New Directions for Network Verification

Aurojit Panda, Katerina Argyraki, Mooly Sagiv, Michael Schapira, Scott Shenker

Brief Summary of This Talk

- *Context*:
  - Proliferation of network verification tools.
  - Build on assumption that the network state is **immutable**.
    - Immutable = Data packets do not change behavior of network

Brief Summary of This Talk

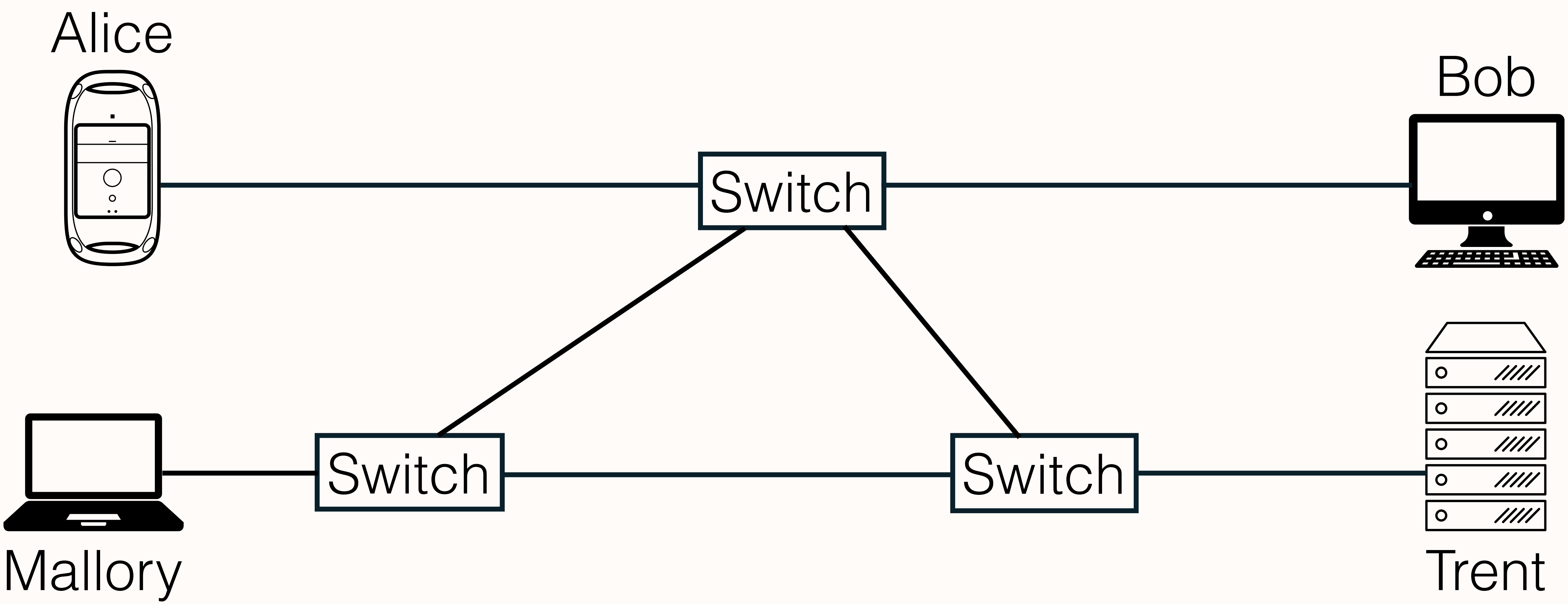
- *Context*:
  - Proliferation of network verification tools.
  - Build on assumption that the network state is **immutable**.
    - Immutable = Data packets do not change behavior of network
- *My point*:
  - Many network elements have **mutable state**
  - **Verifying mutable networks requires new techniques**
  - Two technical challenges: Modeling and Scaling

Outline

- **Background on networks.**
- Background on network verification.
- Verifying mutable networks.

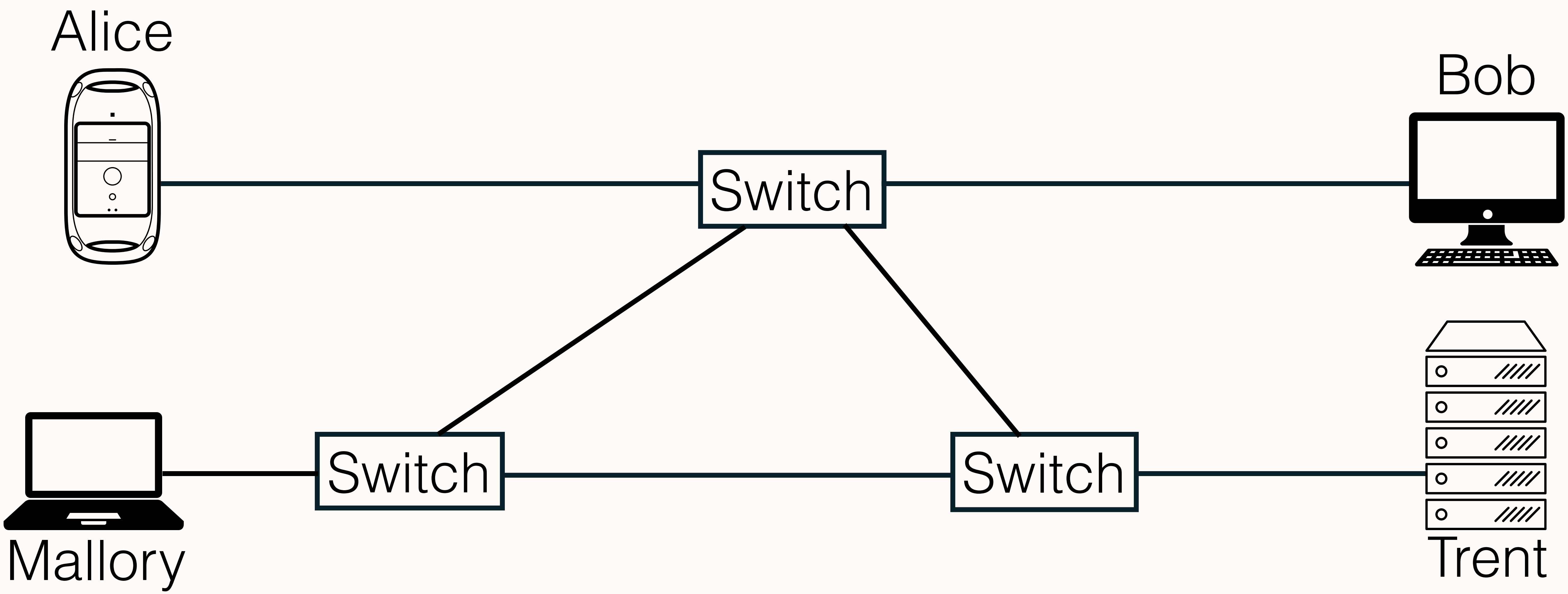
Classical Networking

Ted Stevens was right

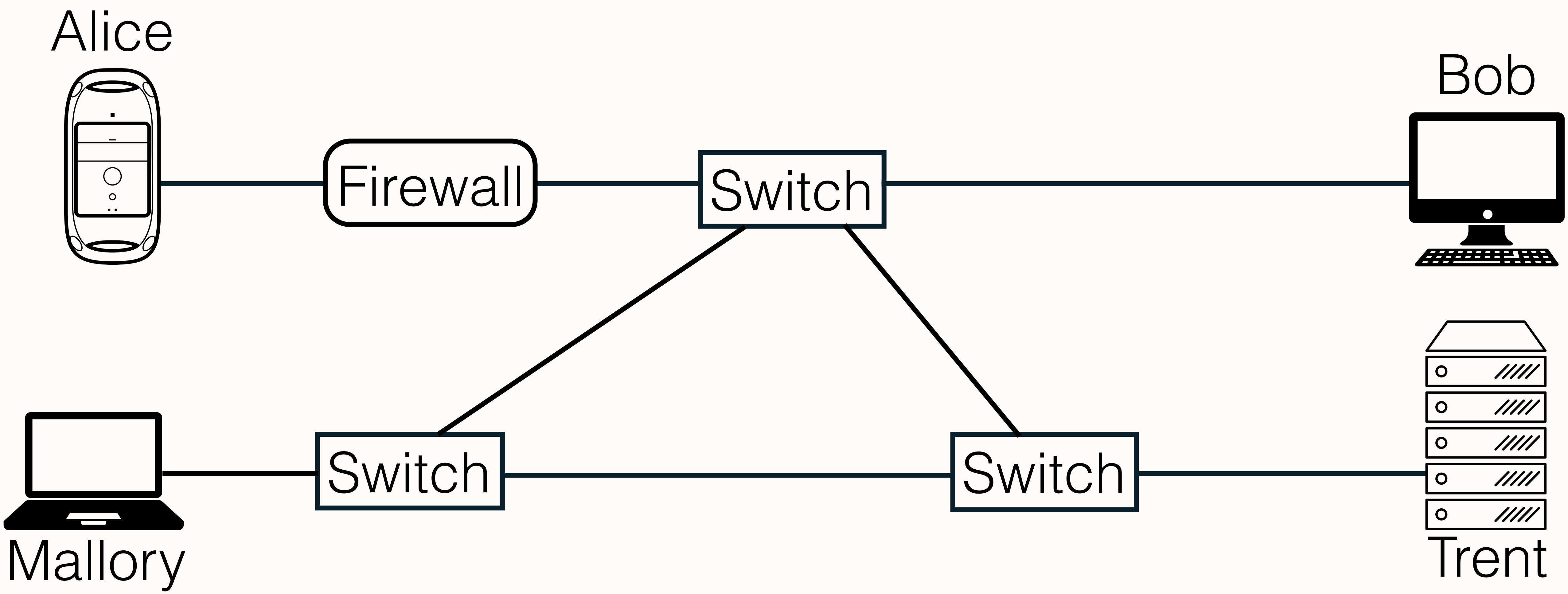


- Networks provide end-to-end connectivity.
- Just contain host and switches.
- All interesting processing at the hosts.

Real Networks have Middleboxes!

