# Abstract Interpretation of Stateful Networks

**Kalev Alpernas**, Roman Manevich, Aurojit Panda,
Mooly Sagiv, Scott Shenker, Sharon Shoham,
and Yaron Velner

TEL AVIV UNIVERSITY אוניברסיטת תל אביב
אוניברסיטת בן-גוריון בנגב
NYU
Berkeley UNIVERSITY OF CALIFORNIA
האוניברסיטה העברית בירושלים THE HEBREW UNIVERSITY OF JERUSALEM

# Collaborators

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

אוניברסיטת בן-גוריון בנגב

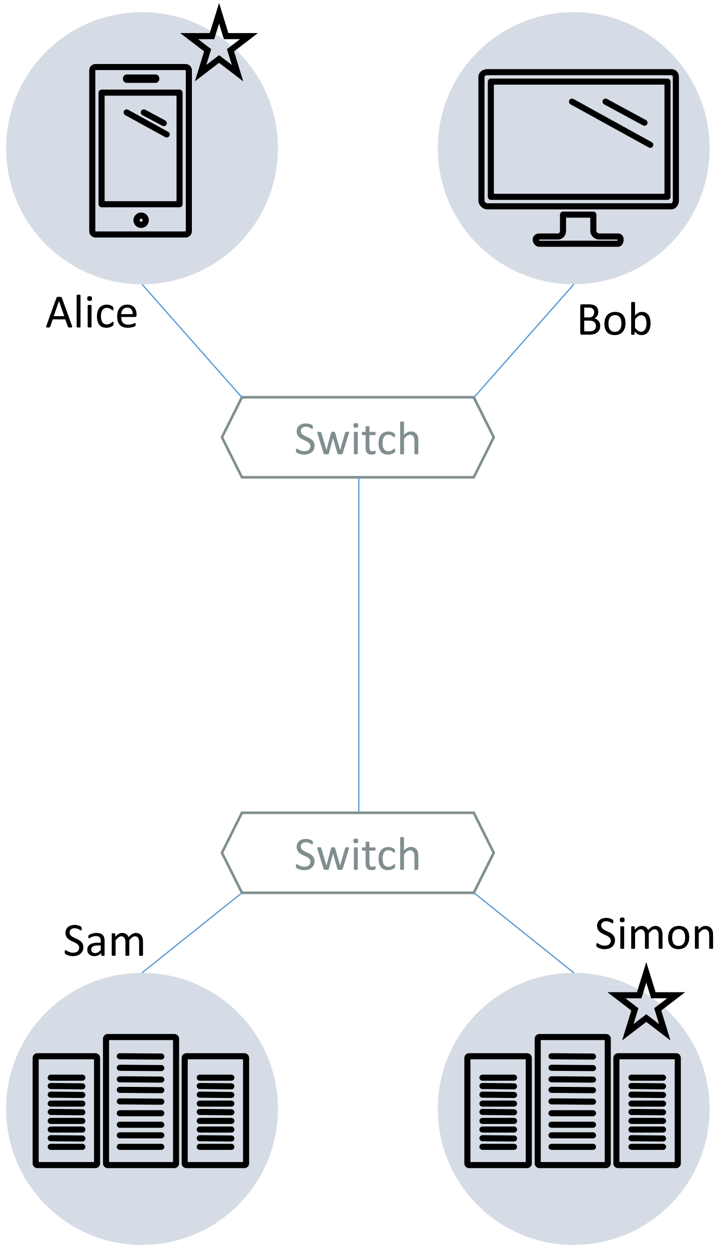
האוניברסיטה העברית בירושלים
THE HEBREW UNIVERSITY OF JERUSALEM

NYU

Berkeley
UNIVERSITY OF CALIFORNIA

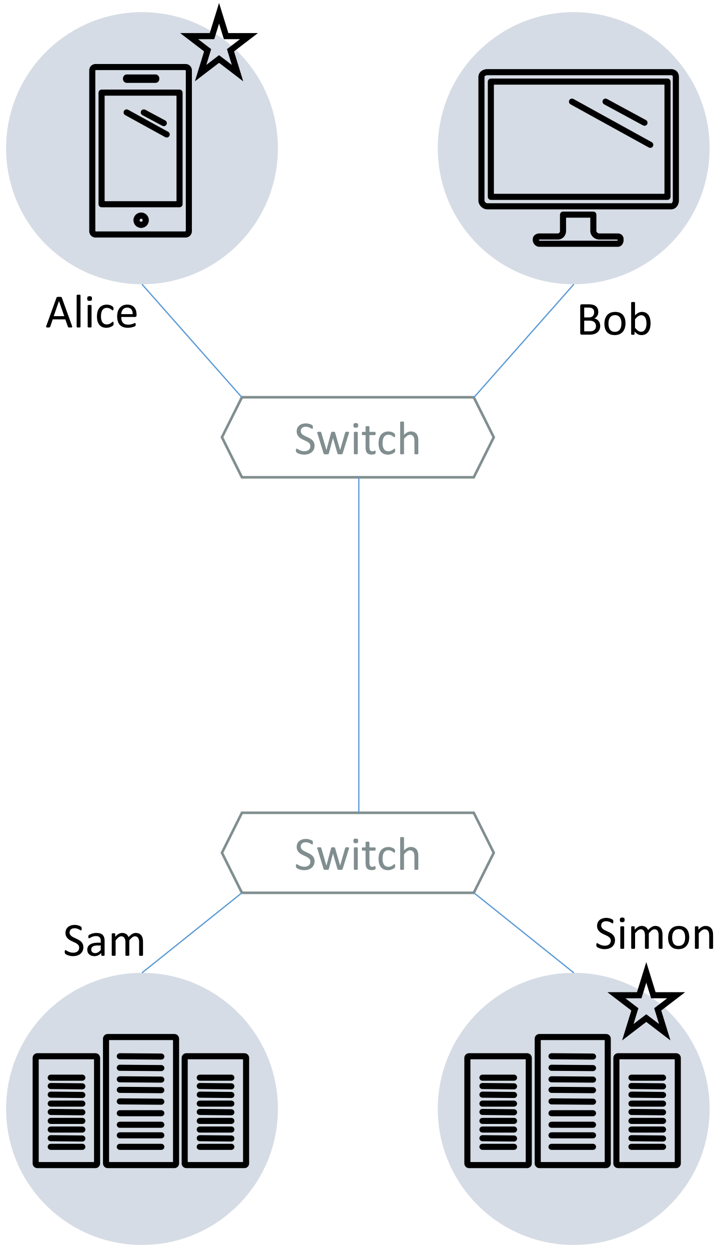
# Network Safety Verification

- Setting: Computer Networks
- Show that something bad cannot happen
- Isolation:
  - A packet of type **t** sent from host **A** never reaches host **B**
  - E.g., no packets from Simon to Bob



