Machine-Verified Controllers

Arjun Guha, Mark Reitblatt, and Nate Foster

Cornell University

Networks in the News

Networks in the News

a network change was performed […] executed incorrectly […] more "stuck" volumes and added more requests to the re-mirroring storm

Networks in the News

a network change was performed […] executed incorrectly […] more "stuck" volumes and added more requests to the re-mirroring storm

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

Networks in the News

a network change was performed […] executed incorrectly […] more "stuck" volumes and added more requests to the re-mirroring storm

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

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

SDN Facilitates Formal Methods

SDN Facilitates Formal Methods

Run-time Monitoring: VeriFlow

by Khurshid, Zhou, Caesar, and Godfrey

SDN Facilitates Formal Methods

Run-time Monitoring: VeriFlow

by Khurshid, Zhou, Caesar, and Godfrey

- runtime safety checking
- detects errors dynamically

SDN Facilitates Formal Methods

Run-time Monitoring: VeriFlow

by Khurshid, Zhou, Caesar, and Godfrey

- runtime safety checking
- detects errors dynamically
- small runtime overhead

SDN Facilitates Formal Methods

Run-time Monitoring: VeriFlow

by Khurshid, Zhou, Caesar, and Godfrey

- runtime safety checking
- detects errors dynamically
- small runtime overhead
- fixes must be out-of-band

SDN Facilitates Formal Methods

Run-time Monitoring: VeriFlow

by Khurshid, Zhou, Caesar, and Godfrey

- runtime safety checking
- detects errors dynamically
- small runtime overhead
- fixes must be out-of-band

Static Bug-Finding: NICE

by Canini, Venzano, Perešíni, Kostić, and Rexford

SDN Facilitates Formal Methods

Run-time Monitoring: VeriFlow

by Khurshid, Zhou, Caesar, and Godfrey

- runtime safety checking
- detects errors dynamically
- small runtime overhead
- fixes must be out-of-band

Static Bug-Finding: NICE

by Canini, Venzano, Perešíni, Kostić, and Rexford

- symbolic execution / model checking
- finds several bugs statically

SDN Facilitates Formal Methods

Run-time Monitoring: VeriFlow

by Khurshid, Zhou, Caesar, and Godfrey

- runtime safety checking
- detects errors dynamically
- small runtime overhead
- fixes must be out-of-band

Static Bug-Finding: NICE

by Canini, Venzano, Perešíni, Kostić, and Rexford

- symbolic execution / model checking
- finds several bugs statically
- seconds to days to test, no runtime cost

SDN Facilitates Formal Methods

Run-time Monitoring: VeriFlow

by Khurshid, Zhou, Caesar, and Godfrey

- runtime safety checking
- detects errors dynamically
- small runtime overhead
- fixes must be out-of-band

Static Bug-Finding: NICE

by Canini, Venzano, Perešíni, Kostić, and Rexford

- symbolic execution / model checking
- finds several bugs statically
- seconds to days to test, no runtime cost
- cannot prove the absence of bugs

SDN Facilitates Formal Methods

Run-time Monitoring: VeriFlow

by Khurshid, Zhou, Caesar, and Godfrey

- runtime safety checking
- detects errors dynamically
- small runtime overhead
- fixes must be out-of-band

Static Bug-Finding: NICE

by Canini, Venzano, Perešíni, Kostić, and Rexford

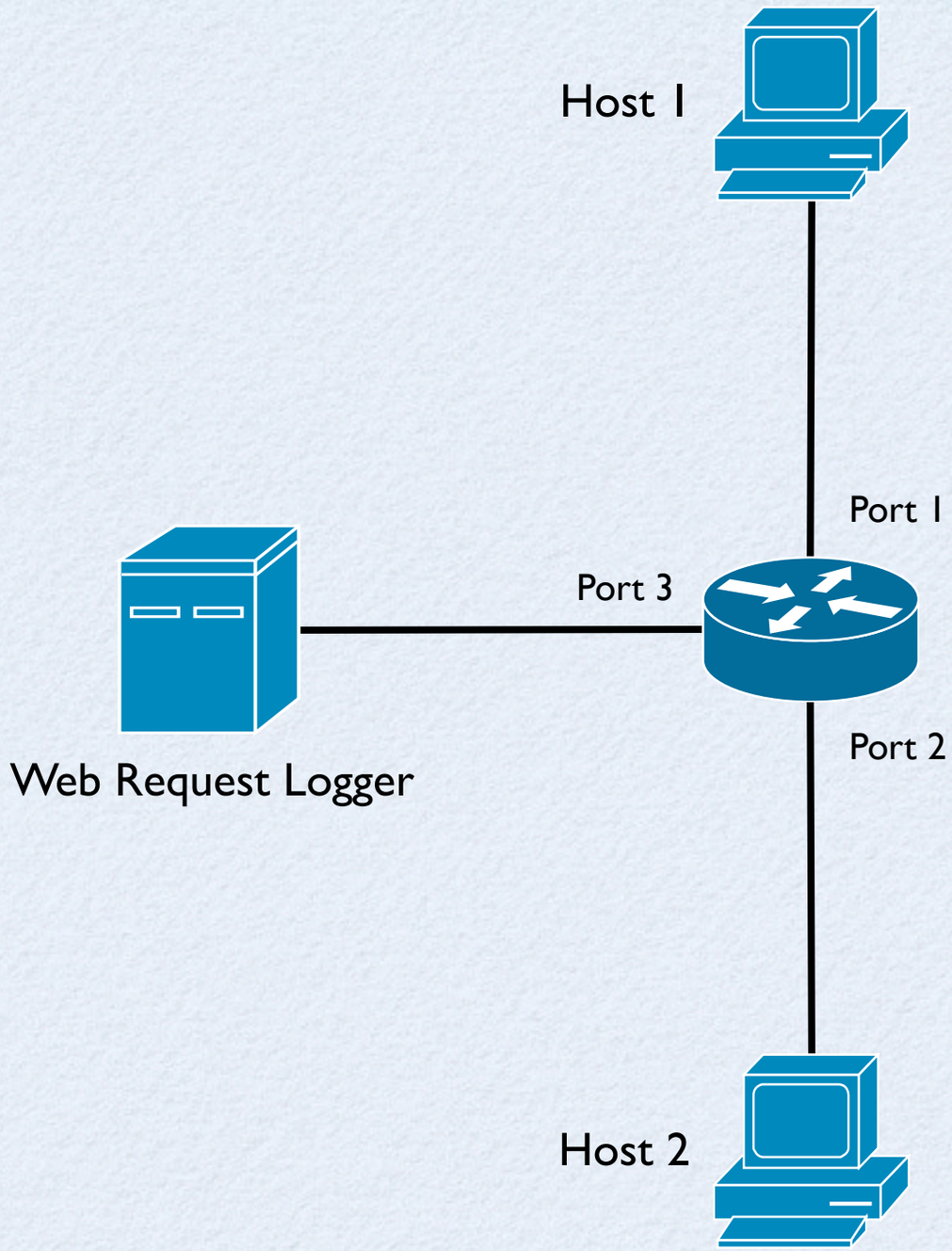
- symbolic execution / model checking
- finds several bugs statically
- seconds to days to test, no runtime cost
- cannot prove the absence of bugs

Verified Controllers: this talk

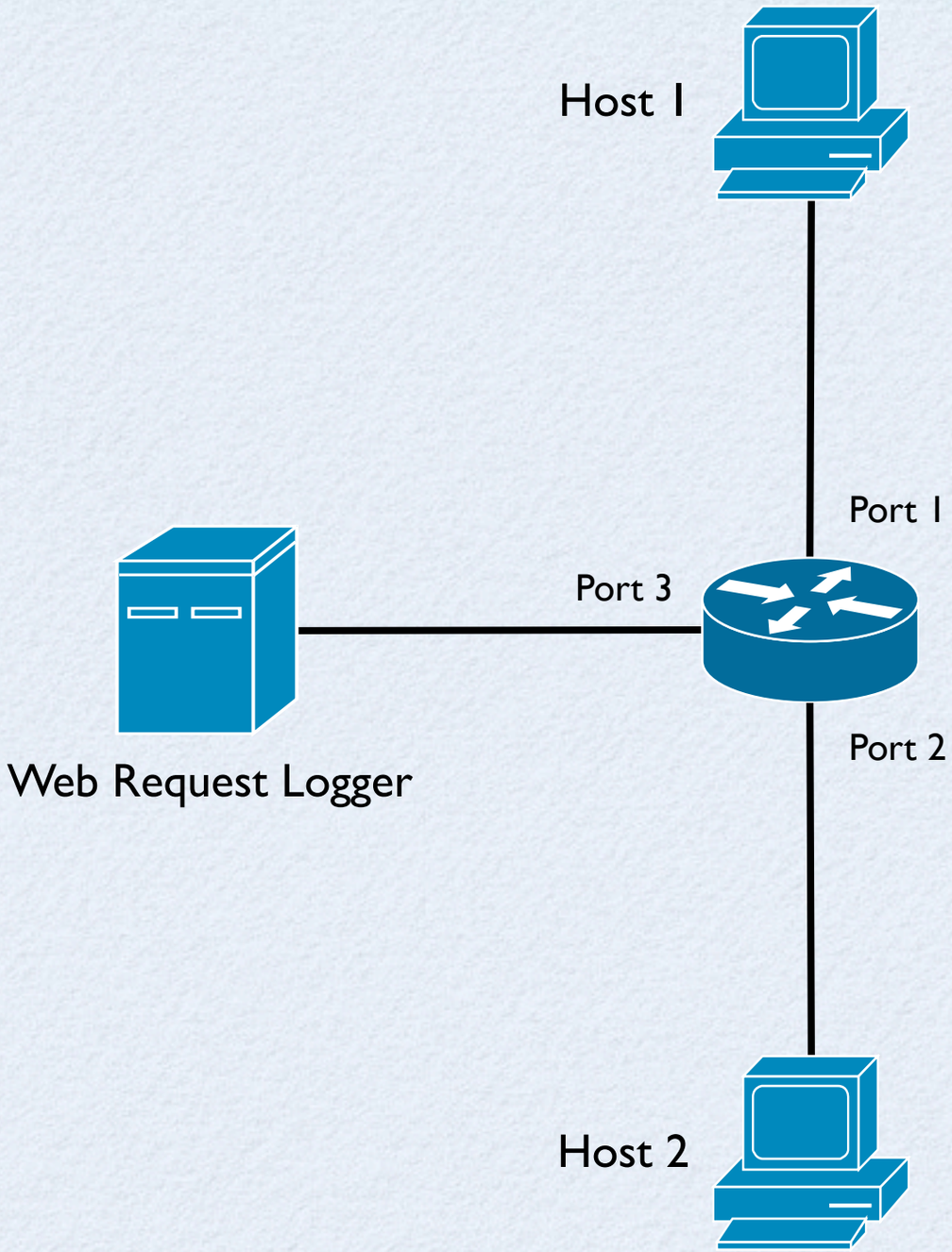
- program verification
- proof of correctness
- no runtime cost
- proof *w.r.t.* a detailed model of OpenFlow forwarding

Bugs in Controllers

Running Example

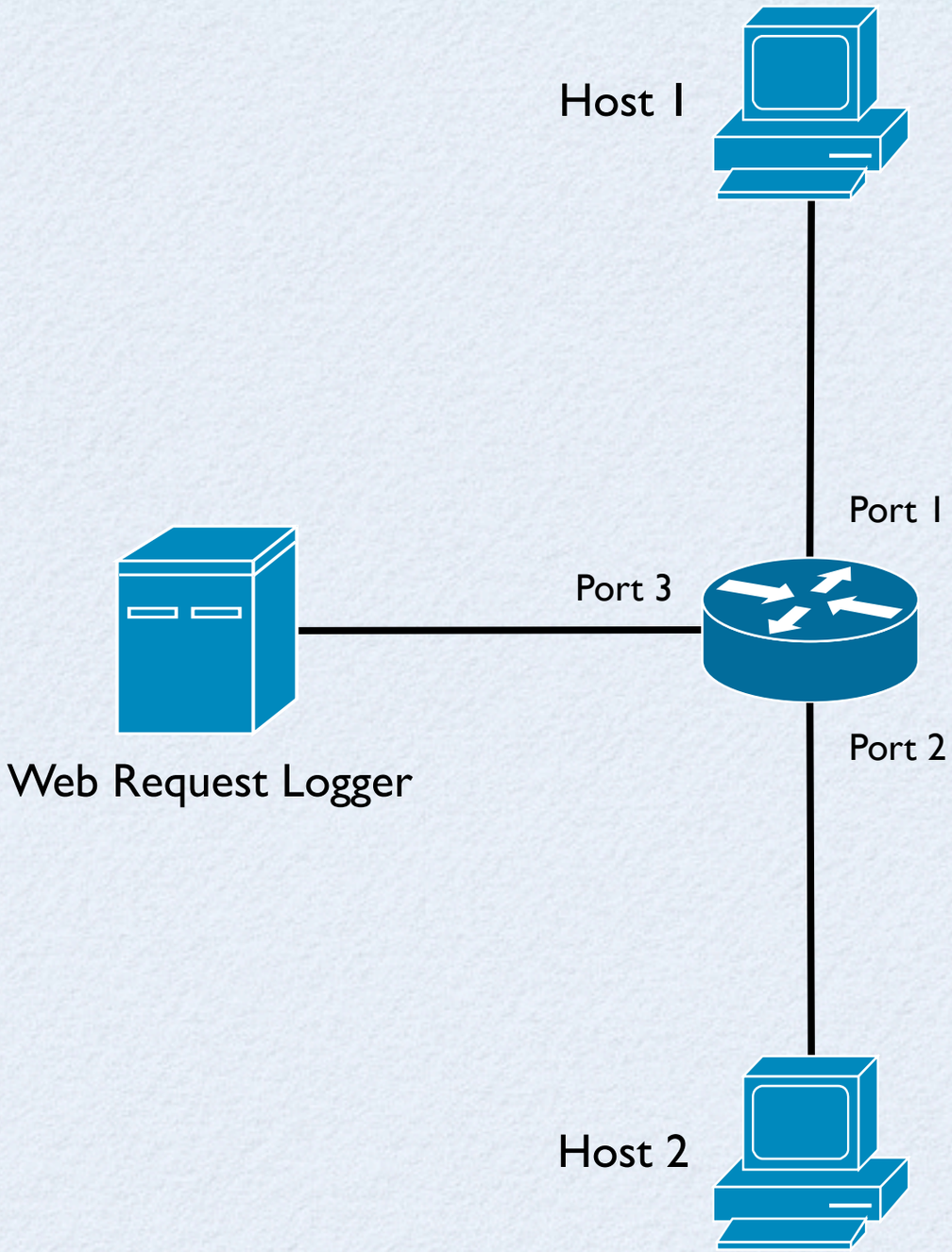


Running Example



- Block SSH traffic
- Forward other traffic normally
- Log Web requests

Running Example



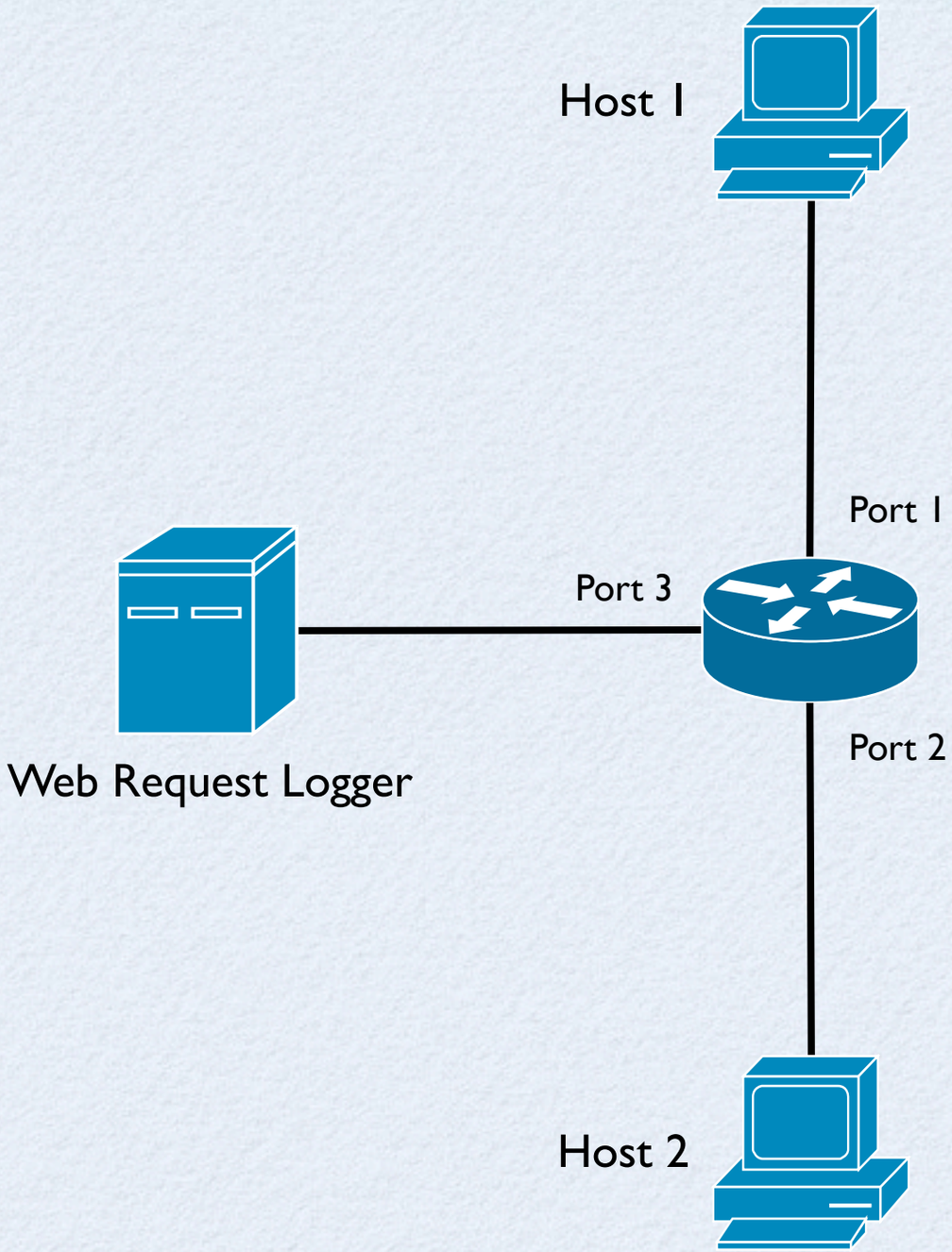
- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

