Verifying Reachability for Stateful Networks

Aurojit Panda, Ori Lahav, Katerina Argyraki, Mooly Sagiv, Scott Shenker

UC Berkeley, MPI-SWS, TAU, ICSI

Stateless vs Stateful Networks

Stateless

- Packets forwarded based on static rules.
- Rules change slowly in response to:
  - Changes in topology.
  - Changes in policy.

Stateless vs Stateful Networks

Stateless

- Packets forwarded based on static rules.
- Rules change slowly in response to:
  - Changes in topology.
  - Changes in policy.

Stateful

- Forwarding depends on rules and state.

Stateless vs Stateful Networks

Stateless

- Packets forwarded based on static rules.
- Rules change slowly in response to:
  - Changes in topology.
  - Changes in policy.

Stateful

- Forwarding depends on rules and state.
- Rules change slowly (same as before).

Stateless vs Stateful Networks

Stateless

- Packets forwarded based on static rules.
- Rules change slowly in response to:
  - Changes in topology.
  - Changes in policy.

Stateful

- Forwarding depends on rules and state.
- Rules change slowly (same as before).
- State changes at packet scales:

Stateless vs Stateful Networks

Stateless

- Packets forwarded based on static rules.
- Rules change slowly in response to:
  - Changes in topology.
  - Changes in policy.

Stateful

- Forwarding depends on rules and state.
- Rules change slowly (same as before).
- State changes at packet scales:
  - Every time a connection is established.

Stateless vs Stateful Networks

Stateless

- Packets forwarded based on static rules.
- Rules change slowly in response to:
  - Changes in topology.
  - Changes in policy.

Stateful

- Forwarding depends on rules and state.
- Rules change slowly (same as before).
- State changes at packet scales:
  - Every time a connection is established.
  - Every time packet is forwarded.

Why consider *stateful* networks?

Networks are Increasingly Stateful

- Middleboxes: 1/3rd of all network devices in enterprises (SIGCOMM'12)

Networks are Increasingly Stateful

- Middleboxes: 1/3rd of all network devices in enterprises (SIGCOMM’12)
- Network function virtualization: Simplifies NF deployment.

Networks are Increasingly Stateful

- Middleboxes: 1/3rd of all network devices in enterprises (SIGCOMM’12)
- Network function virtualization: Simplifies NF deployment.
- Programmable switches (P4) also support state.

Networks are Increasingly Stateful

- Middleboxes: 1/3rd of all network devices in enterprises (SIGCOMM'12)
- Network function virtualization: Simplifies NF deployment.
- Programmable switches (P4) also support state.

Not supported by most existing verification tools.

State impacts invariants

Invariants We Consider

- This work focuses on **reachability** and **isolation** invariants.

Invariants We Consider

- This work focuses on **reachability** and **isolation** invariants.
  - Can packets from host A reach host B?

Invariants We Consider

- This work focuses on **reachability** and **isolation** invariants.
  - Can packets from host A reach host B?
- But the addition of state raises some important issues:

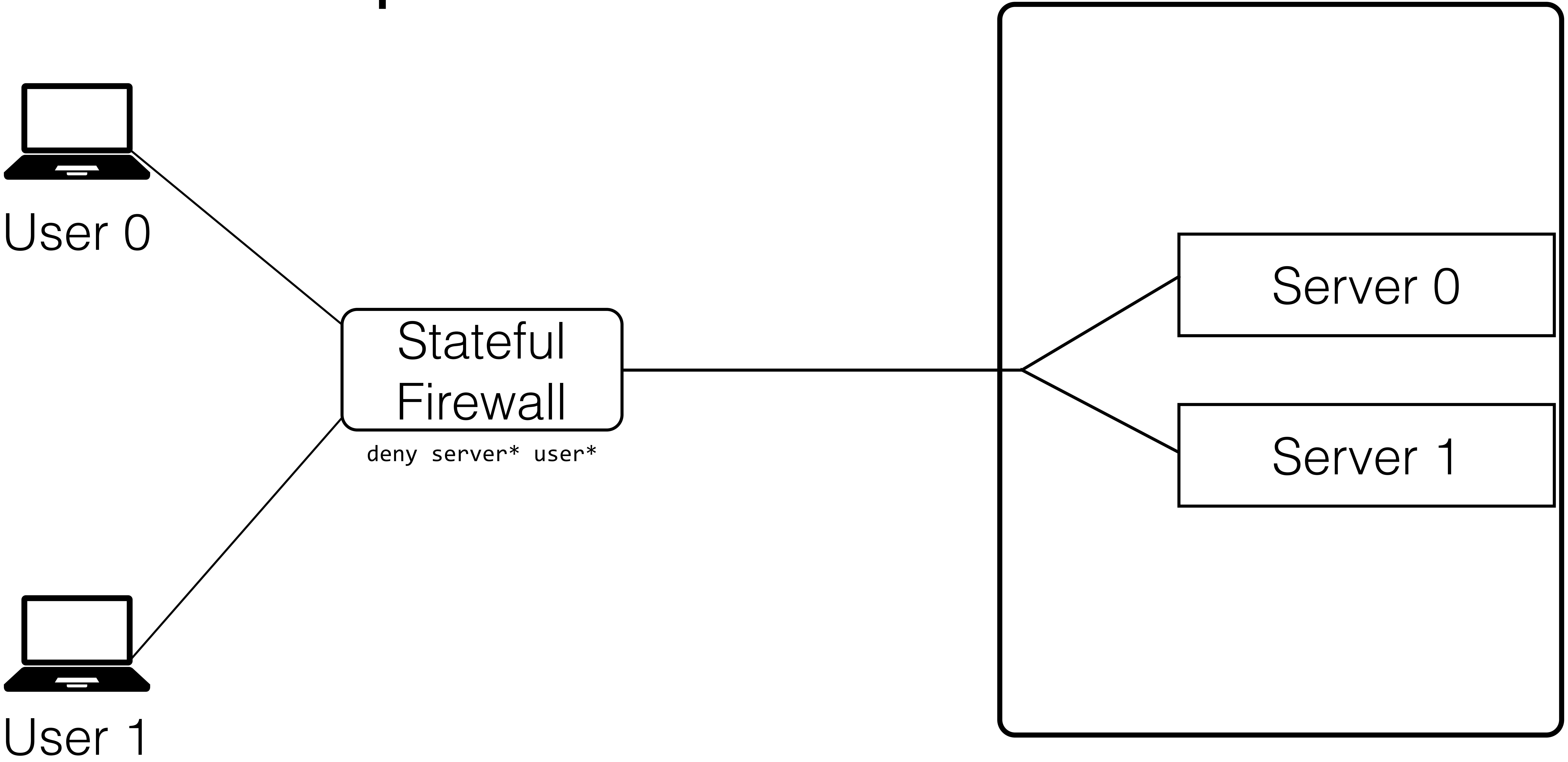
Invariants We Consider

- This work focuses on **reachability** and **isolation** invariants.
  - Can packets from host A reach host B?
- But the addition of state raises some important issues:
  - Invariants can include temporal aspects.

Invariants We Consider

- This work focuses on **reachability** and **isolation** invariants.
    - Can packets from host A reach host B?
- But the addition of state raises some important issues:
    - Invariants can include temporal aspects.
    - Might need to consider more than just packets.