# (Stateless) Networks

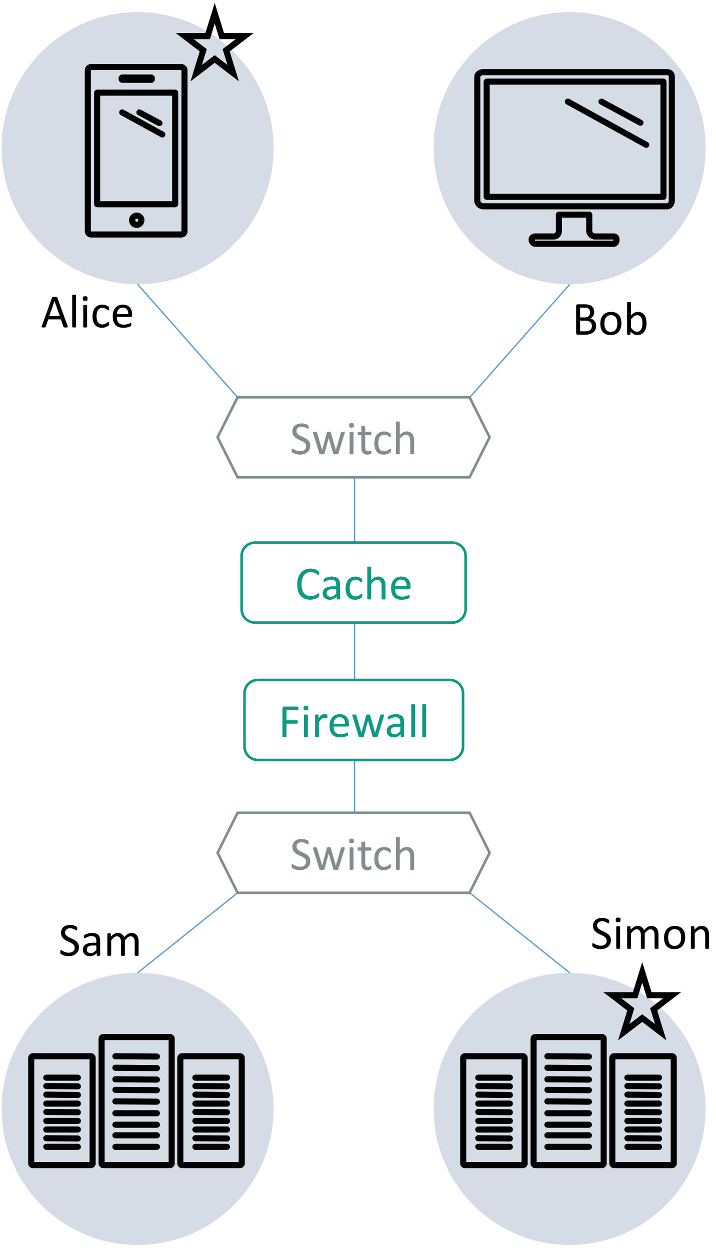
- Hosts
  - Finite set
- Switches
- Channels
- Packets
  - Packet headers
  - Source, destination, type fields



# Stateful Networks

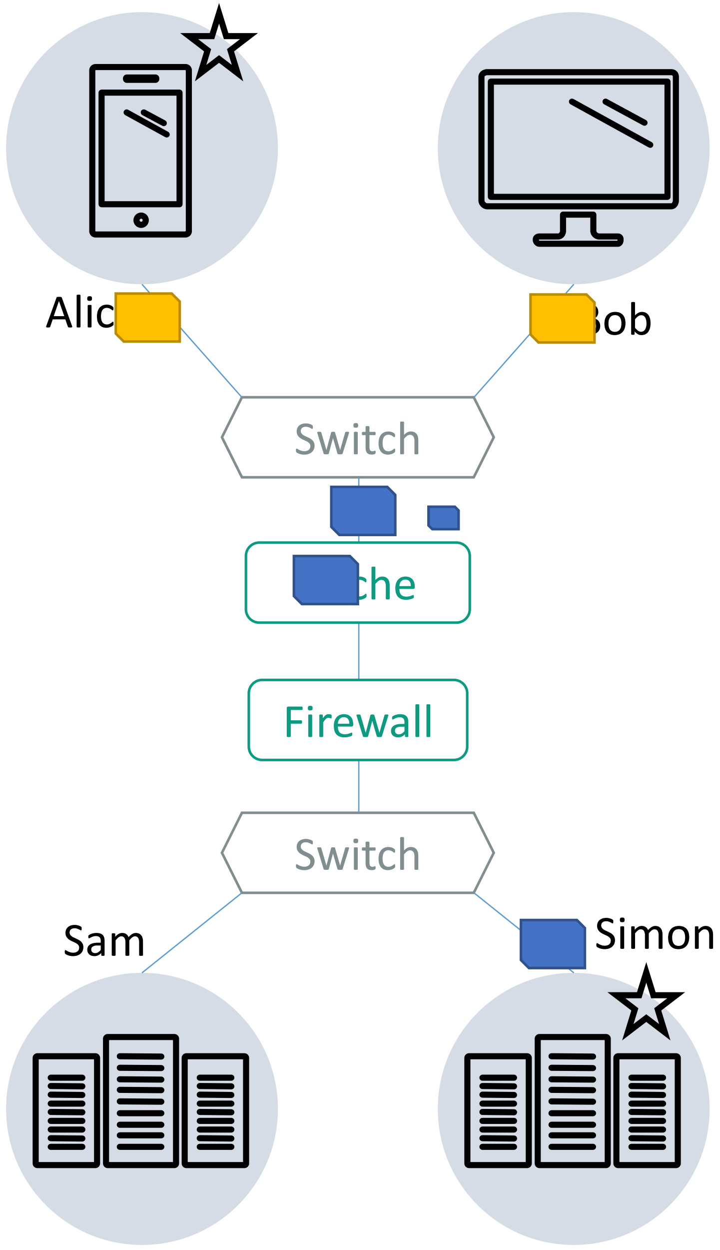
- Hosts
  - Finite set
- Switches
- Channels
- Packets
  - Packet headers
  - Source, destination, type fields

- **Middleboxes**



# Stateful Networks

- Middleboxes: Local functionality enhancements
  - Security (firewalls, IDSs,…)
  - Performance (caches, load balancers,…)
  - New functionality (proxies,…)

