Composing Software-Defined Networks

Chris Monsanto*, **Joshua Reich***
Nate Foster^, Jen Rexford*, David Walker*
www.frenetic-lang.org/pyretic

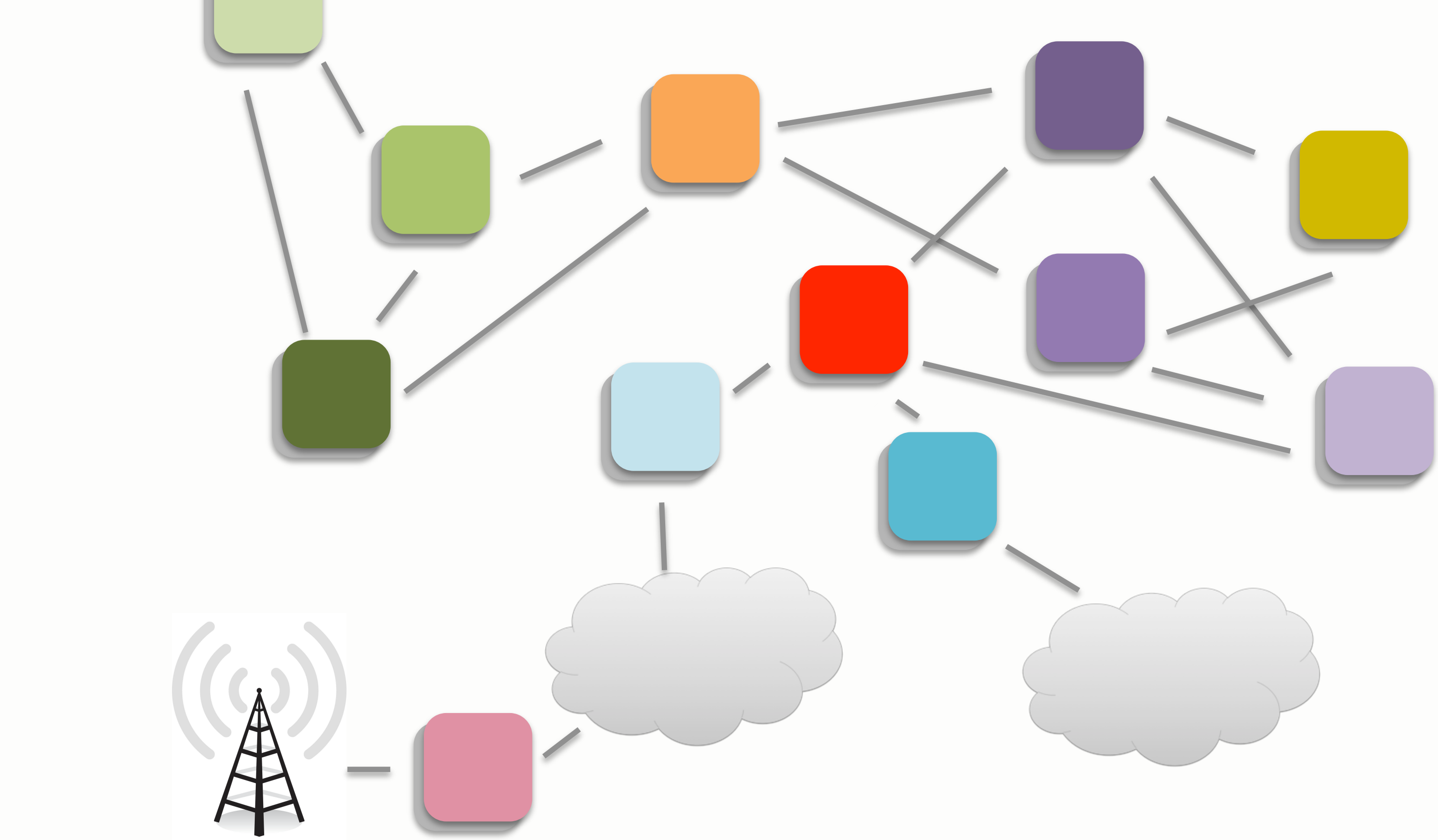
Princeton*   Cornell^

Software Defined Networks (SDN)

