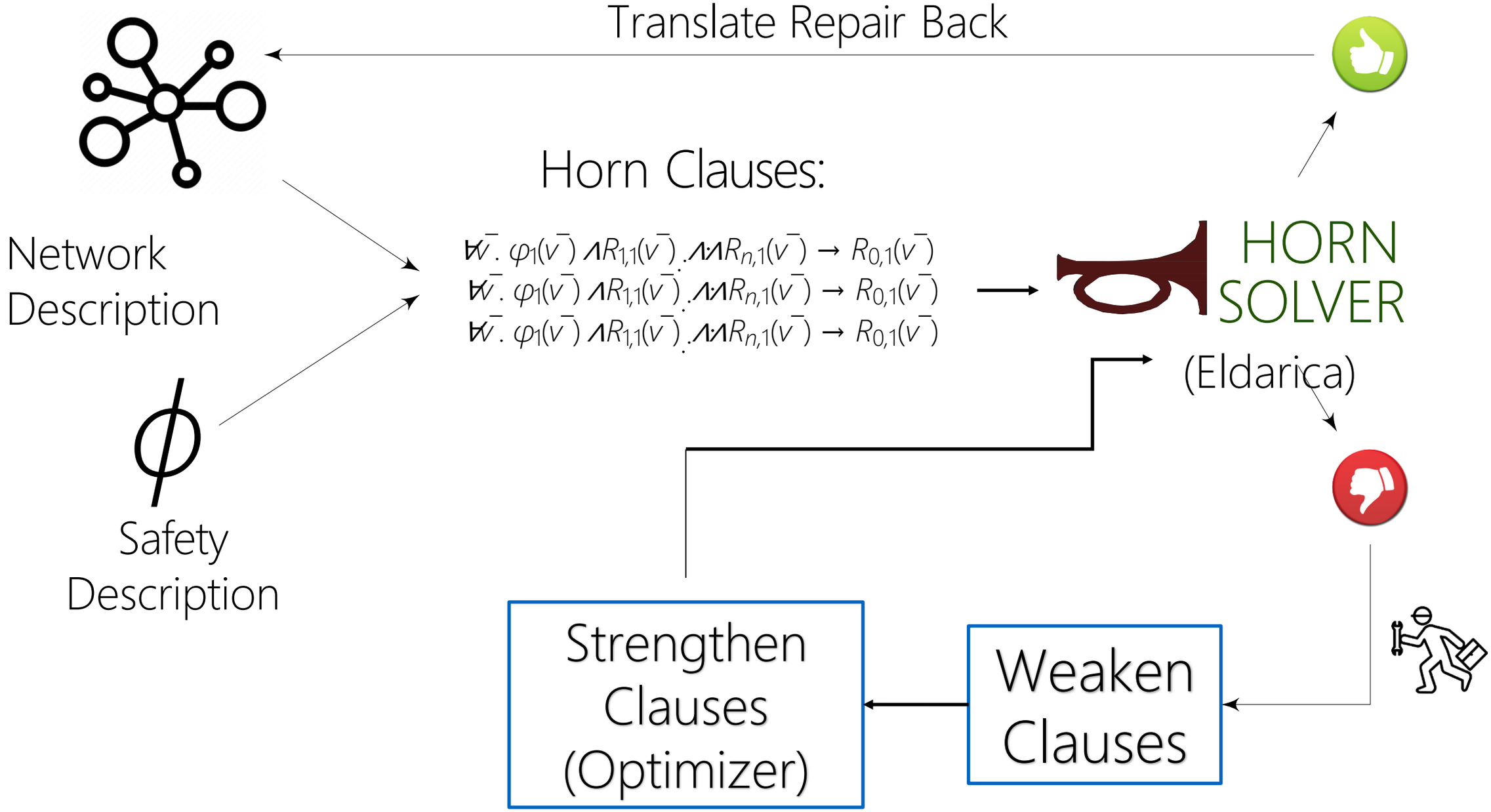

Classifying and Solving Horn Clauses for Verification


Disjunctive Interpolants for Horn-Clause Verification

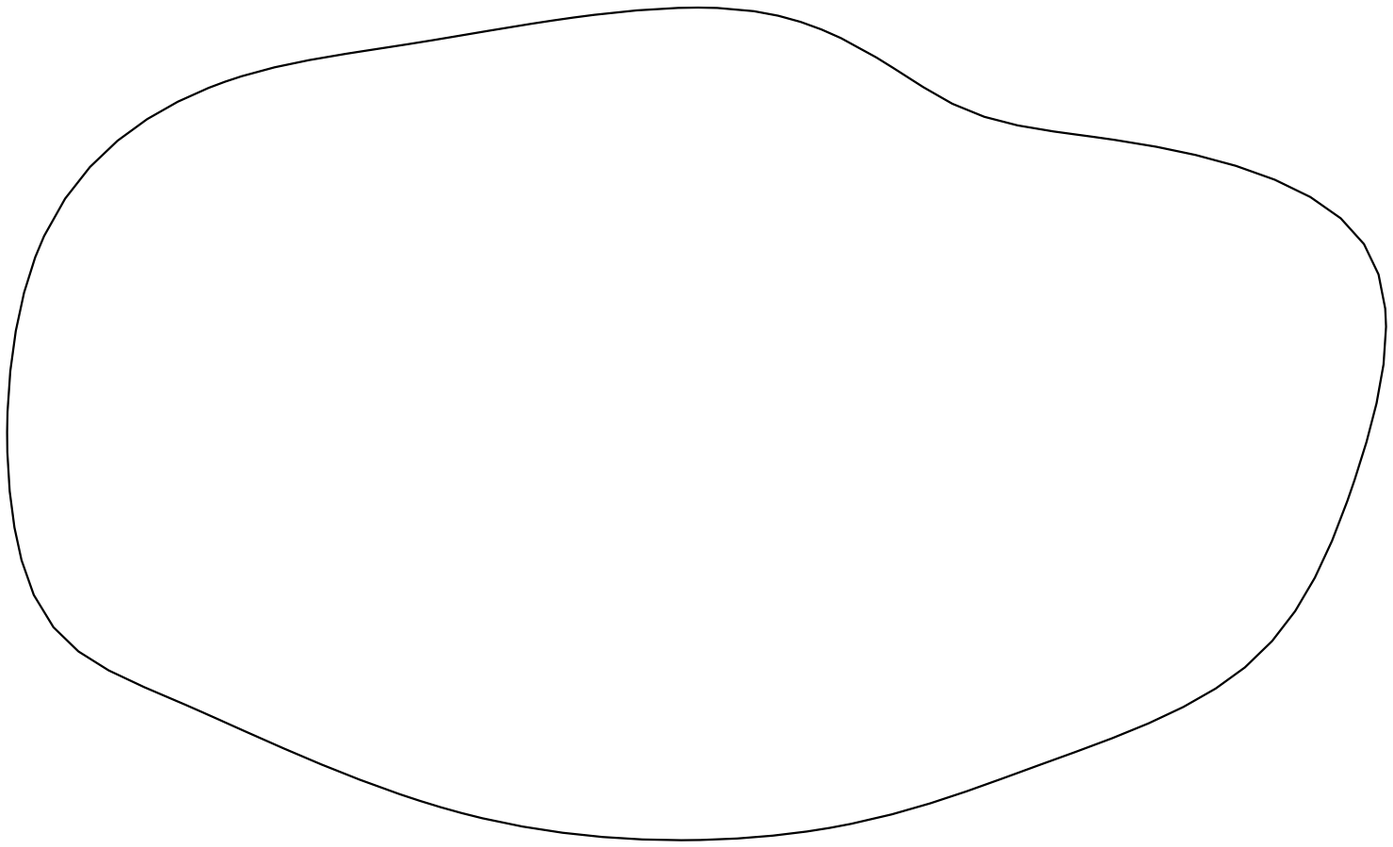

On recursion-free Horn clauses and Craig interpolation

Hossein Hojjat, Philipp Rüemmer
(Filip Konecný, Radu Iosif, Florent Garnier, Pavle Subotic and Viktor Kuncak)

