

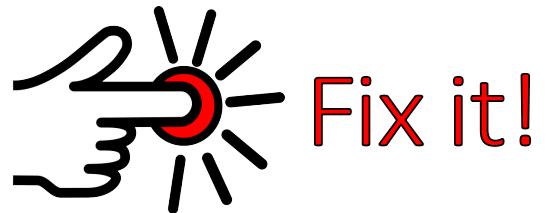


Penn  
UNIVERSITY OF PENNSYLVANIA



GEORGETOWN UNIVERSITY

# Automated Bug Removal for Software-Defined Networks



Yang Wu\*

Ang Chen\*

Andreas Haeberlen\*

Wenchao Zhou<sup>+</sup>

Boon Thau Loo\*

\* University of Pennsylvania

+ Georgetown University

# Motivation: Automated repair

1: if (switch == S0 && protocol == HTTP) then action = balancer().

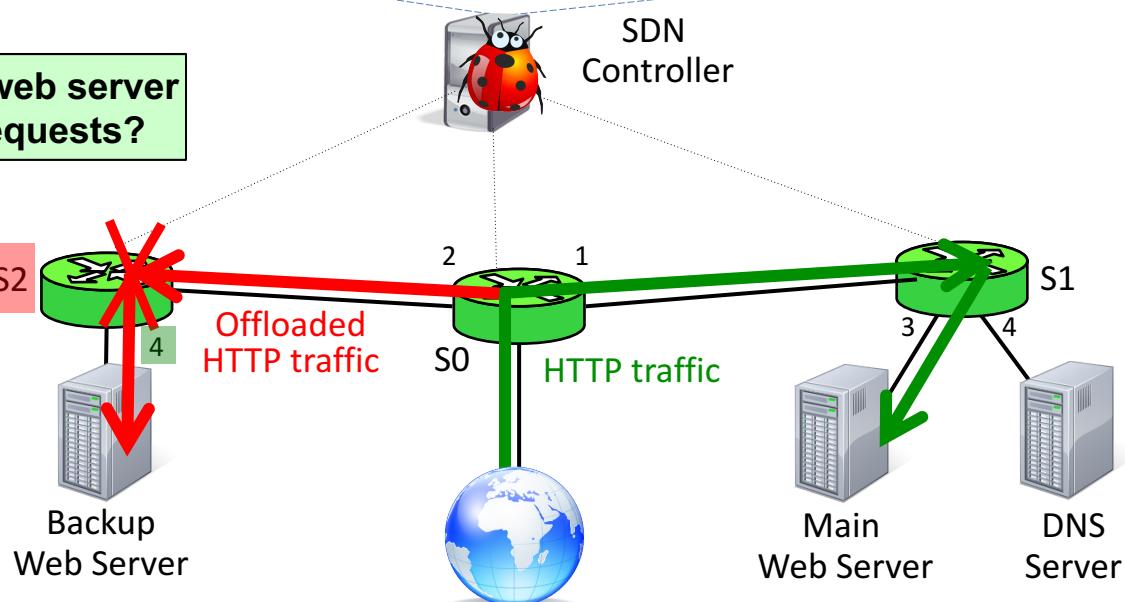
2: else if (switch == S1 && protocol == HTTP) then action = output:3.

3: else if (switch == S1 && protocol == HTTP) then action = output:4.

Copy-and-paste bug!!!

Copy!

Why is the backup web server  
not getting any requests?



# What does the operator need to do?



Aha! There was no matching flow entry on S2!



I can fix this by changing S1 in the controller program to S2!

## Step 1. Diagnosis

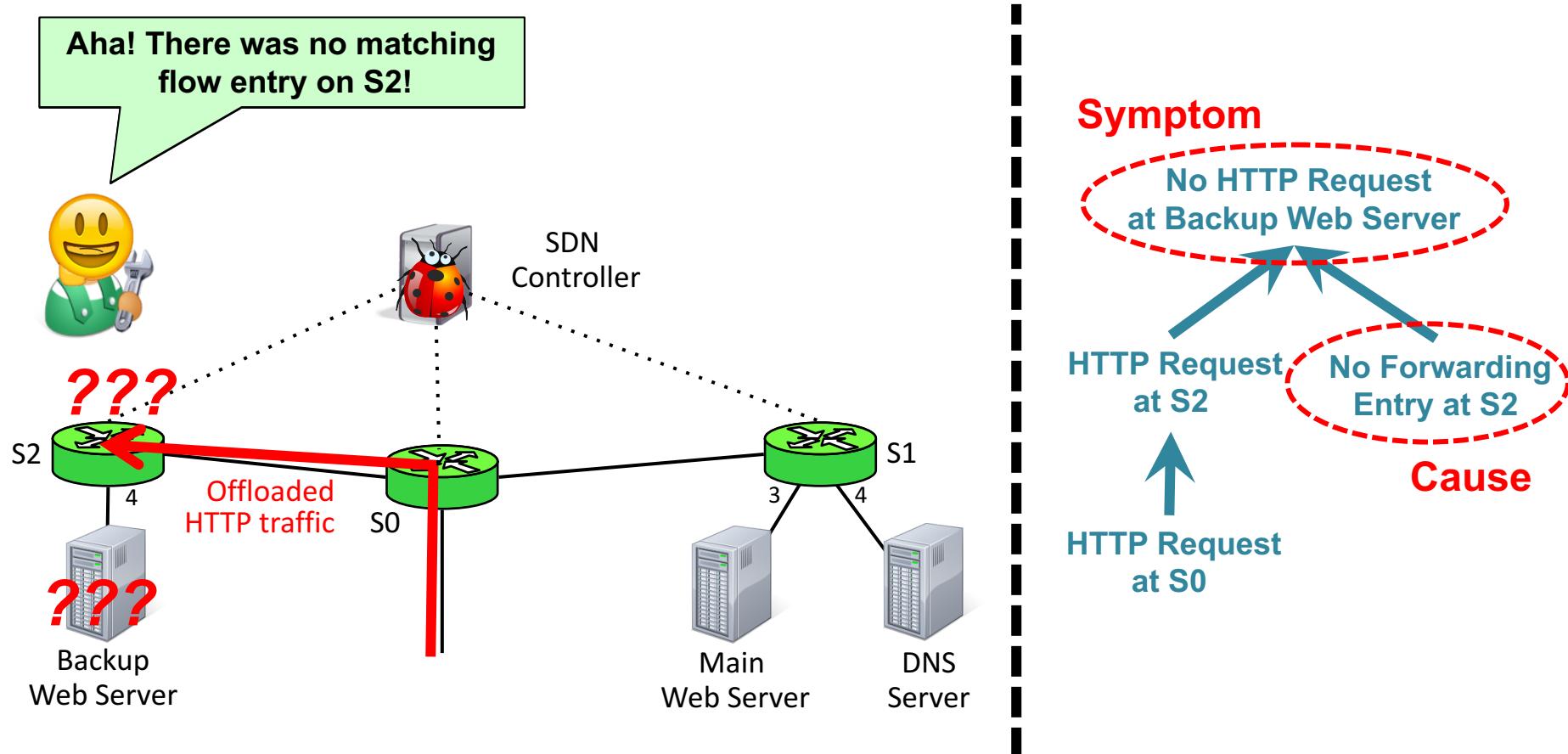
ATPG, DiffProv, EverFlow, ExSPAN, HSA, MCS, NetSight, OFFRewind, SNP, VeriFlow, Y!, etc.

## Step 2. Repair

?

