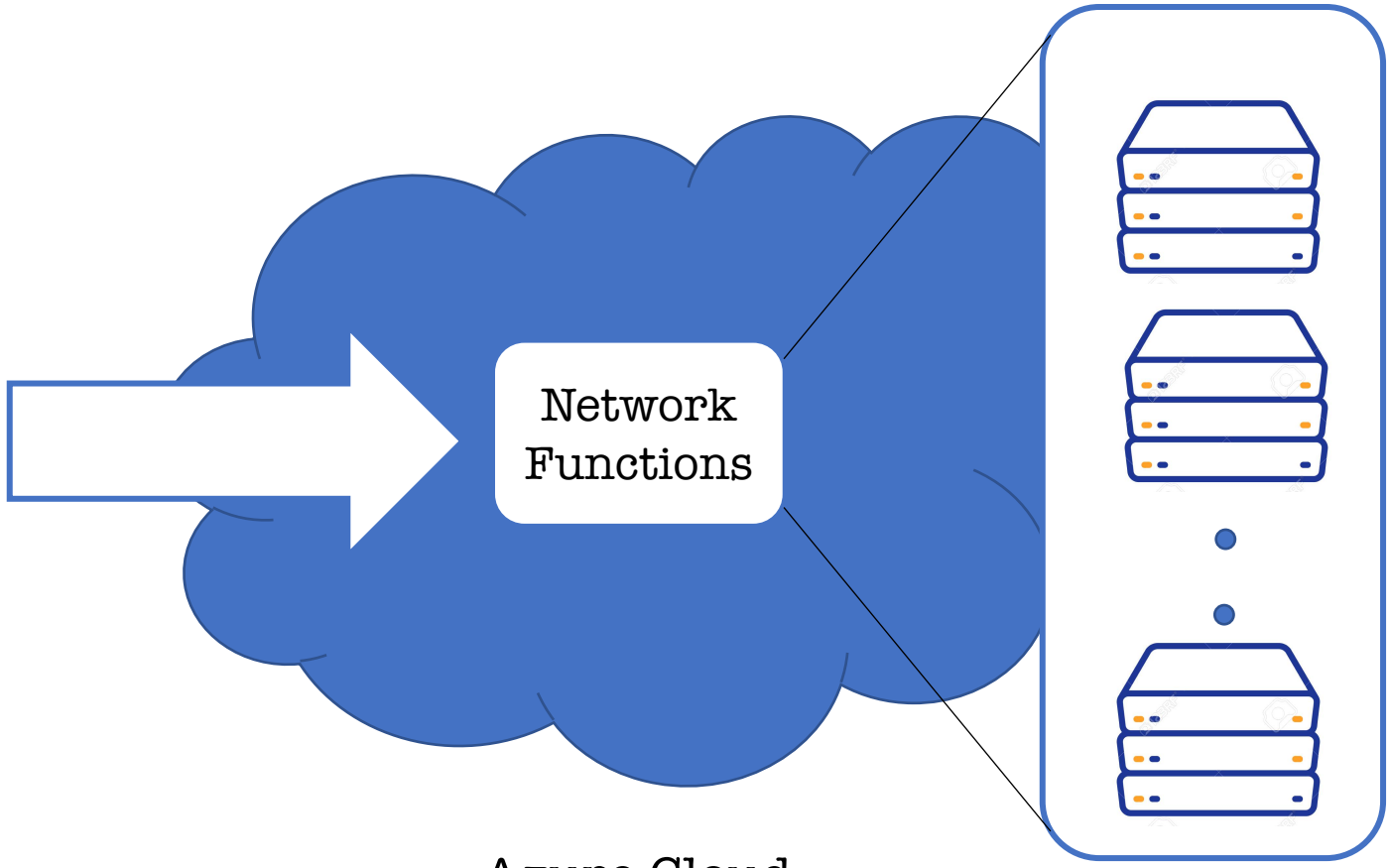
# Aragog: Scalable Runtime Verification of Shardable Networked Systems

***Nofel Yaseen***, Behnaz Arzani, Ryan Beckett, Selim Ciraci, Vincent Liu

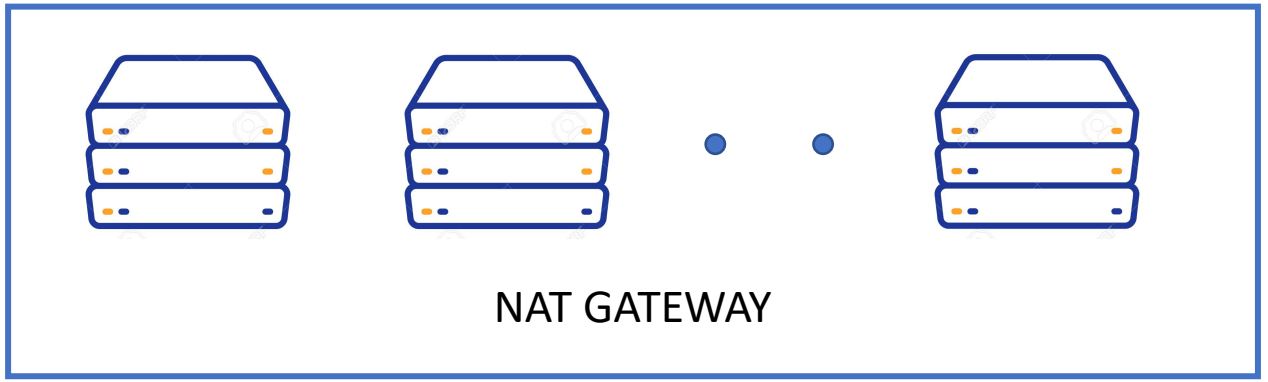

# Azure Cloud

- Networks functions are emerging bottleneck to correctness and availability



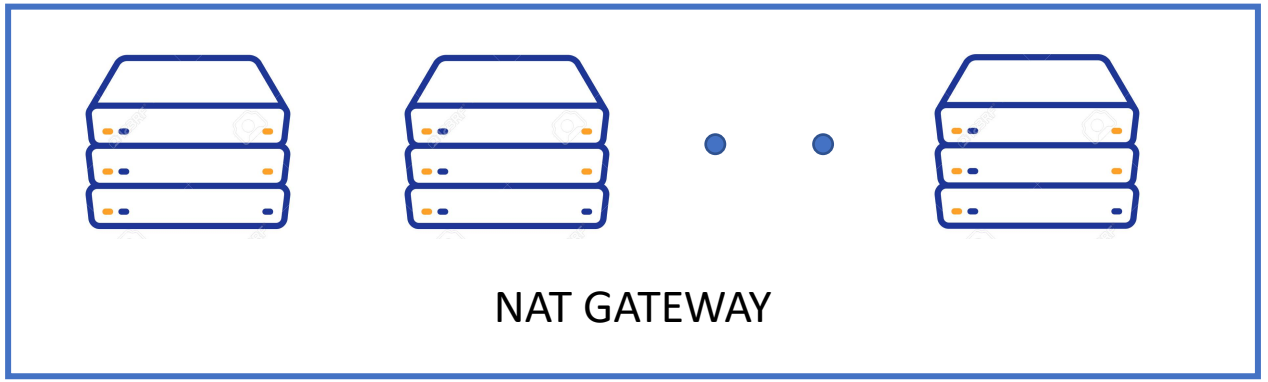
# NAT Gateway (NATGW)

- Network Address Translator (NAT)
- Software Load Balancer (SLB)

