

# 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



EPFL



Cornell

# Formal Methods & Programming Languages



Rochester Institute of Technology

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



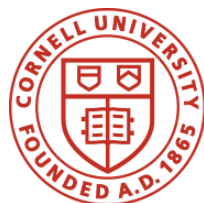
Jedidiah McClurgh



Pavol Cerny



Nate Foster



Philipp Rümmer



# Conventional Networking

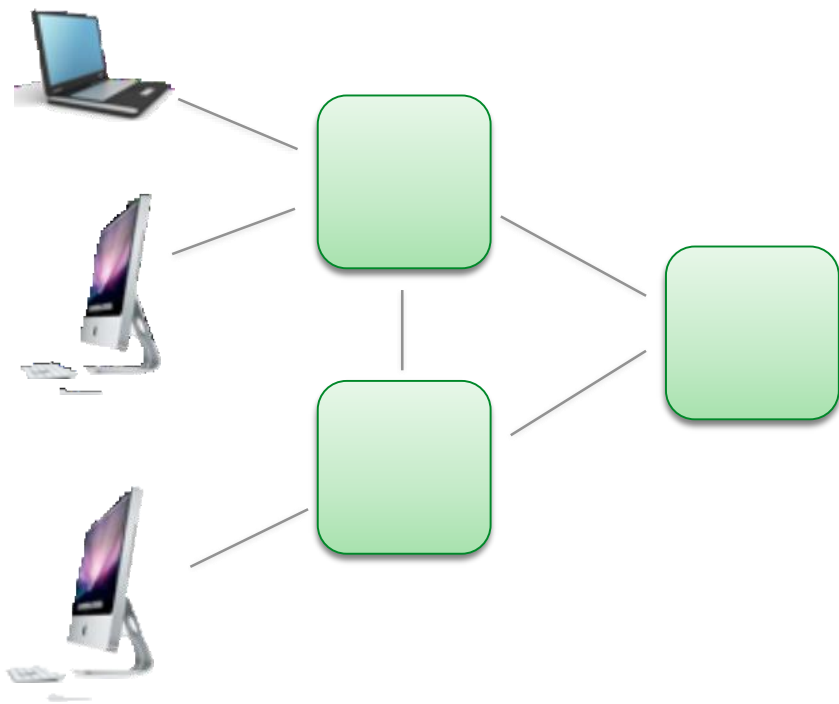
There are  
hosts...





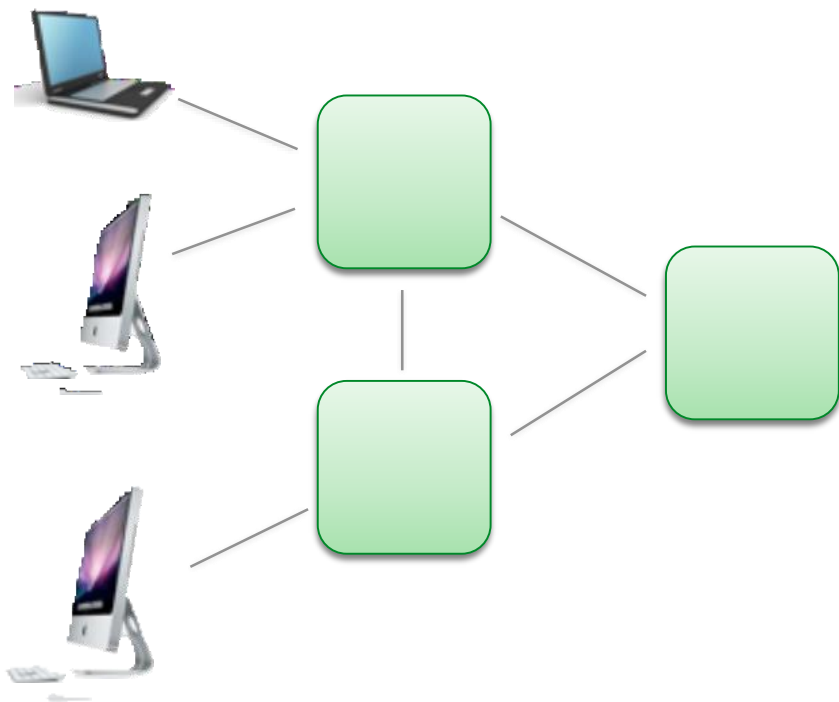
# Conventional Networking

Connected by  
switches...



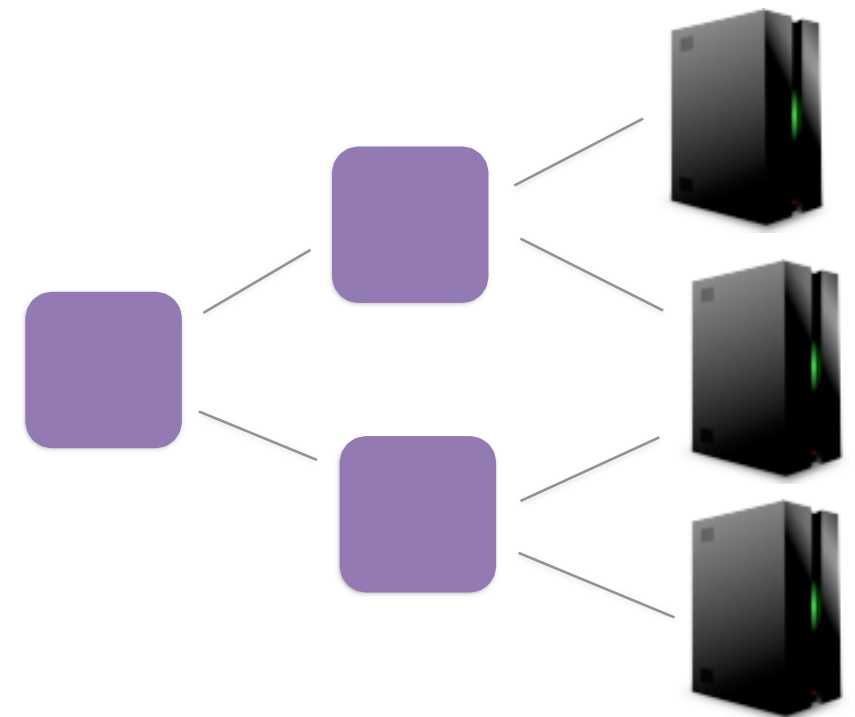
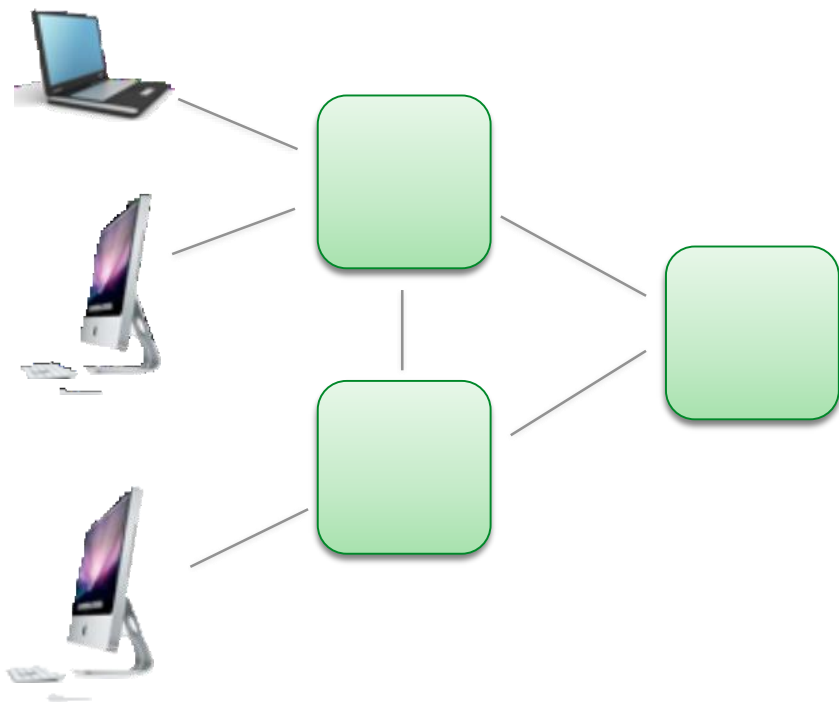
# Conventional Networking

There are also  
servers...



# Conventional Networking

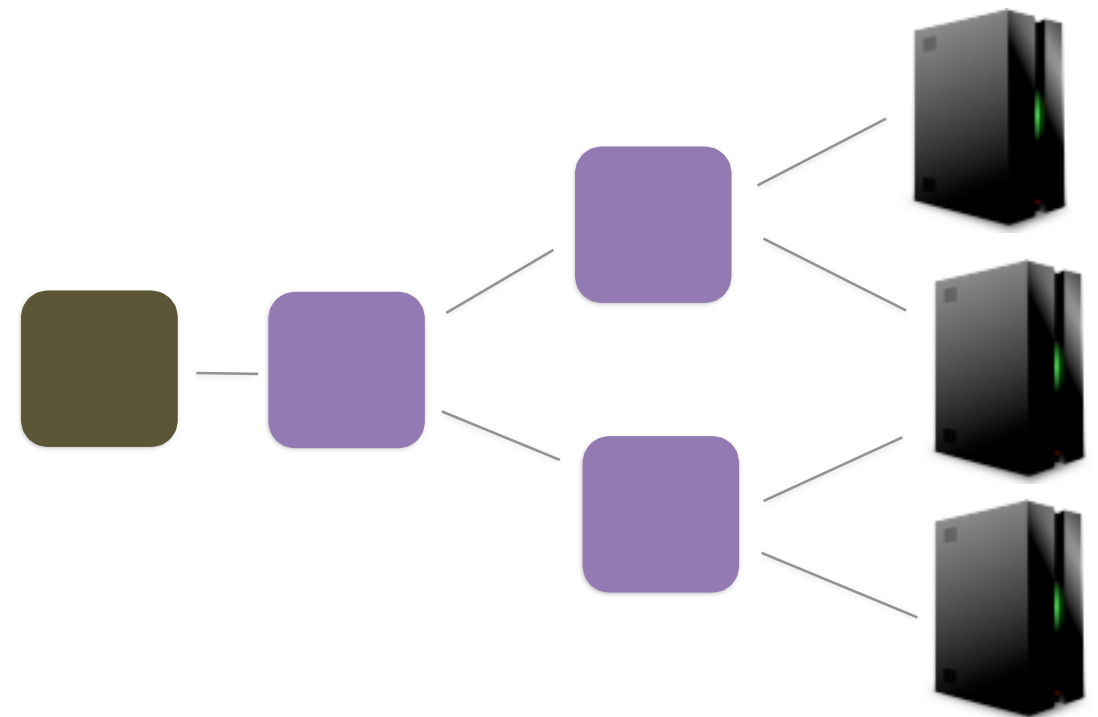
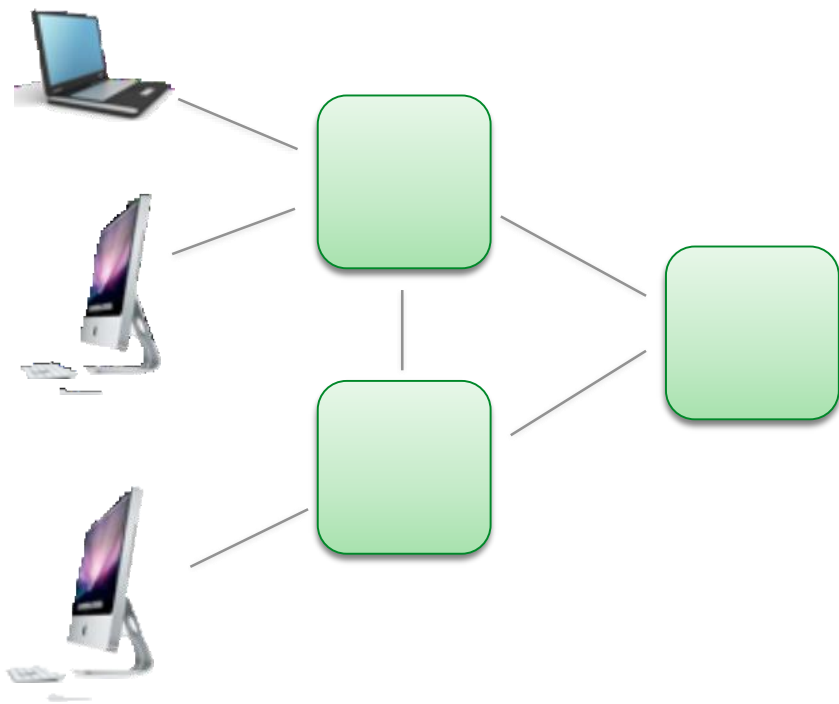
Connected by  
routers...





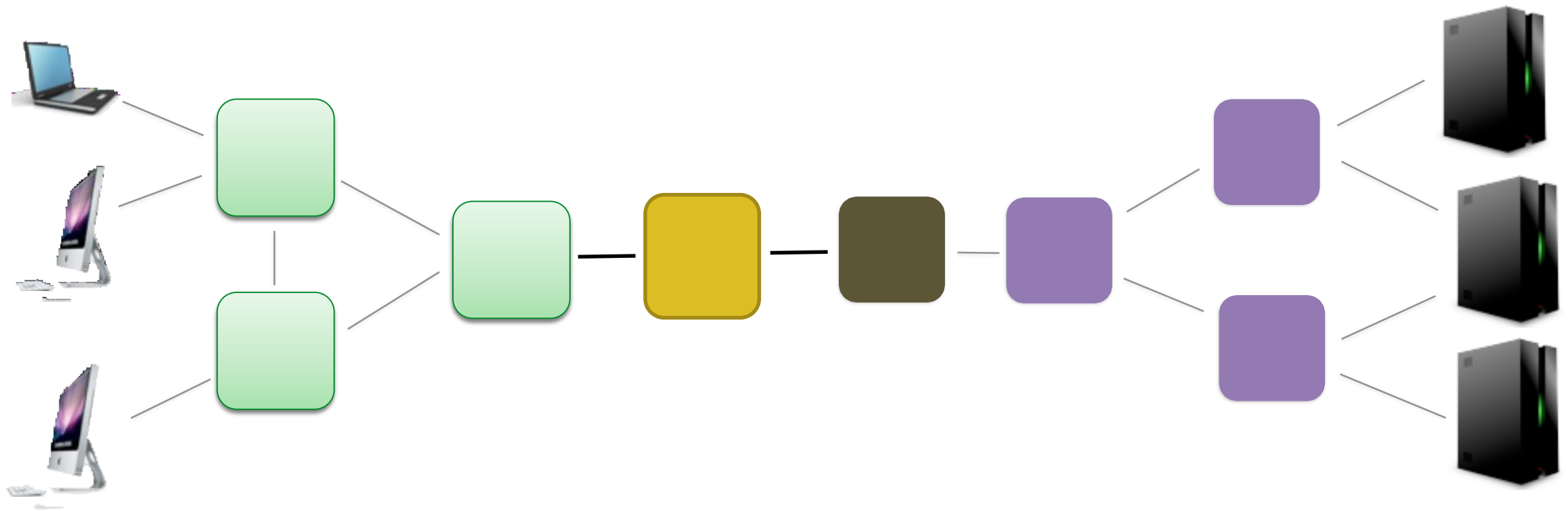
# Conventional Networking

And a load balancer...



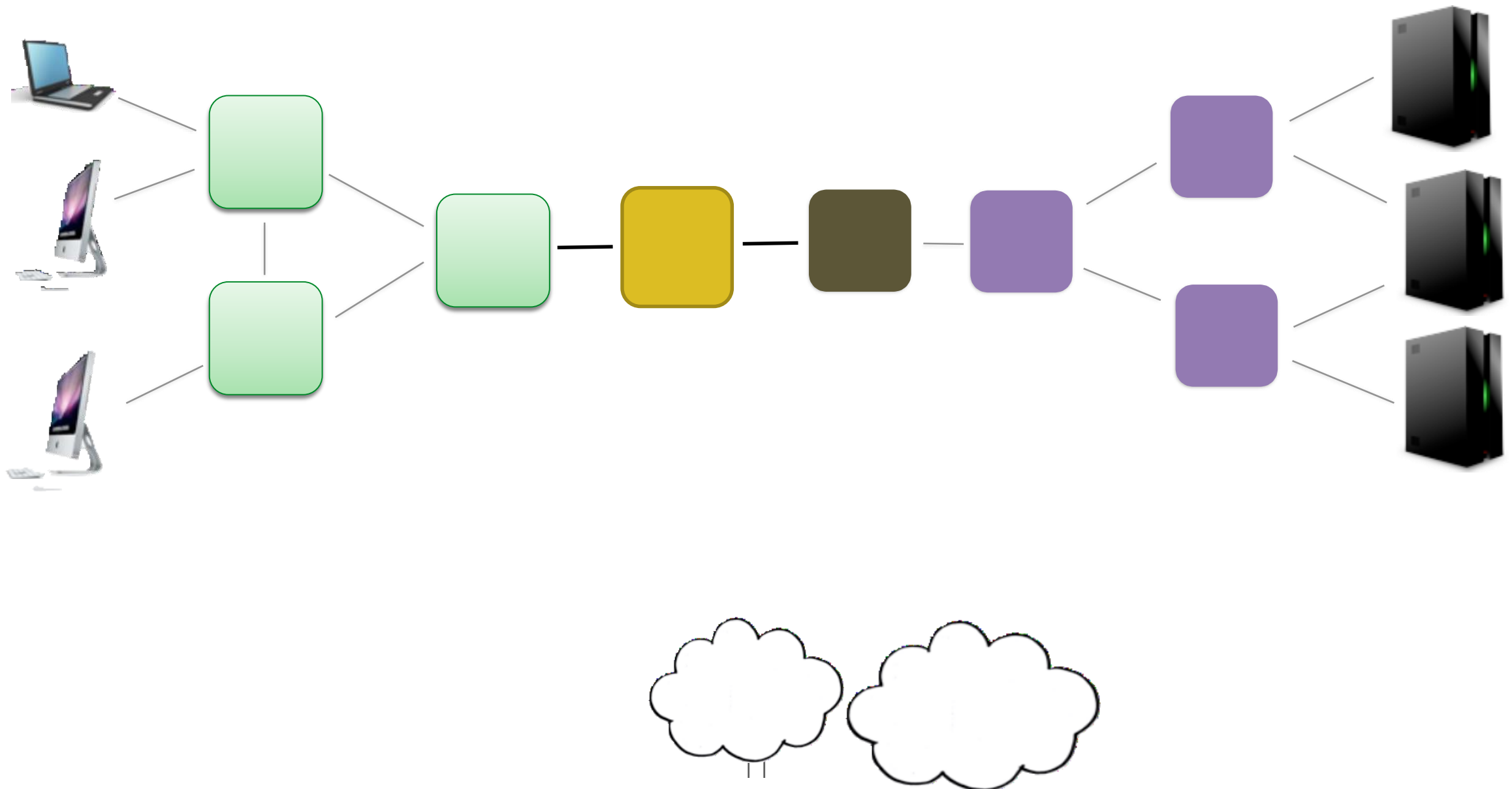
# Conventional Networking

And a gateway  
router...



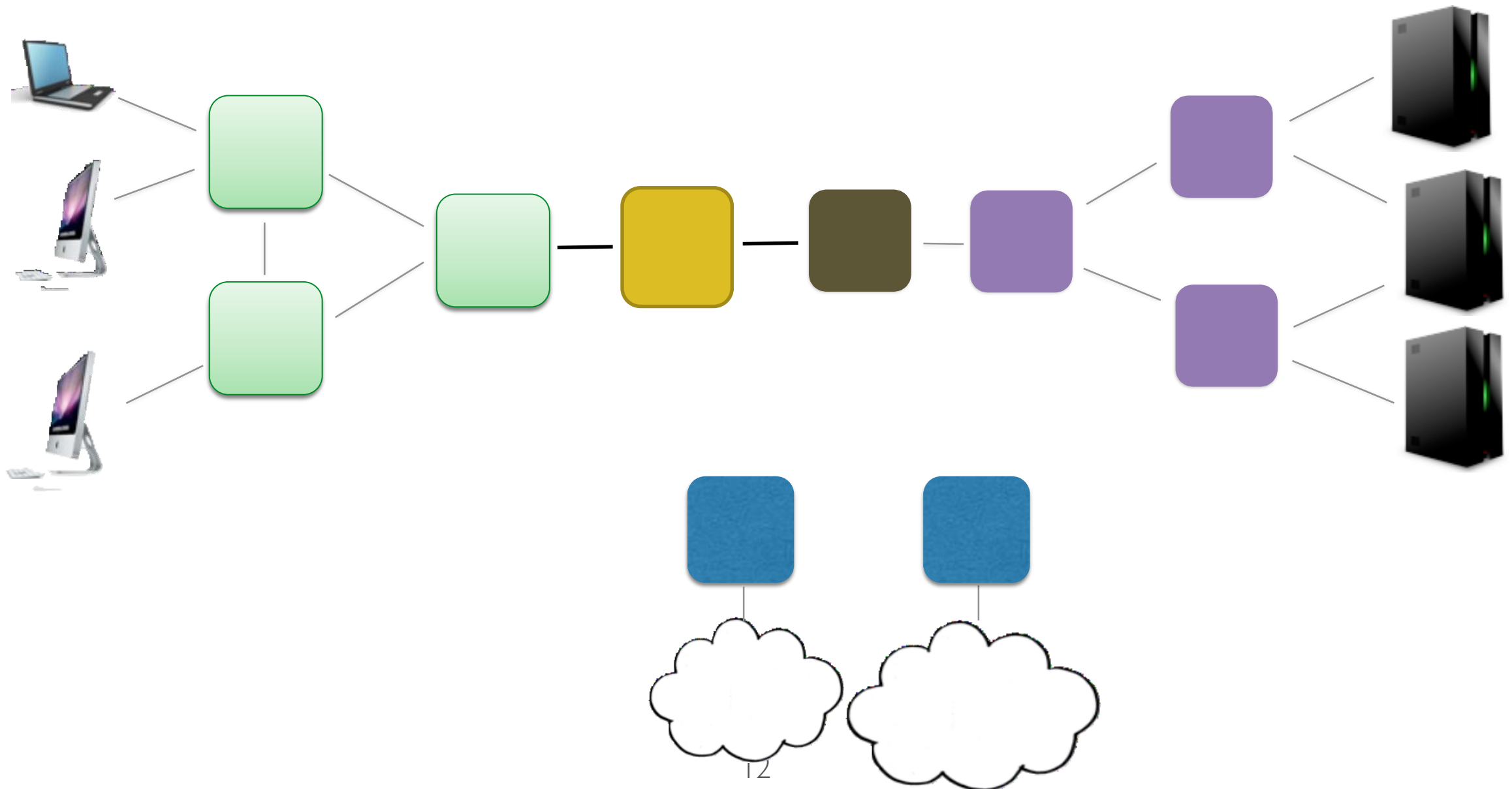
# Conventional Networking

There are other  
ISPs...



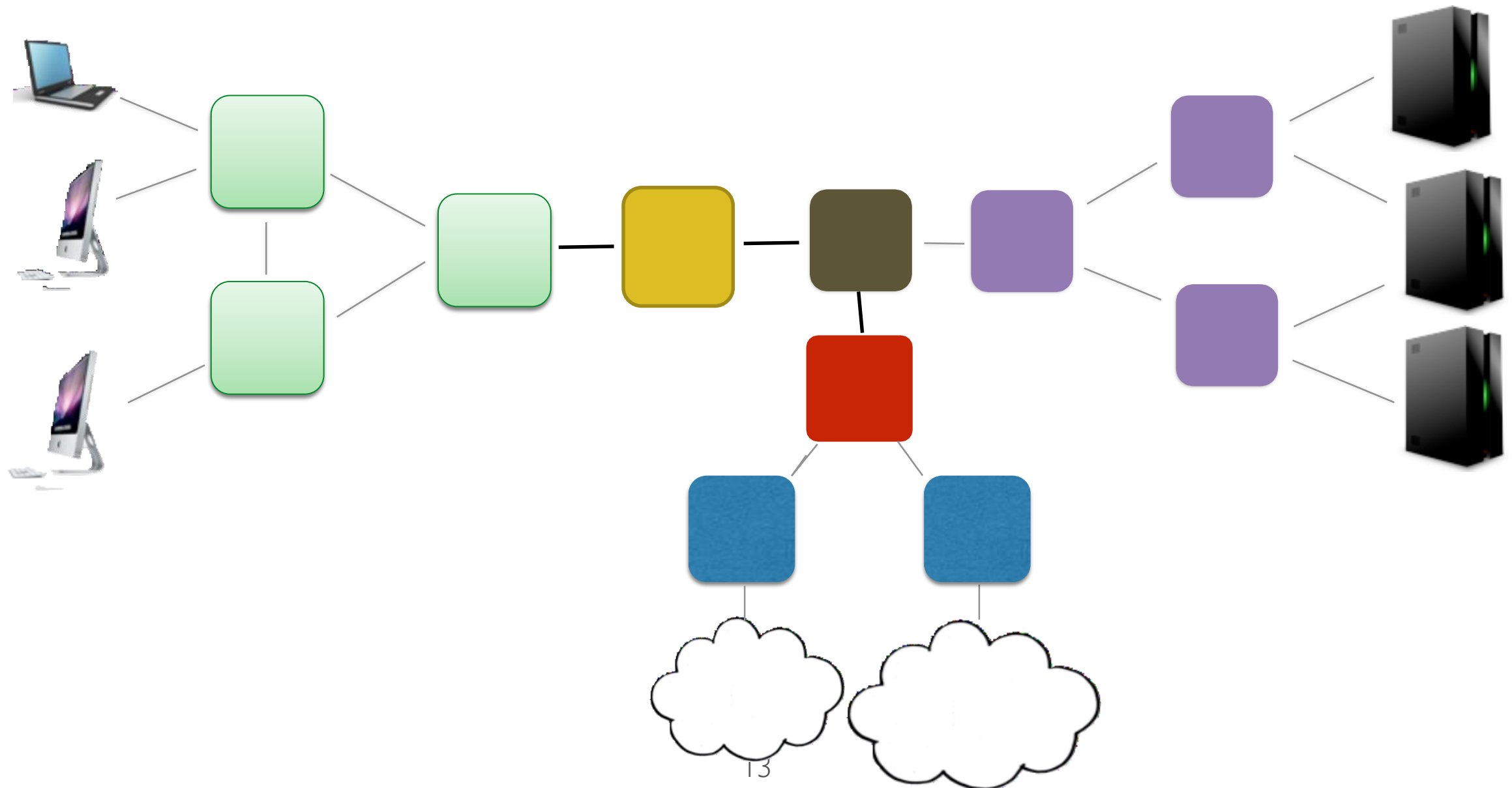
# Conventional Networking

So we need to run  
BGP...



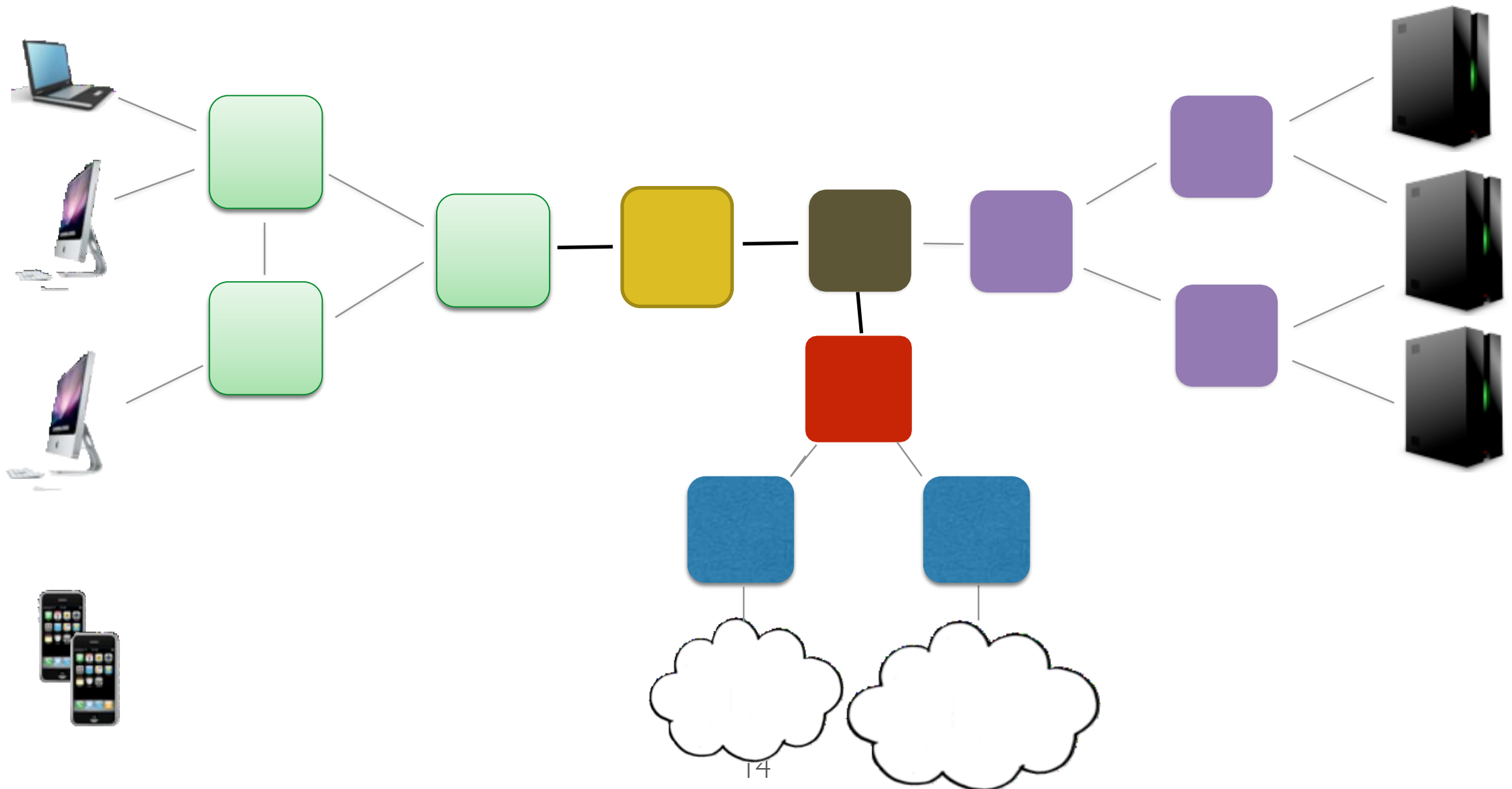
# Conventional Networking

And we need a firewall to filter incoming traffic...



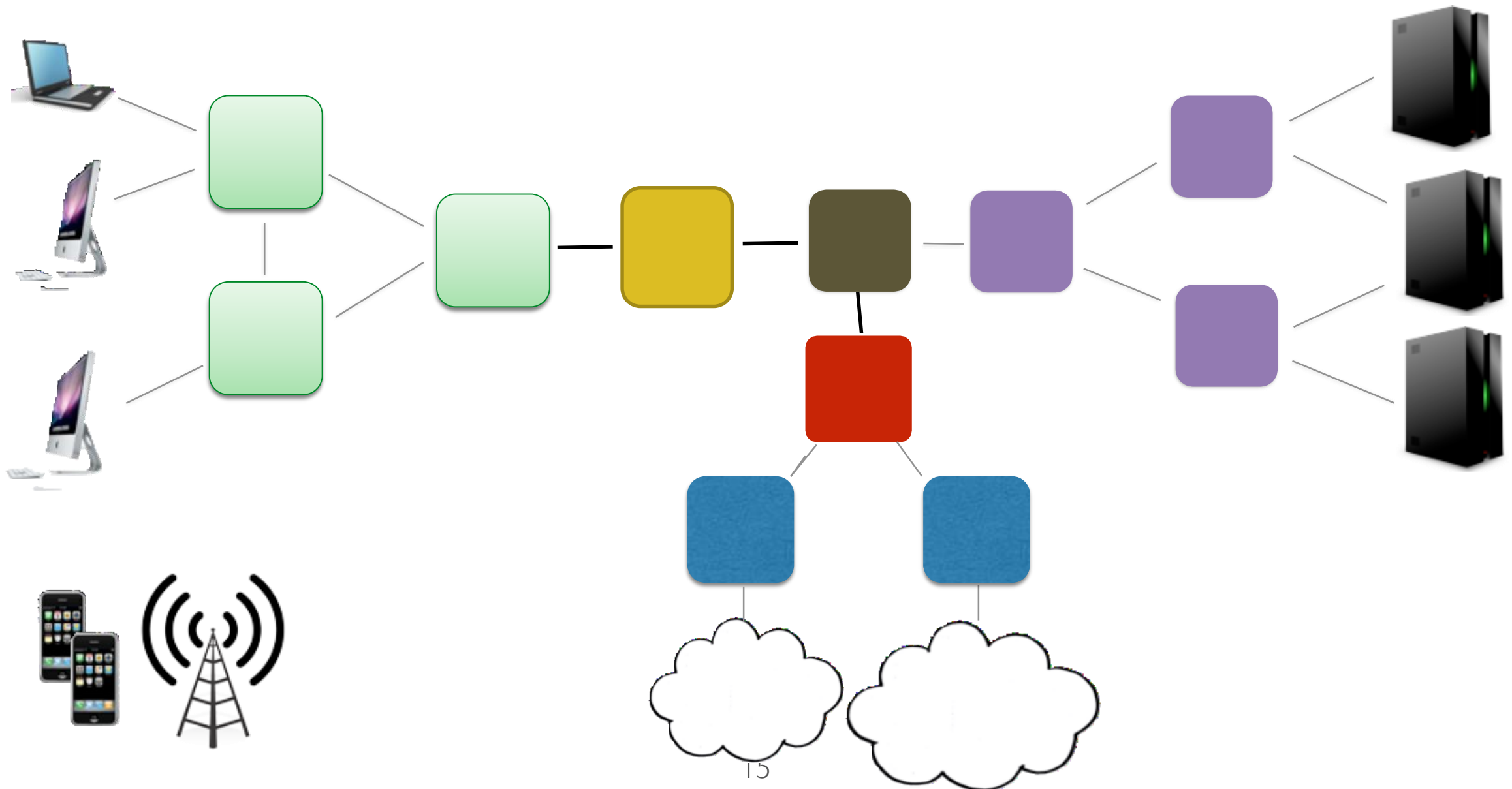
# Conventional Networking

There are also wireless  
hosts...



# Conventional Networking

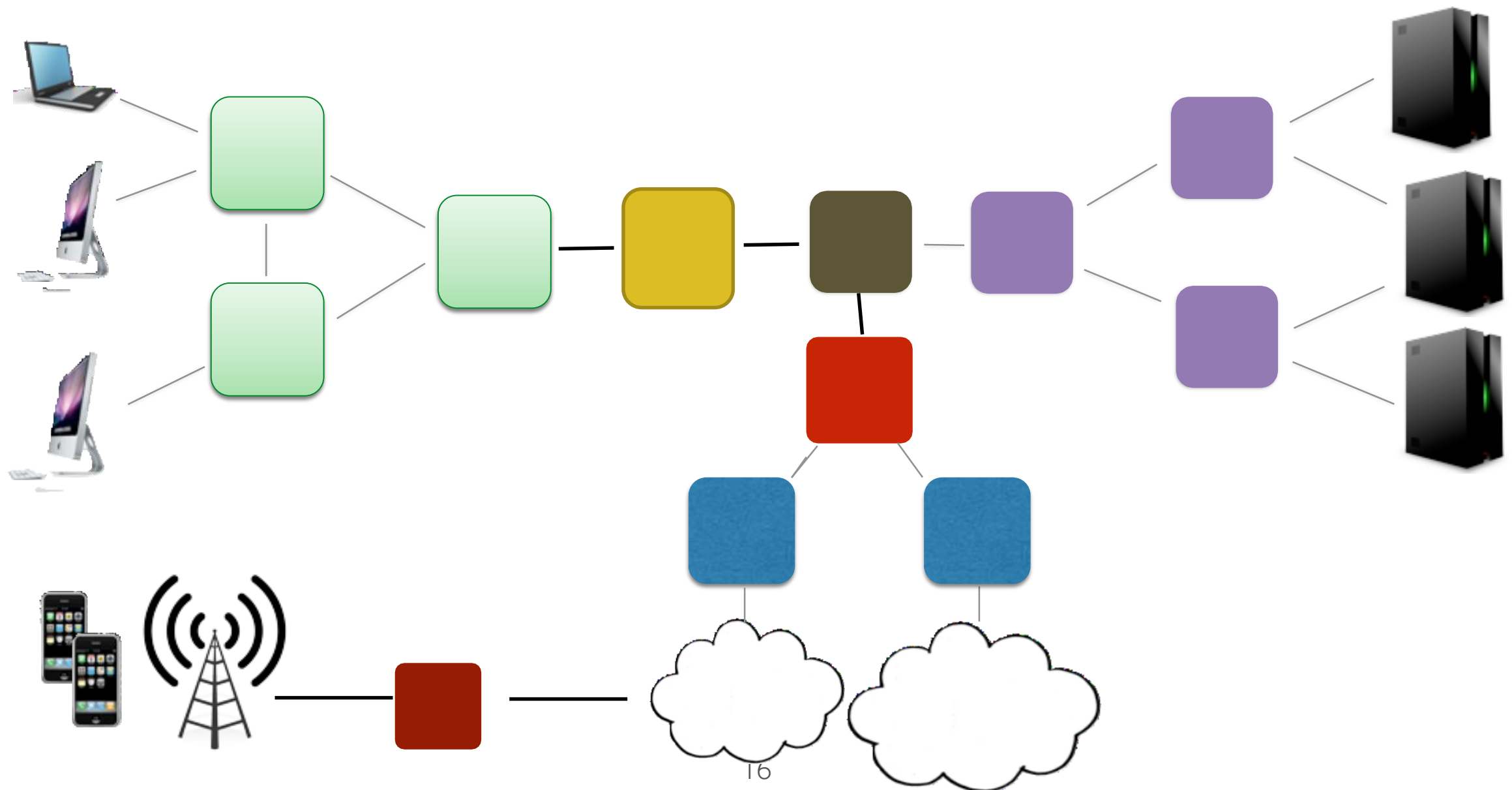
So we need wireless gateways...





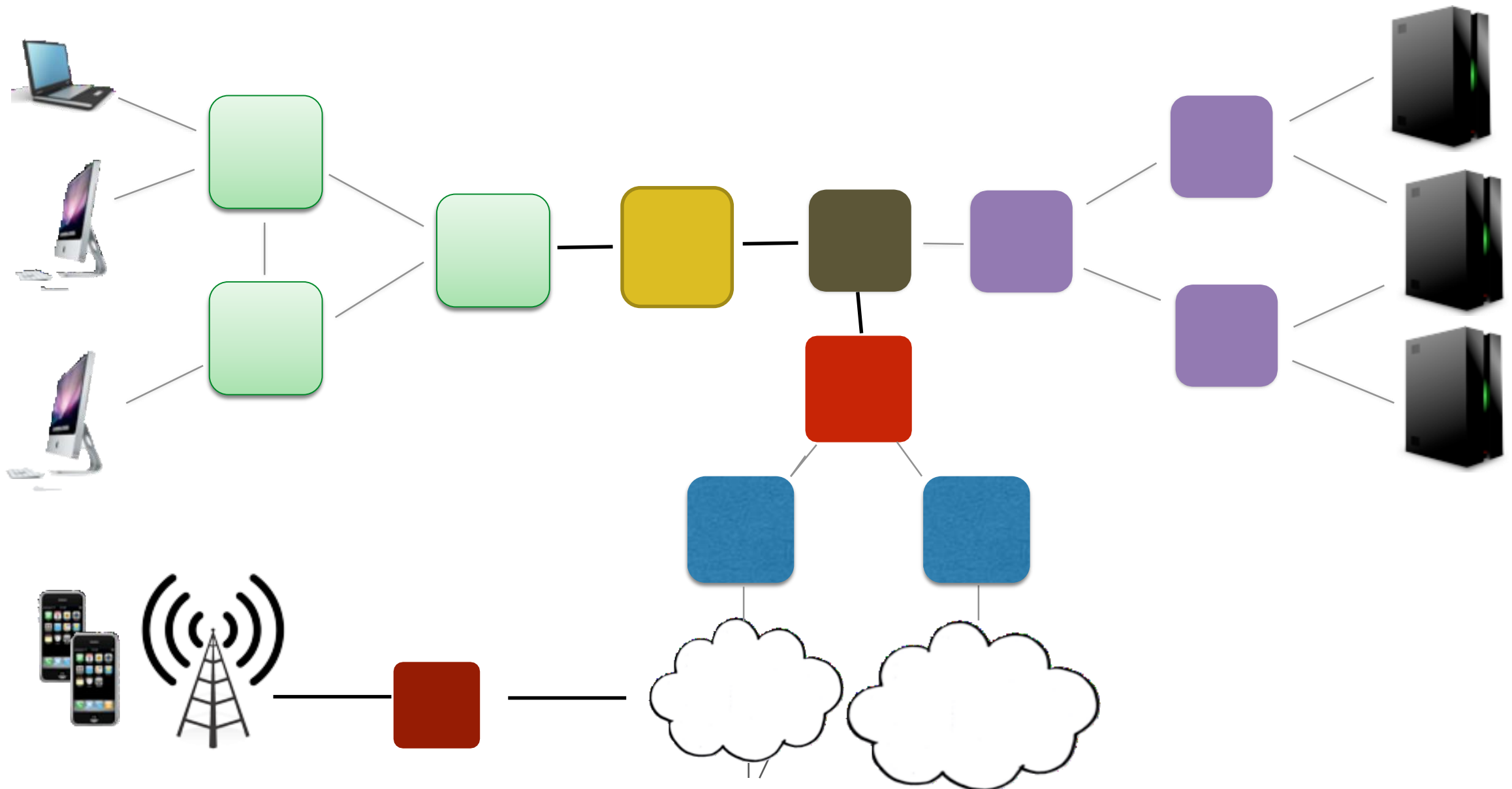
# Conventional Networking

And yet more middleboxes for lawful intercept...



# Conventional Networking

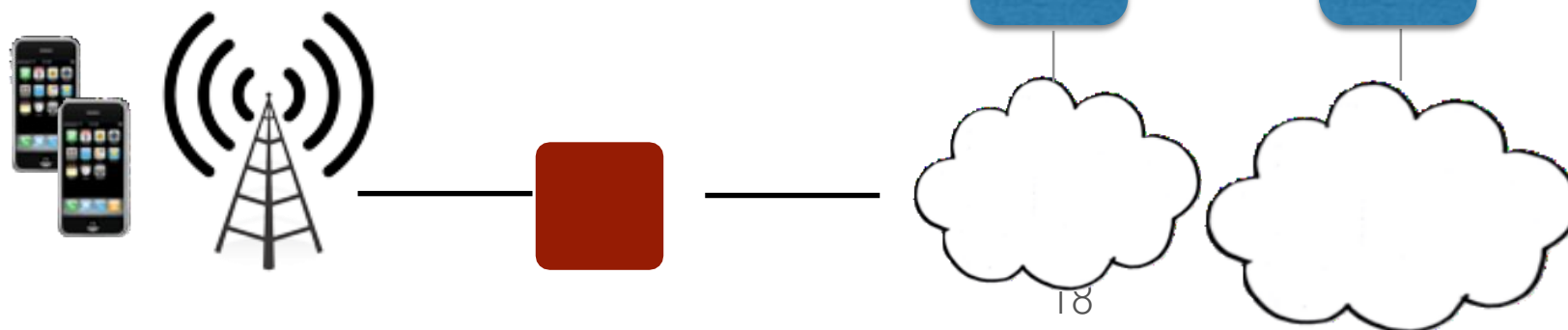
Each color represents a different set of control plane protocols and algorithms...



# 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

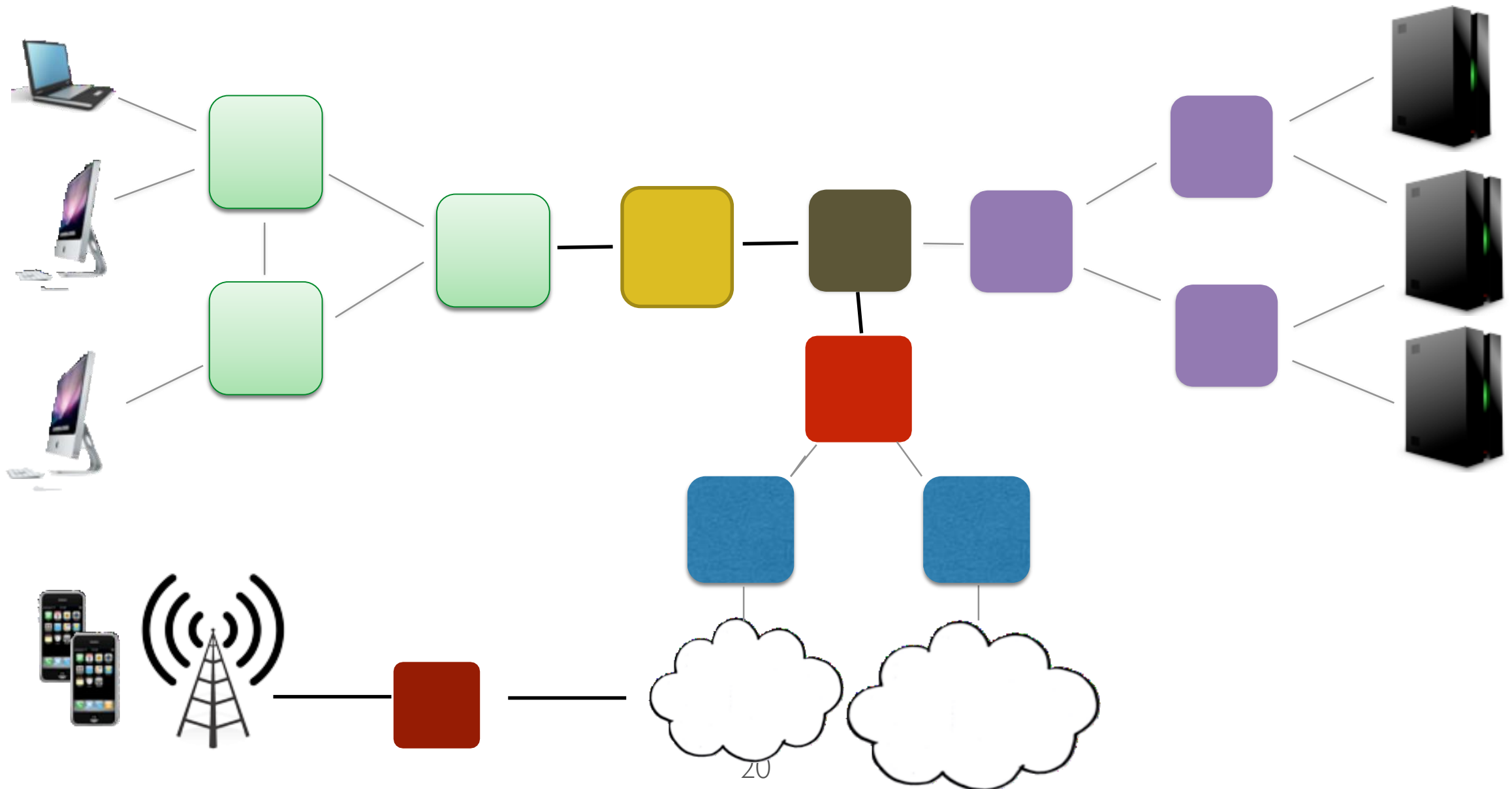


یکی از روترها [...] به دلیل باگی که در نرم افزار مودم وجود داشت به روزرسانی نشد و یک فرد از این آسیب پذیری برای نفوذ استفاده کرد

# Software-Defined Networking

A clean-slate architecture based on two key ideas:

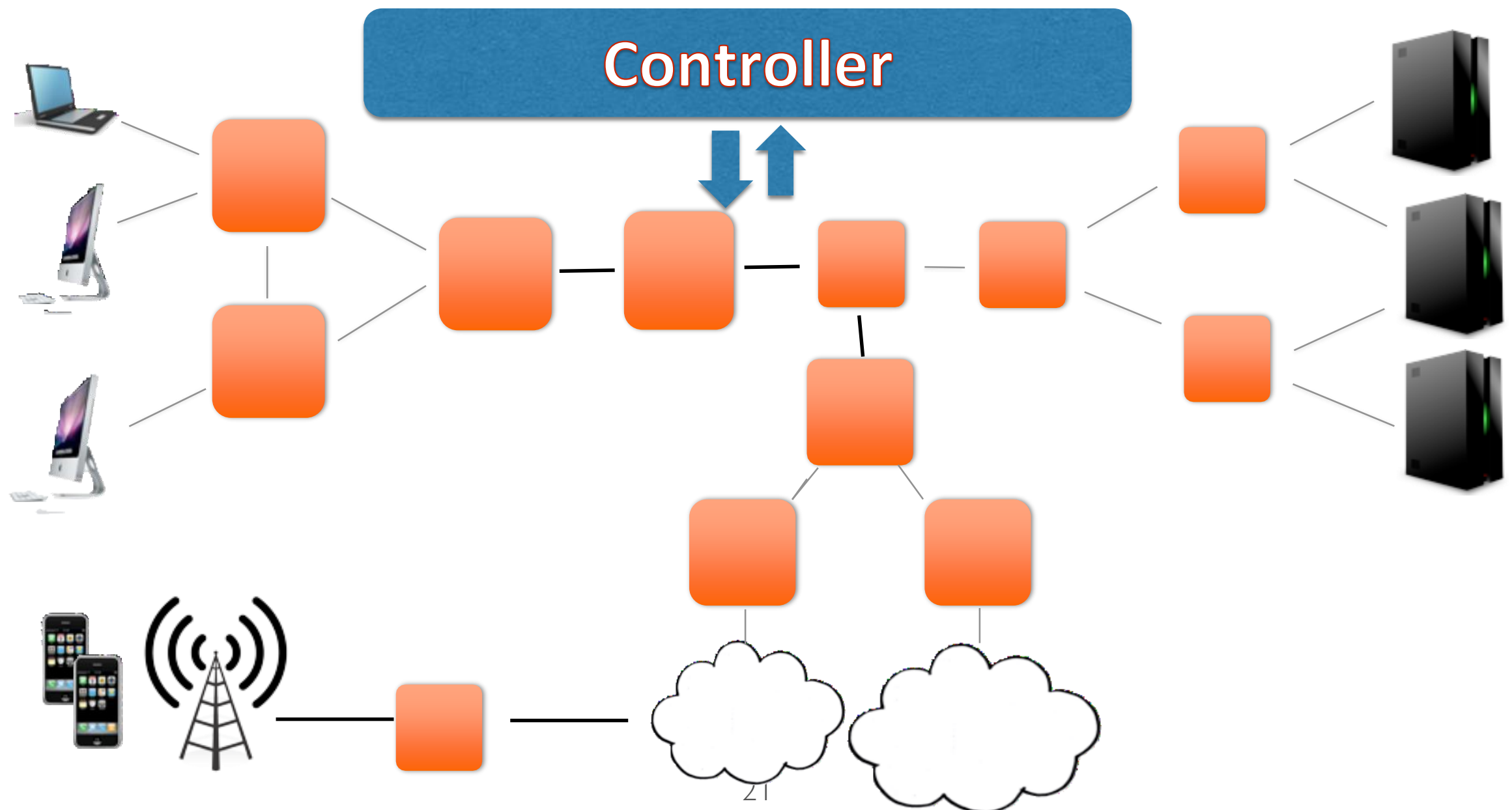
- Generalize network devices
- Separate control and forwarding



# Software-Defined Networking

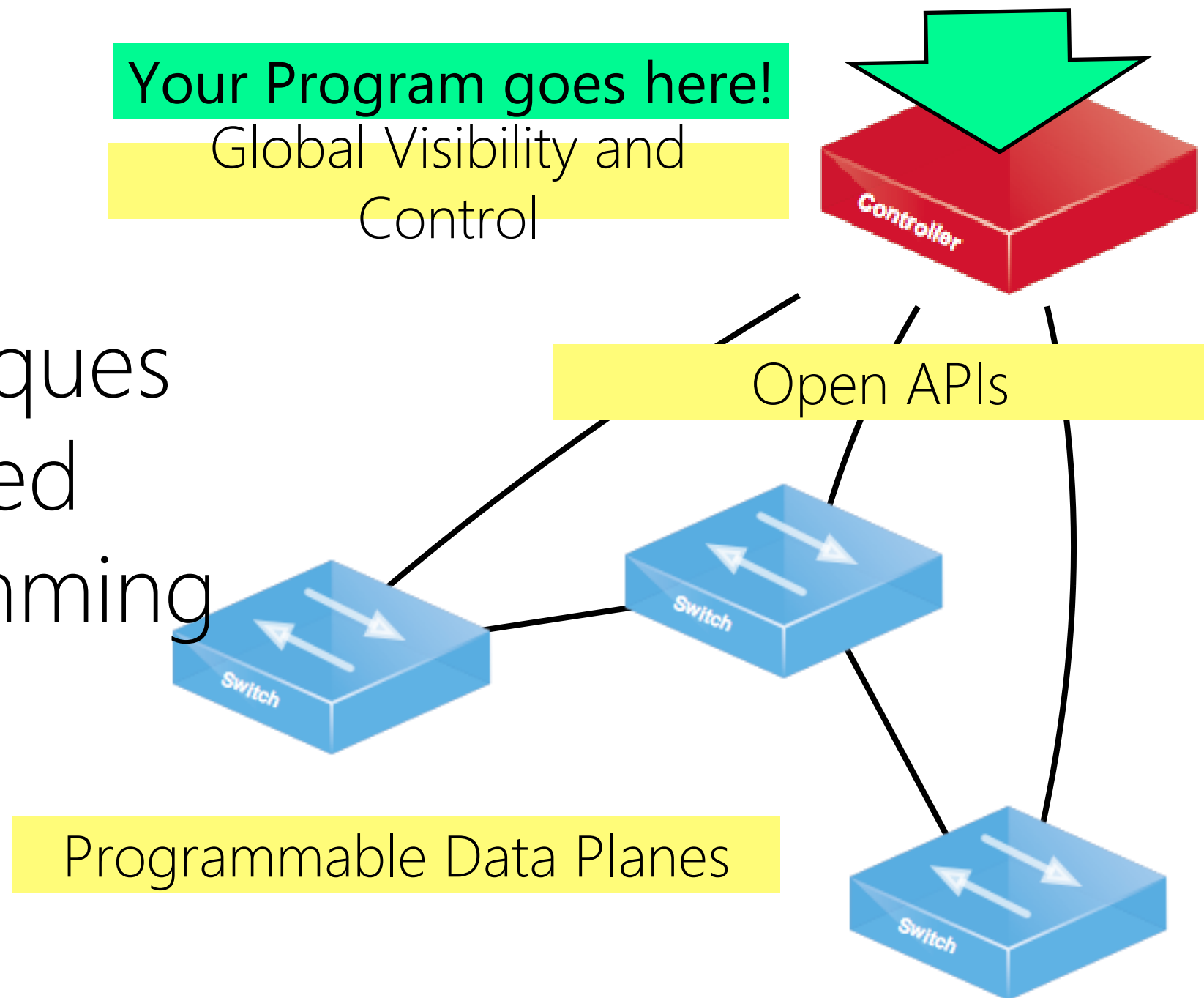
A clean-slate architecture based on two key ideas:

- 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?