Repair Framework

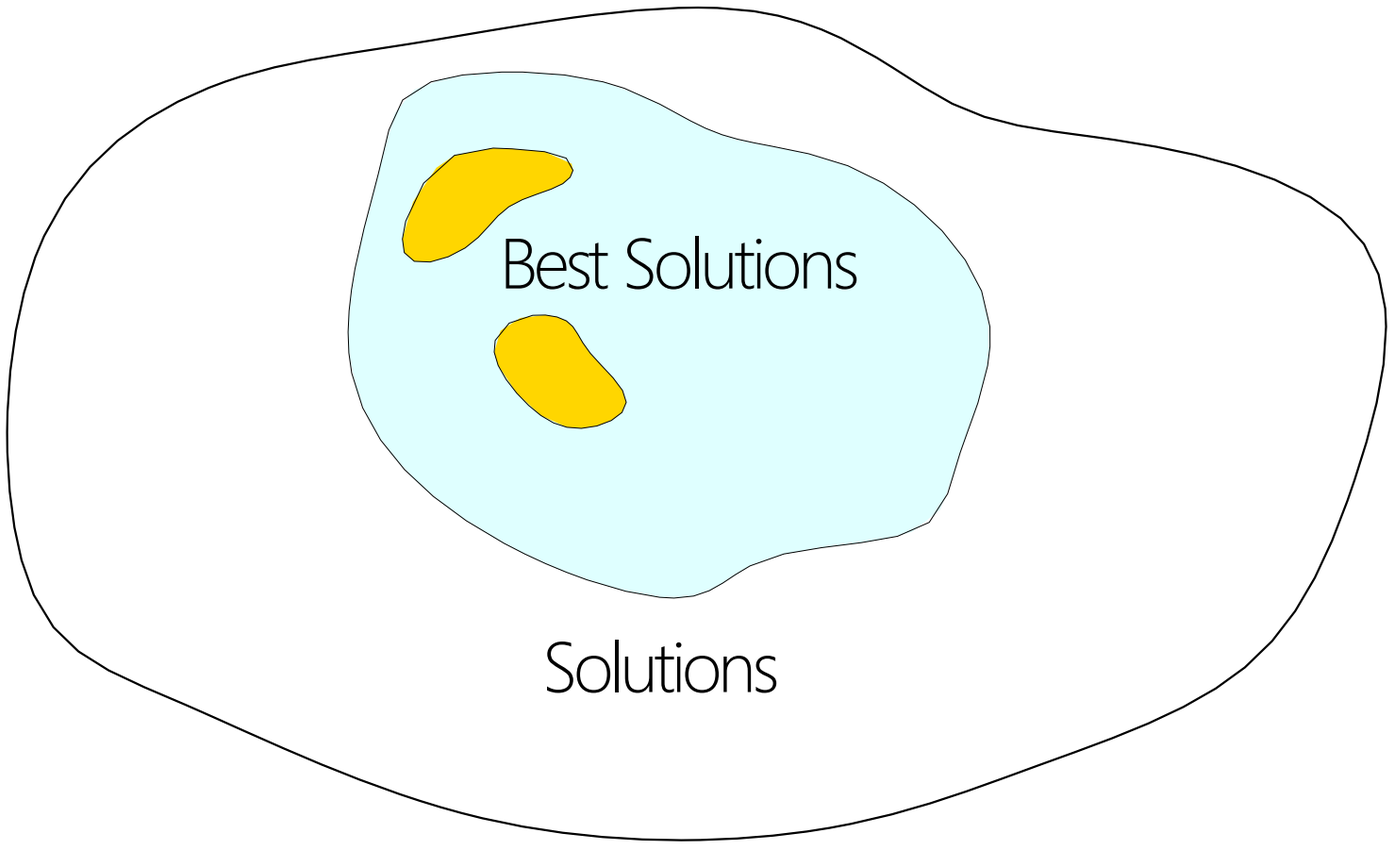


Goal: find solutions for set of Horn clauses subject to objective function



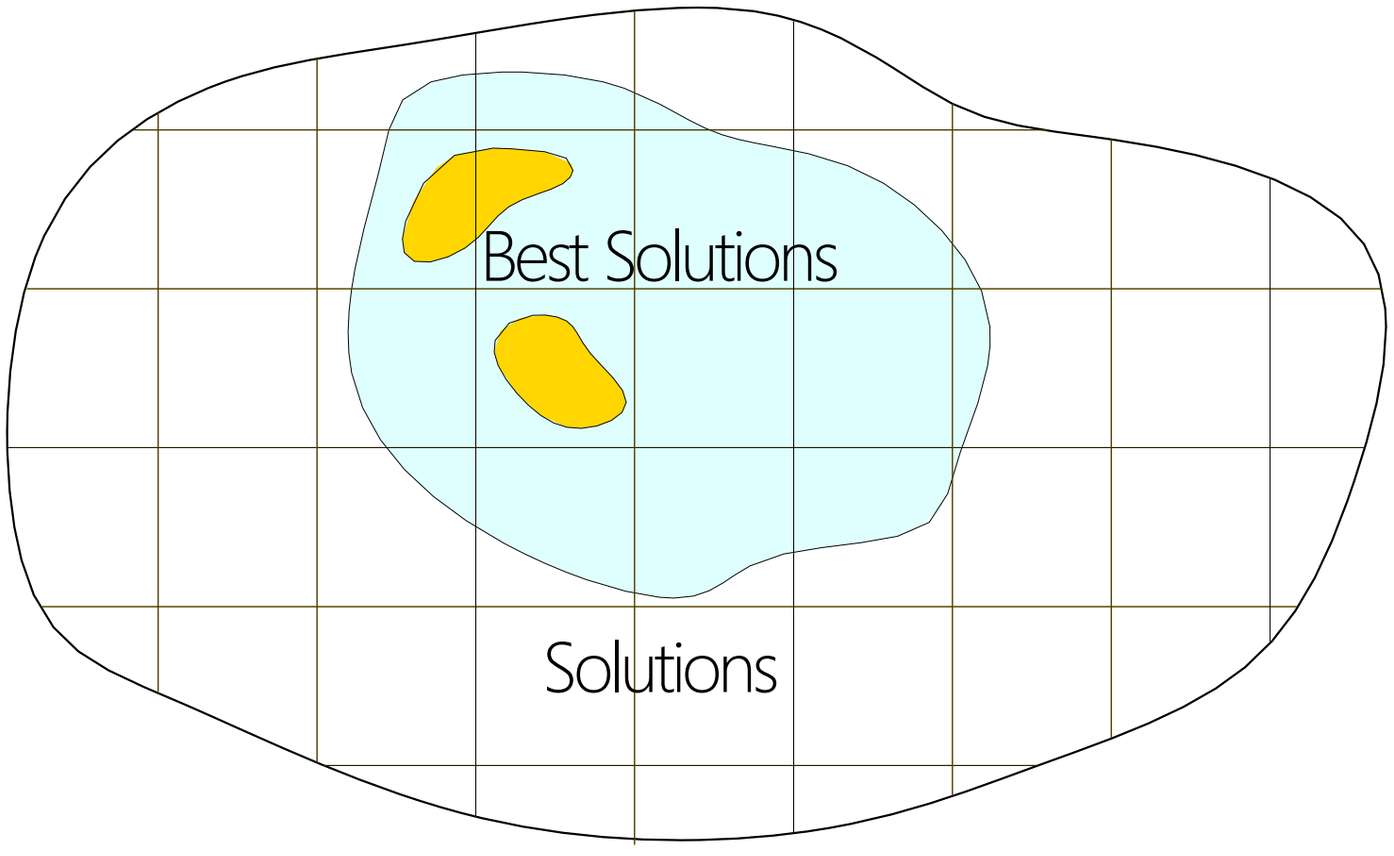
Space of all interpretations of relation symbols

Goal: find solutions for set of Horn clauses subject to objective function



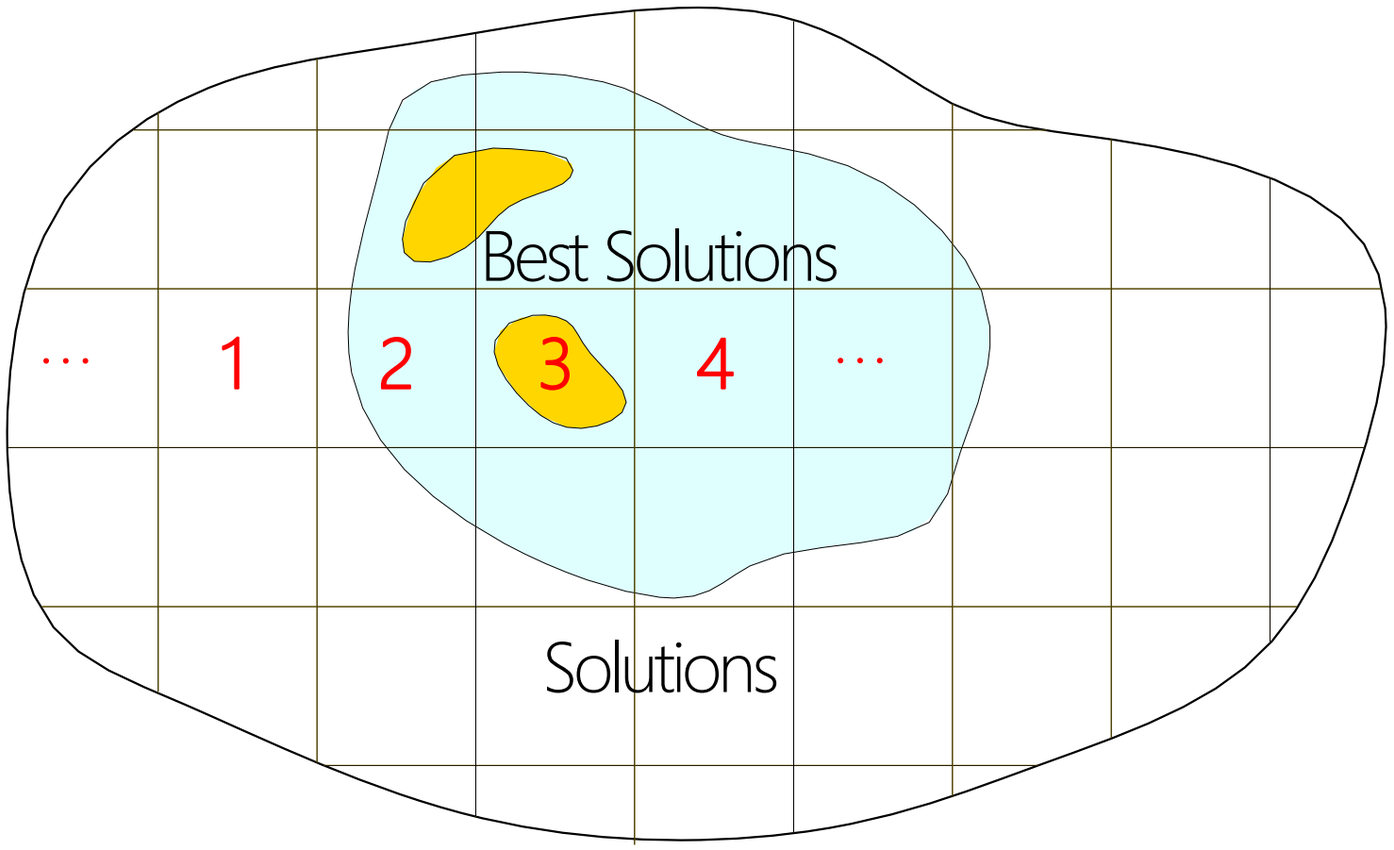
Space of all interpretations of relation symbols

Goal: find solutions for set of Horn clauses subject to objective function

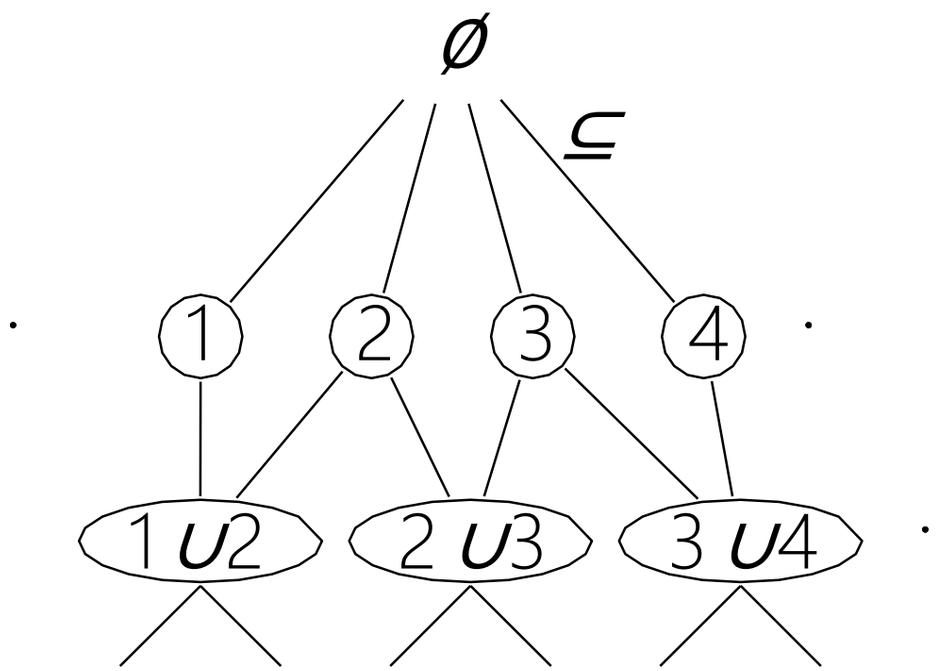


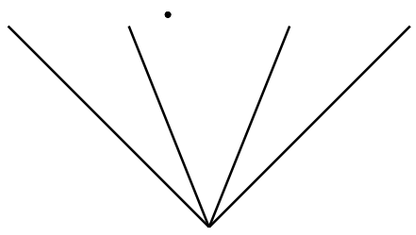
Space of all interpretations of relation symbols

Goal: find solutions for set of Horn clauses subject to objective function



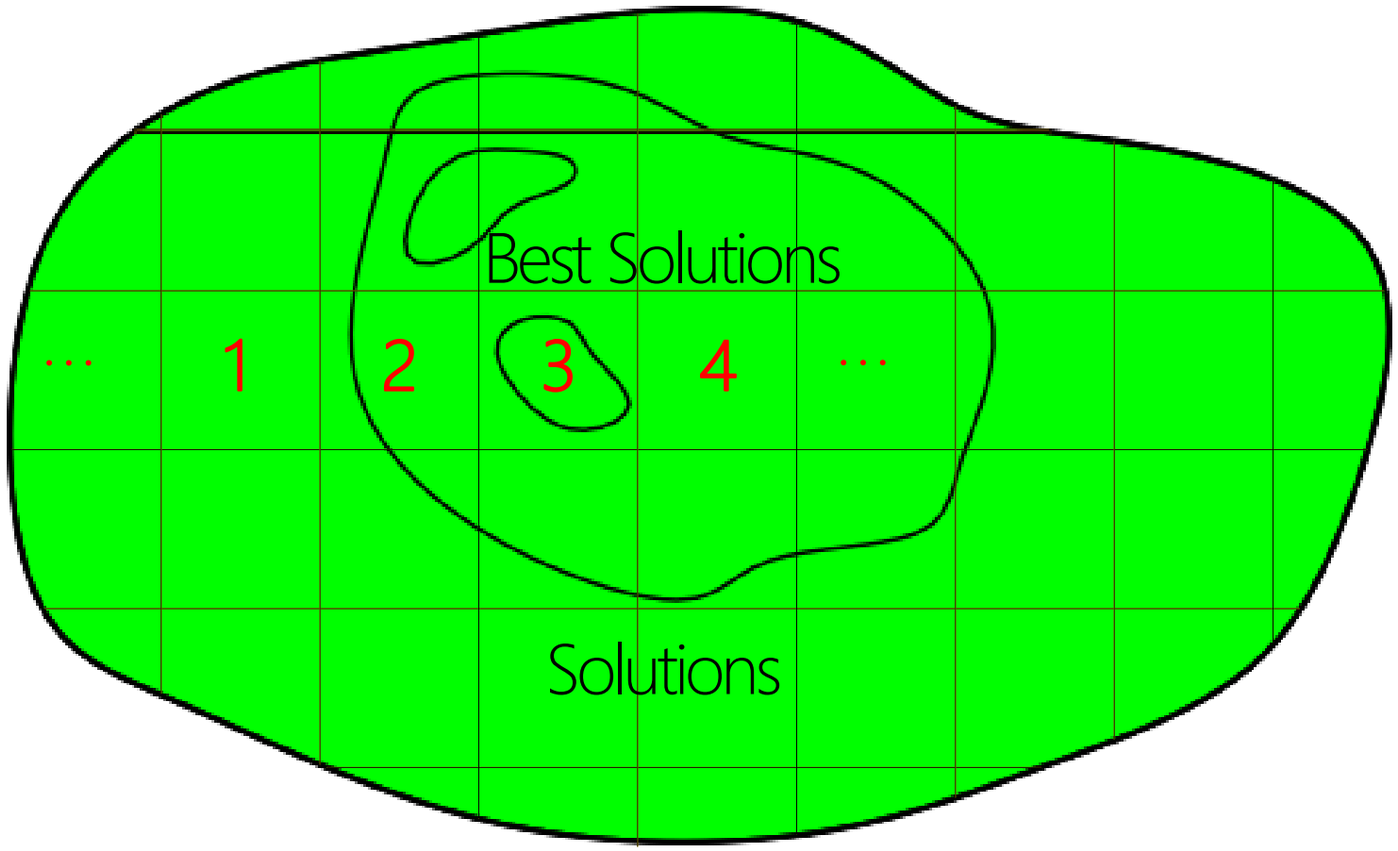
Space of all interpretations of relation symbols



