Splendid Isolation

A Slice Abstraction for Software Defined Networks

Cole Schlesinger

PRINCETON UNIVERSITY

HotSDN

Aug. 2012

Joint work with:

Stephen Gutz

Alec Story

Nate Foster

Cornell University

frenetic ≫

- How does one **read** the state of the network?

[Foster, et al. Frenetic: A Network Programming Language. ICFP '11.]

frenetic ≫

- How does one **read** the state of the network?

  [Foster, et al. Frenetic: A Network Programming Language. ICFP '11.]

- How does one **write** the state of the network?

  [Reitblatt, et al. Abstractions for Network Update. **SIGCOMM '12**]

frenetic ≫

- How does one **read** the state of the network?

  [Foster, et al. Frenetic: A Network Programming Language. ICFP '11.]

- How does one **write** the state of the network?

  [Reitblatt, et al. Abstractions for Network Update. **SIGCOMM '12**]

- How does one **define** a new (virtual) network?

  [Coming soon!]

frenetic ≫

- How does one **read** the state of the network?

  [Foster, et al. Frenetic: A Network Programming Language. ICFP '11.]

- How does one **write** the state of the network?

  [Reitblatt, et al. Abstractions for Network Update. **SIGCOMM '12**]

- How does one **define** a new (virtual) network?

  [Coming soon!]

- How does one **compose** two network programs?

  [This talk.]

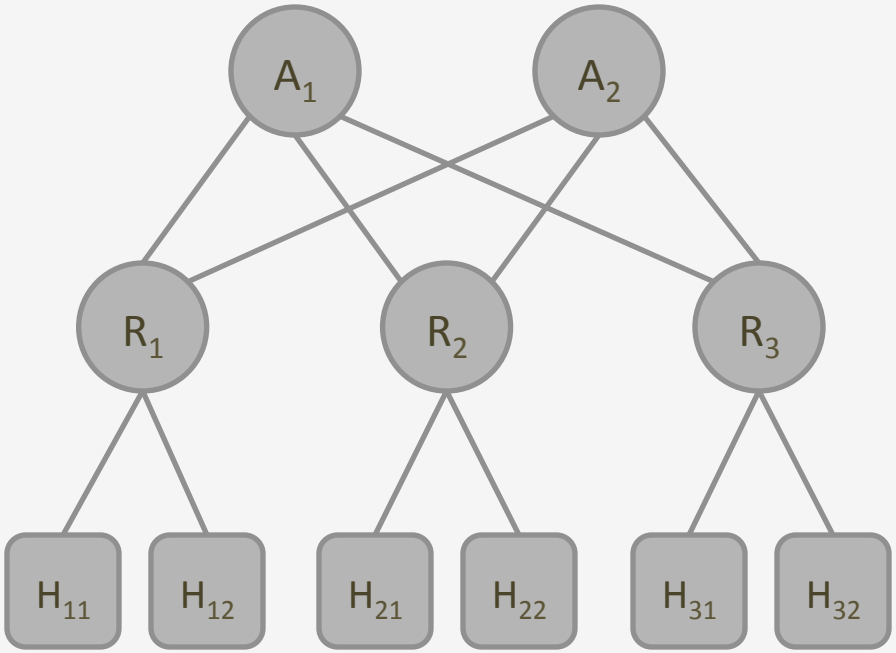
frenetic ≫

- How does one **compose** two network programs?

frenetic ≫

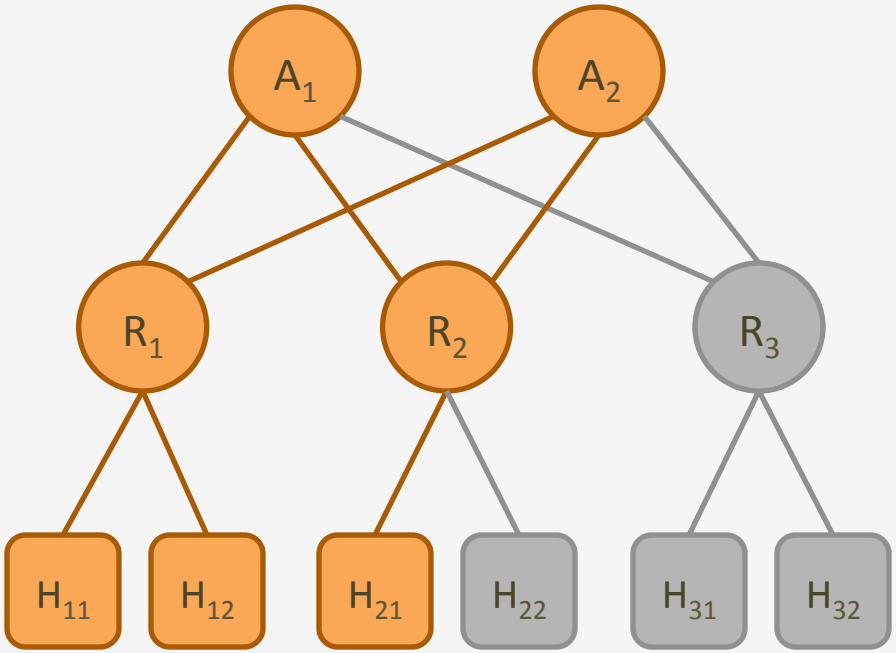
- How does one **compose** two network programs?
  - Define a new slice abstraction.
  - Lift slices (and isolation) into the language.

Data Center Isolation



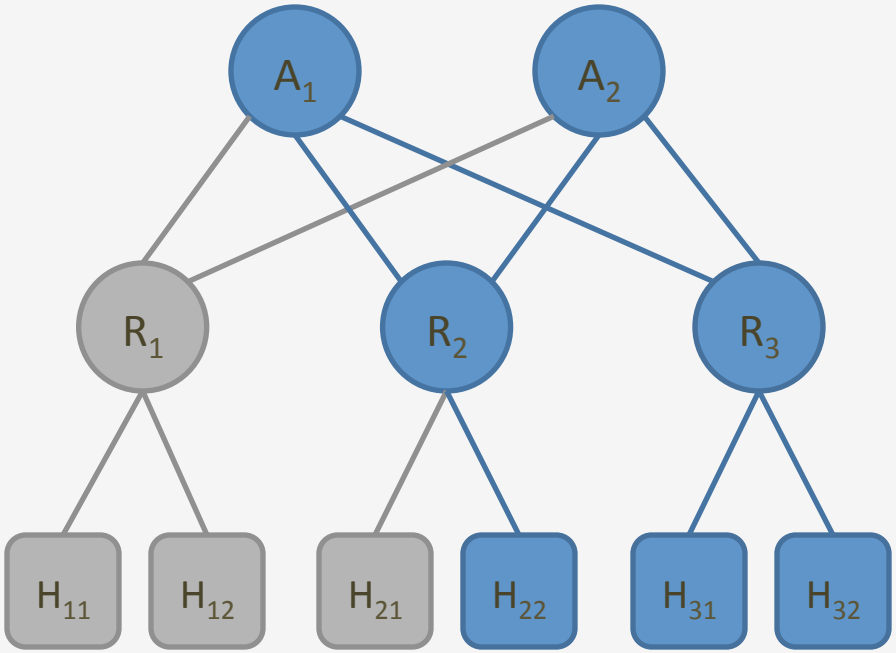
Topology

Policy 1

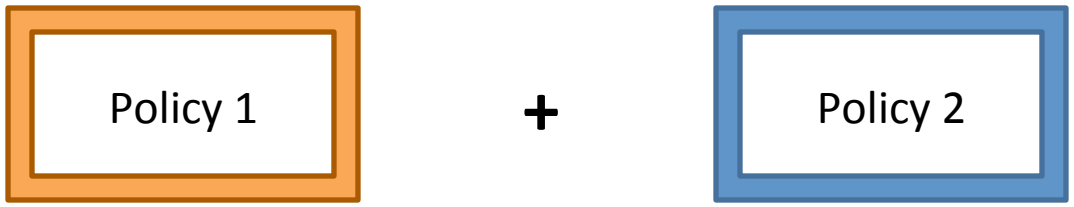


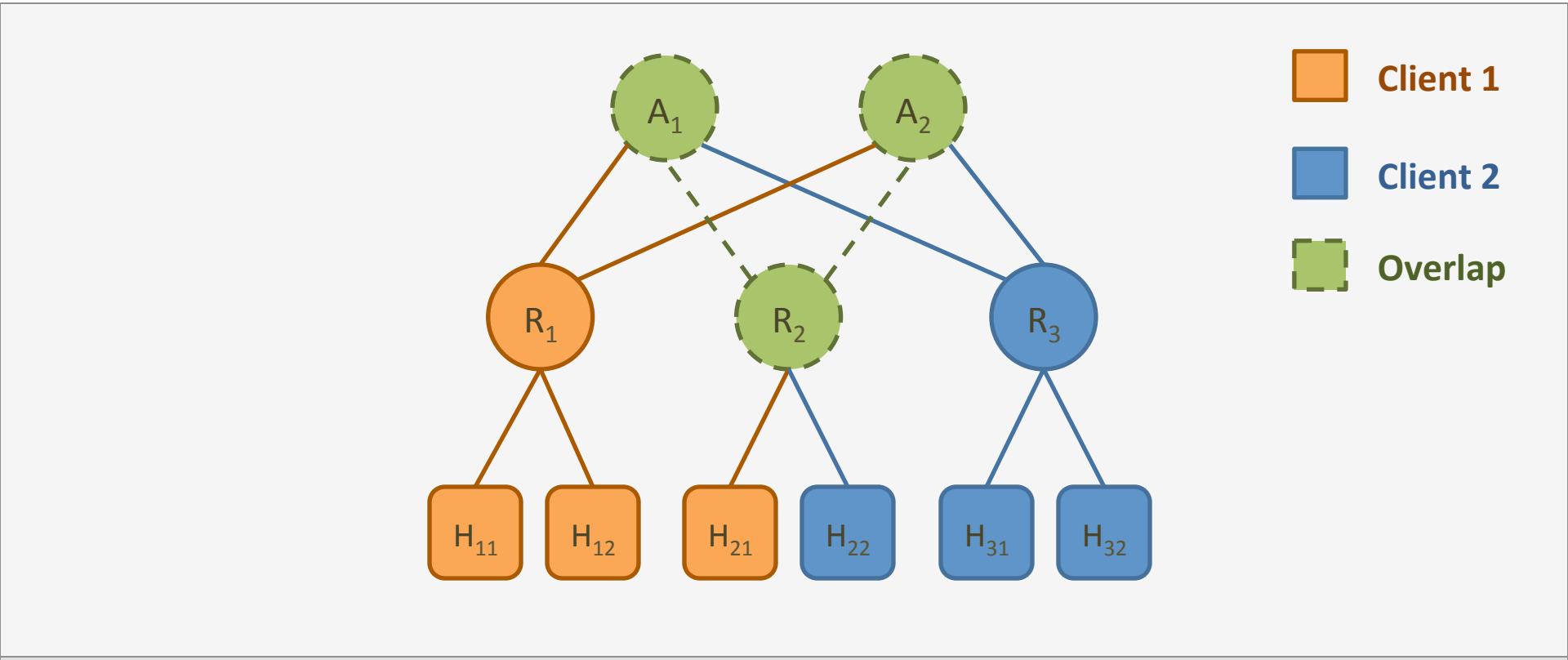
Client 1

Policy 2

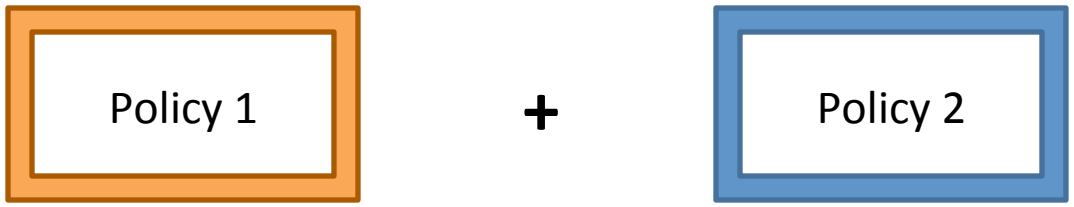


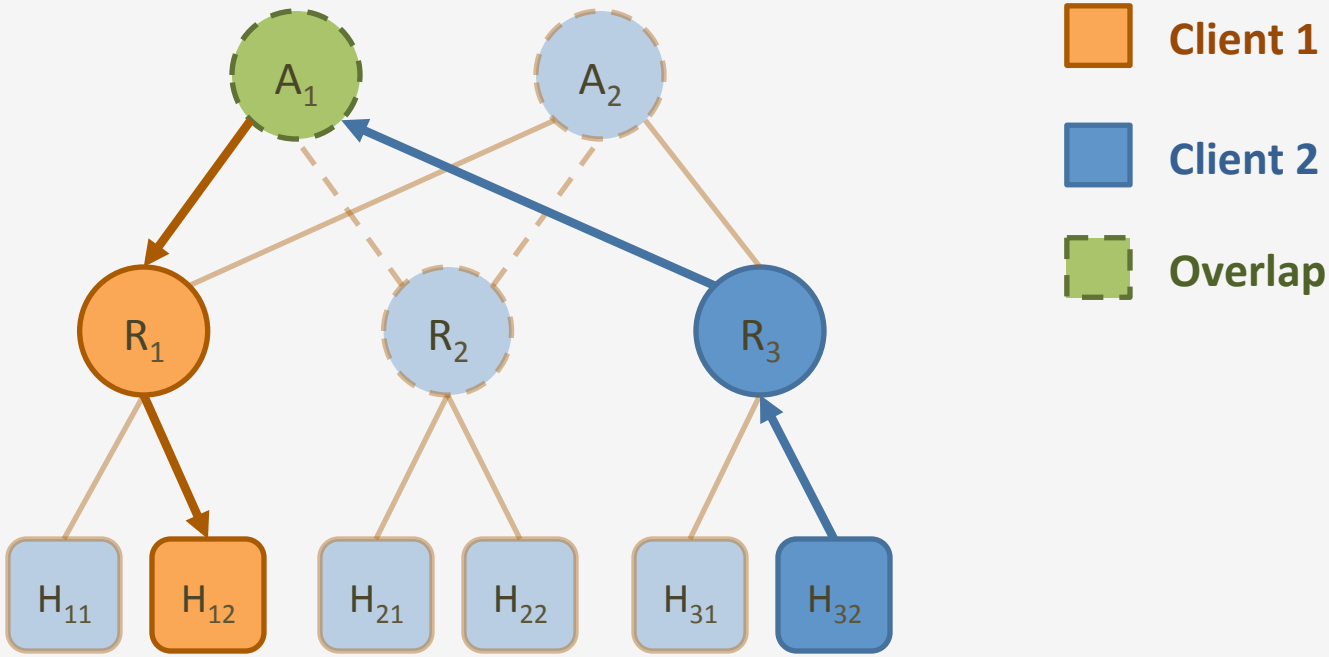
Client 2



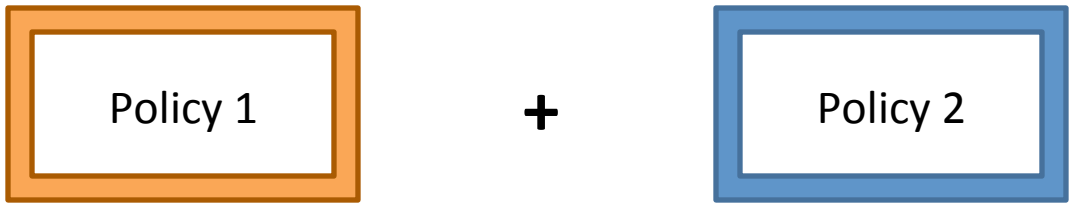


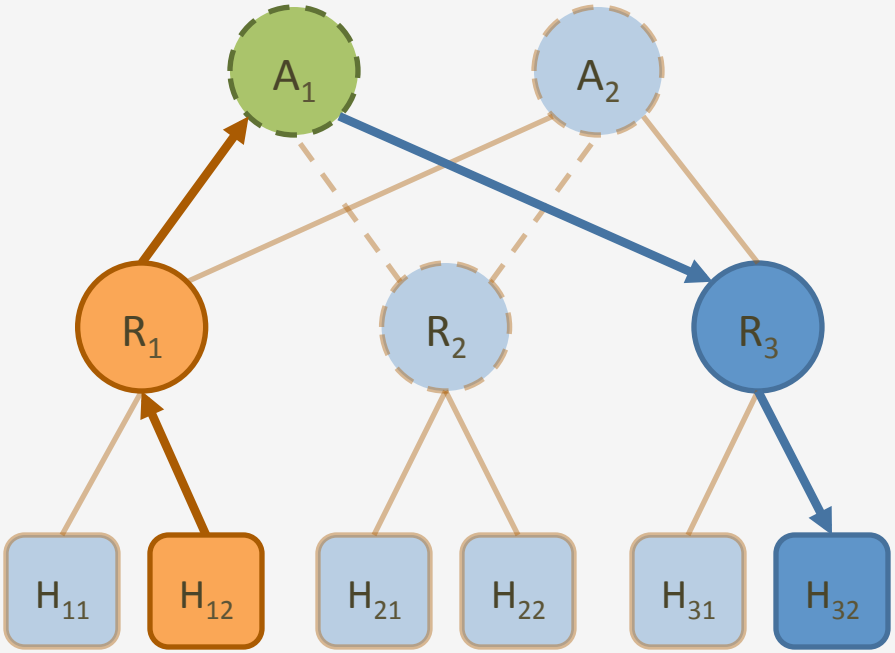
Client 1 + Client 2





Client 2 **injects packets** into Client 1's section of the network!





Client 1

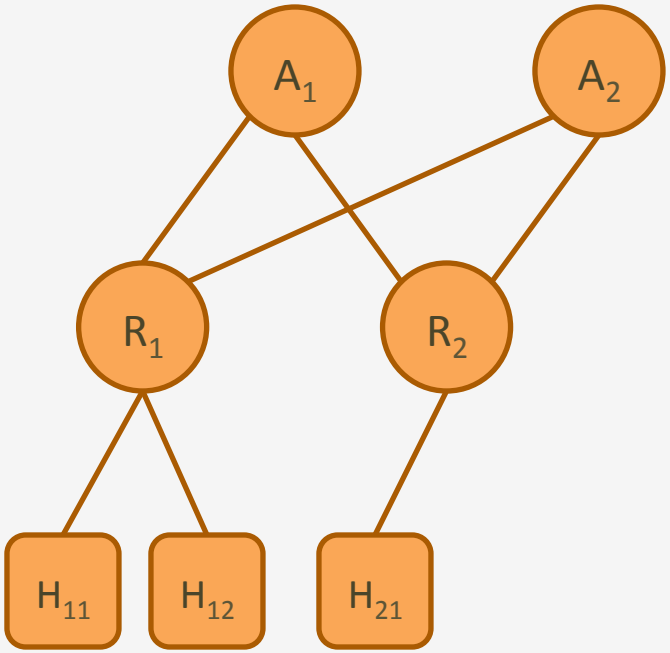
Client 2

Overlap

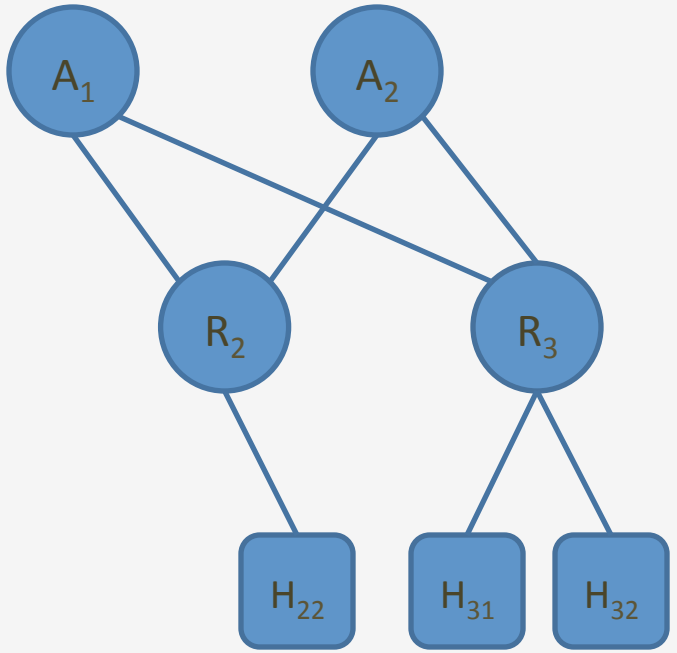
Client 2 **intercepts packets** from Client 1's section of the network!

Policy 1

Policy 2



Client 1

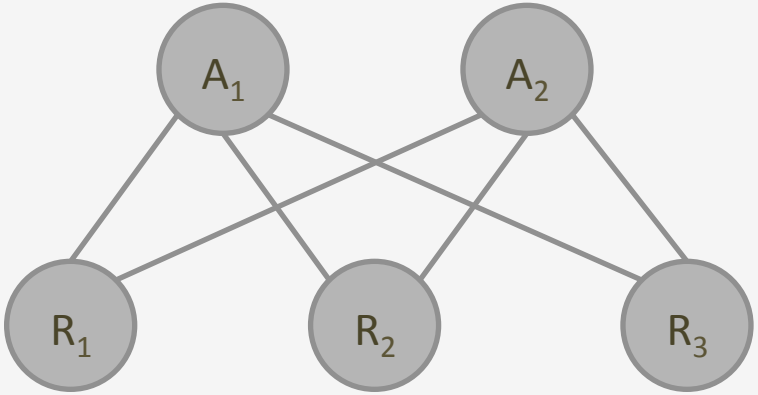


Client 2

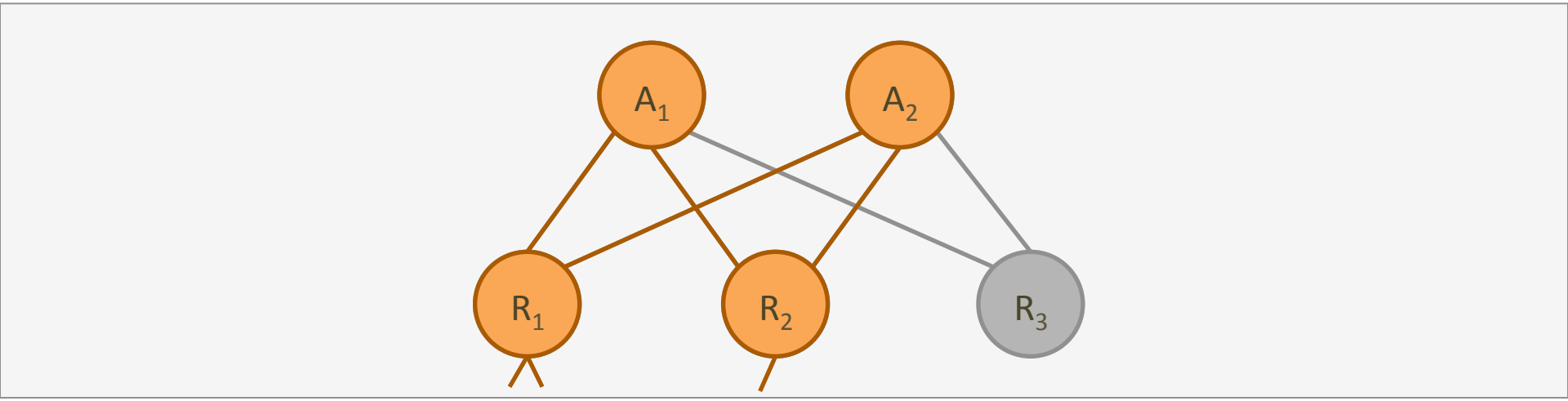
Our Approach

- Make isolation **part of the language**.
  - For *security* and *modularity*.
- Give each client a **slice** of the network which they can assume complete control over, as if they were alone on the network.
- Given a set of slices and a policy for each slice, **compile** them into one whole-network program that enforces isolation.

