

# *Network Updates*

## *Chapter 10*



*Distributed Systems lecture : Roger Wattenhofer*

*Today's lecturer: Klaus-Tycho Foerster*

# Overview

- Software-Defined Networking
- Blackhole-Free Updates
- Loop-Free Updates
- Packet Coherent Updates
- Capacity-Consistent Updates

# Overview

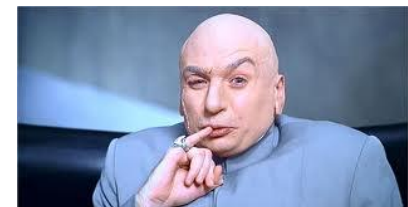
- **Software-Defined Networking**
- Blackhole-Free Updates
- Loop-Free Updates
- Packet Coherent Updates
- Capacity-Consistent Updates

# Network Updates

- The Internet: Designed for selfish participants
  - Often inefficient (low utilization of links), but robust



- But what happens if the WAN is controlled by a single entity?
  - Examples: Microsoft & Amazon & Google ...
  - They spend hundreds of millions of dollars per year



# Software-Defined Networking

- Possible solution: **Software-Defined Networking (SDNs)**



- General Idea: Separate data & control plane in a network
- Centralized controller updates networks rules for optimization
  - Controller (*control plane*) updates the switches/routers (*data plane*)



- Centralized controller implemented with replication, e.g. Paxos

# When will the Network Updates be implemented?



*old network rules*



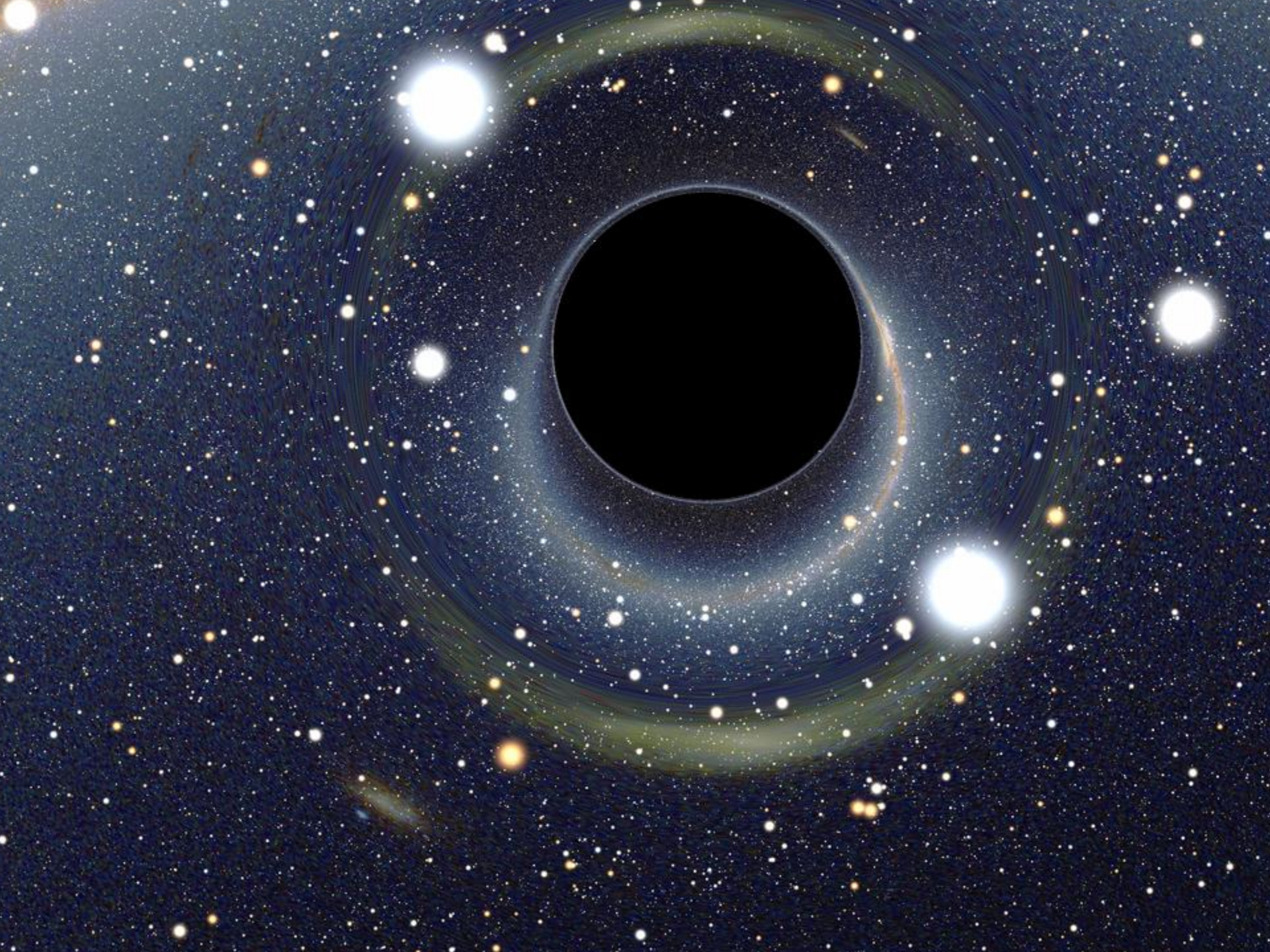
*new network rules*

EVENTUALLY  
EVERYTHING

# Overview

- Software-Defined Networking
- **Blackhole-Free Updates**
- Loop-Free Updates
- Packet Coherent Updates
- Capacity-Consistent Updates

<b>Dependencies</b>	<b>None</b>
<b>Eventual consistency</b>	Always guaranteed





# Blackholes

- Sounds scary? It is!\*
- A packet arrives at some switch...
  - ... while the switch deletes an old rule and implements a new one
  - So the switch does not know what to do with it?!
  - The packet gets dropped 😞
- What can we do?
  - Make sure that the switch always has some rule for every packet!
- How can we solve the problem?
  - “add before remove”
  - Just send everything back to the controller?
  - Send everything somewhere?
    - What is the issue with that?



\*for network operators 😊

# Overview

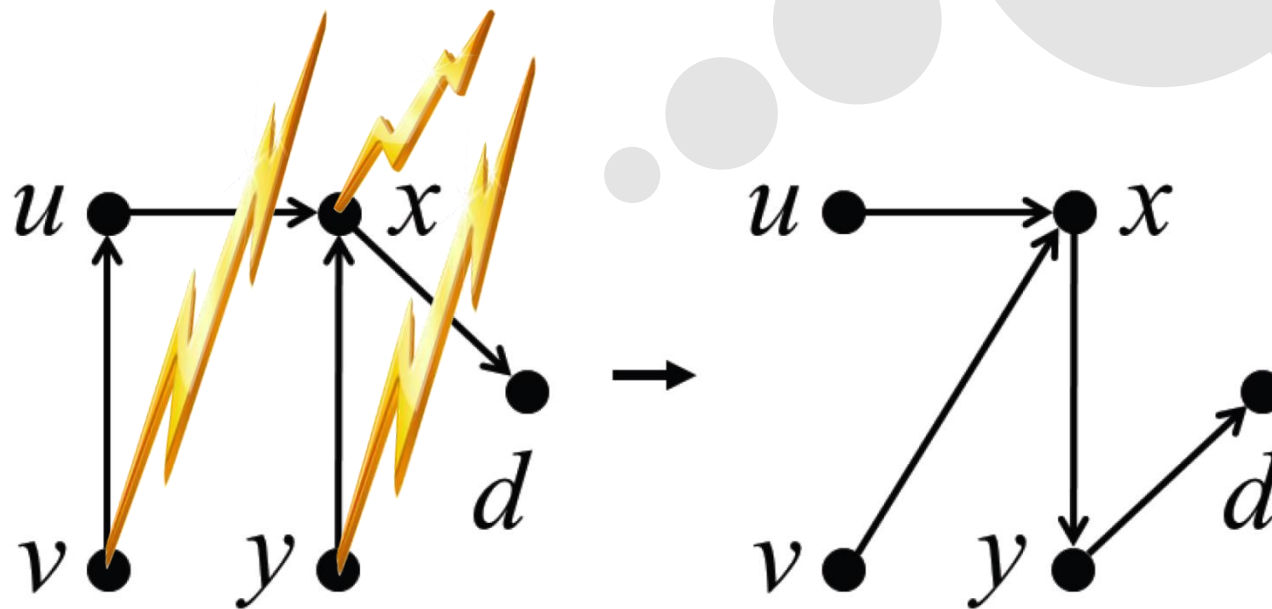
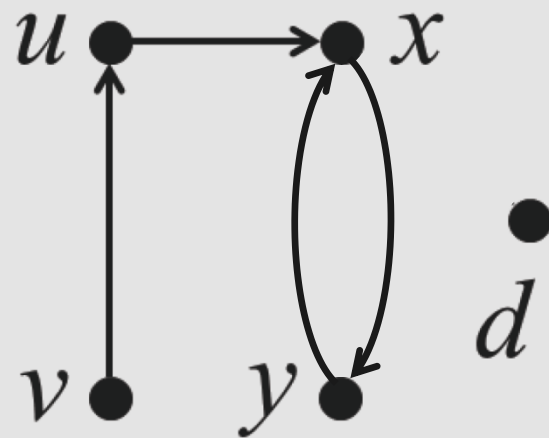
- Software-Defined Networking
- Blackhole-Free Updates
- **Loop-Free Updates**
- Packet Coherent Updates
- Capacity-Consistent Updates

<b>Dependencies</b>	<b>None</b>	<b>Self</b>
<b>Eventual consistency</b>	Always guaranteed	
<b>Blackhole freedom</b>	Impossible	Add before remove

