SDN+NFV: Algorithmic and Security Challenges

Stefan Schmid

Aalborg University, Denmark & TU Berlin, Germany

SDN+NFV: It's a great time to be a researcher!

Rhone and Arve Rivers,
Switzerland

Credits: George Varghese.

SDN/NFV Opportunities: Programmability, (logical) centralization and virtualization (multi-tenancy).

Some (often read) claims:

- ❑ Simpler
- ❑ More flexible
- ❑ Automatically verifiable
- ❑ And hence also more secure?

SDN/NFV Opportunities: Programmability, (logical) centralization and virtualization (multi-tenancy).

Some (often read) claims:

- ❑ Simpler
- ❑ More fl
- ❑ ma
- ❑ er

RSS fedscoop BROUGHT TO YOU BY SNG

TECH | DEFENSE | ACQUISITION | WATCH | LISTEN | ATTEND | COMMUNITY

DEFENSE

Pentagon considering push to software-defined networking

30 October 2017

SDN/NFV Opportunities: Programmability, (logical) centralization and virtualization (multi-tenancy).

Some (often read) claims:

- ❑ Simpler
- ❑ More flexible
- ❑ Automatically verifiable
- ❑ And hence also more secur

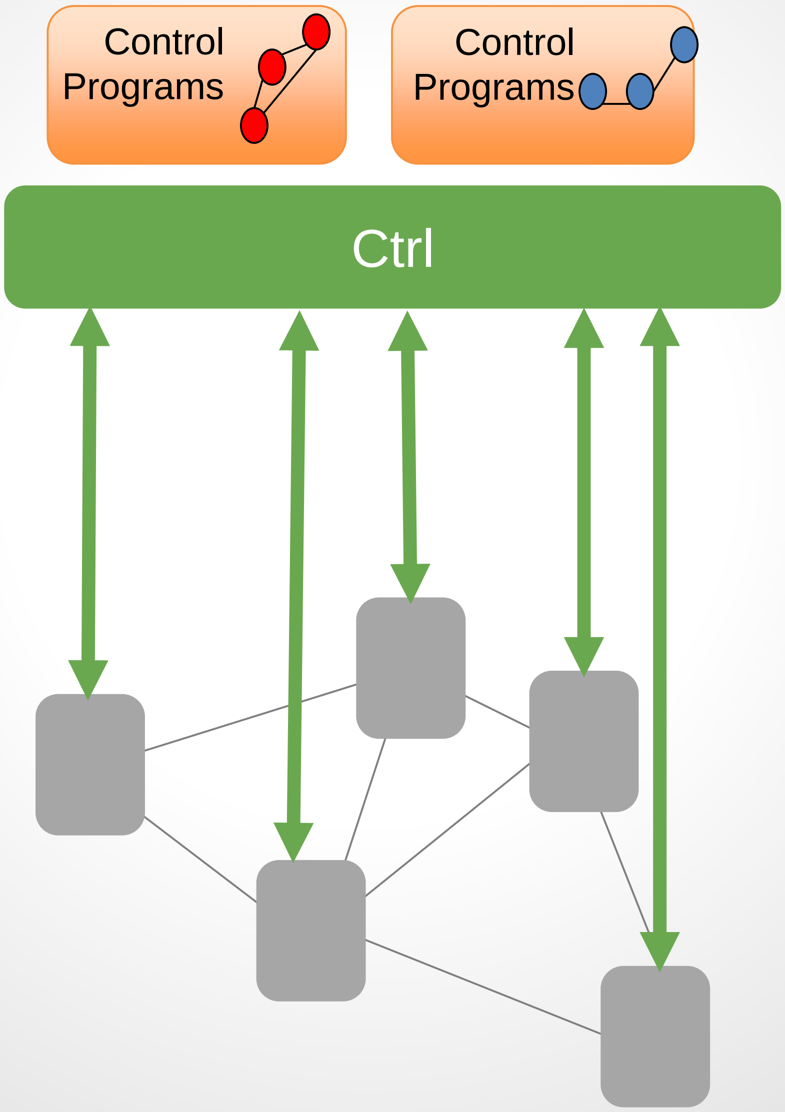
Really?

Algorithms? Avoid instabilities!

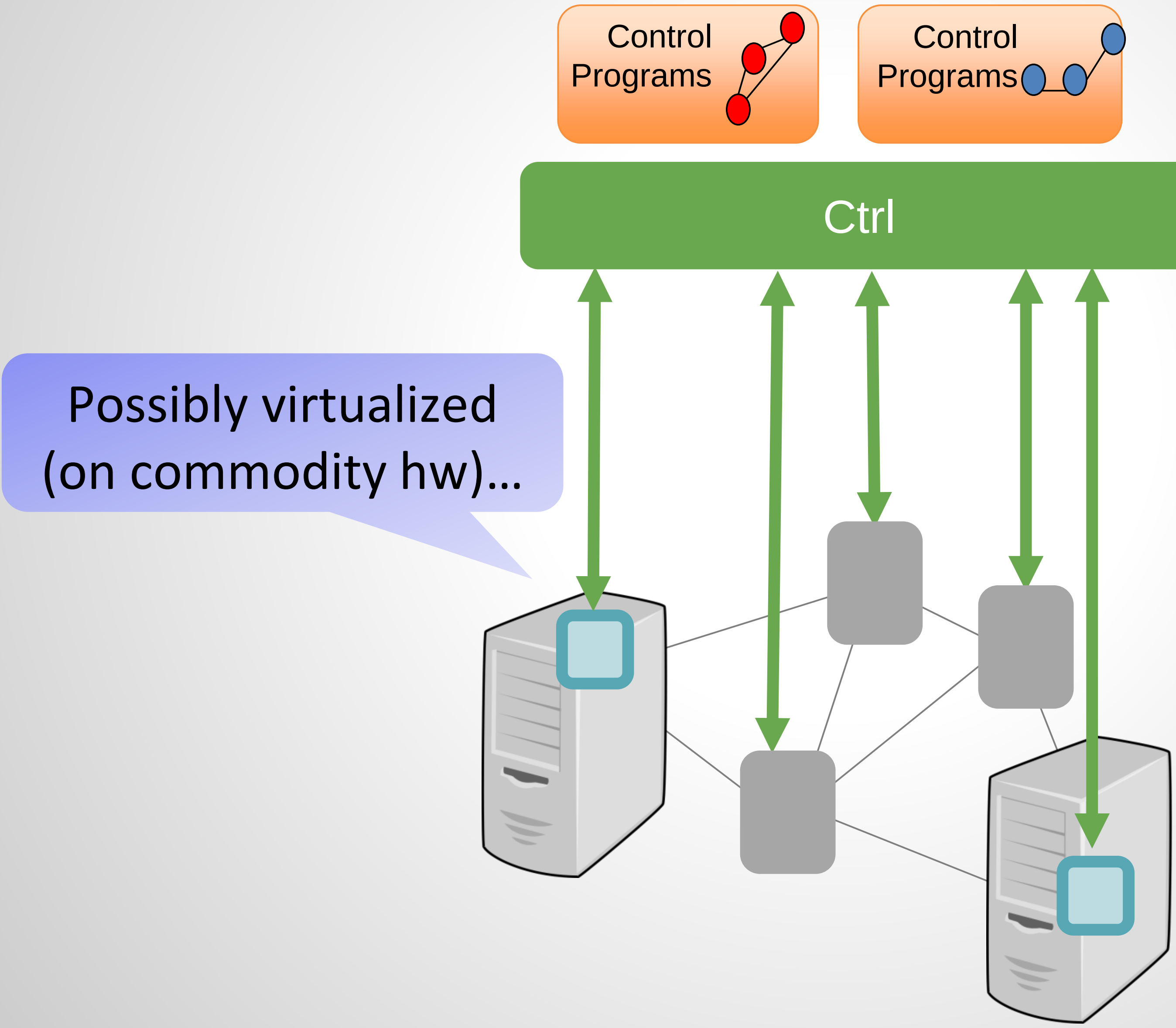
Complexity of this?

New threats?

A Mental Model for This Talk

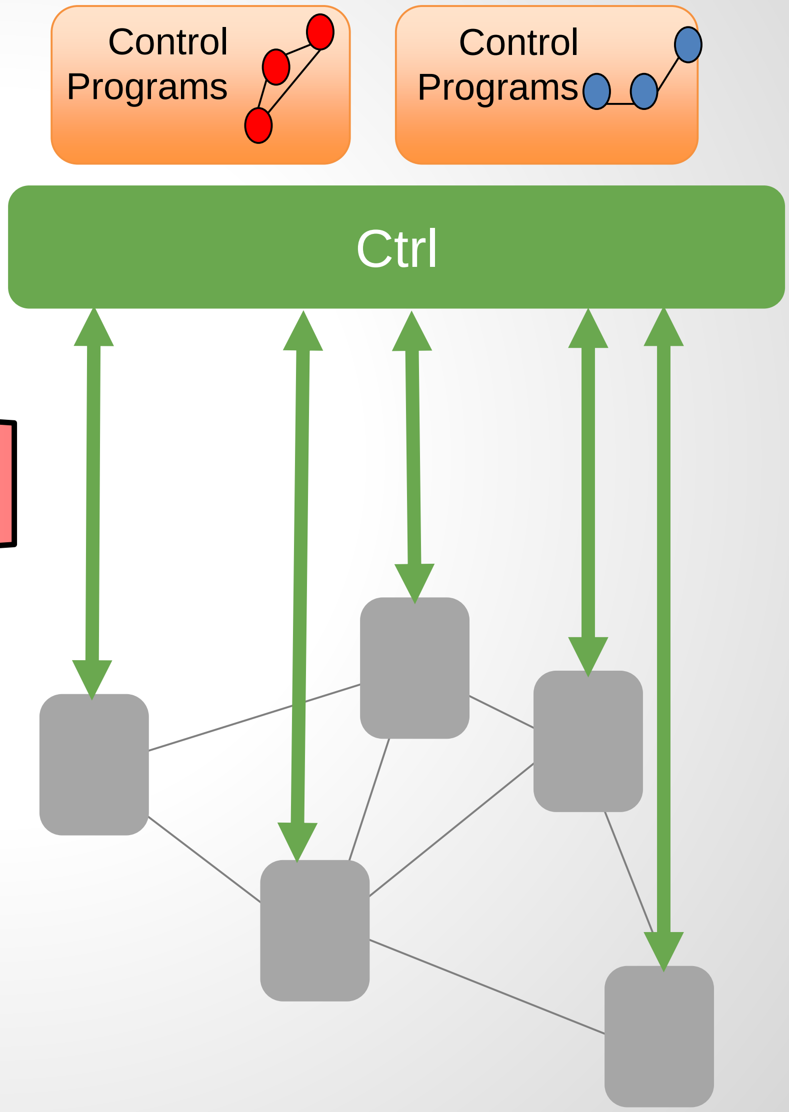


A Mental Model for This Talk



Algorithms

A First (Algorithmic) Challenge: Decoupling



Challenge: centralization and decoupling!

Despite centralization: SDN stays a distributed system!

Recall: Networking 101

- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

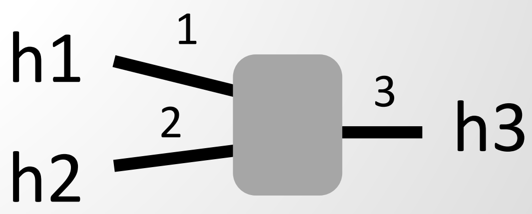
Credits: Jennifer Rexford

Recall: Networking 101

- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

- Example
  - h1 sends to h2:

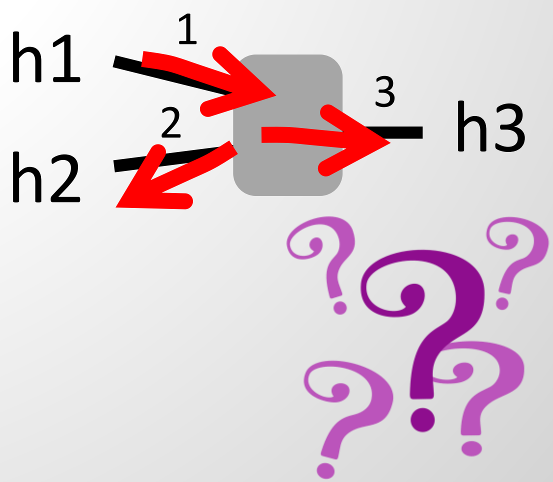


Recall: Networking 101

- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

- Example
  - h1 sends to h2: **flood**



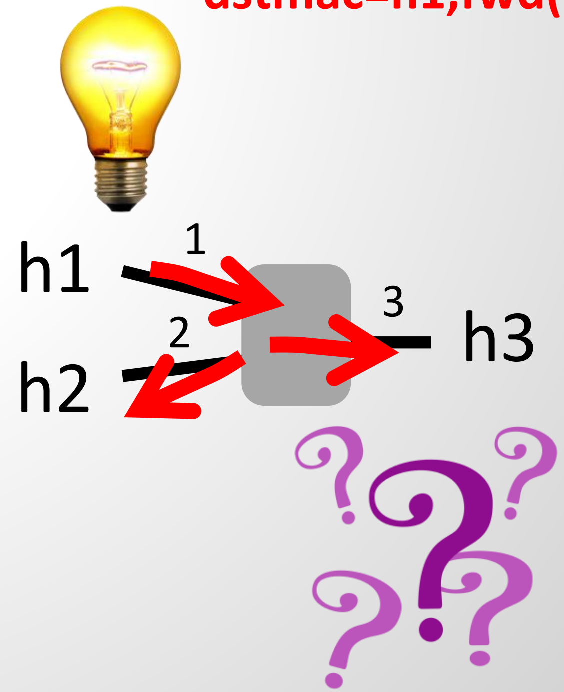
Recall: Networking 101

- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

dstmac=h1,fwd(1)

- Example
  - h1 sends to h2: **flood, learn (h1,p1)**

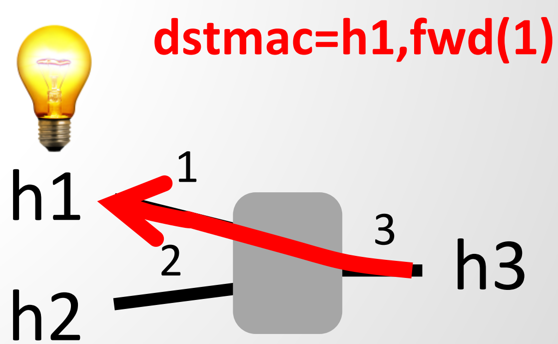


Recall: Networking 101

- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

- Example
  - h1 sends to h2: **flood, learn (h1,p1)**
  - h3 sends to h1: **forward to p1**



Recall: Networking 101

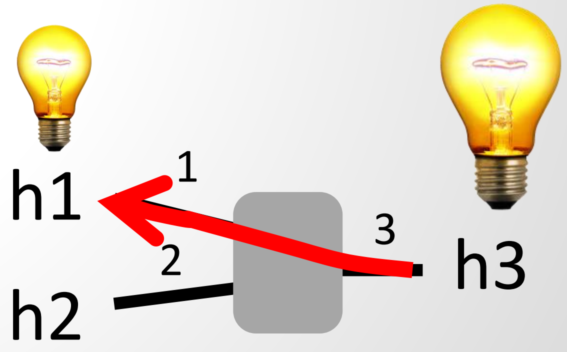
- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

dstmac=h1,fwd(1)

dstmac=h3,fwd(3)

- Example
  - h1 sends to h2: **flood, learn (h1,p1)**
  - h3 sends to h1: **forward to p1, learn (h3,p3)**



Recall: Networking 101

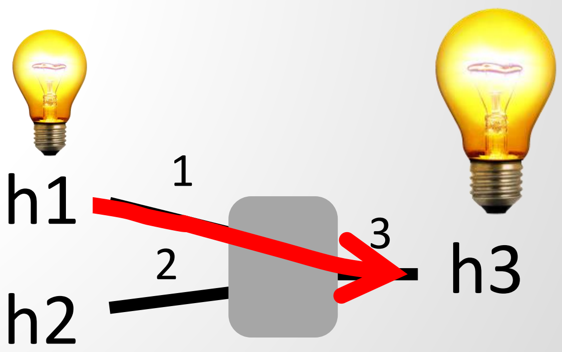
- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

dstmac=h1,fwd(1)

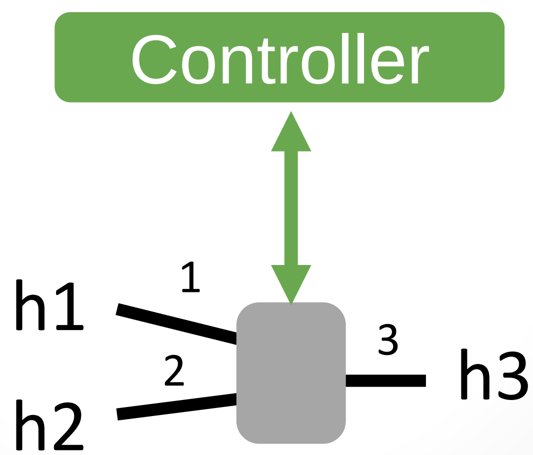
dstmac=h3,fwd(3)

- Example
  - h1 sends to h2: **flood, learn (h1,p1)**
  - h3 sends to h1: **forward to p1, learn (h3,p3)**
  - h1 sends to h3: **forward to p3**

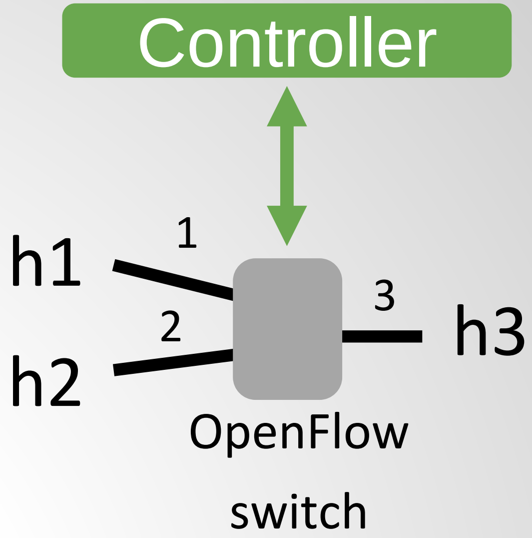


From Traditional Networks to SDN

How to implement this behavior in SDN?



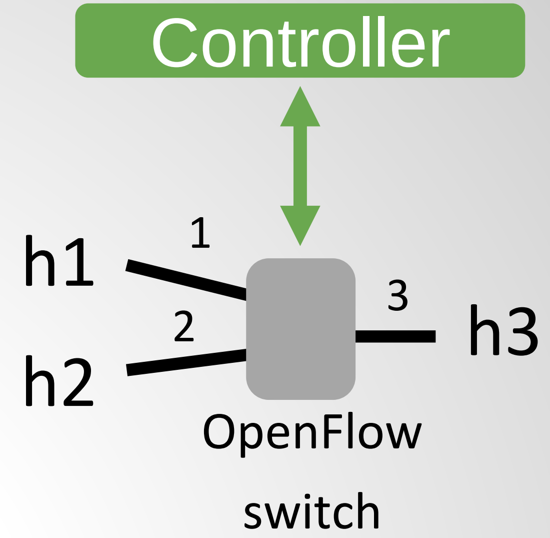
Example: SDN MAC Learning Done Wrong



- Initial table: Send everything to controller

Pattern	Action
*	send to controller

Example: SDN MAC Learning Done Wrong

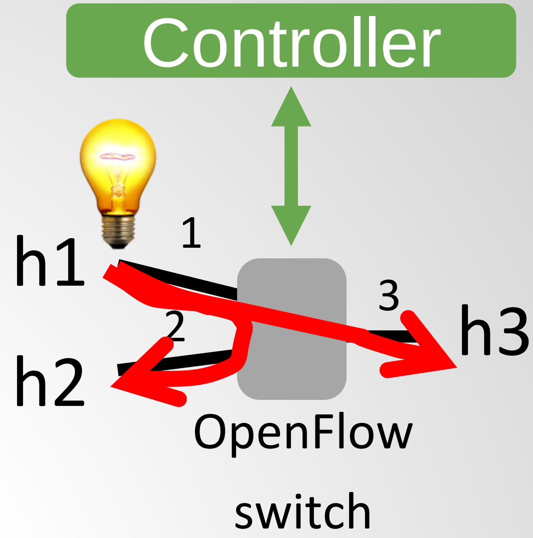


- Initial table: Send everything to controller

Pattern	Action
*	send to controller

- When h1 sends to h2:

Example: SDN MAC Learning Done Wrong



- Principle: only send to ctrl if destination unknown

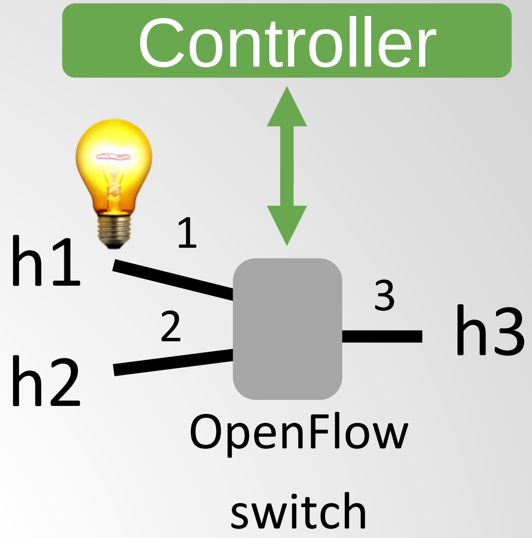
Pattern	Action
*	send to controller

h1 sends to h2

Pattern	Action
dstmac=h1	Forward(1)
*	send to controller

- When h1 sends to h2:
  - Controller learns that h1@p1, updates table, and floods

Example: SDN MAC Learning Done Wrong

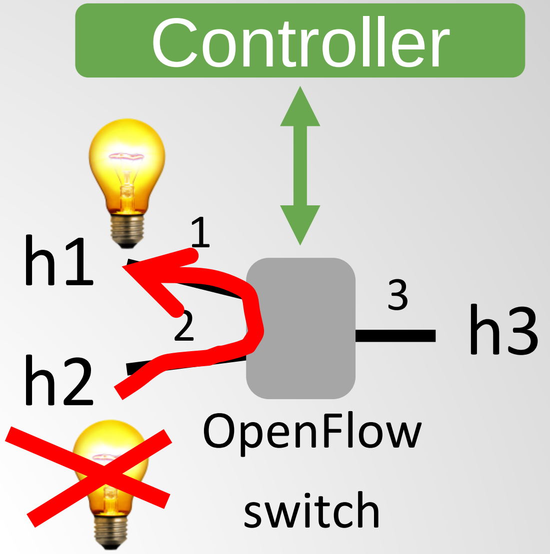


- Principle: only send to ctrl if destination unknown

Pattern	Action
dstmac=h1	Forward(1)
*	send to controller

- Now assume h2 sends to h1:

Example: SDN MAC Learning Done Wrong

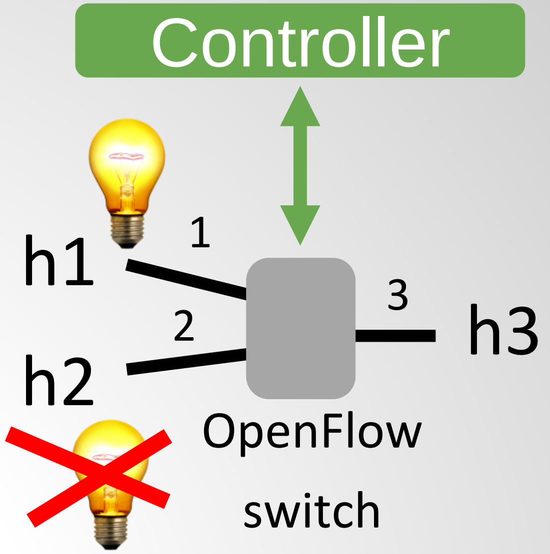


- Principle: only send to ctrl if destination unknown

Pattern	Action
dstmac=h1	Forward(1)
*	send to controller

- Now assume h2 sends to h1:
  - *Switch* knows destination: message forwarded to h1
  - ***BUT***: No controller interaction, does not learn about h2: no new rule for h2

Example: SDN MAC Learning Done Wrong



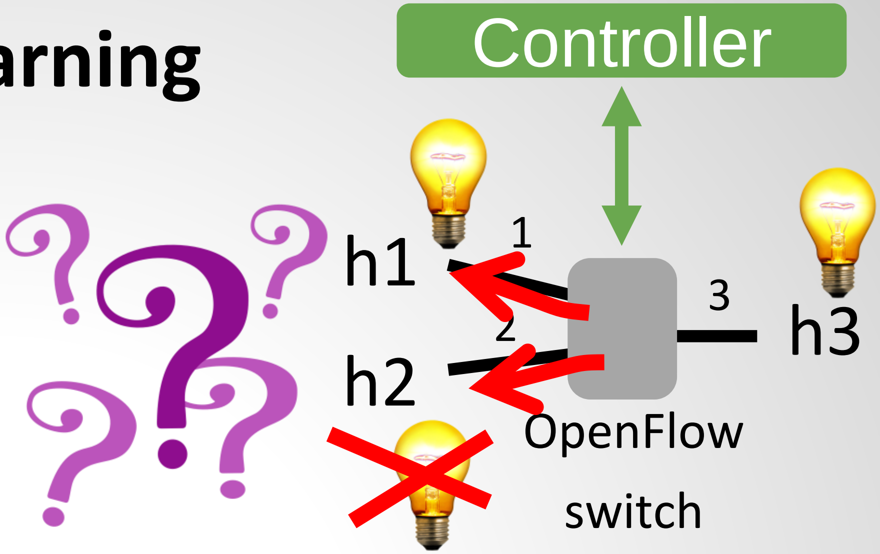
- Principle: only send to ctrl if destination unknown

Pattern	Action
dstmac=h1	Forward(1)
*	send to controller

h3 sends to h2

- Now, when h3 sends to h2:

Example: SDN MAC Learning Done Wrong



- Principle: only send to ctrl if destination unknown

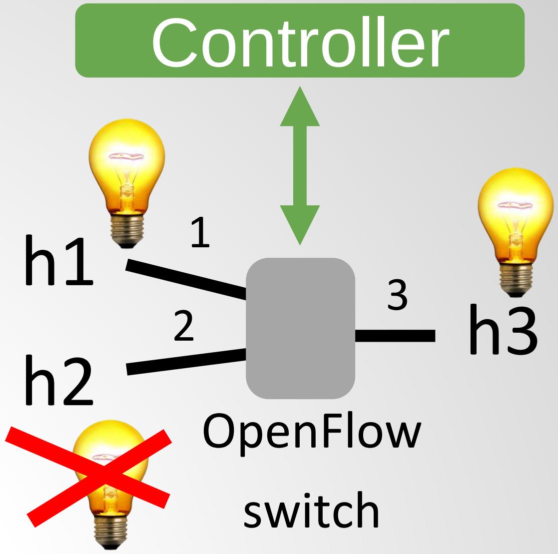
Pattern	Action
dstmac=h1	Forward(1)
*	send to controller

h3 sends to h2

Pattern	Action
dstmac=h3	Forward(3)
dstmac=h1	Forward(1)
*	send to controller

- Now, when h3 sends to h2:
  - Dest unknown: goes to controller which learns about h3
  - And then floods

Example: SDN MAC Learning Done Wrong

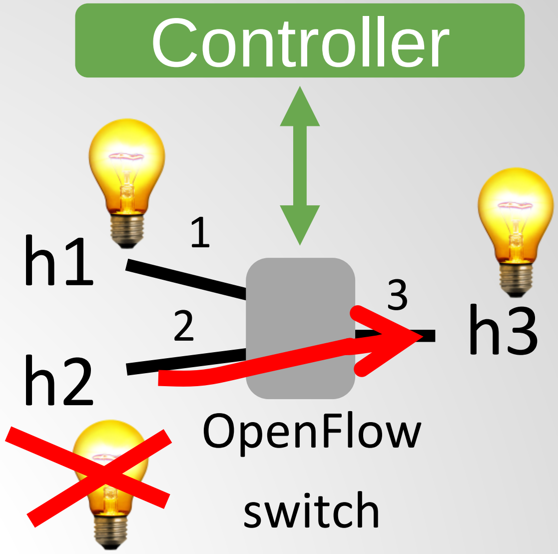


- Principle: only send to ctrl if destination unknown

Pattern	Action
dstmac=h3	Forward(3)
dstmac=h1	Forward(1)
*	send to controller

- Now, if h2 sends to h3 or h1:

Example: SDN MAC Learning Done Wrong

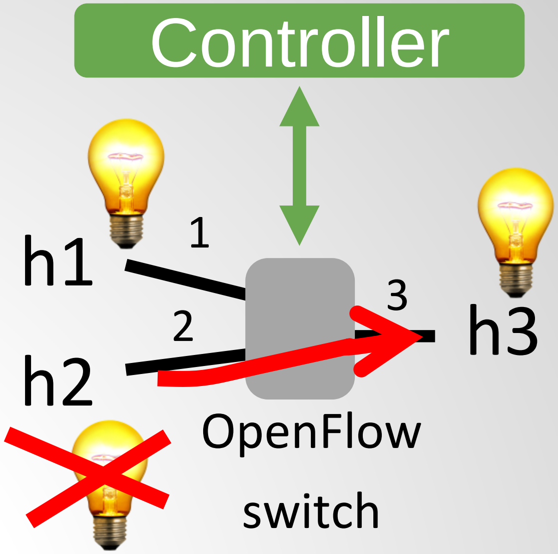


- Principle: only send to ctrl if destination unknown

Pattern	Action
dstmac=h3	Forward(3)
dstmac=h1	Forward(1)
*	send to controller

- Now, if h2 sends to h3 or h1:
  - Destinations known: controller does not learn about h2

Example: SDN MAC Learning Done Wrong

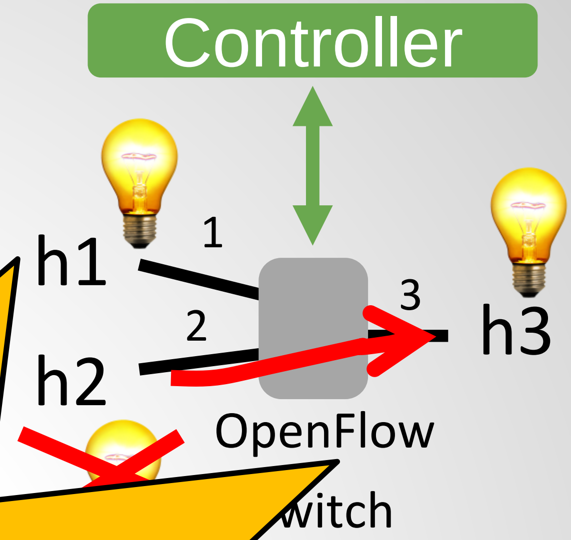


- Principle: only send to ctrl if destination unknown

Pattern	Action
dstmac=h3	Forward(3)
dstmac=h1	Forward(1)
*	send to controller

Ouch! Controller cannot learn about h2 anymore: whenever h2 is source, destination is known. All future requests to h2 will ***all be flooded***: inefficient!

Example: SDN MAC Learning Done Wrong



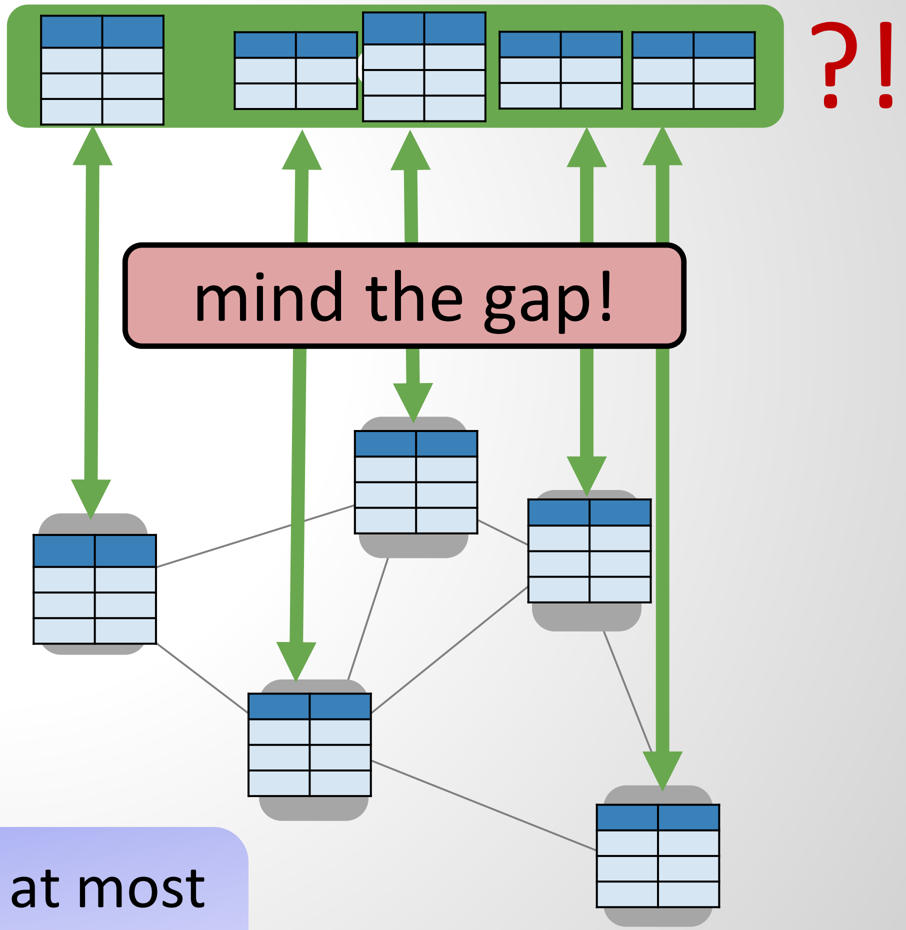
- Principle: only send to ctrl if d ination unkno

How to efficiently detect such problems? And which rules to use to overcome them? An algorithmic problem!

Ouch! Controller c ot learn out h2 anymore: whenever h2 is source destination is known. All future requests to h2 will ***all be flooded***: inefficient!

There Are Many More Reasons Why A Controller May Have Inconsistent View

- ❑ Rules inserted using switch CLI
- ❑ Operator misconfigurations
- ❑ Software/hardware bugs
- ❑ Updates that have been acknowledged wrongfully
- ❑ Malicious behavior, etc.

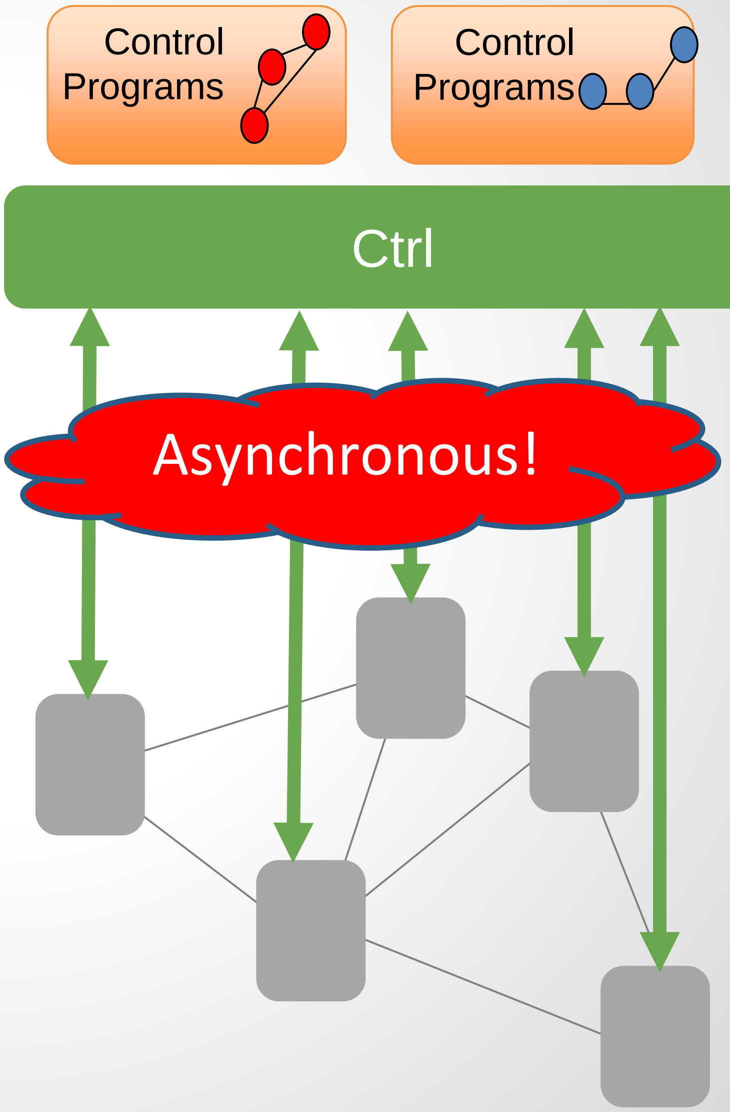


A ***problem*** because ***like in security***: at most as consistent as least consistent part!

Further Reading

Towards Meticulous Data Plane Monitoring (Poster Paper)
Apoorv Shukla, Said Jawad Saidi, Stefan Schmid, Marco Canini, and Anja Feldmann.
EuroSys PhD Forum, Belgrade, Serbia, April 2017.

Another Challenge Arising From Decoupling

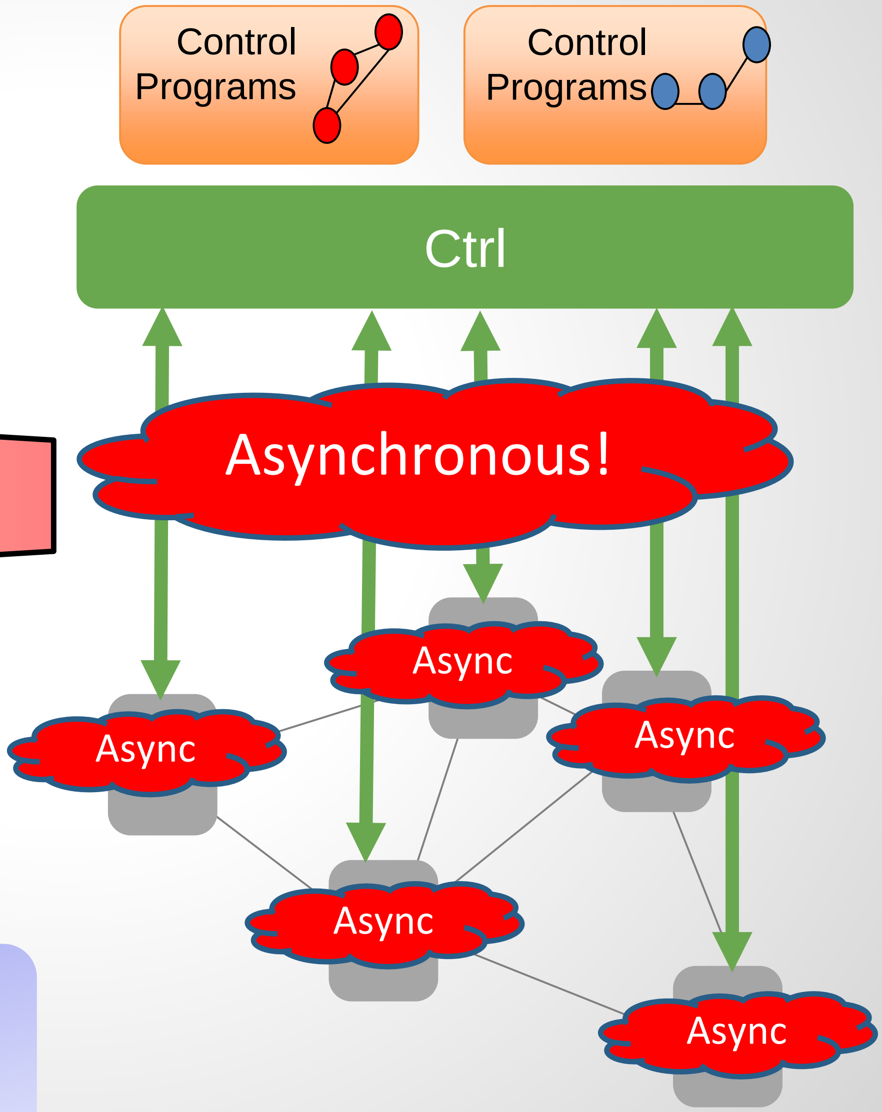


Challenge: Decoupling

Despite centralization: SDN stays a distributed system!

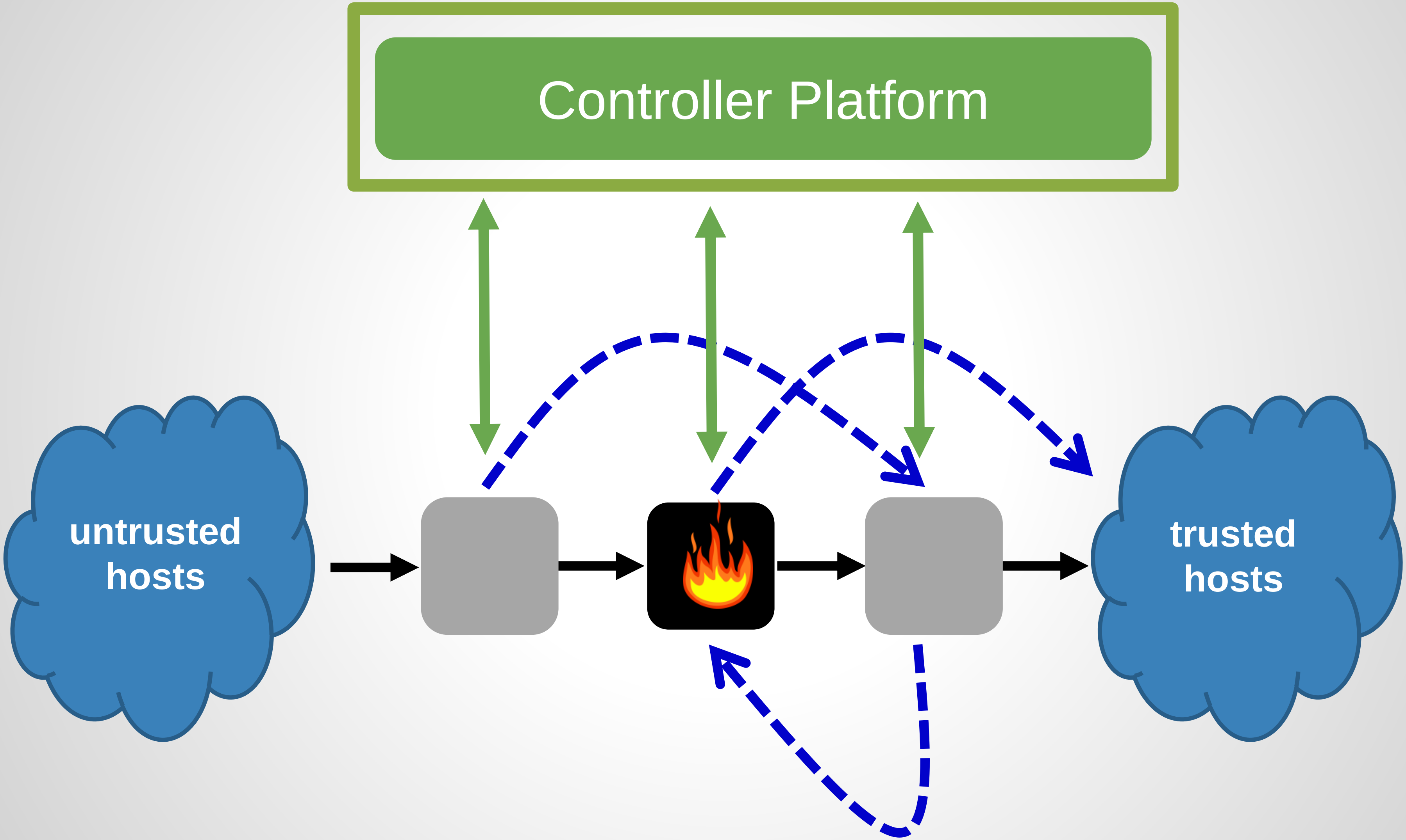
Another Challenge Arising From Decoupling

Challenge: Decoupling



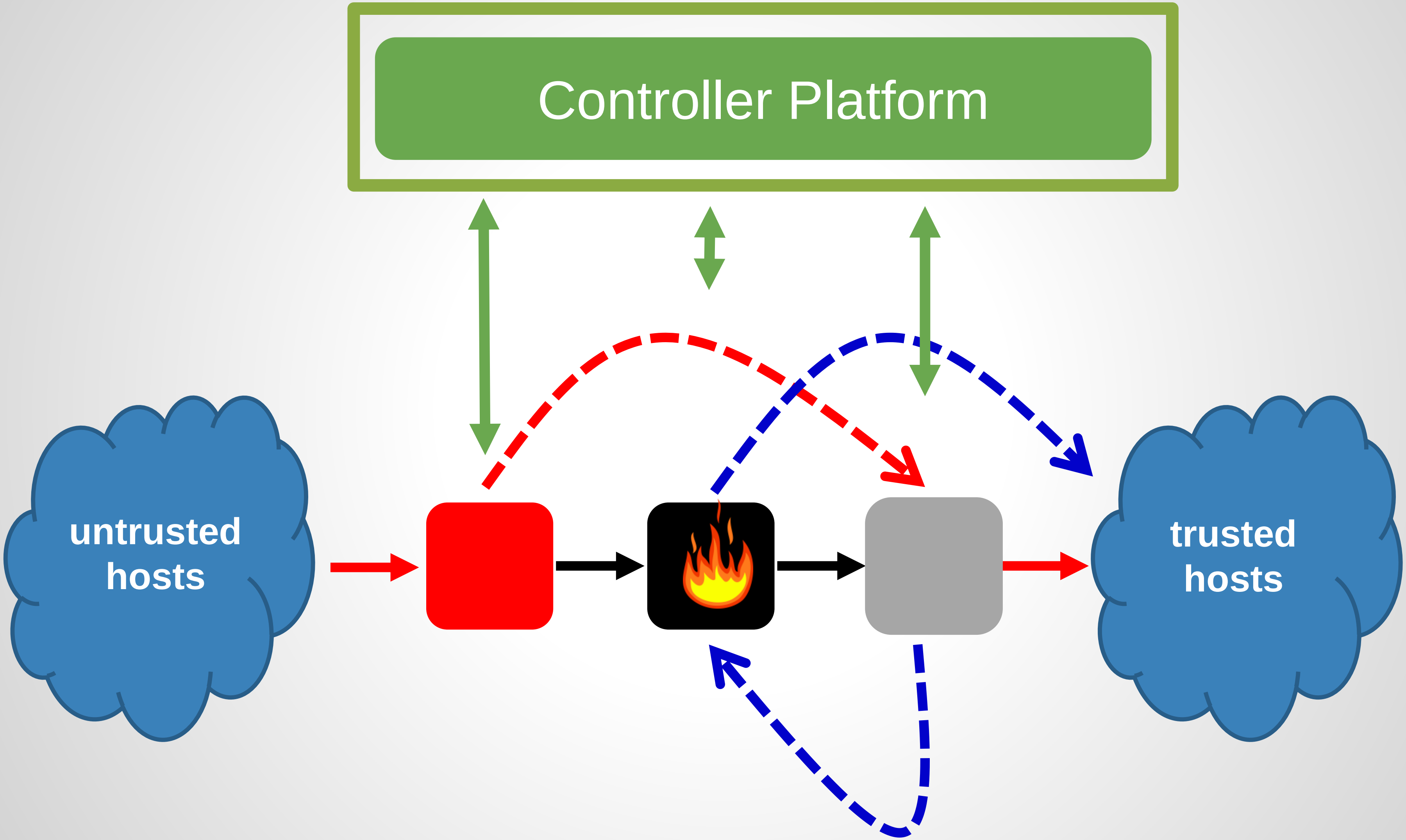
Despite centralization: SDN stays a distributed system!

Example "Route Updates":
What can possibly go wrong?



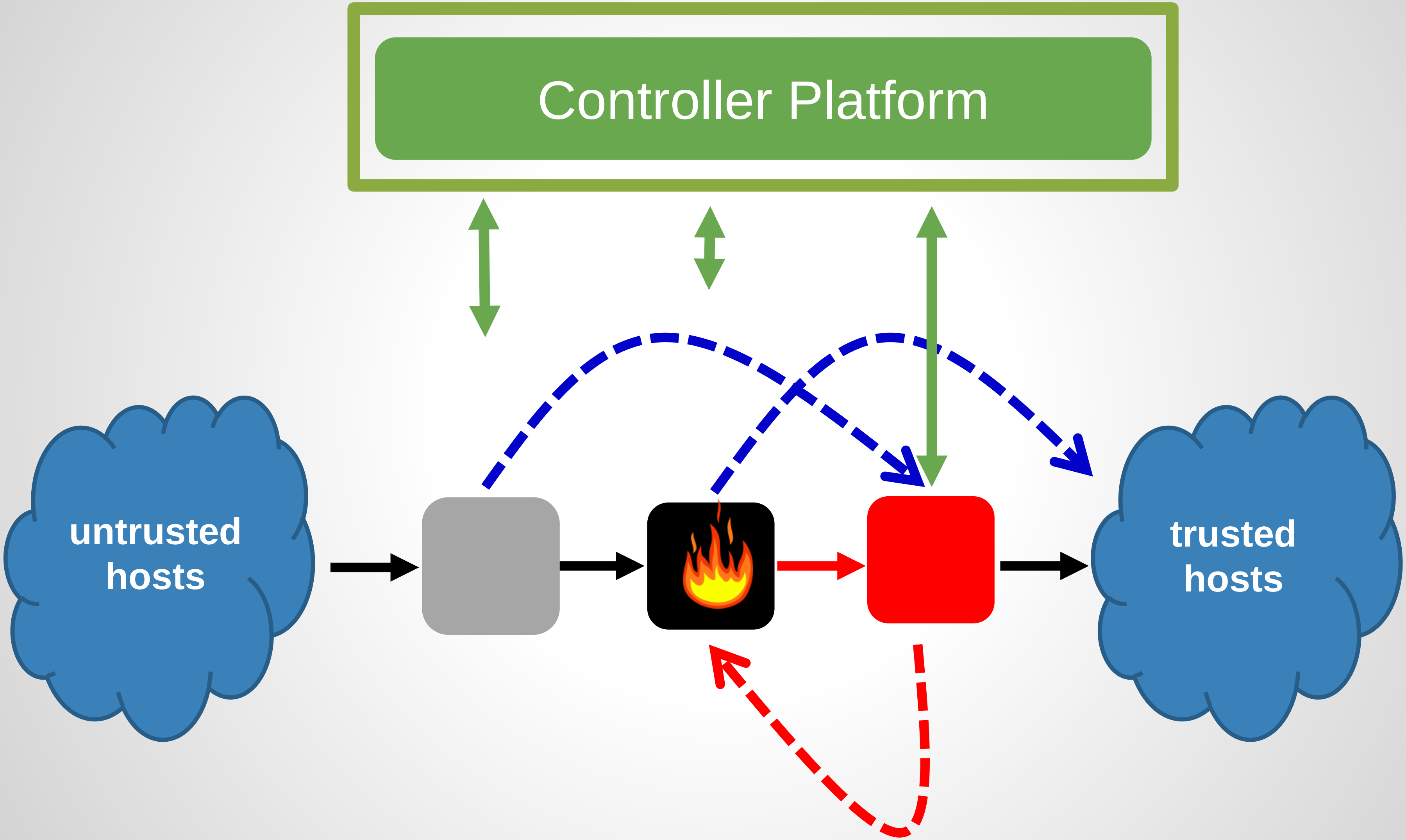
Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

Problem 1: Bypassed Waypoint



Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

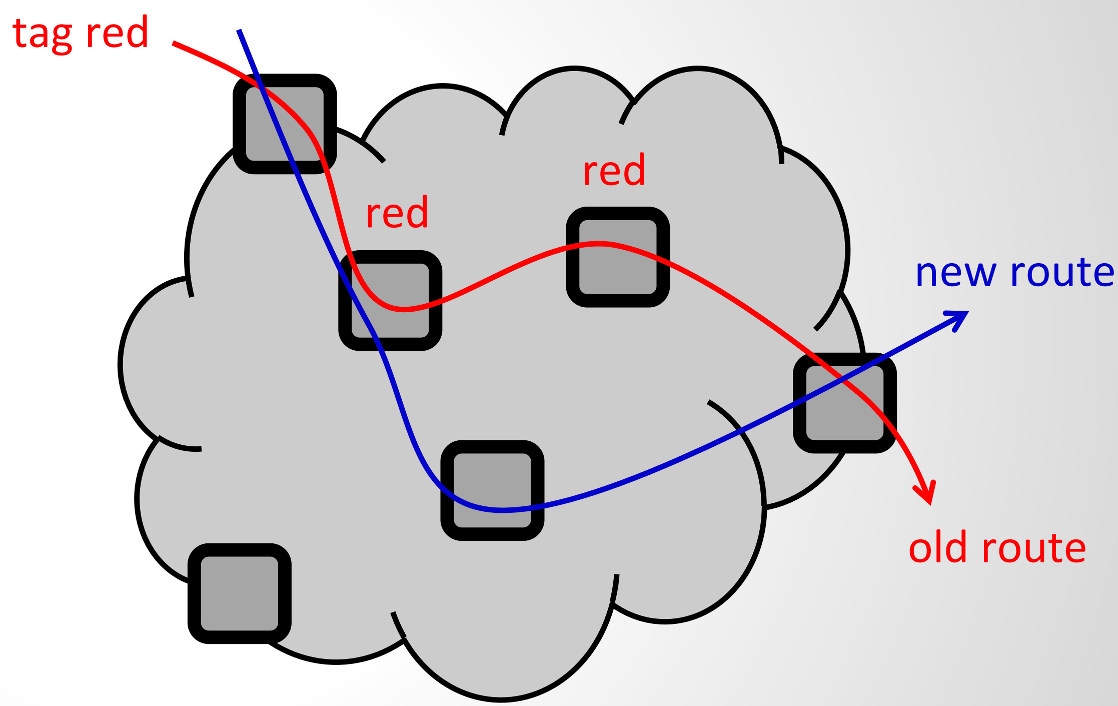
Problem 2: *Transient* Loop



Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

Tagging: A Universal Solution?

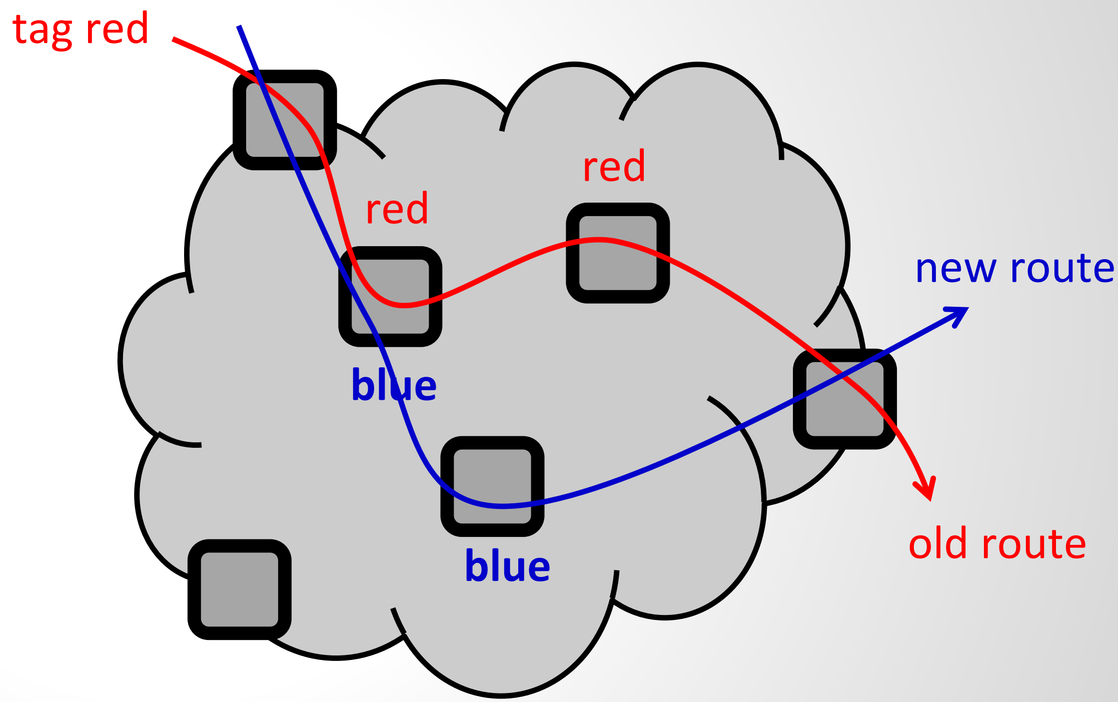
- Old route: red
- New route: blue



Reitblatt et al. Abstractions for Network Update, ACM SIGCOMM 2012.

Tagging: A Universal Solution?

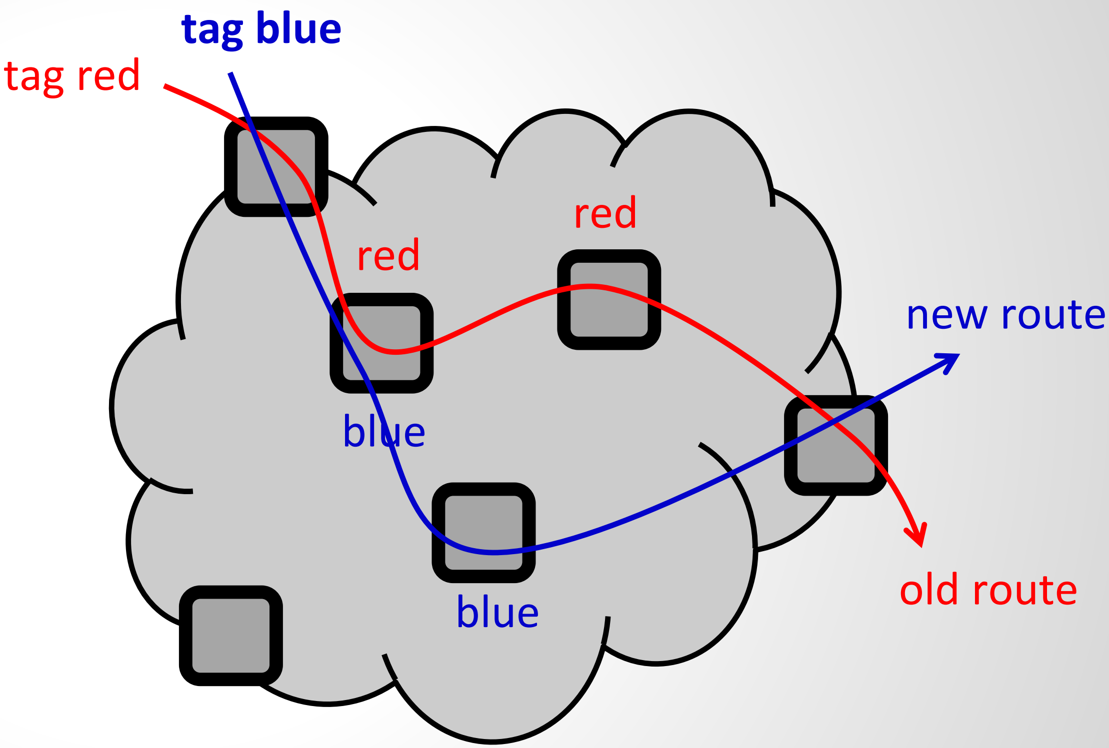
- Old route: red
- New route: blue
- 2-Phase Update:
  - Install blue flow rules internally



Reitblatt et al. Abstractions for Network Update, ACM SIGCOMM 2012.

Tagging: A Universal Solution?

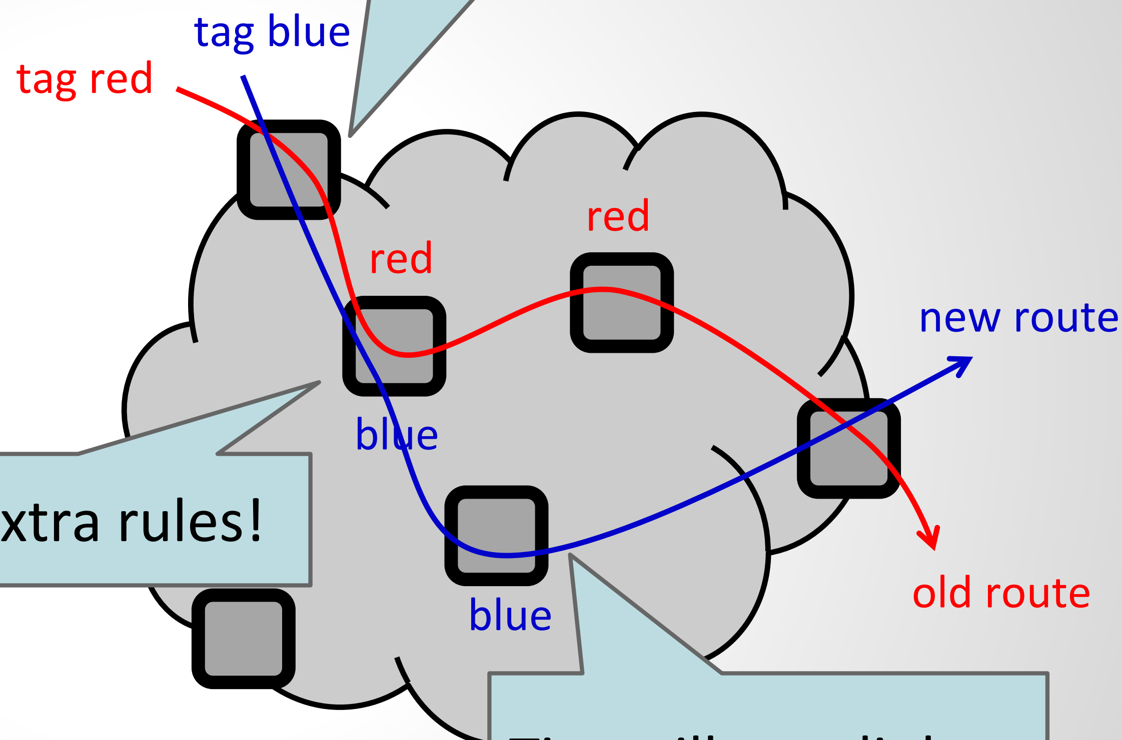
- Old route: red
- New route: blue
- 2-Phase Update:
  - Install blue flow rules internally
  - Flip tag at ingress ports



Reitblatt et al. Abstractions for Network Update, ACM SIGCOMM 2012.

Tagging: A Univers

- Old route: red
- New route: blue
- 2-Phase Update:
  - Install blue rules internally
  - Flip tag at ingress ports



Where to tag?
Header space?
Overhead!

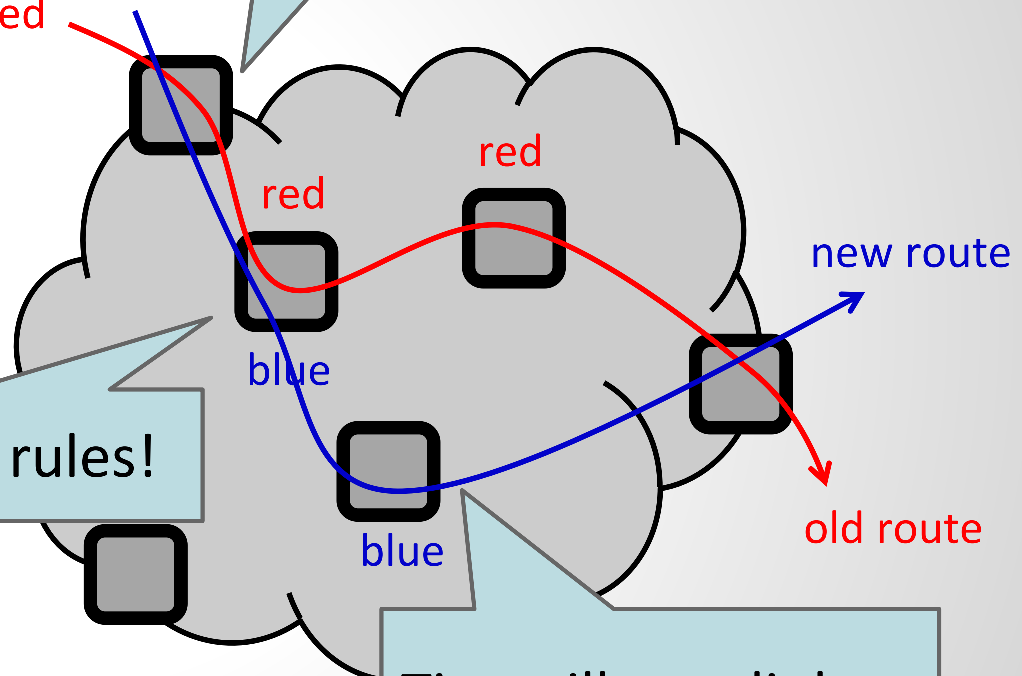
Cost of extra rules!

Time till new link becomes available!

Reitblatt et al. Abstractions for Network Update, ACM SIGCOMM 2012.

Tagging: A Univers

- ❑ Old route: red
- ❑ New route: blue
- ❑ 2-Phase Update:
  - ❑ Install blue rules internally
  - ❑ Flip tag at ingress port



Where to tag?
Header space?
Overhead!

Cost of extra rules!

Time till new link becomes available!

Possible solution without tagging, and at least preserve weaker consistency properties?

Reitblatt et al. Abstractions for Network Update, ACM SIGCOMM 2012.

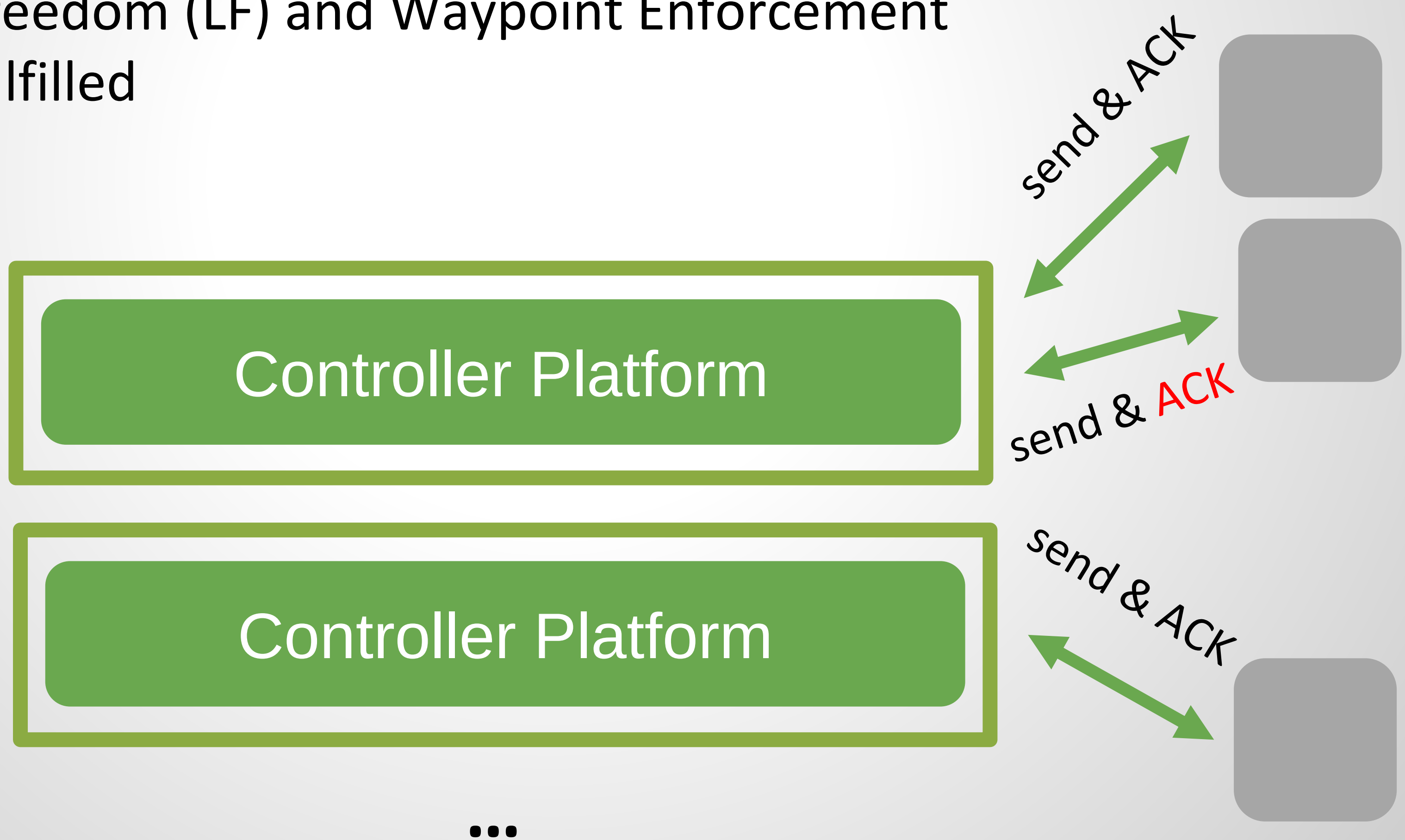
Idea: Schedule "Safe" Subsets of Nodes Only, Then Wait for ACK!

Idea: Schedule safe update subsets in multiple rounds!

Packet may take a mix of old and new path, as long as, e.g., Loop-Freedom (LF) and Waypoint Enforcement (WPE) are fulfilled

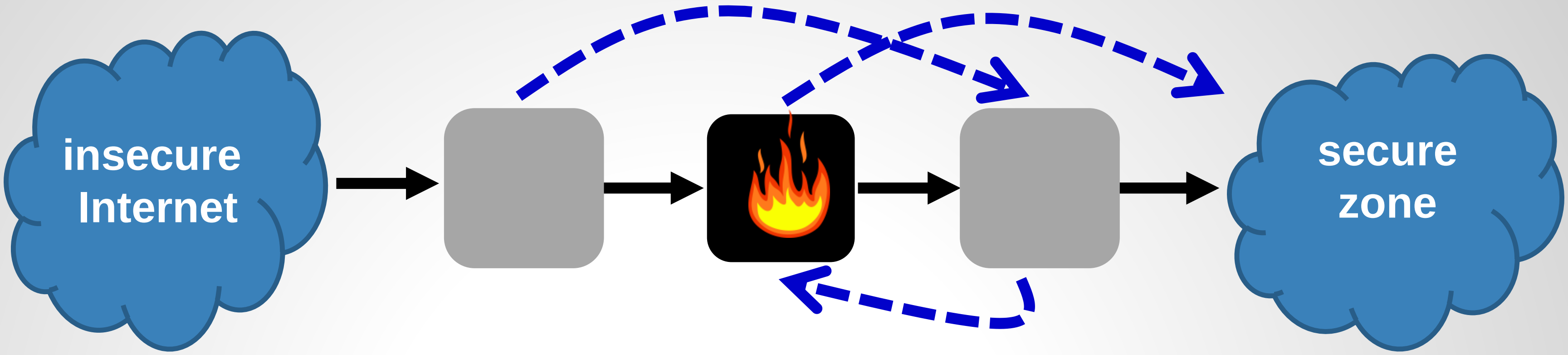
Round 1

Round 2

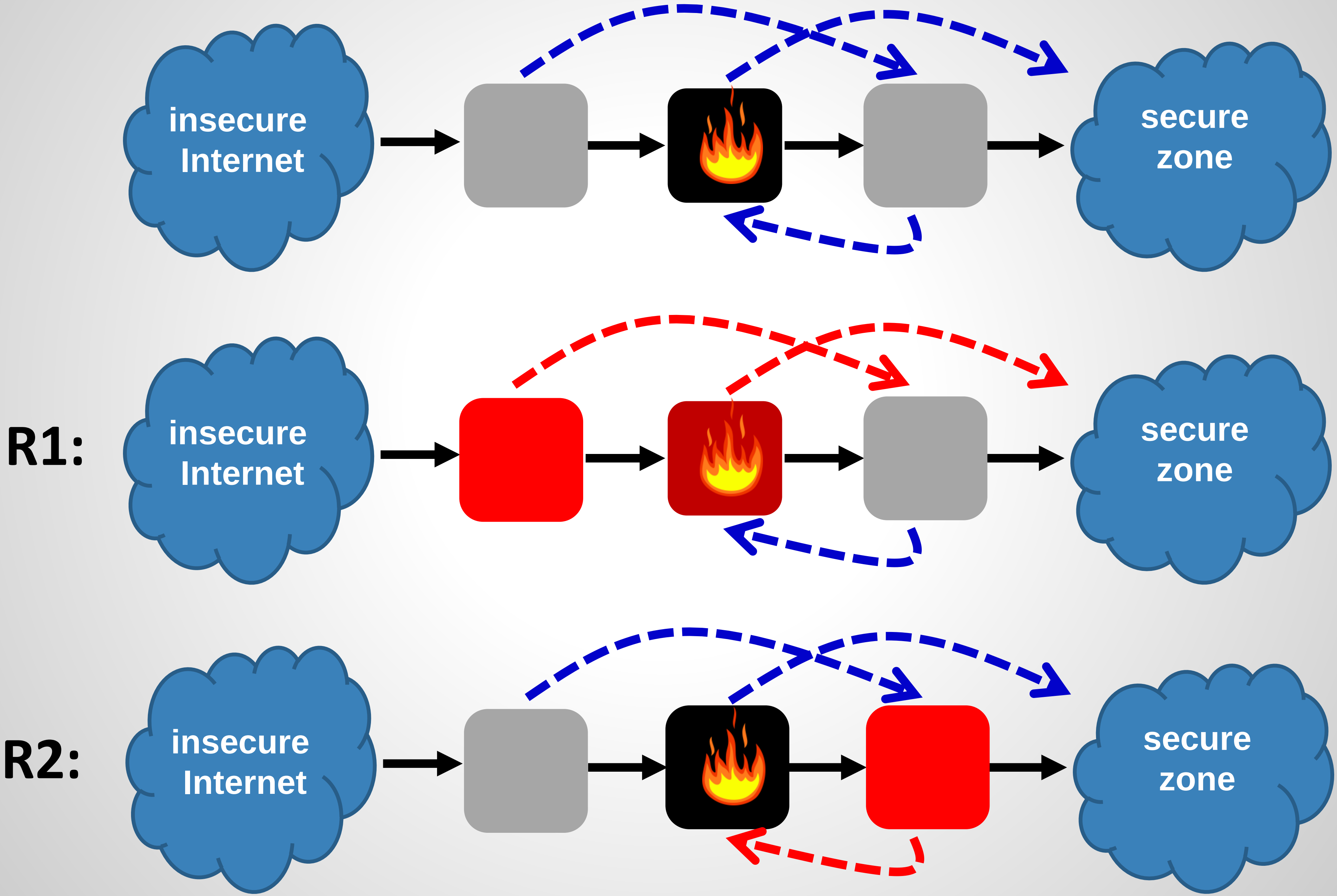


...

