SDNRacer

Concurrency Analysis for SDNs

Ahmed El-Hassany
Jeremie Miserez
Pavol Bielik
Laurent Vanbever
Martin Vechev

ETH zürich

http://sdnracer.ethz.ch

SDNRacer

Finds violations in SDN controllers:

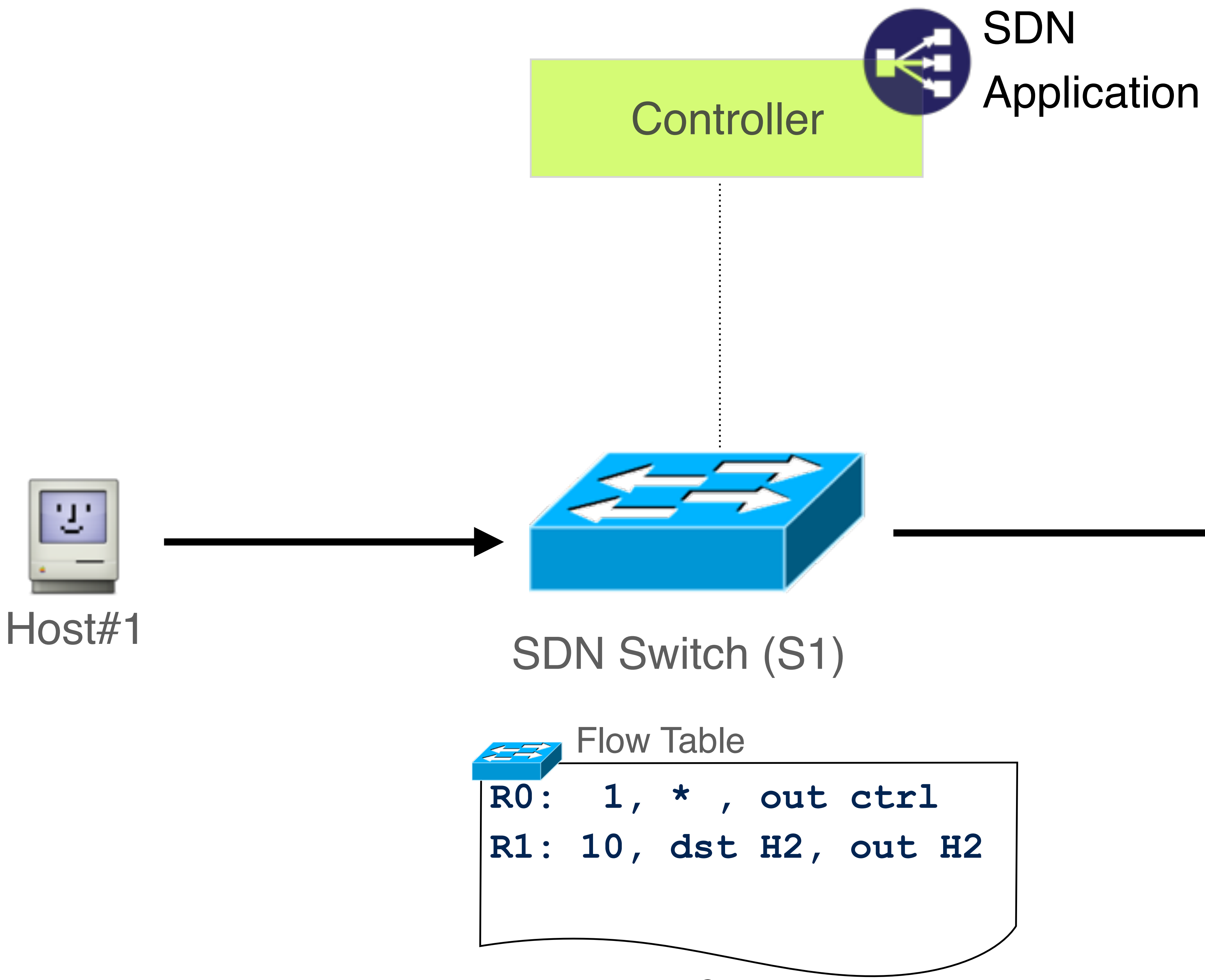
Race Freedom

Update Isolation

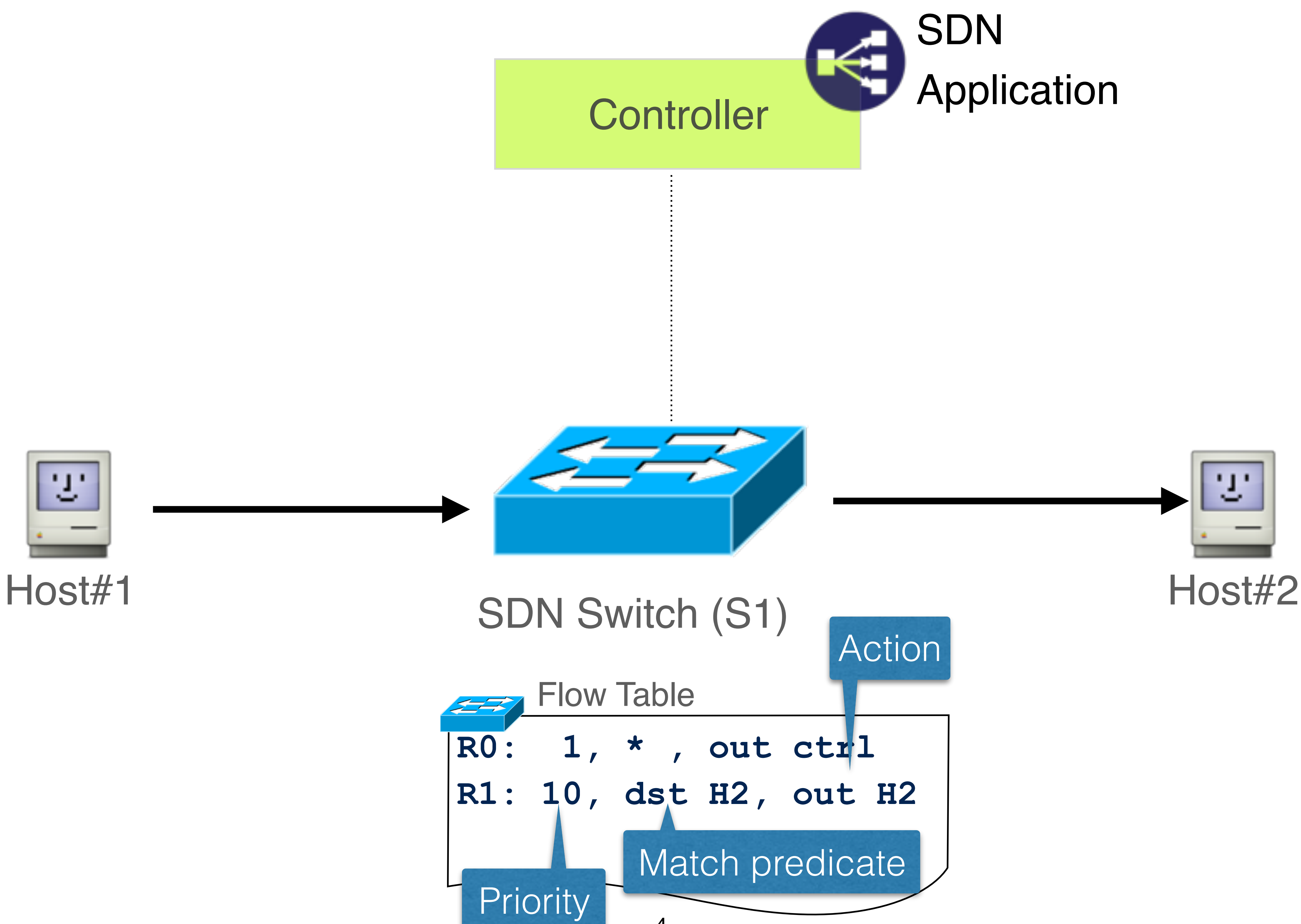
Packet Coherence

Violations of these properties can cause serious bugs in the network.

SDN Overview

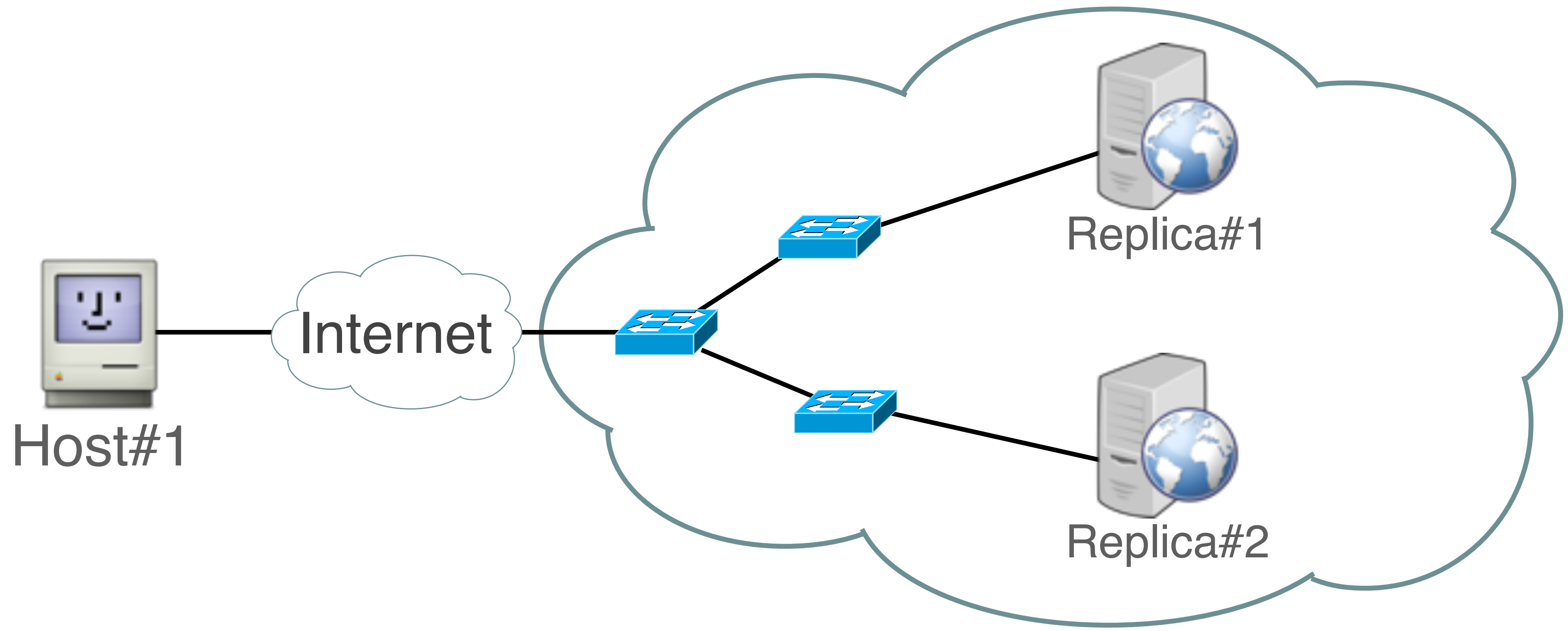


SDN Overview

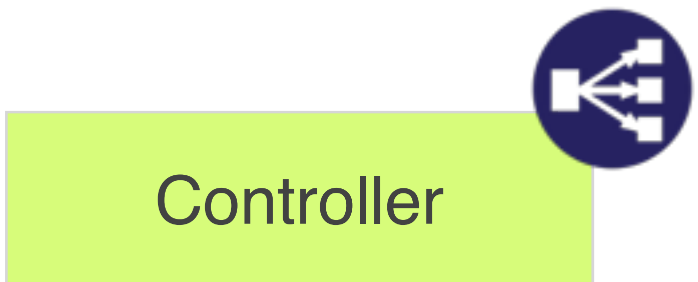


Example SDN App: Load-Balancer

Load Balancer Application



Load Balancer Application




```
if dst == server:
 rep = rep[idx] idx = (idx+1)%2
 install_path(src, rep)
 install_path(rep, src)
 packet_out(pkt,in sw)
```

Load Balancer Application

Controller

Round-Robin Server Selection

```
if dst == server:
 rep = rep[idx] idx = (idx+1)%2
 install_path(src, rep)
 install_path(rep, src)
 packet_out(pkt,in sw)
```

Load Balancer Application

Controller

```
if dst == server:
 rep = rep[idx] idx = (idx+1)%2
 install_path(src, rep)
 install_path(rep, src)
 packet_out(pkt,in sw)
```

1. Find the shortest Path.
2. Write a flow entry on each switch on the path.

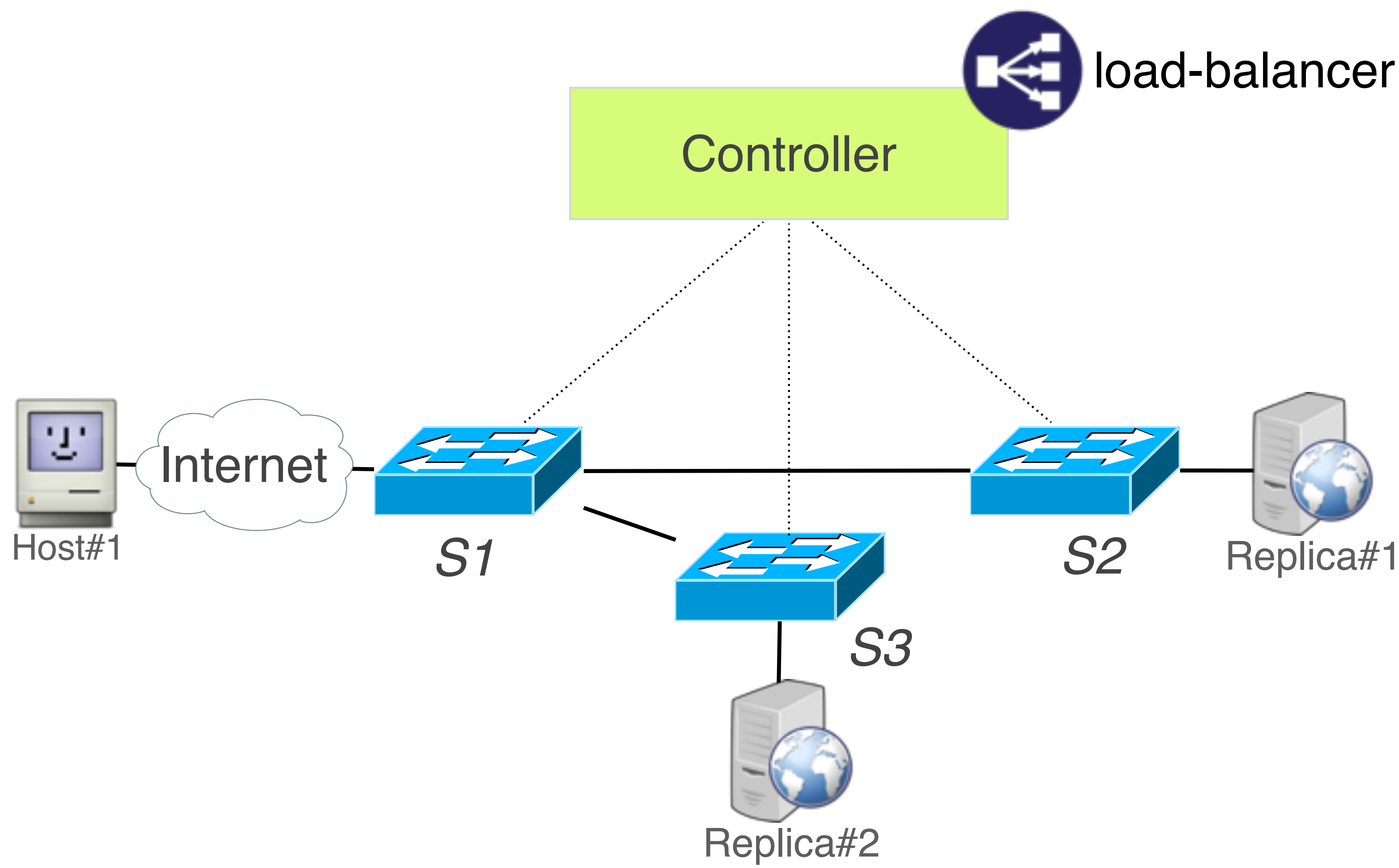
Load Balancer Application

Controller

```
if dst == server:
 rep = rep[idx] idx = (idx+1)%2
 install_path(src, rep)
 install_path(rep, src)
 packet_out(pkt,in sw)
```

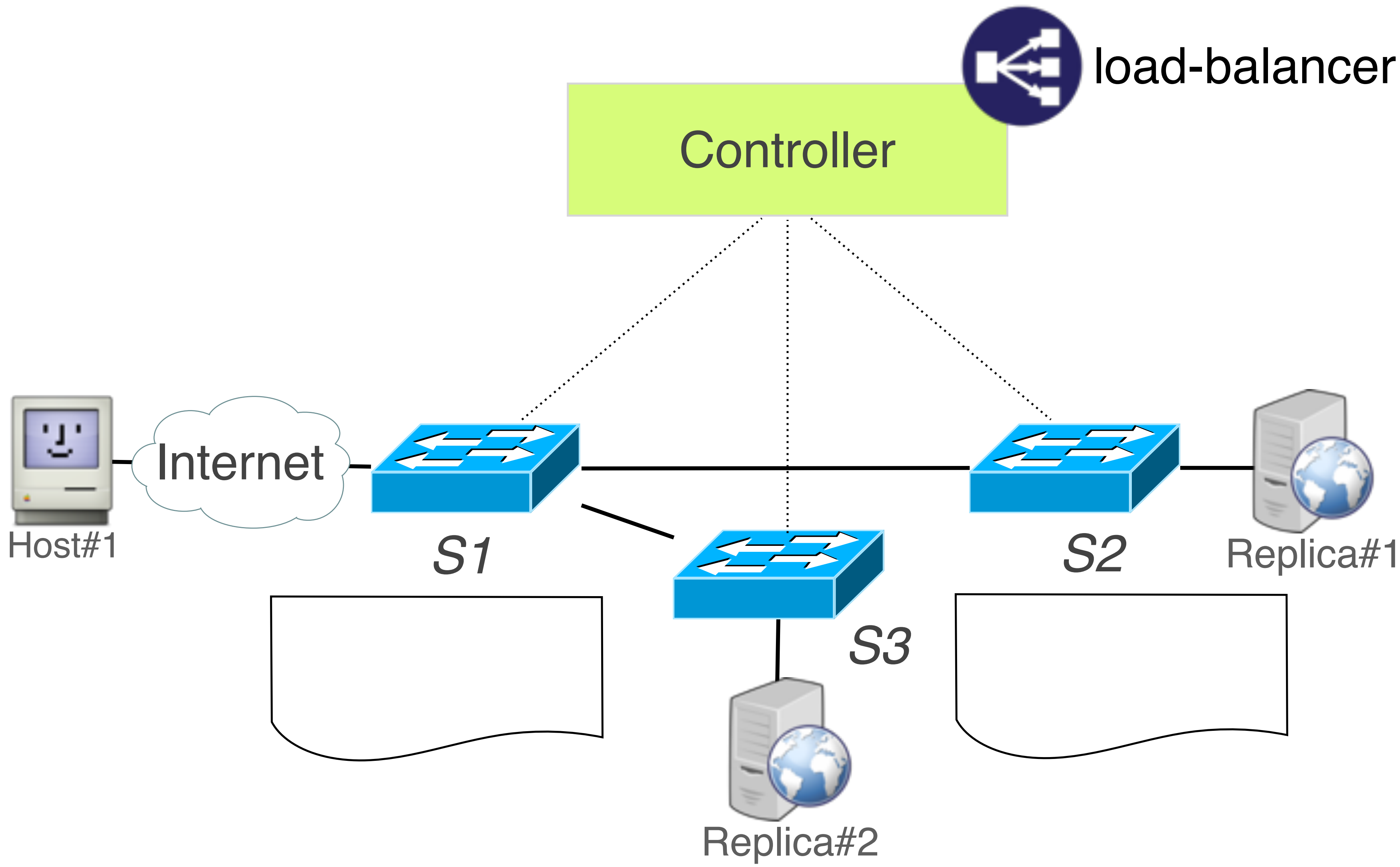
Send the packet back to the dataplane.