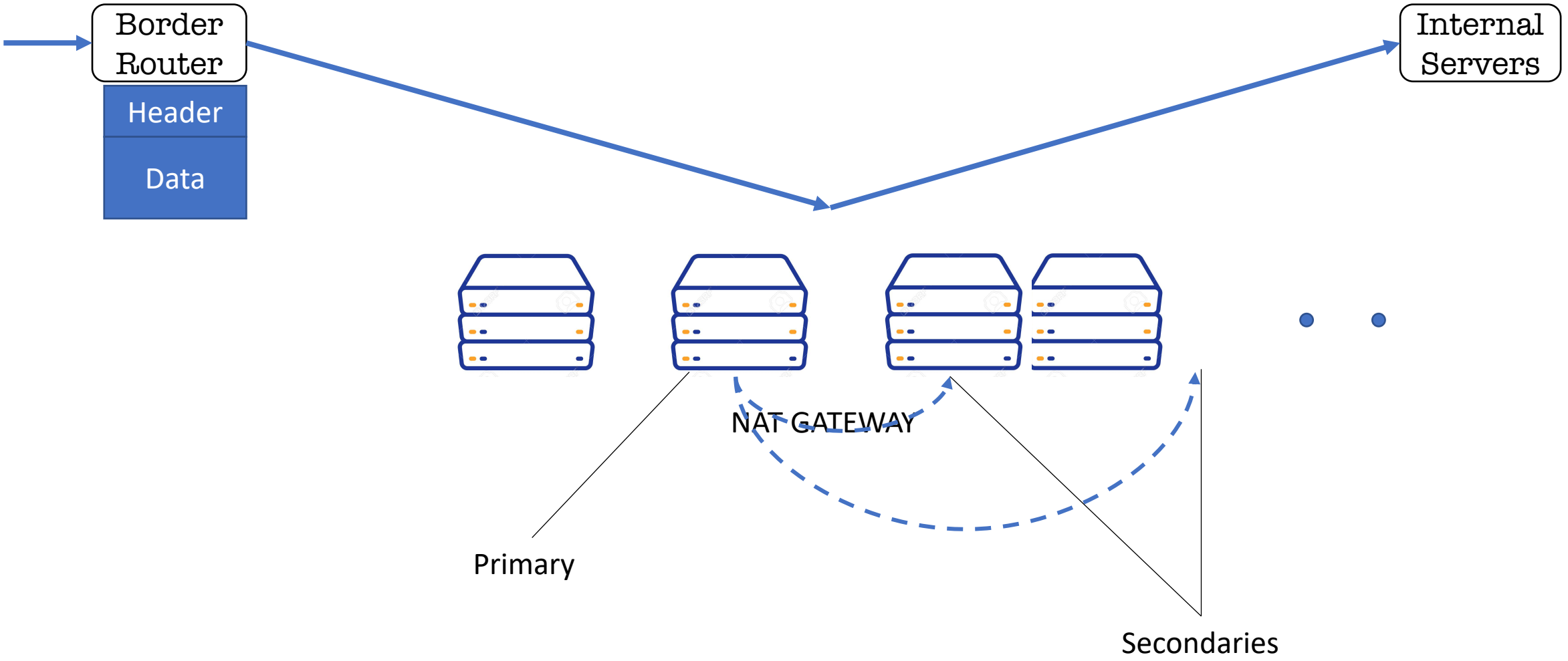


# NAT Gateway (NATGW)

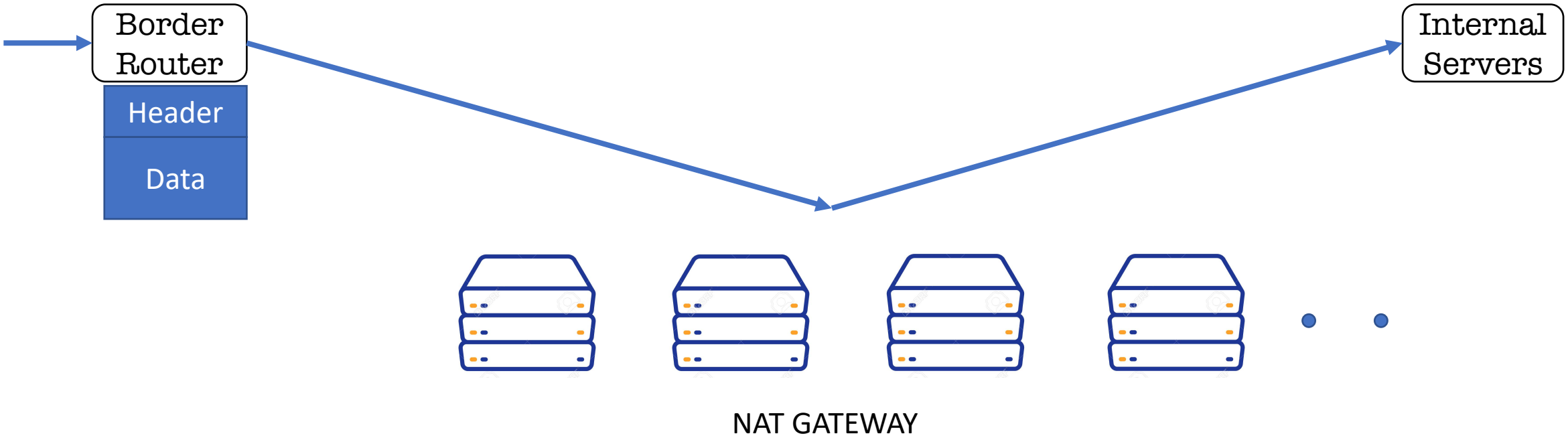
- Network Address Translator (NAT)
- Software Load Balancer (SLB)