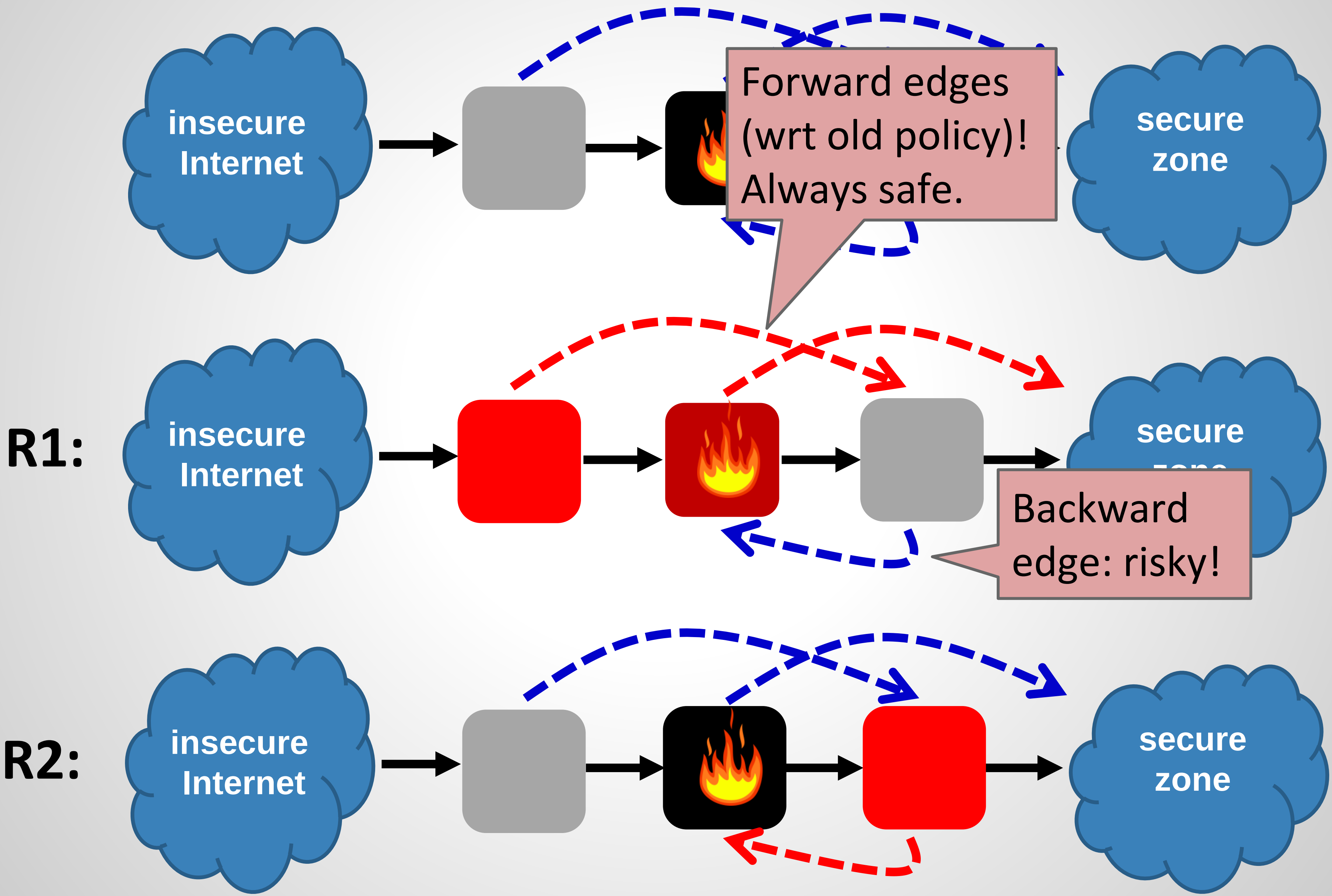
Loop-Free Update Schedule



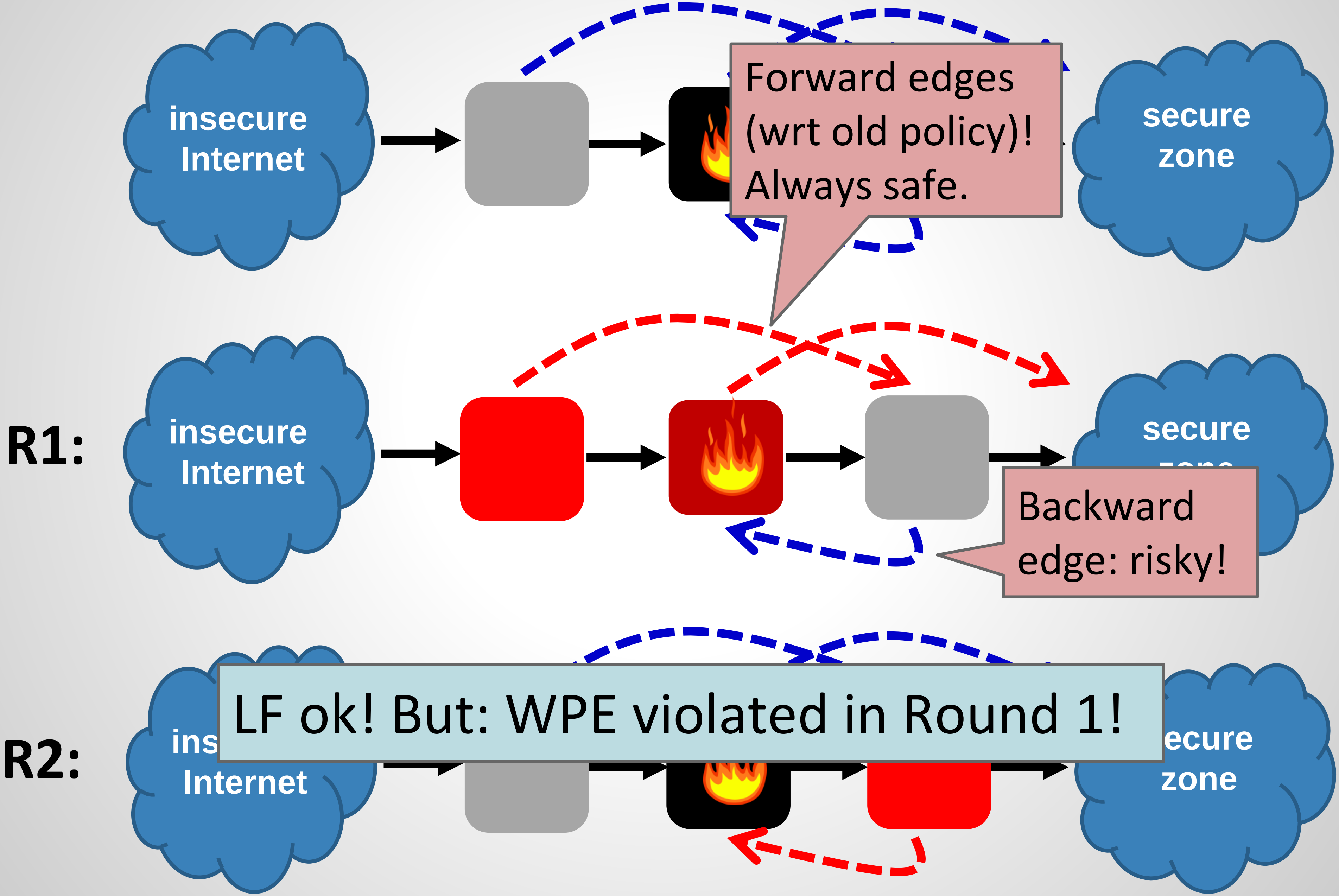
Loop-Free Update Schedule



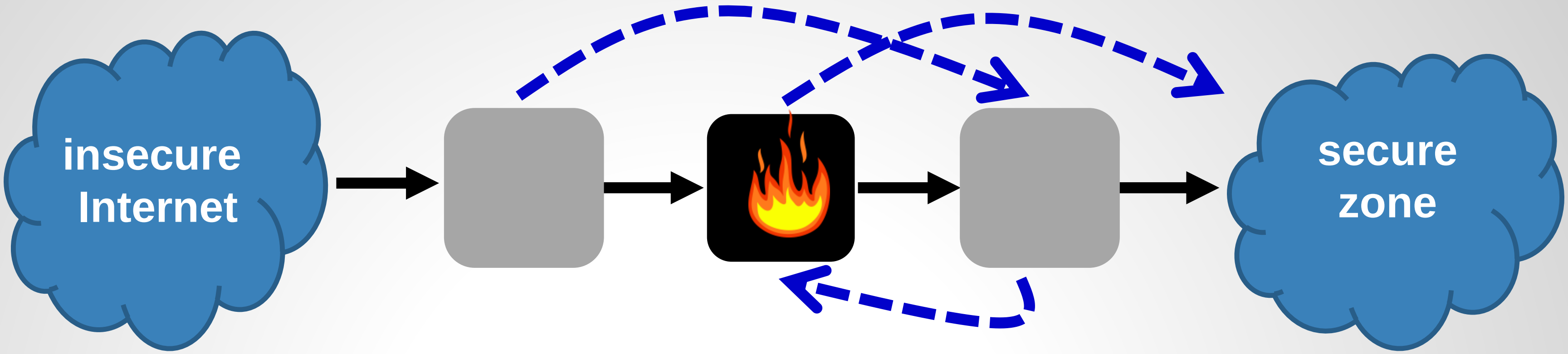
Loop-Free Update Schedule



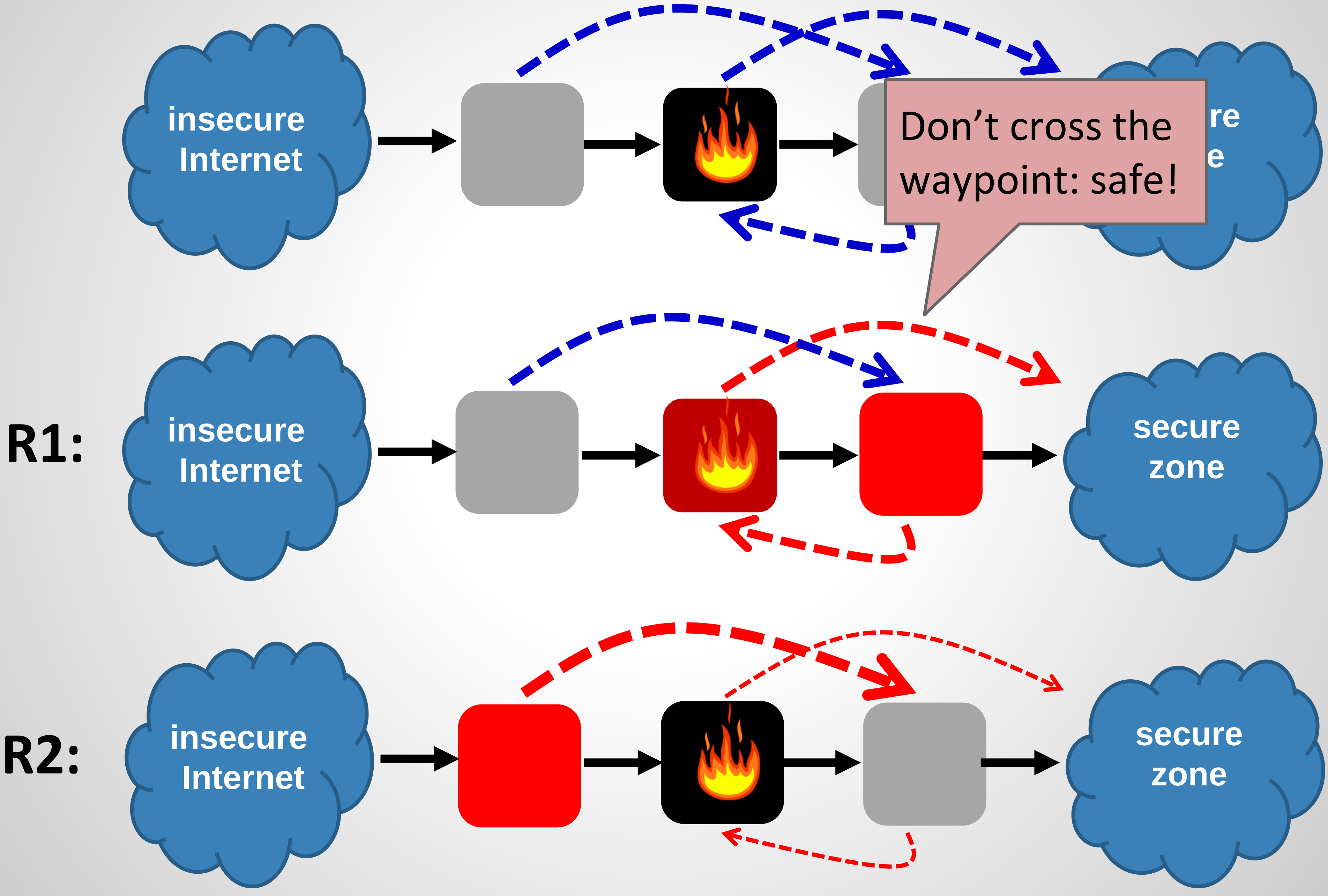
Loop-Free Update Schedule



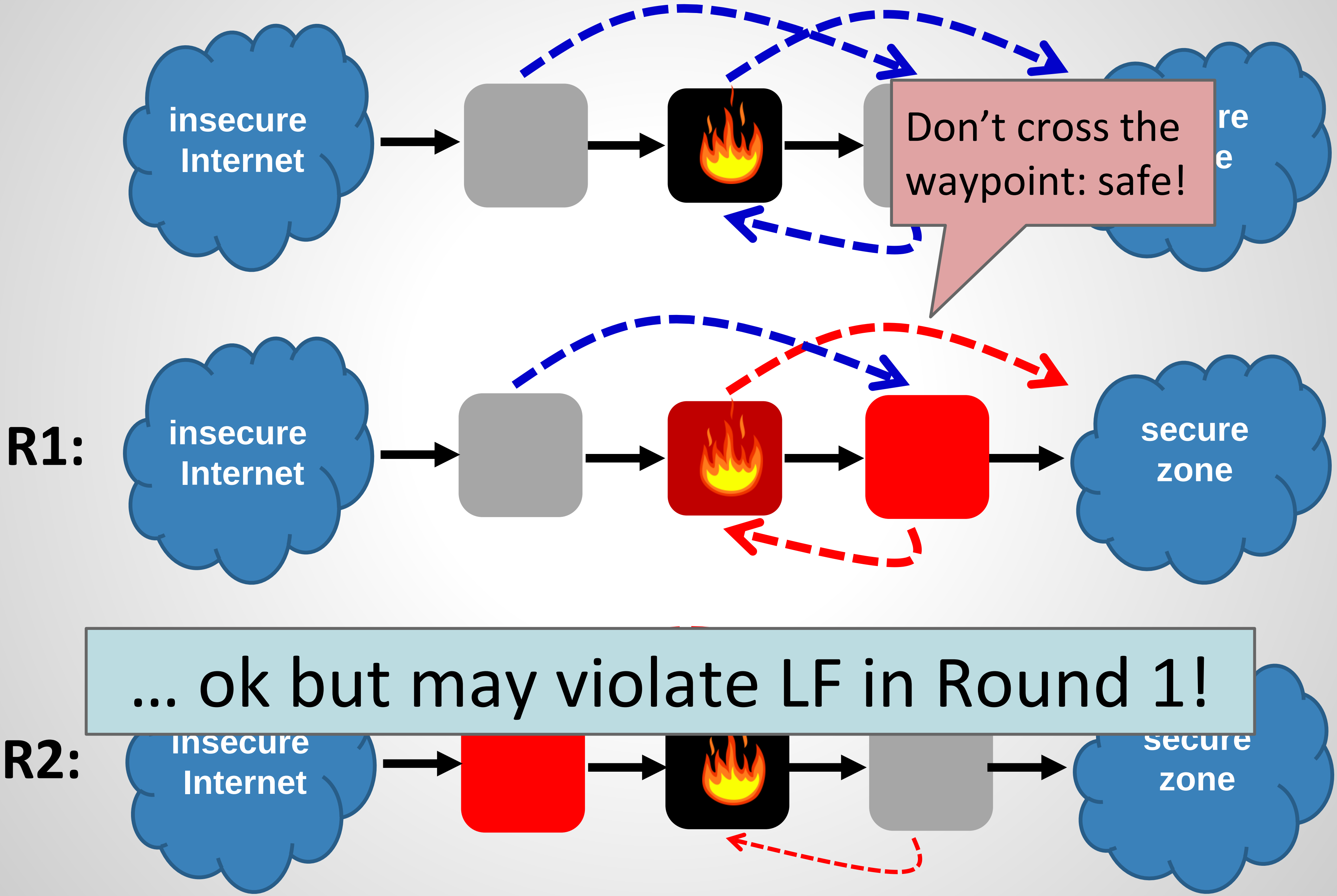
Waypoint Respecting Schedule



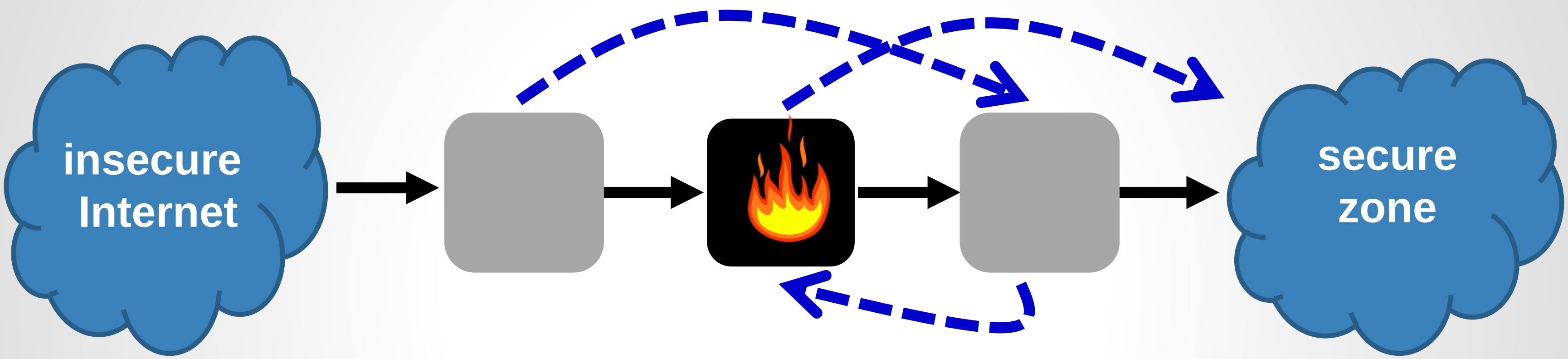
Waypoint Respecting Schedule



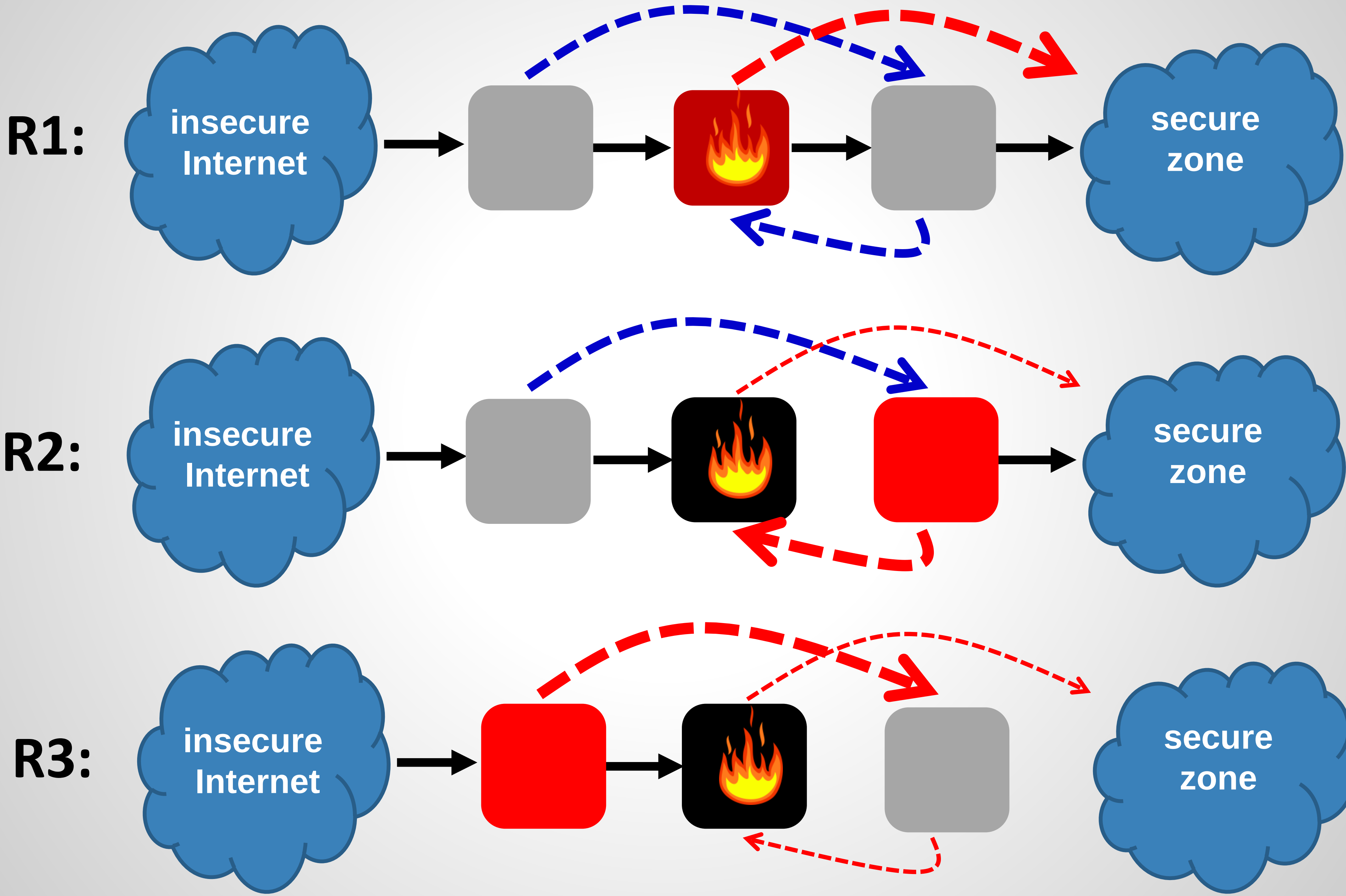
Waypoint Respecting Schedule



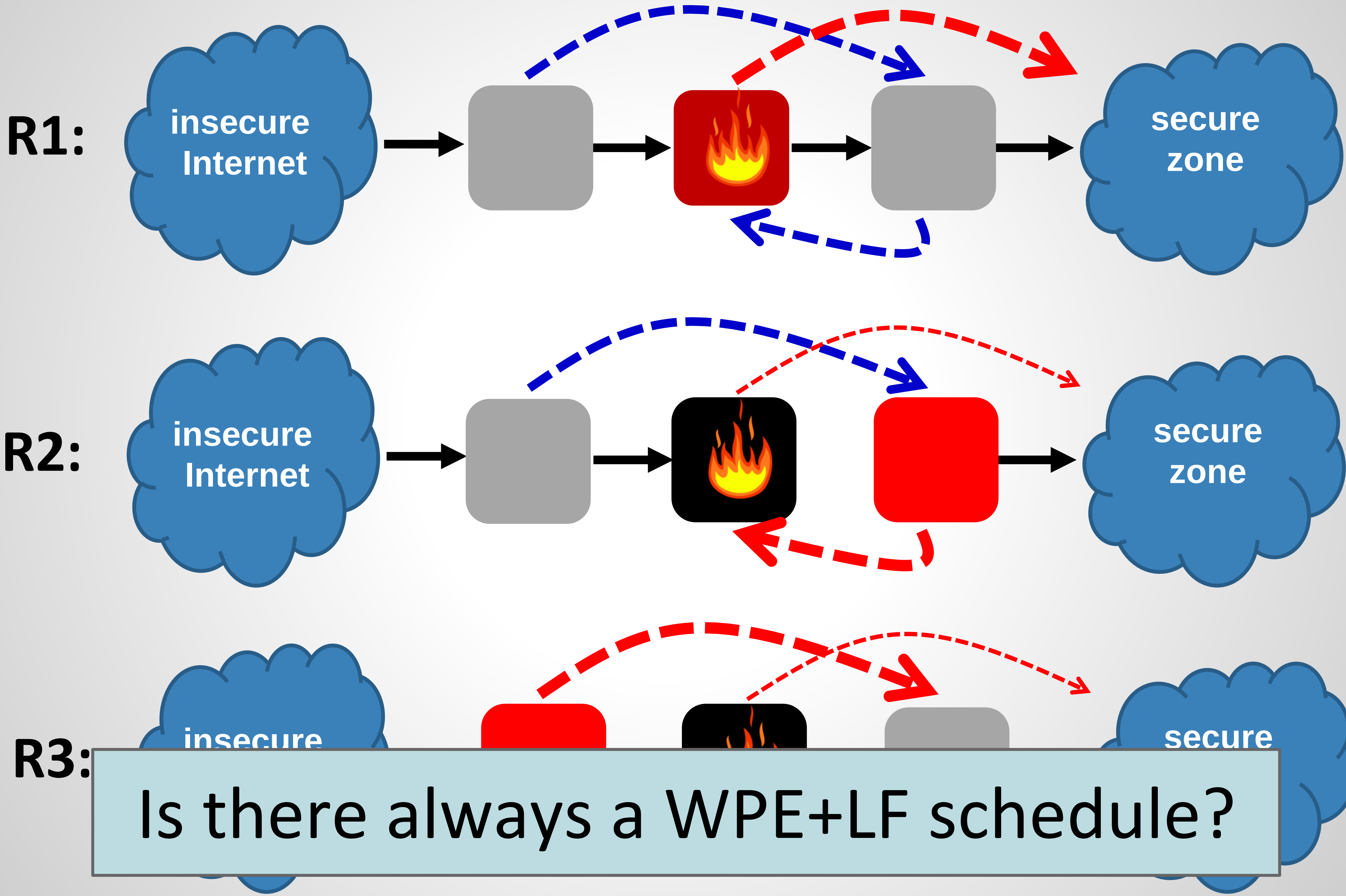
Can we have both LF and WPE?



Yes: but it takes 3 rounds!

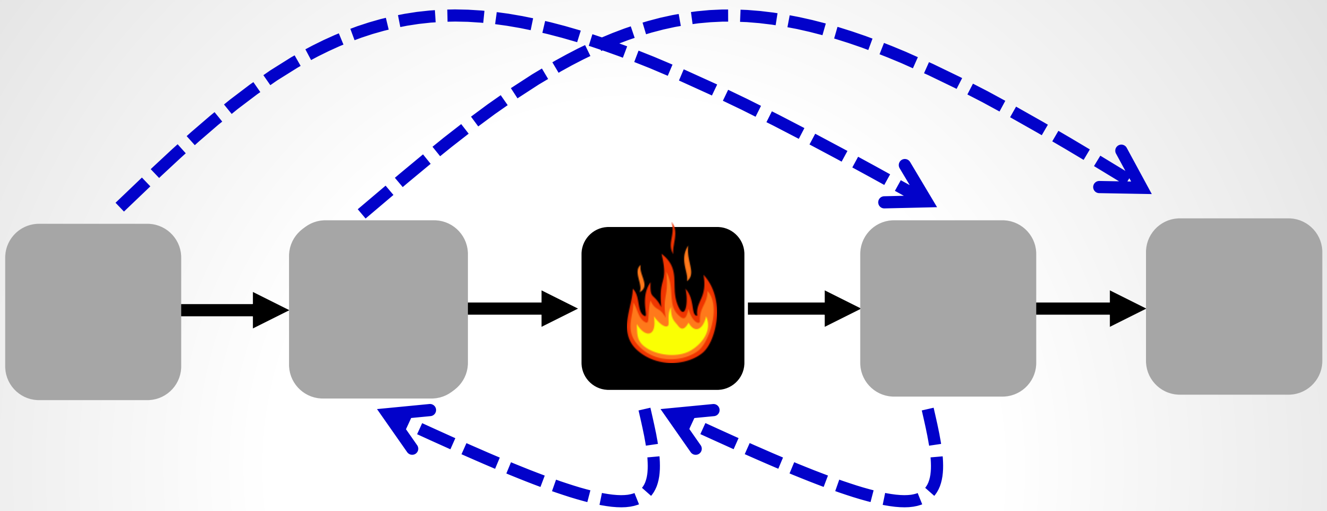


Yes: but it takes 3 rounds!

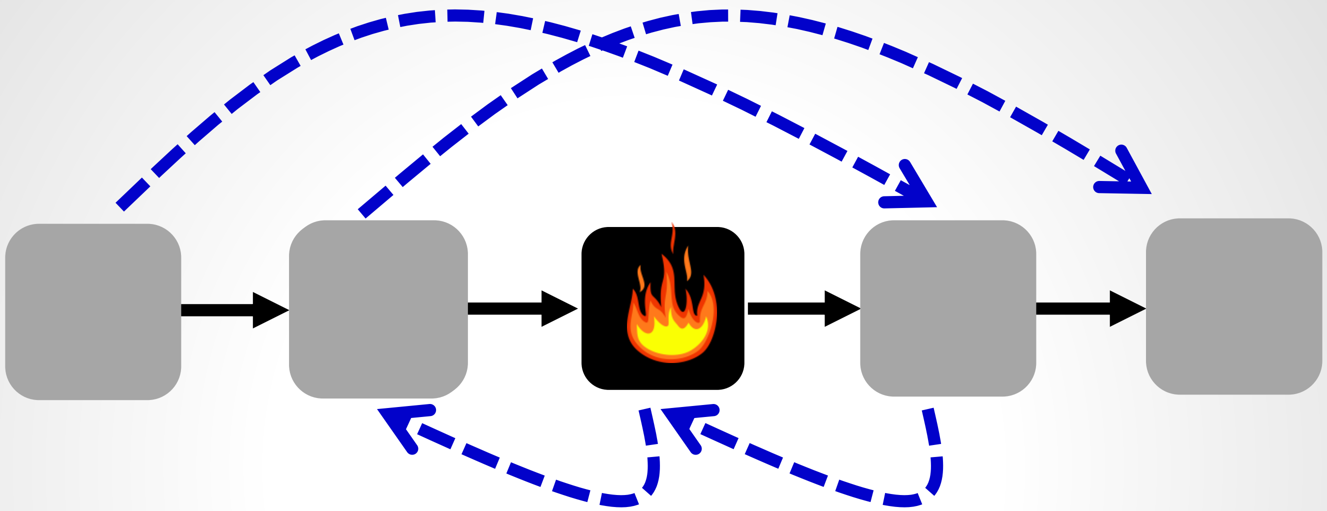


Is there always a WPE+LF schedule?

What about this one?



LF and WPE may conflict!

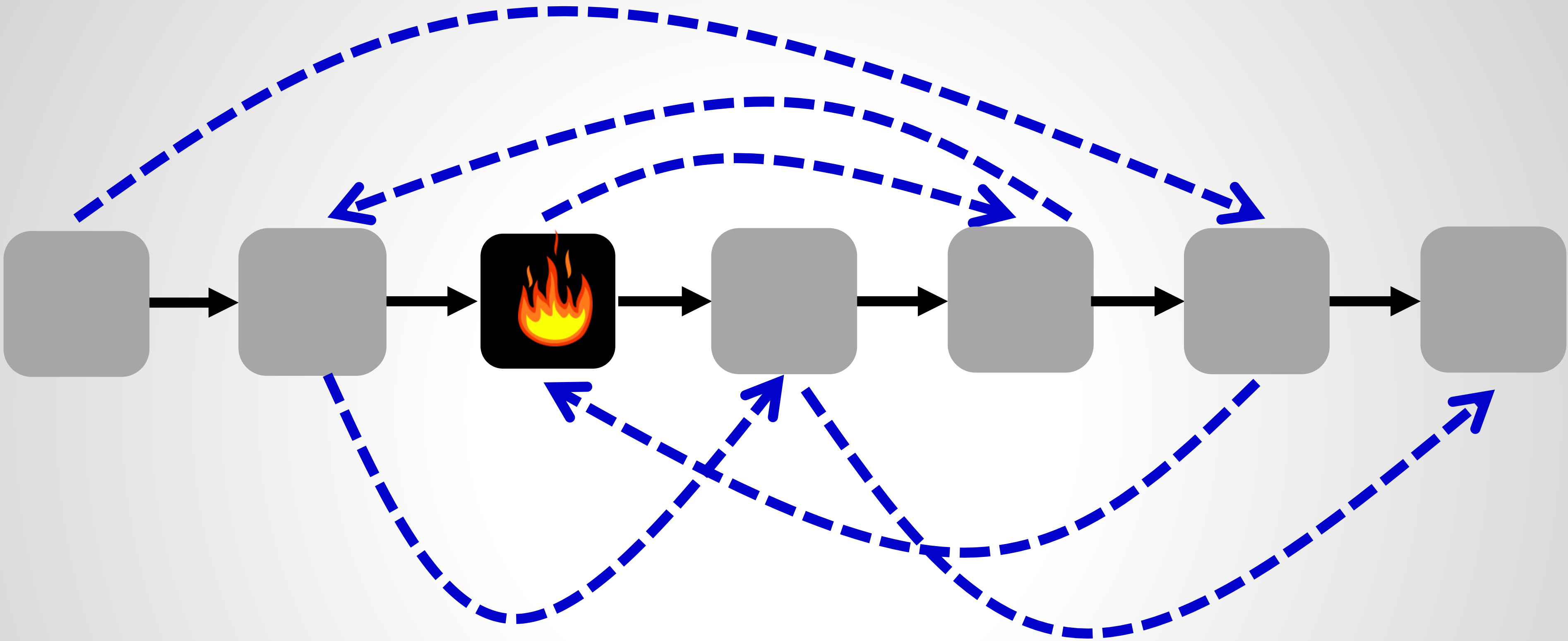


- ❑ Cannot update any **forward edge** in R1: WP
- ❑ Cannot update any **backward edge** in R1: LF

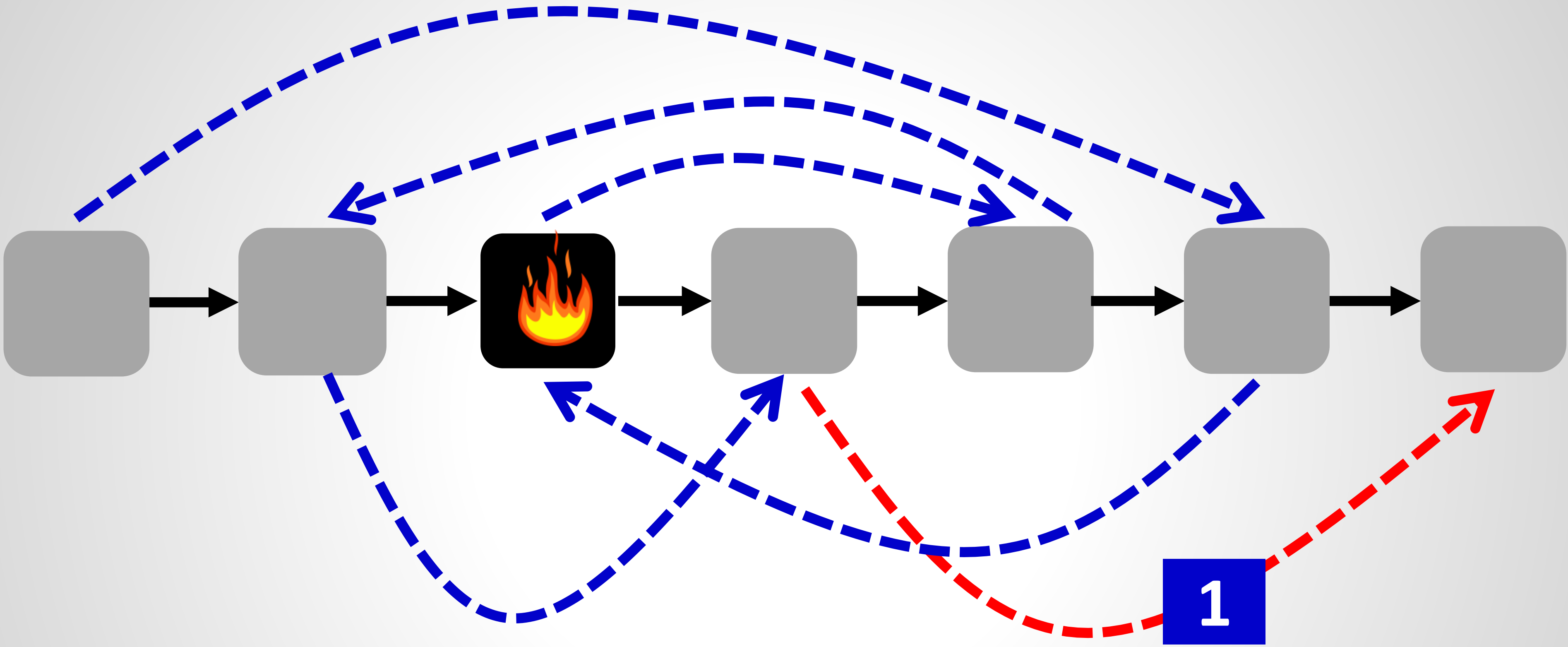
No schedule exists!

Resort to tagging…

What about this one?

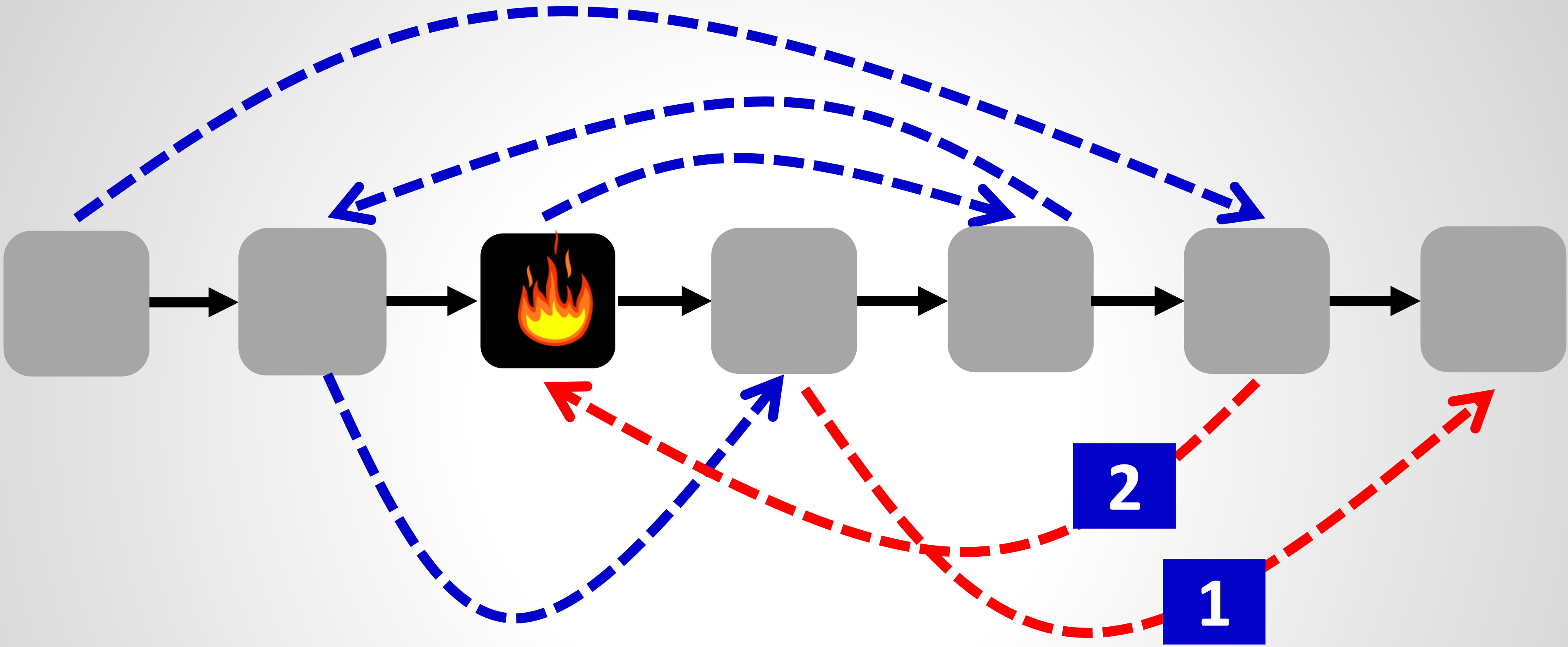


What about this one?



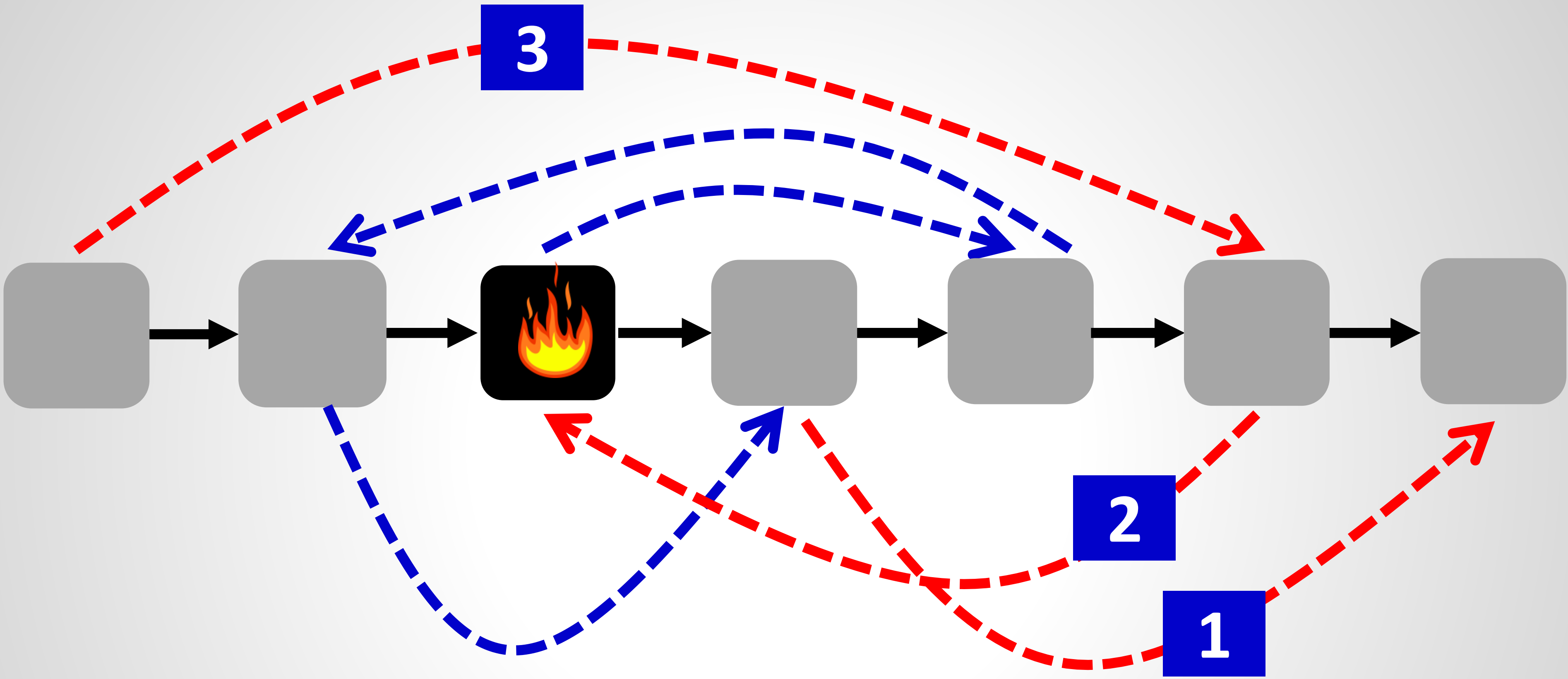
- Forward edge after the waypoint: safe!
  - No loop, no WPE violation

What about this one?



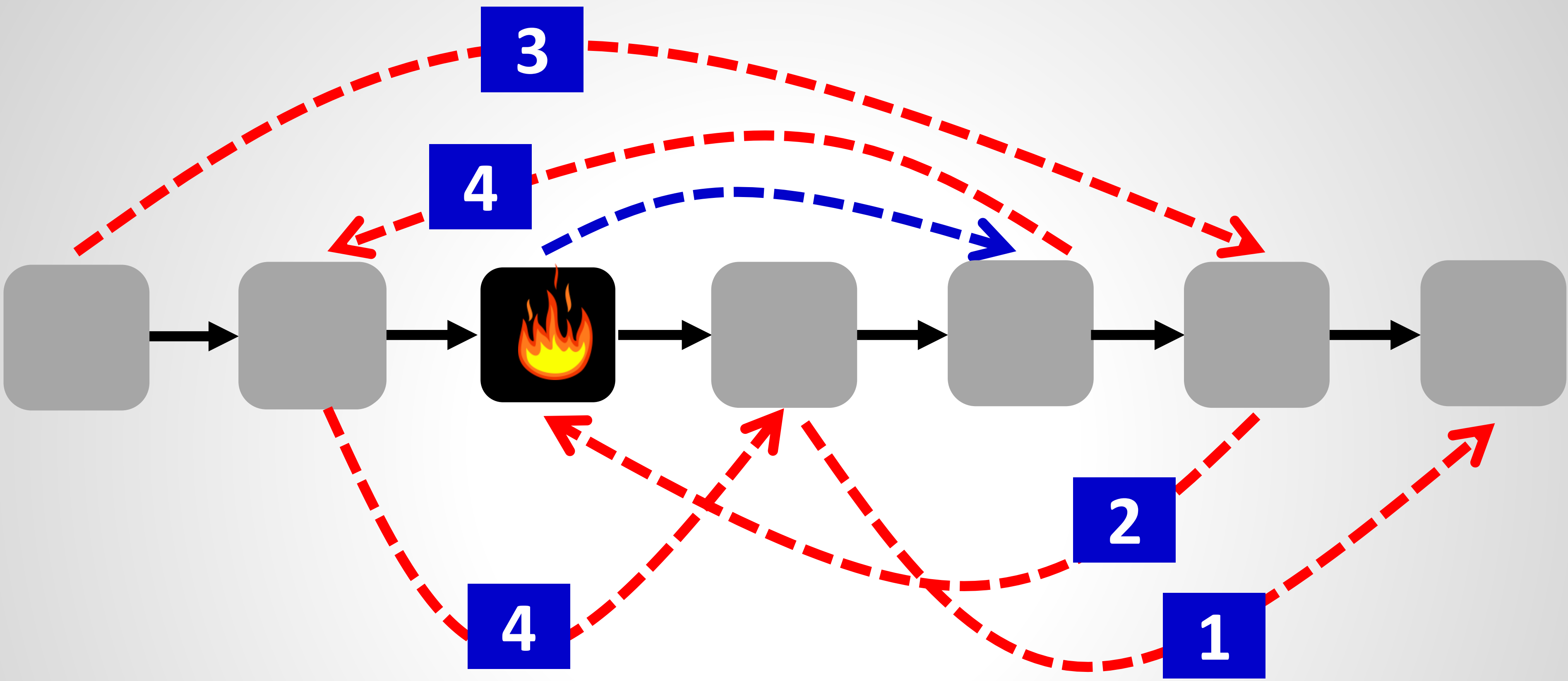
- Now this backward is safe too!
  - No loop because exit through 1

What about this one?



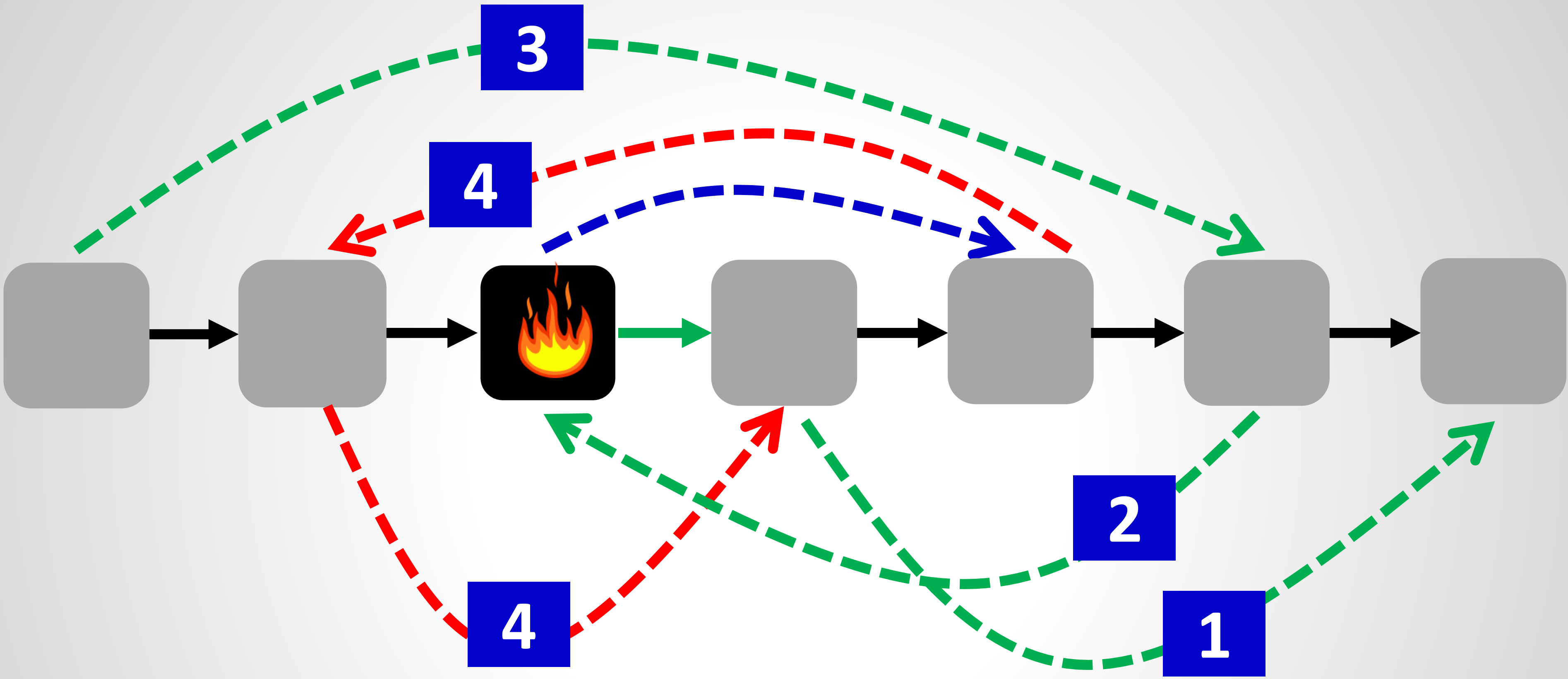
- Now this is safe: $\boxed{2}$ ready back to WP!
  - No waypoint violation

What about this one?



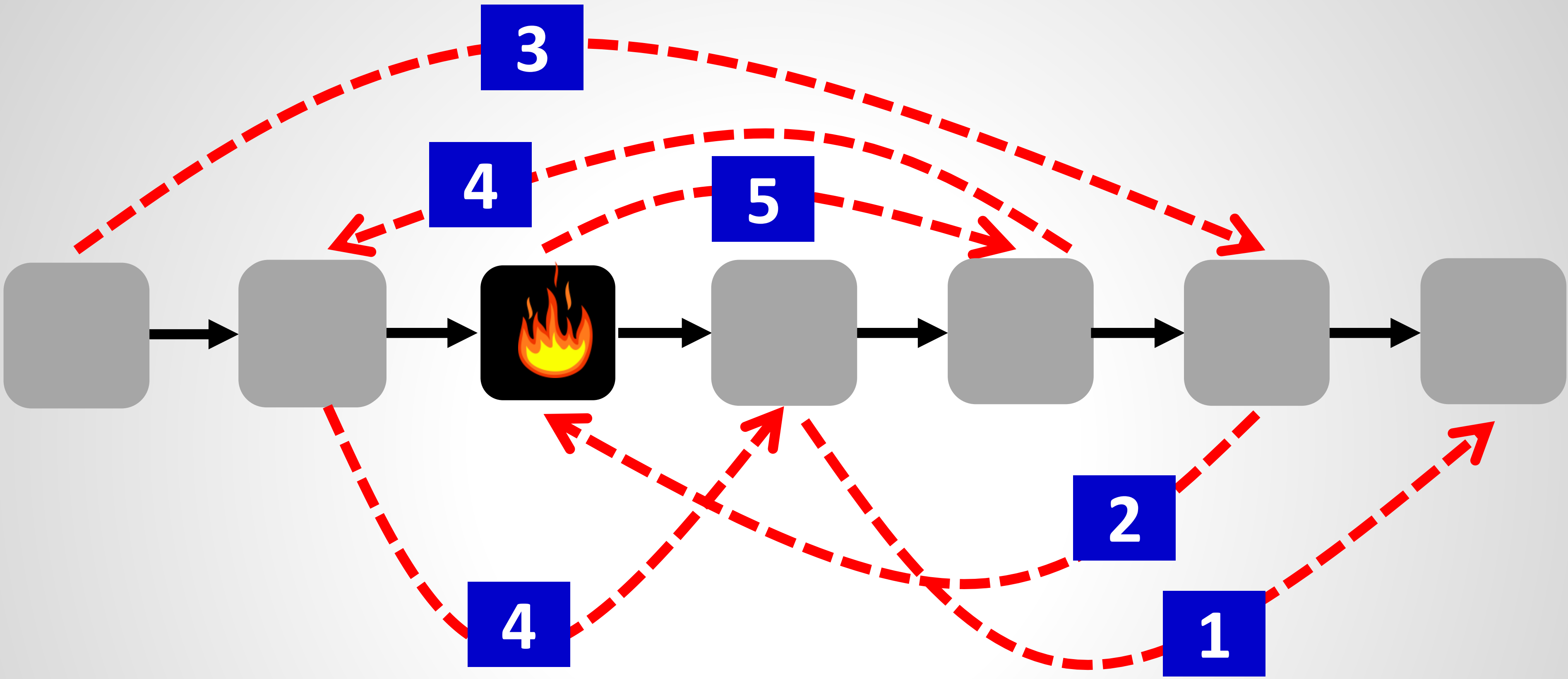
❑ Ok: loop-free and also not on the path (exit via 1)

What about this one?

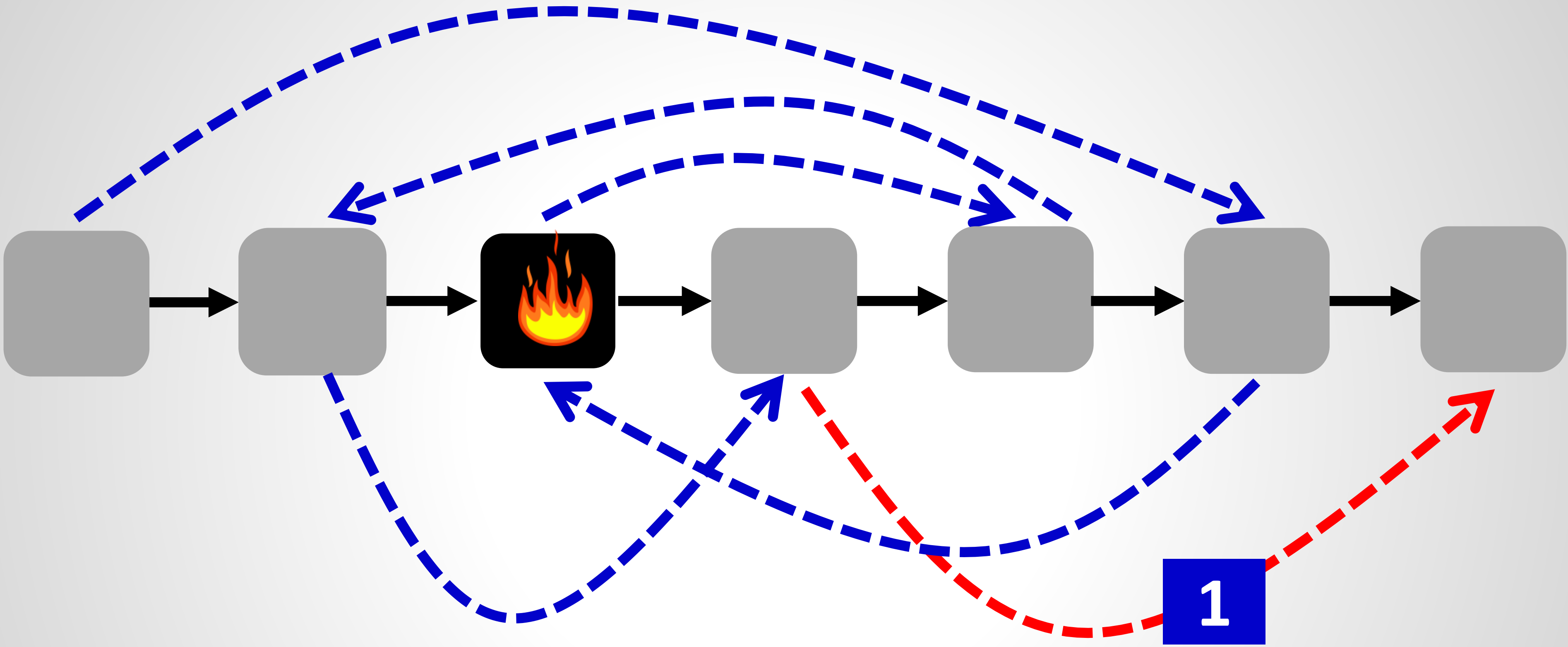


❑ Ok: loop-free and also not on the path (exit via 1)

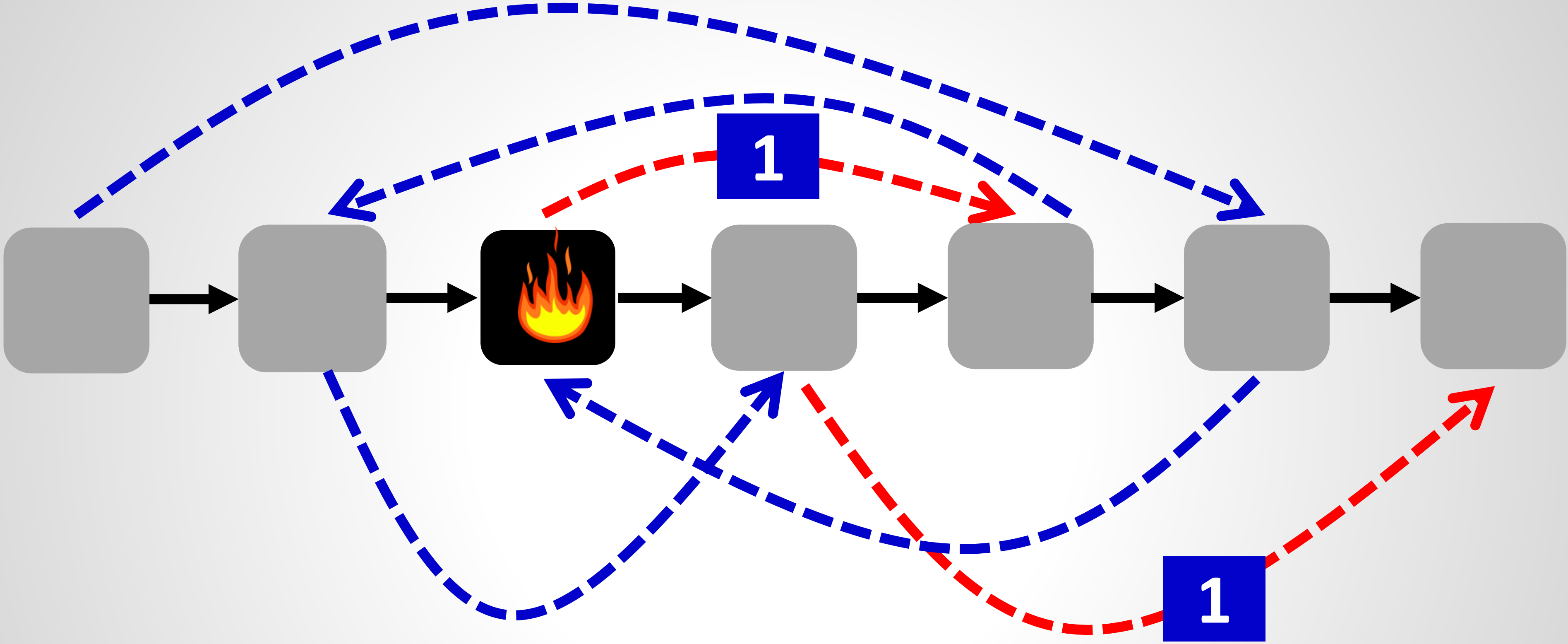
What about this one?



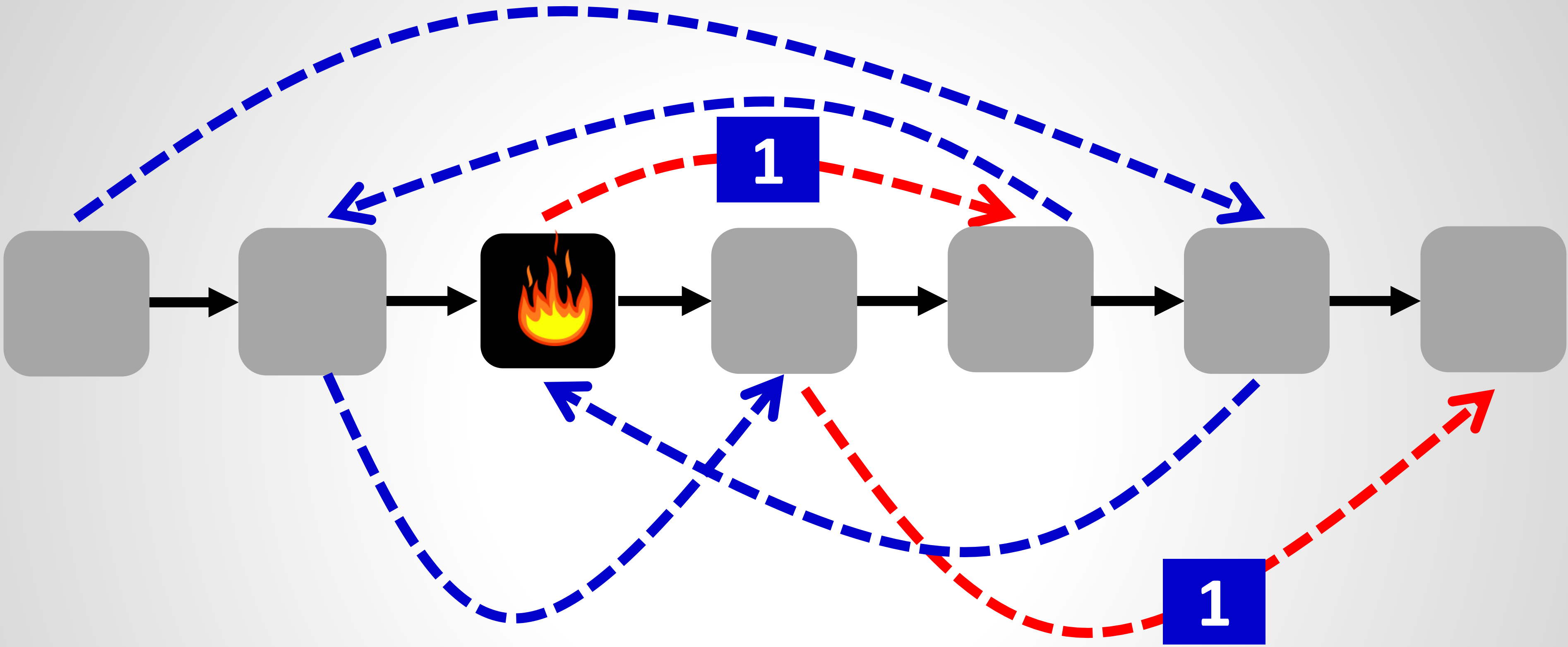
Back to the start: What if....



Back to the start: What if…. also this one?!

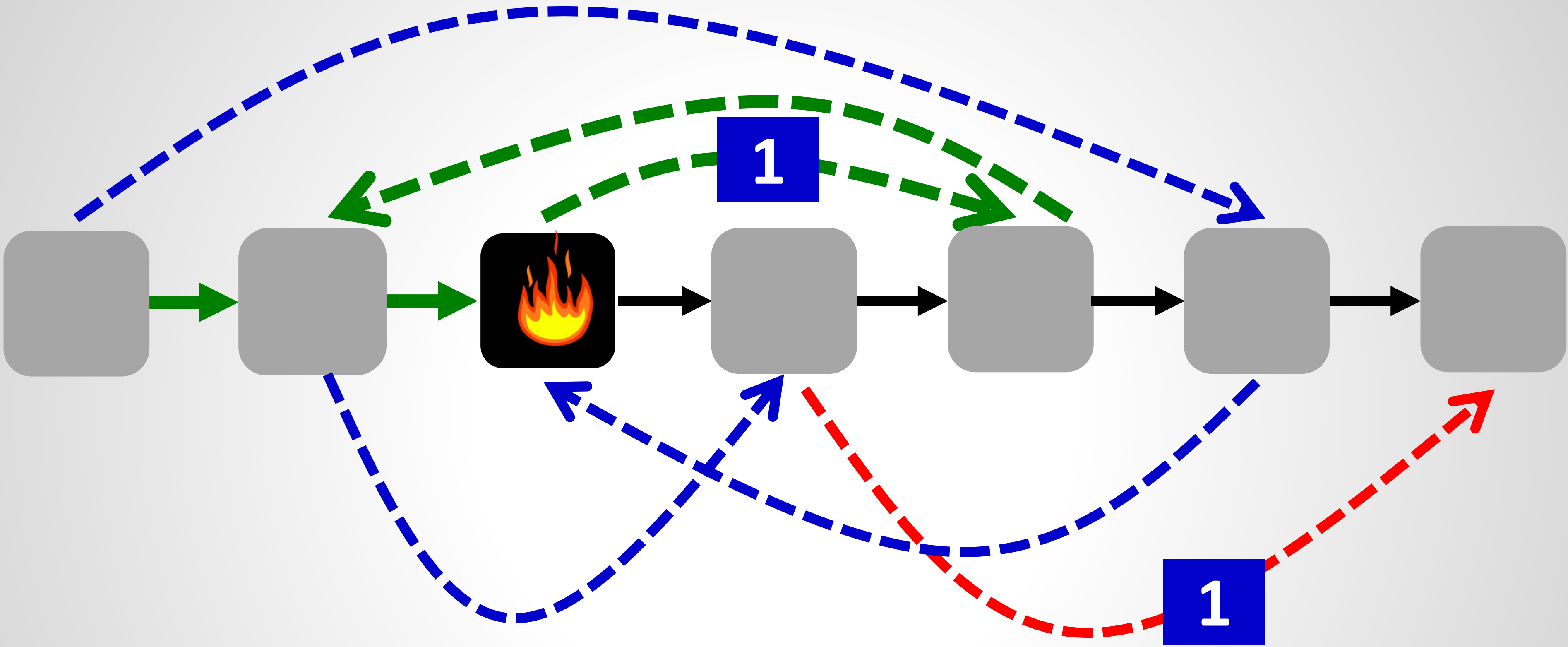


Back to the start: What if…. also this one?!



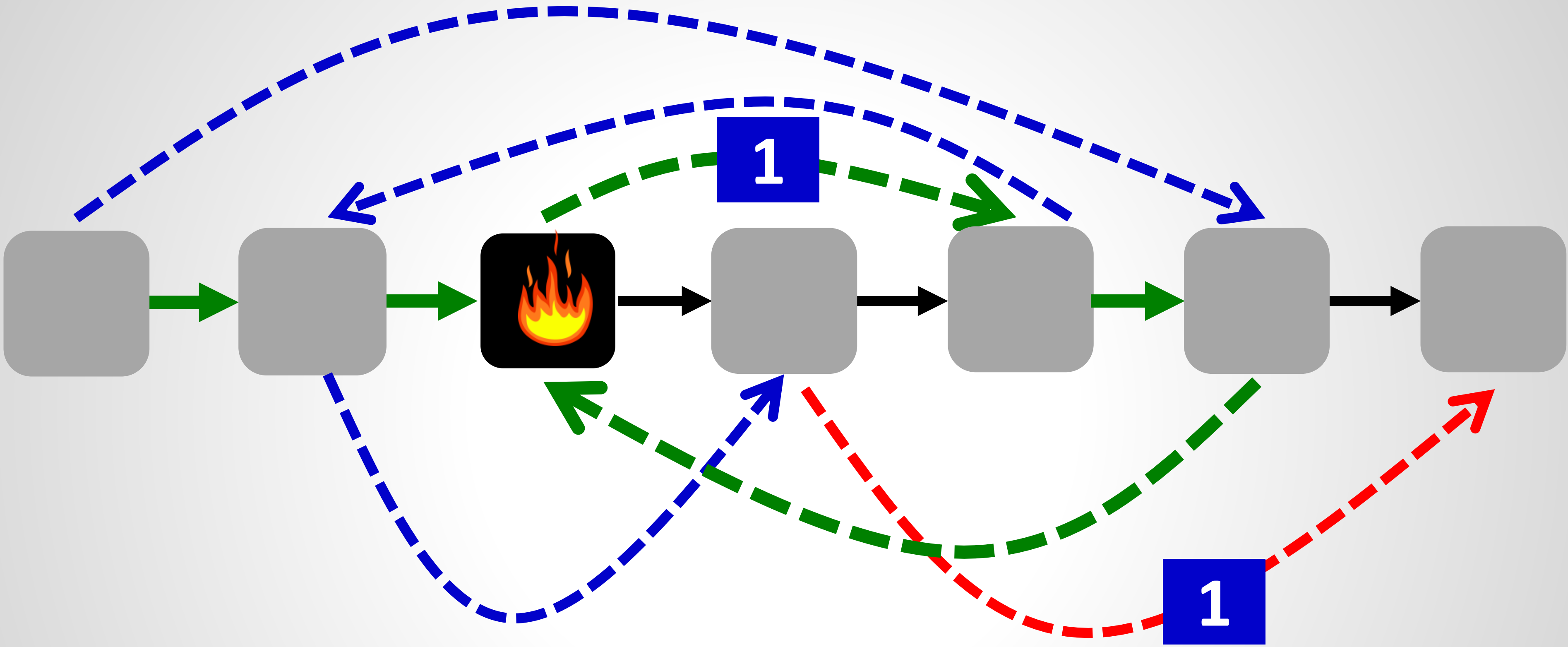
- ❑ Update any of the 2 backward edges? LF ☹

Back to the start: What if…. also this one?!



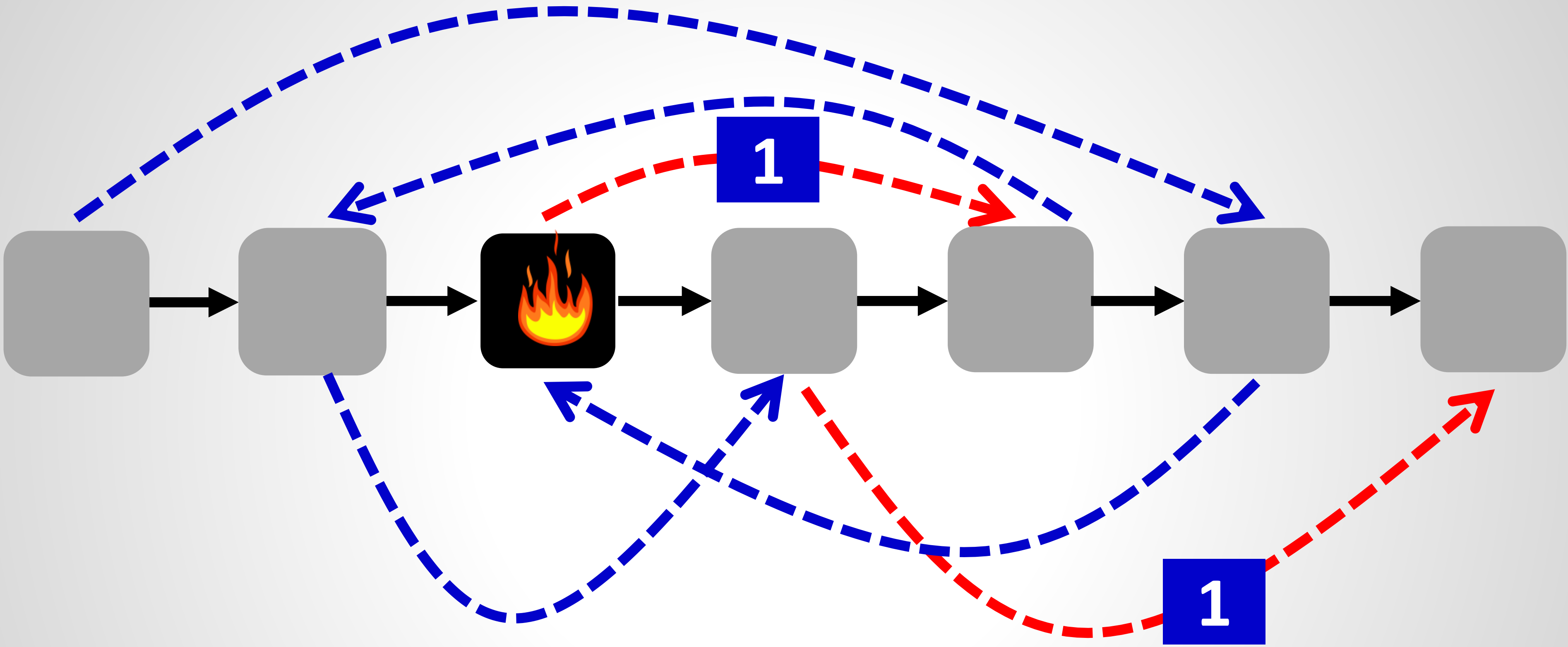
- ❑ Update any of the 2 backward edges? LF ☹

Back to the start: What if…. also this one?!



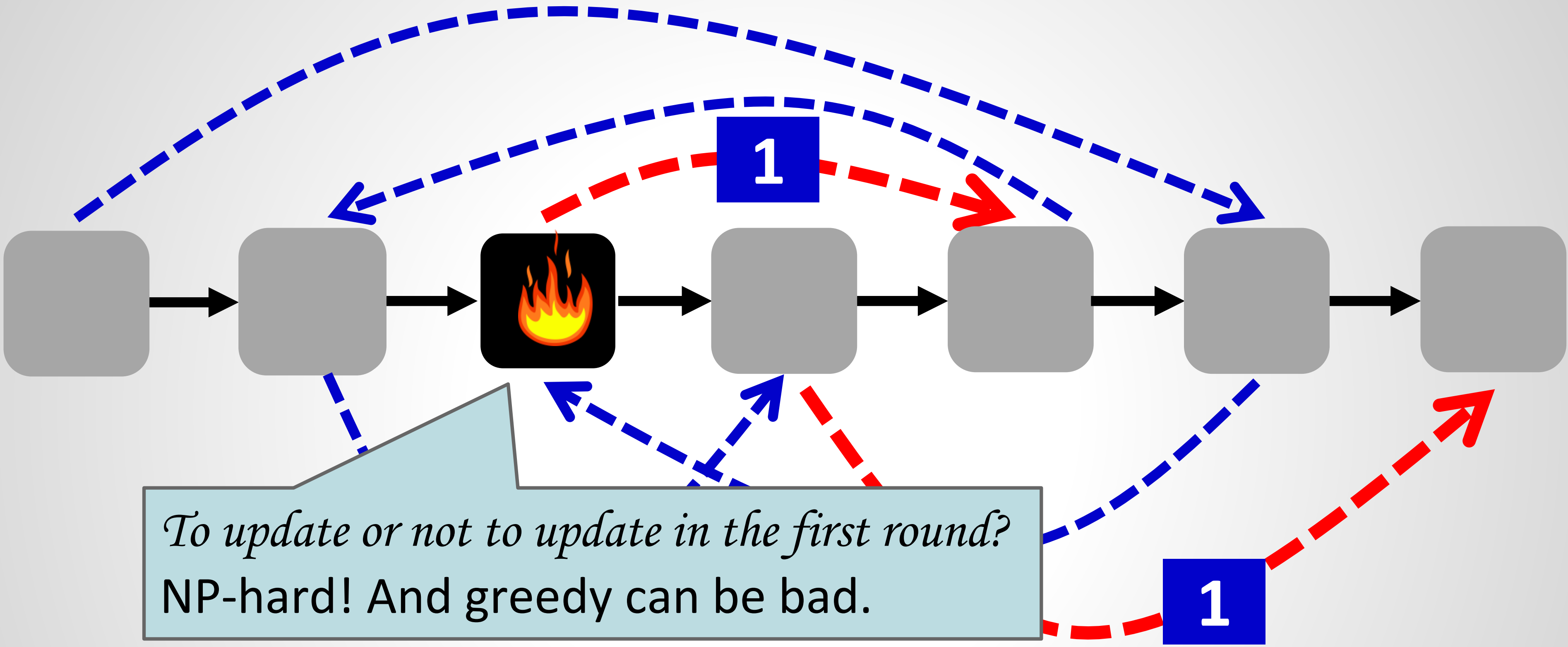
- ❑ Update any of the 2 backward edges? LF ☹

Back to the start: What if…. also this one?!



- ❑ Update any of the 2 backward edges? LF ☹
- ❑ Update any of the 2 other forward edges? WPE ☹
- ❑ What about a combination? No…

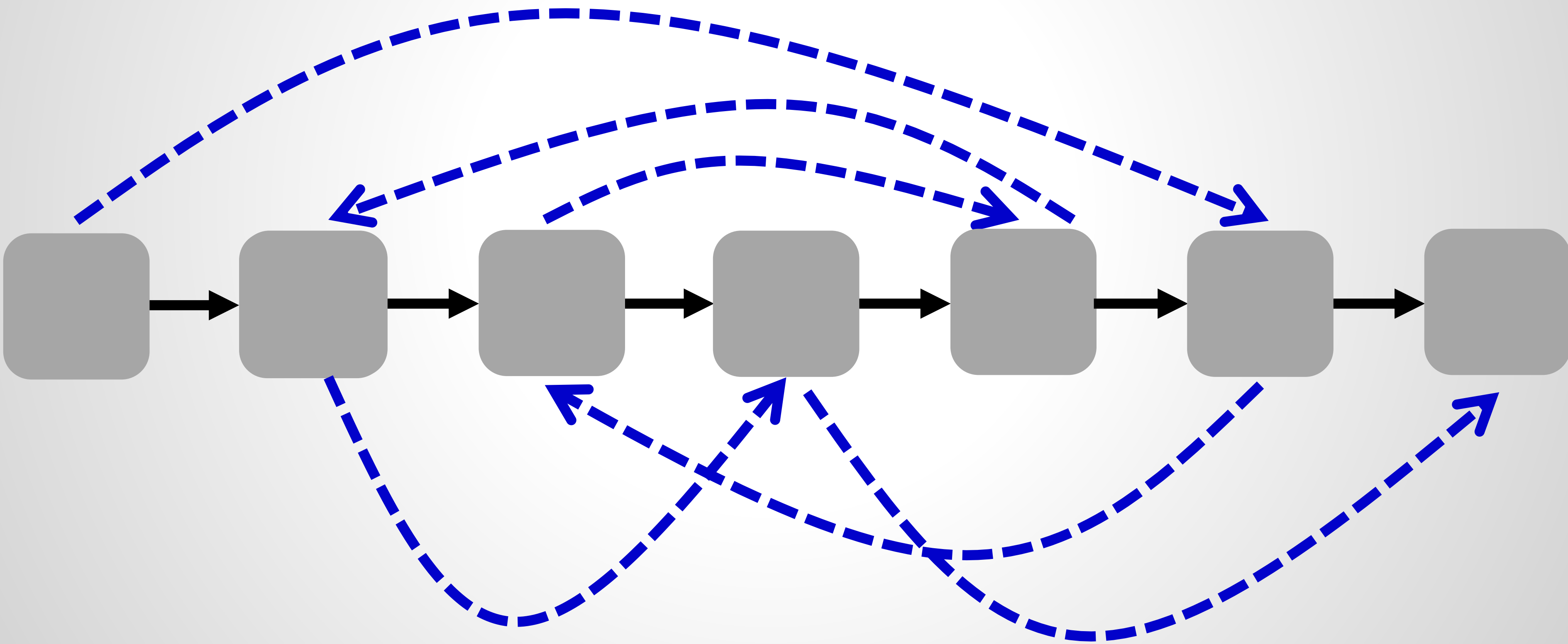
In General: NP-Hard!



