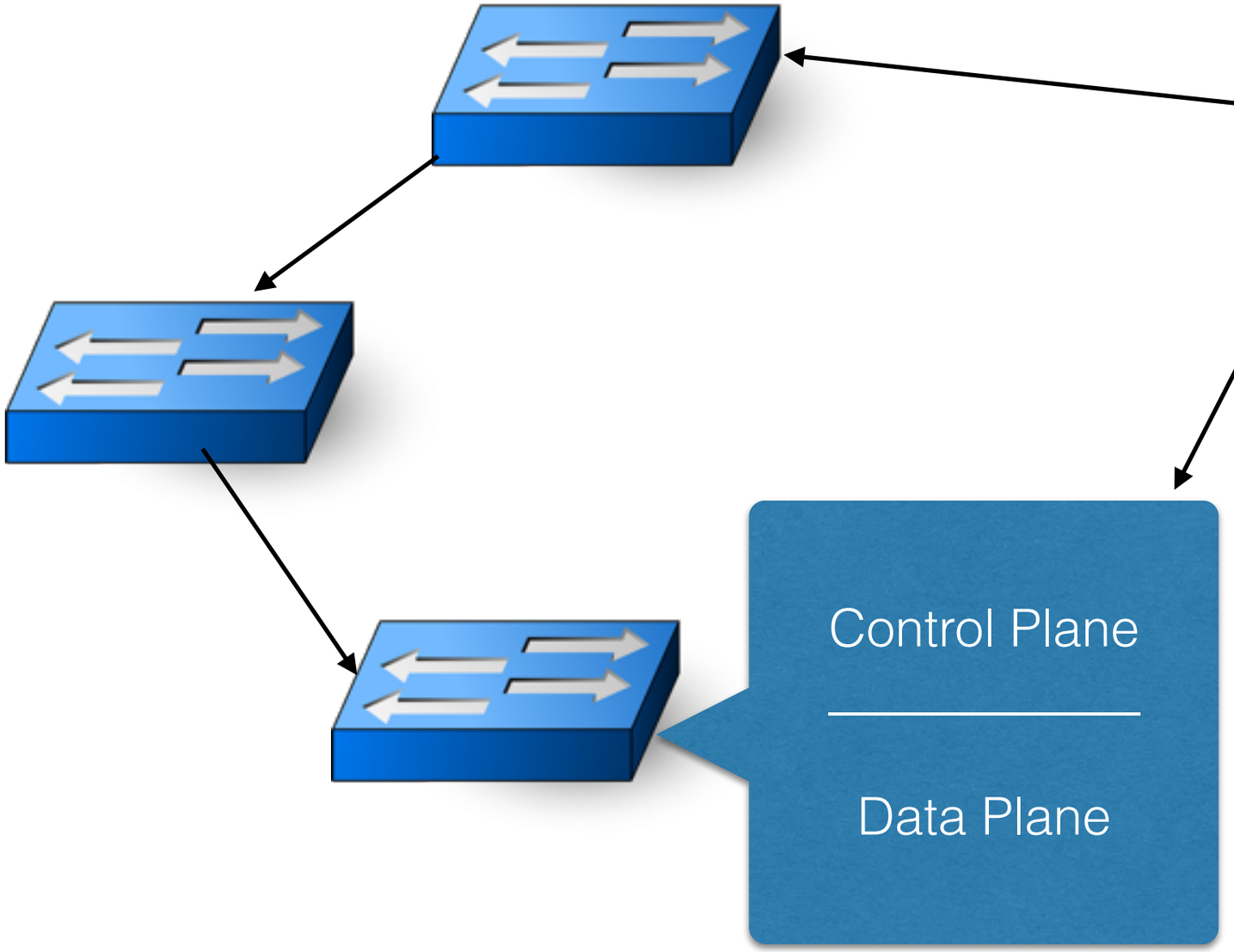
Frenetic

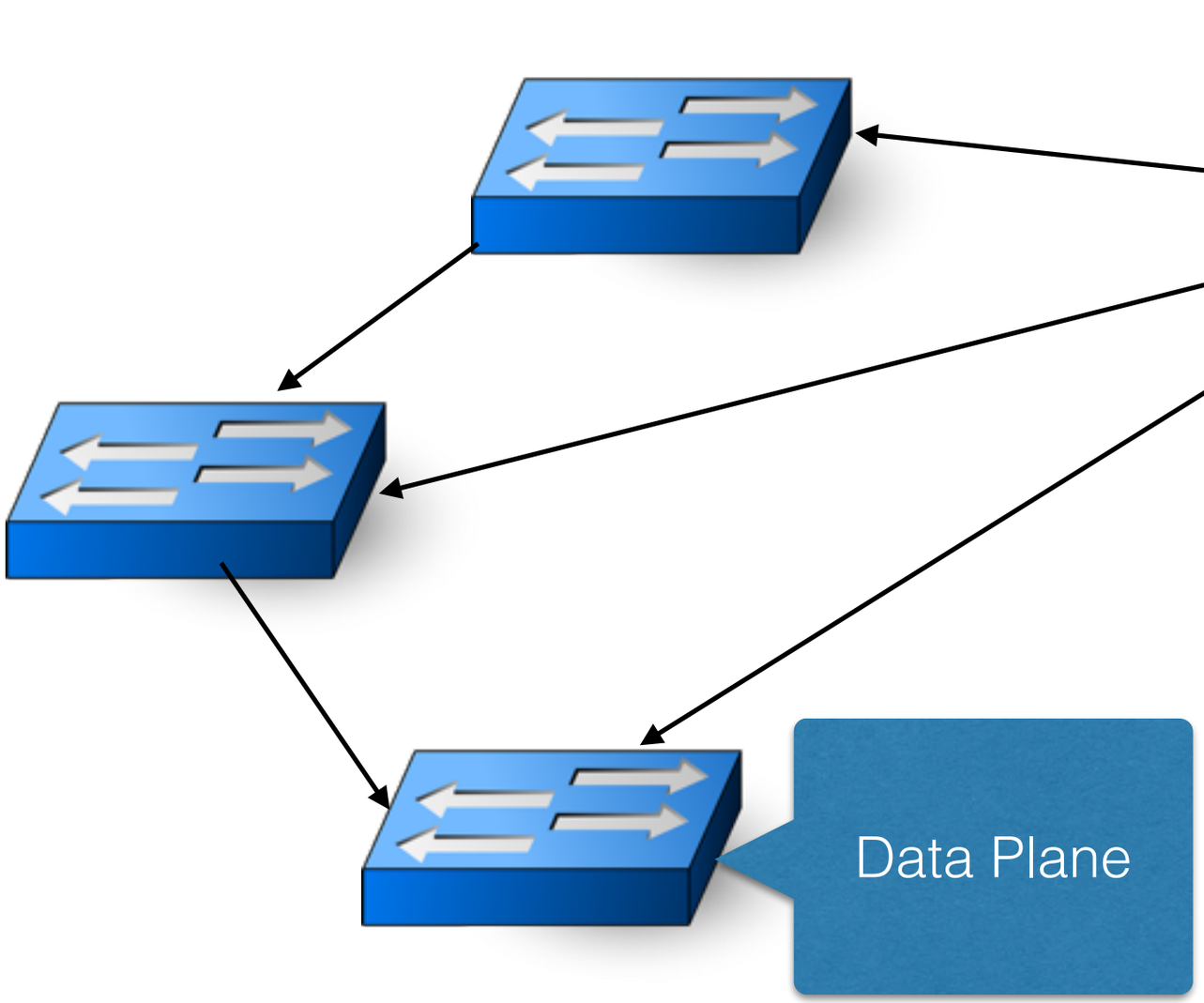
A Network Programming Language