# Loop-Free Updates

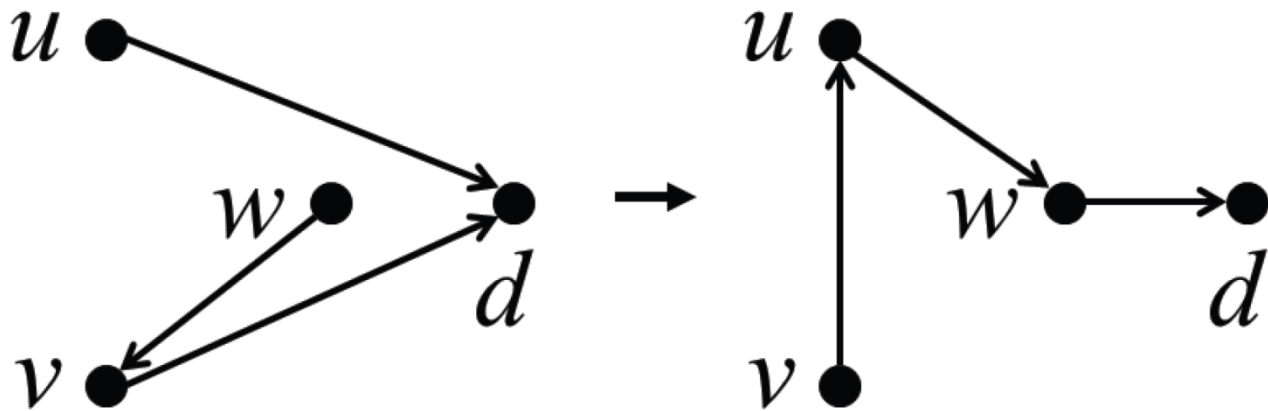


SDN Controller



# Minimum SDN Updates?

# Minimum Updates: Another Example

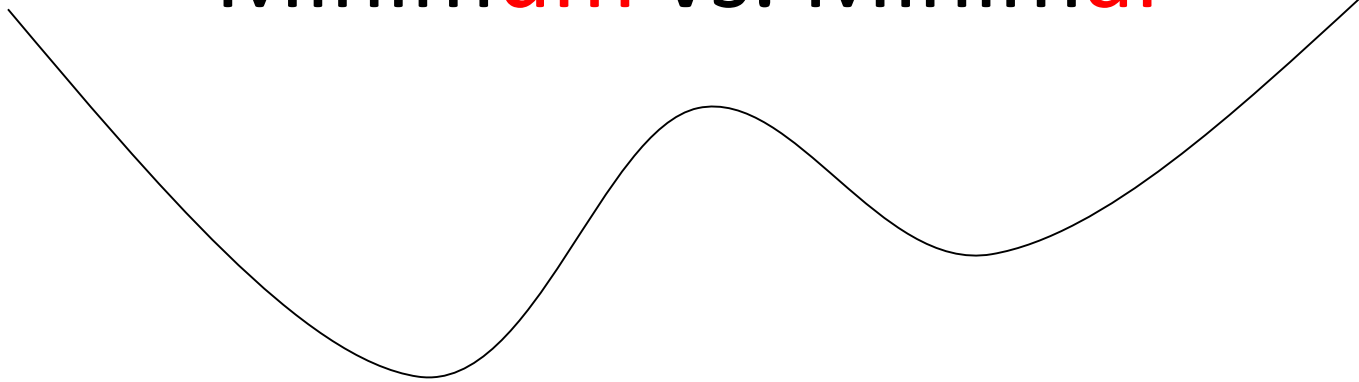


or

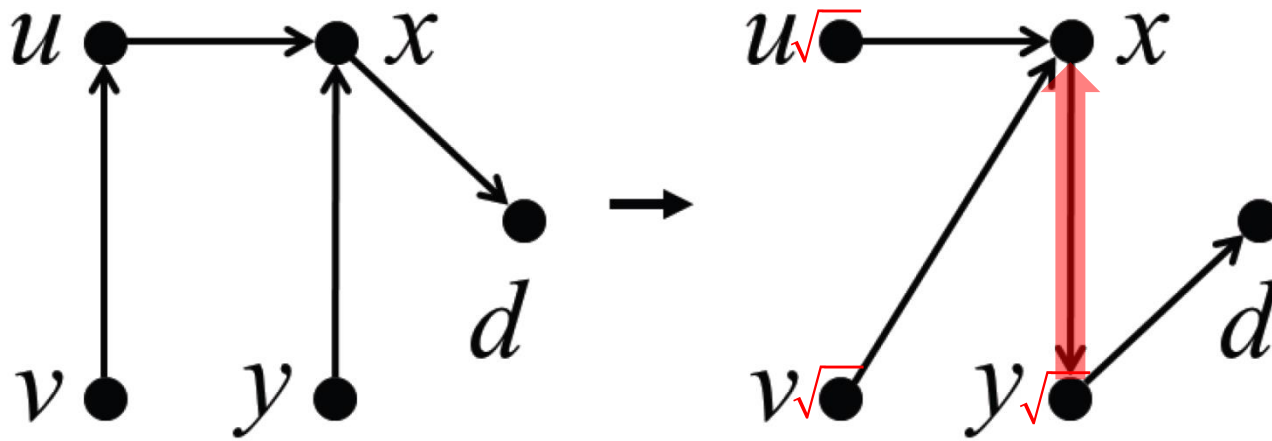


No node can improve  
without hurting another  
node

## Minimum vs. Minimal



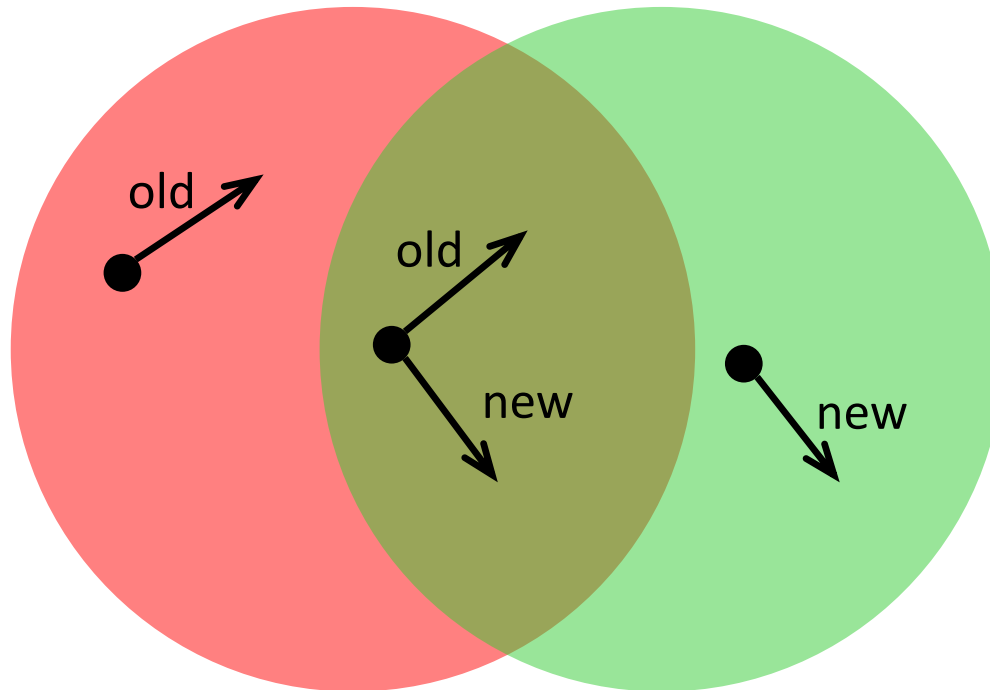
# Minimal Dependency Forest



Next: An algorithm to compute minimal dependency forest.

# Algorithm for Minimal Dependency Forest

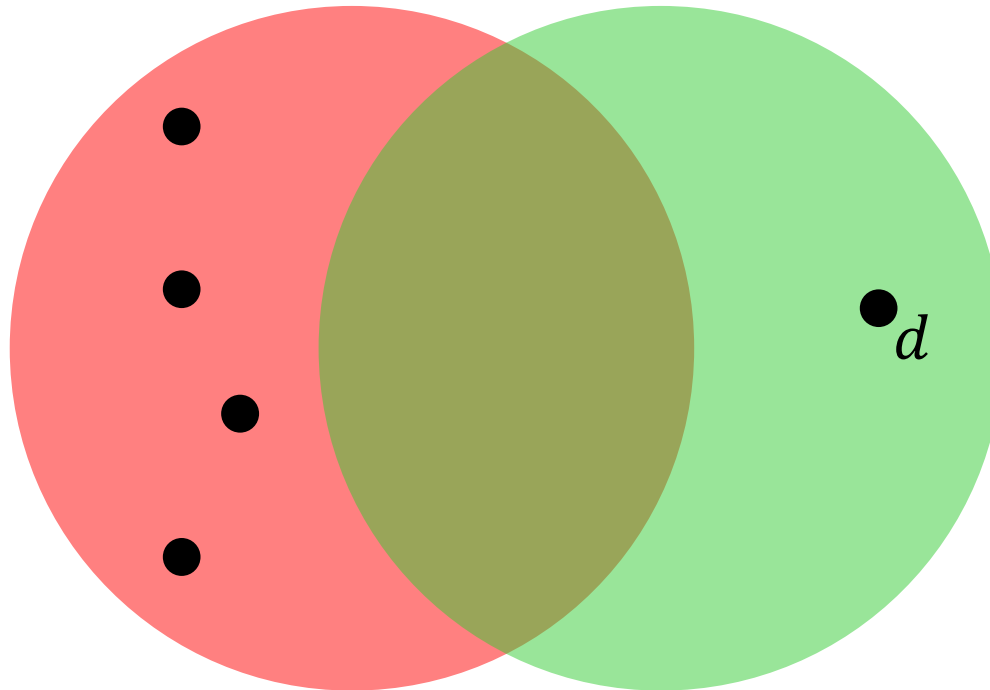
- Each node in one of three states: **old**, **new**, and limbo (both old *and* new)





# Algorithm for Minimal Dependency Forest

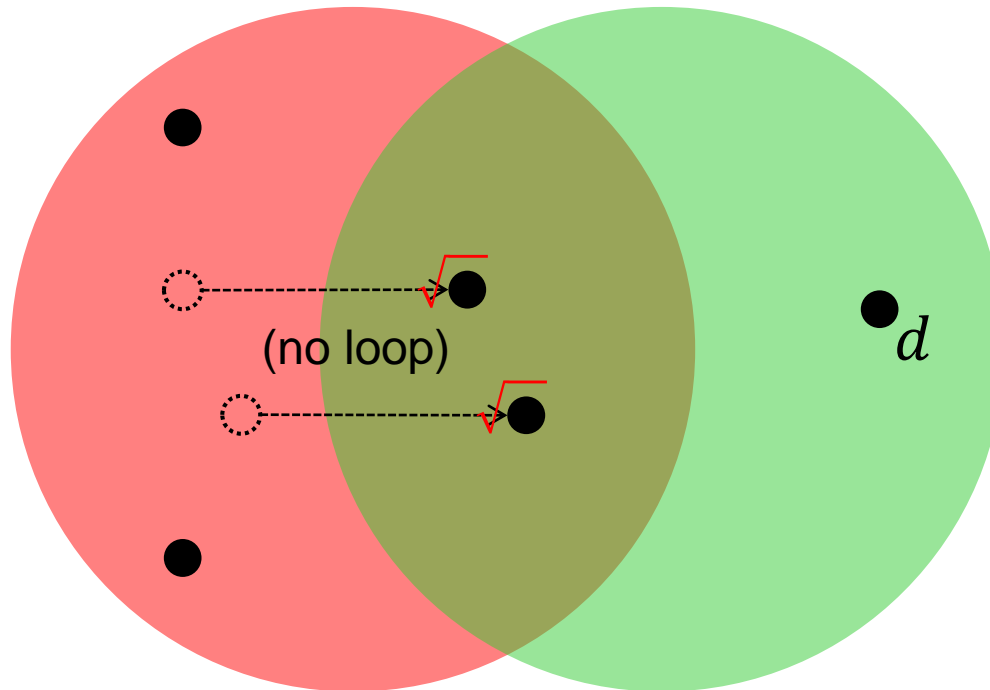
- Each node in one of three states: **old**, **new**, and limbo (both old *and* new)
- Originally, destination node in **new** state, all other nodes in **old** state
- Invariant: No loop!



# Algorithm for Minimal Dependency Forest

## Initialization

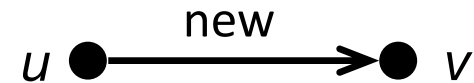
- **Old** node  $u$ : No loop\* when adding **new** pointer, move node to limbo!
- This node  $u$  will be a root in dependency forest



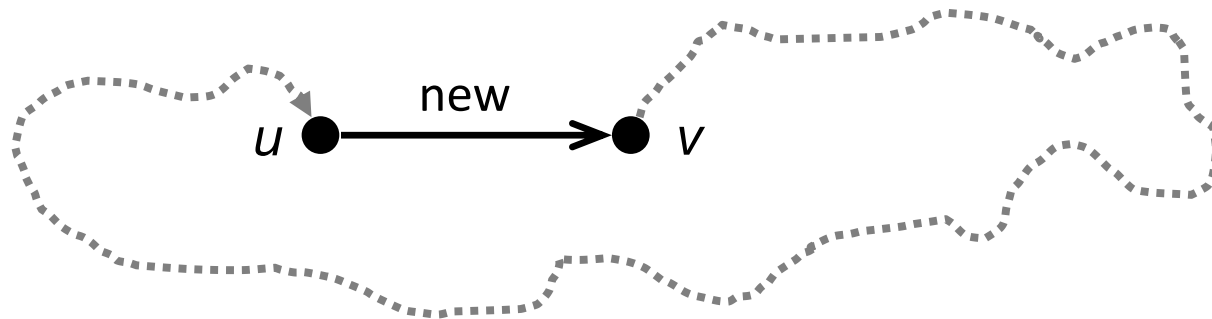
\*Loop Detection: Simple procedure, see next slide

# Loop Detection

- Will a new rule  $u.new = v$  induce a loop?
  - We know that the graph so far has no loops
  - Any new loop *must* contain the edge  $(u,v)$



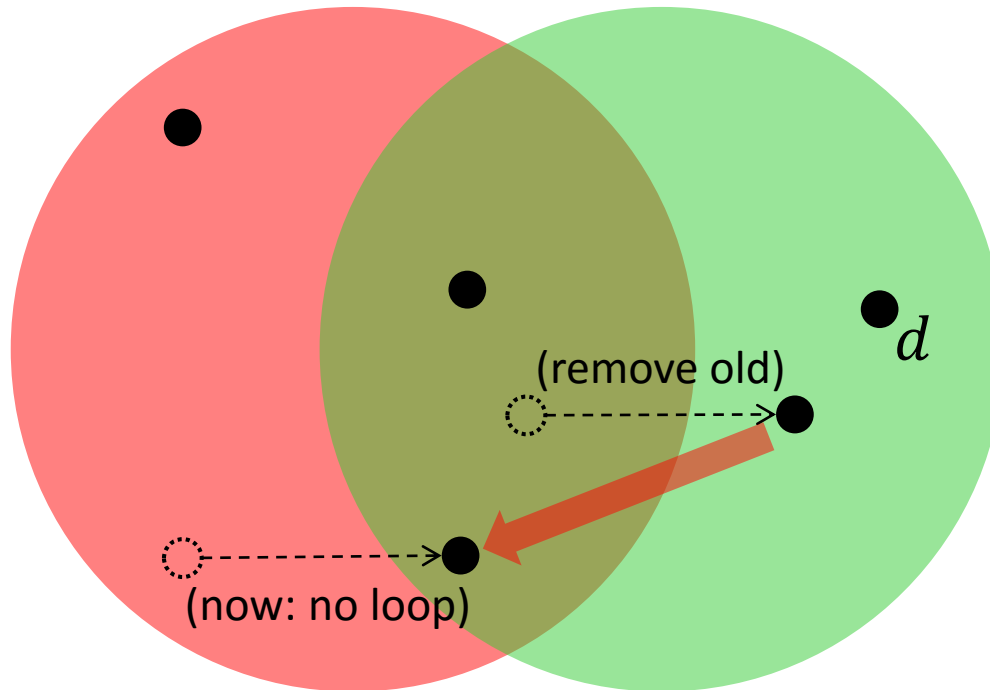
- In other words, is node  $u$  now *reachable* from node  $v$ ?