# NAT Gateway (NATGW)

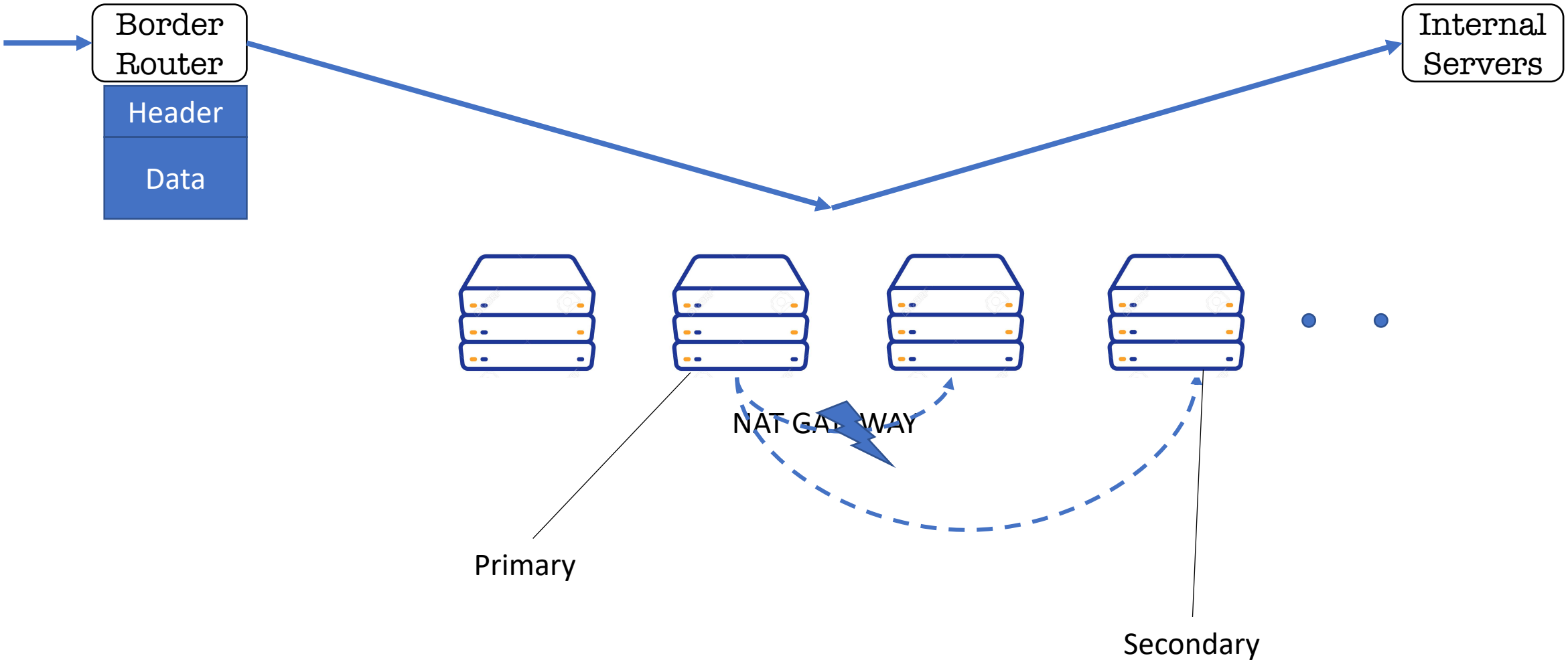


# NAT Gateway (NATGW)

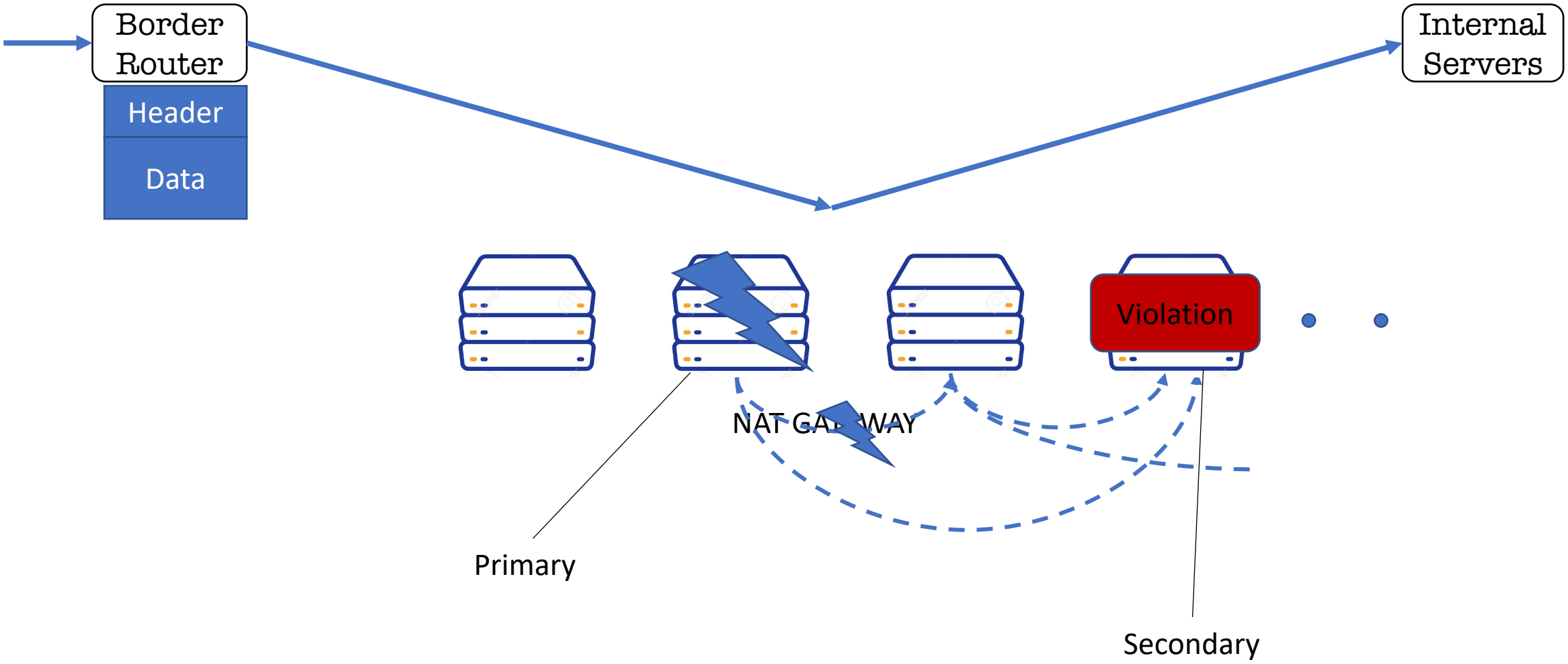



- Distributed
- Handles 100 M flows

# NAT Gateway (NATGW)



# NAT Gateway (NATGW)



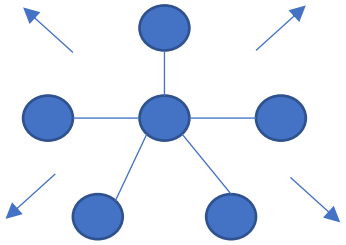
# Ensuring correctness in today's NF

- Testing

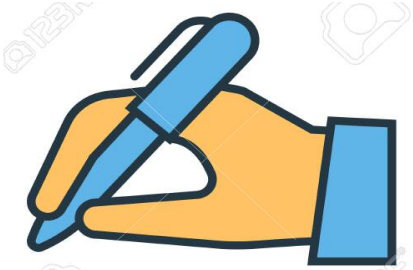
Low level

Expensive

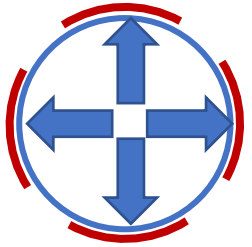
- Static Verification



State space explosion



Handwritten models



Limited Scope of Verification

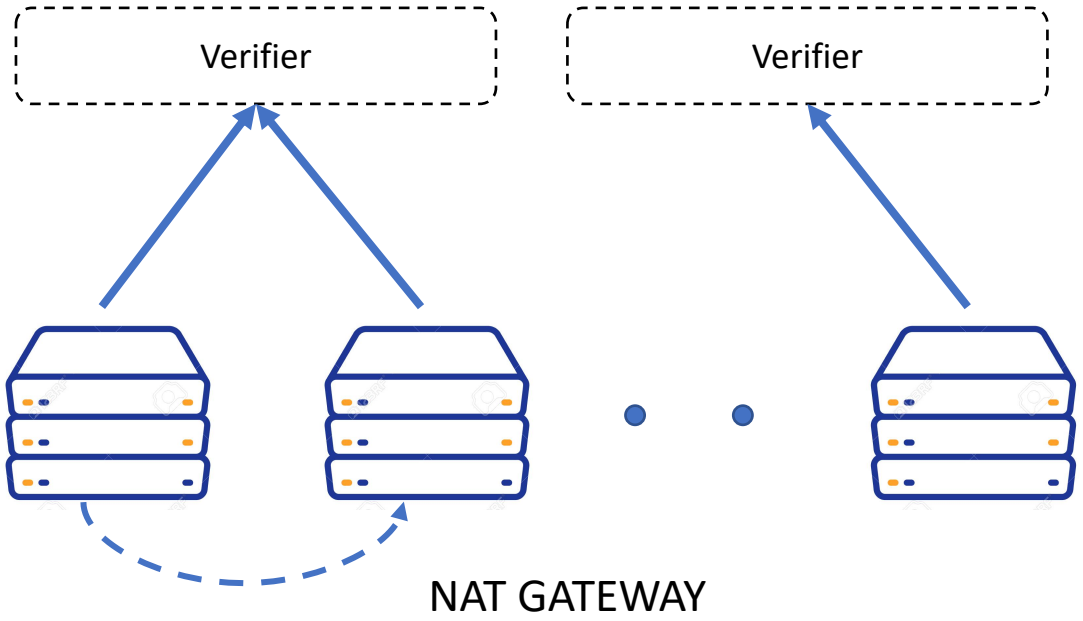
# Ensuring correctness in today's NF

- Runtime Verification
  - Test control flow, workloads, and the deployed systems' implementation.
  - Do not require handwritten models

# Today's Runtime verification

Challenges:

1. Reason about coordination among different nodes.
2. Efficiently aggregate global state from local views.
3. Scale sub-linearly with the size of the original network functions.



# Today's Runtime verification

Challenges:

1. Reason about coordination among different nodes.
2. Efficiently aggregate global state from local views.
3. Scale sub-linearly with the size of the original network functions.

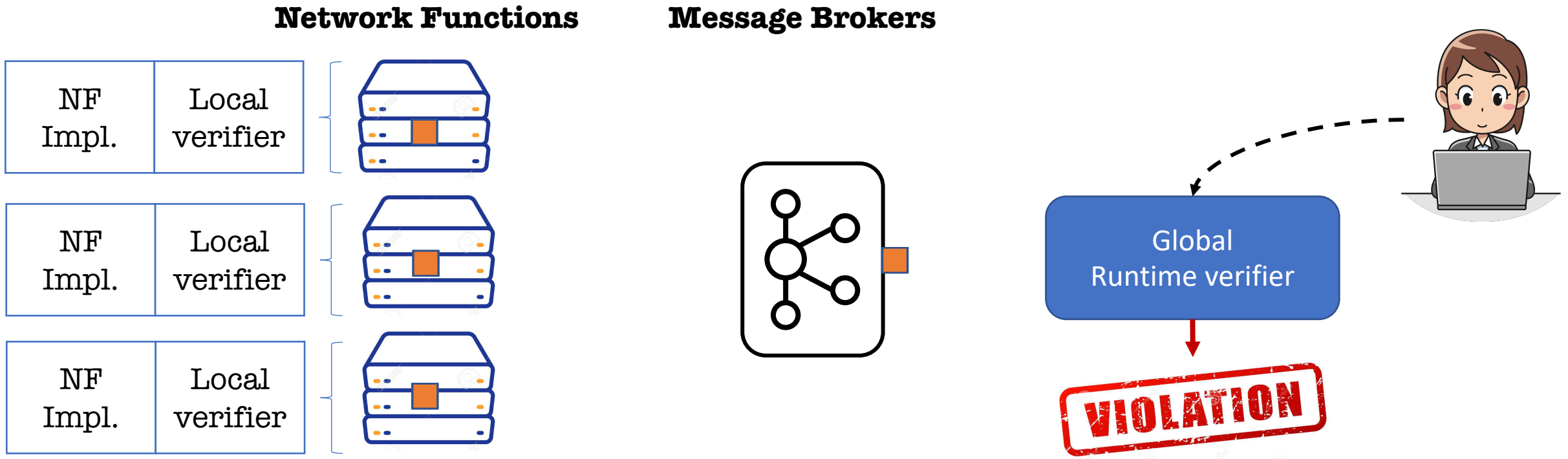
Write invariants to reason about the coordination among nodes