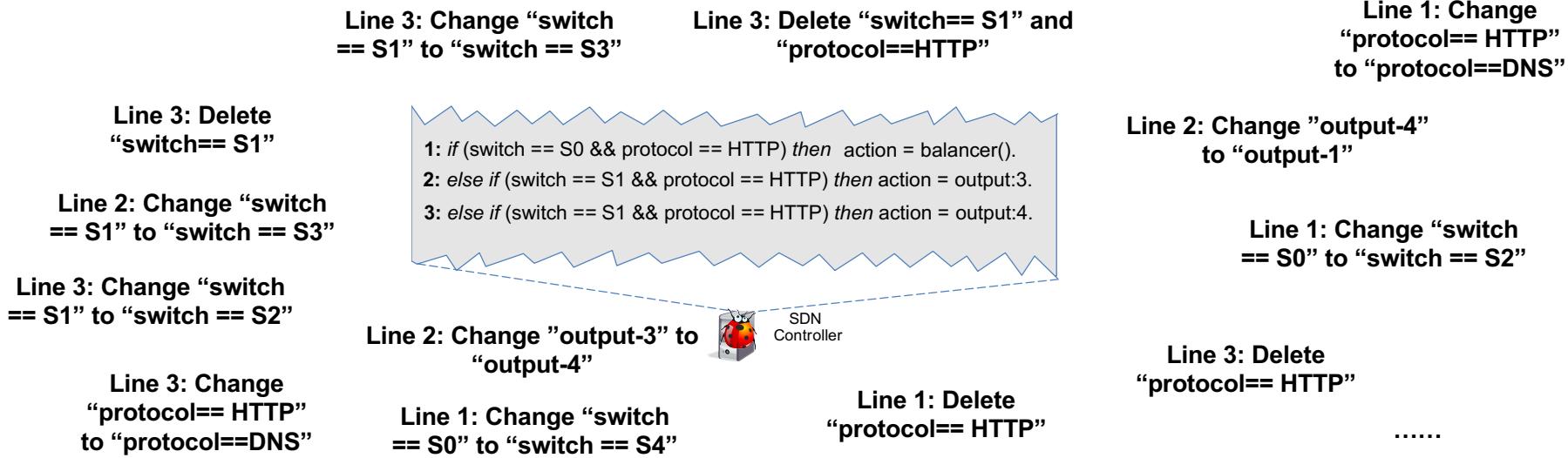
- Diagnosis is hard; but there are tools that can help
- Existing tools do not help much with repair

# What will existing debuggers do?



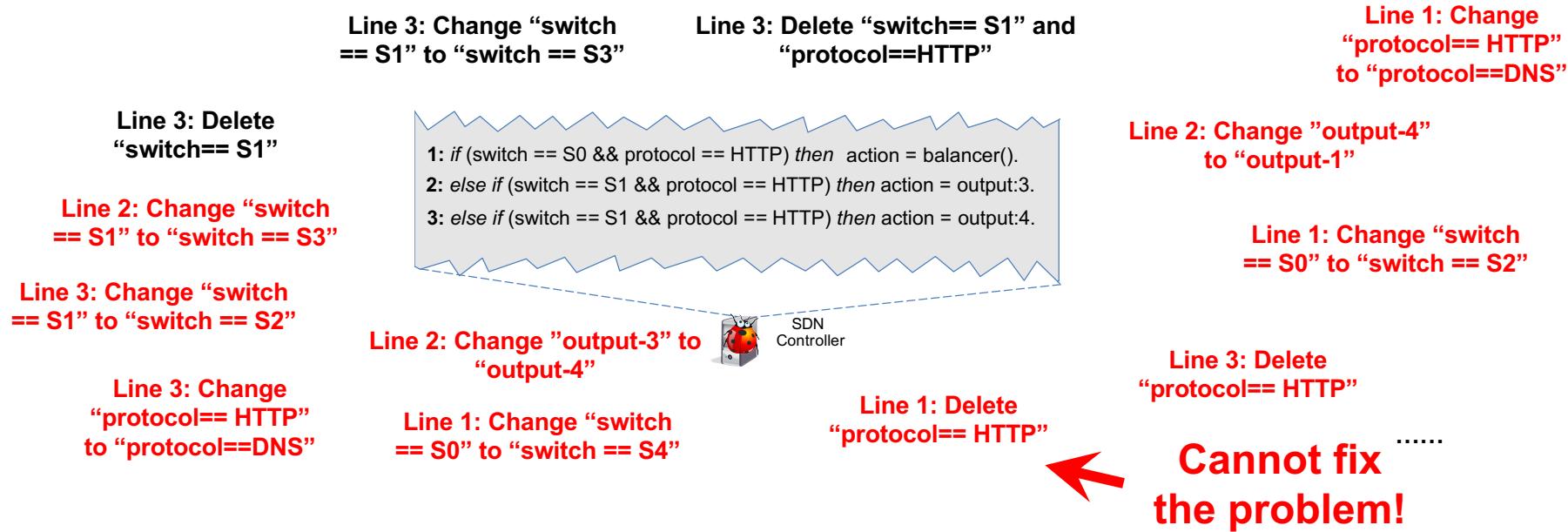
- Example: Y! [SIGCOMM '14]
- Provides a causal explanation of any event of interest

# How to repair the problem?



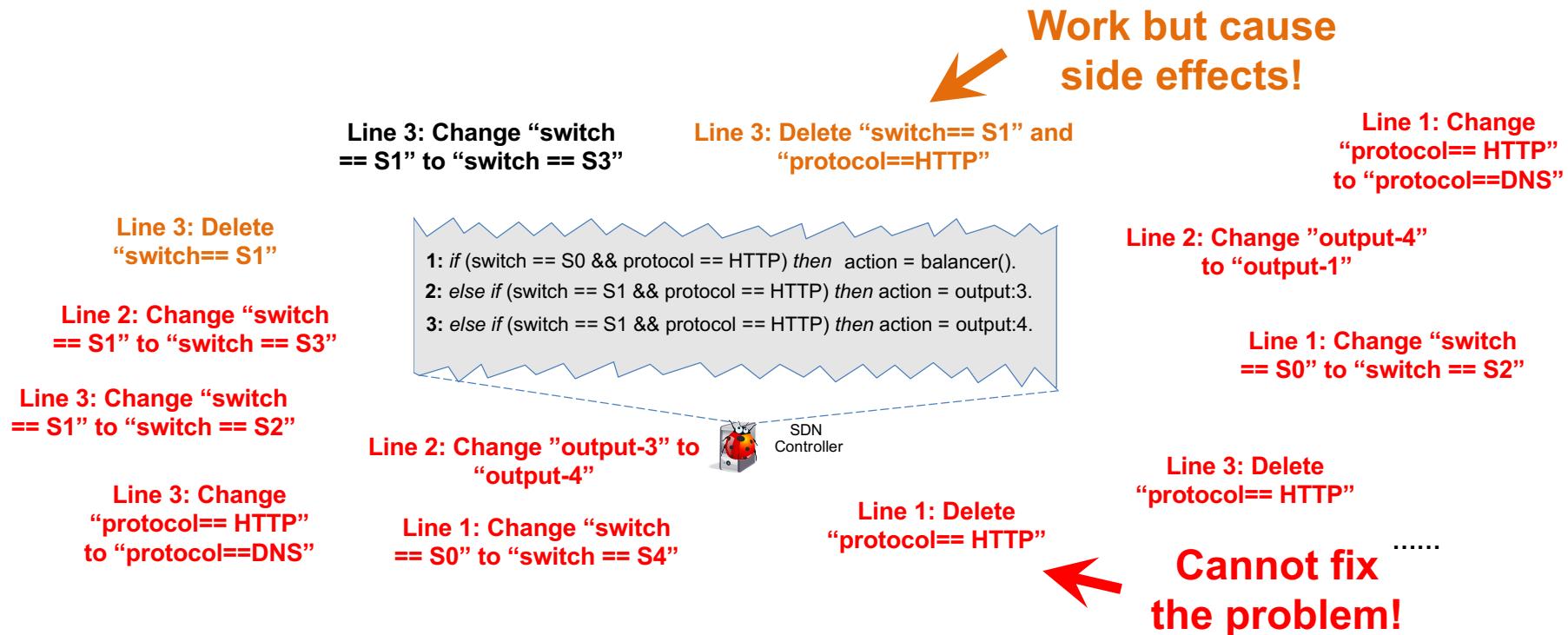
- Problem: Many ways to change a buggy program

# How to repair the problem?



- Problem: Many ways to change a buggy program
  - Most random changes do not fix the problem

# How to repair the problem?



- Problem: Many ways to change a buggy program
  - Most random changes do not fix the problem
  - Changes that do work can have undesirable side effects

# How to repair the problem?

Precisely fix  
the problem!



