Algorithmic Opportunities and Challenges of NFV and SDN
A Guided Tour

Stefan Schmid

Aalborg University, Denmark & TU Berlin, Germany

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

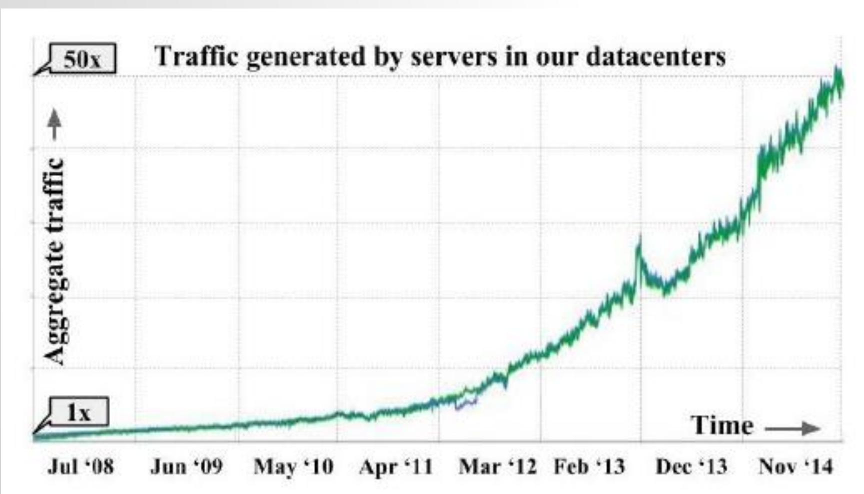
Rhone and Arve Rivers, Switzerland

Credits: George Varghese.

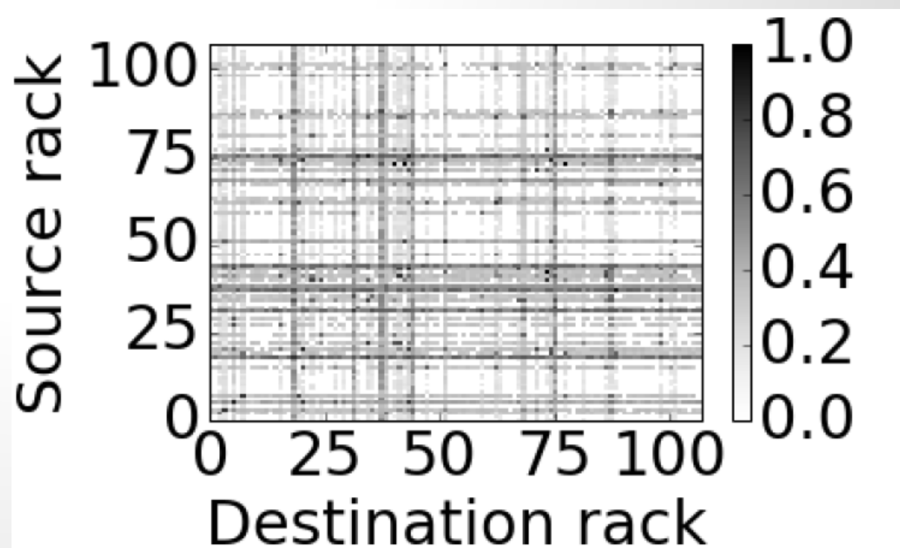
Why Flexibilities? Changing Requirements!

- Microservices deployed using containers introduce rapid changes in traffic workloads
- Augmented reality requires real-time responsiveness
- IoT significantly increases the # connected devices
- Datacenter traffic is growing (but has structure and is sparse):

Jupiter rising @ SIGCOMM 2015



Heatmap of rack-to-rack traffic ProjecToR @ SIGCOMM 2016



Credits: Why (and How) Networks Should Run Themselves. Nick Feamster and Jennifer Rexford

Big Challenge: Dependability & Complexity

Datacenter, enterprise, carrier networks have become mission-critical infrastructure!
But even techsavvy companies struggle to provide reliable operations.

We discovered a misconfiguration on this pair of switches that caused what's called a "bridge loop" in the network.

A network change was […] executed incorrectly […] more "stuck" volumes and added more requests to the re-mirroring storm

Service outage was due to a series of internal network events that corrupted router data tables

Experienced a network connectivity issue […] interrupted the airline's flight departures, airport processing and reservations systems

Credits: Nate Foster

Big Challenge: Debugging and Tools

The Wall Street Bank Anecdote

- Outage of a data center of a Wall Street investment bank: lost revenue measured in USD 10^6 / min!

- Quickly, assembled emergency team:

The compute team: quickly came armed with reams of logs, showing how and when the applications failed, and had already written experiments to reproduce and isolate the error, along with candidate prototype programs to workaround the failure.

The storage team: similarly equipped, showing which file system logs were affected, and already progressing with workaround programs.

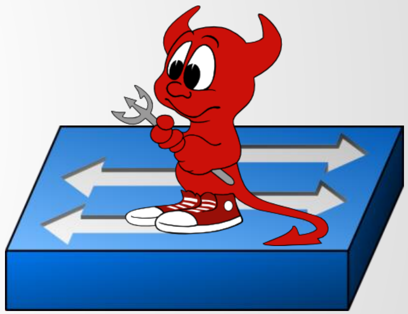
The networking team: All the networking team had were two tools invented over twenty years ago [*ping* and *traceroute*] to merely test end-to-end connectivity. Neither tool could reveal problems with the switches, the congestion experienced by individual packets, or provide any means to create experiments to identify, quarantine and resolve the problem.

Source: «The world's fastest and most programmable networks»
White Paper Barefoot Networks

Security: New Threat Models

- Internet-of-Things, e.g., DDoS Fall 2016
  - "Baby-phone", hacked cameras, etc.
  - Biggest Internet attack ever: >500 Gbps

- Untrusted hardware
  - Attackers repeatedly compromised routers
  - Compromised routers are traded underground
  - Network vendors left backdoors open
  - National security agencies can bug network equipment (e.g., hardware backdoors, Snowden leaks)

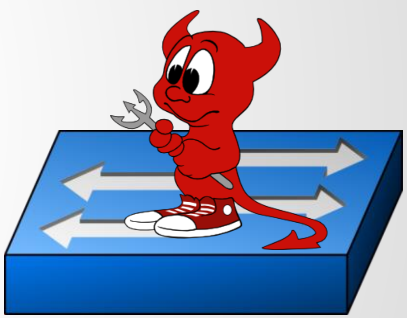


- Hacked wireless/cellular equipment
  - Insecure femto cells
  - Rogue access points

Security: New Threat Models

- Internet-of-Things, e.g., DDoS Fall 2016
  - “Baby-phone”, hacked cameras, etc.
  - Biggest Internet attack ever: >500 Gbps

- Untrusted hardware
  - Attackers repeatedly compromised routers
  - Compromised routers are traded underground
  - Network vendors left backdoors open
  - Natio … equip … den leaks)



- Hacked wireless/cellular equipment
  - Insecure femto cells
  - Rogue access points

Big Challenge: Efficient Resource Utilization

- Wireless infrastructure not used very efficiently today

- E.g., WiFi: huge demand-supply mismatch (e.g., home networks):

Millons of access points

but:

A device can access only through a very small percentage

Solution: virtualization, multi-tenancy, etc.?

Big Challenge: Efficient Resource Utilization

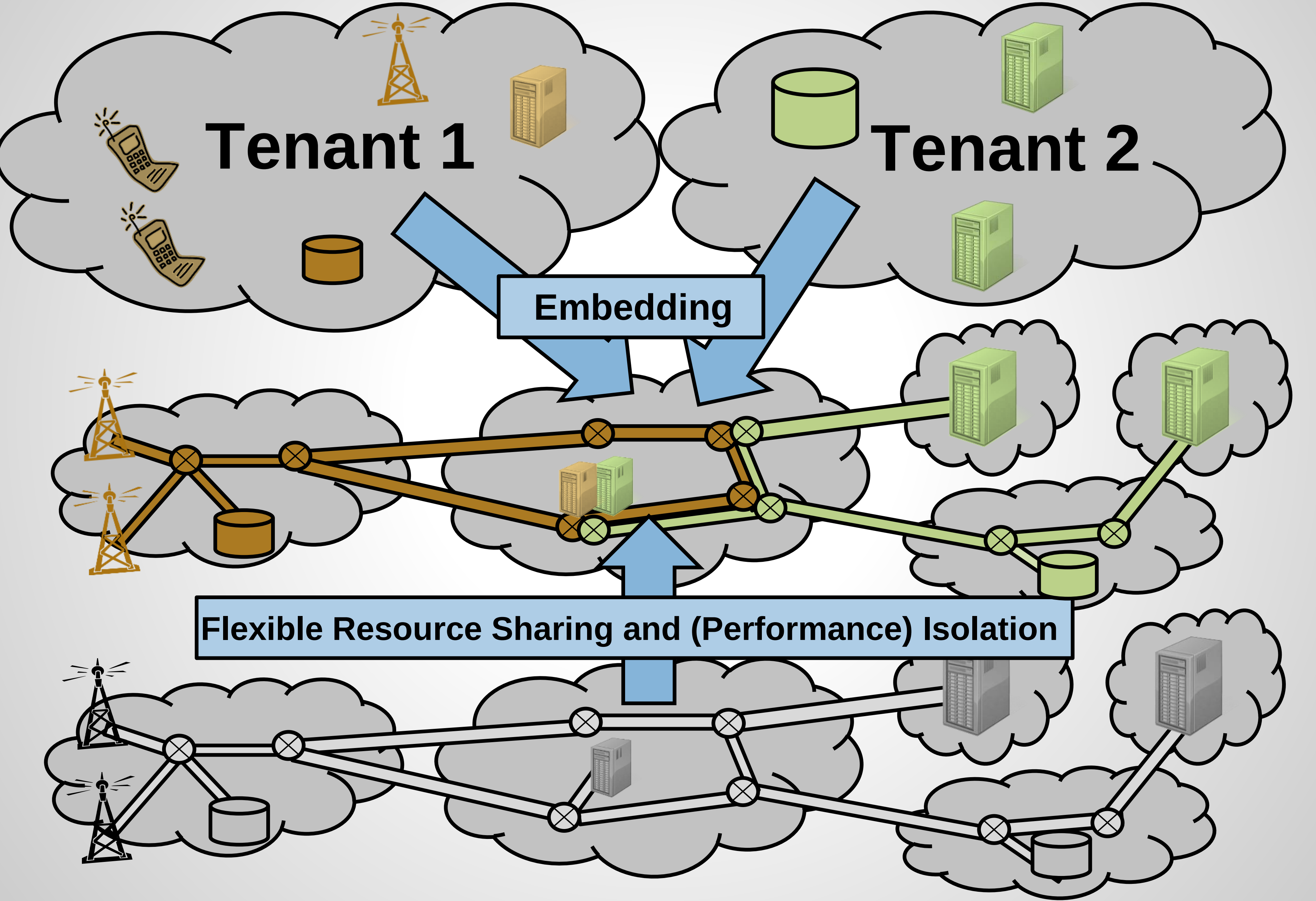
Further reading:

OpenSDWN: Programmatic Control over Home and Enterprise WiFi
Julius Schulz-Zander, Carlos Mayer, Bogdan Ciobotaru, Stefan Schmid, and Anja Feldmann.
ACM Sigcomm Symposium on SDN Research (**SOSR**), Santa Clara, California, USA, June 2015.

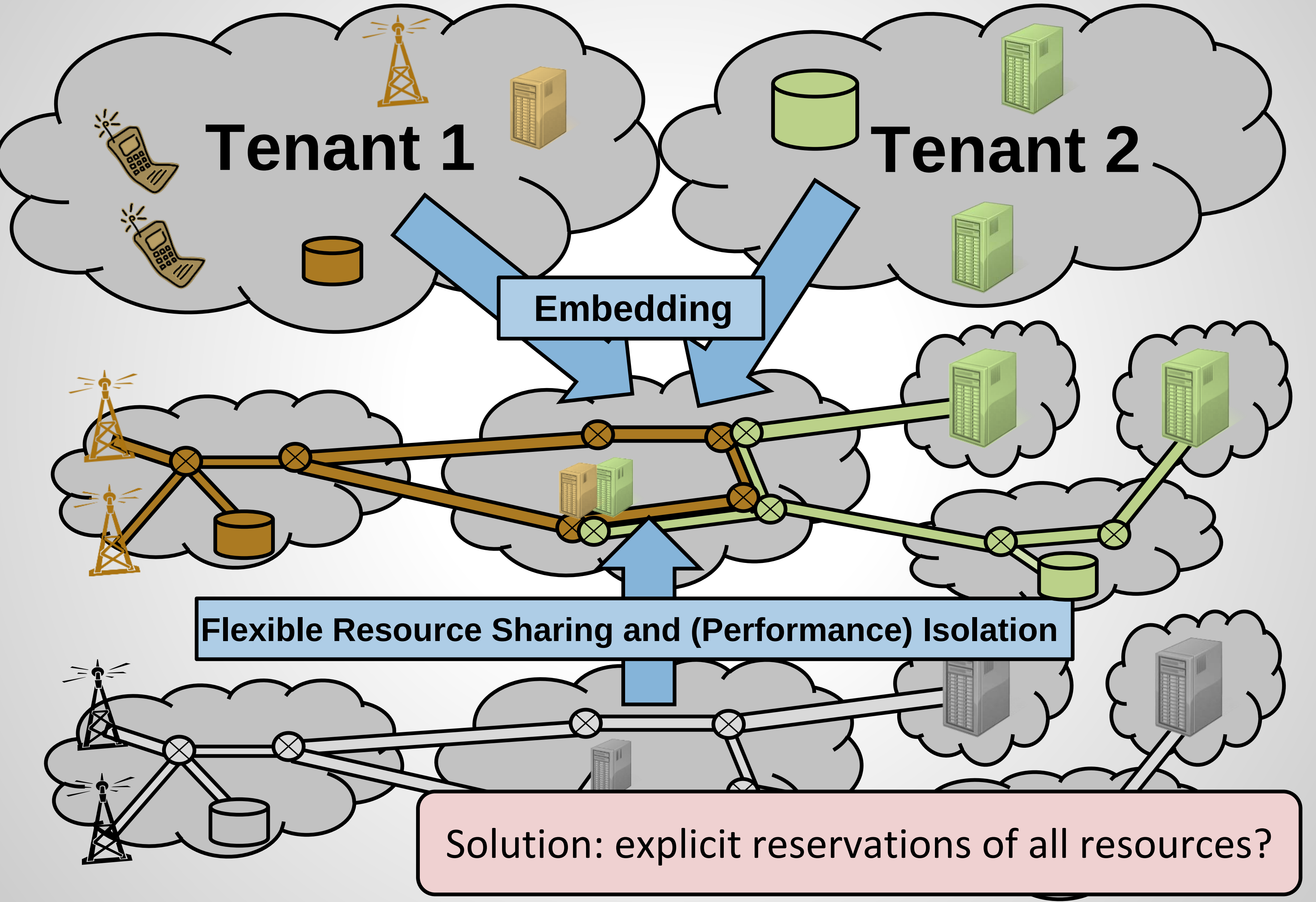
SecuSpot: Toward Cloud-Assisted Secure Multi-Tenant WiFi HotSpot Infrastructures
Julius Schulz-Zander, Raphael Lisicki, Stefan Schmid, and Anja Feldmann.
ACM CoNEXT Workshop on Cloud-Assisted Networking (**CAN**), Irvine, California, USA, December 2016.

Solution: virtualization, multi-tenancy, etc.?

Big Challenges: Sharing and *Predictable* Performance



Big Challenges: Sharing and *Predictable* Performance



Solution: explicit reservations of all resources?

And many more…

- **Slow innovation:** Innovation speed depends on hardware life-cycles, impossible to tailor to specific needs
- **Traffic Engineering (TE):** efficient use of WAN infrastructure through more direct and fine-grained control of traffic (e.g., beyond shortest paths, destination-based routing)
- **Failover:** failover via control plane is slow, especially if control plane is decentralized (reconvegence time)
- **Cost:** special purpose hardware expensive

C.ACM 3/2016

A Purpose-Built Global Network: Google's Move to SDN

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

Some (often read) claims:

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

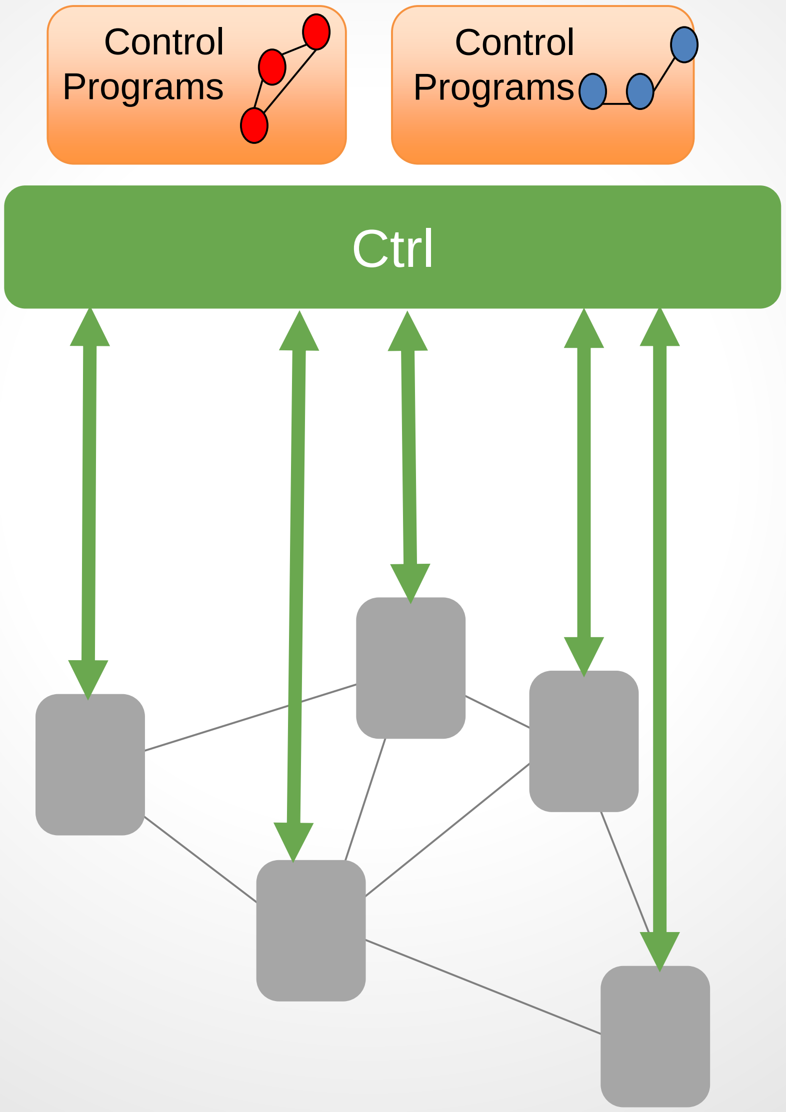
Really?

Algorithms? Avoid instabilities!

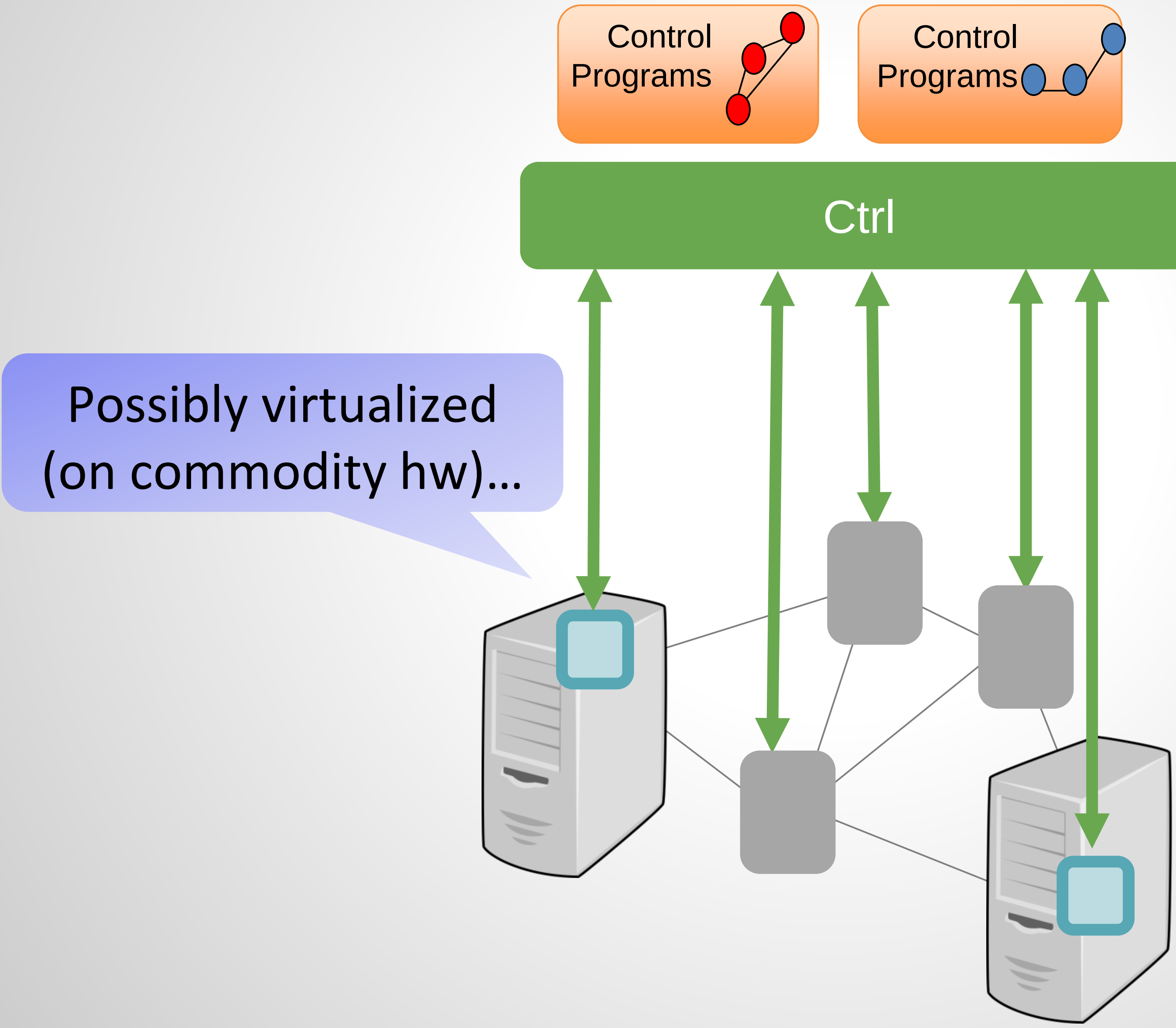
Complexity of this?

New threats?

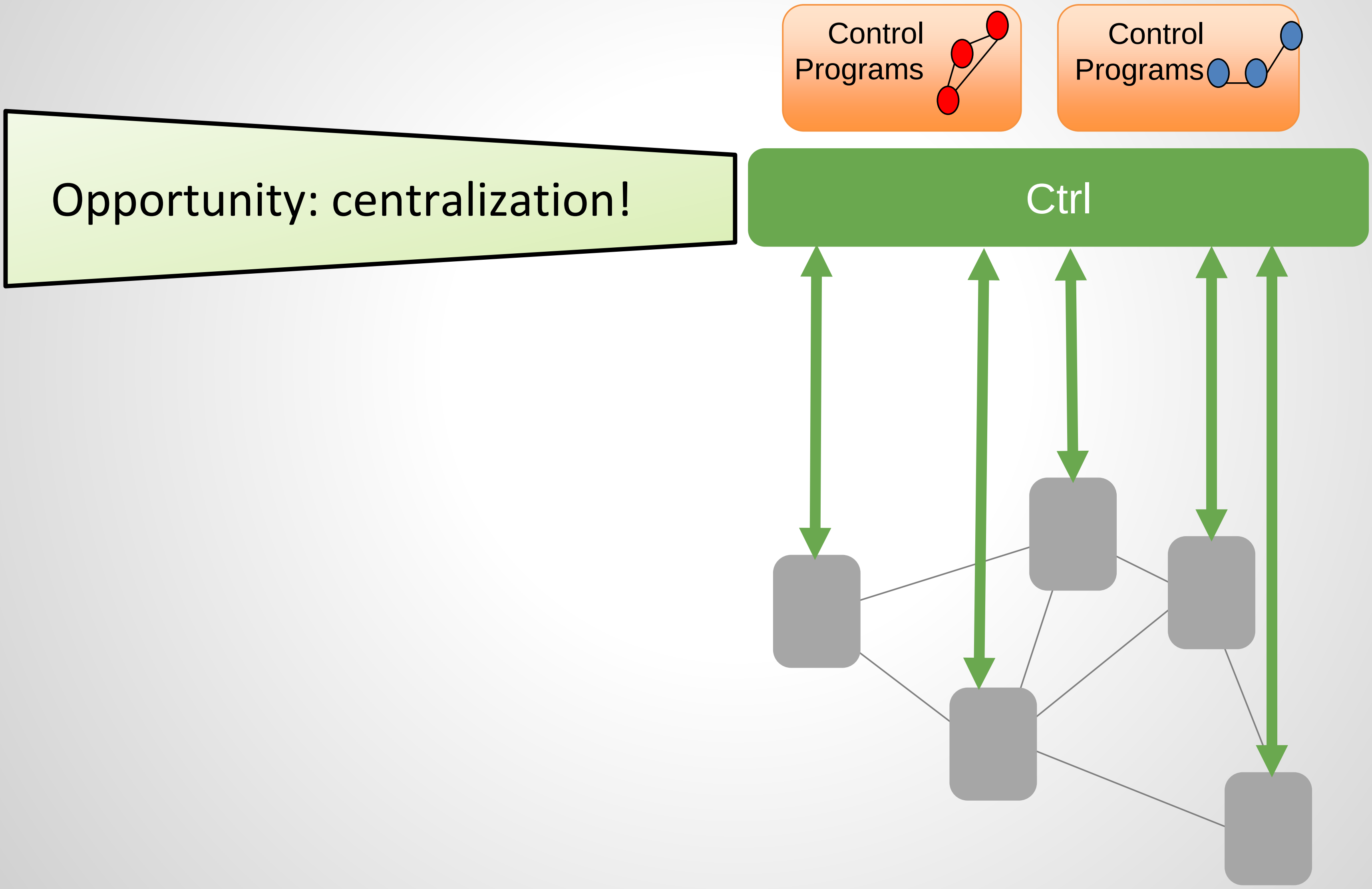
A Mental Model for This Talk



A Mental Model for This Talk



Let's talk about opportunities!



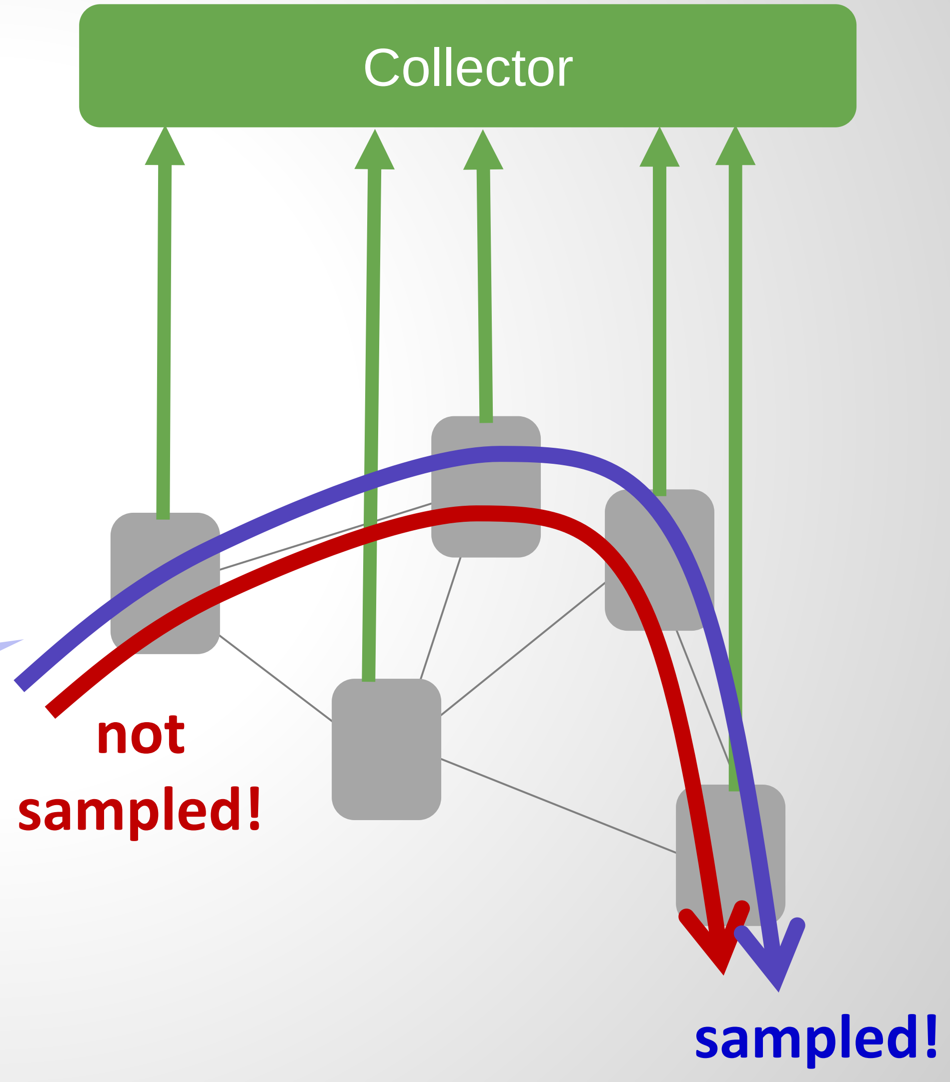
Example: Adversarial Trajectory Sampling

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!

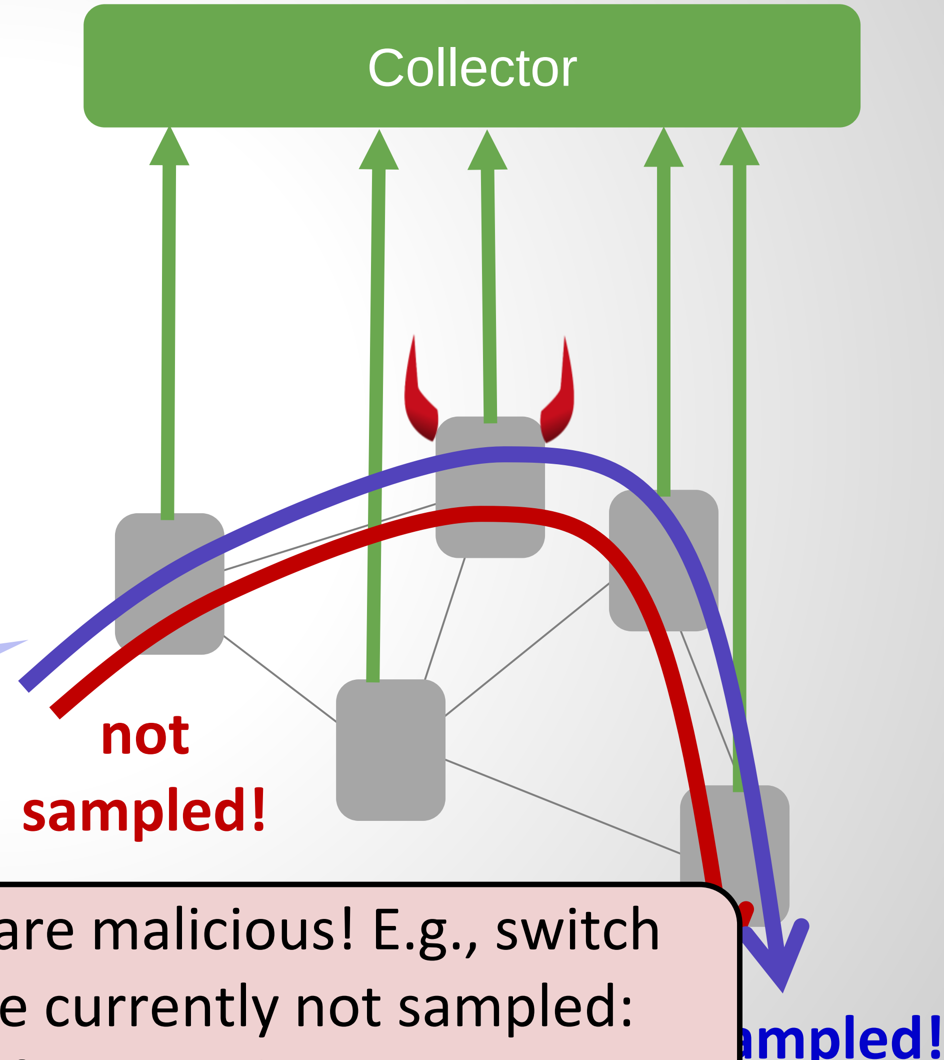


Example: Adversarial Trajectory Sampling

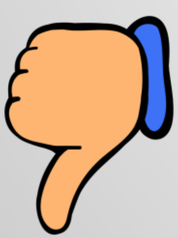
Trajectory Sampling

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

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

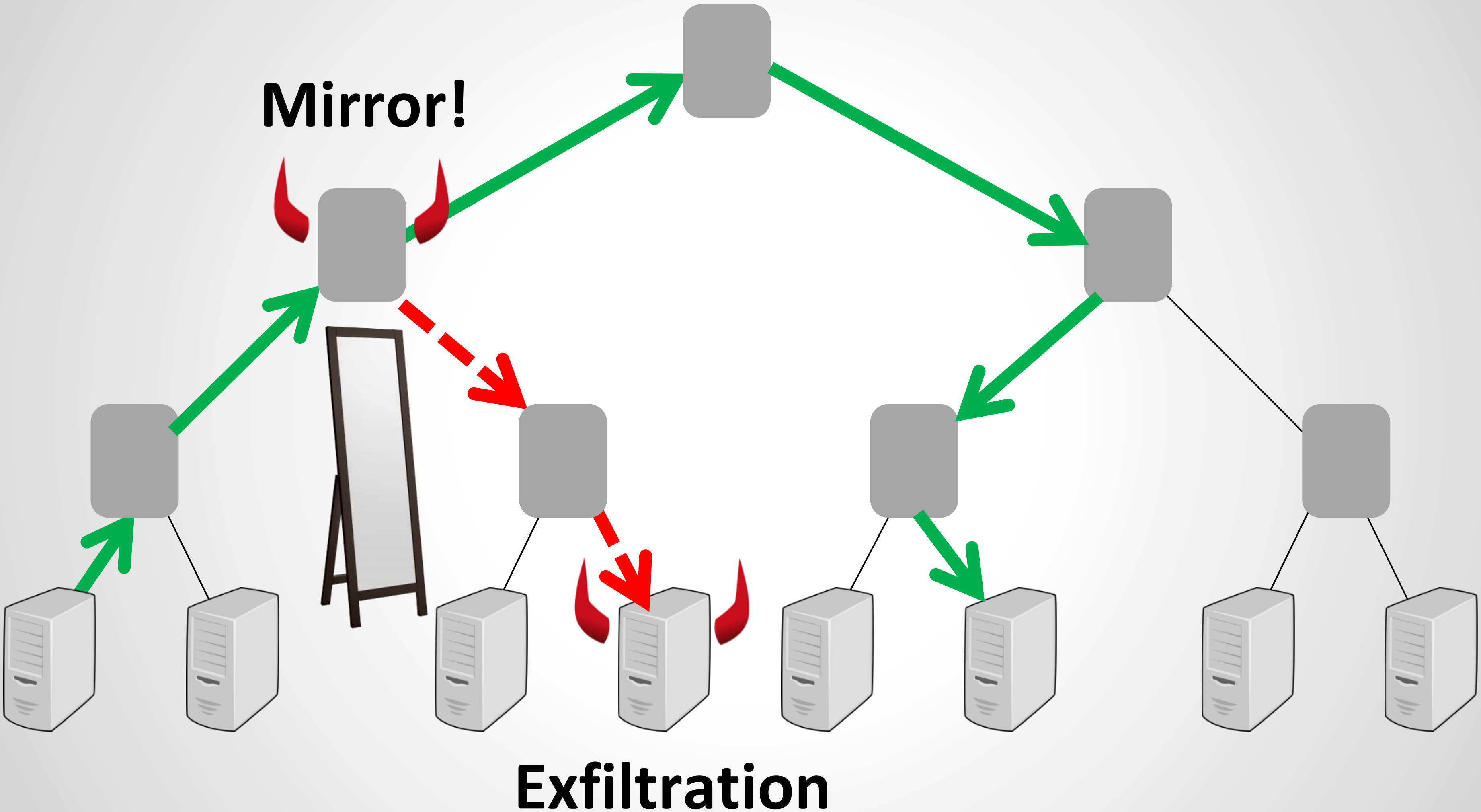


Packets sampled either at all or no location!

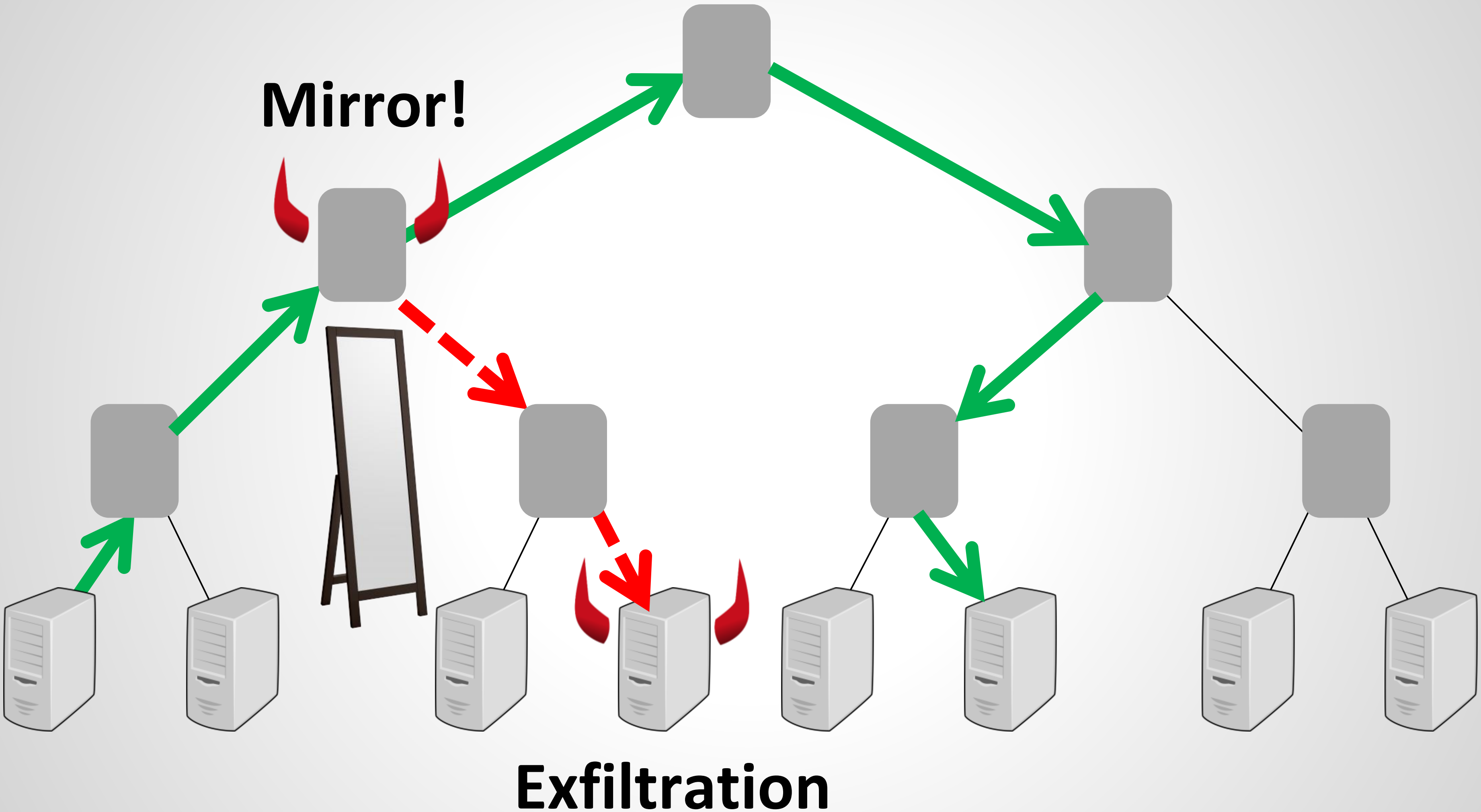


But: Fails when switches are malicious! E.g., switch knows which headers are currently not sampled: **no risk of detection**!

A Malicious Switch Could Do Many Things…

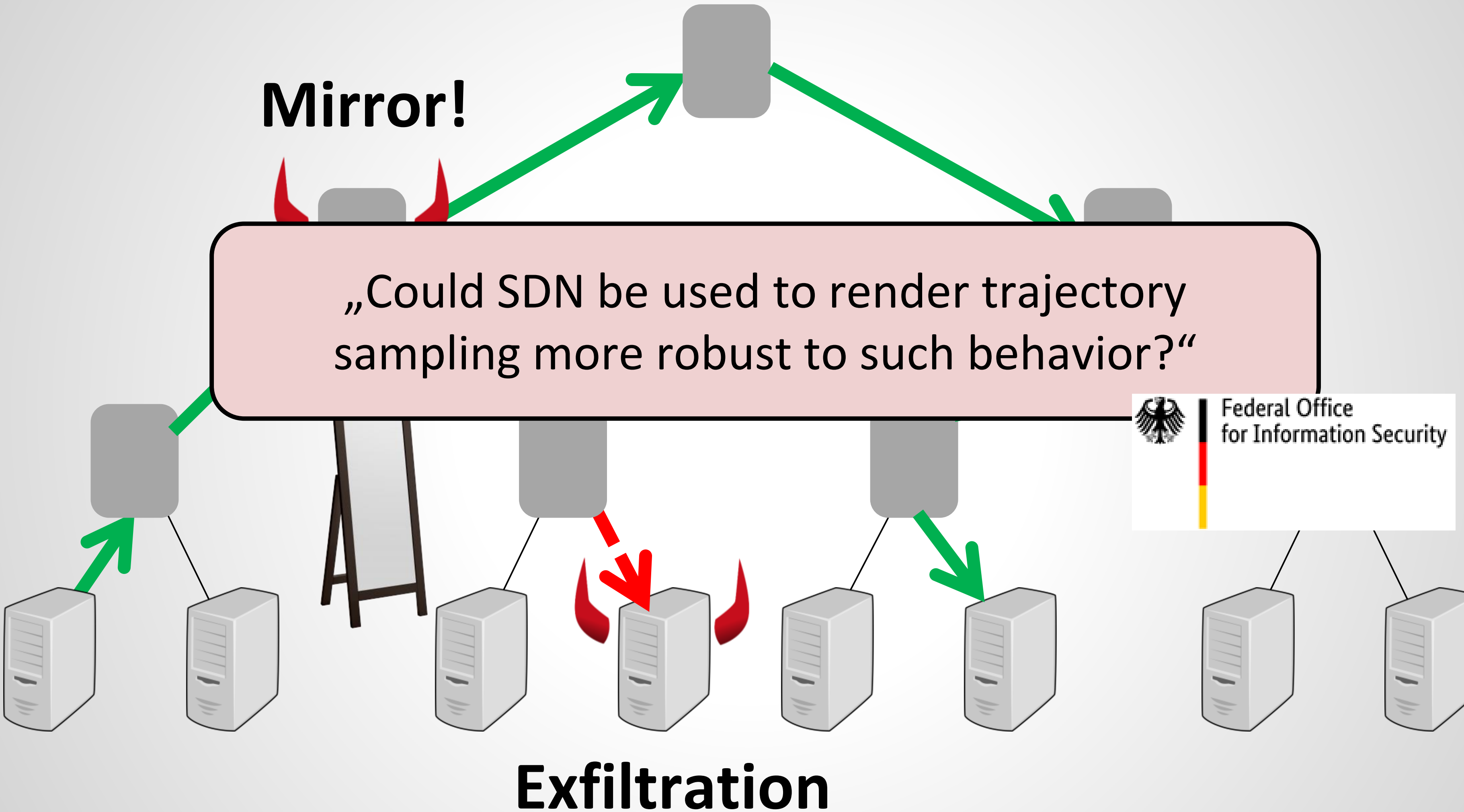


A Malicious Switch Could Do Many Things…



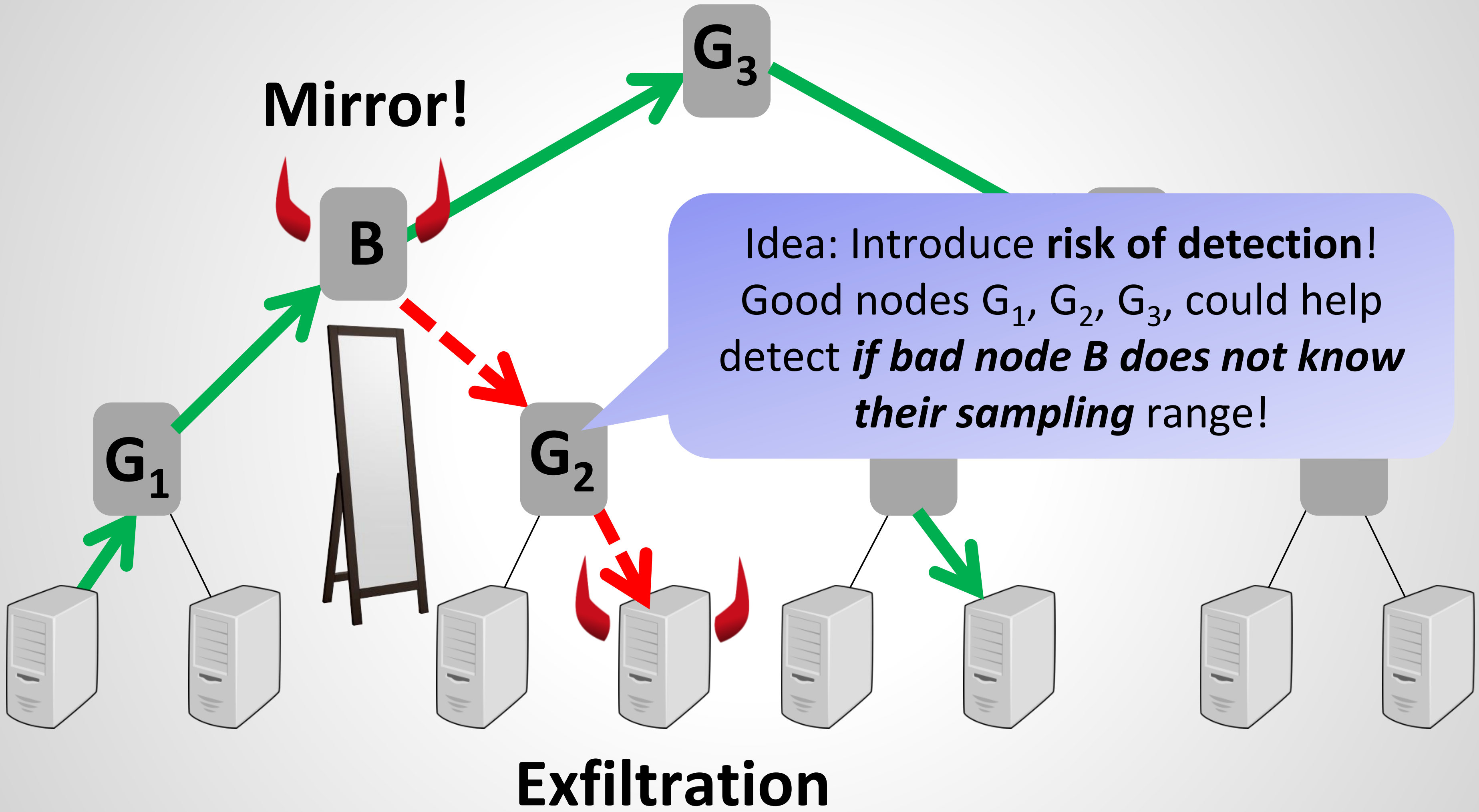
Also: drop packets (that are currently not sampled), inject packets, change VLAN tag, …

A Malicious Switch Could Do Many Things…



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

A Malicious Switch Could Do Many Things…

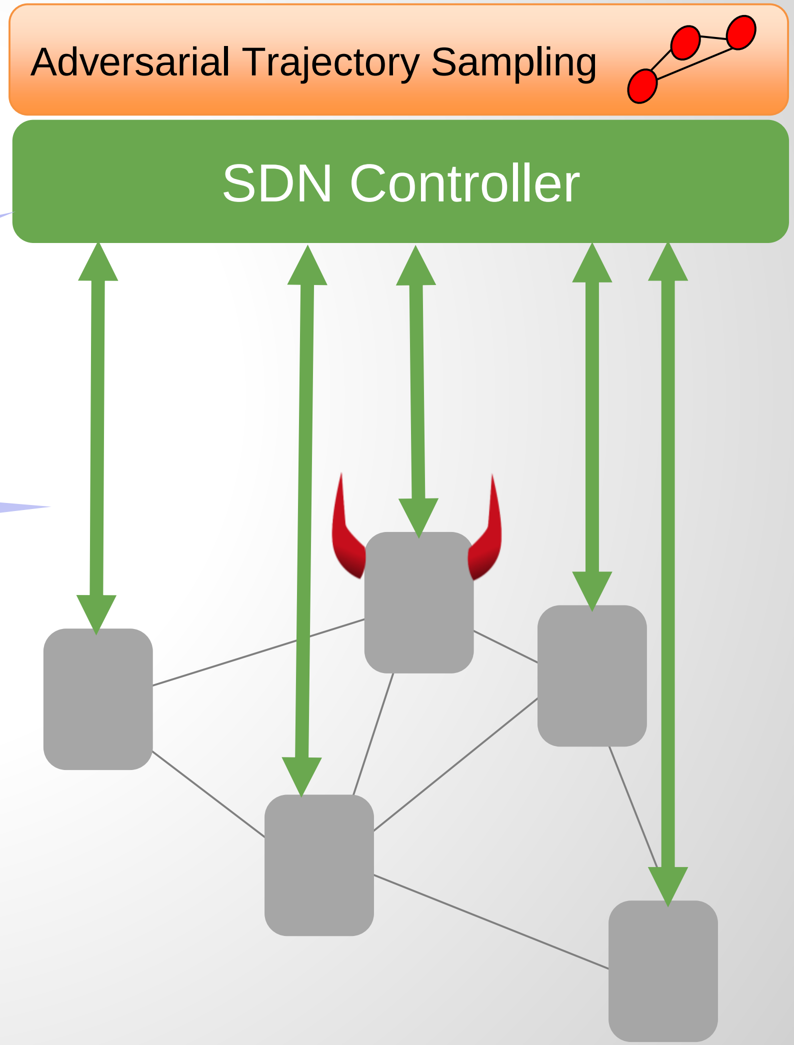


Adversarial Trajectory Sampling: A Case of SDN?

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

Controller distributes hash ranges **redundantly**…

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

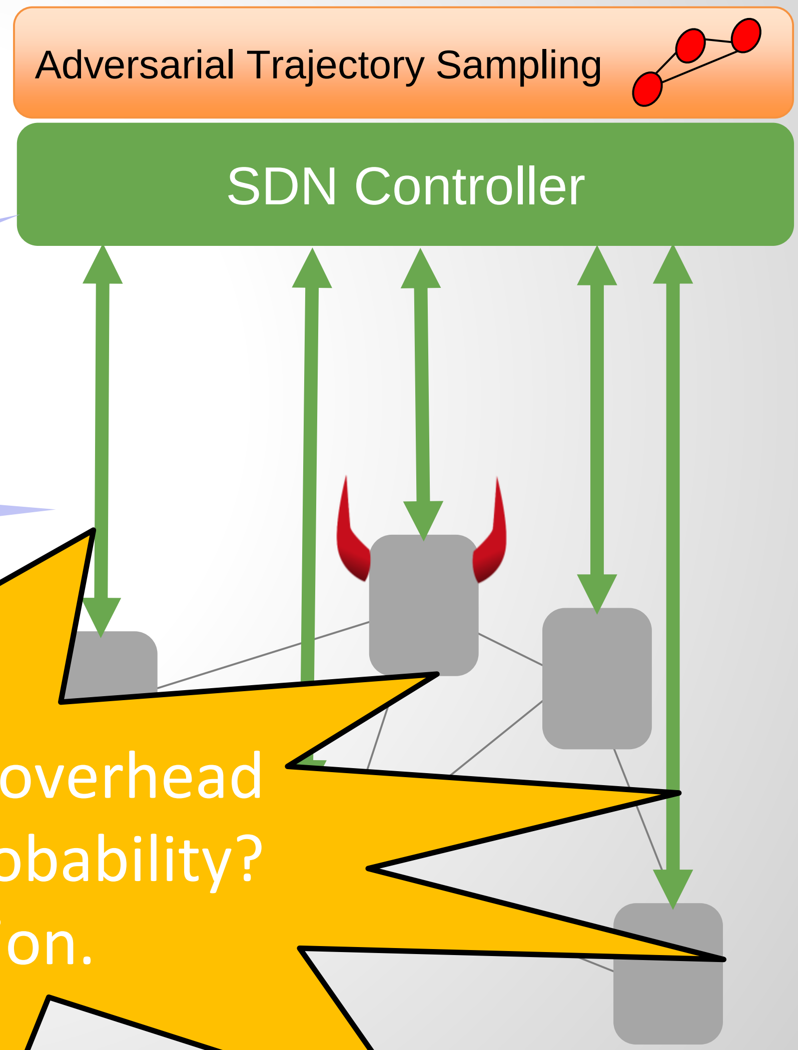


Adversarial Trajectory Sampling: A Case of SDN?

Idea: design SDN application that makes sampling unpredictable!

Controller distributes hash ranges redundantly…

… but securely over (secure) communication channels.



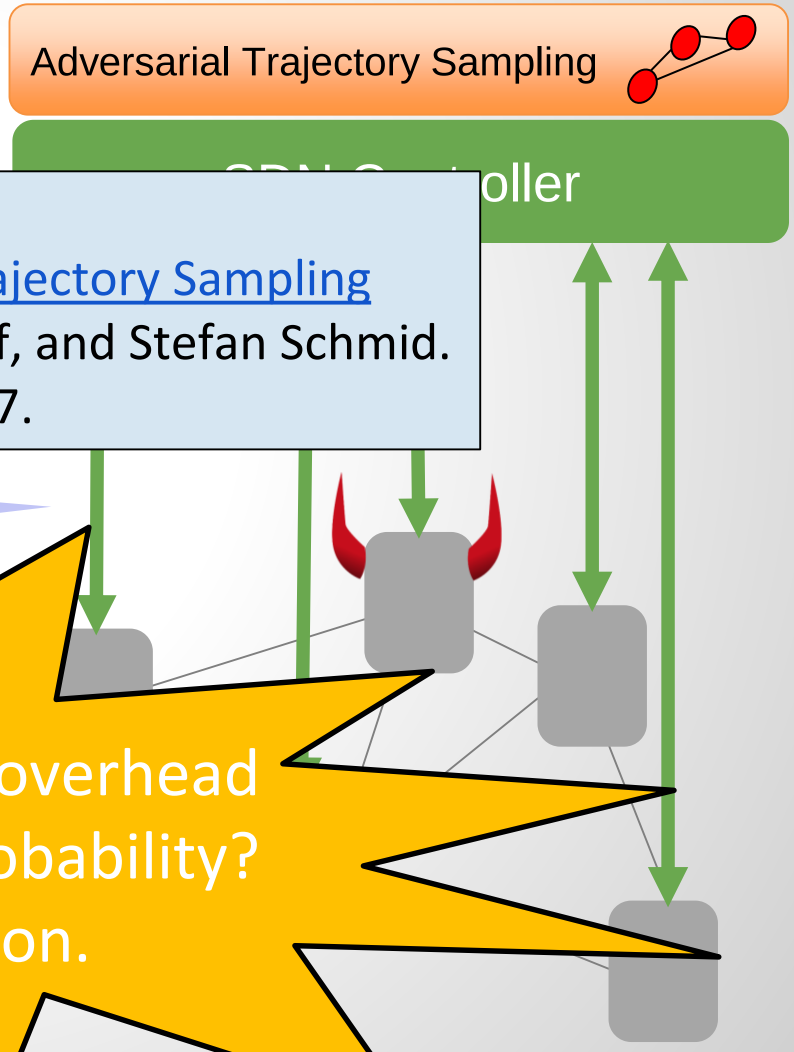
How to minimize sampling overhead and maximize detection probability? An algorithmic question.

Adversarial Trajectory Sampling: A Case of SDN?

Idea: design SDN application that makes sampling unpredictable!

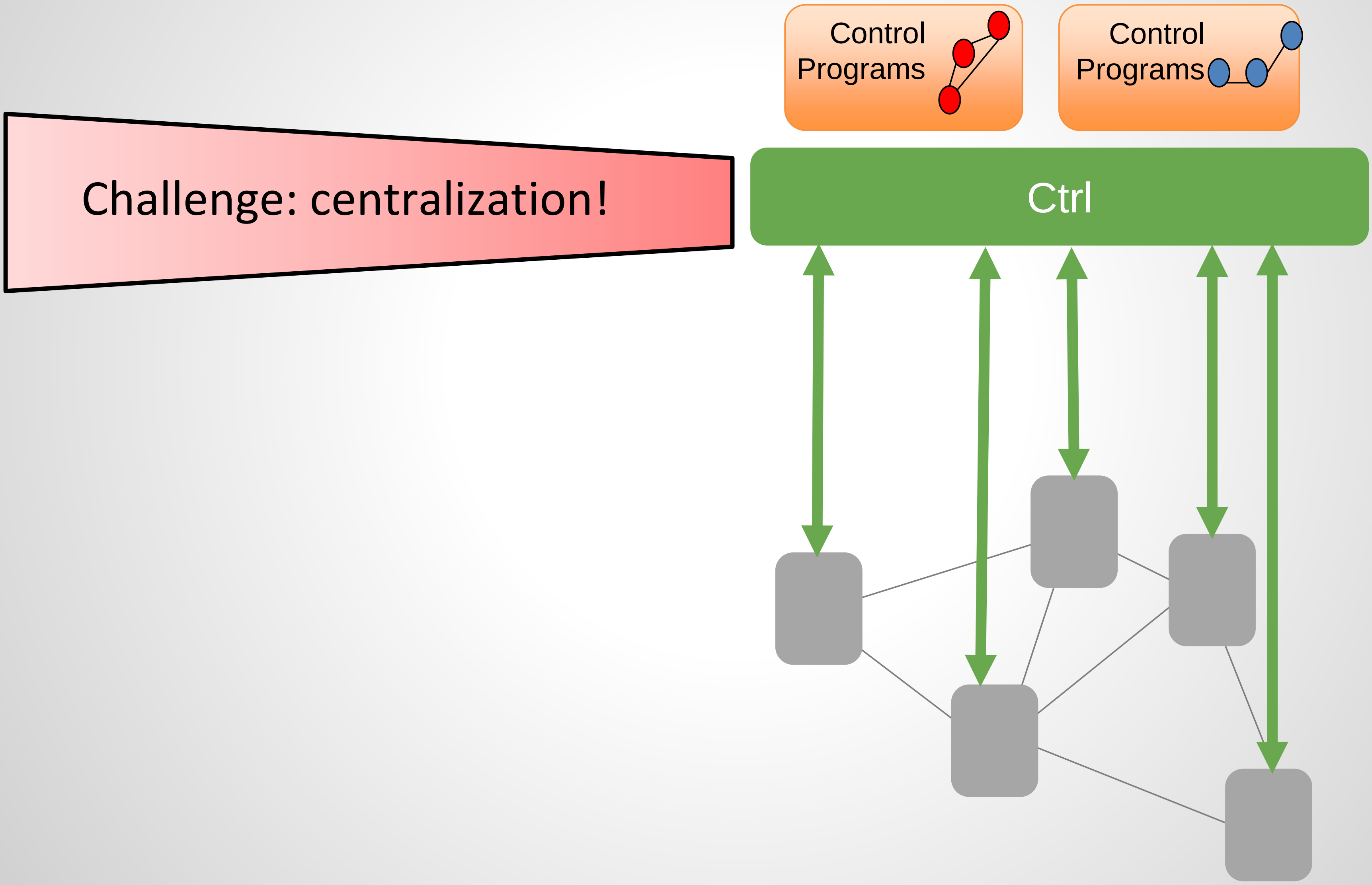
Further reading:
Software-Defined Adversarial Trajectory Sampling
Kashyap Thimmaraju, Liron Schiff, and Stefan Schmid.
ArXiv Technical Report, May 2017.

… but securely over (secure) communication channels.



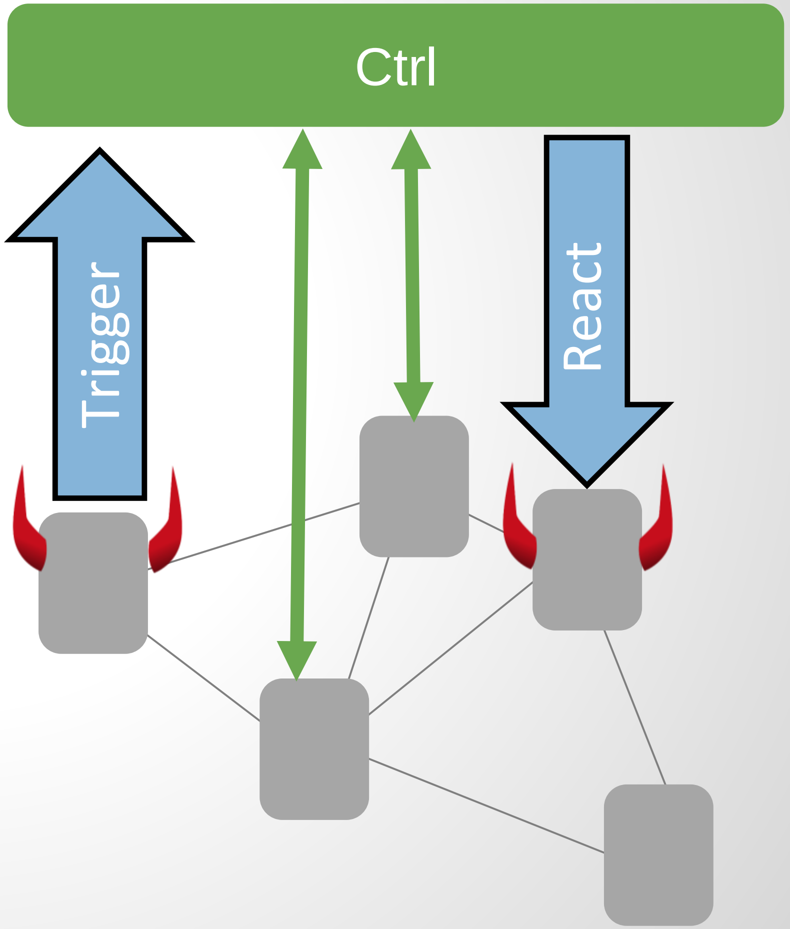
How to minimize sampling overhead and maximize detection probability? An algorithmic question.

