

# CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks

Naga Katta

Omid Alipourfard\*, Jennifer Rexford, David Walker

Princeton University, \*USC



PRINCETON  
UNIVERSITY

# Inadequate Switching Gear

TECHNOLOGY

## Echoes of Y2K: Engineers Buzz That Internet Is Outgrowing Its Gear

Routers That Send Data Online Could Become Overloaded as Number of Internet Routes Hits '512K'



By DREW FITZGERALD [CONNECT](#)

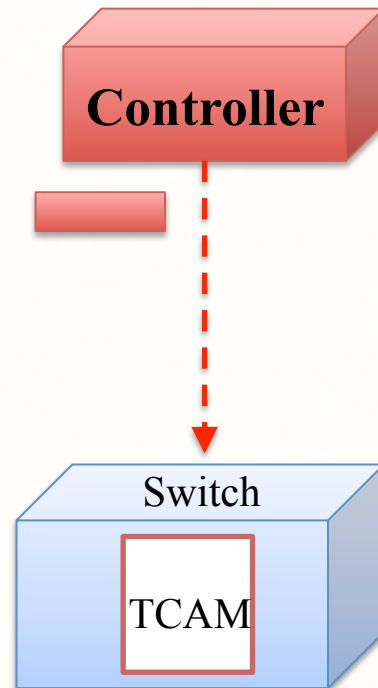
Updated Aug. 13, 2014 7:38 p.m. ET

Network engineers are buzzing this week as the Internet outgrows some of its gear.

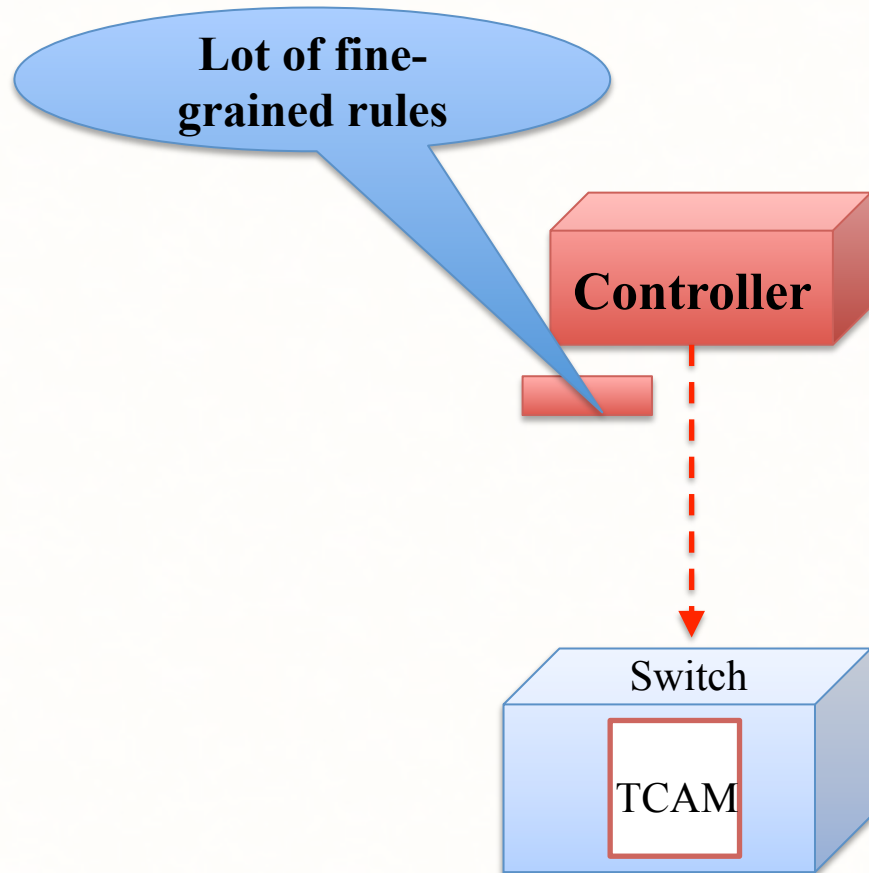
Internet providers, corporations and universities all rely on a common map of routes to send emails, videos and everything else on the Web where it's supposed to go.



