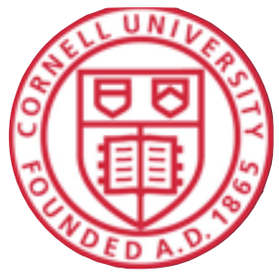
The Frenetic Network Controller

Arjun Guha, Nate Foster, Mark Reitblatt, Cole Schlesinger,
and others:
www.github.com/frenetic-lang/frenetic/contributors

UMass Amherst



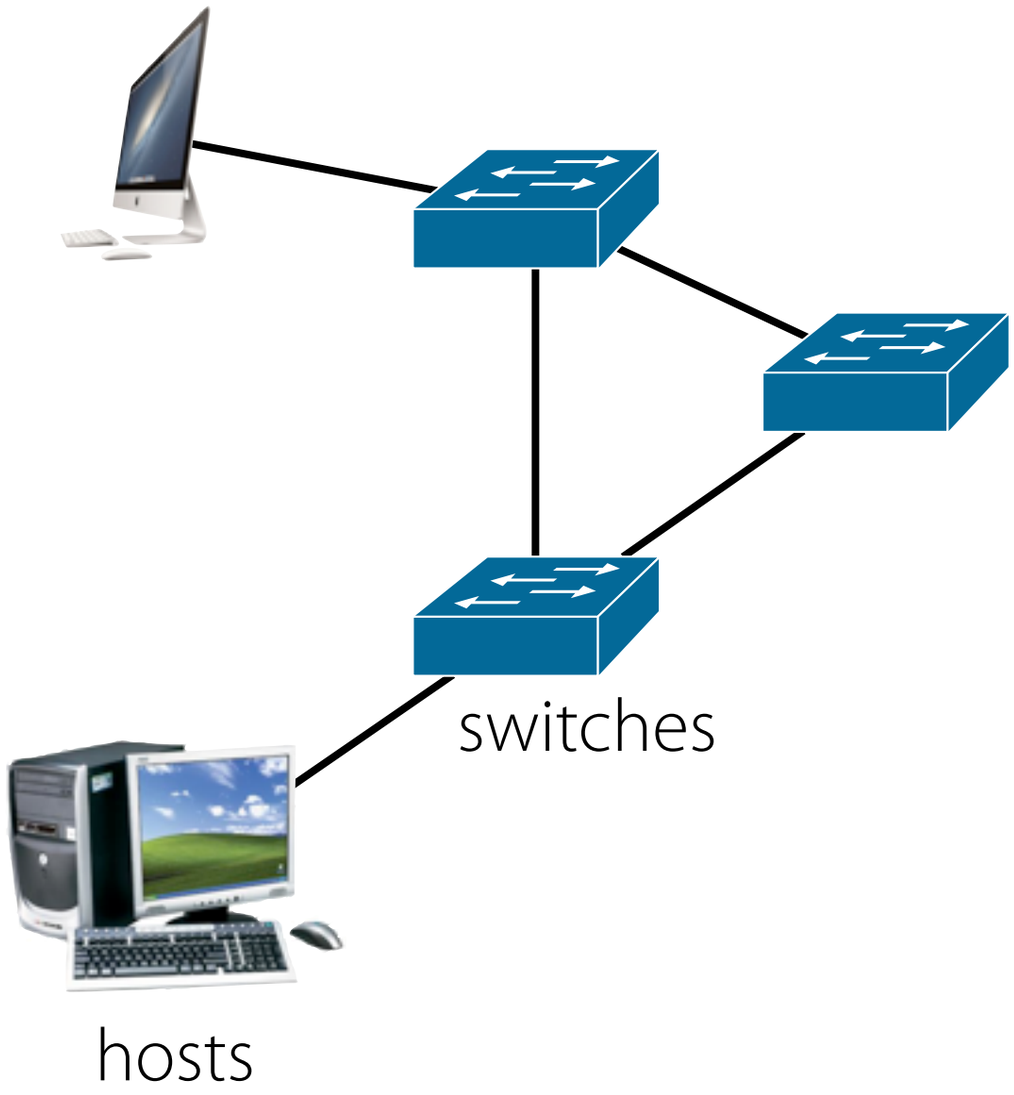
Cornell

Princeton

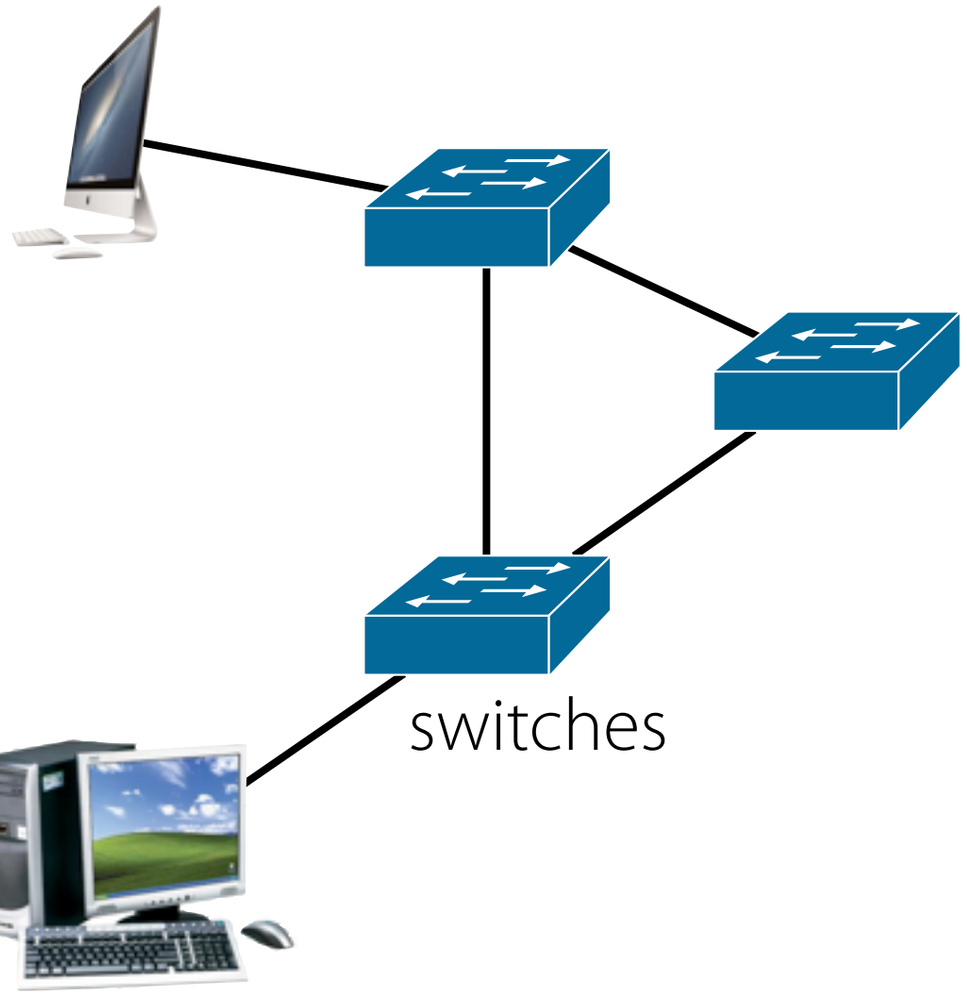
networks today

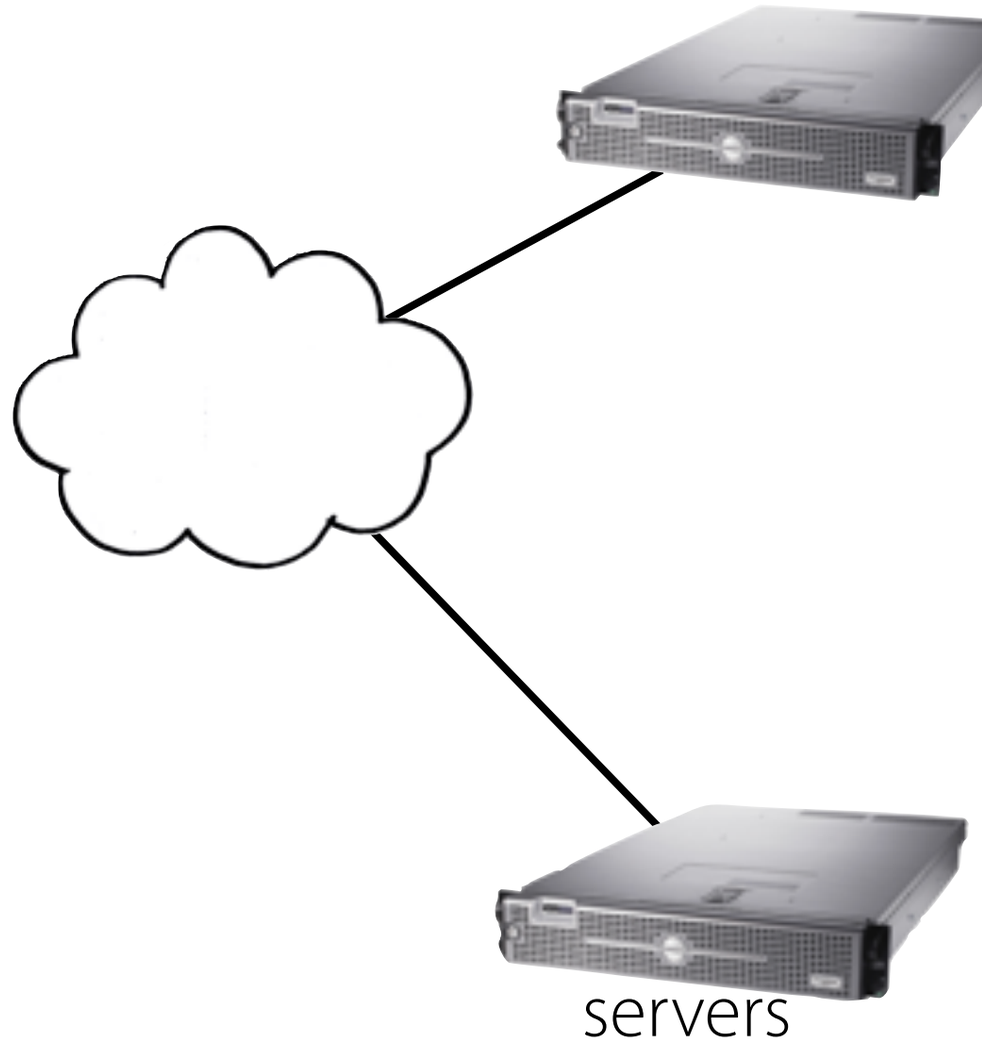
hosts

networks today

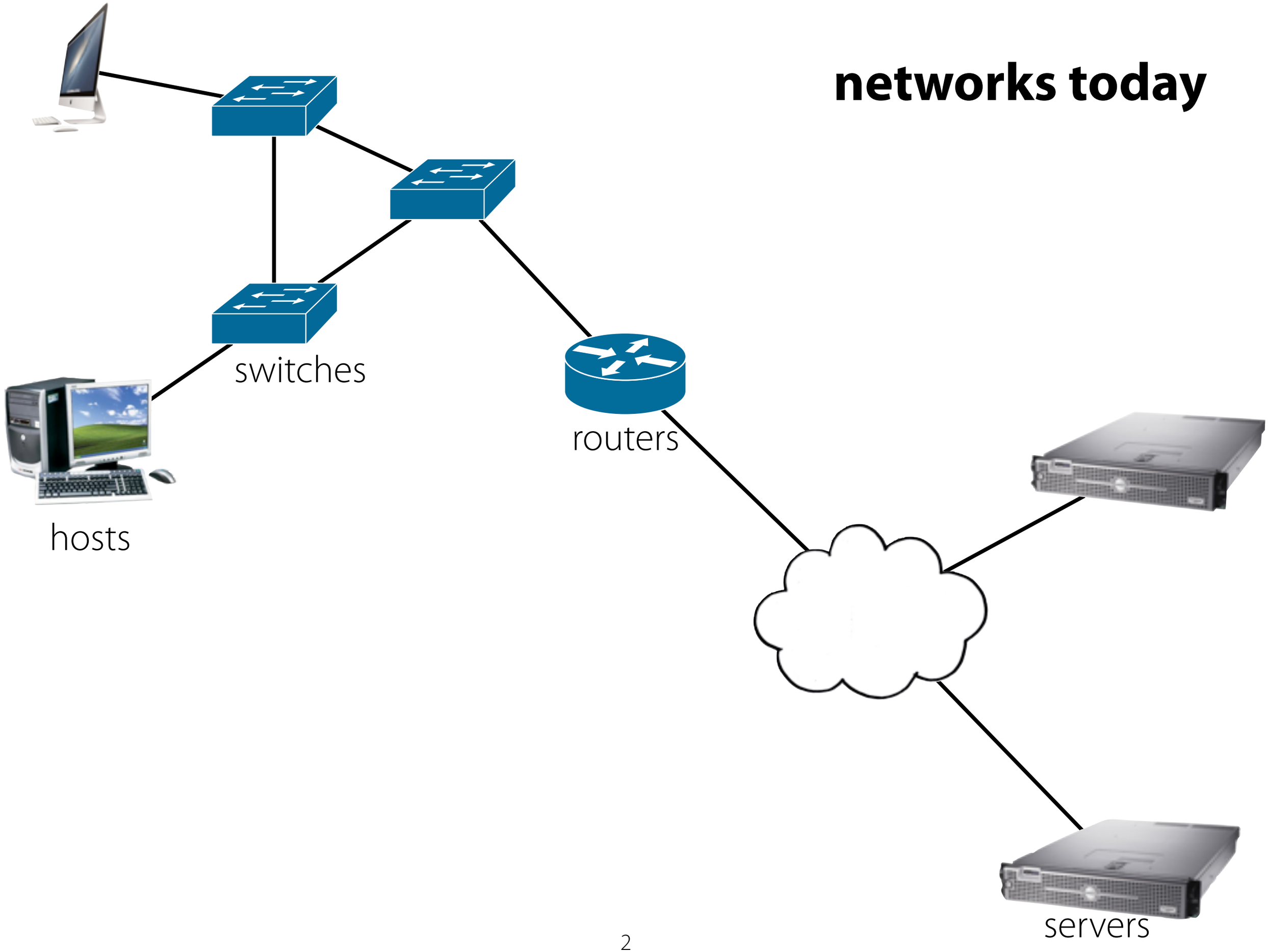


networks today

