# Reflections on Data Integration for SDN

Anduo Wang† Jason Croft* Eduard Dragut†

†Temple University *University of Illinois at Urbana-Champaign

SDN-NFV Security ’17

March 24, 2017

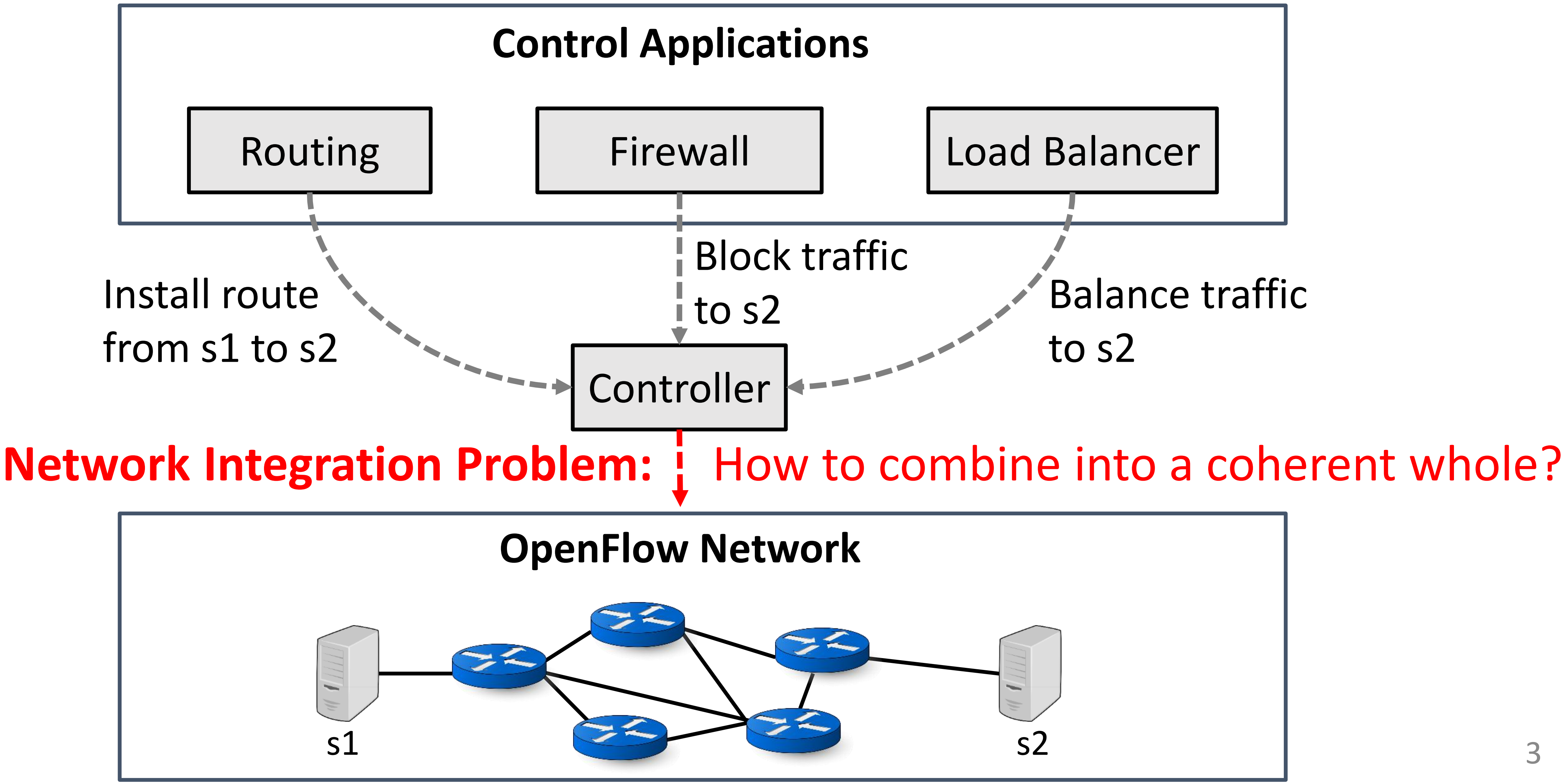
# SDN Design Principles

- SDN builds off principles from other areas of research to simplify control:
  - Programming languages
  - Operating systems
  - Distributed systems