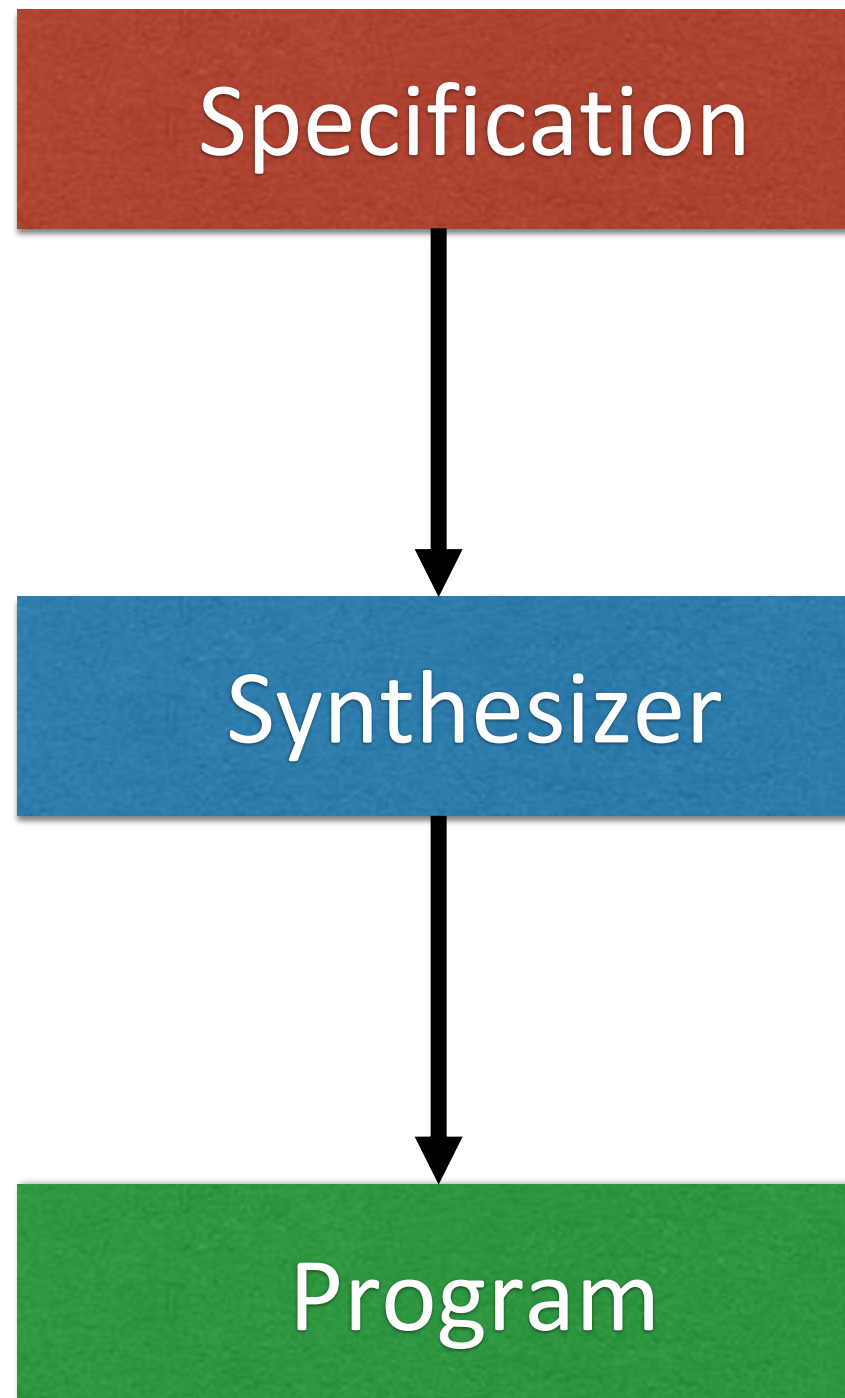
# Software Synthesis

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

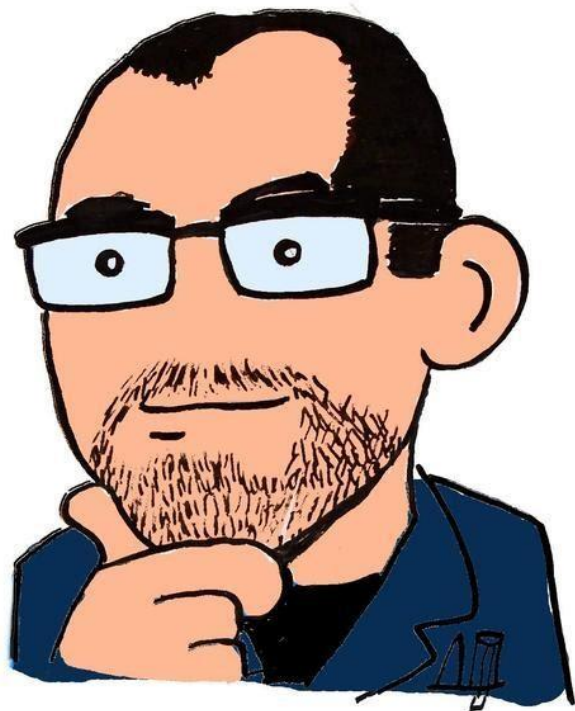
Partial  
Program

Logical  
Formula

Input-Output  
Examples

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

Program



- 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

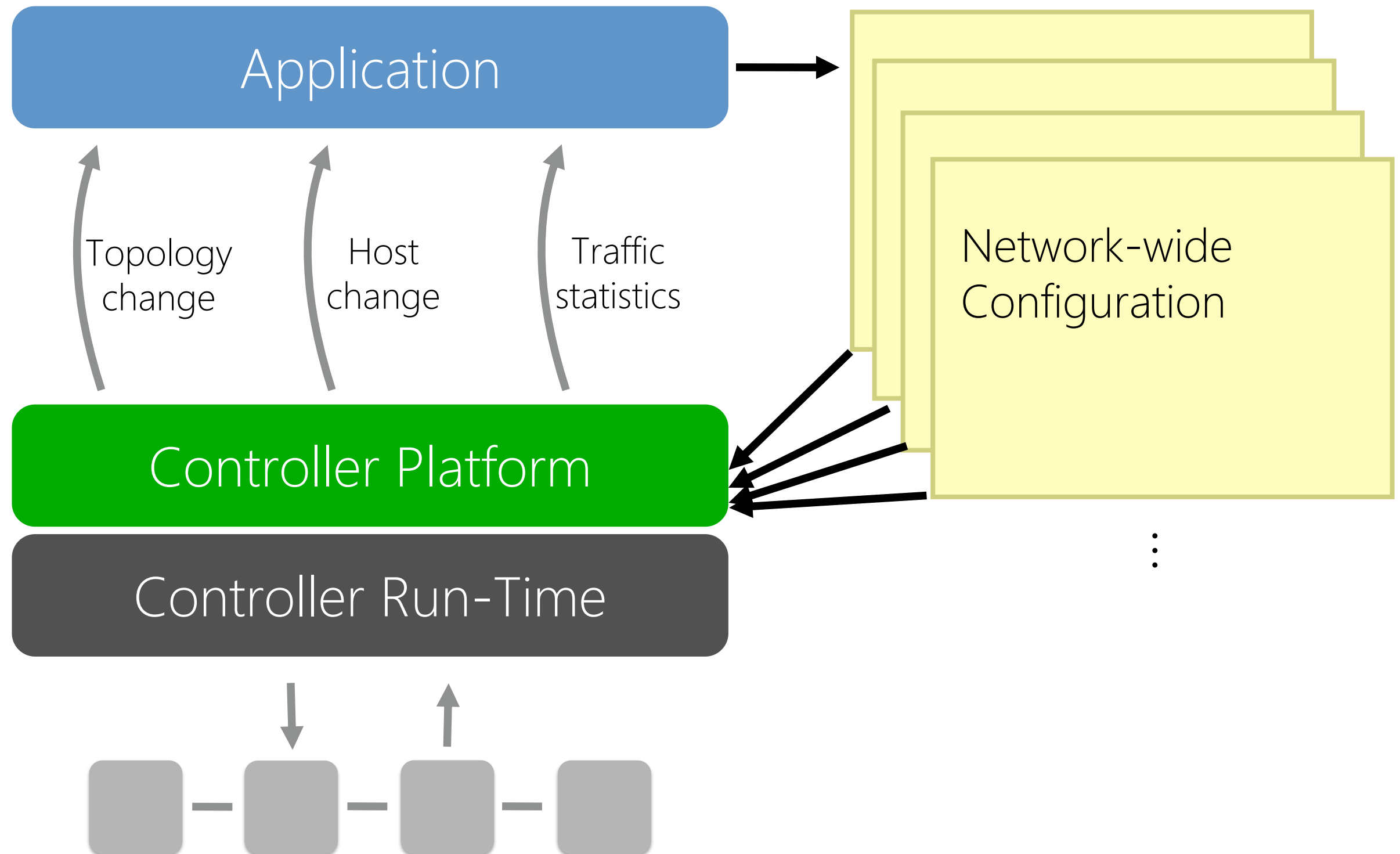
## Outline:

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



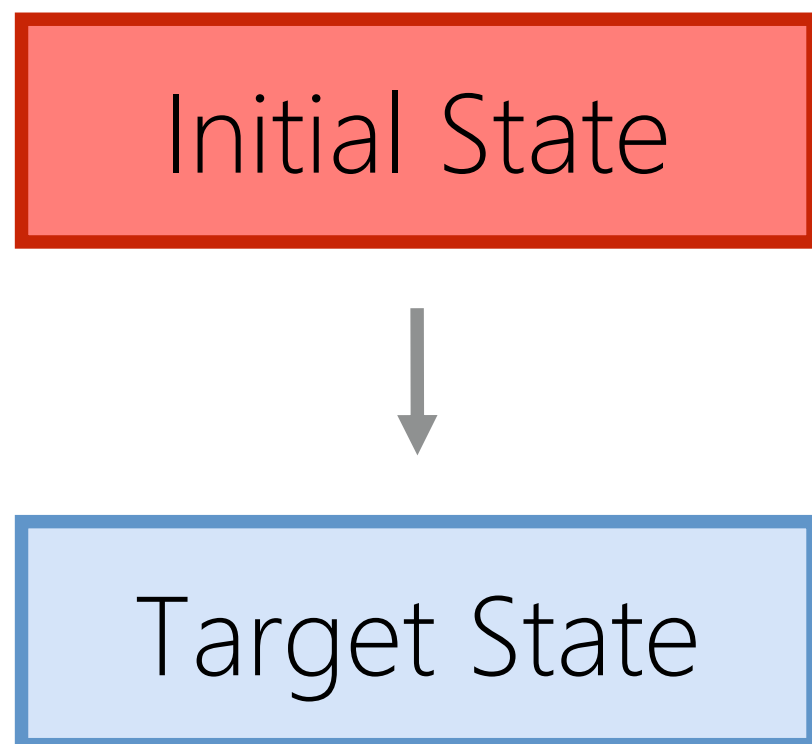
# Efficient Synthesis of Network Updates

# Dynamic SDN Applications

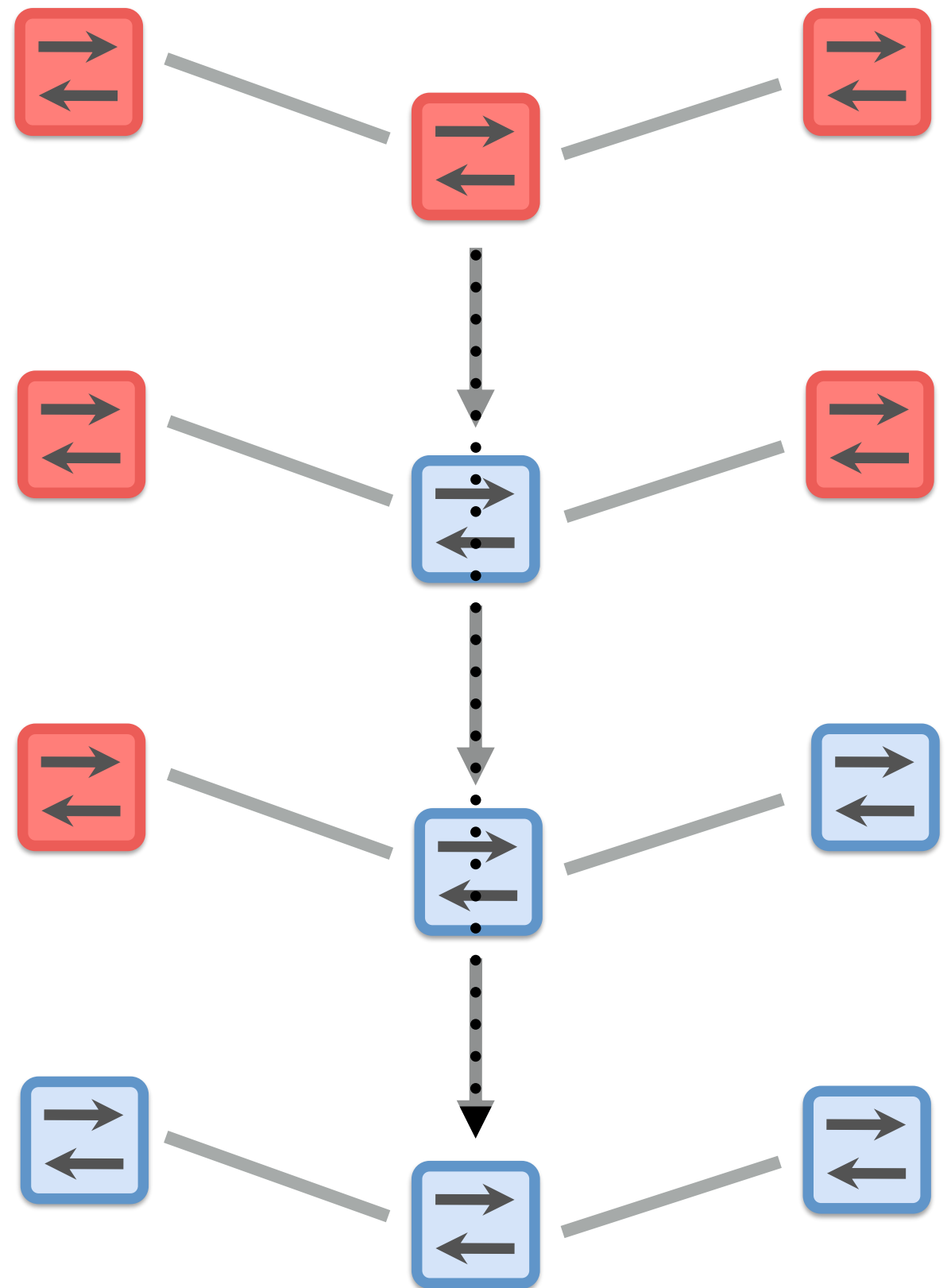


# Network Updates

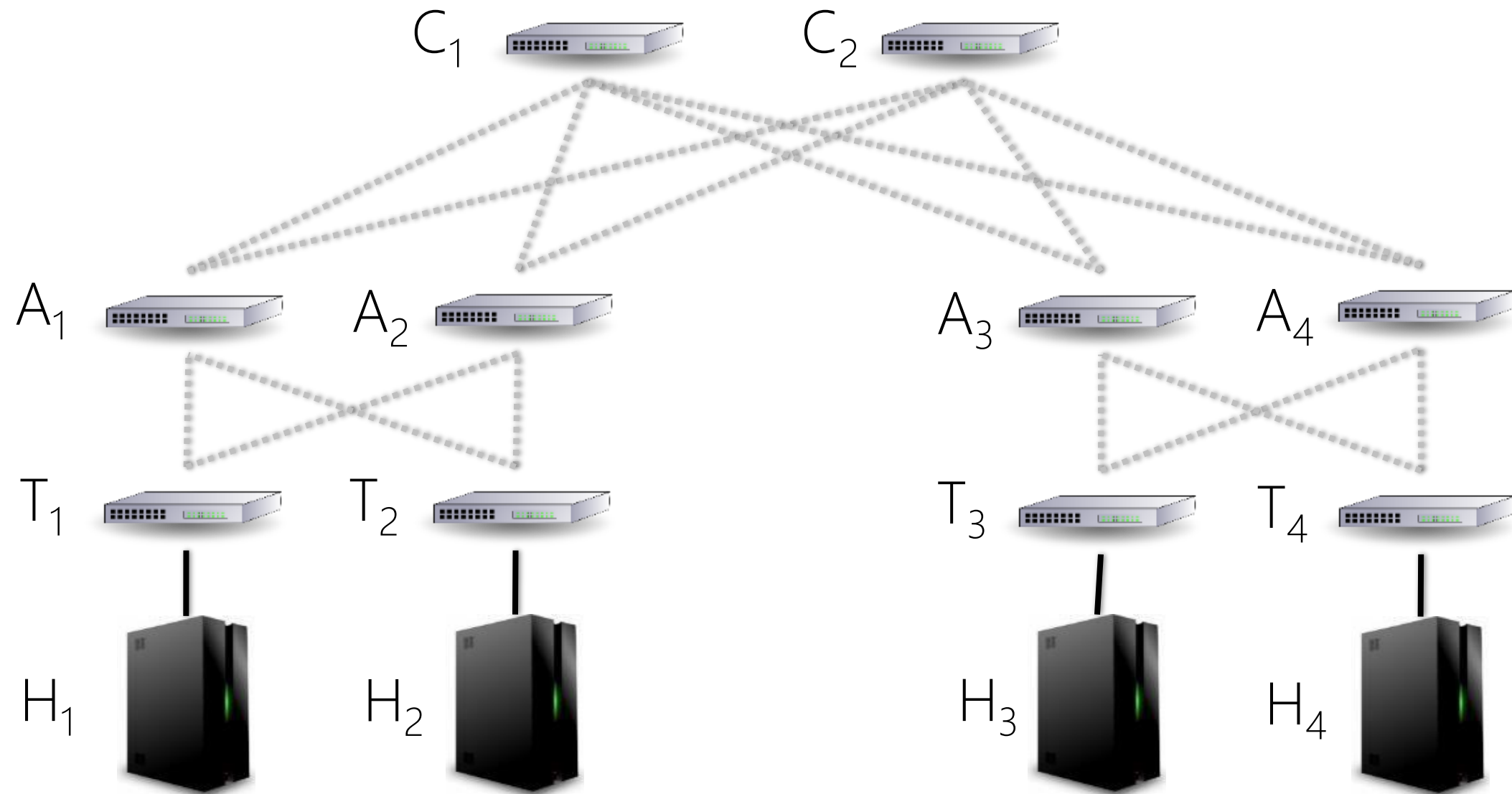
How can we transition between global states?



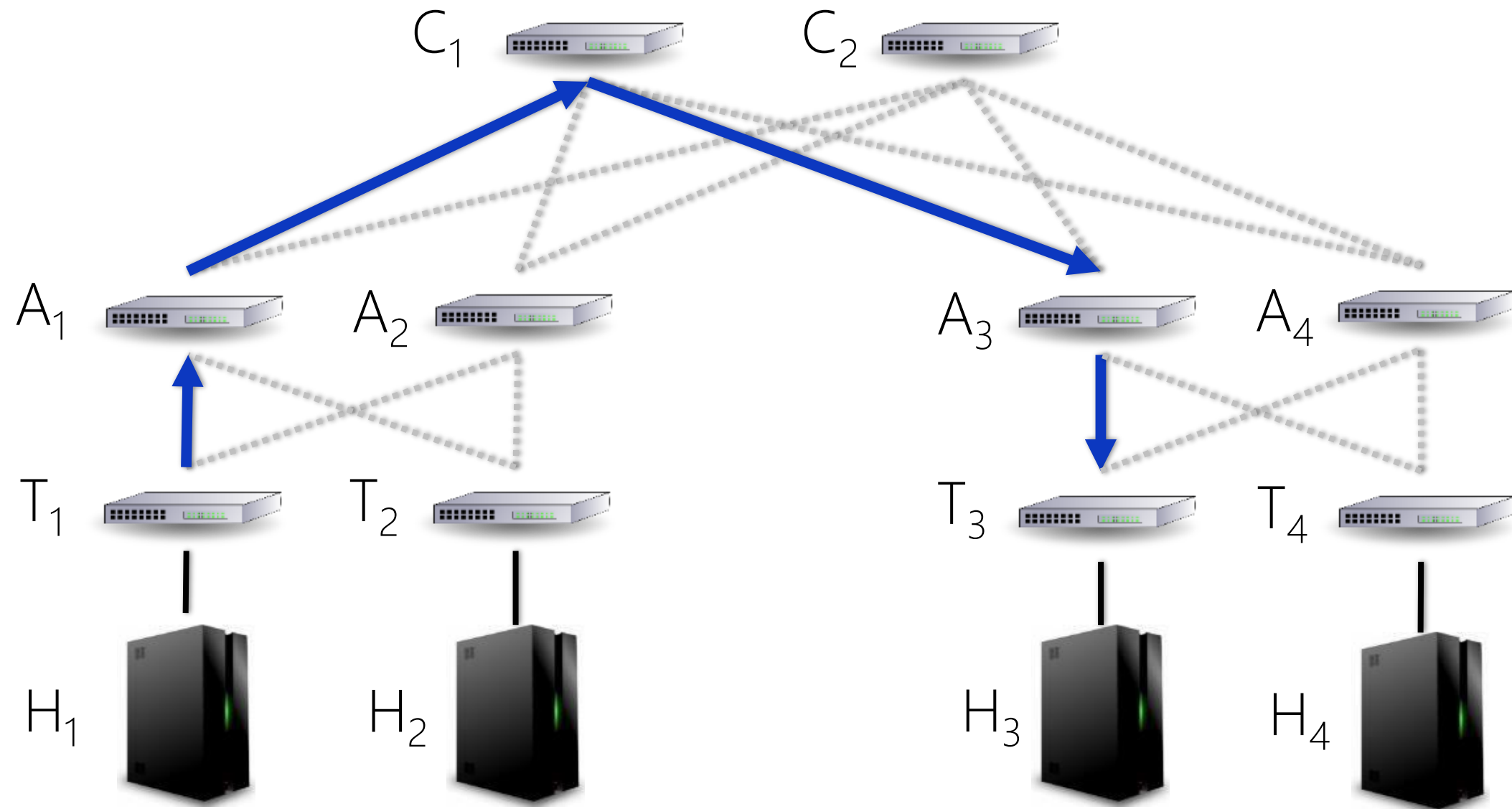
**Problem:** naive updates can break important invariants!



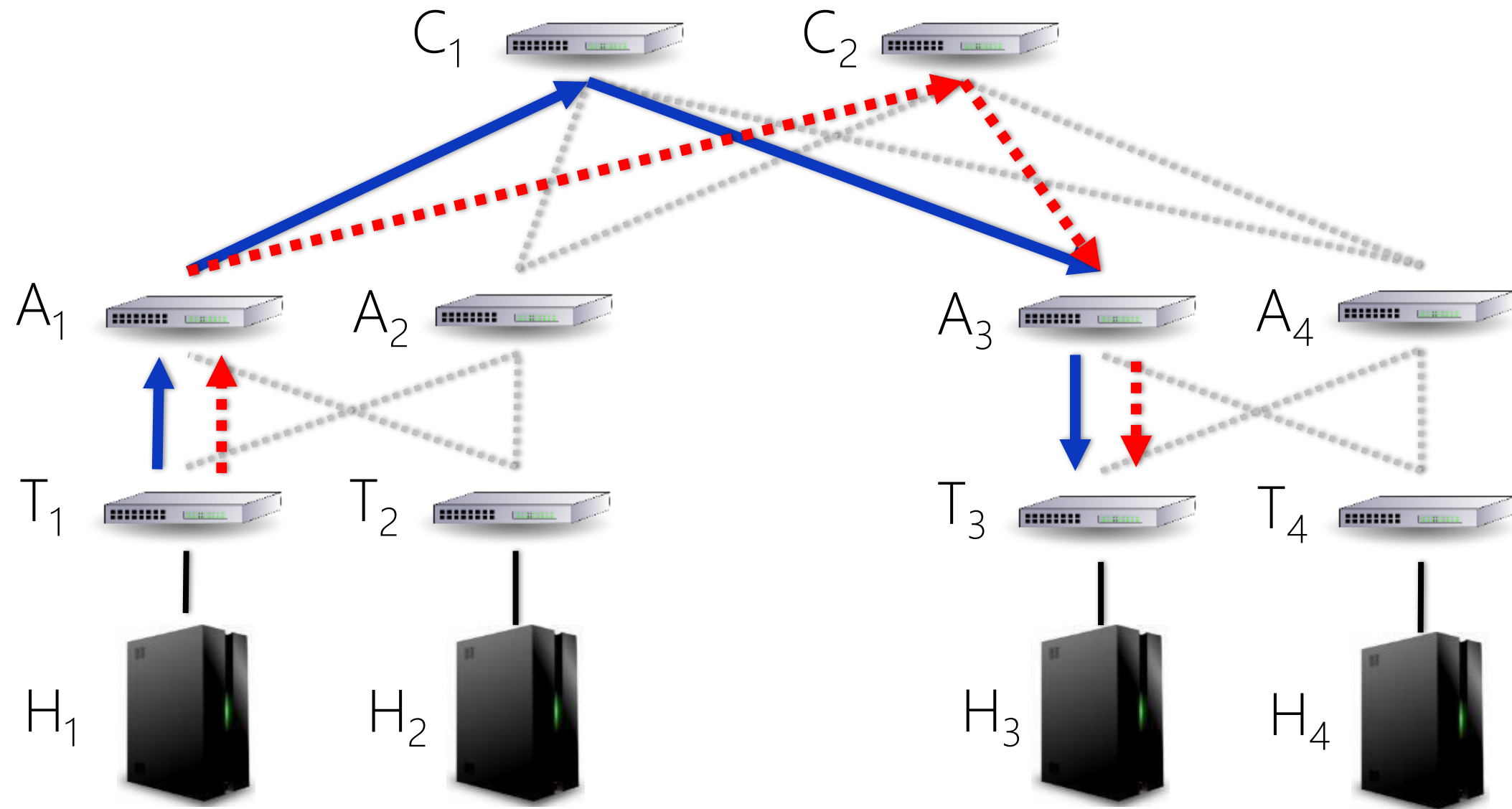
# Example: Data Center



# Network Configuration

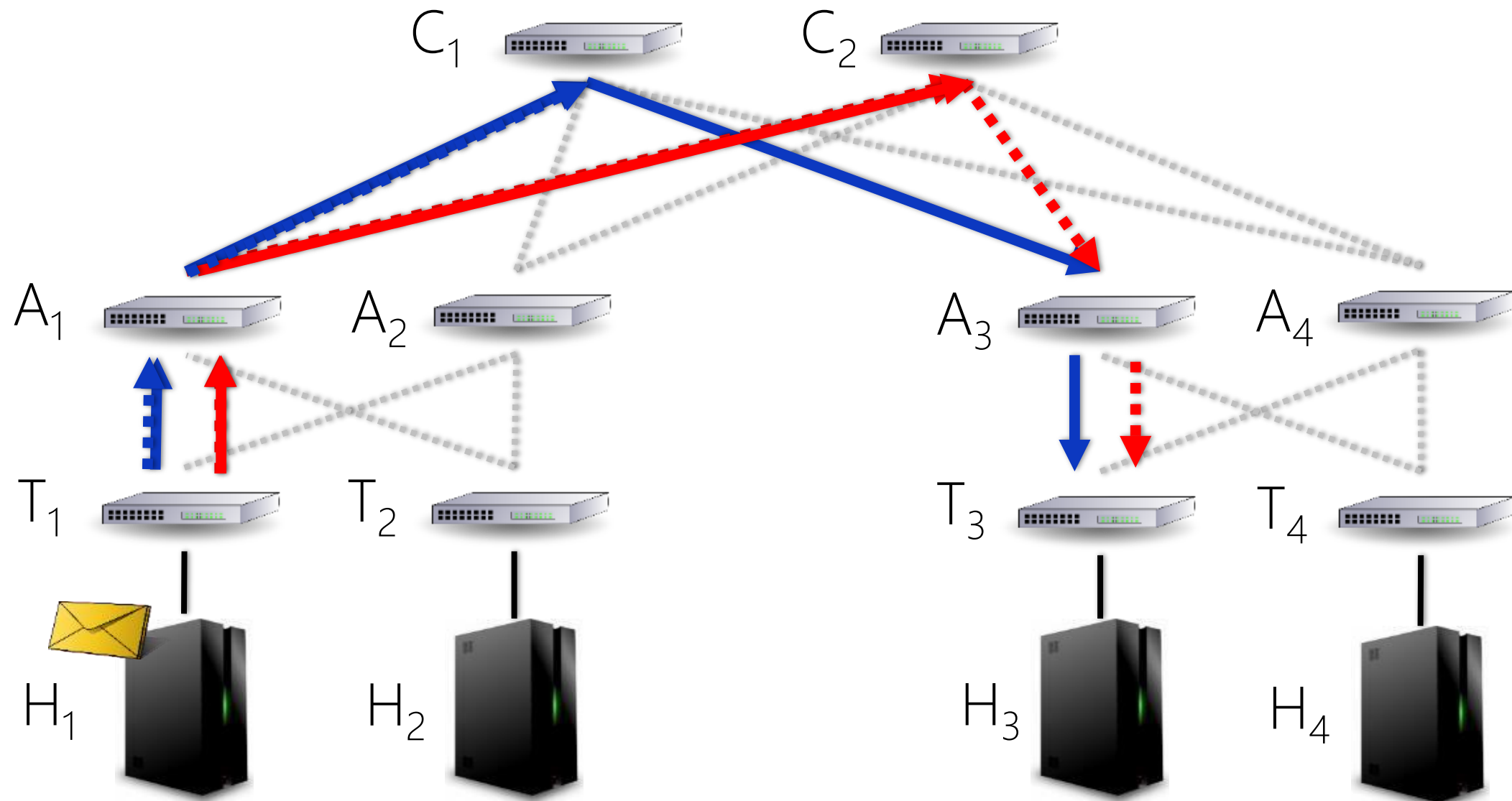


# Network Update



- Update program:  
upd  $T_1$ ; upd  $C_2$ ; upd  $A_3$ ; upd  $A_1$

# Naïve Update

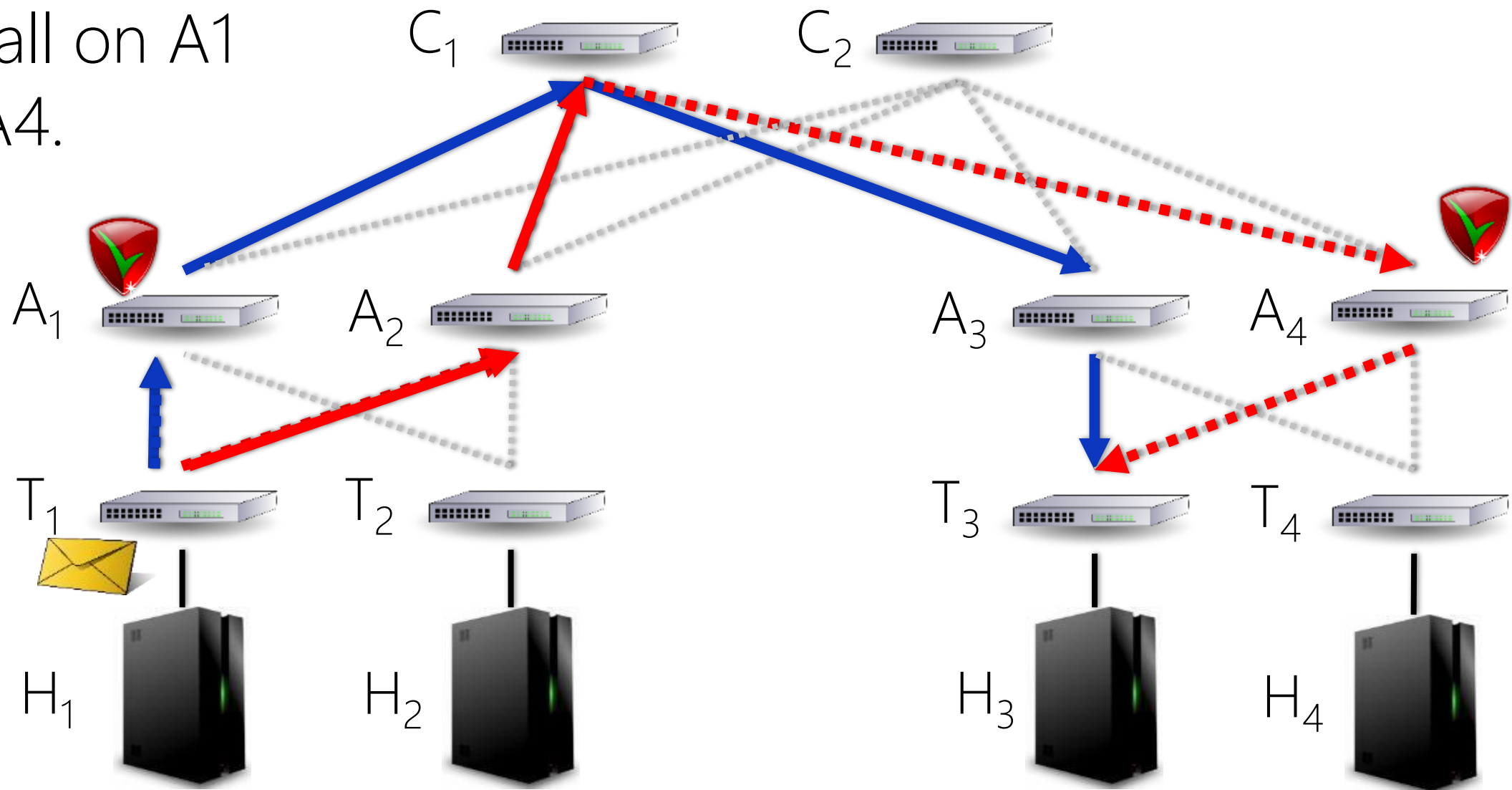


Possible problem: black holes



# 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

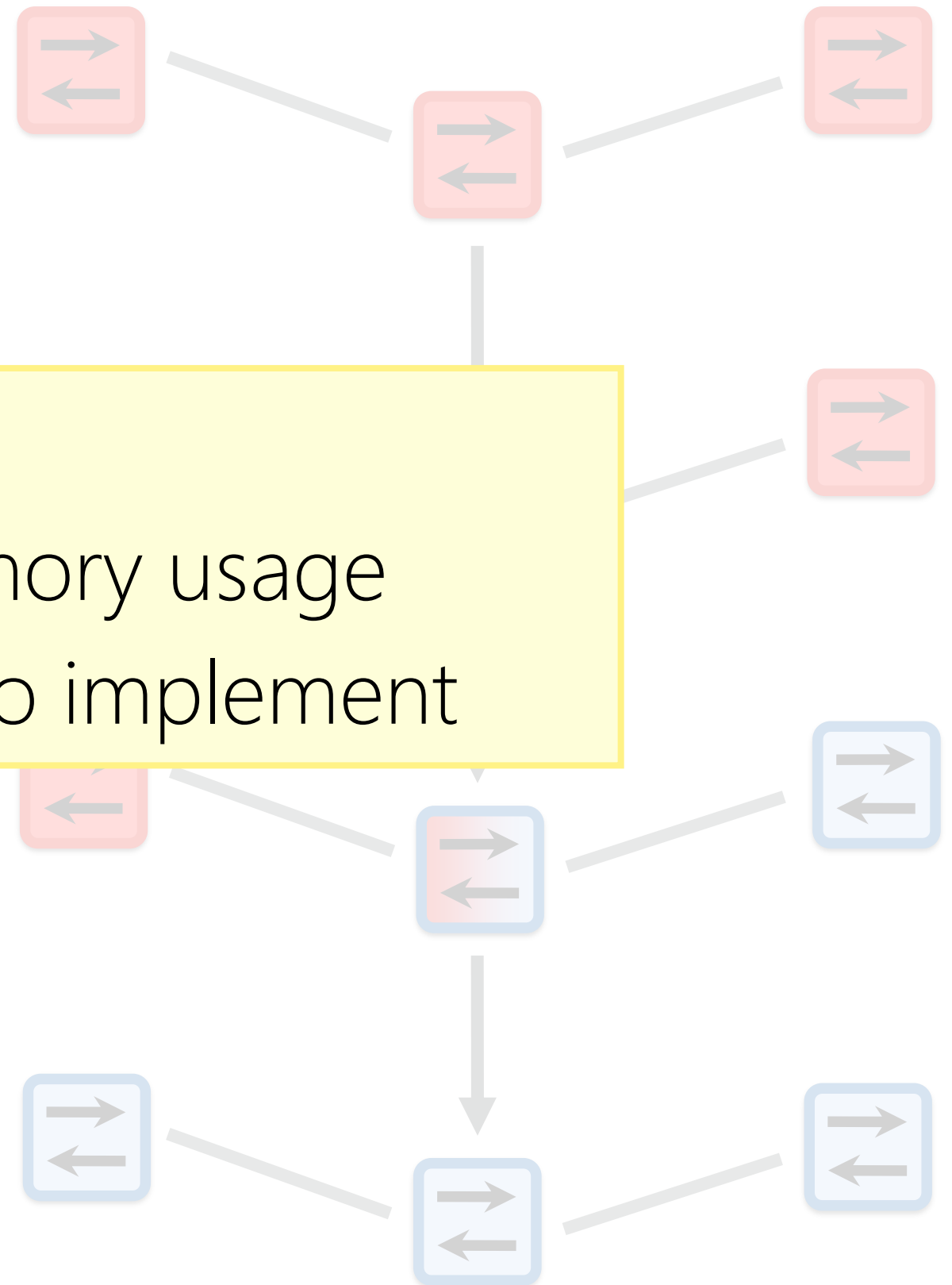
**Consistency Guarantee:** every packet (or flow) in the network “sees” a single policy version

## Two-Phase Update:

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

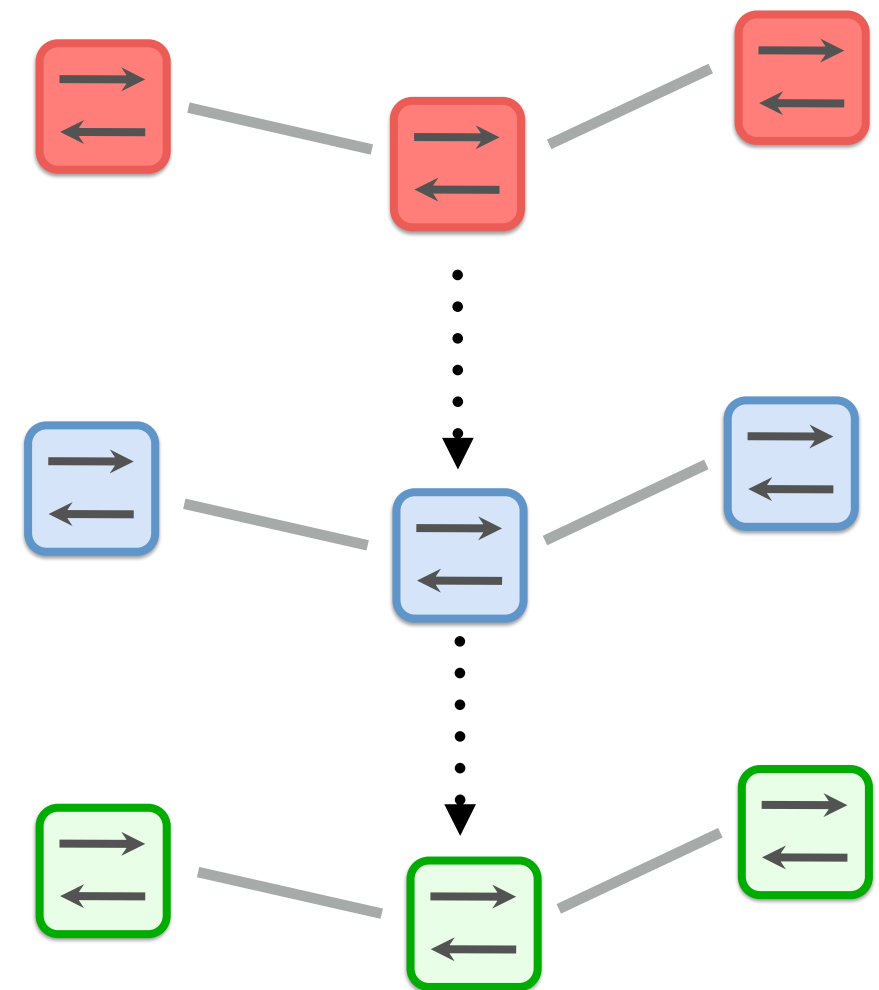
### Limitations:

- 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.

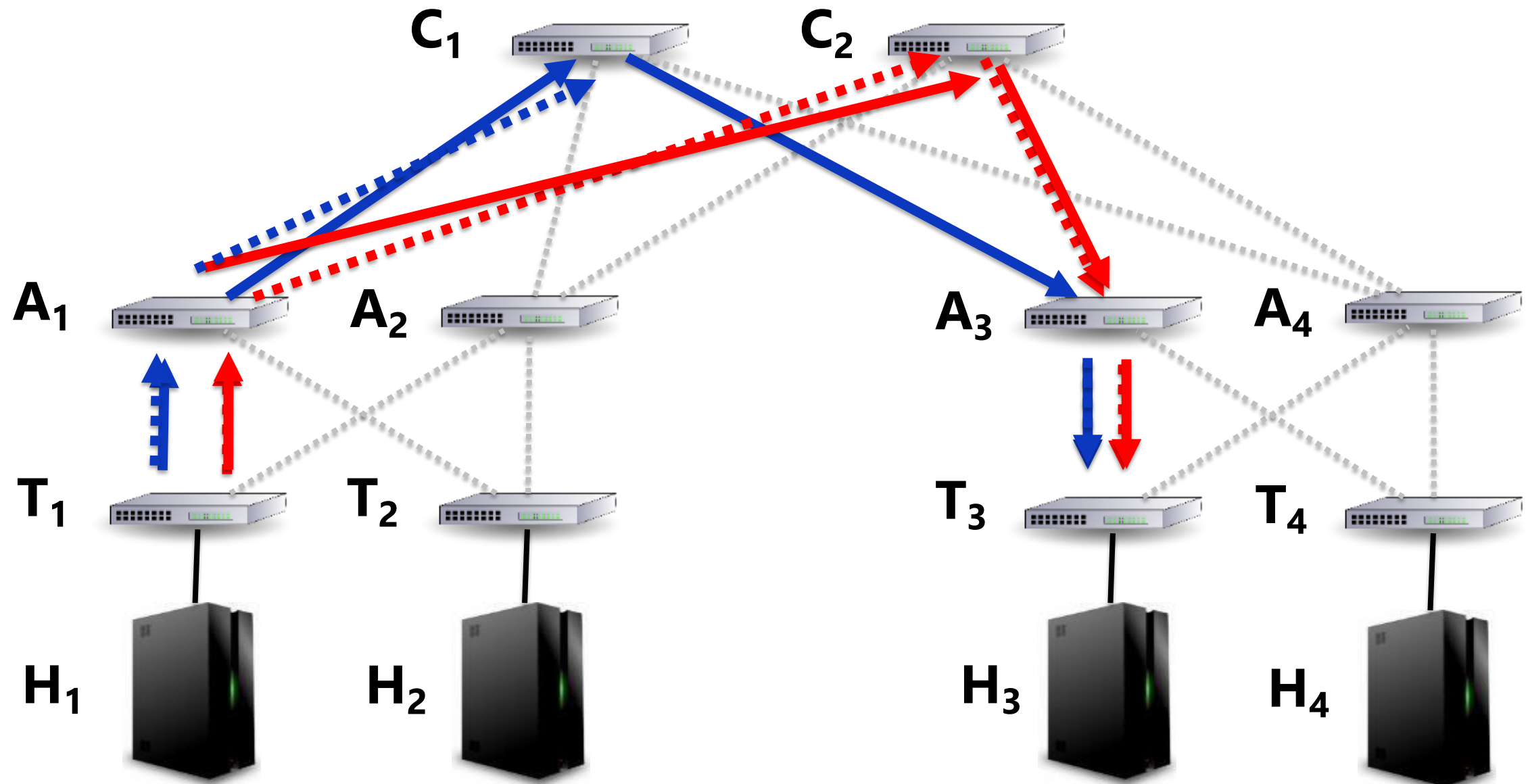


## Questions:

- 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



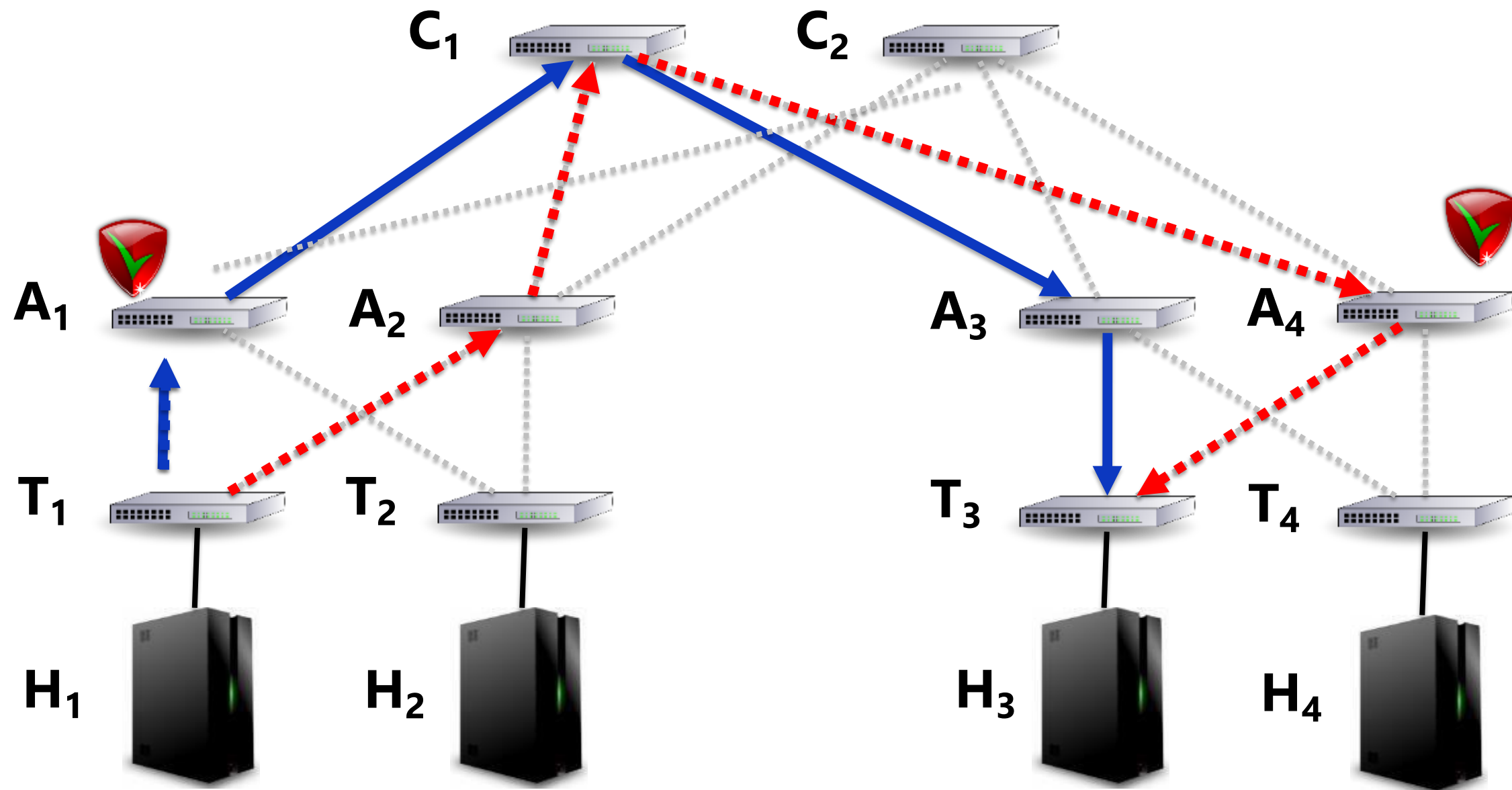
# Example: Data Center



**Update:** upd  $T_1$ ; upd  $C_2$ ; upd  $A_3$ ; upd  $A_1$

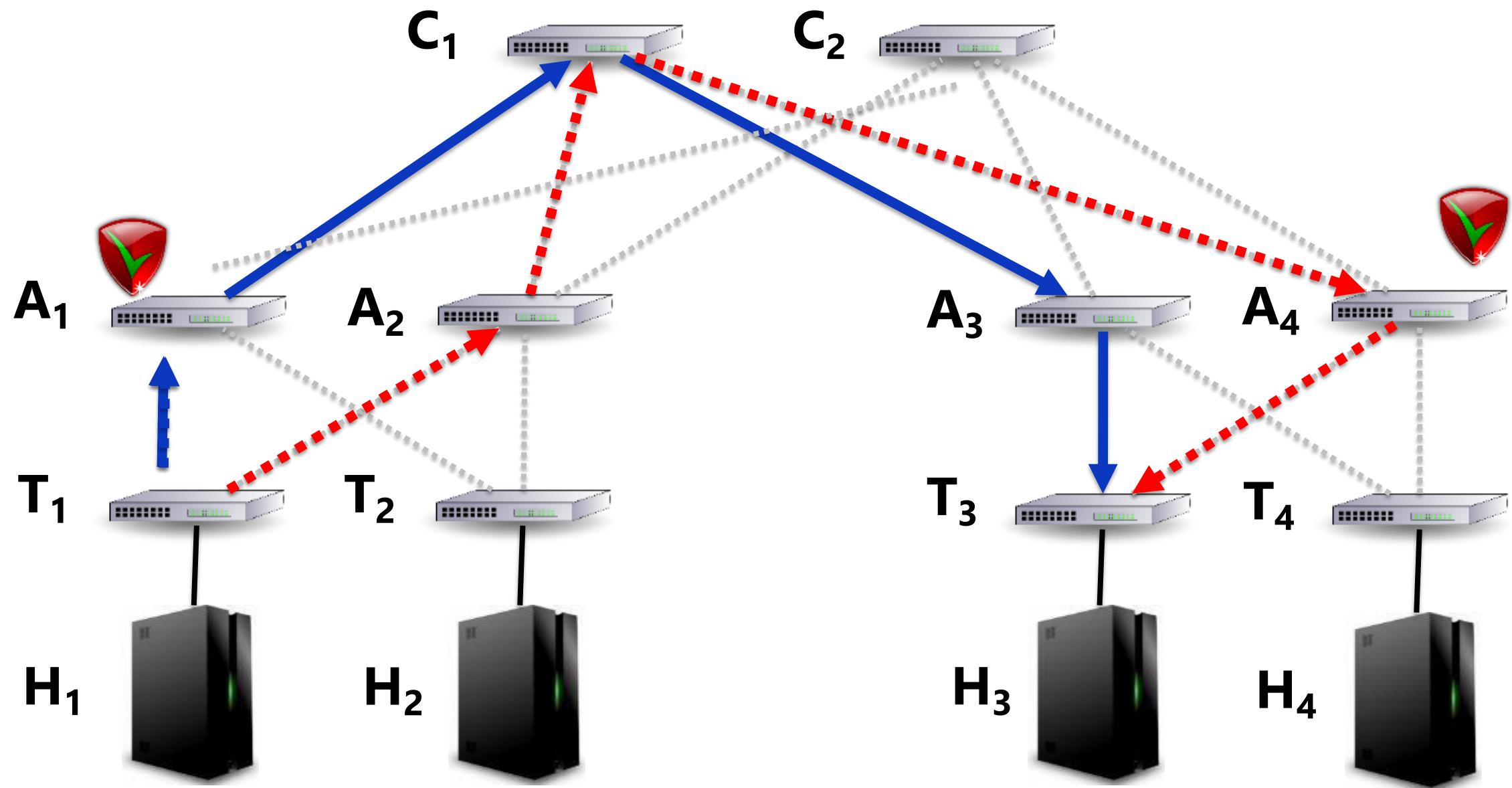


# Naive Update



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

# Relaxing Per-Packet Consistency



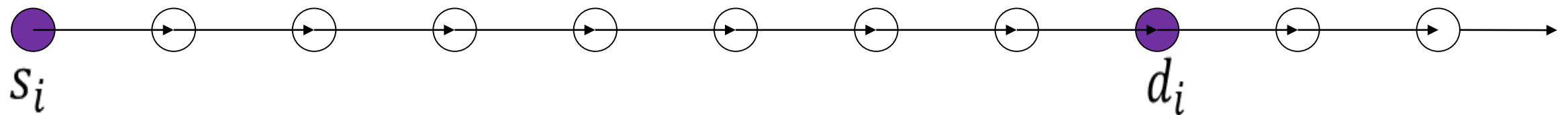
**Idea:** all packets eventually delivered via A<sub>1</sub> or A<sub>4</sub>

- **Update:** upd A<sub>2</sub>; upd A<sub>4</sub>; upd T<sub>1</sub>; upd C<sub>1</sub> X
- **Update:** upd A<sub>2</sub>; upd A<sub>4</sub>; upd C<sub>1</sub>; upd T<sub>1</sub>



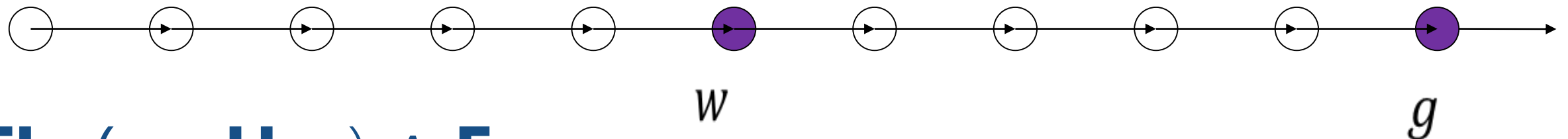
# How to Specify Properties?

