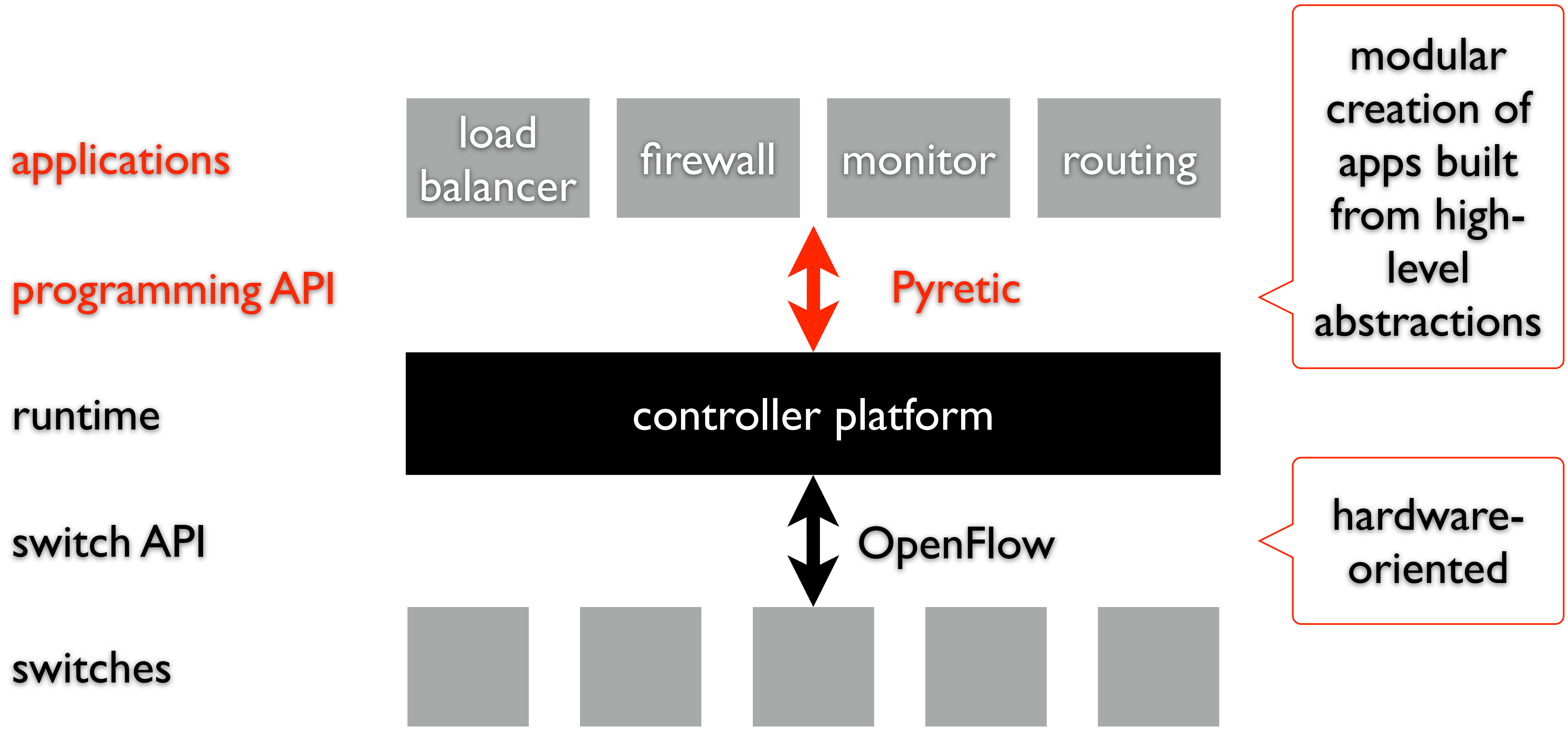
composing controllers

5590: software defined networking

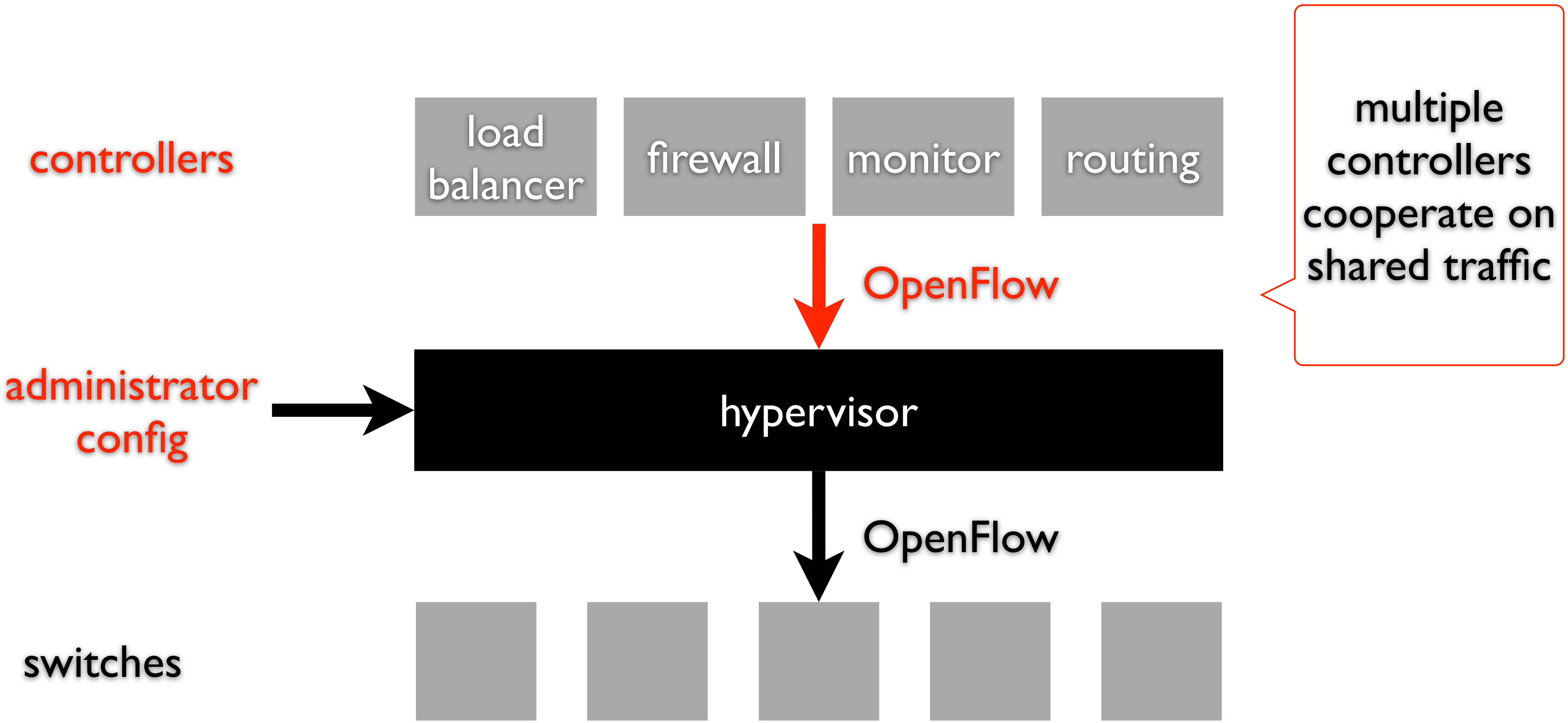
anduo wang, Temple University

T 17:30-20:00

Pyretic: composing policies

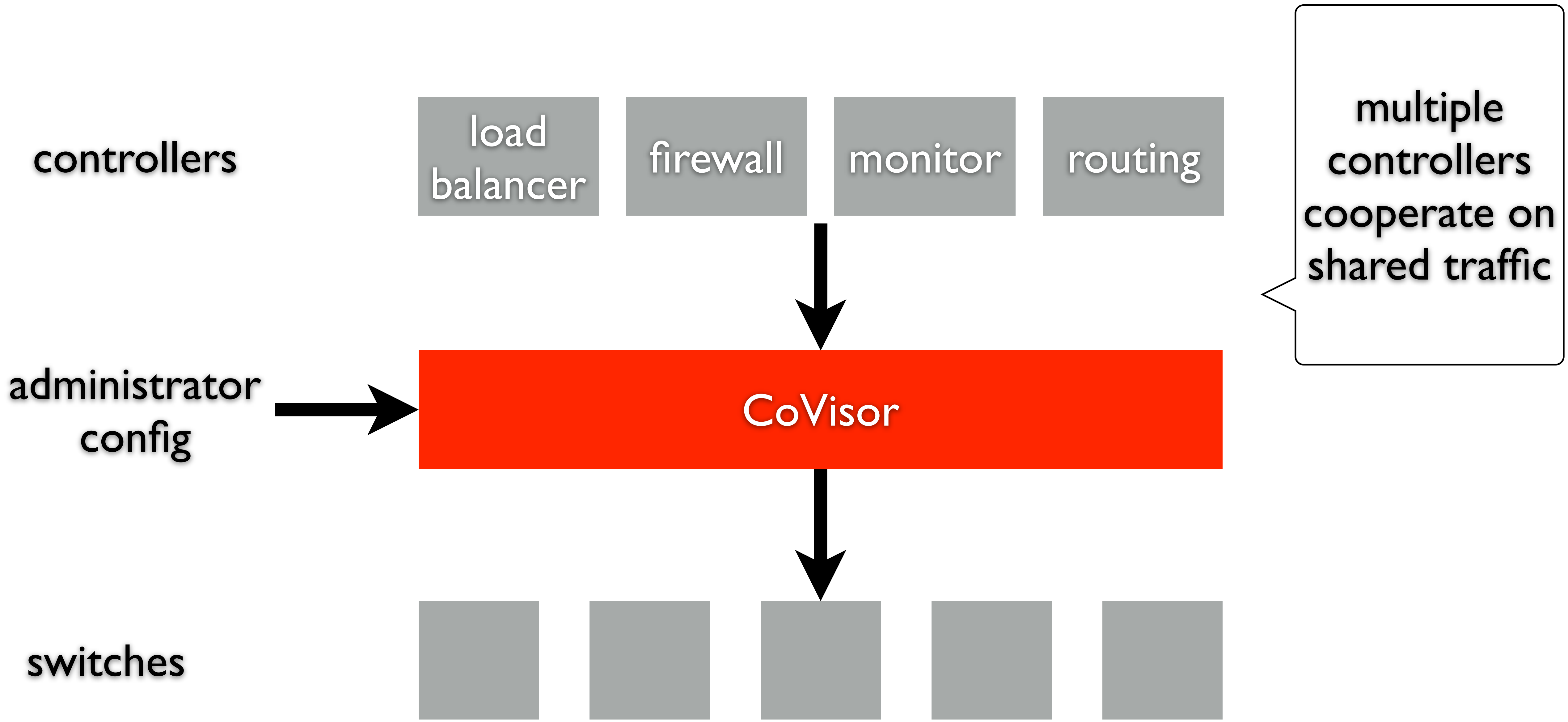


composing controllers



CoVisor

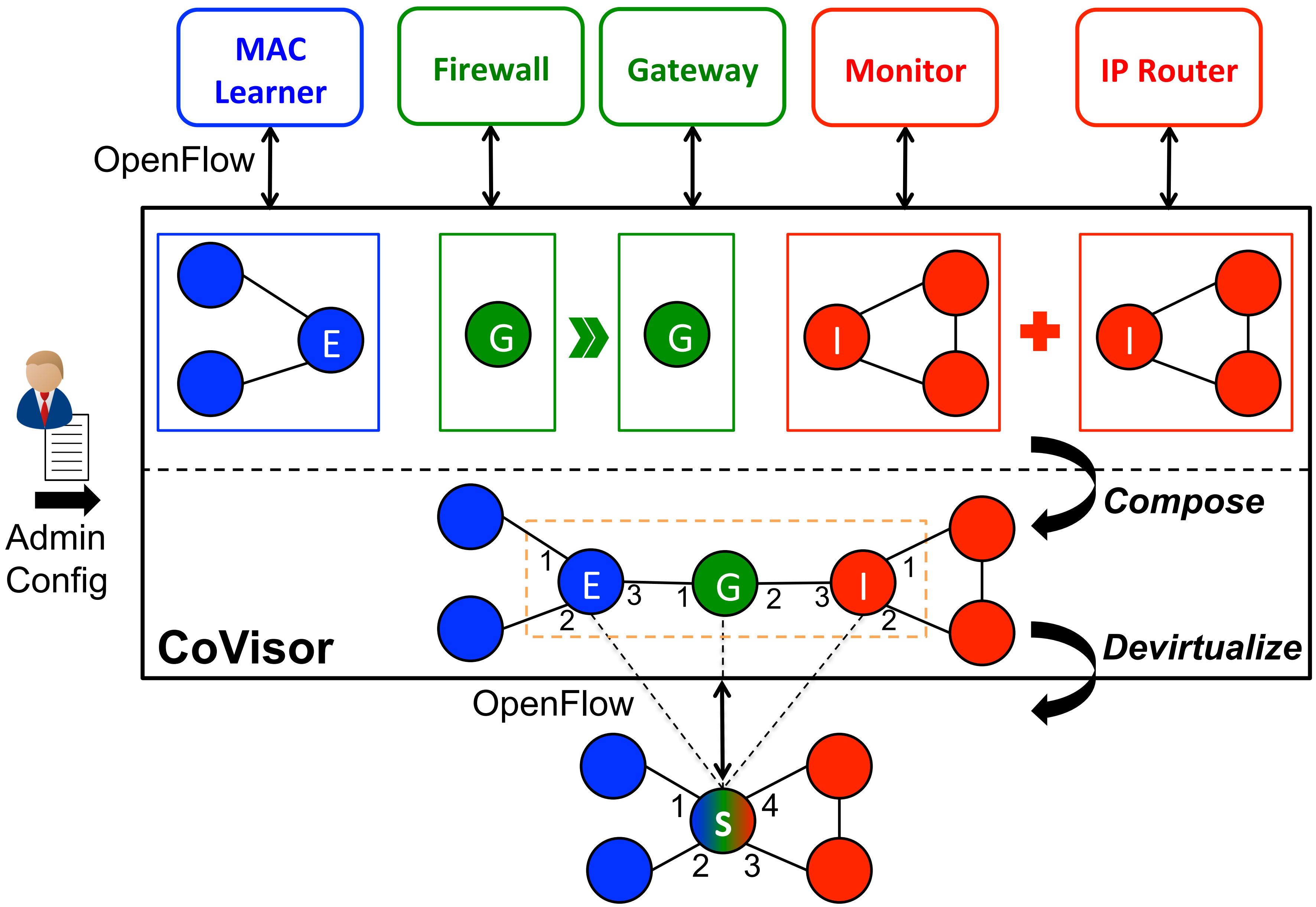
CoVisor



challenges and technical contribution

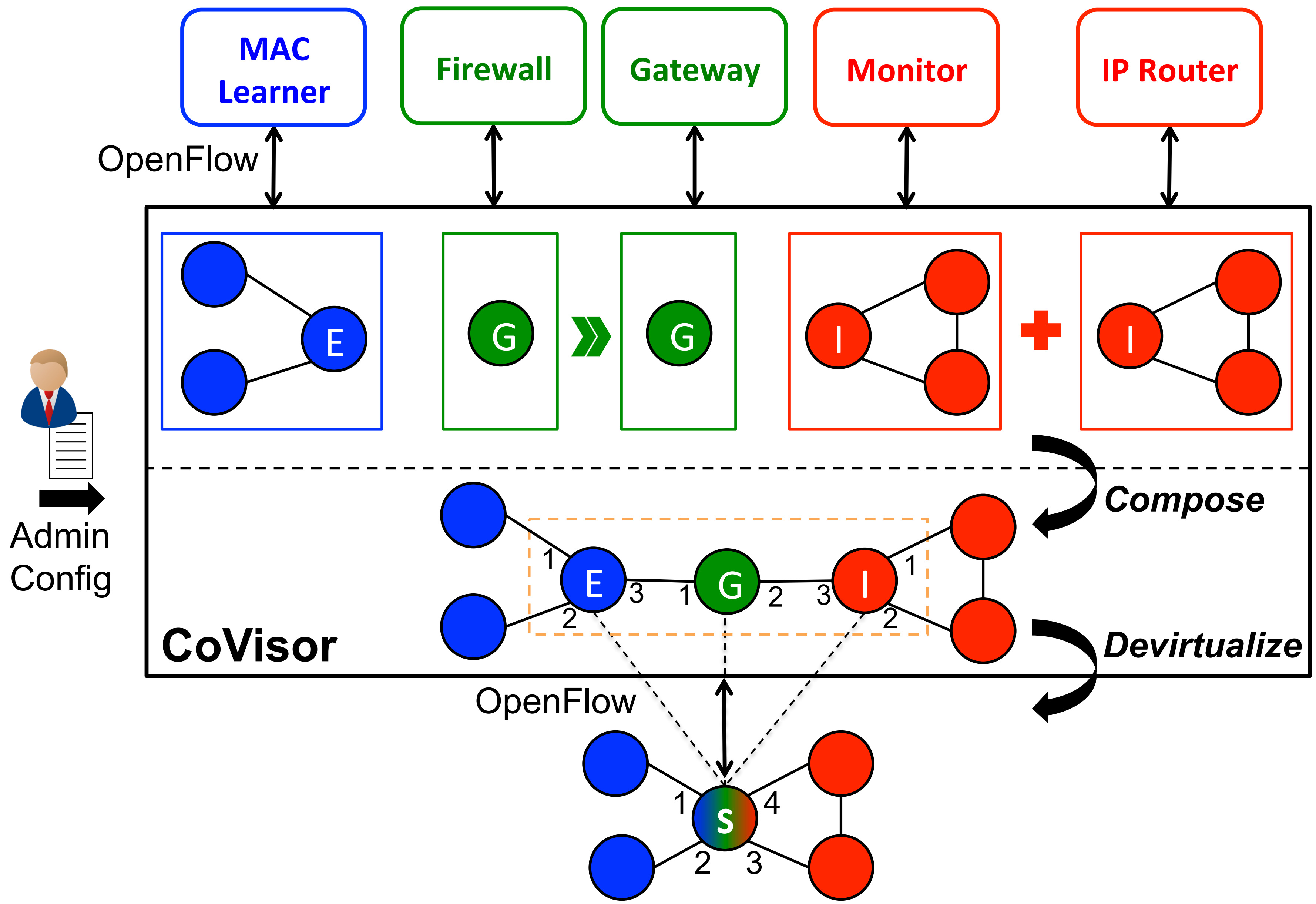
- efficient algorithms

CoVisor



MAC Learner
Firewall
Gateway
Monitor
IP Router
OpenFlow
E
G
G
I
I
Admin Config
Compose
Devirtualize
CoVisor
1
E
3
1
G
2
3
I
1
2
2
OpenFlow
S
1
4
2
3