Traditional Networks



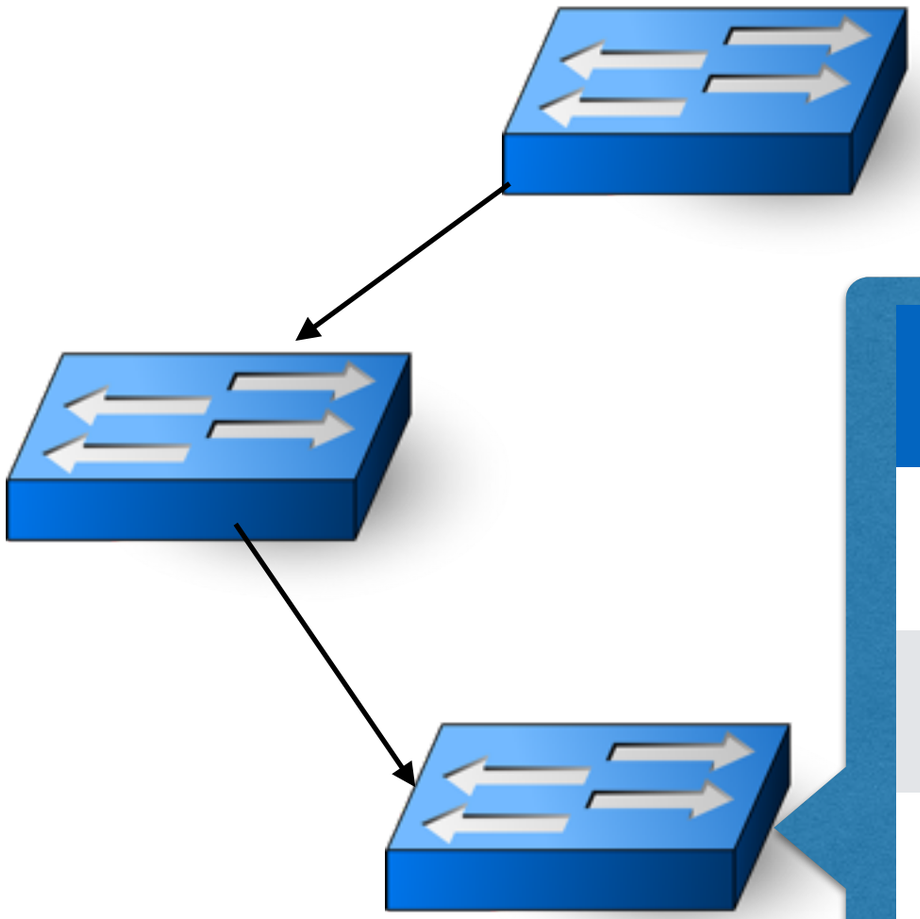
Operators

SDN



Operators

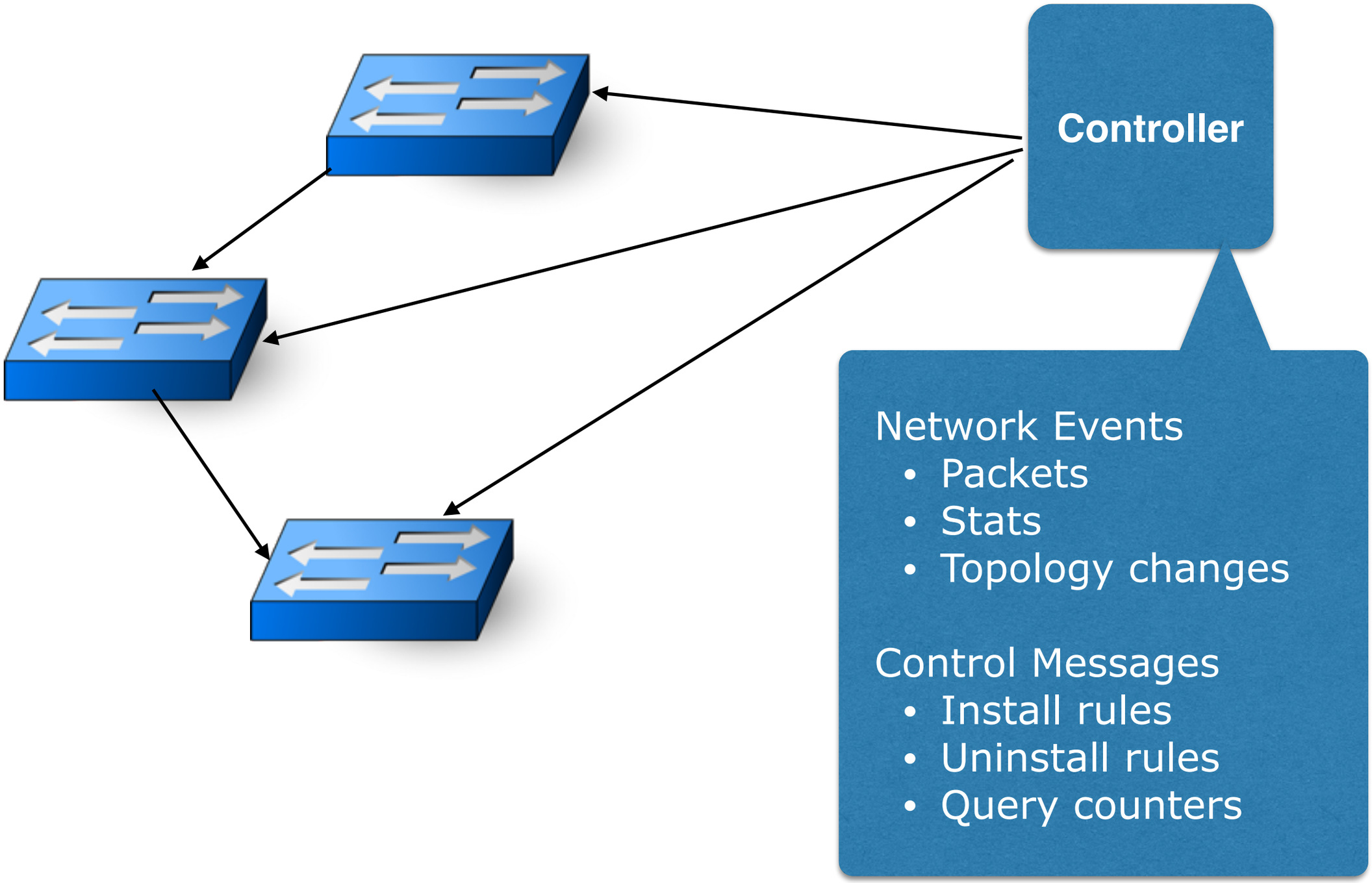
OF Switches



Pattern	Action	Bytes	Packets
1010	Drop	200	10
010*	Forward(3)	100	3
011*	Controller	0	0