Load Balancer Application



Load Balancer Application

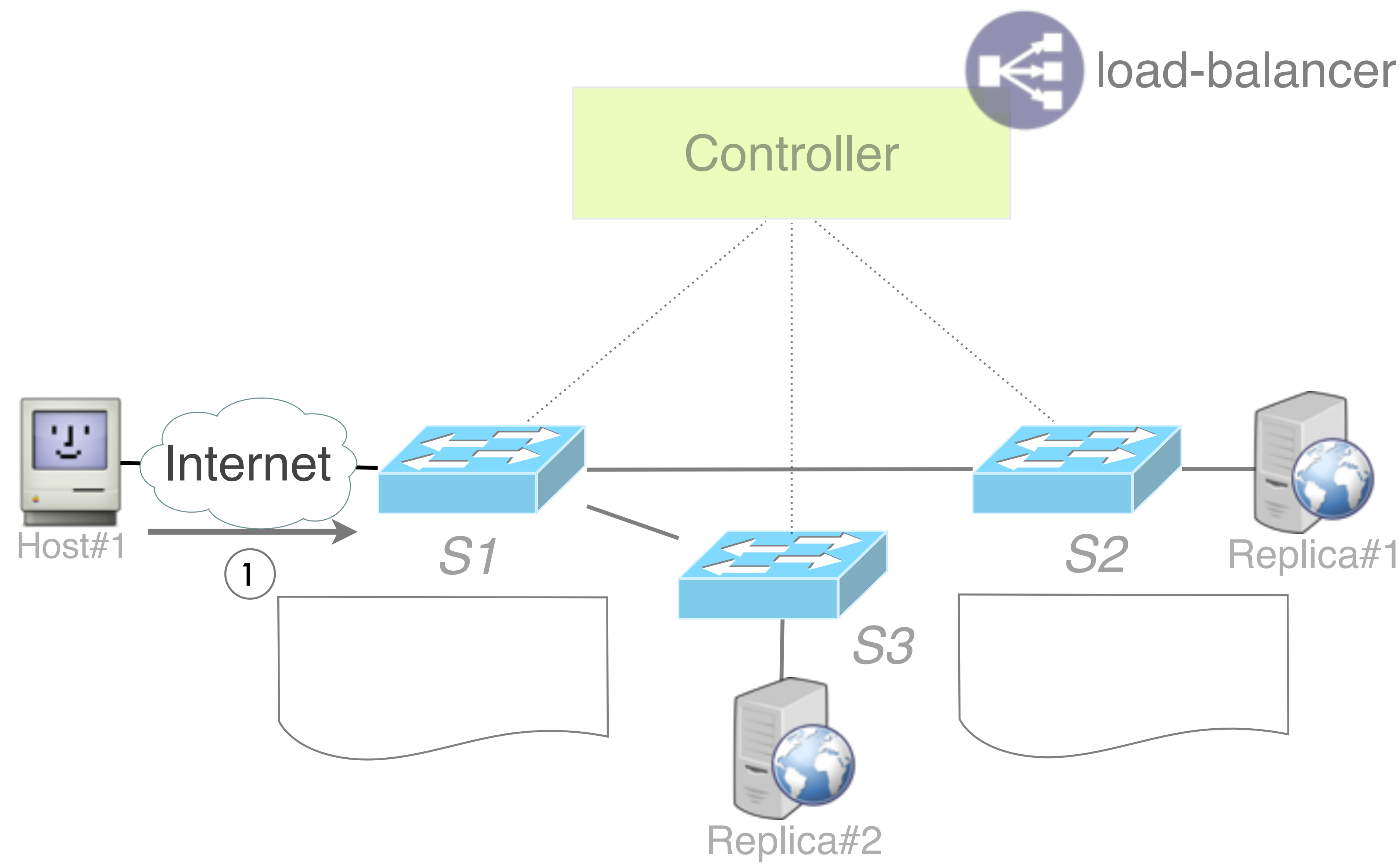
Recorded Event Trace:



Load Balancer Application

Recorded Event Trace:

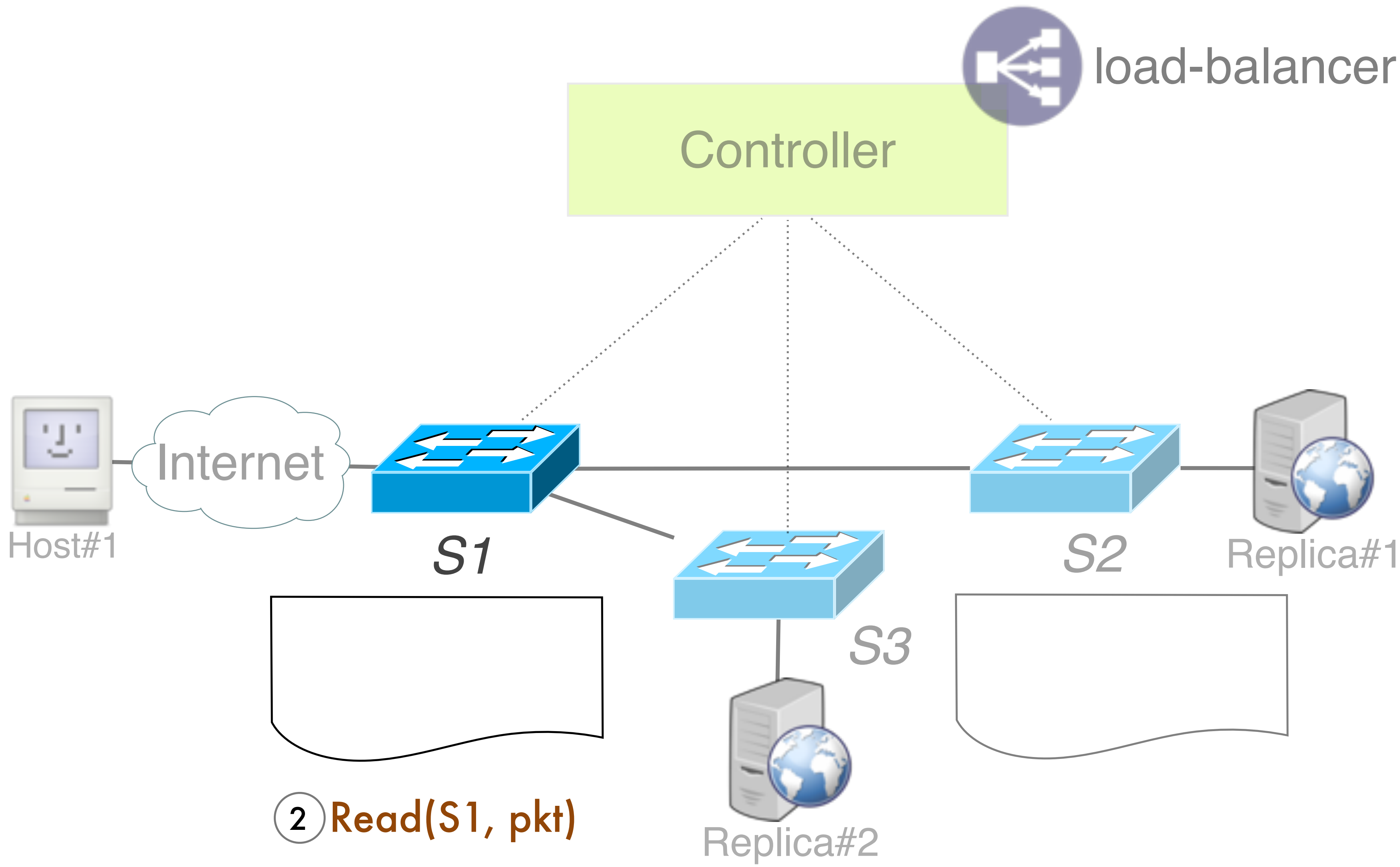
1 Host Send (pkt, dst=srv)



Load Balancer Application

Recorded Event Trace:

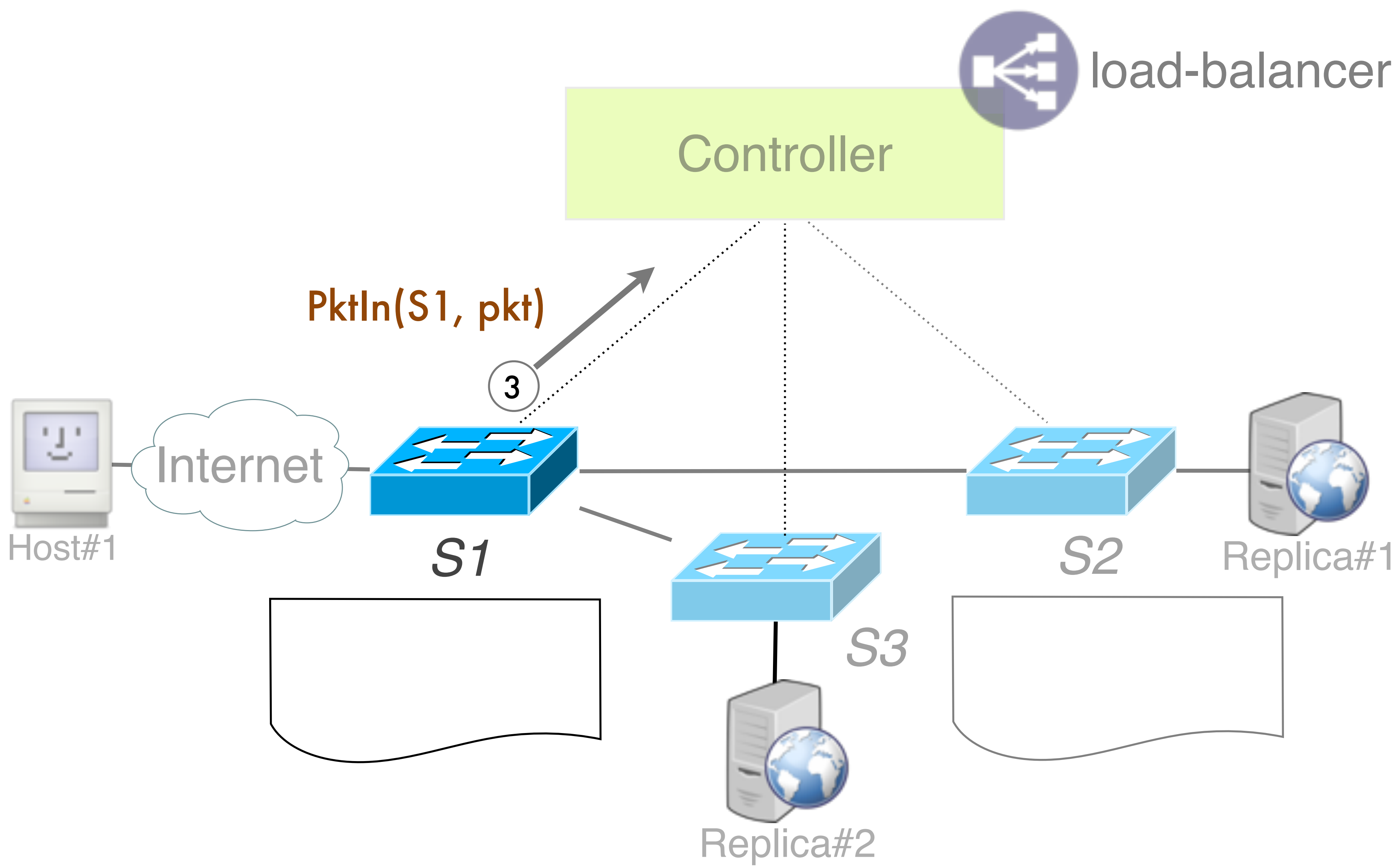
1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)



Load Balancer Application

Recorded Event Trace:

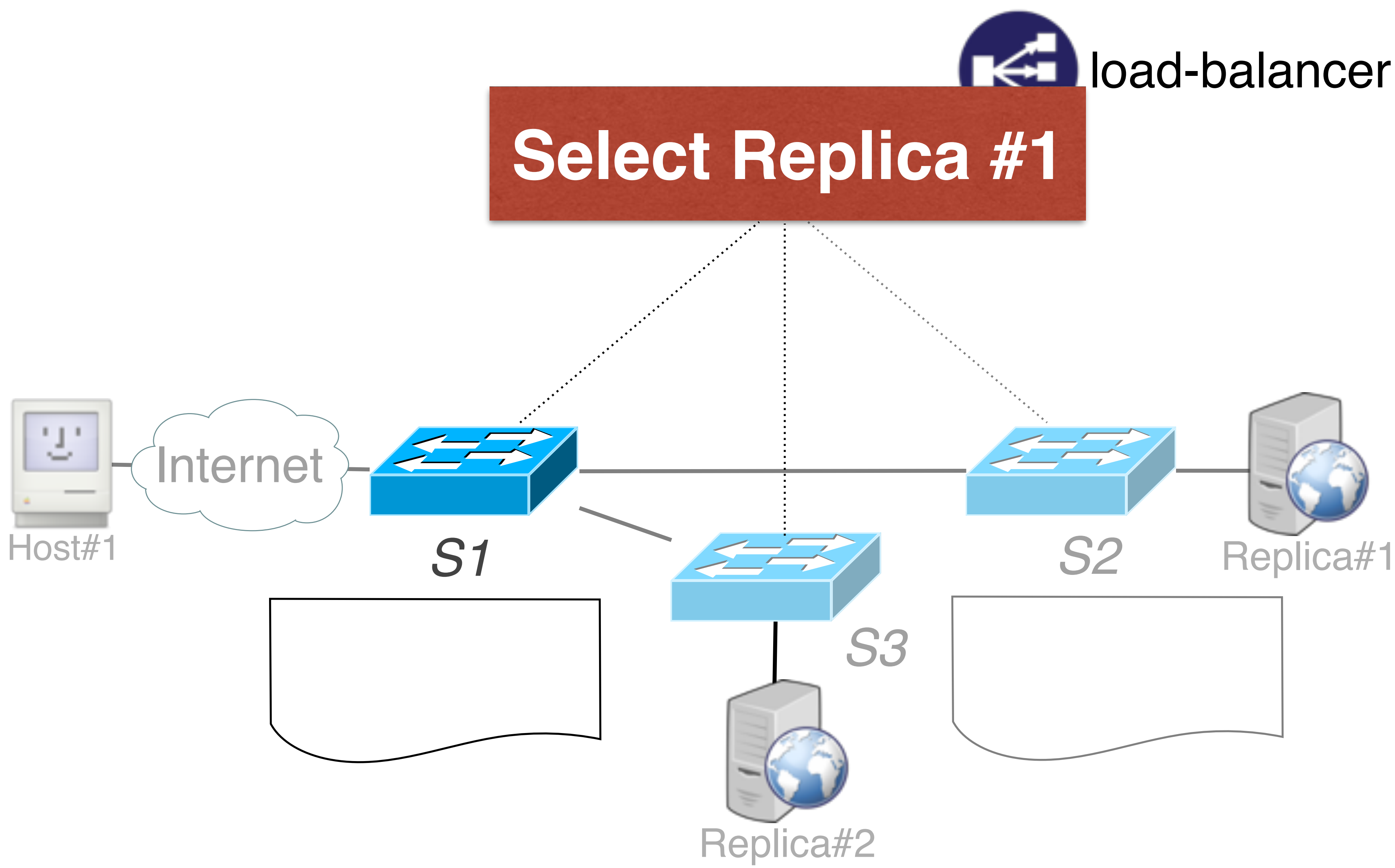
1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)



Load Balancer Application

Recorded Event Trace:

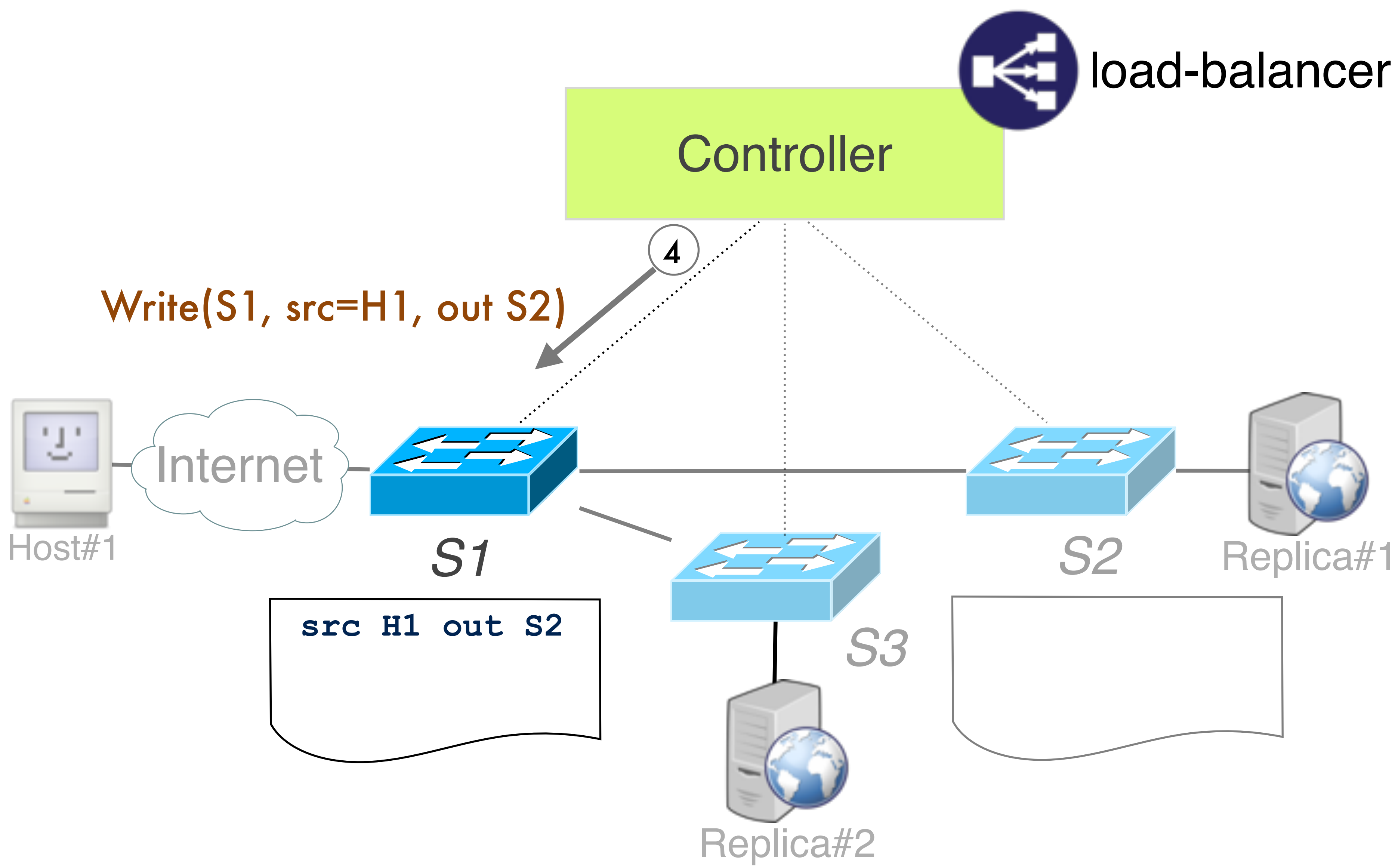
1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)