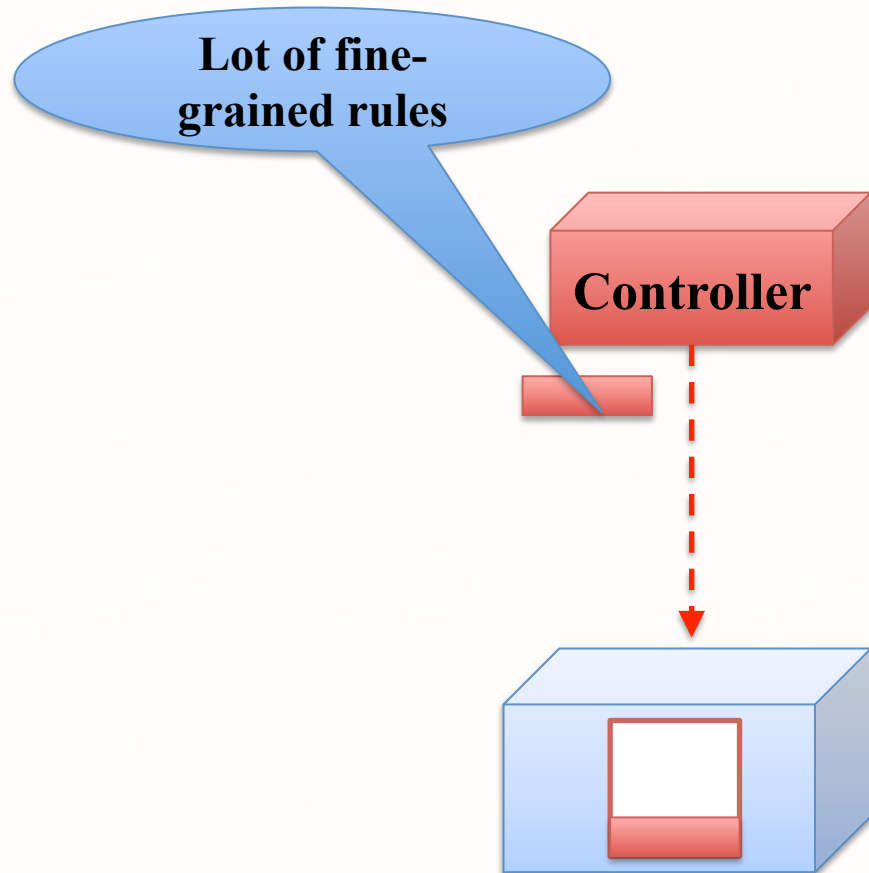
# SDN Promises Flexible Policies



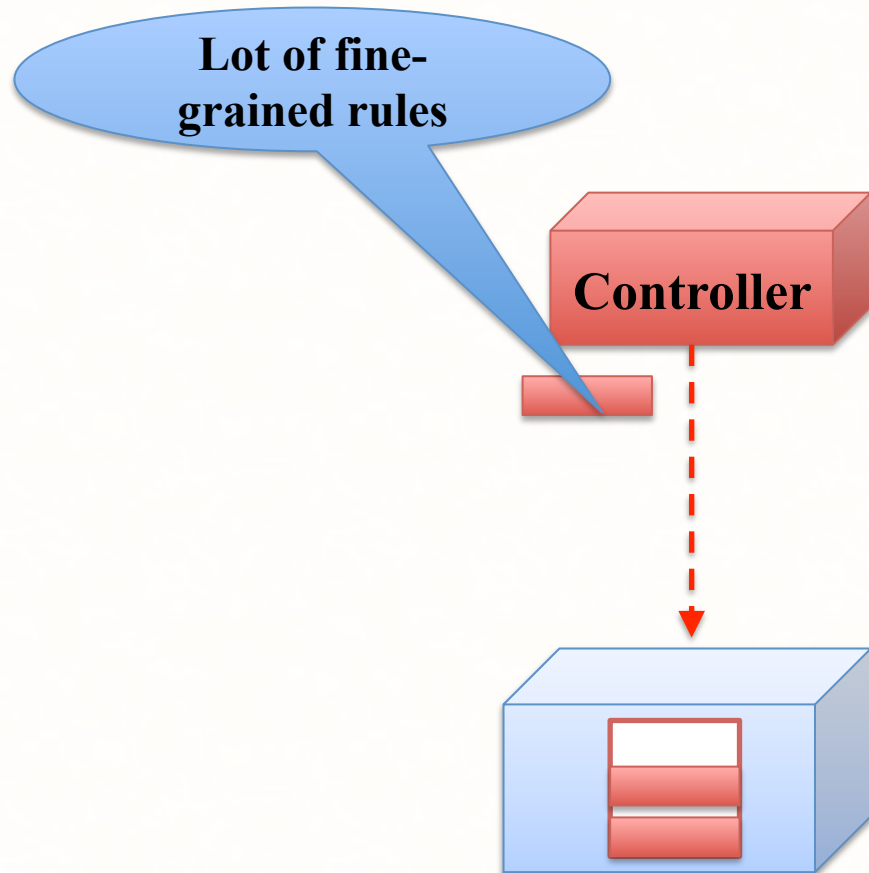
# SDN Promises Flexible Policies



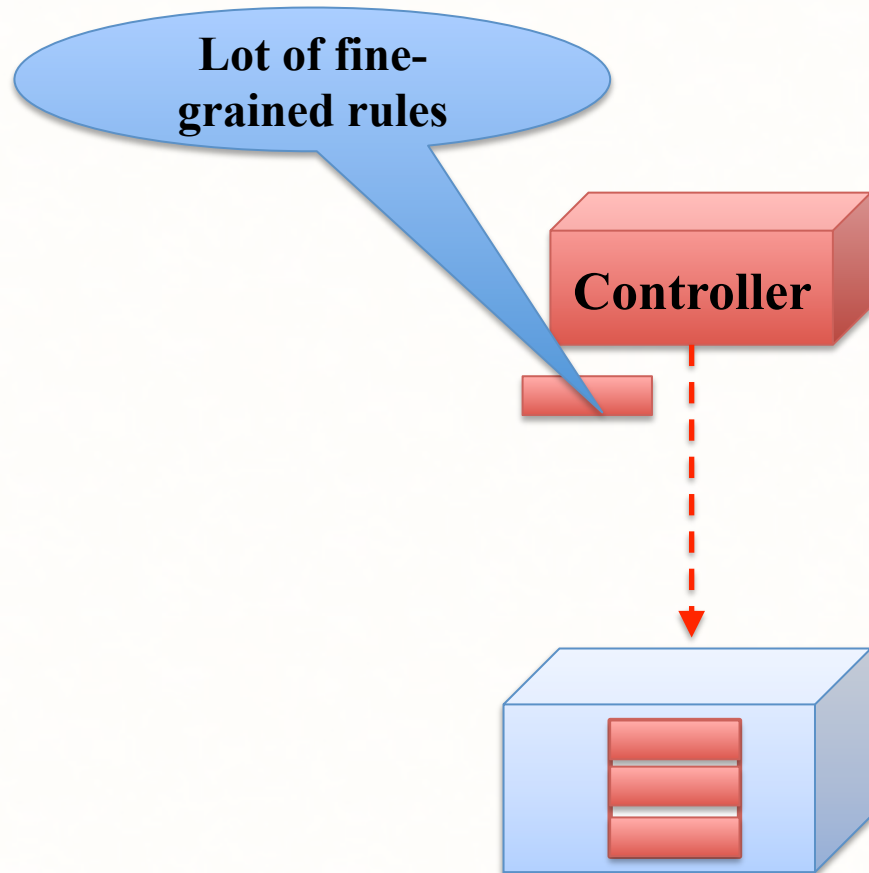
# SDN Promises Flexible Policies



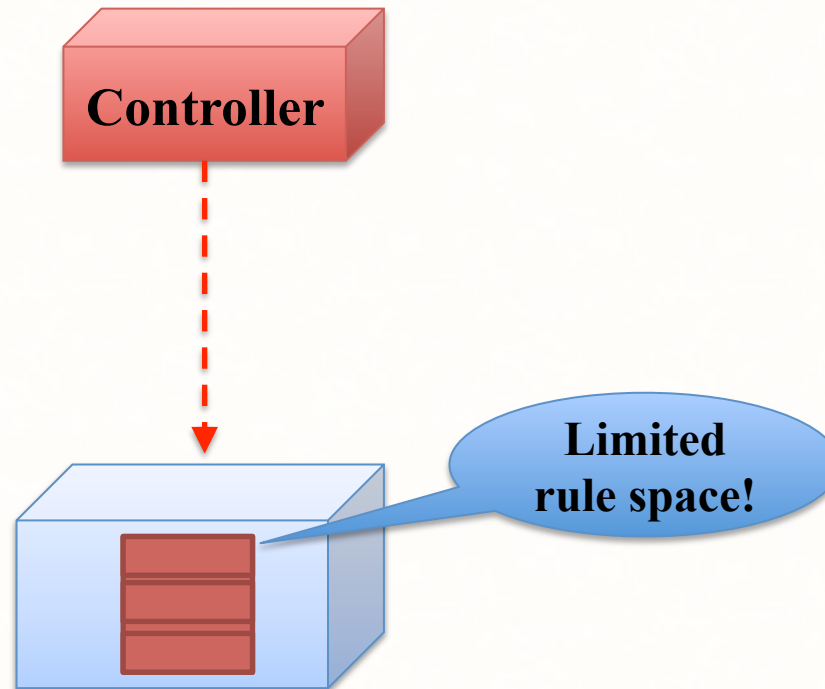
# SDN Promises Flexible Policies



# SDN Promises Flexible Policies

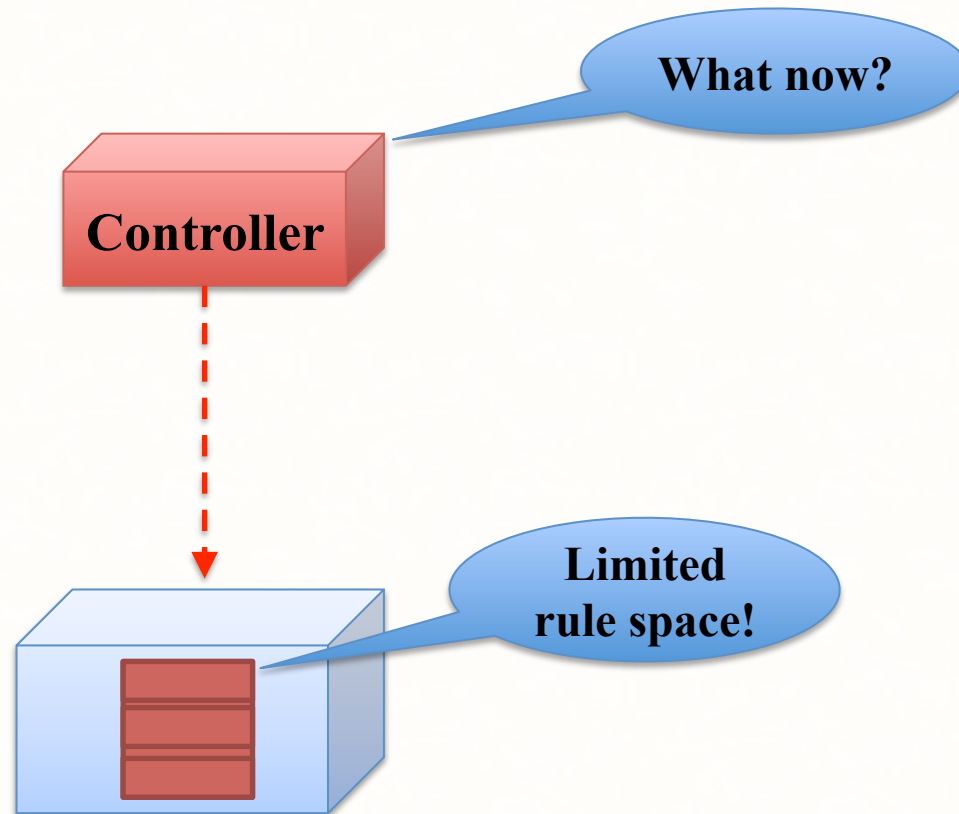


# SDN Promises Flexible Policies





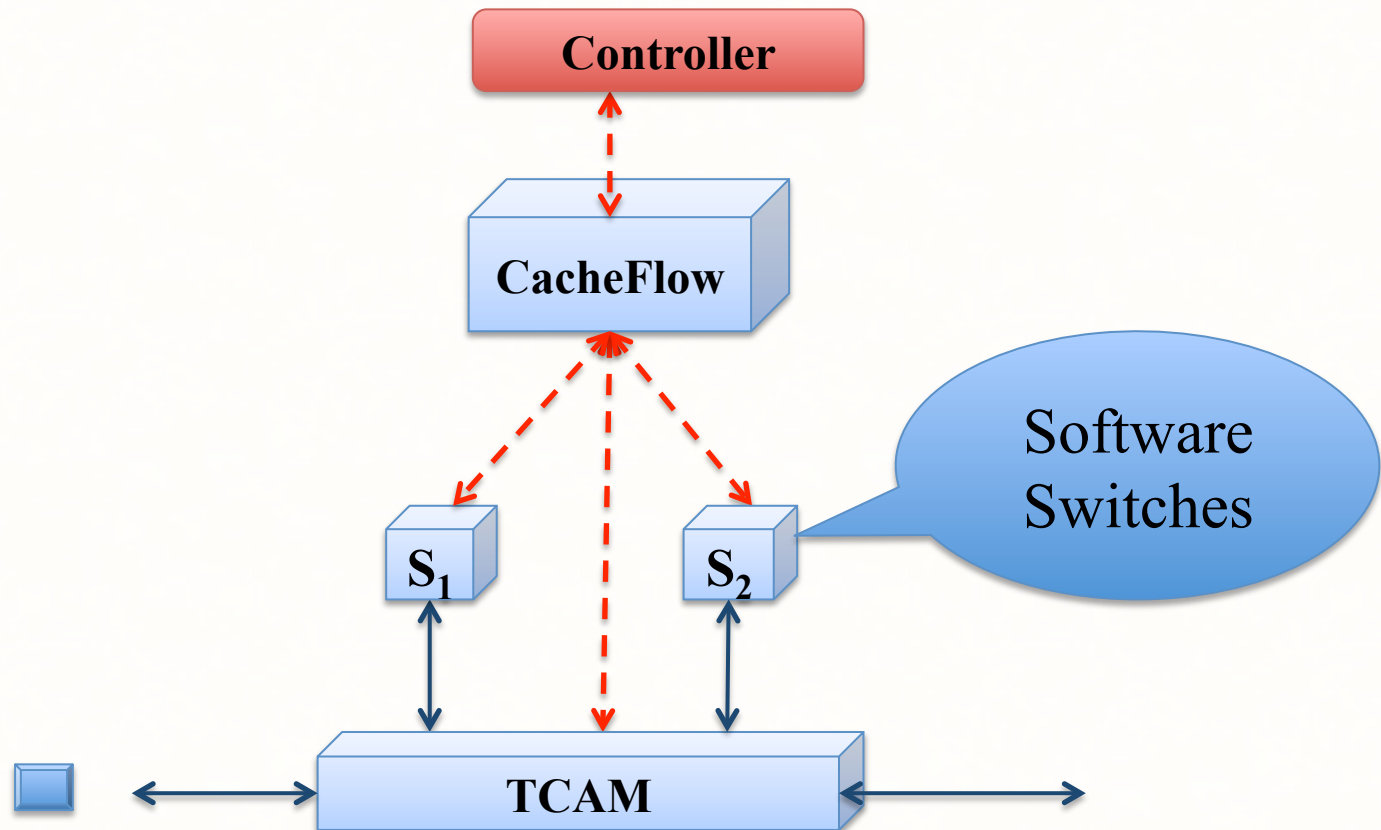