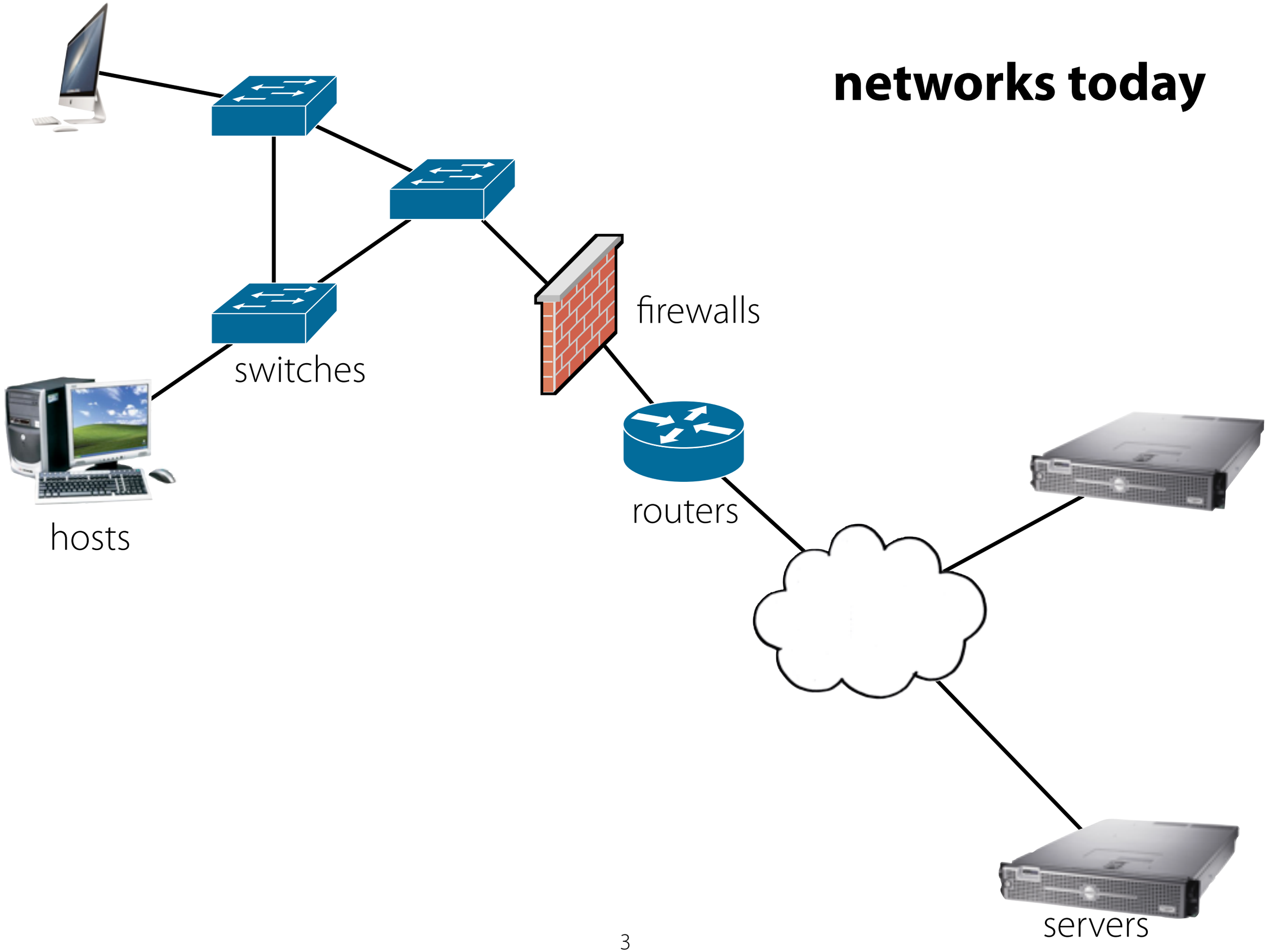




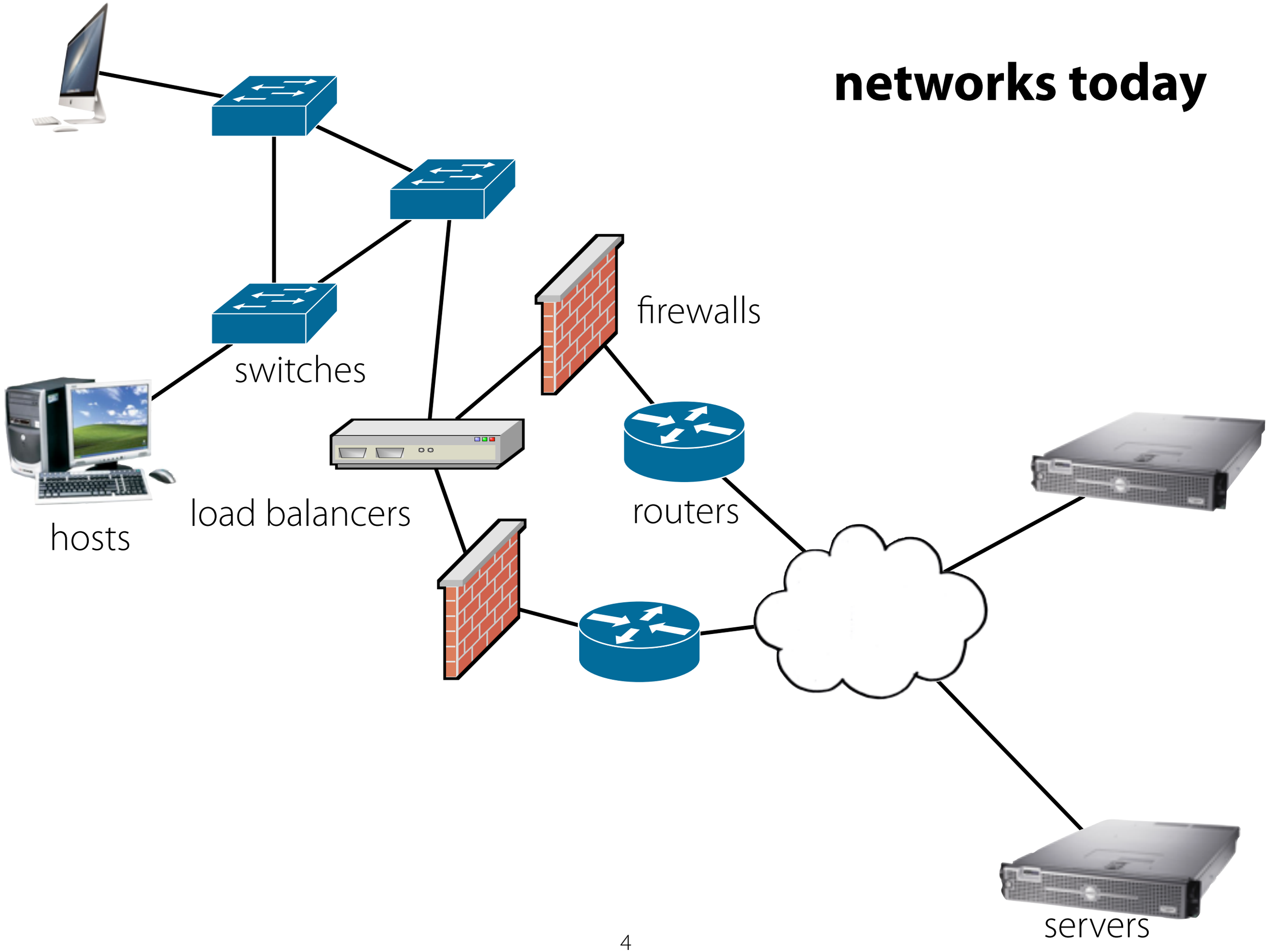
networks today



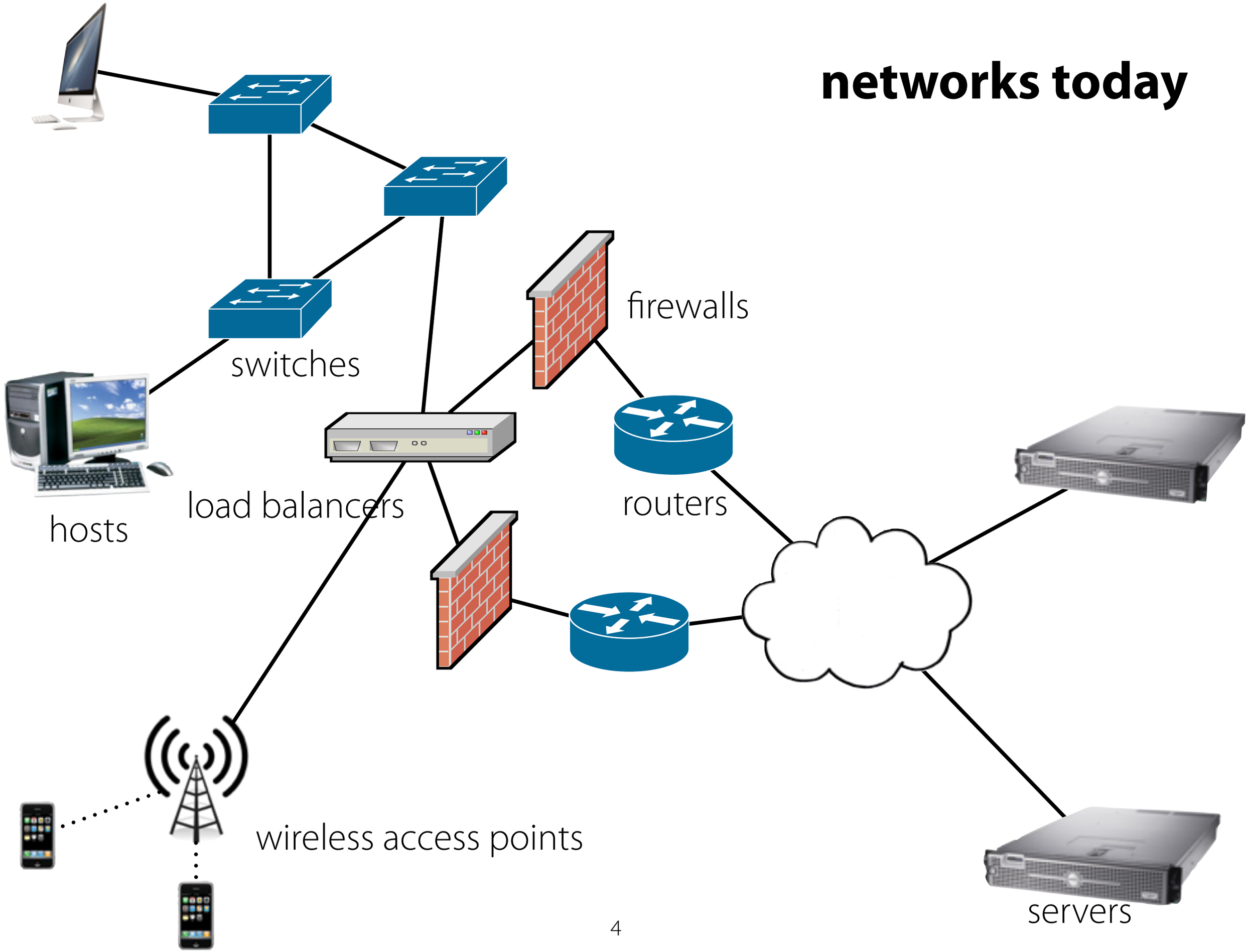
networks today



networks today



networks today



networks today

switches

firewalls

hosts

load balancers

routers

wireless authentication server

wireless access points

servers

Each box:

- Runs specialized, vendor-defined software
- Configured independently via a command-line interface