Slices

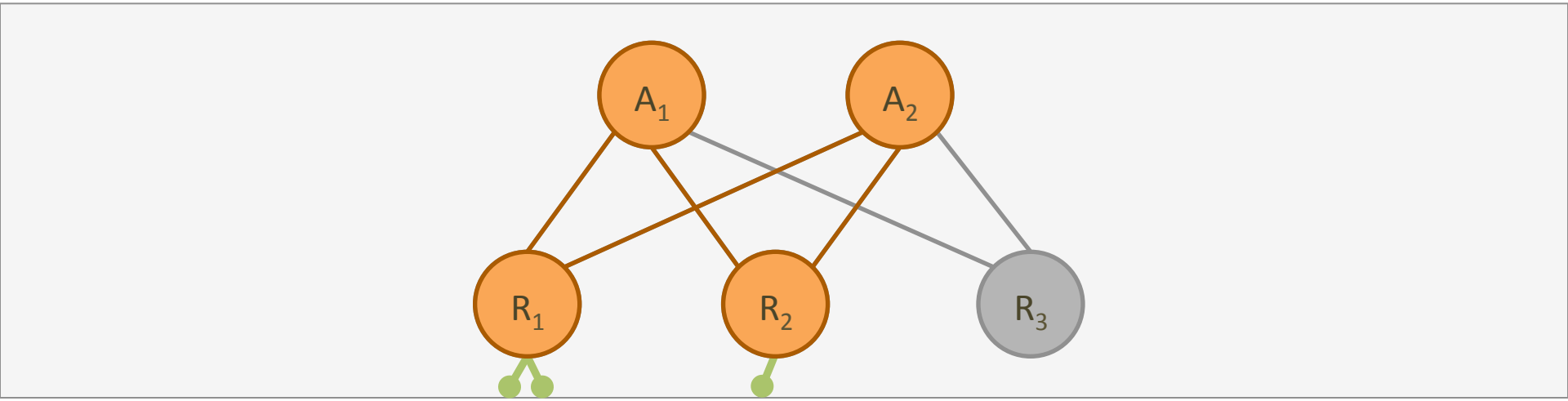


Slices




```
# topology
topo = nxtopo.NXTopo()
topo.add_switch(name="R1",ports=[1,2,3,4])
topo.add_switch(name="R2",ports=[1,3,4])
topo.add_switch(name="A1",ports=[1,2])
topo.add_switch(name="A2",ports=[1,2])
```

Slices

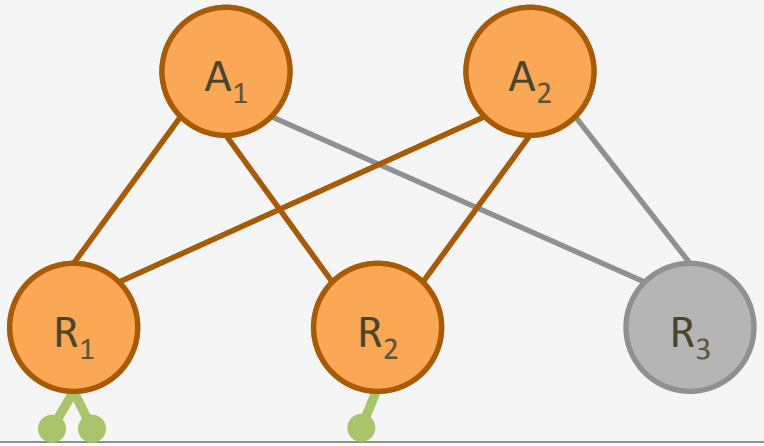



```
# slice entries and exits
edges = [ ("R1", 1, tpDst 80, tpDst 80)
        , ("R1", 2, tpDst 80, tpDst 80)
        , ("R2", 1, tpDst 80, tpDst 80) ]
```

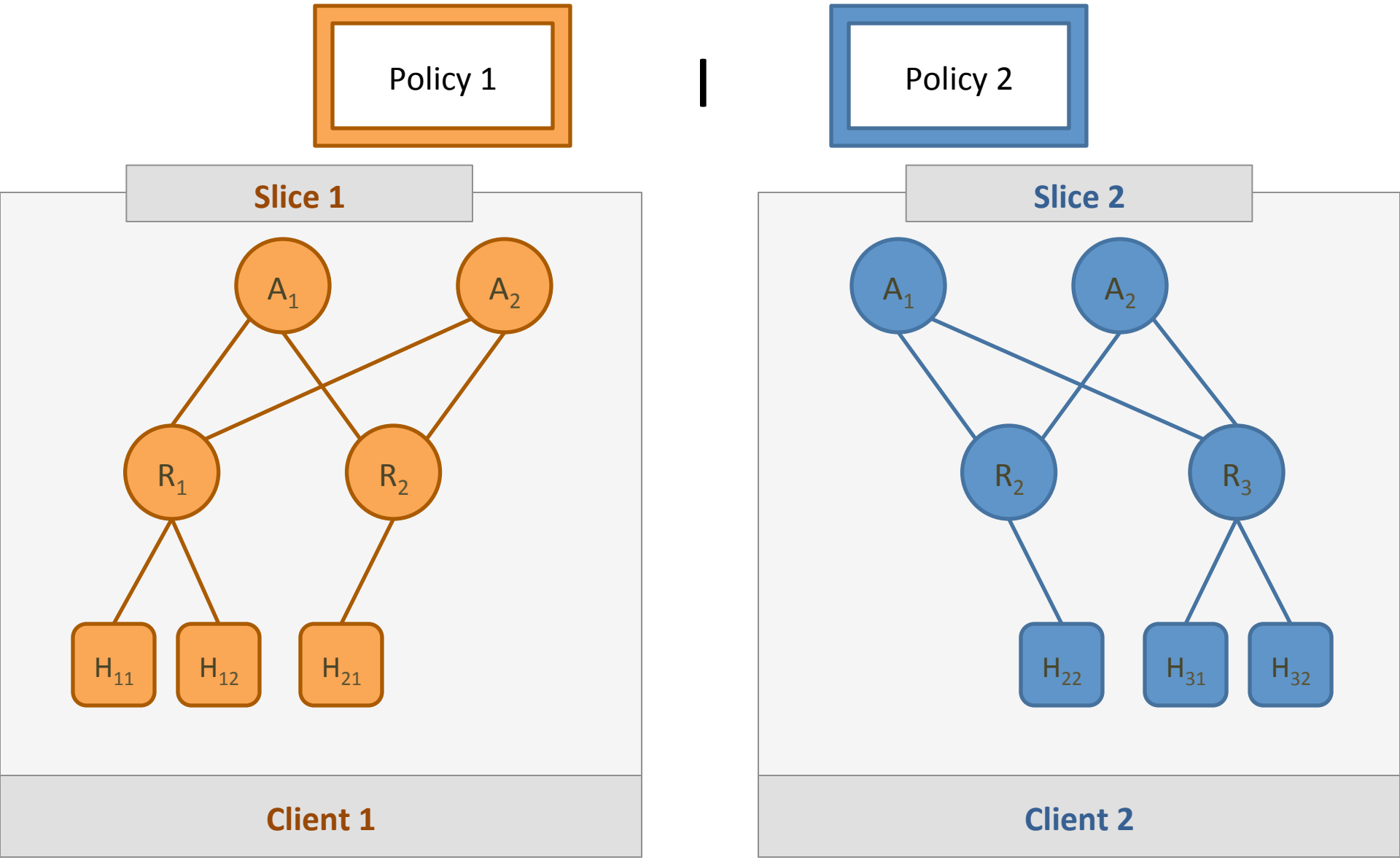
Predicate on **incoming** packets

Predicate on **outgoing** packets

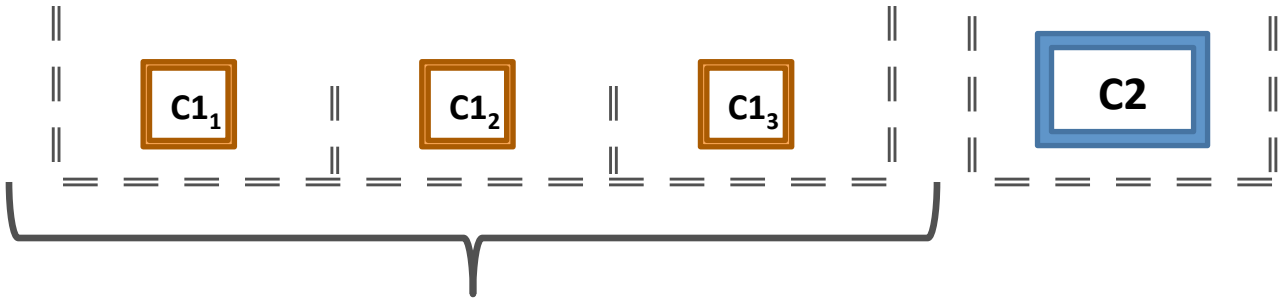
Slices



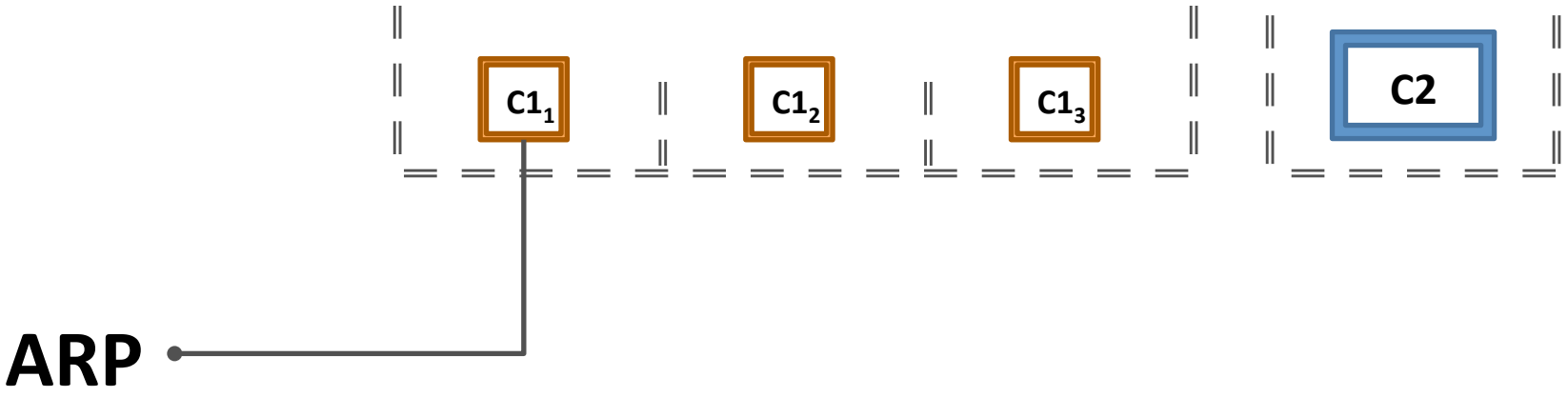

```
# slice constructor
slice = Slice(topo,phys_topo,edges)
```


Policy 1
Policy 2
Slice 1
Slice 2
A1
A2
R1
R2
H11
H12
H21
A1
A2
R2
R3
H22
H31
H32
Client 1
Client 2

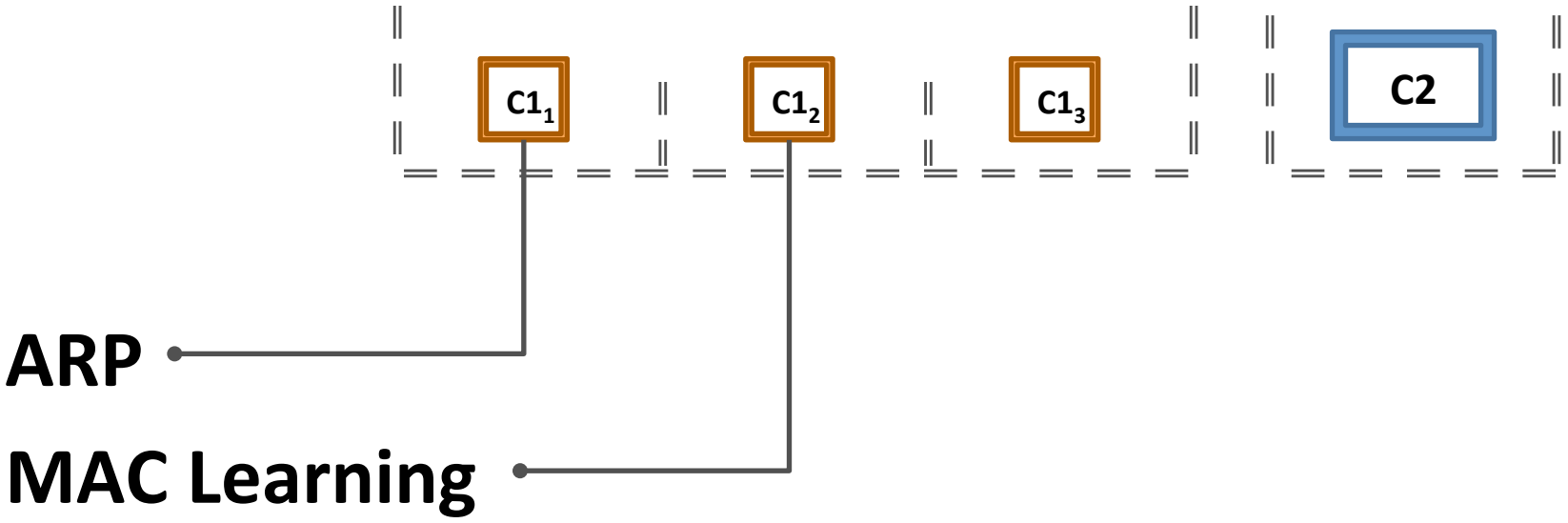
Isolation as Modularity



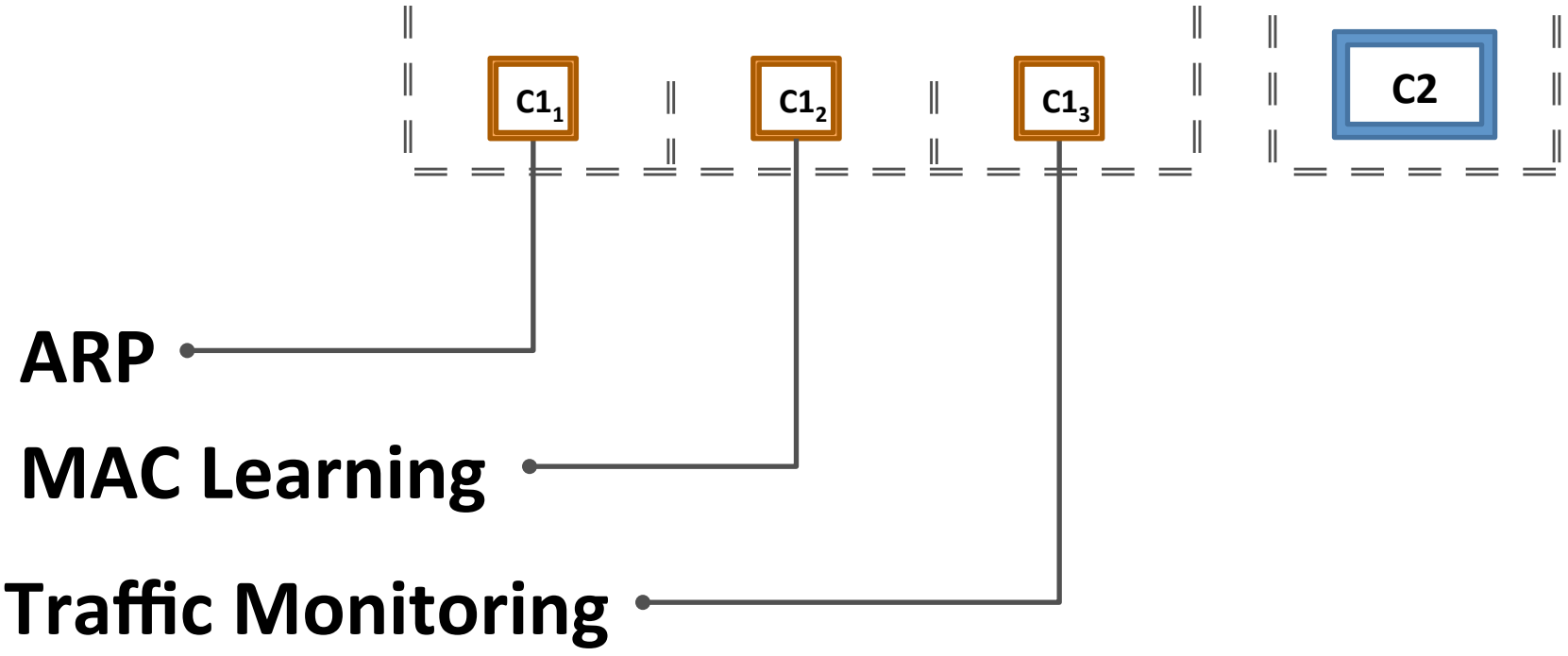
Isolation as Modularity



Isolation as Modularity



Isolation as Modularity



Implementation

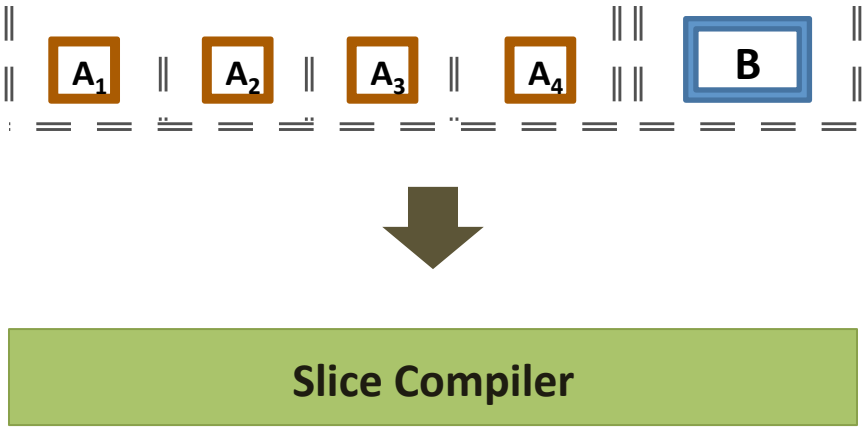
Input: a set of slices and NetCore policies.

(Must be VLAN-independent.)

Implementation

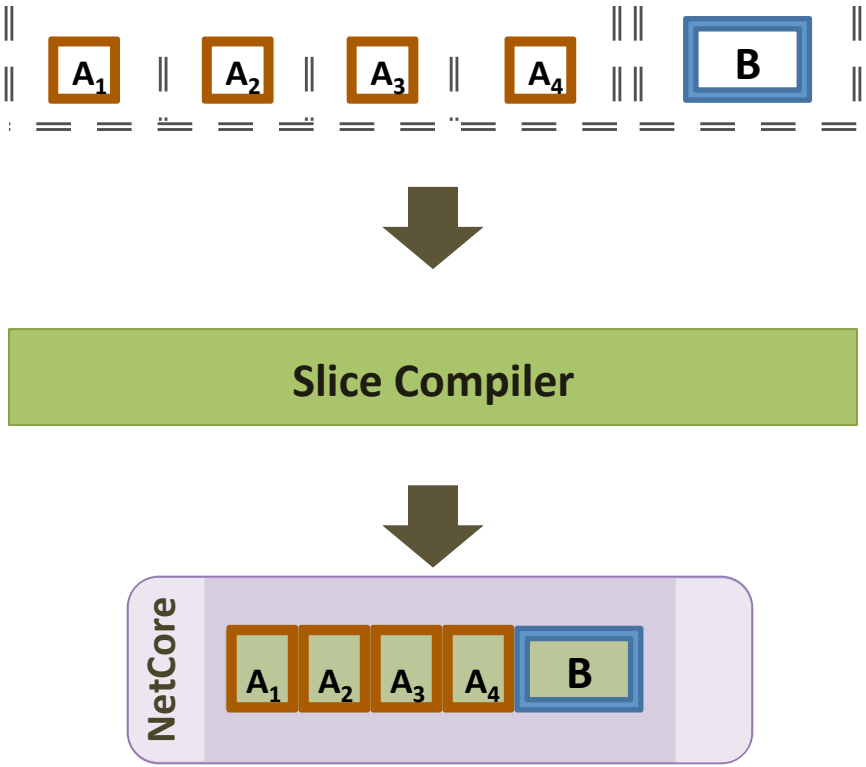
Input: a set of slices and NetCore policies.

(Must be VLAN-independent.)



Implementation

Input: a set of slices and NetCore policies.

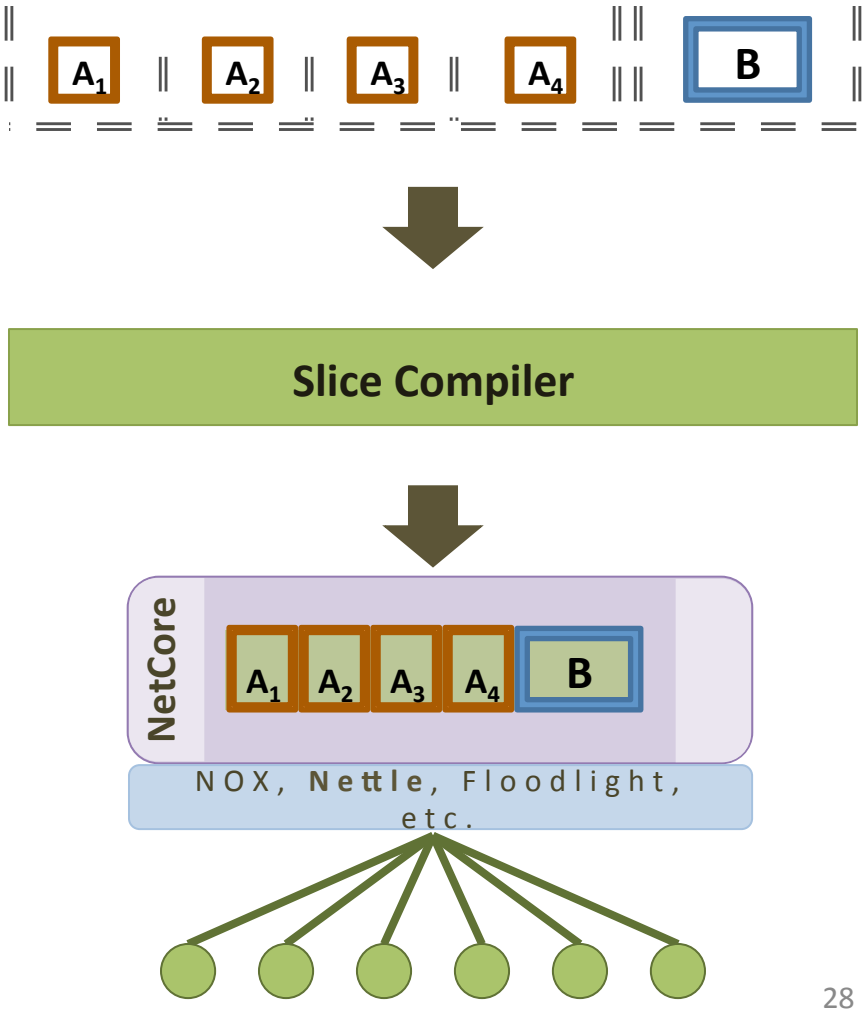


Output: a single, global NetCore policy.