Line 3: Change "switch == S1" to "switch == S3"

Line 3: Delete "switch== S1"

Line 2: Change "switch == S1" to "switch == S3"

Line 3: Change "switch == S1" to "switch == S2"

Line 3: Change "protocol== HTTP" to "protocol==DNS"

```
1: if (switch == S0 && protocol == HTTP) then action = balancer().  
2: else if (switch == S1 && protocol == HTTP) then action = output:3.  
3: else if (switch == S1 && protocol == HTTP) then action = output:4.
```



Line 2: Change "output-3" to "output-4"

Line 1: Change "switch == S0" to "switch == S4"

Work but cause  
side effects!

Line 3: Delete "switch== S1" and "protocol==HTTP"

Line 1: Change "protocol== HTTP" to "protocol==DNS"

Line 2: Change "output-4" to "output-1"

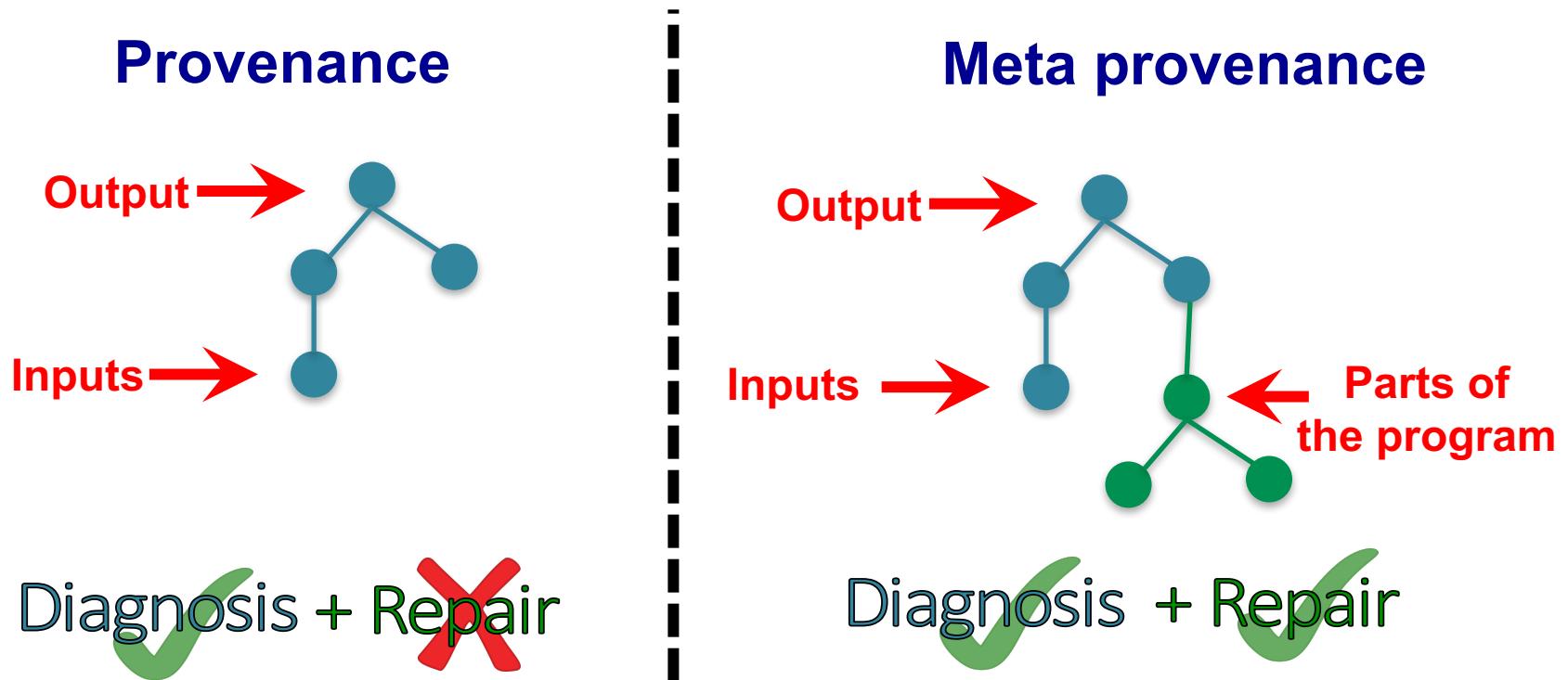
Line 1: Change "switch == S0" to "switch == S2"

Line 3: Delete "protocol== HTTP"

Cannot fix  
the problem!

- Problem: Many ways to change a buggy program
  - Most random changes do not fix the problem
  - Changes that do work can have undesirable side effects
- Goal: Finding a small number of targeted repairs!

# Our approach: Meta provenance

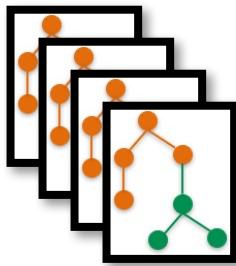


- Provenance tracks causality from inputs to outputs
- Meta provenance additionally tracks causality from program elements to outputs
- It tells us where to look for effective program changes!

# Putting it all together

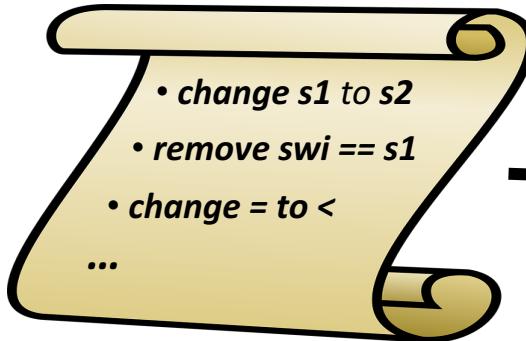


## Step #1: Generate meta provenance



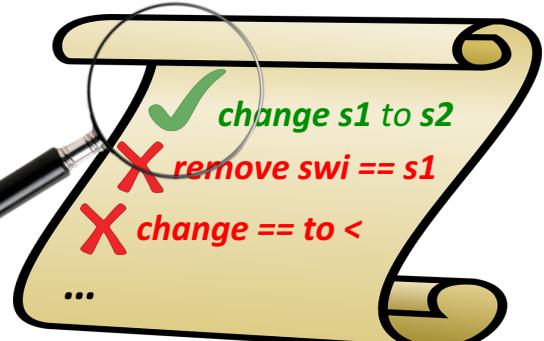
What should be changed?

## Step #2: Extract candidate repairs



What should it be changed to?

## Step #3: Backtest repairs



How to avoid side effects!



Input: The symptom

No HTTP Request  
at Backup Web Server



I can fix this by changing S1 in the controller program to S2!