- Log Web requests

Running Example



- Block SSH traffic

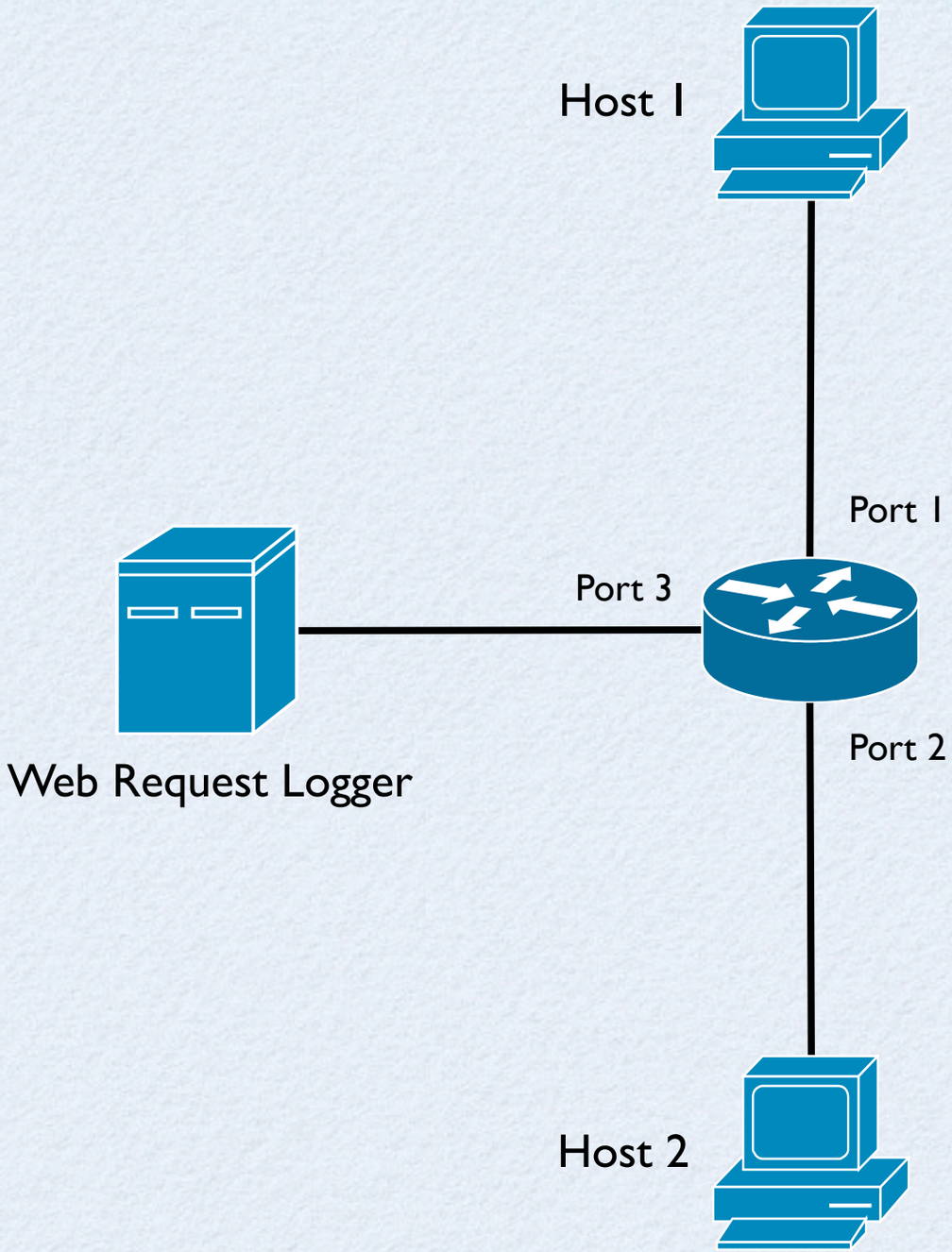
  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1
  dstIP == H2 ⟹ Fwd 2

- Log Web requests

Running Example



- Block SSH traffic

  dstPort == 22 ⟹ Drop

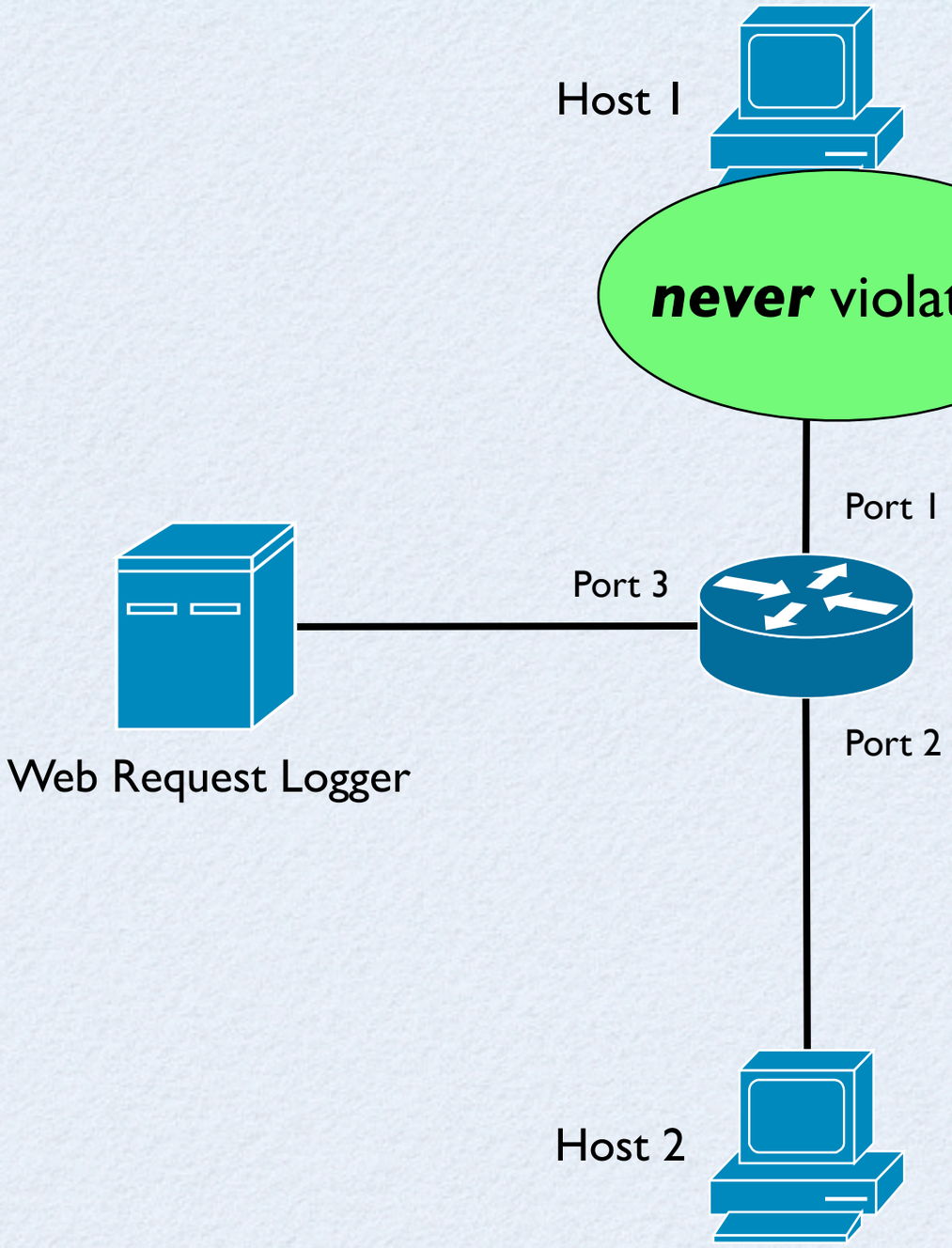
- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1
  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

Running Example



never violated

- Block SSH traffic

  dstPort == 22 ⟹ Drop

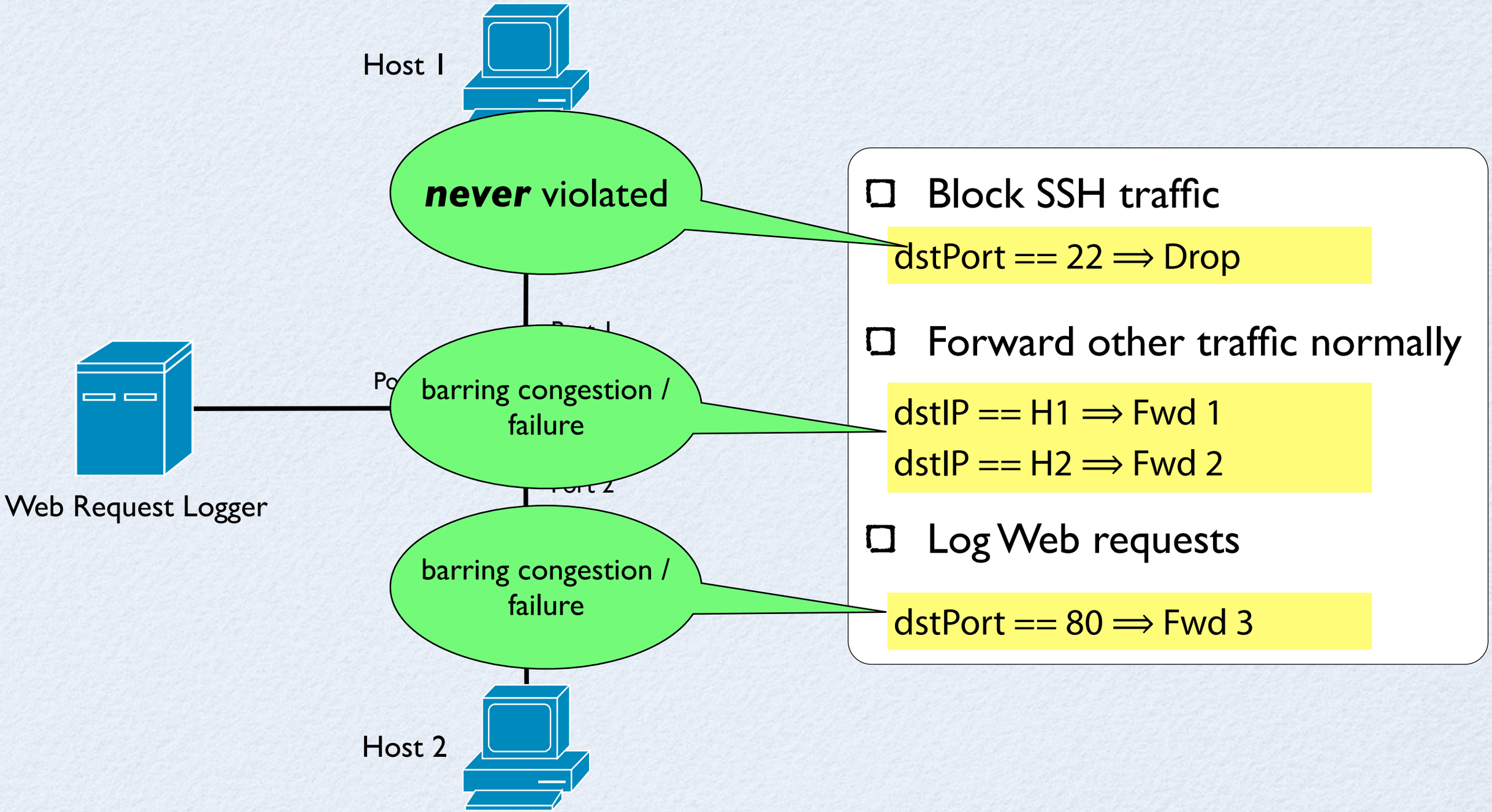
- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1
  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

Running Example



Running Example

- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1

  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

Running Example

Priority	Pattern	Action

- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1

  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

Running Example

Priority	Pattern	Action
10	`dstPort: 22`	`[]`

- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1

  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

Running Example

Priority	Pattern	Action
10	`dstPort: 22`	`[]`
9	`dstIP: H1, dstPort: 80`	`Fwd 1, Fwd 3`
9	`dstIP: H2, dstPort: 80`	`Fwd 2, Fwd 3`

- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1

  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

Running Example

Priority	Pattern	Action
10	`dstPort: 22`	`[]`
9	`dstIP: H1, dstPort: 80`	`Fwd 1, Fwd 3`
9	`dstIP: H2, dstPort: 80`	`Fwd 2, Fwd 3`
8	`dstIP: H1`	`Fwd 1`
8	`dstIP: H2`	`Fwd 2`

- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1

  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])
```

Priority	**Pattern**	**Action**
10	dstPort: 22	[]
9	dstIP: H1, dstPort: 80	Fwd 1, Fwd 3
9	dstIP: H2, dstPort: 80	Fwd 2, Fwd 3
8	dstIP: H1	Fwd 1
8	dstIP: H2	Fwd 2

- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1

  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])
```

- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1

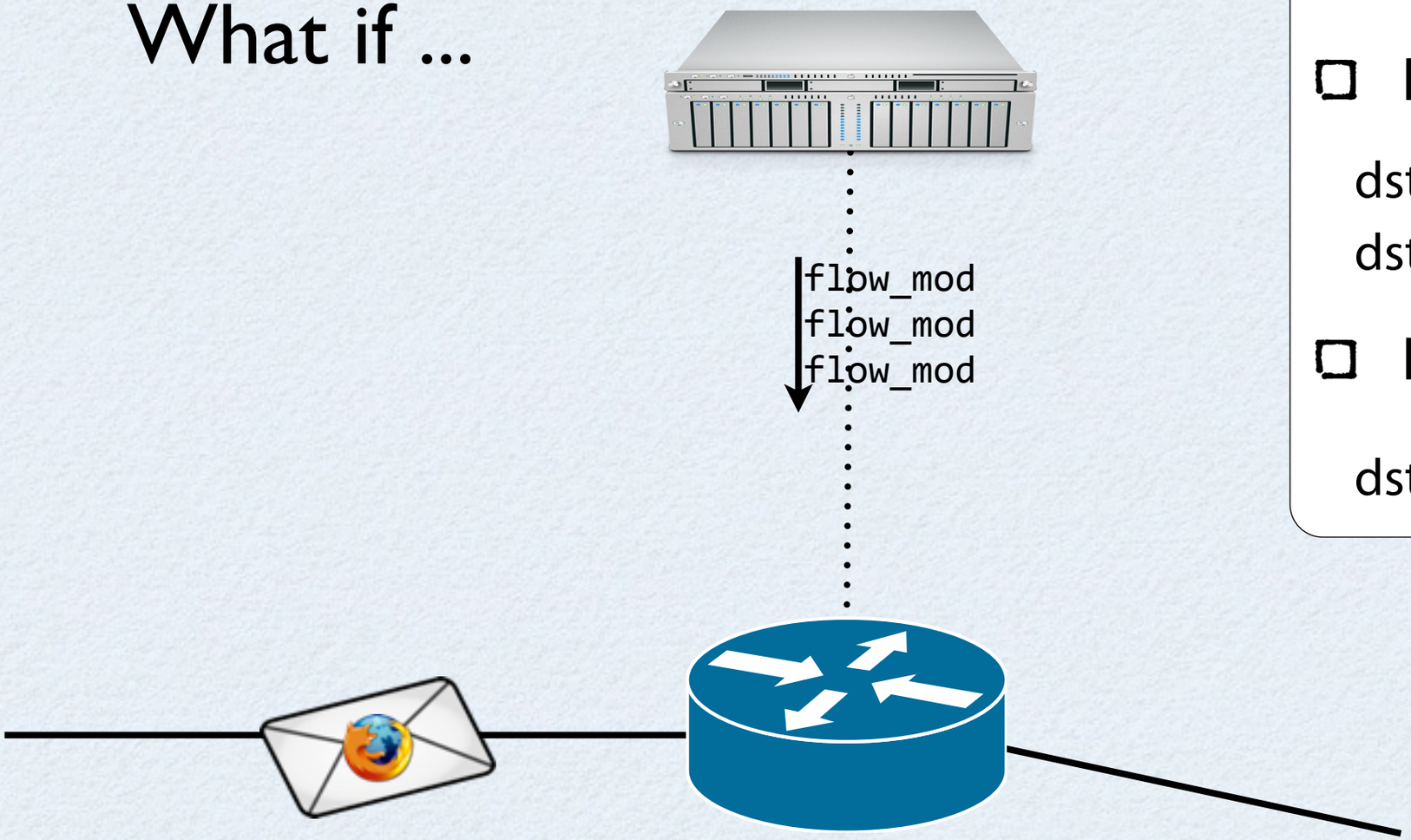
  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])
```

What if ...



- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1

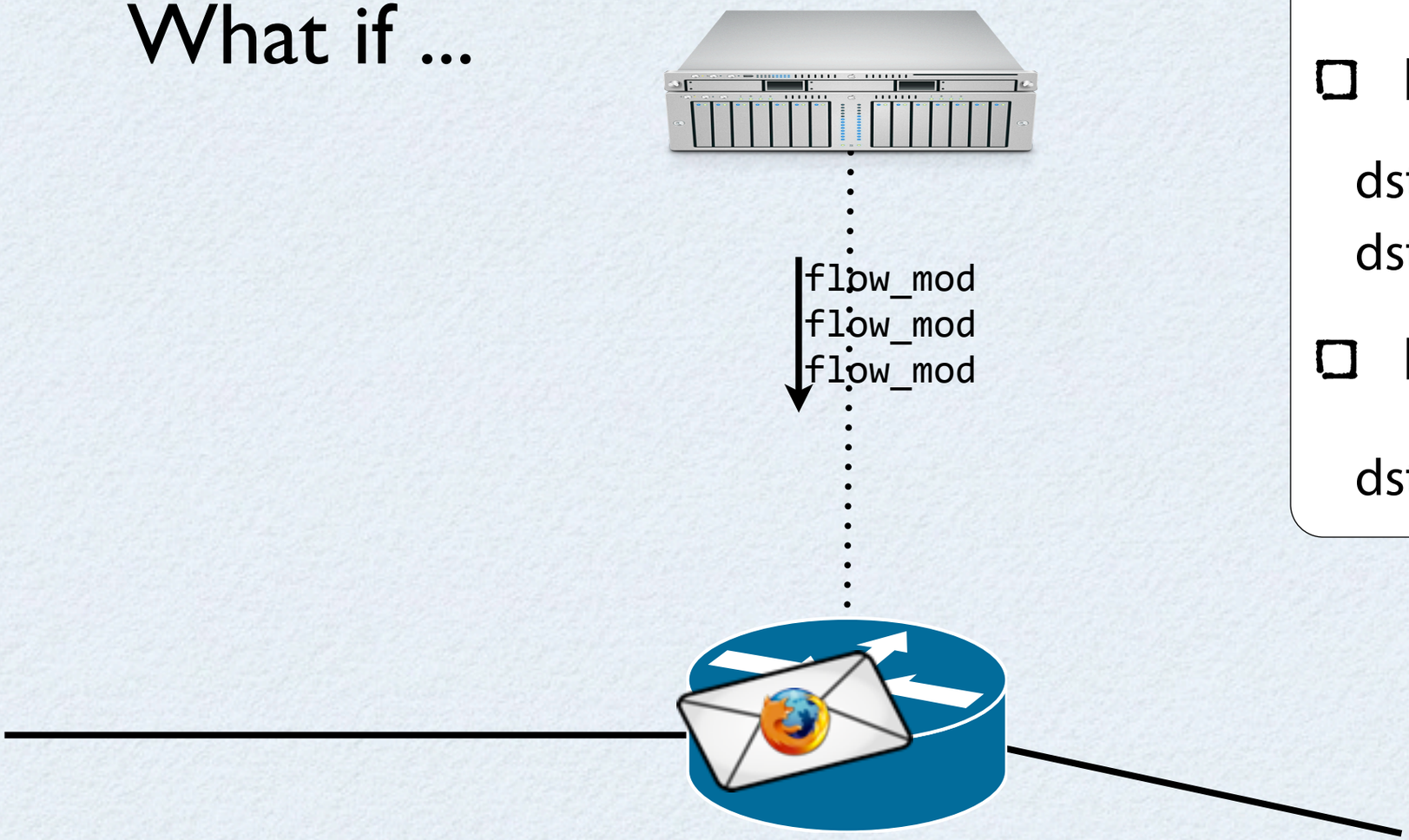
  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])
```

What if ...



- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1

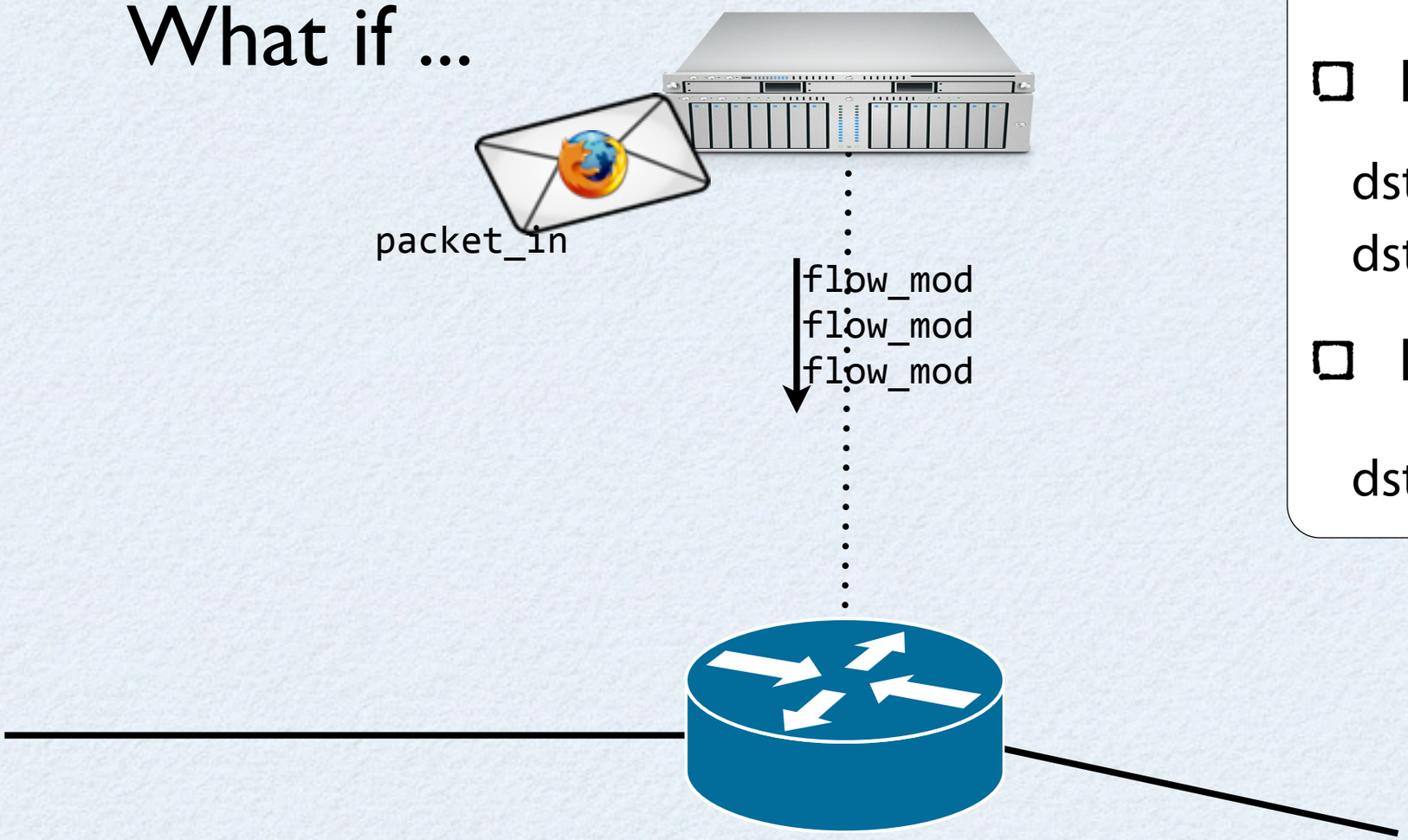
  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])
```

What if ...



- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1

  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])

def packet_in(sw, pkt):
  actions = []
  if pkt.dstPort == 22:
    return
  if pkt.dstPort == 80:
    actions = actions + [Fwd 3]
  if pkt.dstIP == H1:
    actions = actions + [Fwd 1]
  if pkt.dstIP == H2:
    actions = actions + [Fwd 2]
sw.packet_out(pkt, actions)
```

- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1

  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])

def packet_in(sw, pkt):
  actions = []
  if pkt.dstPort == 22:
    return
  if pkt.dstPort == 80:
    actions = actions + [Fwd 3]
  if pkt.dstIP == H1:
    actions = actions + [Fwd 1]
  if pkt.dstIP == H2:
    actions = actions + [Fwd 2]
  sw.packet_out(pkt, actions)
```

must be equivalent

- Block SSH traffic

  dstPort == 22 ⟹ Drop

- Forward other traffic normally

  dstIP == H1 ⟹ Fwd 1

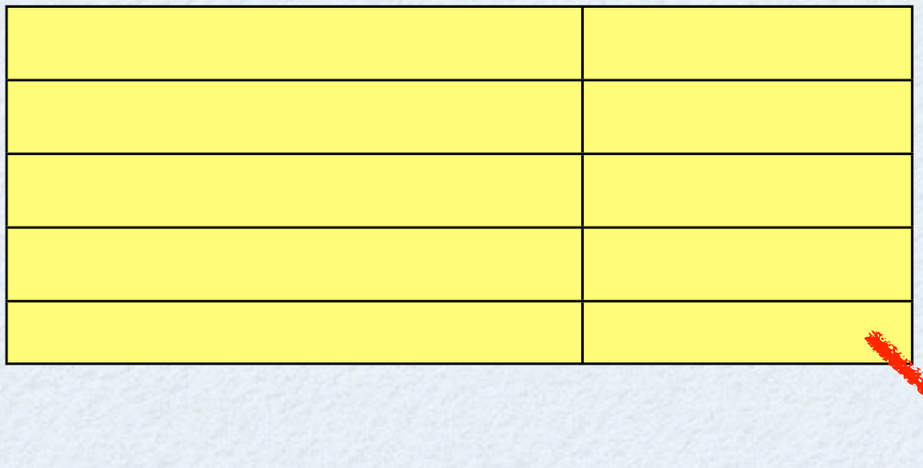
  dstIP == H2 ⟹ Fwd 2

- Log Web requests

  dstPort == 80 ⟹ Fwd 3

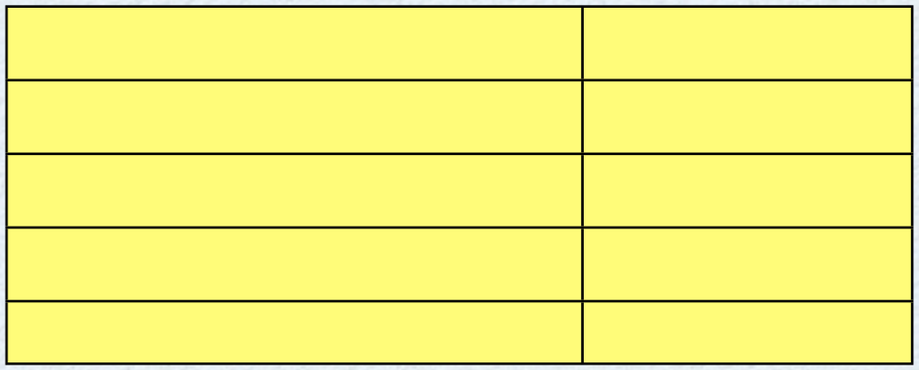
```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])
```

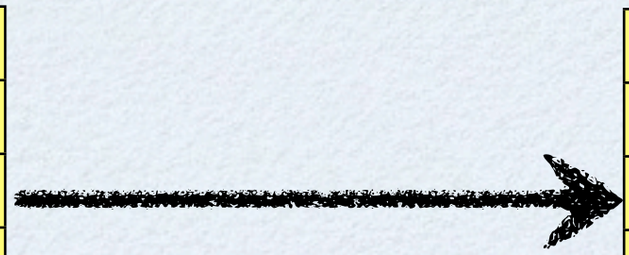
```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])
```

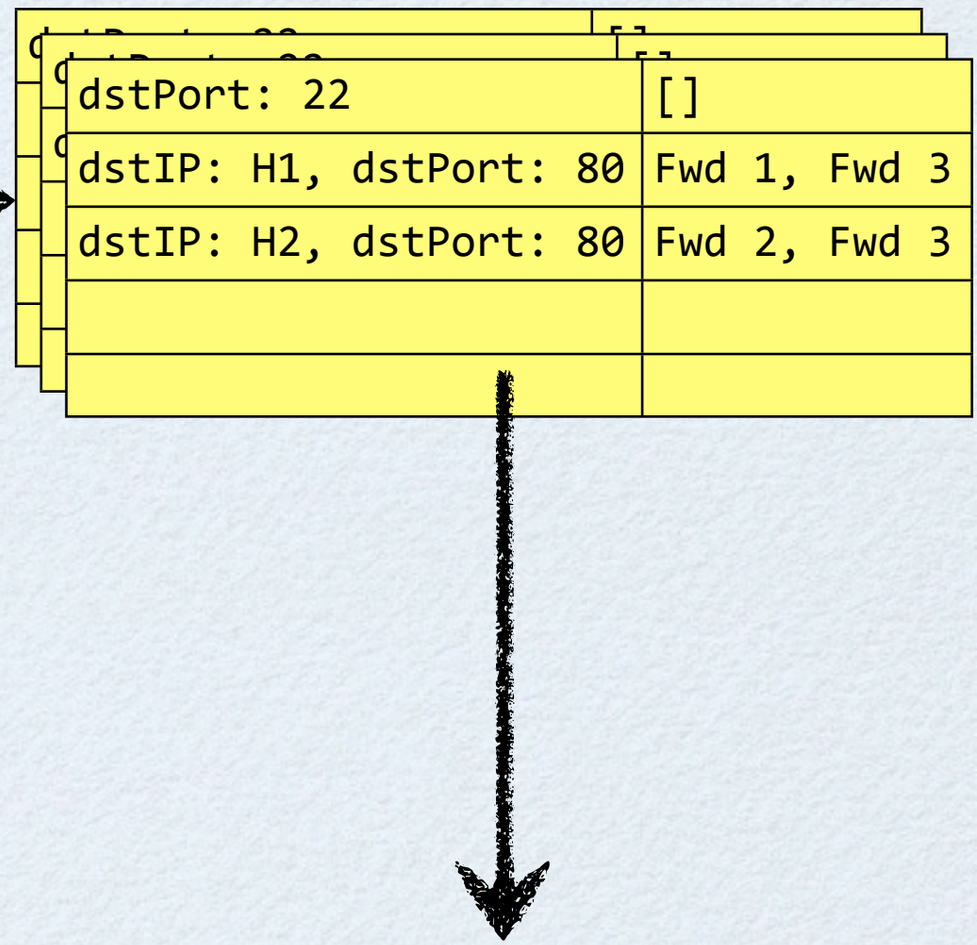


dstPort: 22	[]
dstIP: H1, dstPort: 80	Fwd 1, Fwd 3
dstIP: H2, dstPort: 80	Fwd 2, Fwd 3
dstIP: H1	Fwd 1
dstIP: H2	Fwd 2

```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])
```

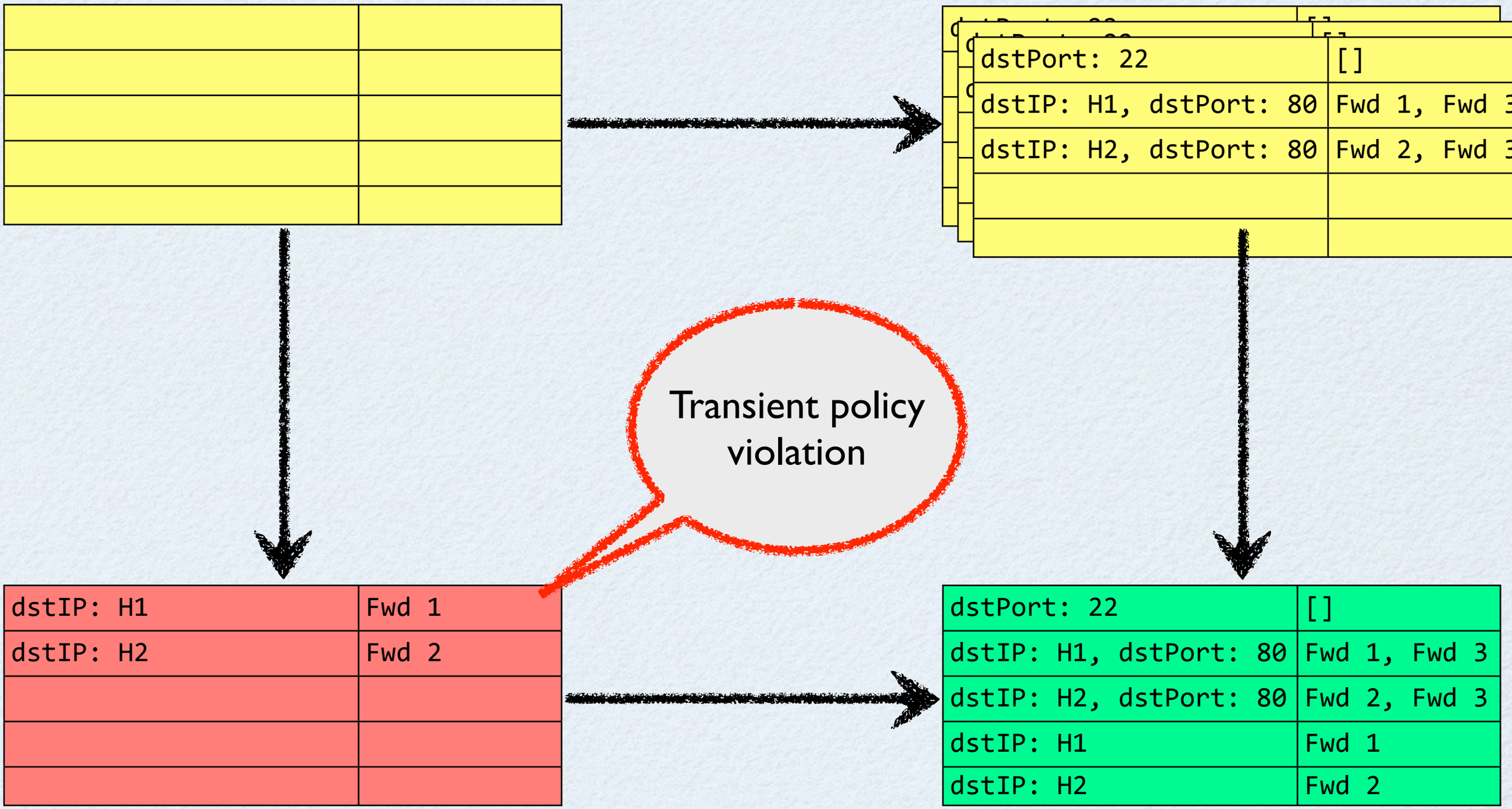






dstPort: 22	[]
dstIP: H1, dstPort: 80	Fwd 1, Fwd 3
dstIP: H2, dstPort: 80	Fwd 2, Fwd 3
dstIP: H1	Fwd 1
dstIP: H2	Fwd 2

```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])
```



```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.barrier_request()
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.barrier_request()
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])

def packet_in(sw, pkt):
  actions = []
  if pkt.dstPort == 22:
    return
if pkt.dstPort == 80:
  actions = actions + [Fwd 3]
if pkt.dstIP == H1:
  actions = actions + [Fwd 1]
if pkt.dstIP == H2:
  actions = actions + [Fwd 2]
sw.packet_out(pkt, actions)
```

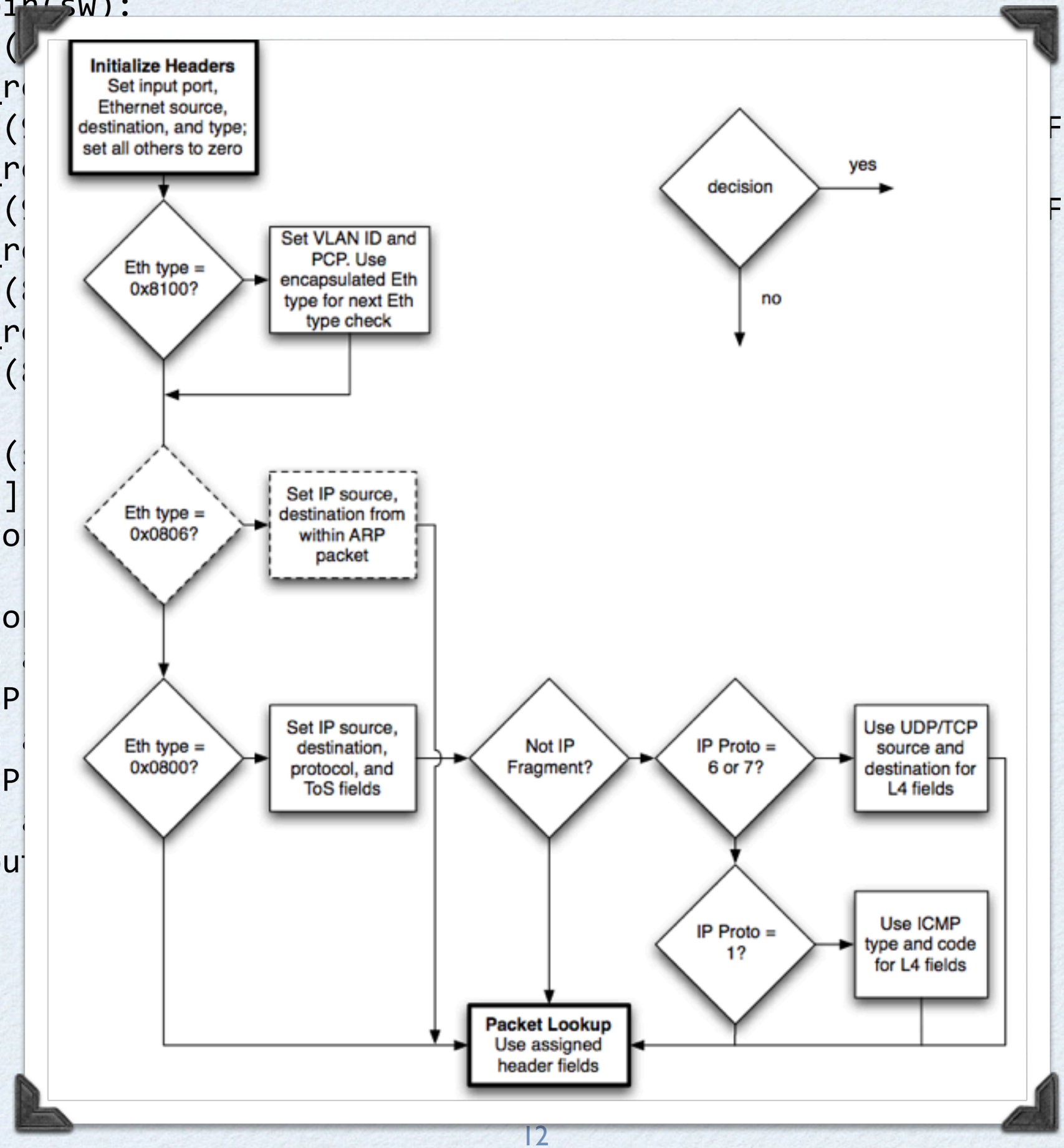
```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.barrier_request()
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.barrier_request()
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])

def packet_in(sw, pkt):
  actions = []
  if pkt.dstPort == 22:
    return
if pkt.dstPort == 80:
  actions = actions + [Fwd 3]
if pkt.dstIP == H1:
  actions = actions + [Fwd 1]
if pkt.dstIP == H2:
  actions = actions + [Fwd 2]
sw.packet_out(pkt, actions)
```

```
def switch_join(sw):
  sw.flow_mod(10, { dstPort: 22 }, [])
  sw.barrier_request()
  sw.flow_mod(9, { dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(9, { dstIP: H2, dstPort: 80 }, [Fwd 2,
  sw.barrier_request()
  sw.flow_mod(8, { dstIP: H1 }, [Fwd 1])
  sw.barrier_request()
  sw.flow_mod(8, { dstIP: H2 }, [Fwd 2])

def packet_in(sw, pkt):
  actions = []
  if pkt.dstPort == 22:
    return
if pkt.dstPort == 80:
  actions = actions + [Fwd 3]
if pkt.dstIP == H1:
  actions = actions + [Fwd 1]
if pkt.dstIP == H2:
  actions = actions + [Fwd 2]
sw.packet_out(pkt, actions)
```

Match *all* packets!?