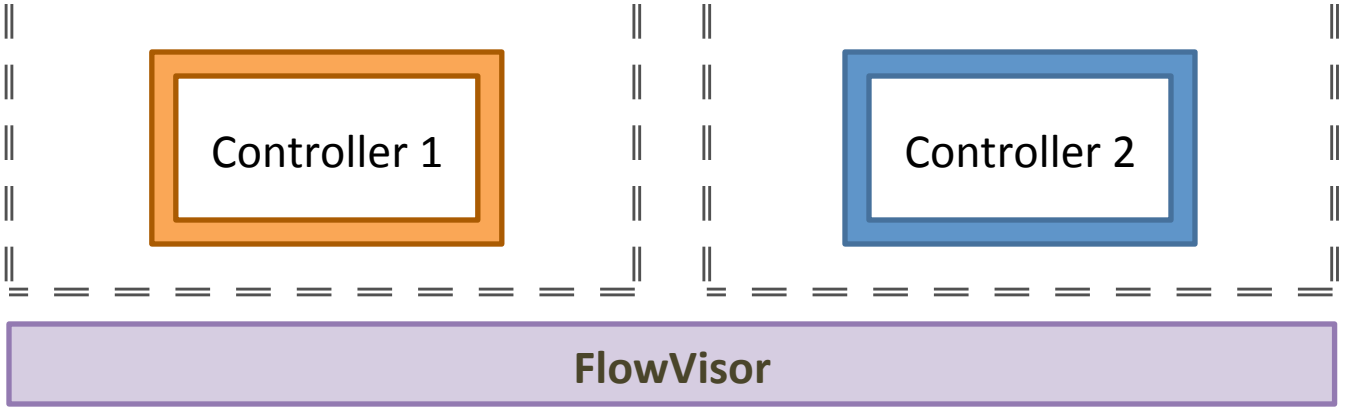
Implementation

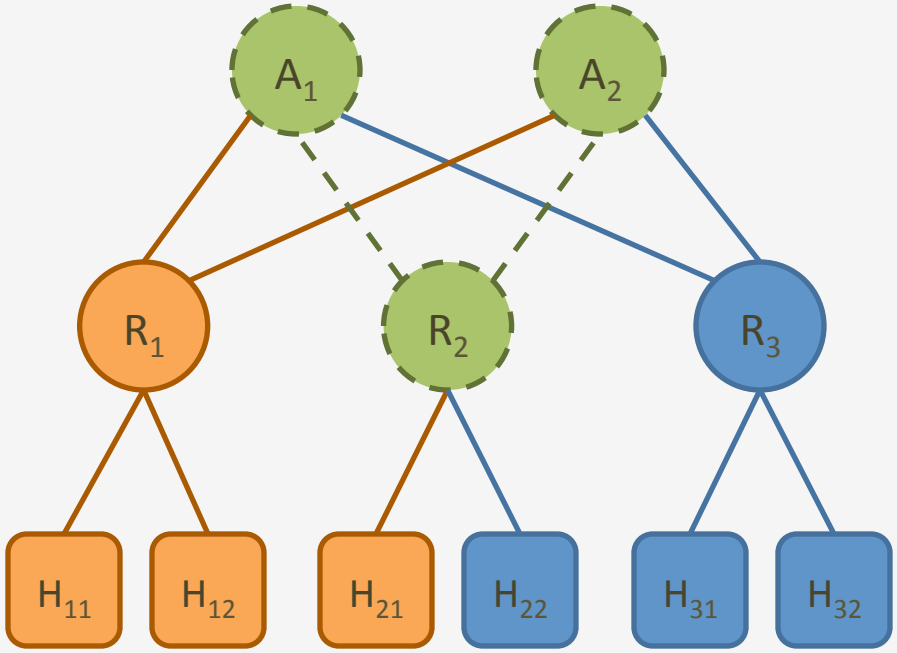
Input: a set of slices and policies.

(Must be VLAN-independent.)

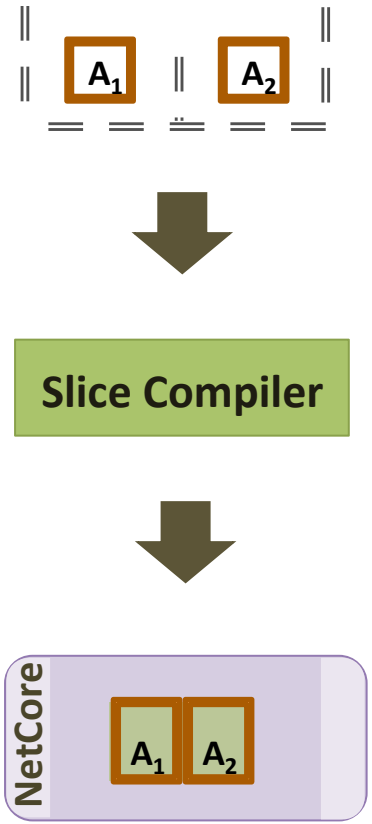


Output: a single, global NetCore policy.


Controller 1
Controller 2
FlowVisor

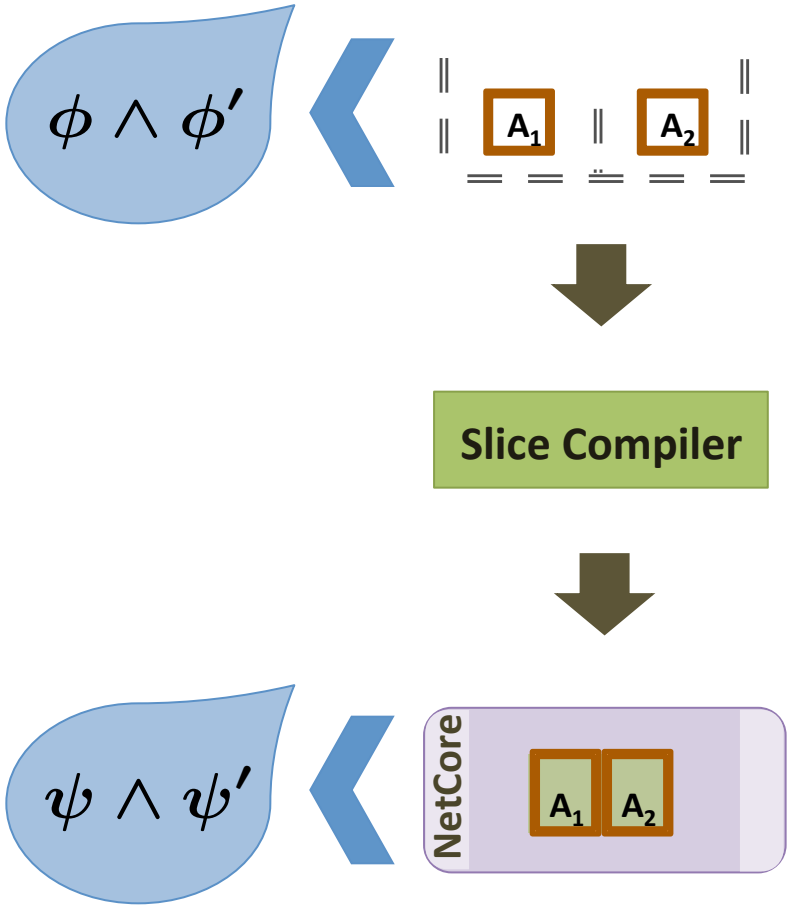

A1
A2
R1
R2
R3
H11
H12
H21
H22
H31
H32

Verification



Verification

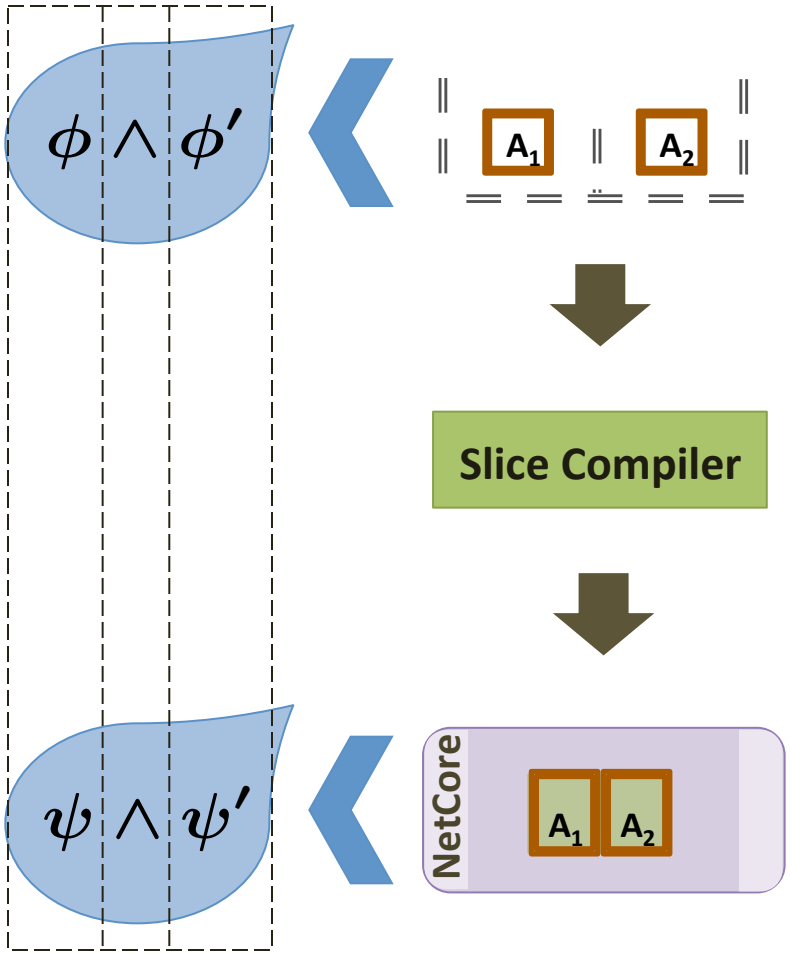
1. Model NetCore policies in SMT (Z3).



Verification

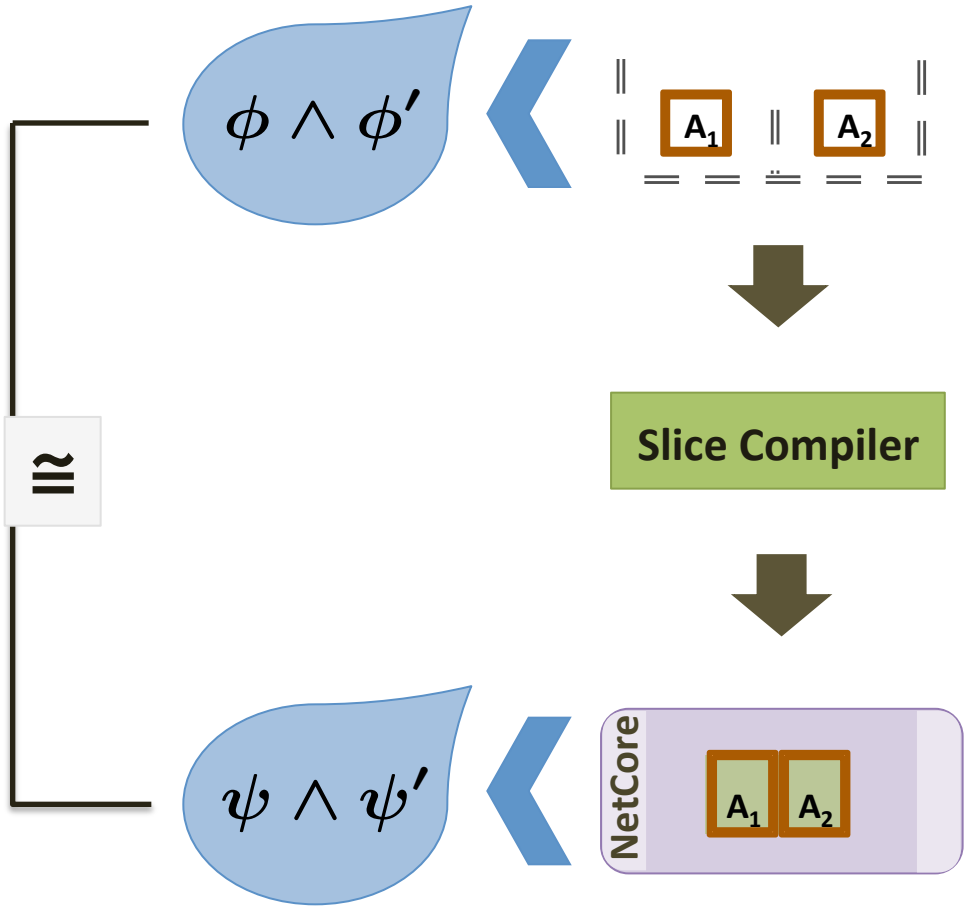
1. Model NetCore policies in SMT (Z3).

2. Verify isolation.



Verification

1. Model NetCore policies in SMT (Z3).
2. Verify isolation.
1. Verify semantic equivalence.

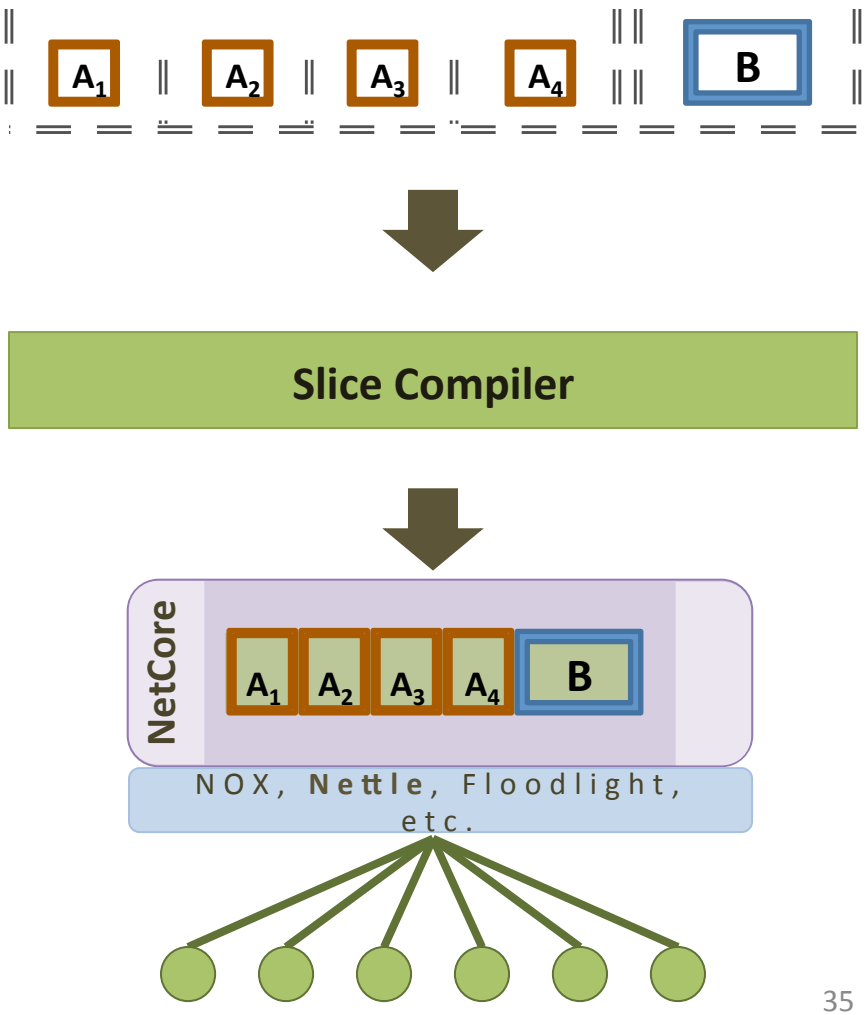


Contributions

- A **new language** for slices.

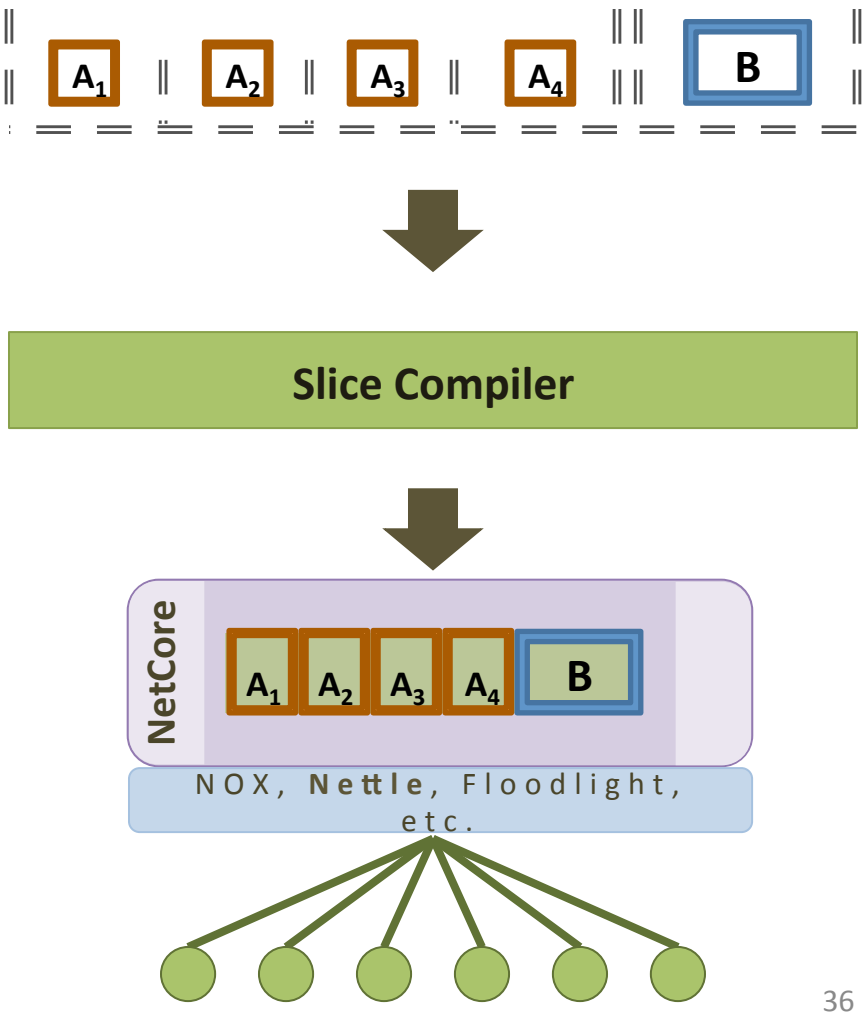
Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.



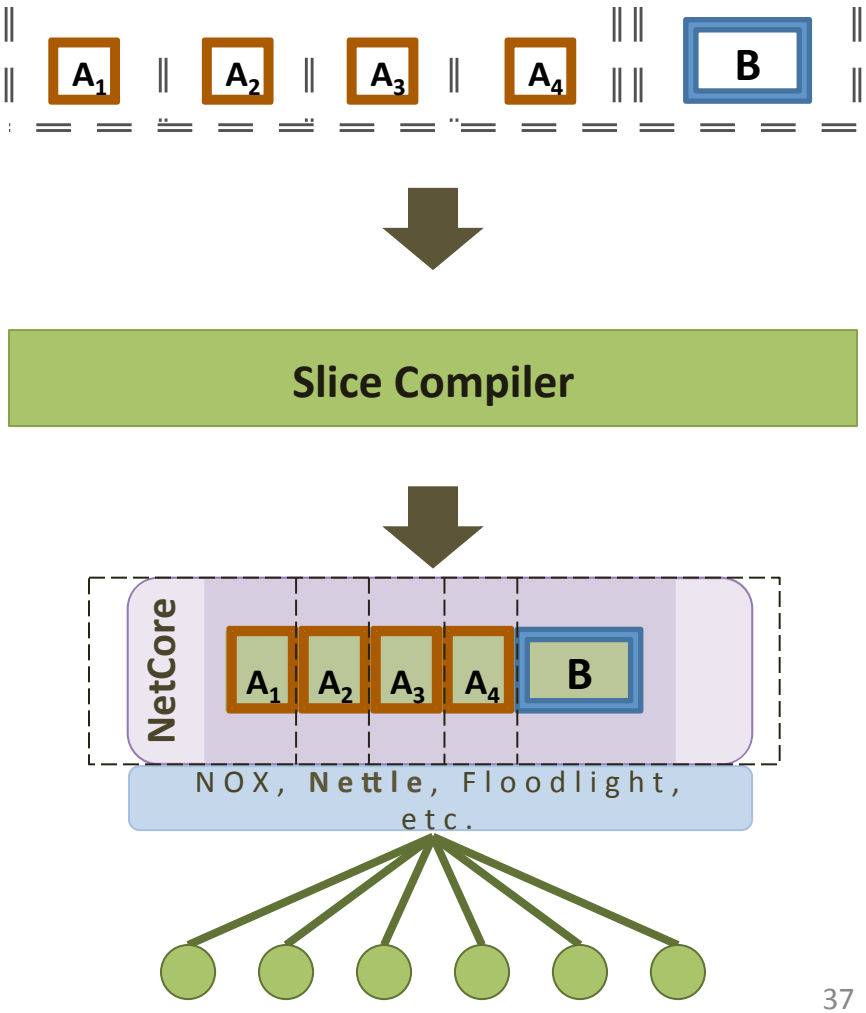
Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.
- A **verifier** that guarantees:



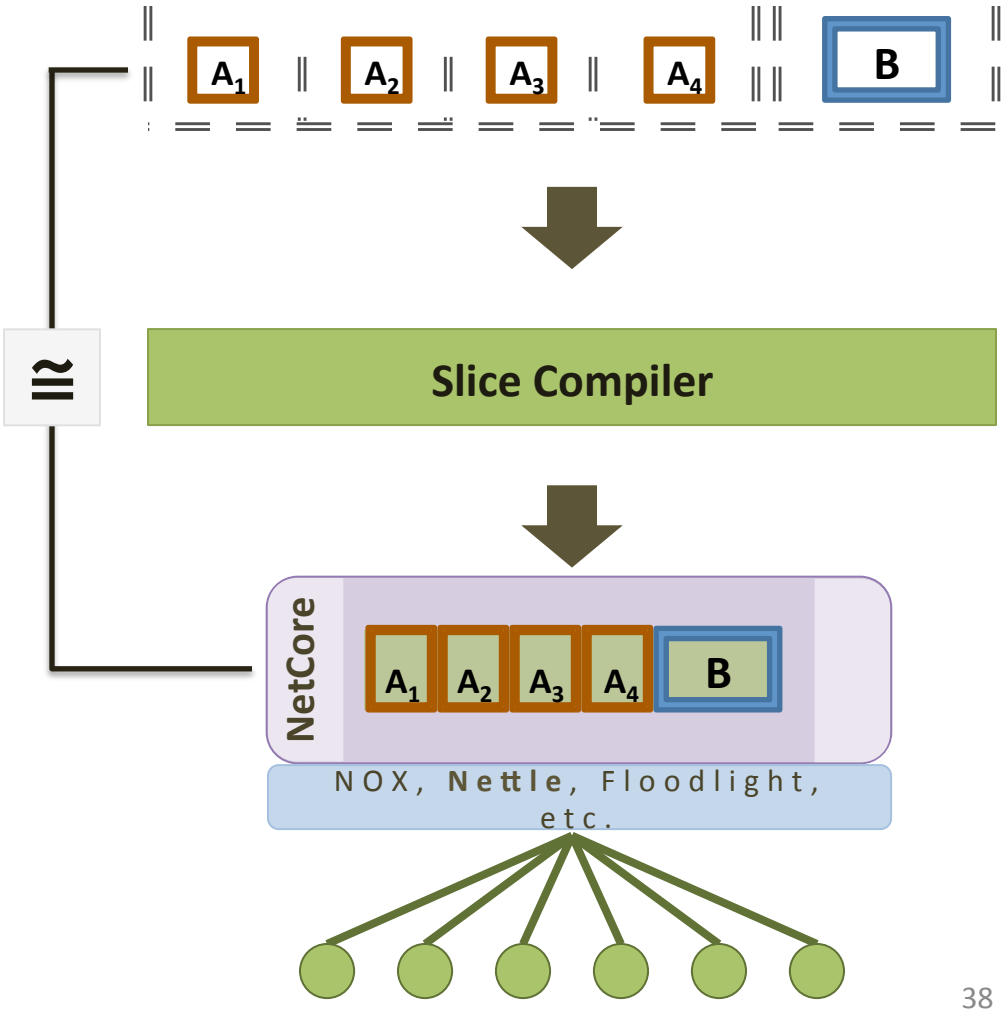
Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.
- A **verifier** that guarantees:
  - isolation



Contributions

- A **new language** for slices.
  - Security
  - Modularity
- A **compiler** that enforces isolation.
- A **verifier** that guarantees:
  - isolation, and
  - semantic equivalence.



Thank you!

Read the paper:
frenetic-lang.org/papers

Get **the code**:
github.com/frenetic-lang/netcore

See the **demo**:
Find me after the talk!

We wish to thank Shrutarshi Basu, Arjun Guha, Josh Reich, Mark Reitblatt, Jennifer Rexford, and David Walker for many helpful comments and suggestions.