# SDN Promises Flexible Policies



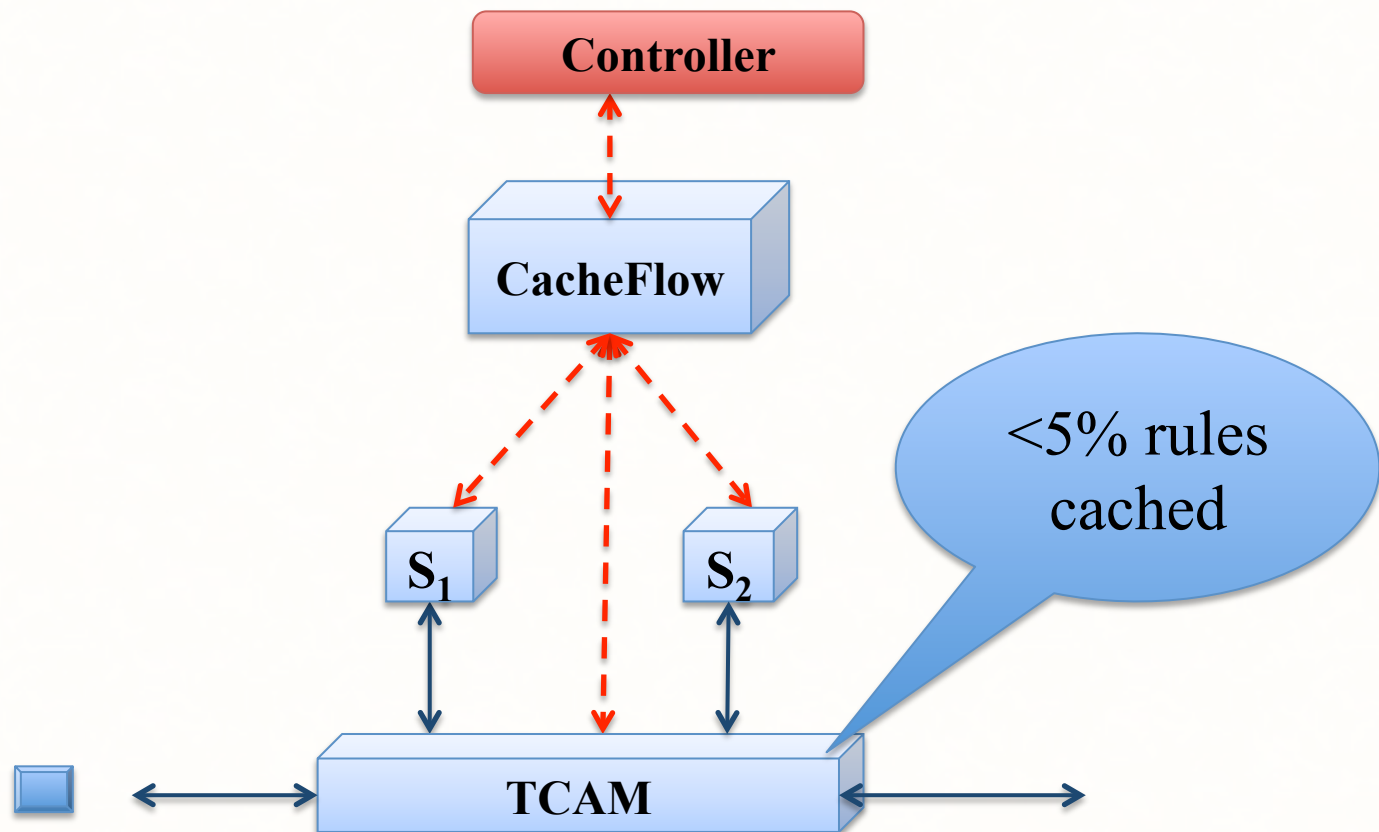
# State of the Art

|                          | Hardware Switch | Software Switch  |
|--------------------------|-----------------|------------------|
| <b>Rule Capacity</b>     | Low (~2K-10K)   | High             |
| <b>Lookup Throughput</b> | High (>400Gbps) | Low (~40Gbps)    |
| <b>Port Density</b>      | High            | Low              |
| <b>Cost</b>              | Expensive       | Relatively cheap |