Difficult to configure, difficult to reason about

switches

firew

hosts

load balancers

routers

wireless authentication server

wireless access points

servers

Recent Network Outages

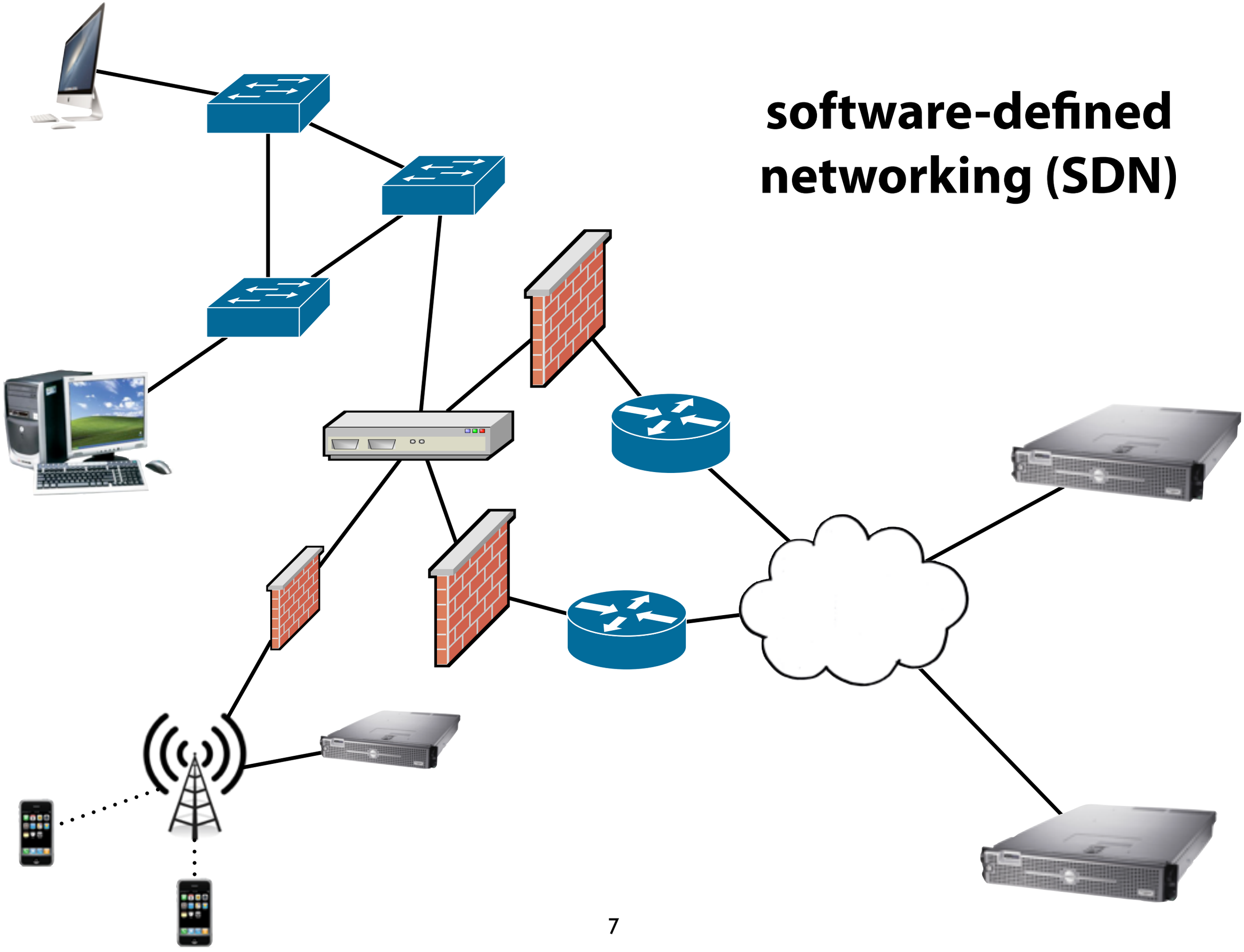
We **discovered a misconfiguration** on this pair of switches that caused what's called a "bridge loop" in the network.

A network **change was [...] executed incorrectly** [...] more "stuck" volumes and added more requests to the re-mirroring storm

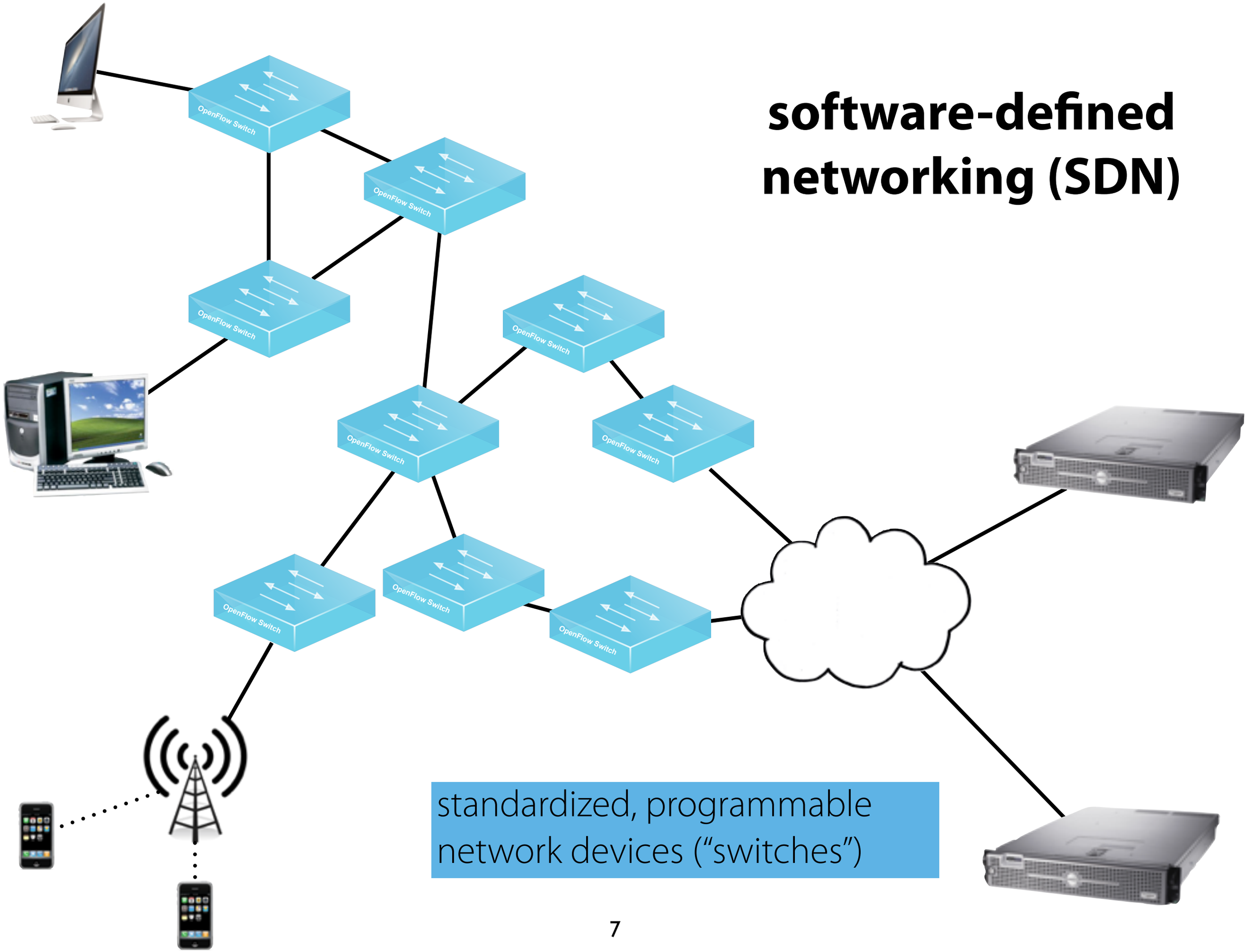
Service outage was due to a series of internal network events that corrupted router data tables

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

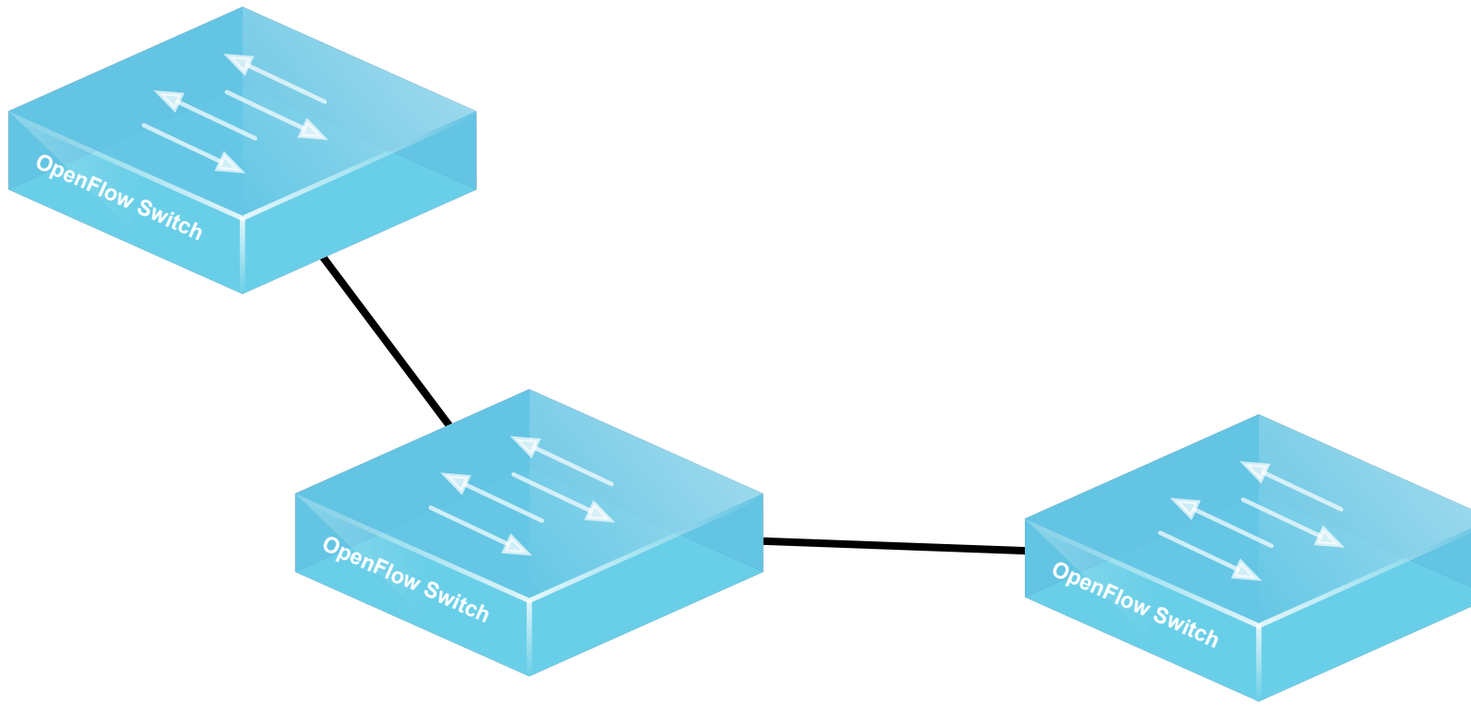
software-defined networking (SDN)

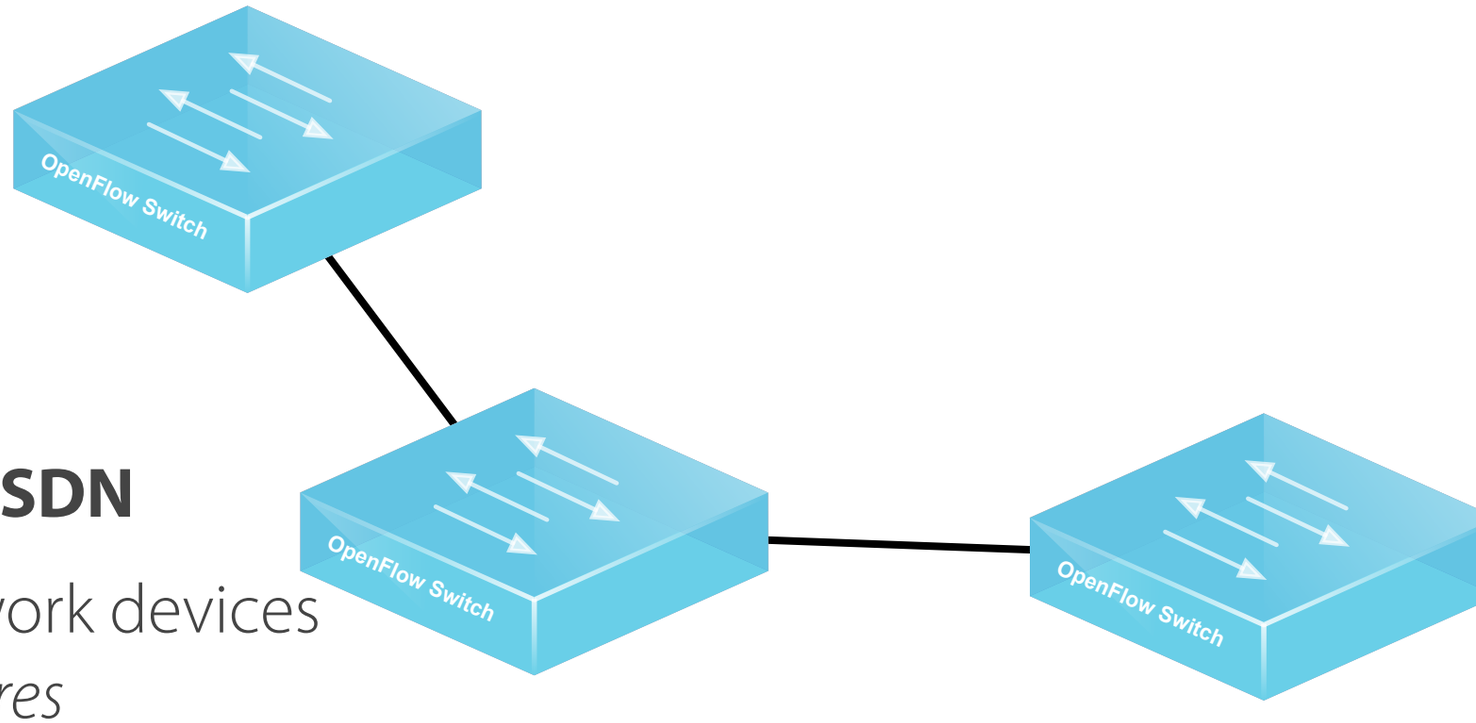


software-defined networking (SDN)