A Mental Model for This Talk



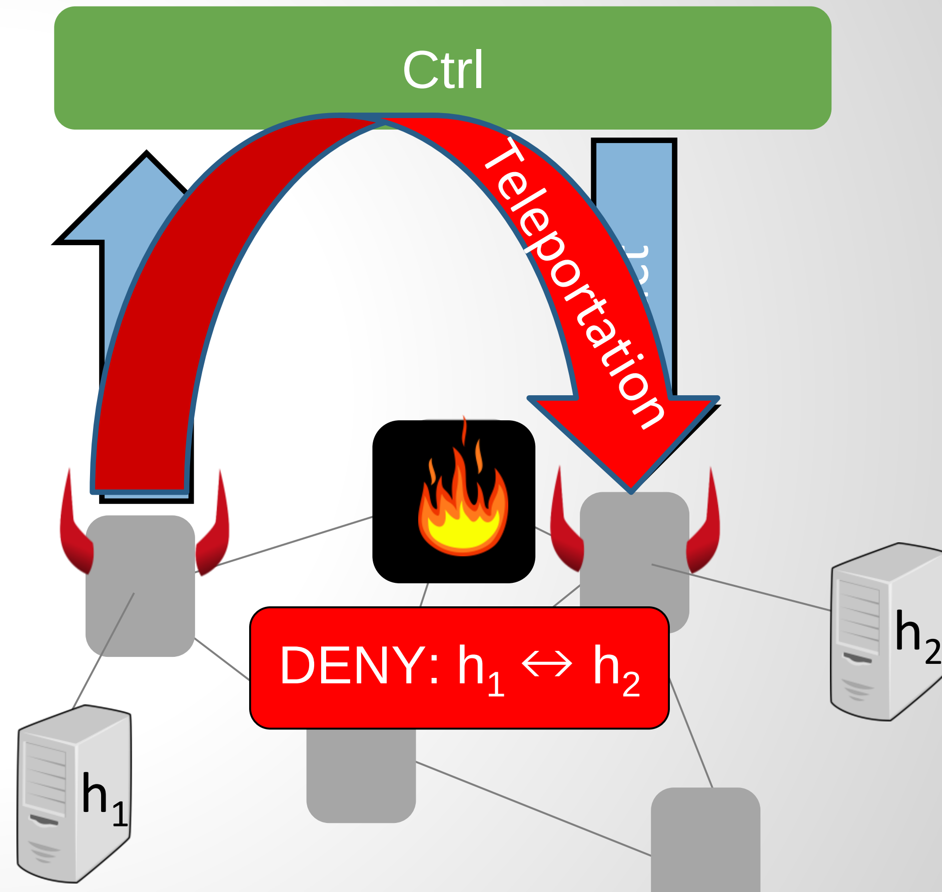
Central Controller Can Increase Attack Surface: E.g., May Be Exploited For Covert Communication

- 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

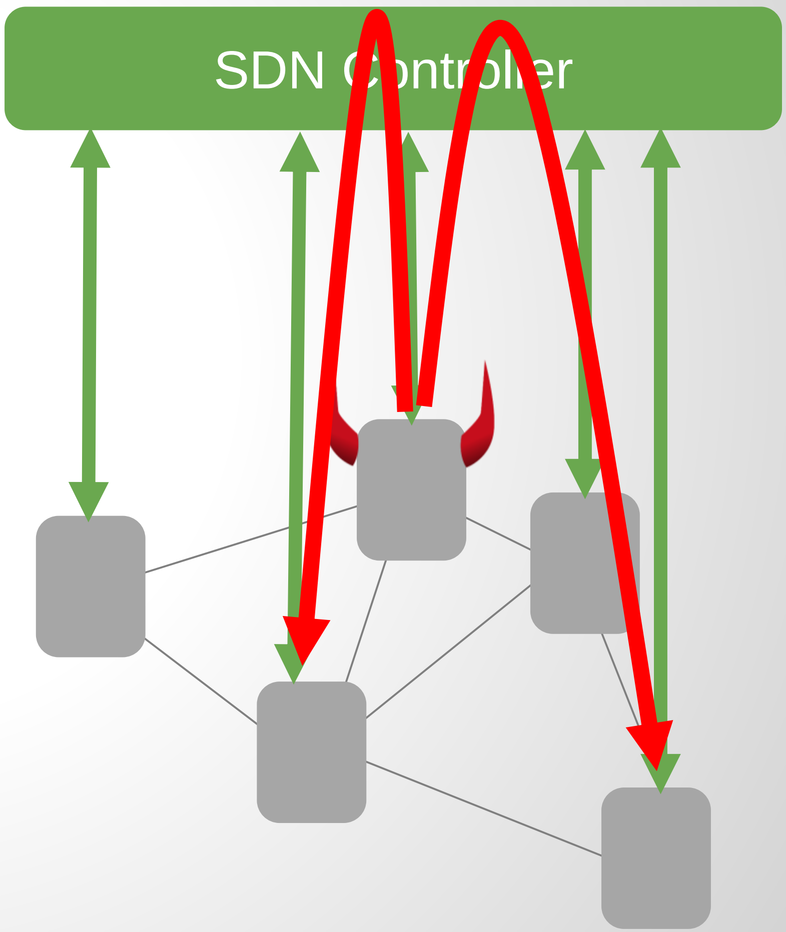
- May be used to bypass firewall
- Not easy to detect:
  - Traffic follows normal pattern of control communication, indirectly via controller
  - Teleportation channel is inside (encrypted) OpenFlow channel
- Need e.g., to correlate packet-ins, packet-outs, flow-mods, etc.



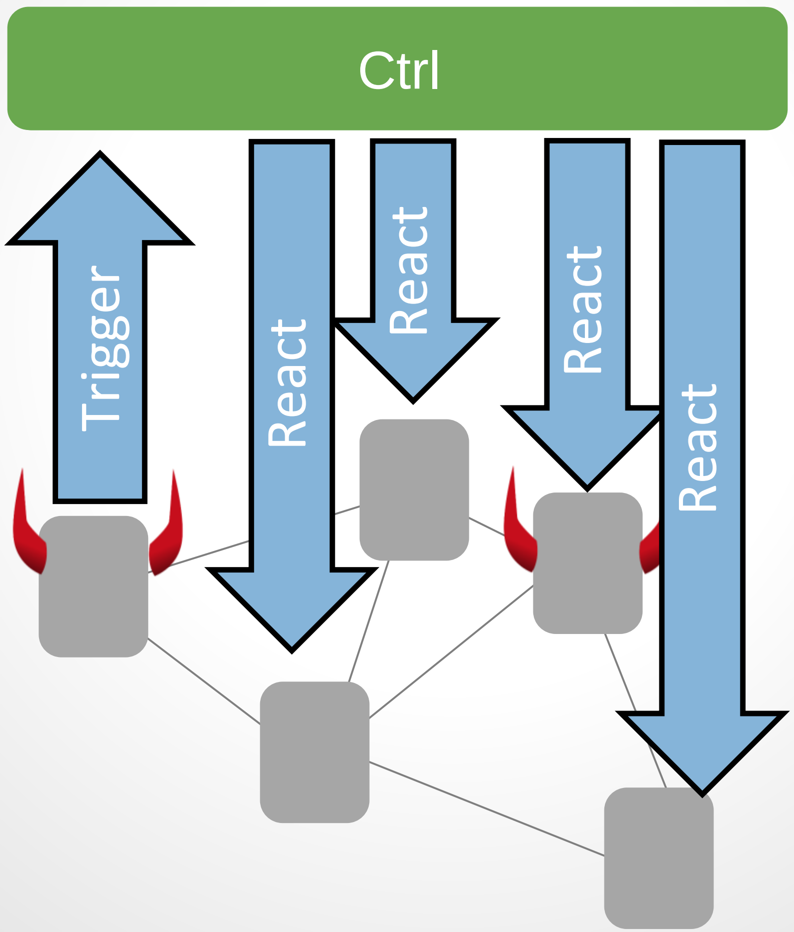
Another Vulnerability: Bidirected Communication Channels?

- Controller has communication channels to all network elements: could be exploited to spread a virus and compromise entire datacenter

- Case study in OvS



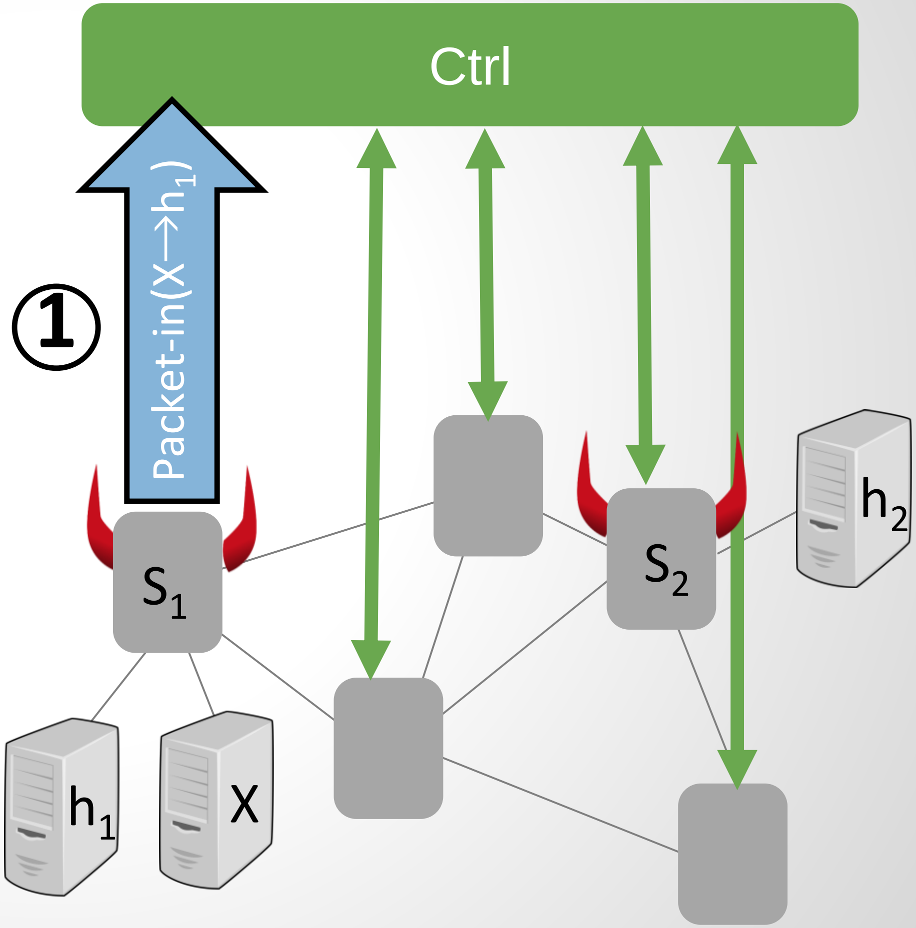
Pave-Path Technique



Teleportation: Path Update

- Controller performs MAC learning and updates paths to support mobility, VM migration, etc.
- Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch

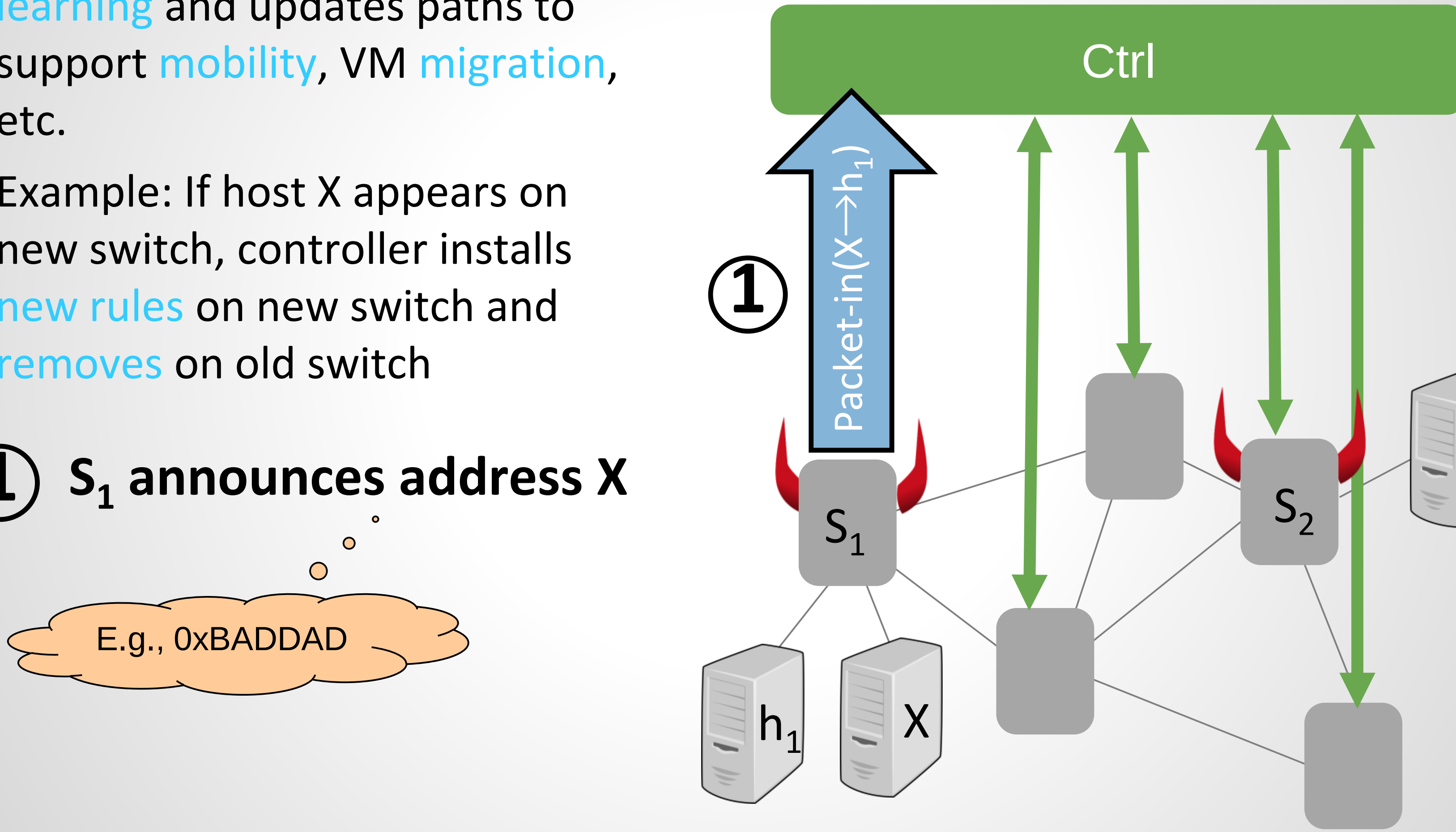
① S_1 announces address X



Teleportation: Path Update

- Controller performs MAC learning and updates paths to support mobility, VM migration, etc.
- Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch

① **S_1 announces address X**

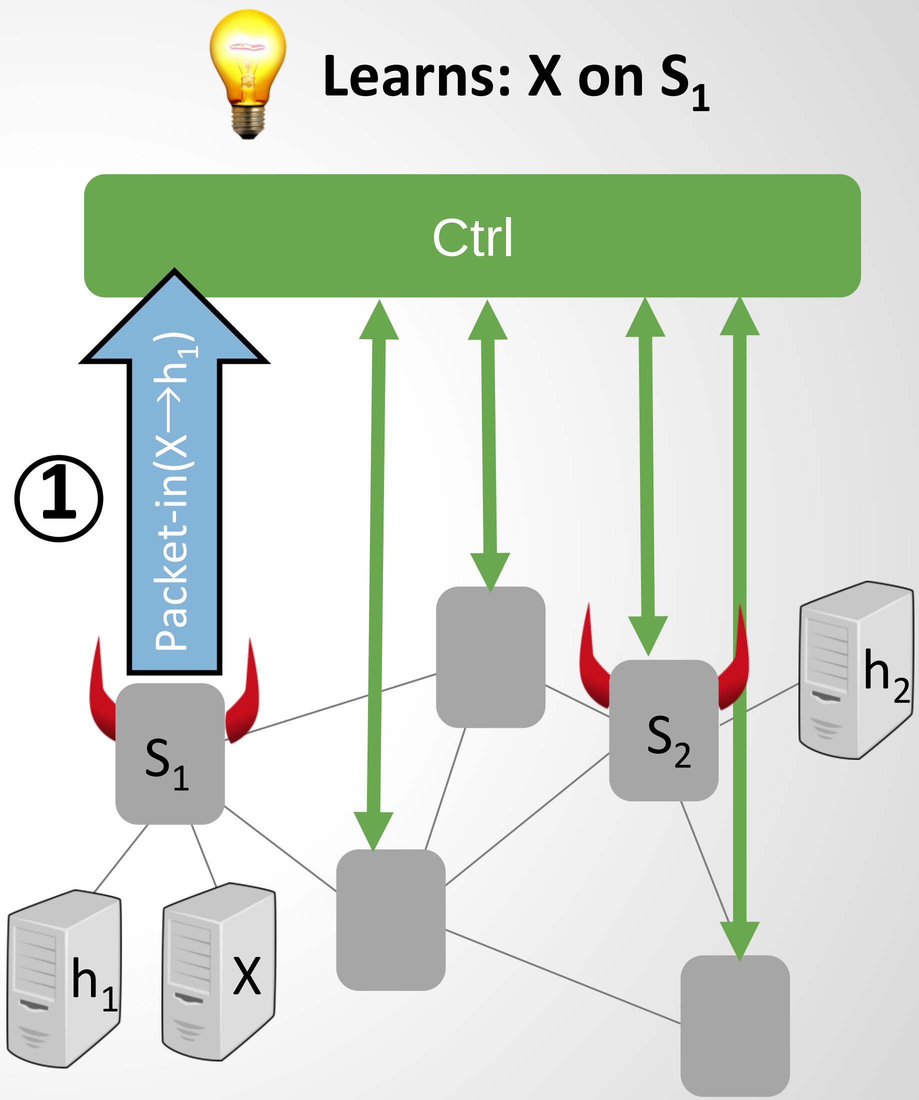


Credits: James Kempf

Teleportation: Path Update

- Controller performs MAC learning and updates paths to support mobility, VM migration, etc.
- Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch

① **S_1 announces address X**

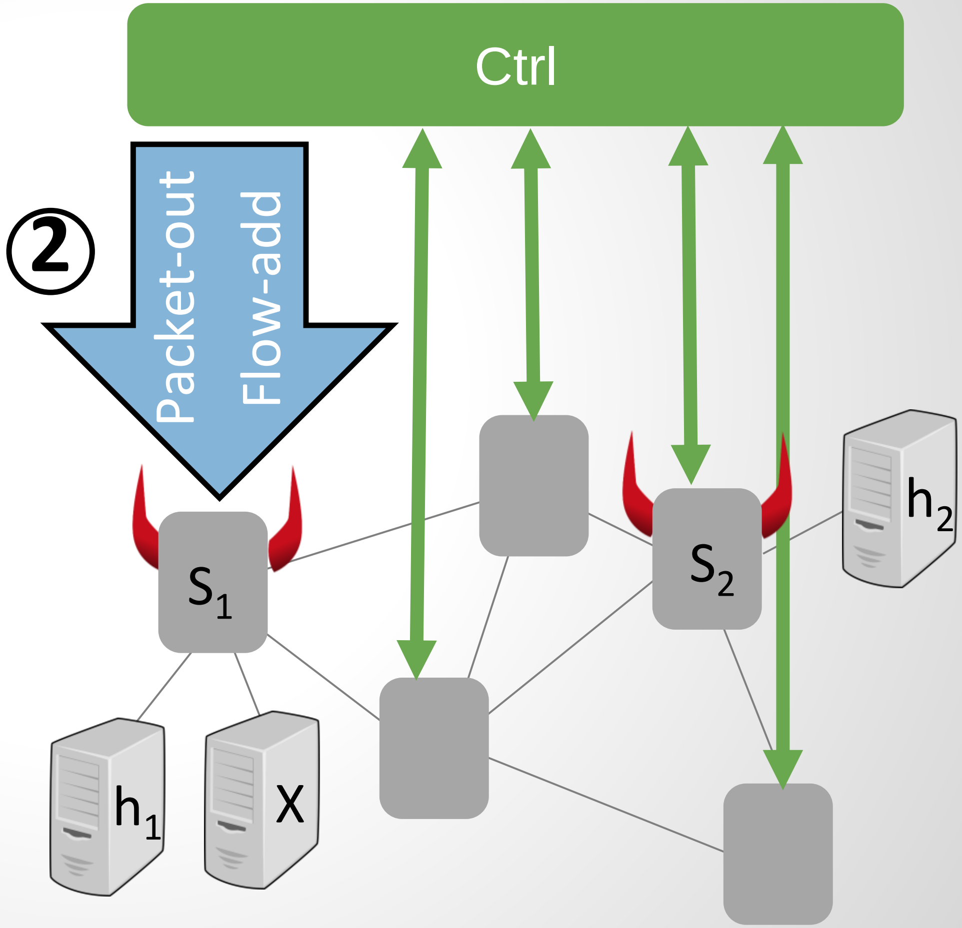


Teleportation: Path Update

- ❑ Controller performs MAC learning and updates paths to support mobility, VM migration, etc.
- ❑ Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch

① **S_1 announces address X**

② **Packet-out to h_1**
Add rule for X



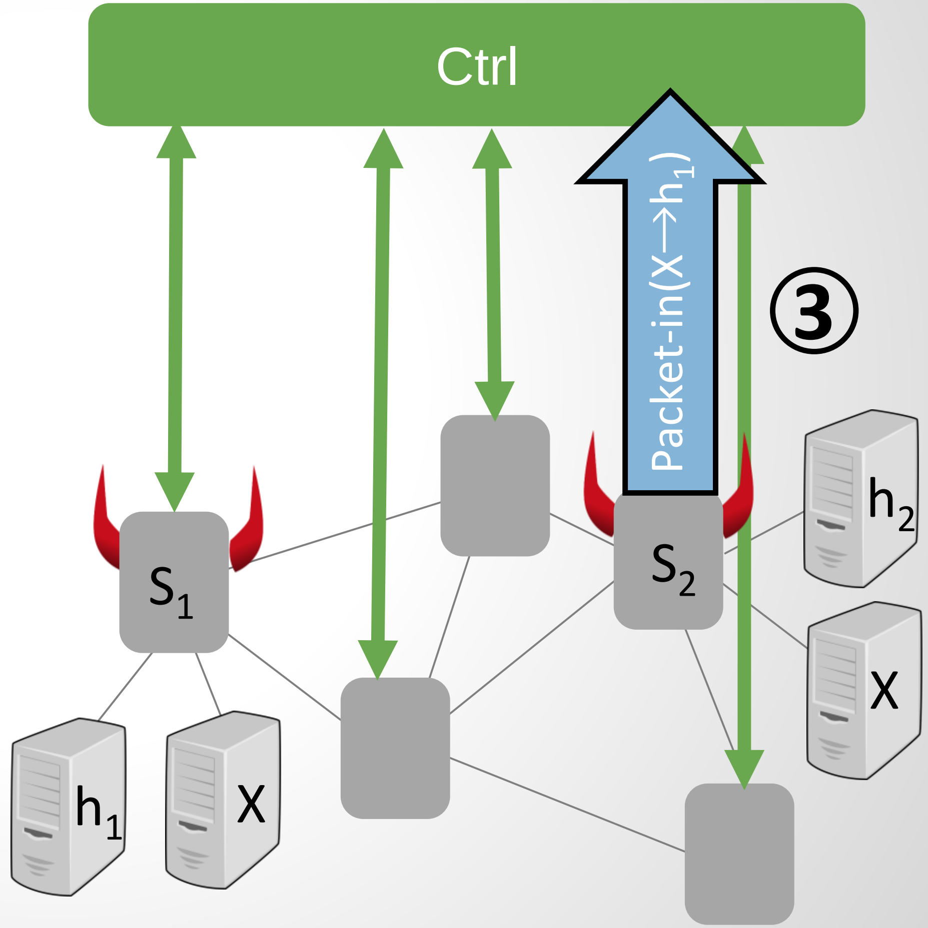
Teleportation: Path Update

- Controller performs MAC learning and updates paths to support mobility, VM migration, etc.
- Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch

① **S_1 announces address X**

② **Packet-out to h_1**
Add rule for X

③ **S_2 announces X**



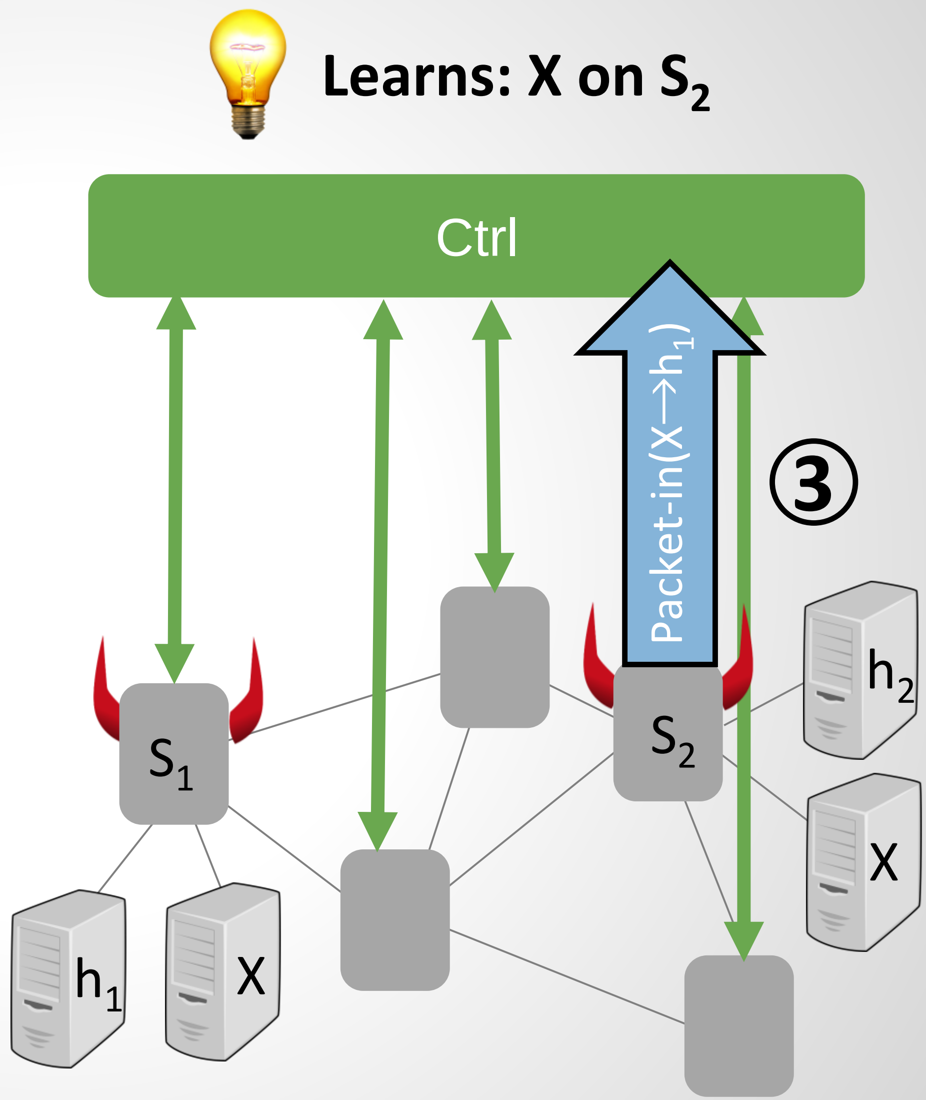
Teleportation: Path Update

- Controller performs MAC learning and updates paths to support mobility, VM migration, etc.
- Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch

① S_1 announces address X

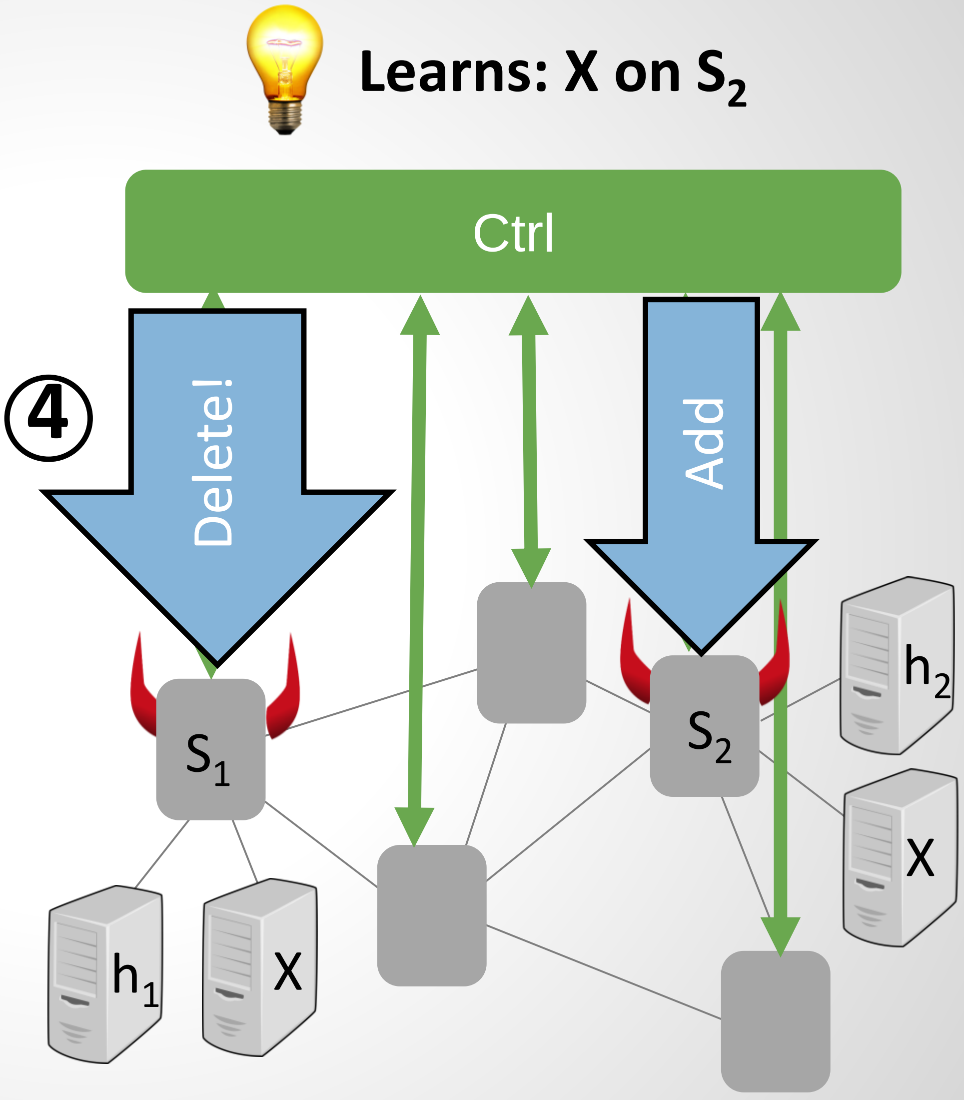
② Packet-out to h_1
Add rule for X

③ S_2 announces X



Teleportation: Path Update

- Controller performs MAC learning and updates paths to support mobility, VM migration, etc.
- Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch



① S_1 announces address X

② Packet-out to h_1
Add rule for X

③ S_2 announces X

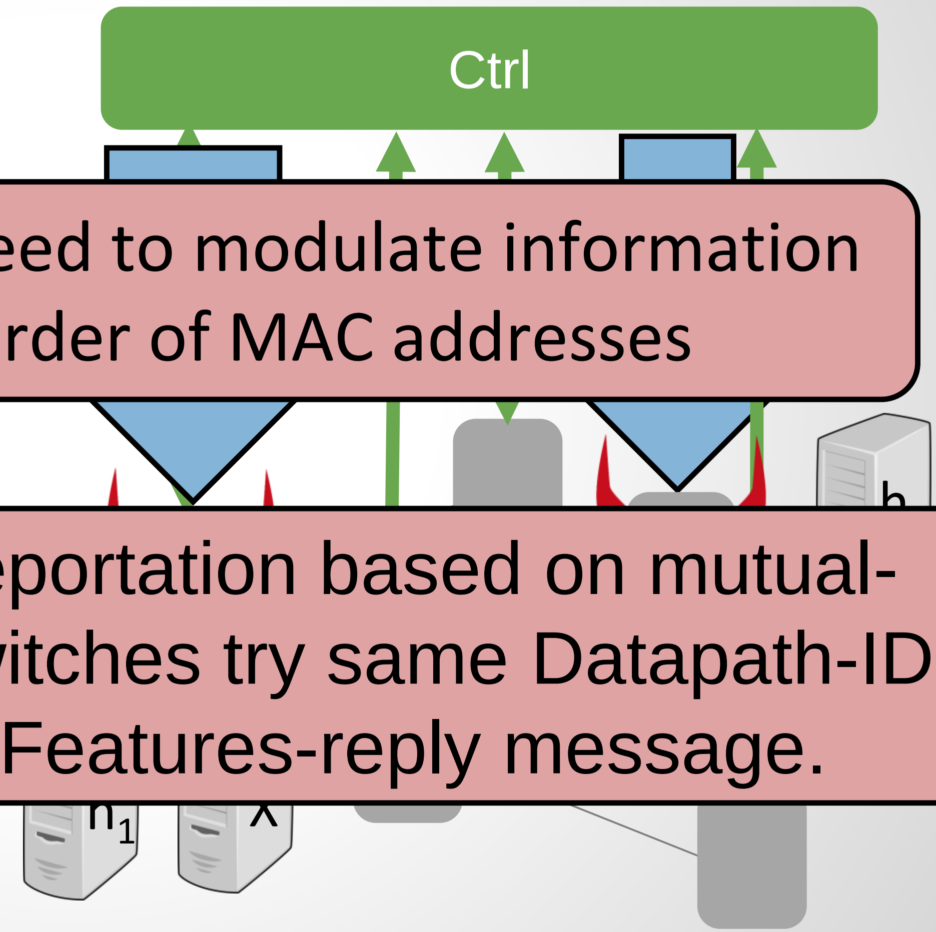
④ Flow delete

Teleportation: Path Update

Learns: X on S_2

- Controller performs MAC learning and updates paths to support mobility, VM migration, etc.

removes on old switch



Admittedly very implicit: need to modulate information e.g., using timing or order of MAC addresses

① S_1

Pa

② Ac

Similar implicit teleportation based on mutual-exclusion. E.g., switches try same Datapath-ID (DPID) field in the Features-reply message.

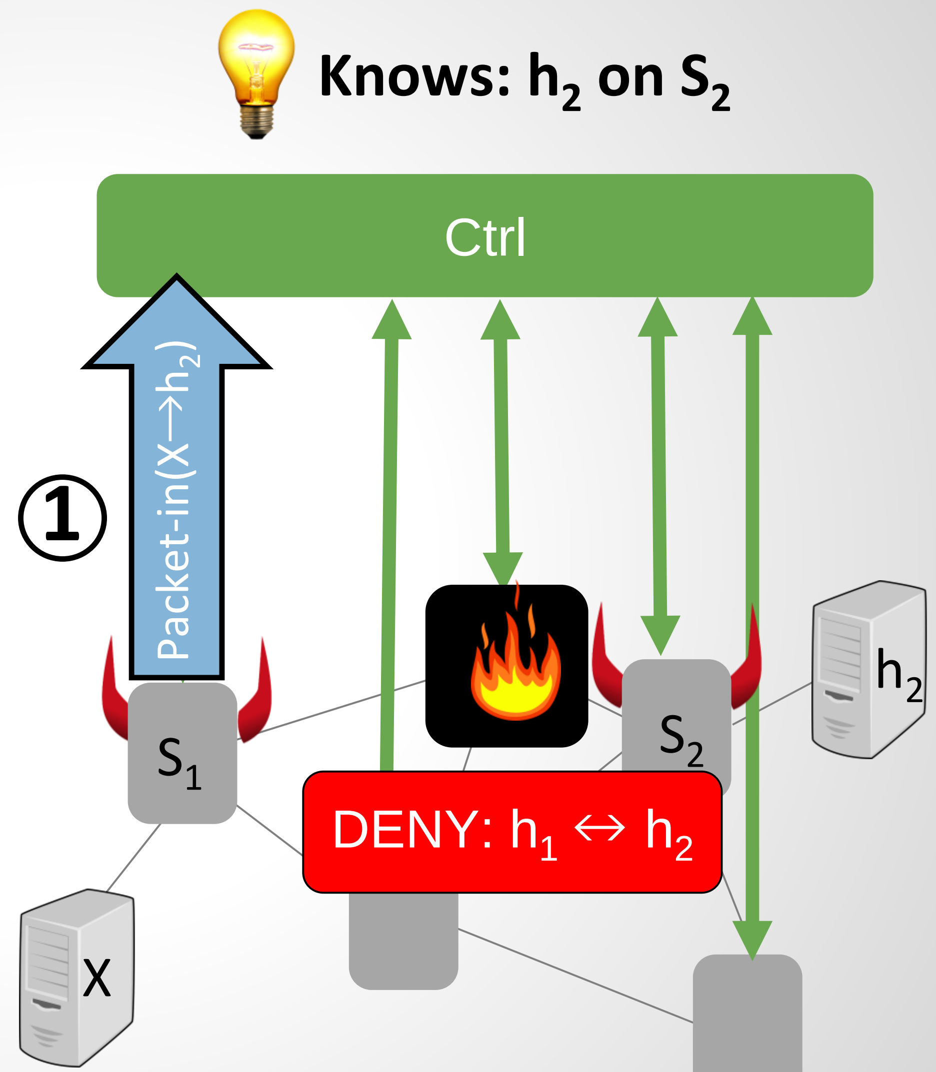
③ S_2 announces X

④ Flow delete

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**

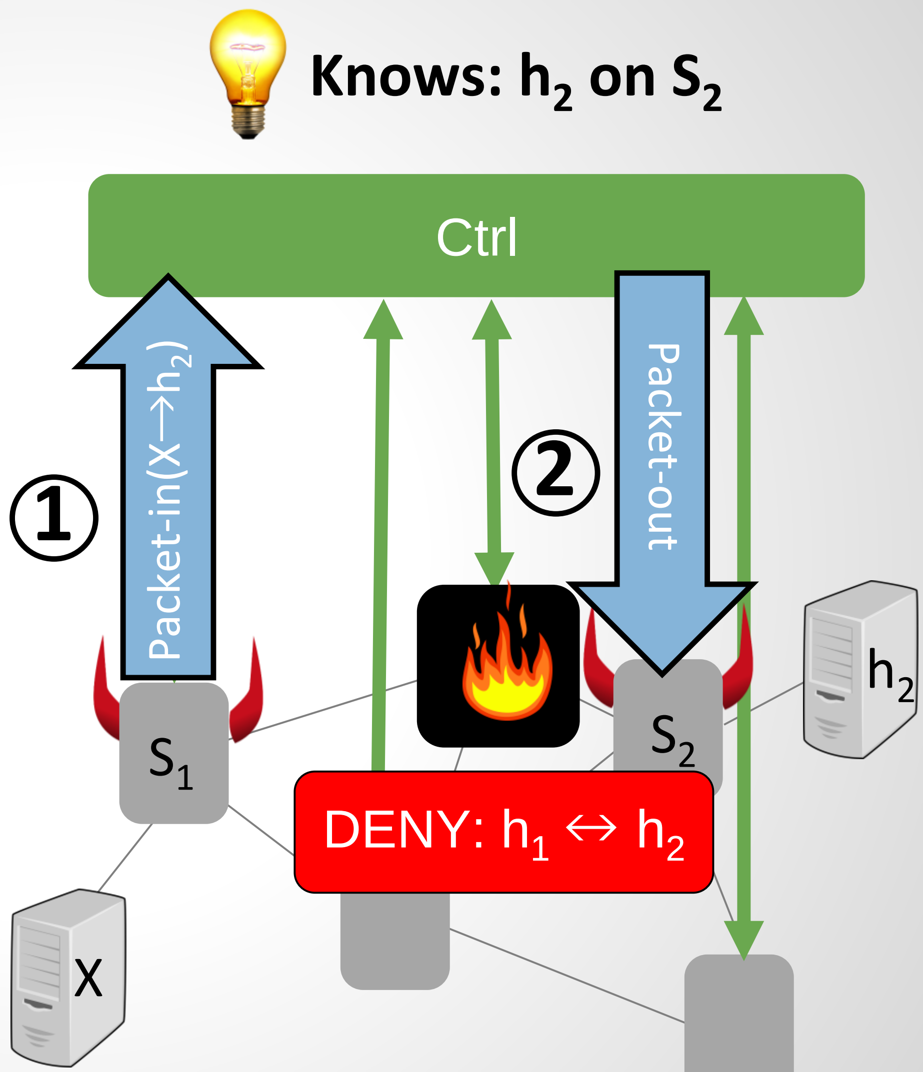


Teleportation: Out-of-Band Forwarding

- 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**



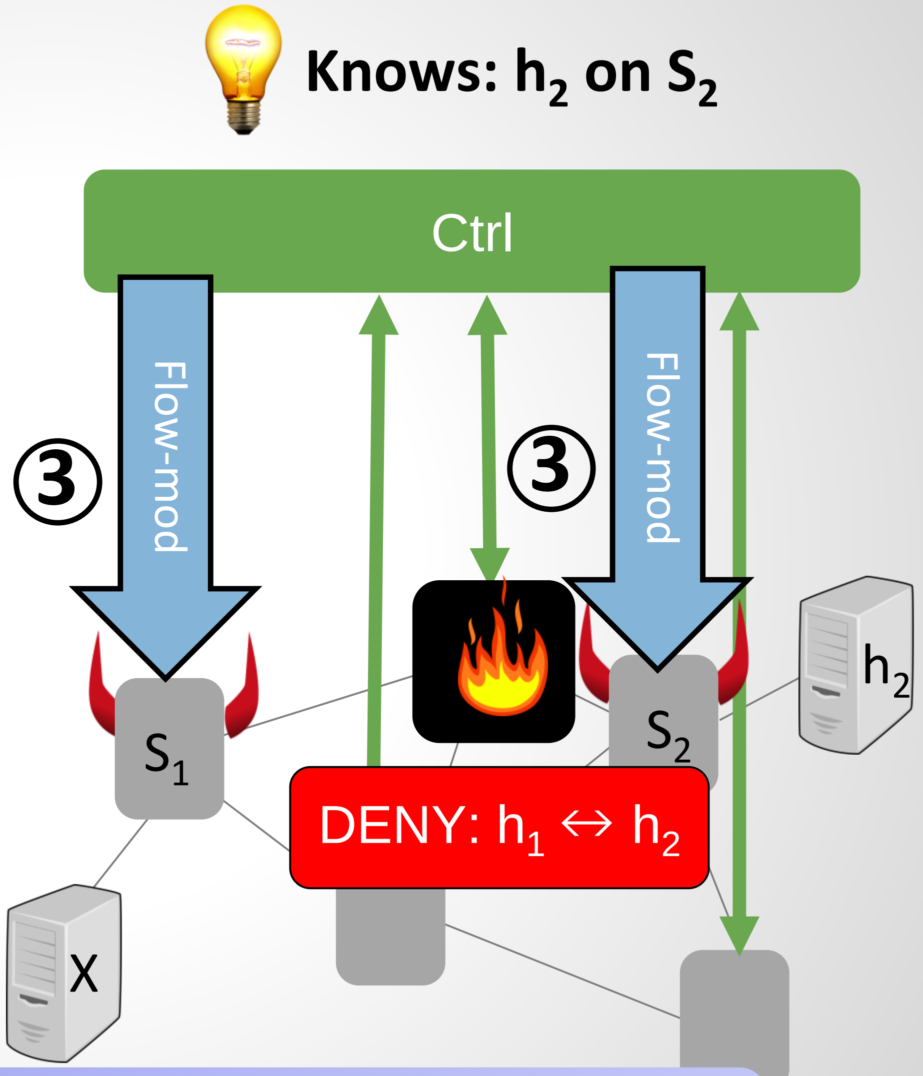
Teleportation: Out-of-Band Forwarding

- 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**



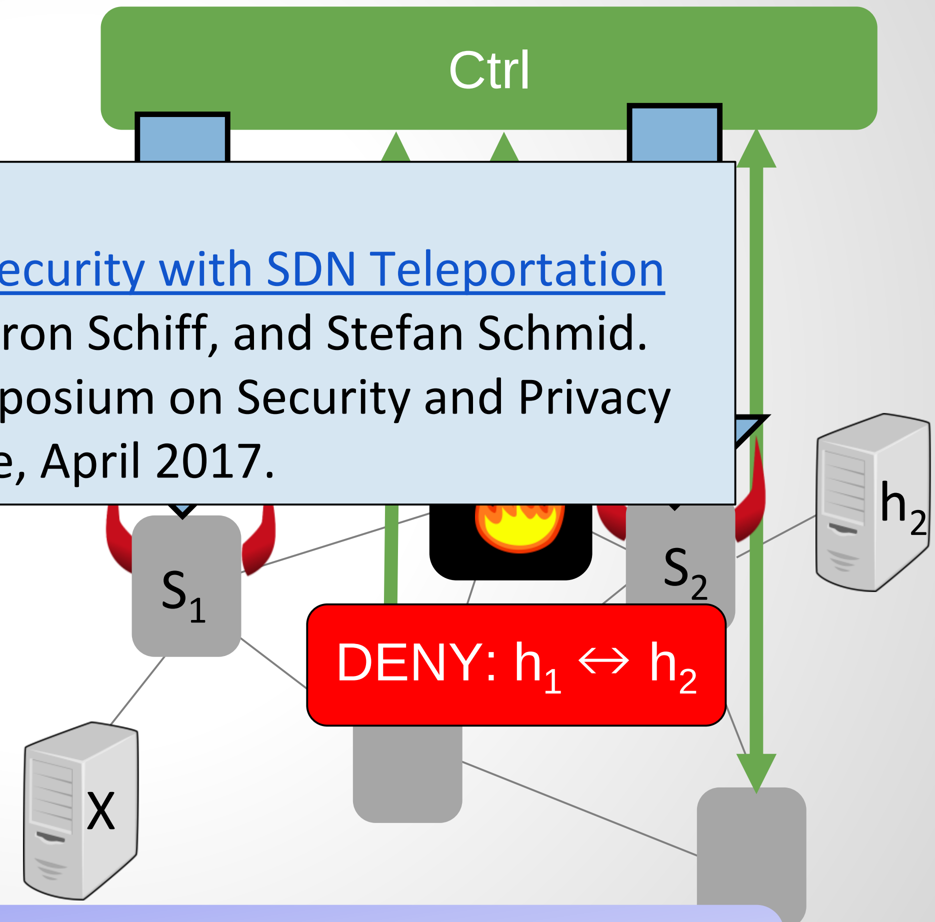
Establish path through firewall: no more packet-ins, blocked. (But could use another MAC address next time.)

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.



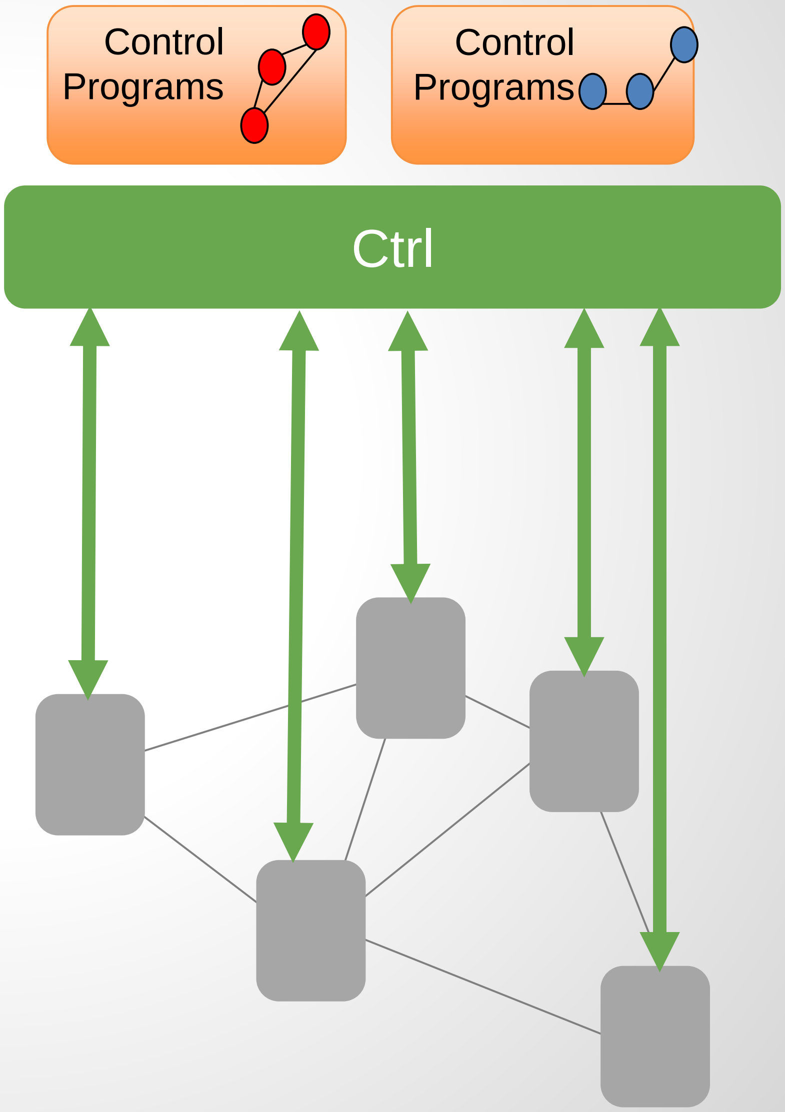
① **Packet-in**

② **Packet-out**

③ **Flow-mod**

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

A Mental Model for This Talk



Challenge: centralization!

Despite centralization: SDN stays a distributed system!

Challenge: Controller may miss events

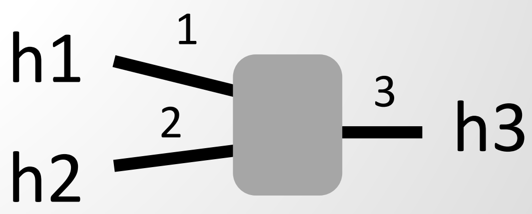
- Basic task: MAC learning

- 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

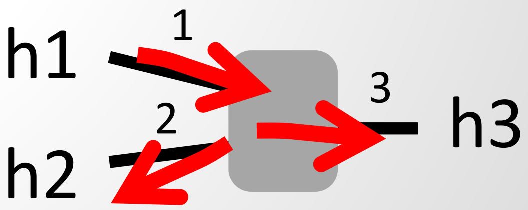
Challenge: Controller may miss events

- Basic task: 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:



Challenge: Controller may miss events

- Basic task: 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**

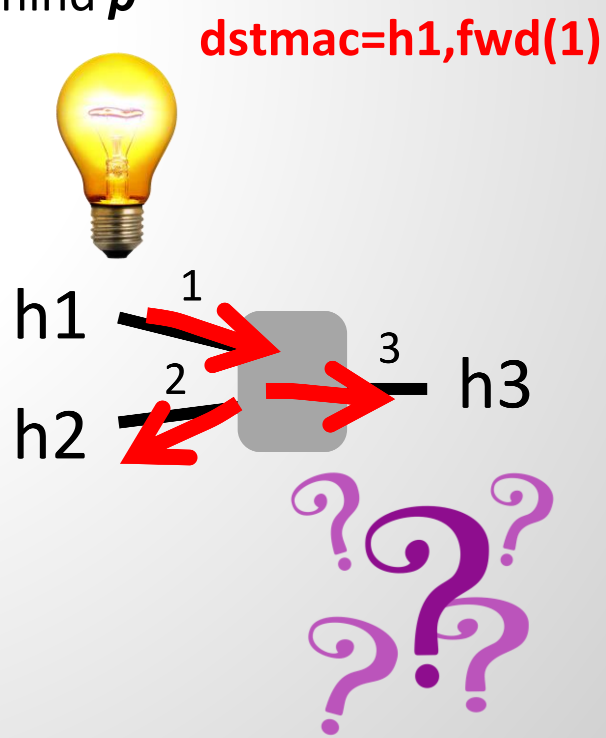


Challenge: Controller may miss events

- Basic task: 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)**

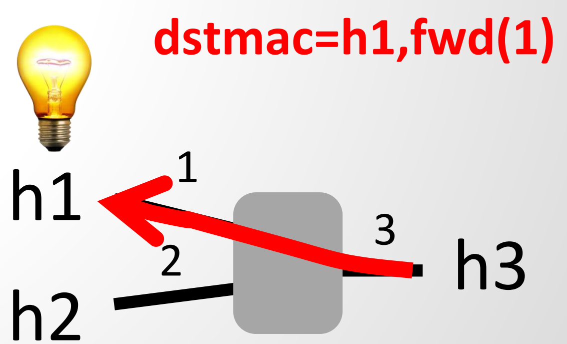


Challenge: Controller may miss events

- Basic task: 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**



Challenge: Controller may miss events

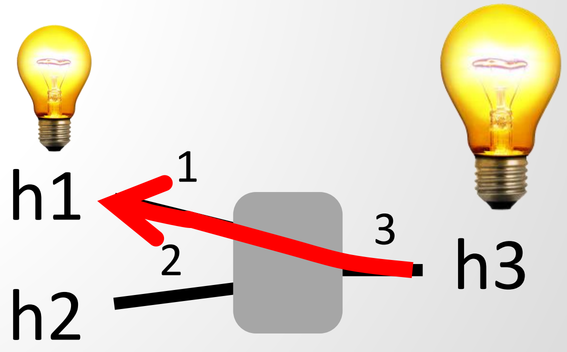
- Basic task: MAC learning

- 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)**



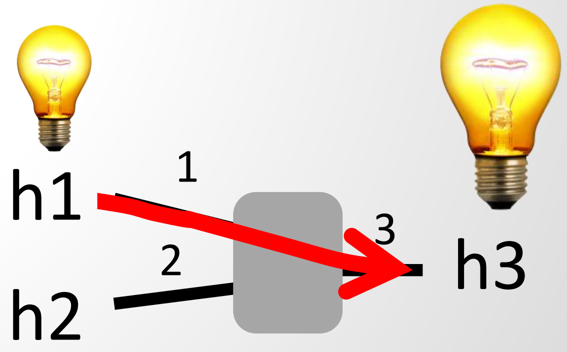
Challenge: Controller may miss events

- Basic task: 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)

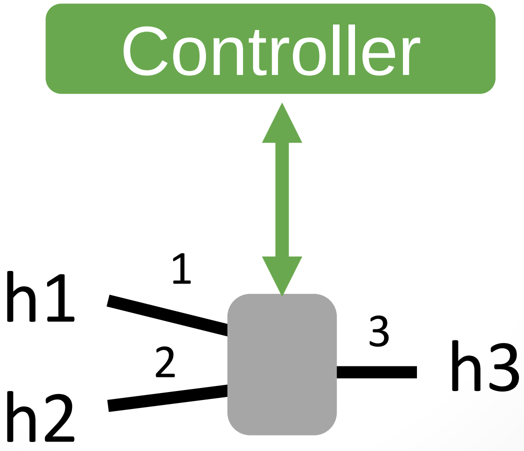
dstmac=h3,fwd(3)

- 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**

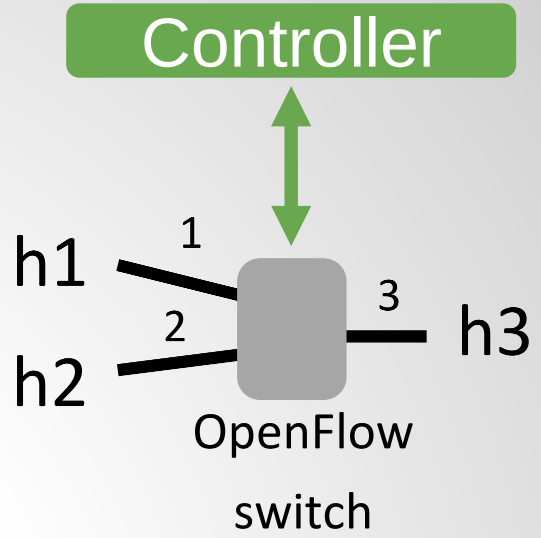


Challenge: Controller may miss events

How to implement this behavior in SDN?