```
def switch_join(sw):
  sw.flow_mod(10, { eth: 0x800, proto: 6, dstPort: 22 }, [])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H1 }, [Fwd 1])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H2 }, [Fwd 2])

def packet_in(sw, pkt):
  actions = []
  if pkt.dstPort == 22:
    return
  if pkt.dstPort == 80:
    actions = actions + [Fwd 3]
  if pkt.dstIP == H1:
    actions = actions + [Fwd 1]
  if pkt.dstIP == H2:
    actions = actions + [Fwd 2]
  sw.packet_out(pkt, actions)
```

```
def switch_join(sw):
  sw.flow_mod(
  sw.barrier_r
  sw.flow_mod(                                          Fwd 1, Fwd 3])
  sw.barrier_r
  sw.flow_mod(                                          Fwd 2, Fwd 3])
  sw.barrier_r
  sw.flow_mod(
  sw.barrier_r
  sw.flow_mod(

def packet_in(
  actions = []
  if pkt.dstPo
    return
if pkt.dstPo
  actions =
if pkt.dstIP
  actions =
if pkt.dstIP
  actions =
sw.packet_ou
```



Recap: Bugs in Controllers

Recap: Bugs in Controllers

Violation	**Guilty Controllers**

Recap: Bugs in Controllers

Violation	**Guilty Controllers**
handles packet_in incorrectly	

Recap: Bugs in Controllers

Violation	**Guilty Controllers**
handles packet_in incorrectly	PANE (absolved)

Recap: Bugs in Controllers

Violation	**Guilty Controllers**
handles packet_in incorrectly	PANE (absolved)
does not use barriers	

Recap: Bugs in Controllers

Violation	**Guilty Controllers**
handles packet_in incorrectly	PANE (absolved)
does not use barriers	NetCore, PANE, Nettle-FRP

Recap: Bugs in Controllers

Violation	**Guilty Controllers**
handles packet_in incorrectly	PANE (absolved)
does not use barriers	NetCore, PANE, Nettle-FRP
synthesizes malformed patterns	NetCore, PANE, Nettle-FRP†

† Nettle is only affected by missing protocol numbers

Recap: Bugs in Controllers

Violation	**Guilty Controllers**
handles packet_in incorrectly	PANE (absolved)
does not use barriers	NetCore, PANE, Nettle-FRP
synthesizes malformed patterns	NetCore, PANE, Nettle-FRP†
optimizer breaks flow tables	NetCore (absolved)

† Nettle is only affected by missing protocol numbers

Recap: Bugs in Controllers

Violation	**Guilty Controllers**
handles packet_in incorrectly	PANE (absolved)
does not use barriers	NetCore, PANE, Nettle-FRP
synthesizes malformed patterns	NetCore, PANE, Nettle-FRP†
optimizer breaks flow tables	NetCore (absolved)

discovered by formalizing OpenFlow

† Nettle is only affected by missing protocol numbers

SDN Facilitates Formal Methods

Run-time Monitoring: VeriFlow

by Khurshid, Zhou, Caesar, and Godfrey

- runtime property checking
- detects errors dynamically
- small runtime overhead
- fixes must be out-of-band

Static Bug-Finding: NICE

by Canini, Venzano, Perešíni, Kostić, and Rexford

- symbolic execution / model checking
- finds several bugs statically
- seconds to days to test, no runtime cost
- cannot prove the absence of bugs

Verified Controllers: this talk

- program verification
- proof of correctness
- no runtime cost
- proof *w.r.t.* a detailed model of OpenFlow

VeriFlow: Reachability

Violation	**Guilty Controllers**
handles packet_in incorrectly	PANE (absolved)
does not use barriers	NetCore, PANE, Nettle-FRP
synthesizes malformed patterns	NetCore, PANE, Nettle-FRP†
optimizer breaks flow tables	NetCore (absolved)

† Nettle is only affected by missing protocol numbers

VeriFlow: Reachability

Violation	Guilty Controllers
handles packet_in incorrectly	NetCore, PANE, Nettle-FRP (absolved)
does not use barriers	NetCore, PANE, Nettle-FRP
synthesizes malformed patterns	NetCore, PANE, Nettle-FRP†
optimizer breaks flow tables	NetCore (absolved)

liveness:
eventually produces
packet_out

† Nettle is only affected by missing protocol numbers

VeriFlow: Reachability

Violation	**Gu[illegible]ollers**
handles packet_in incorrectly	[illegible]ived)
does not use barriers	[illegible]e-FRP
synthesizes malformed patterns	NetCore, PANE, Nettle-FRP†
optimizer breaks flow tables	NetCore (absolved)

liveness: eventually produces packet_out

"acceptable violation"?

† Nettle is only affected by missing protocol numbers

NICE: Library of Properties

Violation	**Guilty Controller**
handles packet_in incorrectly	PANE (absolved)
does not use barriers	NetCore, PANE, Nettle-FRP
synthesizes malformed patterns	NetCore, PANE, Nettle-FRP†
optimizer breaks flow tables	NetCore (absolved)

† Nettle is only affected by missing protocol numbers

NICE: Library of Properties

Violation	Guilty Controller
handles packet_in incorrectly	PANE (absolved)
does not use barriers	e-FRP
synthesizes malformed patterns	NetCore, PANE, Nettle-FRP†
optimizer breaks flow tables	NetCore (absolved)

assumes FIFO message processing

† Nettle is only affected by missing protocol numbers

NICE: Library of Properties

Violation	Guilty Controller
handles packet_in incorrectly	PANE (absolved)
does not use barriers	e-FRP
synthesizes malformed patterns	NetCore, PANE, Nettle-FRP†
optimizer breaks flow tables	

assumes FIFO message processing

requires high-level policy

† Nettle is only affected by missing protocol numbers

A Sample of Recent Verification

Successes:

Tools:

Textbooks:

A Sample of Recent Verification

Successes:

Operating Systems	Compilers
seL4 SOSP 2009	CompCert CACM, July 2009
Verve PLDI 2010	F* POPL 2012

Tools:

Textbooks:

A Sample of Recent Verification

Successes:

Operating Systems	Compilers
seL4 SOSP 2009	CompCert CACM, July 2009
Verve PLDI 2010	F* POPL 2012

Tools:

Textbooks:

A Sample of Recent Verification

Successes:

Operating Systems	Compilers
seL4 SOSP 2009	CompCert CACM, July 2009
Verve PLDI 2010	F* POPL 2012

Tools:

Textbooks:

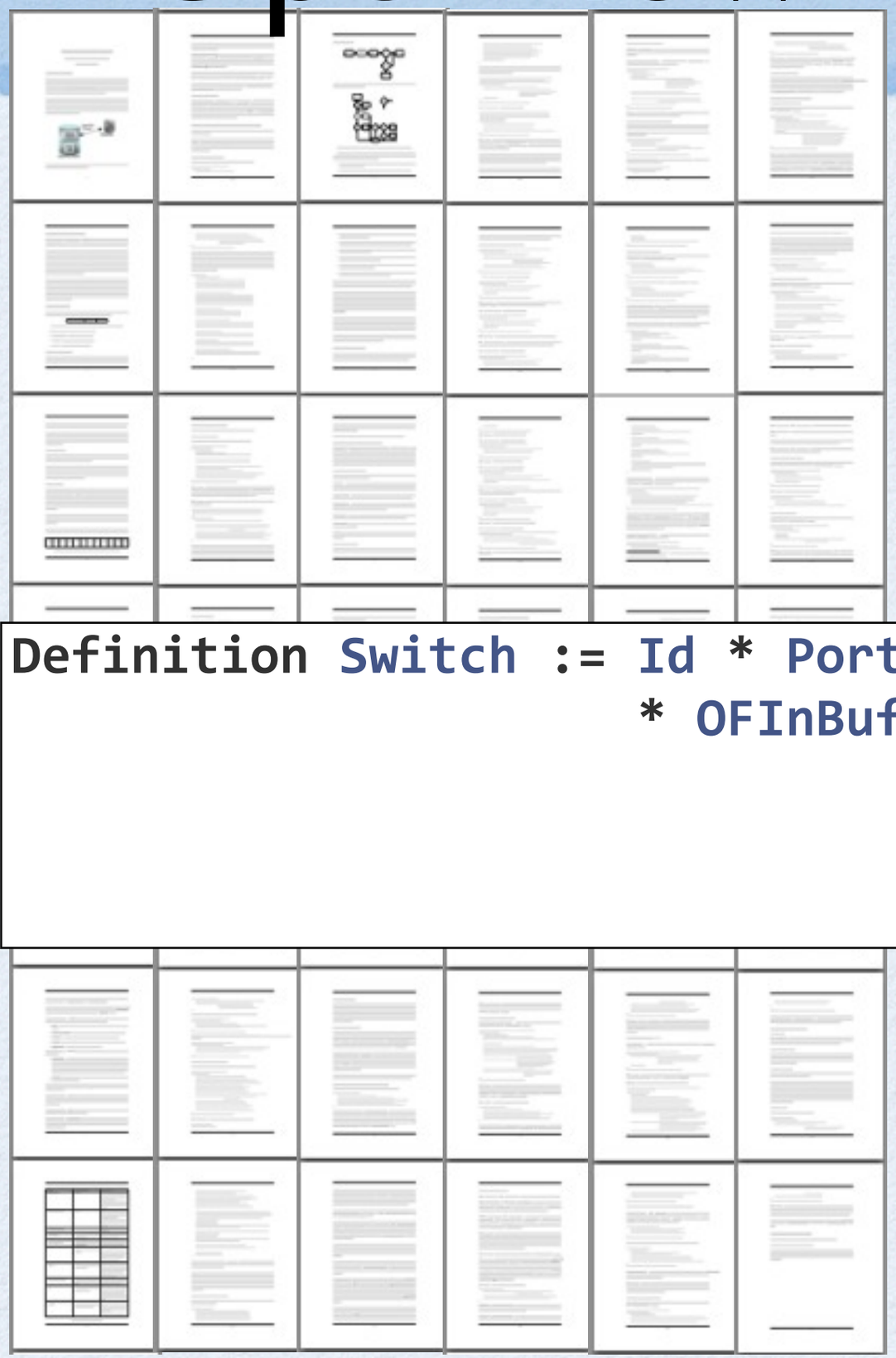
Certified Programming with Dependent Types

Mathematical Foundations of SDN

OpenFlow 1.0 specification

42 pages of prose and C structs ...

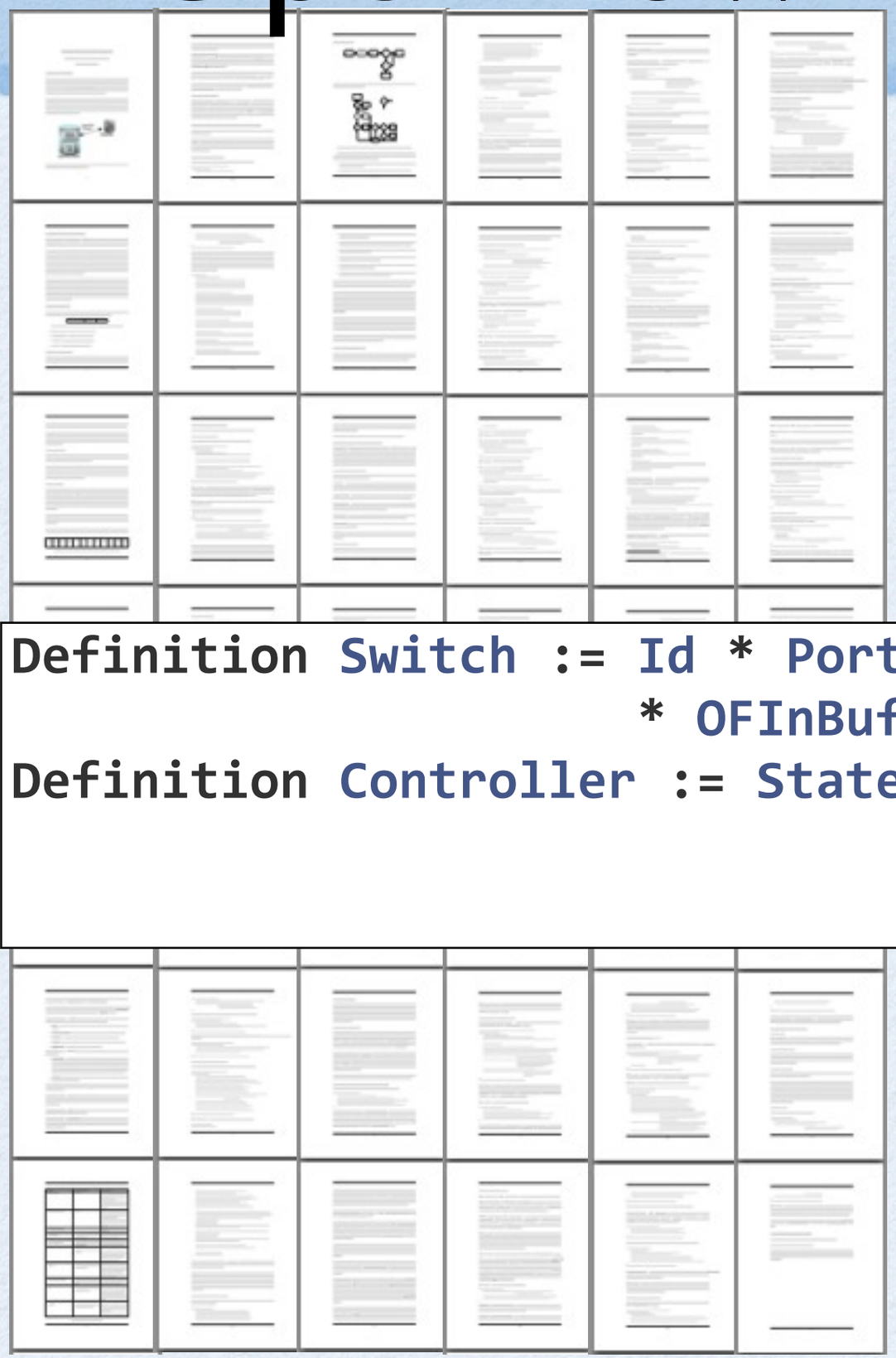
OpenFlow 1.0 specification



42 pages of prose and C structs ...

```
Definition Switch := Id * Ports * FlowTbl * InBuf * OutBuf
                     * OFInBuf * OFOutBuf.
```

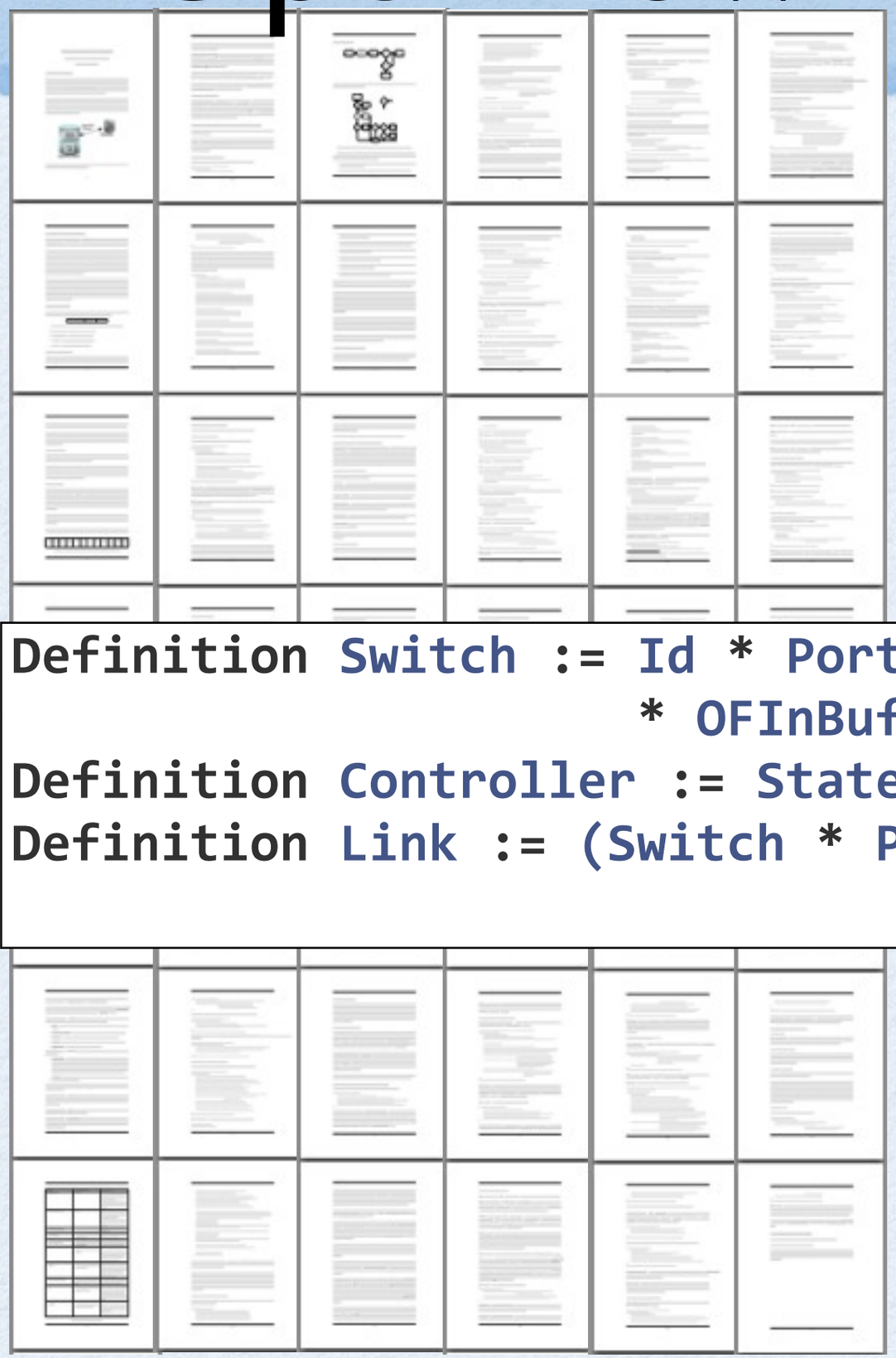
OpenFlow 1.0 specification



42 pages of prose and C structs ...

```
Definition Switch := Id * Ports * FlowTbl * InBuf * OutBuf
                     * OFInBuf * OFOutBuf.
Definition Controller := State * InFunc * OutFunc.
```

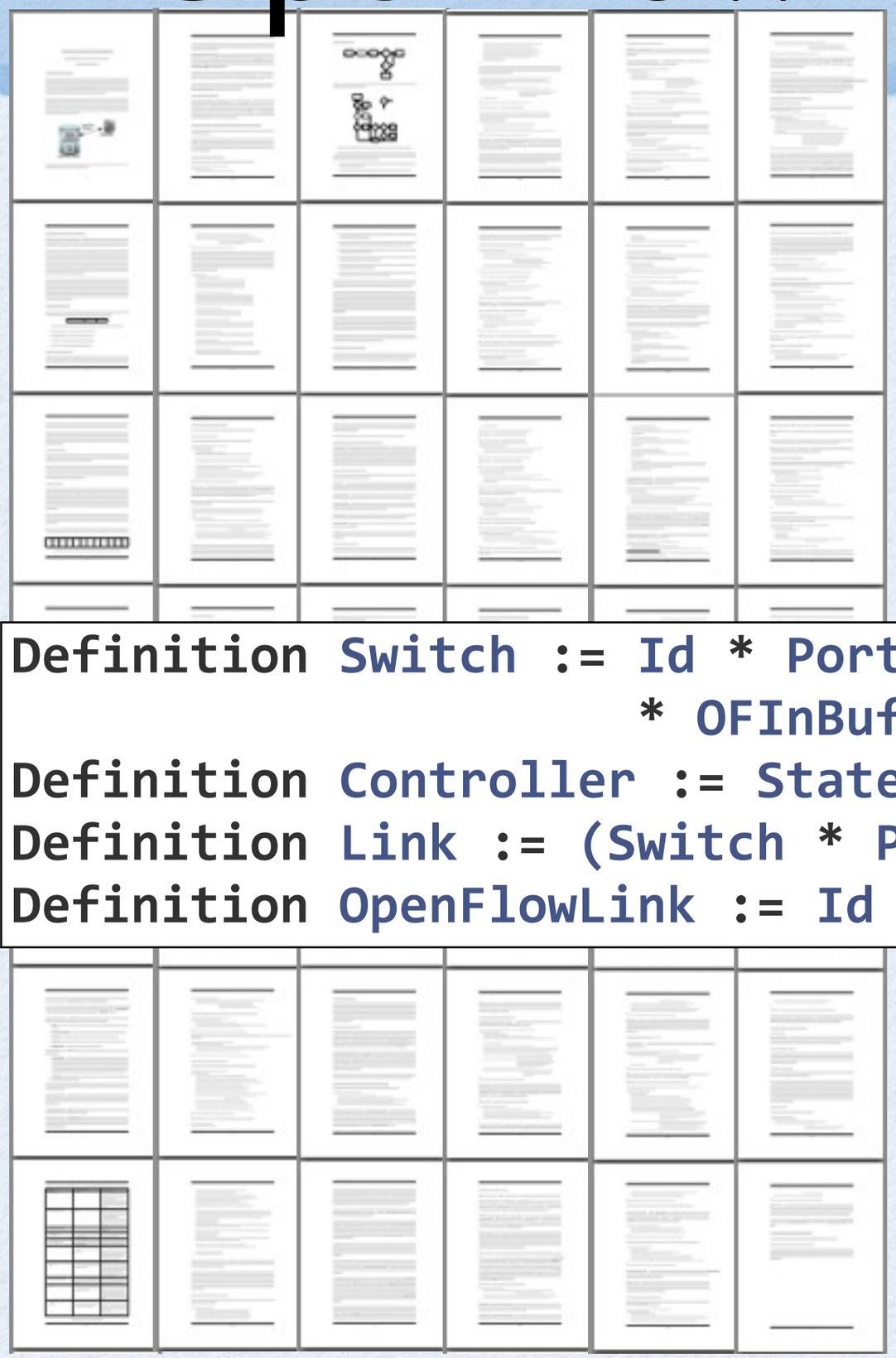
OpenFlow 1.0 specification



42 pages of prose and C structs ...