Load Balancer Application

Recorded Event Trace:

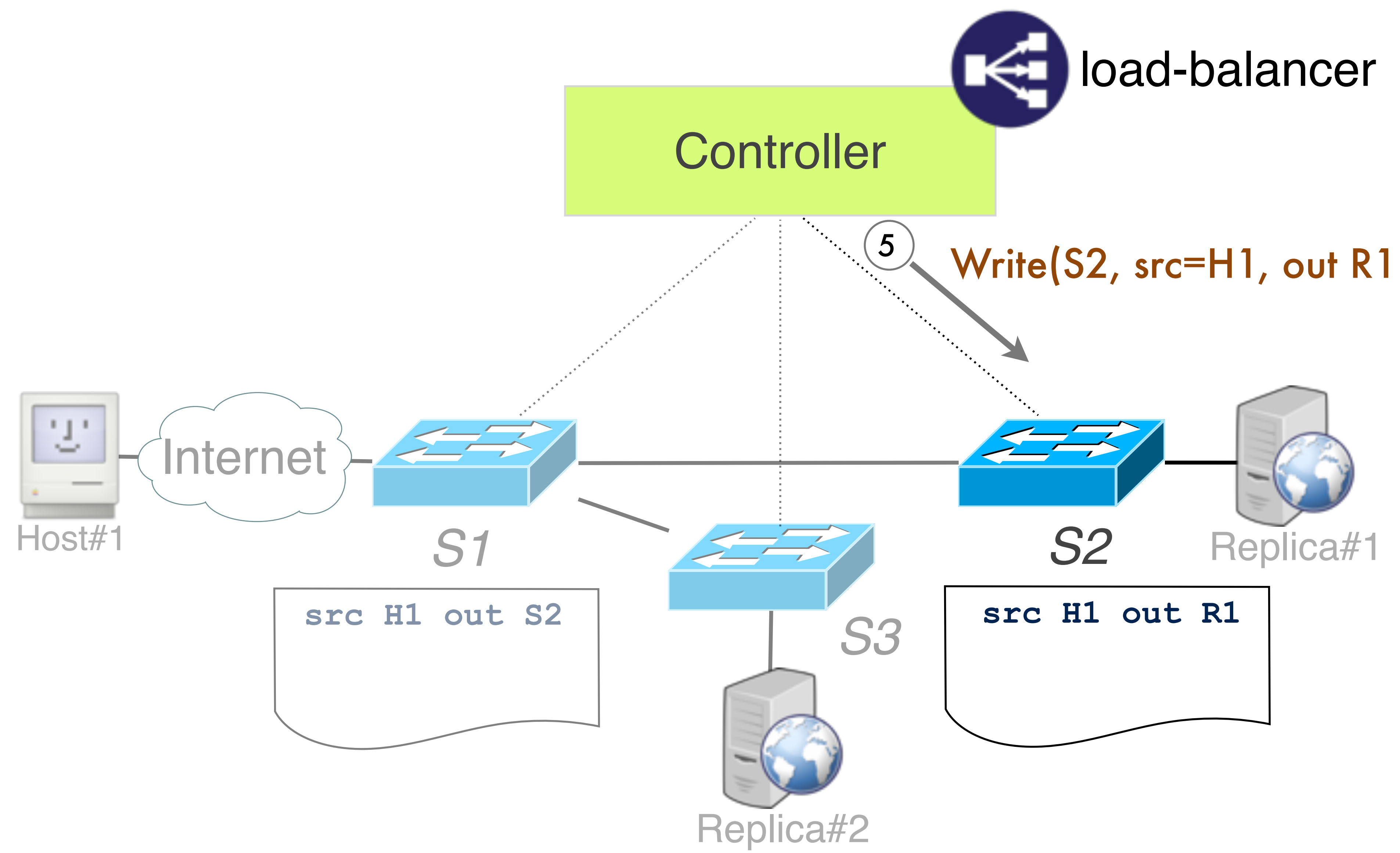
1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)



Load Balancer Application

Recorded Event Trace:

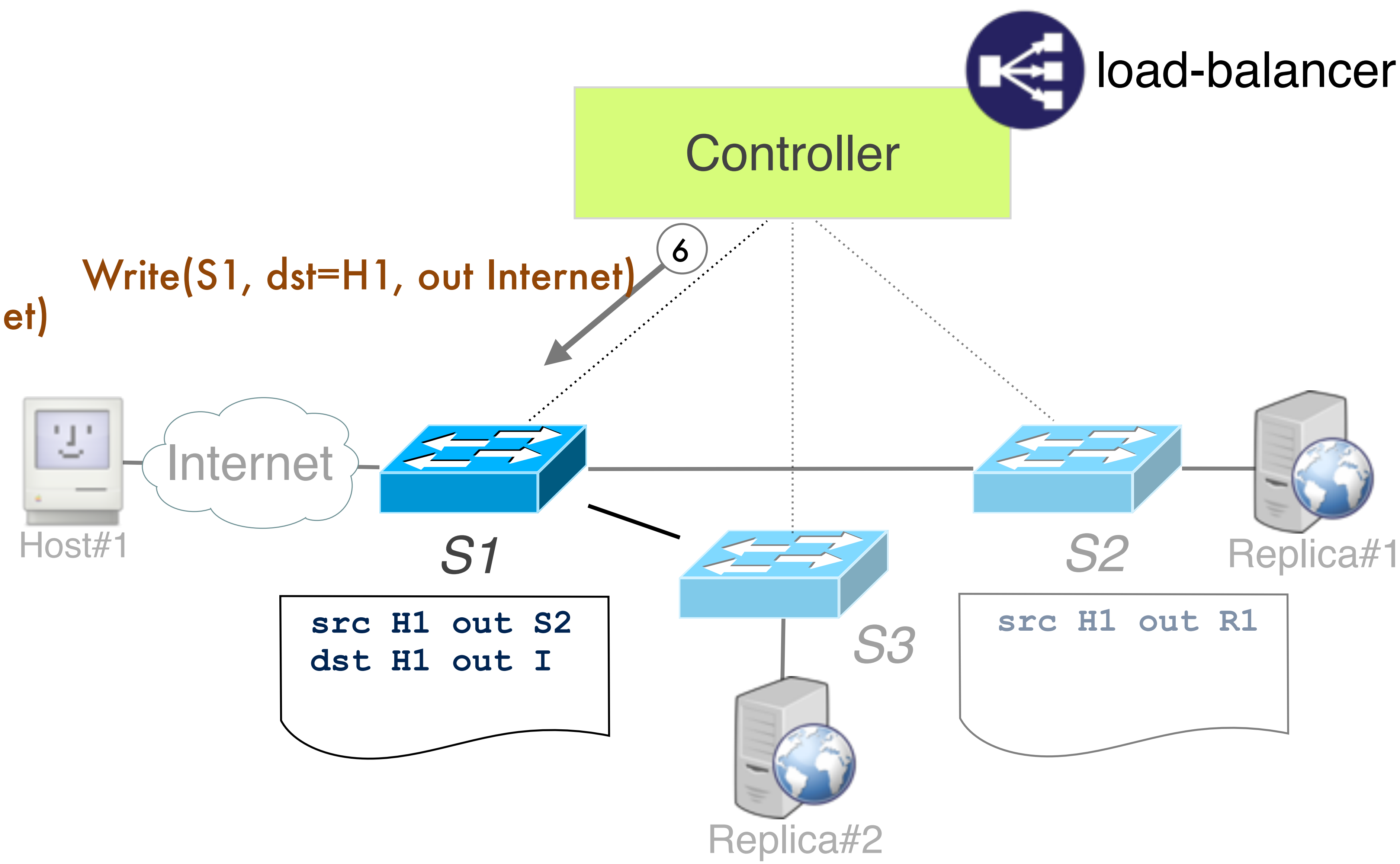
1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
5. Write(S2, src=H1, out R1)



Load Balancer Application

Recorded Event Trace:

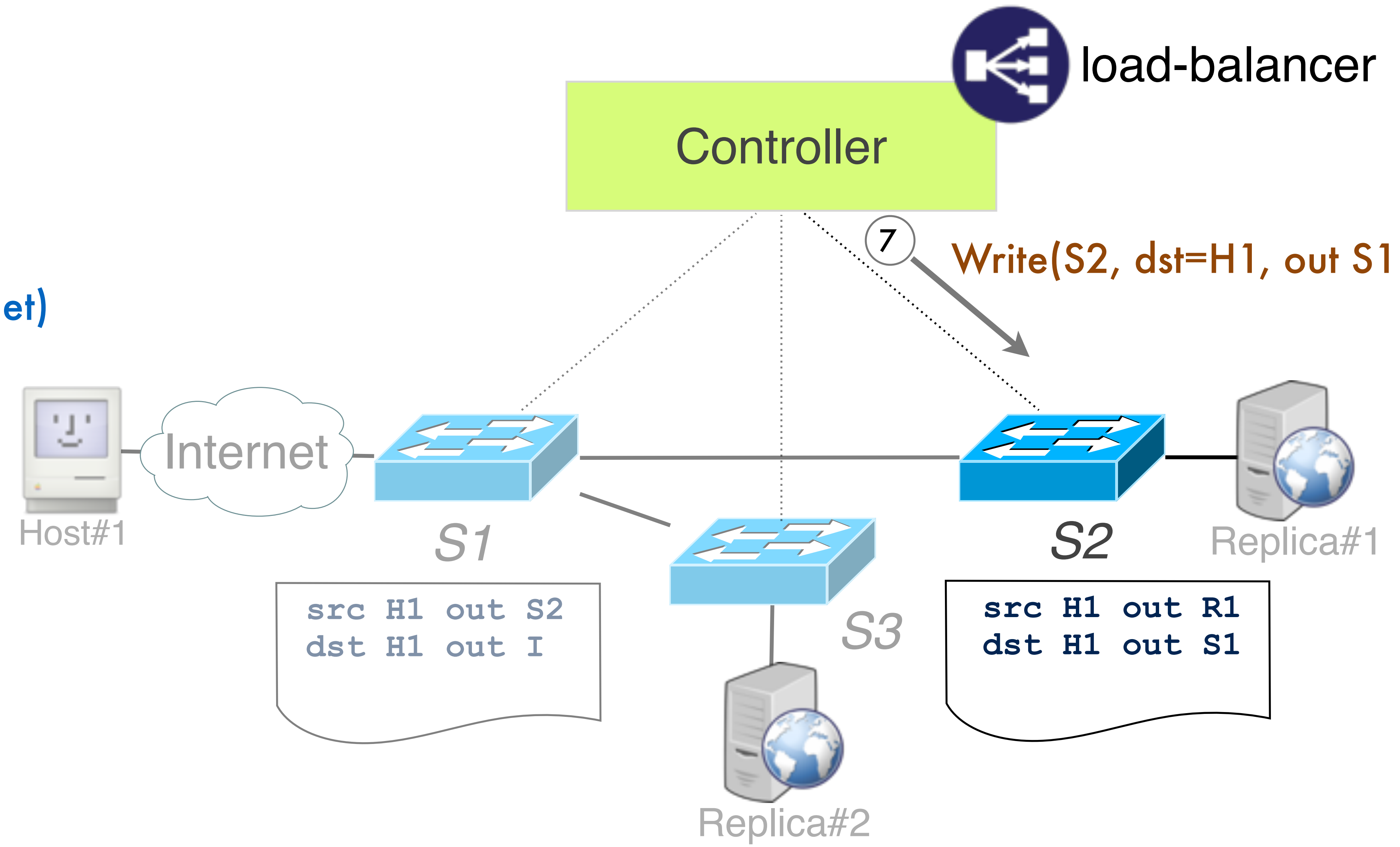
1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
5. Write(S2, src=H1, out R1)
6. Write(S1, dst=H1, out Internet)



Load Balancer Application

Recorded Event Trace:

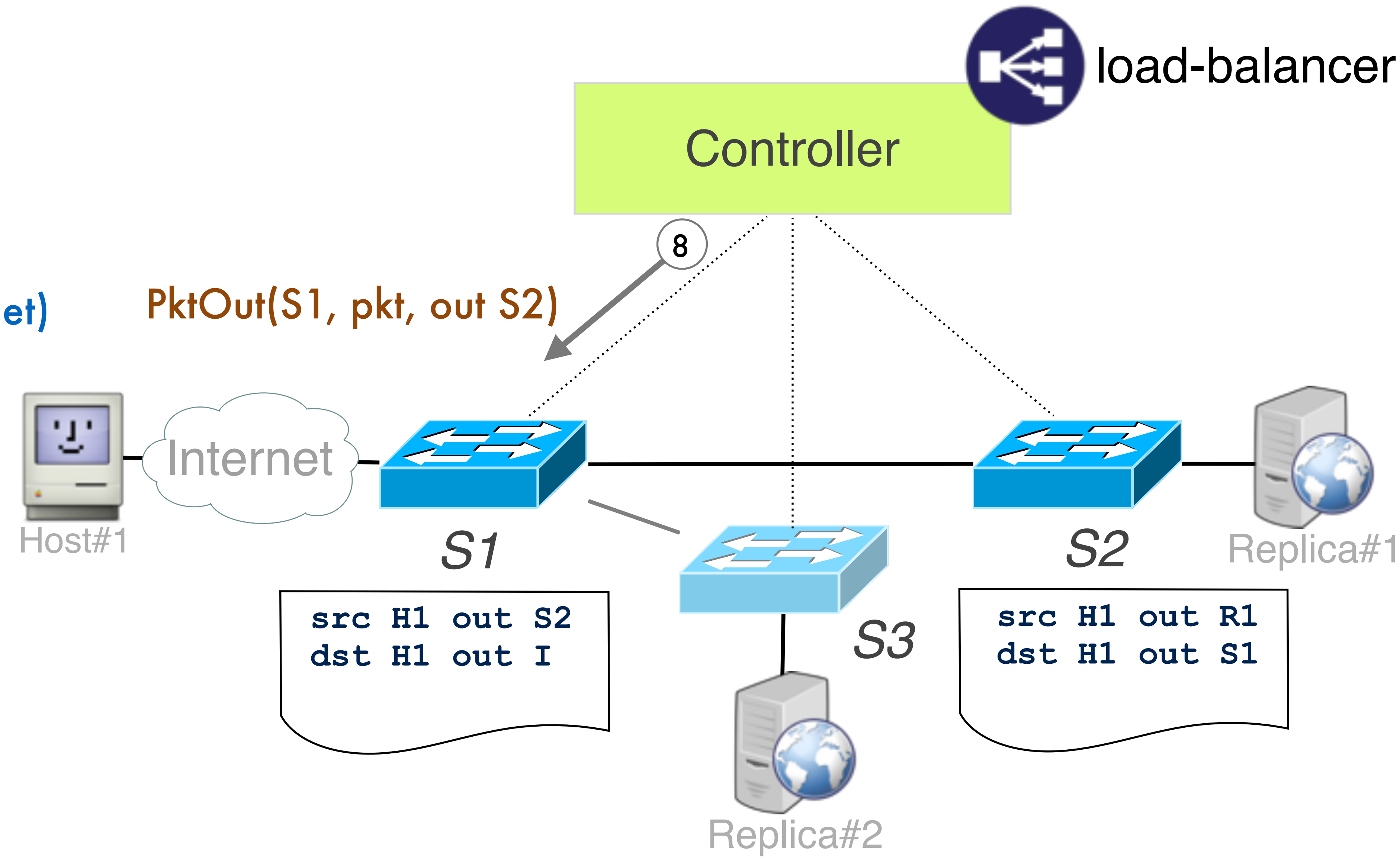
1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
5. Write(S2, src=H1, out R1)
6. Write(S1, dst=H1, out Internet)
7. Write(S2, dst=H1, out S1)



Load Balancer Application

Recorded Event Trace:

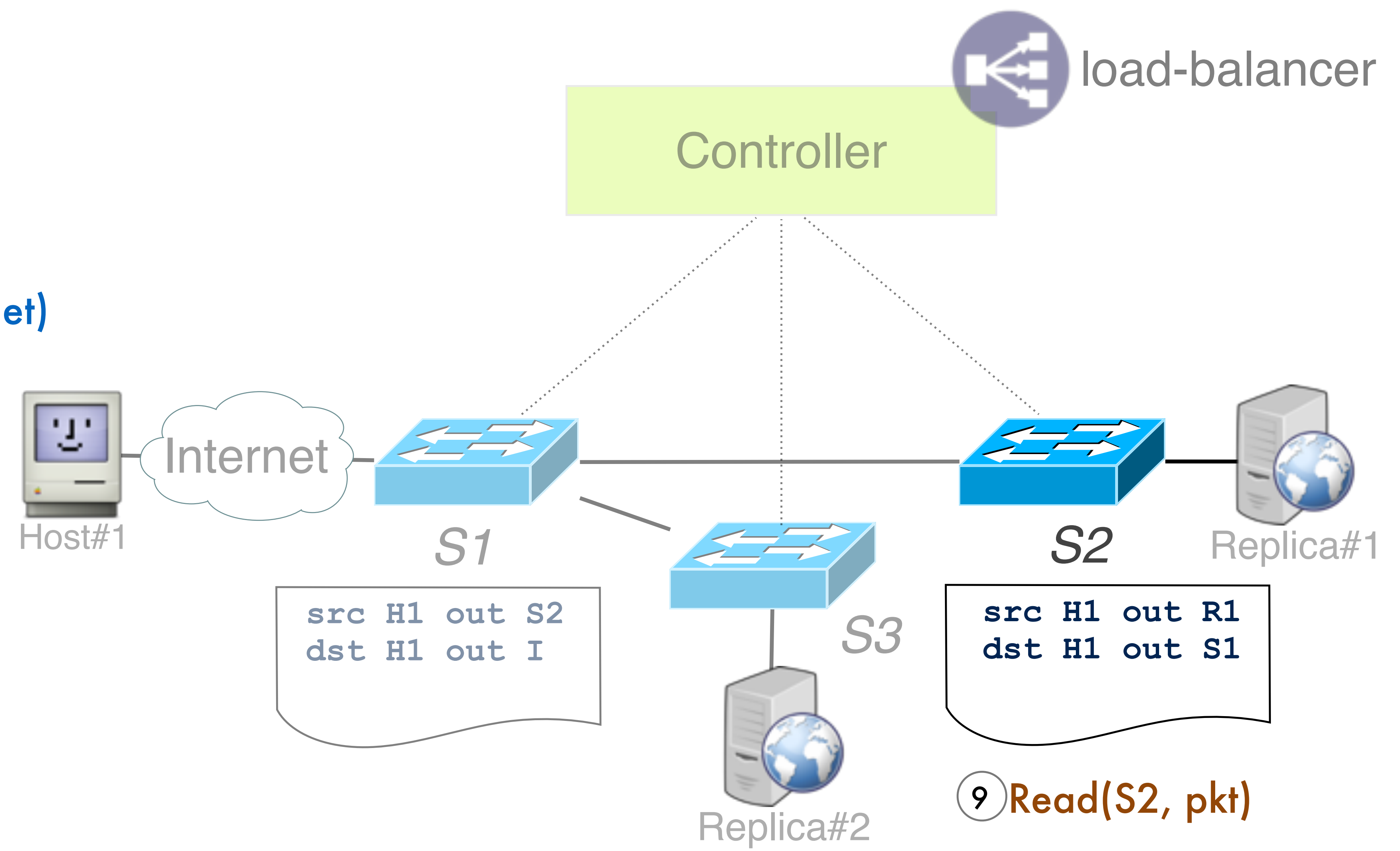
1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
5. Write(S2, src=H1, out R1)
6. Write(S1, dst=H1, out Internet)
7. Write(S2, dst=H1, out S1)
8. PktOut(S1, pkt, S2)