THE END

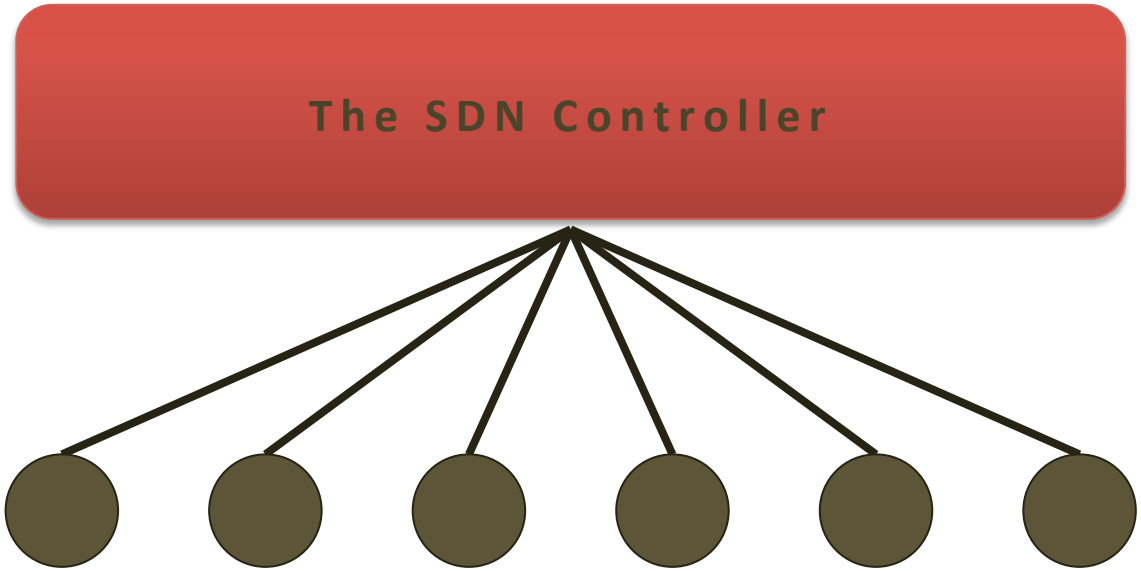
frenetic ≫

Re-imagining the fundamentals of network implementation from a programming languages point of view:

- How does one *read* the state of the network?
- How does one *write* the state of the network?
- How does one *define* a new (virtual) network?
- How does one *compose* two network programs?

Frenetic is a new programming language we are creating to explore these questions and more

One program to rule them all …



NetCore: Program Composition

Repeater

Pattern	Action
inPort = 2	Forward 1
inPort = 1	Forward 2

Monitor

Pattern	Action
tpSrc = 22	Drop
tpSrc = 80	Drop

NetCore: Program Composition

Repeater

Pattern	Action
inPort = 2	Forward 1
inPort = 1	Forward 2

Monitor

Pattern	Action
tpSrc = 22	Drop
tpSrc = 80	Drop

NetCore

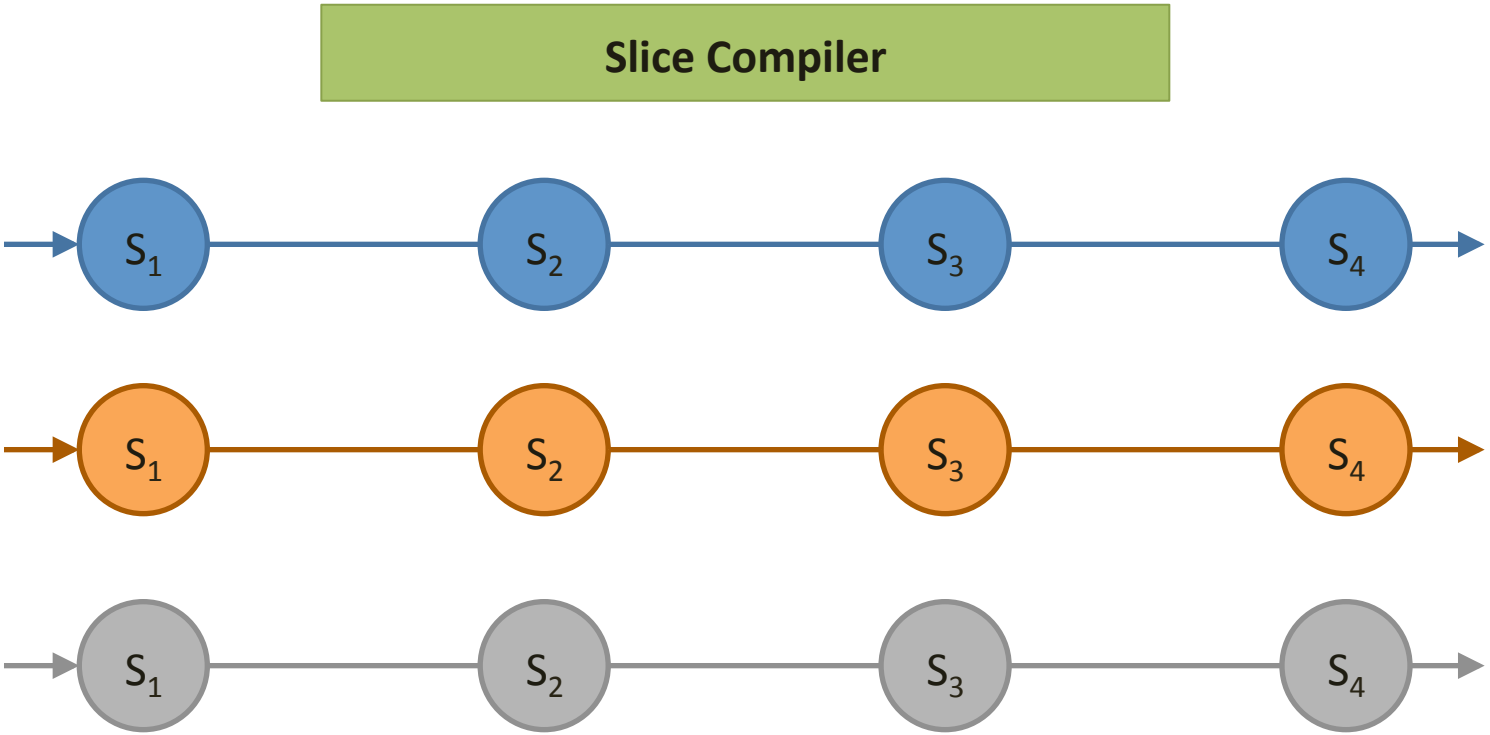
Repeater

```
inPort 2 ==> forward [1] <+>
inPort 1 ==> forward [2]
```

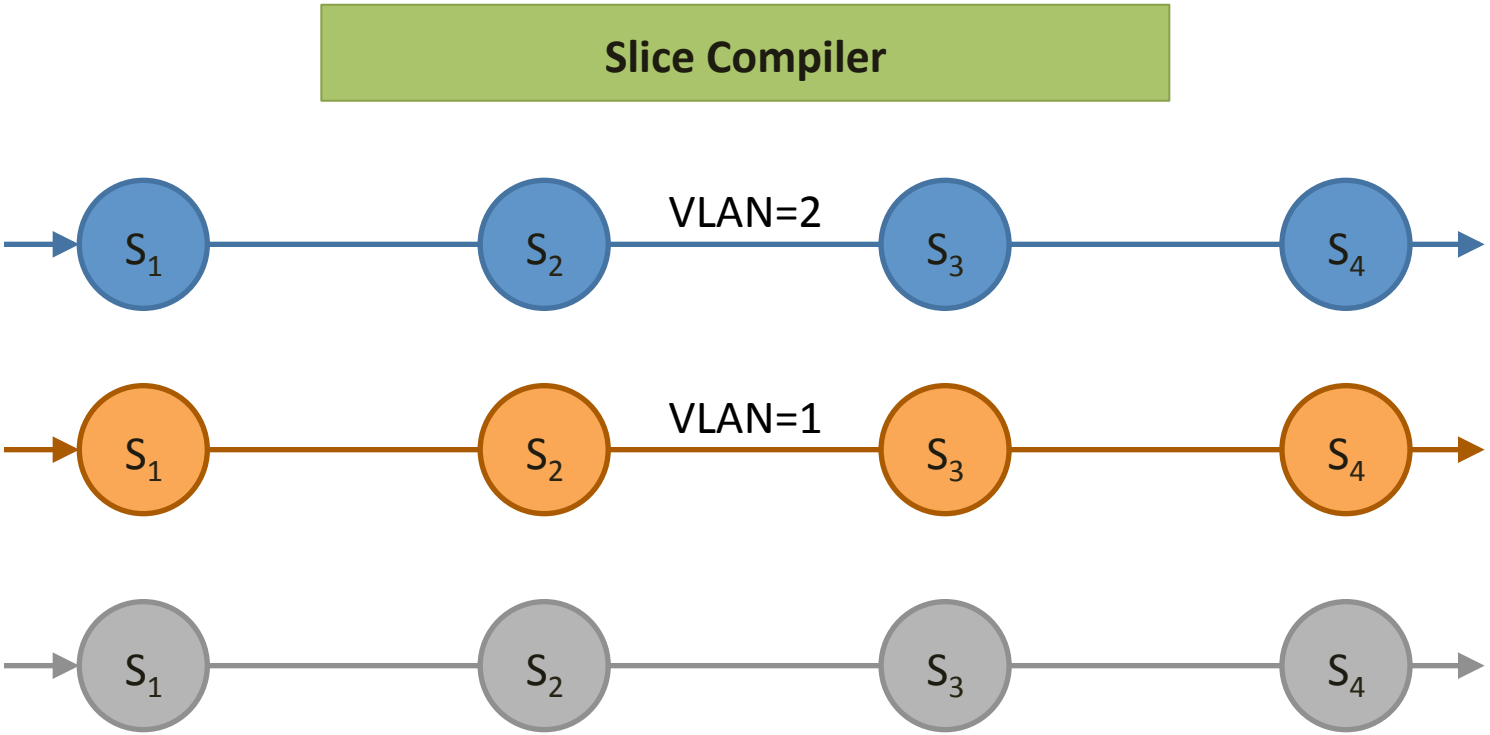
Monitor

```
tpSrc 22 ==> Query_1 <+>
tpSrc 80 ==> Query_2
```

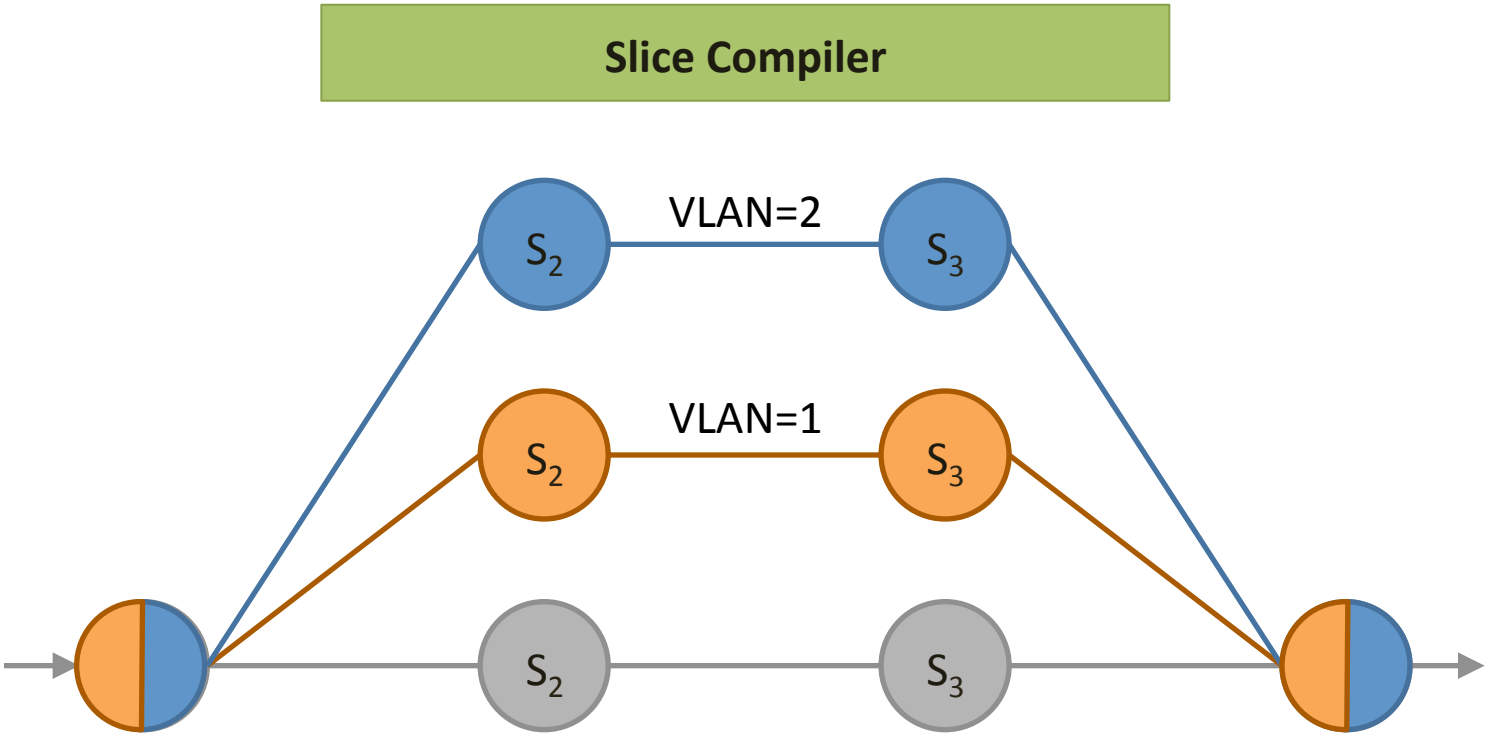
VLAN-based Isolation



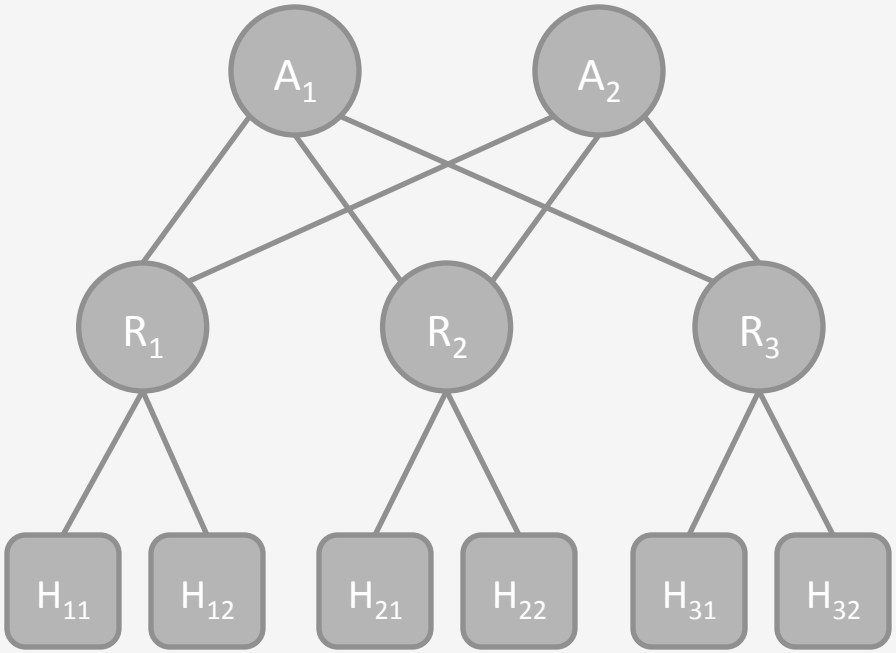
VLAN-based Isolation



VLAN-based Isolation

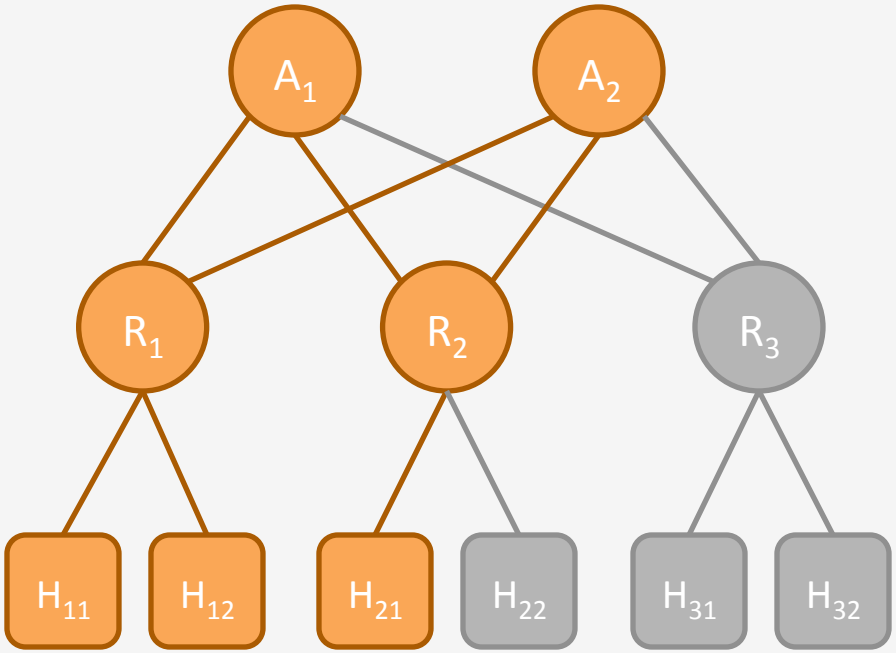


Data Center Isolation



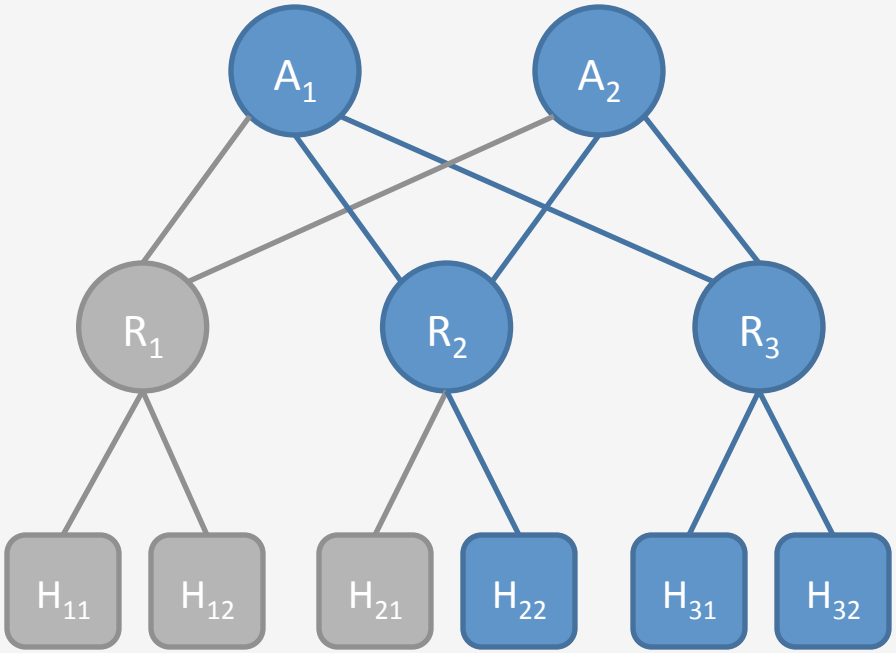
Topology

Controller 1

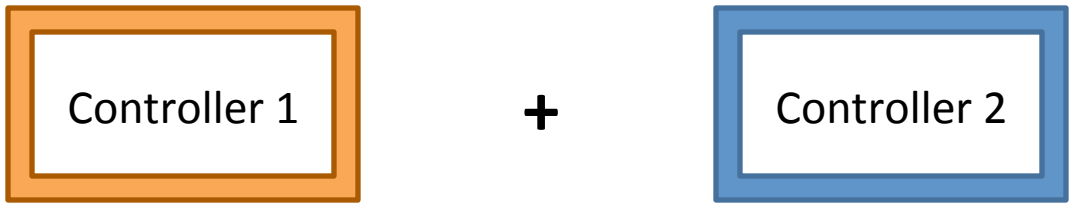


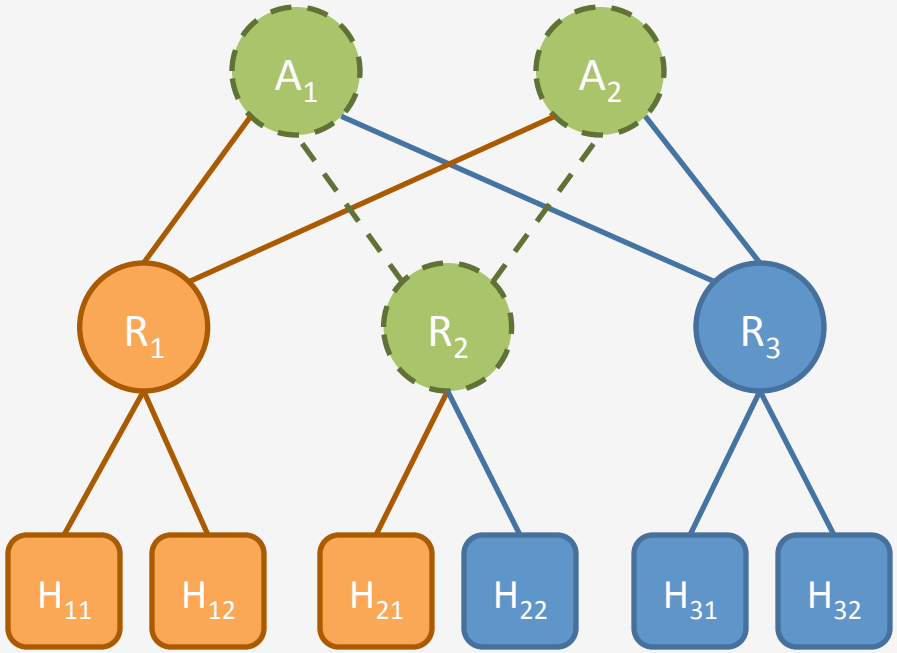
Client 1

Controller 2



Client 2



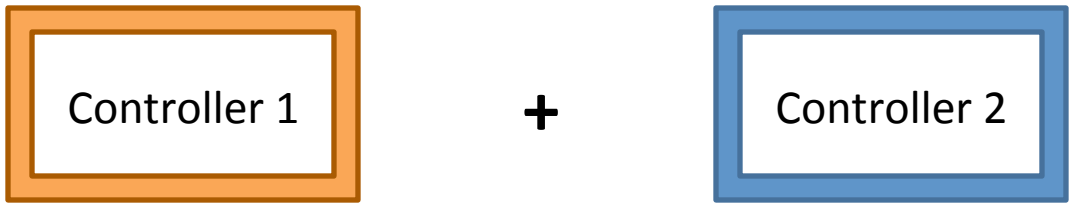


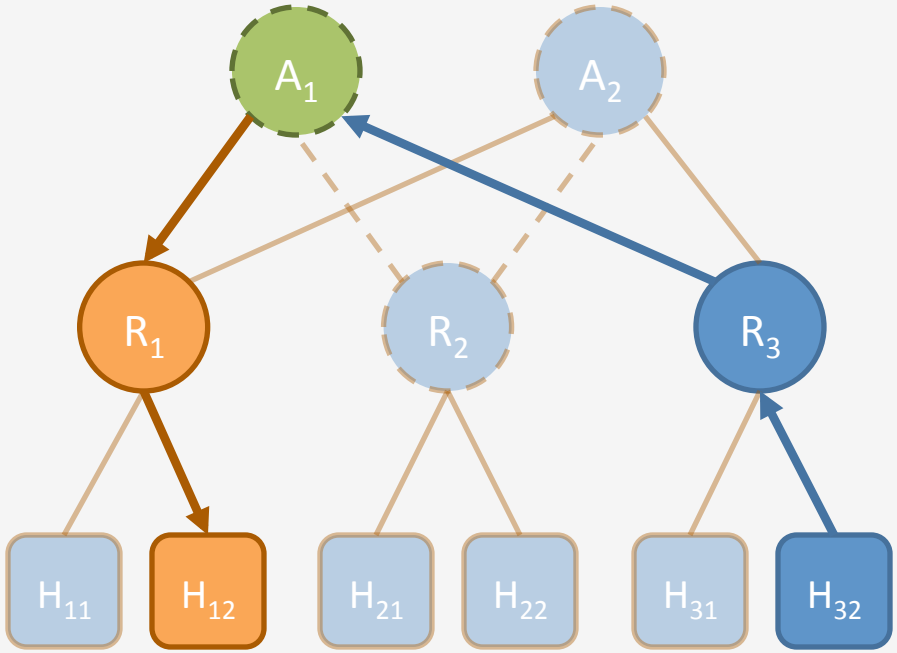
Client 1

Client 2

Overlap

Client 1 + Client 2





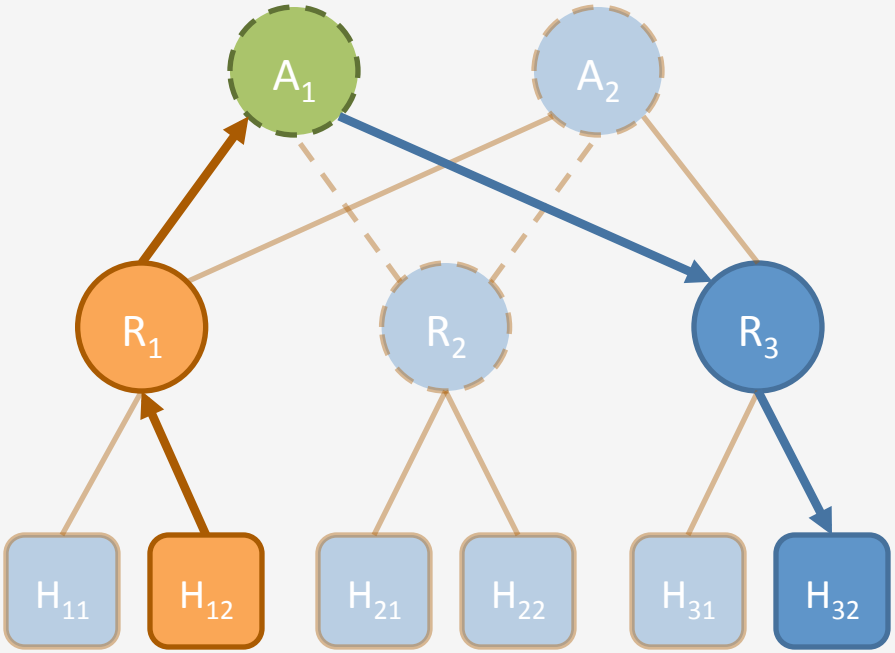
Client 1

Client 2

Overlap

Client 2 **injects packets** into Client 1's section of the network!