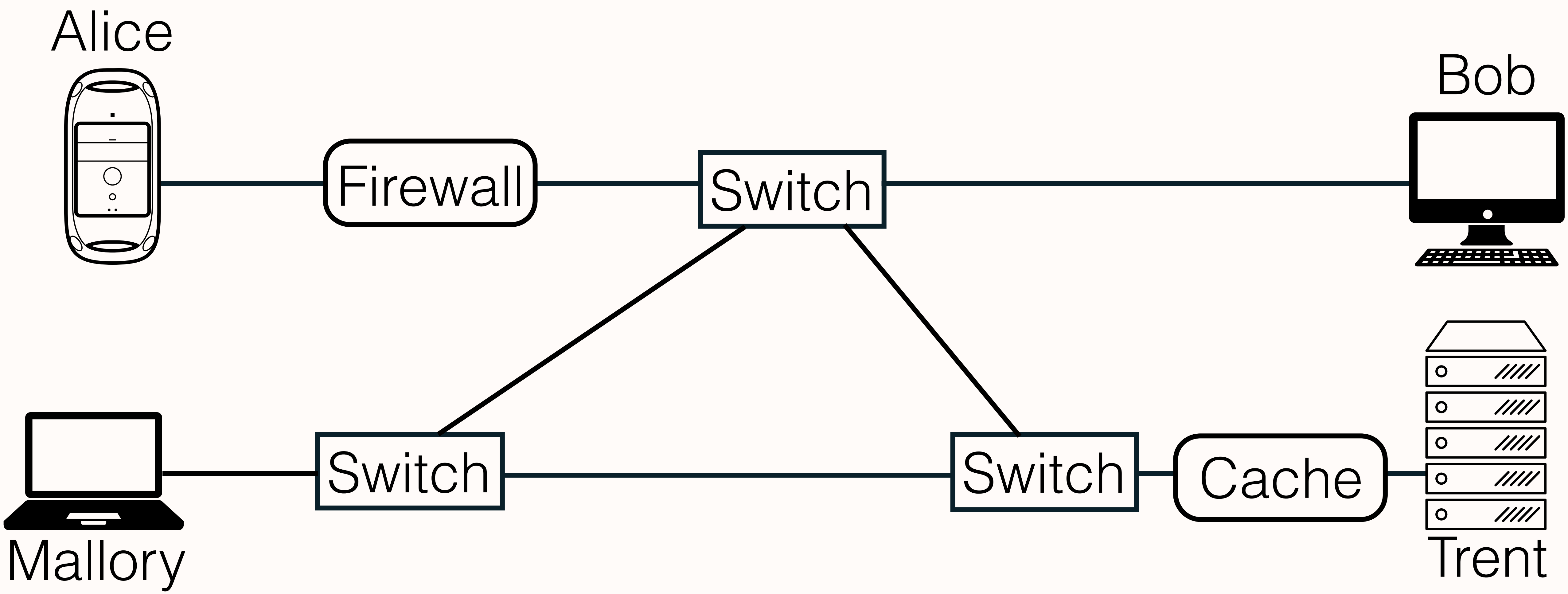


Real Networks have Middleboxes!



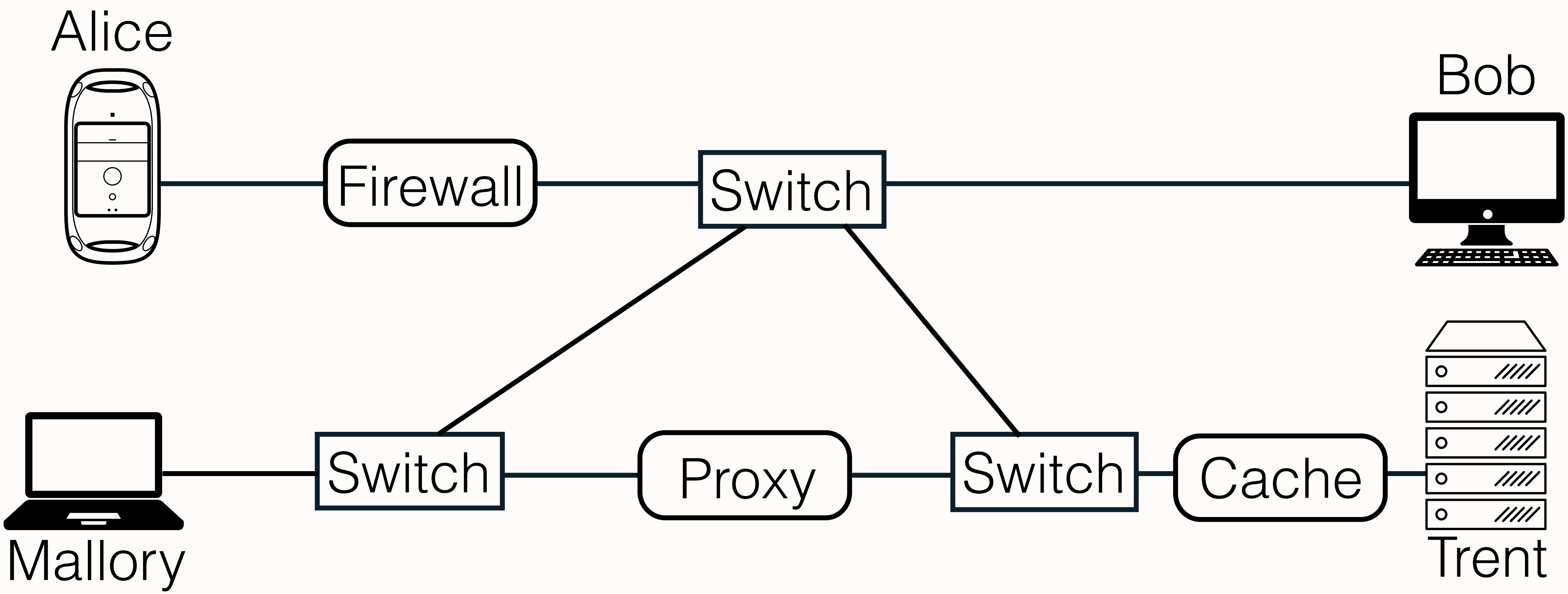
- Security (firewalls, IDSs,…).

Real Networks have Middleboxes!



- Security (firewalls, IDSs,…).
- Performance (caches, load balancers,…).

Real Networks have Middleboxes!



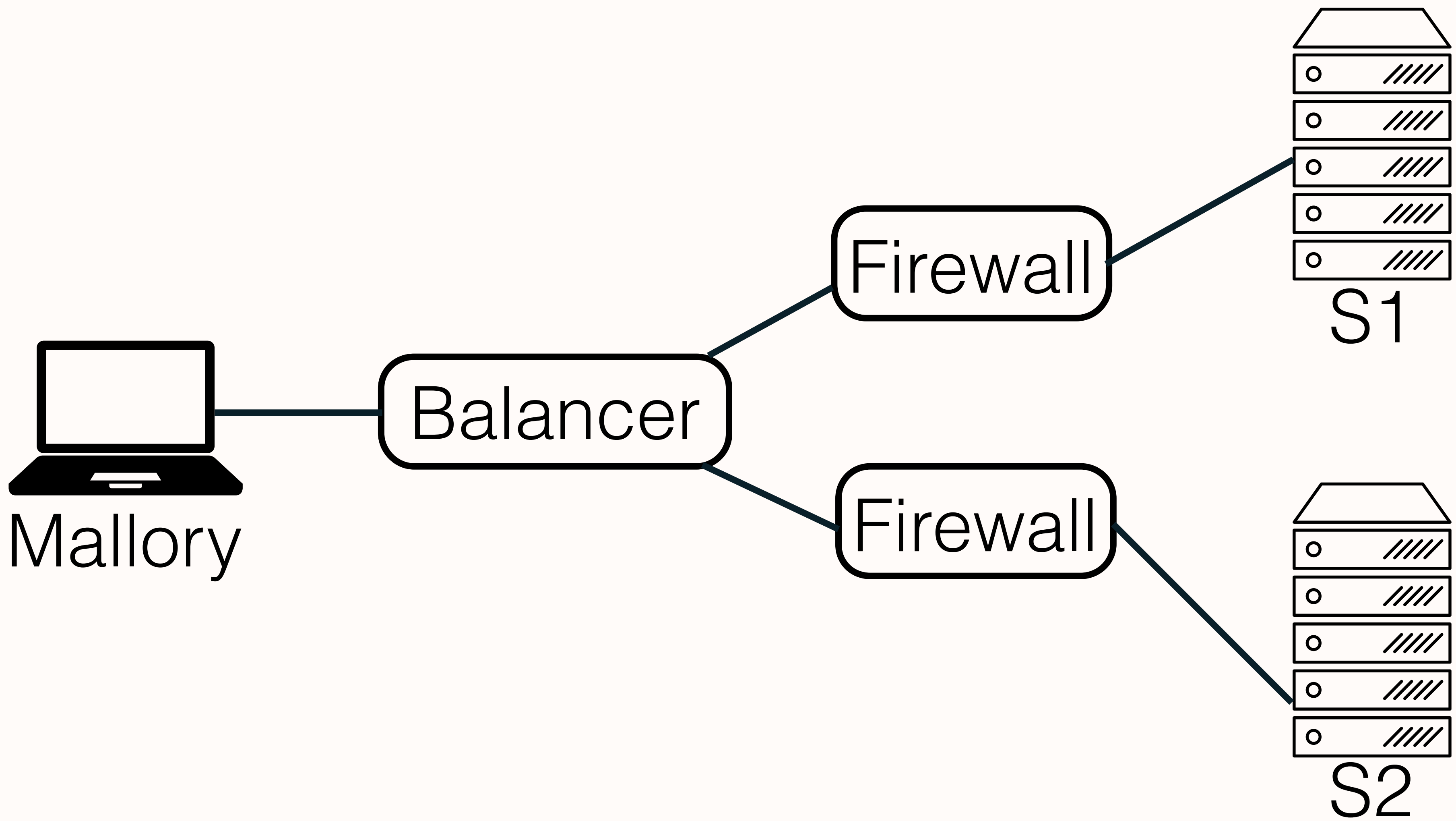
- Security (firewalls, IDSs,…).
- Performance (caches, load balancers,…).
- New functionality (proxies,…).

Outline

- Background on networks.
- **Background on network verification.**
- Verifying mutable networks.