Design logically centralized verifier

Use stream processing framework for scale

# Aragog Architecture

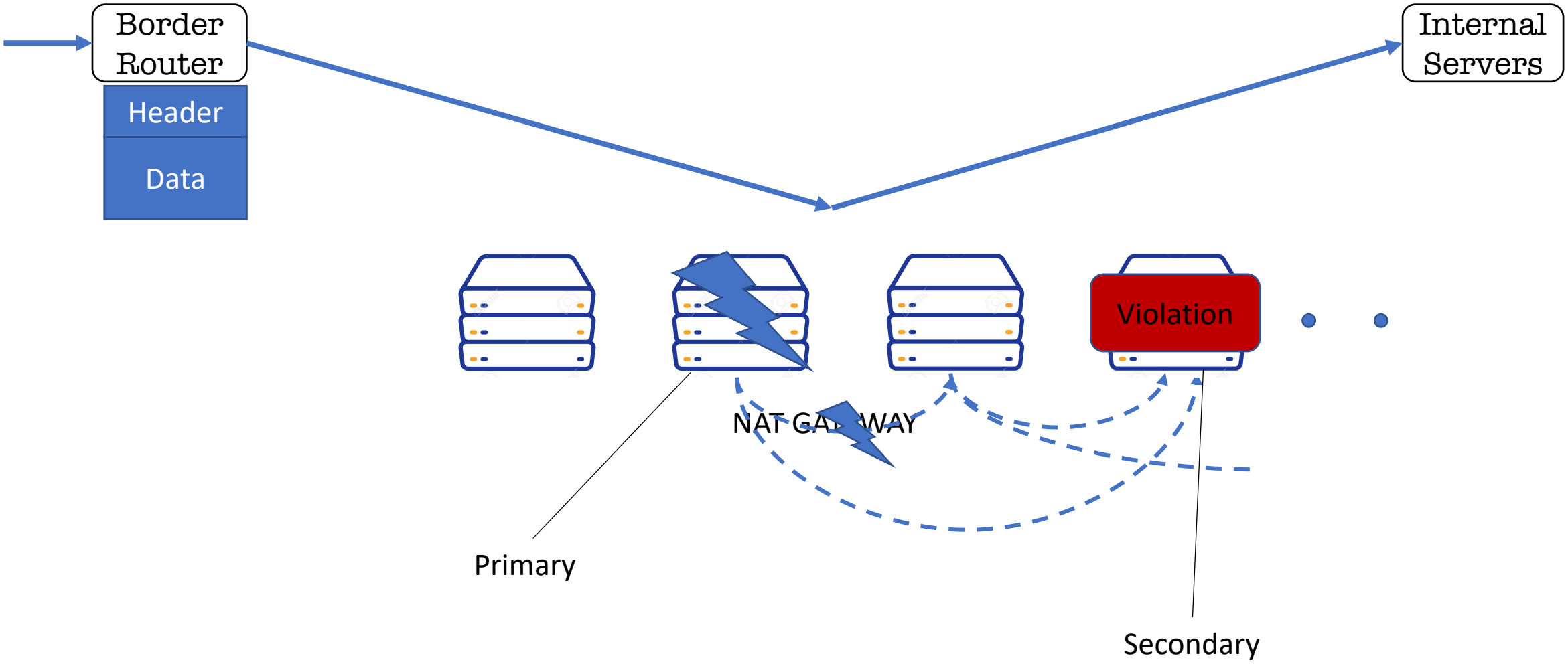
- Language for specifying invariant violations.
- Runtime system to check for violations in real time.



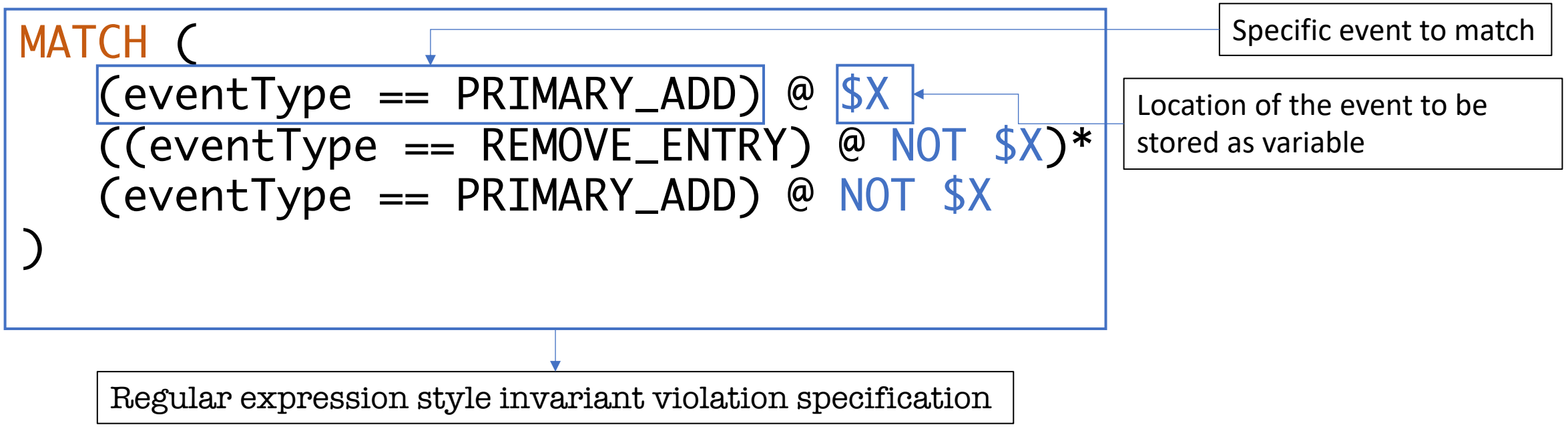
# Outline

- Violation example
- Language
- Compilation to Finite Automata
- Runtime System
- Implementation
- Evaluation

# NAT Gateway (NATGW)

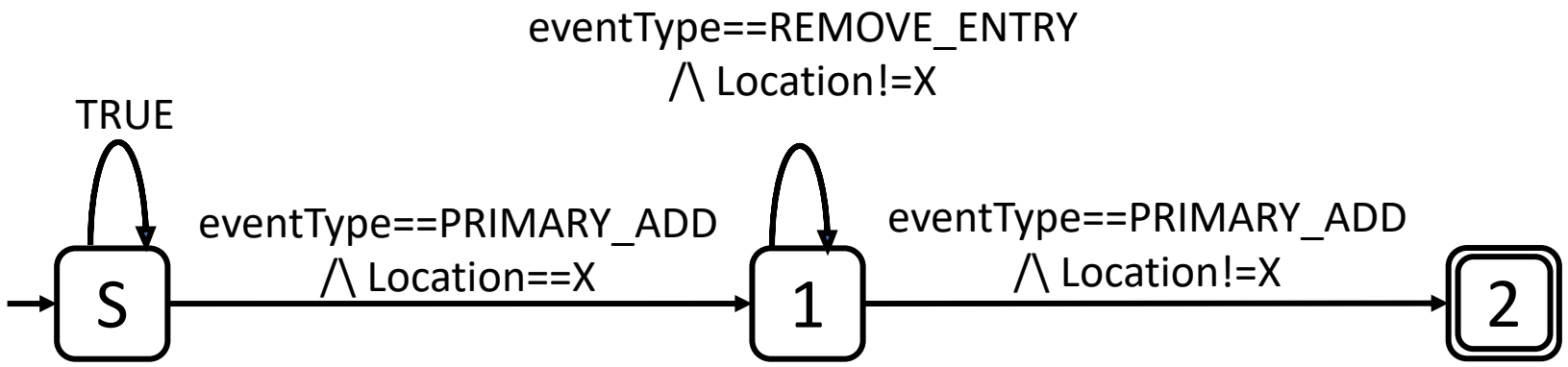


# Specification Language



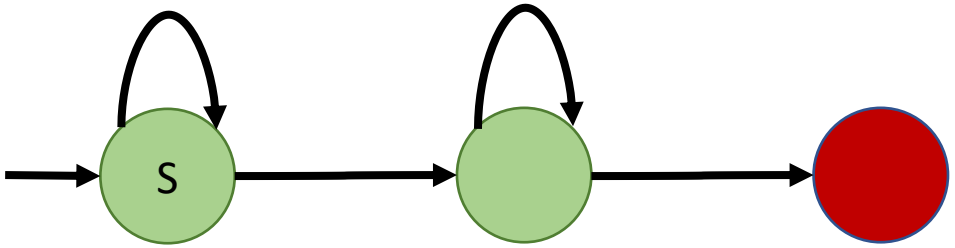
# State Machine Generation

```
MATCH (
    (eventType == PRIMARY_ADD) @ $X
    ((eventType == REMOVE_ENTRY) @ NOT $X)*
    (eventType == PRIMARY_ADD) @ NOT $X
)
```