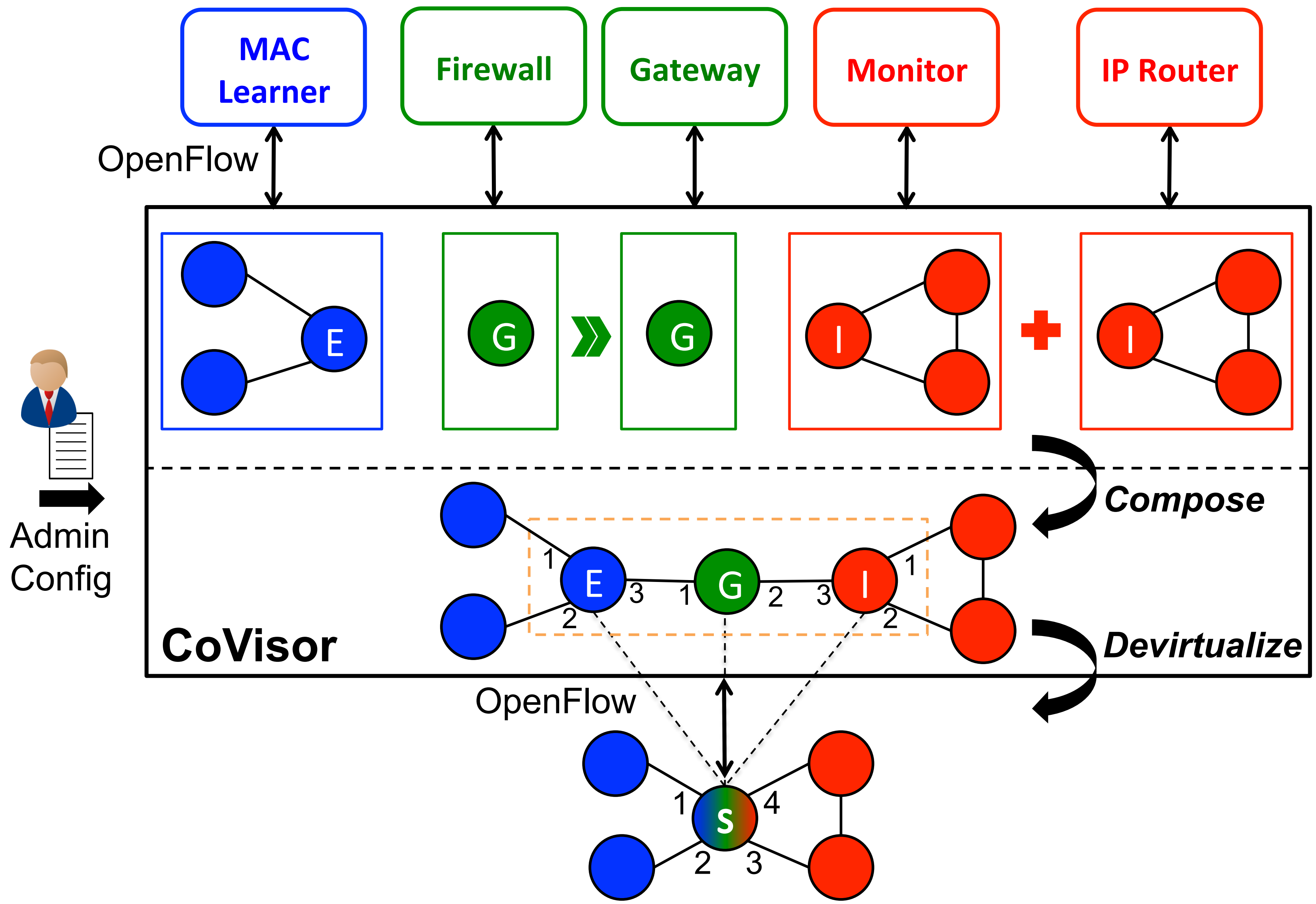
CoVisor



assemble multiple controllers

- parallel, sequential, override

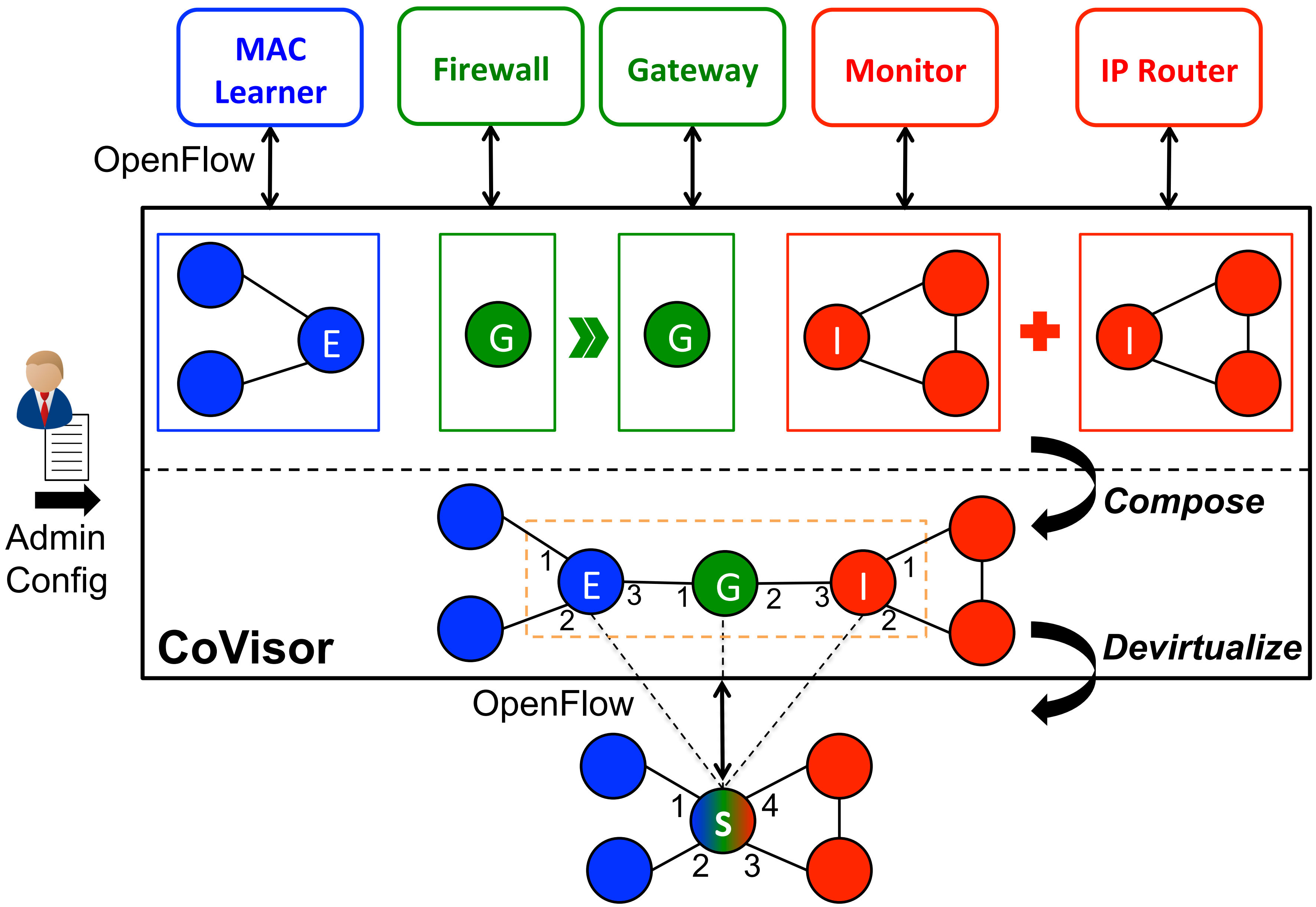
CoVisor



abstract topology

- customer virtual topology to each controller

CoVisor

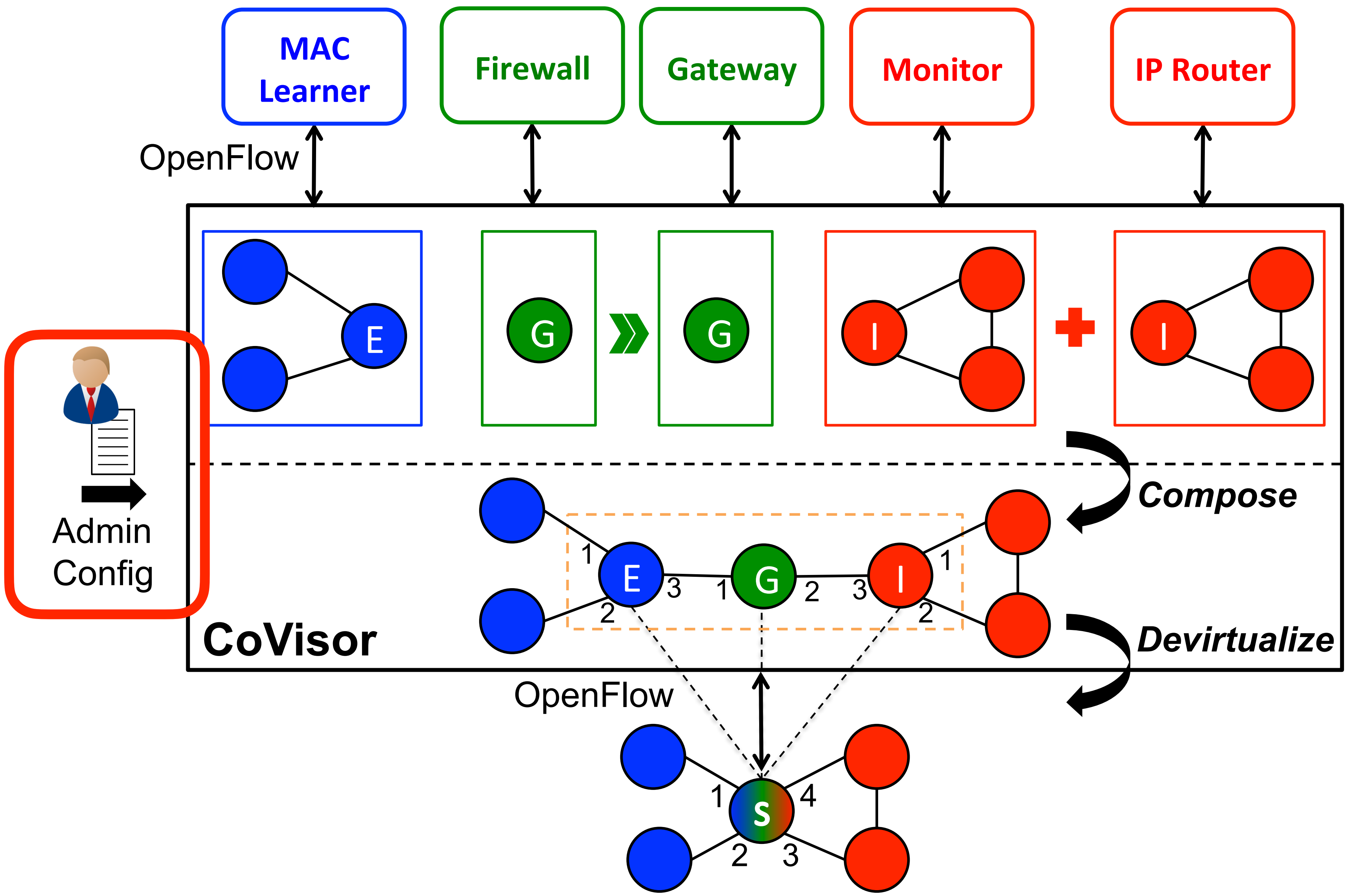


protection

- fine-grained control over how a controller can operate

CoVisor

weakness?



administrator role

administrator role

configure CoVisor to compose policies

administrator role

configure CoVisor to compose policies

- *manual* spec: $T_1 + T_2, T_1 > T_2, T_1 \triangleright T_2$

administrator role

configure CoVisor to compose policies

- *manual* spec: $T_1 + T_2$, $T_1 > T_2$, $T_1 \rhd T_2$
- *proactive* incremental compilation, optimization

administrator role

- configure CoVisor to compose policies
  - *manual* spec: $T_1+T_2, T_1>T_2, T_1 \rhd T_2$
  - *proactive* incremental compilation, optimization
- virtualize the network, sets packet-processing constraints

administrator role

configure CoVisor to compose policies

- *manual* spec: $T_1 + T_2$, $T_1 > T_2$, $T_1 \vartriangleright T_2$
- *proactive* incremental compilation, optimization

virtualize the network, sets packet-processing constraints

- virtual topo: many-to-one, one-to-many (physical-to-virtual)

administrator role

configure CoVisor to compose policies

- *manual* spec: $T_1 + T_2$, $T_1 > T_2$, $T_1 \rhd T_2$
- *proactive* incremental compilation, optimization

virtualize the network, sets packet-processing constraints

- virtual topo: many-to-one, one-to-many (physical-to-virtual)
- packet handling: match, action

the “efficiency” challenge

to host tens of controllers

- each installs tens of thousands rules
- constantly updated rules

naive approach — prohibitively expensive

- time to recompile new policy
- time to install new rules on switches

efficient CoVisor algorithms

incrementally composing controller policies

- priorities form a convenient algebra, obviating recompiling from scratch

devirtualization

- one(physical)-to-many(virtual)

optimizing composition

- smart data structure accelerate compilation

incremental composition

stream of member policy updates → CoVisor → updates to the composed policy

frequently and fast

update (OF rule) r = (p; m; a)

- a → r.action
- m → r.priority
- p → r.match, r.mSet

policy composition revisit

$comp_{+}(R_1, R_2)$

- for every (r_1, r_2) in $(R_1 \times R_2)$
- generate new r if r_1.mSet intersects with r_2.mSet
  - r.match = intersection of r_1.mSet and r_2.mSet
  - r.action = union of r_1.action and r_2.action

policy composition revisit

$comp_{»}(R_1, R_2)$

- for every (r_1, r_2) in $(R_1 \times R_2)$
- generate new r if packets produced by r_1.action intersects with r_2.mSet
  - r.match = ?
  - r.action = ?

policy composition revisit

$\mathrm{comp}_{\triangleright}(R_1, R_2)$

- stacking R1 on top of R2 with higher priority

role of priority

ideally (goal)

- single rule addition in a member policy will NOT
  - recomputing entire composed policy
  - cleaning the physical switch's flow tables

i.e., reduce update overhead

- computation
  - # of rule pairs *comp* needs to iterate
- rule update
  - # of flowmods to update a switch

strawman priority assignment

Monitoring M_R
$(1;\ srcip = 1.0.0.0/24;\ count)$
$(0;\ *;\ drop)$

Routing Q_R
$(1;\ dstip = 2.0.0.1;\ fwd(1))$
$(1;\ dstip = 2.0.0.2;\ fwd(2))$
$(0;\ *;\ drop)$

Parallel composition: $comp_+(M_R, Q_R)$
(**7**; **srcip=1.0.0.0/24,dstip=2.0.0.1**; **fwd(1),count**)
(**6**; **srcip=1.0.0.0/24,dstip=2.0.0.2**; **fwd(2),count**)
(**5**; **srcip=1.0.0.0/24,dstip=2.0.0.3**; **fwd(3),count**)
(**4**; **srcip=1.0.0.0/24**; **count**)
(**3**; **dstip=2.0.0.1**; **fwd(1)**)
(**2**; **dstip=2.0.0.2**; **fwd(2)**)
(**1**; **dstip=2.0.0.3**; **fwd(3)**)
$(0;\ *;\ drop)$

(**1**; **dstip=2.0.0.3**; **fwd(3)**)

strawman priority assignment

Monitoring M_R
$(1;\ srcip = 1.0.0.0/24;\ count)$
$(0;\ *;\ drop)$

Routing Q_R
$(1;\ dstip = 2.0.0.1;\ fwd(1))$
$(1;\ dstip = 2.0.0.2;\ fwd(2))$
$(0;\ *;\ drop)$

add

(**1**; **dstip=2.0.0.3**; **fwd(3)**)