Load Balancer Application

Recorded Event Trace:

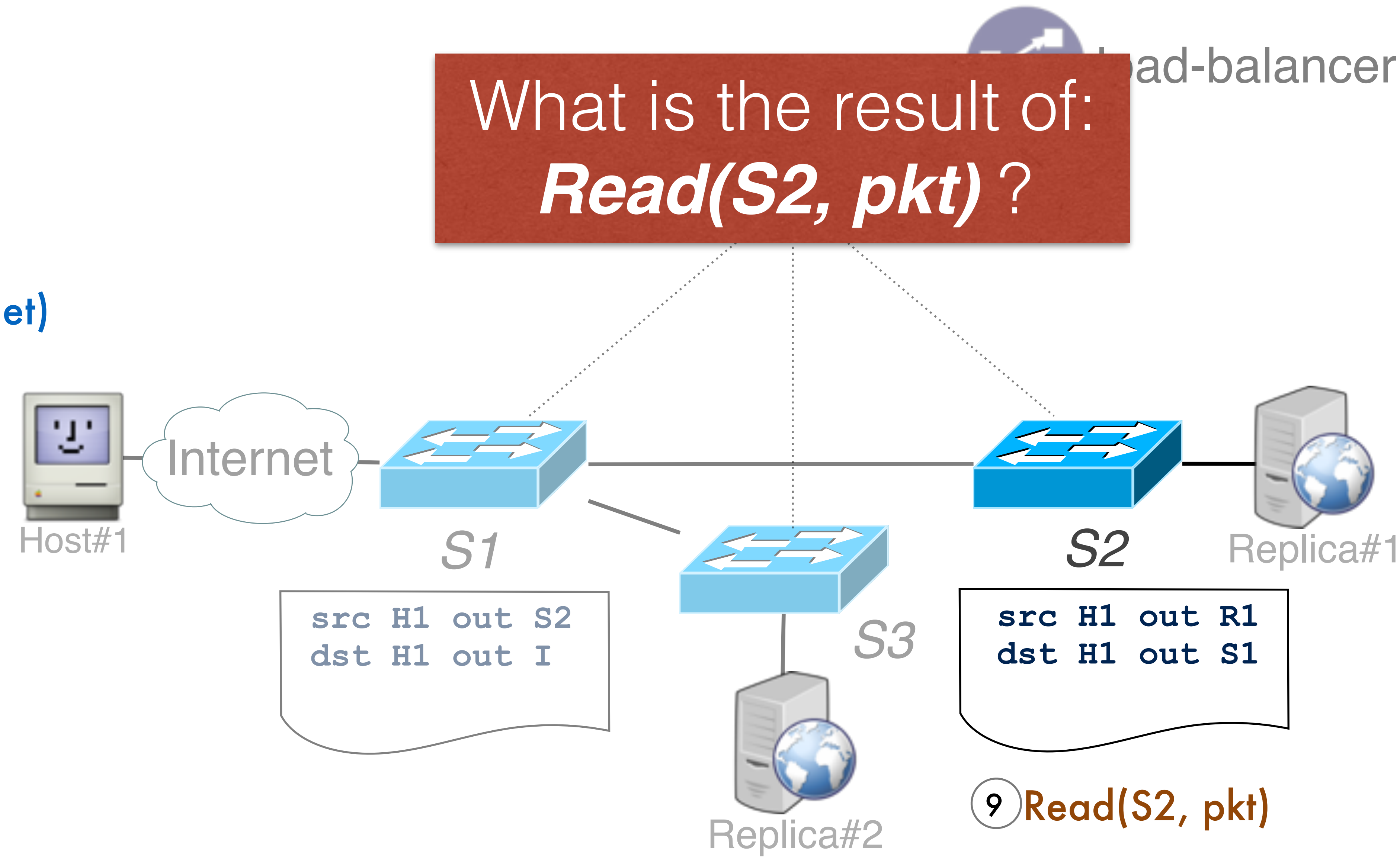
1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
5. Write(S2, src=H1, out R1)
6. Write(S1, dst=H1, out Internet)
7. Write(S2, dst=H1, out S1)
8. PktOut(S1, pkt, S2)
9. Read(S2, pkt)



Load Balancer Application

Recorded Event Trace:

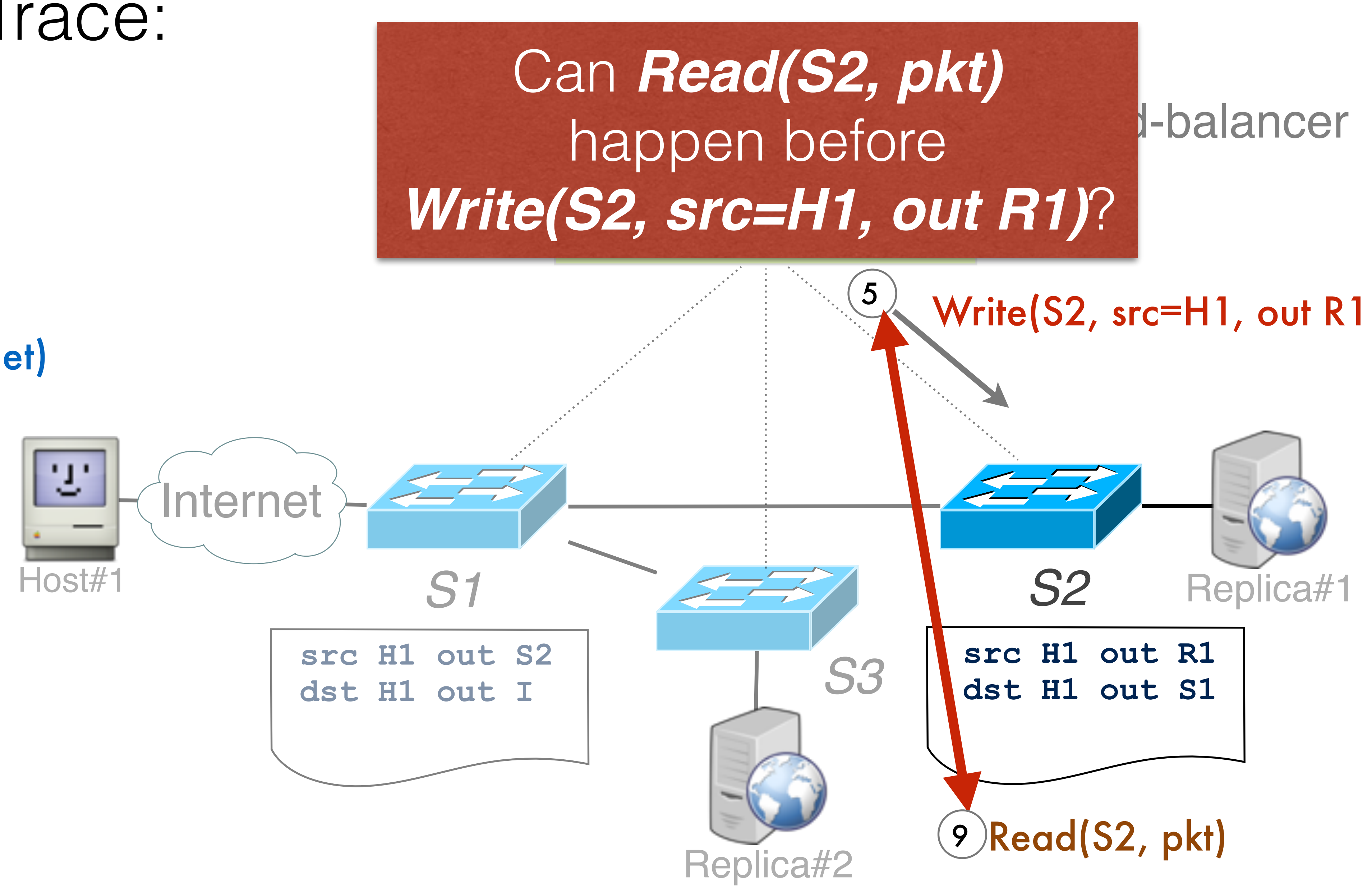
1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
5. Write(S2, src=H1, out R1)
6. Write(S1, dst=H1, out Internet)
7. Write(S2, dst=H1, out S1)
8. PktOut(S1, pkt, S2)
9. Read(S2, pkt)



Load Balancer Application

Recorded Event Trace:

1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
5. **Write(S2, src=H1, out R1)**
6. Write(S1, dst=H1, out Internet)
7. Write(S2, dst=H1, out S1)
8. PktOut(S1, pkt, S2)
9. **Read(S2, pkt)**

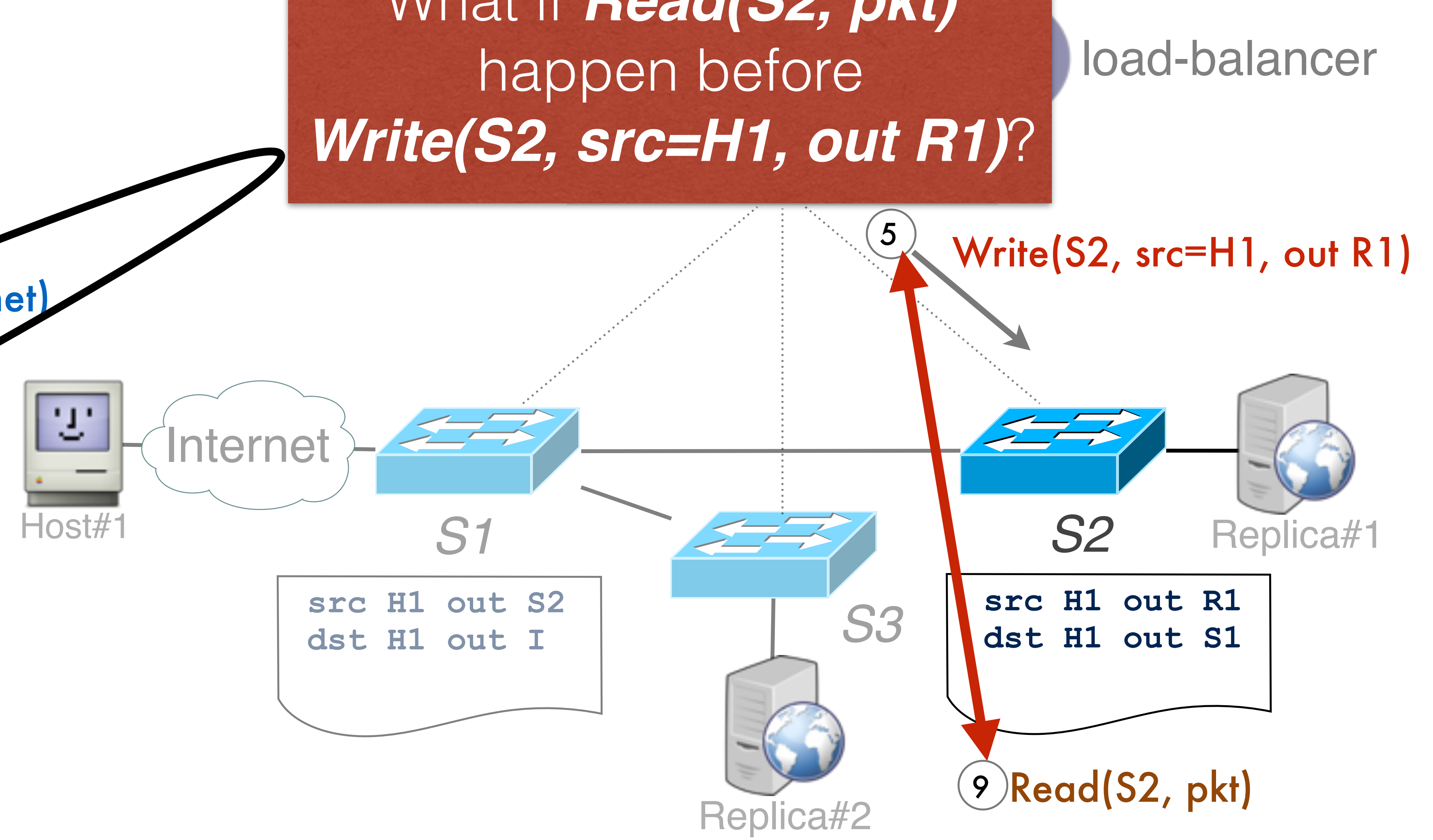


Load Balancer Application

Recorded Event Trace:

1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
5. **Write(S2, src=H1, out R1)**
6. Write(S1, dst=H1, out Internet)
7. Write(S2, dst=H1, out S1)
8. PktOut(S1, pkt, S2)
9. **Read(S2, pkt)**

What if ***Read(S2, pkt)*** happen before ***Write(S2, src=H1, out R1)***?

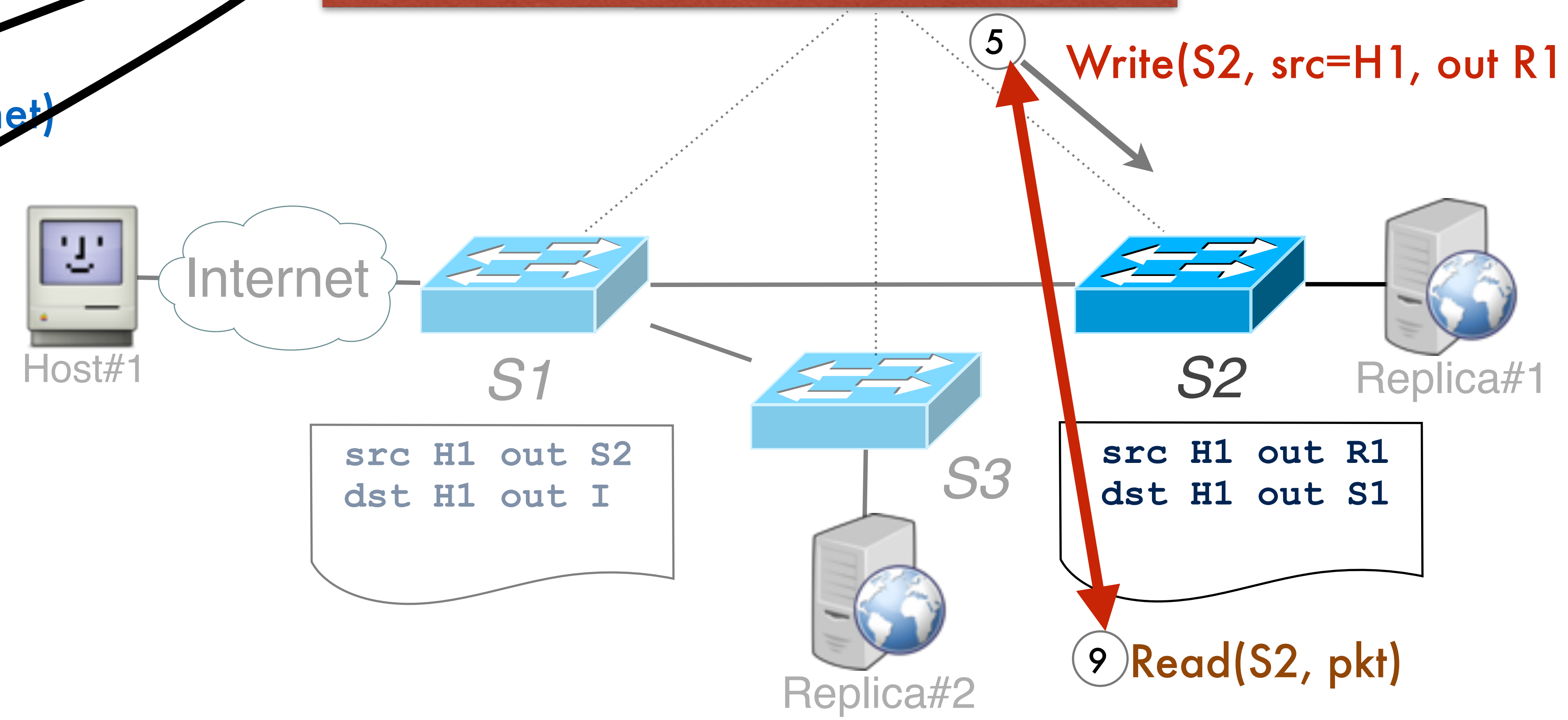


Load Balancer Application

Reordered Event Trace:

1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
9. **Read(S2, pkt)**
6. Write(S1, dst=H1, out Internet)
7. Write(S2, dst=H1, out S1)
8. PktOut(S1, pkt, S2)
5. **Write(S2, src=H1, out R1)**

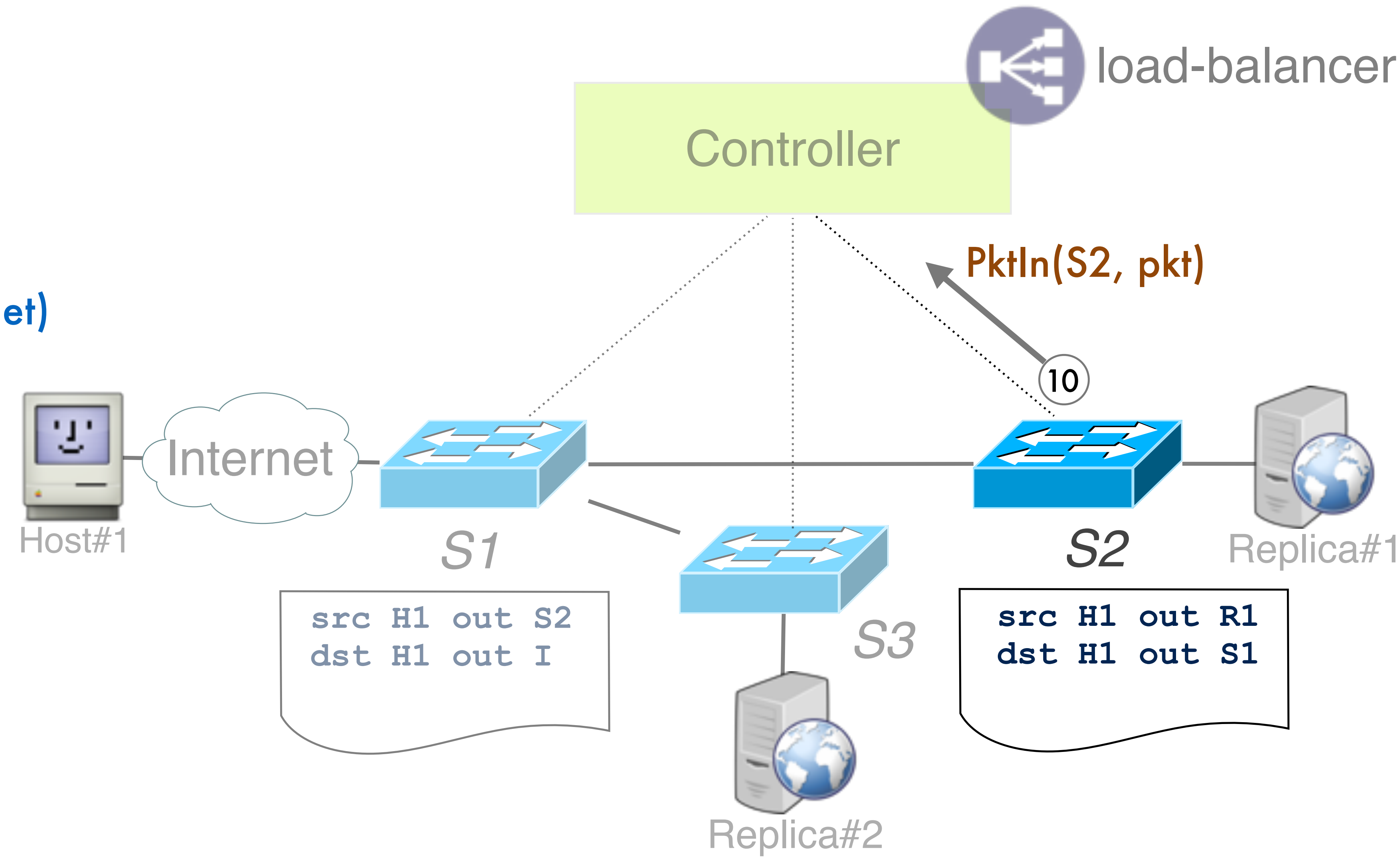
What if ***Read(S2, pkt)*** happen before ***Write(S2, src=H1, out R1)***?