Bad news: Even decidability hard: cannot quickly test feasibility and if infeasible resort to say, tagging solution!

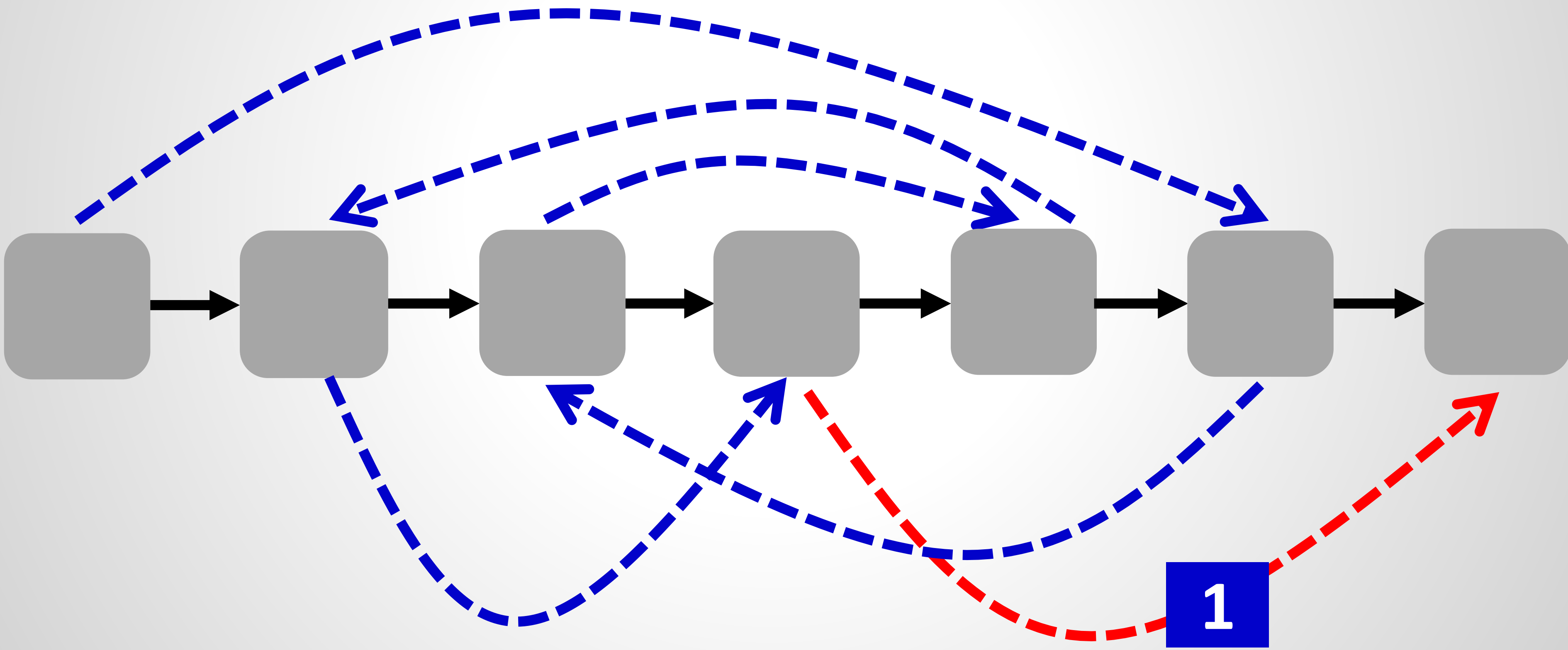
Open question: What is complexity in „typical networks“, like datacenter or enterprise networks?

What about loop-freedom only?



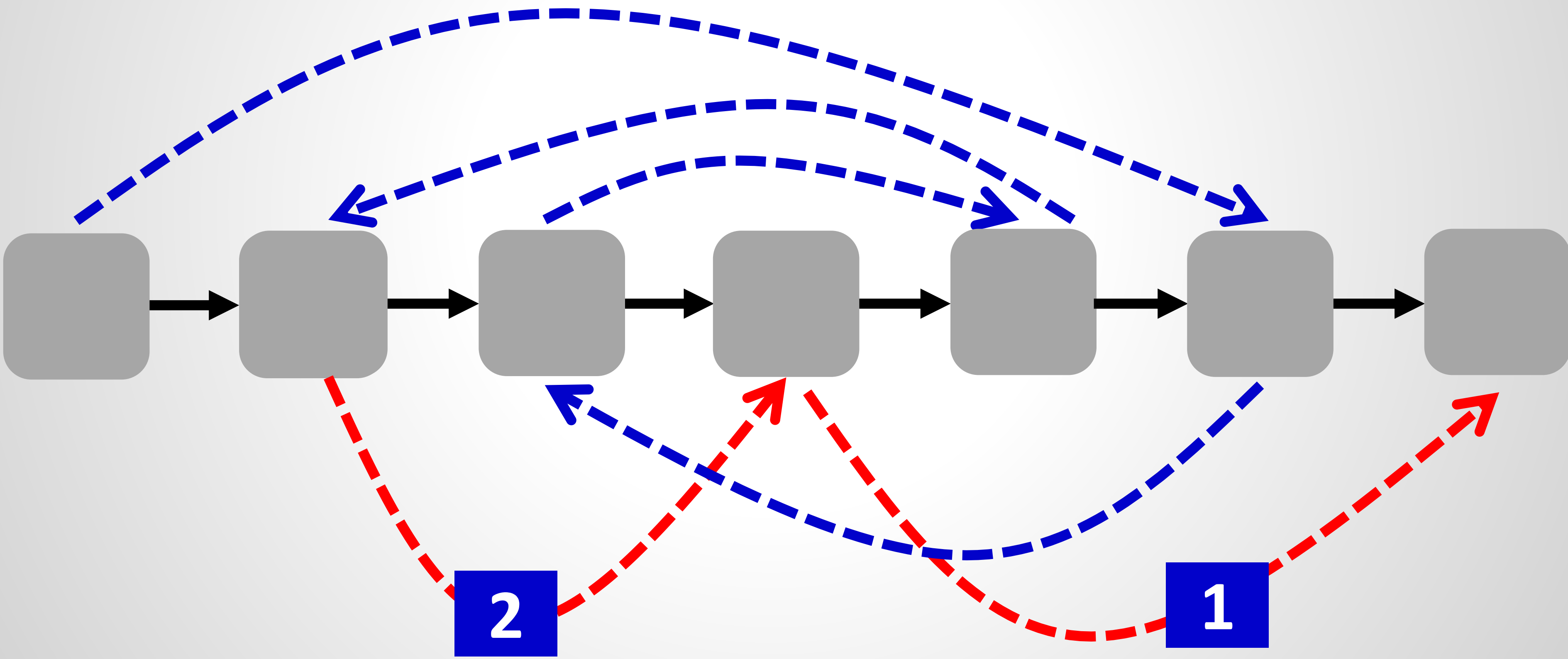
What about loop-freedom only?

Always possible in *n* rounds!



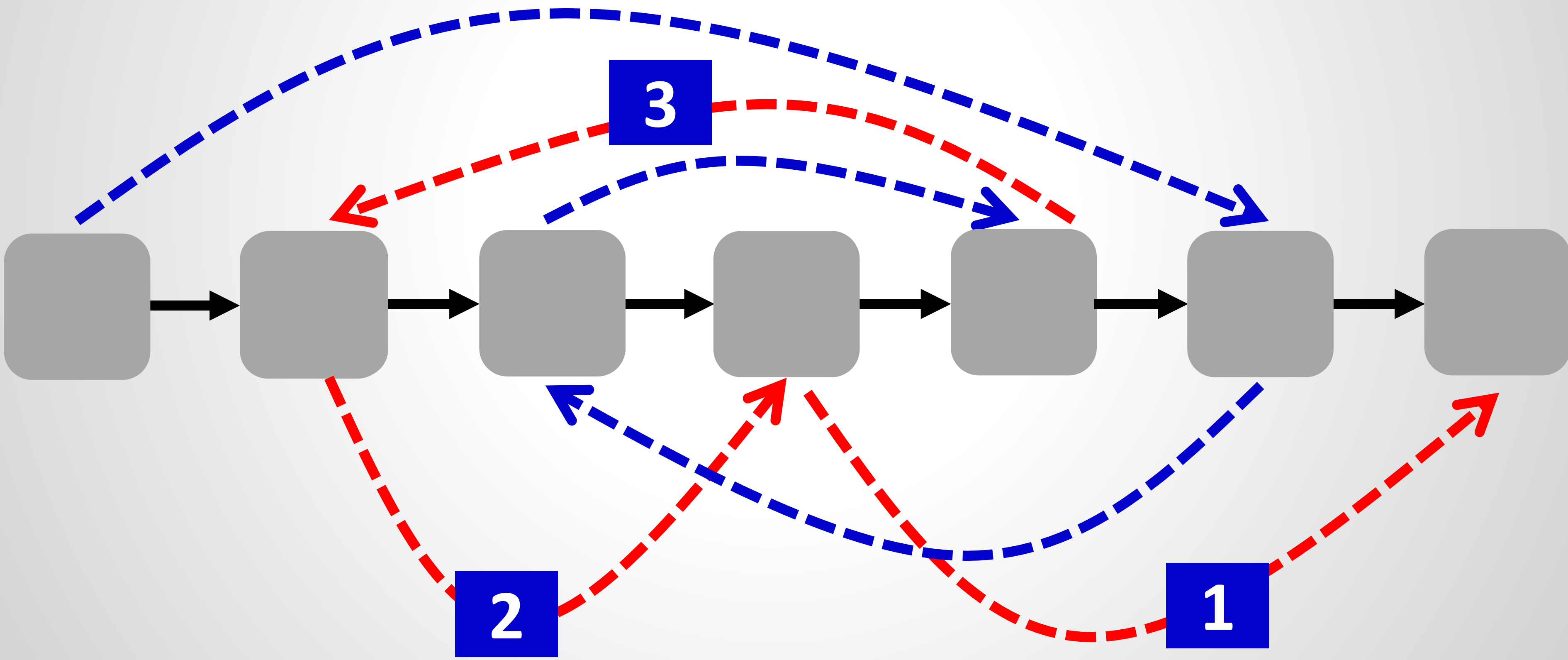
From the destination! Invariant: path suffix updated!

What about loop-freedom only?
Always possible in *n* rounds!



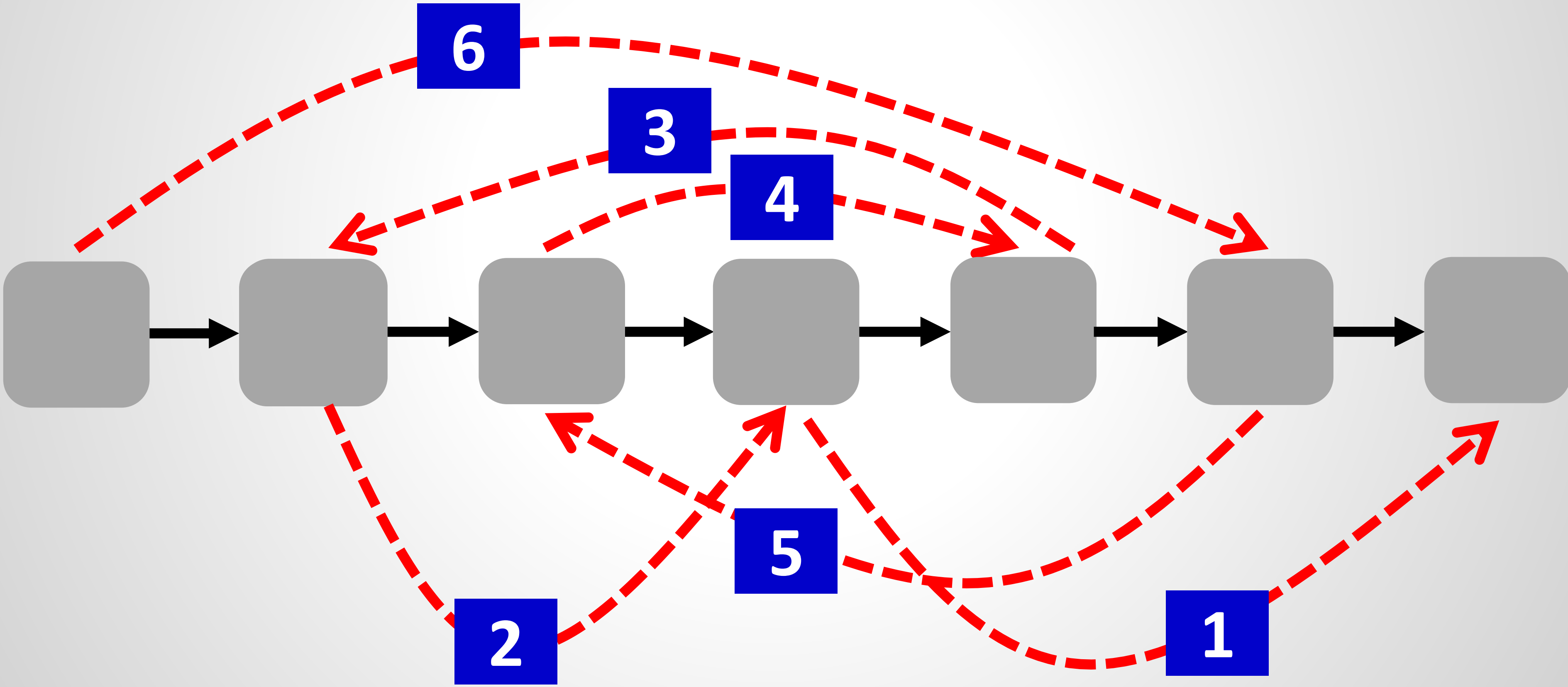
From the destination! Invariant: path suffix updated!

What about loop-freedom only? Always possible in *n* rounds!



From the destination! Invariant: path suffix updated!

What about loop-freedom only? Always possible in *n* rounds!



From the destination! Invariant: path suffix updated!

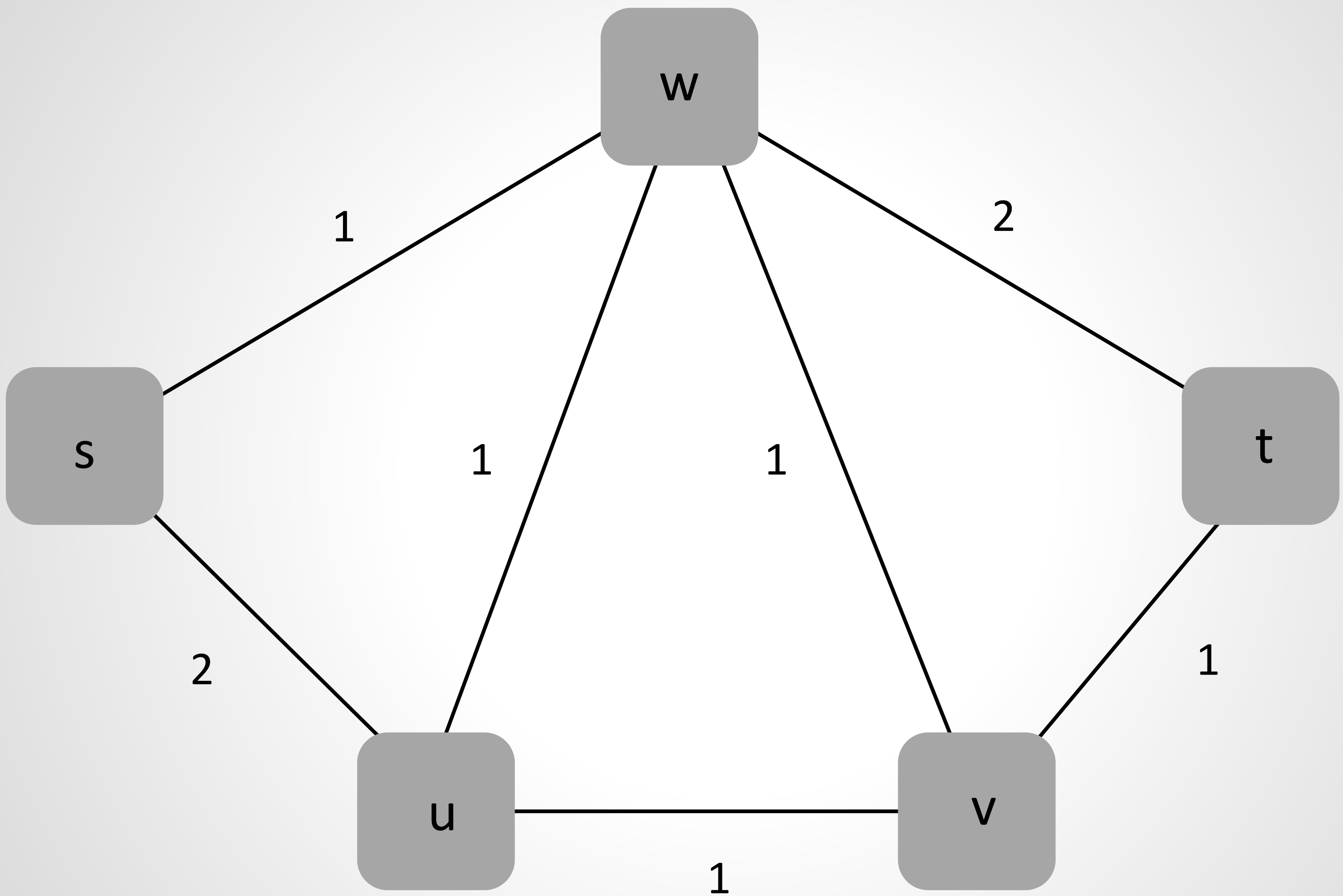
But how to minimize # rounds?

But how to minimize # rounds?

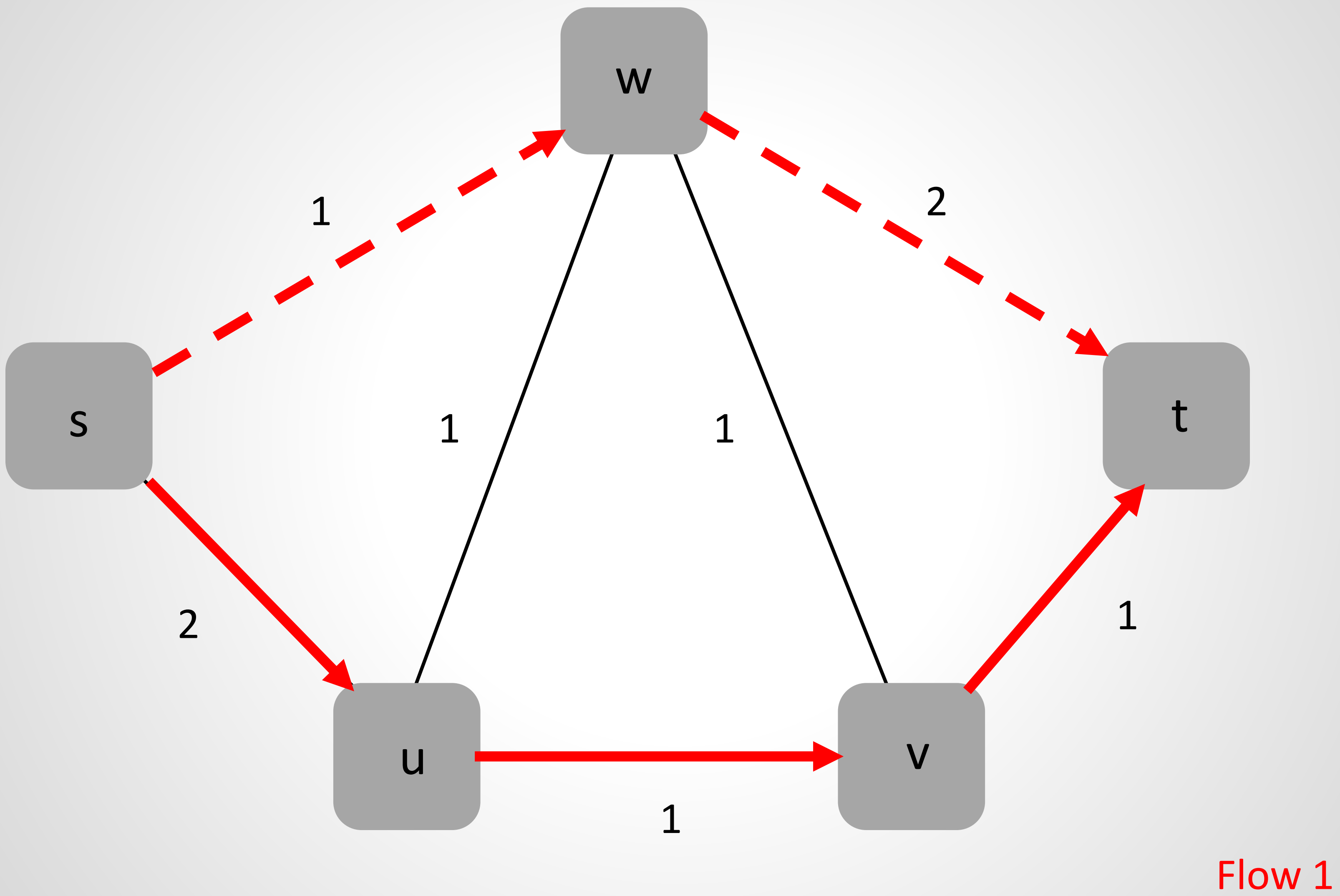
2 rounds easy, 3 rounds NP-hard. Everything else:

We don't know today!

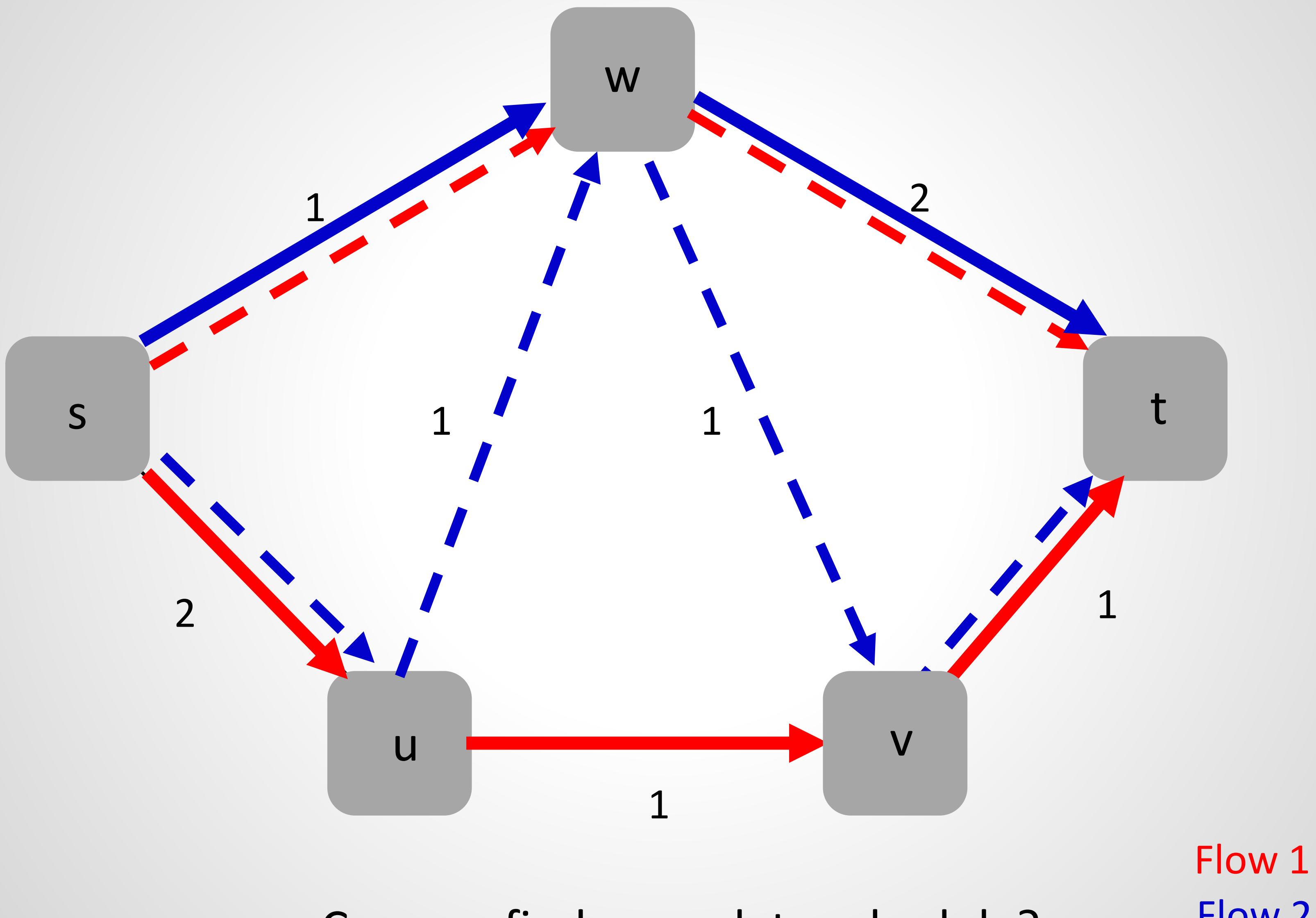
What about capacity constraints?



What about capacity constraints?



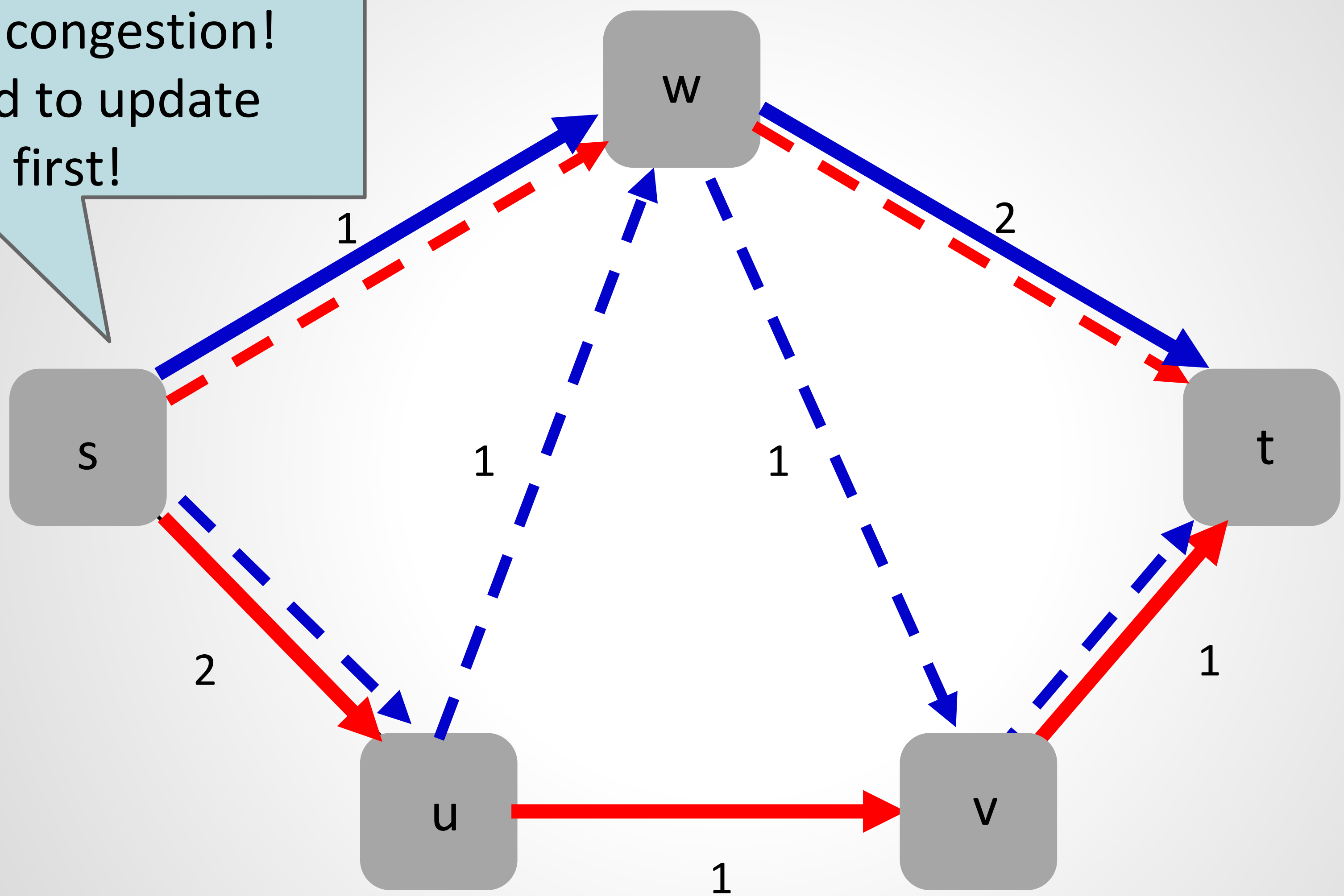
What about capacity constraints?



Can you find an update schedule?

What about capacity constraints?

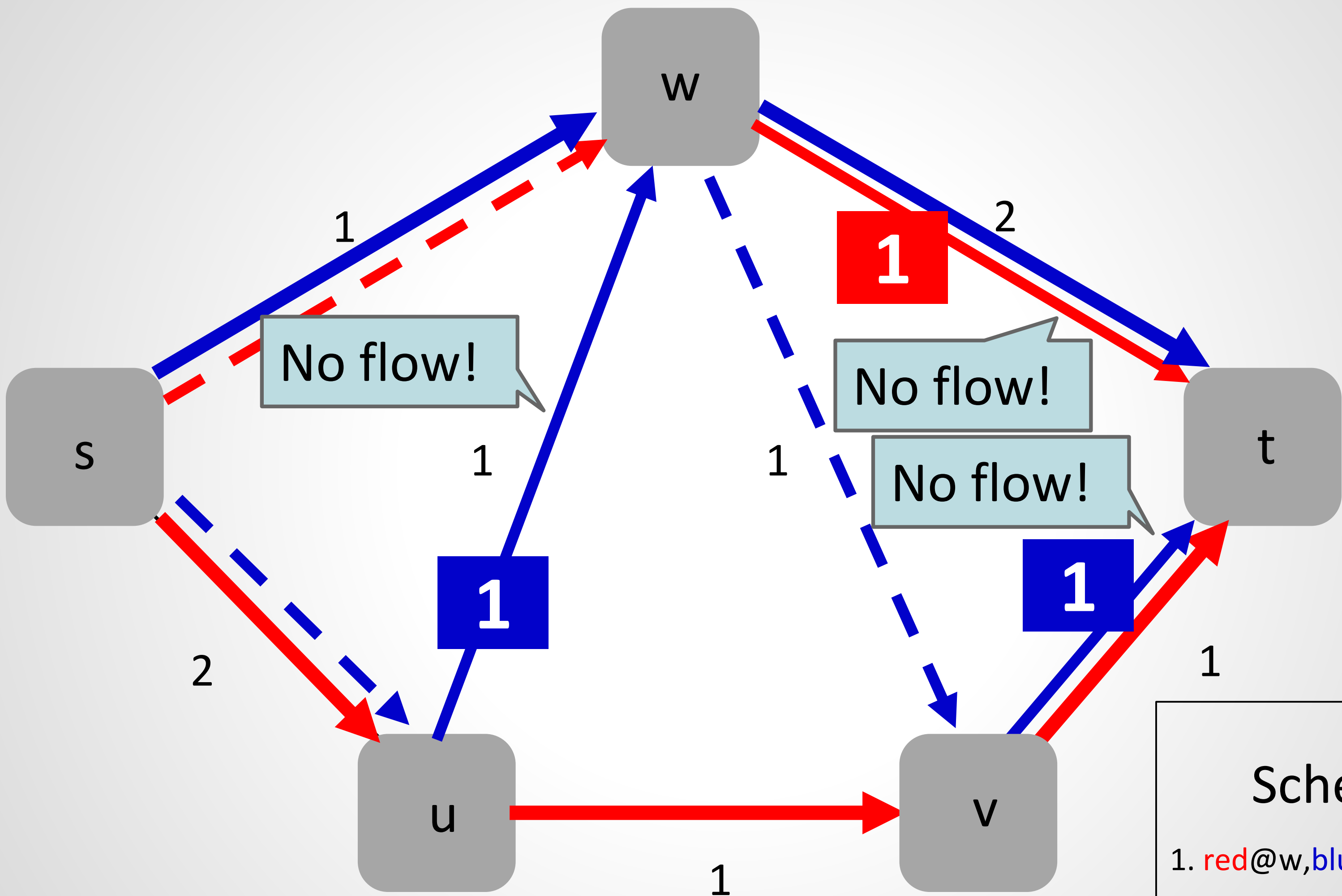
e.g., cannot update red: congestion! Need to update blue first!



Flow 1
Flow 2

Can you find an update schedule?

What about capacity constraints?

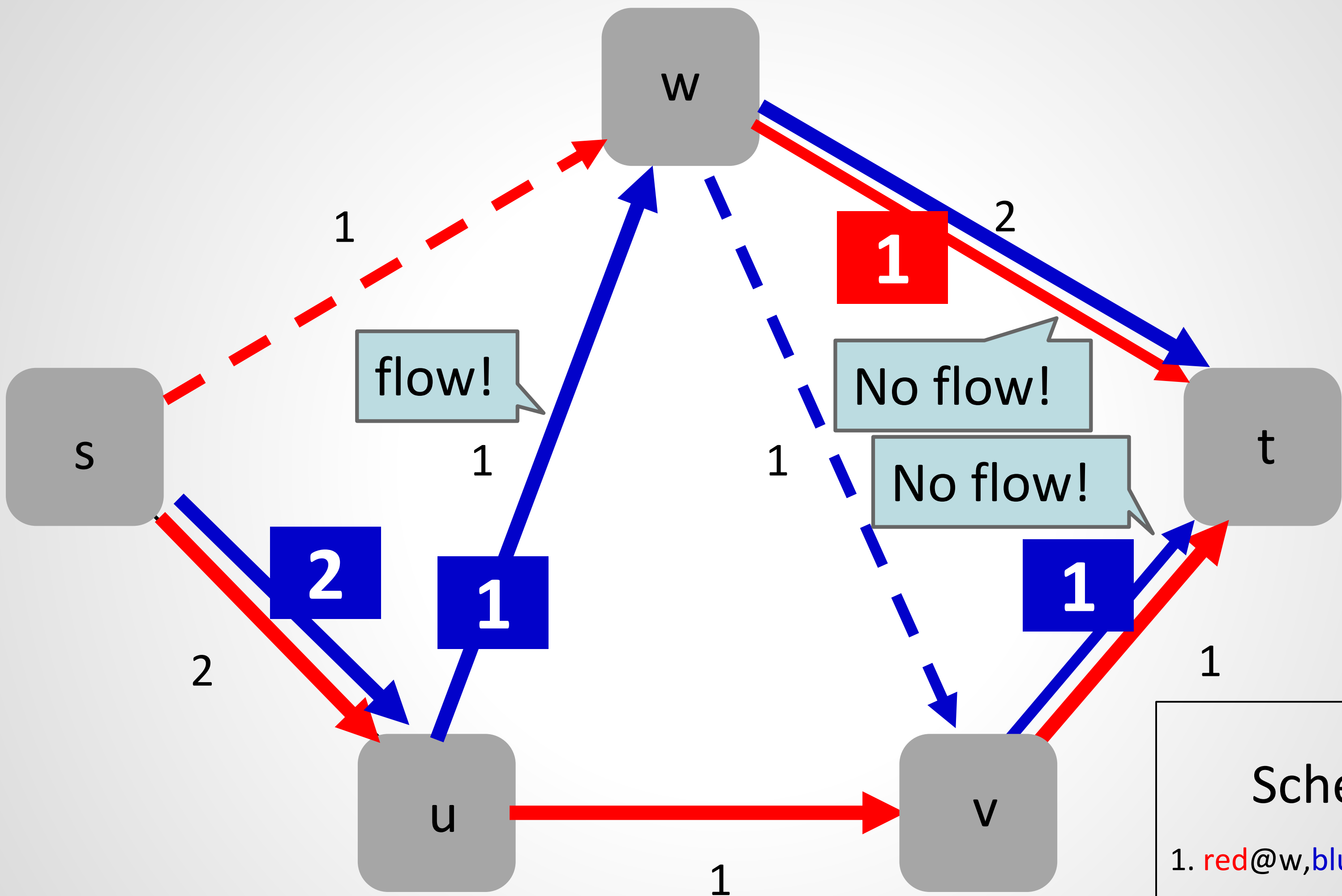


Schedule:

1. red@w,blue@u,blue@v

Round 1: prepare

What about capacity constraints?

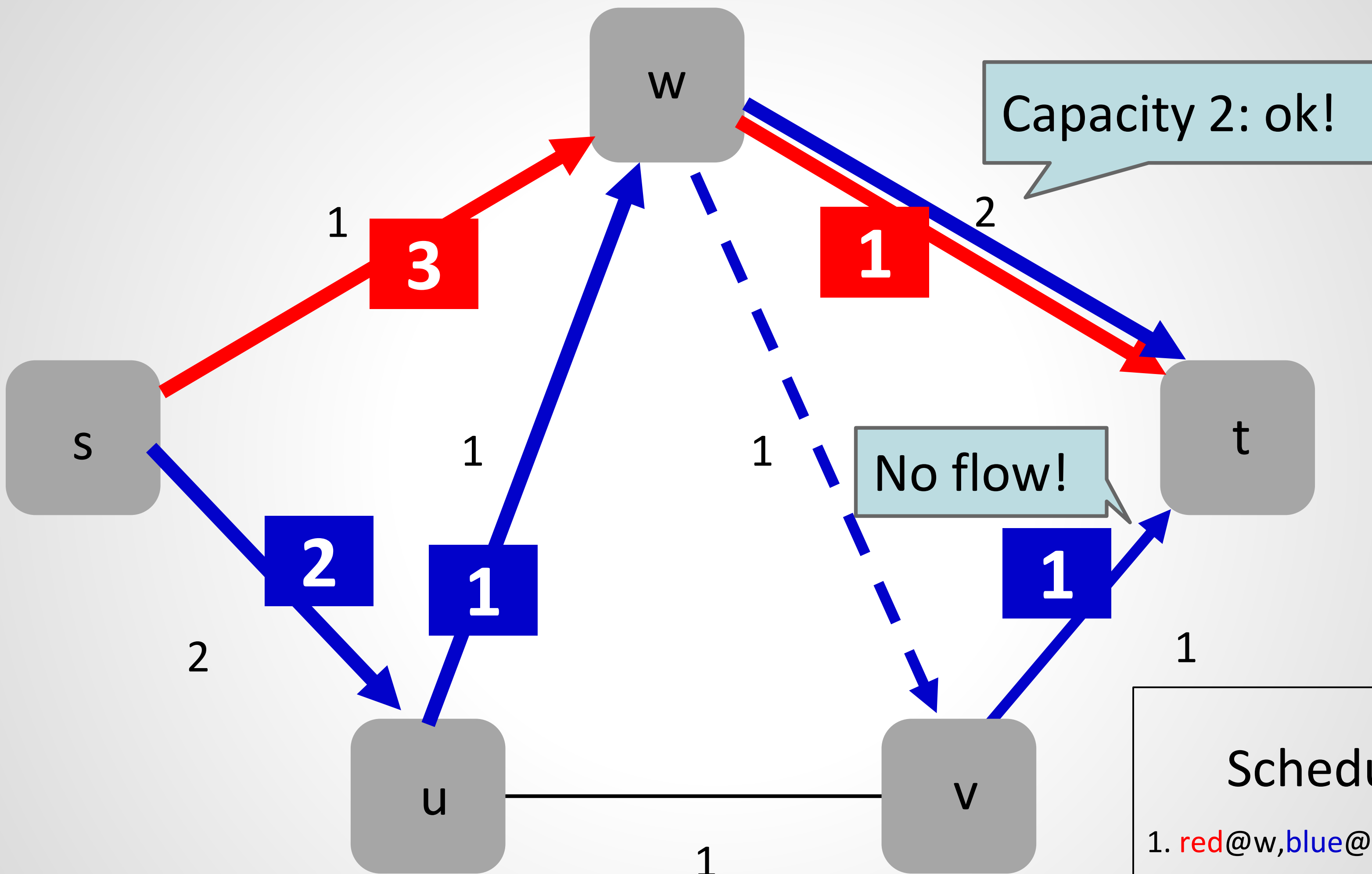


Round 2

Schedule:

1. red@w,blue@u,blue@v
2. blue@s

What about capacity constraints?

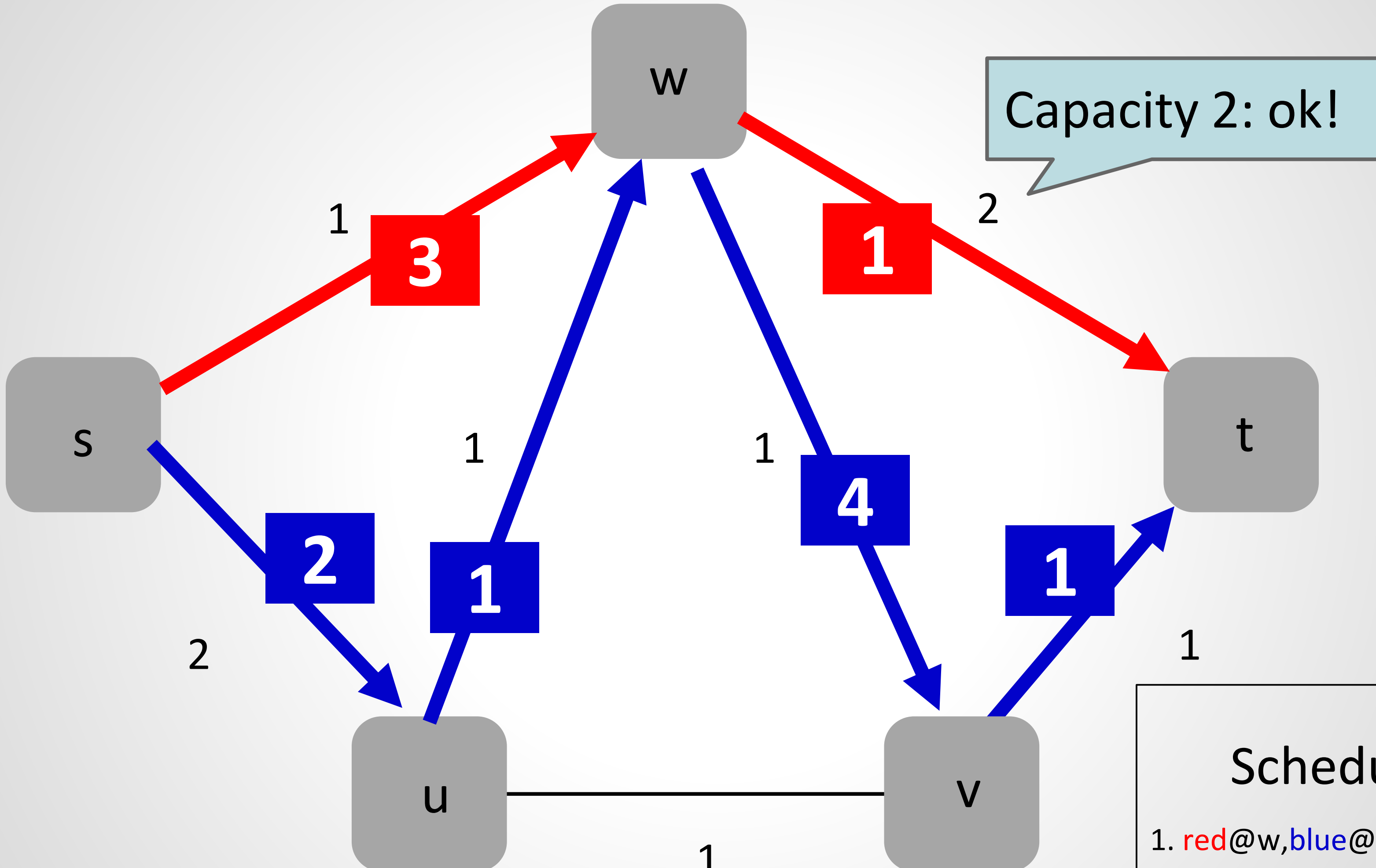


Schedule:

1. red@w,blue@u,blue@v
2. blue@s
3. red@s

Round 3

What about capacity constraints?

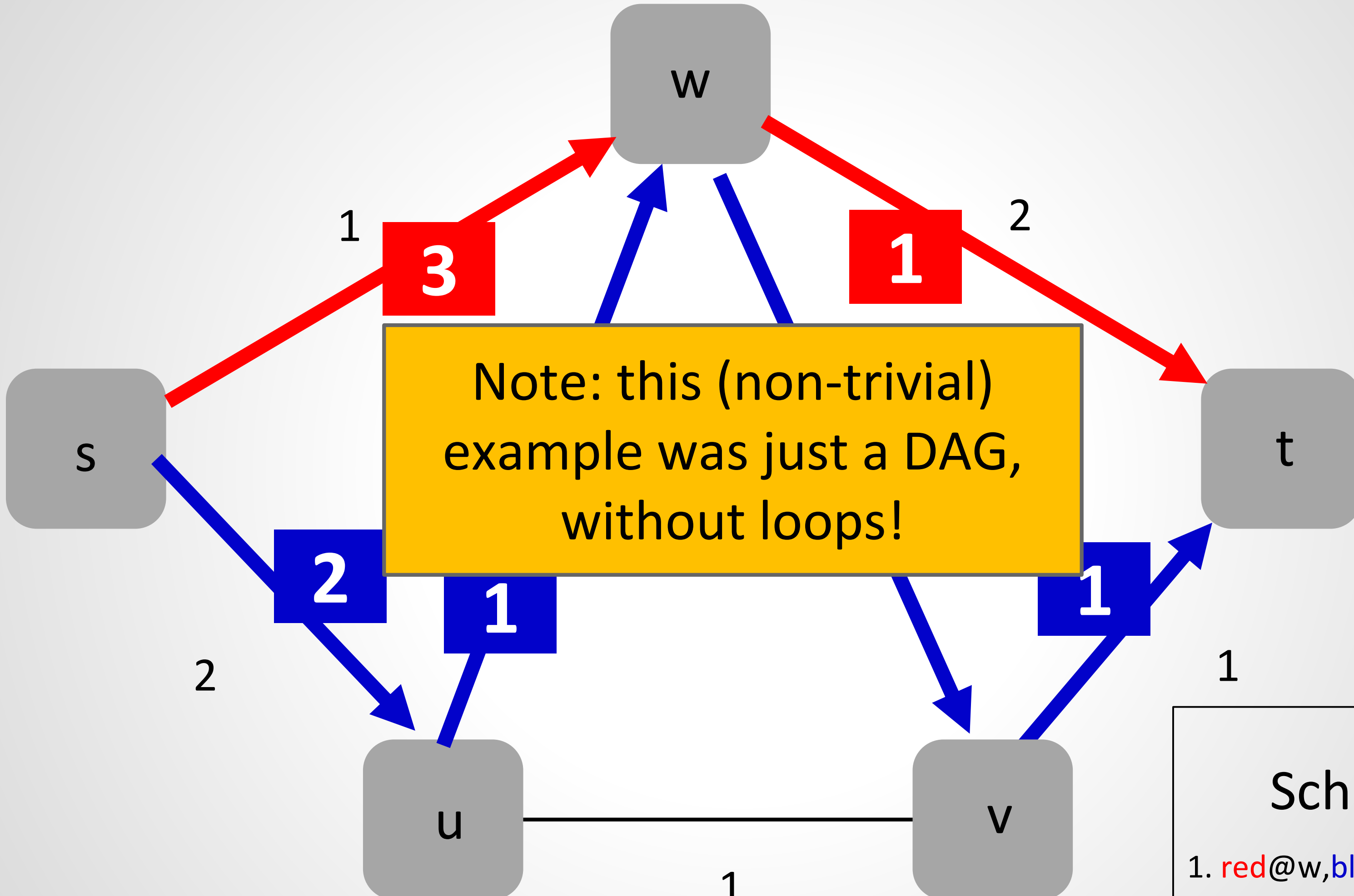


Schedule:

1. red@w,blue@u,blue@v
2. blue@s
3. red@s
4. blue@w

Round 4

What about capacity constraints?



Note: this (non-trivial) example was just a DAG, without loops!

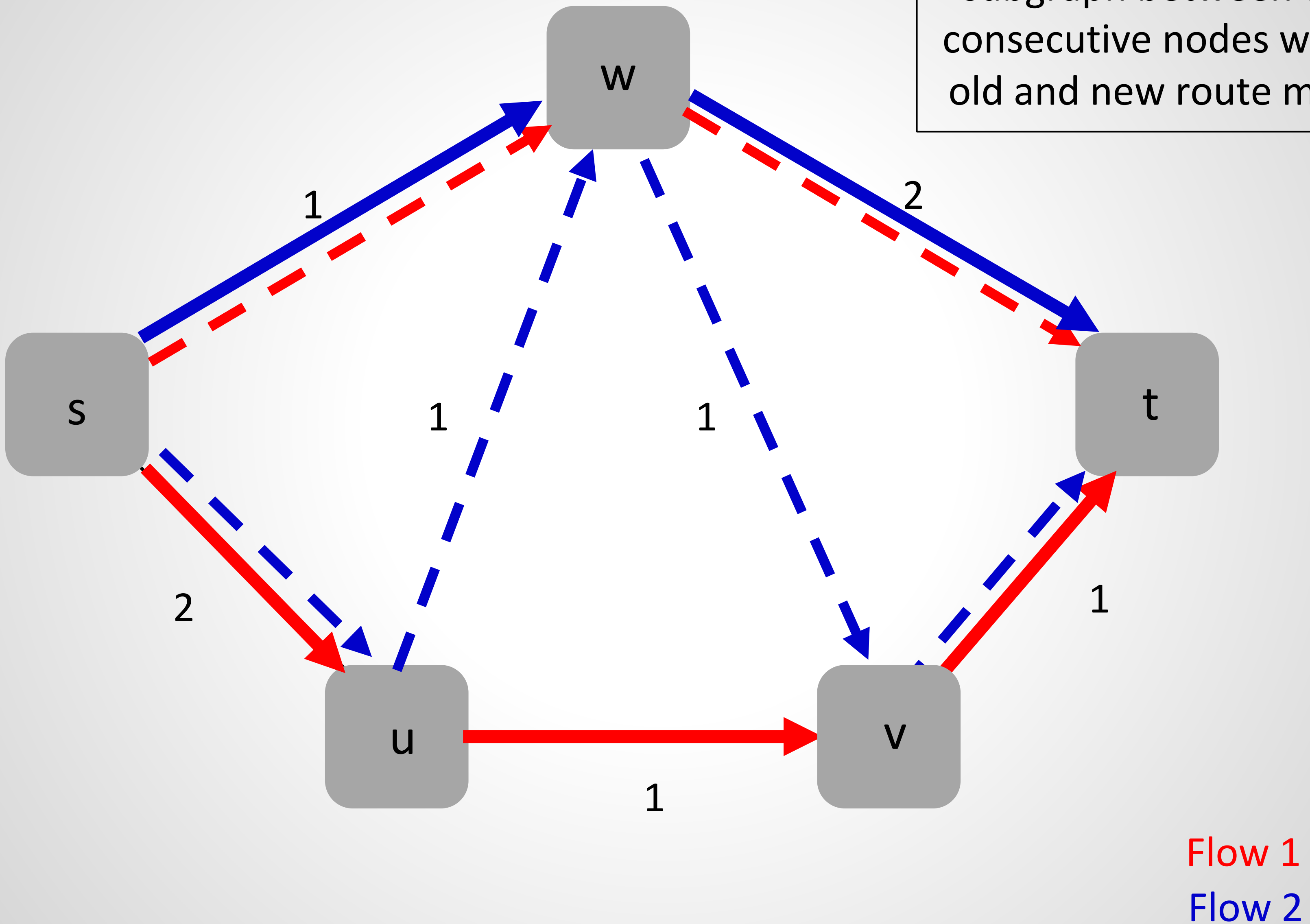
Round 4

Schedule:

1. red@w,blue@u,blue@v
2. blue@s
3. red@s
4. blue@w

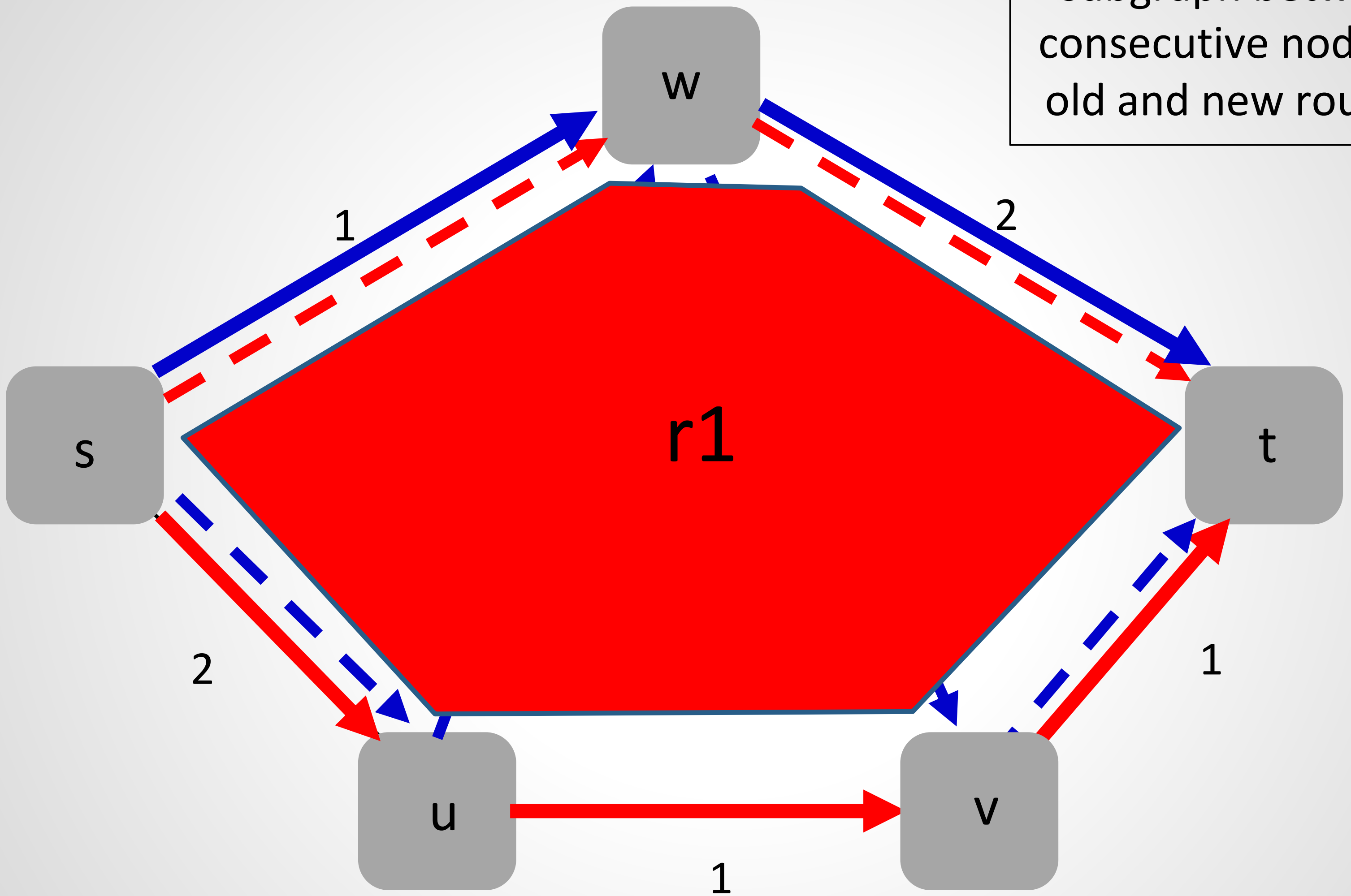
Solution: Dependency Graph on Block Decomposition of DAGs

Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Solution: Dependency Graph on Block Decomposition of DAGs

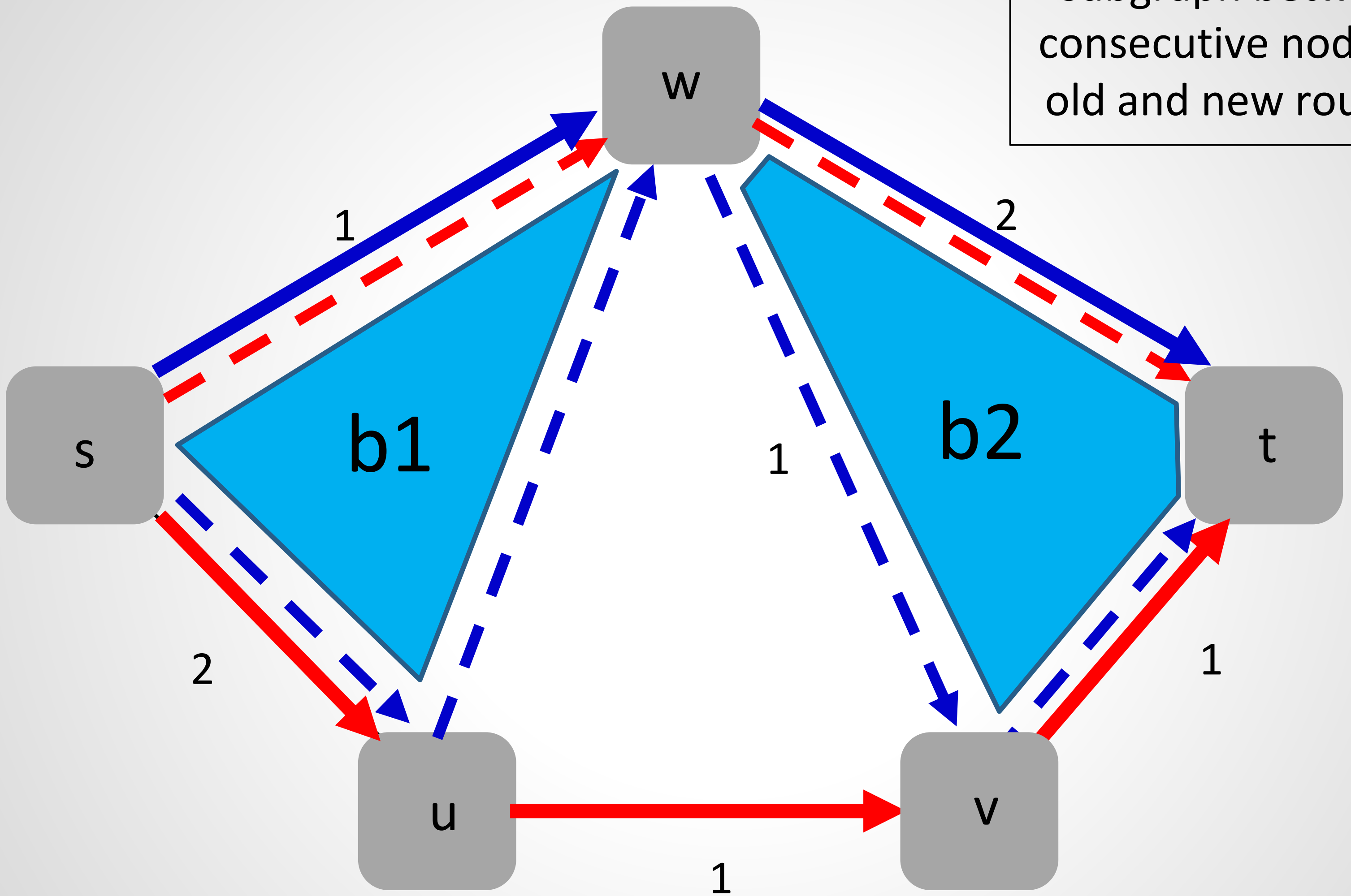
Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Just one red block: r1

Solution: Dependency Graph on Block Decomposition of DAGs

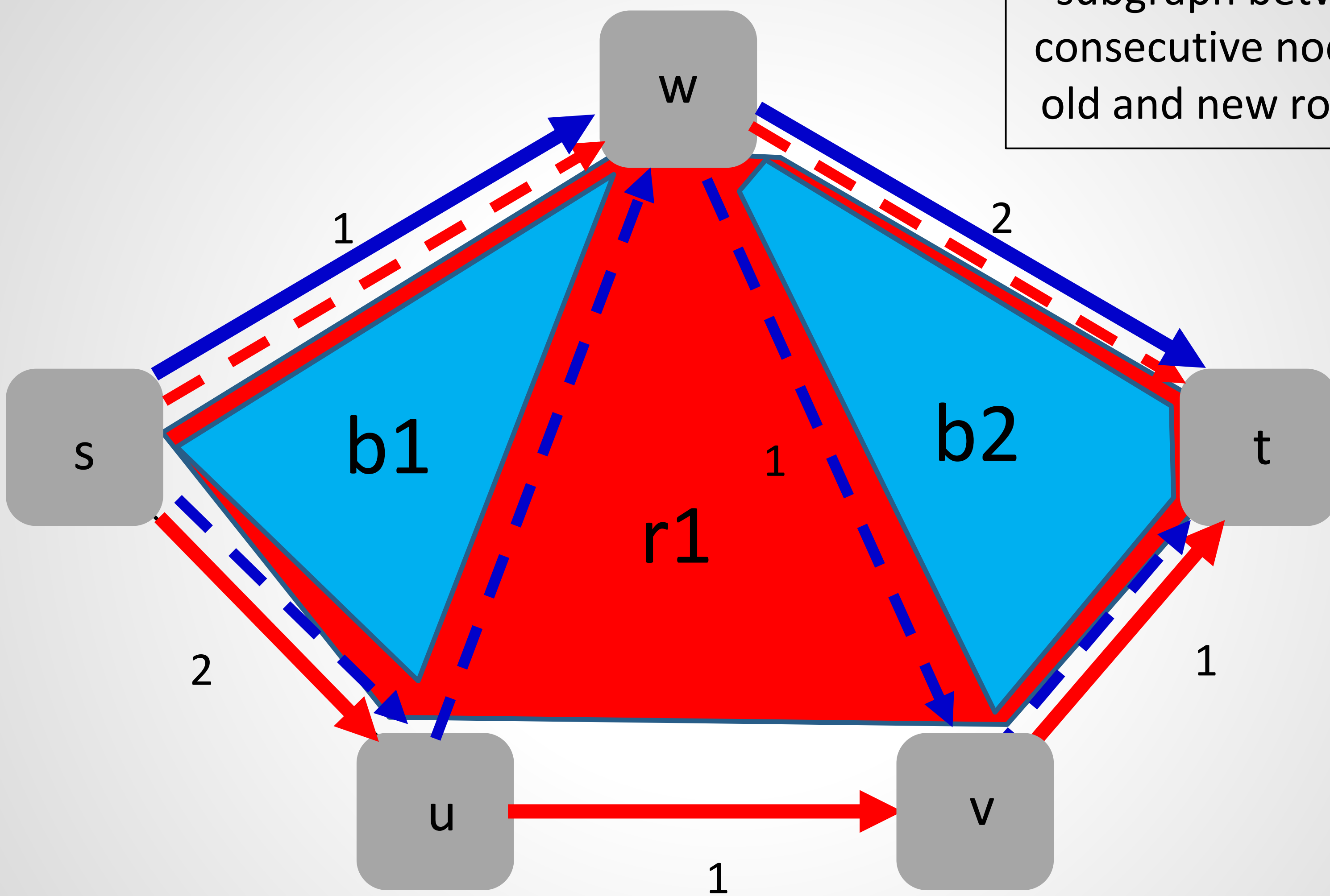
Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Two blue blocks: b1 and b2

Solution: Dependency Graph on Block Decomposition of DAGs

Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Dependencies: update b2 after r1 after b1.

Many Open Problems!

- We know for DAG:
  - For k=2 flows, polynomial-time algorithm to compute schedule with minimal number of rounds!
    - For general k, NP-hard
  - For general k flows, polynomial-time algorithm to compute feasible update
- Everything else: ***unkown!***
  - In particular: what if flow graph is not a DAG?

What’s new about this problem?

- Much classic literature on, e.g.,
  - Disruption-free IGP route changes
  - Ship-in-the-Night techniques

- SDN: new model (centralized and direct control of routes) and new properties
  - Not only connectivity consistency but also policy consistency (e.g., waypoints) and performance consistency

Further reading: 35-page survey!

Survey of Consistent Network Updates
Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio. ArXiv Technical Report, September 2016.

Further Reading:

Can't Touch This: Consistent Network Updates for Multiple Policies
Szymon Dudycz, Arne Ludwig, and Stefan Schmid.
46th IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**), Toulouse, France, June 2016.

loop-freedom
multiple policies

Transiently Secure Network Updates
Arne Ludwig, Szymon Dudycz, Matthias Rost, and Stefan Schmid.
42nd ACM **SIGMETRICS**, Antibes Juan-les-Pins, France, June 2016.

waypointing

Scheduling Loop-free Network Updates: It's Good to Relax!
Arne Ludwig, Jan Marcinkowski, and Stefan Schmid.
ACM Symposium on Principles of Distributed Computing (**PODC**), Donostia-San Sebastian, Spain, July 2015.

loop-freedom

Good Network Updates for Bad Packets: Waypoint Enforcement Beyond Destination-Based Routing Policies
Arne Ludwig, Matthias Rost, Damien Foucard, and Stefan Schmid.
13th ACM Workshop on Hot Topics in Networks (**HotNets**), Los Angeles, California, USA, October 2014.

waypointing

Congestion-Free Rerouting of Flows on DAGs
Saeed Akhoondian Amiri, Szymon Dudycz, Stefan Schmid, and Sebastian Wiederrecht.
ArXiv Technical Report, November 2016.

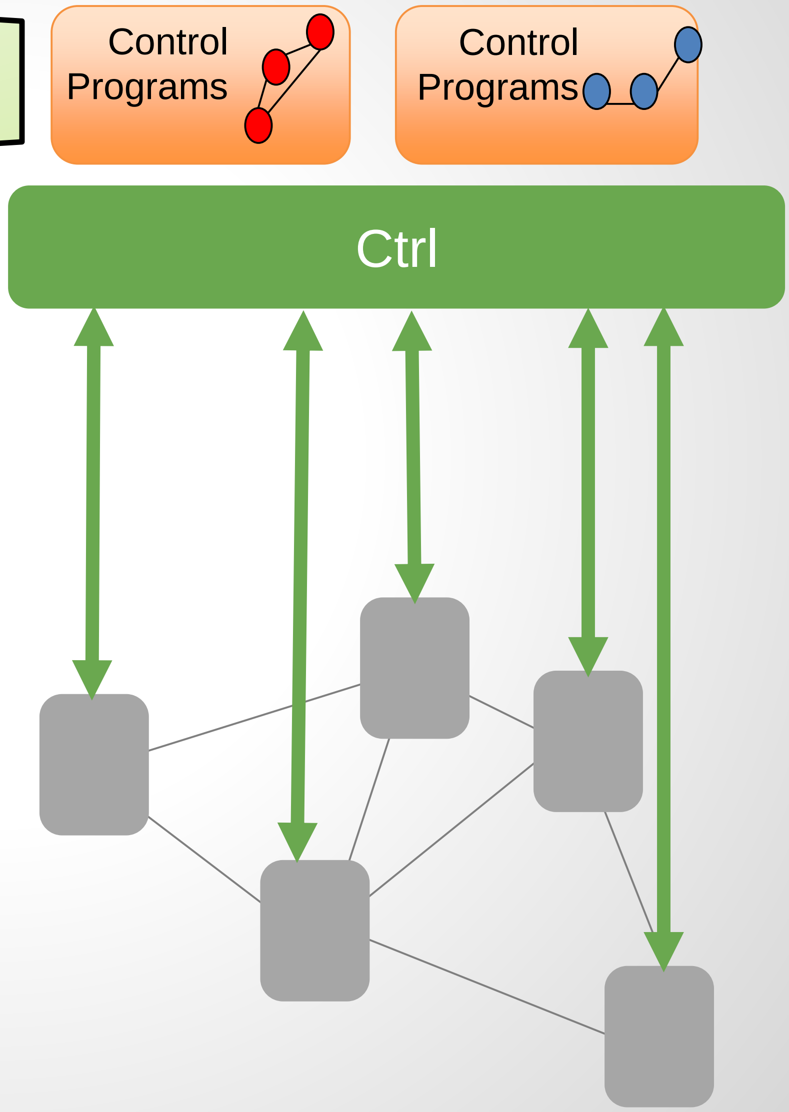
capacity constraints

Survey of Consistent Network Updates
Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio.
ArXiv Technical Report, September 2016.

survey

A Mental Model for This Talk

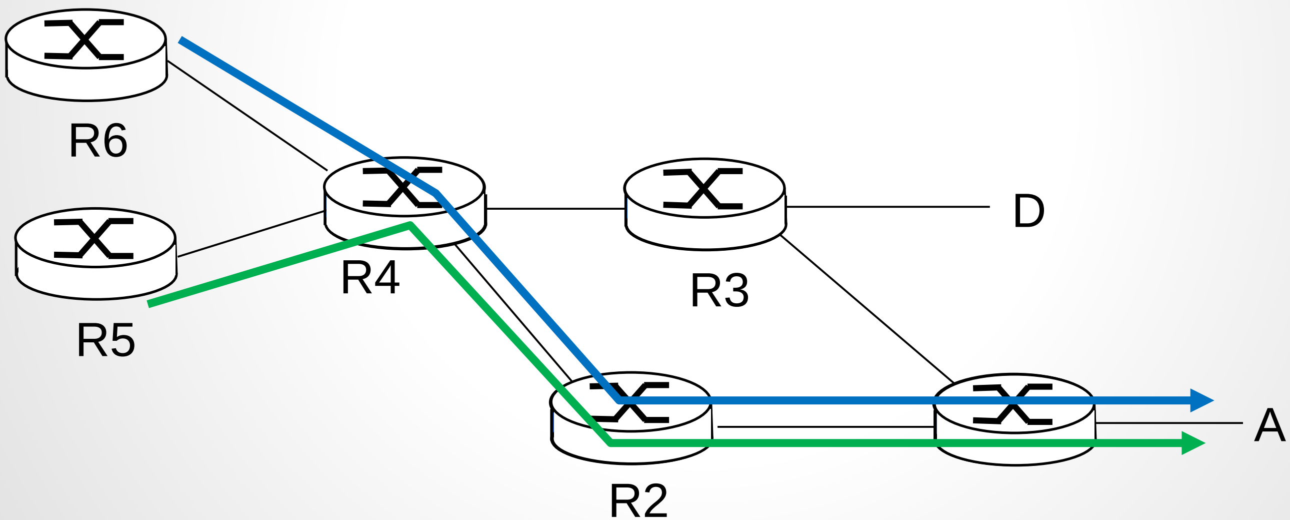
Opportunity: innovative services and algorithms



Example Benefit: Traffic Engineering

- Traditionally: shortest paths, IP destination-based
- SDN: non-shortest, non-confluent, may depend on other header fields (e.g., TCP port), etc.

Example: limitation of traditional networks



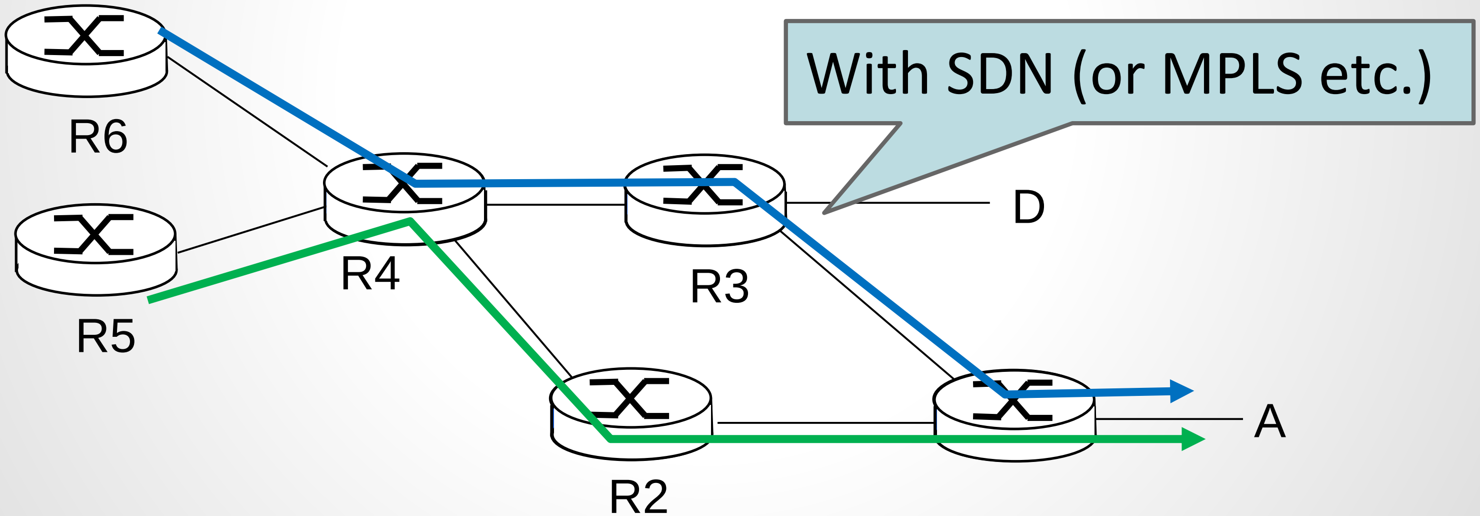
Node R4 can‘t route blue and green traffic differently: same destination (destination-based)!

Credits: Kurose&Ross, Top-Down Approach

Example Benefit: Traffic Engineering

- Traditionally: shortest paths, IP destination-based
- SDN: non-shortest, non-confluent, may depend on other header fields (e.g., TCP port), etc.

Example: limitation of traditional networks

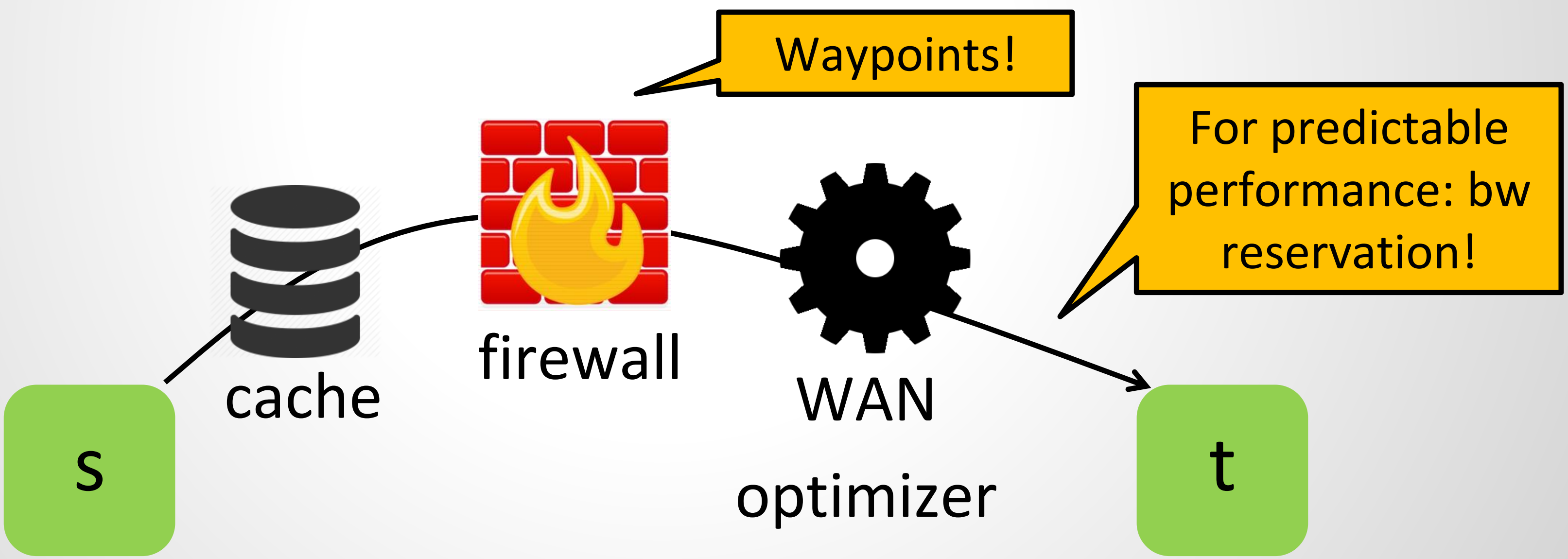


Node R4 can‘t route blue and green traffic differently: same destination (destination-based)!