```
Definition Switch := Id * Ports * FlowTbl * InBuf * OutBuf
                     * OFInBuf * OFOutBuf.
Definition Controller := State * InFunc * OutFunc.
Definition Link := (Switch * Port) * list Packet * (Switch * Port).
```

OpenFlow 1.0 specification

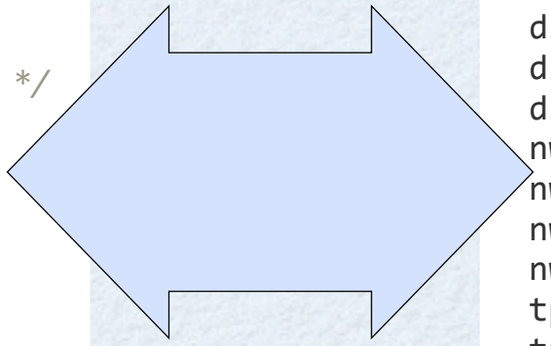


42 pages of prose and C structs ...

```
Definition Switch := Id * Ports * FlowTbl * InBuf * OutBuf
                     * OFInBuf * OFOutBuf.
Definition Controller := State * InFunc * OutFunc.
Definition Link := (Switch * Port) * list Packet * (Switch * Port).
Definition OpenFlowLink := Id * list SwitchMsg * list CtrlMsg.
```

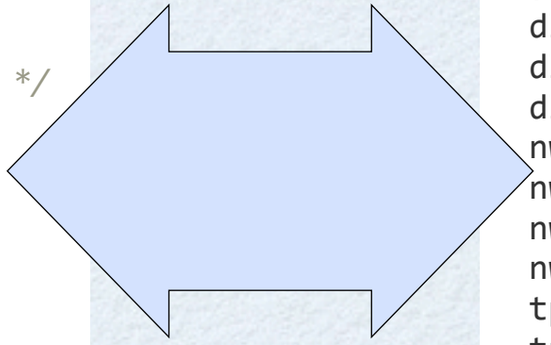
```c
/* Fields to match against flows */
struct ofp_match {
    uint32_t wildcards;         /* Wildcard fields. */
    uint16_t in_port;           /* Input switch port. */
    uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
    uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
    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);
```

```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);
```

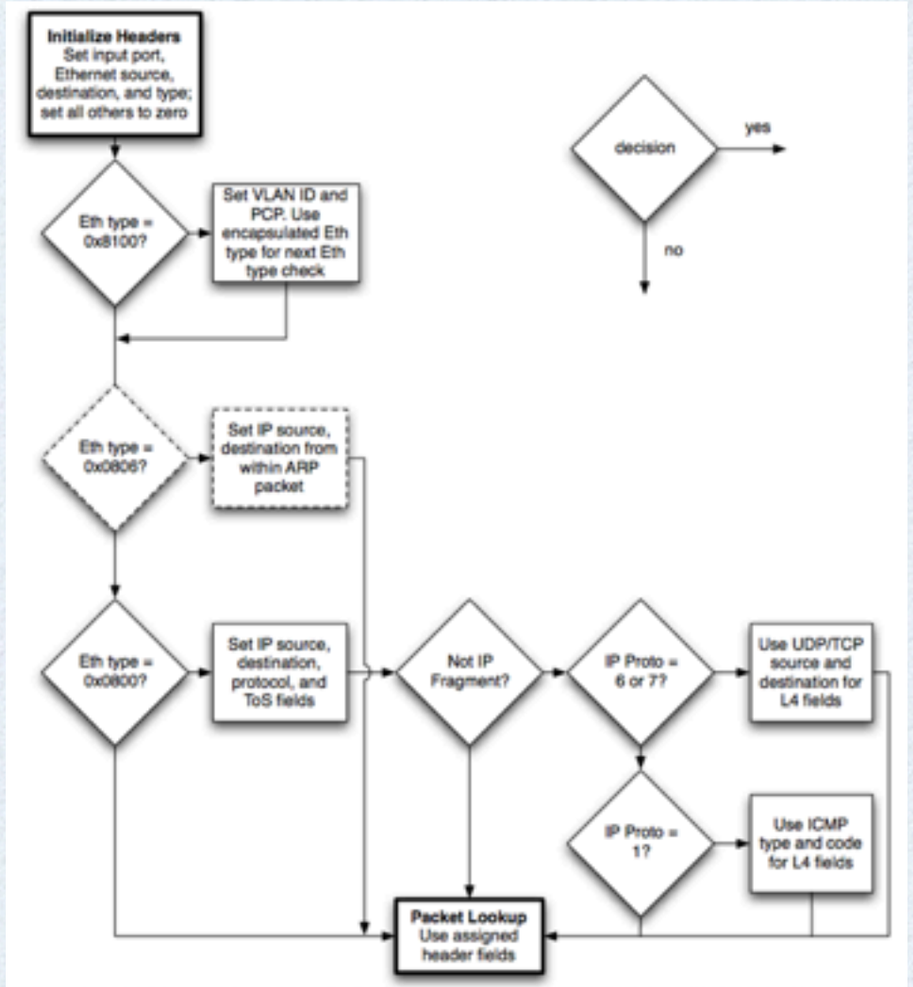


```
Record Pattern : Type := MkPattern {
  dlSrc : Wildcard EthernetAddress;
  dlDst : Wildcard EthernetAddress;
  dlType : Wildcard EthernetType;
  dlVlan : Wildcard VLAN;
  dlVlanPcp : Wildcard VLANPriority;
  nwSrc : Wildcard IPAddress;
  nwDst : Wildcard IPAddress;
  nwProto : Wildcard IPProtocol;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

```
/* 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);
```

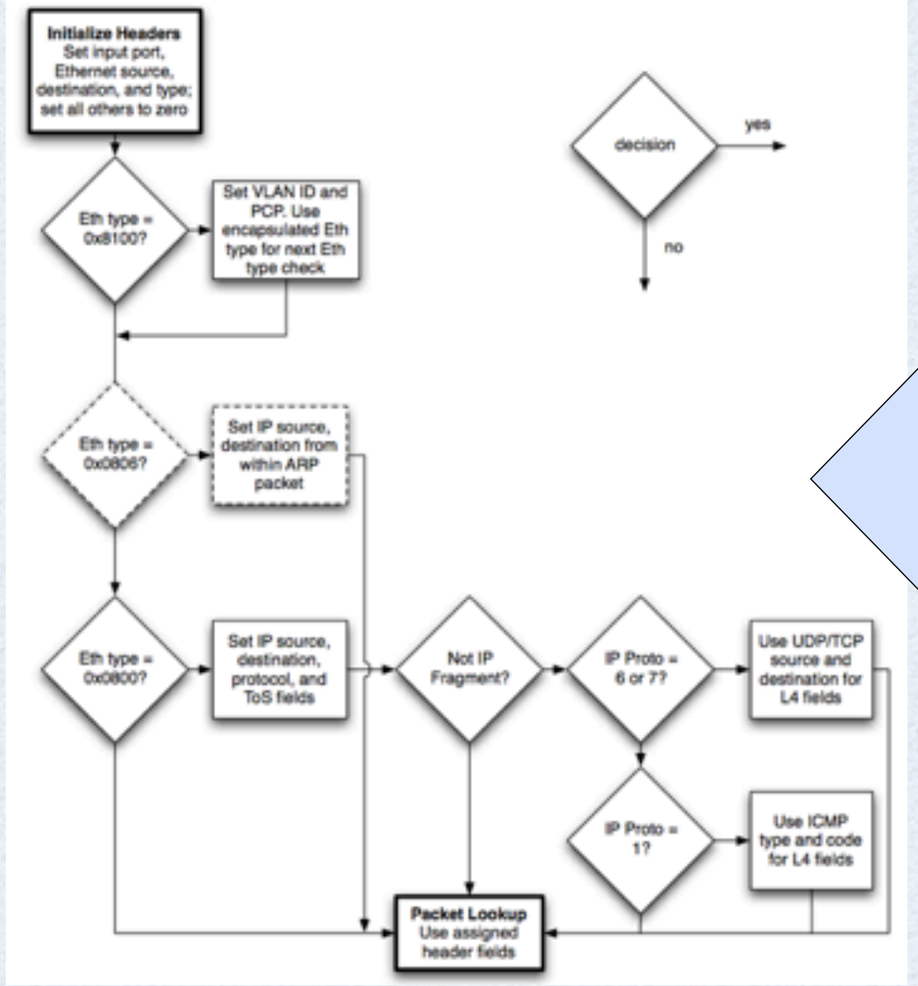


```
Record Pattern : Type := MkPattern {
  dlSrc : Wildcard EthernetAddress;
  dlDst : Wildcard EthernetAddress;
  dlType : Wildcard EthernetType;
  dlVlan : Wildcard VLAN;
  dlVlanPcp : Wildcard VLANPriority;
  nwSrc : Wildcard IPAddress;
  nwDst : Wildcard IPAddress;
  nwProto : Wildcard IPProtocol;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



```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);
```

```
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
}.
```

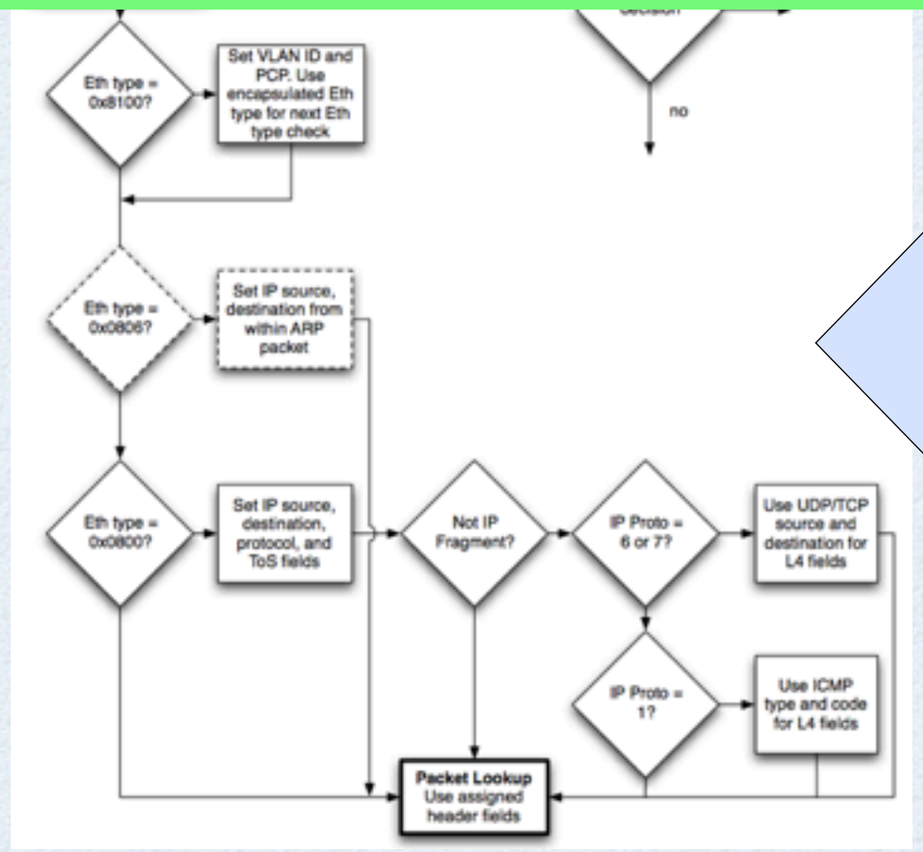



```
Definition Pattern_inter (p p':Pattern) :=
  let dlSrc := Wildcard_inter EthernetAddress.eqdec (ptrnDlSrc p) (ptrnDlSrc p') in
  let dlDst := Wildcard_inter EthernetAddress.eqdec (ptrnDlDst p) (ptrnDlDst p') in
  let dlType := Wildcard_inter Word16.eqdec (ptrnDlType p) (ptrnDlType p') in
  let dlVlan := 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)).
```

detailed model of matching, forwarding, and flow table update

```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);
```

```
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
}.
```



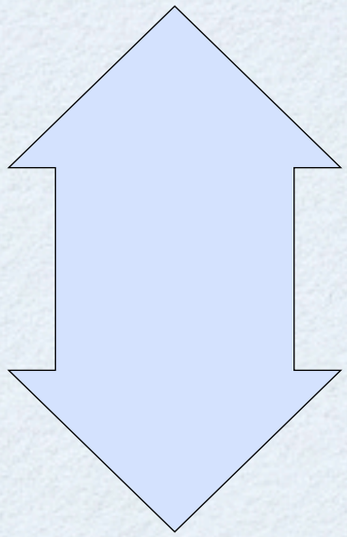

```
Definition Pattern_inter (p p':Pattern) :=
  let dlVlan := Wildcard_inter Word16.eqdec (ptrnDlVlan p) (ptrnDlVlan p') in
  let dlVlanPcp := Wildcard_inter Word8.eqdec (ptrnDlVlanPcp p) (ptrnDlVlanPcp p') in
  let nwSrc := Wildcard_inter Word32.eqdec (ptrnNwSrc p) (ptrnNwSrc p') in
  let nwDst := 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)).
```

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

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

"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 := MultiSet Packet.
Definition OutBuf := MultiSet Packet.
Definition OFInBuf := MultiSet SwitchMsg.
Definition OFOutBuf := MultiSet CtrlMsg.
```