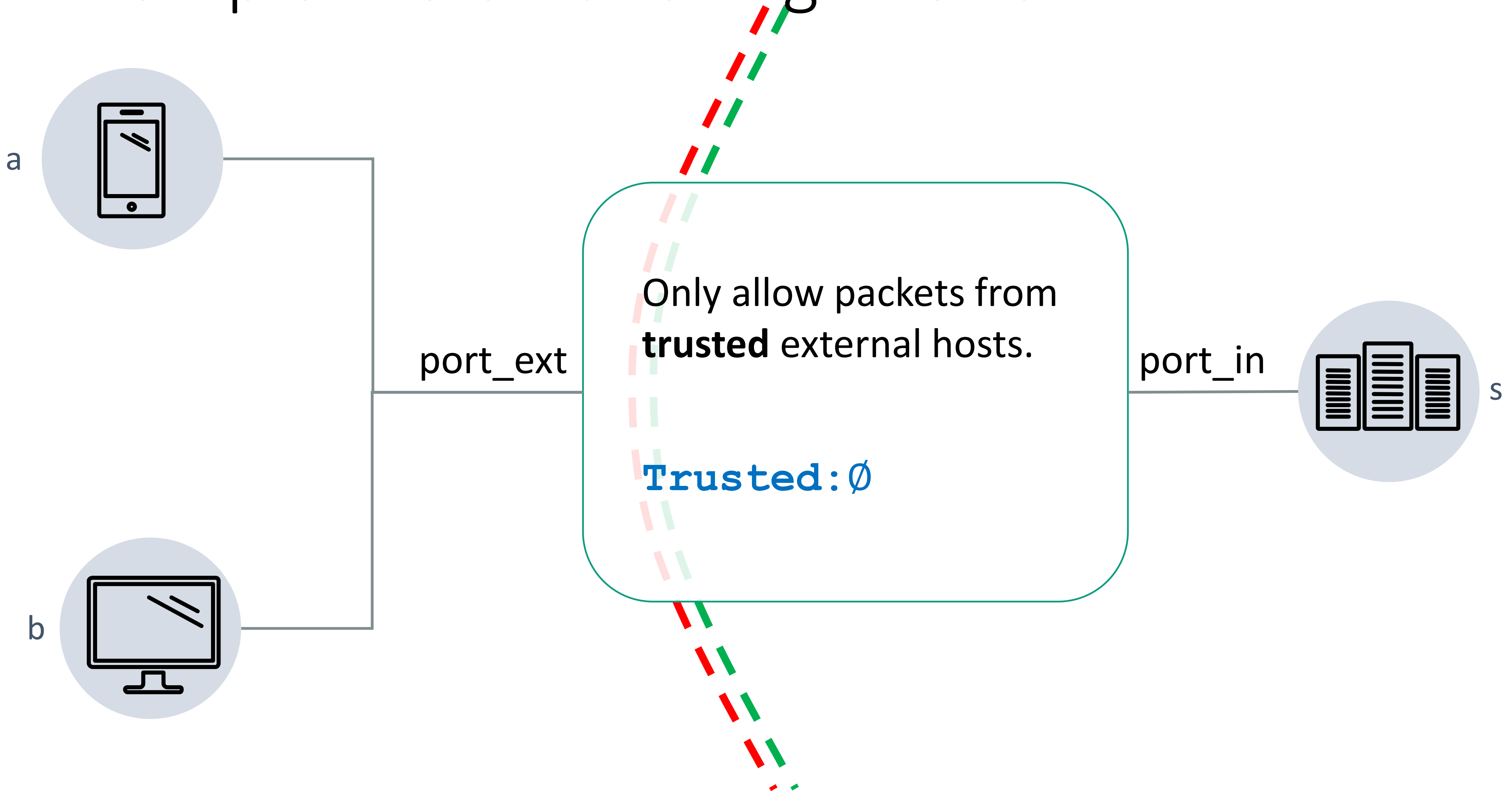


# Safety with Middleboxes

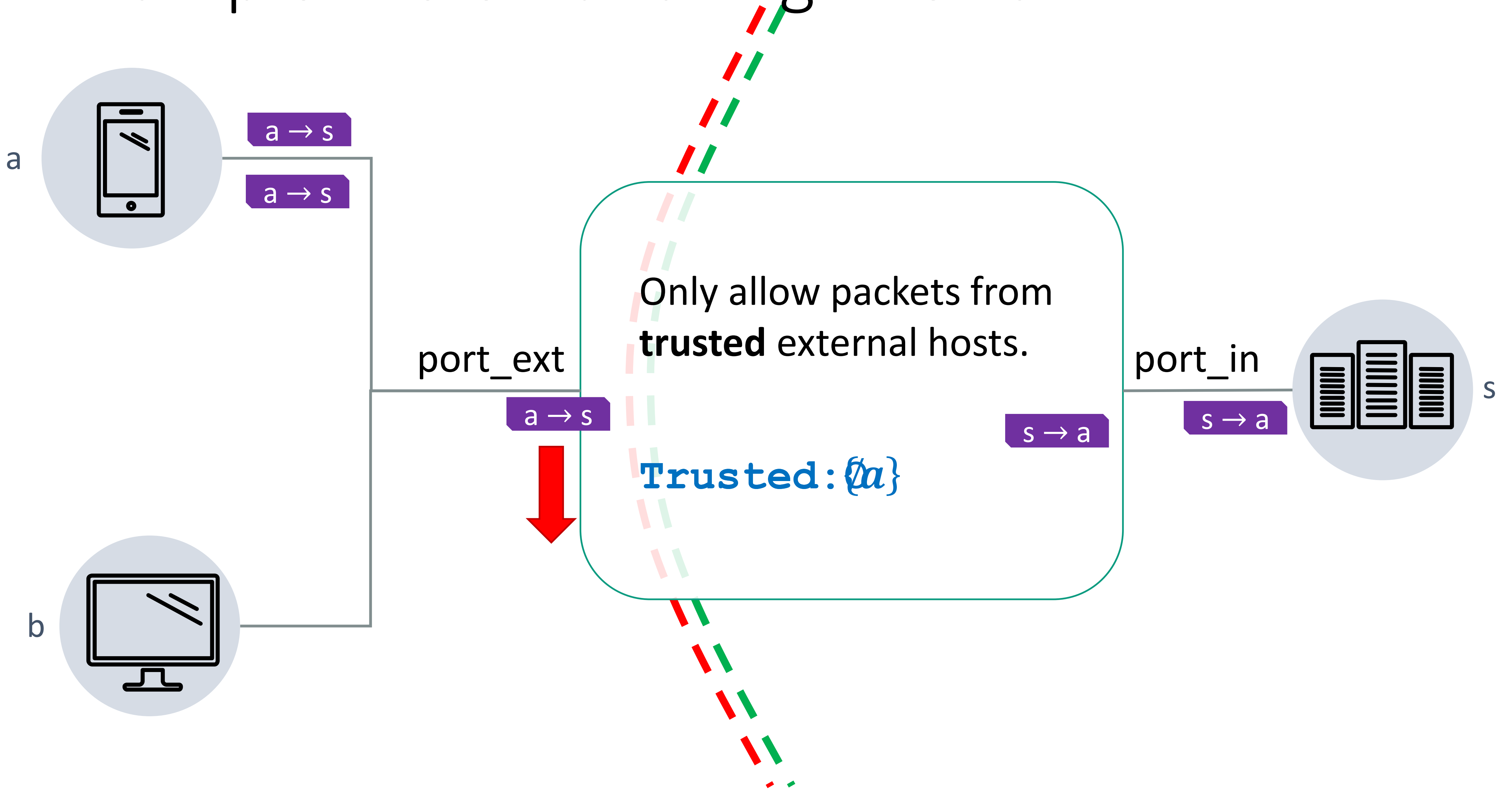
- For stateless networks
  - Safety is reducible to graph reachability

- Middleboxes make everything harder
  - Complex software systems
  - May rewrite packet headers
  - Behave differently over time – need to reason about history
    - Forwarding of a packet depends on previous packets
    - E.g. cache

# Example: Hole-Punching Firewall



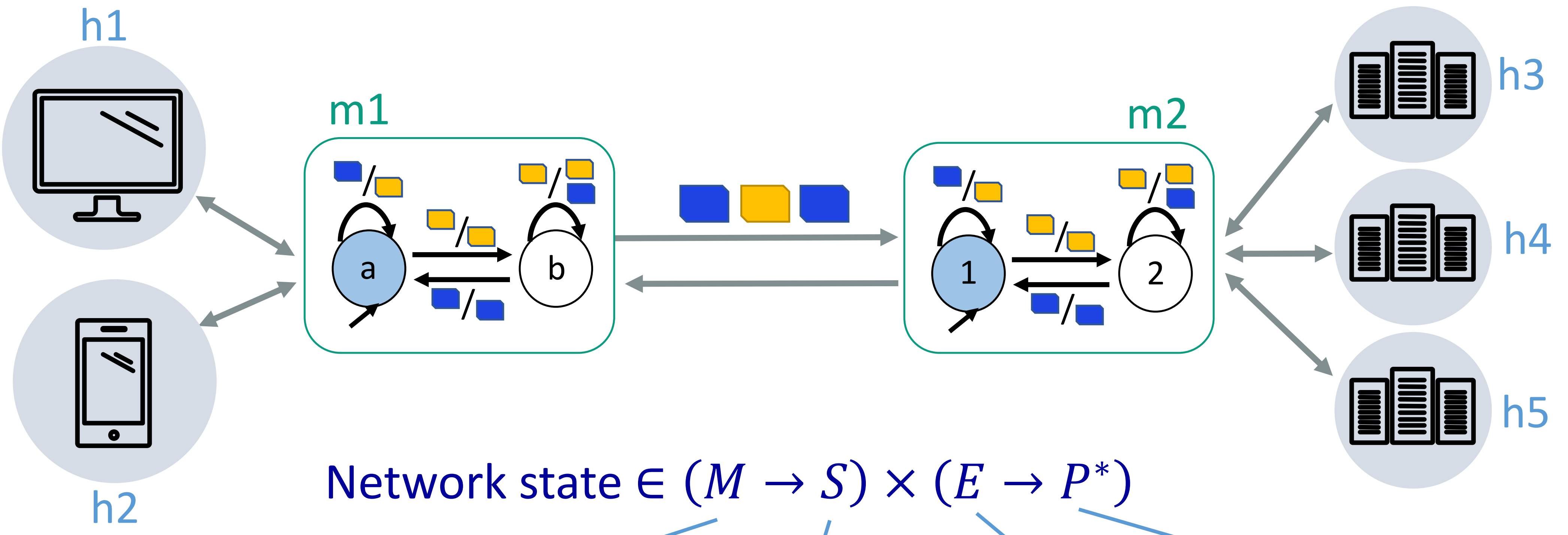
# Example: Hole-Punching Firewall



# What This Work Does and Does not Do

- We do not try to prove the correctness of middlebox implementations
- We do try to prove the correctness of the forwarding behaviour of the network
- FSM models suffice for this purpose