- Depth first search (DFS) at node  $v$ 
  - If we visit node  $u$ : the new rule induces a loop
  - Else: no loop

# Algorithm for Minimal Dependency Forest

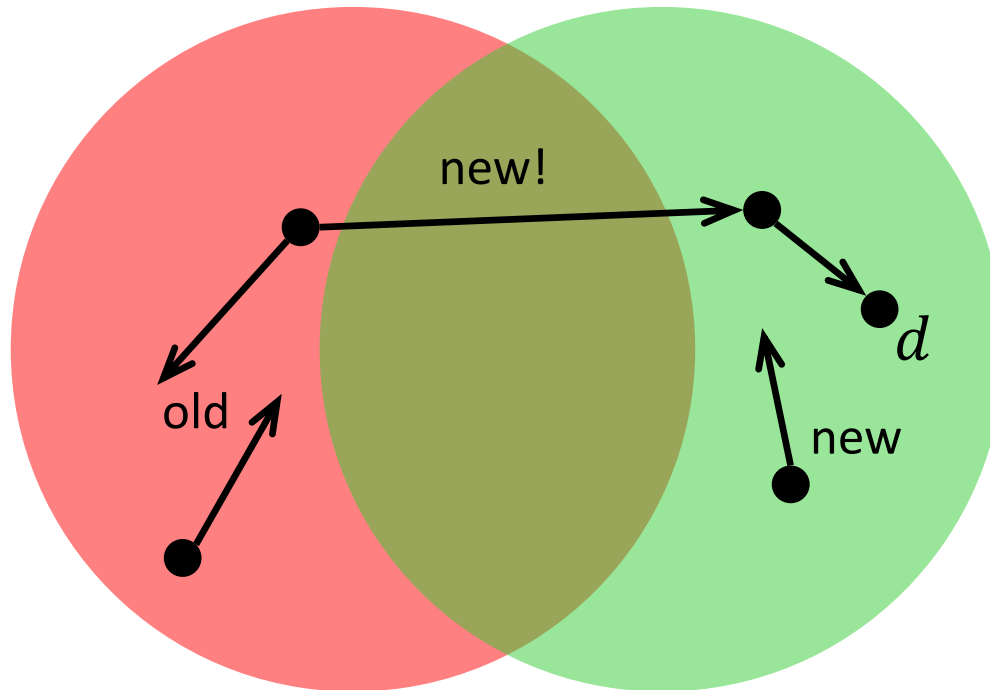
- Limbo node  $u$ : Remove **old** pointer (move node to **new**)
- Consequence: Some **old** nodes  $v$  might move to limbo!
- Node  $v$  will be child of  $u$  in dependency forest!



# Algorithm for Minimal Dependency Forest

Process terminates

- You can always move a node from limbo to **new**.
- Can you ever have **old** nodes but no limbo nodes? No, because...

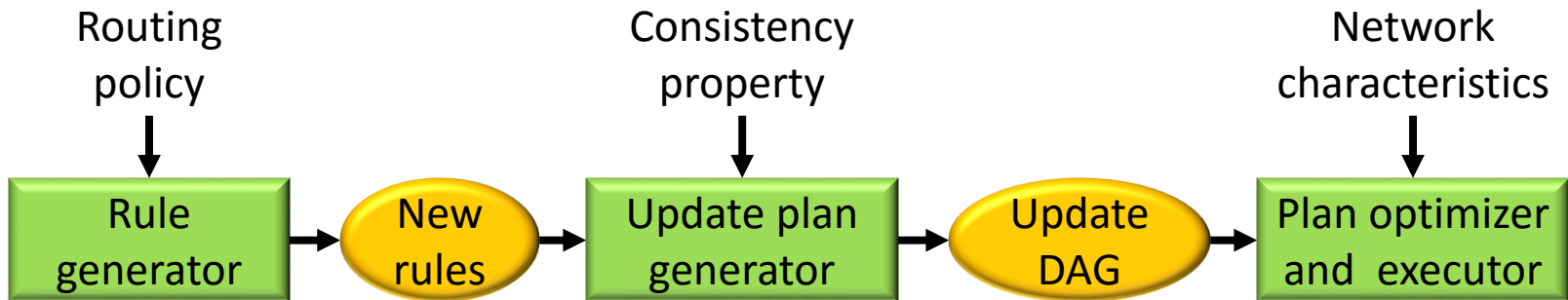


...one can easily derive a contradiction!

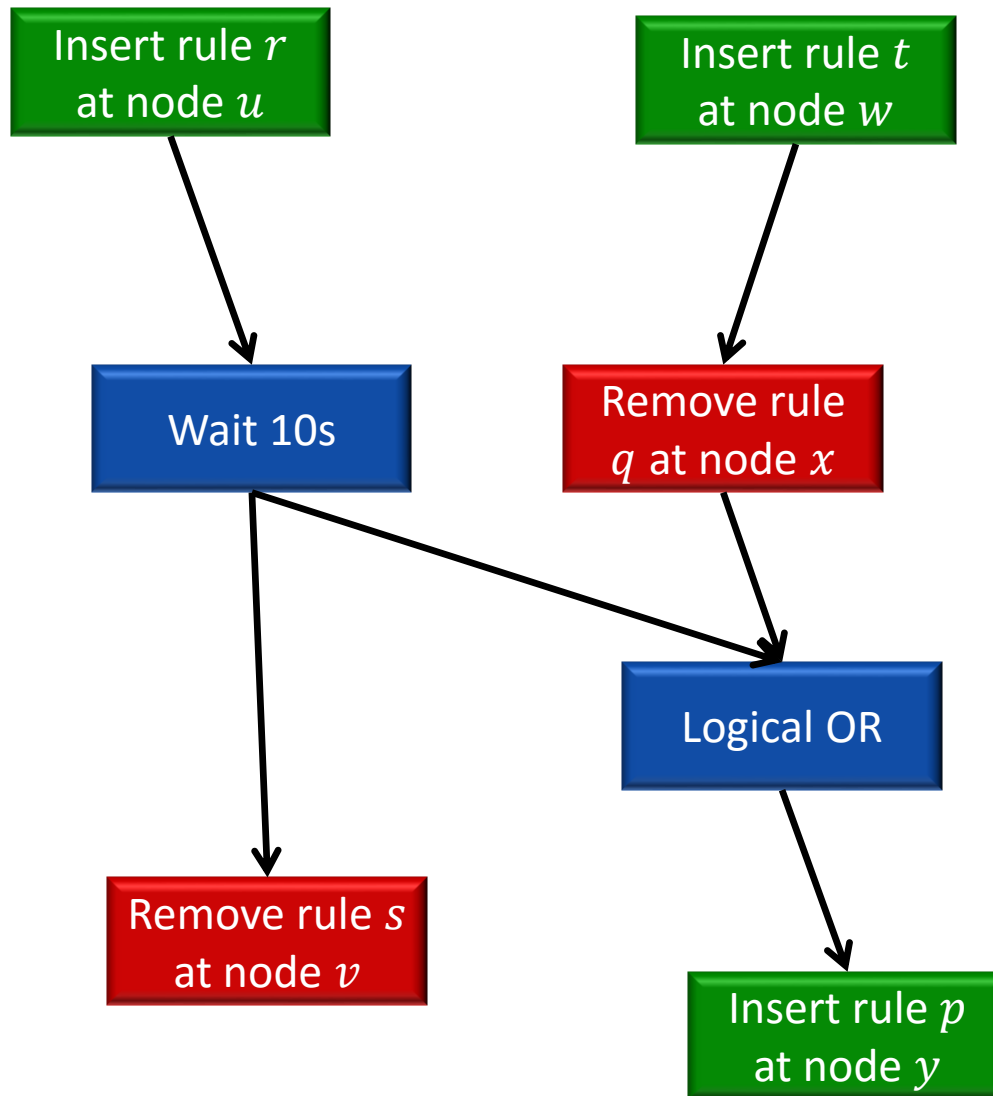
It's *not* just how to compute new rules.

It is also how to gracefully get  
**from current to new** configuration,  
respecting consistency.

# Architecture

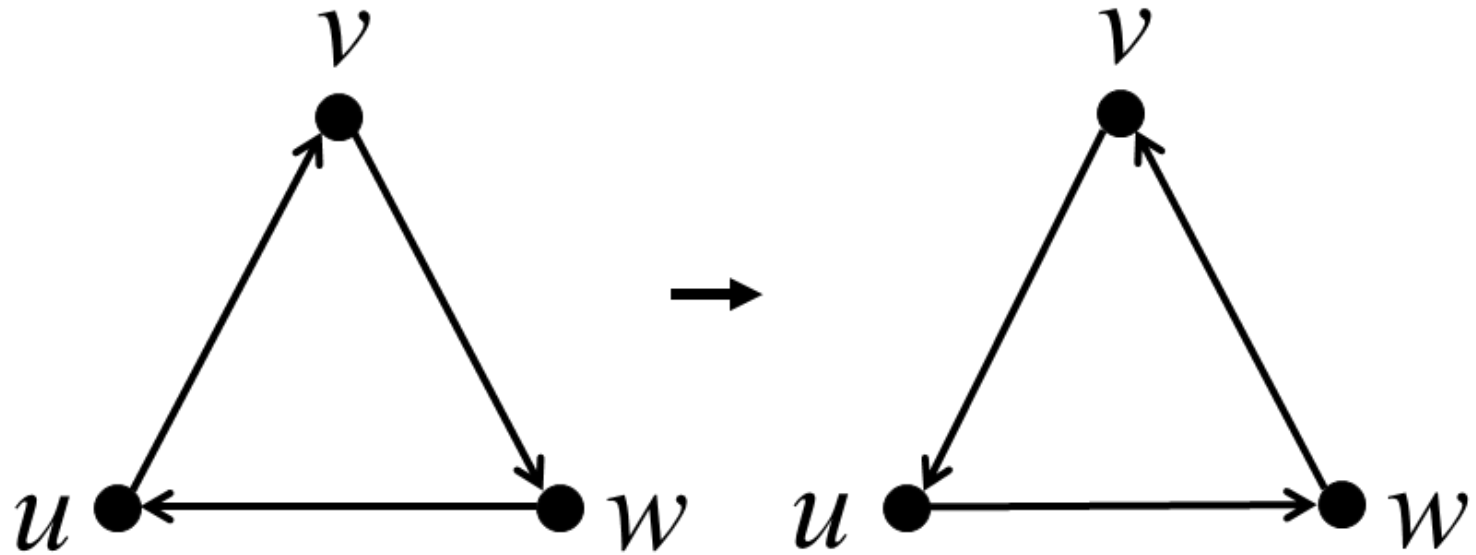


# Update DAG



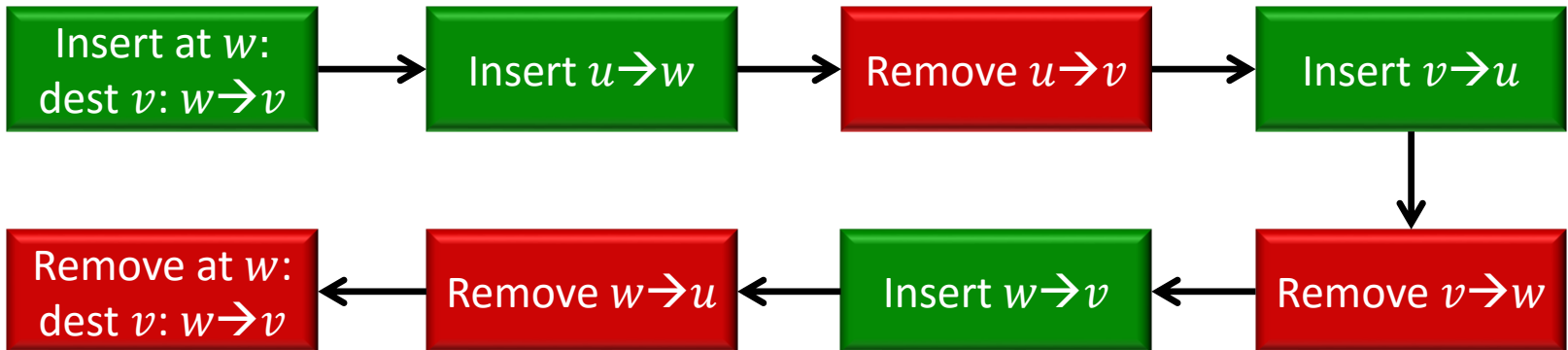
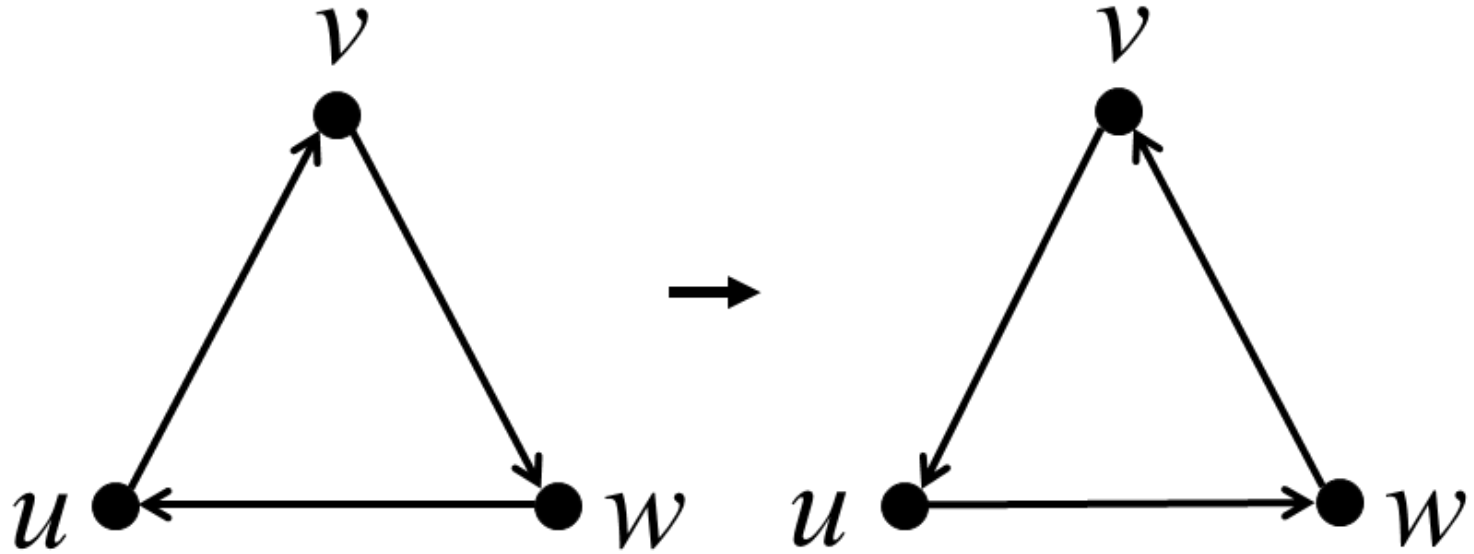