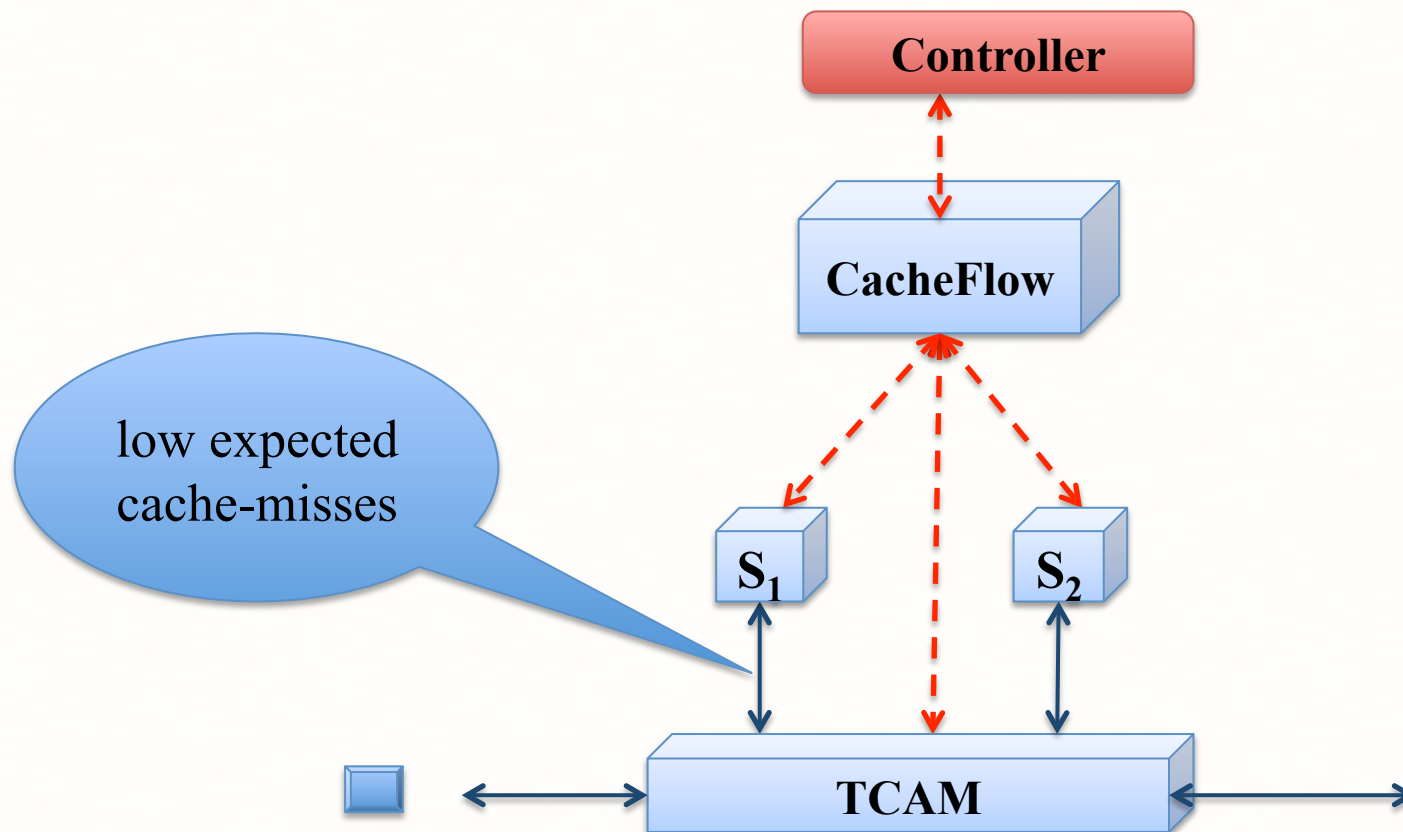
# TCAM as cache



# TCAM as cache

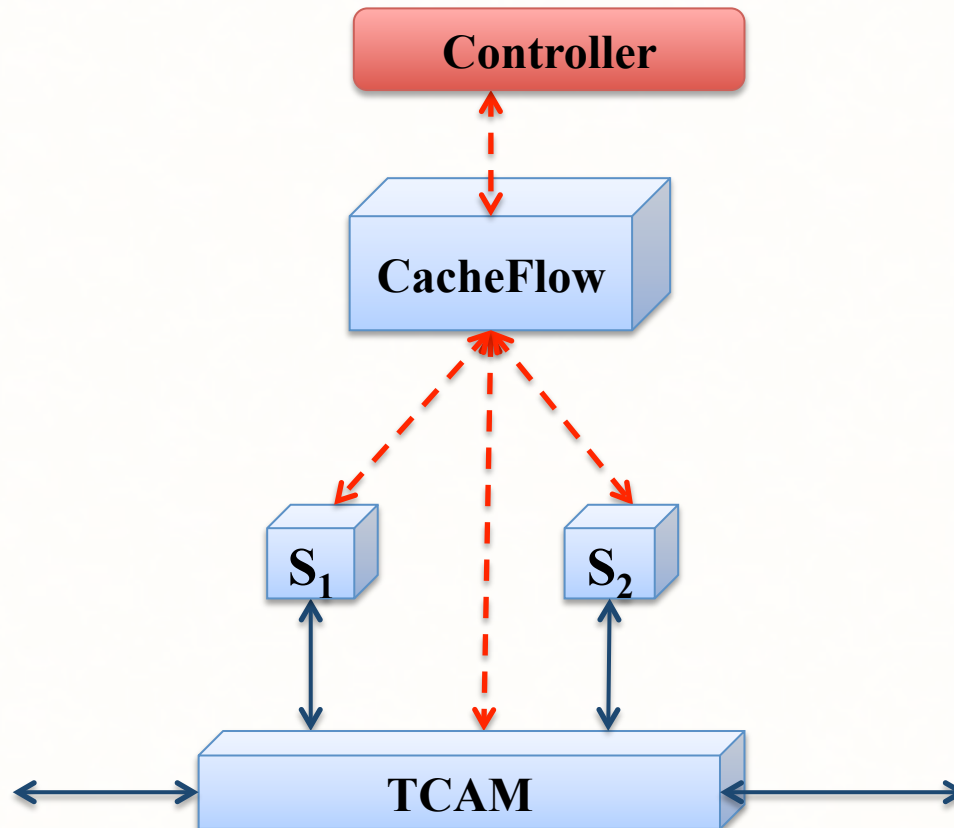


# TCAM as cache



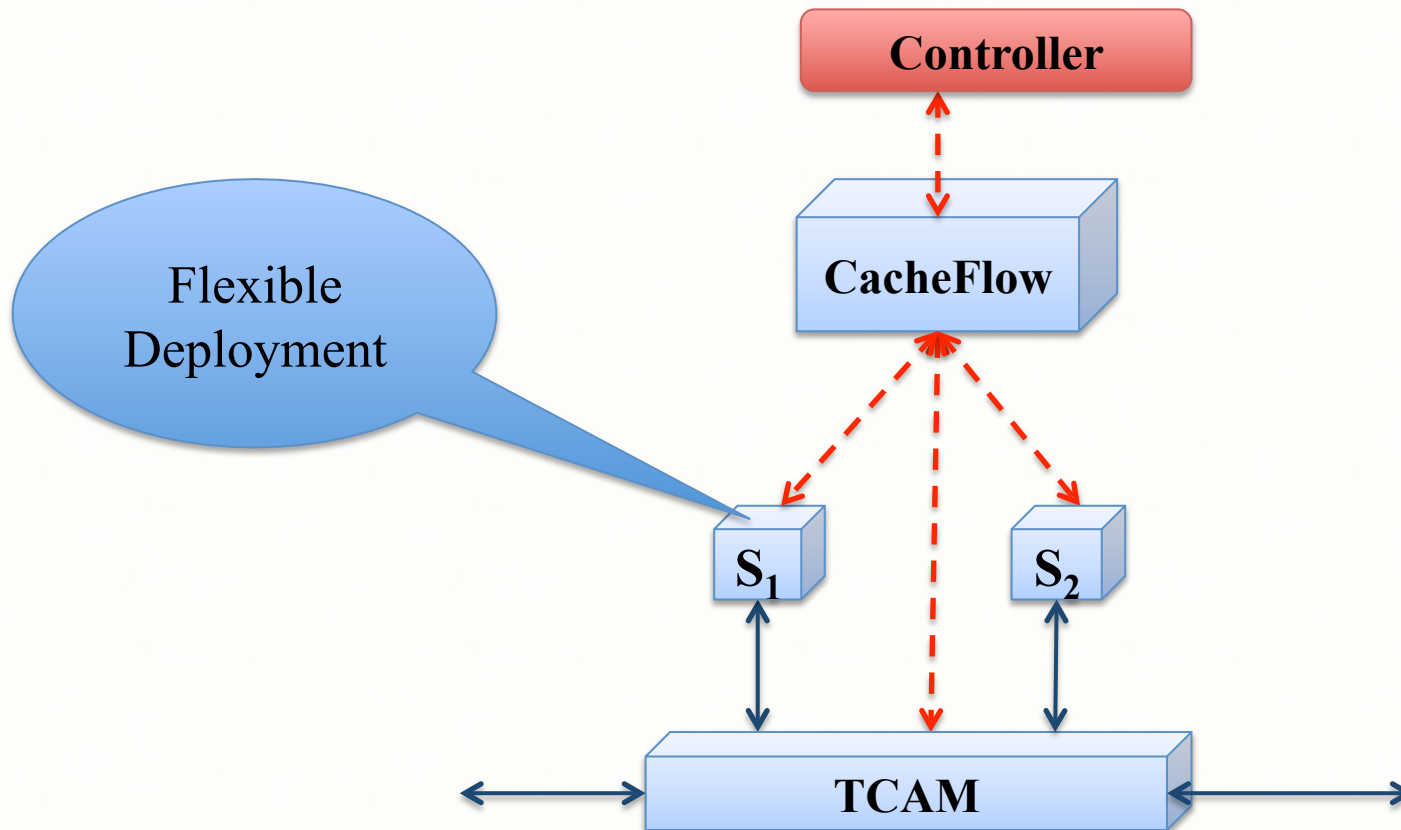
# TCAM as cache

- High throughput + high rule space



# TCAM as cache

- High throughput + high rule space



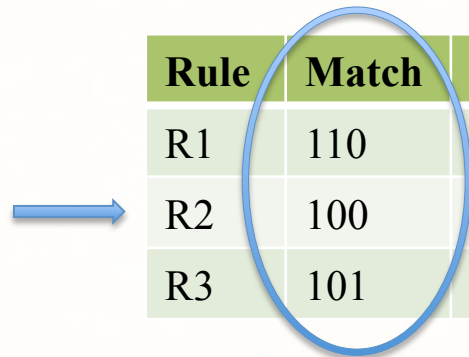
# A *Correct, Efficient* and *Transparent* Caching System

- Abstraction of an “infinite” switch
  - Correct: realizes the policy
  - Efficient: high throughput & large tables
  - Transparent: unmodified applications/switches



# 1. Correct Caching


# Caching under constraints



| Rule | Match | Action | Priority | Traffic |
|------|-------|--------|----------|---------|
| R1   | 110   | Fwd 1  | 3        | 10      |
| R2   | 100   | Fwd 2  | 2        | 60      |
| R3   | 101   | Fwd 3  | 1        | 30      |

Easy: Cache rules greedily

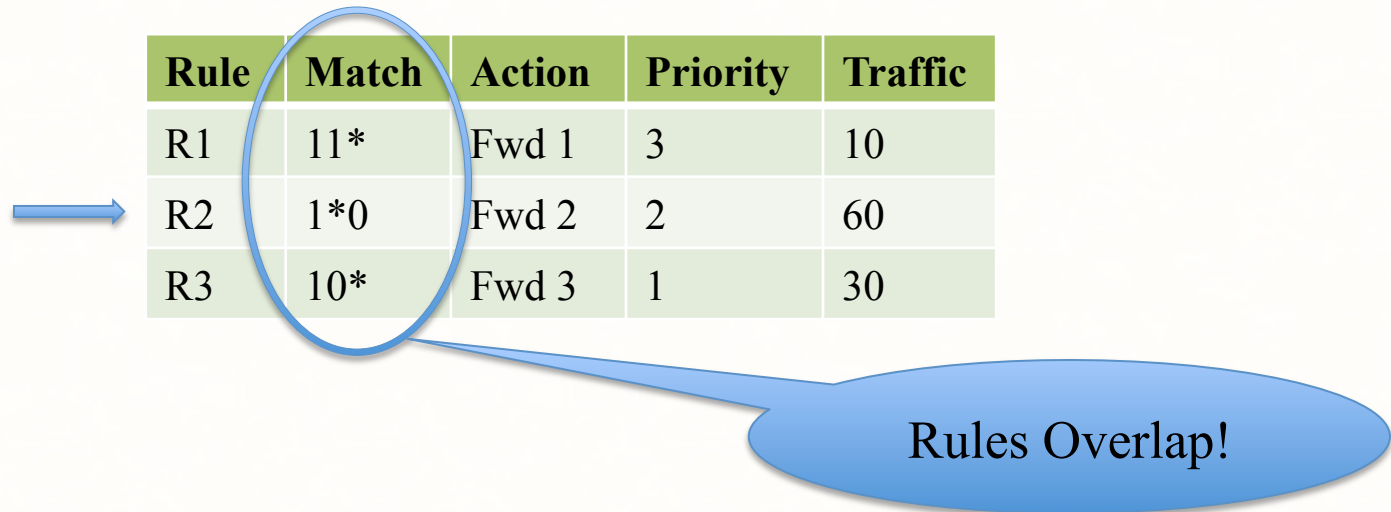
# Caching Ternary Rules



| Rule | Match | Action | Priority | Traffic |
|------|-------|--------|----------|---------|
| R1   | 11*   | Fwd 1  | 3        | 10      |
| R2   | 1*0   | Fwd 2  | 2        | 60      |
| R3   | 10*   | Fwd 3  | 1        | 30      |

- Greedy strategy breaks rule-table semantics

# Caching Ternary Rules

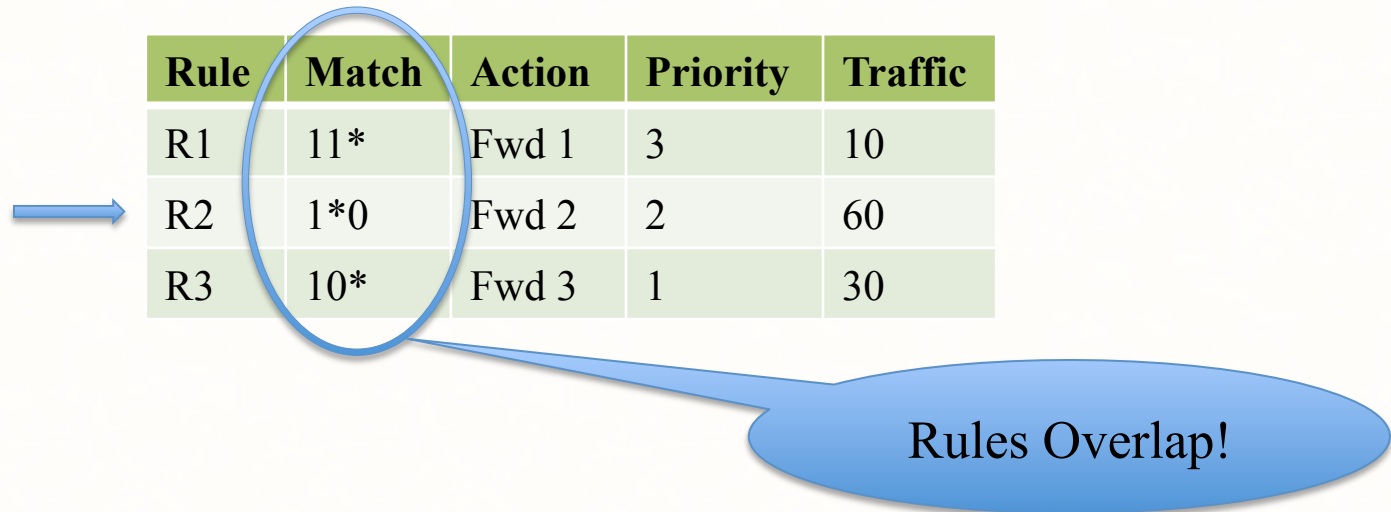


| Rule | Match | Action | Priority | Traffic |
|------|-------|--------|----------|---------|
| R1   | 11*   | Fwd 1  | 3        | 10      |
| R2   | 1*0   | Fwd 2  | 2        | 60      |
| R3   | 10*   | Fwd 3  | 1        | 30      |

Rules Overlap!

- Greedy strategy breaks rule-table semantics

# Caching Ternary Rules



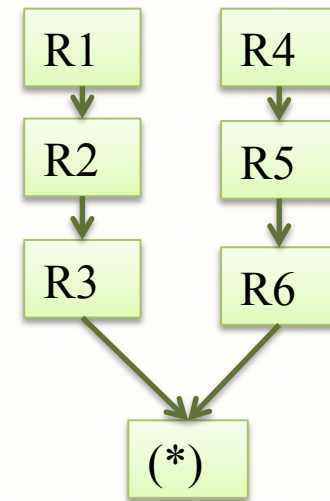
| Rule | Match | Action | Priority | Traffic |
|------|-------|--------|----------|---------|
| R1   | 11*   | Fwd 1  | 3        | 10      |
| R2   | 1*0   | Fwd 2  | 2        | 60      |
| R3   | 10*   | Fwd 3  | 1        | 30      |

Rules Overlap!