Parallel composition: $comp_+(M_R, Q_R)$
(**7**; **srcip=1.0.0.0/24,dstip=2.0.0.1**; **fwd(1),count**)
(**6**; **srcip=1.0.0.0/24,dstip=2.0.0.2**; **fwd(2),count**)
(**5**; **srcip=1.0.0.0/24,dstip=2.0.0.3**; **fwd(3),count**)
(**4**; **srcip=1.0.0.0/24**; **count**)
(**3**; **dstip=2.0.0.1**; **fwd(1)**)
(**2**; **dstip=2.0.0.2**; **fwd(2)**)
(**1**; **dstip=2.0.0.3**; **fwd(3)**)
$(0;\ *;\ drop)$

position of the rule indicates relative priority

affecting all boldfaced rules

strawman priority assignment

Monitoring M_R
$(1;\ srcip = 1.0.0.0/24;\ count)$
$(0;\ *;\ drop)$

Routing Q_R
$(1;\ dstip = 2.0.0.1;\ fwd(1))$
$(1;\ dstip = 2.0.0.2;\ fwd(2))$
$(0;\ *;\ drop)$

add

$(\mathbf{1};\ \textbf{dstip=2.0.0.3};\ \textbf{fwd(3)})$

Parallel composition: $comp_{+}(M_R, Q_R)$
$(\mathbf{7};\ \textbf{srcip=1.0.0.0/24,dstip=2.0.0.1};\ \textbf{fwd(1),count})$
$(\mathbf{6};\ \textbf{srcip=1.0.0.0/24,dstip=2.0.0.2};\ \textbf{fwd(2),count})$
$(\mathbf{5};\ \textbf{srcip=1.0.0.0/24,dstip=2.0.0.3};\ \textbf{fwd(3),count})$
$(\mathbf{4};\ \textbf{srcip=1.0.0.0/24};\ \textbf{count})$
$(\mathbf{3};\ \textbf{dstip=2.0.0.1};\ \textbf{fwd(1)})$
$(\mathbf{2};\ \textbf{dstip=2.0.0.2};\ \textbf{fwd(2)})$
$(\mathbf{1};\ \textbf{dstip=2.0.0.3};\ \textbf{fwd(3)})$
$(0;\ *;\ drop)$

rules in bold count toward rule update overhead

strawman priority assignment

Monitoring M_R
$(1;\ srcip = 1.0.0.0/24;\ count)$
$(0;\ *;\ drop)$

Routing Q_R
$(1;\ dstip = 2.0.0.1;\ fwd(1))$
$(1;\ dstip = 2.0.0.2;\ fwd(2))$
$(0;\ *;\ drop)$

add

(**1**; **dstip=2.0.0.3**; **fwd(3)**)

Parallel composition: $comp_+(M_R, Q_R)$
(**7**; **srcip=1.0.0.0/24,dstip=2.0.0.1**; **fwd(1),count**)
(**6**; **srcip=1.0.0.0/24,dstip=2.0.0.2**; **fwd(2),count**)
(**5**; **srcip=1.0.0.0/24,dstip=2.0.0.3**; **fwd(3),count**)
(**4**; **srcip=1.0.0.0/24**; **count**)
(**3**; **dstip=2.0.0.1**; **fwd(1)**)
(**2**; **dstip=2.0.0.2**; **fwd(2)**)
(**1**; **dstip=2.0.0.3**; **fwd(3)**)
$(0;\ *;\ drop)$

rules in bold count toward rule update overhead

smartly set priority

- to make updates incremental

incremental update and priority algebra

r is computed from r_1 and r_2

- r.priority $\longleftarrow$ r_1.priority, r_2.priority
- incremental update without modifying existing priorities

incremental update and priority algebra

r is computed from r_1 and r_2

- $r.priority \longleftarrow r_1.priority, r_2.priority$

comp$_+$

- $r.priority = r_1.priority + r_2.priority$

comp$_{<<}$

- $r.priority = r_1.priority \times MAX_2 + r_2.priority$

incremental update and priority algebra

r is computed ($comp_{\triangleright}$) from R_1 and R_2

- r.priority = r.priority + MAX_2 if r in R1
- r.priority = r.priority if r in R2

algebra properties

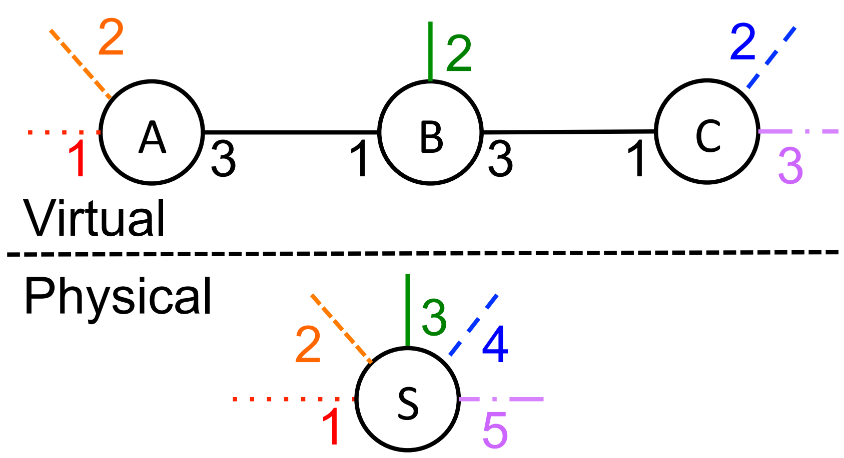
identify and prove properties

- the assignment schema ensures newly generated priority
  - leaves existing priority unchanged
  - together, the new and existing priorities are compliant with the straw man scheme

devirtualization

topology transformation for one-to-many

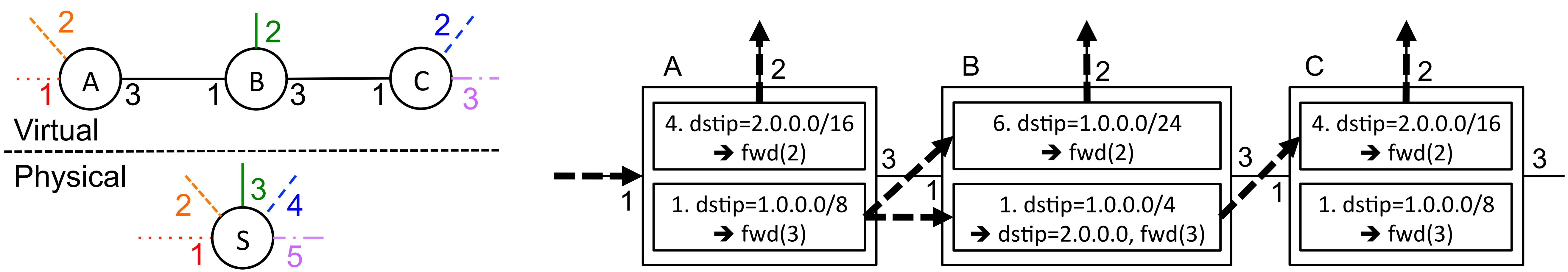
- generate symbolic path (from the virtual ingress to egress)
- on each virtual path, sequentially compose virtual policies into a single (physical) rule



devirtualization

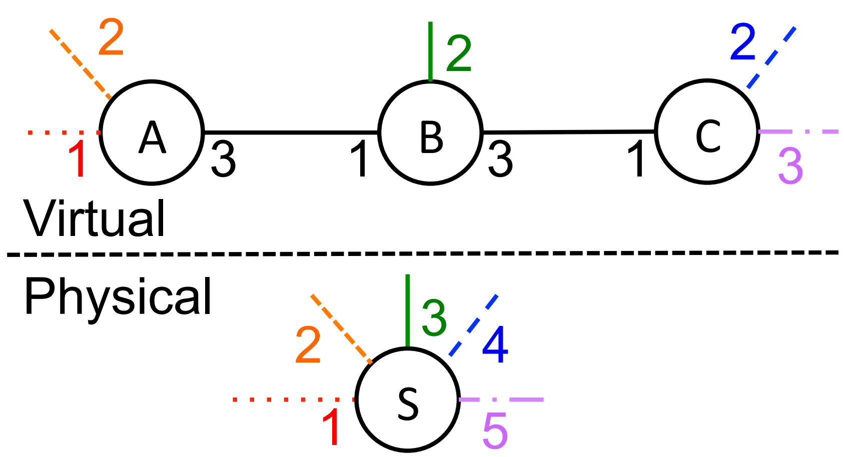
on the virtual topology, find symbolic paths

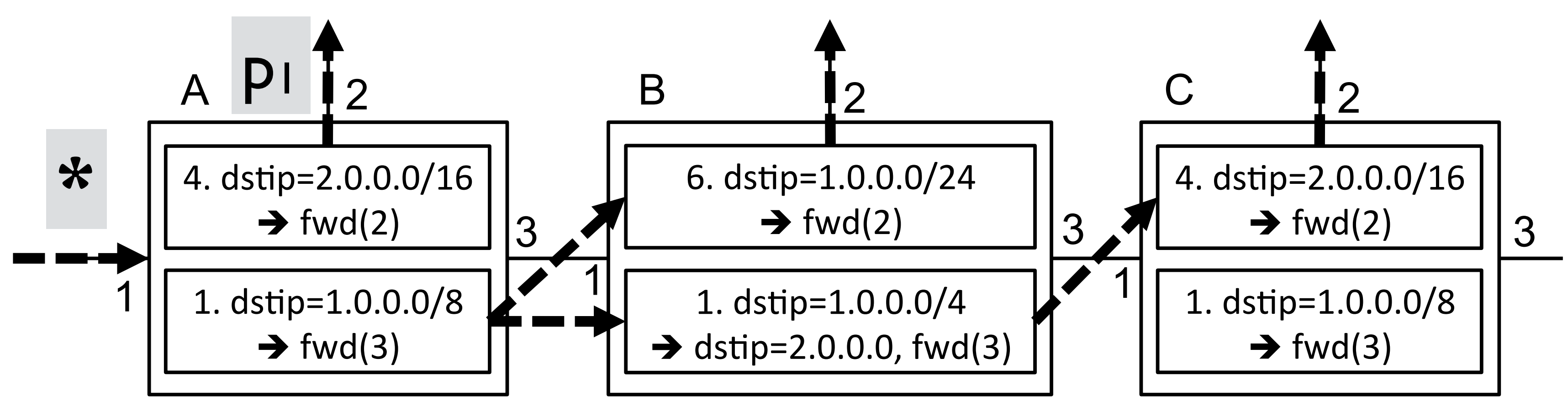
- inject wildcard packet * at ingress
- at each hop
  - evaluate the virtual policy, resulting in new packets
- until all packets reach egress



devirtualization

on the virtual topology, find symbolic paths



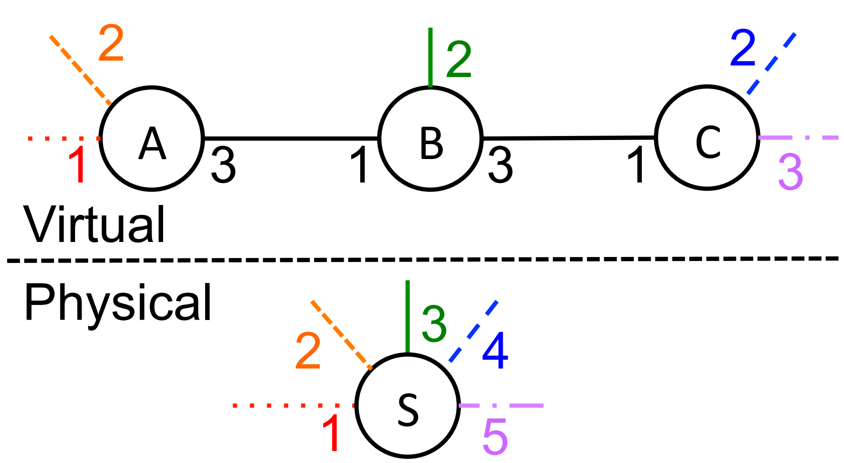


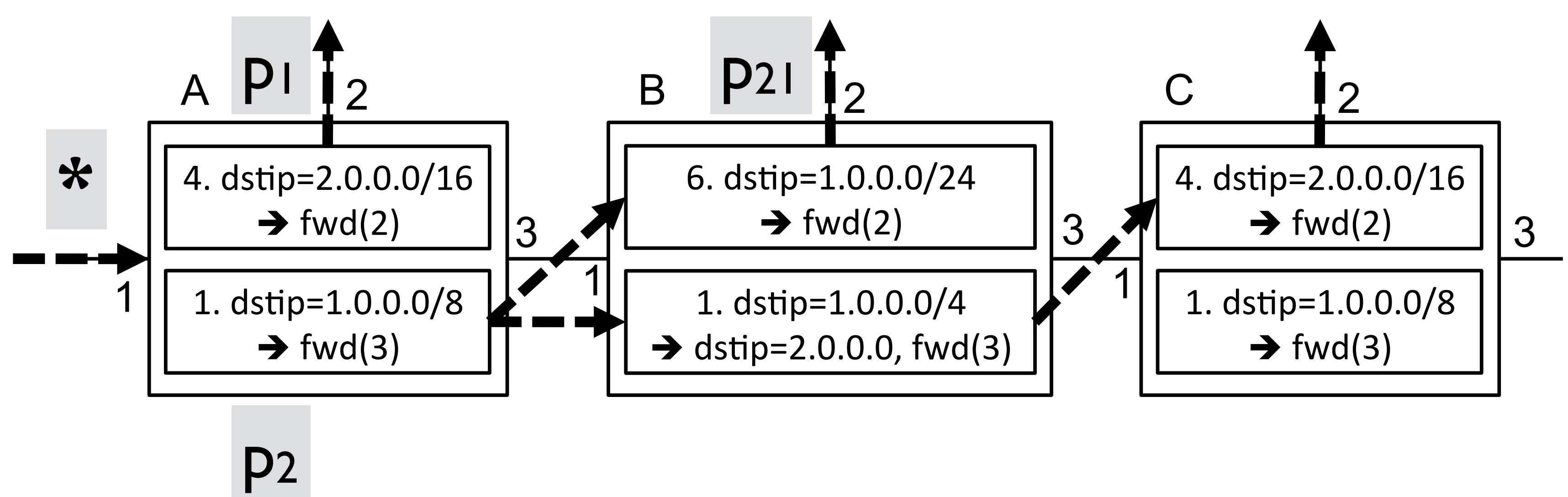
devirtualization

on the virtual topology, find symbolic paths

$* \longrightarrow A(R_1) \longrightarrow p_1$

$* \longrightarrow A(R_2) \longrightarrow p_2 \longrightarrow B(R_1) \longrightarrow p_{21}$





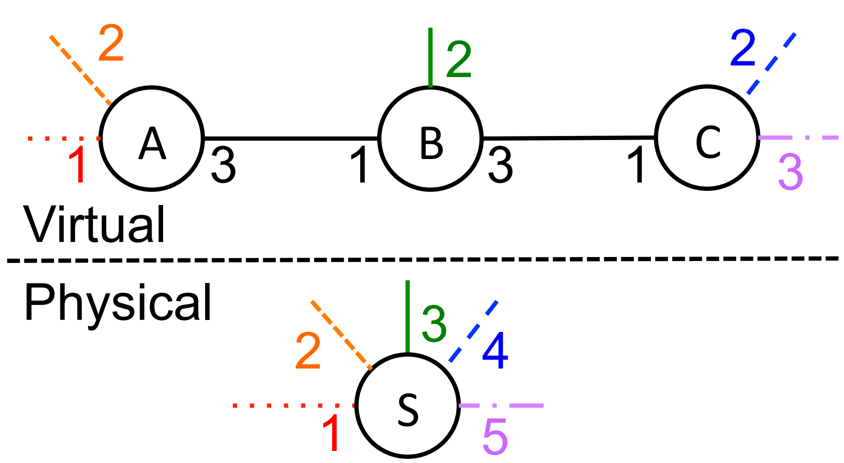
devirtualization

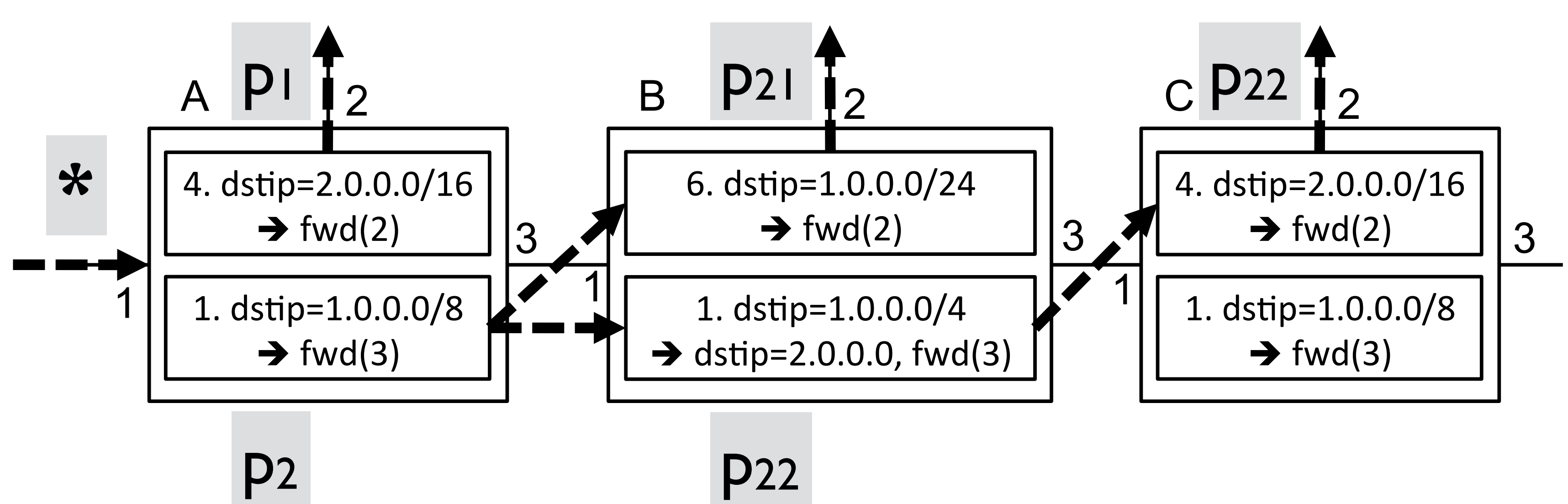
on the virtual topology, find symbolic paths