## Multiple Destinations using Prefix-Based Routing

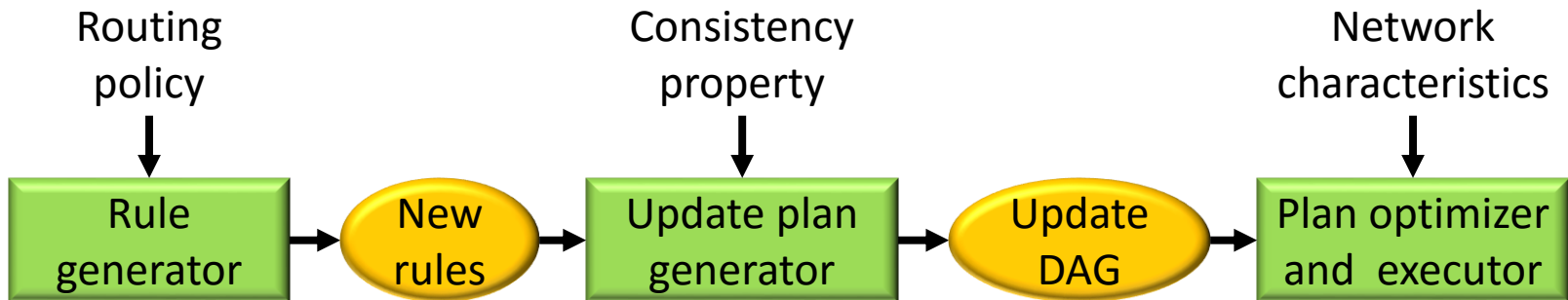


- No new “default” rule can be introduced without causing loops
- Solution: Rule-Dependency Graphs!
- Deciding if simple update schedule exists is hard!

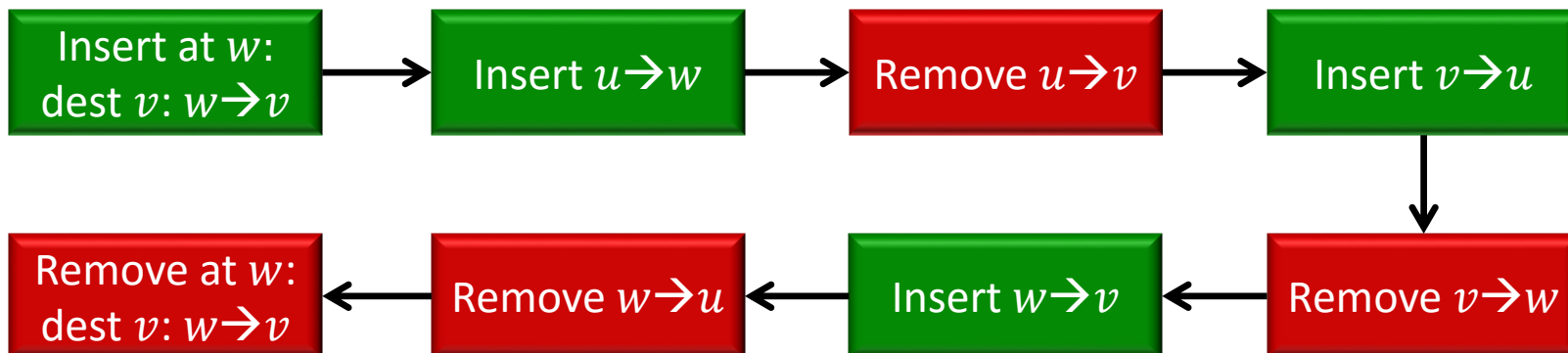
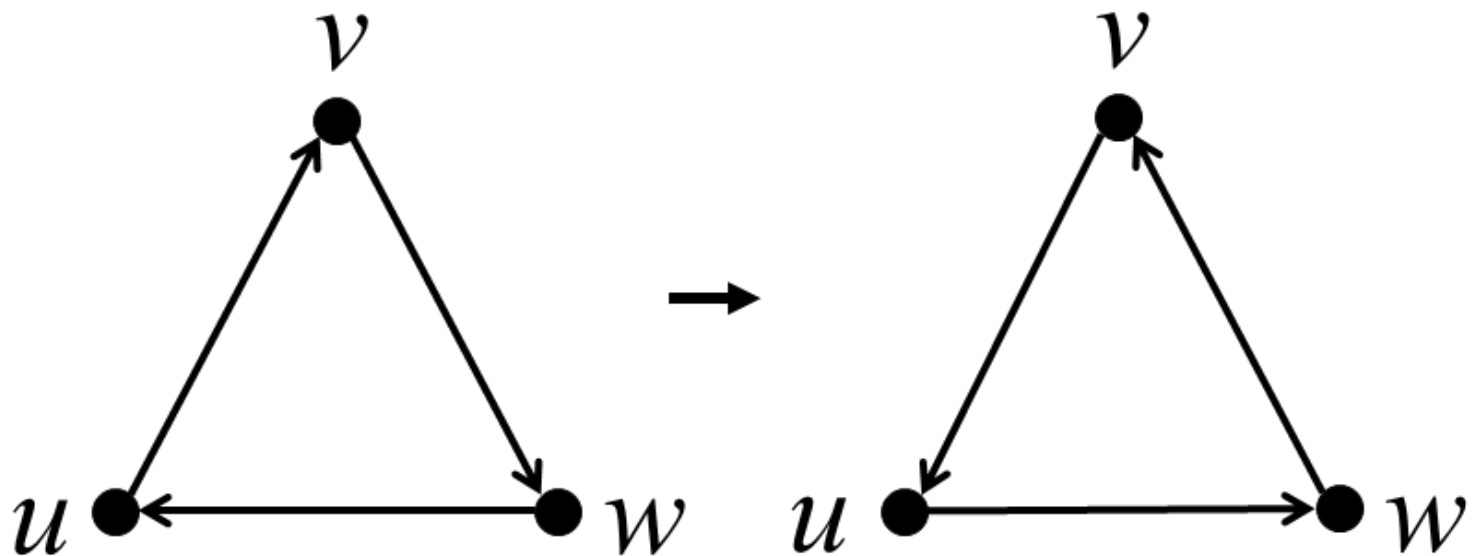
# Breaking Cycles



# Architecture



# Breaking Cycles



# Overview

- Software-Defined Networking
- Blackhole-Free Updates
- Loop-Free Updates
- **Packet Coherent Updates**
- Capacity-Consistent Updates

Dependencies	None	Self	Downstream subset
Eventual consistency	Always guaranteed		
Blackhole freedom	Impossible	Add before remove	
Loop freedom	Impossible		Rule dep. forest

# Packet-Coherent Updates

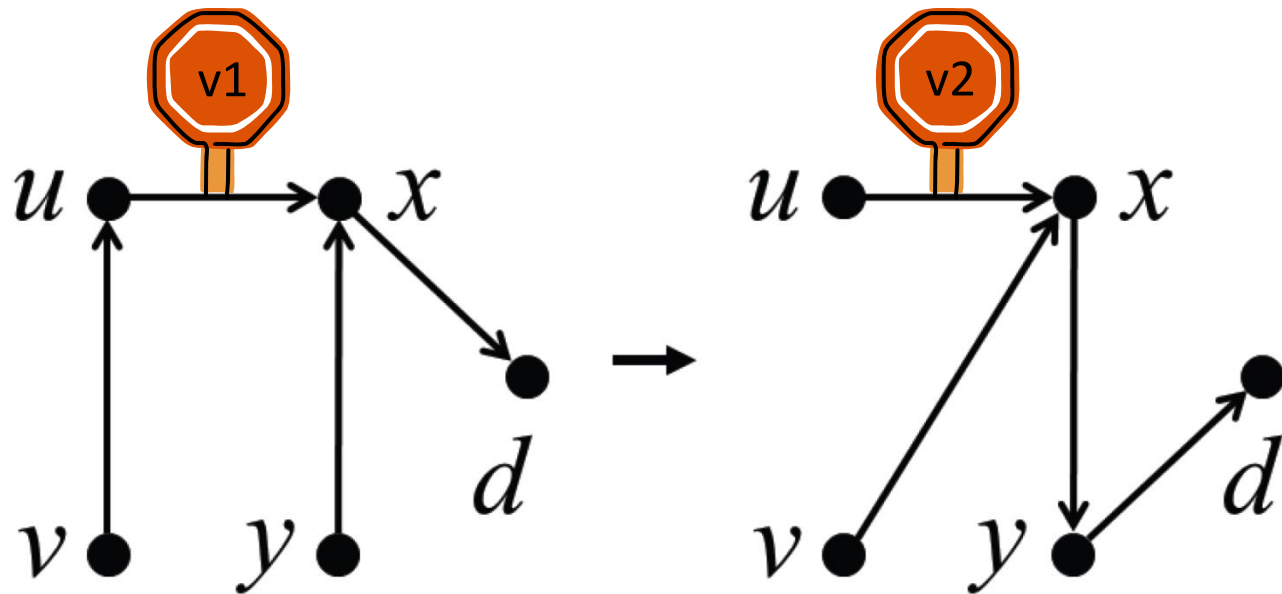
- Definition: A packet should always either
  - Use the old rules
  - Use the new rules
  - Important for waypointing (e.g., firewalls)
  
- General idea:
  - Stamp every packet with a version number
  - Send new rules to all switches
  - When all switches confirmed:
    - Stamp all packets with the next version number
  - Once all old packets are gone
    - Delete old rules



# Example

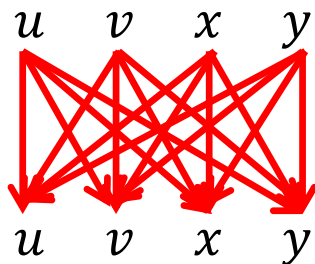


SDN Controller



# Comparison

Version Numbers



Loop Free Solution



## version numbers

- no mix of old and new rules
- loop freedom & packet coherence
- "programmers dream"
  
- more switch memory
- changes packets
- update all involved switches
- when can we delete old rules?

## loop free updates

- mix of old and new rules
- loop freedom, but **no** packet coherence
- needs algorithms
  
- early first effects
- **packets unaffected**

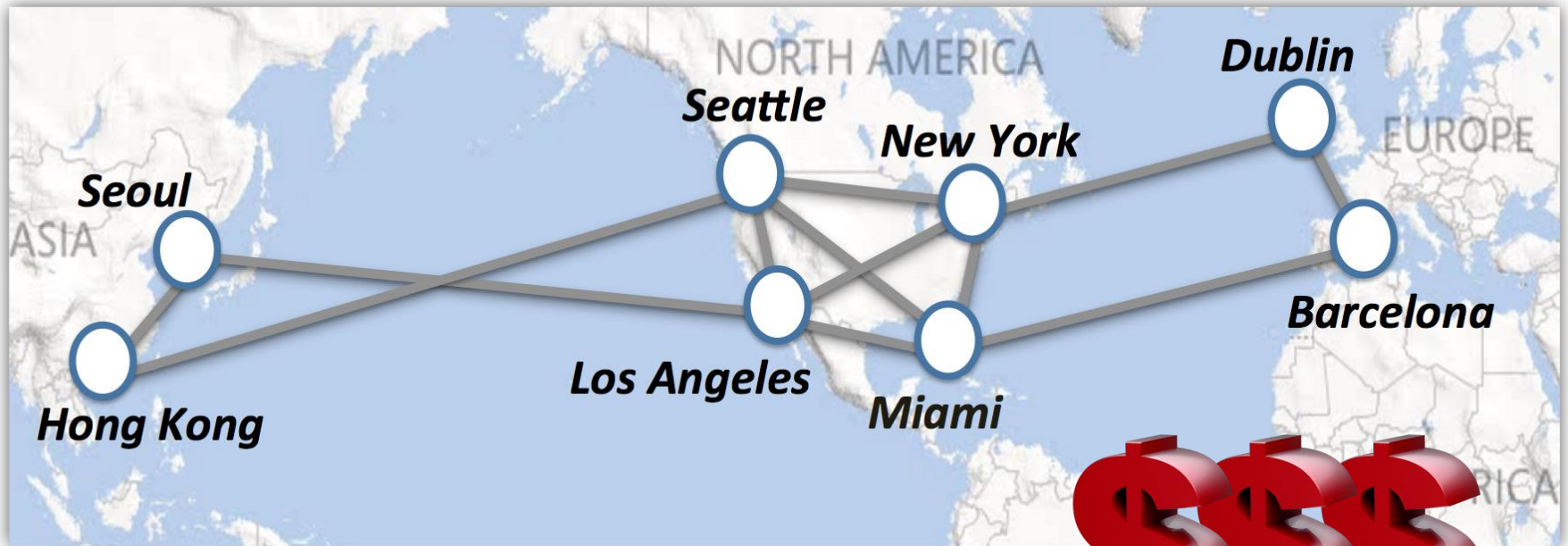


# Overview

- Software-Defined Networking
- Blackhole-Free Updates
- Loop-Free Updates
- Packet Coherent Updates
- **Capacity-Consistent Updates**