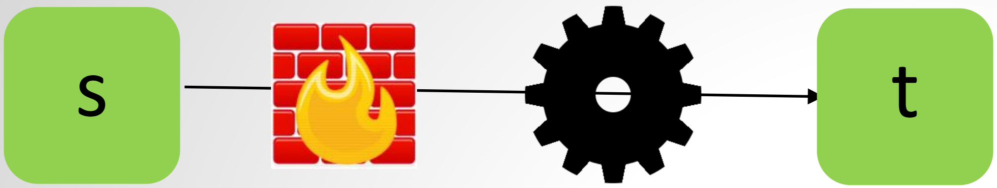
Credits: Kurose&Ross, Top-Down Approach

Example Benefit: Waypoint Routing

- SDN supports even more complex routes
- For example, service chain: traffic is steered (e.g., using SDN) through a sequence of (virtualized) middleboxes to compose a more complex network service

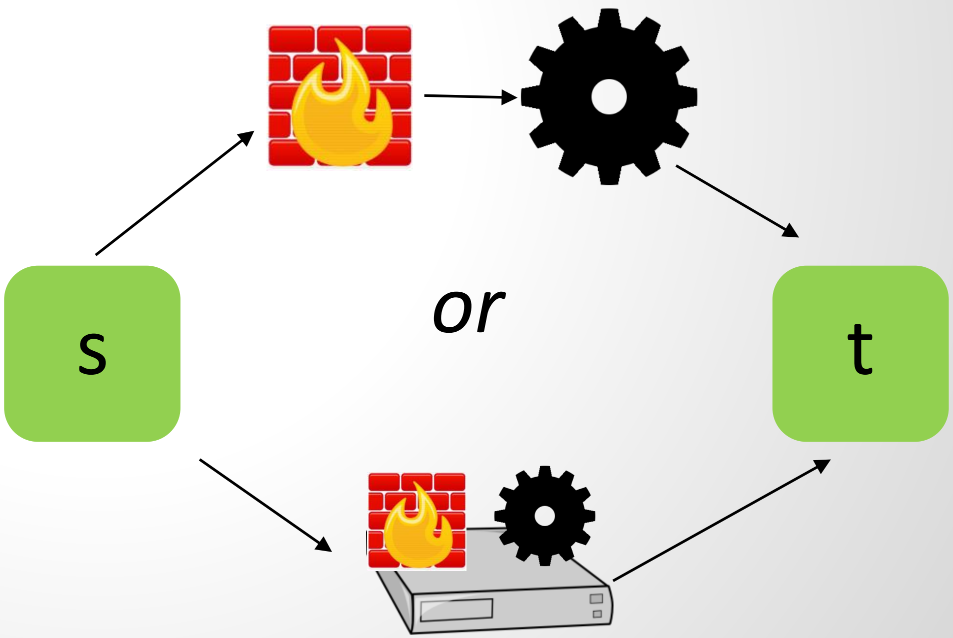


Requests can be more complex

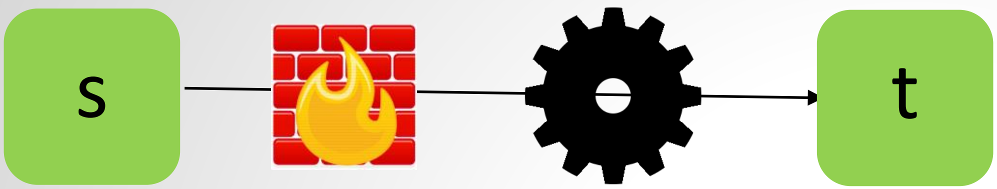


Already non-trivial!

And what if requests allow for alternatives and different decompositions?

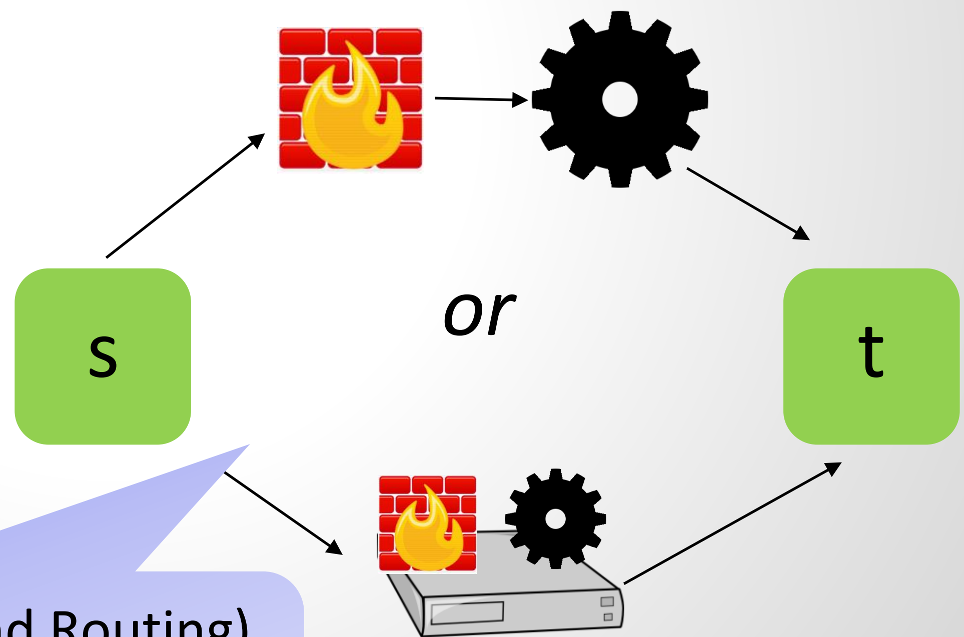


Requests can be more complex



Already non-trivial!

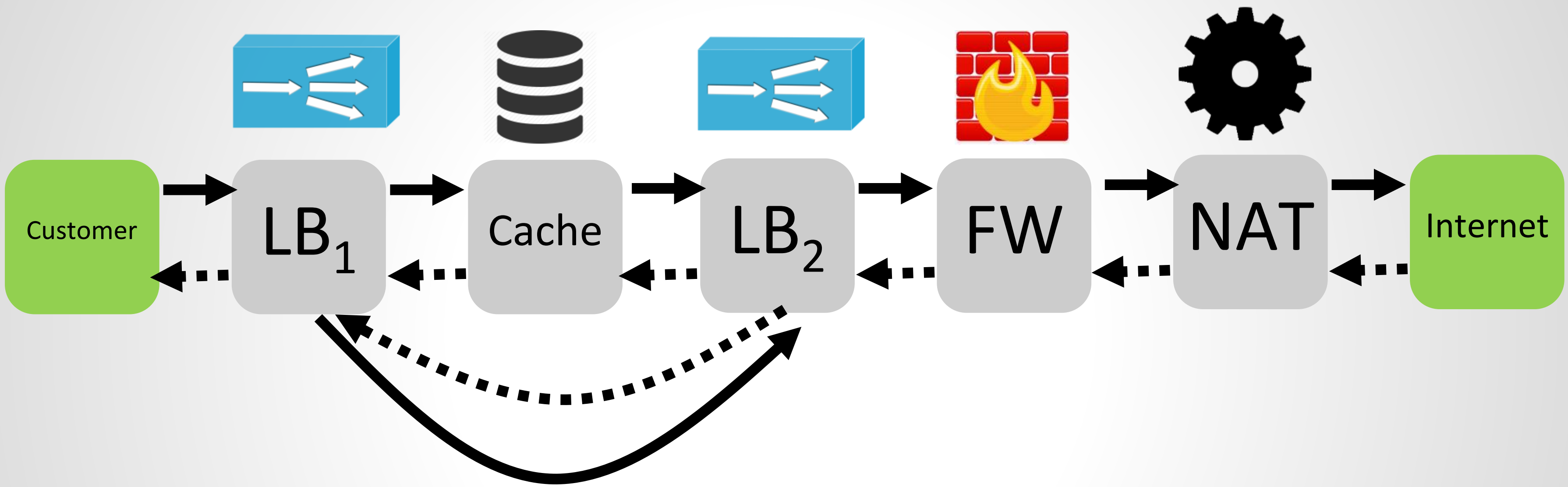
And what if requests allow for alternatives and different decompositions?



Known as PR (Processing and Routing) Graph: allows to model different choices and implementations!

What about this one?!

IETF Draft:

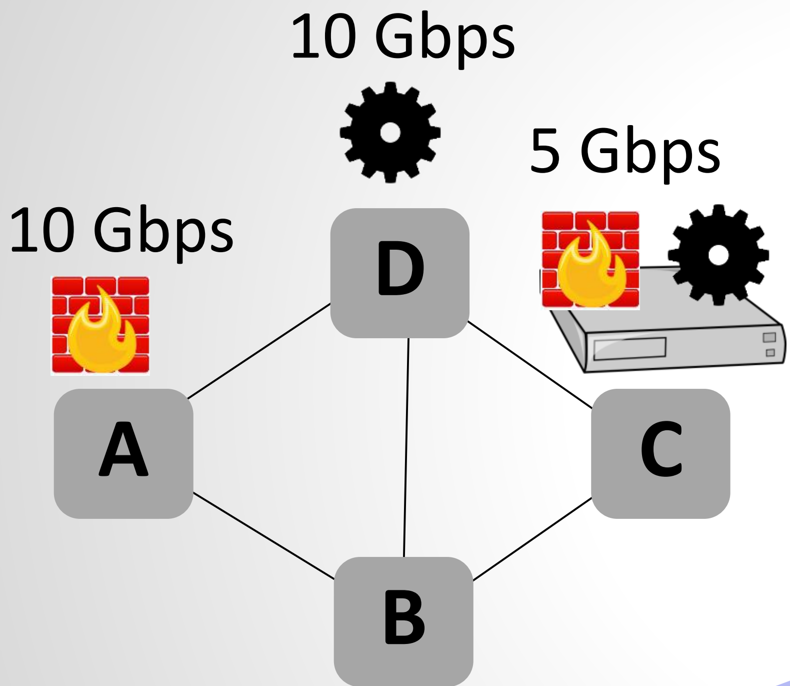


- ❑ Service chain for mobile operators
- ❑ Load-balancers are used to route (parts of) the traffic through cache

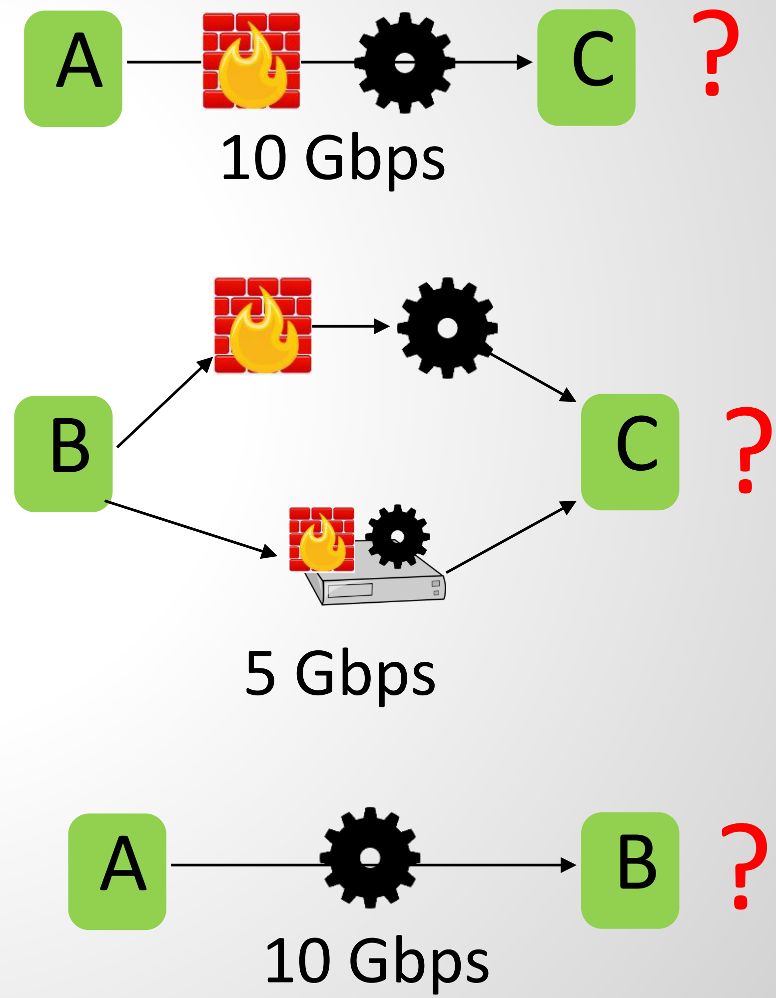
Credits: https://tools.ietf.org/html/draft-ietf-sfc-use-case-mobility-06

Example: admission control and embedding

Substrate:



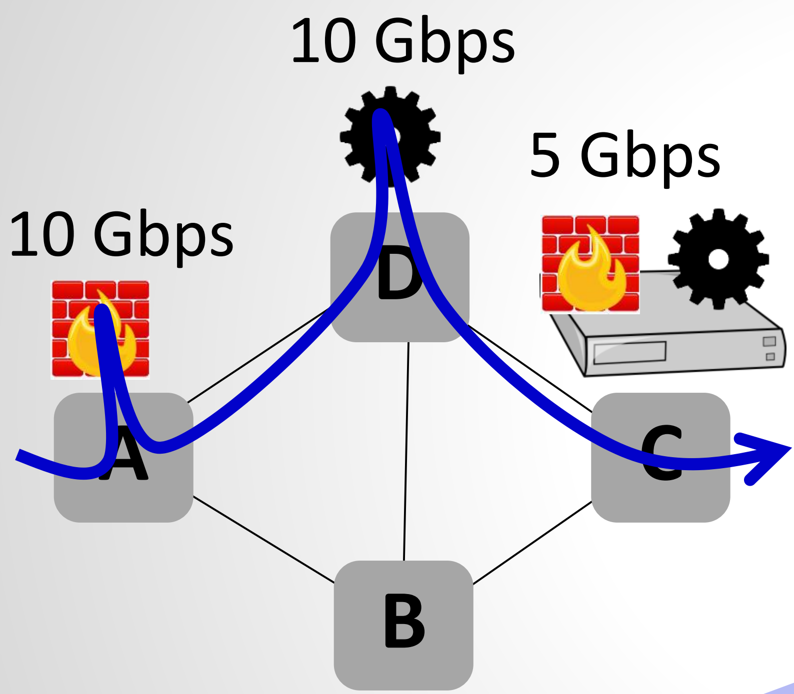
Requests:



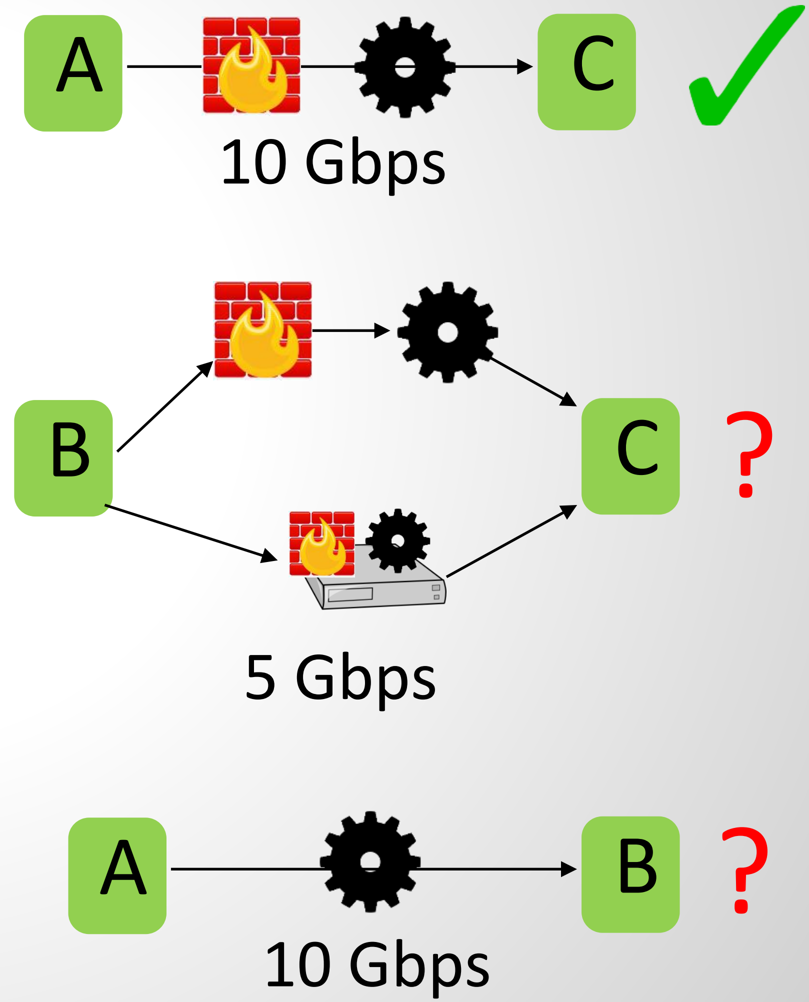
Which ones can be
admitted and embedded?

Example: admission control and embedding

Substrate:



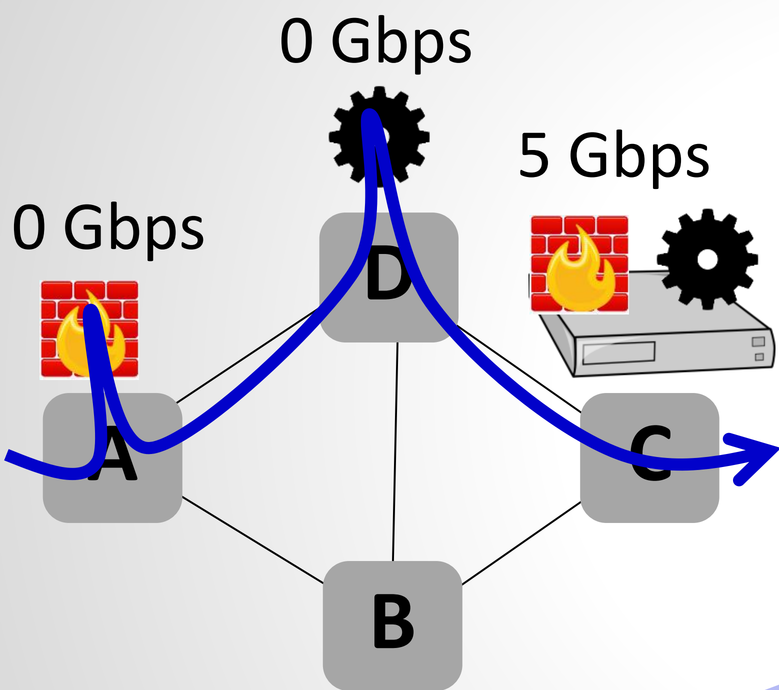
Requests:



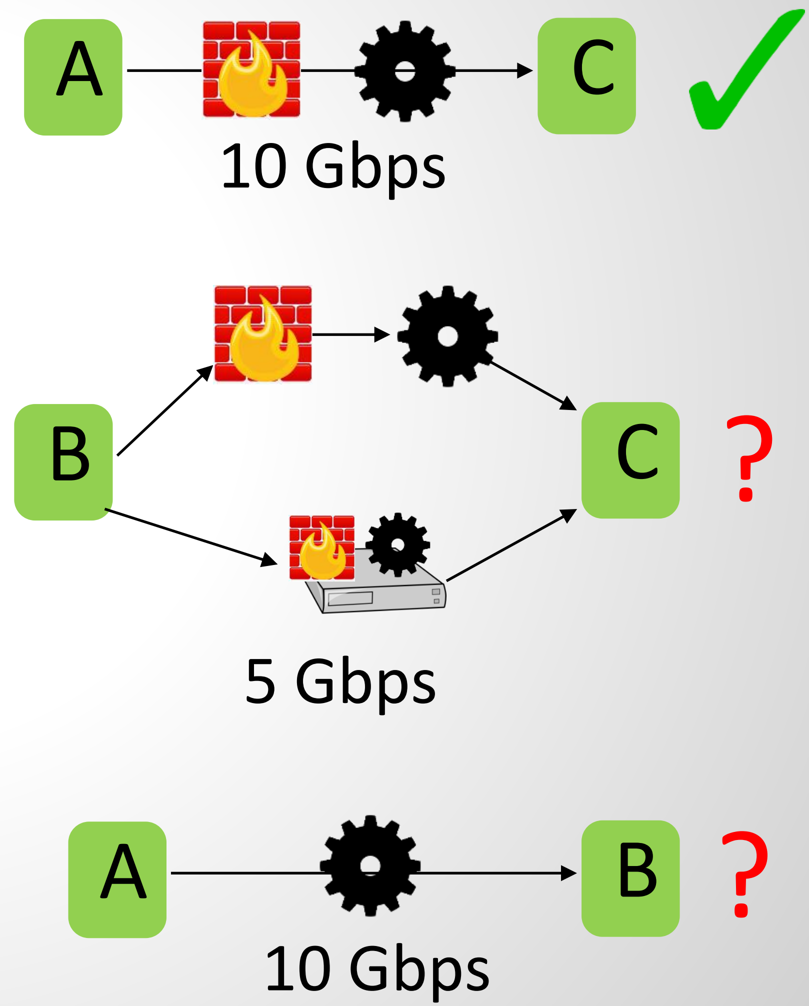
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



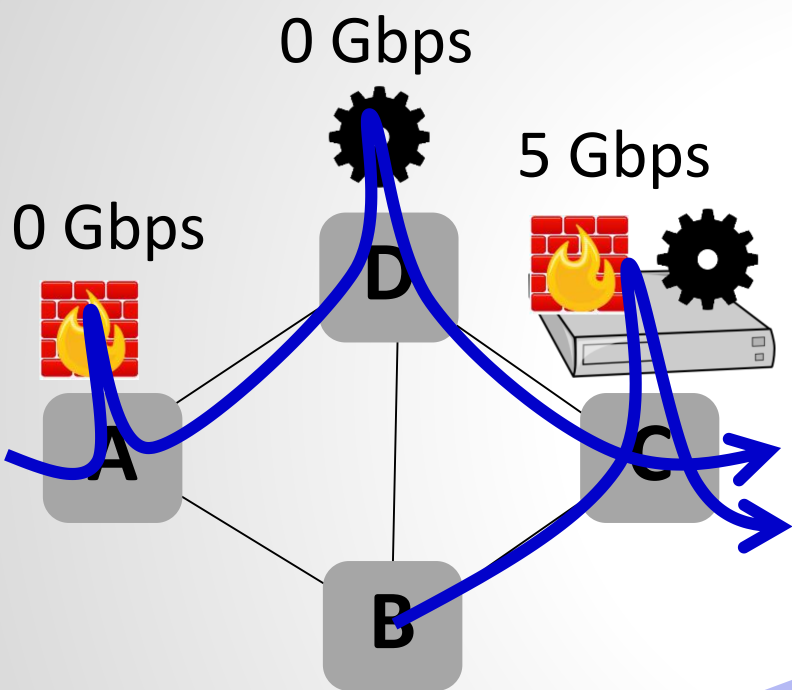
Requests:



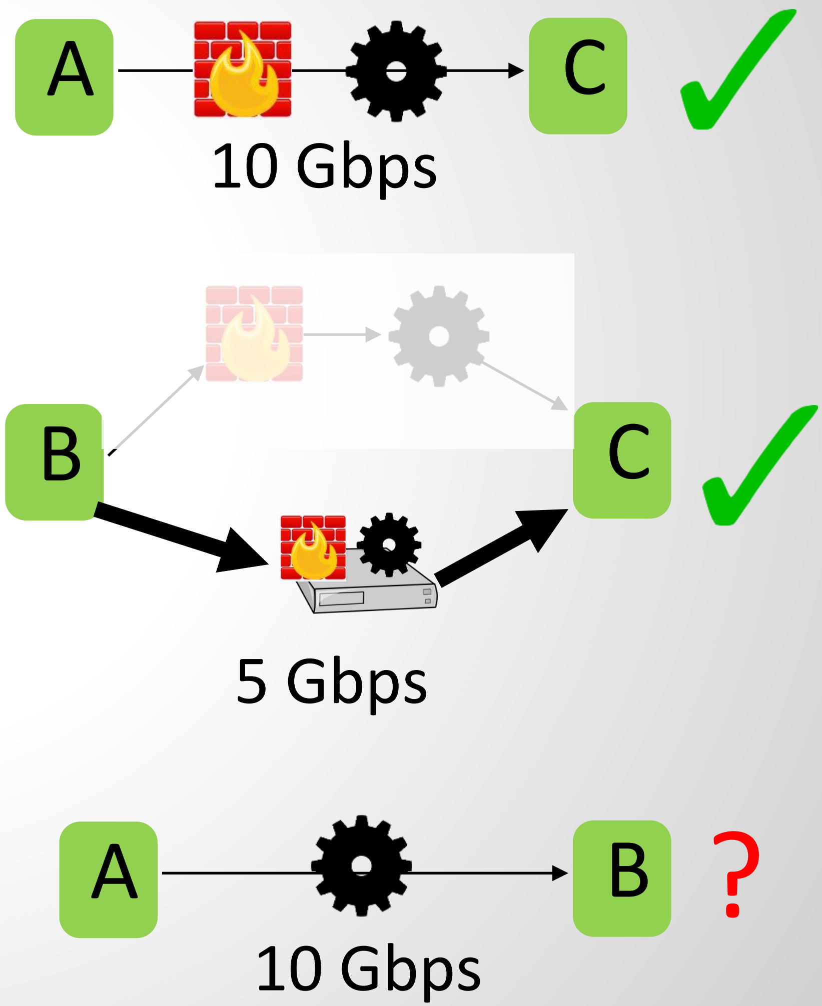
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



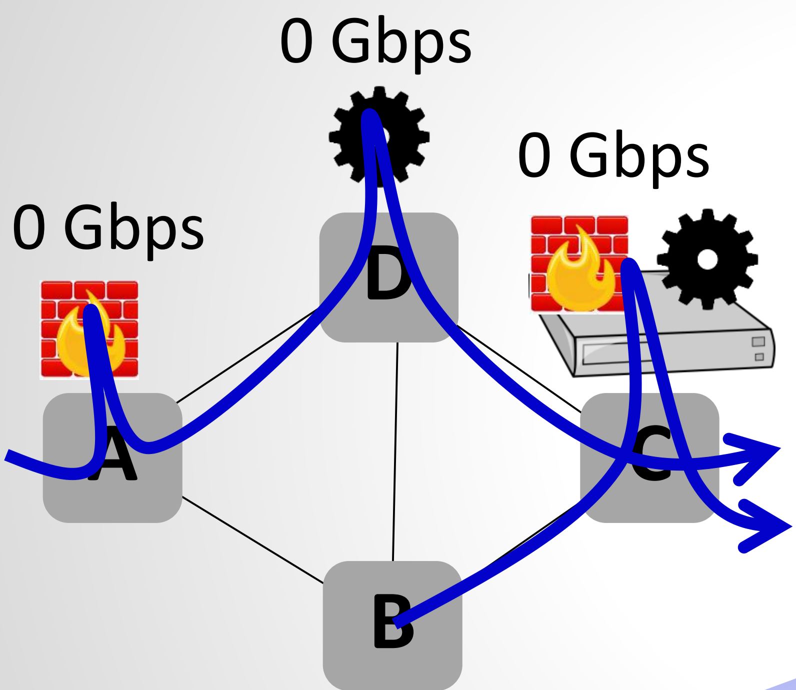
Requests:



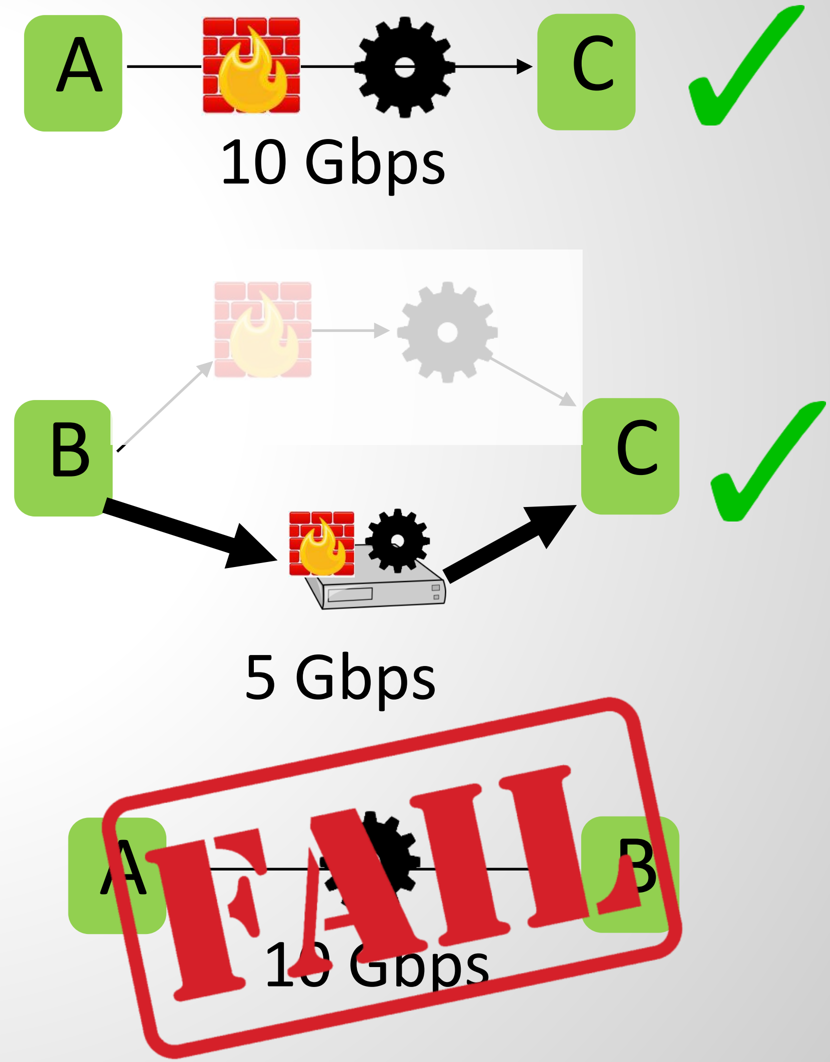
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



Requests:



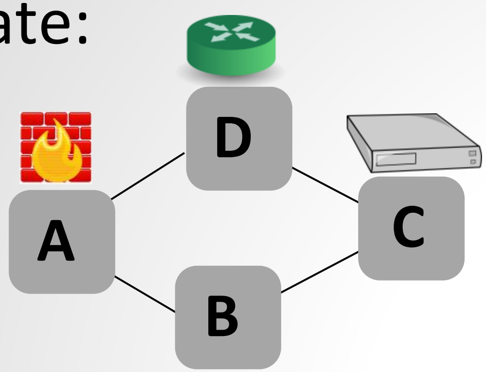
Which ones can be admitted and embedded?

Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for "Fairly Simple" Requests*!

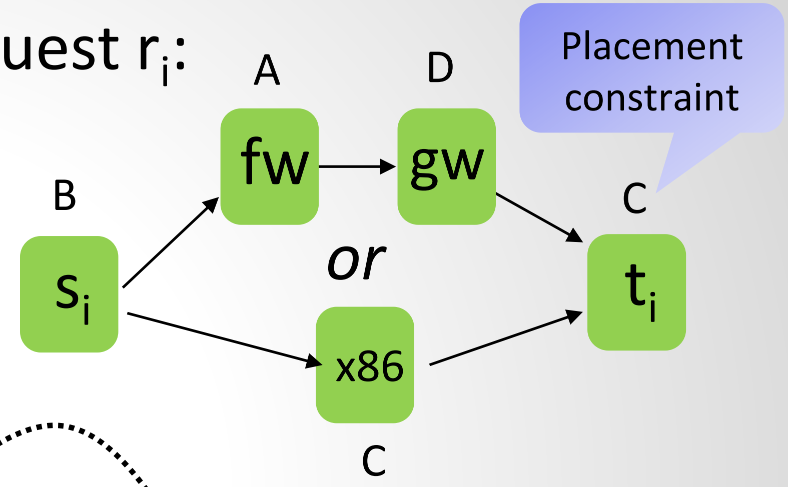
Chains, alternative chains, but even trees. Trick: **reduction to flow problem** using **product graphs**.

Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for “Fairly Simple” Requests*!

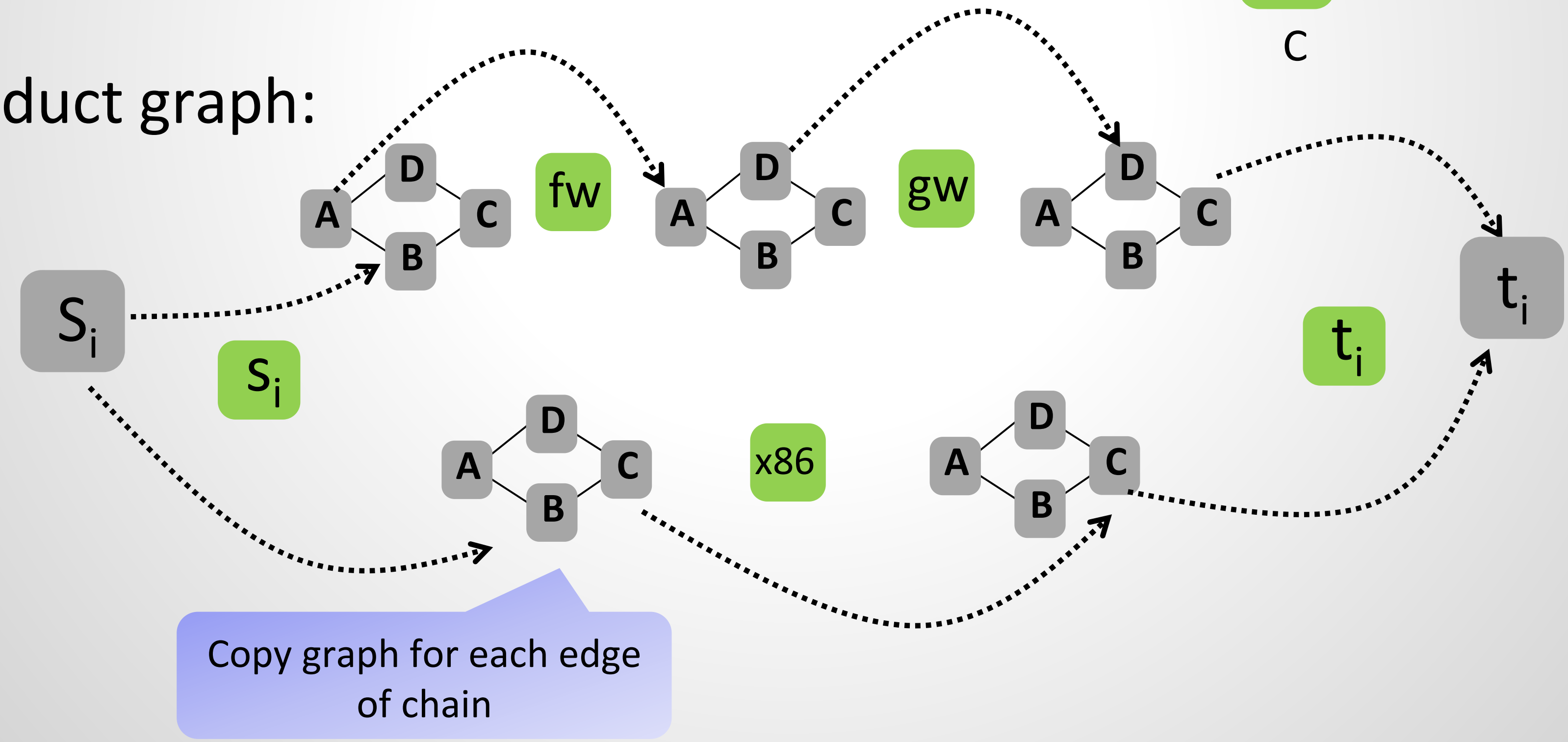
Substrate:



i^{th} request r_i:

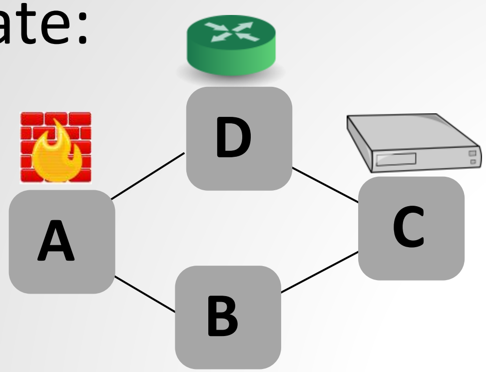


Product graph:

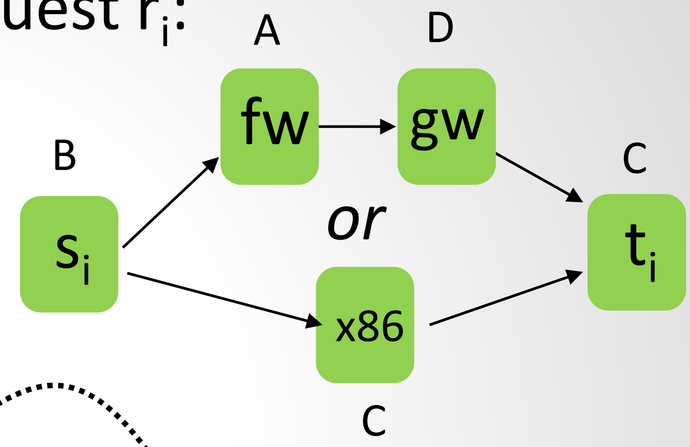


Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for “Fairly Simple” Requests*!

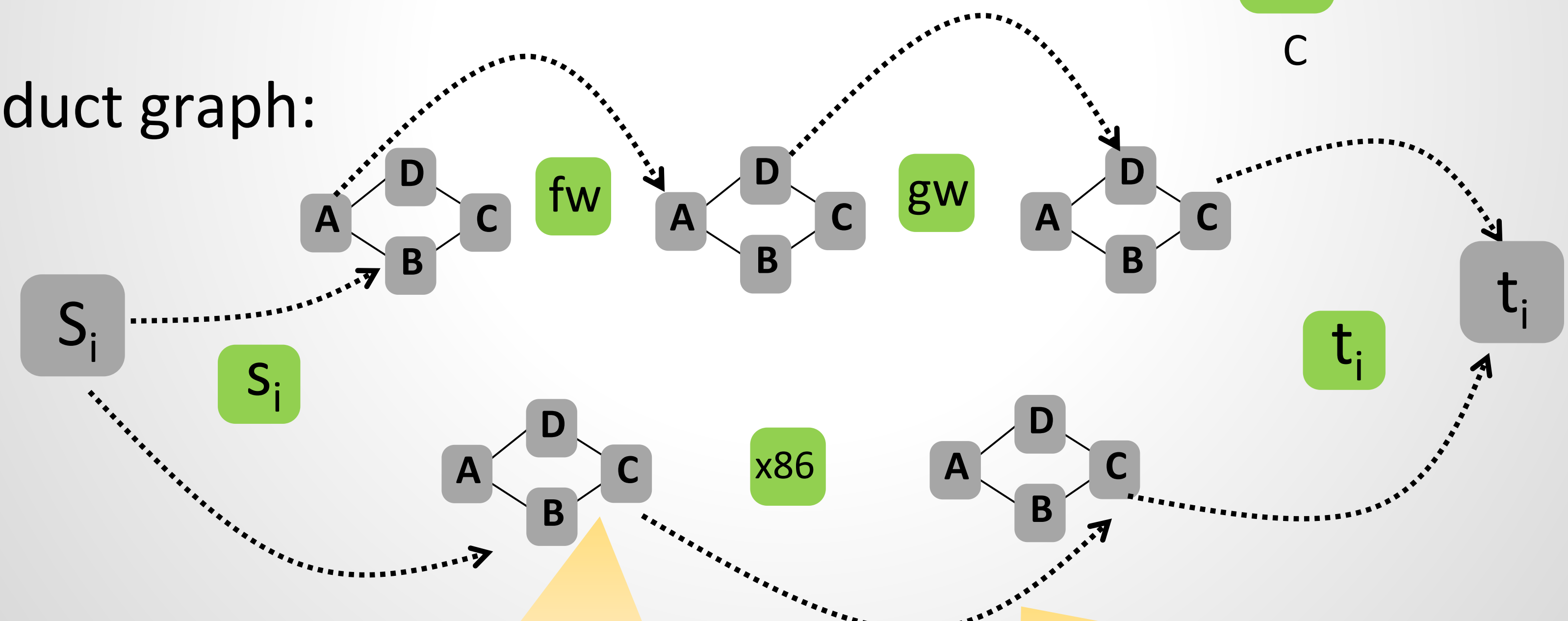
Substrate:



i^{th} request r_i:



Product graph:

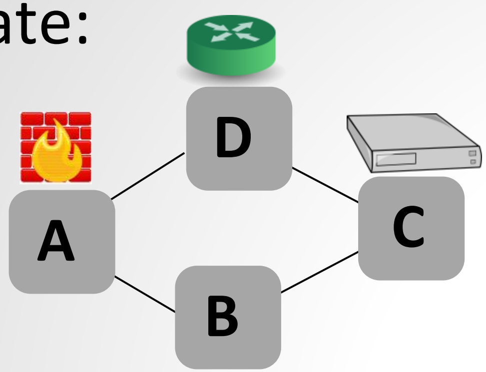


Routing edge: graph edge on same layer

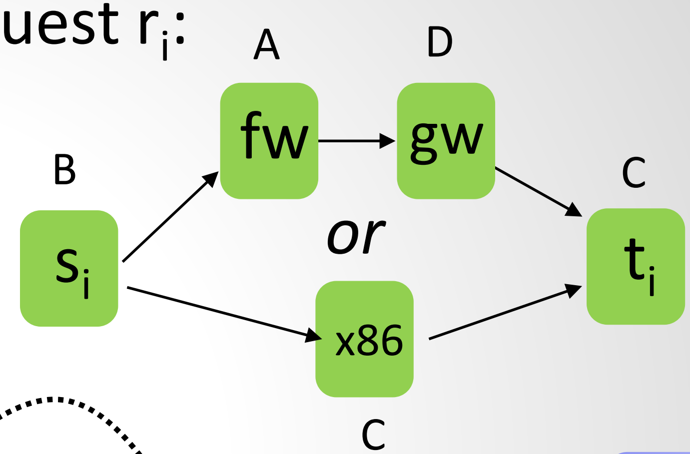
Processing edge: processing happens on C: connect C to C in next layer!

Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for "Fairly Simple" Requests*!

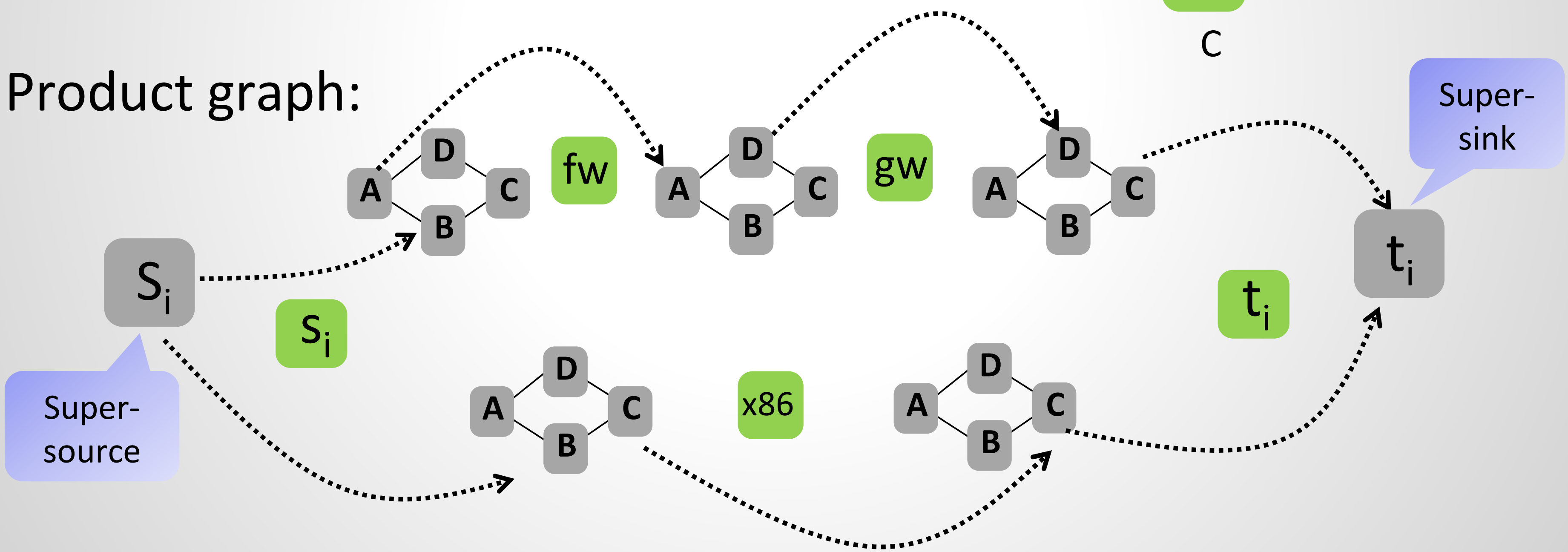
Substrate:



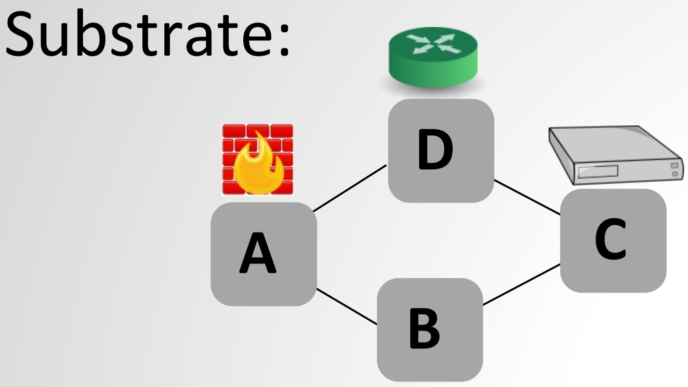
i^{th} request r_i:

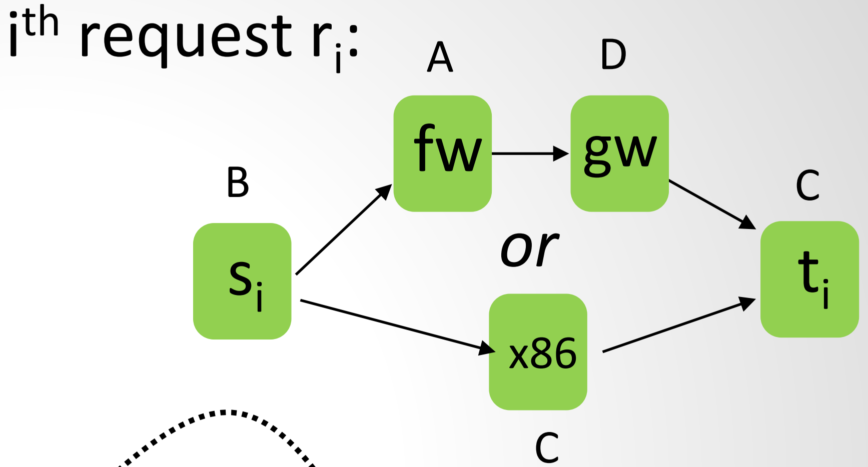


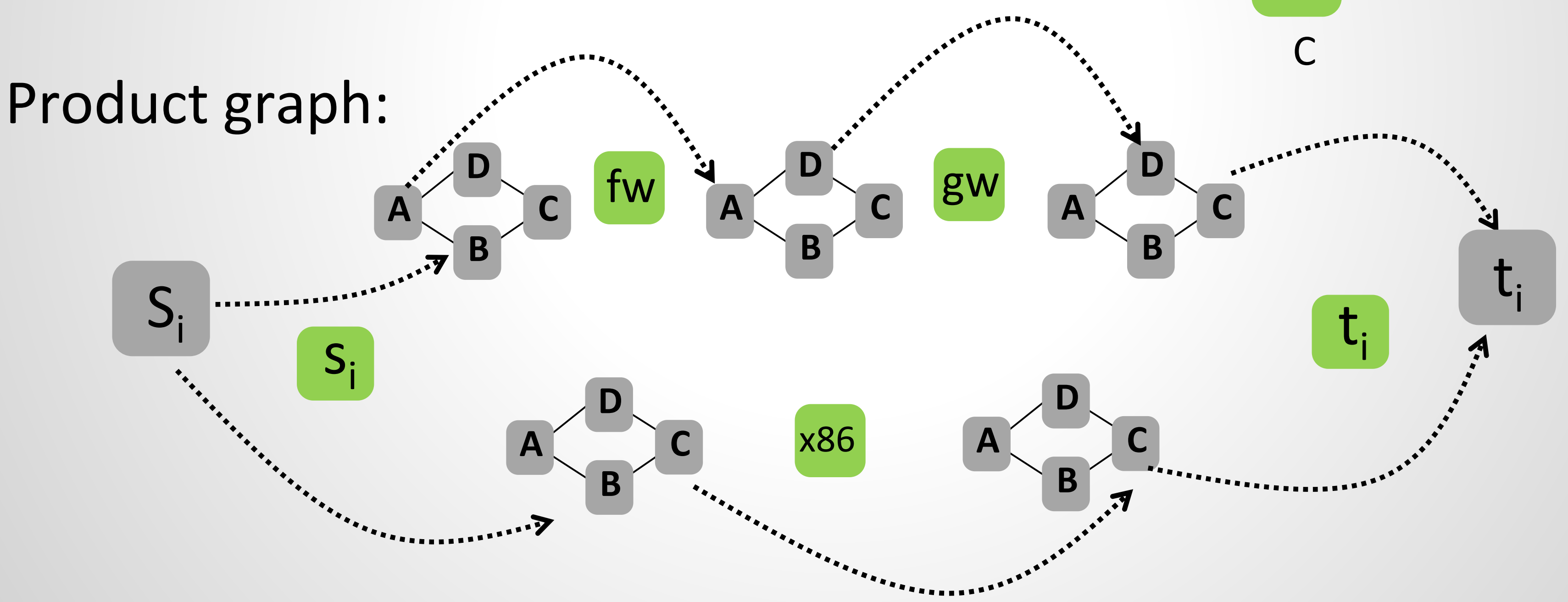
Product graph:



Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for "Fairly Simple" Requests*!

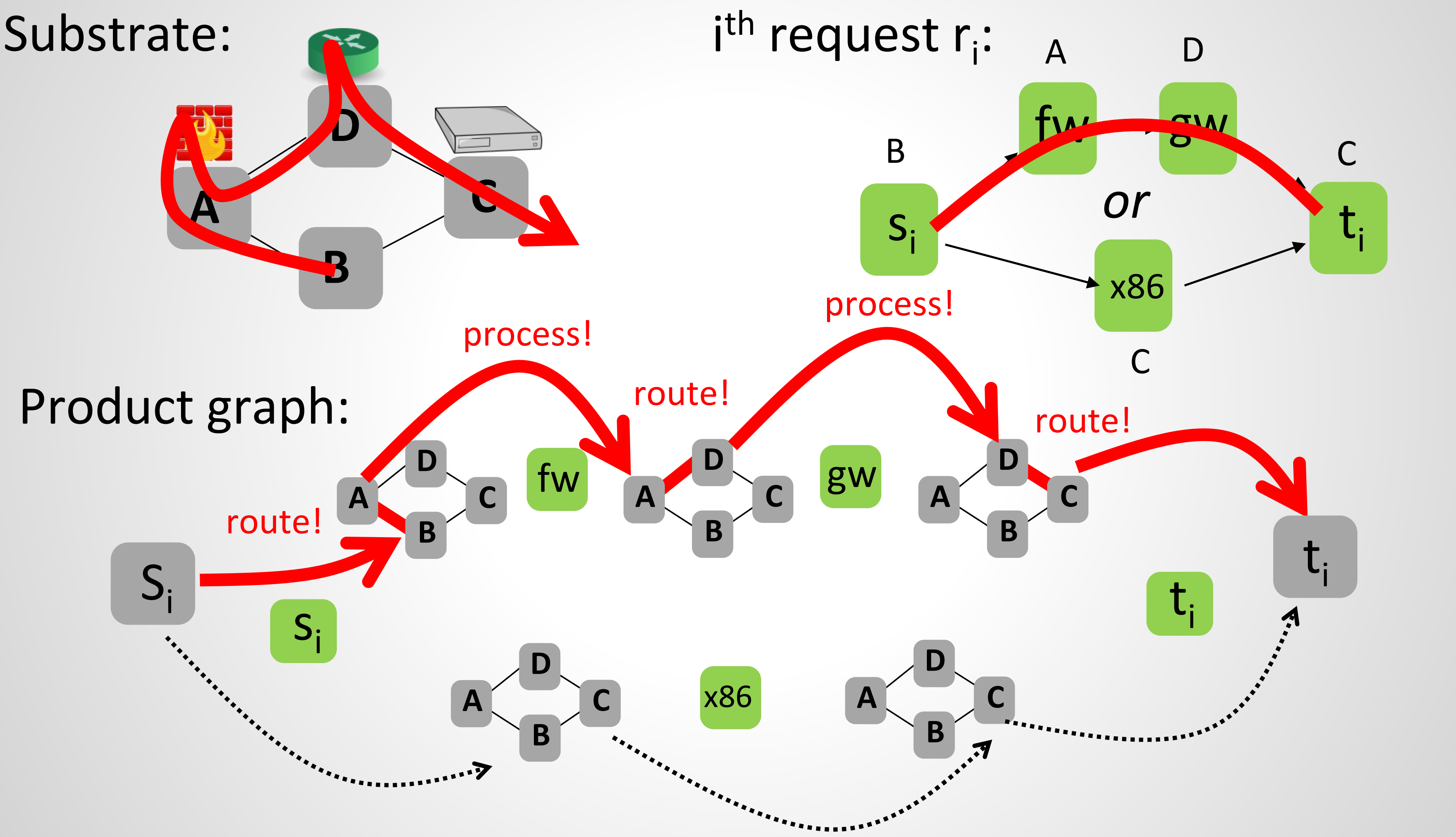






Any (s_i,t_i) flow presents a route of the request r_i!

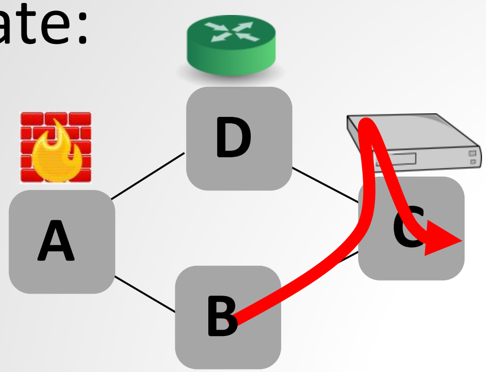
Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for "Fairly Simple" Requests*!



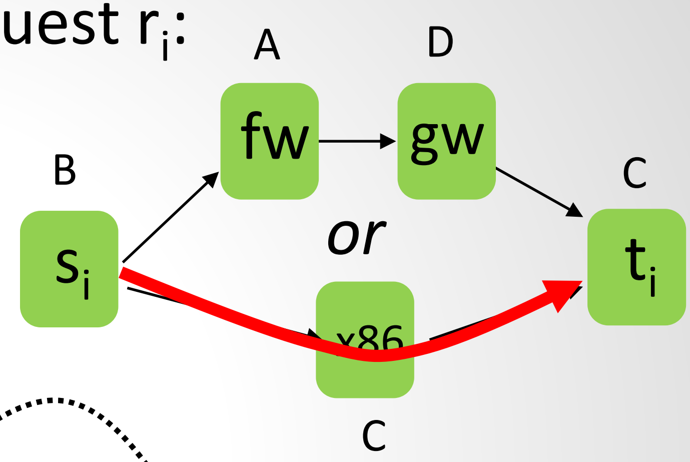
Any (s_i,t_i) flow presents a route of the request r_i!

Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for "Fairly Simple" Requests*!

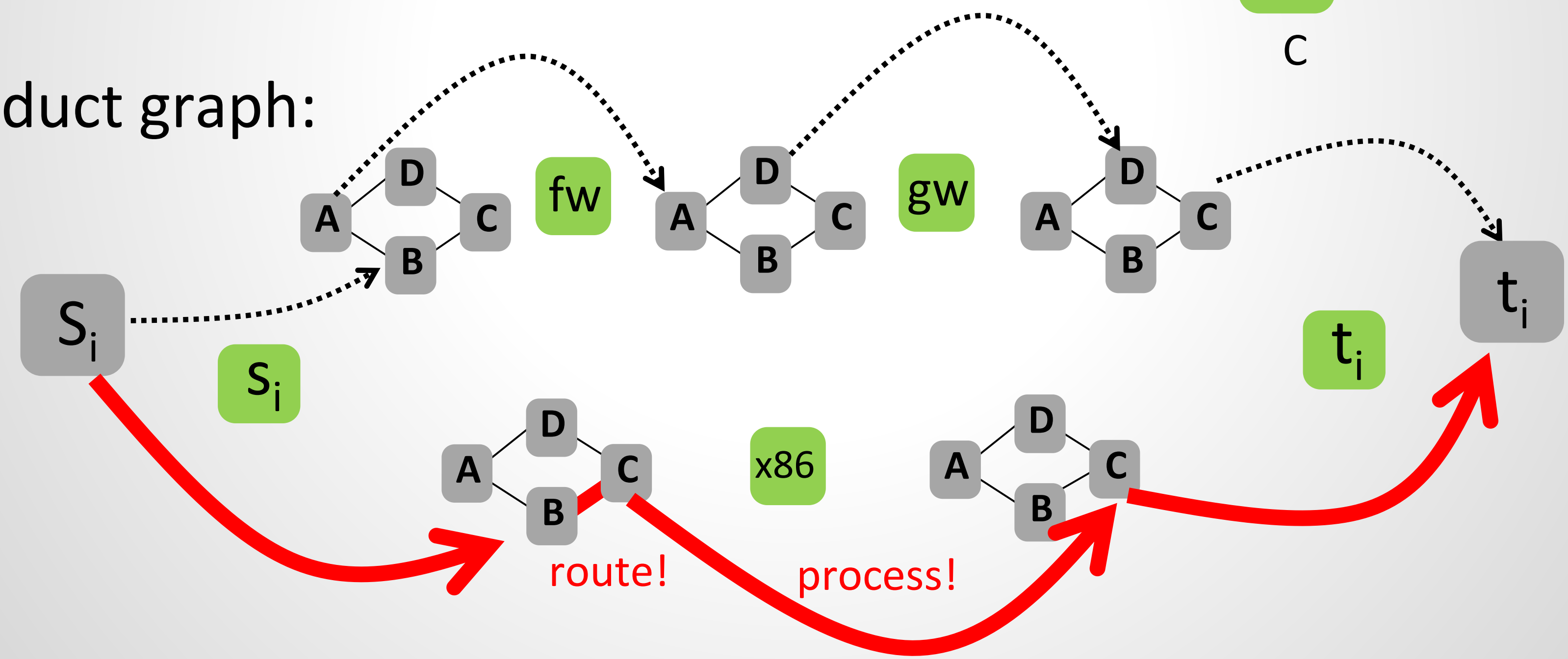
Substrate:



i^{th} request r_i:



Product graph:



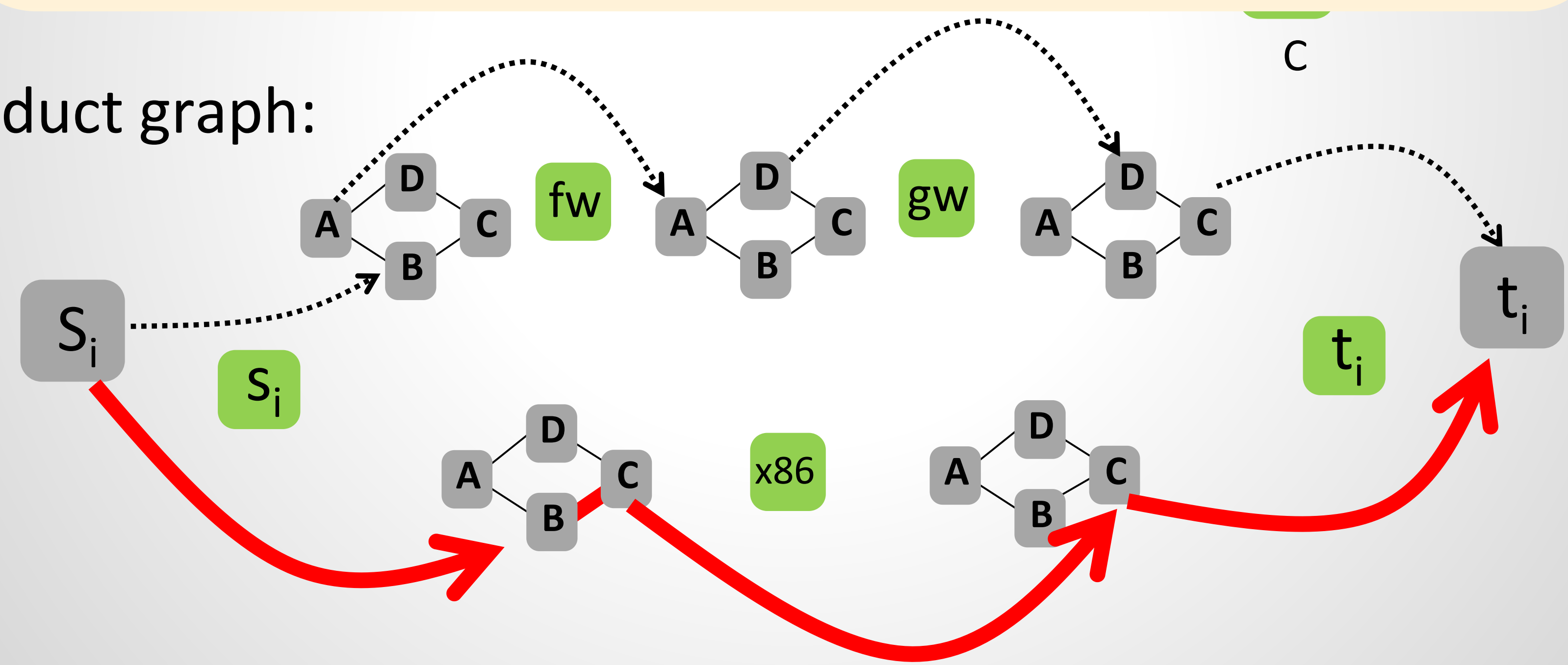
Any (s_i,t_i) flow presents a route of the request r_i!

Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for "Fairly Simple" Requests*!

Su

This problem can be solved using mincost unsplittable multi-commodity flow (approximation) algorithms (e.g., randomized rounding).

Product graph:



Any (s_i,t_i) flow presents a route of the request r_i!

Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for "Fairly Simple" Requests*!

This problem can be solved using mincost unsplittable multi-commodity flow (approximation) algorithms (e.g., randomized rounding).

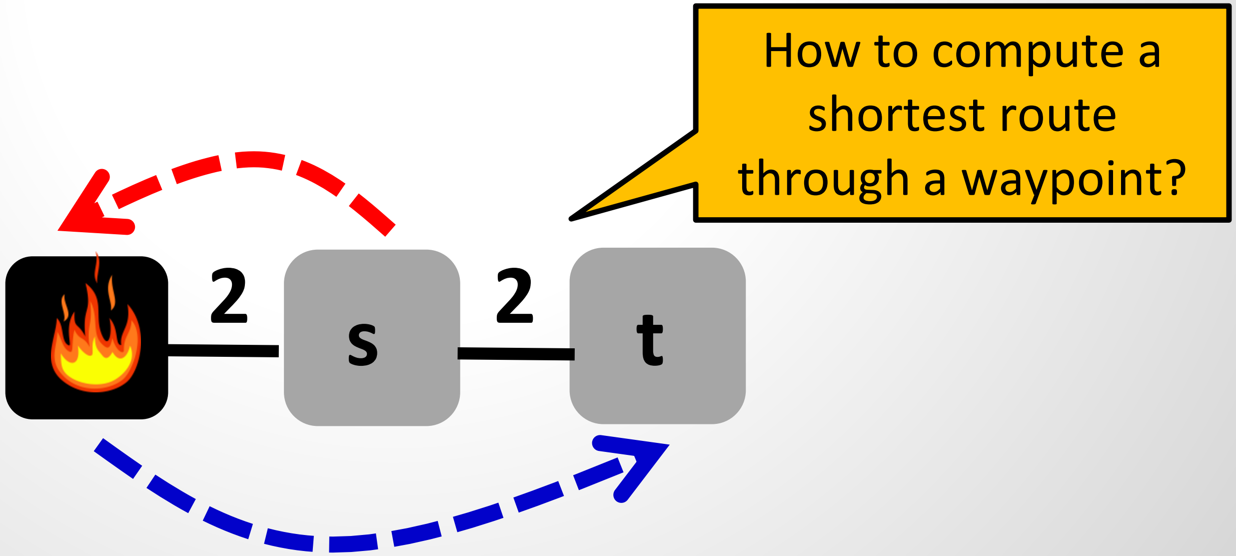
But note: cannot keep track of dependencies across stages (e.g., allocation on links or nodes): may yield oversubscription.

Any (s_i,t_i) flow presents a route of the request r_i!

Approximations Are Okay, But What About *Optimal* Embeddings?

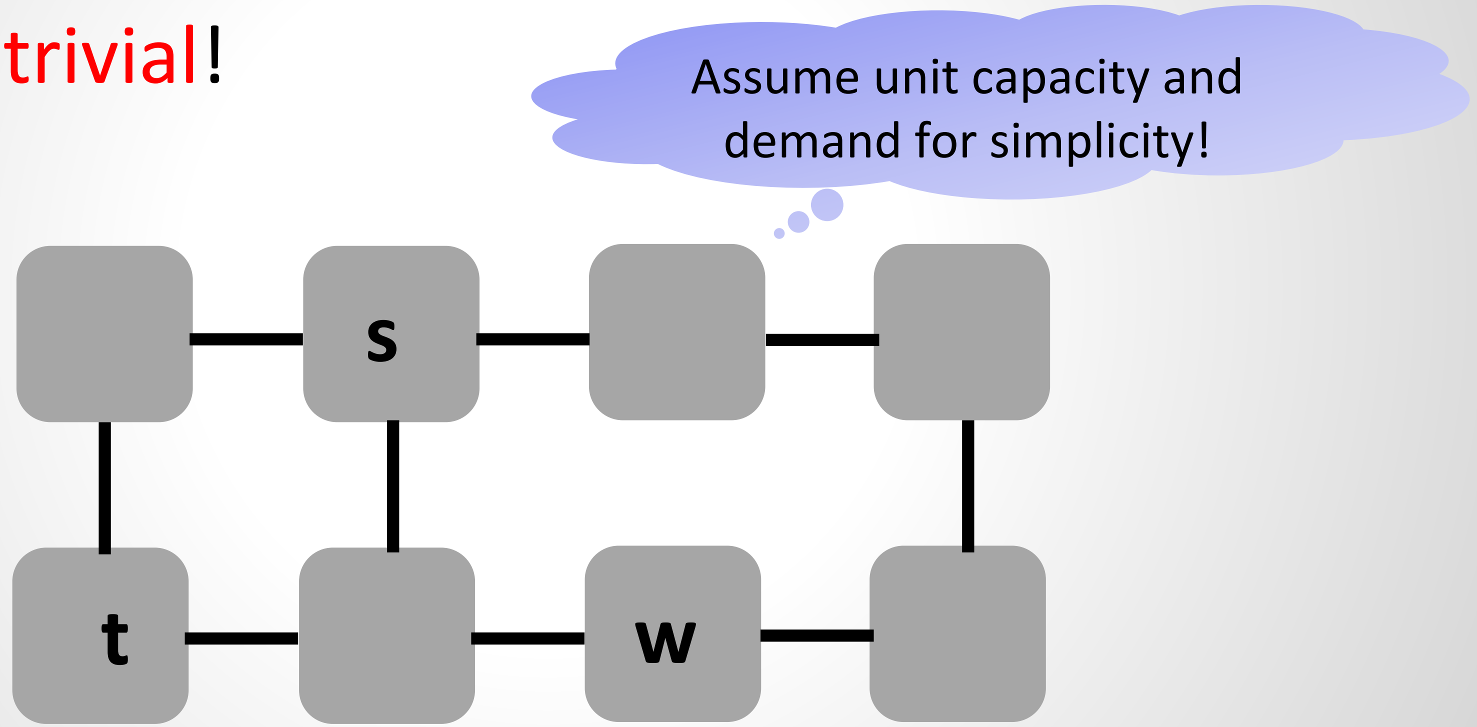
Novelty:

- Traditionally: routes form simple paths (e.g., shortest paths)
- Now: routing through middleboxes may require more general paths, *with loops*: a walk



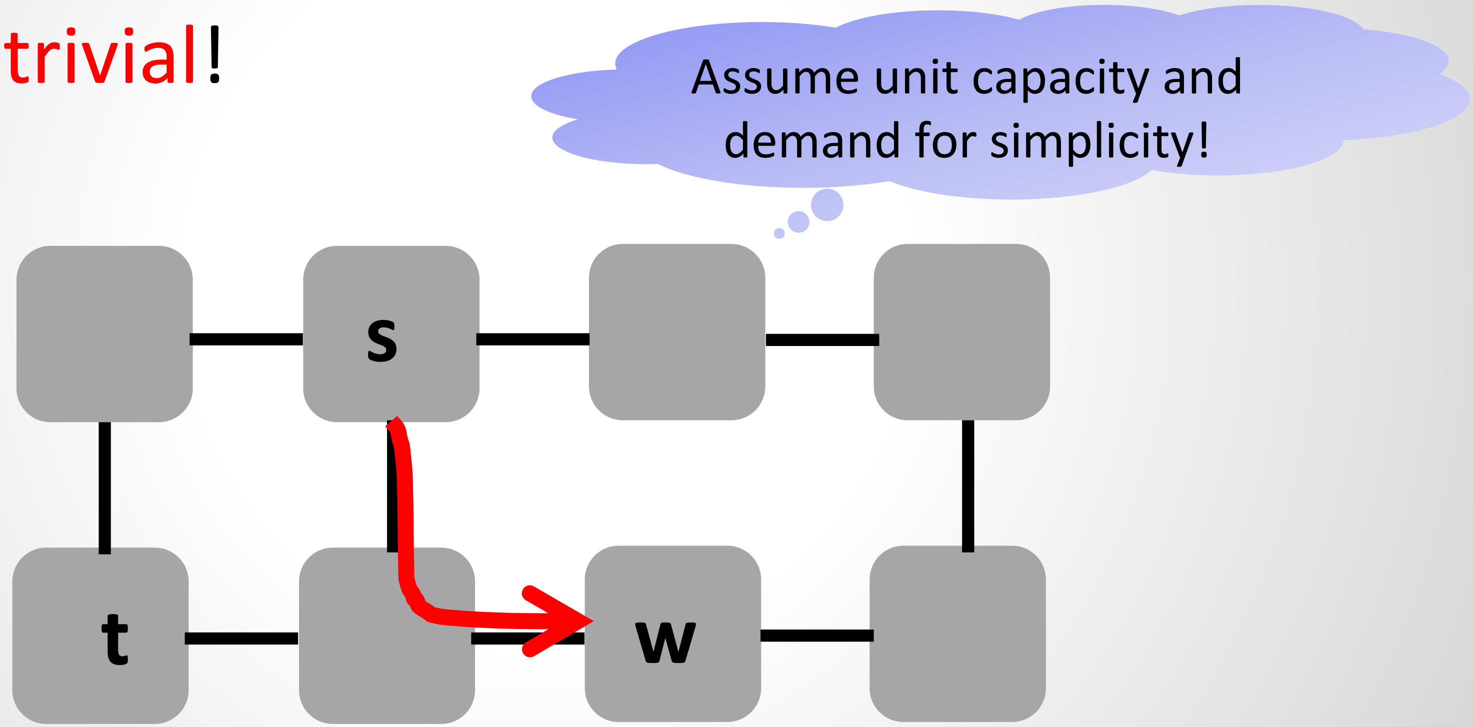
Comuting A Shortest Walk Through A *Single* Given Waypoint is Non-Trivial!

- Computing shortest routes through waypoints is non-trivial!



Comuting A Shortest Walk Through A *Single* Given Waypoint is Non-Trivial!

- Computing shortest routes through waypoints is non-trivial!

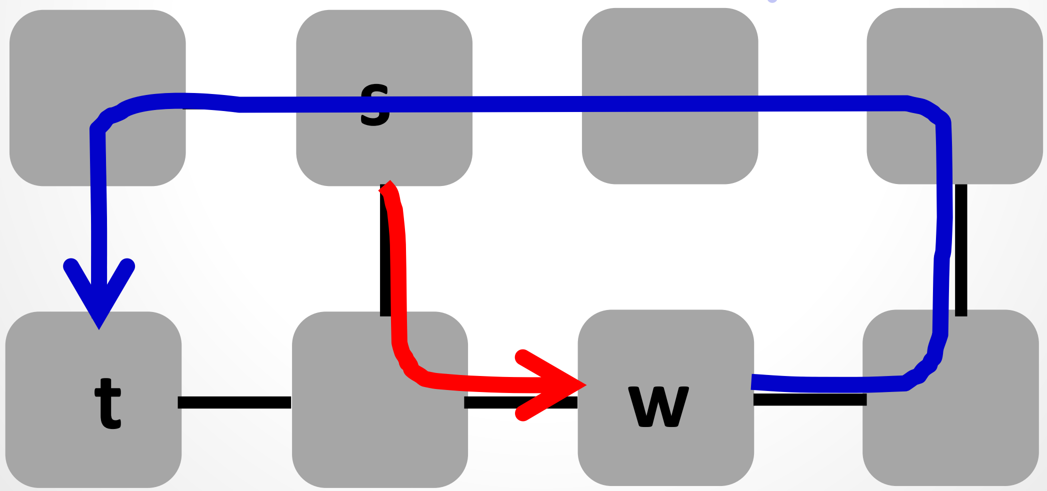


Greedy fails: **choose shortest path from s to w…**

Comuting A Shortest Walk Through A *Single* Given Waypoint is Non-Trivial!

- Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!

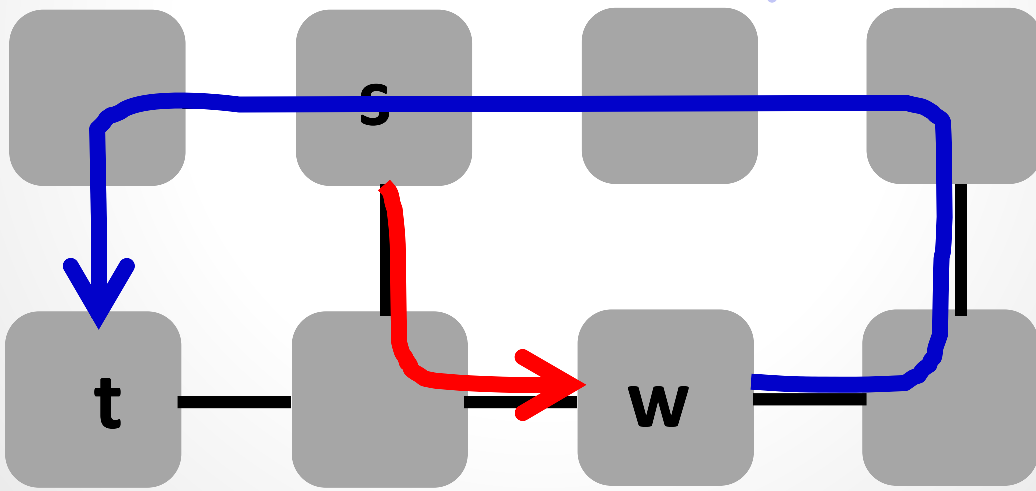


Greedy fails: **… now need long path from w to t**

Comuting A Shortest Walk Through A *Single* Given Waypoint is Non-Trivial!

- Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!