**Reachability:** every packet that starts at  $s_i$  reaches  $d_i$



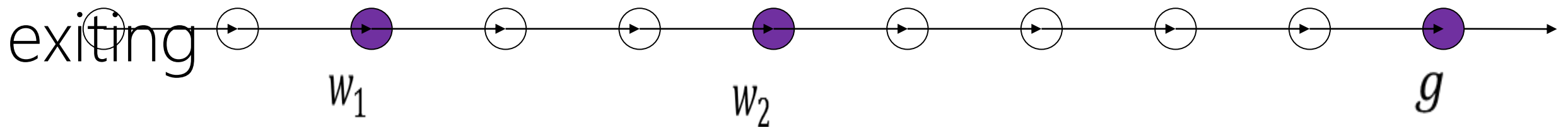
**LTL:**  $(s_i \rightarrow \mathbf{F} d_i)$

**Waypointing:** all packets traverse  $w$  before exiting



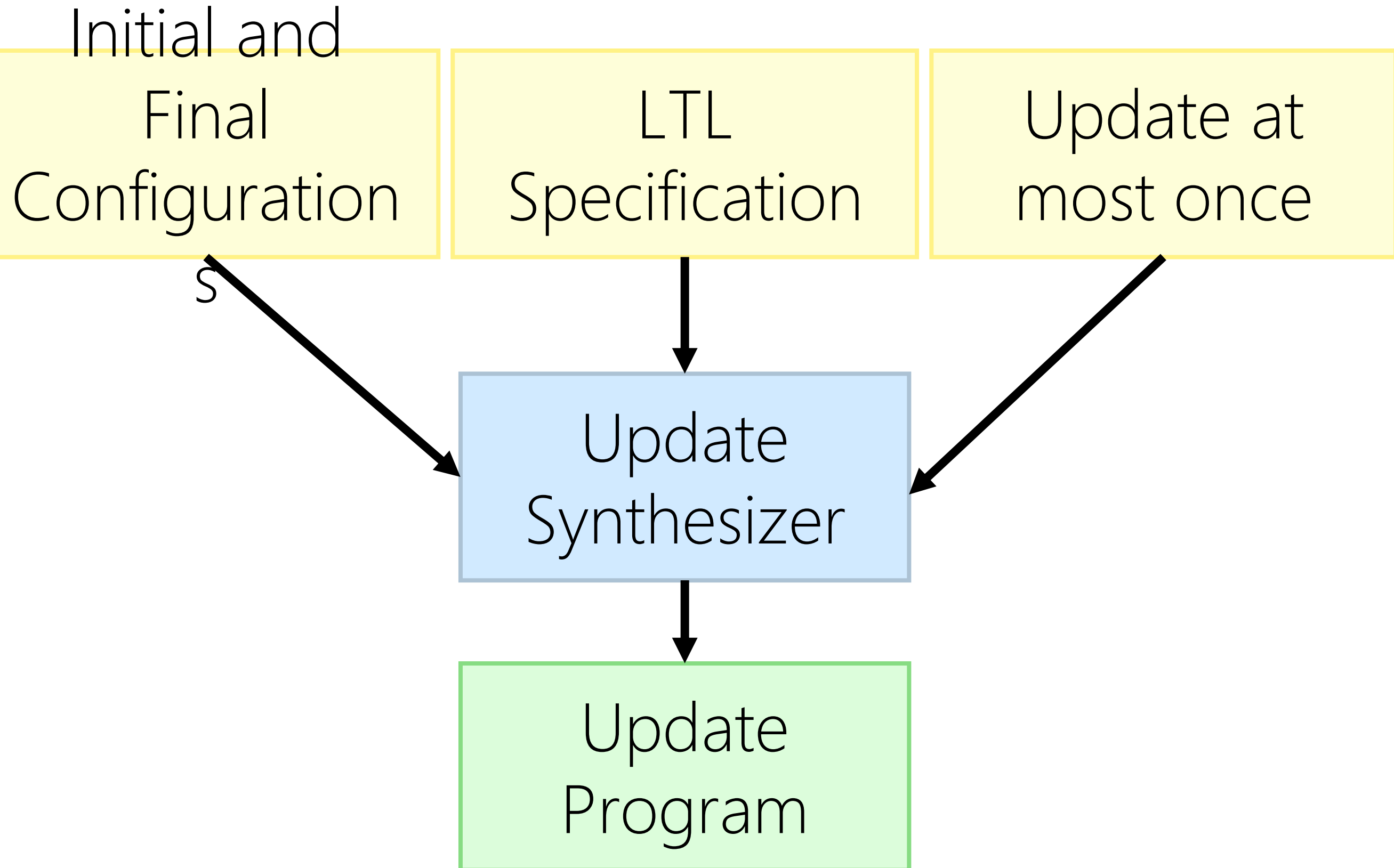
**LTL:**  $(\neg g \mathbf{U} w) \wedge \mathbf{F} g$

**Chaining:** all packets traverse  $w_1$  and  $w_2$  before exiting

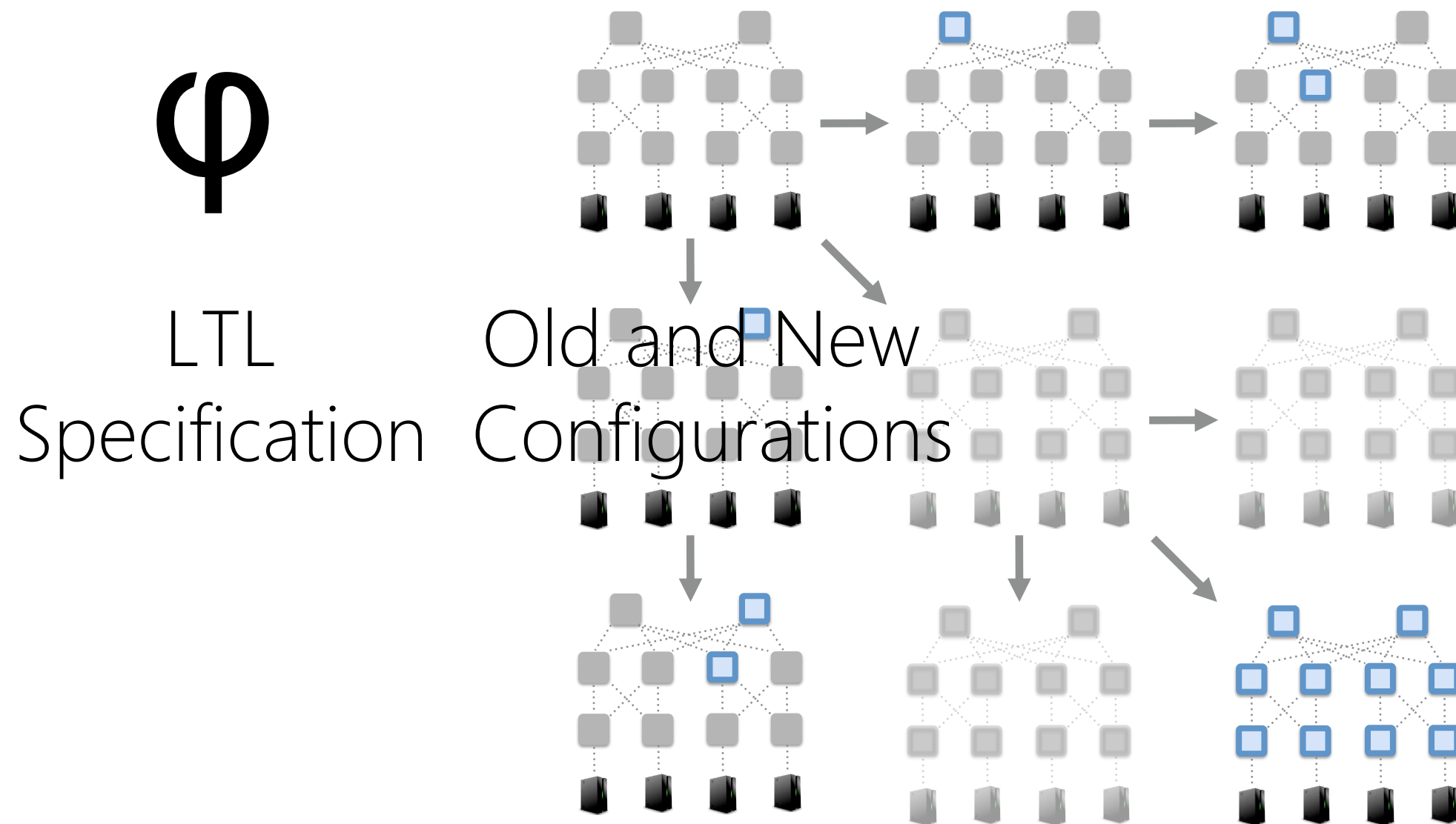


**LTL:**  $(\neg g \mathbf{U} w_2) \wedge (\neg w_2 \mathbf{U} w_1) \wedge \mathbf{F} g$

# Network Update Synthesis



# Synthesis Algorithm



# Synthesis Algorithm

## Depth-First Search:

- Attempt to update the switches one-by-one

• Backtrack

• bad config

• reach

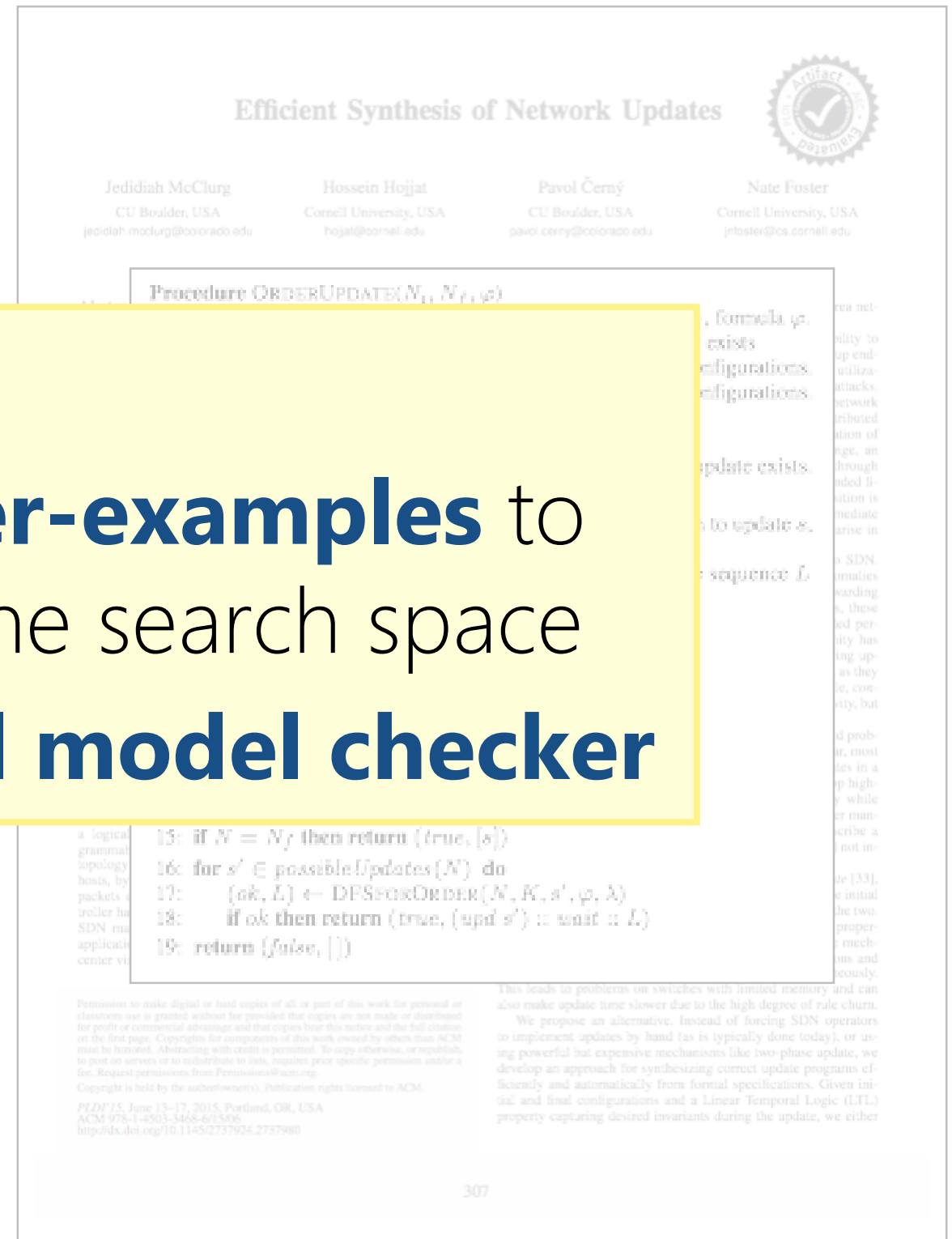
Challenge

• Search

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

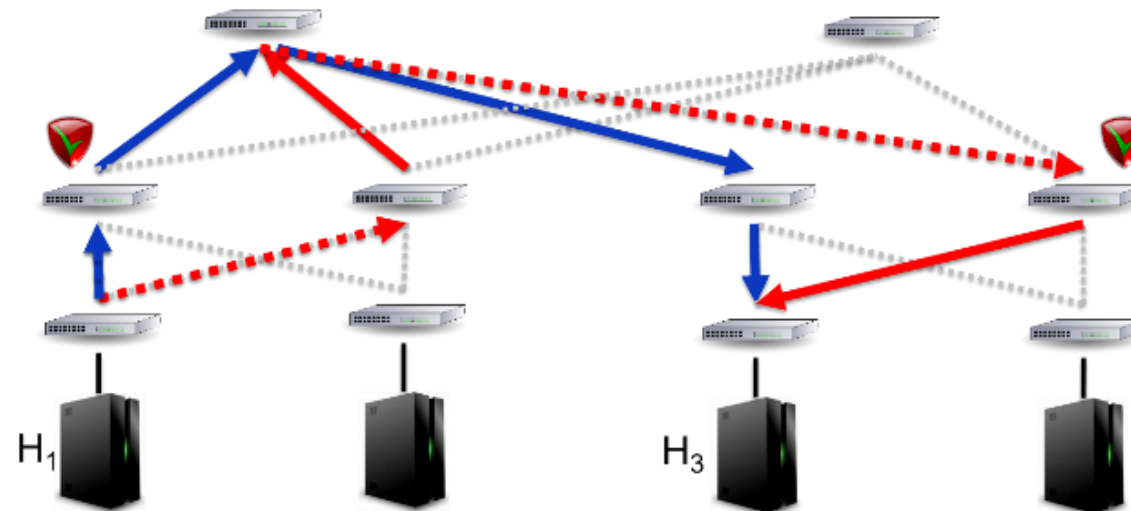
## Two main ideas:

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



# Model Checking

Model M:

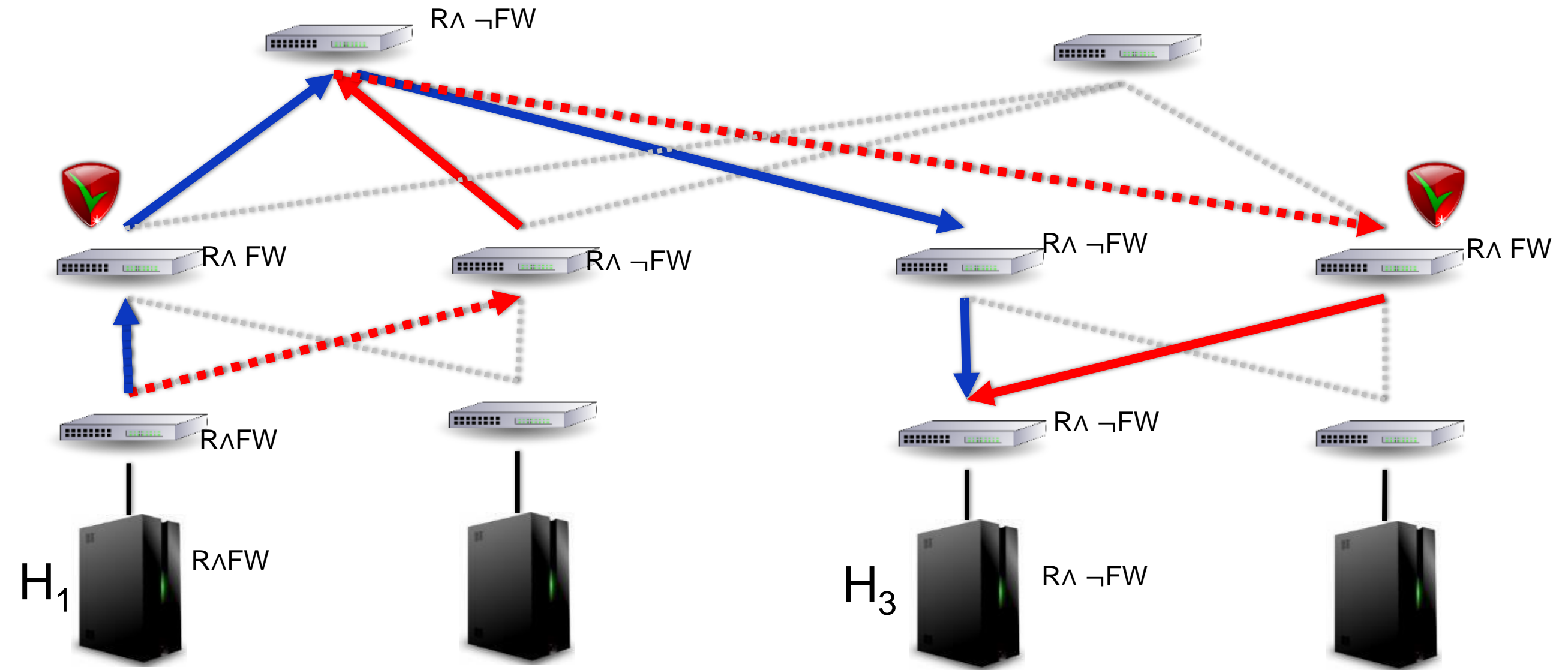


Specification S:

- ☐ all packets reach  $H_3$
- ☐ all packets traverse firewall

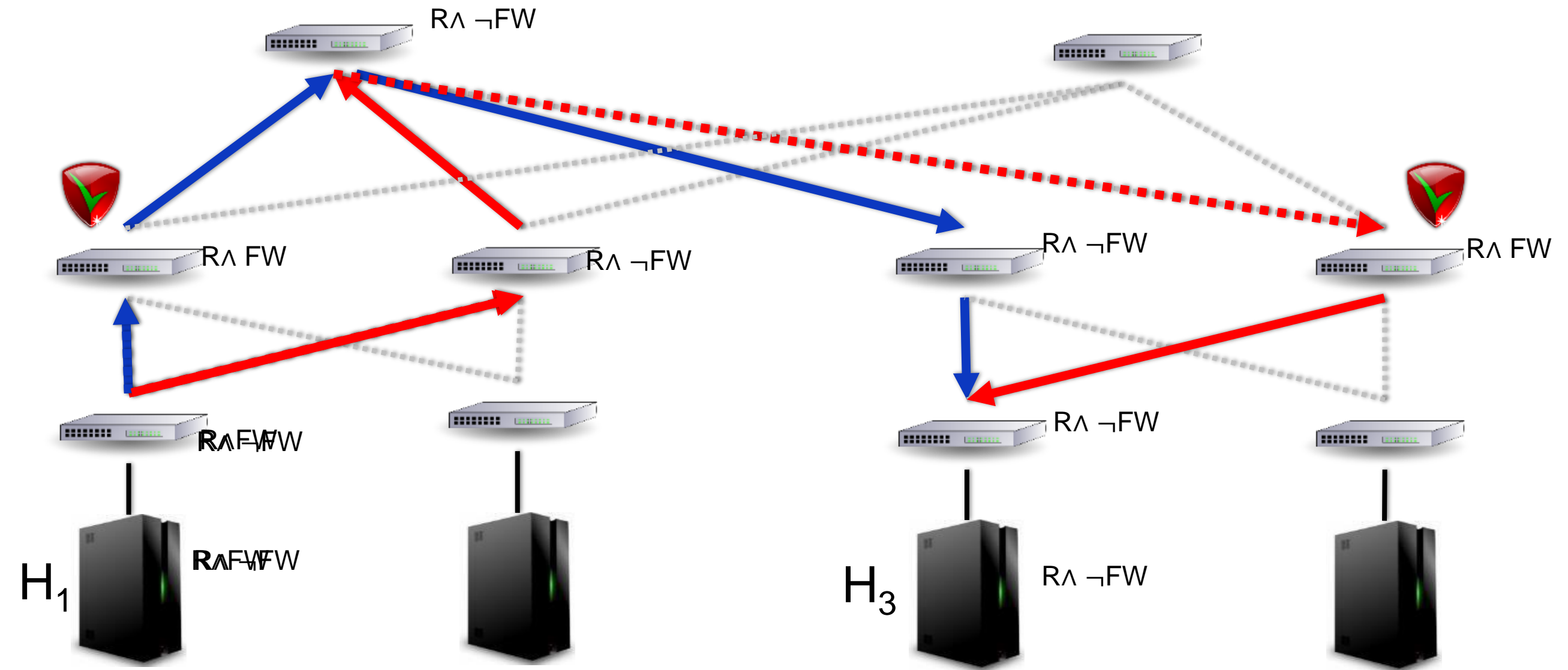
Question: Does M satisfy S?

# Model Checking



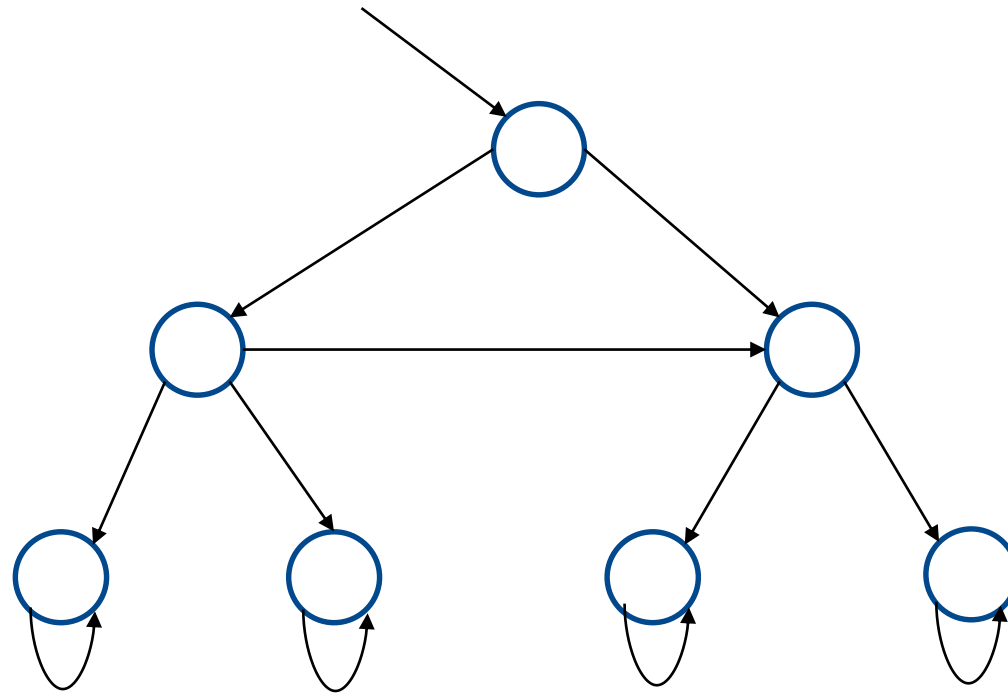
- ☐  $R$  holds at a switch  $s$  if all packets that traverse  $s$  reach  $H_3$
- ☐  $FW$  holds at a switch  $s$  if all packets that traverse  $s$  then traverse firewall

# Incremental model checking



- ☐  $R$  holds at a switch  $s$  if all packets that traverse  $s$  reach  $H_3$
- ☐  $FW$  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)

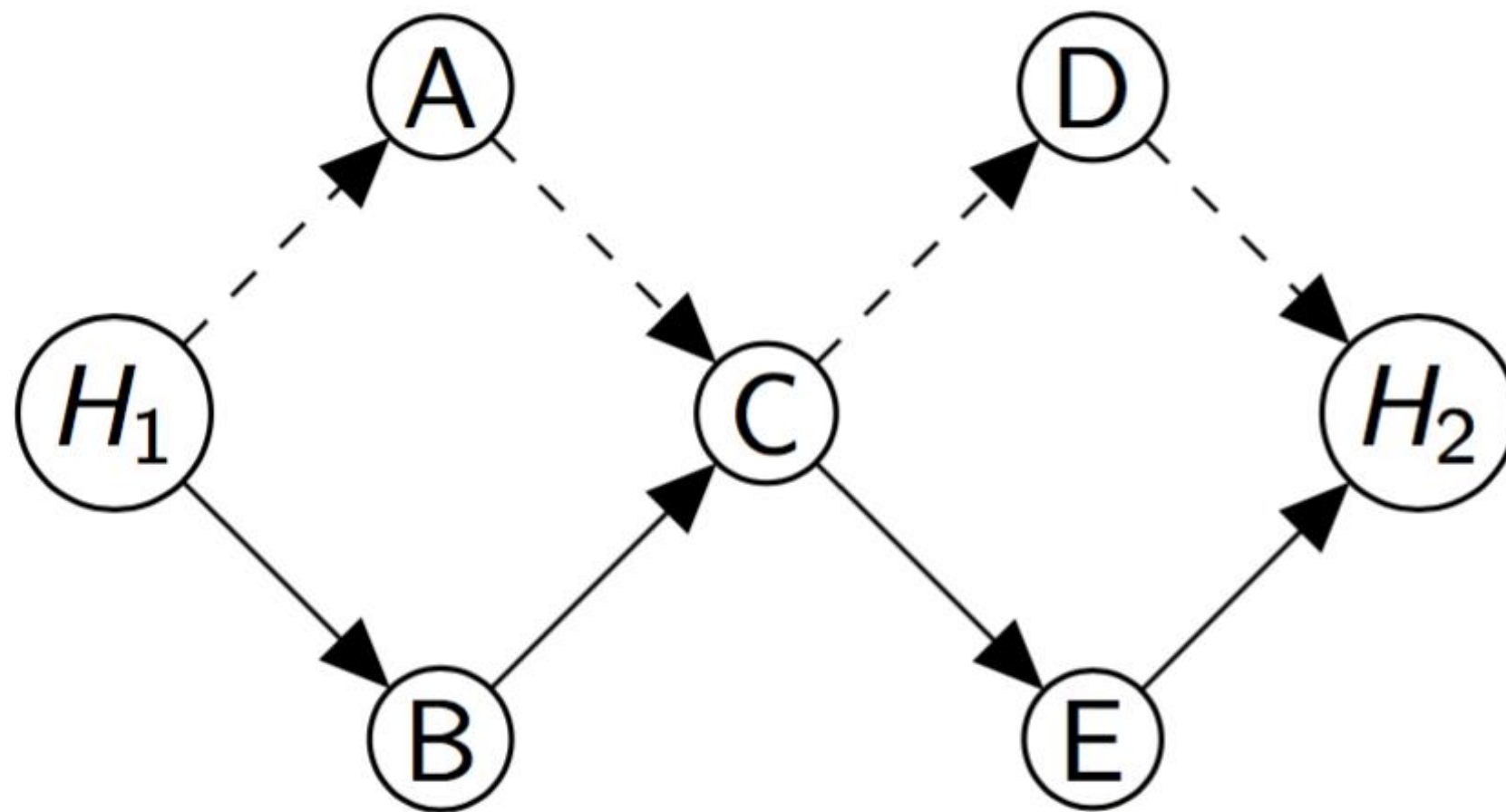


# Main Limitation

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

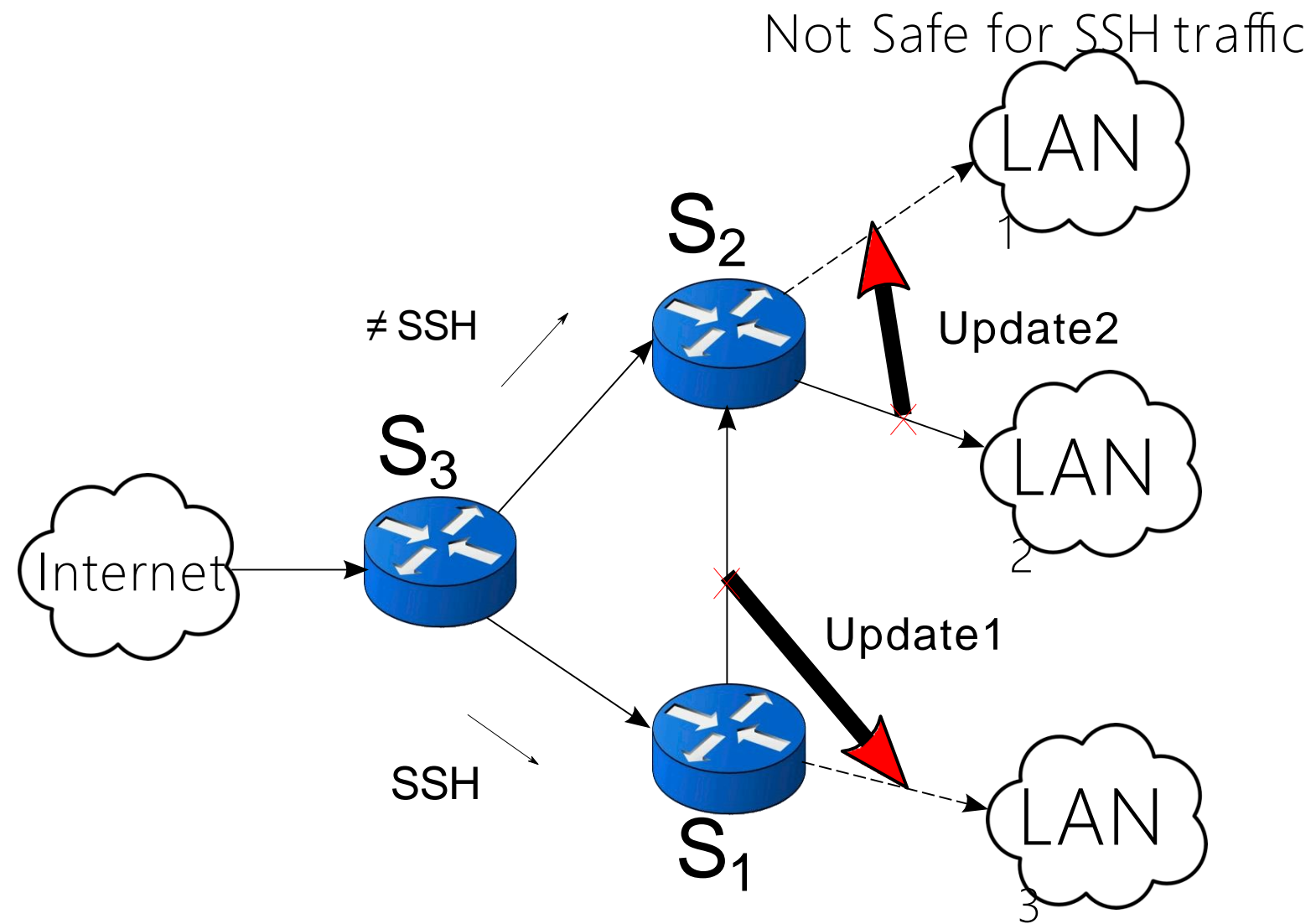
**Example:** "double diamond"

[DISC '16]



Our implementation reverts to a two-phase update...

# Waits



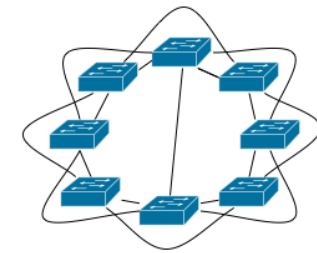
# Waits

- 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

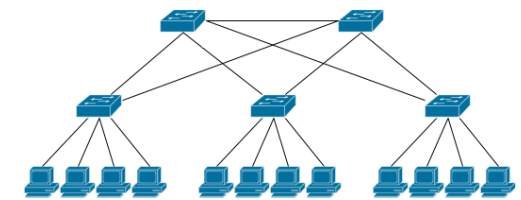
# Evaluation

## Questions:

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



Small-world



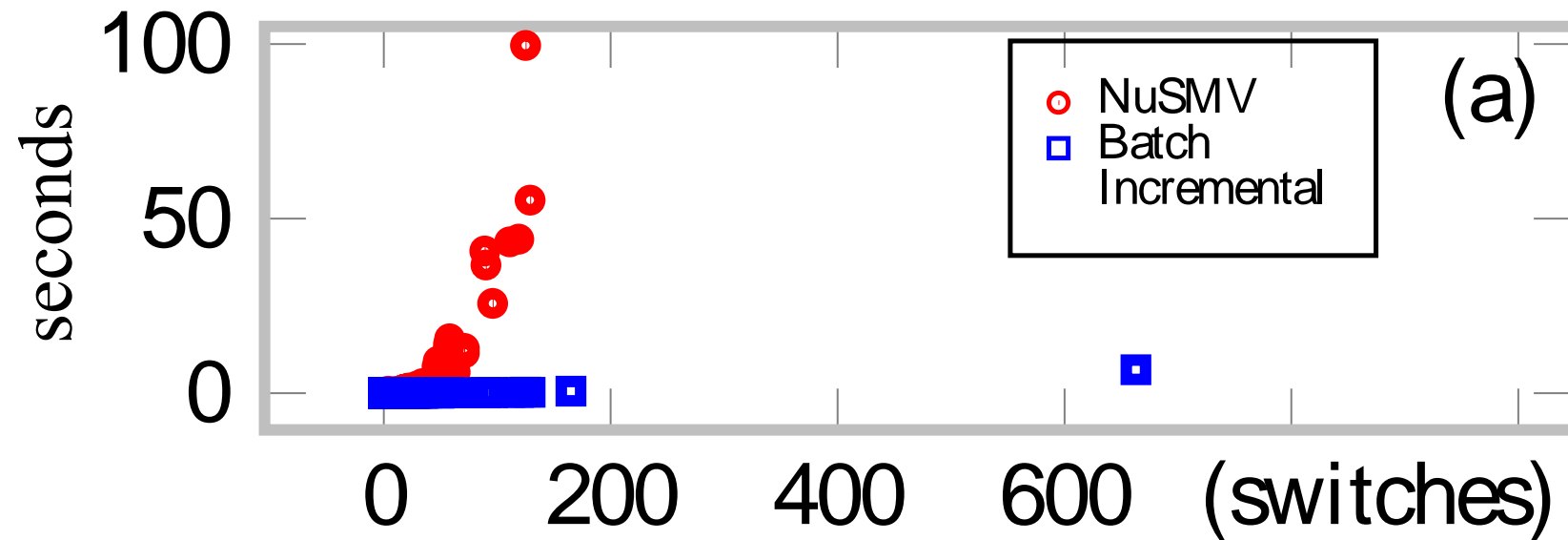
Fattree

## Methodology:

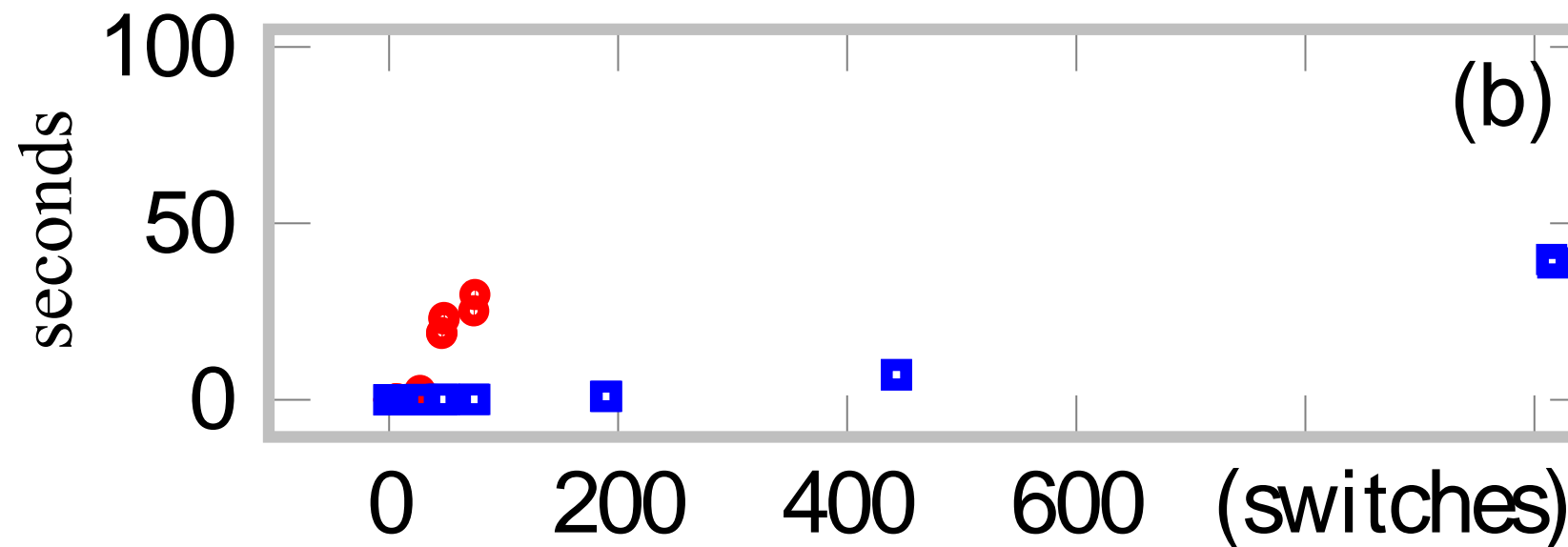
- Real-world topologies (TopoZoo, FatTrees, Small World)
- Synthetic configurations (e.g., shortest-path forwarding)

# Impact of Optimizations

TopoZoo



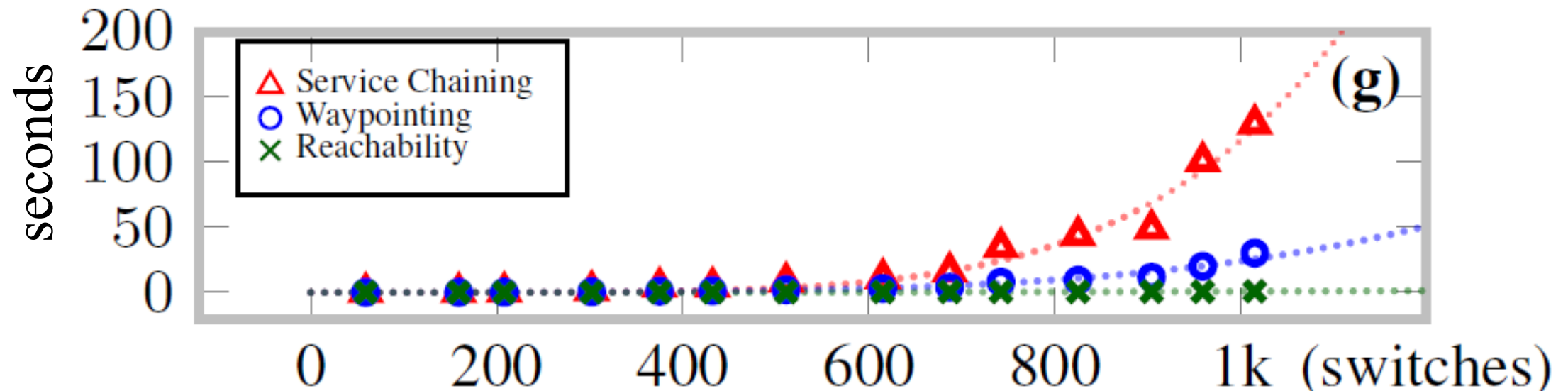
FatTree



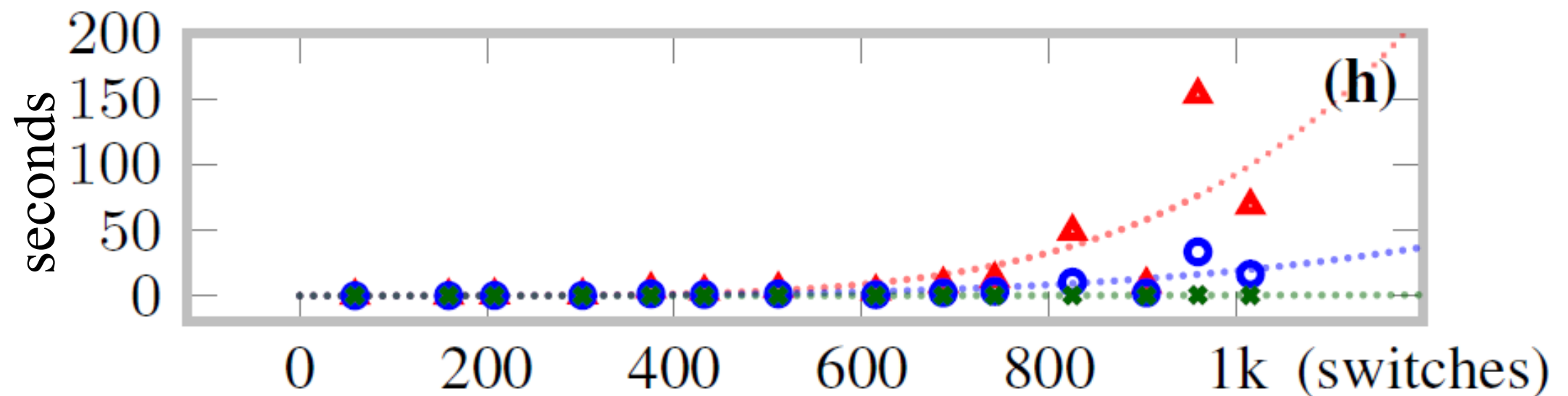
- **Configurations:** shortest-path forwarding

# Scalability

Feasible



Infeasible

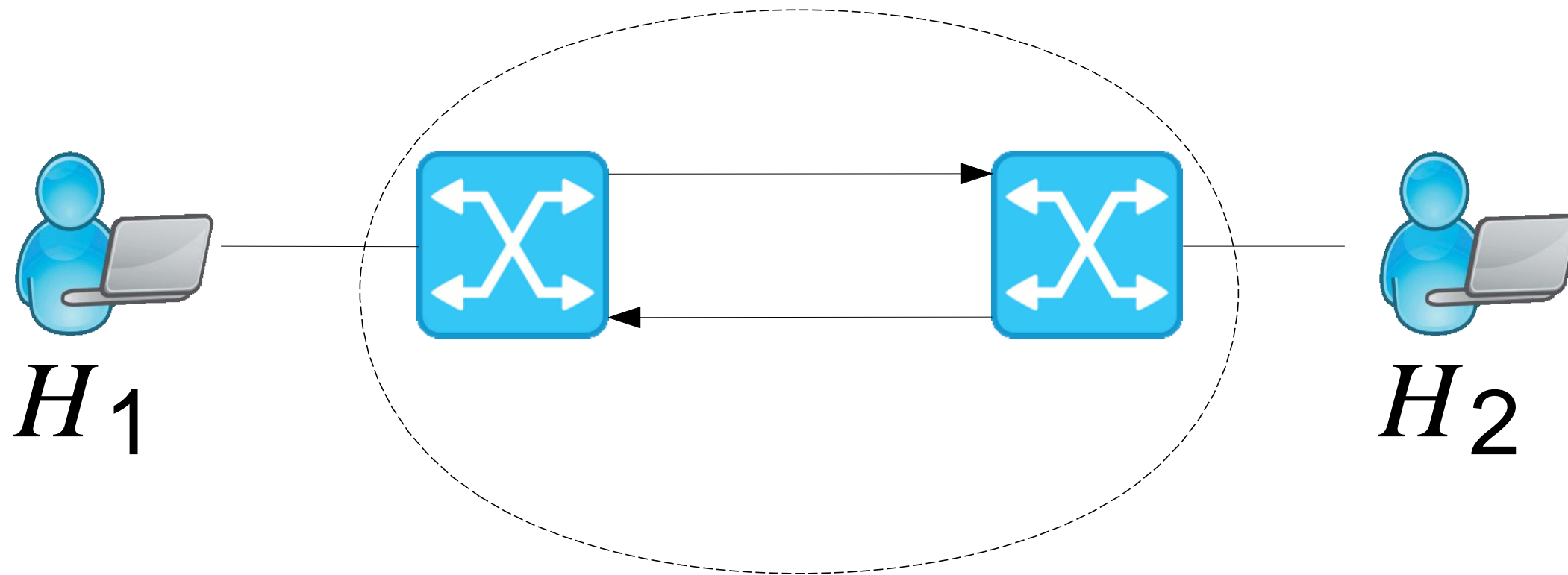


- **Configurations:** "diamond" / "double diamond"
- **Specifications:** reachability, waypointing, chaining



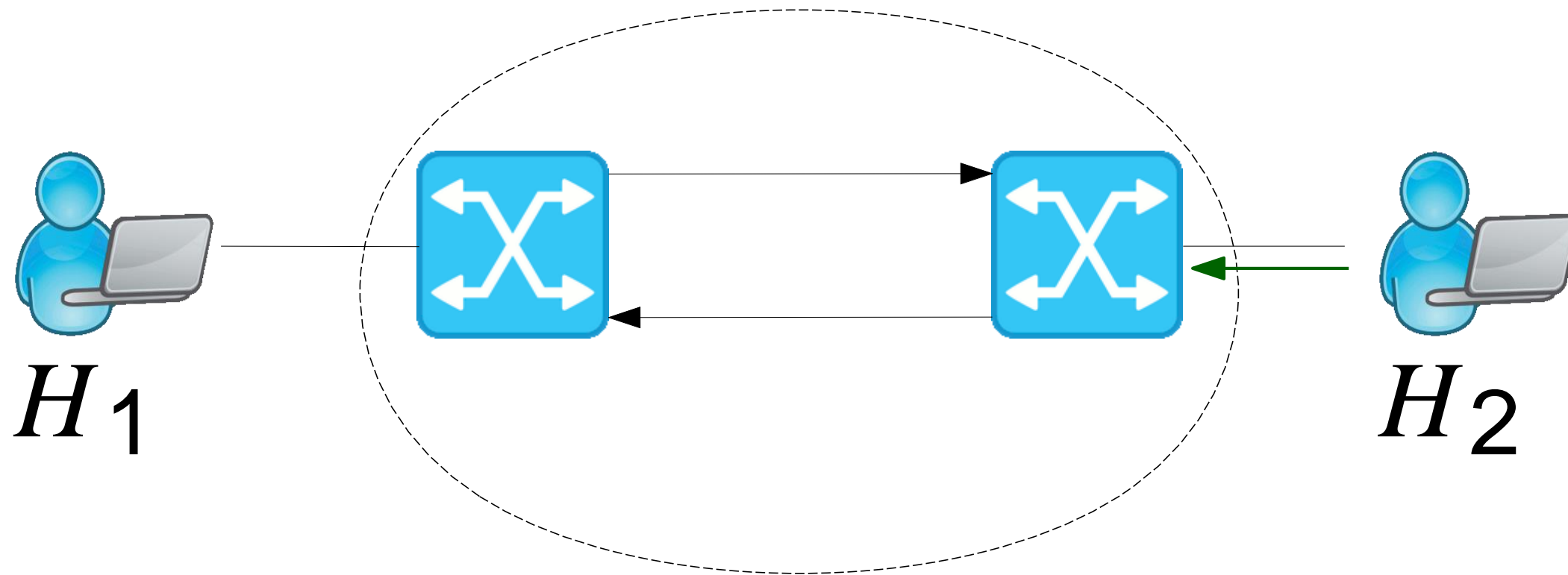
# Synchronization for Network Programs

# Stateful Firewall



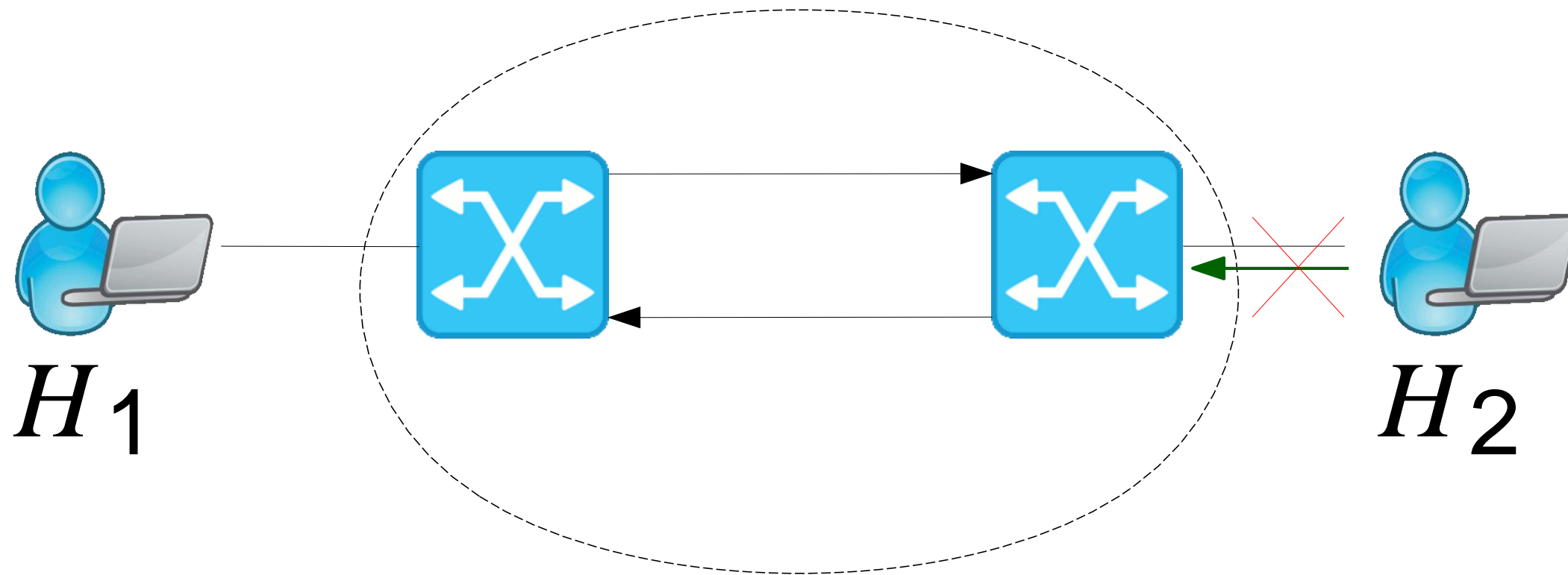
- $H_1$  should be allowed to communicate with  $H_2$
- $H_2$  should only be allowed to communicate with  $H_1$  if  $H_1$  has previously initiated a connection