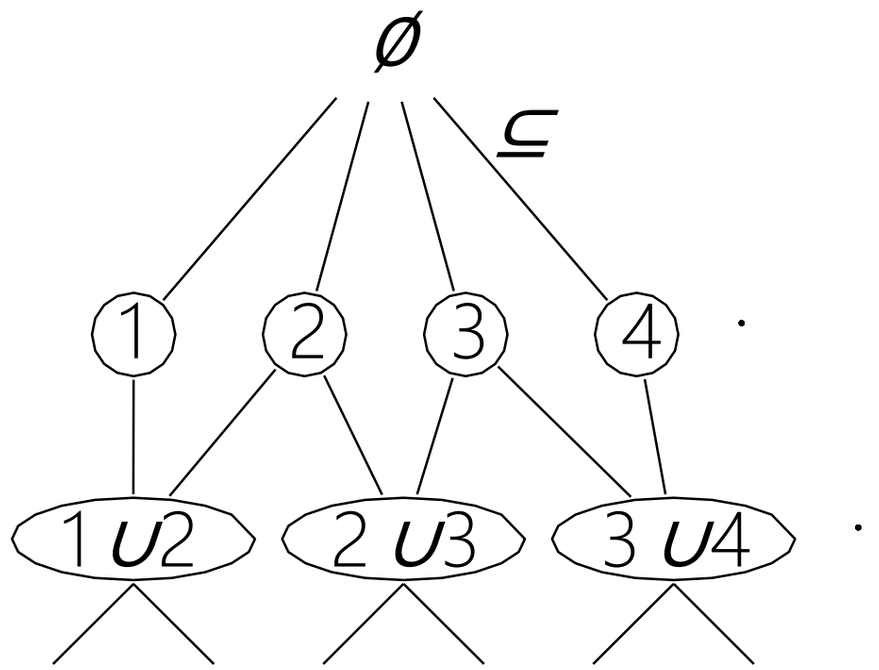


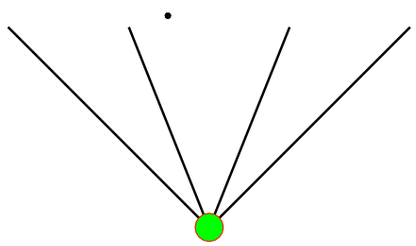
all interpretations

Goal: find solutions for set of Horn clauses subject to objective function



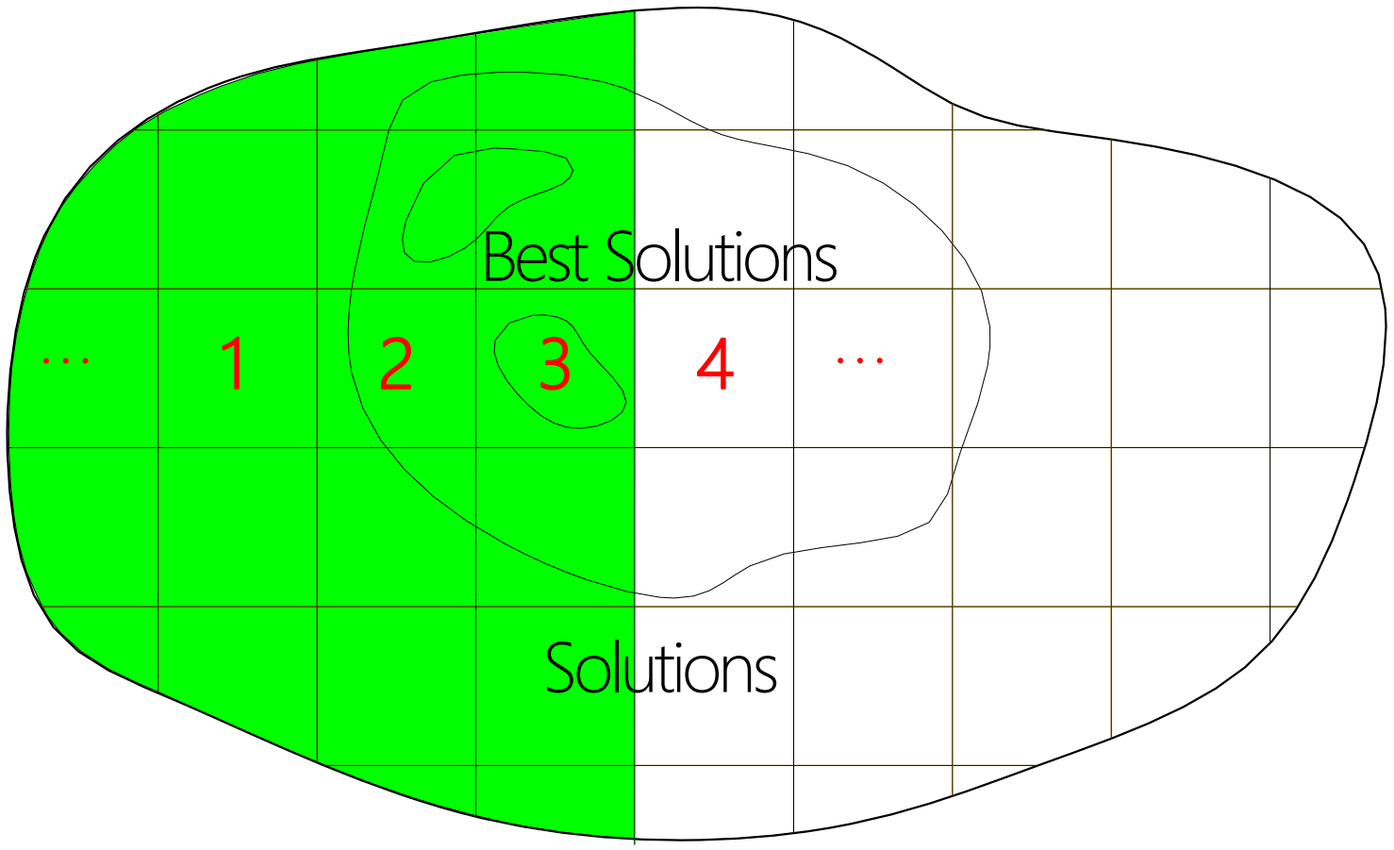
Space of all interpretations of relation symbols



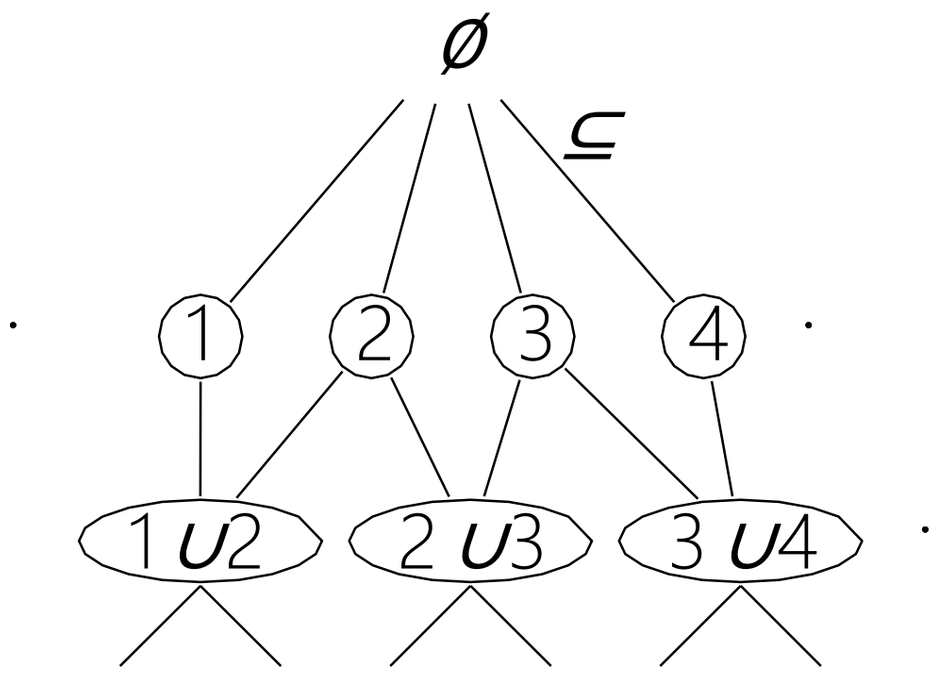


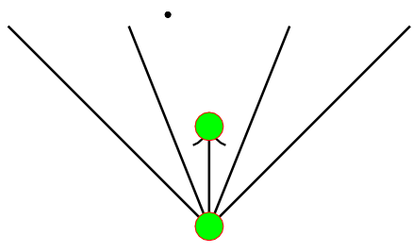
all interpretations

Goal: find solutions for set of Horn clauses subject to objective function



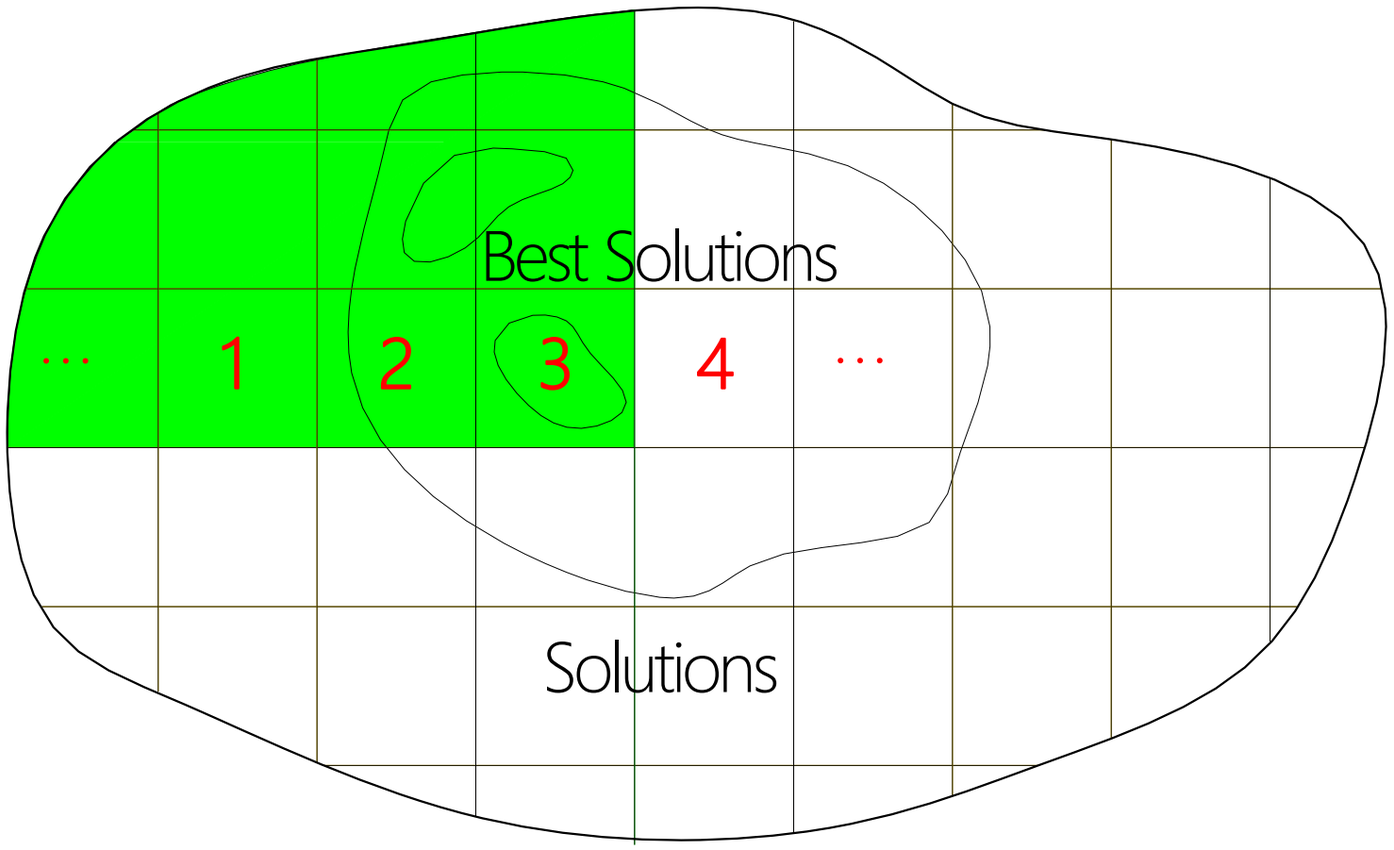
Space of all interpretations of relation symbols