## Overview

- {  Goal: Automated repair
-  Challenge: Finding effective program changes
-  Approach: Meta provenance

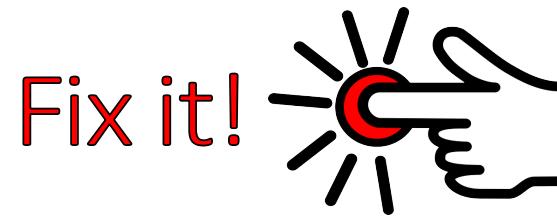
## Approach

- { Generating meta provenance
- Extracting repairs
- Backtesting repairs

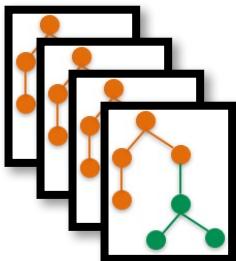
## Evaluation

- { Implementation
- Experimental setup
- Results

# Outline



## Step #1: Generate meta provenance



What should be changed?

## Step #2: Extract candidate repairs

- *change s1 to s2*
- *remove swi == s1*
- *change = to <*
- ...

What should it be changed to?

## Step #3: Backtest repairs

- ✓ *change s1 to s2*
- ✗ *remove swi == s1*
- ✗ *change == to <*
- ...

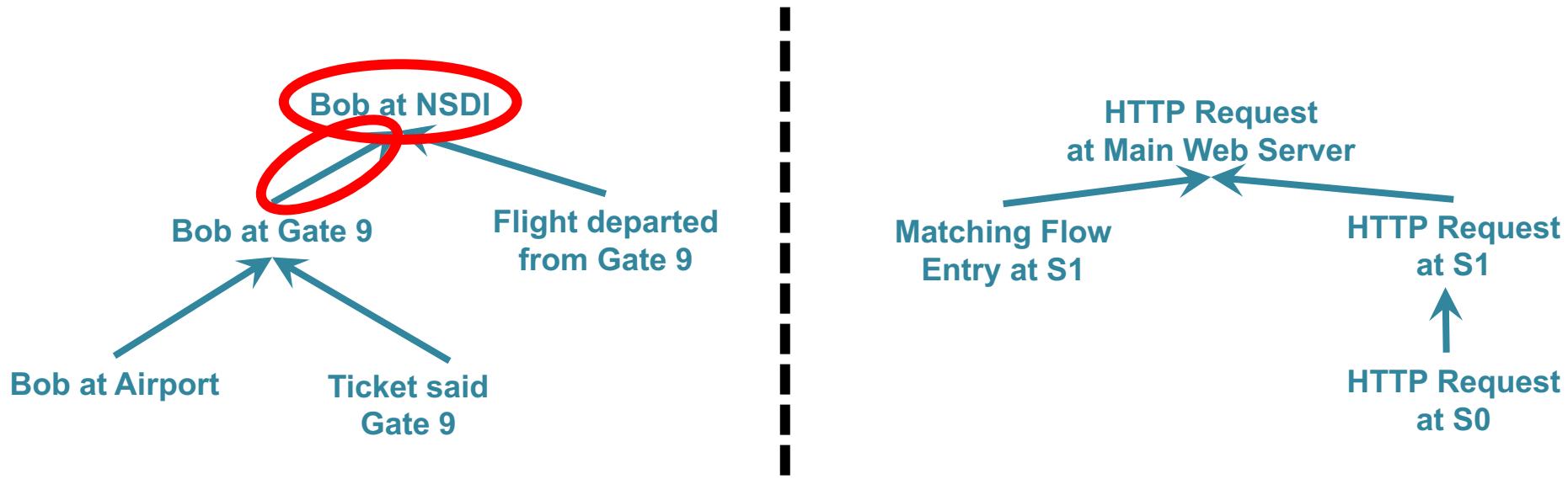
How to avoid side effects!



**Input: The symptom**

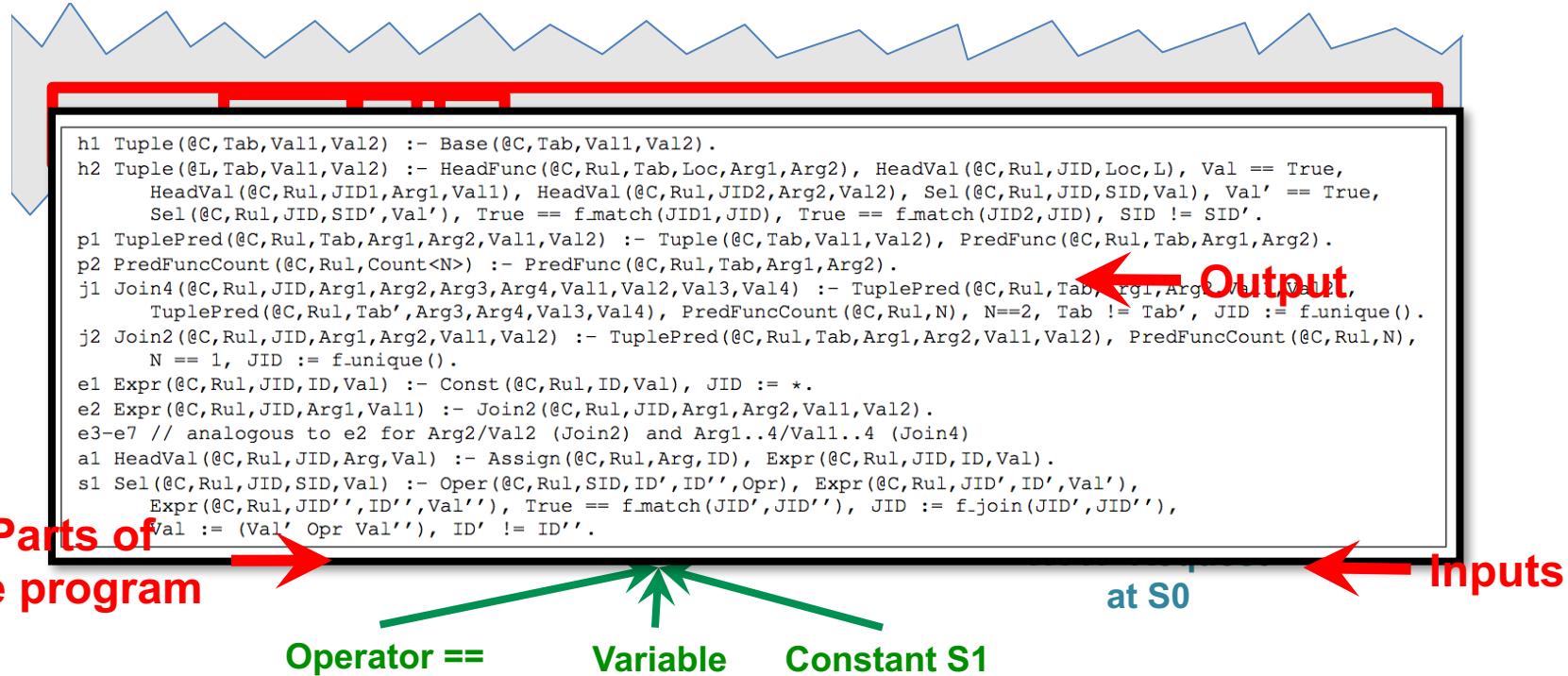
No HTTP Request  
at Backup Web Server

# Background: Provenance



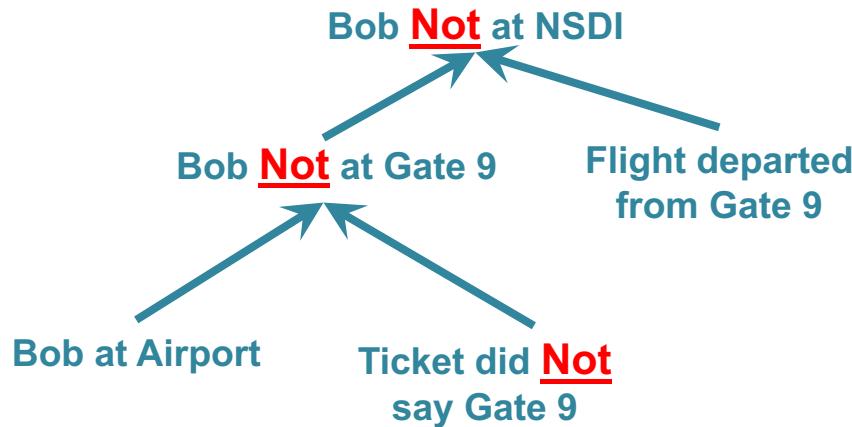
- Provenance: How outputs are derived from inputs
  - Vertices represent events; edges represent causal relationships
- Good: Includes only inputs that actually matter
- Bad: The program does not appear in the provenance!