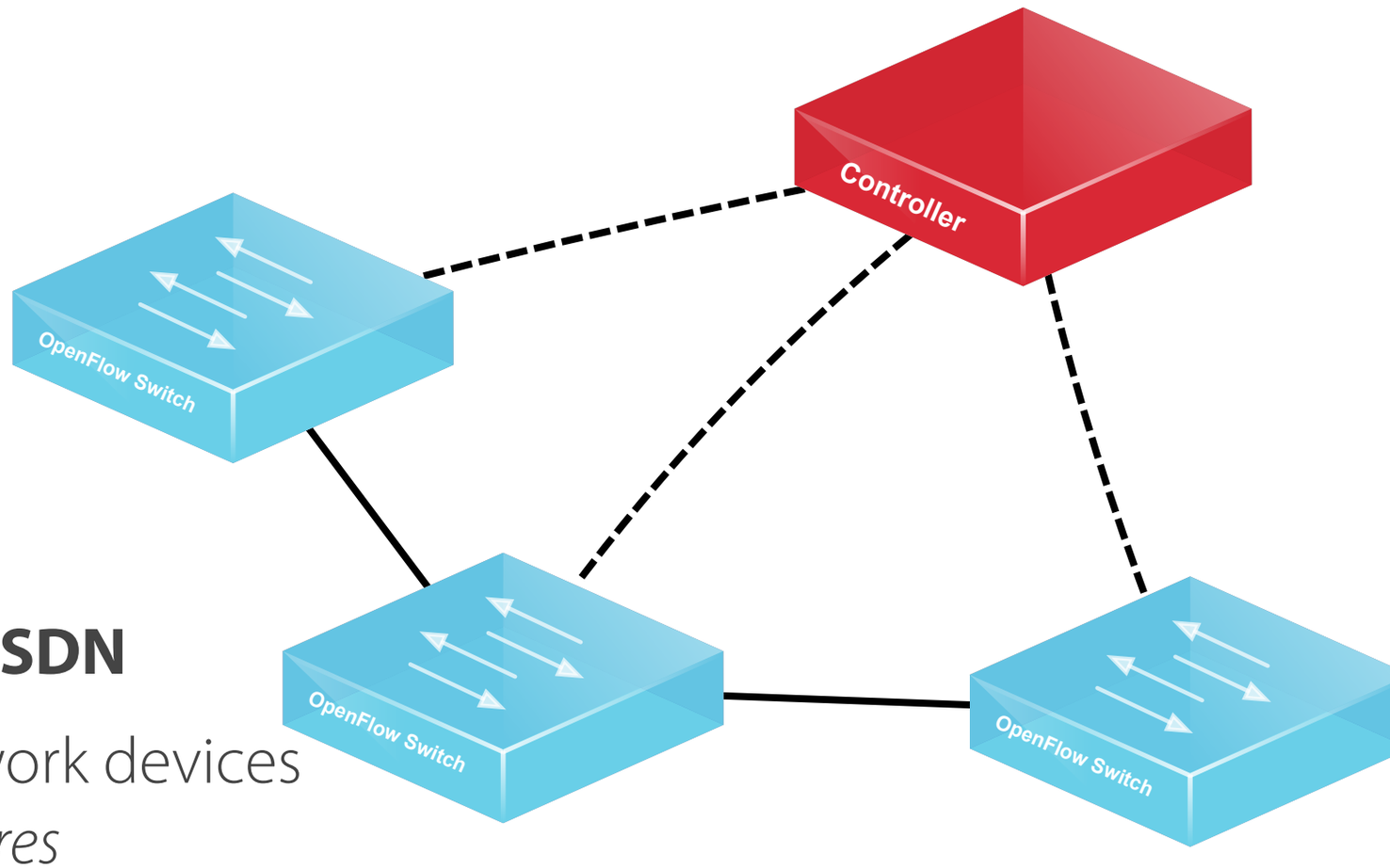
standardized, programmable network devices (“switches”)


OpenFlow Switch
OpenFlow Switch
OpenFlow Switch



Key Features and Advantages of SDN

- Standardized, programmable network devices
  easy to deploy new in-network features



Key Features and Advantages of SDN

- Standardized, programmable network devices
  easy to deploy new in-network features
- Logically centralized controller (beefy server)
  enables reasoning about whole-network behavior

$$f : switch \times port \times packet \rightarrow \{ (port_1, packet_1), ..., (port_n, packet_n) \}$$

Lots of SDN Interest

- By startups and established players
  can buy commercial hardware and software
- 200+ attendees at HotSDN '13
- Six (out of 40) papers at SIGCOMM'13 on SDN

industry-standard SDN protocol

industry-standard SDN protocol

Java

industry-standard SDN protocol

Java 

Python 

C++ 

OpenFlow industry-standard SDN protocol

Java Project Floodlight

Python POX C++ NOX

Haskell

nettle-openflow-0.2.0: OpenFlow protocol messages, binary formats, and servers. | hackageDB | Style

The nettle-openflow package

This package provides data types that model the messages of the OpenFlow protocol, functions that implement serialization and deserialization between these data types and their binary representations in the protocol, and an efficient OpenFlow server. The library is under active development.

OpenFlow industry-standard SDN protocol

Java Project Floodlight

Python POX C++ NOX

Haskell

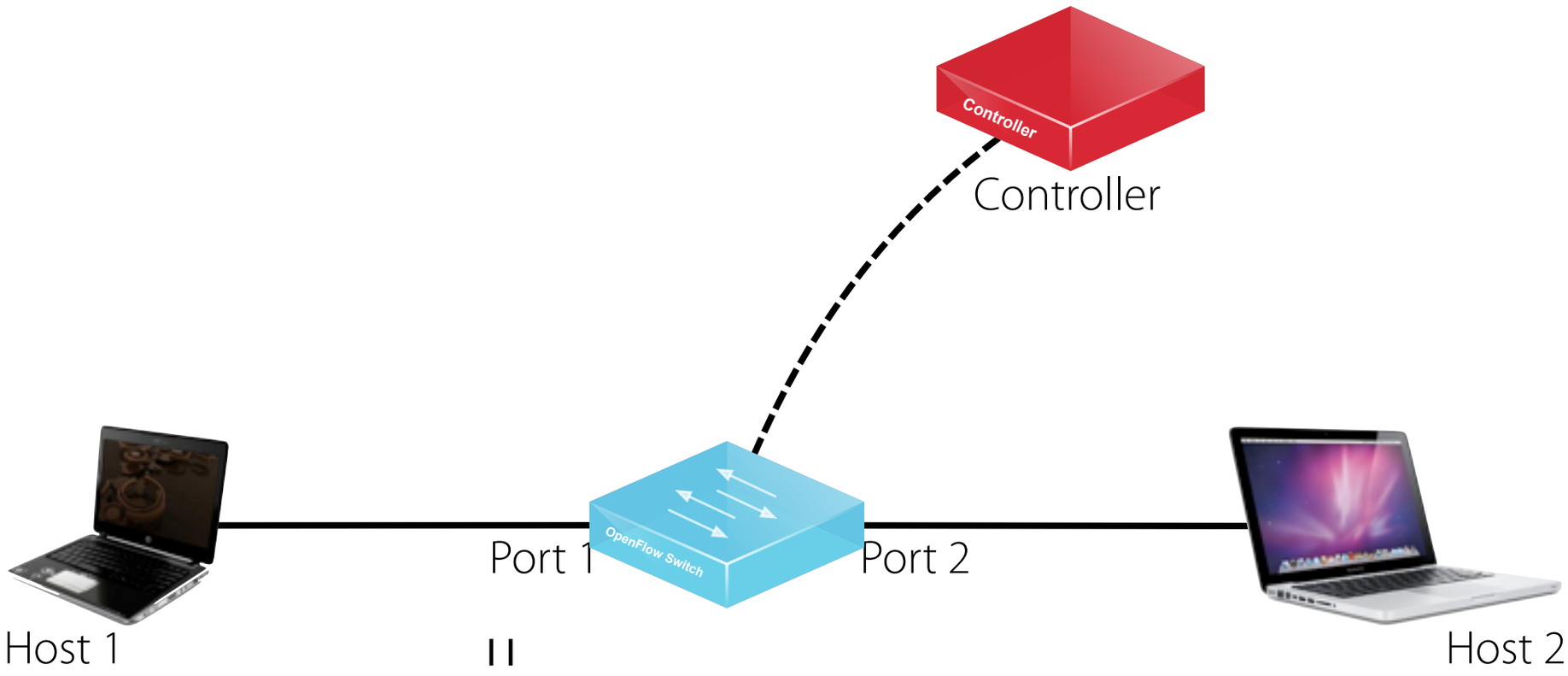
nettle-openflow-0.2.0: OpenFlow protocol messages, binary formats, and servers. | hackageDB | Style

The nettle-openflow package

This package provides data types that model the messages of the OpenFlow protocol, functions that implement serialization and deserialization between these data types and their binary representations in the protocol, and an efficient OpenFlow server. The library is under active development.

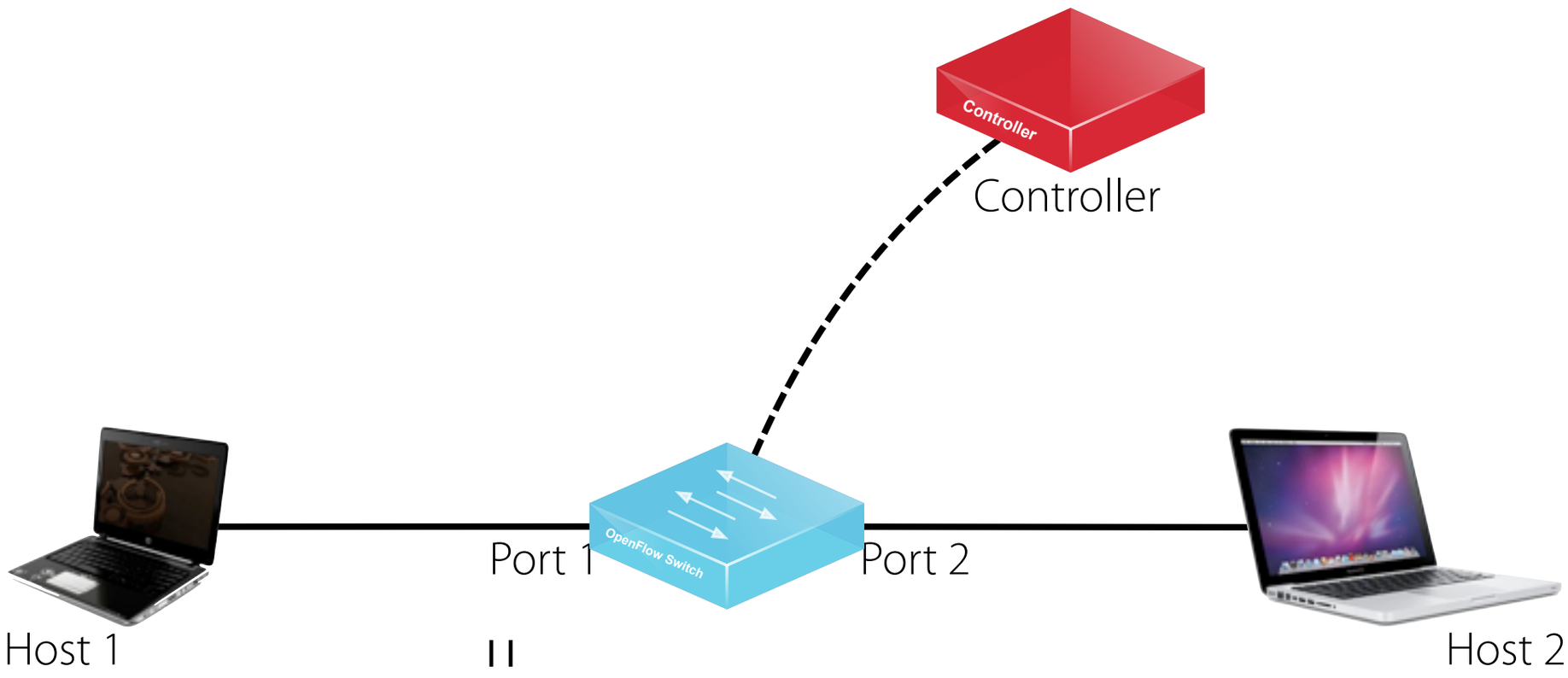
OCaml frenetic ≫

Example


Controller
Controller
OpenFlow Switch
Port 1
Port 2
Host 1
Host 2

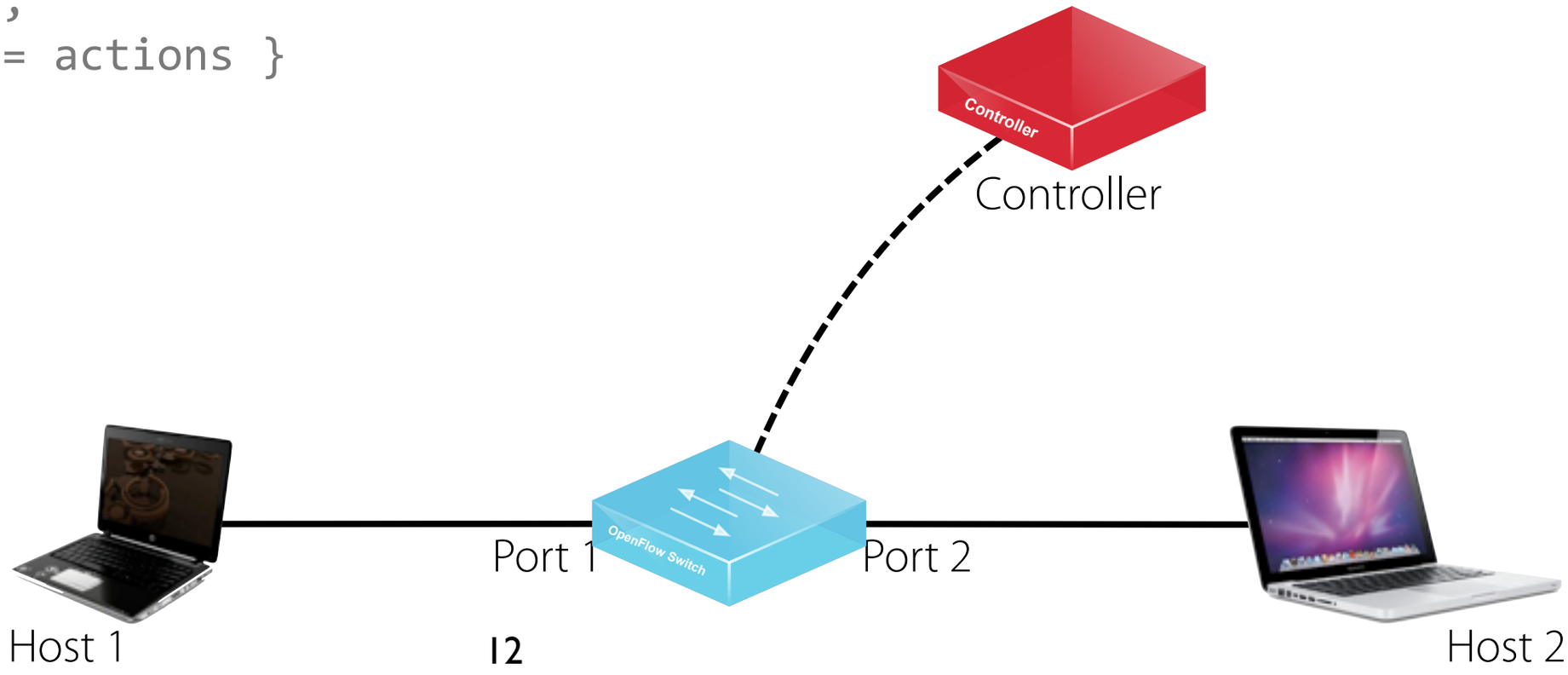
Example

```
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let actions =
    if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
    { output_payload = pktIn.input_payload;
      port_id = None;
      apply_actions = actions }
```



