Network Programming with frenetic

Nate Foster (Cornell)
Arjun Guha (UMass)
Mark Reitblatt (Cornell)
Cole Schlesinger (Princeton)

FMCAD '13 Tutorial

Network Programming with frenetic

http://bit.ly/frenetic-tutorial

Arjun Guha (UMass)
Mark Reitblatt (Cornell)
Cole Schlesinger (Princeton)

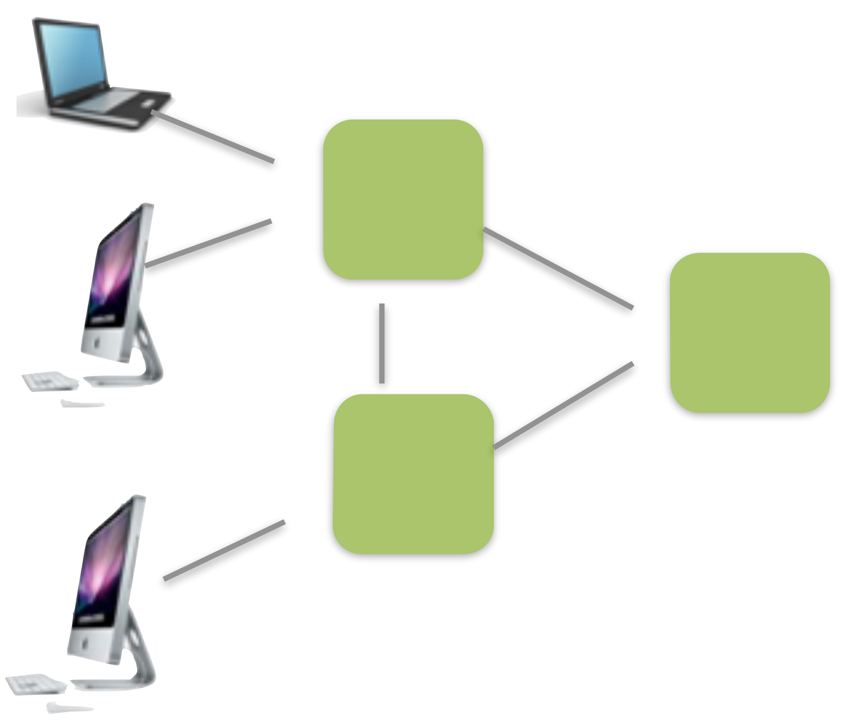
FMCAD '13 Tutorial

Networks Today

There are hosts...

Networks Today

Connected by switches...



Networks Today

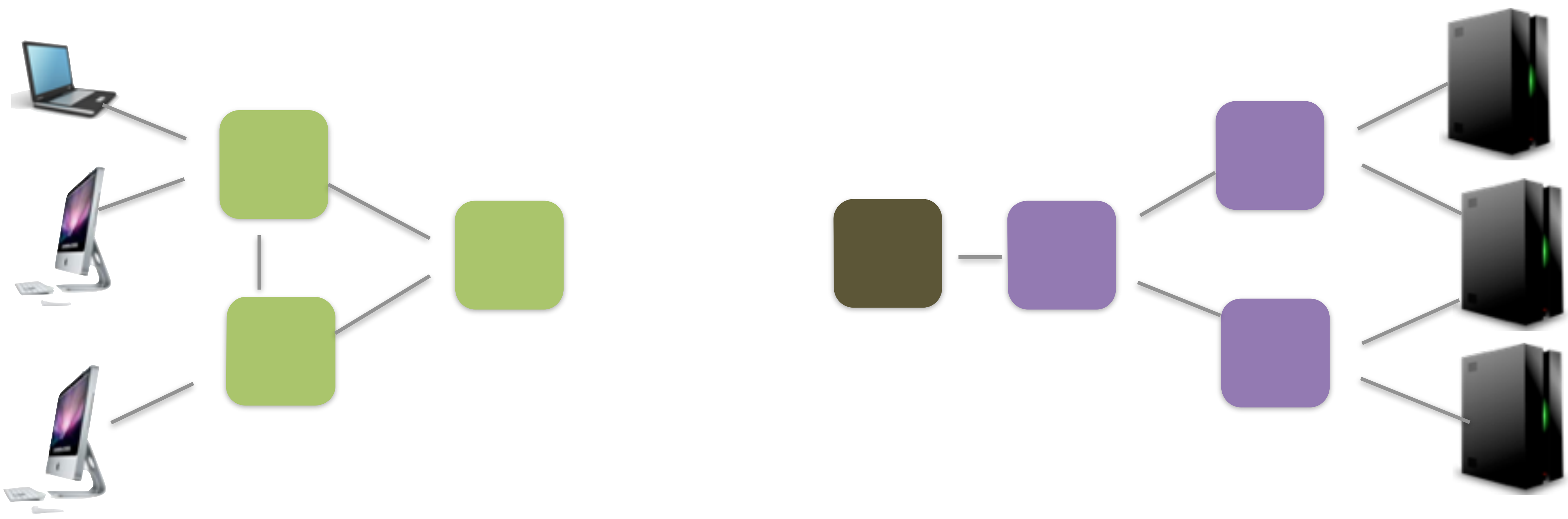
There are also servers...

Networks Today

Connected by routers...

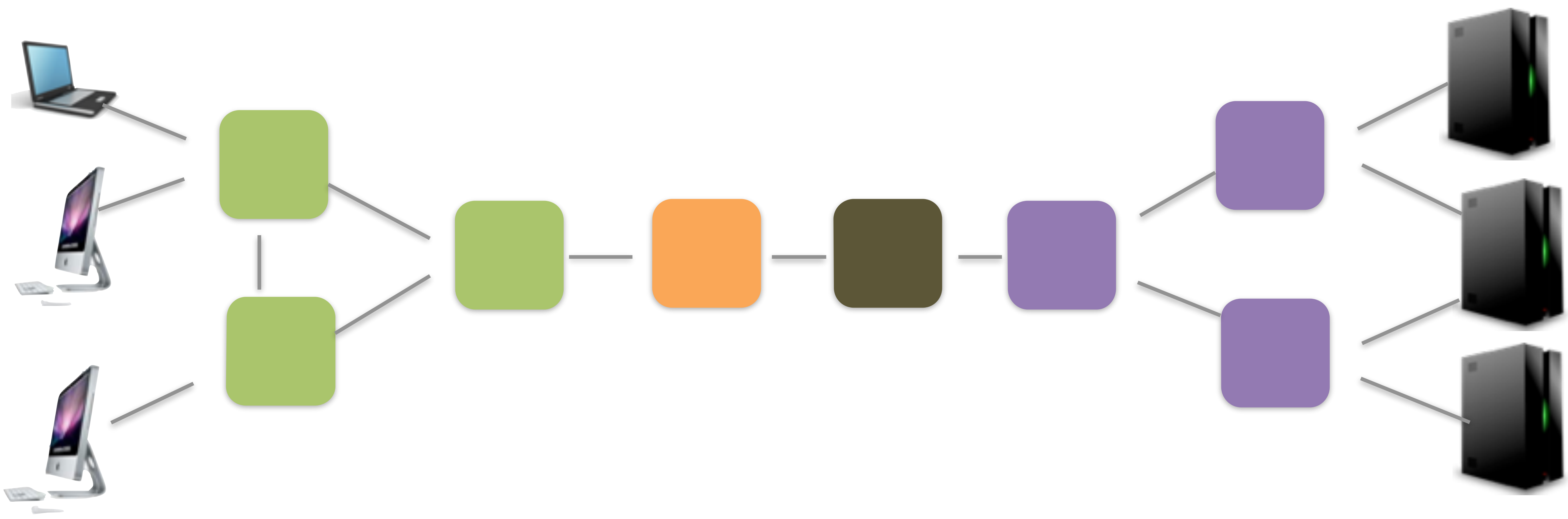
Networks Today

And a load balancer...



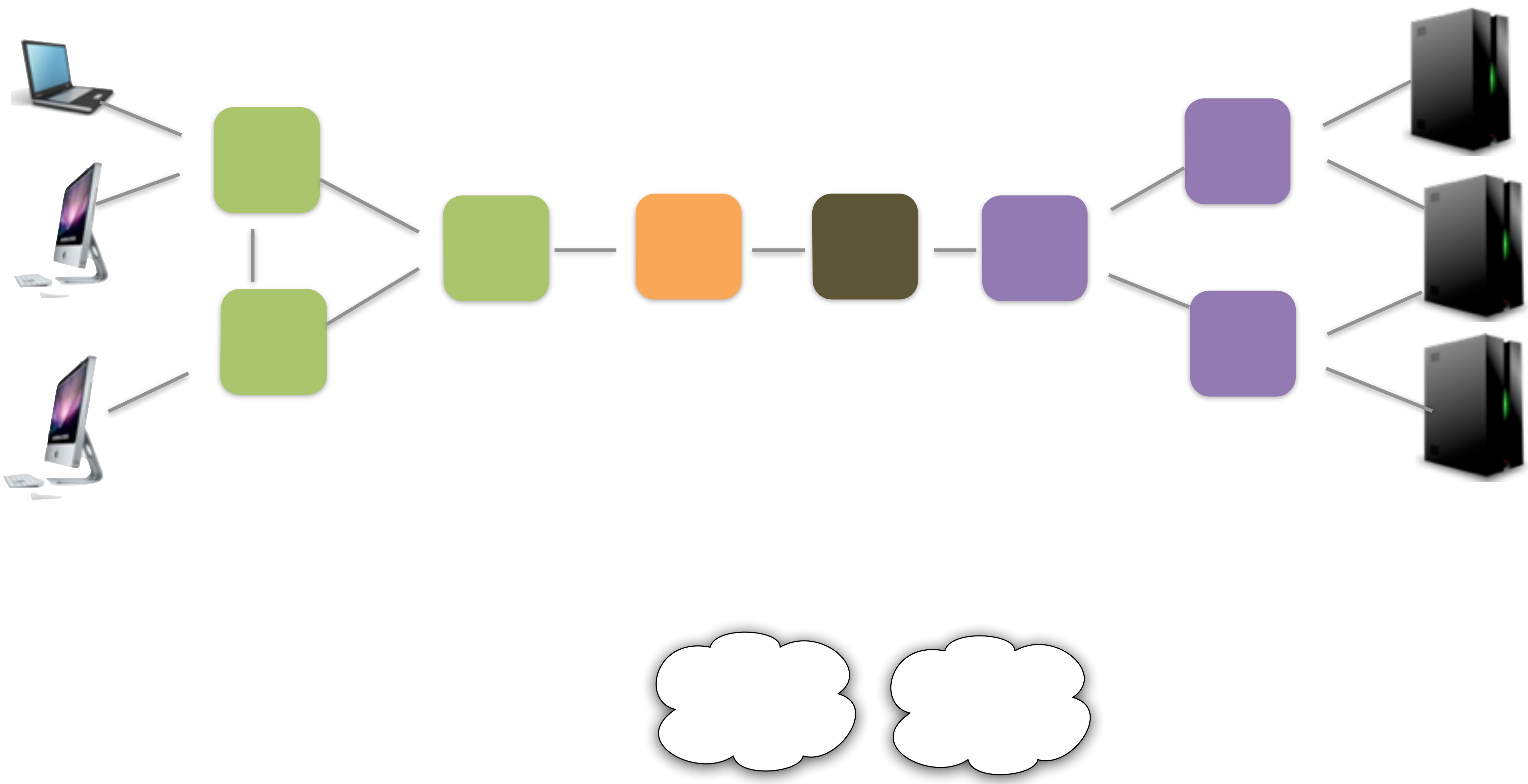
Networks Today

And a gateway router...



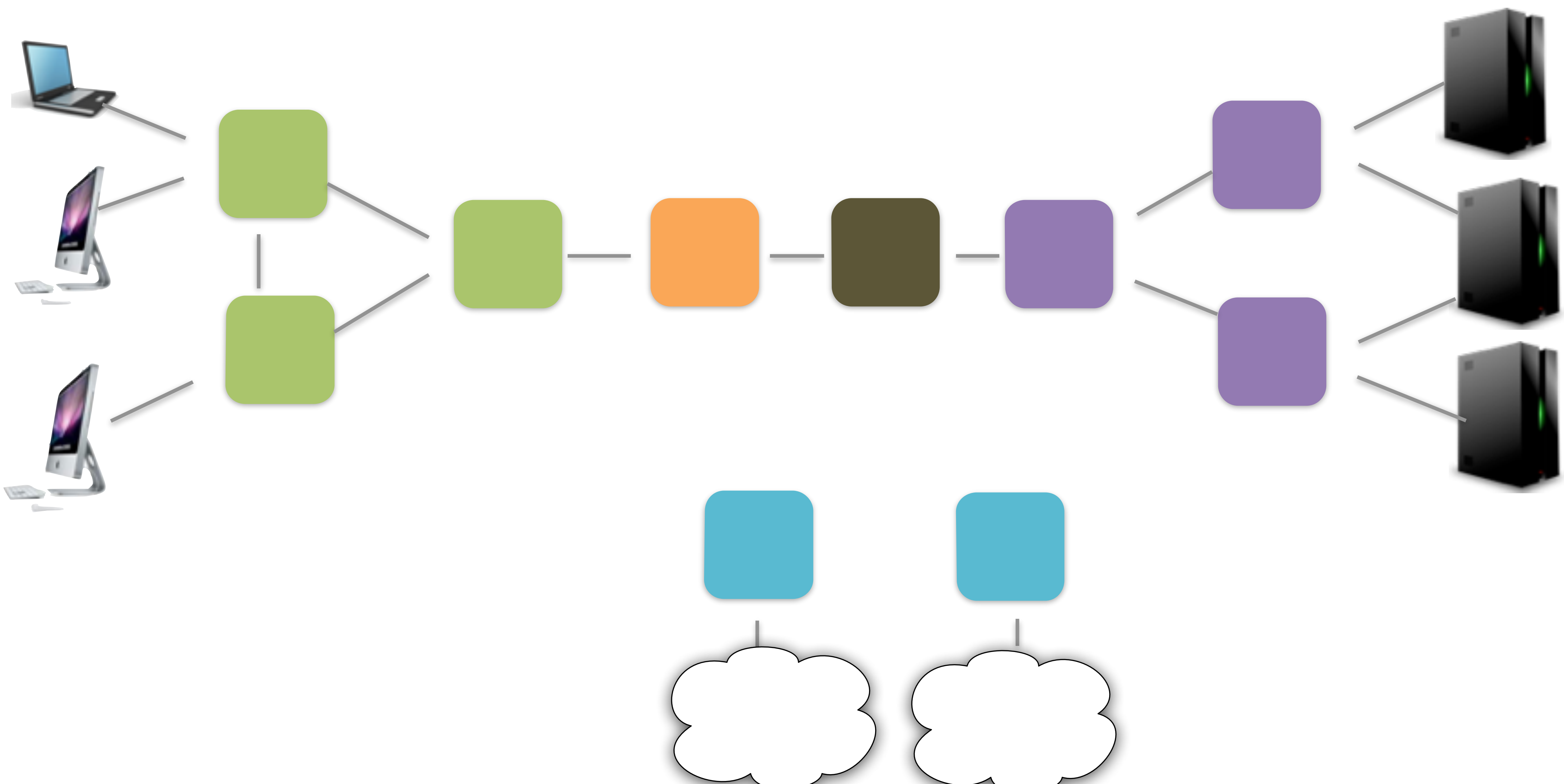
Networks Today

There are other ISPs...



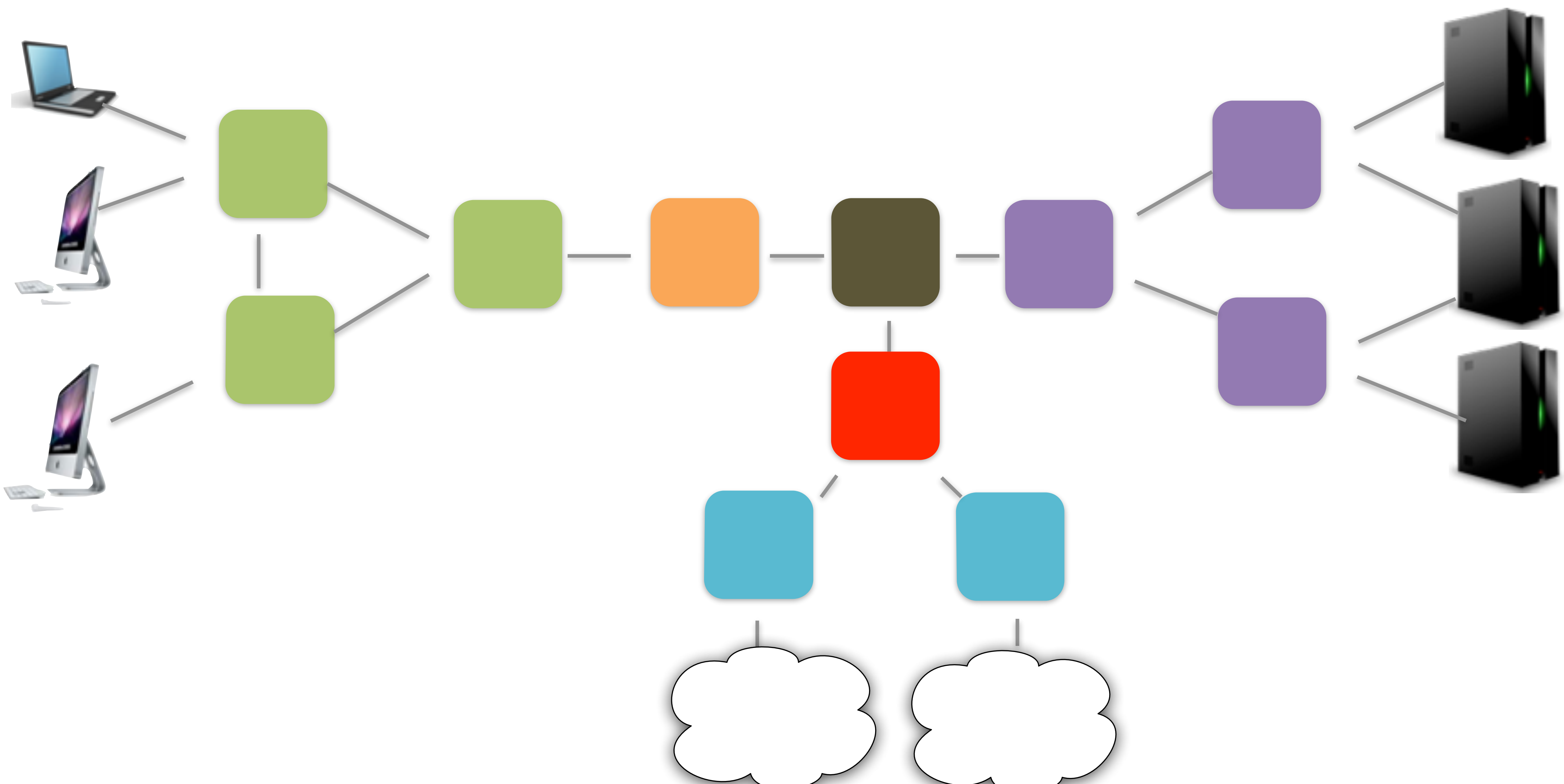
Networks Today

So we need to run BGP...



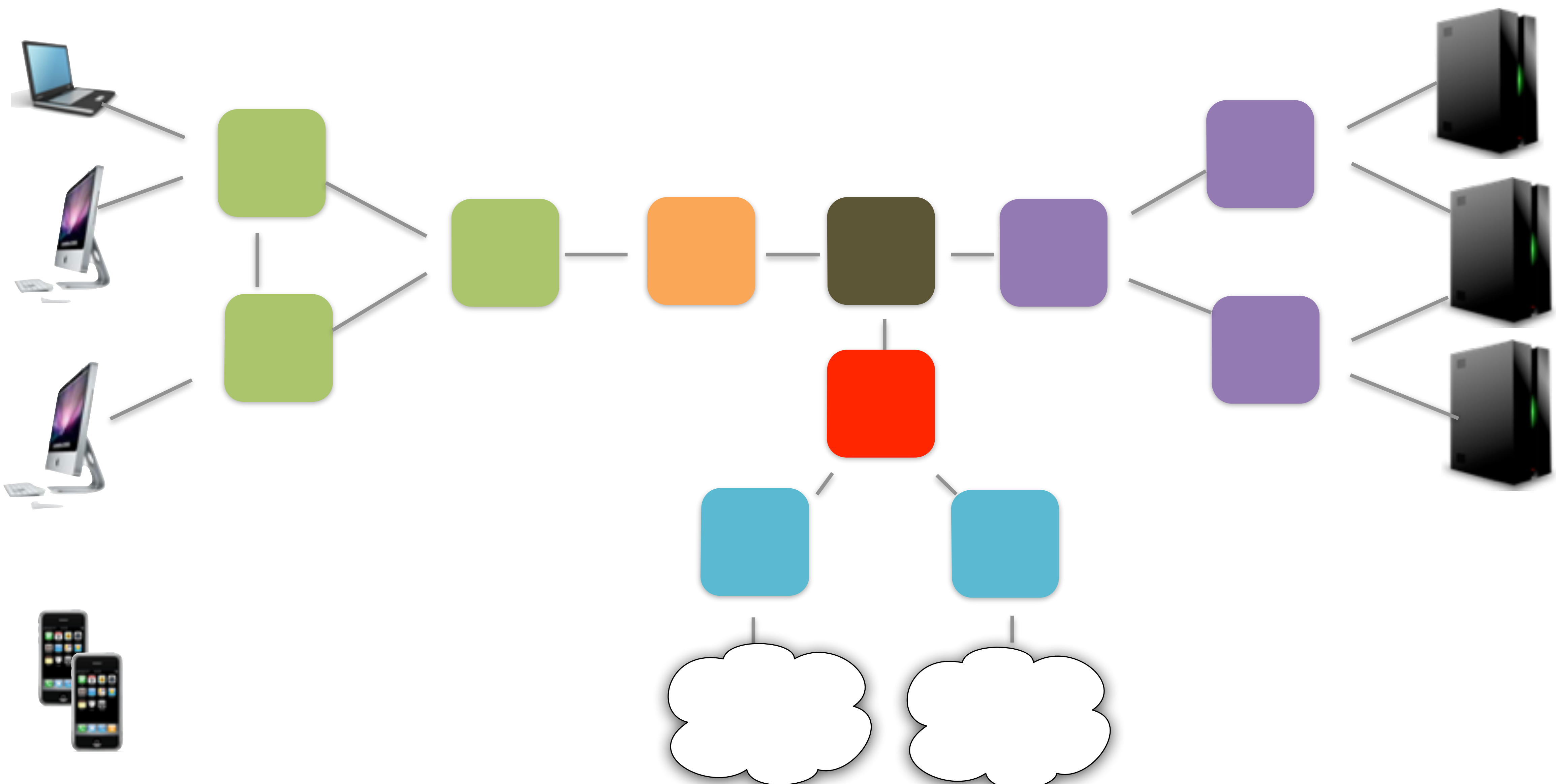
Networks Today

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



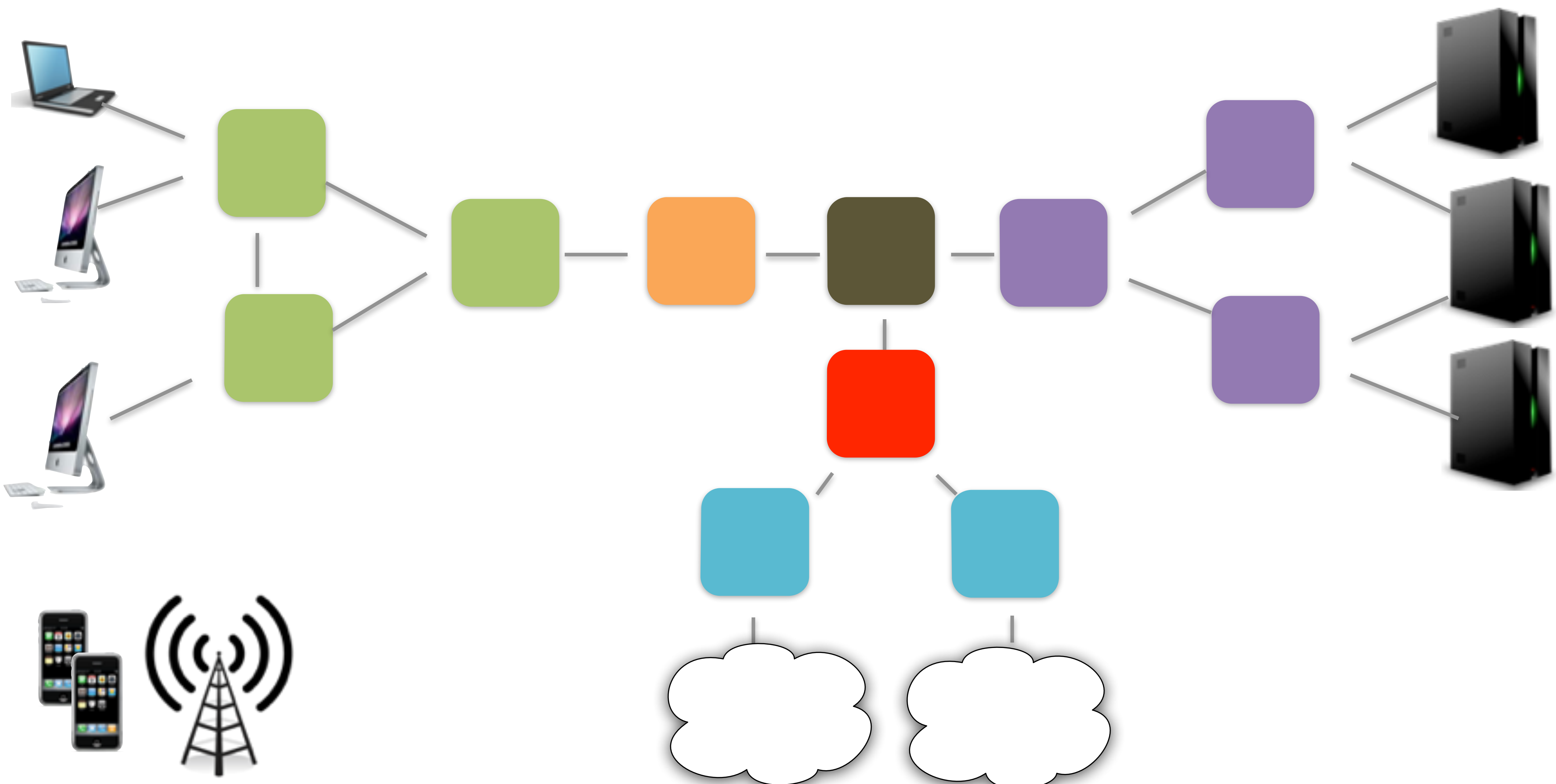
Networks Today

There are also wireless hosts...



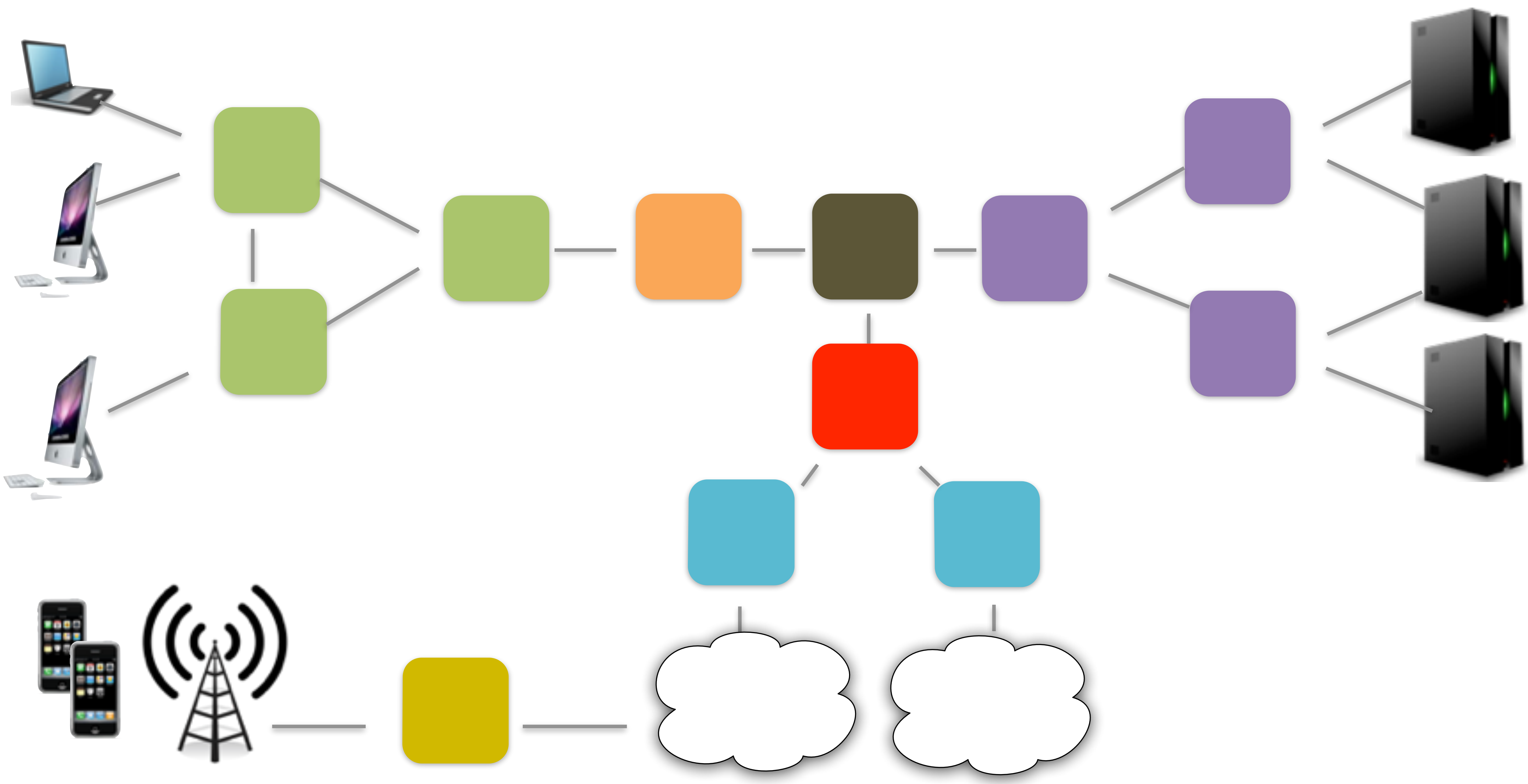
Networks Today

So we need wireless gateways...



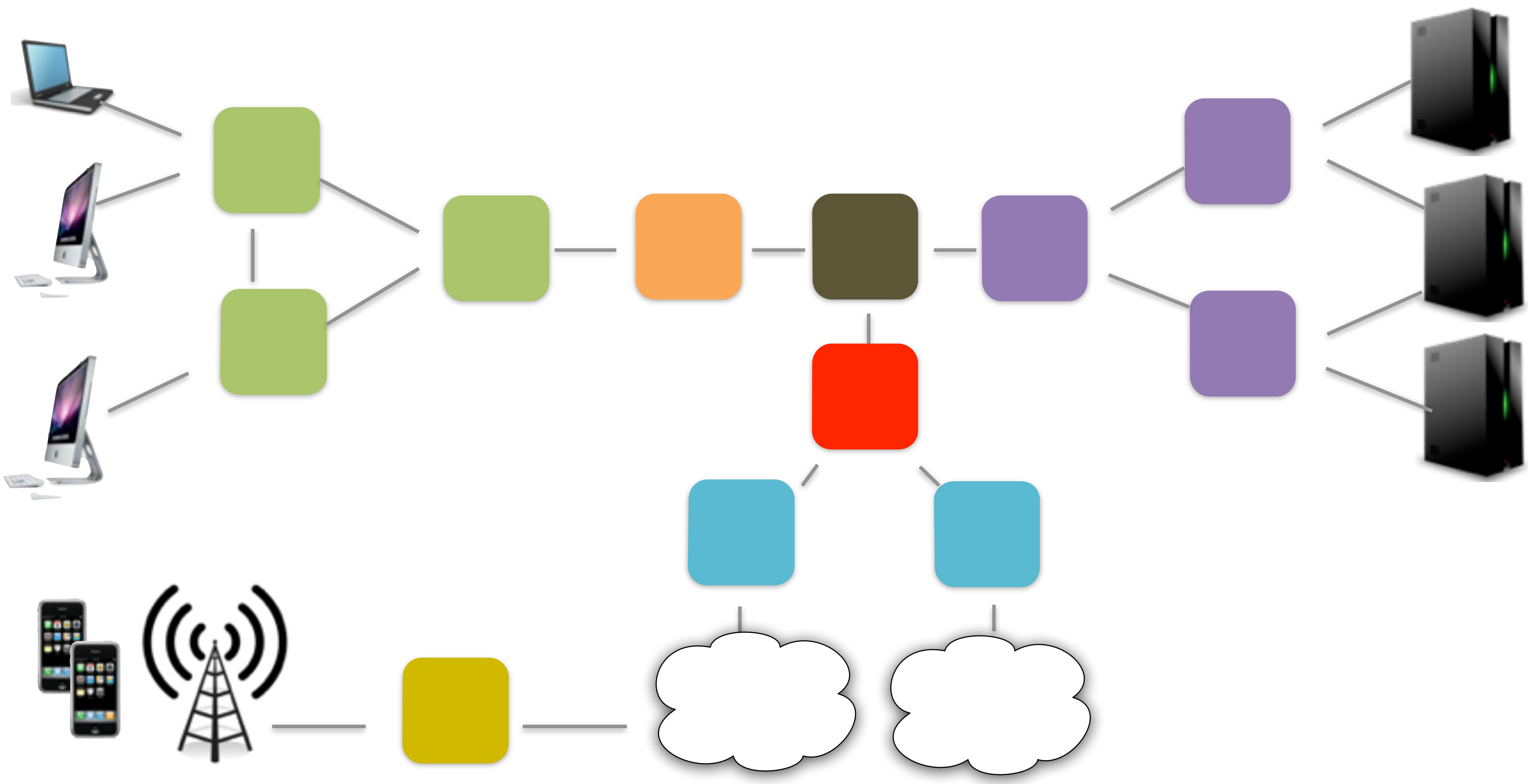
Networks Today

And yet more middleboxes for lawful intercept...