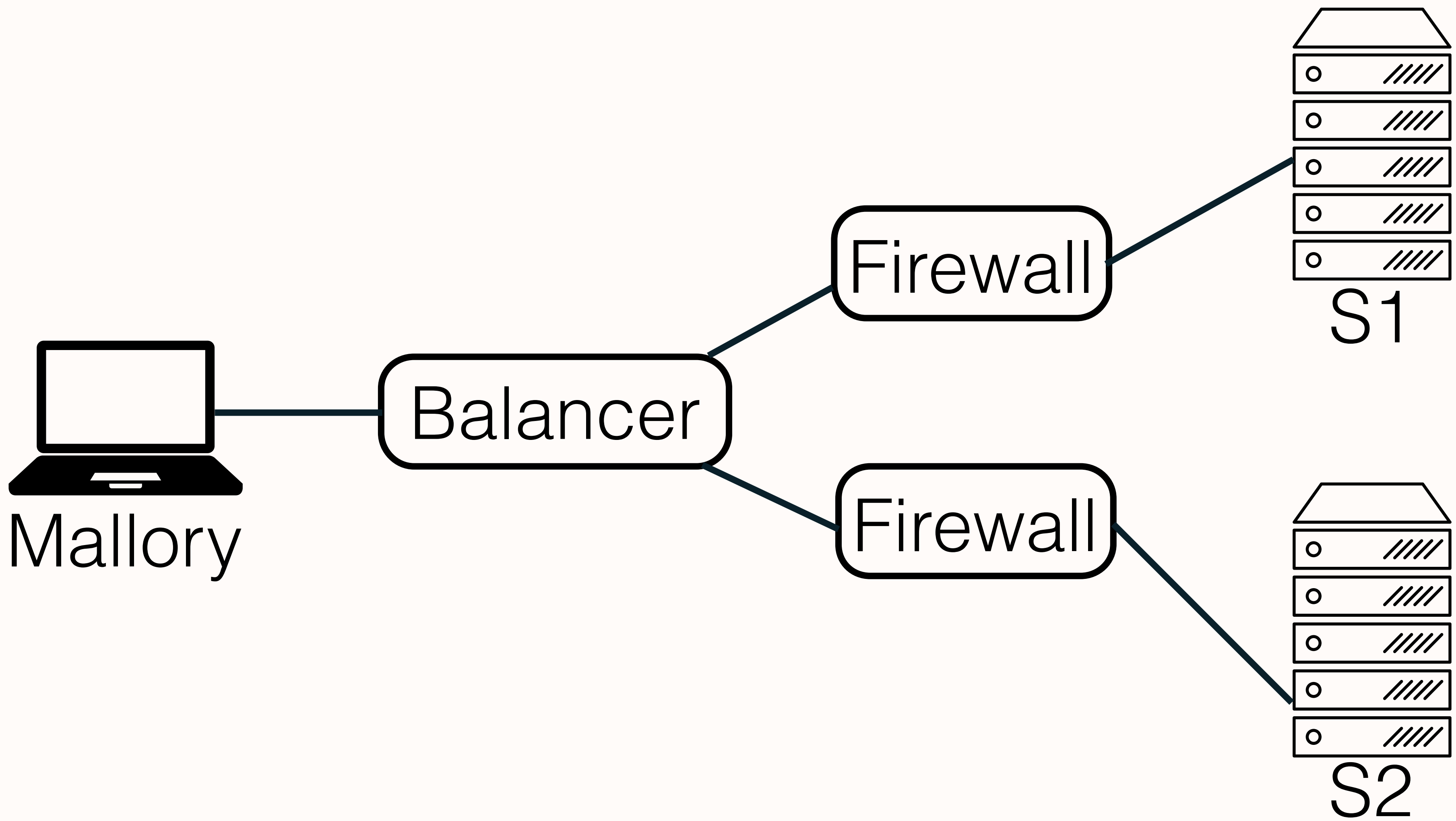
Reachability Invariants

- Focus on reachability invariants
  - Most important in practice, simple to state but already hard



Reachability Invariants

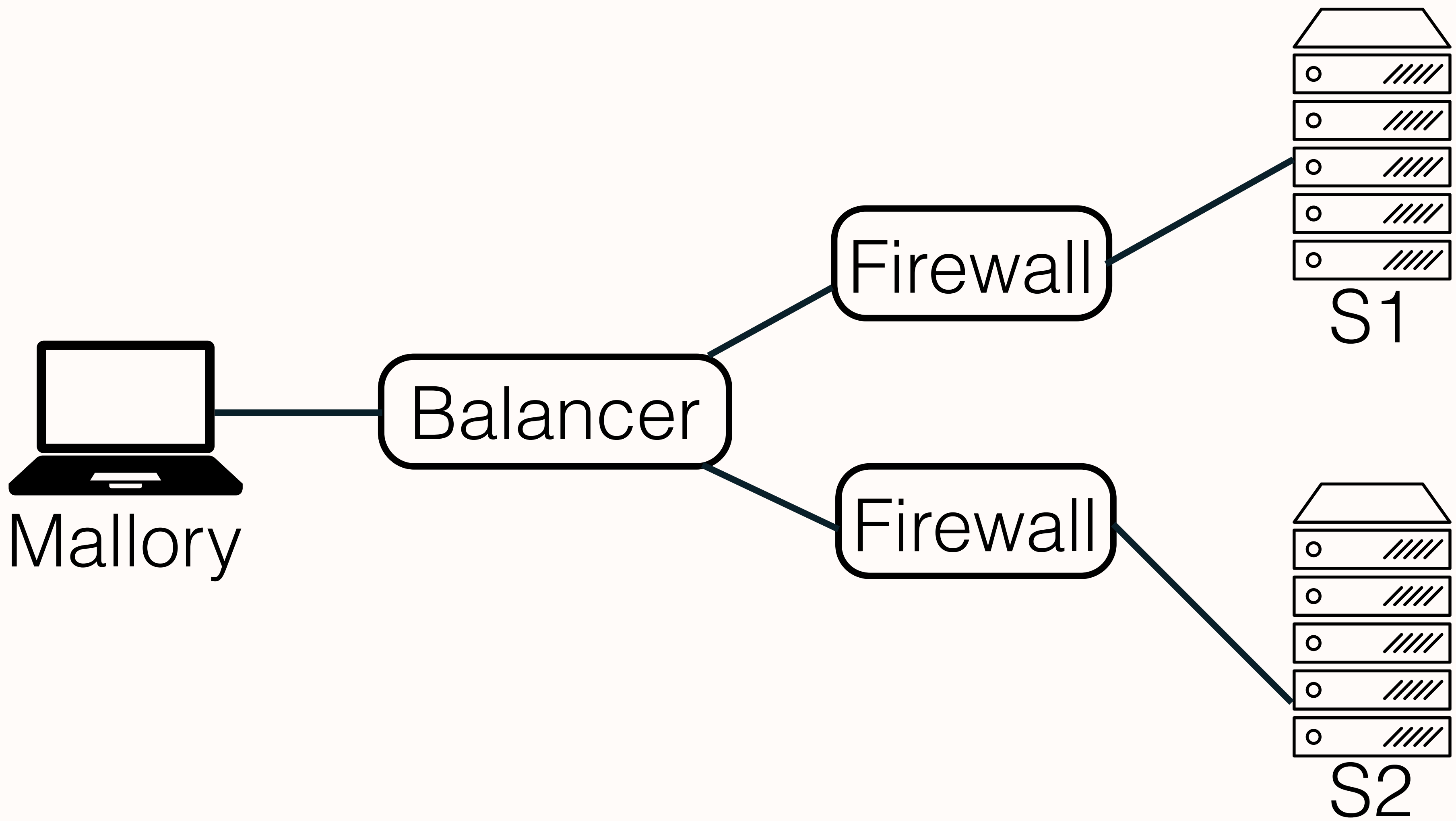
- Focus on reachability invariants
  - Most important in practice, simple to state but already hard



Can S2 receive packets of type T from Mallory?

Reachability Invariants

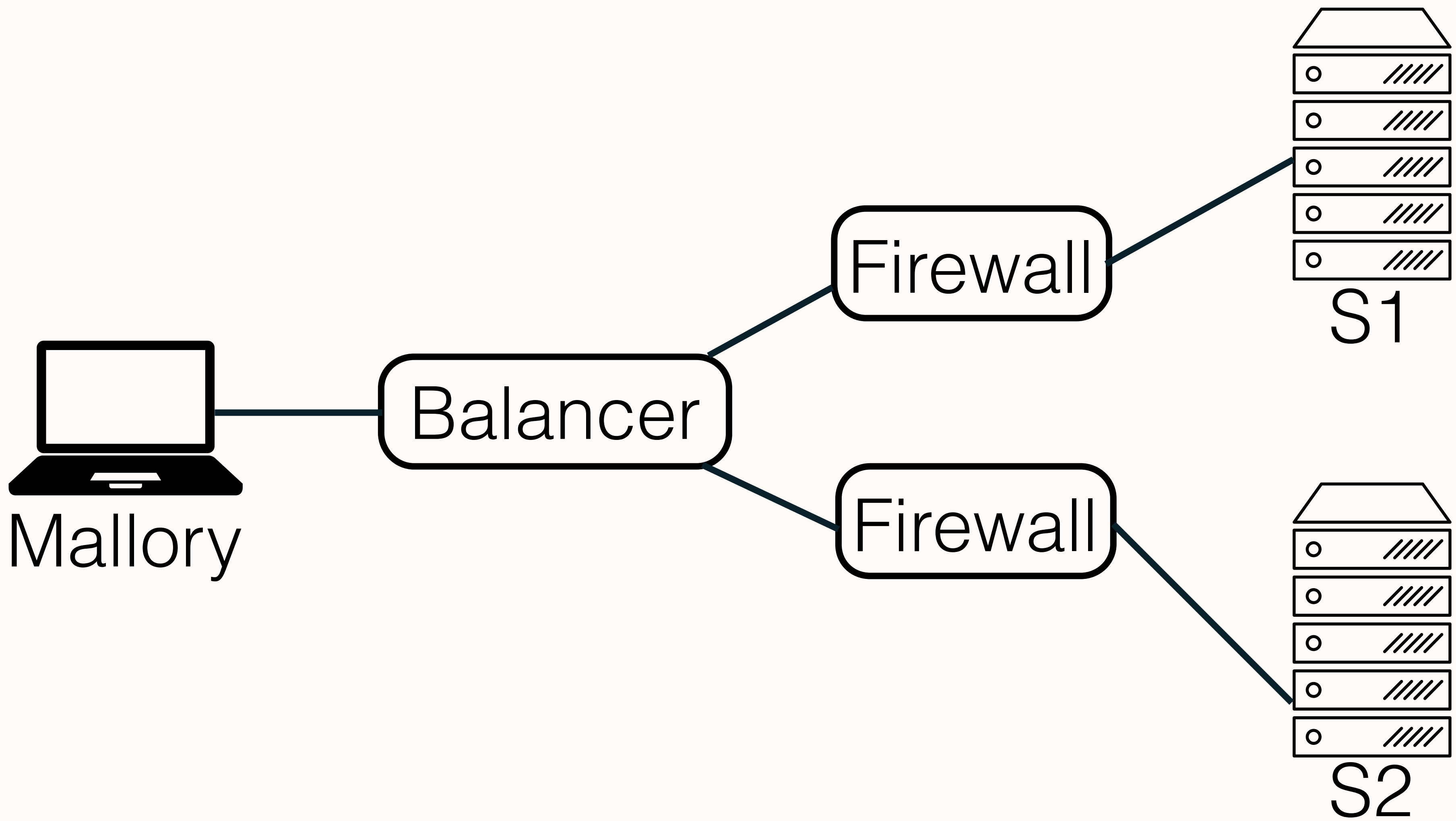
- Focus on reachability invariants
  - Most important in practice, simple to state but already hard



Can S2 receive “infected” packets from Mallory?

Reachability Invariants

- Focus on reachability invariants
  - Most important in practice, simple to state but already hard



Can S2 receive packets from Mallory without a connection?

Abstractions for Invariants

- Operators want to specify packet types using abstractions:

Abstractions for Invariants

- Operators want to specify packet types using abstractions:
  - “infected”

Abstractions for Invariants

- Operators want to specify packet types using abstractions:
  - “infected”
  - from “authenticated user”

Abstractions for Invariants

- Operators want to specify packet types using abstractions:
  - “infected”
  - from “authenticated user”
  - from a given application

Abstractions for Invariants

- Operators want to specify packet types using abstractions:
  - “infected”
  - from “authenticated user”
  - from a given application
- How these types are determined in a network varies

Abstractions for Invariants

- Operators want to specify packet types using abstractions:
  - “infected”
  - from “authenticated user”
  - from a given application
- How these types are determined in a network varies
  - Invariants should not depend on these details

Network Verification Today

- Switches: Forwarding rules in switches.