Controller 1 + Controller 2

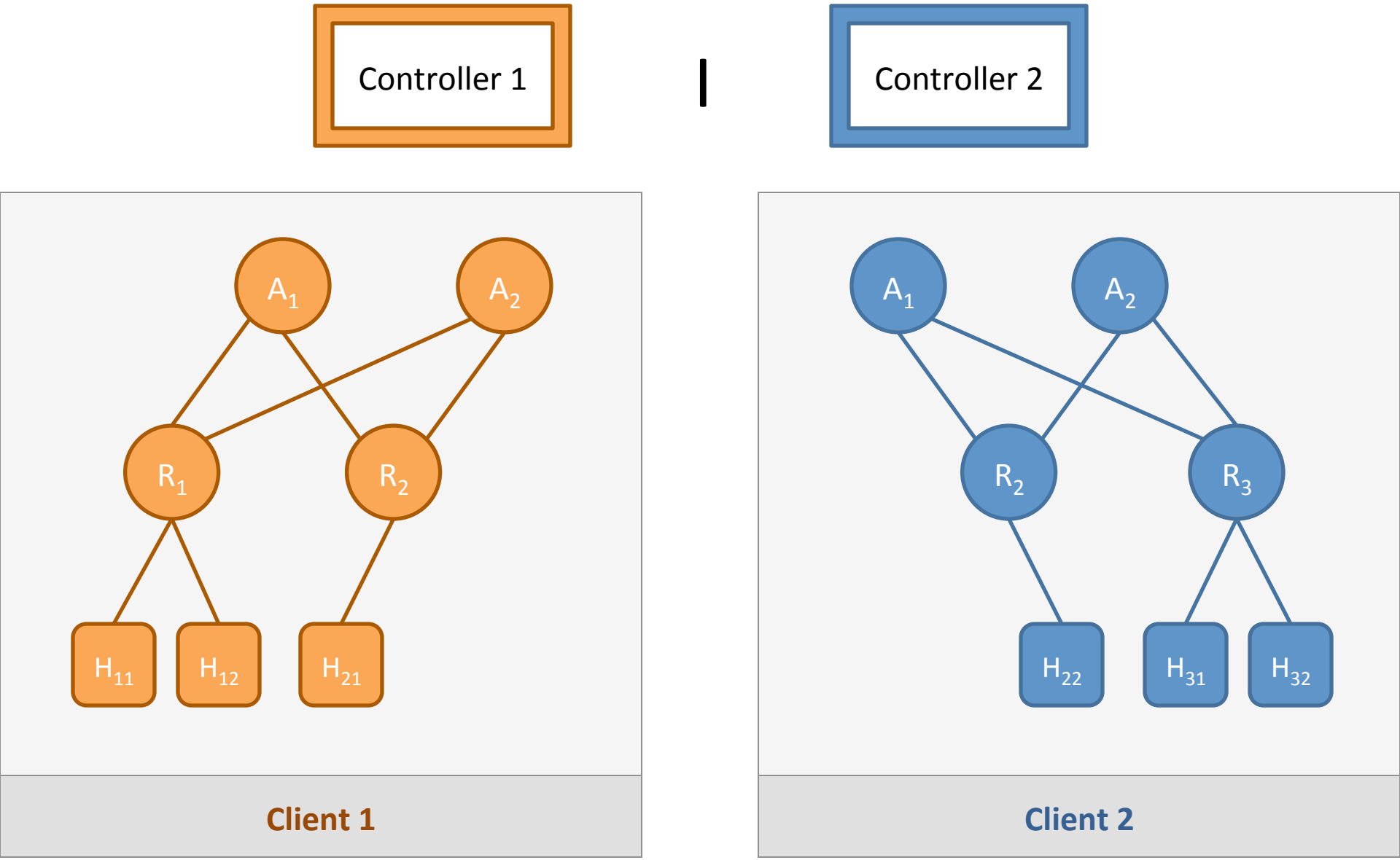


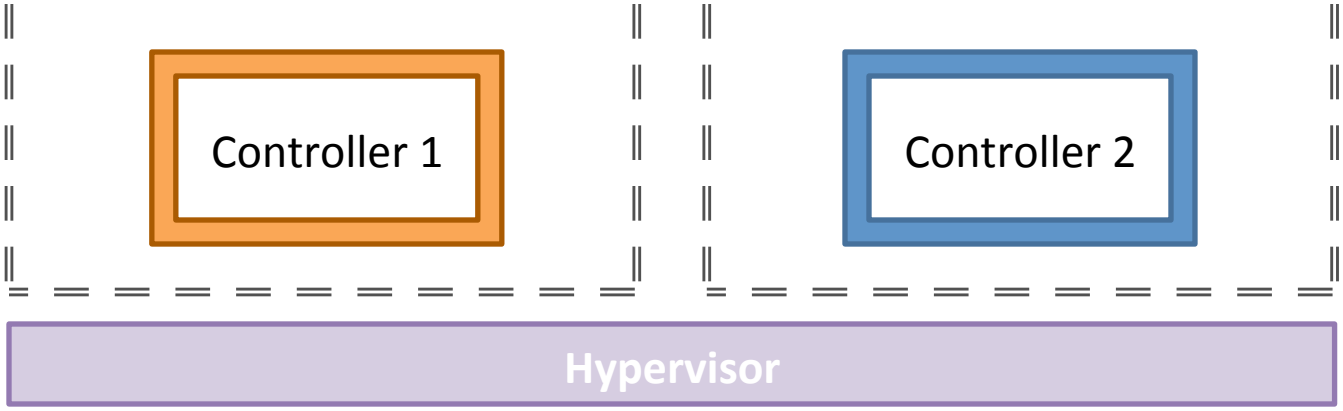
Client 1

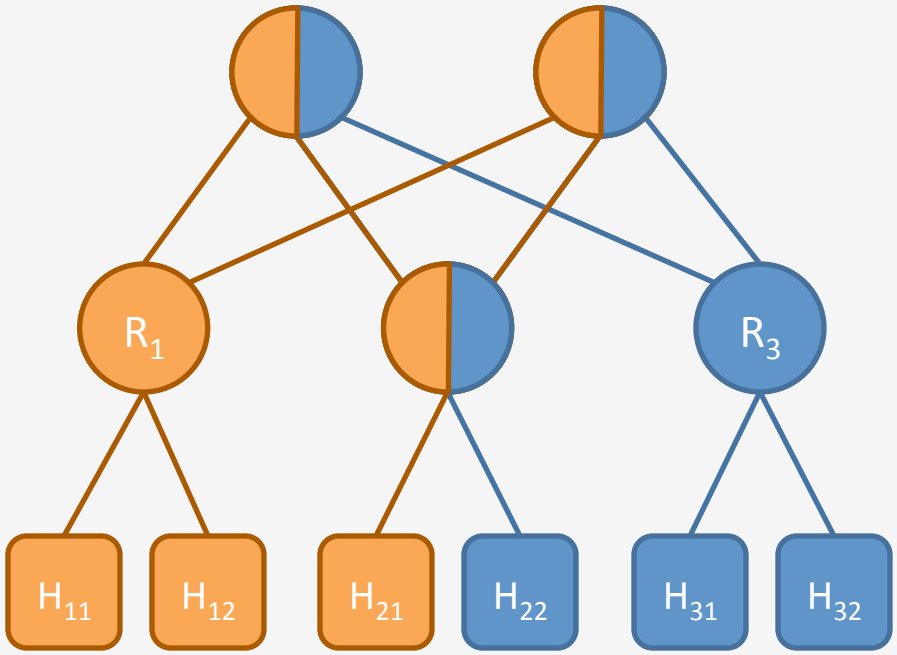
Client 2

Overlap

Client 2 **intercepts packets** from Client 1's section of the network!

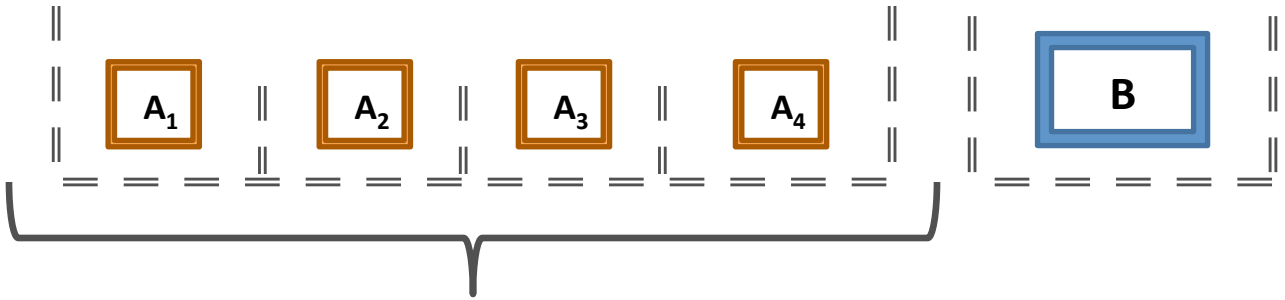

Controller 1
Controller 2
A1
A2
R1
R2
H11
H12
H21
Client 1
A1
A2
R2
R3
H22
H31
H32
Client 2