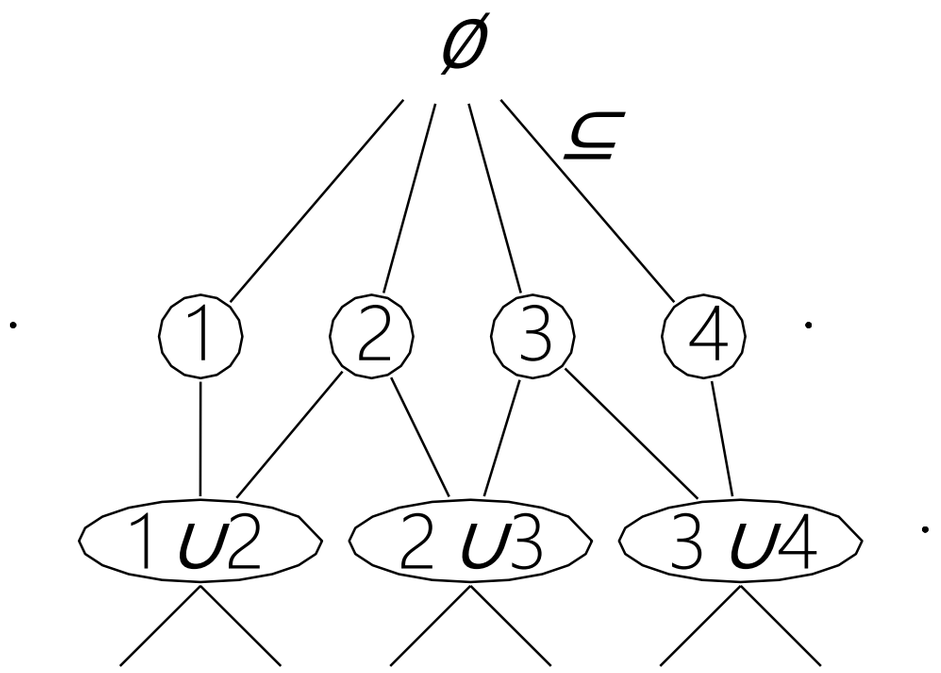


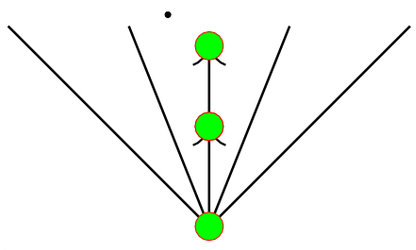
all interpretations

Goal: find solutions for set of Horn clauses subject to objective function



Space of all interpretations of relation symbols



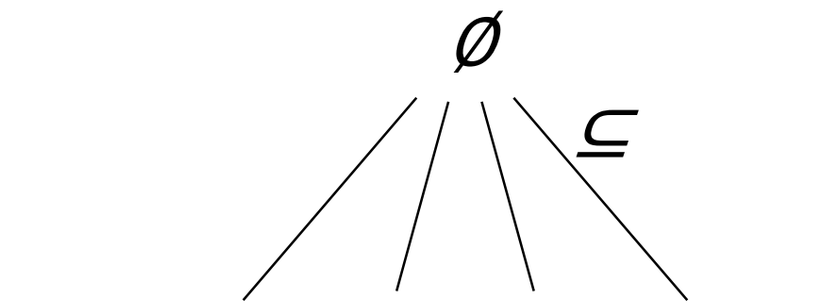


all interpretations

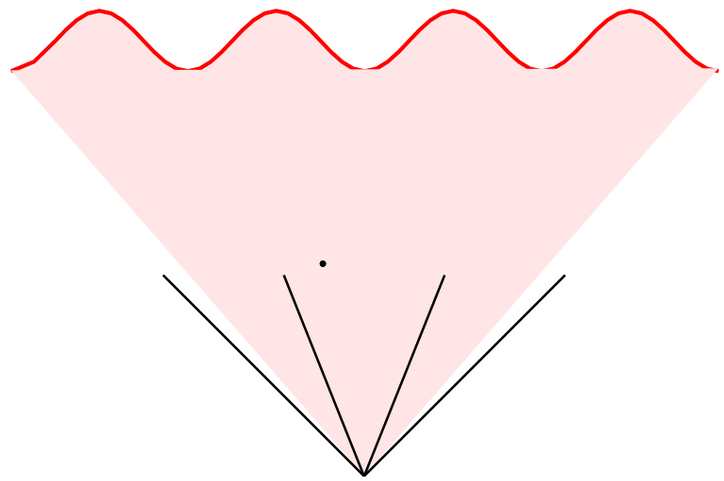
Goal: find solutions for set of Horn clauses subject to objective function

Objective function:

Rank nodes of lattice monotonically



Feasibility Frontier



all interpretations

Goal: find solutions for set of Horn clauses subject to objective function

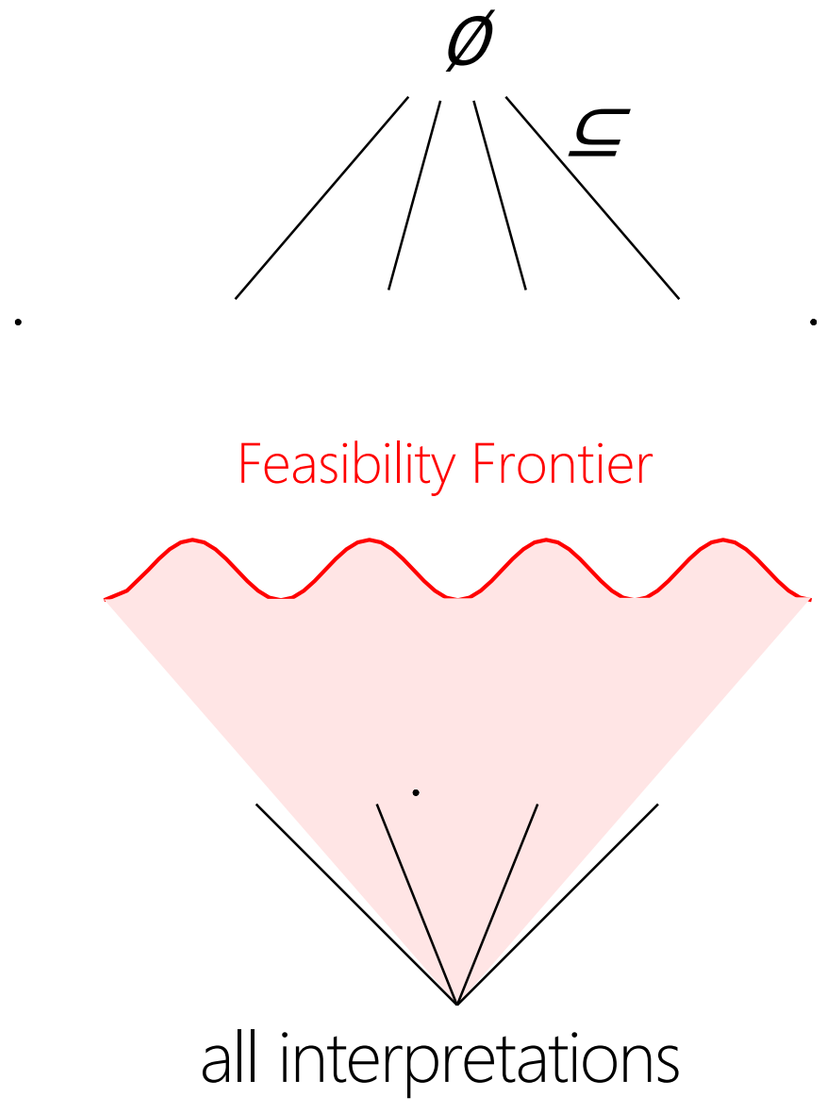
Objective function:

Rank nodes of lattice monotonically

Search Algorithm:

Walk smartly in the lattice to find the **best** solution:

- inside the feasibility cone
- has maximum ranking



Goal: find solutions for set of Horn clauses subject to objective function

Objective function:

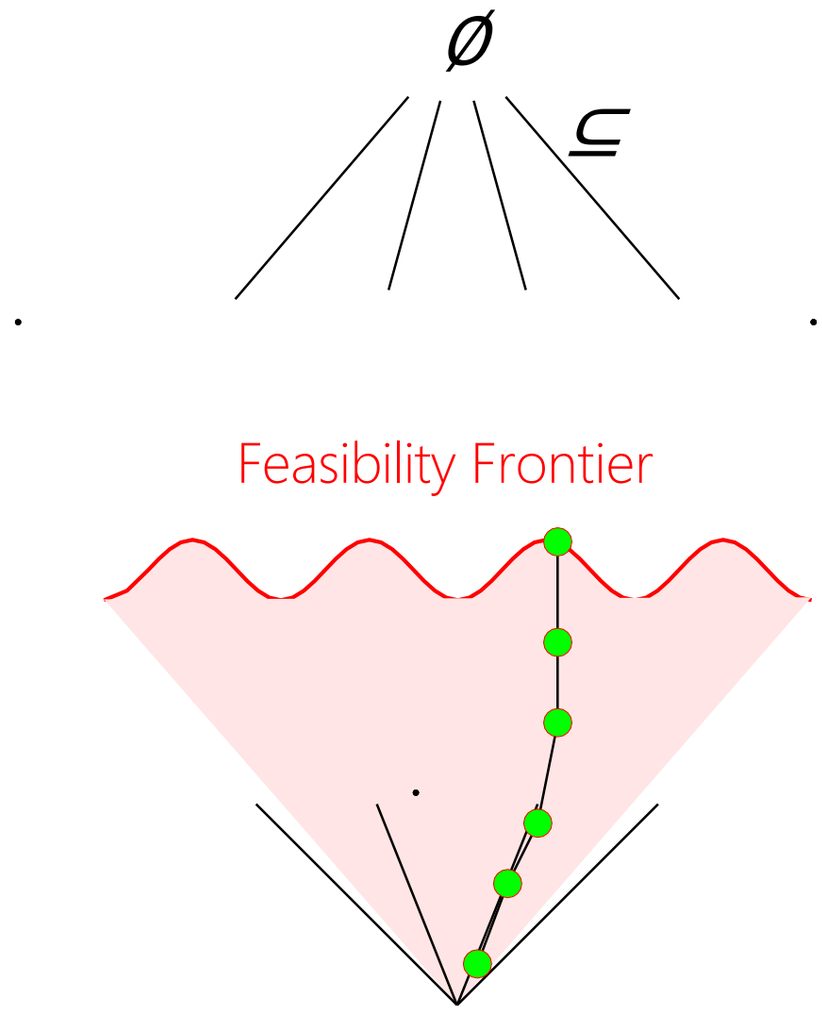
Rank nodes of lattice monotonically

Search Algorithm:

Walk smartly in the lattice to find the **best** solution:

- inside the feasibility cone
- has maximum ranking

- 1 Pick a feasible node and walk until reach frontier



Goal: find solutions for set of Horn clauses subject to objective function

Objective function:

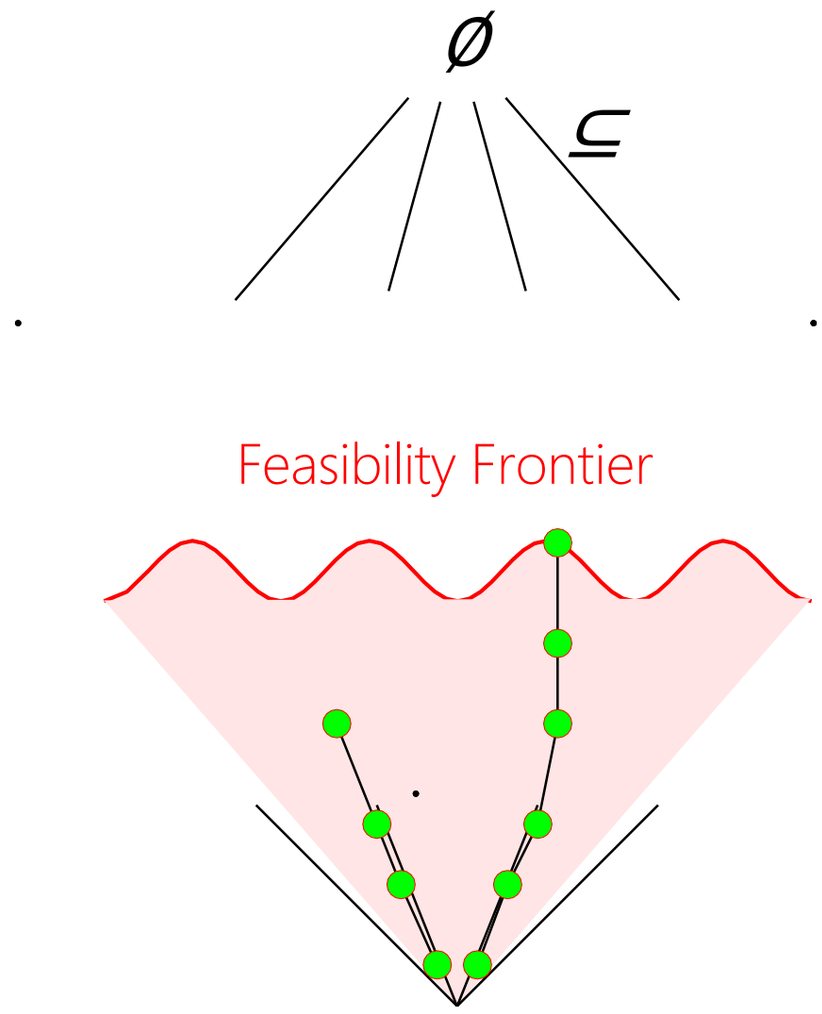
Rank nodes of lattice monotonically

Search Algorithm:

Walk smartly in the lattice to find the **best** solution:

- inside the feasibility cone
- has maximum ranking

1. Pick a feasible node and walk until reach frontier
- Pick a lower rank incomparable node and walk again



Goal: find solutions for set of Horn clauses subject to objective function

Objective function:

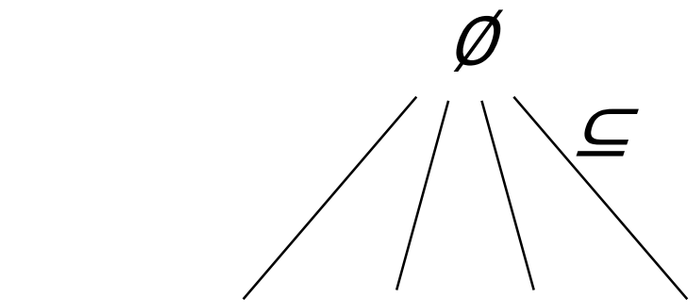
Rank nodes of lattice monotonically

Search Algorithm:

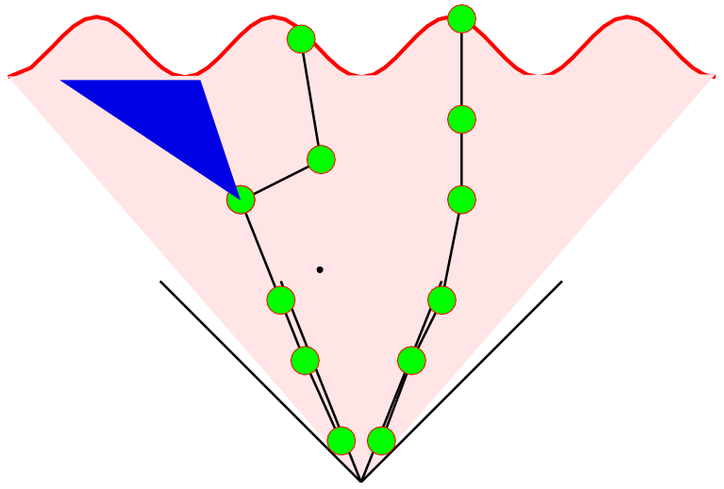
Walk smartly in the lattice to find the **best** solution:

- inside the feasibility cone
- has maximum ranking

1. Pick a feasible node and walk until reach frontier
- Pick a lower rank incomparable node and walk again
- Use feasibility bounds as heuristic to prune search



Feasibility Frontier



all interpretations

Goal: find solutions for set of Horn clauses subject to objective function

Objective function:

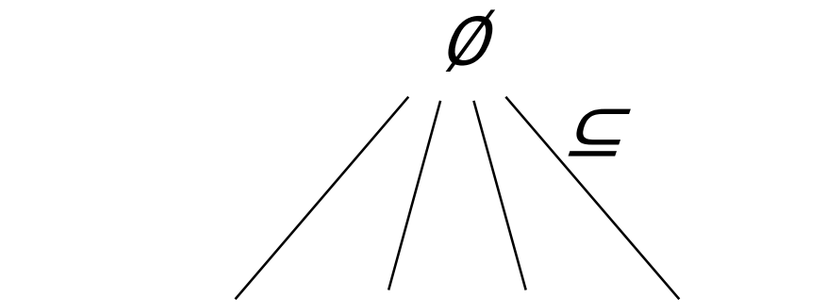
Rank nodes of lattice monotonically

Search Algorithm:

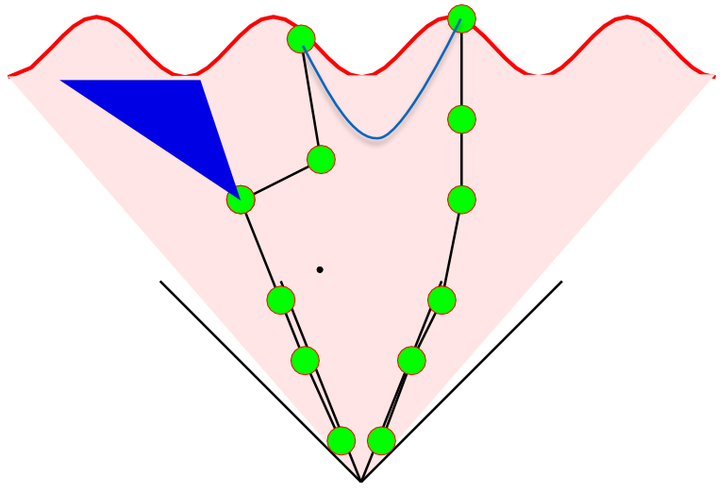
Walk smartly in the lattice to find the **best** solution:

- inside the feasibility cone
- has maximum ranking

1. Pick a feasible node and walk until reach frontier
- Pick a lower rank incomparable node and walk again
- Use feasibility bounds as heuristic to prune search



Feasibility Frontier



all interpretations

- Search algorithm is guaranteed to terminate on finite lattices

Theorem

- Optimization algorithm is sound and complete
  - Always finds the global optimum

Proof

- Induction on lattice structure
  - use monotonicity of feasibility and objective function

Implementation and Experiments

- We use Internet Topology Zoo - real world topologies
- Randomly generate forwarding tables to connect hosts
- Make a set of nodes unsafe for certain types of traffics
- Repair the buggy network with updating a minimal number o switchess

Implementation and Experiments

Benchmarks	#Nodes	#Links	#Rels.	#Lattice	#Eld	Time(s)
Gridnet	9	20	–	–	–	–
Cesnet200304	29	33	3	2.22×10^{10}	145	4.98
Arpanet19706	9	10	3	2.22×10^{10}	91	2.98
Oxford	20	26	8	3.89×10^{27}	664	16.70
Garr200902	54	71	6	4.92×10^{20}	3045	107.62
Getnet	7	8	2	7.90×10^{6}	61	1.45
Surfnet	50	73	3	2.22×10^{10}	101	3.49
Itnet	11	10	1	2.81×10^{3}	17	0.18
Garr199904	23	25	1	2.81×10^{3}	19	0.33
Darkstrand	28	31	5	1.75×10^{17}	425	14.81
Carnet	44	43	2	7.90×10^{6}	37	0.49
Atmnet	21	22	1	2.81×10^{3}	15	0.67
HiberniaCanada	13	14	11	8.63×10^{37}	1795	84.56
Evolink	37	45	1	2.81×10^{3}	14	0.20
Dfn	58	87	–	–	–	–
Ernet	30	32	4	6.23×10^{13}	140	4.94
Bren	37	38	6	4.92×10^{20}	974	25.14
Niif	36	41	2	7.90×10^{6}	48	0.92
Renater2001	24	27	3	2.22×10^{10}	101	3.56
Latnet	69	74	2	7.90×10^{6}	47	0.64

References

- Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. **Abstractions for Network Update**. (SIGCOMM 2012)
- Jedidiah McClurg, Hossein Hojjat, Pavol Cerny, and Nate Foster. **Efficient Synthesis of Network Updates**. (PLDI 2015)
- Jedidiah McClurg, Hossein Hojjat, Nate Foster, and Pavol Cerny. **Event-Driven Network Programming**. (PLDI 2016)
- Hossein Hojjat, Philipp Rümmer, Jedidiah McClurg, Pavol Cerny, and Nate Foster. **Optimizing Horn Solvers for Network Repair**. (FMCAD 2016)
- Jedidiah McClurg, Hossein Hojjat, Pavol Cerny. **Synchronization Synthesis for Network Programs**. (CAV 2017)

Related Work

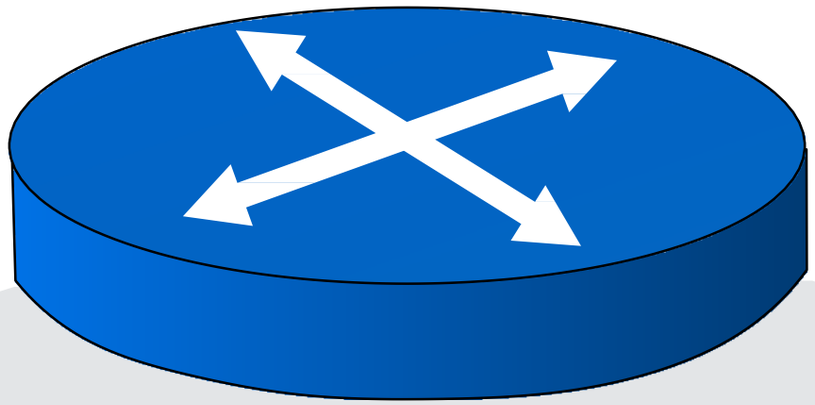
- Ratul Mahajan and Roger Wattenhofer. **On Consistent Updates in Software-Defined Networks**. In ACM SIGCOMM Workshop on Hot Topics in Networking (HotNets), November 2013.

- Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. **Dynamic scheduling of network updates.** in ACM SIGCOMM Conference (SIGCOMM), August 2014.

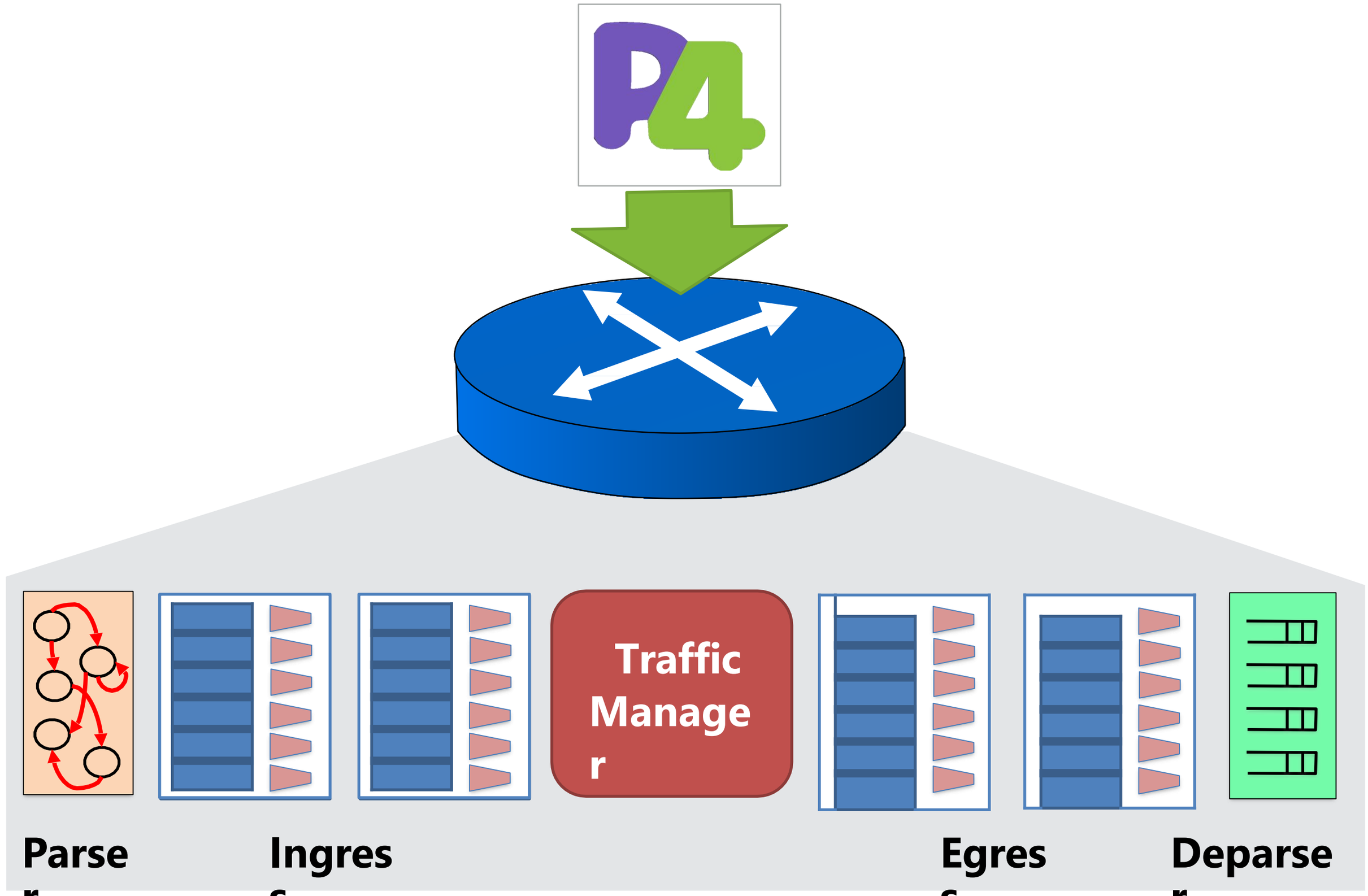
- Giuseppe Bianchi, Marco Bonola, Salvatore Pontarelli, Davide Sanvito, Antonio Capone, and Carmelo Cascone. **Open Packet Processor: a programmable architecture for wire speed platform-independent stateful in-network processing**. In arXiv CoRR abs/1605.01977, May 2016.

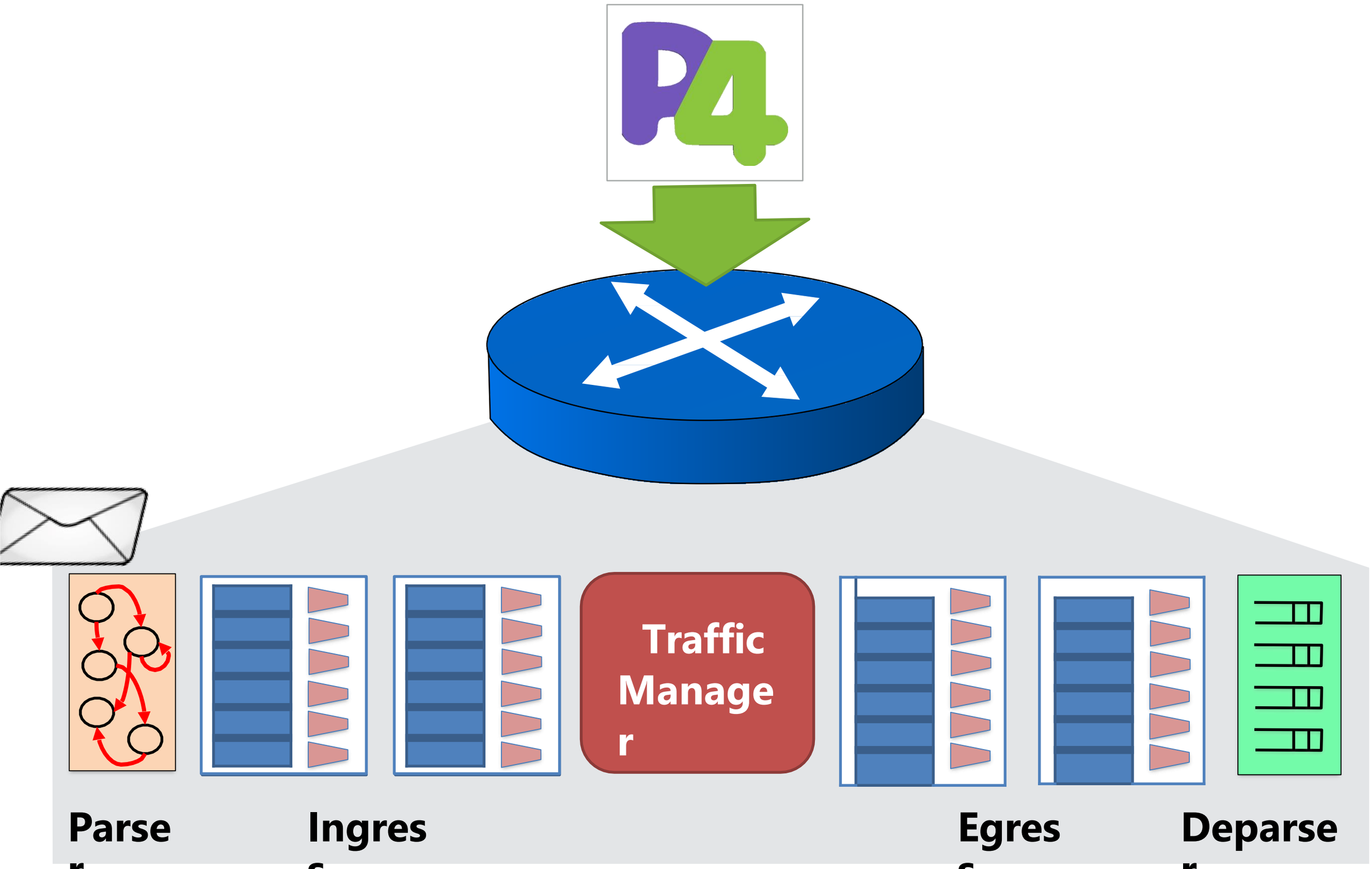
- Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. **SNAP: Stateful network-wide abstractions for packet processing.** In ACM SIGCOMM Conference (SIGCOMM), August 2016.