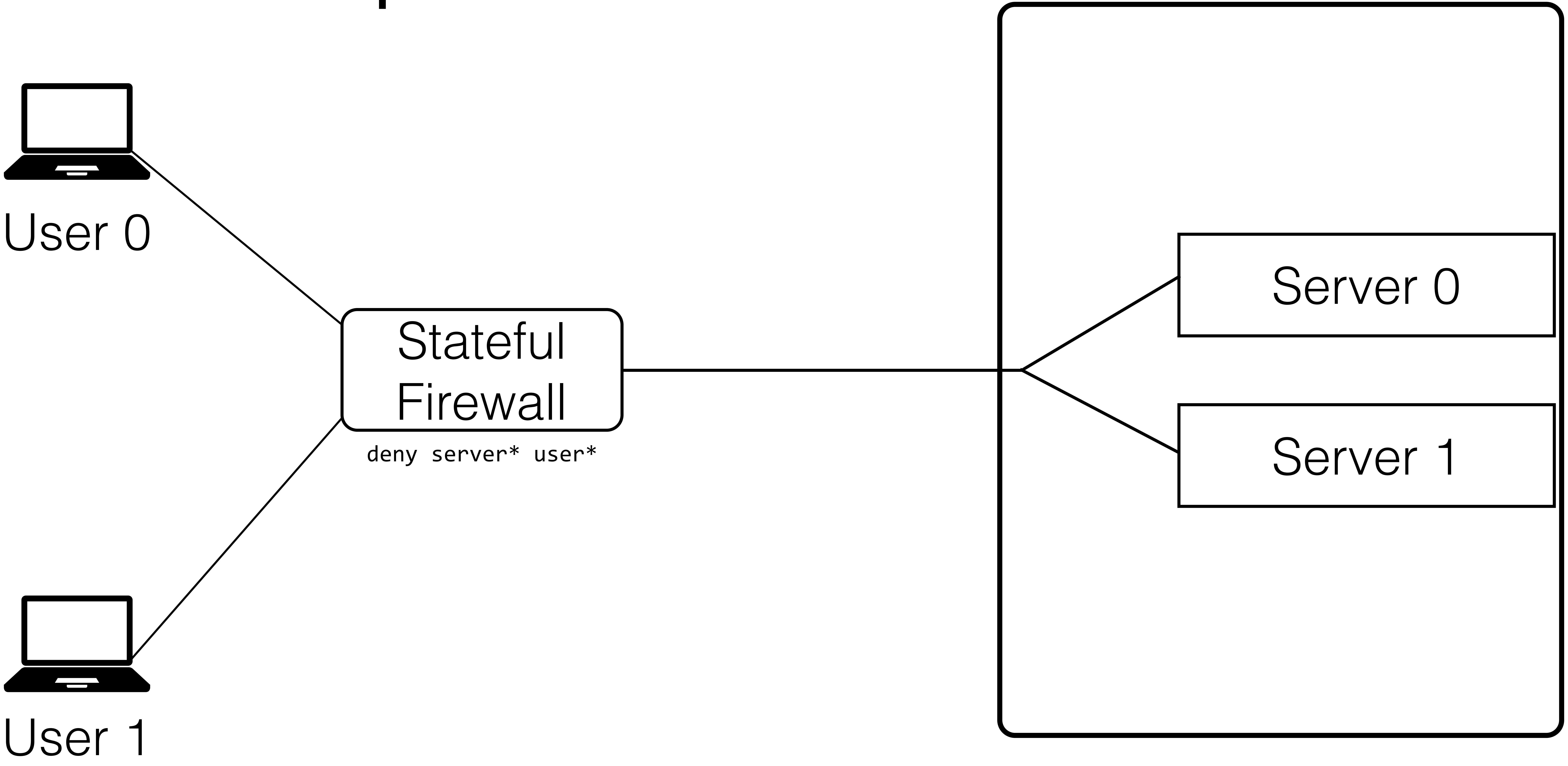
Temporal Invariants



User 1 receives no packets from server 0

Standard Reachability

Temporal Invariants

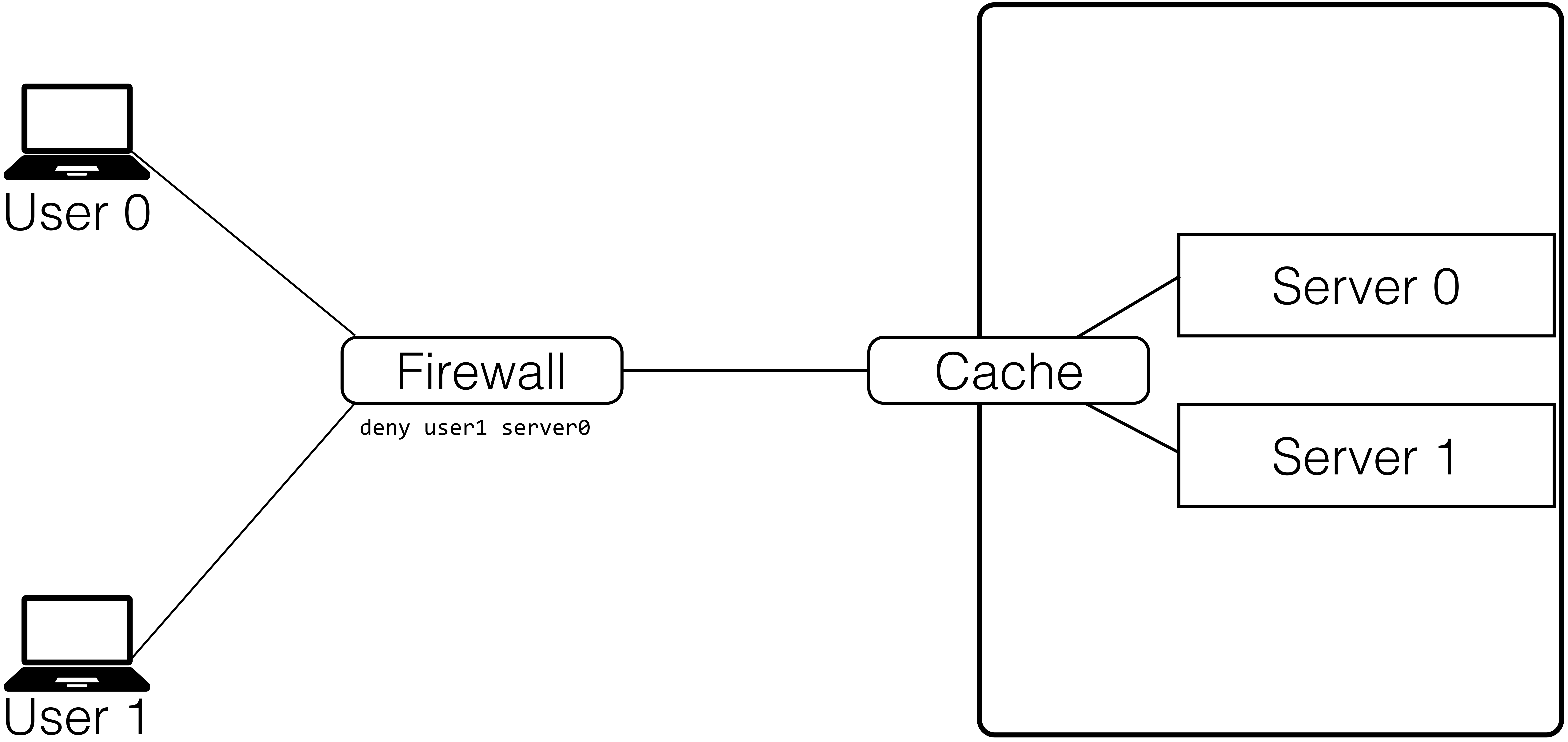


User 1 receives no packets from server 0 without initiating a connection

Standard Reachability

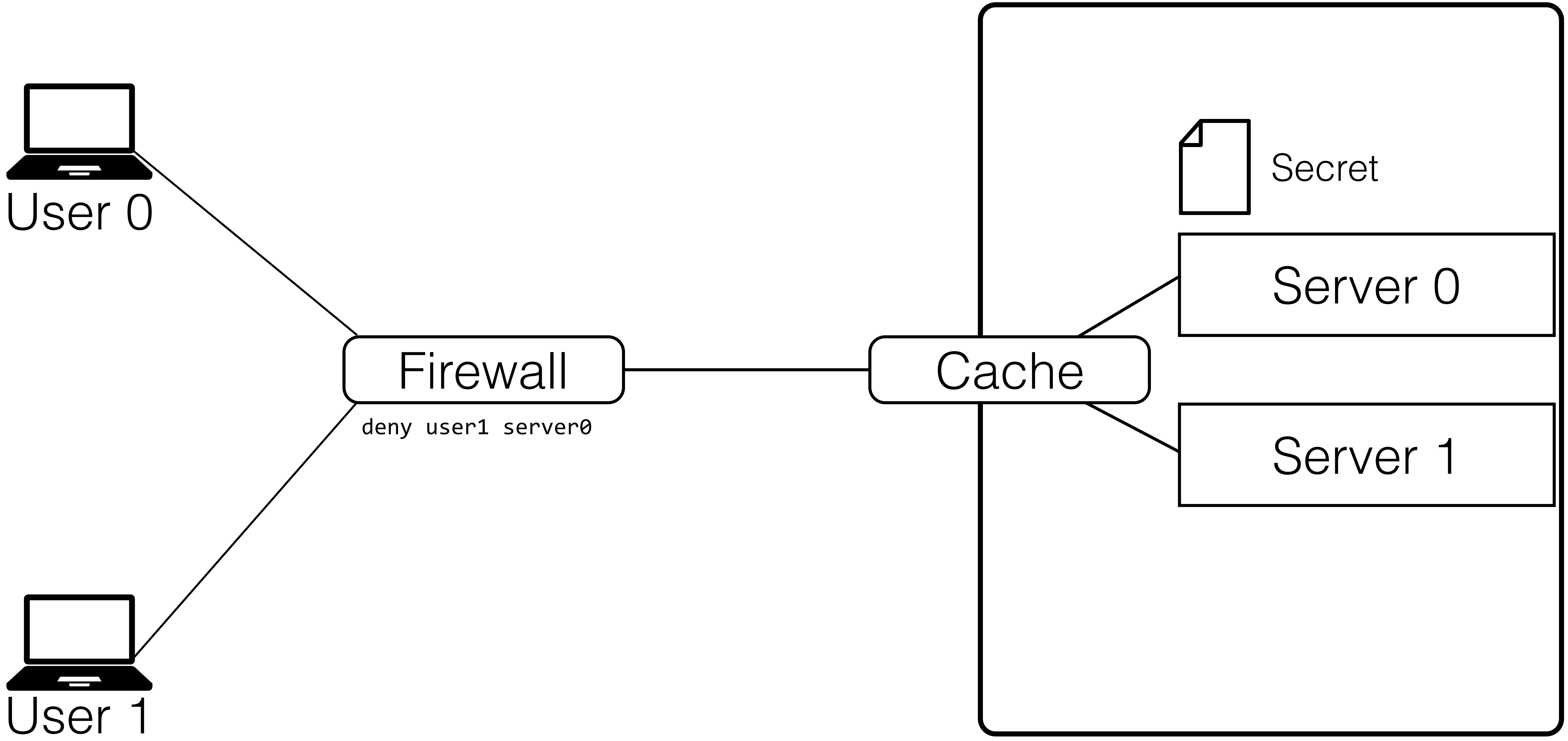
Temporal Property

Consider Data Instead of Packets



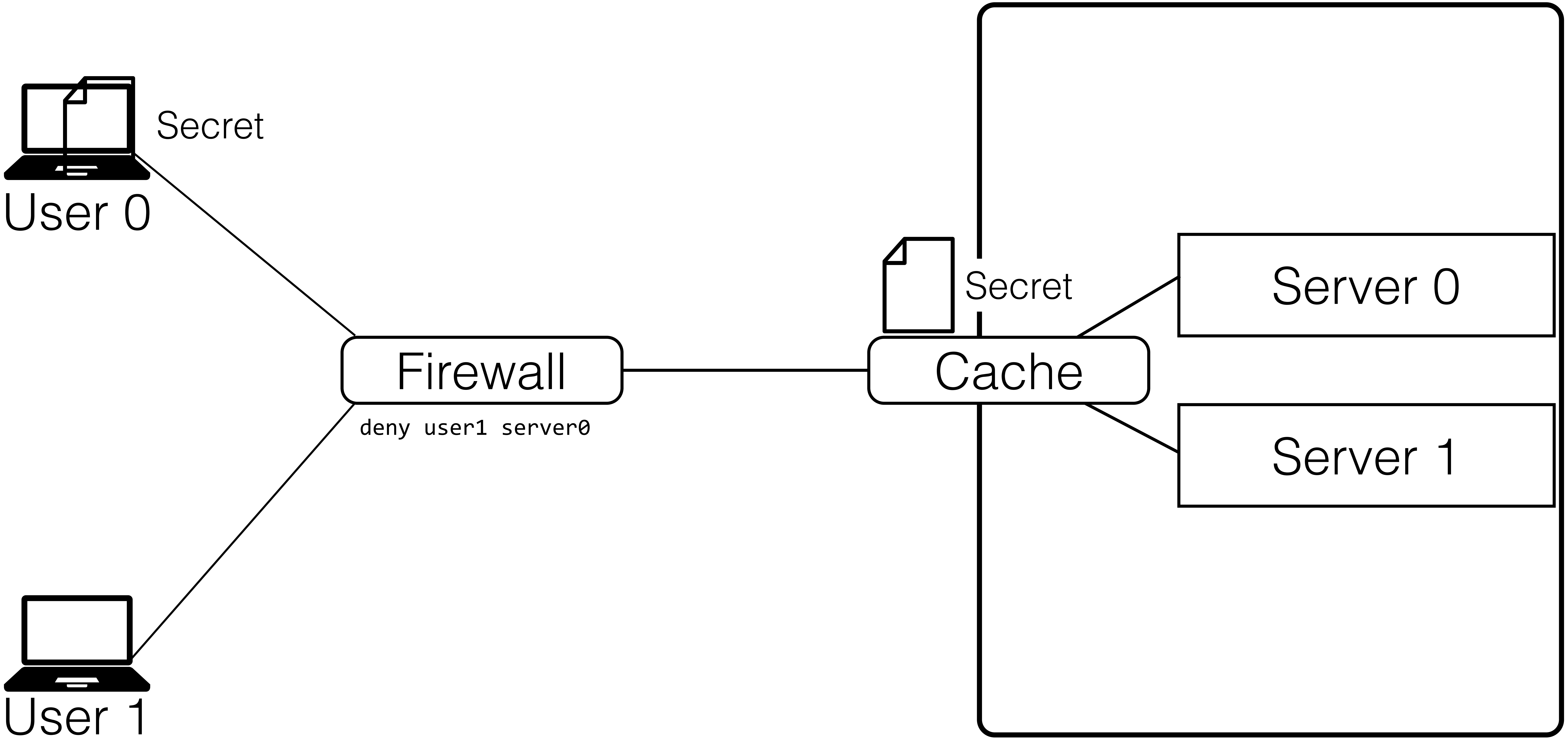
User 1 receives no packet from Server 0

Consider Data Instead of Packets



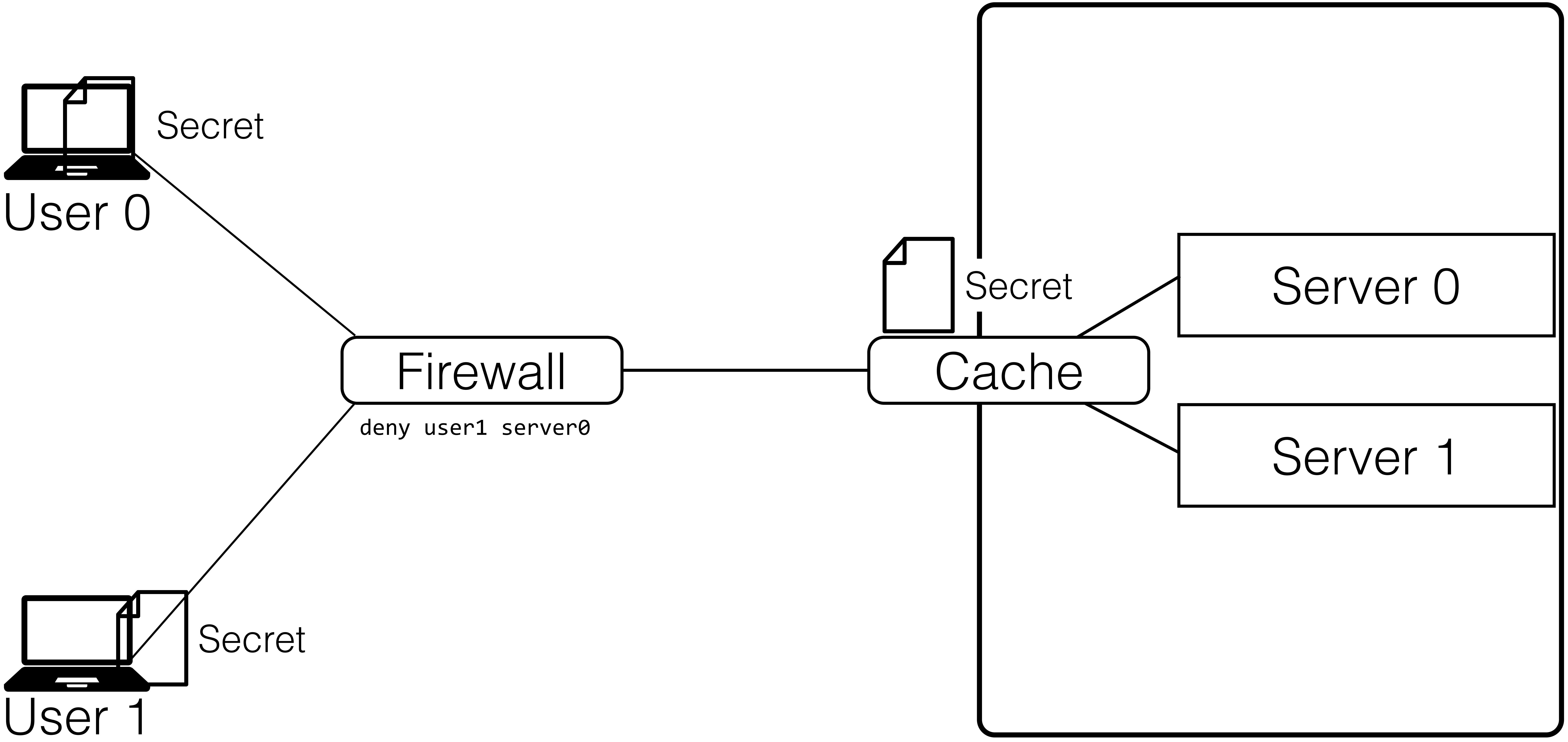
User 1 receives no packet from Server 0

Consider Data Instead of Packets



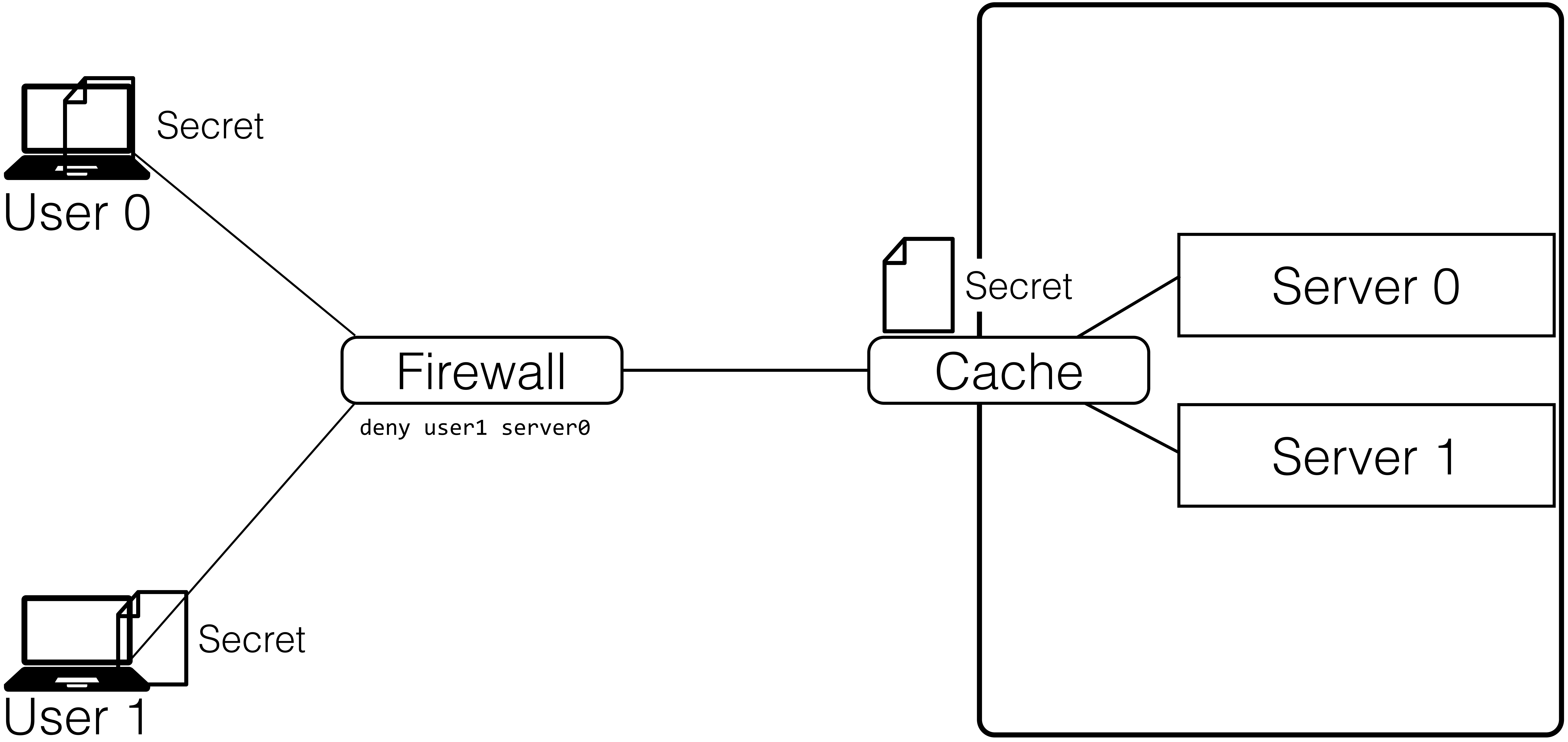
User 1 receives no packet from Server 0

Consider Data Instead of Packets



User 1 receives no packet from Server 0

Consider Data Instead of Packets



~~User 1 receives no packet from Server 0~~

User 1 receives no data from Server 0

Roadmap

- ~~Why stateful networks, and how does state affect invariants?~~
- Existing work on network verification.
- VMN: Our system for verifying networks with state.
- Scaling verification.

Network Verification Today

- Switches and Controllers: Static forwarding rules in switches.

    HSA, Veriflow, NetKAT, Vericon, FlowLog, etc.

Network Verification Today

- Switches and Controllers: Static forwarding rules in switches.

  HSA, Veriflow, NetKAT, Vericon, FlowLog, etc.

- Testing for networks with mutable datapaths

  Buzz: Generate packets that are likely to trigger interesting behavior.

Network Verification Today

- Switches and Controllers: Static forwarding rules in switches.

  HSA, Veriflow, NetKAT, Vericon, FlowLog, etc.

- Testing for networks with mutable datapaths

  Buzz: Generate packets that are likely to trigger interesting behavior.

- Verification for networks with mutable datapaths

  SymNet: Uses symbolic execution, limited state and behaviors.

Roadmap

- ~~Why stateful networks, and how does state affect invariants?~~
- ~~Existing work on network verification.~~
- VMN: Our system for verifying networks with state.
- Scaling verification.

VMN: System for **scalable** verification of **stateful networks.**

VMN Flow

Model each middlebox in the network