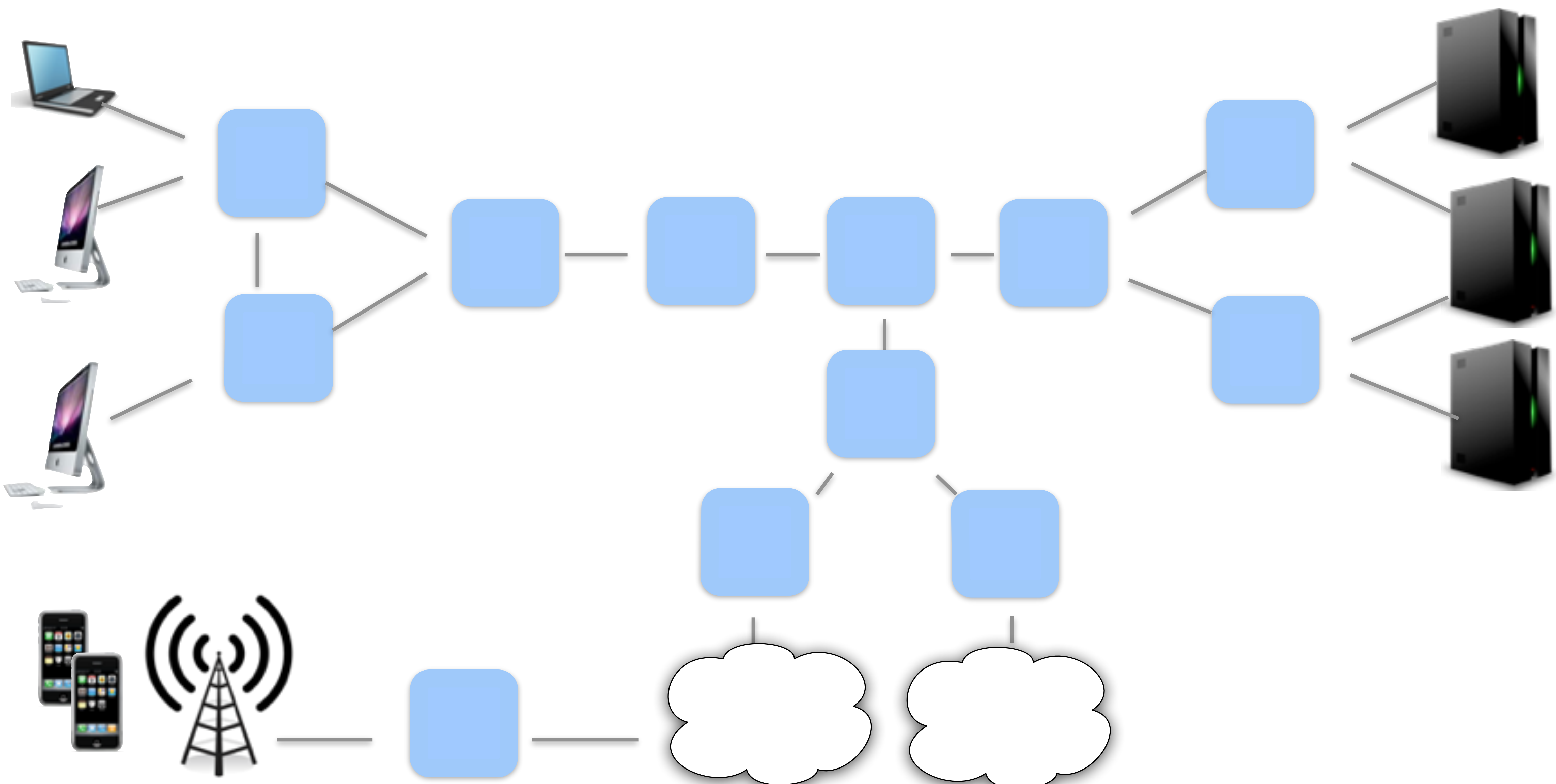
Networks Today

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



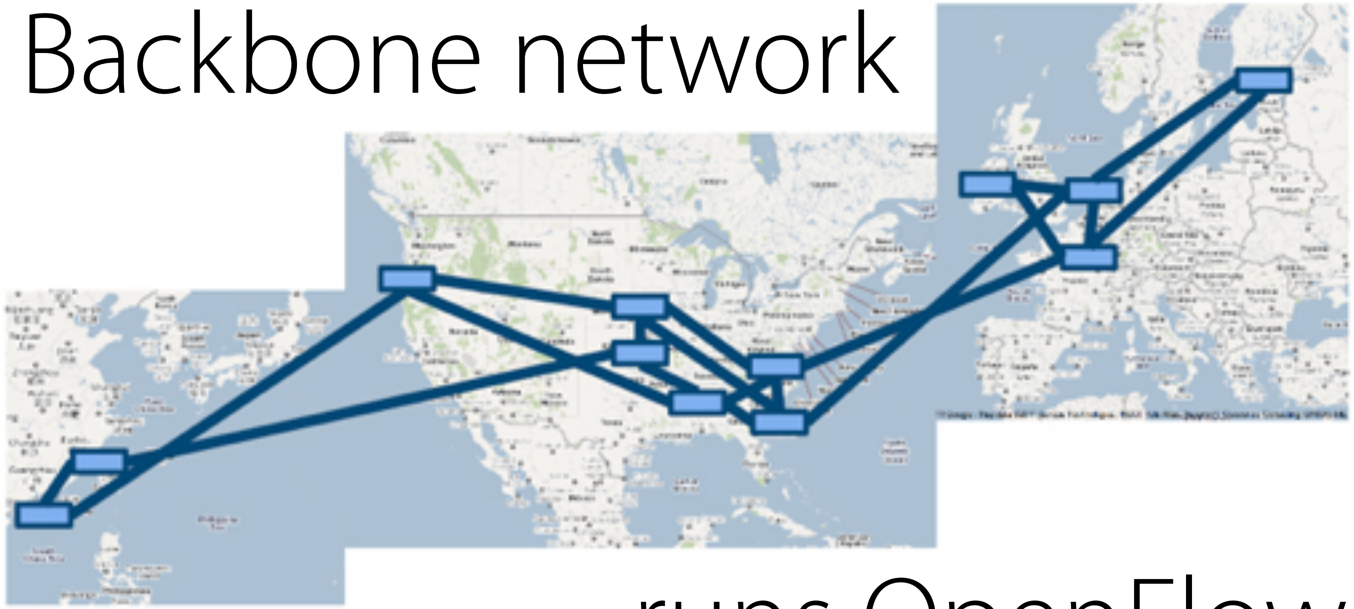
Software-Defined Networking

A clean-slate programmable network architecture



A Major Trend in Networking

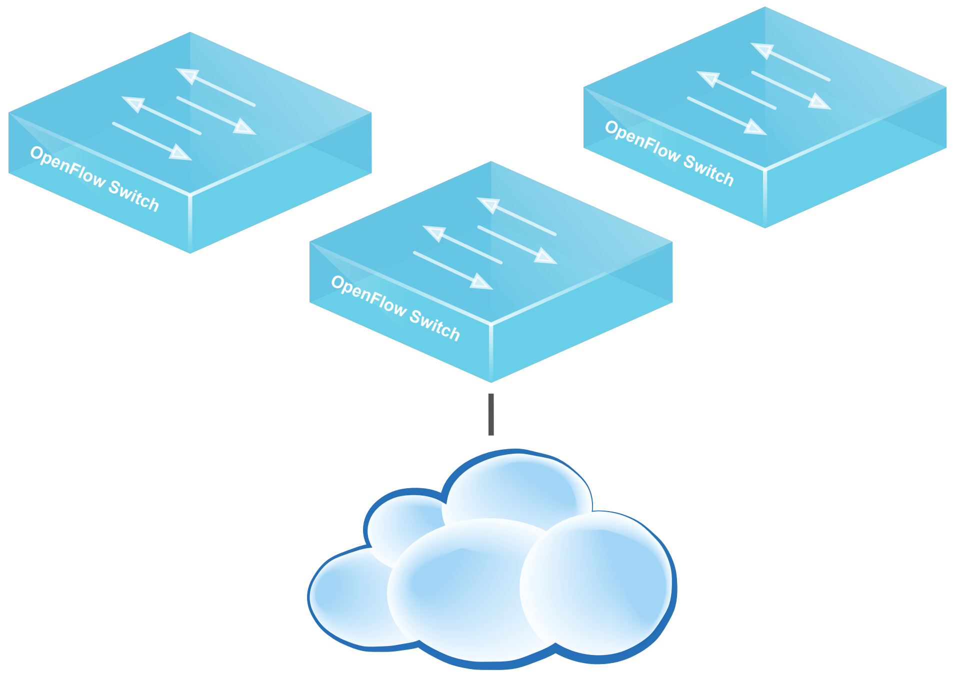
Backbone network



runs OpenFlow

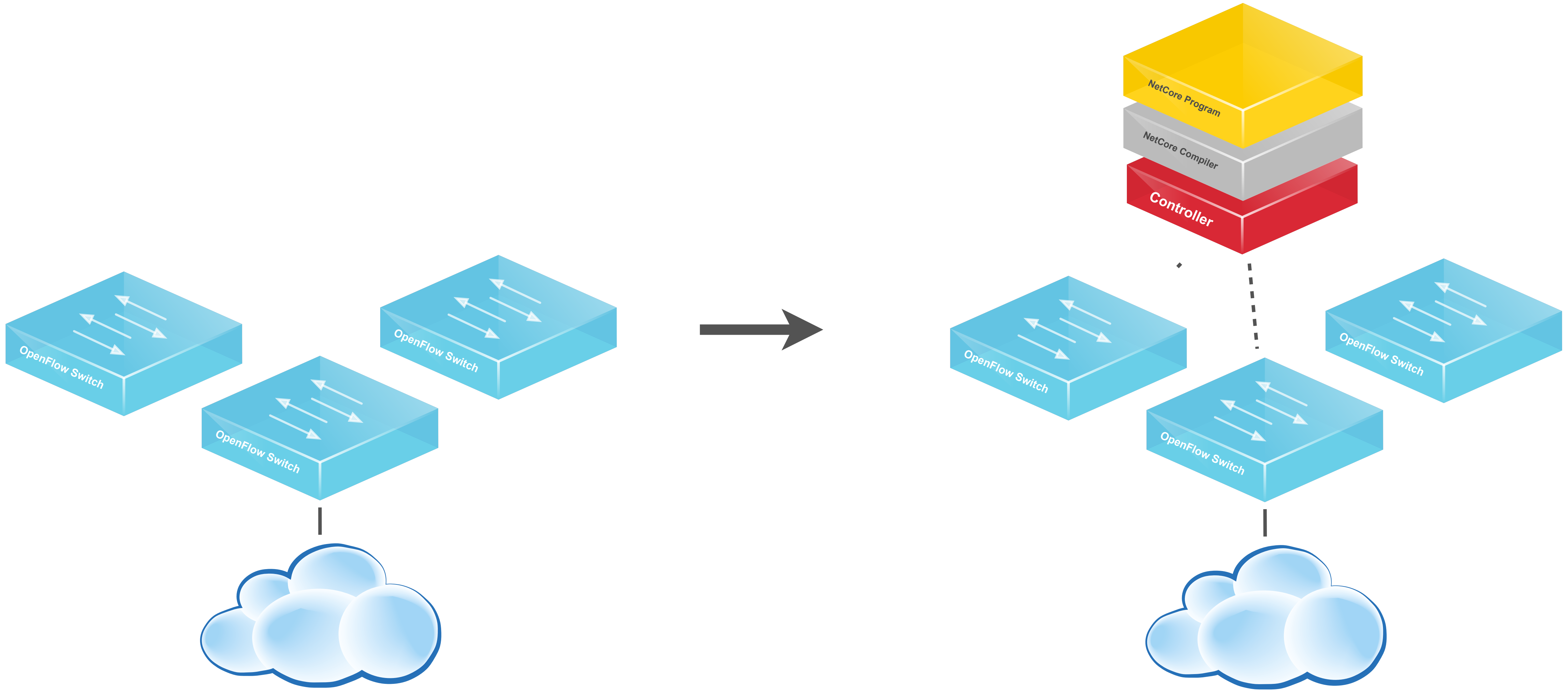
Bought for $1.2B (mostly cash)

Vision: program networks using a high-level language, generate low-level machine code using a compiler, and verify formal properties of networks automatically



frenetic

Vision: program networks using a high-level language, generate low-level machine code using a compiler, and verify formal properties of networks automatically



Tutorial Outline

frenetic

Tutorial Outline

Part I: Ox

- OpenFlow Overview
- Ox Applications

Part II: Frenetic

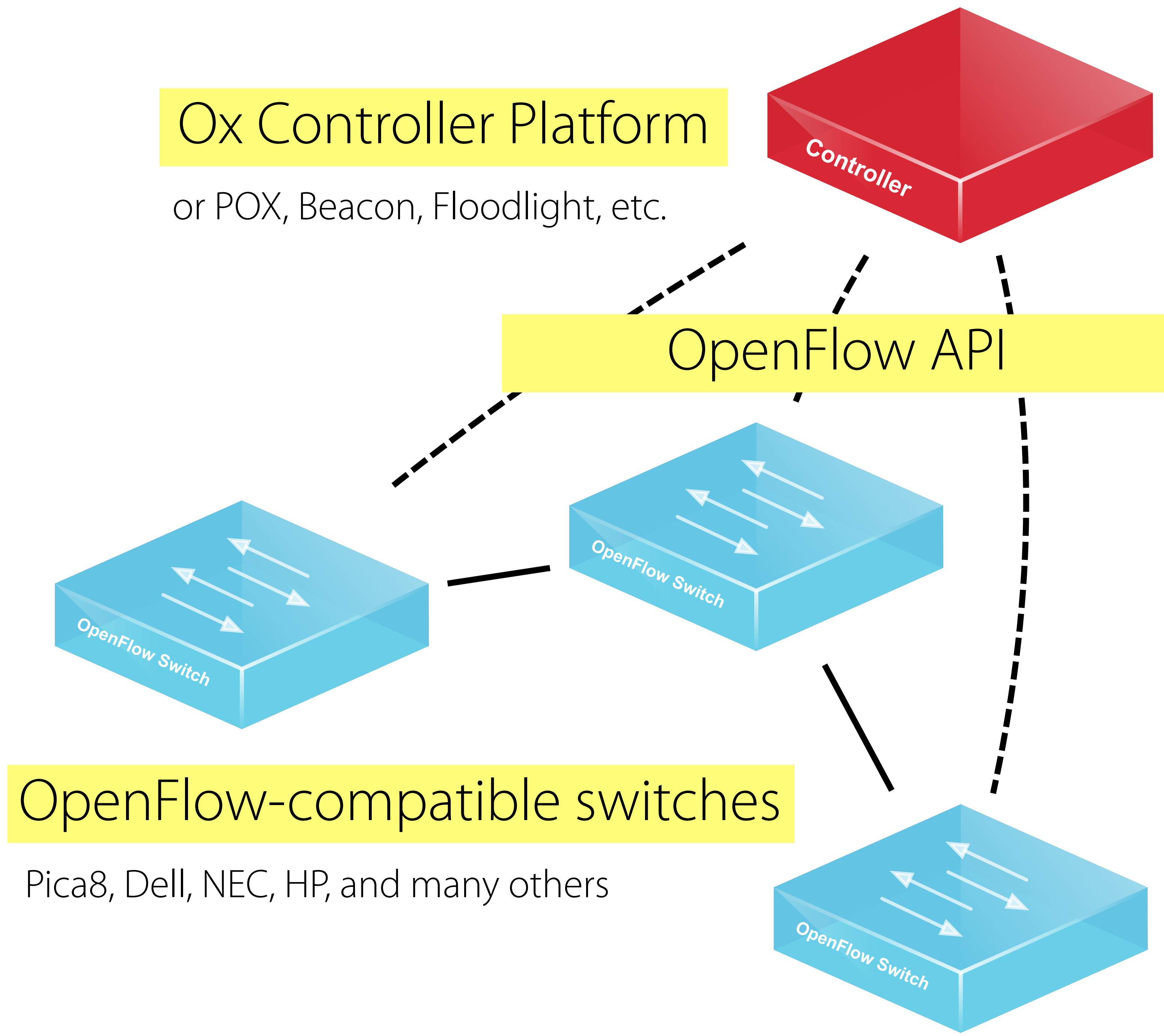
- Frenetic Overview
- Frenetic Applications

Part III: Formal methods

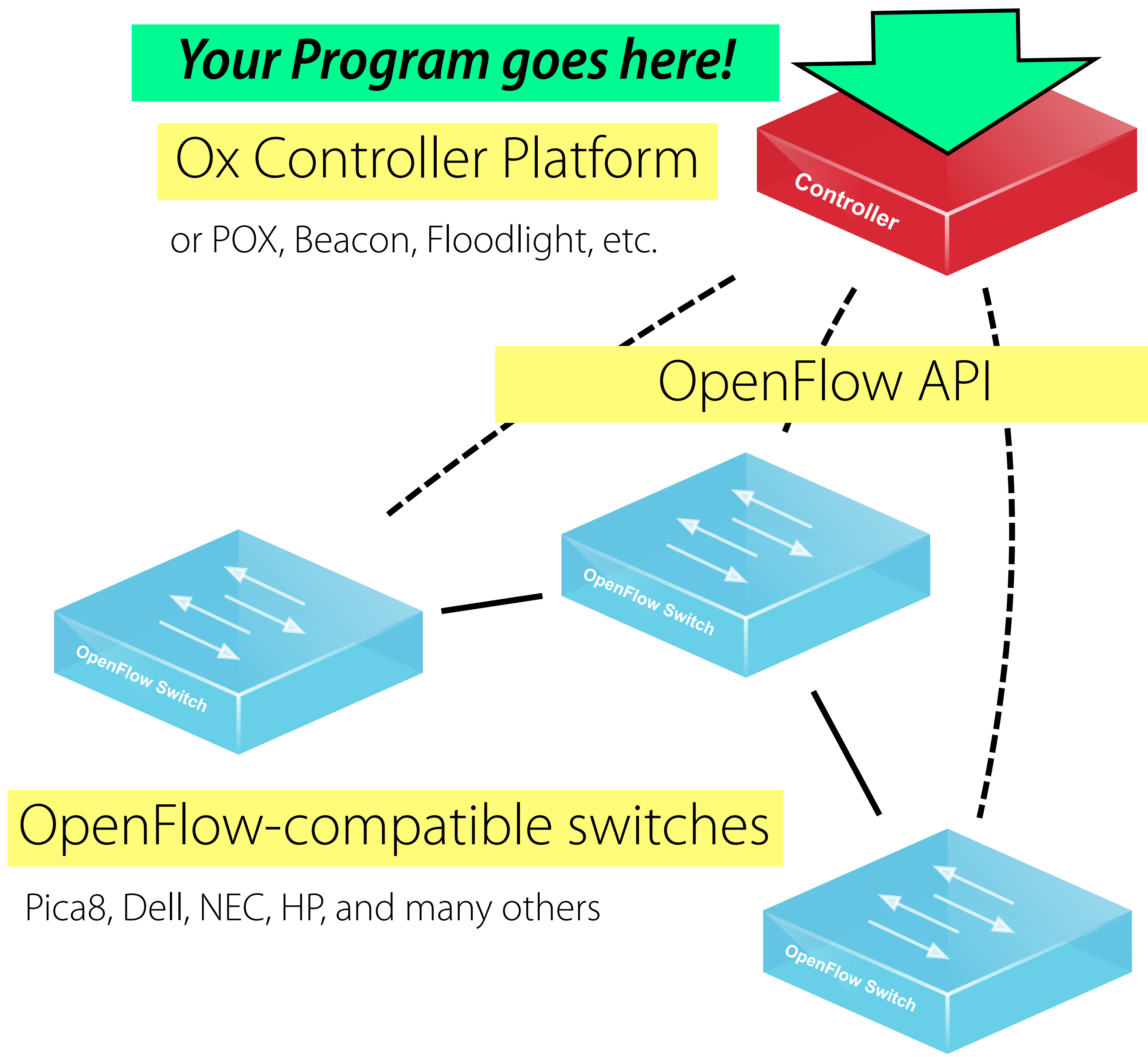
- Update consistency
- Verification and reasoning

