ALEMBIC: AUTOMATED MODEL INFERENCE FOR STATEFUL NETWORK FUNCTIONS

Soo-Jin Moon
Jeffrey Helt, Yifei Yuan, Yves Bieri,
Sujata Banerjee, Vyas Sekar, Wenfei Wu, Mihalis Yannakakis, Ying Zhang

Carnegie Mellon Univ., Princeton Univ., Intentionet, ETH Zurich,
VMware Research, Tsinghua Univ., Columbia Univ., Facebook, Inc.

Contributions by Soo-Jin Moon were made in-part during a former internship at Hewlett Packard Labs.
Other contributors from former employees at Hewlett Packard Labs include Sujata Banerjee, Ying Zhang and Wenfei Wu.

Stateful Network Functions (NFs) in Modern Networks

Firewalls and NATs

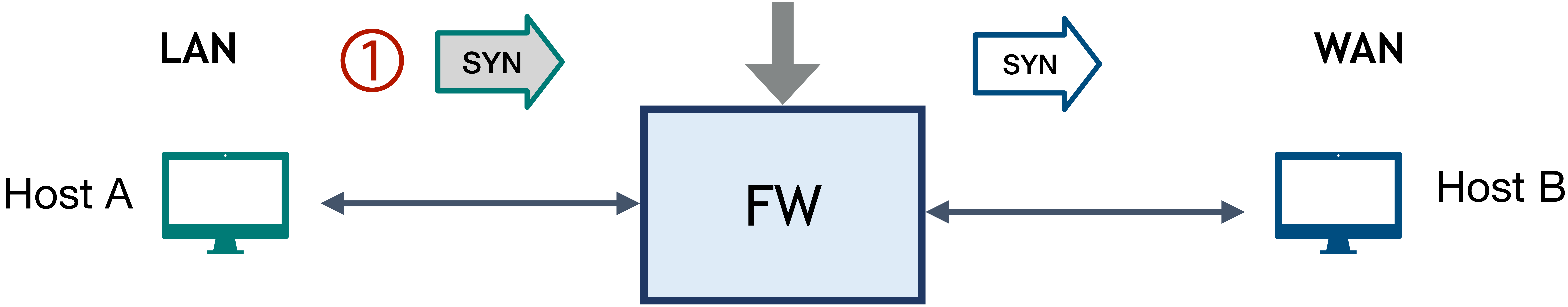
Load balancers

IDS/IPSs

Modern networks contain a wide range of complex **stateful network functions** from **many vendors**

Motivating Example: Stateful Firewall (FW)

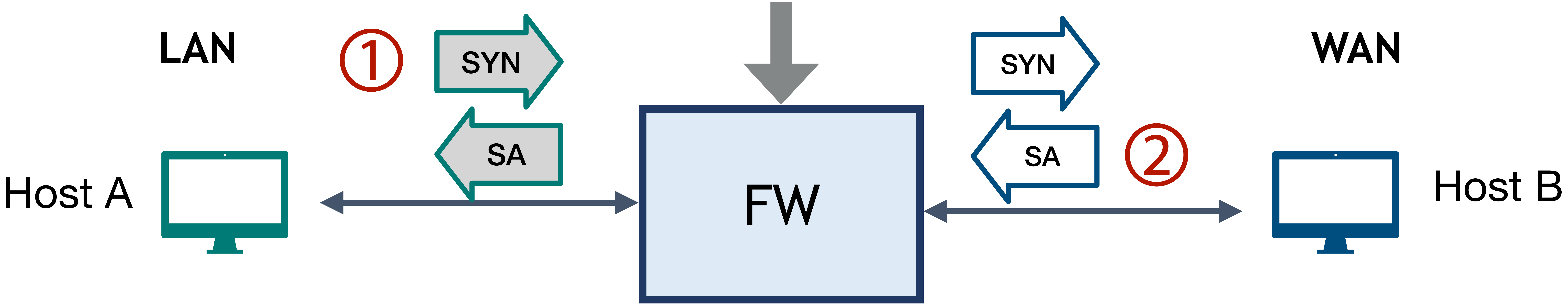
If a connection is ESTABLISHED from the LAN,
allow TCP traffic from the WAN else DROP



Connection Map:

Motivating Example: Stateful Firewall (FW)

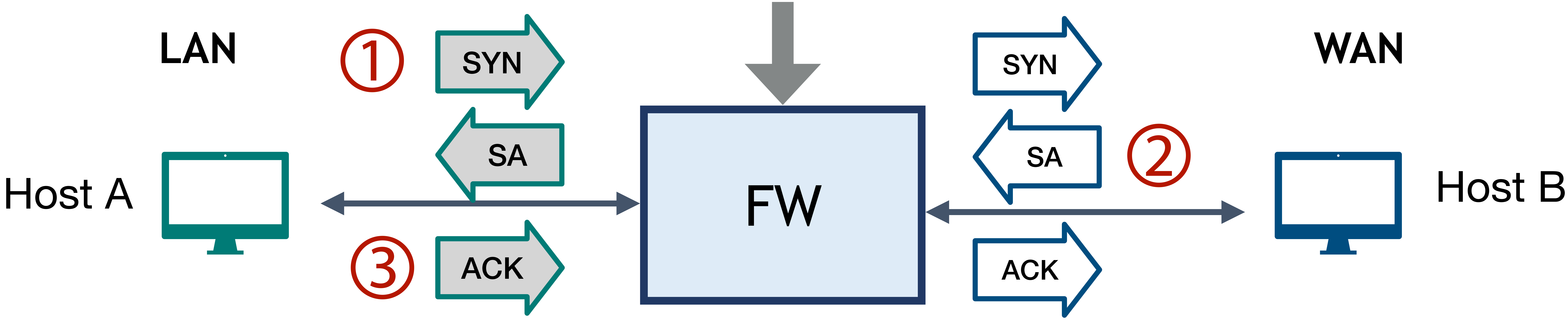
If a connection is ESTABLISHED from the LAN, allow TCP traffic from the WAN else DROP



Connection Map:

Motivating Example: Stateful Firewall (FW)

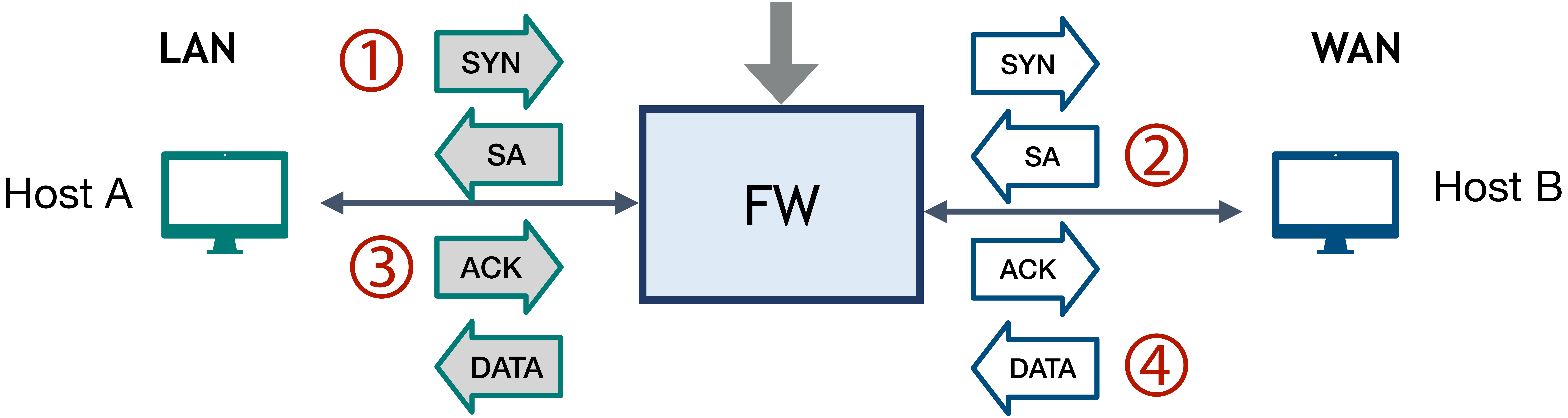
If a connection is ESTABLISHED from the LAN, allow TCP traffic from the WAN else DROP



Connection Map:

Motivating Example: Stateful Firewall (FW)

If a connection is ESTABLISHED from the LAN, allow TCP traffic from the WAN else DROP

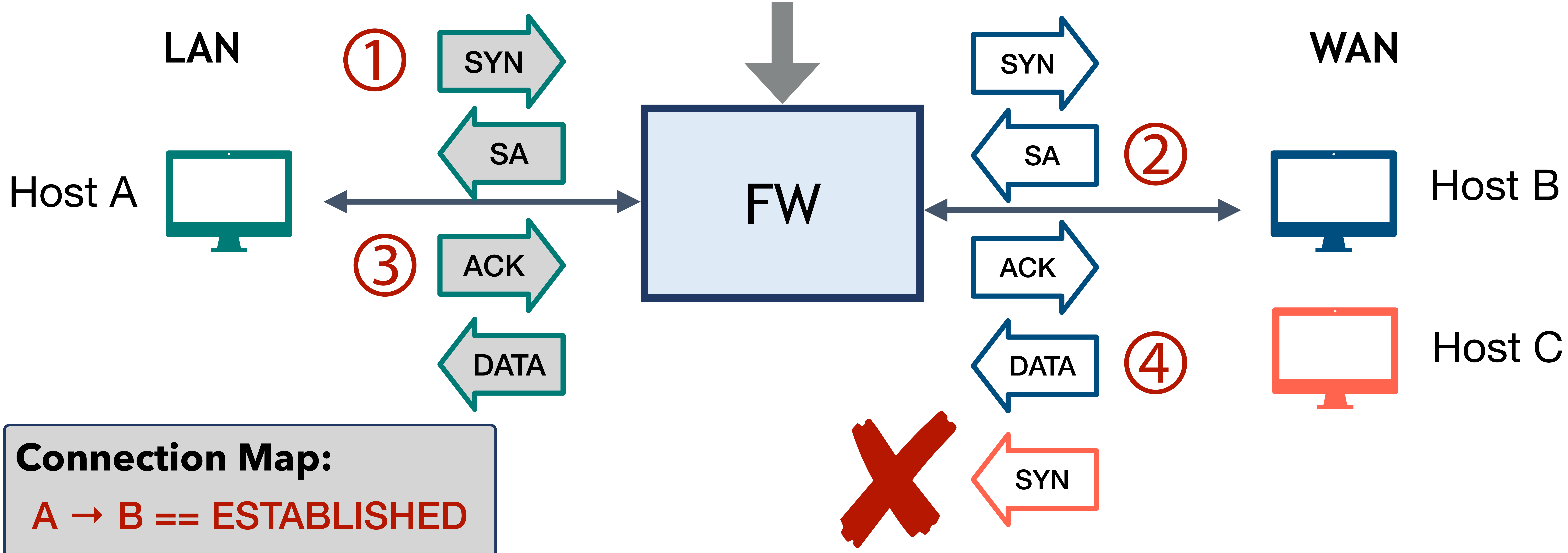


Connection Map:

A → B == ESTABLISHED

Motivating Example: Stateful Firewall (FW)

If a connection is ESTABLISHED from the LAN, allow TCP traffic from the WAN else DROP

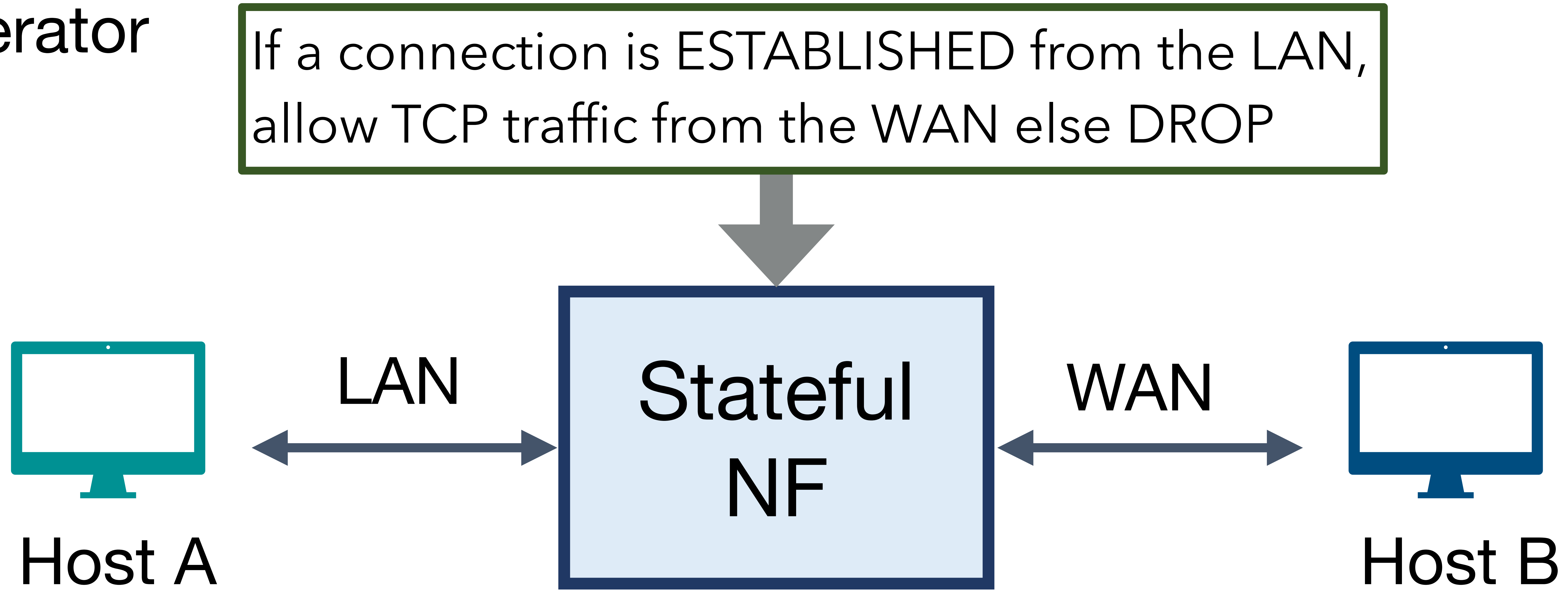


Connection Map:

A → B == ESTABLISHED

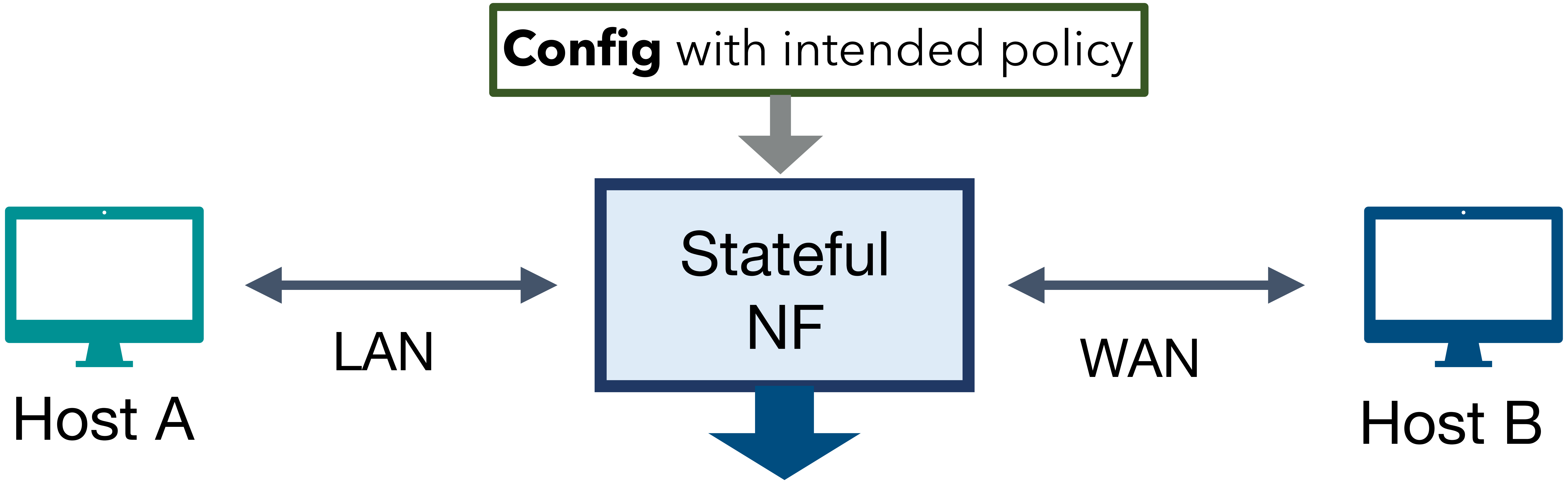
Network Testing and Verification

- Is the policy implemented correctly?
- Can we check before on-boarding?