Priority

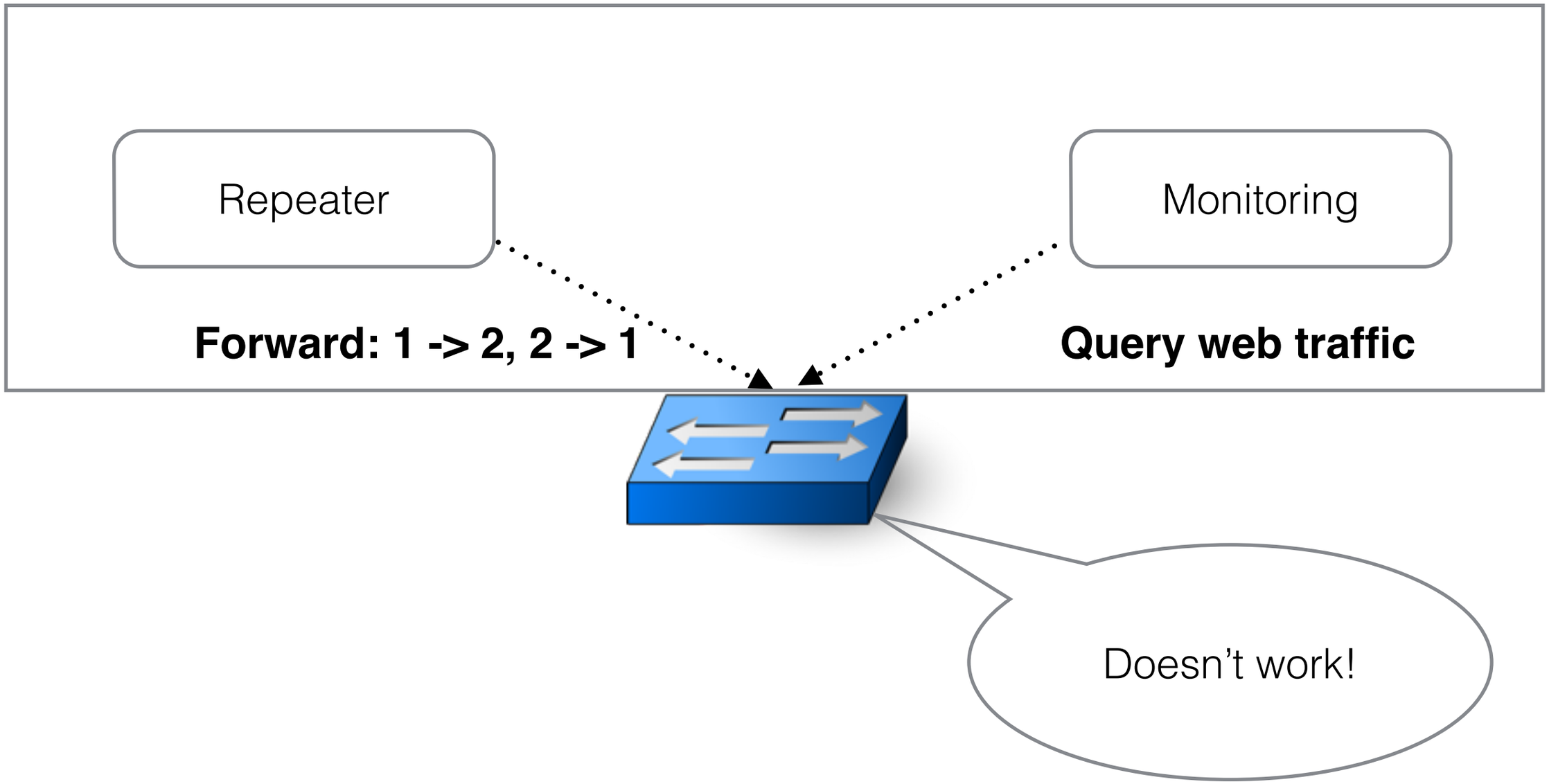
OF Controller



New Challenges

- OpenFlow makes it **possible** to program the network, but it does not make it **easy**!

Problem 1: Anti-Modular



Problem 1: Anti-Modular

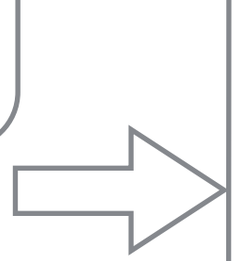
```
def switch_join(sw):
     repeater(sw)

def repeater(sw):
     pat1 = {in_port:1}
     pat2 = {in_port:2}
     install(sw, pat1, DEFAULT, None, [output(2)])
     install(sw, pat2, DEFAULT, None, [output(1)])
```