Enable network innovation by decoupling

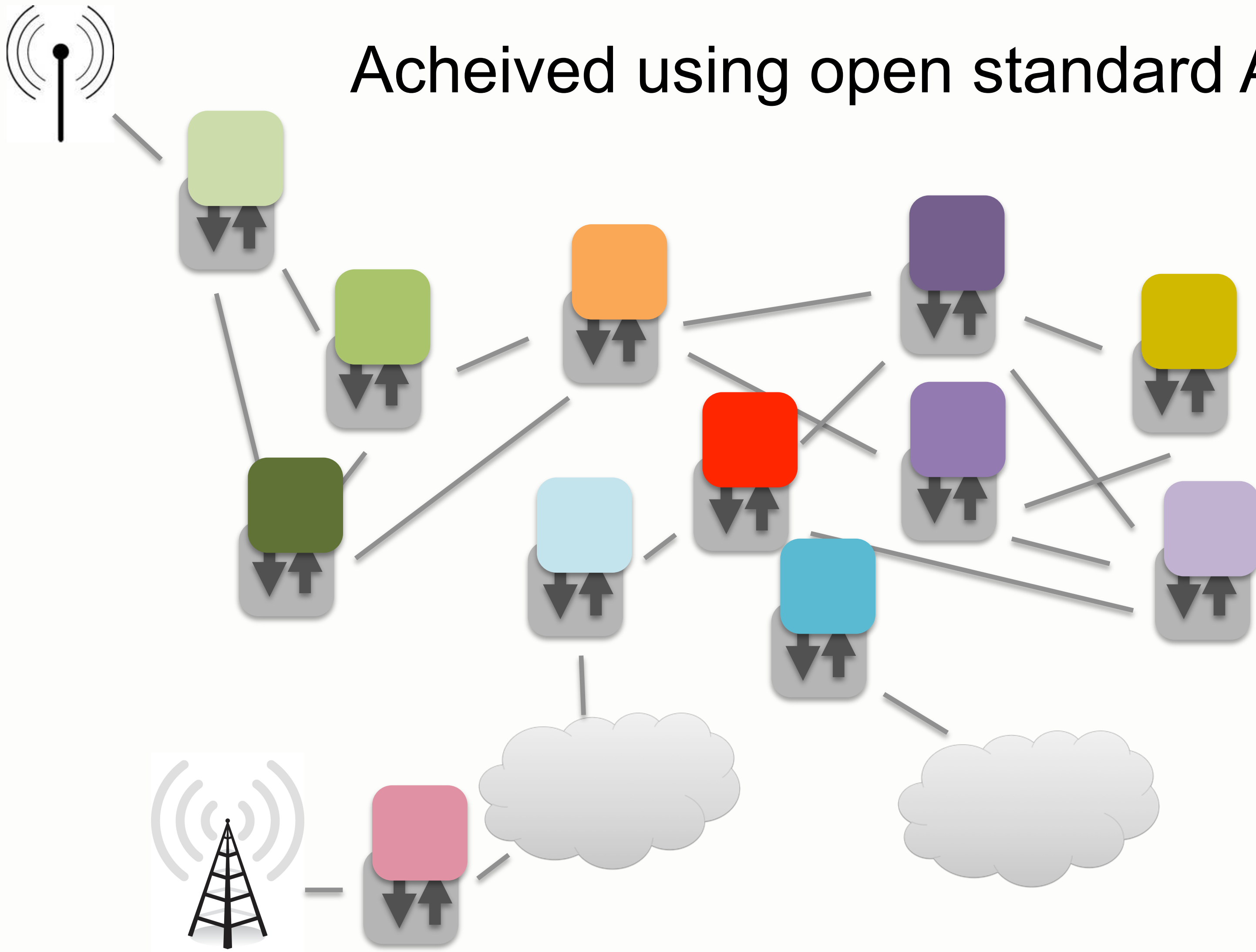
control plane: determines rules

data plane: applies rules to packets

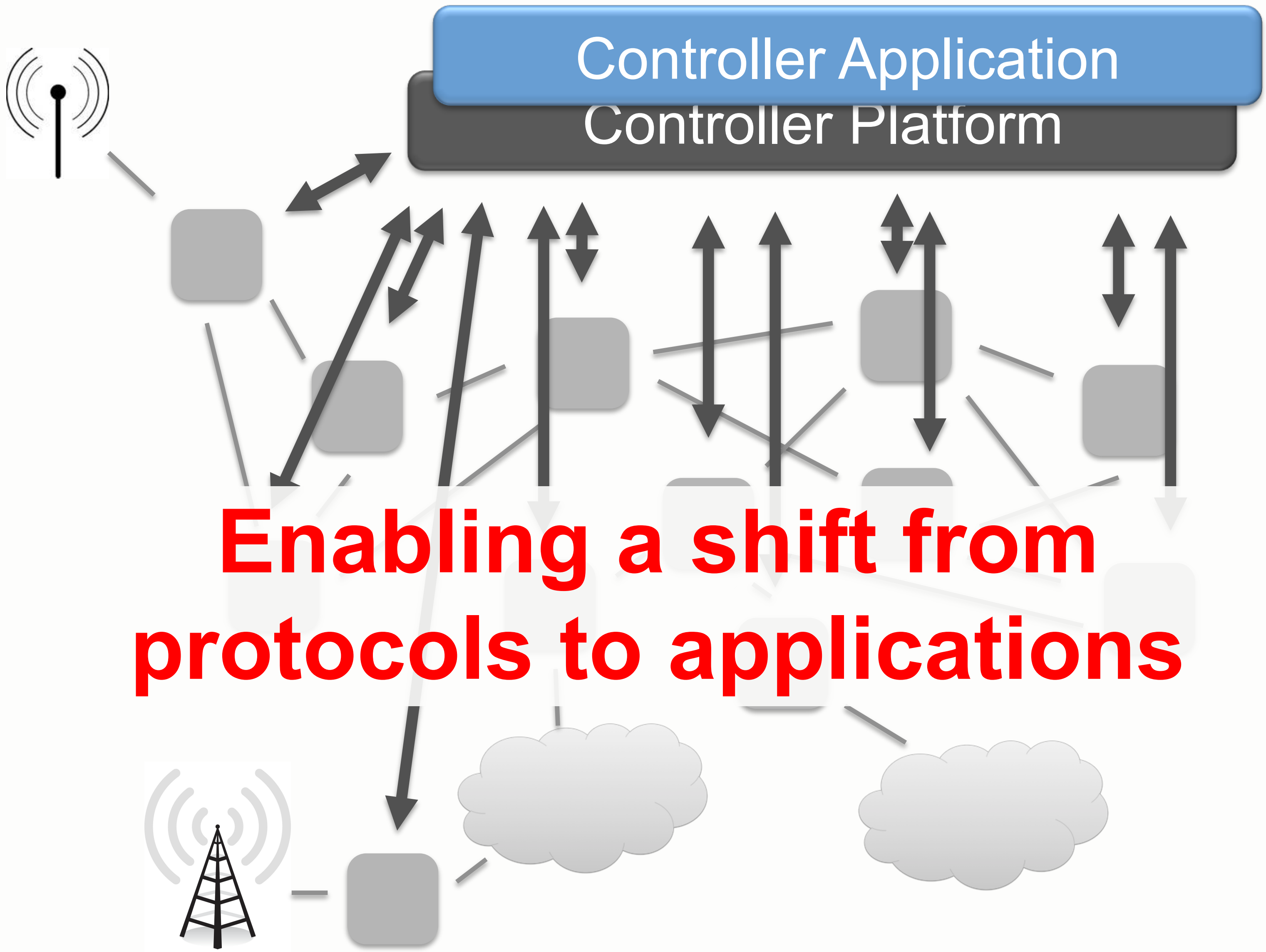


Software Defined Networks (SDN)