# Stateful Firewall



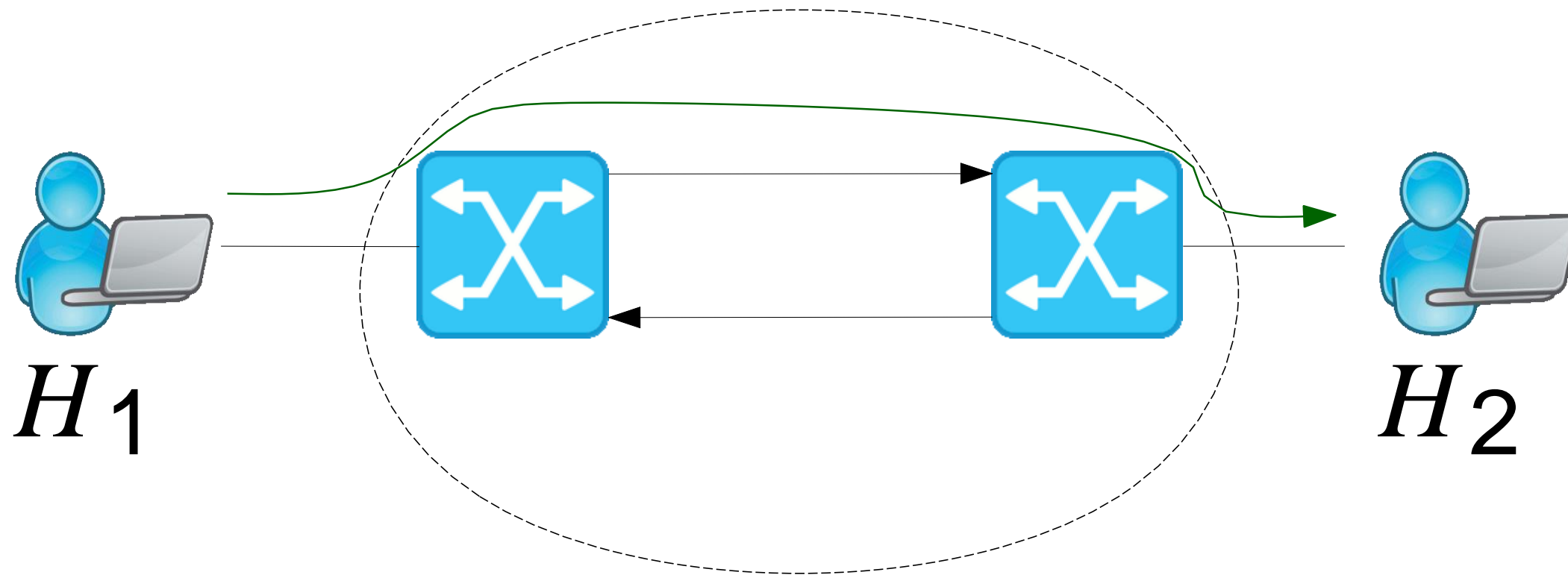
- $H_1$  should be allowed to communicate with  $H_2$
- $H_2$  should only be allowed to communicate with  $H_1$  if  $H_1$  has previously initiated a connection

# Stateful Firewall



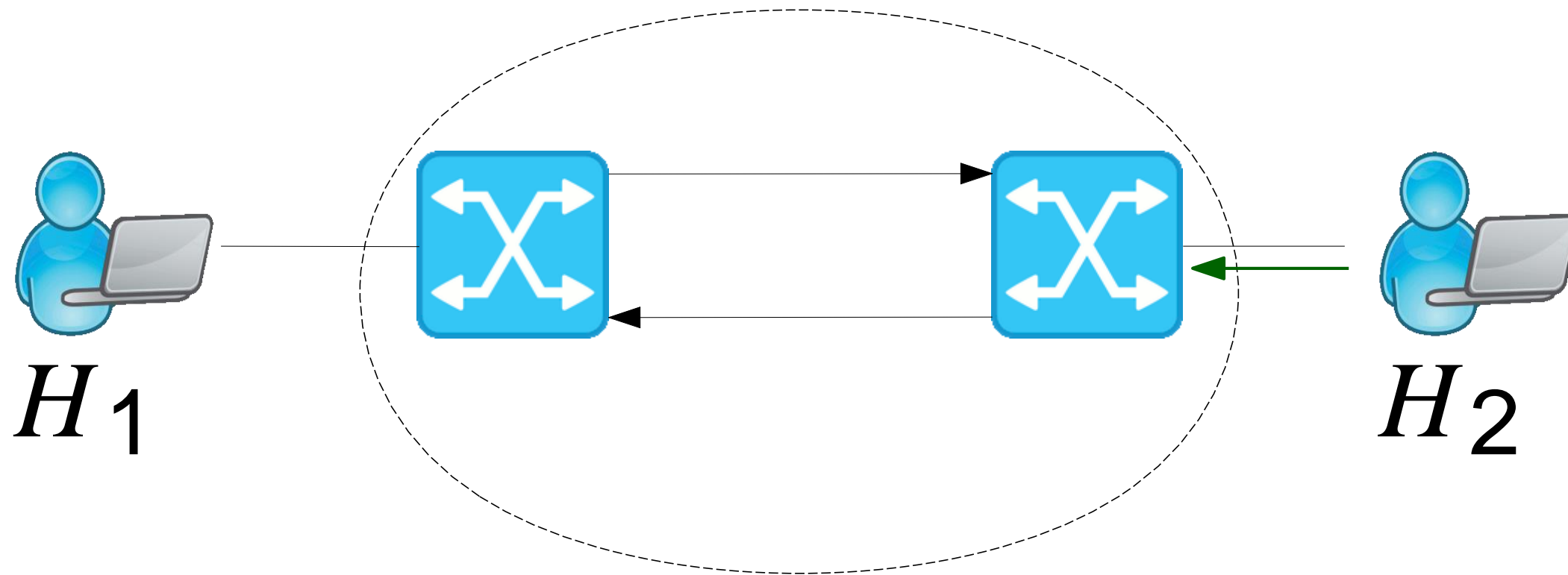
- $H_1$  should be allowed to communicate with  $H_2$
- $H_2$  should only be allowed to communicate with  $H_1$  if  $H_1$  has previously initiated a connection

# 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

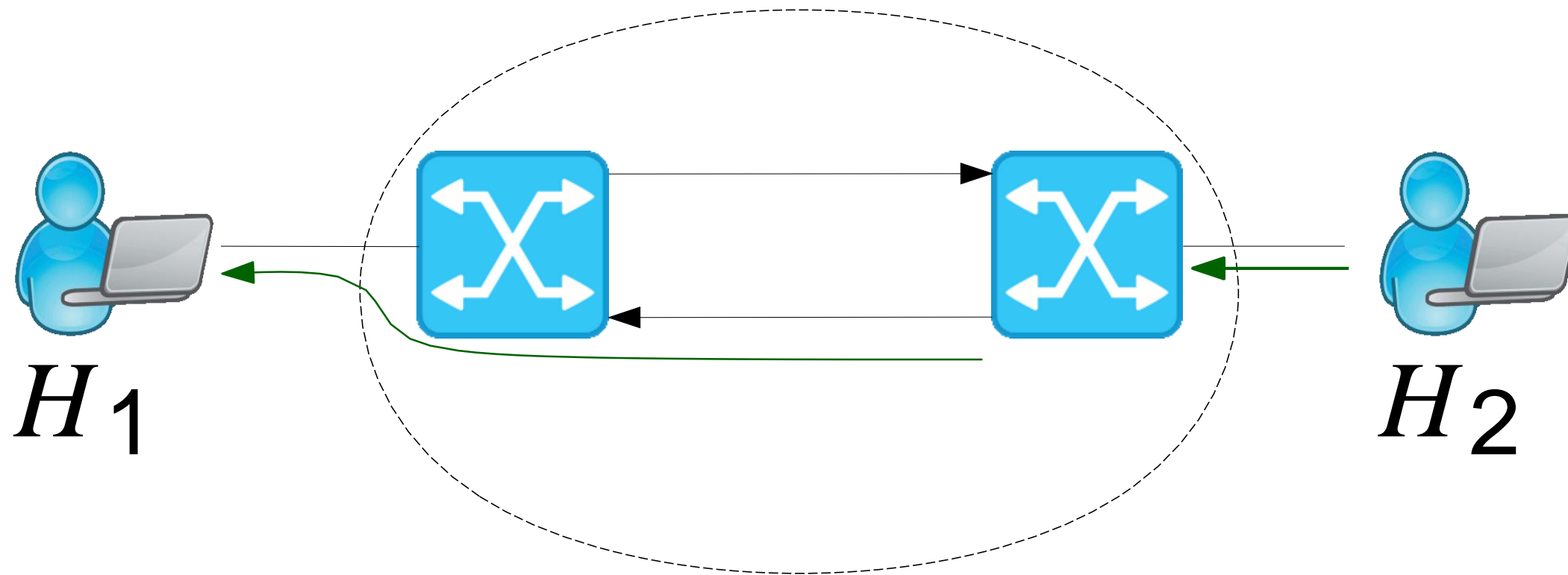
# Stateful Firewall



- $H_1$  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

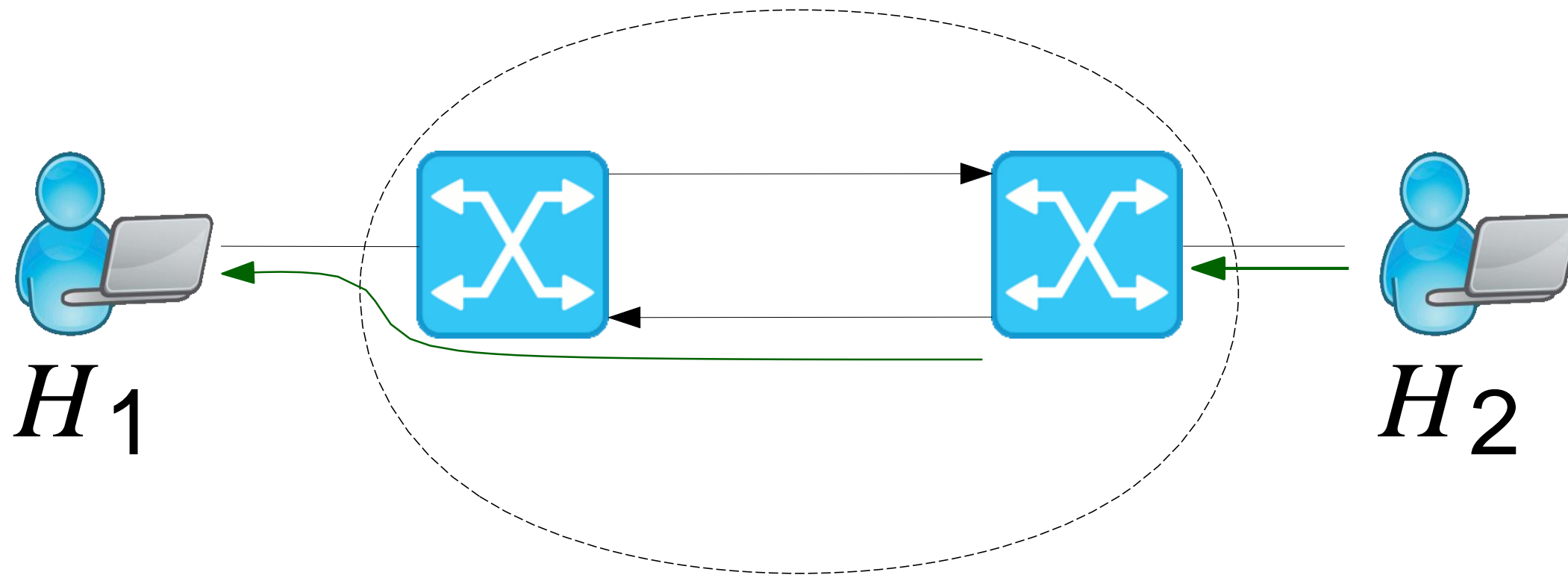


# Stateful Firewall

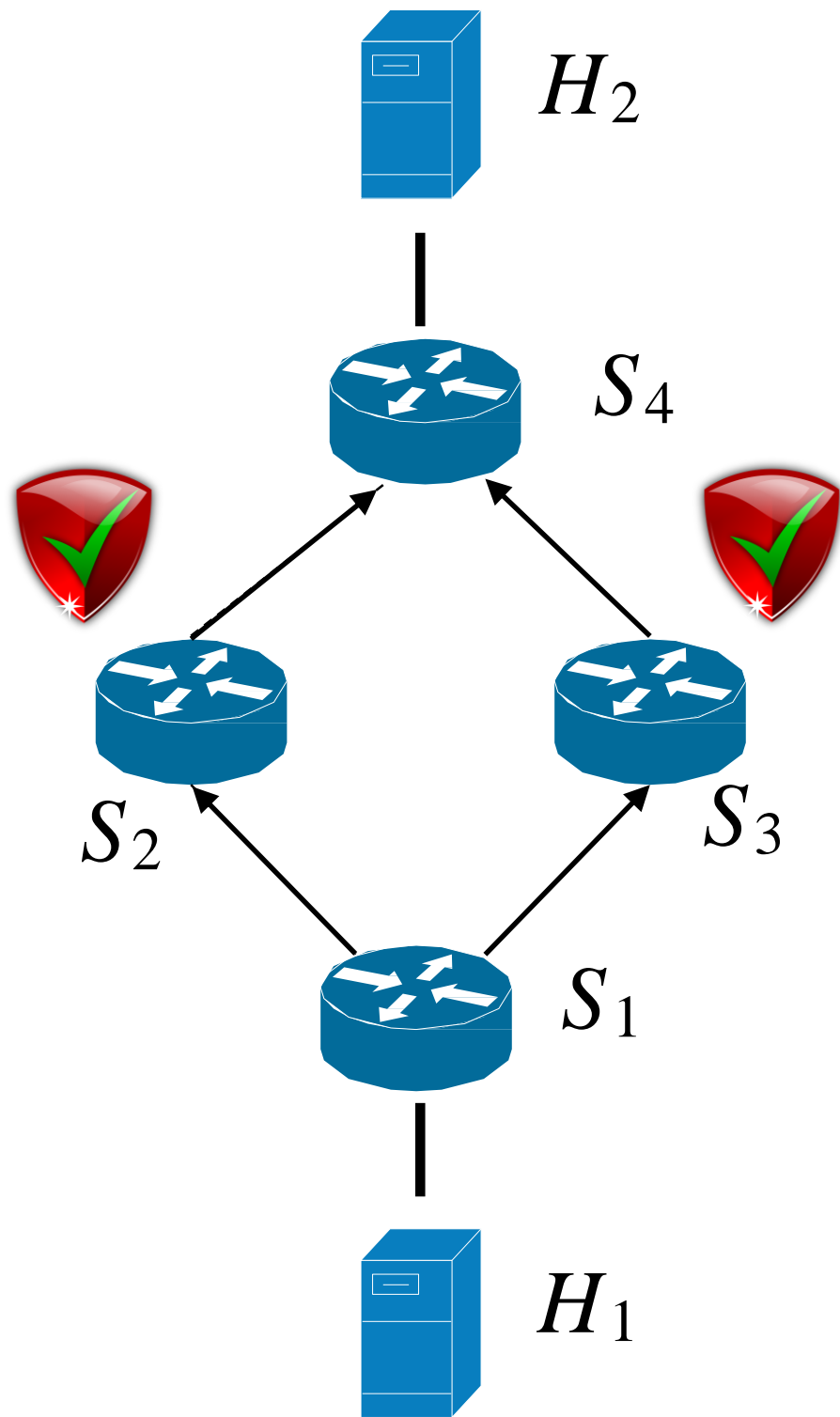


- $H_1$  should be allowed to communicate with  $H_2$
- $H_2$  should only be allowed to communicate with  $H_1$  if  $H_1$  has previously initiated a connection

# 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:

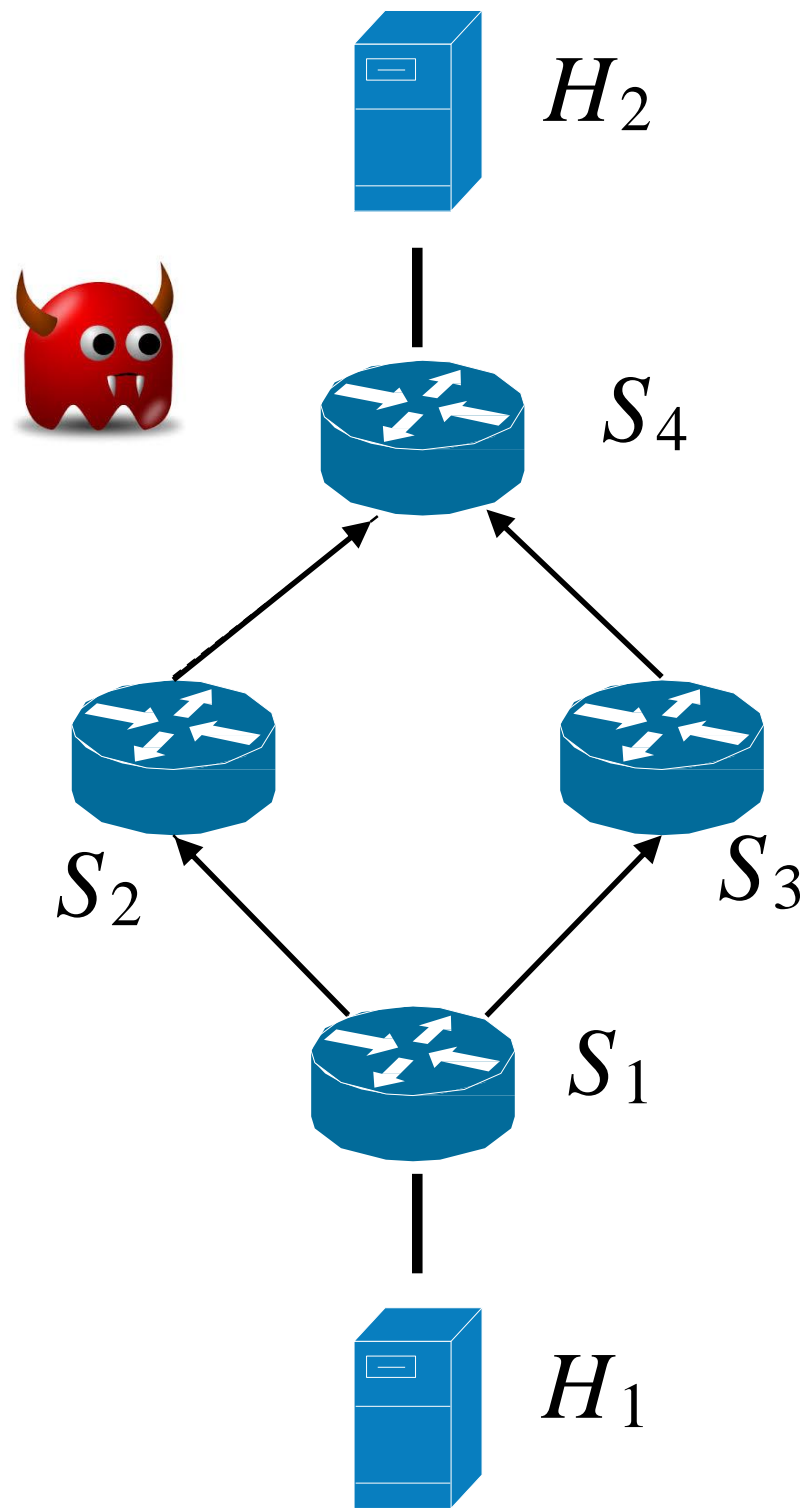
- Forward from  $H_1$  to  $H_2$  via  $S_1$ - $S_3$ - $S_4$
- $S_3$  has a firewall

## 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

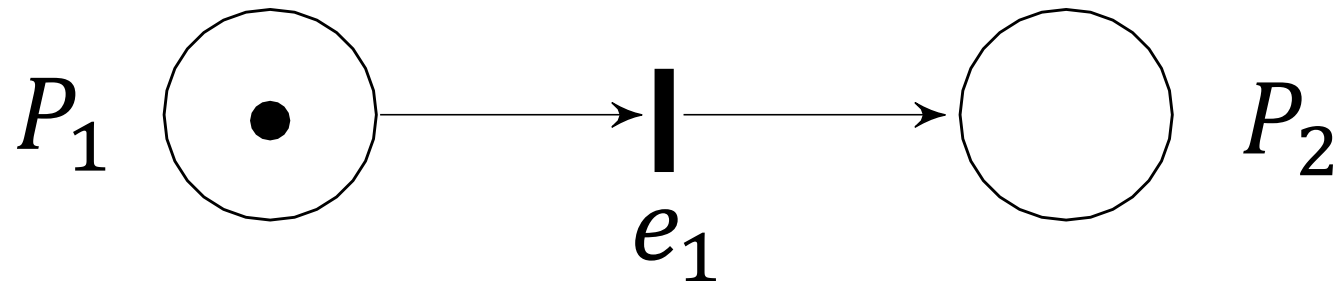
## General problem:

- Synchronization for event-driven network programs

# Solution

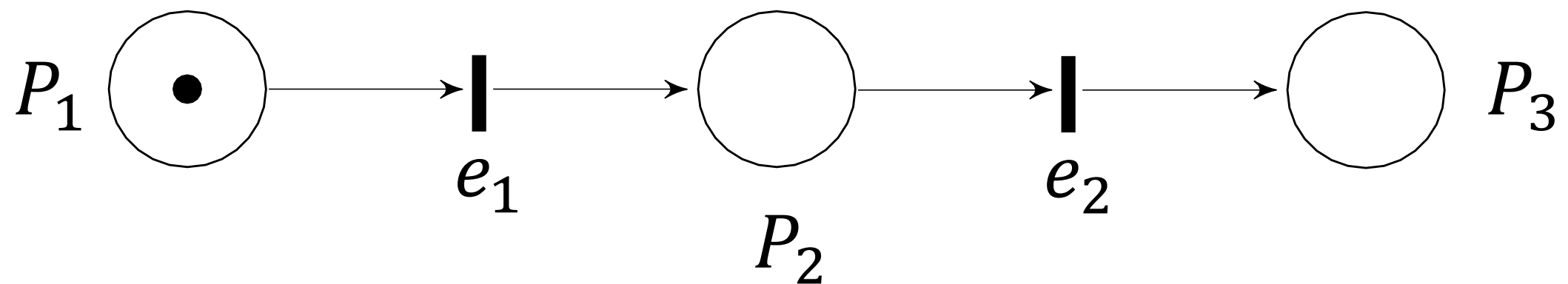
- Programming model: event nets
- Algorithmic synthesis of synchronization

# Event nets: One event-update



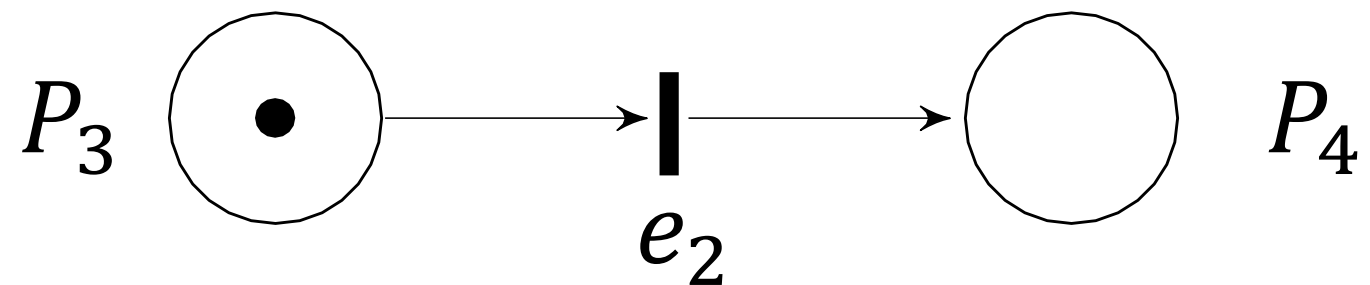
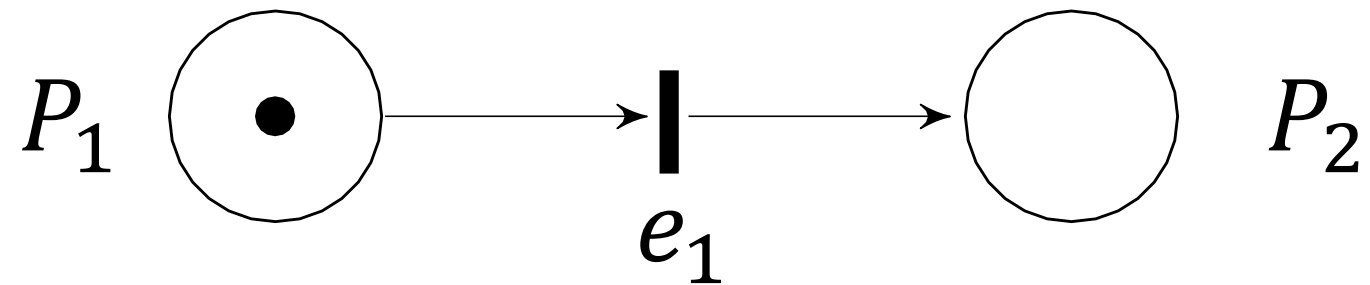
- Places labeled by configurations
- Transitions labeled by events
- 1-safe Petri-nets
- Can be implemented without packet races
  - (first part of the tutorial)
- Logical time bounds on when to change the configuration can be given [PLDI16]

# Event nets: Sequential Composition

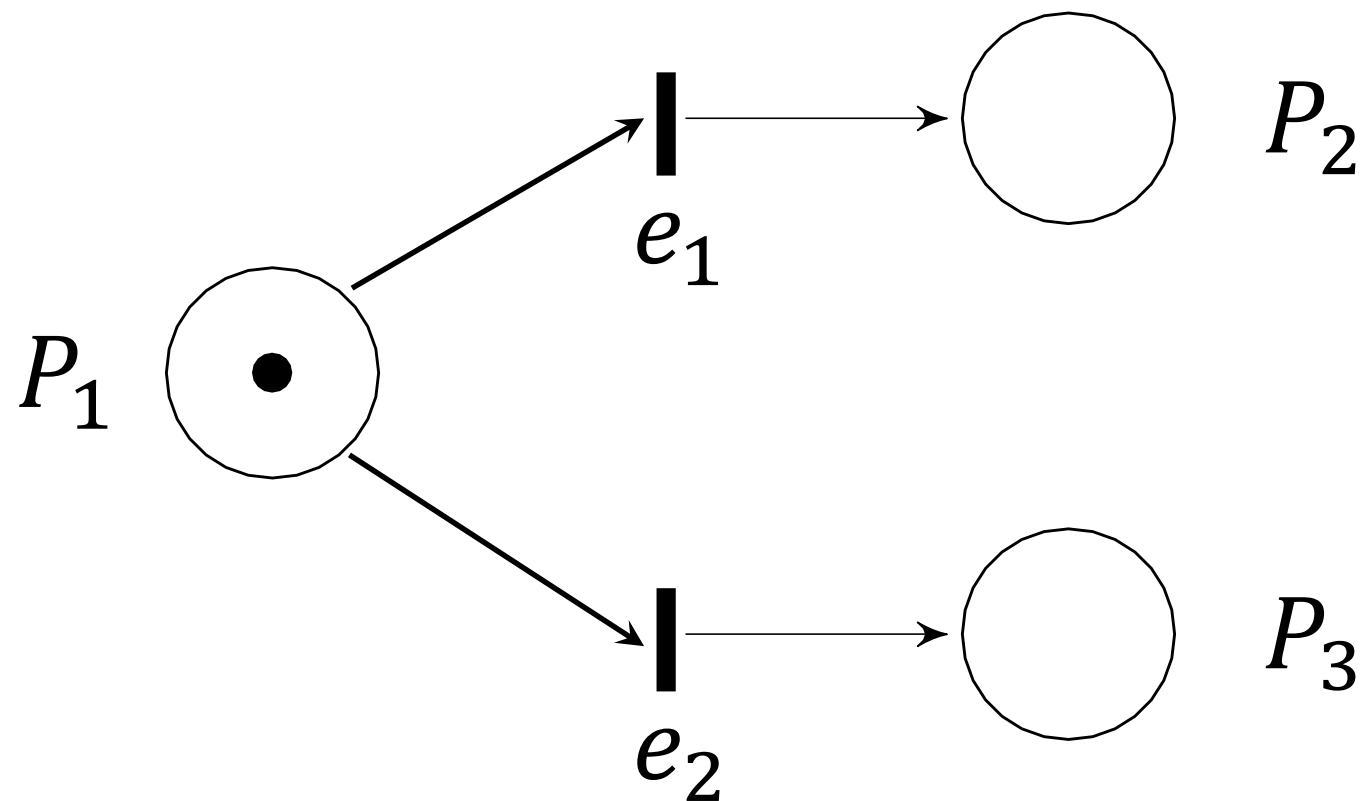




# Event nets: Parallel Composition



# Event nets: Conflicting event-updates

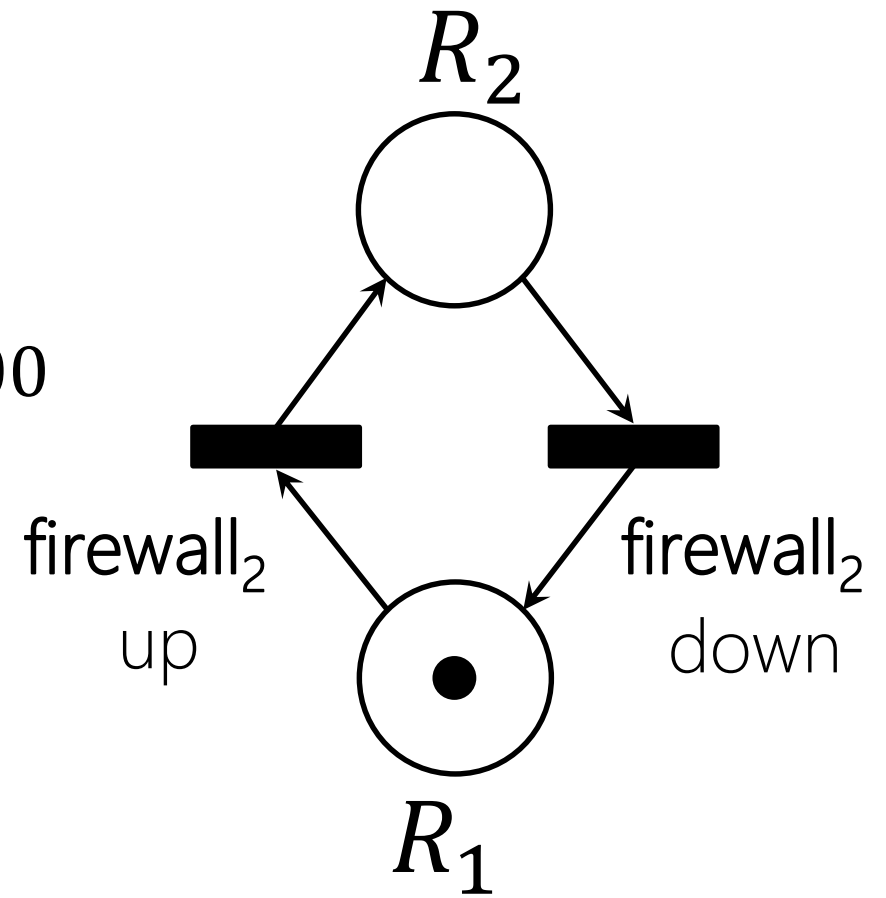
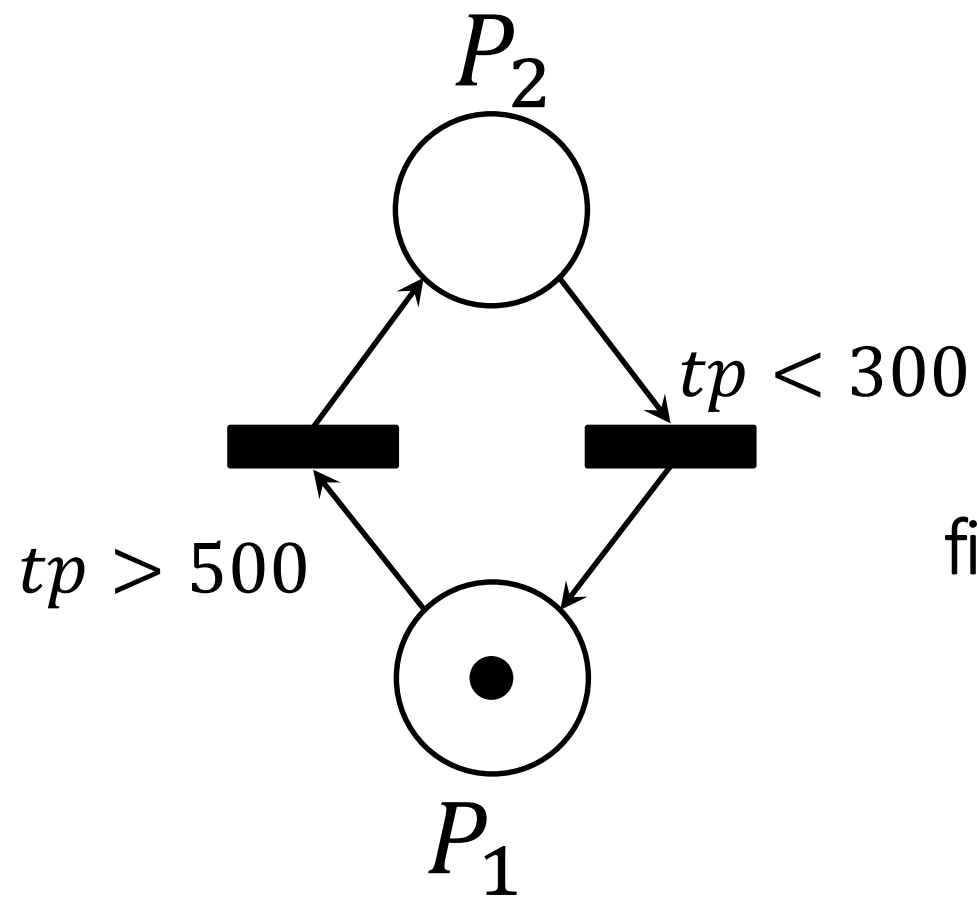
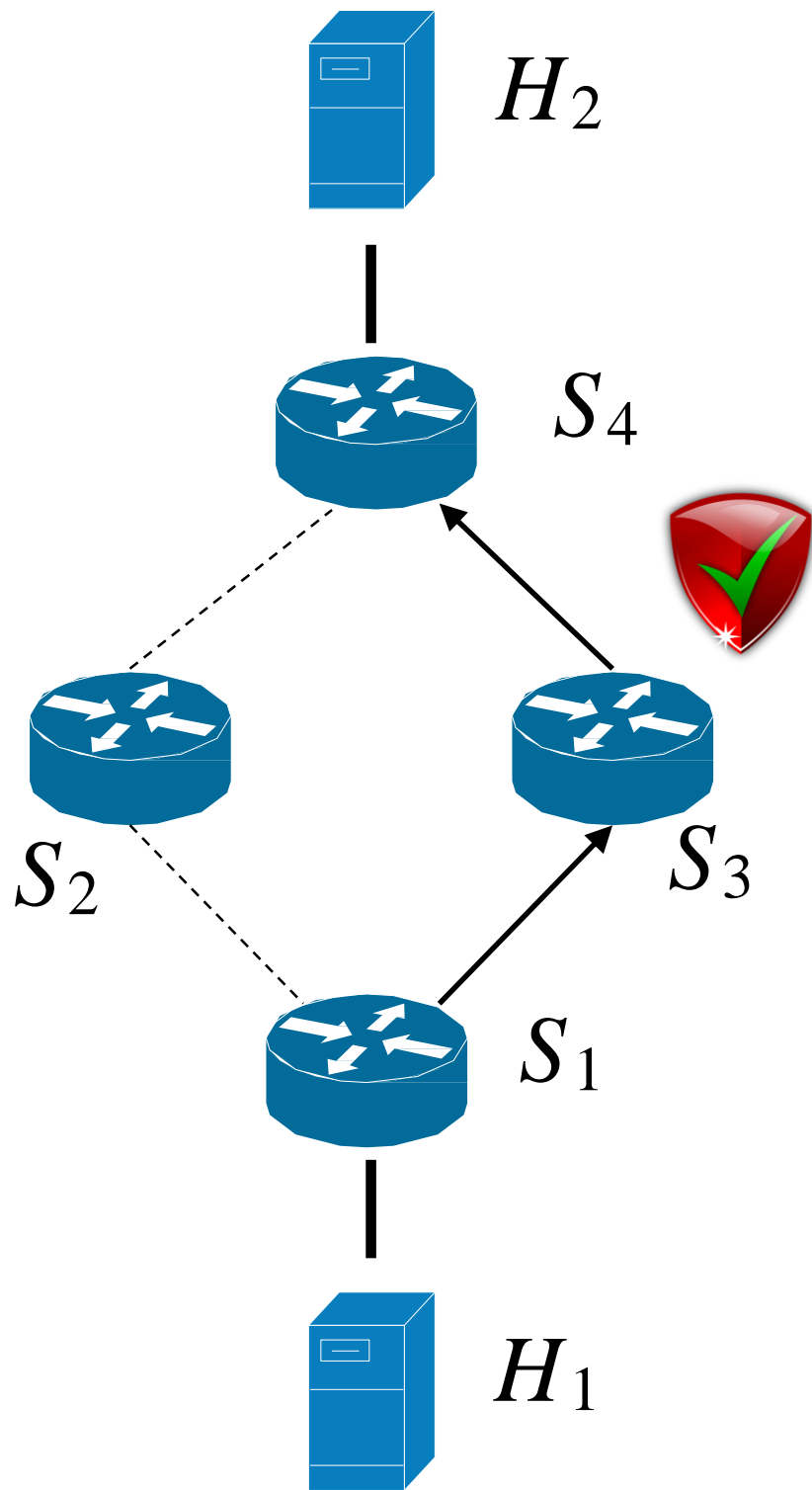


- **Locality condition:**  
which transition to take ( $e_1$  or  $e_2$ ) must be decided locally
- Otherwise availability cannot be maintained [PLDI16]  
(usual Consistency-Availability tension)

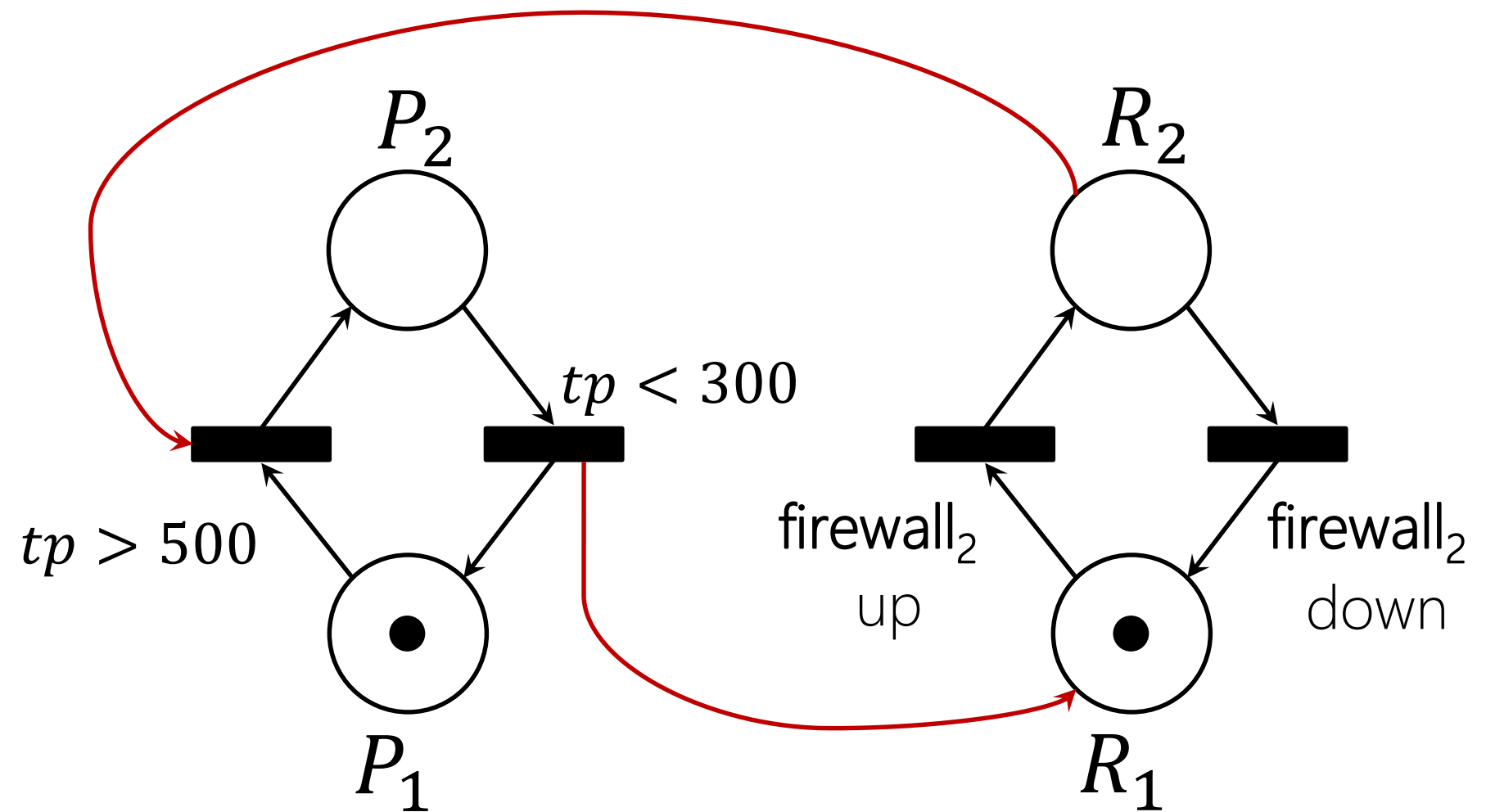
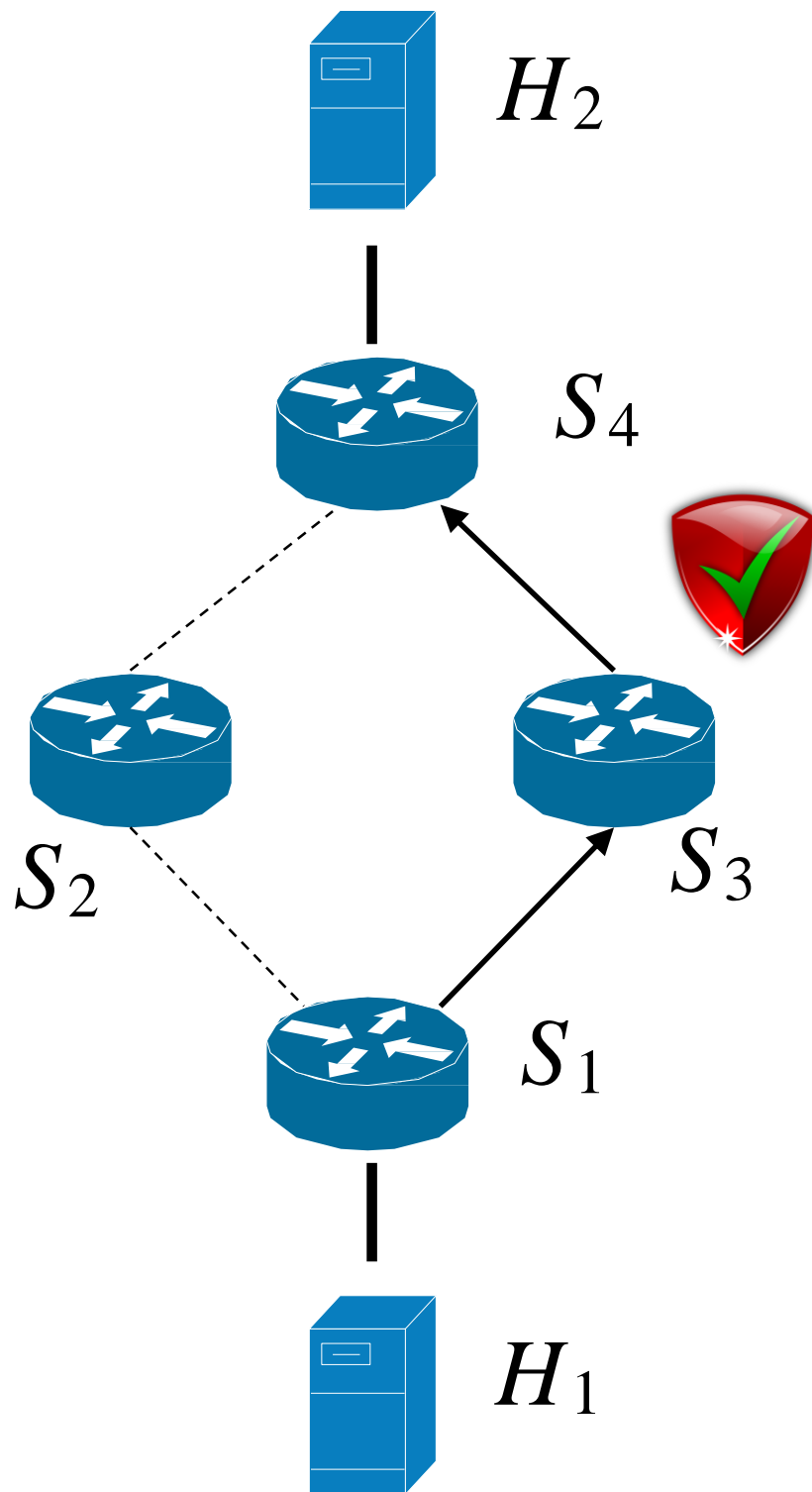
# Programming Model: Event nets

- 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

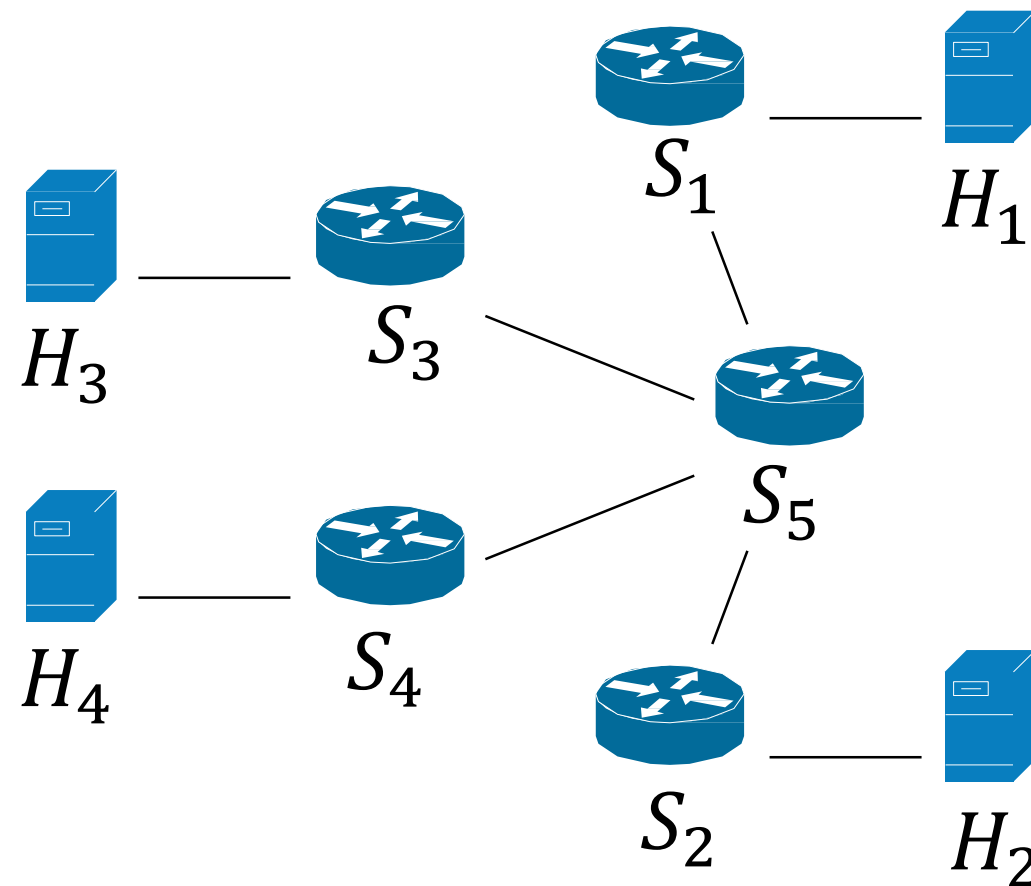


# Synchronization Synthesis

## Eliminating Controller Races

# Concurrent network programs

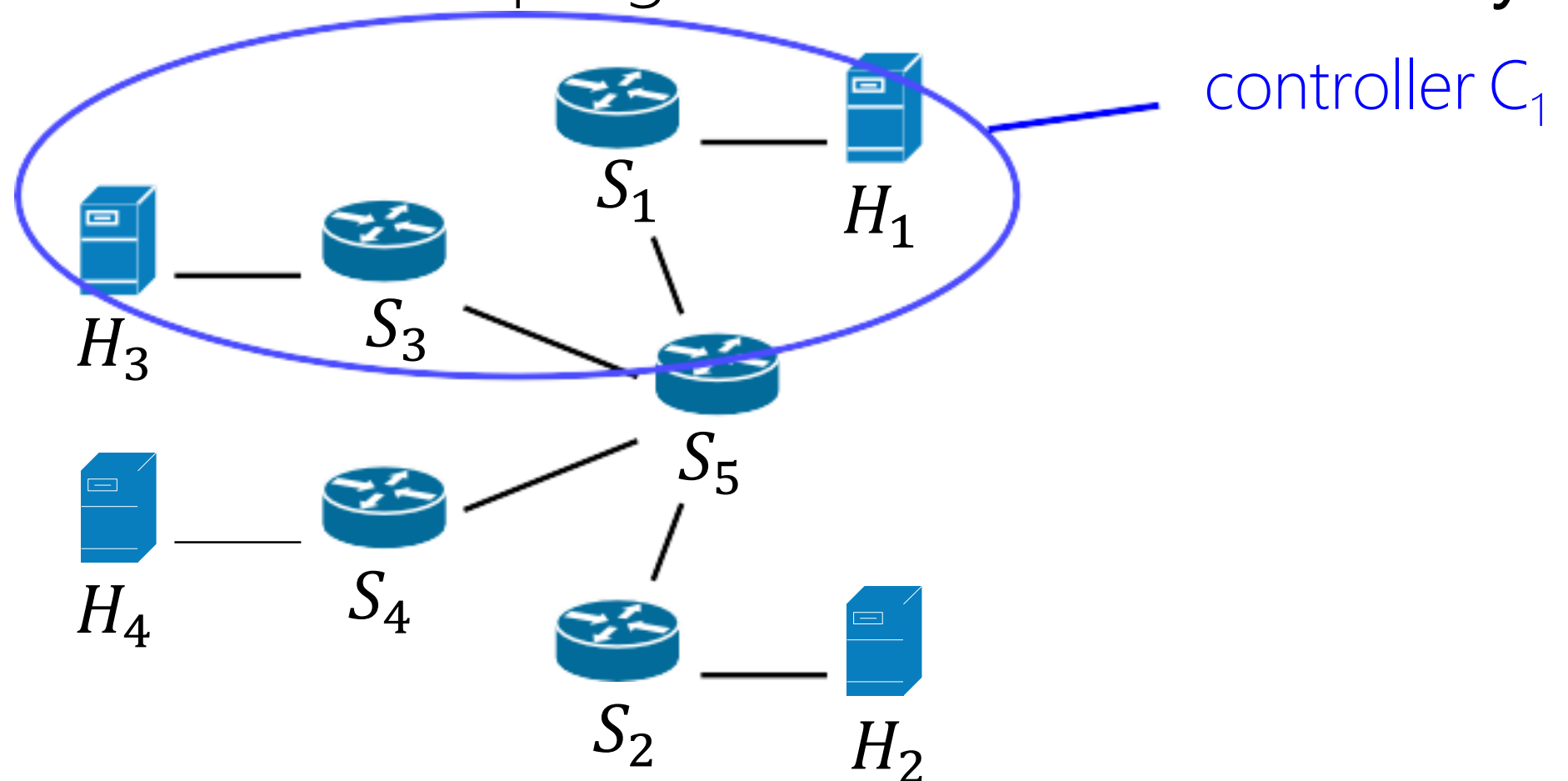
Correctness when network programs execute **concurrently**?





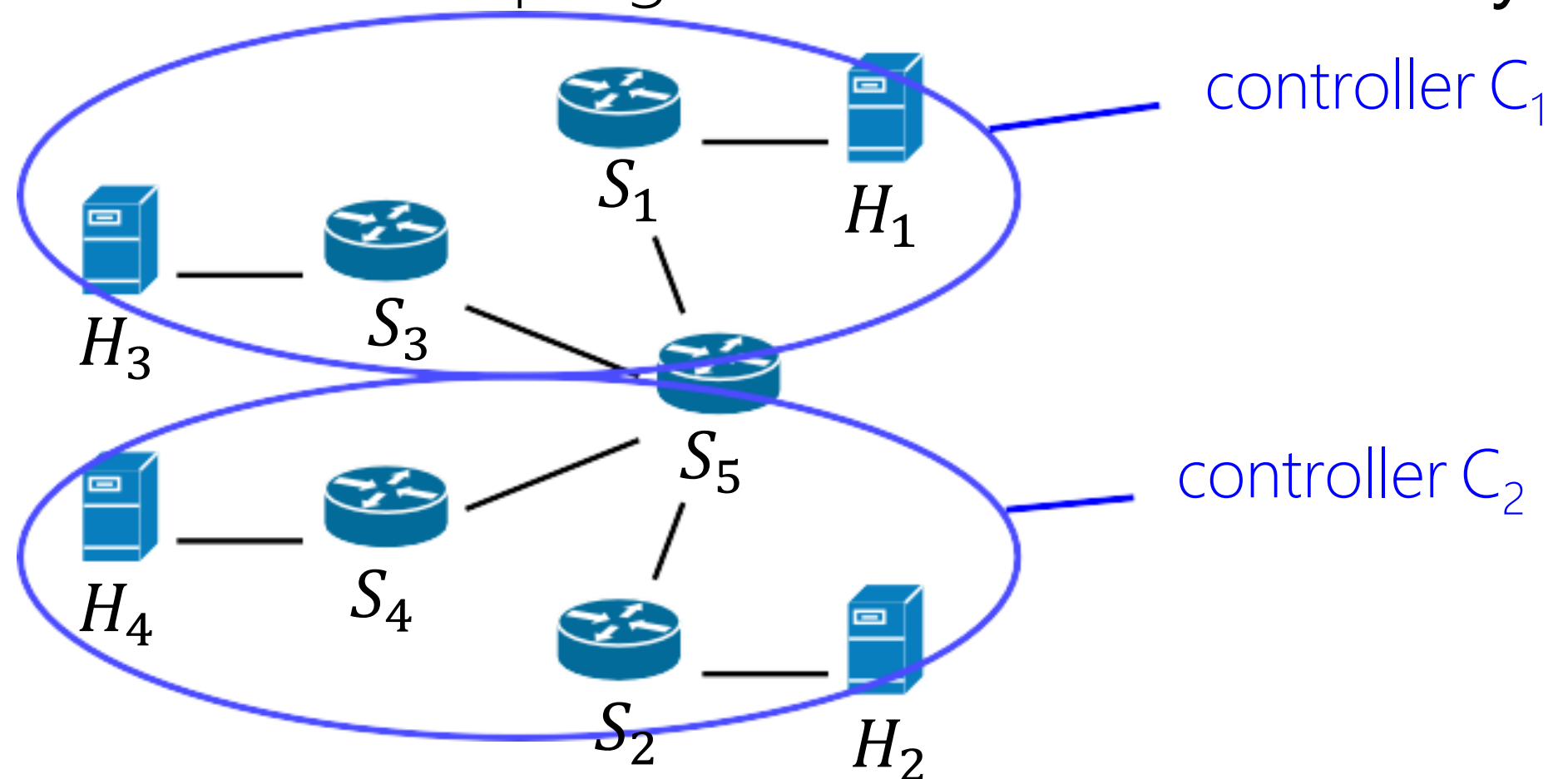
# Concurrent network programs

Correctness when network programs execute **concurrently**?