CONFLICTS!

```
def monitor(sw):
     pat = {in_port:2, tp_src:80}
     install(sw, pat, DEFAULT, None, [])
     query_stats(sw, pat)

def stats_in(sw, xid, pattern, packets, bytes):
     print bytes
     sleep(30)
     query_stats(sw, pattern)
```



```
def switch_join(sw):
     repeater_and_monitor(sw)

def repeater_and_monitor(sw):
     pat1 = {in_port:1}
     pat2 = {in_port:2}
     pat2web = {in_port:2, tp_src:80}
     install(sw, pat1, DEFAULT, None, [output(2)])
     install(sw, pat2web, HIGH, None, [output(1)])
     install(sw, pat2, DEFAULT, None, [output(1)])
     query_stats(sw, pat2web)

def stats_in(sw, xid, pattern, packets, bytes):
     print bytes
     sleep(30)
     query_stats(sw, pattern)
```

Problem 2: Two-tiered Model

- Tricky problem:
  - Controller activity is driven by packets
  - For efficiency, applications install rules to forward packets in hardware
- Constant questions:
  - “Will what packet come to the controller and trigger my computation?”
  - “Or is it already being handled invisibly on the switch?”

Problem 3: Race Conditions

- Example: More than one packets sent to the controller at once
    - **Controller** analyze the 1st packet, updates its state, initializes installation
    - *Laggggggggggggggggggg*
    - **Switch** hasn't received the new rules, send the 2nd packet to controller
    - **Controller** confused…

Root of evil

- Three problems:
  - Anti-modular
  - Two-tiered model
  - Network race conditions
- One cause:
  - No effective **abstractions** for **reading** network states.

The Solution

- Separate network programming into two parts:
  - Abstractions for **querying network state**
  - Abstractions for **specifying a forwarding policy**

Frenetic Language

- Abstractions for querying network state
  - An integrated query language
    - select, filter, group, sample sets of packets or stats
    - designed so that computation can occur on data plane
- Abstractions for specifying a forwarding policy
  - A functional stream processing library (based on FRP)
    - generate streams of network policies
    - transform, split, merge, filter policies and other streams.
- Implementation:
  - A collection of Python libraries on top of NOX.

Frenetic Architecture

High level Language:

- Integrated query language
- Effective support for composition and reuse

Run-time system:

- Interprets queries, policies
- Installs rules
- Tracks stats
- Handles asynchronous events