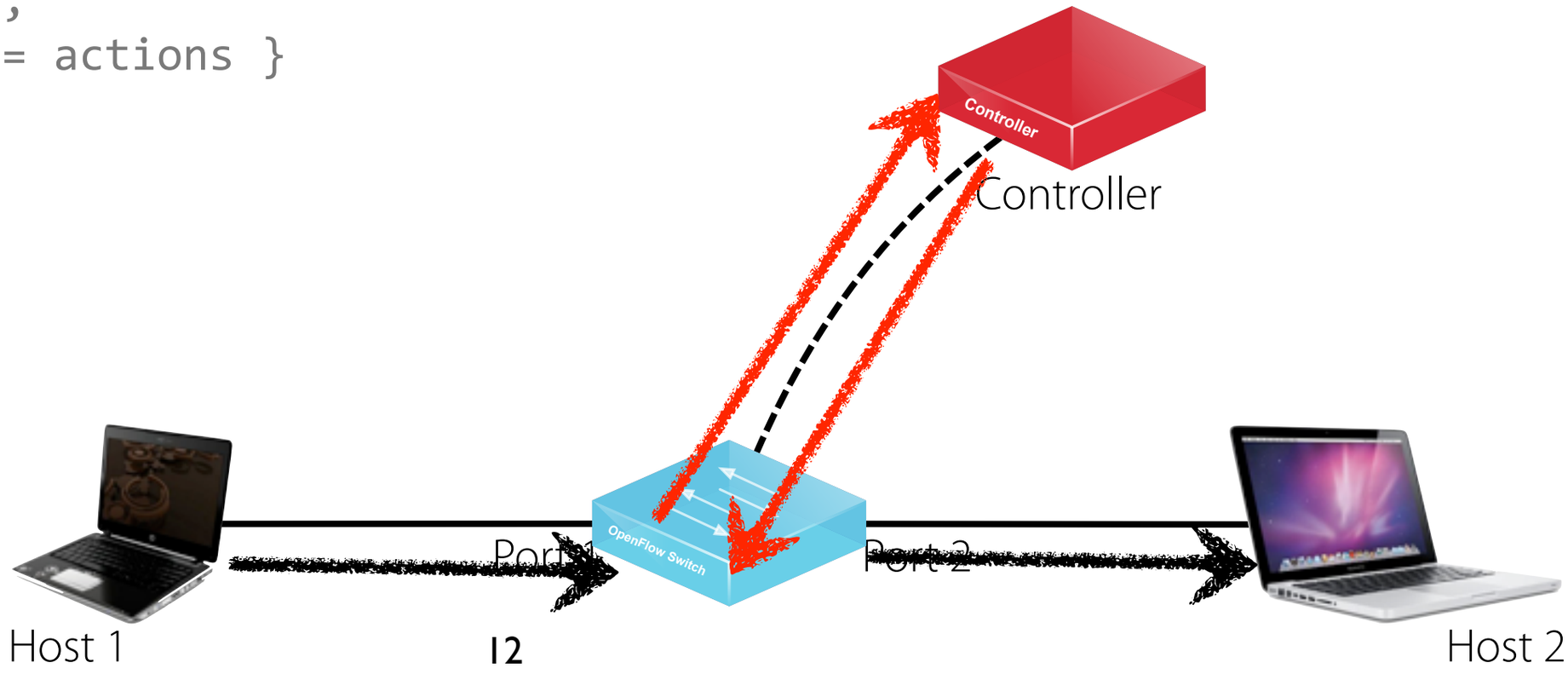
Example

```
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let actions =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
       (Packet.tpDst = 22 || Packet.tpSrc = 22) then
      [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
    { output_payload = pktIn.input_payload;
      port_id = None;
      apply_actions = actions }
```



Example

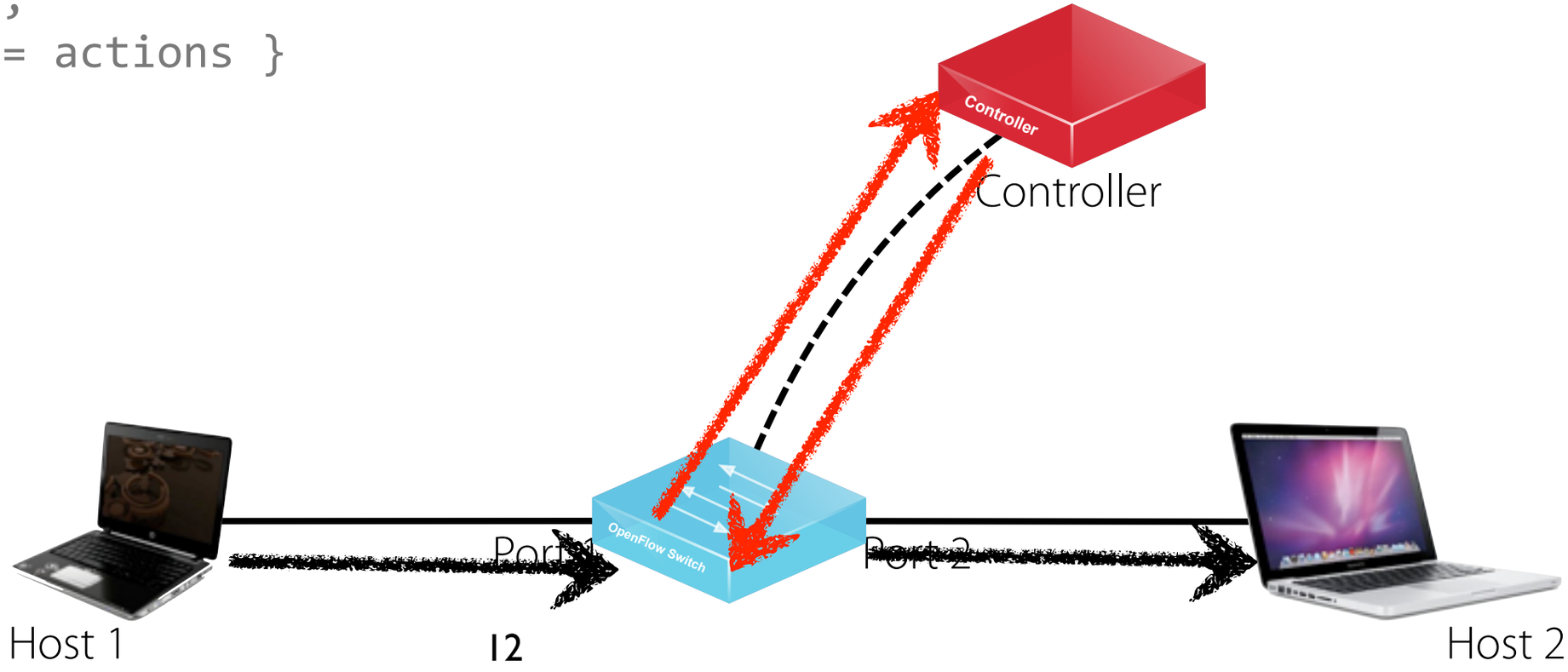
```
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let actions =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
       (Packet.tpDst = 22 || Packet.tpSrc = 22) then
      [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
    { output_payload = pktIn.input_payload;
      port_id = None;
      apply_actions = actions }
```



Example

```
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let actions =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto pk = 6 &&
       (Packet.tpDst = 22 || Packet.tpSrc = 22) then
      [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
    { output_payload = pktIn.input_payload;
      port_id = None;
      apply_actions = actions }
```

Generic packet parser

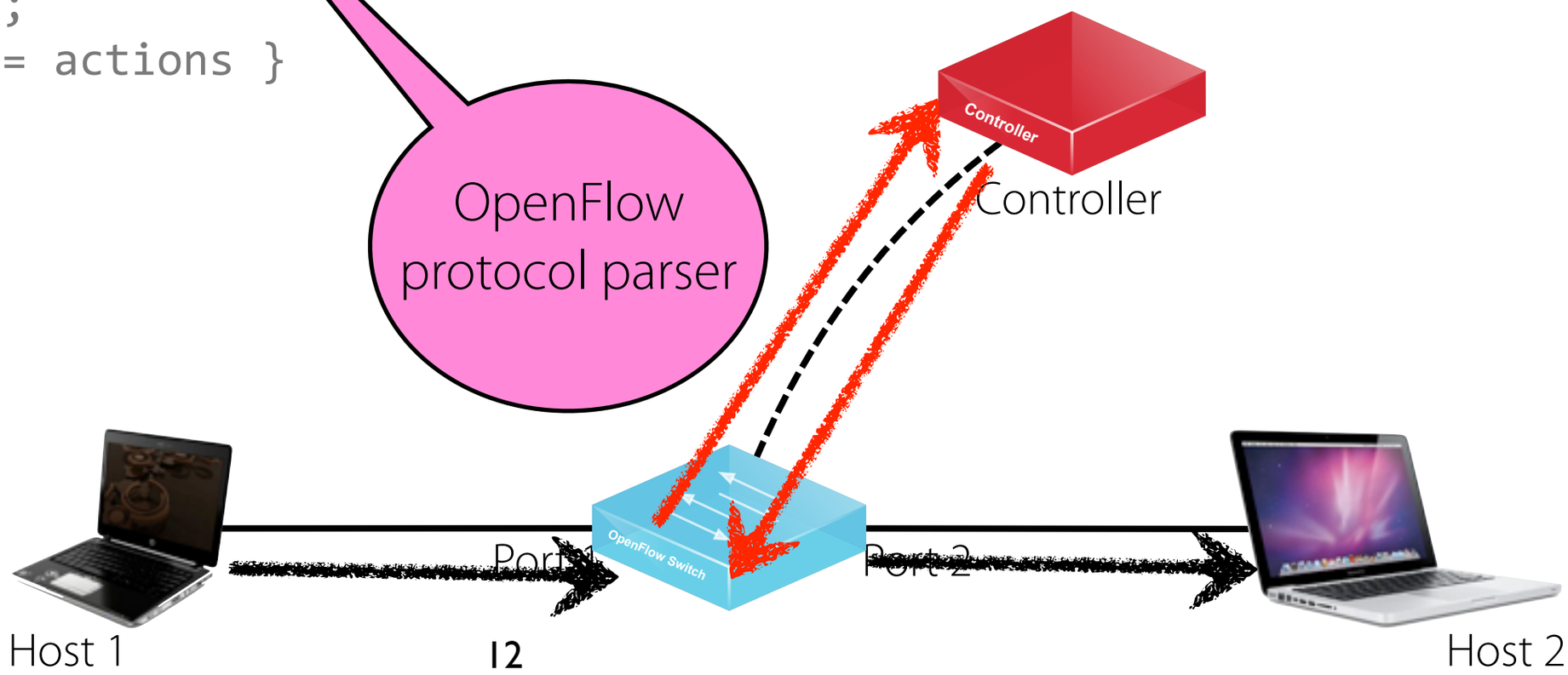


Example

```
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let actions =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto pk = 6 &&
       (Packet.tpDst = 22 || Packet.tpSrc = 22) then
      [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
    { output_payload = pktIn.input_payload;
      port_id = None;
      apply_actions = actions }
```