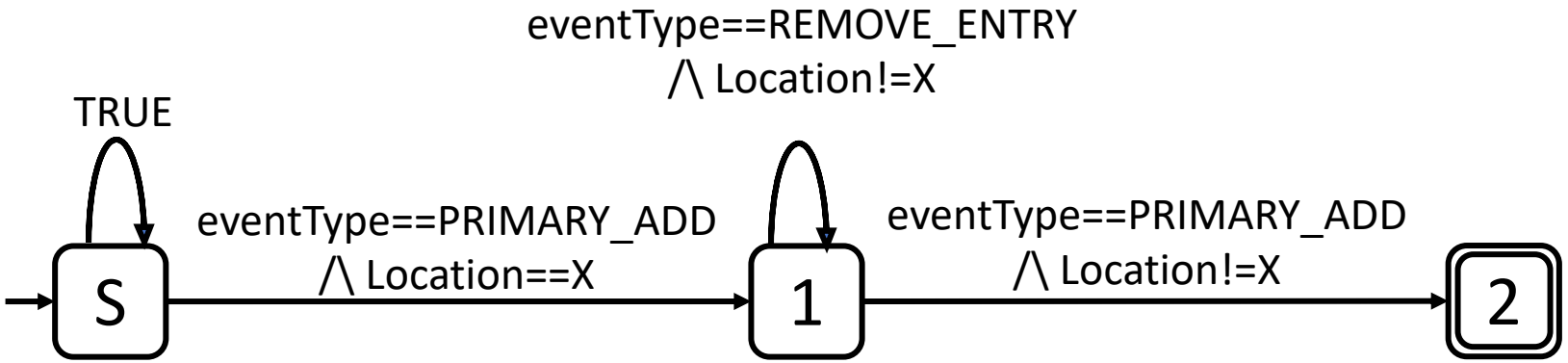



Symbolic Finite Automata (SFA)

# State Machine Generation

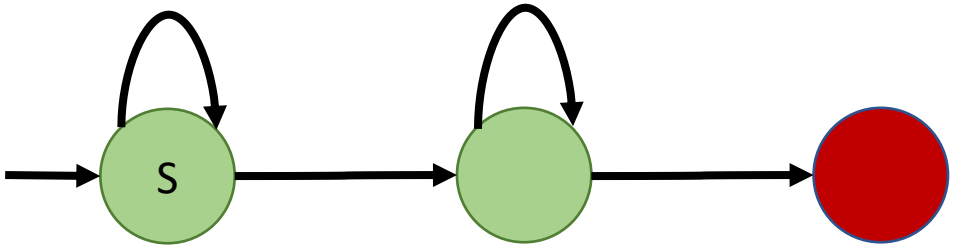



Symbolic Finite Automata (SFA)

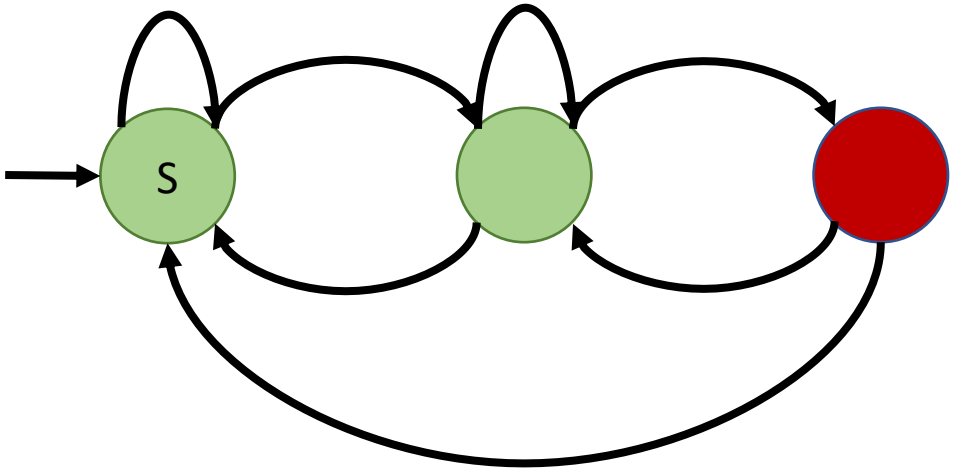



Symbolic Finite Automata (SFA)

# State Machine Generation




Symbolic Finite Automata (SFA)

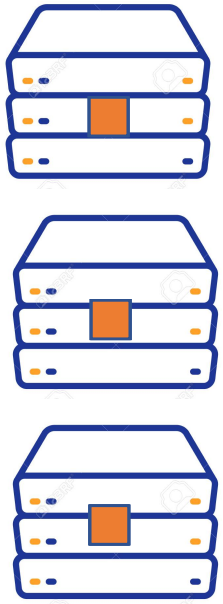



Determinization

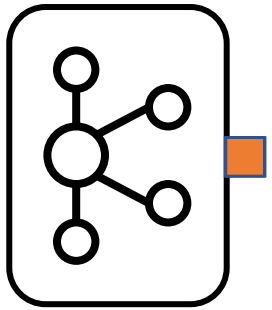
DSFA

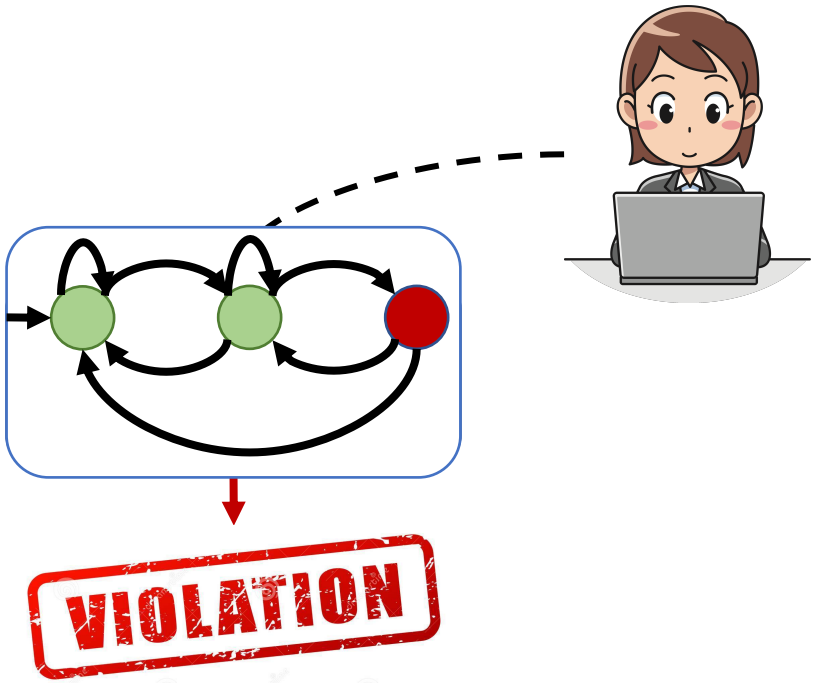
# State Machine Generation

**Network Functions**



**Message Brokers**



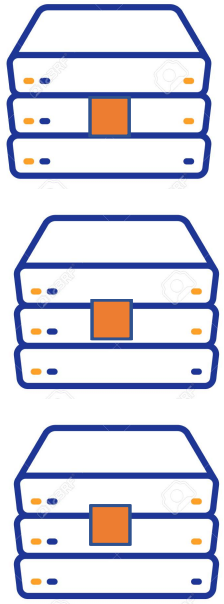


# Optimizations

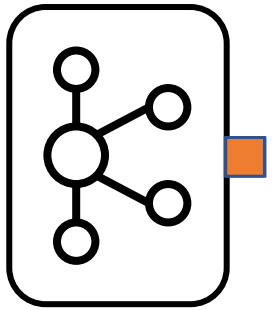
1. Filtering – focus on relevant events
2. Sharding – separate out events for different flows
3. Suppressing – if possible, check invariants locally