Dependencies	None	Self	Downstream subset	Downstream all
Eventual consistency	Always guaranteed			
Blackhole freedom	Impossible	Add before remove		
Loop freedom	Impossible		Rule dep. forest	
Packet coherence	Impossible			Version numbers

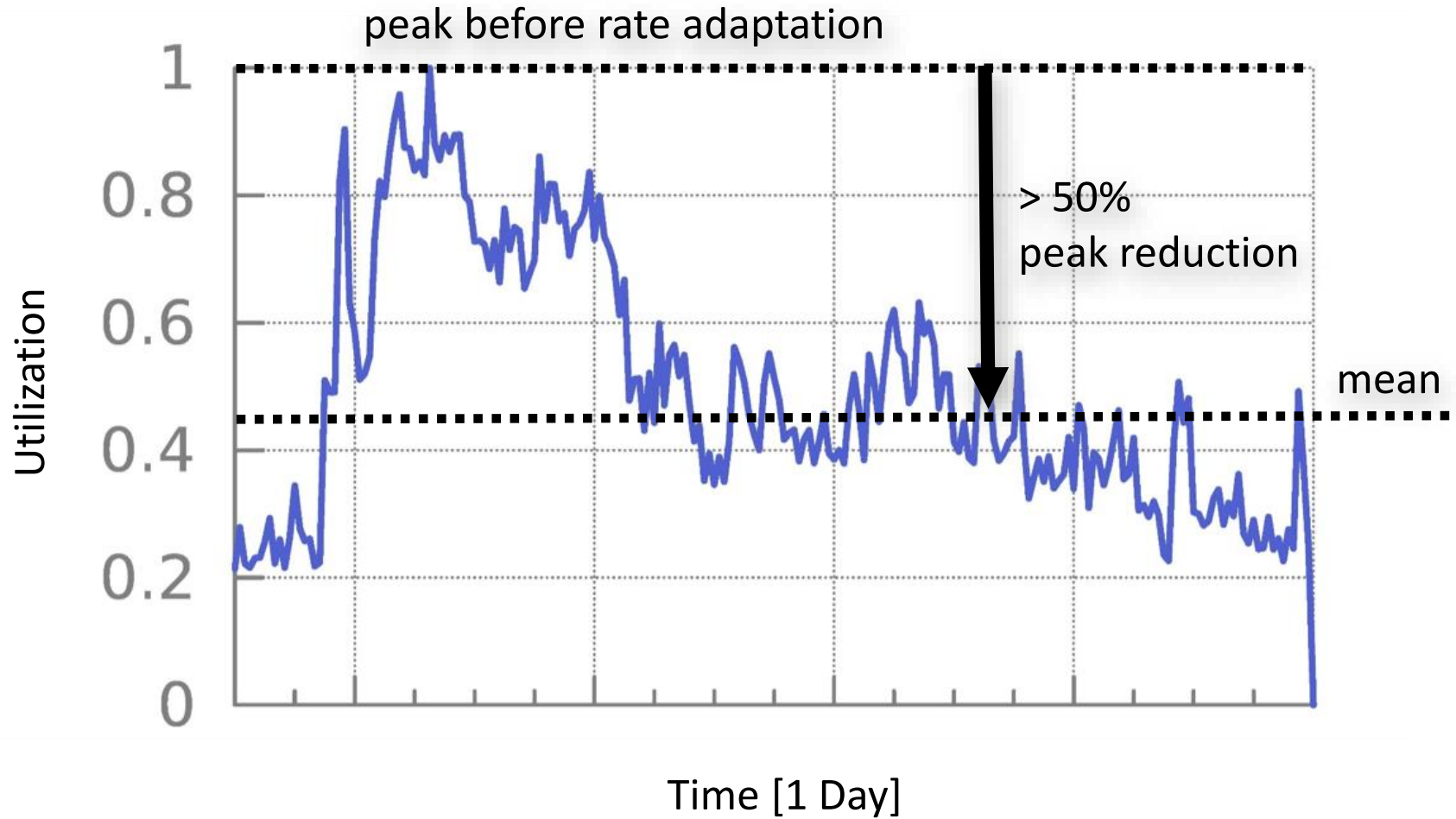
# Real Application: Inter-Data Center WANs