# Concrete Network ≈ Communicating FSMs




Network state $\in (M \rightarrow S) \times (E \rightarrow P^*)$

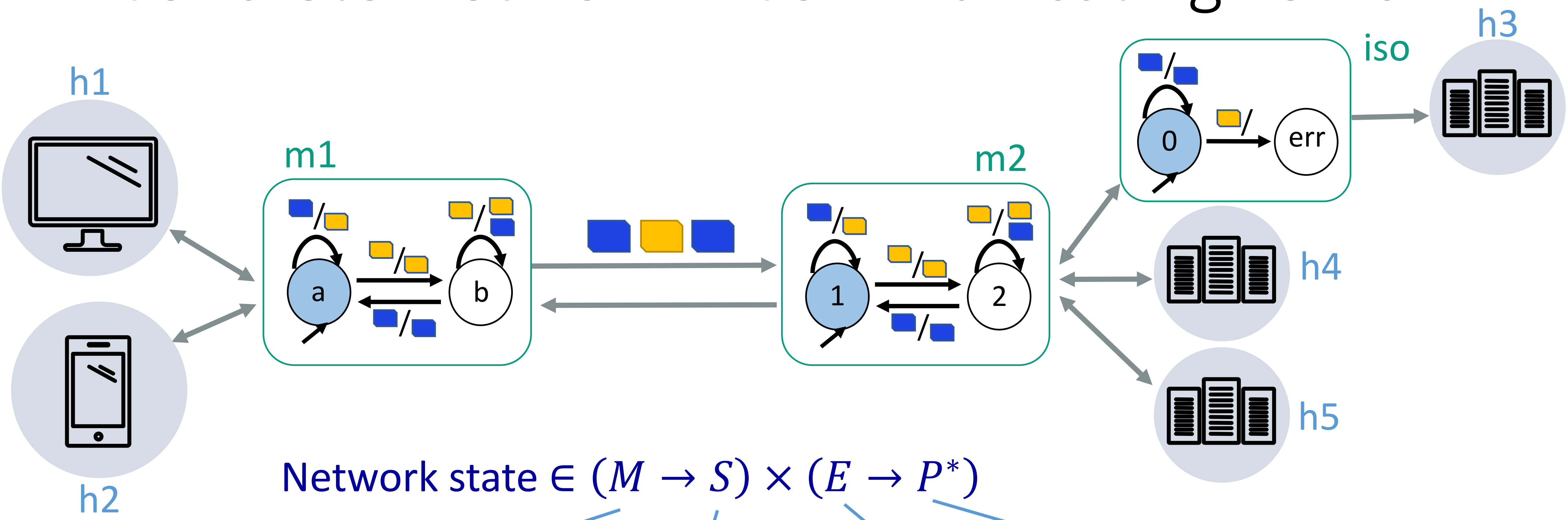
mboxes

mbox states

channels

packets ∈ Hosts × Hosts × T
<src, dst, tpe>

# Concrete Network ≈ Communicating FSMs




Network state $\in (M \rightarrow S) \times (E \rightarrow P^*)$

mboxes | mbox states | channels | packets ∈ Hosts × Hosts × T

<src, dst, tpe>

# Network Abstractions

(0) Concrete domain

mboxes mbox states channels packets

$$C = \mathbb{P}\big((M \to S) \times (E \to P^*)\big)$$

# Network Abstractions

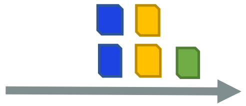
(0) Concrete domain

(1) Unordered channels

- Channels as multisets of packets

mboxes mbox states channels packets

$$C = \mathbb{P}\big((M \to S) \times (E \to P^*)\big)$$



$$\mathbb{P}\big((M \to S) \times (E \to \underbrace{P \to \mathbb{N}}_{\text{Infinite height}})\big)$$

# Network Abstractions

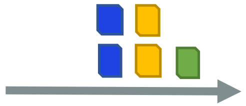
(0) Concrete domain

mboxes mbox states channels packets

$$C = \mathbb{P}\big((M \to S) \times (E \to P^*)\big)$$

(1) Unordered channels

- Channels as multisets of packets



$$\mathbb{P}\big((M \to S) \times (E \to \underbrace{P \to \mathbb{N}}_{\text{Infinite height}})\big)$$

Safety verification is decidable [TACAS'16]

- Reduction to/from Petri Net coverability
- EXPSPACE complexity

[TACAS'16] Y. Velner, K. Alpernas, A. Panda, A. Rabinovich, M. Sagiv, S. Shenker, S. Shoham: Some Complexity Results for Stateful Network Verification

# Network Abstractions

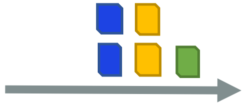
(0) Concrete domain

mboxes mbox states channels packets

$$C = \mathbb{P}\big((M \to S) \times (E \to P^*)\big)$$

(1) Unordered channels

- Channels as multisets of packets



$$\mathbb{P}\big((M \to S) \times (E \to P \to \mathbb{N})\big)$$

(2) Counter abstraction on channels

- Channels as sets of packets

$$\mathbb{P}\big((M \to S) \times (E \to \mathbb{P}(P))\big)$$

# Network Abstractions

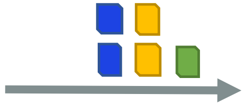
(0) Concrete domain

mboxes  mbox states  channels  packets

$$C = \mathbb{P}\big((M \to S) \times (E \to P^*)\big)$$

(1) Unordered channels

- Channels as multisets of packets



$$\mathbb{P}\big((M \to S) \times (E \to P \to \mathbb{N})\big)$$

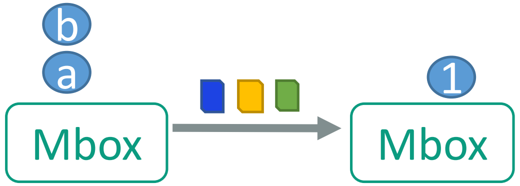
(2) Counter abstraction on channels

- Channels as sets of packets

$$\mathbb{P}\big((M \to S) \times (E \to \mathbb{P}(P))\big)$$

(3) Cartesian Abstraction

- No correlations between mboxes, channels, packets




$$\mathcal{A} = \big(M \to \mathbb{P}(S)\big) \times \big(E \to \mathbb{P}(P)\big)$$

# Network Abstractions

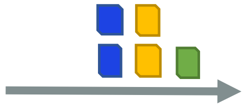
mboxes   mbox states   channels   packets

(0) Concrete domain

$$C = \mathbb{P}\big((M \to S) \times (E \to P^*)\big)$$

(1) Unordered channels

- Channels as multisets of packets



$$\mathbb{P}\big((M \to S) \times (E \to P \to \mathbb{N})\big)$$

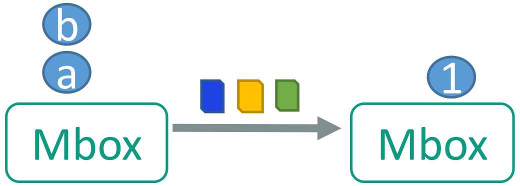
(2) Counter abstraction on channels

- Channels as sets of packets

$$\mathbb{P}\big((M \to S) \times (E \to \mathbb{P}(P))\big)$$

(3) Cartesian Abstraction

- No correlations between mboxes, channels, packets




$$\mathcal{A} = \big(M \to \mathbb{P}(S)\big) \times \big(E \to \mathbb{P}(P)\big)$$