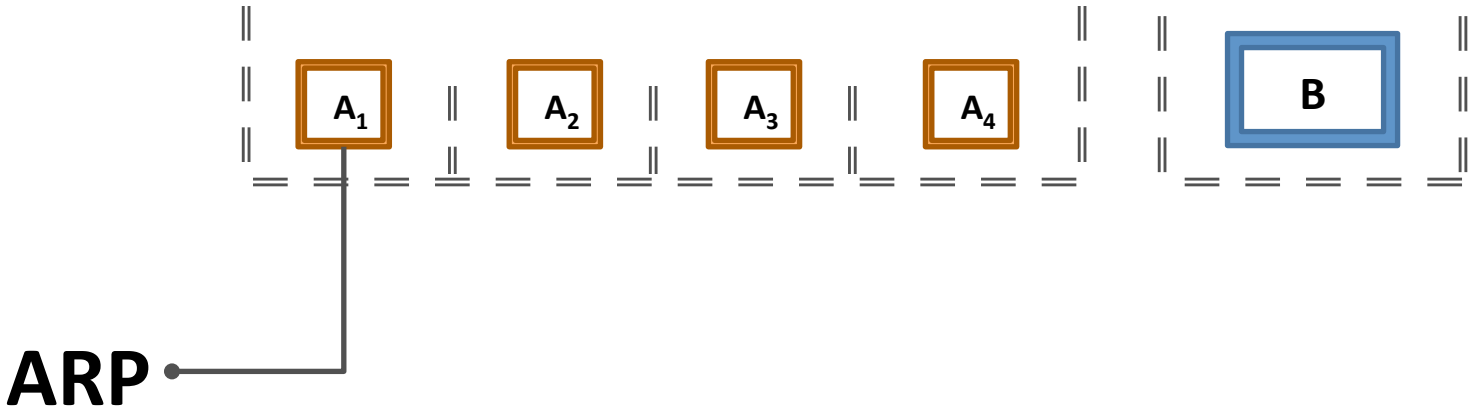



```
PORT == 80
PORT != 80
```

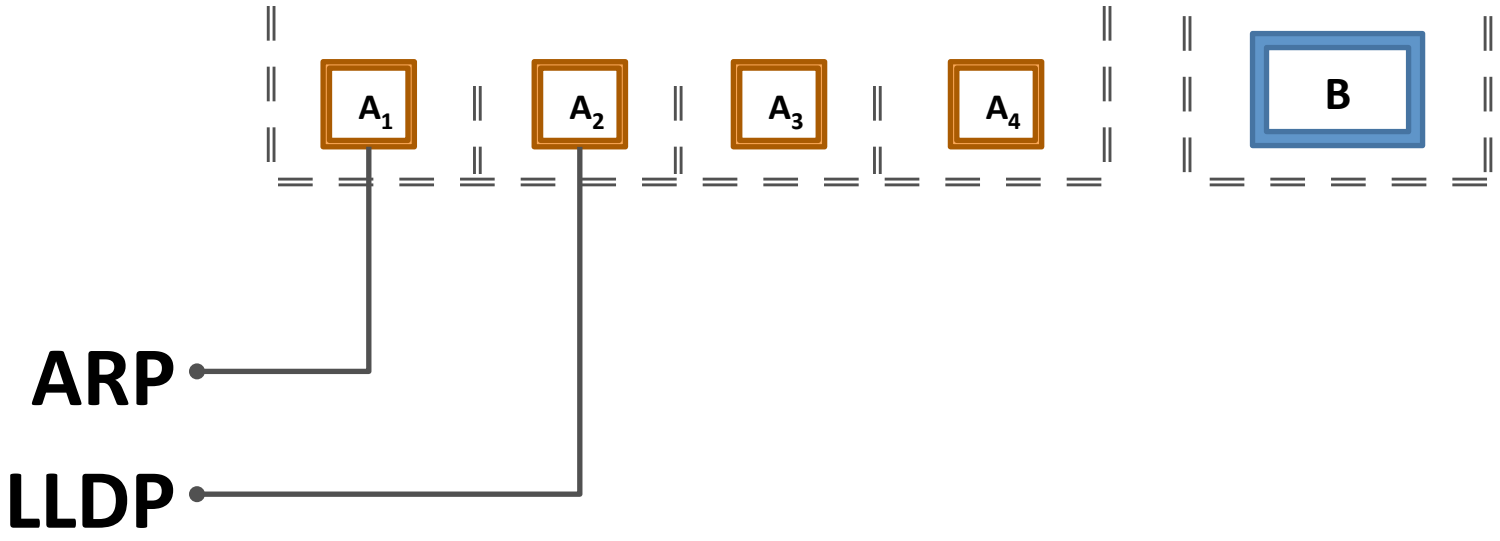
Isolation as Modularity



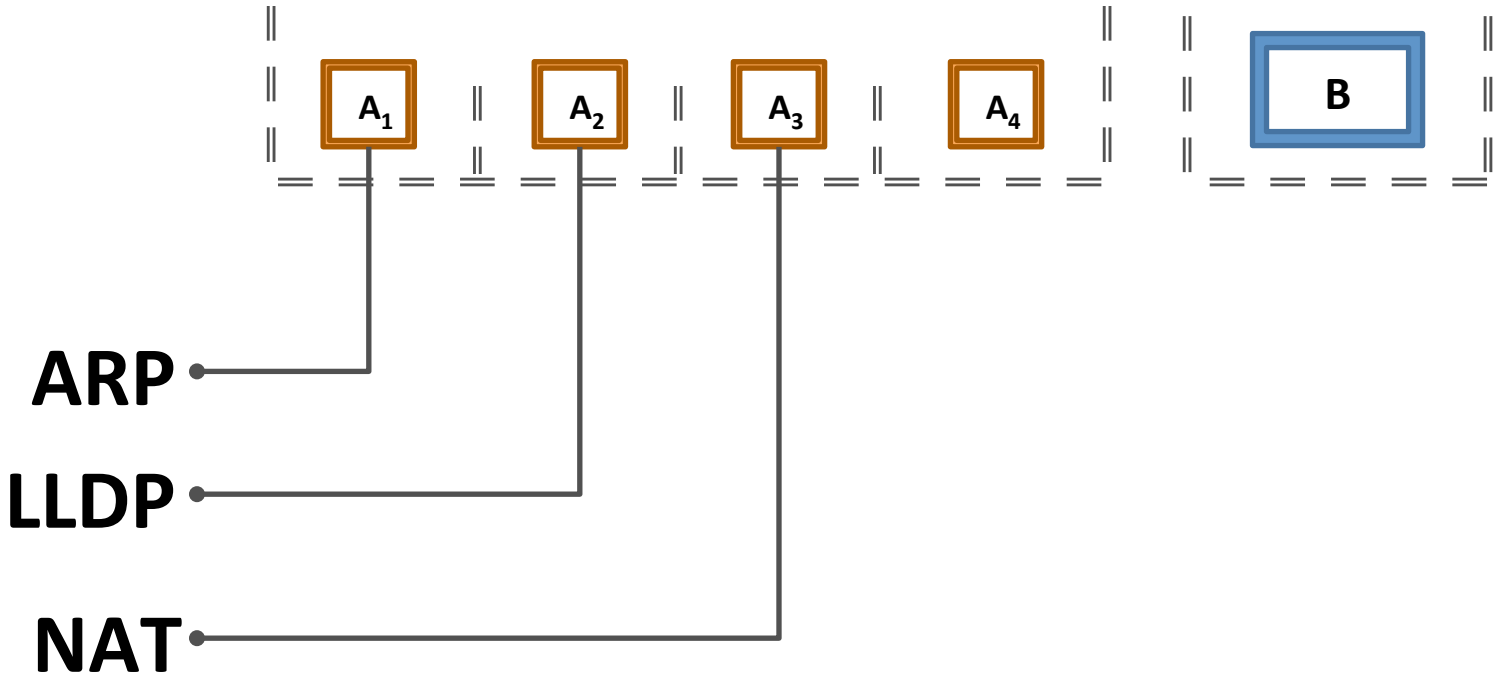
Isolation as Modularity



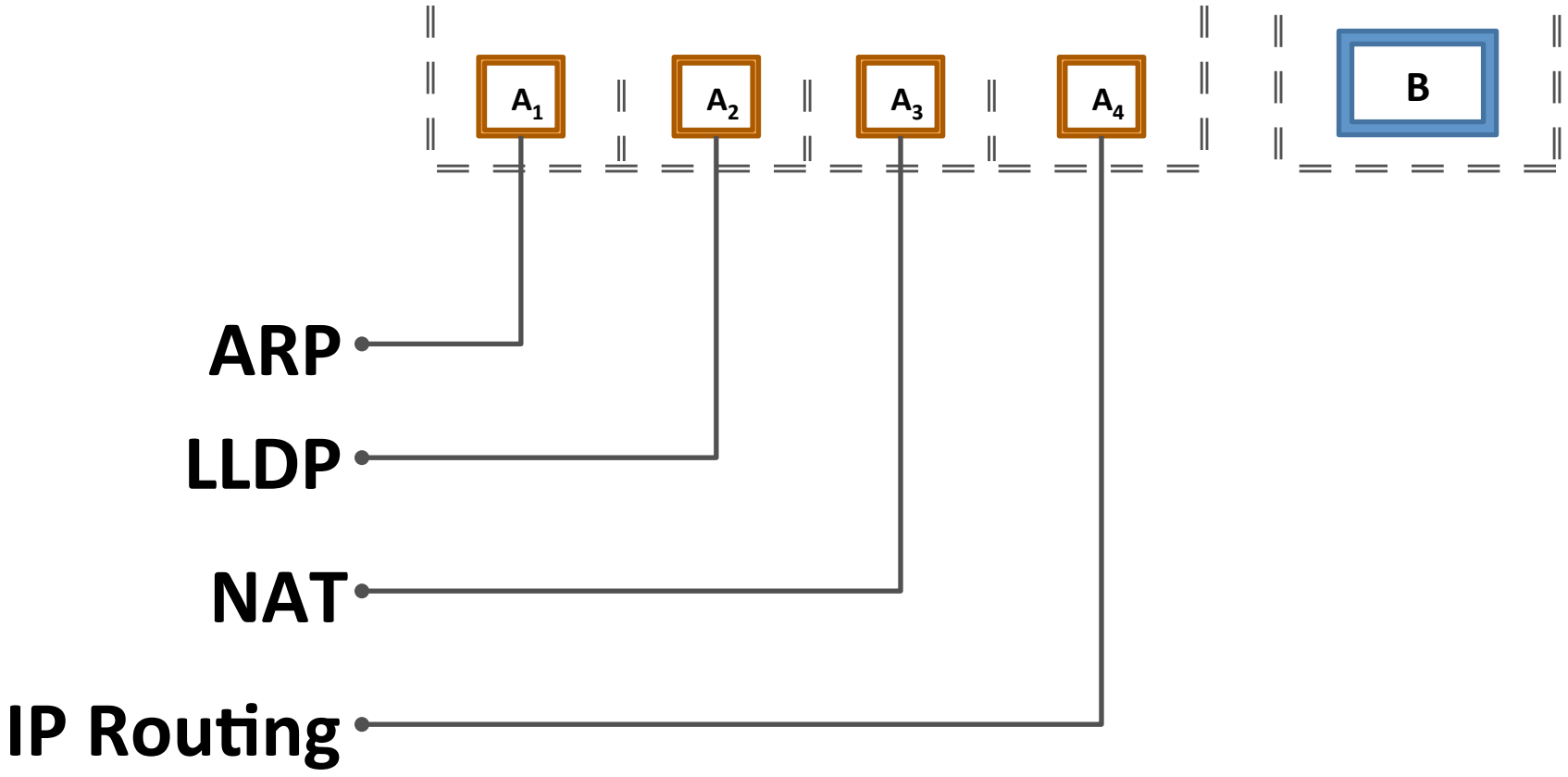
Isolation as Modularity



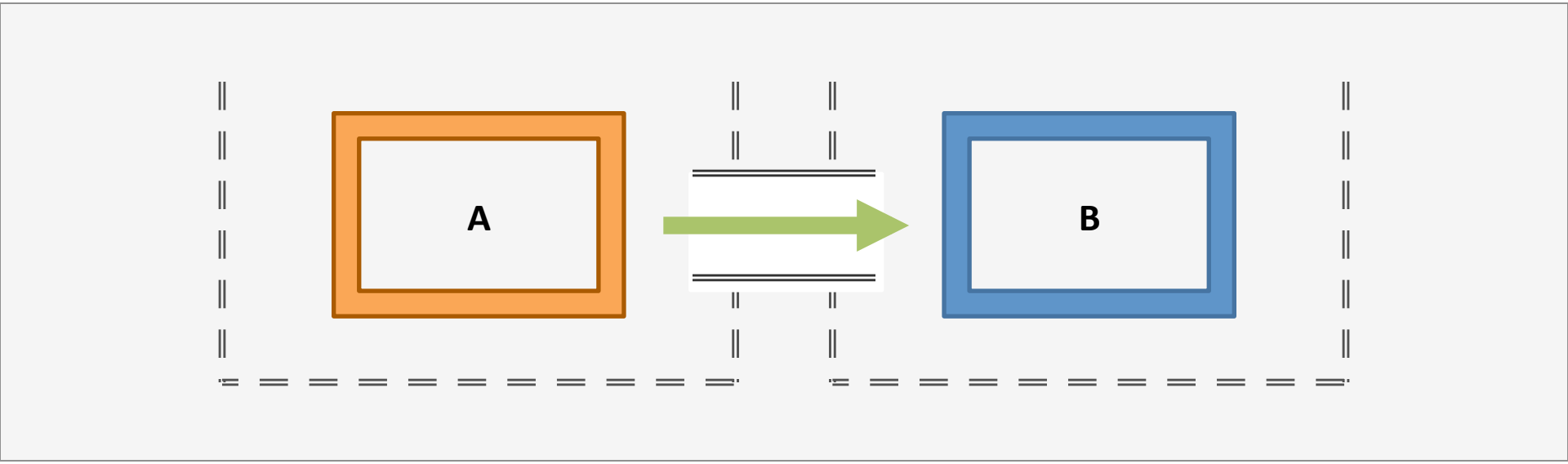
Isolation as Modularity



Isolation as Modularity

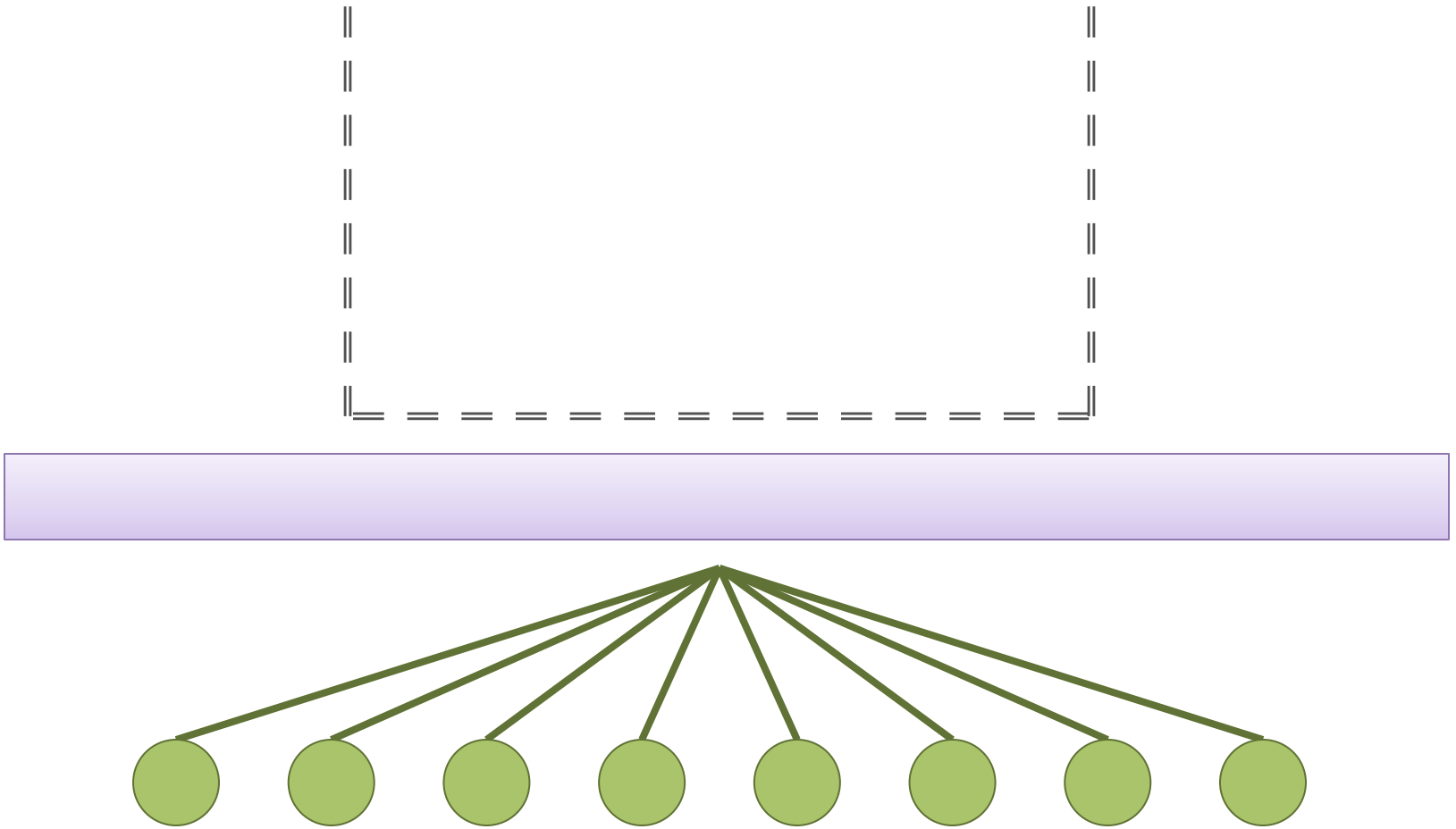


Read-only Slices



- Network monitoring
- Usage-based billing

Precise Semantics

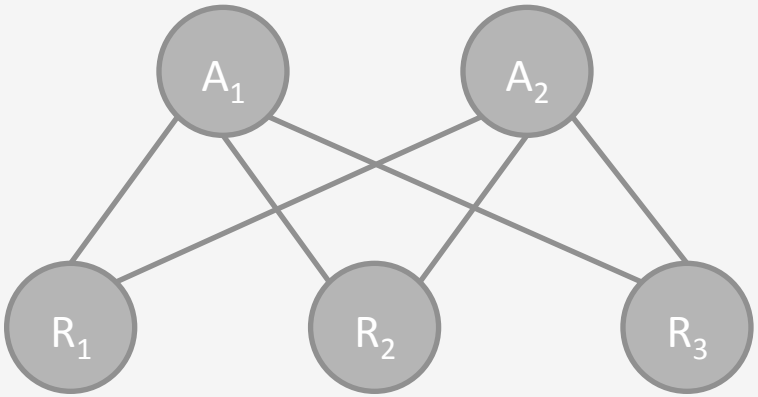


Contributions

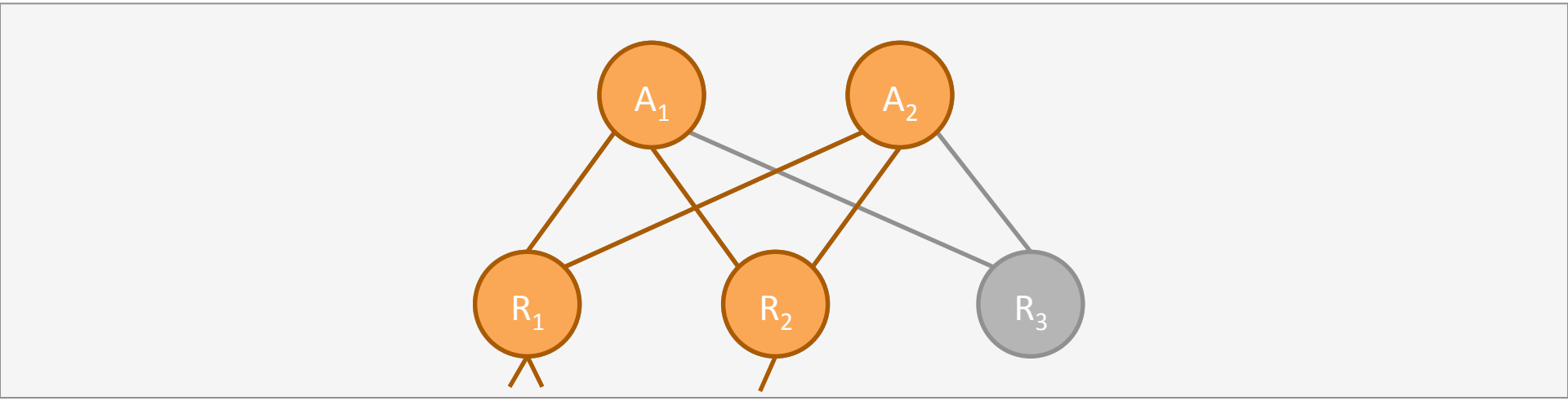
Contributions

- A **new language** for slices.

Slices

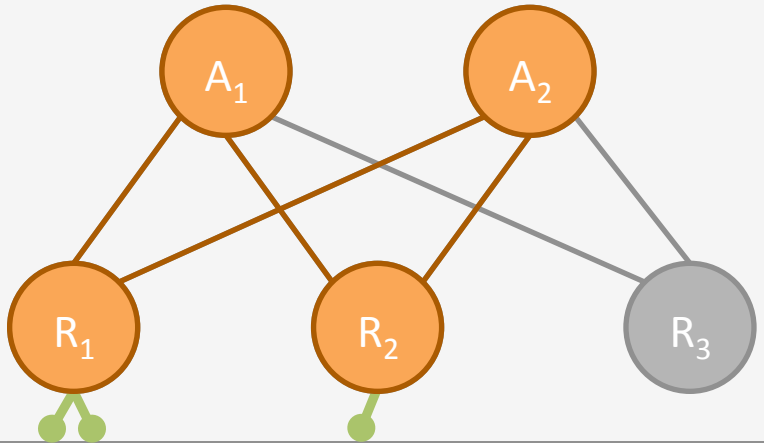


Slices



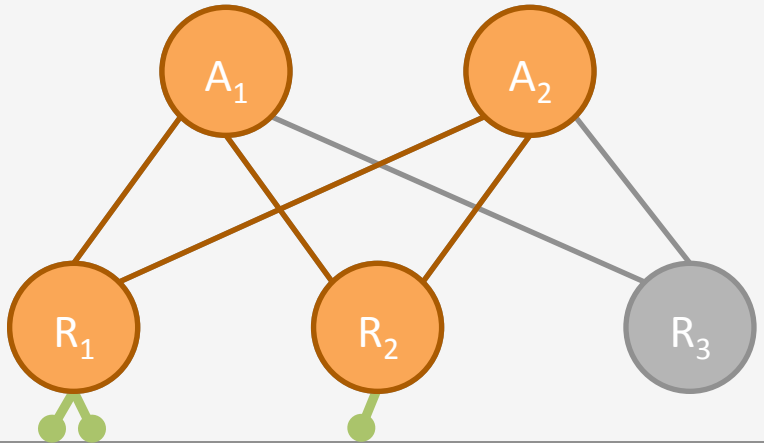

```
# topology
topo = nxtopo.NXTopo()
topo.add_switch(name="R1",ports=[1,2,3,4])
topo.add_switch(name="R2",ports=[1,3,4])
topo.add_switch(name="A1",ports=[1,2])
topo.add_switch(name="A2",ports=[1,2])
```

Slices




```
# slice entries and exits
edges = \
 ([ (p, Top(), Top()) for p in
      topo.edge_ports("R1") ] +
  [ (p, Top(), Top()) for p in
      topo.edge_ports("R2") ])
```

Slices




```
# slice constructor
slice = Slice(topo,phys_topo,edges)
```

Programming with Slices

```
# ARP module
arp_slice = Slice(phys_topo, phys_topo, arp_edges)
arp_policy = gen_arp_policy(arp_slice)
```

Ethernet type:
0x0806

Programming with Slices

```
# ARP module
arp_slice = Slice(phys_topo, phys_topo, arp_edges)
arp_policy = gen_arp_policy(arp_slice)
```

```
# IP module
ip_slice = Slice(phys_topo, phys_topo, ip_edges)
ip_policy = gen_routing_policy(ip_slice)
```

Ethernet type: 0x0806

Ethernet type: 0x0800

Programming with Slices

```
# ARP module
arp_slice = Slice(phys_topo, phys_topo, arp_edges)
arp_policy = gen_arp_policy(arp_slice)
```

```
# IP module
ip_slice = Slice(phys_topo, phys_topo, ip_edges)
ip_policy = gen_routing_policy(ip_slice)
```

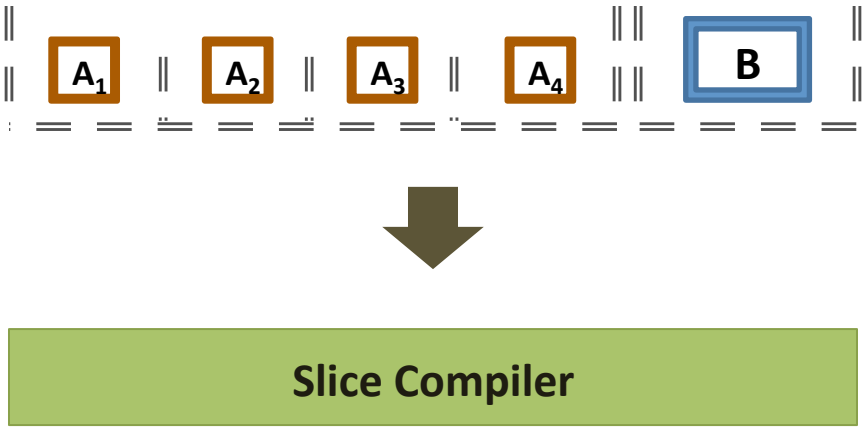
```
# Network-wide policy
slices = [(arp_slice, arp_policy), (ip_slice, ip_policy)]
whole_policy = compile(slices)
```

Ethernet type: 0x0806

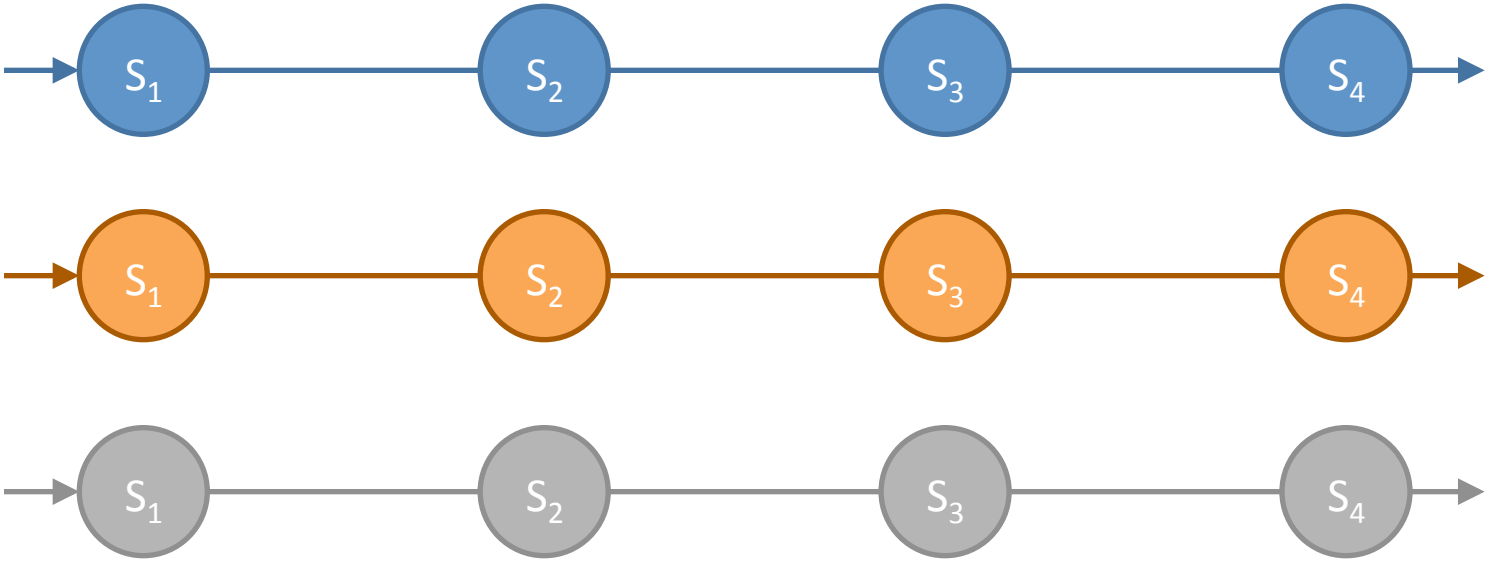
Ethernet type: 0x0800

Contributions

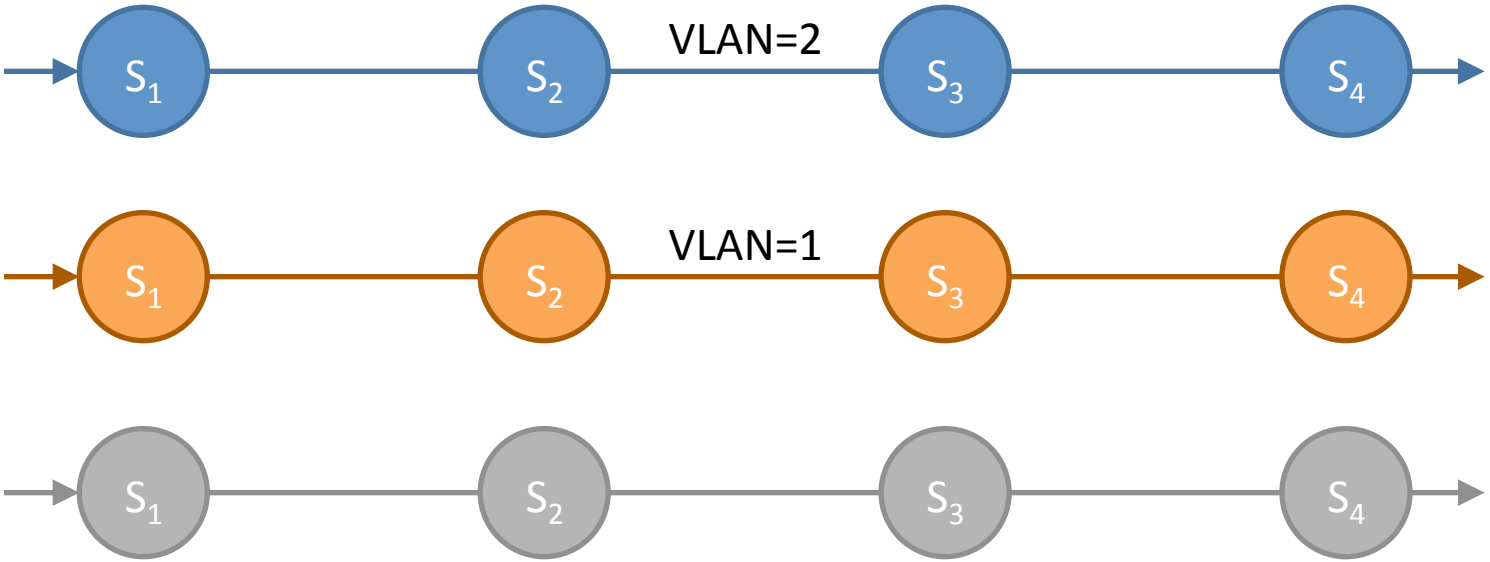
- A **new language** for slices.
- A **compiler** that enforces isolation.



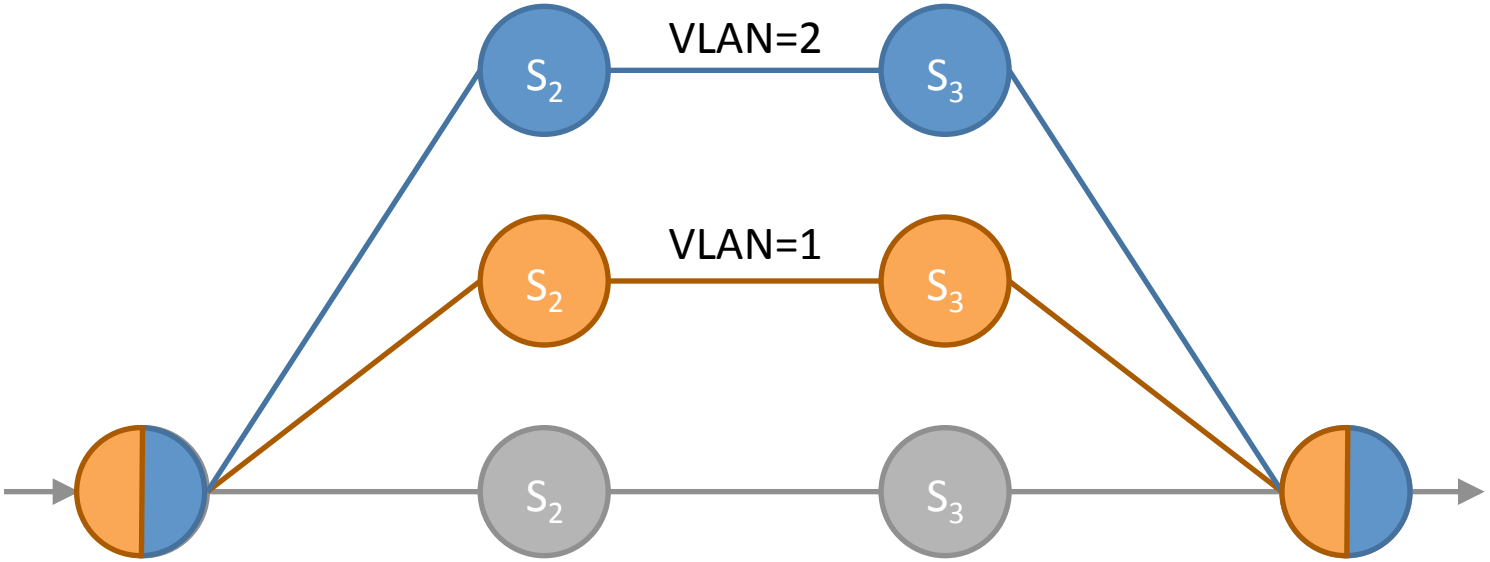
VLAN-based Isolation



VLAN-based Isolation

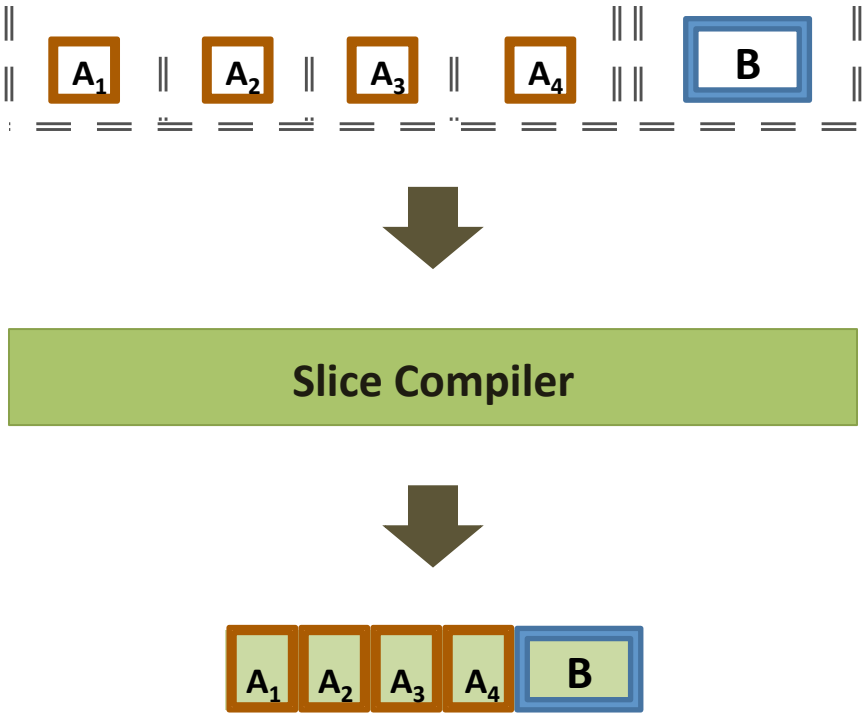


VLAN-based Isolation



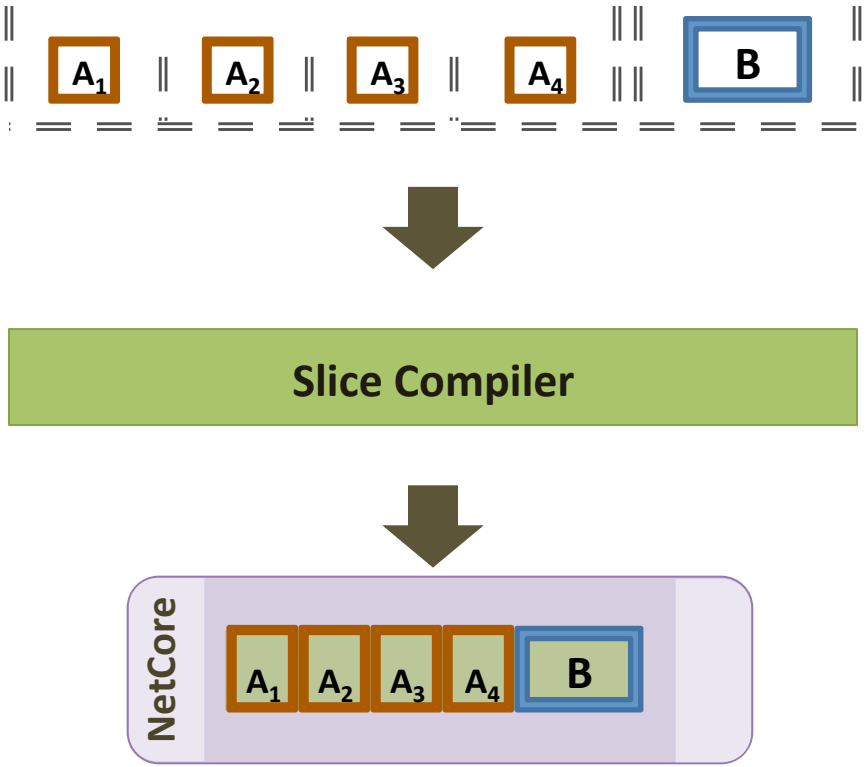
Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.