"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 := MultiSet Packet.
Definition OutBuf := MultiSet Packet.
Definition OFInBuf := MultiSet SwitchMsg.
Definition OFOutBuf := MultiSet CtrlMsg.
```

State : abstract type

f_{out} : State → Switch × CtrlMsg × State'

f_{in} : Switch × SwitchMsg × State → State'

controller model: fully abstract

Featherweight OpenFlow

- ☑ detailed model of matching, forwarding, and flow table update
- ☑ essential asynchrony: packet buffers, message reordering, and barriers
- ☑ controller model: fully abstract

Featherweight OpenFlow

- ☑ detailed model of matching, forwarding, and flow table update
- ☑ essential asynchrony: packet buffers, message reordering, and barriers
- ☑ controller model: fully abstract

Only possible because the specification is simple and open

Featherweight OpenFlow

☑ detailed model of matching, forwarding, and flow table update

☑ essential asynchrony: packet buffers, message reordering, and barriers

☑ controller model: fully abstract

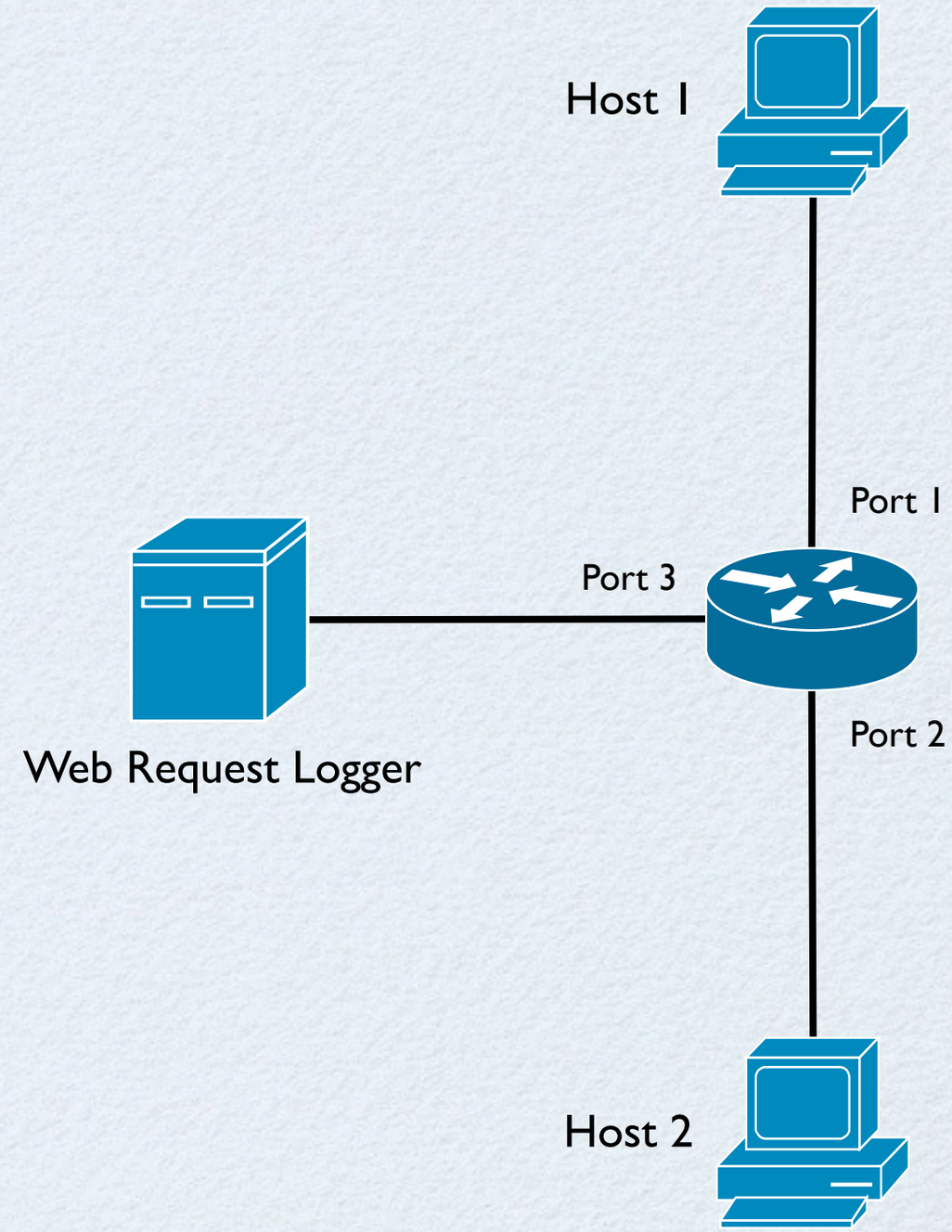
Only possible because the specification is simple and open

- **Future Work:** counters and failures
- **Intentionally Omitted** serializing OpenFlow messages

A Verified Controller

Desired Policy:

- ☐ Block SSH traffic
- ☐ Forward other traffic normally
- ☐ Log Web requests

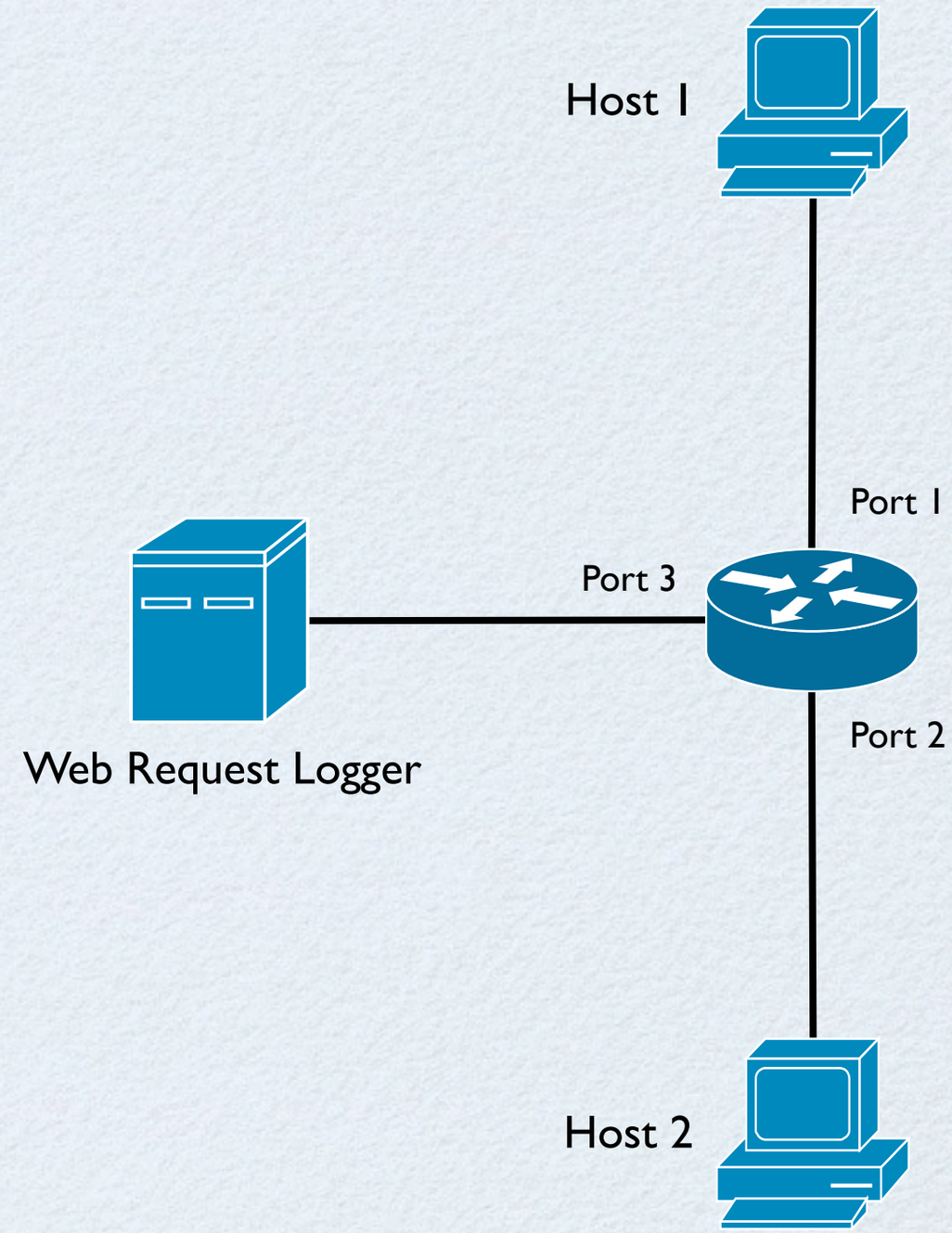


Desired Policy:

- ☐ Block SSH traffic
- ☐ Forward other traffic normally
- ☐ Log Web requests

Desired Flow Table:

`dstPort: 22`	`[]`
`dstIP: H1, dstPort: 80`	`Fwd 1, Fwd 3`
`dstIP: H2, dstPort: 80`	`Fwd 2, Fwd 3`
`dstIP: H1`	`Fwd 1`
`dstIP: H2`	`Fwd 2`



Desired Policy:

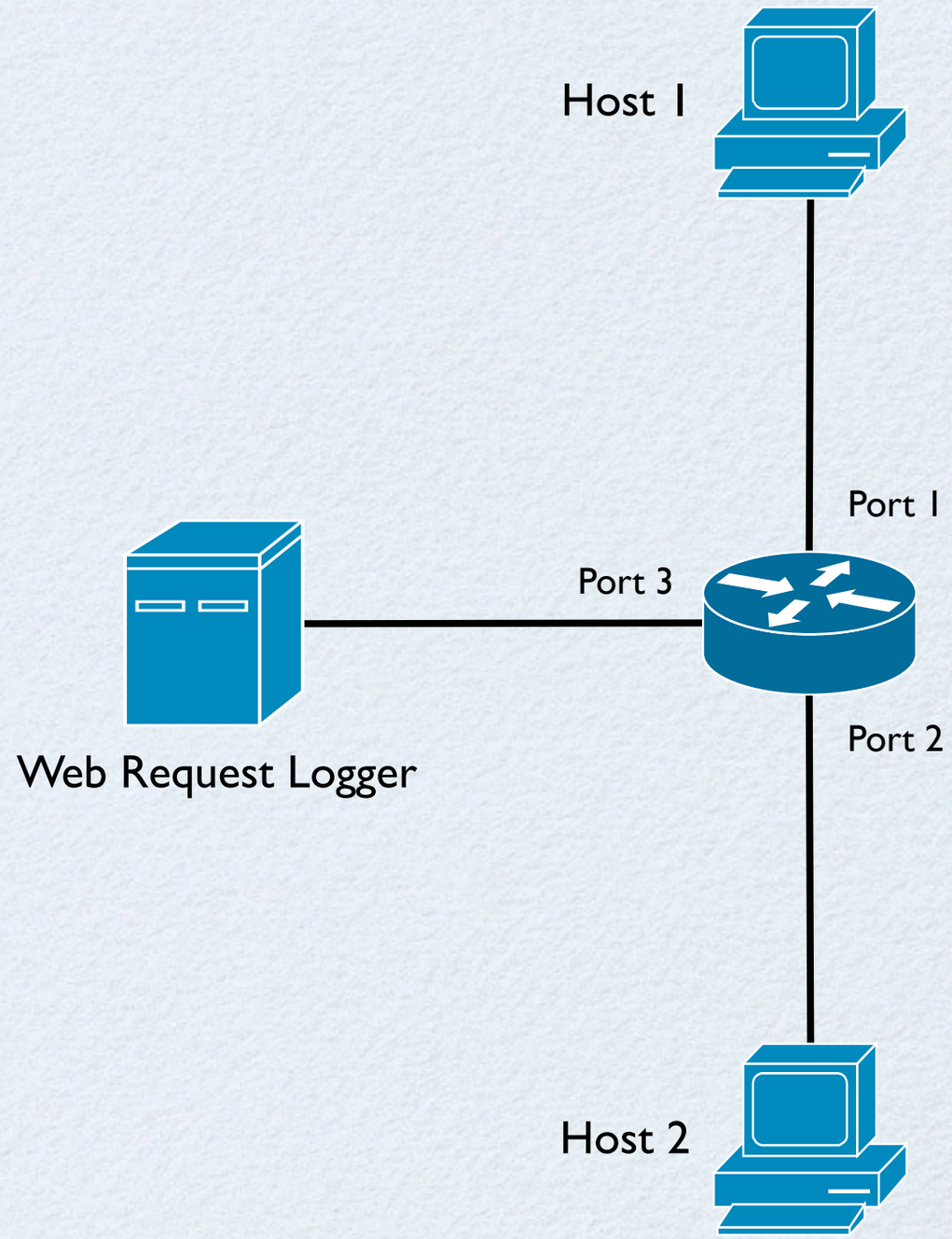
- ☐ Block SSH traffic
- ☐ Forward other traffic normally
- ☐ Log Web requests

Desired Flow Table:

`dstPort: 22`	`[]`
`dstIP: H1, dstPort: 80`	`Fwd 1, Fwd 3`
`dstIP: H2, dstPort: 80`	`Fwd 2, Fwd 3`
`dstIP: H1`	`Fwd 1`
`dstIP: H2`	`Fwd 2`

Actual Controller ...

```
def switch_join(sw):
  sw.flow_mod(10, { eth: 0x800, proto: 6, dstPort: 22 }, [])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H1 }, [Fwd 1])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H2 }, [Fwd 2])

def packet_in(sw, pkt):
  actions = []
  if pkt.dstPort == 22:
    return
  if pkt.dstPort == 80:
    actions = actions + [Fwd 3]
  if pkt.dstIP == H1:
    actions = actions + [Fwd 1]
  if pkt.dstIP == H2:
    actions = actions + [Fwd 2]
  sw.packet_out(pkt, actions)
```



Desired Policy:

- ☐ Block SSH traffic
- ☐ Forward other traffic normally
- ☐ Log Web requests

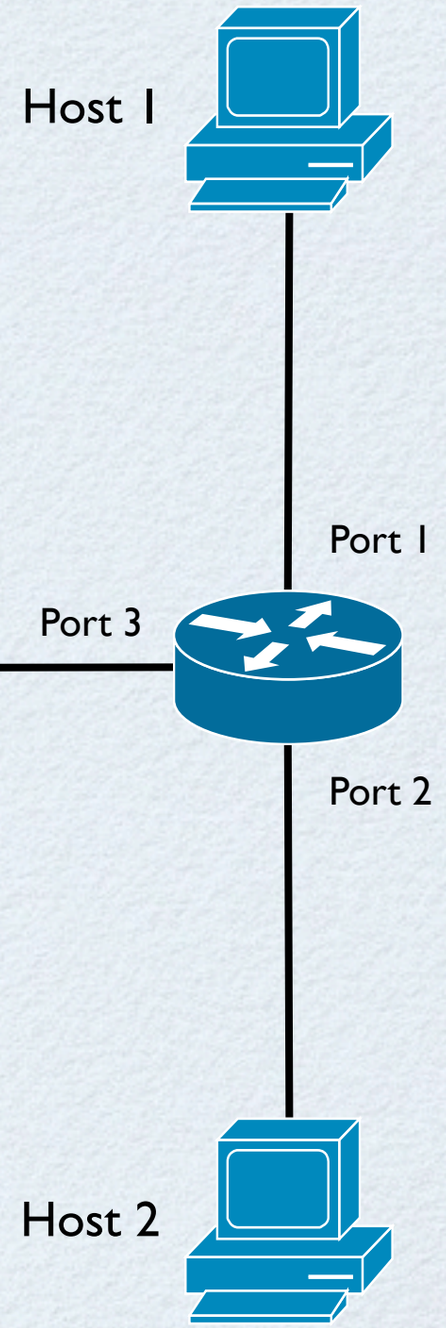
Desired Flow Table:

`dstPort: 22`	`[]`
`dstIP: H1, dstPort: 80`	`Fwd 1, Fwd 3`
`dstIP: H2, dstPort: 80`	`Fwd 2, Fwd 3`
`dstIP: H1`	`Fwd 1`
`dstIP: H2`	`Fwd 2`

Actual Controller ...

```
def switch_join(sw):
  sw.flow_mod(10, { eth: 0x800, proto: 6, dstPort: 22 }, [])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H1 }, [Fwd
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H2 }, [Fwd

def packet_in(sw, pkt):
  actions = []
  if pkt.dstPort == 22:
    return
  if pkt.dstPort == 80:
    actions = actions + [Fwd 3]
  if pkt.dstIP == H1:
    actions = actions + [Fwd 1]
  if pkt.dstIP == H2:
    actions = actions + [Fwd 2]
  sw.packet_out(pkt, actions)
```

- ☐ Barriers
- ☐ packet_in handler
- ☐ Frame type / protocol numbers



Desired Policy:

- ☐ Block SSH traffic
- ☐ Forward other traffic normally
- ☐ Log Web requests

Desired Flow Table:

`dstPort: 22`	`[]`
`dstIP: H1, dstPort: 80`	`Fwd 1, Fwd 3`
`dstIP: H2, dstPort: 80`	`Fwd 2, Fwd 3`
`dstIP: H1`	`Fwd 1`
`dstIP: H2`	`Fwd 2`

Actual Controller …

```
def switch_join(sw):
  sw.flow_mod(10, { eth: 0x800, proto: 6, dstPort: 22 }, [])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H1 }, [Fwd 1])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H2 }, [Fwd 2])

def packet_in(sw, pkt):
  actions = []
  if pkt.dstPort == 22:
    return
  if pkt.dstPort == 80:
    actions = actions + [Fwd 3]
  if pkt.dstIP == H1:
    actions = actions + [Fwd 1]
  if pkt.dstIP == H2:
    actions = actions + [Fwd 2]
  sw.packet_out(pkt, actions)
```

Desired Policy:

- ☐ Block SSH traffic
- ☐ Forward other traffic normally
- ☐ Log Web requests

Desired Flow Table:

dstPort: 22	[]
dstIP: H1, dstPort: 80	Fwd 1, Fwd 3
dstIP: H2, dstPort: 80	Fwd 2, Fwd 3
dstIP: H1	Fwd 1
dstIP: H2	Fwd 2

written by OpenFlow programmer

Actual Controller …

```
def switch_join(sw):
  sw.flow_mod(10, { eth: 0x800, proto: 6, dstPort: 22 }, [])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H1 }, [Fwd 1])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H2 }, [Fwd 2])

def packet_in(sw, pkt):
  actions = []
  if pkt.dstPort == 22:
    return
  if pkt.dstPort == 80:
    actions = actions + [Fwd 3]
  if pkt.dstIP == H1:
    actions = actions + [Fwd 1]
  if pkt.dstIP == H2:
    actions = actions + [Fwd 2]
  sw.packet_out(pkt, actions)
```

Desired Policy:

- ☐ Block SSH traffic
- ☐ Forward other traffic normally
- ☐ Log Web requests

Desired Flow Table:

`dstPort: 22`	`[]`
`dstIP: H1, dstPort: 80`	`Fwd 1, Fwd 3`
`dstIP: H2, dstPort: 80`	`Fwd 2, Fwd 3`
`dstIP: H1`	`Fwd 1`
`dstIP: H2`	`Fwd 2`

written by OpenFlow programmer

inserted by controller

Actual Controller ...

```
def switch_join(sw):
  sw.flow_mod(10, { eth: 0x800, proto: 6, dstPort: 22 }, [])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H1 }, [Fwd 1])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H2 }, [Fwd 2])

def packet_in(sw, pkt):
  actions = []
  if pkt.dstPort == 22:
    return
  if pkt.dstPort == 80:
    actions = actions + [Fwd 3]
  if pkt.dstIP == H1:
    actions = actions + [Fwd 1]
  if pkt.dstIP == H2:
    actions = actions + [Fwd 2]
  sw.packet_out(pkt, actions)
```

Desired Policy:

- ☐ Block SSH traffic
- ☐ Forward other traffic normally
- ☐ Log Web requests

Desired Flow Table:

`dstPort: 22`	`[]`
`dstIP: H1, dstPort: 80`	`Fwd 1, Fwd 3`
`dstIP: H2, dstPort: 80`	`Fwd 2, Fwd 3`
`dstIP: H1`	`Fwd 1`
`dstIP: H2`	`Fwd 2`

written by OpenFlow programmer

inserted by controller

Actual Controller ...

```
def switch_join(sw):
  sw.flow_mod(10, { eth: 0x800, proto: 6, dstPort: 22 }, [])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H1, dstPort: 80 }, [Fwd 1, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(9, { eth: 0x800, proto: 6, dstIP: H2, dstPort: 80 }, [Fwd 2, Fwd 3])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H1 }, [Fwd 1])
  sw.barrier_request()
  sw.flow_mod(8, { eth: 0x800, dstIP: H2 }, [Fwd 2])