  HSA, Veriflow, NetKAT, etc.

Network Verification Today

- Switches: Forwarding rules in switches.

    HSA, Veriflow, NetKAT, etc.

- SDN Controller: Code generating these rules.

    Vericon, FlowLog, etc.

Network Verification Today

- Switches: Forwarding rules in switches.

    HSA, Veriflow, NetKAT, etc.

- SDN Controller: Code generating these rules.

    Vericon, FlowLog, etc.

- Firewalls: Verify firewall configuration.

    Fang, Margrave, etc.

Existing Assumptions/Limitations

Switches

- Limited computational model (rule-based forwarding).
- **Immutable**, functionality only changes with new rules.
- Limited set of invariants enforced by networks.

Existing Assumptions/Limitations

Switches

- Limited computational model (rule-based forwarding).
- **Immutable**, functionality only changes with new rules.
- Limited set of invariants enforced by networks.

Controllers

- All state and actions are **centralized**. (Globally ordered)
- Data plane itself is **immutable**.

Existing Assumptions/Limitations

Switches

- Limited computational model (rule-based forwarding).
- **Immutable**, functionality only changes with new rules.
- Limited set of invariants enforced by networks.

Controllers

- All state and actions are **centralized**. (Globally ordered)
- Data plane itself is **immutable**.

Firewalls

- Treated as if they contain **Immutable** state.
- Assume a particular (simple) computational model.

Existing Assumptions/Limitations

Switches

- Limited computational model (rule-based forwarding).
- **Immutable**, functionality only changes with new rules.
- Limited set of invariants enforced by networks.

Controllers

- All state and actions are **centralized**. (Globally ordered)
- Data plane itself is **immutable**.

Firewalls

- Treated as if they contain **Immutable** state.
- Assume a particular (simple) computational model.

Violated by many middleboxes

Outline

- Background on networks.
- Background on network verification.
- **Verifying mutable networks.**

Verification of Mutable Networks

- Naive approach
  - Verify a program equivalent to the entire network.

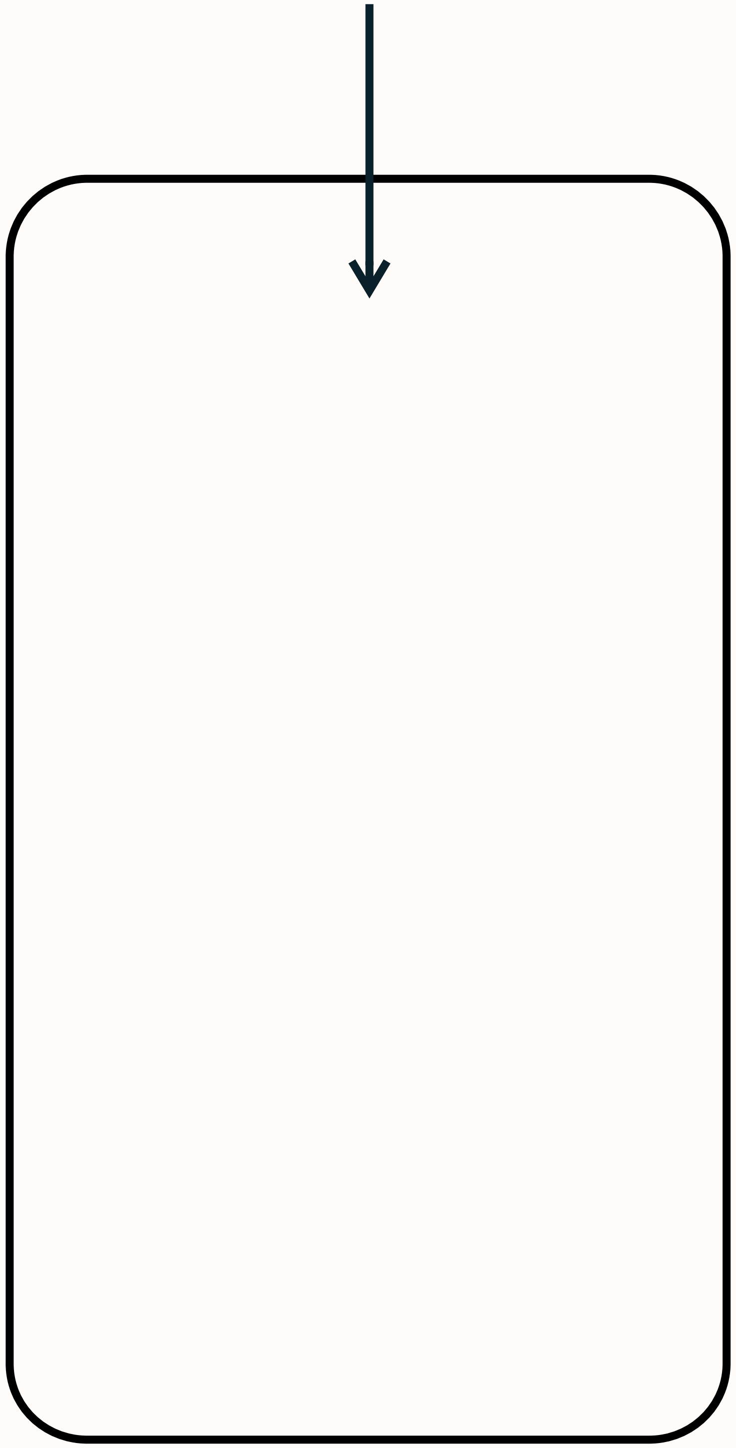
Verification of Mutable Networks

- Naive approach
  - Verify a program equivalent to the entire network.
- Feasibility is not clear
  - Large, proprietary code bases (Bro ~102K lines of code).

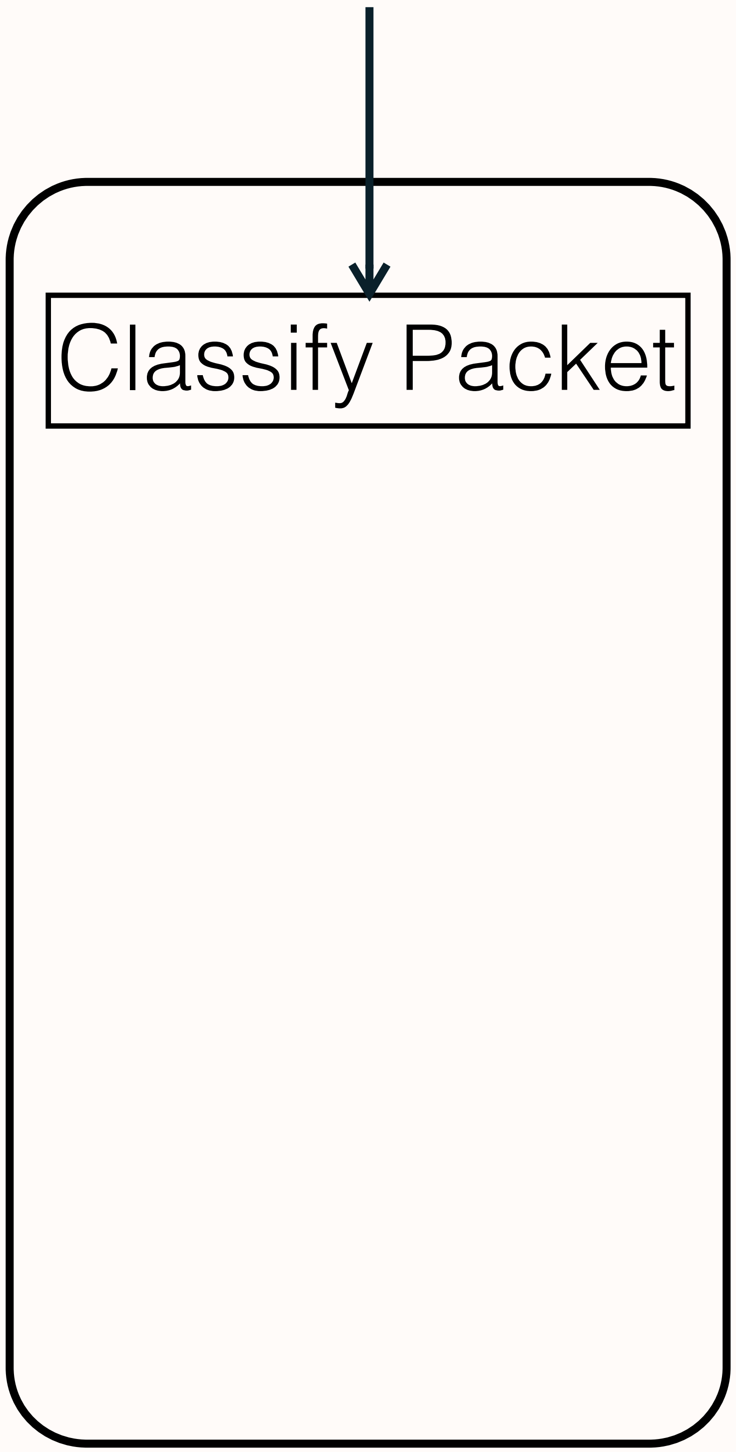
Verification of Mutable Networks

- Naive approach
  - Verify a program equivalent to the entire network.
- Feasibility is not clear
  - Large, proprietary code bases (Bro ~102K lines of code).
- Scalability is crucial
  - Networks contain several 1000 middleboxes or more.