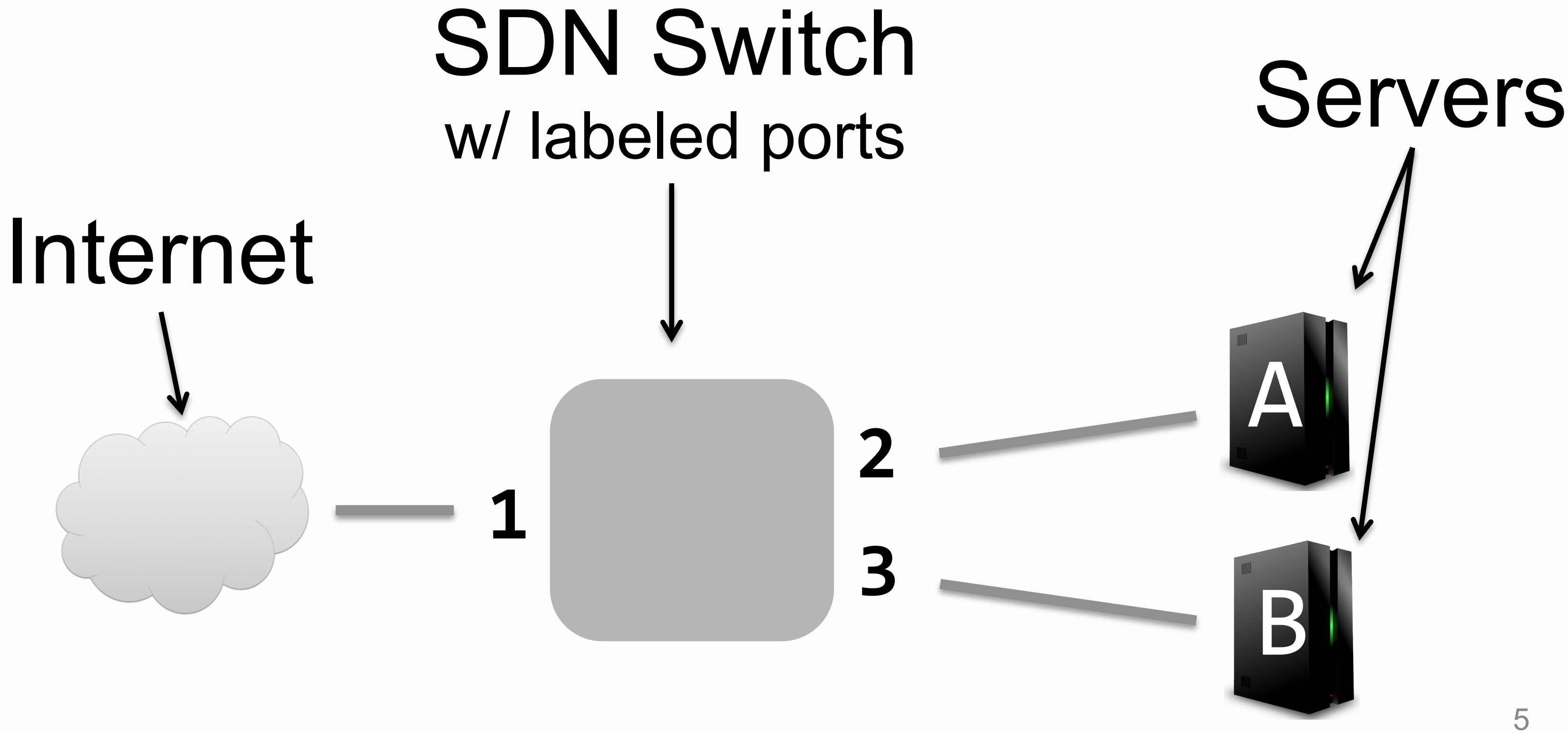
Acheived using open standard API



Software Defined Networks (SDN)



Running Example Network



Programming in OpenFlow

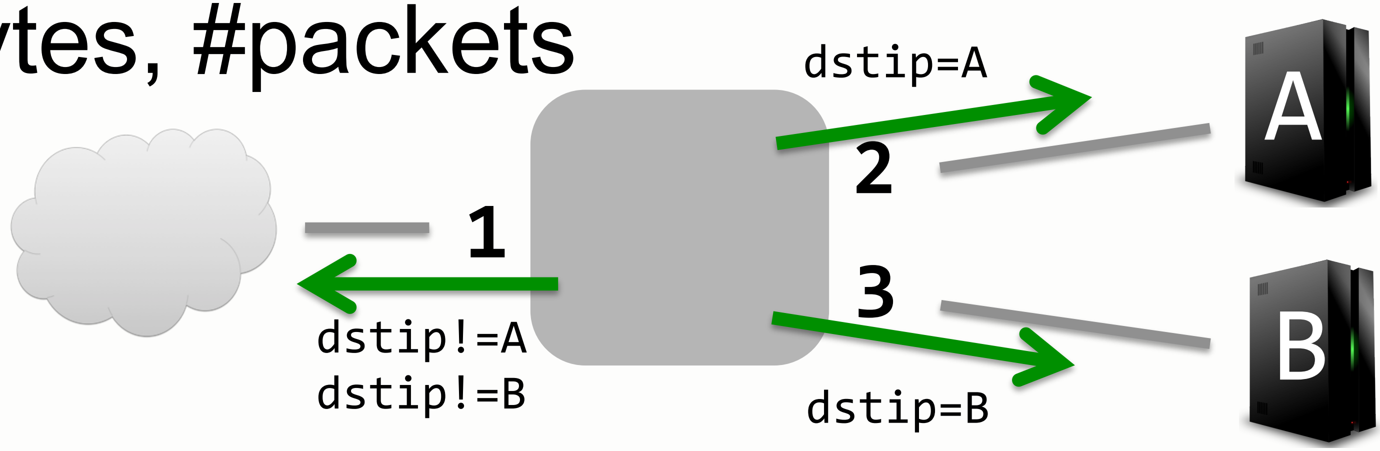
Priority

```
2:match(dstip=A)[fwd(2)]
1:match(*      )[fwd(1)]
2:match(dstip=B)[fwd(3)]
```

Pattern　　Action

***Route*: IP/fwd**

Counters for each rule
- #bytes, #packets



One API, Many Uses

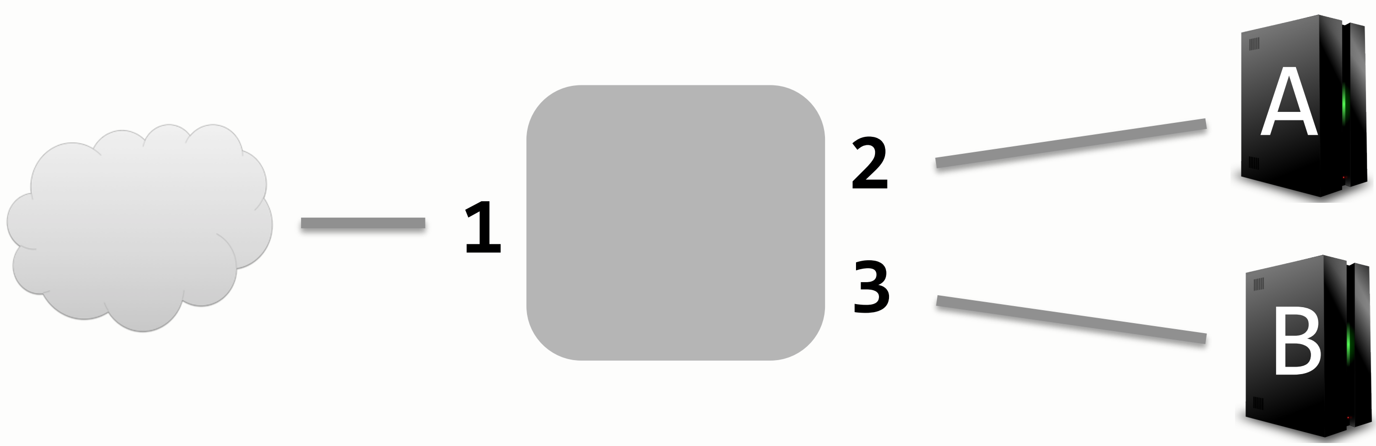
Priority Ordered

```
match(dstmac=A)[fwd(2)]
match(dstmac=B)[fwd(3)]
match(*        )[fwd(1)]
```