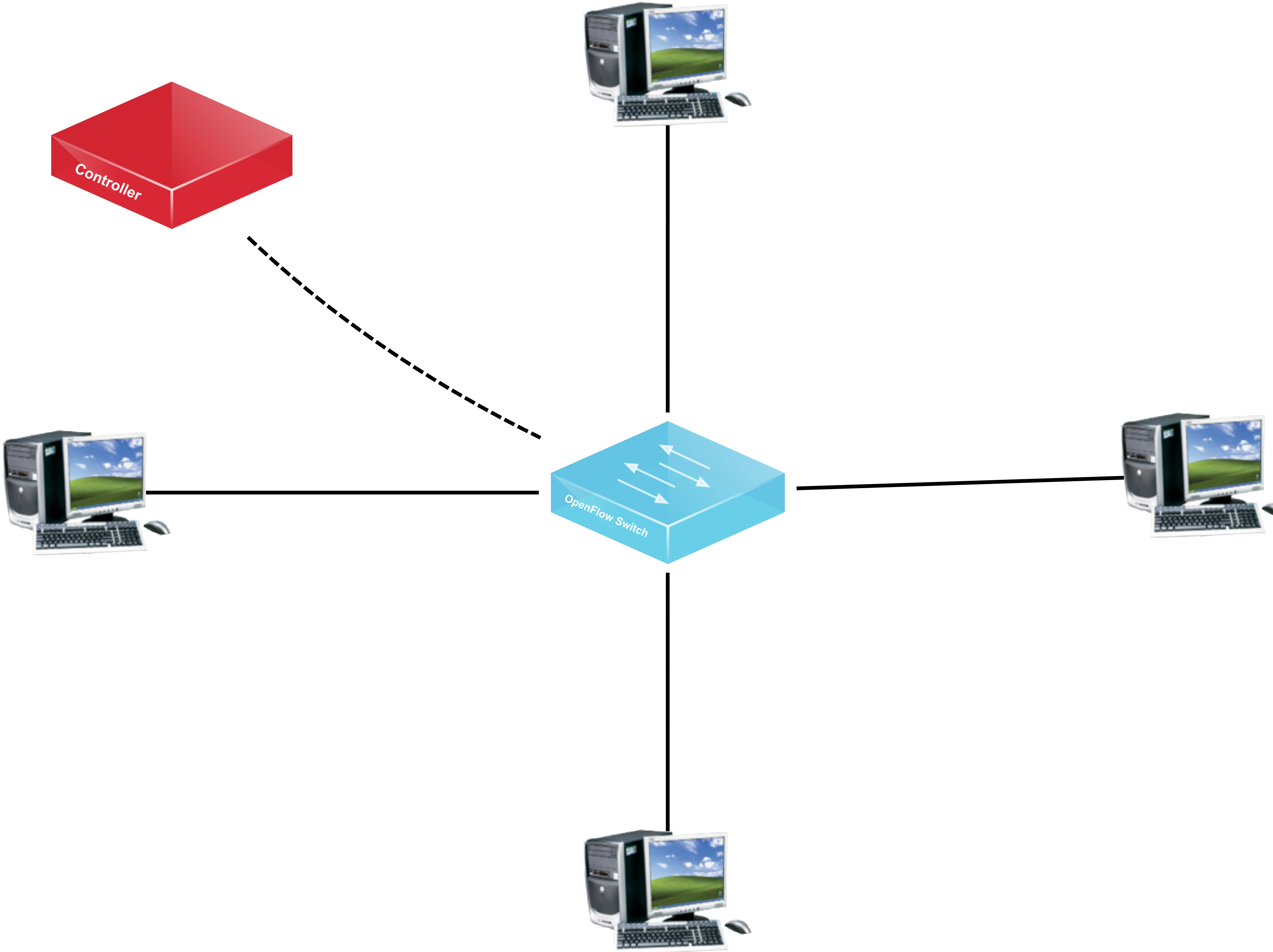
OpenFlow Overview

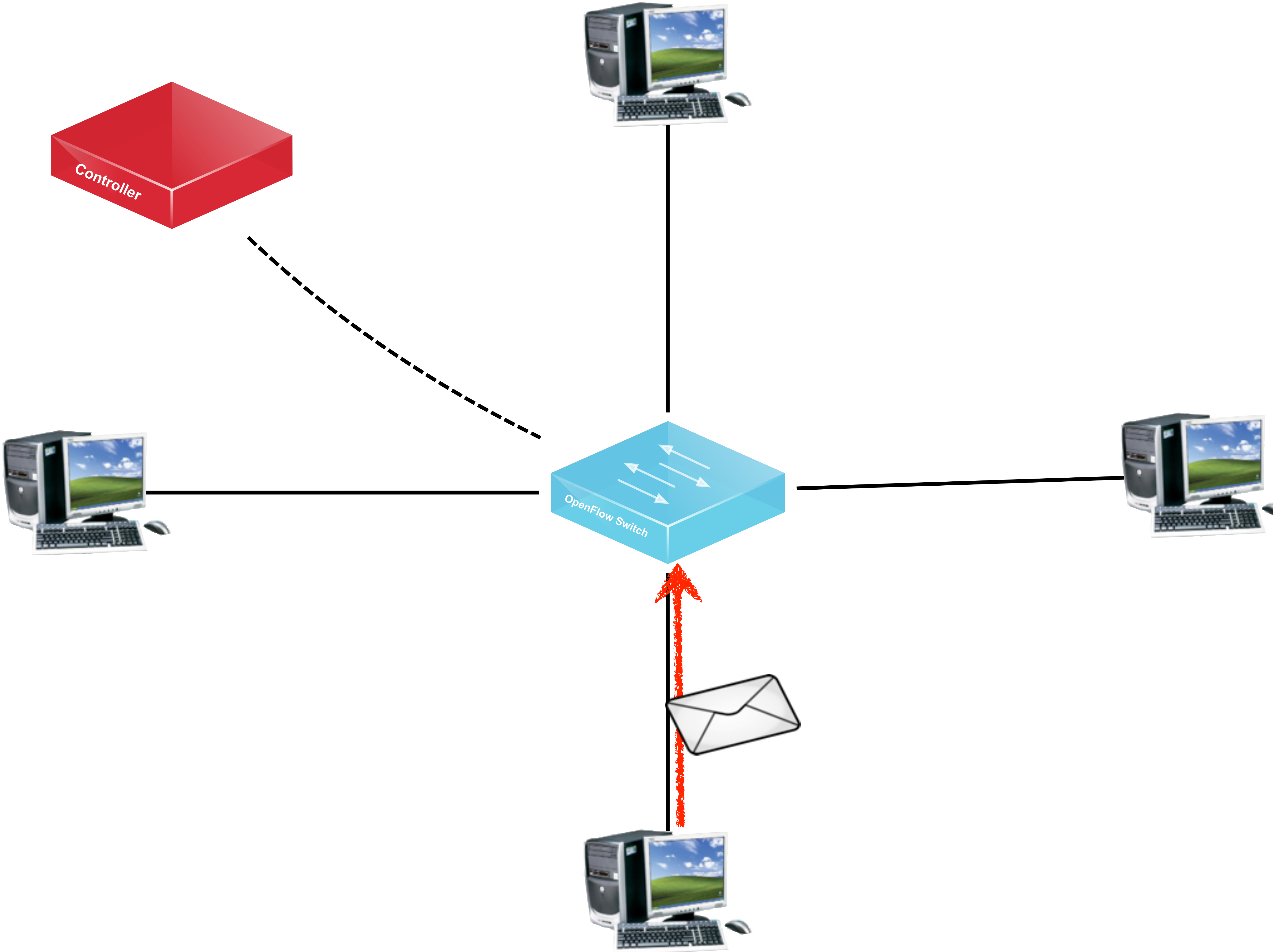
OpenFlow Architecture

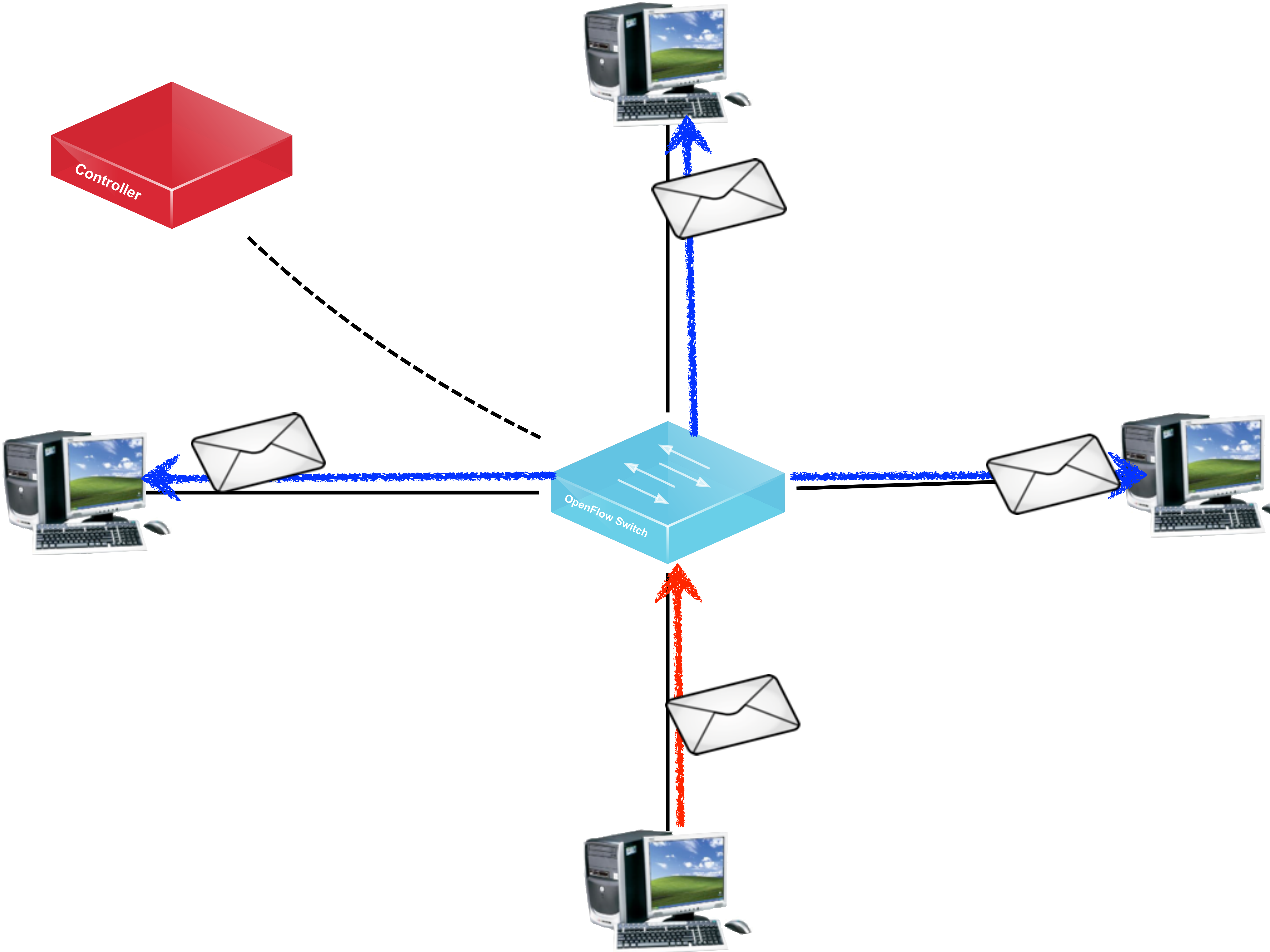


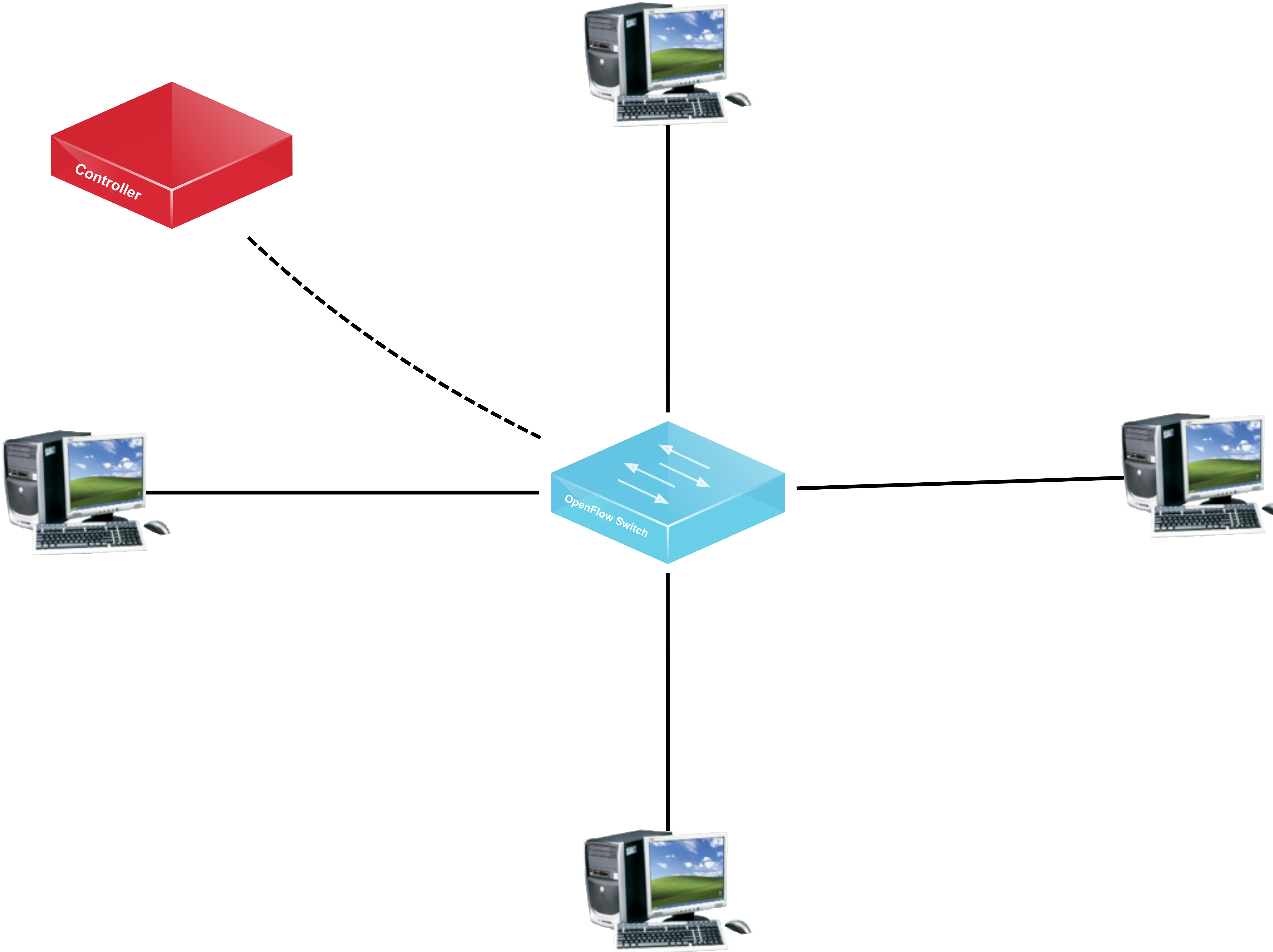
OpenFlow Architecture

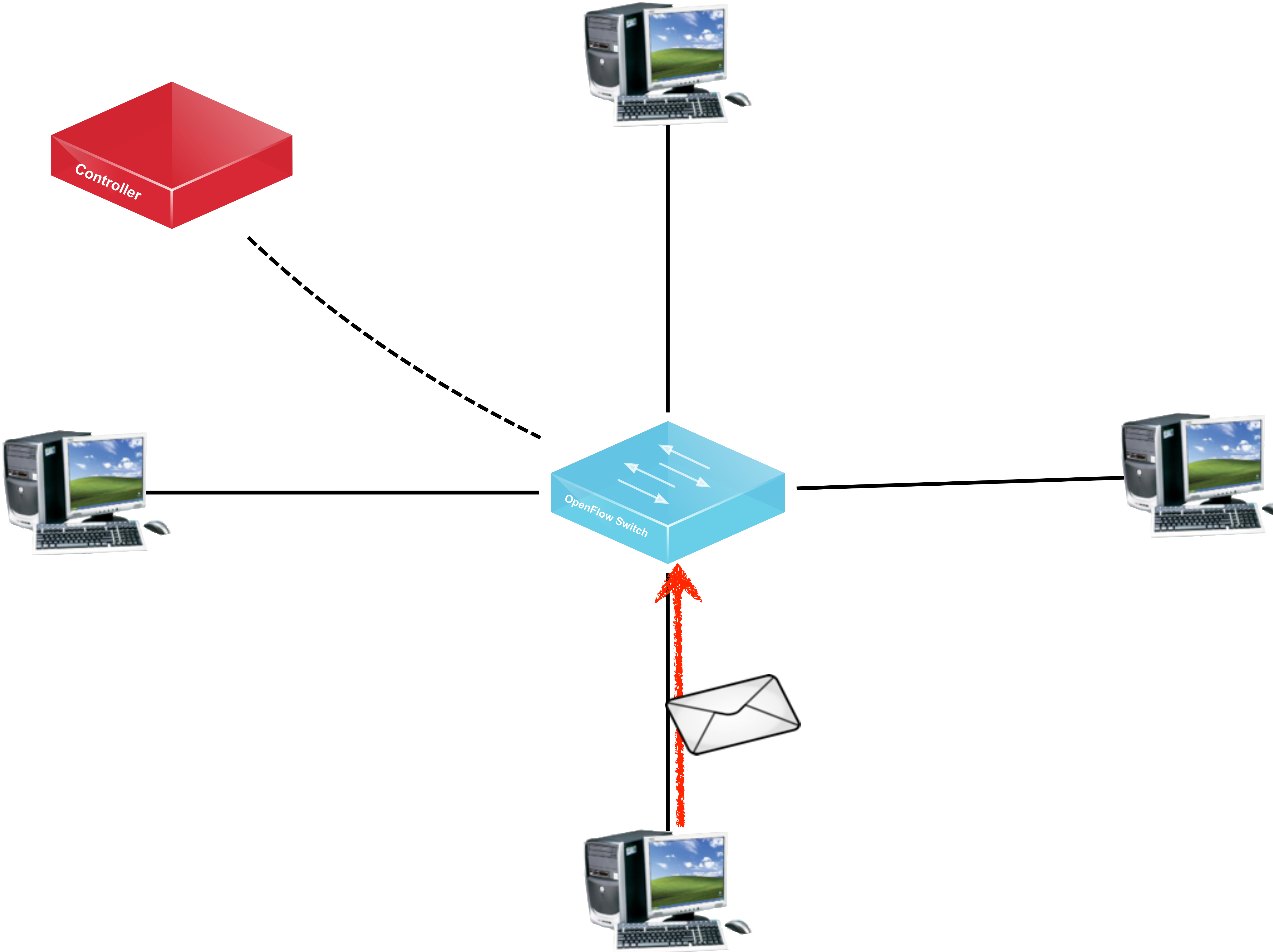


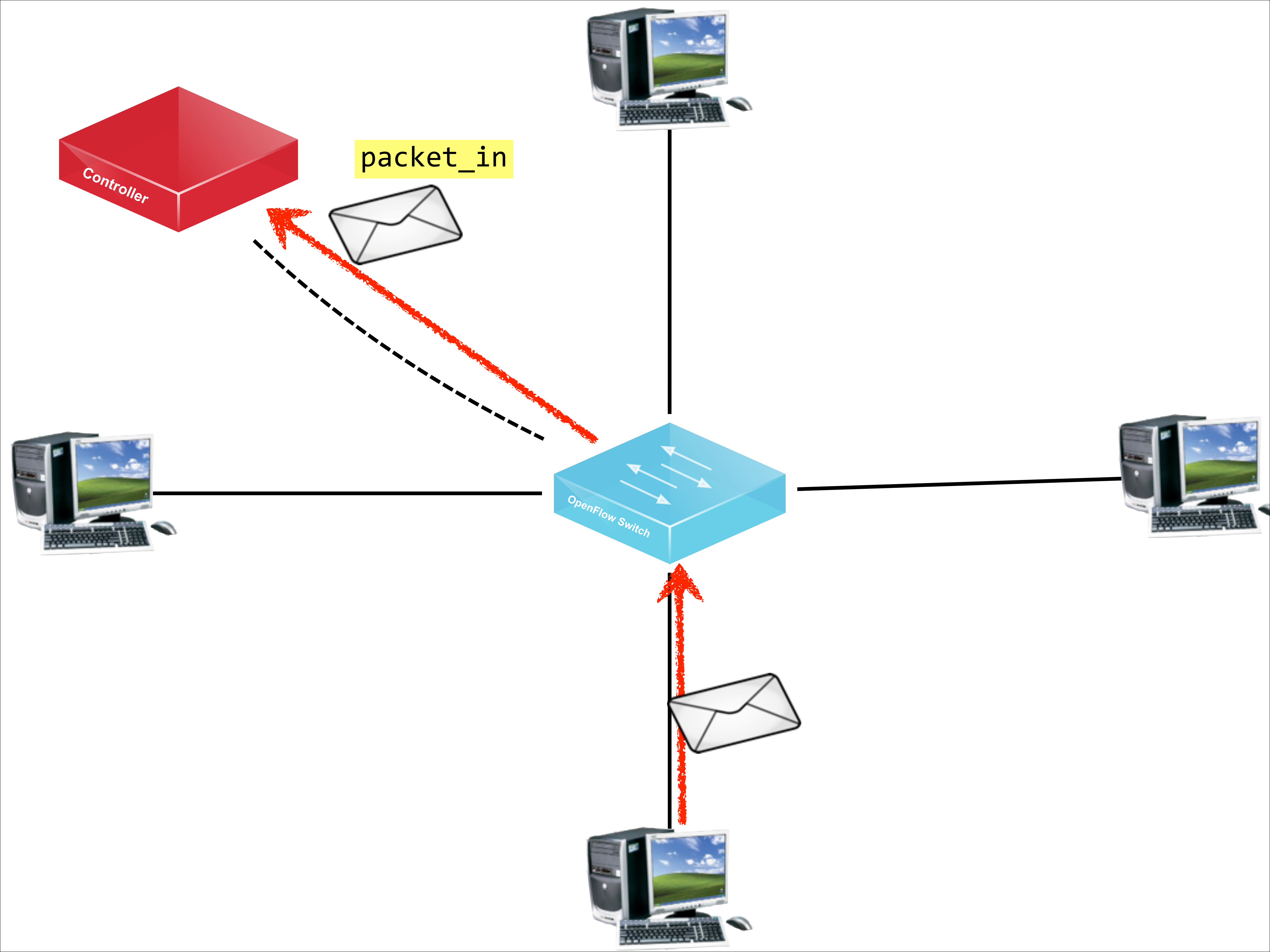

Controller
OpenFlow Switch


Controller
OpenFlow Switch

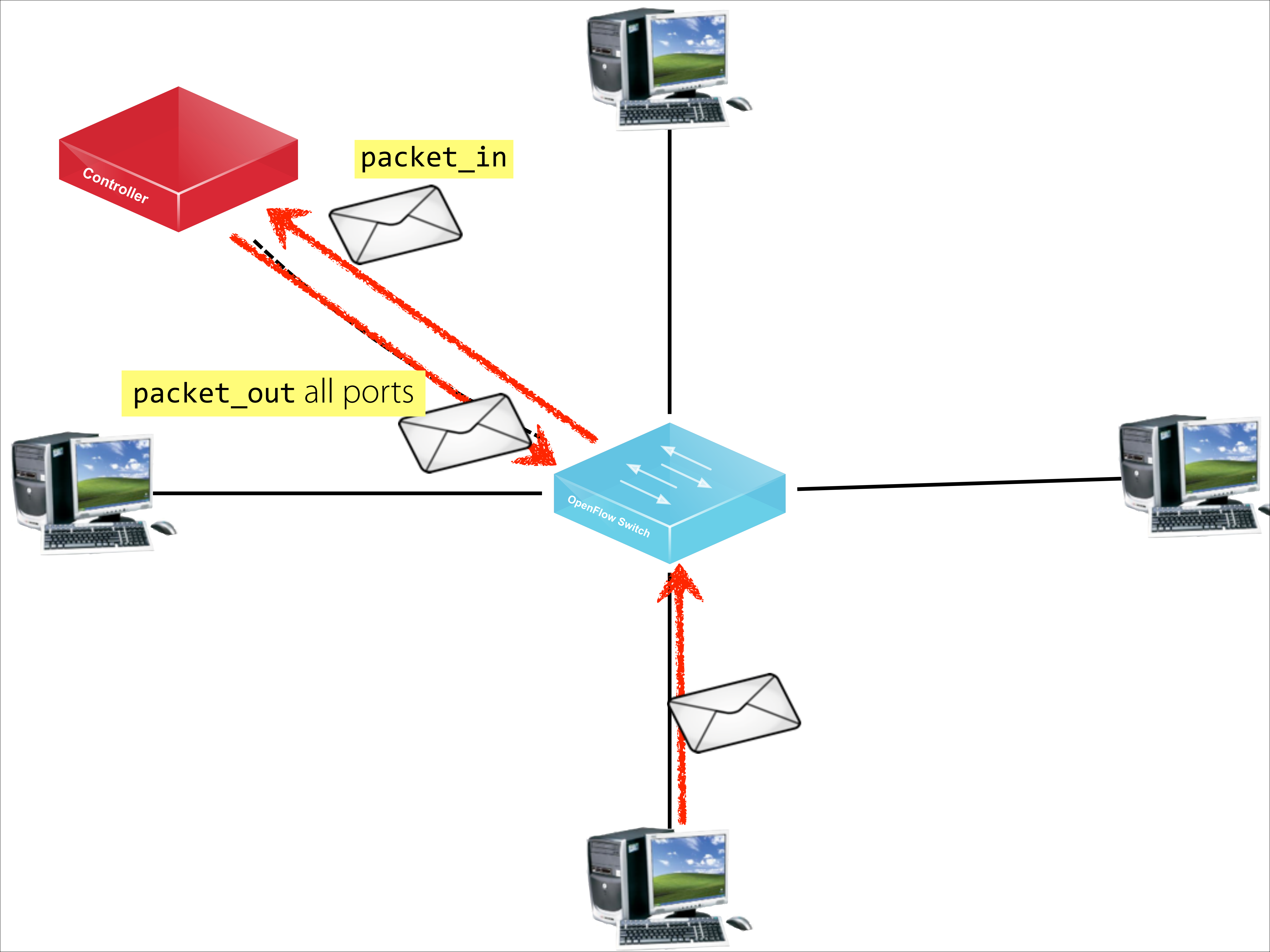

Controller
OpenFlow Switch


Controller
OpenFlow Switch

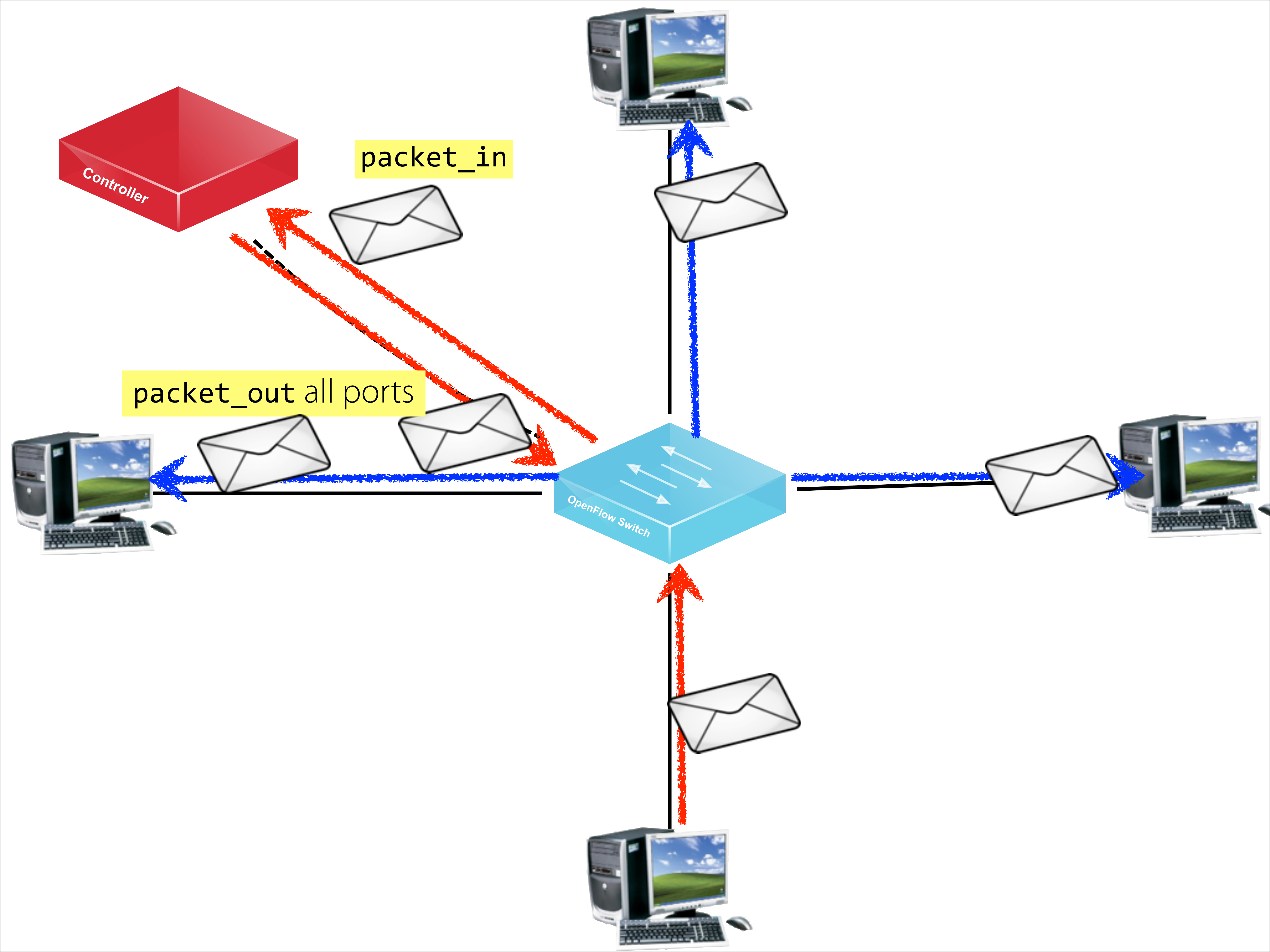

Controller
OpenFlow Switch



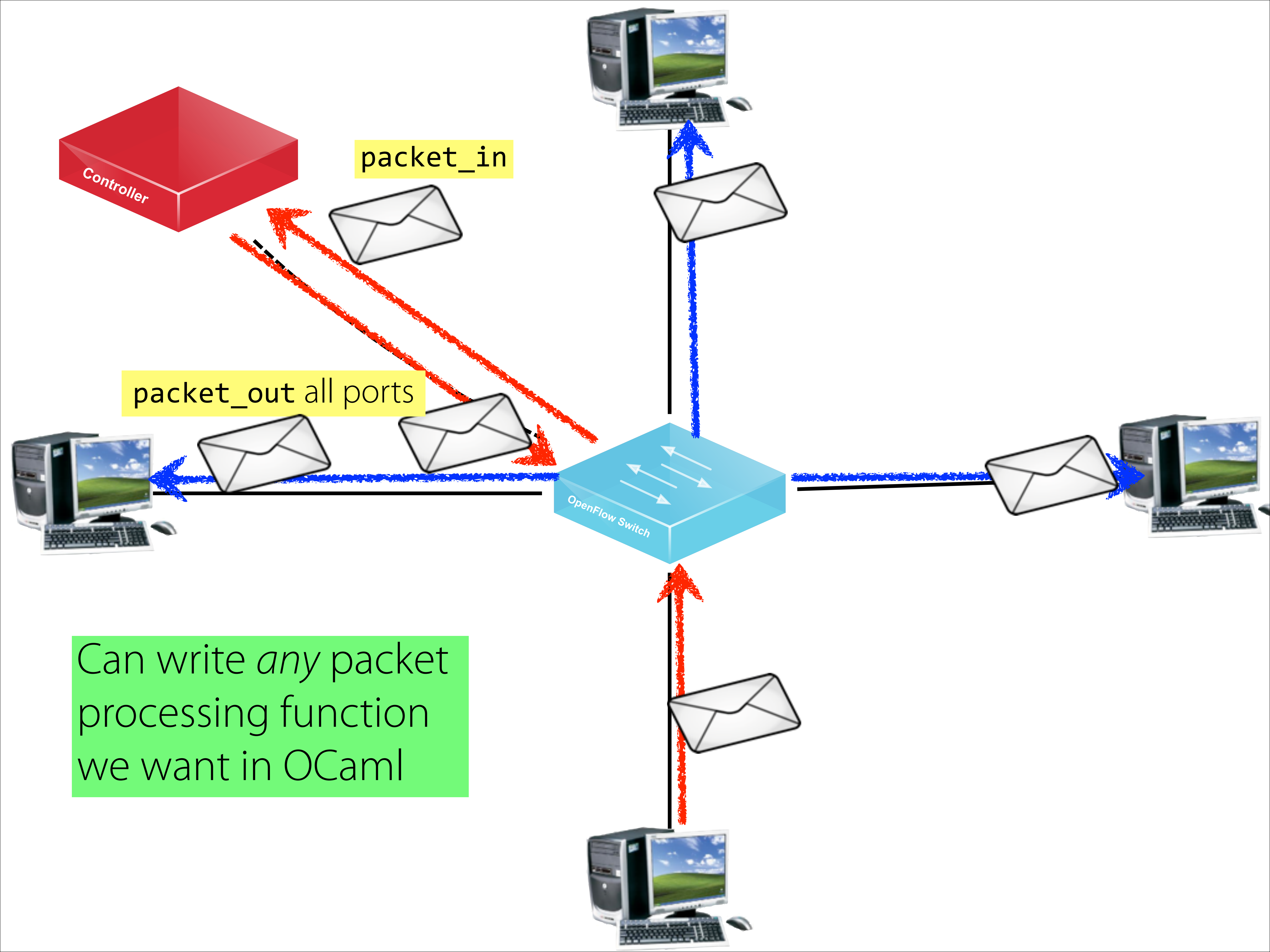
Controller
packet_in
OpenFlow Switch

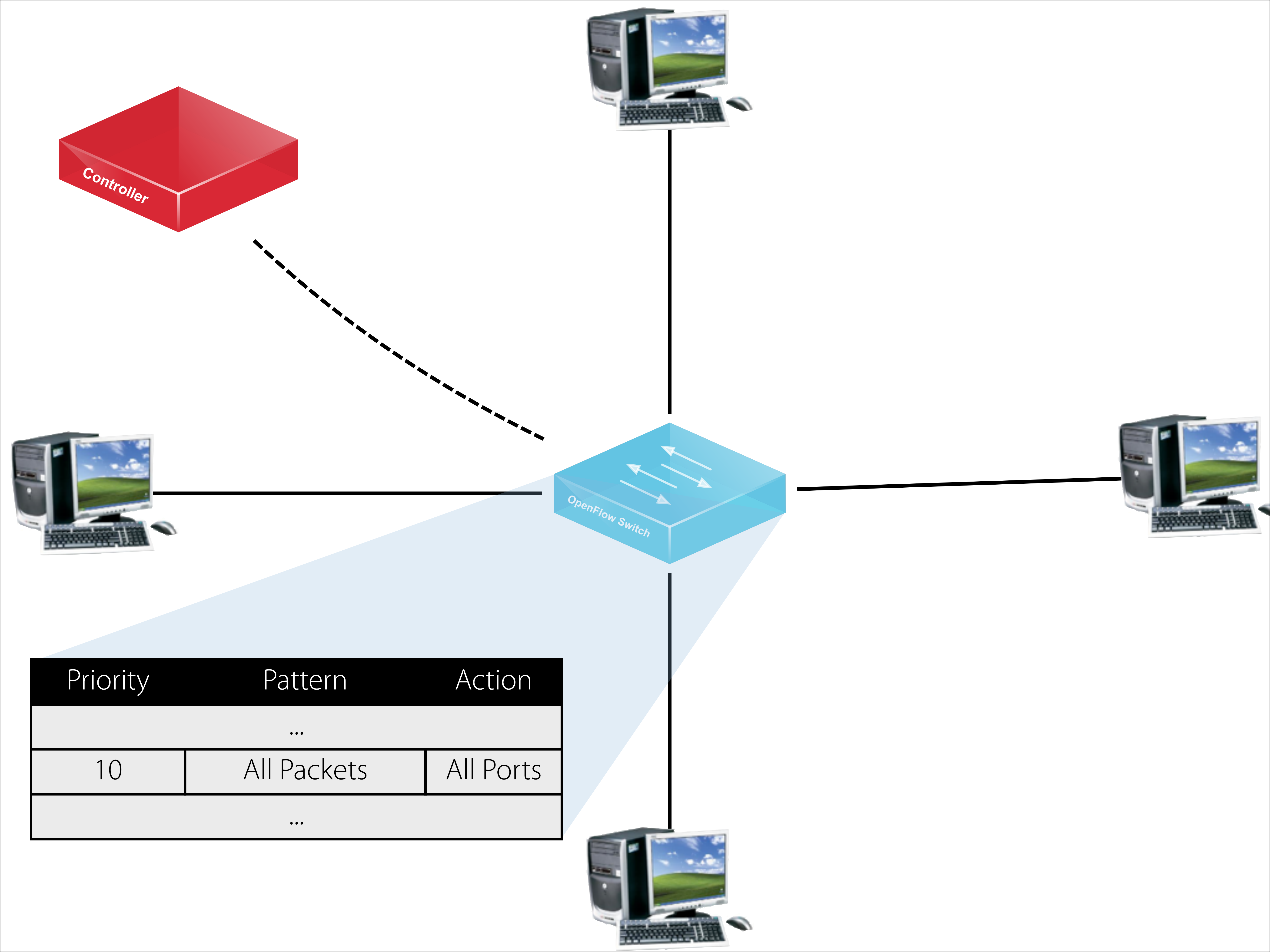


Controller
packet_in
packet_out all ports
OpenFlow Switch

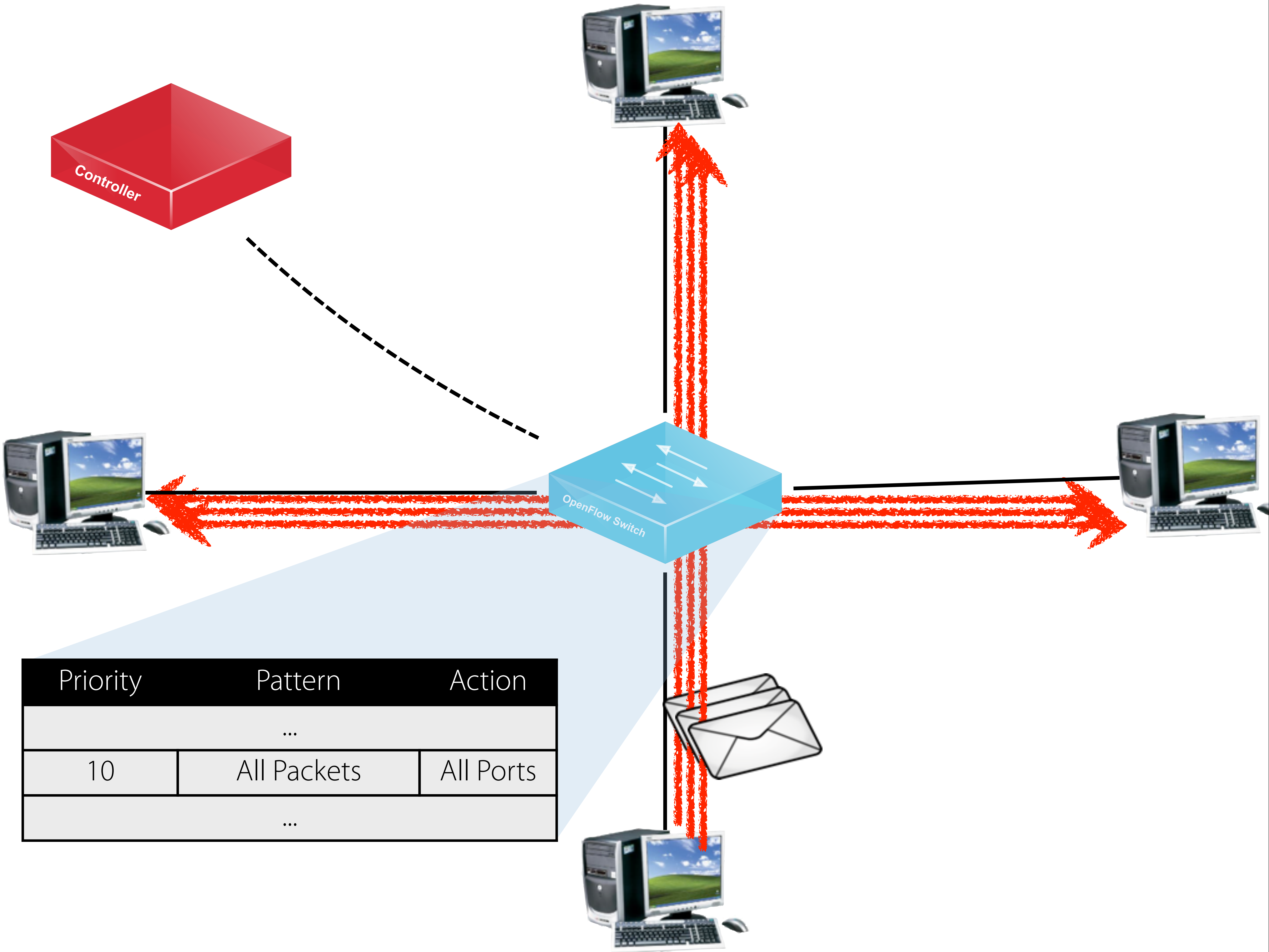


packet_in
Controller
packet_out all ports
OpenFlow Switch


Controller
packet_in
packet_out all ports
OpenFlow Switch
Can write *any* packet processing function we want in OCaml



Priority	Pattern	Action
...		
10	All Packets	All Ports
...		

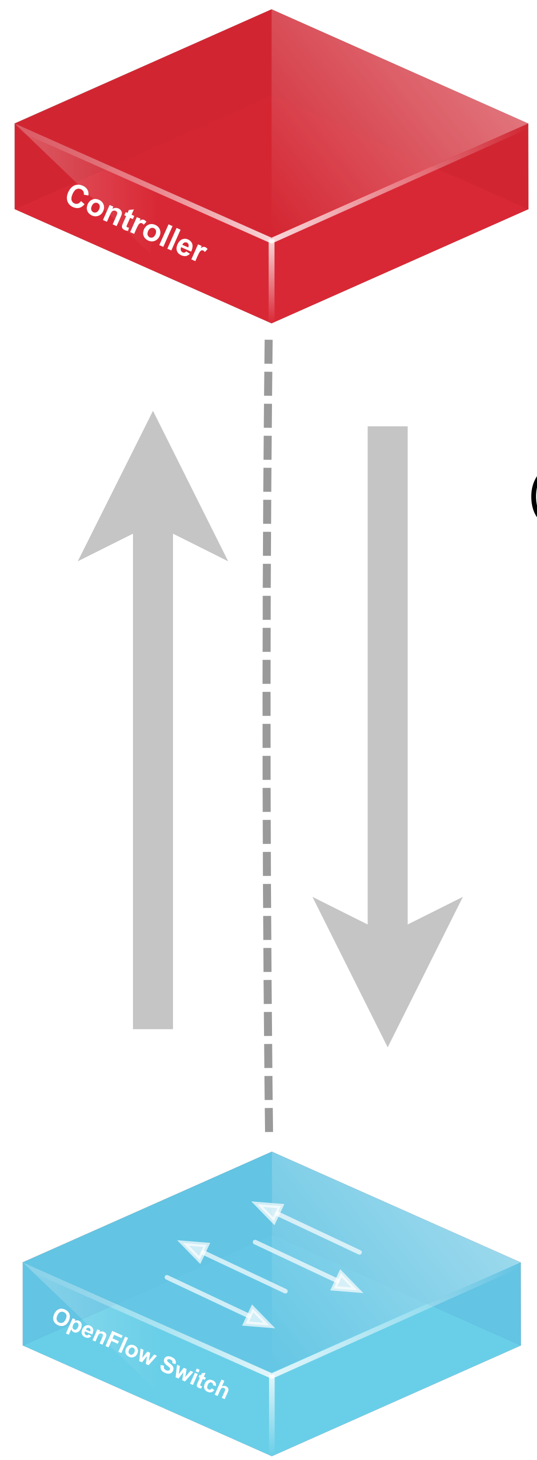


Priority	Pattern	Action
...		
10	All Packets	All Ports
...		

OpenFlow API

Switch to controller:

- switch_connected
- switch_disconnected
- packet_in
- stats_reply



Controller to switch:

- packet_out
- flow_mod
- stats_request

Ox Programming

OpenFlow in Ox

```
open OxPlatform
open OpenFlow0x01_Core

module MyApplication = struct

  include OxStart.DefaultTutorialHandlers

  let switch_connected (sw : switchId) : unit =
    send_flow_mod sw 0l (add_flow prio pat act);
    send_flow_mod sw 0l (add_flow prio pat act);
    send_flow_mod sw 0l (add_flow prio pat act)

  let packet_in (sw : switchId) (xid : xid) (pk : packetIn) : unit =
    send_packet_out sw 0l
      { output_payload = pk.input_payload;
        port_id = None;
        apply_actions = actions
      }
end

module Controller = OxStart.Make (MyApplication)
```

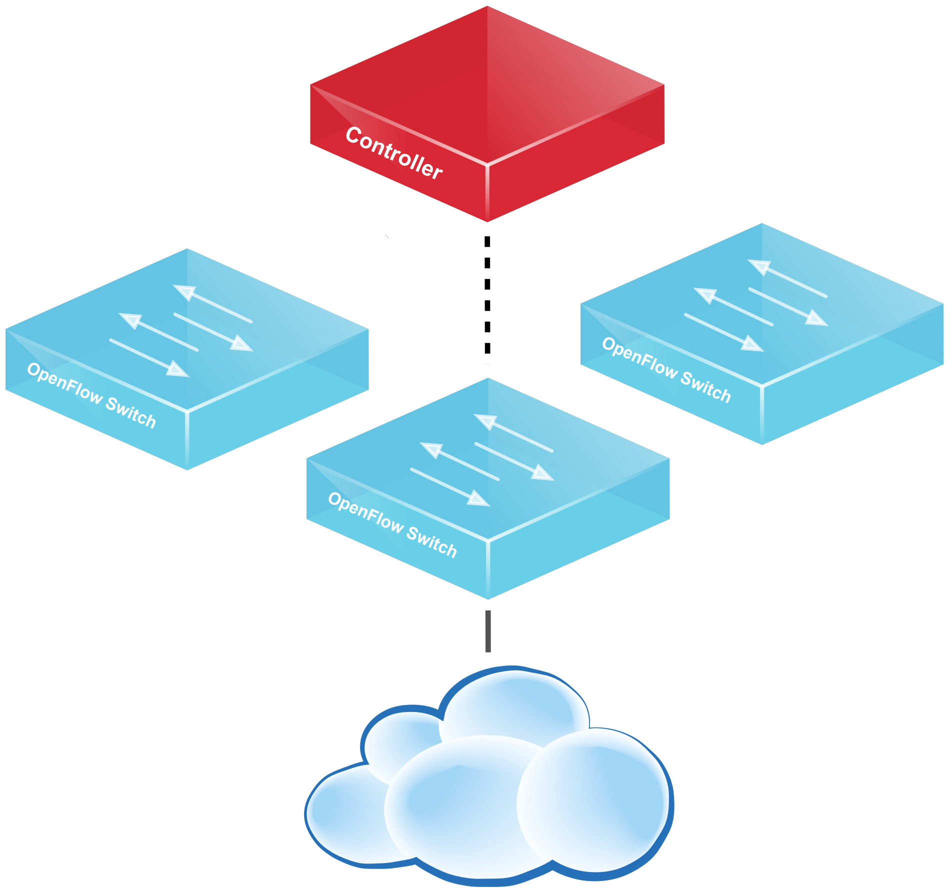
Frenetic Overview

Machine Languages

OpenFlow is a machine language

Programmers must think in terms of low-level concepts such as:

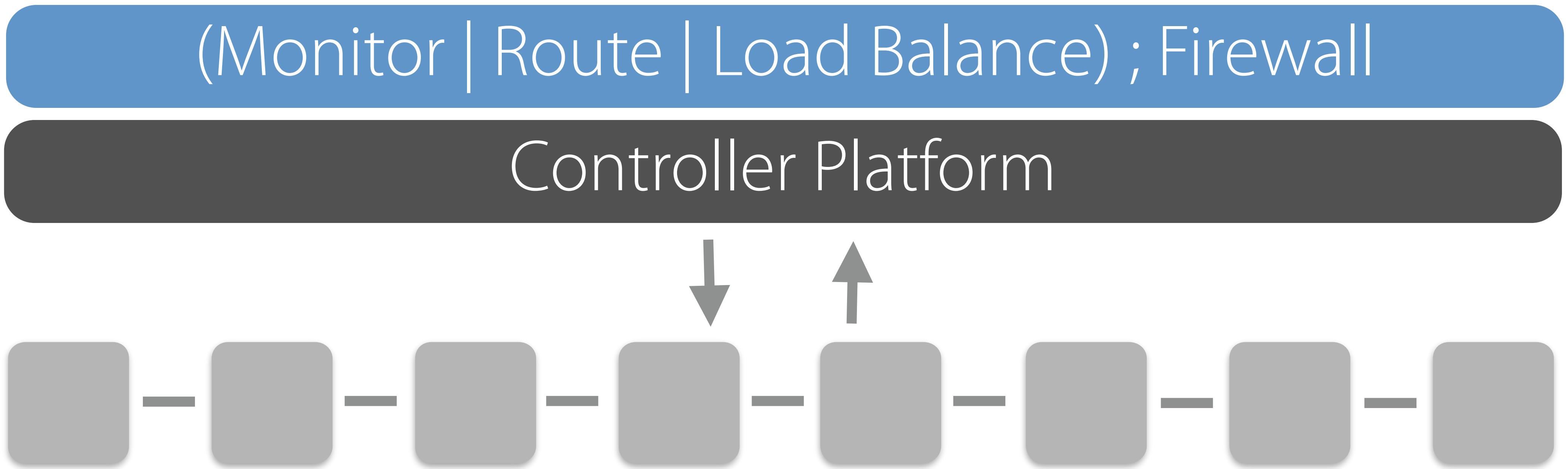
- Flow tables
- Matches
- Priorities
- Timeouts
- Events
- Callbacks



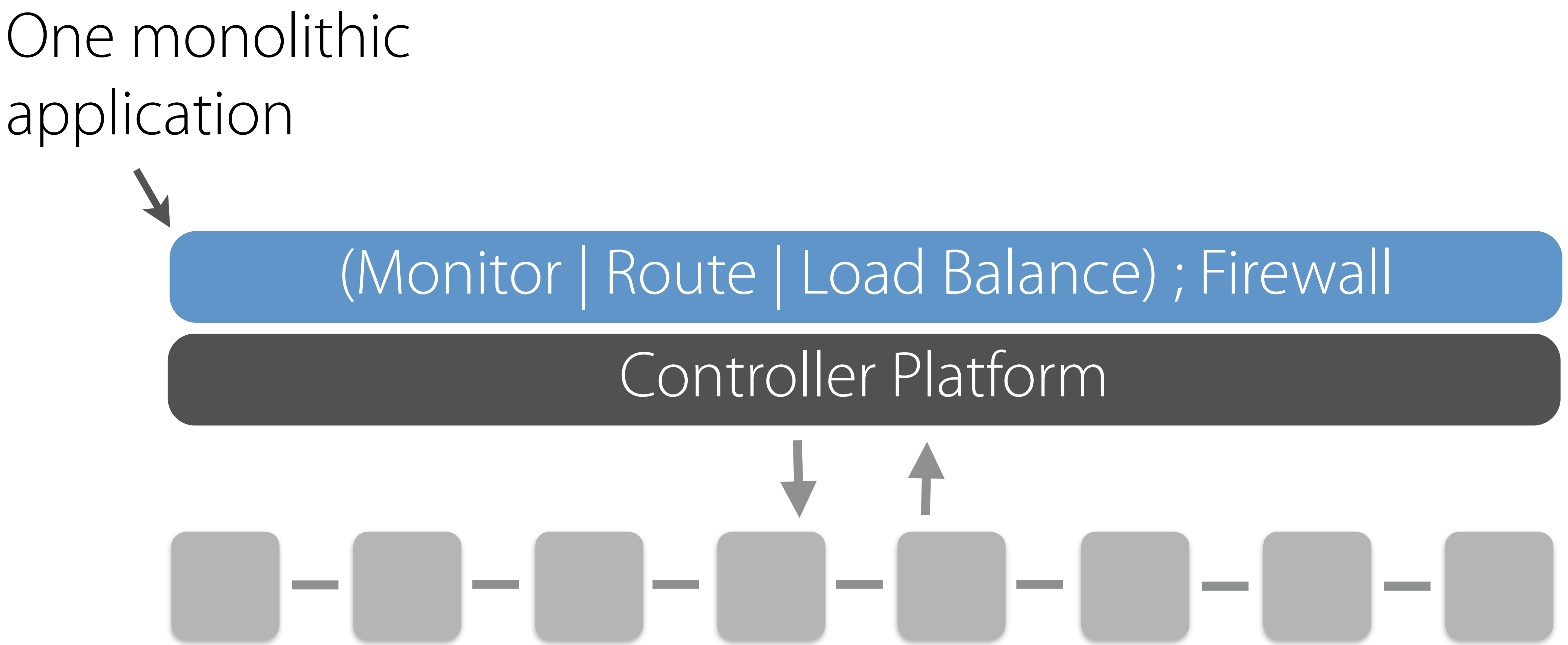
Key issue: programs don't compose!

Can't write `(firewall; routing)`

Current Controllers

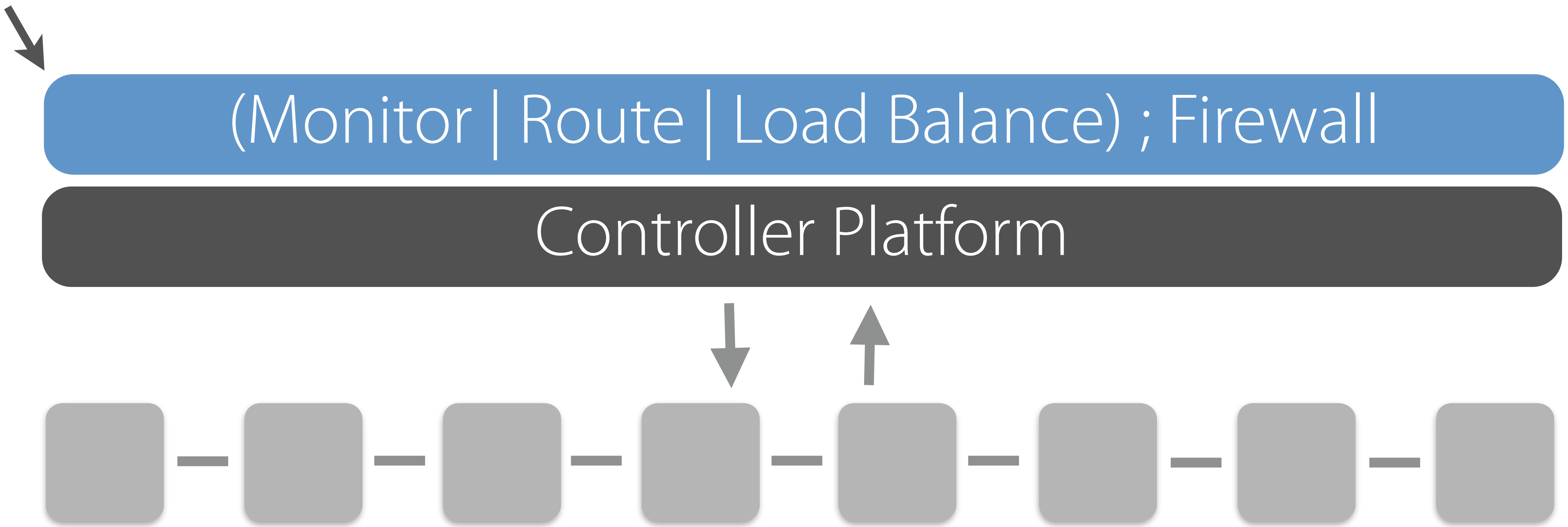


Current Controllers



Current Controllers

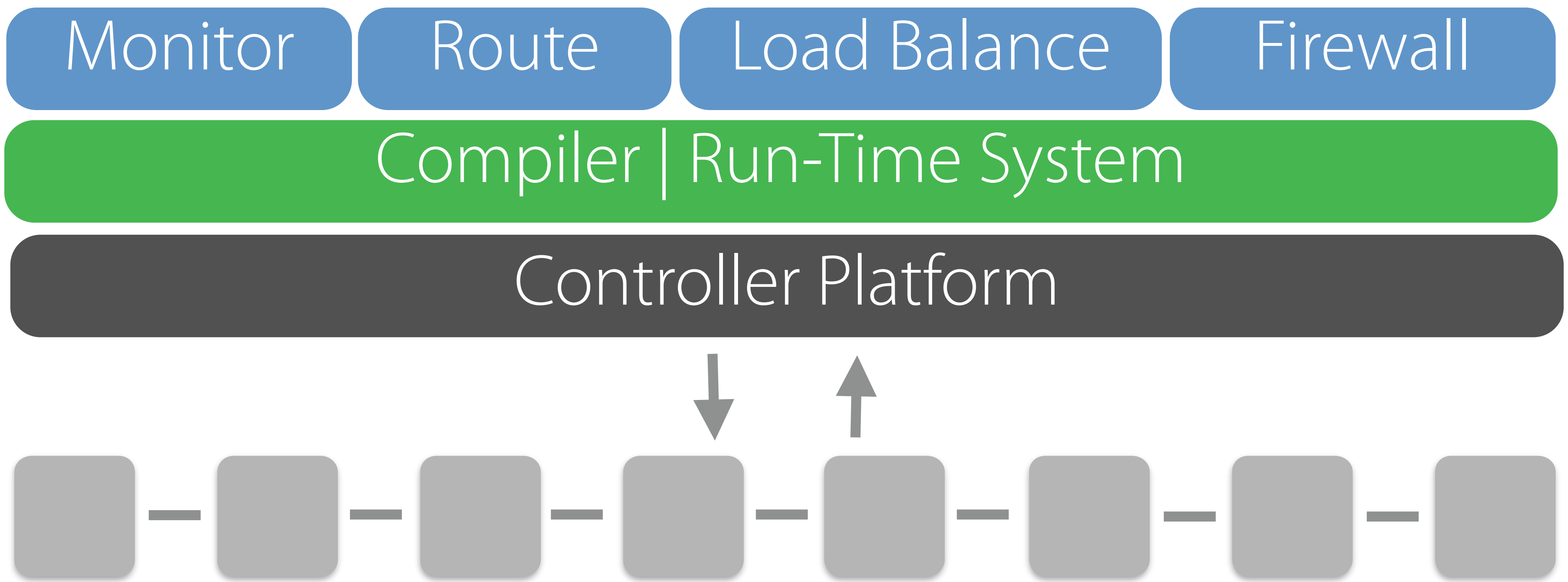
One monolithic application



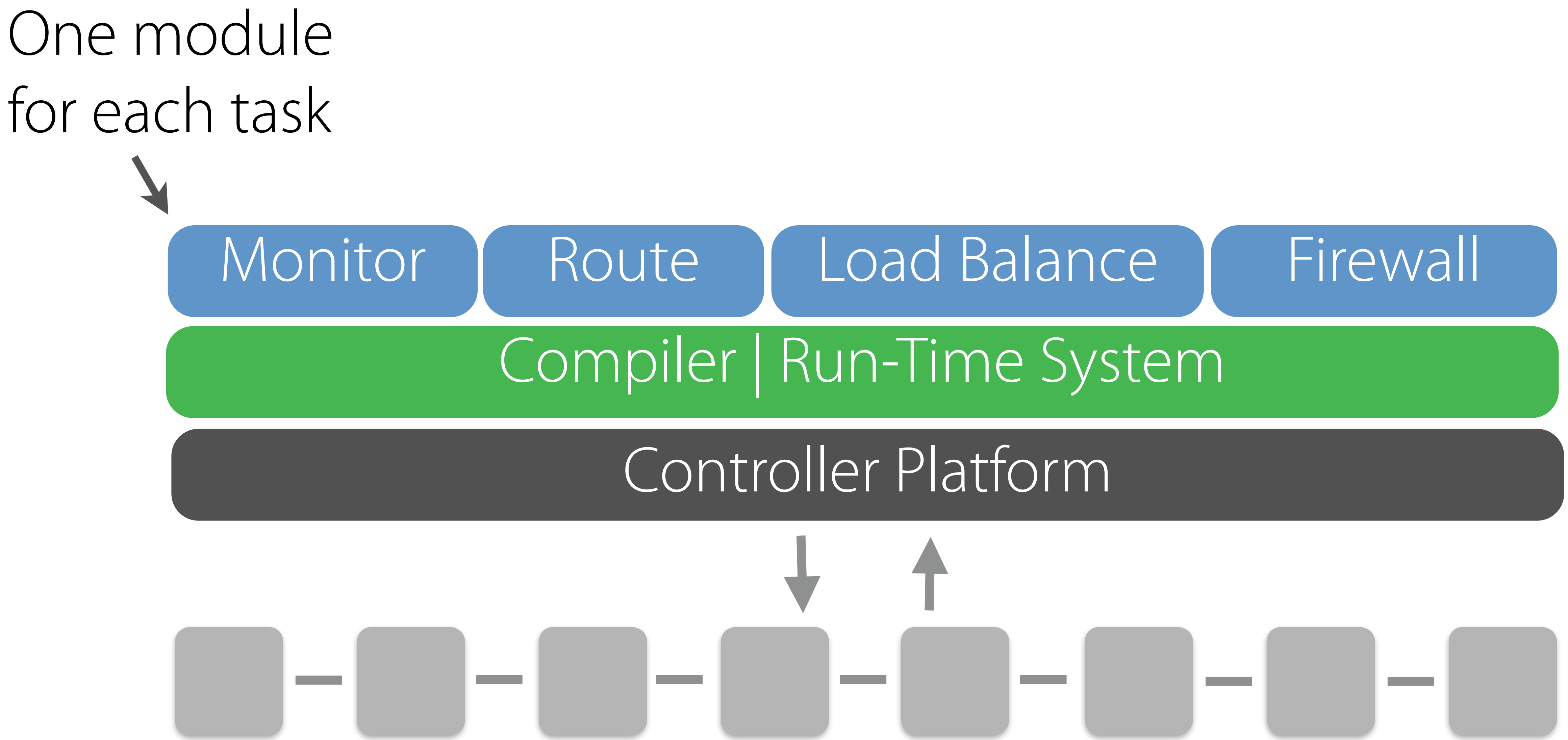
Challenges:

- Writing, testing, and debugging programs
- Reusing code across applications
- Porting applications to new platforms

Language-Based Approach

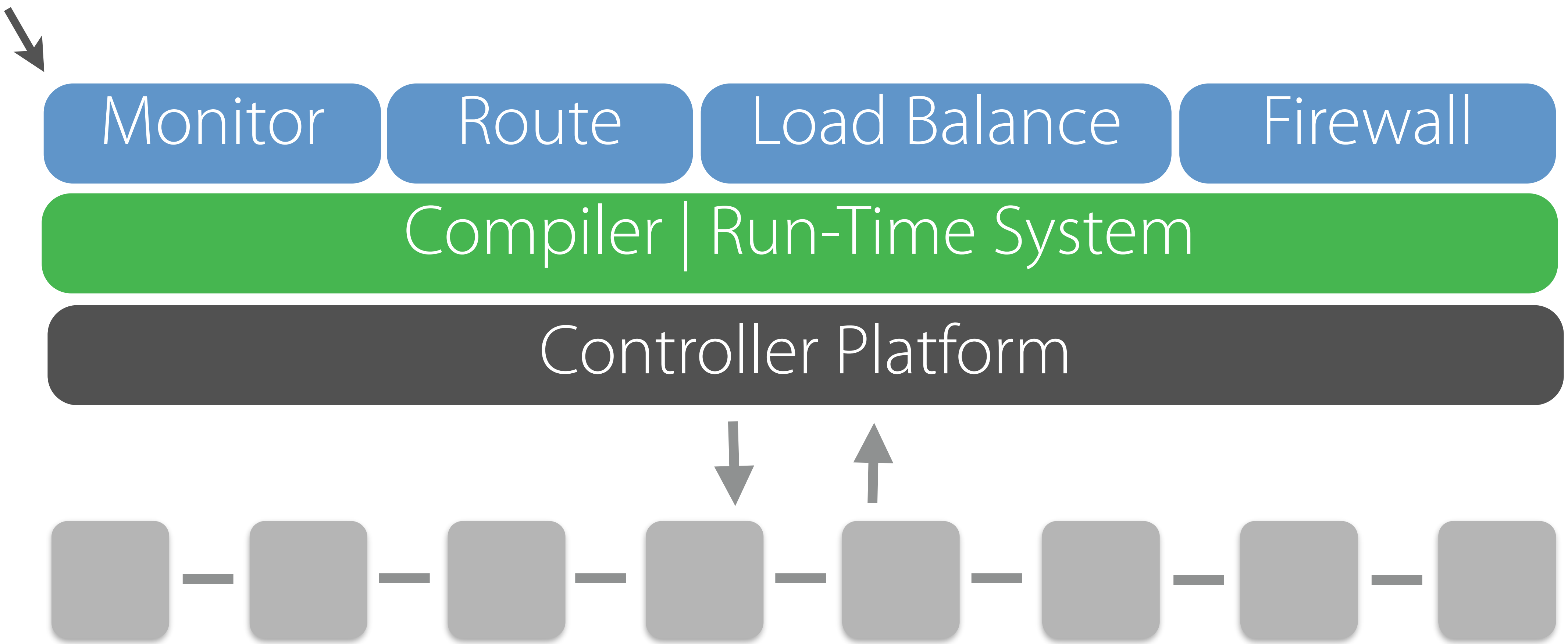


Language-Based Approach



Language-Based Approach

One module
for each task



Benefits:

- Easier to write, test, and debug programs
- Can reuse modules across applications
- Possible to port applications to new platforms

Programming Languages

Frenetic is a programming language

Programmers work in terms of natural constructs:

- Functions
- Predicates
- Relational operators
- Logical properties

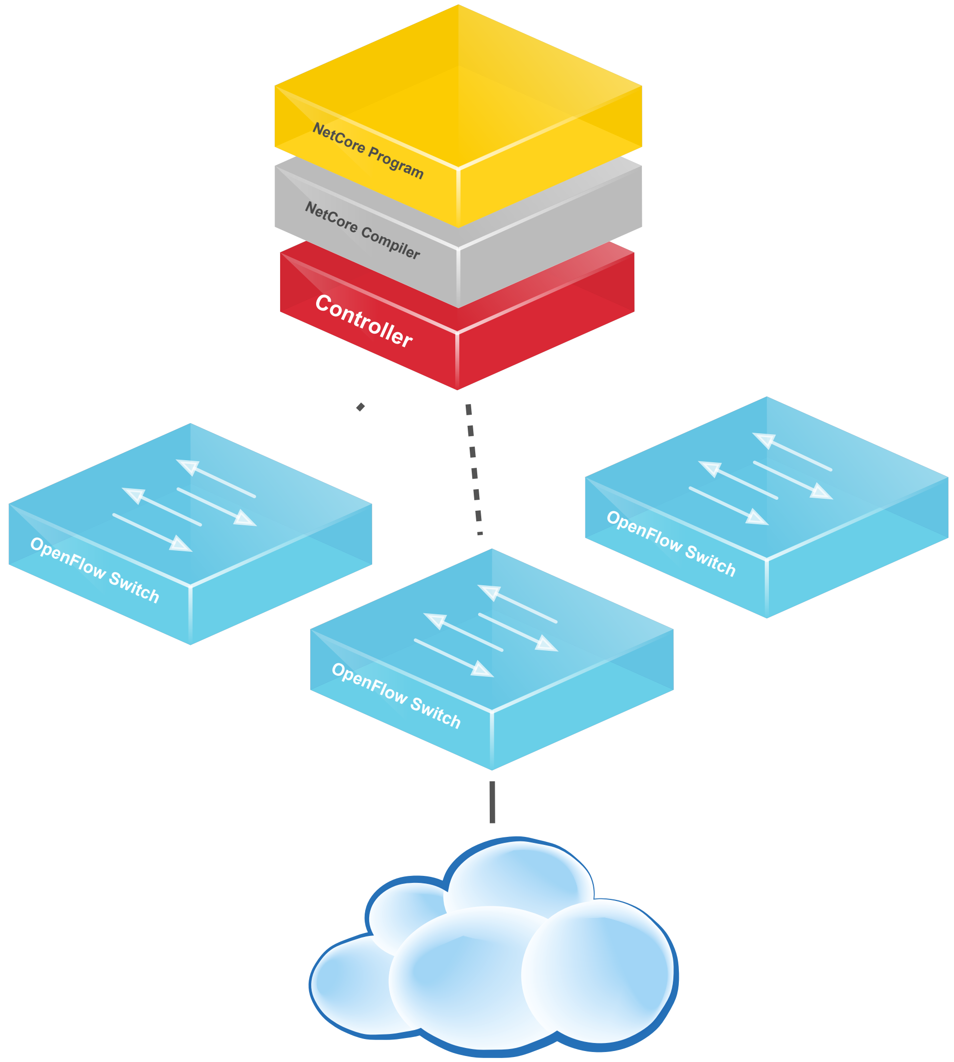
Compiler bridges the gap between these abstractions and their implementations in OpenFlow

Programming Languages

Frenetic is a programming language

Programmers work in terms of natural constructs:

- Functions
- Predicates
- Relational operators
- Logical properties



Compiler bridges the gap between these abstractions and their implementations in OpenFlow

Predicates

Frenetic *predicates* denotes sets of *located packets*

Syntax

```
• switch = s          (* switch location *)
• inPort = n          (* port location *)
• field = v           (* field value *)
• pred1 && pred2      (* conjunction *)
• pred1 || pred2      (* disjunction *)
• !pred               (* negation *)
```

Examples

```
switch = 0l && !(inPort = 1)
dlSrc = 00:00:00:00:00:01
dlTyp = ip && nwProto = icmp
```

Policies

Frenetic *policies* denote functions from located packets to sets of located packets

Syntax

```
• fwd(n)                          (* forward *)
• all                             (* flood *)
• pass                            (* identity *)
• drop                            (* drop *)
• field v -> v                    (* modify *)
• if pred then pol1 else pol2     (* conditional *)
• let x = pol1 in pol2            (* variable binding *)
• pol1 + pol2                     (* parallel composition *)
• pol1 ; pol2                     (* sequential composition *)
```

Examples

```
if switch = 01 then all else drop
firewall; (monitor + route)
```

Frenetic Programming

The Frenetic System

The full Frenetic system contains a number of additional features including:

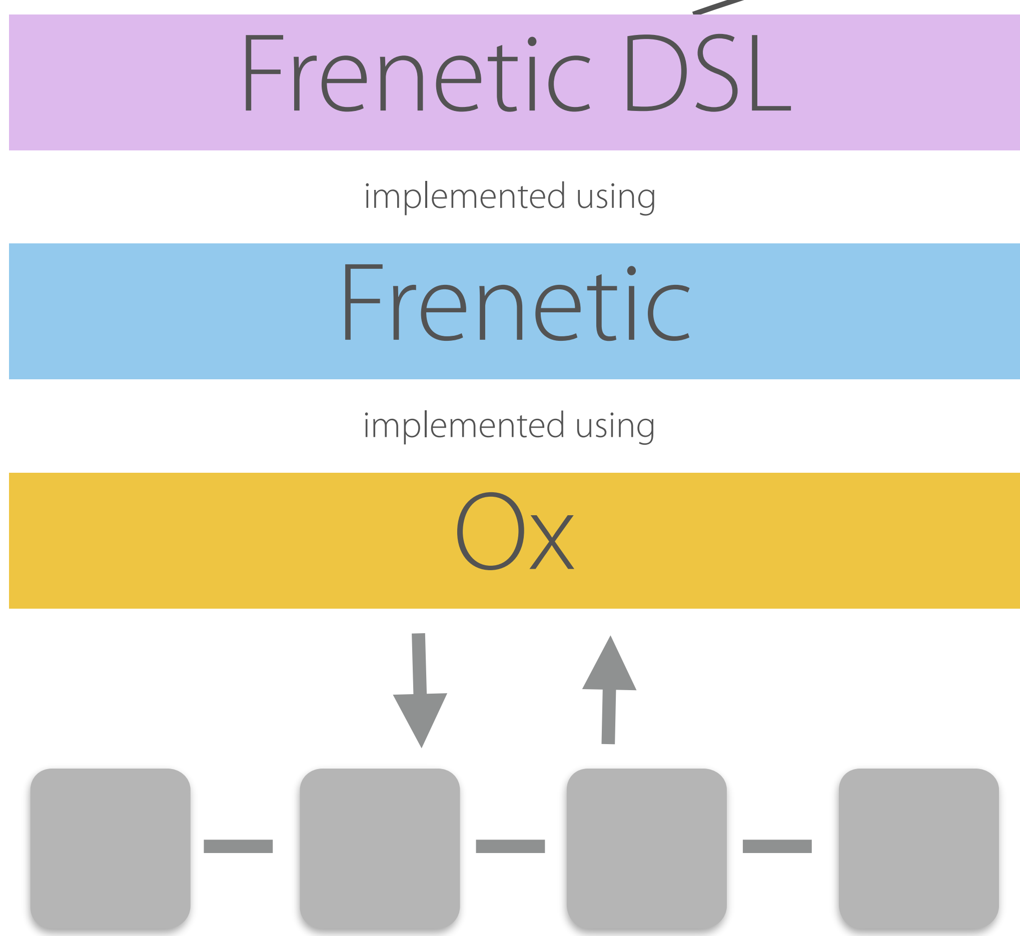
- Functions for transforming packets
- Stateful applications (learning, NAT, firewall)
- Integrated testing and debugging

Example

```
let my_nat =
  let priv, pub = nat(publicIP = 10.0.0.254) in
    if switch = 1 && inPort = 1 then (priv; fwd(2))
    else if switch = 1 && inPort = 2 then (pub; fwd(1))

monitorPolicy(learn; my_nat)
```

The frenetic System



Domain-specific language

- predicates and policies
- monitoring
- mac learning
- network address translation

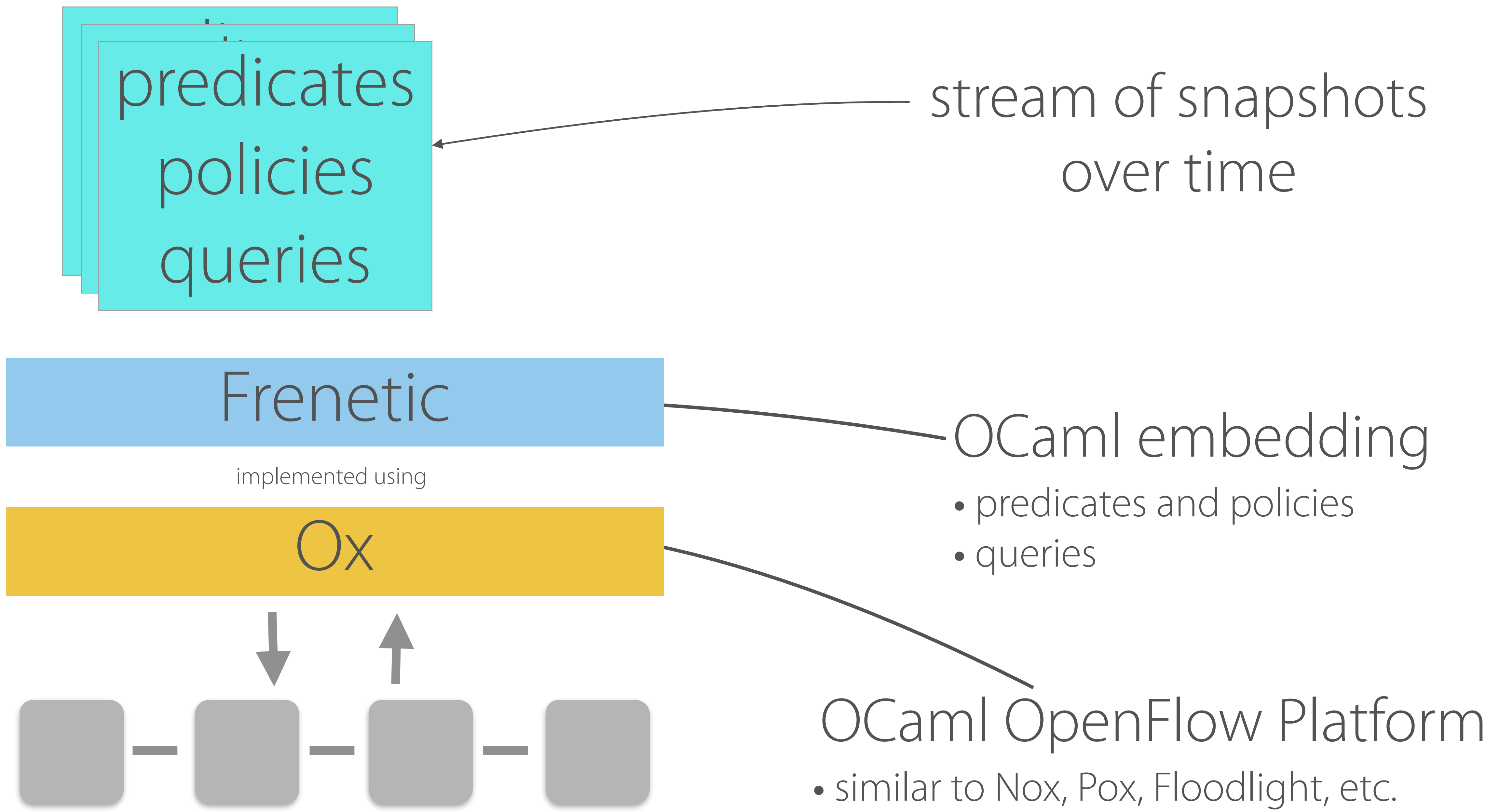
OCaml embedding

- predicates and policies
- queries

OCaml OpenFlow Platform

- similar to Nox, Pox, Floodlight, etc.

The *frenetic* System



Frenetic in OCaml

Can also implement dynamic and stateful programs in Frenetic

Use **`CStruct`** and **`Lwt`** libraries internally

```
type pred =
  | Hdr of ptrn
  | OnSwitch of switchId
  | Or of pred * pred
  | And of pred * pred
  | Not of pred
  | Everything
  | Nothing

type switchEvent =
  | SwitchUp of switchId * SwitchFeatures.t
  | SwitchDown of switchId

type pol =
  | HandleSwitchEvent of (switchEvent -> unit)
  | Action of action
  | Filter of pred
  | Union of pol * pol
  | Seq of pol * pol
  | ITE of pred * pol * pol

  ...
```

Frenetic @ Home

TP-Link TL-WR1043ND

$50

Open firmware:

`www.dd-wrt.com`

Frenetic @ Home

TP-Link TL-WR1043ND

$50

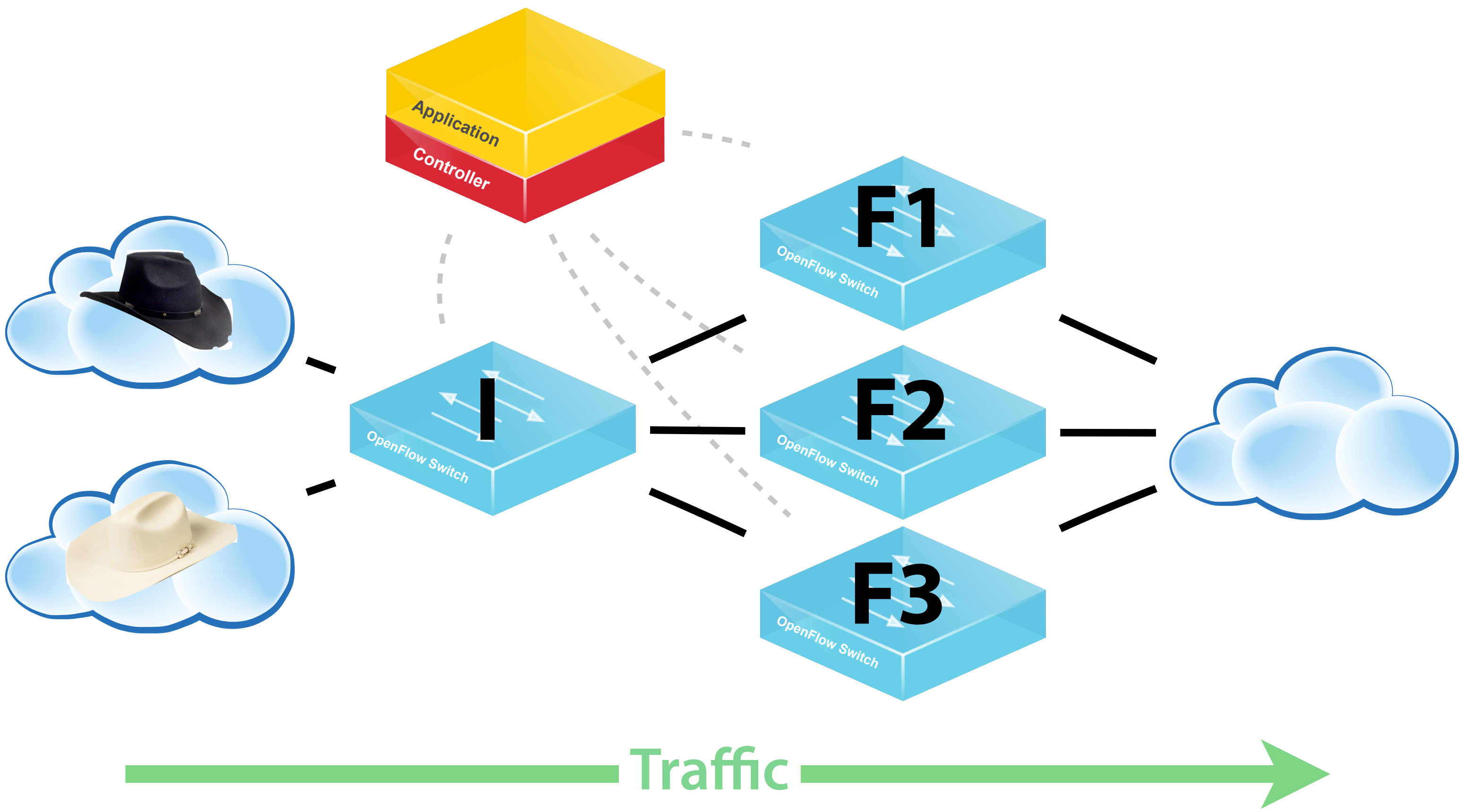
Open firmware:

`www.dd-wrt.com`

Consistent Updates

[SIGCOMM '12]

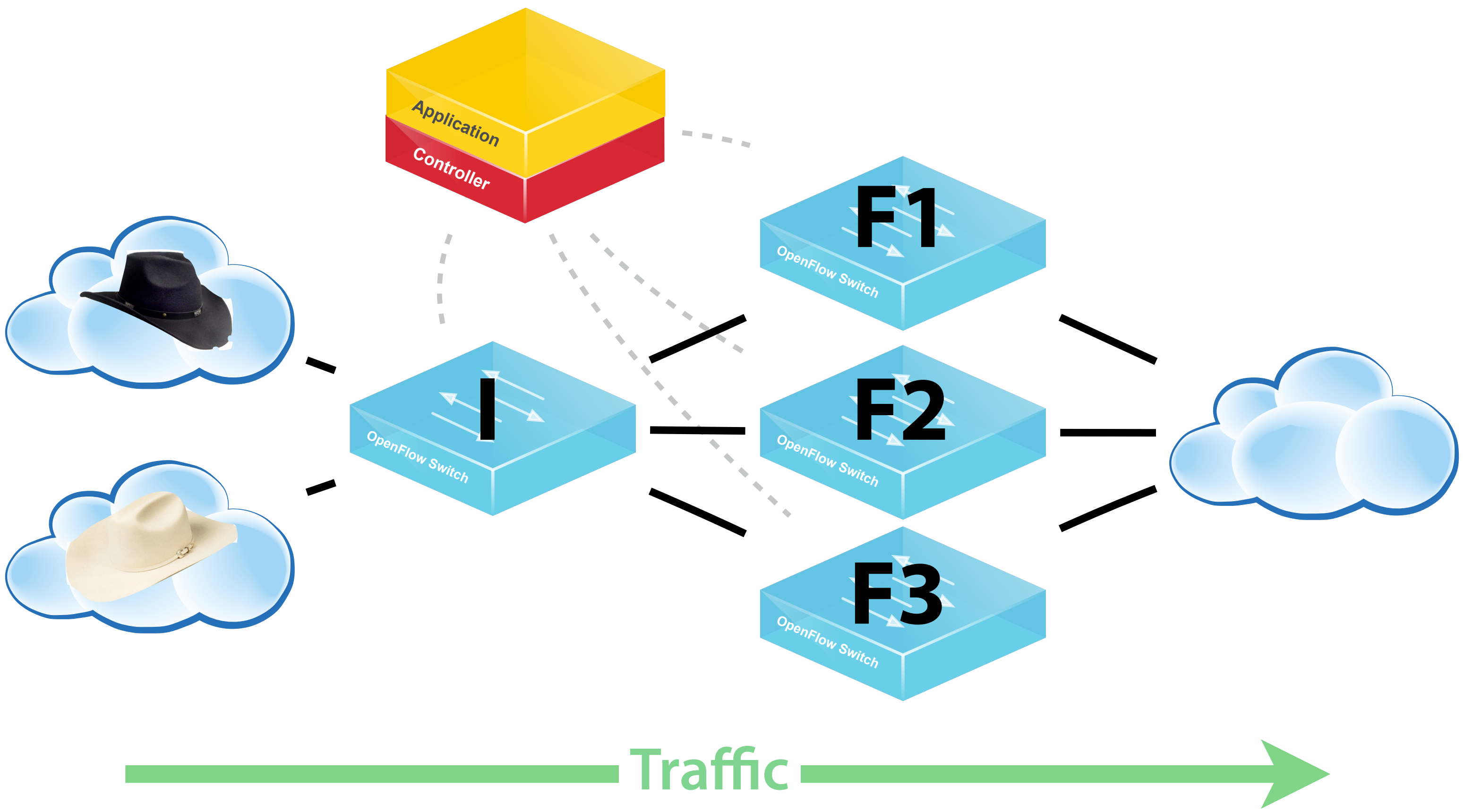
Example: Access Control



Security Policy

Src	Traffic	Action
	Web	Allow
	Non-web	Drop
	Any	Allow

Example: Access Control



Security Policy

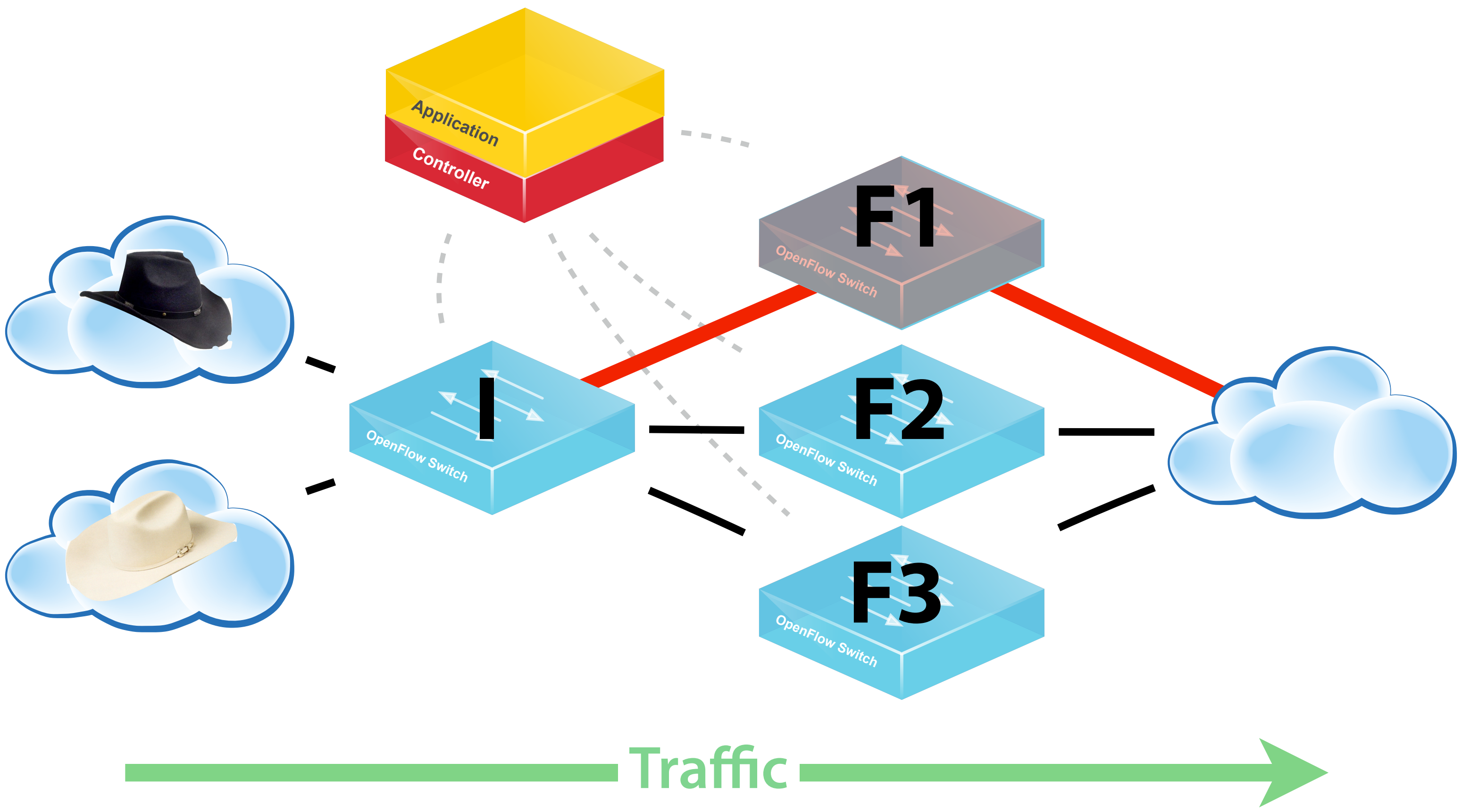
Src	Traffic	Action
	Web	Allow
	Non-web	Drop
	Any	Allow

Configuration A

Process black-hat traffic on F1

Process white-hat traffic on {F2,F3}

Example: Access Control



Security Policy

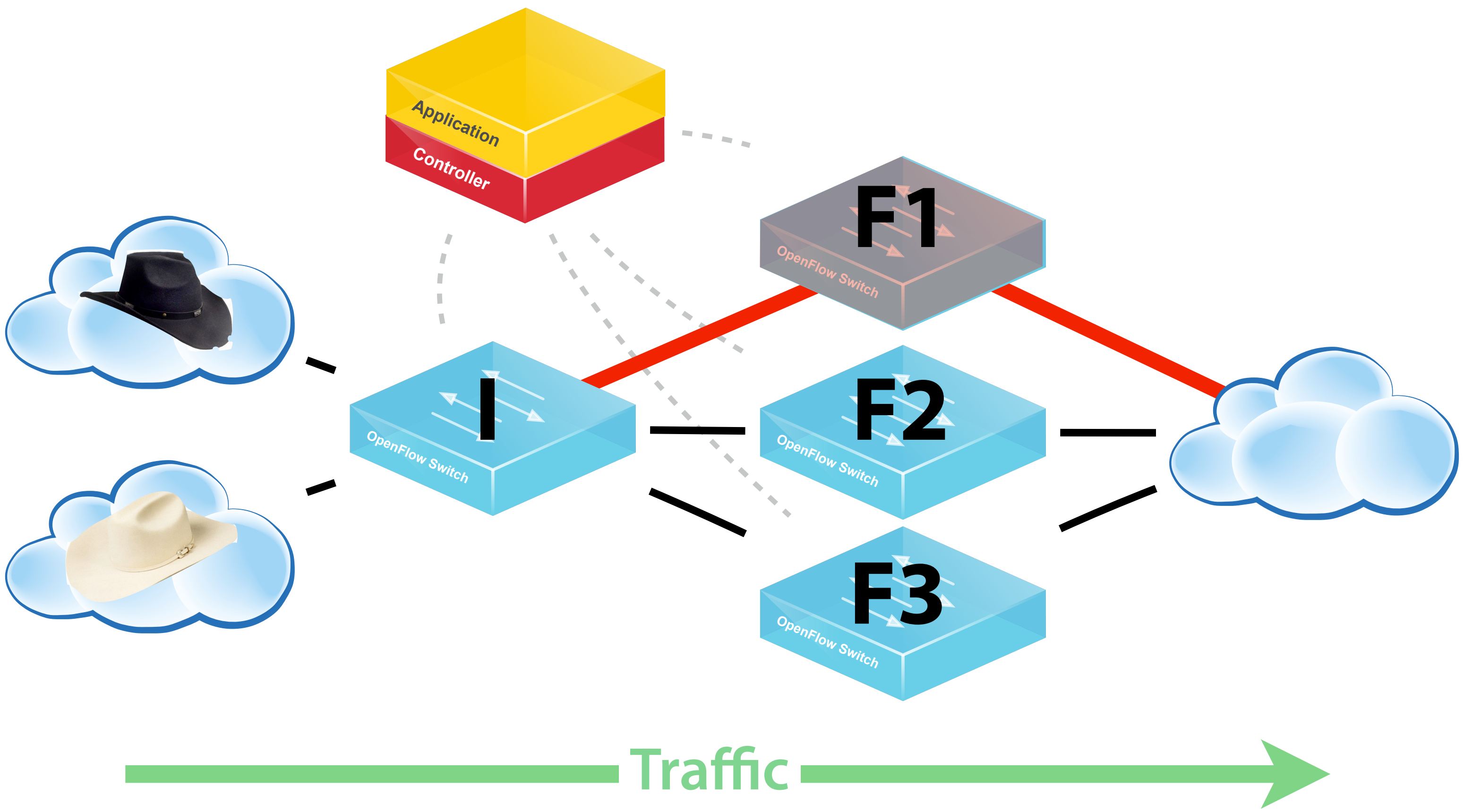
Src	Traffic	Action
	Web	Allow
	Non-web	Drop
	Any	Allow

Configuration A

Process black-hat traffic on F1

Process white-hat traffic on {F2,F3}

Example: Access Control



Security Policy

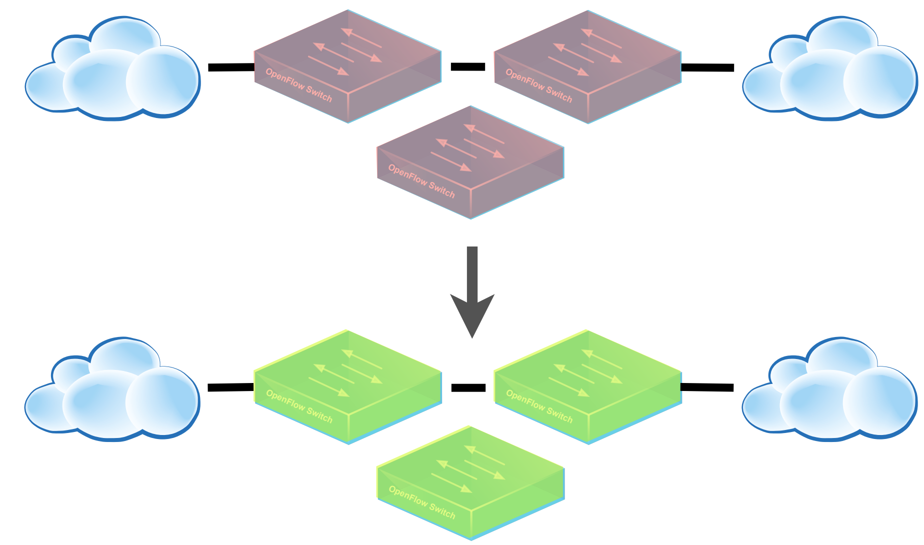
Src	Traffic	Action
	Web	Allow
	Non-web	Drop
	Any	Allow

Configuration A
Process black-hat traffic on F1
Process white-hat traffic on {F2,F3}

?