Modeling Middleboxes

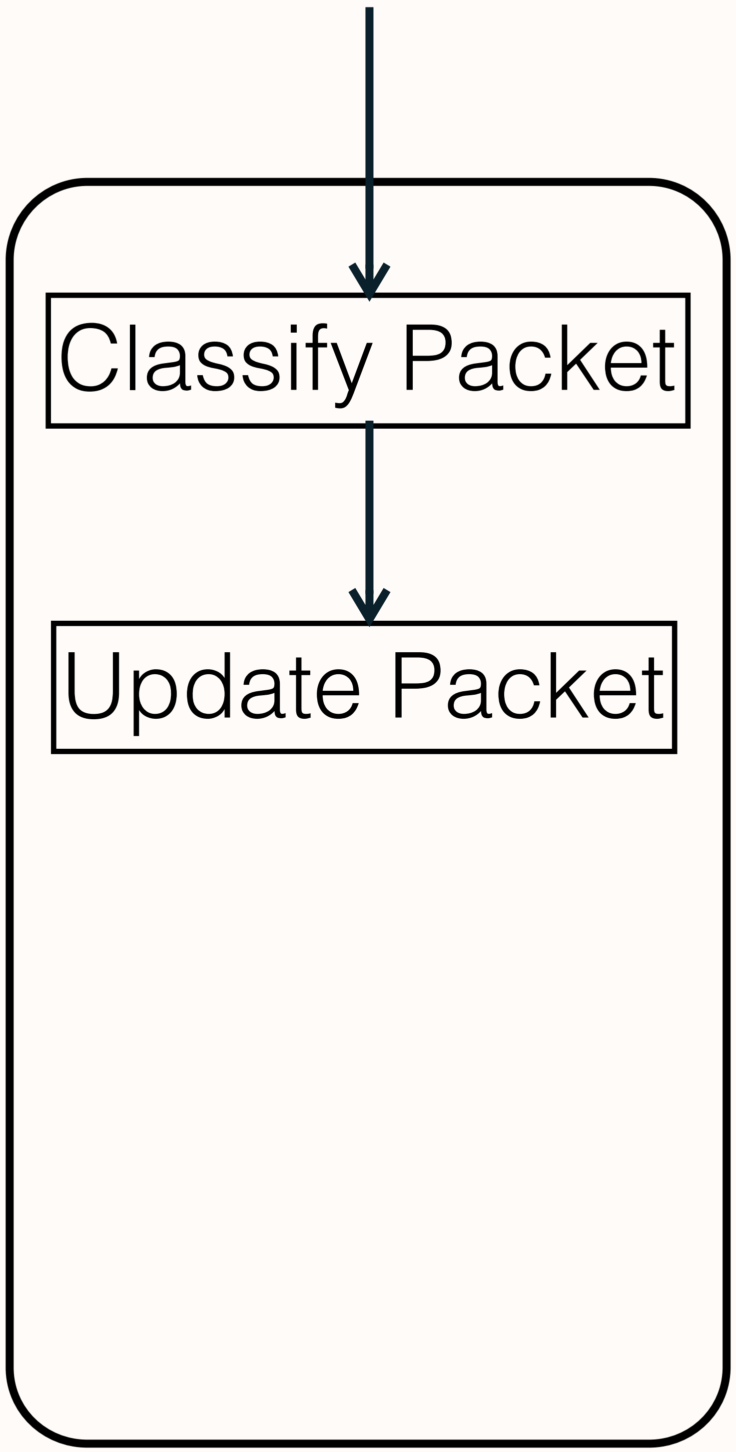


Modeling Middleboxes



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

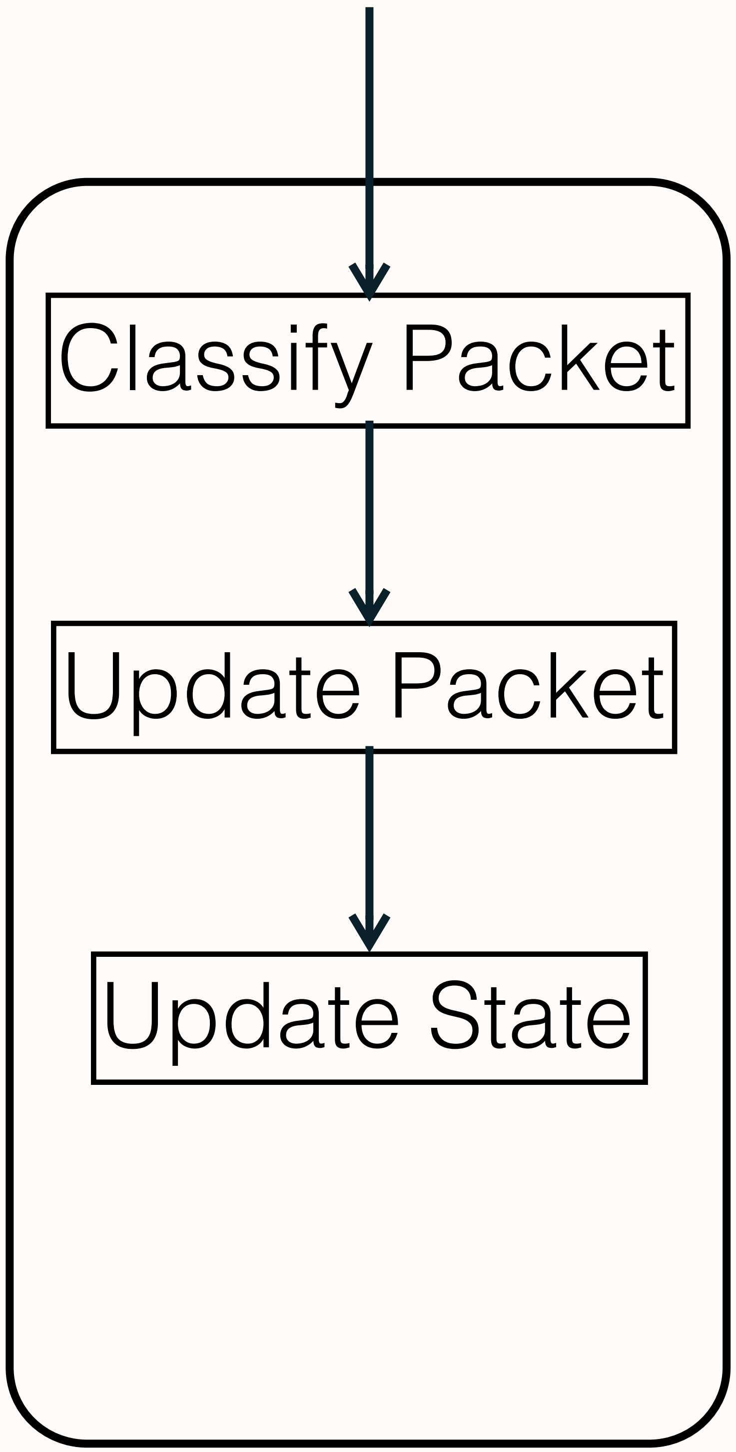
Modeling Middleboxes



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

Updating payload is complex (compression, etc.)
Updating header is simple (fixed format).

Modeling Middleboxes

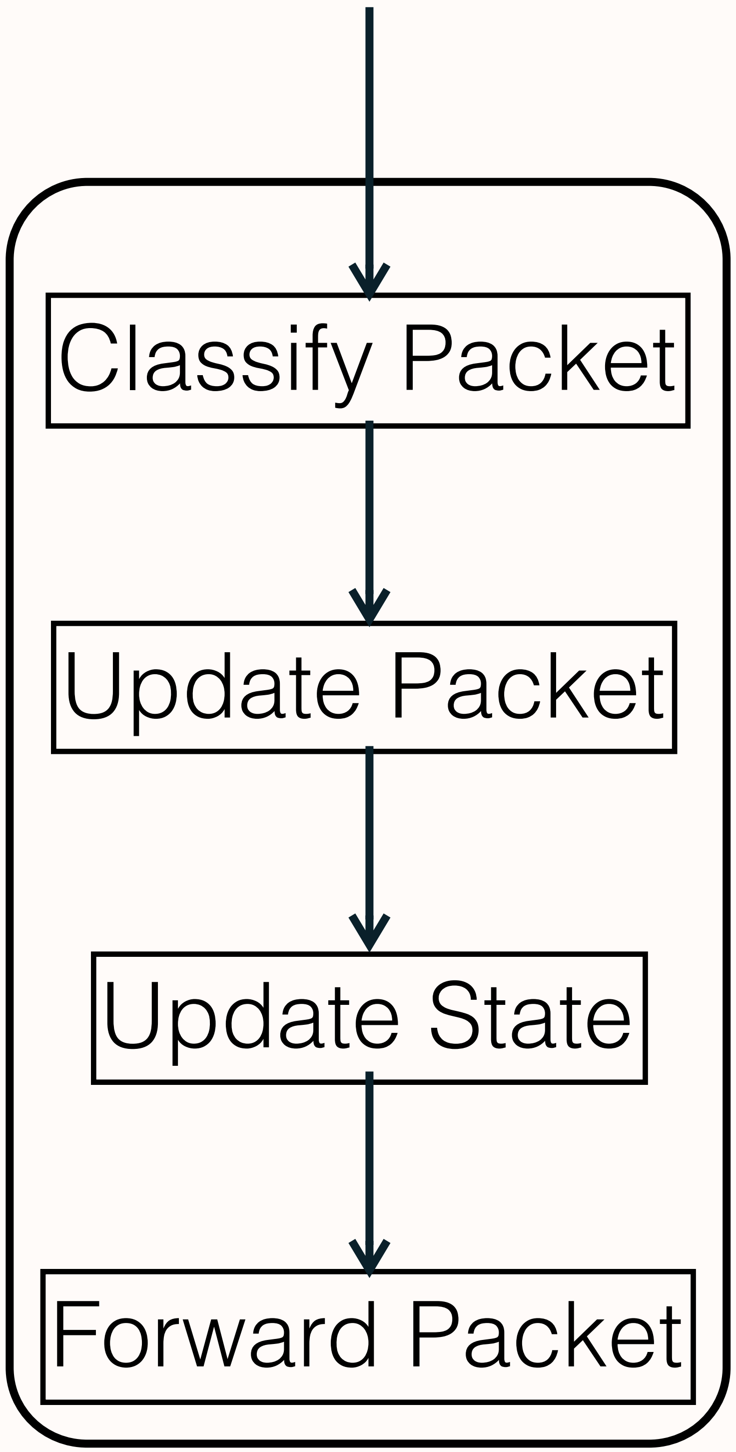


Determines what application sent a packet, etc.
Complex, proprietary processing.

Updating payload is complex (compression, etc.)
Updating header is simple (fixed format).

Could be simple (remember packets)
or complex (update many hash tables).