Pattern

Action

***Switch*: MAC/fwd**



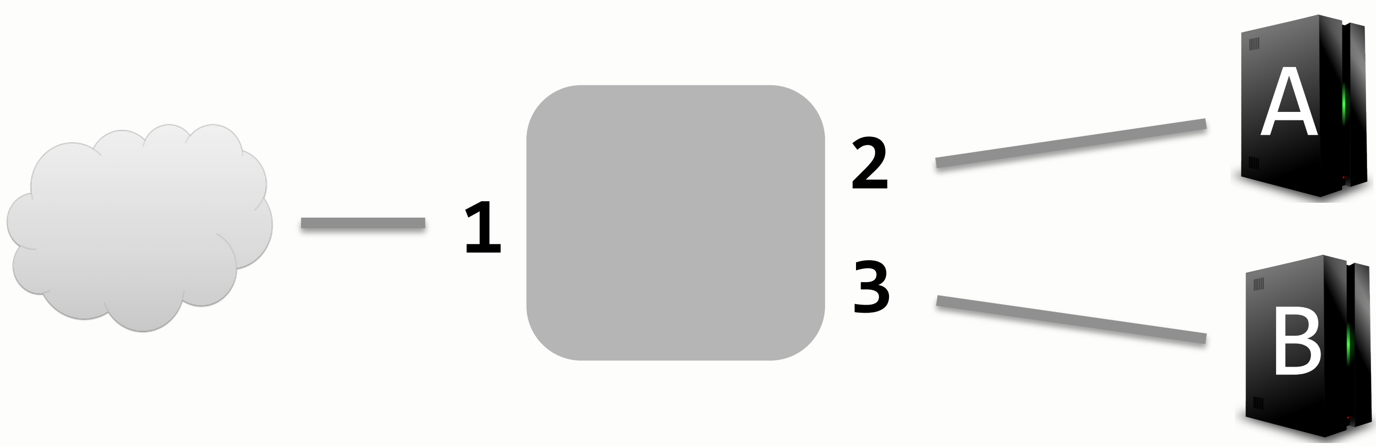
One API, Many Uses

```
match(srcip=0*,dstip=P)[mod(dstip=A)]
match(srcip=1*,dstip=P)[mod(dstip=B)]
```

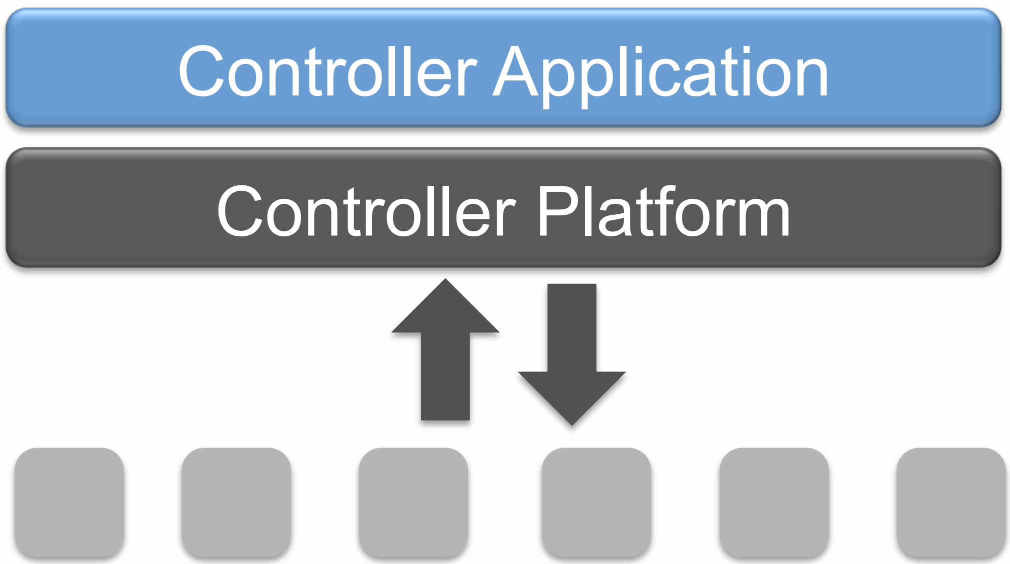
Pattern

Action

***Load Balancer*: IP/mod**



But Only Half of the Story



Hardware Interface

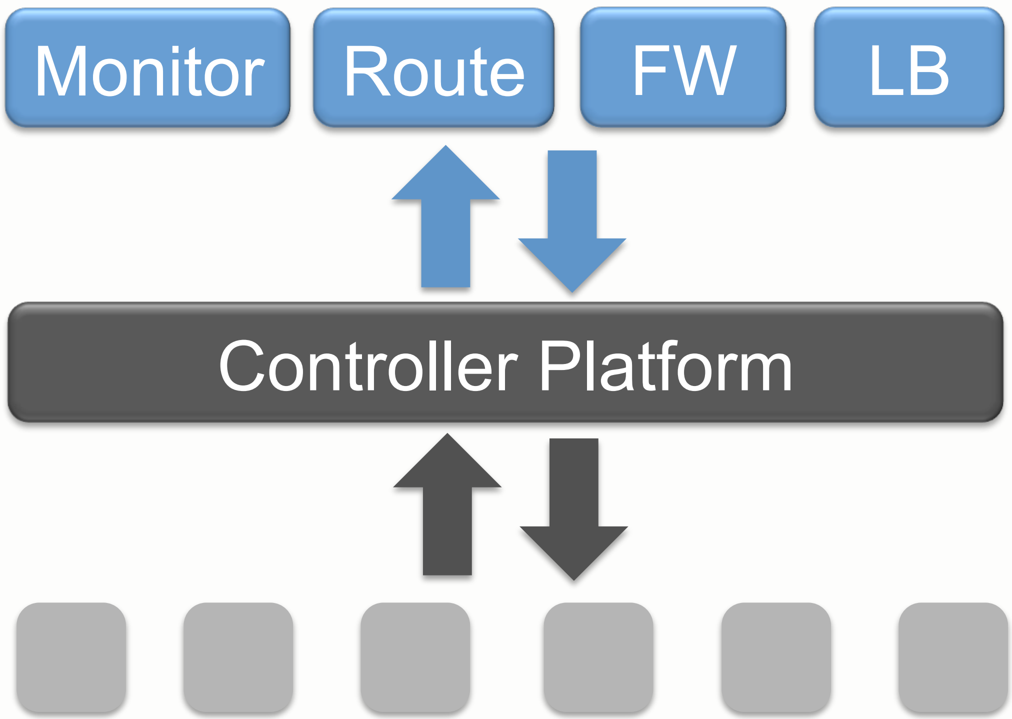
OpenFlow (assembly)

Hardware Flow-Table Rules (Machine Language)

But Only Half of the Story

Modular & Intuitive

Programmer Interface



?