Generic packet parser

OpenFlow protocol parser



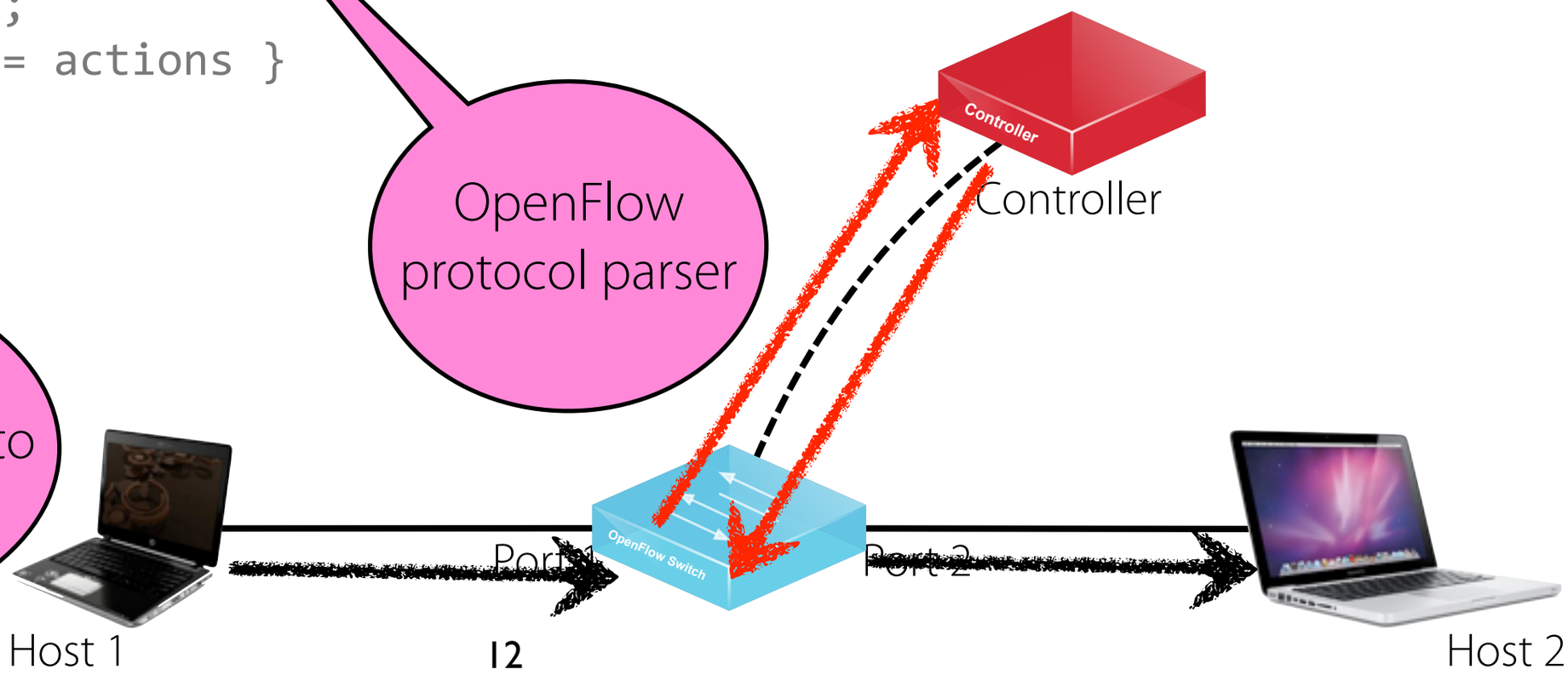
Example

```
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let actions =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto pk = 6 &&
       (Packet.tpDst = 22 || Packet.tpSrc = 22) then
      [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
    { output_payload = pktIn.input_payload;
      port_id = None;
      apply_actions = actions }
```

Generic packet parser

OpenFlow protocol parser

Runtime system to manage connections to switches



ocaml-packet

ocaml-packet

- Serialization / deserialization for several packet formats

  TCP, IP, ARP, ICMP, Ethernet, 802.1Q

ocaml-packet

- Serialization / deserialization for several packet formats

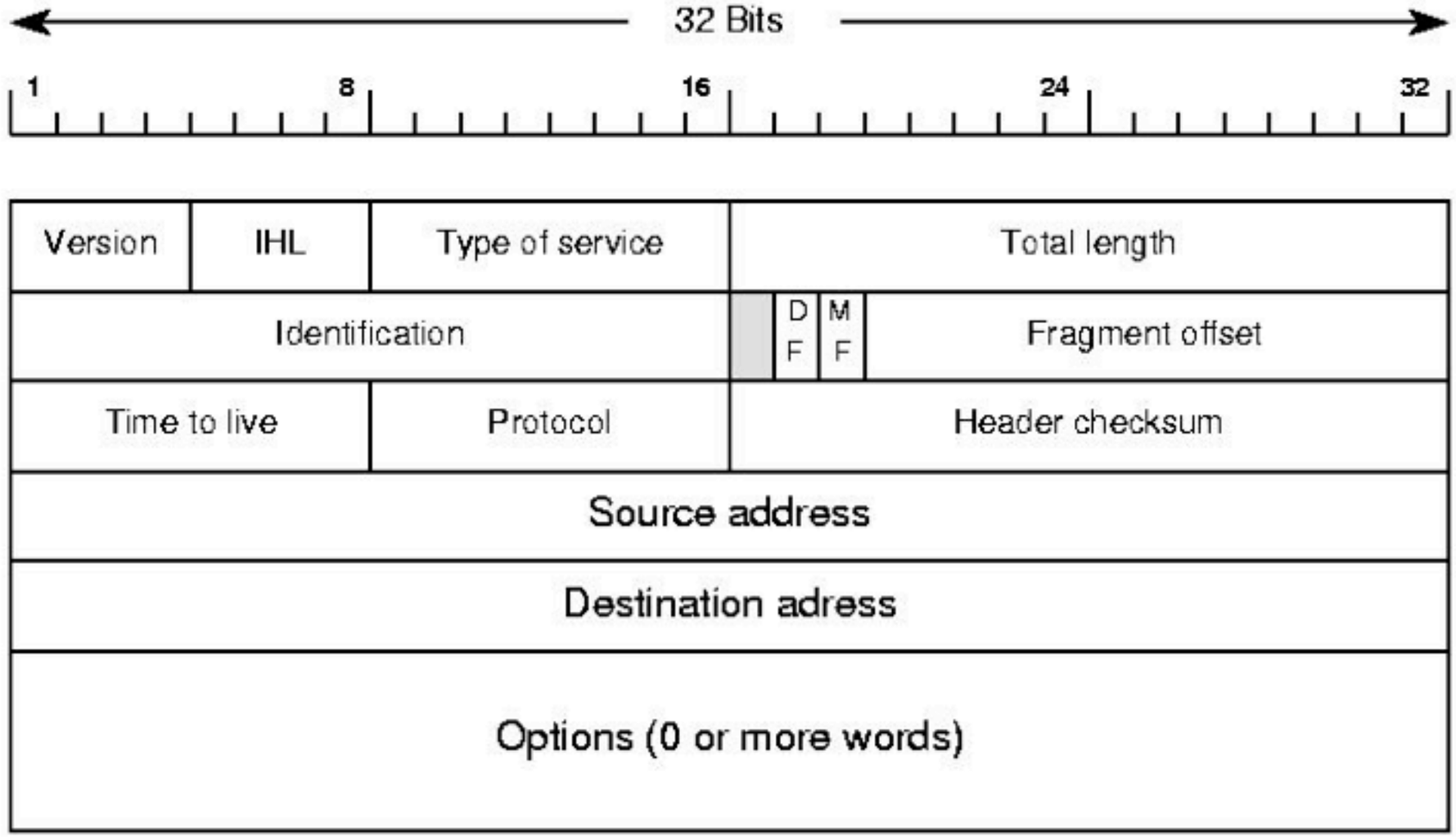
  TCP, IP, ARP, ICMP, Ethernet, 802.1Q
- Only depends on cstruct

  by Anil Madhavapeddy et al.

ocaml-packet

- Serialization / deserialization for several packet formats
  TCP, IP, ARP, ICMP, Ethernet, 802.1Q
- Only depends on cstruct
  by Anil Madhavapeddy et al.

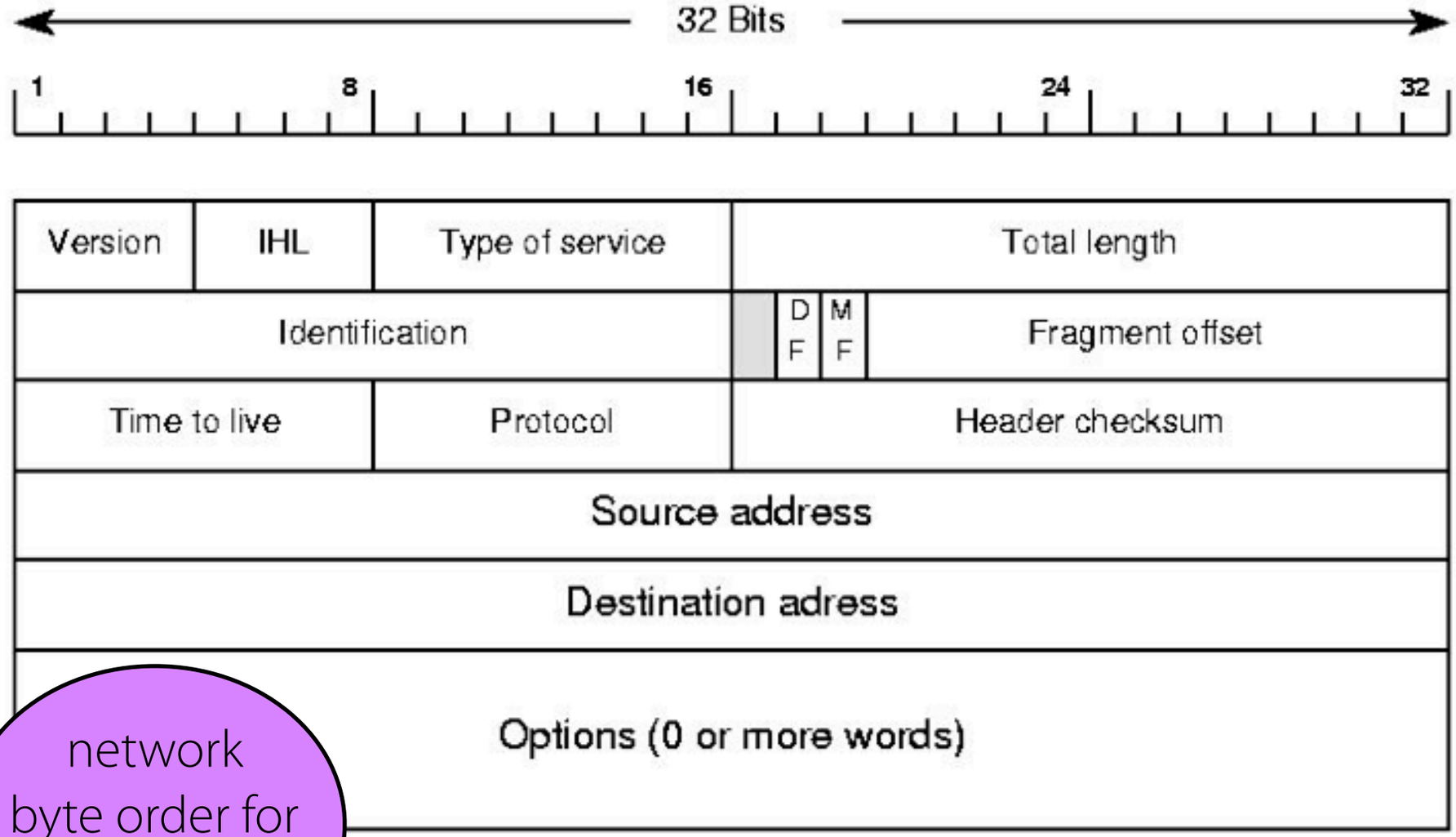
```
cstruct ip {
  uint8_t vhl;
  uint8_t tos;
  uint16_t len;
  uint16_t ident;
  uint16_t frag;
  uint8_t ttl;
  uint8_t proto;
  uint16_t chksum;
  uint32_t src;
  uint32_t dst;
  uint32_t options
} as big_endian
```



ocaml-packet

- Serialization / deserialization for several packet formats
  TCP, IP, ARP, ICMP, Ethernet, 802.1Q
- Only depends on cstruct
  by Anil Madhavapeddy et al.

```
cstruct ip {
  uint8_t vhl;
  uint8_t tos;
  uint16_t len;
  uint16_t ident;
  uint16_t frag;
  uint8_t ttl;
  uint8_t proto;
  uint16_t chksum;
  uint32_t src;
  uint32_t dst;
  uint32_t options
} as big_endian
```



network byte order for free

ocaml-openflow

ocaml-openflow

- Serialization for OpenFlow 1.0 and 1.3
  - OpenFlow 1.0 support based on mirage-openflow, using ideas from Nettle (Haskell)
  - OpenFlow 1.3 is less complete, but still usable
  - Cstruct-based serialization

ocaml-openflow

- Serialization for OpenFlow 1.0 and 1.3
  - OpenFlow 1.0 support based on mirage-openflow, using ideas from Nettle (Haskell)
  - OpenFlow 1.3 is less complete, but still usable
  - Cstruct-based serialization
- Runtime systems for OpenFlow 1.0 and 1.3
  - Listens for TCP connections from switches
  - Does the OpenFlow handshake, keeps connections to switches alive

ocaml-openflow

- Serialization for OpenFlow 1.0 and 1.3
  - OpenFlow 1.0 support based on mirage-openflow, using ideas from Nettle (Haskell)
  - OpenFlow 1.3 is less complete, but still usable
  - Cstruct-based serialization
- Runtime systems for OpenFlow 1.0 and 1.3
  - Listens for TCP connections from switches
  - Does the OpenFlow handshake, keeps connections to switches alive

```
type t (* A handle to an OpenFlow switch. *)

val connect : Lwt_unix.file_descr -> t option Lwt.t
val send : t -> xid -> message -> unit Lwt.t
val recv : t -> (xid * message) Lwt.t
val disconnect : t -> unit Lwt.t
val wait_disconnect : t -> unit Lwt.t
```

ocaml-openflow

- Serialization for OpenFlow 1.0 and 1.3
  - OpenFlow 1.0 support based on mirage-openflow, using ideas from Nettle (Haskell)
  - OpenFlow 1.3 is less complete, but still usable
  - Cstruct-based serialization
- Runtime systems for OpenFlow 1.0 and 1.3
  - Listens for TCP connections from switches
  - Does the OpenFlow handshake, keeps connections to switches alive

```
type t (* A handle to an OpenFlow switch. *)

val connect : Lwt_unix.file_descr -> t option Lwt.t
val send : t -> xid -> message -> unit Lwt.t
val recv : t -> (xid * message) Lwt.t
val disconnect : t -> unit Lwt.t
val wait_disconnect : t -> unit Lwt.t
```