Load Balancer Application

Reordered Event Trace:

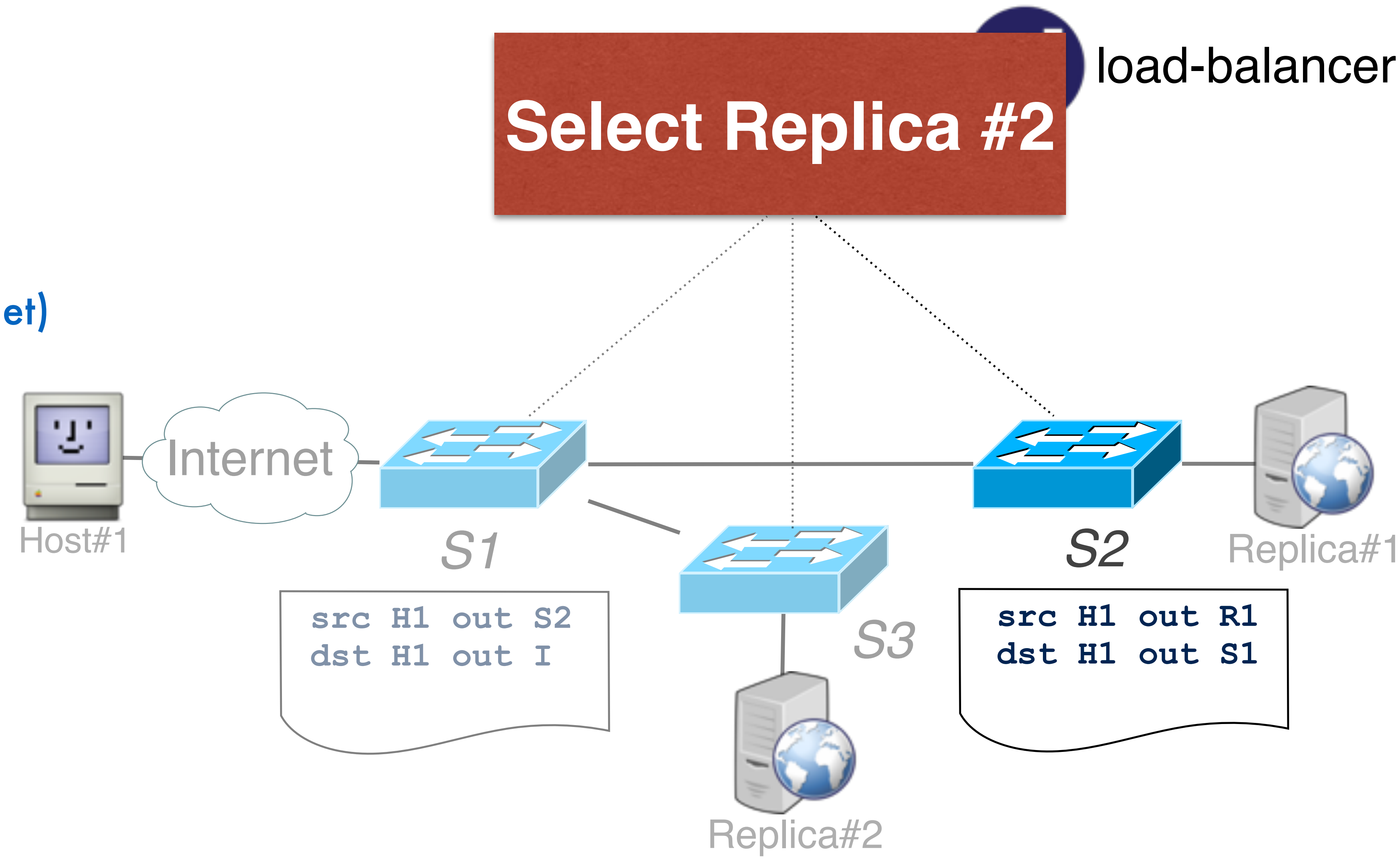
1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
9. Read(S2, pkt)
6. Write(S1, dst=H1, out Internet)
7. Write(S2, dst=H1, out S1)
8. PktOut(S1, pkt, S2)
5. Write(S2, src=H1, out R1)
10. PktIn(S2, pkt)



Load Balancer Application

Reordered Event Trace:

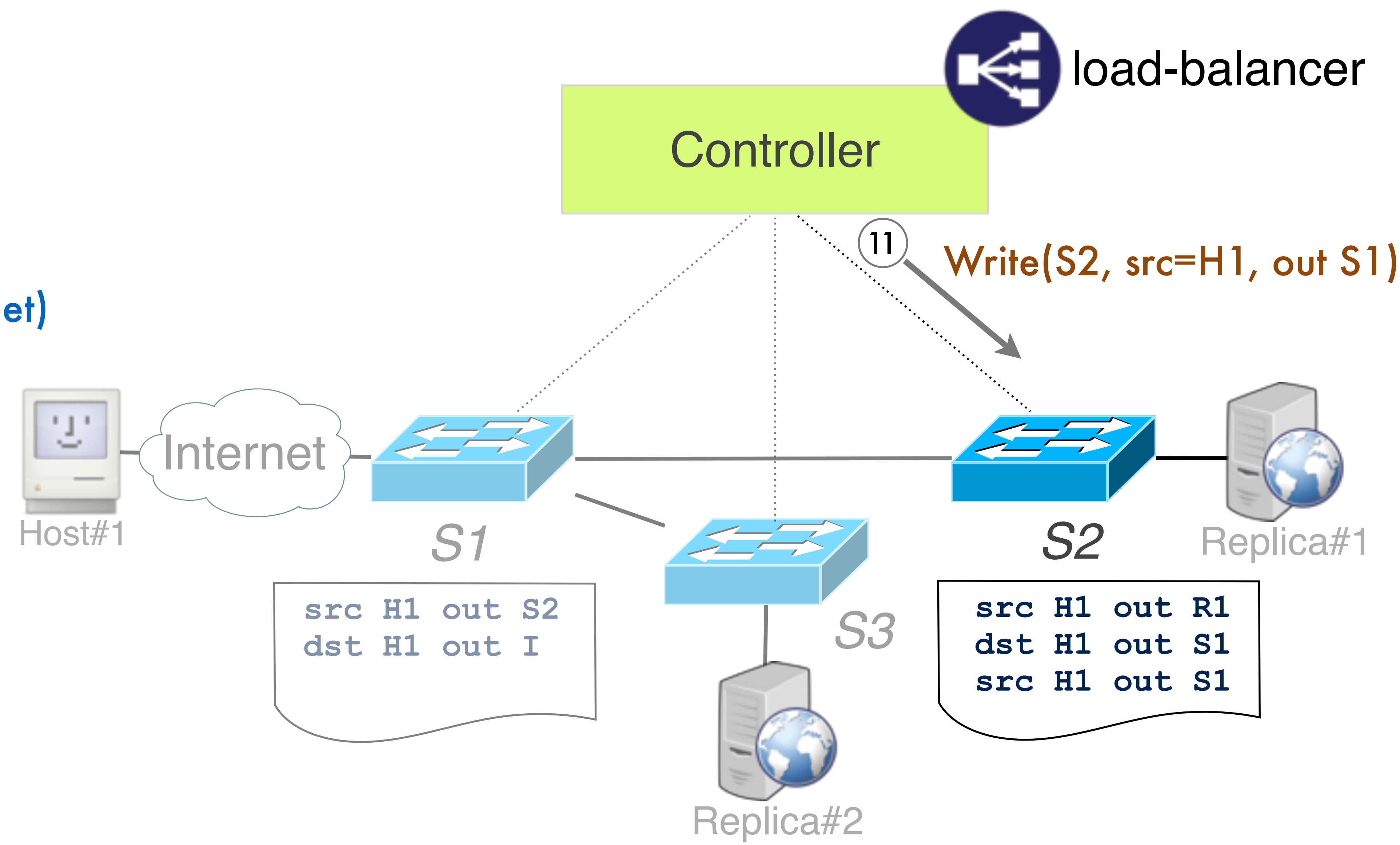
1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
9. Read(S2, pkt)
6. Write(S1, dst=H1, out Internet)
7. Write(S2, dst=H1, out S1)
8. PktOut(S1, pkt, S2)
5. Write(S2, src=H1, out R1)
10. PktIn(S2, pkt)



Load Balancer Application

Reordered Event Trace:

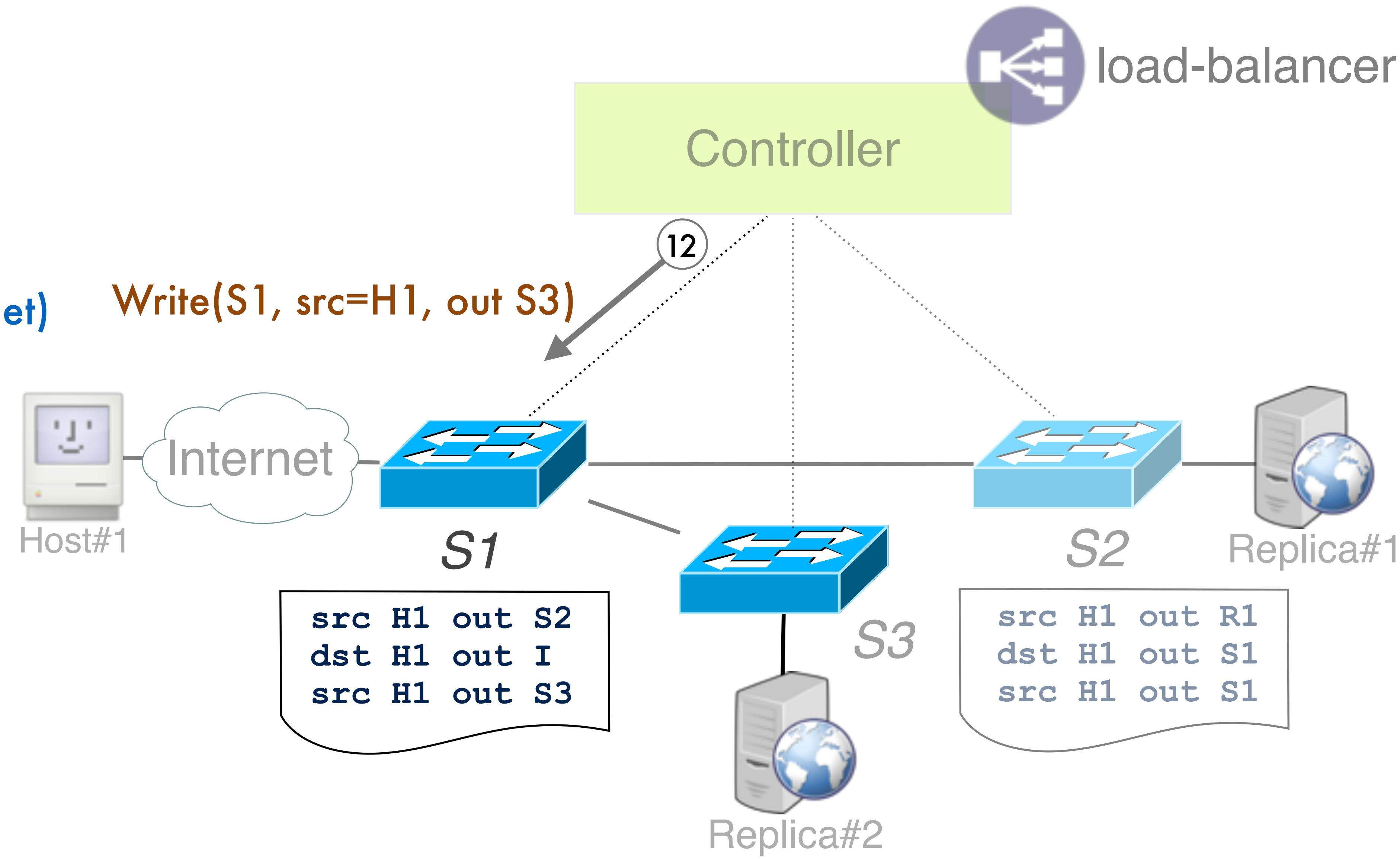
1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)
4 Write(S1, src=H1, out S2)
9 Read(S2, pkt)
6 Write(S1, dst=H1, out Internet)
7 Write(S2, dst=H1, out S1)
8 PktOut(S1, pkt, S2)
5 Write(S2, src=H1, out R1)
10 PktIn(S2, pkt)
11 Write(S2, src=H1, out S1)



Load Balancer Application

Reordered Event Trace:

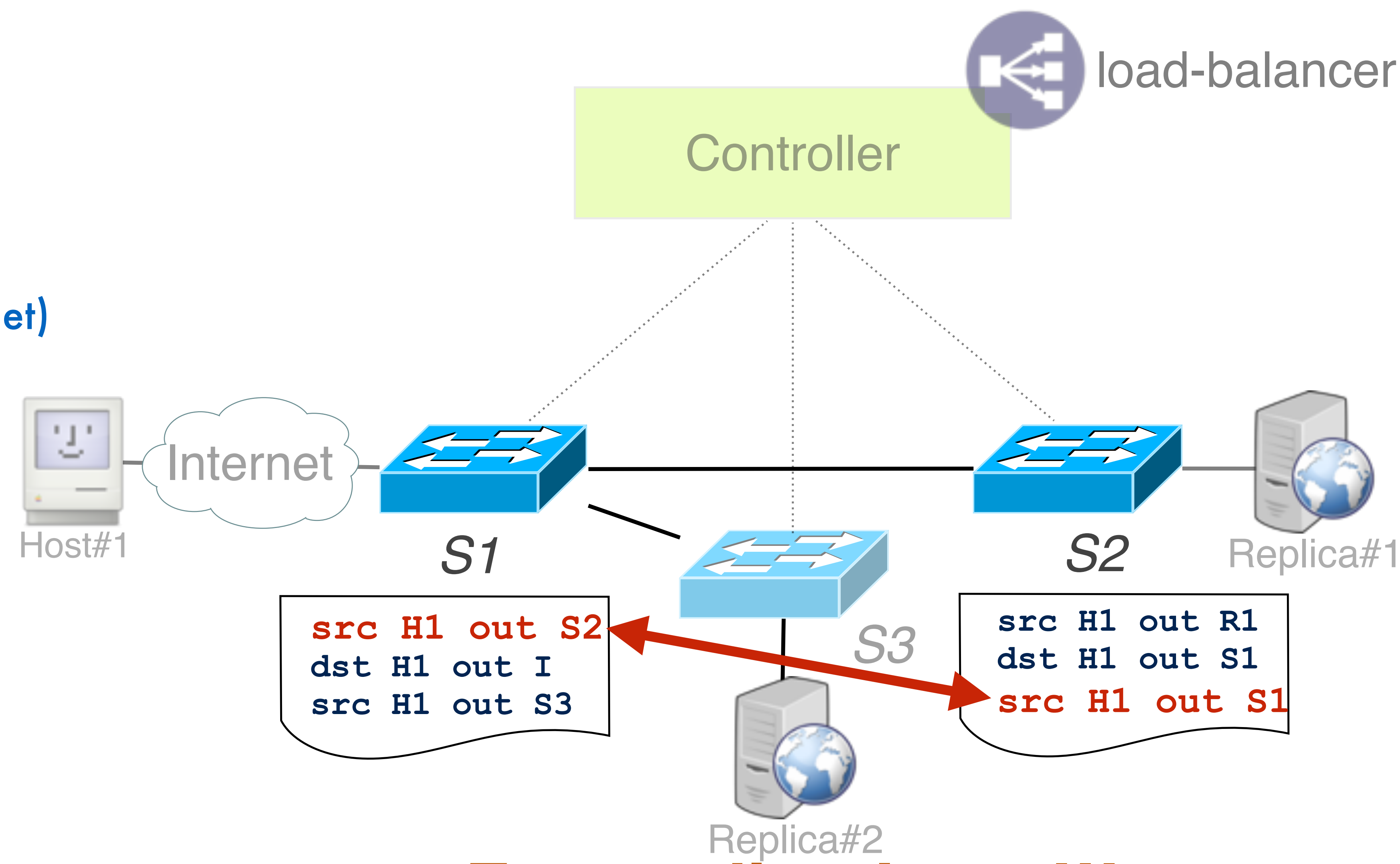
1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
9. Read(S2, pkt)
6. Write(S1, dst=H1, out Internet)
7. Write(S2, dst=H1, out S1)
8. PktOut(S1, pkt, S2)
5. Write(S2, src=H1, out R1)
10. PktIn(S2, pkt)
11. Write(S2, src=H1, out S1)
12. Write(S1, src=H1, out S3)



Load Balancer Application

Reordered Event Trace:

1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)
4 Write(S1, src=H1, out S2)
9 Read(S2, pkt)
6 Write(S1, dst=H1, out Internet)
7 Write(S2, dst=H1, out S1)
8 PktOut(S1, pkt, S2)
5 Write(S2, src=H1, out R1)
10 PktIn(S2, pkt)
11 Write(S2, src=H1, out S1)
12 Write(S1, src=H1, out S3)



Forwarding Loop!!!

SDNRacer detected this **real bug** in
Floodlight's Load Balancer.

Project
Floodlight

Key Observation

The cause of this bug is **interference** on the Flow Table caused by **concurrent** writes by the controller and reads triggered by packets in the network.

SDNRacer

Detecting concurrency violations

Precise notion of interference

Checks for high-level properties

Implementation and evaluation

Data Races in SDN

Data Race: two *unordered* events accessing the Flow Table where one is a write.

- Flow tables are memory locations.
- The controller generate writes events.
- Packets trigger read events.

Formalizing Asynchrony in SDN

Need to identify *causality* between events.