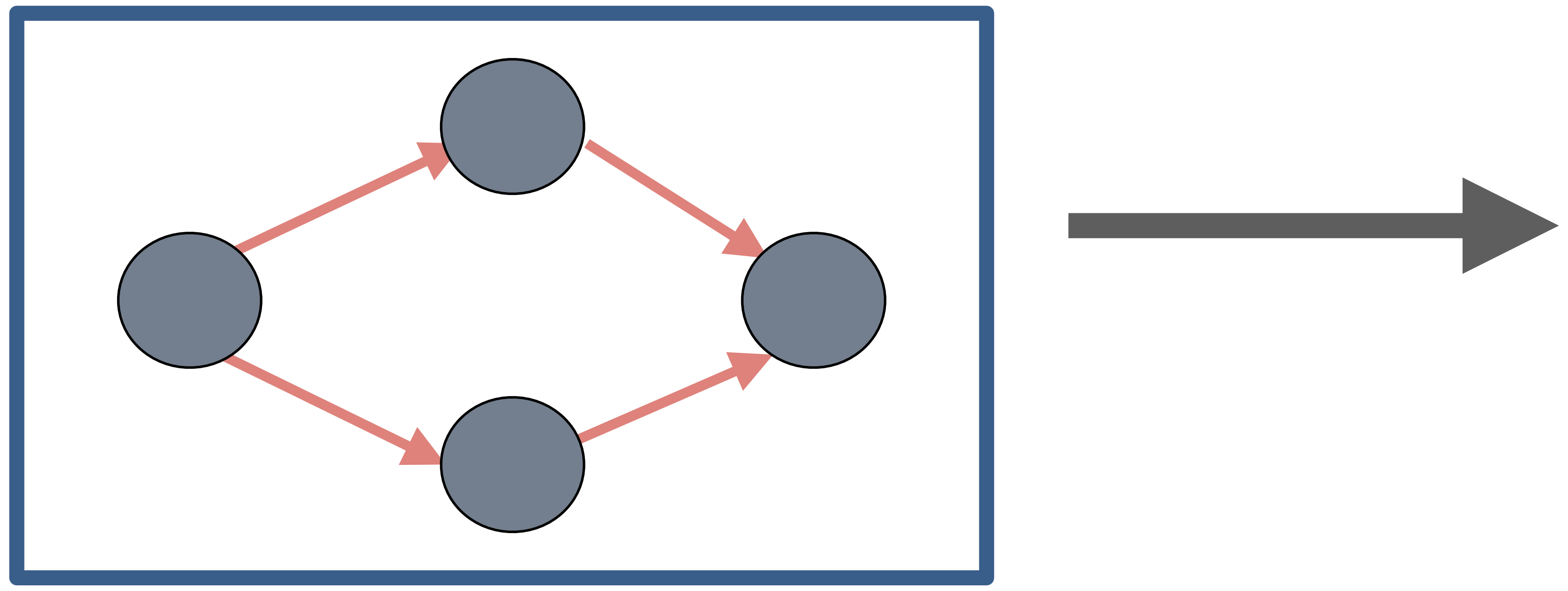
We need network testing/verification tools (e.g.,VMN , SYMNET, BUZZ...)

Today: Need NF Models for Testing and Verification



Model

(e.g., finite state machine)

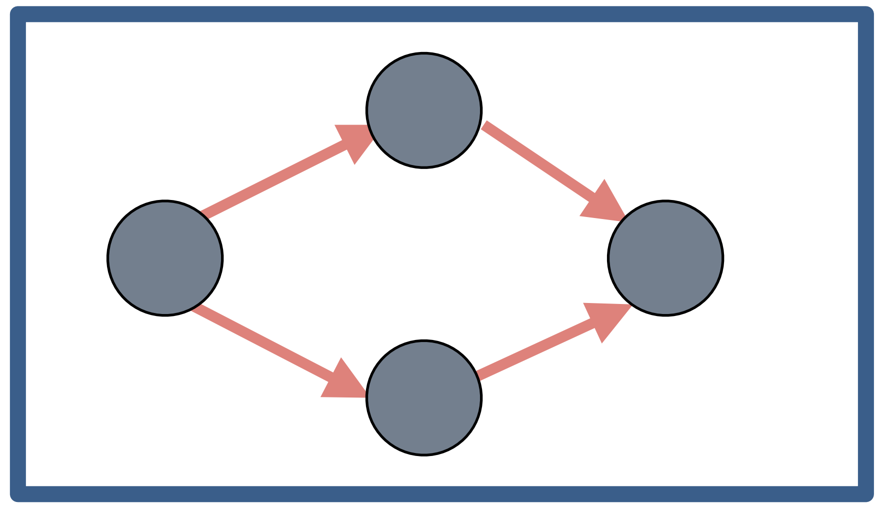


Testing
Verification
On-boarding

Today, these NF models are **handwritten** based on manual investigation

Limitation of Handwritten Model: Inaccuracy

Handwritten FW model



Network testing tool
e.g., BUZZ [NSDI 16]

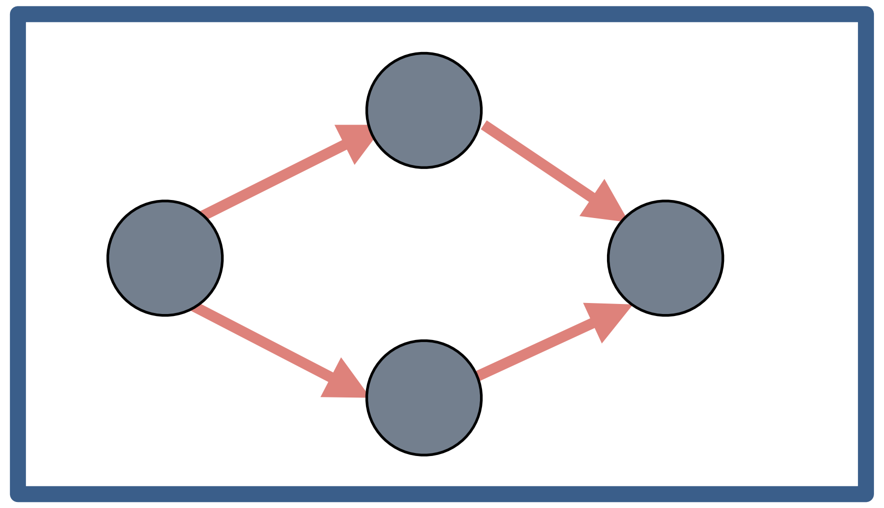
≠

Error!

Test traffic (from BUZZ)	Intended Policy	Real FW	Handwritten Model
SYN	SYN	SYN	SYN
SA	SA	SA	SA
SYN	✗	✗	SYN ✓

Limitation of Handwritten Model: Inaccuracy

Handwritten
FW model



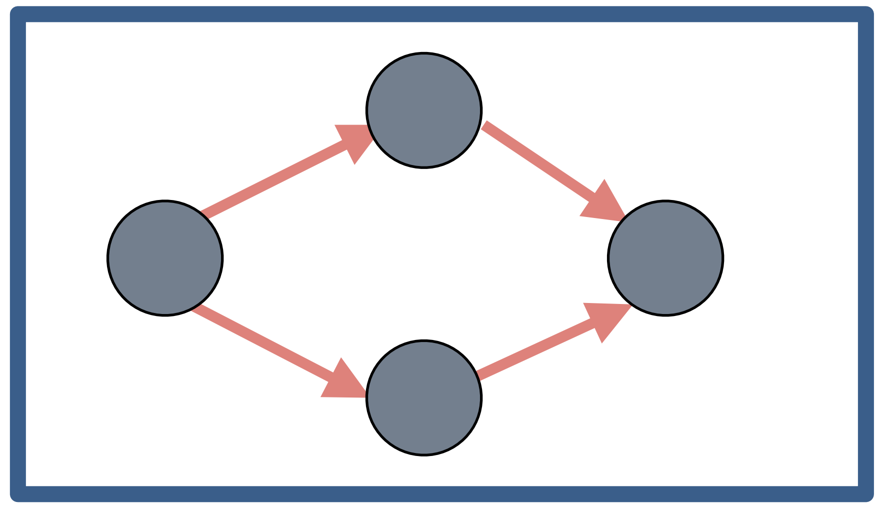
Network testing tool
e.g., BUZZ [NSDI 16]

≠

Test traffic (from BUZZ)	Intended Policy	Real FW	Handwritten Model
SYN	SYN	SYN	SYN
SA	SA	SA	SA
SYN	✗	✗	SYN ✓

Limitation of Handwritten Model: Inaccuracy

Handwritten FW model

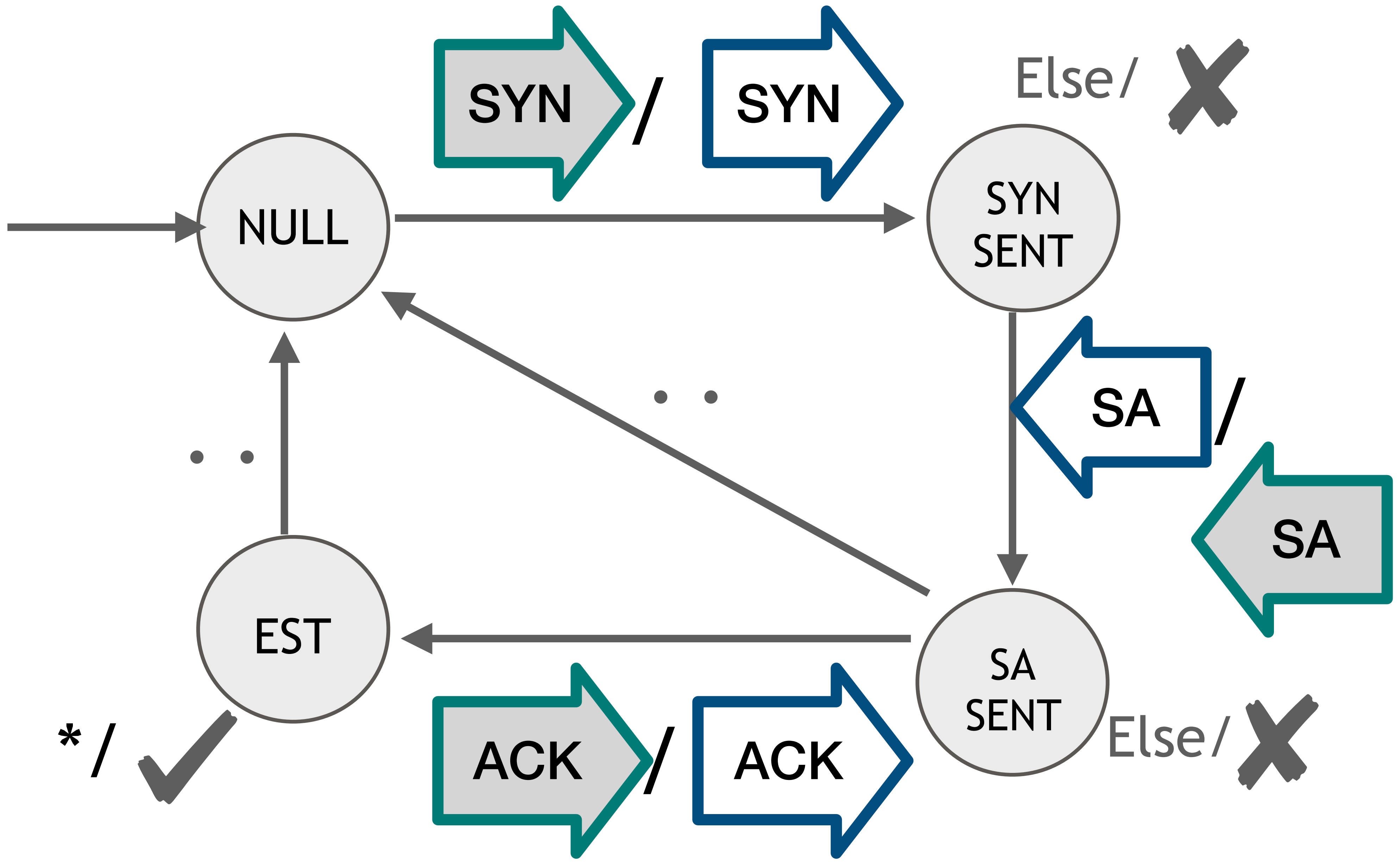


→

Network testing tool
e.g., BUZZ [NSDI 16]

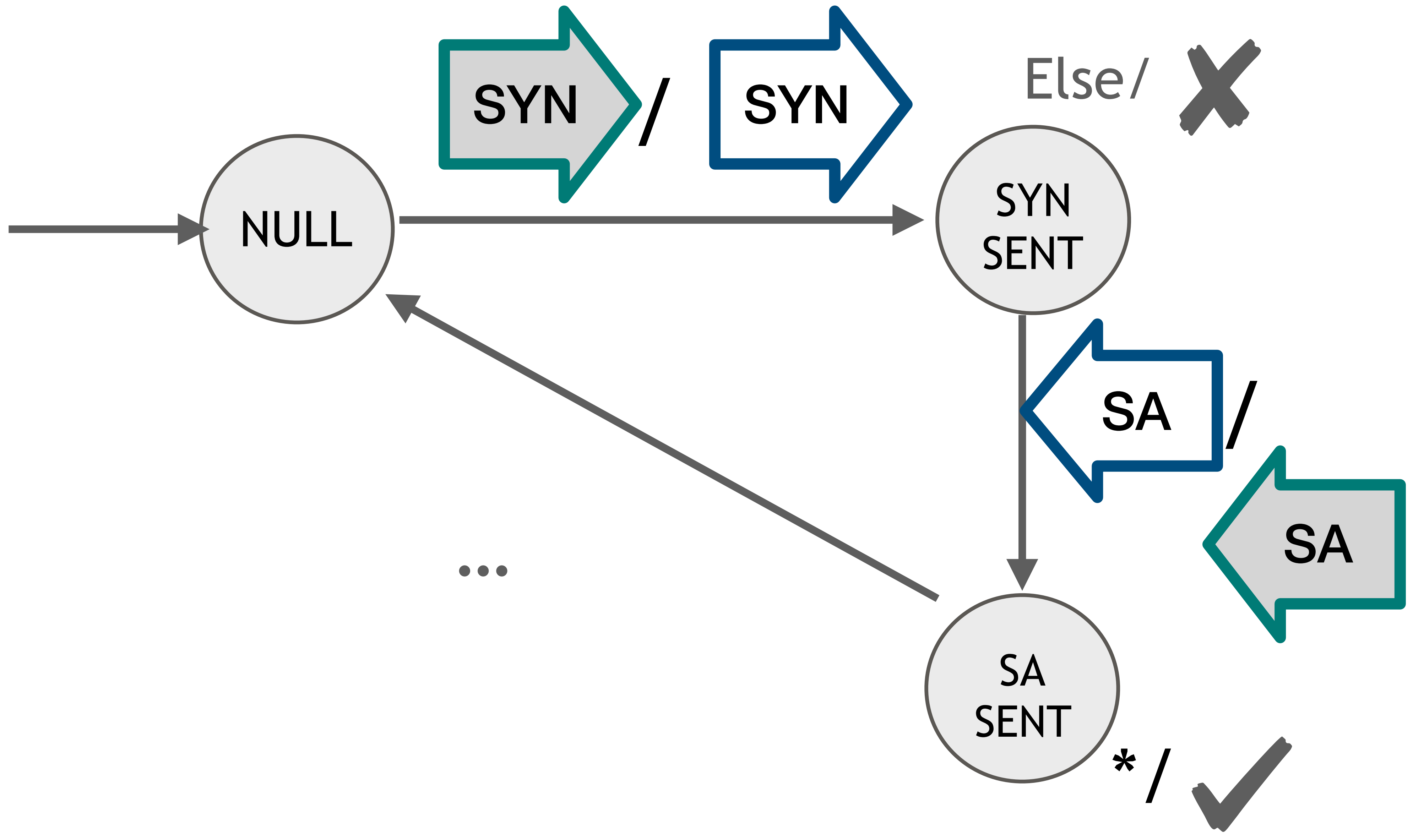
≠

Real FW implementation

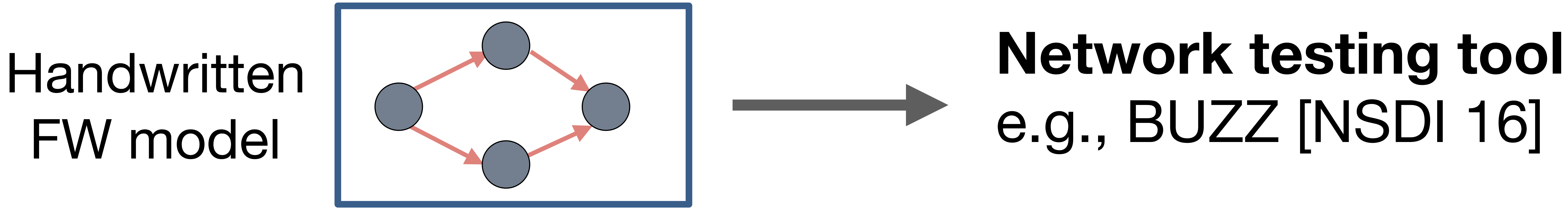


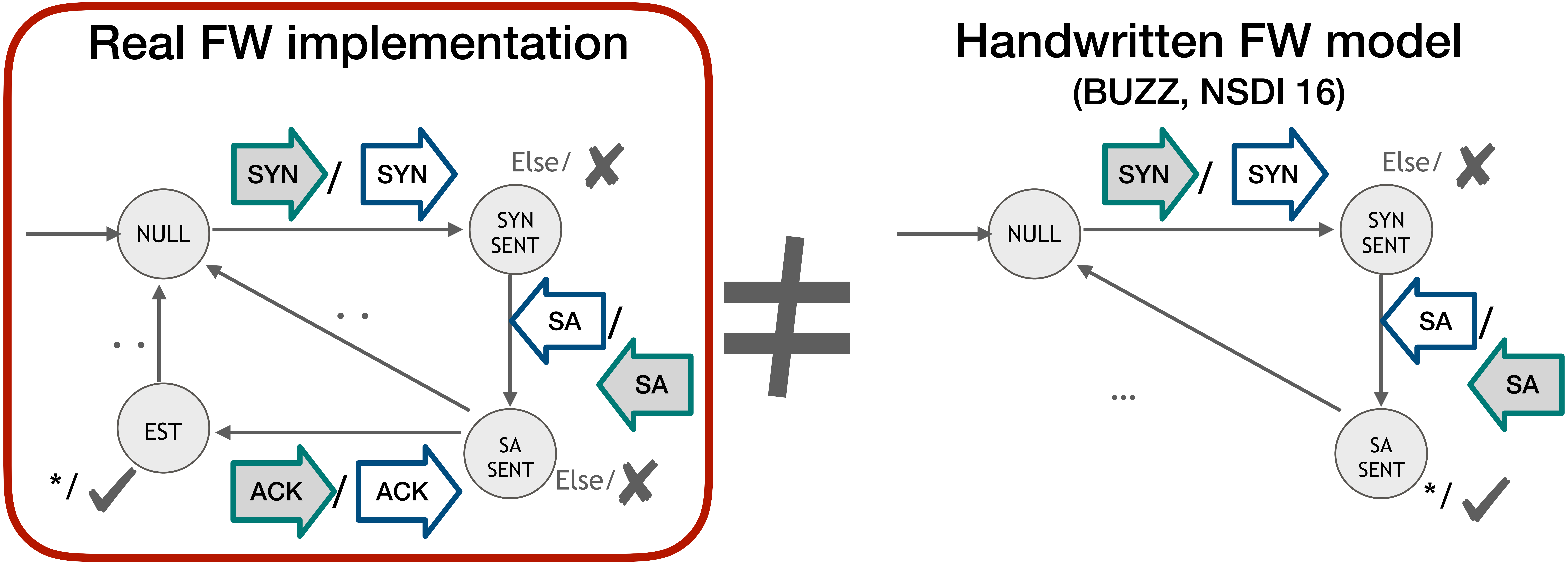
≠

Handwritten FW model
(BUZZ, NSDI 16)