def packet_in(sw, pkt):
  actions = []
  if pkt.dstPort == 22:
    return
  if pkt.dstPort == 80:
    actions = actions + [Fwd 3]
  if pkt.dstIP == H1:
    actions = actions + [Fwd 1]
  if pkt.dstIP == H2:
    actions = actions + [Fwd 2]
  sw.packet_out(pkt, actions)
```

formally verified

- ☑ Barriers
- ☑ packet_in handler
- ☑ Frame type / protocol numbers
- ☑ transparent optimizations

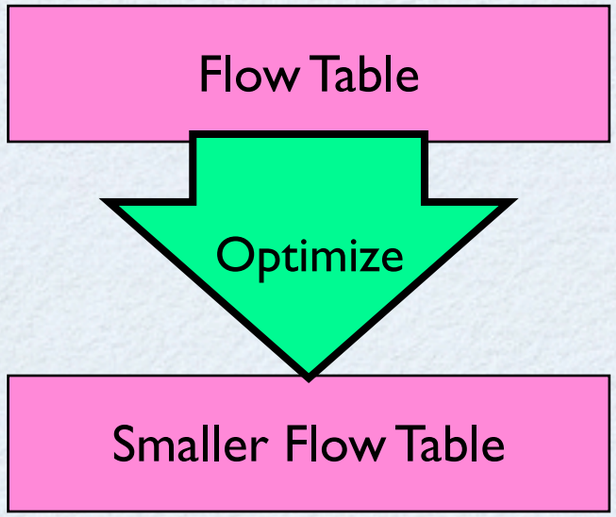
verified code

serialization and physical network

Flow Table

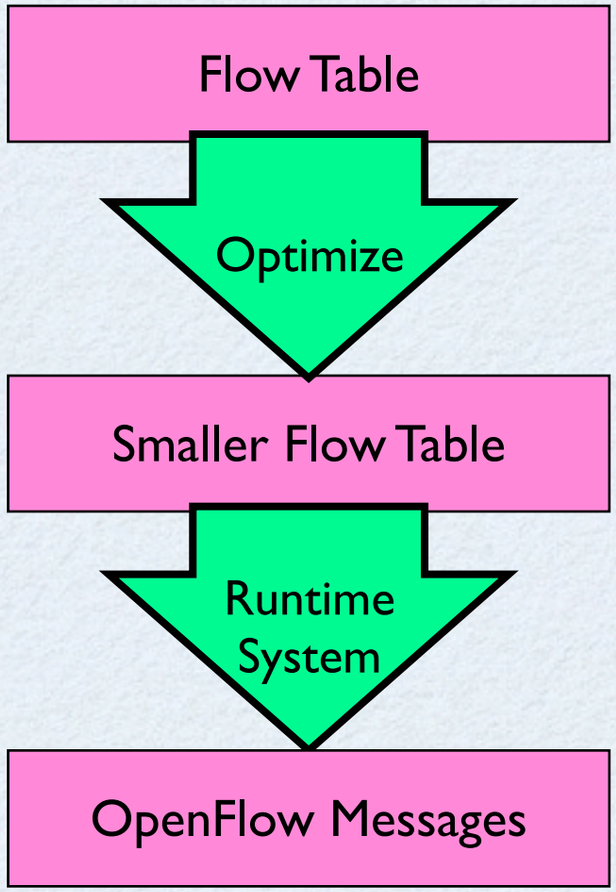
verified code

serialization and physical network



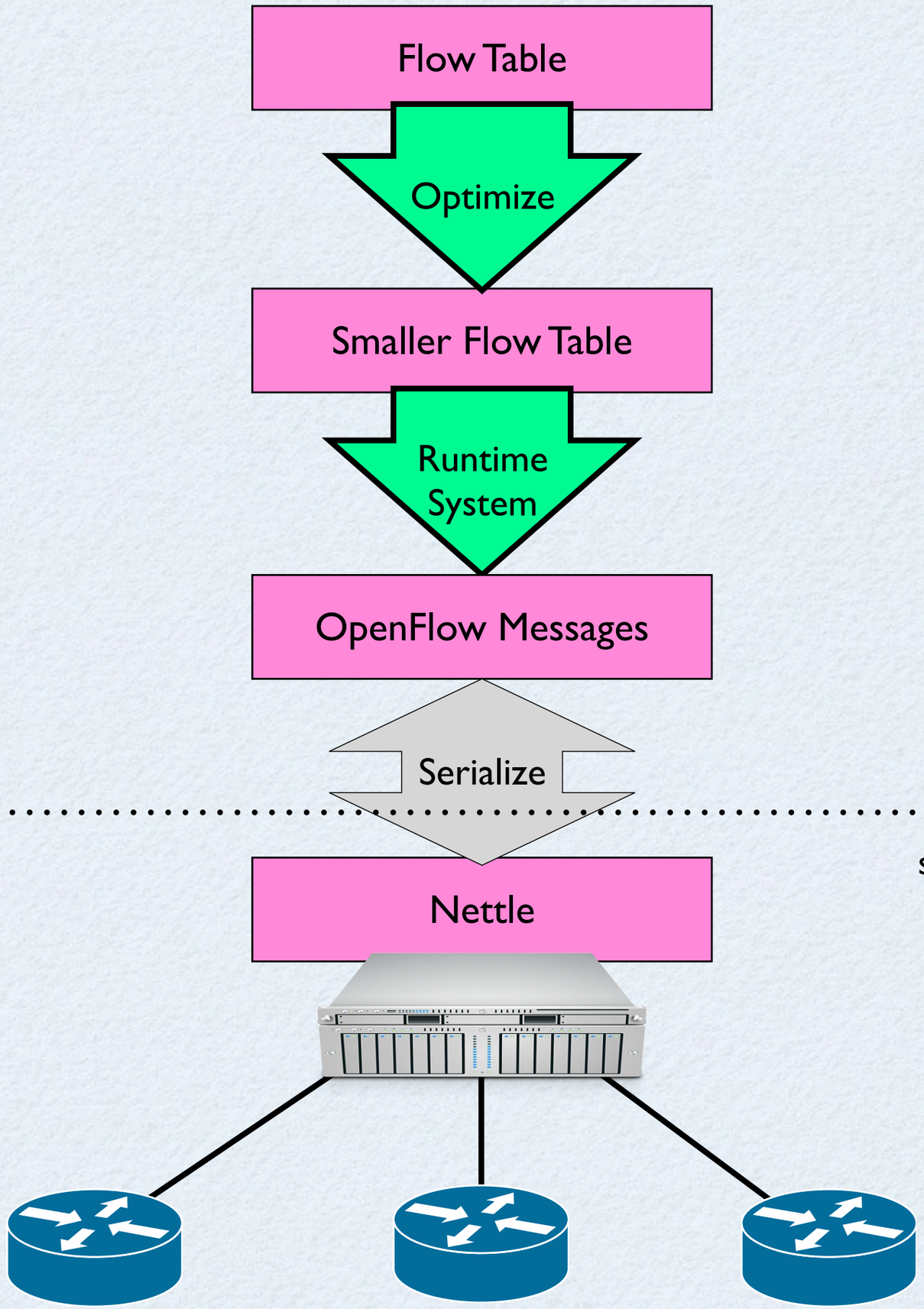
verified code

serialization and physical network



verified code

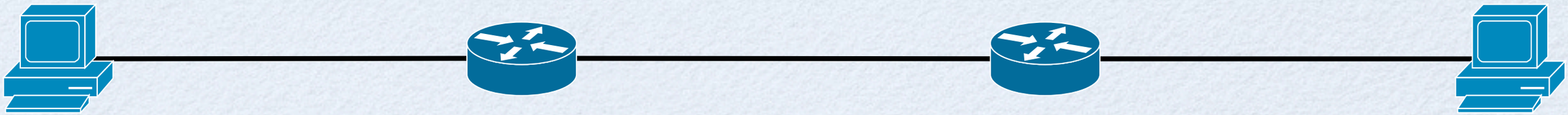
serialization and physical network



verified code

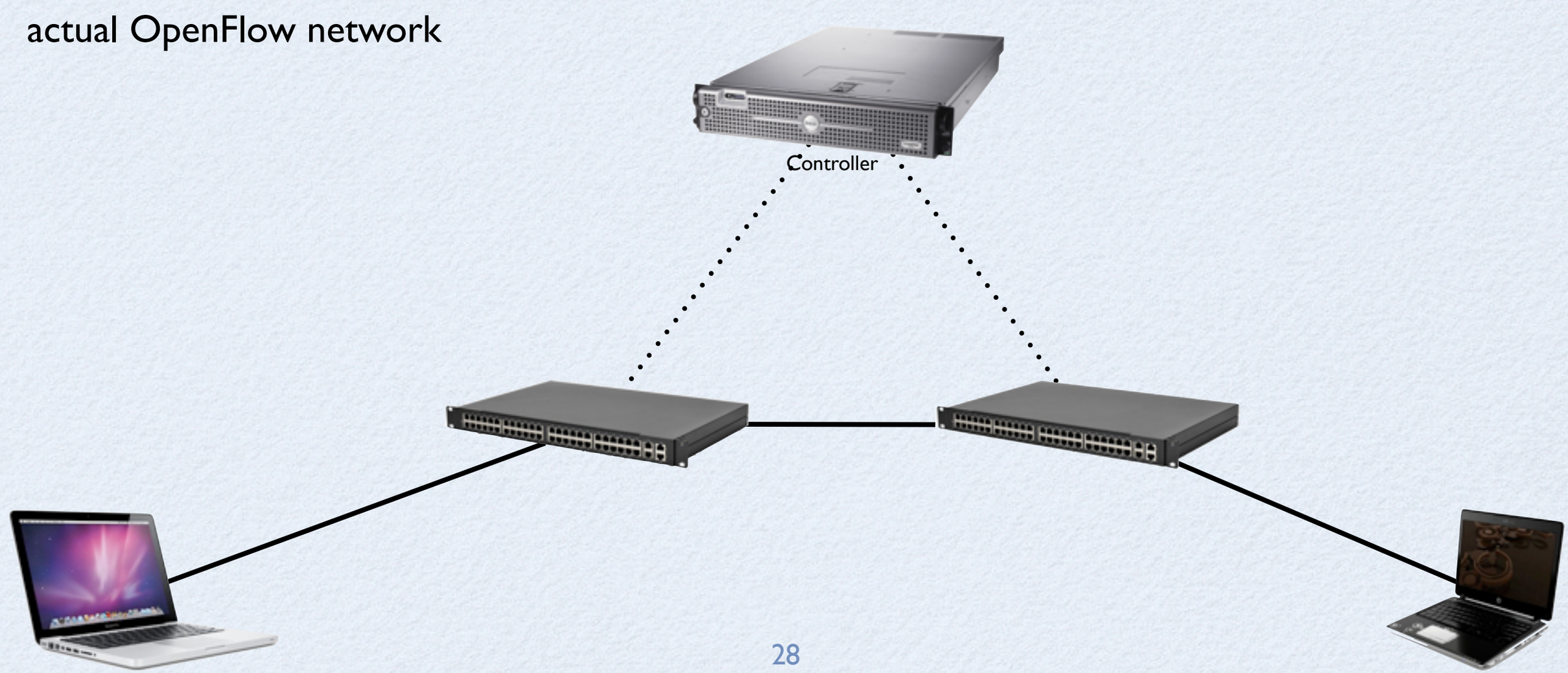
serialization and physical network

Bisimulation

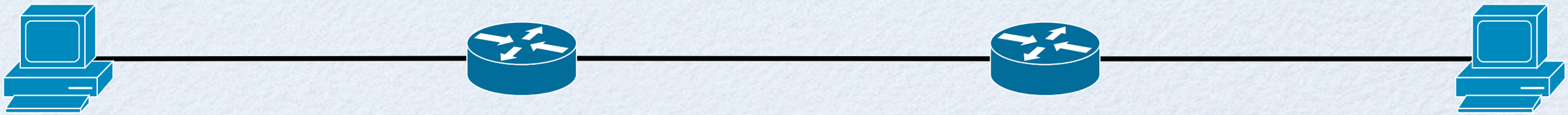


abstract model

actual OpenFlow network

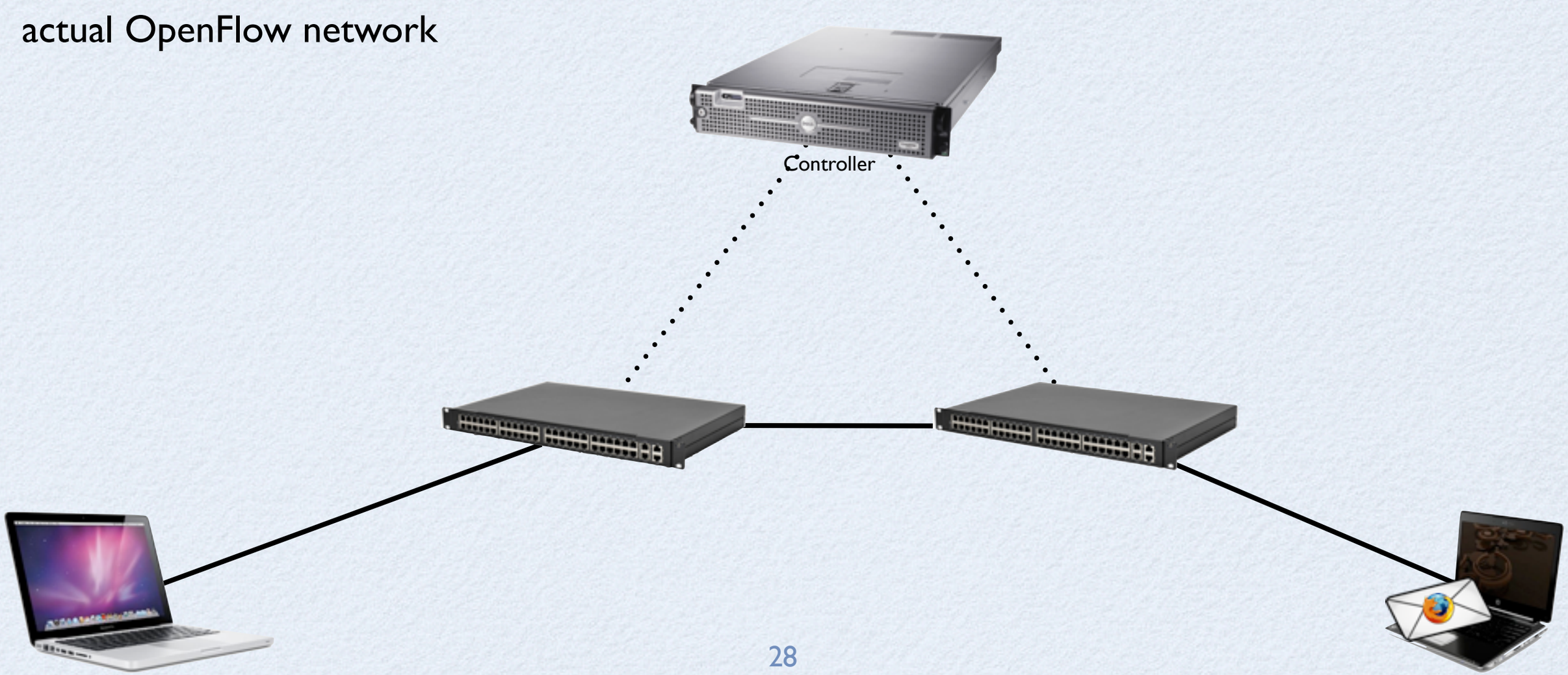


Bisimulation

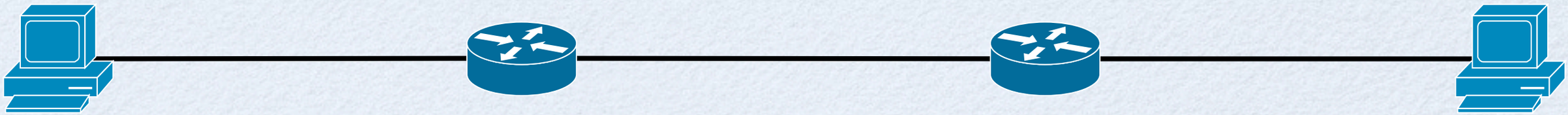


abstract model

actual OpenFlow network

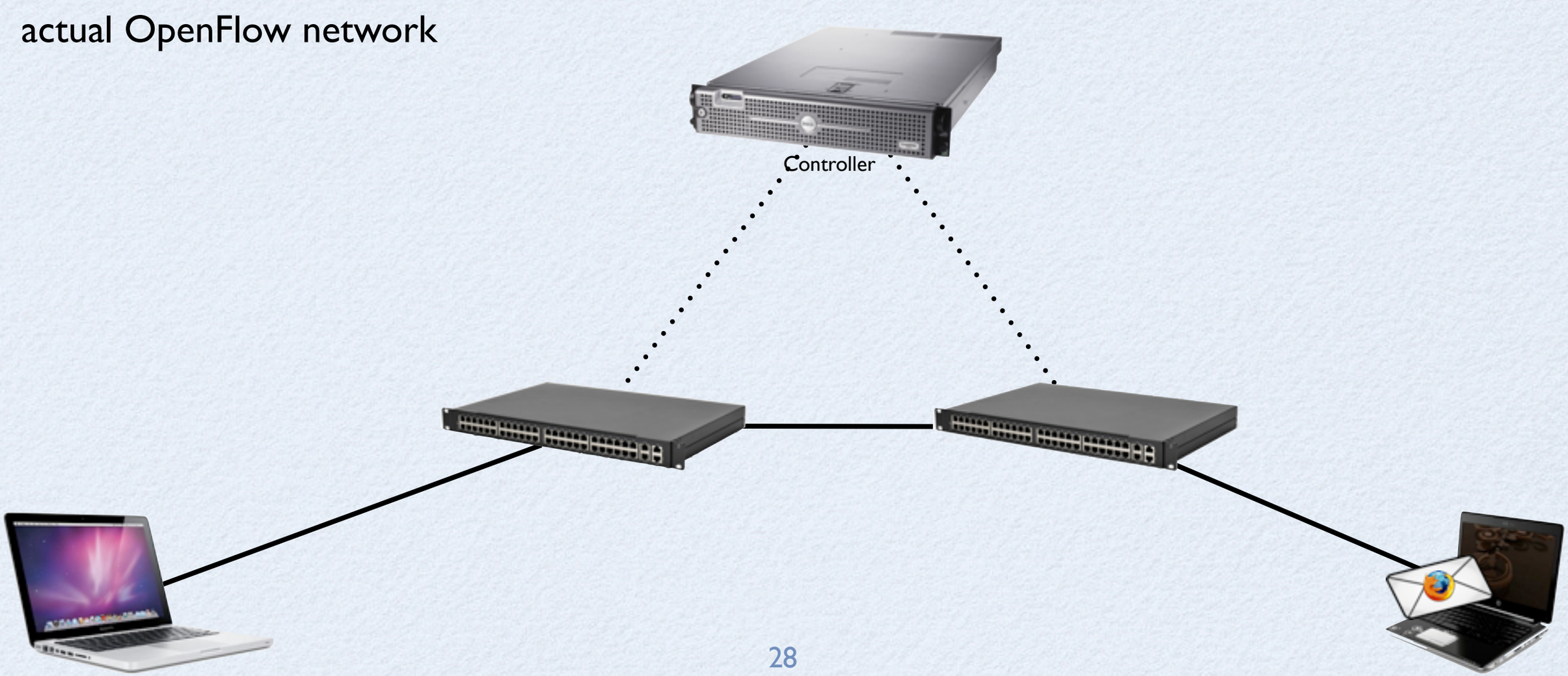


Bisimulation

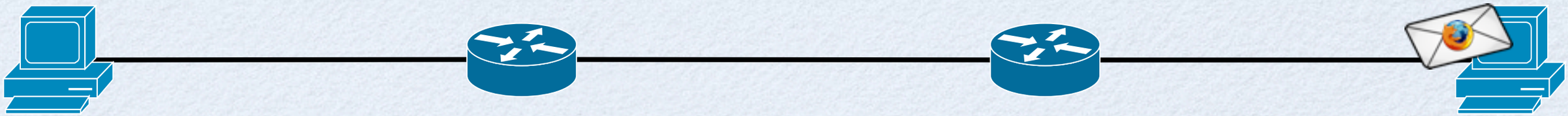


abstract model

actual OpenFlow network

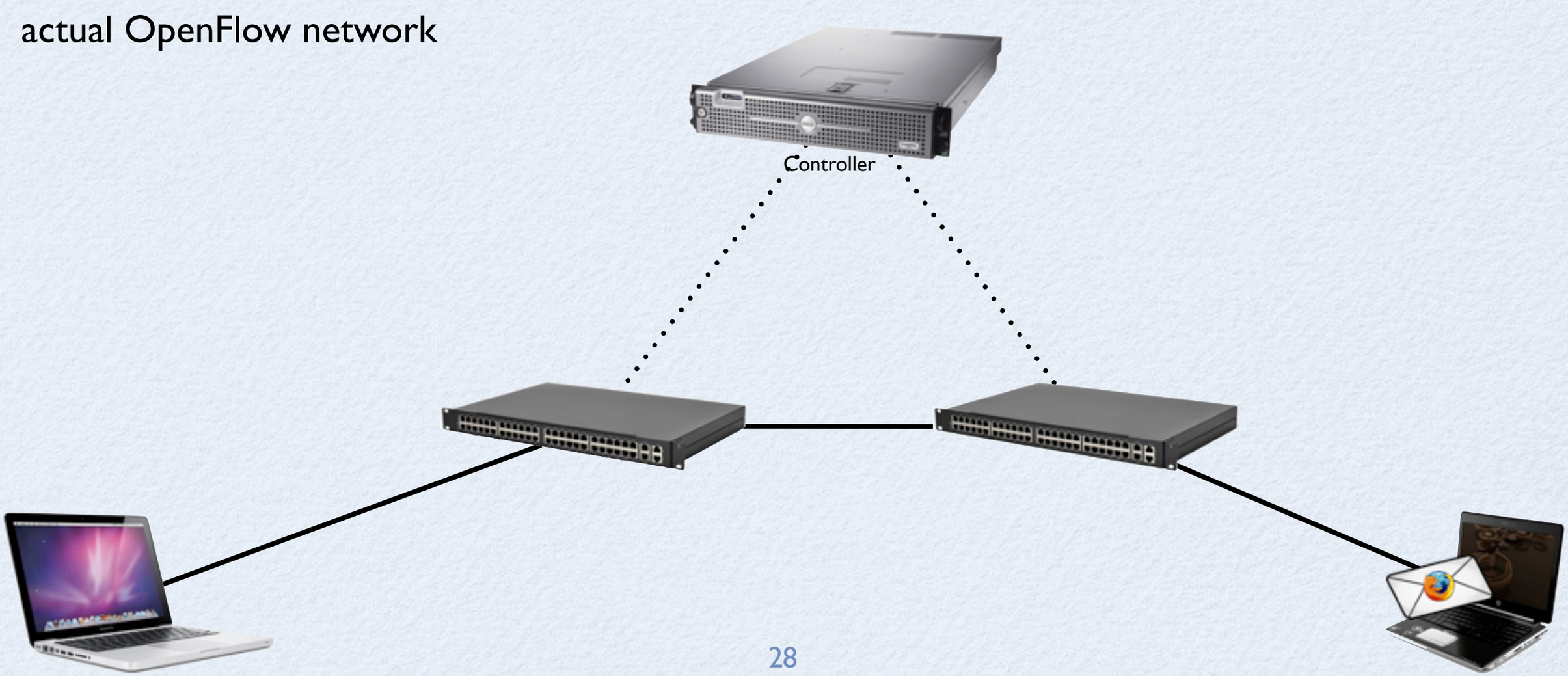


Bisimulation



abstract model

actual OpenFlow network



Performance

theorems are slow

Performance

Throughput (CBench):

Verified Controller	3.5K messages / second
NetCore Controller	3.0K messages / second

theorems are slow

Performance

Throughput (CBench):

Verified Controller	3.5K messages / second
NetCore Controller	3.0K messages / second

theorems are slow

Controller Traffic

ping-all five times on
fattree with
six switches

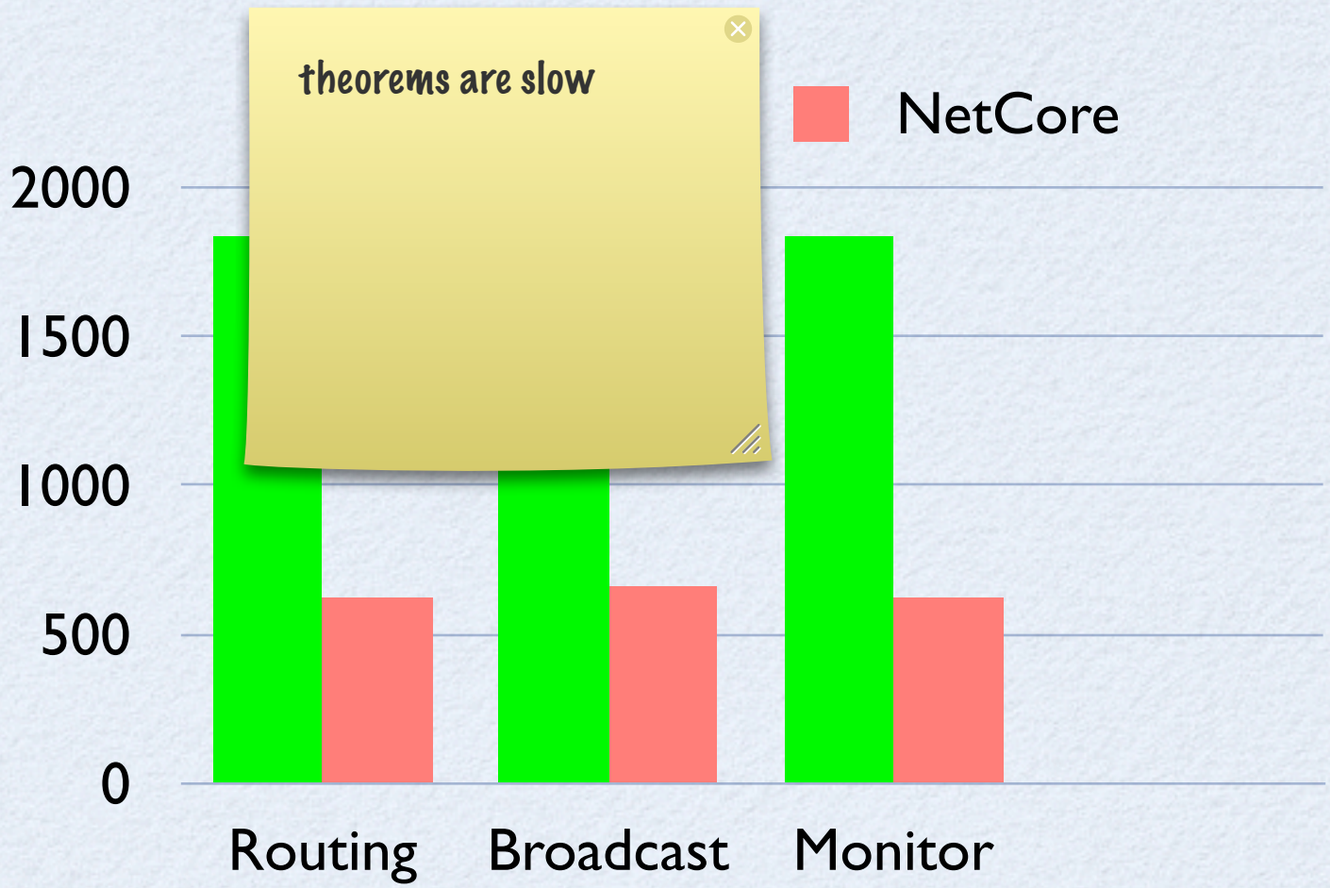
Performance

Throughput (CBench):

Verified Controller	3.5K messages / second
NetCore Controller	3.0K messages / second

Controller Traffic

ping-all five times on fattree with six switches



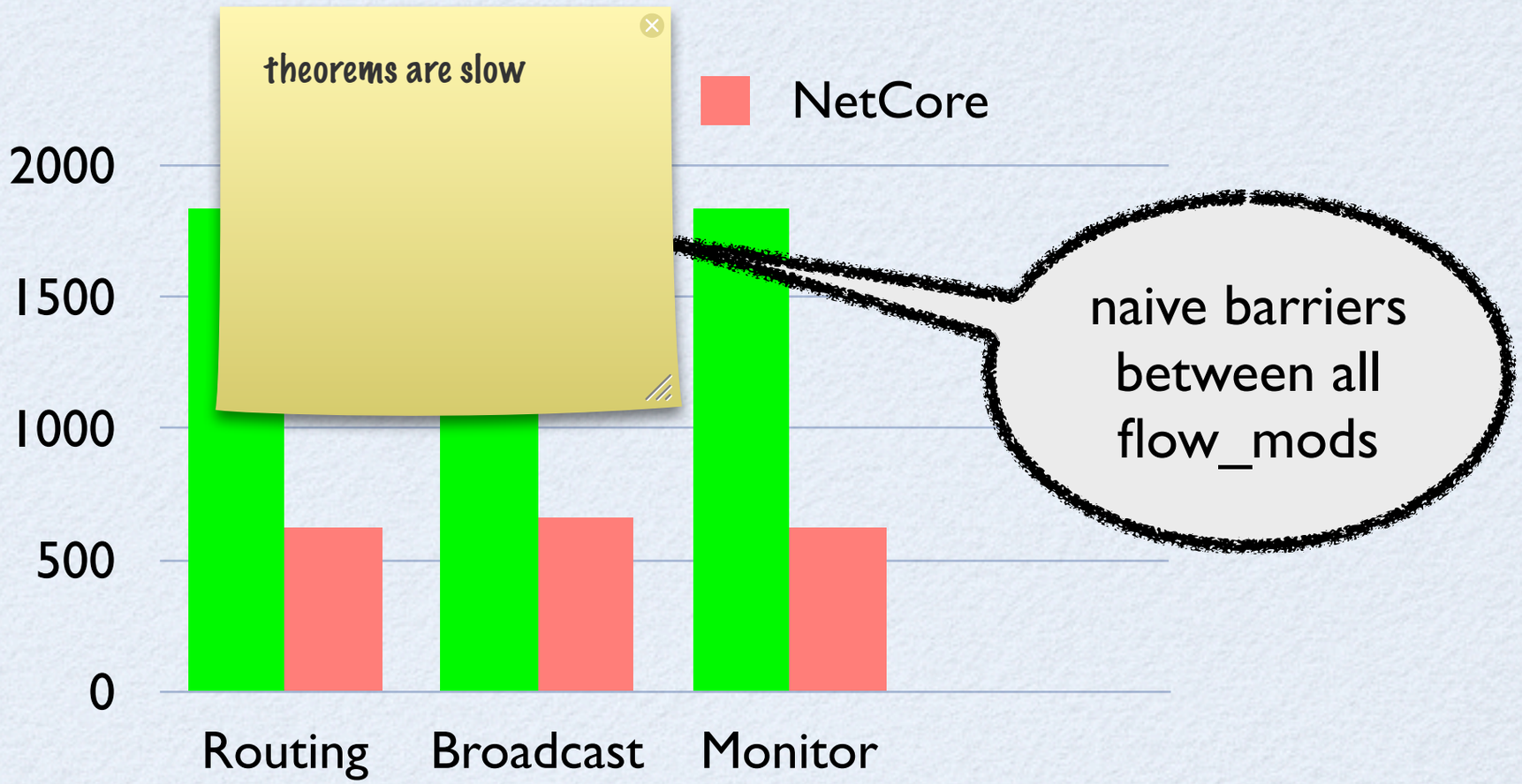
Performance

Throughput (CBench):

Verified Controller	3.5K messages / second
NetCore Controller	3.0K messages / second

Controller Traffic

ping-all five times on fattree with six switches



- **Block SSH traffic**
  dstPort == 22 ⟹ Drop
- **Forward other traffic normally**
  dstIP == H1 ⟹ Fwd 1
  dstIP == H2 ⟹ Fwd 2
- **Log Web requests**
  dstPort == 80 ⟹ Fwd 3

Priority	Pattern	Action
10	dstPort: 22	[]
9	dstIP: H1, dstPort: 80	Fwd 1, Fwd 3
9	dstIP: H2, dstPort: 80	Fwd 2, Fwd 3
8	dstIP: H1	Fwd 1
8	dstIP: H2	Fwd 2

- **Block SSH traffic**
  dstPort == 22 ⟹ Drop
- **Forward other traffic normally**
  dstIP == H1 ⟹ Fwd 1
  dstIP == H2 ⟹ Fwd 2
- **Log Web requests**
  dstPort == 80 ⟹ Fwd 3

how?

Priority	Pattern	Action
10	dstPort: 22	[]
9	dstIP: H1, dstPort: 80	Fwd 1, Fwd 3
9	dstIP: H2, dstPort: 80	Fwd 2, Fwd 3
8	dstIP: H1	Fwd 1
8	dstIP: H2	Fwd 2

- **Block SSH traffic**
  dstPort == 22 ⟹ Drop
- **Forward other traffic normally**
  dstIP == H1 ⟹ Fwd 1
  dstIP == H2 ⟹ Fwd 2
- **Log Web requests**
  dstPort == 80 ⟹ Fwd 3

NetCore. Monsanto et al., POPL 2012.