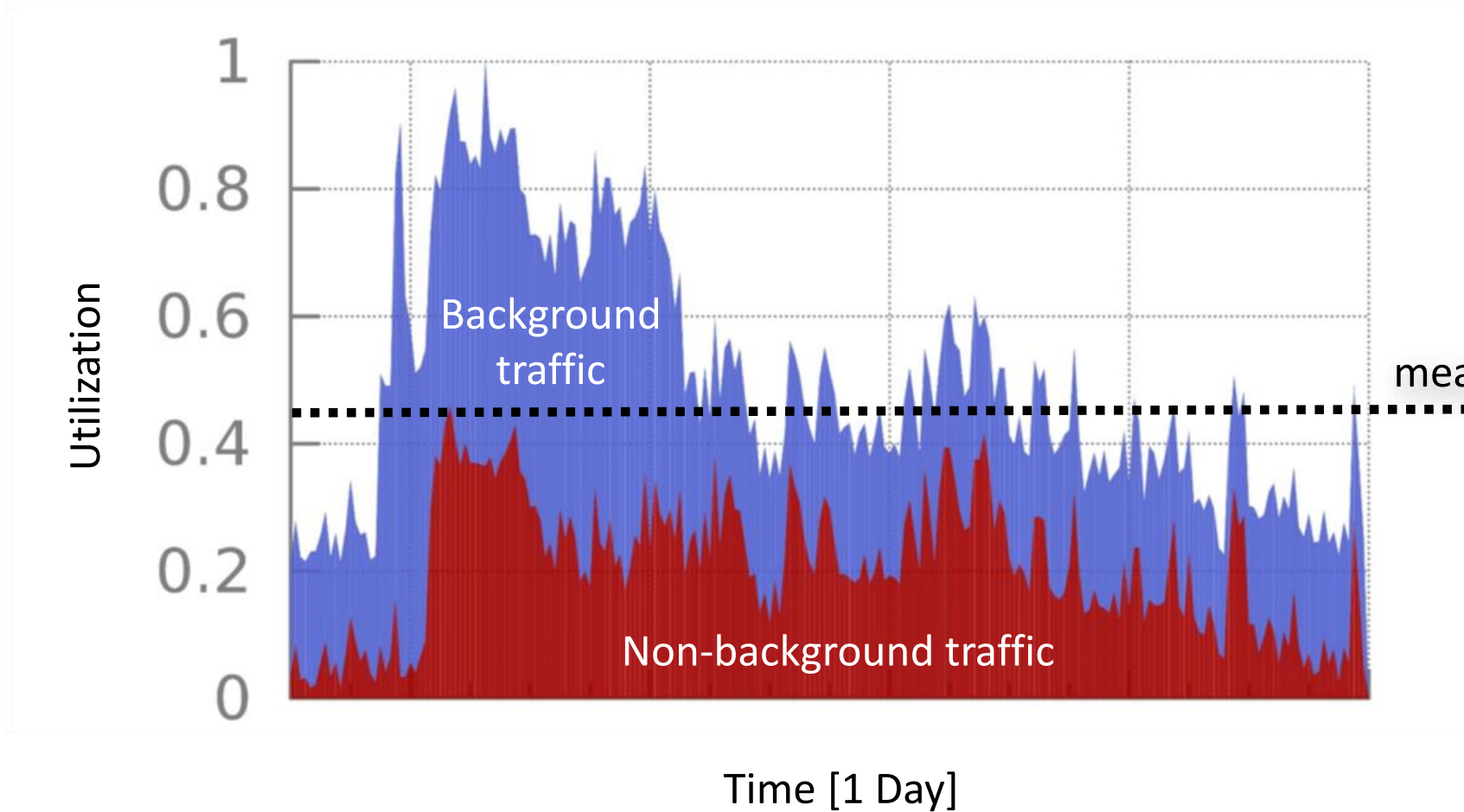
Think: Google, Amazon, Microsoft



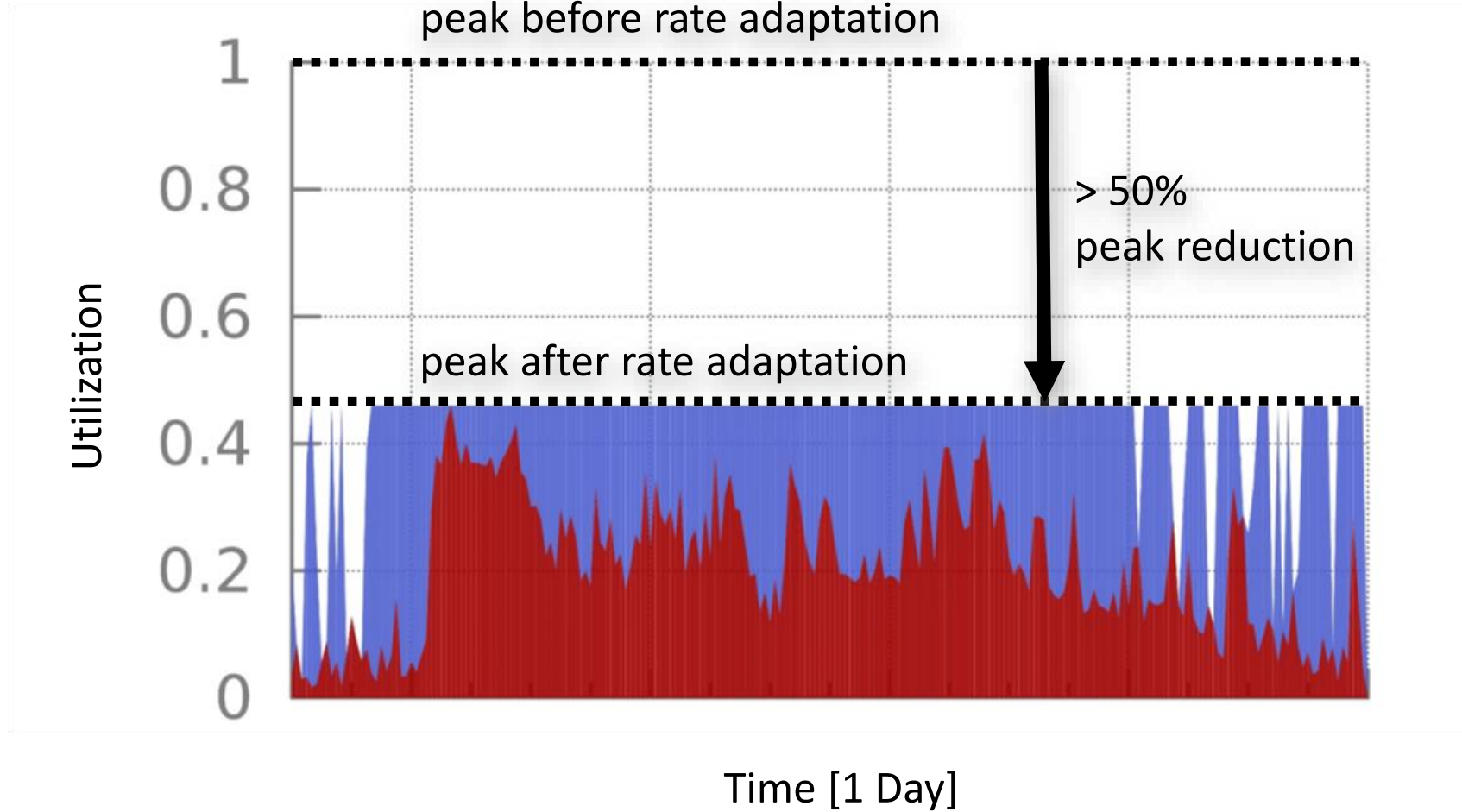
# Problem: Typical Network Utilization



# Problem: Typical Network Utilization



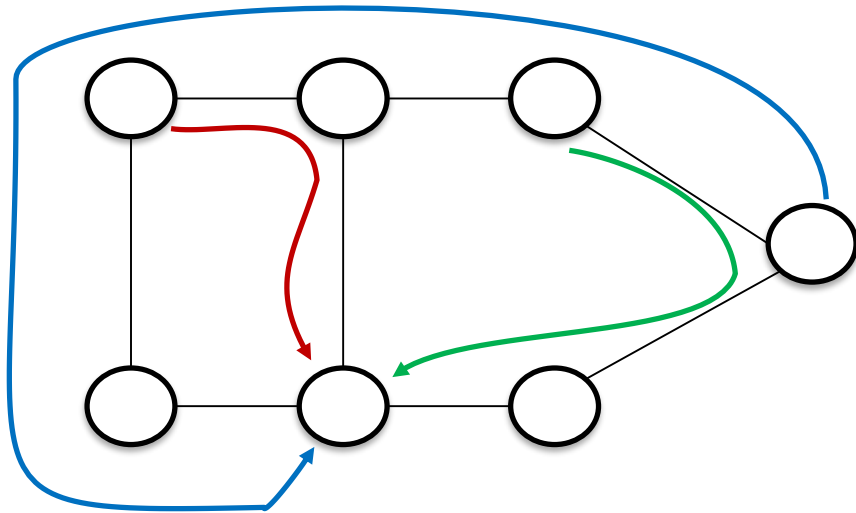
# Problem: Typical Network Utilization



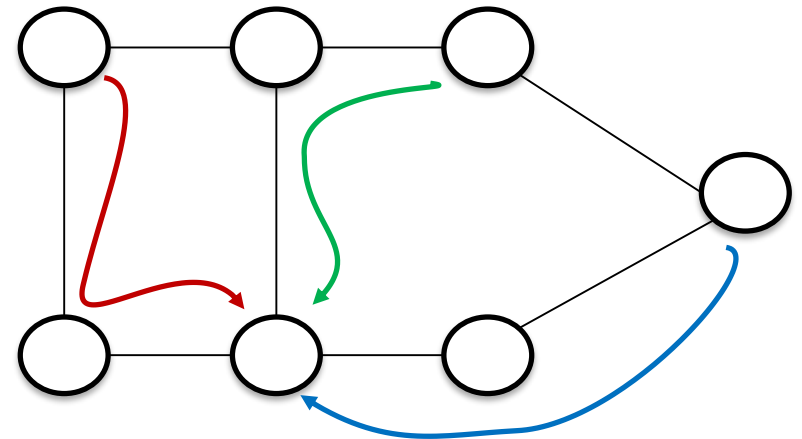
## Another Problem: Online Routing Decisions

flow arrival order: A, B, C

each link can carry at most one flow (in both directions)



MPLS-TE

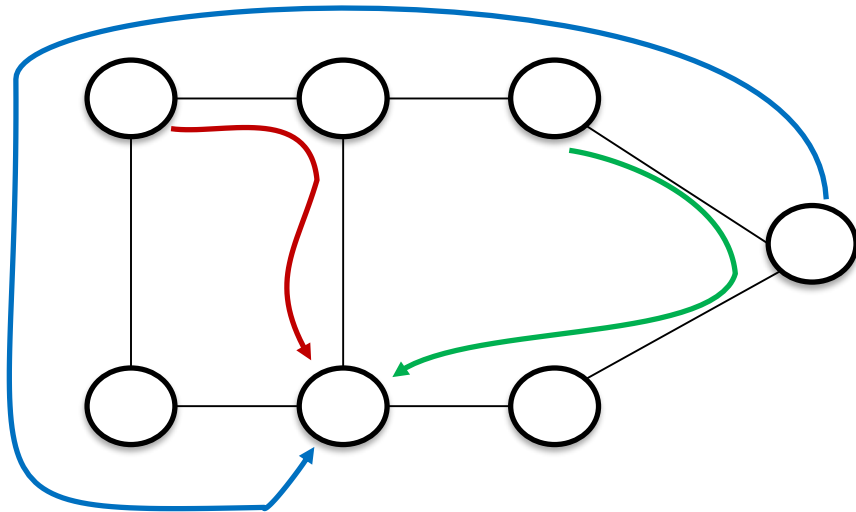


Better

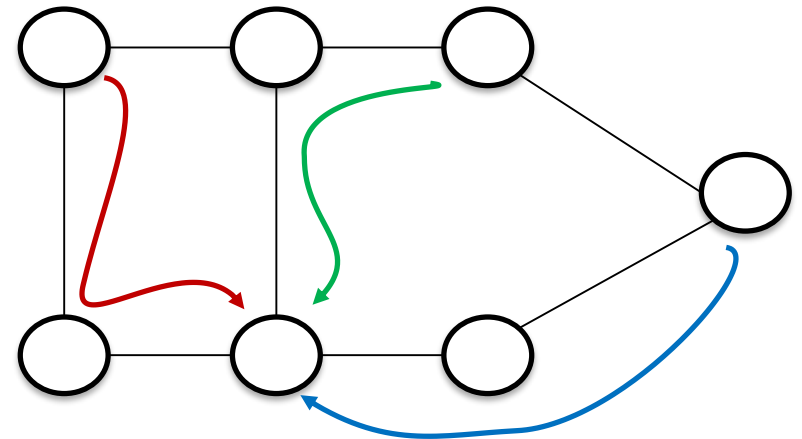
## Another Problem: Online Routing Decisions

flow arrival order: A, B, C

each link can carry at most one flow (in both directions)



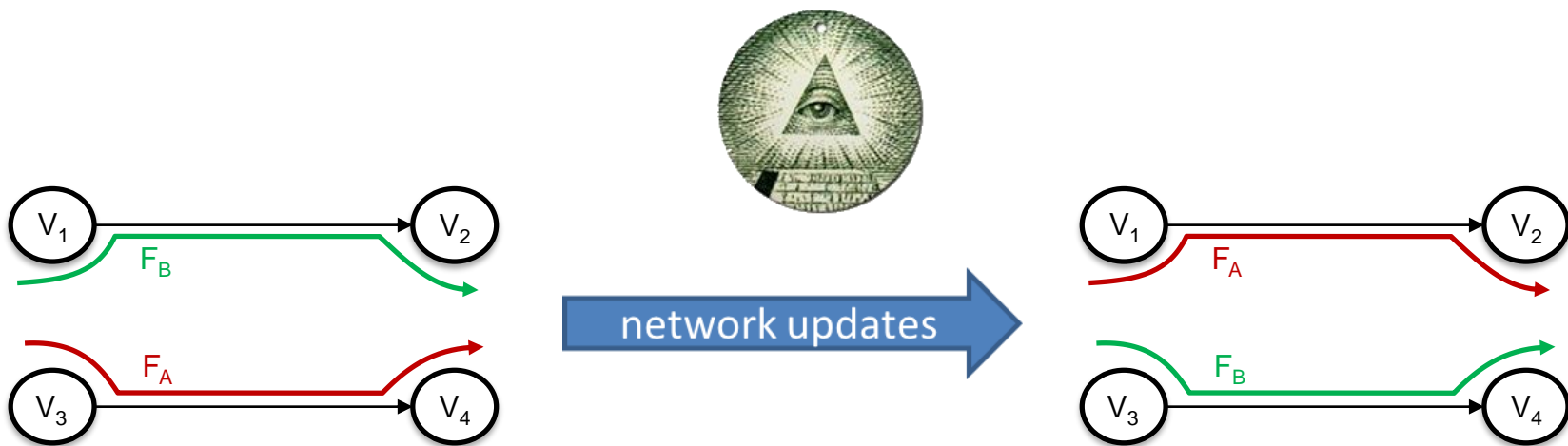
MPLS-TE



Better

How to move flows?

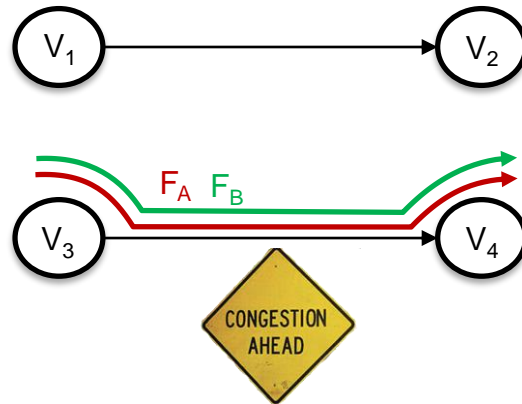
# Introductory Example



size of each flow: 2  
capacity of links: 3

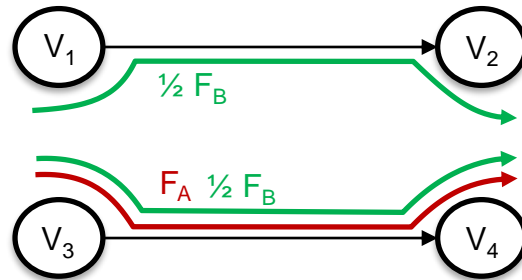


# Just Switch? Congestion!



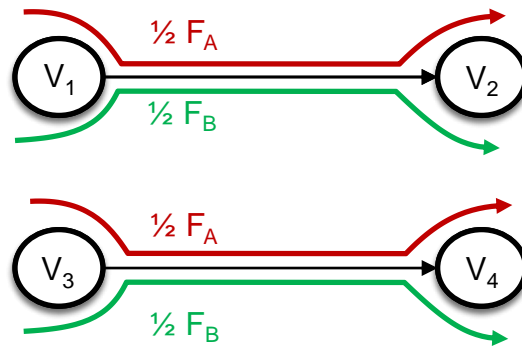
size of each flow: 2  
capacity of links: 3

# Migrate only parts of the flow



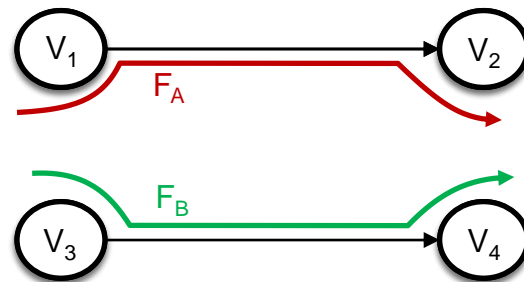
size of each flow: 2  
capacity of links: 3

Can even do both flows at once



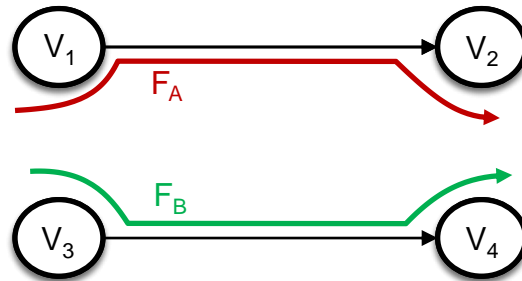
size of each flow: 2  
capacity of links: 3

Done in two steps



size of each flow: 2  
capacity of links: 3

Done in two steps

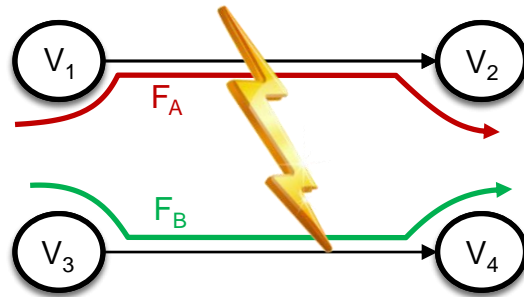


If all links have a slack of  $x$ , then  $-1+1/(x)$  steps

E.g., 20% free capacity everywhere?  $-1+1/(1/5)=4$  steps

size of each flow: 2  
capacity of links: 3

But not always possible!



size of each flow: 2  
capacity of links: **2**

# Two-fold approach of SWAN



$$\begin{aligned}
 \text{Inputs: } & \left\{ \begin{array}{ll} q, & \text{sequence length} \\ b_{i,j}^0 = b_{i,j}, & \text{initial configuration} \\ b_{i,j}^q = b'_{i,j}, & \text{final configuration} \\ c_l, & \text{capacity of link } l \\ I_{jl}, & \text{indicates if tunnel } j \text{ using link } l \end{array} \right. \\
 \text{Outputs: } & \{b_{i,j}^a\} \quad \forall a \in \{1, \dots, q\} \text{ if feasible} \\
 \text{maximize} & \quad c_{\text{margin}} // \text{remaining capacity margin} \\
 \text{subject to} & \quad \forall i, a : \sum_j b_{i,j}^a = b_i; \\
 & \quad \forall l, a : c_l \geq \sum_{i,j} \max(b_{i,j}^a, b_{i,j}^{a+1}) \cdot I_{j,l} + c_{\text{margin}}; \\
 & \quad \forall (i, j, a) : b_{i,j}^a \geq 0; c_{\text{margin}} \geq 0;
 \end{aligned}$$

Figure 7: LP to find if a congestion-free update sequence of length  $q$  exists.

a) free capacity on every link

b) LP-based search

# Two-fold approach of SWAN



$$\begin{aligned}
 \text{Inputs: } & \left\{ \begin{array}{ll} q, & \text{sequence length} \\ b_{i,j}^0 = b_{i,j}, & \text{initial configuration} \\ b_{i,j}^q = b'_{i,j}, & \text{final configuration} \\ c_l, & \text{capacity of link } l \\ I_{jl}, & \text{indicates if tunnel } j \text{ using link } l \end{array} \right. \\
 \text{Outputs: } & \{b_{i,j}^a\} \quad \forall a \in \{1, \dots, q\} \text{ if feasible} \\
 \text{maximize} & \quad c_{\text{margin}} // \text{remaining capacity margin} \\
 \text{subject to} & \quad \forall i, a : \sum_j b_{i,j}^a = b_i; \\
 & \quad \forall l, a : c_l \geq \sum_{i,j} \max(b_{i,j}^a, b_{i,j}^{a+1}) \cdot I_{j,l} + c_{\text{margin}}; \\
 & \quad \forall (i, j, a) : b_{i,j}^a \geq 0; c_{\text{margin}} \geq 0;
 \end{aligned}$$

Figure 7: LP to find if a congestion-free update sequence of length  $q$  exists.