Modeling Middleboxes



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

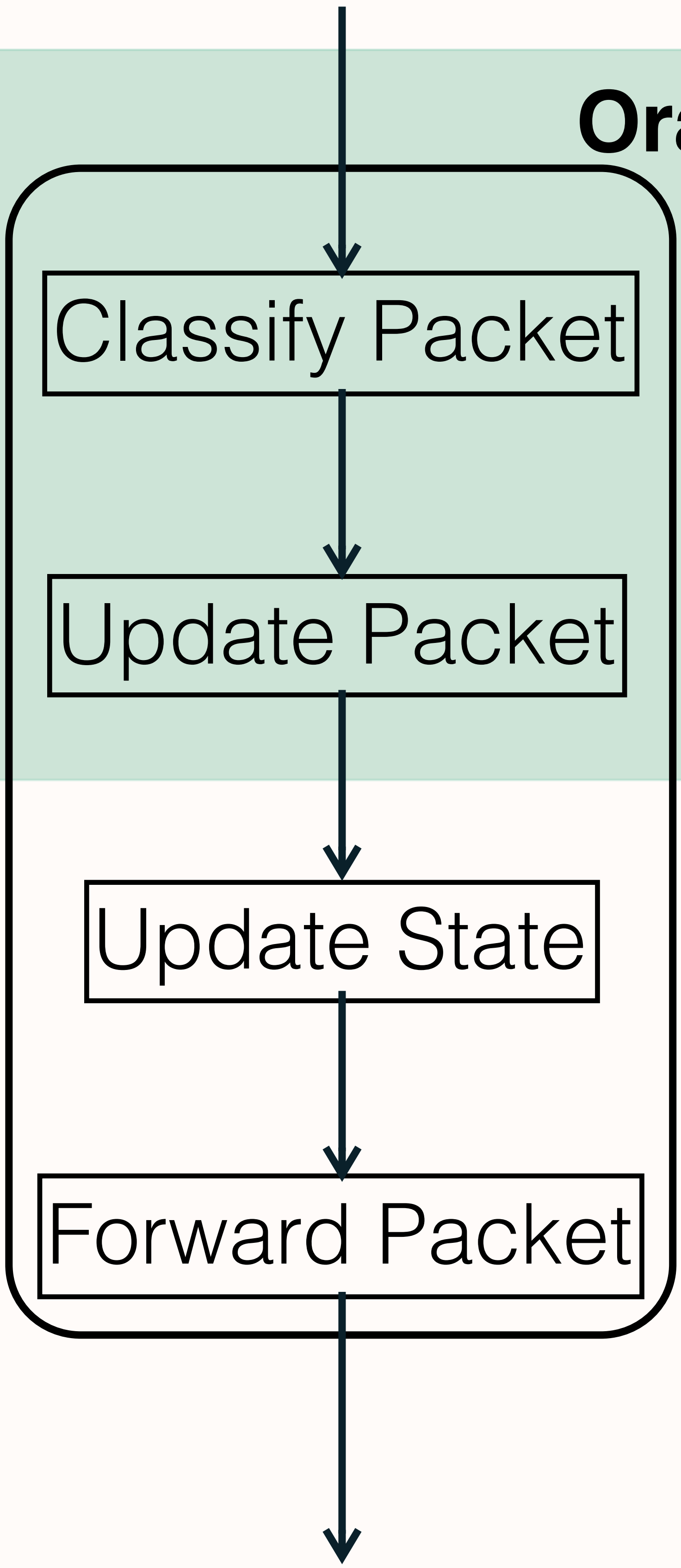
Updating payload is complex (compression, etc.)
Updating header is simple (fixed format).

Could be simple (remember packets)
or

Always simple: forward or drop packets.

Modeling Middleboxes

Oracle: Specify data dependencies and outputs



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

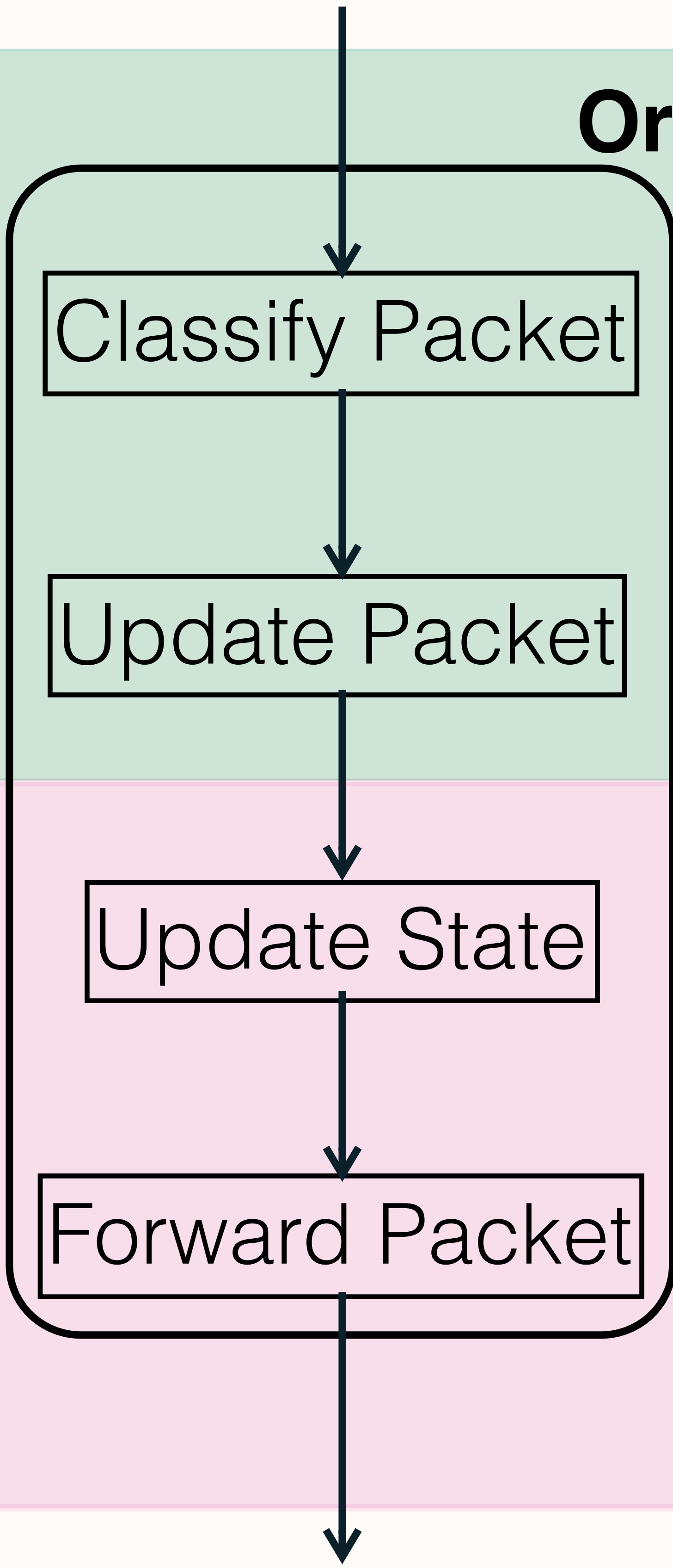
Updating payload is complex (compression, etc.)
Updating header is simple (fixed format).

Could be simple (remember packets)
or

Always simple: forward or drop packets.

Modeling Middleboxes

Oracle: Specify data dependencies and outputs



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

Updating payload is complex (compression, etc.)
Updating header is simple (fixed format).

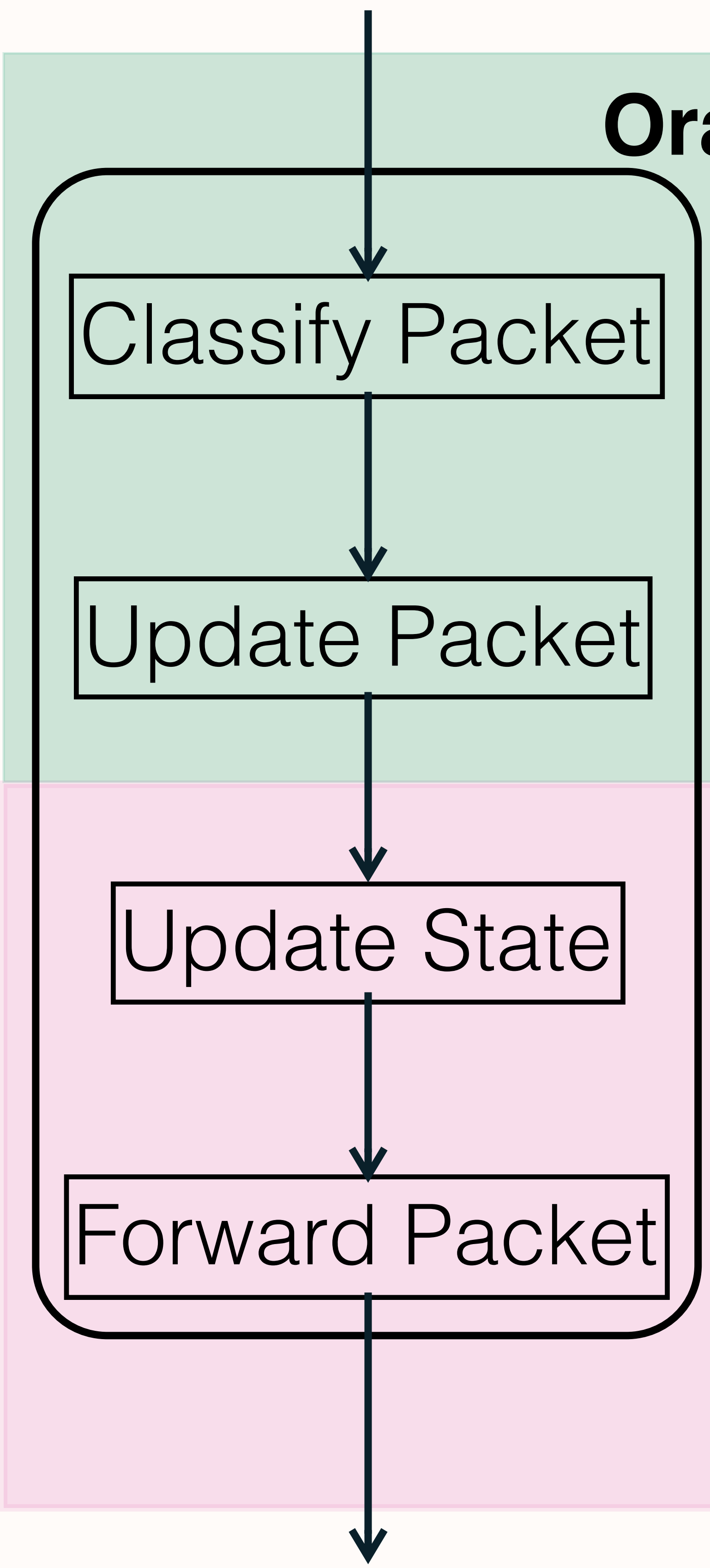
Could be simple (remember packets)
or

Always simple: forward or drop packets.