↓

Build network forwarding model

↓

Logical Invariants

↓

SMT Solver (Z3 from MSR)

↓ Invariant Holds

↓ Example of violation

VMN Flow

Model each middlebox in the network

↓

Build network forwarding model

↓

Logical Invariants

↓

SMT Solver (Z3 from MSR)

↓ Invariant Holds

↓ Example of violation

Modeling Middleboxes

- One approach: Extract model from code

Modeling Middleboxes

- One approach: Extract model from code
- **Problem**: At the wrong level of abstraction.

Modeling Middleboxes

- One approach: Extract model from code
- **Problem**: At the wrong level of abstraction.
  - **Code** written to match bit patterns in packet, etc.

Modeling Middleboxes

- One approach: Extract model from code
- **Problem**: At the wrong level of abstraction.
  - **Code** written to match bit patterns in packet, etc.
  - **Configuration** is in terms of **higher level abstractions**

Modeling Middleboxes

- One approach: Extract model from code
- **Problem**: At the wrong level of abstraction.
    - **Code** written to match bit patterns in packet, etc.
    - **Configuration** is in terms of **higher level abstractions**
        - Example source and destination addresses, payload is infected, etc.

Modeling Middleboxes

- One approach: Extract model from code
- **Problem**: At the wrong level of abstraction.
  - **Code** written to match bit patterns in packet, etc.
  - **Configuration** is in terms of **higher level abstractions**
    - Example source and destination addresses, payload is infected, etc.
- Verify invariants which are also expressed in these terms.

Challenges When Modeling Middleboxes

- Example configuration:

Challenges When Modeling Middleboxes

- Example configuration:

  Drop all packets from connections transmitting infected files.

Challenges When Modeling Middleboxes

- Example configuration:

  Drop all packets from connections transmitting infected files.

- How to **define** infected files: large, growing set of bit patterns.

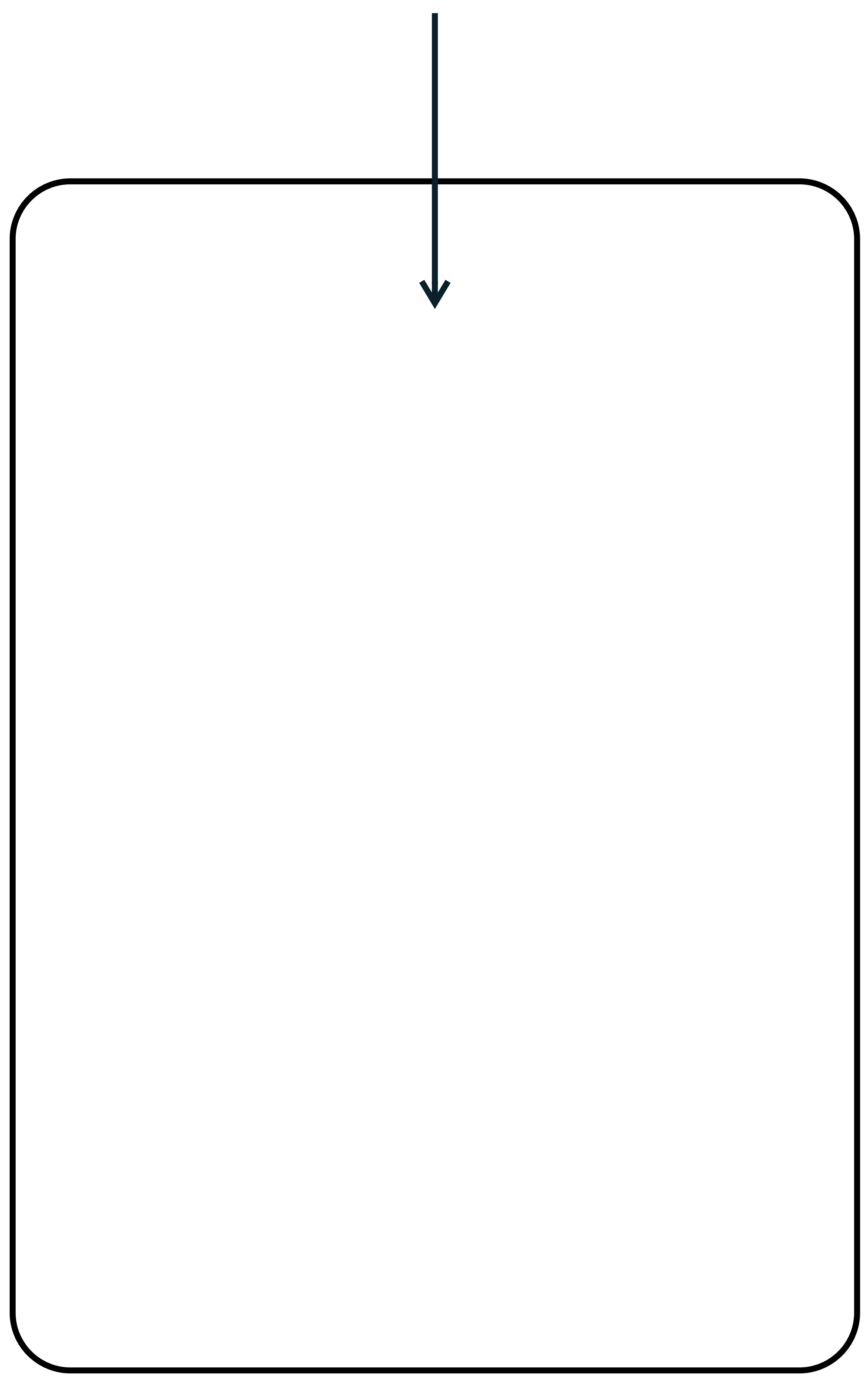
Challenges When Modeling Middleboxes

- Example configuration:

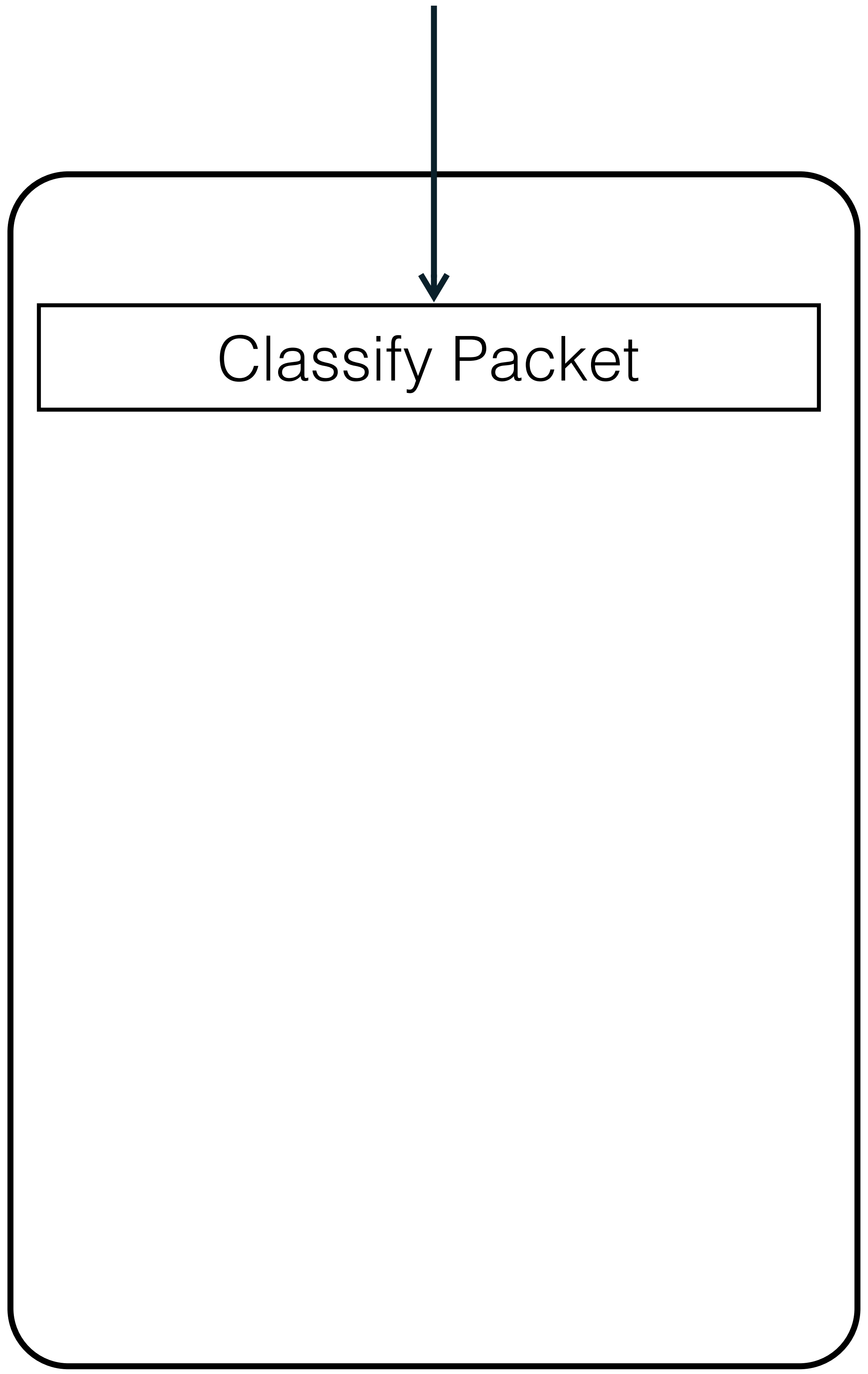
    Drop all packets from connections transmitting infected files.