Configuration B
Process black-hat traffic on {F1,F2}
Process white-hat traffic on F3

Network Updates



Challenges

- The network is a distributed system
- Can only update one element at a time

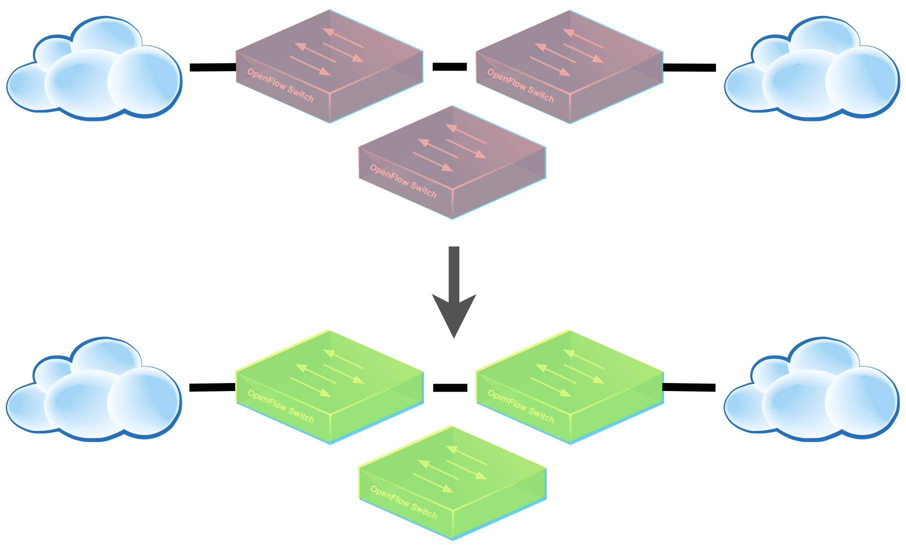
Our Approach

- Provide programmers with a construct for updating the entire network at once
- Semantics ensures "reasonable" behavior
- Engineer efficient implementations:
  - Compiler constructs update protocols
  - Optimizations applied automatically

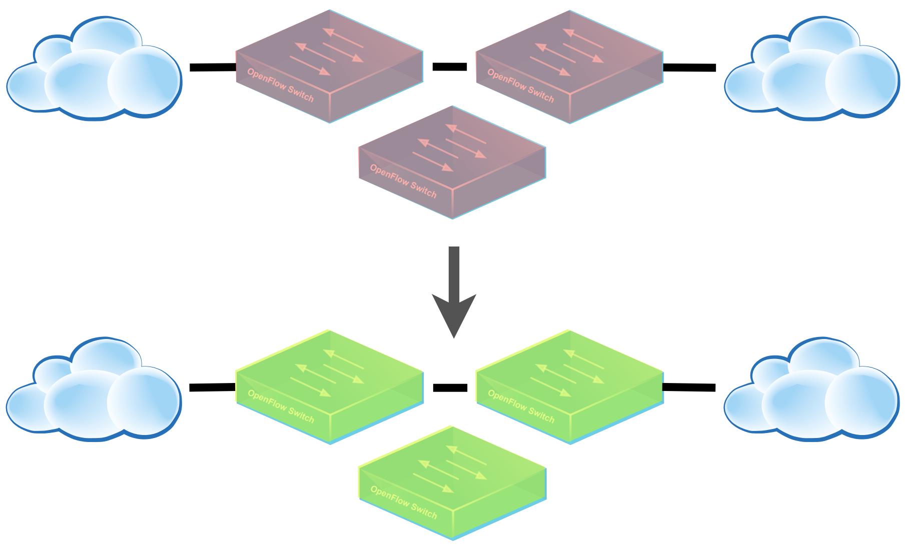
Update Semantics

Atomic Updates

- Seem sensible...
- but costly to implement...
- and difficult to reason about, due to behavior on in-flight packets



Update Semantics



Atomic Updates

- Seem sensible...
- but costly to implement...
- and difficult to reason about, due to behavior on in-flight packets

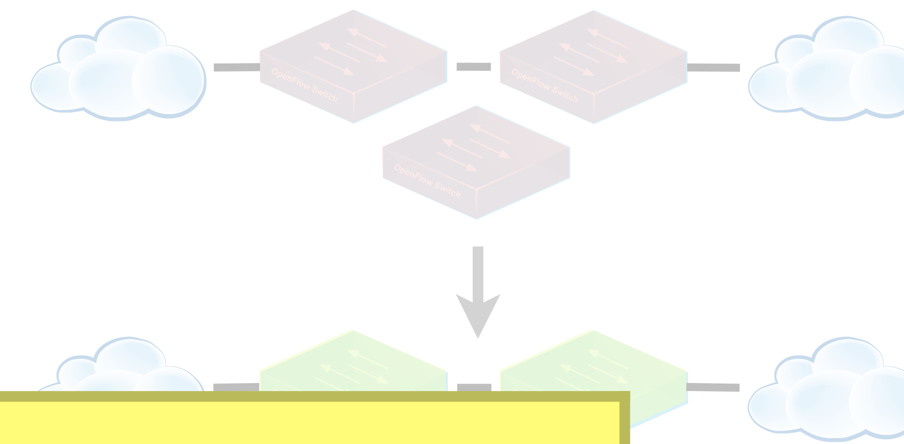
Per-Packet Consistent Updates

Every packet processed with old or new configuration, but not a mixture of the two

Per-Flow Consistent Updates

Every set of related packets processed with old or new configuration, but not a mixture of the two

Update Semantics



Atomic Updates

- Seem sensible...
- but costly to implement...
- and difficult to reason about, due to
  behavi

Per-Pac

Every pa
configu

Theorem (Universal Property Preservation)

An update is per-packet consistent if and only if it preserves all safety properties.

Per-Flow Consistent Updates

Every set of related packets processed with old or new configuration, but not a mixture of the two

Implementation

Two-phase commit

- Build versioned internal and edge switch configurations
- Phase 1: Install internal configuration
- Phase 2: Install edge configuration

Pure Extension

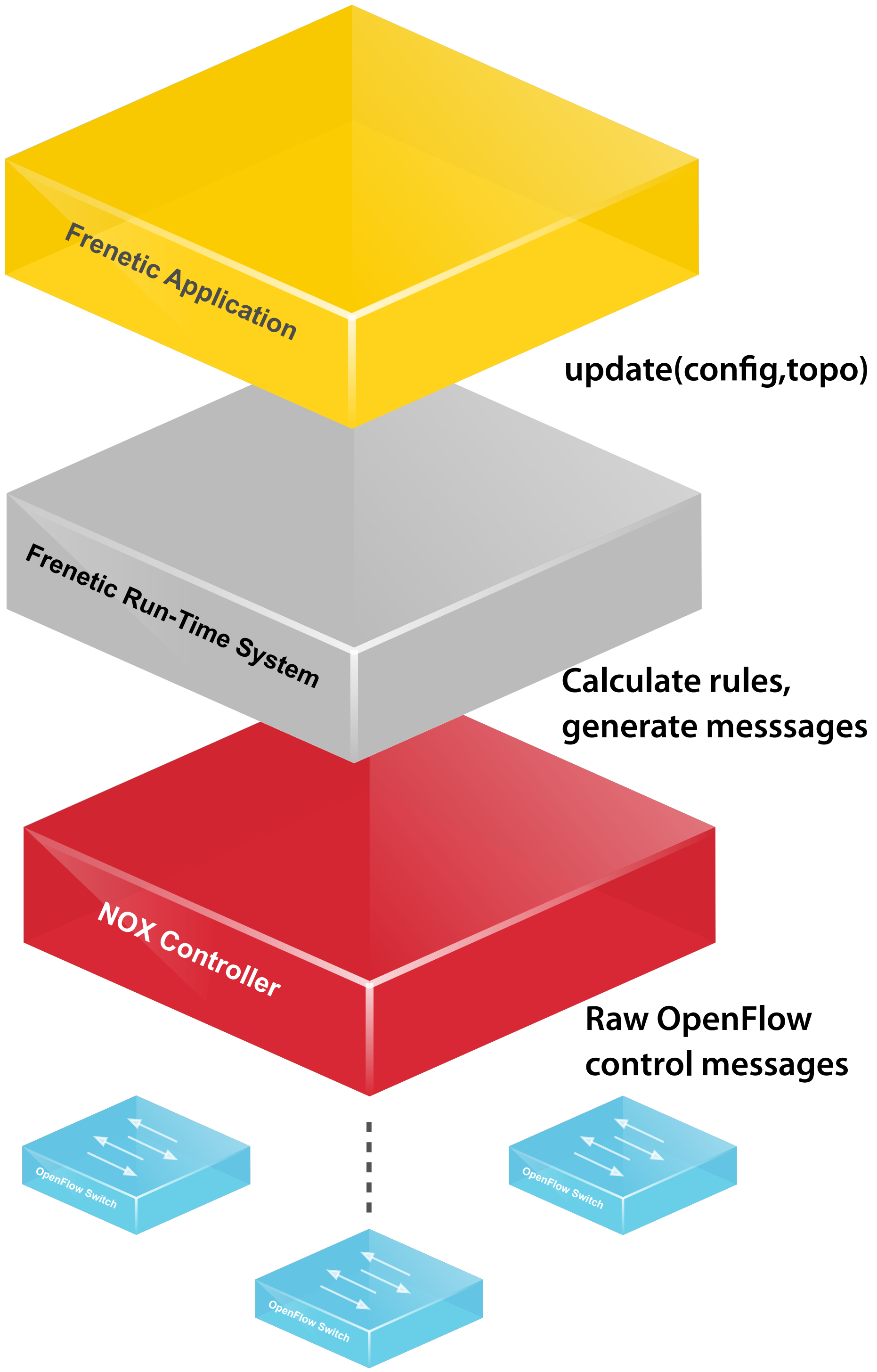
- Update strictly adds paths

Pure Retraction

- Update strictly removes paths

Sub-space updates

- Update modifies a small number of paths



Verification

[PLDI ’13]

Recent Network 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

Service outage was due to a series of internal network events that corrupted router data tables

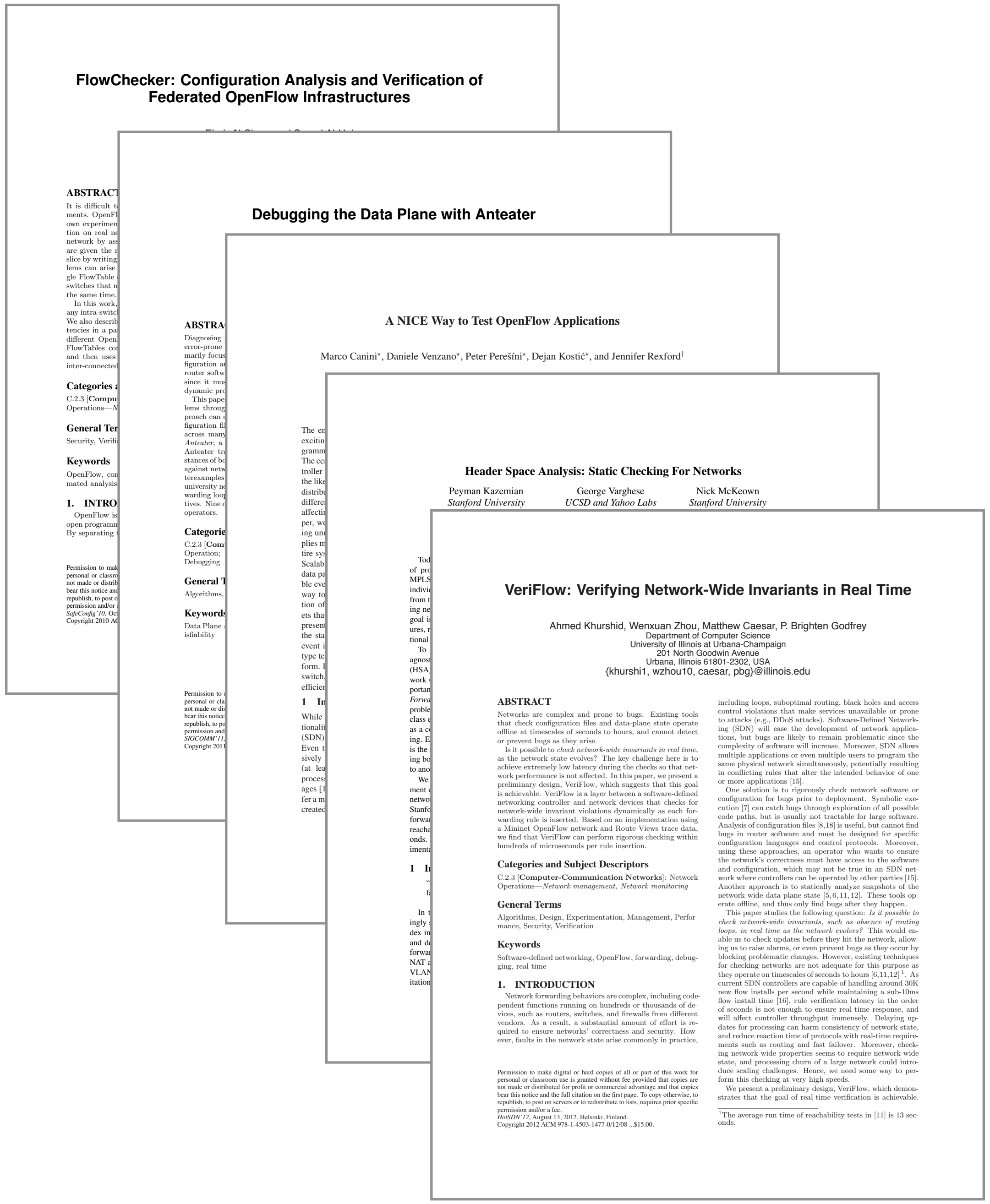
Experienced a network connectivity issue […] **interrupted the airline's flight departures**, airport processing and reservations systems

Existing Tools

There is a cottage industry in configuration-checking tools...

- FlowChecker [SafeConfig '10]
- AntEater [SIGCOMM '11]
- NICE [NSDI '12]
- Header Space Analysis [NSDI '12]
- VeriFlow [HotSDN '12]
- and many others...

This is exciting, because the networking community is starting to get serious about using formal methods

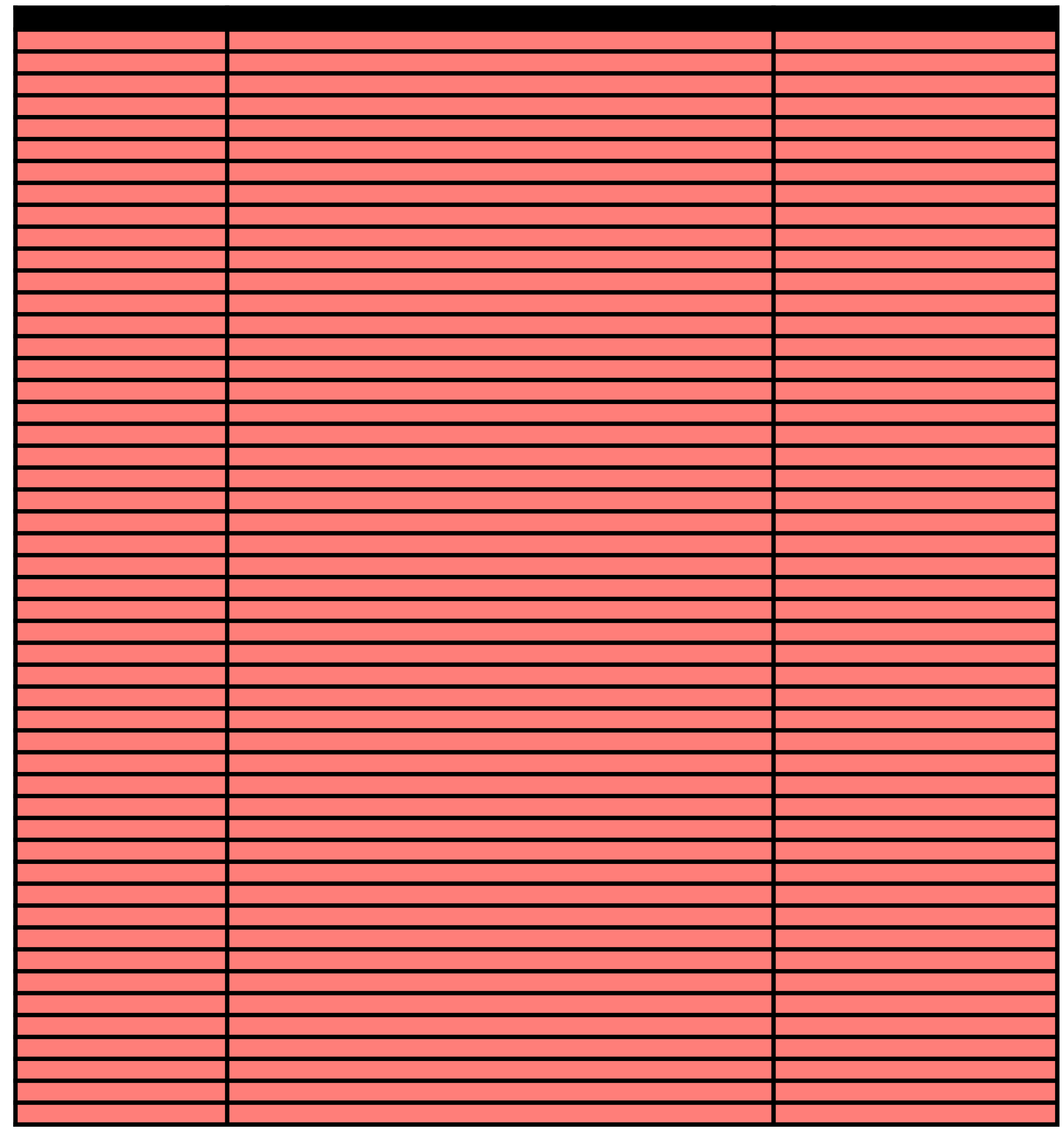


Distributed Programming

Switch flow tables cannot be updated atomically

- Flow tables are huge
- Instructions only add/delete individual entries
- Must update the table live, while it is processing packets

$$\mathtt{table}_{\mathtt{sw}} \subseteq \mathtt{f}|_{\mathtt{sw}}$$



Non-Atomic Updates

Priority	Predicate	Action

$\subseteq$

Priority	Predicate	Action
10	SSH	Drop

$\subseteq$

Priority	Predicate	Action
10	SSH	Drop
5	dst_ip = H1	Fwd 1

$\subseteq$

Priority	Predicate	Action
10	SSH	Drop
5	dst_ip = H1	Fwd 1
5	dst_ip = H2	Fwd 2

Non-Atomic Updates

Priority	Predicate	Action

$\subseteq$

Priority	Predicate	Action
10	SSH	Drop

Priority	Predicate	Action
5	dst_ip = H1	Fwd 1

$\subseteq$

Priority	Predicate	Action
10	SSH	Drop
5	dst_ip = H1	Fwd 1

update re-ordering

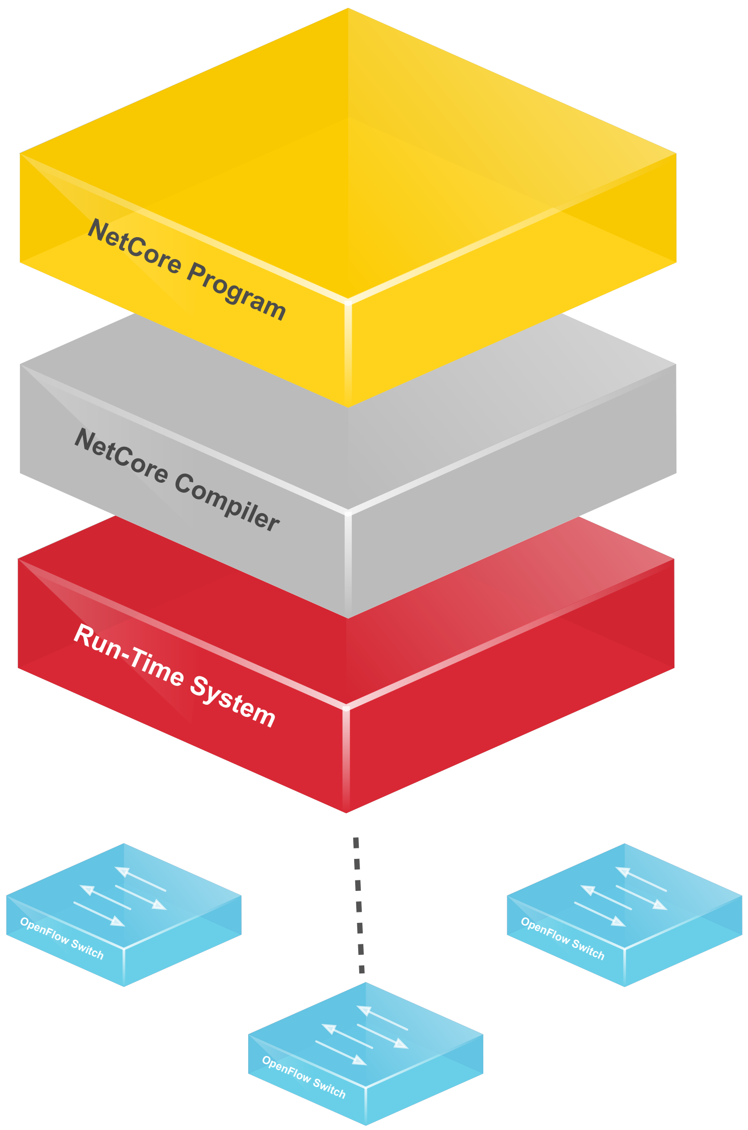
$\subseteq$

Priority	Predicate	Action
5	dst_ip = H1	Fwd 1
5	dst_ip = H2	Fwd 2

Priority	Predicate	Action
10	SSH	Drop
5	dst_ip = H1	Fwd 1
5	dst_ip = H2	Fwd 2

Our Approach

- Write programs in a high-level declarative network programming language
- Use a compiler and run-time system to generate low-level instructions
- Certify the compiler and run-time system using the Coq proof assistant
- Extract to OCaml and run on real hardware



Certified Software Systems

Recent success stories

- seL4 microkernel [SOSP '09]
- CompCert compiler [CACM '09]

Tools

Textbooks

Certified Software Systems

Recent success stories

- seL4 microkernel [SOSP '09]
- CompCert compiler [CACM '09]

Tools

Textbooks

Certified Programming with Dependent Types

```
Inductive pred : Type :=
  | OnSwitch : Switch -> pred
  | InPort : Port -> pred
  | DlSrc : EthernetAddress -> pred
  | DlDst : EthernetAddress -> pred
  | DlVlan : option VLAN -> pred
  | ...
  | And : pred -> pred -> pred
  | Or : pred -> pred -> pred
  | Not : pred -> pred
  | All : pred
  | None : pred

Inductive act : Type :=
  | FwdMod : Mod -> PseudoPort -> act
  | ...

Inductive pol : Type :=
  | Policy : pred -> list act -> pol
  | Union : pol -> pol -> pol
  | Restrict : pol -> pred -> pol
  | ...
```

Write code

Certified Software Systems

Recent success stories

- seL4 microkernel [SOSP '09]
- CompCert compiler [CACM '09]

Tools

Textbooks

Certified Programming with Dependent Types

```
Inductive pred : Type :=
  | OnSwitch : Switch -> pred
```

```
Lemma inter_wildcard_other : forall x,
   Wildcard_inter WildcardAll x = x.
 Proof.
   intros; destruct x; auto.
 Qed.

 Lemma inter_wildcard_other1 : forall x,
   Wildcard_inter x WildcardAll = x.
 Proof.
   intros; destruct x; auto.
 Qed.

 Lemma inter_exact_same : forall x,
   Wildcard_inter (WildcardExact x)
(WildcardExact x) = WildcardExact x.
 Proof.
   intros.
   unfold Wildcard_inter.
   destruct (eqdec x x); intuition.
 Qed.
```

Write code

Prove correct

Certified Software Systems

Recent success stories

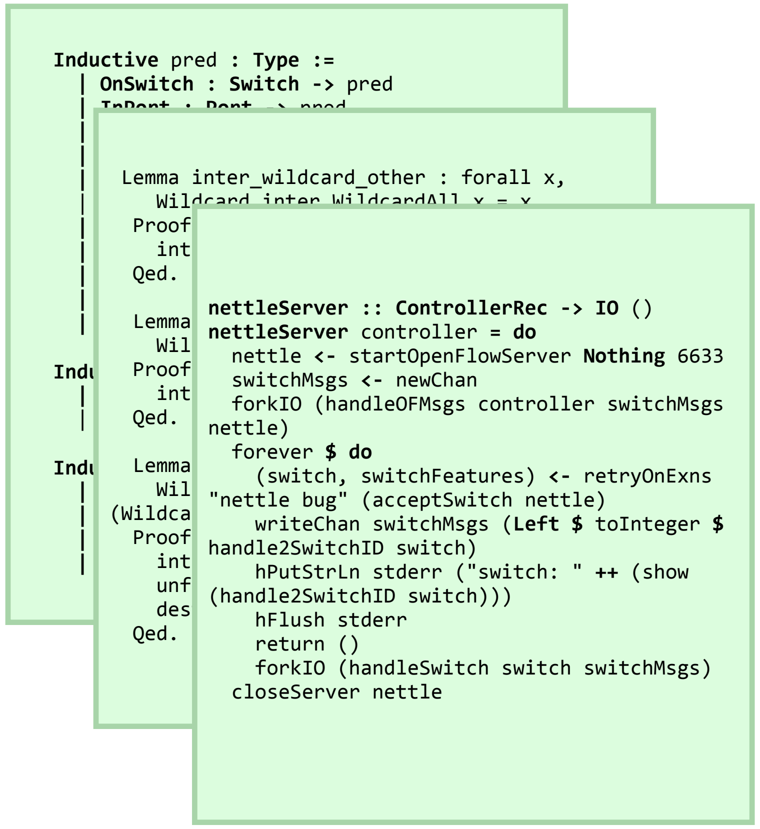
- seL4 microkernel [SOSP '09]
- CompCert compiler [CACM '09]

Tools

ACL2, Isabelle, Coq

Textbooks

Certified Programming with Dependent Types



Write code

Prove correct

Extract code

Certified Software Systems

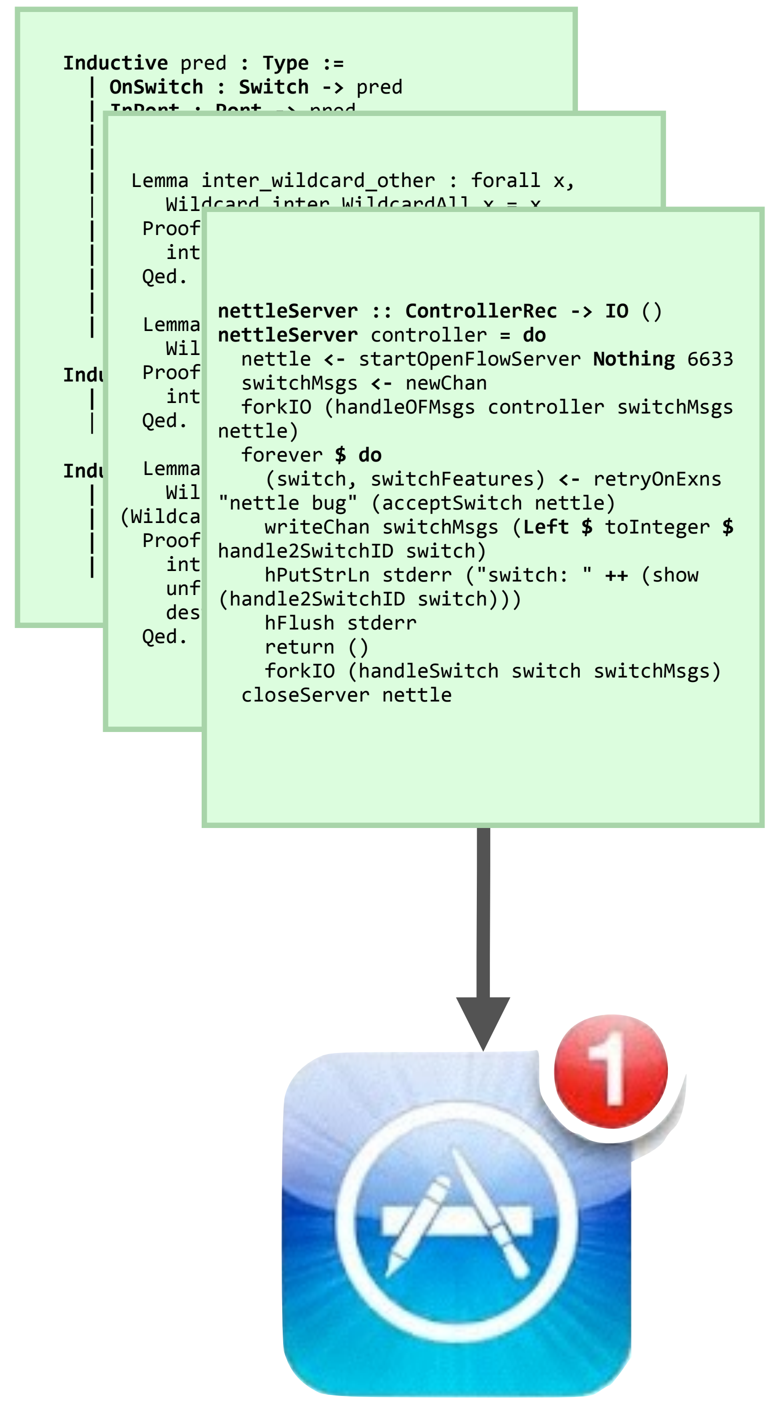
Recent success stories

- seL4 microkernel [SOSP '09]
- CompCert compiler [CACM '09]

Tools

Textbooks

Certified Programming with Dependent Types



Write code

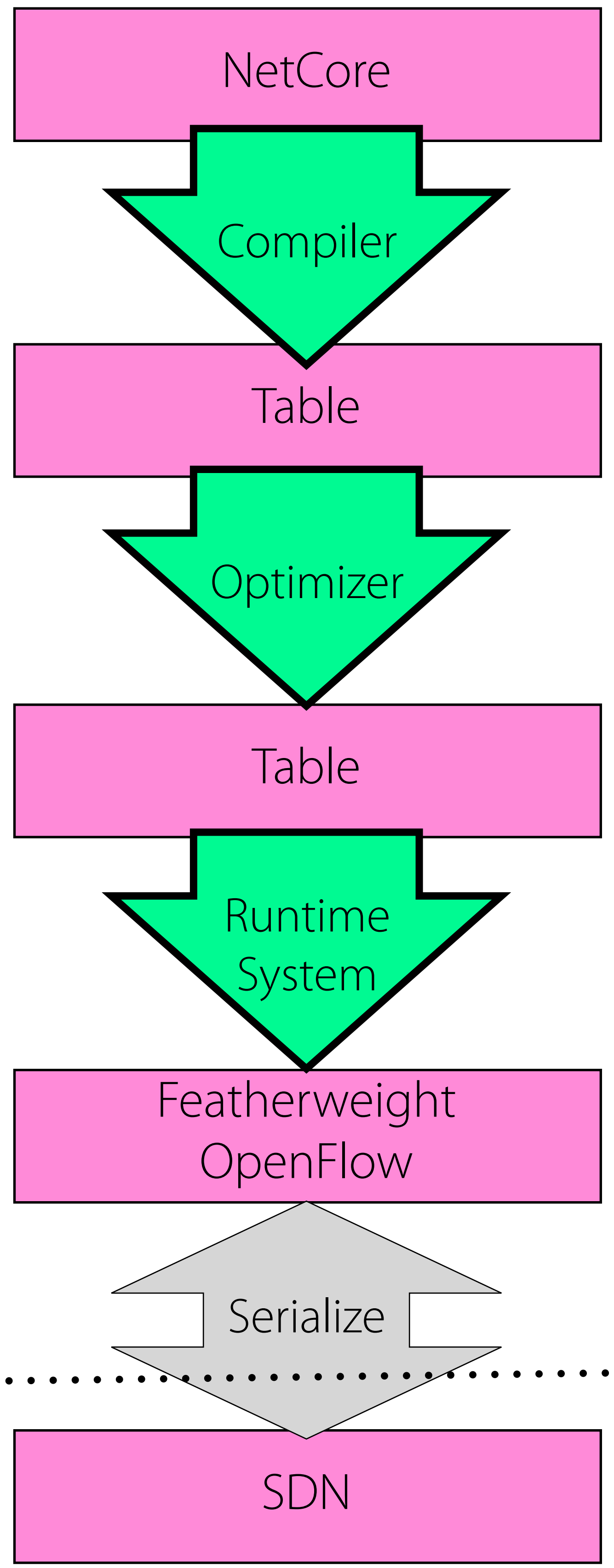
Prove correct

Extract code

Certified executable

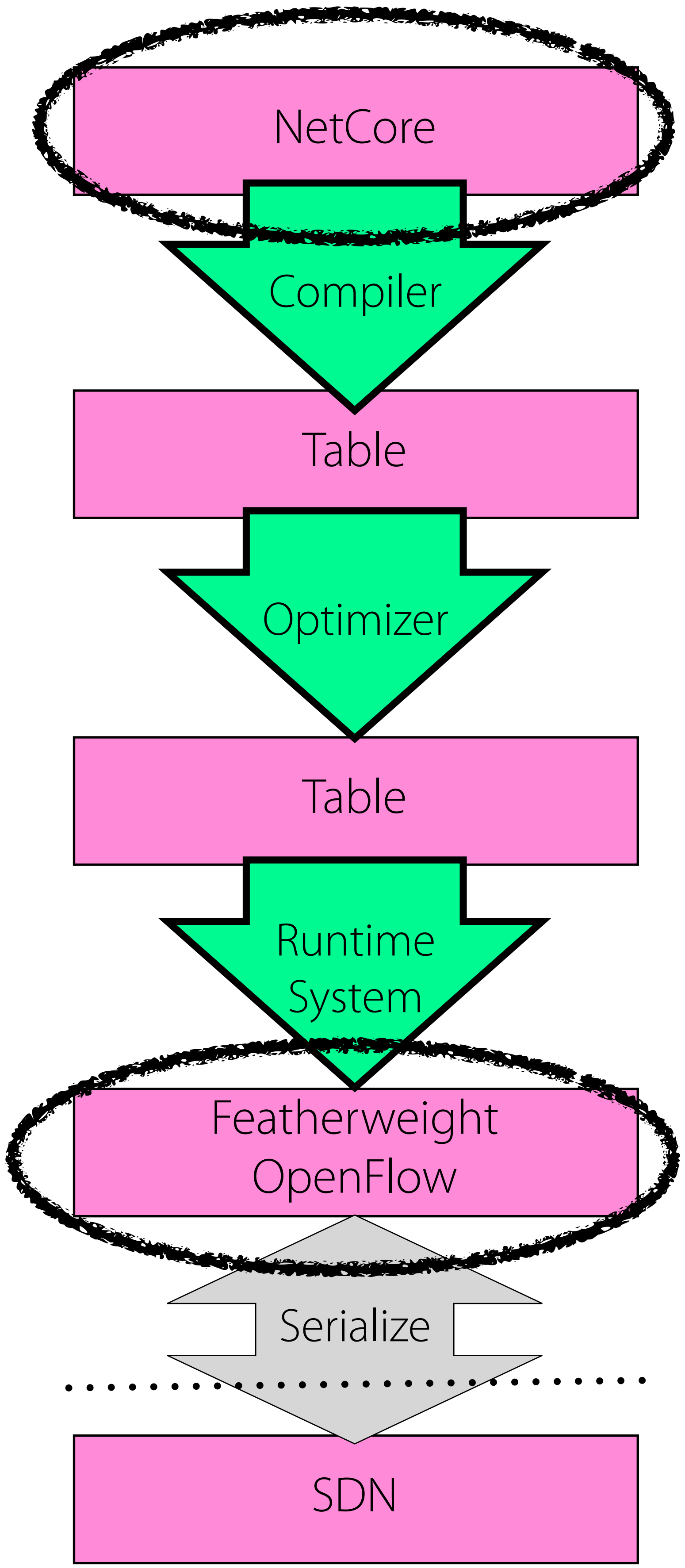
Our Approach

- Write network programs in a high-level declarative programming language
- Use a compiler and run-time system to generate low-level instructions
- Certify the compiler and run-time system using the Coq proof assistant
- Extract to OCaml and execute on real network hardware



Our Approach

- Write network programs in a high-level declarative programming language
- Use a compiler and run-time system to generate low-level instructions
- Certify the compiler and run-time system using the Coq proof assistant
- Extract to OCaml and execute on real network hardware



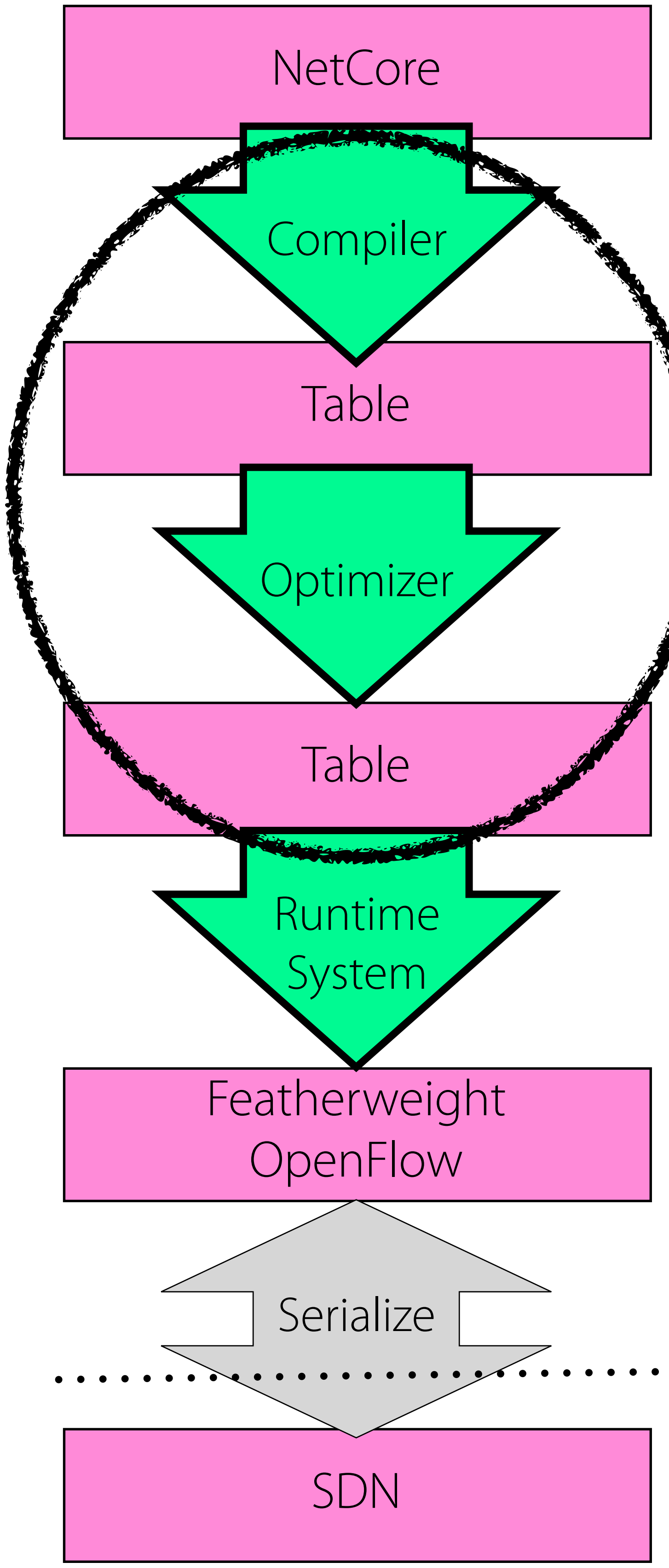
Compiler Correctness

Formalization Highlights

- Library of algebraic properties of tables
- New tactic for proving equalities on bags
- General-purpose table optimizer
- Key invariant: all synthesized predicates are well-formed (w.r.t. protocol types)