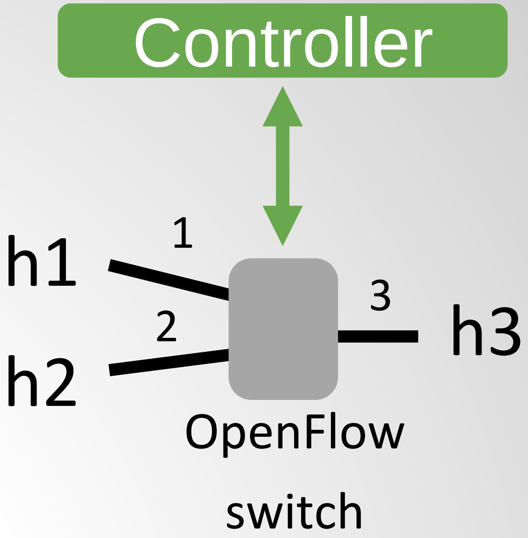
Example: SDN MAC Learning Done Wrong



- Initially table: Send everything to controller

Pattern	Action
*	send to controller

Example: SDN MAC Learning Done Wrong

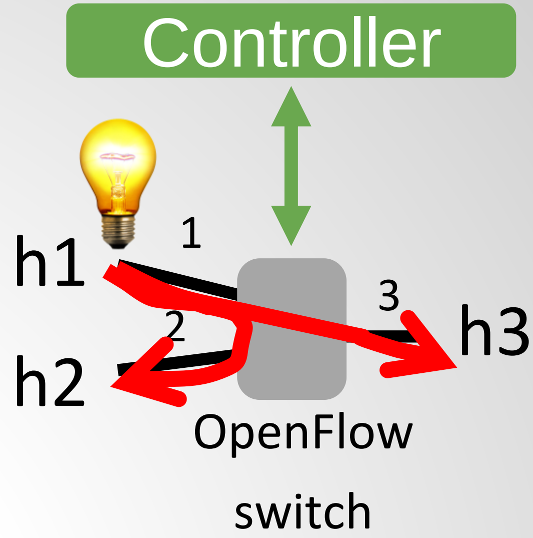


- Initially table: Send everything to controller

Pattern	Action
*	send to controller

- When h1 sends to h2:

Example: SDN MAC Learning Done Wrong



- Principle: only send to ctrl if destination unknown

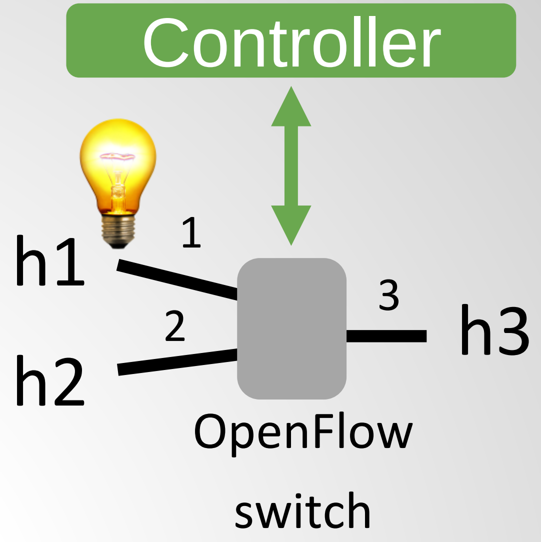
Pattern	Action
*	send to controller

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

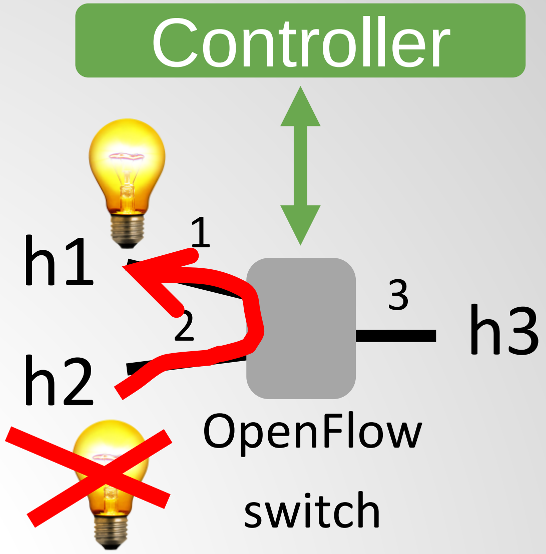


- Principle: only send to ctrl if destination unknown

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

- Now assume h2 sends to h1:

Example: SDN MAC Learning Done Wrong

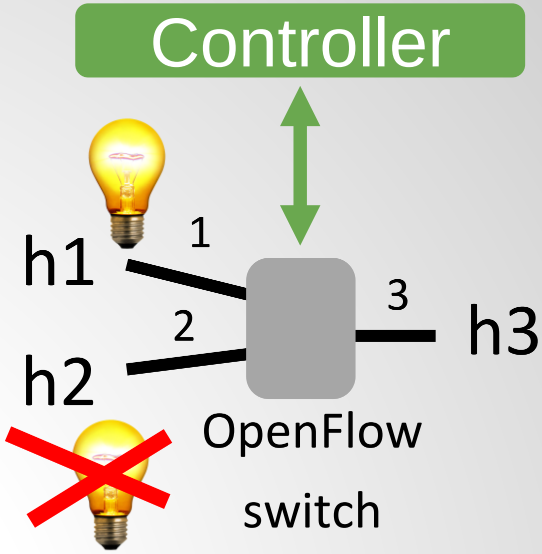


- Principle: only send to ctrl if destination unknown

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

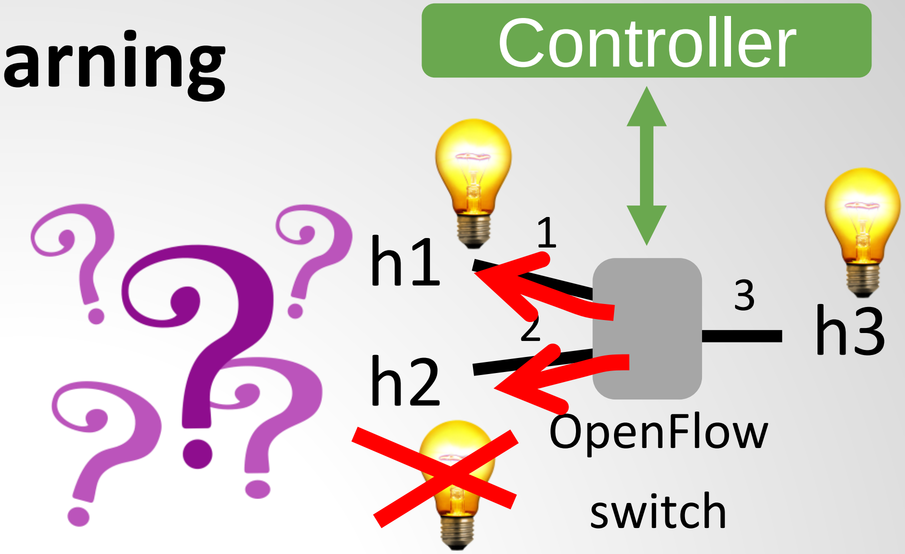


- Principle: only send to ctrl if destination unknown

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

- Now, when h3 sends to h2:

Example: SDN MAC Learning Done Wrong



- Principle: only send to ctrl if destination unknown

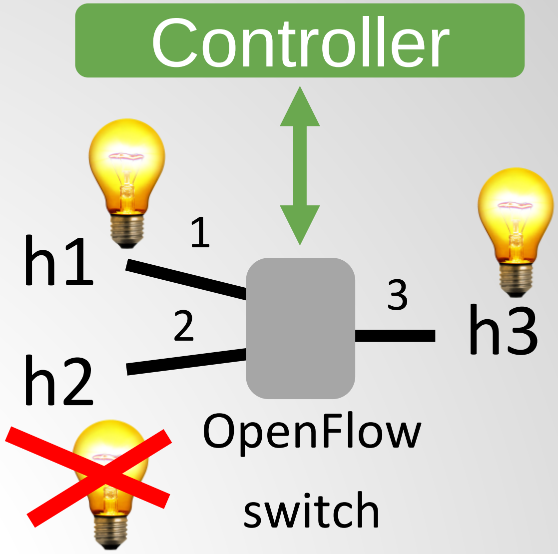
Pattern	Action
dstmac=h1	Forward(1)
*	send to controller

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

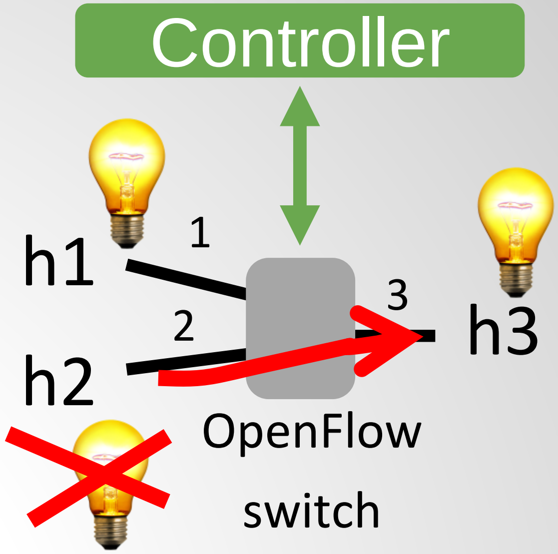


- Principle: only send to ctrl if destination unknown

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

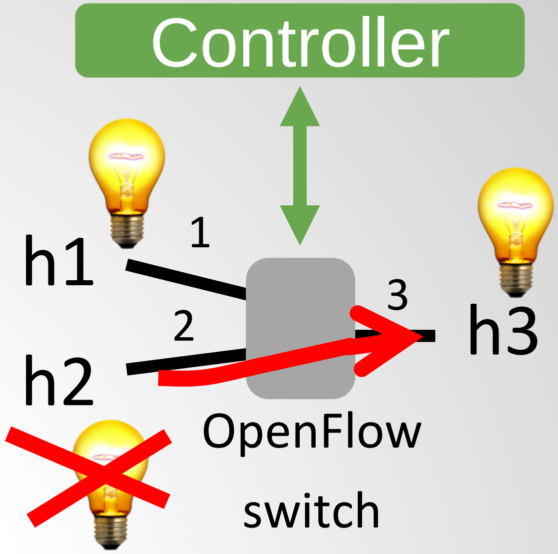


- Principle: only send to ctrl if destination unknown

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

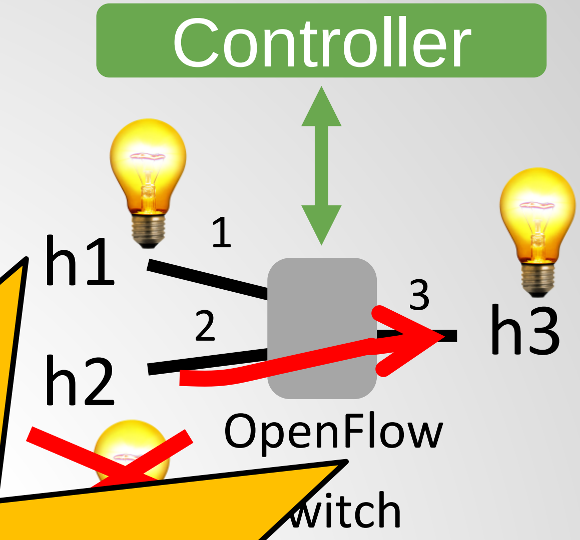


- Principle: only send to ctrl if destination unknown

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



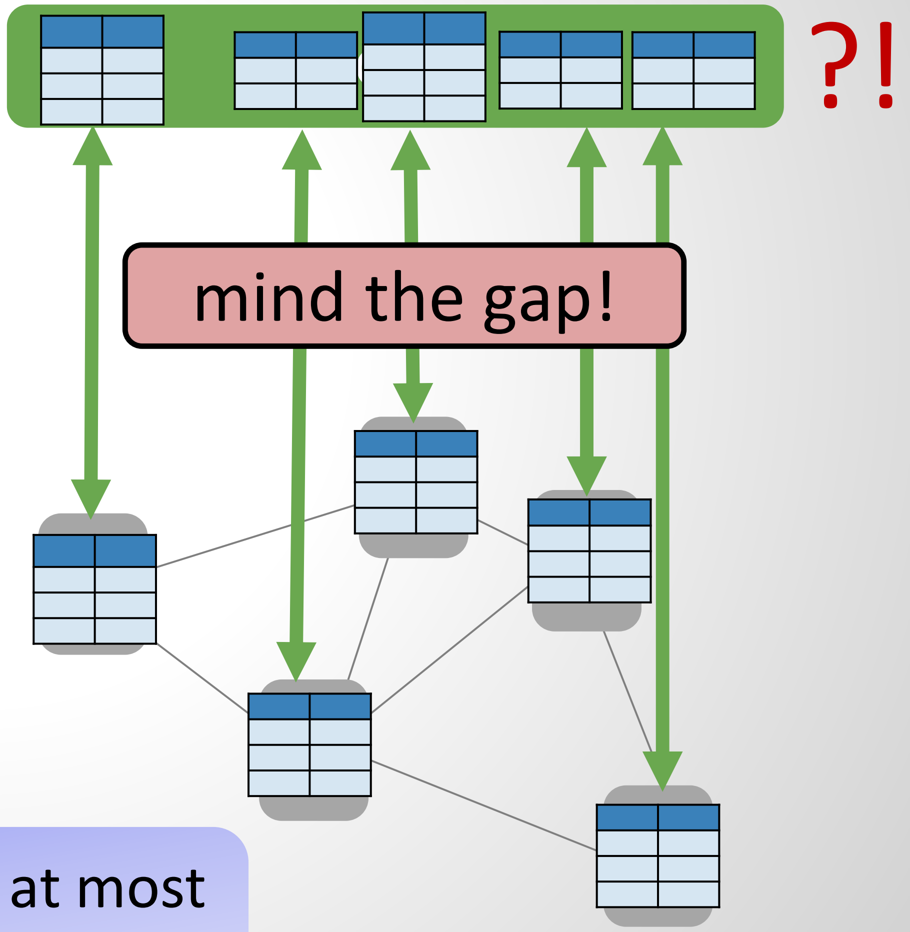
- Principle: only send to ctrl if d… ination unkno…

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.

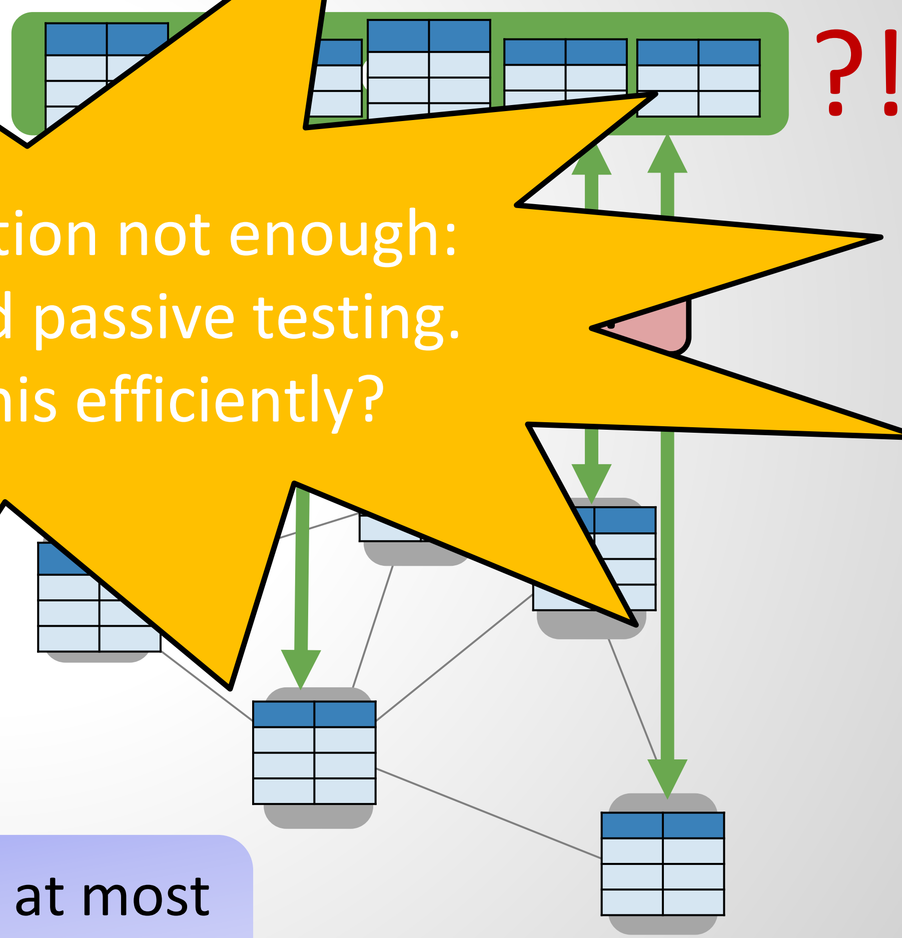


A ***problem*** because ***like in security***: at most as consistent as least consistent part!

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

- ❑ Rules inserted using switch CLI
- ❑ Operator m
- ❑ Softwa
- ❑ Updates that hav
  acknowledged wrongfully
- ❑ Malicious behavior, etc.

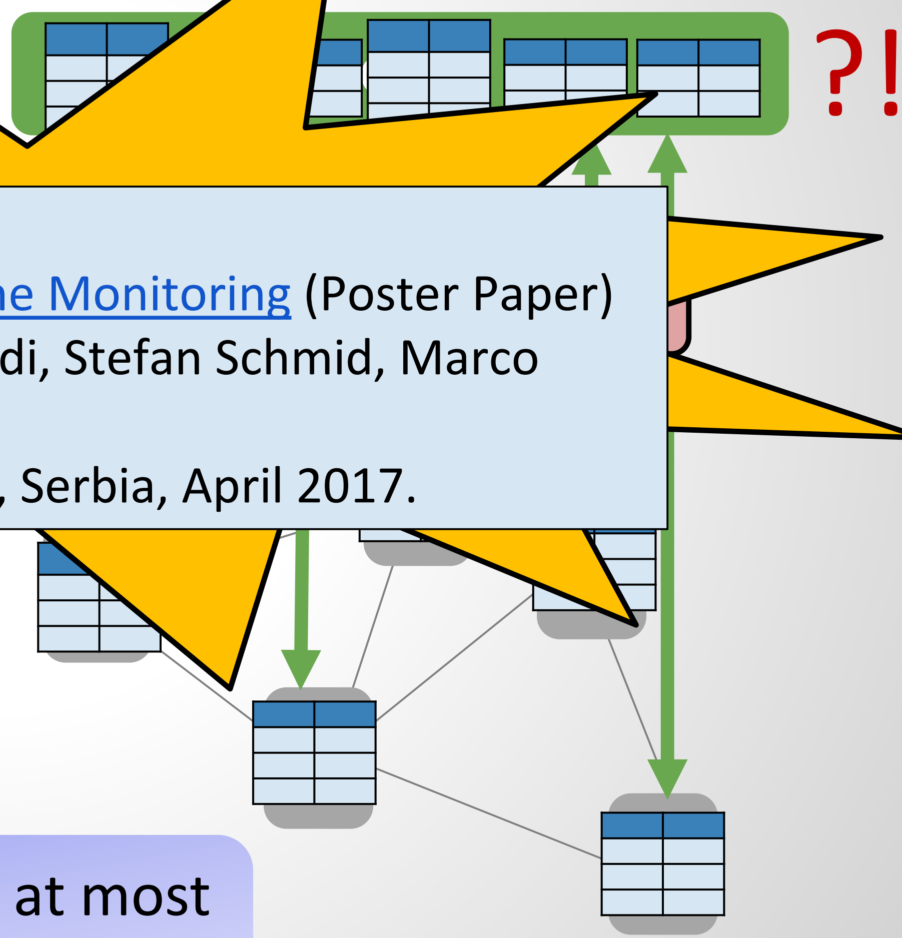
Logical verification not enough: need active and passive testing. How to do this efficiently?



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

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

- ❑ Rules inserted using switch CLI
- ❑ Operator m...
- ❑ Softwa...
- ❑ Update... acknowledged wrongfully
- ❑ Malicious behavior, etc.



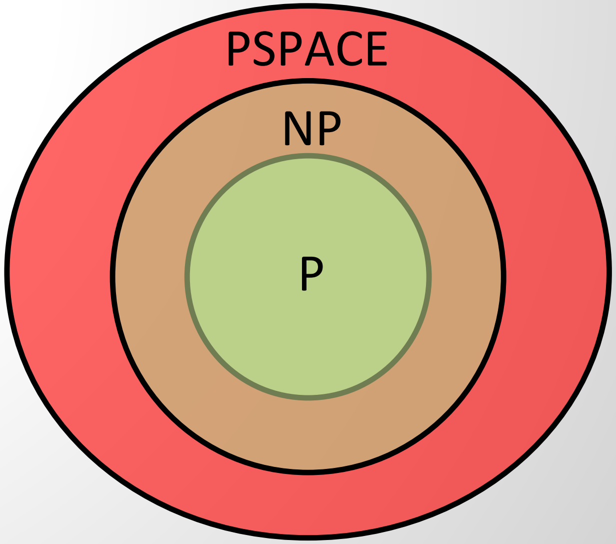
Further reading:
Towards Meticulous Data Plane Monitoring (Poster Paper)
Apoorv Shukla, Said Jawad Saidi, Stefan Schmid, Marco Canini, and Anja Feldmann.
EuroSys PhD Forum, Belgrade, Serbia, April 2017.

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

Bad News: Automated Testing and Verification Can Be Non-Trivial!

- Seamingly simple ***reachability questions*** are hard in SDN:
  - «Is it possible to reach egress port y from ingress port x for certain header spaces?»
  - Or what-if-analysis: «What is reachability matrix if there are *f* link failures?

- Tools like NetKat, UPPAAL, …: PSPACE complete, tools like wNetKAT can even encounter undecidability!

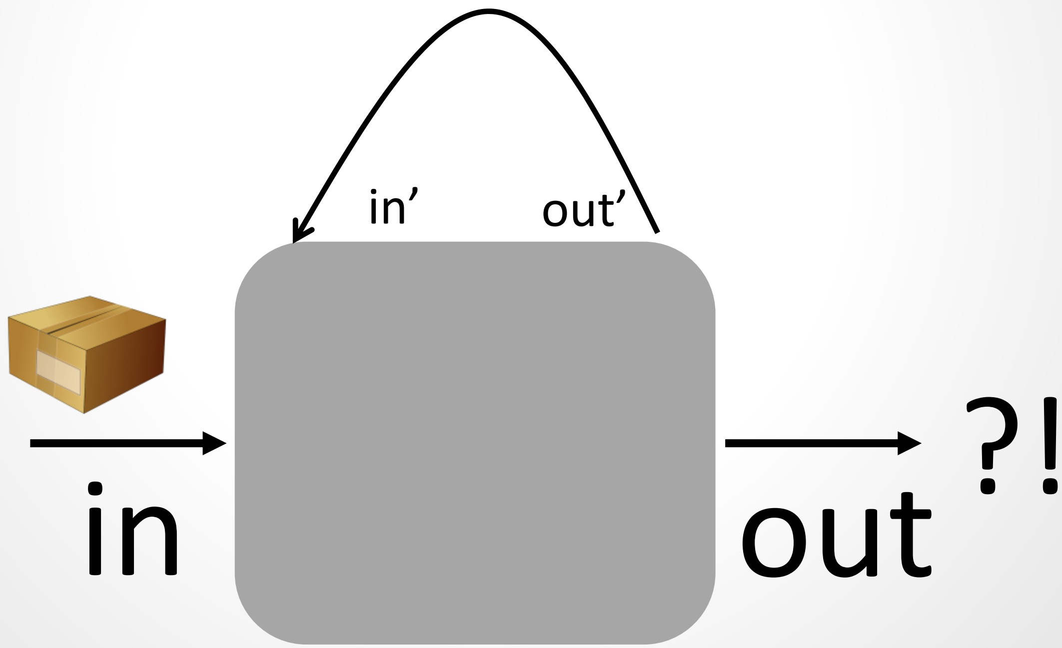


- … and this is only on the logical level and for stateless data planes!
  - Still need to actually test dataplane consistency (e.g., using packet generation)
  - What if dataplane is stateful?

Tractability of Automation/Verification

Even without failures: reachability test is undecidable in SDN!

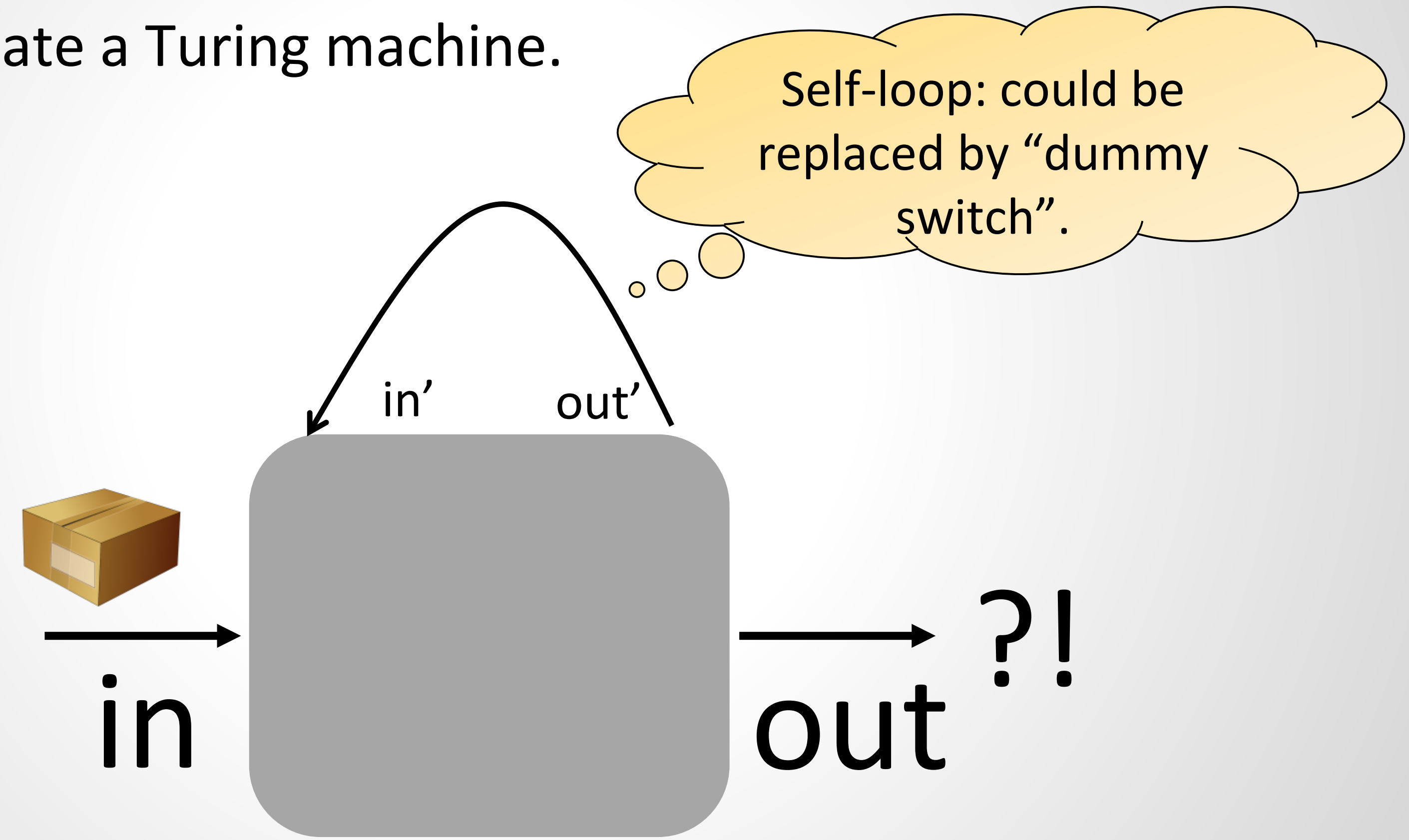
Proof: Can emulate a Turing machine.



Tractability of Automation/Verification

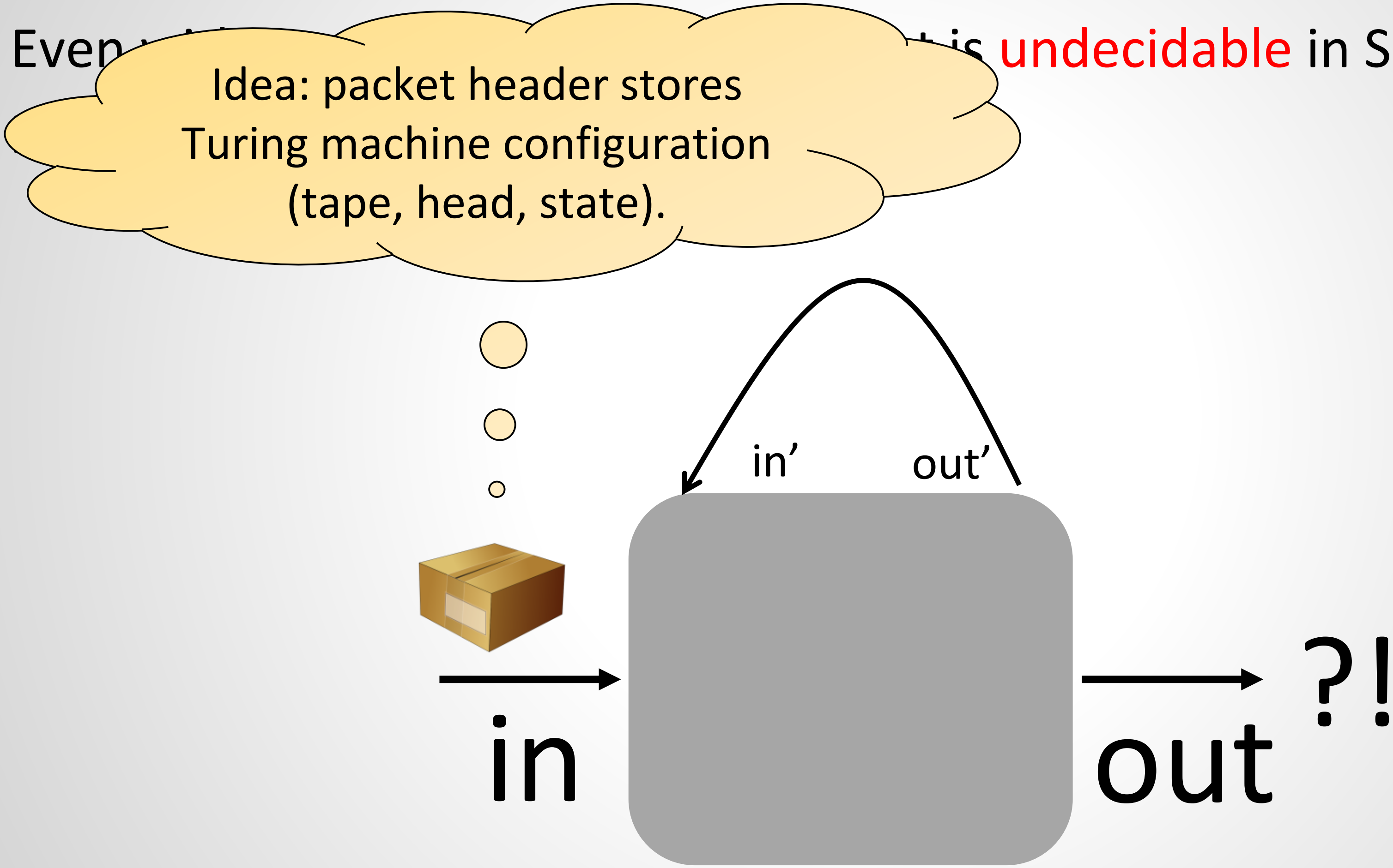
Even without failures: reachability test is undecidable in SDN!

Proof: Can emulate a Turing machine.



Tractability of Automation/Verification

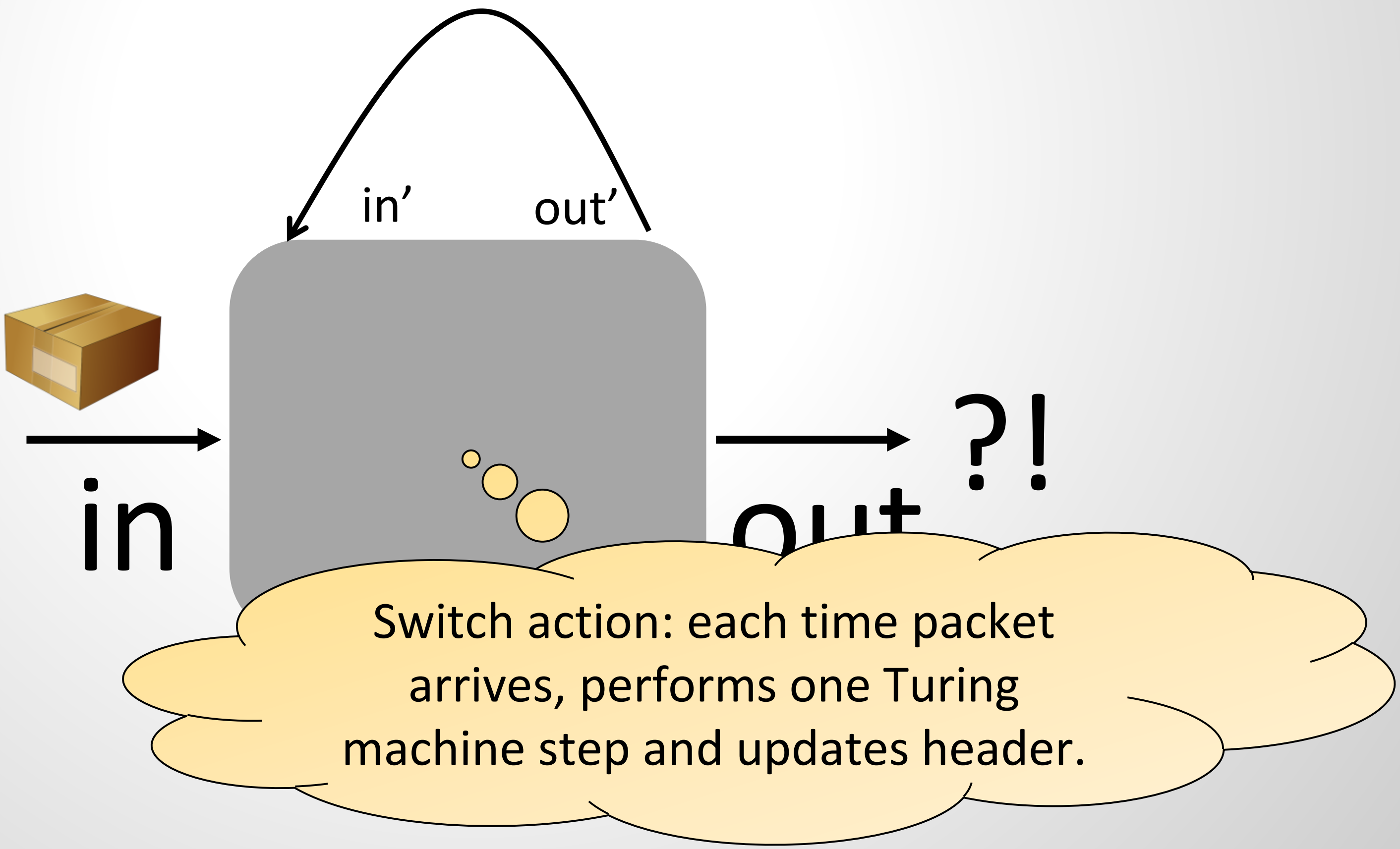
Even ... t is undecidable in SDN!



Tractability of Automation/Verification

Even without failures: reachability test is undecidable in SDN!

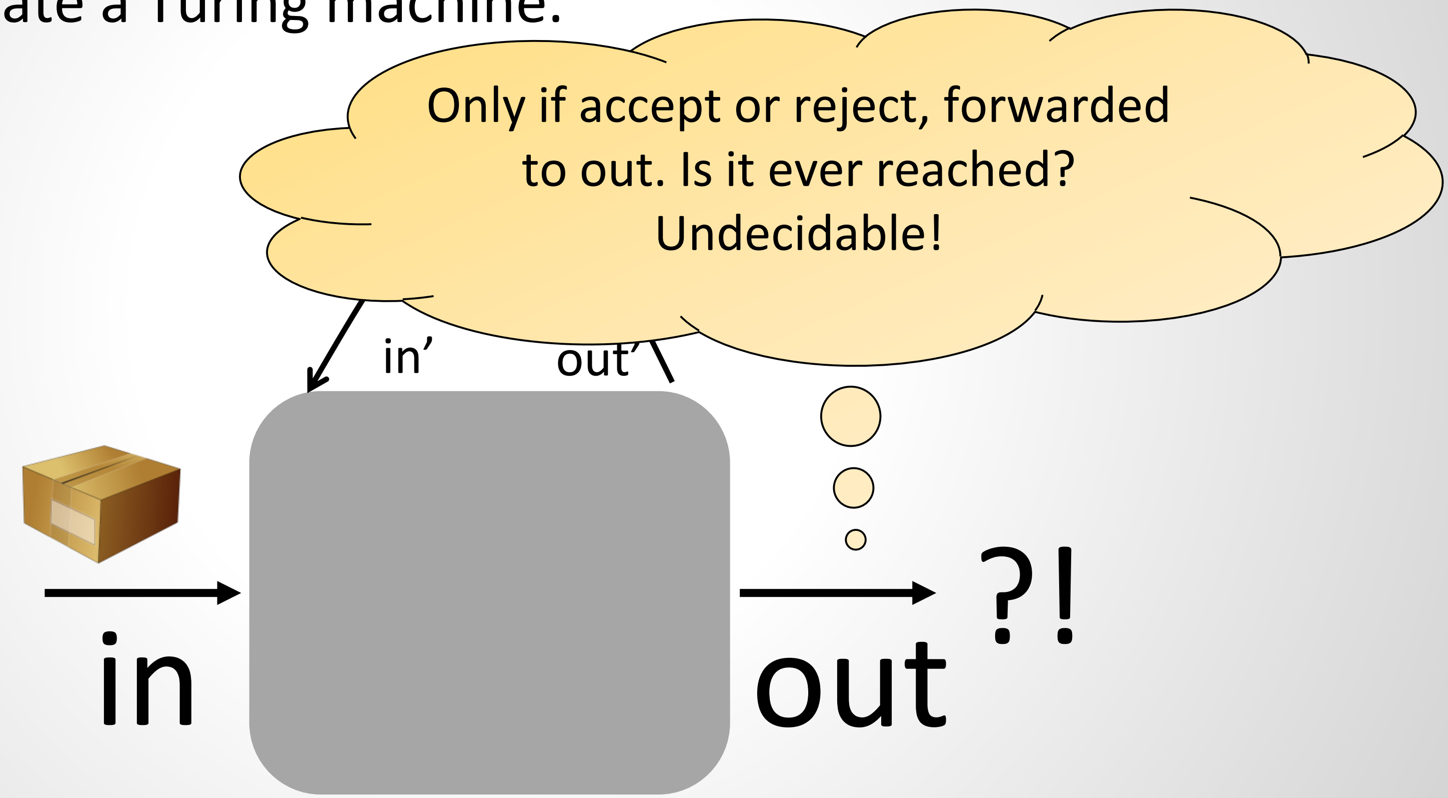
Proof: Can emulate a Turing machine.



Tractability of Automation/Verification

Even without failures: reachability test is undecidable in SDN!

Proof: Can emulate a Turing machine.

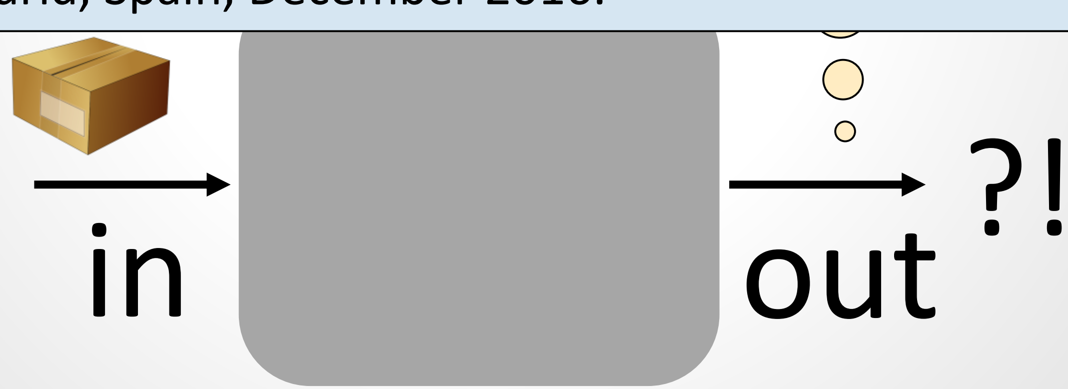


Tractability of Automation/Verification

Even without failures: reachability test is undecidable in SDN!

Proof: Can emulate a Turing machine.

Further reading:
[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.

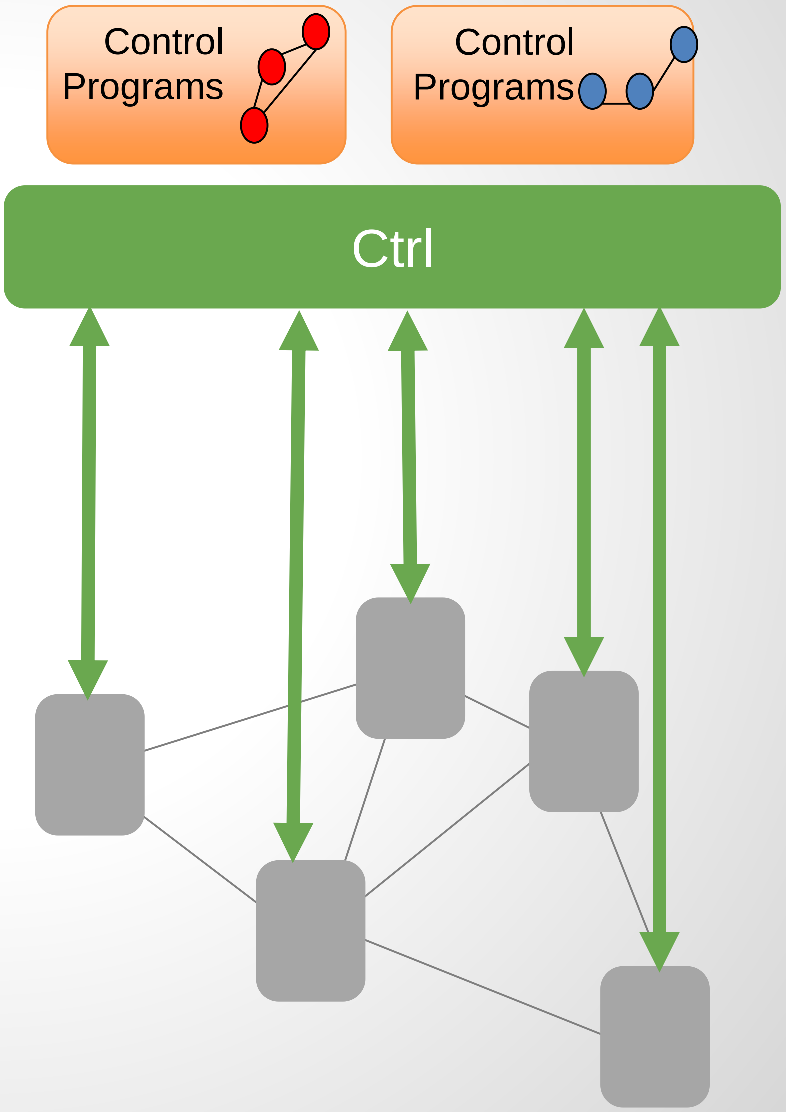


Many Open Research Questions

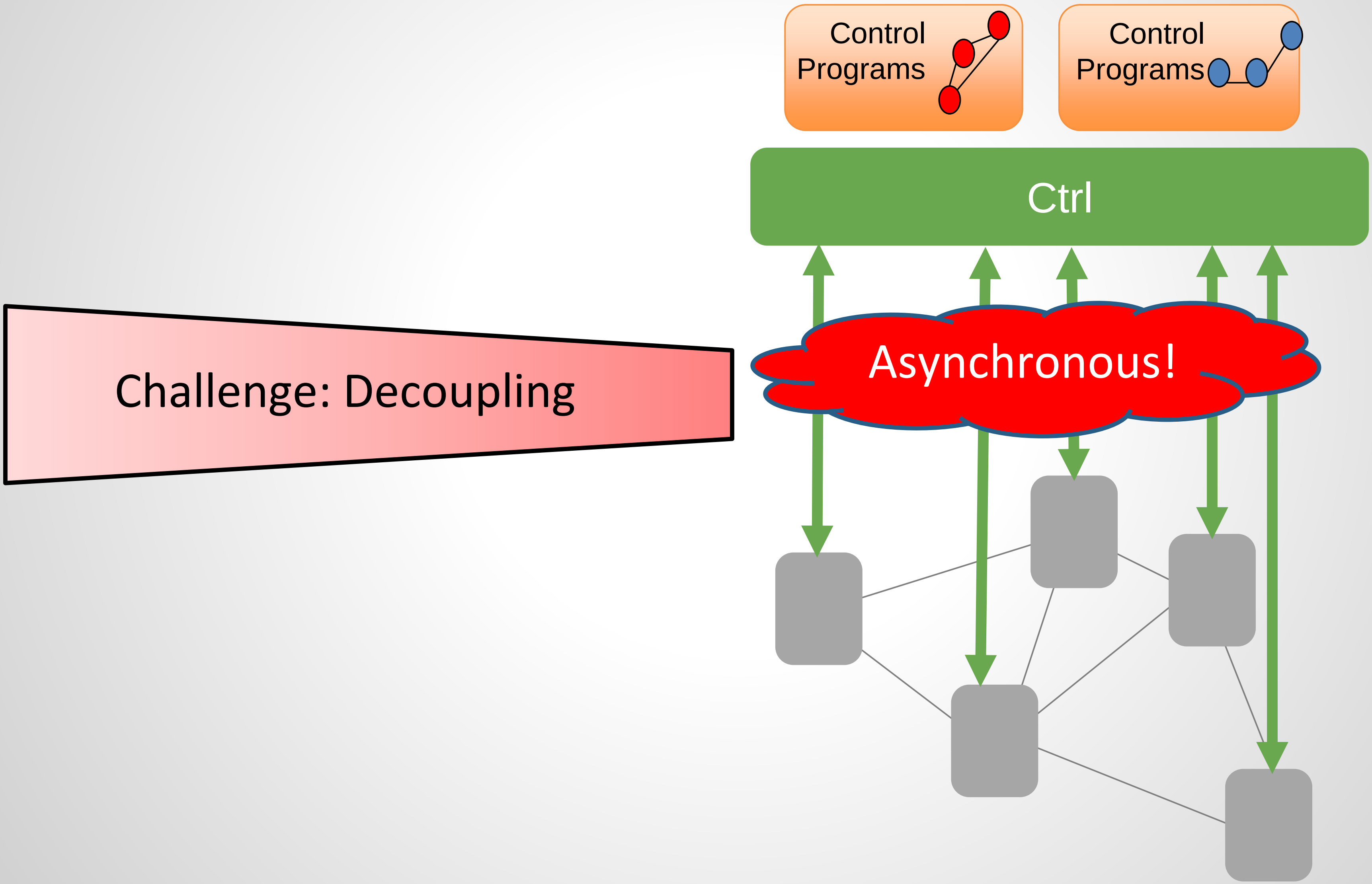
- Tradeoff expressiveness of rule and verification complexity?
- Is it worth using less general rules so fast (automated) verification is possible?
- Example: MPLS is not hard to verify!
- What about more programmable and stateful dataplanes?

A Mental Model for This Talk

Challenge: Decoupling

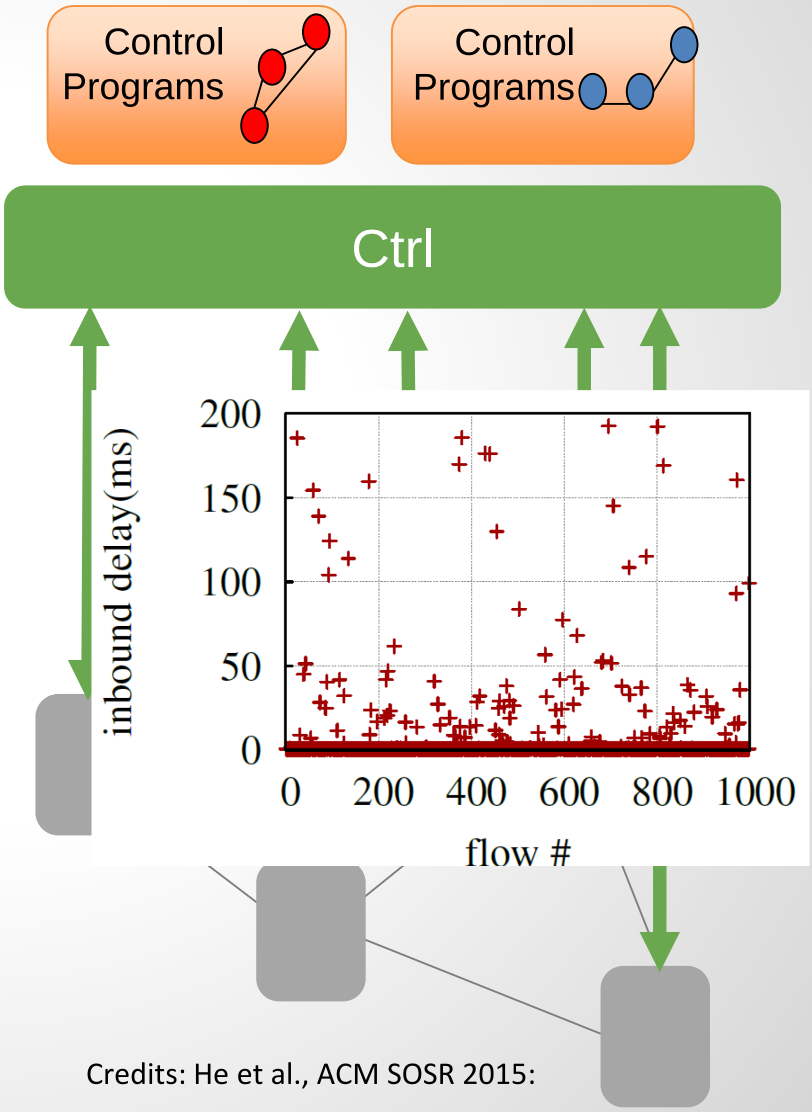


A Mental Model for This Talk



A Mental Model for This Talk

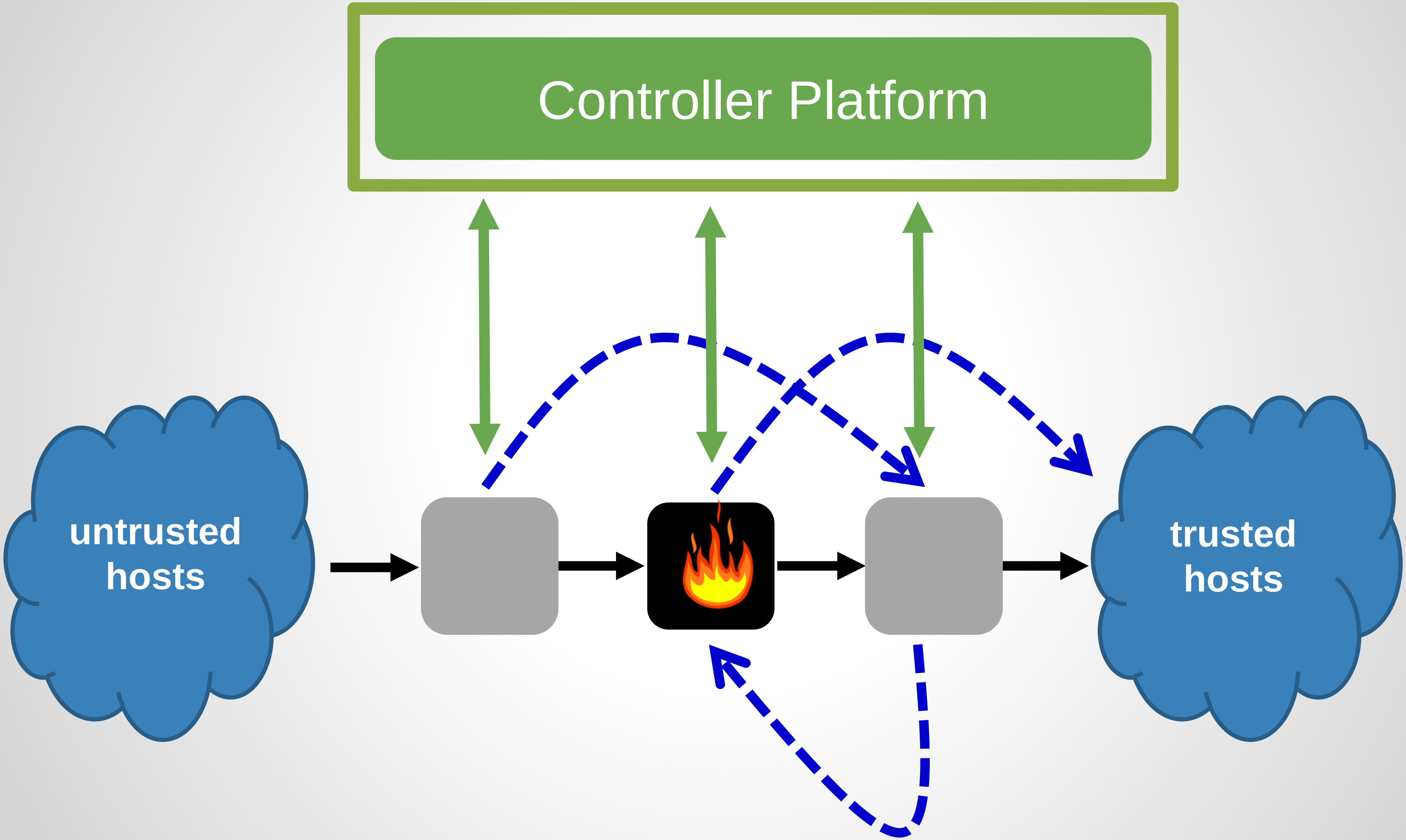
Challenge: Decoupling



Credits: He et al., ACM SOSR 2015: without network latency

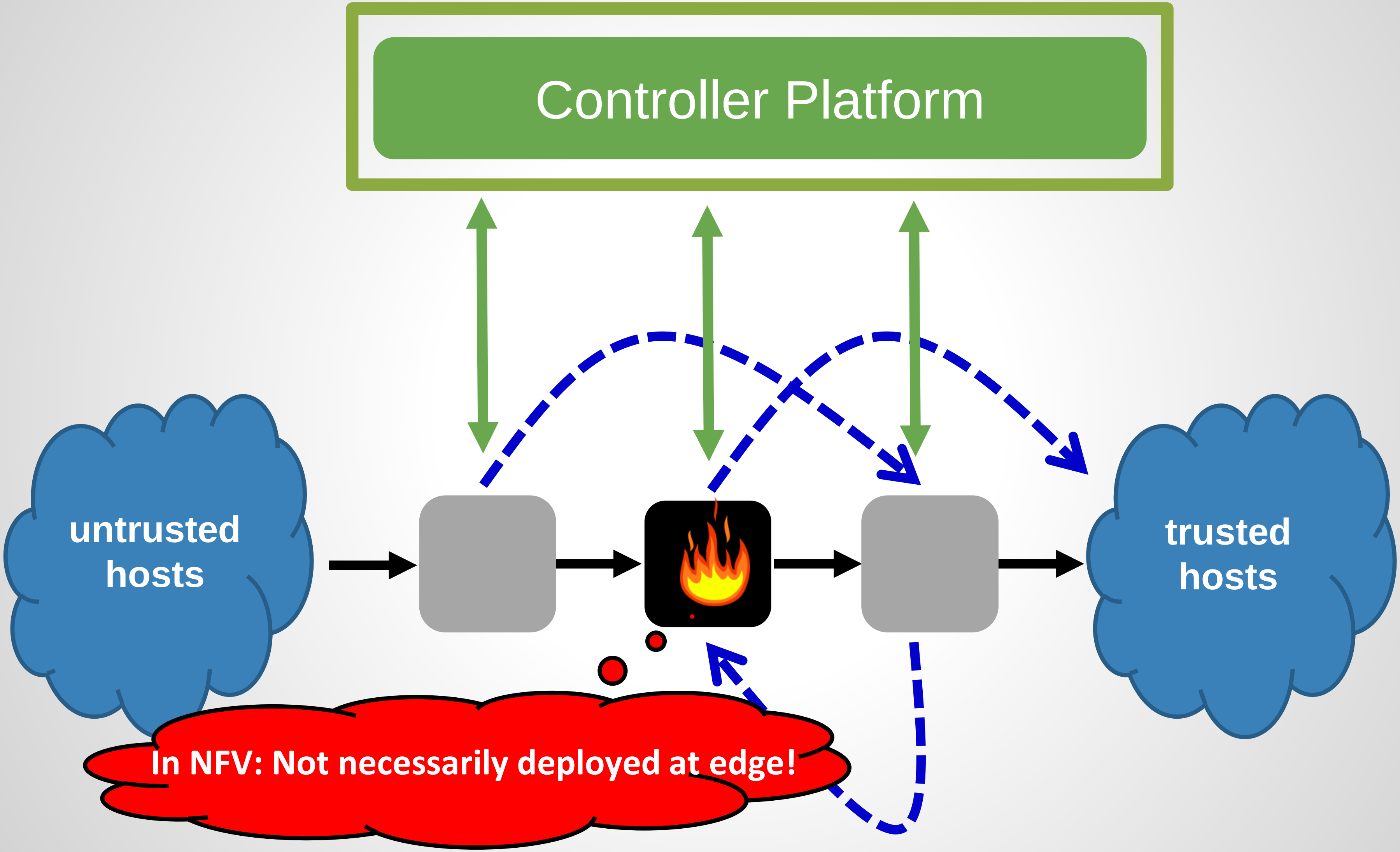
Despite centralization: SDN stays a distributed system!

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



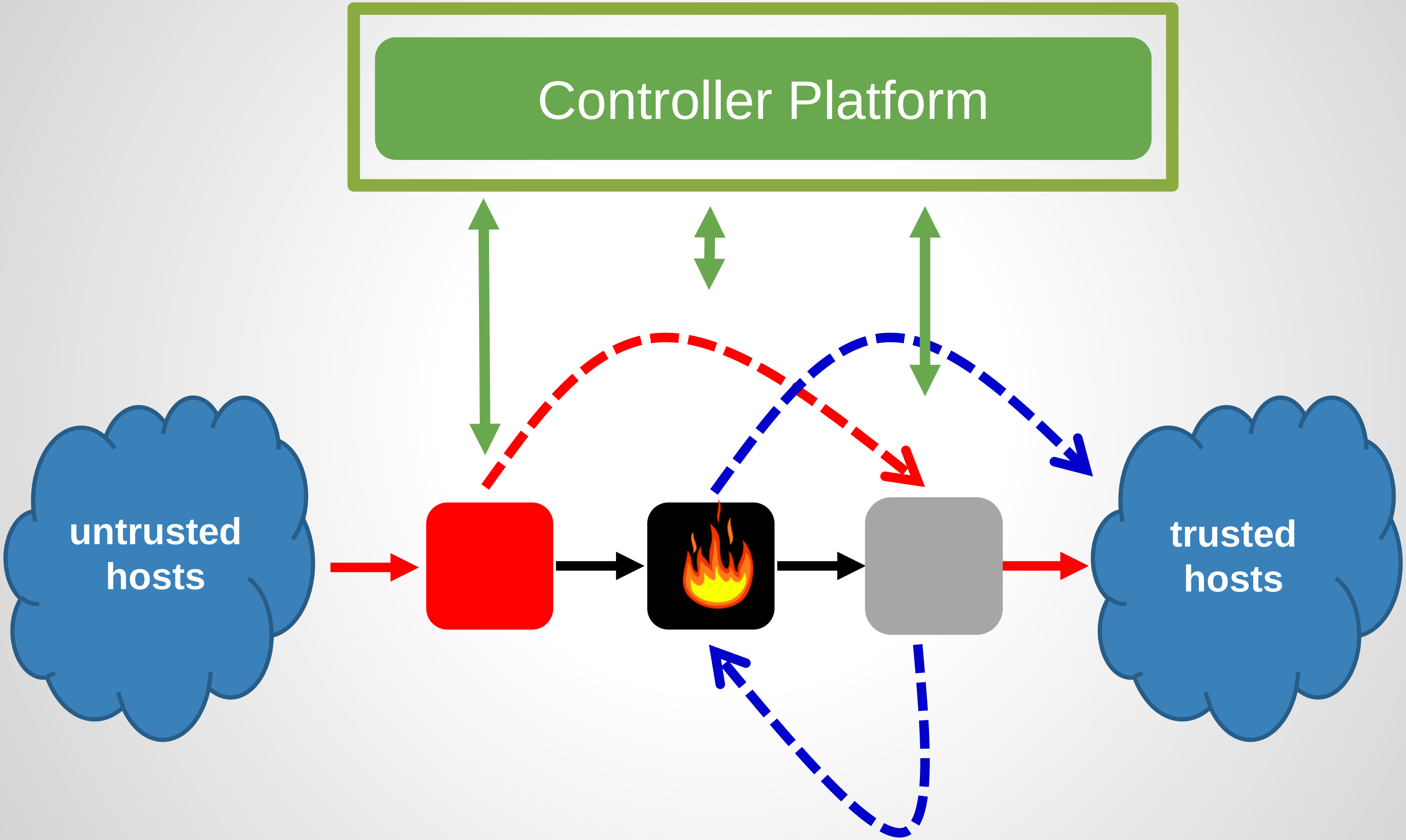
Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

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



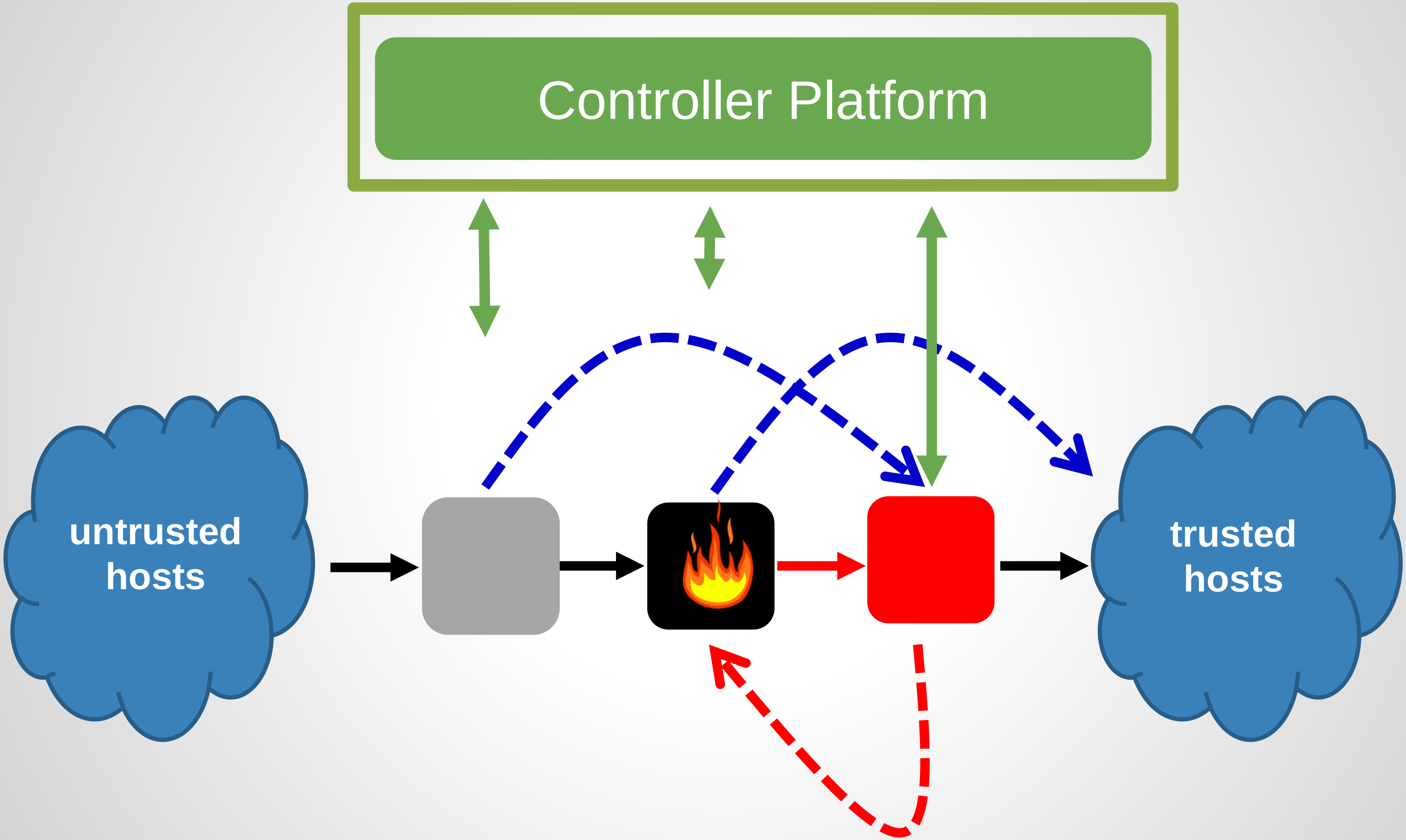
Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

Problem 1: Bypassed Waypoint



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

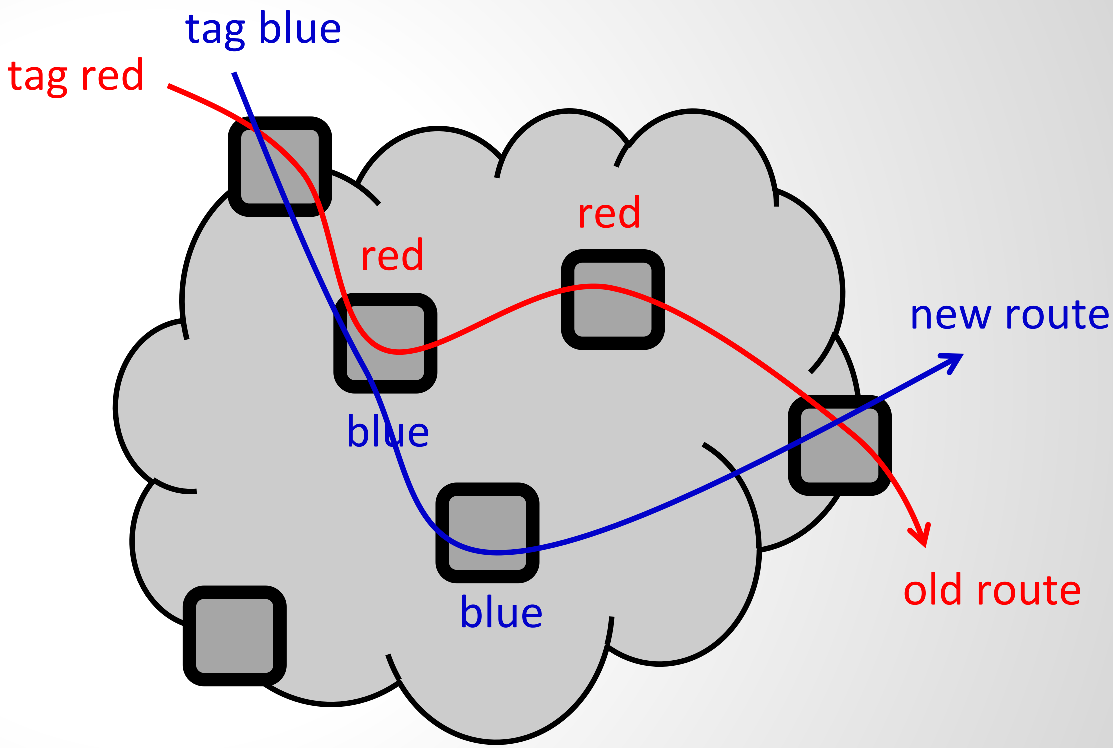
Problem 2: *Transient* Loop



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

Tagging: A Universal Solution?