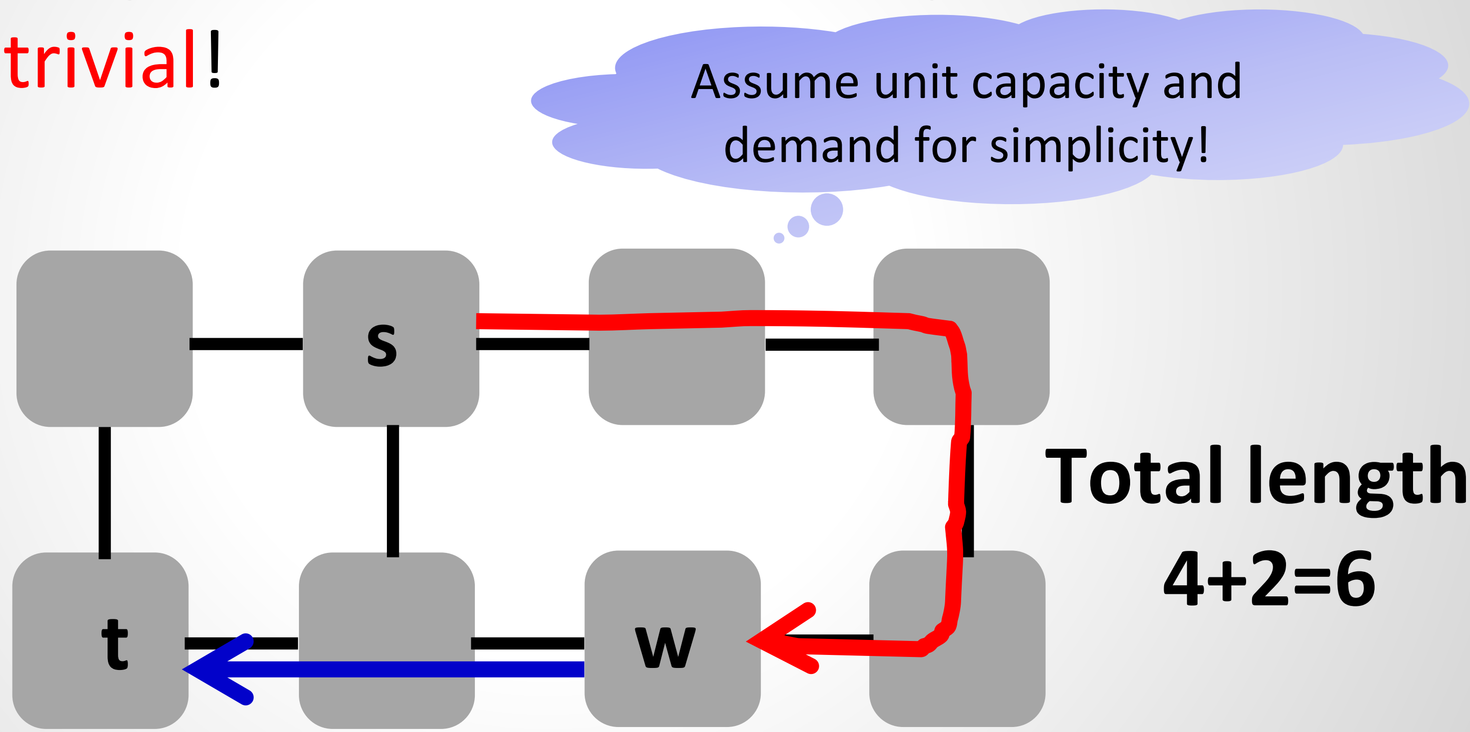


Total length: 2+6=8

Greedy fails: **… now need long path from w to t**

Comuting A Shortest Walk Through A *Single* Given Waypoint is Non-Trivial!

- Computing shortest routes through waypoints is non-trivial!

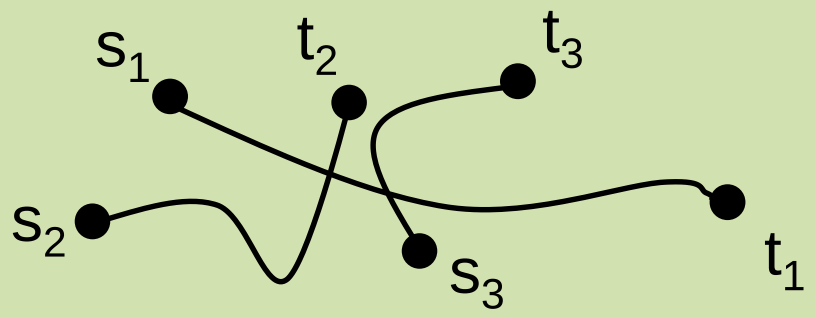


Total length: 4+2=6

A *better solution*: *jointly* optimize the two segments!

Relationship to *Shortest Disjoint Paths*

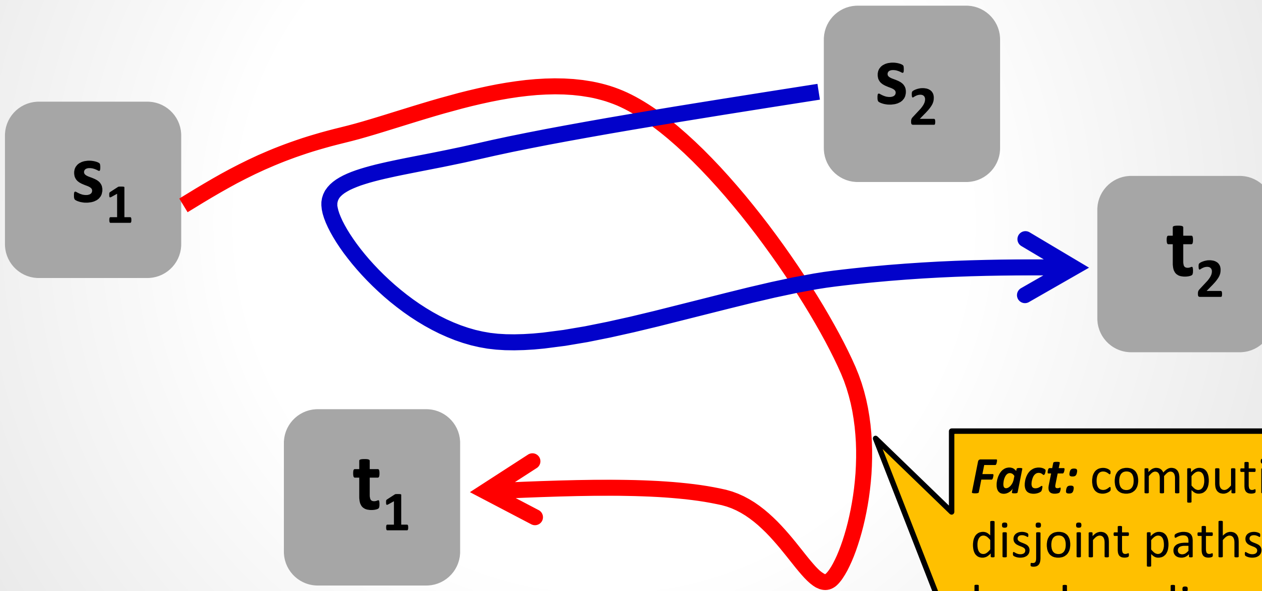
If capacities are 1, segments need to be edge-disjoint: A **disjoint paths problem**



- A well-known combinatorial problem!
- NP-hard on directed networks
- Feasibility in P on undirected networks for small (constant) number of flows
- Polytime randomized algorithm for 2 disjoint paths (recent result!)

NP-hard on *Directed* Networks: Reduction from Disjoint Paths Problem

Reduction: *From* joint shortest paths $(s_1,t_1),(s_2,t_2)$ *to* shortest walk (s,w,t) problem

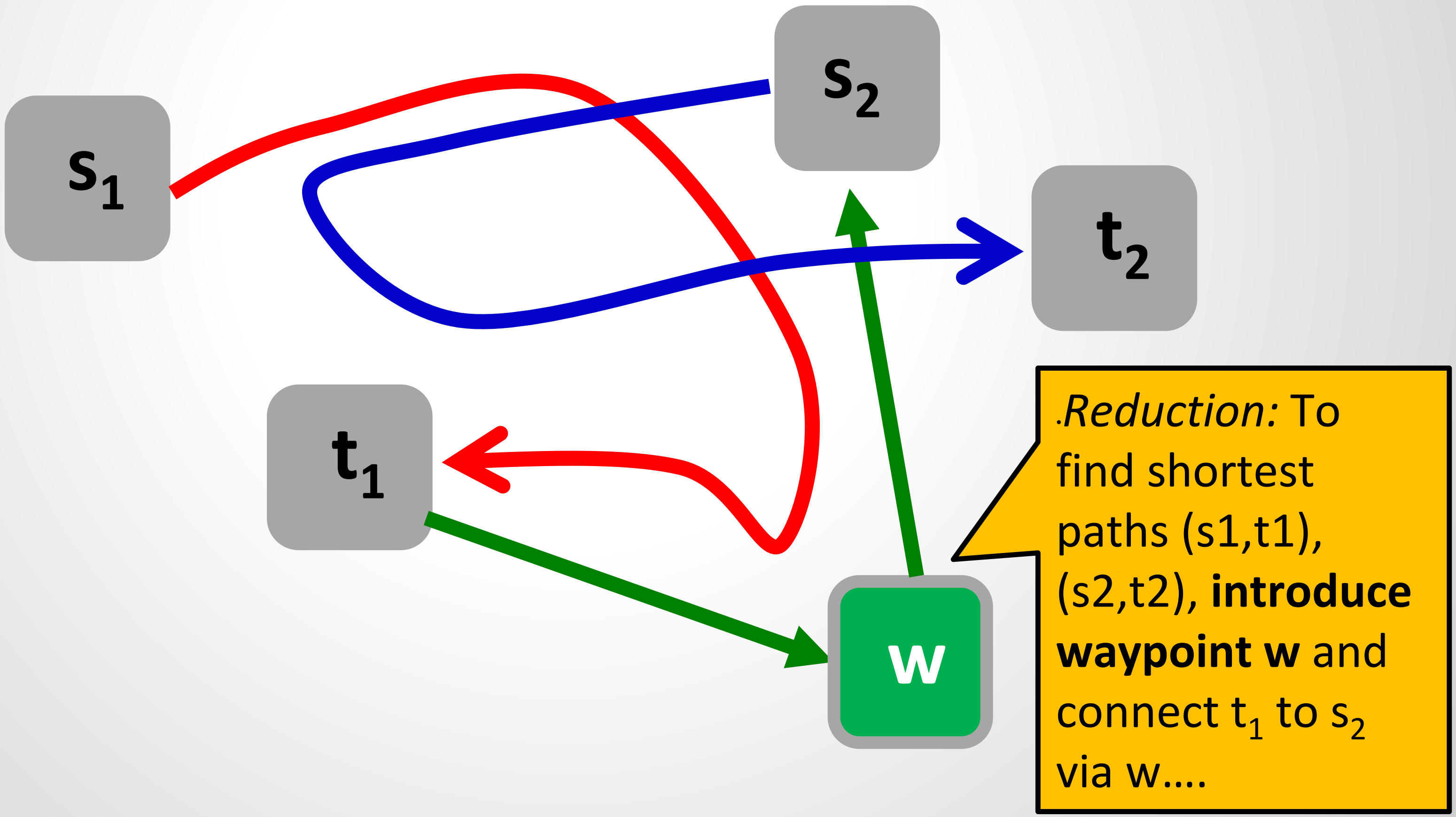


Fact: computing 2-disjoint paths (2DP) is NP-hard on directed graphs.

We show: If waypoint routing was in P, we could solve 2DP fast. ***Contradiction!***

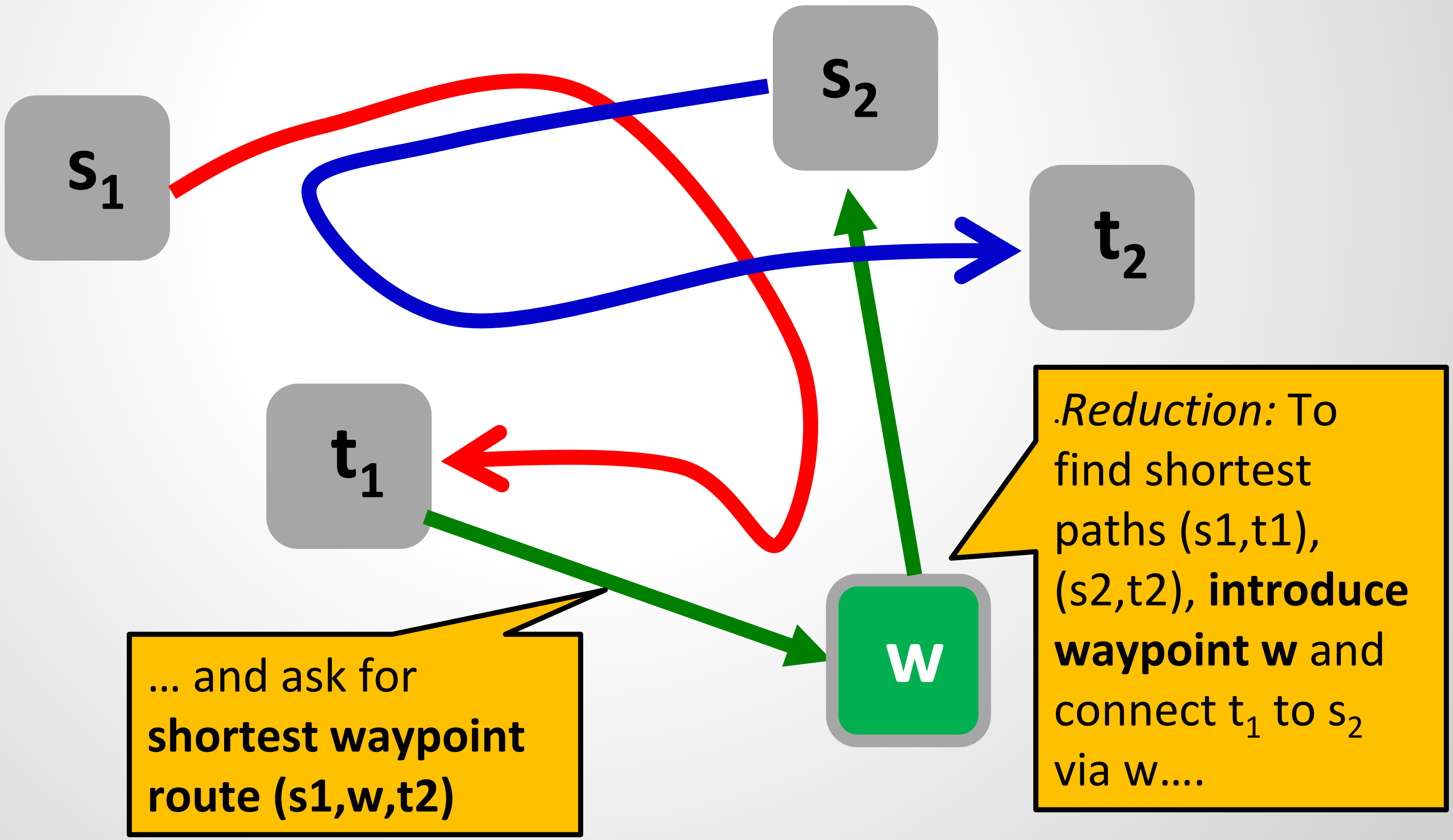
NP-hard on *Directed* Networks: Reduction from Disjoint Paths Problem

Reduction: *From* joint shortest paths $(s_1,t_1),(s_2,t_2)$ *to* shortest walk (s,w,t) problem



NP-hard on *Directed* Networks: Reduction from Disjoint Paths Problem

Reduction: *From* joint shortest paths $(s_1,t_1),(s_2,t_2)$ *to* shortest walk (s,w,t) problem



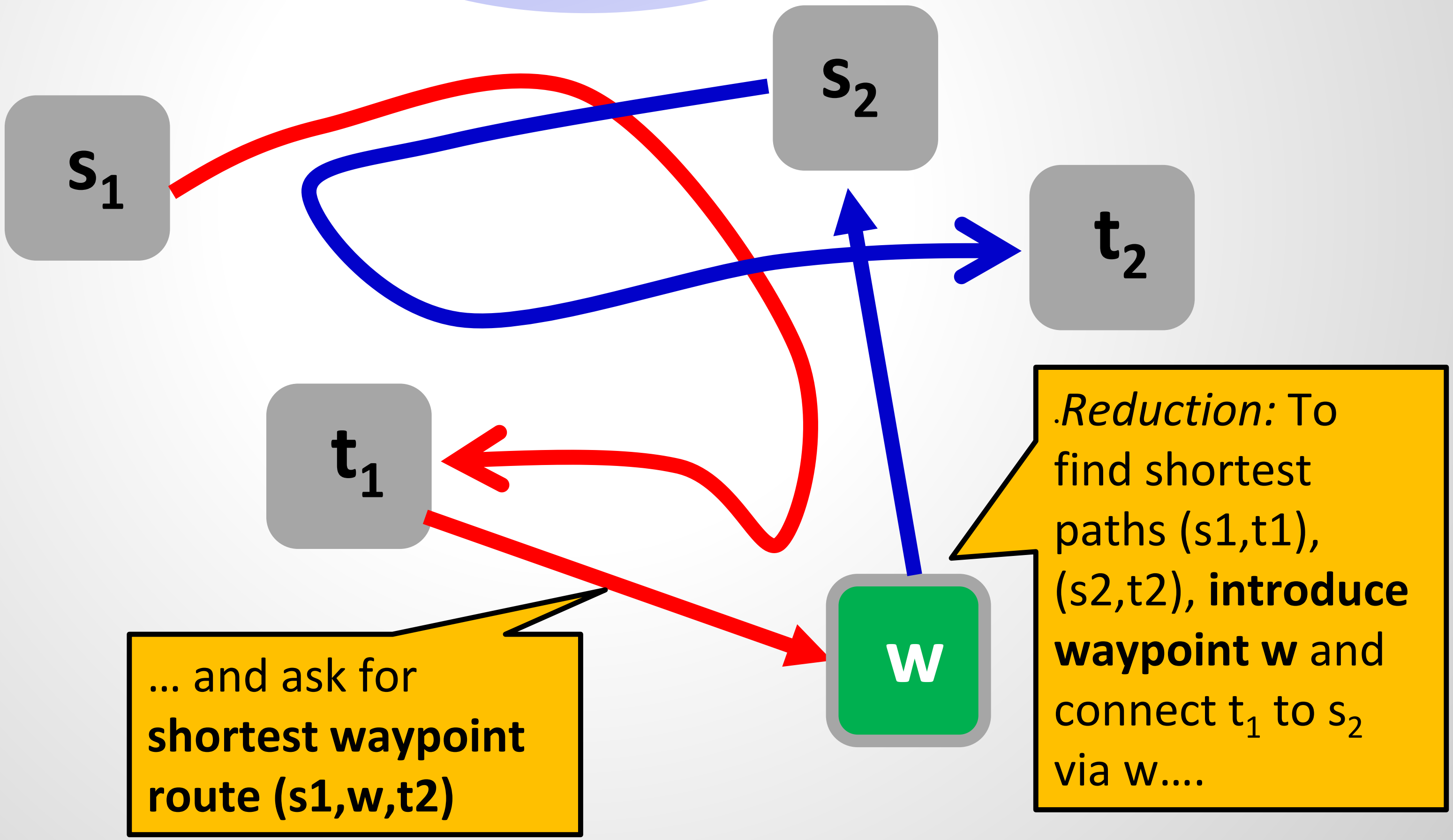
NP-hard on Direct

F

Reduc

to shortest w

The walk (s_1,w,t_2) walk defines a (s_1,t_1) and a (s_2,t_2) path pair before/after the waypoint! Solves original problem: **Contradiction!**



What about waypoint routes on *undirected* networks?

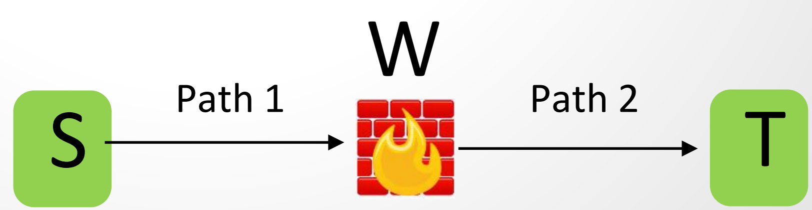
What about waypoint routes on *undirected* networks? (2)

Idea: Reduce it ***to*** disjoint paths problem!

- For a single waypoint, can even compute ***shortest route (walk)***!
  - Recall: there is a randomized polytime algorithm for 2 disjoint paths

Step 1: replace weights with parallel links

Step 2: compute 2 disjoint paths (A,W) and (W,B)



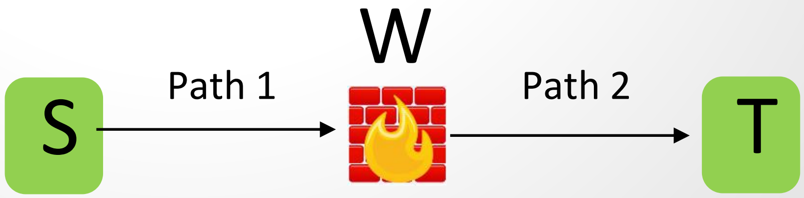
What about waypoint routes on *undirected* networks? (2)

Idea: Reduce it ***to*** disjoint paths problem!

- For a single waypoint, can even compute ***shortest route (walk)***!
  - Recall: there is a randomized polytime algorithm for 2 disjoint paths

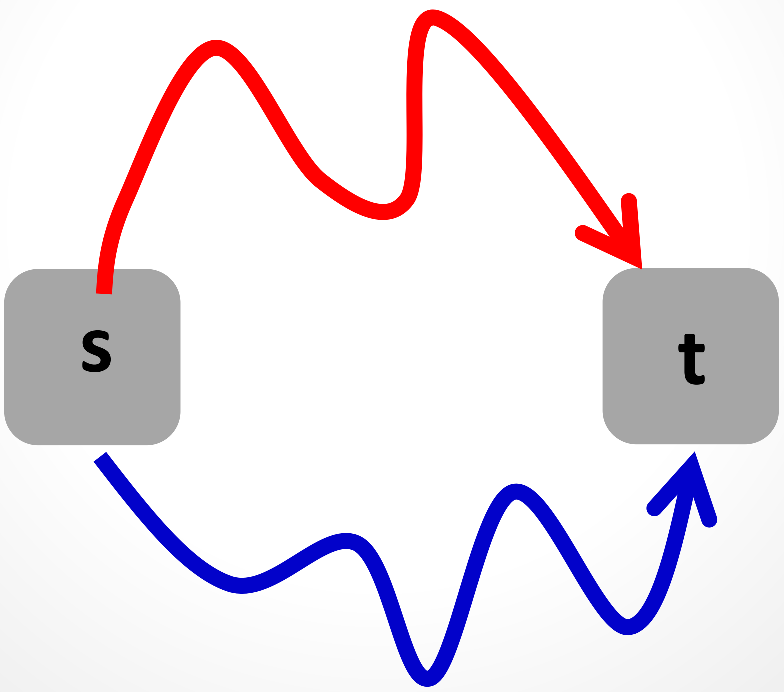
Good news: For a single waypoint, ***shortest*** paths can be computed even ***faster***!

Step 2: compute 2 disjoint paths (A,W) and (W,B)



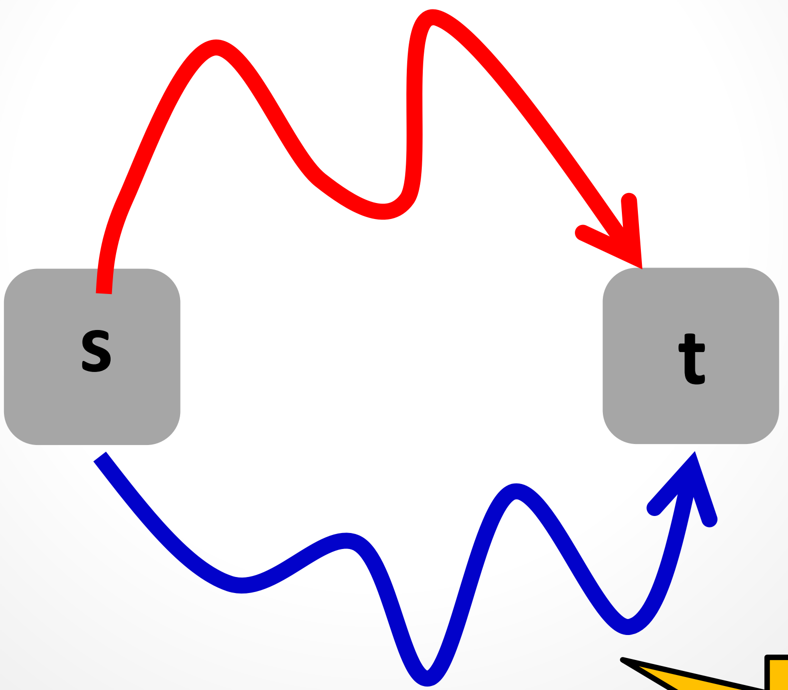
Walking Through a Waypoint on Steroids: Suurballe's Algorithm

- Suurballe's algorithm: finds two (edge-)disjoint shortest paths between same endpoints:



Walking Through a Waypoint on Steroids: Suurballe's Algorithm

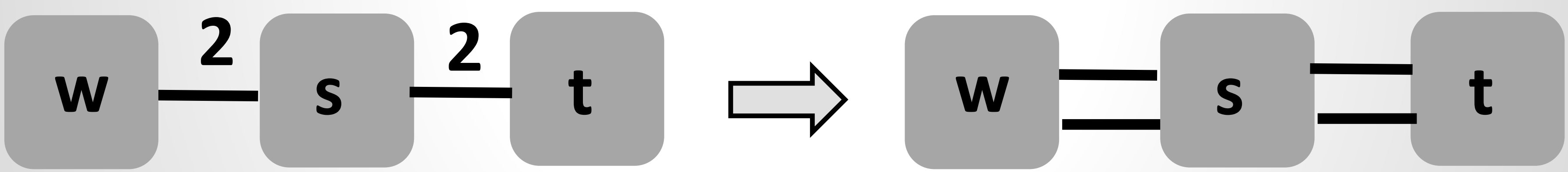
- Suurballe's algorithm: finds two (edge-)disjoint shortest paths between same endpoints:



.How to compute a shortest (s,w,t) route with this algorithm??

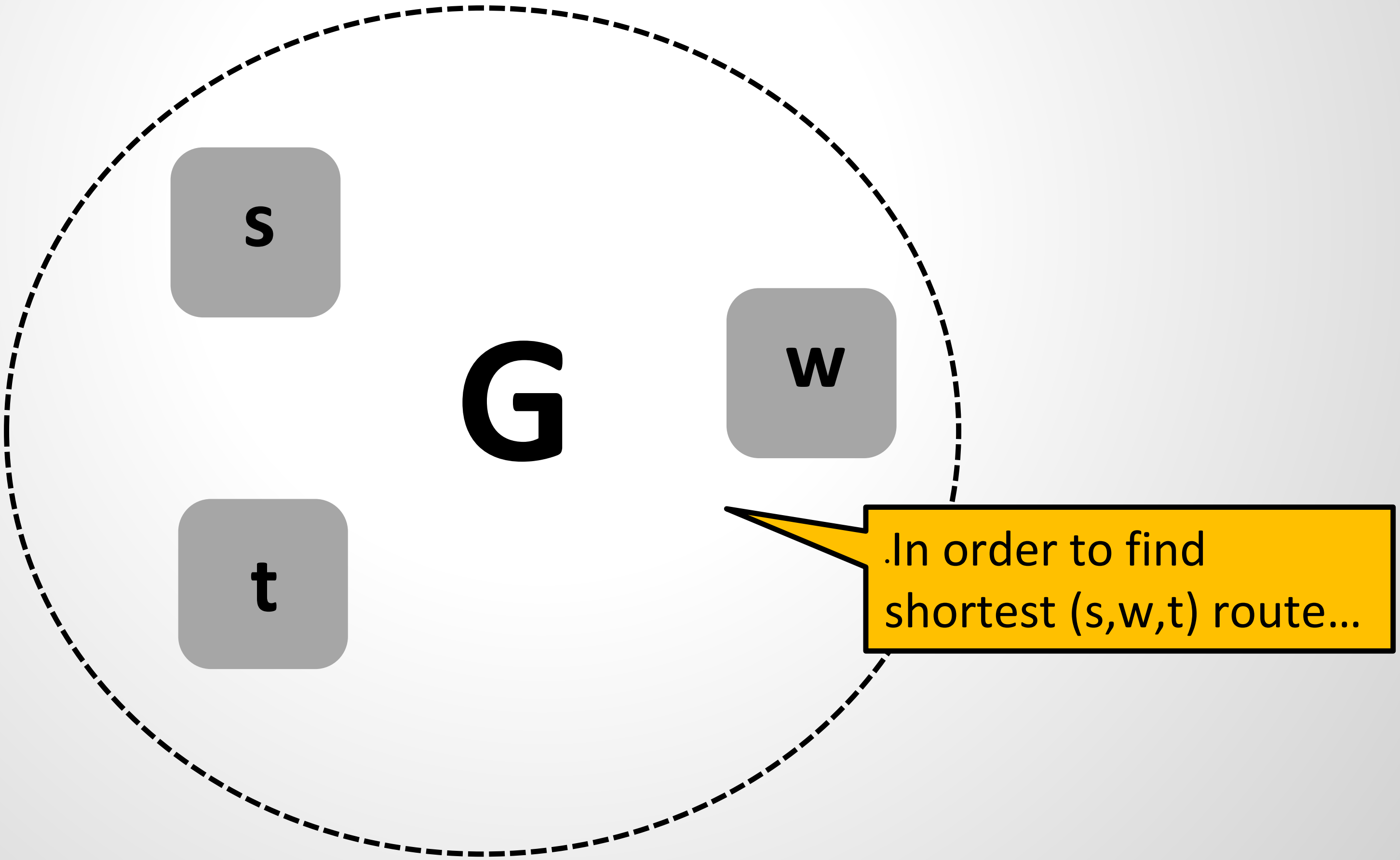
Walking Through a Waypoint on Steroids: Reduction to Suurballe's Algorithm

- **Step 1:** replace capacities with parallel edges: paths will become edge-disjoint



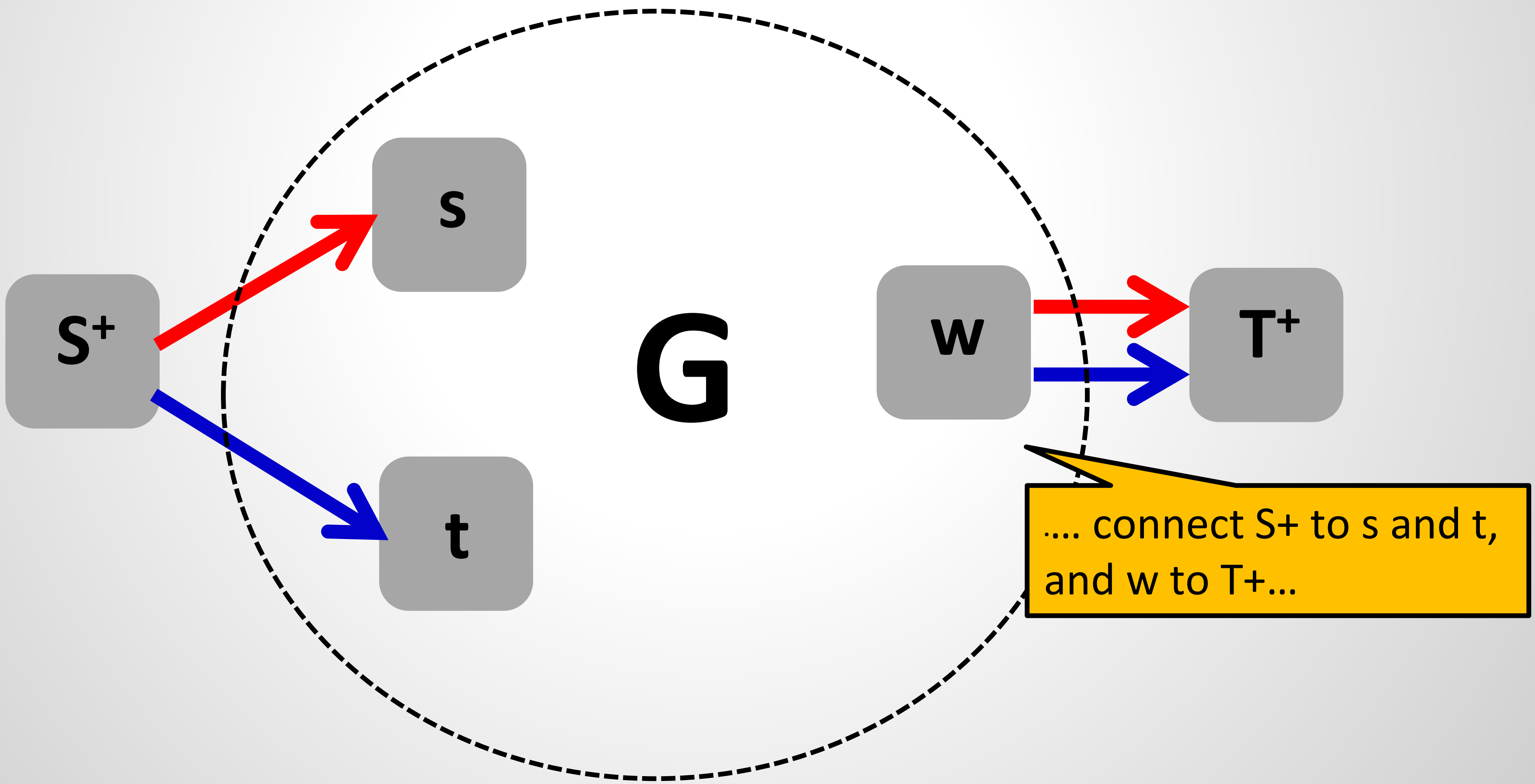
Walking Through a Waypoint on Steroids: Reduction to Suurballe's Algorithm

- **Step 2:** Reduction to Suurballe's algorithm:



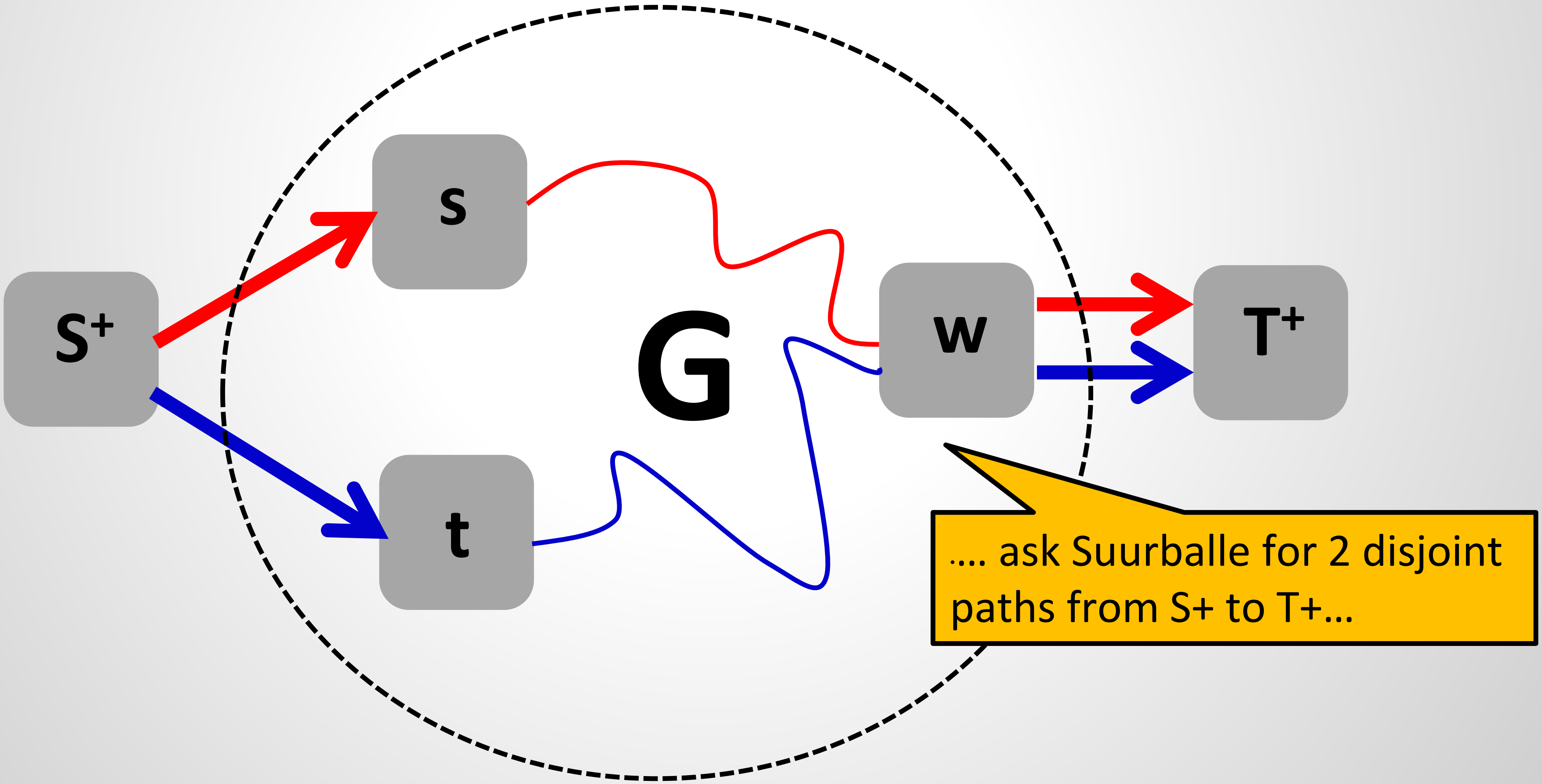
Walking Through a Waypoint on Steroids: Reduction to Suurballe's Algorithm

- **Step 2:** Reduction to Suurballe's algorithm:



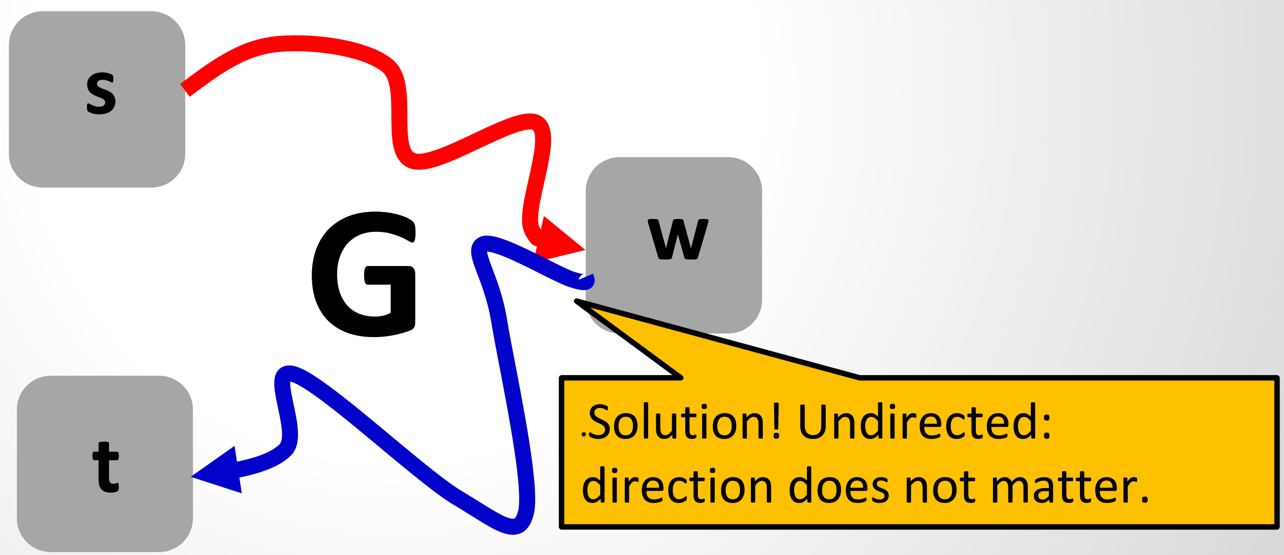
Walking Through a Waypoint on Steroids: Reduction to Suurballe's Algorithm

- **Step 2:** Reduction to Suurballe's algorithm:



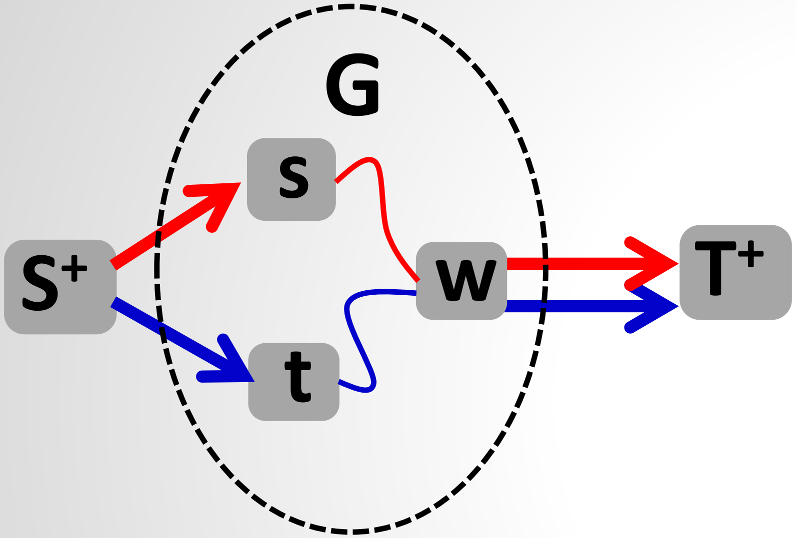
Walking Through a Waypoint on Steroids: Reduction to Suurballe's Algorithm

- **Step 2:** Reduction to Suurballe's algorithm:



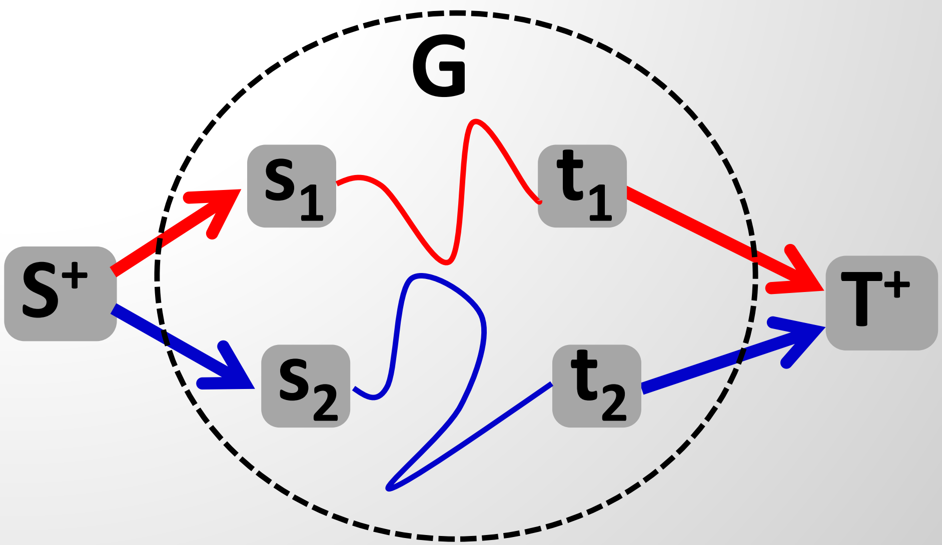
Wait A Moment…!?

Can we not use Suurballe as well to solve 2 disjoint paths?



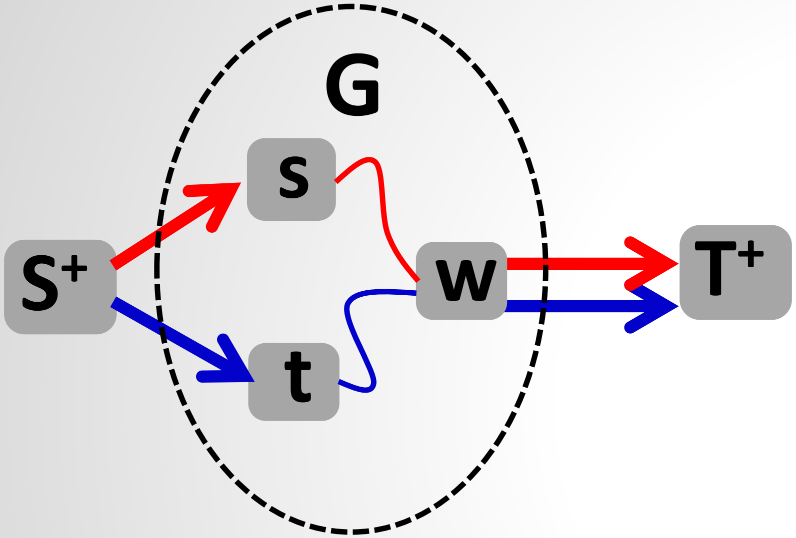
Reduction
Waypoint Routing $\Rightarrow$ Suurballe

Reduction
2 Disjoint Paths $\Rightarrow$ Suurballe



Wait A Moment…!?

No! Solves a much easier problem: 2 routes from $\{s_1,s_2\}$ to $\{t_1,t_2\}$.

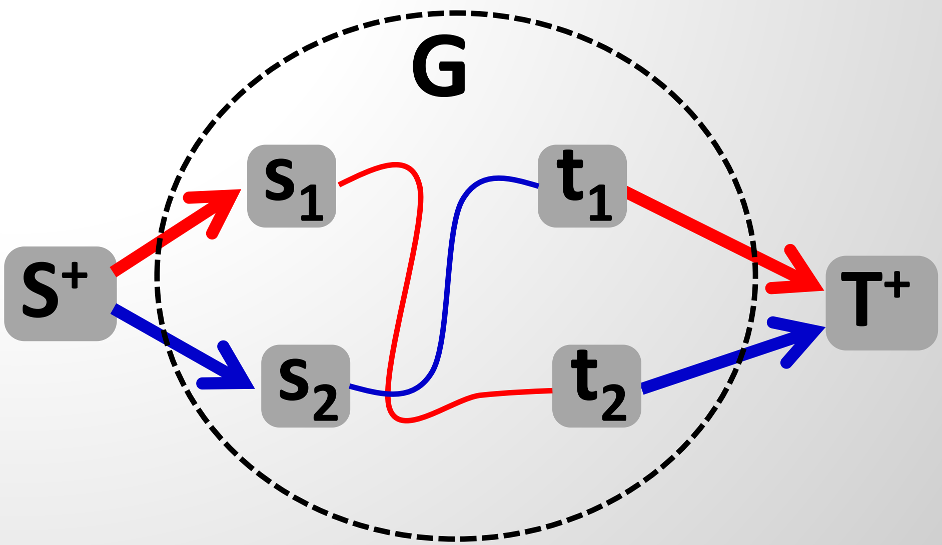


Reduction

Waypoint Routing $\Rightarrow$ Suurballe

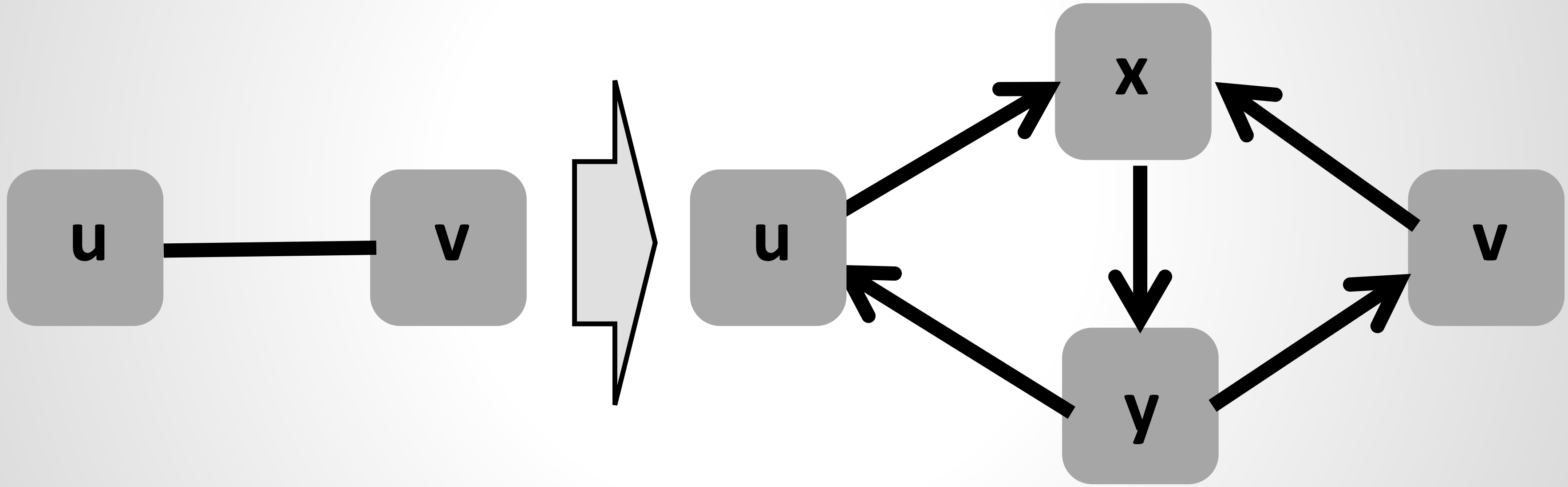
Reduction

2 Disjoint Paths $\Rightarrow$ Suurballe



Remarks: Under the rug…

- **Remark 1:** Suurballe is actually for directed substrate graphs, so need gadget to transform problem in right form:



- **Remark 2:** Suurballe: for vertex disjoint
  - Suurballe & Tarjan: edge disjoint

Further Reading

An Approximation Algorithm for Path Computation and Function Placement in SDNs
Guy Even, Matthias Rost, and Stefan Schmid.
23rd International Colloquium on Structural Information and Communication Complexity (**SIROCCO**), Helsinki, Finland, July 2016.

Charting the Complexity Landscape of Waypoint Routing
Saeed Akhoondian Amiri, Klaus-Tycho Foerster, Riko Jacob, and Stefan Schmid. ArXiv Technical Report, May 2017.

Online Admission Control and Embedding of Service Chains
Tamás Lukovszki and Stefan Schmid.
SIROCCO, July 2015.

Walking Through Waypoints
Saeed Akhoondian Amiri, Klaus-Tycho Foerster, and Stefan Schmid. ArXiv Technical Report, August 2017.

Competitive and Deterministic Embeddings of Virtual Networks
Guy Even, Moti Medina, Gregor Schaffrath, and Stefan Schmid.
Journal Theoretical Computer Science (**TCS**), Elsevier, 2013.

You: Great, I can embed service chains at low resource cost and providing minimal bandwidth guarantees!

You: Great, I can embed service chains at low resource cost and providing minimal bandwidth guarantees!

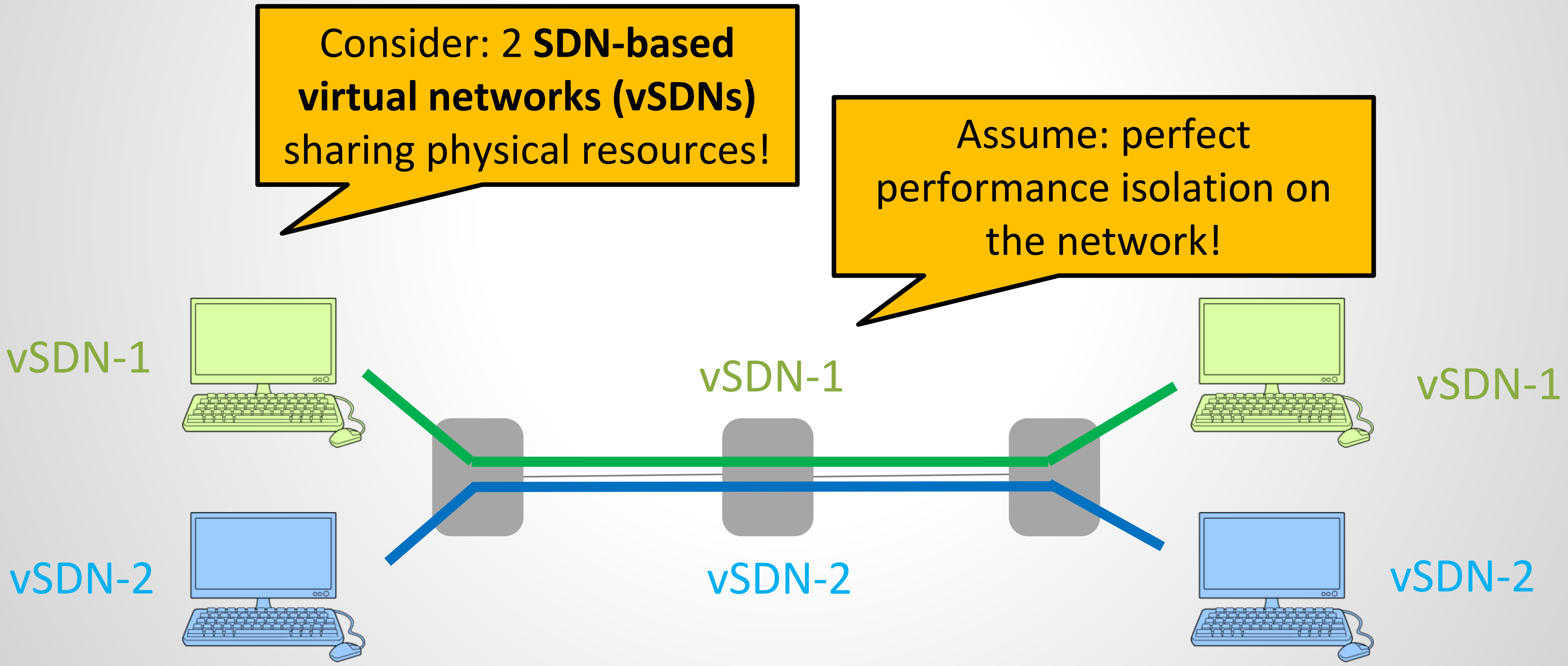
Boss: So can I promise our customers a predictable performance?

You: Great, I can embed service chains at low resource cost and providing minimal bandwidth guarantees!

Boss: So can I promise our customers a predictable performance?

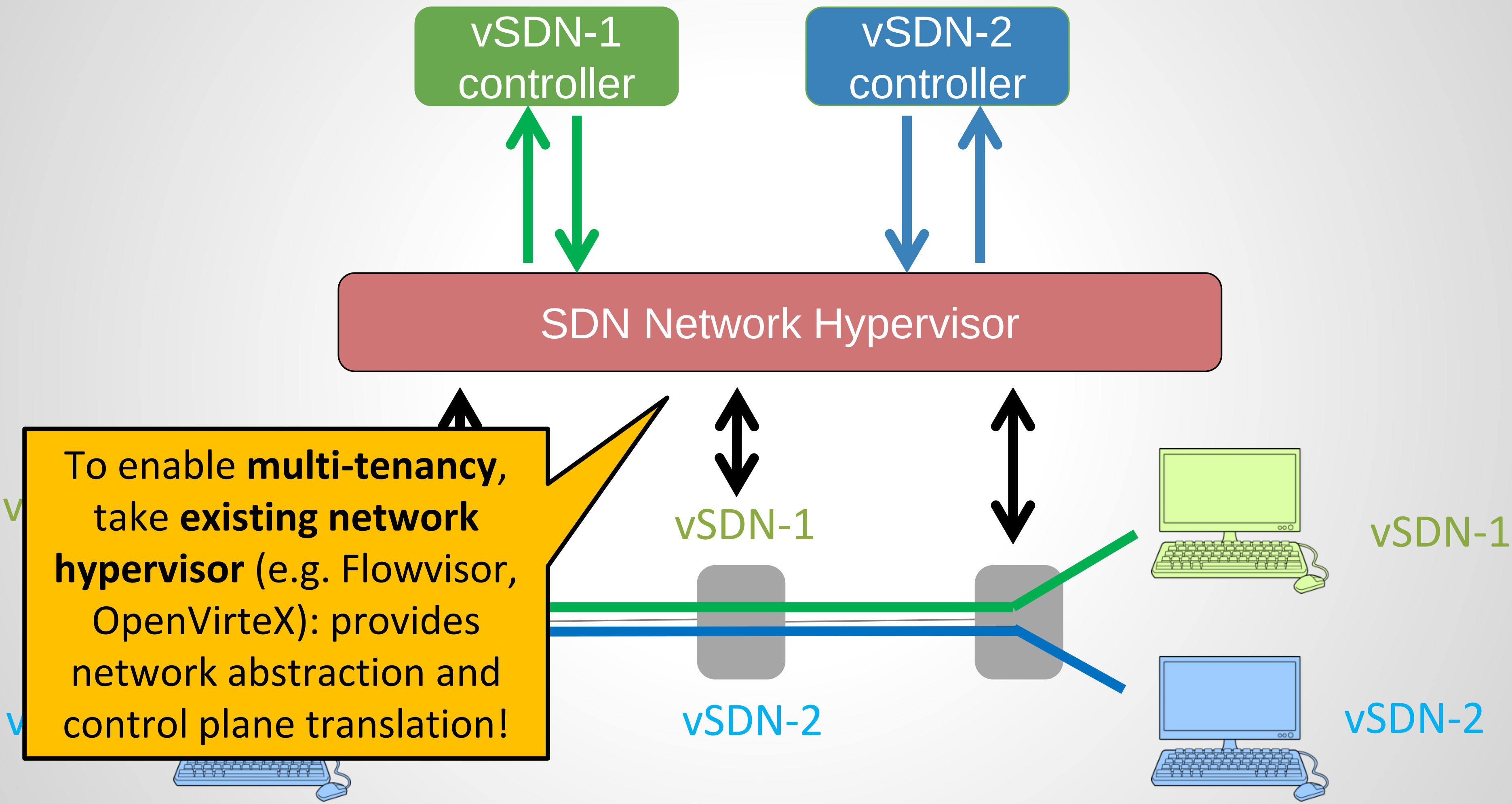
You: hmmm….

The Many Faces of Performance Interference



An Experiment: 2 vSDNs with bw guarantee!

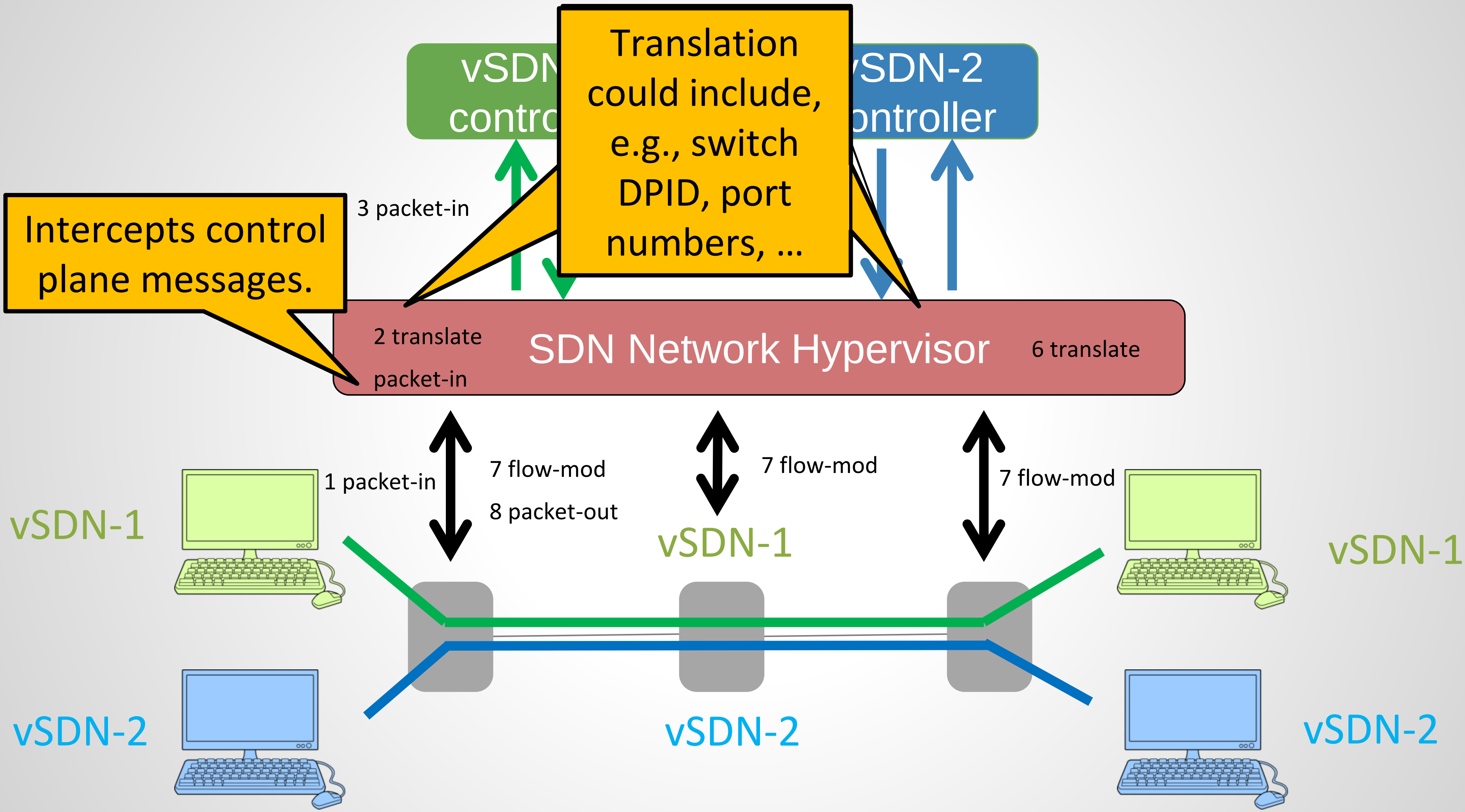
The Many Faces of Performance Interference



To enable **multi-tenancy**, take **existing network hypervisor** (e.g. Flowvisor, OpenVirteX): provides network abstraction and control plane translation!

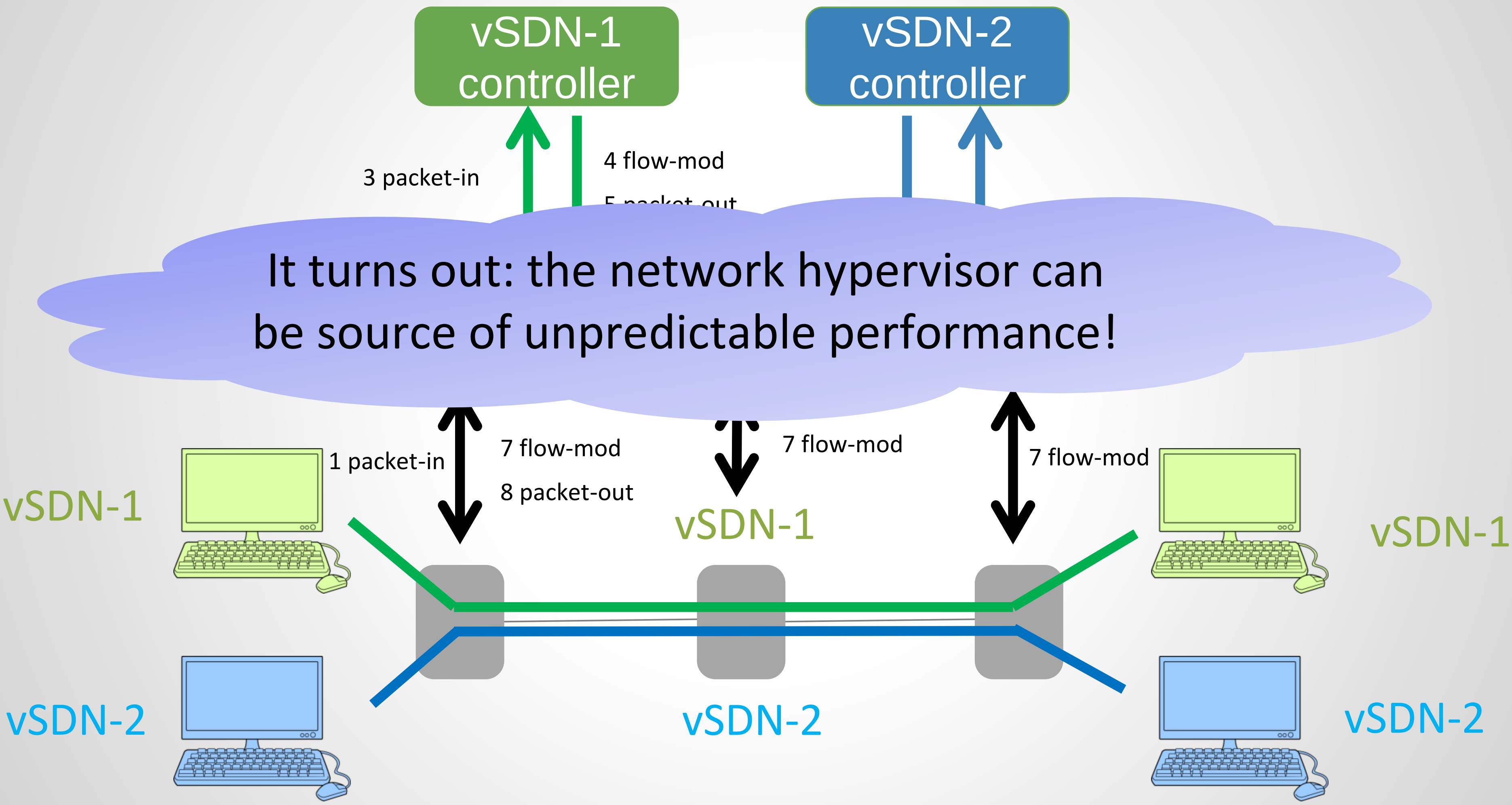
An Experiment: 2 vSDNs with bw guarantee!

The Many Faces of Performance Interference



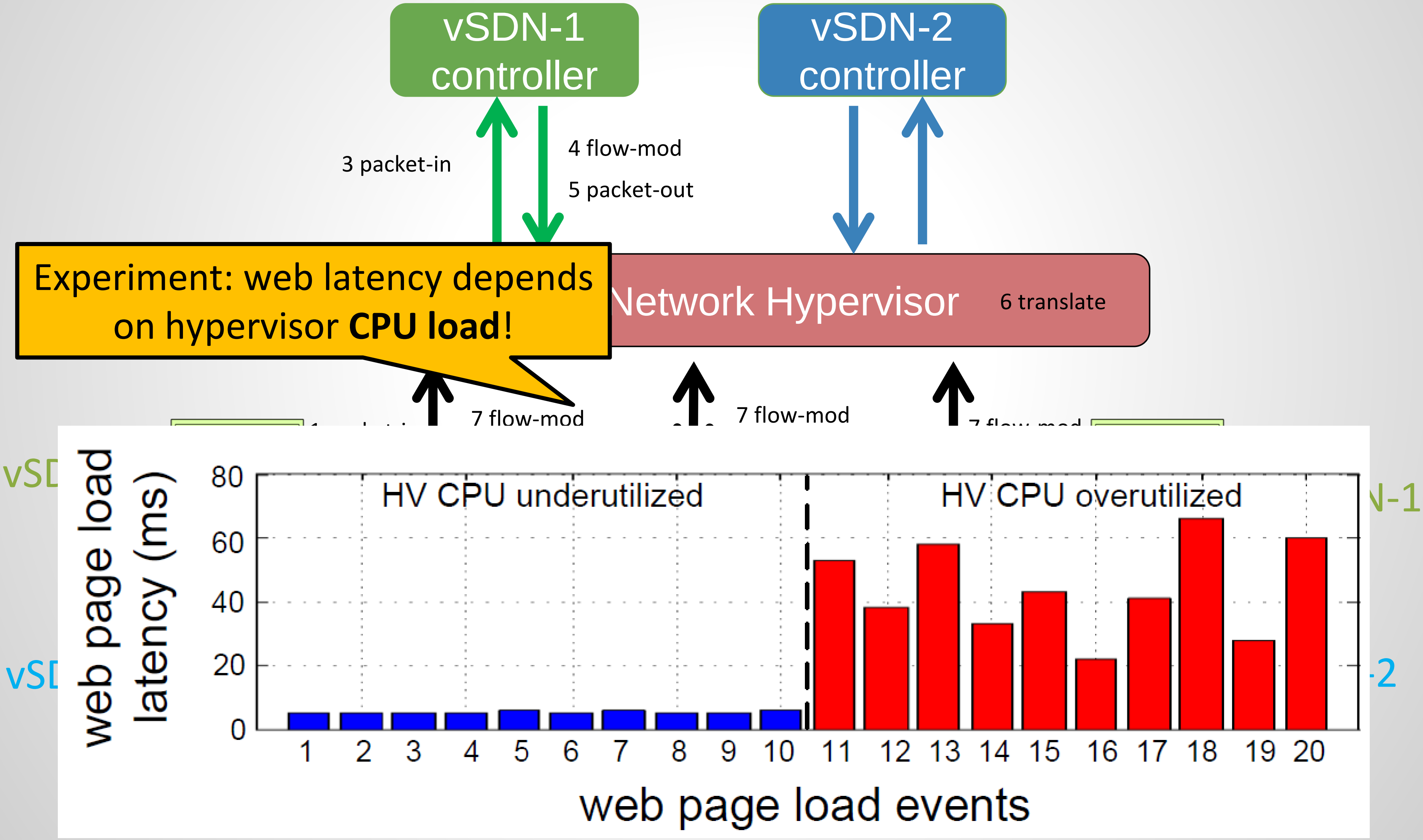
An Experiment: 2 vSDNs with bw guarantee!

The Many Faces of Performance Interference

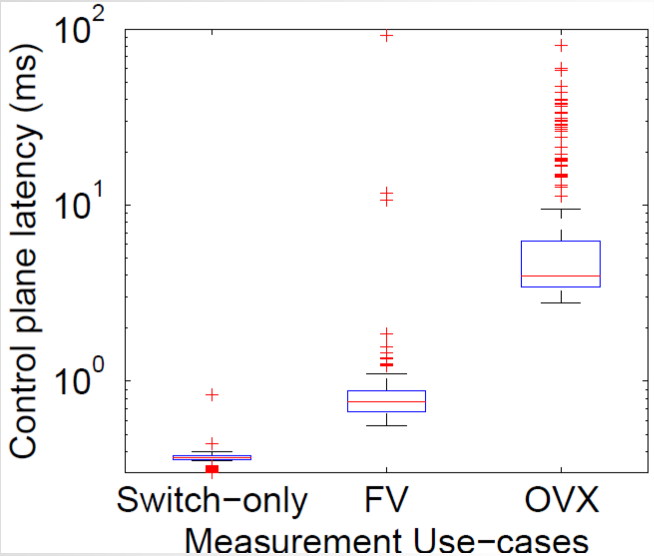


An Experiment: 2 vSDNs with bw guarantee!

The Many Faces of Performance Interference



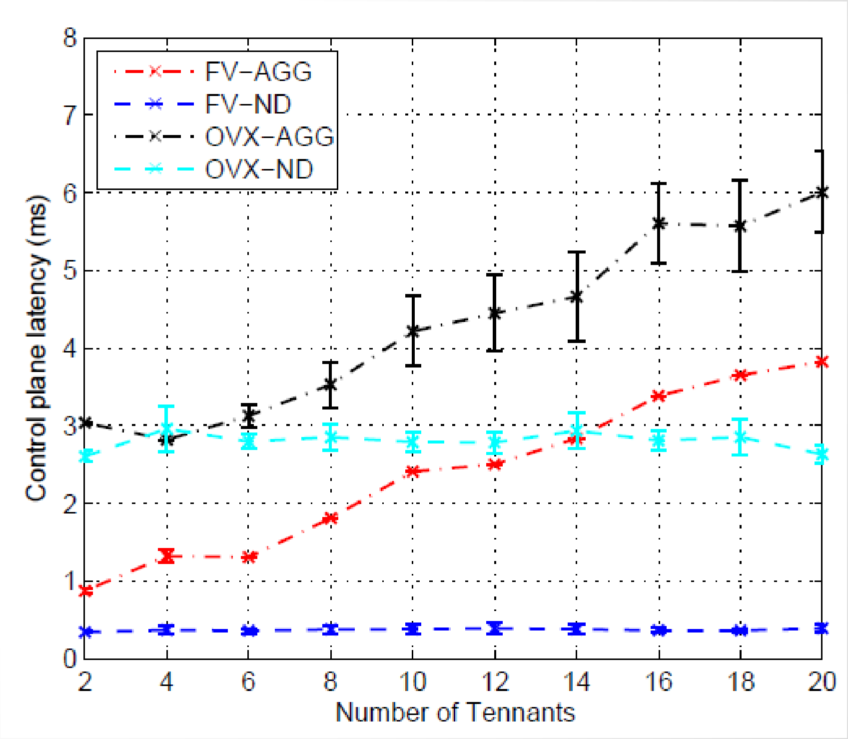
The Many Faces of Performance Interference



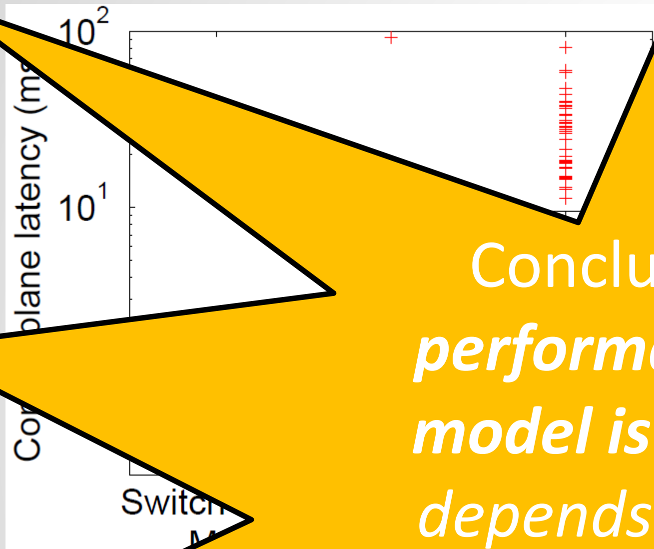
Performance also depends on hypervisor type…

(multithreaded or not, which version of Nagle's algorithm, etc.)

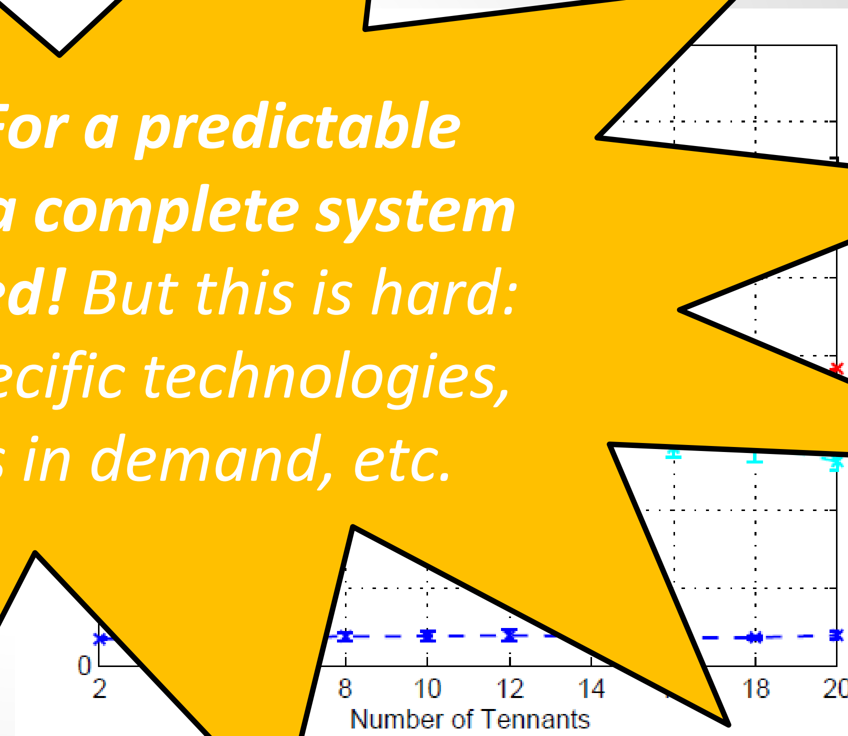
… number of tenants…



The Many Faces of Performance Interference



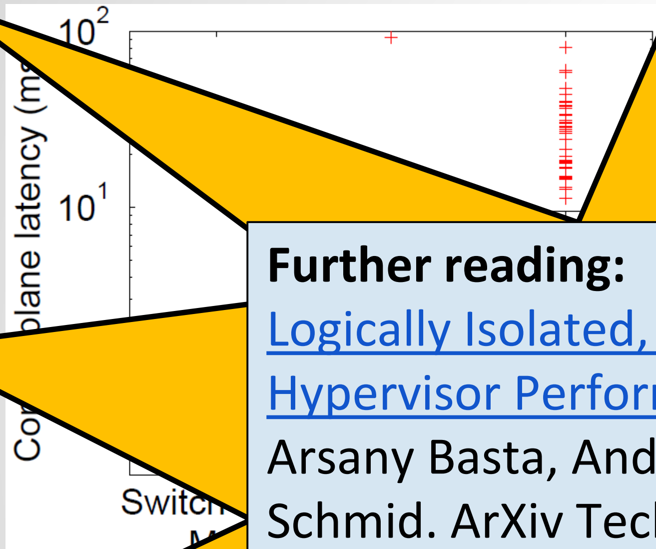
… er of tenants…



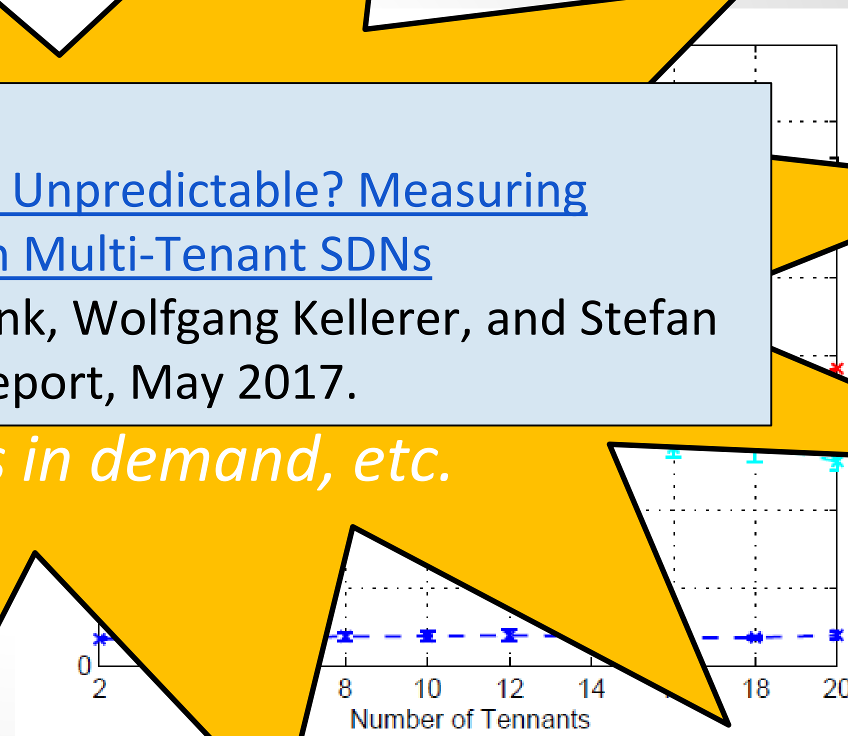
Performance a
on hypervi type
(multithreaded or not, which
of Nagle's algorithm, etc.)