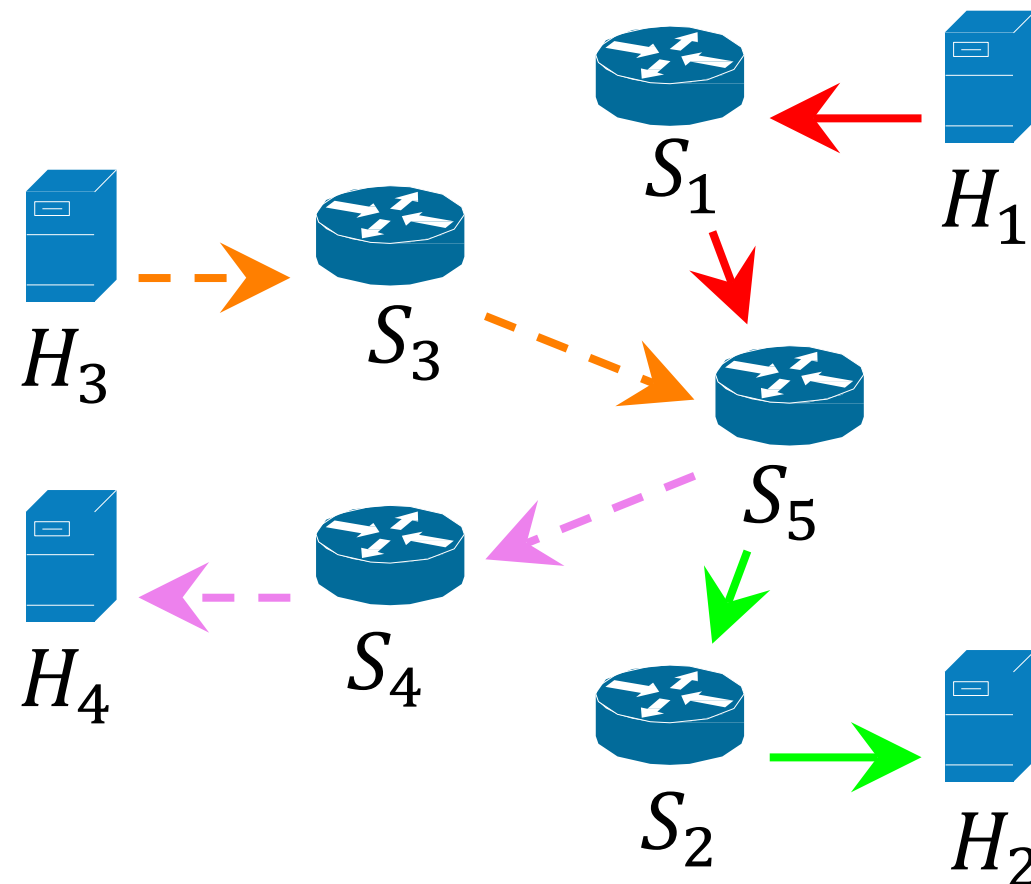
# Concurrent network programs

Correctness when network programs execute **concurrently**?



# Concurrent network programs

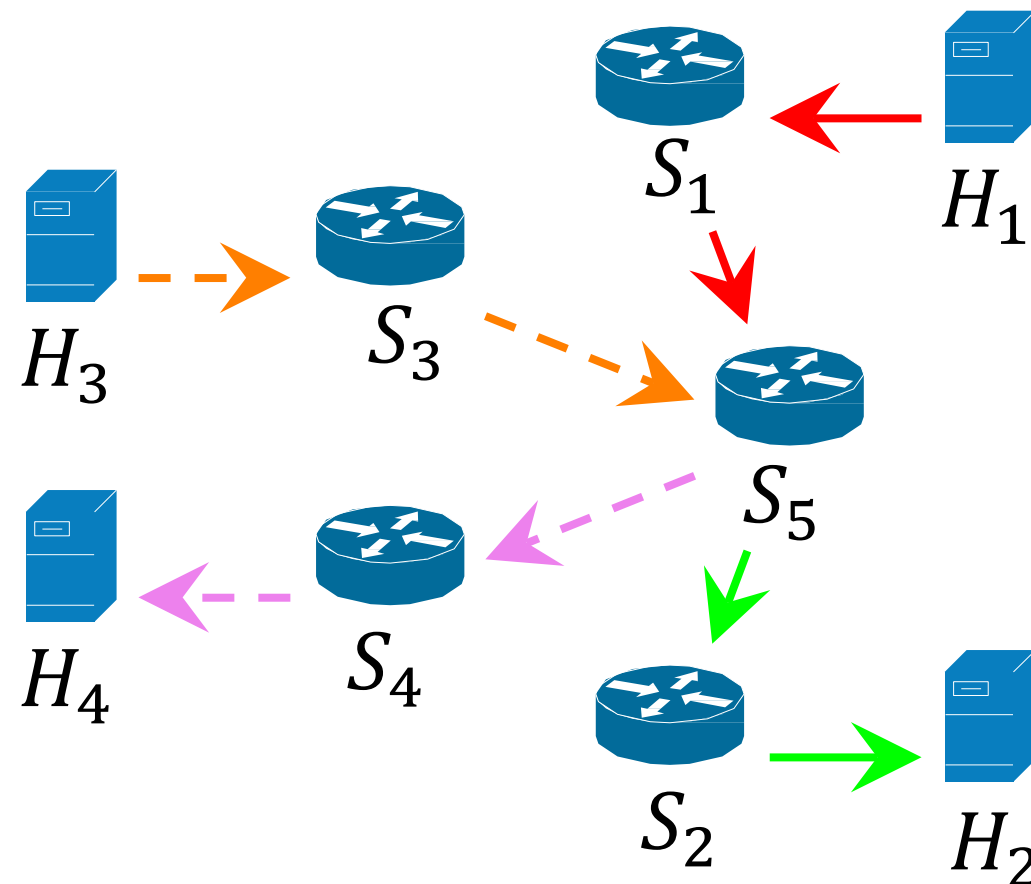
Correctness when network programs execute concurrently?



- 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

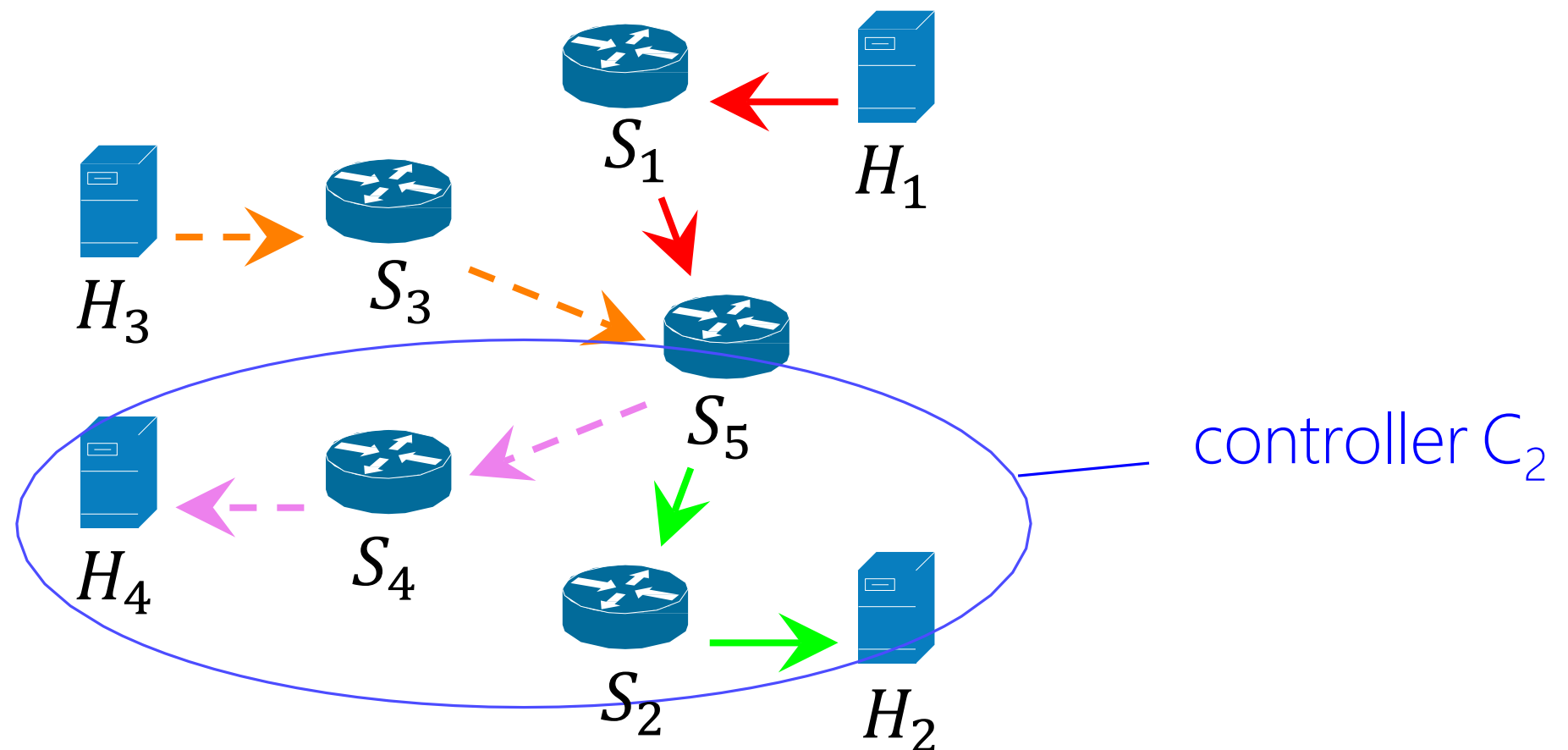
Correctness when network programs execute concurrently?



- 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

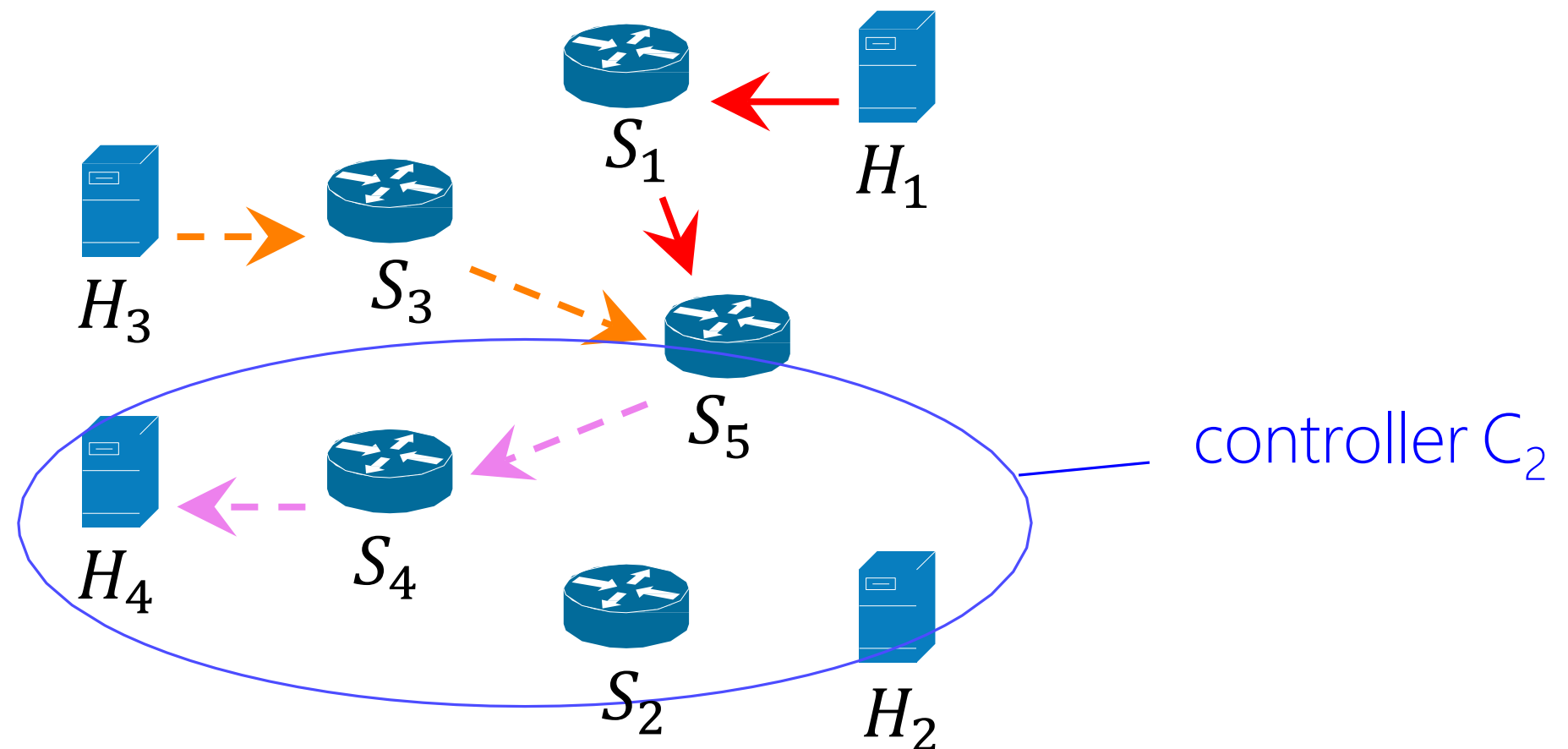
Correctness when network programs execute **concurrently**?



- 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

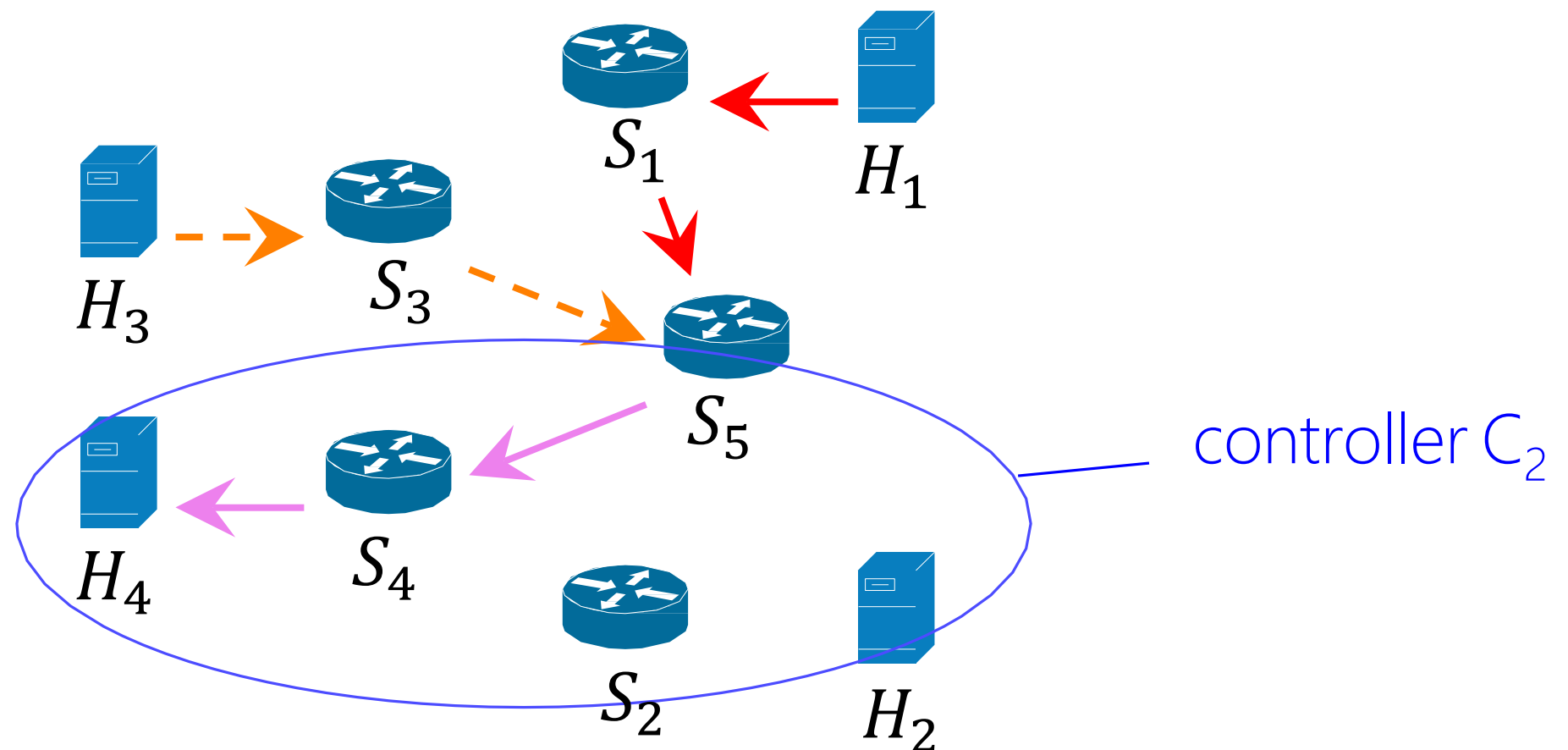
Correctness when network programs execute **concurrently**?



- 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**?



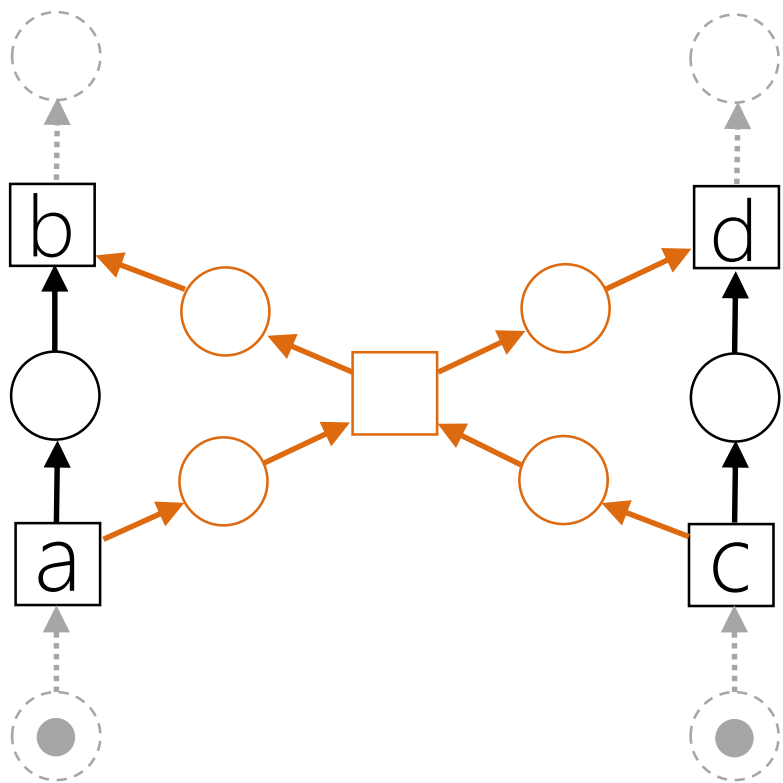
- Network operator wants to take down the  $H_1 \rightarrow H_2$  forwarding rules, and install  $H_3 \rightarrow H_4$  rules
- Example property: *isolation* — *all packets entering the network from  $H_1$  must exit at  $H_2$ .*
- Potential bug: *controller race*



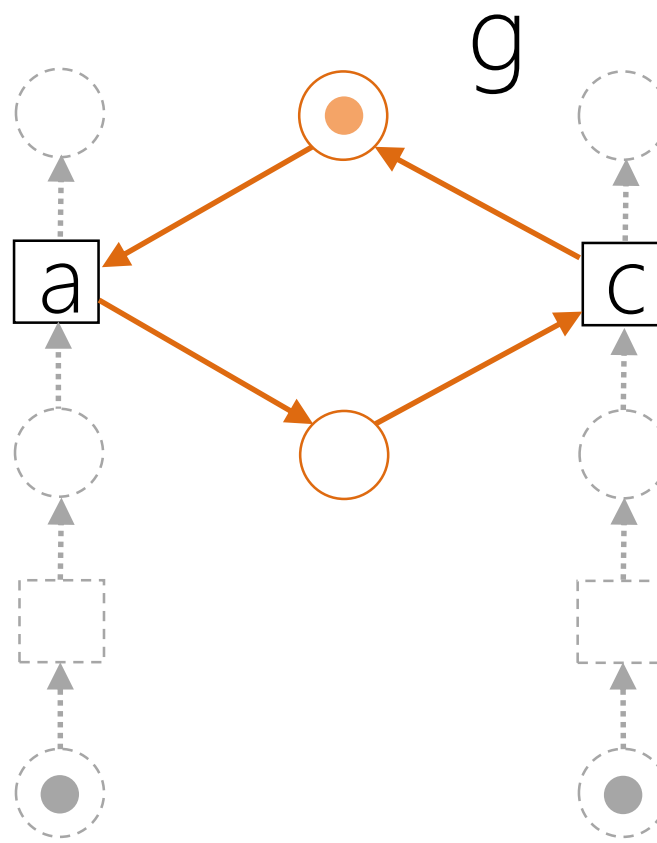
# Adding synchronization

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

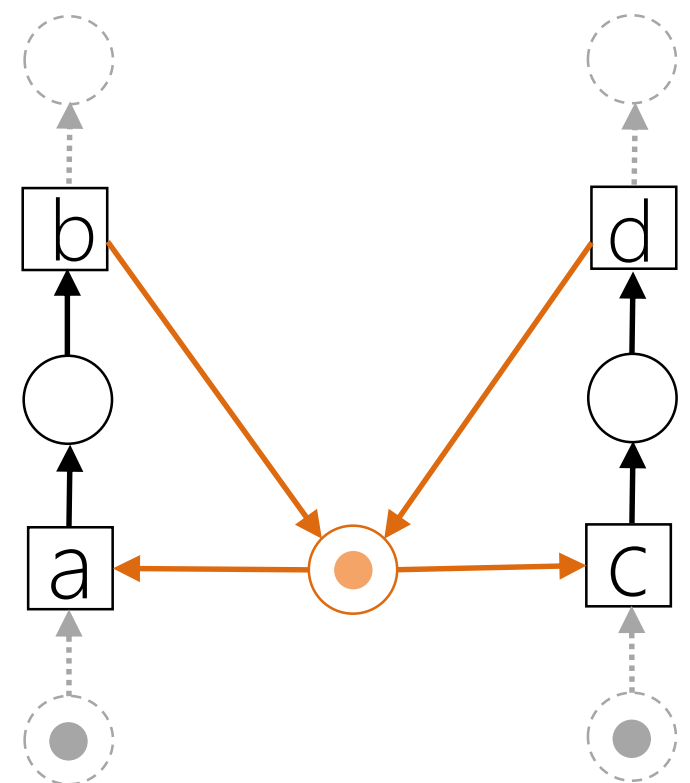
barrier



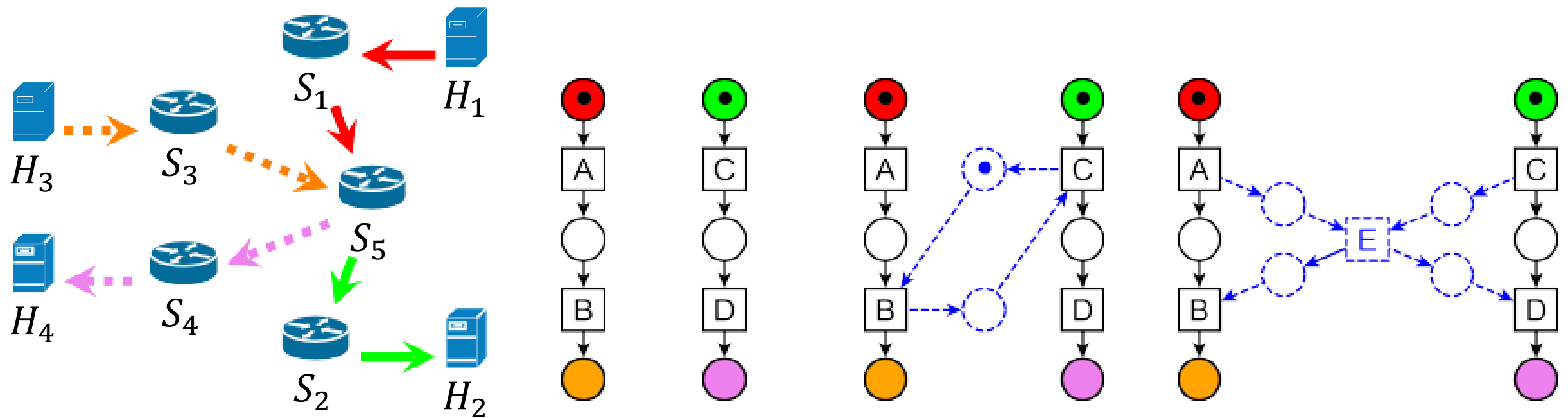
ordering



mutex

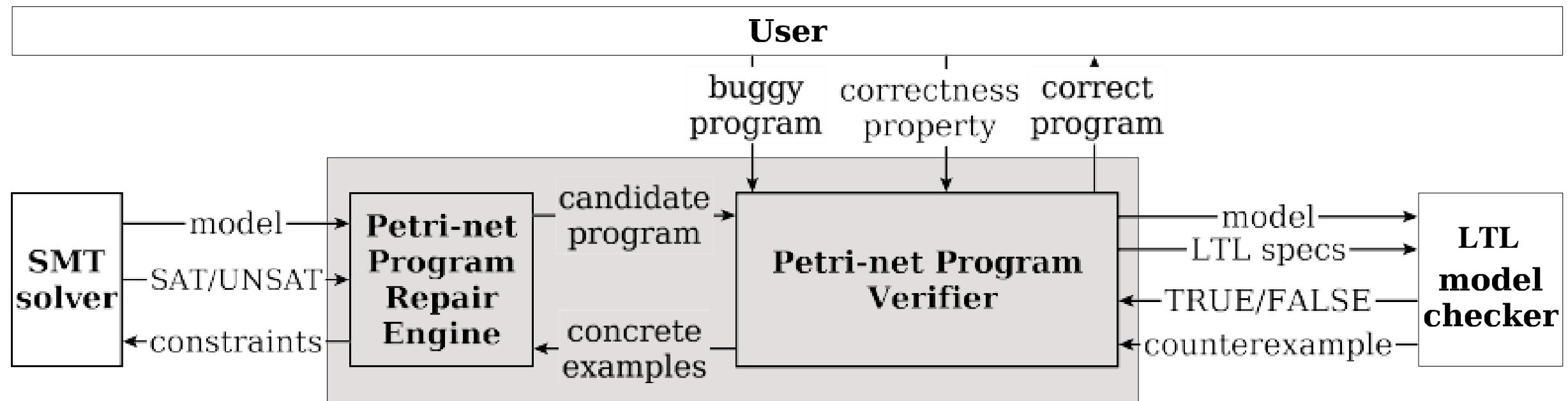


# Synthesizing petri-net programs



- $\varphi_1$ : no packet originating at  $H_1$  should arrive at  $H_4$
- $\varphi_2$ : no packet originating at  $H_3$  should arrive at  $H_2$
- First counterexample:  $[C, D]$ , because  $\{\bullet \text{ (red)}, \bullet \text{ (purple)}\}$  violates the spec
- Second counterexample:  $[A, B]$ , because  $\{\bullet \text{ (orange)}, \bullet \text{ (green)}\}$  violates the spec

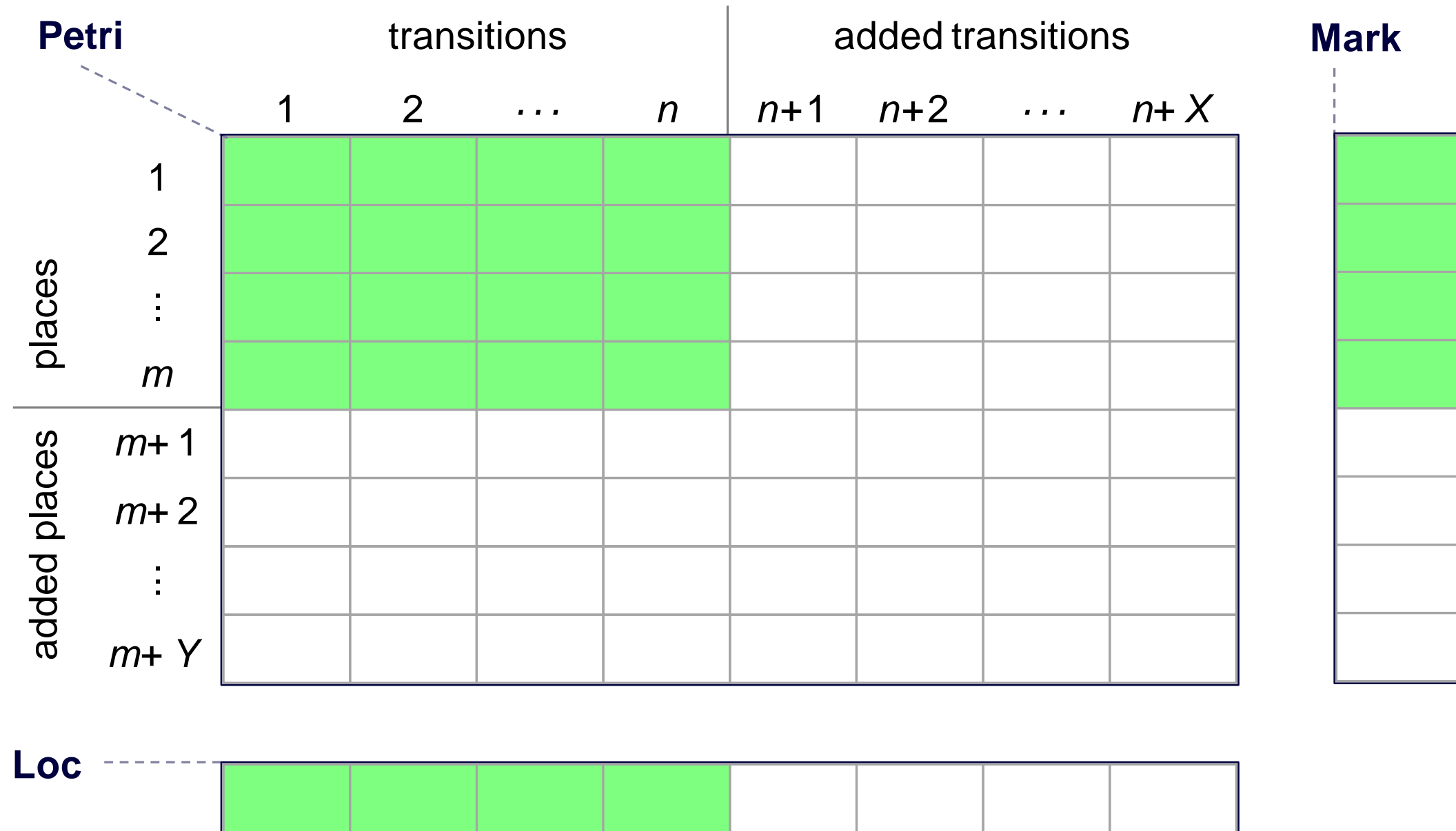
# Synthesizer Architecture



- 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

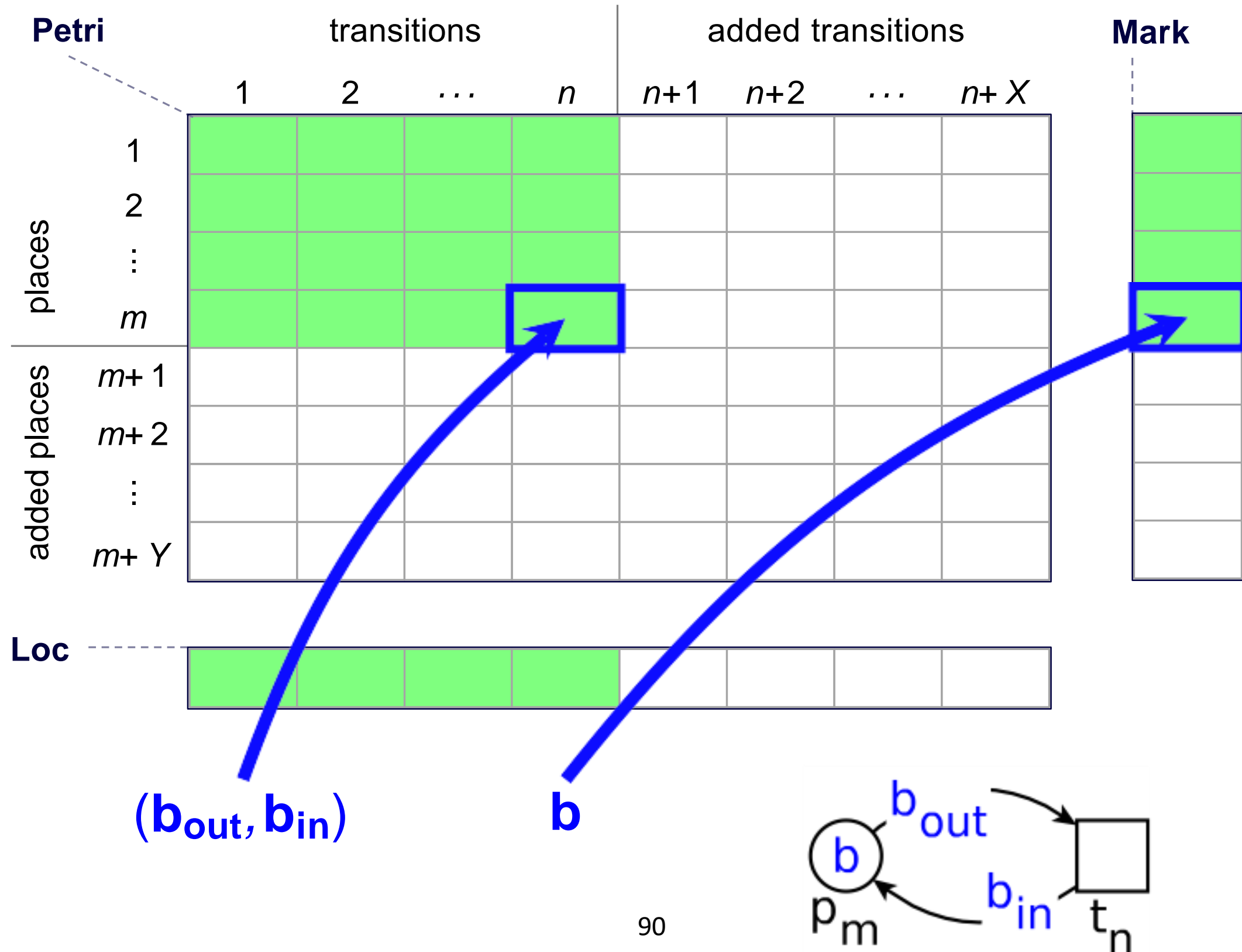
# Petri-net Synthesis Engine

SMT encoding for Petri-net programs:



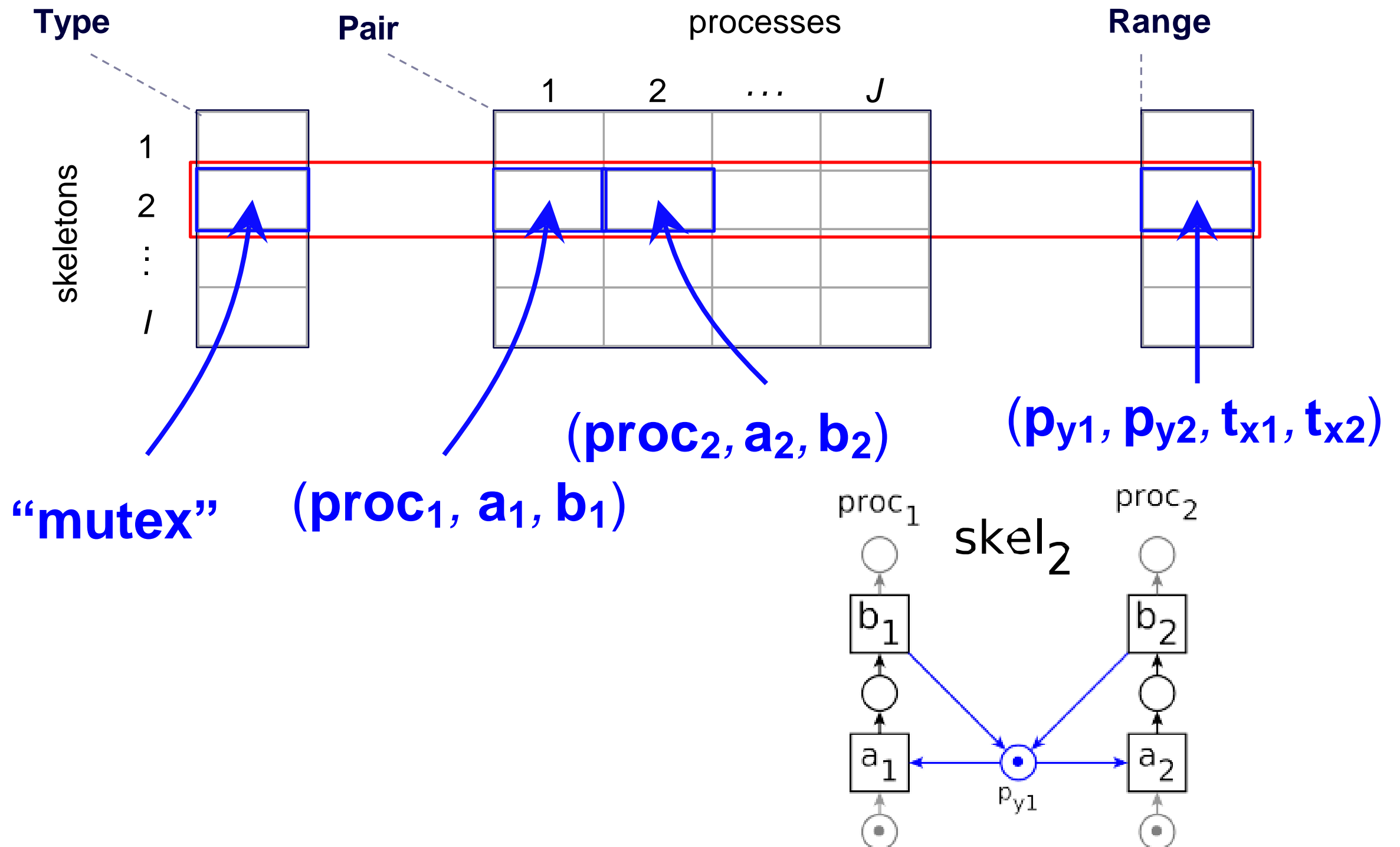
# Petri-net Synthesis Engine

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

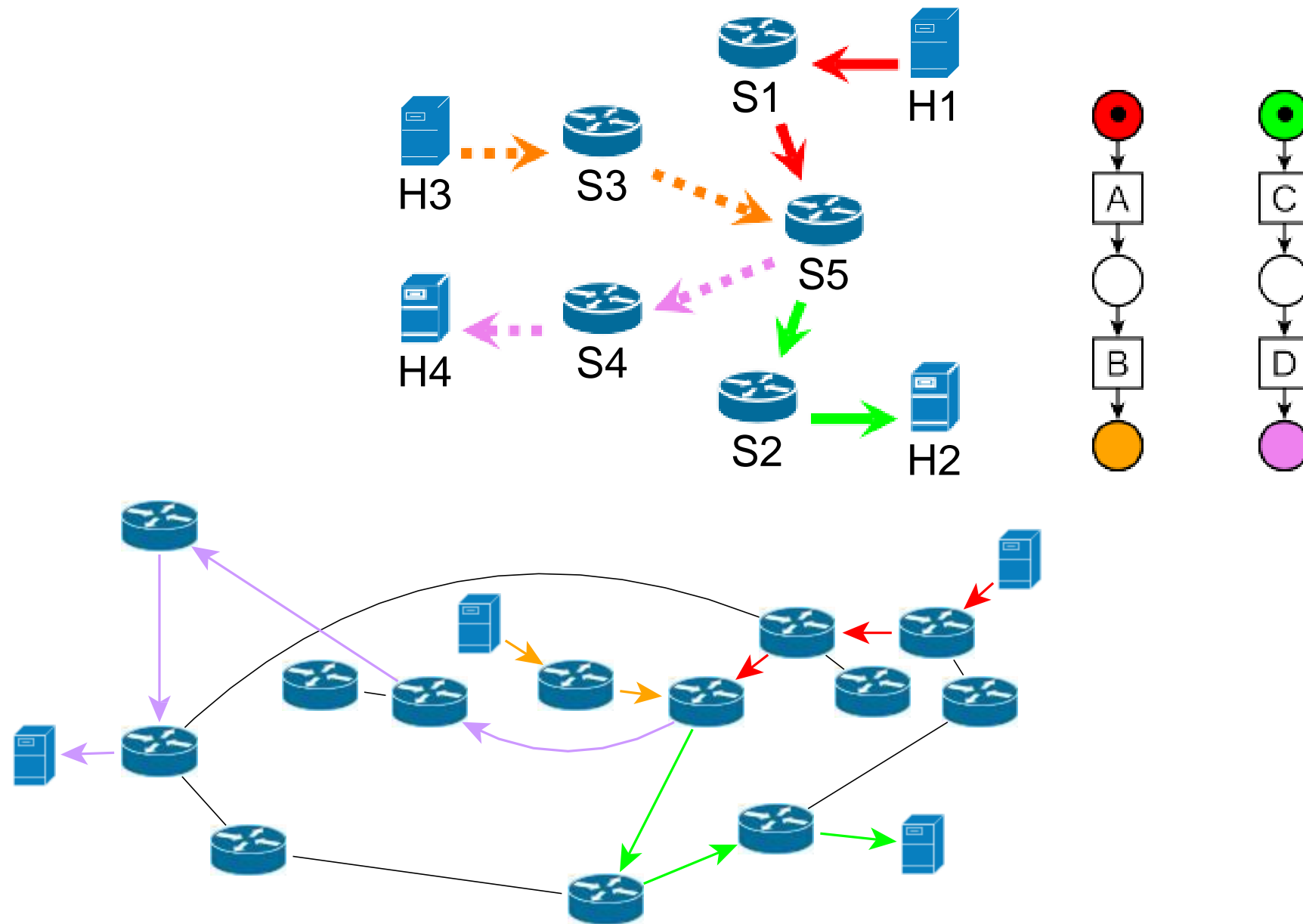
# Experimental results - expressiveness

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

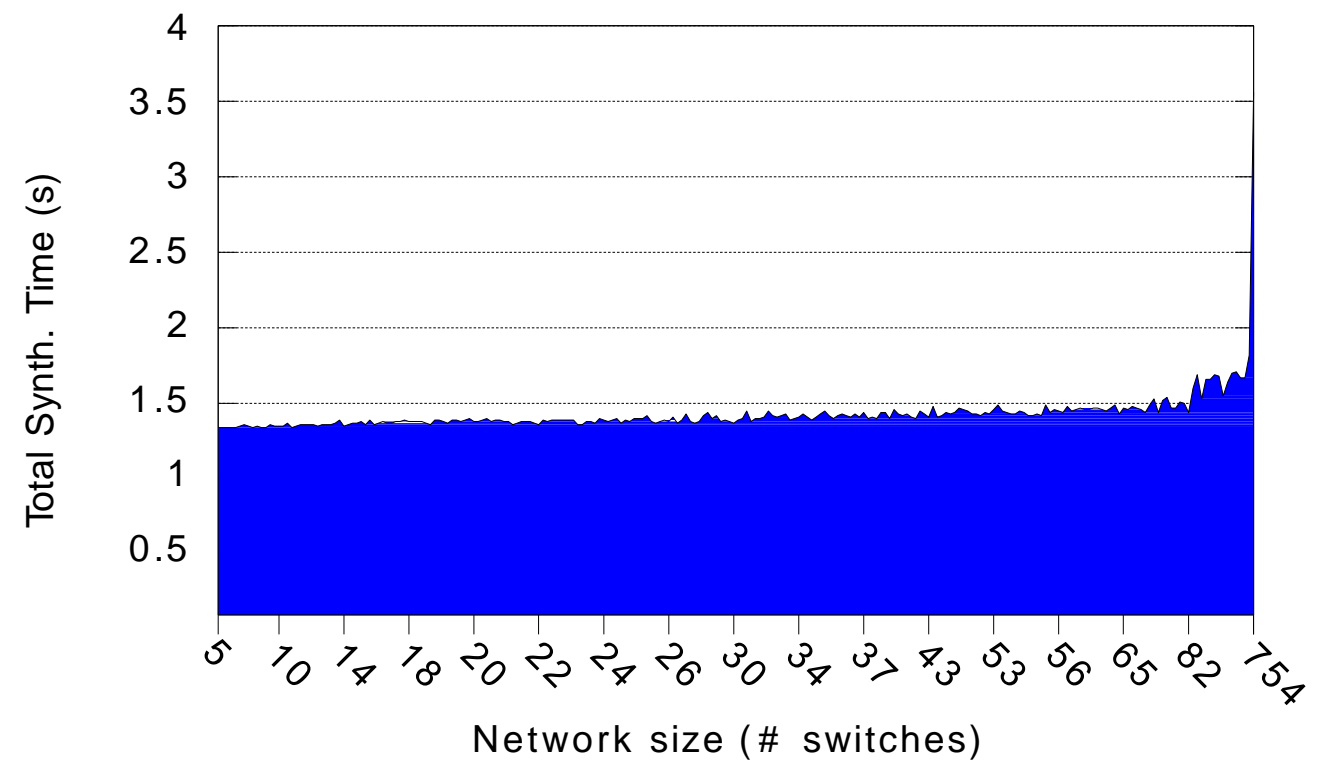
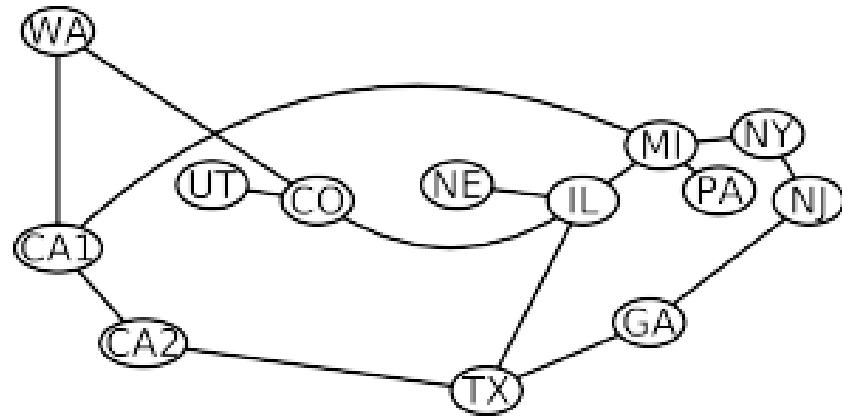
We scaled up the topology on the previously-discussed Isolation example



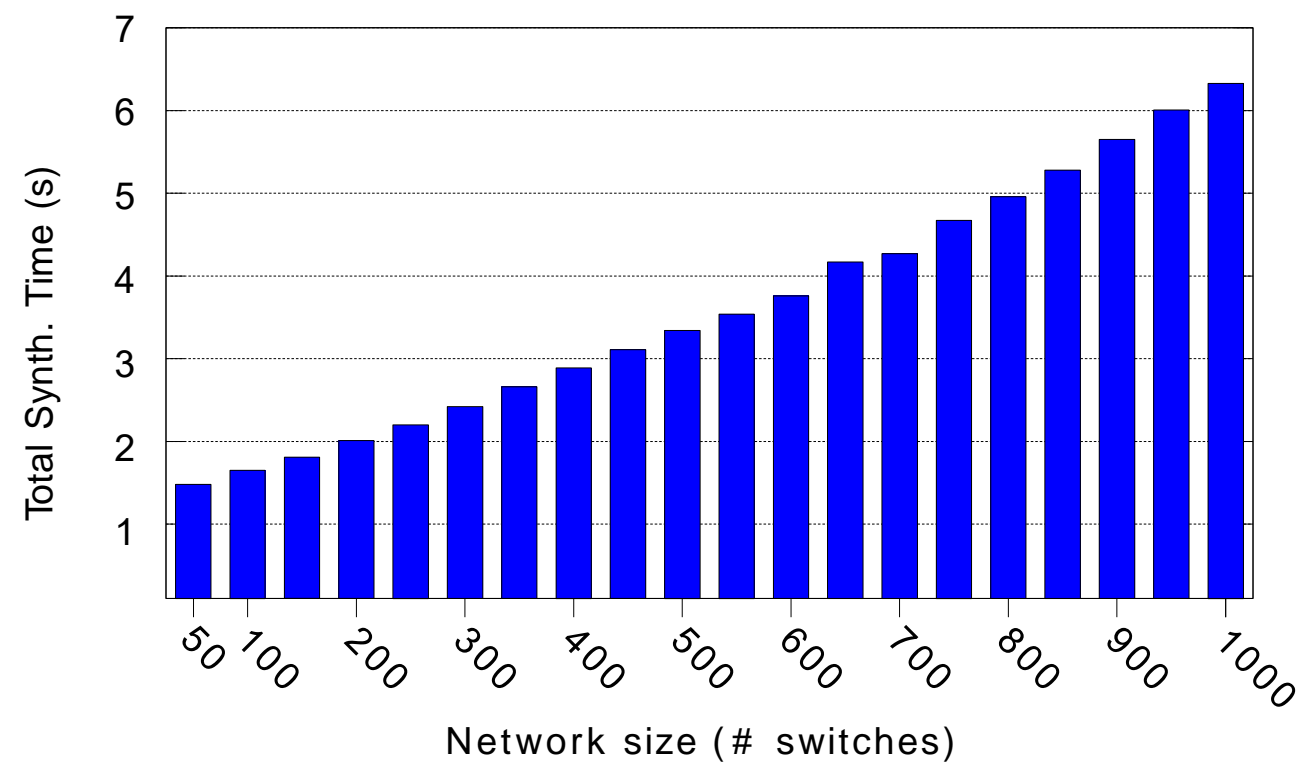
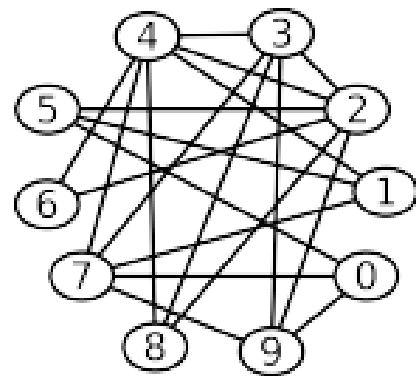
We then measured total synthesizer runtime versus topology size