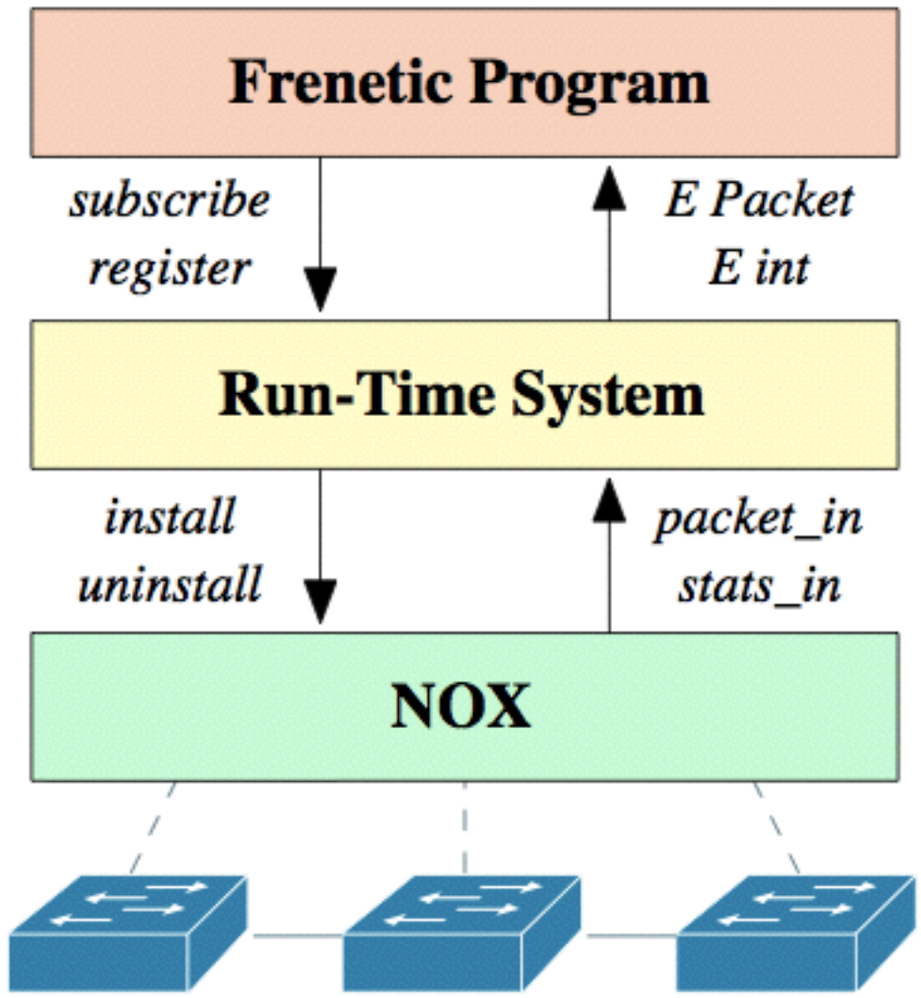