$$* \rightarrow A(R_1) \rightarrow p_1$$

$$* \rightarrow A(R_2) \rightarrow p_2 \rightarrow B(R_1) \rightarrow p_{21}$$

$$* \rightarrow A(R_2) \rightarrow p_2 \rightarrow B(R_2) \rightarrow p_{22} \rightarrow C(R_1) \rightarrow p_{22}$$





devirtualization

sequentially compose policies on each path

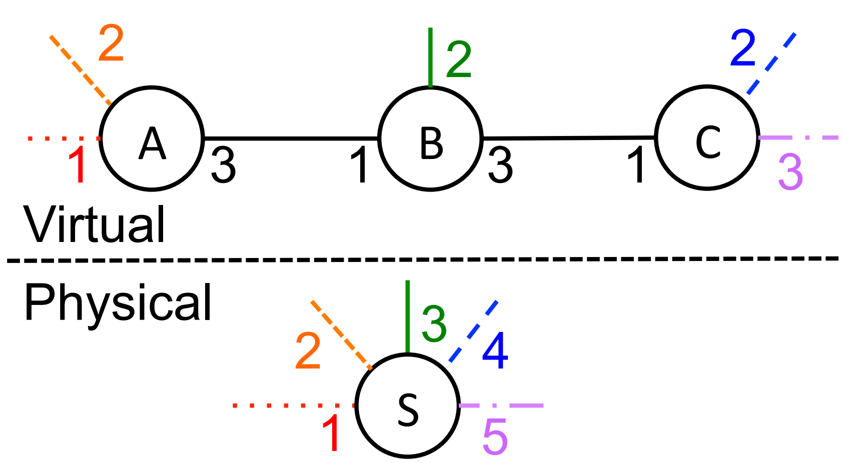
priority

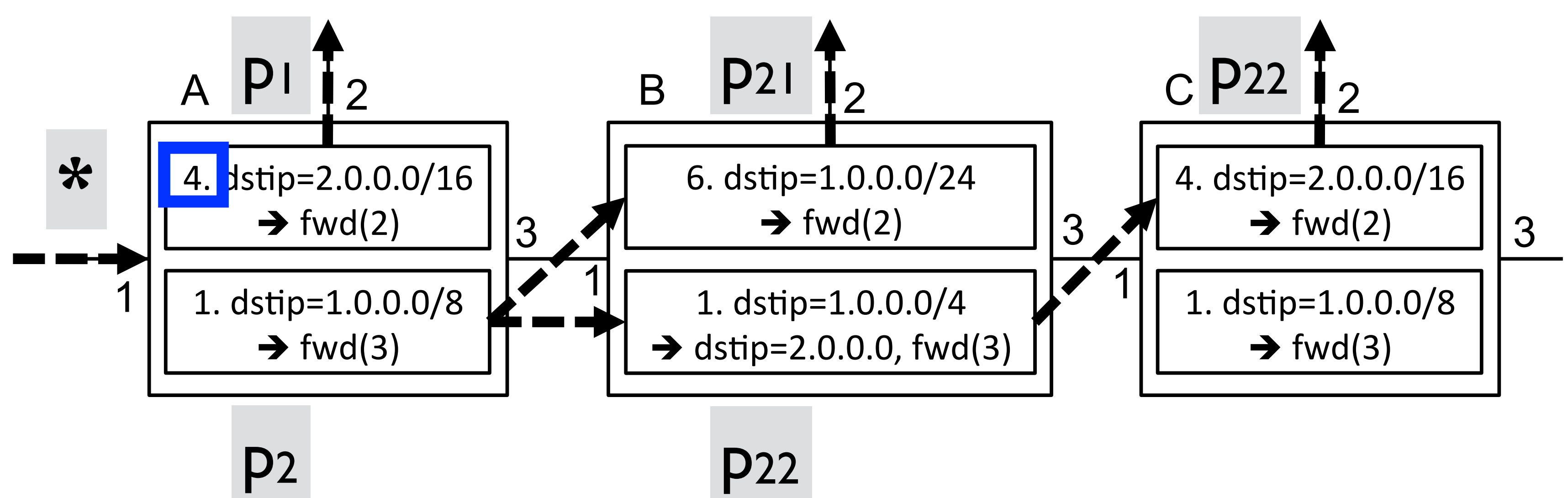
4

$* \longrightarrow A(R_1) \longrightarrow p_1$

$* \longrightarrow A(R_2) \longrightarrow p_2 \longrightarrow B(R_1) \longrightarrow p_{21}$

$* \longrightarrow A(R_2) \longrightarrow p_2 \longrightarrow B(R_2) \longrightarrow p_{22} \longrightarrow C(R_1) \longrightarrow p_{22}$

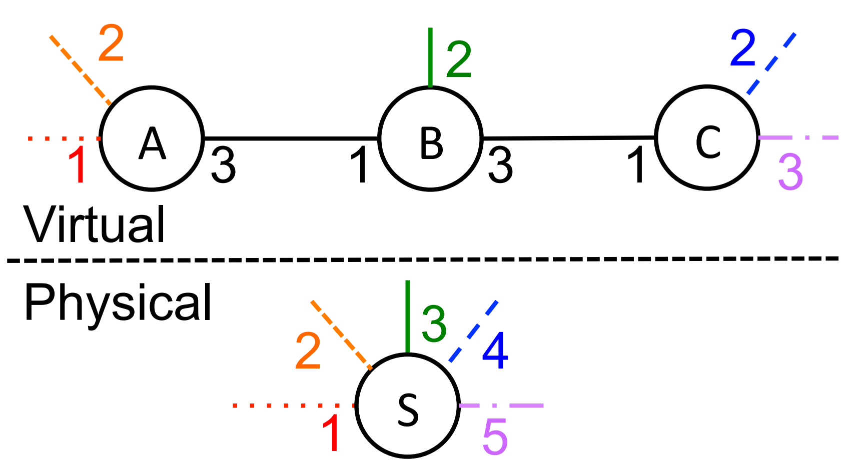


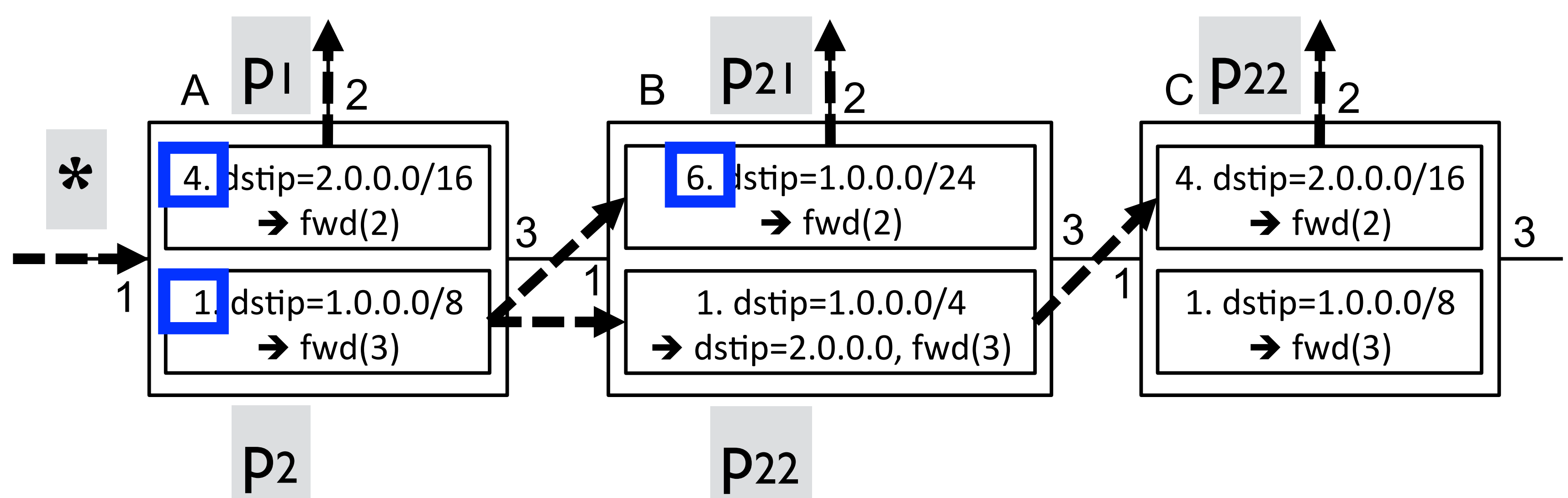


devirtualization

sequentially compose policies on each path

priority	
4	$* \rightarrow A(R_1) \rightarrow p_1$
$1 \circ 6$ (=14)	$* \rightarrow A(R_2) \rightarrow p_2 \rightarrow B(R_1) \rightarrow p_{21}$
	$* \rightarrow A(R_2) \rightarrow p_2 \rightarrow B(R_2) \rightarrow p_{22} \rightarrow C(R_1) \rightarrow p_{22}$



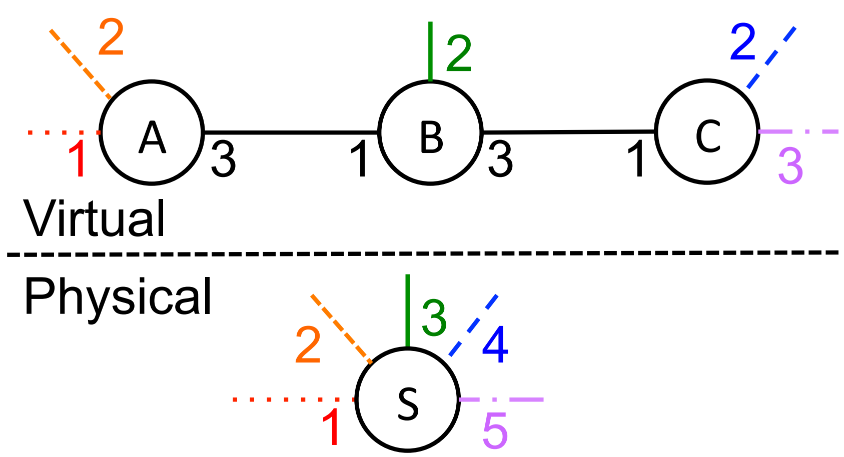


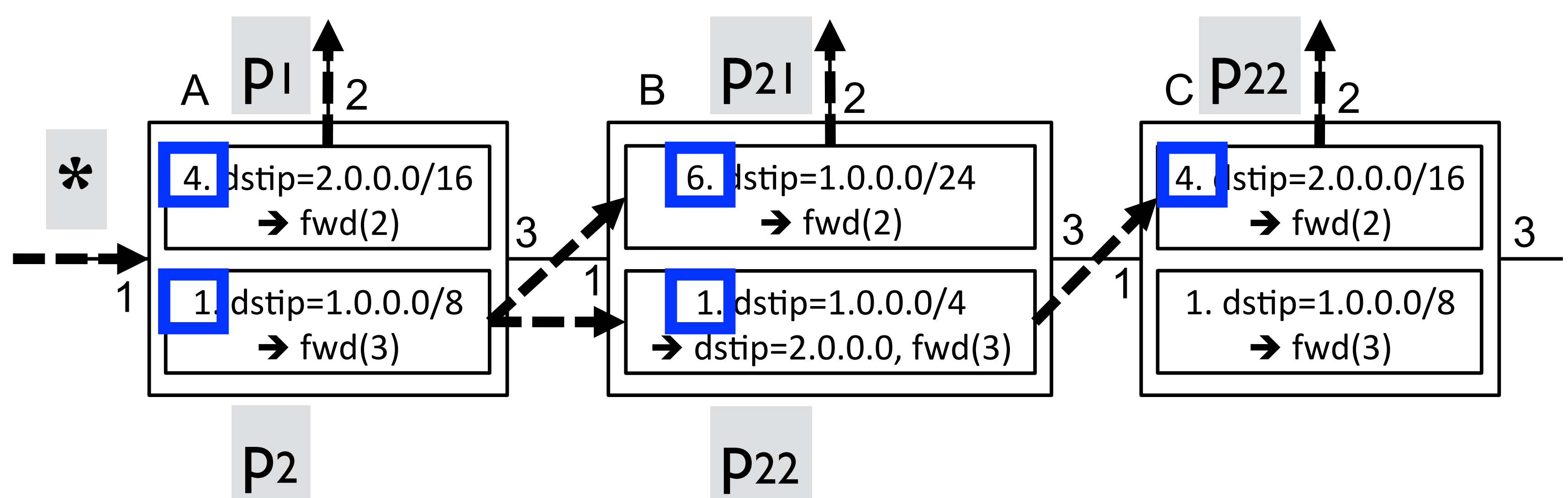
devirtualization

sequentially compose policies on each path

priority (assuming priority space for each switch is [0.8))

4		$* \rightarrow A(R_1) \rightarrow p_1$
$1 \circ 6$	(=14)	$* \rightarrow A(R_2) \rightarrow p_2 \rightarrow B(R_1) \rightarrow p_{21}$
$1 \circ 1 \circ 4$	(=76)	$* \rightarrow A(R_2) \rightarrow p_2 \rightarrow B(R_2) \rightarrow p_{22} \rightarrow C(R_1) \rightarrow p_{22}$





devirtualization

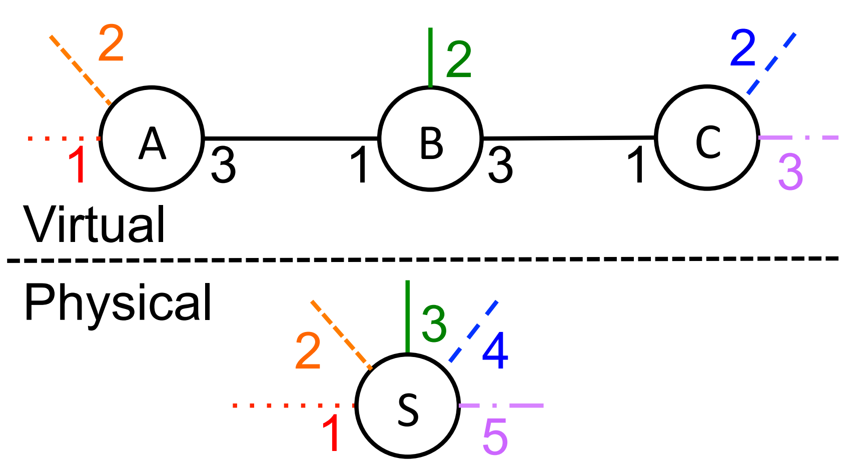
sequentially compose policies on each path

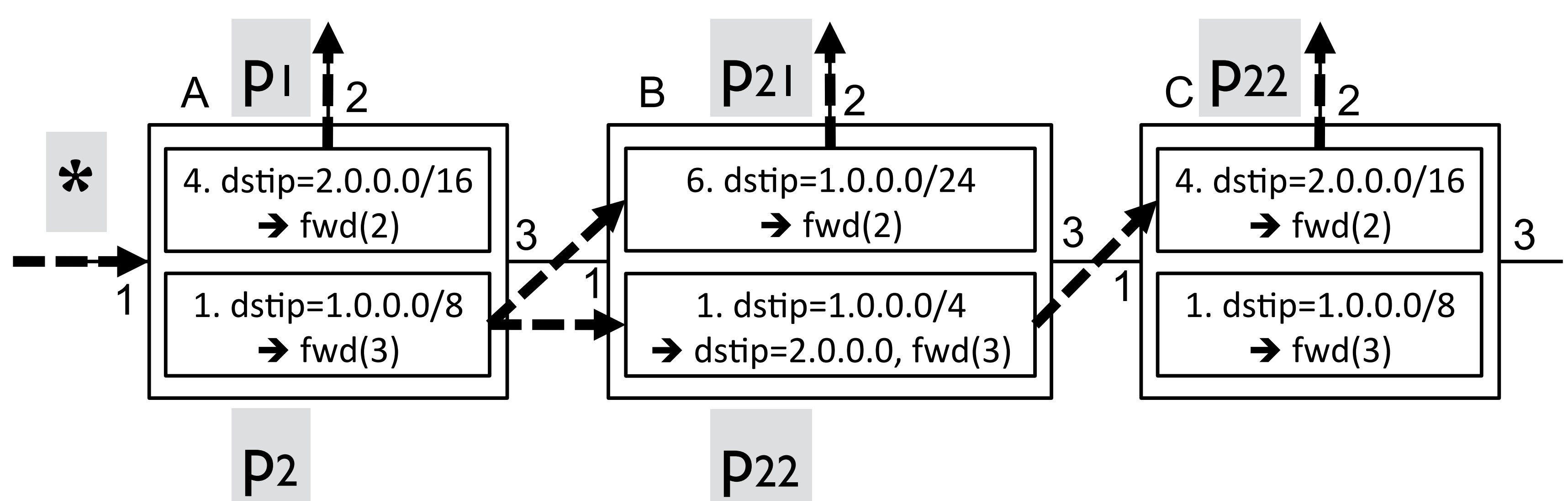
priority

$4 \circ 0 \circ 0$ (=256) $* \longrightarrow A(R_1) \longrightarrow p_1$

$1 \circ 6 \circ 0$ (=112) $* \longrightarrow A(R_2) \longrightarrow p_2 \longrightarrow B(R_1) \longrightarrow p_{21}$