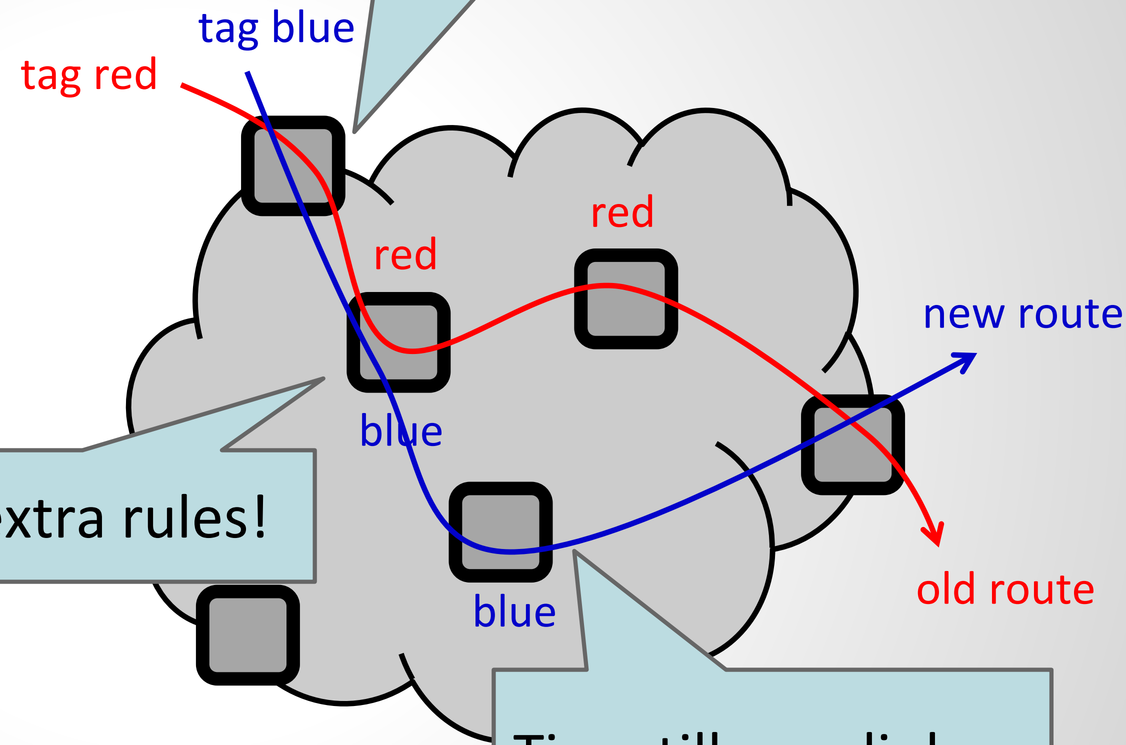
- Old route: red
- New route: blue
- 2-Phase Update:
  - Install blue flow rules internally
  - Flip tag at ingress ports



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.

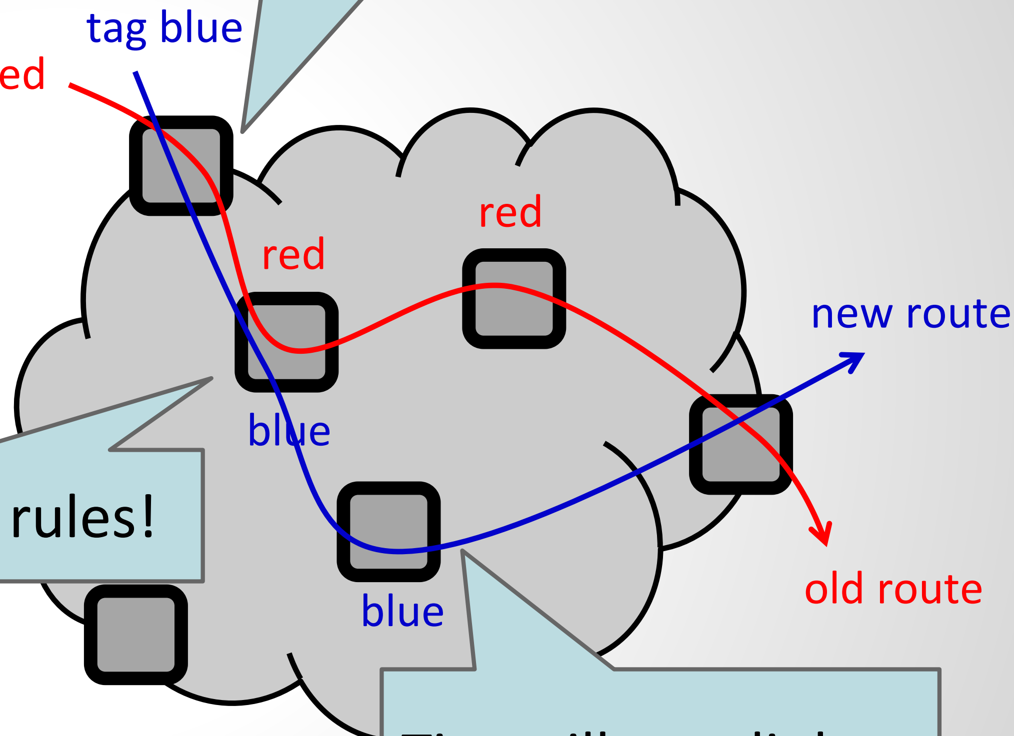
Tagging: A Univers

- 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!



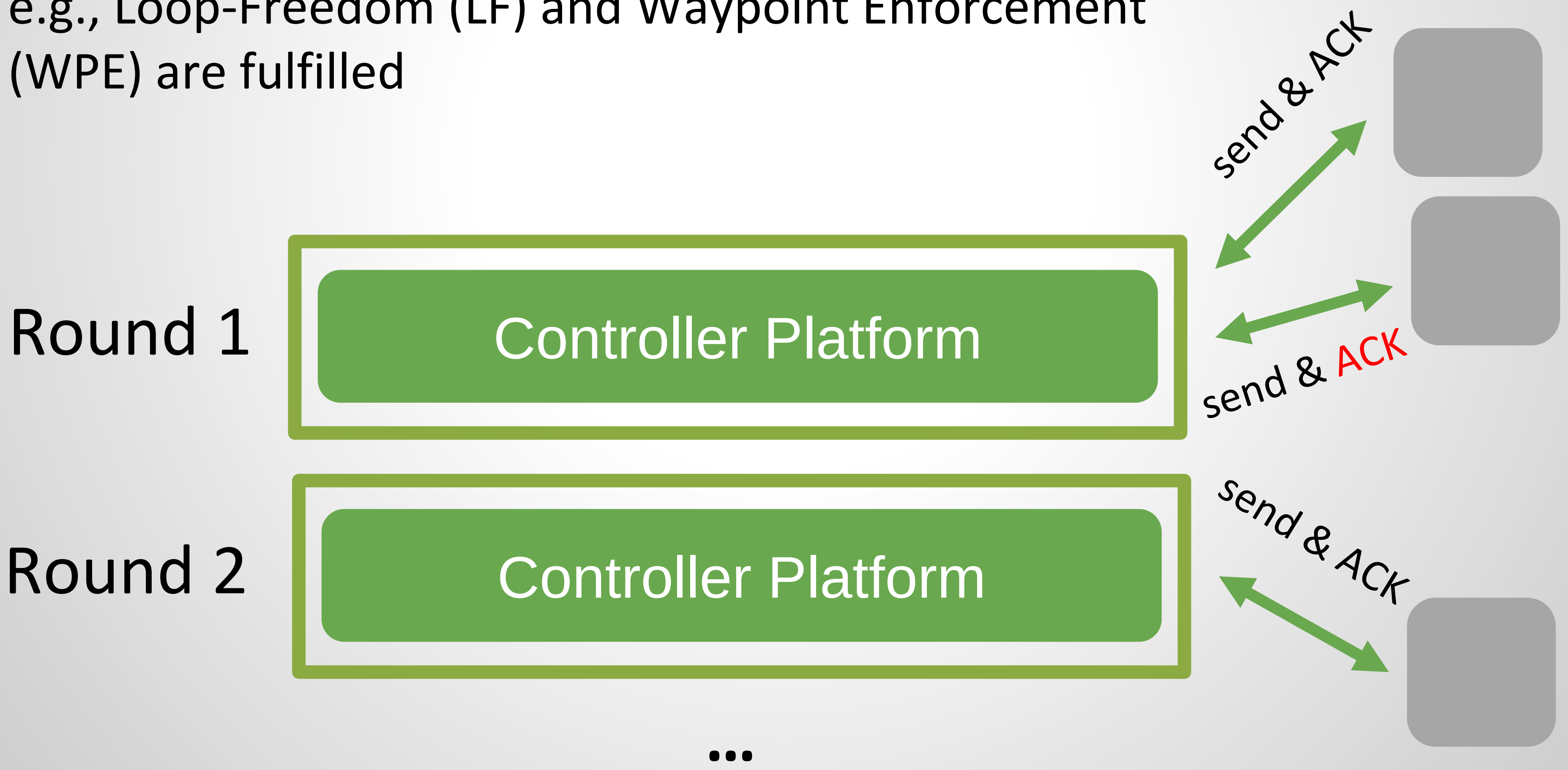
Possible solution without tagging, and at least preserve weaker consistency properties?

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

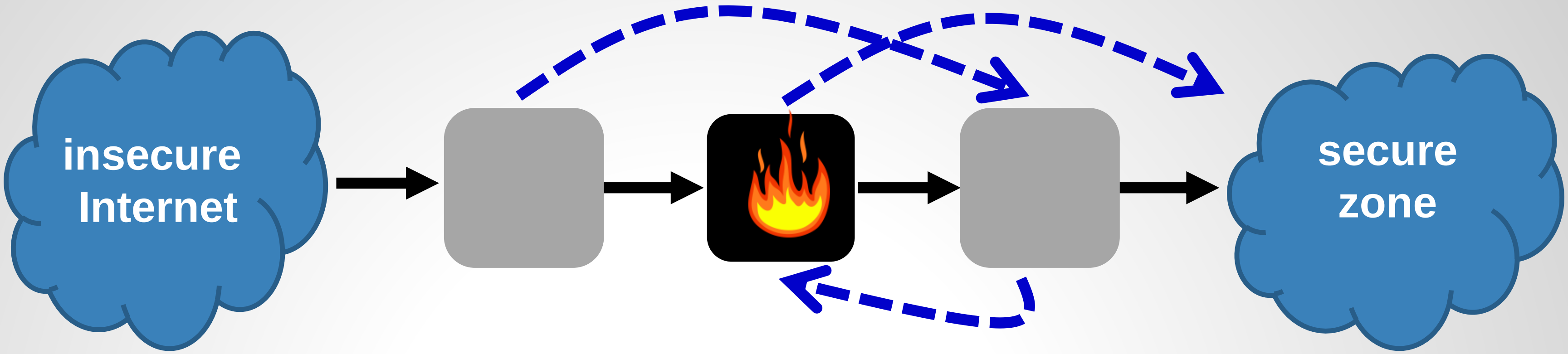
Idea: Schedule "Safe" Subsets of Nodes Only, Then Wait for ACK!

Idea: Schedule safe update subsets in multiple rounds!

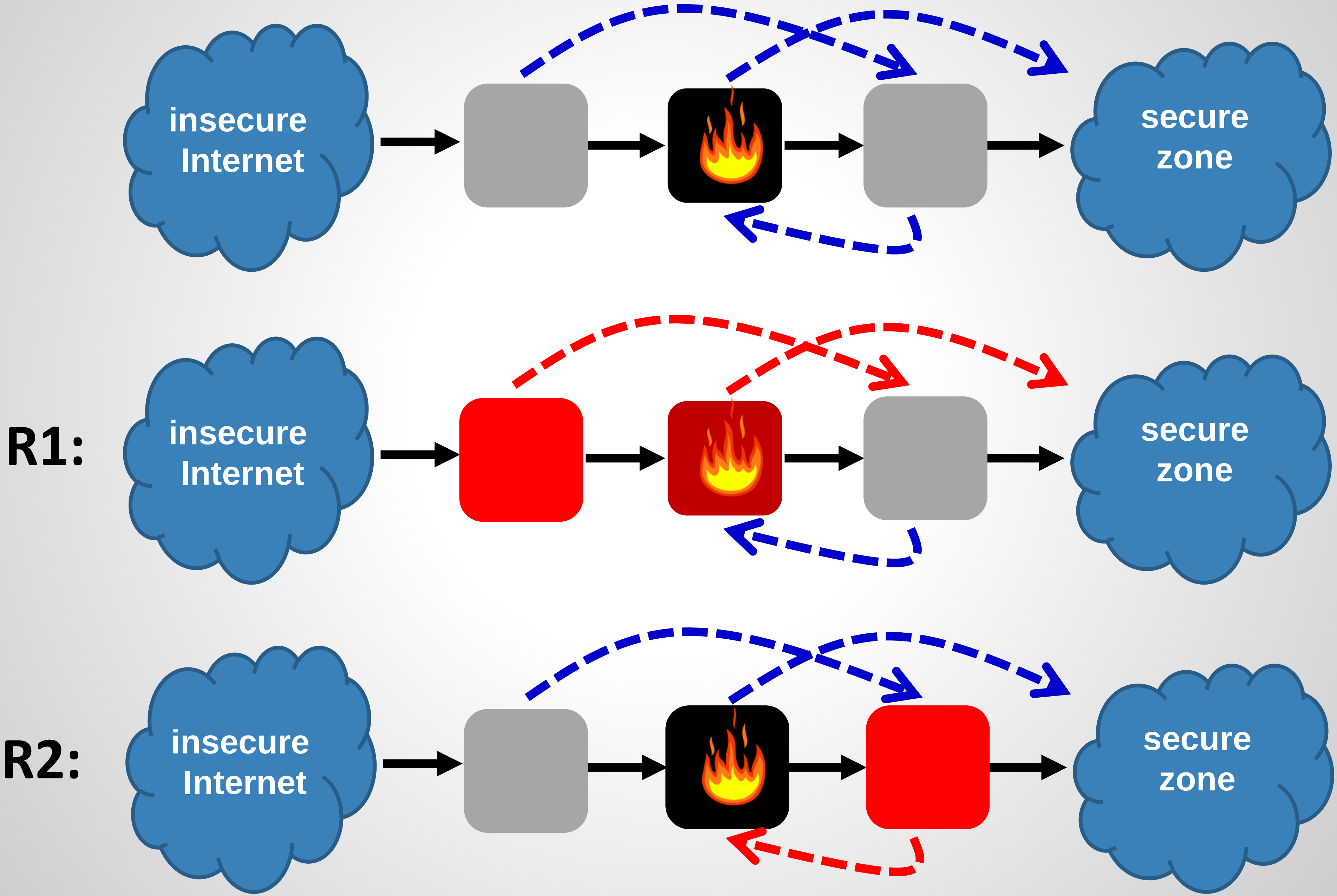
Packet may take a mix of old and new path, as long as, e.g., Loop-Freedom (LF) and Waypoint Enforcement (WPE) are fulfilled



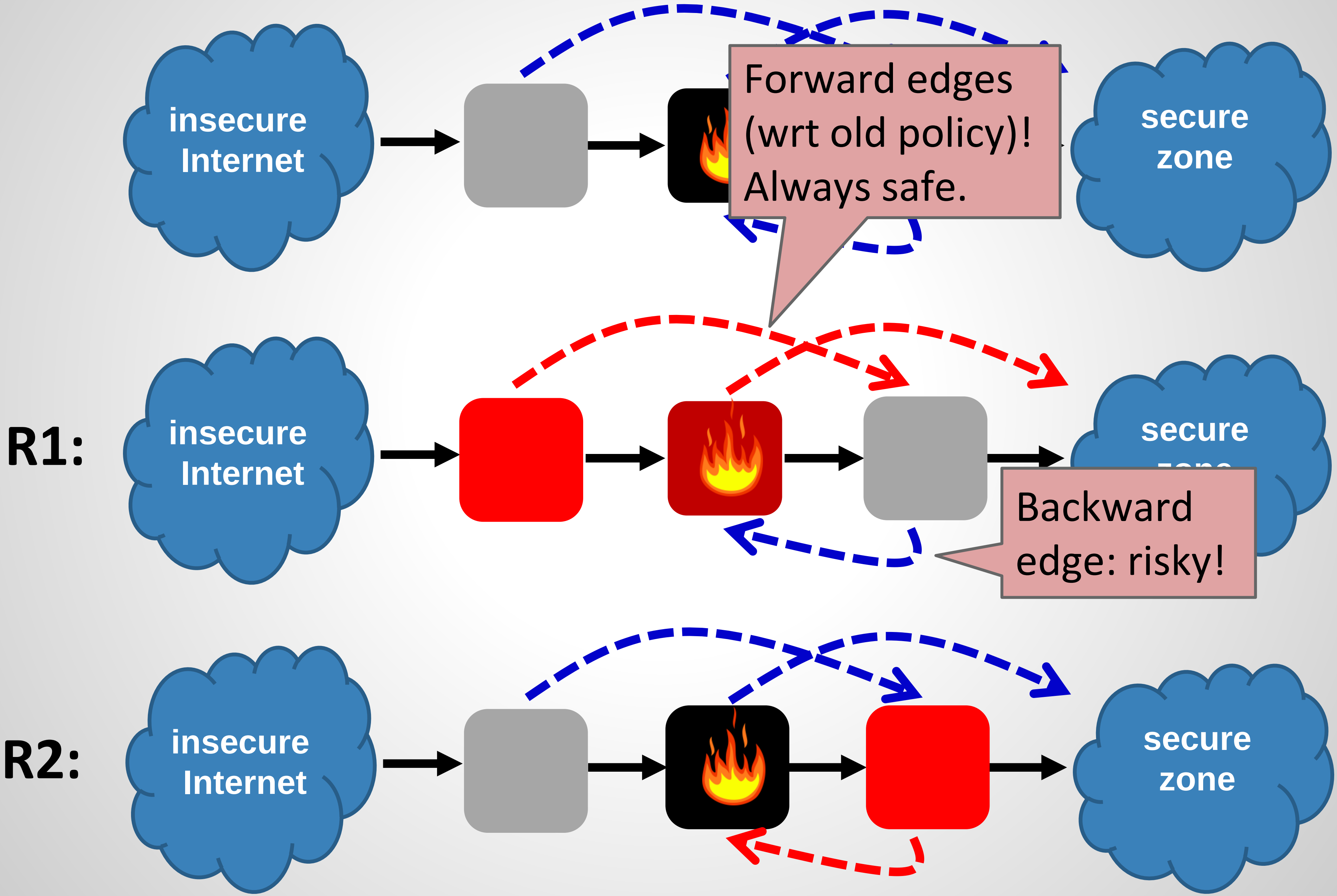
Loop-Free Update Schedule



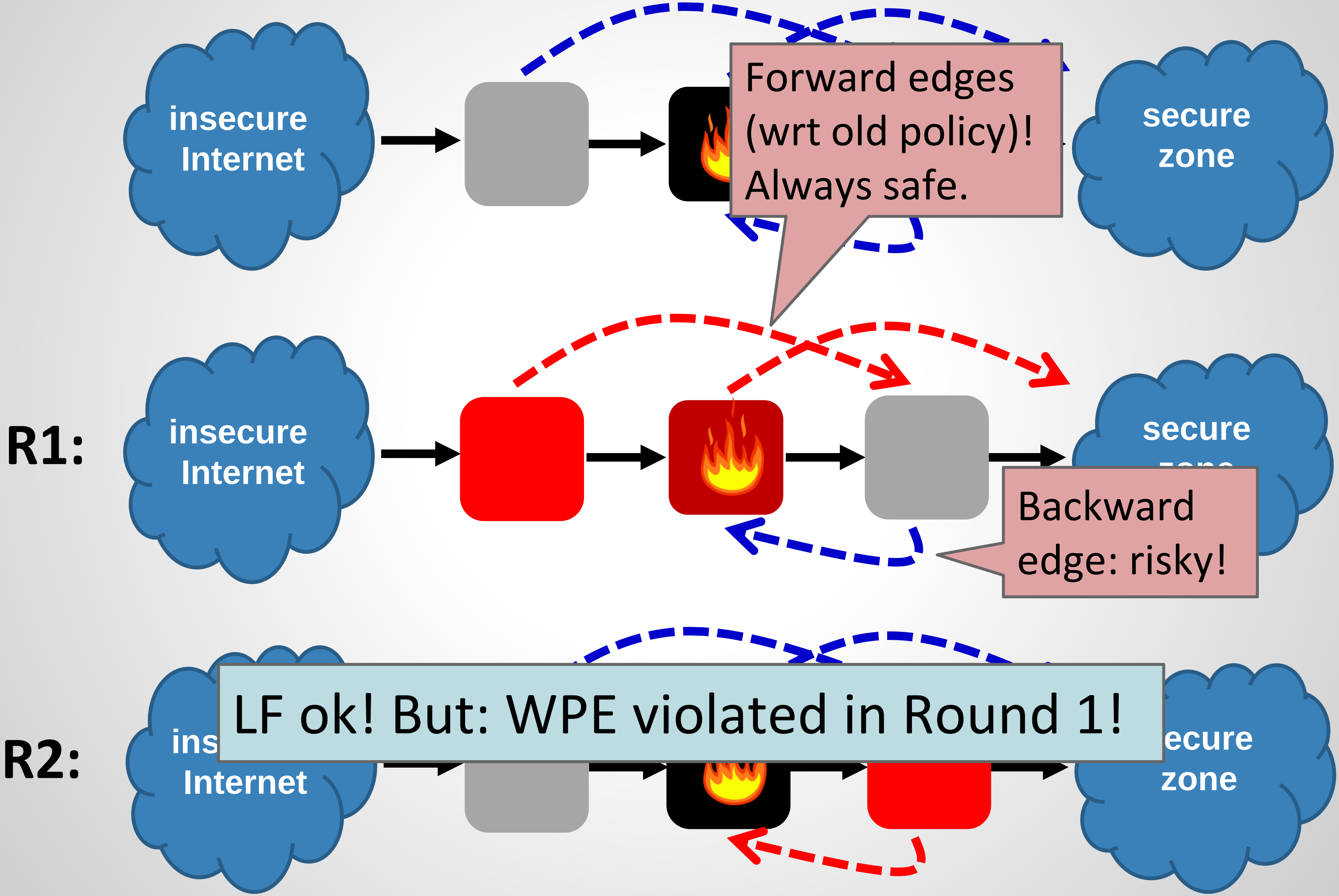
Loop-Free Update Schedule



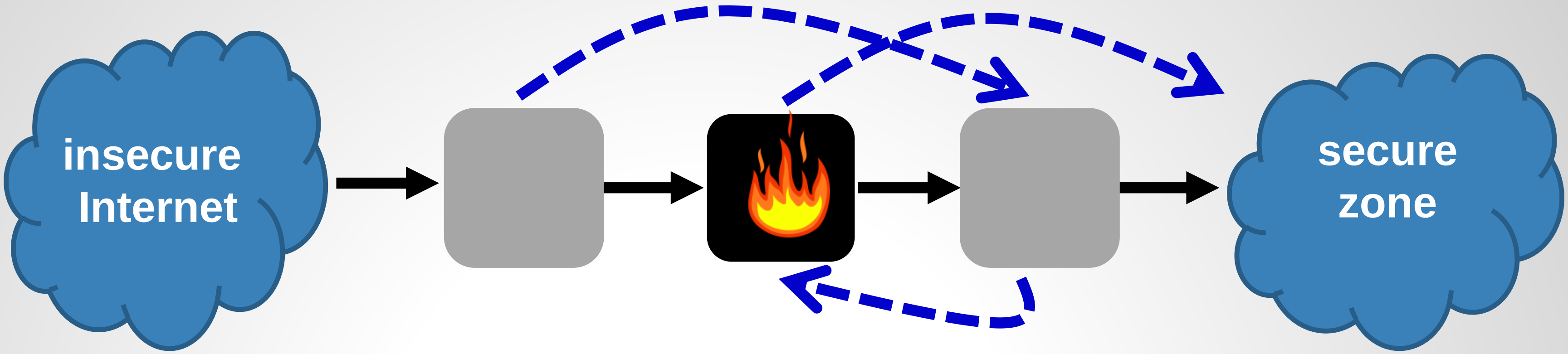
Loop-Free Update Schedule



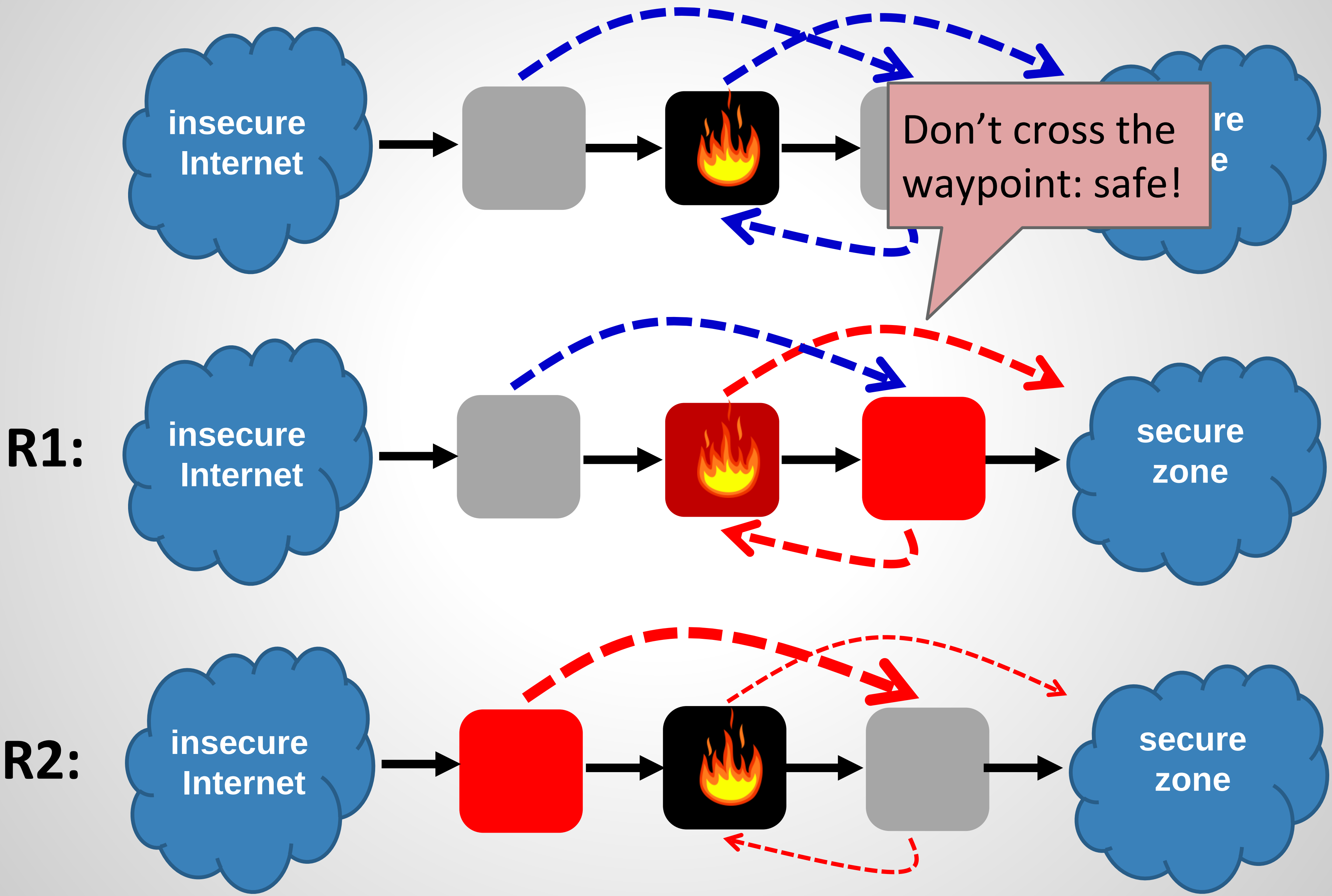
Loop-Free Update Schedule



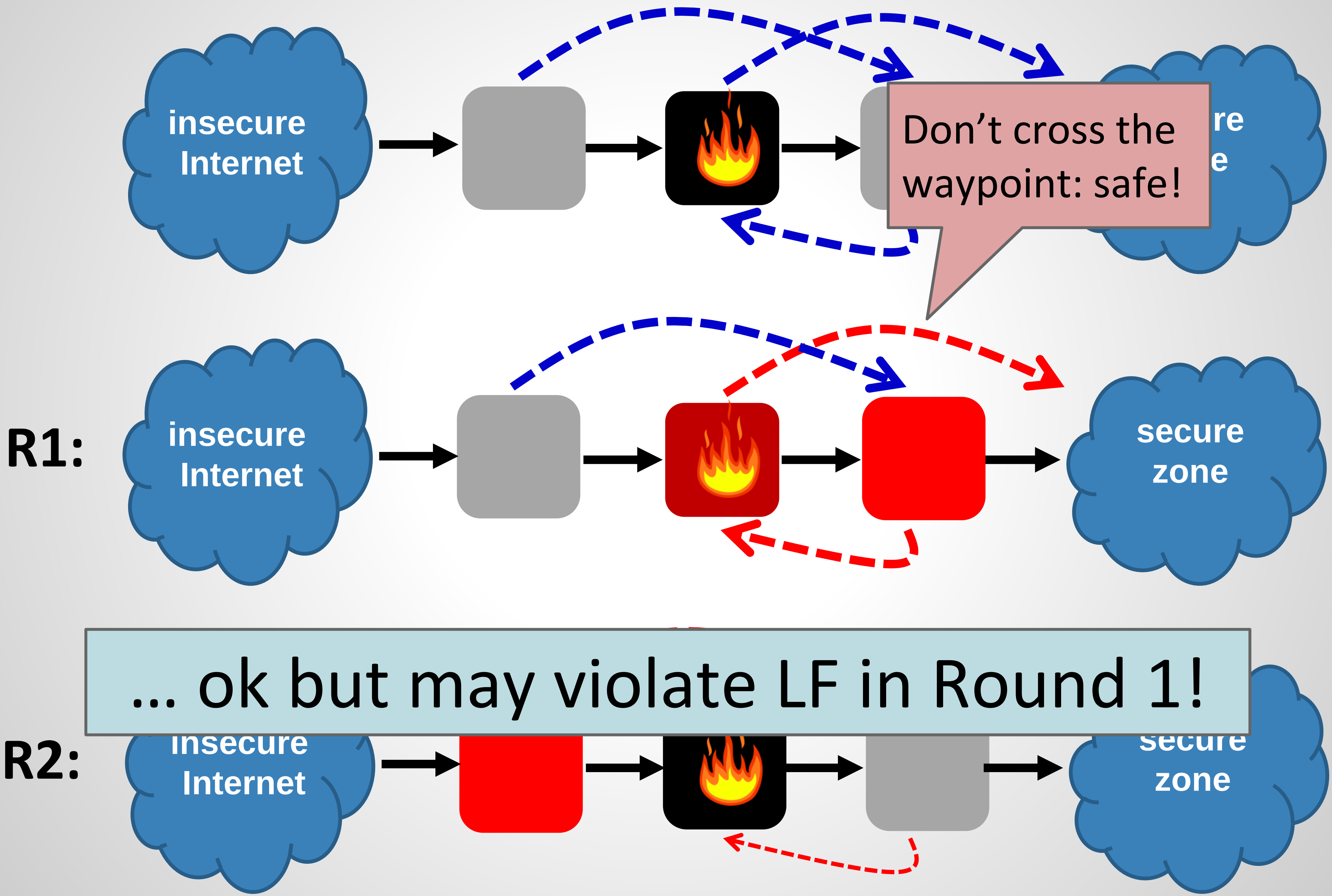
Waypoint Respecting Schedule



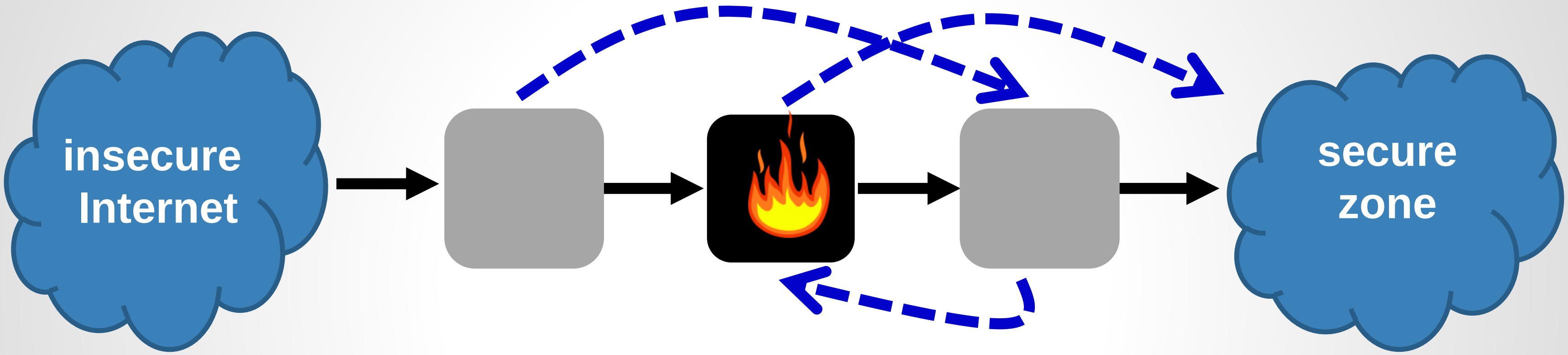
Waypoint Respecting Schedule



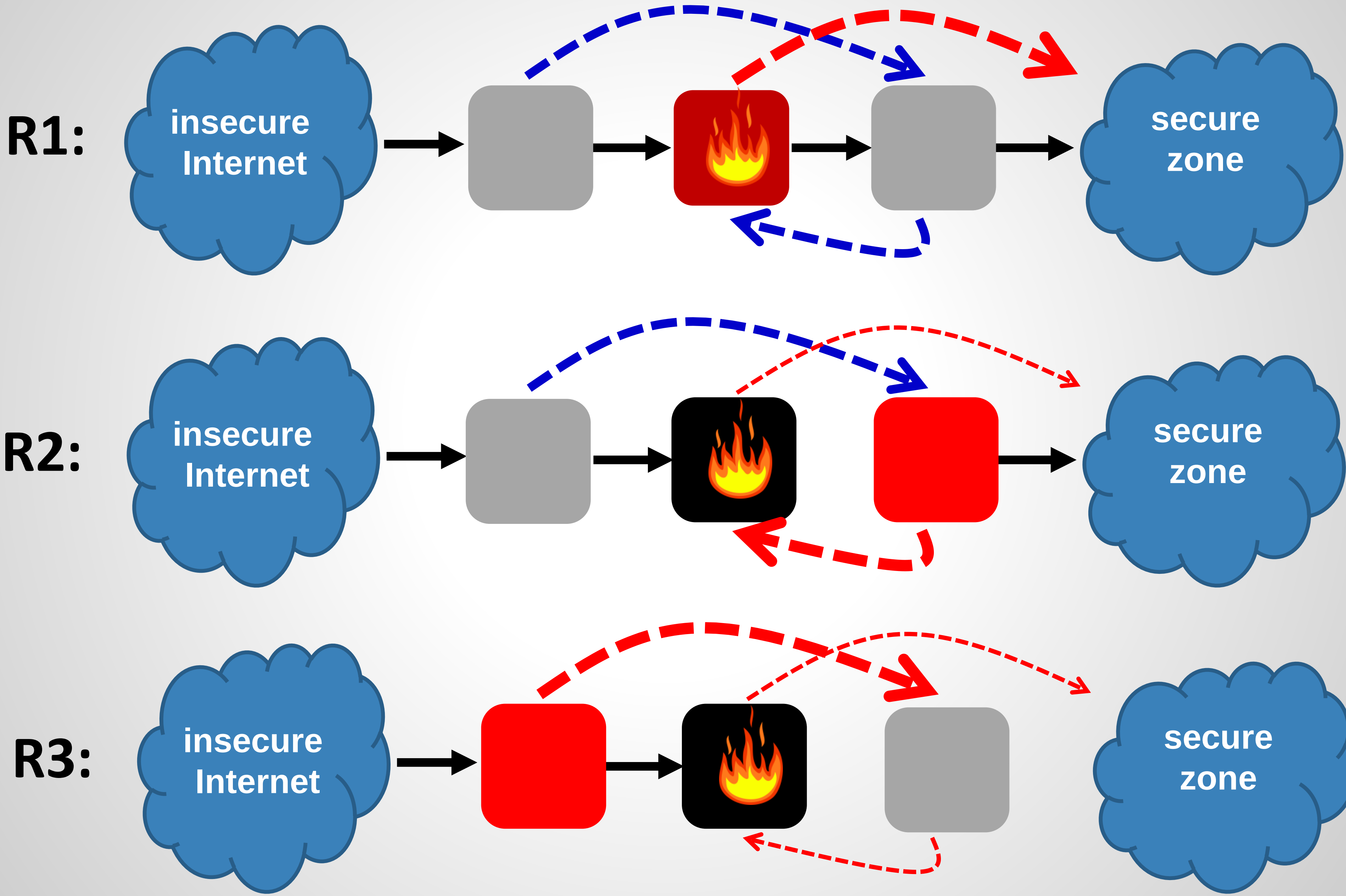
Waypoint Respecting Schedule



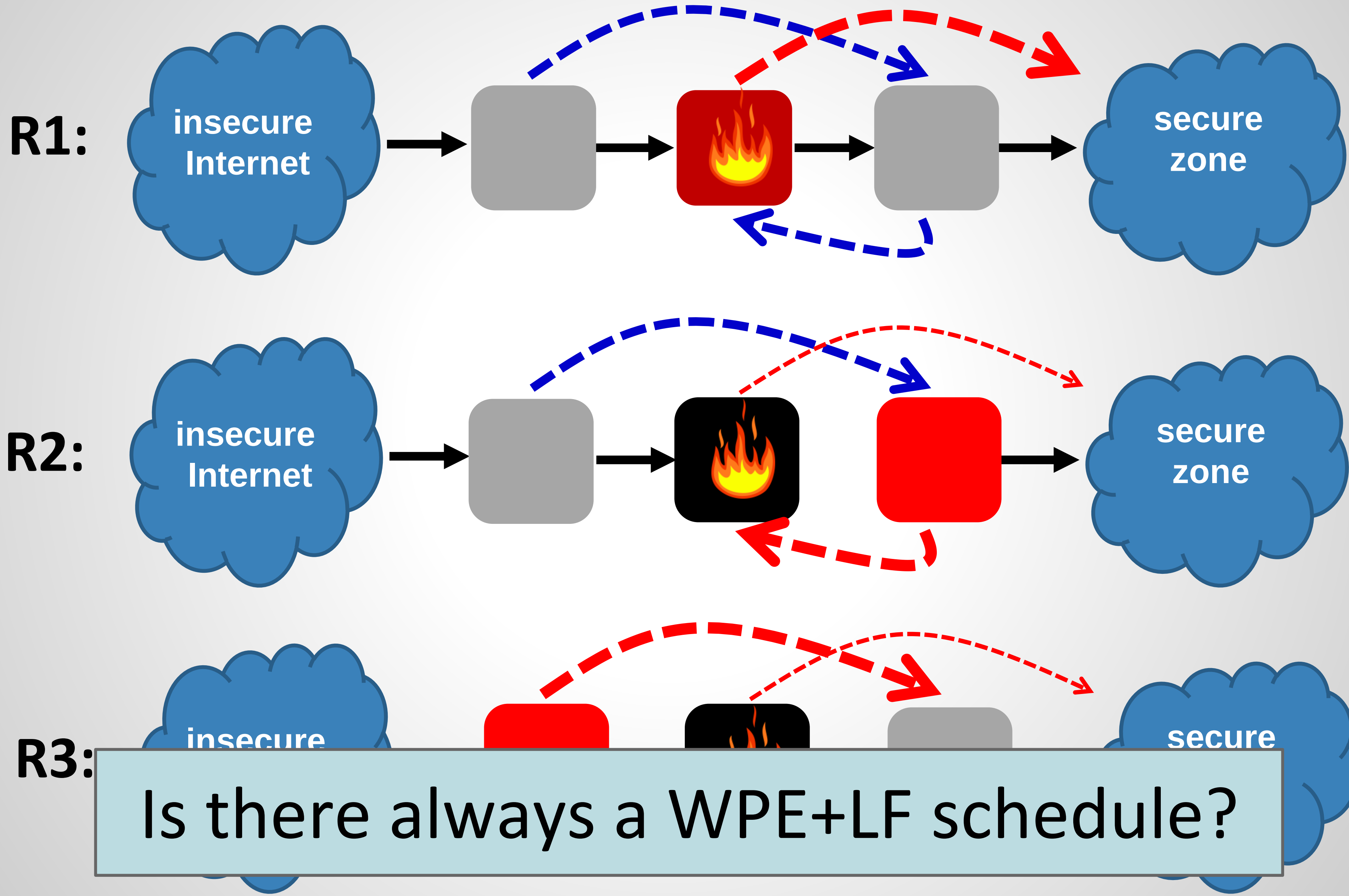
Can we have both LF and WPE?



Yes: but it takes 3 rounds!

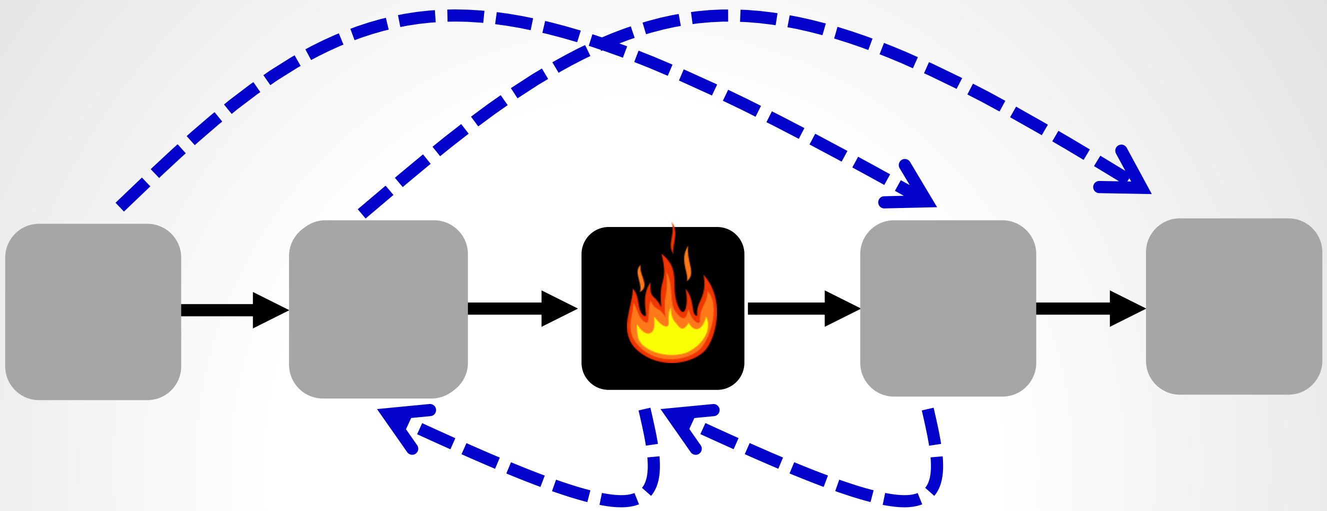


Yes: but it takes 3 rounds!

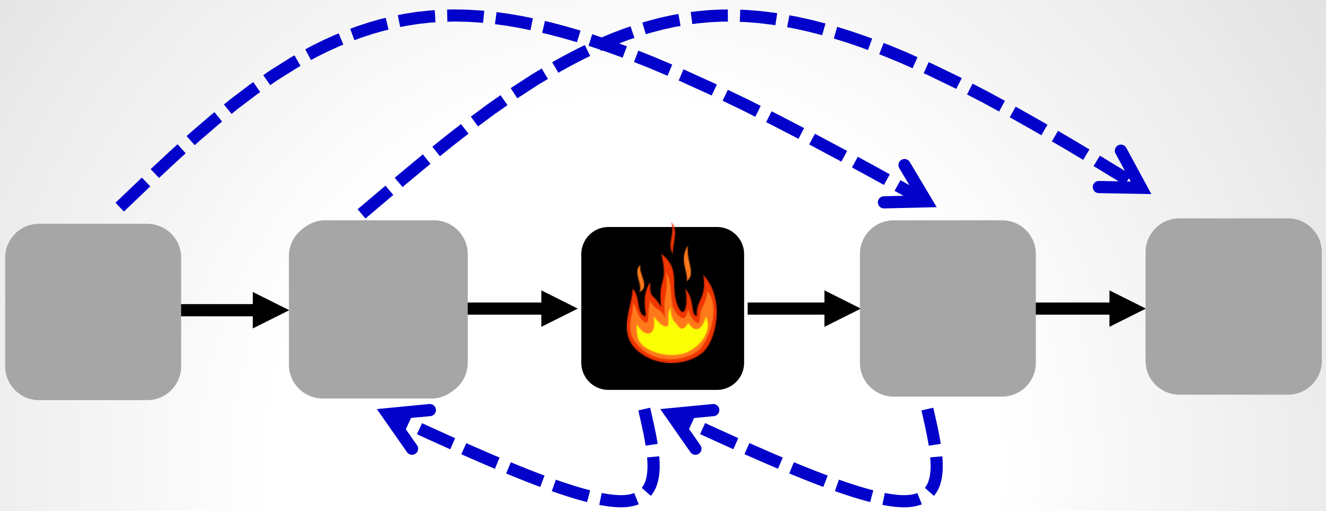


Is there always a WPE+LF schedule?

What about this one?



LF and WPE may conflict!

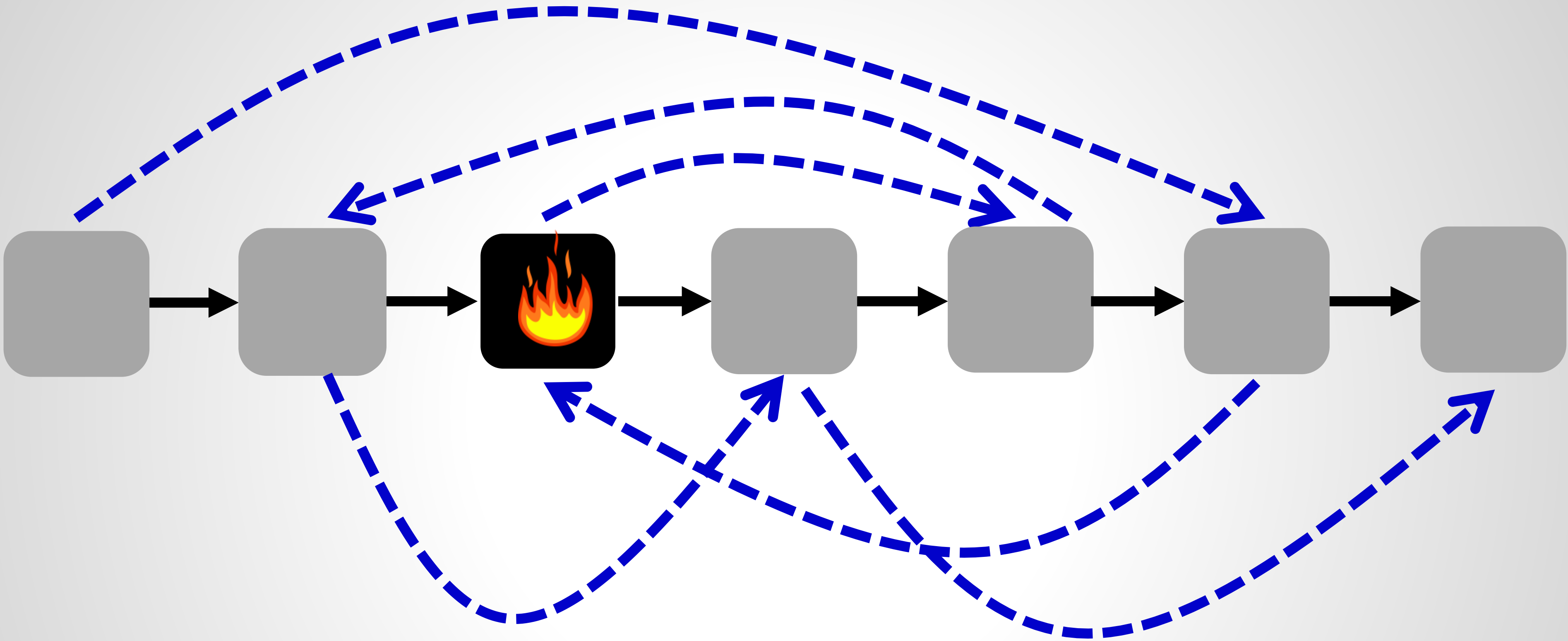


- ❑ Cannot update any **forward edge** in R1: WP
- ❑ Cannot update any **backward edge** in R1: LF

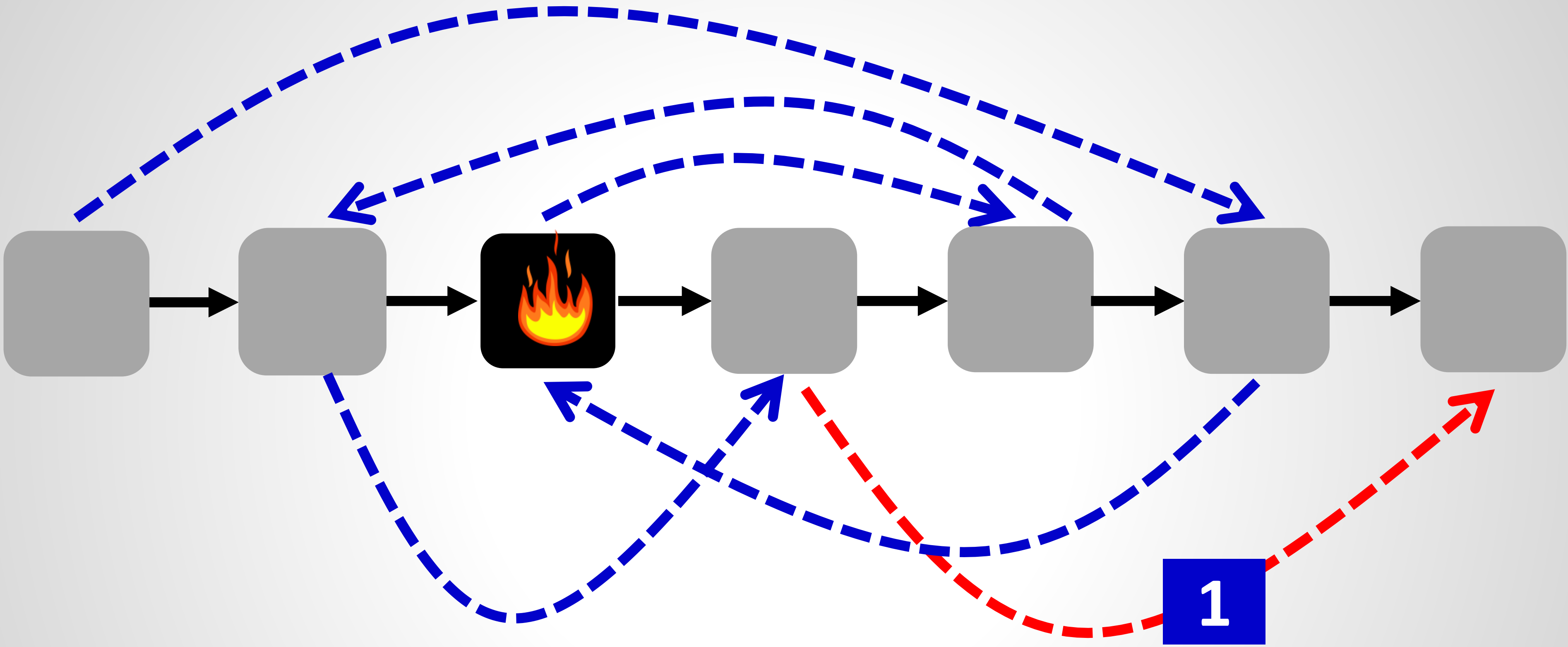
No schedule exists!

Resort to tagging…

What about this one?

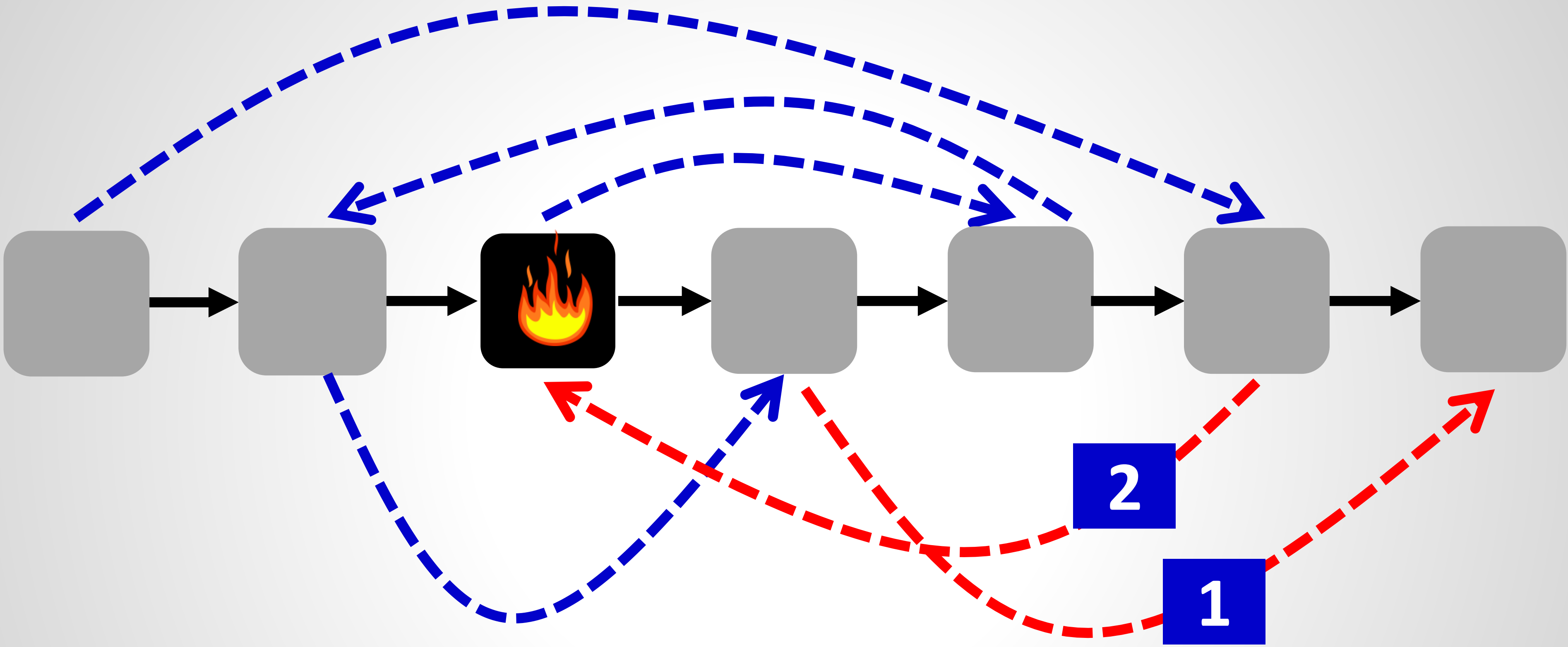


What about this one?



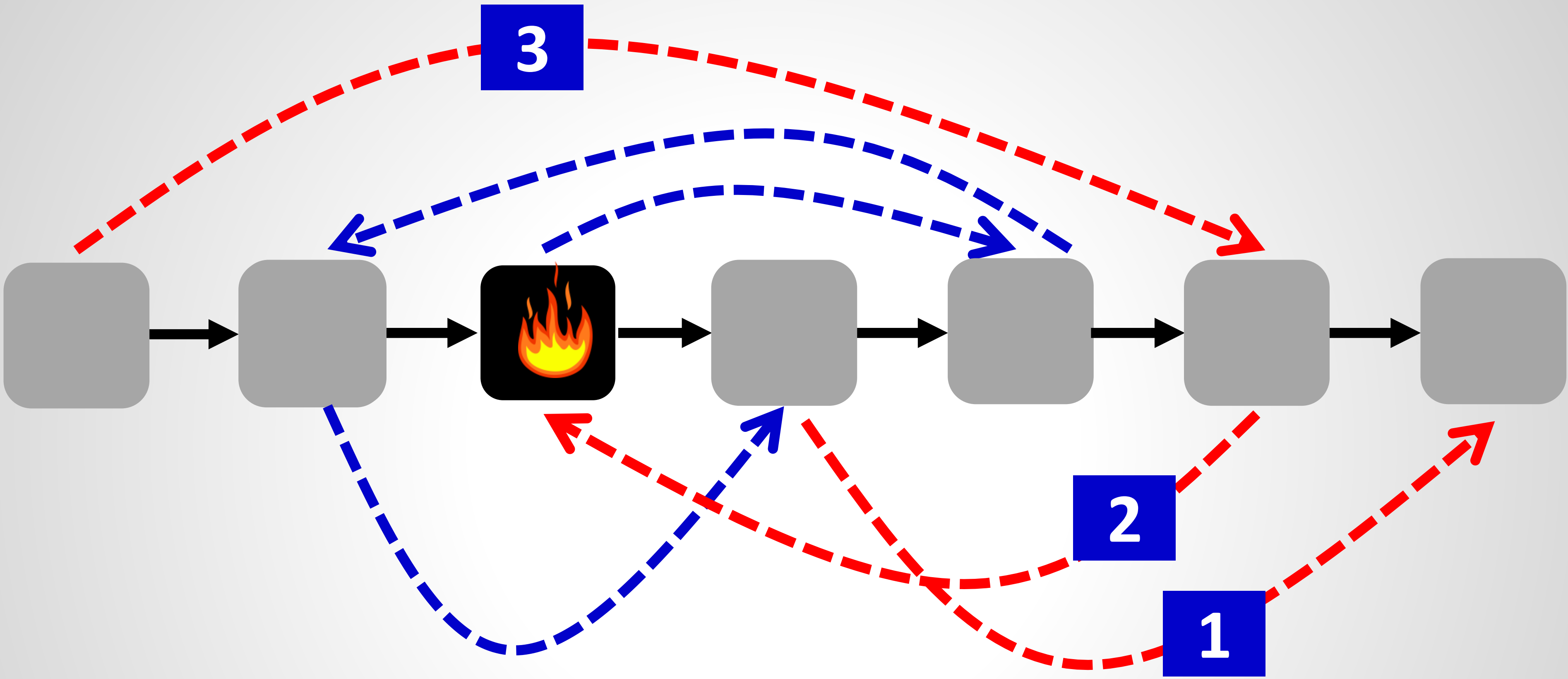
- Forward edge after the waypoint: safe!
  - No loop, no WPE violation

What about this one?



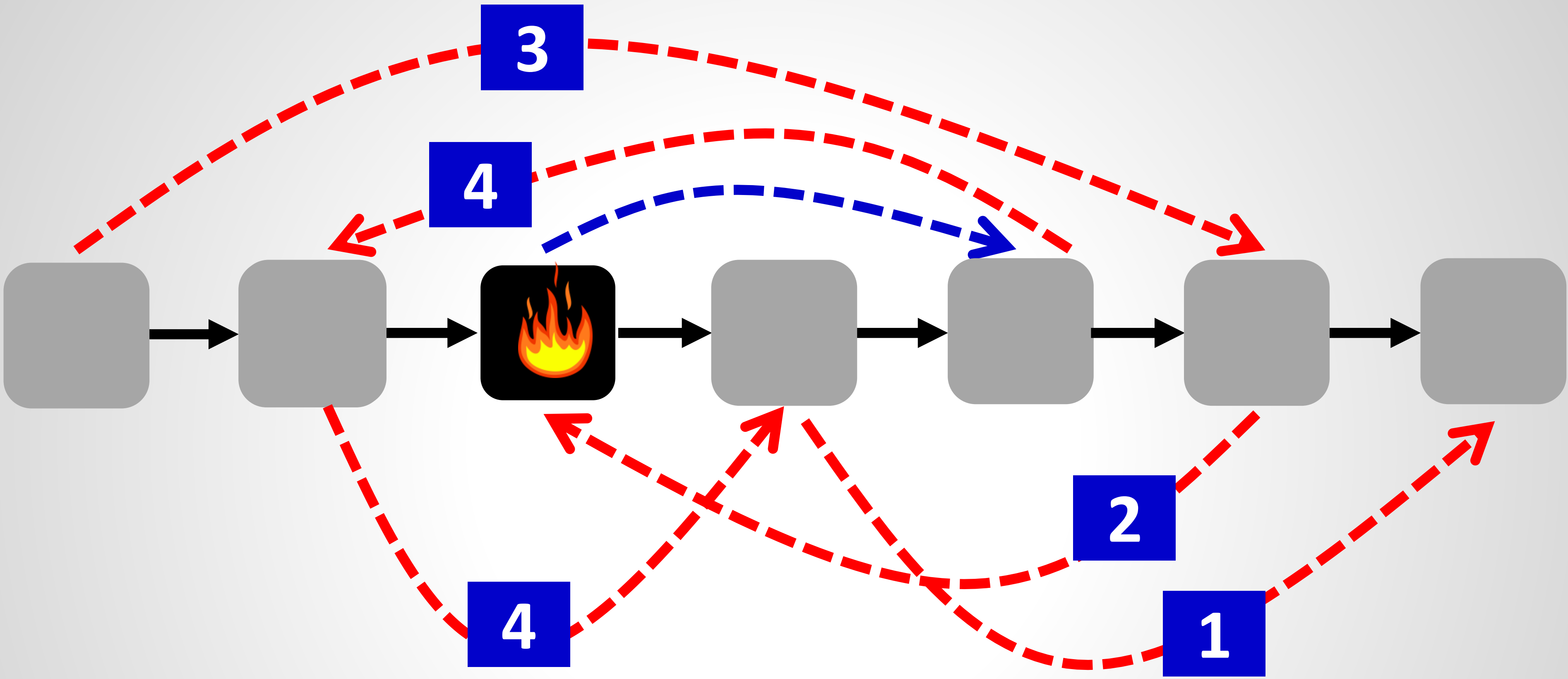
- Now this backward is safe too!
  - No loop because exit through 1

What about this one?