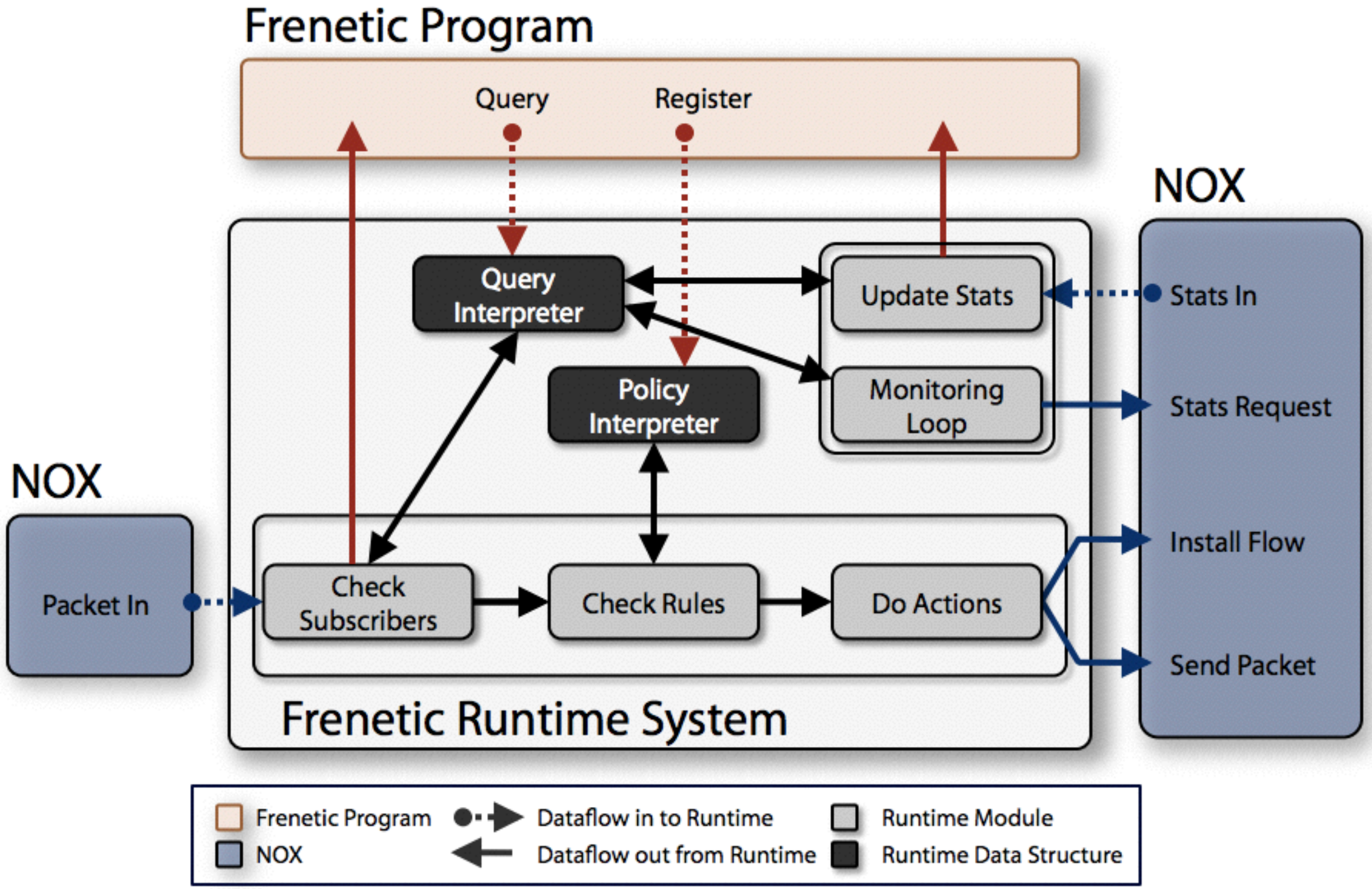