Time(LFP$^{\#}$) = poly(|M|, |S|, |E|, |P|)

# Network Abstractions

mboxes mbox states channels packets

(0) Concrete domain

$$C = \mathbb{P}\big((M \to S) \times (E \to P^*)\big)$$

(1) Unordered channels

- Channels as multisets of packets

$$\mathbb{P}\big((M \to S) \times (E \to P \to \mathbb{N})\big)$$

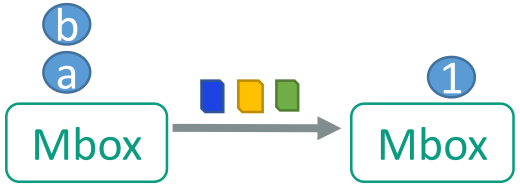
(2) Counter abstraction on channels

- Channels as sets of packets

$$\mathbb{P}\big((M \to S) \times (E \to \mathbb{P}(P))\big)$$

(3) Cartesian Abstraction

- No correlations between mboxes, channels, packets




$$\mathcal{A} = \big(M \to \mathbb{P}(S)\big) \times \big(E \to \mathbb{P}(P)\big)$$

Unfortunately
|S| = exp (|Hosts|)

Time(LFP$^{\#}$) = poly(|M|, |S|, |E|, |P|)

# Example: Firewall

AMDL: Abstract MBox Def. Lang.

- Similar to [SIGCOMM'16]

```
hole_punching_firewall =
  port_in ? <src,dst,tpe> =>
    trusted(dst) := true;
    port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
    src in trusted =>
      port_in ! <src,dst,tpe>
```

- States ≈ n-ary relations
- Topology agnostic
- Encode FSM compactly
  - For fixed topology finite state

[SIGCOMM'16] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, S. Licking: **Packet Transactions: High-Level Programming for Line-Rate**

# Example: Firewall

AMDL: Abstract MBox Def. Lang.

- Similar to [SIGCOMM’16]

- States ≈ n-ary relations
- Topology agnostic
- Encode FSM compactly
  - For fixed topology finite state

```
hole_punching_firewall =
  port_in ? <src,dst,tpe> =>
    trusted(dst) := true;
    port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
    src in trusted =>
      port_in ! <src,dst,tpe>
```

[SIGCOMM’16] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, S. Licking: **Packet Transactions: High-Level Programming for Line-Rate**

# Example: Firewall

AMDL: Abstract MBox Def. Lang.

- Similar to [SIGCOMM'16]

- States ≈ n-ary relations
- Topology agnostic
- Encode FSM compactly
  - For fixed topology finite state

```
hole_punching_firewall =
  port_in ? <src,dst,tpe> =>
    trusted(dst) := true;
    port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
    src in trusted =>
      port_in ! <src,dst,tpe>
```

[SIGCOMM'16] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, S. Licking: **Packet Transactions: High-Level Programming for Line-Rate**

# Example: Firewall

AMDL: Abstract MBox Def. Lang.

- Similar to [SIGCOMM'16]

- States ≈ n-ary relations
- Topology agnostic
- Encode FSM compactly
  - For fixed topology finite state

```
hole_punching_firewall =
  port_in ? <src,dst,tpe> =>
    trusted(dst) := true;
    port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
    src in trusted =>
      port_in ! <src,dst,tpe>
```

[SIGCOMM'16] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, S. Licking: **Packet Transactions: High-Level Programming for Line-Rate**

# Example: Firewall

AMDL: Abs

- Similar to [S

trusted: $\mathbb{P}$(Hosts)
$|S| = 2^{|Hosts|}$

```
hole_punching_firewall =
  port_in ? <src,dst,tpe> =>
    trusted(dst) := true;
    port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
    src in trusted =>
      port_in ! <src,dst,tpe>
```

- States ≈ n-ary relations
- Topology agnostic
- Encode FSM compactly
  - For fixed topology finite state

[SIGCOMM'16] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, S. Licking: **Packet Transactions: High-Level Programming for Line-Rate**

# Middlebox-level Abstraction

$$\mathrm{Time}(\mathrm{LFP}^{\#}) = \mathrm{poly}(|M|, |S|, |E|, |P|)$$

- Problem: Middlebox state space exponential in number of hosts

# Middlebox-level Abstraction

$$\text{Time}(\text{LFP}^{\#}) = \text{poly}(|M|, |S|, |E|, |P|)$$

- Problem: Middlebox state space exponential in number of hosts

- Proposal: apply another Cartesian abstraction
  - Ignore some correlations *within* a middlebox state

# Middlebox-level Abstraction

$$\text{Time}(\text{LFP}^{\#}) = \text{poly}(|M|, |S|, |E|, |P|)$$

- Problem: Middlebox state space exponential in number of hosts
- Proposal: apply another Cartesian abstraction
  - Ignore some correlations *within* a middlebox state
  - **How to decompose a state into sub-states?**

# Packet State

- Alternative (isomorphic) state representation

- Maps each input to the ‘description of the program execution’

- Depends on:
  - The restrictions **AMDL** places on middlebox queries and state updates
  - Finite packet space
  - Reactive communicating system

# Packet State Example

(src, dst, type) ↦ `{queries which hold}`

```
(1,_,_) ↦ {}
(2,_,_) ↦ {}
```

```
hole_punching_firewall =      // hosts ∈ {1, 2}
  port_in ? <src,dst,tpe> =>
     trusted(dst) := true; port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
     srcT: src in trusted => port_in ! <src,dst,tpe>
```

Query name

Query

# Packet State Example

(src, dst, type) ↦ `{queries which hold}`

```
(1,_,_) ↦ {}
(2,_,_) ↦ {}
```

(1,_,_)

```
hole_punching_firewall =      // hosts ∈ {1, 2}
  port_in ? <src,dst,tpe> =>
     trusted(dst) := true; port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
     srcT: src in trusted => port_in ! <src,dst,tpe>
```

Query name

Query

# Packet State Example

(src, dst, type) ↦ **{queries which hold}**

```
(1,_,_) ↦ {}
(2,_,_) ↦ {}
```

(1,_,_)

```
hole_punching_firewall =      // hosts ∈ {1, 2}
  port_in ? <src,dst,tpe> =>
     trusted(dst) := true; port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
     srcT: src in trusted => port_in ! <src,dst,tpe>
```

Query name

Query

# Packet State Example

(src, dst, type) ↦ **{queries which hold}**

```
(1,_,_) ↦ {}
(2,_,_) ↦ {}
```

(_,1,_)

```
hole_punching_firewall =      // hosts ∈ {1, 2}
  port_in ? <src,dst,tpe> =>
     trusted(dst) := true; port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
     srcT: src in trusted => port_in ! <src,dst,tpe>
```

Query name

Query

# Packet State Example

(src, dst, type) ↦ `{queries which hold}`

```
(1,_,_) ↦ {}
(2,_,_) ↦ {}
```

```
(1,_,_) ↦ {srcT}
(2,_,_) ↦ {}
```

(_,1,_)

```
hole_punching_firewall =      // hosts ∈ {1, 2}
  port_in ? <src,dst,tpe> =>
     trusted(dst) := true; port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
     srcT: src in trusted => port_in ! <src,dst,tpe>
```

Query name

Query

# Packet State Example

(src, dst, type) ↦ `{queries which hold}`

```
(1,_,_) ↦ {}
(2,_,_) ↦ {}
```

```
(1,_,_) ↦ {srcT}
(2,_,_) ↦ {}
```

(1,_,_)

```
hole_punching_firewall =      // hosts ∈ {1, 2}
  port_in ? <src,dst,tpe> =>
     trusted(dst) := true; port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
     srcT: src in trusted => port_in ! <src,dst,tpe>
```