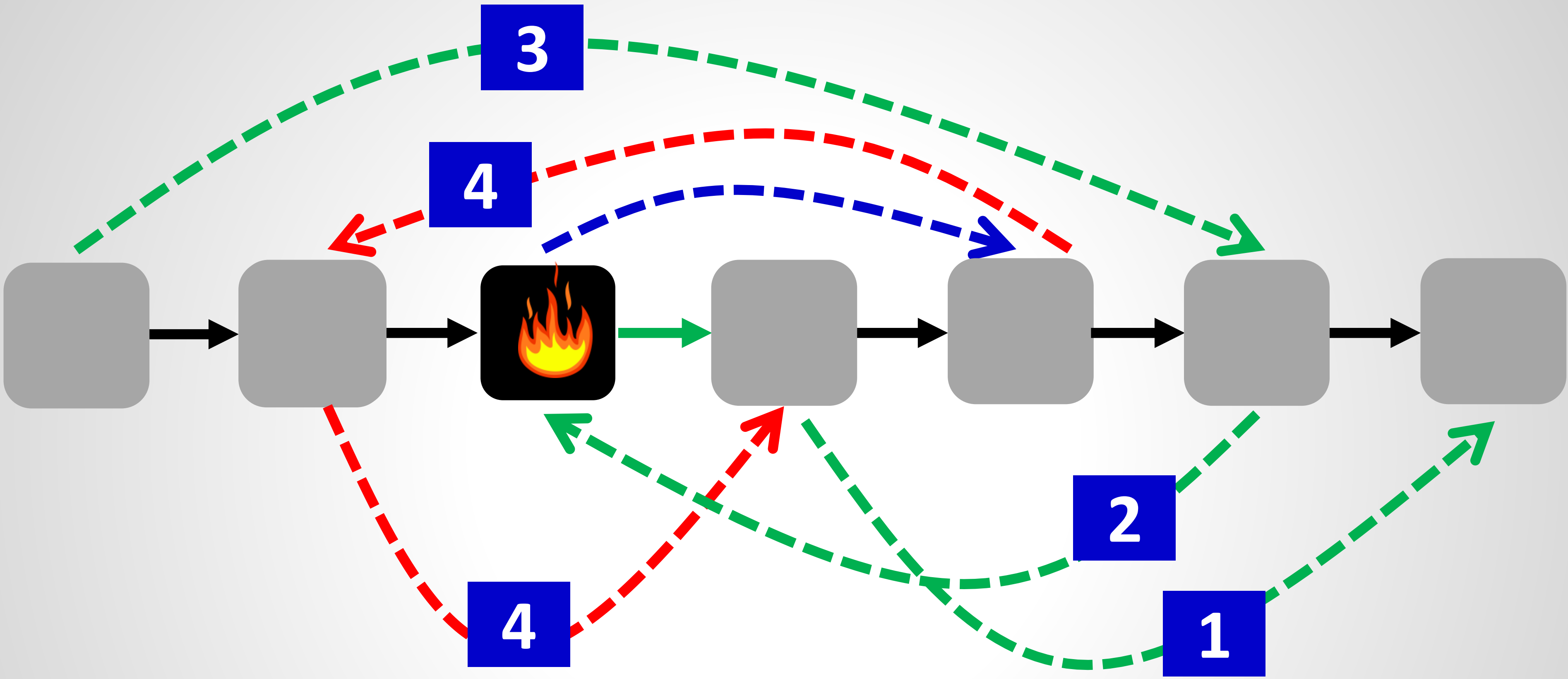
- Now this is safe: **2** ready back to WP!
  - No waypoint violation

What about this one?



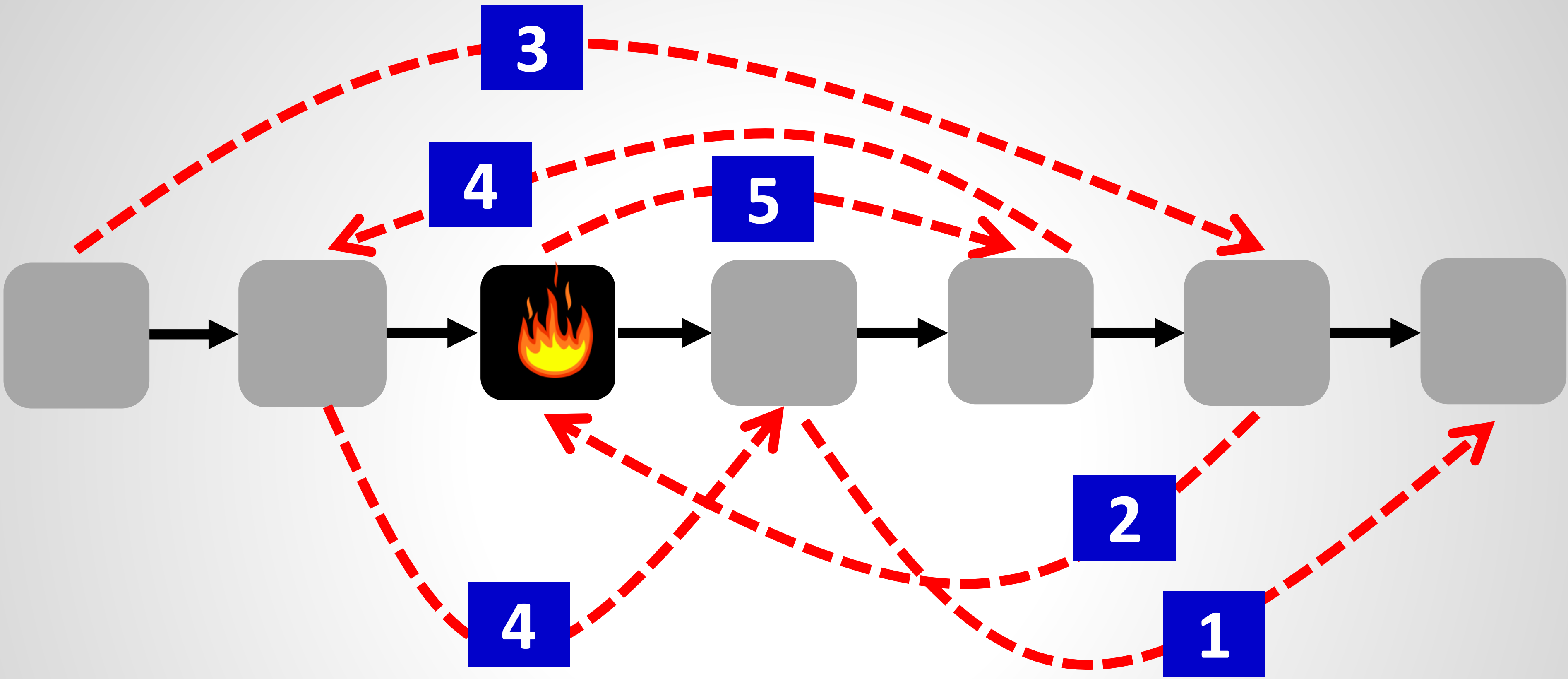
- Ok: loop-free and also not on the path (exit via 1)

What about this one?

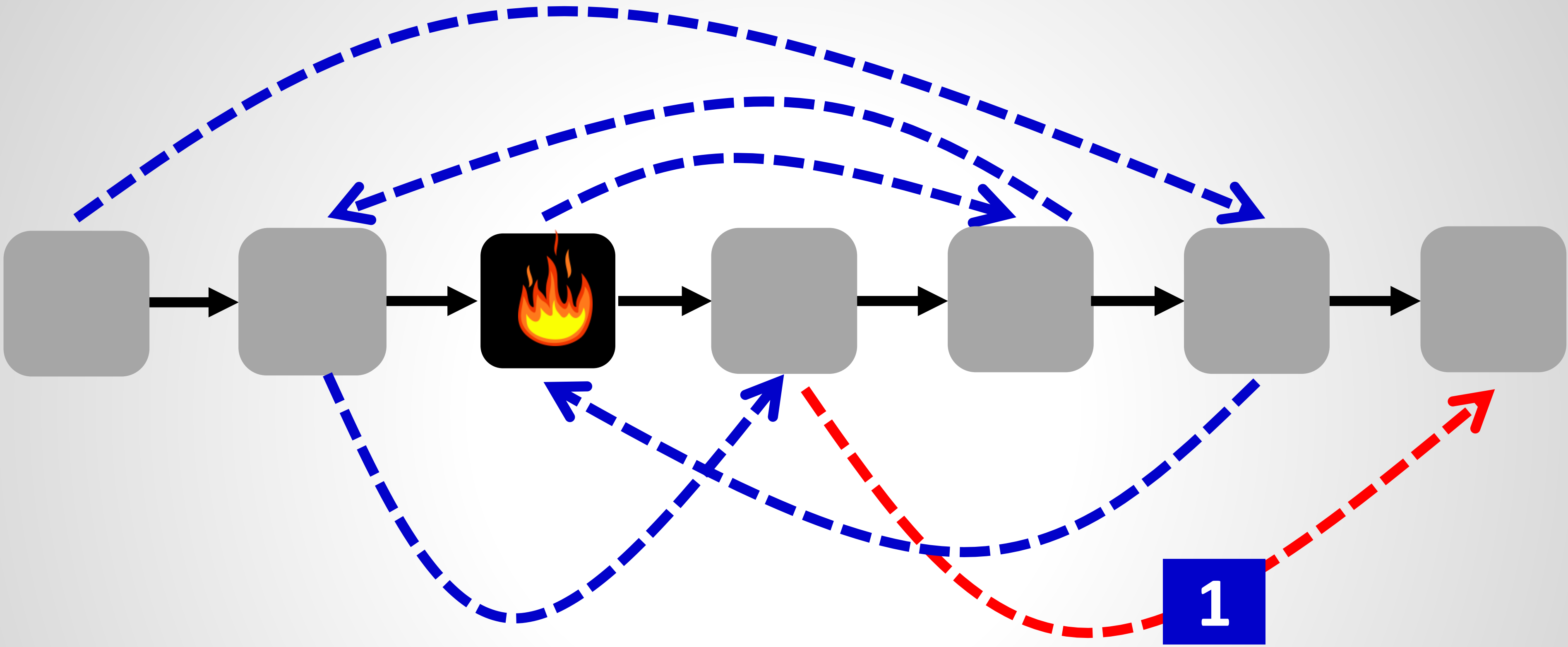


- Ok: loop-free and also not on the path (exit via 1)

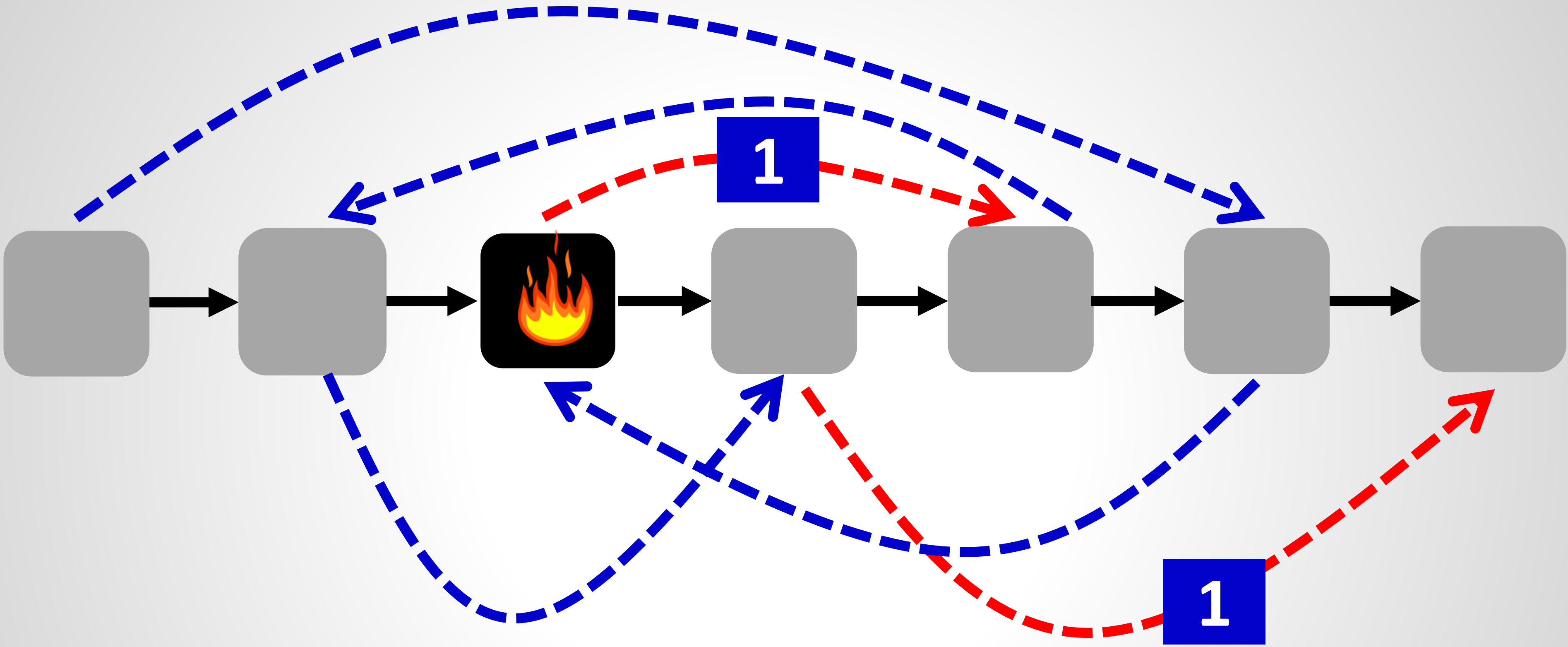
What about this one?



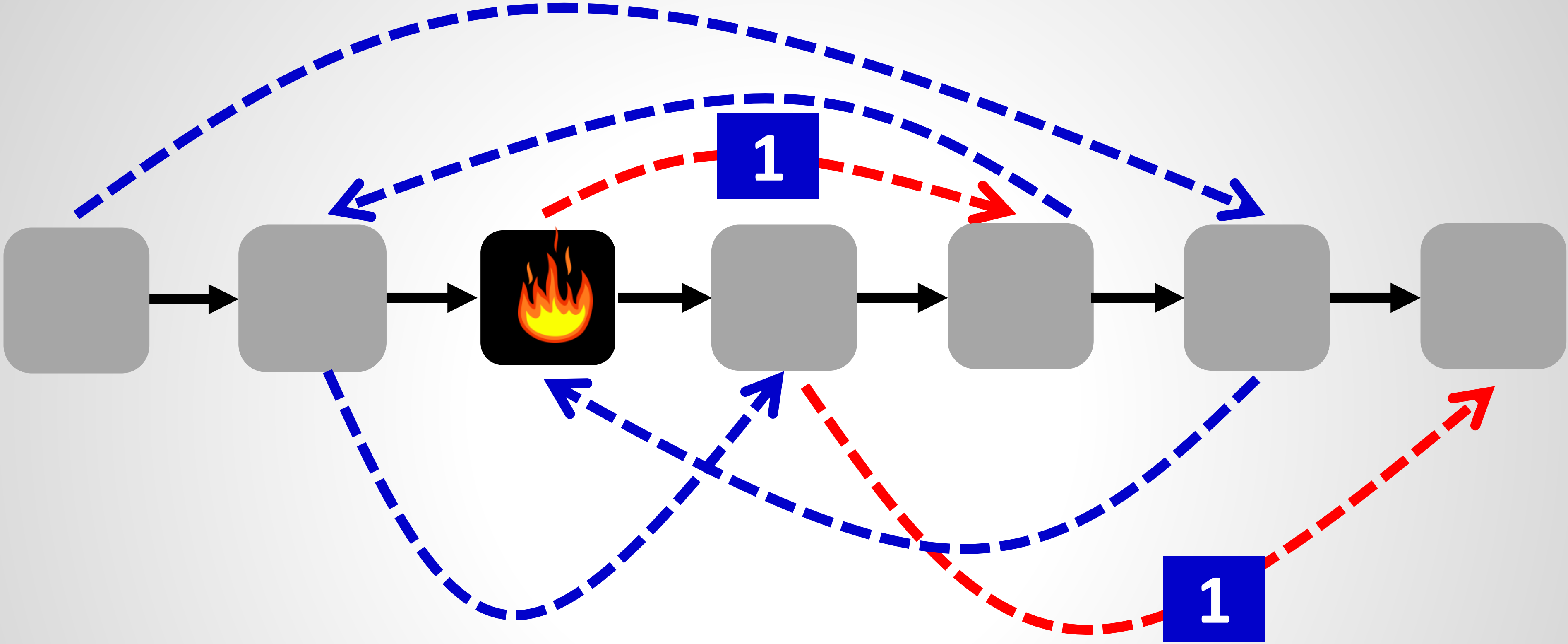
Back to the start: What if....



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

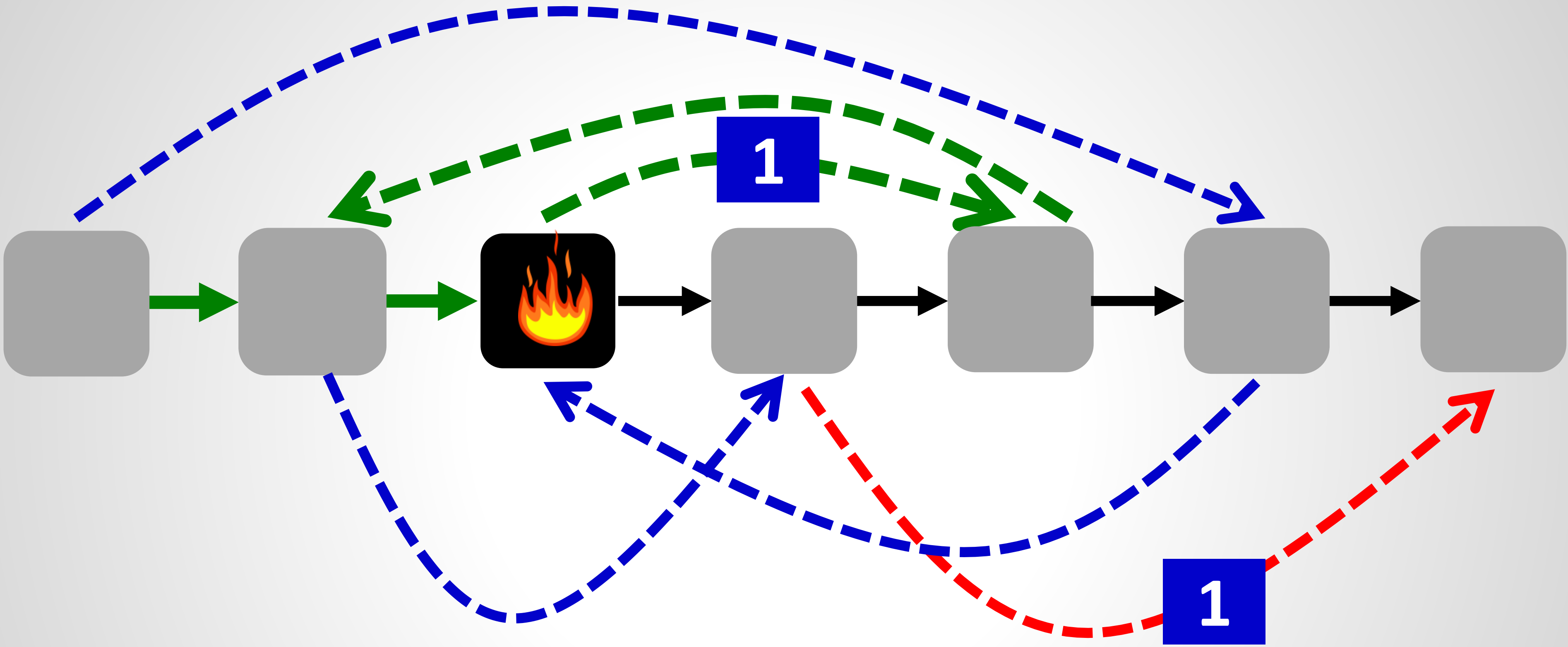


Back to the start: What if…. also this one?!



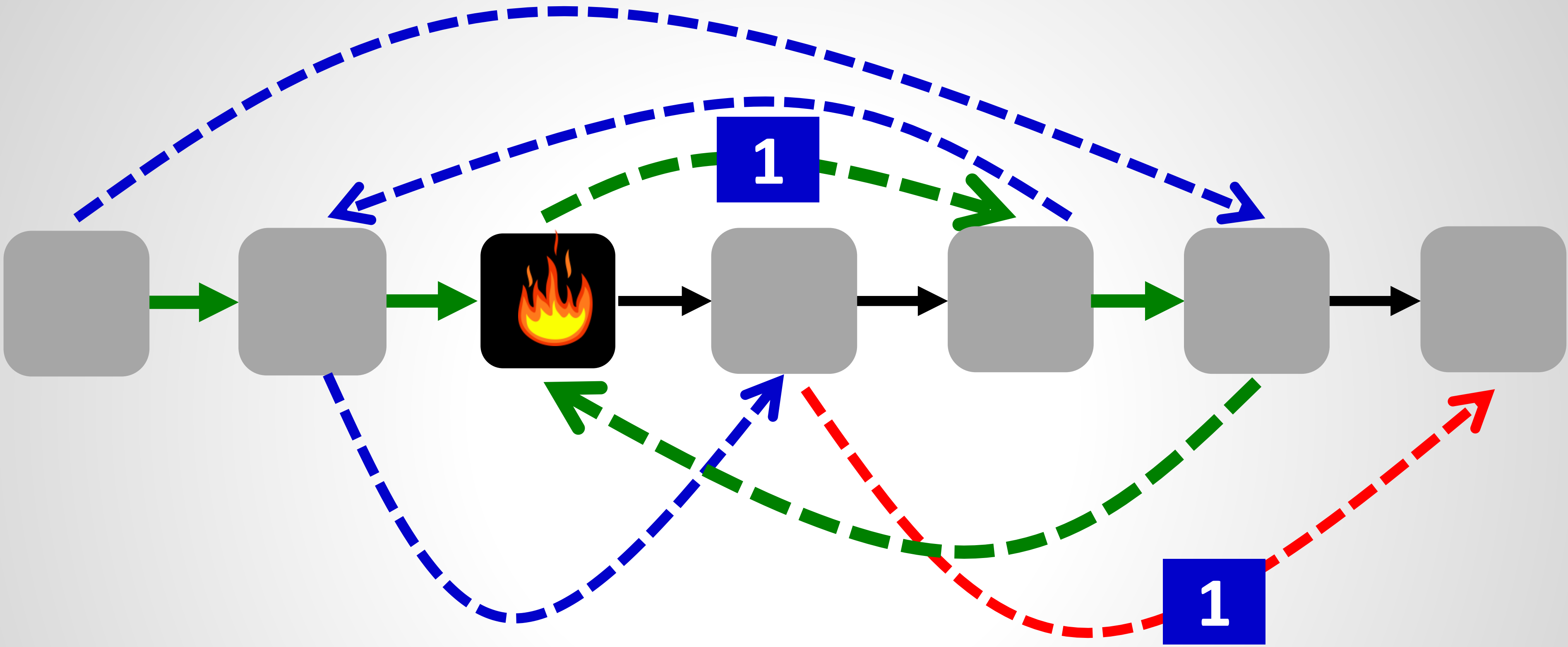
- ❑ Update any of the 2 backward edges? LF ☹

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



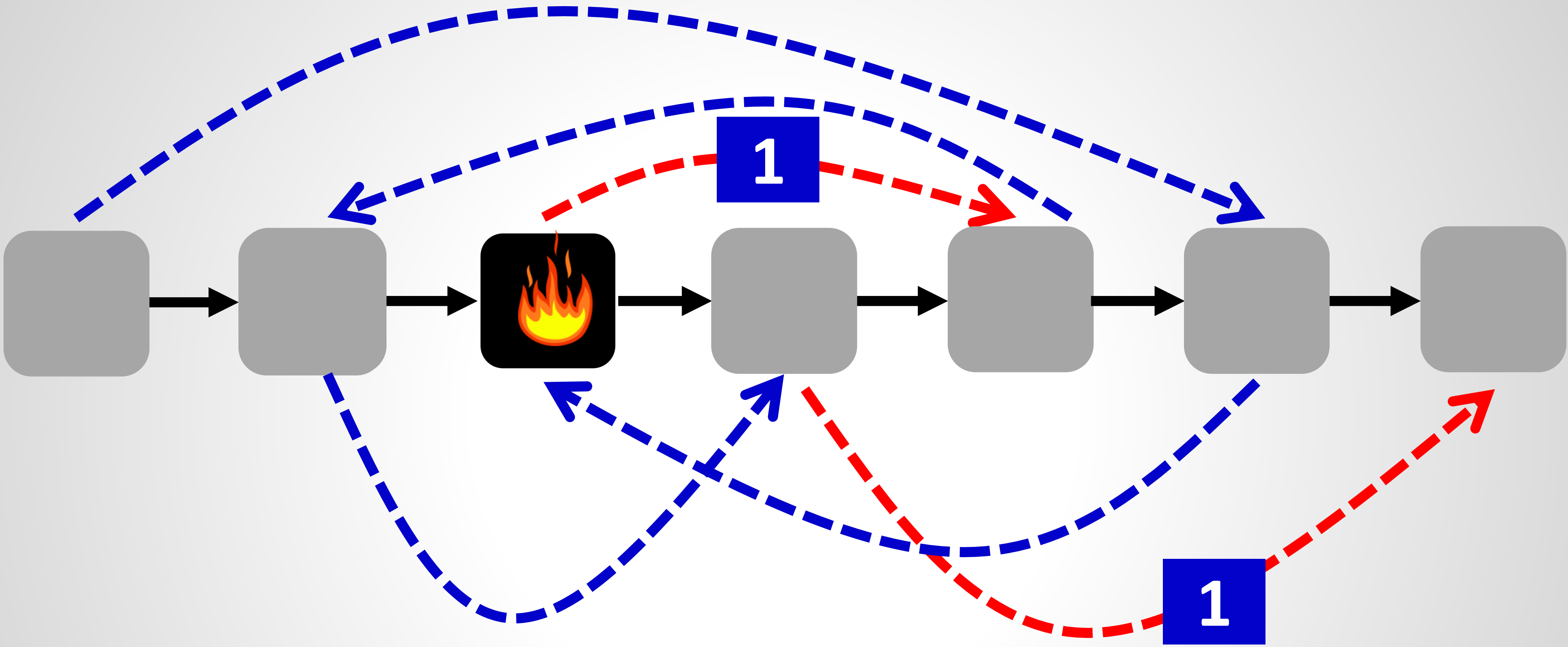
- ❑ Update any of the 2 backward edges? LF ☹

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



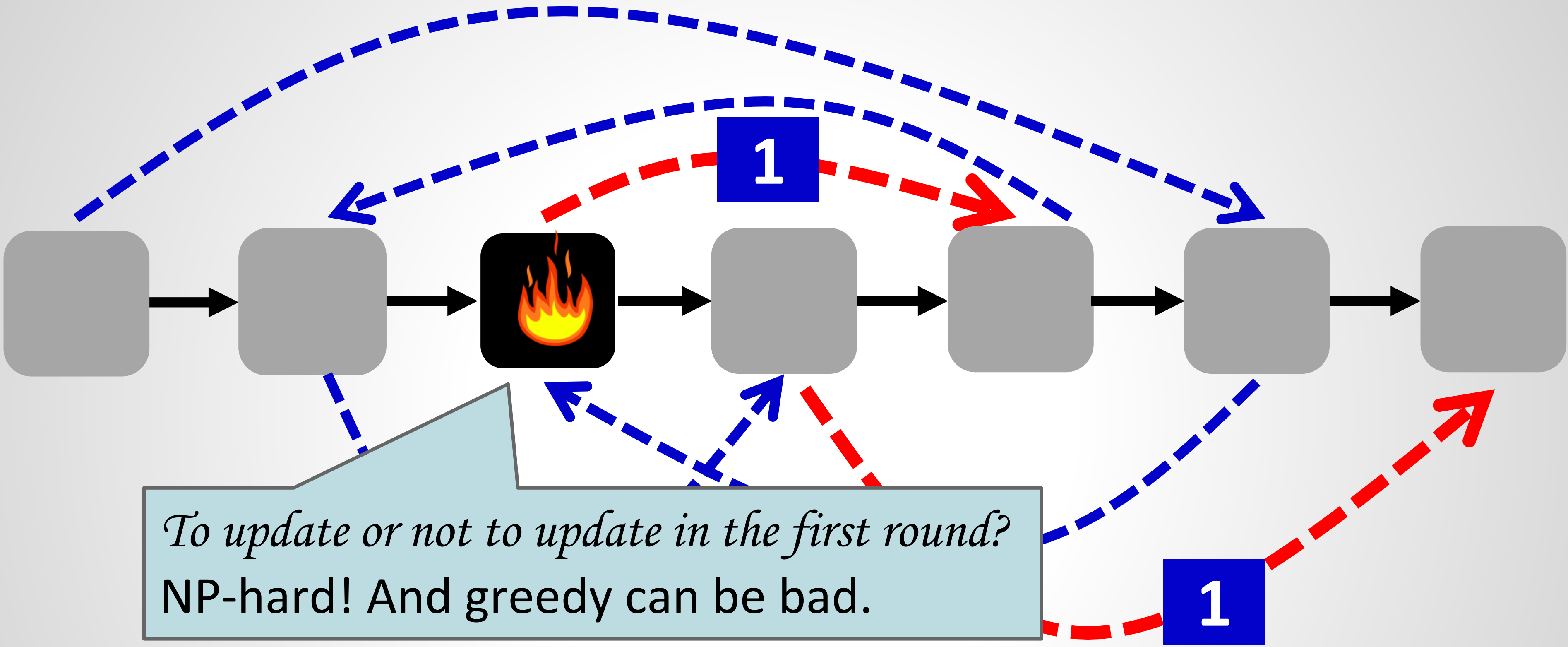
- Update any of the 2 backward edges? LF ☹

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!

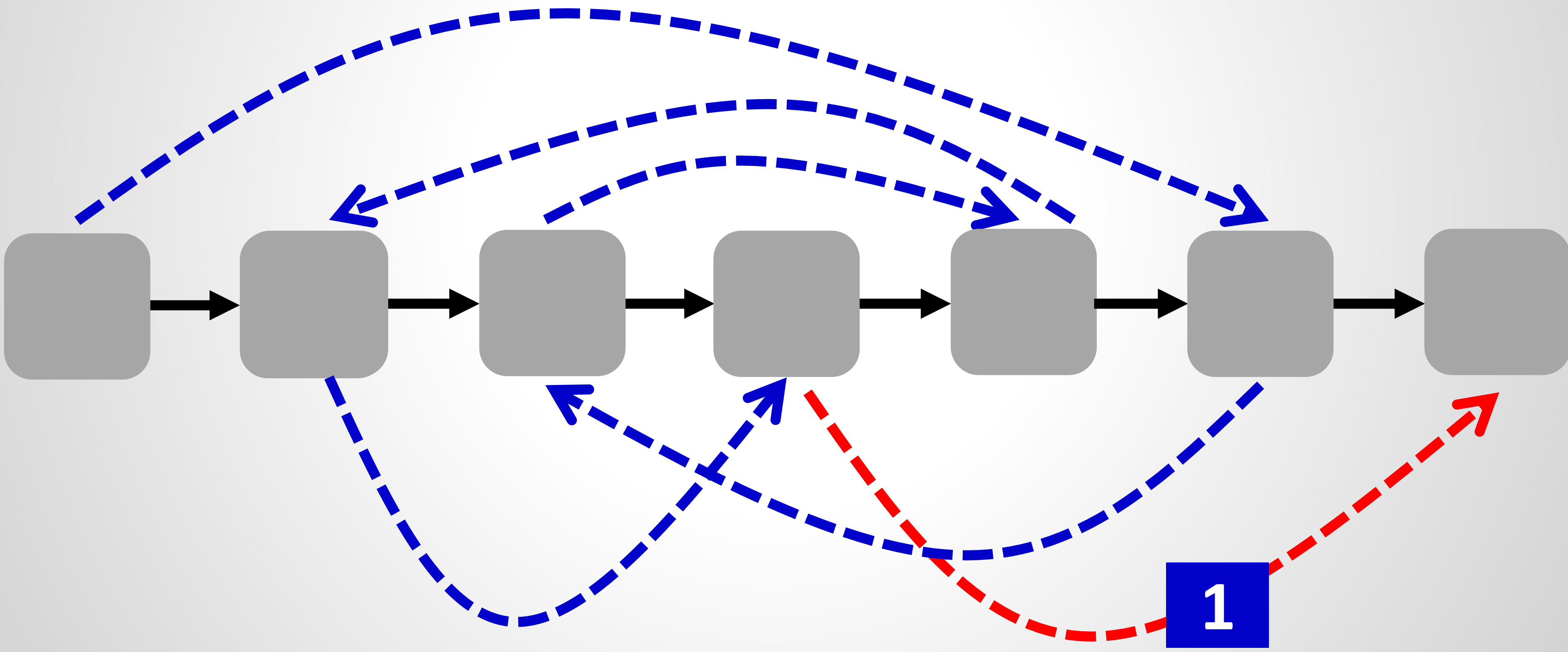


Bad news: Even decidability hard: cannot quickly test feasibility and if infeasible resort to say, tagging solution!

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

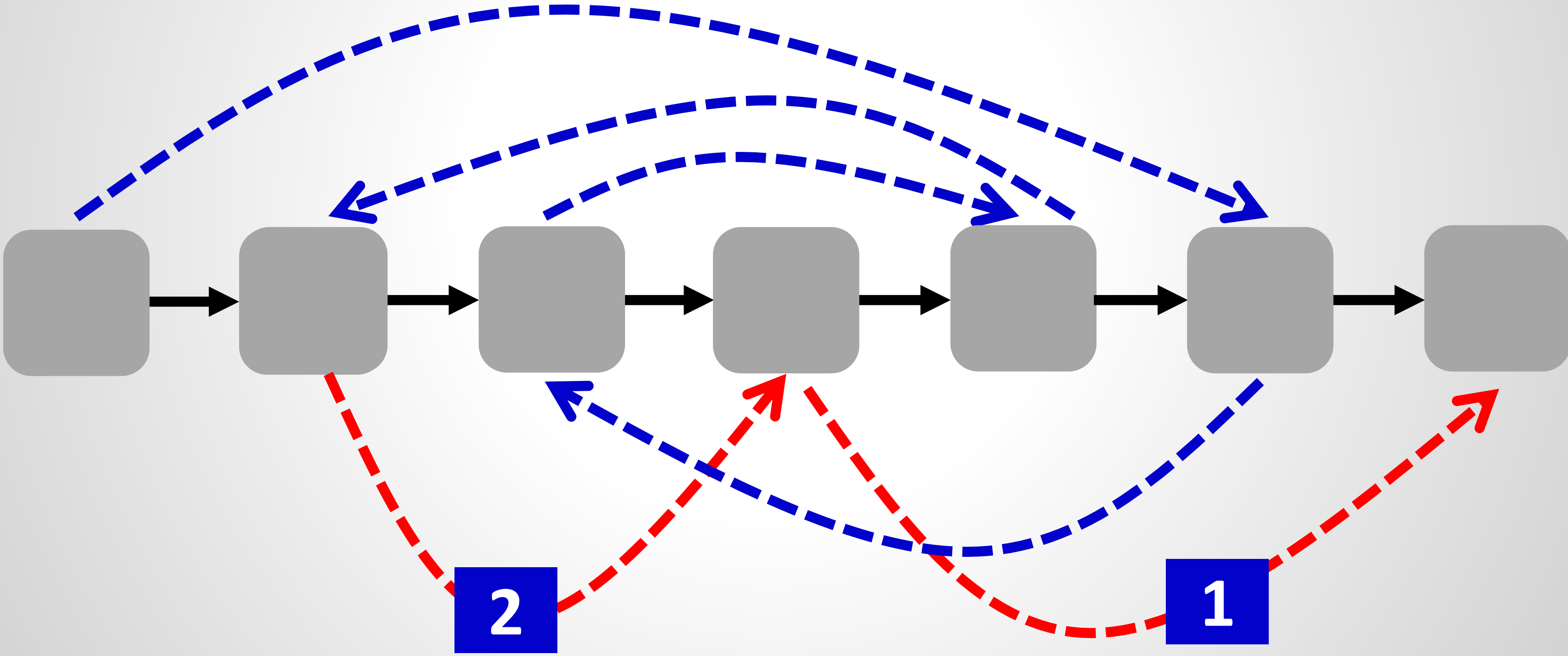
What about loop-freedom only? Always possible in *n* rounds!

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



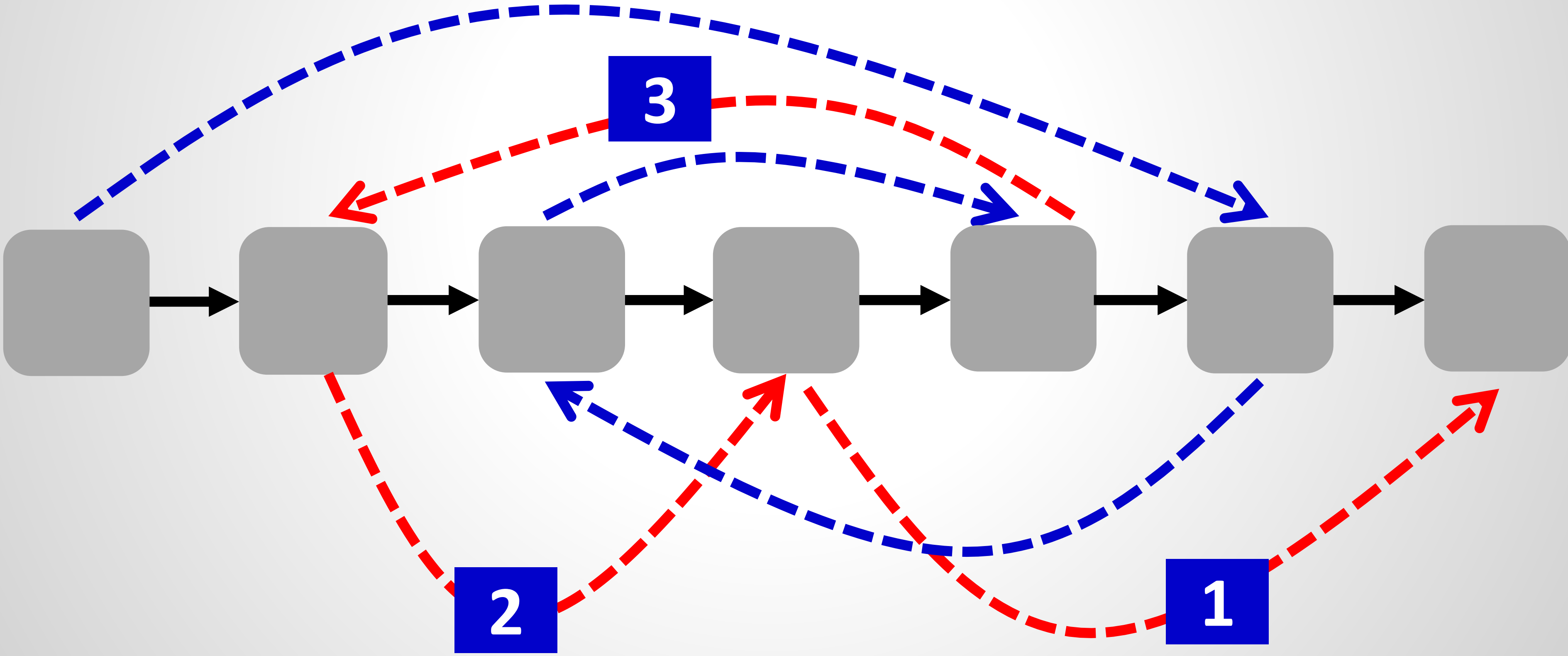
From the destination! Invariant: path suffix updated!

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



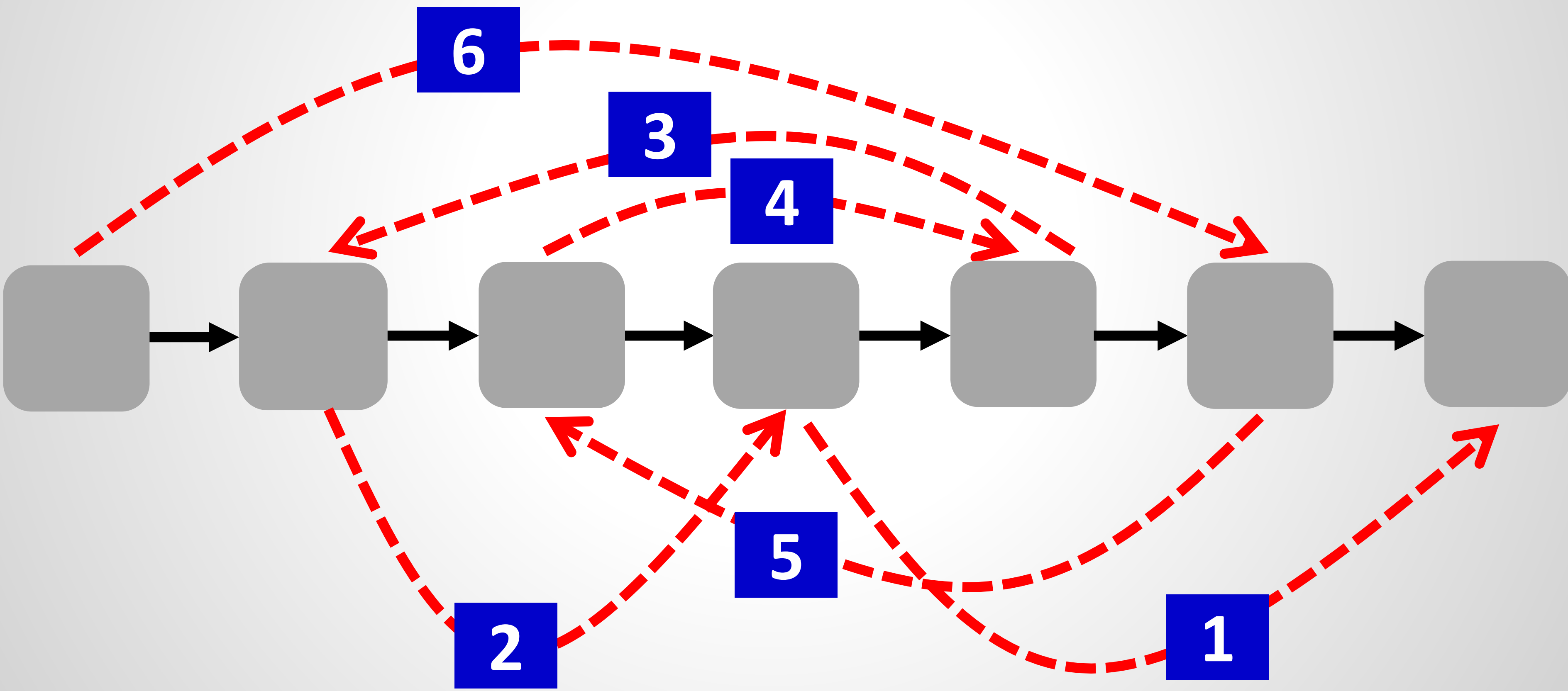
From the destination! Invariant: path suffix updated!

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



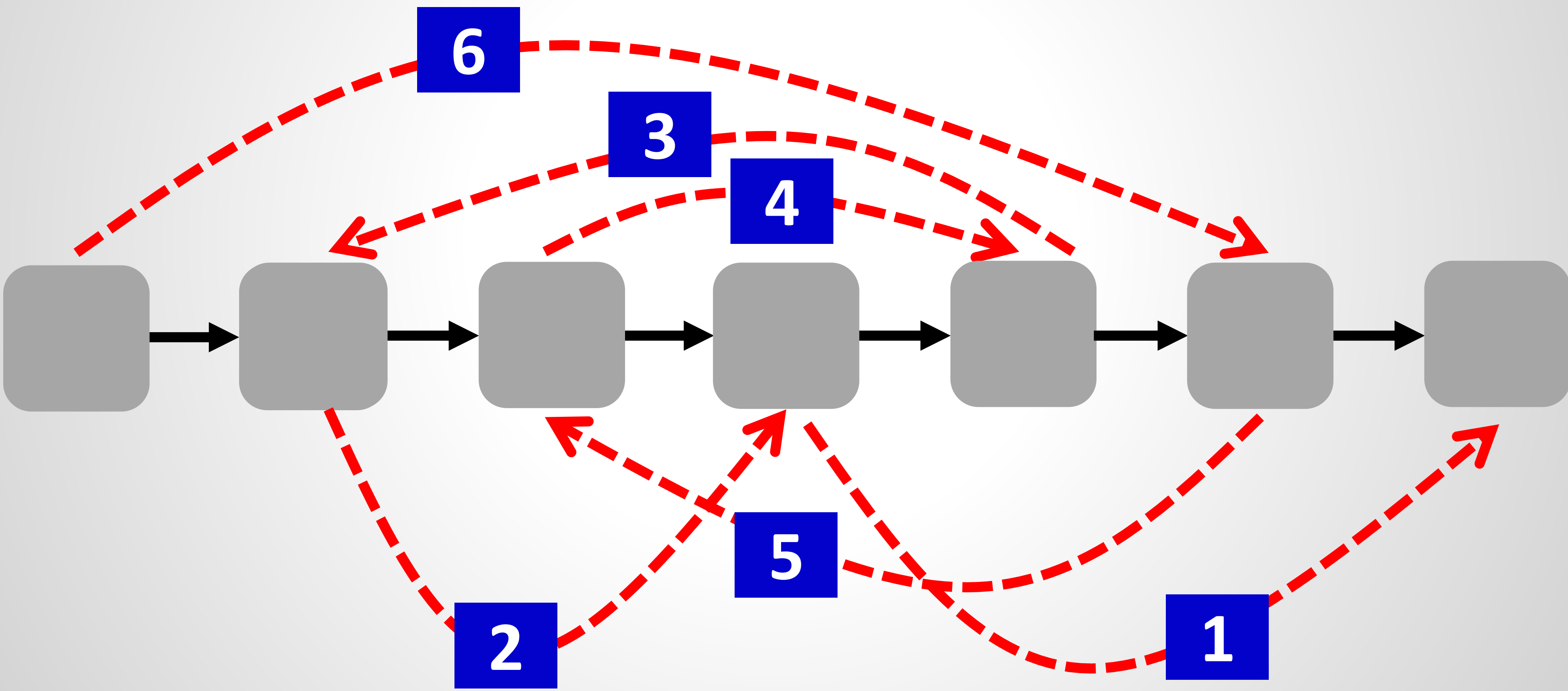
From the destination! Invariant: path suffix updated!

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



From the destination! Invariant: path suffix updated!

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



From the destination! Invariant: path suffix updated!

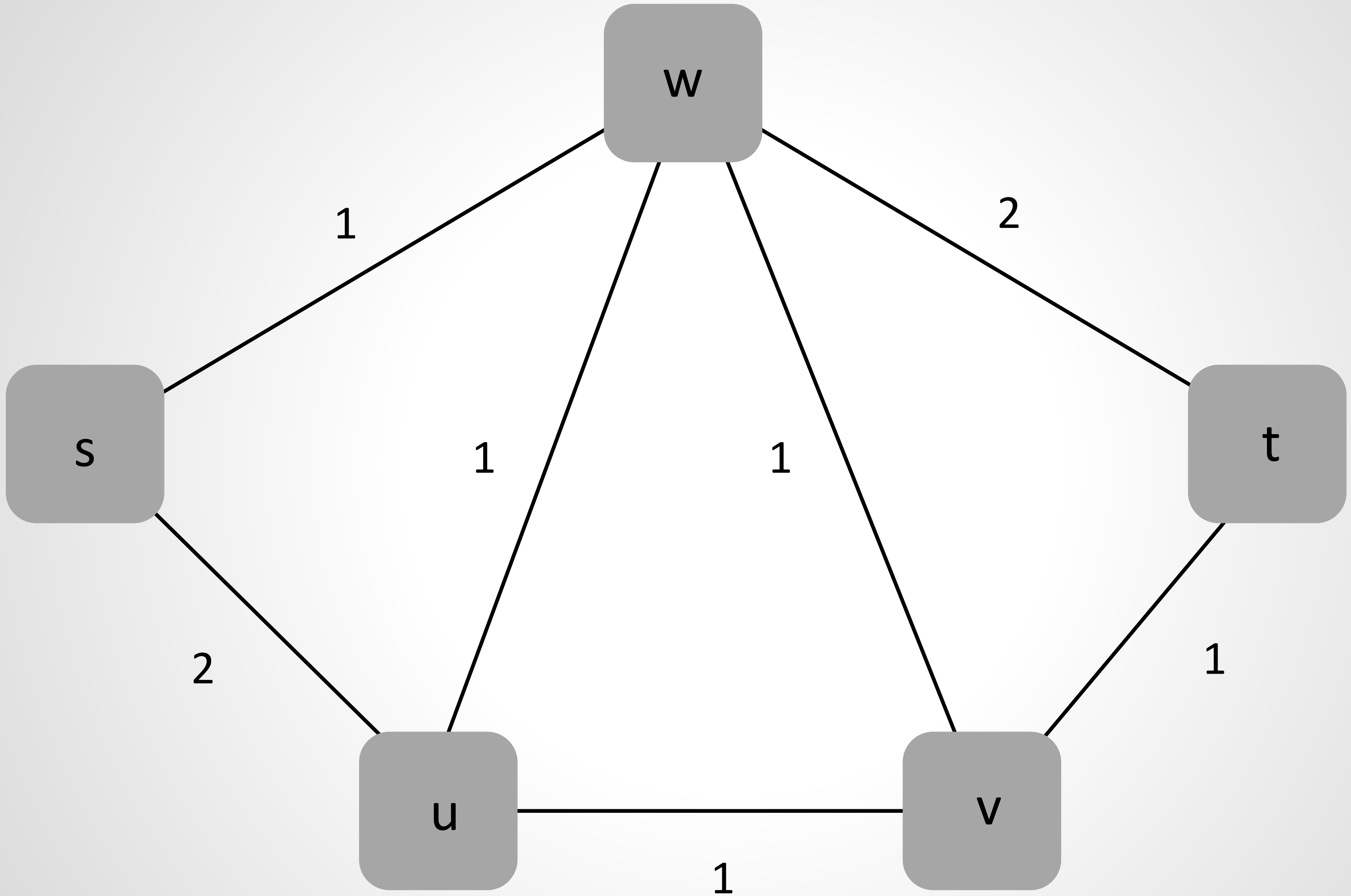
But how to minimize # rounds?

But how to minimize # rounds?

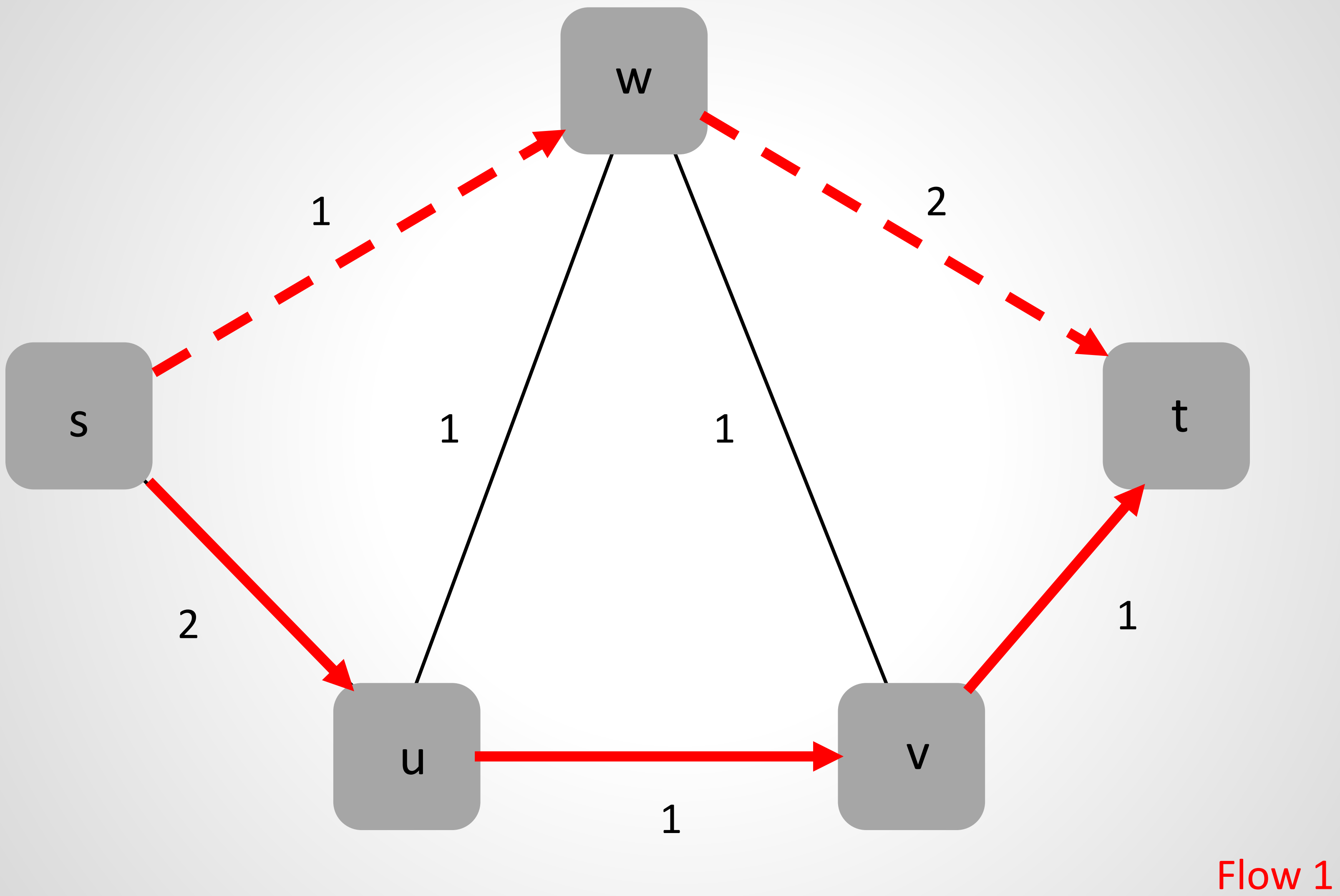
2 rounds easy, 3 rounds NP-hard. Everything else:

We don't know today!

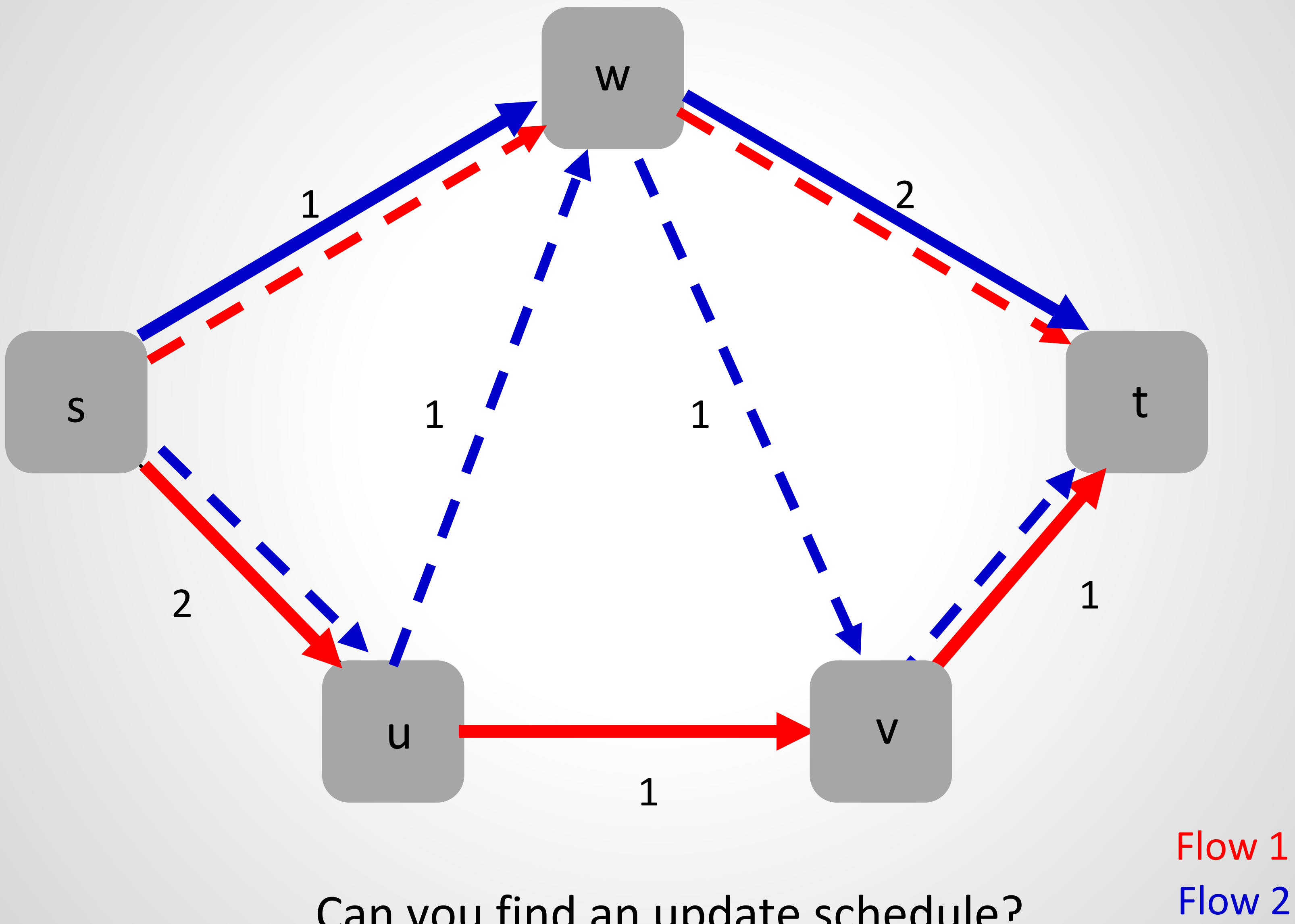
What about capacity constraints?



What about capacity constraints?

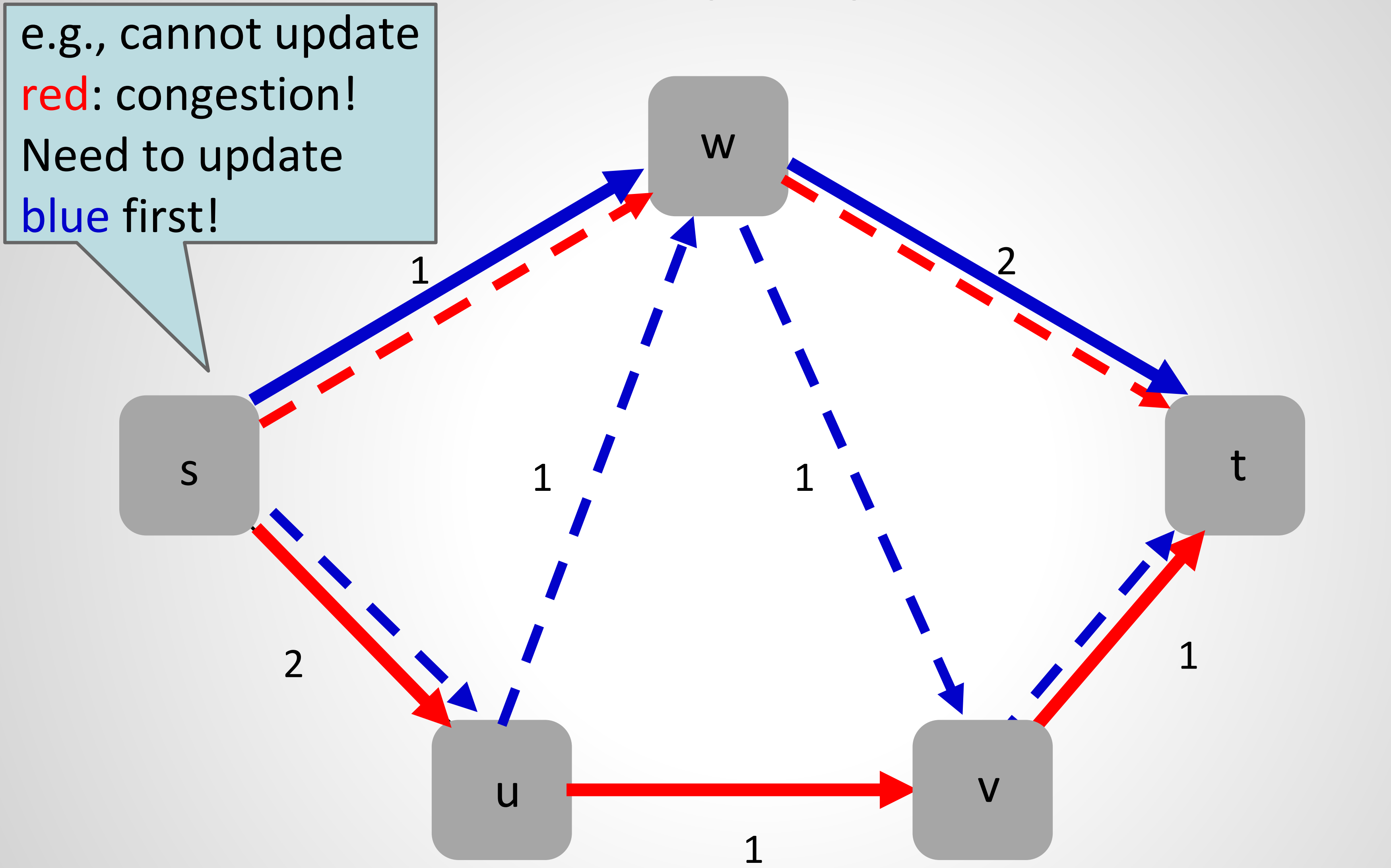


What about capacity constraints?