- How to **define** infected files: large, growing set of bit patterns.
- Complexity of matching code prevents verification in even small networks.

Modeling Middleboxes

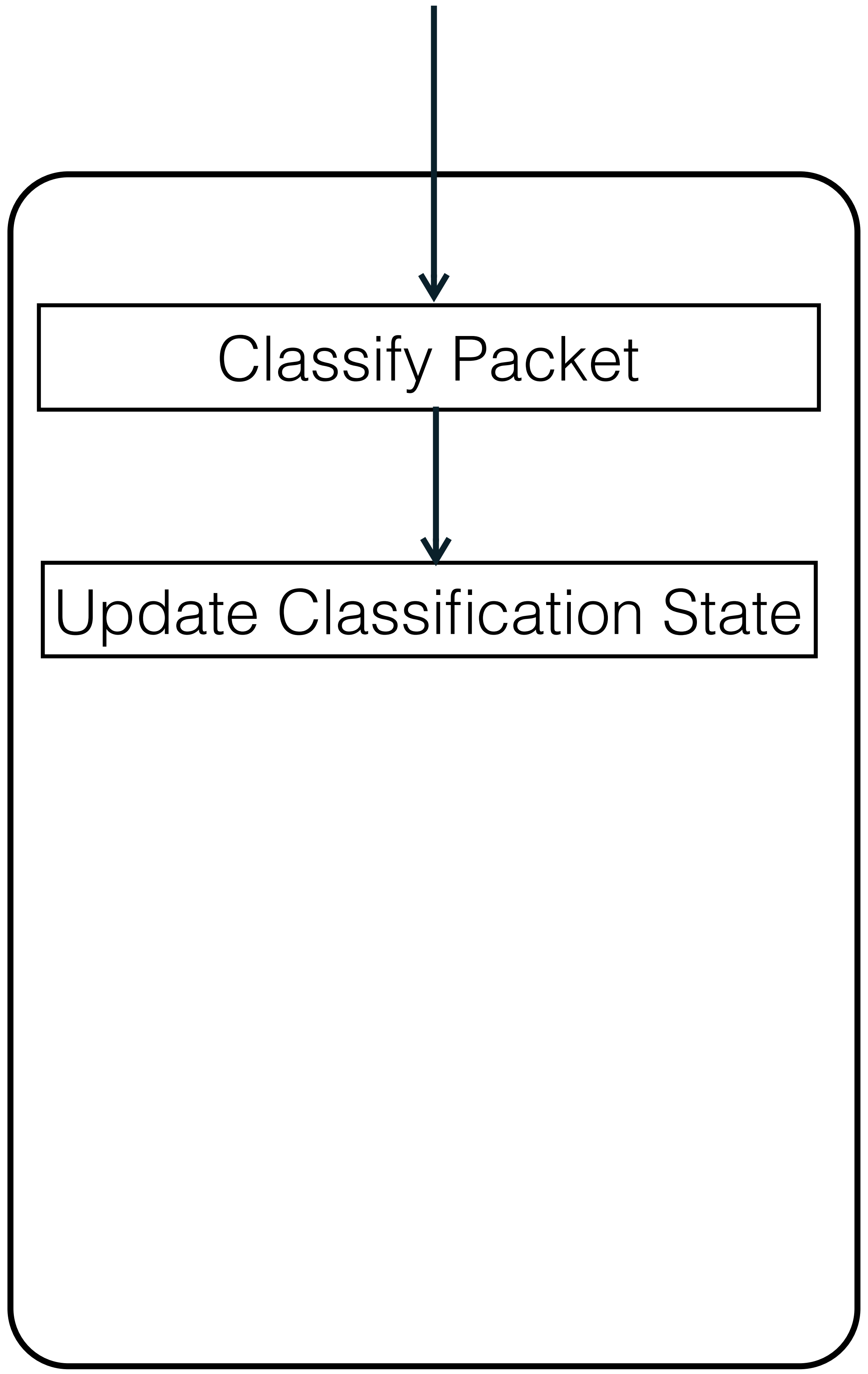


Modeling Middleboxes



Determines what application sent a packet, etc.
Complex, proprietary processing.

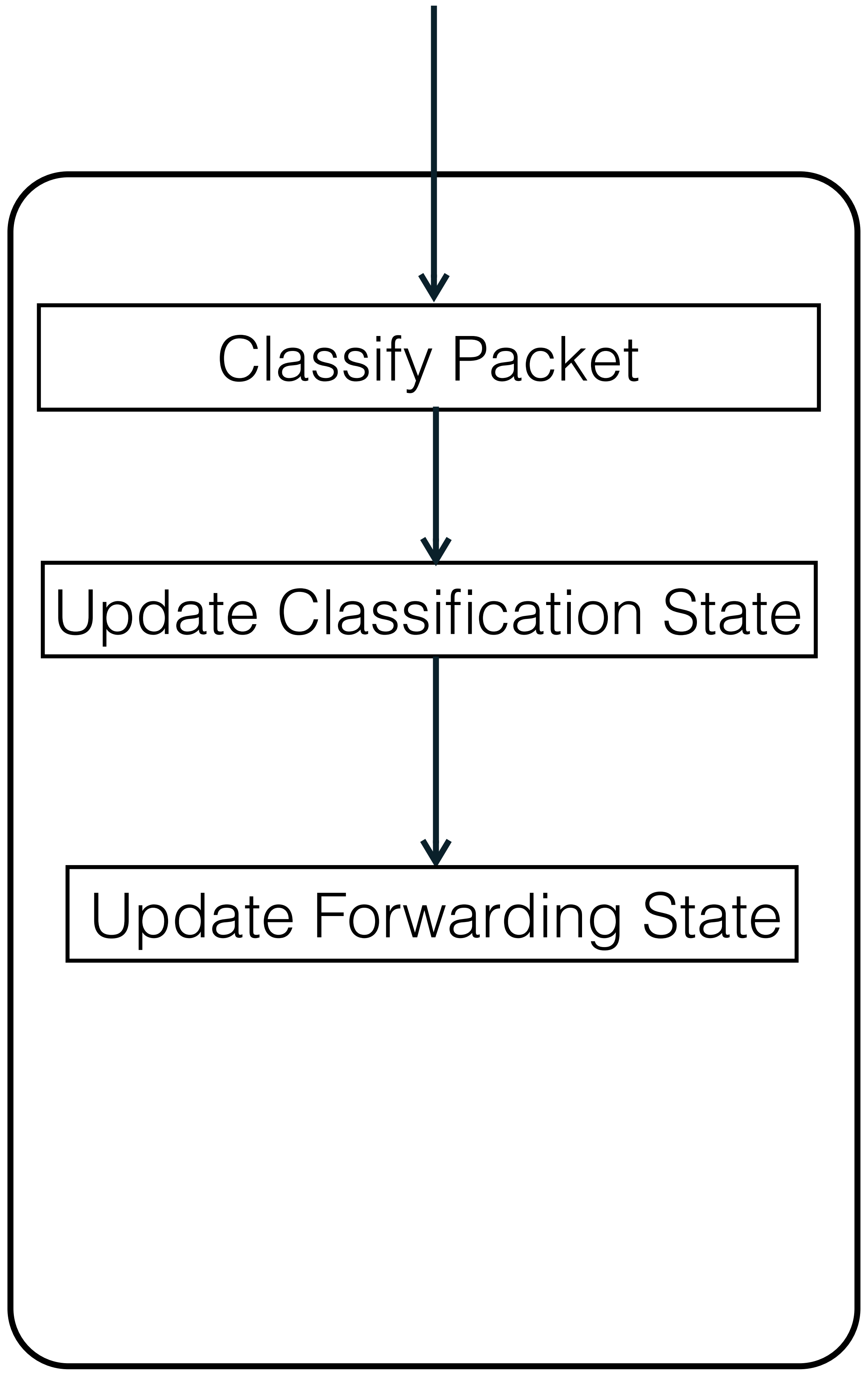
Modeling Middleboxes



Determines what application sent a packet, etc.
Complex, proprietary processing.

Update state required for classification.

Modeling Middleboxes

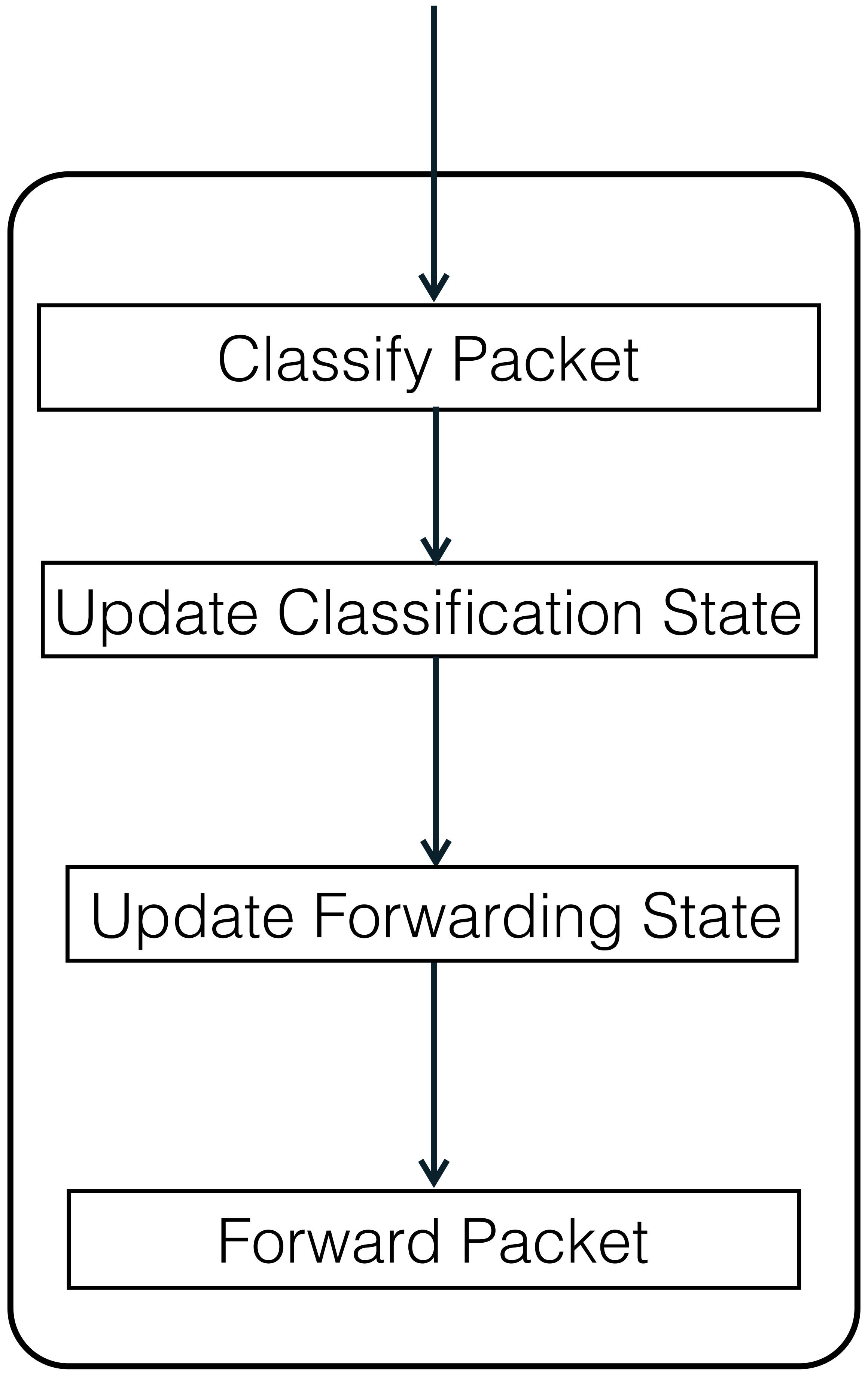


Determines what application sent a packet, etc.
Complex, proprietary processing.

Update state required for classification.

Update forwarding State.