Conclusion: ***For a predictable performance, a complete system model is needed!*** *But this is hard: depends on specific technologies, uncertainties in demand, etc.*

The Many Faces of Performance Interference



… er of tenants…

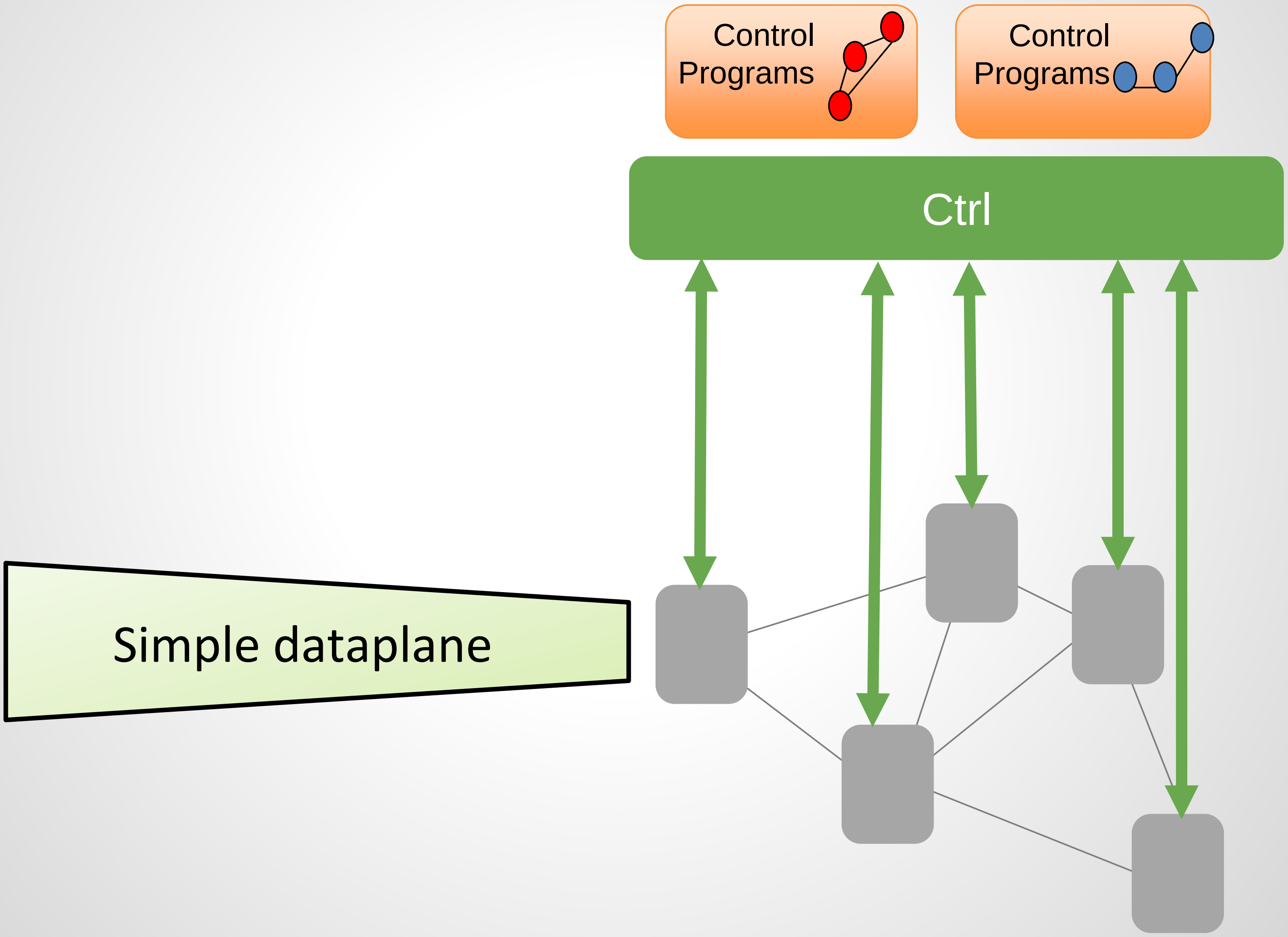


Further reading:
[Logically Isolated, Actually Unpredictable? Measuring Hypervisor Performance in Multi-Tenant SDNs]()
Arsany Basta, Andreas Blenk, Wolfgang Kellerer, and Stefan Schmid. ArXiv Technical Report, May 2017.

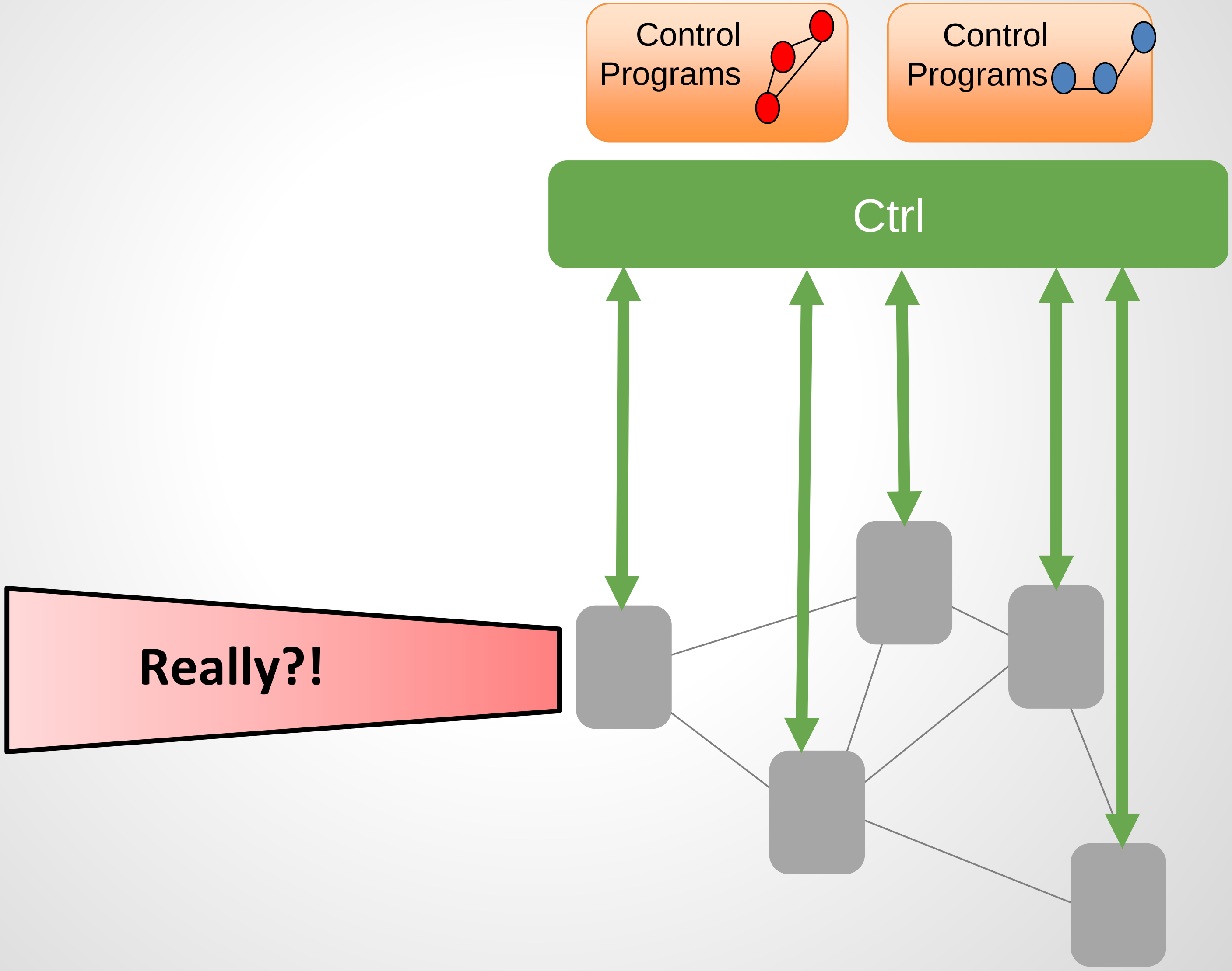
uncertainties in demand, etc.

Performance a…
on hypervi… typ…
(multithreaded or not, which …
of Nagle's algorithm, etc.

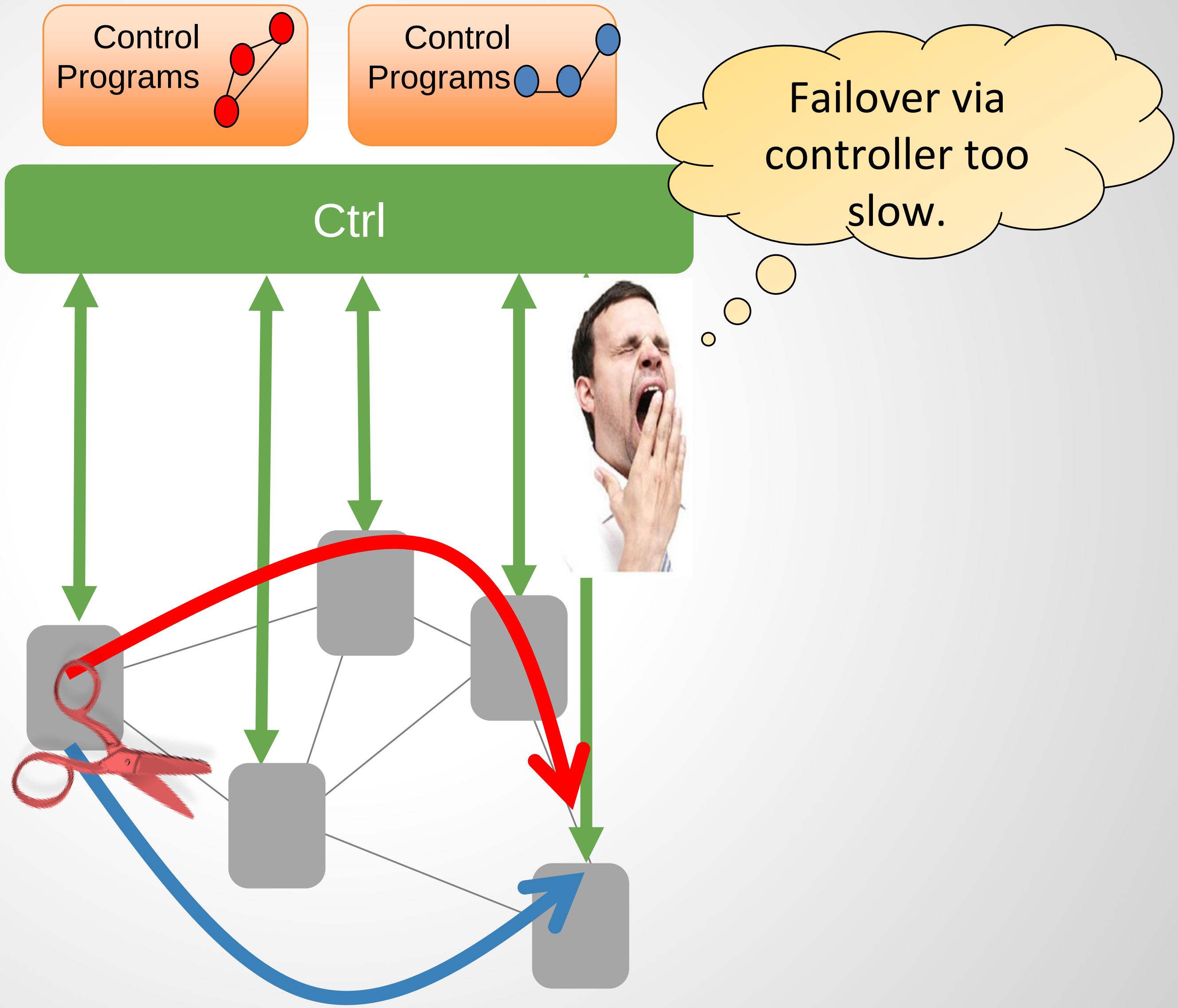
A Mental Model for This Talk



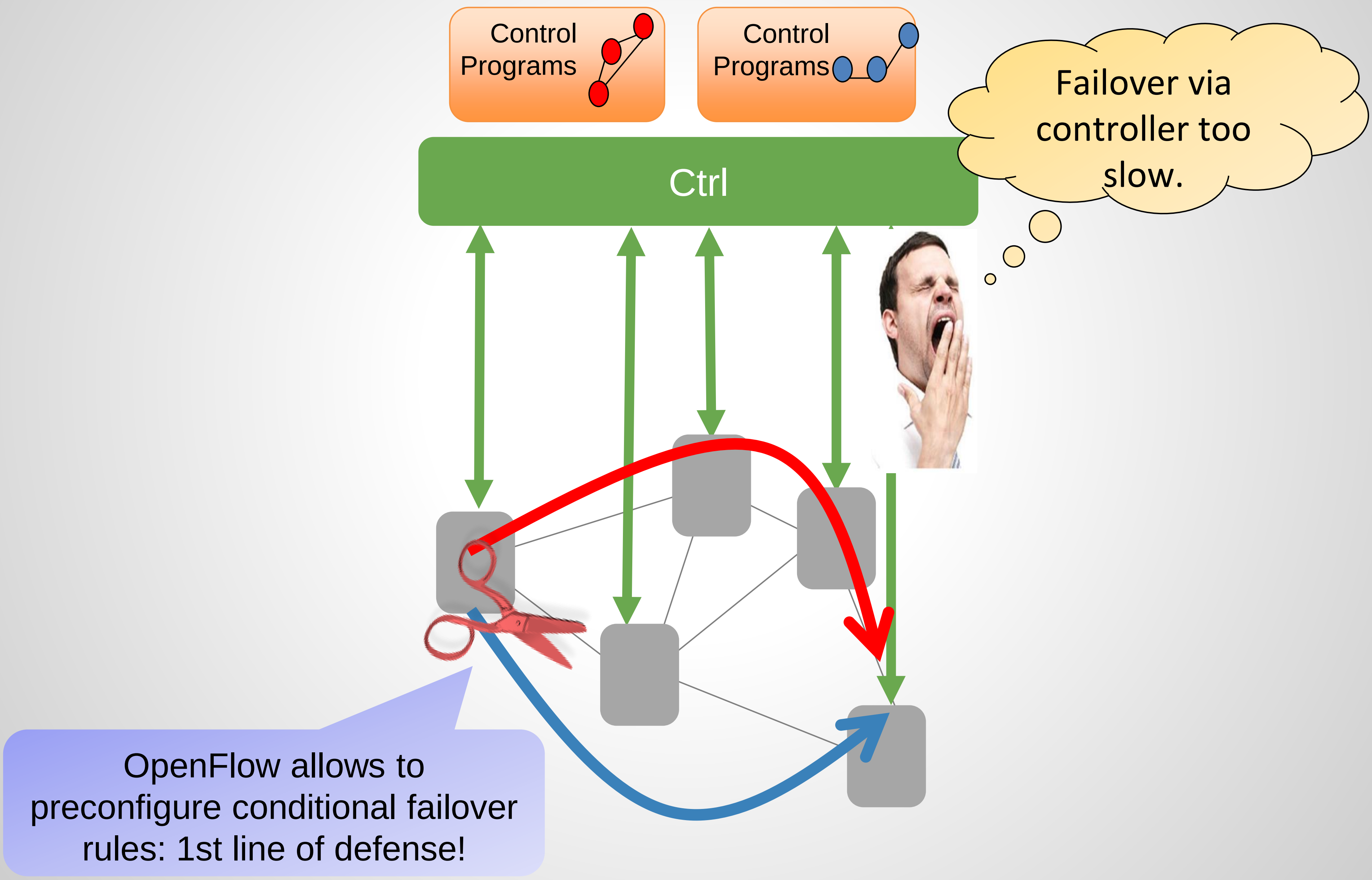
A Mental Model for This Talk



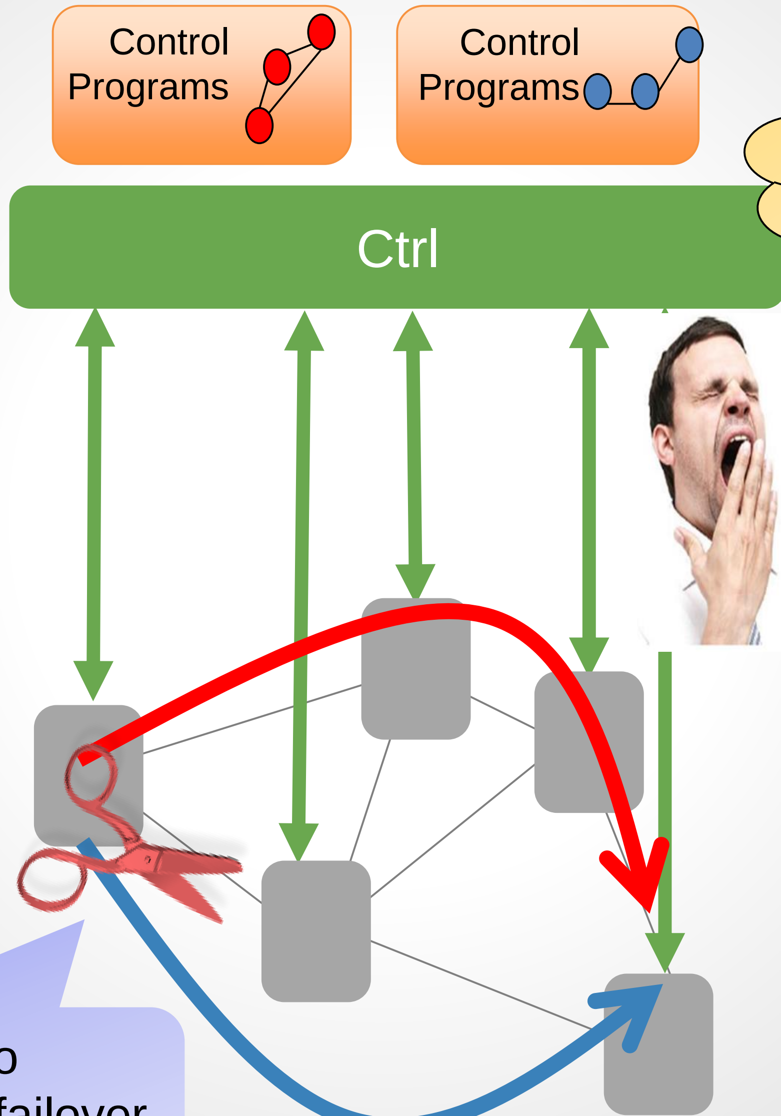
A Mental Model for This Talk



A Mental Model for This Talk



A Mental Model for This Talk



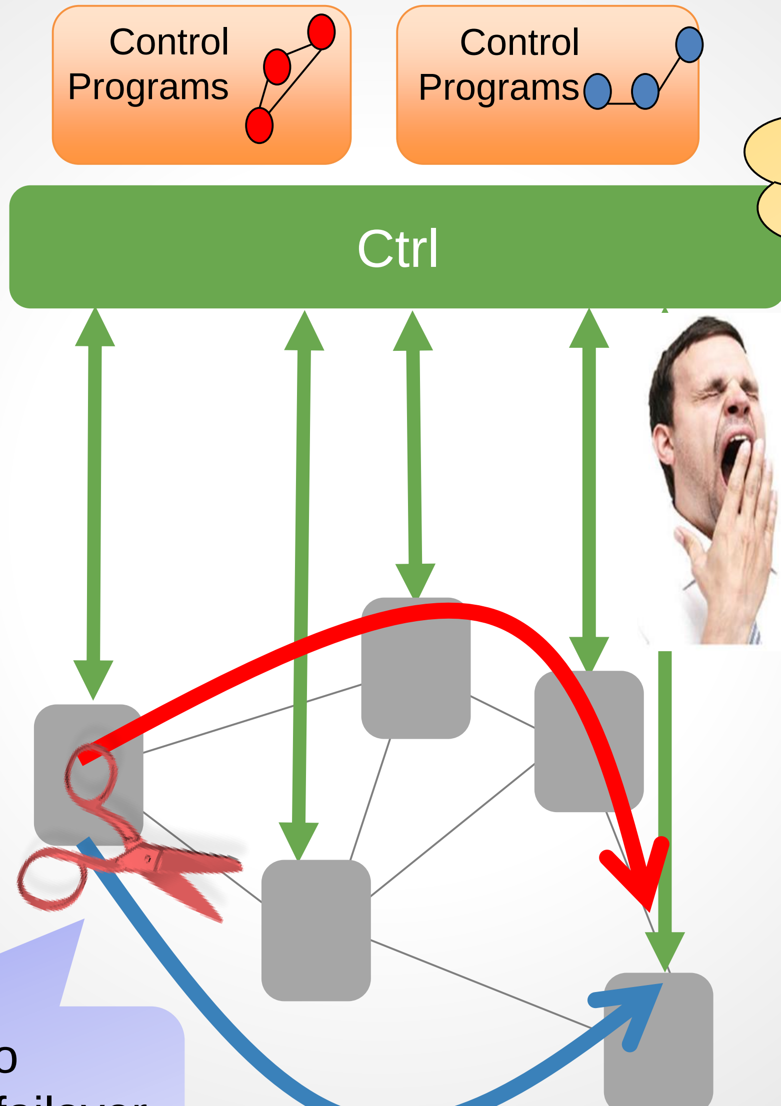
Failover via controller too slow.

The Crux: **How to define conditional rules which have local failure knowledge only?**

OpenFlow allows to preconfigure conditional failover rules: 1st line of defense!

A Mental Model for This Talk

Open problem:* *How many link failures can be tolerated in k-connected network without going through controller?



Failover via controller too slow.

***The Crux:* How to define conditional rules which have local failure knowledge only?**

OpenFlow allows to preconfigure conditional failover rules: 1st line of defense!

Solution: Use Arborescences (Chiesa et al.)

- Assume:
  - *k*-connected network *G*
  - destination *d*
  - *G* decomposed into *k d*-rooted arc-disjoint spanning arborescences

Known result: always exist in k-connected graphs (efficient)

Basic principle:

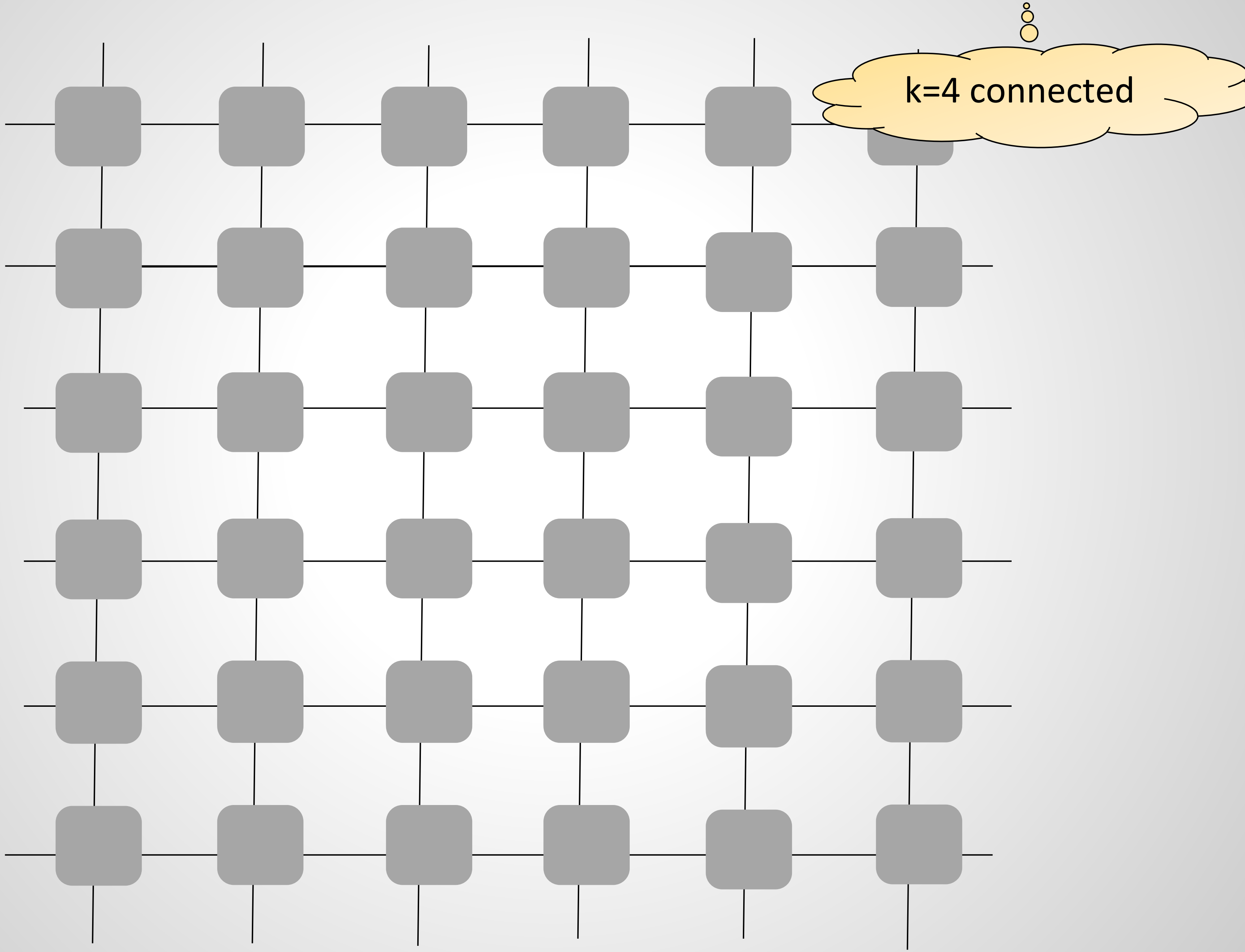
- Route along fixed arborescence (“directed spanning tree”) towards the destination *d*
- If packet hits a failed edge at vertex *v*, reroute along a different arborescence

The Crux: which arborescence to choose next? Influences resiliency!

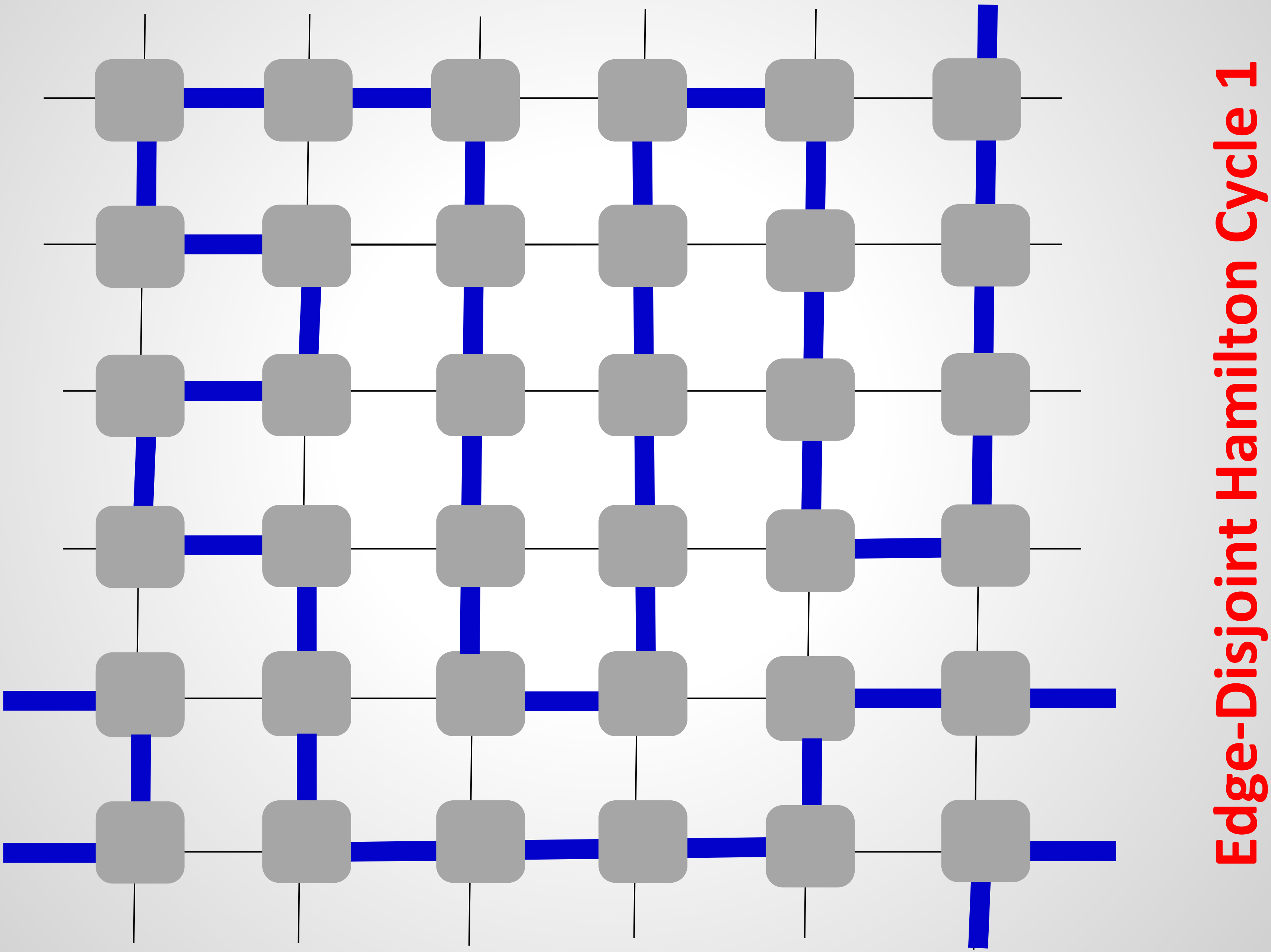
Simple Example: Hamilton Cycle

Chiesa et al.: if *k*-connected graph has *k* arc disjoint Hamilton Cycles, *k*-1 resilient routing can be constructed!

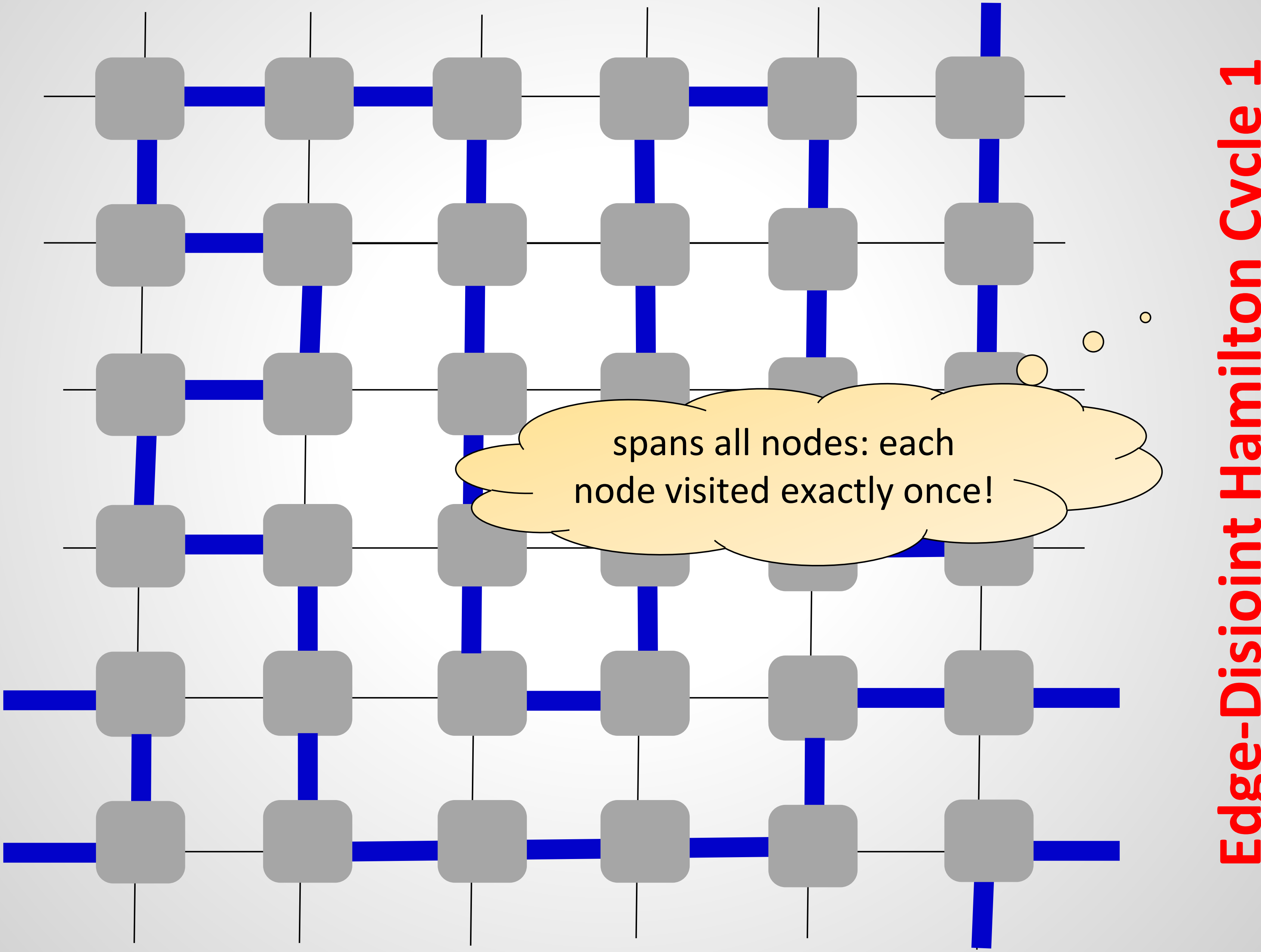
Example: 3-Resilient Routing Function for 2-dim Torus



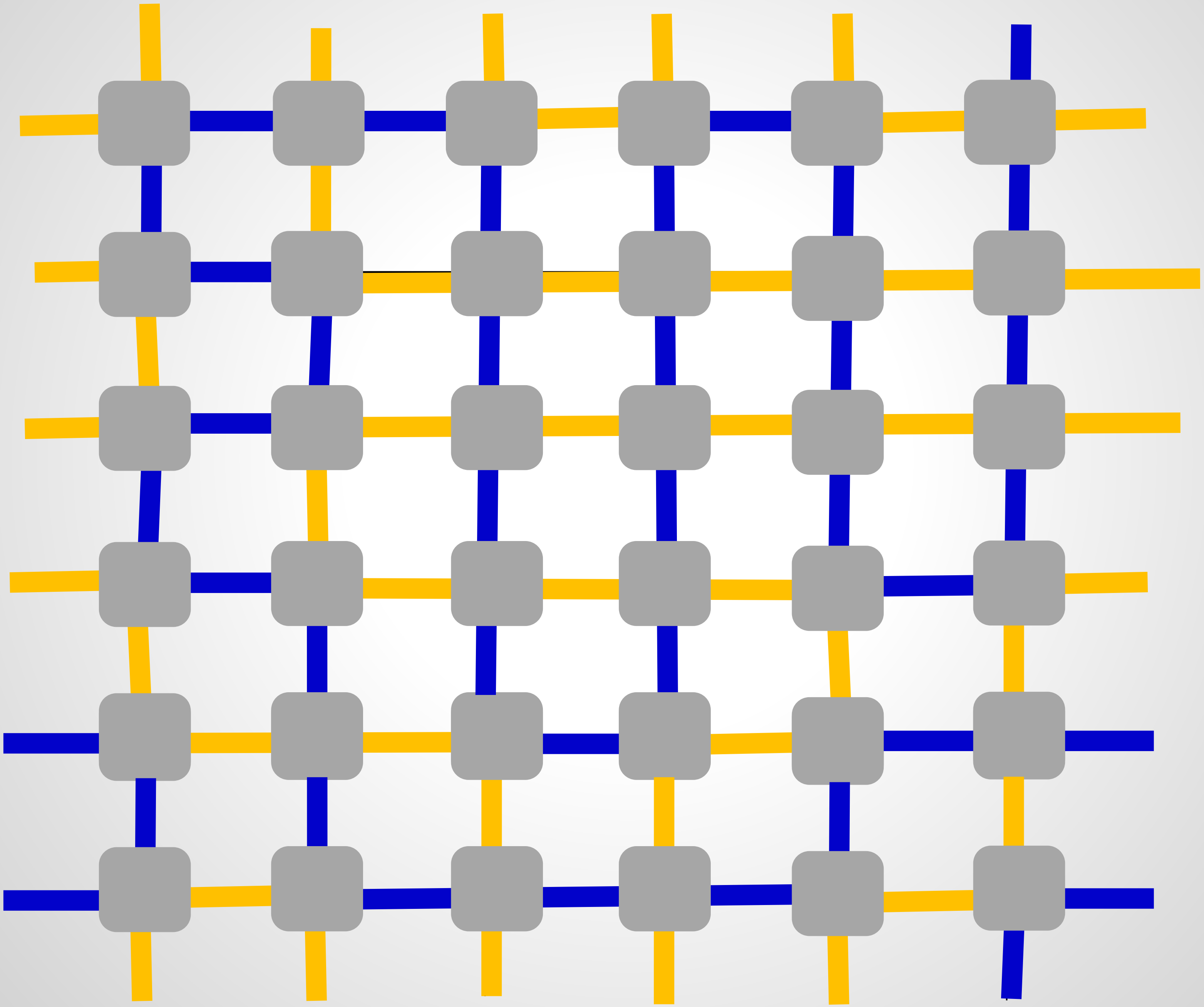
Example: 3-Resilient Routing Function for 2-dim Torus



Example: 3-Resilient Routing Function for 2-dim Torus

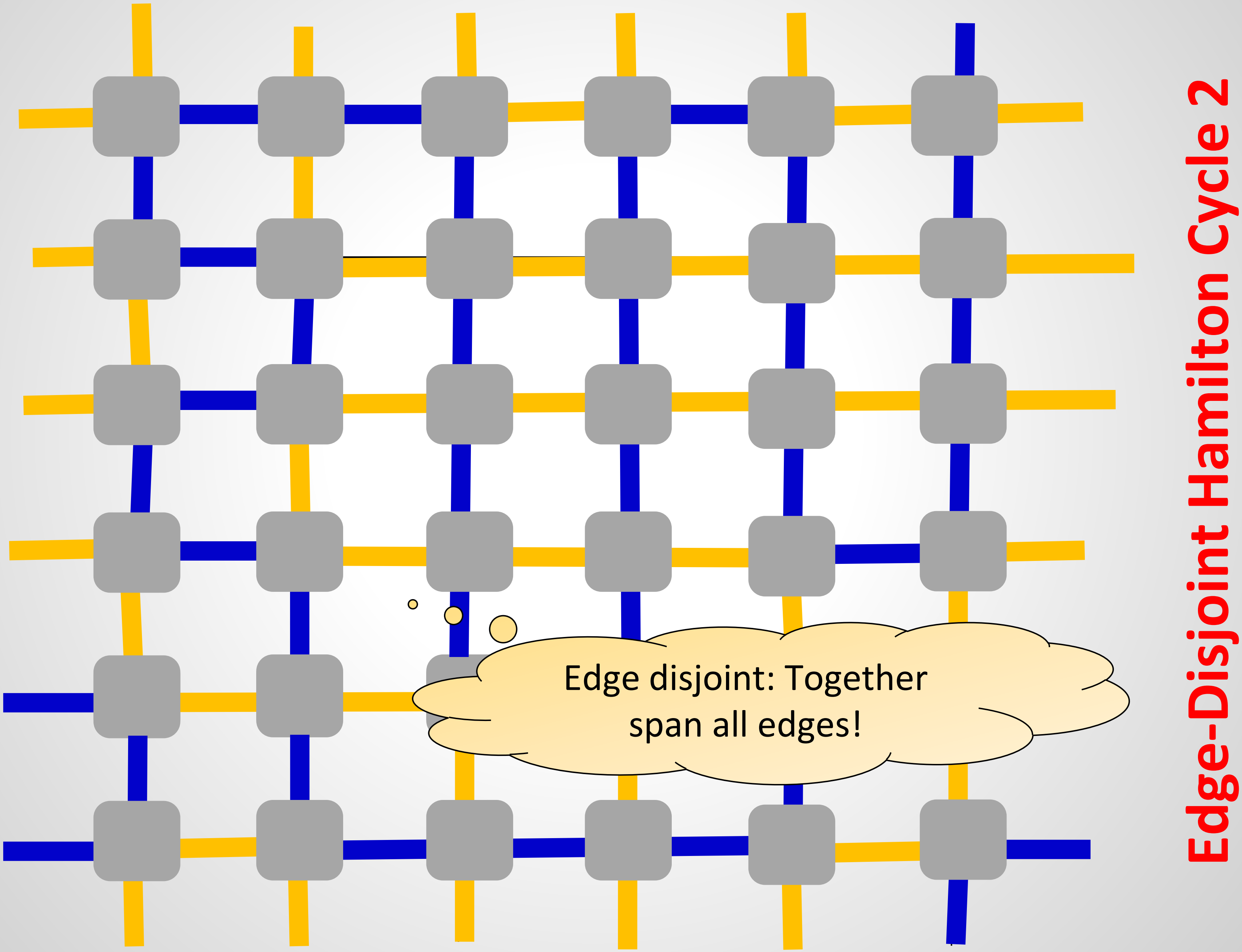


Example: 3-Resilient Routing Function for 2-dim Torus

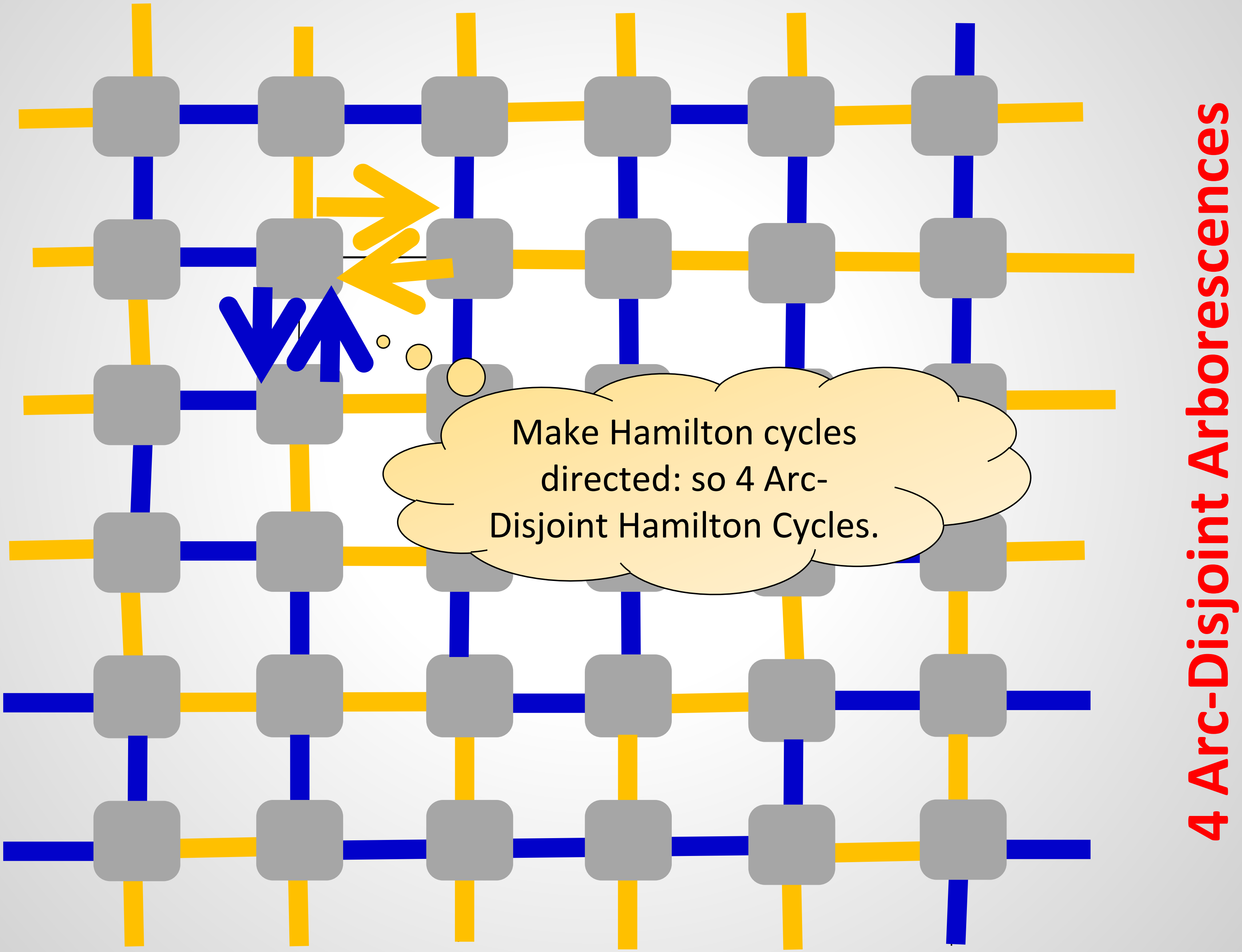


Edge-Disjoint Hamilton Cycle 2

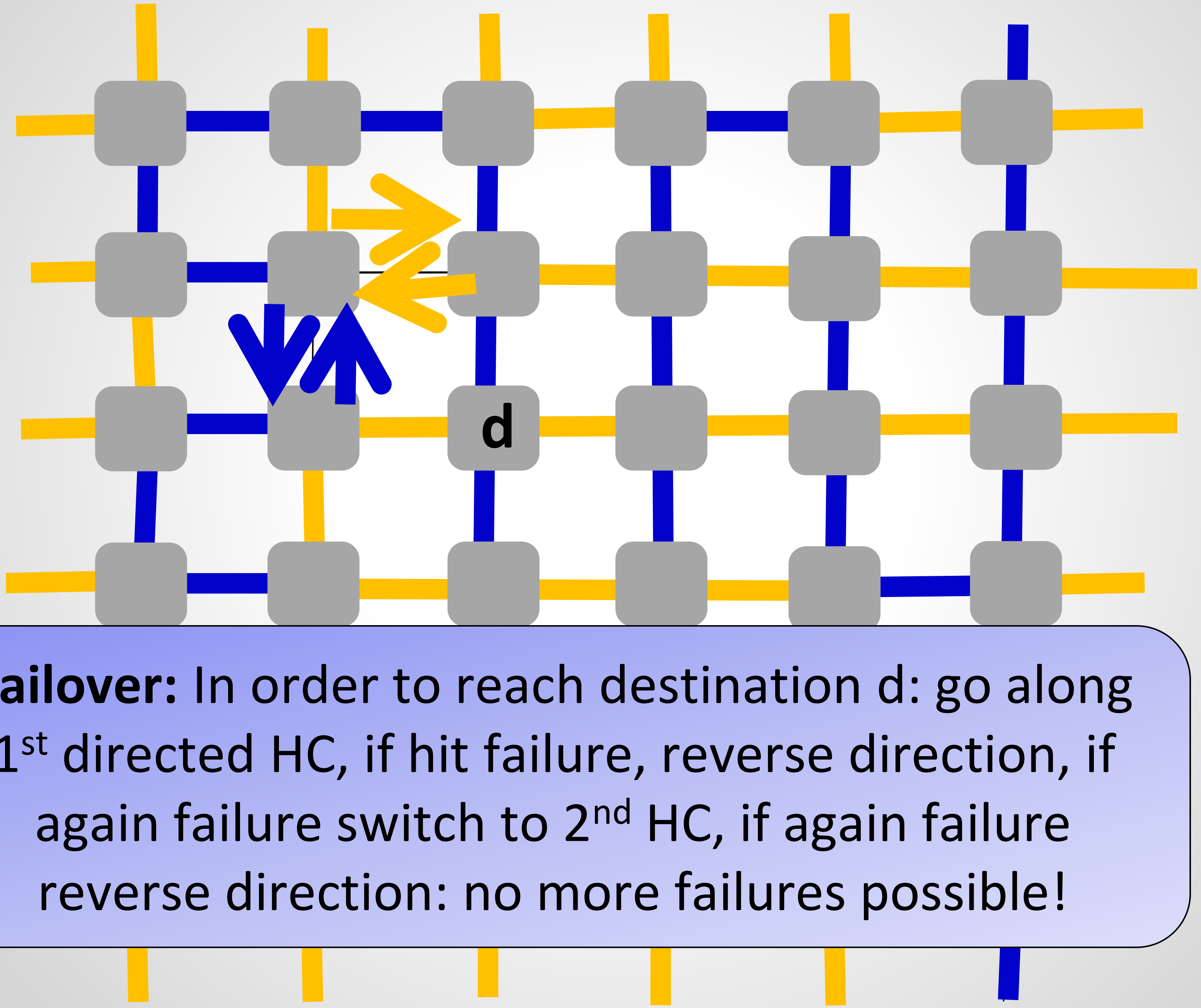
Example: 3-Resilient Routing Function for 2-dim Torus



Example: 3-Resilient Routing Function for 2-dim Torus



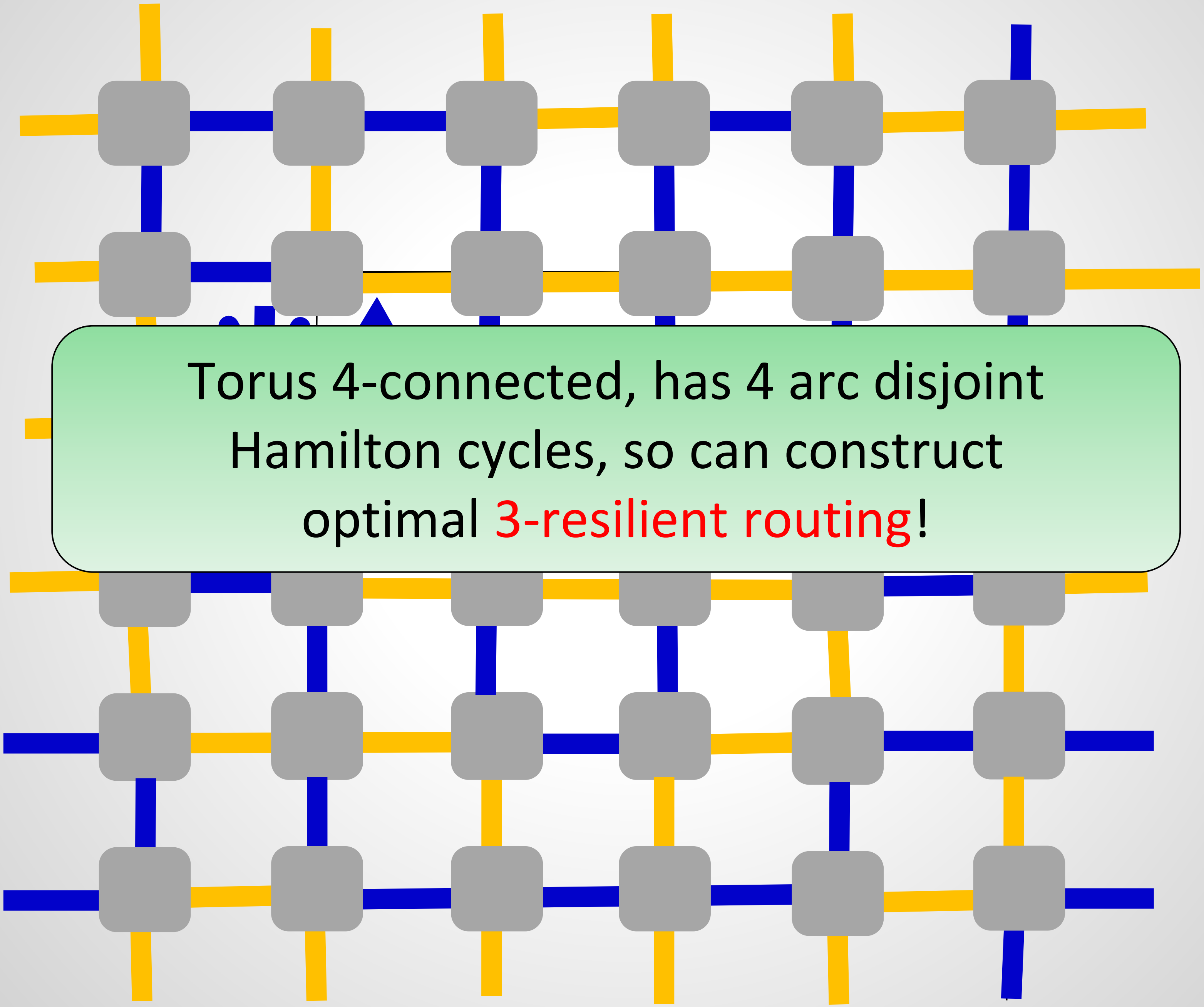
Example: 3-Resilient Routing Function for 2-dim Torus



Failover: In order to reach destination d: go along 1st directed HC, if hit failure, reverse direction, if again failure switch to 2nd HC, if again failure reverse direction: no more failures possible!

4 Arc-Disjoint Arborescences

Example: 3-Resilient Routing Function for 2-dim Torus



Torus 4-connected, has 4 arc disjoint Hamilton cycles, so can construct optimal 3-resilient routing!

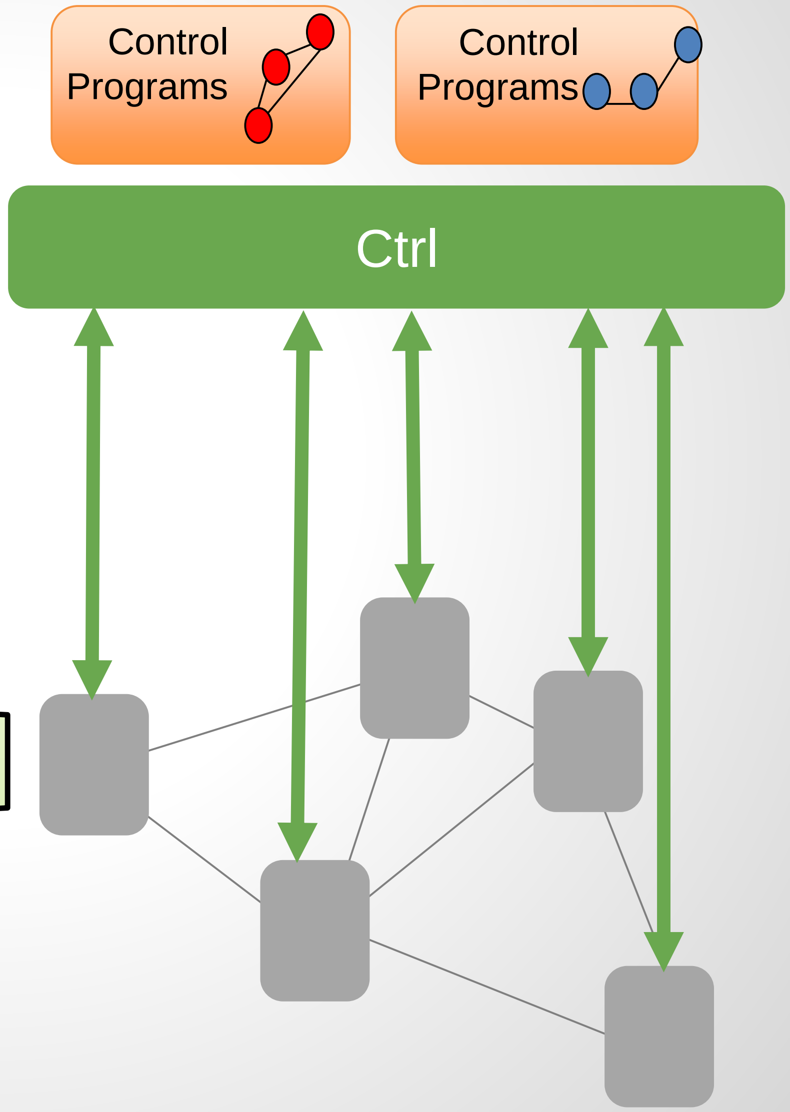
4 Arc-Disjoint Arborescences

Further Reading

Exploring the Limits of Static Failover Routing
Marco Chiesa, Andrei Gurtov, Aleksander Mądry, Slobodan Mitrović, Ilya Nikolaevkiy, Aurojit Panda, Michael Schapira, Scott Shenker. Arxiv Technical Report, 2016.

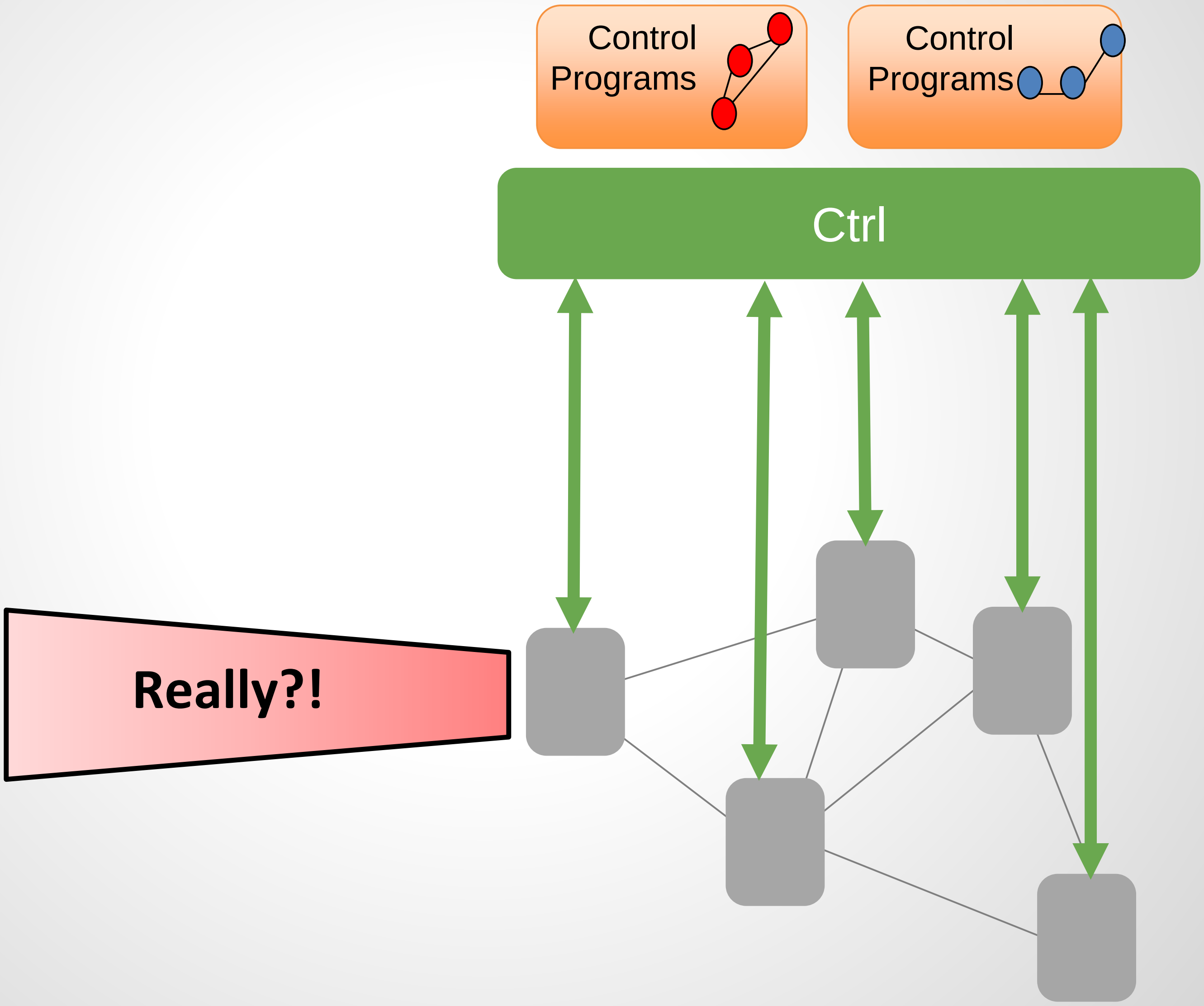
Load-Optimal Local Fast Rerouting for Dependable Networks
Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
47th IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**), Denver, Colorado, USA, June 2017.

A Mental Model for This Talk



SDNs support formal verifiability

A Mental Model for This Talk



Examples: Reachability and What-if Analysis

Questions operators may have:

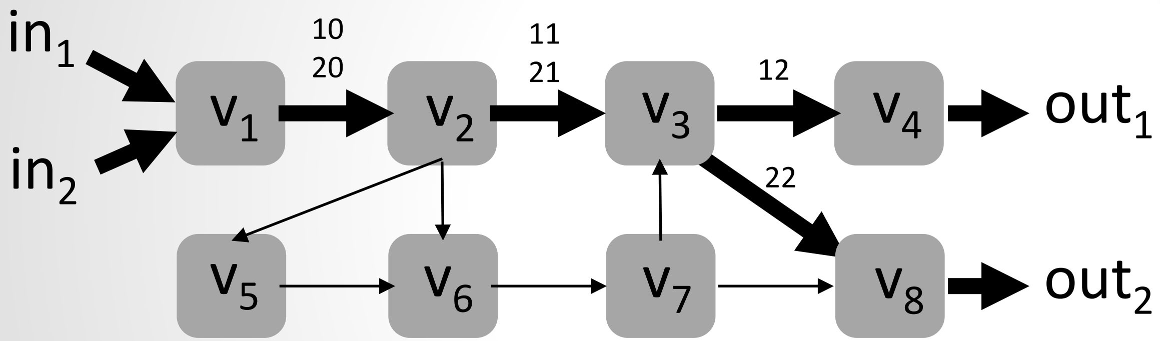
- **Reachability:** «Is it possible / not possible to reach, from ingress port x, egress port y?»
  - To ensure connectivity
  - But also policies: professor network not reachable from student dorms (logical isolation)

- **What-if analysis:** «How can the forwarding behavior look like if there are up to *k* concurrent link failures?»

Policy-compliance under failures is difficult!

«Simple» in MPLS

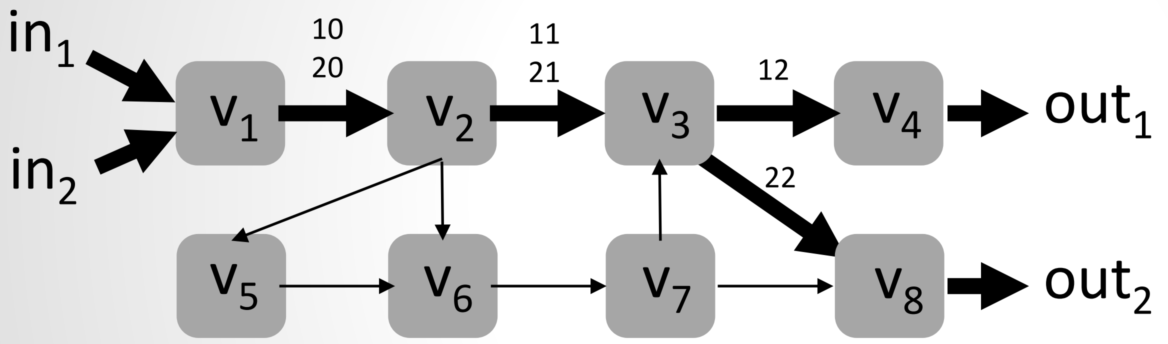
- MPLS = forwarding based on a label stack
  - Idea: forward according to top label
  - Usually, top label swapped at each hop



Default routing of two flows

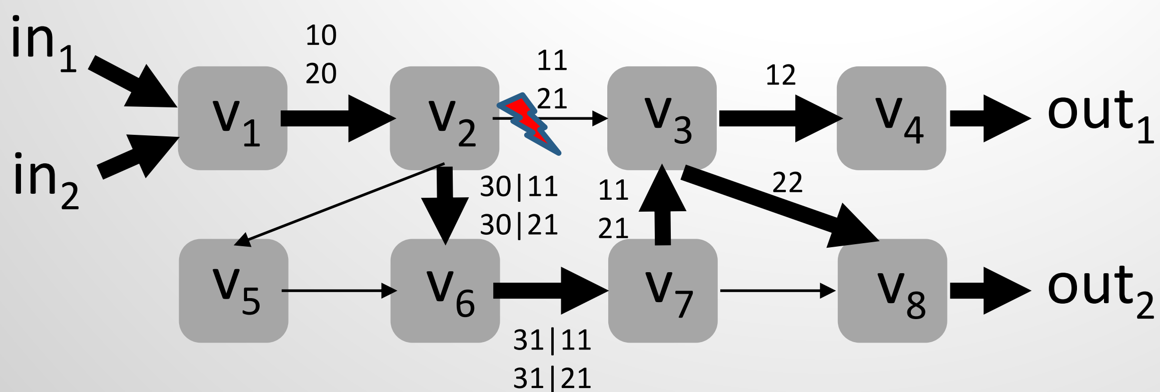
«Simple» in MPLS

- MPLS = forwarding based on a label stack
  - Idea: forward according to top label
  - Usually, top label swapped at each hop



Default routing of two flows

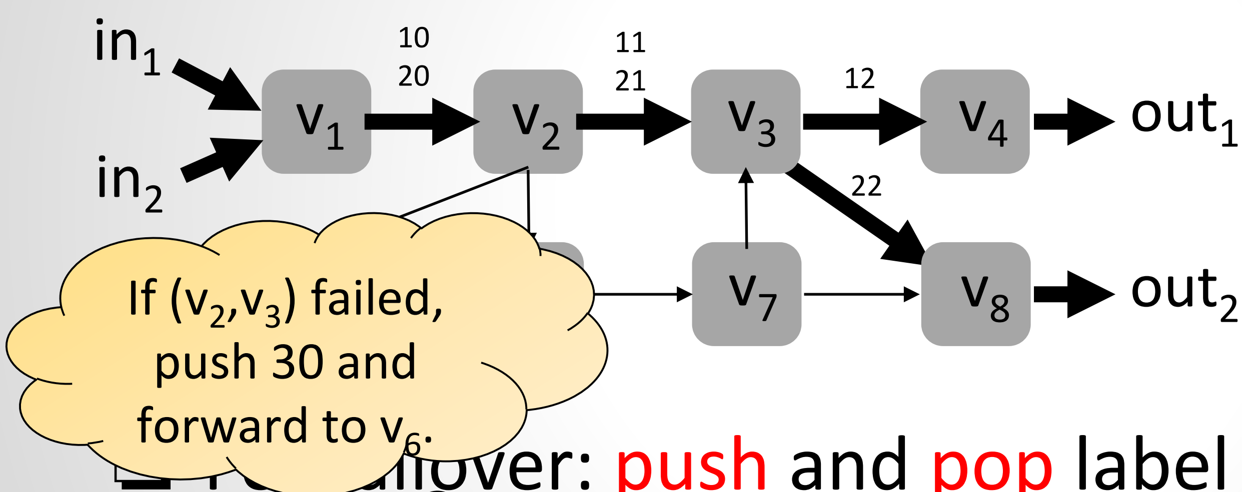
- For failover: push and pop label



One failure: push 30: route around (v_2,v_3)

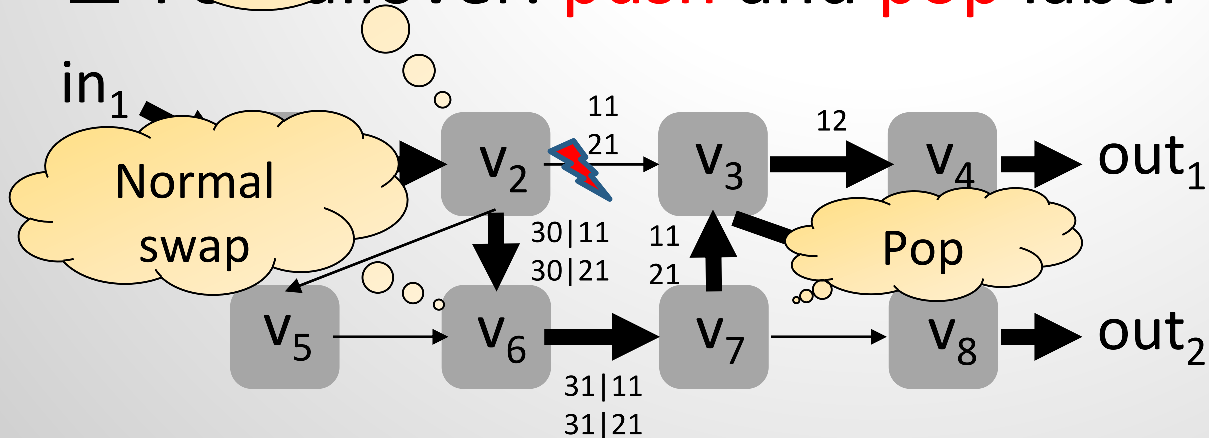
«Simple» in MPLS

- MPLS = forwarding based on a label stack
  - Idea: forward according to top label
  - Usually, top label swapped at each hop



Default routing of two flows

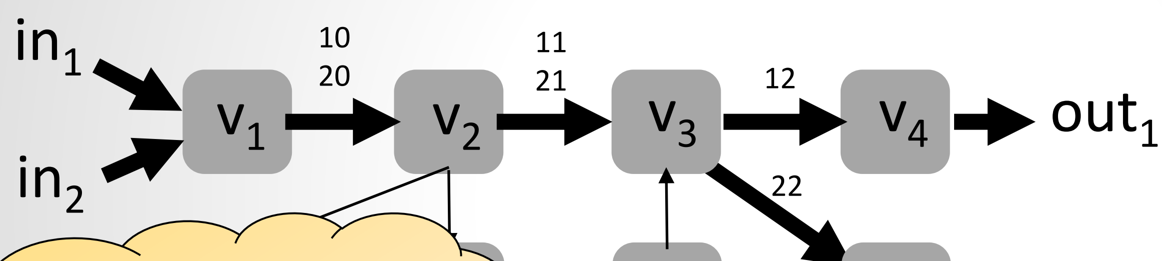
- …ailover: push and pop label



One failure: push 30: route around (v_2,v_3)

«Simple» in MPLS

- MPLS = forwarding based on a label stack
  - Idea: forward according to top label
  - Usually, top label swapped at each hop

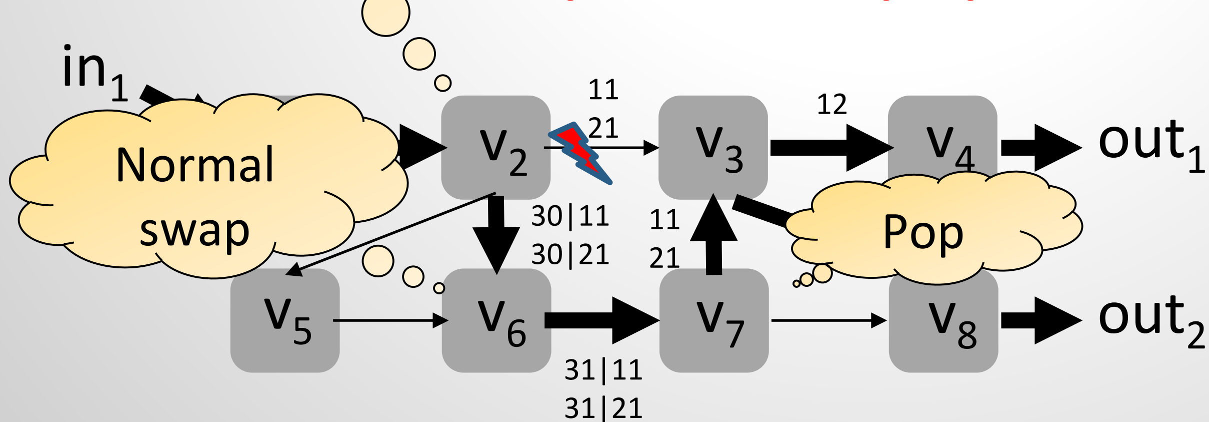


Default routing of two flows

If (v2,…
push…
forward to v6.

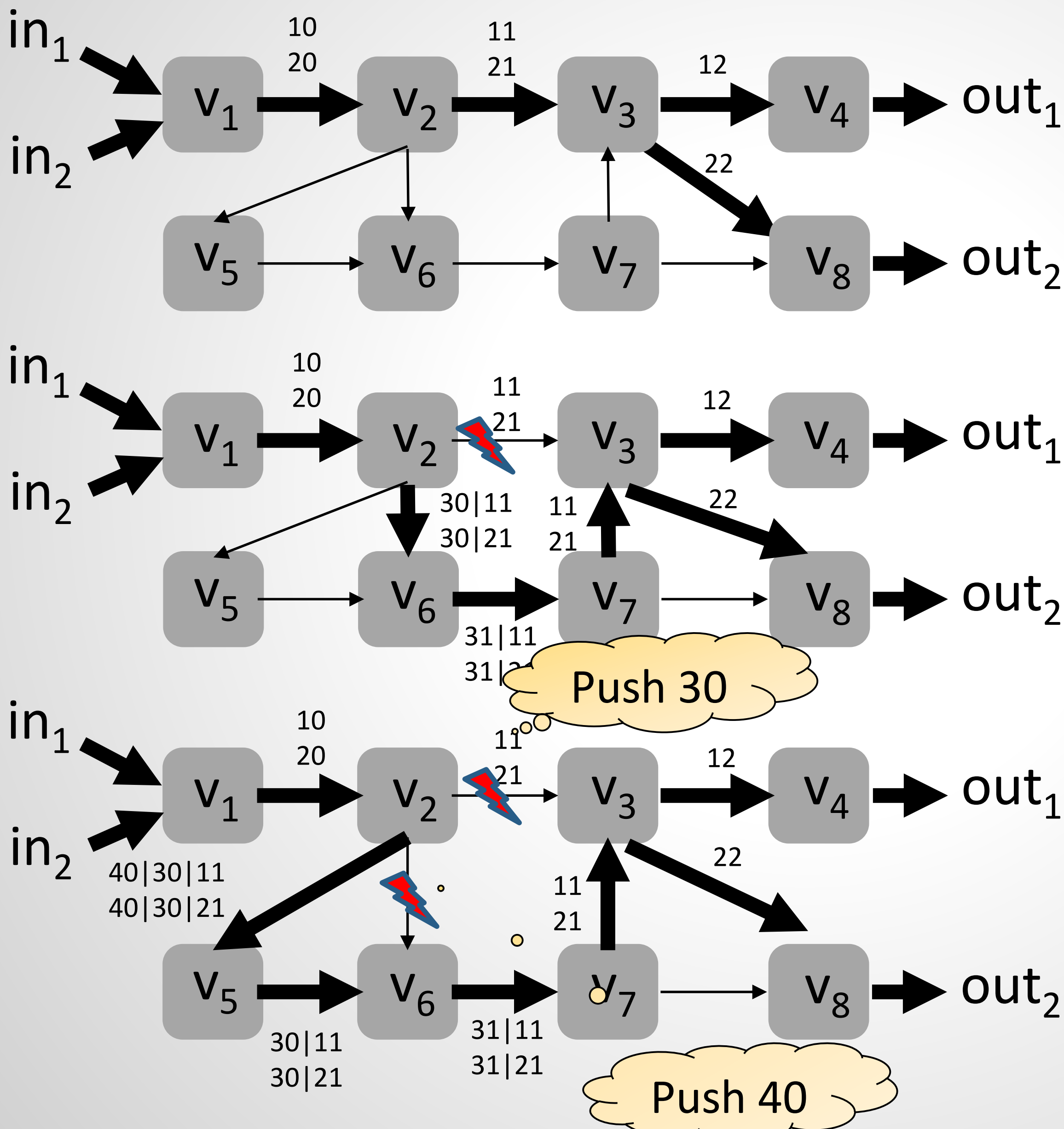
What about multiple link failures?