Figure 7. Frenetic architecture.

Frenetic Run-time



…from bits to functions…

Openflow Syntax

$$
\begin{array}{lll}
\textit{Integers} & & n \\
\textit{Rules} & r ::= & \langle pat, pri, t, [a_1, \ldots, a_n] \rangle \\
\textit{Patterns} & pat ::= & \{h_1 : n_1, \ldots, h_k : n_k\} \\
\textit{Priorities} & pri ::= & n \\
\textit{Timeouts} & t ::= & n \mid \texttt{None} \\
\textit{Actions} & a ::= & \texttt{output}(op) \mid \texttt{modify}(h, n) \\
\textit{Headers} & h ::= & \texttt{in\_port} \mid \texttt{vlan} \mid \texttt{dl\_src} \mid \texttt{dl\_dst} \mid \texttt{dl\_type} \mid \\
 & & \texttt{nw\_src} \mid \texttt{nw\_dst} \mid \texttt{nw\_proto} \mid \texttt{tp\_src} \mid \texttt{tp\_dst} \\
\textit{Ports} & op ::= & n \mid \texttt{flood} \mid \texttt{controller}
\end{array}
$$

Frenetic Syntax

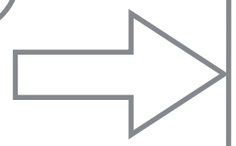
Queries	q	::= `Select`(a) *
		`Where`(fp) *
		`GroupBy`$([qh_1, \ldots, qh_n])$ *
		`SplitWhen`$([qh_1, \ldots, qh_n])$ *
		`Every`(n) *
		`Limit`(n)
Aggregates	a	::= `packets` | `sizes` | `counts`
Headers	qh	::= `inport` | `srcmac` | `dstmac` | `ethtype` |
		`vlan` | `srcip` | `dstip` | `protocol` |
		`srcport` | `dstport` | `switch`
Patterns	fp	::= `true_fp`() | qh`_fp`(n) |
		`and_fp`$([fp_1, \ldots, fp_n])$ |
		`or_fp`$([fp_1, \ldots, fp_n])$ |
		`diff_fp`(fp_1, fp_2) | `not_fp`(fp)

Frenetic

```
def web_query():
    return (Select(counts) *
            Where(inport_fp(1)) *
            GroupBy([srcmac]) *
            Every(60))
```

```
rules = [Rule(inport_fp(1), [forward(2)]),
         Rule(inport_fp(2), [forward(1)])]

def repeater():
    return (SwitchJoin() >>
            Lift(lambda switch: {switch:rules}))
```



```
def main():
    web_query() >> Print()
    secure(Merge(web_query(), repeater()))
>> Register()
```

Further challenges

- Performance evaluation & optimization
- Extend queries & controls to end hosts
- More abstractions
  - Virtual network topologies
  - Network updates with improved semantics

Thank you!