# Meta provenance



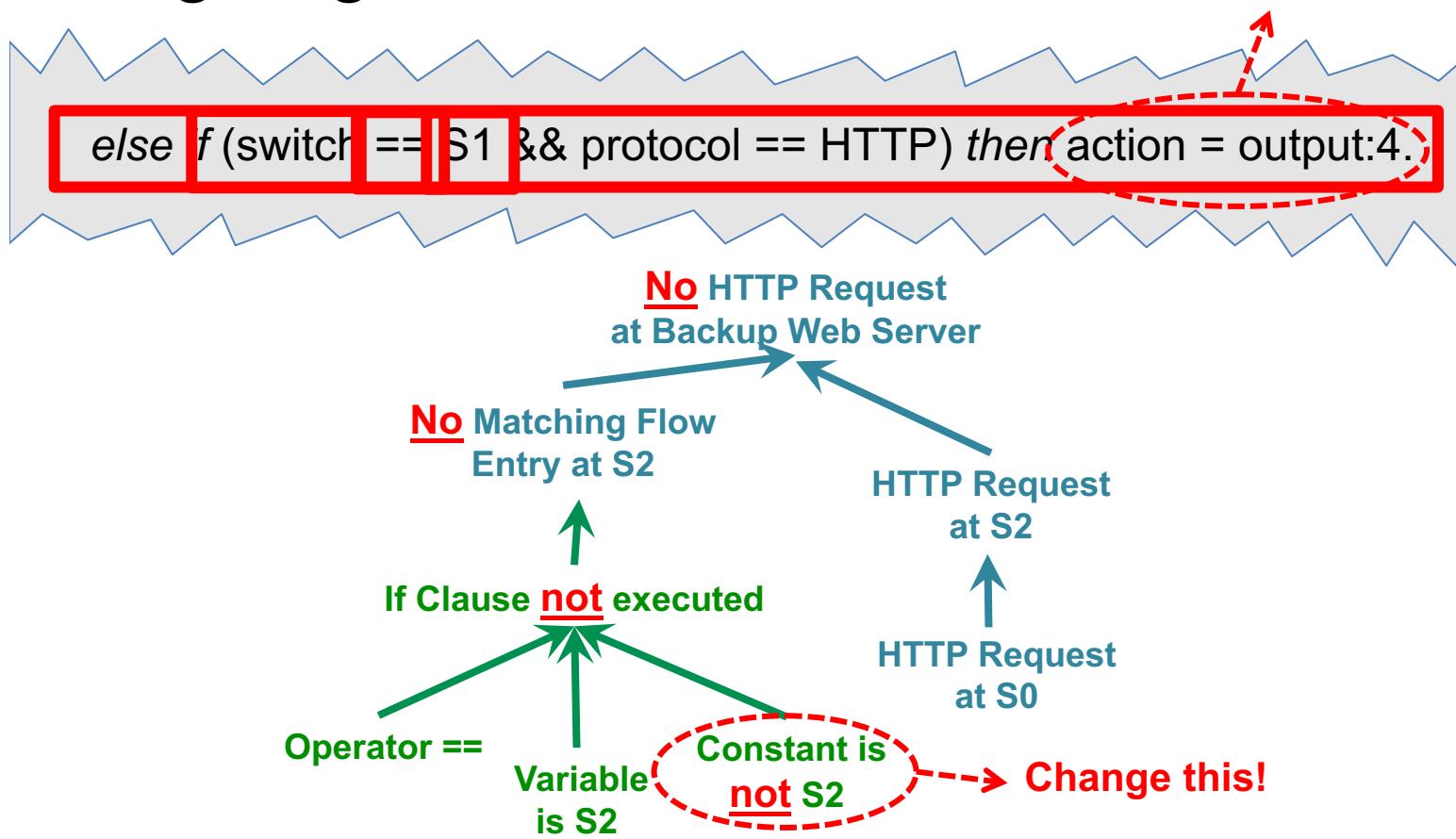
- Idea: Build a meta model that describes the semantics of the programming language
- Apply provenance at the meta level
- Result: The resulting meta provenance covers both inputs and program elements

# Handling negative events



- Problem: What if something failed to happen?
- This can be handled with counter-factual reasoning

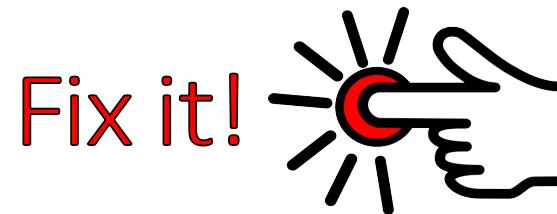
# Handling negative events



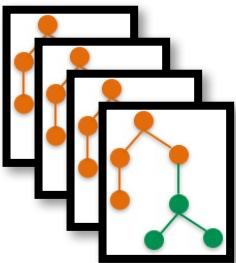
- How does this work in our example?

Meta provenance finds the program elements that can be changed to affect the observed symptom!

# Outline



## Step #1: Generate meta provenance



What should  
be changed?

## Step #2: Extract candidate repairs

- *change s1 to s2*
- *remove swi == s1*
- *change = to <*
- ...

What should it be  
changed to?

## Step #3: Backtest repairs

- ✓ *change s1 to s2*
- ✗ *remove swi == s1*
- ✗ *change == to <*
- ...

How to avoid  
side effects!