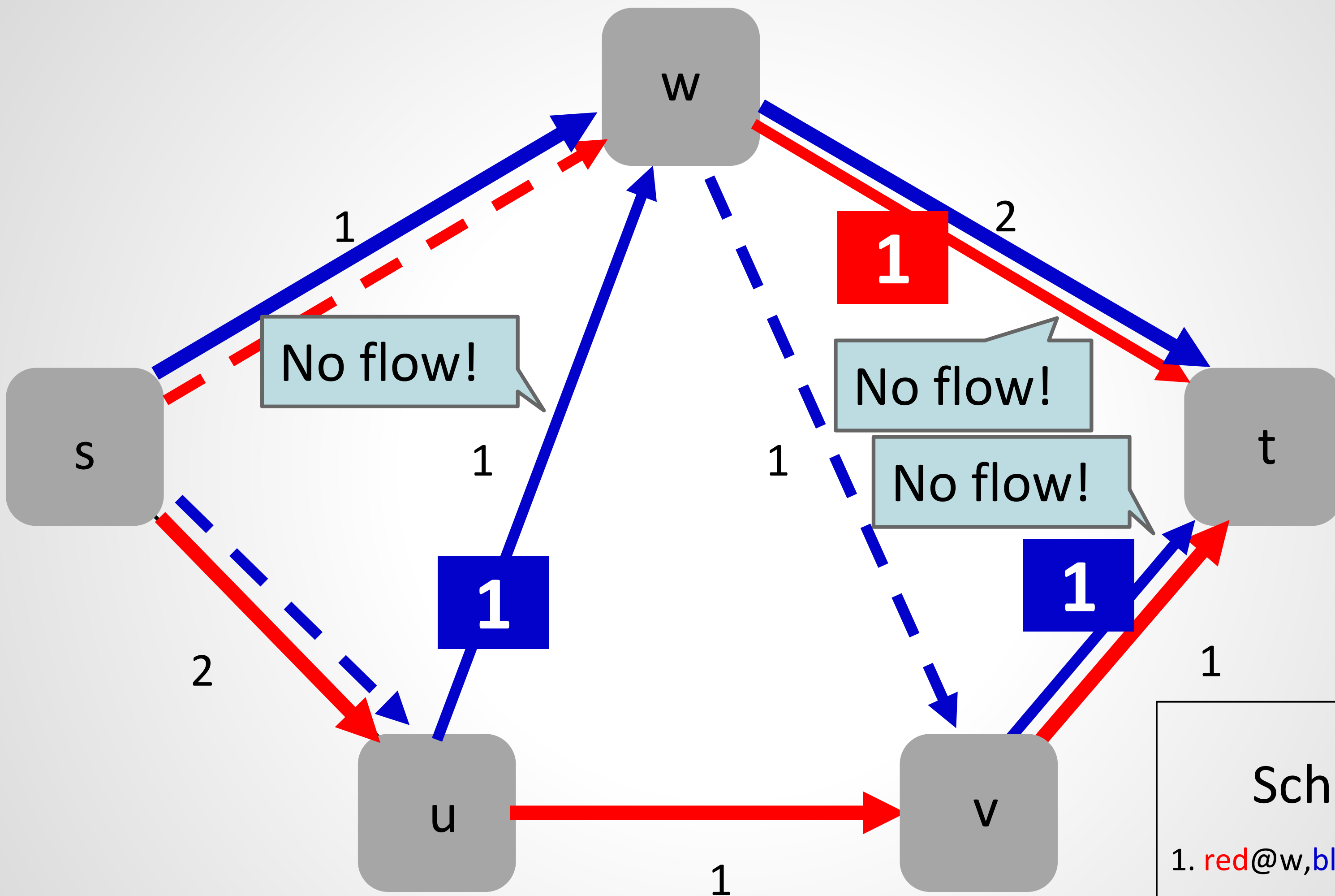
Can you find an update schedule?

What about capacity constraints?



Can you find an update schedule?

What about capacity constraints?

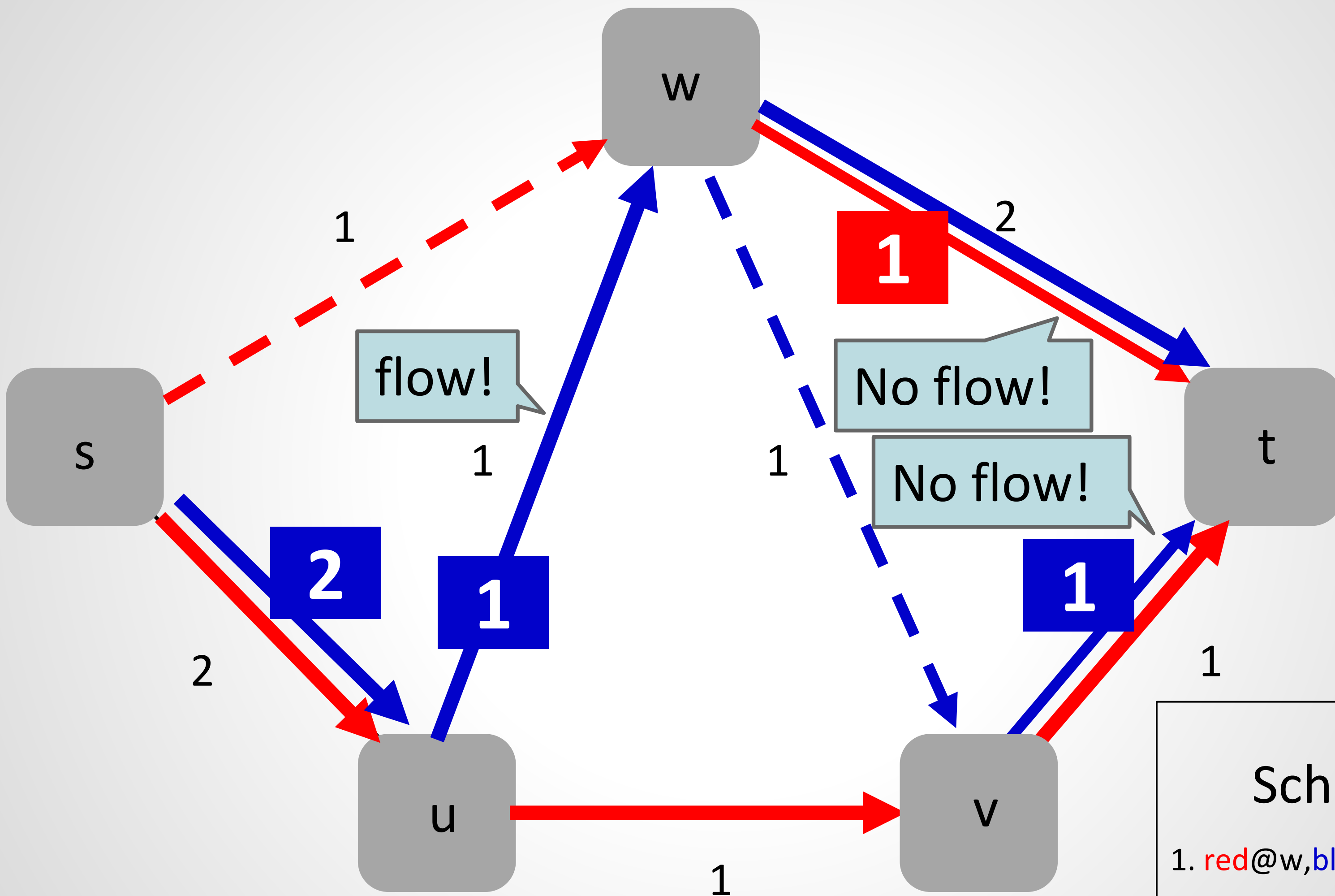


Schedule:

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

Round 1: prepare

What about capacity constraints?

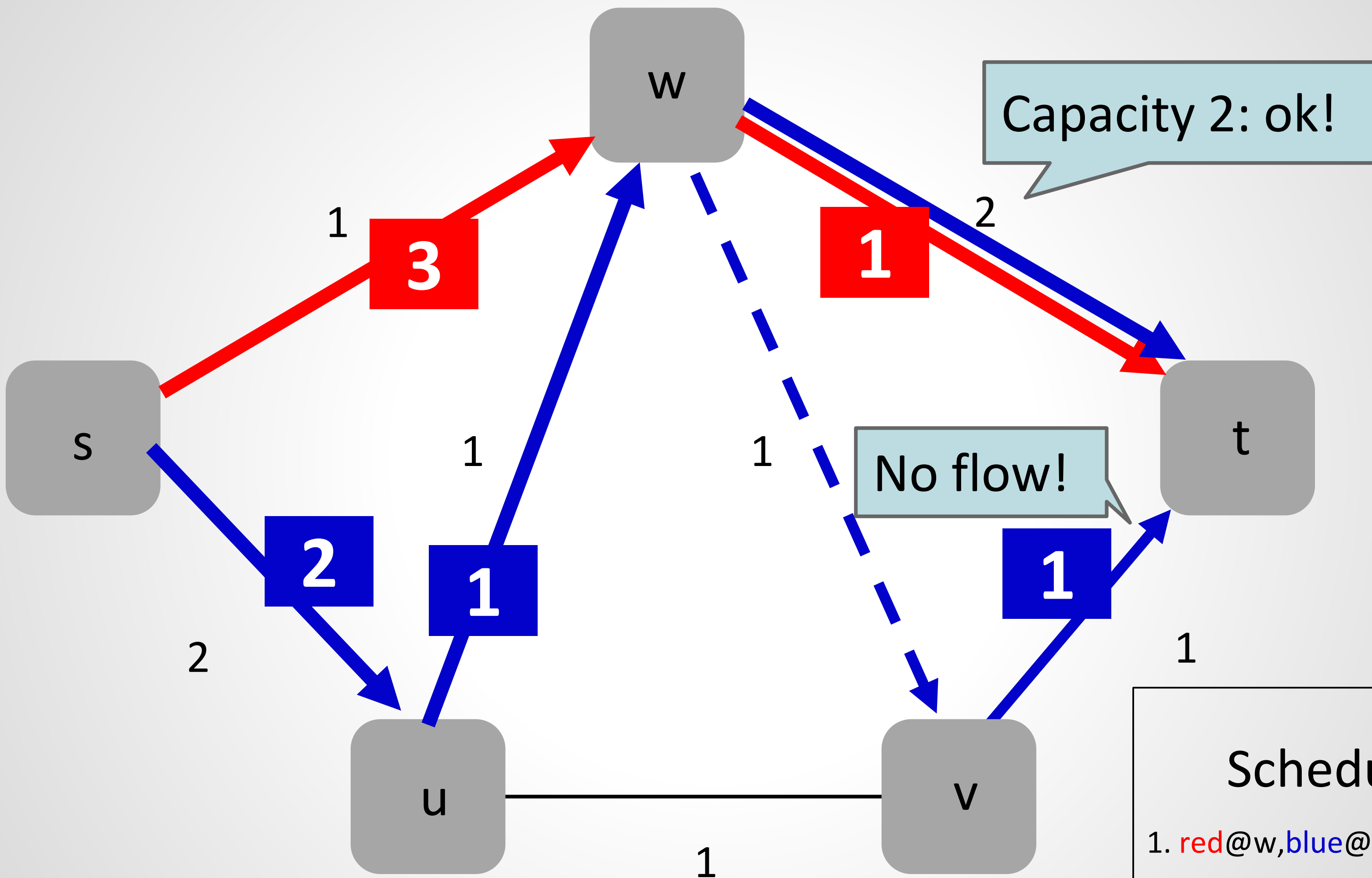


Schedule:

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

Round 2

What about capacity constraints?

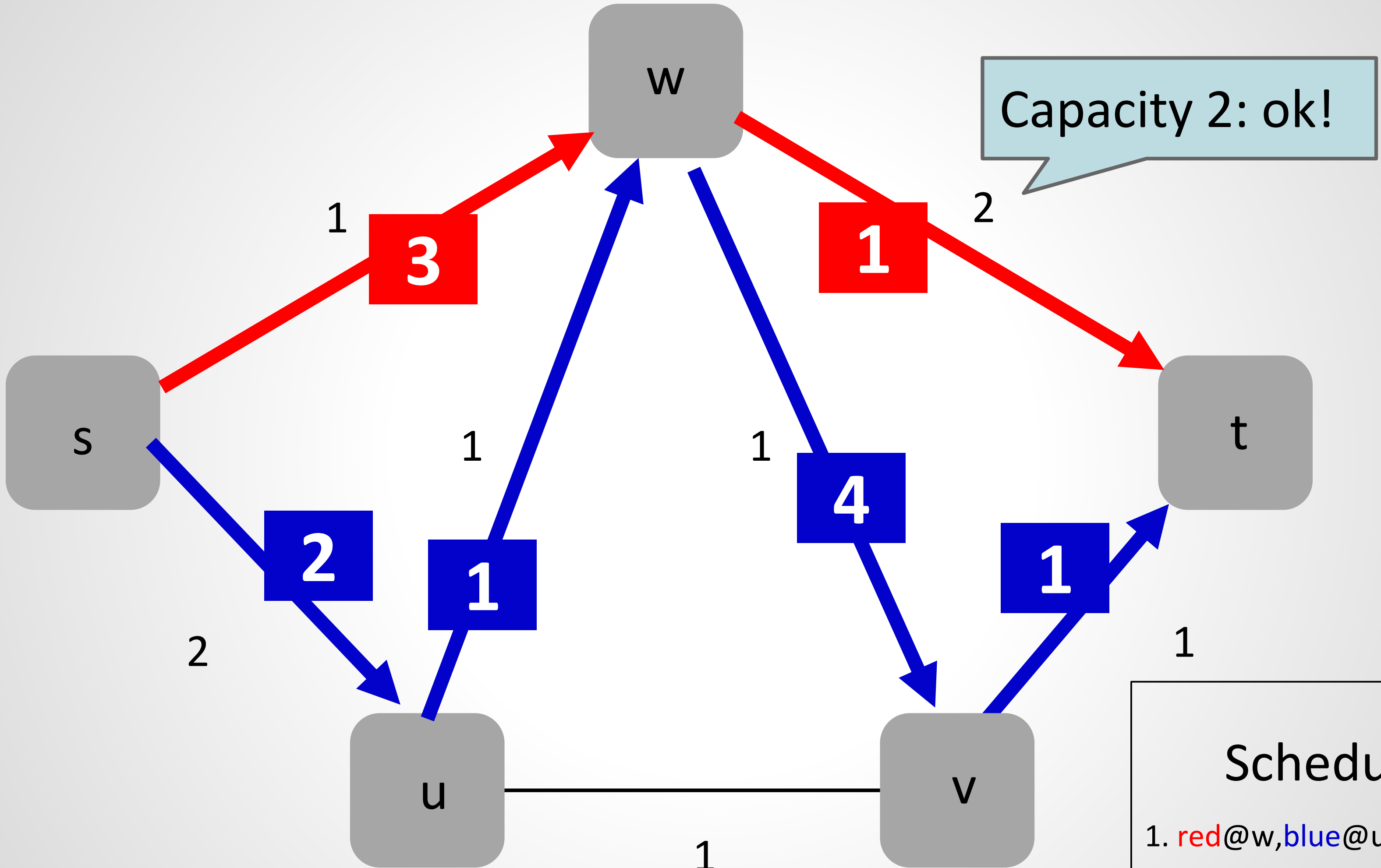


Round 3

Schedule:

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

What about capacity constraints?

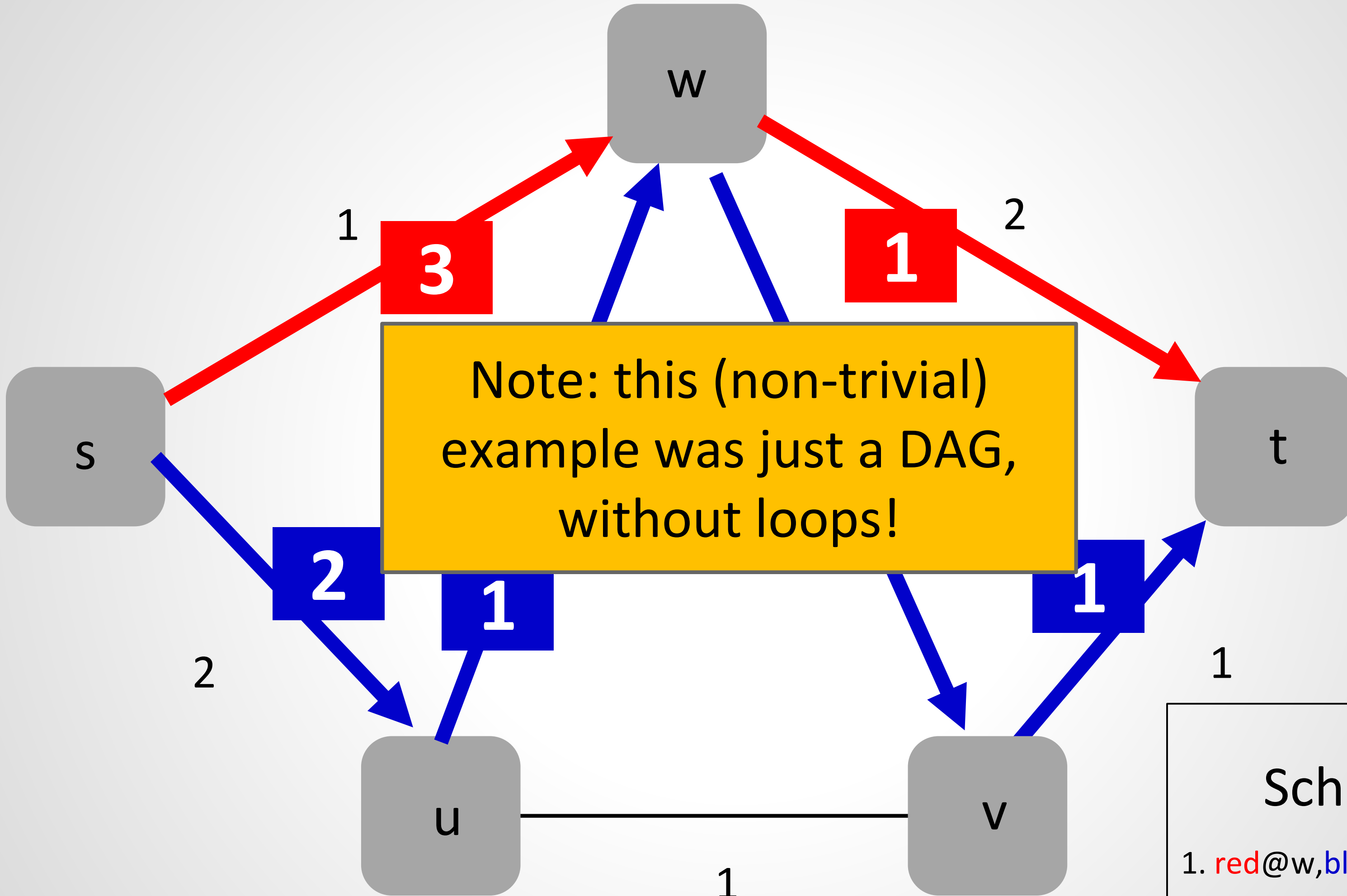


Schedule:

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!

Schedule:

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

Round 4

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

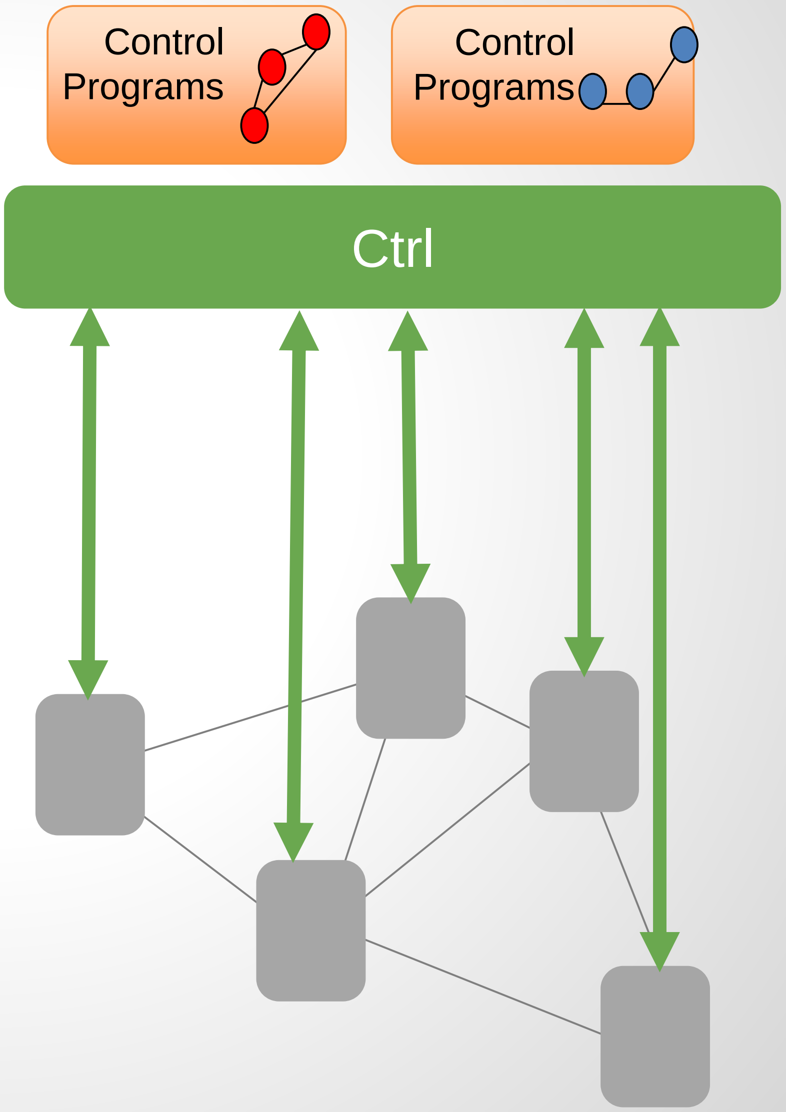
Congestion-Free Rerouting of Flows on DAGs
Saeed Akhoondian Amiri, Szymon Dudycz, Stefan Schmid, and Sebastian Wiederrecht.
ArXiv Technical Report, November 2016.

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

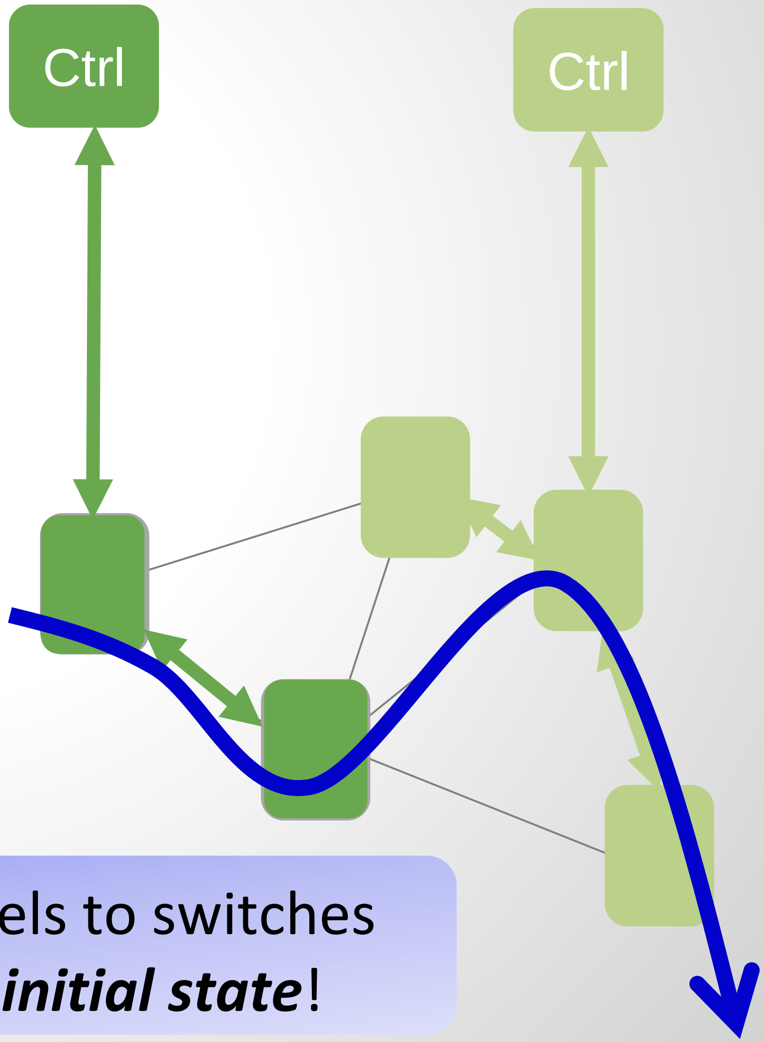


Challenge: How to maintain connectivity?

In-band Management

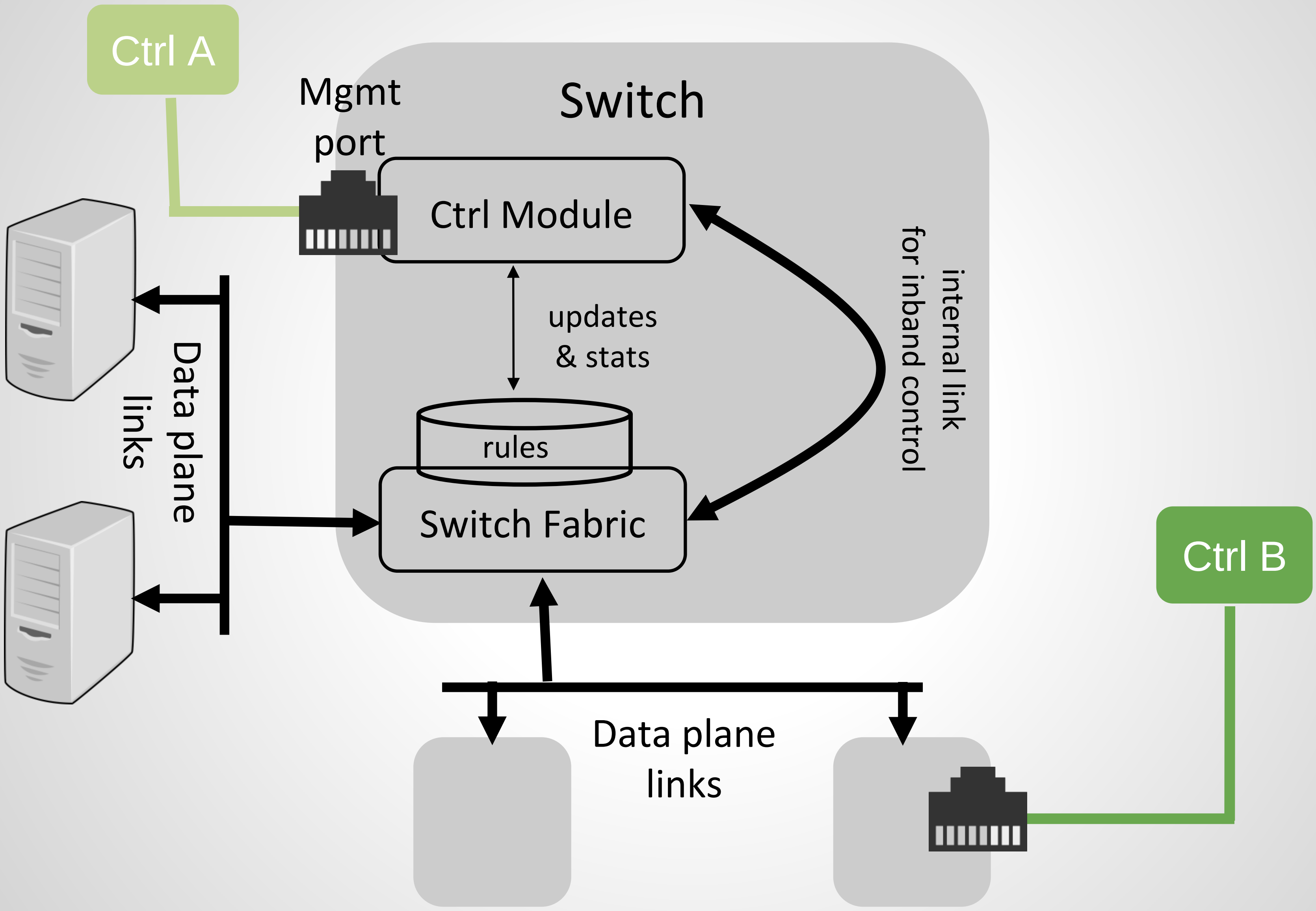
How to provide connectivity between the planes?

- ❑ In large-scale networks: distributed…
- ❑ … and inband control
  - ❑ Control and data plane traffic interleaved: arrives at the same port
  - ❑ No need for dedicated infrastructure

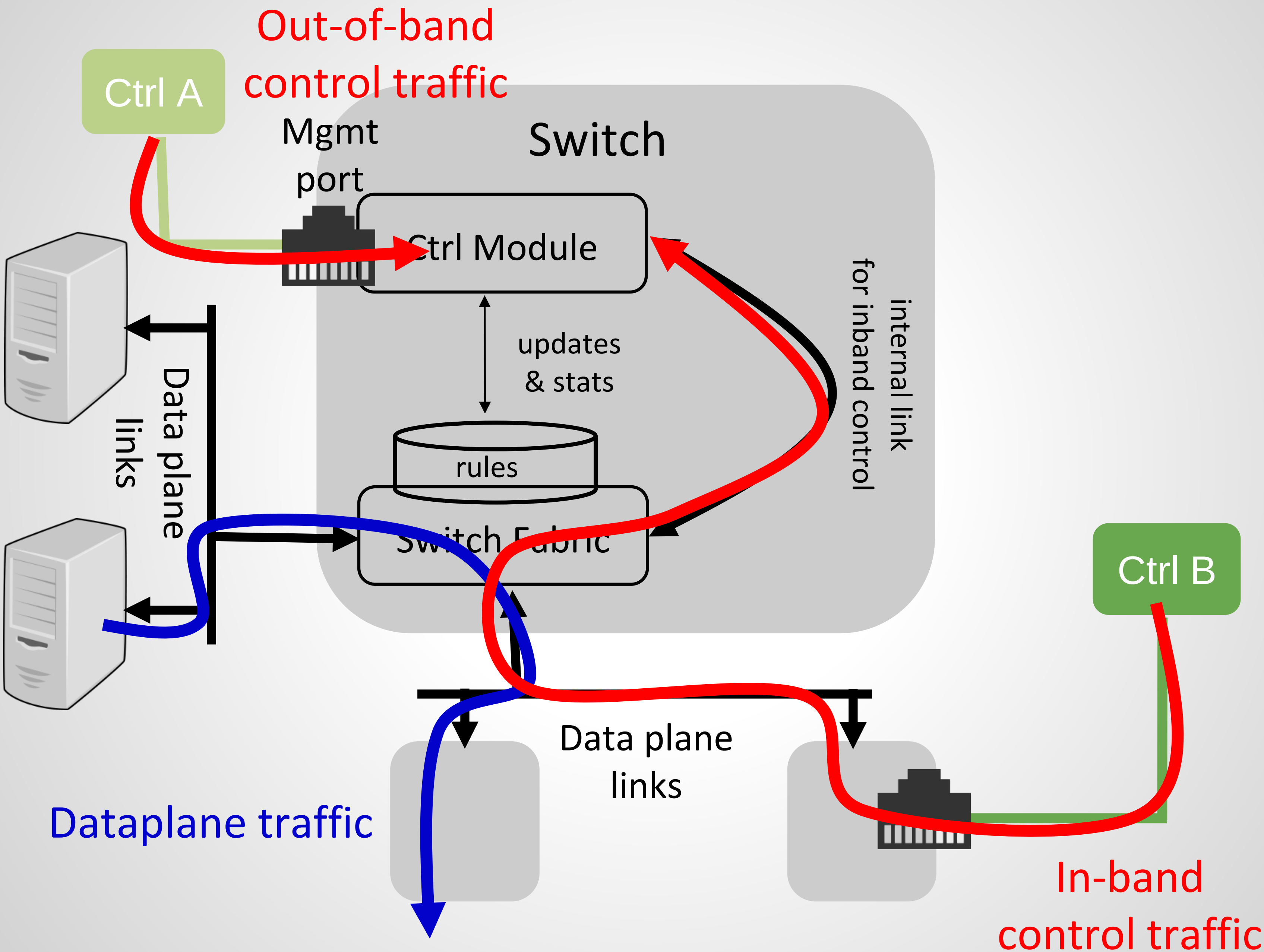


Ideally, self-stabilizing: ensure channels to switches and between controllers ***from any initial state***!

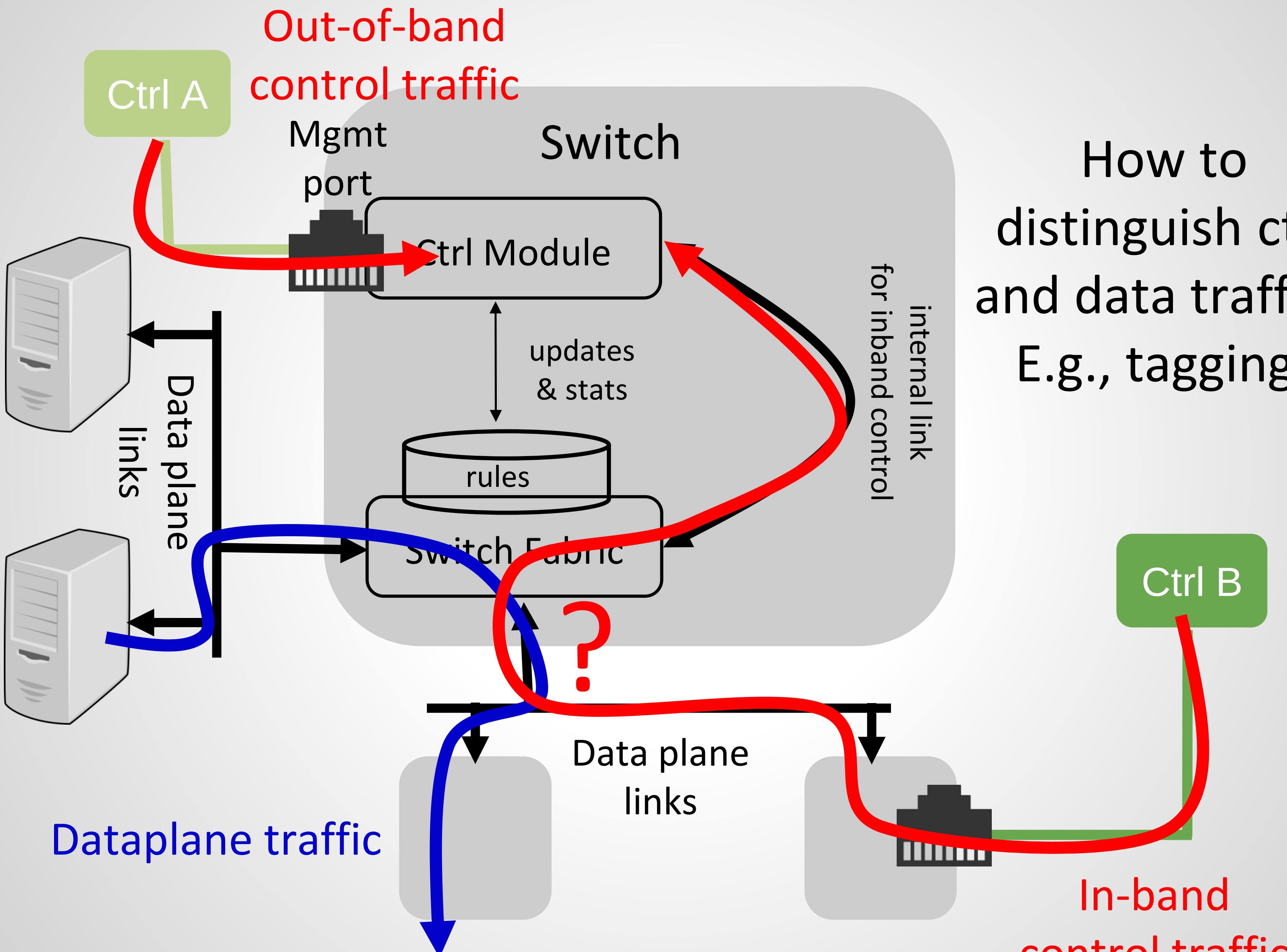
Self-Stabilizing Connectivity is Non-Trivial



Self-Stabilizing Connectivity is Non-Trivial

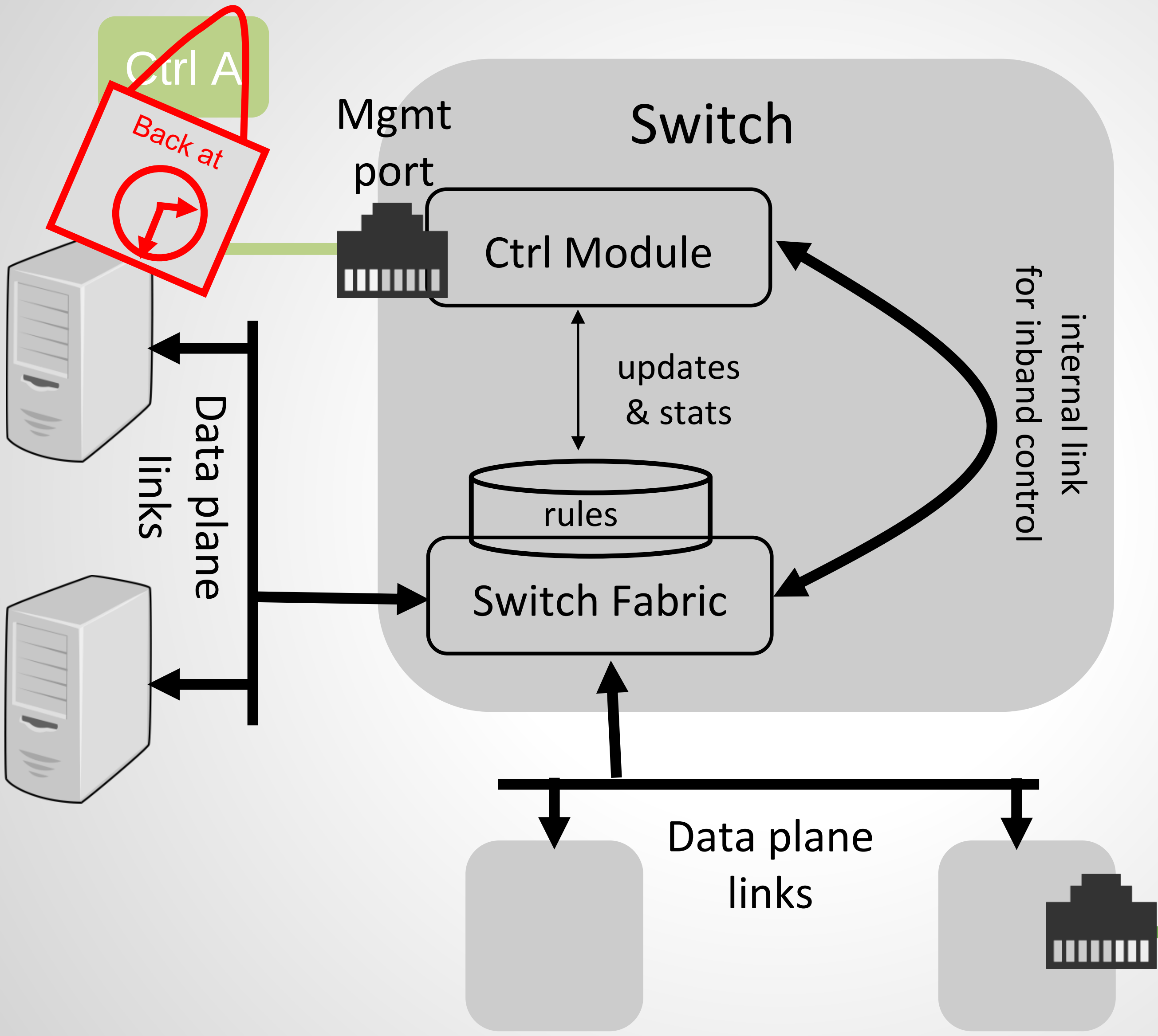


Self-Stabilizing Connectivity is Non-Trivial



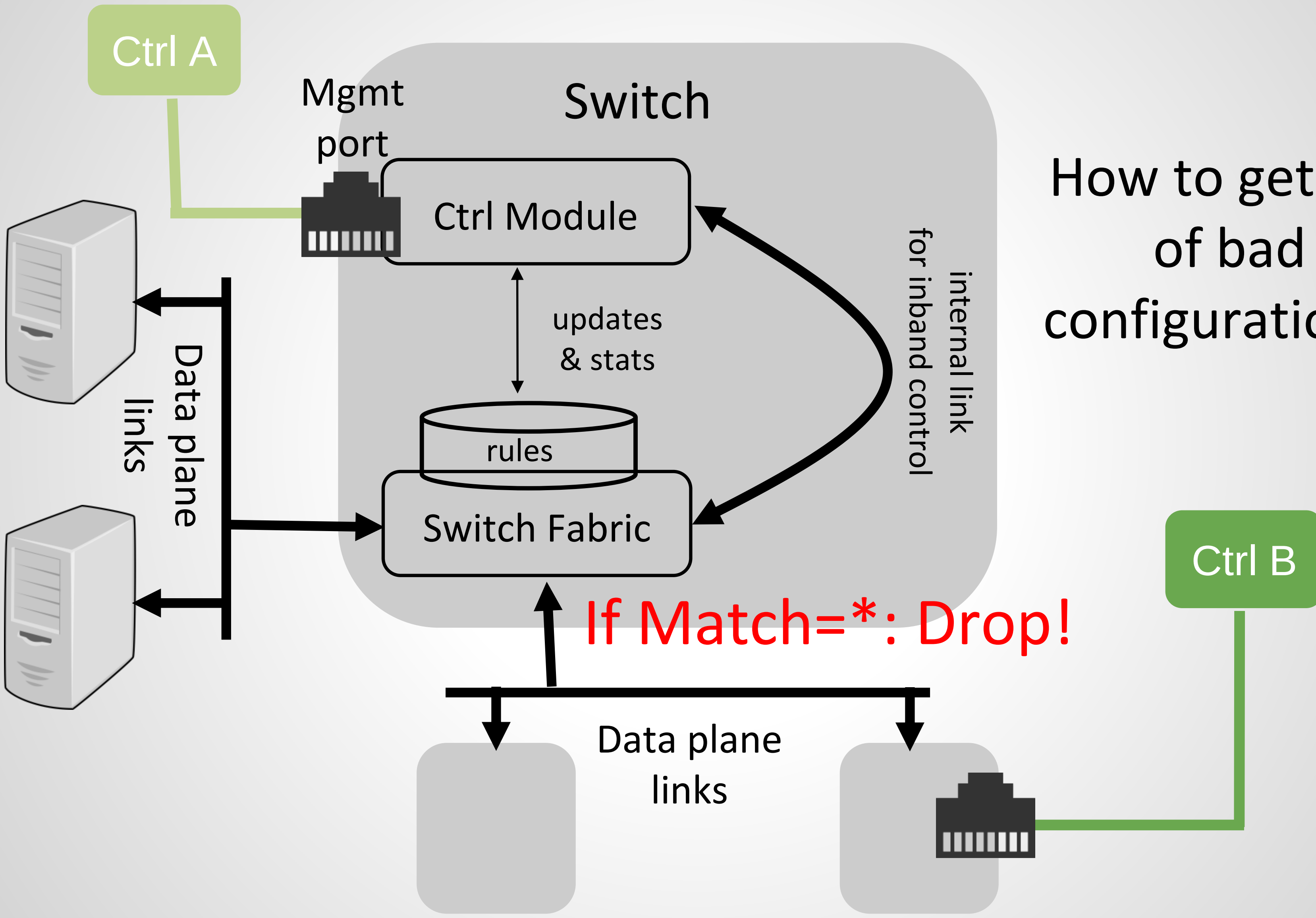
How to distinguish ctrl and data traffic? E.g., tagging!

Self-Stabilizing Connectivity is Non-Trivial



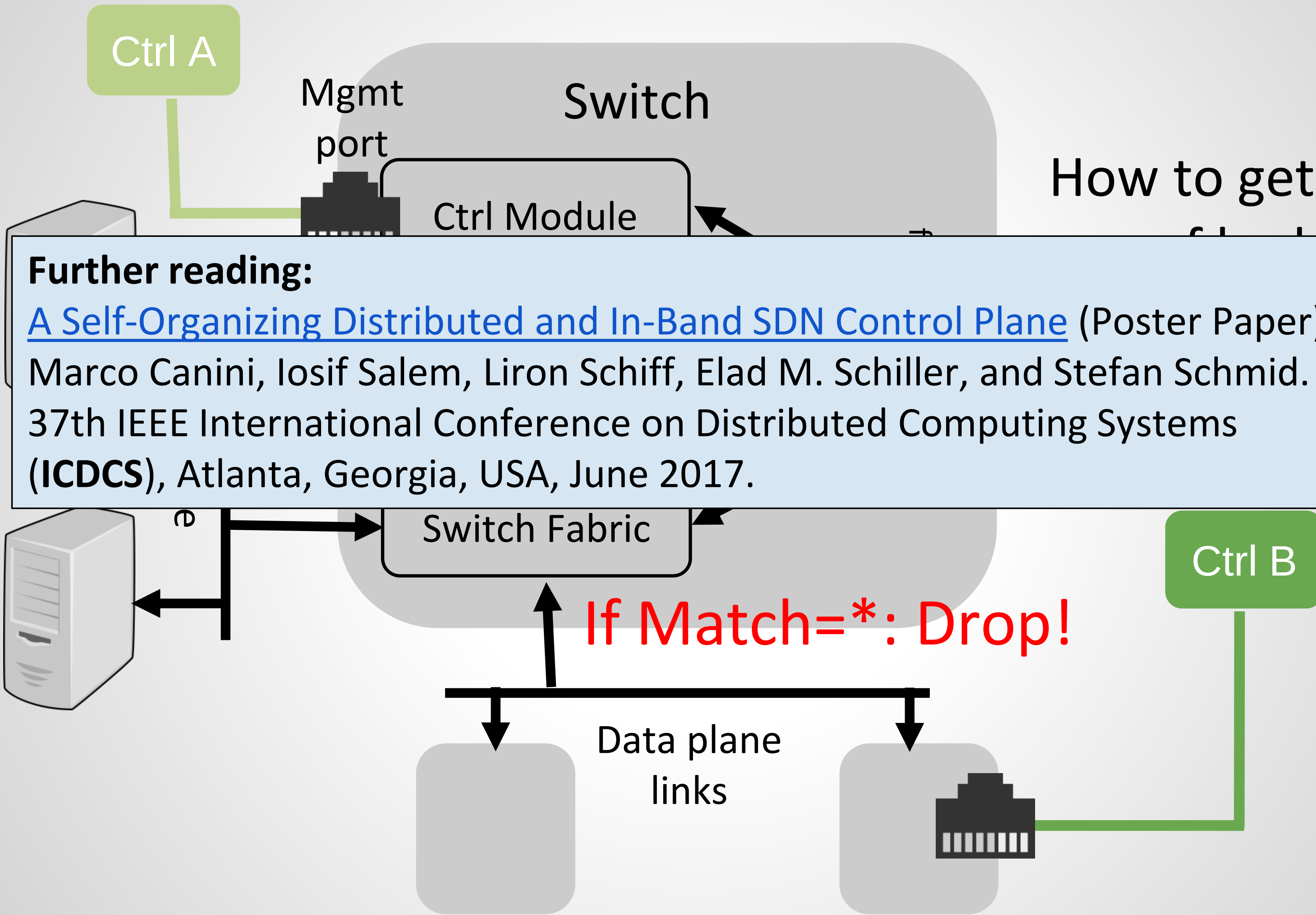
How can Ctrl B know that Ctrl A failed?

Self-Stabilizing Connectivity is Non-Trivial



How to get out of bad configurations?

Self-Stabilizing Connectivity is Non-Trivial

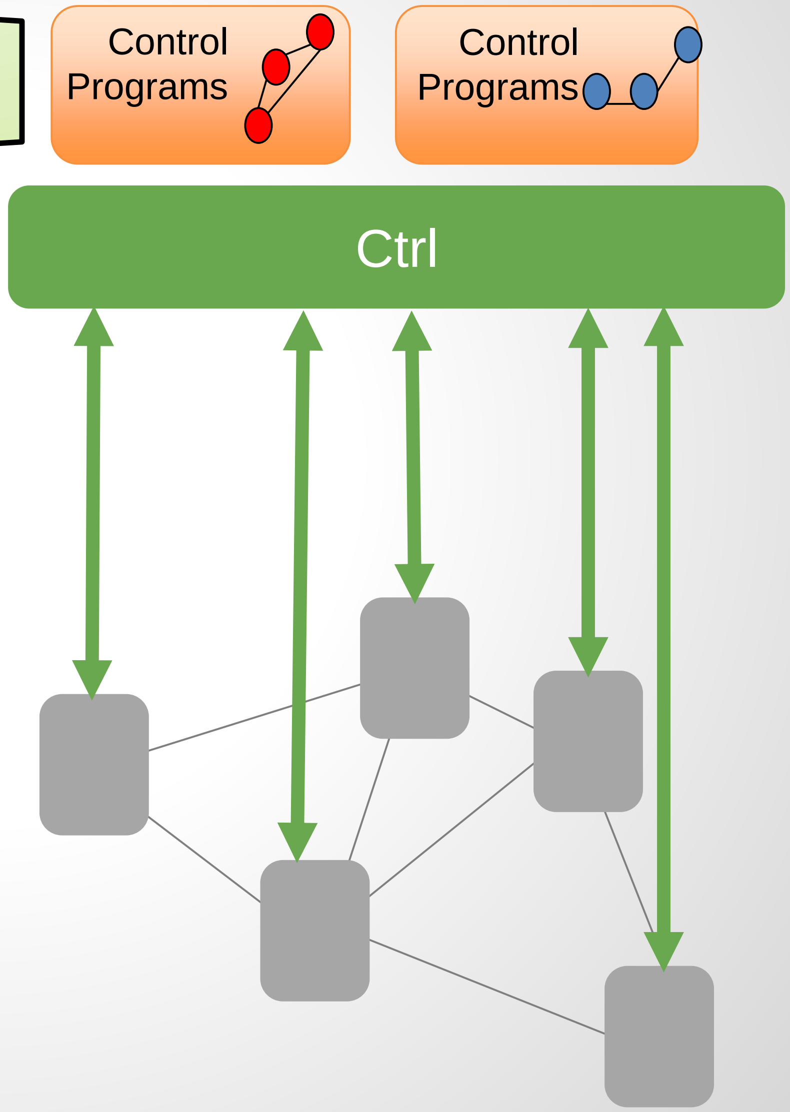


How to get out ... s?

Further reading:
A Self-Organizing Distributed and In-Band SDN Control Plane (Poster Paper)
Marco Canini, Iosif Salem, Liron Schiff, Elad M. Schiller, and Stefan Schmid.
37th IEEE International Conference on Distributed Computing Systems (**ICDCS**), Atlanta, Georgia, USA, June 2017.

A Mental Model for This Talk

Opportunity: innovative services and algorithms



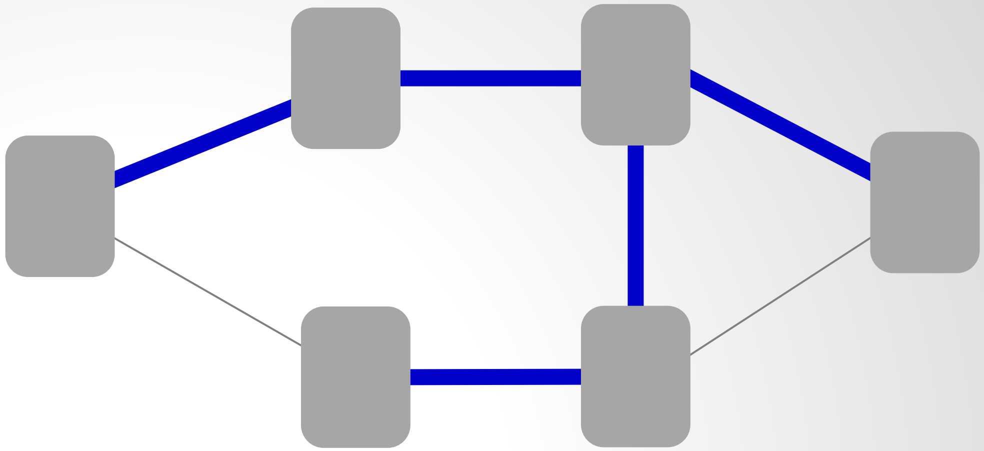
Example Benefit 1: Lying

- Cannot only find innovative routing algorithms etc. …
- … but also interact with and manipulate legacy networks in novel ways («hybrid SDNs»)
- E.g., trick it into better traffic engineering, faster failover, etc.

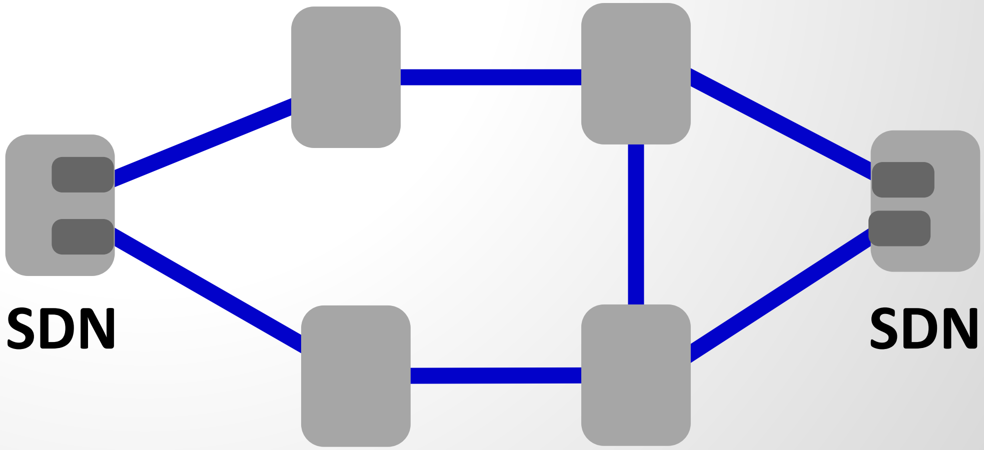
Credits: Marco Chiesa

Example Benefit 1: Lying

STP in legacy network:
loop-free

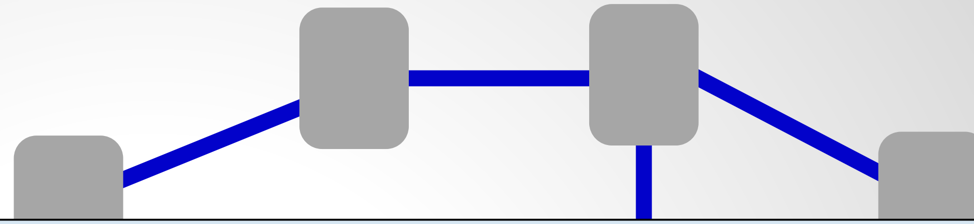


Improved capacity:
STP in legacy network
still loop-free



Example Benefit 1: Lying

STP in legacy network:



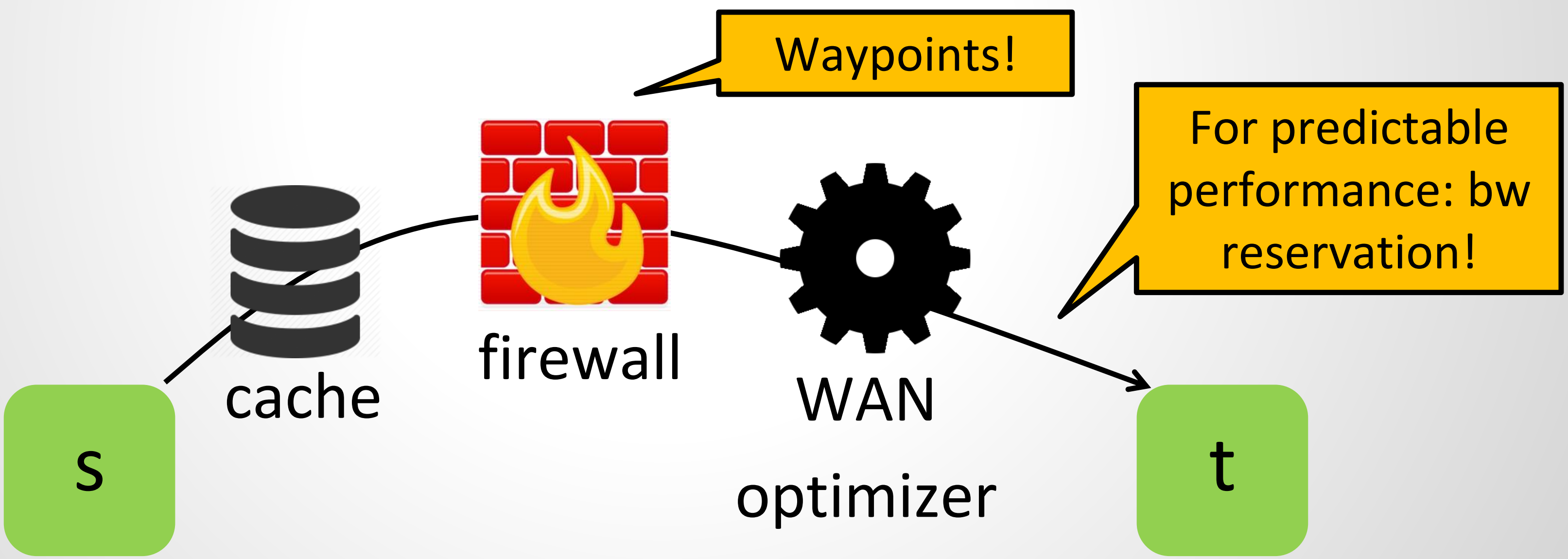
Further reading:

SHEAR: A Highly Available and Flexible Network Architecture: Marrying Distributed and Logically Centralized Control Planes
Michael Markovitch and Stefan Schmid.
23rd IEEE International Conference on Network Protocols (**ICNP**), San Francisco, California, USA, November 2015.

Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks
Dan Levin, Marco Canini, Stefan Schmid, Fabian Schaffert, and Anja Feldmann.
USENIX Annual Technical Conference (**ATC**), Philadelphia, Pennsylvania, USA, June 2014.

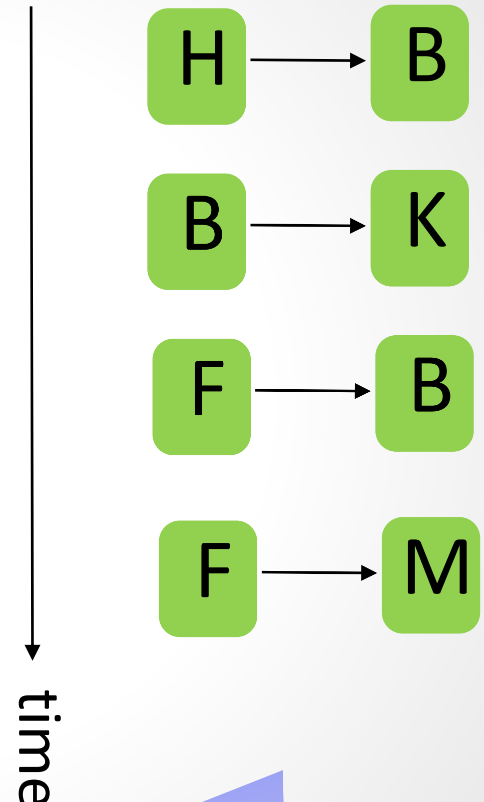
Example Benefit 2: Flexible Waypoint Routing

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



What is new and interesting here?

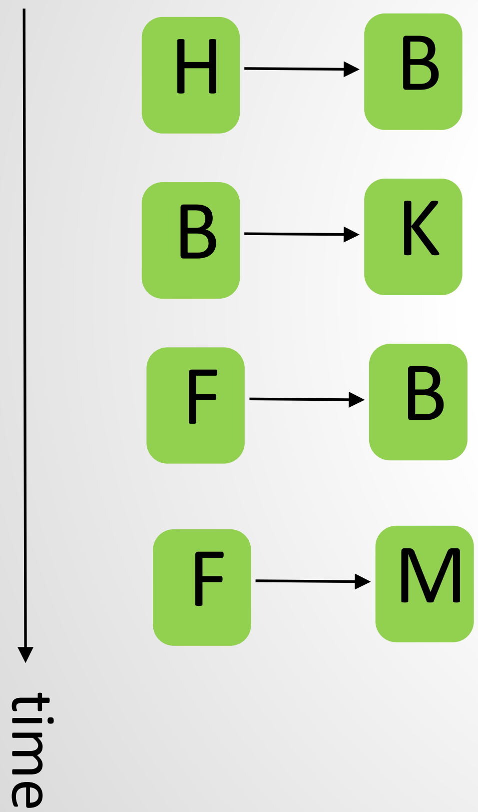
Generalizes call admission!