Happens-Before for SDN

- A switch may send a *PktIn* message after reading it

Read(s1, pkt)→ PktIn(s1, pkt)

- A controller may issue *Write* after *PktIn*

PktIn(s1, pkt) →write(s1, match predicate, out S2)

PktIn(s1, pkt) →write(s2, match predicate, out R1)

HB-relation example

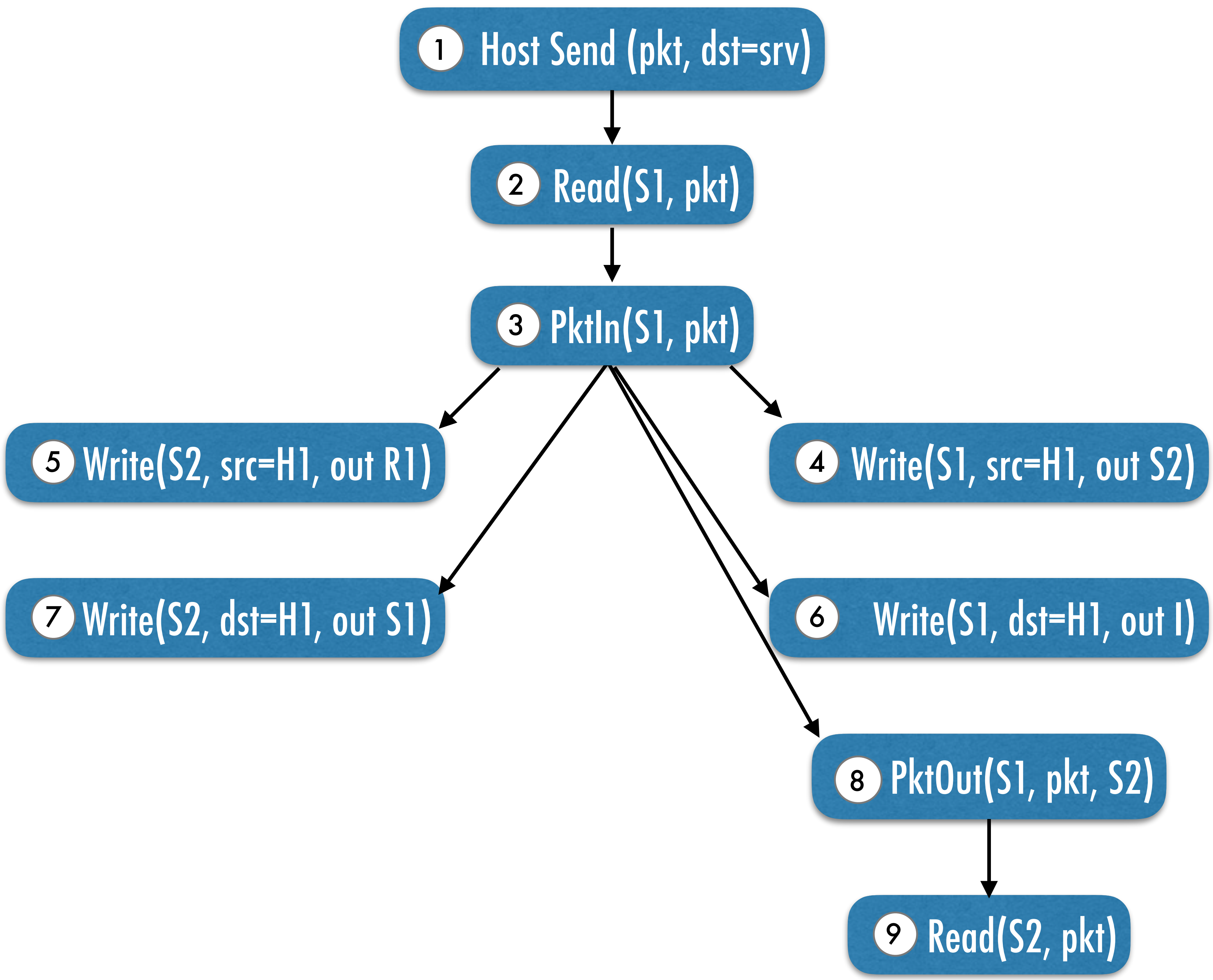
Events Trace:

1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)
4 Write(S1, src=H1, out S2)
9 Read(S2, pkt)
6 Write(S1, dst=H1, out Internet)
7 Write(S2, dst=H1, out S1)
8 PktOut(S1, pkt, S2)
5 Write(S2, src=H1, out R1)
10 PktIn(S2, pkt)

HB-relation example

Events Trace:

1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)
4 Write(S1, src=H1, out S2)
9 Read(S2, pkt)
6 Write(S1, dst=H1, out Internet)
7 Write(S2, dst=H1, out S1)
8 PktOut(S1, pkt, S2)
5 Write(S2, src=H1, out R1)
10 PktIn(S2, pkt)



Analysis Results

For Floodlight Load Balancer

703,864 Races

Analysis Results

For Floodlight Load Balancer

703,864 Races

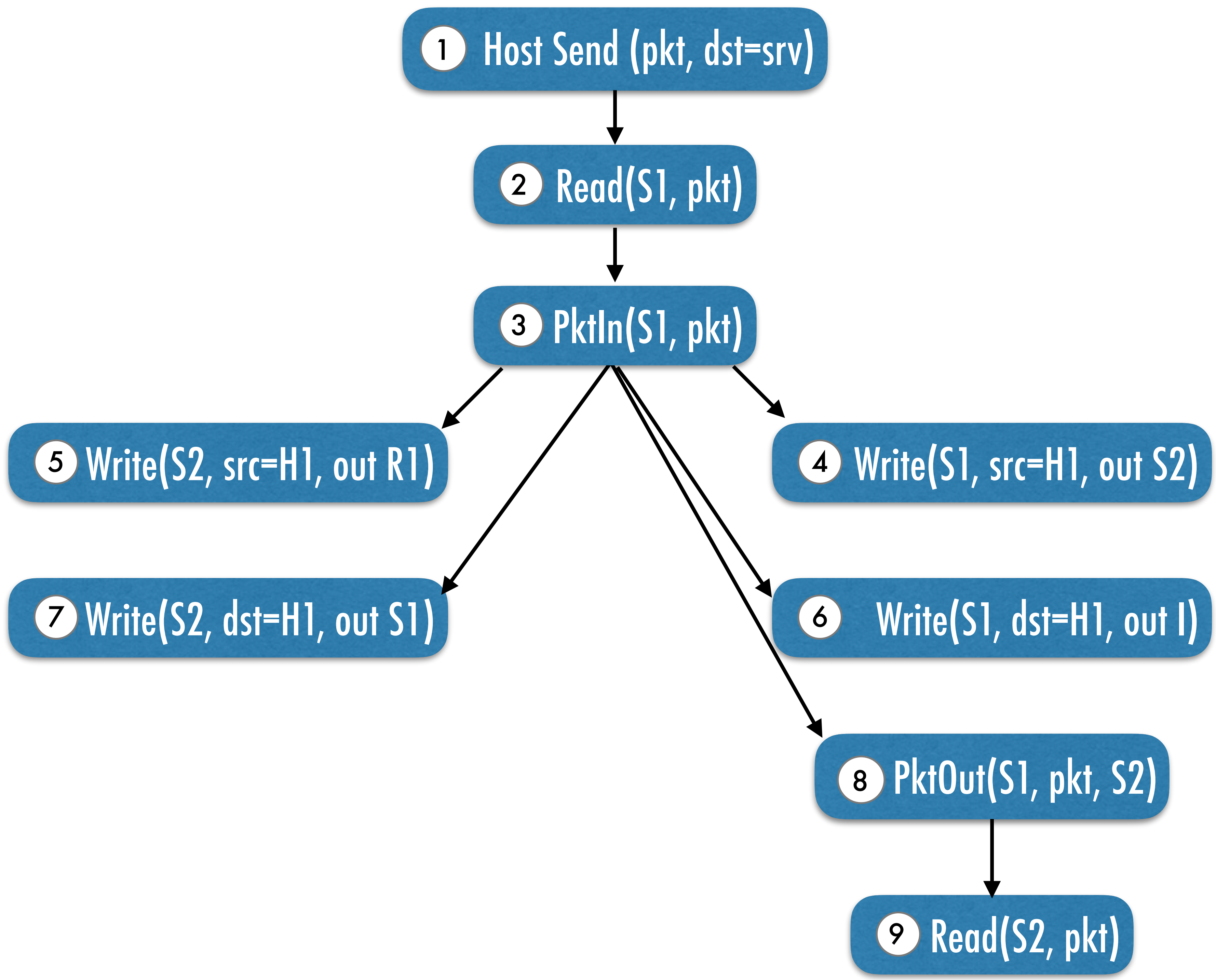
Too imprecise!!!

How can we **reduce**
the number of reports?

HB-relation example

Events Trace:

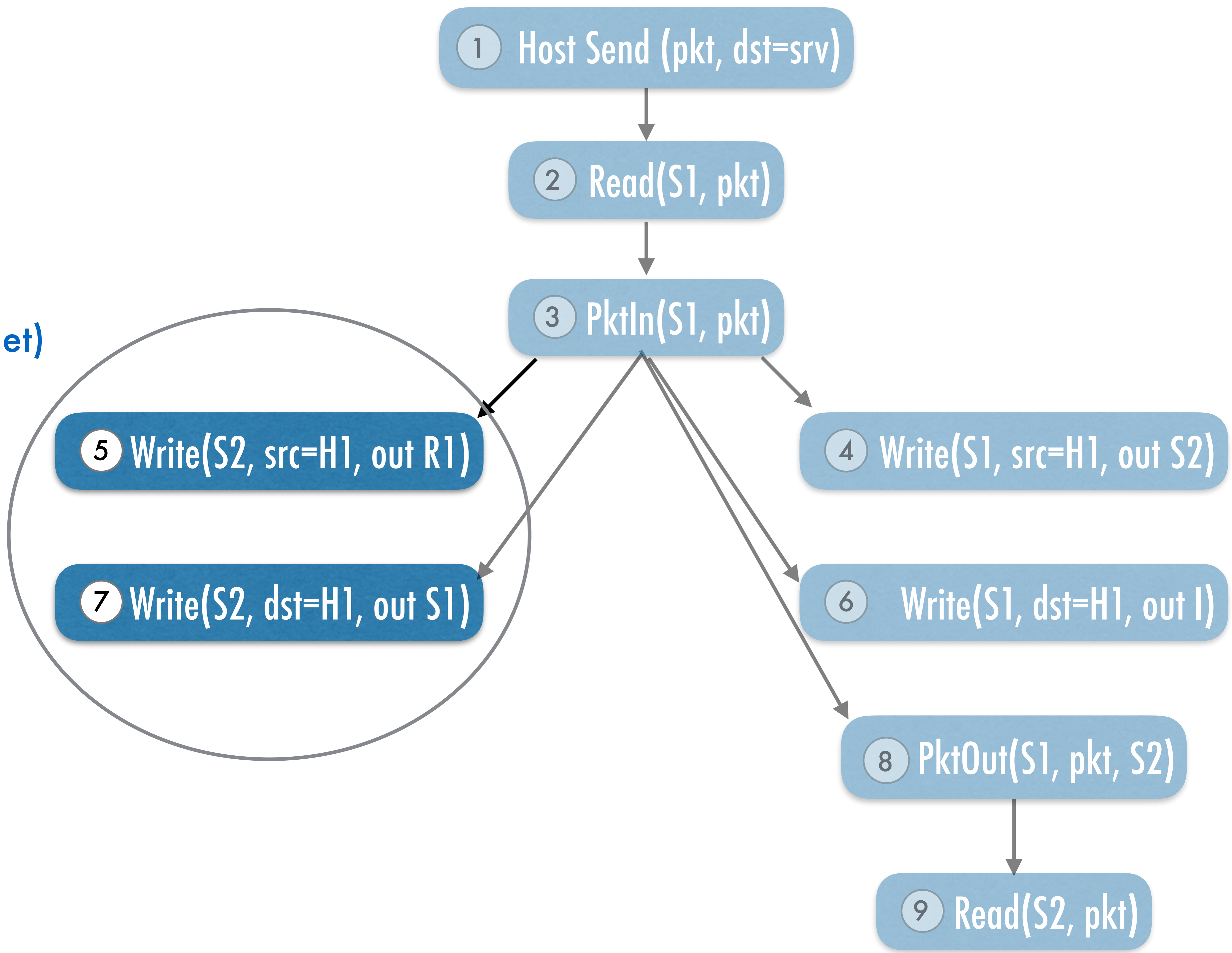
1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)
4 Write(S1, src=H1, out S2)
9 Read(S2, pkt)
6 Write(S1, dst=H1, out Internet)
7 Write(S2, dst=H1, out S1)
8 PktOut(S1, pkt, S2)
5 Write(S2, src=H1, out R1)
10 PktIn(S2, pkt)



HB-relation example

Events Trace:

1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)
4 Write(S1, src=H1, out S2)
9 Read(S2, pkt)
6 Write(S1, dst=H1, out Internet)
7 **Write(S2, dst=H1, out S1)**
8 PktOut(S1, pkt, S2)
5 **Write(S2, src=H1, out R1)**
10 PktIn(S2, pkt)



Non-Interfering Events

Events Trace:

1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)
4 Write(S1, src=H1, out S2)
9 Read(S2, pkt)
6 Write(S1, dst=H1, out Internet)
7 **Write(S2, dst=H1, out S1)**
8 PktOut(S1, pkt, S2)
5 **Write(S2, src=H1, out R1)**
10 PktIn(S2, pkt)

5 Write(S2, src=H1, out R1)

7 Write(S2, dst=H1, out S1)

Event 5 match predicate true for packets with source H1.

Event 7 match predicate true for packets with destination H1.

Non-Interfering Events

Read (s1, **dst=X**)

Write(s1, **dst=Y**, output port 1)

Write(s1, **dst=Z**, output port 2)

Events with the same net effect

Write(s1, dst=X, **output port 1**)

Write(s1, dst=X, **output port 1**)

SDNRacer

Detecting concurrency violations

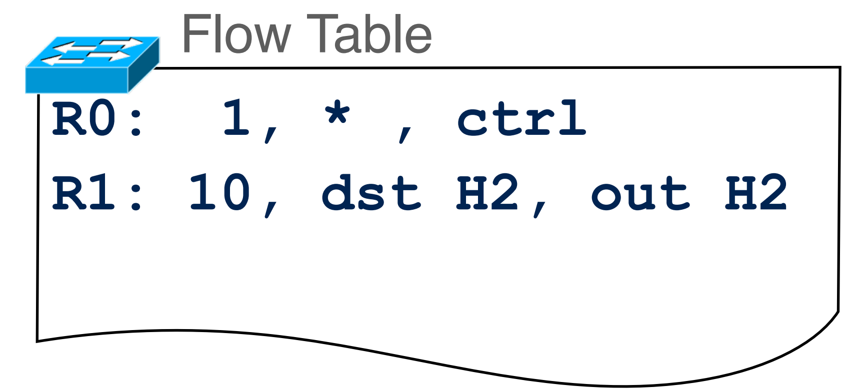
Precise notion of interference

Checks for high-level properties

Implementation and evaluation

Capture interference at the Flow Table

Observation: Flow Table can be seen as a high-level ADT with operations:



read(pkt)/e_{read}

A packet is matched against entry in the flow table.

add(e_{add}, no overlap)

Add a new entry to the flow table.

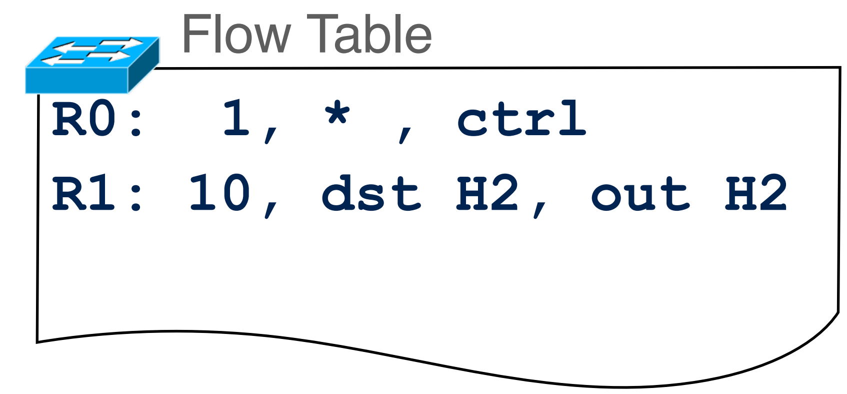
mod(e_{mod}, strict)

A mod operation modifies existing entries in the flow table.

del(e_{del}, strict)

A del operation deletes all entries that match the entry in the flow table.

Capture interference at the Flow Table



Capture high-level interference via commutativity: Two operations commute if reordering them have the same effect on the the network state.

Commutativity Spec of Flow Table ADT

$$\varphi^{read(pkt)/e_{read}}_{add(e_{add},\, no\_overlap)} := \begin{cases} \neg(e_{read} \neq none \wedge e_{read} = e_{add}) & \text{if } add <_\pi read \\ \neg(pkt.h \subseteq e_{add}.m \wedge (e_{read} = none \\ \quad \vee (e_{read}.p \leq e_{add}.p \wedge e_{read}.a \neq e_{add}.a))) & \text{if } read <_\pi add \end{cases}$$

$$\varphi^{read(pkt)/e_{read}}_{mod(e_{mod},\, strict)} := \begin{cases} \neg(e_{read} \neq none \wedge e_{read} \overset{strict}{\subseteq} e_{mod} \wedge e_{read}.a = e_{mod}.a) & \text{if } mod <_\pi read \\ \neg(e_{read} \neq none \wedge pkt.h \subseteq e_{mod}.m \wedge e_{read}.a \neq e_{mod}.a) & \text{if } read <_\pi mod \end{cases}$$

$$\varphi^{read(pkt)/e_{read}}_{del(e_{del},\, strict)} := \begin{cases} \neg(pkt.h \subseteq e_{del}.m) & \text{if } del <_\pi read \\ \neg(e_{read} \neq none \wedge deletes(e_{del}, e_{read}, strict)) & \text{if } read <_\pi del \end{cases}$$

$$\varphi^{del(e_{del},\, strict_{del})}_{mod(e_{mod},\, strict_{mod})} := \begin{cases} \neg(deletes(e_{del}, e_{mod}, true)) & \text{if } strict_{mod} \\ \neg(e_{del}.m \cap e_{mod}.m \neq \emptyset) & \text{otherwise} \end{cases}$$

$$\varphi^{add(e_{add},\, no\_overlap)}_{del(e_{del},\, strict)} := \neg(deletes(e_{del}, e_{add}, strict) \vee (no\_overlap \wedge e_{add} \cap e_{del} \neq \emptyset))$$

$$\varphi^{mod(e_1,\, strict_1)}_{mod(e_2,\, strict_2)} := \begin{cases} \neg(e_1.m \cap e_2.m \neq \emptyset \wedge e_1.a \neq e_2.a) & \text{if } \neg strict_1 \wedge \neg strict_2 \\ \neg(e_1.m = e_2.m \wedge e_1.p = e_2.p \wedge e_1.a \neq e_2.a) & \text{if } strict_1 \wedge strict_2 \\ \neg((e_1 \overset{strict_2}{\subseteq} e_2 \vee e_2 \overset{strict_1}{\subseteq} e_1) \wedge e_1.a \neq e_2.a) & \text{otherwise} \end{cases}$$

$$\varphi^{add(e_{add},\, no\_overlap)}_{mod(e_{mod},\, strict)} := \begin{cases} \neg(e_{add} \overset{strict}{\subseteq} e_{mod} \wedge e_{add}.a \neq e_{mod}.a) & \text{if } \neg no\_overlap \\ \neg(e_{add} \cap e_{mod} \neq \emptyset) & \text{otherwise} \end{cases}$$

$$\varphi^{add(e_1,\, no\_overlap_1)}_{add(e_2,\, no\_overlap_2)} := \begin{cases} \neg(e_1.m \cap e_2.m \neq \emptyset \wedge e_1.p = e_2.p) & \text{if } no\_overlap_1 \vee no\_overlap_2 \\ \neg(e_1.m = e_2.m \wedge e_1.p = e_2.p \wedge e_1.a \neq e_2.a) & \text{otherwise} \end{cases}$$

Figure 3: Commutativity specification of an OpenFlow switch. Two *read* or two *del* operations always commute.