- Greedy strategy breaks rule-table semantics
- Beware of switches that claim large rule tables

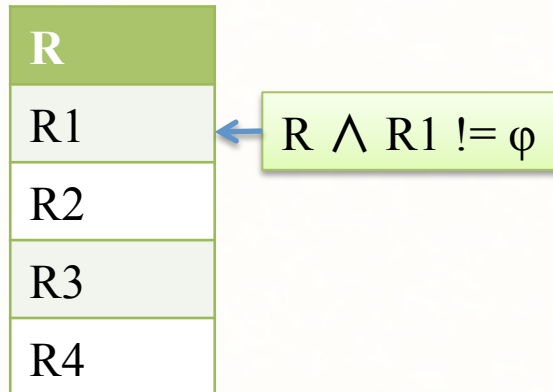
# Dependency Graph

| Rule | Match | Action | Priority | Traffic |
|------|-------|--------|----------|---------|
| R1   | 0000  | Fwd 1  | 6        | 10      |
| R2   | 000*  | Fwd 2  | 5        | 20      |
| R3   | 00**  | Fwd 3  | 4        | 90      |
| R4   | 111*  | Fwd 4  | 3        | 5       |
| R5   | 11**  | Fwd 5  | 2        | 10      |
| R6   | 1***  | Fwd 6  | 1        | 120     |



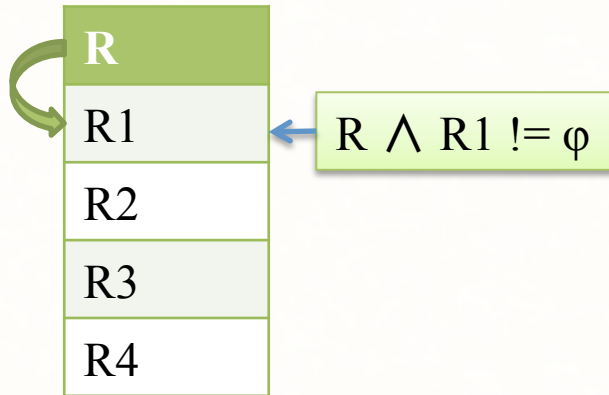
# How to get the dependency graph?

- For a given rule R
  - Find all the rules that its packets may hit if R is removed



# How to get the dependency graph?

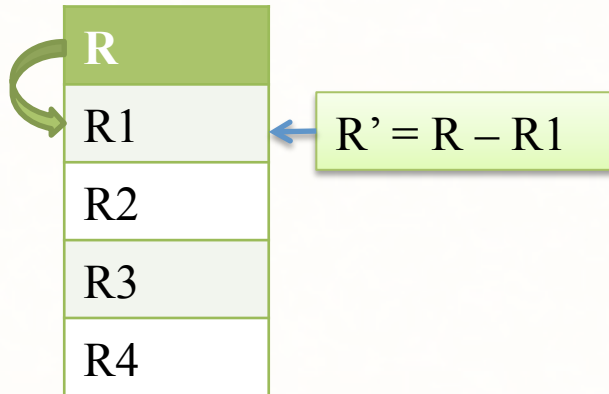
- For a given rule R
  - Find all the rules that its packets may hit if R is removed





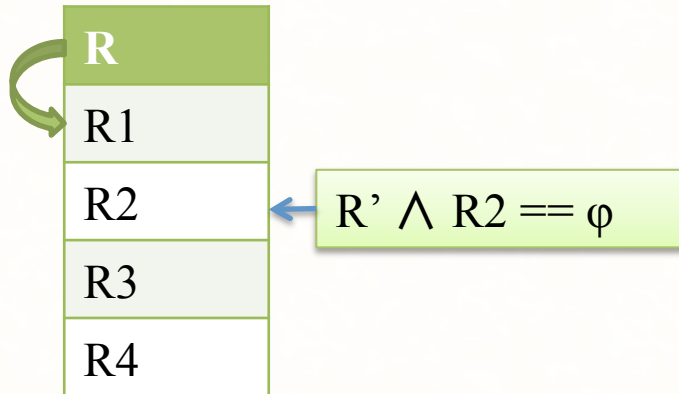
# How to get the dependency graph?

- For a given rule R
  - Find all the rules that its packets may hit if R is removed



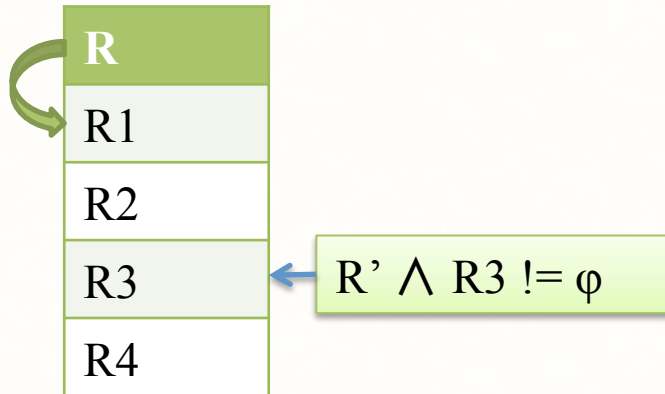
# How to get the dependency graph?

- For a given rule R
  - Find all the rules that its packets may hit if R is removed



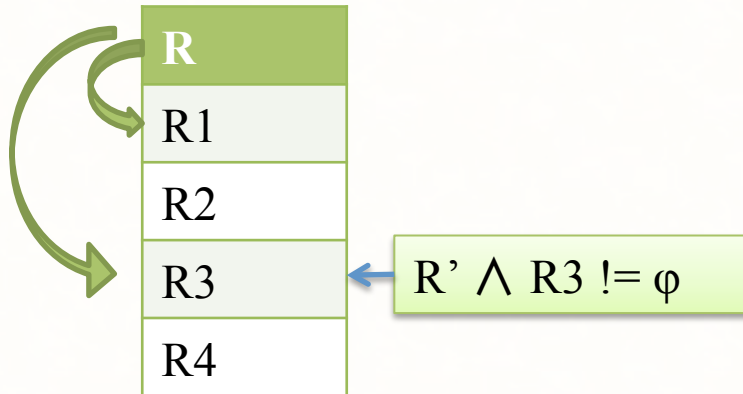
# How to get the dependency graph?

- For a given rule R
  - Find all the rules that its packets may hit if R is removed



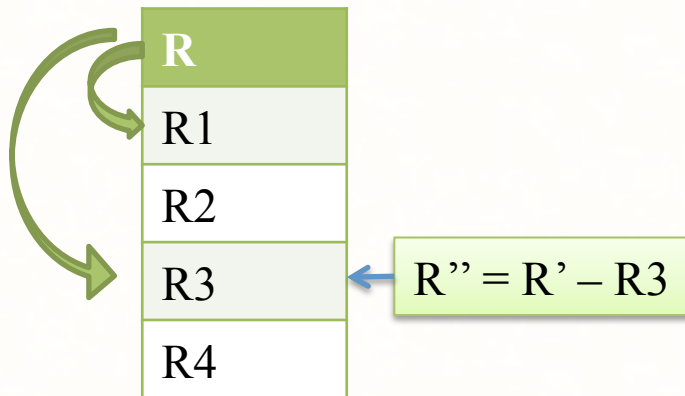
# How to get the dependency graph?

- For a given rule R
  - Find all the rules that its packets may hit if R is removed



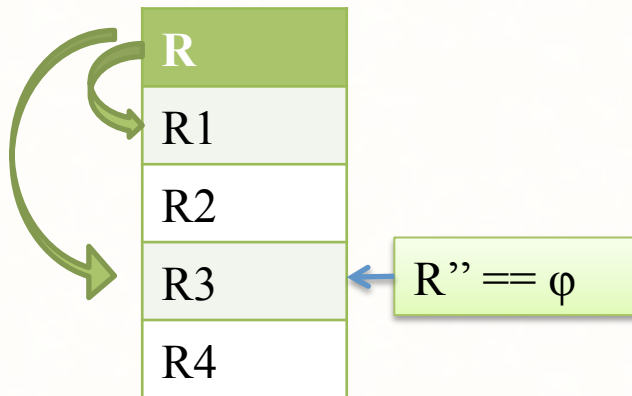
# How to get the dependency graph?

- For a given rule R
  - Find all the rules that its packets may hit if R is removed



# How to get the dependency graph?

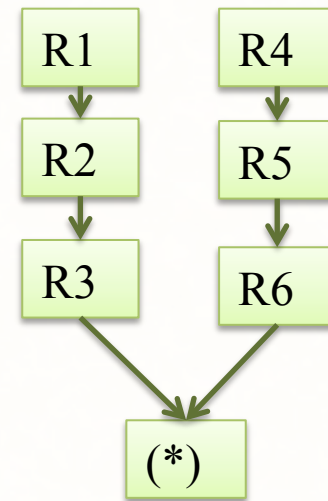
- For a given rule R
  - Find all the rules that its packets may hit if R is removed



(End of dependencies originating from R)

# Dependency Graph

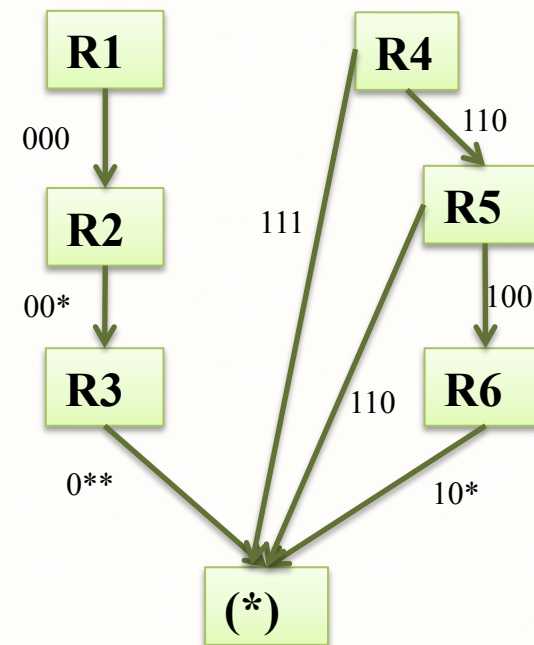
| Rule | Match | Action | Priority | Traffic |
|------|-------|--------|----------|---------|
| R1   | 0000  | Fwd 1  | 6        | 10      |
| R2   | 000*  | Fwd 2  | 5        | 20      |
| R3   | 00**  | Fwd 3  | 4        | 90      |
| R4   | 111*  | Fwd 4  | 3        | 5       |
| R5   | 11**  | Fwd 5  | 2        | 10      |
| R6   | 1***  | Fwd 6  | 1        | 120     |