Theorem

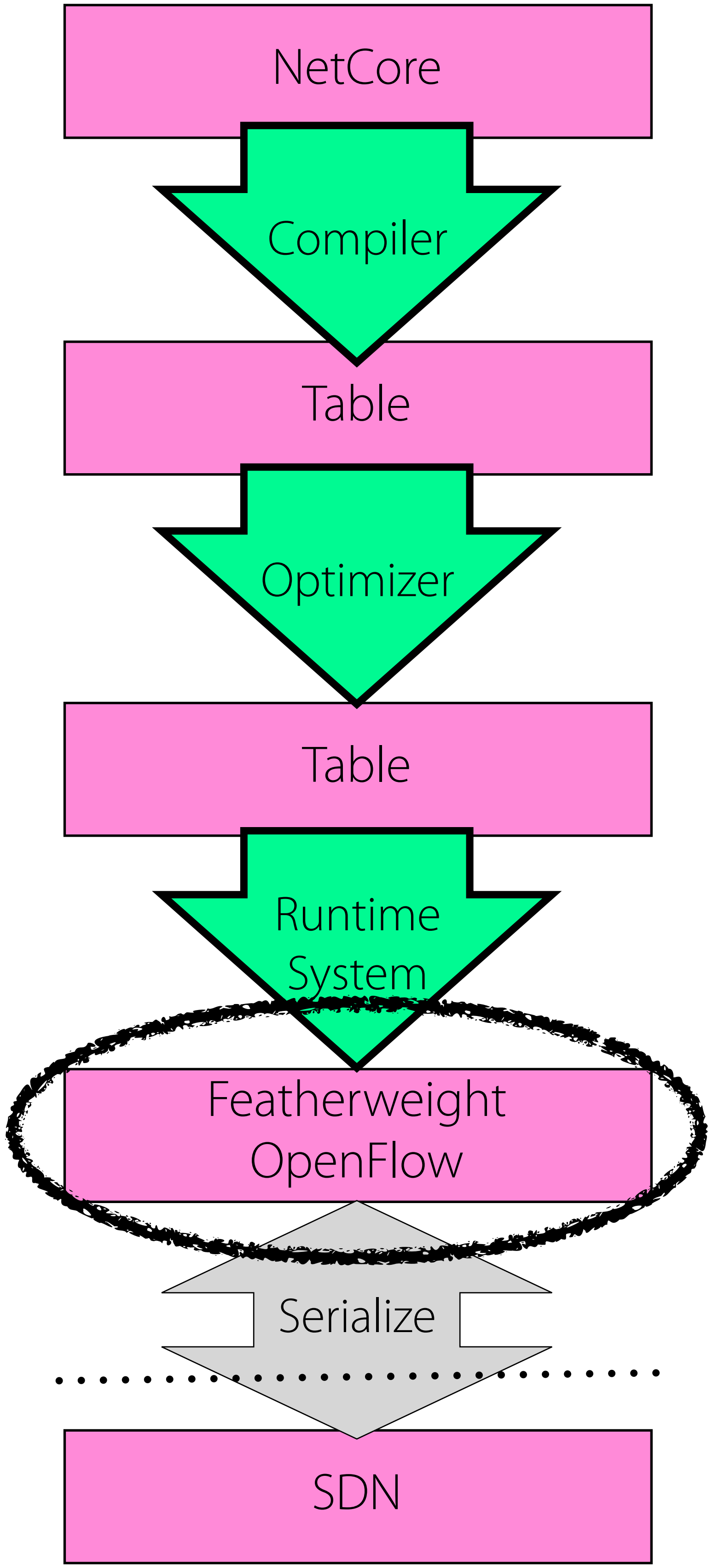
```
Theorem compile_correct :
  forall pol sw pt pk,
    netcore_eval pol sw pt pk =
    table_eval (compile pol sw) pt pk.
```



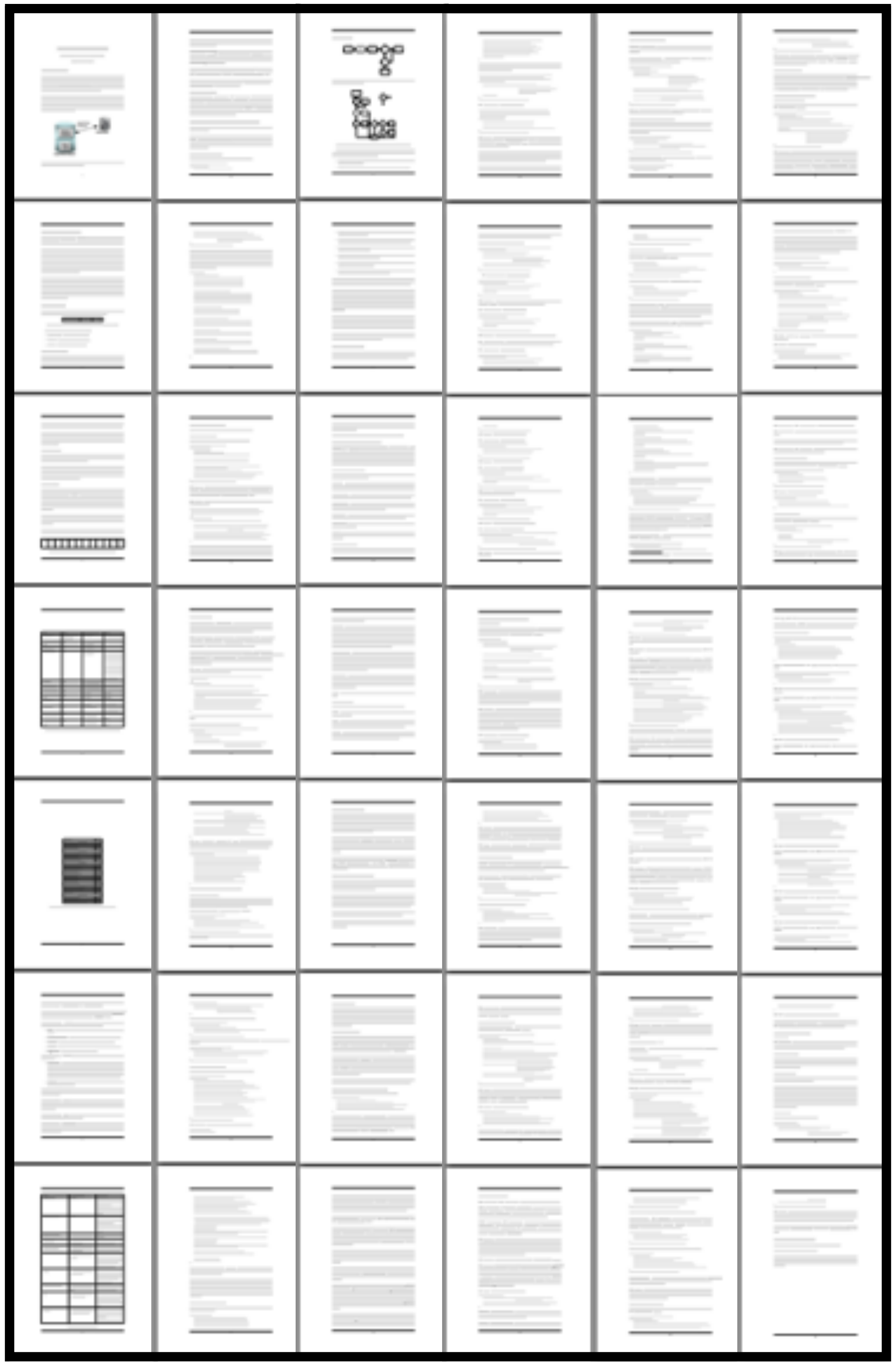
Low-level Semantics

OpenFlow: an open, standardized protocol for programming switches

Dell, HP, NEC, Pica8, OpenVSwitch, etc.



OpenFlow Specification



42 pages...

...of informal prose

...diagrams and flow charts

...and C struct definitions

Featherweight OpenFlow

Syntax

Devices	Switch	S	$::= \mathbb{S}(sw, pts, RT, inp.outp, inm, out$
	Controller	C	$::= \mathbb{C}(\sigma, f_{in}, f_{out})$
	Link	L	$::= \mathbb{L}(loc_{src}, pks, loc_{dst})$
	OpenFlow Link to Controller	M	$::= \mathbb{M}(sw, SMS, CMS)$
Packets and Locations	Packet	pk	$::= abstract$
	Switch ID	sw	$\in \mathbb{N}$
	Port ID	pt	$\in \mathbb{N}$
	Location	loc	$\in sw \times pt$
	Located Packet	lp	$\in loc \times pk$
Controller Components	Controller state	σ	$::= abstract$
	Controller input relation	f_{in}	$\in sw \times CM \times \sigma \rightsquigarrow \sigma$
	Controller output relation	f_{out}	$\in \sigma \rightsquigarrow sw \times SM \times \sigma$
Switch Components	Rule table	RT	$::= abstract$
	Rule table Interpretation	$[\![RT]\!]$	$\in lp \rightarrow \{\!\!\mid lp_1 \cdots lp_n \mid\!\!\} \times \{\!\!\mid CM_1 \cdots C$
	Rule table modifier	ΔRT	$::= abstract$
	Rule table modifier interpretation	apply	$\in \Delta RT \rightarrow RT \rightarrow \Delta RT$
	Ports on switch	pts	$\in \{pt_1 \cdots pt_n\}$
	Consumed packets	inp	$\in \{\!\!\mid lp_1 \cdots lp_n \mid\!\!\}$
	Produced packets	$outp$	$\in \{\!\!\mid lp_1 \cdots lp_n \mid\!\!\}$
	Messages from controller	inm	$\in \{\!\!\mid SM_1 \cdots SM_n \mid\!\!\}$
	Messages to controller	$outm$	$\in \{\!\!\mid CM_1 \cdots CM_n \mid\!\!\}$
Link Components	Endpoints	loc_{src}, loc_{dst}	$\in loc$ where $loc_{src} \neq loc_{dst}$
	Packets from loc_{src} to loc_{dst}	pks	$\in [pk_1 \cdots pk_n]$
Controller Link	Message queue from controller	SMS	$\in [SM_1 \cdots SM_n]$
	Message queue to controller	CMS	$\in [CM_1 \cdots CM_n]$
Abstract OpenFlow Protocol	Message from controller	SM	$::= \textbf{FlowMod}\ \Delta RT \mid \textbf{PktOut}\ pt\ p$
	Message to controller	CM	$::= \textbf{PktIn}\ pt\ pk \mid \textbf{BarrierReply}\ n$

Models all features related to packet forwarding, and *all* essential asynchrony

Semantics

$$\frac{(outp', outm') = [\![RT]\!](lp)}{\mathbb{S}(sw, pts, RT, \{\!\!\mid lp \mid\!\!\} \uplus inp, outp, inm, outm) \xrightarrow{lp} \mathbb{S}(sw, pts, RT, inp, outp' \uplus outp, inm, outm' \uplus outm)} \quad (\textsc{Pkt-Process})$$

$$\frac{}{\begin{array}{ll} & \mathbb{S}(sw, pts, RT, inp, \{\!\!\mid (sw, pt, pk) \mid\!\!\} \uplus outp, inm, outm) \mid \mathbb{L}((sw, pt), pks, loc') \\ \longrightarrow & \mathbb{S}(sw, pts, RT, inp, outp, inm, outm) \mid \mathbb{L}((sw, pt), [pk] +\!\!+ pks, loc') \end{array}} \quad (\textsc{Send-Wire})$$

$$\frac{}{\begin{array}{ll} & \mathbb{L}(loc, pks +\!\!+ [pk], (sw, pt)) \mid \mathbb{S}(sw, pts, RT, inp, outp, inm, outm) \\ \xrightarrow{(sw,pt,pk)} & \mathbb{L}(loc, pks, (sw, pt)) \mid \mathbb{S}(sw, pts, RT, \{\!\!\mid (sw, pt, pk) \mid\!\!\} \uplus inp, outp, inm, outm) \end{array}} \quad (\textsc{Recv-Wire})$$

$$\frac{RT' = \text{apply}(\Delta RT, RT)}{\mathbb{S}(sw, pts, RT, inp, outp, \{\!\!\mid \textbf{FlowMod}\ \Delta RT \mid\!\!\} \uplus inm, outm) \longrightarrow \mathbb{S}(sw, pts, RT', inp, outp, inm, outm)} \quad (\textsc{Switch-FlowMod})$$

$$\frac{pt \in pts}{\mathbb{S}(sw, pts, RT, inp, outp, \{\!\!\mid \textbf{PktOut}\ pt\ pk \mid\!\!\} \uplus inm, outm) \longrightarrow \mathbb{S}(sw, pts, RT, inp, \{\!\!\mid (sw, pt, pk) \mid\!\!\} \uplus outp, inm, outm)} \quad (\textsc{Switch-PktOut})$$

$$\frac{f_{out}(\sigma) \rightsquigarrow (sw, SM, \sigma')}{\mathbb{C}(\sigma, f_{in}, f_{out}) \mid \mathbb{M}(sw, SMS, CMS) \longrightarrow \mathbb{C}(\sigma', f_{in}, f_{out}) \mid \mathbb{M}(sw, [SM] +\!\!+ SMS, CMS)} \quad (\textsc{Ctrl-Send})$$

$$\frac{f_{in}(sw, \sigma, CM) \rightsquigarrow \sigma'}{\mathbb{C}(\sigma, f_{in}, f_{out}) \mid \mathbb{M}(sw, SMS, CMS +\!\!+ [CM]) \longrightarrow \mathbb{C}(\sigma', f_{in}, f_{out}) \mid \mathbb{M}(sw, SMS, CMS)} \quad (\textsc{Ctrl-Recv})$$

$$\frac{SM \neq \textbf{BarrierRequest}\ n}{\begin{array}{ll} & \mathbb{M}(sw, SMS +\!\!+ [SM], CMS) \mid \mathbb{S}(sw, pts, RT, inp, outp, inm, outm) \\ \longrightarrow & \mathbb{M}(sw, SMS, CMS) \mid \mathbb{S}(sw, pts, RT, inp, outp, \{\!\!\mid SM \mid\!\!\} \uplus inm, outm) \end{array}} \quad (\textsc{Switch-Recv-Ctrl})$$

$$\frac{}{\begin{array}{ll} & \mathbb{M}(sw, SMS +\!\!+ [\textbf{BarrierRequest}\ n], CMS) \mid \mathbb{S}(sw, pts, RT, inp, outp, \emptyset, outm) \\ \longrightarrow & \mathbb{M}(sw, SMS, CMS) \mid \mathbb{S}(sw, pts, RT, inp, outp, \emptyset, \{\!\!\mid \textbf{BarrierReply}\ n \mid\!\!\} \uplus outm) \end{array}} \quad (\textsc{Switch-Recv-Barrier})$$

$$\frac{}{\begin{array}{ll} & \mathbb{S}(sw, pts, RT, inp, outp, inm, \{\!\!\mid CM \mid\!\!\} \uplus outm) \mid \mathbb{M}(sw, SMS, CMS) \\ \longrightarrow & \mathbb{S}(sw, pts, RT, inp, outp, inm, outm) \mid \mathbb{M}(sw, SMS, [CM] +\!\!+ CMS) \end{array}} \quad (\textsc{Switch-Send-Ctrl})$$

Forwarding

```
/* Fields to match against flows */
struct ofp_match {
    uint32_t wildcards;        /* Wildcard fields. */
    uint16_t in_port;          /* Input switch port. */
    uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
    uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
    uint16_t dl_vlan;          /* Input VLAN. */
    uint8_t dl_vlan_pcp;       /* Input VLAN priority. */
    uint8_t pad1[1];           /* Align to 64-bits. */
    uint16_5 dl_type;          /* Ethernet frame type. */
    uint8_t nw_tos;            /* IP ToS (DSCP field, 6 bits). */
    uint8_t nw_proto;          /* IP protocol or lower 8 bits of
                                  ARP opcode. */
    uint8_t pad2[2];           /* Align to 64-bits. */
    uint32_t nw_src;           /* IP source address. */
    uint32_t nw_dst;           /* IP destination address. */
    uint16_t tp_src;           /* TCP/UDP source port. */
    uint16_t tp_dst;           /* TCP/UDP destination port. */
};
OFP_ASSERT(sizeof(struct ofp_match) == 40);
```

Forwarding

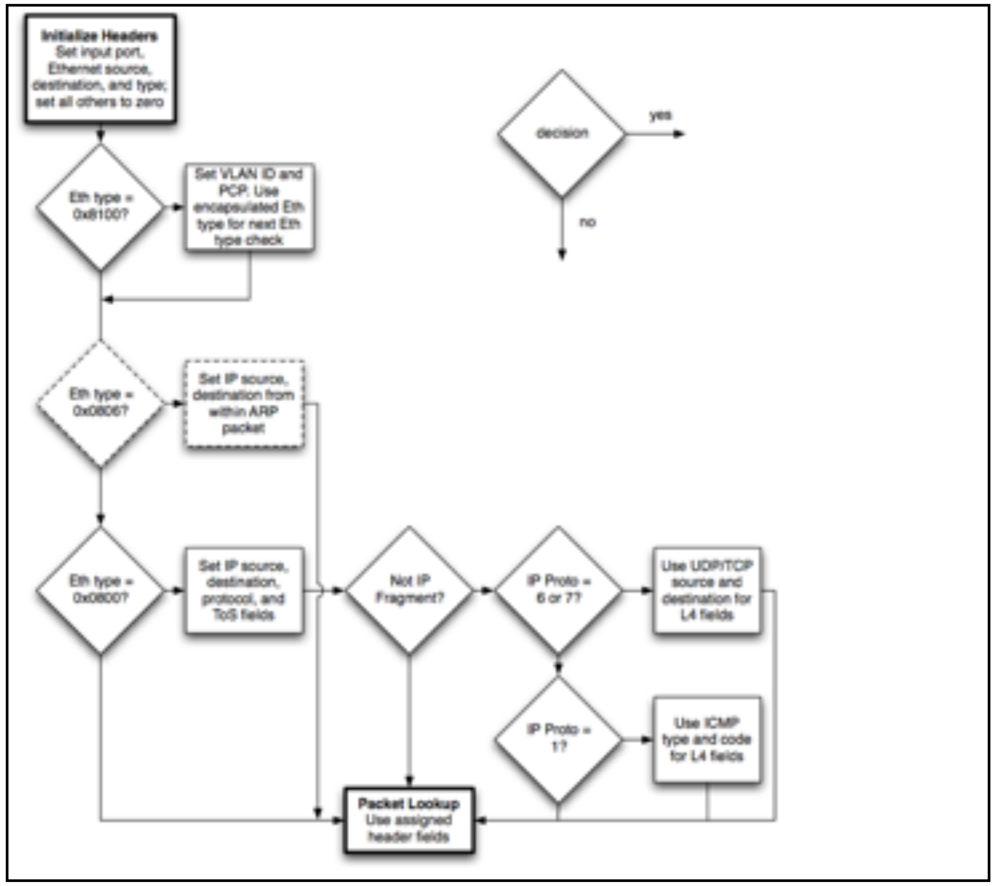
```c
/* Fields to match against flows */
struct ofp_match {
    uint32_t wildcards;        /* Wildcard fields. */
    uint16_t in_port;          /* Input switch port. */
    uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
    uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
    uint16_t dl_vlan;          /* Input VLAN. */
    uint8_t dl_vlan_pcp;       /* Input VLAN priority. */
    uint8_t pad1[1];           /* Align to 64-bits. */
    uint16_5 dl_type;          /* Ethernet frame type. */
    uint8_t nw_tos;            /* IP ToS (DSCP field, 6 bits). */
    uint8_t nw_proto;          /* IP protocol or lower 8 bits of
                                  ARP opcode. */
    uint8_t pad2[2];           /* Align to 64-bits. */
    uint32_t nw_src;           /* IP source address. */
    uint32_t nw_dst;           /* IP destination address. */
    uint16_t tp_src;           /* TCP/UDP source port. */
    uint16_t tp_dst;           /* TCP/UDP destination port. */
};
OFP_ASSERT(sizeof(struct ofp_match) == 40);
```

```coq
Record Pattern : Type := MkPattern {
  dlSrc : Wildcard EthernetAddress;
  dlDst : Wildcard EthernetAddress;
  dlType : Wildcard EthernetType;
  dlVlan : Wildcard VLAN;
  dlVlanPcp : Wildcard VLANPriority;
  nwSrc : Wildcard IPAddress;
  nwDst : Wildcard IPAddress;
  nwProto : Wildcard IPProtocol;
  nwTos : Wildcard IPTypeOfService;
  tpSrc : Wildcard TransportPort;
  tpDst : Wildcard TransportPort;
  inPort : Wildcard Port
}.
```

Forwarding

```
/* Fields to match against flows */
struct ofp_match {
    uint32_t wildcards;        /* Wildcard fields. */
    uint16_t in_port;          /* Input switch port. */
    uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
    uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
    uint16_t dl_vlan;          /* Input VLAN. */
    uint8_t dl_vlan_pcp;       /* Input VLAN priority. */
    uint8_t pad1[1];           /* Align to 64-bits. */
    uint16_5 dl_type;          /* Ethernet frame type. */
    uint8_t nw_tos;            /* IP ToS (DSCP field, 6 bits). */
    uint8_t nw_proto;          /* IP protocol or lower 8 bits of
                                  ARP opcode. */
    uint8_t pad2[2];           /* Align to 64-bits. */
    uint32_t nw_src;           /* IP source address. */
    uint32_t nw_dst;           /* IP destination address. */
    uint16_t tp_src;           /* TCP/UDP source port. */
    uint16_t tp_dst;           /* TCP/UDP destination port. */
};
OFP_ASSERT(sizeof(struct ofp_match) == 40);
```

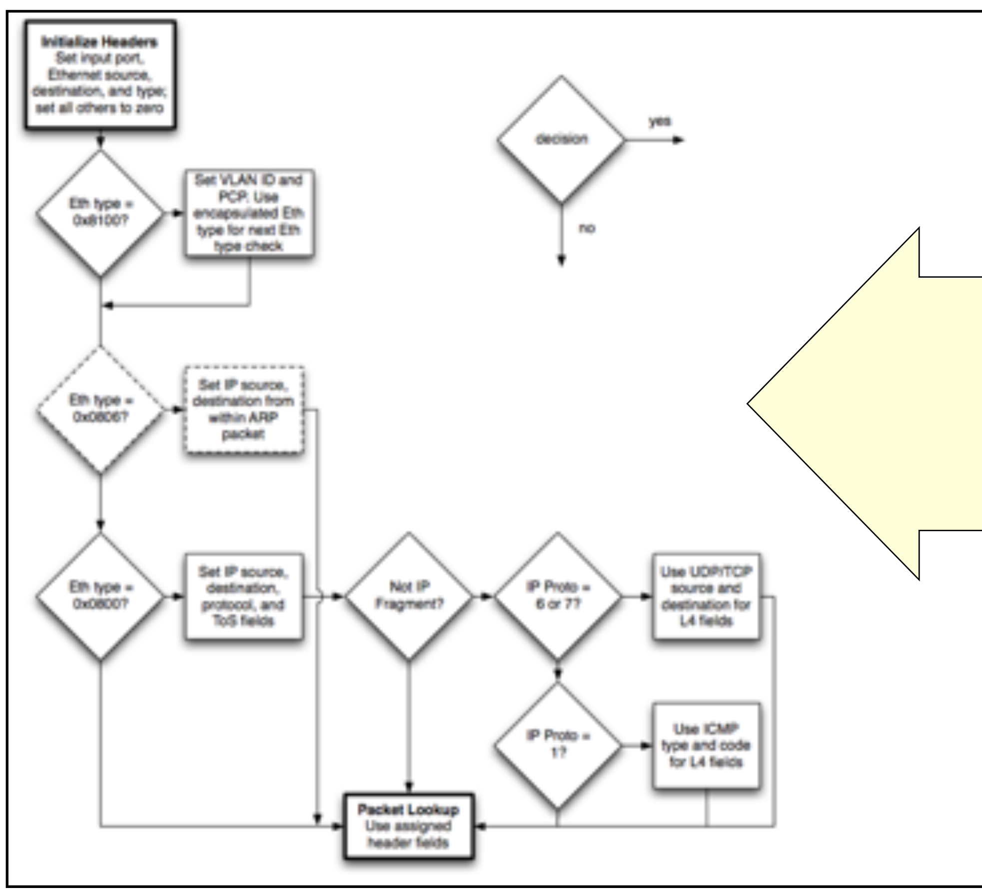
```
Record Pattern : Type := MkPattern {
  dlSrc : Wildcard EthernetAddress;
  dlDst : Wildcard EthernetAddress;
  dlType : Wildcard EthernetType;
  dlVlan : Wildcard VLAN;
  dlVlanPcp : Wildcard VLANPriority;
  nwSrc : Wildcard IPAddress;
  nwDst : Wildcard IPAddress;
  nwProto : Wildcard IPProtocol;
  nwTos : Wildcard IPTypeOfService;
  tpSrc : Wildcard TransportPort;
  tpDst : Wildcard TransportPort;
  inPort : Wildcard Port
}.
```



Forwarding

```c
/* Fields to match against flows */
struct ofp_match {
    uint32_t wildcards;         /* Wildcard fields. */
    uint16_t in_port;           /* Input switch port. */
    uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
    uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
    uint16_t dl_vlan;           /* Input VLAN. */
    uint8_t dl_vlan_pcp;        /* Input VLAN priority. */
    uint8_t pad1[1];            /* Align to 64-bits. */
    uint16_5 dl_type;           /* Ethernet frame type. */
    uint8_t nw_tos;             /* IP ToS (DSCP field, 6 bits). */
    uint8_t nw_proto;           /* IP protocol or lower 8 bits of
                                   ARP opcode. */
    uint8_t pad2[2];            /* Align to 64-bits. */
    uint32_t nw_src;            /* IP source address. */
    uint32_t nw_dst;            /* IP destination address. */
    uint16_t tp_src;            /* TCP/UDP source port. */
    uint16_t tp_dst;            /* TCP/UDP destination port. */
};
OFP_ASSERT(sizeof(struct ofp_match) == 40);
```

```coq
Record Pattern : Type := MkPattern {
  dlSrc : Wildcard EthernetAddress;
  dlDst : Wildcard EthernetAddress;
  dlType : Wildcard EthernetType;
  dlVlan : Wildcard VLAN;
  dlVlanPcp : Wildcard VLANPriority;
  nwSrc : Wildcard IPAddress;
  nwDst : Wildcard IPAddress;
  nwProto : Wildcard IPProtocol;
  nwTos : Wildcard IPTypeOfService;
  tpSrc : Wildcard TransportPort;
  tpDst : Wildcard TransportPort;
  inPort : Wildcard Port
}.
```



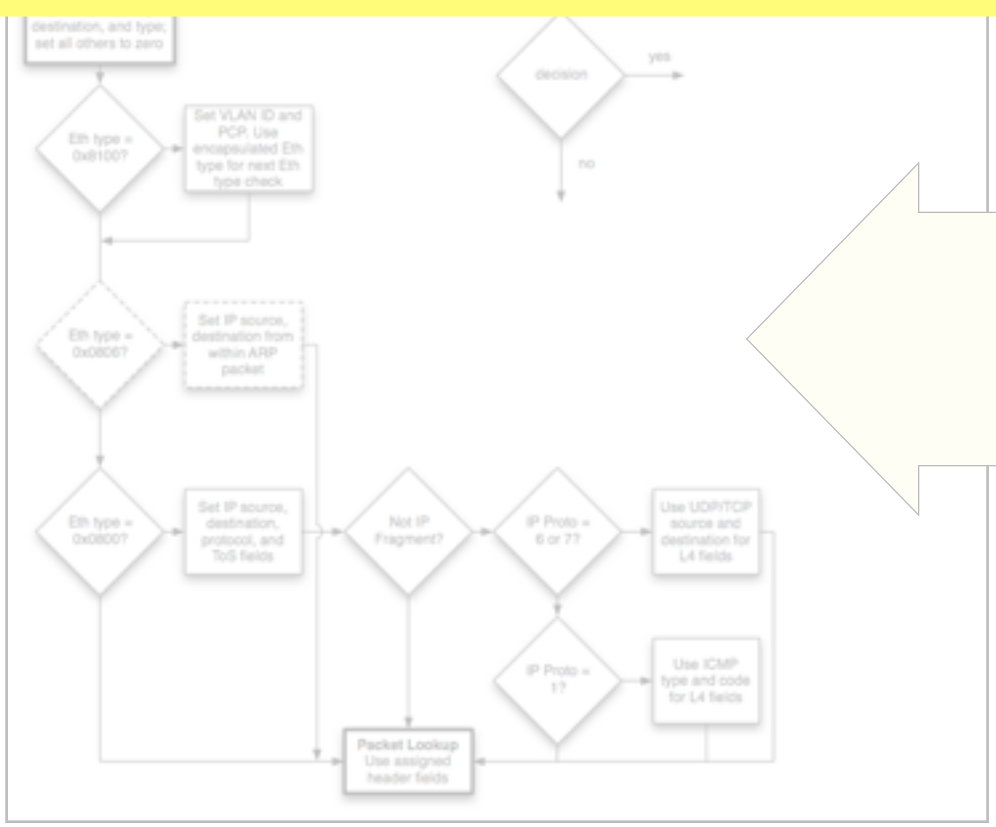

```coq
Definition Pattern_inter (p p':Pattern) :=
  let dlSrc := Wildcard_inter EthernetAddress.eqdec (ptrnDlSrc p) (ptrnDlSrc p') in
  let dlDst := Wildcard_inter EthernetAddress.eqdec (ptrnDlDst p) (ptrnDlDst p') in
  let dlType := Wildcard_inter Word16.eqdec (ptrnDlType p) (ptrnDlType p') in
  let dlVlan := Wildcard_inter Word16.eqdec (ptrnDlVlan p) (ptrnDlVlan p') in
  let dlVlanPcp := Wildcard_inter Word8.eqdec (ptrnDlVlanPcp p) (ptrnDlVlanPcp p') in
  let nwSrc := Wildcard_inter Word32.eqdec (ptrnNwSrc p) (ptrnNwSrc p') in
  let nwDst := Wildcard_inter Word32.eqdec (ptrnNwDst p) (ptrnNwDst p') in
  let nwProto := Wildcard_inter Word8.eqdec (ptrnNwProto p) (ptrnNwProto p') in
  let nwTos := Wildcard_inter Word8.eqdec (ptrnNwTos p) (ptrnNwTos p') in
  let tpSrc := Wildcard_inter Word16.eqdec (ptrnTpSrc p) (ptrnTpSrc p') in
  let tpDst := Wildcard_inter Word16.eqdec (ptrnTpDst p) (ptrnTpDst p') in
  let inPort := Wildcard_inter Word16.eqdec (ptrnInPort p) (ptrnInPort p') in
    MkPattern dlSrc dlDst dlType dlVlan dlVlanPcp
              nwSrc nwDst nwProto nwTos
              tpSrc tpDst
              inPort.
Definition exact_pattern (pk : Packet) (pt : Word16.T) : Pattern :=
  MkPattern
    (WildcardExact (pktDlSrc pk)) (WildcardExact (pktDlDst pk))
    (WildcardExact (pktDlTyp pk))
    (WildcardExact (pktDlVlan pk)) (WildcardExact (pktDlVlanPcp pk))
    (WildcardExact (pktNwSrc pk)) (WildcardExact (pktNwDst pk))
    (WildcardExact (pktNwProto pk)) (WildcardExact (pktNwTos pk))
    (Wildcard_of_option (pktTpSrc pk)) (Wildcard_of_option (pktTpDst pk))
    (WildcardExact pt).
Definition match_packet (pt : Word16.T) (pk : Packet) (pat : Pattern) : bool :=
  negb (Pattern_is_empty (Pattern_inter (exact_pattern pk pt) pat)).
```

Forwarding

```c
/* Fields to match against flows */
struct ofp_match {
    uint32_t wildcards;        /* Wildcard fields. */
    uint16_t in_port;          /* Input switch port. */
    uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
    uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
    uint16_t dl_vlan;          /* Input VLAN. */
    uint8_t dl_vlan_pcp;       /* Input VLAN priority. */
    uint8_t pad1[1];           /* Align to 64-bits. */
    uint16_5 dl_type;          /* Ethernet frame type. */
    uint8_t nw_tos;            /* IP ToS (DSCP field, 6 bits). */
    uint8_t nw_proto;          /* IP protocol or lower 8 bits of
                                  ARP opcode. */
    uint8_t pad2[2];           /* Align to 64-bits. */
    uint32_t nw_src;           /* IP source address. */
    uint32_t nw_dst;           /* IP destination address. */
    uint16_t tp_src;           /* TCP/UDP source port. */
    uint16_t tp_dst;           /* TCP/UDP destination port. */
};
OFP_ASSERT(sizeof(struct ofp_match) == 40);
```

```coq
Record Pattern : Type := MkPattern {
  dlSrc : Wildcard EthernetAddress;
  dlDst : Wildcard EthernetAddress;
  dlType : Wildcard EthernetType;
  dlVlan : Wildcard VLAN;
  dlVlanPcp : Wildcard VLANPriority;
  nwSrc : Wildcard IPAddress;
  nwDst : Wildcard IPAddress;
  nwProto : Wildcard IPProtocol;
  nwTos : Wildcard IPTypeOfService;
  tpSrc : Wildcard TransportPort;
  tpDst : Wildcard TransportPort;
  inPort : Wildcard Port
}.
```

Detailed model of matching, forwarding, and flow table update




```coq
  let dlDst := Wildcard_inter EthernetAddress.eqdec (ptrnDlDst p) (ptrnDlDst p') in
  let dlType := Wildcard_inter Word16.eqdec (ptrnDlType p) (ptrnDlType p') in
  let dlVlan := Wildcard_inter Word16.eqdec (ptrnDlVlan p) (ptrnDlVlan p') in
  let dlVlanPcp := Wildcard_inter Word8.eqdec (ptrnDlVlanPcp p) (ptrnDlVlanPcp p') in
  let nwSrc := Wildcard_inter Word32.eqdec (ptrnNwSrc p) (ptrnNwSrc p') in
  let nwDst := Wildcard_inter Word32.eqdec (ptrnNwDst p) (ptrnNwDst p') in
  let nwProto := Wildcard_inter Word8.eqdec (ptrnNwProto p) (ptrnNwProto p') in
  let nwTos := Wildcard_inter Word8.eqdec (ptrnNwTos p) (ptrnNwTos p') in
  let tpSrc := Wildcard_inter Word16.eqdec (ptrnTpSrc p) (ptrnTpSrc p') in
  let tpDst := Wildcard_inter Word16.eqdec (ptrnTpDst p) (ptrnTpDst p') in
  let inPort := Wildcard_inter Word16.eqdec (ptrnInPort p) (ptrnInPort p') in
    MkPattern dlSrc dlDst dlType dlVlan dlVlanPcp
              nwSrc nwDst nwProto nwTos
              tpSrc tpDst
              inPort.
Definition exact_pattern (pk : Packet) (pt : Word16.T) : Pattern :=
  MkPattern
    (WildcardExact (pktDlSrc pk)) (WildcardExact (pktDlDst pk))
    (WildcardExact (pktDlTyp pk))
    (WildcardExact (pktDlVlan pk)) (WildcardExact (pktDlVlanPcp pk))
    (WildcardExact (pktNwSrc pk)) (WildcardExact (pktNwDst pk))
    (WildcardExact (pktNwProto pk)) (WildcardExact (pktNwTos pk))
    (Wildcard_of_option (pktTpSrc pk)) (Wildcard_of_option (pktTpDst pk))
    (WildcardExact pt).
Definition match_packet (pt : Word16.T) (pk : Packet) (pat : Pattern) : bool :=
  negb (Pattern_is_empty (Pattern_inter (exact_pattern pk pt) pat)).
```

Asynchrony

“In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance.”

“There is no packet output ordering guaranteed within a port.”

Asynchrony

"In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance."

"There is no packet output ordering guaranteed within a port."

```
Definition InBuf := Bag Packet.
Definition OutBuf := Bag Packet.
Definition OFInBuf := Bag SwitchMsg.
Definition OFOutBuf := Bag CtrlMsg.
```

Asynchrony

"In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance."

"There is no packet output ordering guaranteed within a port."