Commutativity Specification of SDN

$$\varphi^{read(pkt)/e_{read}}_{add(e_{add}, no\_overlap)} := \begin{cases} \neg(e_{read} \neq none \wedge e_{read} = e_{add}) & \text{if } add <_\pi read \\ \neg(pkt.h \subseteq e_{add}.m \wedge (e_{read} = none \\ \quad \vee (e_{read}.p \leq e_{add}.p \wedge e_{read}.a \neq e_{add}.a))) & \text{if } read <_\pi add \end{cases}$$

$$\varphi^{read(pkt)/e_{read}}_{mod(e_{mod}, strict)} := \begin{cases} \neg(e_{read} \neq none \wedge e_{read} \overset{strict}{\subseteq} e_{mod} \wedge e_{read}.a = e_{mod}.a) & \text{if } mod <_\pi read \\ \neg(e_{read} \neq none \wedge pkt.h \subseteq e_{mod}.m \wedge e_{read}.a \neq e_{mod}.a) & \text{if } read <_\pi mod \end{cases}$$

$read(pkt)/e_{read}$ $\neg(pkt.h \subseteq e_{del}.m)$ if $del <_\pi read$

$$\varphi^{add(e_1, no\_overlap_1)}_{add(e_2, no\_overlap_2)} := \neg(e_1.m \cap e_2.m \neq \emptyset \wedge e_1.p = e_2.p)$$

$del(e_{del}, strict)$

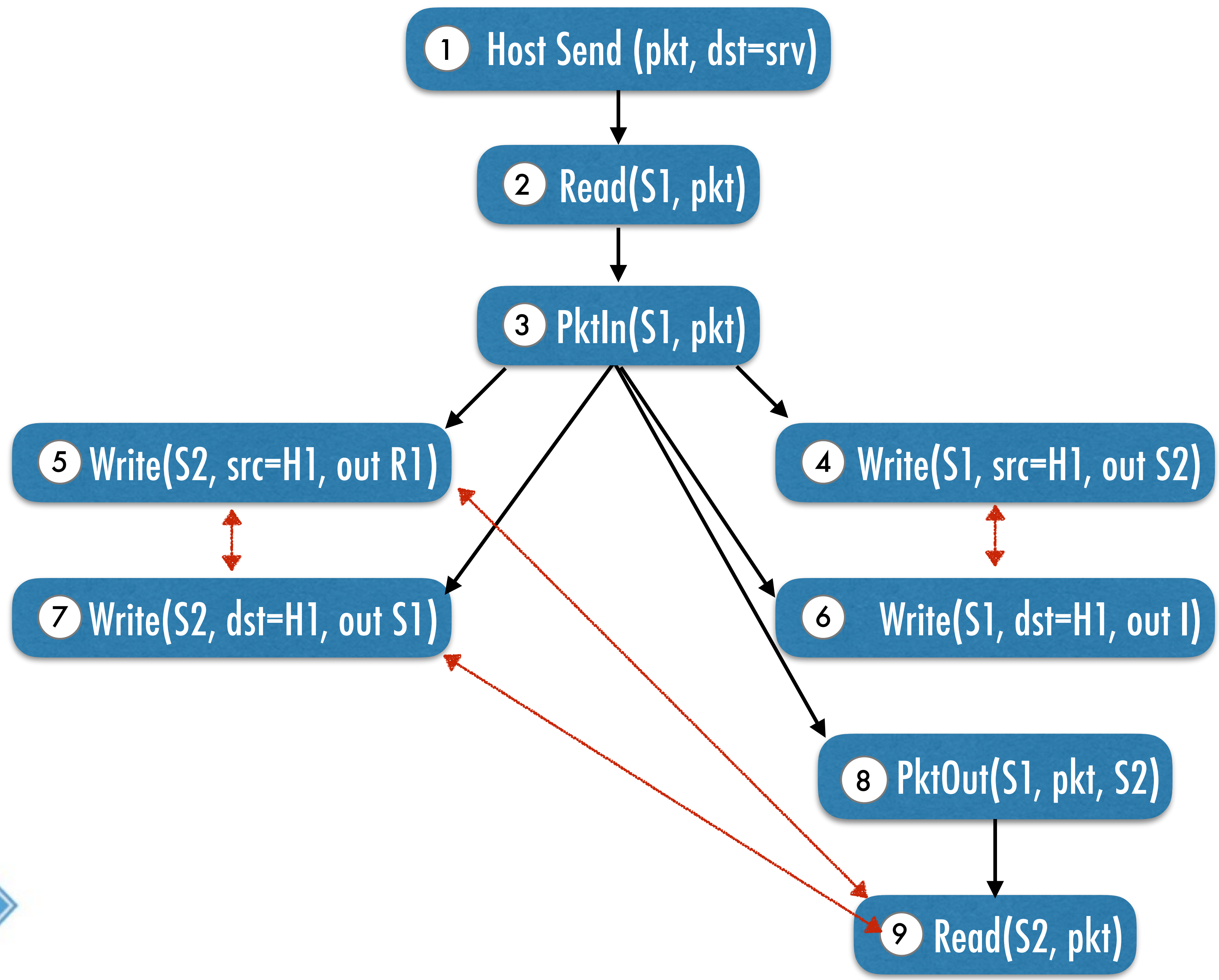
$$\varphi^{mod(e_1, strict_1)}_{mod(e_2, strict_2)} := \begin{cases} \neg(e_1.m \cap e_2.m \neq \emptyset \wedge e_1.a \neq e_2.a) & \text{if } \neg strict_1 \wedge \neg strict_2 \\ \neg(e_1.m = e_2.m \wedge e_1.p = e_2.p \wedge e_1.a \neq e_2.a) & \text{if } strict_1 \wedge strict_2 \\ \neg((e_1 \overset{strict_2}{\subseteq} e_2 \vee e_2 \overset{strict_1}{\subseteq} e_1) \wedge e_1.a \neq e_2.a) & \text{otherwise} \end{cases}$$

$$\varphi^{add(e_{add}, no\_overlap)}_{mod(e_{mod}, strict)} := \begin{cases} \neg(e_{add} \overset{strict}{\subseteq} e_{mod} \wedge e_{add}.a \neq e_{mod}.a) & \text{if } \neg no\_overlap \\ \neg(e_{add} \cap e_{mod} \neq \emptyset) & \text{otherwise} \end{cases}$$

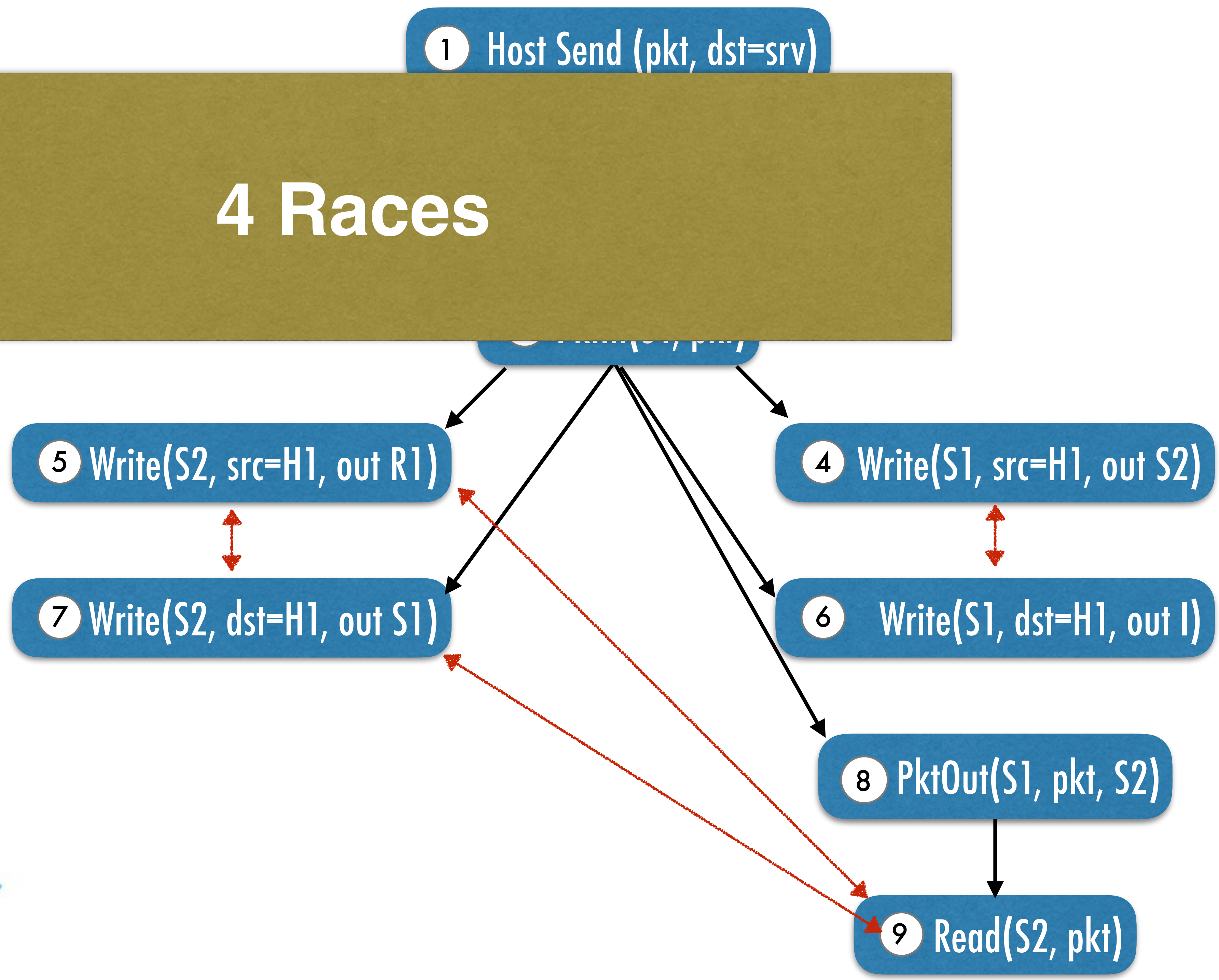
$$\varphi^{add(e_1, no\_overlap_1)}_{add(e_2, no\_overlap_2)} := \begin{cases} \neg(e_1.m \cap e_2.m \neq \emptyset \wedge e_1.p = e_2.p) & \text{if } no\_overlap_1 \vee no\_overlap_2 \\ \neg(e_1.m = e_2.m \wedge e_1.p = e_2.p \wedge e_1.a \neq e_2.a) & \text{otherwise} \end{cases}$$

Figure 3: Commutativity specification of an OpenFlow switch. Two *read* or two *del* operations always commute.

Without Commutativity

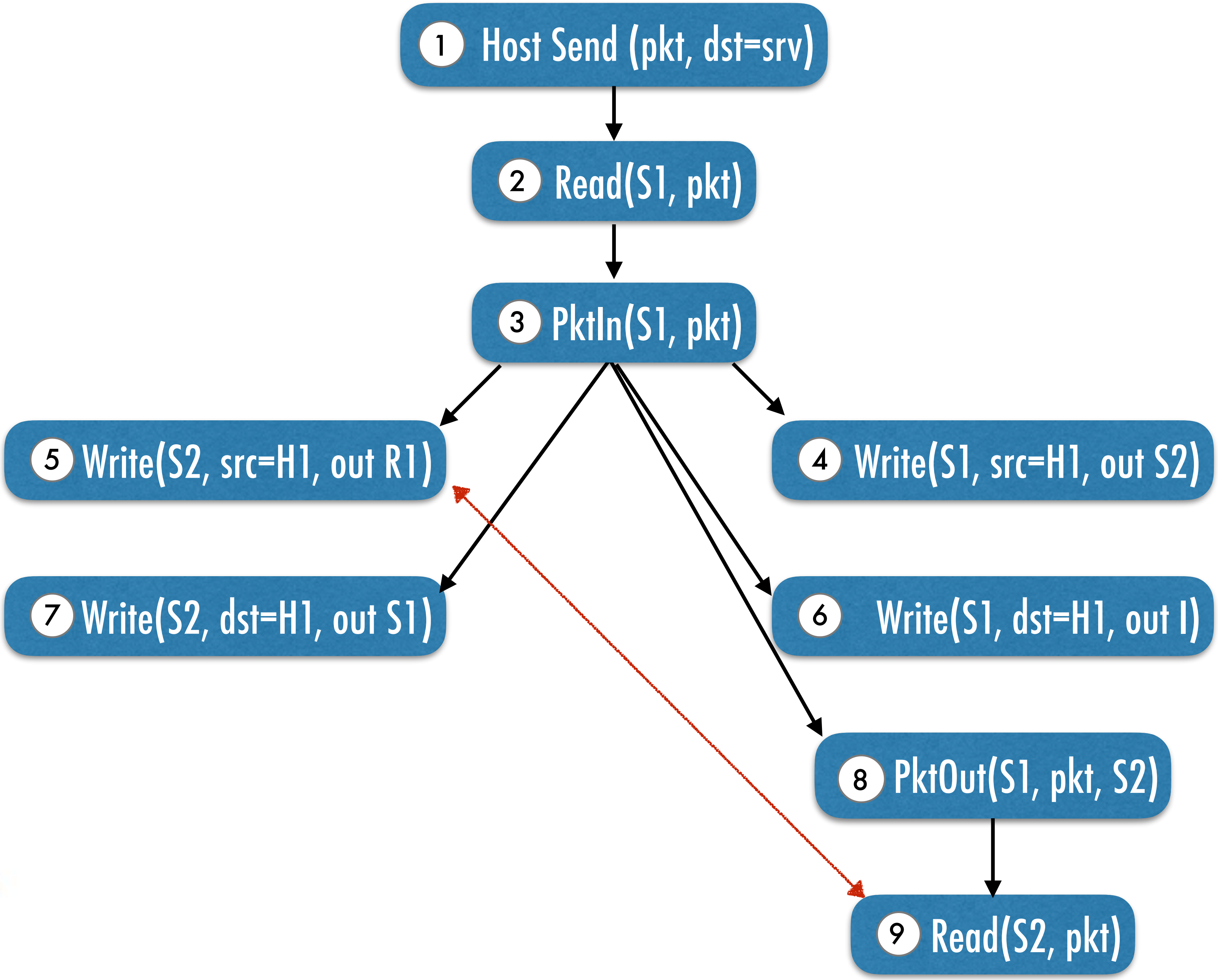


Without Commutativity



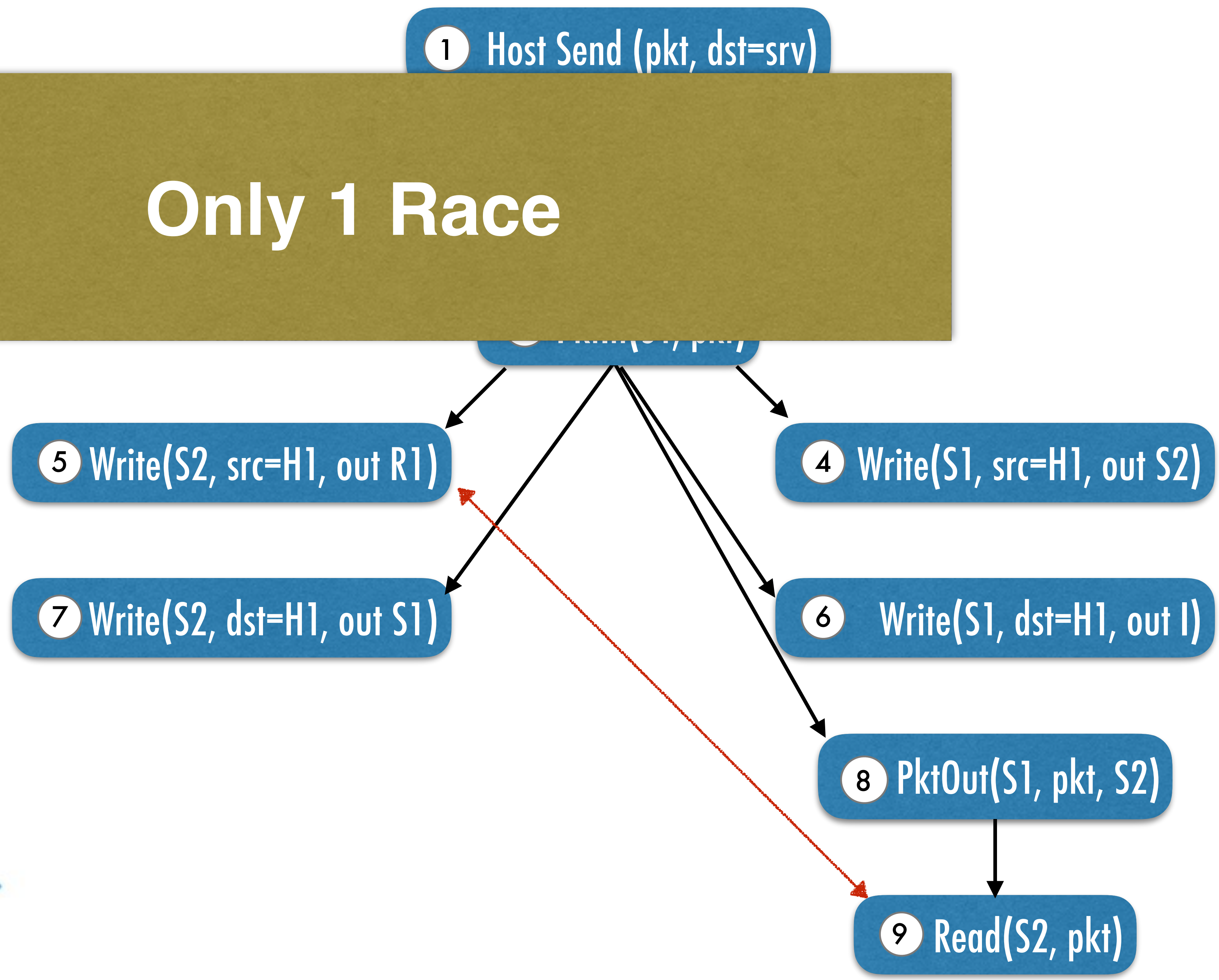
4 Races

Using Commutativity



Using Commutativity

Only 1 Race



Effect of Commutativity

For Floodlight Load Balancer

703,864 Races

↓

18,706 Races

Effect of Commutativity

For Floodlight Load Balancer

703,864 Races

18,706 Races

What about the remaining 18,706 races???

Infeasible races

Write(S1, pkt, out Replica 1)

Infeasible races

Write(S1, pkt, out Replica 1)

Infeasible races

Write(S1, pkt, out Replica 1)

Read(S1, pkt)

Time-Based Filter

Add HB edges between events that are cannot be be reordered due physical limits of the network.

Overall reduction

For Floodlight Load Balancer

703,864 Races

Commutativity filter

18,706 Races

Time-based filter

2214 (0.31%) Remaining

Can we leverage the race detector to check for high-level properties?