- Contributes to design of network control via high level abstractions

- We propose: building on principles from databases, namely data integration

# Composing SDN Application is Still Hard



# Example: Firewall and Load Balancer

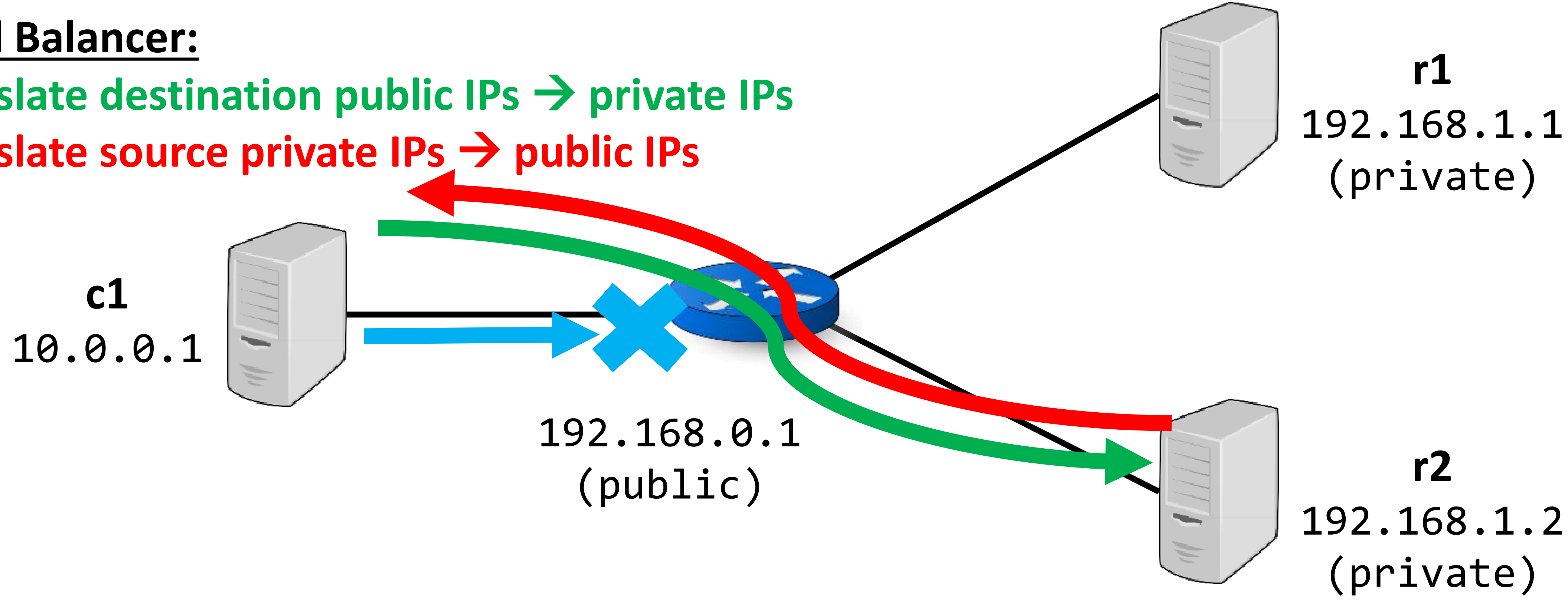
**Firewall:**

Blacklist (public IP, client IP)