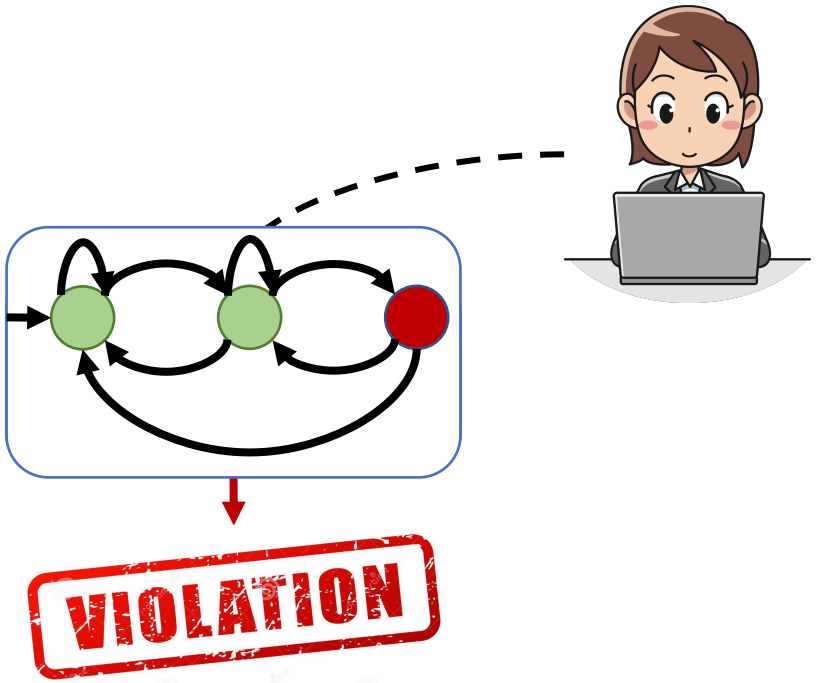
# Filtering

**Network Functions**

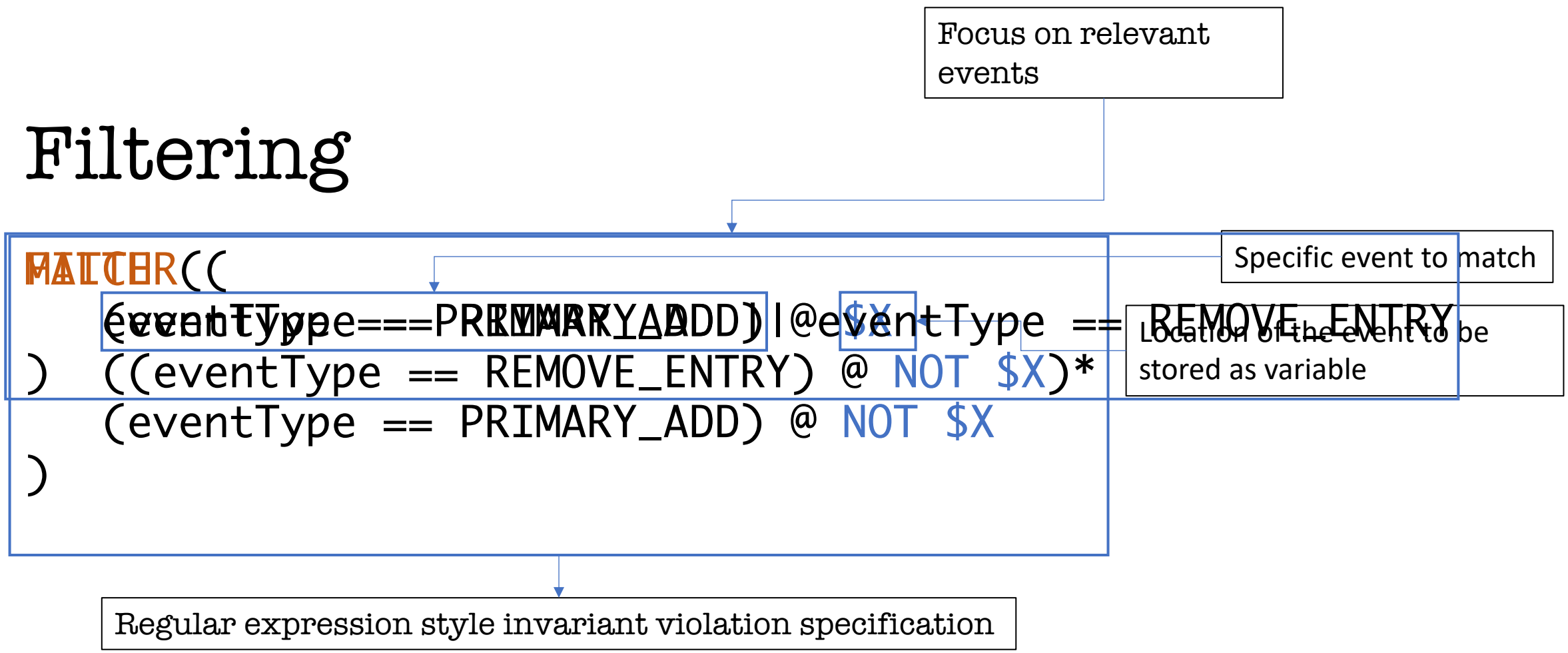


**Message Brokers**



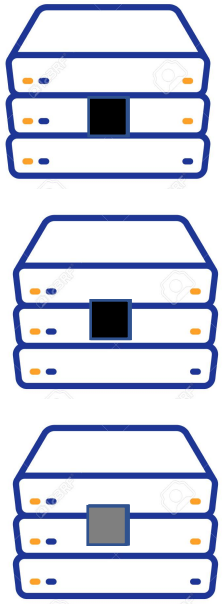


# Filtering

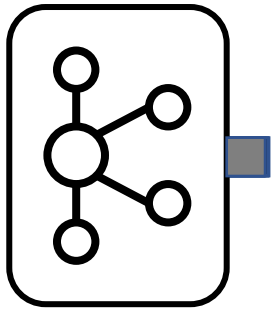

Focus on relevant events
PATTERN((
(eventType === PRIMARY_ADD)@$X
) ((eventType == REMOVE_ENTRY) @ NOT $X)*
(eventType == PRIMARY_ADD) @ NOT $X
))
Specific event to match
Location of the event to be stored as variable
Regular expression style invariant violation specification

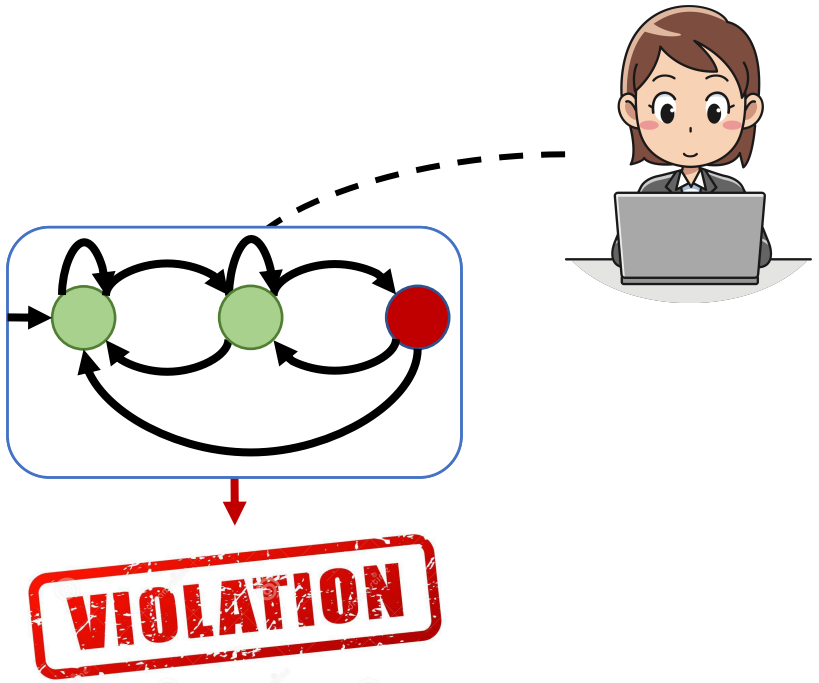
# Sharding

**Network Functions**



**Message Brokers**





# Sharding

Focus on relevant events

Allows to separate out violation check for each flow

```
FILTER (
    eventType == PRIMARY_ADD || eventType == REMOVE_ENTRY
)
GROUPBY (srcIP, dstIP, srcPort, dstPort, proto )
```

```
MATCH (
    (eventType == PRIMARY_ADD) @ $X
    ((eventType == REMOVE_ENTRY) @ NOT $X)*
    (eventType == PRIMARY_ADD) @ NOT $X
)
```

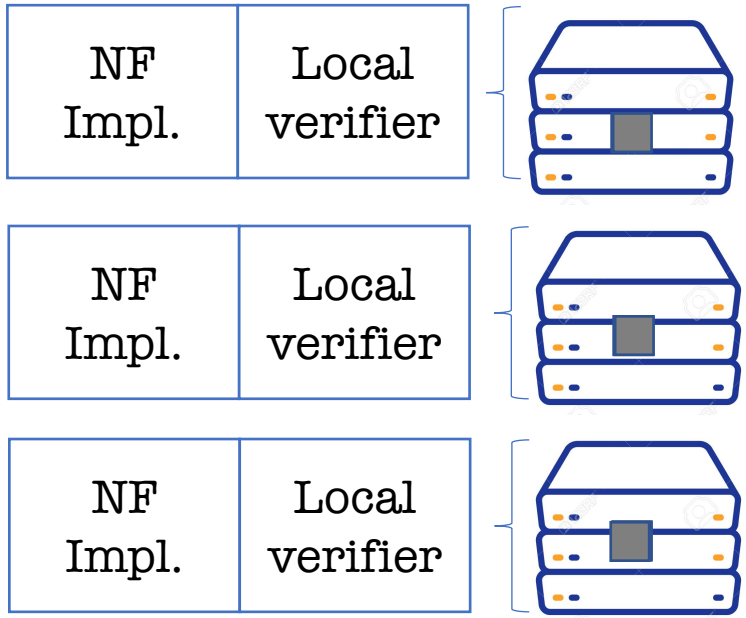
Specific event to match

Location of the event to be stored as variable

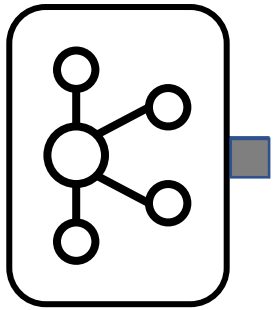
Regular expression style invariant violation specification