Query name

Query

# Packet State Example

(src, dst, type) ↦ `{queries which hold}`

```
(1,_,_) ↦ {}
(2,_,_) ↦ {}
```

```
(1,_,_) ↦ {srcT}
(2,_,_) ↦ {}
```

(1,_,_)

```
hole_punching_firewall =      // hosts ∈ {1, 2}
  port_in ? <src,dst,tpe> =>
     trusted(dst) := true; port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
     srcT: src in trusted => port_in ! <src,dst,tpe>
```

Query name

Query

# Cartesian Abstraction Over the Packet State

```
(1,_,_) ↦ {}   {srcT}

(2,_,_) ↦ {}   {srcT}
```

```
hole_punching_firewall =      // hosts ∈ {1, 2}
  port_in ? <src,dst,tpe> =>
     trusted(dst) := true; port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
     srcT: src in trusted => port_in ! <src,dst,tpe>
```

Query name

Query

# Cartesian Abstraction Over the Packet State

```
(1,_,_) ↦ {}    {srcT}

(2,_,_) ↦ {}    {srcT}
```

$\gamma$

```
(1,_,_) ↦ {}          (1,_,_) ↦ {}
(2,_,_) ↦ {}          (2,_,_) ↦ {srcT}

(1,_,_) ↦ {srcT}      (1,_,_) ↦ {srcT}
(2,_,_) ↦ {}          (2,_,_) ↦ {srcT}
```

```
hole_punching_firewall =      // hosts ∈ {1, 2}
  port_in ? <src,dst,tpe> =>
     trusted(dst) := true; port_ext ! <src,dst,tpe>
  |
  port_ext ? <src,dst,tpe> =>
     srcT: src in trusted => port_in ! <src,dst,tpe>
```

Query name

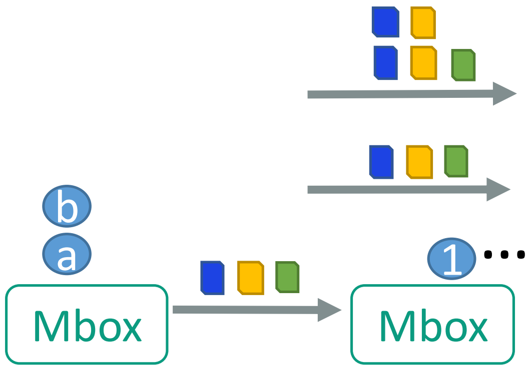
Query

# Summary: Network Abstractions

(1) Unordered channels

(2) Counter abstraction on channels

(3) Network-level Cartesian Abstraction

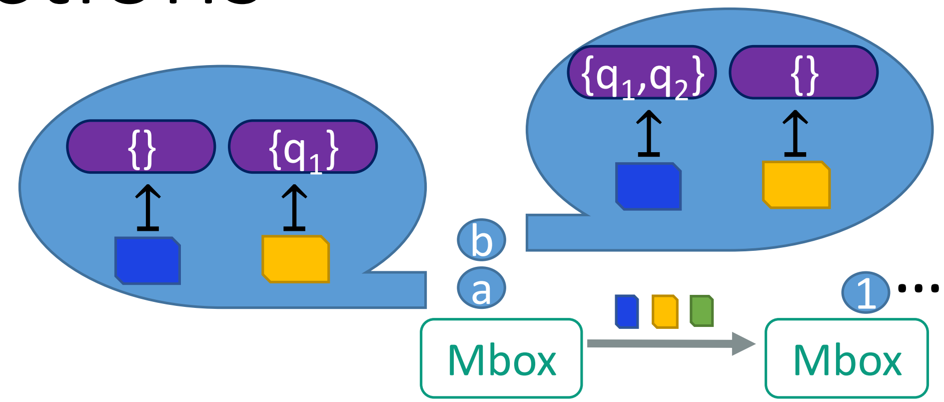


# Summary: Network Abstractions

(1) Unordered channels

(2) Counter abstraction on channels

(3) Network-level Cartesian Abstraction



# Summary: Network Abstractions

(1) Unordered channels

(2) Counter abstraction on channels

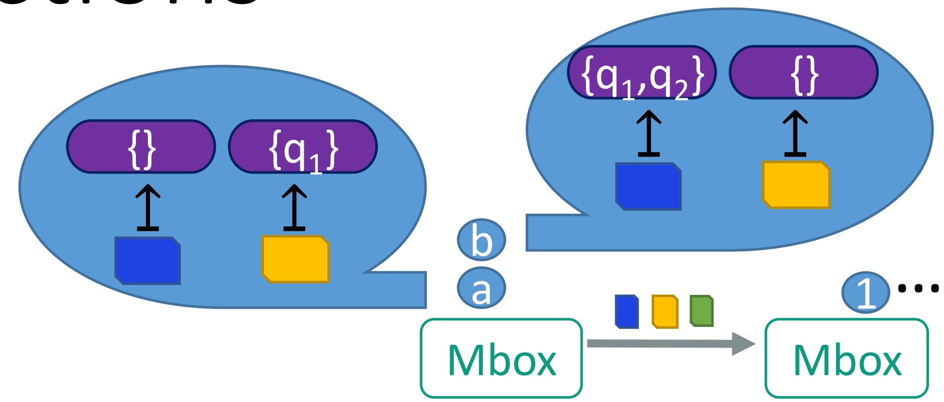
(3) Network-level Cartesian Abstraction

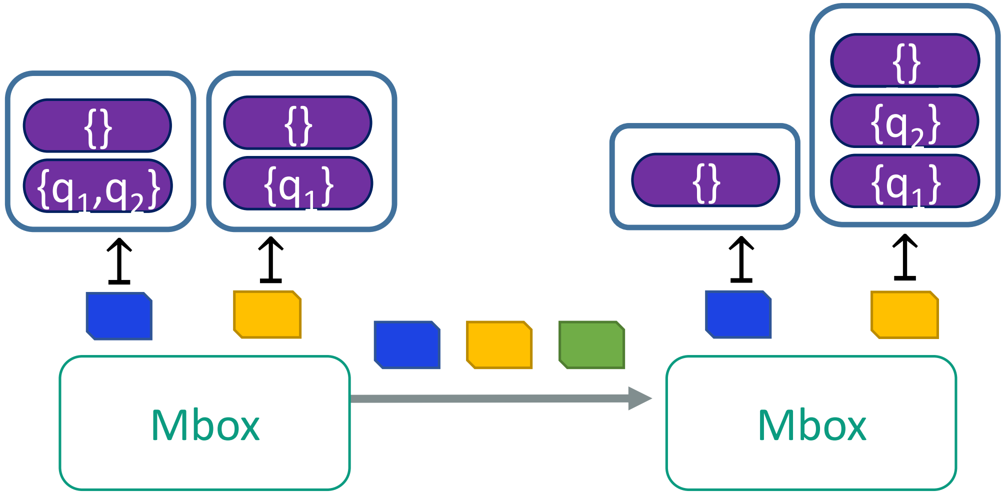
(4) Middlebox-level Cartesian abstraction

- No correlations between packet states
- But **keep** correlations between **queries**

$$\mathcal{A} = \left(M \to P \to \mathbb{P}(\mathbb{P}(Q))\right) \times \left(E \to \mathbb{P}(P)\right)$$

Time(LFP#) = poly(|M|, |P|, $2^{|Q|}$, |E|)