$1 \circ 1 \circ 4$ (=76) $* \longrightarrow A(R_2) \longrightarrow p_2 \longrightarrow B(R_2) \longrightarrow p_{22} \longrightarrow C(R_1) \longrightarrow p_{22}$





devirtualization

sequentially compose policies on each path

flow table of S

256; *inport* = 1,*dstip* = 2.0.0.0/16; ; *fwd*(2)

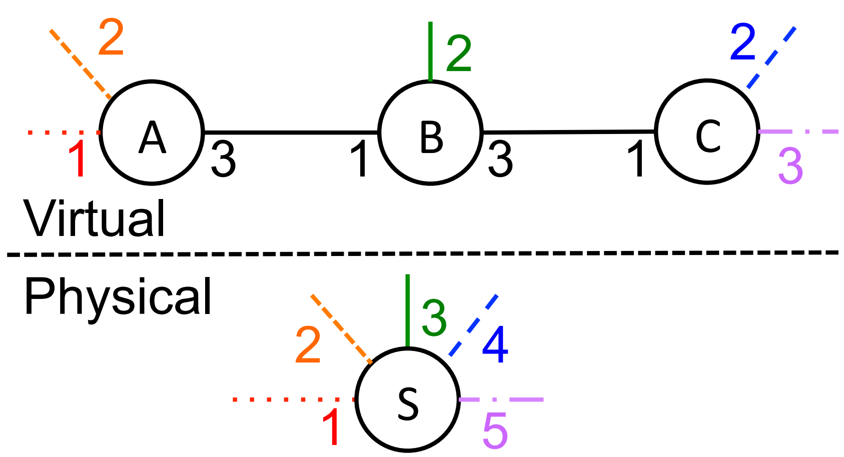
112; inport = 1,dstip = 1.0.0.0/24 ; fwd(3)

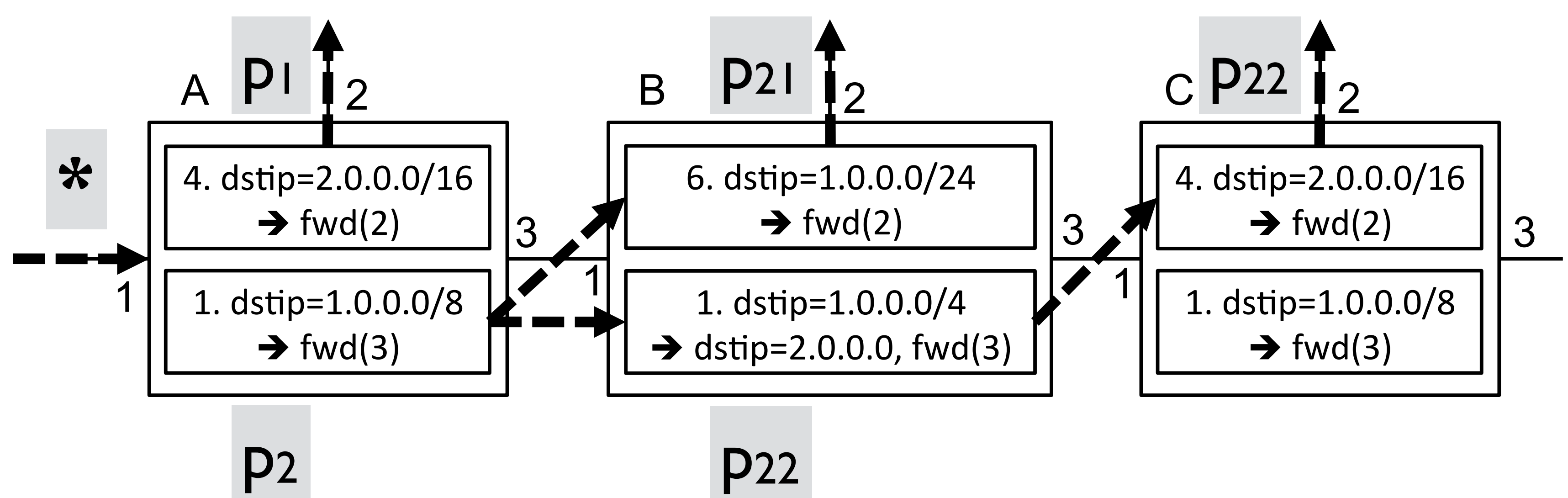
76; inport = 1,dstip = 1.0.0.0/8; dstip = 2.0.0.0, fwd(4)

$* \rightarrow A(R_1) \rightarrow p_1$

$* \rightarrow A(R_2) \rightarrow p_2 \rightarrow B(R_1) \rightarrow p_{21}$

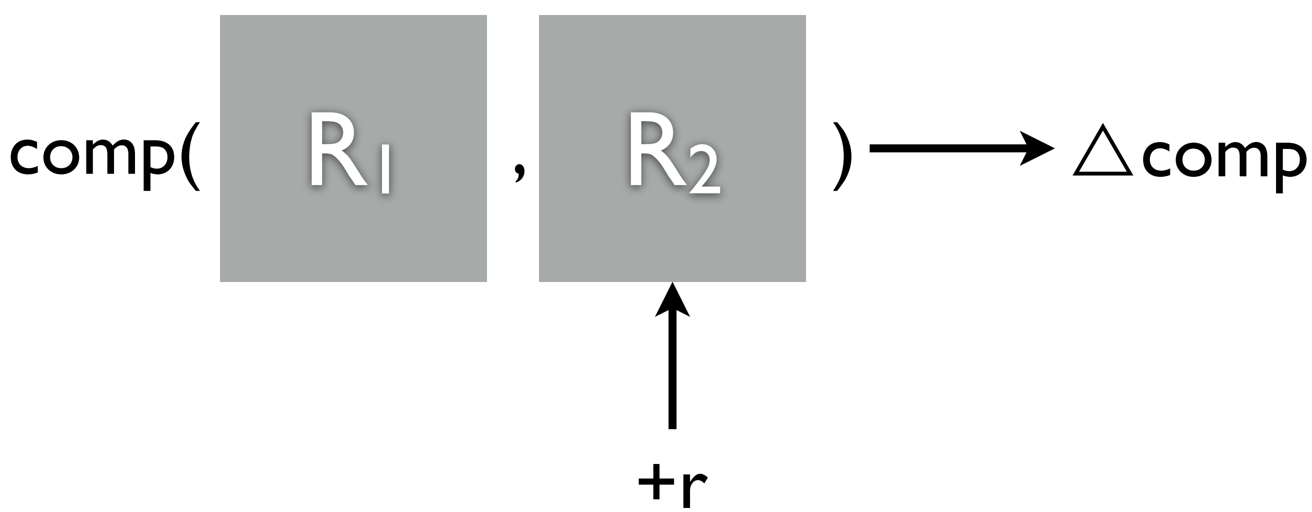
$* \rightarrow A(R_2) \rightarrow p_2 \rightarrow B(R_2) \rightarrow p_{22} \rightarrow C(R_1) \rightarrow p_{22}$





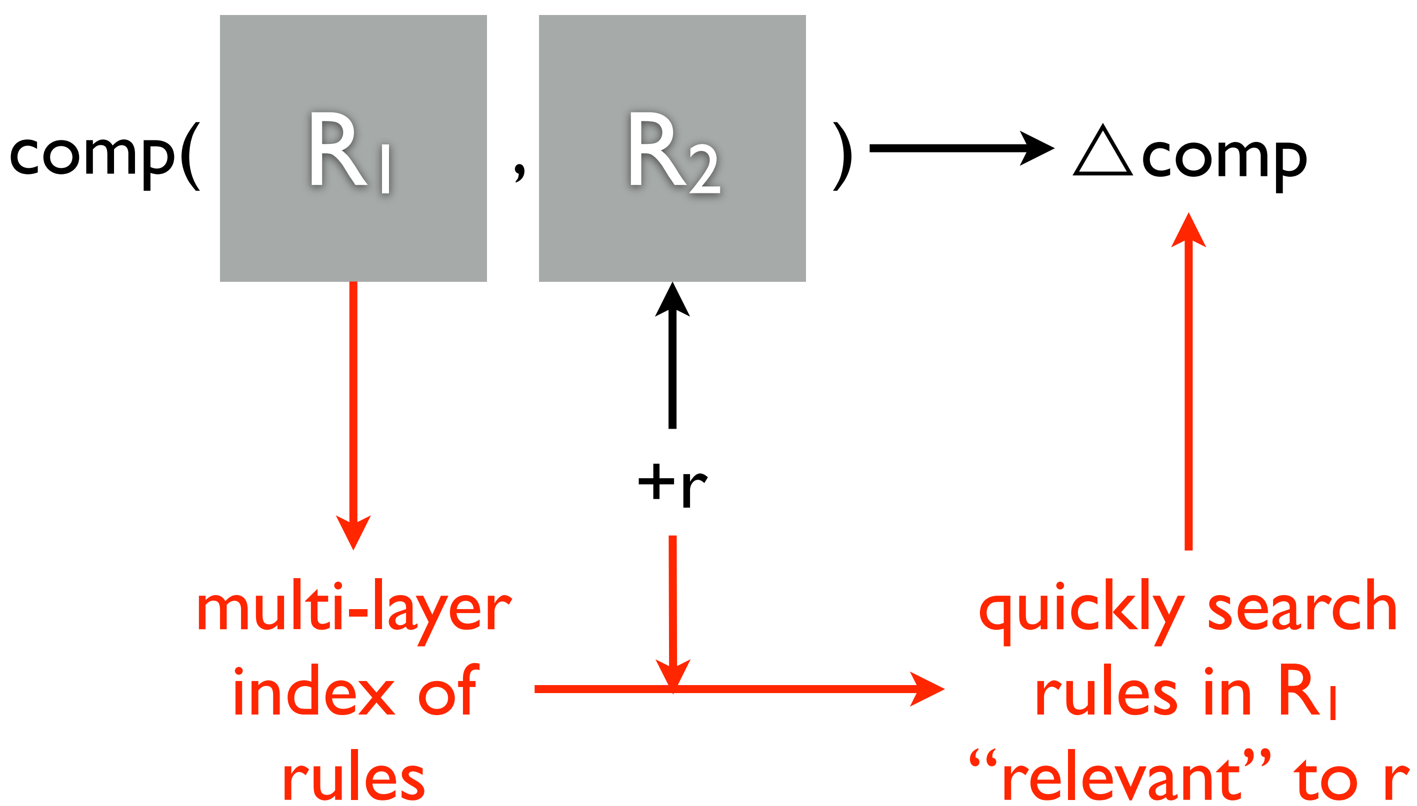
optimization: indexing rules

accelerate complication with smart data structure



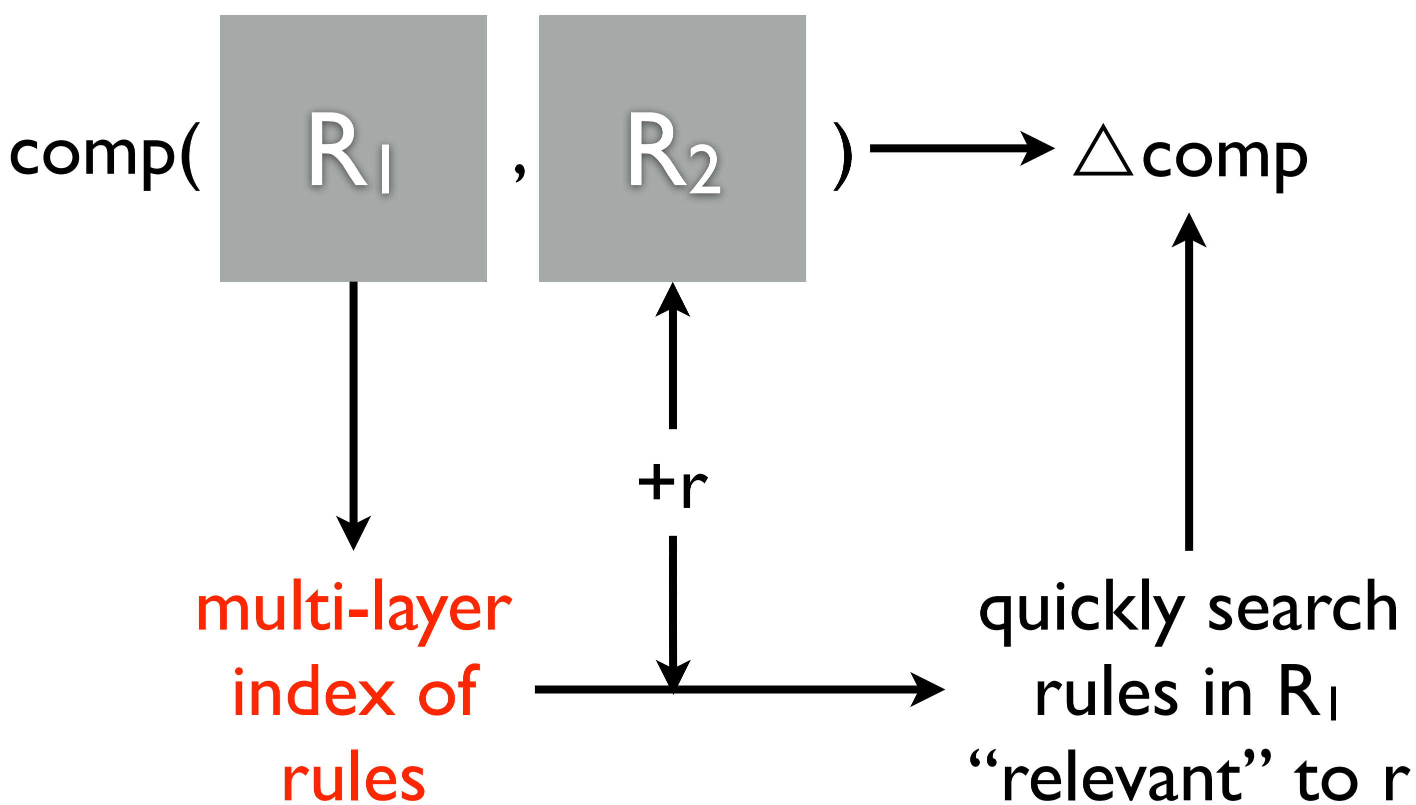
optimization: indexing rules

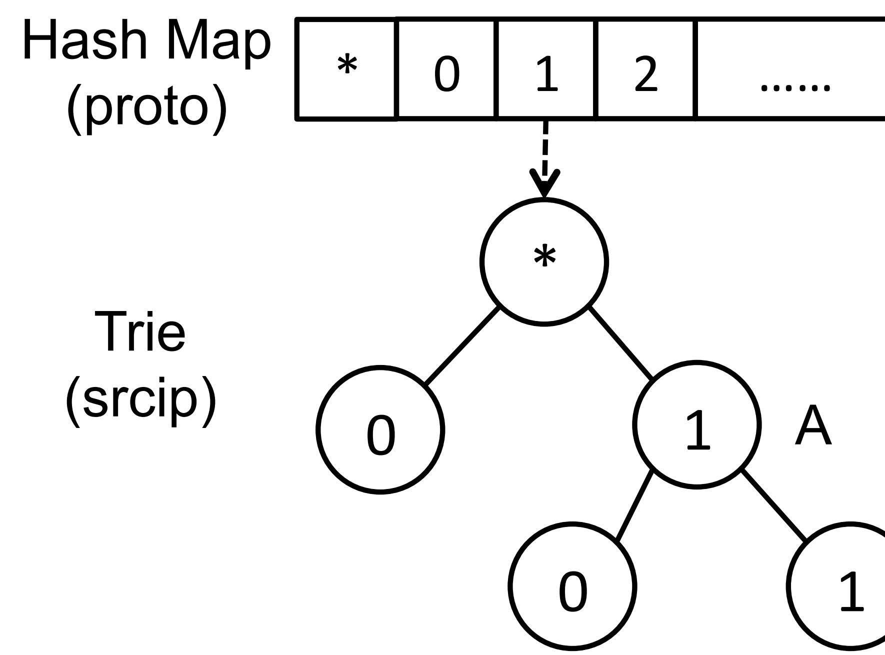
accelerate complication with smart data structure