**Load Balancer:**

Translate destination public IPs → private IPs

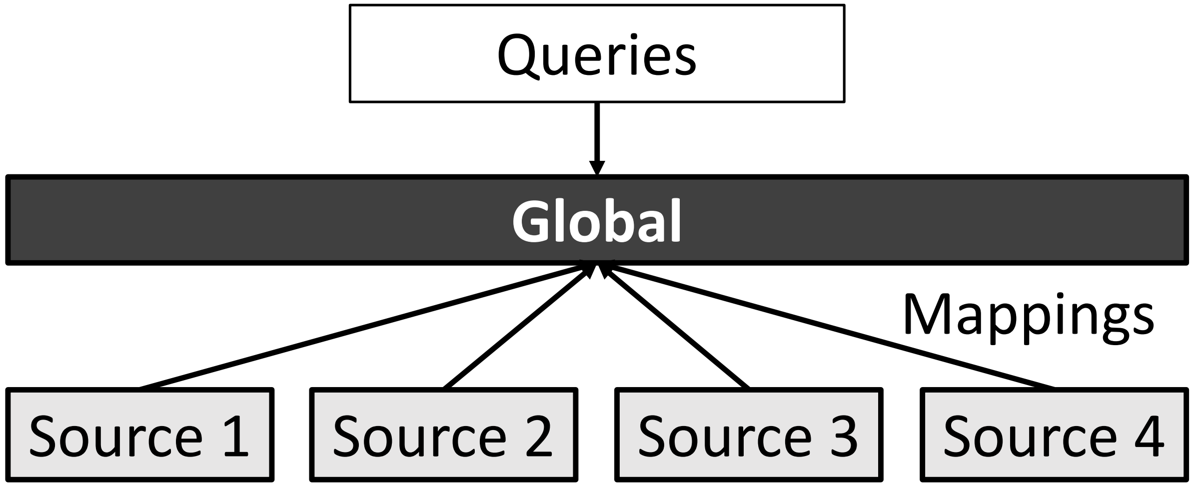
Translate source private IPs → public IPs




Correct composition: `if(from_client, fw>>lb, lb>>fw)`

# Building on Data Integration

- **Data integration:** combining data from multiple sources to create a unified whole
- Data integration system: $I = \langle G, S, M \rangle$
  - G: global schema
  - S: data sources
  - M: semantic mappings



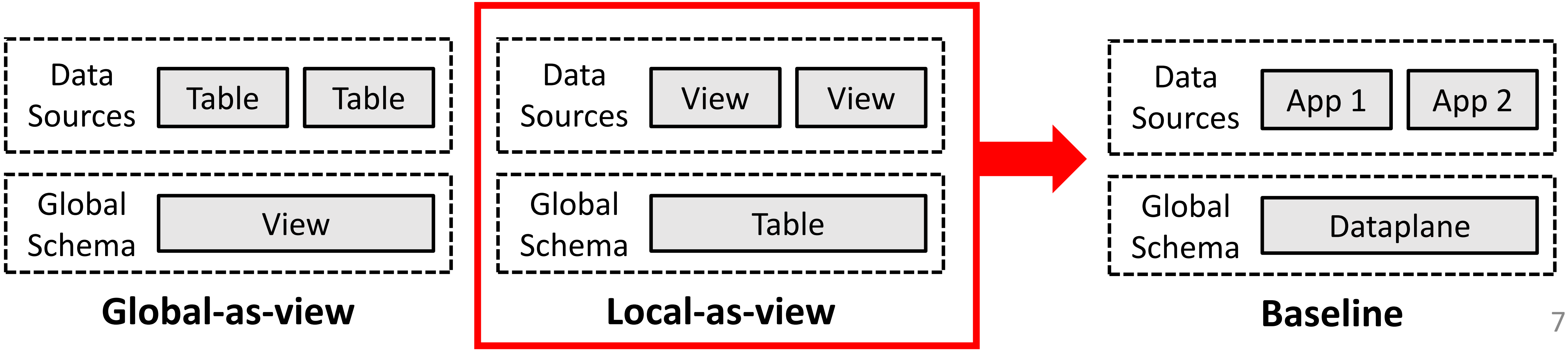
# Network Integration Problem

- Network integration system: $I^N = <G^N, S^N, M^N>$
  - $\boldsymbol{G^N}$ **:** consistent dataplane, with integrity constraints
  - $\boldsymbol{S^N}$ **:** network states contributed by applications
  - $\boldsymbol{M^N}$ **:** mapping synchronizing application states and dataplane under integrity constraints

- Two challenges:
  1. Performance: fast updates of global data arbitrarily complex integrity constraints
  2. Correctness: behavioral dependency between sources

# Challenge #1: Performance

- SDN applications have rich semantics, complex integrity constraints
- Dataplane must support these arbitrarily complex constraints
  - Each update must be checked against constraints, rolled back if violated
- Problem: fast writes and constraint checking
- **Solution: baseline design**