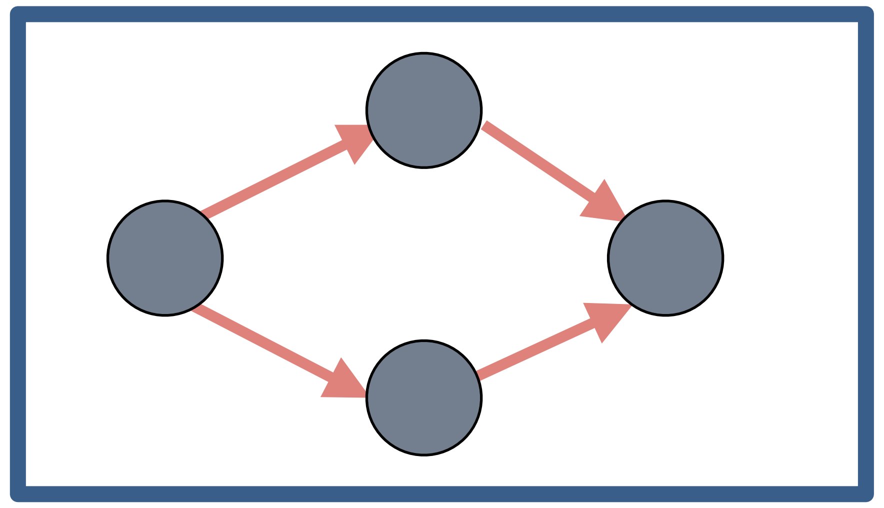
Limitation of Handwritten Model: Inaccuracy





Limitation of Handwritten Model: Inaccuracy

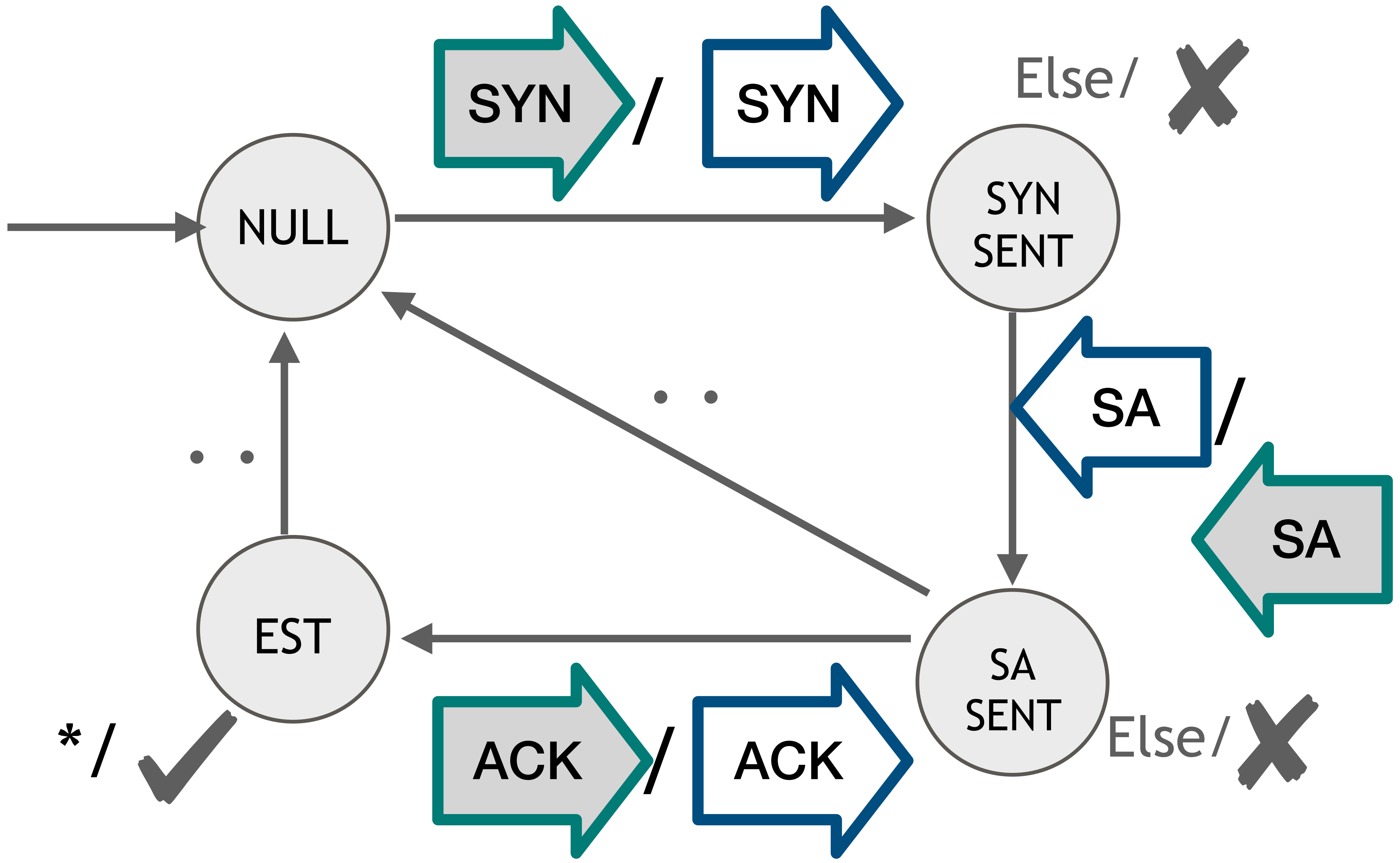
Handwritten FW model



Network testing tool

e.g., BUZZ [NSDI 16]

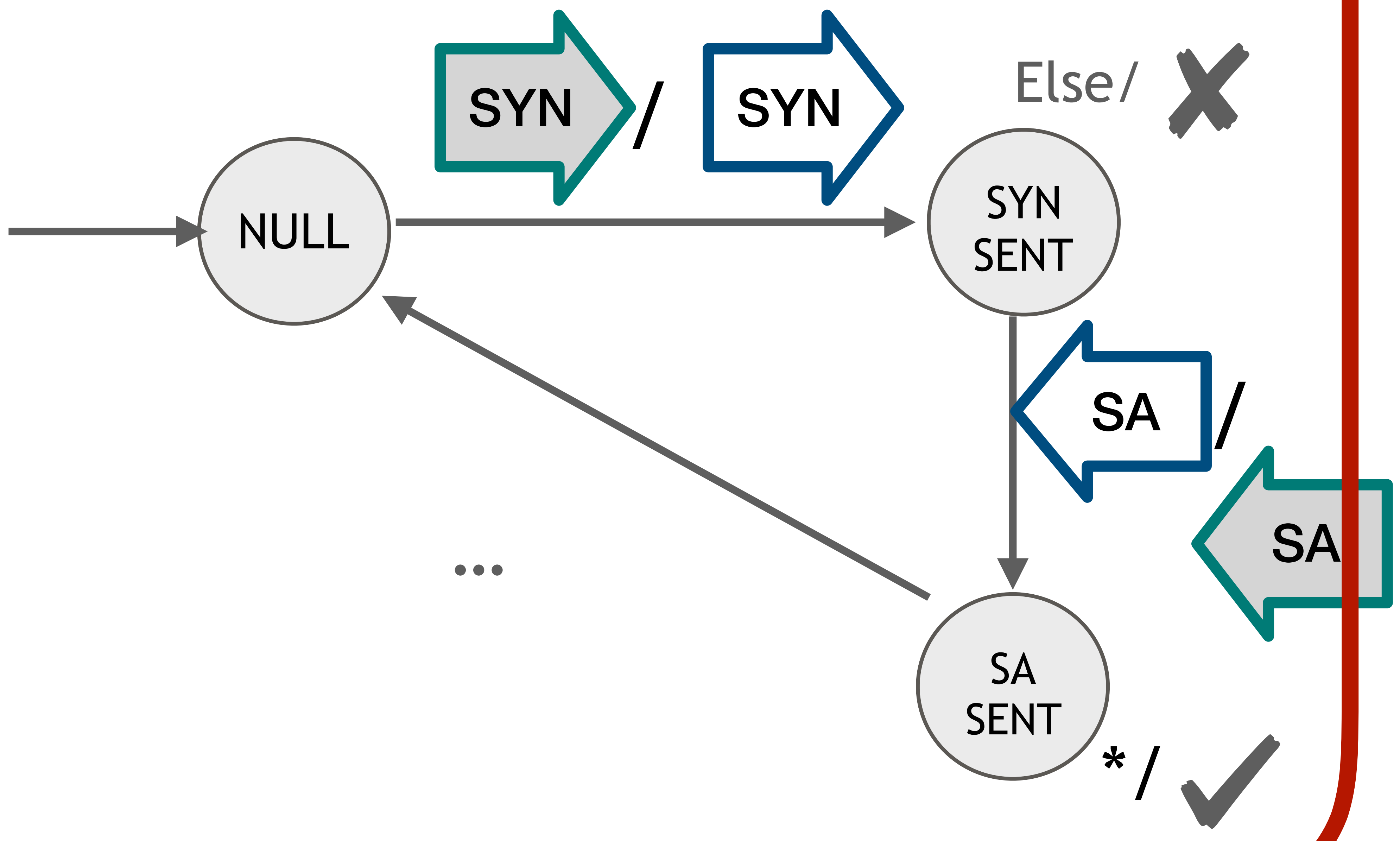
Real FW implementation



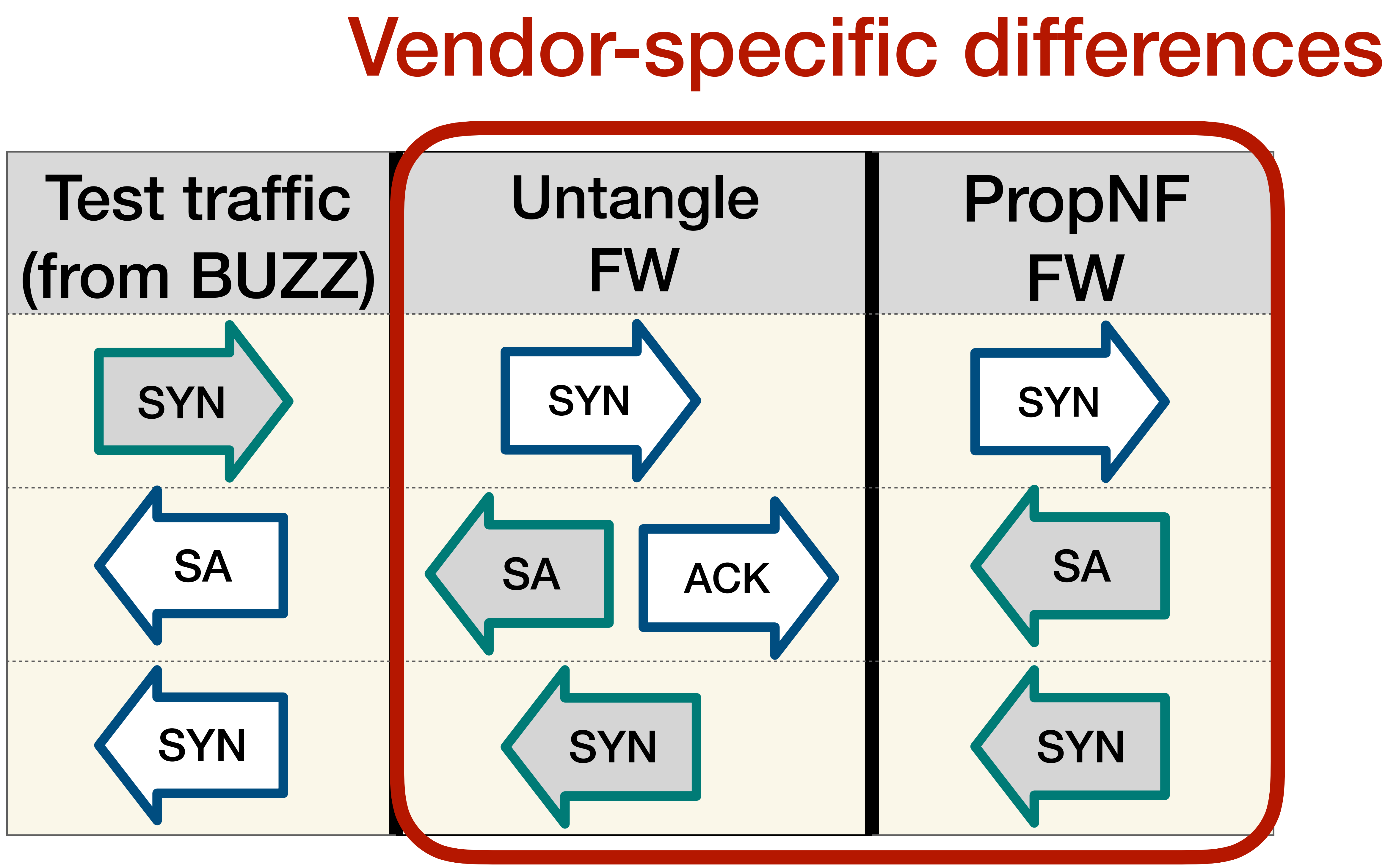
≠

Handwritten FW model

(BUZZ, NSDI 16)



Limitation of Handwritten Model: Vendor Diversity



Vendors have different implementations!

Our Work: Alembic

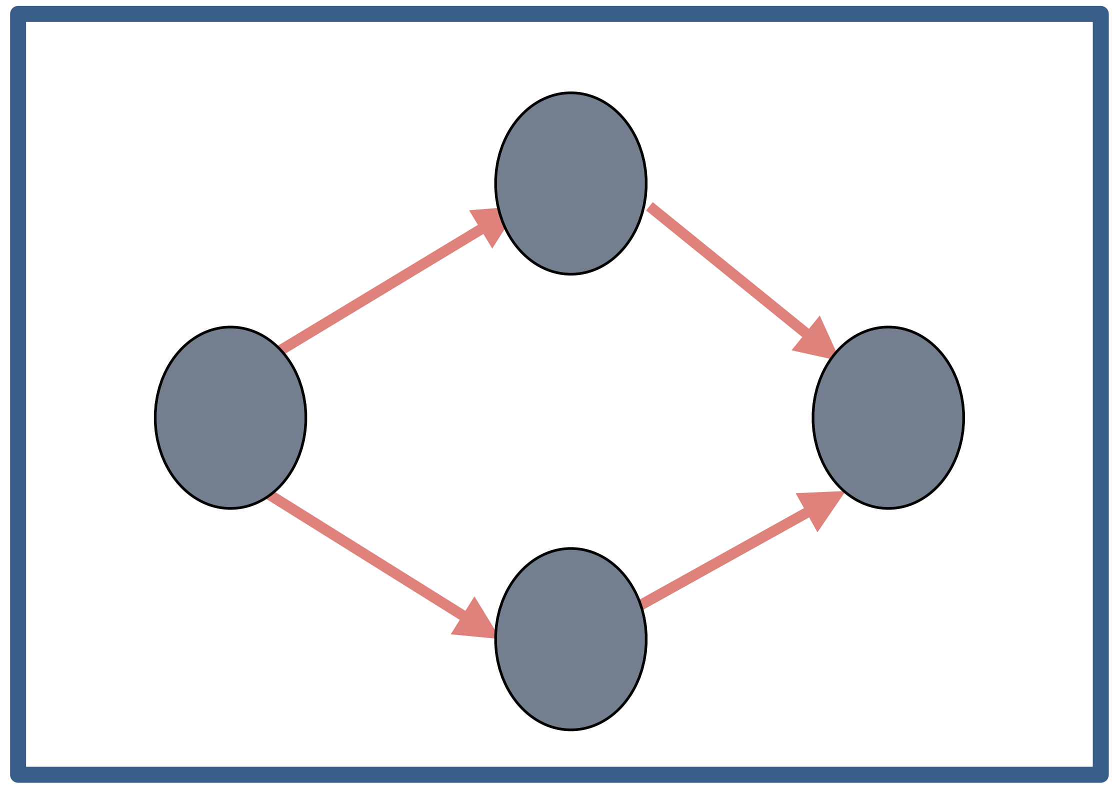
Automatically infer a behavioral model of the NF for a configuration

Config

Stateful NF

Model = NF(config)

Finite State Machine (FSM)