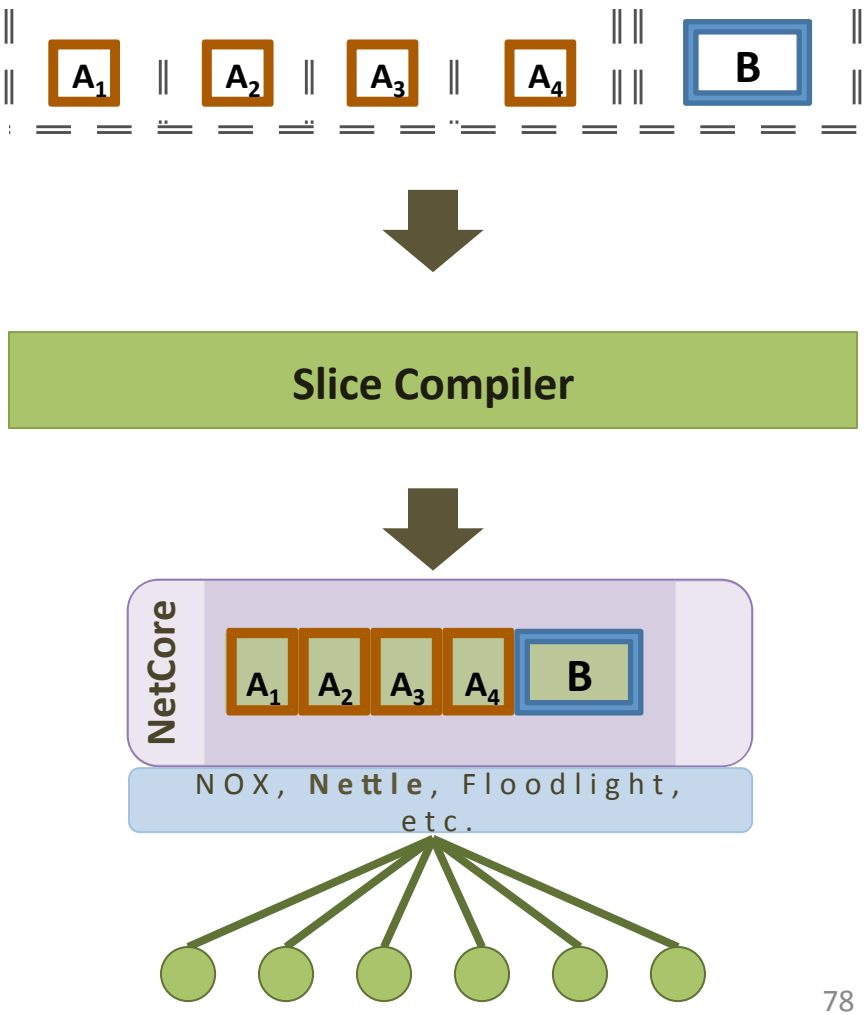
Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.



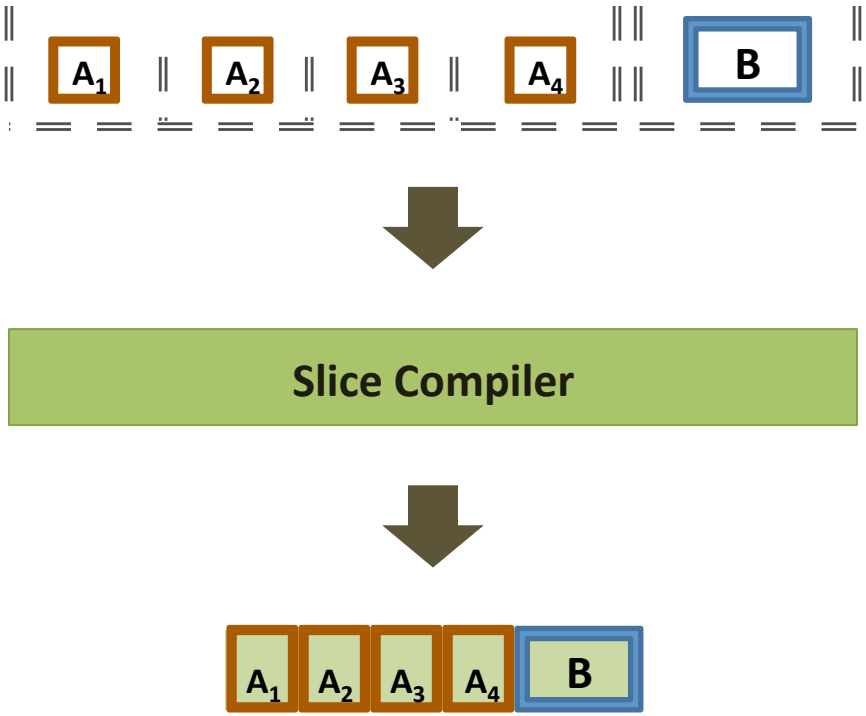
Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.



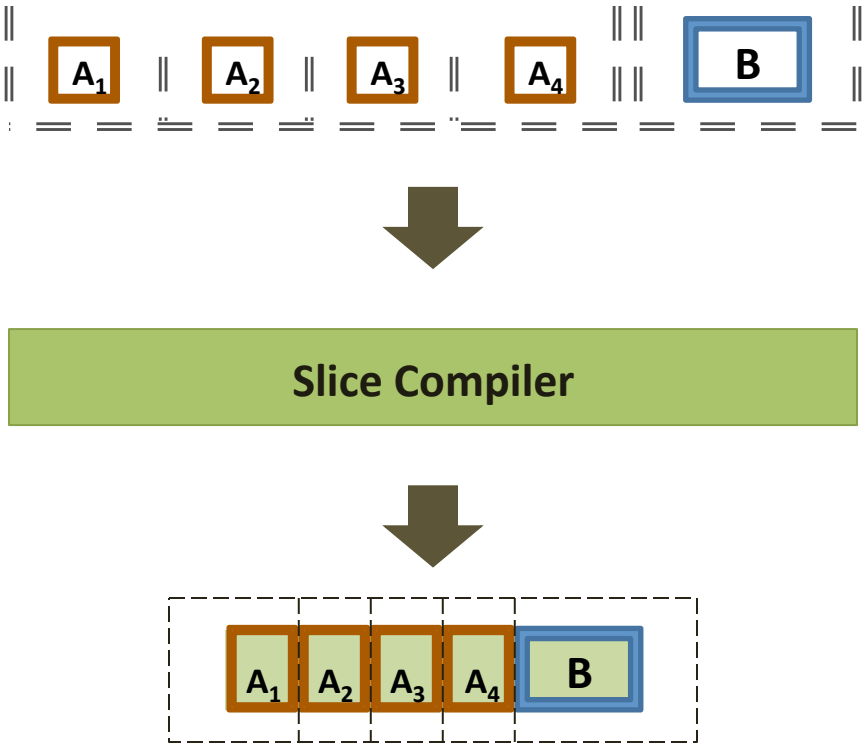
Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.
- A **verifier** that guarantees:



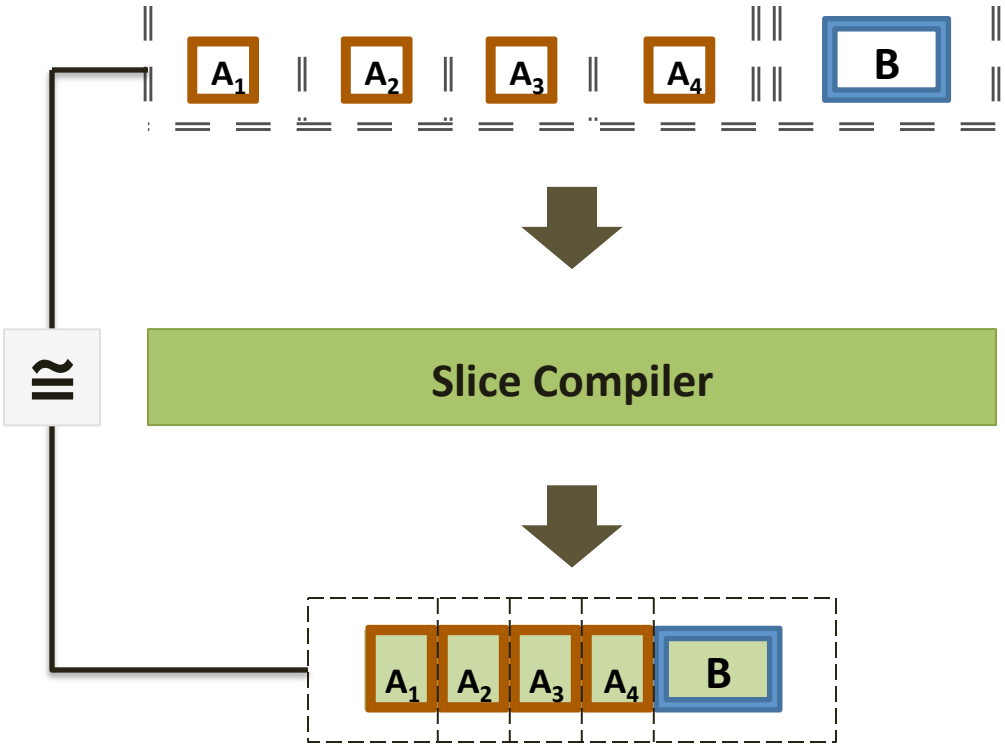
Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.
- A **verifier** that guarantees:
  - isolation

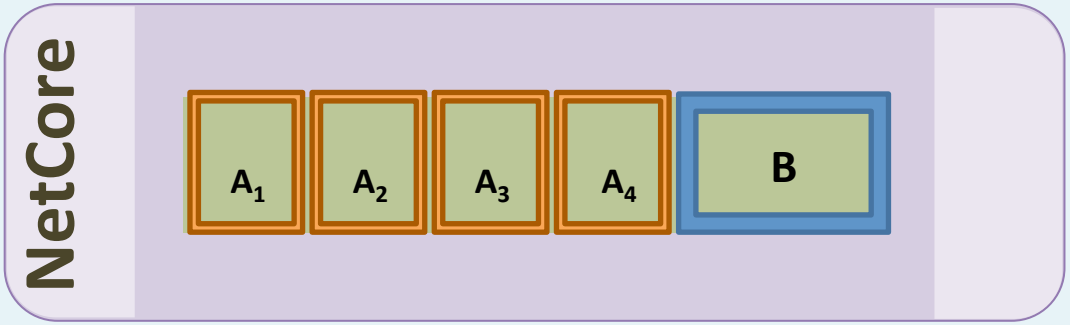


Contributions

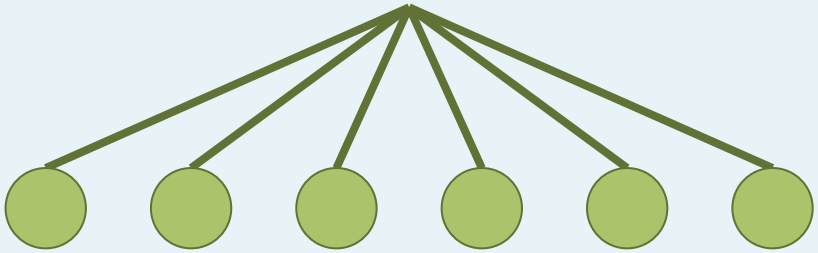
- A **new language** for slices.
- A **compiler** that enforces isolation.
- A **verifier** that guarantees:
  - isolation, and
  - semantic equivalence.



Verifying the Results



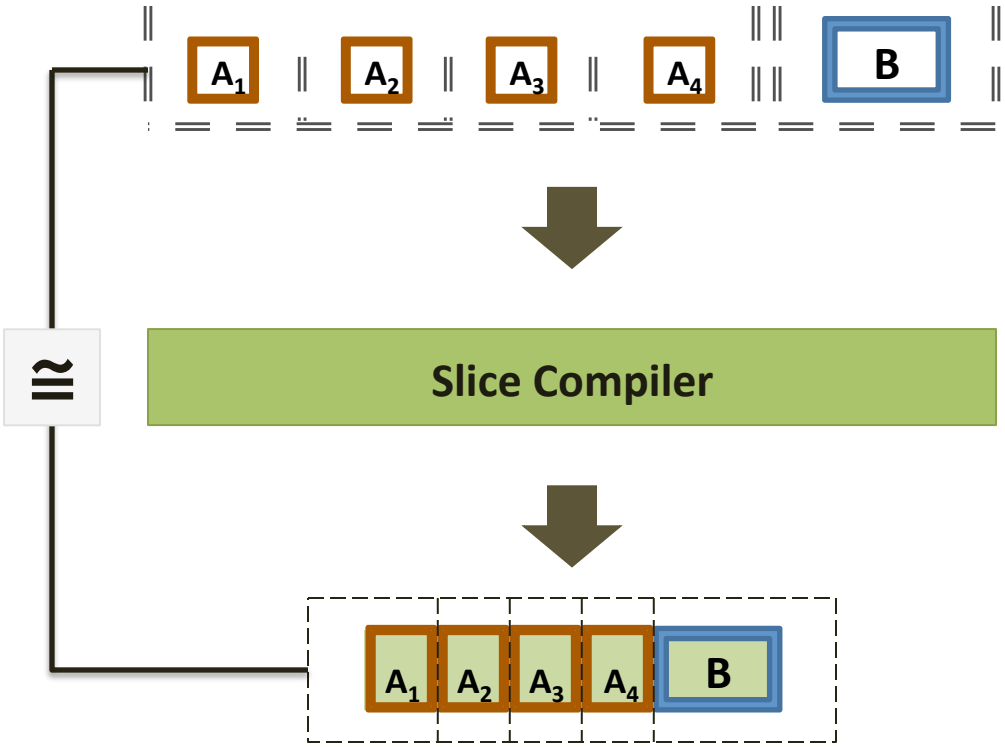
Encoded using **SMT** (Z3)



Encoded using **model checking** (NuSMV)

Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.
- A **verifier** that guarantees:
  - isolation, and
  - semantic equivalence.



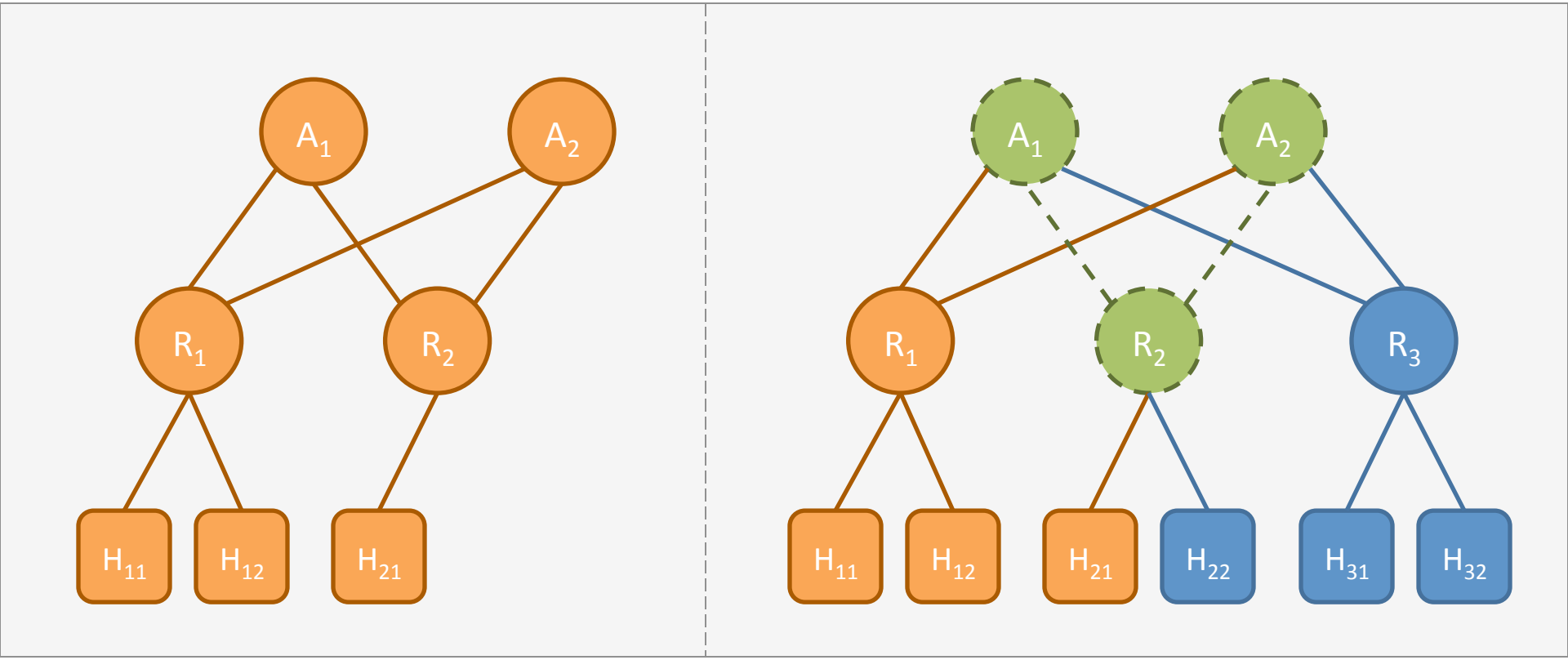
Thank you!