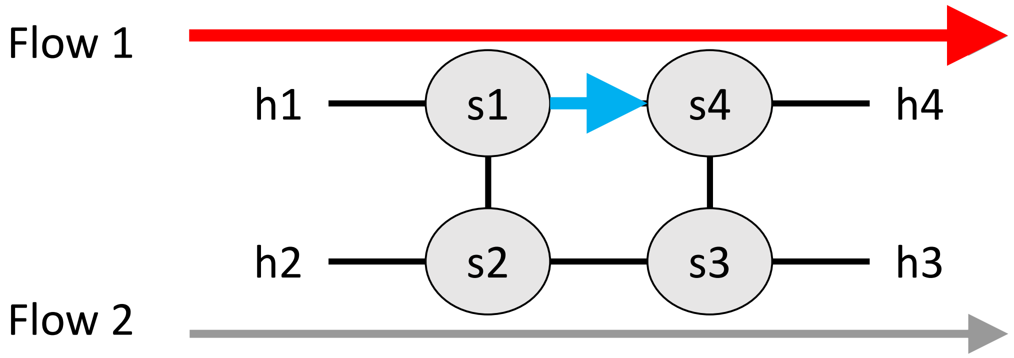
# Baseline Design

- Global dataplane ($G^N$) modeled as:

| topology | |
|---|---|
| sid | nid |
| s1 | s2 |
| s1 | h1 |
| s1 | s4 |
| … | |

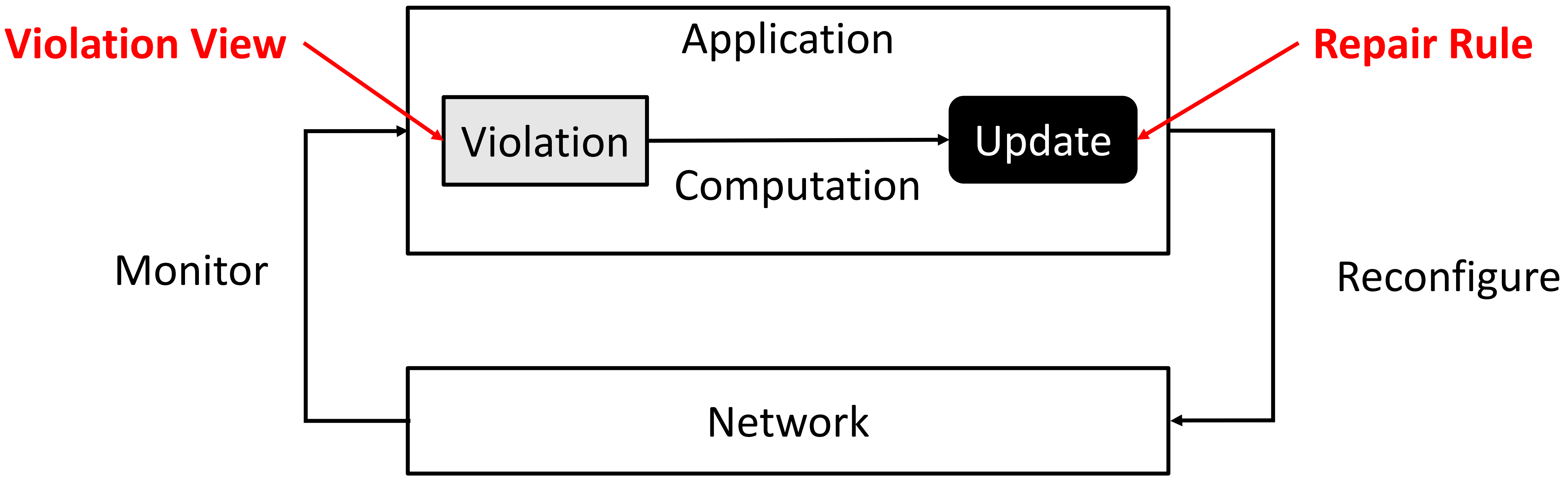
| reachability_matrix | | | | |
|---|---|---|---|---|
| fid | src | dst | vol | … |
| 1 | h1 | h4 | 1 | |
| 2 | h2 | h3 | 1 | |
| … | | | | |

| configuration | | |
|---|---|---|
| fid | sid | nid |
| 1 | s1 | s4 |
| 1 | s4 | h4 |
| … | | |