Hardware Interface

OpenFlow (assembly)

Hardware Flow-Table Rules (Machine Language)

OpenFlow Isn't Modular

Balance then Route

```
match(srcip=0*,dstip=P)[mod(dstip=A)]
match(srcip=1*,dstip=P)[mod(dstip=B)]
```

```
match(dstip=A)[fwd(2)]
match(dstip=B)[fwd(3)]
match(*      )[fwd(1)]
```

Combined Rules?
(only one match)

```
match(srcip=0*,dstip=P)[mod(dstip=A)]
match(srcip=1*,dstip=P)[mod(dstip=B)]
match(         dstip=A)[fwd(2)      ]
match(         dstip=B)[fwd(3)      ]
match(*               )[fwd(1)      ]
```

Balances w/o Forwarding!

OpenFlow Isn't Modular

Balance then Route

```
match(srcip=0*,dstip=P)[mod(dstip=A)]
match(srcip=1*,dstip=P)[mod(dstip=B)]
```

```
match(dstip=A)[fwd(2)]
match(dstip=B)[fwd(3)]
match(*      )[fwd(1)]
```

Combined Rules?
(only one match)

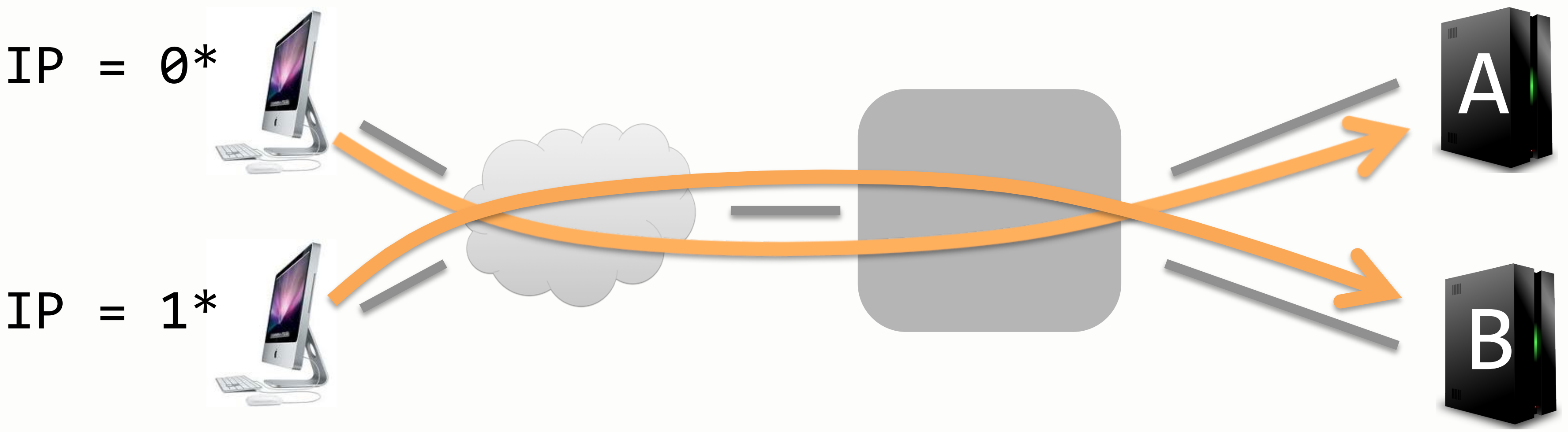
```
match(         dstip=A)[fwd(2)      ]
match(         dstip=B)[fwd(3)      ]
match(*               )[fwd(1)      ]
match(srcip=0*,dstip=P)[mod(dstip=A)]
match(srcip=1*,dstip=P)[mod(dstip=B)]
```

Forwards w/o Balancing!

Pyretic (Contributions)

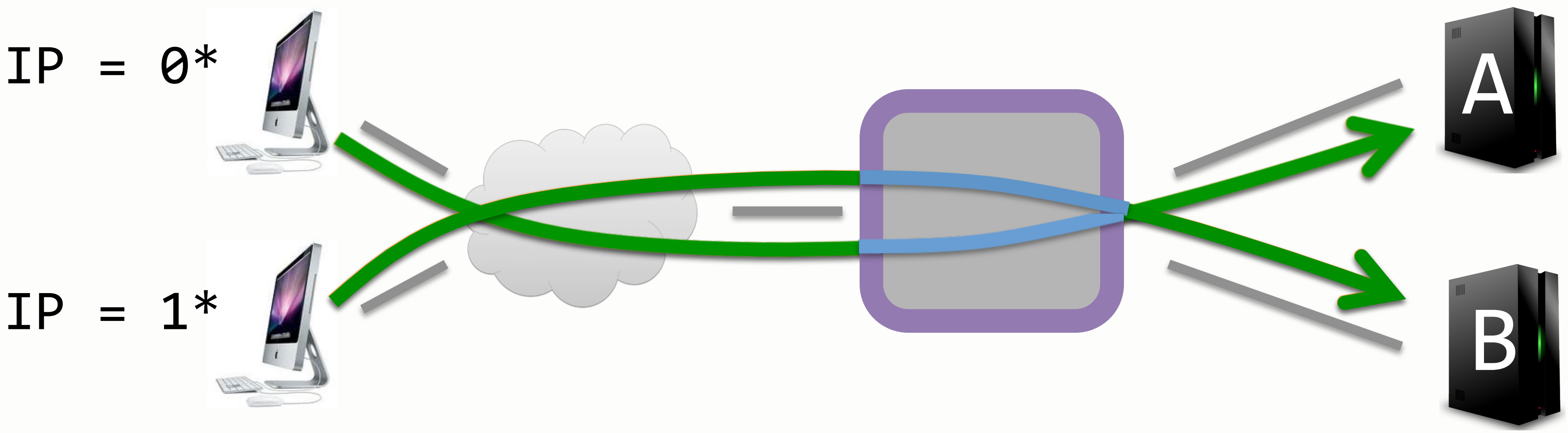
Abstracts	Providing	Supporting
Policy	Compositional Operators	Functional Composition
Network	Layered Abstract Topologies	Topological Decomposition
Packet	Extensible Headers	Policy & Network Abstractions

Compositional Operators:
A Monitoring Load Balancer



- Traffic to P re-addressed and fowarded to either A or B, based on source
- Counted if from source X

Compositional Operators:
A Monitoring Load Balancer



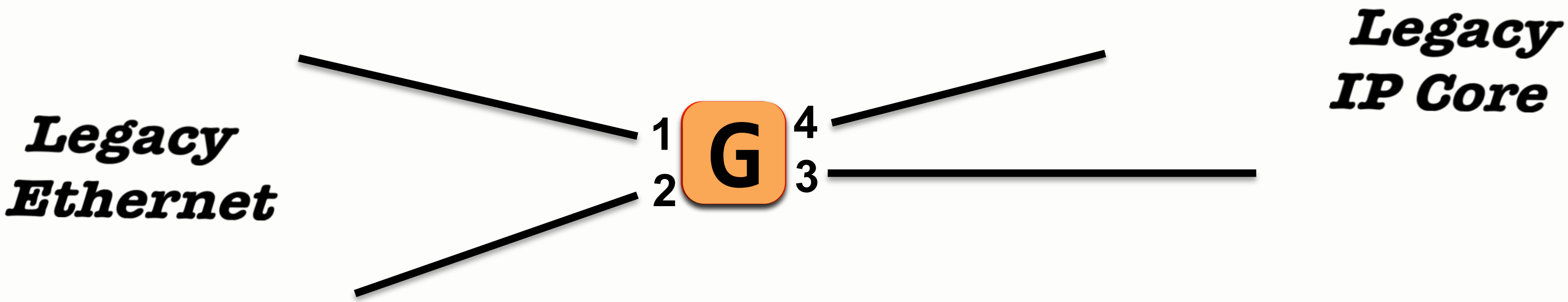
Module 1: **Balance** then Module 2: **Route**, and Module 3: **Monitor**