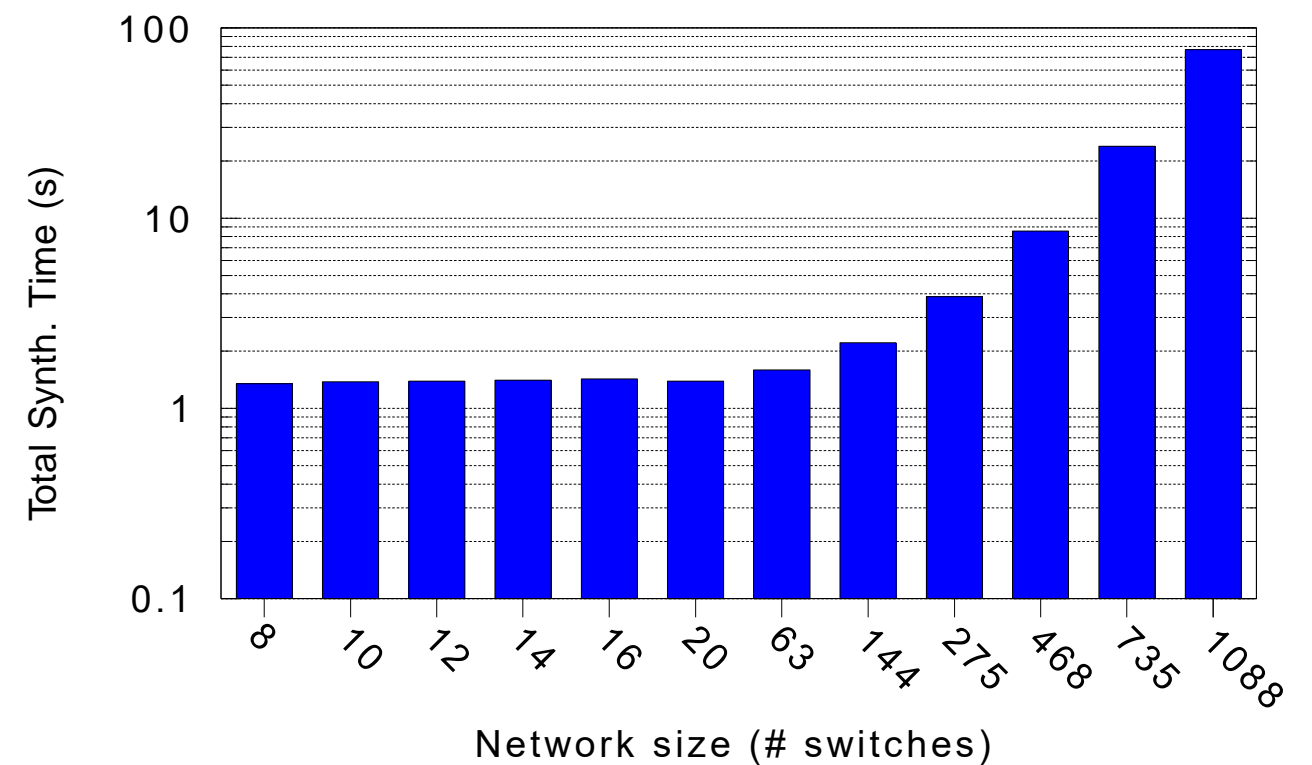
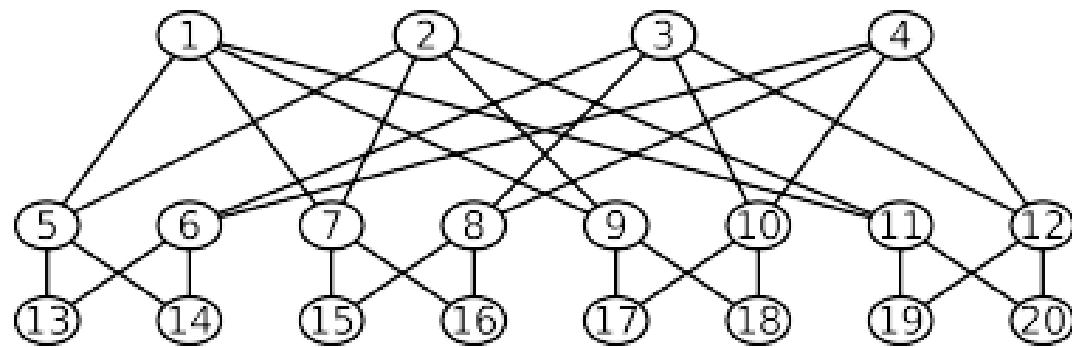
# Experimental results – Topology Zoo



# Experimental results – Small World

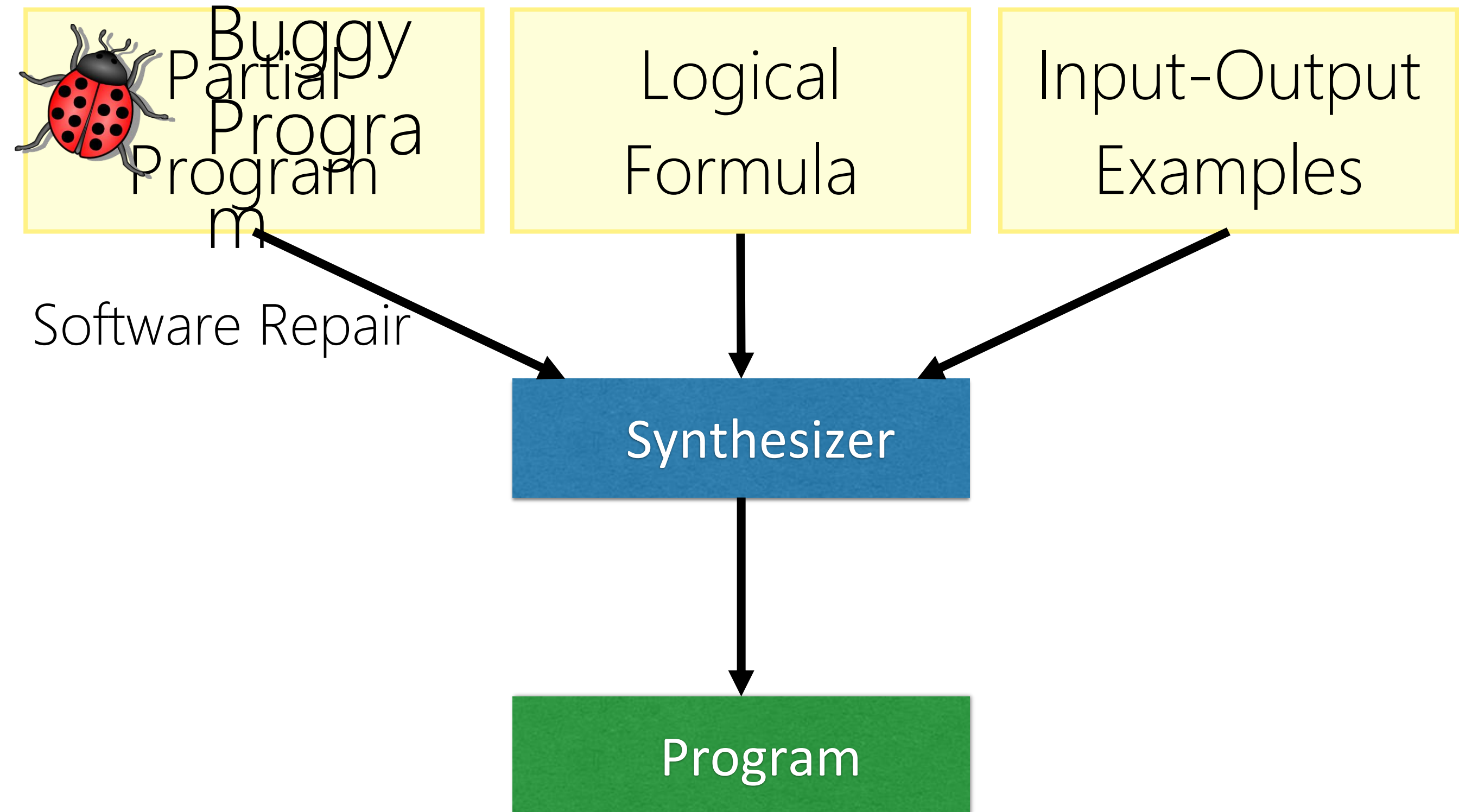


# Experimental results – FatTree

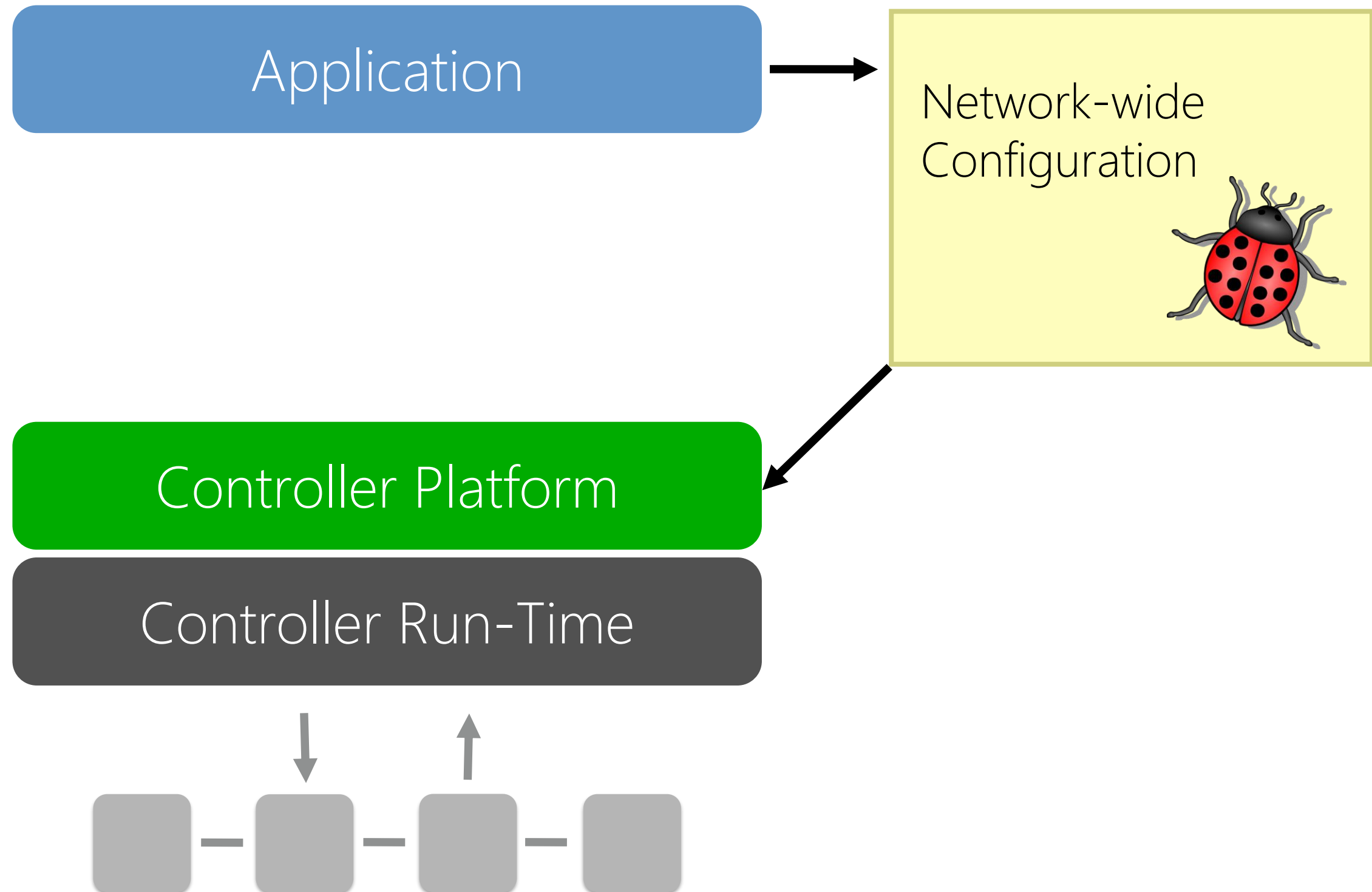


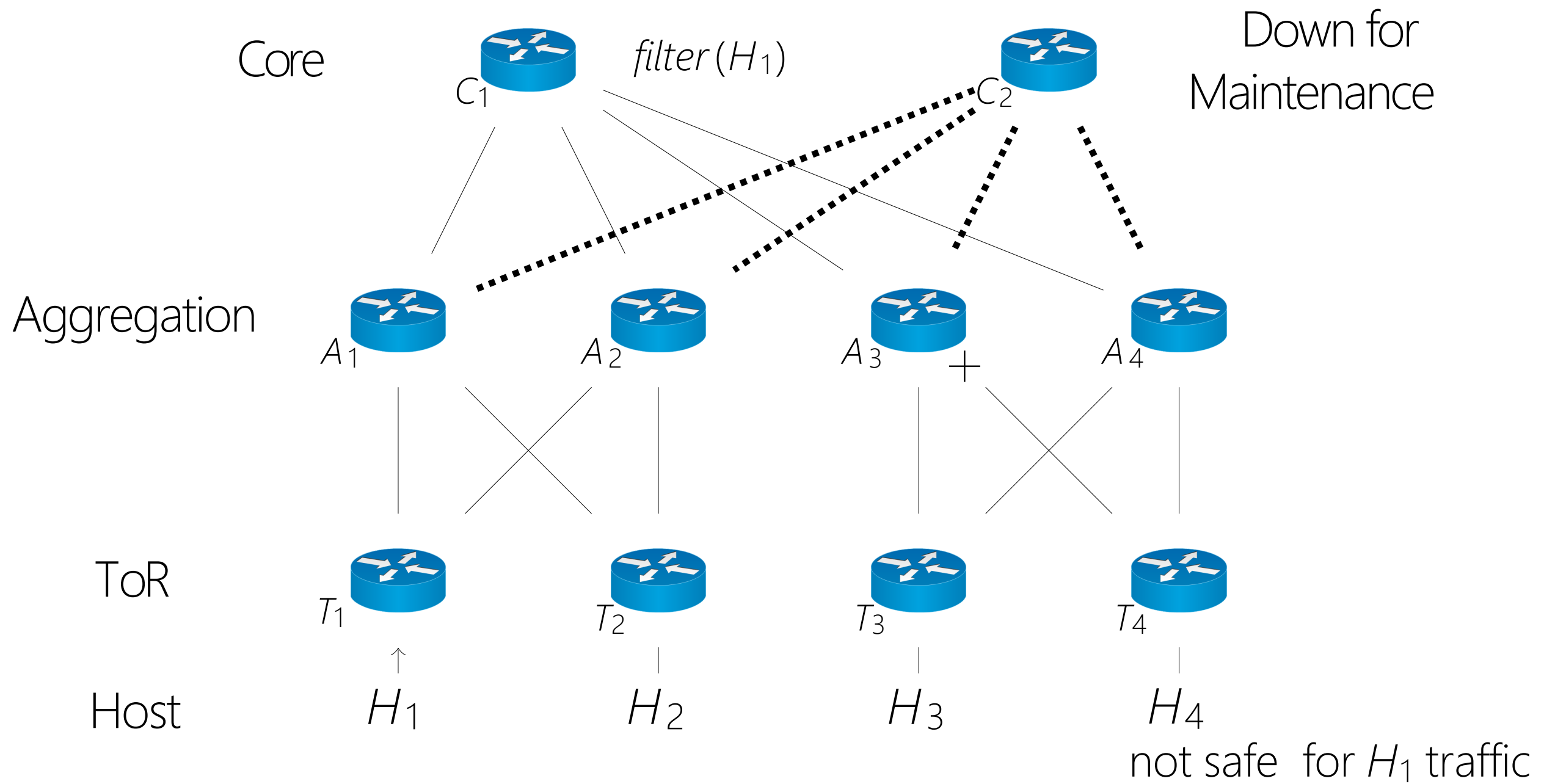
# Optimizing Horn Solvers for Network Repair

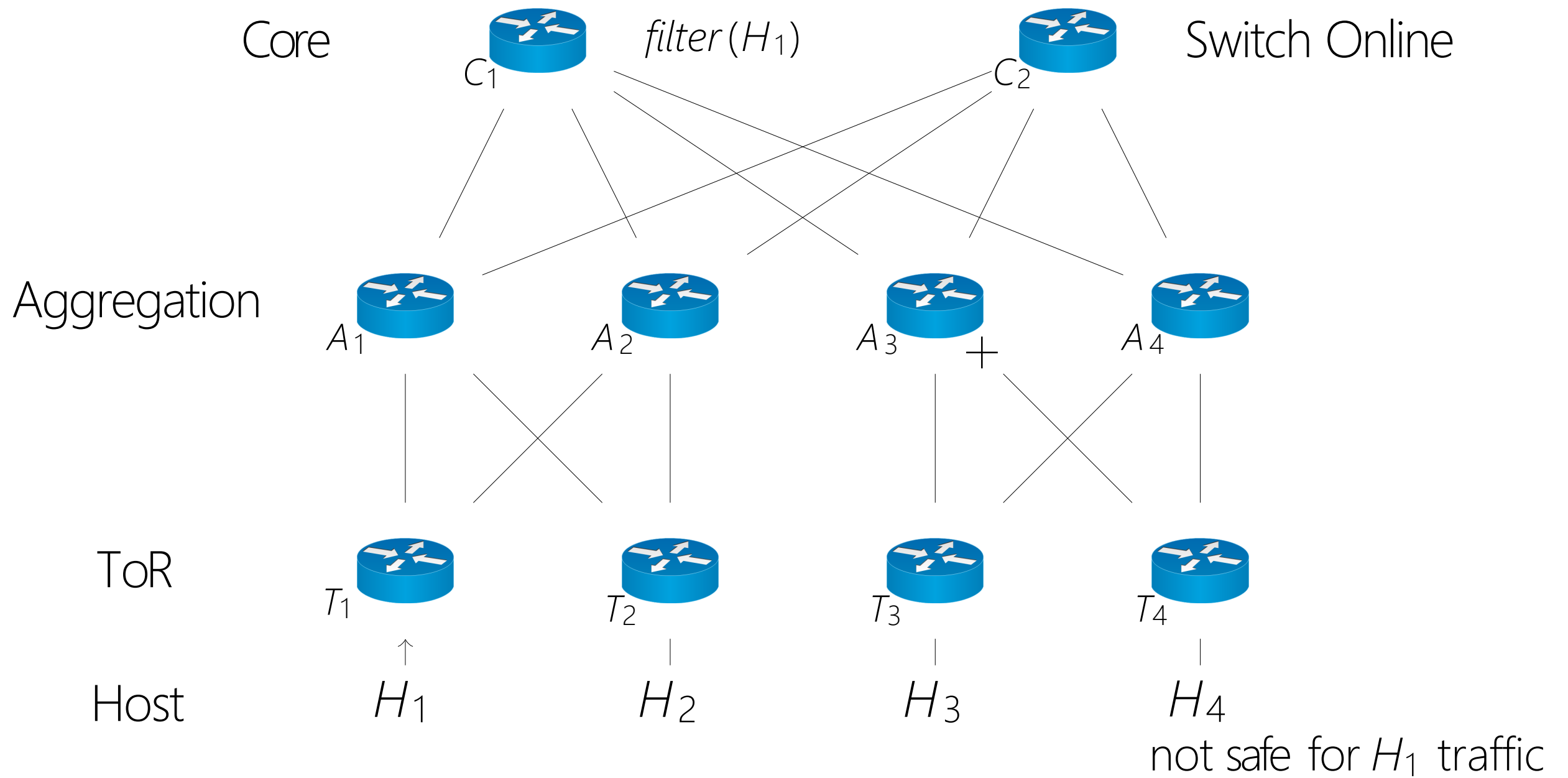
# Software Synthesis



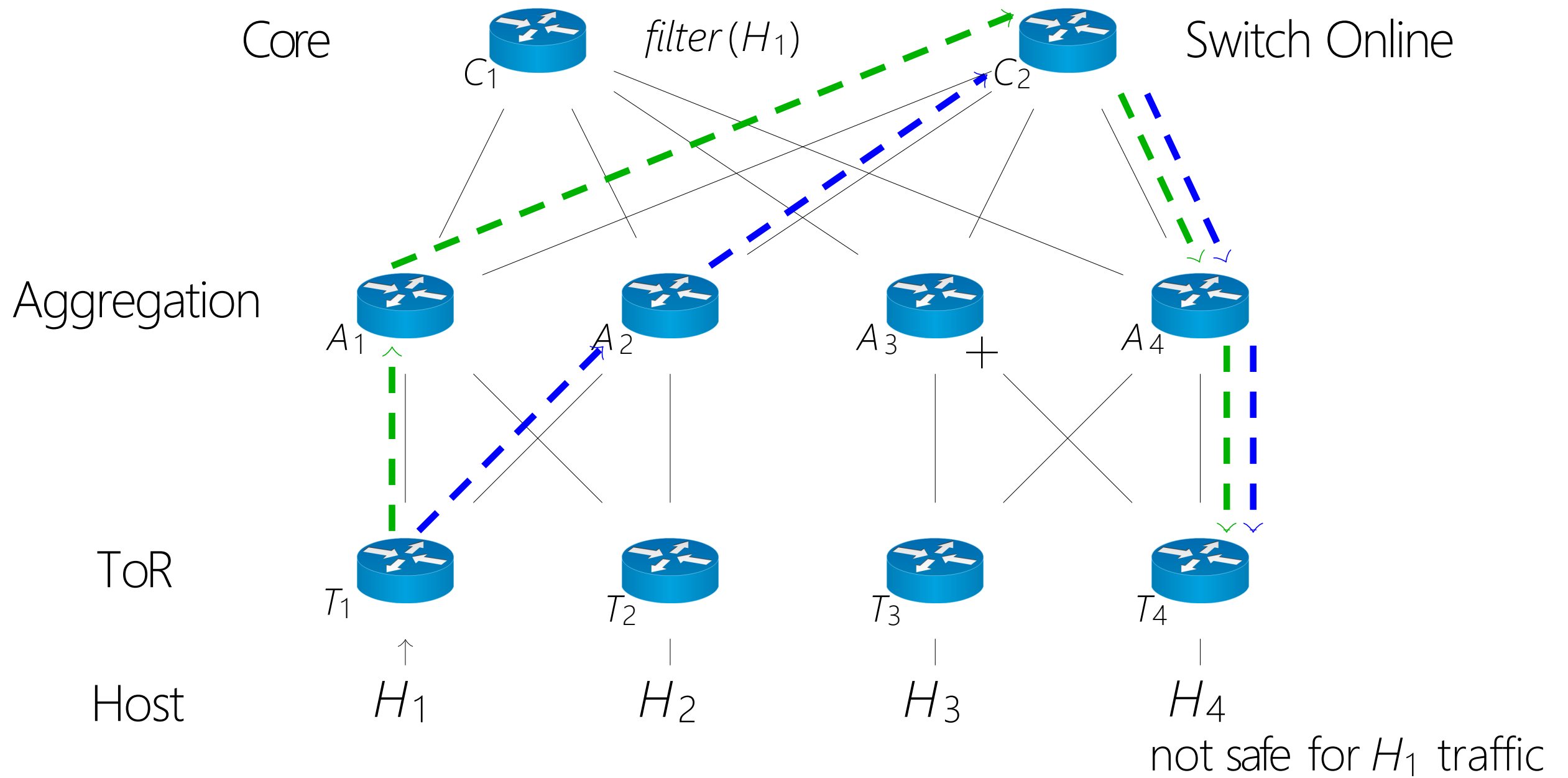
# SDN with Buggy Configuration







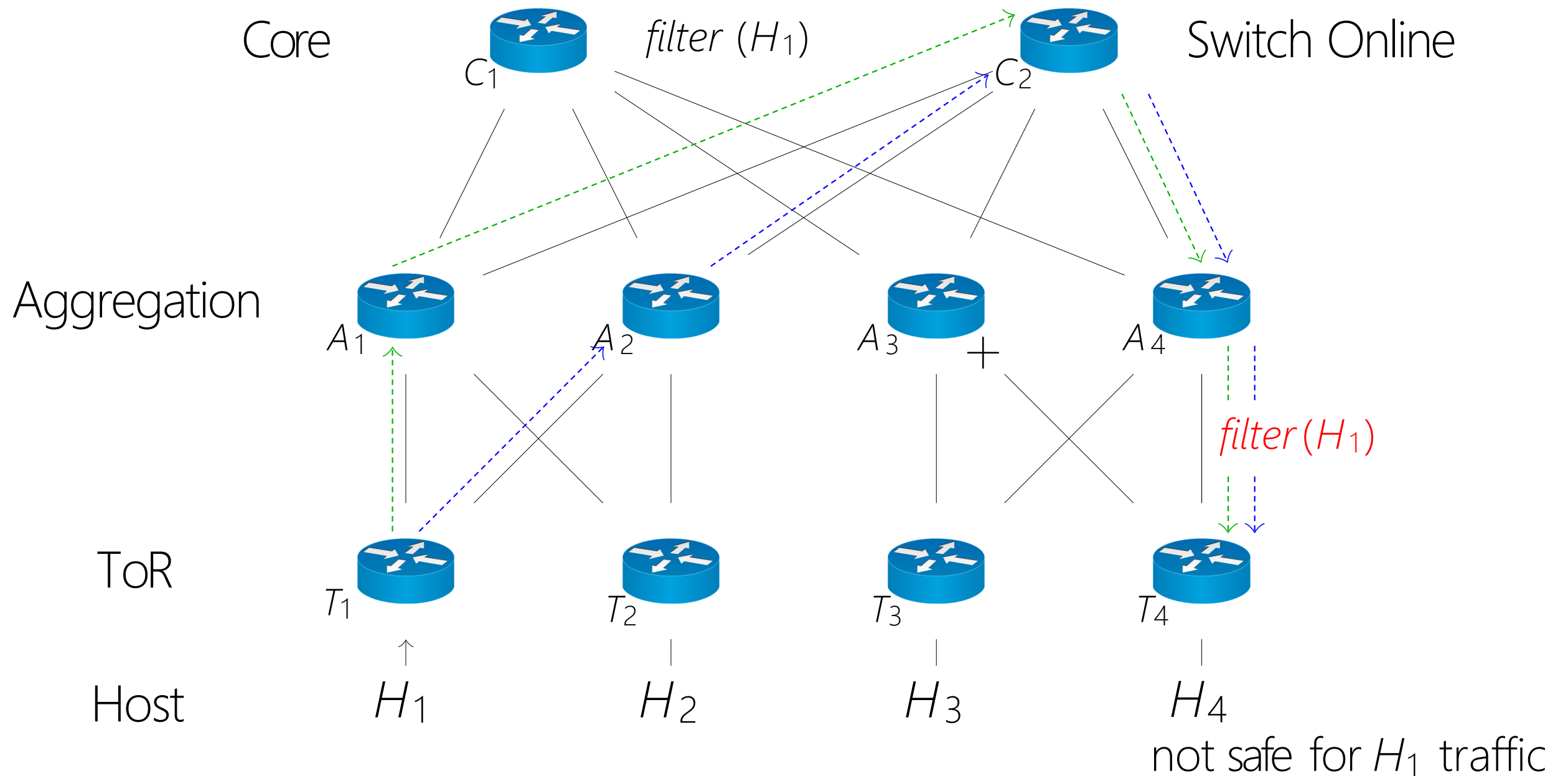








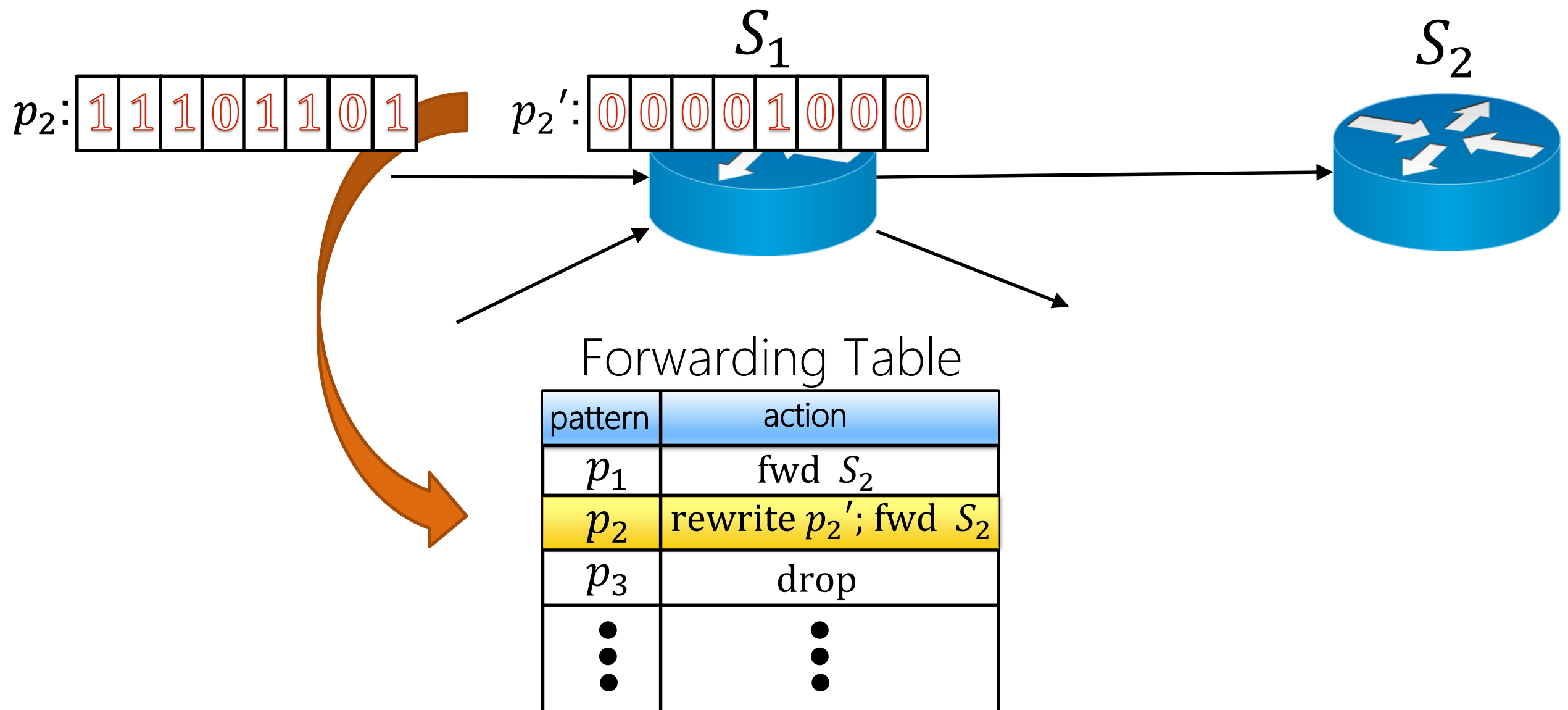




- How can we return back to safety by adding filters on links?
- There are several possible repair solutions
- Interested in **best** solutions:

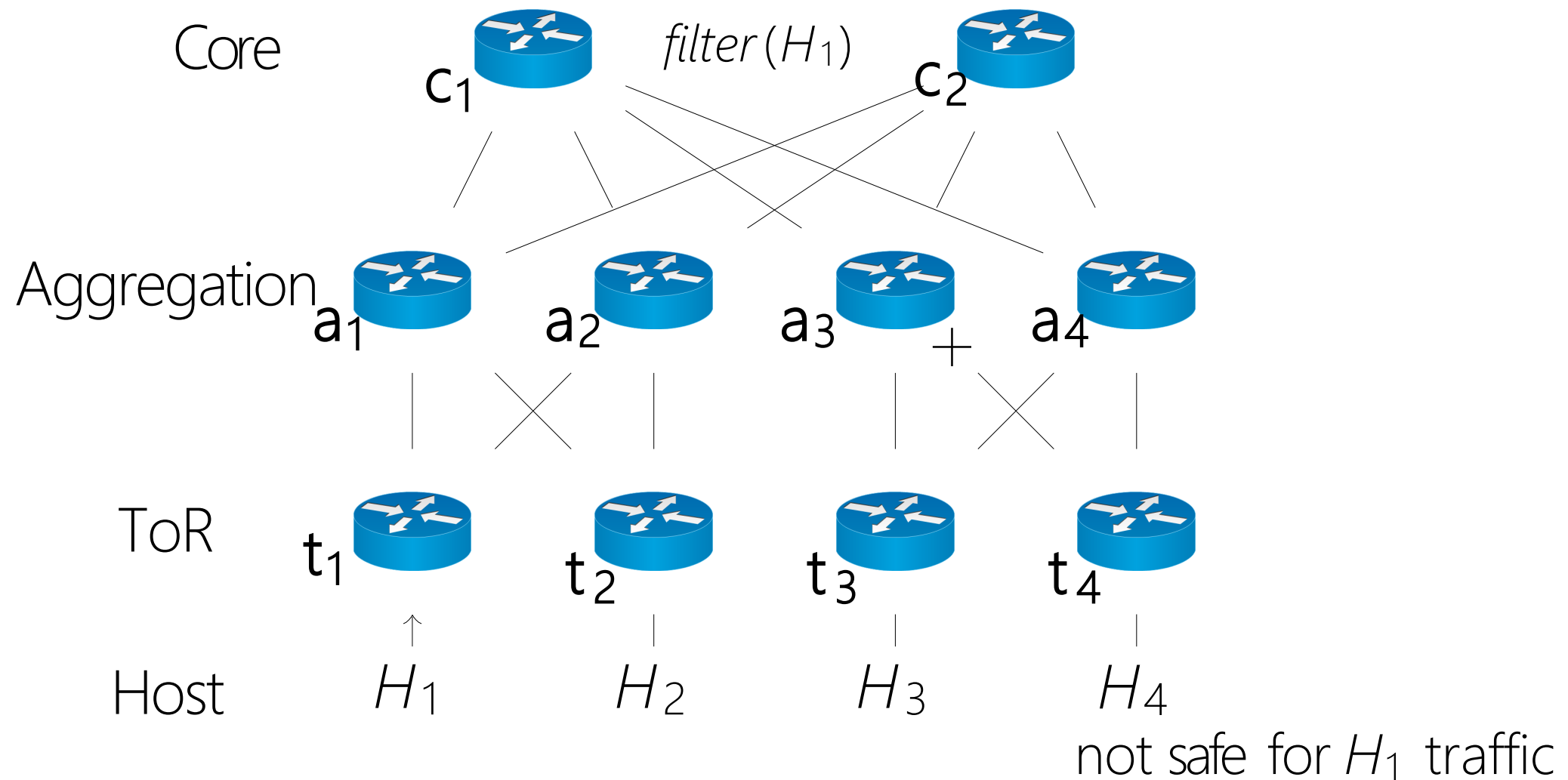
# Contributions

- 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



- 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

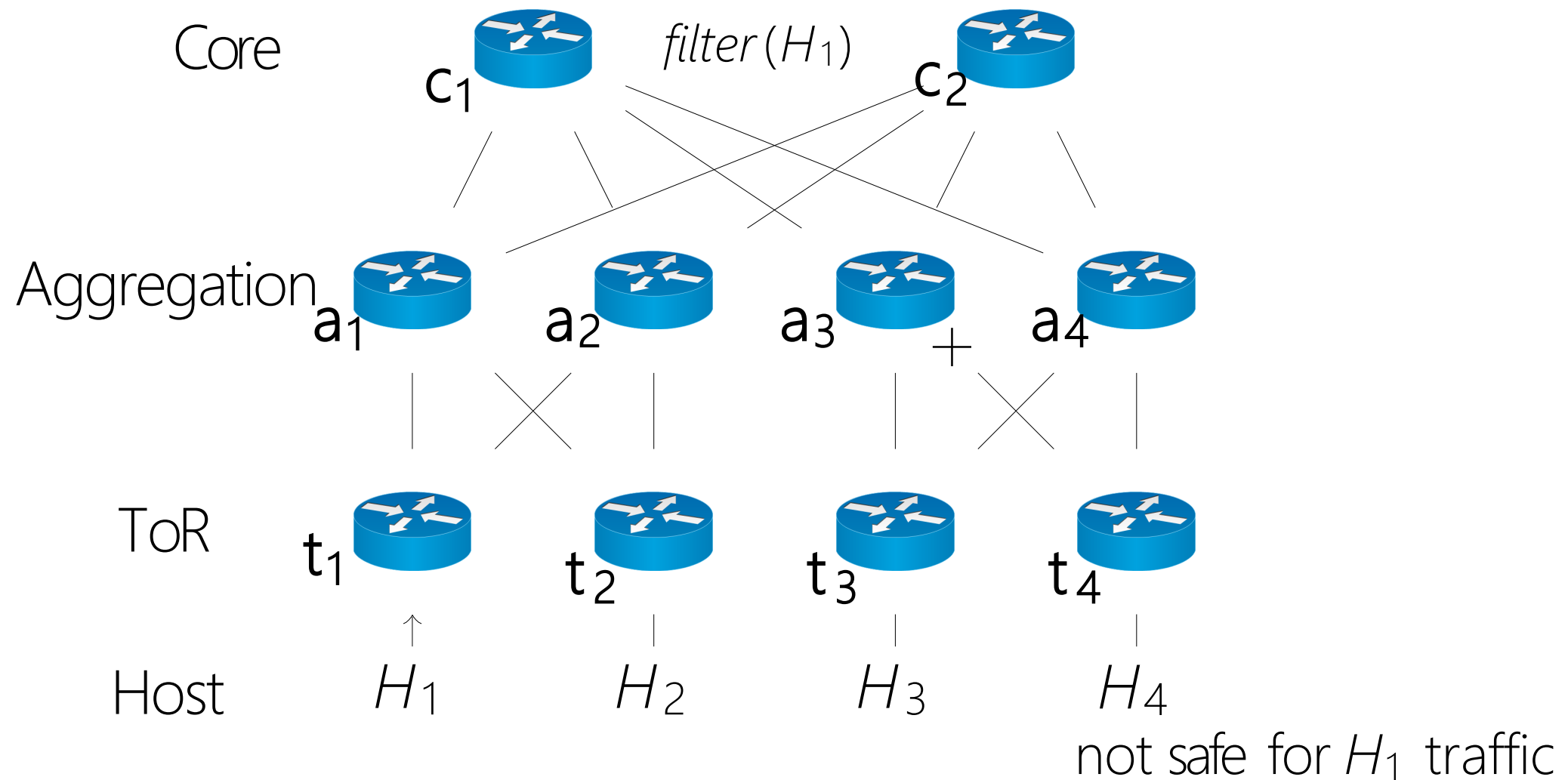


Ingress.  $H_1$  sends out the special traffic type 0

$$\begin{aligned}
 (typ = 0 \wedge dst \in \{2, 3, 4\}) &\rightarrow t_1(dst, typ) \\
 (typ > 0 \wedge typ < 8 \wedge dst \in \{1, 3, 4\}) &\rightarrow t_2(dst, typ) \\
 (typ > 0 \wedge typ < 8 \wedge dst \in \{1, 2, 4\}) &\rightarrow t_3(dst, typ) \\
 (typ > 0 \wedge typ < 8 \wedge dst \in \{1, 2, 3\}) &\rightarrow t_4(dst, typ)
 \end{aligned}$$



# Horn Clauses for Network



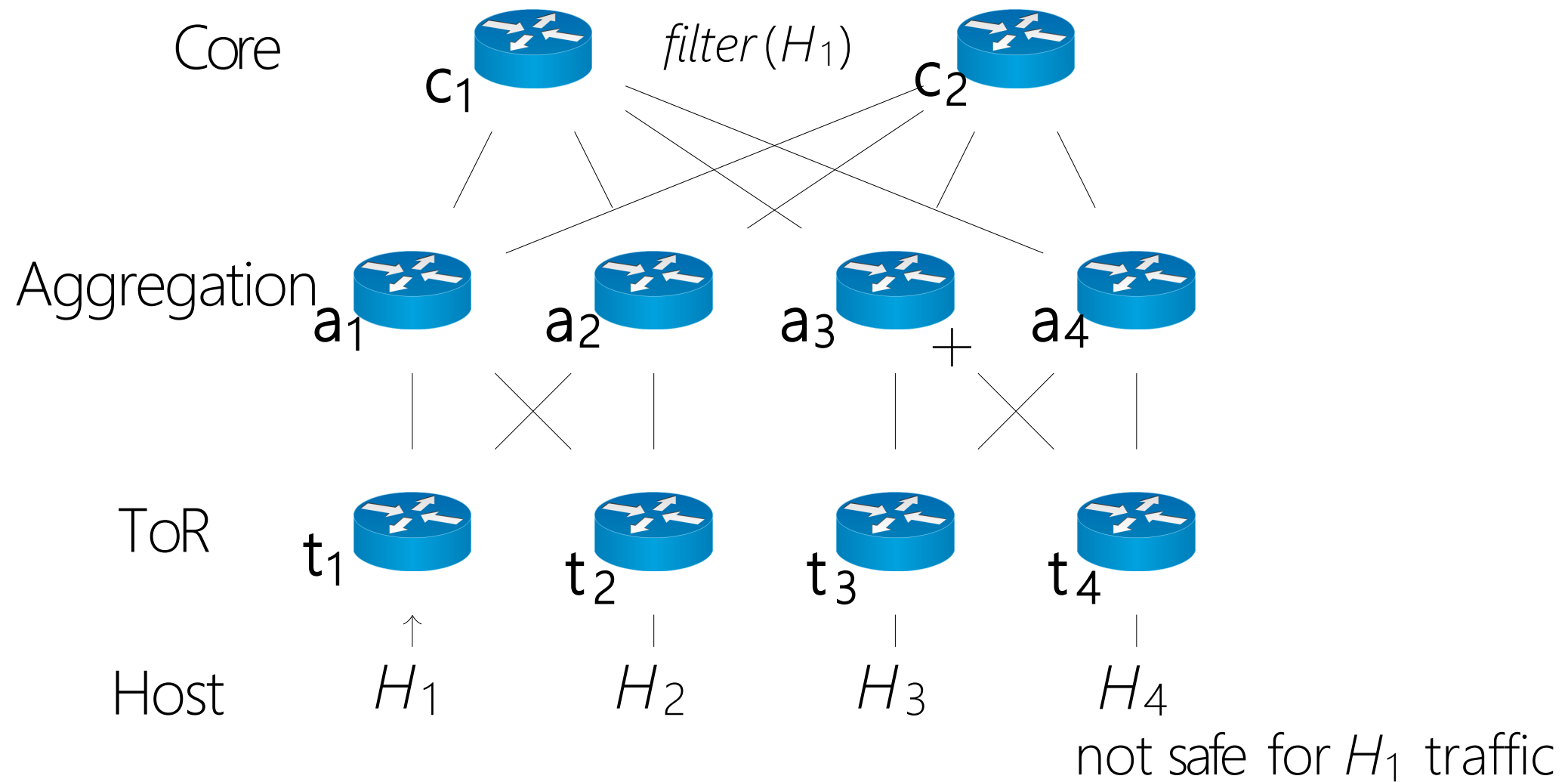
We use a special relation symbol **D** for dropping a packet

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

$$t_1(dst, typ) \wedge (dst \neq 1) \rightarrow a_2(dst, typ)$$

$$t_1(dst, typ) \wedge \neg ( (dst \geq 1) \wedge (dst \leq 4) \wedge (typ \geq 0) \wedge (typ \leq 7) ) \rightarrow D(dst, typ)$$

# Horn Clauses for Network



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

$$t_4(dst, typ) \wedge (typ = 0) \rightarrow false$$

$$D(dst, typ) \wedge (typ = 0) \rightarrow false$$

$$(typ = 0 \wedge dst \in \{2, 3, 4\}) \rightarrow \mathbf{t}_1(dst, typ)$$

...

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

...

$$\mathbf{a}_1(dst, typ) \wedge (dst \neq 1) \wedge (dst \neq 2) \rightarrow \mathbf{c}_2(dst, typ)$$

...

$$\mathbf{c}_2(dst, typ) \wedge (dst = 3 \vee dst = 4) \rightarrow \mathbf{a}_4(dst, typ)$$

...

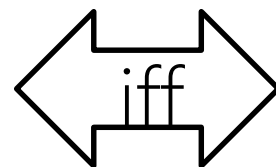
$$\mathbf{a}_4(dst, typ) \wedge (dst = 4) \rightarrow \mathbf{t}_4(dst, typ)$$

...

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

- 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/>

Specification

Program

Eldarica

Correct

Incorrect + counter-example

Time-out  
Loop forever



• Novel techniques for abstraction refinement:

- 1 Accelerated Interpolants
- 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\*

Hossein Hojjat<sup>1</sup>, Filip Konečný<sup>2,3</sup>, Florent Garnier<sup>4</sup>,  
Radu Iosif<sup>5</sup>, Viktor Kuncak<sup>6</sup>, and Philipp Rümmer<sup>7</sup>

<sup>1</sup> Swiss Federal Institute of Technology Lausanne (EPFL)  
<sup>2</sup> Virming, Grenoble, France  
<sup>3</sup> Uppsala University, Sweden  
<sup>4</sup> Swiss University of Technology, Czech Republic

**Abstract.** This paper presents a publicly available toolkit and a benchmark suite for rigorous verification of Integer Numerical Transition Systems (INTS), which can be viewed as control flow graphs whose edges are annotated by Presburger arithmetic formulas. We present FLINT and ELDERICA, two verification tools for INTS. The FLINT system is based on precise acceleration of the transition relations, while the ELDERICA system is based on predicate abstraction with interpolation-based counterexample-driven refinement. The ELDERICA verifier uses the PREDICAB theorem prover as a sound and complete interpolating prover for Presburger arithmetic. Both systems can solve several examples for which previous approaches failed, and present a useful baseline for verifying integer programs. The information is a starting point for rigorous benchmarking, comparison, and standardized communication between tools.

#### 1 Introduction

Common representation formats, benchmarks, and tool competitions have helped research in a number of areas, including constraint solving, theorem proving, and compilation. To bring such benefits to the area of software verification, we are preparing a standard logical format for programs, in terms of hierarchical infinite-state transition systems. The advantage of using a formally defined common format is enabling methodologies of programming language semantics and helping to separate semantic modeling from designing verification algorithms. This paper focuses on systems whose transition relation is expressed in Presburger arithmetic. Integer Numerical Transition Systems (abbreviated INTS in this paper), also known as counter automata, counter systems, or counter machines, are an infinite-state extension of the model of finite-state hybrid transition systems, a model extensively used in the area of software verification [8]. The interest for INTS comes from the fact that they can encode various classes of systems with unbounded (or very large) data domains, such as hardware circuits, cache

\* Supported by the Rich Model Toolkit initiative, <http://richmodeltoolkit.org>, the Czech Science Foundation projects P103/10:100 and J02/08:1402, the Czech Ministry of Education ECOST OC1009 and MSM 0021620820, the EU COST/ITN/Programme Centre of Excellence projects CZ-1.05/1.1.0002.0016, the BVT project ITT-12-1 and the Microsoft Innovation Center for Embedded Software.

© Springer-Verlag Berlin Heidelberg 2012

#### Accelerating Interpolants\*

Hossein Hojjat<sup>1</sup>, Radu Iosif<sup>2</sup>,  
Filip Konečný<sup>3,4</sup>, Viktor Kuncak<sup>5</sup>, and Philipp Rümmer<sup>7</sup>

<sup>1</sup> Swiss Federal Institute of Technology Lausanne (EPFL)  
<sup>2</sup> Virming, Grenoble, France  
<sup>3</sup> Uppsala University, Sweden  
<sup>4</sup> Swiss University of Technology, Czech Republic

**Abstract.** We present Counterexample-Guided Accelerated Abstraction Refinement (CEGAR-AR), a new algorithm for verifying infinite-state transition systems. CEGAR-AR combines interpolation-based predicate discovery in counterexample-guided predicate abstraction with an acceleration technique for computing the transition closure of loops. CEGAR-AR applies acceleration to dynamically discovered looping patterns in the unfolding of the transition system, and combines over-approximation with under-approximation. It constructs inductive invariants that rule out an infinite family of spurious counterexamples, alleviating the problem of divergence in predicate abstraction without losing its adaptive nature. We present theoretical and experimental justification for the effectiveness of CEGAR-AR, showing that inductive interpolants can be computed from classical Craig interpolants and transfer closures of loops. We present an implementation of CEGAR-AR that verifies integer transition systems. We show that the resulting implementation robustly handles a number of difficult transition systems that cannot be handled using interpolation-based predicate discovery or acceleration alone.

#### 1 Introduction

This paper contributes to the fundamental problem of precise reachability analysis for infinite-state systems. Predicate abstraction from using interpolation has emerged as an effective technique in this domain. The underlying idea is to verify a program by reasoning about its abstraction rather than to analyze it directly, and is defined with respect to a set of predicates [17]. The set of predicates is refined to achieve the precision needed to prove the absence of the presence of errors. A key difficulty in this approach is to automatically find predicates to make the abstraction sufficiently precise [2]. A breakthrough technique is to generate predicates based on Craig interpolants [13] derived from the proof of unsatisfiability of a spurious trace [19].

While empirically successful on a variety of domains, abstraction refinement using interpolants suffers from the unsatisfiability of interpolants computed by provers.

\* Supported by the Rich Model Toolkit initiative, <http://richmodeltoolkit.org>, the Czech Science Foundation projects P103/10:100 and J02/08:1402, the Czech Ministry of Education ECOST OC1009 and MSM 0021620820, the EU COST/ITN/Programme Centre of Excellence projects CZ-1.05/1.1.0002.0016, the BVT project ITT-12-1 and the Microsoft Innovation Center for Embedded Software.

© Springer-Verlag Berlin Heidelberg 2012

#### Classifying and Solving Horn Clauses for Verification

Philipp Rümmer<sup>1</sup>, Hossein Hojjat<sup>2</sup>, and Viktor Kuncak<sup>3</sup>

<sup>1</sup> Uppsala University, Sweden  
<sup>2</sup> Swiss Federal Institute of Technology Lausanne (EPFL)

**Abstract.** As a promising direction to overcome difficulties of verification, researchers have recently proposed the use of Horn constraints as intermediate representation. Horn constraints are related to Craig interpolants, which is one of the main techniques used to construct and refine abstractions in verification, and to synthesize inductive loop invariants. We give a classification of the different forms of Craig interpolation problems found in literature, and show that all of them correspond to natural fragments of recursive first Horn constraints. For a logic that has the binary interpolation property, all of these problems are solvable, but have different complexity. In addition to presenting the theoretical classification and solvability results, we present a publicly available collection of benchmarks to evaluate solvers for Horn constraints, categorized according to their complexity. The benchmarks are derived from real-world verification problems. The behavior with our tools as well as with 23 previous solvers indicates the importance of Horn clause solving as distinct from the general problem of solving quantified constraints by quantifier elimination.

#### 1 Introduction

Predicate abstraction [14] has emerged as a prominent and effective way for model checking software systems. A key ingredient in predicate abstraction is analyzing the spurious counterexamples to refine abstractions [4]. The refinement problem was a significant progress when Craig interpolants extracted from unsatisfiability proofs were used as relevant predicates [20]. While interpolation has enjoyed a significant progress for various logical constraints [7–9, 24], there have been substantial proposals for more general forms of interpolation [1, 19, 26].

As a promising direction to extend the reach of automated verification methods to programs with procedures, and concurrent programs, among others, recently the use of Horn constraints as intermediate representation has been proposed [15, 16, 28]. This paper examines the relationship between various forms of Craig interpolation and syntactically defined fragments of recursive first Horn clauses. We systematically examine binary interpolation, inductive interpolant sequences, tree interpolants, restricted DAG interpolants, and disjunctive interpolants, and show the recursive first Horn clauses problem to which they correspond. We present algorithms for solving each of these classes of problems by reduction to elementary interpolation problems. We also give a taxonomy of the various interpolation problems, and the corresponding system of Horn clauses, in terms of their computational complexity.

© Springer-Verlag Berlin Heidelberg 2012

#### Disjunctive Interpolants for Horn-Clause Verification

Philipp Rümmer<sup>1</sup>, Hossein Hojjat<sup>2</sup>, and Viktor Kuncak<sup>3</sup>

<sup>1</sup> Uppsala University, Sweden  
<sup>2</sup> Swiss Federal Institute of Technology Lausanne (EPFL)

**Abstract.** One of the main challenges in software verification is efficient and precise computational analysis of programs with procedures and loops. Interpolation methods remain one of the most promising techniques for such verification, and are closely related to solving first Horn clause constraints. We introduce a new notion of interpolants, disjunctive interpolants, which solve a more general class of problems in one step compared to previous notions of interpolants, such as tree interpolants or inductive sequences of interpolants. We present algorithms and complexity for construction of disjunctive interpolants, as well as their use within an abstraction-refinement loop. We have implemented three clause verification algorithms that use disjunctive interpolants and evaluate them on benchmarks reported in Horn clauses over the theory of integer linear arithmetic.

#### 1 Introduction

Software model checking has greatly benefited from the combination of a number of seminal ideas: automated abstraction through Bounded Model Checking [3], exploration of infinite-state abstractions, and counterexample-driven refinement [13]. Even though these techniques can be viewed independently, the effectiveness of verification has been consistently improving by providing more explicit and communication between these steps. Often, carefully chosen search aspects are being pushed into a learning-enabled constraint solver, resulting in better overall verification performance. An essential advance was to use interpolants derived from unsatisfiability proofs to refine the abstraction [13]. In recent years, we have seen significant progress in interpolating methods for different logical constraints [4, 5, 21], and a wealth of more general forms of interpolants [1, 12, 21, 24]. In this paper we identify a new notion, disjunctive interpolants, which are more general than tree interpolants and inductive sequences of interpolants. Like tree interpolation [12, 21], a disjunctive interpolation query is a tree-shaped constraint specifying the interpolants to be derived, however, in disjunctive interpolation, branching in the tree can represent both conjunctions and disjunctions. We present an algorithm for solving the interpolation problem, relating it to a subfamily of recursive first Horn clauses [18, 22, 23]. We then consider solving general recursive first Horn clauses and show that this problem is solvable whenever the logic admits interpolation. We establish tight complexity bounds for solving recursive first Horn clauses for propositional logic (PSPACE) and for integer linear arithmetic (co-NP/TIME). In contrast, the disjunctive interpolation problem remains in coNP for these logics. We also show how to use solvers for recursive first Horn clauses to verify recursive Horn clauses using counterexample-driven predicate abstraction. We present an algorithm and experimental results on publicly available benchmarks.

© Springer-Verlag Berlin Heidelberg 2012

Form Methods Syst Des (2015) 47:1–29  
DOI 10.1007/s00340-014-9509-7



#### On recursion-free Horn clauses and Craig interpolation

Philipp Rümmer · Hossein Hojjat · Viktor Kuncak

Published online: 13 December 2014  
© Springer Science+Business Media New York 2014

**Abstract.** One of the main challenges in software verification is efficient and precise analysis of programs with procedures and loops. Interpolation methods remain among the most promising techniques for such verification. To accommodate the demands of various programming language features, over the past years several extended forms of interpolation have been introduced. We give a precise, ontology of such extended interpolation methods, and investigate the relationship between interpolation and fragments of constrained recursive first Horn clauses. We also introduce a new notion of interpolants, disjunctive interpolants, which solve a more general class of problems in one step compared to previous notions of interpolants, such as tree interpolants or inductive sequences of interpolants. We present algorithms and complexity for construction of interpolants, as well as the corresponding decision problems for recursive first Horn clauses. Finally, we give an extensive empirical evaluation using a solver for recursive first Horn problems, in particular comparing the performance of tree interpolation and disjunctive interpolation for constraints modelling software verification tasks.