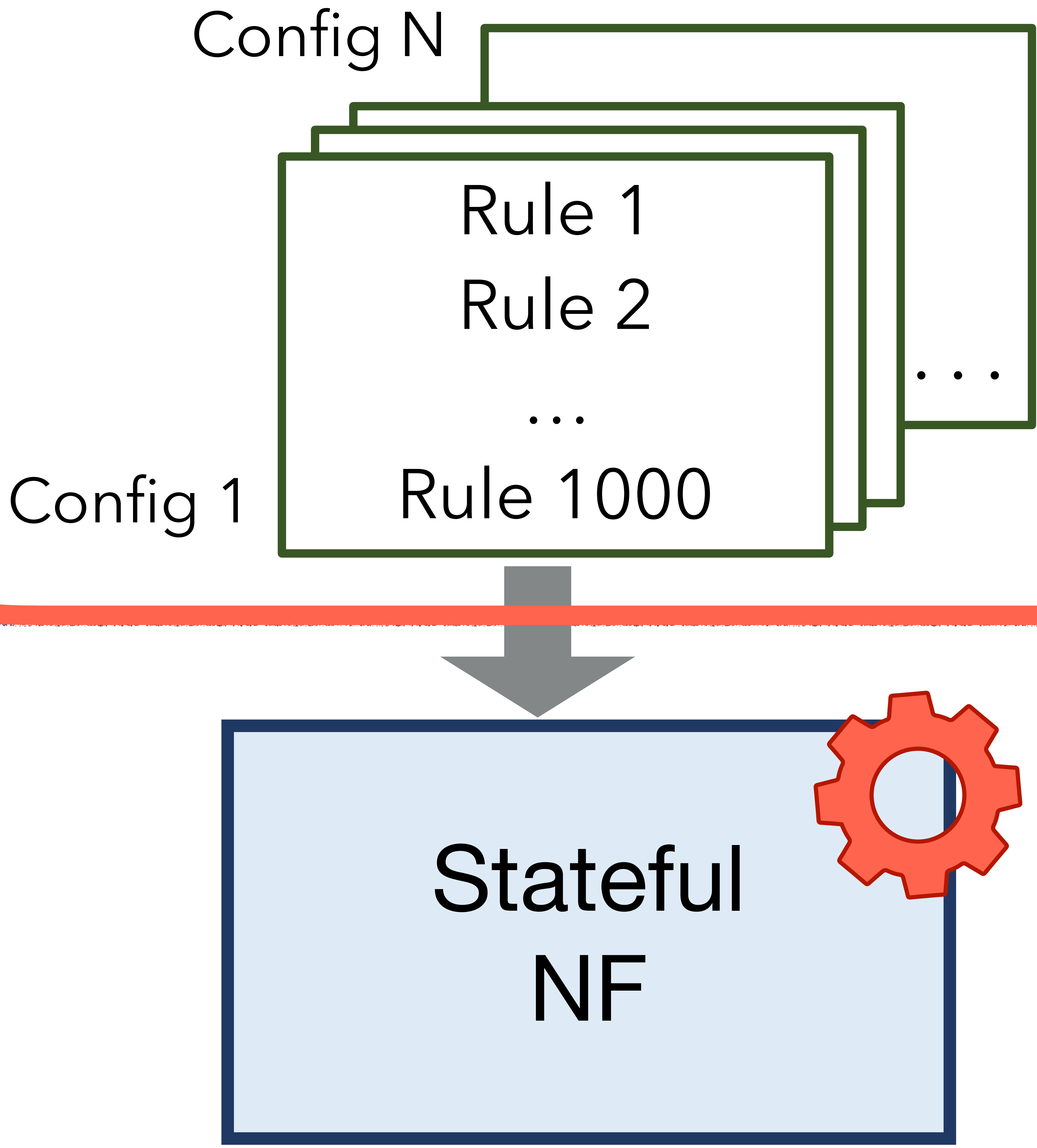
Customers:

1) BUZZ [NSDI16]
2) SYMNET [SIGCOMM16]
3) VMN [NSDI17]

Talk Outline

- Motivation and Goal
- **Challenges and Insights**
- Overall Workflow
- Evaluation

High-Level Challenges



Large configuration space

Inferring NF behavior

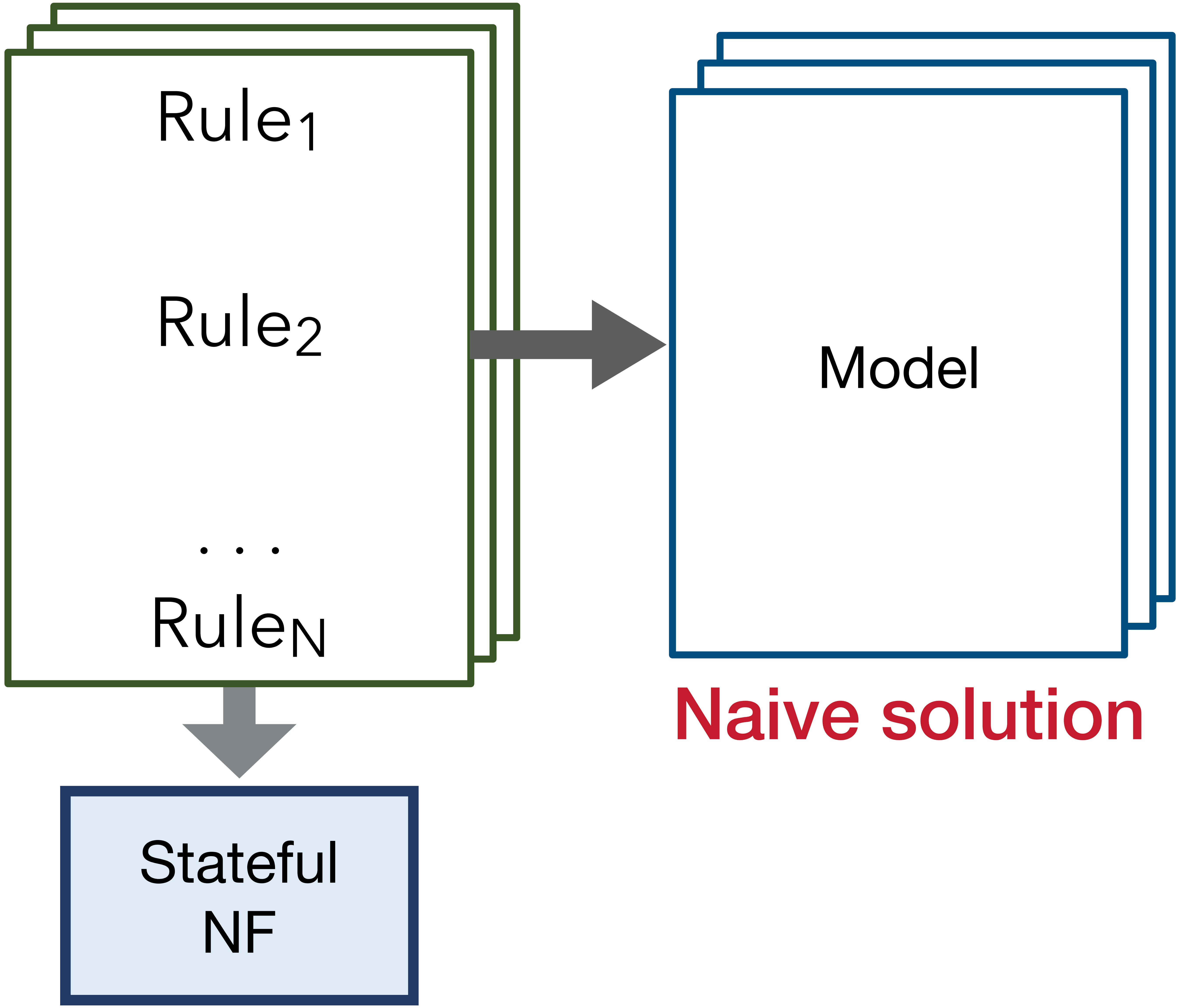
Challenges on Large Configuration Space

- Configuration → **many rules**

- Rule → IP/port fields take **large sets of values** (e.g., 2^{32} for IPs)

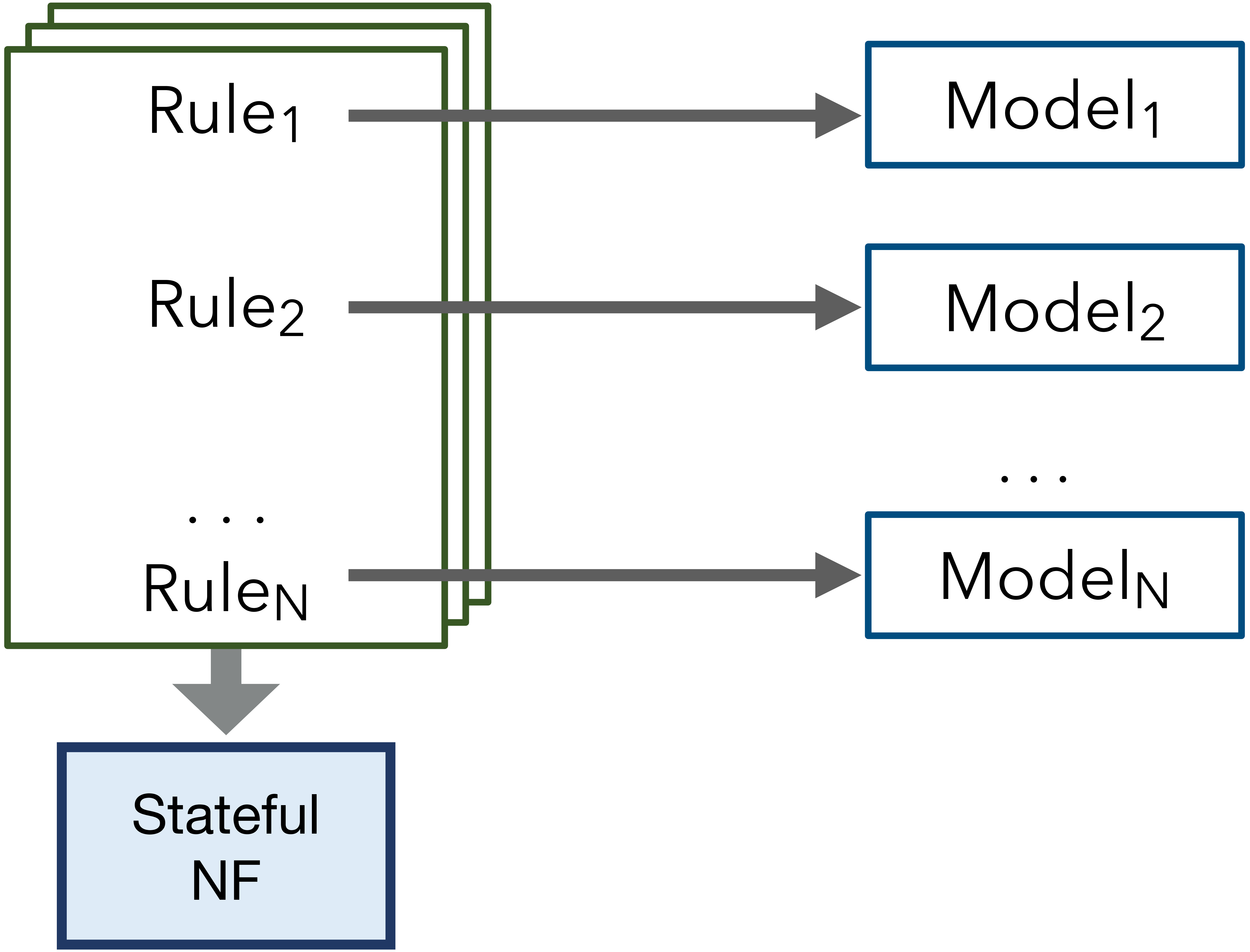
- Rule → IP/port fields can be **ranges** (e.g., /16 for IP prefixes)