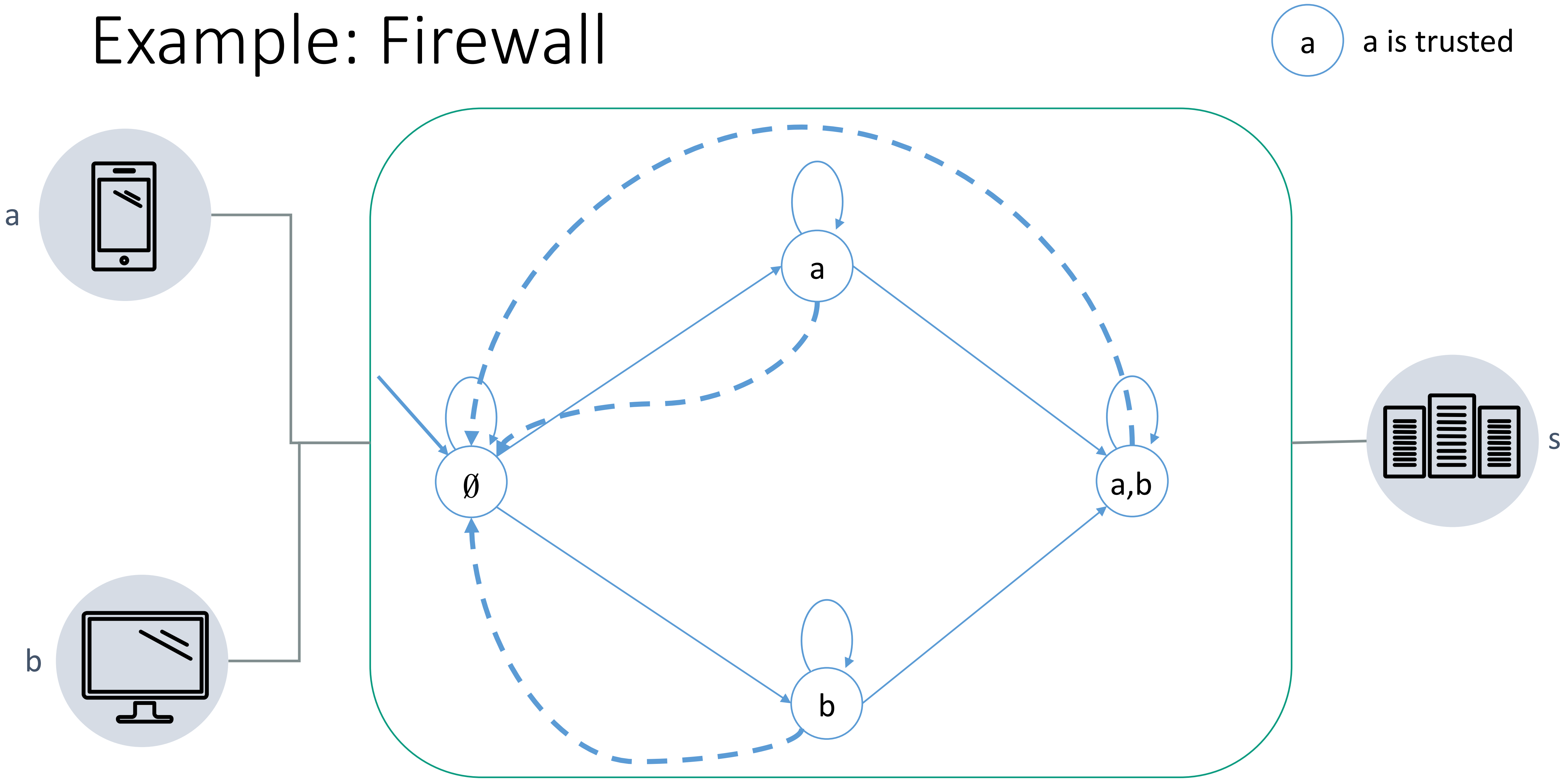
# When is This Precise?

# Reverting Middleboxes

- Middleboxes may independently revert to their initial state
  - Non-deterministically

- Similar to recovery from hardware failure

# Example: Firewall



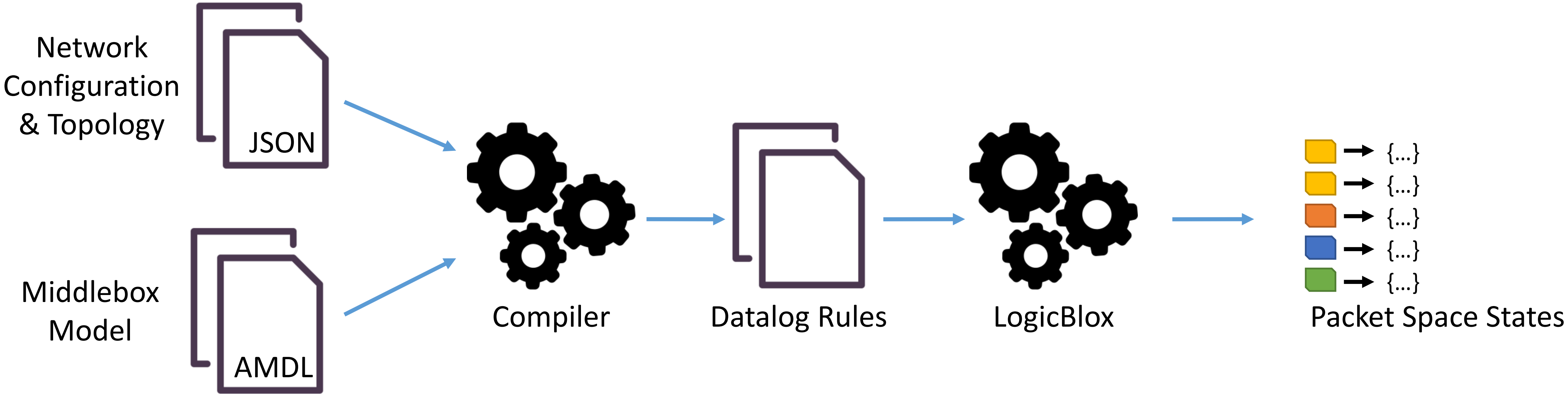
# Reverting Middleboxes

*Theorem*: If the network is correct in the presence of packet reordering and middlebox reverts then our analysis is precise.

- Common wisdom: Network resets make verification harder
  - Reachability for Petri nets with reset arcs is undecidable [1]
- But: Simplifies the task of automatic verification of networks
  - The analysis is precise for isolation
  - No false alarms

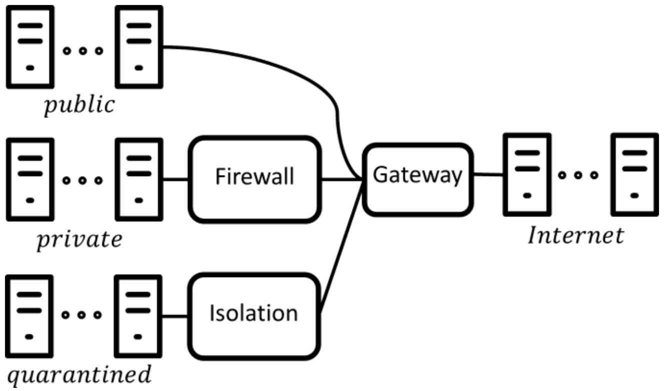
[1] Araki, T., & Kasami, T. (1976). Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*.