OX

OX

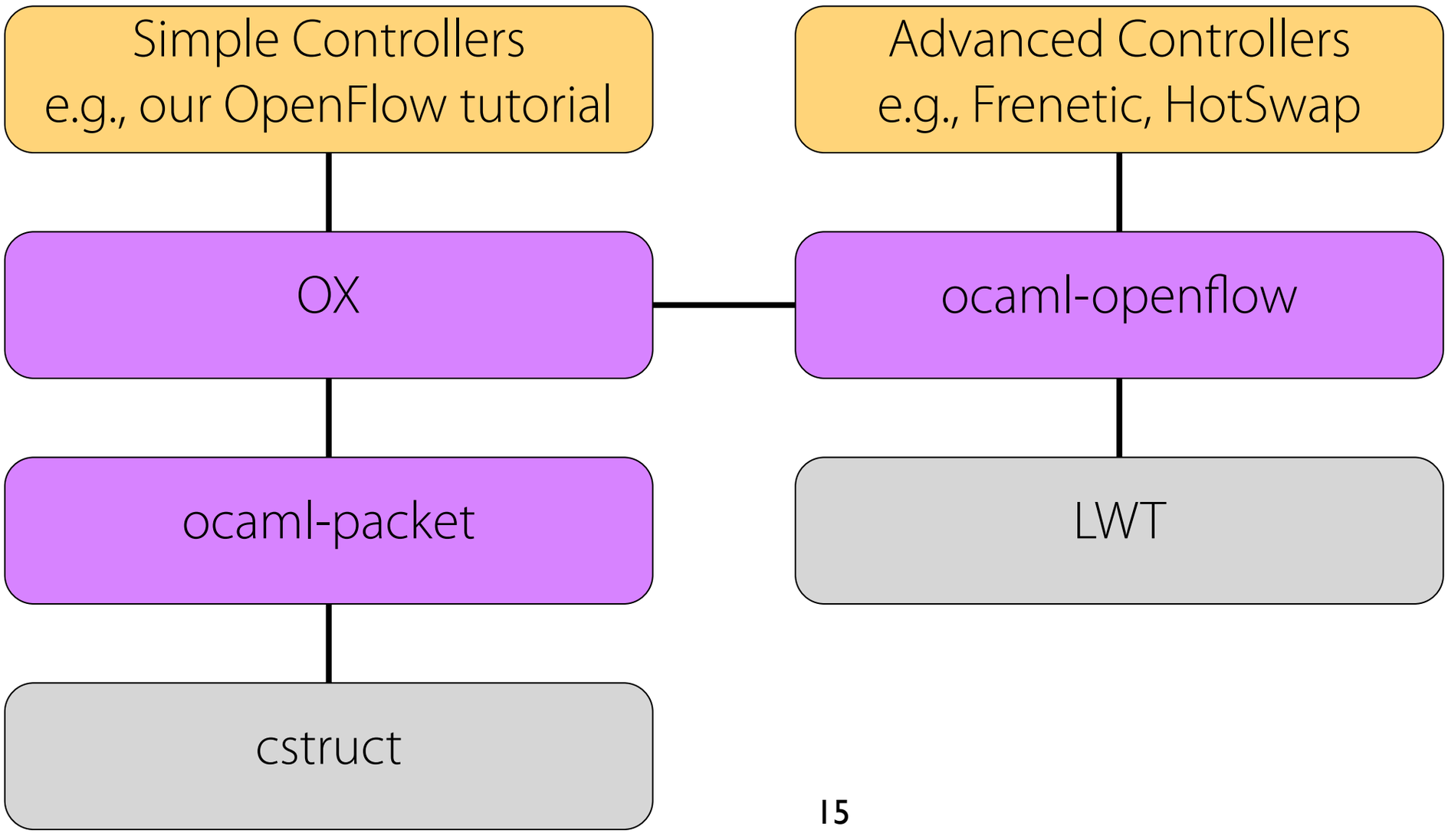
- Based on popular OpenFlow platforms
  - Inspired by POX (Python) and NOX (Python/C++)

OX

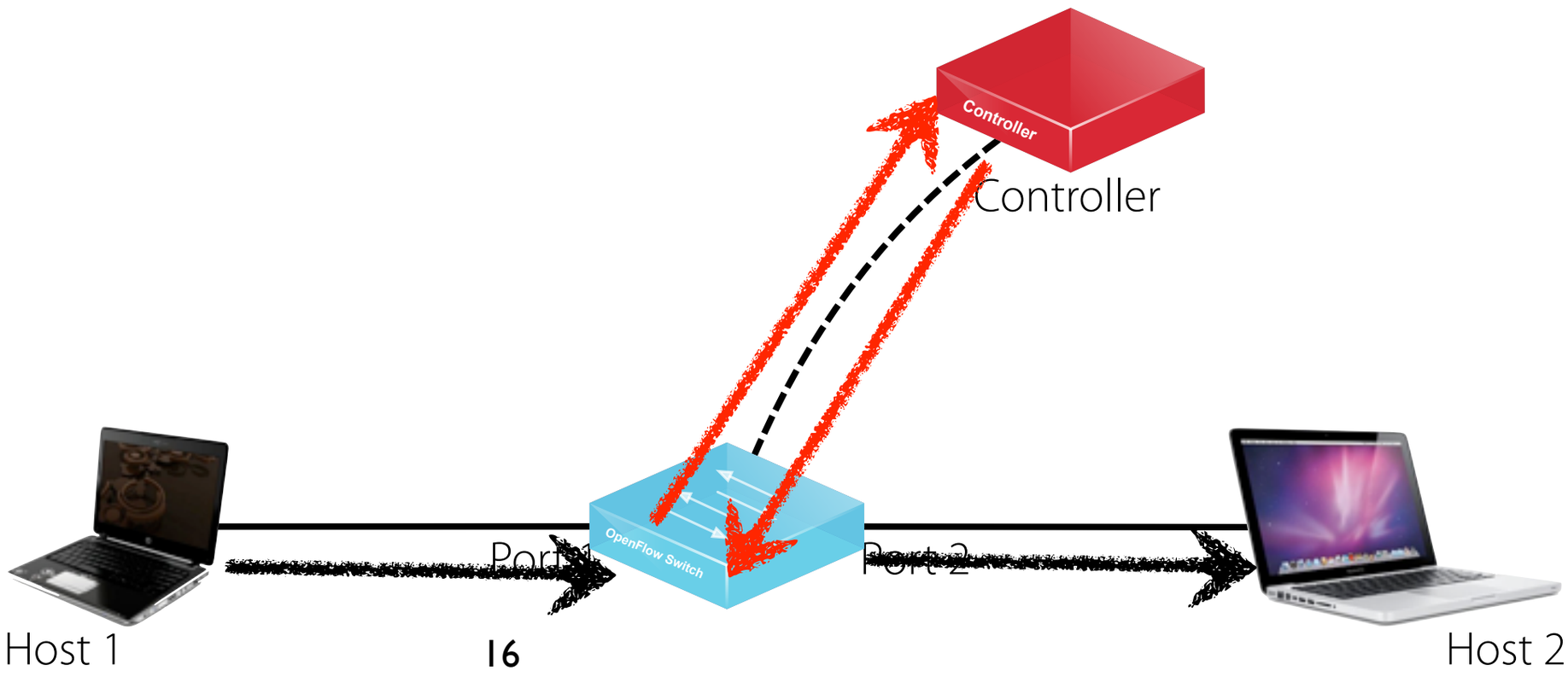
- Based on popular OpenFlow platforms
  - Inspired by POX (Python) and NOX (Python/C++)
- Simple, single-threaded OpenFlow interface
  - Uses LWT and ocaml-openflow internally

OX

- Based on popular OpenFlow platforms
  - Inspired by POX (Python) and NOX (Python/C++)
- Simple, single-threaded OpenFlow interface
  - Uses LWT and ocaml-openflow internally

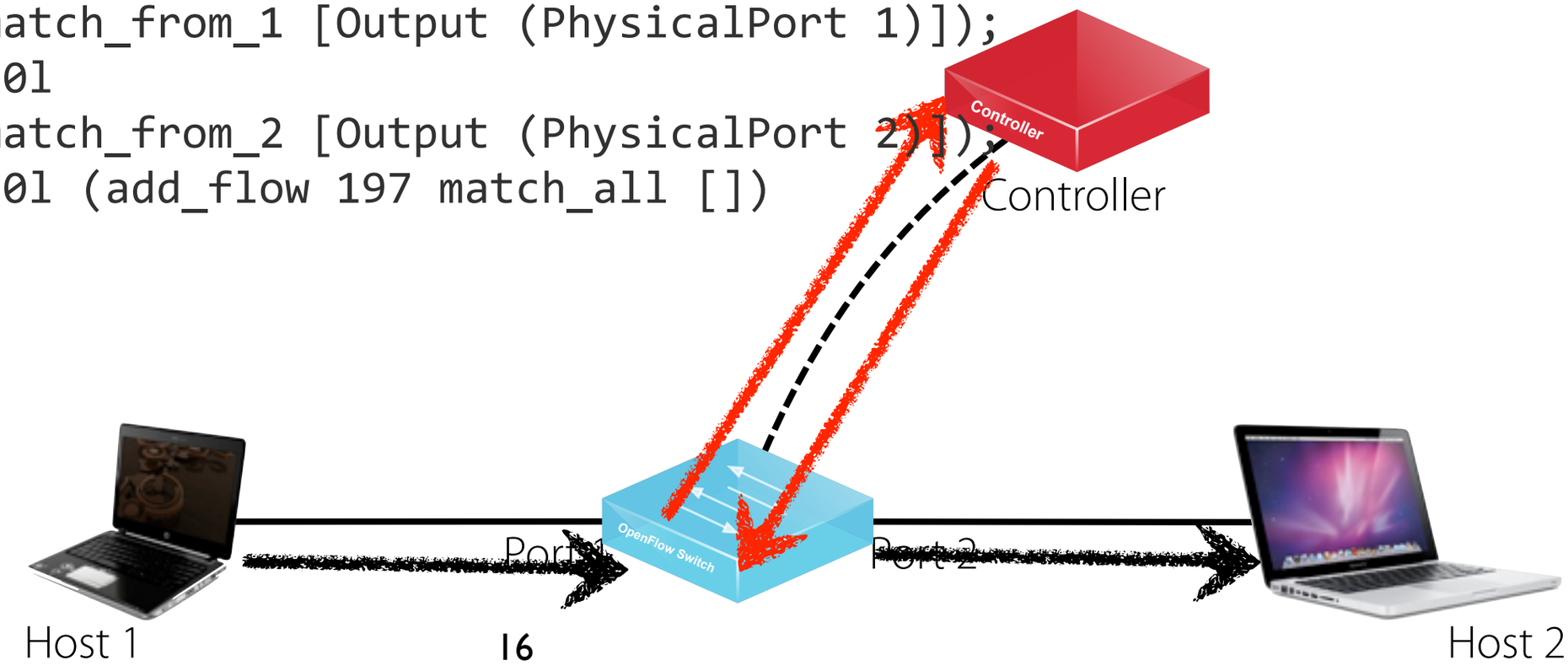


```
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let action =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
       (Packet.tpDst = 22 || Packet.tpSrc = 22) then
      [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
    { output_payload = pktIn.input_payload; port_id = None;
      apply_actions = action }
```



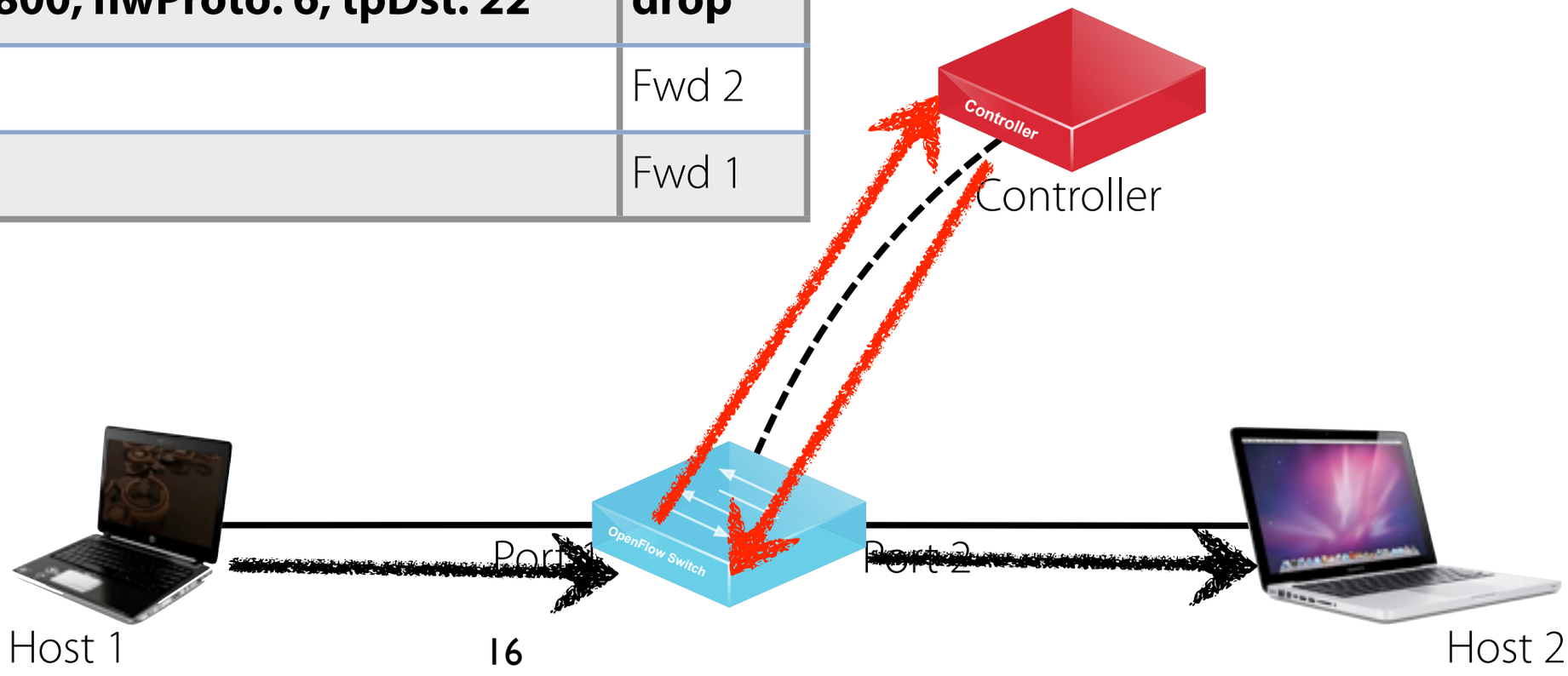
```
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let action =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
       (Packet.tpDst = 22 || Packet.tpSrc = 22) then
      [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
    { output_payload = pktIn.input_payload; port_id = None;
      apply_actions = action }
```

```
let switch_connected (sw : switchId) : unit =
  send_flow_mod sw 0l (add_flow 200 match_ssh_src []);
  send_flow_mod sw 0l (add_flow 200 match_ssh_dst []);
  send_flow_mod sw 0l
    (add_flow 199 match_from_1 [Output (PhysicalPort 1)]);
  send_flow_mod sw 0l
    (add_flow 198 match_from_2 [Output (PhysicalPort 2)]);
  send_flow_mod sw 0l (add_flow 197 match_all [])
```