Forwarding Model: Specify Completely

Example

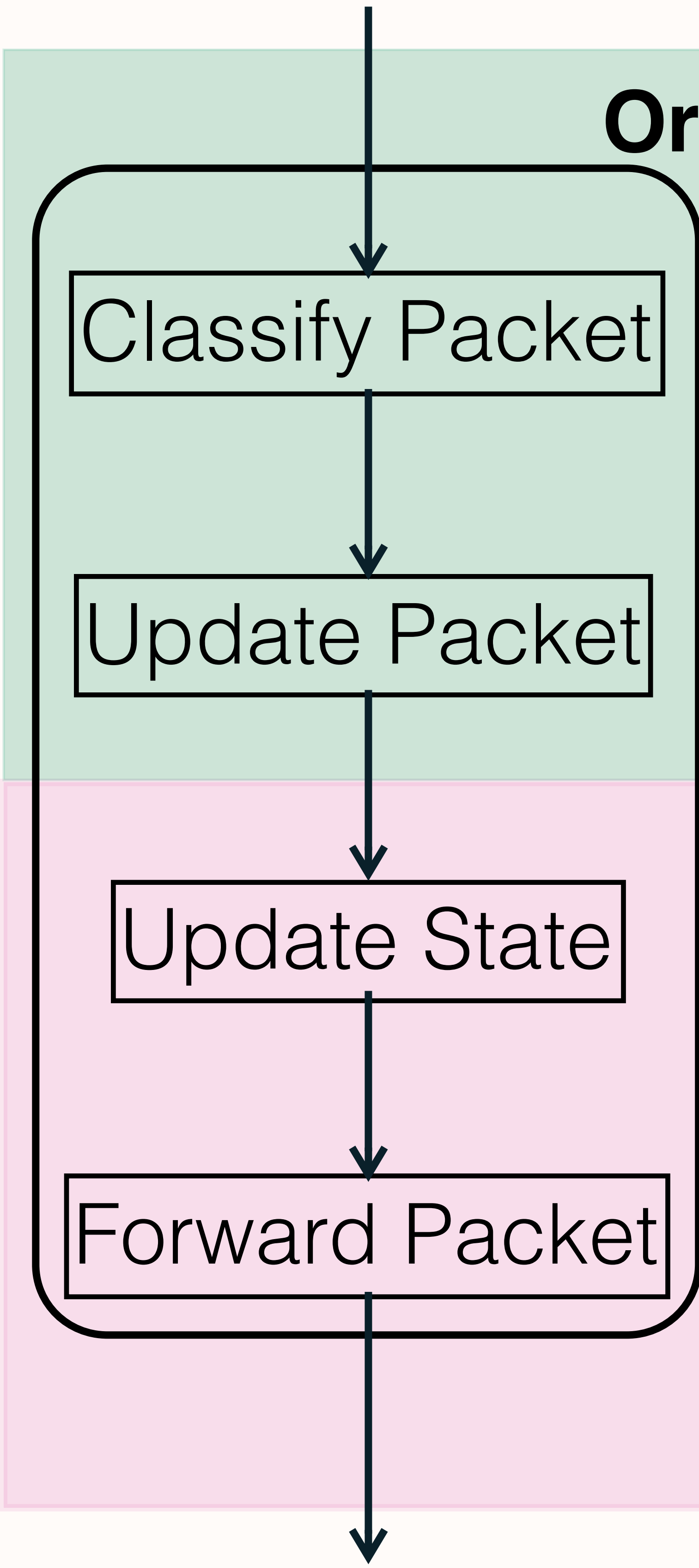
Oracle: Specify data dependencies and outputs



Forwarding Model: Specify Completely

Example

Oracle: Specify data dependencies and outputs



Dependencies

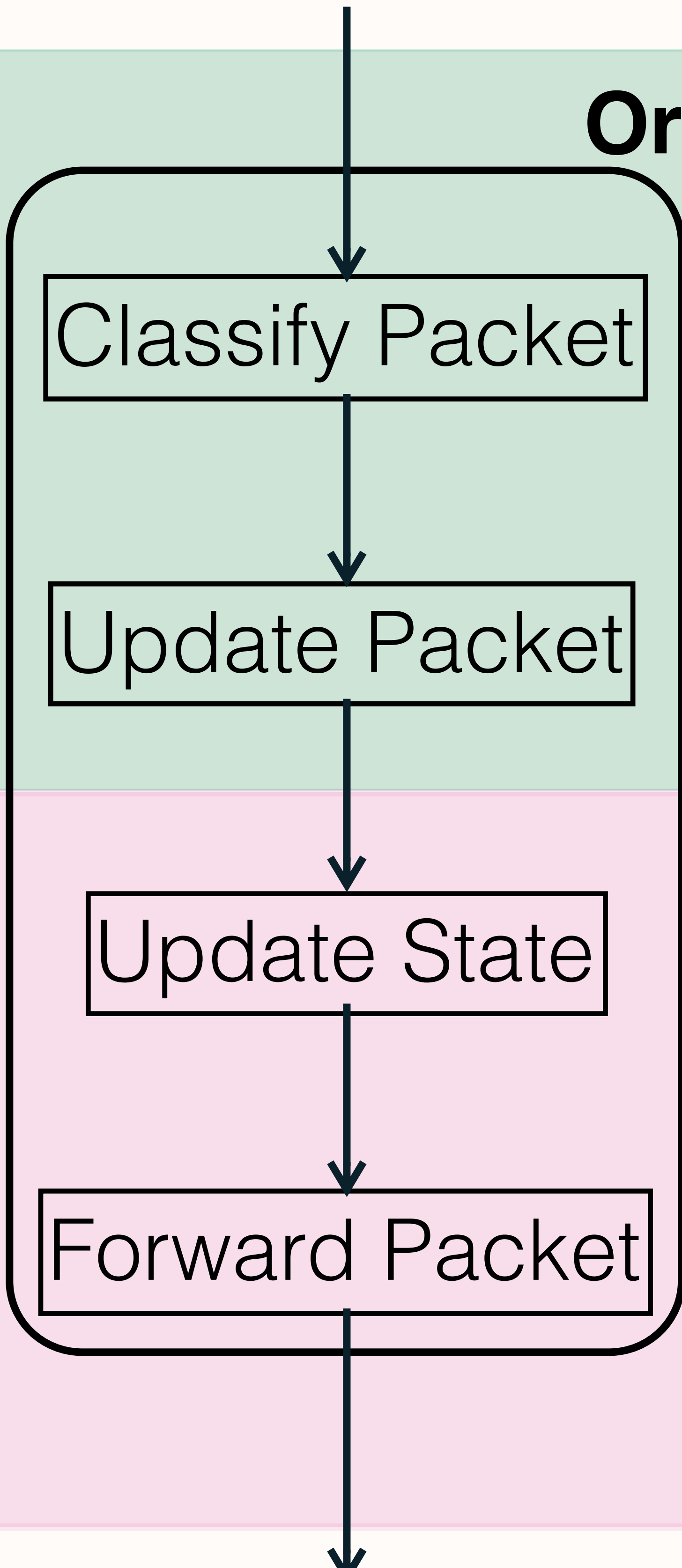
See all packets in connection (flow).

Outputs

Is packet **infected**.

Forwarding Model: Specify Completely

Example



Oracle: Specify data dependencies and outputs

Dependencies

See all packets in connection (flow).

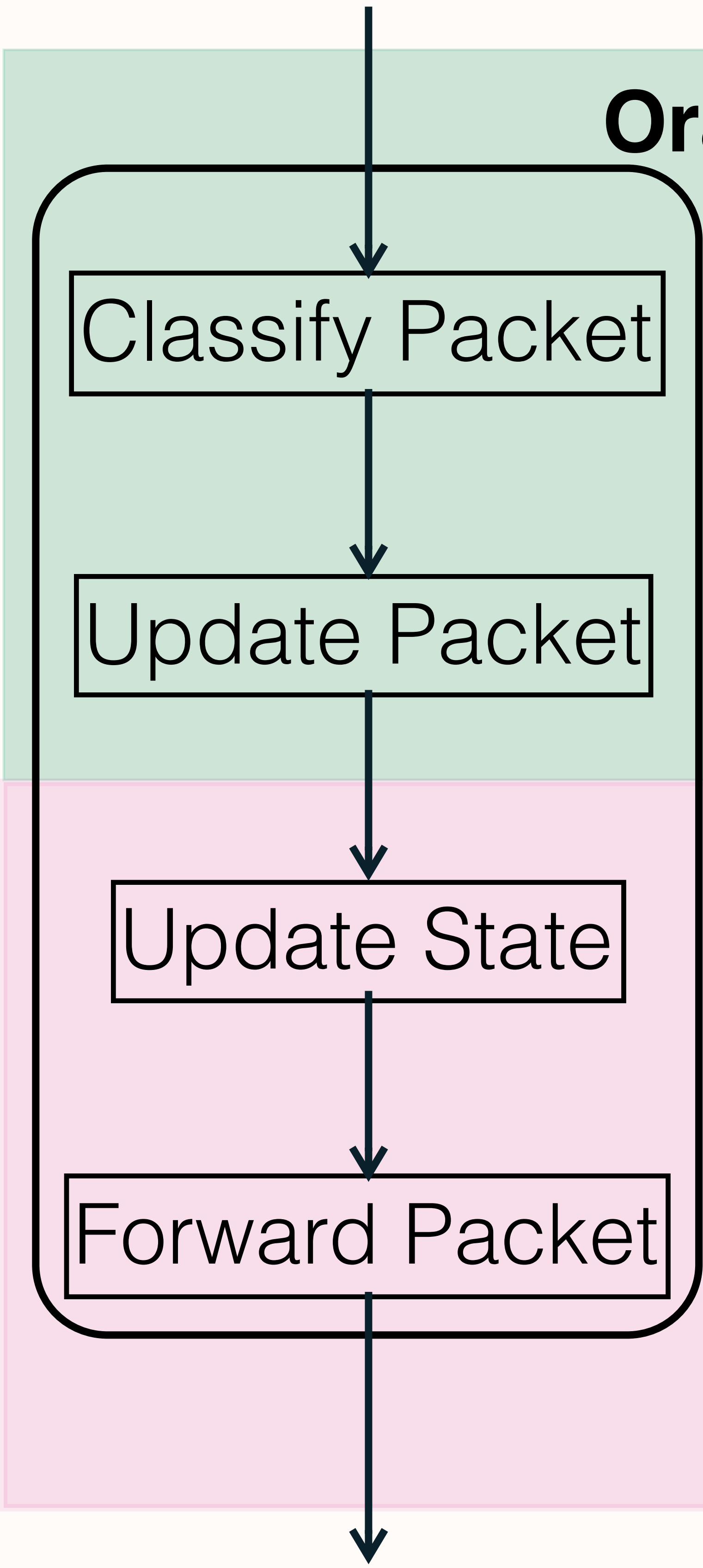
Outputs

Is packet **infected**.

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

Forwarding Model: Specify Completely

Example



Oracle: Specify data dependencies and outputs

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);
}

Forwarding Model: Specify Completely

Scaling Verification

Scaling Verification

- Middleboxes are "flow-parallel"

Scaling Verification

- Middleboxes are “flow-parallel”
  - State is partitioned between “flows.”