Insight 1: We Can Compose Models of Individual Rules

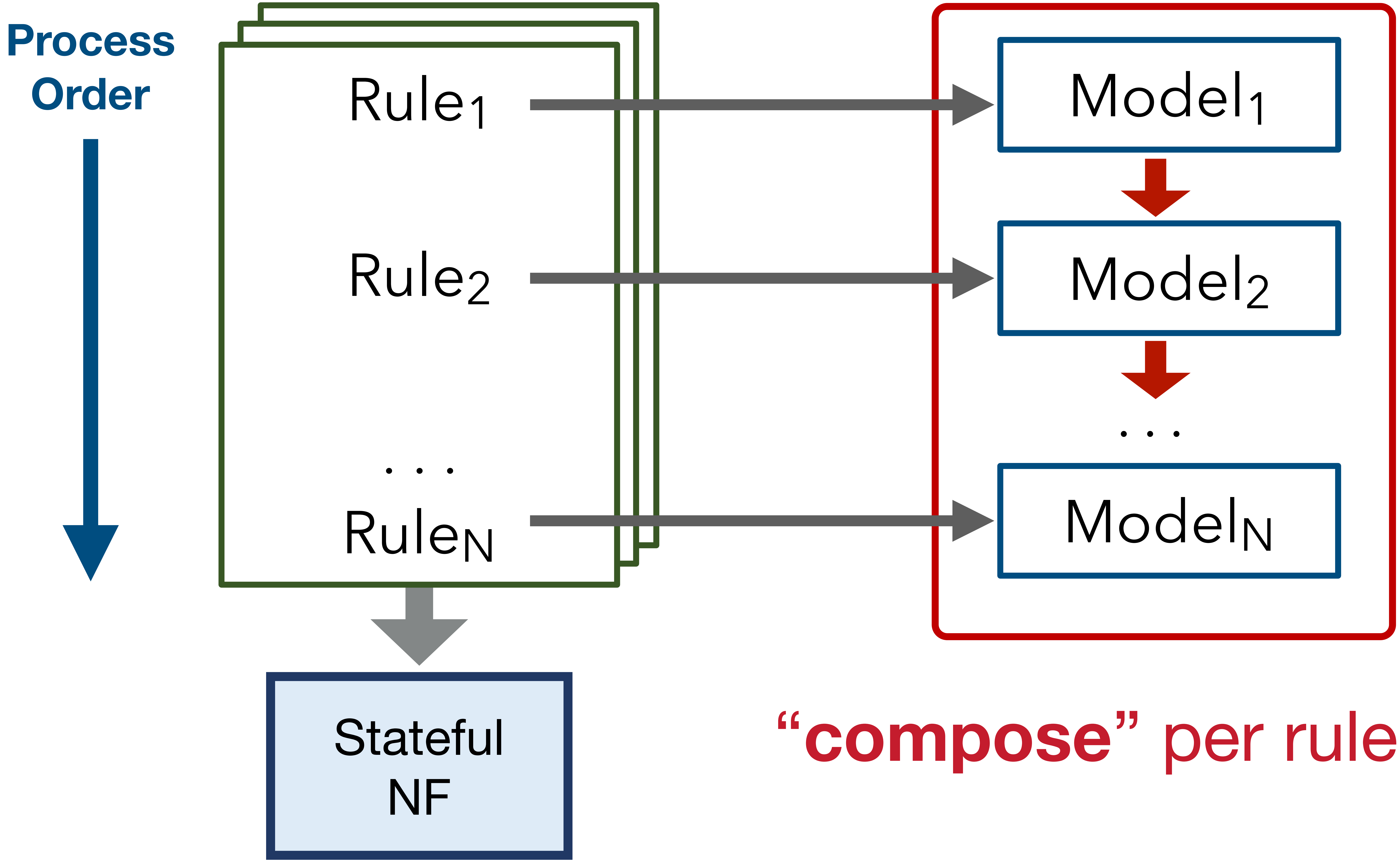


Naive solution

Insight 1: We Can Compose Models of Individual Rules



Insight 1: We Can Compose Models of Individual Rules



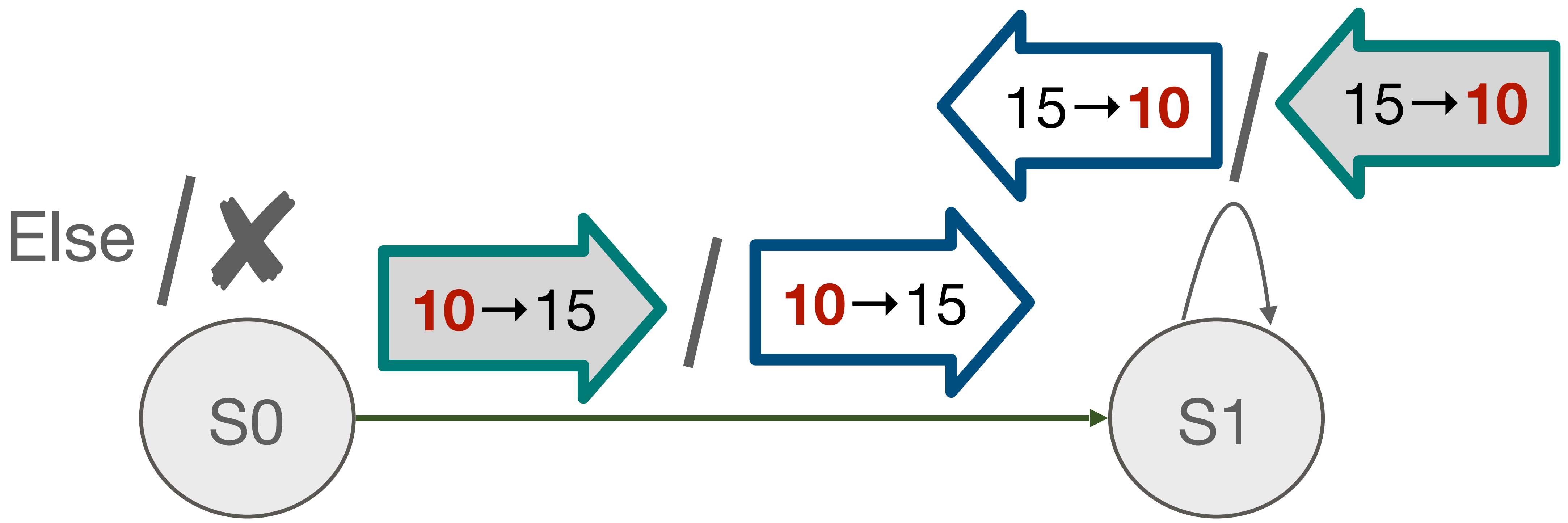
“**compose**” per rule models

Challenges on Large Configuration Space

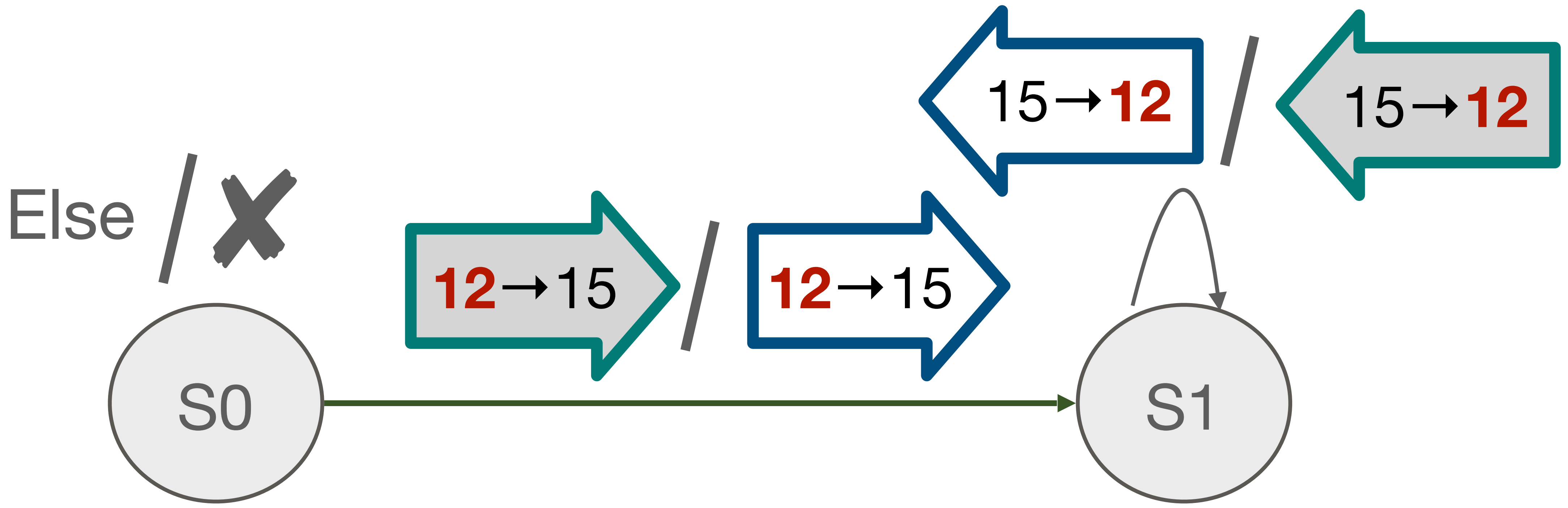
- Configuration → **many rules**
- Rule → IP/port fields take **large sets of values** (e.g., 2^{32} for IPs)
- Rule → IP/port fields can be **ranges** (e.g., /16 for IP prefixes)

Insight 2: Use Symbolic Models to represent Large Sets

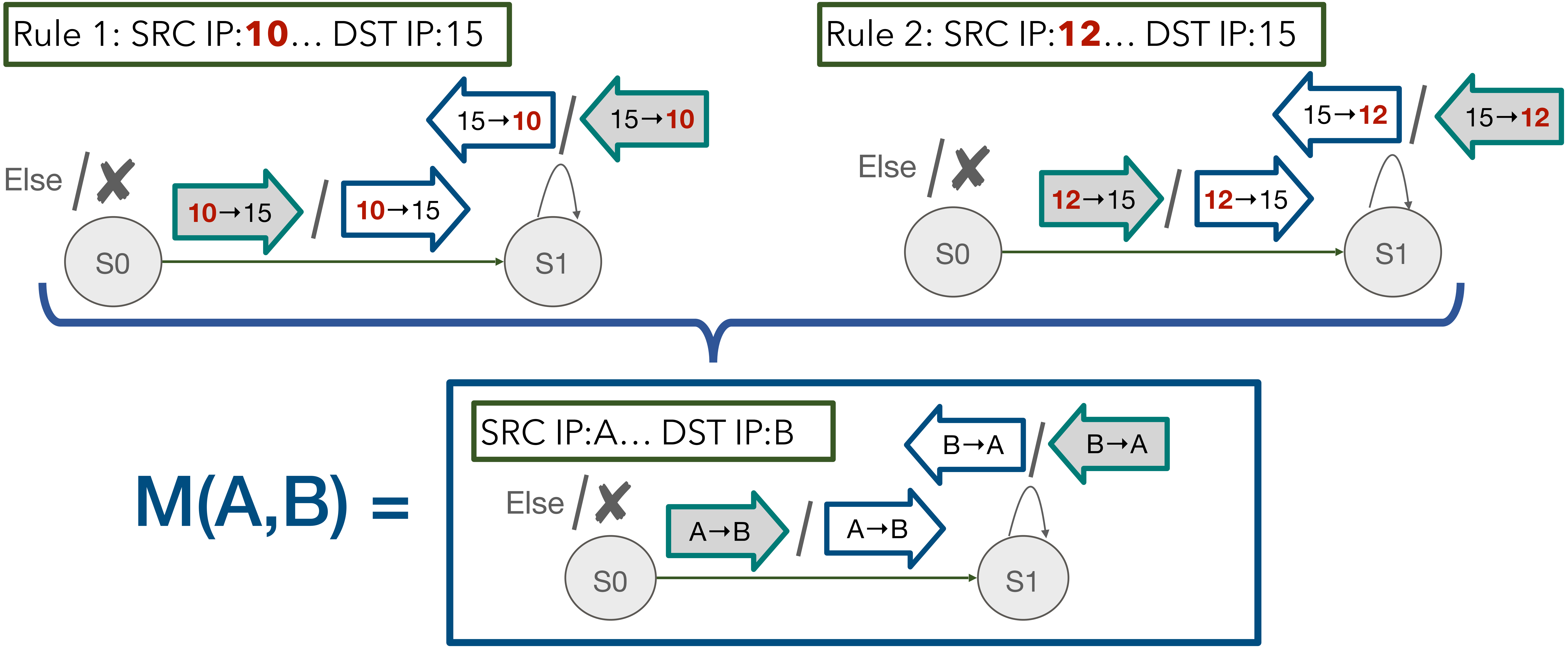
Rule 1: SRC IP:**10**… DST IP:15



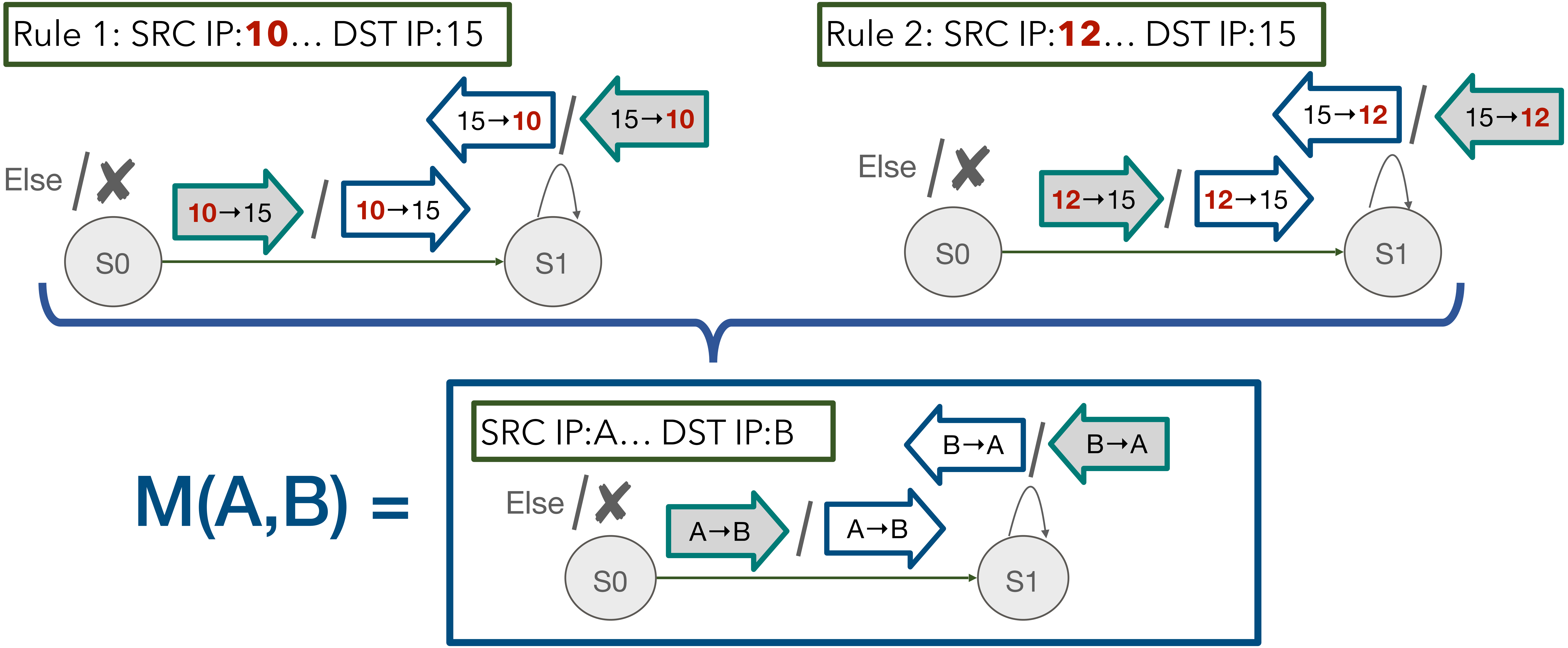
Rule 2: SRC IP:**12**… DST IP:15



Insight 2: Use Symbolic Models to represent Large Sets



Insight 2: Use Symbolic Models to represent Large Sets



If we get a new config: SRC IP:13… DST IP:16 M(A,B) where A = 13, B = 16

Challenges on Large Configuration Space

- Configuration → **many rules**

- Rule → IP/port fields take **large sets of values** (e.g., 2^{32} for IPs)

- Rule → IP/port fields can be **ranges** (e.g., /16 for IP prefixes)

Insight 3: Exploit Independence to Create an Ensemble of FSMs

SRC IP:10.1.1.0/**16**...DST IP:15.1.1.0/**16**

Insight 3: Exploit Independence to Create an Ensemble of FSMs

SRC IP:10.1.1.0/**16**…DST IP:15.1.1.0/**16** → Per-connection

Independent packet processing per **connection**

Conn 1 : 10.1.1.1 → 15.1.1.1
Conn 2 : 10.1.1.2 → 15.1.1.2

States do not interfere

Insight 3: Exploit Independence to Create an Ensemble of FSMs

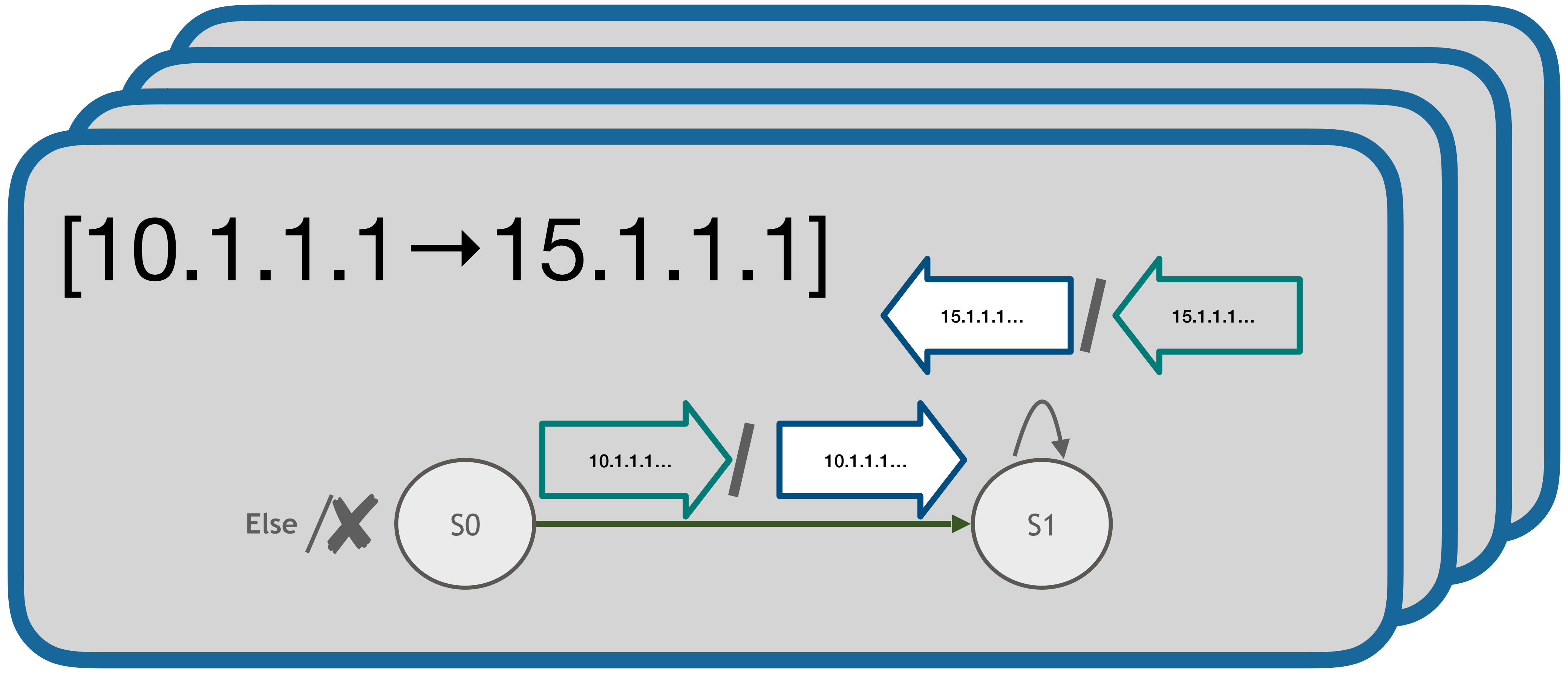
SRC IP:10.1.1.0/**16**…DST IP:15.1.1.0/**16** → Per-connection

Independent packet processing per **connection**

Learn M(A, B)

(symbolic model from insight 2)

Instantiate → **at runtime**



Ensemble of FSMs

An ensemble of concrete FSMs can represent a rule with IP/port ranges

Summary of Insights to Address Large Configuration Space

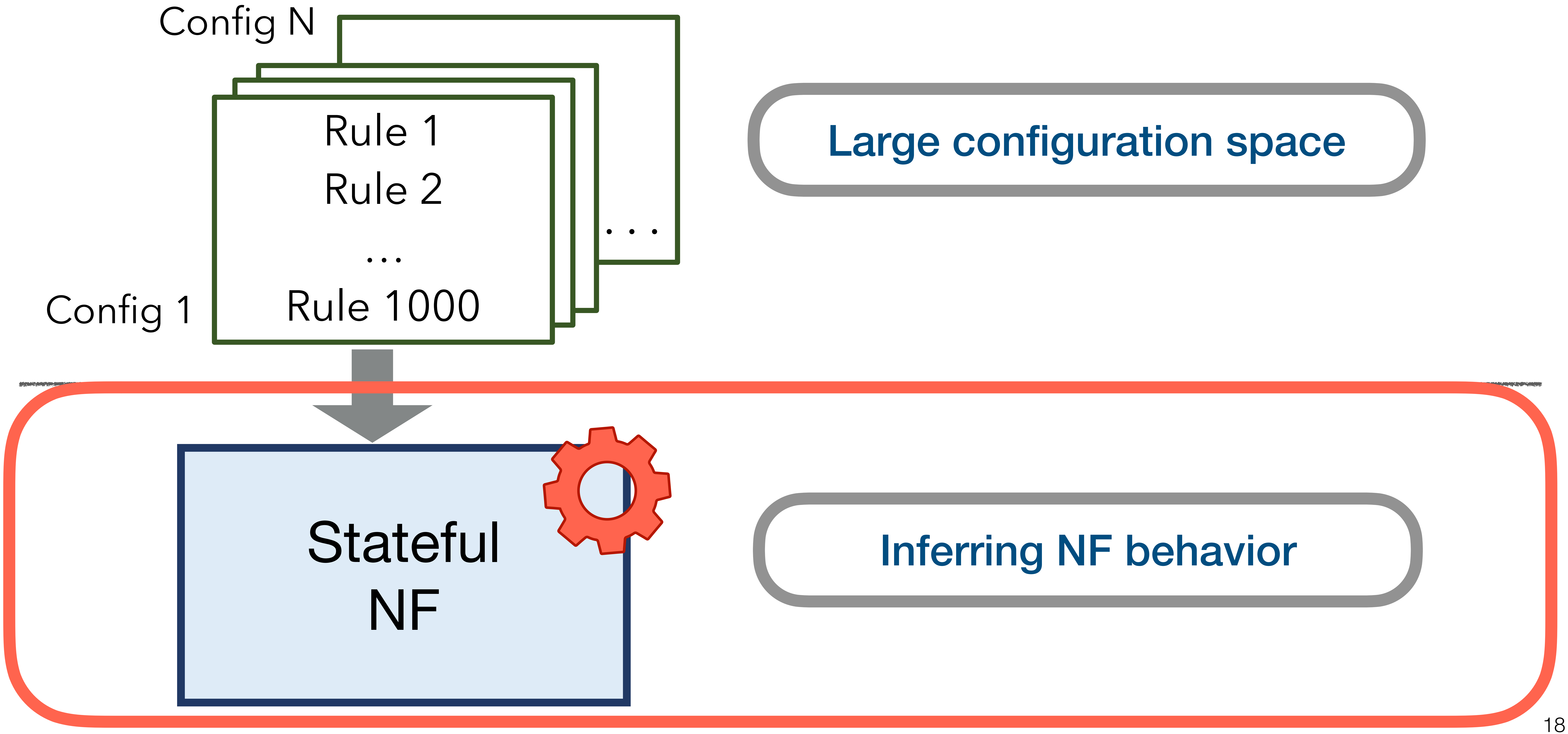
A configuration is composed of **many number of rules**

Compositional Model

A rule contains **IP/port fields** which take **large sets of values** and **ranges**.