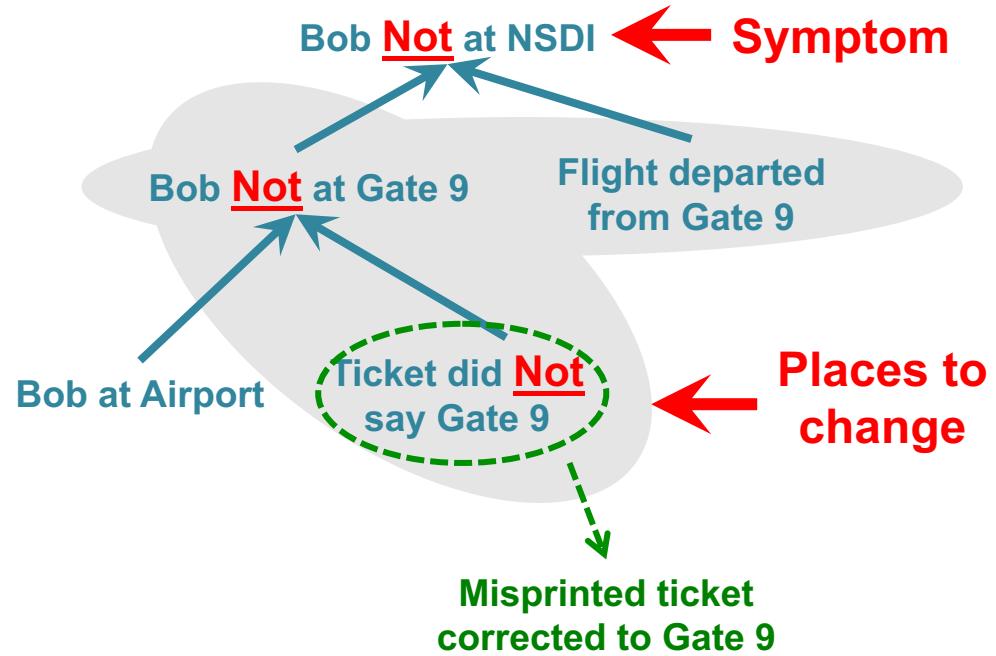


**Input: The symptom**

No HTTP Request  
at Backup Web Server

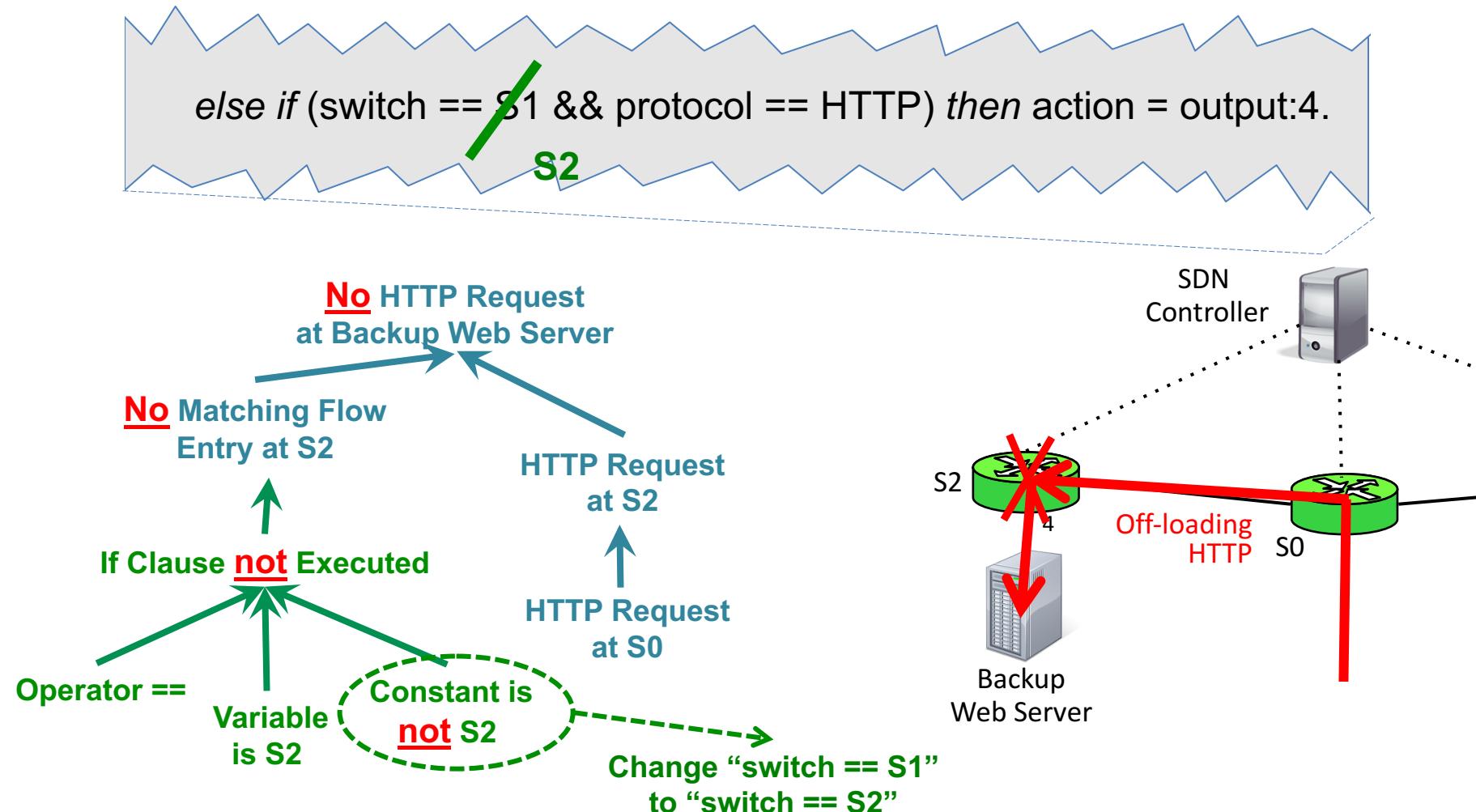
# Extracting candidate repairs

Constraints

$$\text{Flight.Gate}(9) == \text{Bob.Gate}$$
$$\text{Bob.Gate} == \text{Ticket.Gate}$$


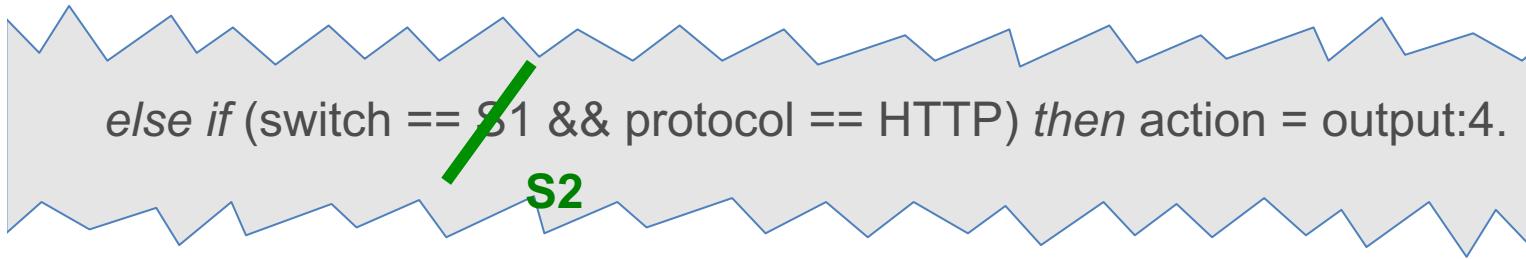
- We can change the root by changing leaves
- But which values will have the desired effect?
- Idea: Extract a set of constraints from the tree
  - The actual constraints are more complicated
  - Constraints can be given to a solver to find suitable values

# Extracting candidate repairs



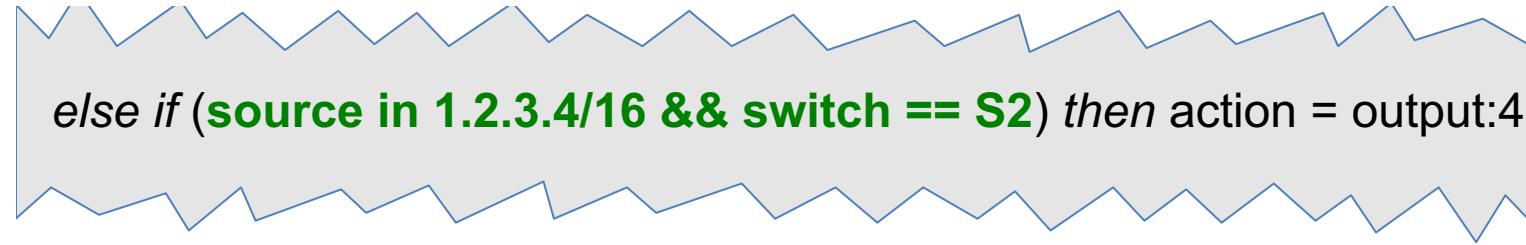
Meta provenance generates targeted repairs!