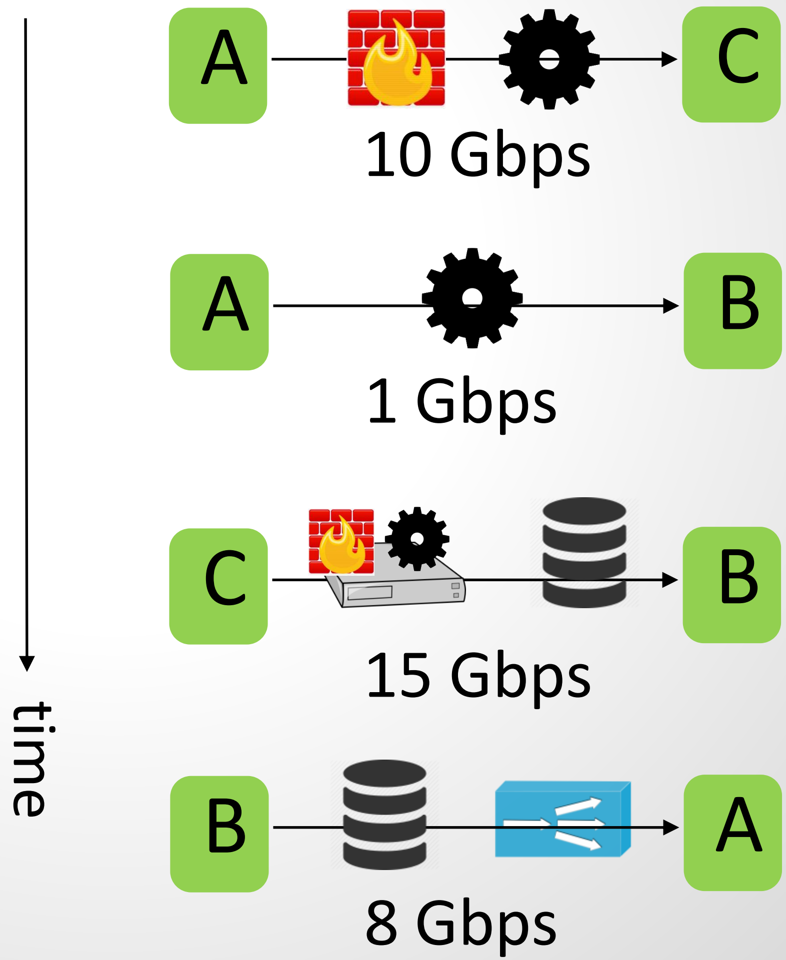
Which calls to admit? And how to route them? Limited resources!

What is new and interesting here?

Online call admission:

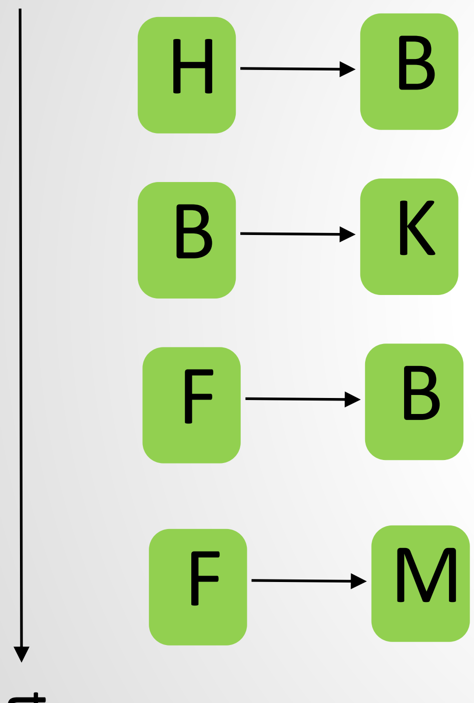


Admit and route requests through waypoints:

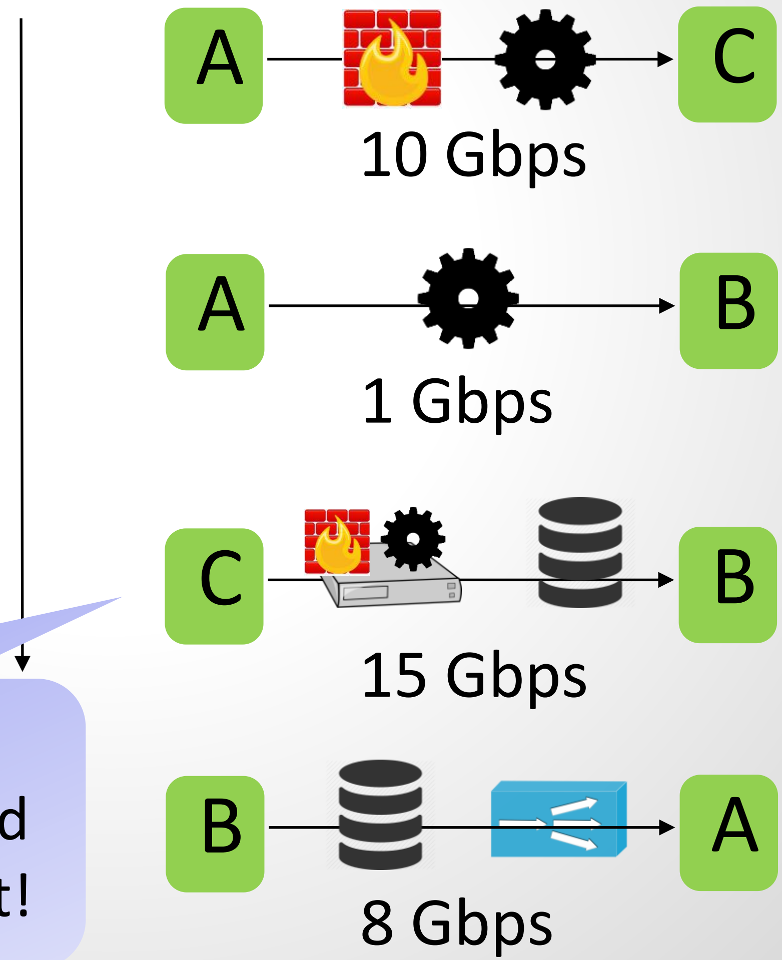


What is new and interesting here?

Online call admission:

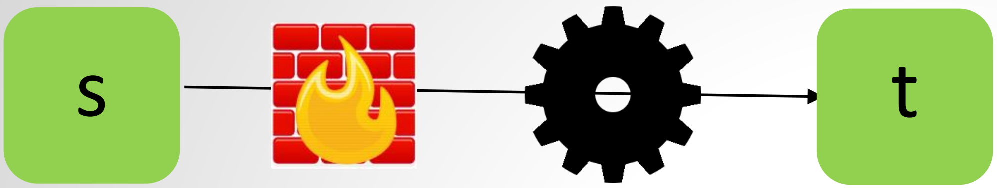


Admit and route requests through waypoints:



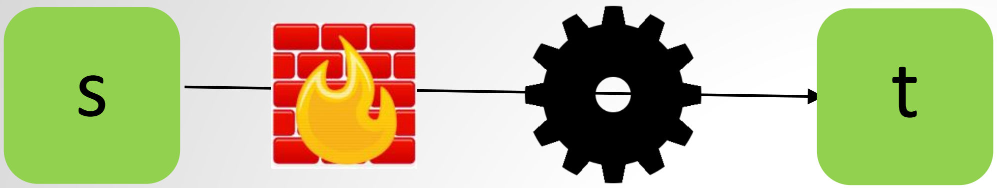
Harder than embedding segments like in calls: need to admit **all or no** segment!

What do we know today…
… *about complex requests?*



How to embed s.t. resource footprint is minimal?

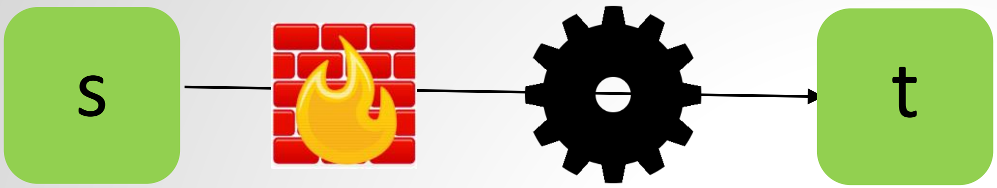
What do we know today…
… *about complex requests?*



How to embed s.t. resource footprint is minimal?

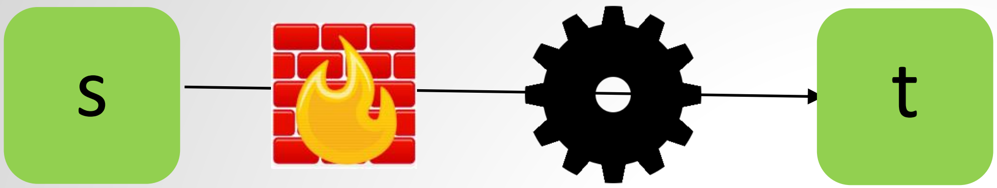
Fairly well-understood if **approximations** are allowed. E.g., reduce to **flow problem** using a **product graph** (and **randomized rounding**)!

What do we know today…
… *about complex requests?*



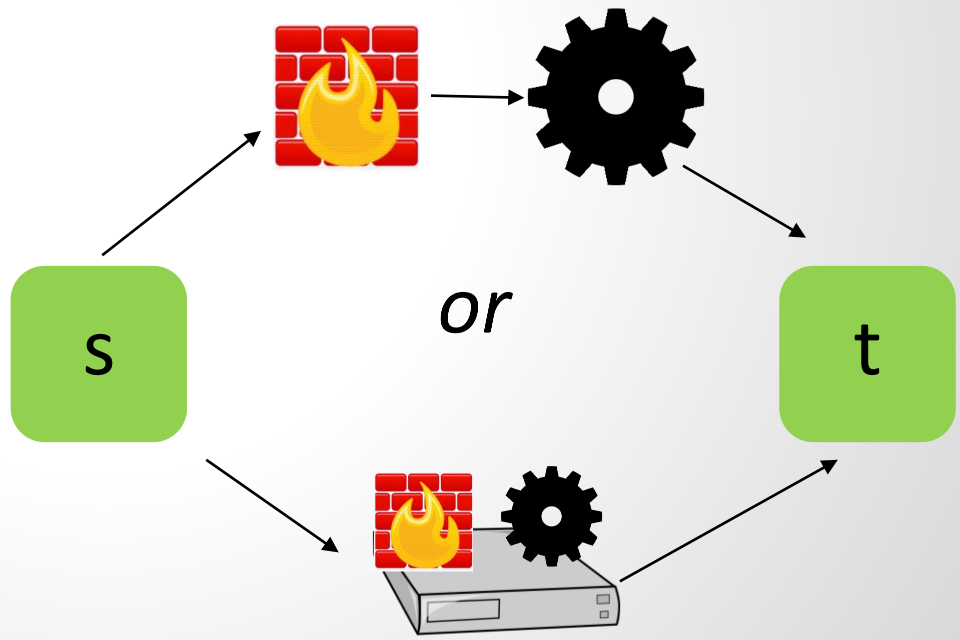
Approximate function chain embedding: fairly well-understood

What do we know today…
… *about complex requests?*

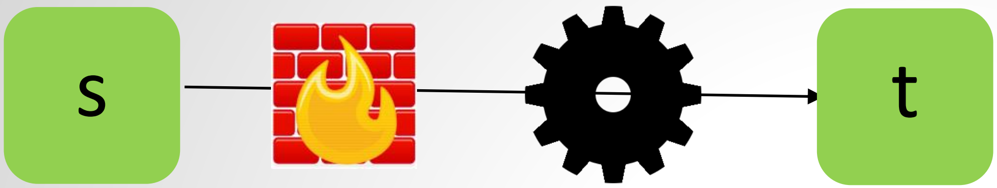


Approximate function chain embedding: fairly well-understood

What about if requests allow for alternatives and different decompositions?

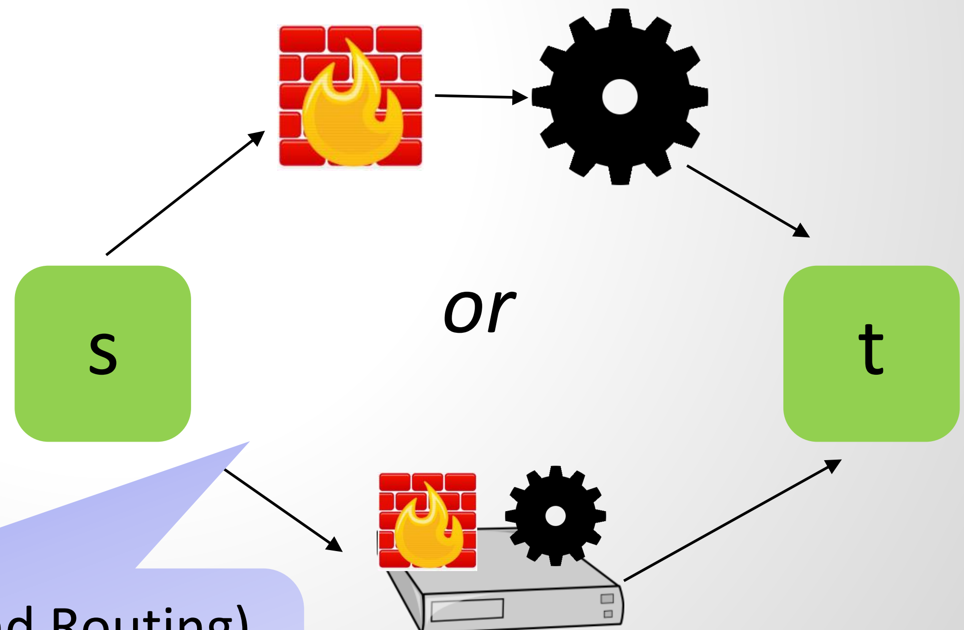


What do we know today…
… *about complex requests?*



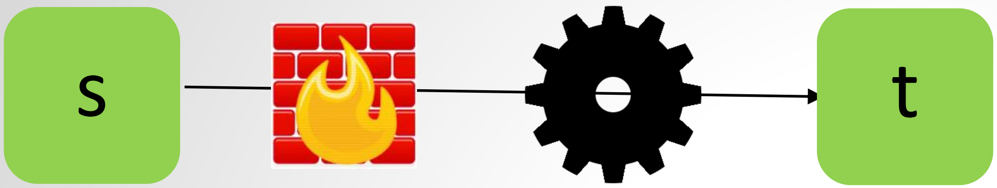
Approximate function chain embedding: fairly well-understood

What about if requests allow for alternatives and different decompositions?



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

What do we know today…
… *about complex requests?*

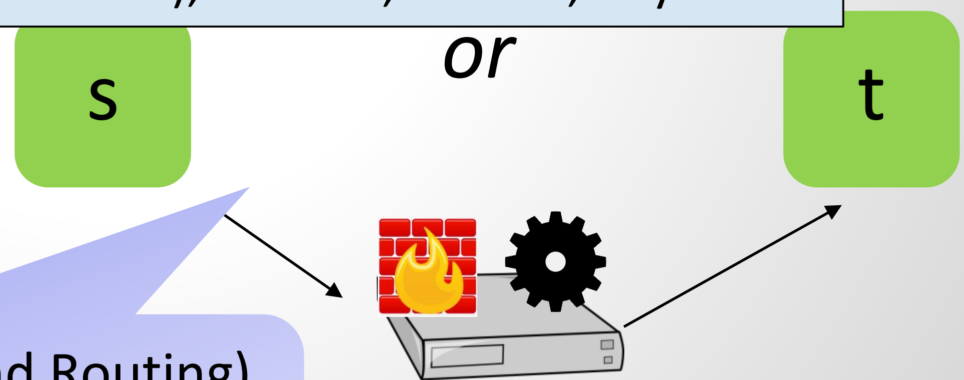


Approximate function chain embedding: fairly well-understood

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.

Wha
allo

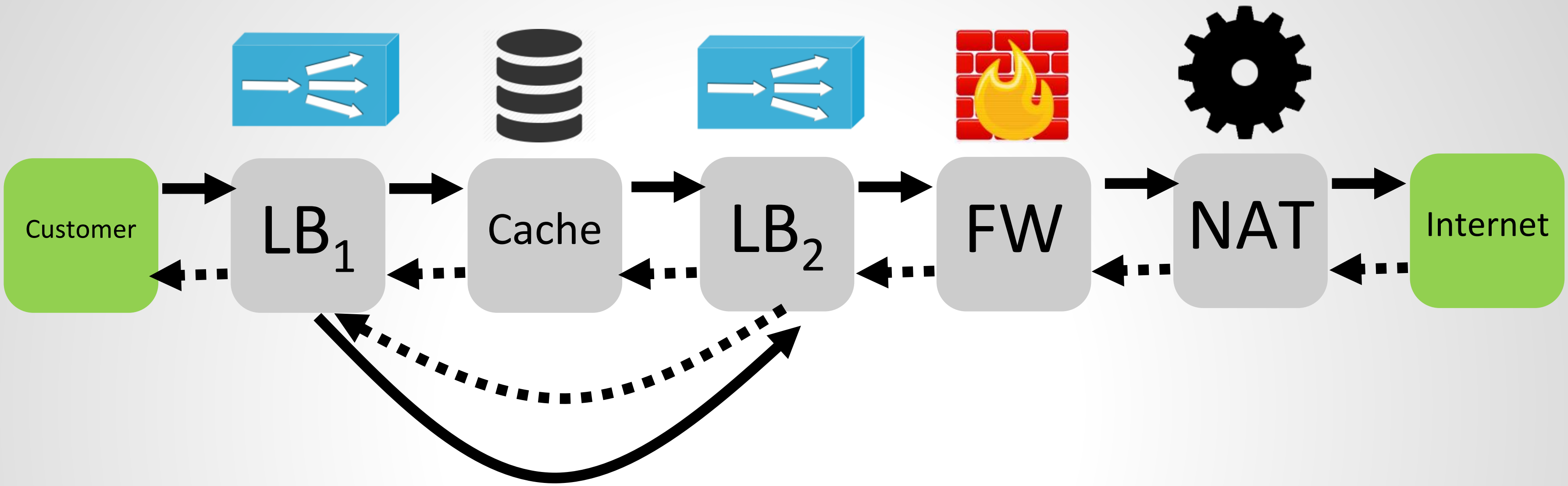
and different decompositions?



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

What about this one?!

IETF Draft:

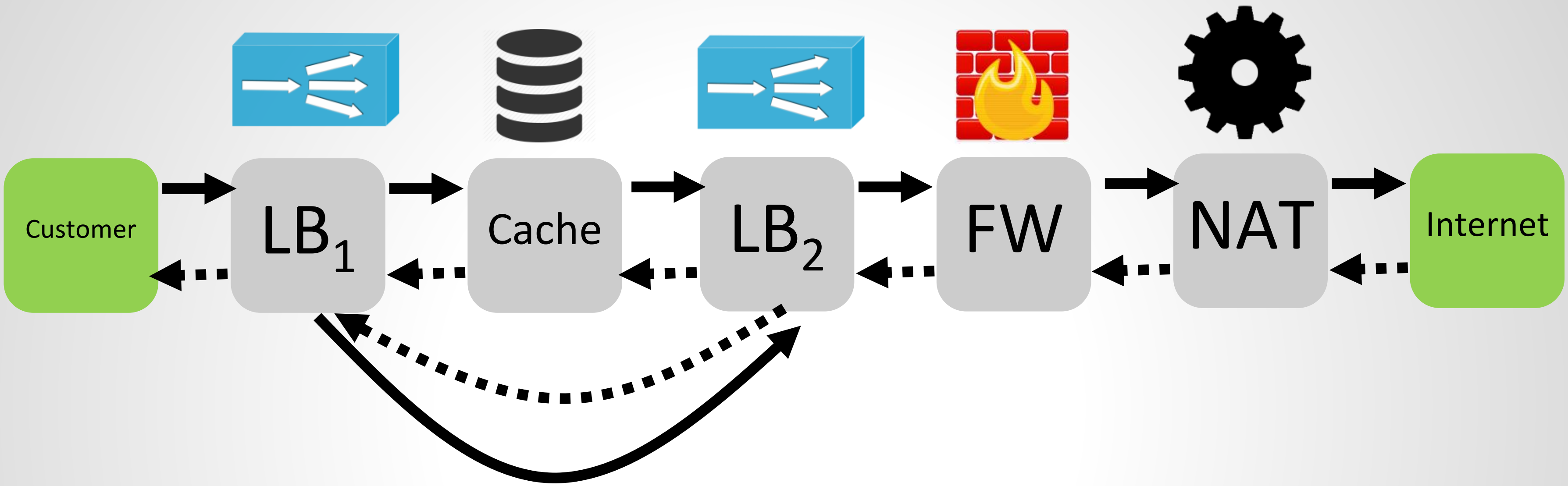


- ❑ Service chain for mobile operators
- ❑ Load-balancers are used to route (parts of) the traffic through cache

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

What about this one?!

IETF Draft:



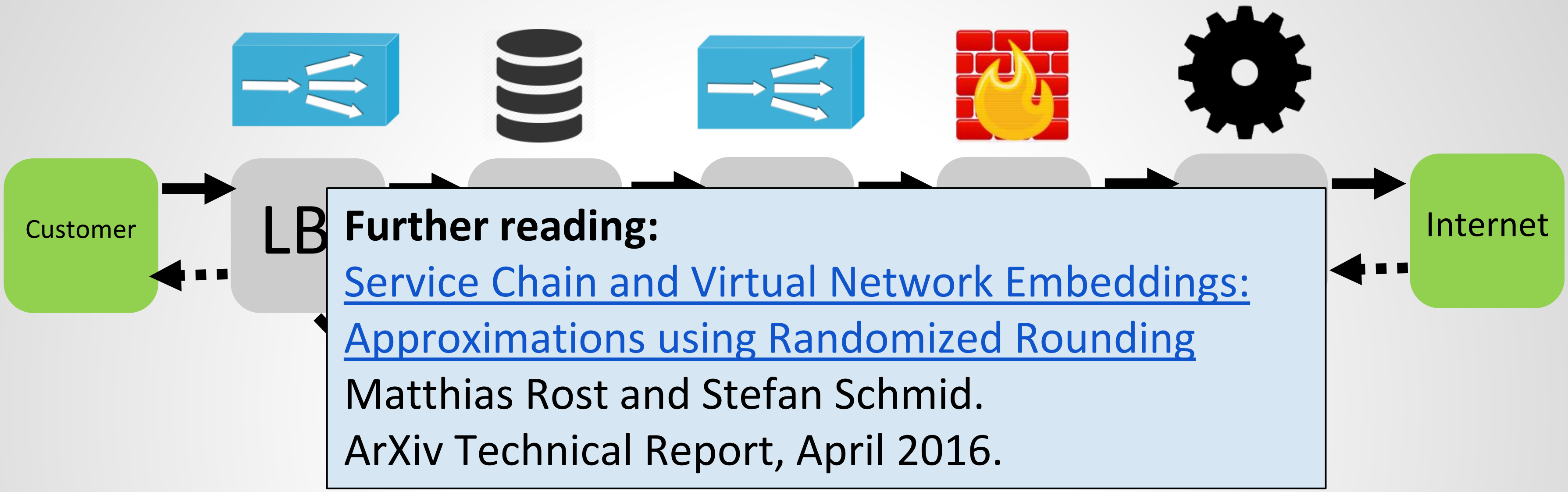
- Service chain for mobile operators
- Load-balancers are used to route (parts of) the traffic through cache

Has loops: the standard approach **no longer works**! There are first insights on advanced techniques for such graphs, but it's an **open question** how far they can be pushed.

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

What about this one?!

IETF Draft:



Further reading:
Service Chain and Virtual Network Embeddings: Approximations using Randomized Rounding
Matthias Rost and Stefan Schmid.
ArXiv Technical Report, April 2016.

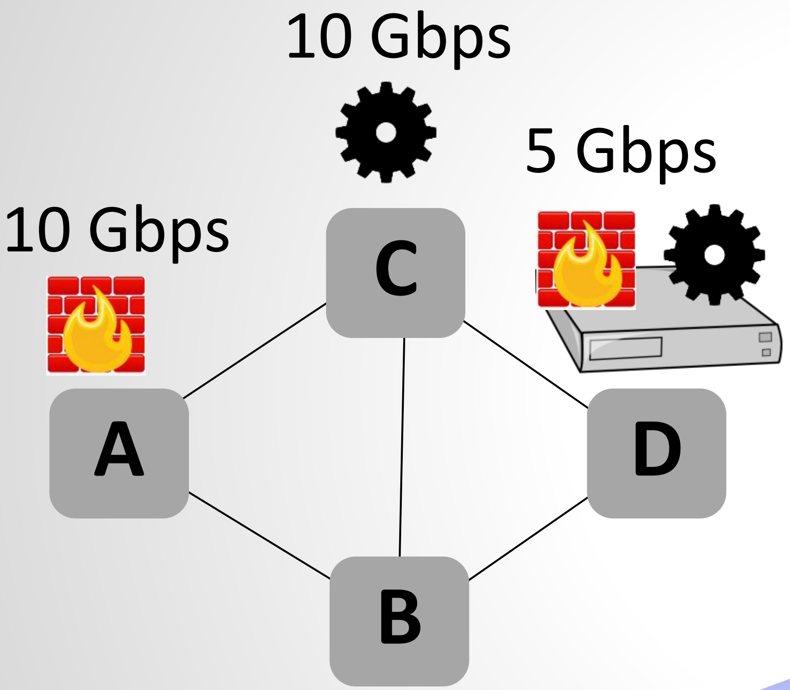
- Service chain for mobile operators
- Load-balancers are used to route (parts of) the traffic through cache

Has loops: the standard approach **no longer works**! There are first insights on advanced techniques for such graphs, but it’s an **open question** how far they can be pushed.

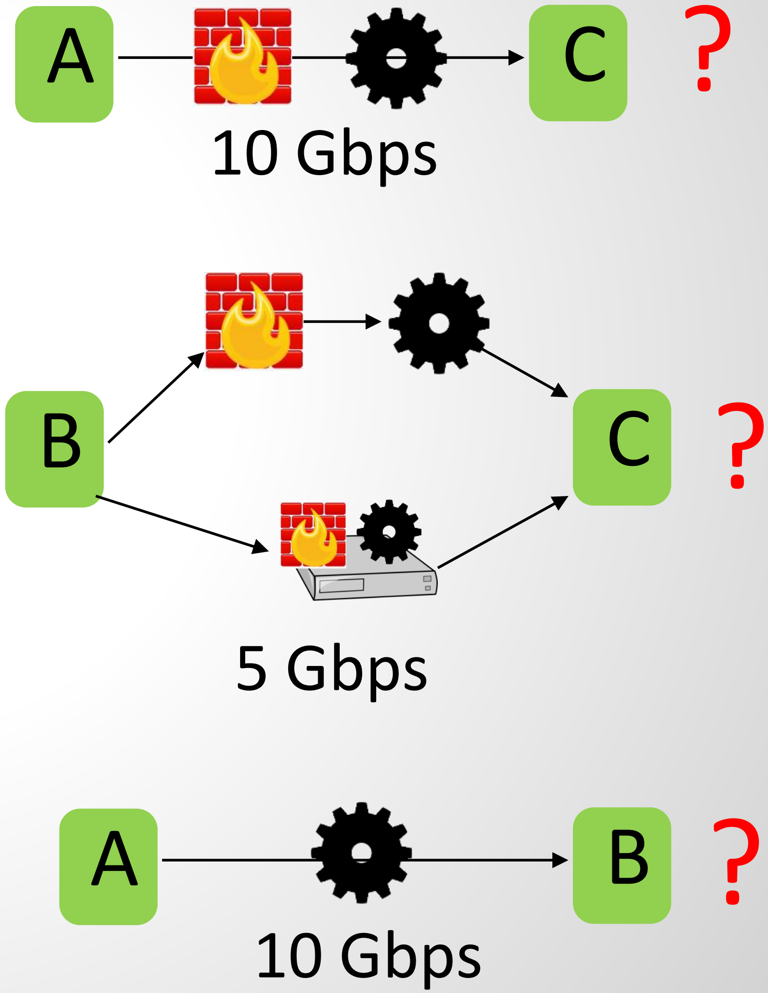
Credits: https://tools.ietf.org/html/draft-ietf-sfc-use-case-mobility-06

Example: admission control and embedding

Substrate:



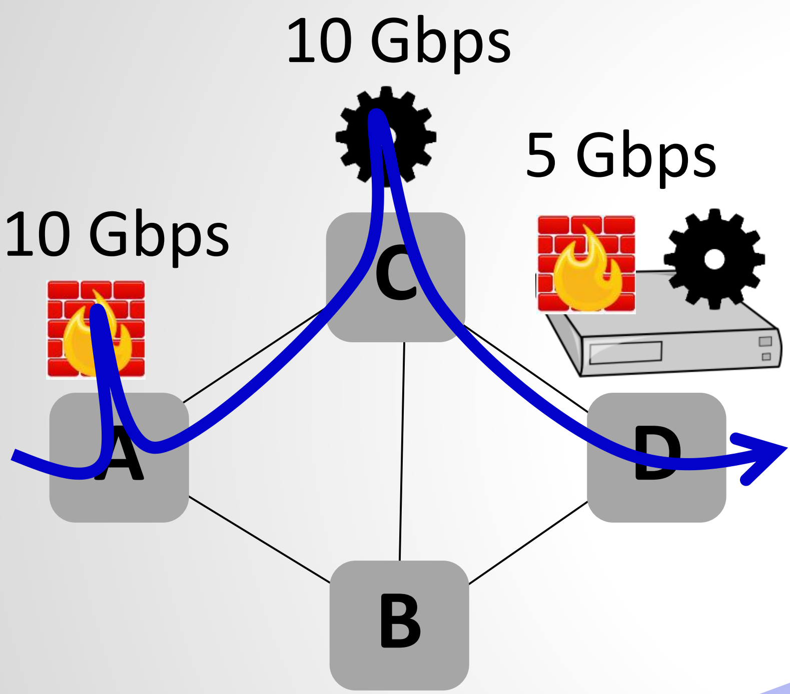
Requests:



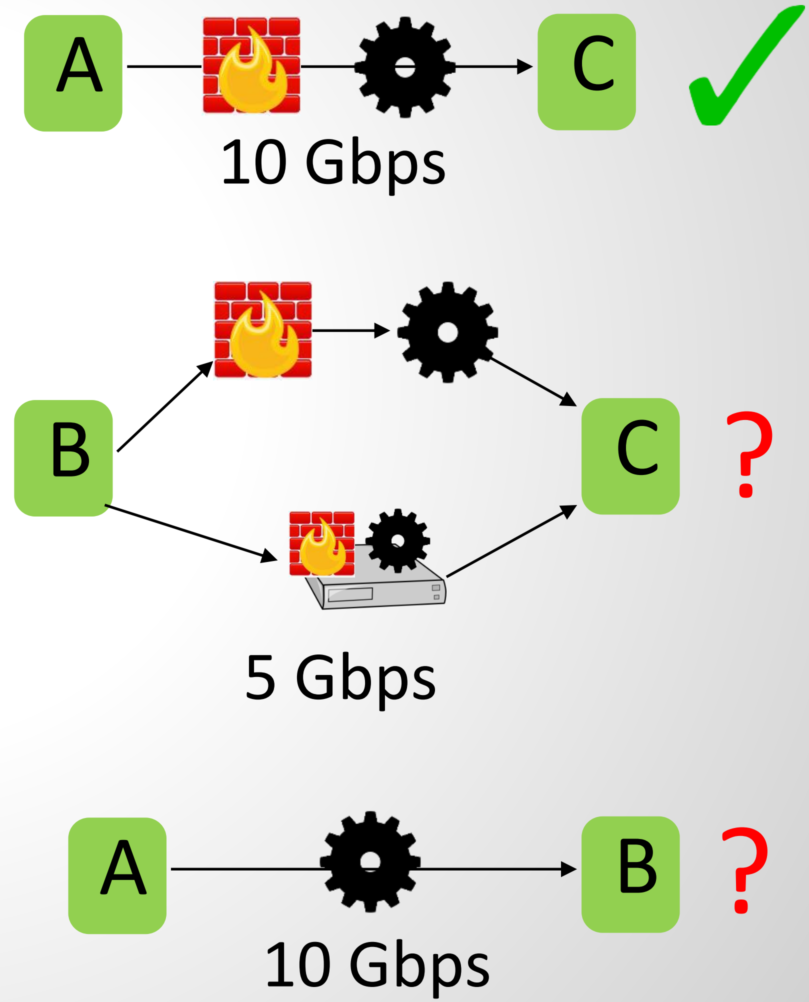
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



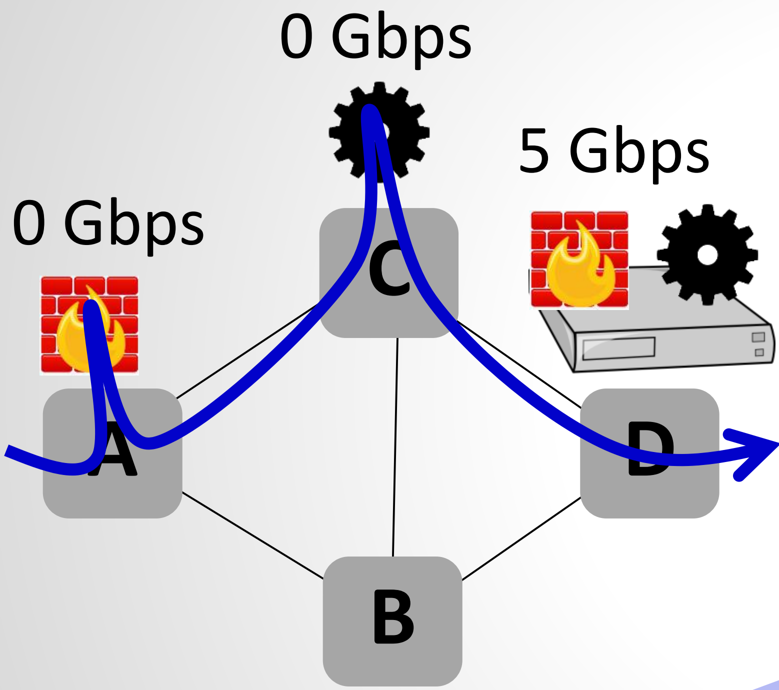
Requests:



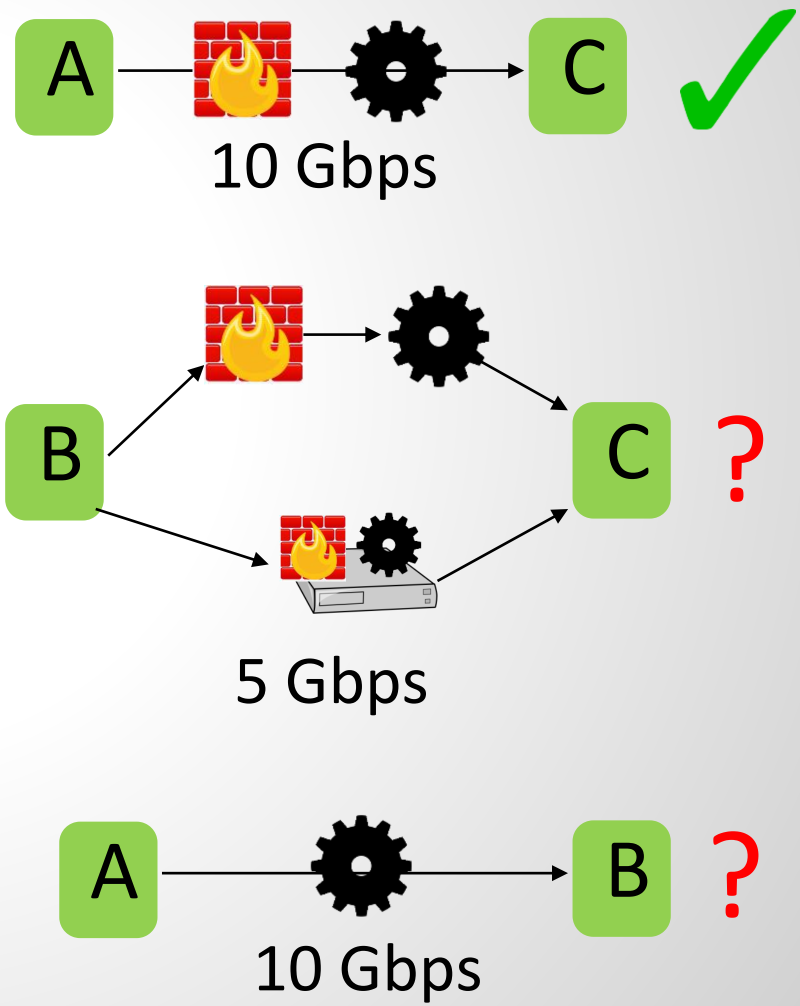
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



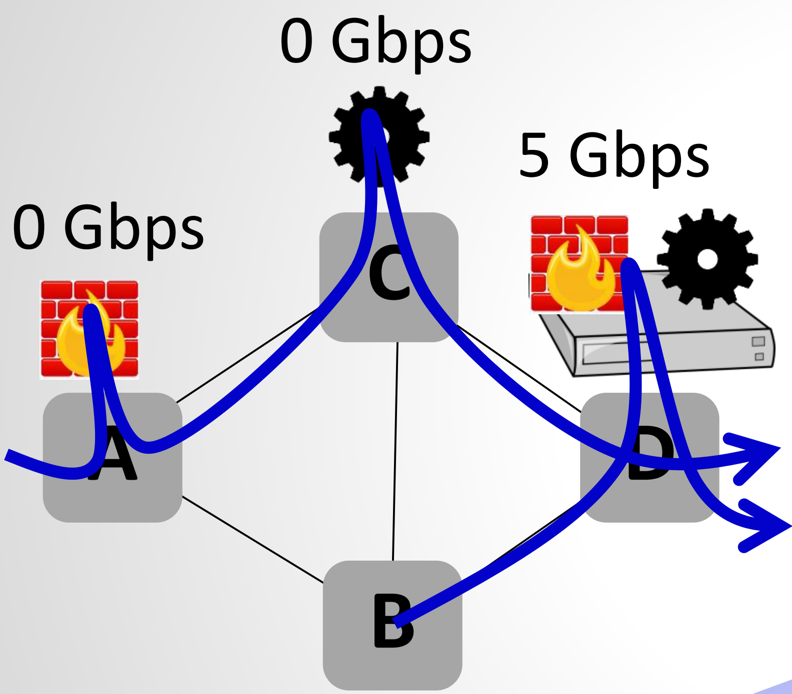
Requests:



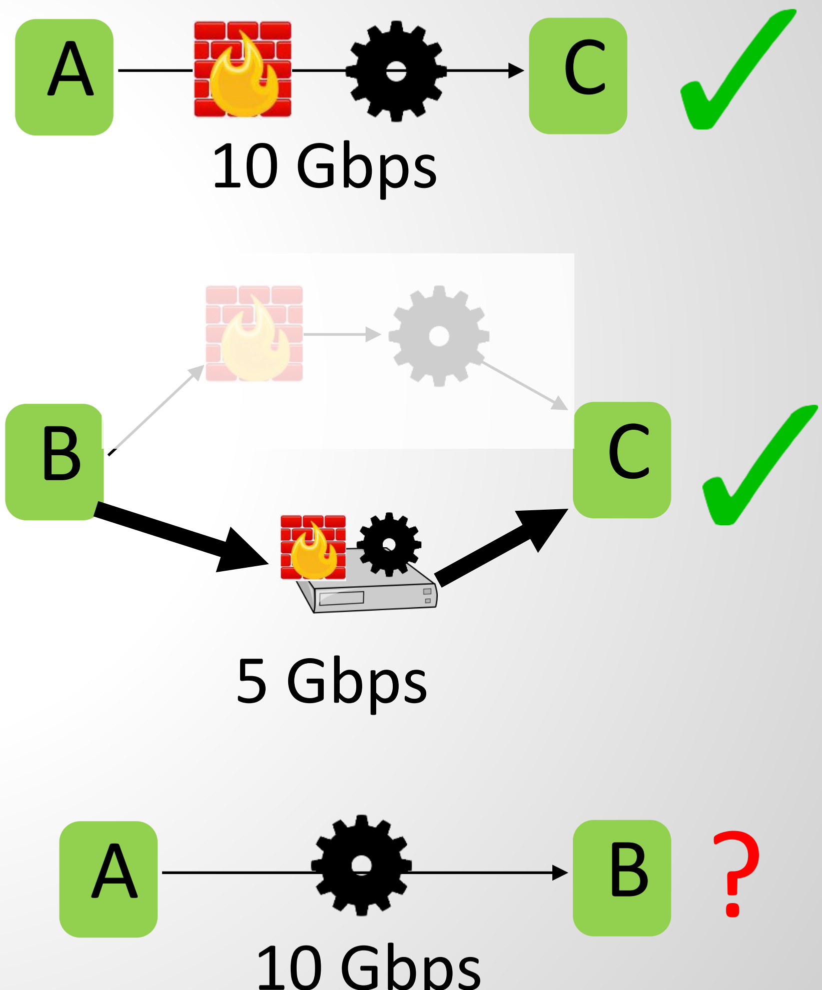
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



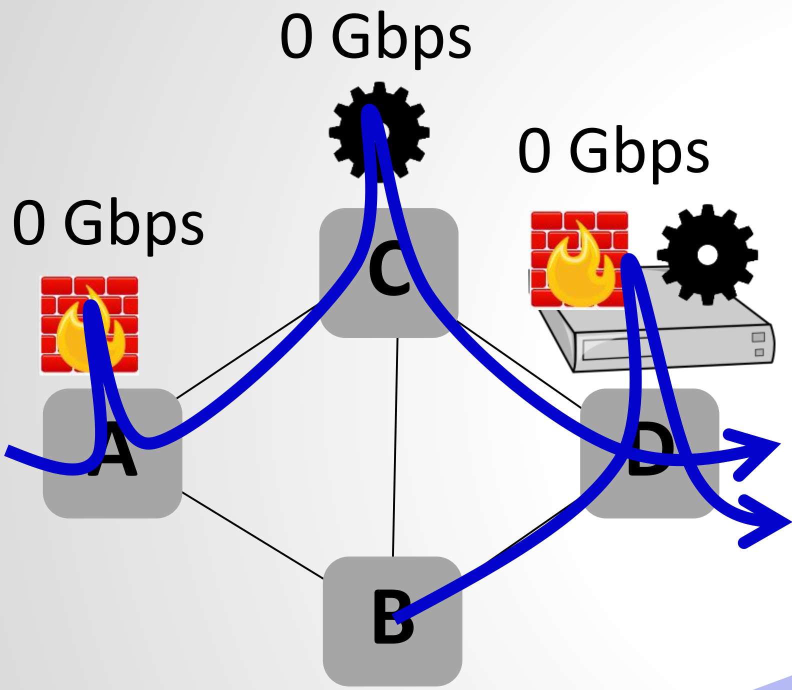
Requests:



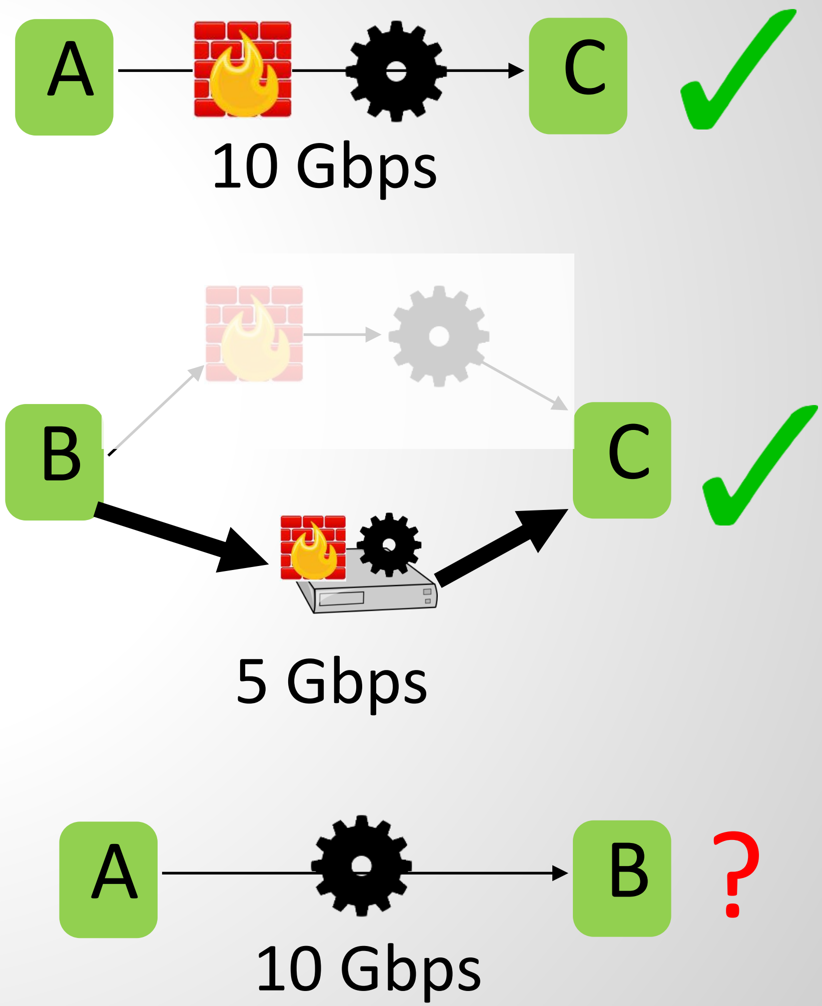
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



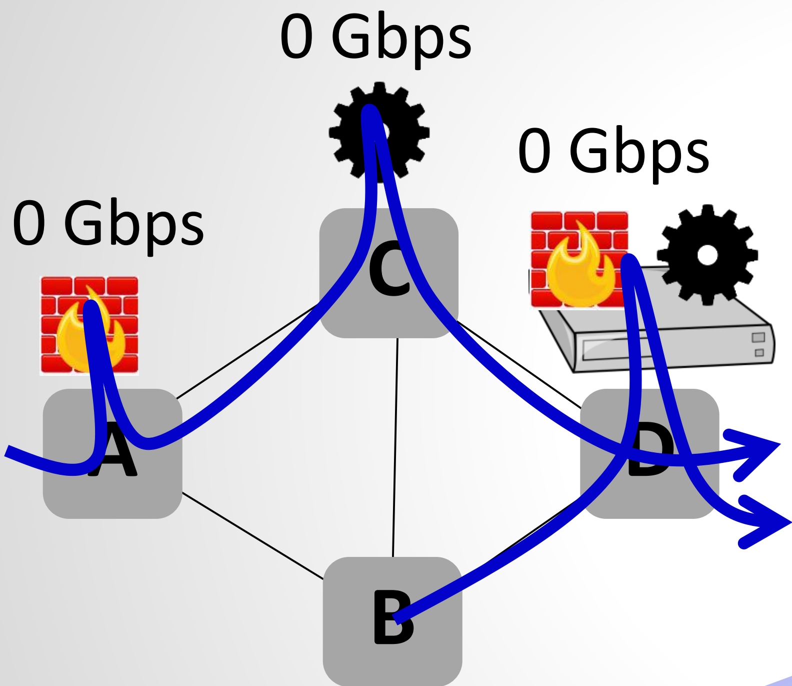
Requests:



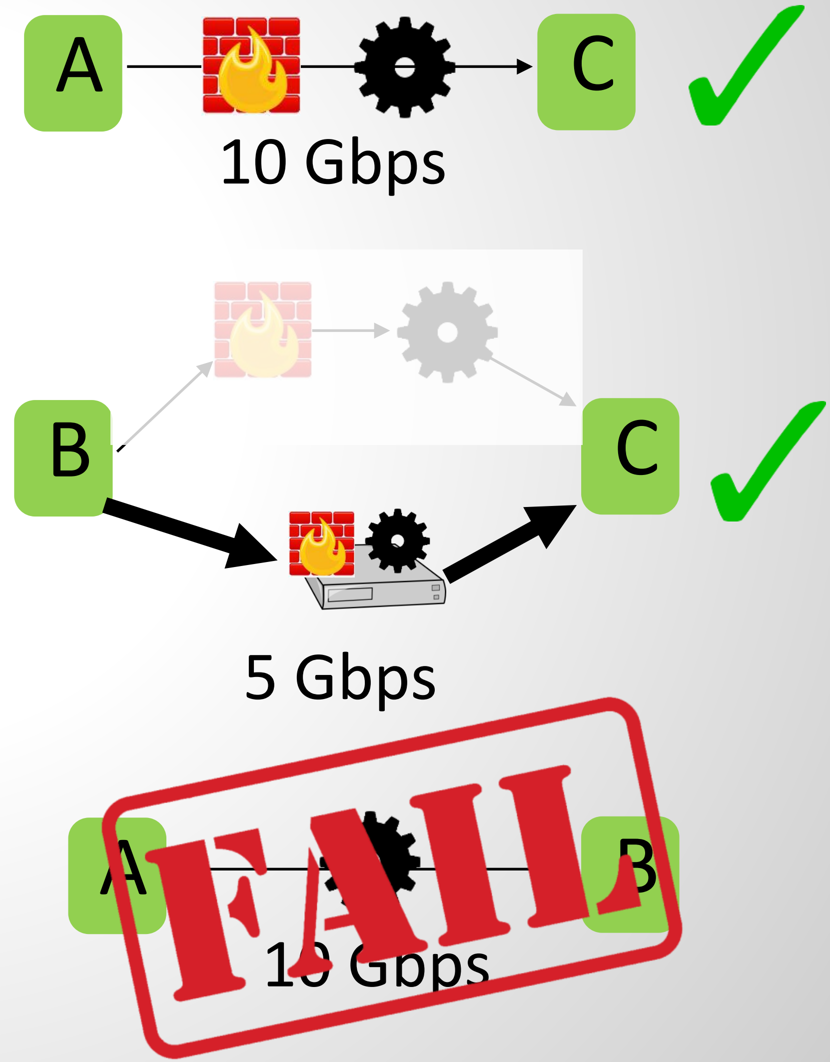
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



Requests:



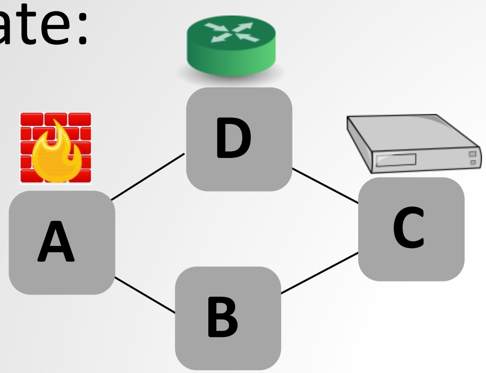
Which ones can be admitted and embedded?

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

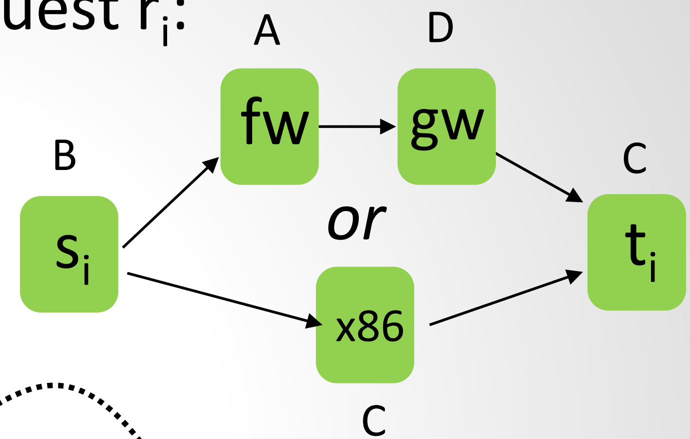
Chains, alternative chains, but even trees. Trick: **reduction to flow problem** using **product graphs**.

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

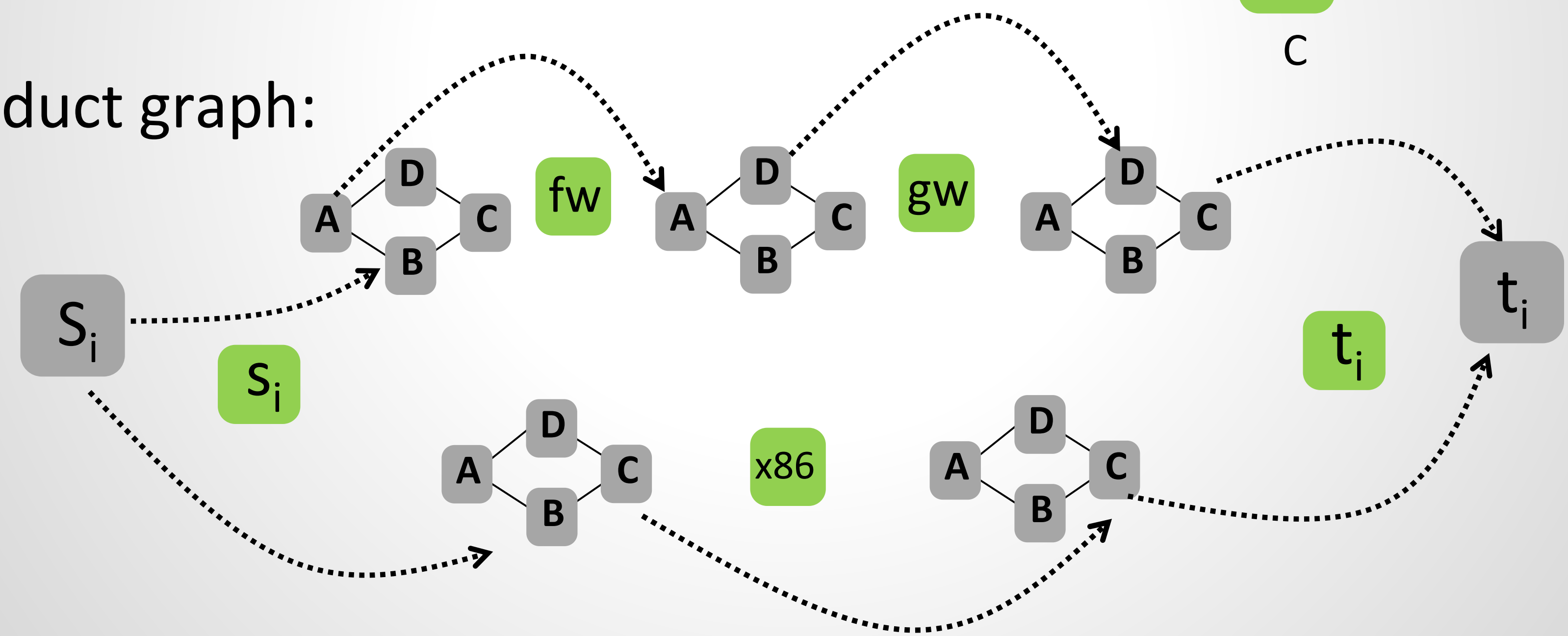
Substrate:



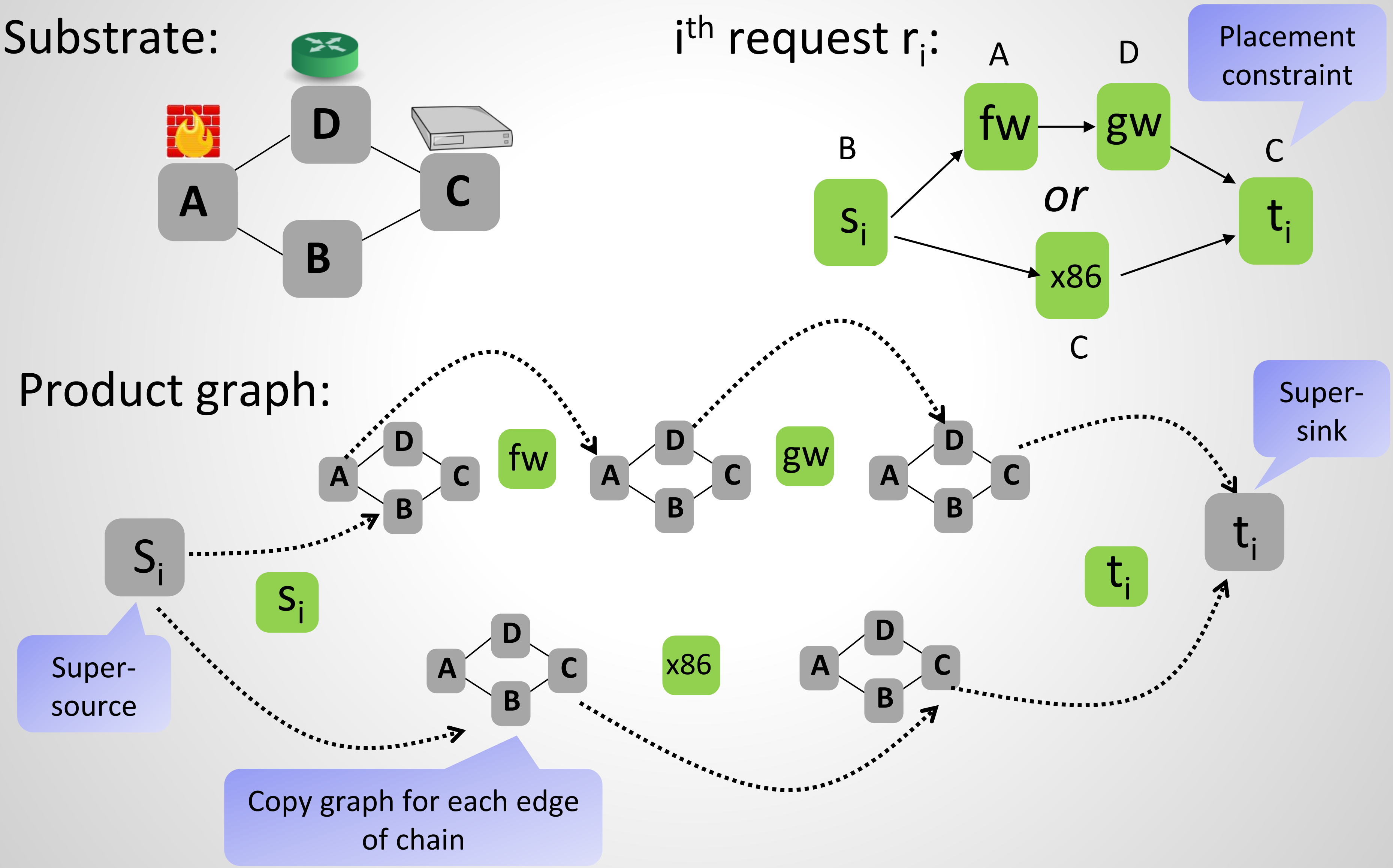
i^{th} request r_i:



Product graph:

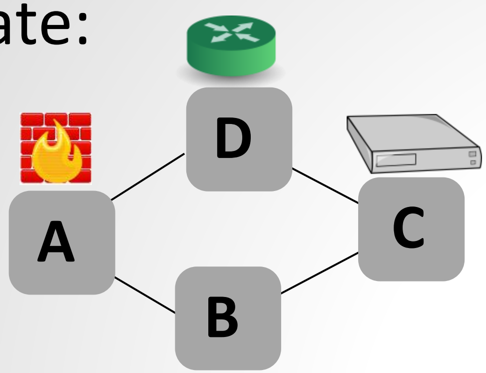


Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for "Fairly Simple" Requests*!

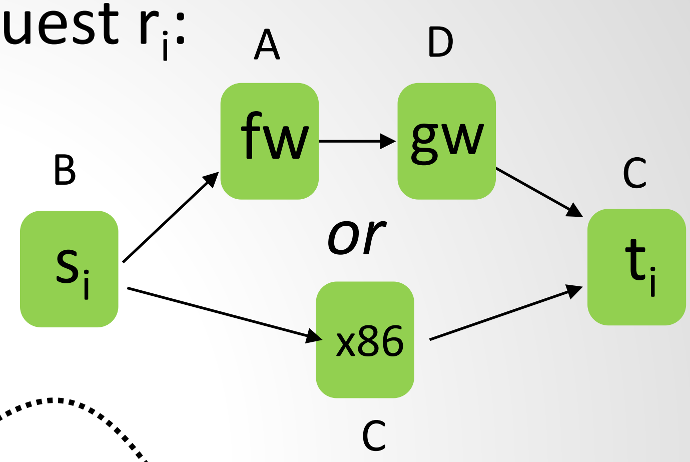


Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for "Fairly Simple" Requests*!