# Multi-field ternary match -> complex DAG

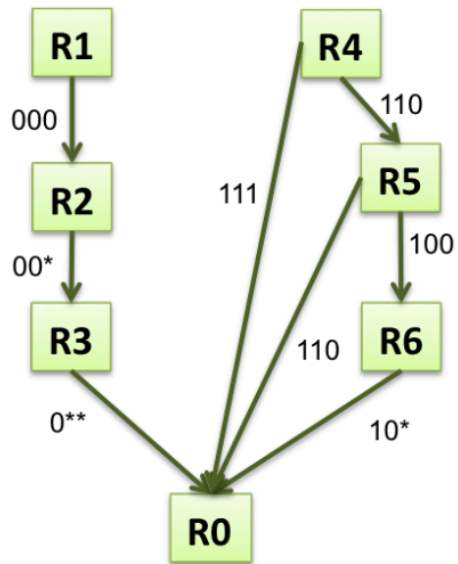


| Rule | (dst_ip, dst_port)   | Ternary Match | Action | Priority |
|------|----------------------|---------------|--------|----------|
| R1   | (10.10.10.10/32, 10) | 000           | Fwd 1  | 6        |
| R2   | (10.10.10.10/32, *)  | 00*           | Fwd 2  | 5        |
| R3   | (10.10.0.0/16, *)    | 0**           | Fwd 3  | 4        |
| R4   | (11.11.11.11/32, *)  | 11*           | Fwd 4  | 3        |
| R5   | (11.11.0.0/16, 10)   | 1*0           | Fwd 5  | 2        |
| R6   | (11.11.10.10/32, *)  | 10*           | Fwd 6  | 1        |

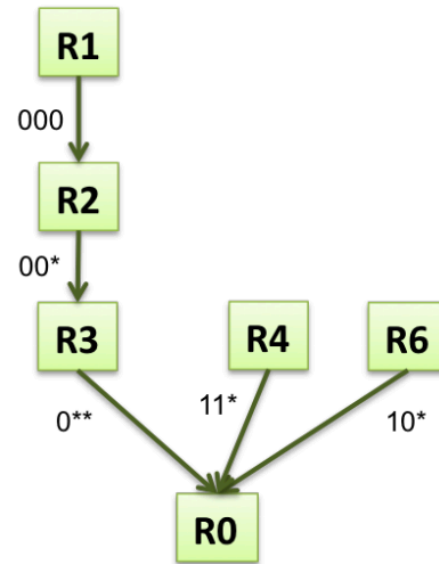




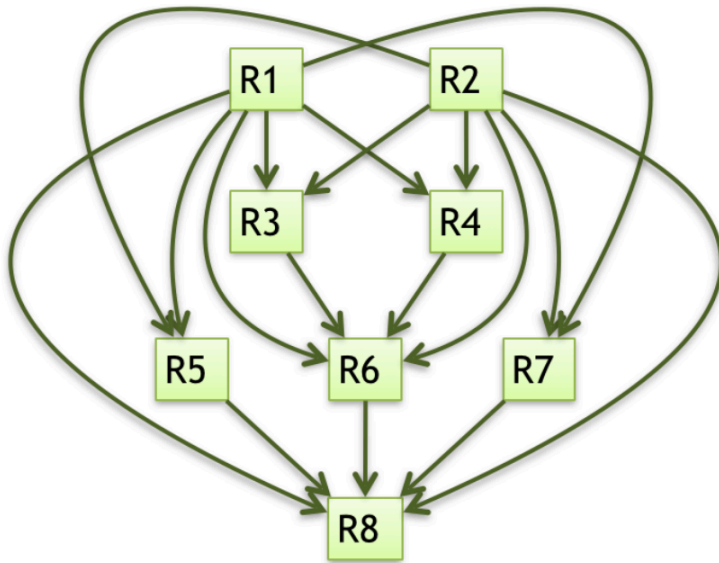
# Incremental Updates



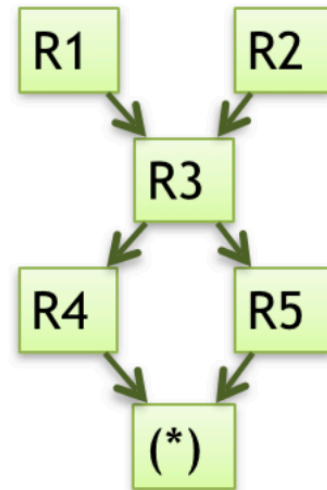
**Delete(R5)**  
→  
←  
**Insert(R5)**



# Complex dependencies in the wild



Reanzz Network

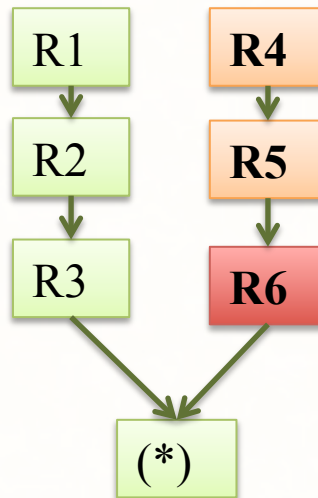


CoVisor (NSDI'15)

# Dependency-aware Caching

# Dependent-Set Caching

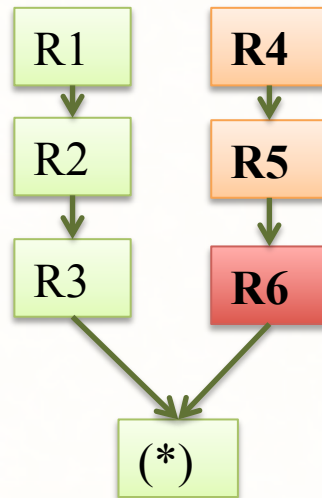
- All descendants in DAG are dependents
- Cache dependent rules for correctness



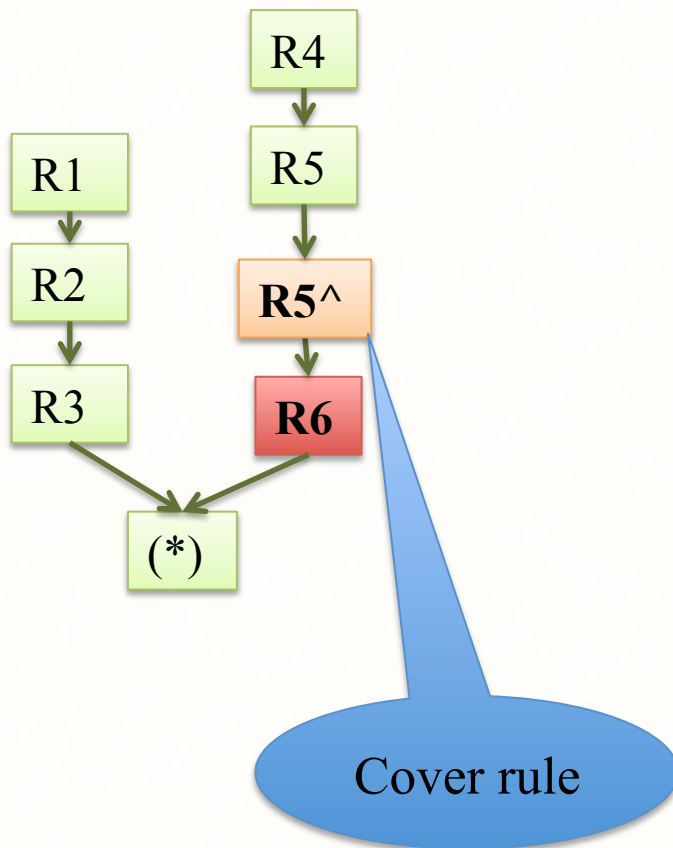
## 2. Efficient Caching

# Dependent-Set Overhead

Too Costly?

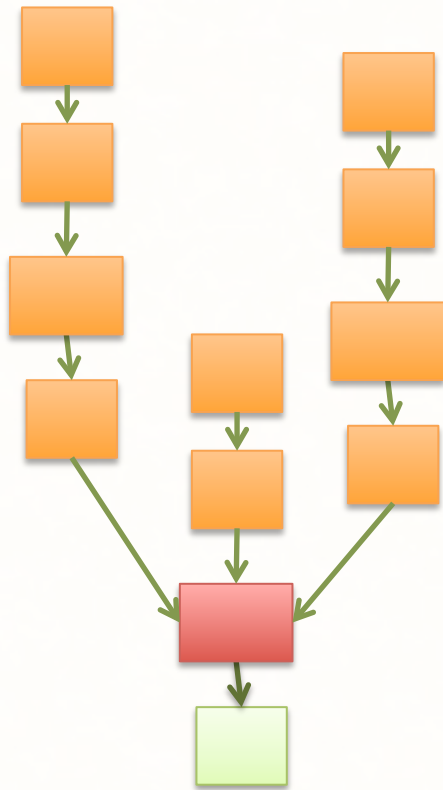


# Cover-Set

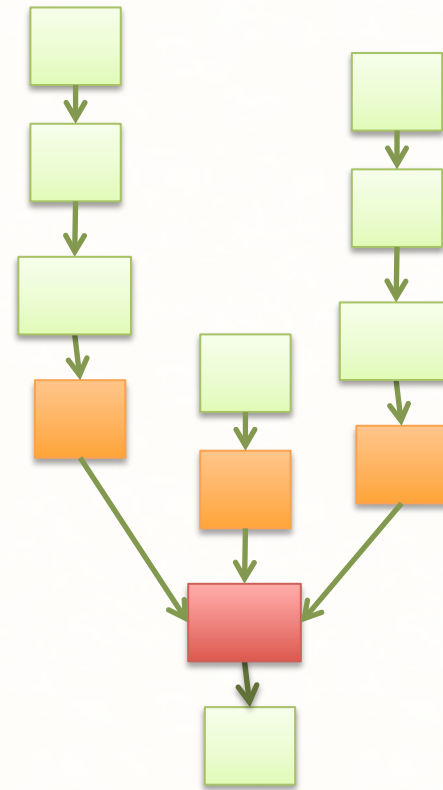


| Rule       | Match      | Action       |
|------------|------------|--------------|
| R1         | 000        | Fwd 1        |
| R2         | 00*        | Fwd 2        |
| R3         | 0**        | Fwd 3        |
| R4         | 11*        | Fwd 4        |
| R5         | 1*0        | Fwd 5        |
| <b>R5^</b> | <b>1*0</b> | <b>To_SW</b> |
| R6         | 10*        | Fwd 6        |
| (*)        | ***        | To_SW        |