Modeling Middleboxes



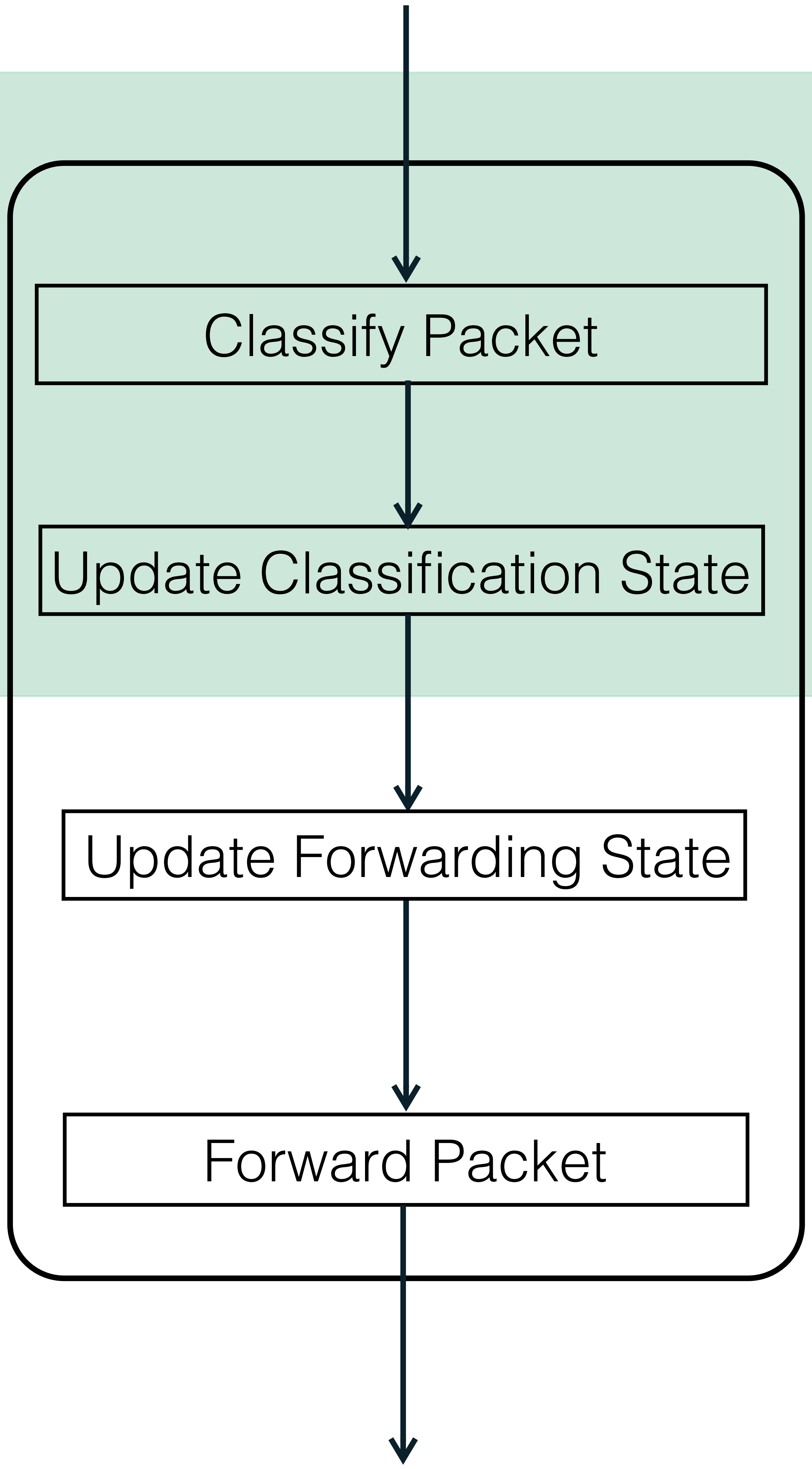
Determines what application sent a packet, etc.
Complex, proprietary processing.

Update state required for classification.

Update forwarding State.

Always simple: forward or drop packets.

Modeling Middleboxes



Oracle: Specify data dependencies and outputs

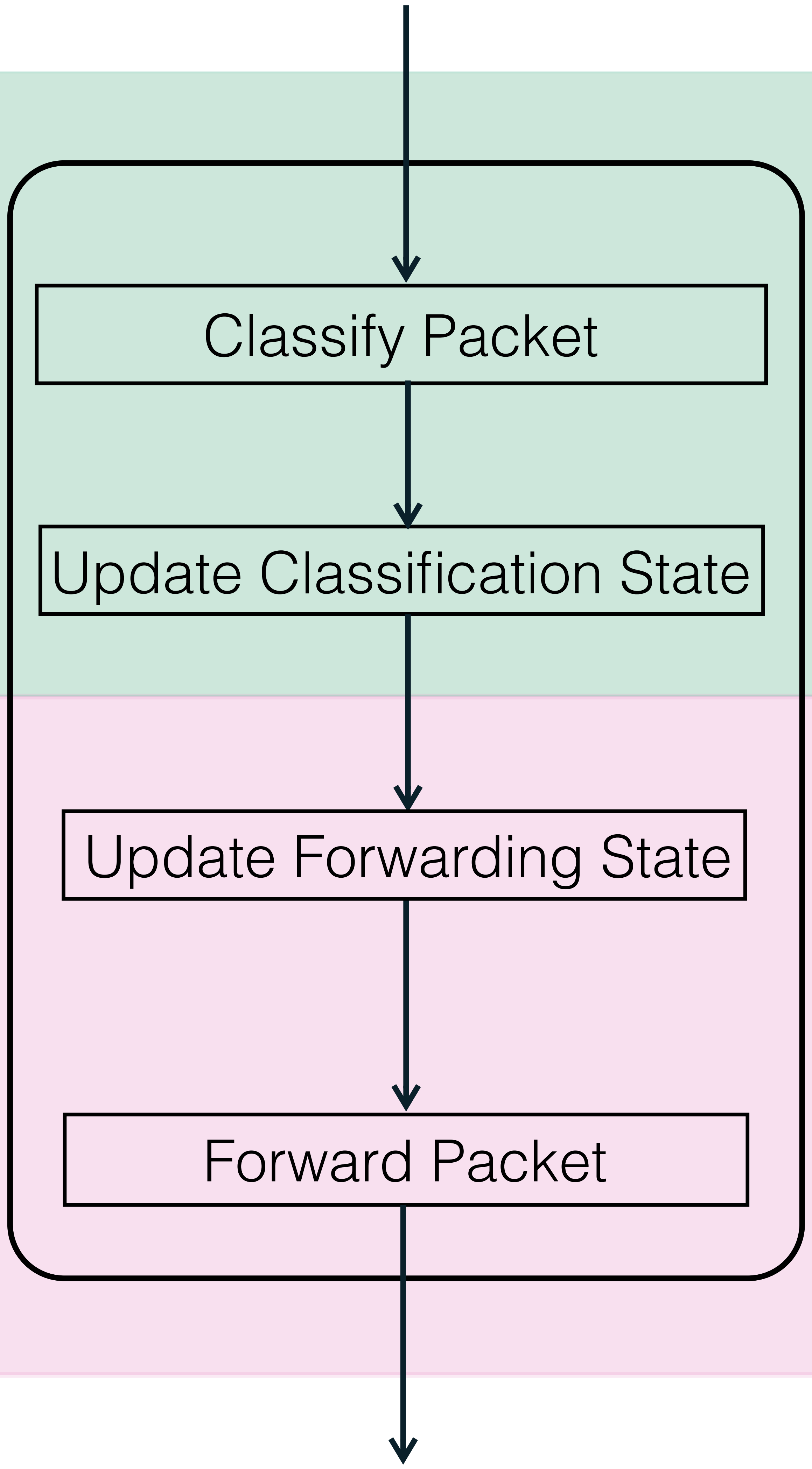
Determines what application sent a packet, etc.
Complex, proprietary processing.

Update state required for classification.

Update forwarding State.

Always simple: forward or drop packets.

Modeling Middleboxes



Oracle: Specify data dependencies and outputs

Determines what application sent a packet, etc.
Complex, proprietary processing.

Update state required for classification.

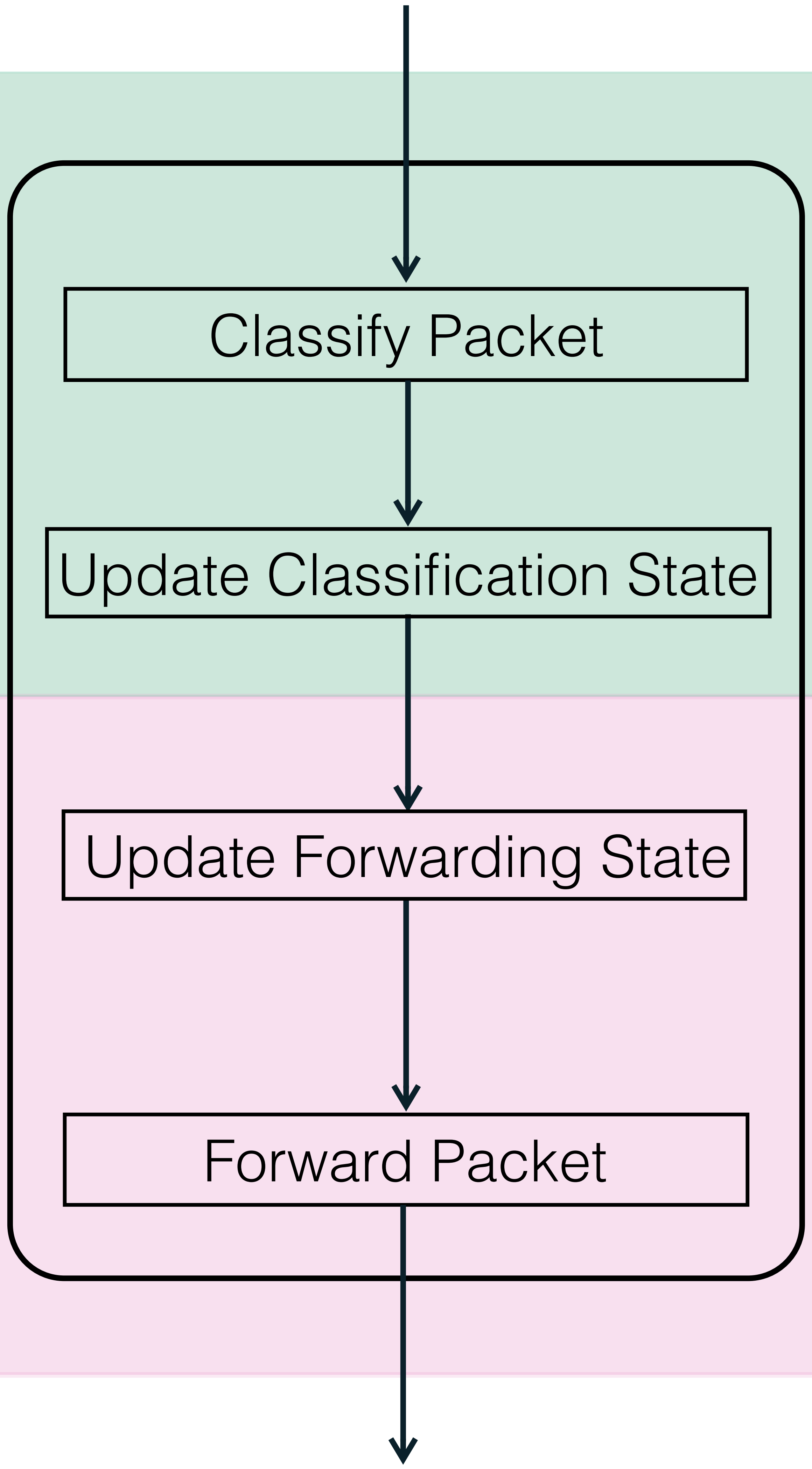
Update forwarding State.

Always simple: forward or drop packets.

Forwarding Model: Specify Completely

Modeling Middleboxes

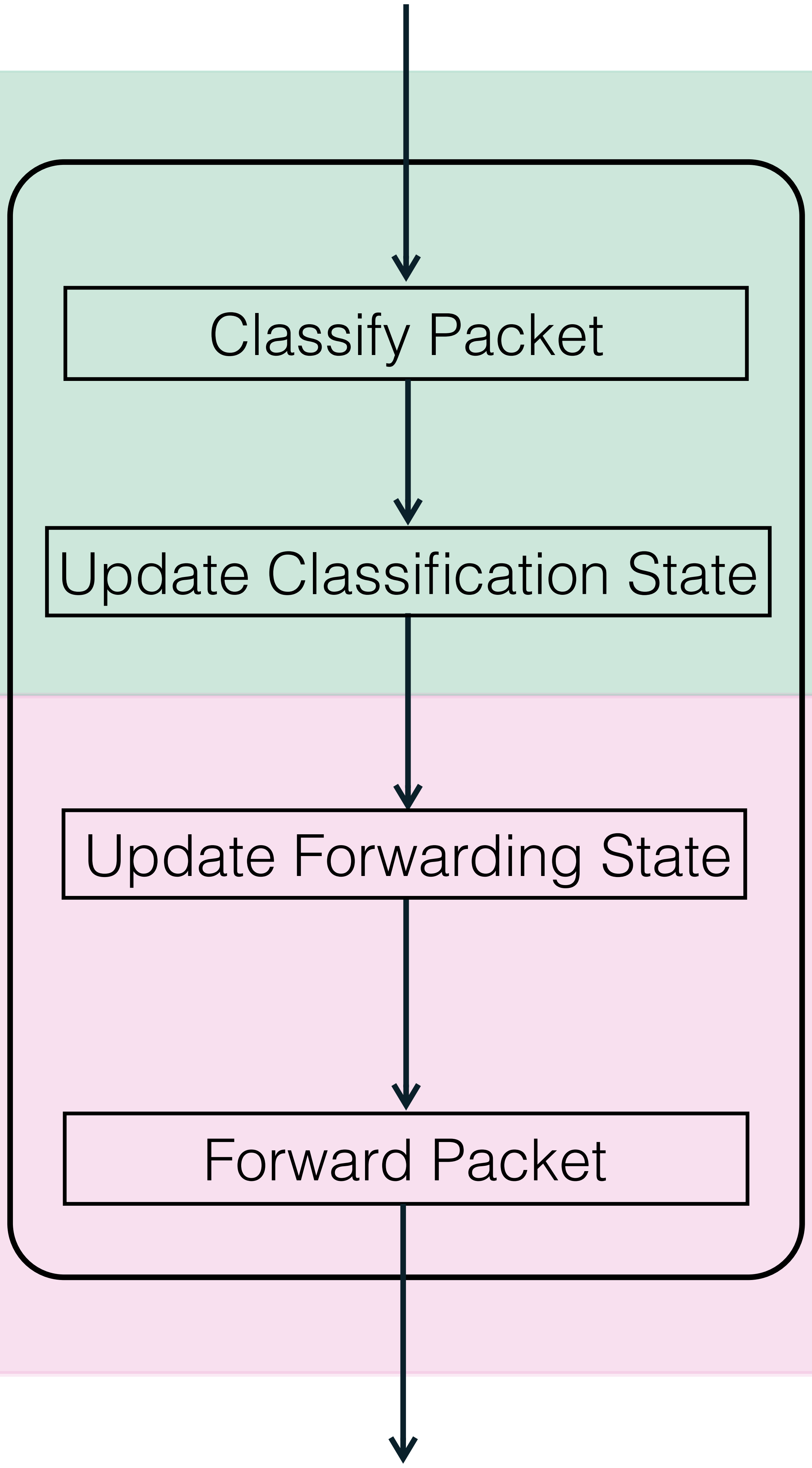
Modeling Middleboxes



Dependencies
See all packets in connection (flow).