Essential asynchrony: packet buffers, message reordering, and barriers

```
Definition InBuf := Bag Packet.
Definition OutBuf := Bag Packet.
Definition OFInBuf := Bag SwitchMsg.
Definition OFOutBuf := Bag CtrlMsg.
```

Weak Bisimulation

$(H_1,$  $)$

Weak Bisimulation

$(H_1,$  $) \longrightarrow (S_1, pt_1,$  $)$

Weak Bisimulation

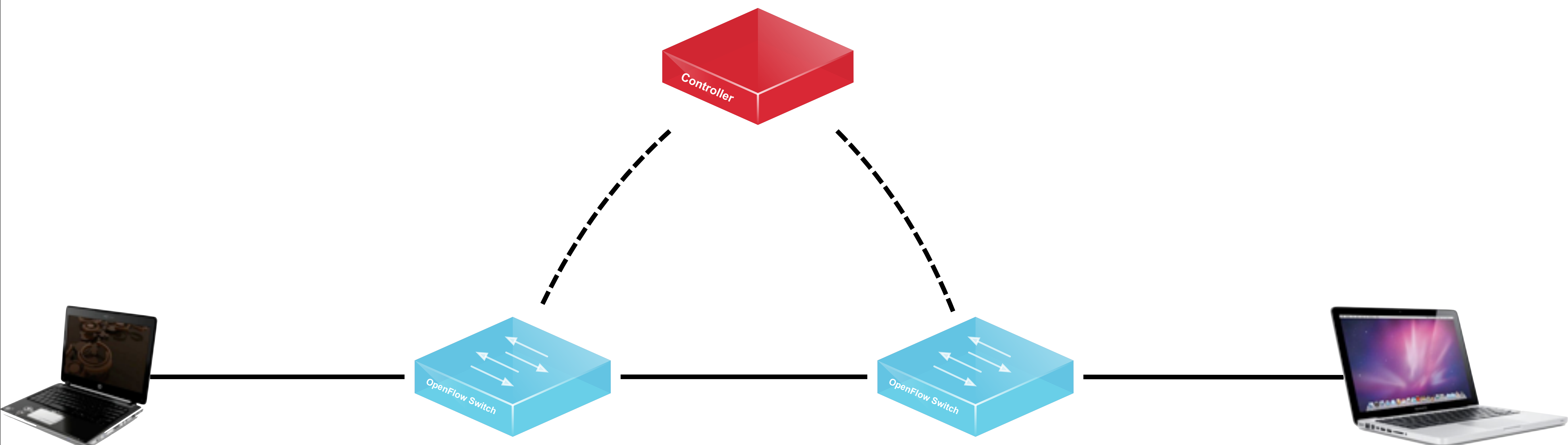
$(H_1,$  $) \longrightarrow (S_1, pt_1,$  $) \longrightarrow (S_2, pt_1,$  $)$

Weak Bisimulation

$(H_1,$  $) \longrightarrow (S_1, pt_1,$  $) \longrightarrow (S_2, pt_1,$  $) \longrightarrow (H_2,$  $)$

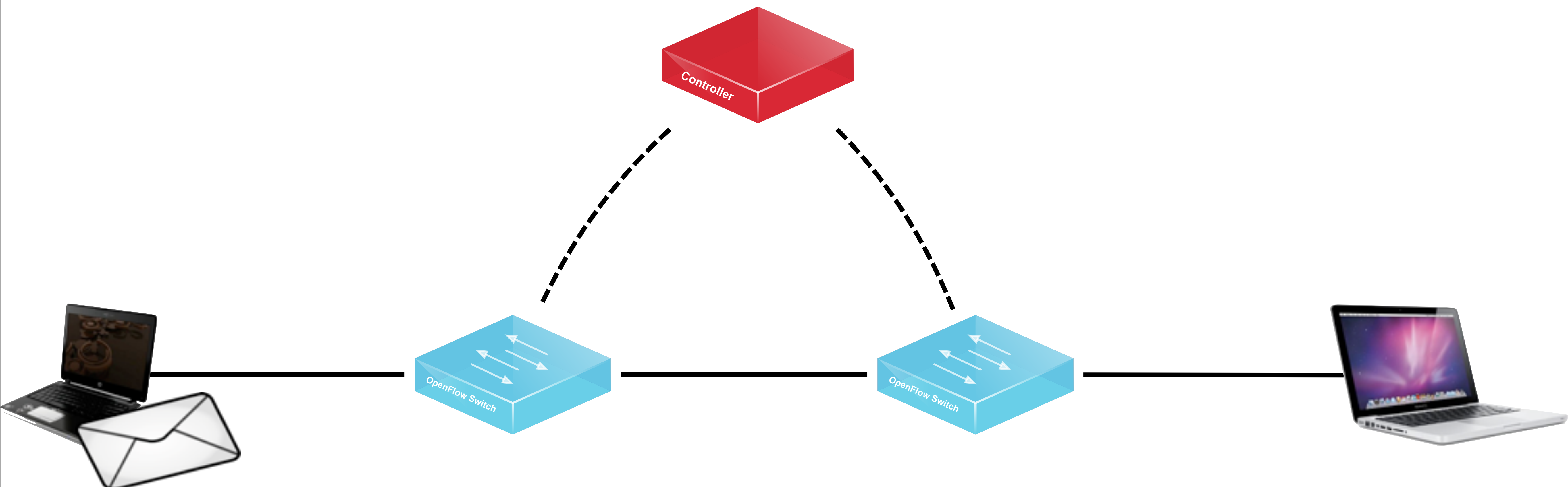
Weak Bisimulation

$(H_1,$  $) \longrightarrow (S_1, pt_1,$  $) \longrightarrow (S_2, pt_1,$  $) \longrightarrow (H_2,$  $)$



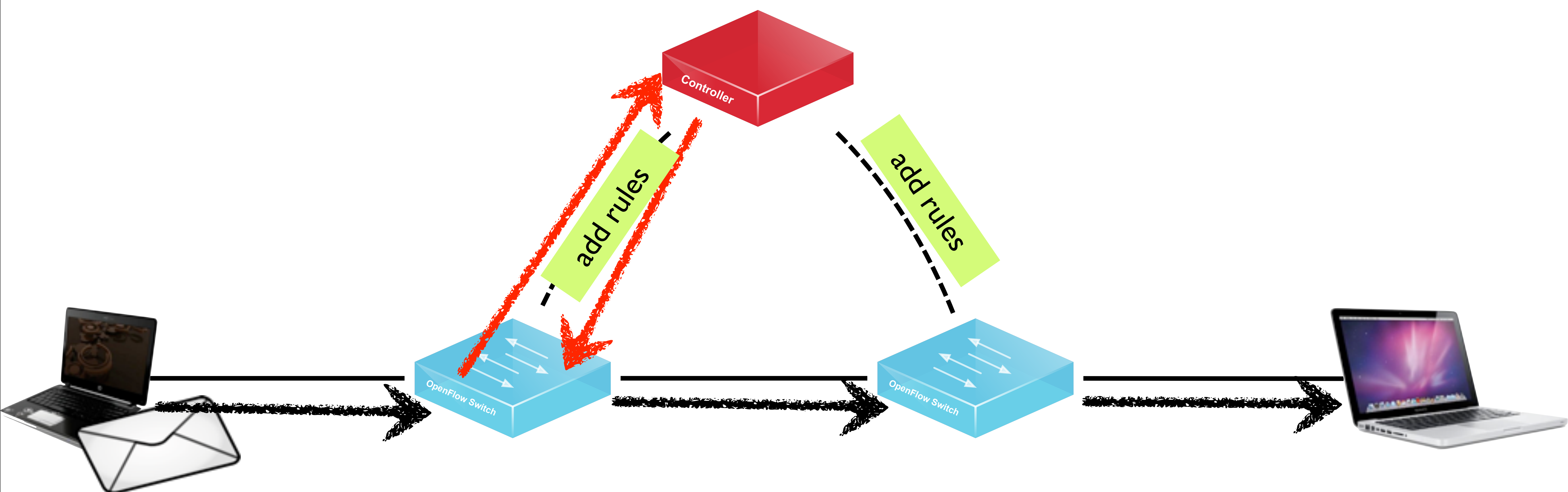
Weak Bisimulation

$(H_1,$  $) \longrightarrow (S_1, pt_1,$  $) \longrightarrow (S_2, pt_1,$  $) \longrightarrow (H_2,$  $)$

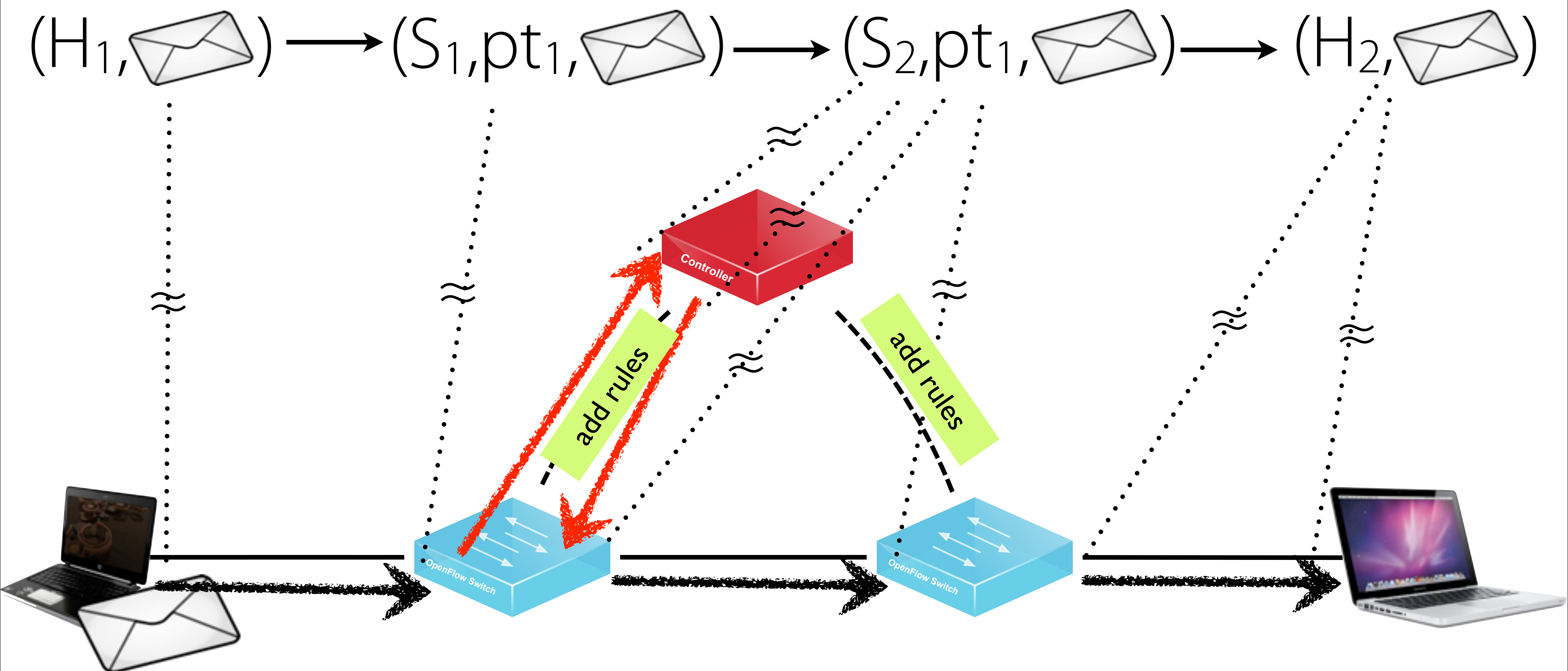


Weak Bisimulation

$(H_1,$  $) \longrightarrow (S_1, pt_1,$  $) \longrightarrow (S_2, pt_1,$  $) \longrightarrow (H_2,$  $)$



Weak Bisimulation



Theorem: NetCore abstract semantics is weakly bisimilar to Featherweight OpenFlow + NetCore controller

Parameterized Weak Bisimulation

Invariants

- *Safety:* at all times, the rules installed on switches are a *subset* of the controller function
- *Liveness:* the controller eventually processes all packets diverted to it by switches

Theorem

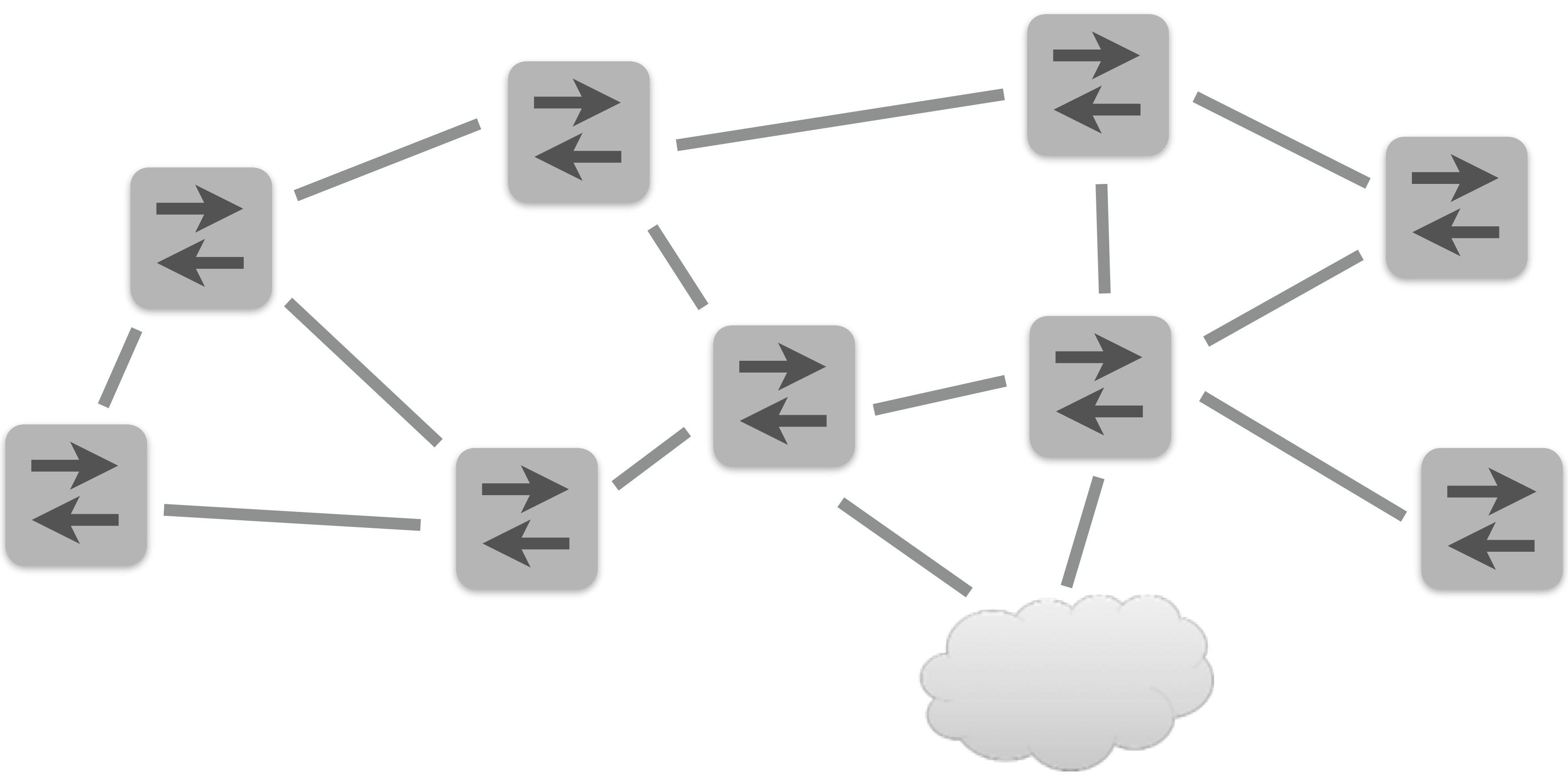
```
Module RelationDefinitions :=
  FwOF.FwOFRelationDefinitions.Make (AtomsAndController).
  ...
Theorem fwof_abst_weak_bisim :
  weak_bisimulation
  concreteStep
  abstractStep
  bisim_relation.
```

Equational Reasoning

[POPL '14]

Network Features

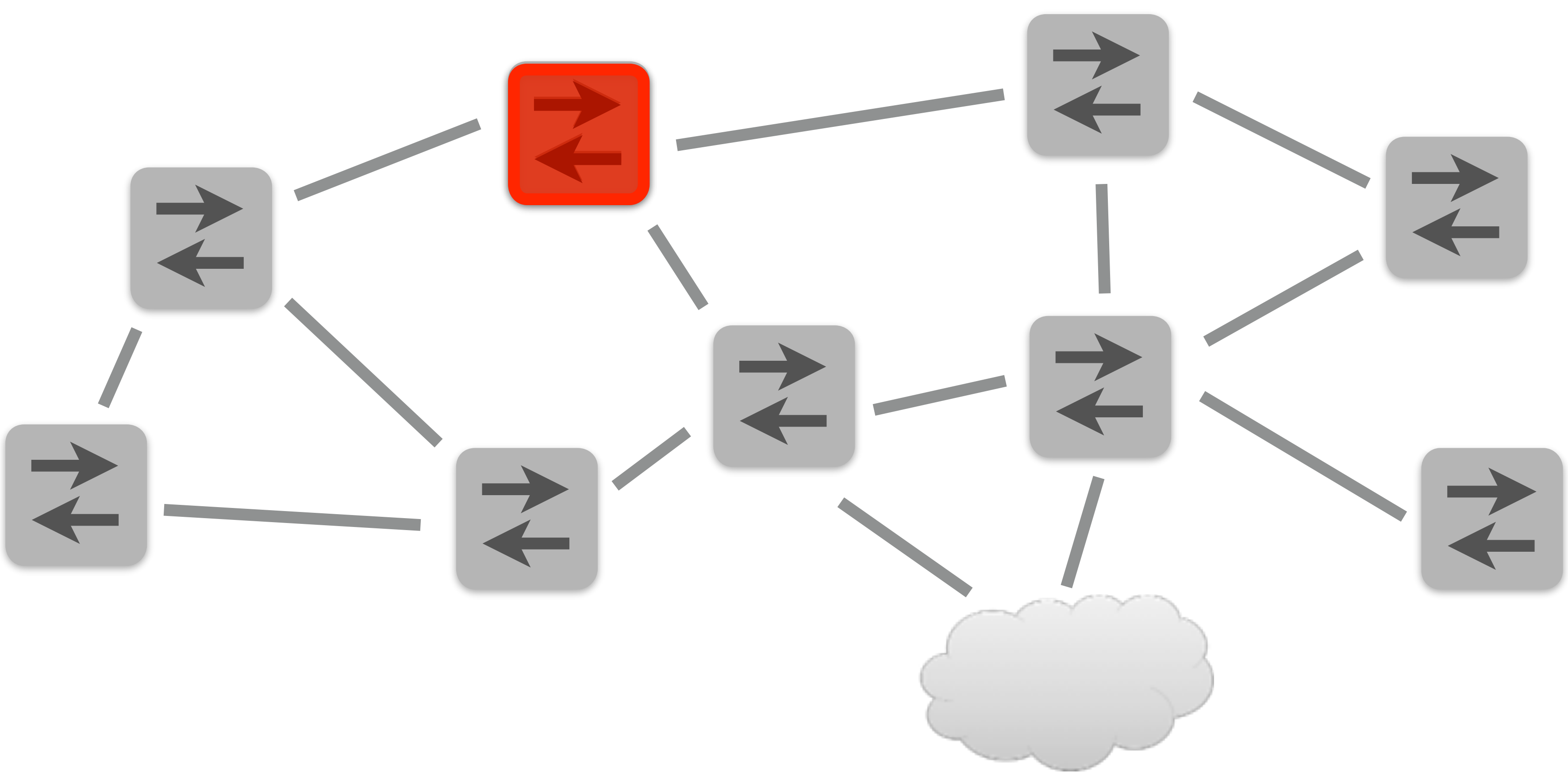
What network features should a logical framework model?*