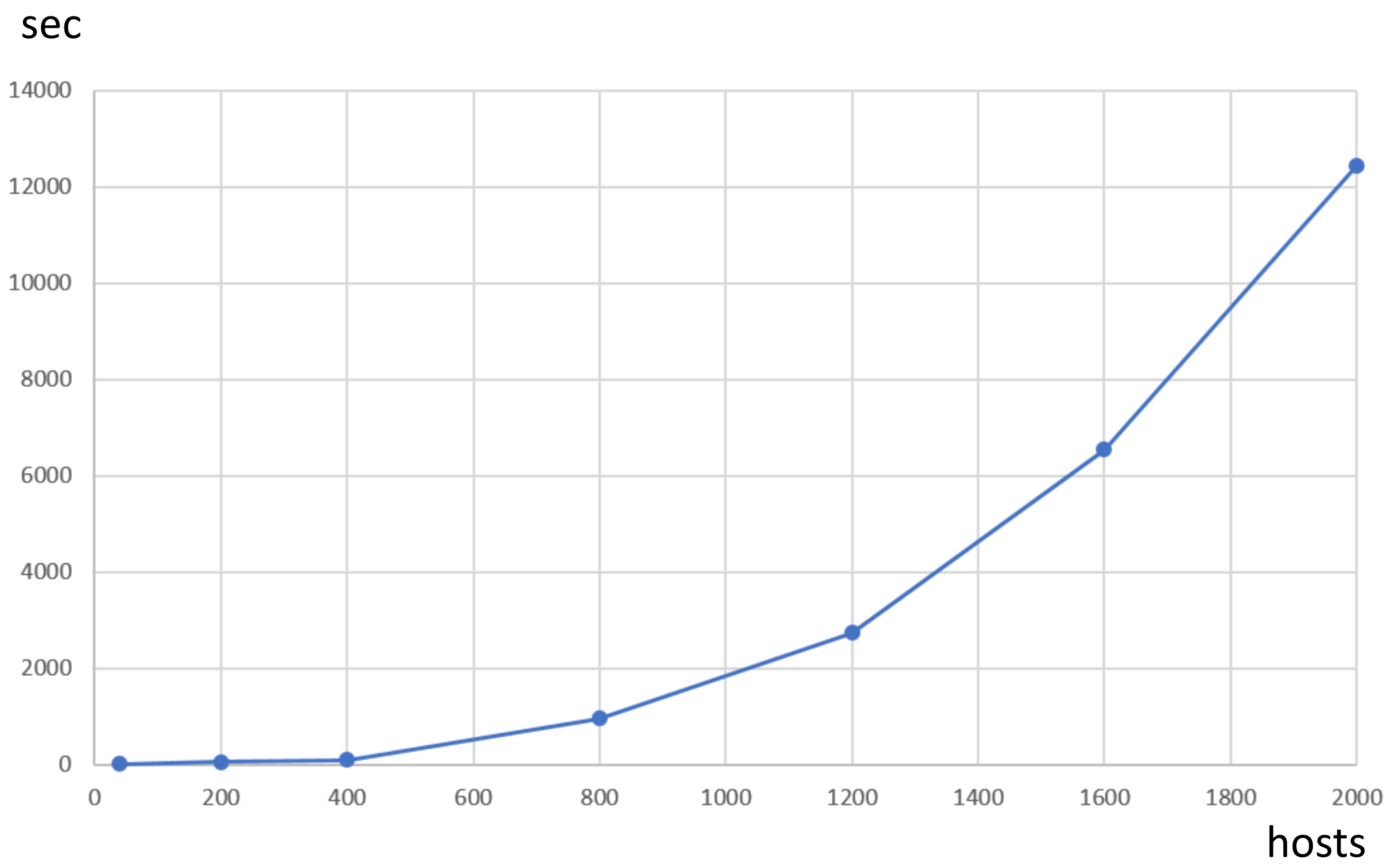
# Experimental Results



# Scalability Testing - Hosts

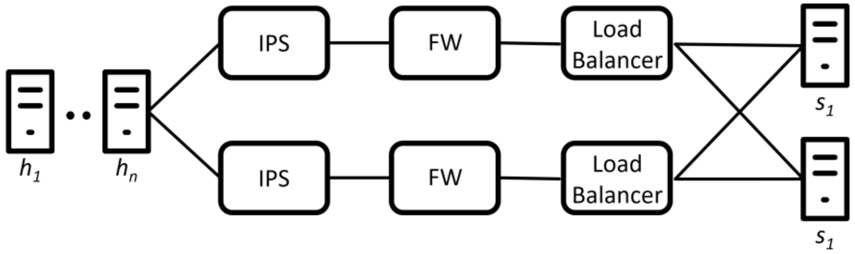
- Enterprise network with 3 subnets
  - Each with a different security policy
- Isolation between *quarantined* and *Internet*

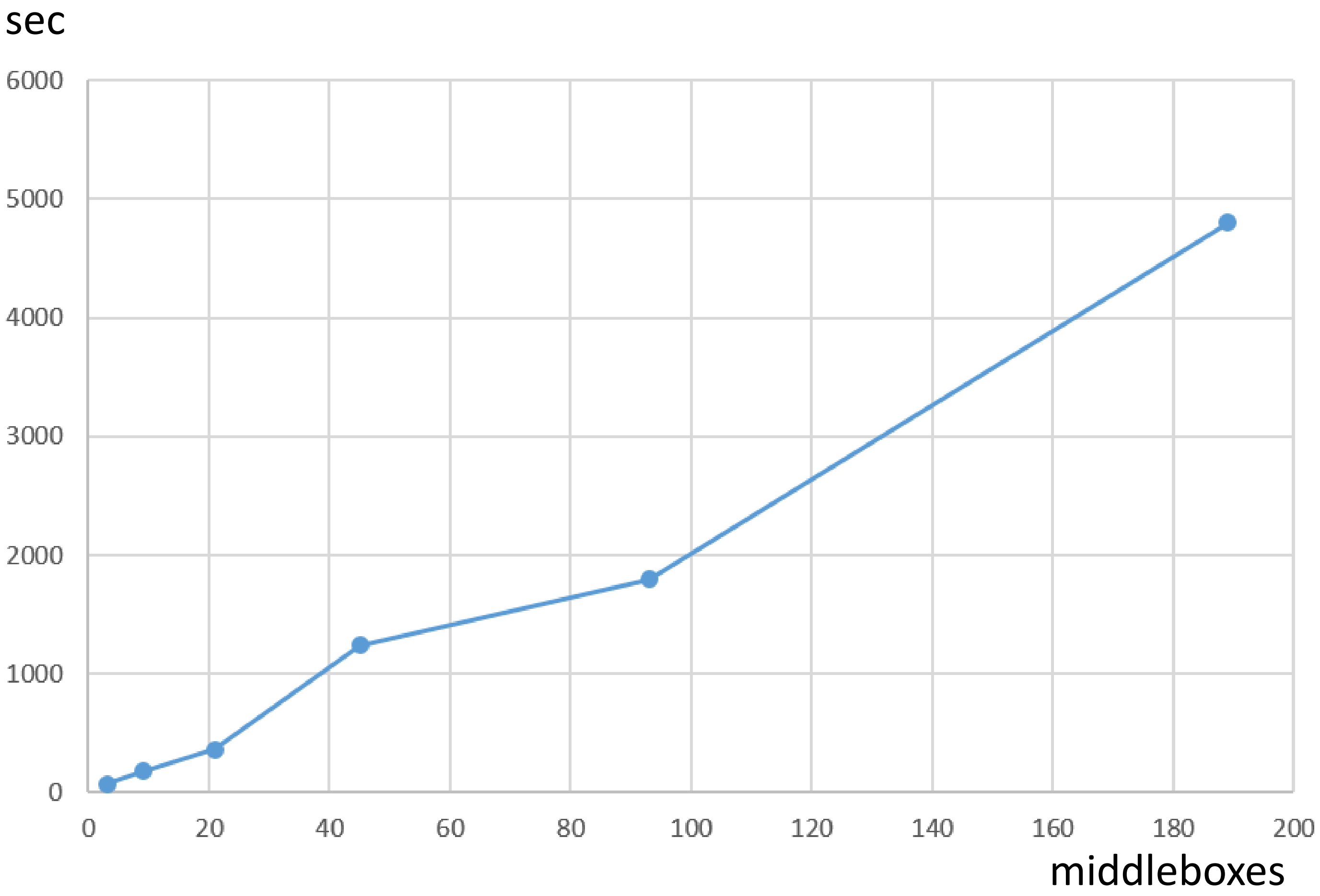


# Scalability Testing - Middleboxes

- Servers with parallel middlebox chains
- Scaled the number of chains
- Isolation – packets from $h_1$ never reach bottom flow



# Summary

- Abstract interpretation of stateful networks
  - Unordered + Counter + Cartesian X2
- Packet effect semantics for middelboxes
  - Enables middlebox-level Cartesian abstraction
- Precise for unordered channels + reverting middleboxes