Outputs
Is packet **infected**.

Modeling Middleboxes

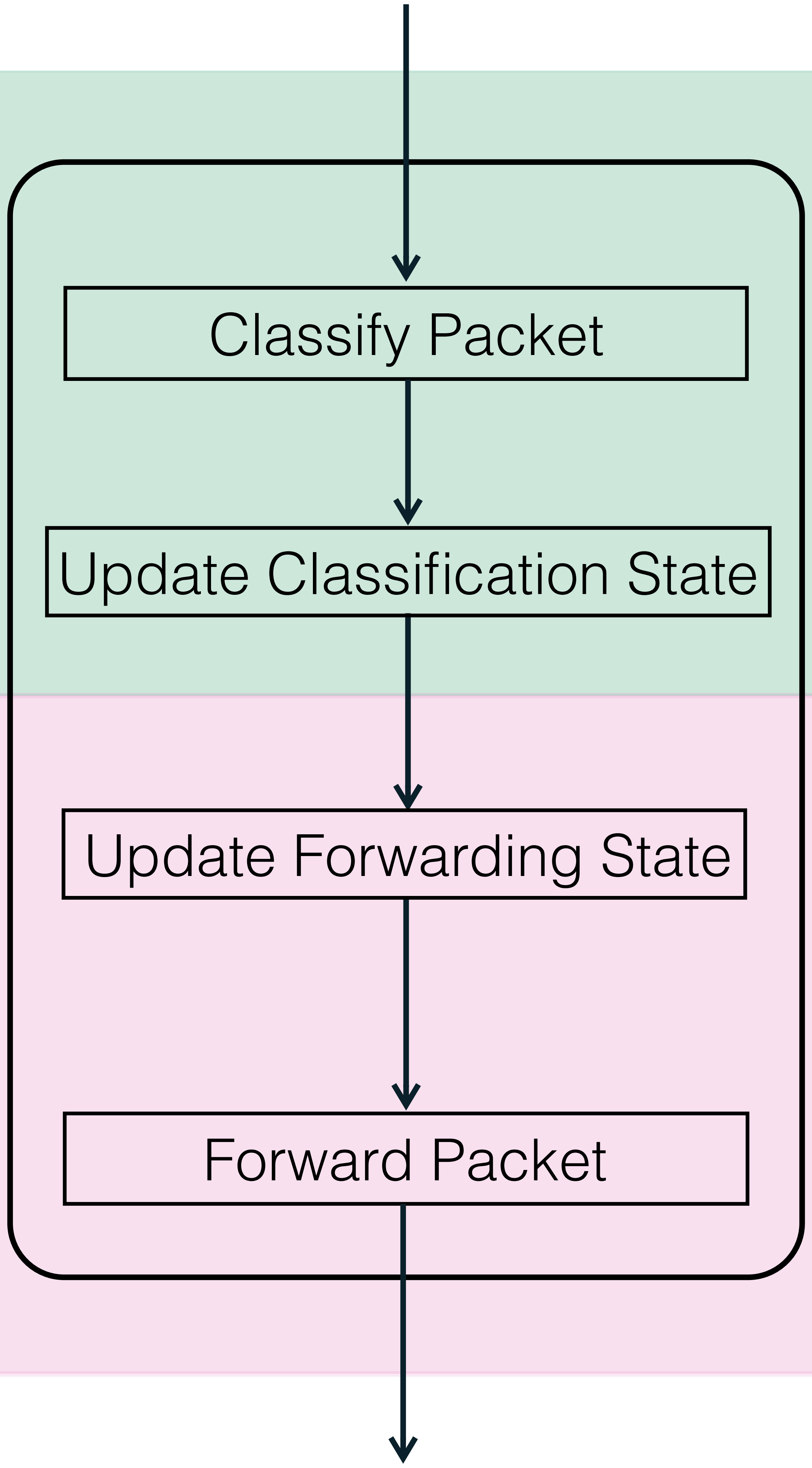


Dependencies
See all packets in connection (flow).

Outputs
Is packet **infected**.

```
if (infected) {
    infected_connections.add(packet.flow)
}
```

Modeling Middleboxes



Dependencies
See all packets in connection (flow).

Outputs
Is packet **infected**.

if (**infected**) {
    infected_connections.add(packet.flow)
}

if (**packet.flow** not in **infected_connections**) {
    forward (packet);
}

Sample Model

```
class Firewall (acls: Set[(Address, Address)])
{
  abstract malicious(p: Packet): bool
  val tainted: Set[Address]
  def model (p: Packet) = {
    tainted.contains(p.src) => forward(Empty)
    acls.contains((p.src, p.dst)) =>
         forward(Empty)
    malicious(p) => tainted.add(p.src);
         forward(Empty)
    _ => forward(Seq(p))
  }
}
```

Sample Model

```
class Firewall (acls: Set[(Address, Address)])
{
  abstract malicious(p: Packet): bool
  val tainted: Set[Address]
  def model (p: Packet) = {
    tainted.contains(p.src) => forward(Empty)
    acls.contains((p.src, p.dst)) =>
         forward(Empty)
    malicious(p) => tainted.add(p.src);
         forward(Empty)
    _ => forward(Seq(p))
  }
}
```

Oracle

Sample Model

```
class Firewall (acls: Set[(Address, Address)])
{
  abstract malicious(p: Packet): bool
  val tainted: Set[Address]
  def model (p: Packet) = {
    tainted.contains(p.src) => forward(Empty)
    acls.contains((p.src, p.dst)) =>
         forward(Empty)
    malicious(p) => tainted.add(p.src);
         forward(Empty)
    _ => forward(Seq(p))
  }
}
```

Oracle

State

Sample Model

```
class Firewall (acls: Set[(Address, Address)])
{
  abstract malicious(p: Packet): bool
  val tainted: Set[Address]
  def model (p: Packet) = {
    tainted.contains(p.src) => forward(Empty)
    acls.contains((p.src, p.dst)) =>
         forward(Empty)
    malicious(p) => tainted.add(p.src);
         forward(Empty)
    _ => forward(Seq(p))
  }
}
```

Oracle

State

Forwarding Model

Network Forwarding Model

- Builds on **network transfer functions**.

Network Forwarding Model

- Builds on **network transfer functions**.
  - Existing work from HSA, Veriflow, etc.

Network Forwarding Model

- Builds on **network transfer functions**.
  - Existing work from HSA, Veriflow, etc.
- Abstracts all switches and routers into **one big switch**.

Network Forwarding Model

- Builds on **network transfer functions**.
  - Existing work from HSA, Veriflow, etc.
- Abstracts all switches and routers into **one big switch**.
- Details in the paper.

Roadmap

- ~~Why consider stateful networks?~~
- ~~The current state of stateful network verification?~~
- ~~VMN: Our system for verifying networks with state.~~
- Scaling verification.

Networks are Large

- Networks are huge in practice

Networks are Large

- Networks are huge in practice
    - For example Google had 900K machines (approximately) in 2011

Networks are Large

- Networks are huge in practice
  - For example Google had 900K machines (approximately) in 2011
  - ISPs connect large numbers of machines.

Networks are Large

- Networks are huge in practice
  - For example Google had 900K machines (approximately) in 2011
  - ISPs connect large numbers of machines.
- Lots of middleboxes in these networks

Networks are Large

- Networks are huge in practice
  - For example Google had 900K machines (approximately) in 2011
  - ISPs connect large numbers of machines.
- Lots of middleboxes in these networks
  - In datacenter each machine might be one or more middlebox.

Networks are Large

- Networks are huge in practice
  - For example Google had 900K machines (approximately) in 2011
  - ISPs connect large numbers of machines.
- Lots of middleboxes in these networks
  - In datacenter each machine might be one or more middlebox.
- How do we address this?

Scaling Techniques Thus Far

- Abstract middlebox models

Scaling Techniques Thus Far

- Abstract middlebox models
    - Simplify what needs to be considered per-middlebox.

Scaling Techniques Thus Far

- Abstract middlebox models
  - Simplify what needs to be considered per-middlebox.
- Abstract network

Scaling Techniques Thus Far

- Abstract middlebox models
  - Simplify what needs to be considered per-middlebox.
- Abstract network
  - Simplify network forwarding.

Those Techniques are not Enough

- TACAS 2016: Network verification with state is EXPSPACE-complete.

Those Techniques are not Enough

- TACAS 2016: Network verification with state is EXPSPACE-complete.
- Practically for us SMT solvers timeout with large instances.

Those Techniques are not Enough

- TACAS 2016: Network verification with state is EXPSPACE-complete.
- Practically for us SMT solvers timeout with large instances.
- **Other methods also do not handle such large instances**
  - Symbolic execution is exponential in number of branches, not better.

Those Techniques are not Enough

- TACAS 2016: Network verification with state is EXPSPACE-complete.
- Practically for us SMT solvers timeout with large instances.
- **Other methods also do not handle such large instances**
    - Symbolic execution is exponential in number of branches, not better.
- Our techniques work for small instances, what to do about large instances?

Scaling Verification

- Two techniques: Slicing and symmetry.

Scaling Verification

- Two techniques: Slicing and symmetry.
- **Slicing**: Run verification on a subnetwork of size independent of network.

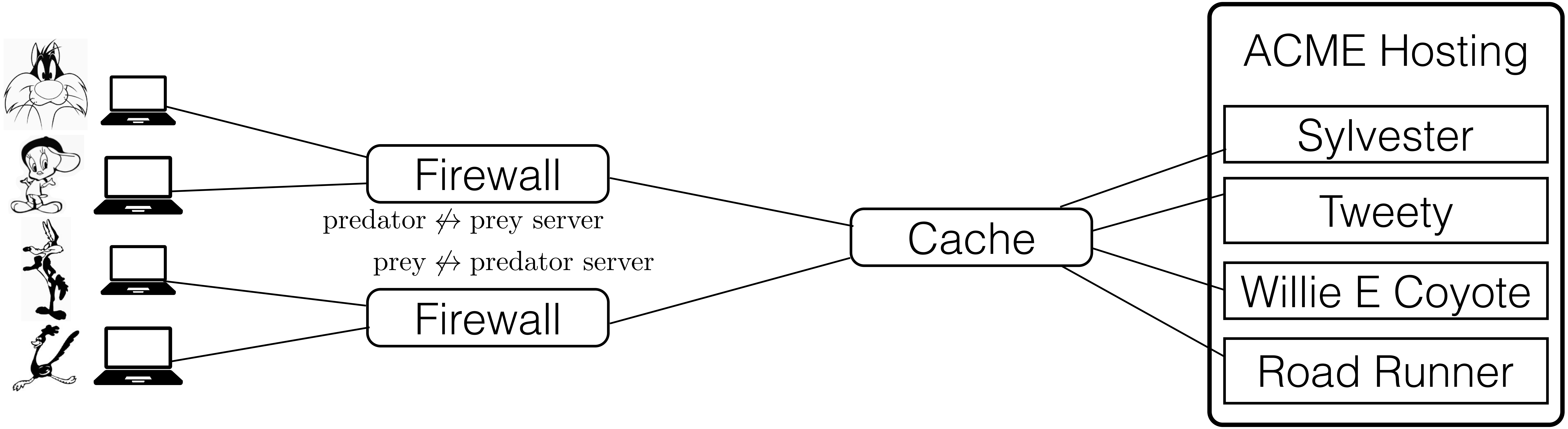
Scaling Verification

- Two techniques: Slicing and symmetry.
- **Slicing**: Run verification on a subnetwork of size independent of network.
- **Symmetry**: Reduce number of invariants to verify by leveraging symmetry in policy.

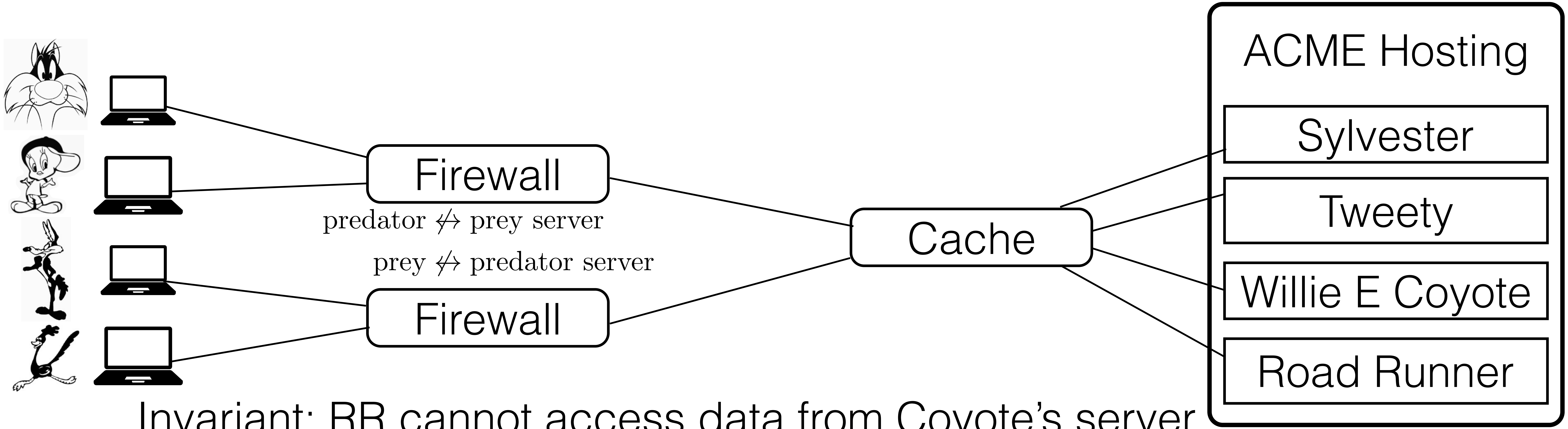
Network Slices

- **Slices:** Subnetworks for which a bisimulation with the original network exists.
  - Ensures equivalent step in subnetwork for each step in the original network
- Slices are selected depending on the invariant being checked.