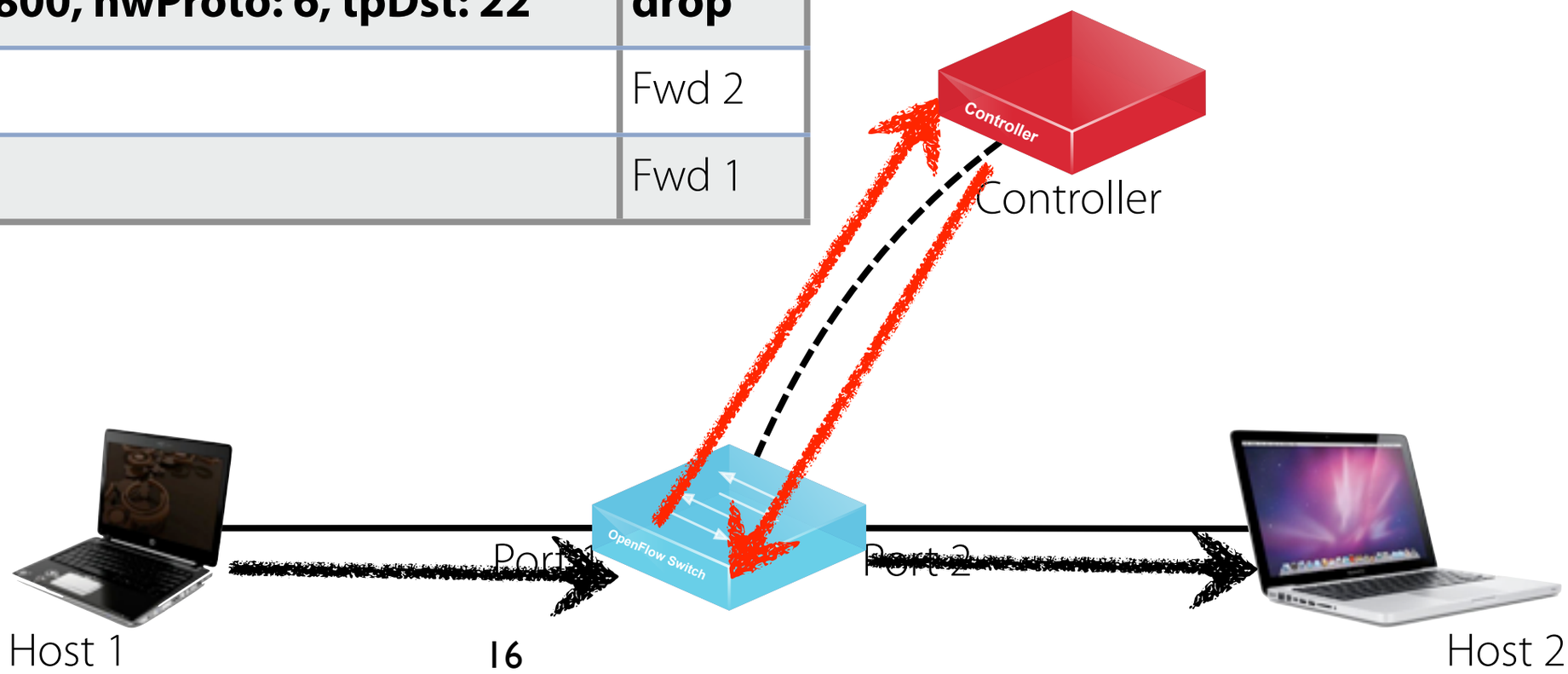
```
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let action =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
       (Packet.tpDst = 22 || Packet.tpSrc = 22) then
      [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
    { output_payload = pktIn.input_payload; port_id = None;
      apply_actions = action }
```

Priority	Pattern	Actions
200	**dlType:0x800, nwProto: 6, tpSrc: 22**	**drop**
200	**dlType:0x800, nwProto: 6, tpDst: 22**	**drop**
199	inPort: 1	Fwd 2
198	inPort: 2	Fwd 1



```
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let action =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
       (Packet.tpDst = 22 || Packet.tpSrc = 22) then
      [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
    { output_payload = pktIn.input_payload; port_id = None;
      apply_actions = action }
```

Priority	Pattern	Actions
200	**dlType:0x800, nwProto: 6, tpSrc: 22**	**drop**
200	**dlType:0x800, nwProto: 6, tpDst: 22**	**drop**
199	inPort: 1	Fwd 2
198	inPort: 2	Fwd 1



Monsanto et al., POPL 2012

Priority	Pattern	Actions
65535	dstIP: H1	Forward

+

Priority	Pattern	Actions
65535	SSH	Monitor

Priority	Pattern	Actions
65535	dstIP: H1	Forward

+

Priority	Pattern	Actions
65535	SSH	Monitor

↓

Priority	Pattern	Actions
65535	SSH, dstIP: H1	Forward, Monitor
65534	dstIP: H1	Forward
65533	SSH	Monitor

Priority	Pattern	Actions
65535	dstIP: H1	Forward

+

Priority	Pattern	Actions
65535	SSH	Monitor

↓

Priority	Pattern	Actions
65535	SSH, dstIP: H1	Forward, Monitor
65534	dstIP: H1	Forward
65533	SSH	Monitor

- Other issues:
  - Well-formedness criteria on patterns; rules applied non-deterministically in certain situations *(Guha et al., PLDI 2013)*
  - Cannot atomically update all switches *(Rietblatt et al., SIGCOMM 2012)*

frenetic >>

frenetic >>

- DSL for programming OpenFlow networks
  - Boolean predicates to match packets
  - Several policy composition operators
  - All compile to OpenFlow tables
  - Abstractions address several fundamental problems of SDN

frenetic >>

- DSL for programming OpenFlow networks
  - Boolean predicates to match packets
  - Several policy composition operators
  - All compile to OpenFlow tables
  - Abstractions address several fundamental problems of SDN
- Embedded in OCaml
  - Access to standard OCaml data structures and functions
  - Must use LWT for concurrency

frenetic ≫

- DSL for programming OpenFlow networks
  - Boolean predicates to match packets
  - Several policy composition operators
  - All compile to OpenFlow tables
  - Abstractions address several fundamental problems of SDN
- Embedded in OCaml
  - Access to standard OCaml data structures and functions
  - Must use LWT for concurrency
- Surface syntax

```
if tpSrc = 22 || tpDst = 22 then
  drop
else if inPort = 1 then
  fwd(2)
else
  fwd(1)
```

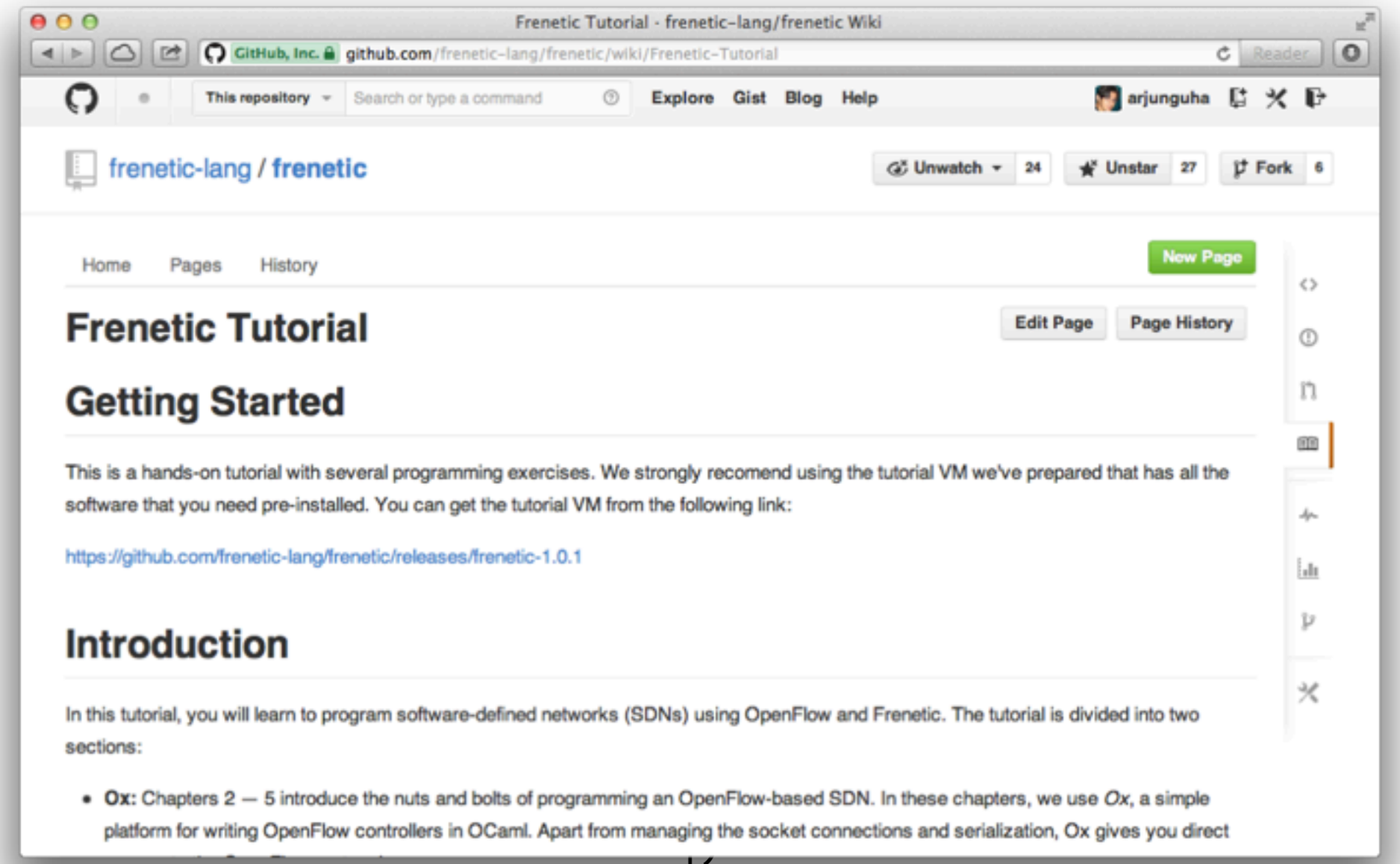
frenetic >>

frenetic >>

- Implements key ideas from several published papers
  - ICFP'11, POPL'12, SIGCOMM'11, PLDI'13, HotSDN'12, HotSDN'13
- Open source development of current work

frenetic >>

- Implements key ideas from several published papers
  - ICFP'11, POPL'12, SIGCOMM'11, PLDI'13, HotSDN'12, HotSDN'13
- Open source development of current work
- Frenetic and OpenFlow tutorial in OCaml



From Haskell to OCaml :)

From Haskell to OCaml :)

- Core compiler and runtime system done in Coq
  - Extracting to Haskell is much more painful
  - Haskell extraction made us abandon some Coq functors

From Haskell to OCaml :)

- Core compiler and runtime system done in Coq
  - Extracting to Haskell is much more painful
  - Haskell extraction made us abandon some Coq functors
- Functors helped us discover key algebraic properties
  - Helped establish connection to Kleene Algebra with Tests (KAT)
  - Basis of a paper in submission

From Haskell to OCaml :)

- Core compiler and runtime system done in Coq
  - Extracting to Haskell is much more painful
  - Haskell extraction made us abandon some Coq functors
- Functors helped us discover key algebraic properties
  - Helped establish connection to Kleene Algebra with Tests (KAT)
  - Basis of a paper in submission
- Easier for our networking collaborators to grok OCaml
  - Design discussions over .mli files with networking collaborators

From Haskell to OCaml :)

- Core compiler and runtime system done in Coq
  - Extracting to Haskell is much more painful
  - Haskell extraction made us abandon some Coq functors
- Functors helped us discover key algebraic properties
  - Helped establish connection to Kleene Algebra with Tests (KAT)
  - Basis of a paper in submission
- Easier for our networking collaborators to grok OCaml
  - Design discussions over .mli files with networking collaborators
- OPAM and oasis are *great*
  - All our packages are on OPAM
  - OPAM overlays provide stability

From Haskell to OCaml :(

From Haskell to OCaml :(

- Lwt programming is mind-boggling for beginners
  - Syntax extension helps a lot
  - E.g., very difficult to reason about exceptions

From Haskell to OCaml :(

- Lwt programming is mind-boggling for beginners
  - Syntax extension helps a lot
  - E.g., very difficult to reason about exceptions
- No more multicore
  - Needed for large networks (Voellmy *et al.* at **Haskell '13**)
  - Needed to compile fast-changing policies (Ferguson *et al.* at SIGCOMM'13)

Hack your Network in OCaml

Hack your Network in OCaml

Hack at any layer

- Packet serialization
- OpenFlow serialization
- Ox controller
- Frenetic

Hack your Network in OCaml

Ongoing work in OCaml

Hack at any layer

- Packet serialization
- OpenFlow serialization
- Ox controller
- Frenetic

- Network Hypervisor
  Laurent Vanbever, *et al.*
- Datalog-based SDN language
  Tim Nelson, *et al.*
- Fault Tolerant Frenetic
  Mark Reitblatt, *et al.*
- Property-checking Frenetic
  Rebecca Coombes, Matthew Milano, *et al.*