optimization: indexing rules

accelerate complication with smart data structure



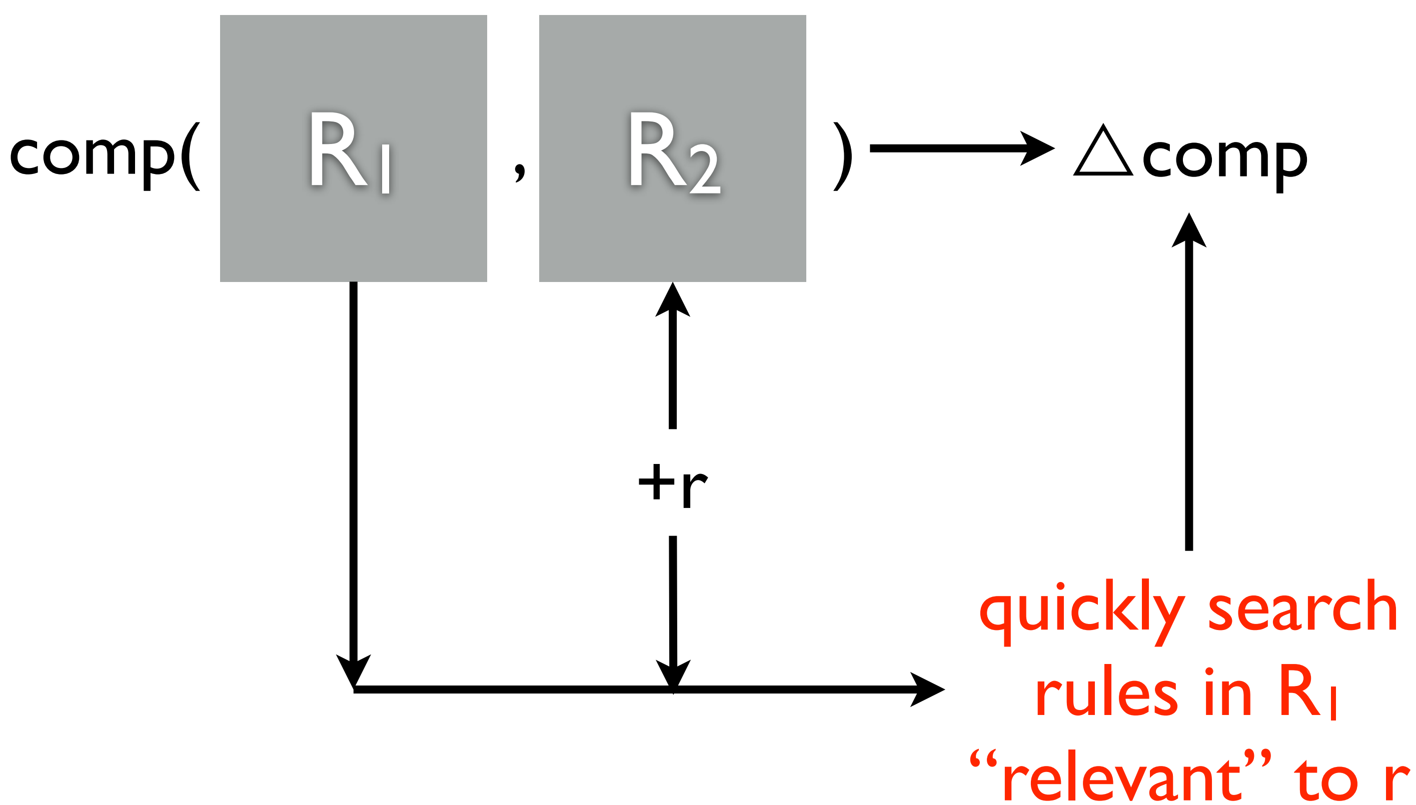


example multi-layer index

- hash table for exact match
- trie for prefix match
- list for arbitrary wildcard-match

optimization: indexing rules

reduce index size by policy correlation



only index the "correlated" info

R_1.index = R_2.index

= R_1.fields ∩ R_2.fields

evaluation

three evaluation scenarios

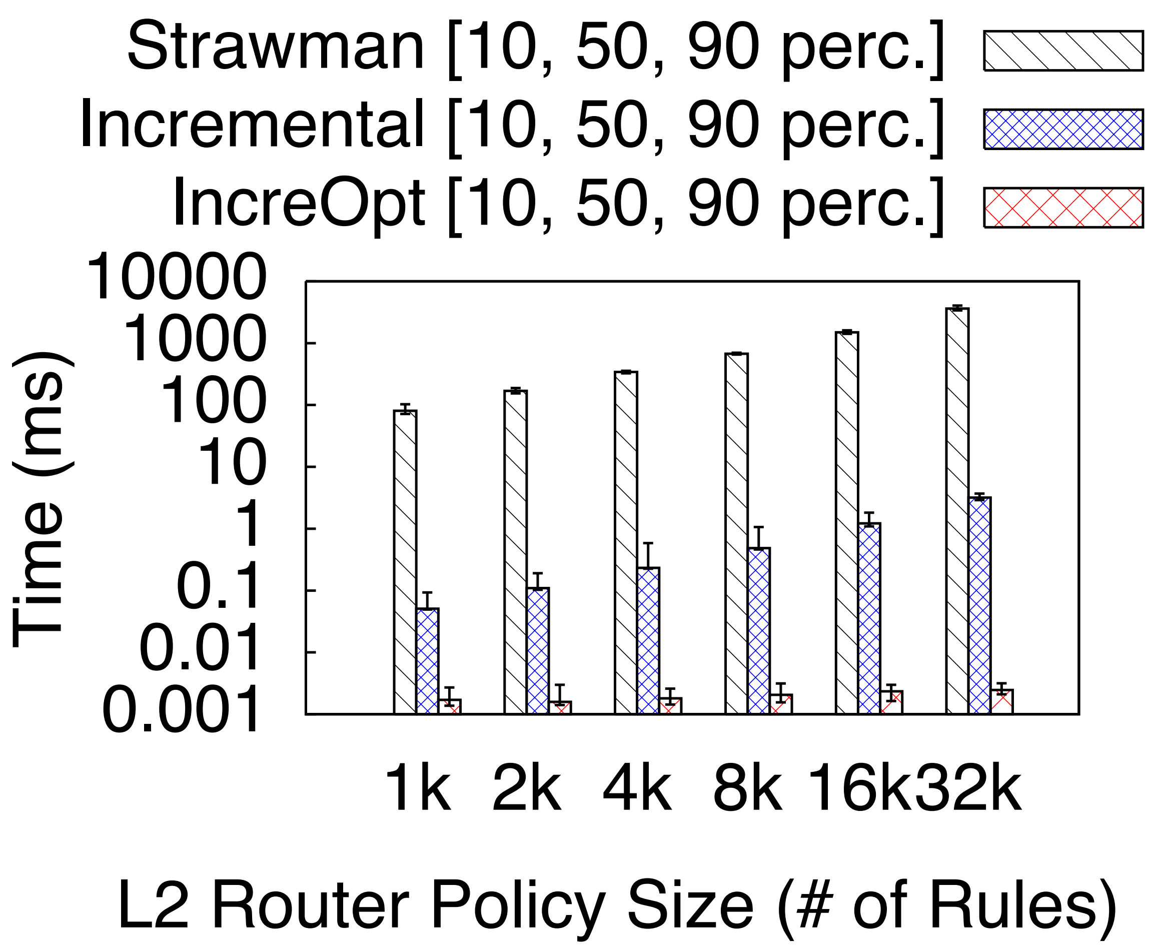
- composition (compilation, update) efficiency
- devirtualization efficiency
- stress policy size

evaluation — L2 monitor + L2 router

setup

- initialize L2 monitor policy with 1000 rules
- add 10 rules to measure overhead
- vary policy size from 1k to 32k

evaluation — L2 monitor + L2 router

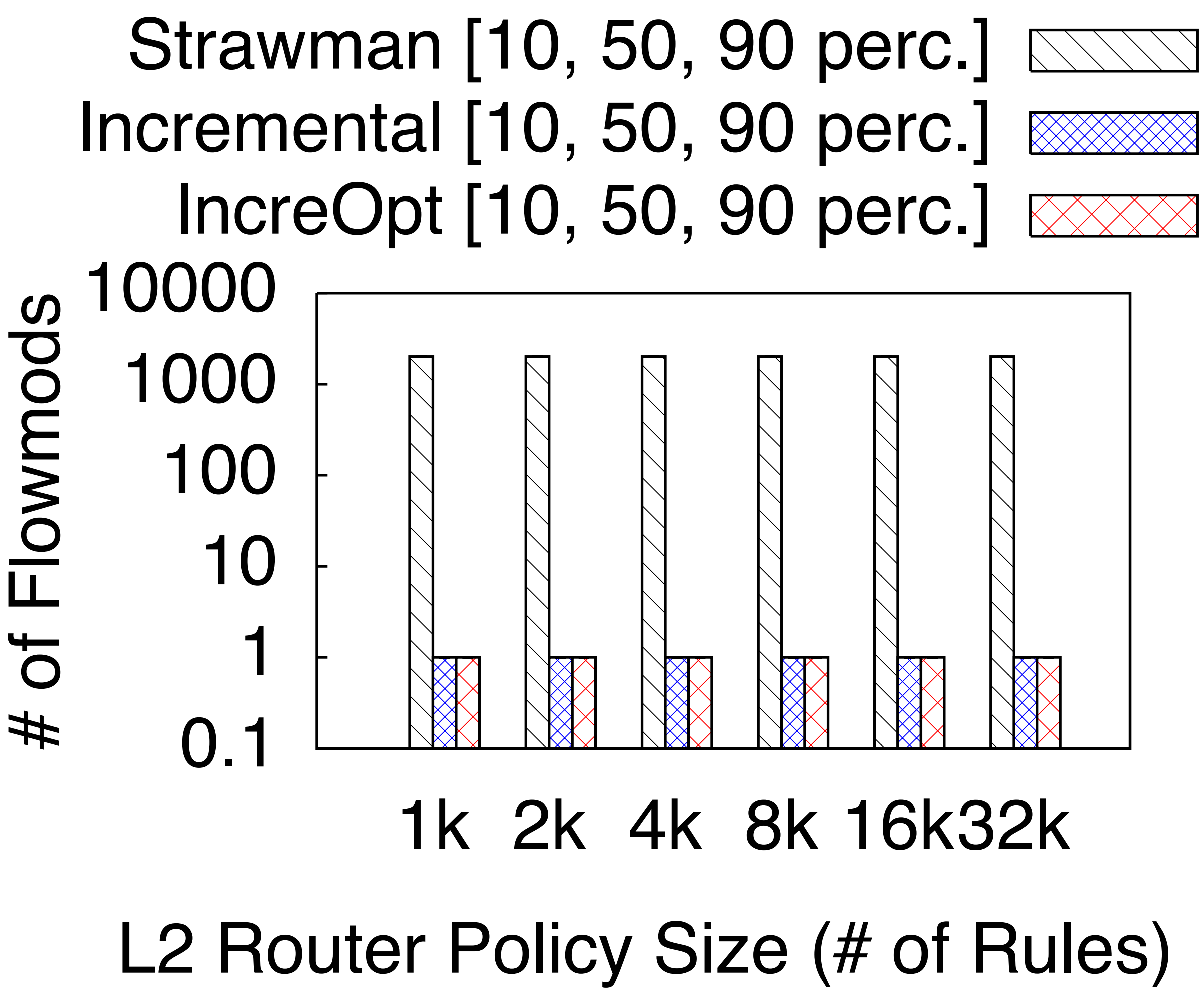


(a) Compilation Time

compilation efficiency

- smart priority assignment (Incremental, IncreOpt) achieves orders of magnitude faster compilation

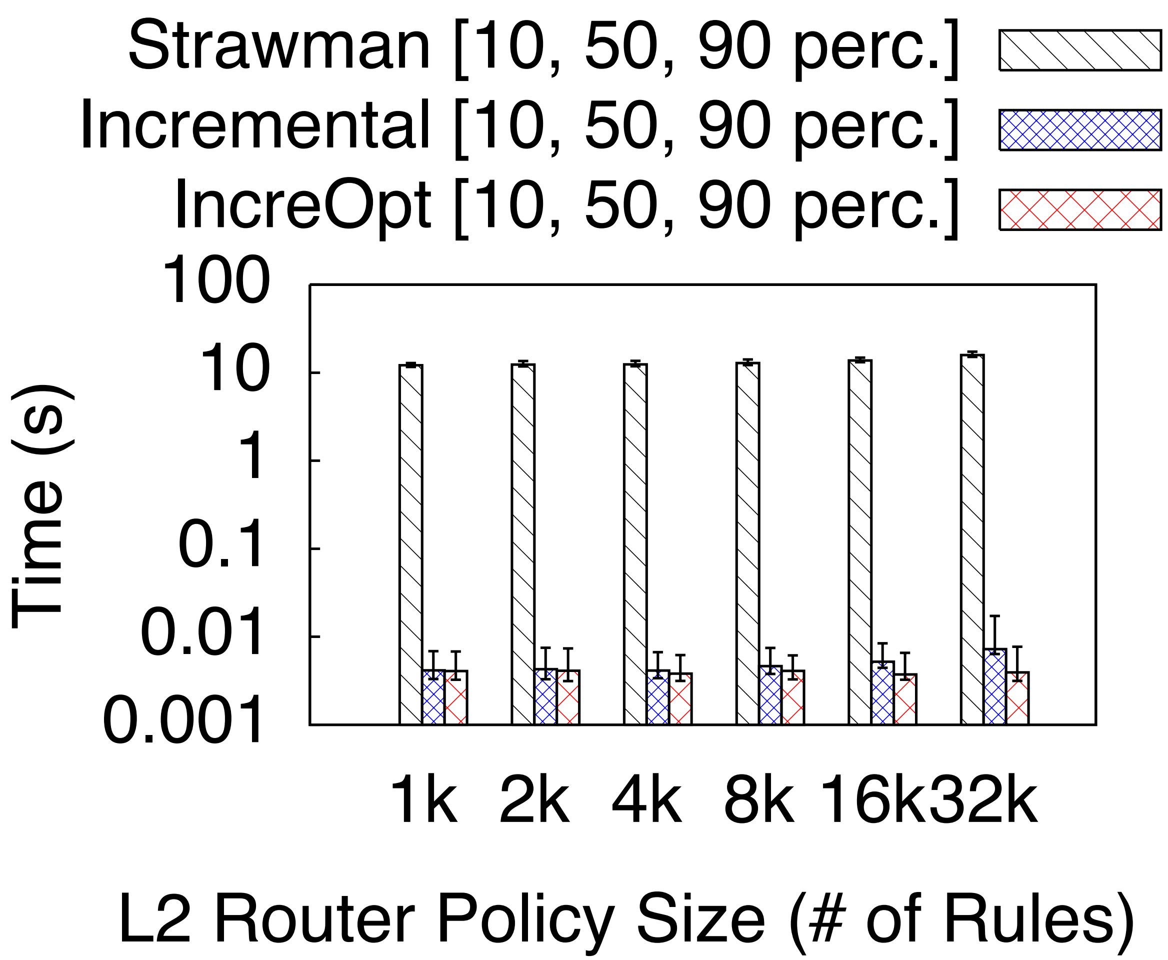
evaluation — L2 monitor + L2 router



update efficiency

- why Incremental/ IncreOpt not much difference?

evaluation — L2 monitor + L2 router



total completion time

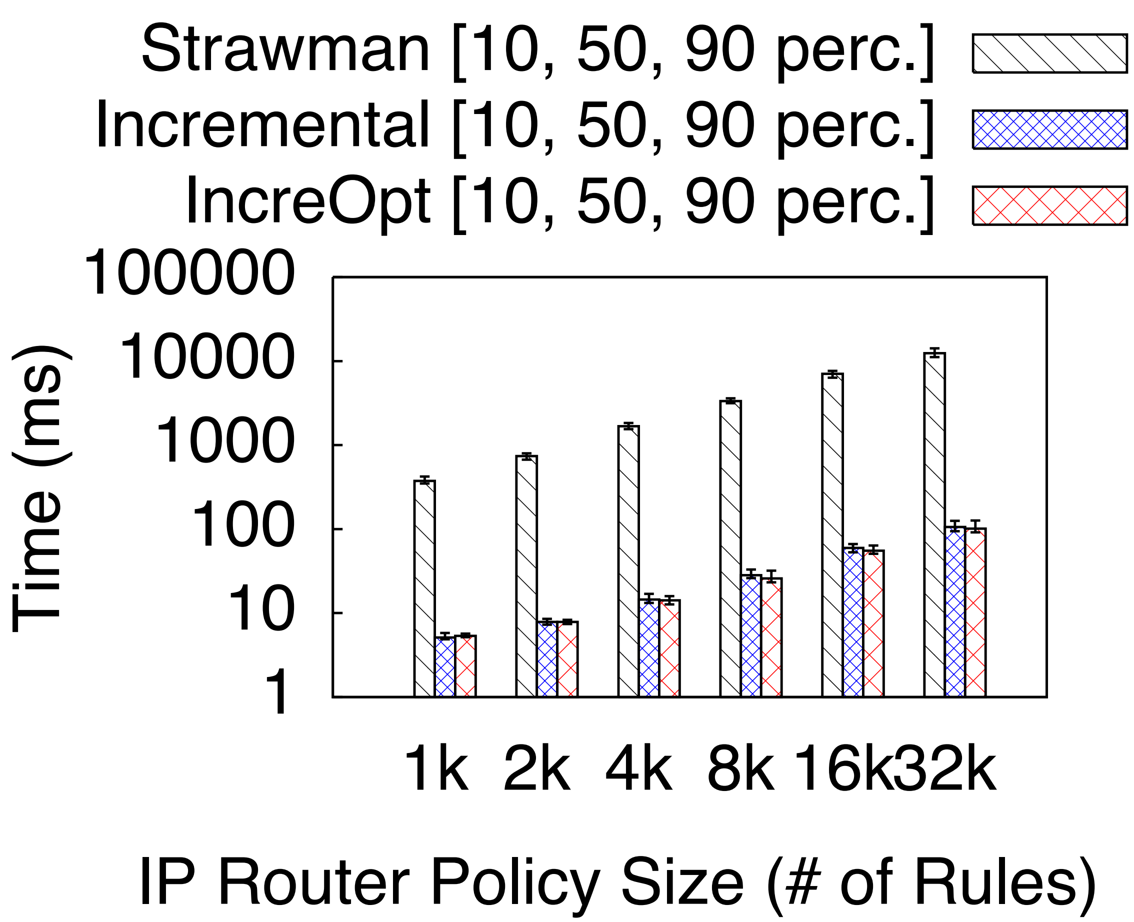
- additional OpenFlow overhead

evaluation — devirtualization

setup

- MAC learner
  - Ethernet island connecting 100 hosts
- initialize MAC learner with 1000 rules
  - vary size from 1k to 32k
- add new hosts to measure overhead