Rewrite `dstip` P to
A, if `srcip=0*`
B, if `srcip=1*`

Based on `dstip`

count if `srcip=X`

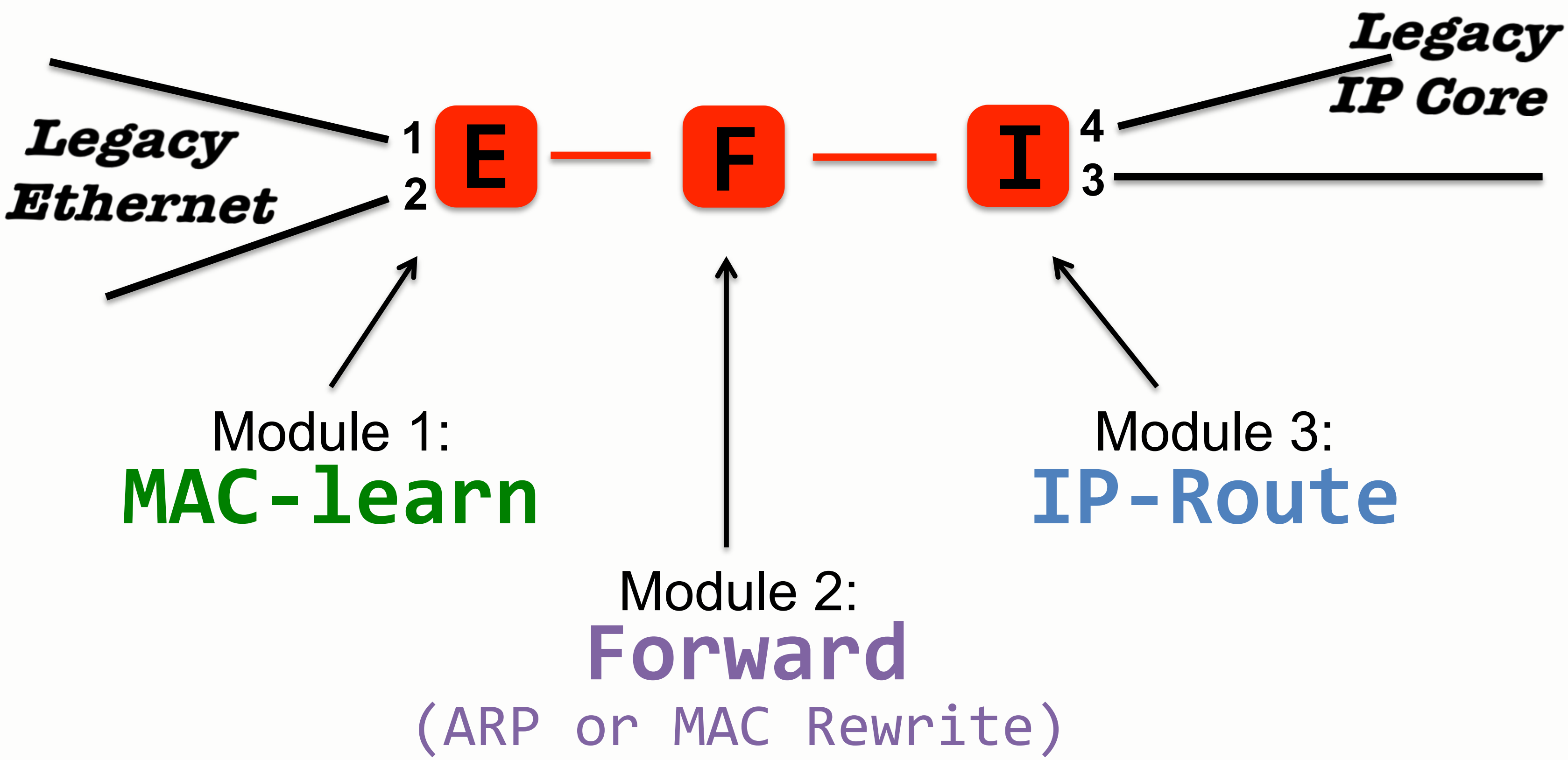
Topology Abstraction:
A Legacy Gateway Replacement



Gateway acts like:

- Legacy router
- Legacy switch
- ARP responder
- Hybrid MAC-rewriter, legacy router/switch

Topology Abstraction:
A Legacy Gateway Replacement



Pyretic's Design

- Monitoring Load Balancer
  - Encode policies as functions
  - Compositional operators
  - Queries as forwarding policies
- MAC-Learning Module
  - Dynamic Policies
- "One Big Switch" Topology Abstraction
  - Extensible packet model

`Pyretic` Drop Policy

Goal: Drop packets (i.e., `OpenFlow` drop)

Write: `drop`

Means: `eval(drop,p) = {}`

evaluate given policy on packet results in

`Pyretic` Forward Policy

Goal: Forward packets out port a

Write: `fwd(a)`

Means: `eval(fwd(a),p) = {p[outport:=a]}`

located packet w/ fields for

- `switch`
- `inport`
- `outport`

One `Pyretic` Policy For Each `OpenFlow` Action

`drop`	`fwd(port)`	`flood`	`mod(h=v)`
↓	↓	↓	↓
0 packets	1 packets	0,1, or more packets	1 packets

`Pyretic` Policy

A function mapping a located packet to a set of located packets

```
eval(policy,packet) = {packet}
```

Puts focus on ***meaning*** instead of ***mechanics***

Enabling Compositional Operators

Parallel '|': *Do both C1 and C2 simultaneously*

```
eval(C1 | C2,p) =  eval(C1,p) U eval(C2,p)
```

Sequential '>>': *First do C1 and then do C2*

```
eval(C1 >> C2,p) =  U {eval(C2,p')
                        for p' in eval(C1,p)}
```

No priorities needed!

```
match(dstip=A)[fwd(2)] |
match(dstip=B)[fwd(3)] |
~(match(dstip=A) | match(dstip=b))[fwd(1)]
```