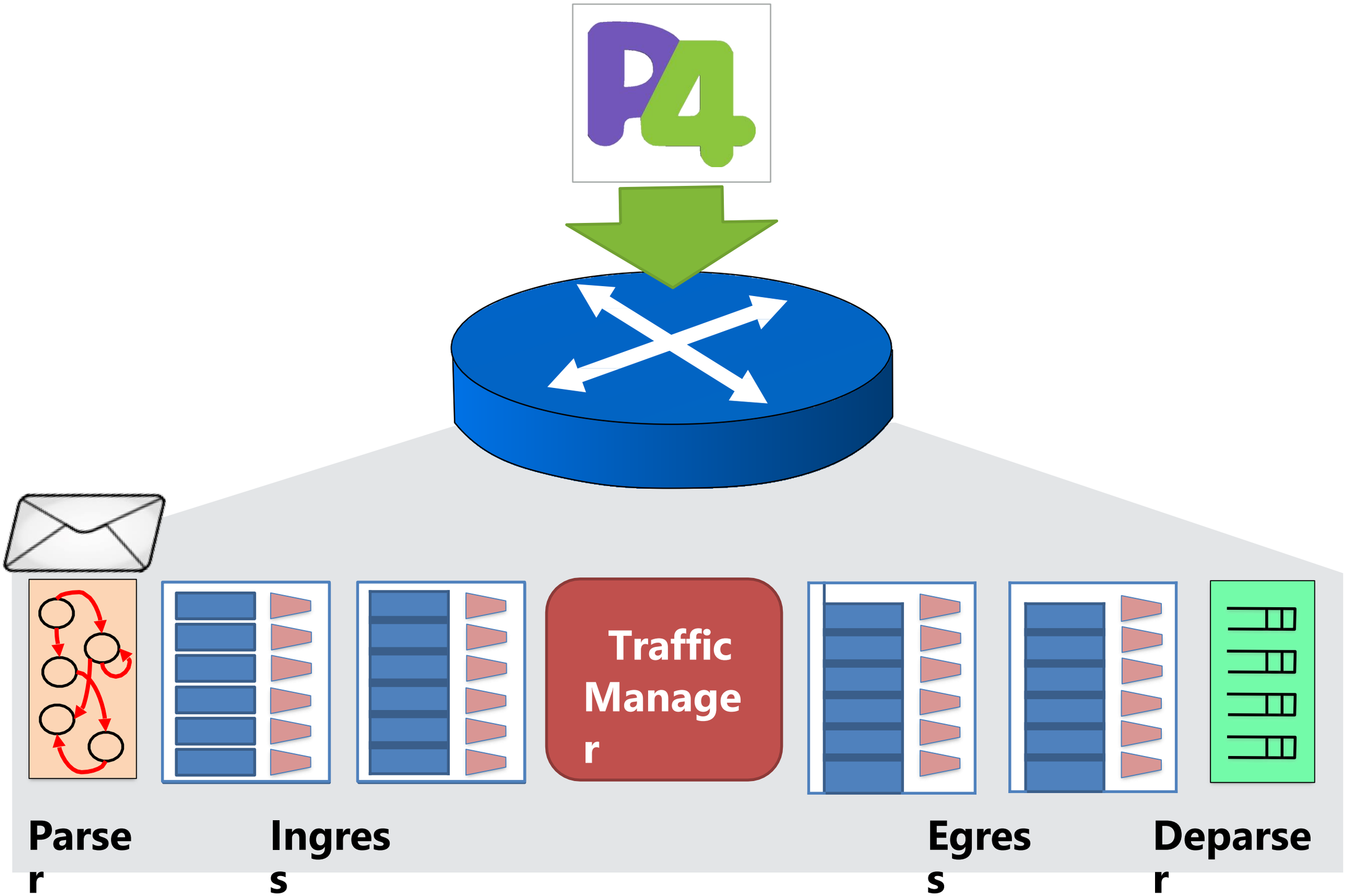
What is next?