# Dependency Splicing reduces rule cost!



Dependent-Set

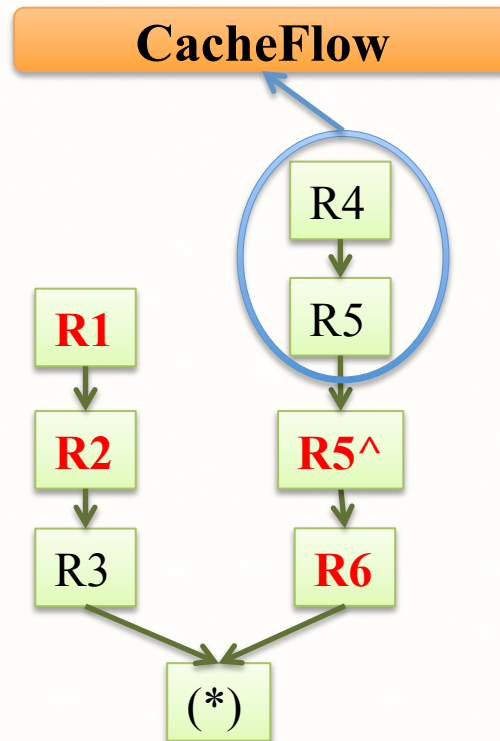


Cover-Set

 Rule Space Cost



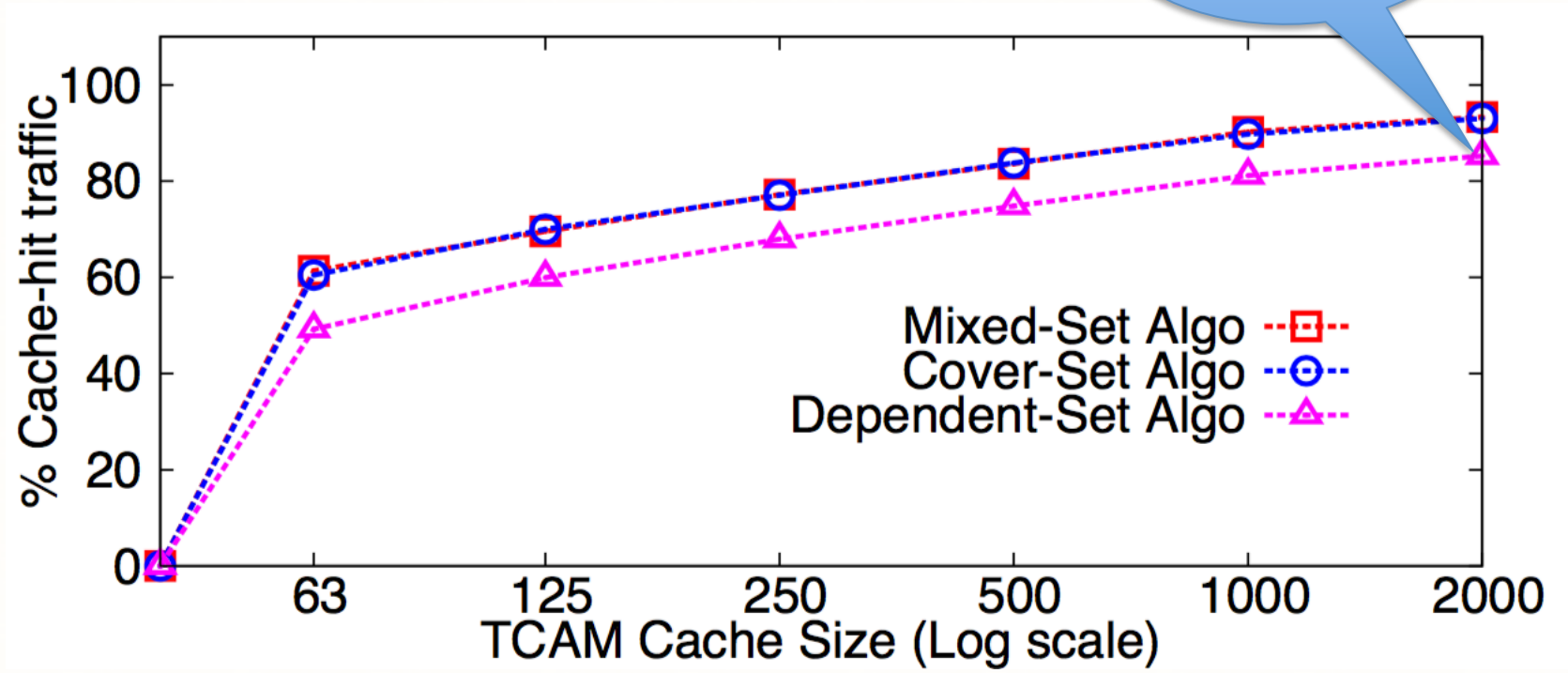
# Mixed Set: Best of both worlds



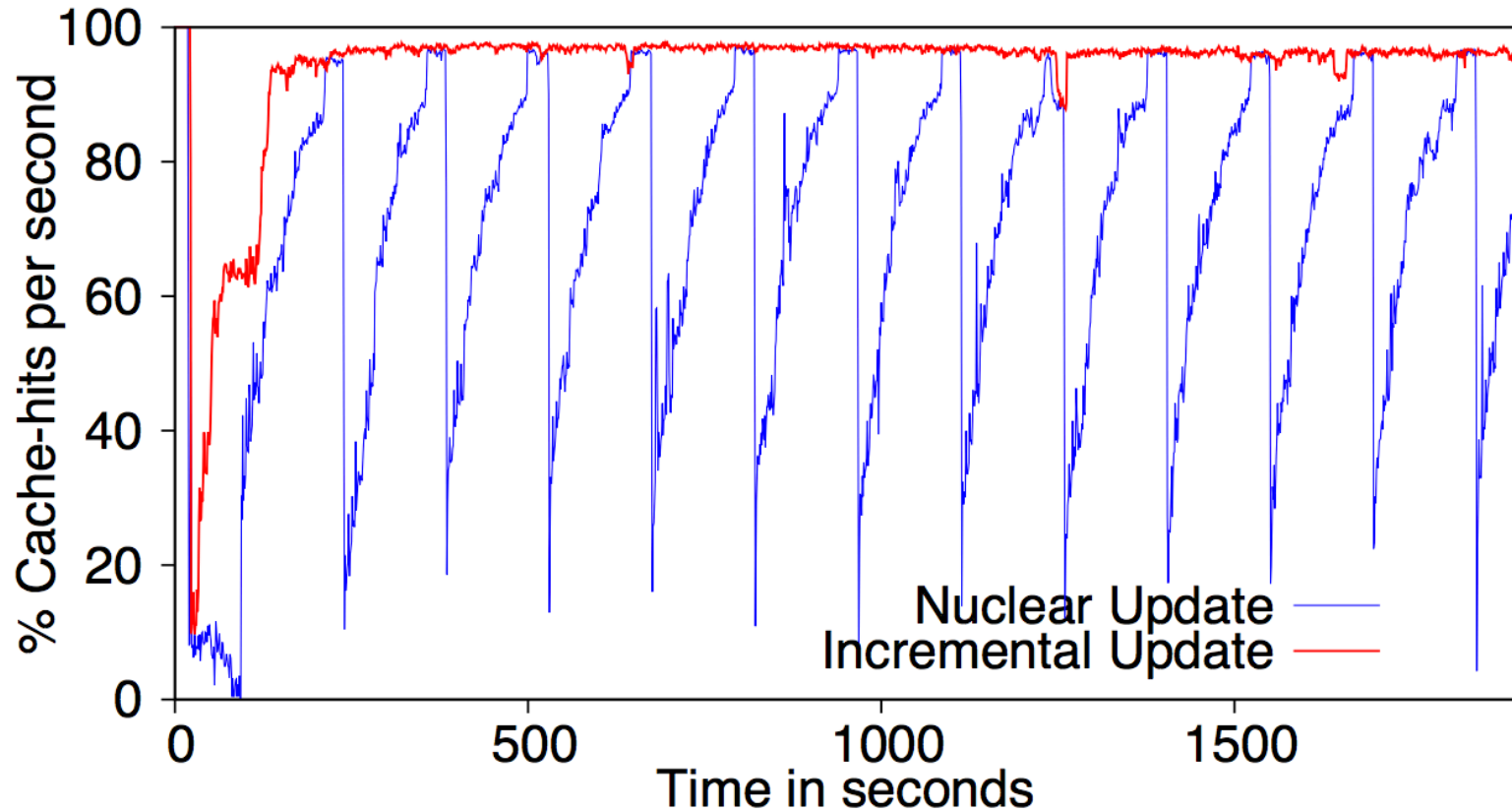
# Dependency Chains – Clear Gain

- CAIDA packet trace

3% rules  
85% traffic

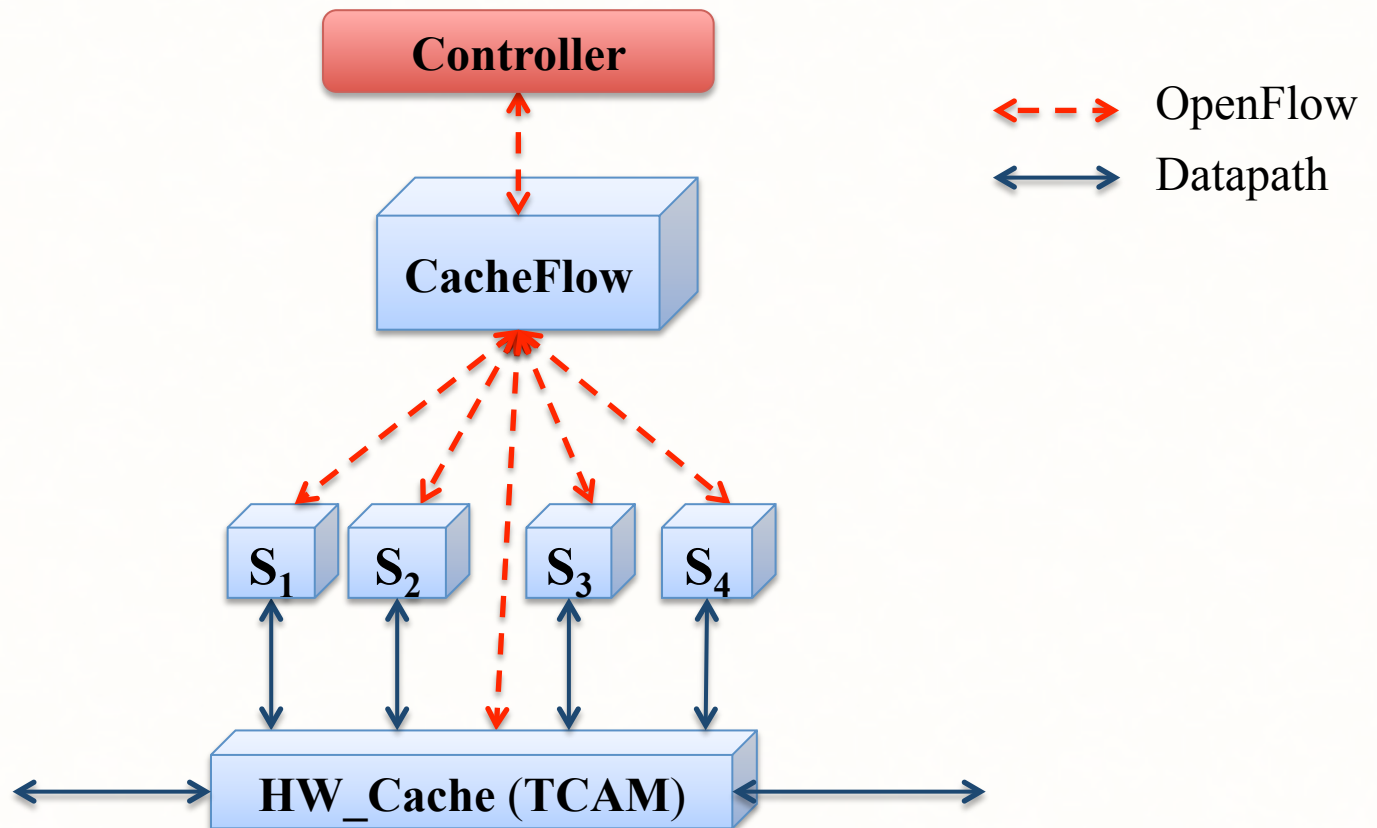


# Incremental update is more stable

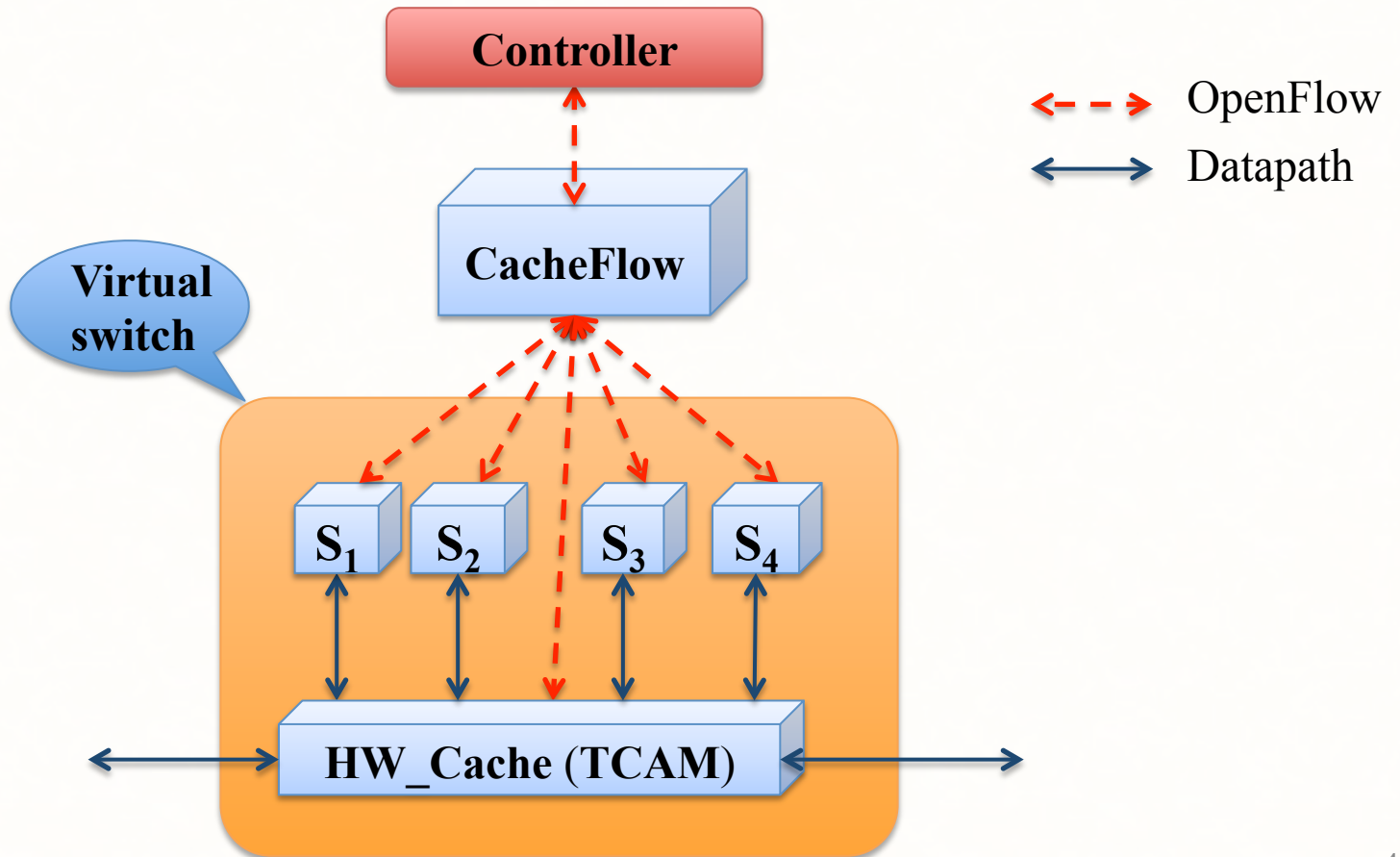


## 3. Transparent Caching

# 3. Transparent Design

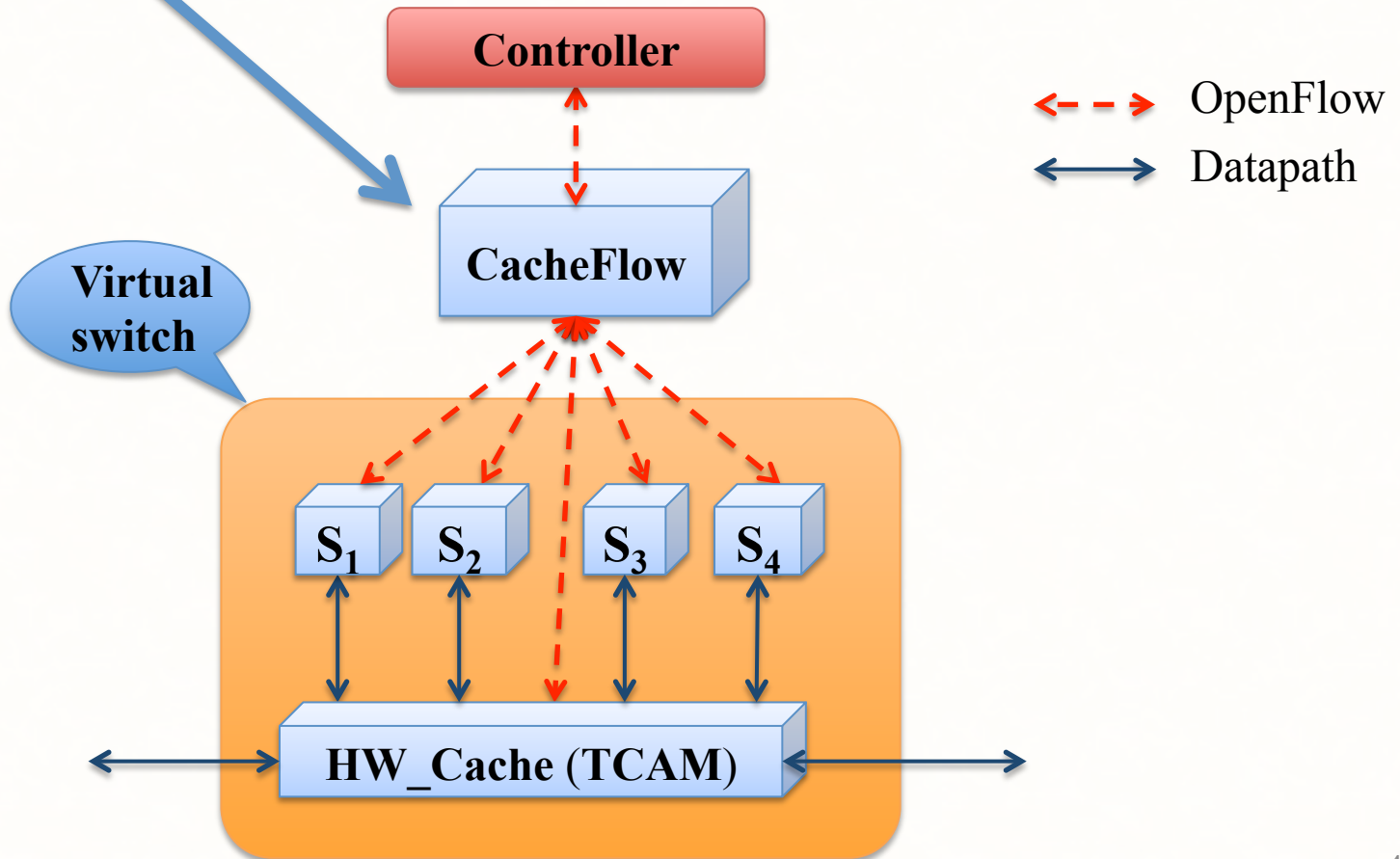


# 3. Transparent Design



# 3. Transparent Design

Emulates counters, barriers, timeouts etc.



# Conclusion

- Rule caching for OpenFlow rules
  - Dependency analysis for *correctness*
  - Splicing dependency chains for *efficiency*
  - *Transparent* design



# Conclusion

- Rule caching for OpenFlow rules
  - Dependency analysis for *correctness*
  - Splicing dependency chains for *efficiency*
  - *Transparent* design
  
- Get ready for infinite Ca\$hflow!

**Thank You**

# Backup Slides

# Dependency Chains – Clear Gain

- Stanford Backbone Router Config