SDNRacer

Detecting concurrency violations

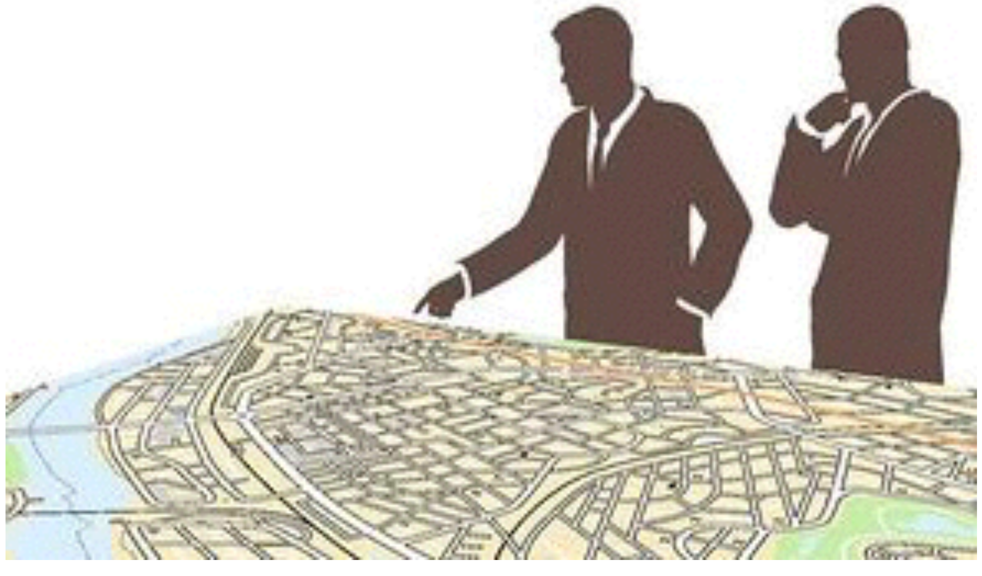
Precise notion of interference

Checks for high-level properties

Implementation and evaluation

Network Update

A set of write events that together reflects high-level network-wide policy change.



Network Update Example

Events Trace:

1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)
4 Write(S1, src=H1, out S2)
9 Read(S2, pkt)
6 Write(S1, dst=H1, out Internet)
7 Write(S2, dst=H1, out S1)
8 PktOut(S1, pkt, S2)
5 Write(S2, src=H1, out R1)
10 PktIn(S2, pkt)
11 Write(S2, src=H1, out S1)
12 Write(S1, src=H1, out S3)

Network Update Example

Events Trace:

1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. **Write(S1, src=H1, out S2)**
9. Read(S2, pkt)
6. **Write(S1, dst=H1, out Internet)**
7. **Write(S2, dst=H1, out S1)**
8. PktOut(S1, pkt, S2)
5. **Write(S2, src=H1, out R1)**
10. PktIn(S2, pkt)
11. Write(S2, src=H1, out S1)
12. Write(S1, src=H1, out S3)

Update #1

Send Traffic from H1

to Replica#1

4 Write(S1, src=H1, out S2)

5 Write(S2, src=H1, out R1)

6 Write(S1, dst=H1, out I)

7 Write(S2, dst=H1, out S1)

Network Update Example

Events Trace:

1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)
4 **Write(S1, src=H1, out S2)**
9 Read(S2, pkt)
6 **Write(S1, dst=H1, out Internet)**
7 **Write(S2, dst=H1, out S1)**
8 PktOut(S1, pkt, S2)
5 **Write(S2, src=H1, out R1)**
10 PktIn(S2, pkt)
11 **Write(S2, src=H1, out S1)**
12 **Write(S1, src=H1, out S3)**

Update #1
Send Traffic from H1
to Replica#1

- 4 Write(S1, src=H1, out S2)
- 5 Write(S2, src=H1, out R1)
- 6 Write(S1, dst=H1, out I)
- 7 Write(S2, dst=H1, out S1)

Update #2
Send Traffic from H1
to Replica#2

- 11 Write(S2, src=H1, out S1)
- 12 Write(S1, src=H1, out S3)

Network Update Isolation Property

Network updates are isolated if they are **serializable**.

Network Update Isolation

SDNRacer checks if there are **no** data races between write events of different update sets.

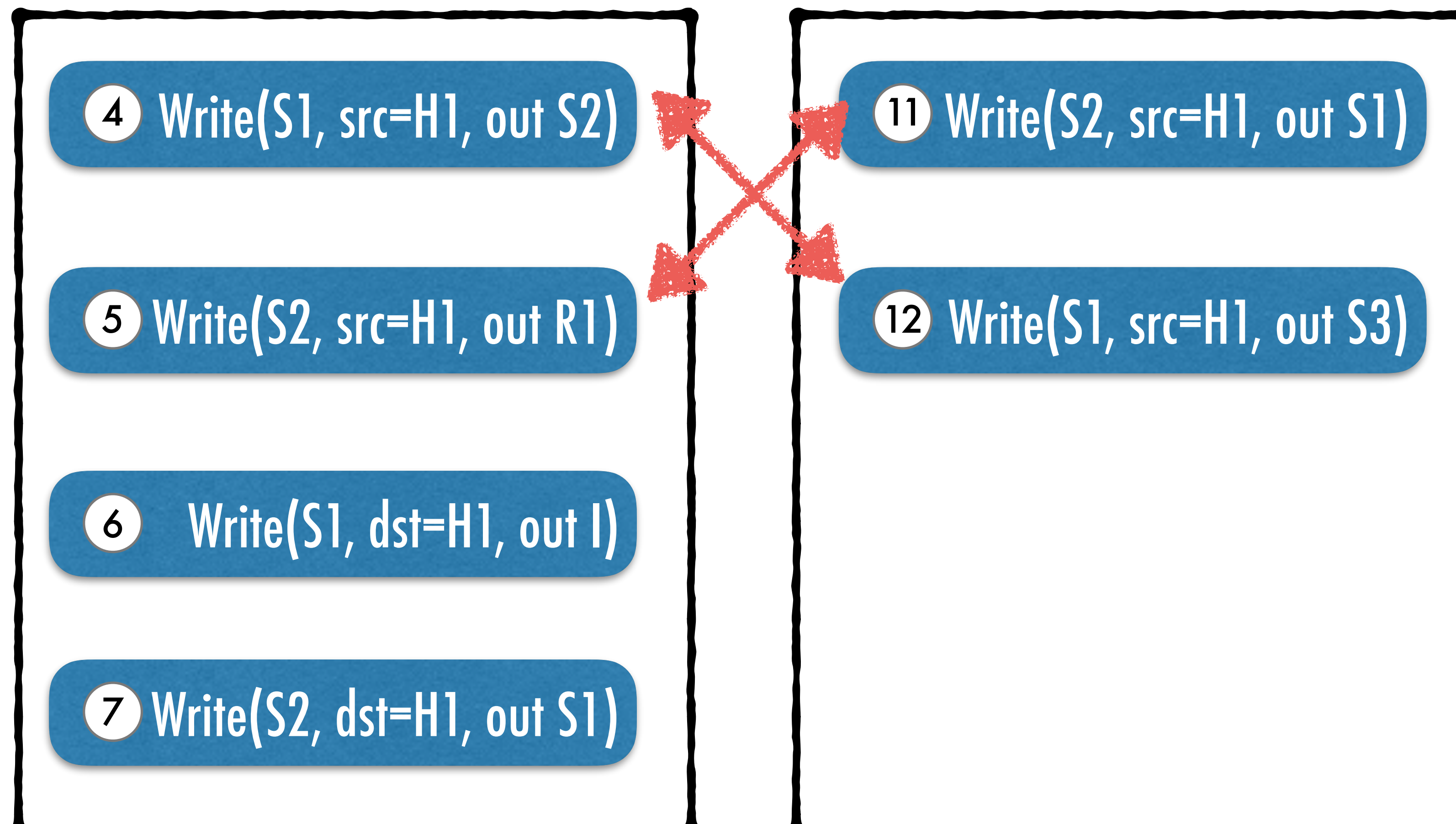
Network Update Example

Events Trace:

1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)
4 **Write(S1, src=H1, out S2)**
9 Read(S2, pkt)
6 **Write(S1, dst=H1, out Internet)**
7 **Write(S2, dst=H1, out S1)**
8 PktOut(S1, pkt, S2)
5 **Write(S2, src=H1, out R1)**
10 PktIn(S2, pkt)
11 **Write(S2, src=H1, out S1)**
12 **Write(S1, src=H1, out S3)**

Update #1
Send Traffic from H1
to Replica#1

Update #2
Send Traffic from H1
to Replica#2



Network Update Example

Events Trace:

1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)
4 **Write(S1, s**
9 Read(S2, p
6 **Write(S1, d**
7 **Write(S2, d**
8 PktOut(S1, pkt, S2)
5 **Write(S2, src=H1, out R1)**
10 PktIn(S2, pkt)
11 **Write(S2, src=H1, out S1)**
12 **Write(S1, src=H1, out S3)**

Update #1
Send Traffic from H1
to Replica#1

6 Write(S1, dst=H1, out I)
7 Write(S2, dst=H1, out S1)

Update #2
Send Traffic from H1
to Replica#2

S1)
S3)

Update #1 and Update #2 are not serializable!!!

Packet Coherence Property

A **packet** is coherent if it is processed entirely using one consistent global network configuration.

Packet Coherence Example

Events Trace:

1. Host Send (pkt, dst=srv)
2. Read(S1, pkt)
3. PktIn(S1, pkt)
4. Write(S1, src=H1, out S2)
9. Read(S2, pkt)
6. Write(S1, dst=H1, out Internet)
7. Write(S2, dst=H1, out S1)
8. PktOut(S1, pkt, S2)
5. Write(S2, src=H1, out R1)
10. PktIn(S2, pkt)
11. Write(S2, src=H1, out S1)
12. Write(S1, src=H1, out S3)

Update #1
Send Traffic from H1
to Replica#1

- 4 Write(S1, src=H1, out S2)
- 5 Write(S2, src=H1, out R1)
- 6 Write(S1, dst=H1, out I)
- 7 Write(S2, dst=H1, out S1)

Update #2
Send Traffic from H1
to Replica#2

- 11 Write(S2, src=H1, out S1)
- 12 Write(S1, src=H1, out S3)

Packet Coherence Example

Events Trace:

1 Host Send (pkt, dst=srv)
2 Read(S1, pkt)
3 PktIn(S1, pkt)
4 Write(S1, src=H1, out S2)
9 Read(S2, pkt)
6 Write(S1, dst=H1, out Internet)
7 Write(S2, dst=H1, out S1)
8 PktOut(S1, pkt, S2)
5 Write(S2, src=H1, out R1)
10 PktIn(S2, pkt)
11 Write(S2, src=H1, out S1)
12 Write(S1, src=H1, out S3)

Update #1
Send Traffic from H1
to Replica#1

Read(s1, pkt)

Update #2
Send Traffic from H1
to Replica#2

Read(s1, pkt)

Packet Coherence Check

1. Take note of the first version the packet sees.
2. Check at every switch if the read operations triggered by the packet races with write operations from different version than originally observed.

SDNRacer: Guarantees

- **SDNRacer** checks are more general than other tools which take a snapshot of the network.

- **SDNRacer** guarantees that the properties will hold for all possible reordering of the given trace.

Benefit: Fewer traces to explore

SDNRacer

Detecting concurrency violations

Precise notion of interference

Checks for high-level properties

Implementation and evaluation

Implementation

- Instrumentation of SDN troubleshooting system (STS) [SIGCOMM'14].
- Concurrency analyzer that implements happens-before rules, commutativity specification and speculative time-based filter.
- Property Checker for network update isolation and packet coherence violations.
- Around 3,000 of Python code

http://sdnracer.ethz.ch

Evaluation: Controllers

Tested with off-the-shelf controllers:

Evaluation: Applications

We tested **SDNRacer** with five different applications shipped with each controller:

- MAC-learning
- Forwarding
- Circuit Pusher
- Admission Control
- Load Balancer

Experimental Setup

- Three different network topologies: Single Switch, Two switches, and Binary Tree.
- Run every controller and every application, if possible, with randomly generated input.
- We collected 29 traces.

SDNRacer filters more than **90%** of all races in **89%** of examined traces.

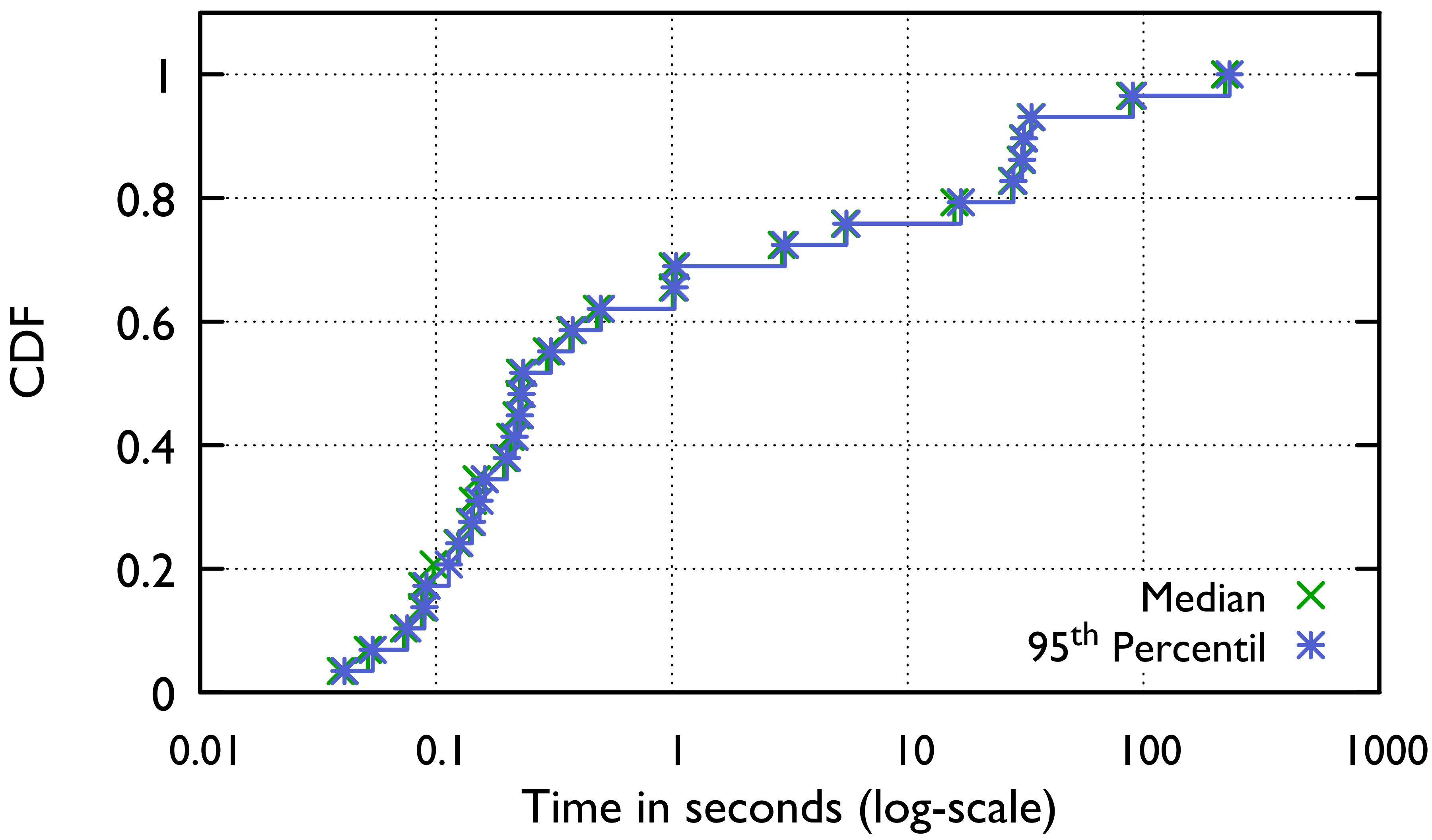
Detected Bugs

SDNRacer detected **two** update isolation violations

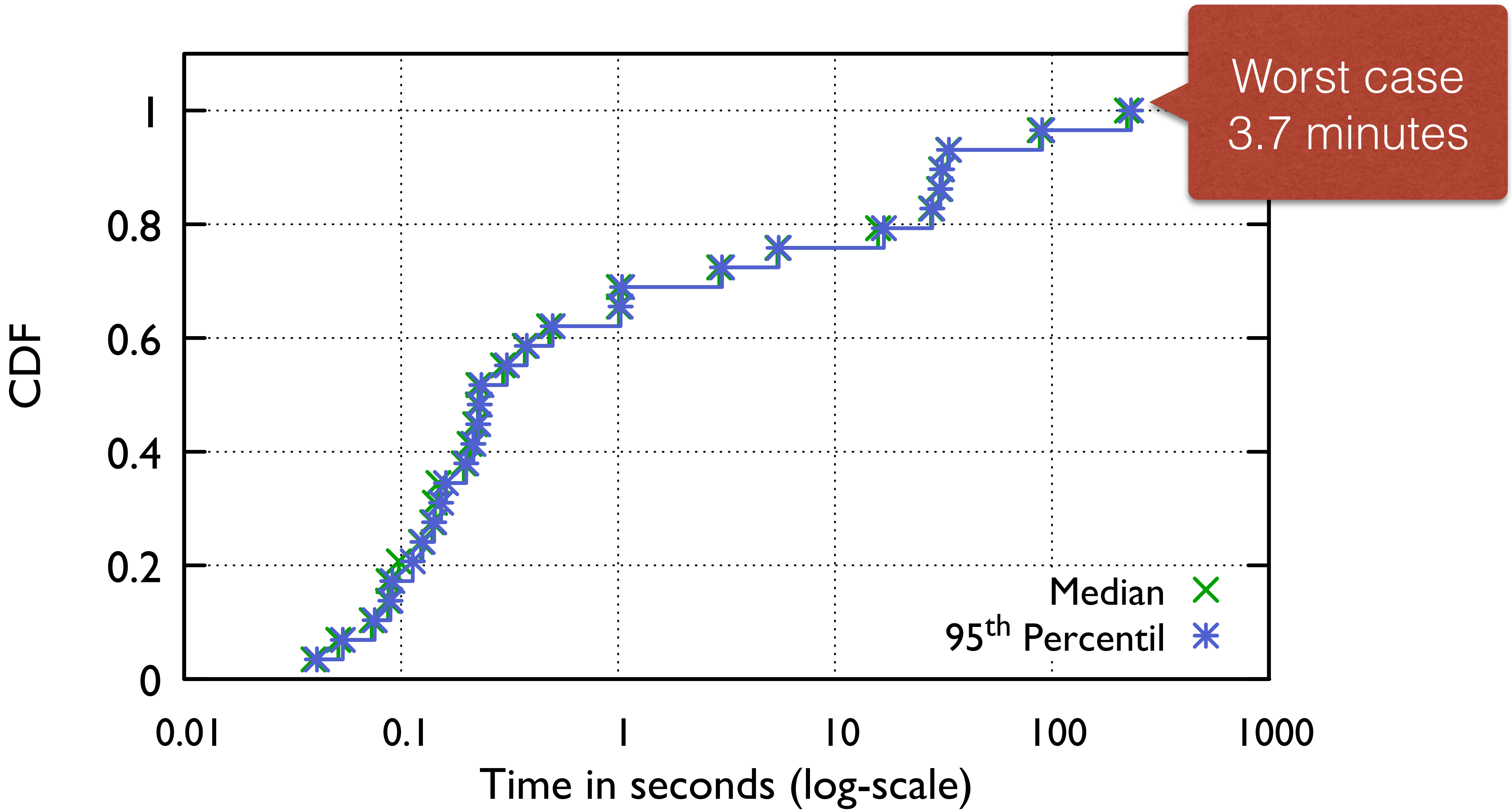
Violation #1: Floodlight Load Balancer distributes flows inconsistently

Violation #2: POX forwarding module deletes rules installed by other modules

In **90%** of the studied traces **SDNRacer** can analyze the traces in **less** than **30 seconds**.



In **90%** of the cases **SDNRacer** can analyze the traces in **less** than **30 seconds**.



Conclusion

Happens-Before Model for SDN

Captures asynchrony of SDN

Flow Table Commutativity Spec

Captures Interference

Concurrency Analysis

- Race Freedom
- Network Update Isolation
- Packet Coherence

Implementation and Evaluation

- Found bugs in existing apps: ONOS, POX, Floodlight

http://sdnracer.ethz.ch