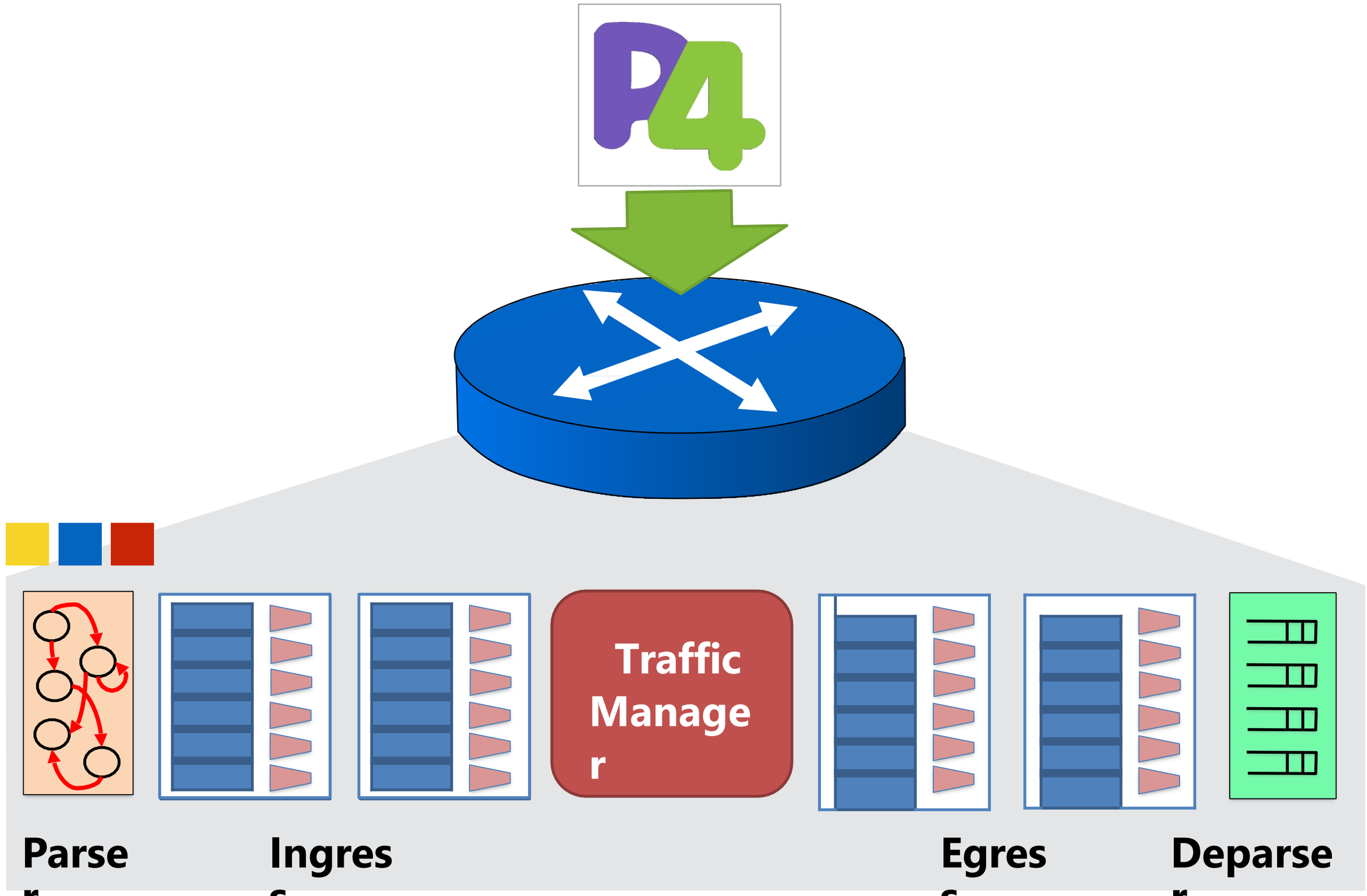


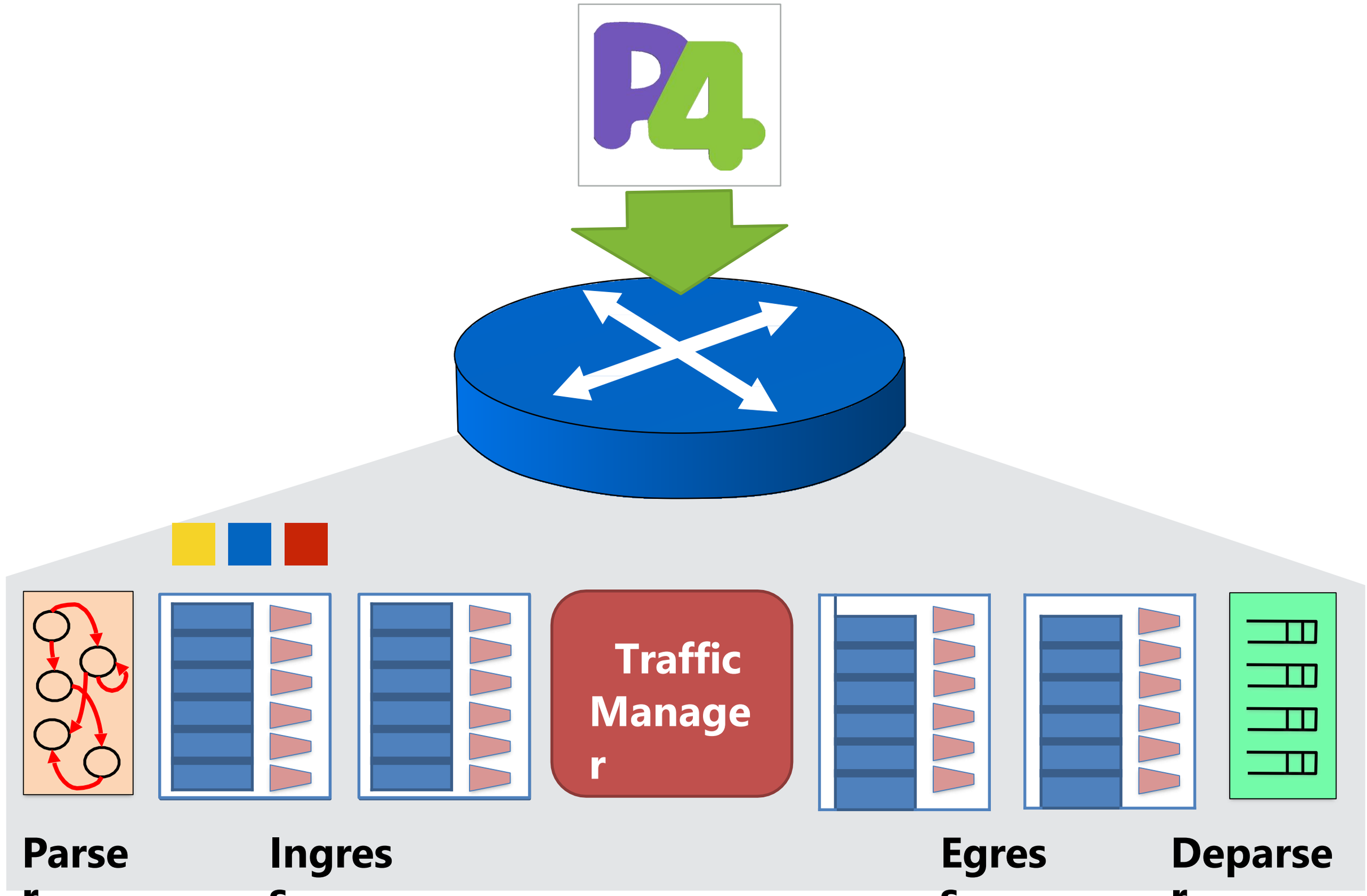

Traffic Manage r
Parse r
Ingres s
Egres s
Deparse r

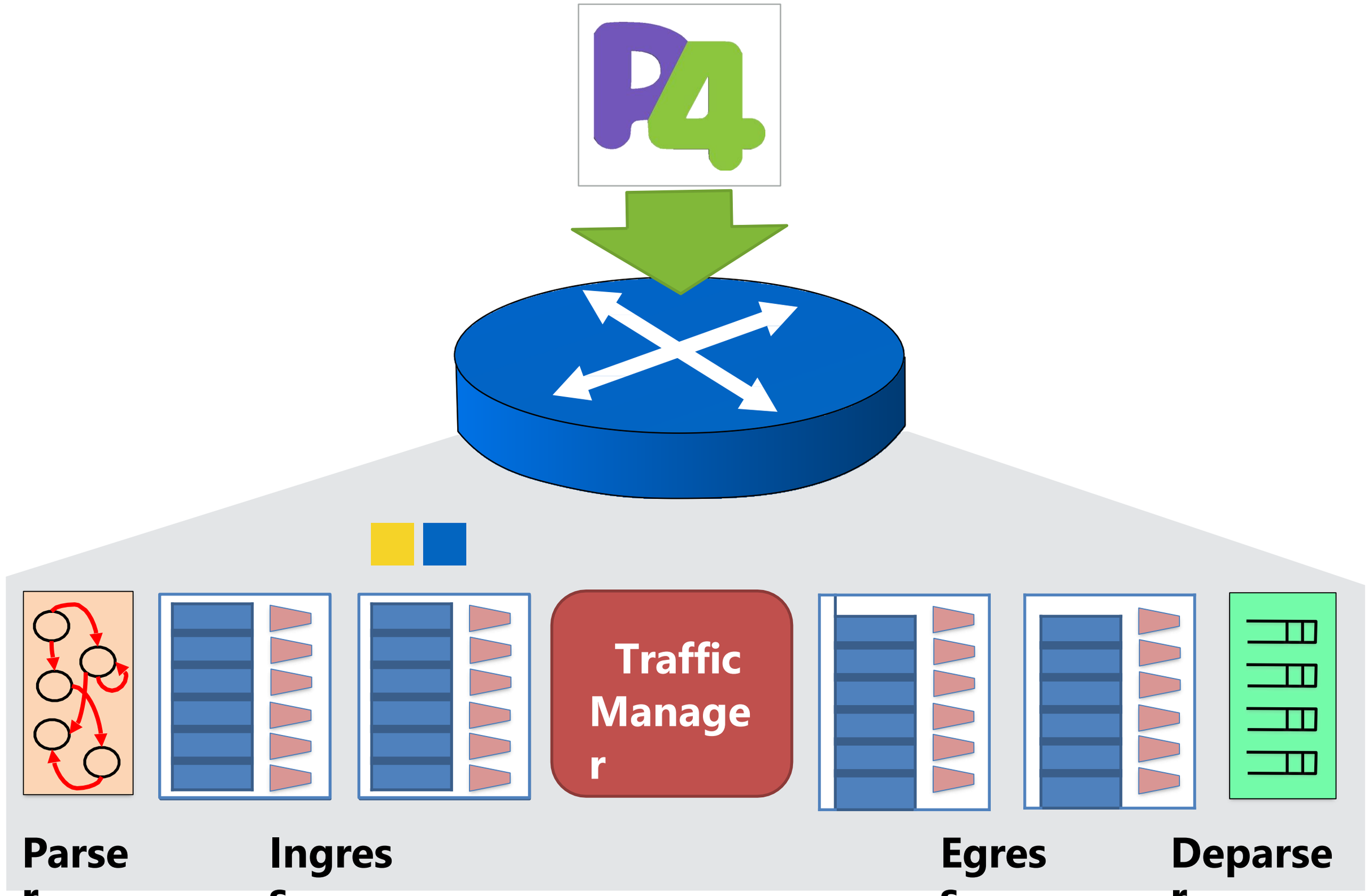

Traffic Manager
Parser
Ingress
Egress
Deparser

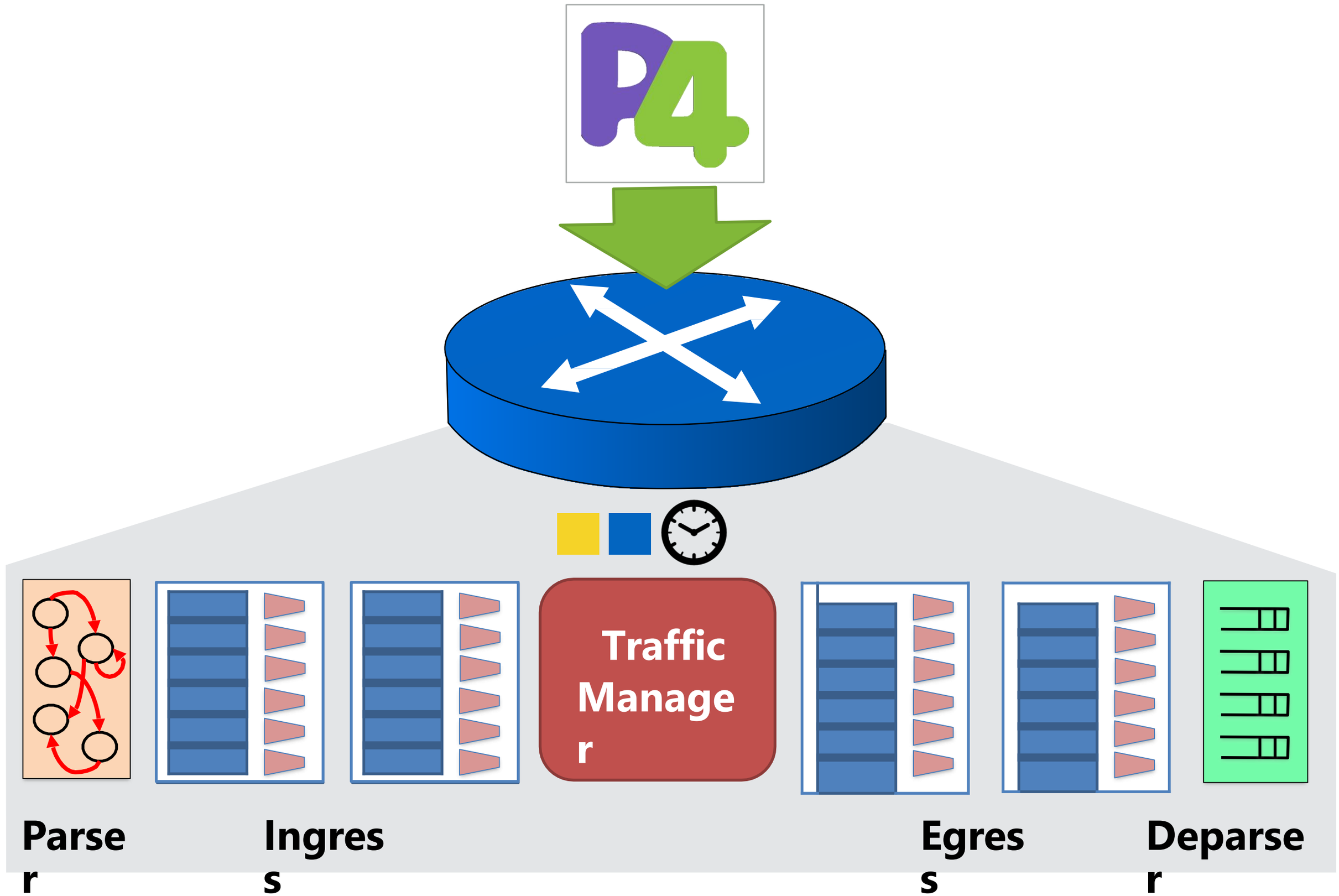

Traffic Manage r
Parse r
Ingres s
Egres s
Deparse r

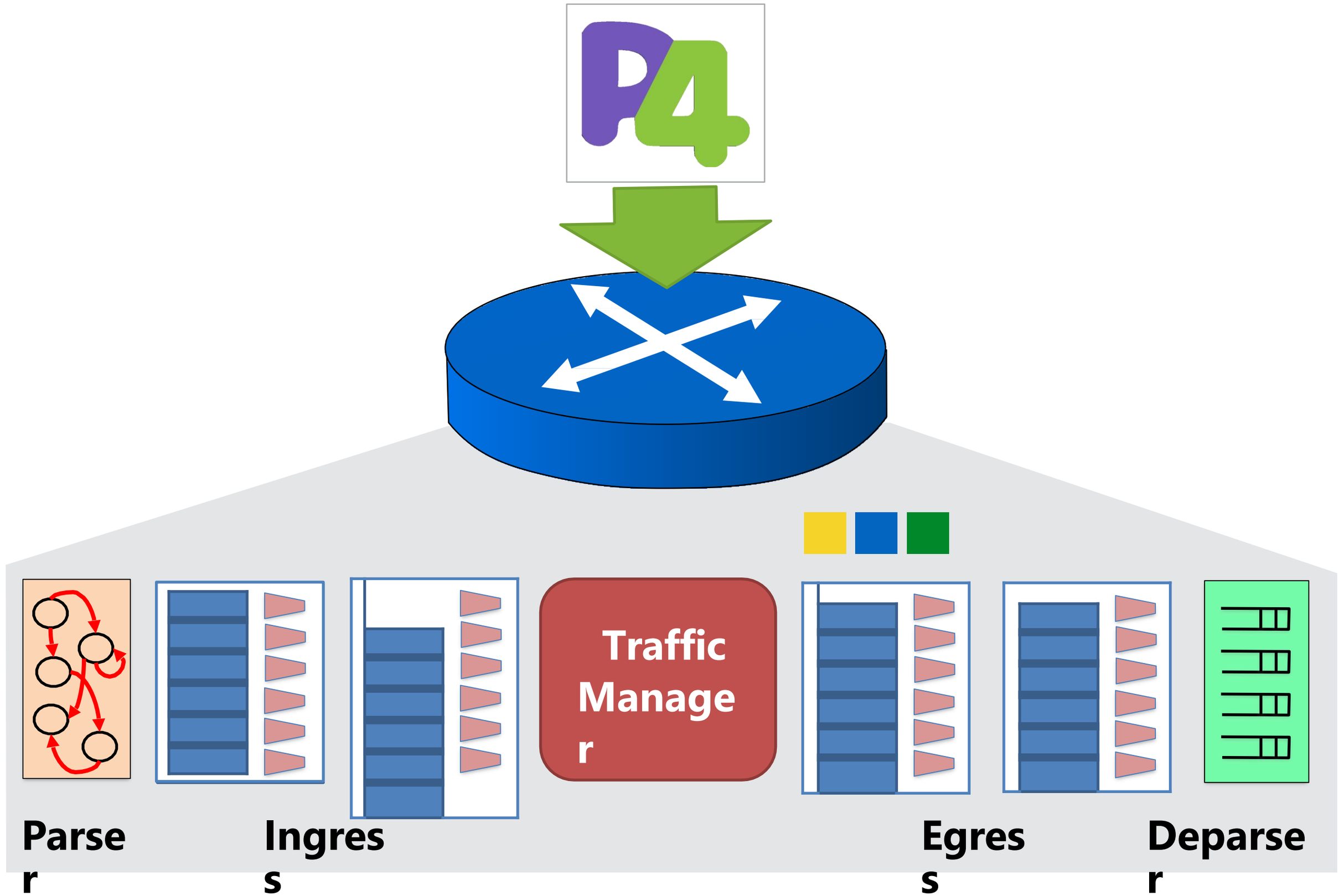

Traffic Manager
Parser
Ingress
Egress
Deparser

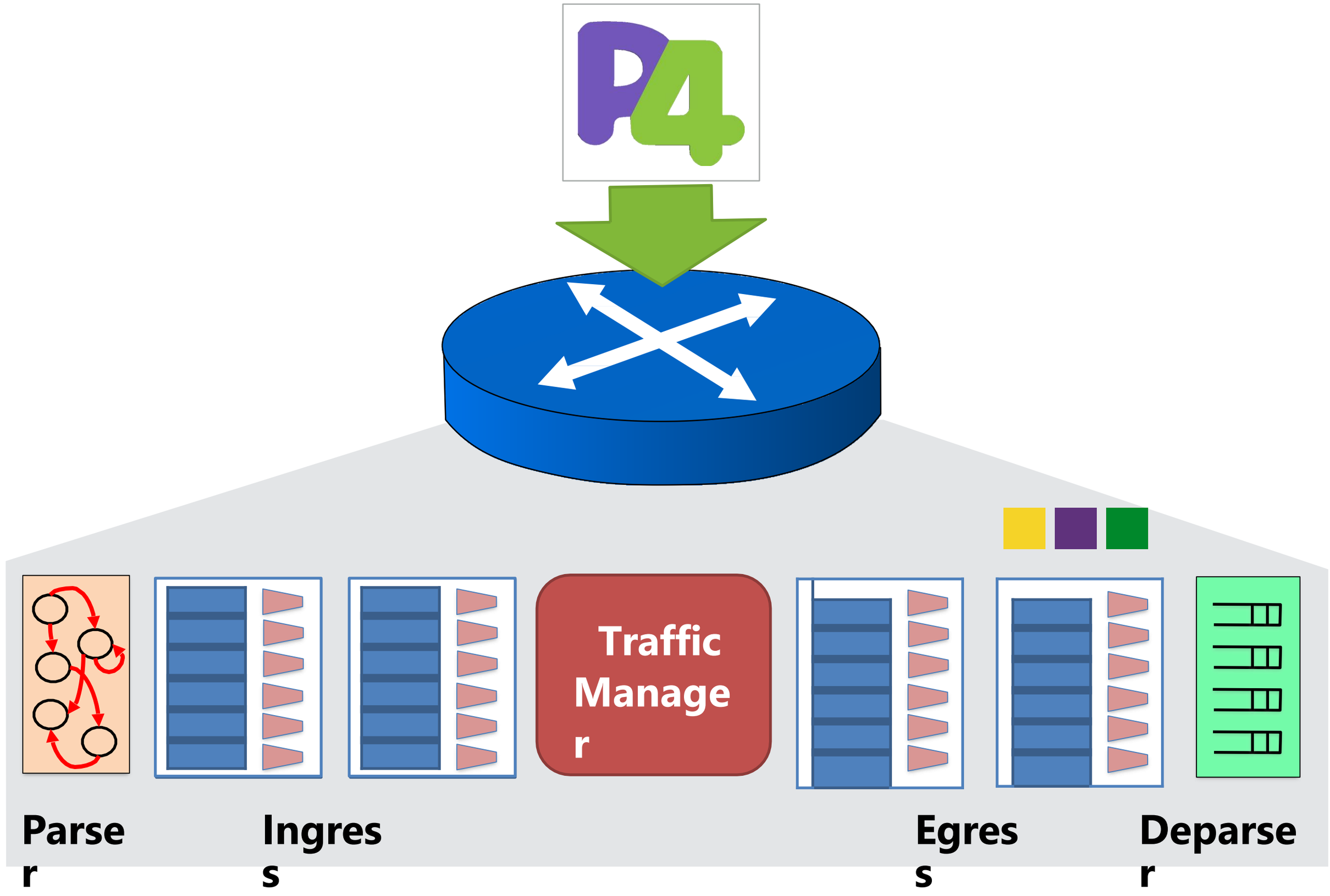

Traffic Manager
Parser
Ingress
Egress
Deparser

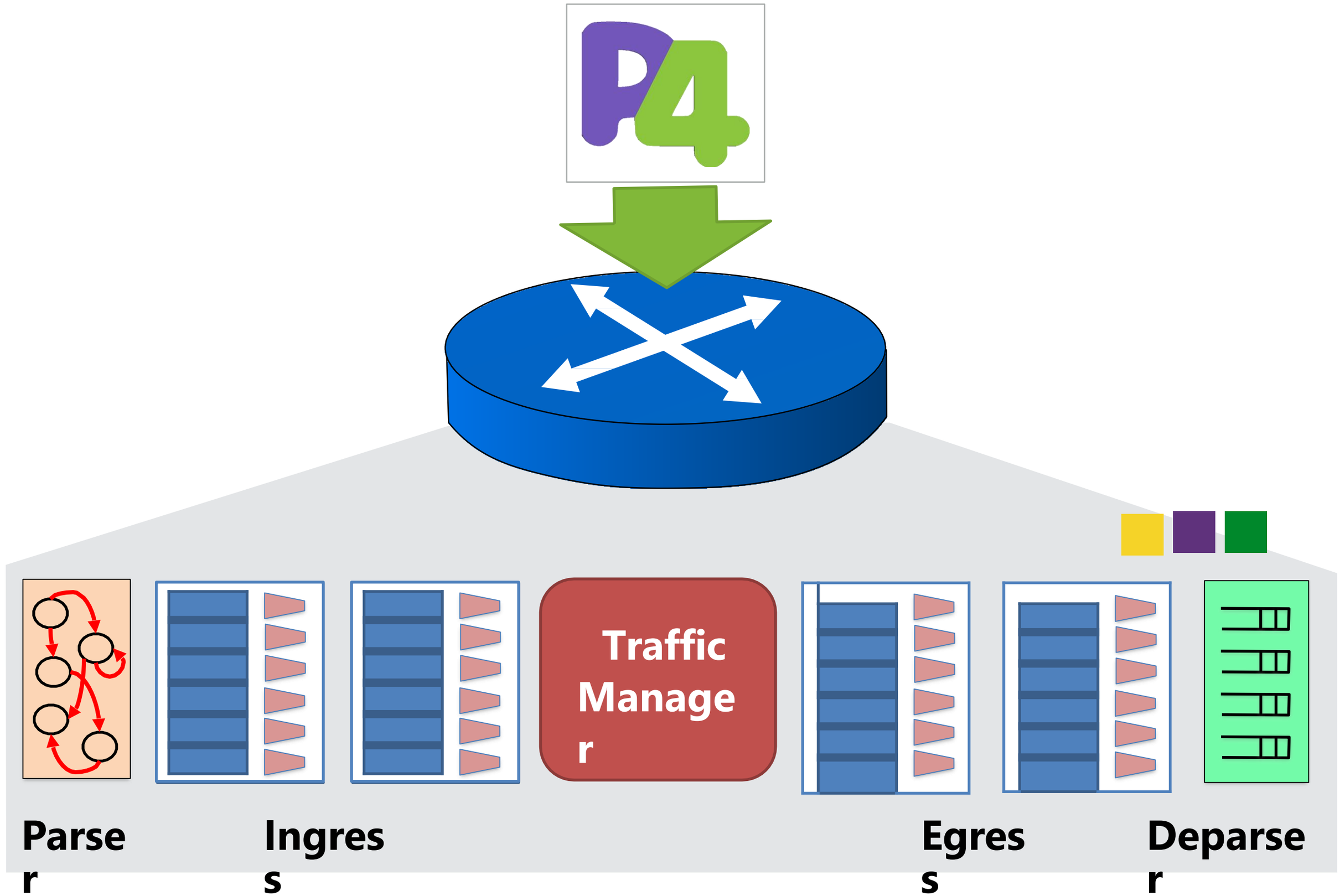

Traffic Manager
Parser
Ingress
Egress
Deparser

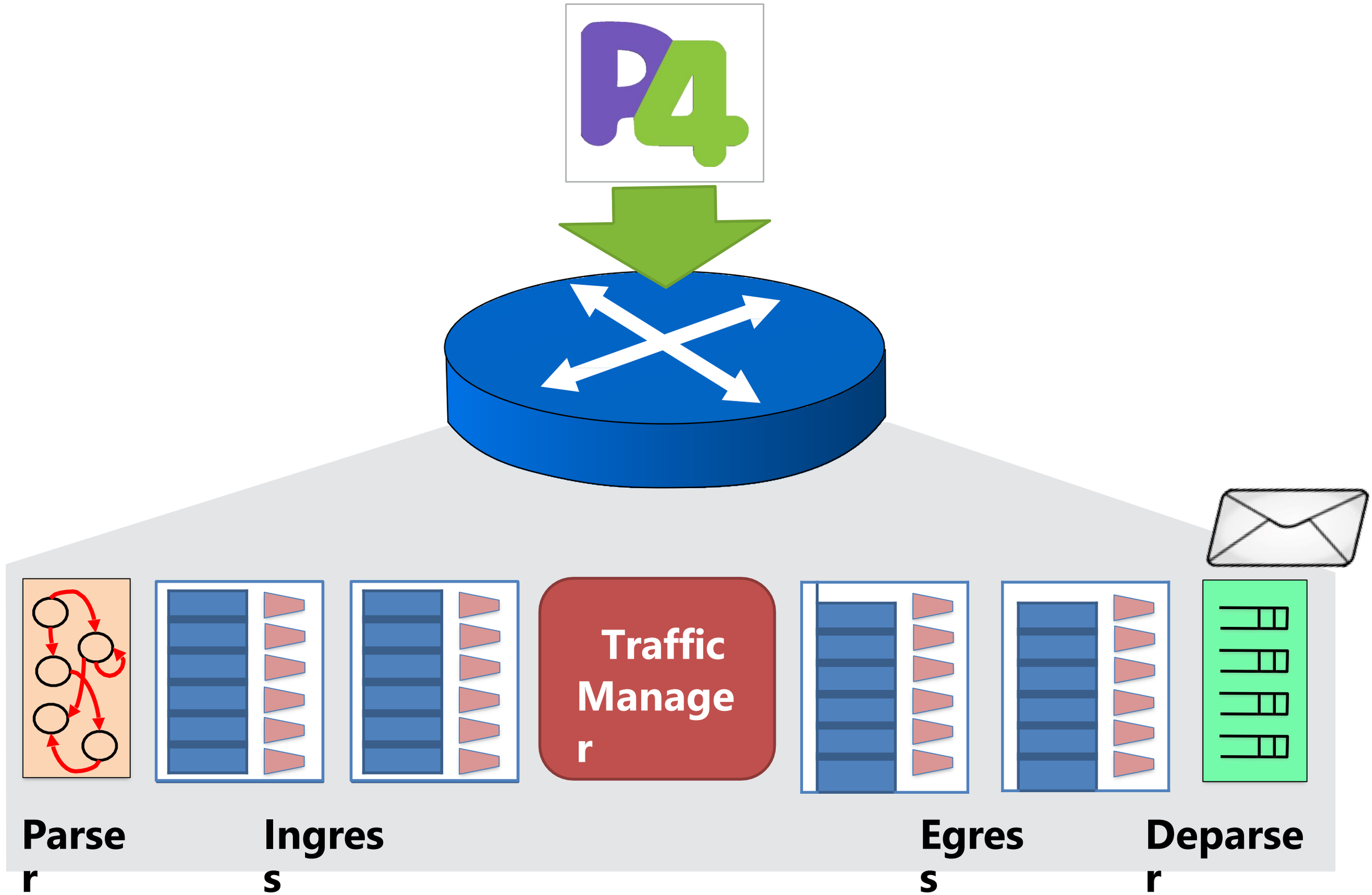

Traffic Manager
Parser
Ingress
Egress
Deparser


Traffic Manage r
Parse r
Ingres s
Egres s
Deparse r


Traffic Manager
Parser
Ingress
Egress
Deparser


Traffic Manager
Parser
Ingress
Egress
Deparser

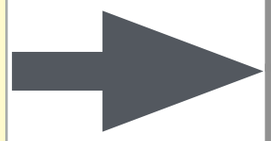

Traffic Manage r
Parse r
Ingres s
Egres s
Deparse r


Traffic Manager
Parser
Ingress
Egress
Deparser

- Slogan: "constant work in constant time"
  - No pointers or complex data types
  - Bounded state
  - No loops

- Key construct is a match-action table

```
action learn() {
  generate_digest(RECV, learn_digest);
}

table smac {
  reads { ethernet.srcAddr : exact; }
  actions { learn; nop; }
  default_action: nop;
}
```



Match	Action
00:00:00:00:00:01	learn
00:00:00:00:00:02	learn
*	nop

Example: Ethernet Switch

```
header_type ethernet_t {
  fields {
    dstAddr : 48;
    srcAddr : 48;
    etherType : 16;
  }
}

header_type intrinsic_metadata_t {
  fields {
    mcast_grp : 4;
    egress_rid : 4;
    mcast_hash : 16;
    lf_field_list: 32;
  }
}

header ethernet_t ethernet;
metadata intrinsic_metadata_t intrinsic_metadata;

parser start {
  return parse_ethernet;
}

parser parse_ethernet {
  extract(ethernet);
  return ingress;
}

field_list mac_learn_digest {
  ethernet.srcAddr;