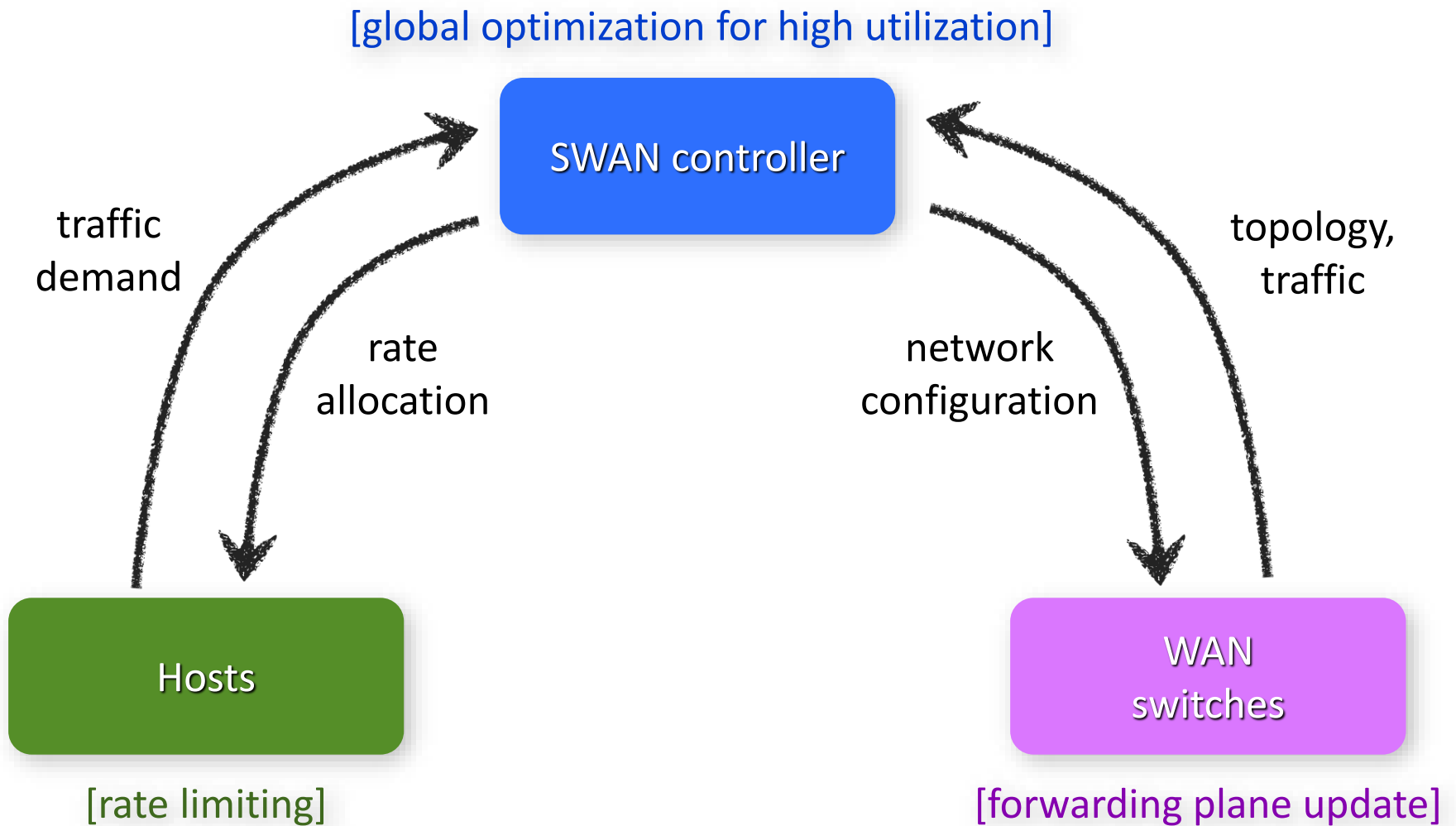
a) free capacity on every link

b) LP-based search

Note: The SWAN framework does much more!



# The SWAN Project



# Do proper network updates exist?

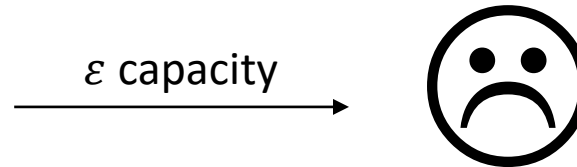


**Inputs:**  $\begin{cases} q, & \text{sequence length} \\ b_{i,j}^0 = b_{i,j}, & \text{initial configuration} \\ b_{i,j}^q = b_{i,j}, & \text{final configuration} \\ c_l, & \text{capacity of link } l \\ I_{j,l}, & \text{indicates if tunnel } j \text{ using link } l \end{cases}$   
**Outputs:**  $\{b_{i,j}^a\} \forall a \in \{1, \dots, q\}$  if feasible  
 maximize  $c_{\text{margin}} // \text{remaining capacity margin}$   
 subject to  $\forall i, a: \sum_j b_{i,j}^a = b_i;$   
 $\forall l, a: c_l \geq \sum_{i,j} \max(b_{i,j}^a, b_{i,j}^{a+1}) \cdot I_{j,l} + c_{\text{margin}};$   
 $\forall (i, j, a): b_{i,j}^a \geq 0; c_{\text{margin}} \geq 0;$



Figure 7: LP to find if a congestion-free update sequence of length  $q$  exists.

# Number of steps can be unbounded

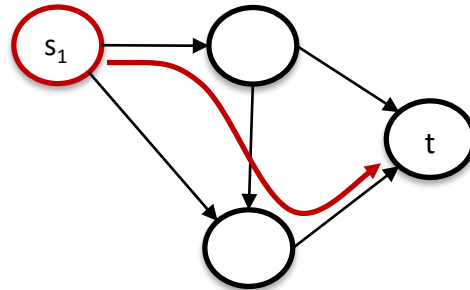


Calculate for an infinite amount of time?

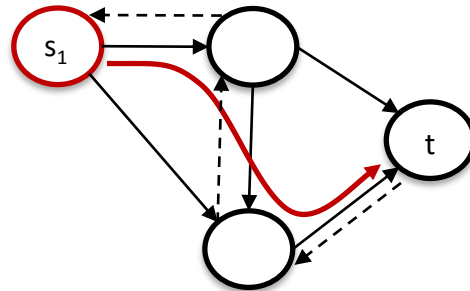
# An old method for a new problem

- Key observation in SWAN:
  - only migrate flows to links with free capacity
- However, LPs do not seem to be the way to go
- Other method: Augmenting flows!
  - “push back” flows to free link capacity

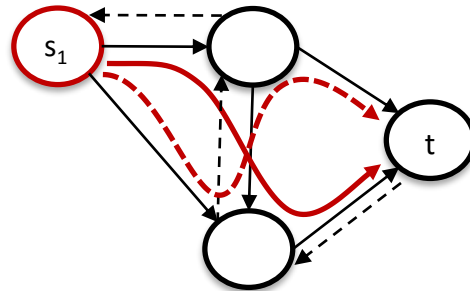
# Short introduction to augmenting flows



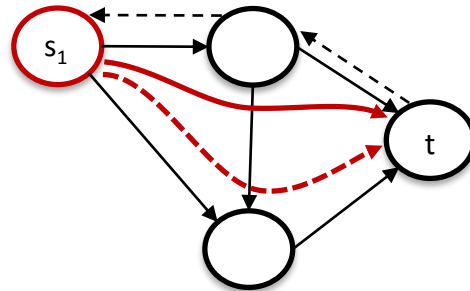
# Consider the residual network too



Now we can find a new flow

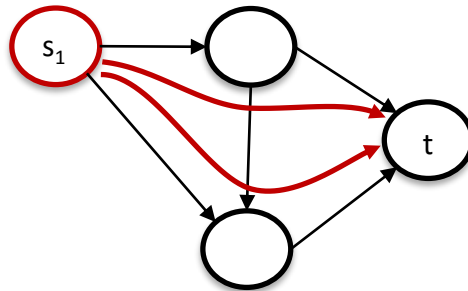


# Push back the old flow





# And insert the new flow



some edge can be reduced from full capacity



a augmenting path exists that creates slack on some full edge\*

thus, we can decide in polynomial time 😊

\*not necessarily the same

# Recap of the situation

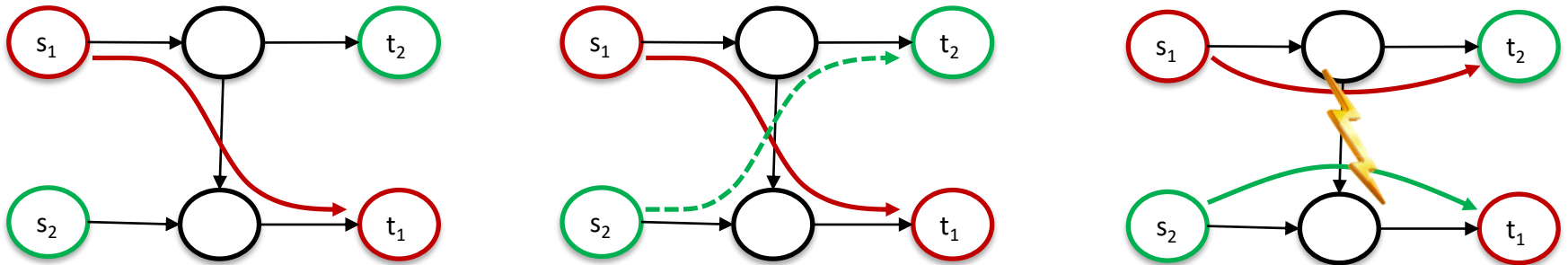
- the **good**: deciding and finding a schedule is fast
  - by creating slack everywhere, if possible
  - let us keep the speed that way 😊
- the **bad**: fastest schedule can be arbitrarily long
  - limit them to linear time!
  - idea: we choose where to put flows
  - lets use augmenting paths again

# Flow augmentation for many destinations

- **Advantage:** Flows are only re-routed along free paths!

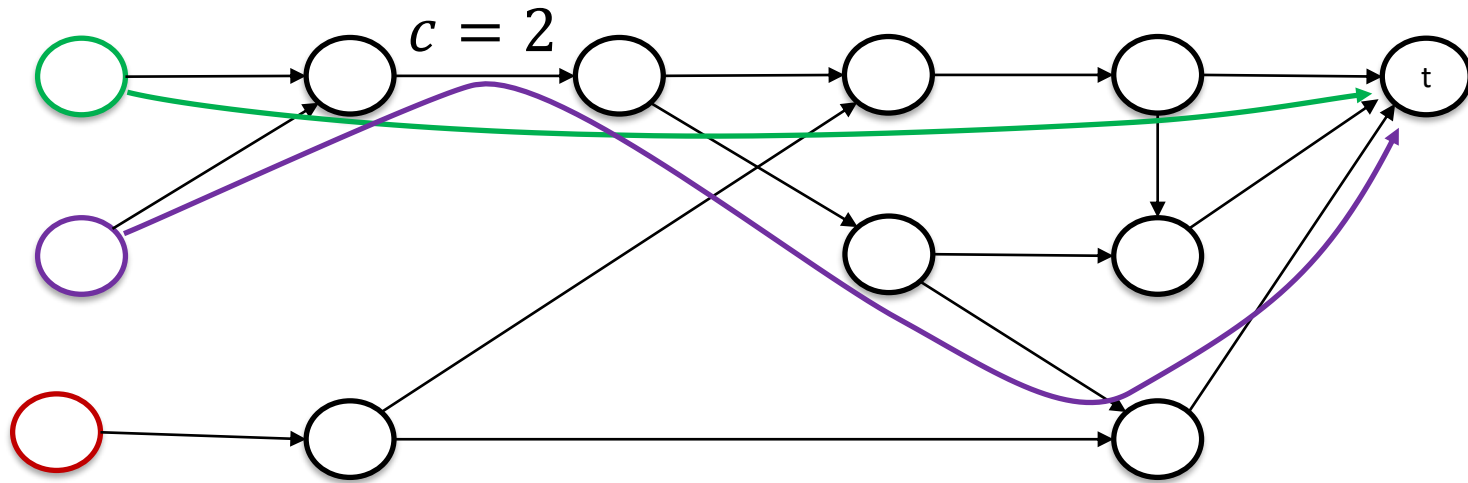
# Flow augmentation for many destinations

- **Advantage:** Flows are only re-routed along free paths!