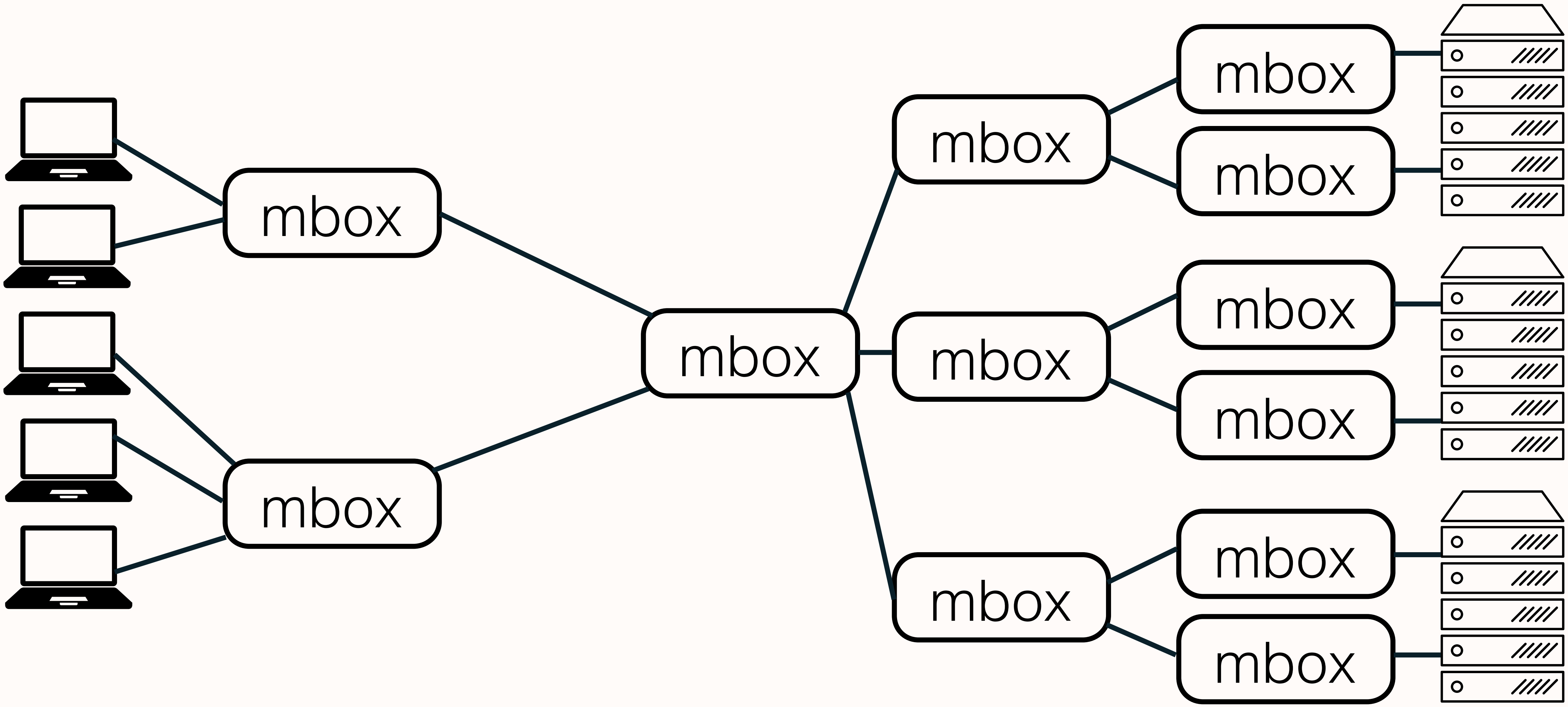
Scaling Verification

- Middleboxes are “flow-parallel”
  - State is partitioned between “flows.”
- This enables “compositional verification”

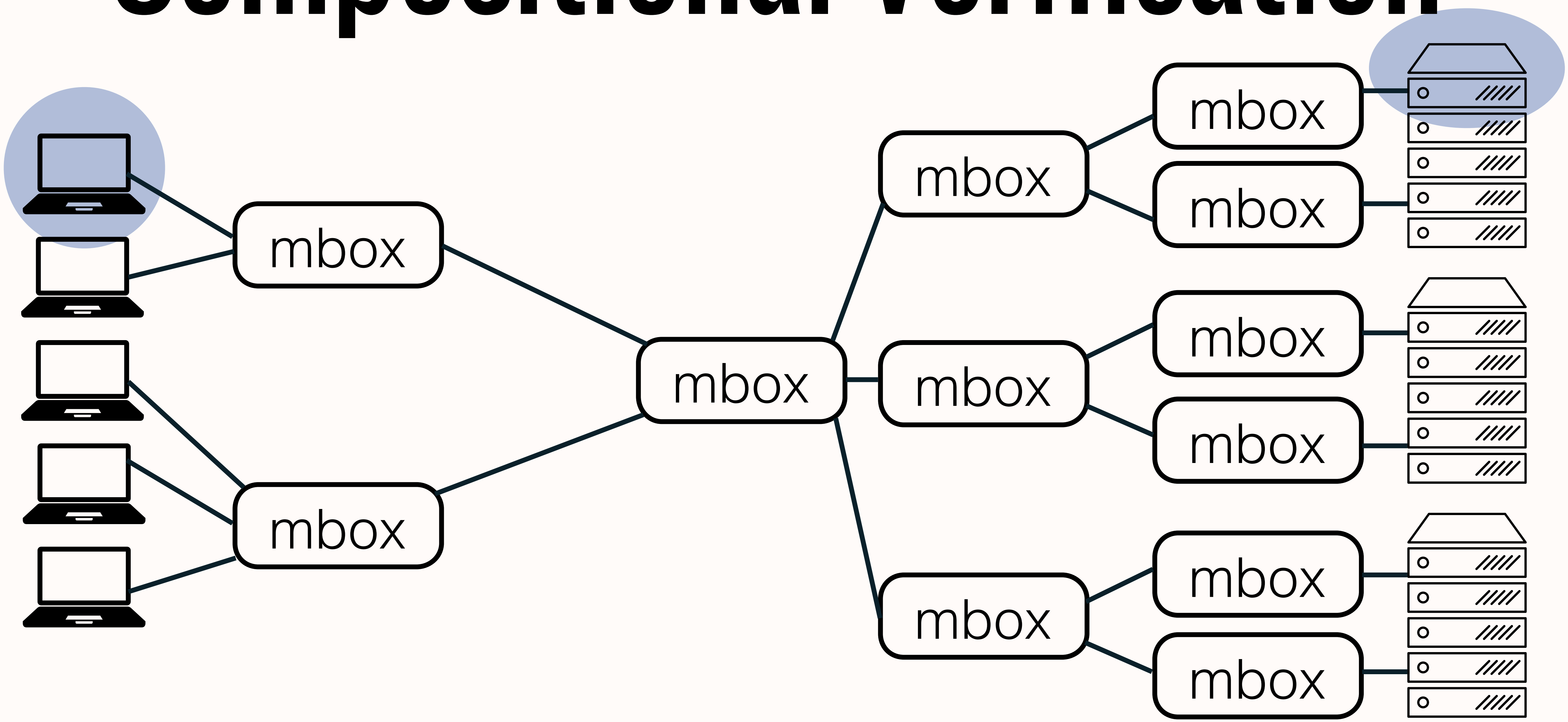
Scaling Verification

- Middleboxes are “flow-parallel”
  - State is partitioned between “flows.”
- This enables “compositional verification”
  - 30,000 middlebox networks verified in 5 minutes

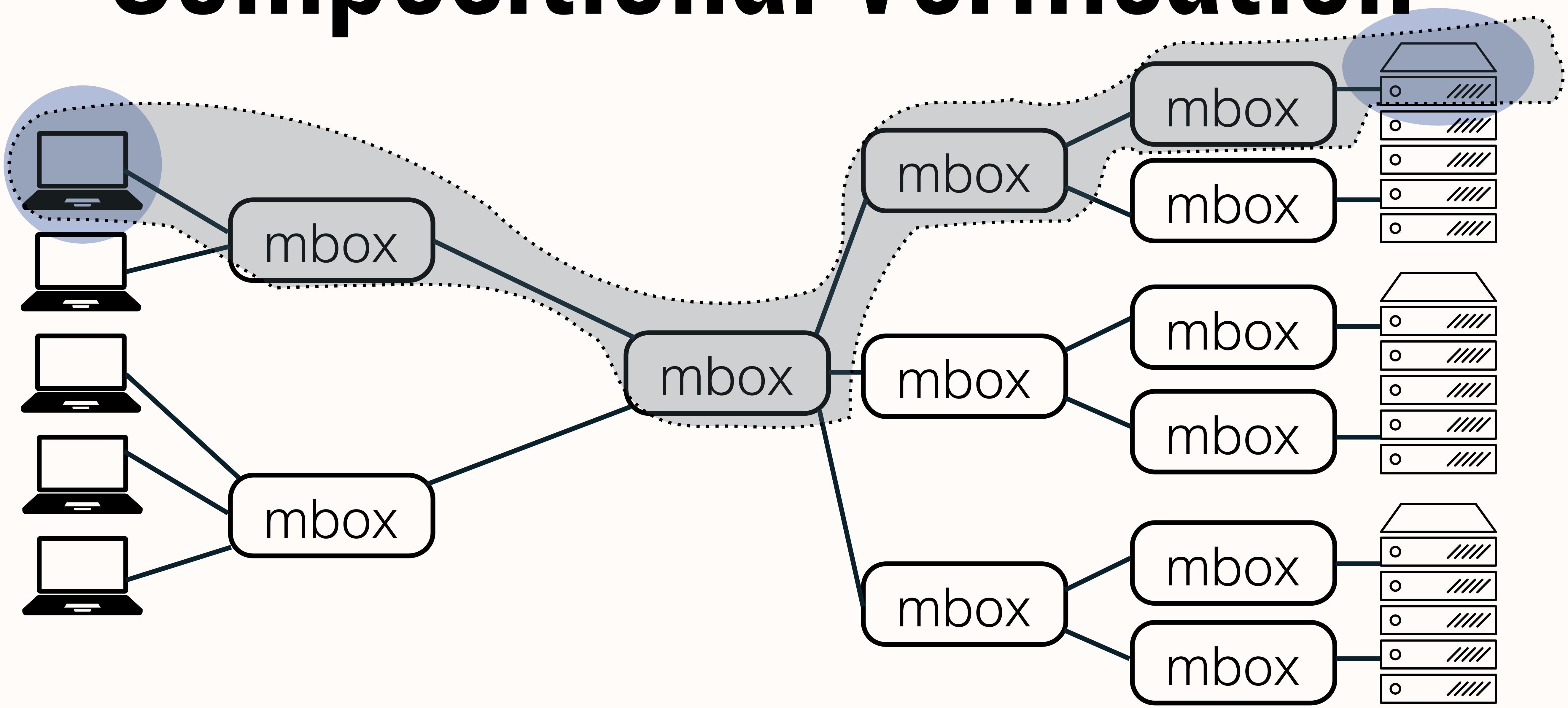
Compositional Verification



Compositional Verification



Compositional Verification



- Invariants talk about pairs of hosts.
  - When flow-parallel, need-only verify path.

Conclusion

- Real networks:
  - Contain mutable middleboxes.
  - Used to enforce rich connectivity invariants.

- Network verification needs to evolve to handle this.
- Several challenges
  - Right level of abstraction for specifying middleboxes.
  - Scalability, by leveraging compositional verification.
  - Future: Tractability of verification.

Backup

Does State Mutation Matter

- Do we even need to look at state evolution?
  - Check invariant for all possible states.
  - Approach used in tools like Margrave.
    - # of states is small (just whether connection established).
- False positives, some states may never occur.

Does State Mutation Matter

- Do we even need to look at state evolution?
  - Check invariant for all possible states.
  - Approach used in tools like Margrave.
    - # of states is small (just whether connection established).
- False positives, some states may never occur.

$conn(a \rightarrow b)$ Connection started by a to b.
Requires a to send packet to b, and b to respond

Can a packet from 'a' reach 'b'?