Symbolic Model $\xrightarrow{\textbf{Instantiation}}$ An Ensemble of FSMs

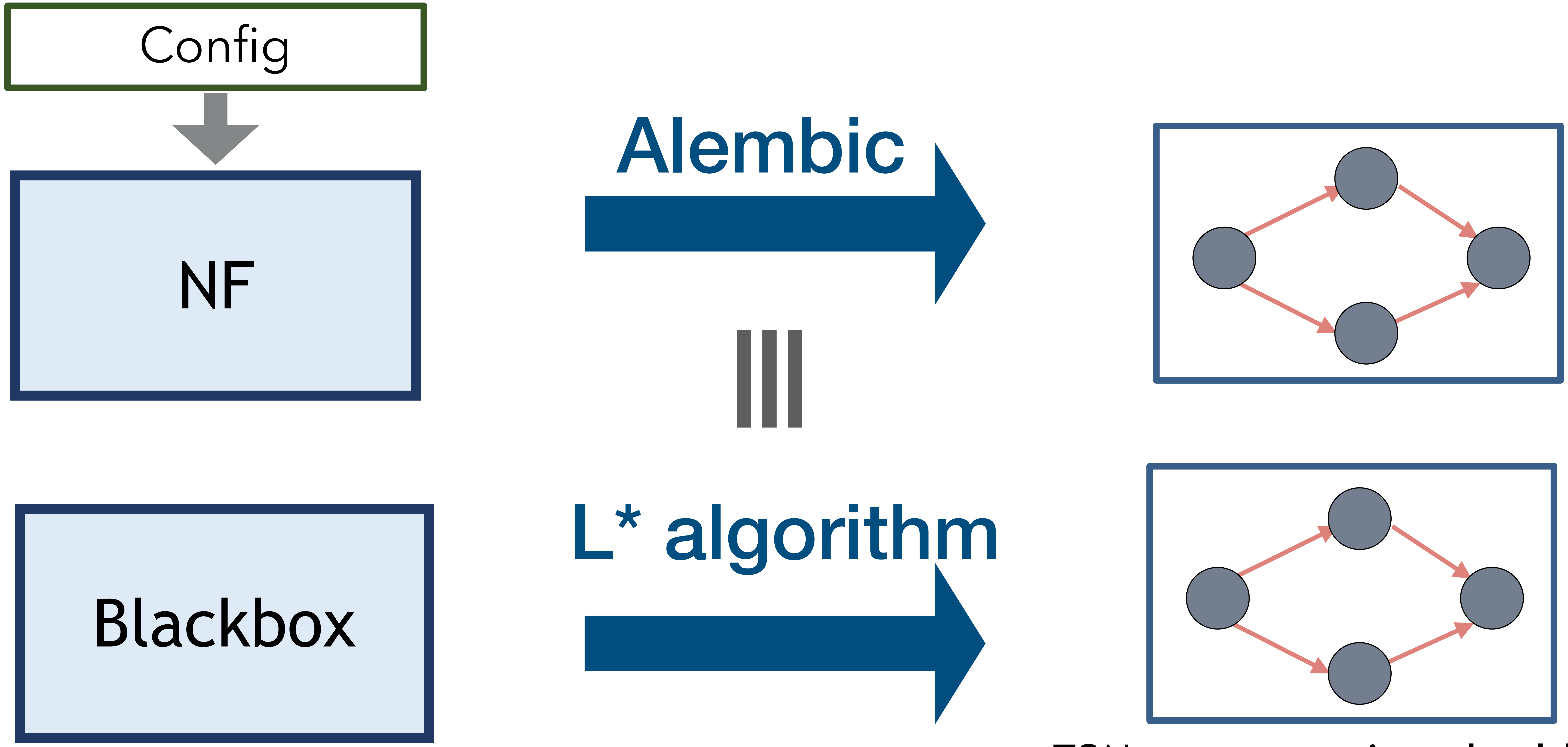
Back to High-Level Challenges



Challenges on Inferring NF Behavior

- Inferring the **symbolic FSM**
- Inferring the **state granularity**
- Handling **dynamic header modification**

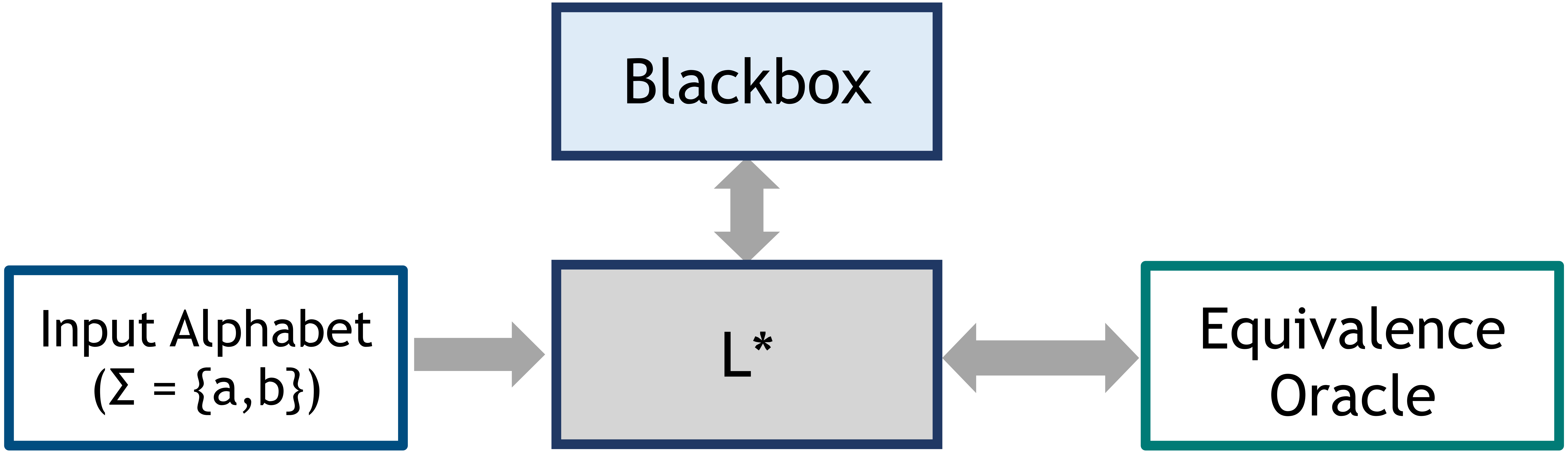
Insight: Leverage L* Algorithm to Infer a Symbolic FSM



FSM representing the blackbox

We can use the L* algorithm!

Background on L* for Black-box FSM Inference

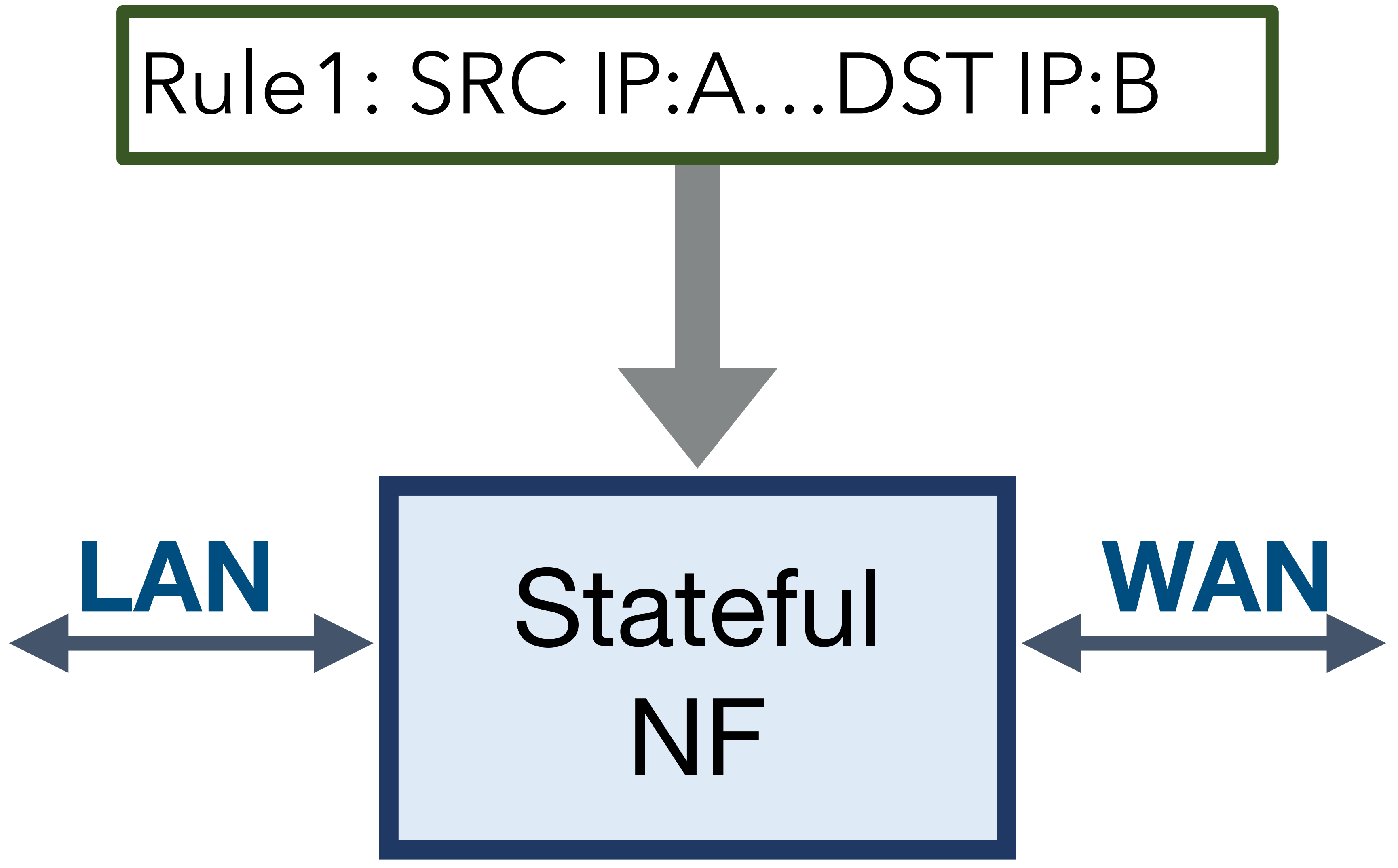


- Generates sequences (e.g., aa, aba) and probes the blackbox
- Builds a hypothesis FSM with input-output pairs seen so far
- Queries an Equivalence Oracle (EO) for counterexamples

Practical Challenges of Applying L* for an NF

- Generate **input alphabet**
- Classify **output of an NF**
- Build an **Equivalence Oracle**

Generating Input Alphabet to handle Large Traffic Space



Naive solutions:

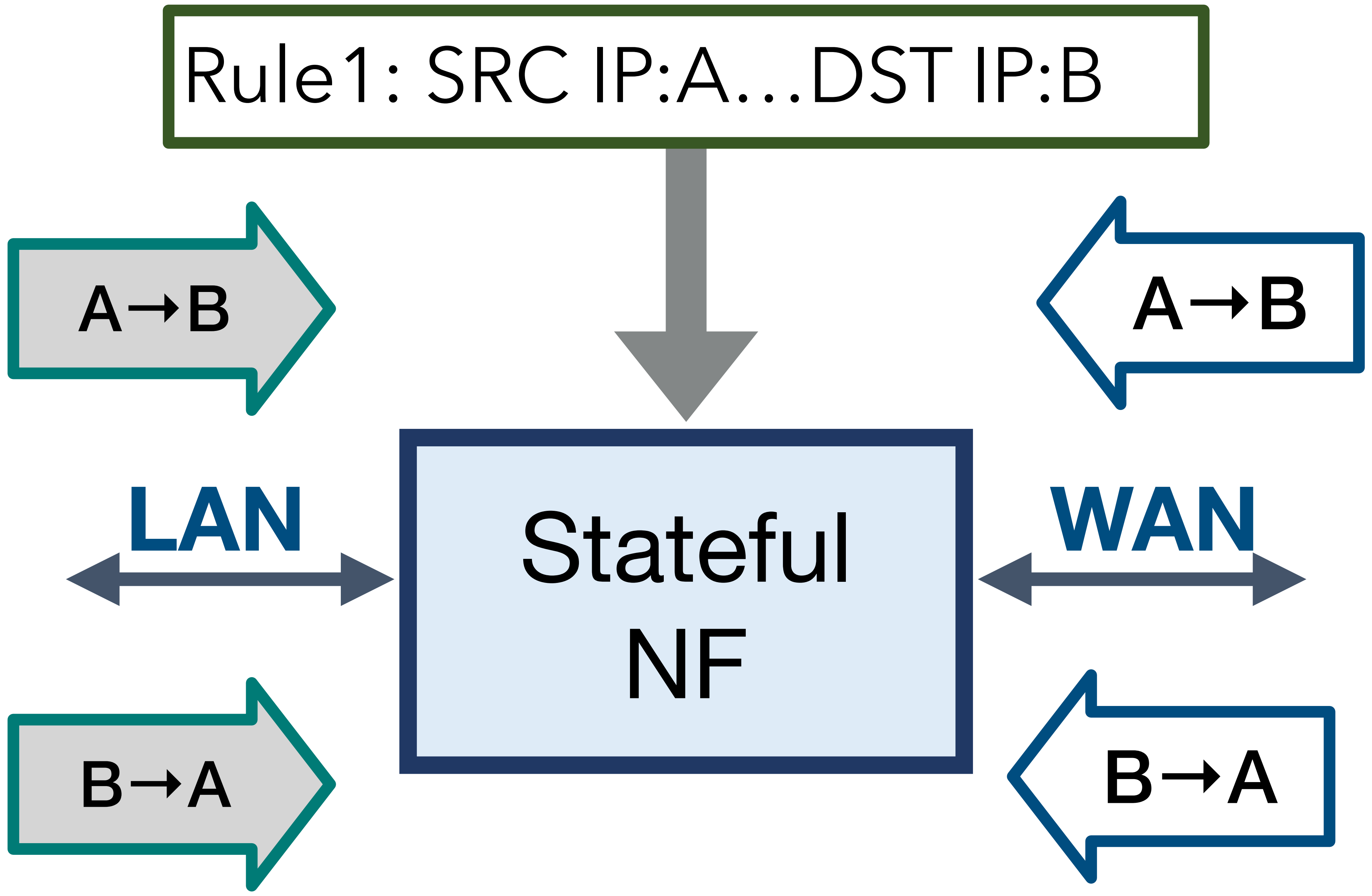
~~1. Exhaustively generating packets~~

Infeasible

~~2. Randomly generating packets~~

Does not explore the relevant state space

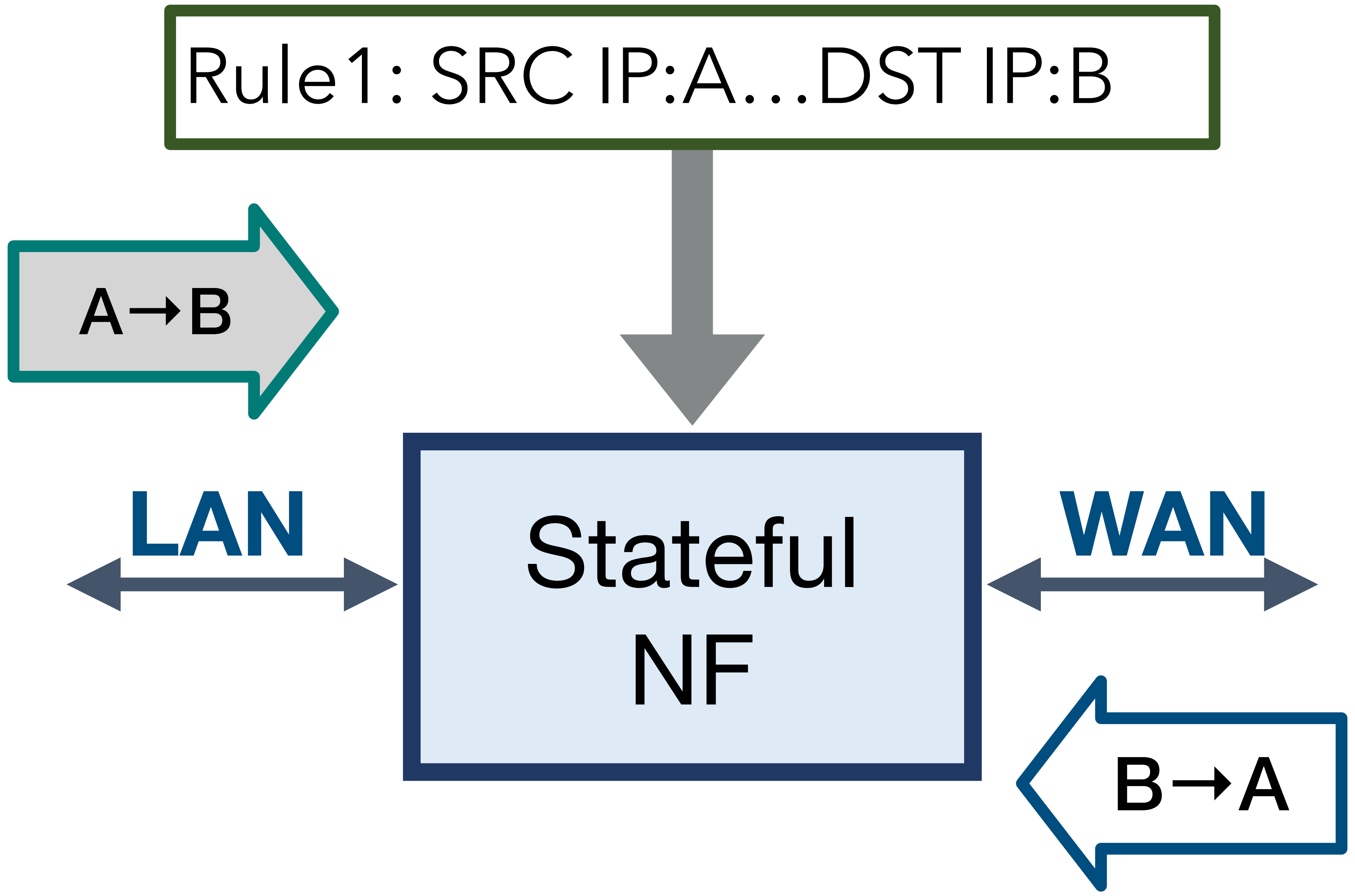
Generating Input Alphabet to handle Large Traffic Space