- F… ailover: push and pop label



One failure: push 30: route around (v_2,v_3)

Multiple Link Failures: Push Recursively!



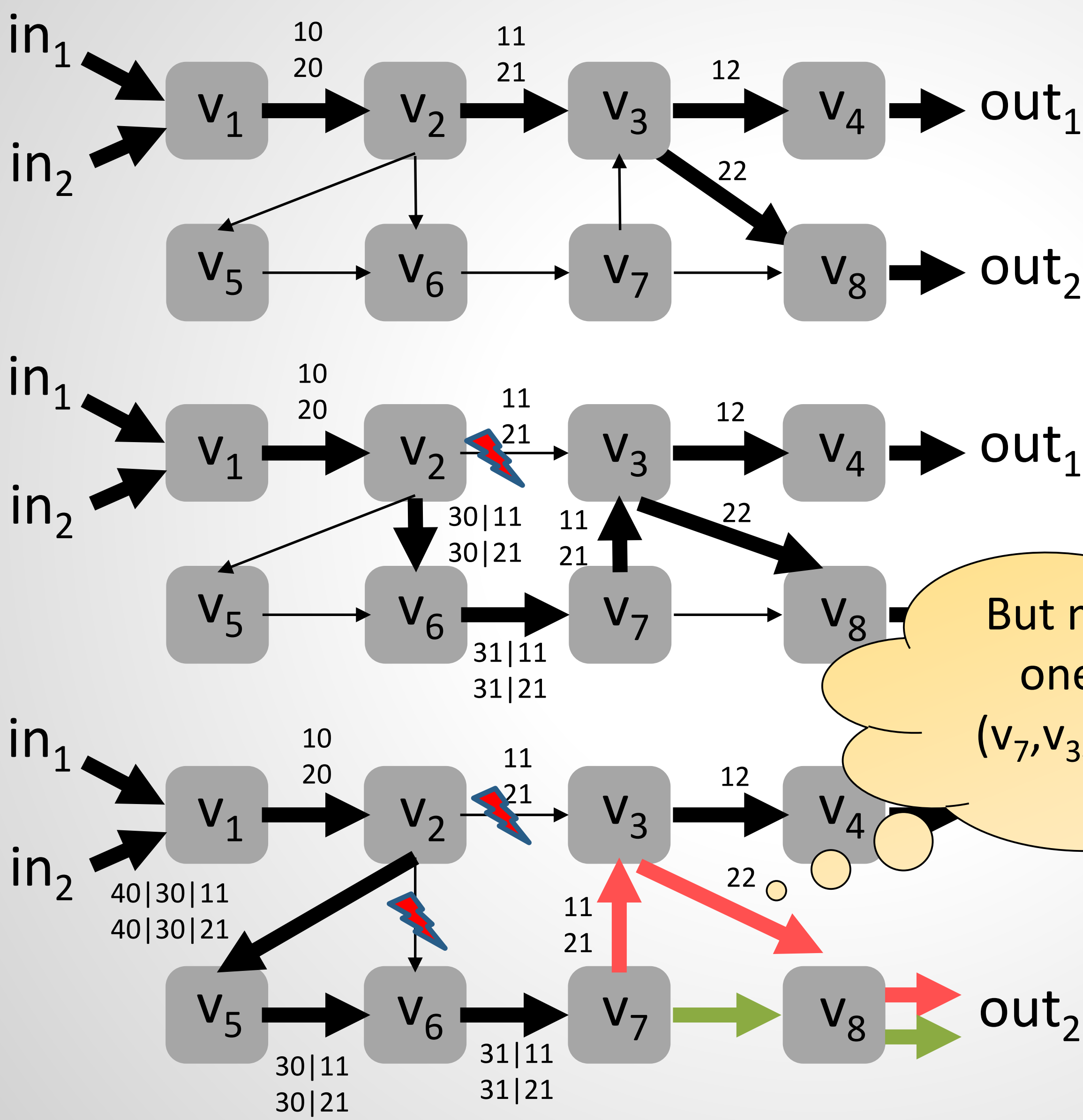
Original Routing

One failure: push 30: route around (v_2,v_3)

Two failures: first push 30: route around (v_2,v_3)

Recursively push 40: route around (v_2,v_6)

Multiple Link Failures: Push Recursively!



Original Routing

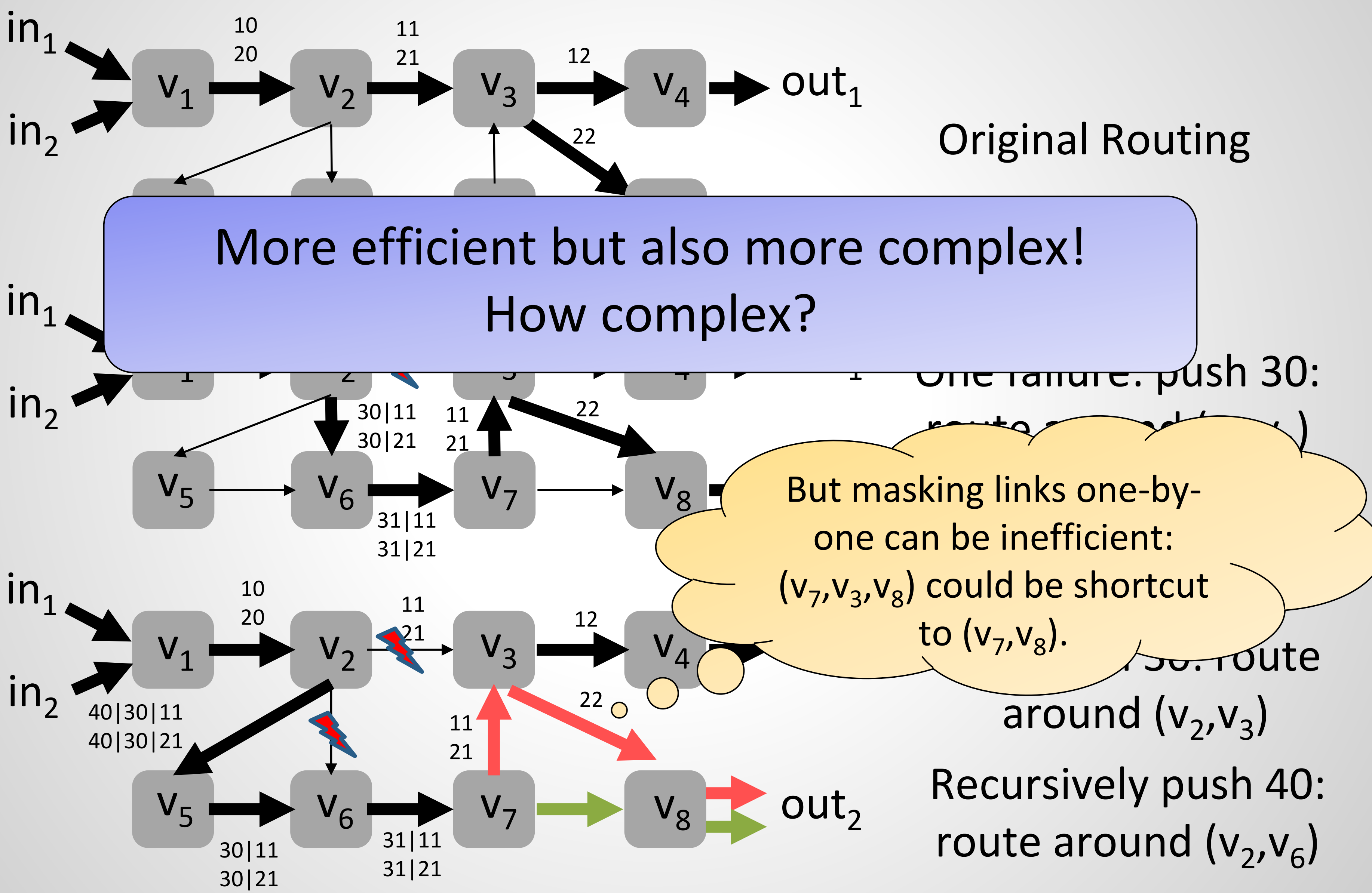
One failure: push 30:

But masking links one-by-one can be inefficient: (v_7,v_3,v_8) could be shortcut to (v_7,v_8).

around (v_2,v_3)

Recursively push 40: route around (v_2,v_6)

Multiple Link Failures: Push Recursively!



Tables

FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_7, v_3)	21	(v_3, v_8)	$swap(22)$
τ_{v_4}	(v_3, v_4)	12	out_1	pop
τ_{v_5}	(v_2, v_5)	40	(v_5, v_6)	pop
τ_{v_6}	(v_2, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	61	(v_6, v_7)	$swap(62)$
	(v_5, v_6)	71	(v_6, v_7)	$swap(72)$
τ_{v_7}	(v_6, v_7)	31	(v_7, v_3)	pop
	(v_6, v_7)	62	(v_7, v_3)	$swap(11)$
	(v_6, v_7)	72	(v_7, v_8)	$swap(22)$
τ_{v_8}	(v_3, v_8)	22	out_2	pop
	(v_7, v_8)	22	out_2	pop

Flow Table

Protected link

Alternative link

Label

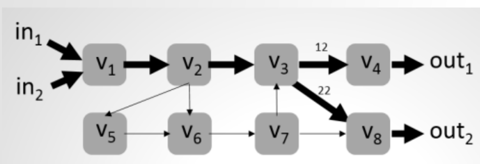
local FFT	Out-I	In-Label	Out-I	op
τ_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$push(30)$
	(v_2, v_3)	21	(v_2, v_6)	$push(30)$
	(v_2, v_6)	30	(v_2, v_5)	$push(40)$

global FFT	Out-I	In-Label	Out-I	op
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$

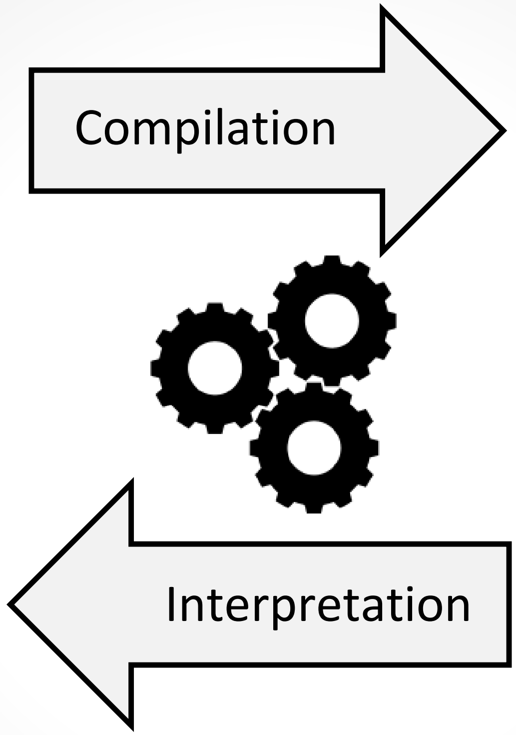
Failover Tables

Can be verified in polynomial time:
Leverage automata theory!

FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_7, v_3)	21	(v_3, v_8)	$swap(22)$
τ_{v_4}	(v_3, v_4)	12	out_1	pop
τ_{v_5}	(v_2, v_5)	40	(v_5, v_6)	pop
τ_{v_6}	(v_2, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	61	(v_6, v_7)	$swap(62)$
	(v_5, v_6)	71	(v_6, v_7)	$swap(72)$
τ_{v_7}	(v_6, v_7)	31	(v_7, v_3)	pop
	(v_6, v_7)	62	(v_7, v_3)	$swap(11)$
	(v_6, v_7)	72	(v_7, v_8)	$swap(22)$
τ_{v_8}	(v_3, v_8)	22	out_2	pop
	(v_7, v_8)	22	out_2	pop



local FFT	Out-I	In-Label	Out-I	op
τ_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$push(30)$
	(v_2, v_3)	21	(v_2, v_6)	$push(30)$
	(v_2, v_6)	30	(v_2, v_5)	$push(40)$
global FFT	**Out-I**	**In-Label**	**Out-I**	**op**
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$



$$pX \Rightarrow qXX$$
$$pX \Rightarrow qYX$$
$$qY \Rightarrow rYY$$
$$rY \Rightarrow r$$
$$rX \Rightarrow pX$$

MPLS configurations,
Segment Routing etc.

Pushdown Automaton and Prefix
Rewriting System Theory

MPLS vs SDN

- (Simplified) MPLS rules: prefix rewriting

  FT: $in \times L \rightarrow out \times OP$, where $OP = \{swap, push, pop\}$

  FFT: $out \times L \rightarrow out \times OP$, where $OP = \{swap, push, pop\}$

vs

- Simple compared to what we can do with SDN:

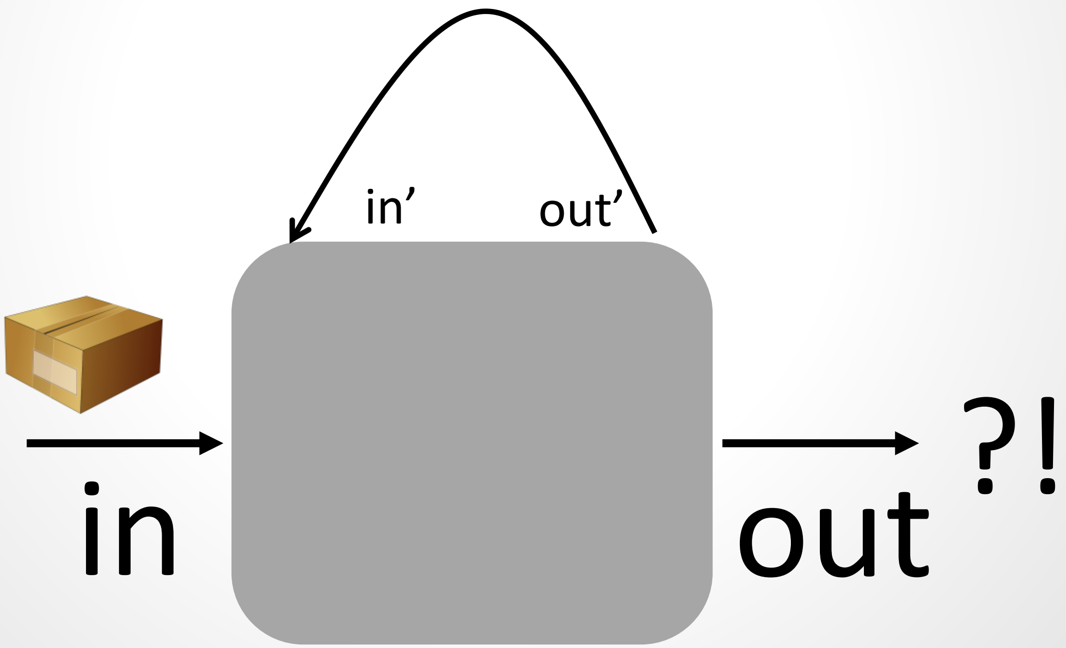
$$in \times L^* \rightarrow out \times L^*$$

Arbitrary string replacement!

Tractability of Verification

Even without failures: reachability test is undecidable in SDN!

Proof: Can emulate a Turing machine.

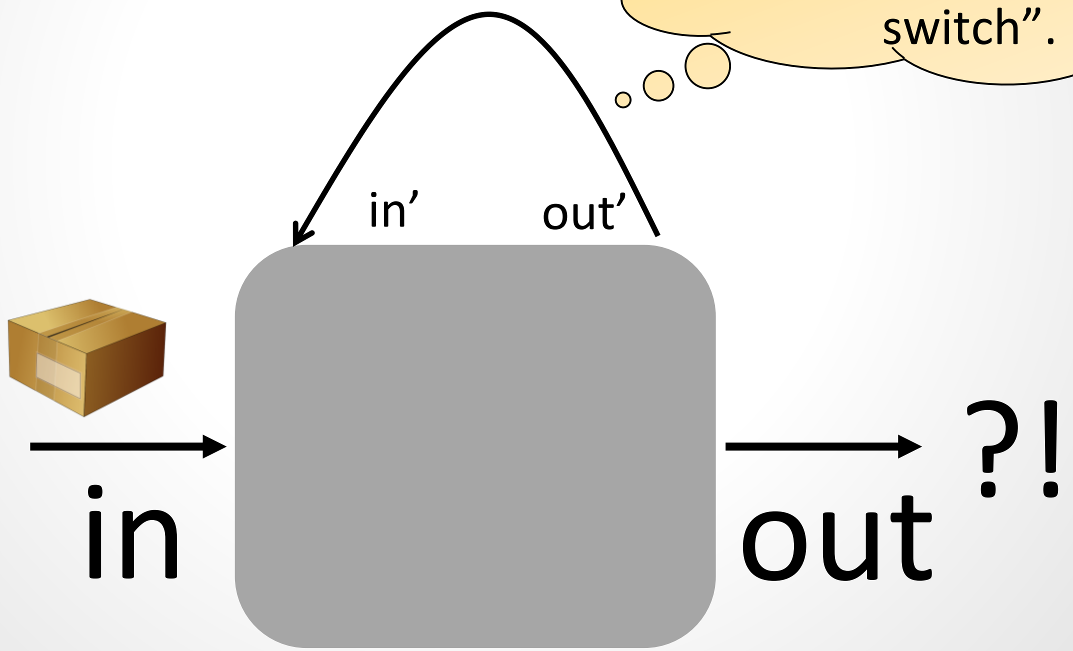


Tractability of Verification

Even without failures: reachability test is undecidable in SDN!

Proof: Can emulate a Turing machine.

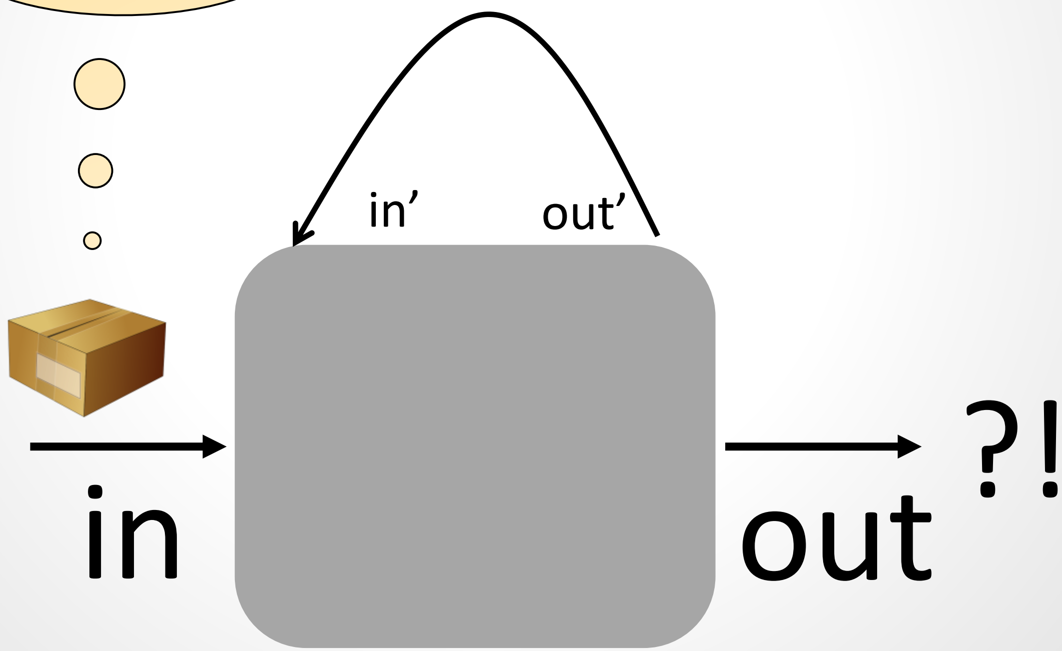
Self-loop: could be replaced by “dummy switch”.



Tractability of Verification

Even … is undecidable in SDN!

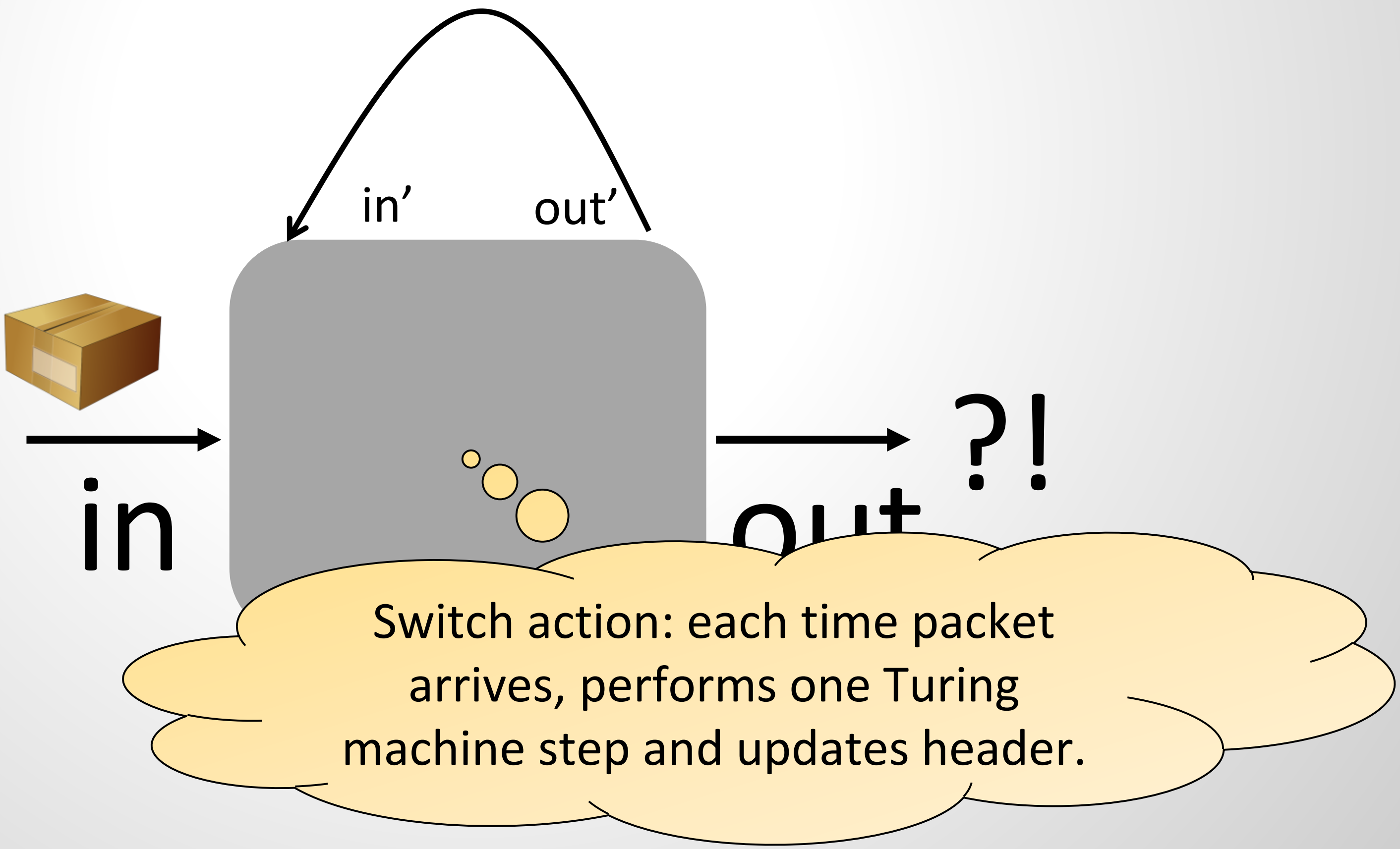
Idea: packet header stores
Turing machine configuration
(tape, head, state).



Tractability of Verification

Even without failures: reachability test is undecidable in SDN!

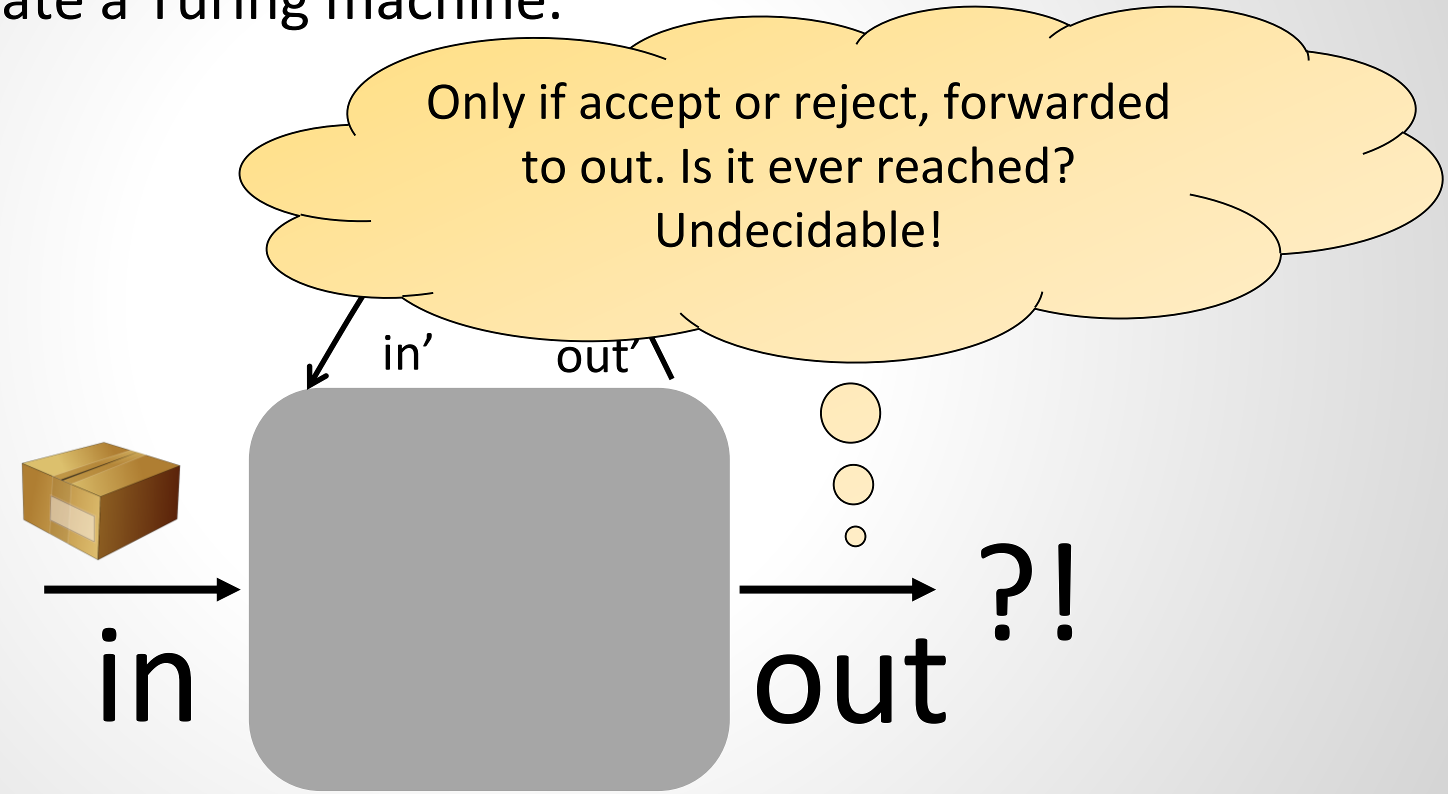
Proof: Can emulate a Turing machine.



Tractability of Verification

Even without failures: reachability test is undecidable in SDN!

Proof: Can emulate a Turing machine.



Further Reading

Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks
Stefan Schmid and Jiri Srba. 37th IEEE Conference on Computer Communications (**INFOCOM**), Honolulu, Hawaii, USA, April 2018.

WNetKAT: A Weighted SDN Programming and Verification Language
Kim G. Larsen, Stefan Schmid, and Bingtian Xue.
20th International Conference on Principles of Distributed Systems (**OPODIS**), Madrid, Spain, December 2016.

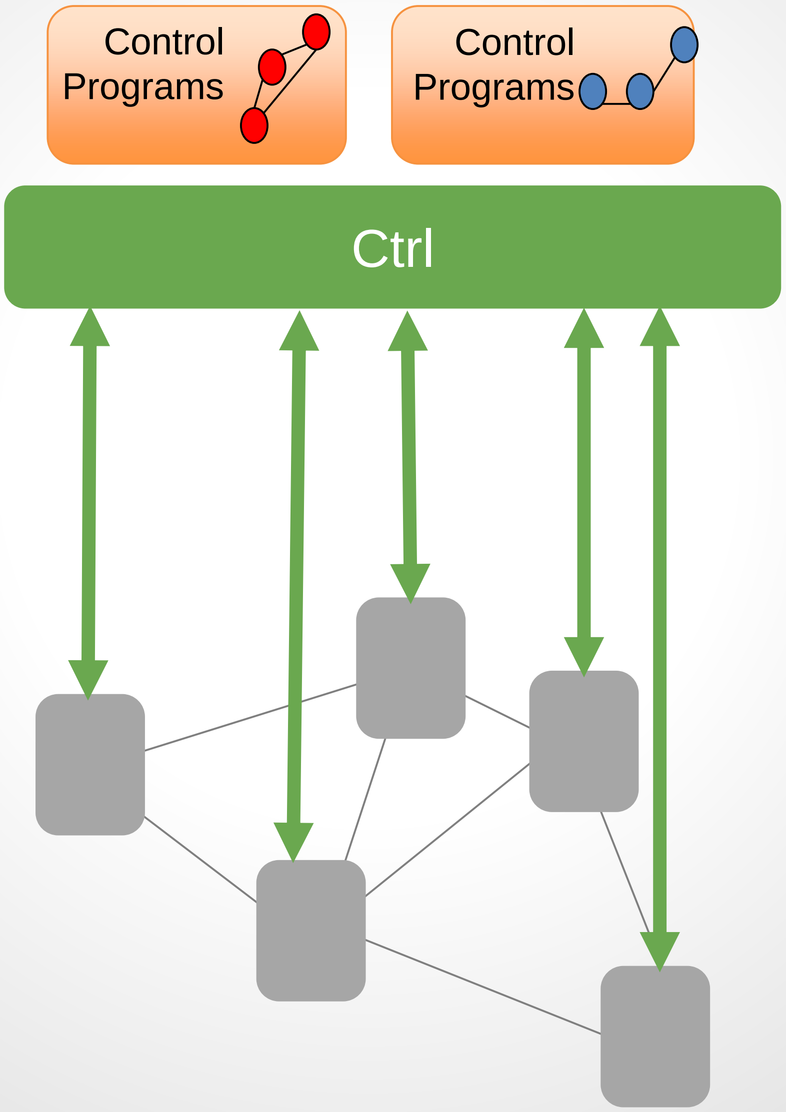
Many Open Research Questions

- Tradeoff expressiveness of rule and verification complexity?
- Is it worth using less general rules so fast (automated) verification is possible?
- Example: MPLS is not hard to verify!
- What about more programmable and stateful dataplanes?

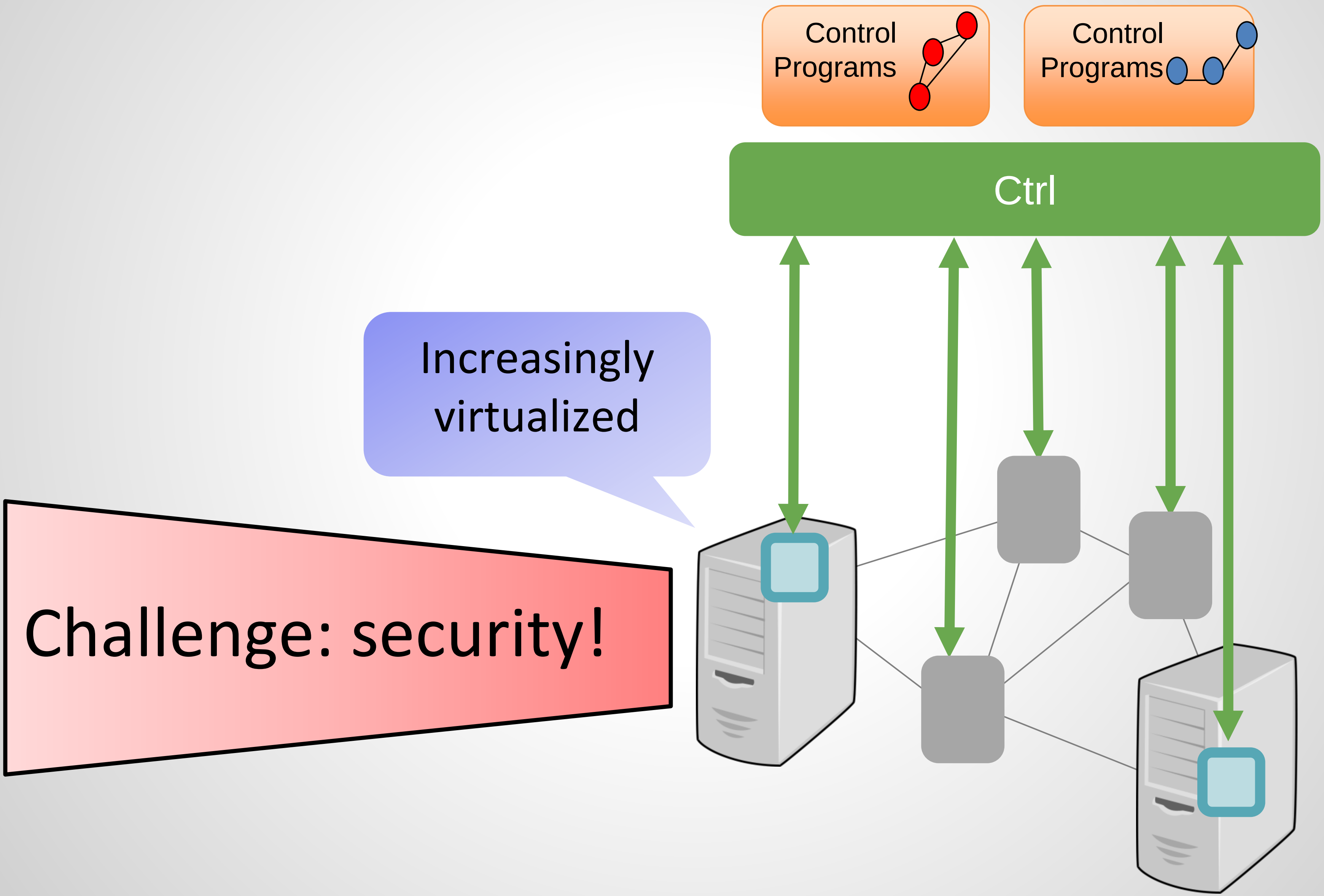
End of Algorithms

Security

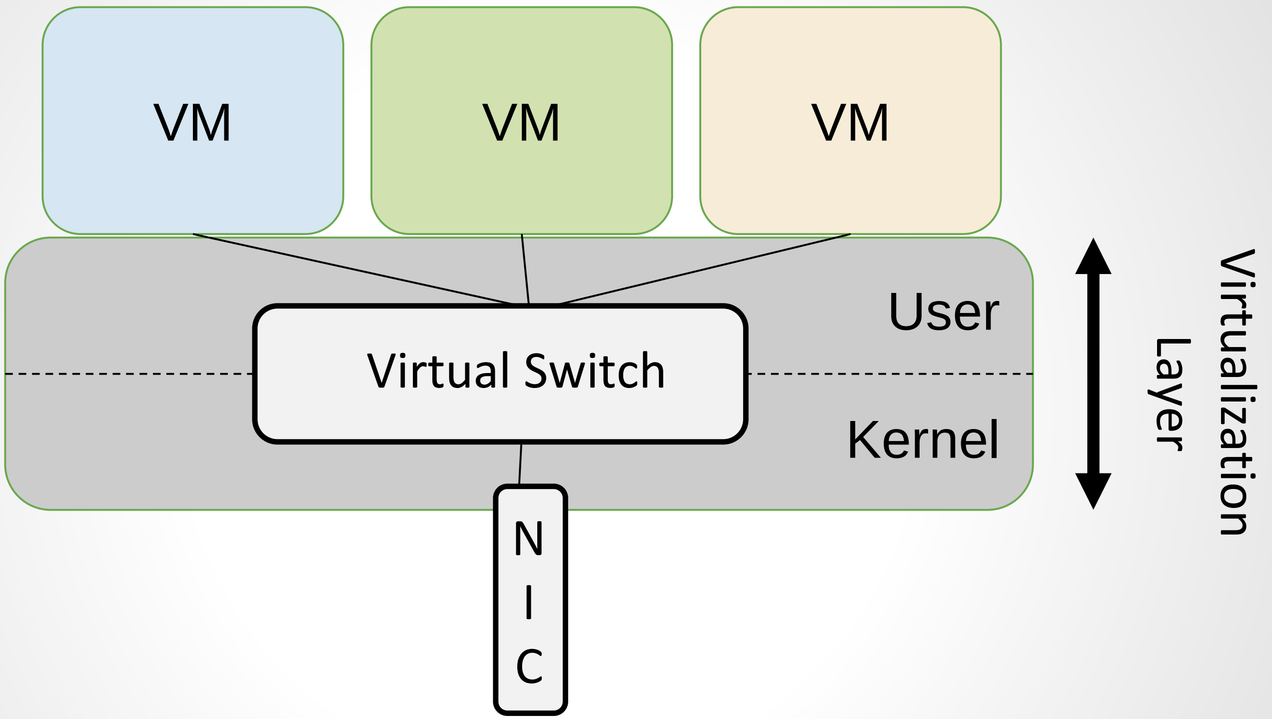
A Mental Model for This Talk



A Mental Model for This Talk



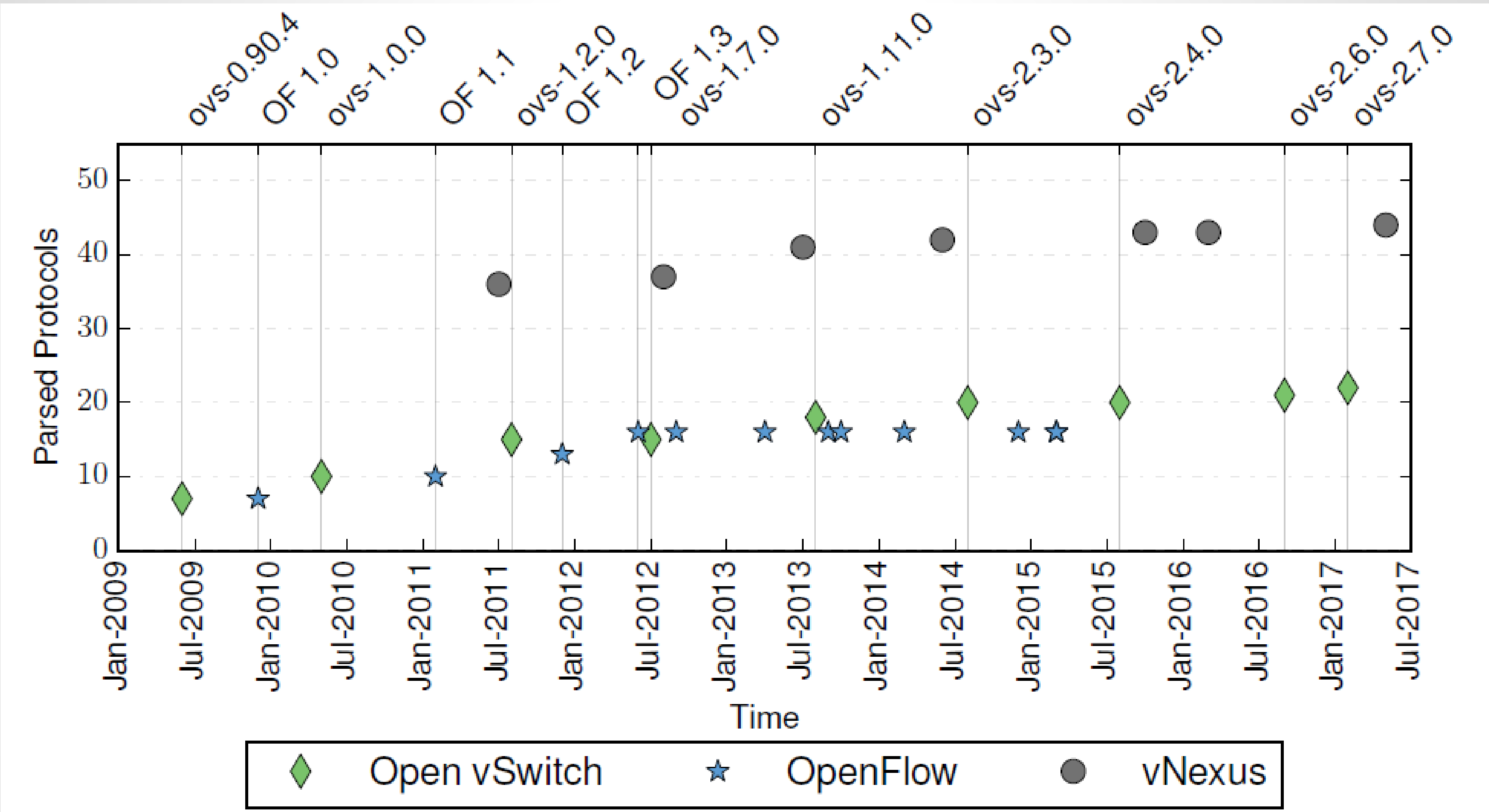
Virtual Switches



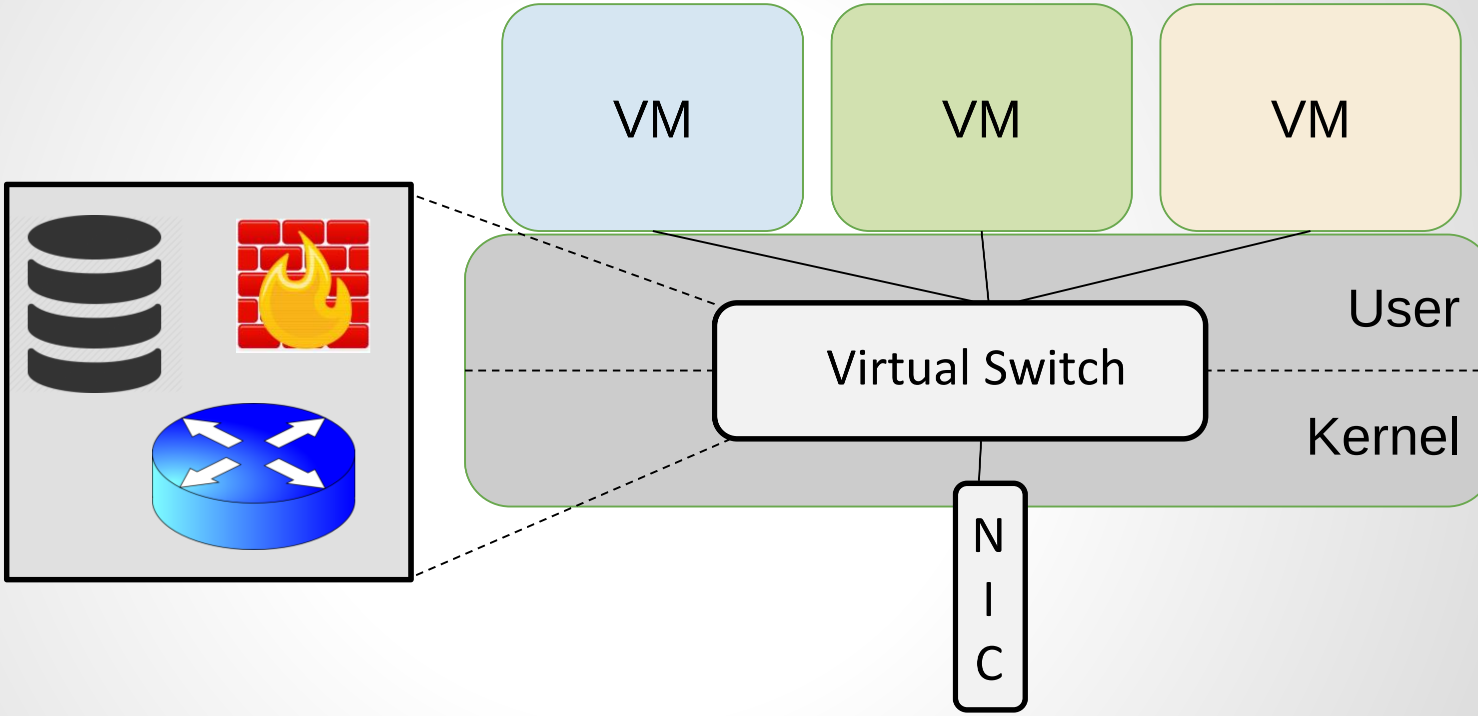
Virtual switches reside in the **server's virtualization layer** (e.g., Xen's Dom0). Goal: provide connectivity and isolation.

Increasing Complexity:
 # Parsed Protocols

Number of parsed high-level protocols constantly increases:



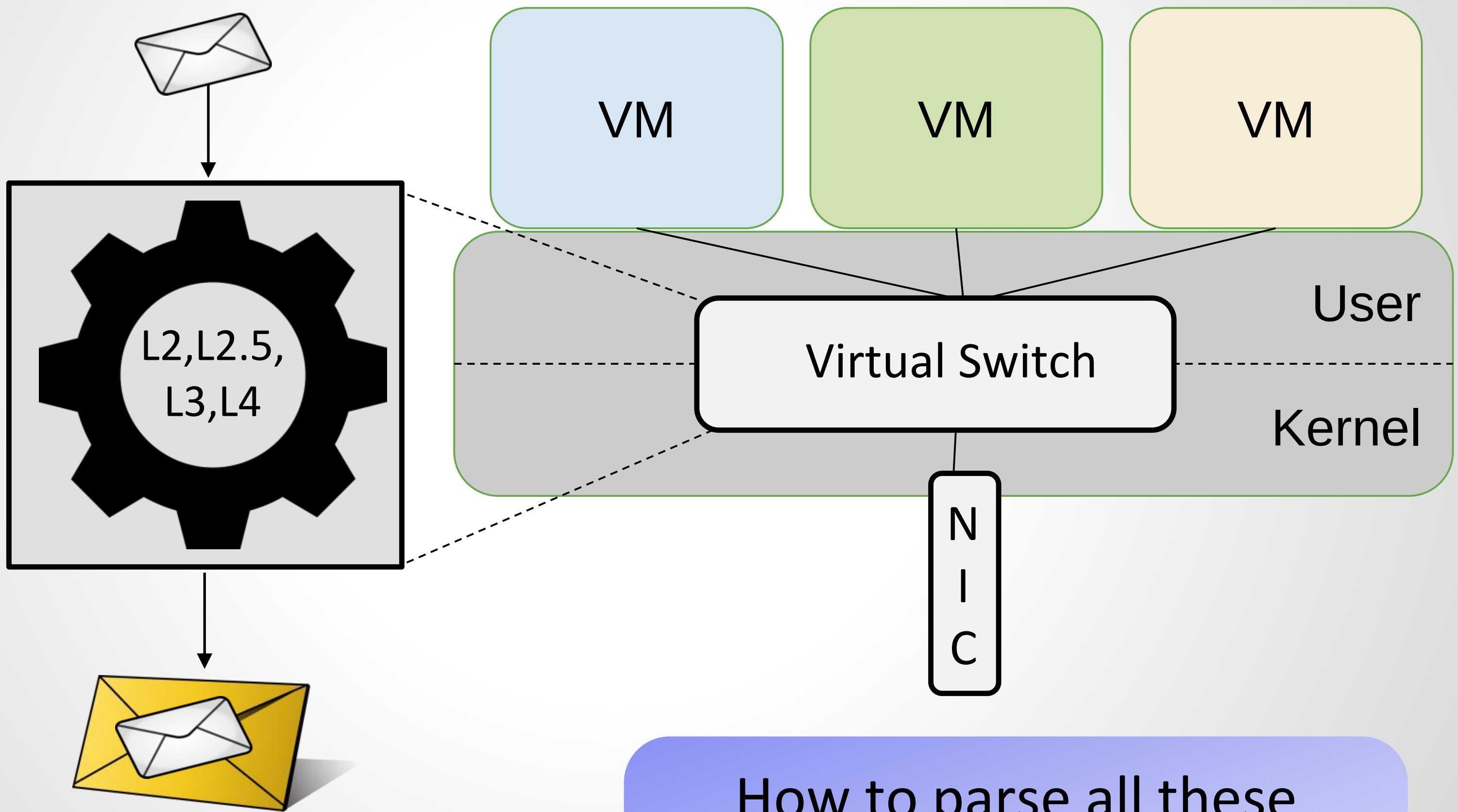
Increasing Complexity:
Introduction of middlebox functionality



Increasing workloads and advancements in network virtualization drive virtual switches to **implement middlebox functions** such as load-balancing, DPI, firewalls, etc.

Increasing Complexity:
Unified Packet Parsing

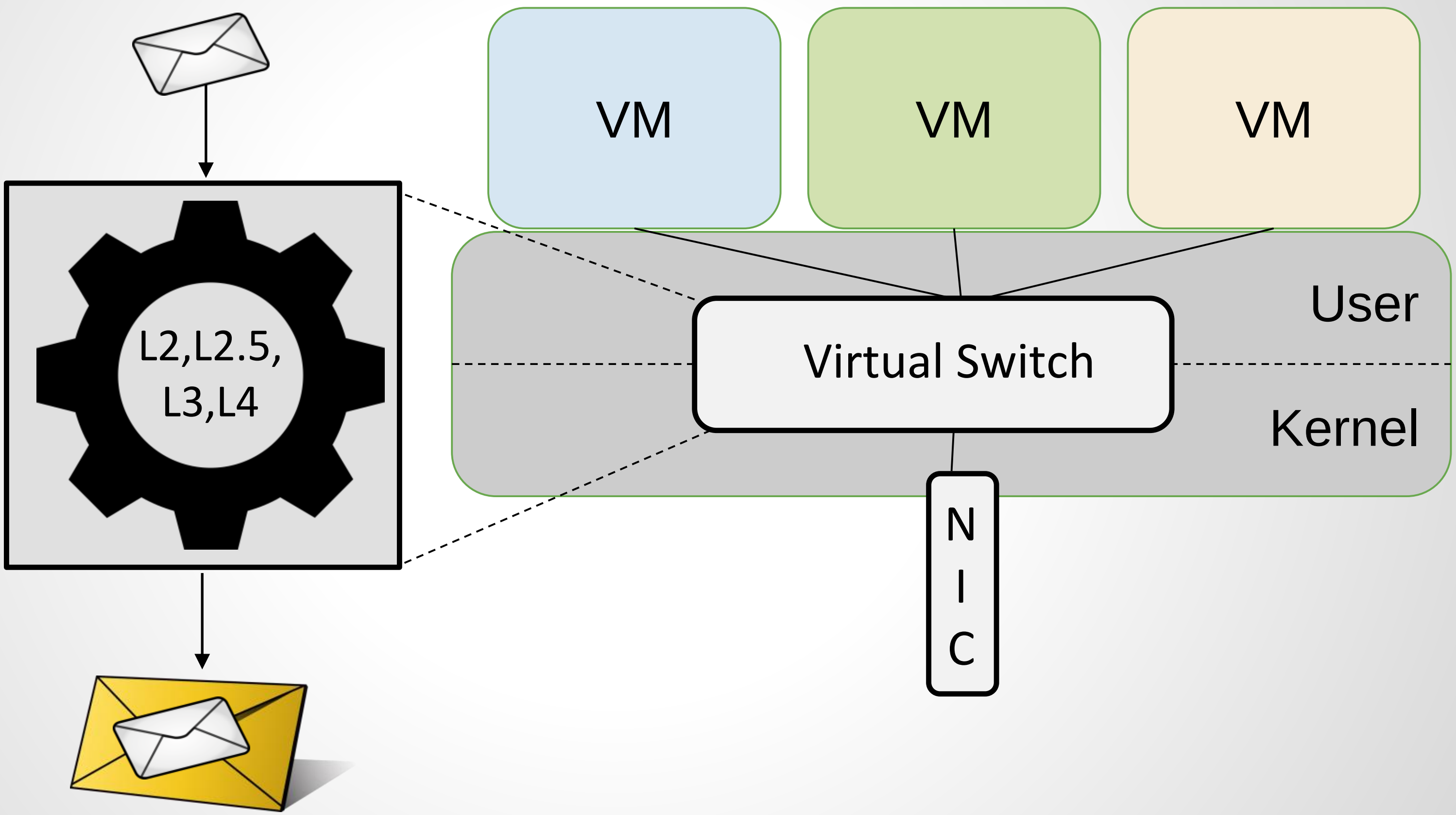
Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



How to parse all these protocols without lowering forwarding performance?!

Increasing Complexity: *Unified Packet Parsing*

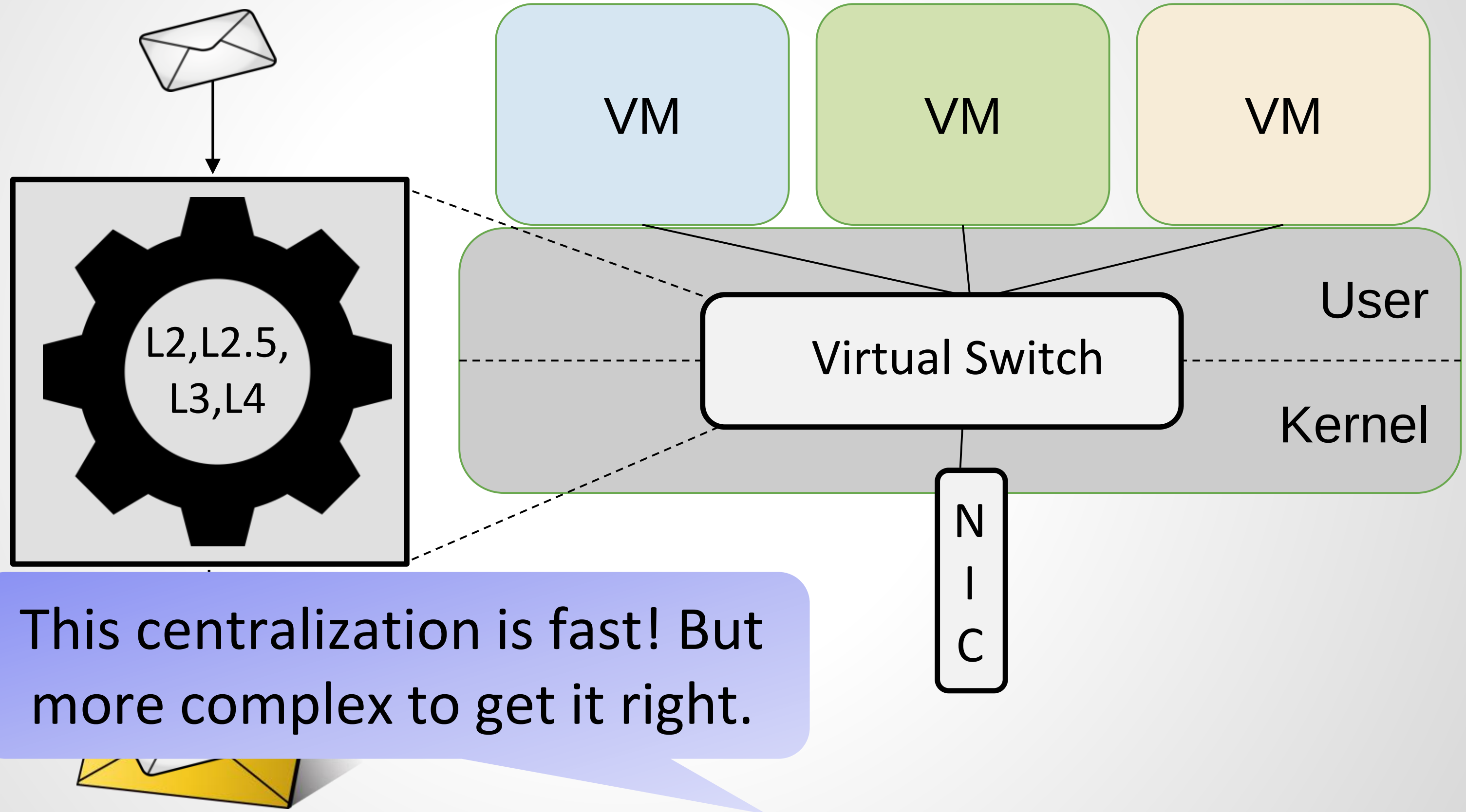
Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



Unified packet parsing allows parse more and more protocols efficiently: in a single pass!

Increasing Complexity: *Unified Packet Parsing*

Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



This centralization is fast! But more complex to get it right.

Unified packet parsing allows parse more and more protocols efficiently: in a single pass!

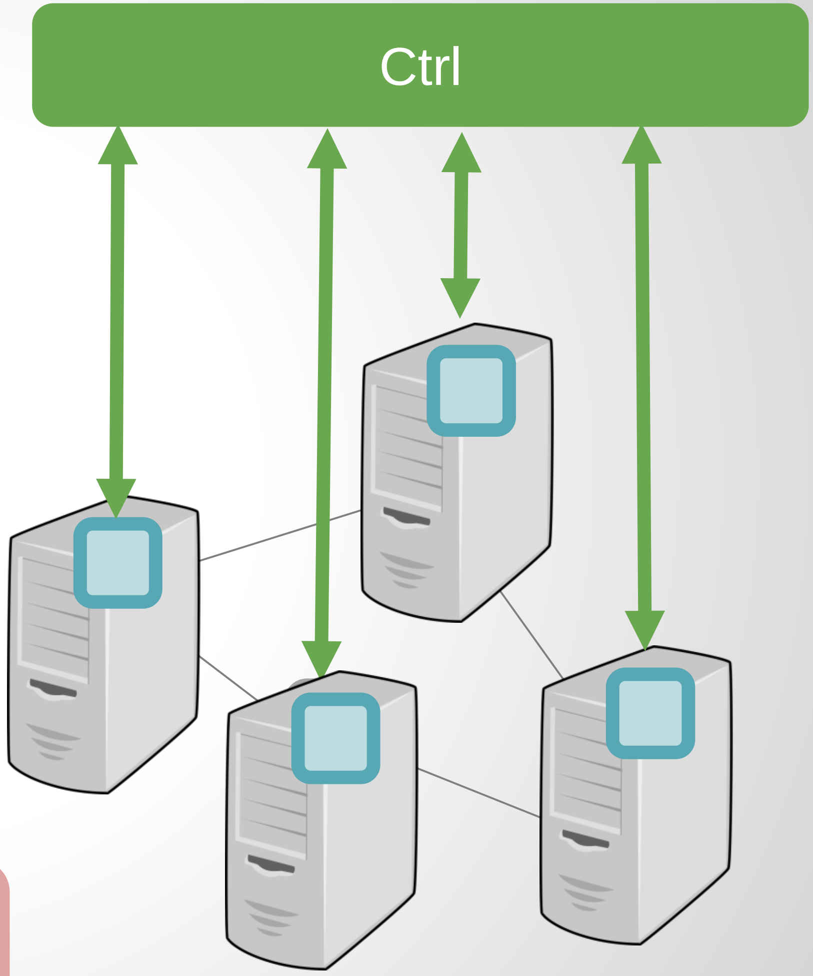
Complexity: The Enemy of Security!

- Data plane security not well-explored (in general, not only virtualized): most security research on control plane

- Two conjectures:

1. Virtual switches increase the attack surface.

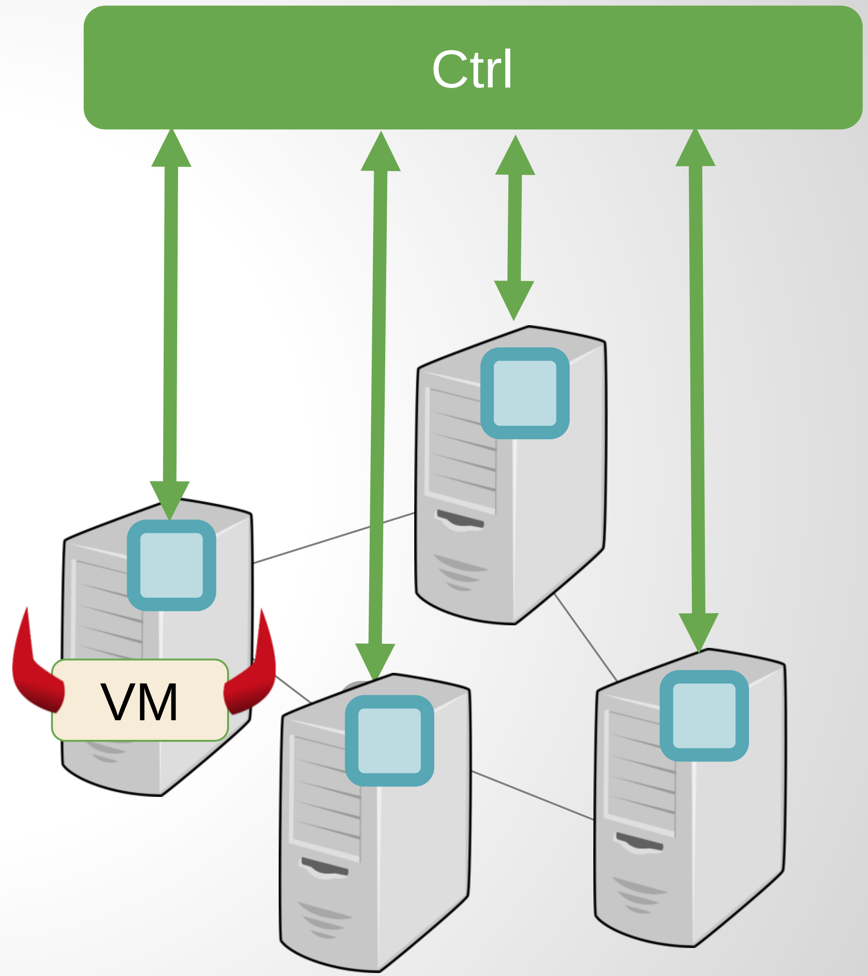
2. Impact of attack larger than with traditional data planes.



The Attack Surface: Closer…

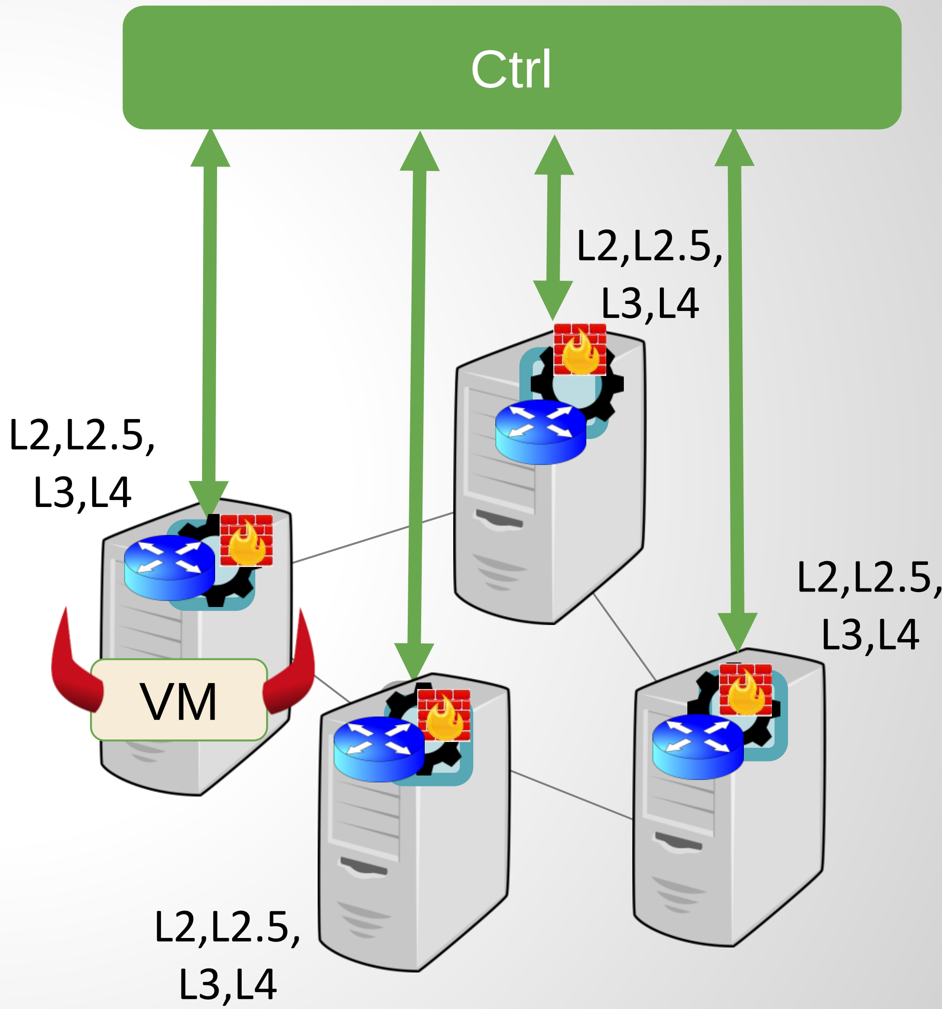
Attack surface becomes closer:

- ❑ Packet parser typically integrated into the code base of virtual switch
- ❑ First component of the virtual switch to process network packets it receives from the network interface
- ❑ May process attacker-controlled packets!



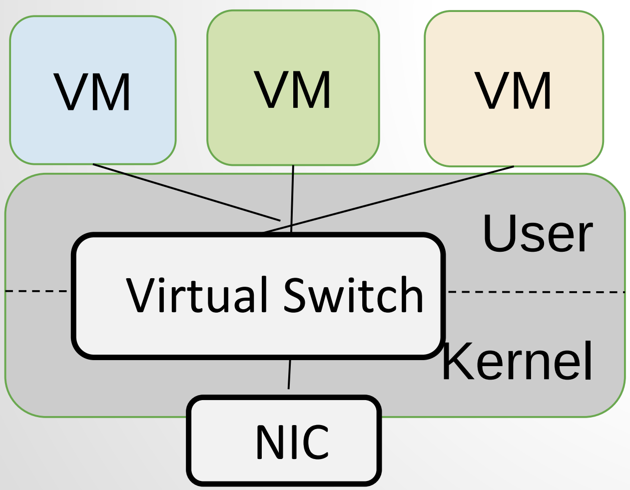
The Attack Surface: … More Complex …

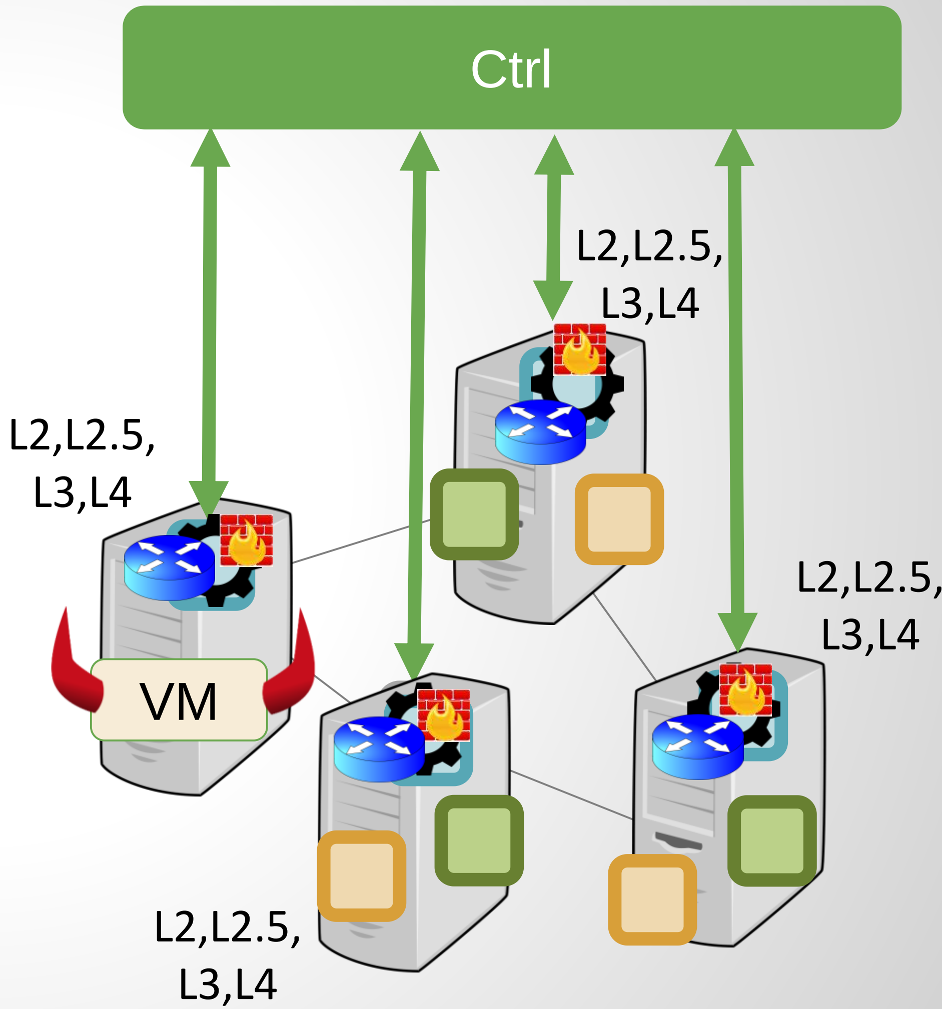
Ethernet	PBB
LLC	IPv6 EXT HDR
VLAN	TUNNEL-ID
MPLS	IPv6 ND
IPv4	IPv6 EXT HDR
ICMPv4	IPv6HOPOPTS
TCP	IPv6ROUTING
UDP	IPv6Fragment
ARP	IPv6DESTOPT
SCTP	IPv6ESP
IPv6	IPv6 AH
ICMPv6	RARP
IPv6 ND	IGMP
GRE	
LISP	
VXLAN	



… Elevated Priviledges and Collocation …

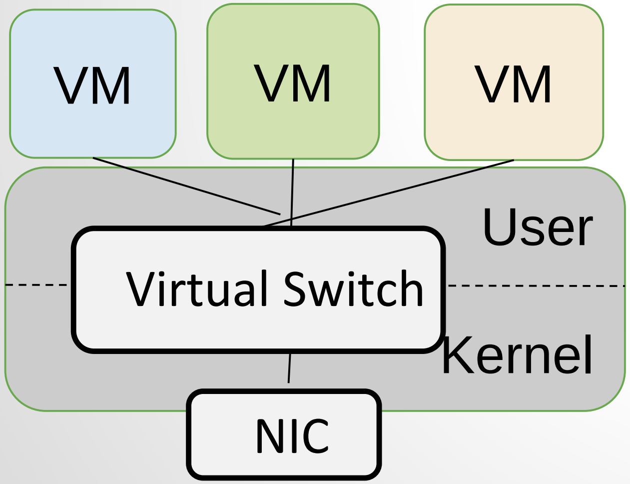
- Collocated (at least partially) with hypervisor's Dom0 kernel space, guest VMs, image management, block storage, identity management, …

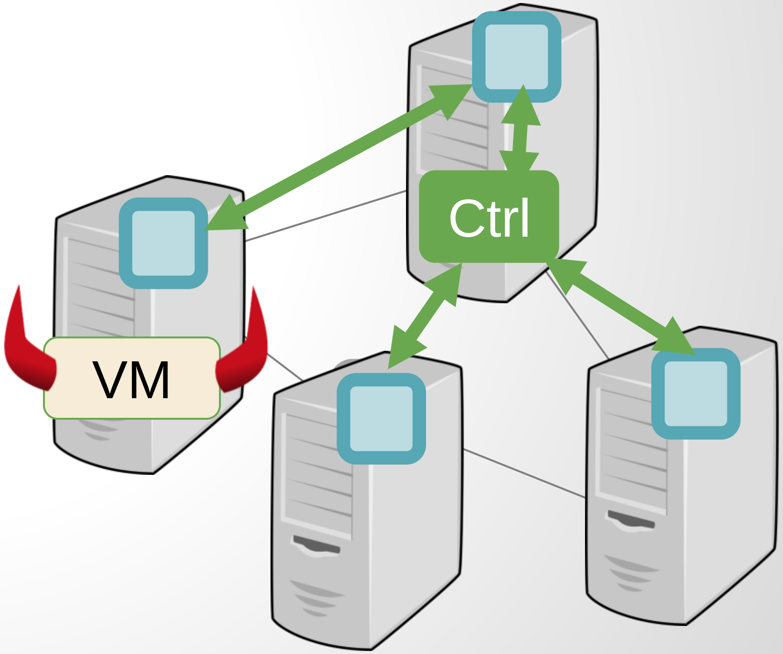




… Elevated Priviledges and Collocation …

- Collocated (at least partially) with hypervisor's Dom0 kernel space, guest VMs, image management, block storage, identity management, …



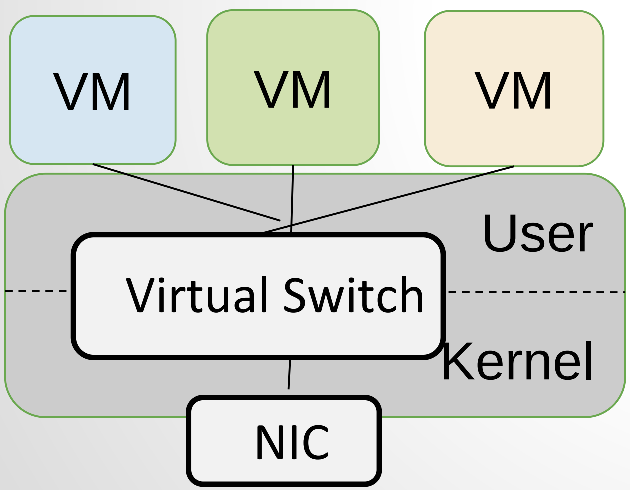


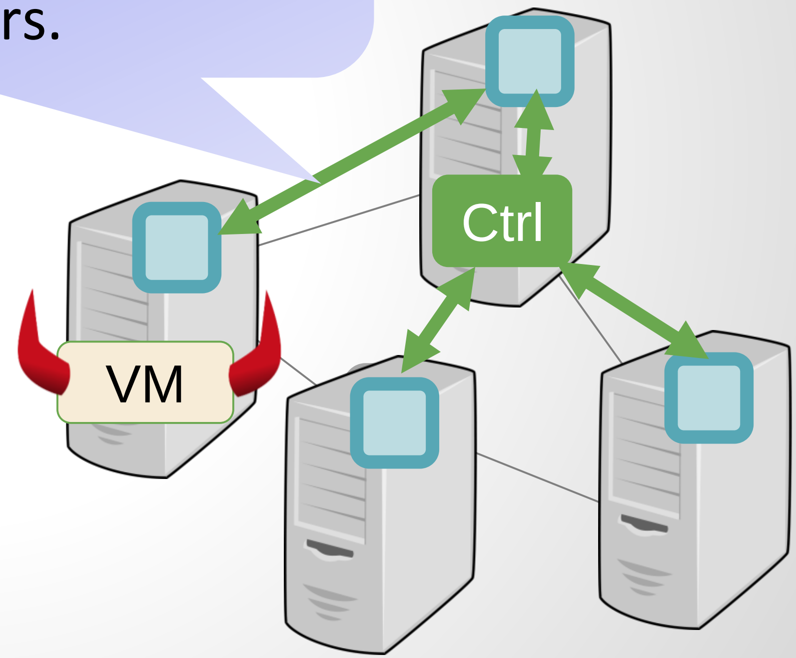
- … the controller itself.

… Centralization …

- Collocate
with hype
space), g
managen
identity m

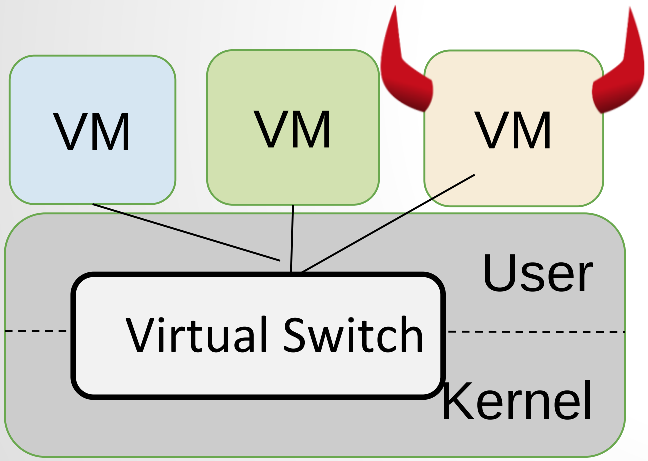
Available communication channels to (SDN/Openstack) controller! Controller needs to be reachable from all servers.





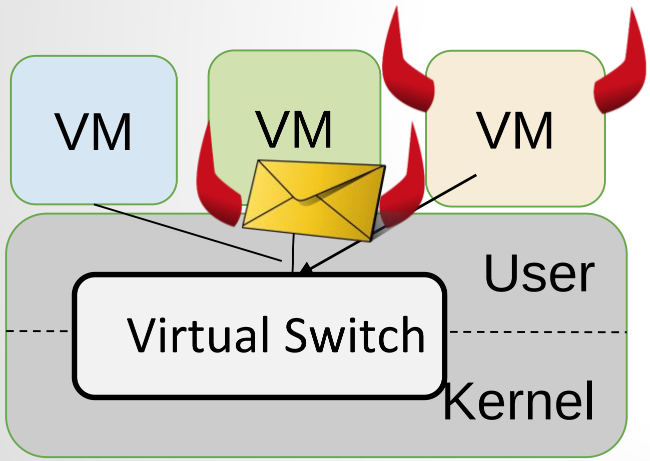
- … the controller itself.