**Keywords.** Horn clause verification · Interpolation · Software verification · SMT solver · Software model checking · Predicate abstraction

P. Rümmer  
Department of Information Technology, Uppsala University, Box 357, 751 08 Uppsala, Sweden  
e-mail: philipp.ruemmer@it.uu.se

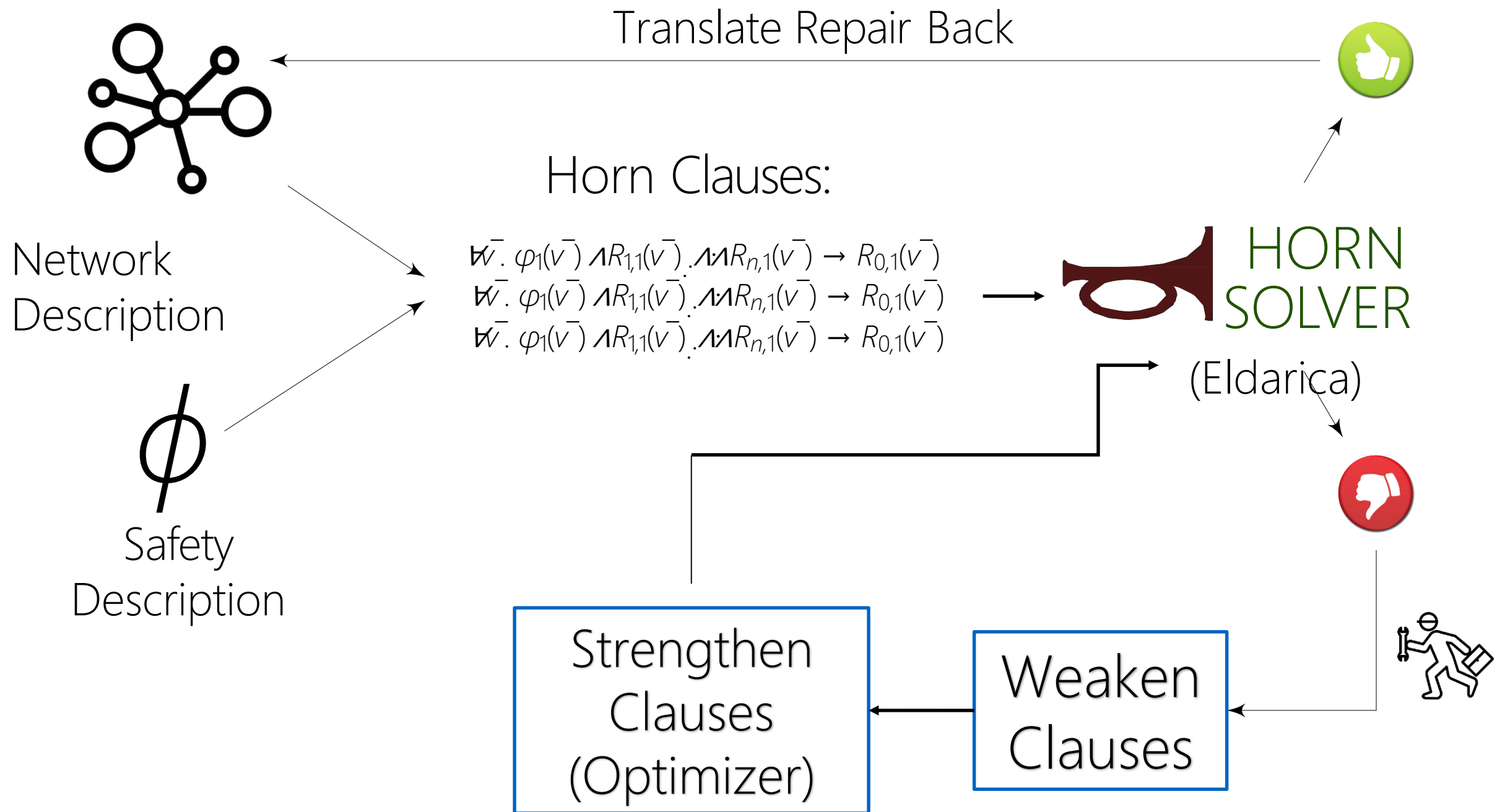
H. Hojjat  
Department of Computer Science, Cornell University, 402 Gates Hall, Ithaca, NY 14853, USA  
e-mail: hojjat@cs.cornell.edu

V. Kuncak  
EPFL, CH-1500, Lausanne, Switzerland  
e-mail: viktor.kuncak@epfl.ch

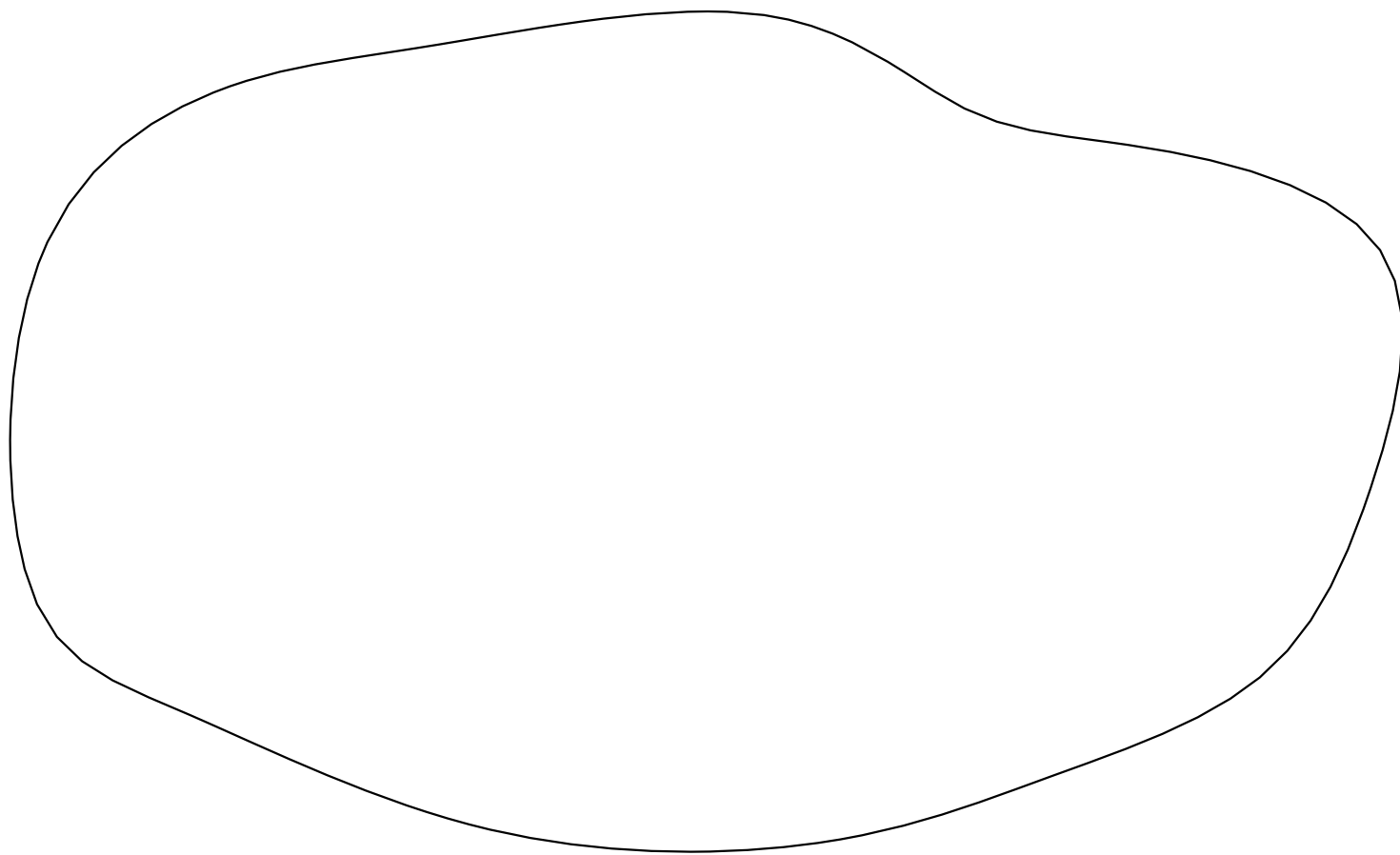


Hossein Hojjat, Philipp Rümmer  
(Filip Konečný, Radu Iosif, Florent Garnier, Pavle Subotic and Viktor Kuncak)

# Repair Framework

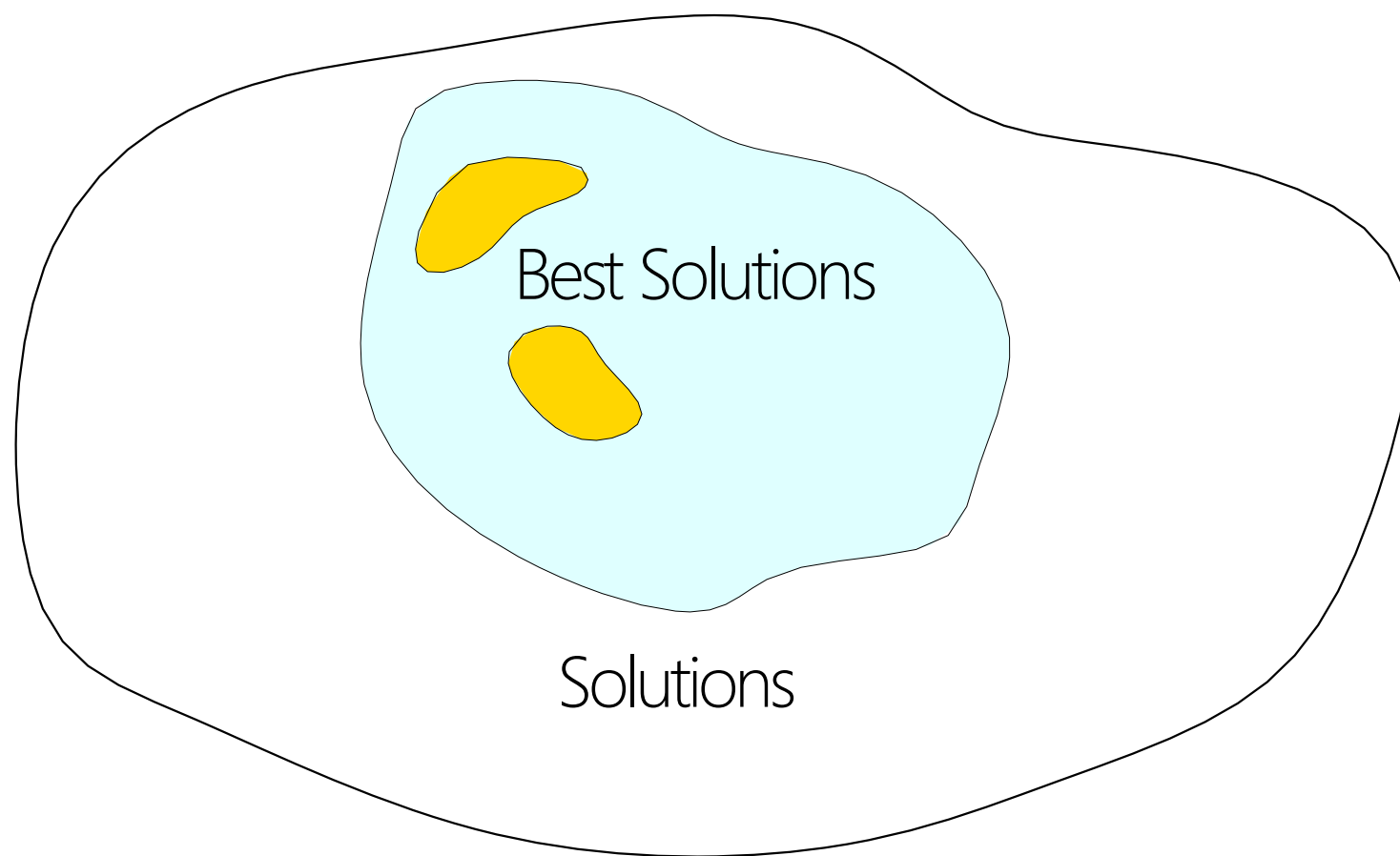


**Goal:** find solutions for set of Horn clauses subject to objective function



Space of all interpretations of relation symbols

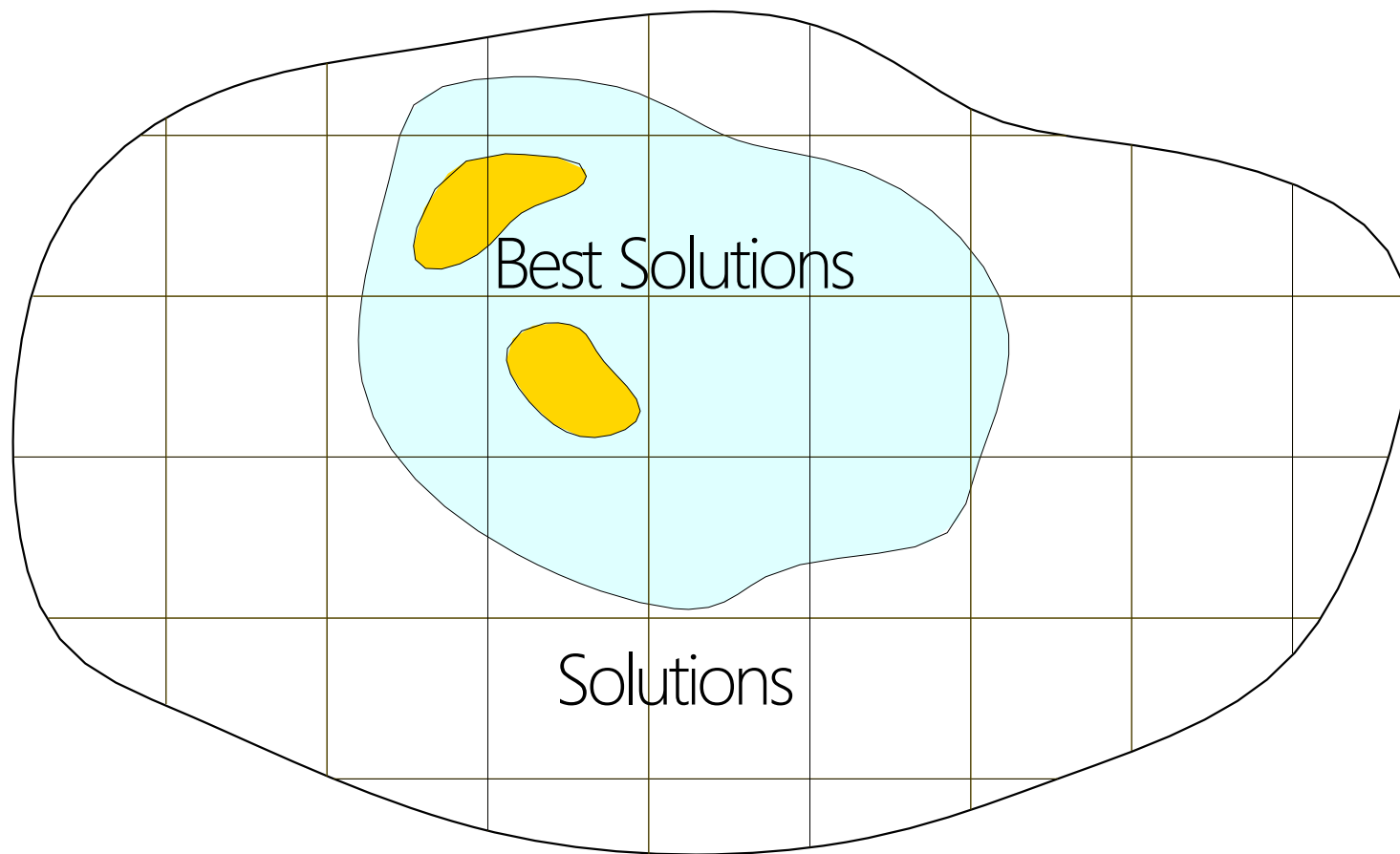
**Goal:** find solutions for set of Horn clauses subject to objective function



Space of all interpretations of relation symbols

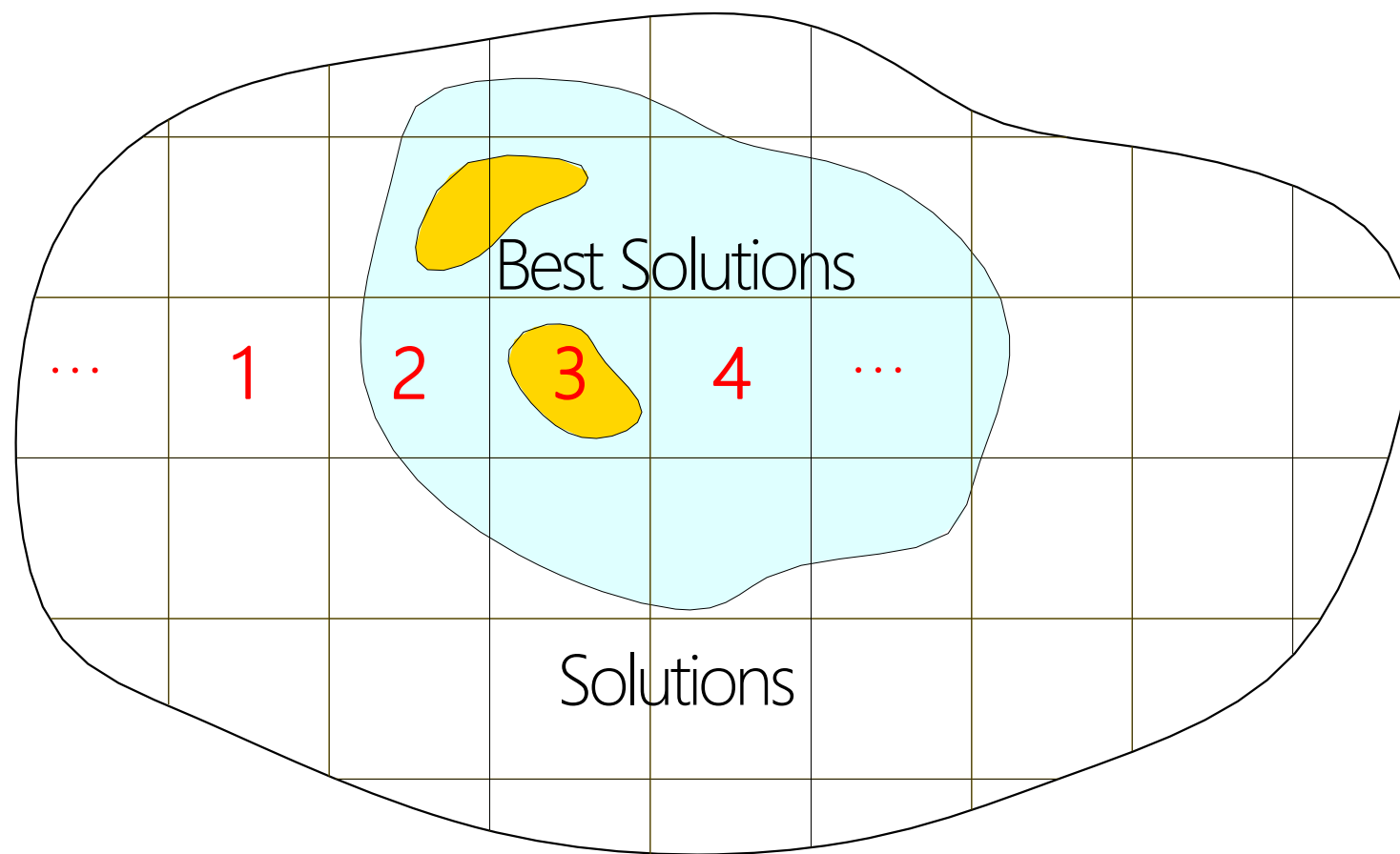


**Goal:** find solutions for set of Horn clauses subject to objective function

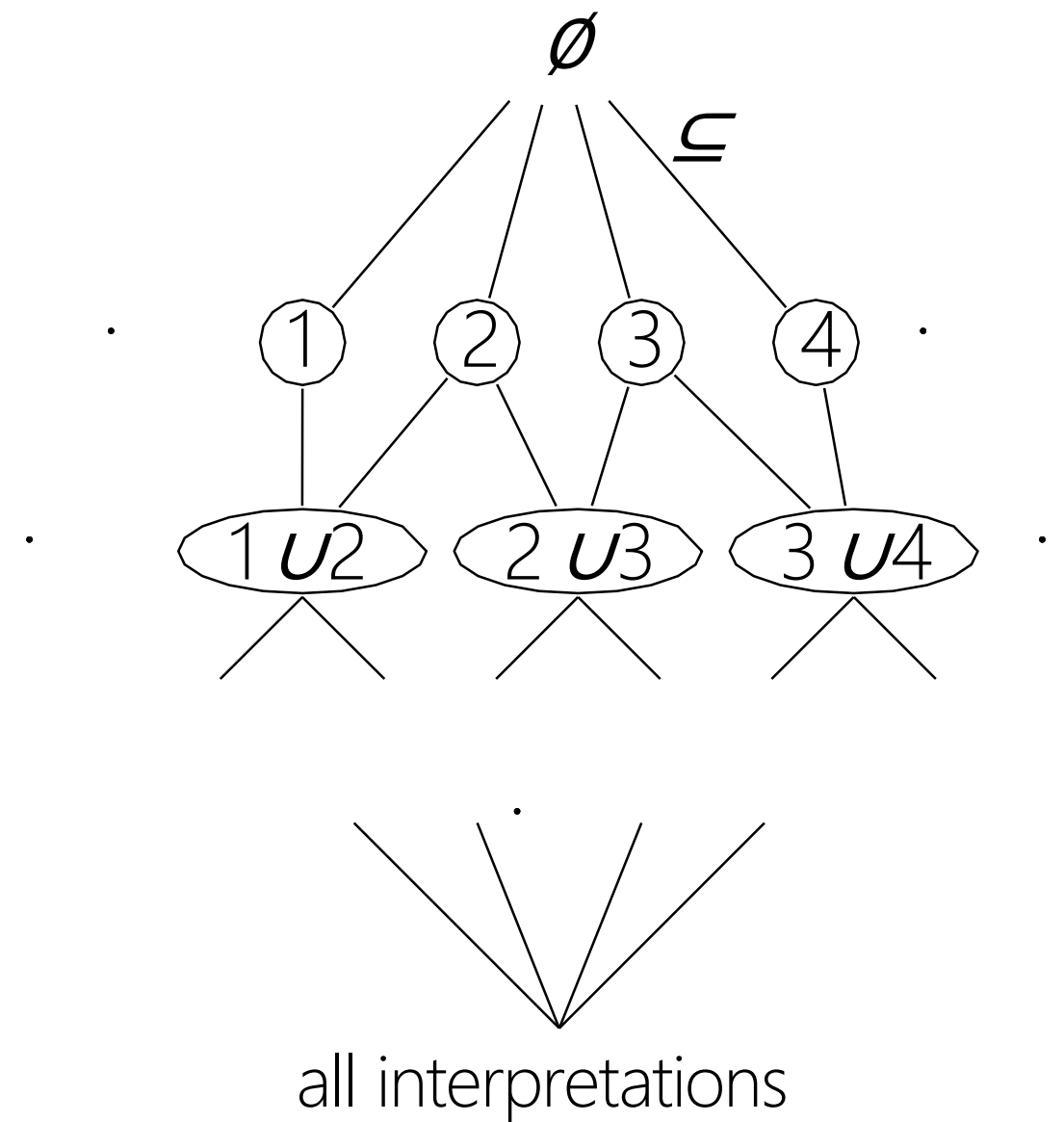


Space of all interpretations of relation symbols

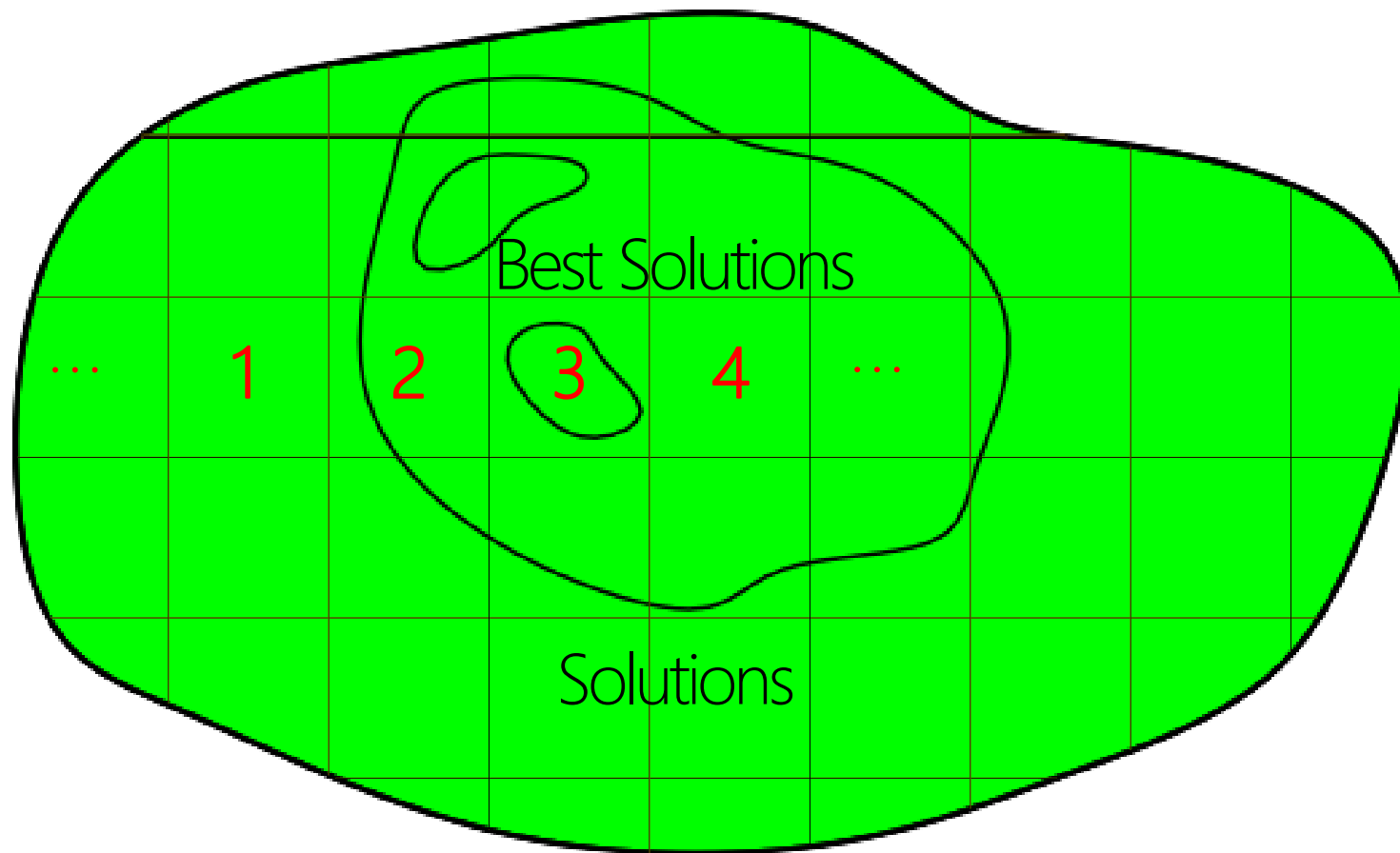
**Goal:** find solutions for set of Horn clauses subject to objective function



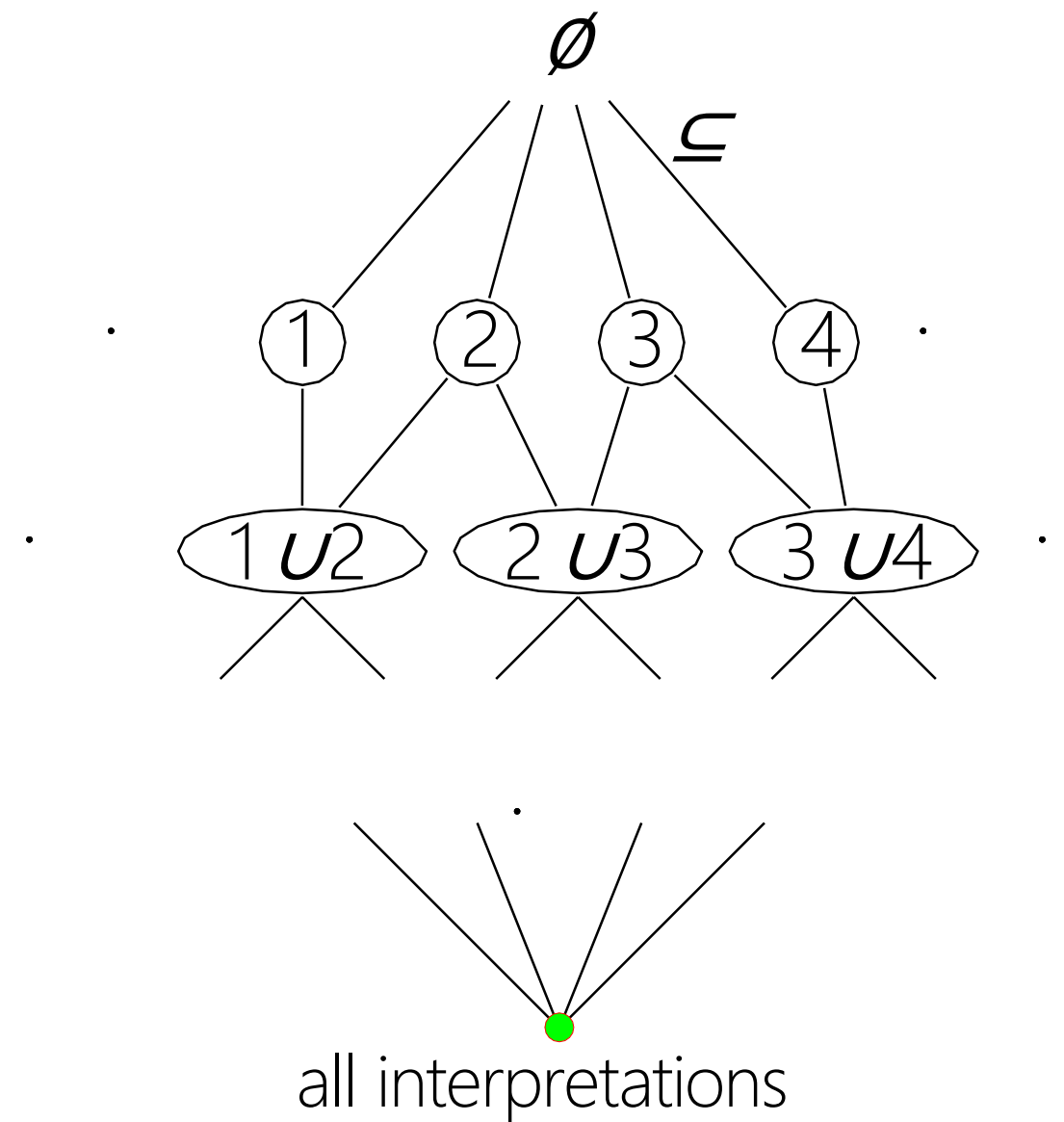
Space of all interpretations of relation symbols



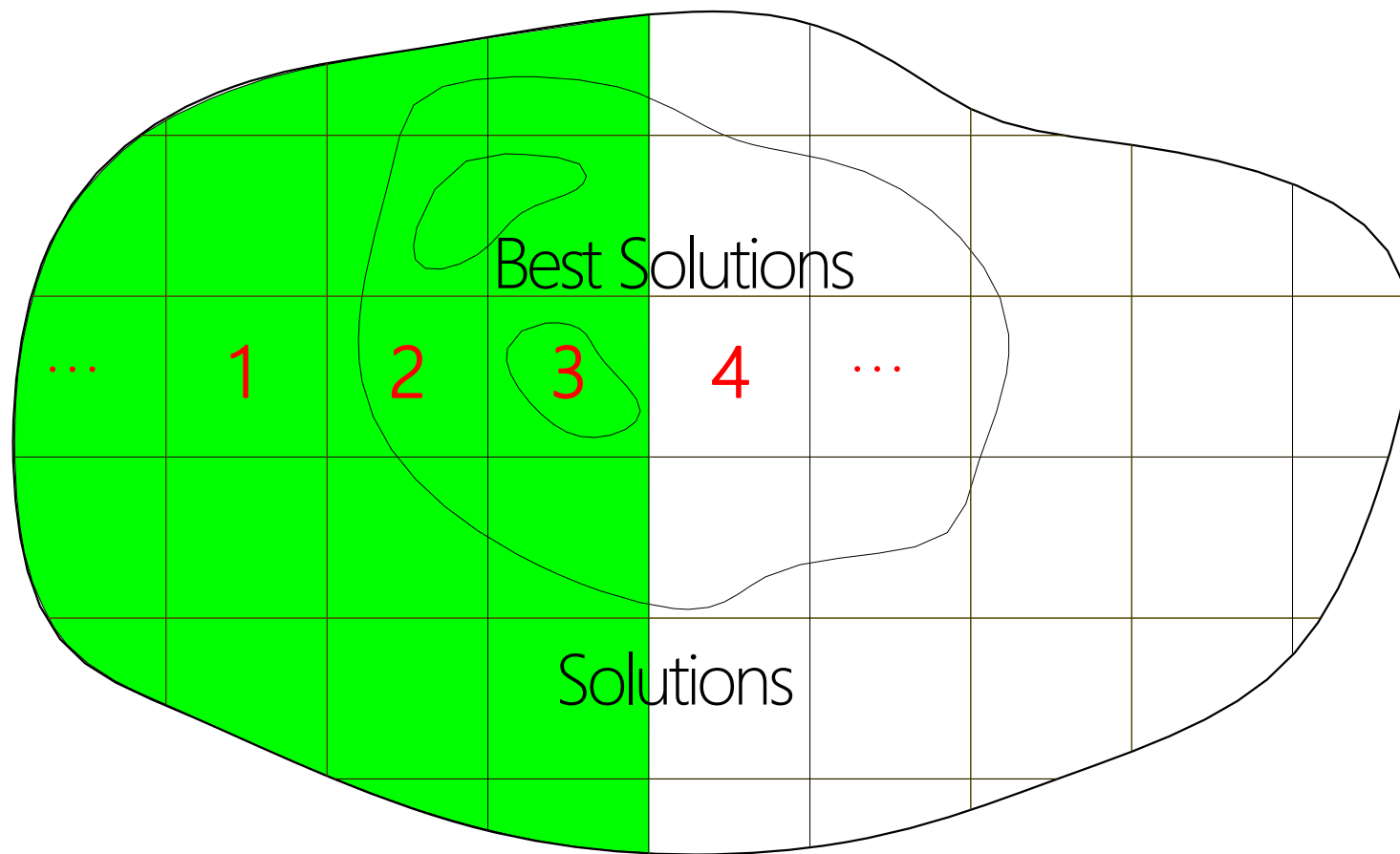
**Goal:** find solutions for set of Horn clauses subject to objective function



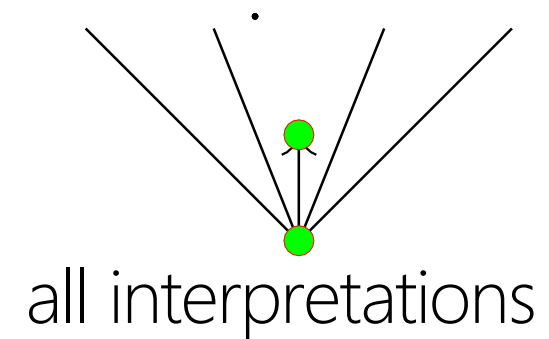
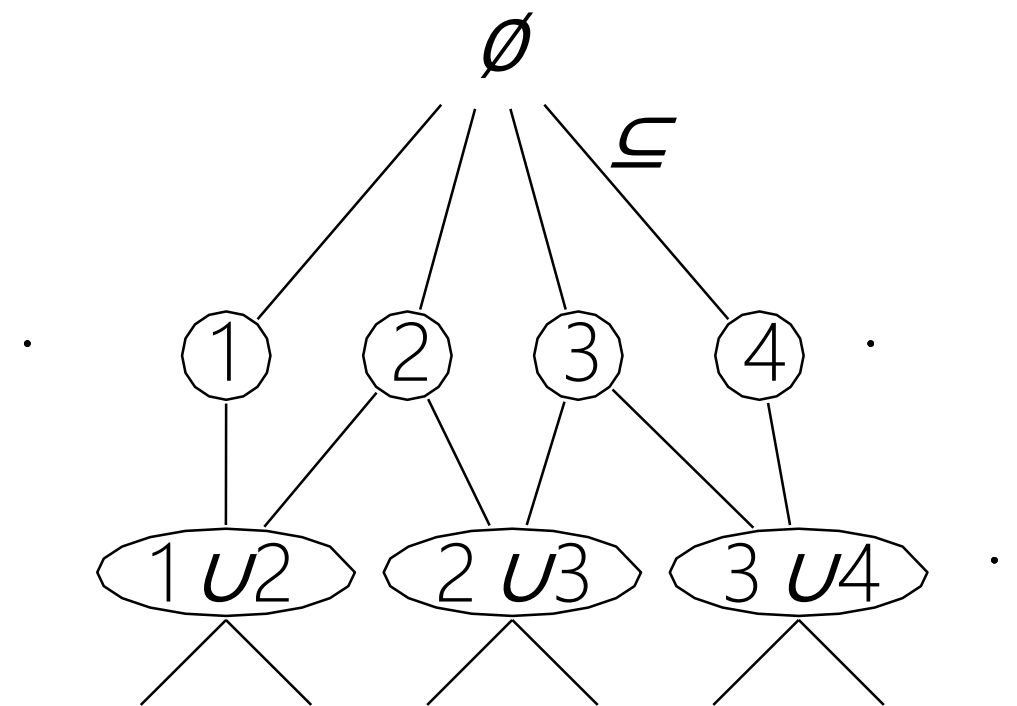
Space of all interpretations of relation symbols



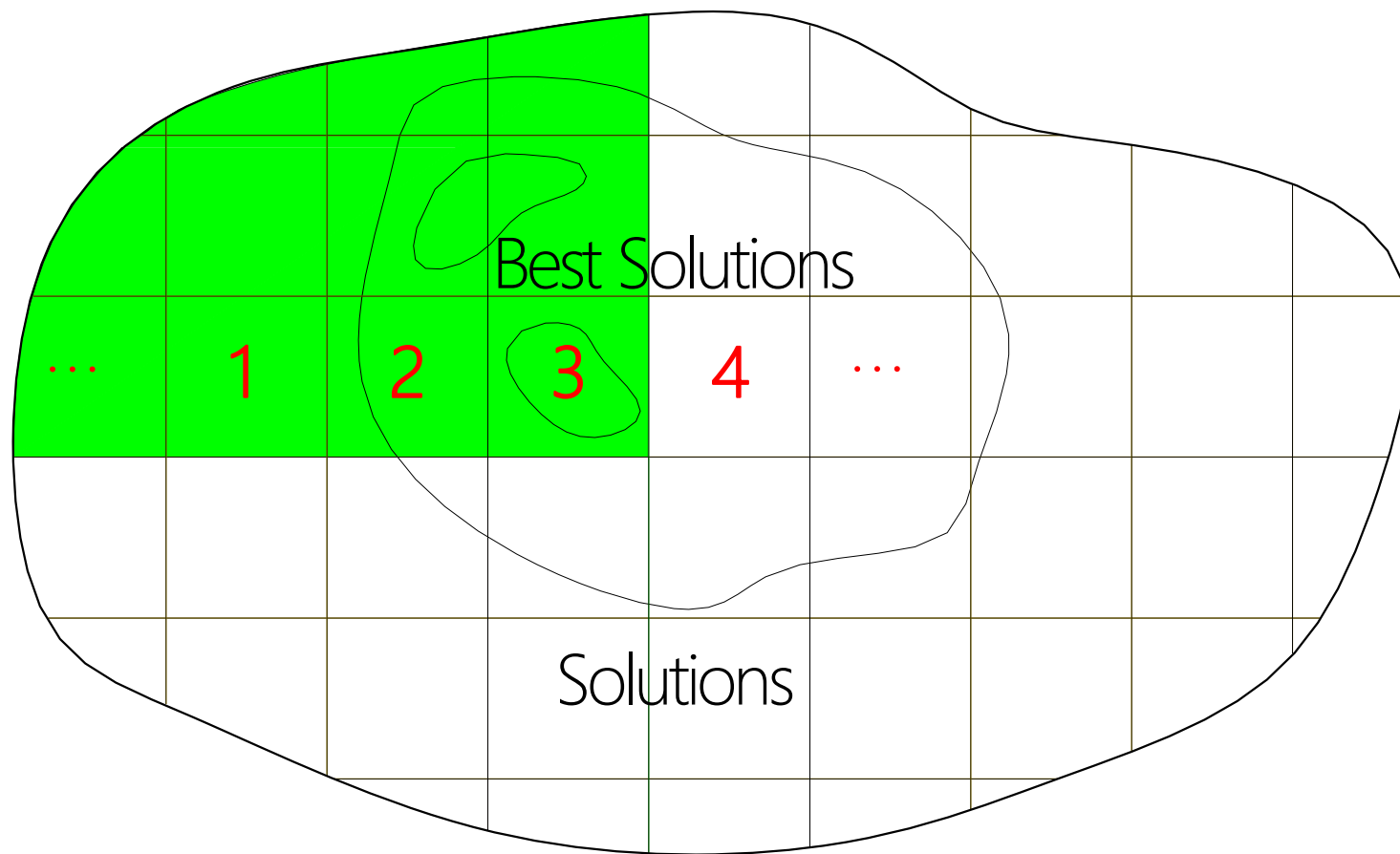
**Goal:** find solutions for set of Horn clauses subject to objective function



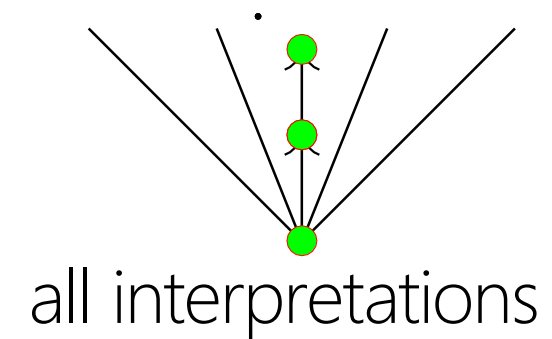
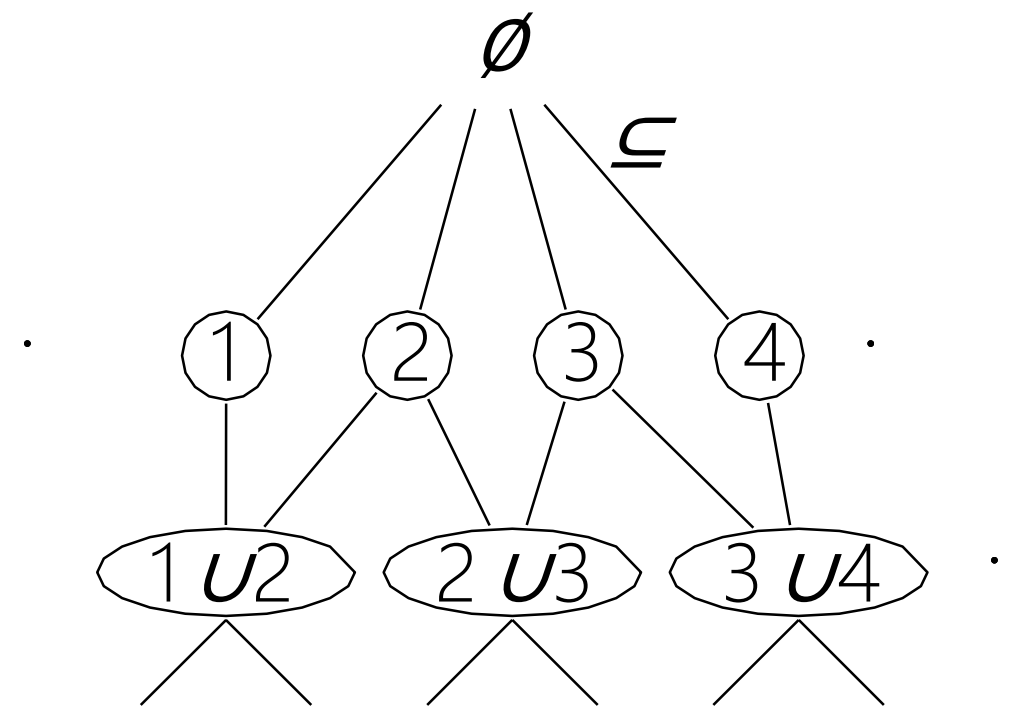
Space of all interpretations of relation symbols



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



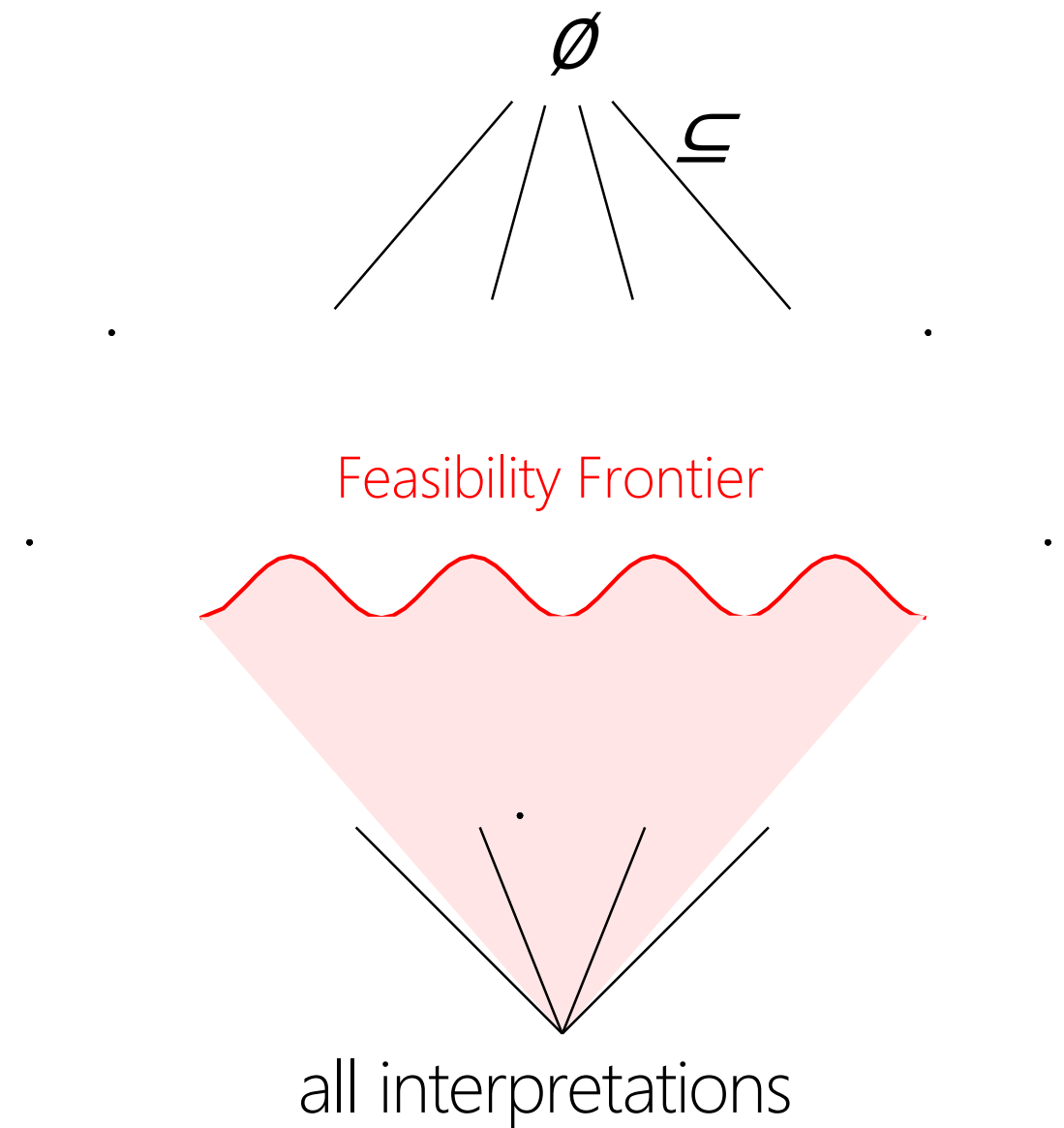
Space of all interpretations of relation symbols



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

Objective function:

Rank nodes of lattice monotonically



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

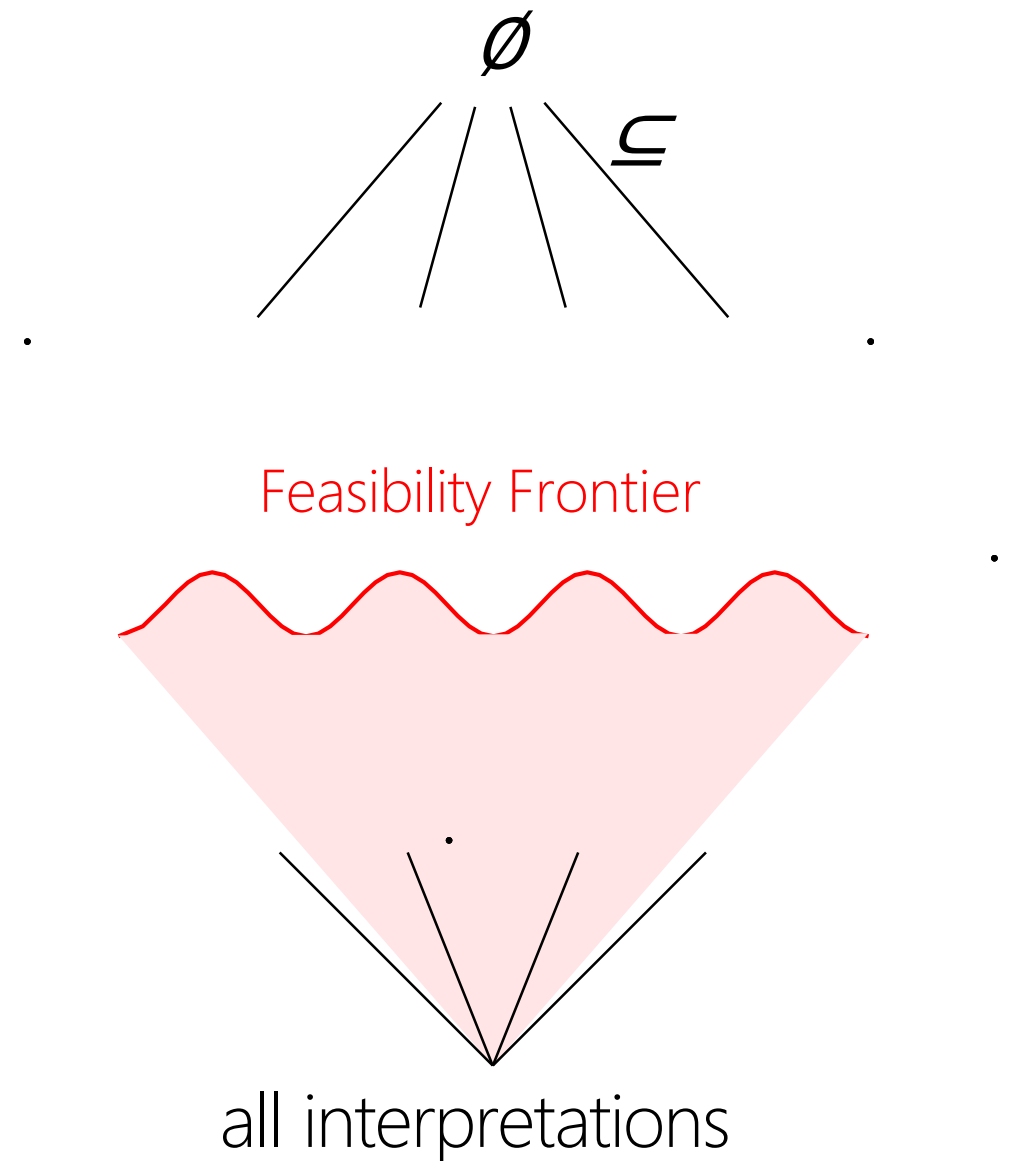
Objective function:

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:

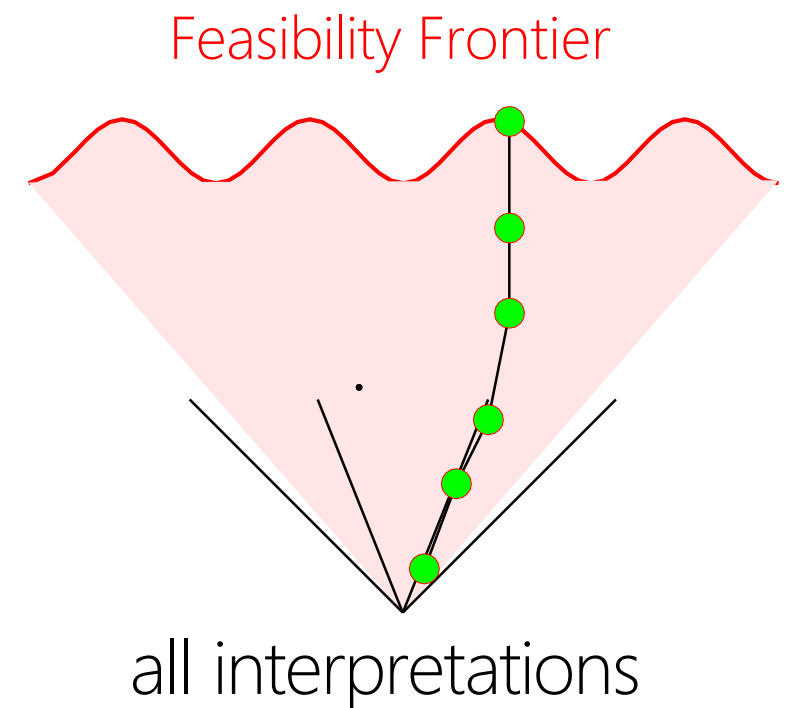
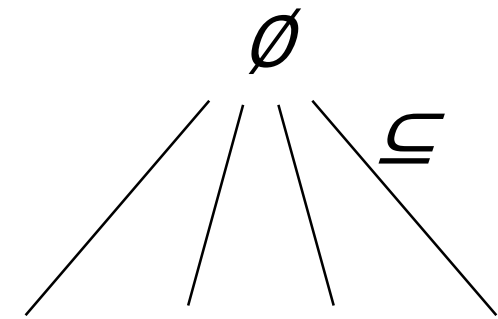
Rank nodes of lattice monotonically

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:

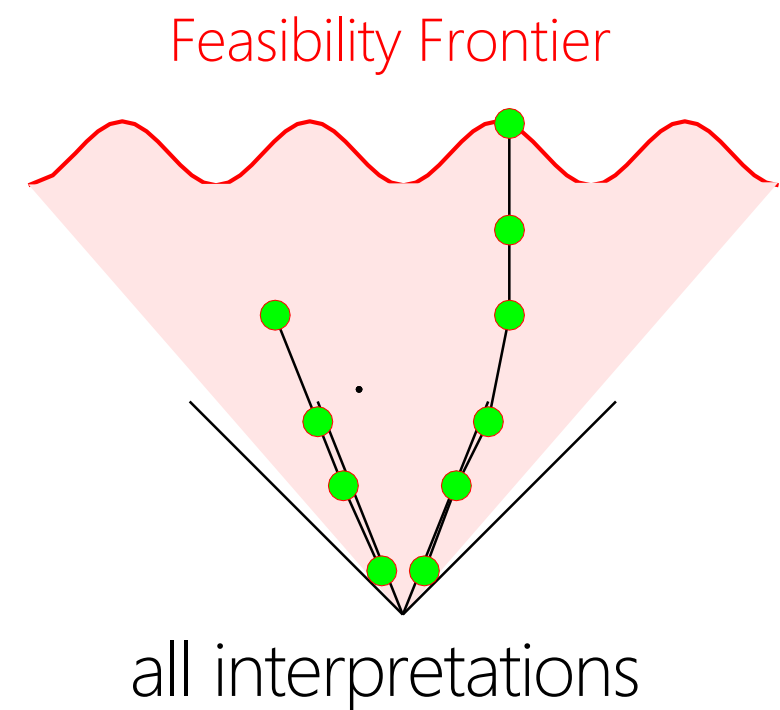
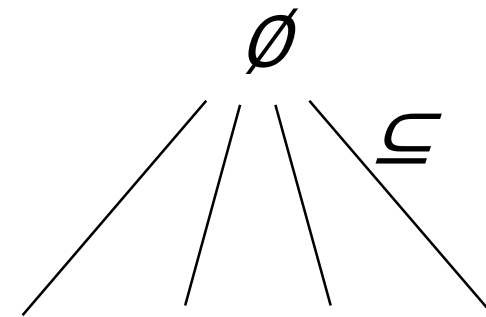
Rank nodes of lattice monotonically

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:

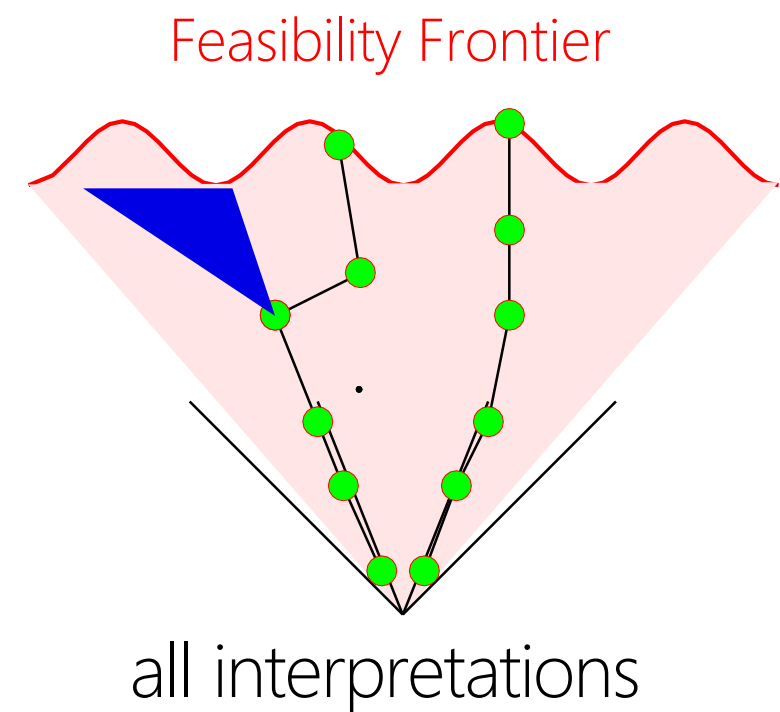
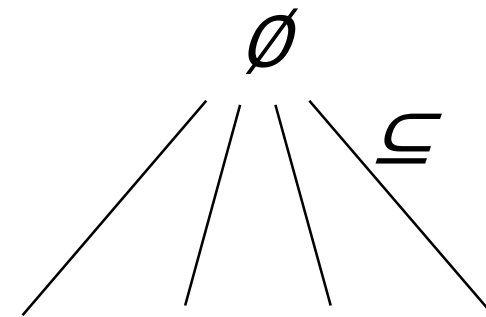
Rank nodes of lattice monotonically

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



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

Objective function:

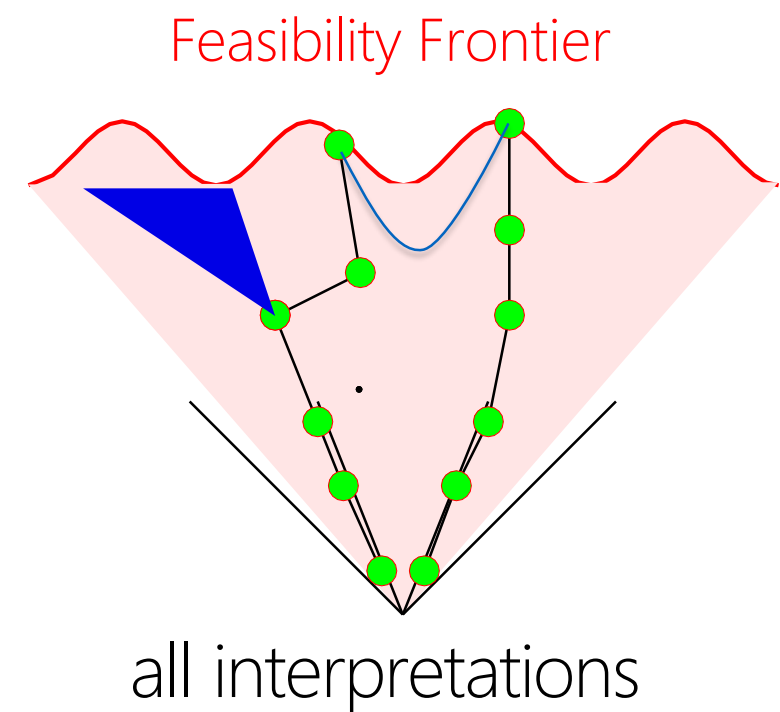
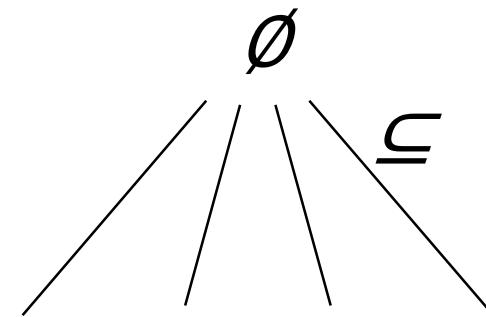
Rank nodes of lattice monotonically

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



- 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 of switchess

# Implementation and Experiments

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

# References

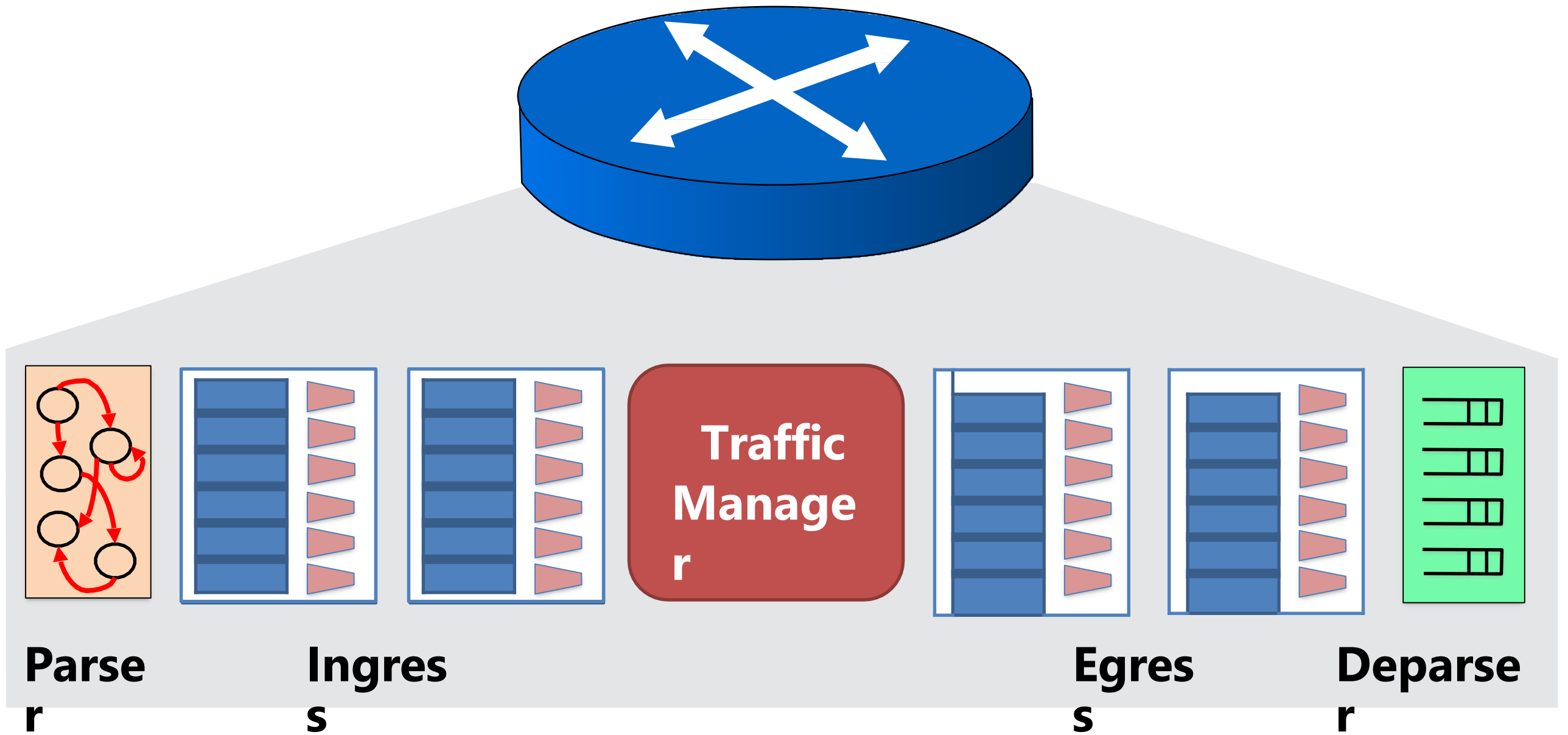
- Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. **Abstractions for Network Update**. (SIGCOMM 2012)
- Jedidiah McClurg, Hossein Hojjat, Pavol Cerny, and Nate Foster. **Efficient Synthesis of Network Updates**. (PLDI 2015)
- Jedidiah McClurg, Hossein Hojjat, Nate Foster, 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)

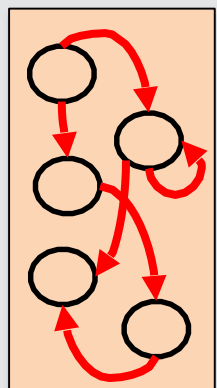
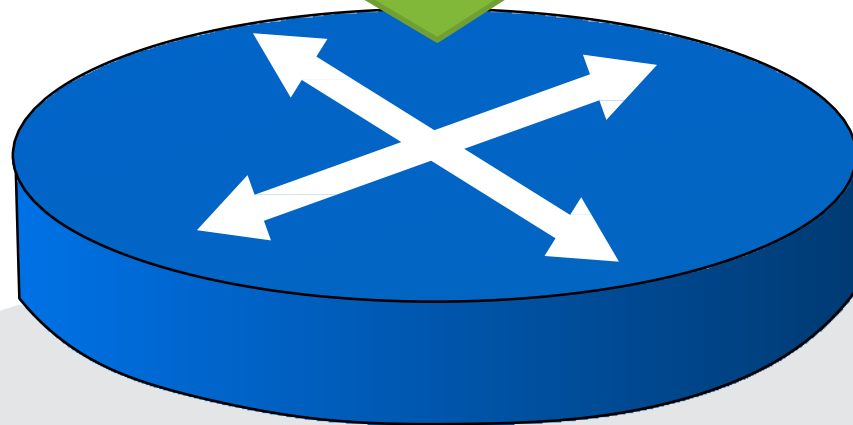
# Related Work

- 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.

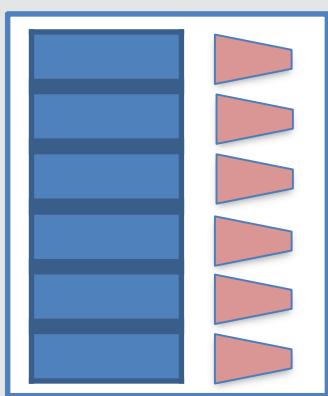


# **What is next?**

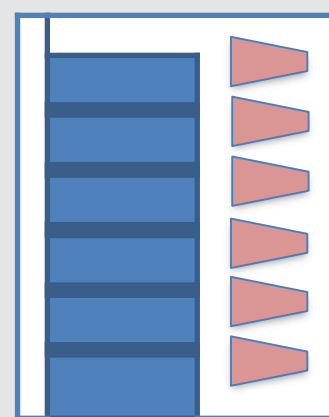
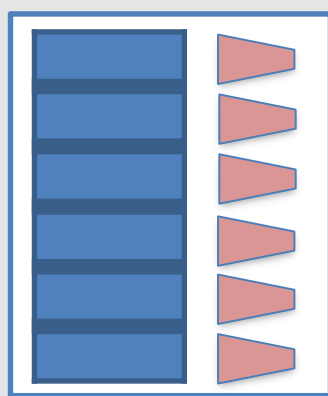




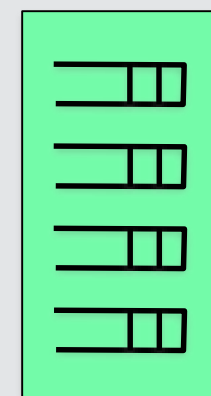
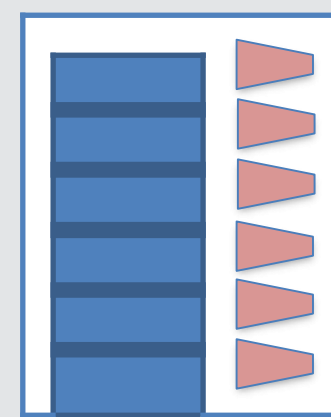
**Parse  
r**



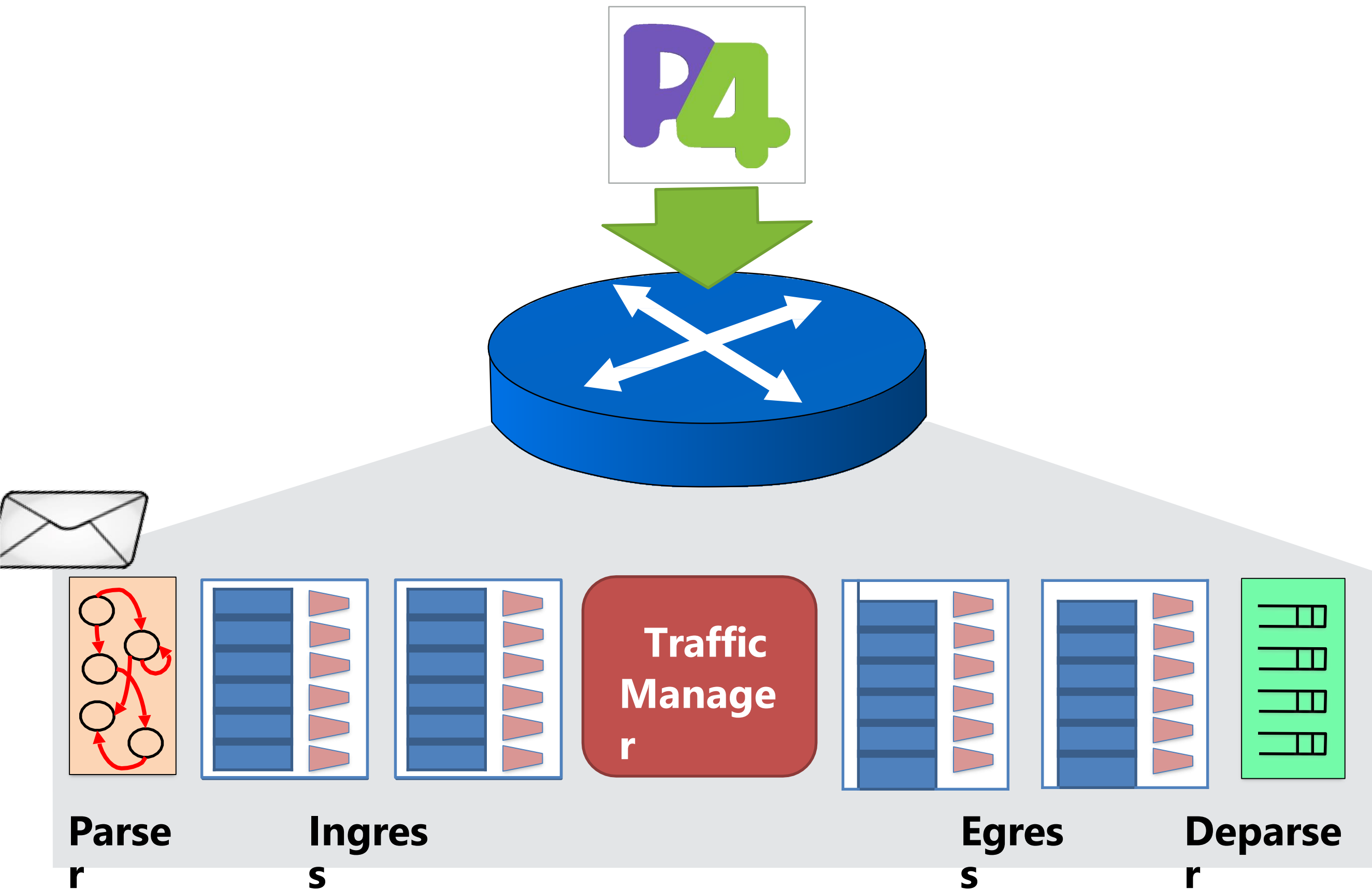
**Ingres  
s**

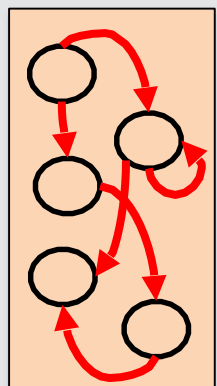
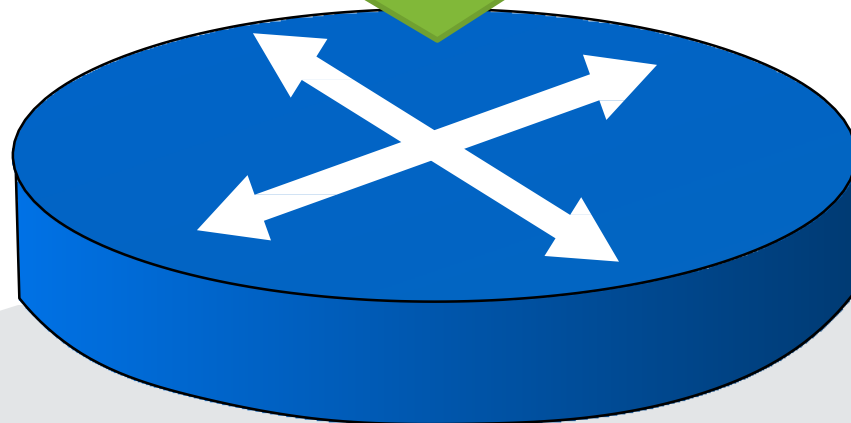


**Egres  
s**

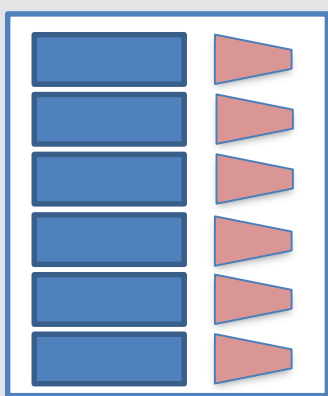


**Deparse  
r**

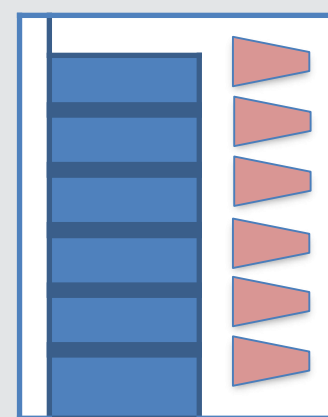
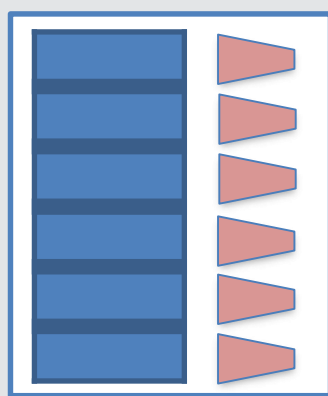




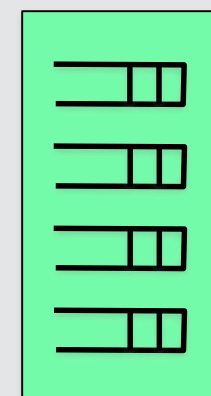
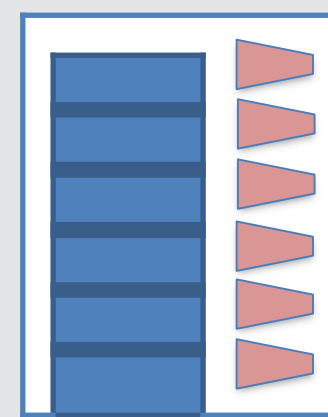
**Parse  
r**



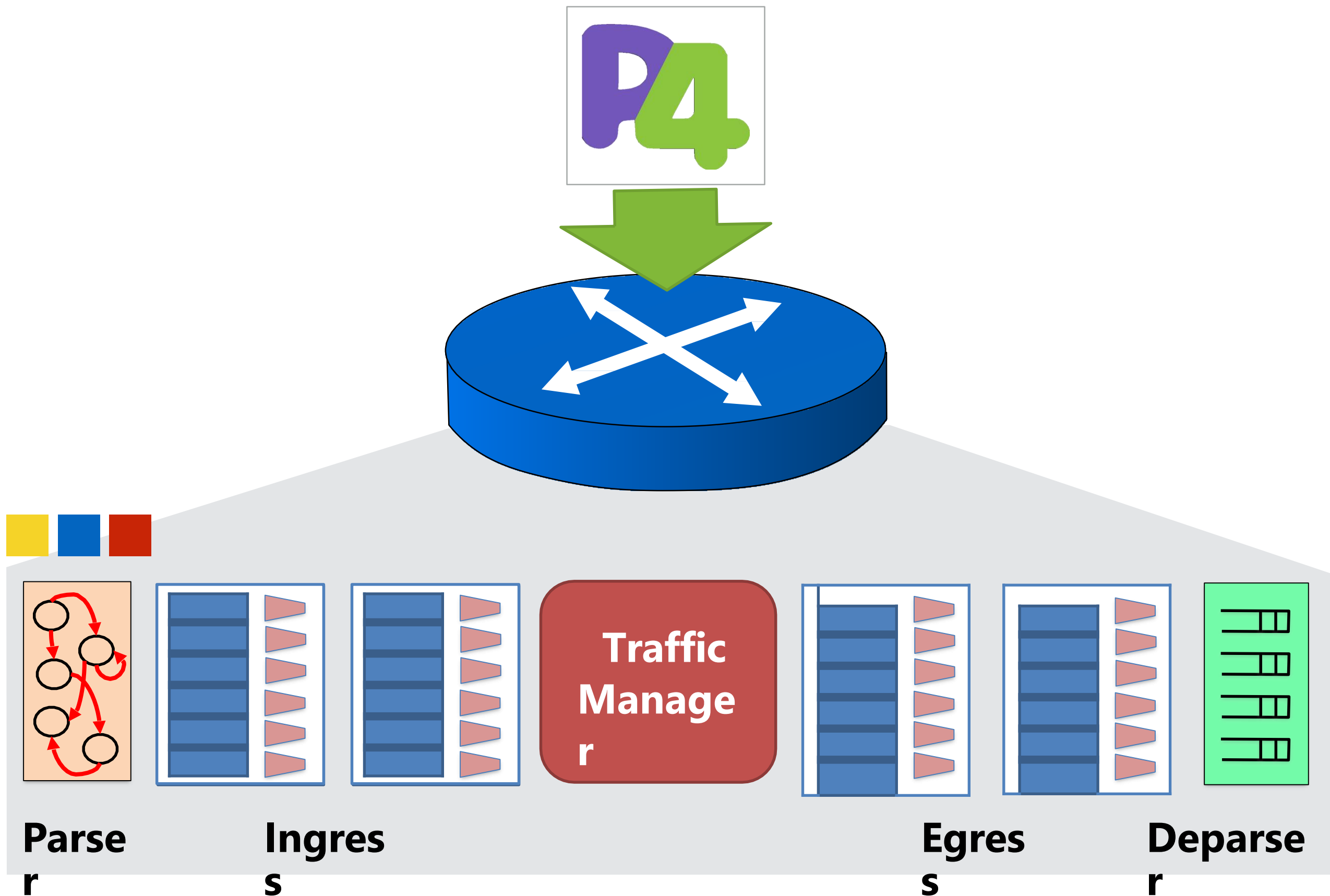
**Ingres  
s**

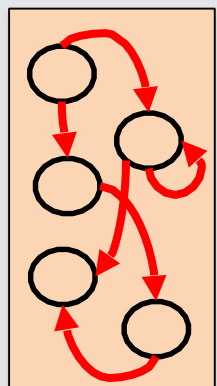
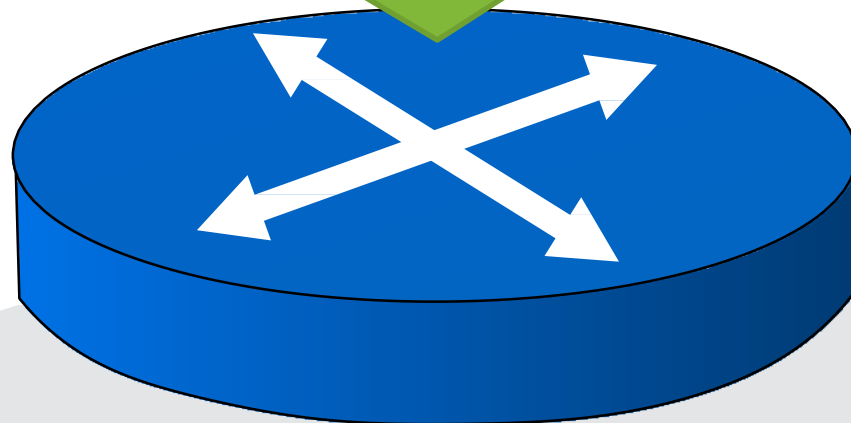


**Egres  
s**

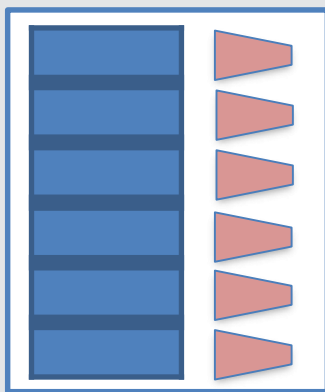


**Deparse  
r**

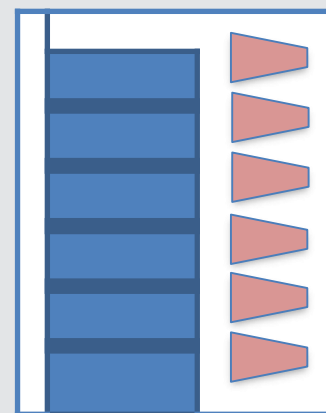
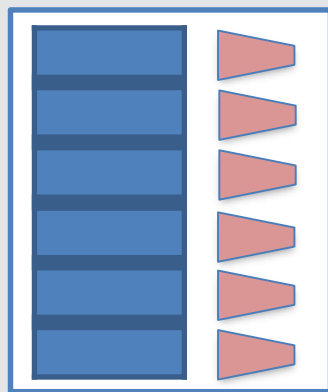




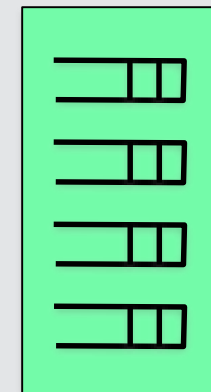
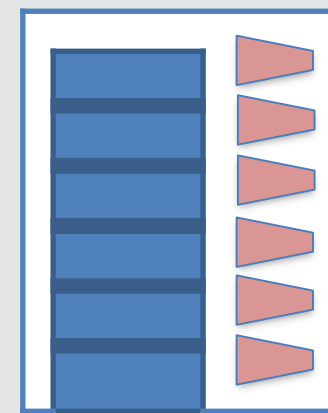
**Parse  
r**



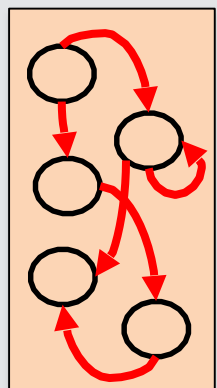
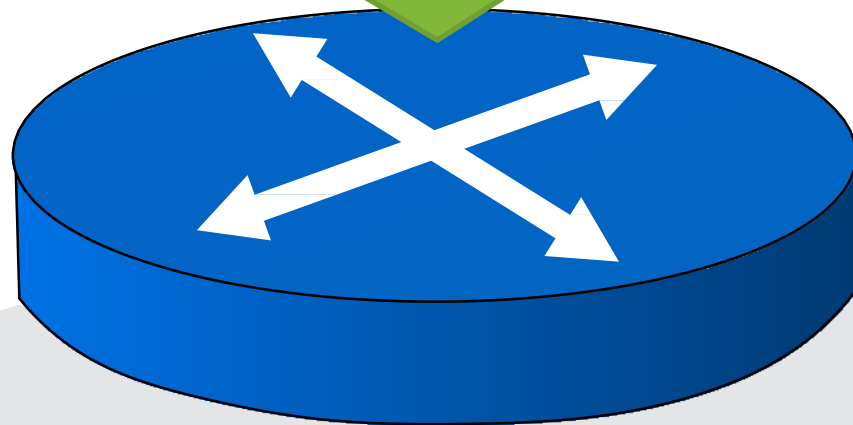
**Ingres  
s**



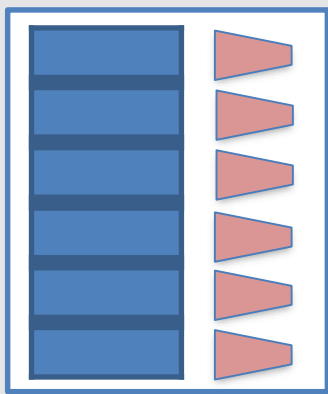
**Egres  
s**



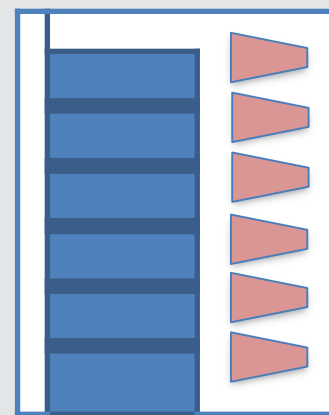
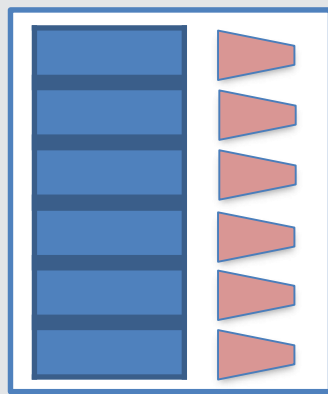
**Deparse  
r**



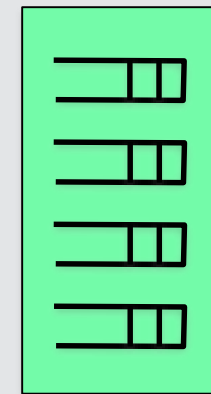
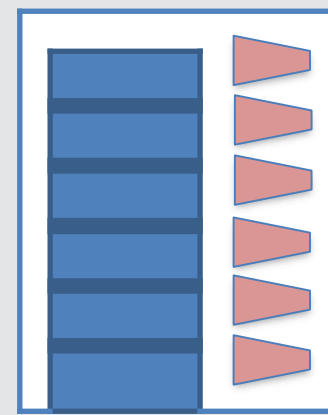
**Parse  
r**



**Ingres  
s**

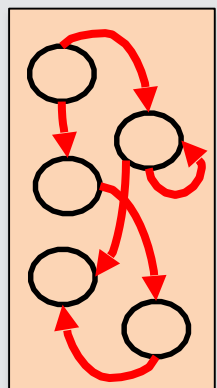
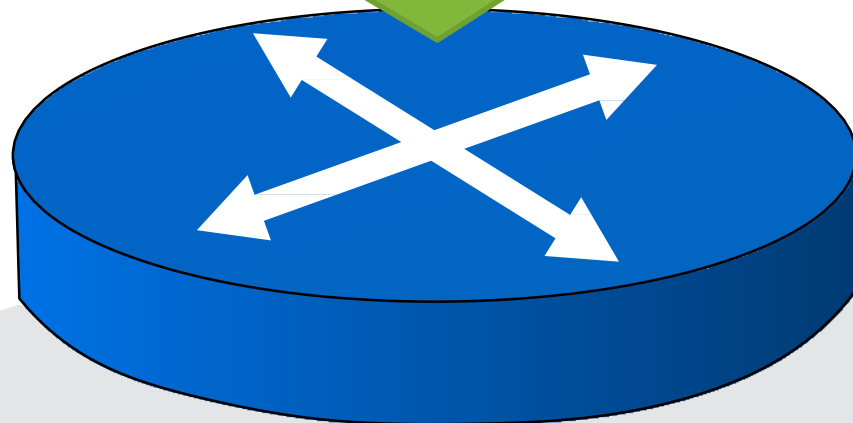


**Egres  
s**

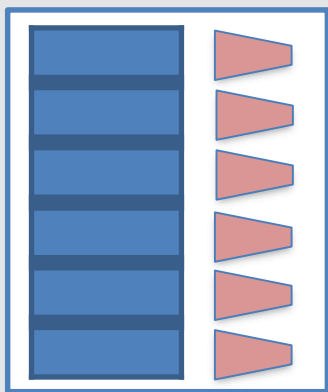


**Deparse  
r**

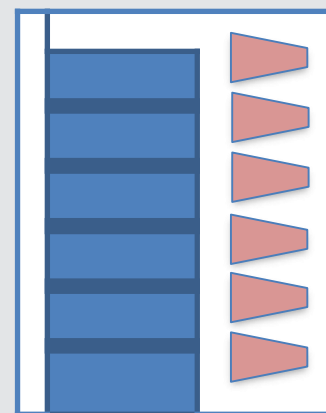
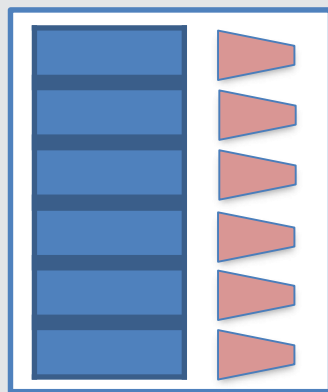




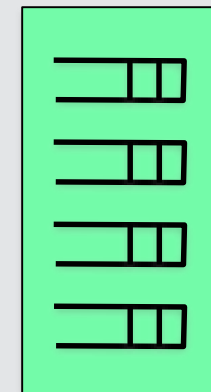
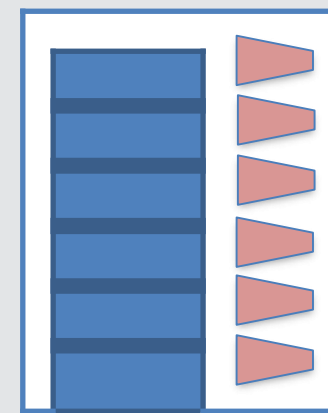
**Parse  
r**



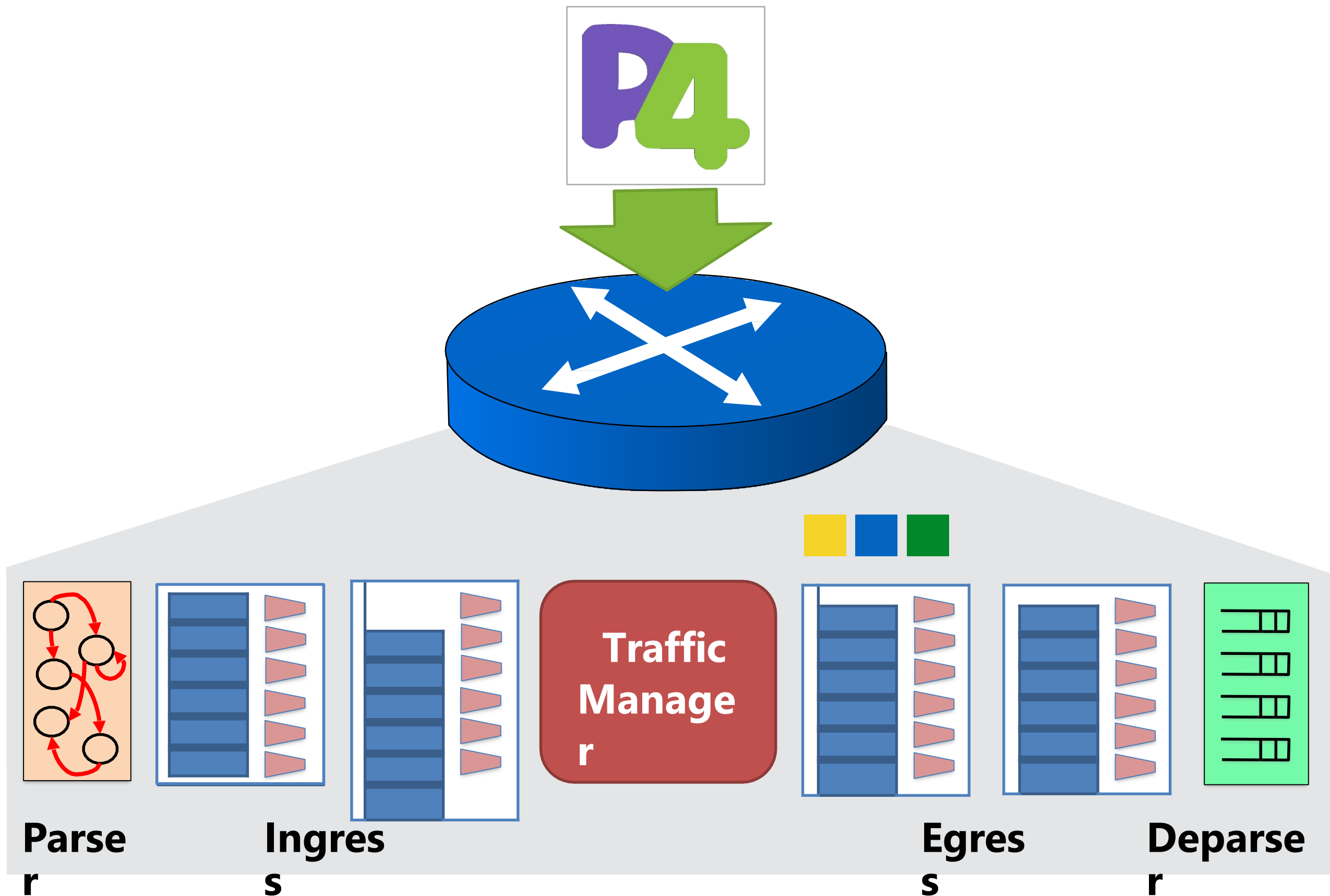
**Ingres  
s**

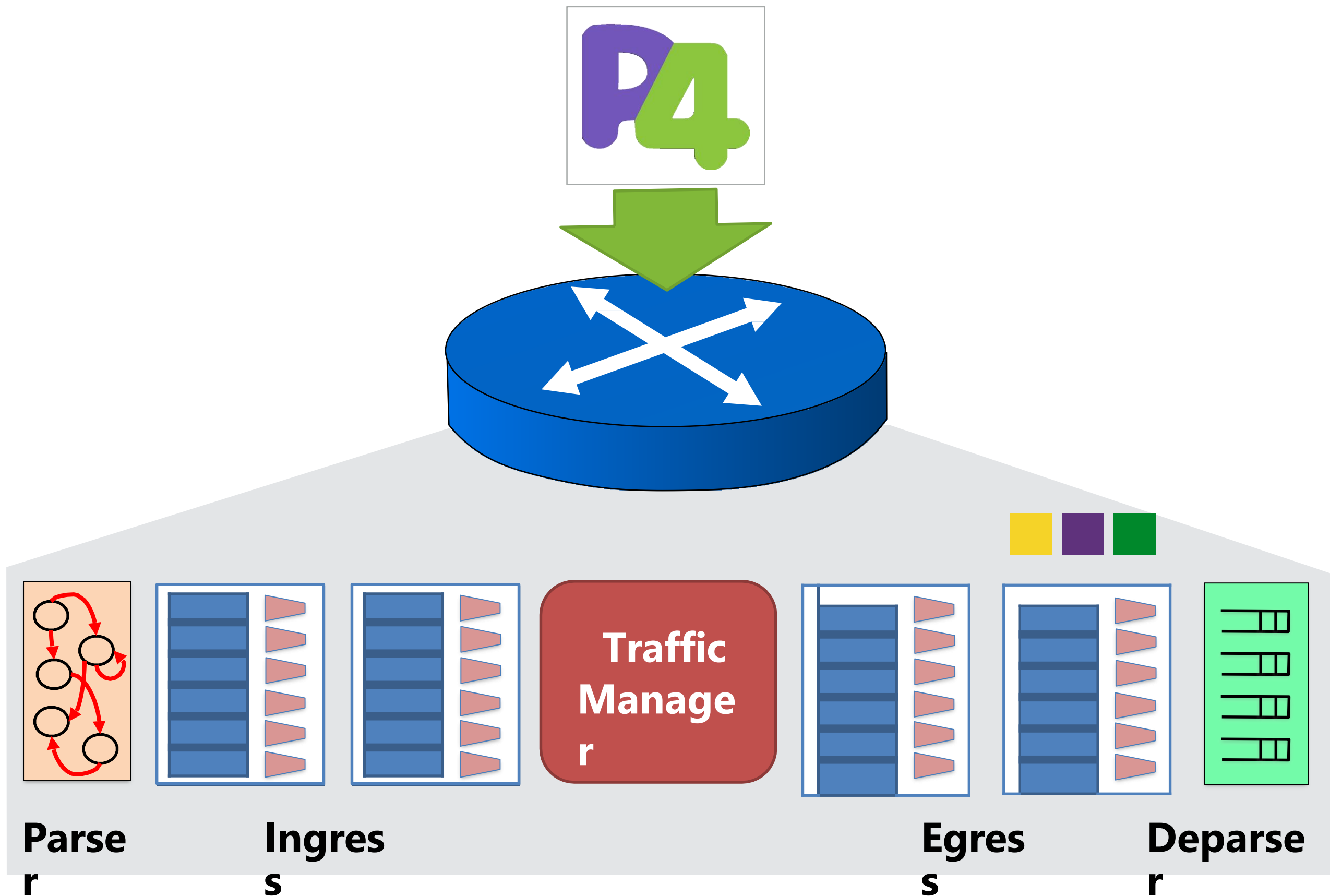


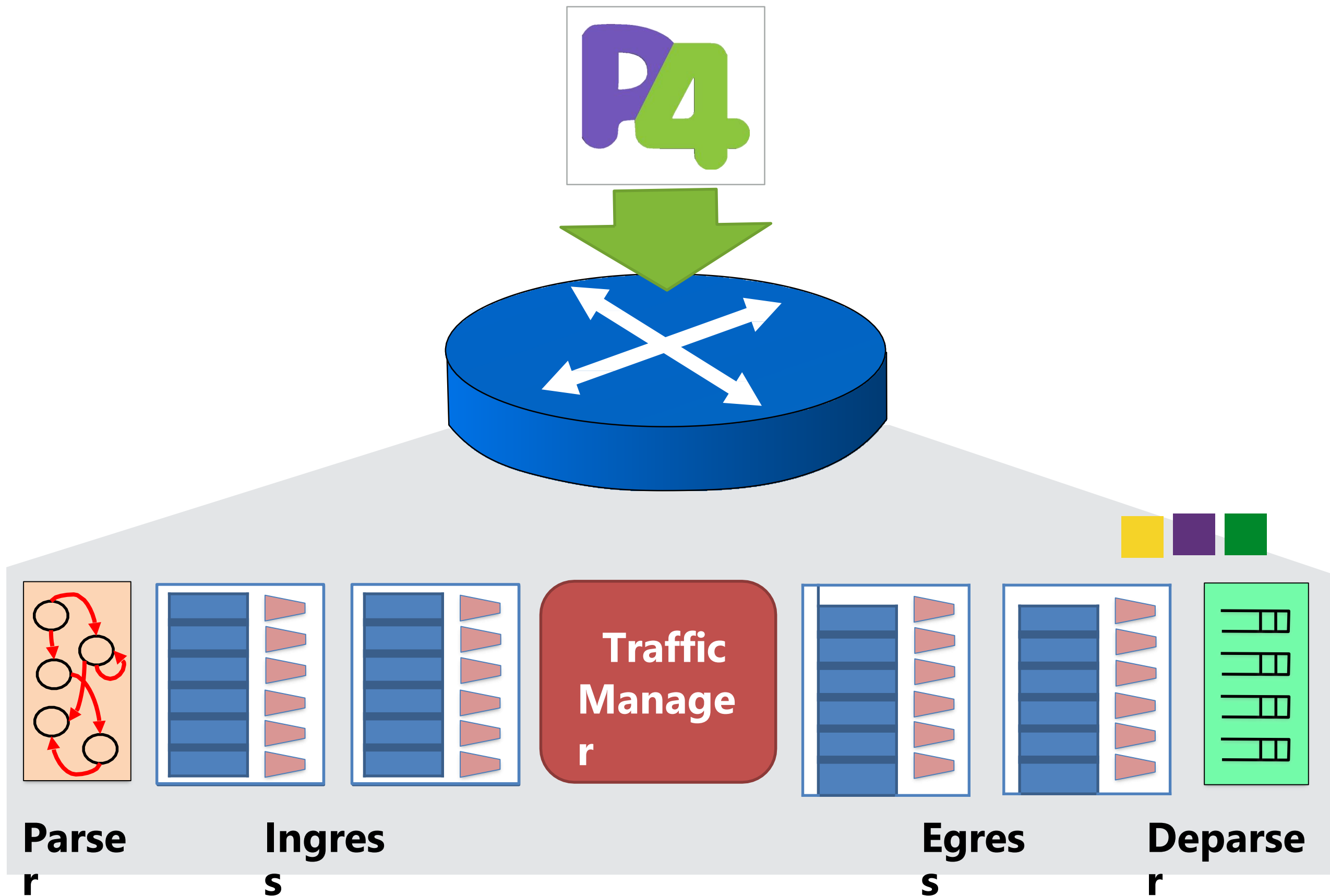
**Egres  
s**

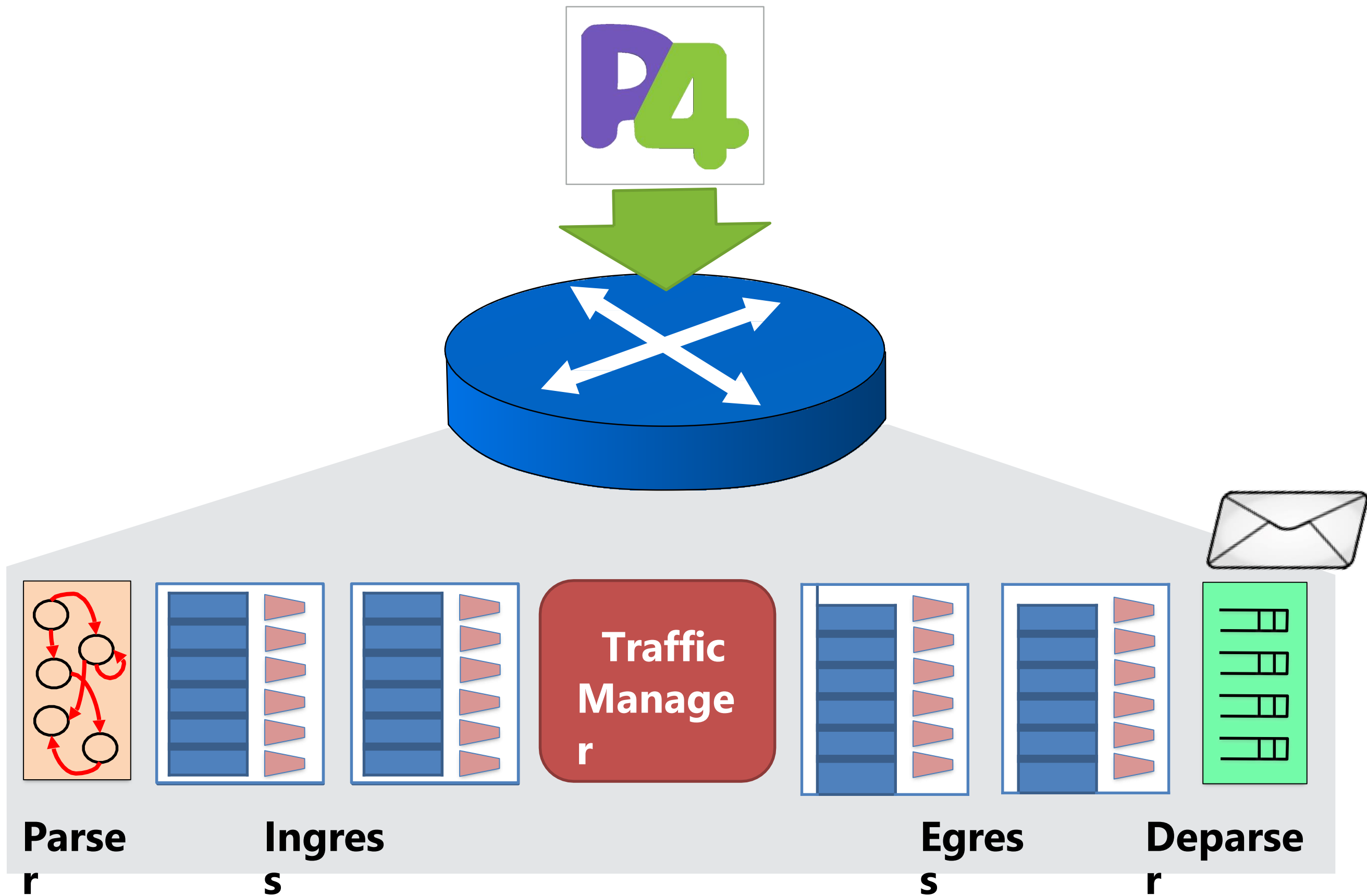


**Deparse  
r**



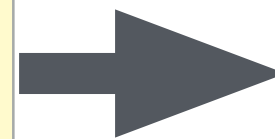






- Slogan: "constant work in constant time"
  - No pointers or complex data types
  - Bounded state
  - No loops
- Key construct is a match-action table

```
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;
}

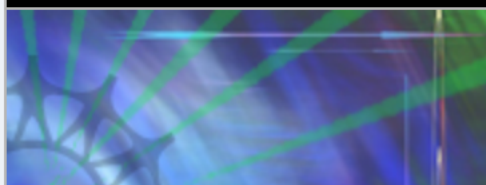
table mcast_src_pruning {
    reads {
        standard_metadata.instance_type : exact;
    }
    actions {
        _nop;
        drop;
    }
    size : 1;
}

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

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



## 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:	<a href="#">CNS</a> <a href="#">Division Of Computer and Network Systems</a>
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)