1) **Find IP/port fields that appear in the rule**
   Generate the packet for for all interfaces using A and B

To exercise the rule, we generate packets with IP/ports in the rule

Generating Input Alphabet to handle Large Traffic Space

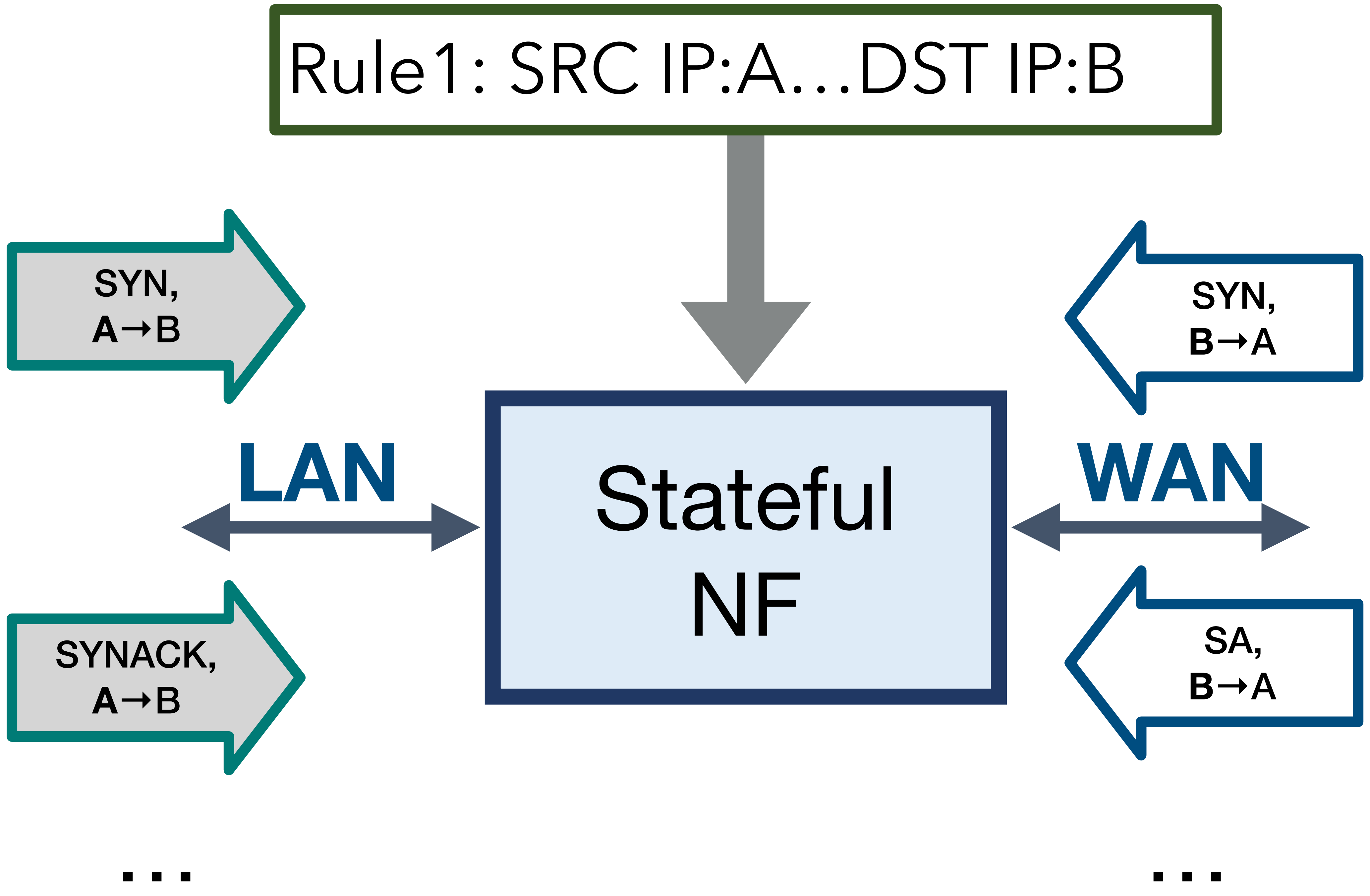


1) **Find IP/port fields that appear in the rule**
   Generate the packet for for all interfaces using A and B

2) **(Optional) Prune based on reachability**

To exercise the rule, we generate packets with IP/ports in the rule

Generating Input Alphabet to handle Large Traffic Space



1) **Find IP/port fields that appear in the rule**
   Generate the packet for for all interfaces using A and B

2) **(Optional) Prune based on reachability**

3) **Plug in “packet types”**

To exercise the rule, we generate packets with IP/ports in the rule

Practical Challenges of Applying L* for an NF

- Generate **input alphabet**
- Classify **output of an NF**
  - Configure the “timeout” to classify output
  - Translating to/from symbolic and concrete packets
- Build an **Equivalence Oracle**

Challenges on Inferring NF Behavior

- Inferring the **symbolic model (FSM)**
- Inferring the **state granularity**
- Handling **dynamic header modification**

Different Types of State Granularity

State Granularity: the state variables (IP/ports) that the NF uses to keep state

Cross-connection → One FSM for all connections

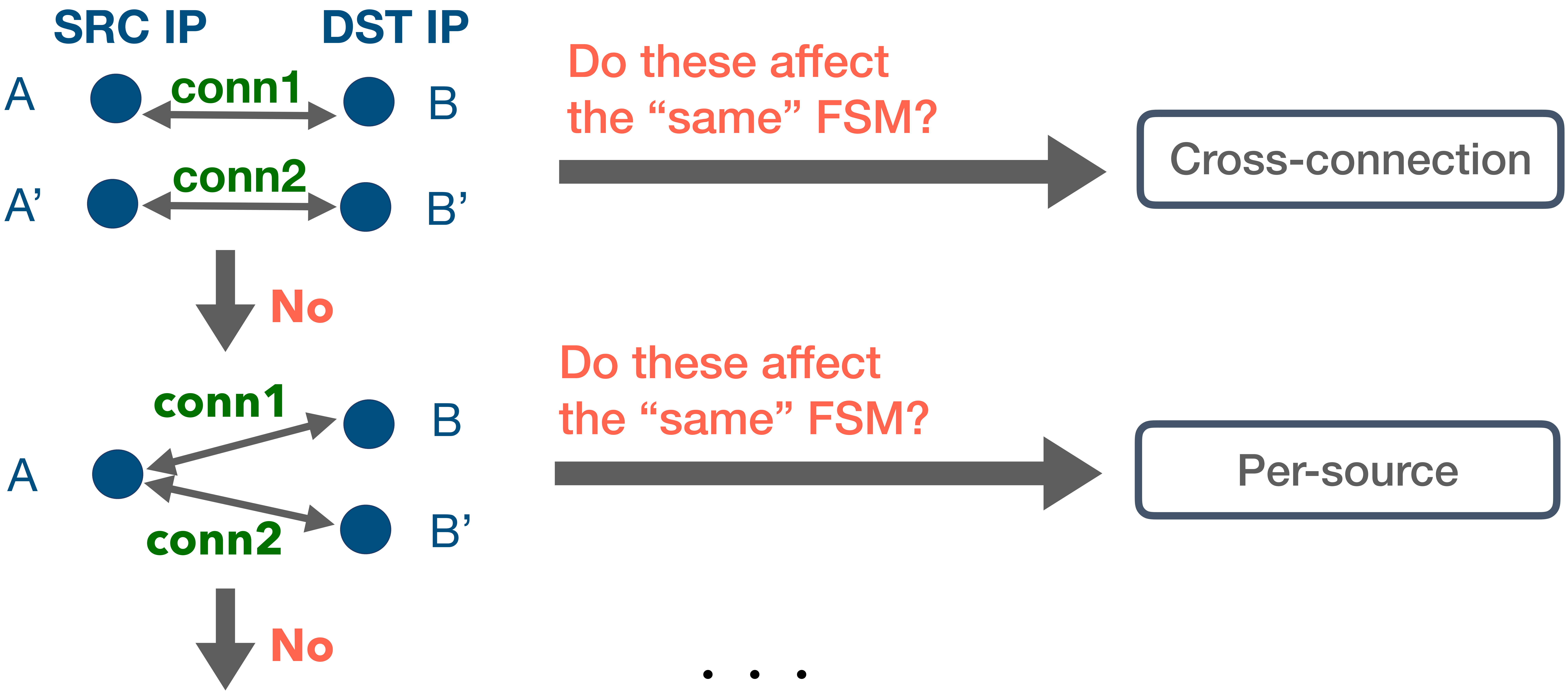
Per-source → One FSM for each srcip

Per-destination → One FSM for each dstip

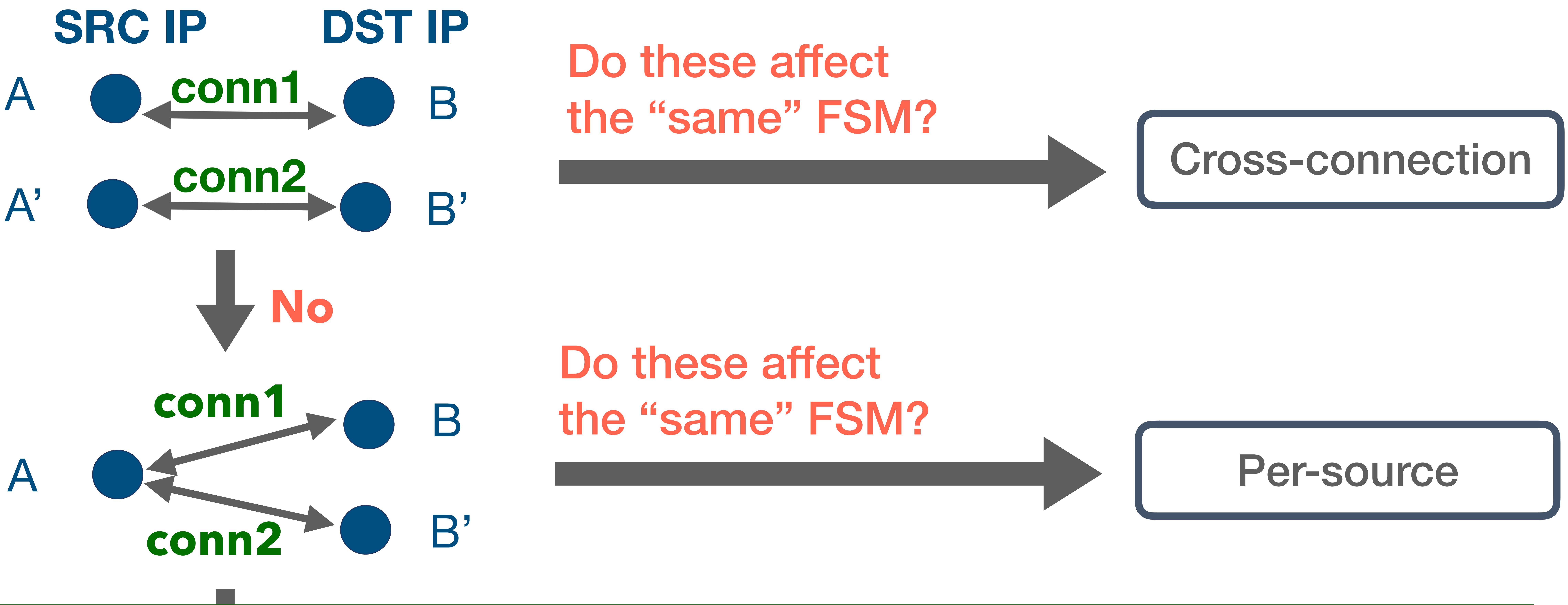
Per-connection → One FSM for every IP-port pair

This is like a “key” mapping to the FSM

Learning the State Granularity



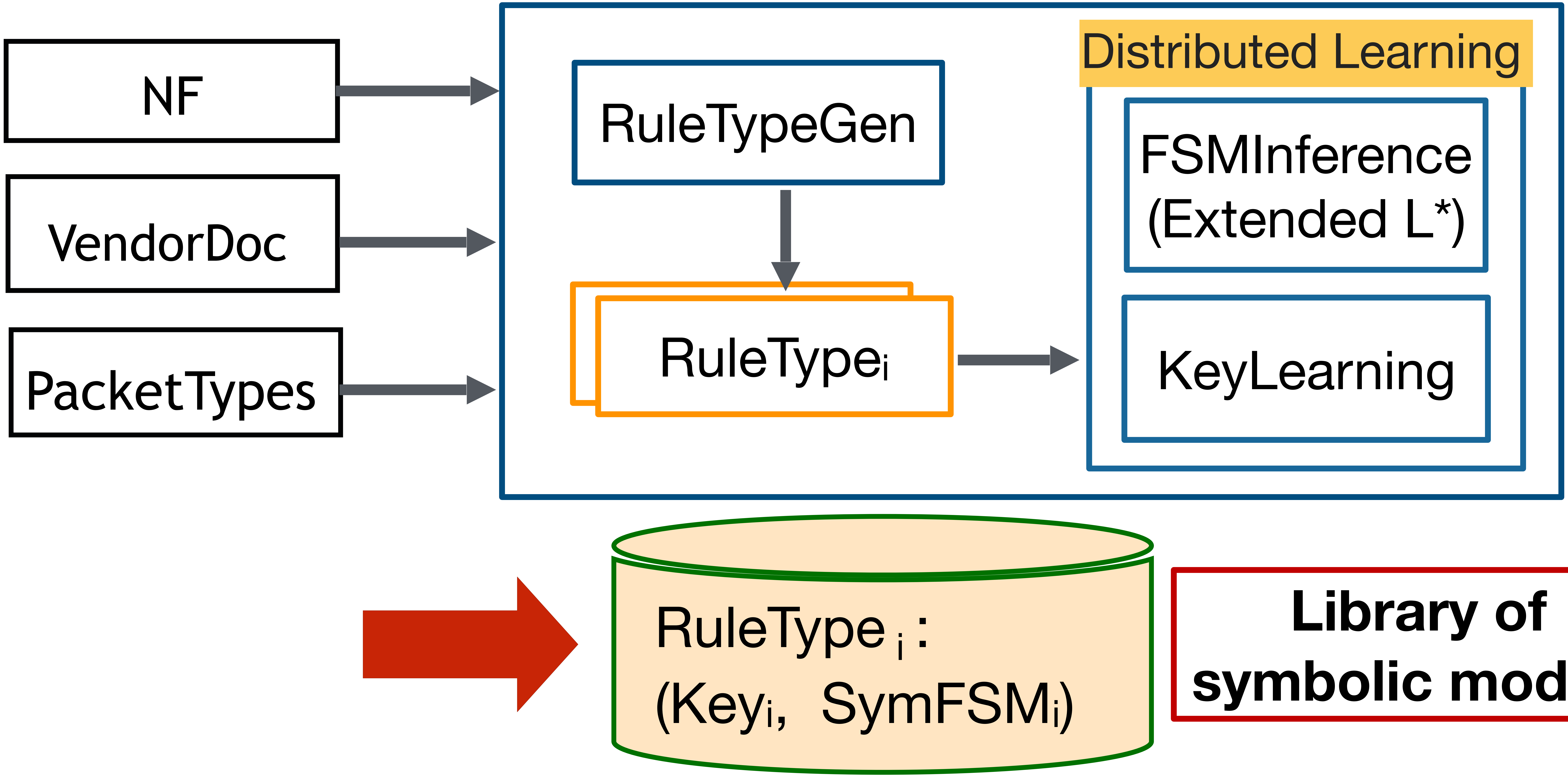
Learning the State Granularity



Construct test cases for **independence across connections**

Alembic Workflow: Offline

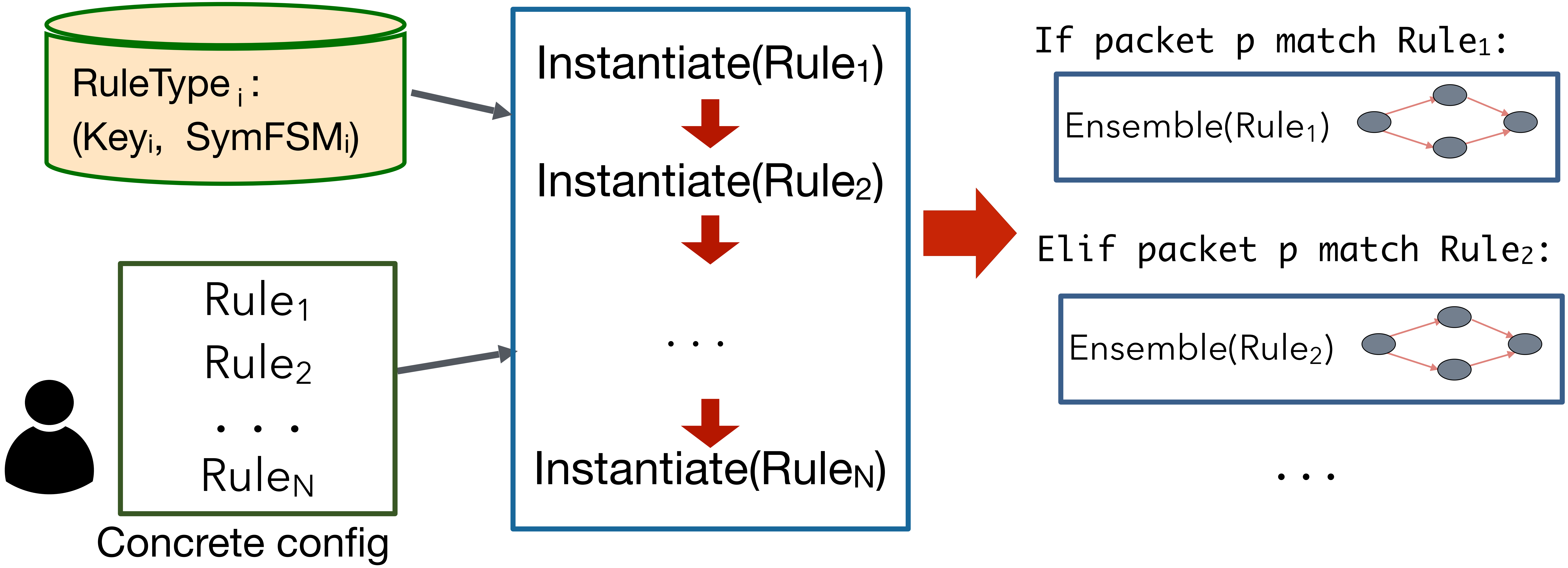
Runs once per NF



Library of symbolic models

Alembic Workflow: Online

Runs for every config



Evaluation Summary

- Alembic-generated models are **accurate**
- Case Studies: Alembic finds **differences across NF implementations**
- Alembic workflow is **scalable**
- Alembic-generated models **improve the accuracy of network testing/verification tools**