```
restrictTo (tcpPort != 22)
```

how?

Priority	Pattern	Action
10	dstPort: 22	[]
9	dstIP: H1, dstPort: 80	Fwd 1, Fwd 3
9	dstIP: H2, dstPort: 80	Fwd 2, Fwd 3
8	dstIP: H1	Fwd 1
8	dstIP: H2	Fwd 2

- **Block SSH traffic**
  dstPort == 22 ⟹ Drop
- **Forward other traffic normally**
  dstIP == H1 ⟹ Fwd 1
  dstIP == H2 ⟹ Fwd 2
- **Log Web requests**
  dstPort == 80 ⟹ Fwd 3

NetCore. Monsanto et al., POPL 2012.

```
restrictTo (tcpPort != 22)
  dstIP    == H1 ⟹ Fwd 1 ∪
```

how?

Priority	Pattern	Action
10	dstPort: 22	[]
9	dstIP: H1, dstPort: 80	Fwd 1, Fwd 3
9	dstIP: H2, dstPort: 80	Fwd 2, Fwd 3
8	dstIP: H1	Fwd 1
8	dstIP: H2	Fwd 2

- **Block SSH traffic**
  dstPort == 22 ⟹ Drop
- **Forward other traffic normally**
  dstIP == H1 ⟹ Fwd 1
  dstIP == H2 ⟹ Fwd 2
- **Log Web requests**
  dstPort == 80 ⟹ Fwd 3

NetCore. Monsanto et al., POPL 2012.

```
restrictTo (tcpPort != 22)
  dstIP   == H1 ⟹ Fwd 1 U
  dstIP   == H2 ⟹ Fwd 2 U
```

how?

Priority	Pattern	Action
10	dstPort: 22	[]
9	dstIP: H1, dstPort: 80	Fwd 1, Fwd 3
9	dstIP: H2, dstPort: 80	Fwd 2, Fwd 3
8	dstIP: H1	Fwd 1
8	dstIP: H2	Fwd 2

- **Block SSH traffic**
  dstPort == 22 ⟹ Drop
- **Forward other traffic normally**
  dstIP == H1 ⟹ Fwd 1
  dstIP == H2 ⟹ Fwd 2
- **Log Web requests**
  dstPort == 80 ⟹ Fwd 3

NetCore. Monsanto et al., POPL 2012.

```
restrictTo (tcpPort != 22)
  dstIP   == H1 ⟹ Fwd 1 U
  dstIP   == H2 ⟹ Fwd 2 U
  tcpPort == 80 ⟹ Fwd 3
```

how?

Priority	Pattern	Action
10	dstPort: 22	[]
9	dstIP: H1, dstPort: 80	Fwd 1, Fwd 3
9	dstIP: H2, dstPort: 80	Fwd 2, Fwd 3
8	dstIP: H1	Fwd 1
8	dstIP: H2	Fwd 2

- **Block SSH traffic**
  dstPort == 22 ⟹ Drop
- **Forward other traffic normally**
  dstIP == H1 ⟹ Fwd 1
  dstIP == H2 ⟹ Fwd 2
- **Log Web requests**
  dstPort == 80 ⟹ Fwd 3

NetCore. Monsanto et al., POPL 2012.

```
restrictTo (tcpPort != 22)
  tcpPort == 80 ⟹ Fwd 3 U
  dstIP   == H1 ⟹ Fwd 1 U
  dstIP   == H2 ⟹ Fwd 2 U
```

how?

Priority	Pattern	Action
10	dstPort: 22	[]
9	dstIP: H1, dstPort: 80	Fwd 1, Fwd 3
9	dstIP: H2, dstPort: 80	Fwd 2, Fwd 3
8	dstIP: H1	Fwd 1
8	dstIP: H2	Fwd 2

- **Block SSH traffic**
  dstPort == 22 ⟹ Drop
- **Forward other traffic normally**
  dstIP == H1 ⟹ Fwd 1
  dstIP == H2 ⟹ Fwd 2
- **Log Web requests**
  dstPort == 80 ⟹ Fwd 3

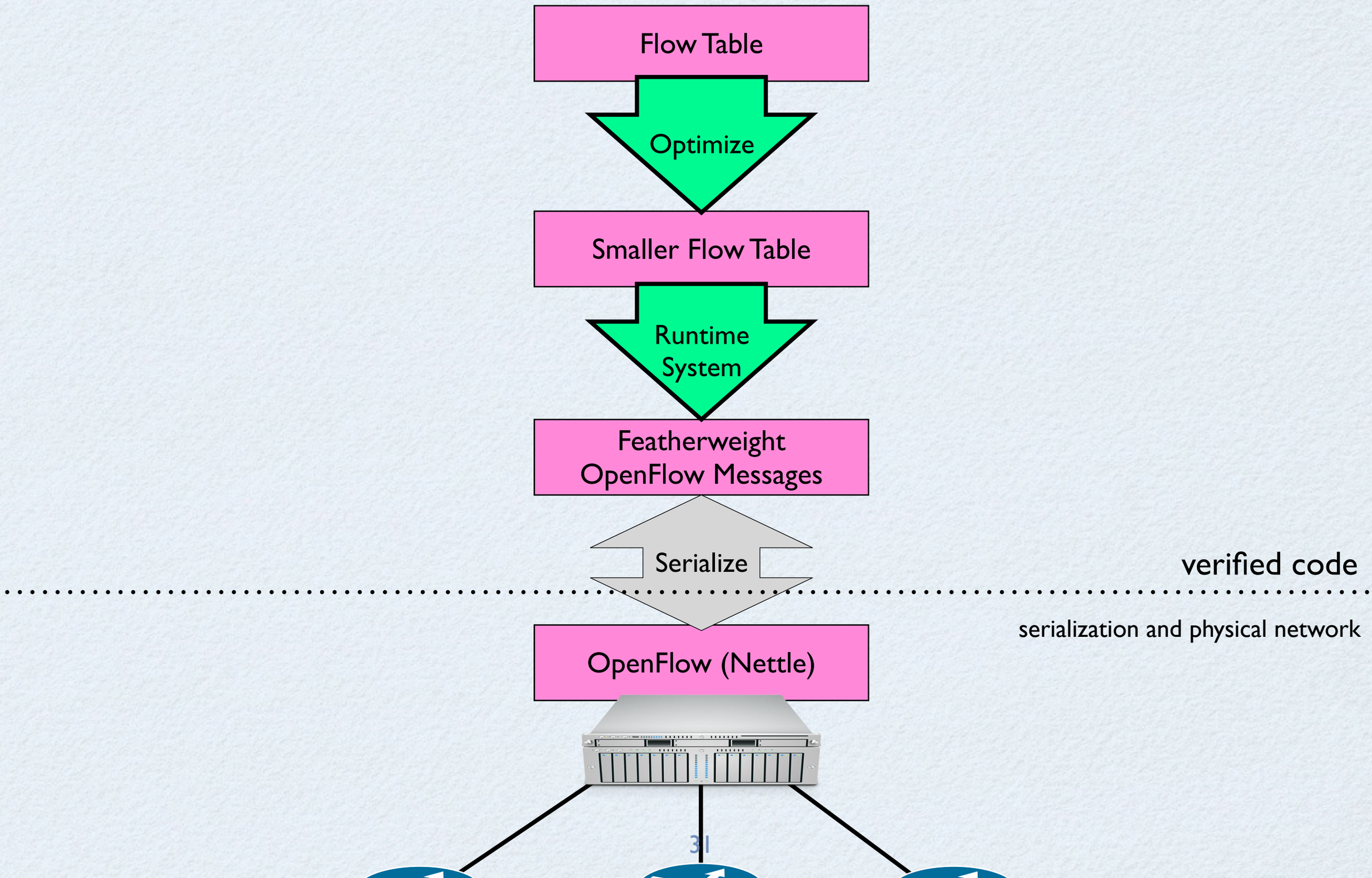
how?

NetCore. Monsanto et al., POPL 2012.

```
restrictTo (tcpPort != 22)
  tcpPort == 80 ⟹ Fwd 3 U
  dstIP   == H1 ⟹ Fwd 1 U
  dstIP   == H2 ⟹ Fwd 2 U
```

new, verified compiler

Priority	Pattern	Action
10	dstPort: 22	[]
9	dstIP: H1, dstPort: 80	Fwd 1, Fwd 3
9	dstIP: H2, dstPort: 80	Fwd 2, Fwd 3
8	dstIP: H1	Fwd 1
8	dstIP: H2	Fwd 2


Flow Table
Optimize
Smaller Flow Table
Runtime System
Featherweight OpenFlow Messages
Serialize
verified code
serialization and physical network
OpenFlow (Nettle)

NetCore

Compiler

Flow Table

Optimize

Smaller Flow Table

Runtime System

Featherweight OpenFlow Messages

Serialize

verified code

serialization and physical network

OpenFlow (Nettle)

Conclusion

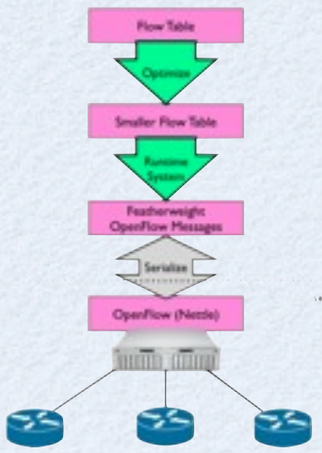
Conclusion

Featherweight OpenFlow

A foundational model for verifying SDN

Conclusion

Featherweight OpenFlow
A foundational model for verifying SDN

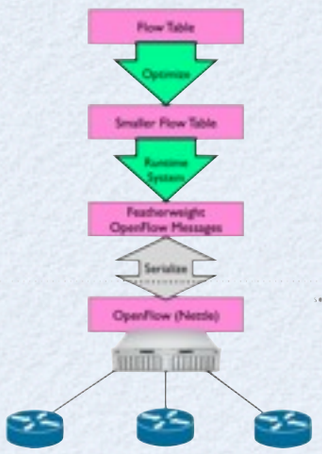


Verified Controller
inserts barriers, handles packet_in messages, and optimizes flow tables

Conclusion

Featherweight OpenFlow

A foundational model for verifying SDN



Verified Controller

inserts barriers, handles packet_in messages, and optimizes flow tables

```
restrictTo (tcpPort != 22)
 dstIP   == H1 ⟹ Fwd 1 ||
 dstIP   == H2 ⟹ Fwd 2 ||
 tcpPort == 80 ⟹ Fwd 3
```

Verified NetCore Compiler

compiles to flow tables, uses the verified controller

A Grand Collaboration:

Languages + Networking

Frenetic Cornell

Shrutarshi Basu (PhD)
Nate Foster (Faculty)
Arjun Guha (Postdoc)
Stephen Gutz (Undergrad)
Mark Reitblatt (PhD)
Robert Soulé (Postdoc)
Alec Story (Undergrad)

Frenetic Princeton

Chris Monsanto (PhD)
Joshua Reich (Postdoc)
Jen Rexford (Faculty)
Cole Schlesinger (PhD)
Dave Walker (Faculty)
Naga Praveen Katta (PhD)

frenetic >>

http://frenetic-lang.org