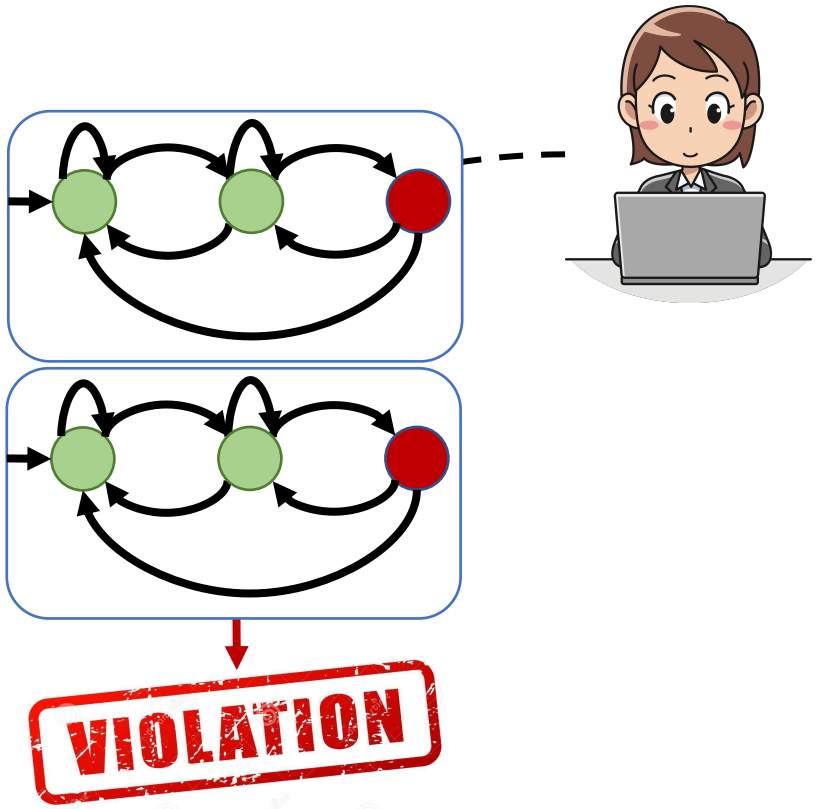
# Suppressing

**Network Functions**

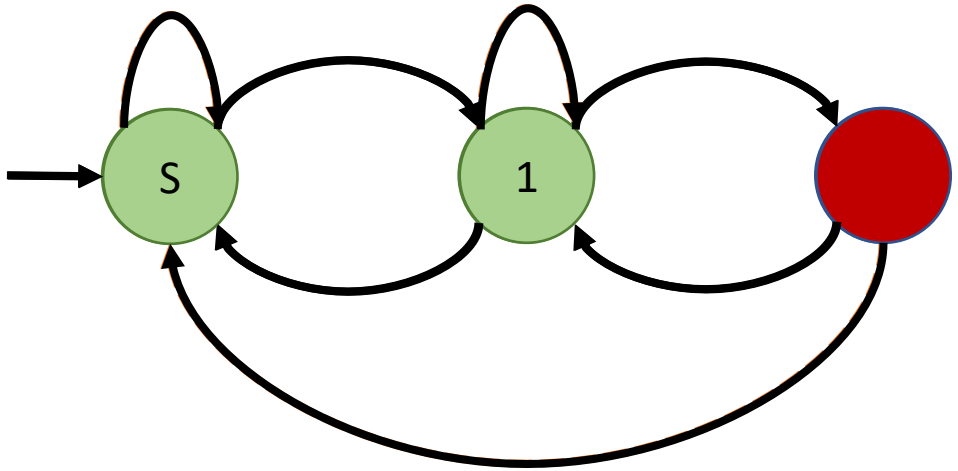



**Message Brokers**

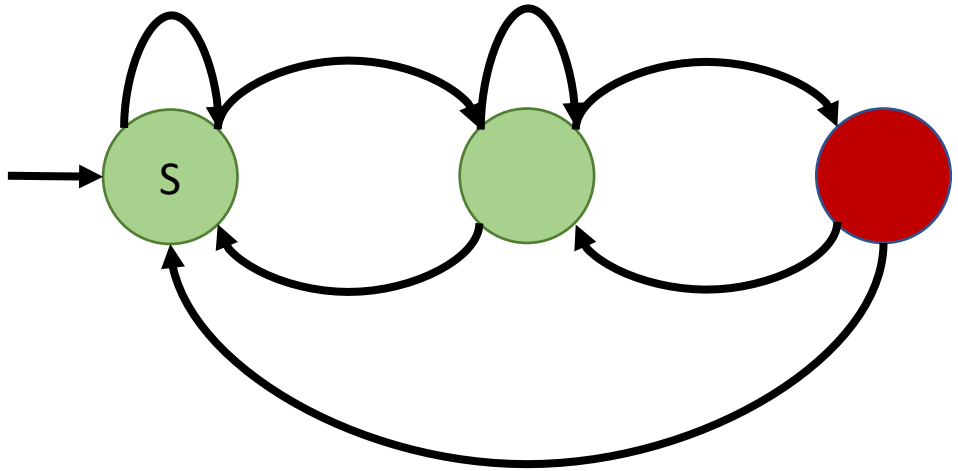


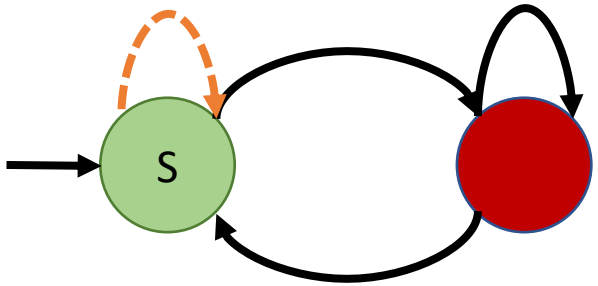


# Suppressing




DSFA

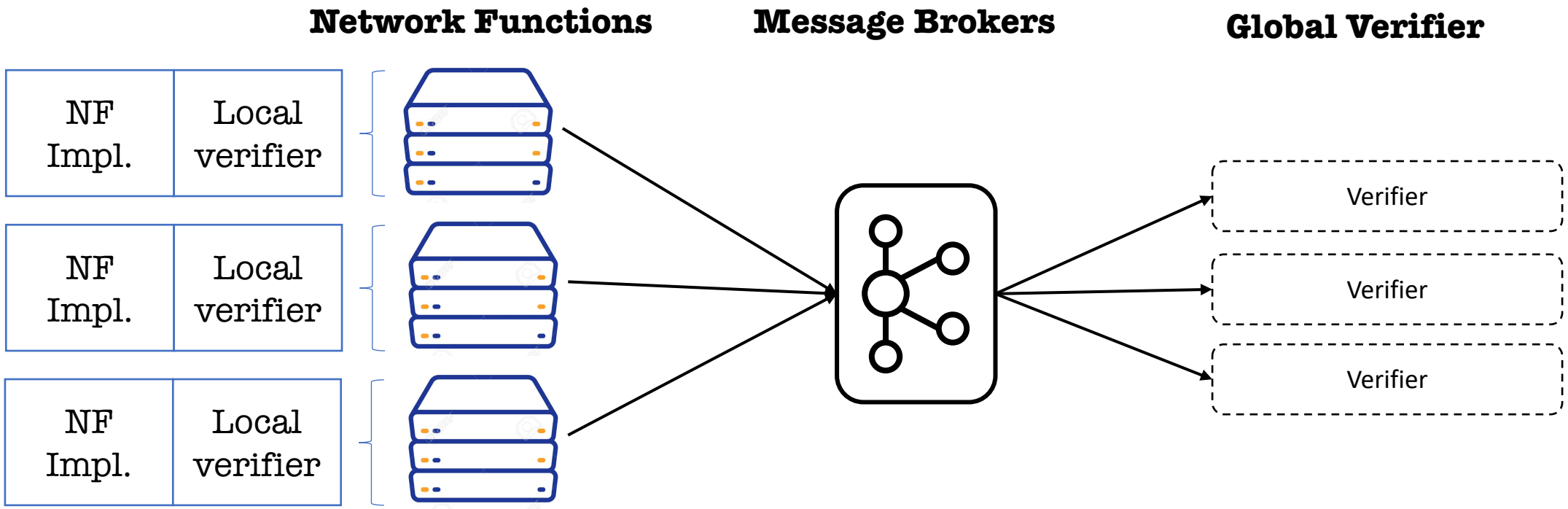



LOCAL DSFA

Create Local DSFA

# Runtime System



# Implementation

**SFA Generation**

- Java
- Antlr
- Symbolic Automata
- z3

**Local Verifier**

- C++
- Antlr
- CppKafka

**Global Verifier**

- Java
- Antlr
- Apache Kafka
- Apache Flink

# Evaluation

- NATGW
  - 2 traces from test cluster with 32 M total events
  - 8 invariants

- Distributed firewall
  - 10-minute runs based on DCTCP traffic distribution
  - 3 invariants