# Which repairs should we change?



*else if(switch == S1 && protocol == HTTP) then action = output:4.*

**S2**



*else if(**source in 1.2.3.4/16** && switch == S2) then action = output:4.*

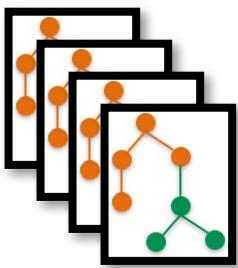


- Complex repairs are unlikely to be useful in practice
  - About **9%** of all semantic bugs are typos [Li et. al, ASID 2006]
  - Only **10-30%** of bugs fixes create new statements [Pan et. al, ESE 2009]
- Idea: Rank repairs by complexity

# Outline

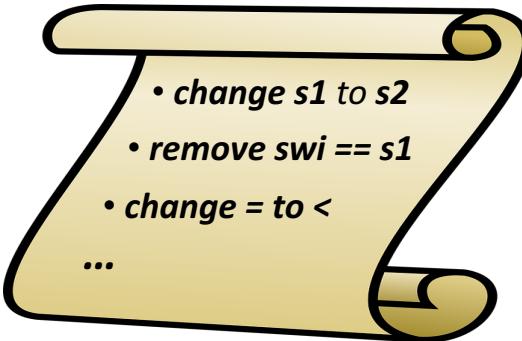


## Step #1: Generate meta provenance



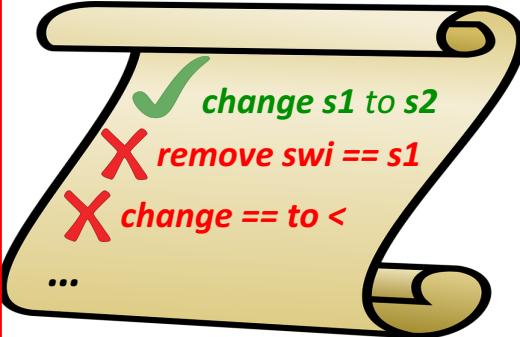
What should  
be changed?

## Step 2: Extract candidate repairs



What should it be  
changed to?

## Step #3: Backtest repairs



How to avoid  
side effects!



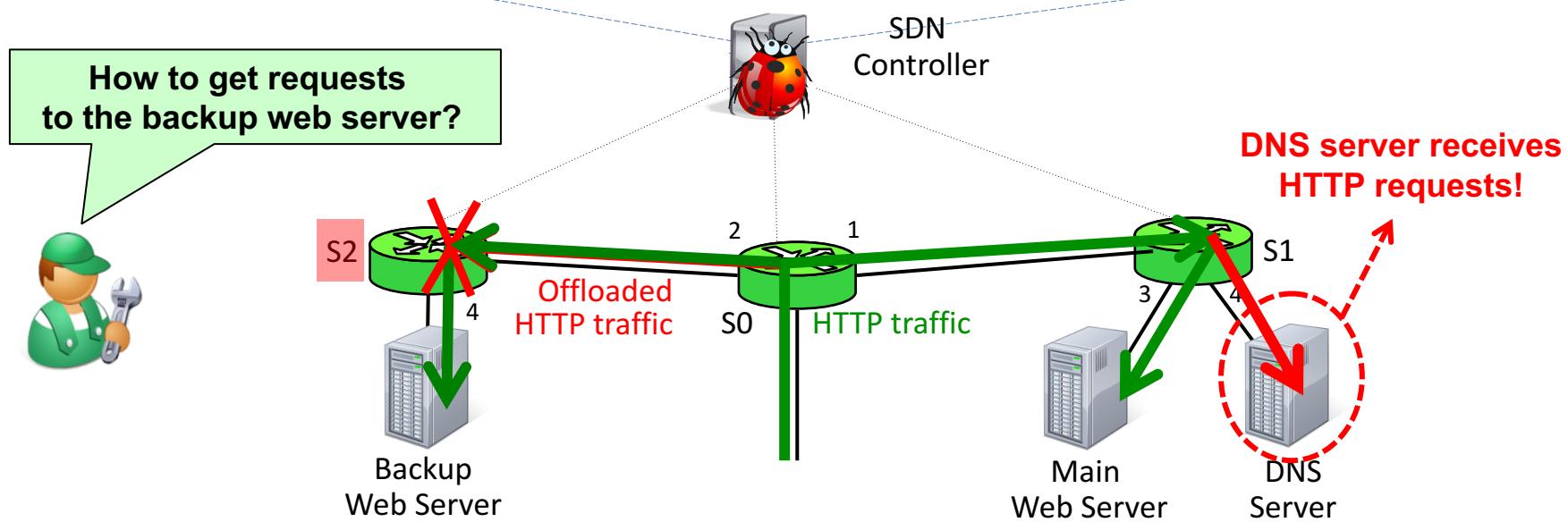
**Input: The symptom**

No HTTP Request  
at Backup Web Server

# Challenge: Undesired side effects

~~else if (switch == S1 && protocol == HTTP) then~~ action = output:4.

# Copy-and-paste bug!!!



- Idea: Backtest repairs using historical data
    - Replay historical data on the repaired network
    - Prioritize repairs that do not significantly disrupt traffic distributions
    - Idea: Efficient backtesting using multi-query optimization

## Overview

- ✓ Goal: Automated repair
- ✓ Challenge: Finding effective program changes
- ✓ Approach: Meta provenance

## Approach

- ✓ Generating meta provenance
- ✓ Extracting repairs
- ✓ Backtesting repairs

## Evaluation

- Implementation
- Experimental setup
- Results

# Implementation and experimental setup

We support three languages (NDlog, Pyretic, Trema)



We used five SDN bugs from other papers

- Copy-and-paste error [OSDI 2004]
- Forwarding error [CoNEXT 2012]
- Uncoordinated policy update [CoNEXT 2014]
- Forgotten packets [NSDI 2012]
- Incorrect MAC learning [HotSDN 2014]

Augmented Stanford campus topology with 16~269 switches

A mix of reactive traffic and background traffic

Meta provenance is extracted automatically

# Evaluation: Questions

Does the algorithm generate reasonable fixes?

What is the runtime overhead?

How much time do diagnostic queries take?

Does meta provenance scale with network size?

How easy is it to add a new language?