*Focusing just on packet forwarding

Network Features

What network features should a logical framework model?*

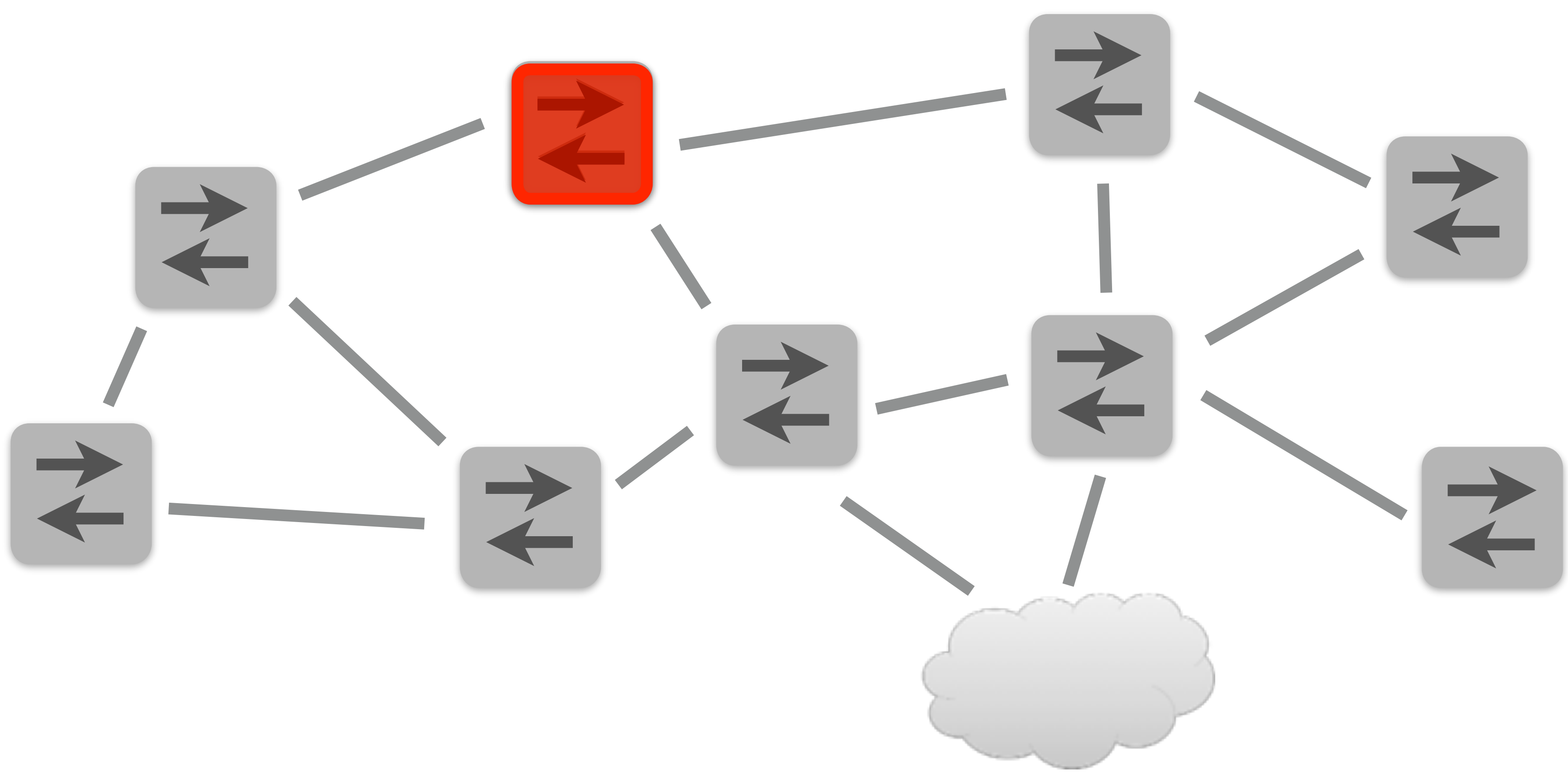


*Focusing just on packet forwarding

Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations

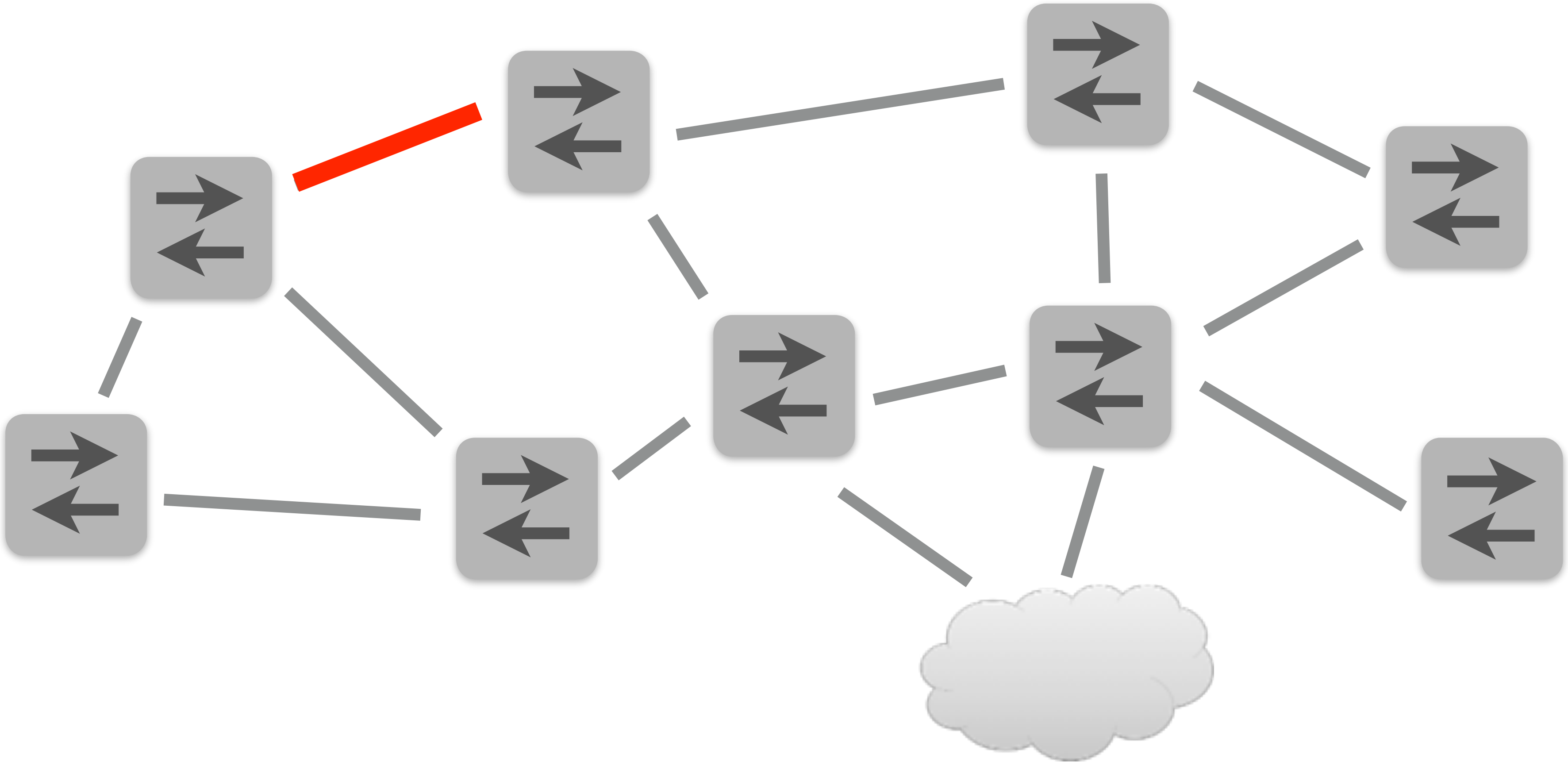


*Focusing just on packet forwarding

Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations

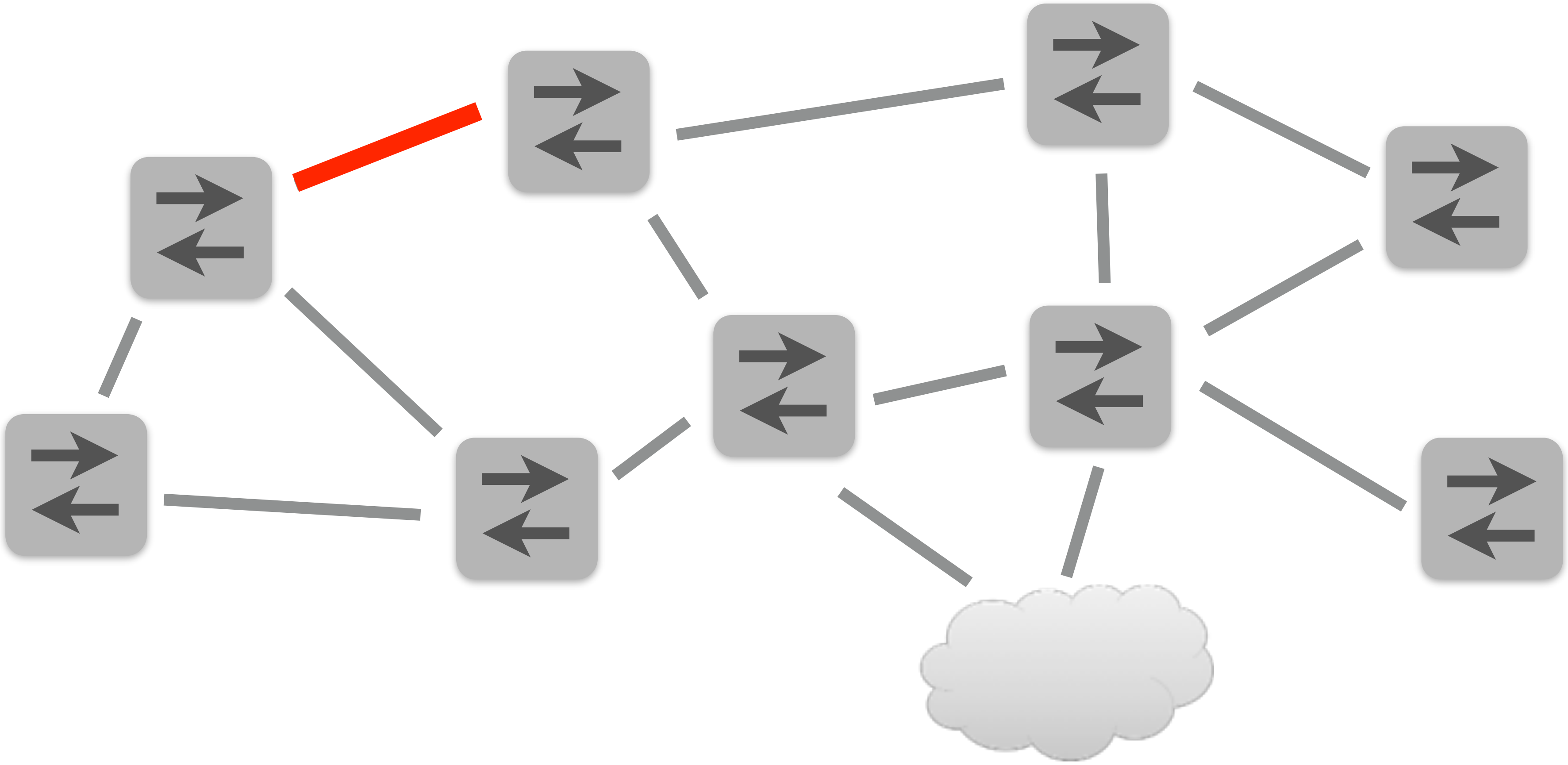


*Focusing just on packet forwarding

Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction

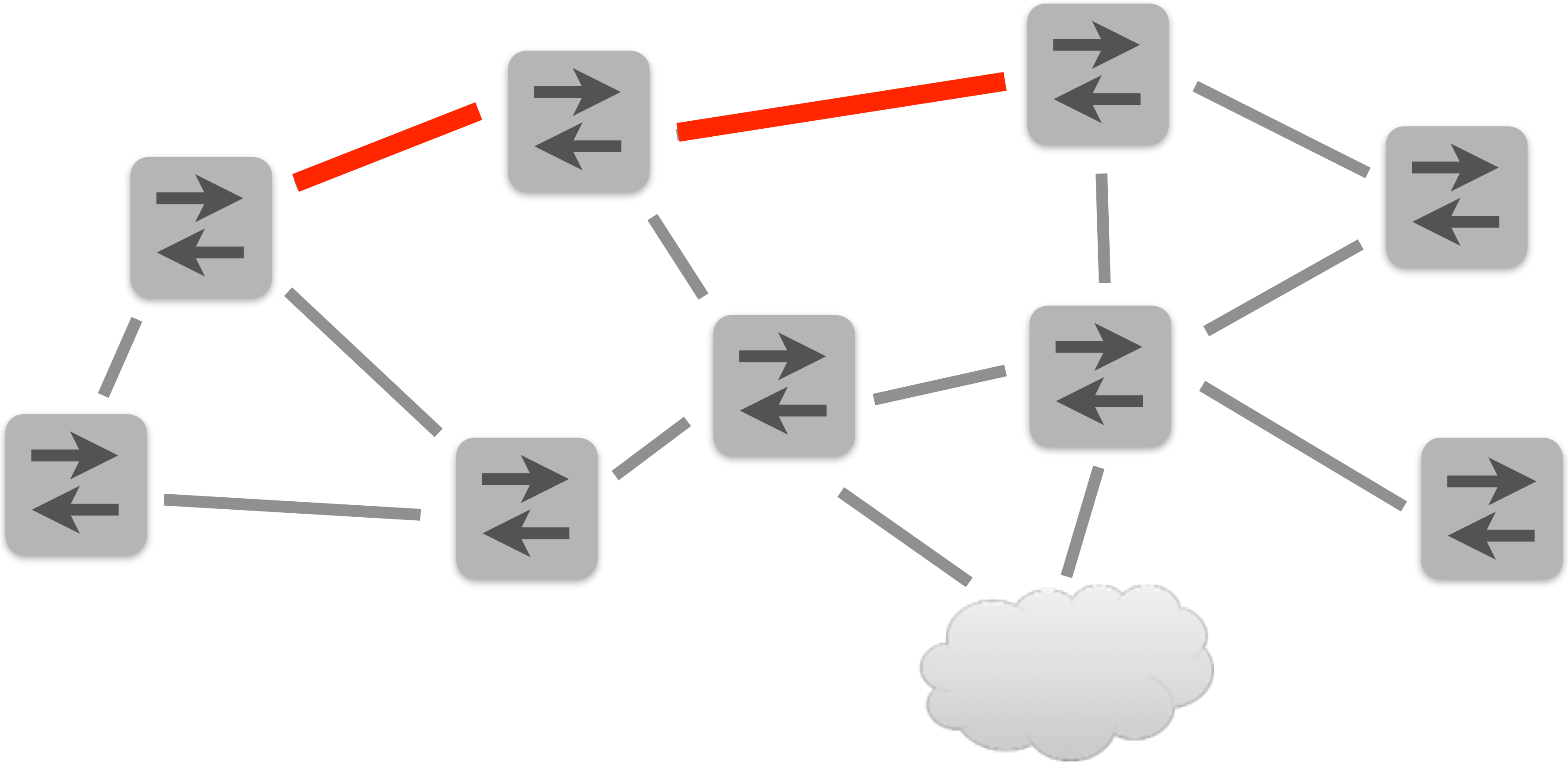


*Focusing just on packet forwarding

Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction

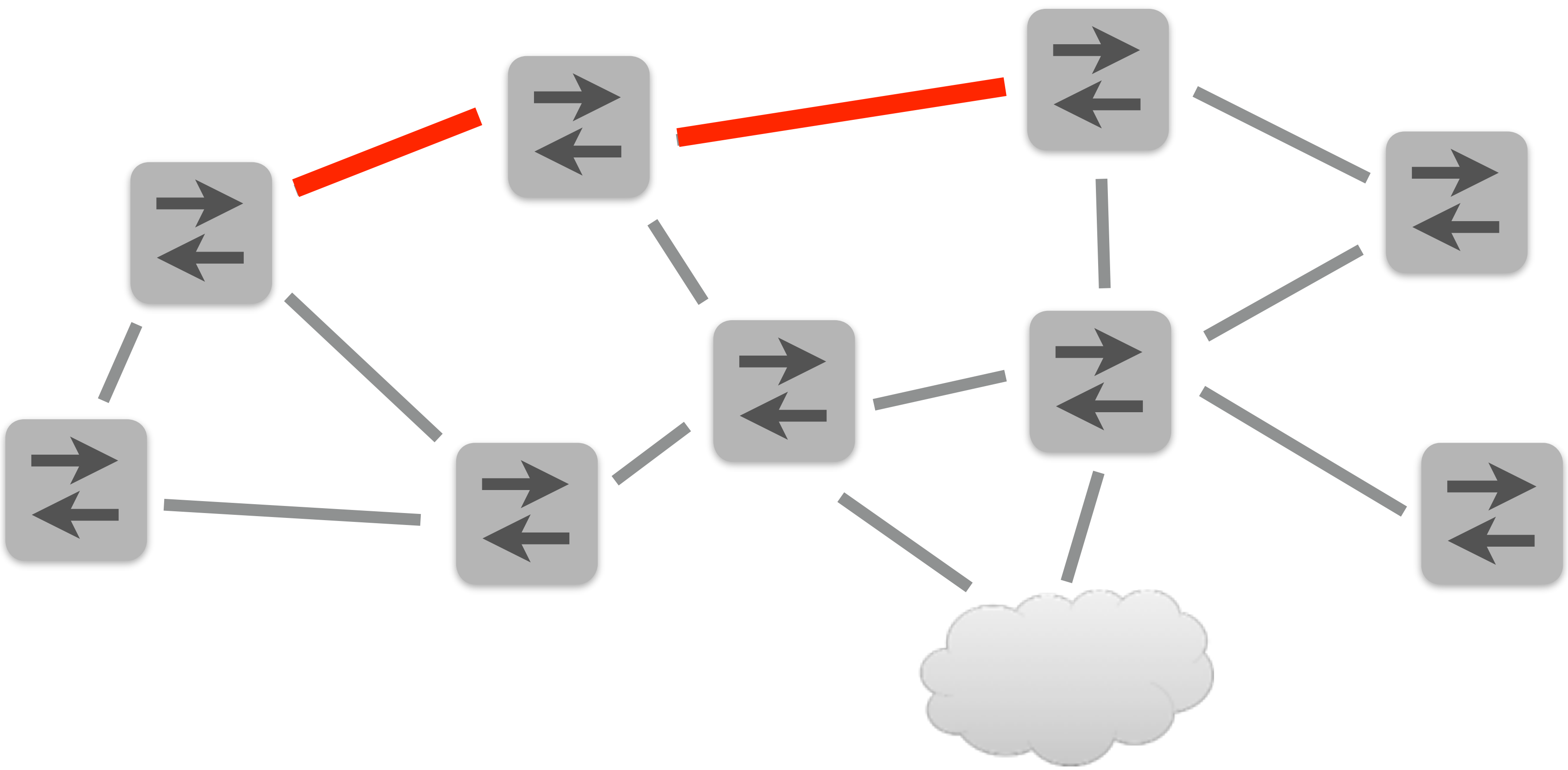


*Focusing just on packet forwarding

Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation

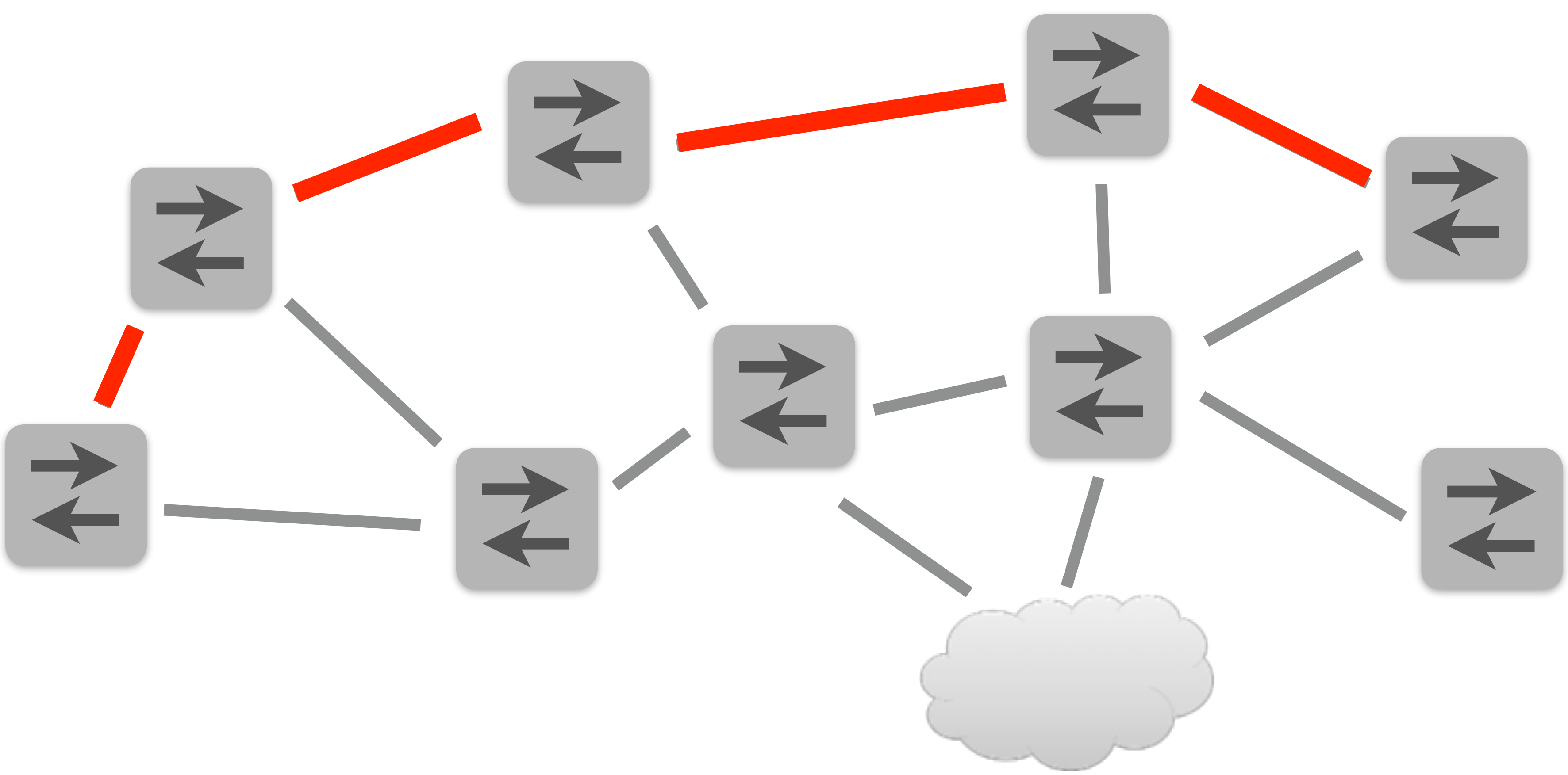


*Focusing just on packet forwarding

Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation

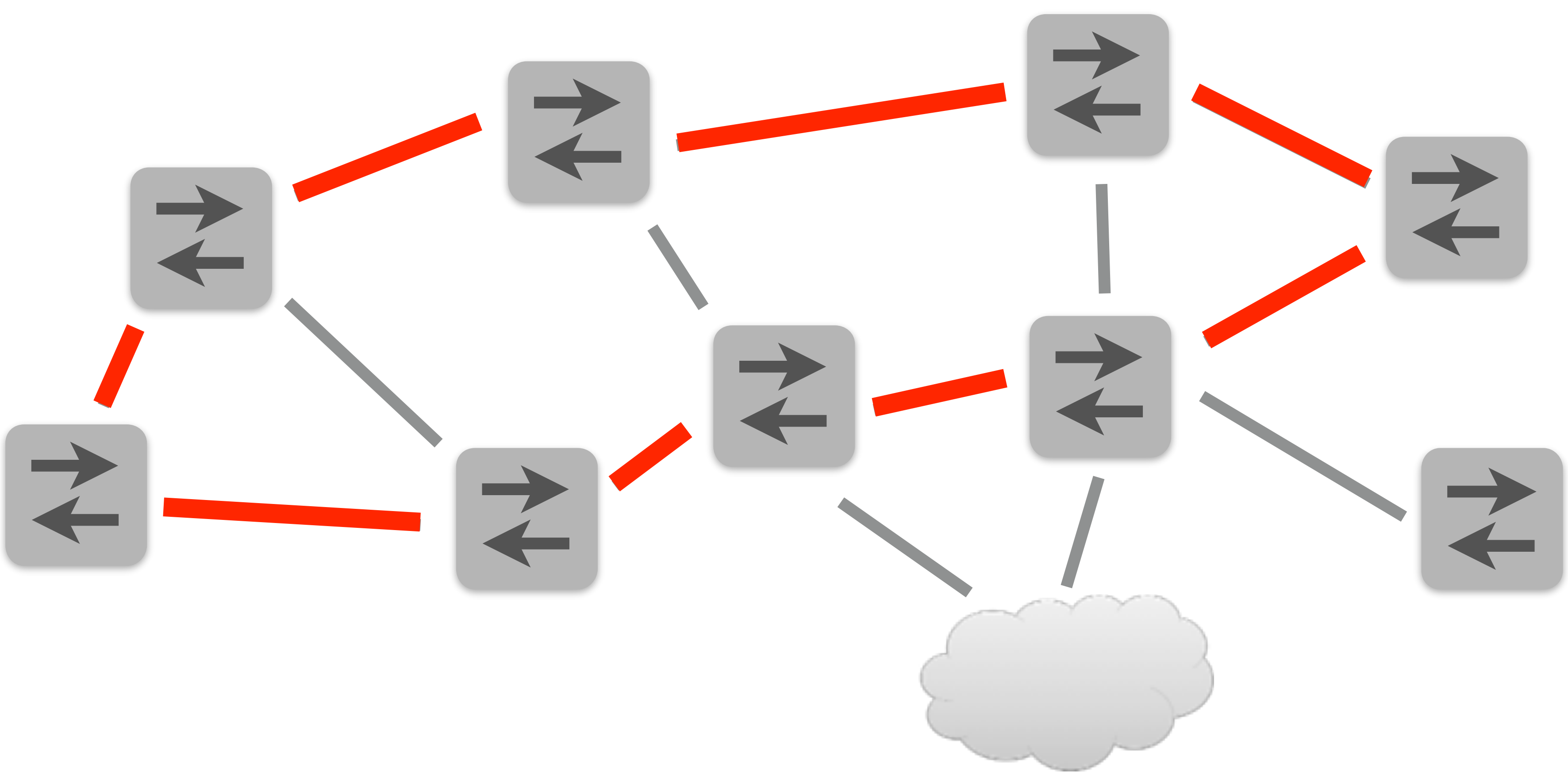


*Focusing just on packet forwarding

Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation

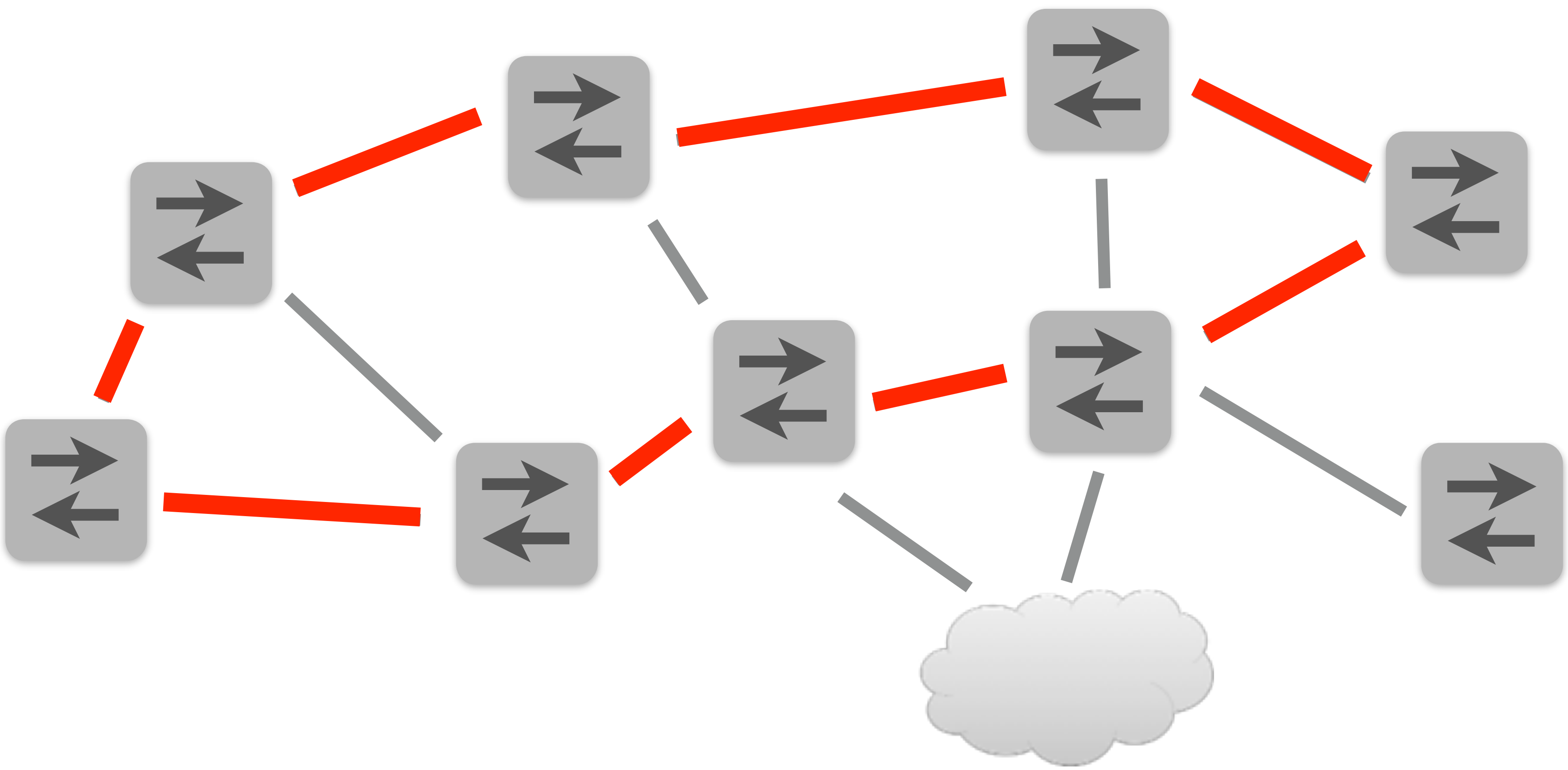


*Focusing just on packet forwarding

Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation
- Path union

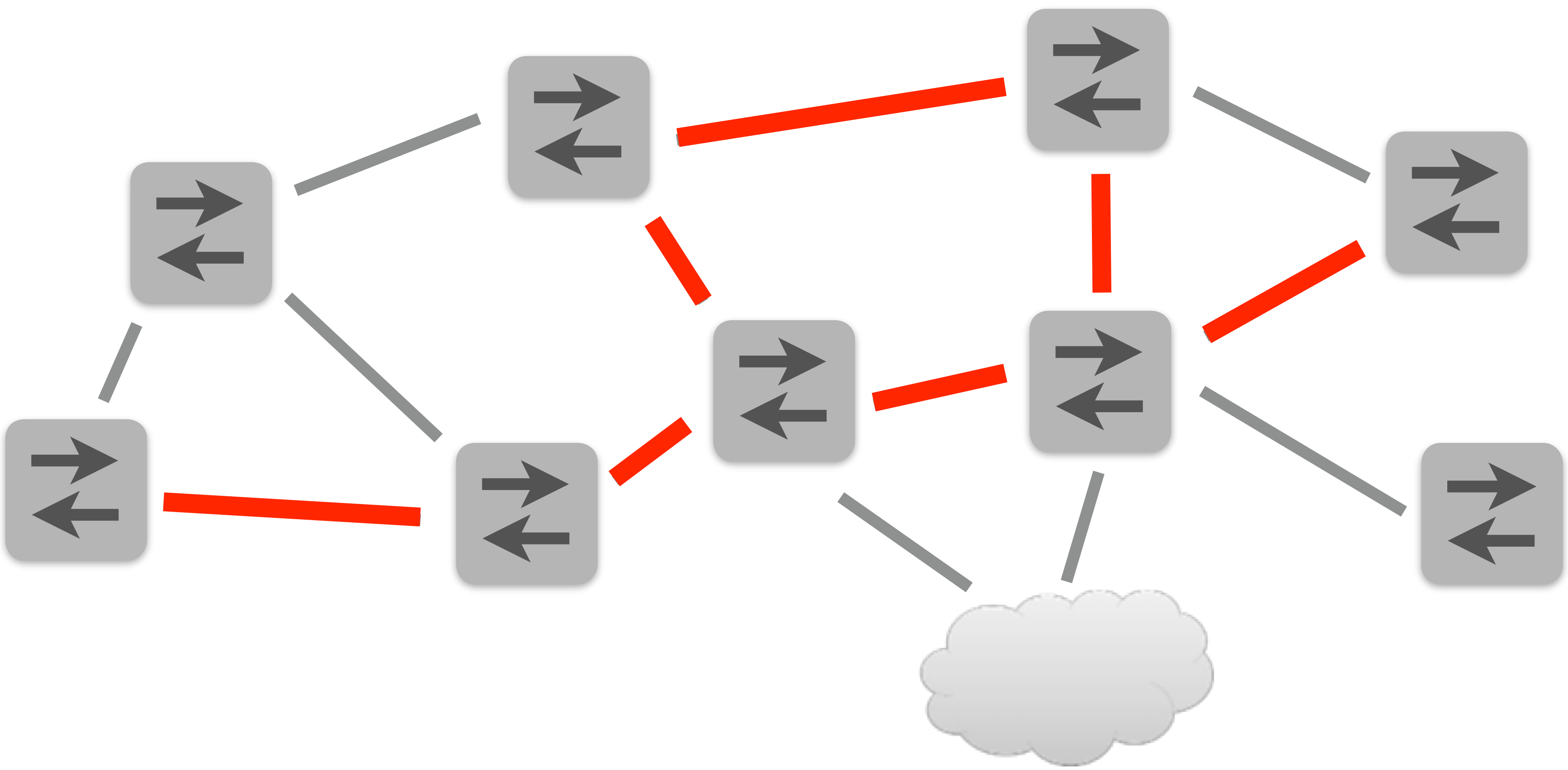


*Focusing just on packet forwarding

Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation
- Path union

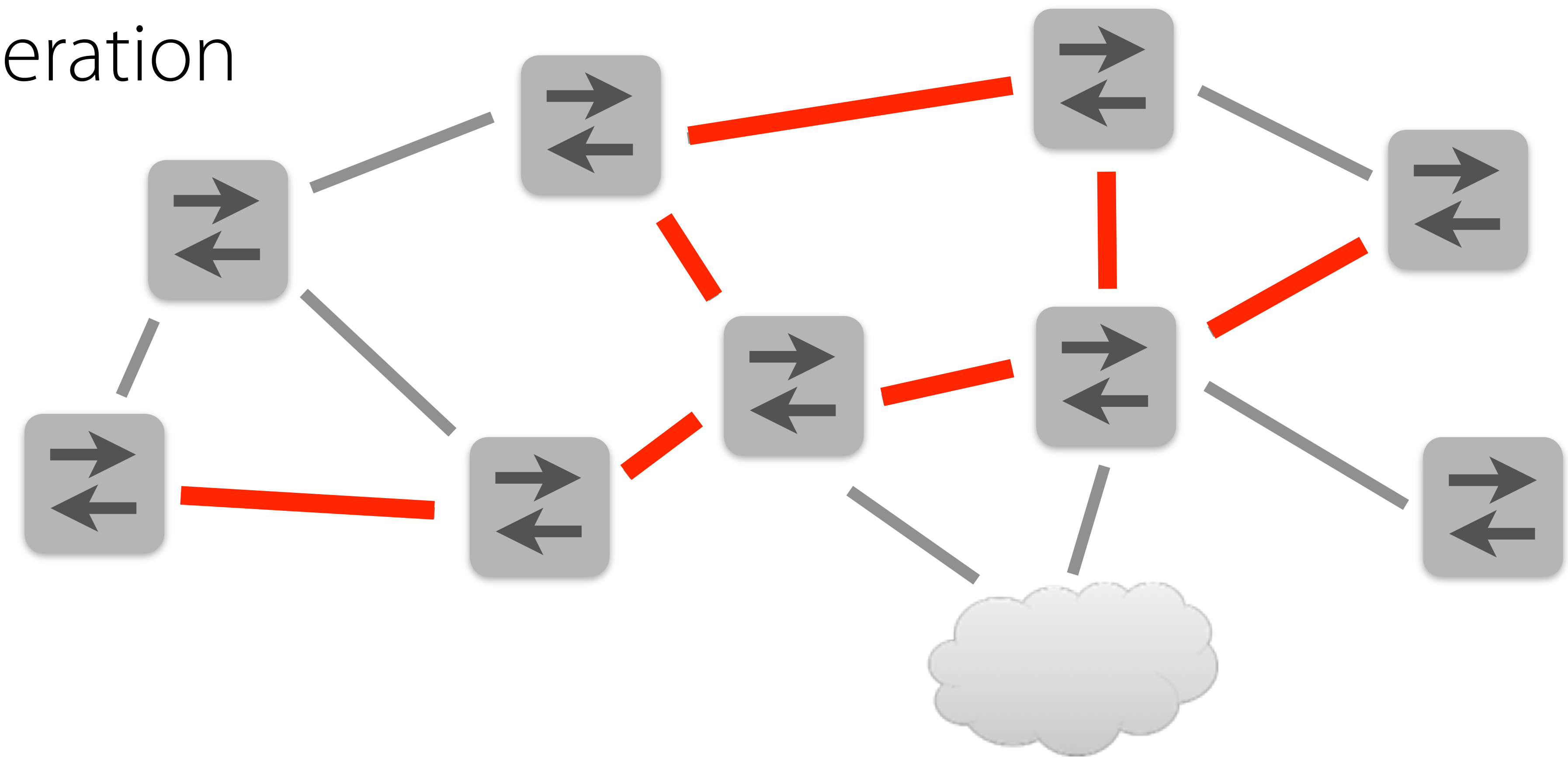


*Focusing just on packet forwarding

Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation
- Path union
- Path iteration



*Focusing just on packet forwarding

NetKAT

f ::= **switch** | **inport** | **srcmac** | **dstmac** | ...

v ::= **0** | **1** | **2** | **3** | ...

a,b,c ::= **true** (* true *)

| **false** (* false *)

| f **=** v (* test *)

| a_1 **|** a_2 (* disjunction *)

| a_1 **&** a_2 (* conjunction *)

| **!** a (* negation *)

p,q,r ::= **filter** a (* filter *)

| f **:=** v (* modification *)

| p_1 **|** p_2 (* union *)

| p_1 **;** p_2 (* sequence *)

| p^* (* iteration *)

| **dup** (* duplication *)

NetKAT

f ::= **switch** | **inport** | **srcmac** | **dstmac** | ...

v ::= **0** | **1** | **2** | **3** | ...

a,b,c ::= **true** (* true *)

| **false** (* false *)

| f **=** v (* test *)

| a_1 **|** a_2 (* disjunction *)

| a_1 **&** a_2 (* conjunction *)

| **!** a (* negation *)

p,q,r ::= **filter** a (* filter *)

| f **:=** v (* modification *)

| p_1 **|** p_2 (* union *)

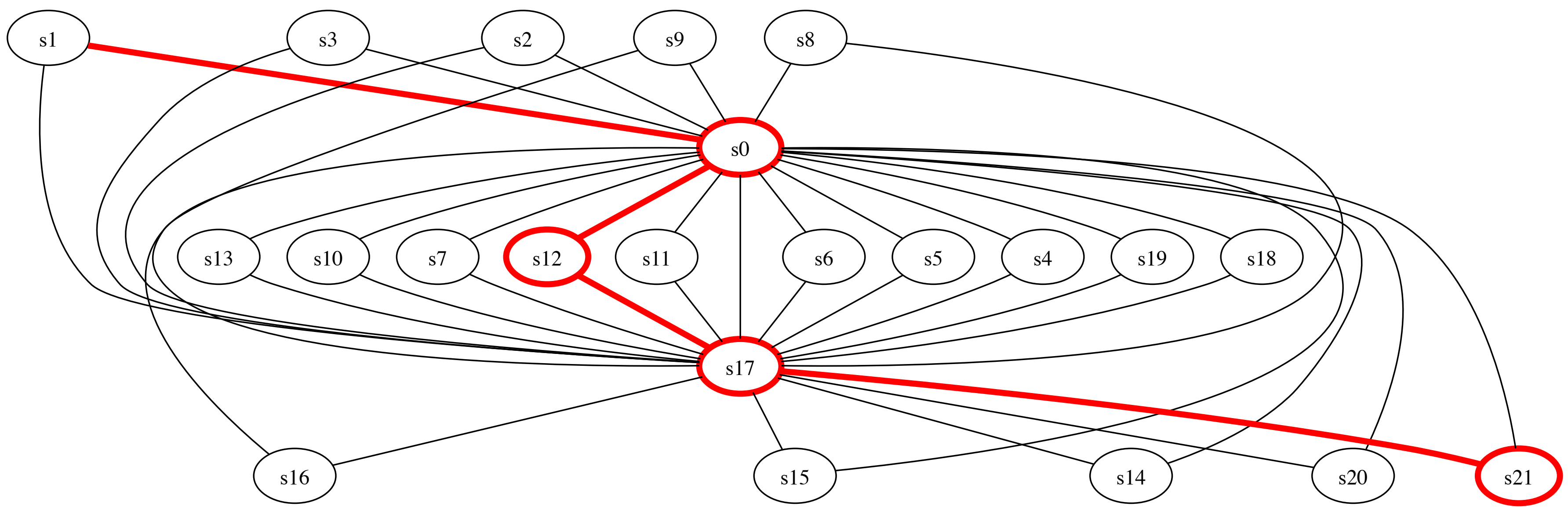
| p_1 **;** p_2 (* sequence *)

| p^* (* iteration *)

| **dup** (* duplication *)

if a **then** p_1 **else** p_2 ≜ (**filter** a**;** p_1) **|** (**filter !**a**;** p_2)

Example: Reachability



Given:

- Ingress predicate i
- Egress predicate e
- Topology t
- Switch program p

Test:

filter i**; dup;** (p**; dup;** t)***; filter** e ~ **filter false**

Kleene Algebra with Tests

The design of NetKAT is not an accident!

Its foundation rests upon canonical mathematical structure:

- Regular operators (**|**, **;**, and *****) encode paths through topology
- Boolean operators (**&**, **|**, and **!**) encode switch tables

This is called a ***Kleene Algebra with Tests (KAT)*** [Kozen '96]

KAT has an accompanying proof system for showing equivalences of the form p ~ q

Kleene Algebra with Tests

The design of NetKAT is not an accident!

Its foundation rests upon canonical mathematical structure:

- Regular operators (**|**, **;**, and *****) encode paths through topology
- Boolean operators (**&**, **|**, and **!**) encode switch tables

This is called a ***Kleene Algebra with Tests (KAT)*** [Kozen '96]

KAT has an accompanying proof system for showing equivalences of the form p ~ q

Theorems

- *Soundness*: programs related by the axioms are equivalent
- *Completeness*: equivalent programs are related by the axioms
- *Decidabilty*: there is an algorithm for deciding equivalence

NetKAT Equational Theory

Kleene Algebra

p | (q | r) ~ (p | q) | r

p | q ~ q | p

p | **filter false** ~ p

p | p ~ p

p **;** (q **;** r) ~ (p **;** q) **;** r

p; (q | r) ~ p **;** q | p **;** r

(p | q) **;** r ~ p **;** r | q **;** r

filter true ; p ~ p

p ~ p **; filter true**

filter false ; p ~ **filter false**

p **; filter false** ~ **filter false**

filter true | p **;** p^* ~ p^*

filter true | p^* **;** p ~ p^*

p | q **;** r | r ~ r $\Rightarrow$ p^* **;** q | r ~ r

p | q **;** r | q ~ q $\Rightarrow$ p **;** r^* | q ~q

Boolean Algebra

a | (b **&** c) ~ (a | b) **&** (a | c)

a | **true** ~ **true**

a | **!** a ~ **true**

a **&** b ~ b **&** a

a **& !**a ~ **false**

a **&** a ~ a

Packet Algebra

f **:=** n**;** f′ **:=** n′ ~ f′ **:=** n′ ; f **:=** n if $f \neq f'$

f **:=** n**;** f′ **=** n′ ~ f′ **=** n′; f **:=** n if $f \neq f'$

f **:=** n**;** f **=** n ~ f **:=** n

f **=** n**;** f **:=** n ~ f **=** n

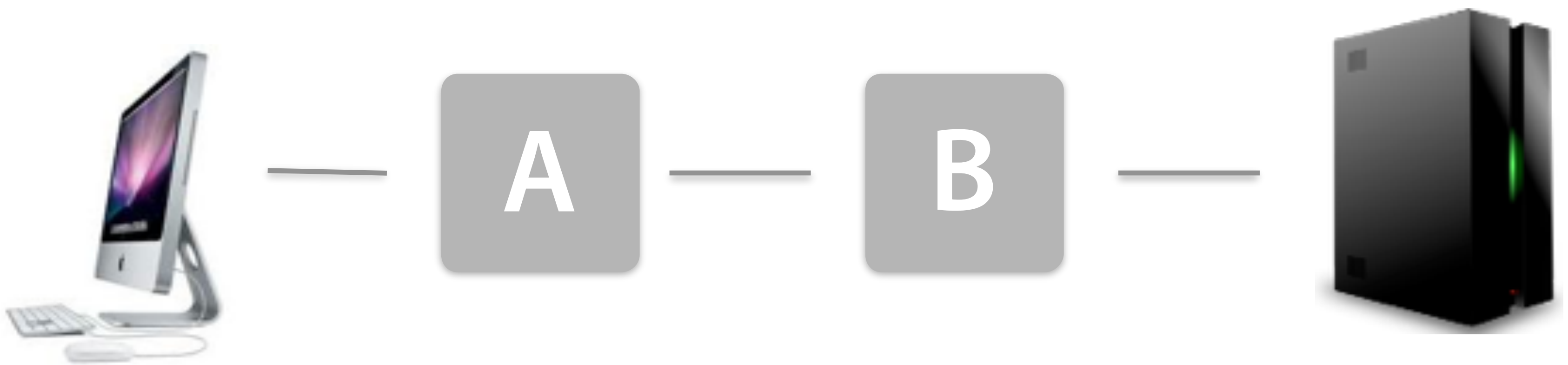
f **:=** n**;** f := n′ ~ f **:=** n′

f **=** n; f **=** n′ ~ **filter false** if $n \neq n'$

dup ; f = n ~ f = n **; dup**

Application: Optimization

Given a program and a topology:

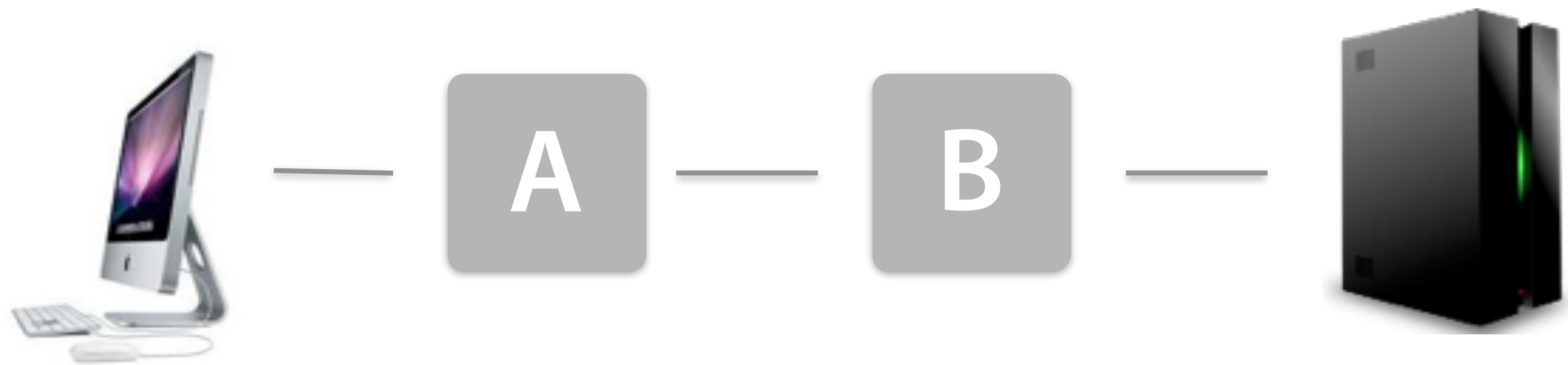


Want to be able to answer questions like:

“Will my network behave the same if I put the firewall rules on A, or on switch B (or both)?”

Application: Optimization

Given a program and a topology:



Want to be able to answer questions like:

> "Will my network behave the same if I put the firewall rules on A, or on switch B (or both)?"

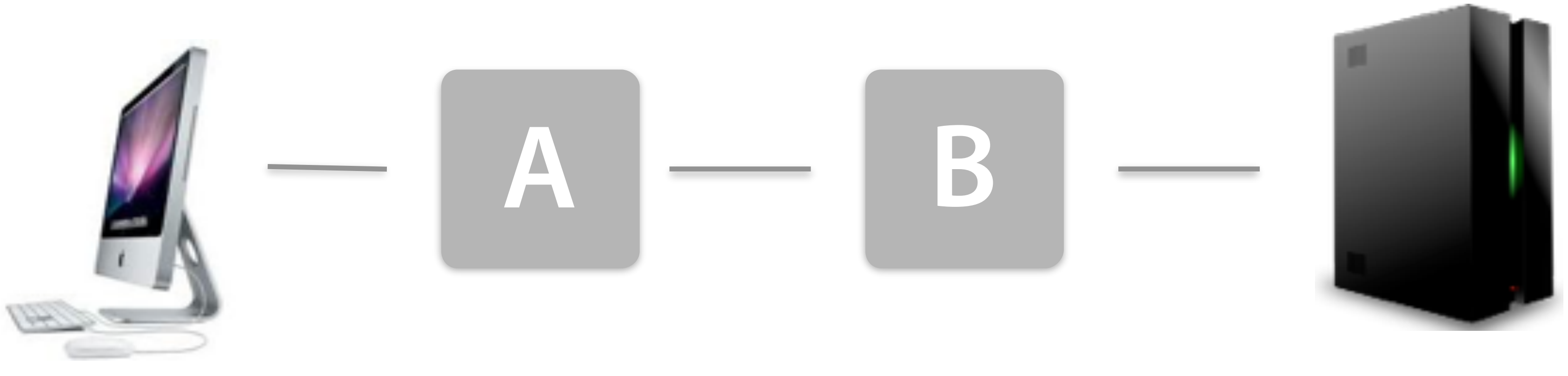
Formally, does the following equivalence hold?

(**filter** switch = A **;** firewall; routing) **|** (**filter** switch = B**;** routing)

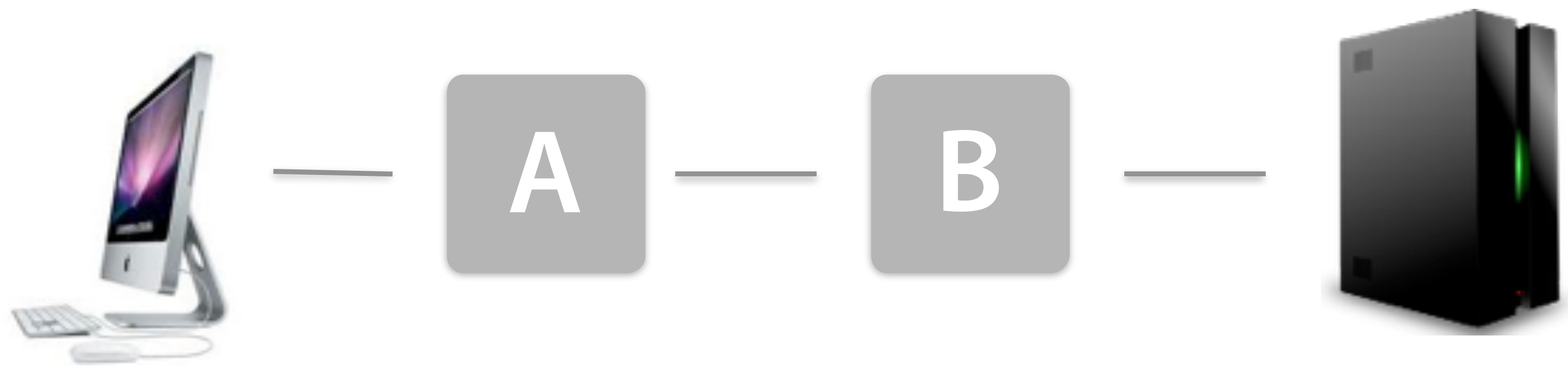
~

(**filter** switch = A **;** routing) **|** (**filter** switch = B**;** firewall**;** routing)

Optimization Proof



Optimization Proof



$in; (p_A; t)^*; p_A; out$

$\equiv \{$ definition in, out, and p_A $\}$

$s_A; \textsc{ssh}; ((s_A; \neg\textsc{ssh}; p + s_B; p); t)^*; p_A; s_B$

$\equiv \{$ KAT-Invariant $\}$

$s_A; \textsc{ssh}; ((s_A; \neg\textsc{ssh}; p + s_B; p); t; \textsc{ssh})^*; p_A; s_B$

$\equiv \{$ KA-Seq-Dist-R $\}$

$s_A; \textsc{ssh}; (s_A; \neg\textsc{ssh}; p; t; \textsc{ssh} + s_B; p; t; \textsc{ssh})^*; p_A; s_B$

$\equiv \{$ KAT-Commute $\}$

$s_A; \textsc{ssh}; (s_A; \neg\textsc{ssh}; \textsc{ssh}; p; t + s_B; p; t; \textsc{ssh})^*; p_A; s_B$

$\equiv \{$ BA-Contra $\}$

$s_A; \textsc{ssh}; (s_A; \text{drop}; p; t + s_B; p; t; \textsc{ssh})^*; p_A; s_B$

$\equiv \{$ KA-Seq-Zero, KA-Zero-Seq, KA-Plus-Comm, KA-Plus-Zero $\}$

$s_A; \textsc{ssh}; (s_B; p; t; \textsc{ssh})^*; p_A; s_B$

$\equiv \{$ KA-Unroll-L $\}$

$s_A; \textsc{ssh}; (\text{id} + (s_B; p; t; \textsc{ssh}); (s_B; p; t; \textsc{ssh})^*); p_A; s_B$

$\equiv \{$ KA-Seq-Dist-L and KA-Seq-Dist-R $\}$

$(s_A; \textsc{ssh}; p_A; s_B)+$

$(s_A; \textsc{ssh}; s_B; p; t; \textsc{ssh}; (s_B; p; t; \textsc{ssh})^*; p_A; s_B)$

$\equiv \{$ KAT-Commute $\}$

$(s_A; s_B; \textsc{ssh}; p_A)+$

$(s_A; s_B; \textsc{ssh}; p; t; \textsc{ssh}; (s_B; p; t; \textsc{ssh})^*; p_A; s_B)$

$\equiv \{$ PA-Contra $\}$

$(\text{drop}; \textsc{ssh}; p_A)+$

$(\text{drop}; \textsc{ssh}; p; t; \textsc{ssh}; (s_B; p; t; \textsc{ssh})^*; p_A; s_B)$

$\equiv \{$ KA-Zero-Seq, KA-Plus-Idem $\}$

drop

$\equiv \{$ KA-Seq-Zero, KA-Zero-Seq, KA-Plus-Idem $\}$

$s_A; (p_B; t)^*; (\textsc{ssh}; \text{drop}; p + s_B; \text{drop}; p; s_B)$

$\equiv \{$ PA-Contra and BA-Contra $\}$

$s_A; (p_B; t)^*; (\textsc{ssh}; s_A; s_B; p + s_B; \textsc{ssh}; \neg\textsc{ssh}; p; s_B)$

$\equiv \{$ KAT-Commute $\}$

$s_A; (p_B; t)^*; (\textsc{ssh}; s_A; p; s_B + \textsc{ssh}; s_B; \neg\textsc{ssh}; p; s_B)$

$\equiv \{$ KA-Seq-Dist-L and KA-Seq-Dist-R $\}$

$s_A; (p_B; t)^*; \textsc{ssh}; (s_A; p + s_B; \neg\textsc{ssh}; p); s_B$

$\equiv \{$ KAT-Commute $\}$

$s_A; \textsc{ssh}; (p_B; t)^*; (s_A; p + s_B; \neg\textsc{ssh}; p); s_B$

$\equiv \{$ definition in, out, and p_B $\}$

$in; (p_B; t)^*; p_B; out$

Wrapping Up

Conclusion

- Networks are an promising area for applications of formal methods
- Software-defined networking is a new architecture that makes it easy to deploy formal verification tools
- Frenetic is a high-level language for programming networks and reasoning about their behavior:
  - Consistent updates
  - Machine-verified compiler and run-time system
  - Equational reasoning in NetKAT

Thank you!

The Team

Carolyn Anderson

Nate Foster

Arjun Guha

Jean-Baptiste Jeannin

Dexter Kozen

Mark Reitblatt

Jen Rexford

Cole Schlesinger

David Walker

Carbon

frenetic

http://frenetic-lang.org

SCHLOSS DAGSTUHL
Leibniz-Zentrum für Informatik

Formal Foundations for Networking

Seminar 30-0613, Feb 2015

- Nikolaj Bjørner (MSR)
- Nate Foster (Cornell)
- Brighten Godfrey (UIUC)
- Pamela Zave (AT&T)