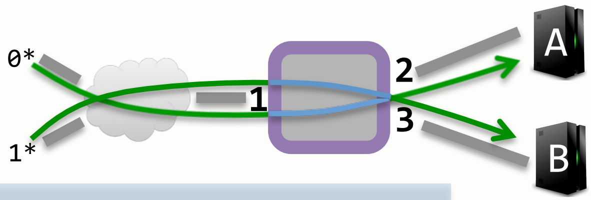
Querying as Forwarding

```
bucket(limit,[h])    count_bucket(every,[h])
```

Abstract location corresponding to a data-structure that store packet-data and callback processing routines

```
b = count_bucket(every=1)
b.register_callback(print)
match(srcip=X)[fwd(b)]
```

Monitoring Load Balancer




```
balance =
  match(srcip=0*,dstip=P)[mod(dstip=A)] |
  match(srcip=1*,dstip=P)[mod(dstip=B)] |
  ~match(         dstip=P)[id           ]
```

eval(id,p) = {p}

```
route =
  match(dstip=A)[fwd(2)] |
  match(dstip=B)[fwd(3)] |
  ~(match(dstip=A) | match(dstip=B))[fwd(1)]
```

```
b = counts(every=1)
b.register_callback(print)
monitor = match(srcip=X)[fwd(b)]
```

```
mlb = (balance >> route) | monitor
```

Compared to

```
install_flowmod(5,srcip=X &  dstip=P,[mod(dstip=A), fwd(2)])
install_flowmod(4,srcip=0* & dstip=P,[mod(dstip=A), fwd(2)])
install_flowmod(4,srcip=1* & dstip=P,[mod(dstip=B), fwd(3)])
install_flowmod(4,srcip=X & dstip=A ,[                fwd(2)])
install_flowmod(4,srcip=X &  dstip=B,[                fwd(3)])
install_flowmod(3,           dstip=A,[                fwd(2)])
install_flowmod(3,           dstip=B,[                fwd(3)])
install_flowmod(2,srcip=X           ,[                fwd(1)])
install_flowmod(1,*                 ,[                fwd(3)])
```

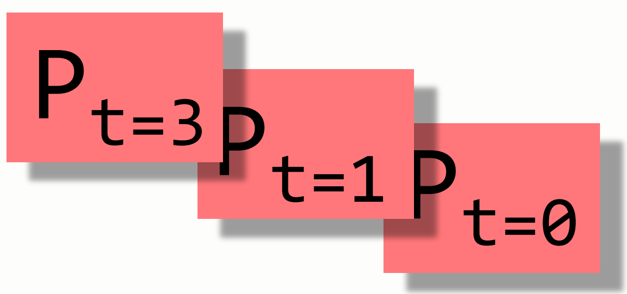
Pyretic's Design

- Monitoring Load Balancer
  - Encode Policies as Functions
  - Compositional Operators
  - Queries as Forwarding Policies
- MAC-Learning Module
  - Dynamic Policies
- "One Big Switch" Topology Abstraction
  - Extensible packet model

How Do We Change Policies?

Dynamic policy

a time-series of policies



MAC-Learning Module

```
class learn():
  def init(self):
    b = bucket(limit=1,['srcmac','switch'])
    b.register_callback(update)
    self.P = flood | fwd(b)
```

if_(P,C1,C2) = P[C1] | ~P[C2]

Update current val

to flood

Otherwise, policy unchanged

```
  def update(self,pkt):
    self.P = if_(match(dstmac=pkt['srcmac'],
                        switch=pkt['switch']),
                 fwd(pkt['inport']), self.P)
```

If newly learned MAC

Forward directly to learned port

MAC-Learning Module

Time-series object

First packet with unique
`srcmac, switch`

```
class learn():
  def init(self):
    b = bucket(limit=1,['srcmac','switch'])
    b.register_callback(update)
    self.P = flood | fwd(b)
```

Defined momentarily

to flood

and query

Initialize current
value of time series

Pyretic's Design

- Monitoring Load Balancer
  - Encode Policies as Functions
  - Compositional Operators
  - Queries as Forwarding Policies
- MAC-Learning Module
  - Dynamic Policies
- "One Big Switch" Topology Abstraction
  - Extensible packet model

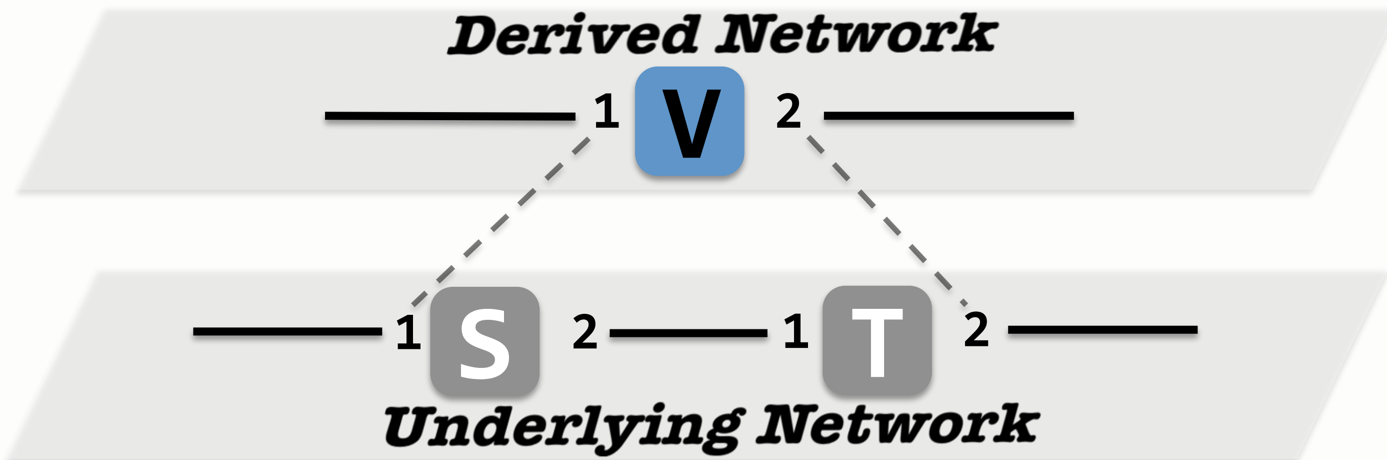
Extensible `Pyretic` Packet Model

Field	Val[0]	Val[1]
srcmac		
dstmac		
proto		
srcip		
...		
switch		
inport		
outport		

- All OpenFlow fields
- Location fields
- Virtual fields
- Stacks of values
  - `push(h=v)`
  - `pop(h)`
  - Actions and matches use (currently) top value

Implemented on OpenFlow by mapping extended field values to VLAN tags/MPLS labels

“One Big Switch” Topology Abstraction



- Simplest of topology abstraction examples
- Build a distributed middlebox
  by running centralized middlebox app on V!