Read the paper:
frenetic-lang.org/papers

Get **the code**:
github.com/frenetic-lang/netcore

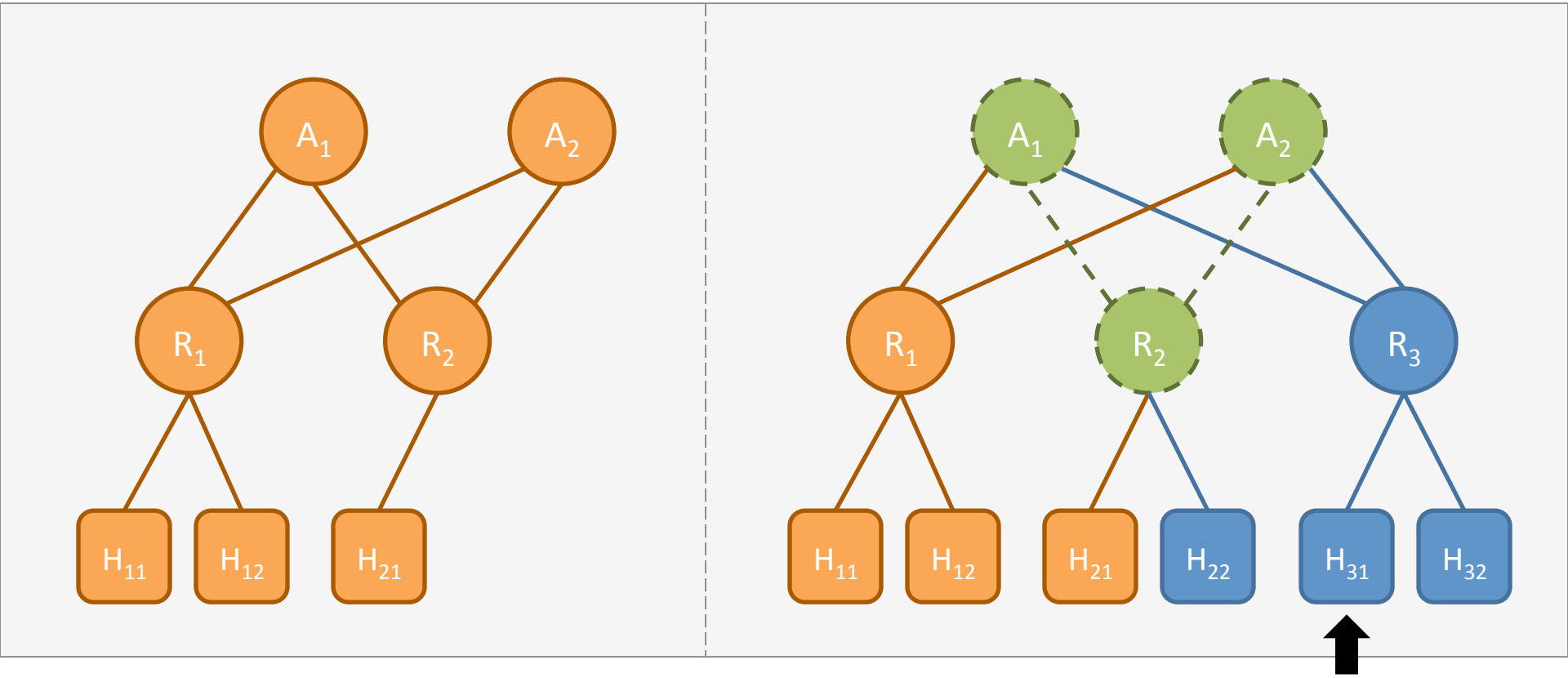
Integrity

1. Slice 2 cannot **inject** packets into Slice 1.



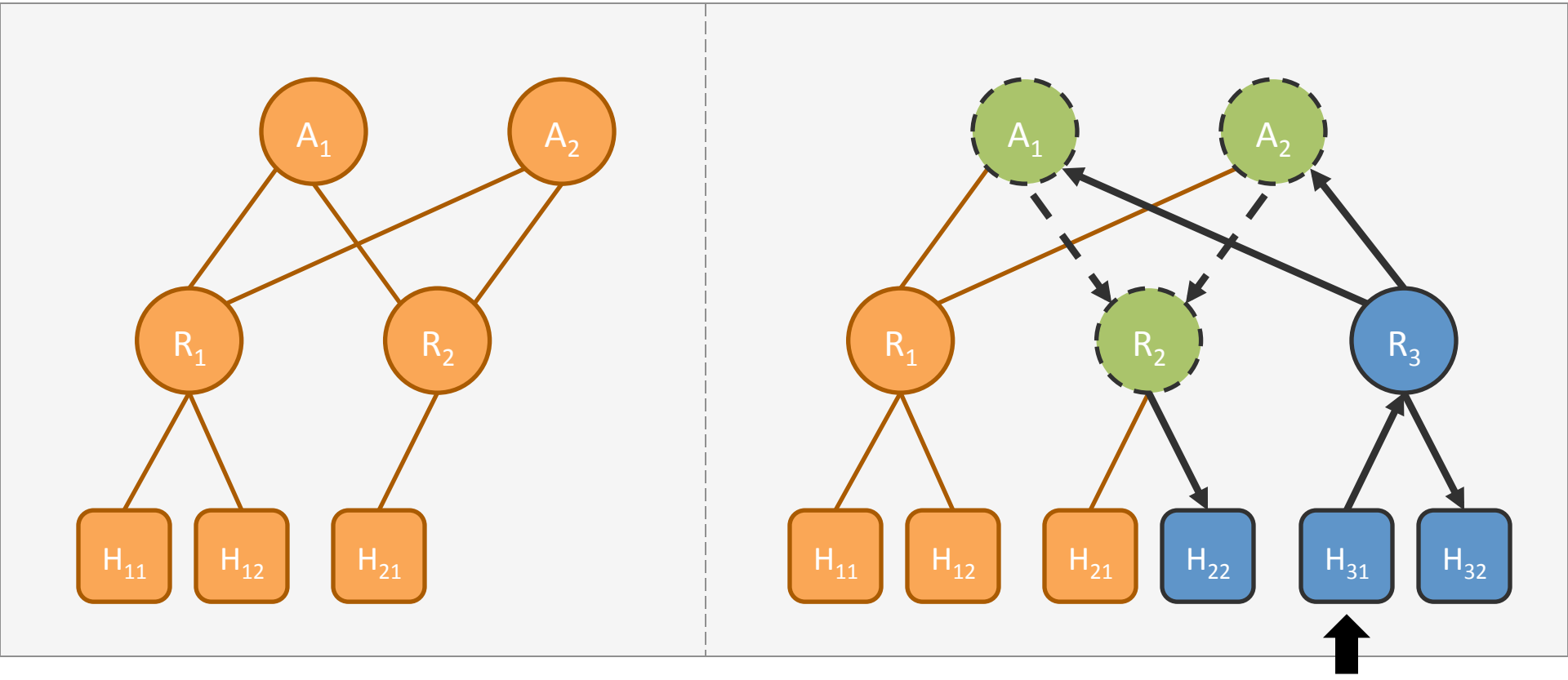
Integrity

1. Slice 2 cannot **inject** packets into Slice 1.



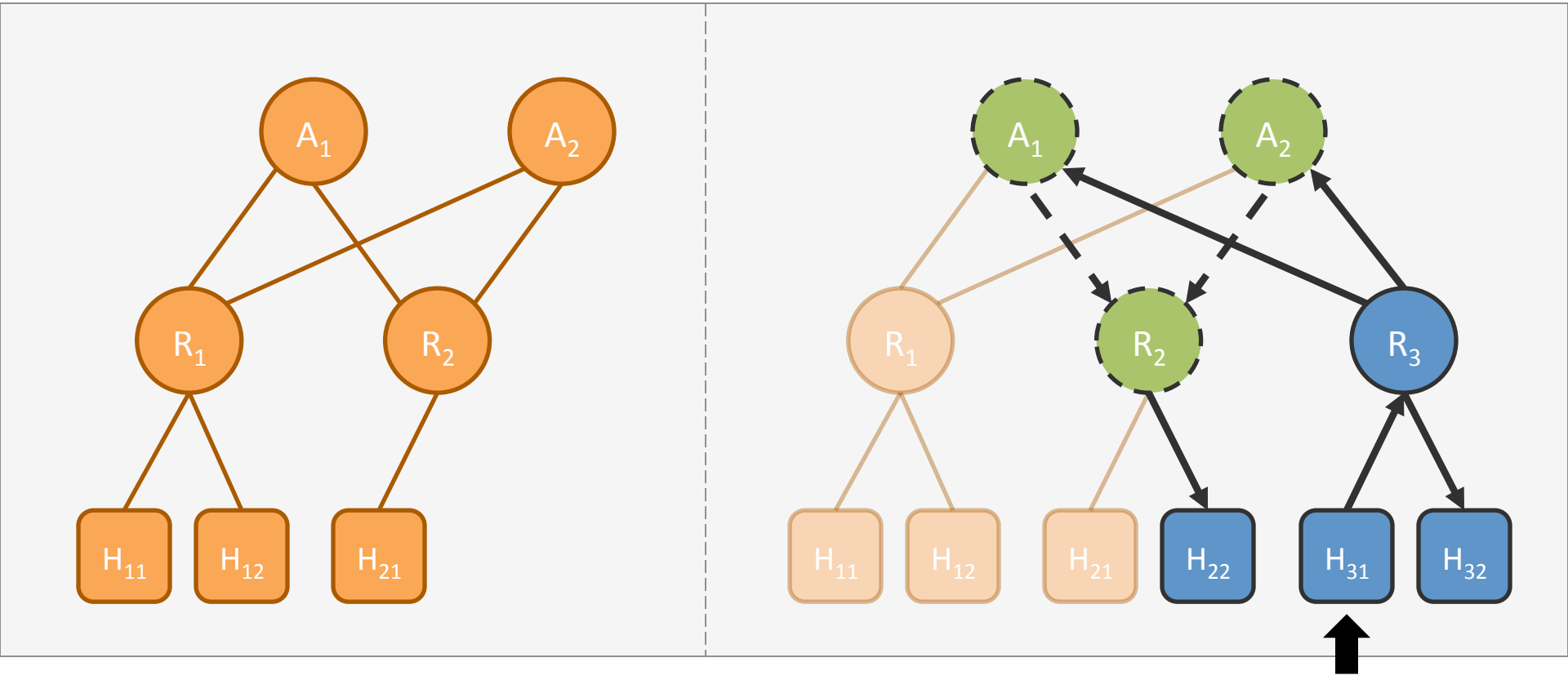
Integrity

1. Slice 2 cannot inject packets into Slice 1.



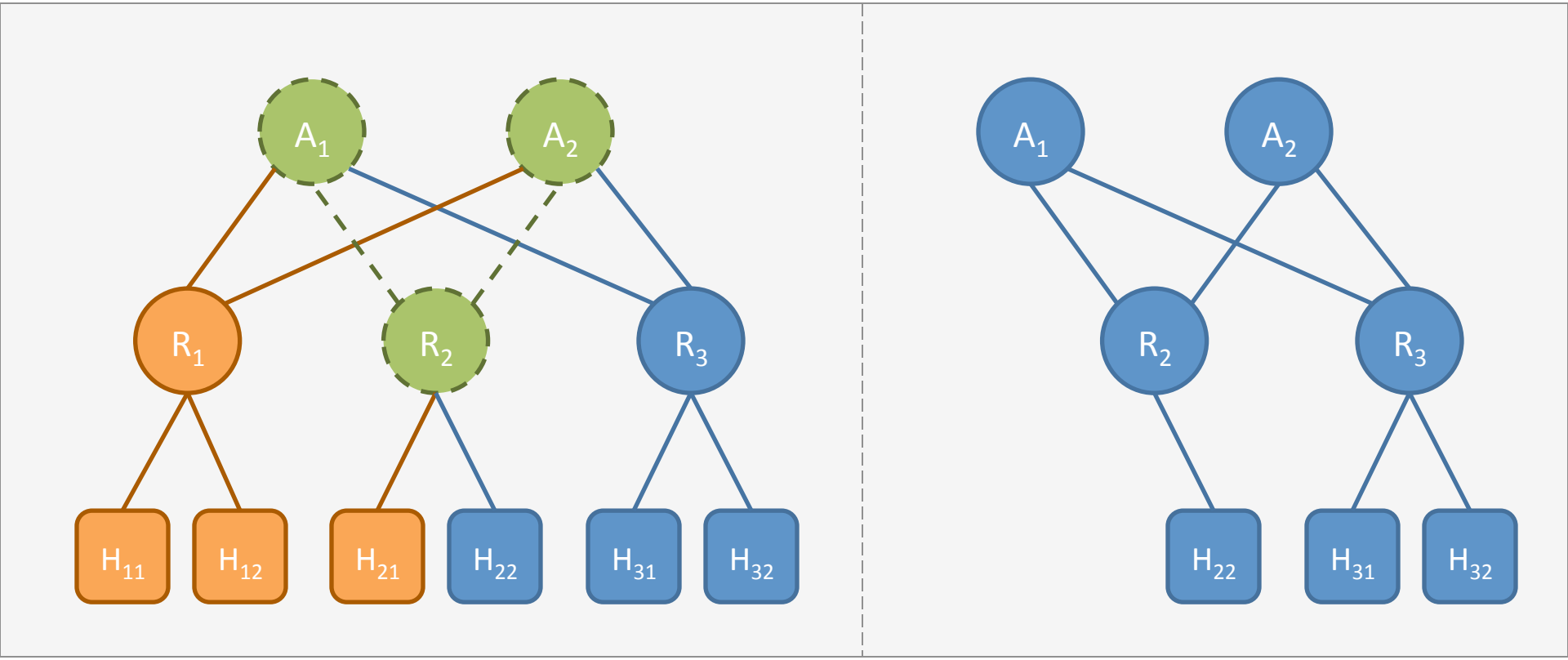
Integrity

1. Slice 2 cannot inject packets into Slice 1.



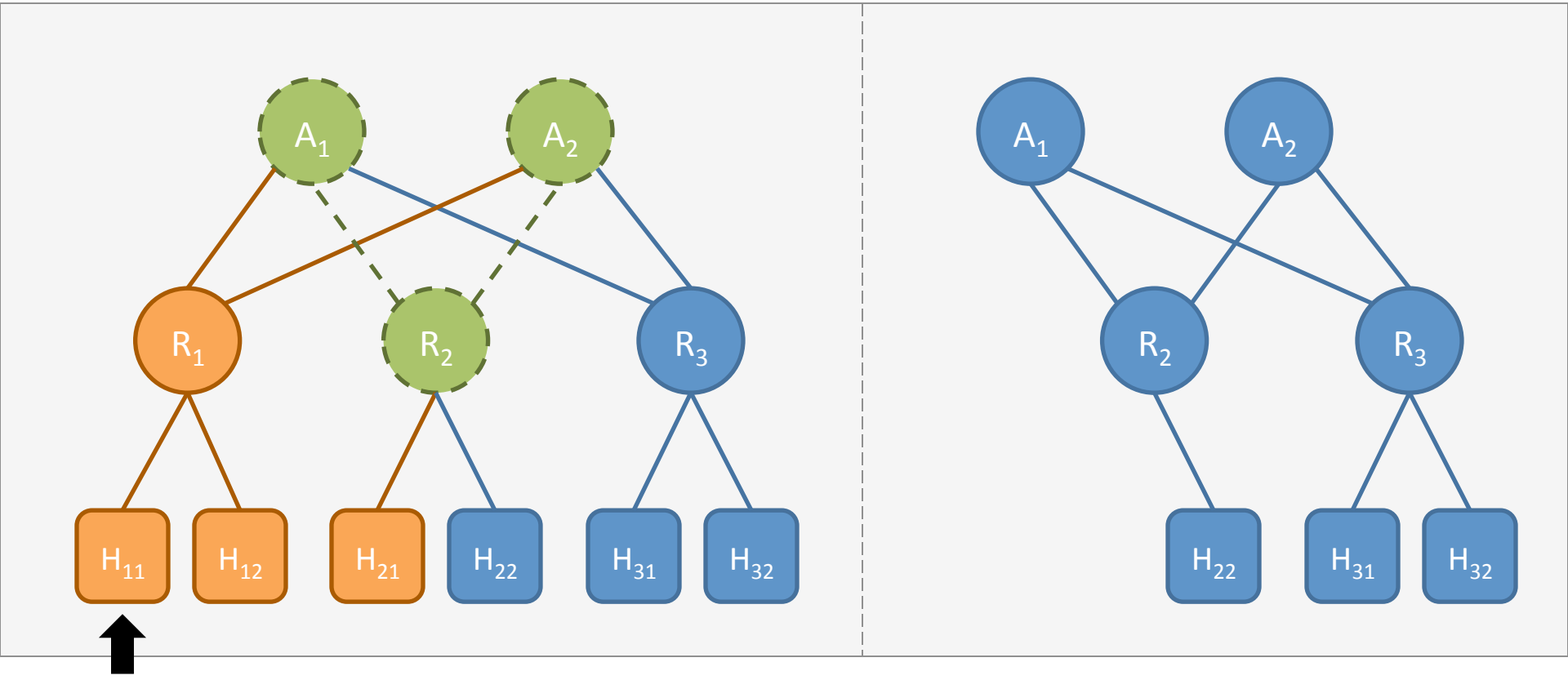
Confidentiality

- Slice 2 cannot siphon packets from Slice 1.



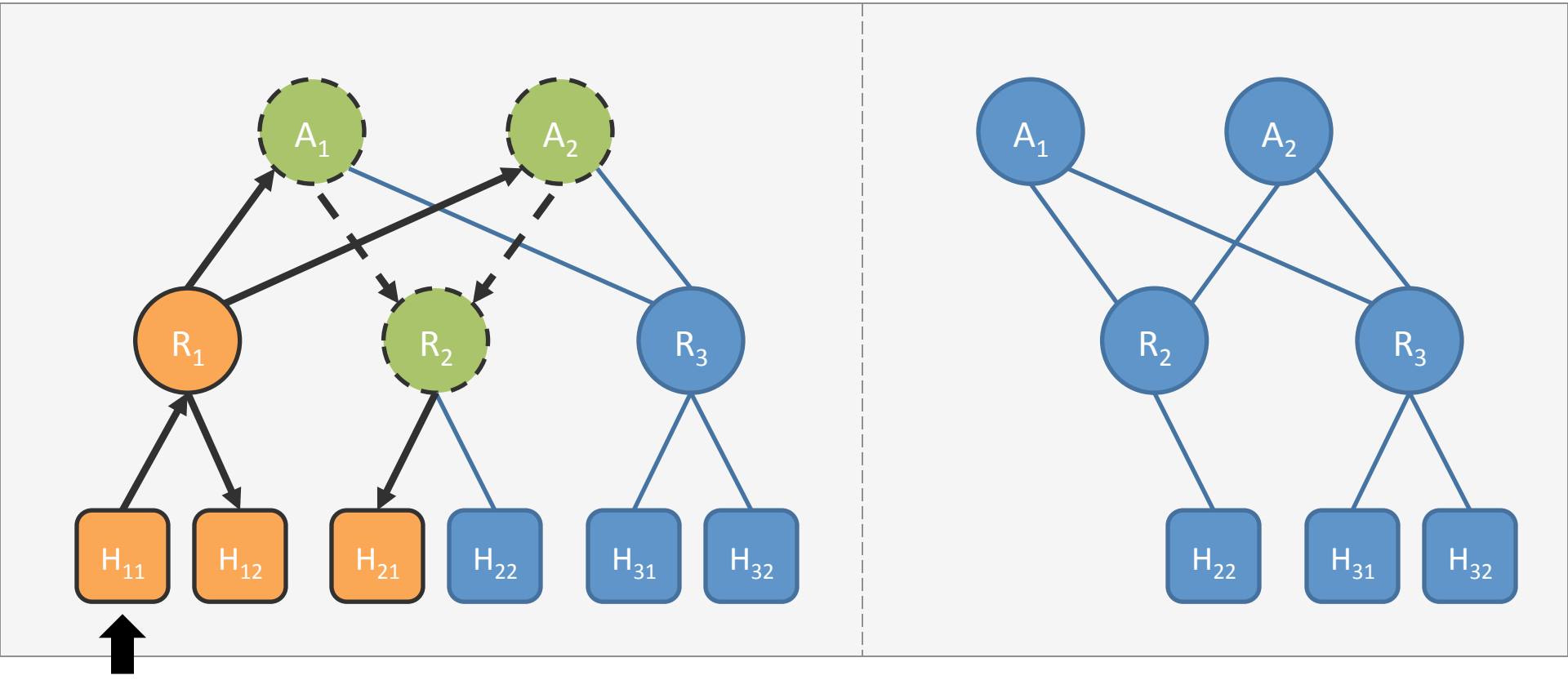
Confidentiality

- Slice 2 cannot siphon packets from Slice 1.



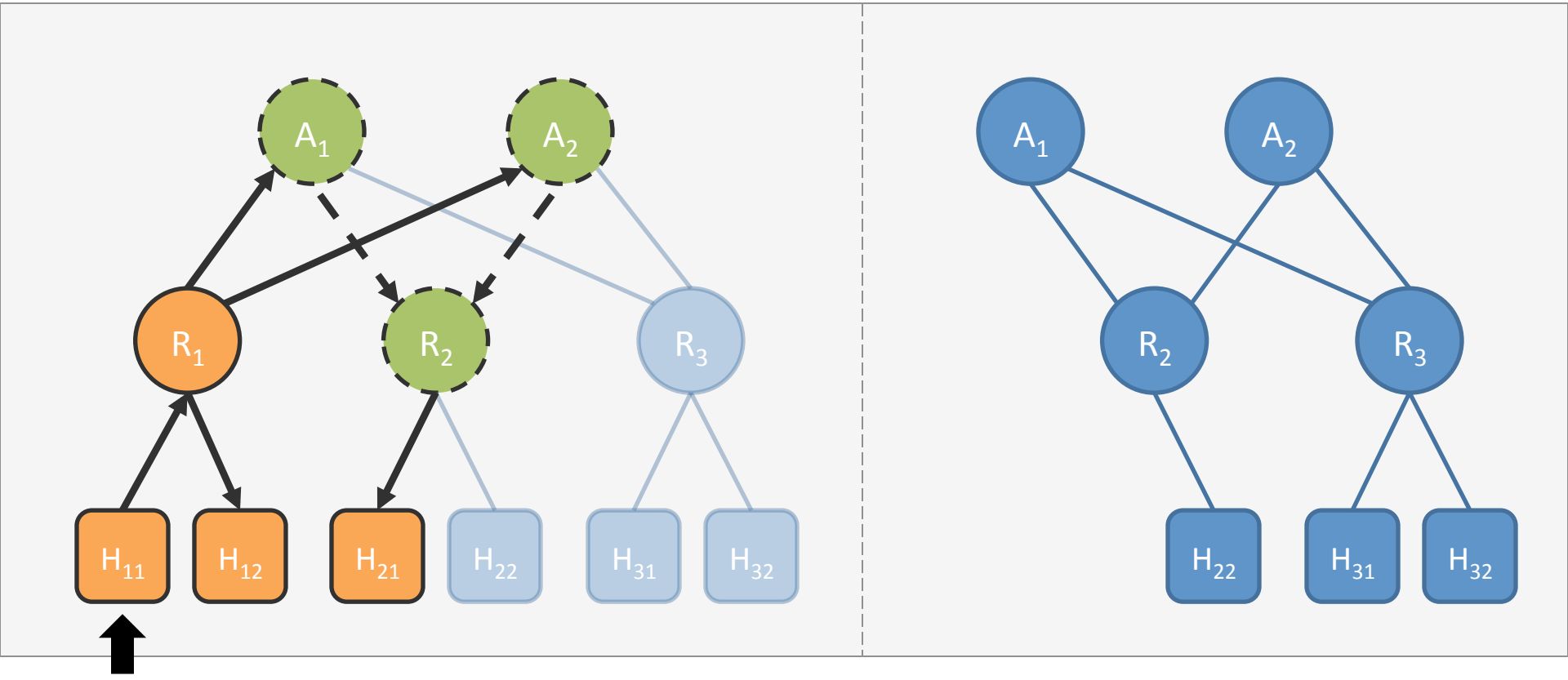
Confidentiality

- Slice 2 cannot siphon packets from Slice 1.



Confidentiality

Slice 2 cannot siphon packets from Slice 1.



Establishing Isolation

- Confidentiality and integrity are global, end-to-end properties.
- But we can establish isolation by enforcing simple, local properties.

Implementation

- VLAN-based implementation.

Verification

- SAT encoding (with Z3)
- **Separate:** given two compiled slices, guarantee that they are separate.
- **Semantics-preserving:** given a source slice + program and a compiled program, verify that they are semantically equivalent.

Read the paper:

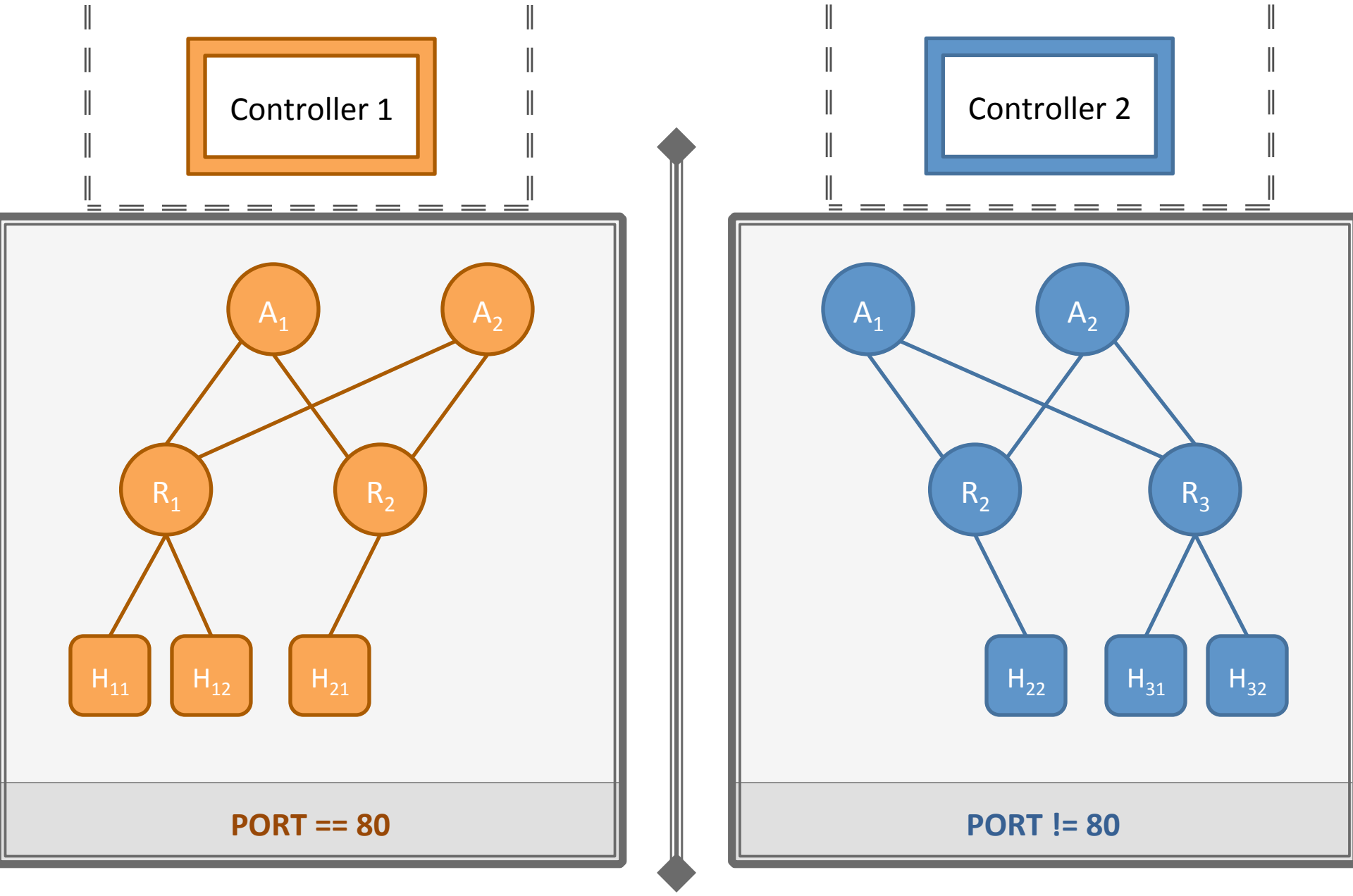
http://www.cs.princeton.edu/~cschlesi

Download the code:

https://github.com/frenetic-lang/netcore

Isolation as Modularity

- Queries/network monitoring.
- ARP


Controller 1
Controller 2
A1
A2
R1
R2
H11
H12
H21
A1
A2
R2
R3
H22
H31
H32
PORT == 80
PORT != 80

One approach: like SFI

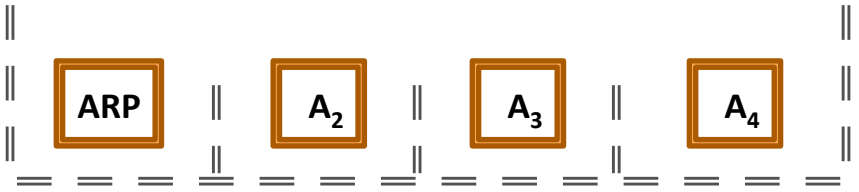
- Draw a box around each network program and prevent them from broaching their respective boxes (slices).
  - Absolute.
  - Says nothing about what happens within a slice.
- FlowVisor takes this approach.

Problem: Very Coarse-grained

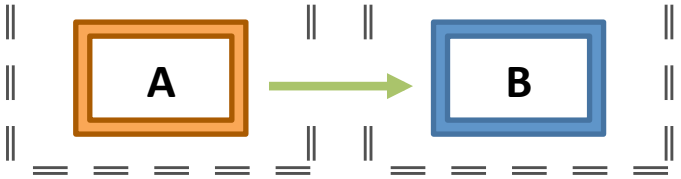
- 1: We want isolation and semantics preserving by construction.
- 2: We want read-only slices.
  - Consider an admin/billing slice that monitors use. Isolation is too strong, but without isolation, what do we have?
- 3: Isolation as modularity.

Limitations

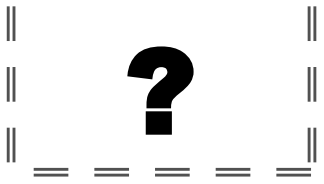
- Isolation as modularity.



- Inter-slice interaction.



- Intra-slice semantics.



Splendid Isolation

Cole Schlesinger

PRINCETON UNIVERSITY

HotSDN

Aug. 2012

Joint work with:


Cornell University
Stephen Gutz
Alec Story
Nate Foster

PRINCETON UNIVERSITY
David Walker