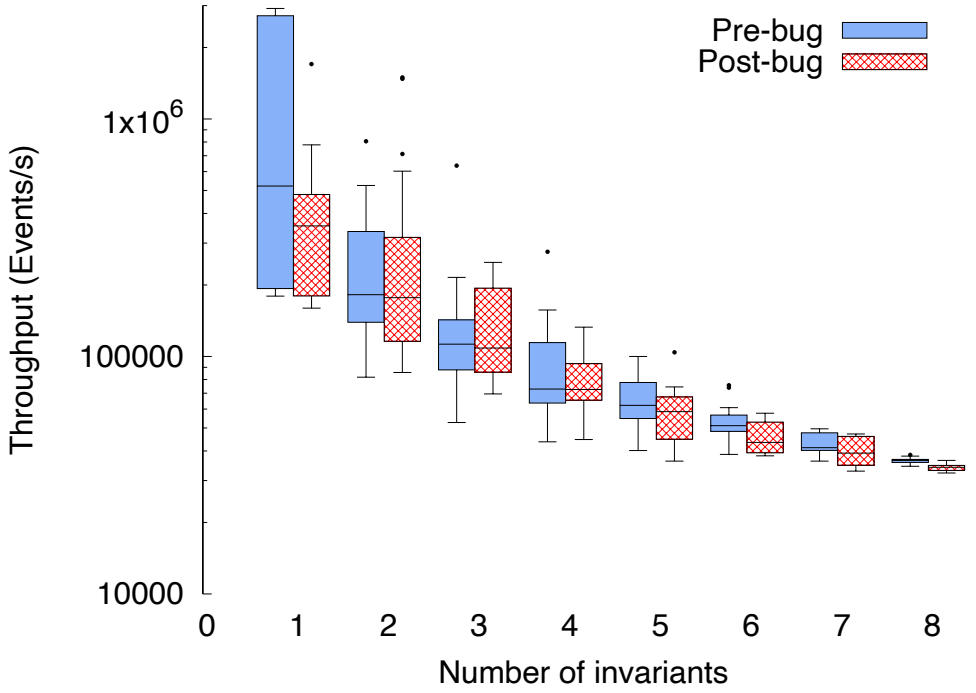
# NATGW violations

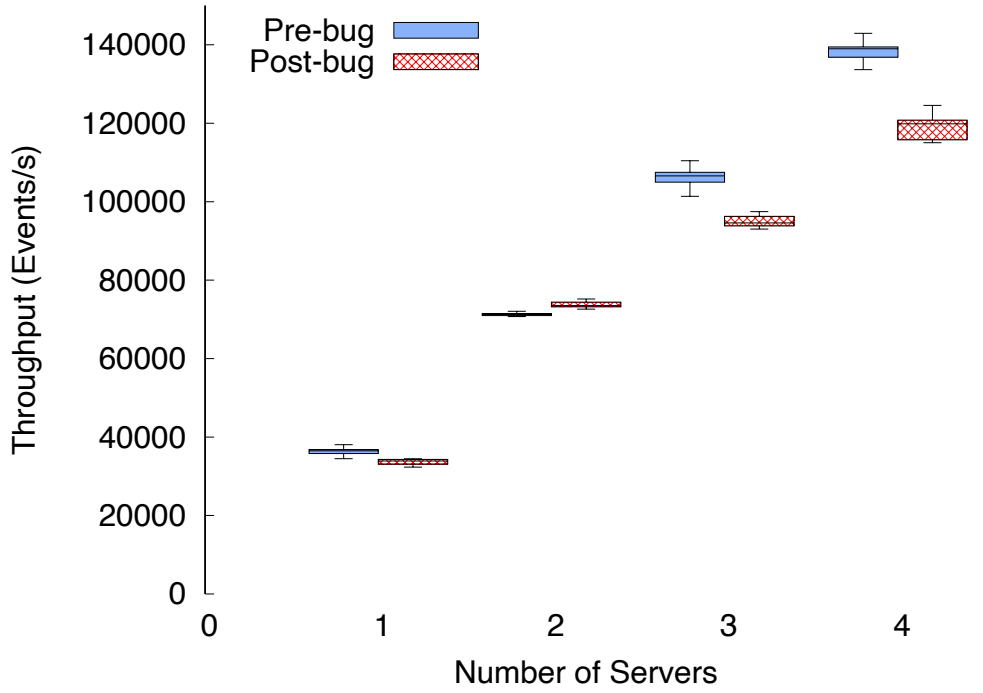
| Invariant | Lines of Code |
|---|---|
| consensus | 5 |
| open_to | 5 |
| syn_to | 5 |
| primary_single | 10 |
| same_consensus | 12 |
| udp_same_consensus | 12 |
| primary_to | 13 |
| decider_open | 14 |

# NATGW violations

| Invariant | Lines of Code | Pre-bug Trace | Post-bug Trace |
|---|---|---|---|
| consensus | 5 | 0 | 0 |
| open_to | 5 | 0 | 45019 |
| syn_to | 5 | 0 | 2697 |
| primary_single | 10 | 0 | 0 |
| same_consensus | 12 | 536 | 259 |
| udp_same_consensus | 12 | 0 | 0 |
| primary_to | 13 | 1 | 29964 |
| decider_open | 14 | 0 | 0 |

# Throughput

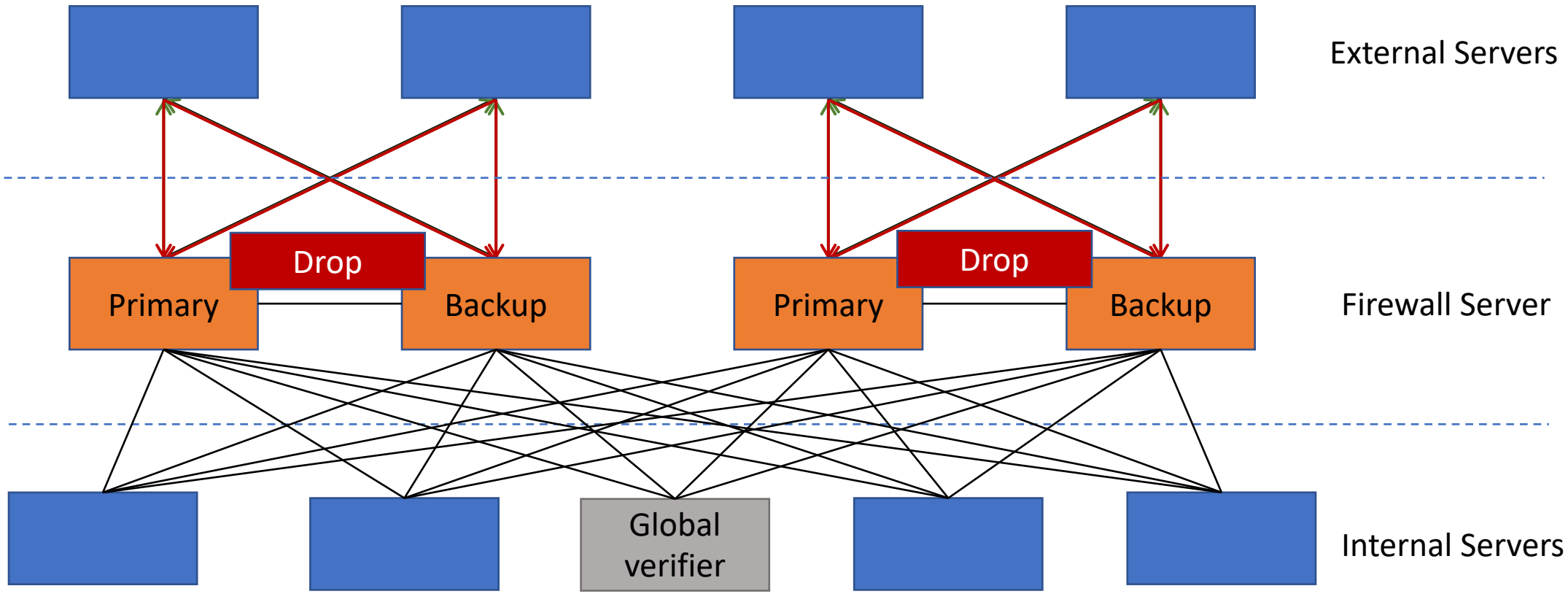



- Can process for more than a million event/s for a single invariant
- Can process more than 30,000 events/s for 8 invariants

- Aragog scales linearly as we add more servers
- Scaling with multiple servers avoid the bottleneck of CPU/IO

# Distributed Firewall



# Distributed Firewall

- All violations found

- Local verifier CPU usage: ~ 15 %
- Local verifier Memory usage: ~ 20 MB

- Global verifier CPU usage: 50 – 100 % mostly
- Global verifier Memory usage: ~ 1200 MB

- Latency to alert violation: ~ 100 ms

# Aragog summary

- Need to verify distributed network functions

- Aragog: Scalable Runtime Verification of Shardable Networked Systems
  - Language to specify invariant violation
  - Automata Based Verification

- https://github.com/microsoft/aragog
- Questions: nyaseen@seas.upenn.edu
- Thank you!!