Evaluation Setup

- Validated Alembic using Click-based NFs where we know the ground truth
- **Real NFs** we modeled :
  - PfSense (FW, static NAT, random NAT, LB)
  - Proprietary NF (FW, static NAT)
  - Untangle (FW)
  - HAproxy (LB)

- **Packet types** used:
  - Correct-Seq: {SYN_C, $SYN\text{-}ACK_C$, ACK_C, $FIN\text{-}ACK_C$, $RST\text{-}ACK_C$}
  - Combined-Seq: extend the correct-seq set with incorrect seq and ack, {$SYN\text{-}ACK_I$, ACK_I, $FIN\text{-}ACK_I$, $RST\text{-}ACK_I$}

Accuracy Evaluation

Since we do not have the ground-truth, we designed **complementary testing methodology** to test the accuracy of our models

- Config generation: 1 to 100 rules in a configuration
- Packet generation: 20 to 300 packets in a sequence

1) **Iperf testing:** 100% across all settings for all NFs
2) **Random Packet testing (randomly choosing IP/port):**
   99.8% to 100% across all settings for all NFs
3) **Rule Activation testing (choosing IP/port to activate one rule):**
   94.8% to 100% across all settings for all NFs

Evaluation Summary

- Alembic-generated models are **accurate**
- Case Studies: Alembic finds **differences across NF implementations**
- Alembic workflow is **scalable**
- Alembic-generated models **improve the accuracy of network testing/verification tools**

Firewall Case Study

	PfSense	ProprietaryNF
Packet sequence before the FW allows TCP traffic from an external host (B) to an internal host (A)	SYN, $A \rightarrow B$	SYN, $A \rightarrow B$ / SA, $B \rightarrow A$

Firewall Case Study

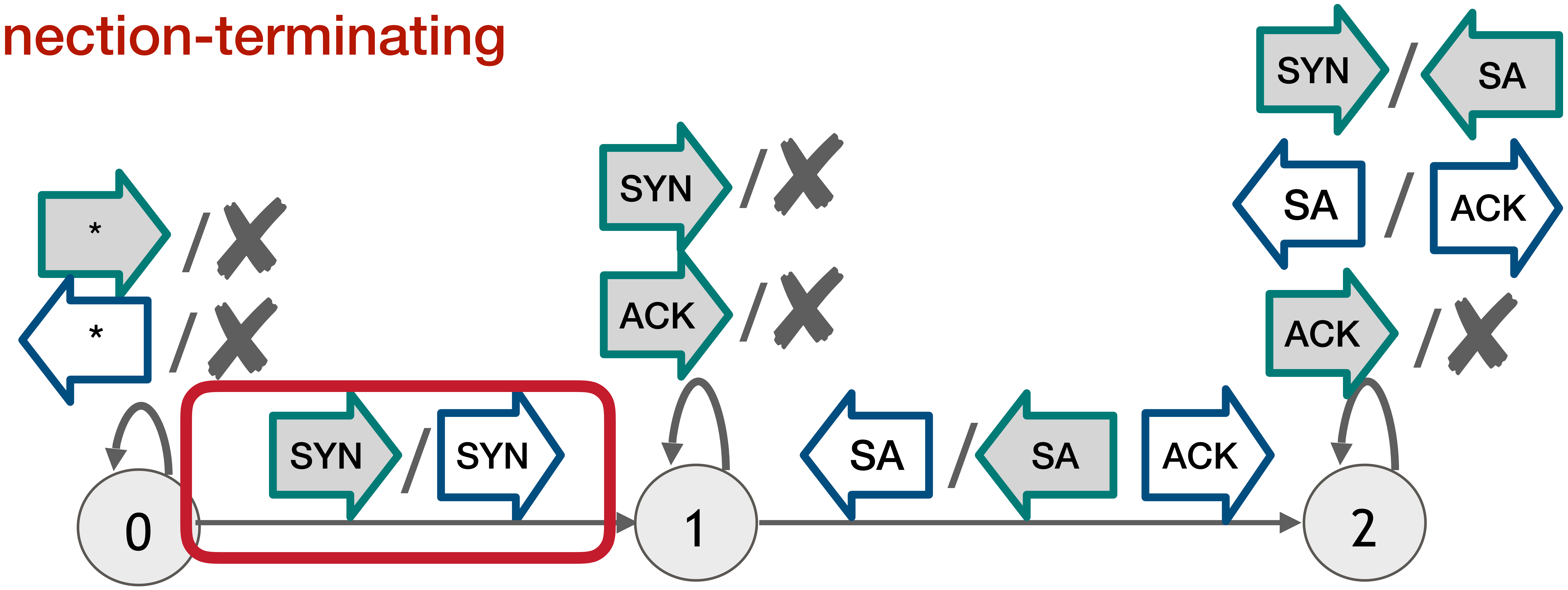
	PfSense	ProprietaryNF
Packet sequence before the FW allows TCP traffic from an external host (B) to an internal host (A)	SYN, A→B	SYN, A→B; SA, B→A
Number of states	3	79

Firewall Case Study

	PfSense	ProprietaryNF
Packet sequence before the FW allows TCP traffic from an external host (B) to an internal host (A)	SYN, A→B	SYN, A→B SA, B→A
Number of states	3	79
Default behavior	Default Drop	Default Drop

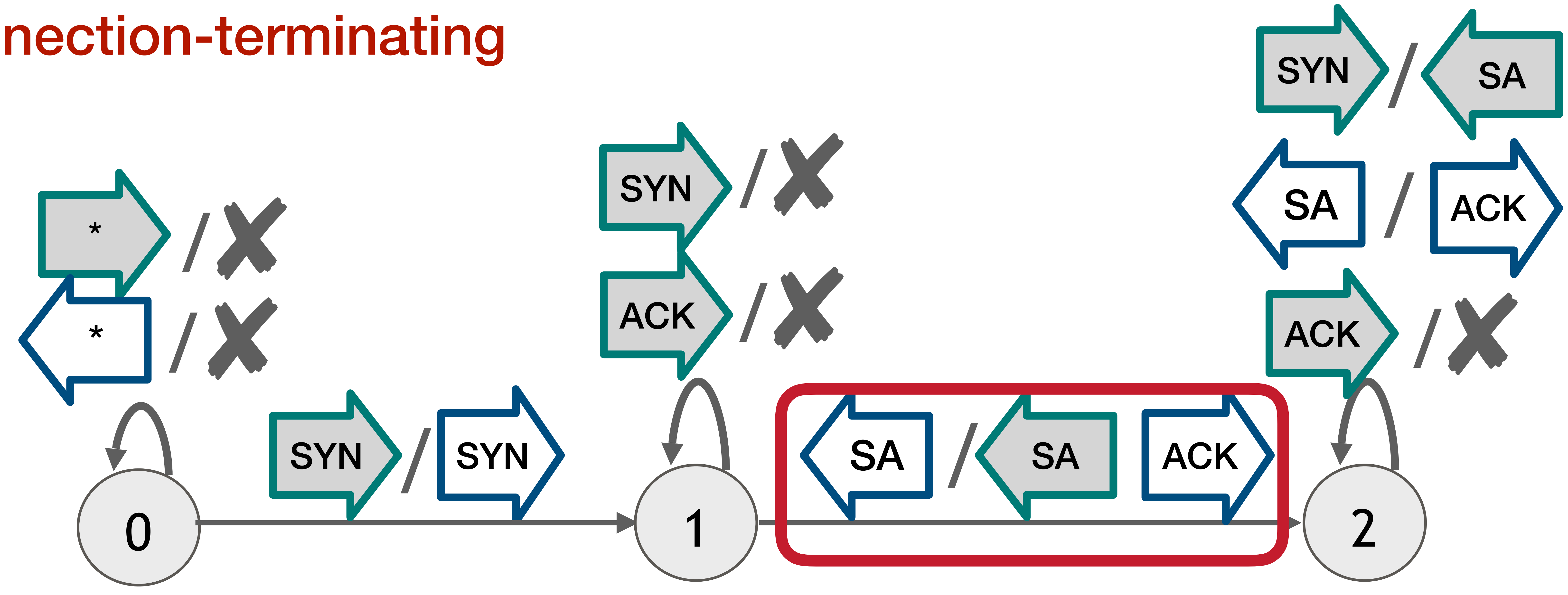
Firewall Case Study: Untangle Firewall

- Implements “default allow”
- Connection-terminating



Firewall Case Study: Untangle Firewall

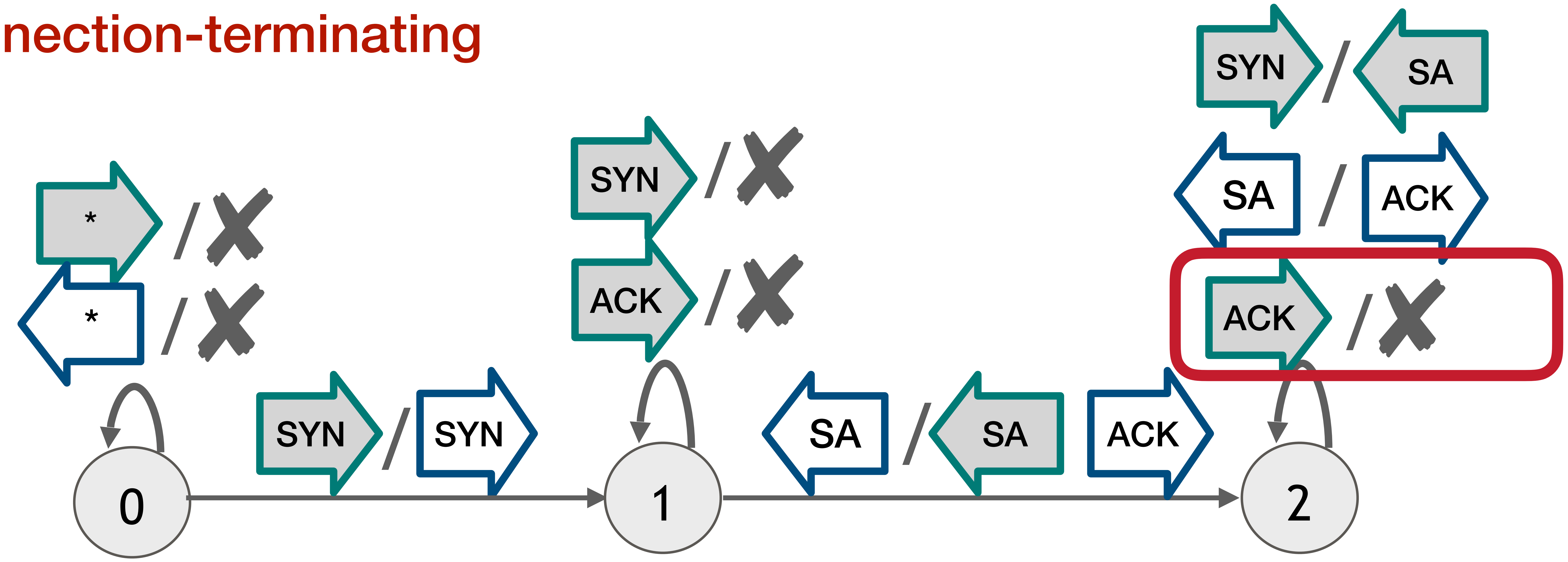
- Implements “default allow”
- Connection-terminating



When B responds with SA,
the FW preemptively responds with ACK

Firewall Case Study: Untangle Firewall

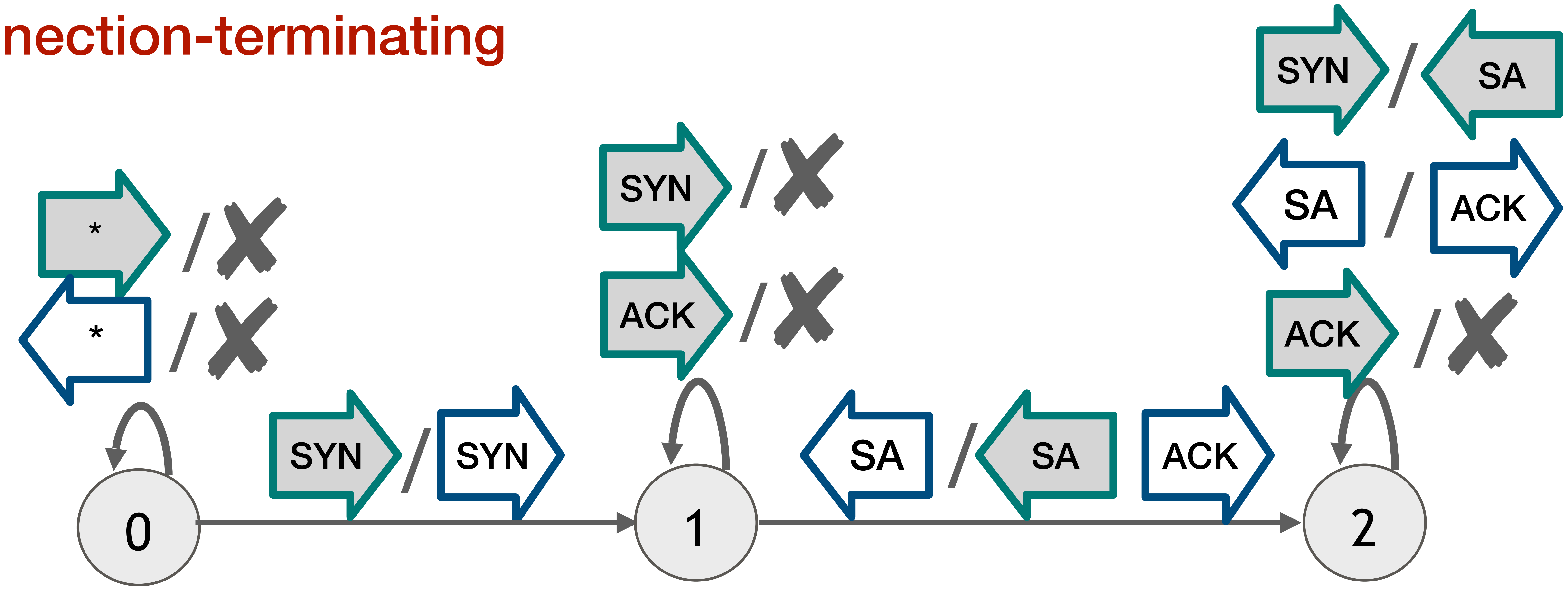
- Implements "default allow"
- Connection-terminating



When A replies with ACK, the FW drops to prevent duplicates

Firewall Case Study: Untangle Firewall

- Implements “default allow”
- Connection-terminating



Takeaways:

1) Vendor diversity (no common practice)
2) The real FSMs are complex and are infeasible for humans to manually generate

Other Findings

- **FW:** models with incorrect seq → large FSM (257 states for PfSense)
- **FW:** many do not correctly handle out-of-window packets
- **LB:** HAproxy (connection-terminating) vs. PfSense (destination NAT)

. . .

Evaluation Summary

- Alembic-generated models are **accurate**
- Case Studies: Alembic finds **differences across NF implementations**
- Alembic workflow is **scalable**
- Alembic-generated models **improve the accuracy of network testing/verification tools**

Scalability of Alembic Online

Number of Rules	Runtime
10	0.075 s
100	0.6 s
1,000	5 s

Alembic can generate concrete models in a few seconds for a large config

Limitations and Future Work

Assumption on configurations:

- Assume at most one rule is applied
- States across different state granularities (i.e., keys) are independent
- Assume that IP/port fields are treated homogeneously such that we can pick one representative sample and infer a model

Assumption on NF actions:

- Focused on modeling TCP-relevant behavior where actions are restricted to dropping and forwarding, possibly with IP/port modifications
- Do not explicitly model temporal effects
- Support the following state granularity types: per-connection, per-source, per-destination, cross-connection, and stateless

Future work:

- Dealing with more complex NFs (e.g., rate-limiting NF, modeling temporal effects)

Conclusions: Alembic can accurately model stateful NFs

- Network testing and verification today need NF models
- Handwritten models: tedious, error-prone, and inaccurate
- Alembic: infers a high-fidelity NF model given a configuration
- Our evaluations show:
  - Alembic finds implementation-specific behavior of NFs
  - Alembic-generated models increase the accuracy of testing/verification
  - Alembic is scalable and accurate

Soo-Jin Moon: soojinm@andrew.cmu.edu