Network Slices

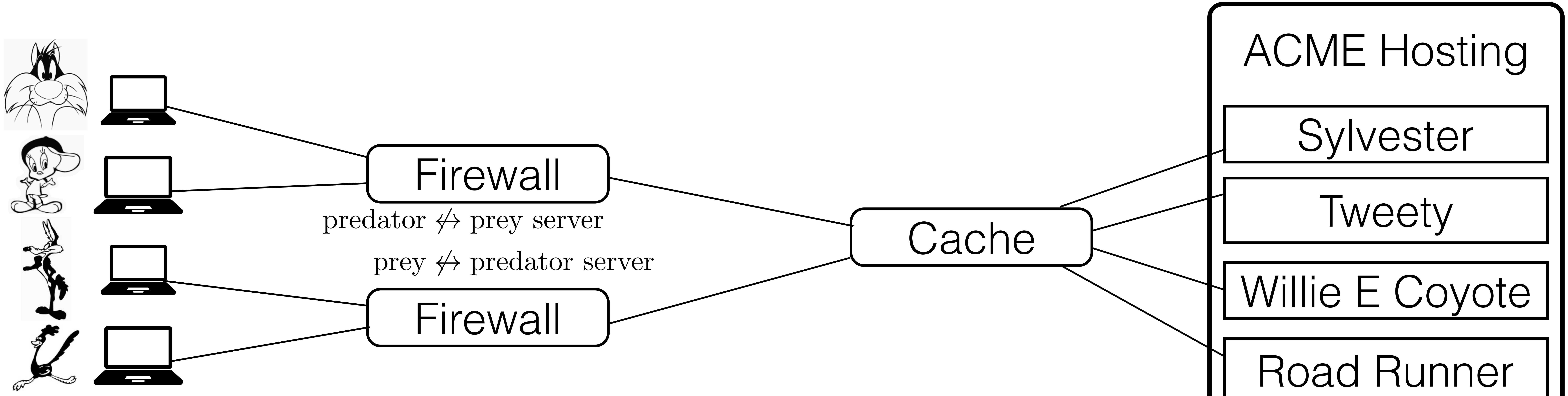


Network Slices



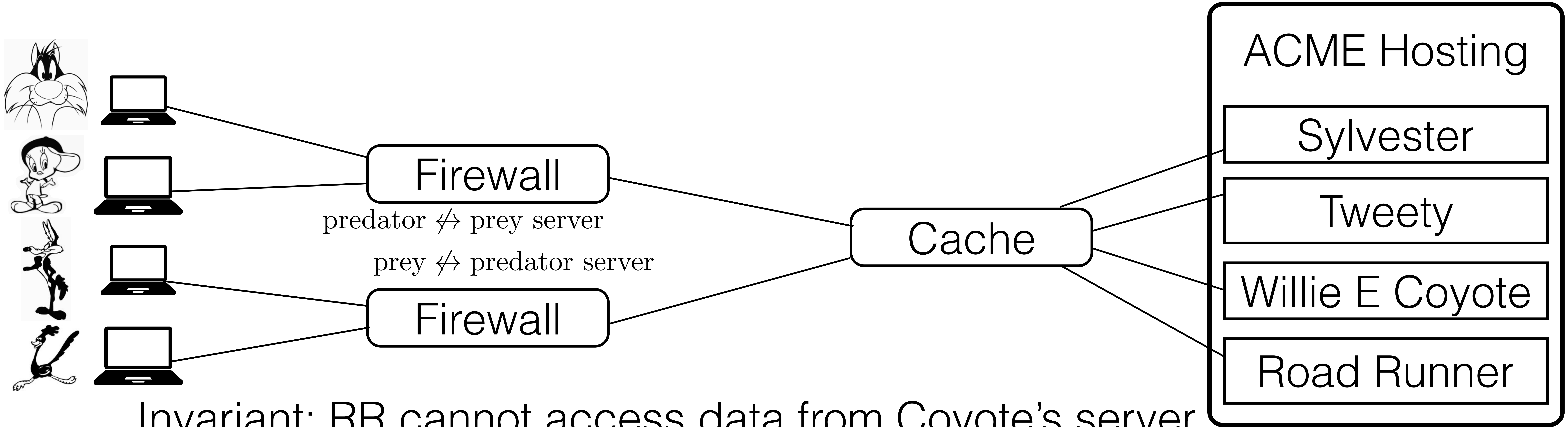
Invariant: RR cannot access data from Coyote's server

Network Slices

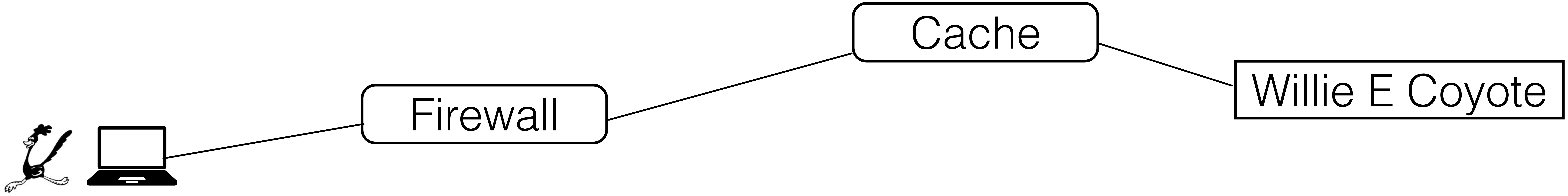


Invariant: RR cannot access data from Coyote's server

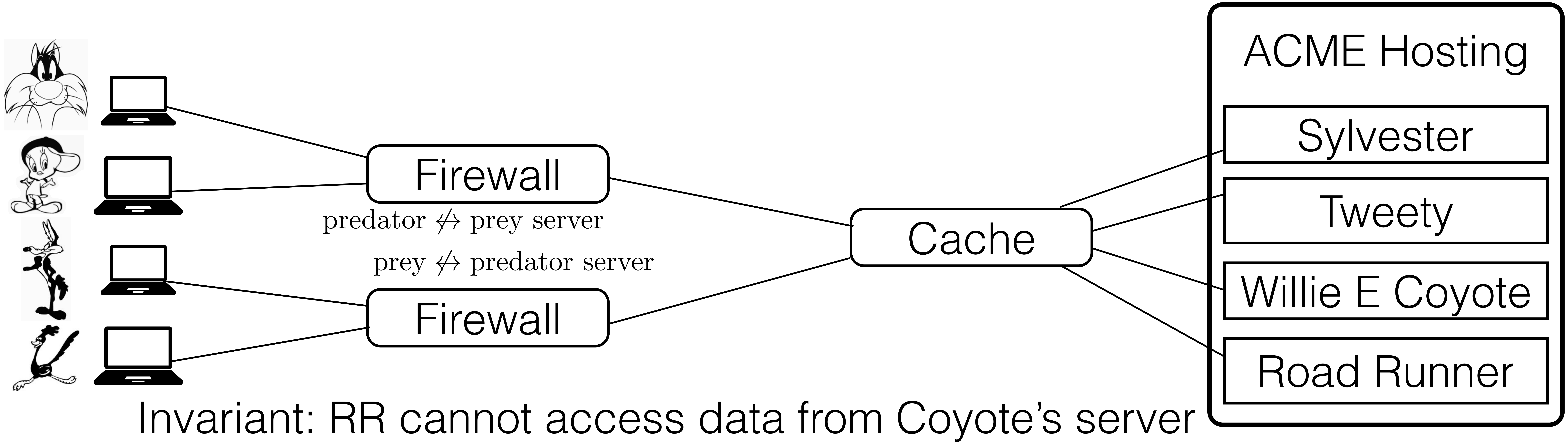
Network Slices

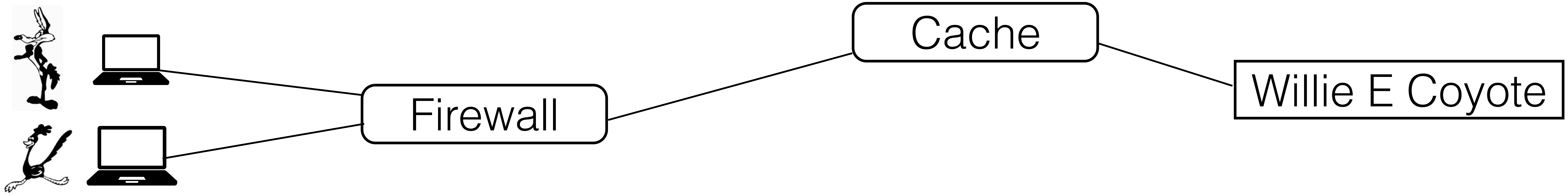


Invariant: RR cannot access data from Coyote's server

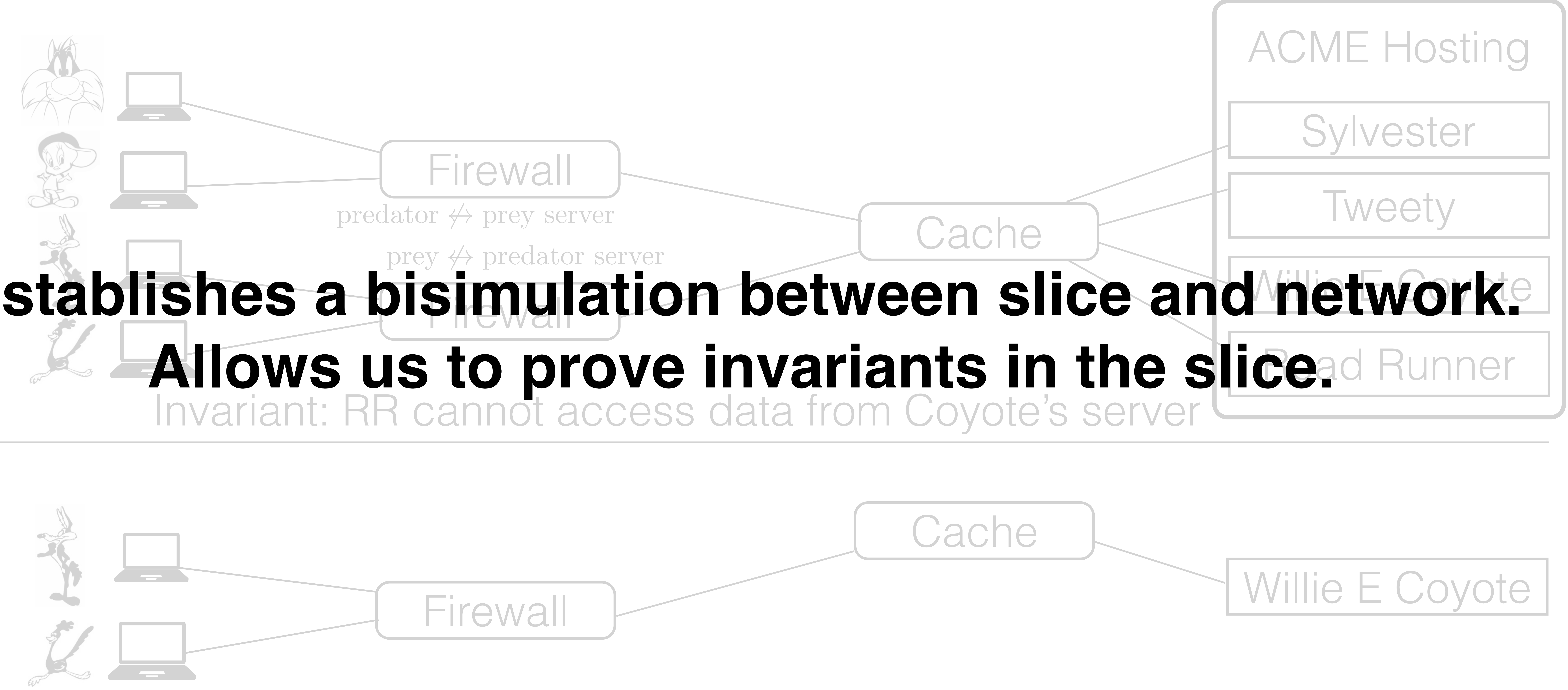


Network Slices





Network Slices



Establishes a bisimulation between slice and network.

Allows us to prove invariants in the slice.

Cannot always find such a slice.

Finding Slices

- **Flow parallel middleboxes** - partition network by flows.
- **Origin agnostic middleboxes** - partition network by policy equivalence class.
- Details in paper.

Evaluation Setup: Datacenter

- Consider AWS like multi-tenant datacenter.

Evaluation Setup: Datacenter

- Consider AWS like multi-tenant datacenter.
- Each tenant has policies for private and public hosts.