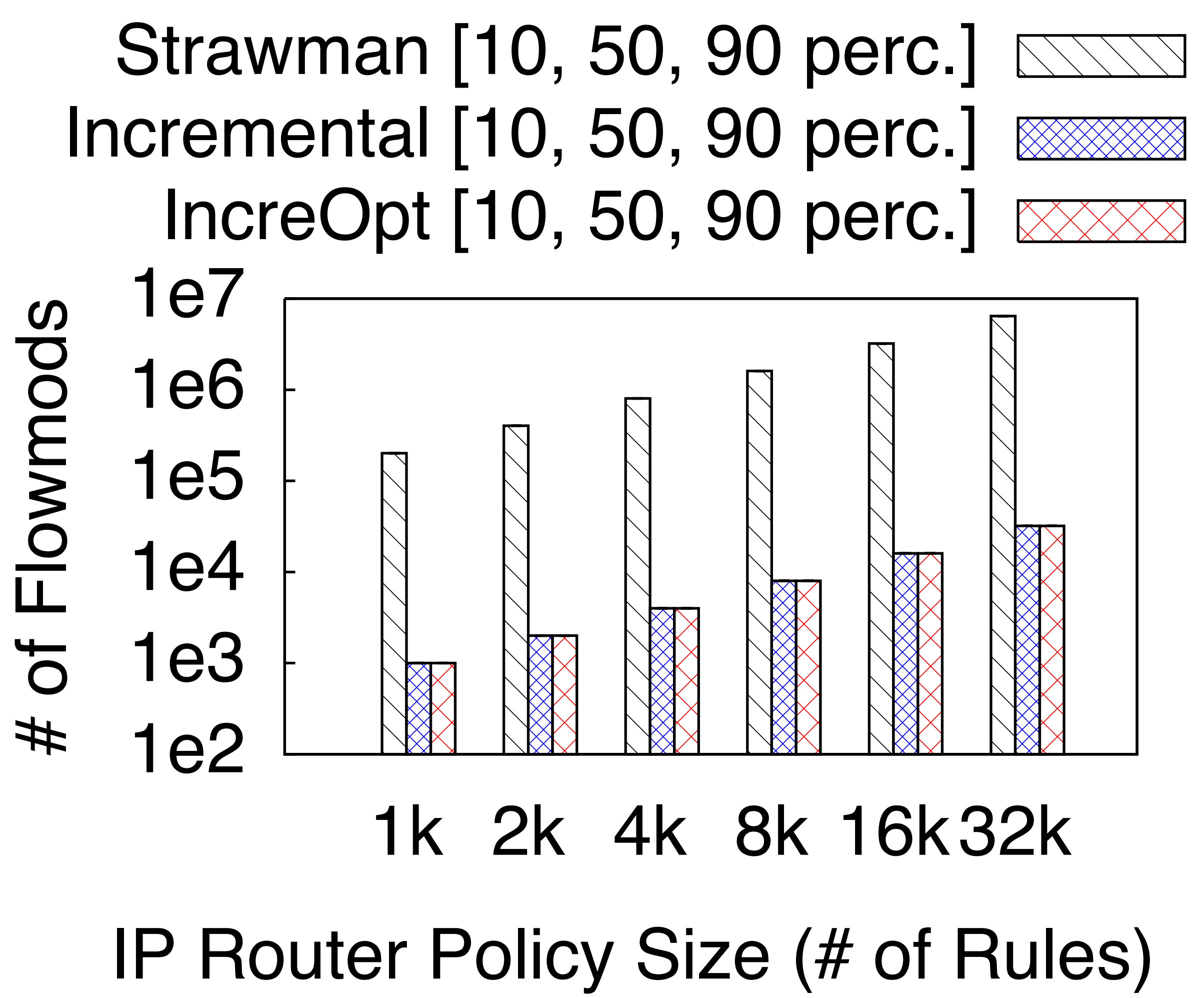
evaluation — devirtualization



(a) Compilation Time

compilation

evaluation — devirtualization



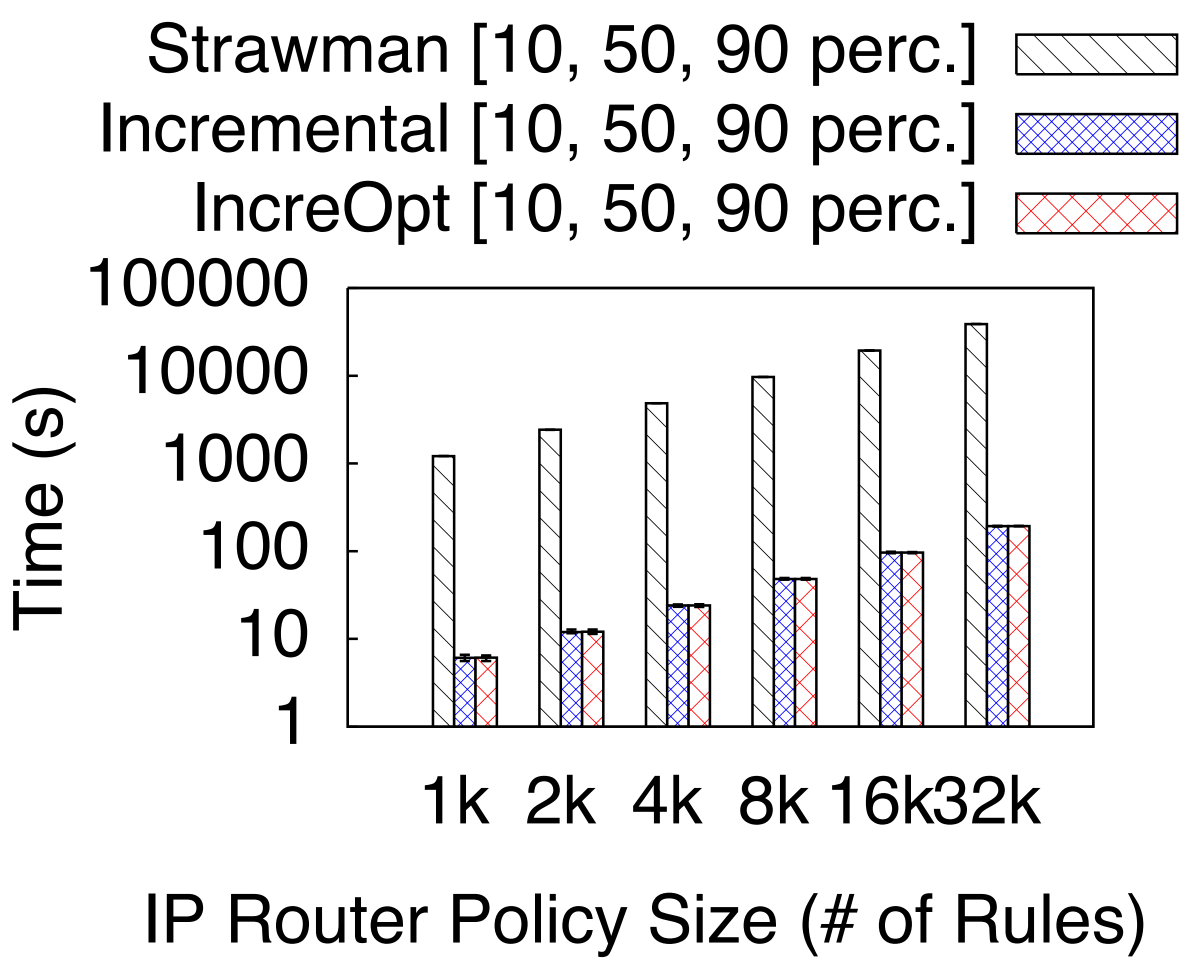
(b) Rule Update Overhead

update

- why is the absolute values of delays large?

evaluation — devirtualization

total completion time



(c) Total Update Time (Hardware)

Kinetic

dynamics

network conditions are dynamic, but current approaches to (re)configure the network are NOT

example: dynamic net config

University of Illinois

- an instructed class, 4 restricted classes
- downgrade a user's traffic to a different class based on past usage

current approach

- complex instrumentation
- "wrapper" that dynamically change low-level net config

Kinetic

goals

- capture dynamics, automatically verifies temporal properties

Kinetic language

- dynamic policy as finite state machine (FSM)
- states: distinct forwarding behavior
- transition: triggering network events

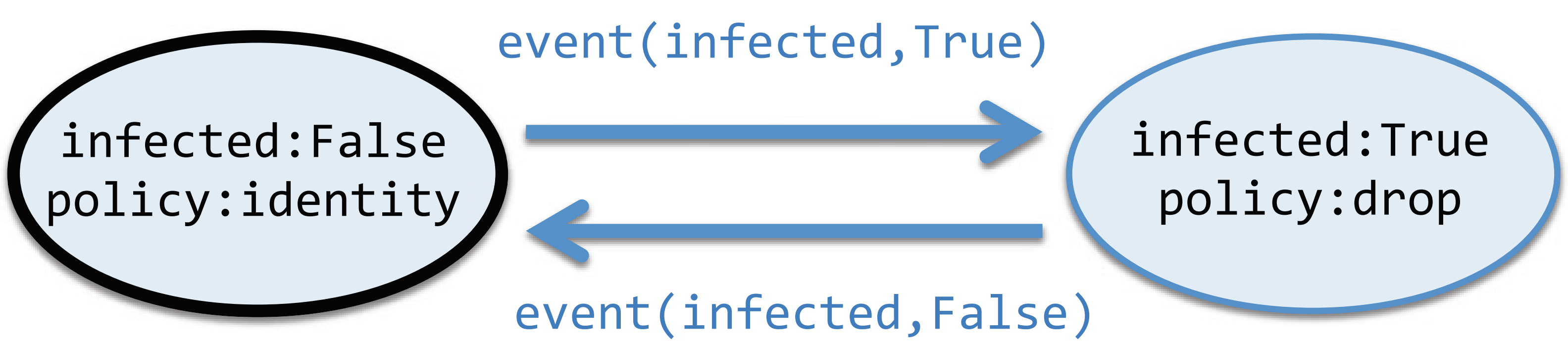
Kinetic handler listens to events

- triggers transition in a policy
- updates the data plane

dynamic policy as FSM

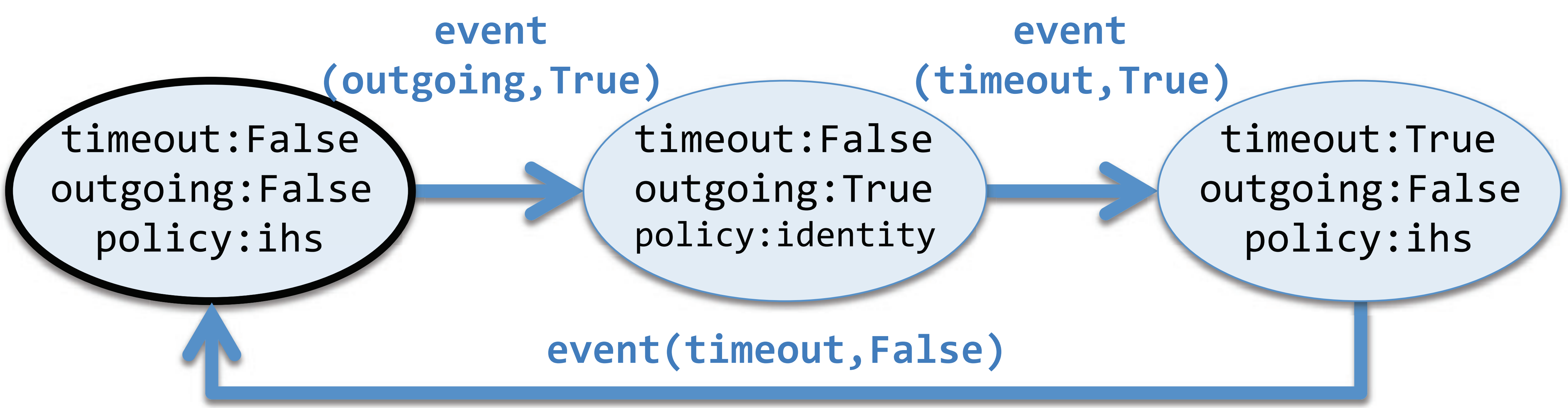
FSM specifies how a (Pyretic) policy evolves in response to events