# View-Based Applications

- Control applications as data sources
  - Partial view and control of global schema $G^N$
  - Easily extensible
- SDN control software coded as a control loop with a monitor-reconfigure pattern



# Fast Updates with Violation Views

- Firewall example:

Policy Definition

```
CREATE TABLE fw_blacklist (
       end1   integer,
       end2   integer
);
```

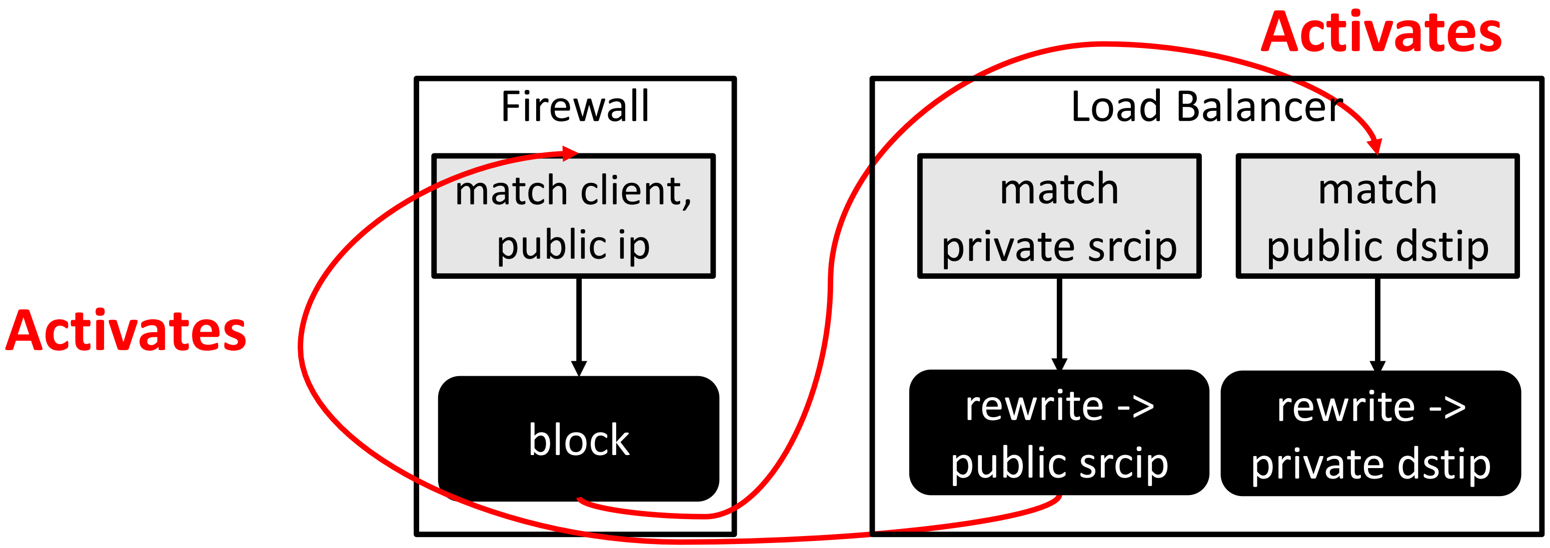
Violation View

```
CREATE VIEW fw_violation AS (
       SELECT fid FROM reachability_matrix
       WHERE (src, dst) NOT IN
       (SELECT end1, end2 FROM acl)
);
```

- Disable default constraint checking, rollbacks
- Instead, applications make smart updates that are guaranteed to respect constraints in the first place

# Challenge #2: Correctness

- Complex interactions between applications
- Applications require orchestration to resolve conflicts
- **Dependency:** one module's update may trigger violation of another
- If an operation in A depends on an operation in B, then A **activates** B



# Looking Forward: Building on Irrelevant Updates

- Cast as database irrelevant updates problem for views
  - Can an update to a base table (dataplane) affect a view (an application)?
- Statically analyze application and examine attributes
- Solve dependency as SAT problem