# Evaluation: Quality of the generated repairs

- Example: copy-and-paste error

A: Manually install a flow entry

B: Change Swi == 2 in r7 to Swi == 3

C: ~~Change Swi == 2 in r7 to Swi != 2~~

D: ~~Change Swi == 2 in r7 to Swi >= 2~~

E: ~~Change Swi == 2 in r7 to Swi > 2~~

F: ~~Delete Swi == 2 in r7~~

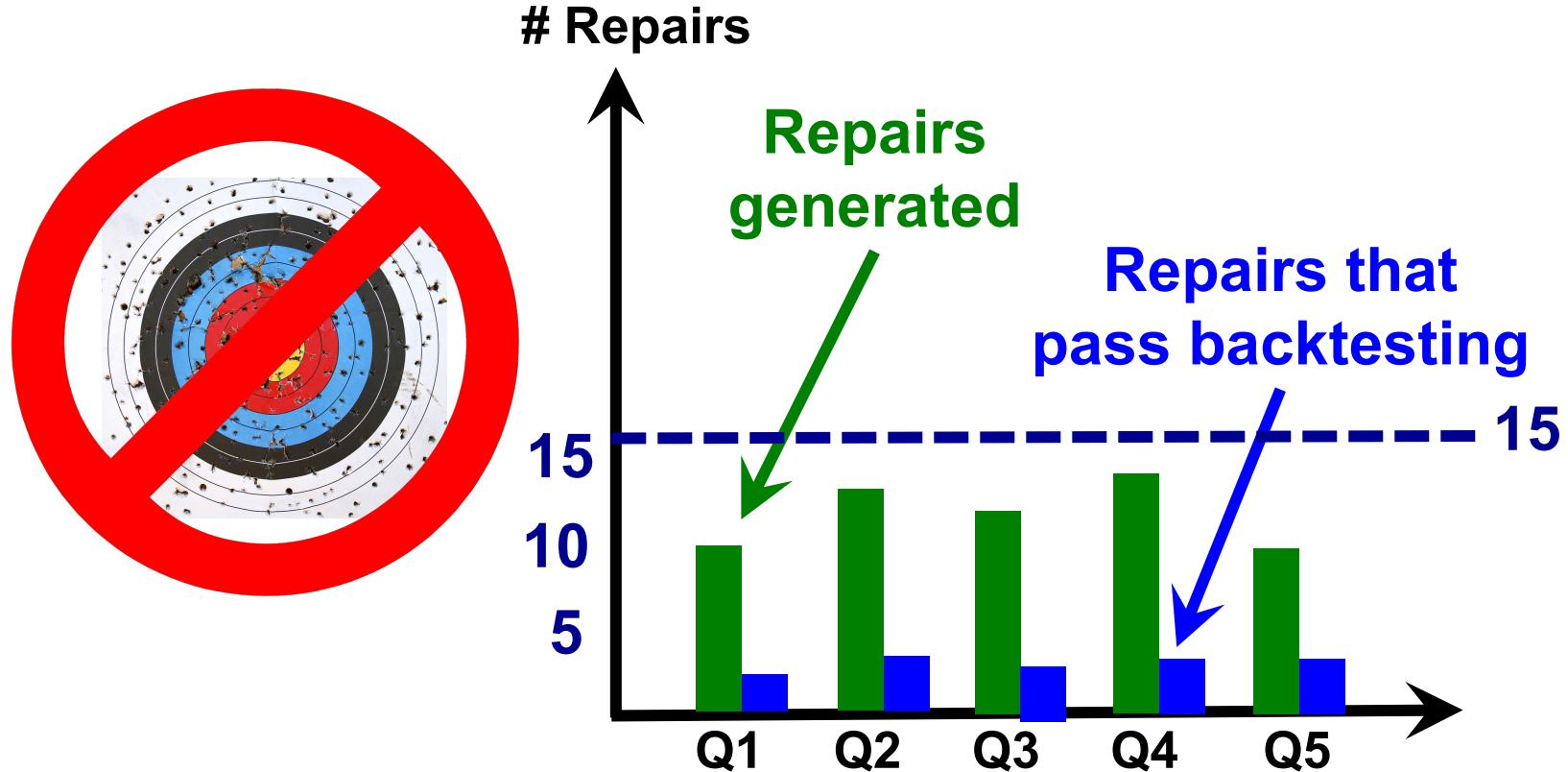
G: ~~Delete Swi == 2 and Dpt == 53 in r6~~

H: ~~Delete Swi == 2 and Dpt == 80 in r7~~

I: ~~Change Swi == 2 and Act=output-1 in r5 to Swi == 3 and Act=output-2~~

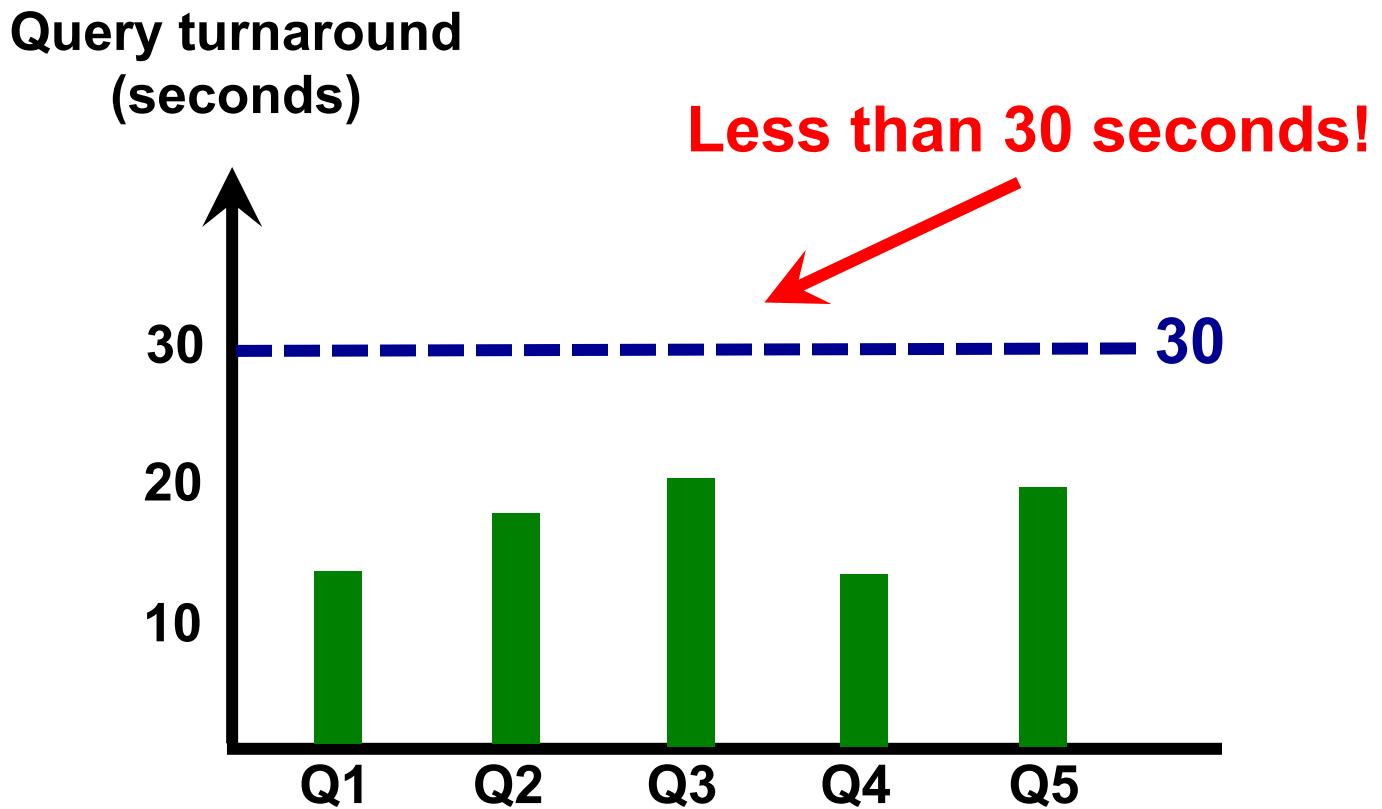
Meta provenance generates targeted repairs!

# Evaluation: Quality of the generated repairs



Meta provenance generates targeted repairs!

# Evaluation: Time to answer queries



Fast enough for interactive use.



# Summary

- **Goal: Automated repairs for buggy SDN programs**
  - Ideally, an automatic “Fix it!” button
  - Challenge: Finding program changes that are effective and avoid side effects
- **Approach: Meta Provenance**
  - A generalization of data provenance
  - Finds the data and code that are causally connected to a given event
  - Can be used to efficiently find program changes that will have an effect on an observed problem
  - Backtesting can be used to avoid changes that have undesirable side effects.
- **Evaluation with three different SDN languages**

Repairs are effective and can be found quickly

**Thank you!**