Evaluation Setup: Datacenter

- Consider AWS like multi-tenant datacenter.
- Each tenant has policies for private and public hosts.
- Three verification tasks

Evaluation Setup: Datacenter

- Consider AWS like multi-tenant datacenter.
- Each tenant has policies for private and public hosts.
- Three verification tasks
  - Private hosts for one tenant cannot reach another

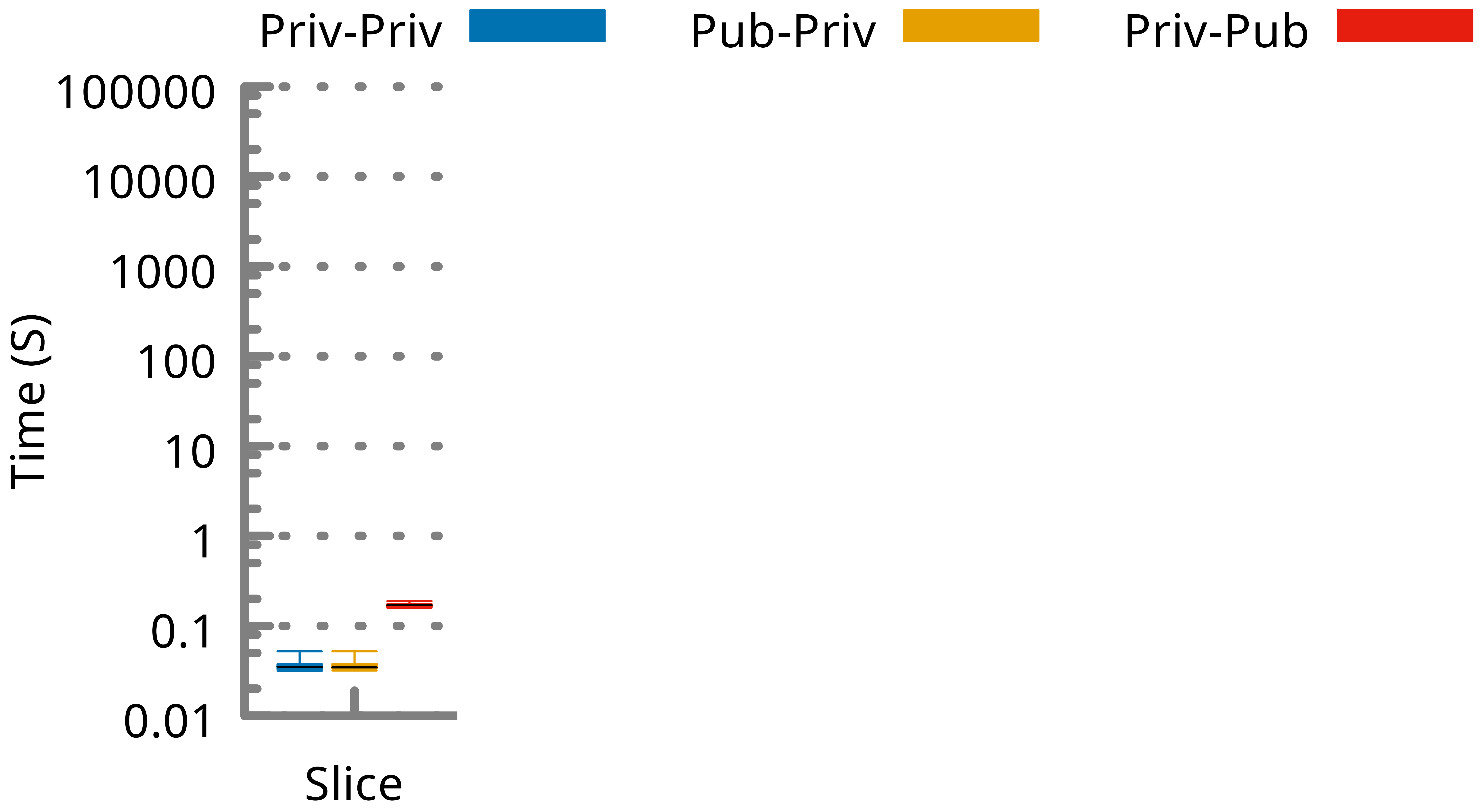
Evaluation Setup: Datacenter

- Consider AWS like multi-tenant datacenter.
- Each tenant has policies for private and public hosts.
- Three verification tasks
    - Private hosts for one tenant cannot reach another
    - Public host for one tenant cannot reach private hosts for another

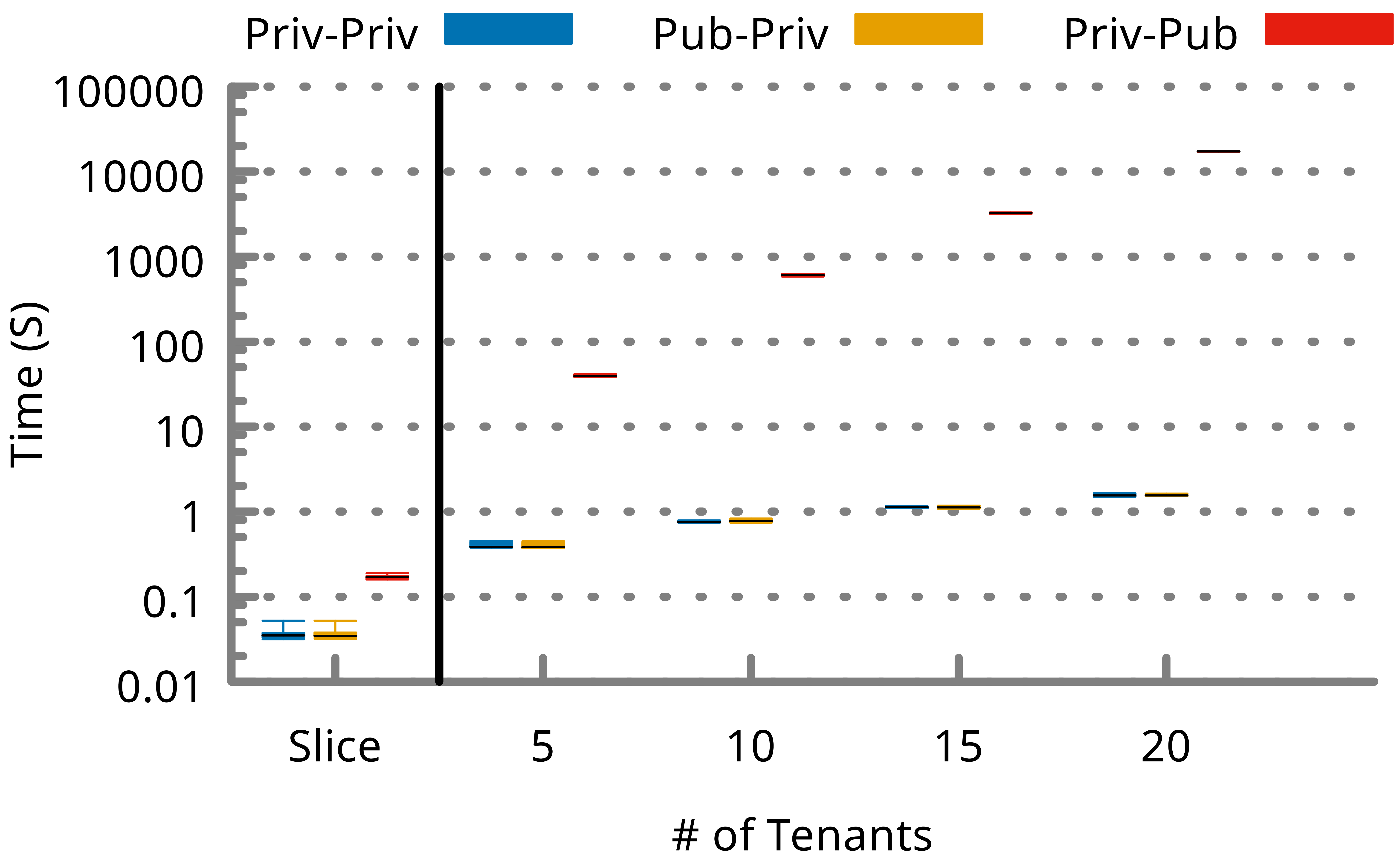
Evaluation Setup: Datacenter

- Consider AWS like multi-tenant datacenter.
- Each tenant has policies for private and public hosts.
- Three verification tasks
  - Private hosts for one tenant cannot reach another
  - Public host for one tenant cannot reach private hosts for another
  - Public hosts are universally reachable.

Verification Time (Datacenter)



Verification Time (Datacenter)



Role of Symmetry

- Consider a private datacenter

Role of Symmetry

- Consider a private datacenter
  - Use verification to prevent some bugs from a Microsoft DC (IMC 2013)

Role of Symmetry

- Consider a private datacenter
  - Use verification to prevent some bugs from a Microsoft DC (IMC 2013)
- Bugs include

Role of Symmetry

- Consider a private datacenter
  - Use verification to prevent some bugs from a Microsoft DC (IMC 2013)
- Bugs include
  - Misconfigured firewalls

Role of Symmetry

- Consider a private datacenter
  - Use verification to prevent some bugs from a Microsoft DC (IMC 2013)
- Bugs include
  - Misconfigured firewalls
  - Misconfigured redundant firewalls

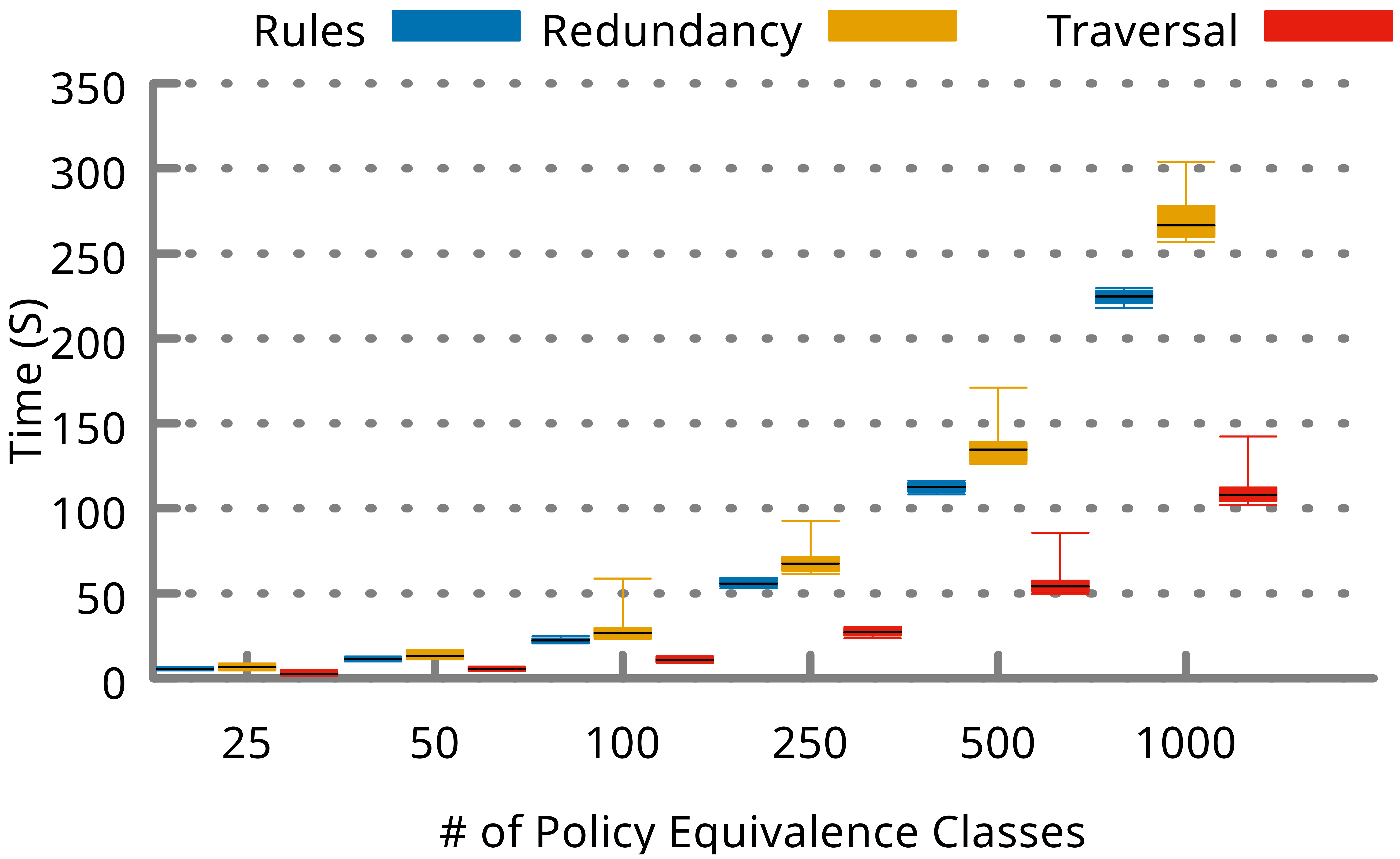
Role of Symmetry

- Consider a private datacenter
  - Use verification to prevent some bugs from a Microsoft DC (IMC 2013)
- Bugs include
  - Misconfigured firewalls
  - Misconfigured redundant firewalls
  - Misconfigured redundant routing

Role of Symmetry

- Consider a private datacenter
  - Use verification to prevent some bugs from a Microsoft DC (IMC 2013)
- Bugs include
  - Misconfigured firewalls
  - Misconfigured redundant firewalls
  - Misconfigured redundant routing
- Measure time to verify as a function of number of symmetric policy groups

Verification Time (With Symmetry)



Conclusion

- Verifying stateful networks is increasingly important.
- The primary challenge is scaling to realistic network.
- Two methods to scale
  - Models where oracles are separated from forwarding behavior.
  - Split the network into smaller verifiable portions is necessary.