Larger Impact: Case Study OVS



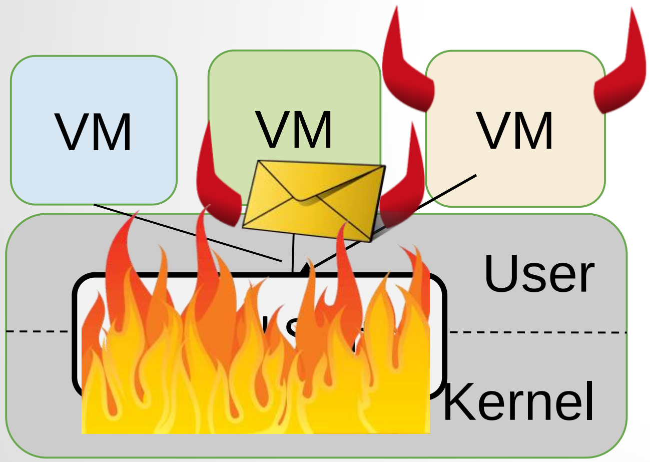
1. Rent a VM in the cloud (**cheap**)

Larger Impact: Case Study OVS



2. Send **malformed MPLS packet** to virtual switch (**unified parser** parses label stack packet beyond the threshold)

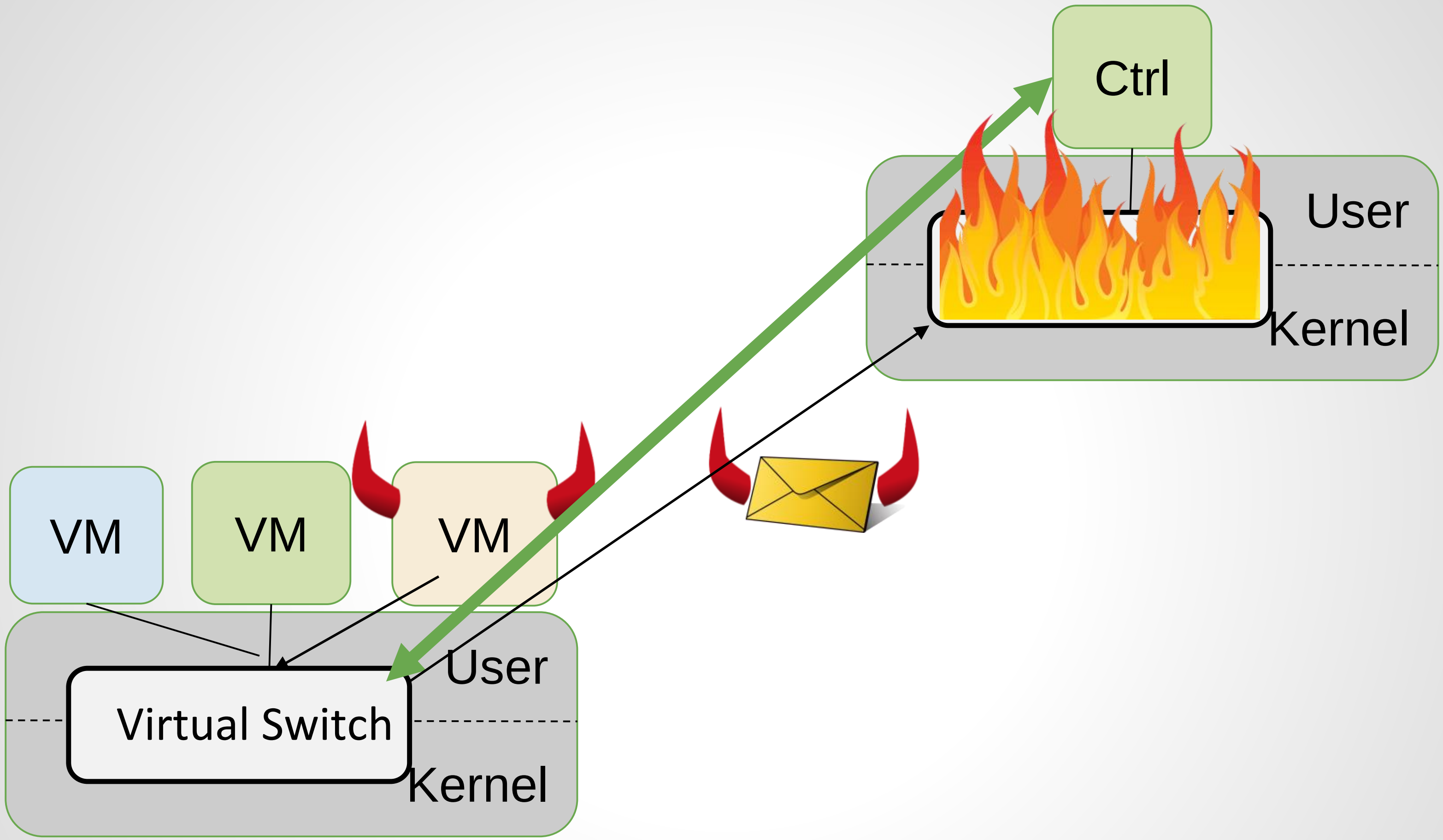
Larger Impact: Case Study OVS



3. **Stack buffer overflow** in (unified) MPLS parsing code:

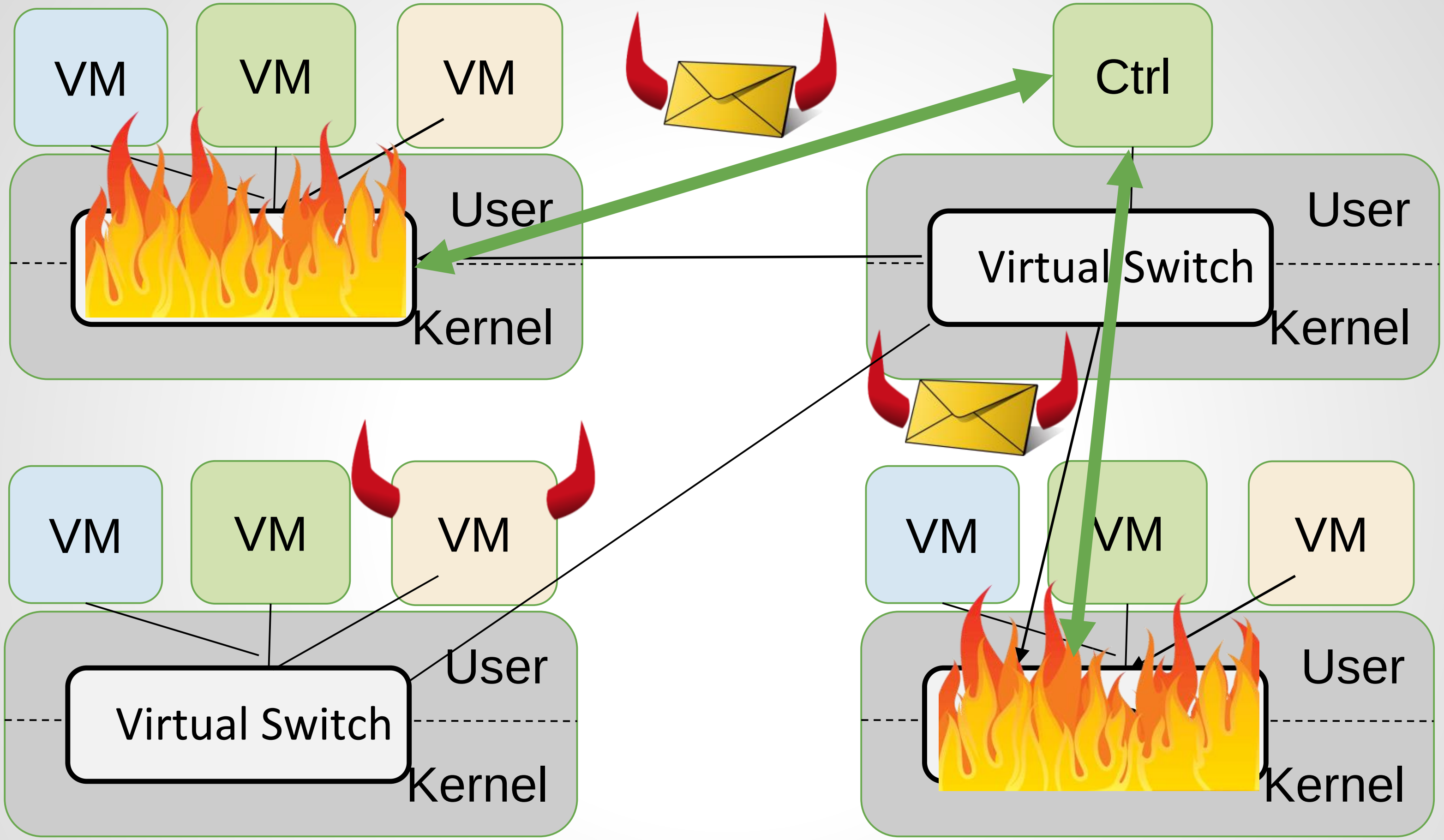
enables remote code execution

Larger Impact: Case Study OVS



4. Send malformed packet to server (virtual switch) where controller is located (use existing communication channel)

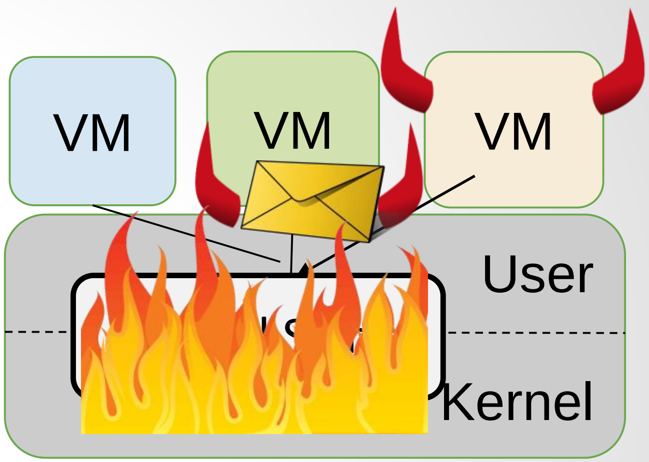
Larger Impact: Case Study OVS



5. Spread

A New Threat Model

- Limited skills required
  - Use standard fuzzer to find crashes
  - Construct malformed packet
  - Build ROP chain
- Limited resources
  - rent a VM in the cloud
- No physical access needed



No need to be a state-level attacker to compromise the dataplane (and beyond)!

Similar problems in NFV: need even more complex parsing/processing. And are often built on top of OvS.

Countermeasures

- Software countermeasures already exist
  - but come at overhead
- Better designs
  - Virtualize dataplane components: decouple them from hypervisor?
  - Remote attestation for OvS Flow Tables?
  - Control plane communication firewalls?
  - …

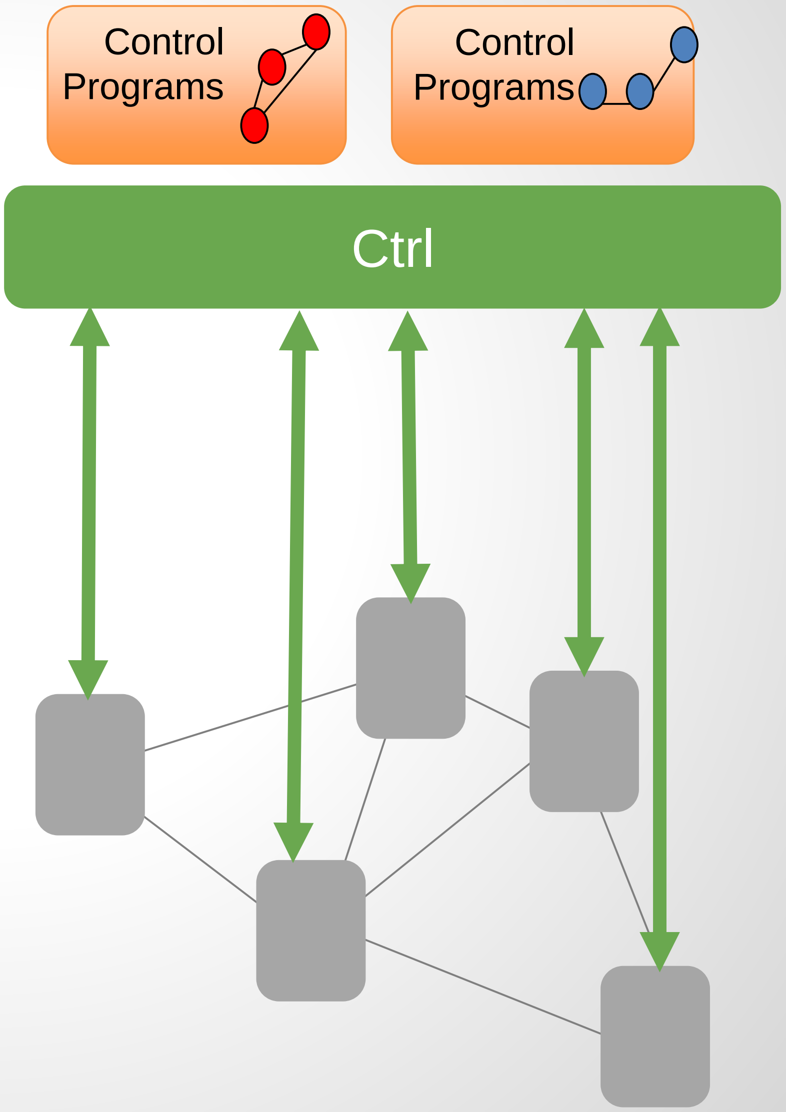
Further Reading

The vAMP Attack: Taking Control of Cloud Systems via the Unified Packet Parser
Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid.
9th ACM Cloud Computing Security Workshop (**CCSW**), collocated with ACM CCS, Dallas, Texas, USA, November 2017.

Reigns to the Cloud: Compromising Cloud Systems via the Data Plane
Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid.
ArXiv Technical Report, October 2016.

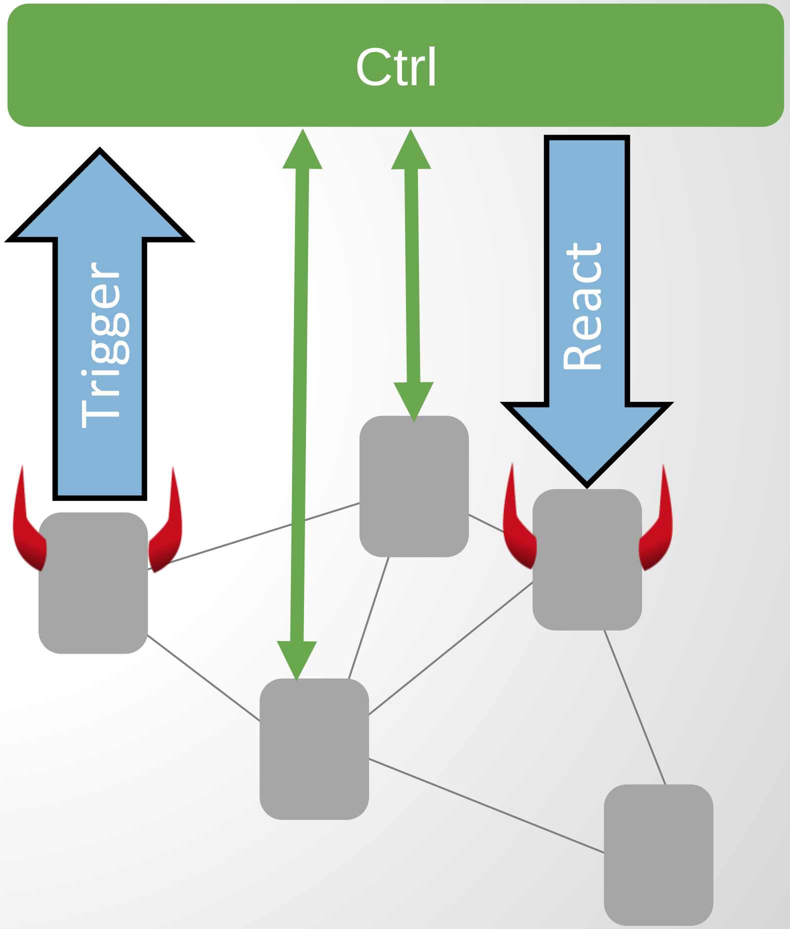
A Mental Model for This Talk

Challenge: centralization!



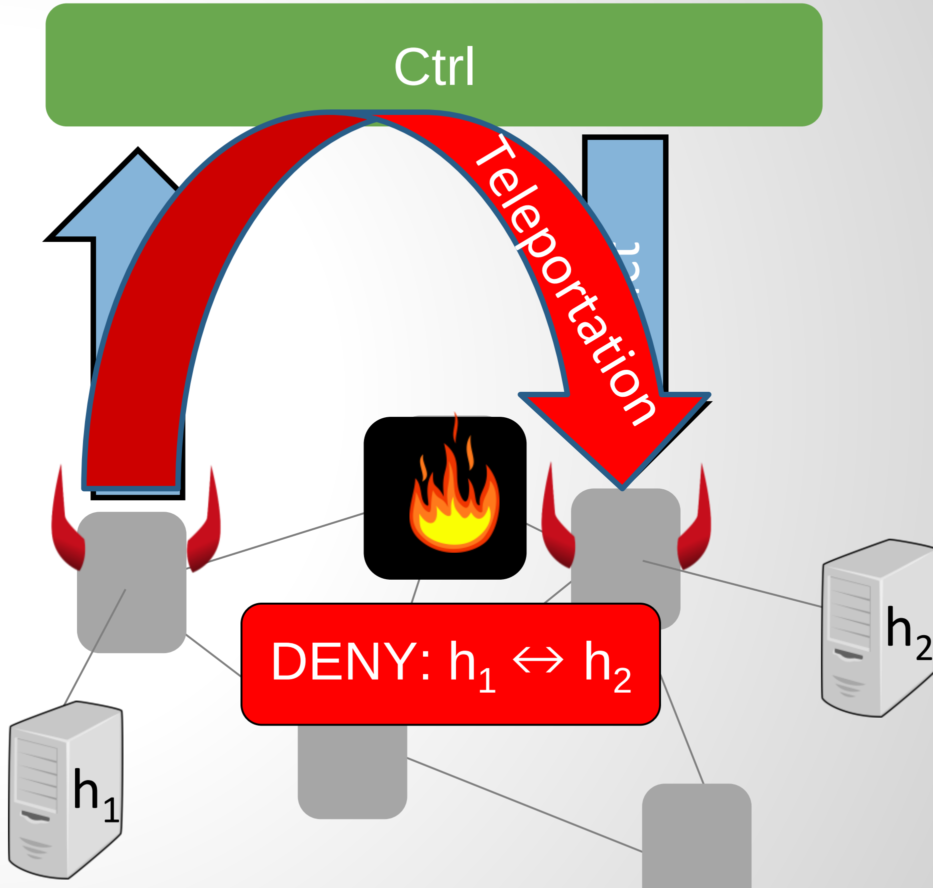
Central Controller Can Increase Attack Surface: E.g., May Be Exploited For Covert Communication

- Controllers react to switch events (packet-ins, link failures, etc.) for MAC learning, support mobility, VM migration, failover, etc.
- Reaction: send flowmods, packet-outs, performing path-paving…
- Triggering such events may be exploited for (covert) communication or even port scans, etc. even in presence of firewall/IDS/…



Teleportation

- May be used to bypass firewall
- Not easy to detect:
  - Traffic follows normal pattern of control communication, indirectly via controller
  - Teleportation channel is inside (encrypted) OpenFlow channel
- Need e.g., to correlate packet-ins, packet-outs, flow-mods, etc.

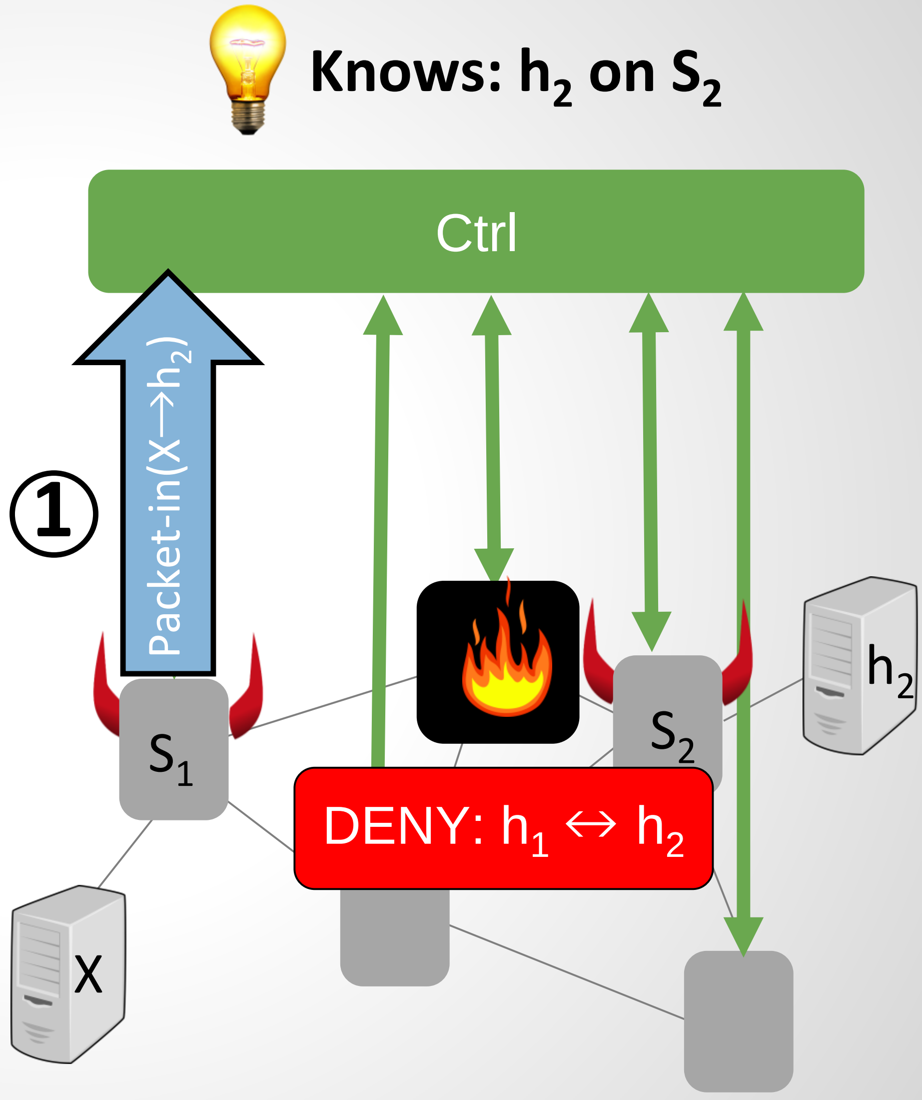


Teleportation: Out-of-Band Forwarding

- E.g., exploiting ONOS Intent Reactive Forwarding (ifwd)

- By default, ifwd installs host-to-host connectivity when receiving a packet-in for which no flows exist (using path-pave technique)

① **Packet-in**



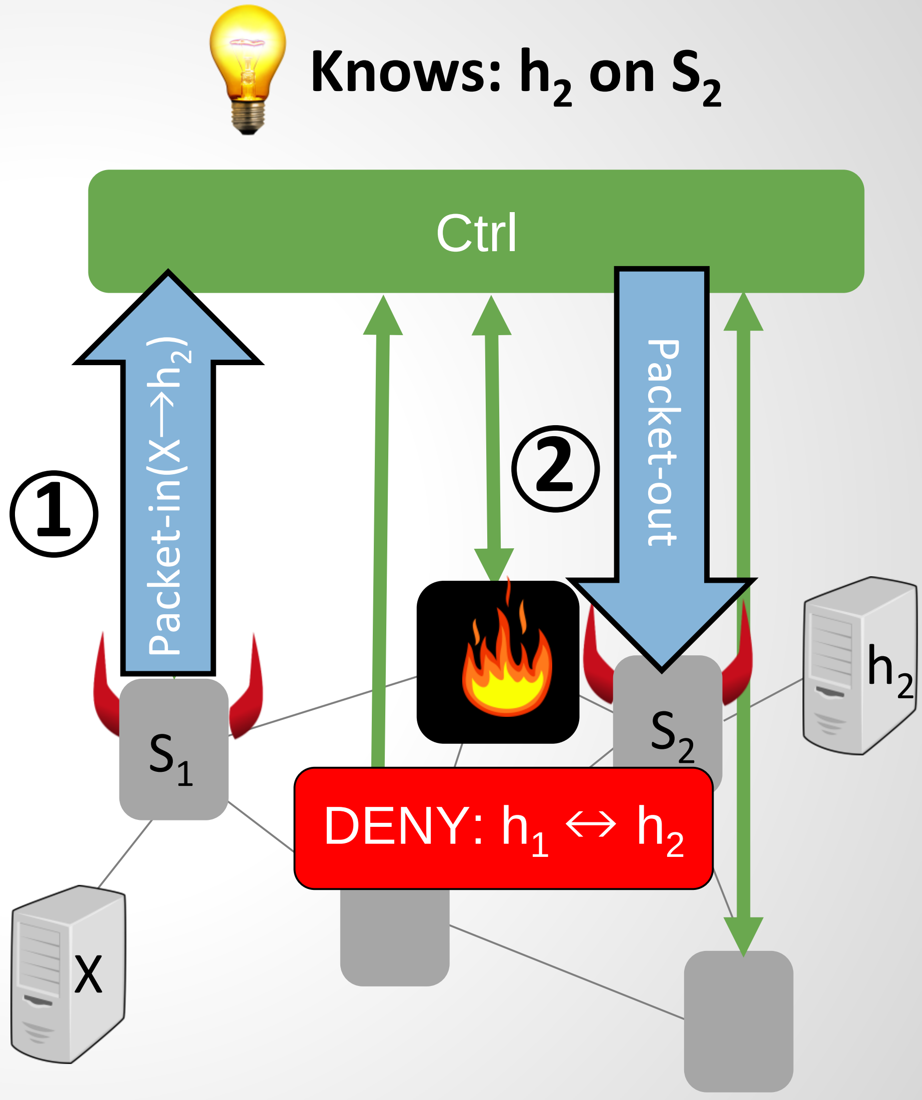
Teleportation: Out-of-Band Forwarding

- E.g., exploiting ONOS Intent Reactive Forwarding (ifwd)

- By default, ifwd installs host-to-host connectivity when receiving a packet-in for which no flows exist (using path-pave technique)

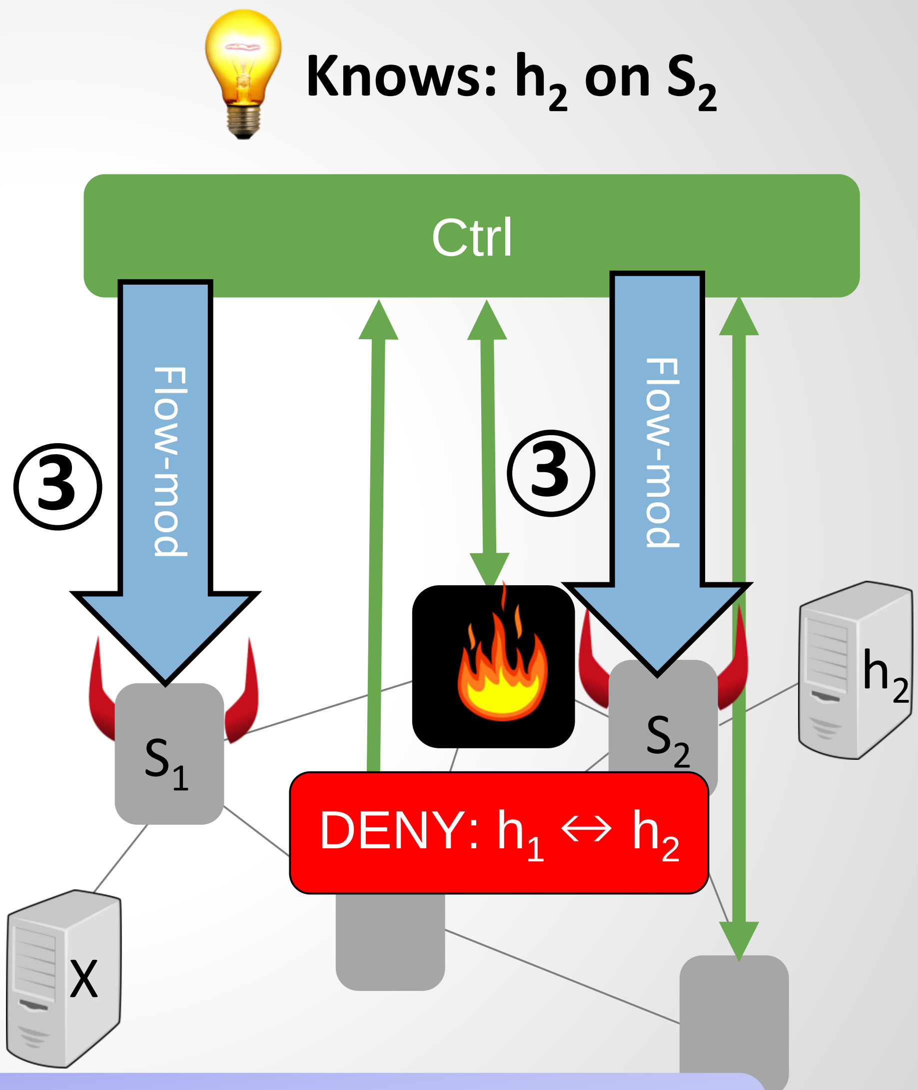
① **Packet-in**

② **Packet-out**



Teleportation: Out-of-Band Forwarding

- E.g., exploiting ONOS Intent Reactive Forwarding (ifwd)
- By default, ifwd installs host-to-host connectivity when receiving a packet-in for which no flows exist (using path-pave technique)



① **Packet-in**

② **Packet-out**

③ **Flow-mod**

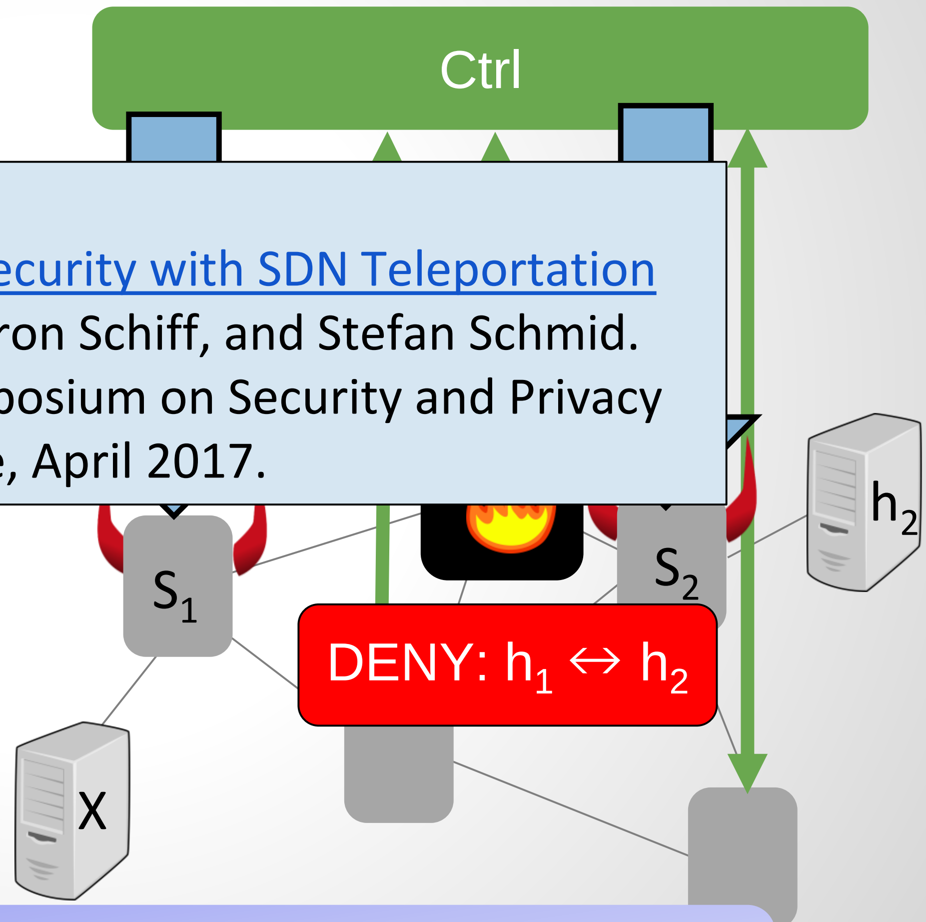
Establish path through firewall: no more packet-ins, blocked. (But could use another MAC address next time.)

Teleportation: Out-of-Band Forwarding

Knows: h_2 on S_2

- E.g., exploiting ONOS Intent Reactive Forwarding (ifwd)
- By defau
host con
a packet-
exist (usi

Further reading:
[Outsmarting Network Security with SDN Teleportation]
Kashyap Thimmaraju, Liron Schiff, and Stefan Schmid.
2nd IEEE European Symposium on Security and Privacy (**EuroS&P**), Paris, France, April 2017.



① **Packet-in**

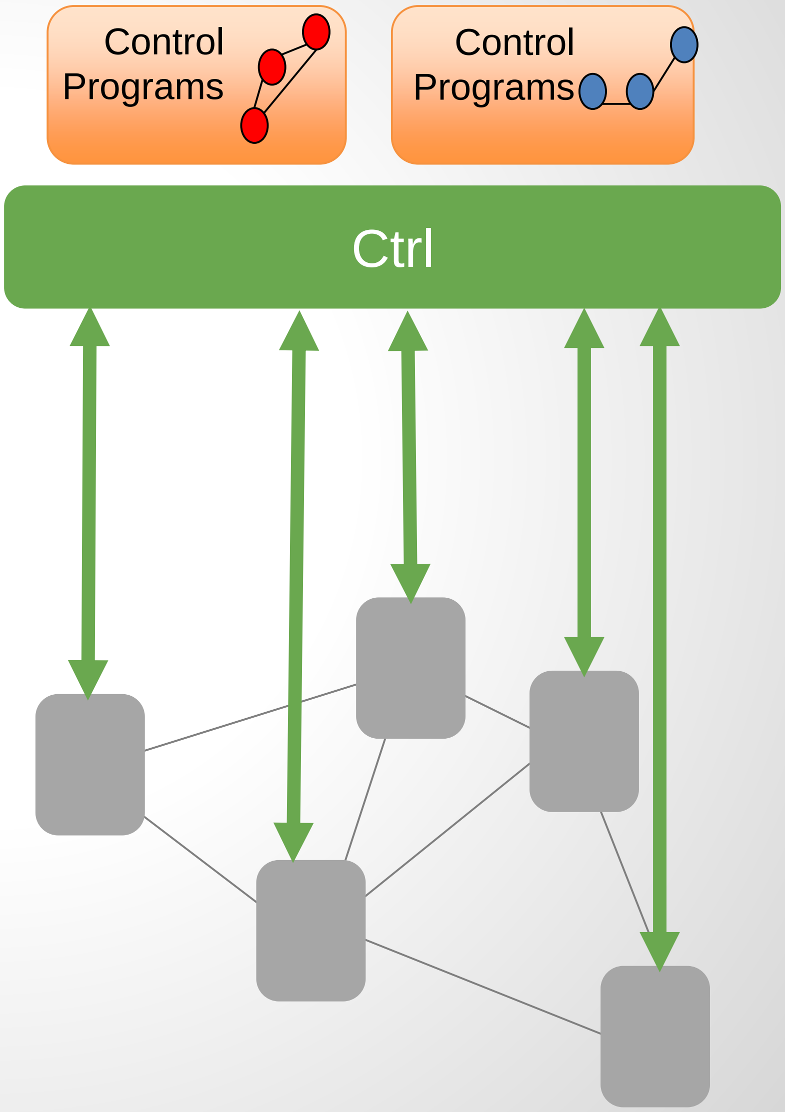
② **Packet-out**

③ **Flow-mod**

Establish path through firewall: no more packet-ins, blocked. (But could use another MAC address next time.)

Let’s talk about opportunities!

Opportunity: centralization!



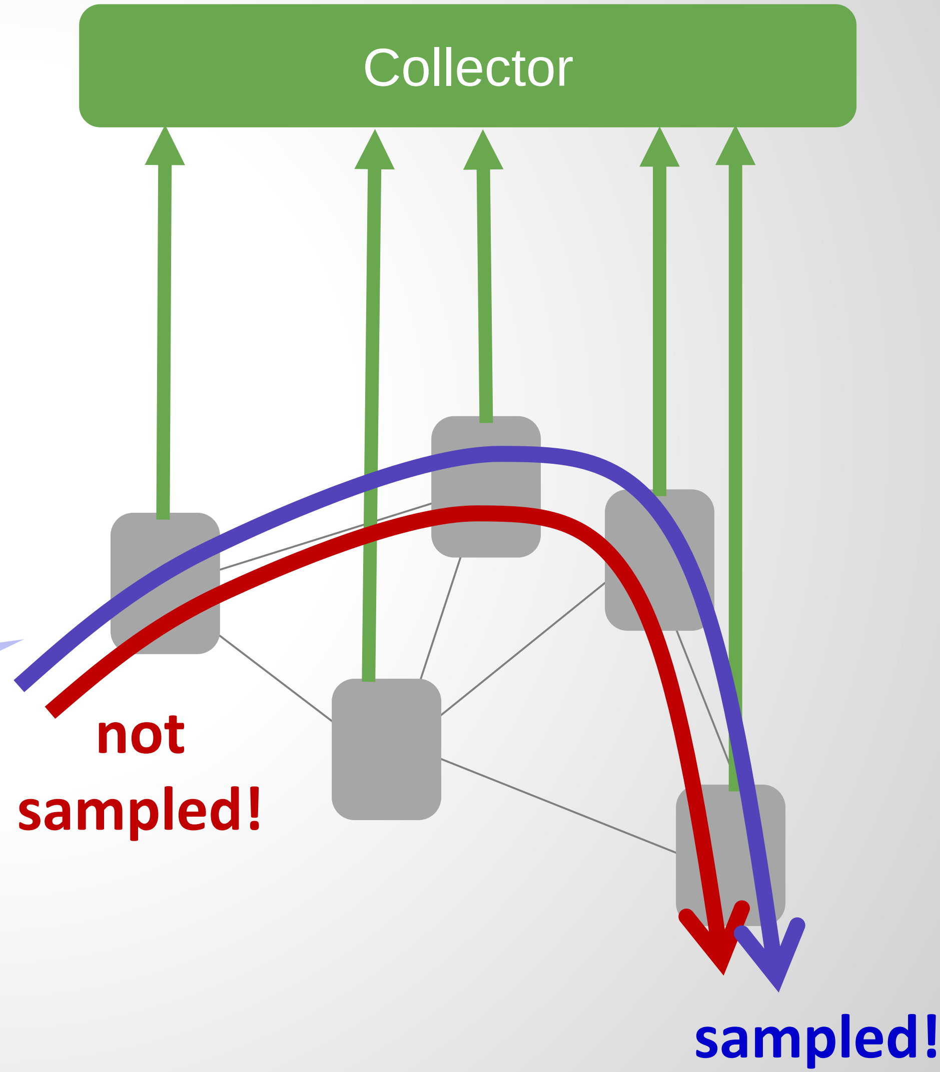
Example: Adversarial Trajectory Sampling

Trajectory Sampling

- ❑ Method to infer packet routes
- ❑ Low overhead, direct and passive measurement 

Principle: Sample subset of packets consistently (e.g., hash over immutable fields)

Packets sampled either at all or no location!



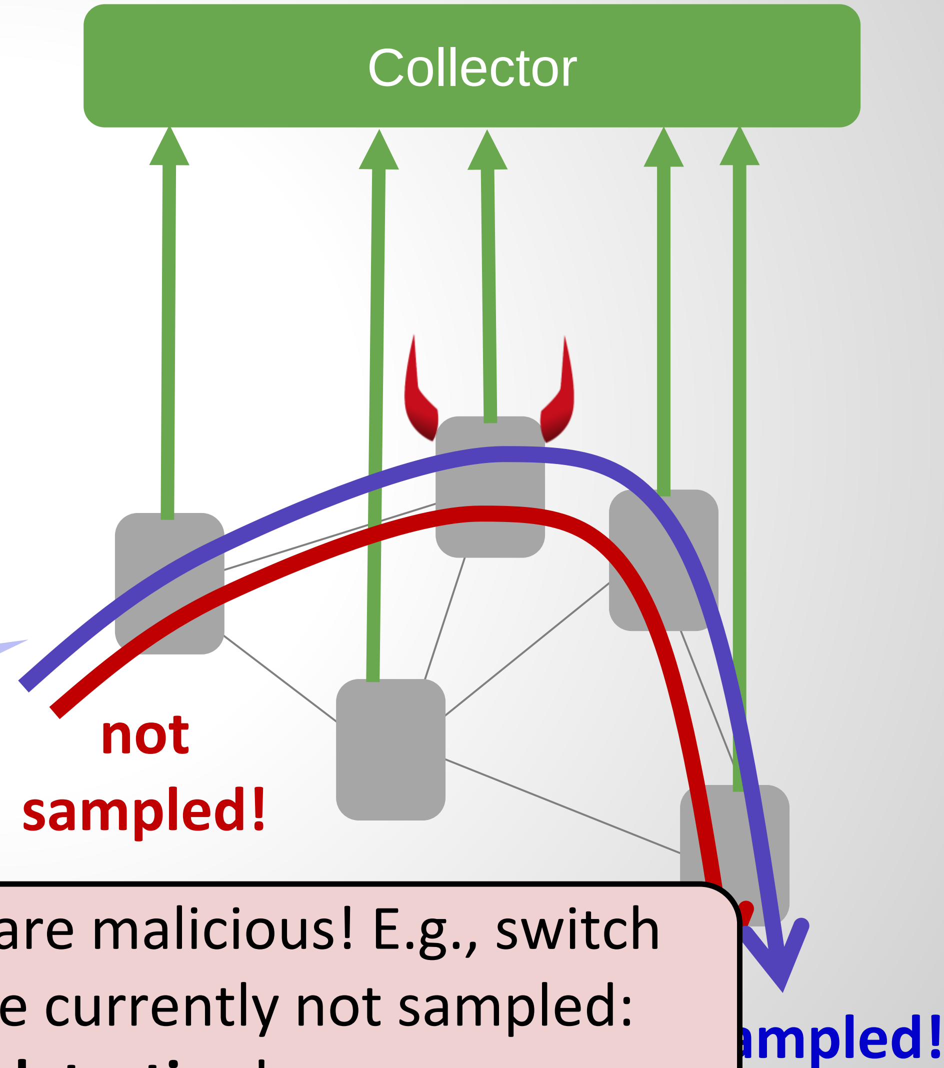
Example: Adversarial Trajectory Sampling

Trajectory Sampling

- Method to infer packet routes
- Low overhead, direct and passive measurement

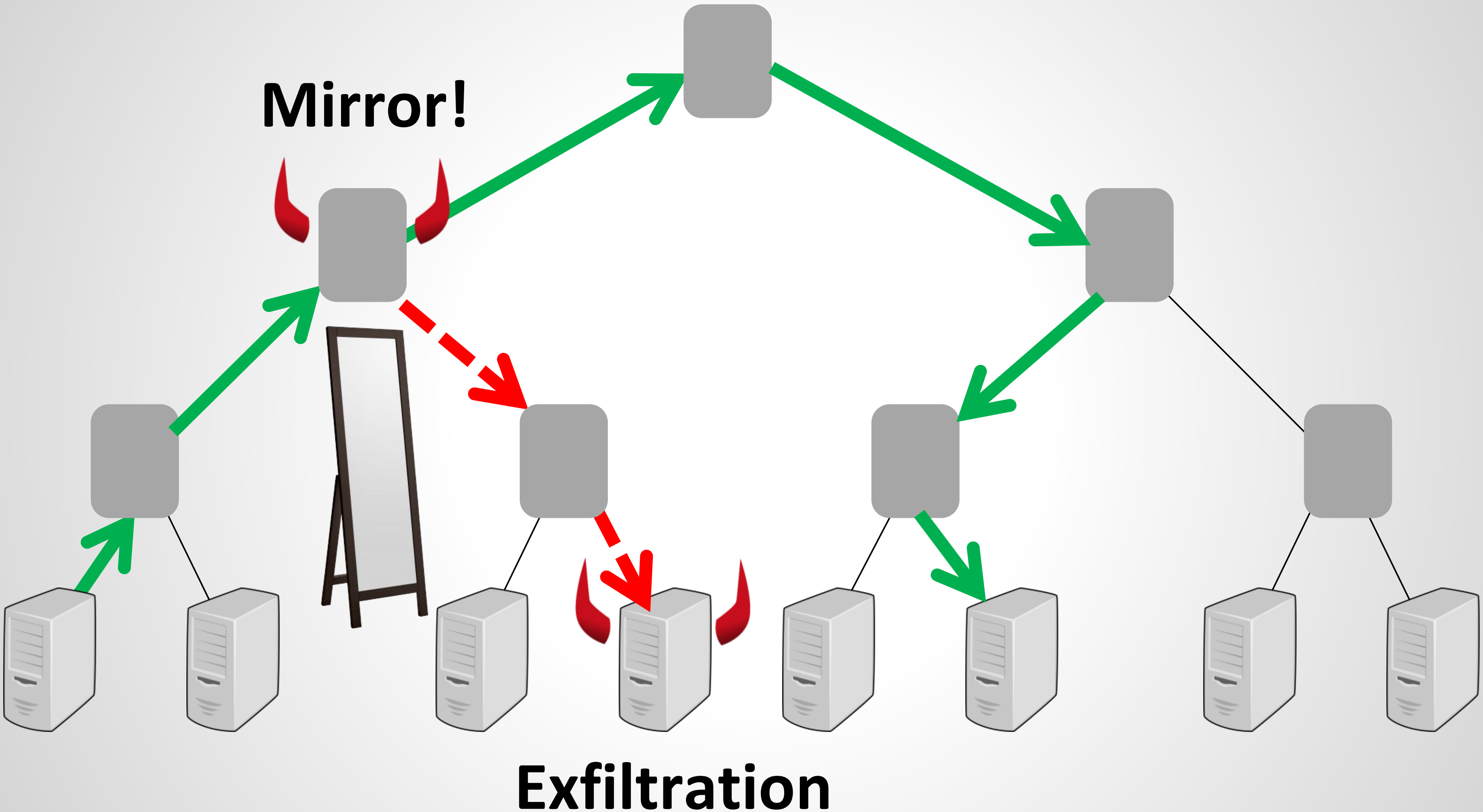
Principle: Sample subset of packets consistently (e.g., hash over immutable fields)

Packets sampled either at all or no location!

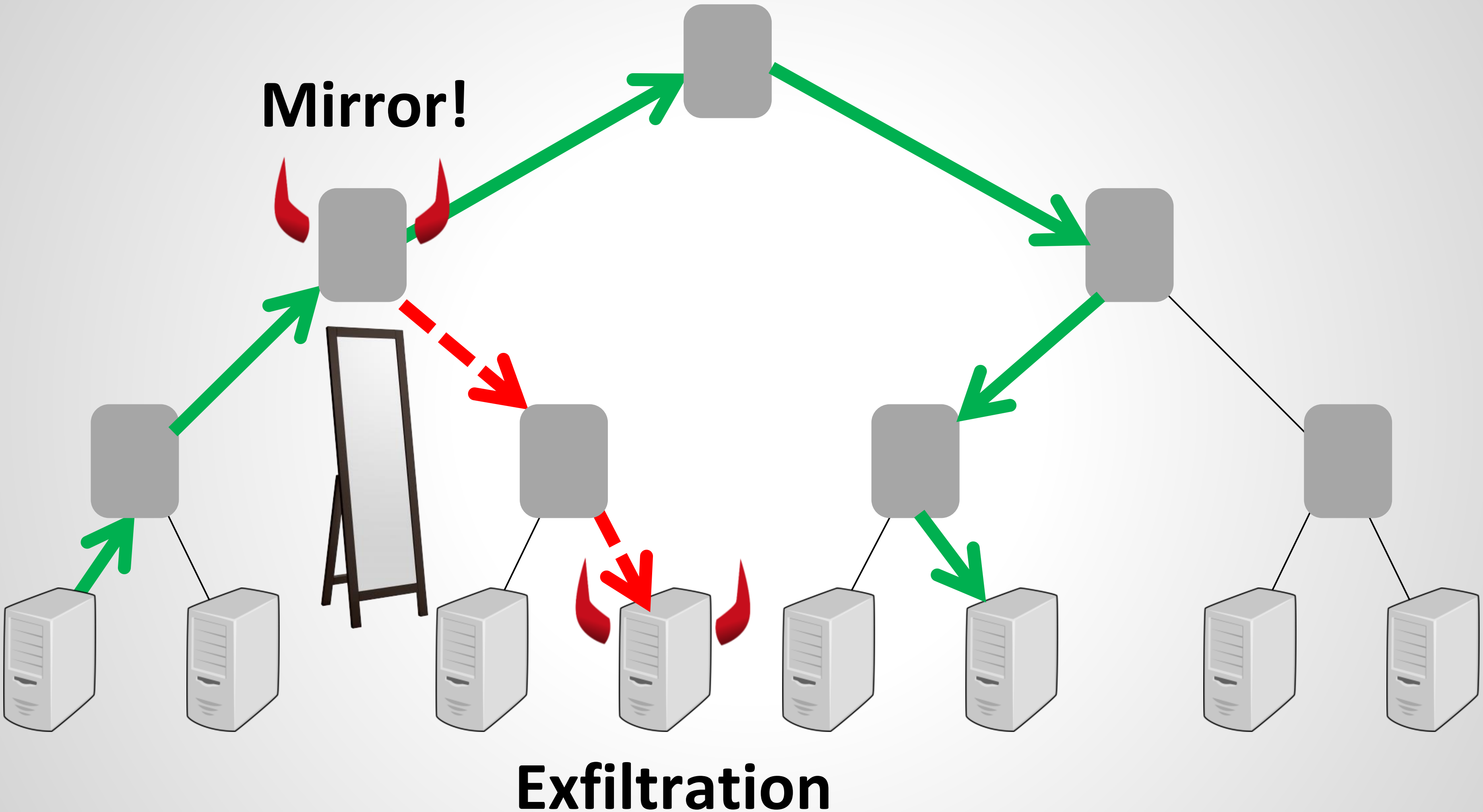


But: Fails when switches are malicious! E.g., switch knows which headers are currently not sampled: **no risk of detection**!

A Malicious Switch Could Do Many Things…

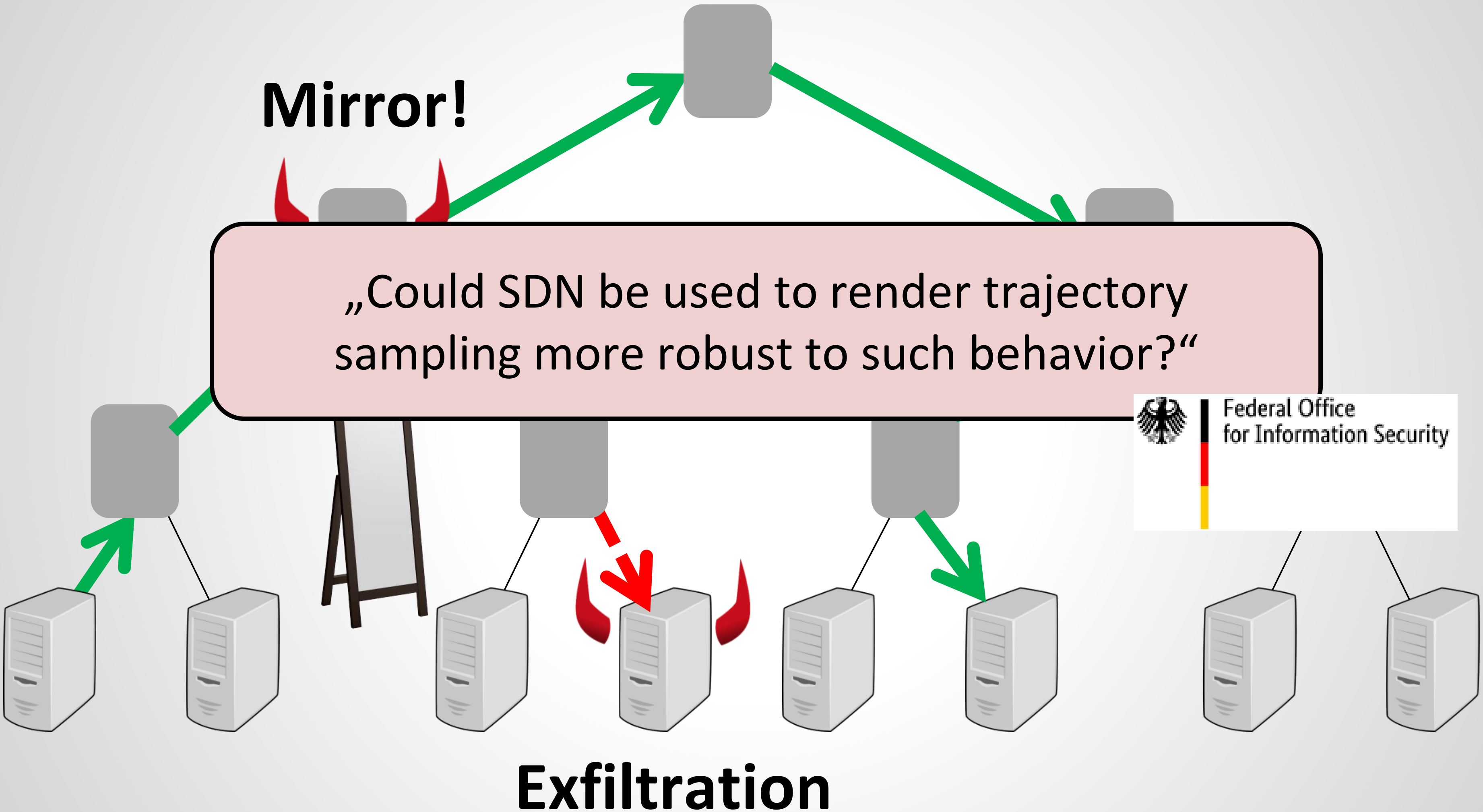


A Malicious Switch Could Do Many Things…



Also: drop packets (that are currently not sampled), inject packets, change VLAN tag, …

A Malicious Switch Could Do Many Things…

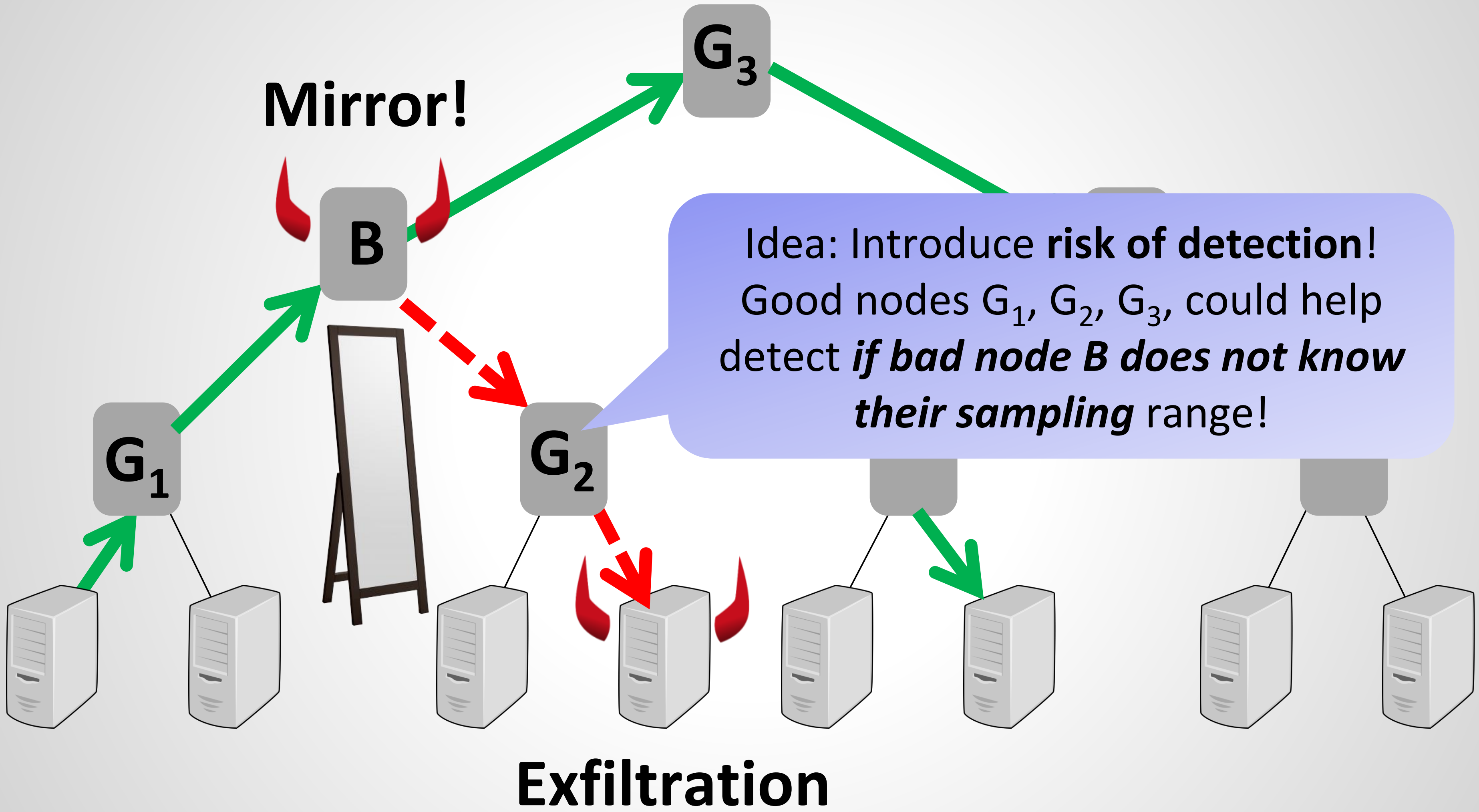


„Could SDN be used to render trajectory sampling more robust to such behavior?“

Federal Office for Information Security

Also: drop packets (that are currently not sampled), inject packets, change VLAN tag, …

A Malicious Switch Could Do Many Things…

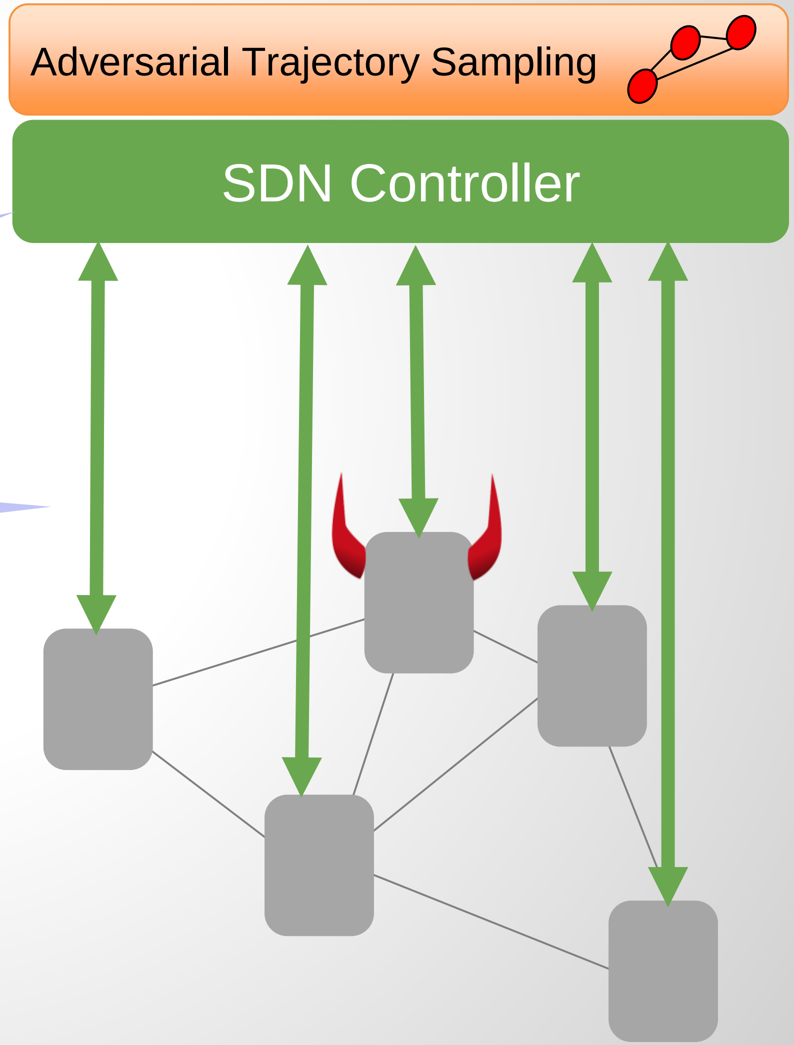


Adversarial Trajectory Sampling: A Case of SDN?

Idea: design SDN application that makes sampling **unpredictable**!

Controller distributes hash ranges **redundantly**…

… but securely over (**secure**) communication channels.

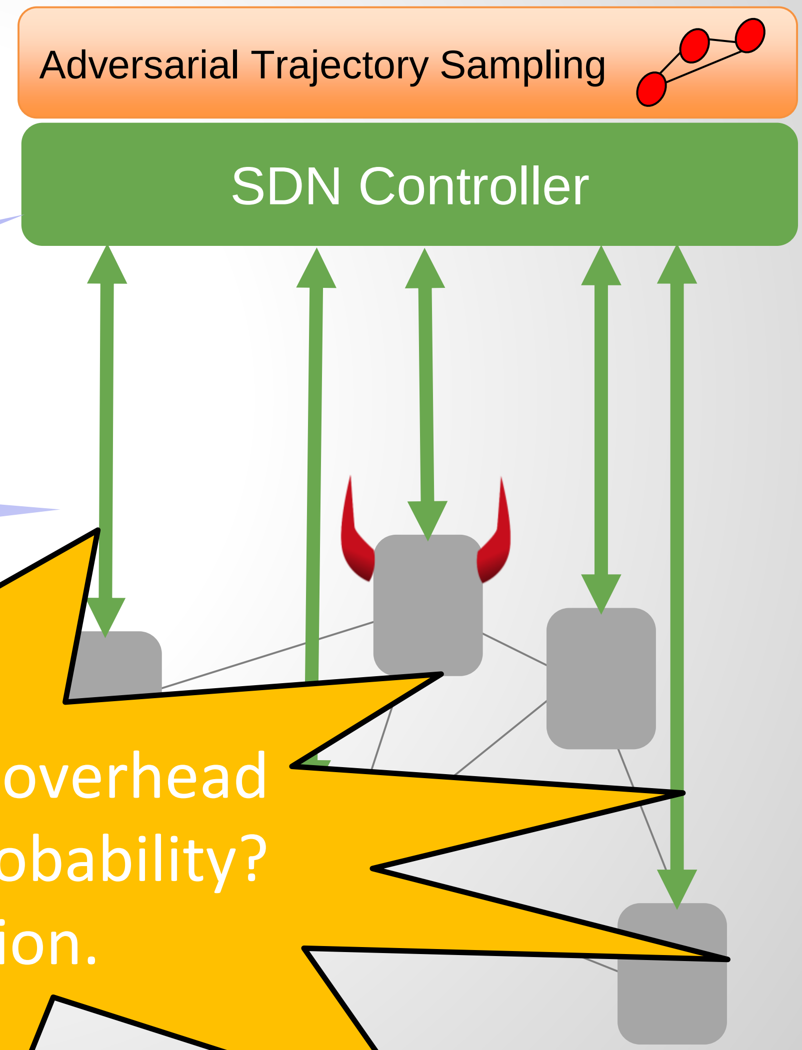


Adversarial Trajectory Sampling: A Case of SDN?

Idea: design SDN application that makes sampling unpredictable!

Controller distributes hash ranges redundantly…

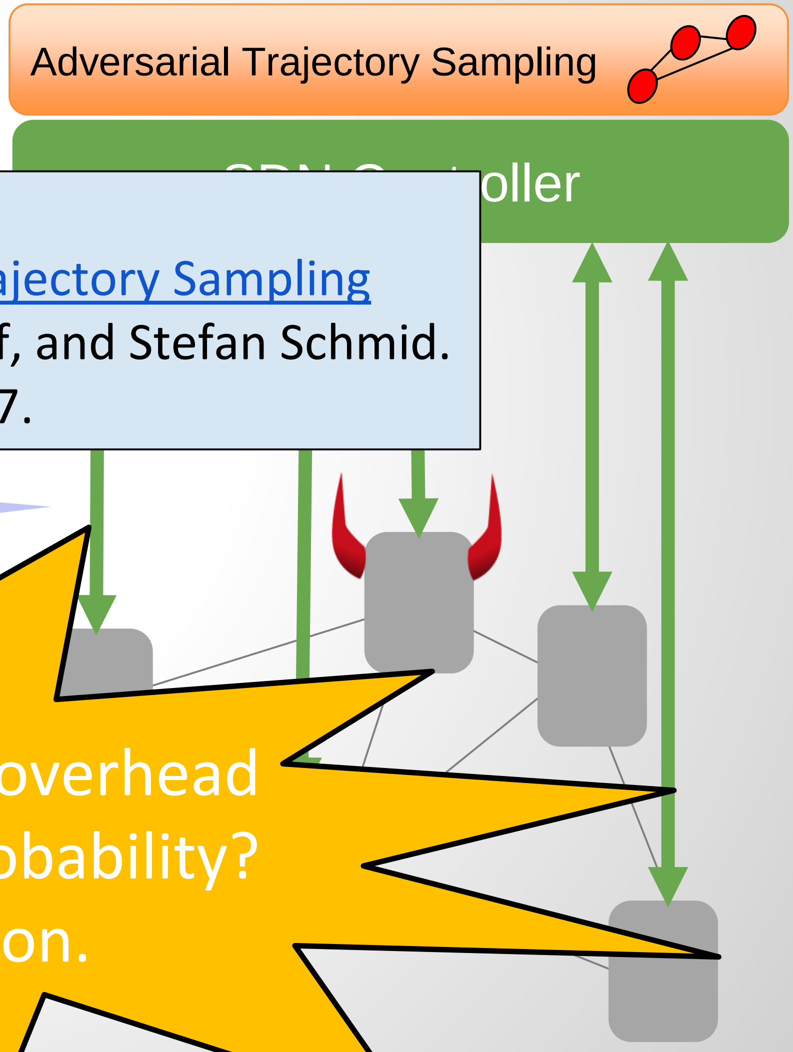
… but securely over (secure) communication channels.



How to minimize sampling overhead and maximize detection probability? An algorithmic question.

Adversarial Trajectory Sampling: A Case of SDN?

Idea: design SDN application that makes sampling unpredictable!



Cont
ra

Further reading:
Software-Defined Adversarial Trajectory Sampling
Kashyap Thimmaraju, Liron Schiff, and Stefan Schmid.
ArXiv Technical Report, May 2017.

... but securely over (secure) communication channels.

How to minimize sampling overhead and maximize detection probability? An algorithmic question.

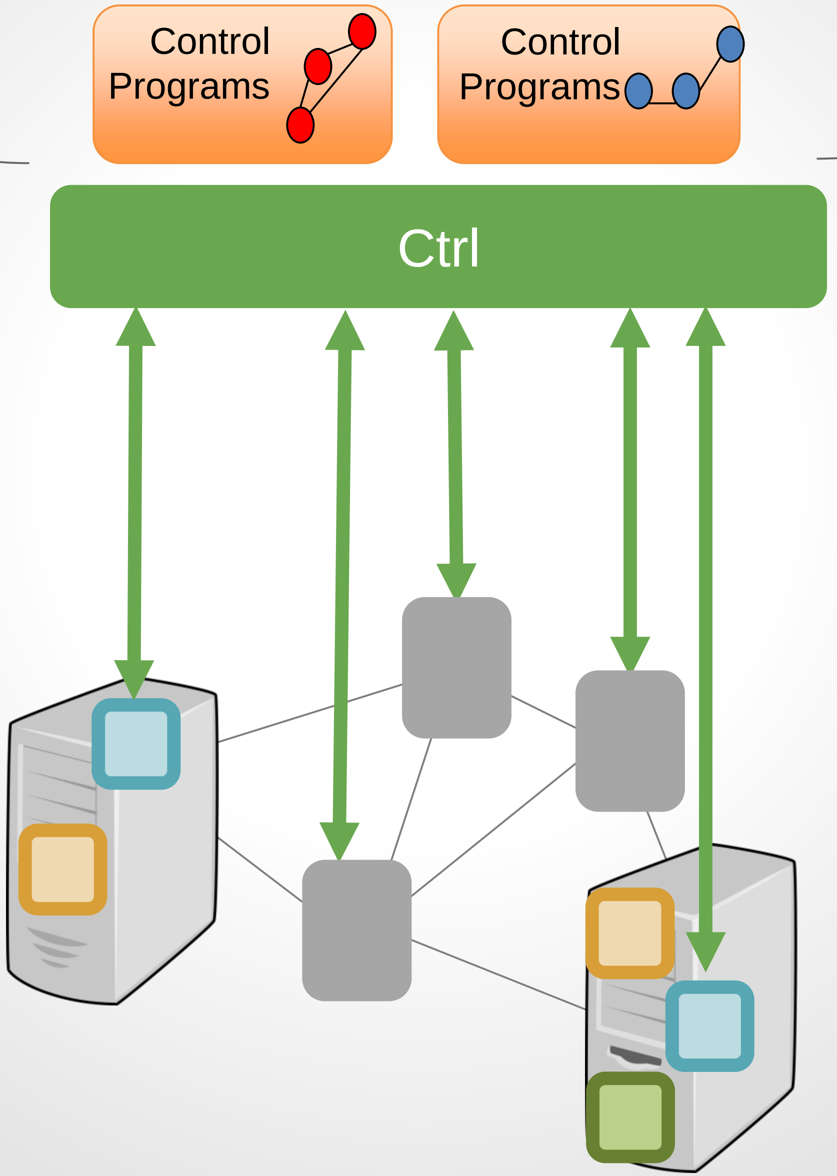
Conclusions

Opportunities

Challenges

E.g., innovative services

E.g., waypoint routing, traffic engineering



Conclusions

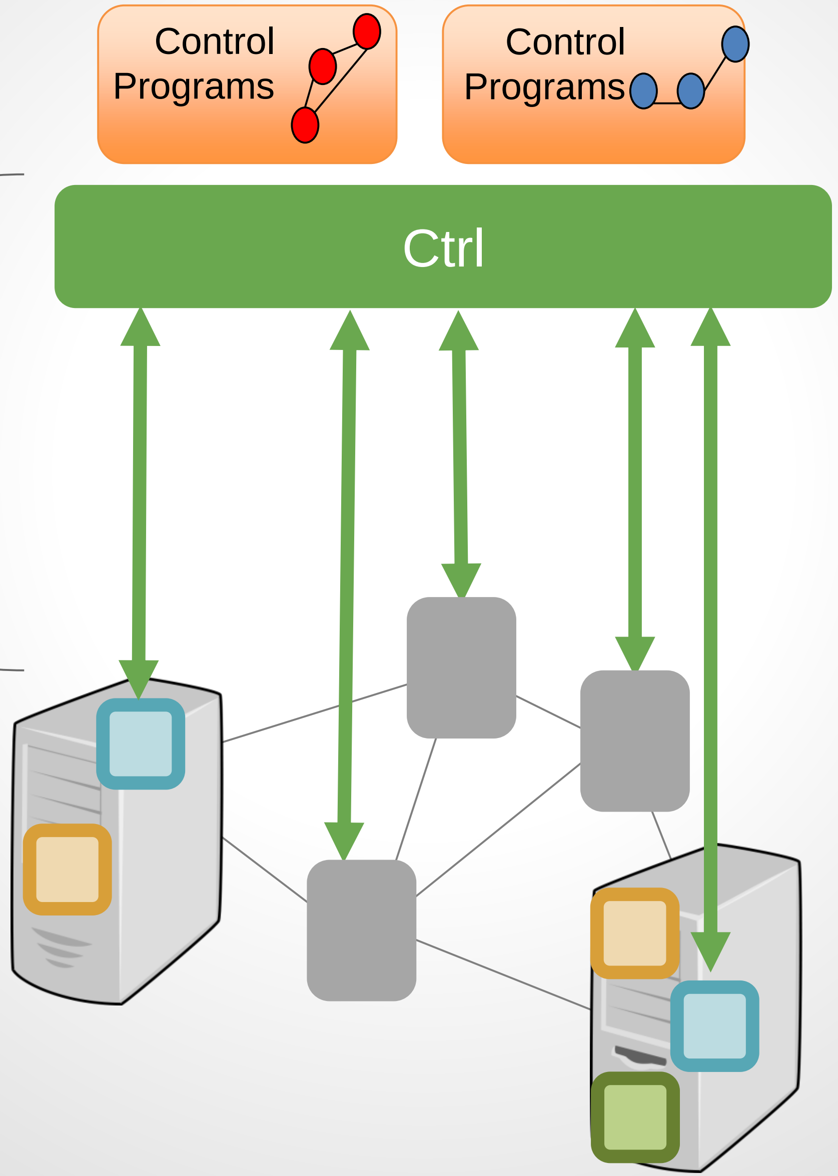
Opportunities

E.g., decoulping: evolve control plane independently of dataplane

Challenges

E.g., keeping controller up-to-date

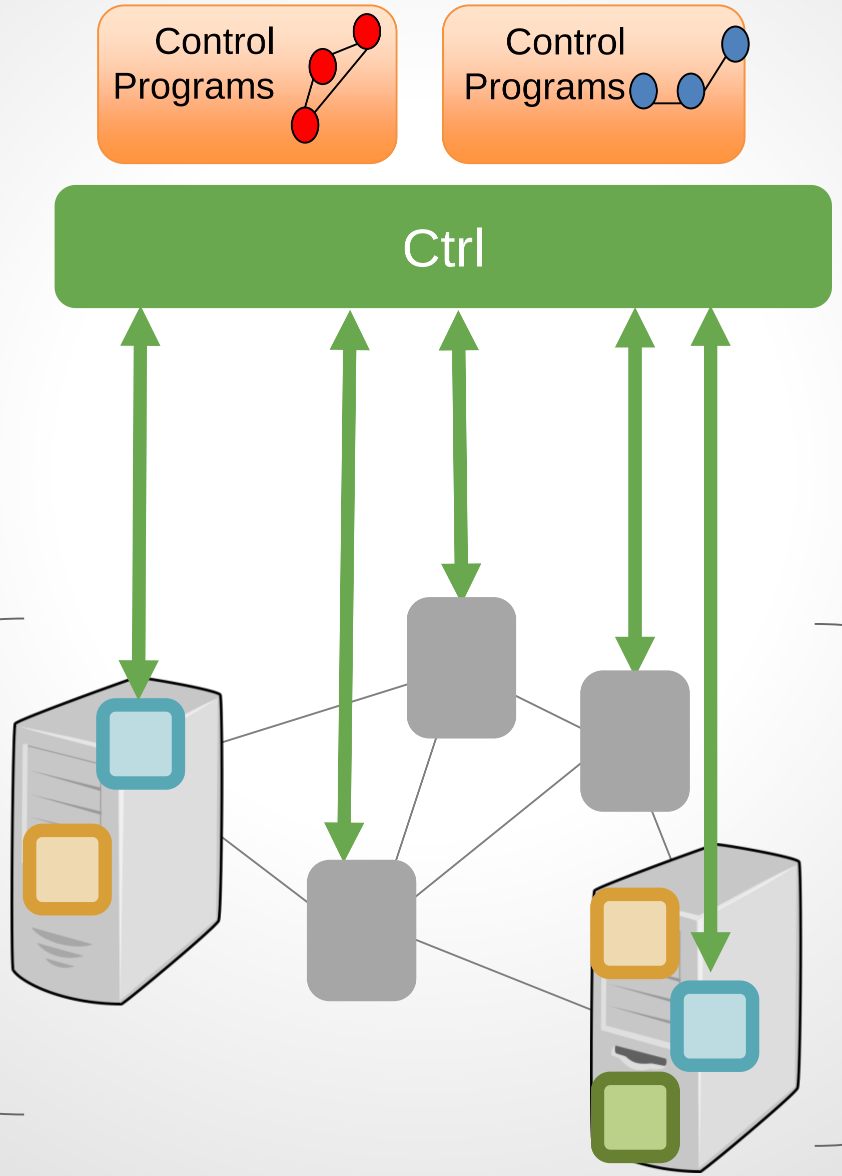
E.g., consistent network update



Conclusions

Opportunities

Challenges



E.g., simple and open interface

E.g., functionality that should stay here?

E.g., complexity of verification, local failover, ….?

Thank you! Questions?