Topology Abstraction

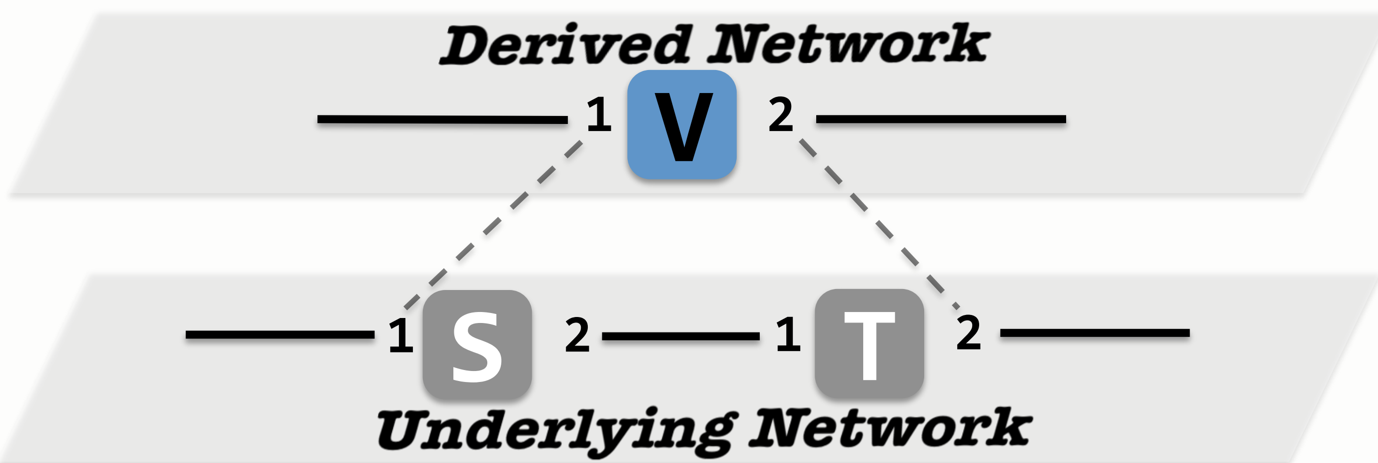
`abstract(ingress,fabric,egress,derived)`

using 3 partial transformation policies to:

- handle packets entering abstract switch
- move packets through abstract switch
- handle packets exiting abstract switch

Returns a new policy for the underlying network (i.e., on nodes `S` and `T`)

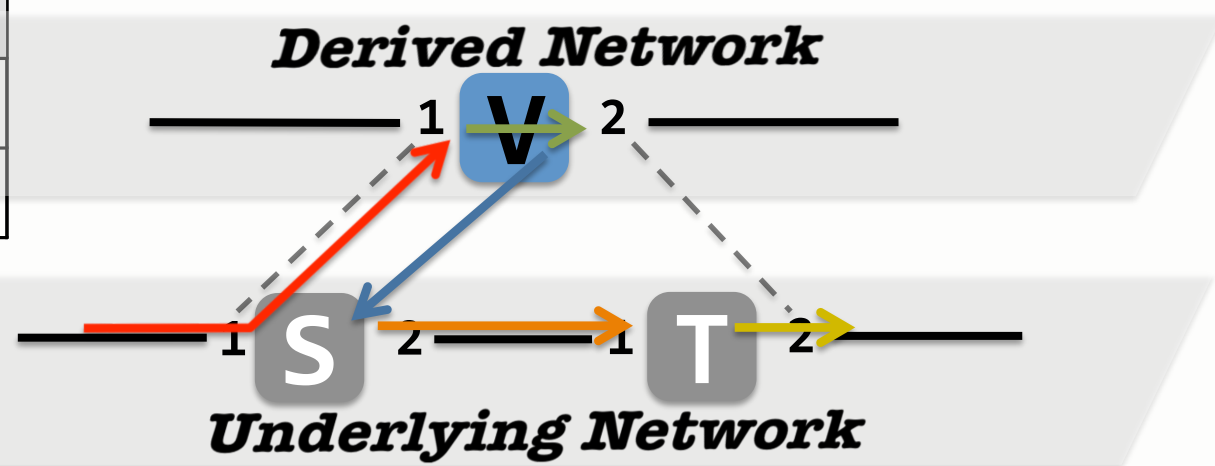
that “does” the `derived` policy on the abstract topology (i.e., on node `V`)



Implementing abstract()

```
def abstract(ingress,fabric,egress,derived):
  return ingress >>        # defines part of transform
         derived >>        # app run on abstract topo
         lower_packet >>   # built-in
         fabric >>         # defines part of transform
         egress            # defines part of transform
```

Field	V_0	V_1
switch	S	V
inport	1	1
vswitch / outport	V / 2	
vinport	1	
voutport	2	



Summary: `Pyretic` Policy Syntax

(You may already be a `Pyretic` programmer!)

8 Actions

A ::= `drop` | `fwd(port)` | `flood` | `mod(h=v)` |
`id` | `push(h=v)` | `pop(h)` | `move(h1=h2)`

6 Predicates

P ::= `all_packets` | `no_packets` | `match(h=v)` |
| *P* & *P* | (*P* | *P*) | ~*P*

2 Query Buckets

B ::= `bucket(limit,[h])` | `count_bucket(every,[h])`

5 Policies

C ::= *A* | `fwd`(*B*) | *P* [*C*] | (*C* | *C*) | *C* >> *C*

Summary: Abstractions

	Pyretic	**Current APIs**
Policy	Rich Composition	Little Composition
Network	Layered Abstract Topologies	Concrete Network
Packet	Extensible Headers	Fixed OpenFlow Headers

Related Work:
[Frenetic, Maestro, FRESCO] / [Click]

	`Pyretic`	**Current APIs**
Policy	Rich Composition	**Some** / **Full** Composition
Network	Layered Abstract Topologies	Concrete Network
Packet	Extensible Headers	Fixed OpenFlow Headers

But only for a single software switch
not multiple hardware switches

Related Work:
[FlowVisor] / [Nicira NVP, OpenStack Quantum]

	Pyretic	Current APIs
Policy	Rich Composition	Little Composition
Network	Layered Abstract Topologies	**Disjoint Slices / Topology Hiding**
Packet	Extensible Headers	Fixed OpenFlow Headers

Both approaches support multi-tenancy, but not topological decomposition (of functional composition)

Pyretic Interpreter and Suite of Apps

Available at www.frenetic-lang.org/pyretic

- Monitoring & DPI
- Load Balancers
- Hub
- ARP
- Firewalls
- MAC learner

- Abstractions
  - Big switch (one-to-many)
  - Spanning tree (many-to-many)
  - Gateway (many-to-one)

And bigger applications built by combining these.

And More!

Available at www.frenetic-lang.org/pyretic

- Features Request
- Bug reporting
- Link to github
- Discuss list
- Join the project

- Dev Roadmap
  - Reactive (microflow) runtime
  - Proactive (compilation) runtime
  - Optimizations
  - Caching

Thanks for Listening!