- **Downside:** Flows end up at the wrong destination!
- So let's stick with one destination for now
  - E.g., a server in another network with multiple entry-points

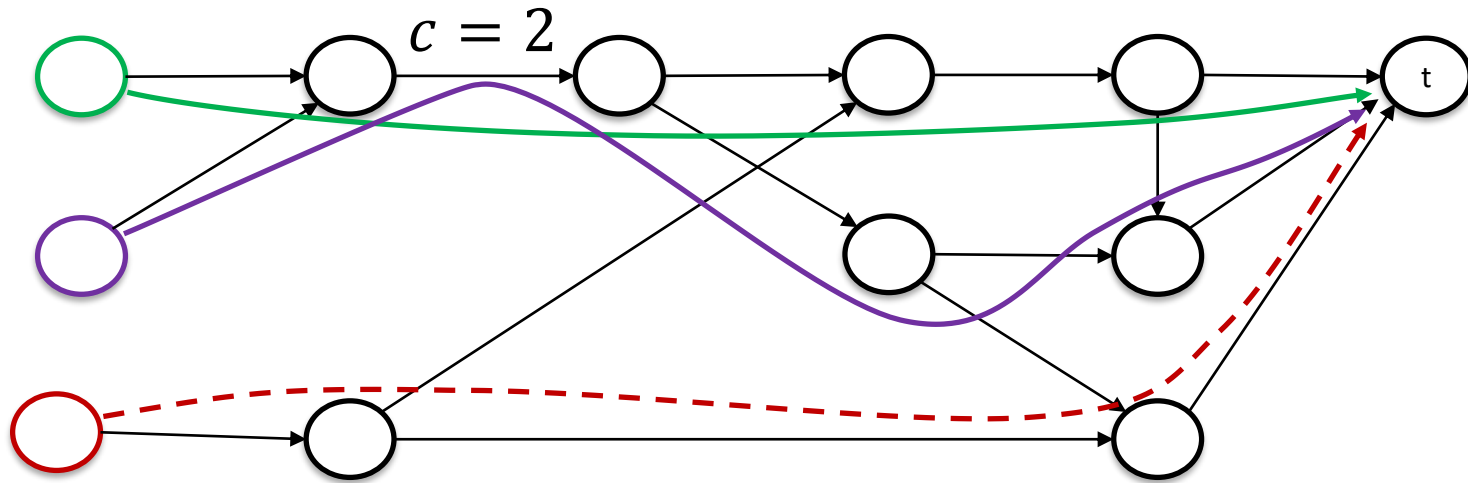
# No free path to the destination



size of each flow: 1  
capacity of each links: 1

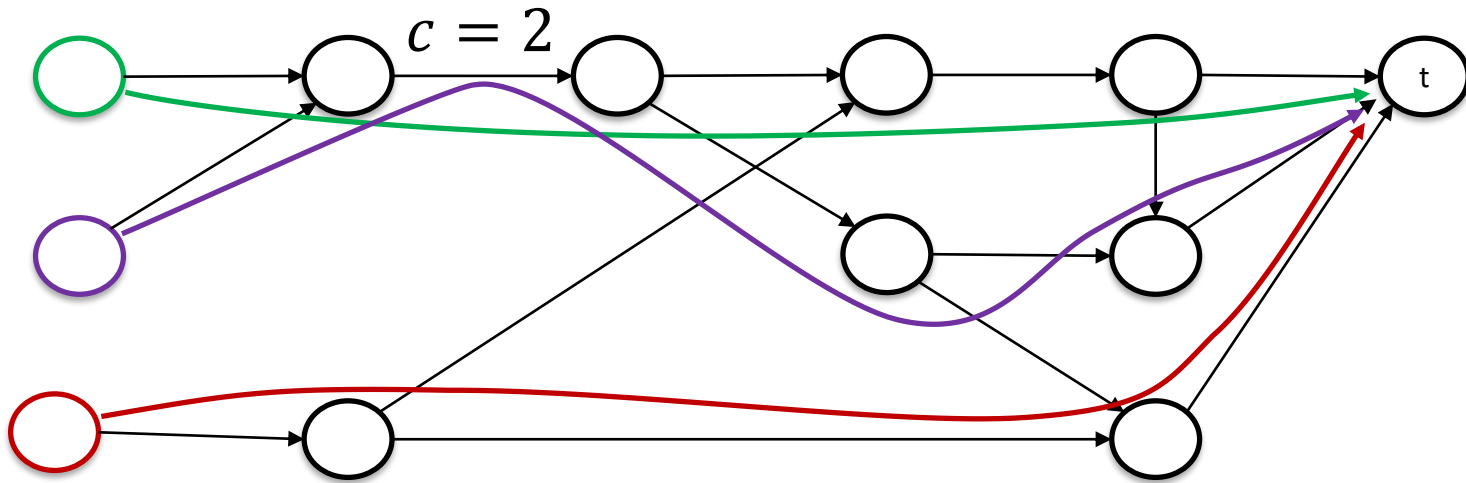


# Old flows get re-routed



size of each flow: 1  
capacity of each links: 1

# And new flow inserted

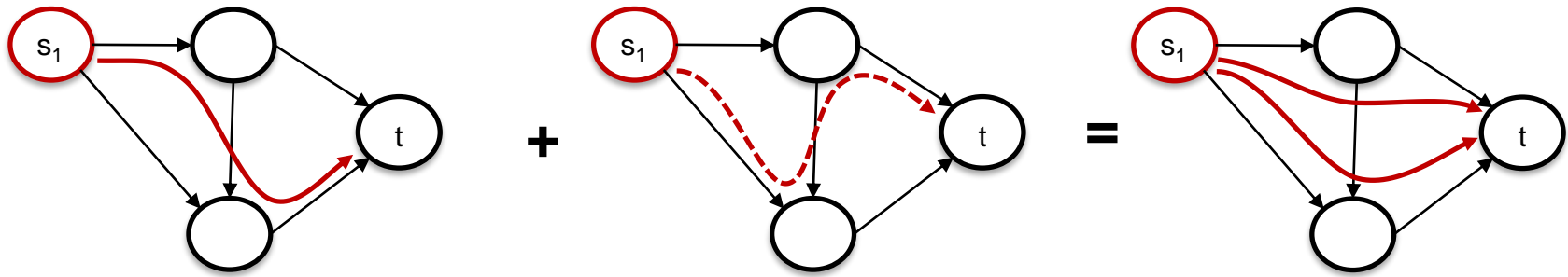


size of each flow: 1  
capacity of each links: 1



# High-level mechanism idea

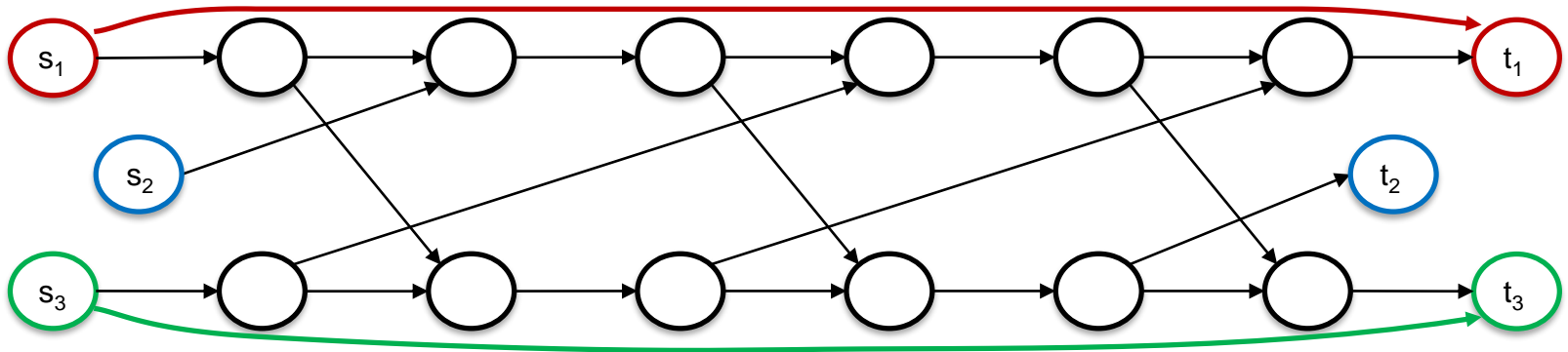
1. *Difference* between two flows  $\rightarrow$  augmenting flow



# High-level mechanism idea

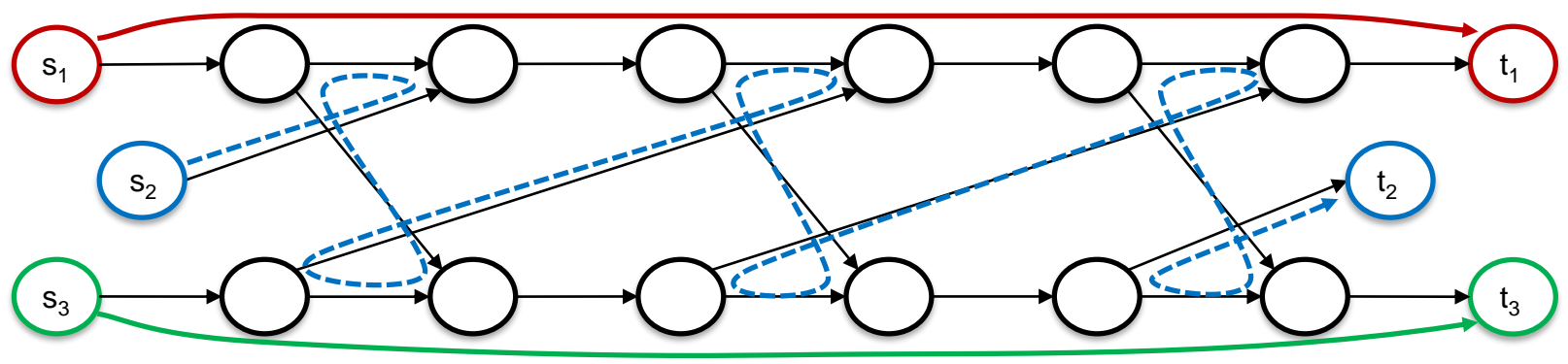
1. *Difference* between two flows  $\rightarrow$  augmenting flow
2. Calculate desired flow sizes with LP
  - *offline* computation
3. Apply augmenting flow for each commodity
  - linear # re-routing *in the network*

# Extension beyond one logical destination?



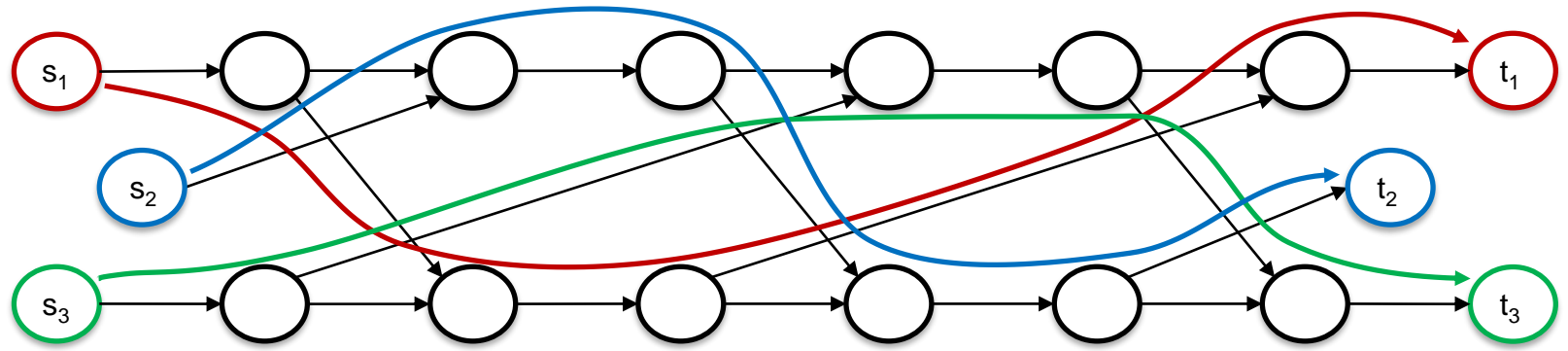
size of each flow: 1  
capacity of each links: 1

# Augmenting flows that don't mix up the destinations?



size of each flow: 1  
capacity of each links: 1

# Augmenting flows that don't mix up the destinations?



size of each flow: 1  
capacity of each links: 1



# Capacity-Consistent Updates

- Fast if enough slack everywhere
- Decidable in polynomial time
- Migrate in linear time with one destination
- Open: Extend fast mechanisms beyond one destination



# Summary

<b>Dependencies</b>	<b>None</b>	<b>Self</b>	<b>Downstream subset</b>	<b>Downstream all</b>	<b>Global</b>
<b>Eventual consistency</b>	Always guaranteed				
<b>Blackhole freedom</b>	Impossible	Add before remove			
<b>Loop freedom</b>	Impossible		Rule dep. forest		
<b>Packet coherence</b>	Impossible			Version numbers	
<b>Bandwidth limits</b>	Impossible				Staged partial moves



# References

- Introducing consistent network updates, Reitblatt et al., SIGCOMM 2012
- For minimal dependencies in updates in general, and loop-free updates in particular, see Ratul Mahajan et al., HotNets 2013
- Deciding if a simple update schedule exists is hard was proven in Laurent Vanbever et al., IEEE/ACM Trans. Netw. 2012
  - See also his recent inaugural lecture @ETH: <http://goo.gl/TMZCyg> (watch 14:50 - 19:25, or better yet, the whole video)
- Loop-detection by Tarjan, Depth-first search & linear graph alg., 1972
- Google B4 SDN project, Sushant Jain et. al., SIGCOMM 2013
- SWAN SDN project, Chi-Yao Hong et. al., SIGCOMM 2013
- Deciding if flows can be moved, Sebastian Brandt et al., INFOCOM 2016
- Fast method for one destination, Sebastian Brandt et al., ICDCN 2016

# Thank You!

Questions & Comments?