  standard_metadata.ingress_port;
}

action mac_learn() {
  generate_digest(MAC_LEARN_RECEIVER, mac_learn_digest);
}

action forward(port) {
  modify_field(standard_metadata.egress_spec, port);
}

action broadcast() {
  modify_field(intrinsic_metadata.mcast_grp, 1);
}
```

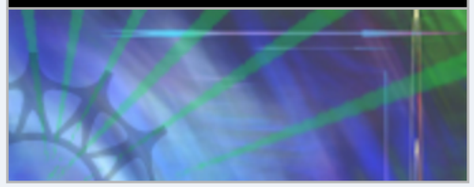
```
table smac {
  reads {
    ethernet.srcAddr : exact;
  }
  actions {
    mac_learn;
    nop;
  }
  size : 512;
}

table dmac {
  reads {
    ethernet.dstAddr : exact;
  }
  actions {
    forward;
    broadcast;
  }
  size : 512;
}

table mcast_src_pruning {
  reads {
    standard_metadata.instance_type : exact;
  }
  actions {_
    nop;
    drop;
  }
  size : 1;
}

control ingress {
  apply(smac);
  apply(dmac);
}

control egress {
  (if(standard_metadata.ingress_port ==
      standard_metadata.egress_port) {
    apply(mcast_src_pruning);
  }
}
```

National Science Foundation
WHERE DISCOVERIES BEGIN

SEARCH

HOME | RESEARCH AREAS | FUNDING | AWARDS | DOCUMENT LIBRARY | NEWS | ABOUT NSF

Awards



- Search Awards
- Recent Awards
- Presidential and Honorary Awards
- About Awards

How to Manage Your Award

- Grant Policy Manual
- Grant General Conditions
- Cooperative Agreement Conditions
- Special Conditions
- Federal Demonstration Partnership
- Policy Office Website

Award Abstract #1718036

SaTC: CORE: Small: Collaborative: A New Approach to Federated Network Security

NSF Org:	CNS Division Of Computer and Network Systems
Initial Amendment Date:	July 19, 2017
Latest Amendment Date:	July 19, 2017
Award Number:	1718036
Award Instrument:	Standard Grant
Program Manager:	Sol J. Greenspan CNS Division Of Computer and Network Systems CSE Direct For Computer & Info Scie & Enginr
Start Date:	September 1, 2017
End Date:	August 31, 2020 (Estimated)
Awarded Amount to Date:	$167,450.00
Investigator(s):	Hossein Hojjat hxhvcs@rit.edu (Principal Investigator)