- FSM state contains a policy
- FSM transition corresponds to net events



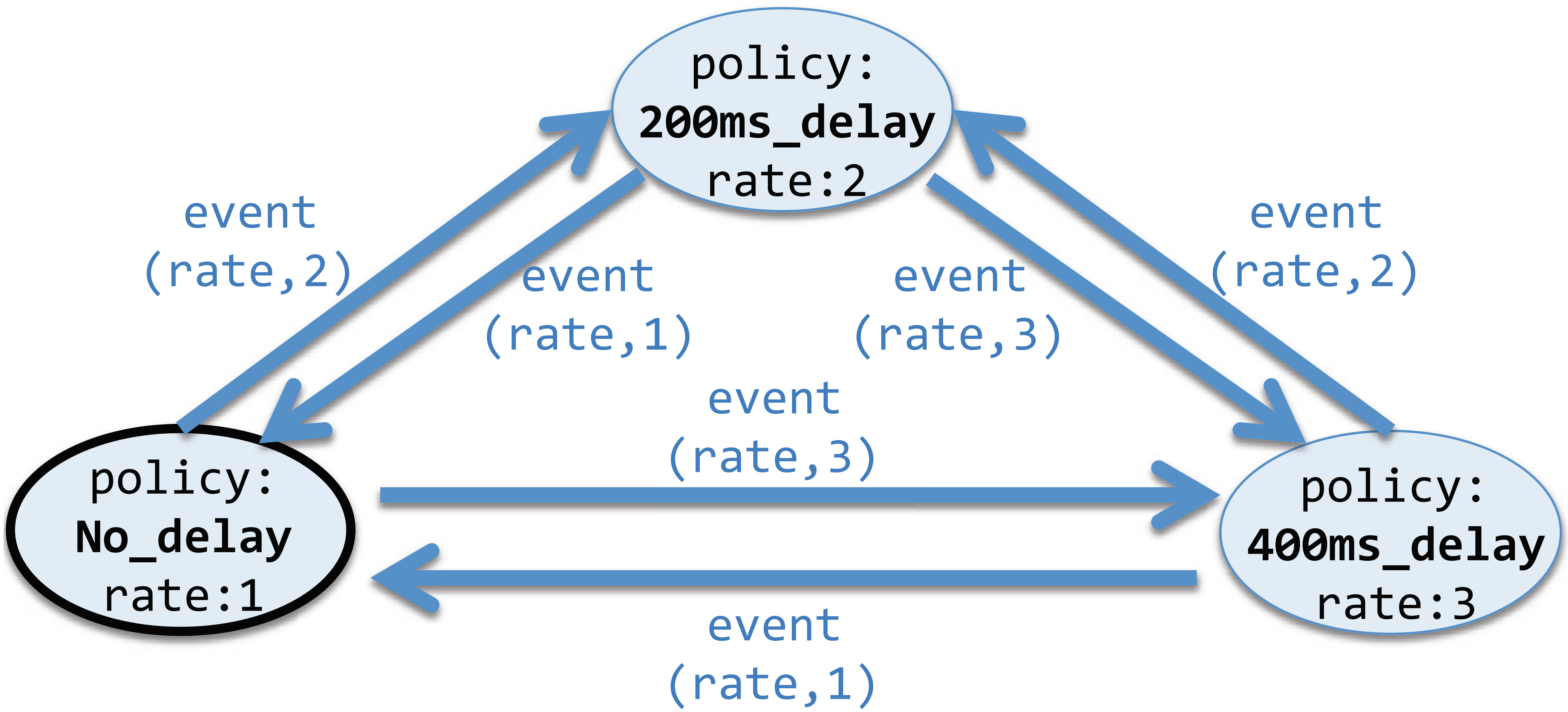
intrusion detection *FSM*

dynamic policy as FSM



stateful firewall *FSM*

dynamic policy as FSM

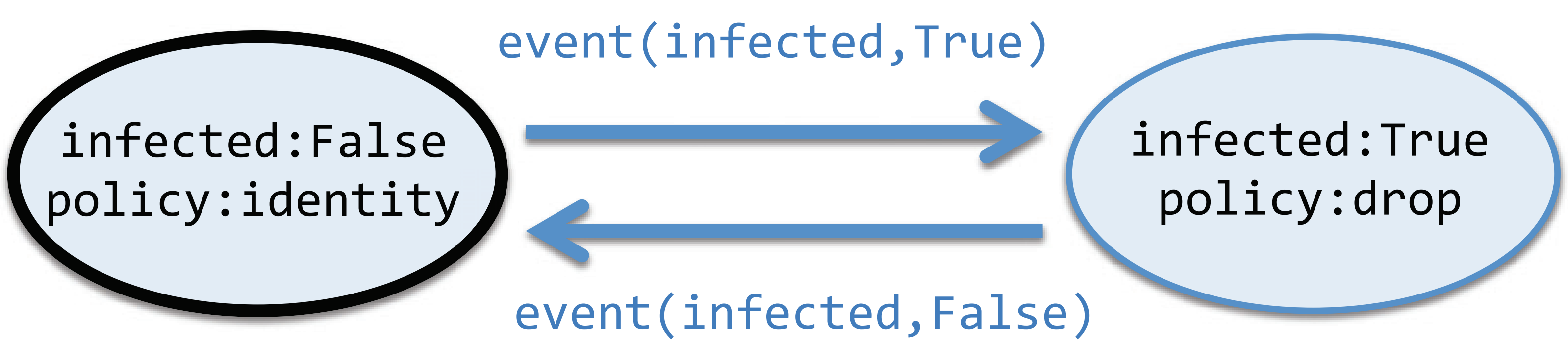


data usage based rate limiter

the state explosion challenge

FSM specifies how a (Pyretic) policy evolves in response to events

- FSM state contains a policy
- FSM transition corresponds to net events



N hosts $\longrightarrow$ 2^N states

the state explosion challenge

dynamic policy defined over a state space *exponential* in the number of

- hosts, flows, …
- N hosts $\longrightarrow$ 2^N FSM states

a monolithic FSM

- break into N smaller FMSs, each with a_i states
- $\prod a_i$ states

Kinetic — technical contribution

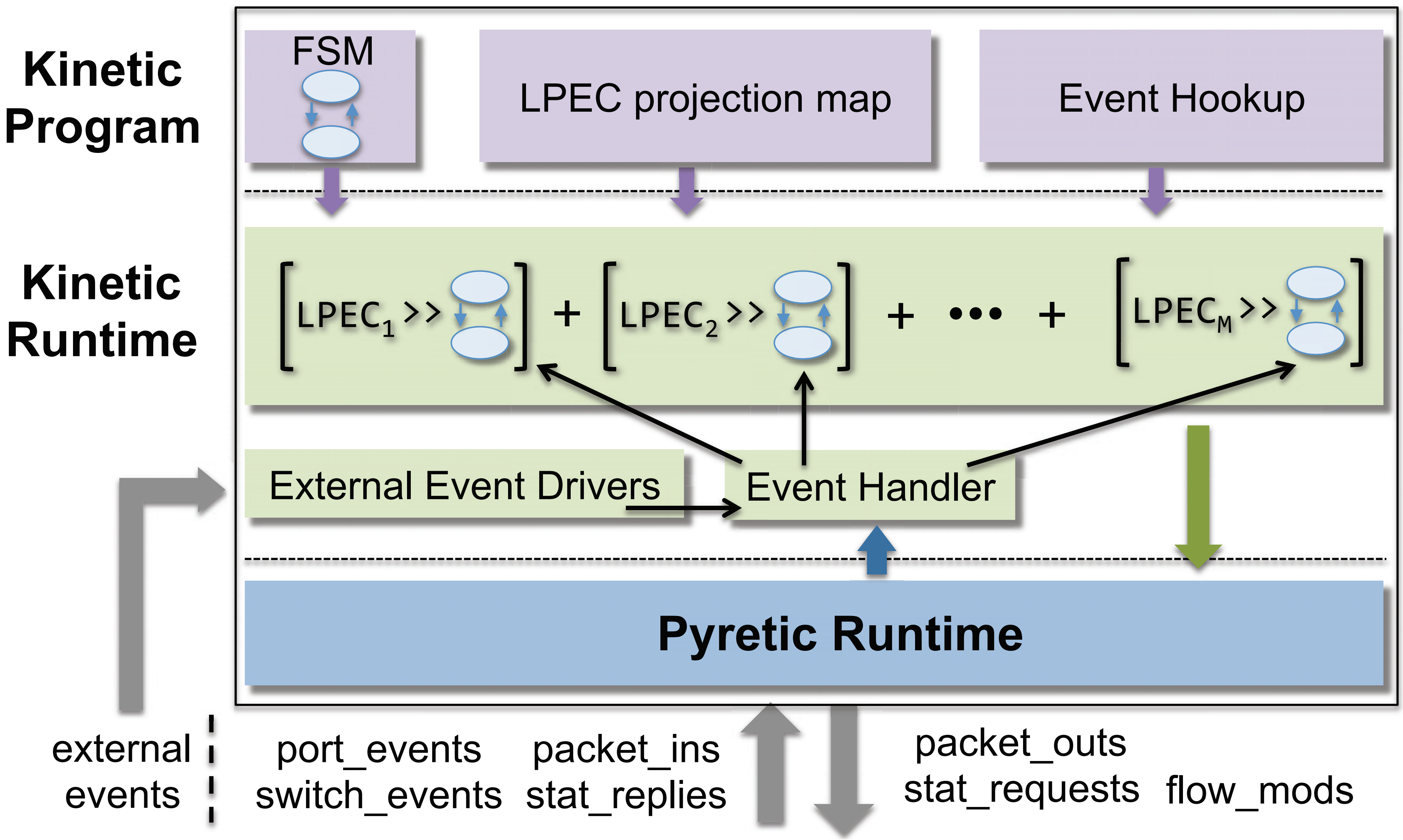
introduce located packet equivalence class (LPEC)

- divide the state space into isolated FSMs

use Pyretic composition

- express large FSMs as smaller ones
- prevent FSM state explosion

Kinetic architecture

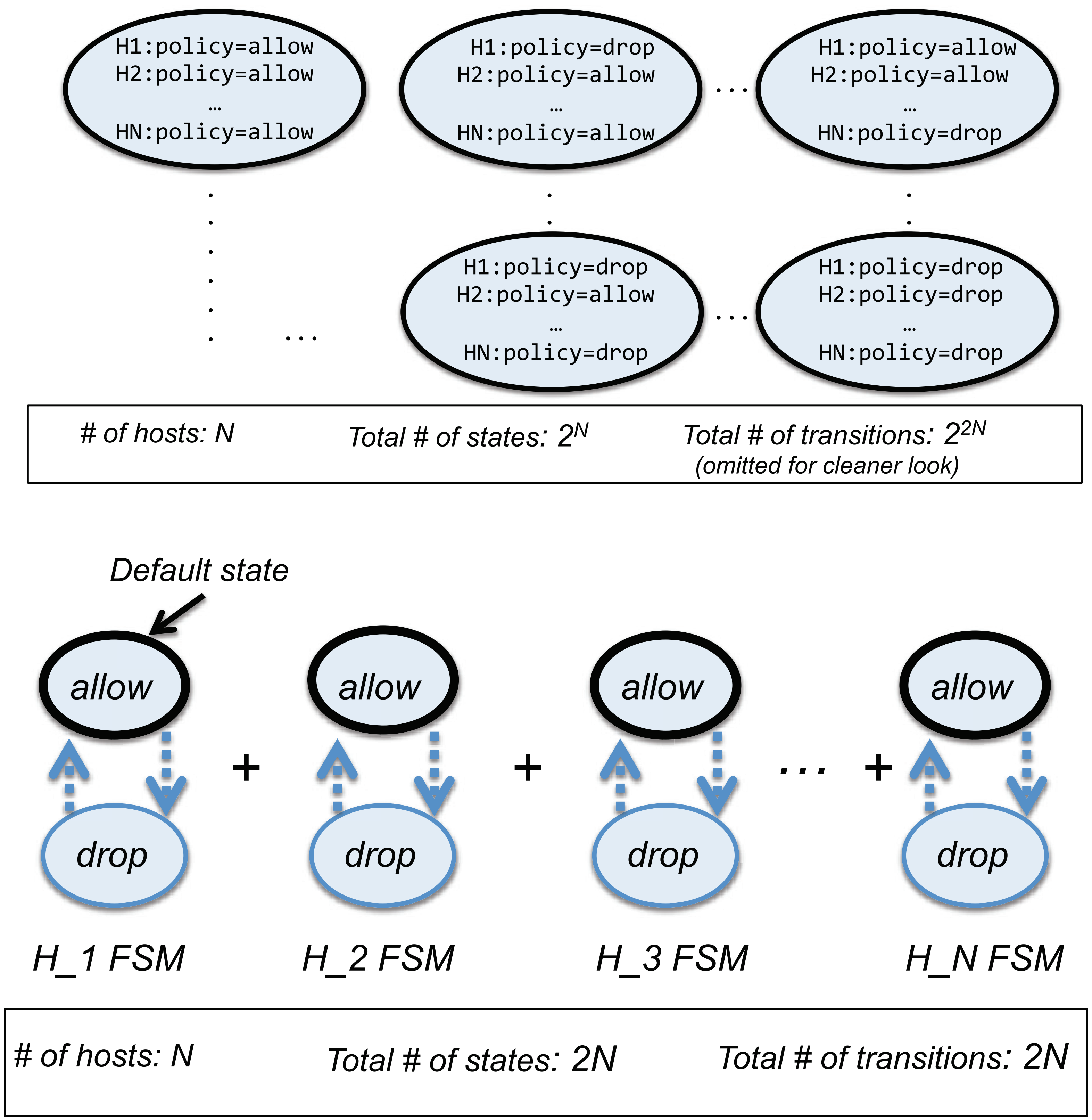


located packet equivalence class

2^N states

N LPEC

N LPECs

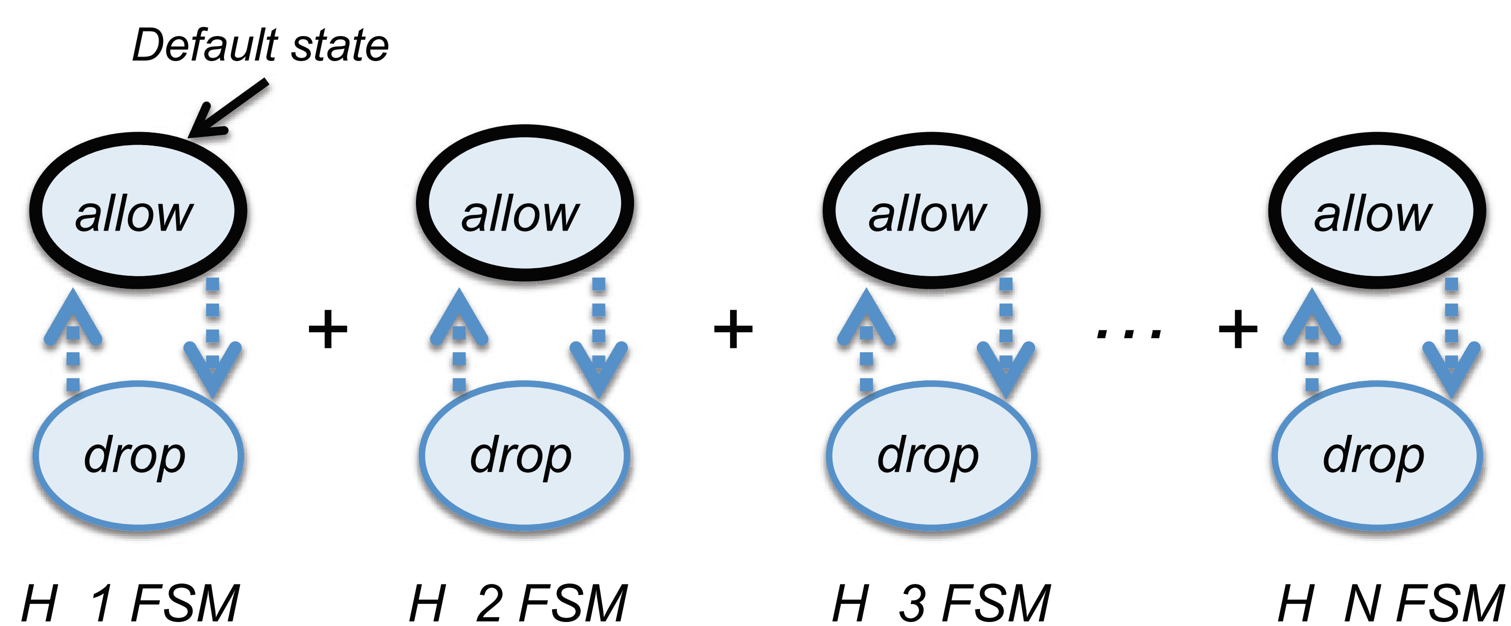


located packet equivalence class

LPEC: packets always in the same FSM state

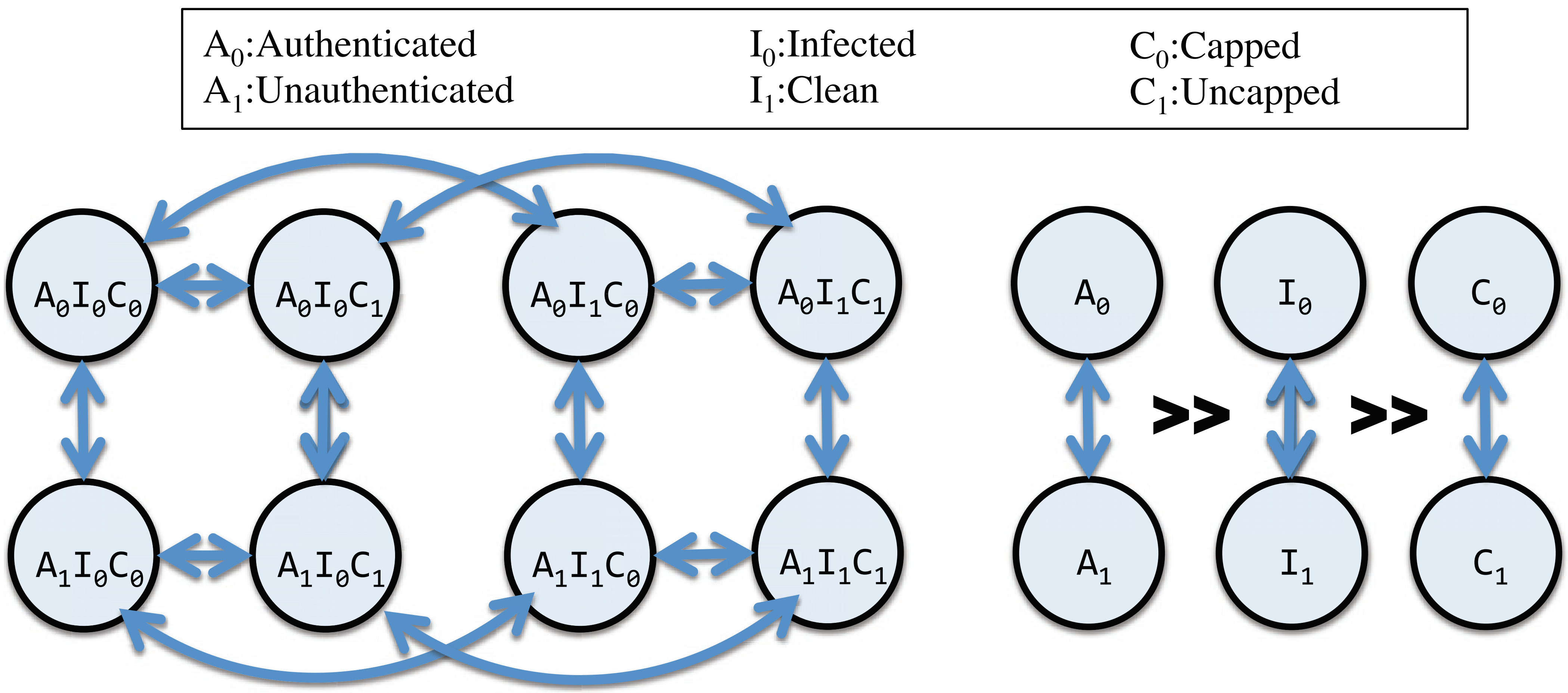
- dynamics for each LPEC defined by an isolated FSM
- for each LPEC:
  - events $\longrightarrow$ FSM transition $\longrightarrow$ Pyretic recompilation $\longrightarrow$ switch update

N isolated FSMs



# of hosts: N	*Total # of states: 2N*	*Total # of transitions: 2N*

FSM (sequential) composition

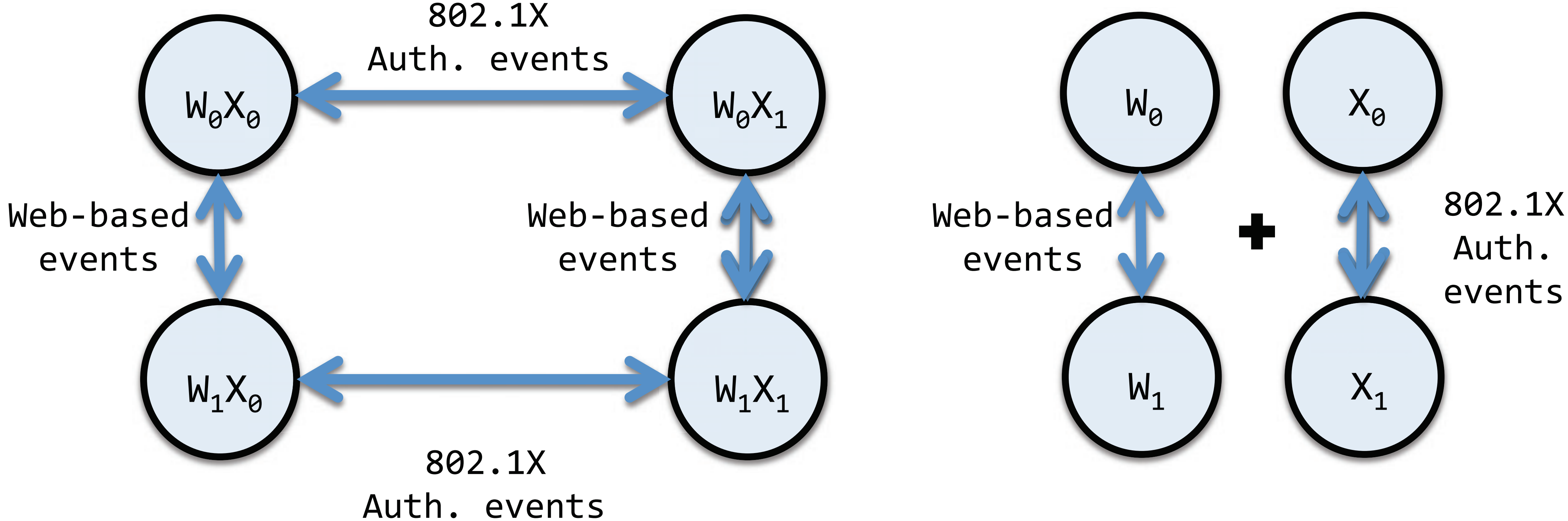


$\prod^{N}_{i=1} a_i$

$\sum^{N}_{i=1} a_i$

FSM (parallel) composition

W_0:Web-Authenticated
W_1:Web-Unauthenticated
X_0: 802.1X-Authenticated
X_1: 802.1X-Unauthenticated



$\prod_{i=1}^{N} a_i$

$\sum_{i=1}^{N} a_i$