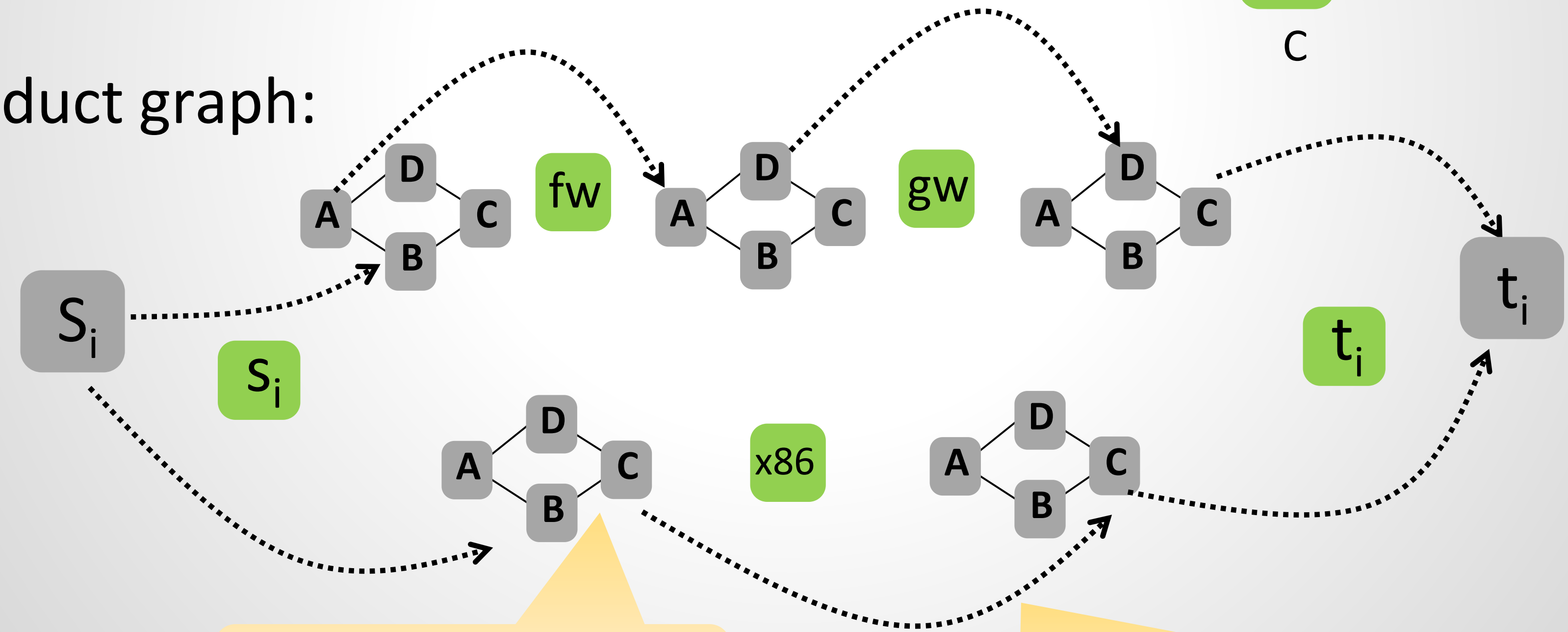
Substrate:



i^{th} request r_i:



Product graph:

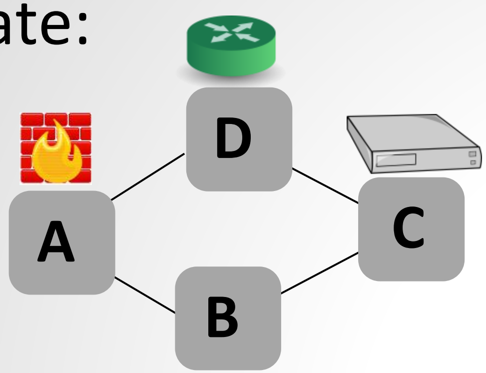


Routing edge: graph edge on same layer

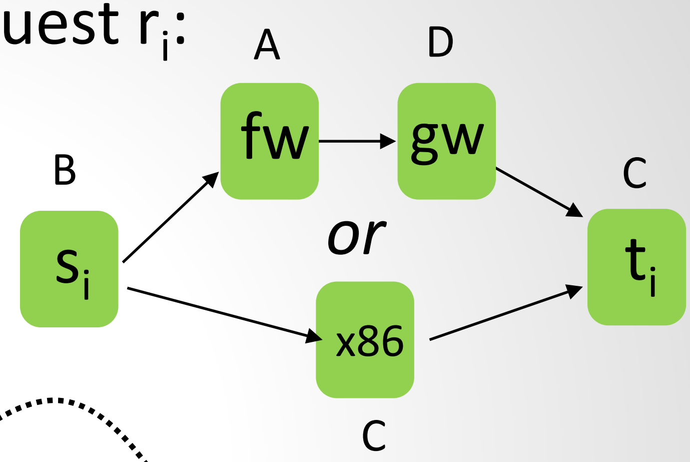
Processing edge: processing happens on C: connect C to C in next layer!

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

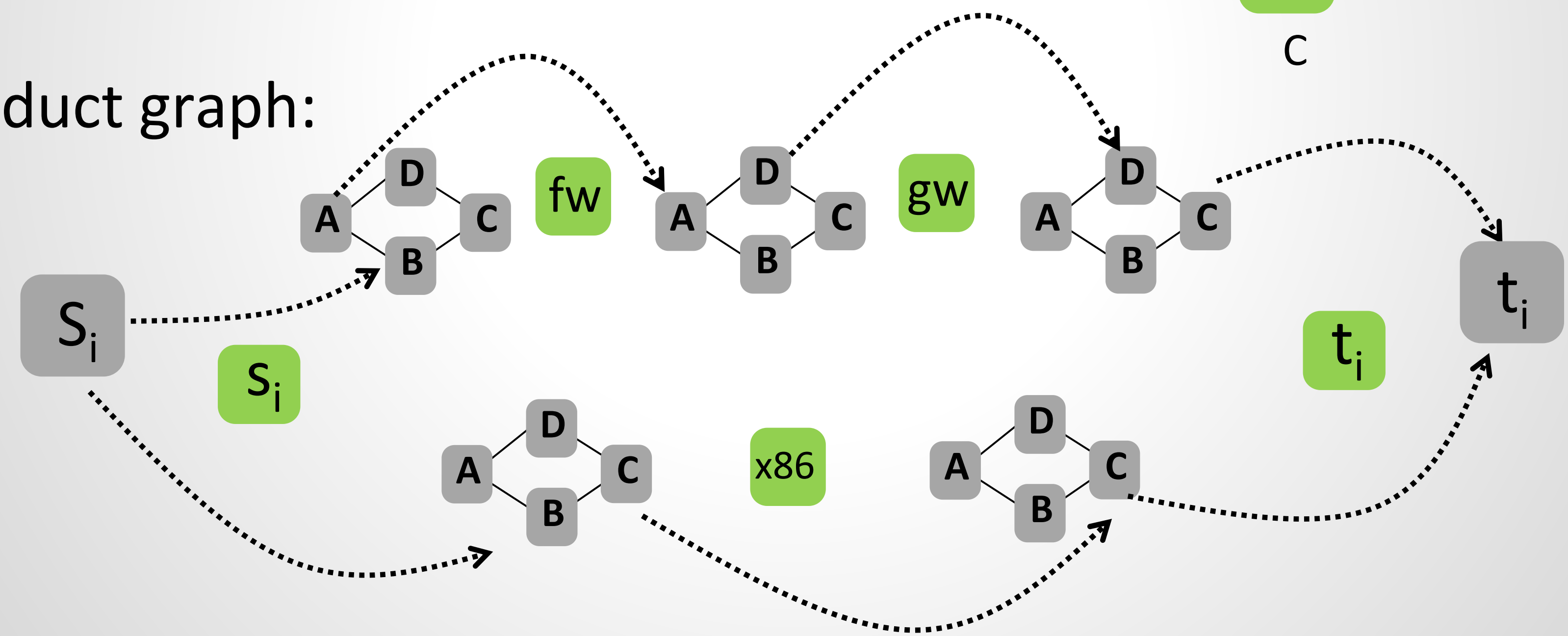
Substrate:



i^{th} request r_i:

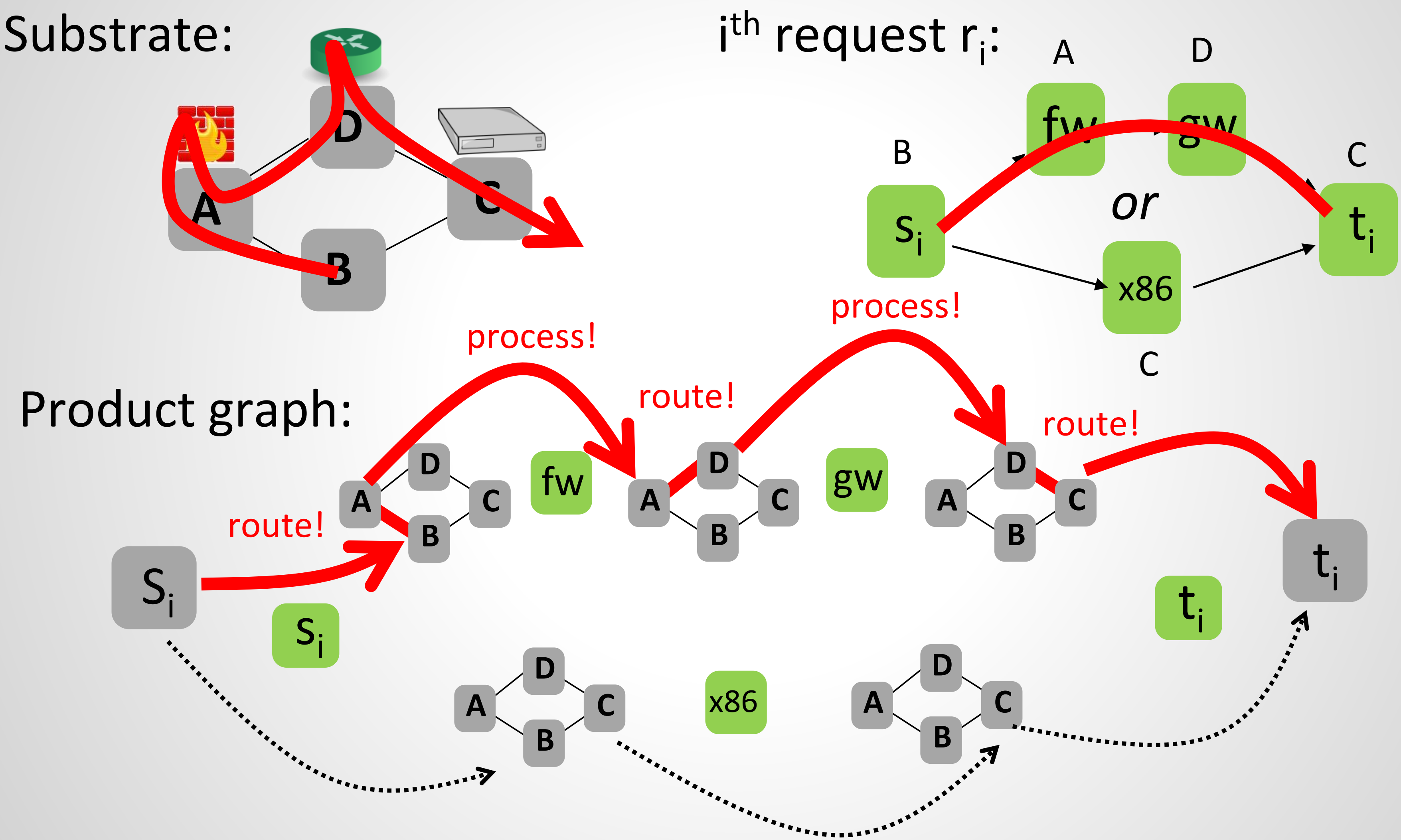


Product graph:



Any valid (s_i,t_i) path presents a valid realization of the request r_i!

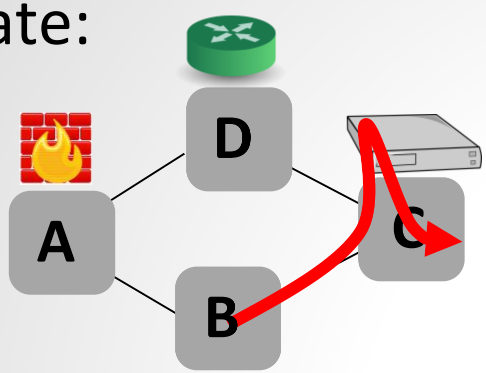
Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for "Fairly Simple" Requests*!



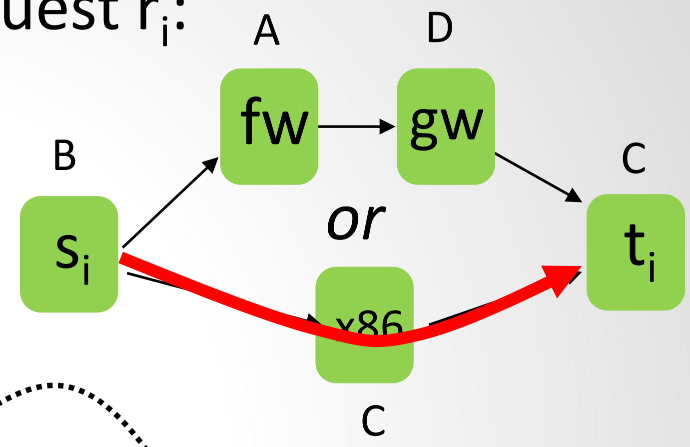
Any valid (s_i,t_i) path presents a valid realization of the request r_i!

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

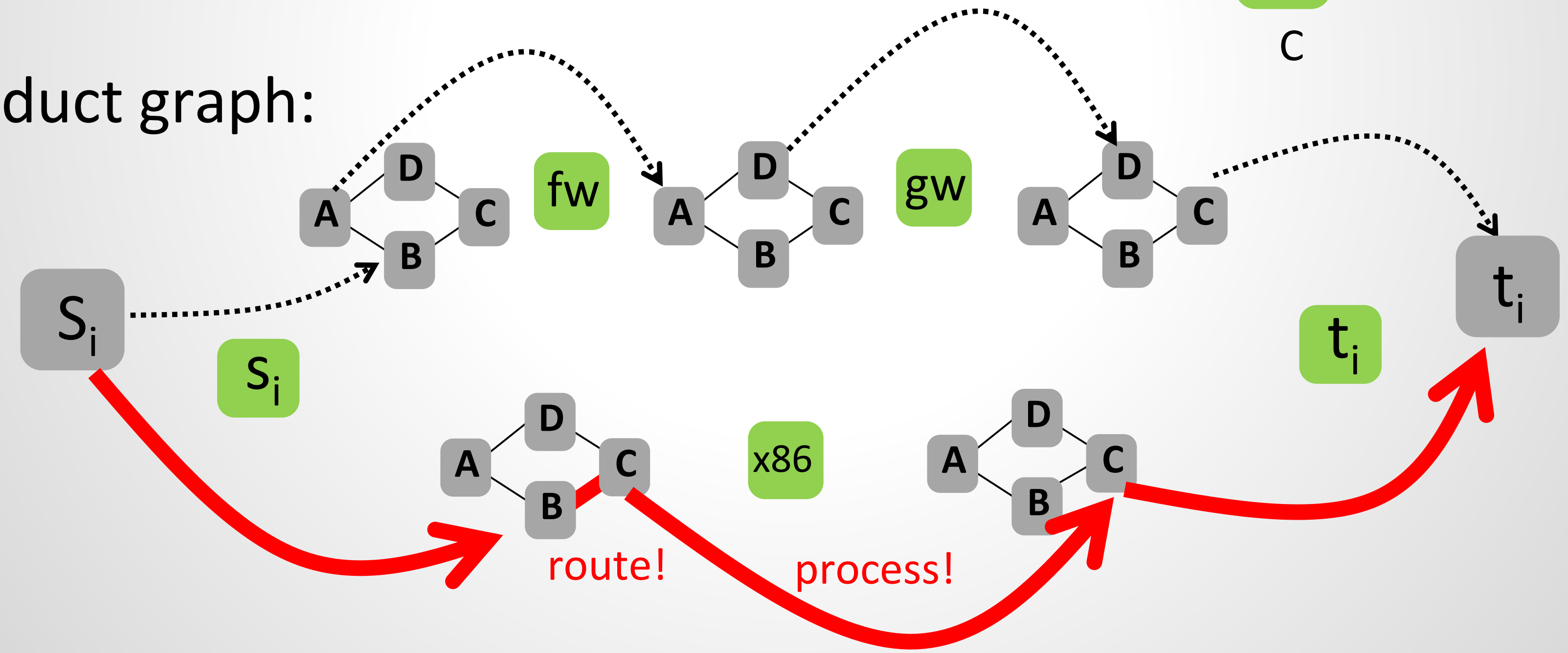
Substrate:



i^{th} request r_i:

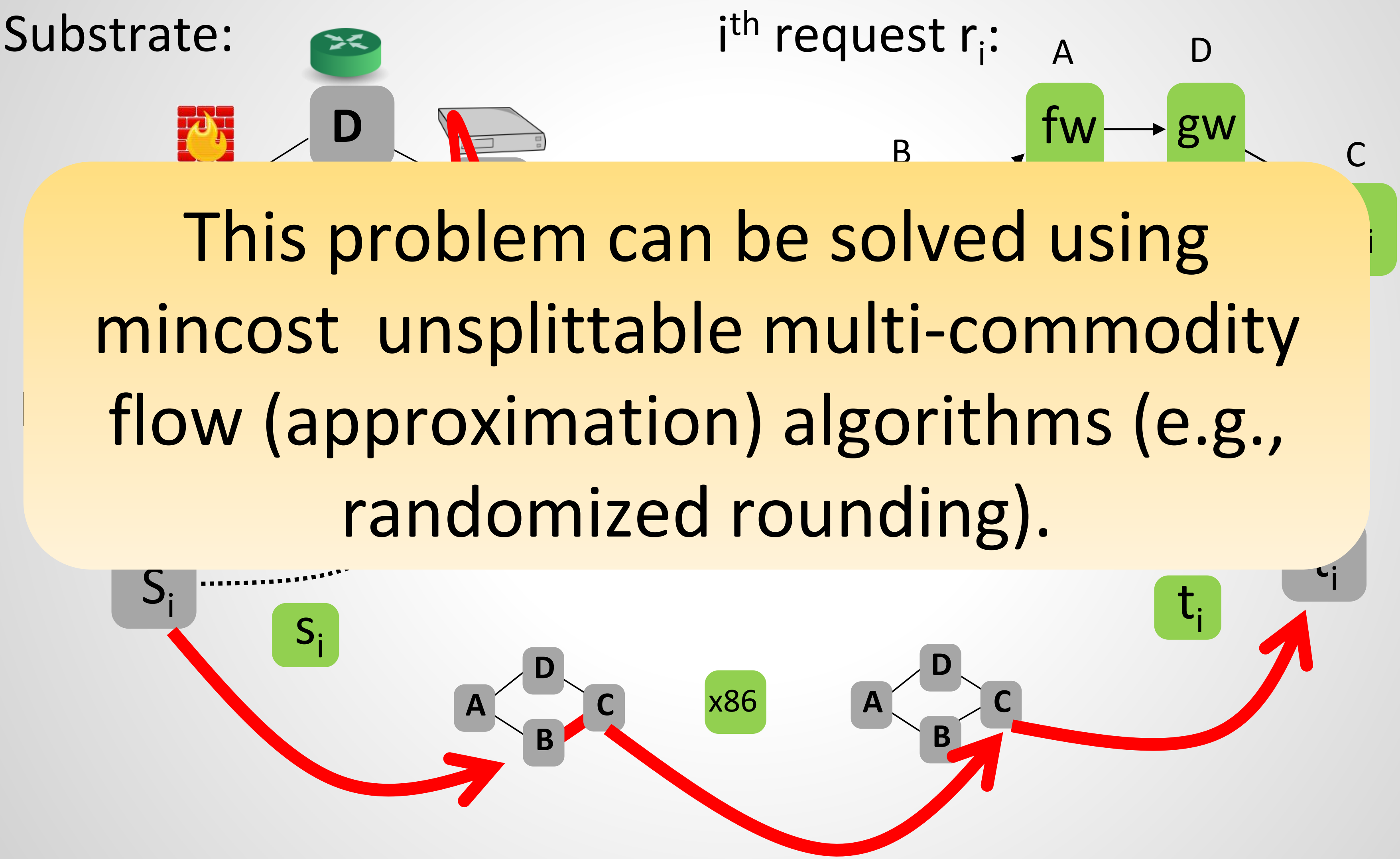


Product graph:



Any valid (s_i,t_i) path presents a valid realization of the request r_i!

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



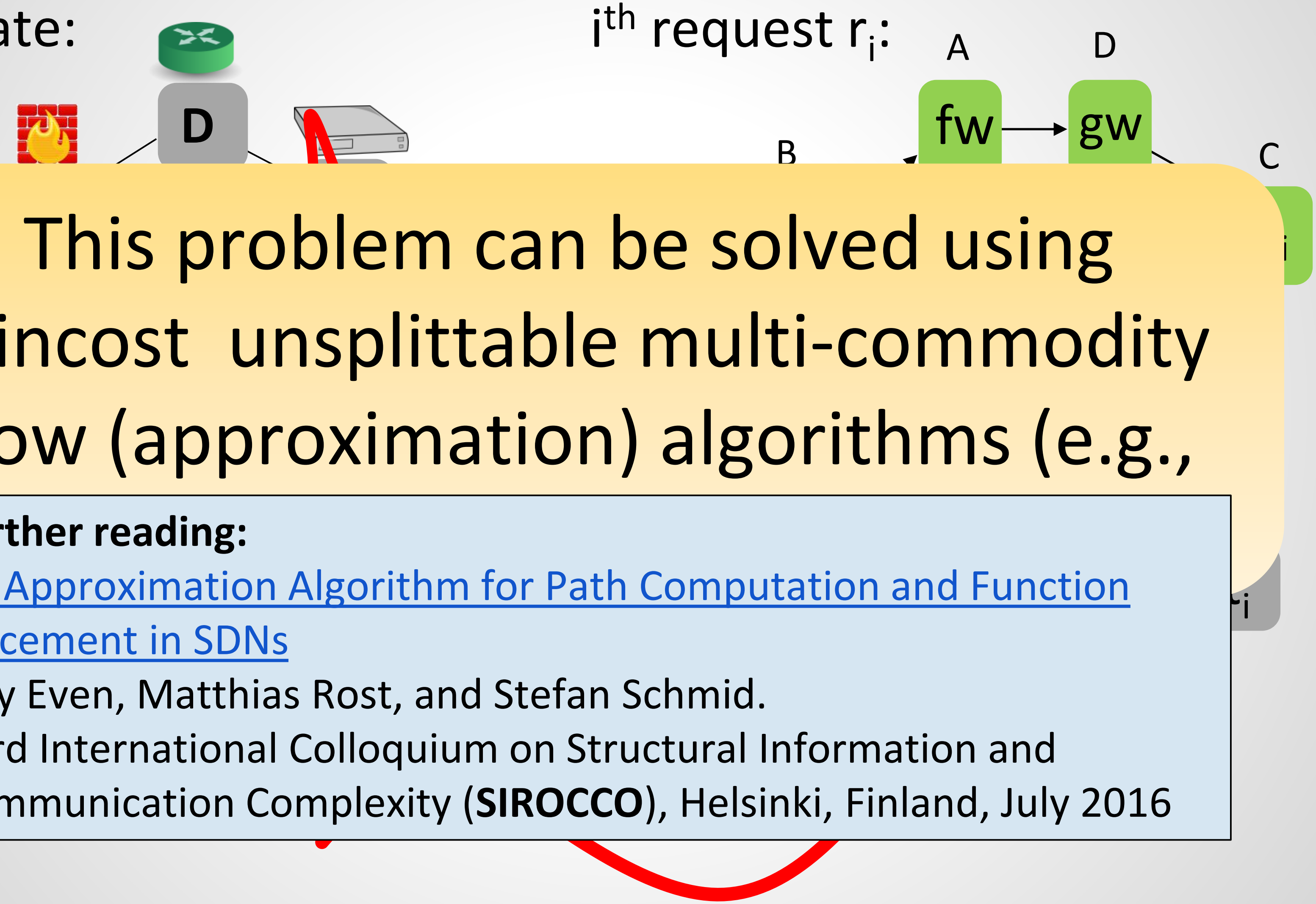
This problem can be solved using mincost unsplittable multi-commodity flow (approximation) algorithms (e.g., randomized rounding).

Any valid (s_i,t_i) path presents a valid realization 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:

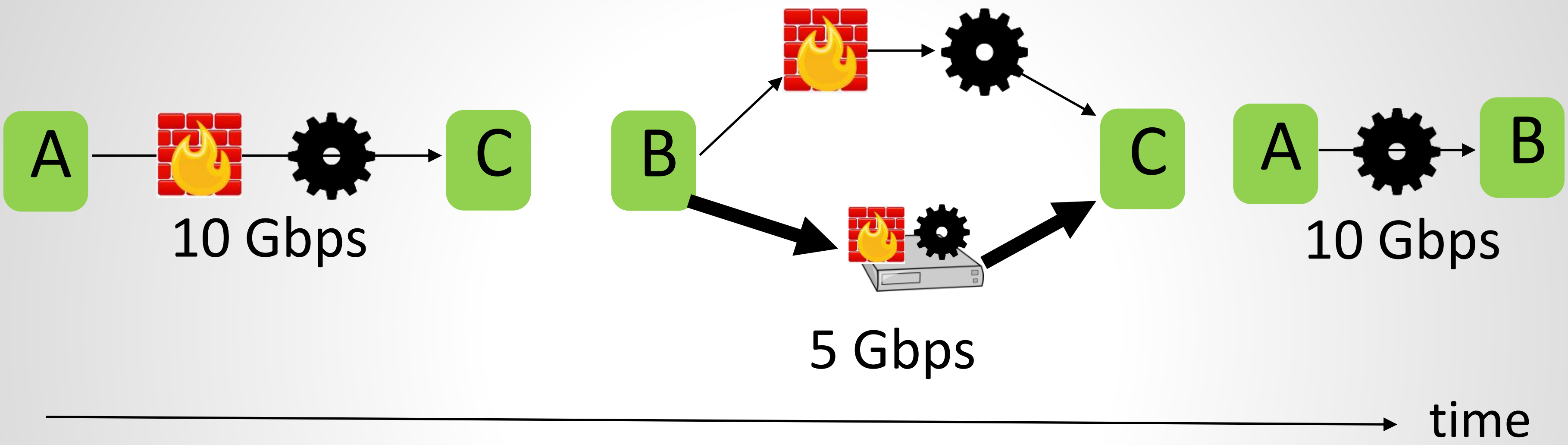


This problem can be solved using mincost unsplittable multi-commodity flow (approximation) algorithms (e.g.,

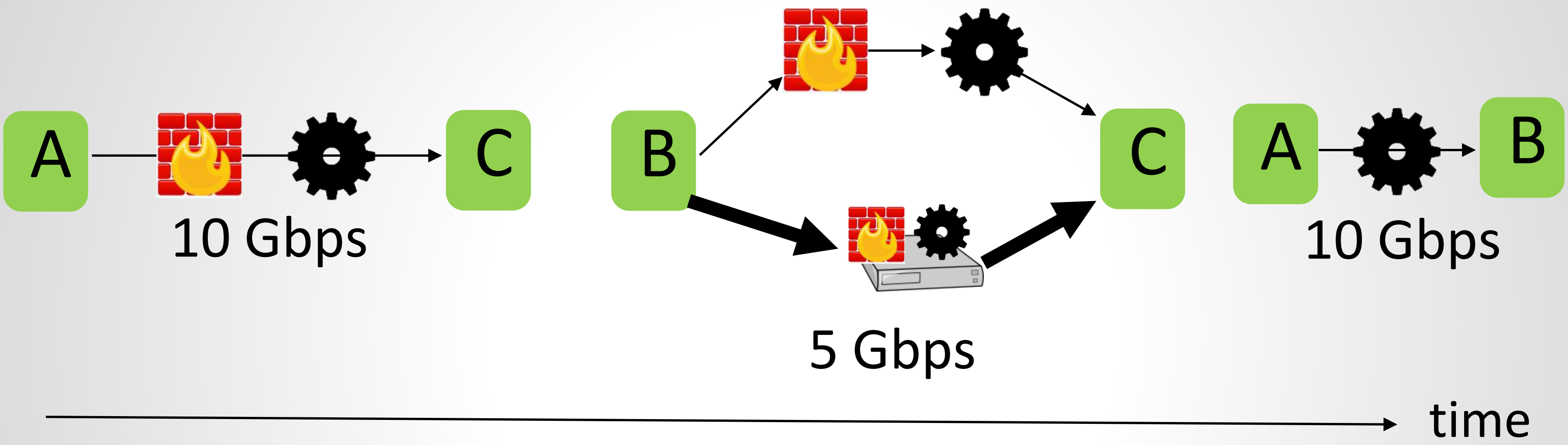
Further reading:
An Approximation Algorithm for Path Computation and Function Placement in SDNs
Guy Even, Matthias Rost, and Stefan Schmid.
23rd International Colloquium on Structural Information and Communication Complexity (**SIROCCO**), Helsinki, Finland, July 2016

Any valid (s_i,t_i) path presents a valid realization of the request r_i!

What if requests arrive over time?
Can we admit and embed requests efficiently?



Good News 2: Yes, given offline embedding algorithm, *can do it online, over time, as well!*



Online primal dual-framework Buchbinder&Naor:

Even without knowing anything about future requests, we can approximate an optimal offline solution that knows the future.

The Buchbinder-Naor Approach

Primal and Dual

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$ $Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$ $X, Z_j \geq \mathbf{0}$	$\max B_j^T \cdot Y_j \;\; s.t.$ $A_j \cdot Y_j \leq C$ $D_j \cdot Y_j \leq \mathbf{1}$ $Y_j \geq \mathbf{0}$

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

The Buchbinder-Naor Approach

Primal and Dual

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$ $Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$ $X, Z_j \geq \mathbf{0}$	$\max B_j^T \cdot Y_j \;\; s.t.$ $A_j \cdot Y_j \leq C$ $D_j \cdot Y_j \leq \mathbf{1}$ $Y_j \geq \mathbf{0}$

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure) ← Offline embedding!
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

The Buchbinder-Naor Approach

Primal and Dual

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \ \ s.t.$ $Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$ $X, Z_j \geq \mathbf{0}$	$\max B_j^T \cdot Y_j \ \ s.t.$ $A_j \cdot Y_j \leq C$ $D_j \cdot Y_j \leq \mathbf{1}$ $Y_j \geq \mathbf{0}$

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

Embedding cost vs profit?

The Buchbinder-Naor Approach

Primal and Dual

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$ $Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$ $X, Z_j \geq \mathbf{0}$	$\max B_j^T \cdot Y_j \;\; s.t.$ $A_j \cdot Y_j \leq C$ $D_j \cdot Y_j \leq \mathbf{1}$ $Y_j \geq \mathbf{0}$

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Fairly well-understood!

Some caveats! ☺

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

Embedding cost vs profit?

The Buchbinder-Naor Approach

Primal and Dual

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$ $Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$ $X, Z_j \geq \mathbf{0}$	$\max B_j^T \cdot Y_j \;\; s.t.$ $A_j \cdot Y_j \leq C$ $D_j \cdot Y_j \leq \mathbf{1}$ $Y_j \geq \mathbf{0}$

Fig. 1: (

Fairly well-understood!

Some caveats! ☺

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

Algorithm

Algorithm

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do
   $$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$
   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

← Embedding cost vs profit?

What about using randomized rounding?

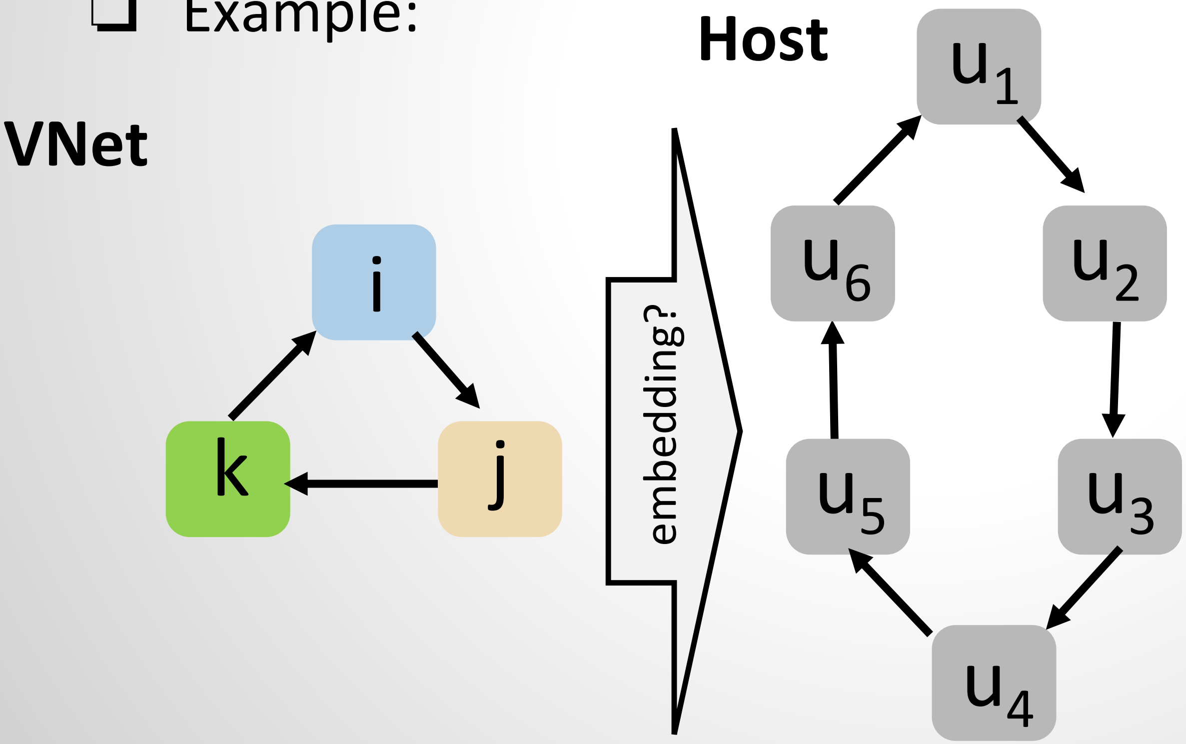
- Problem 1: relaxed solutions may not be very meaningful
  - see example for splittable flows before
- Problem 2: also for unsplittable flows, if using a standard Multi-Commodity Flow (MCF) formulation of VNEP, the integrality gap can be huge
  - Tree-like VNets are still ok
  - VNets with cycles: randomized rounding not applicable, since problem not decomposable

> The linear solutions can be decomposed into convex combinations of valid mappings.

Non-Decomposability

- Relaxations of classic MCF formulation cannot be decomposed into convex combinations of valid mappings (so we need different formulations!)

- Example:

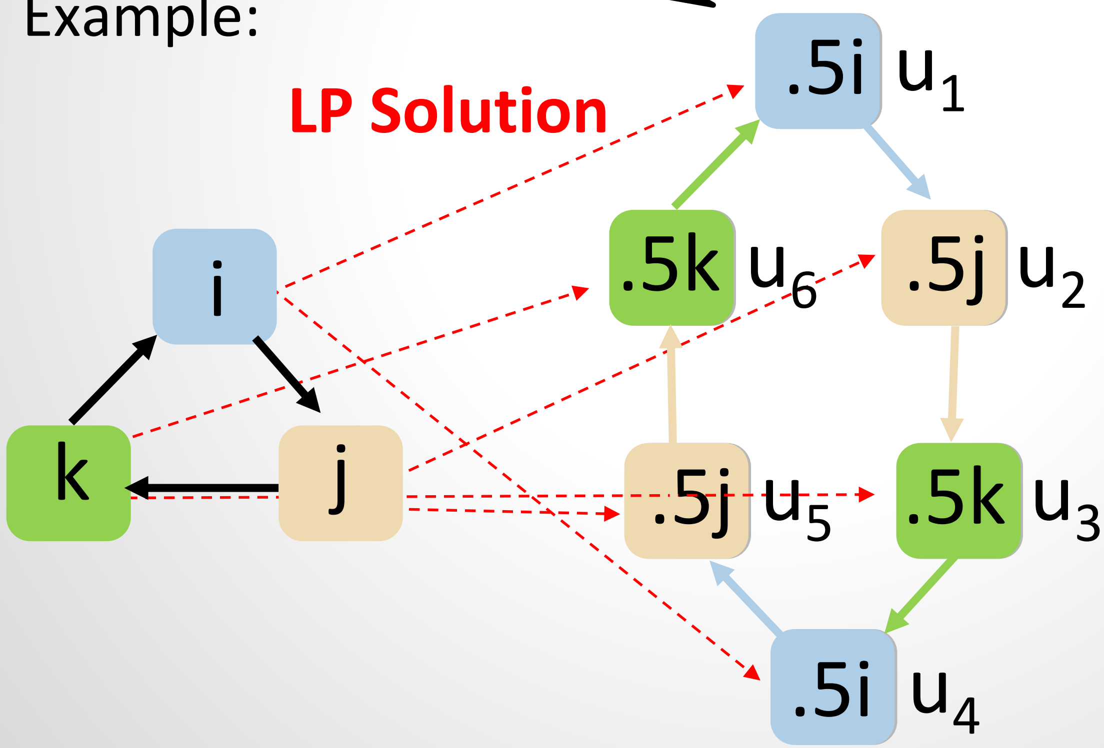


Non-Decomposability

- … rmulation cannot be decomposed … valid mappings (so we need

> Valid LP solution: virtual node mappings sum to 1 and each virtual node connects to its neighboring node **with half a unit of flow…**

- Example:



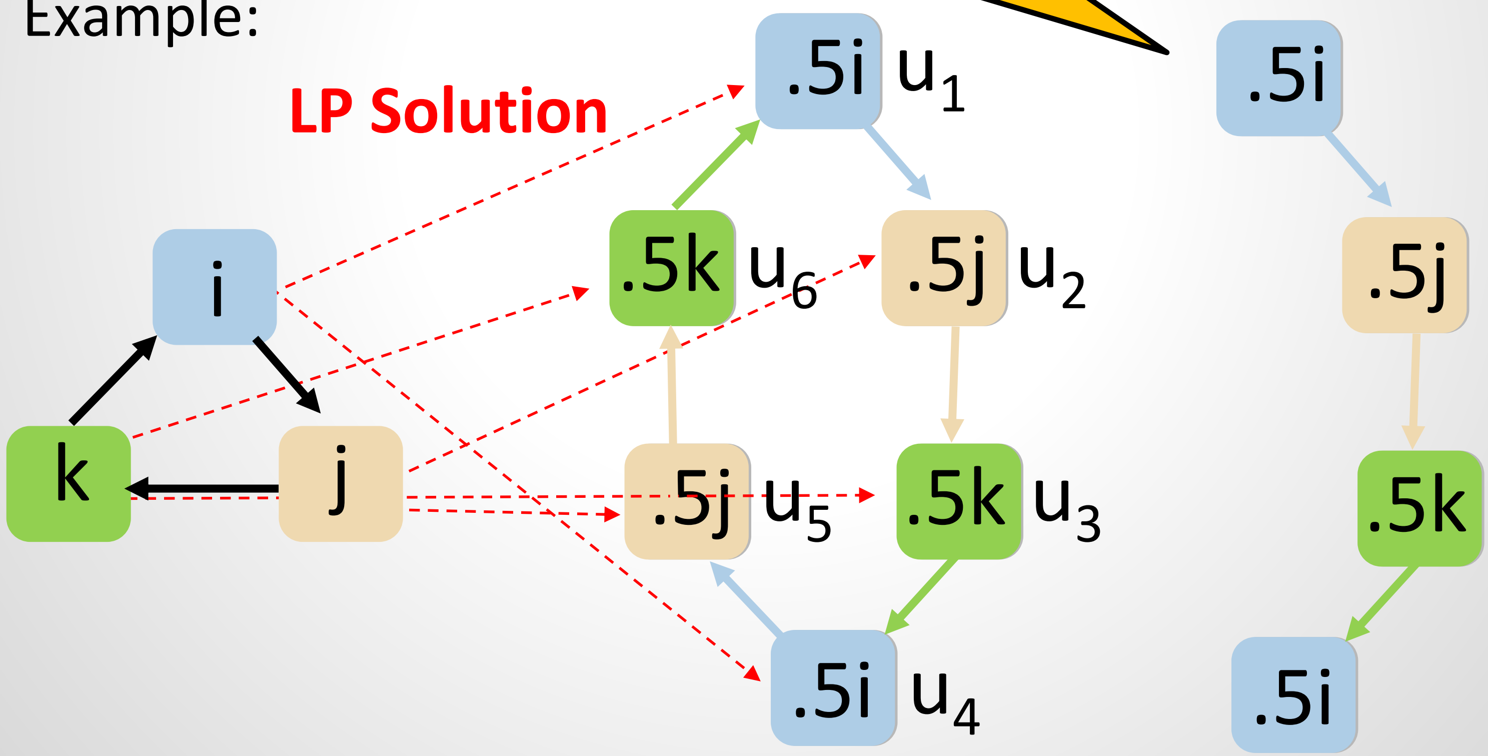
Impossible to decompose and extract **any single valid mapping**. **Intuition:** Node i is mapped to u_1 and the only neighboring node that hosts j is u_2, so i must be fully mapped on u_1 and j on u_2. Similarly, k must be mapped on u_3. But flow of virtual edge (k,i) leaving u_3 only leads to u_4, so **i must be mapped on both u_1 and u_4**. This is **impossible**, even if **capacities are infinite**.

ot be decomposed

s (so we need

Partial Decomposition

- Example:



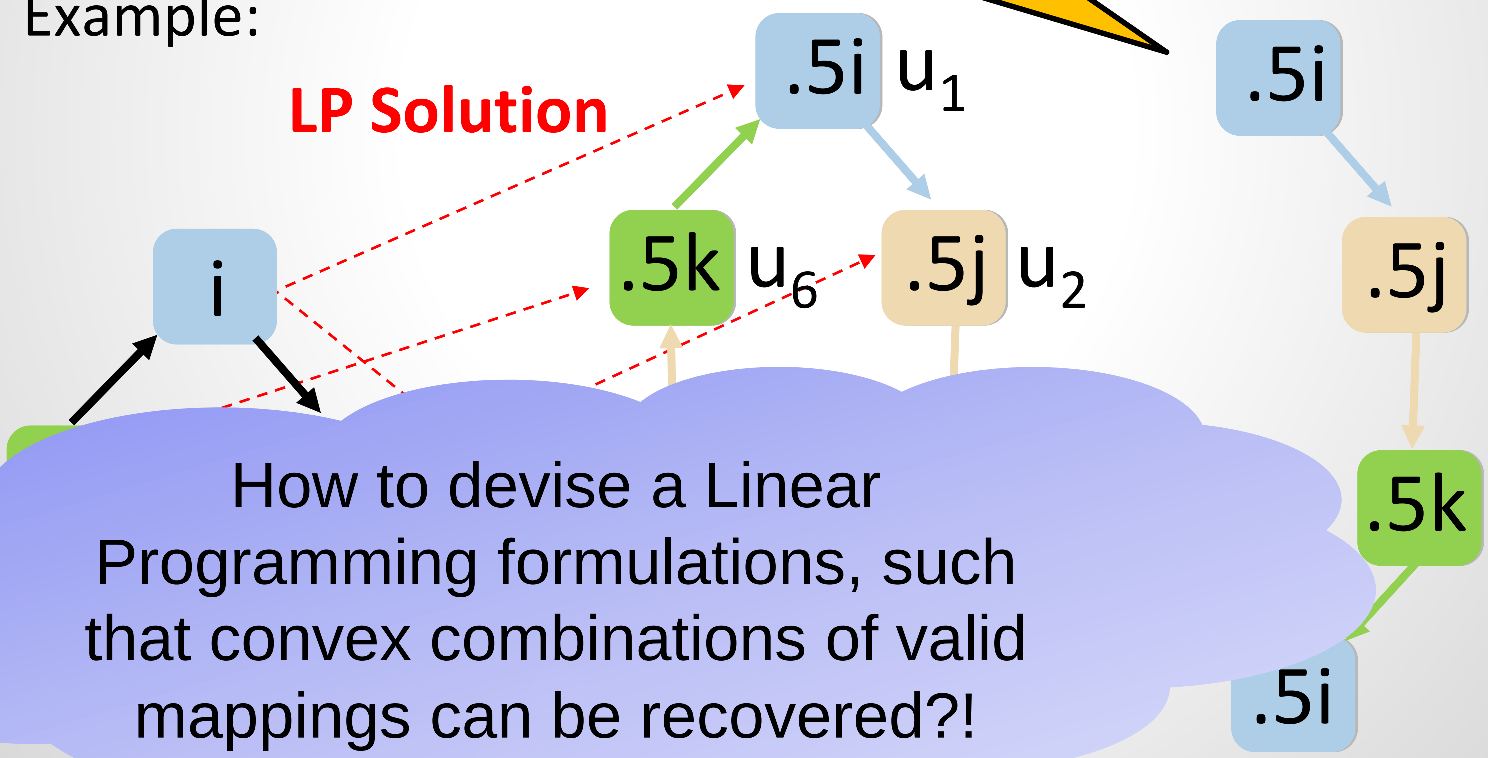
Impossible to decompose and extract **any single valid mapping**. **Intuition:** Node i is mapped to u_1 and the only neighboring node that hosts j is u_2, so i must be fully mapped on u_1 and j on u_2. Similarly, k must be mapped on u_3. But flow of virtual edge (k,i) leaving u_3 only leads to u_4, so **i must be mapped on both u_1 and u_4**. This is **impossible**, even if **capacities are infinite**.

not be decomposed

s (so we need

Partial Decomposition

- Example:



How to devise a Linear Programming formulations, such that convex combinations of valid mappings can be recovered?!

Impossible to decompose and extract **any single valid mapping**. **Intuition:** Node i is mapped to u_1 and the only neighboring node that hosts j is u_2, so i must be fully mapped on u_1 and j on u_2. Similarly, k must be mapped on u_3. But flow of virtual edge (k,i) leaving u_3 only leads to u_4, so **i must be mapped on both u_1 and u_4**. This is **impossible**, even if **capacities are infinite**.

ot be decomposed

s (so we need

Partial Decomposition

- Example:

LP Solution

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

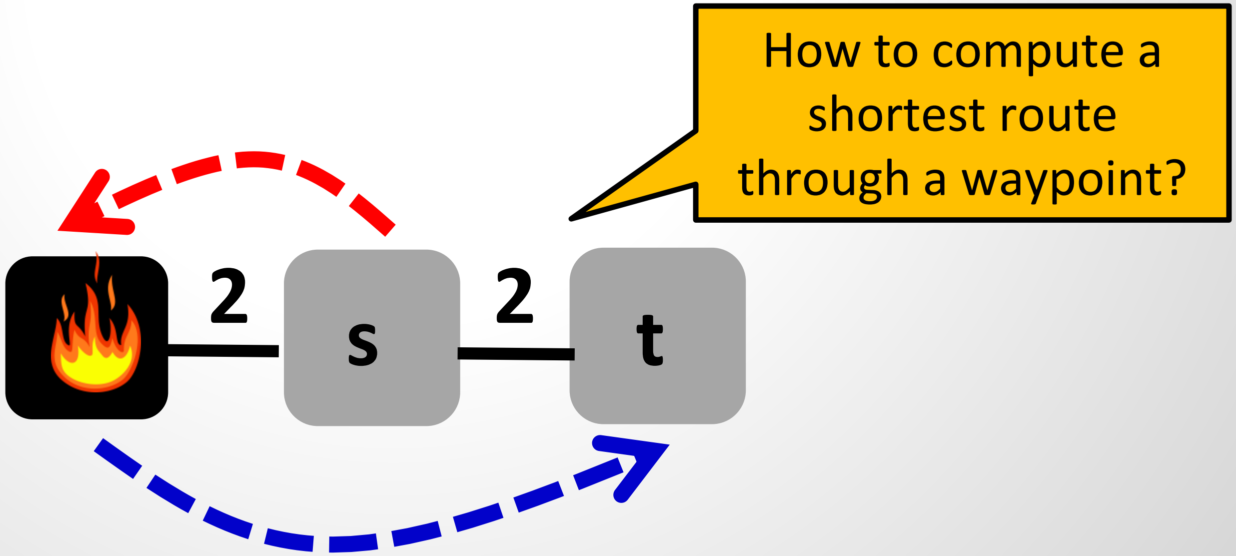
mappings can be recovered?!

.5i

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

Novelty:

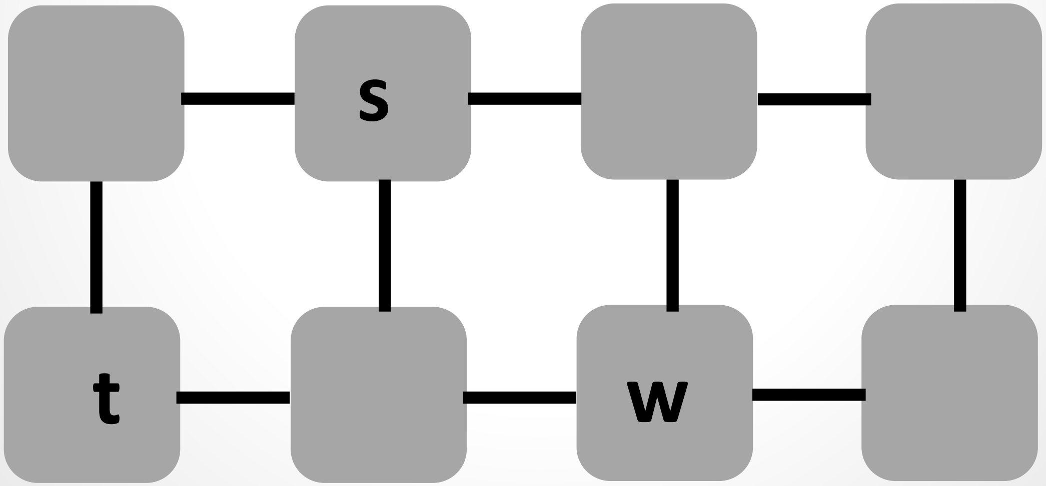
- Traditionally: routes form simple paths (e.g., shortest paths)
- Now: routing through middleboxes may require more general paths, with loops: a walk



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

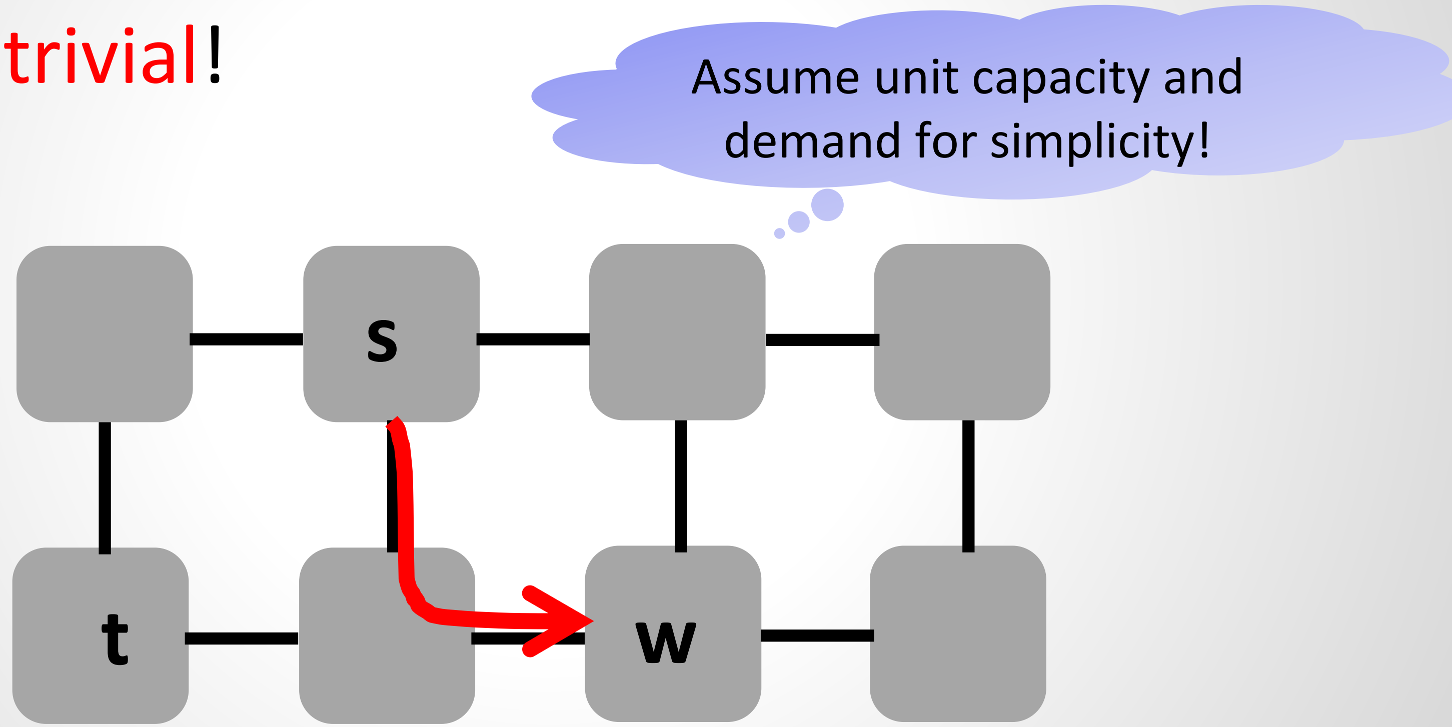
- Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!



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

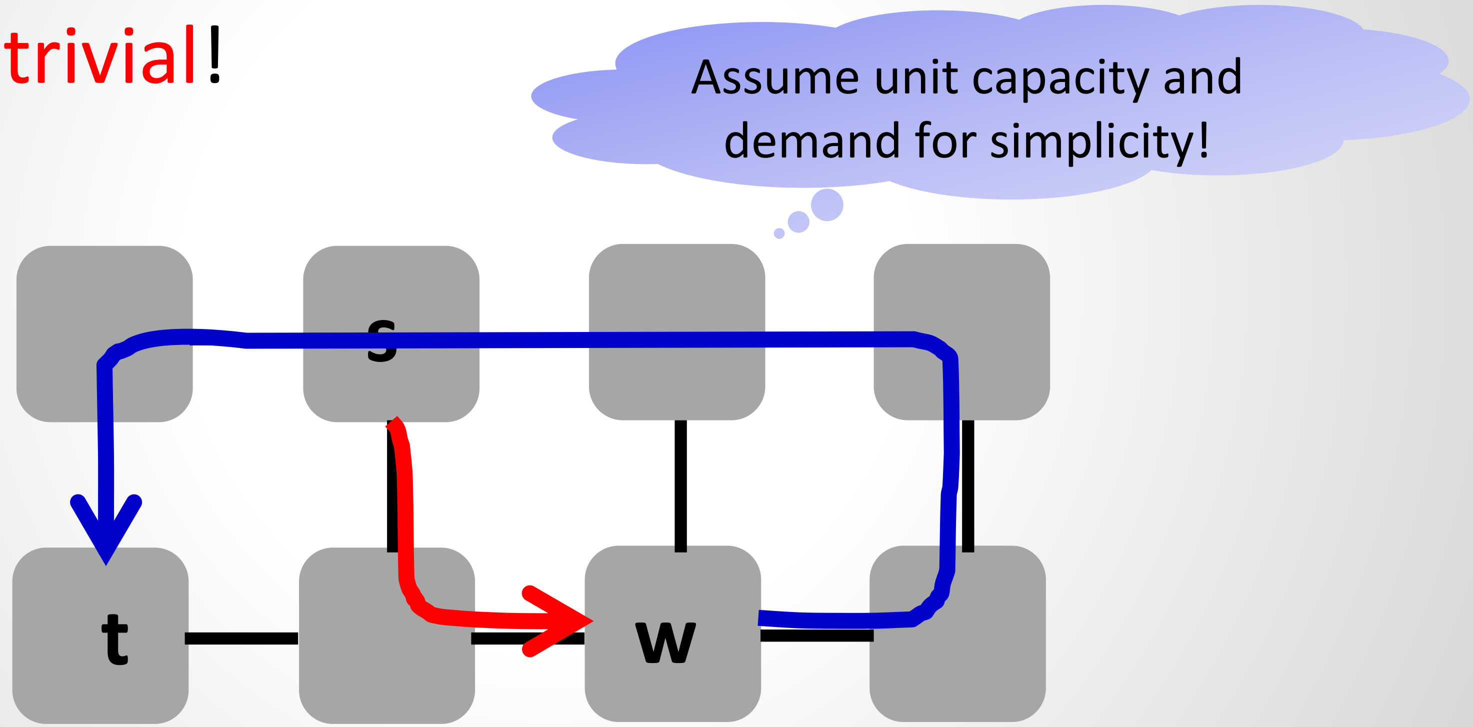
- Computing shortest routes through waypoints is non-trivial!



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

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

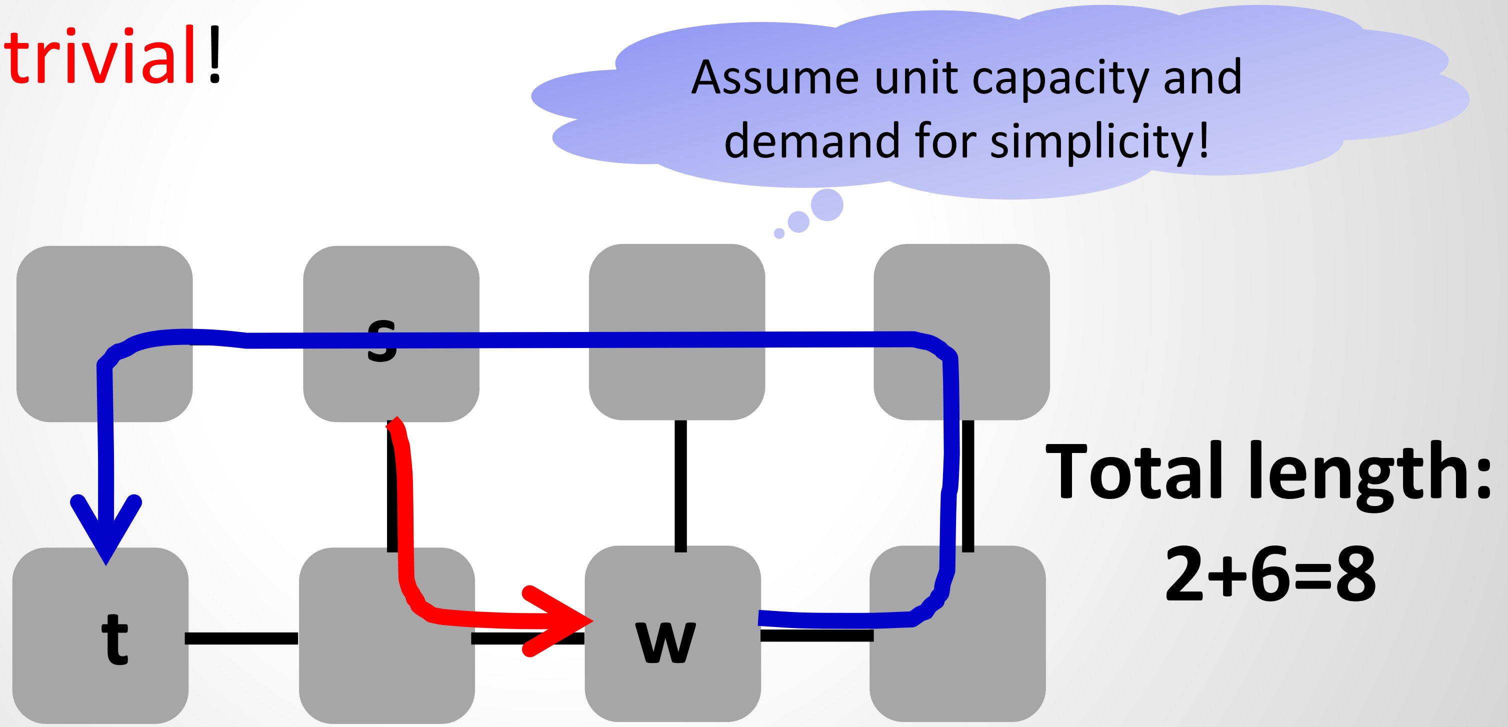
- Computing shortest routes through waypoints is non-trivial!



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

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

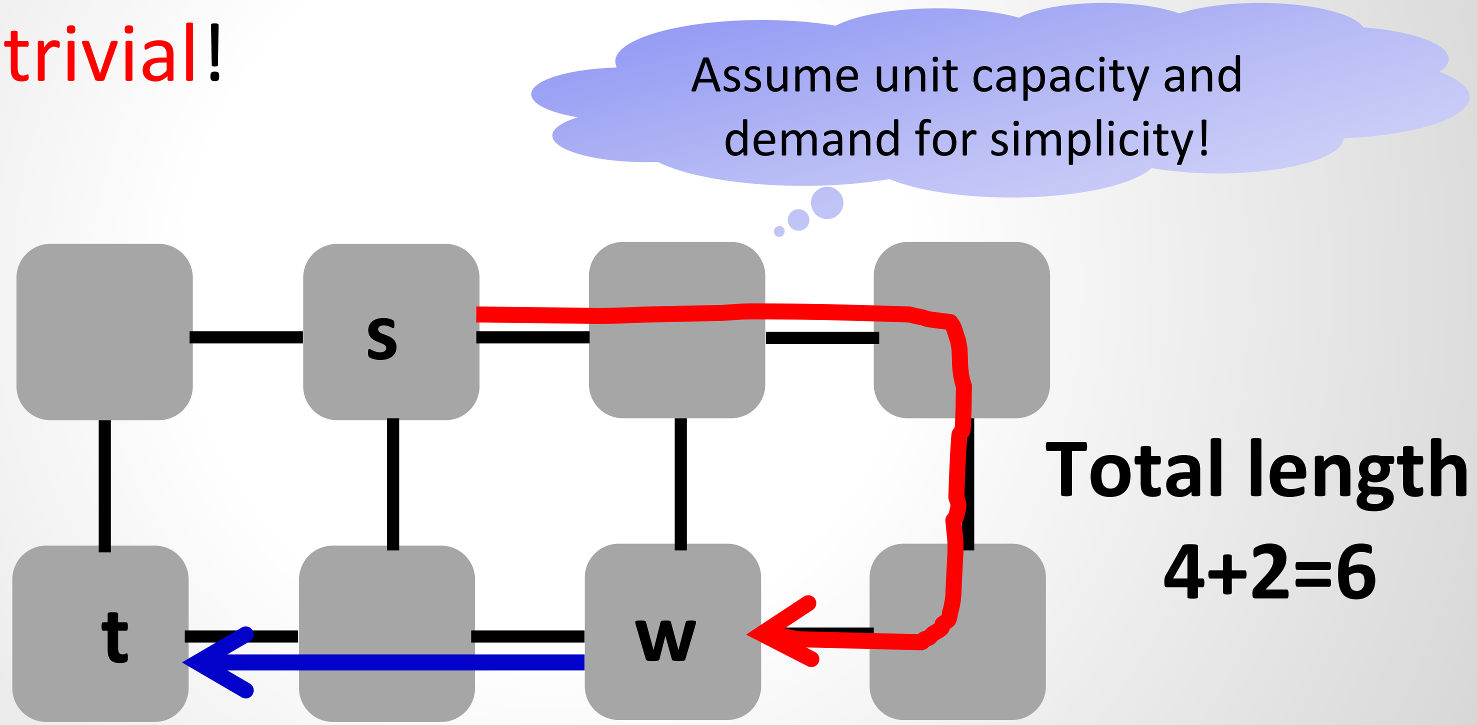
- Computing shortest routes through waypoints is non-trivial!



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!



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

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!

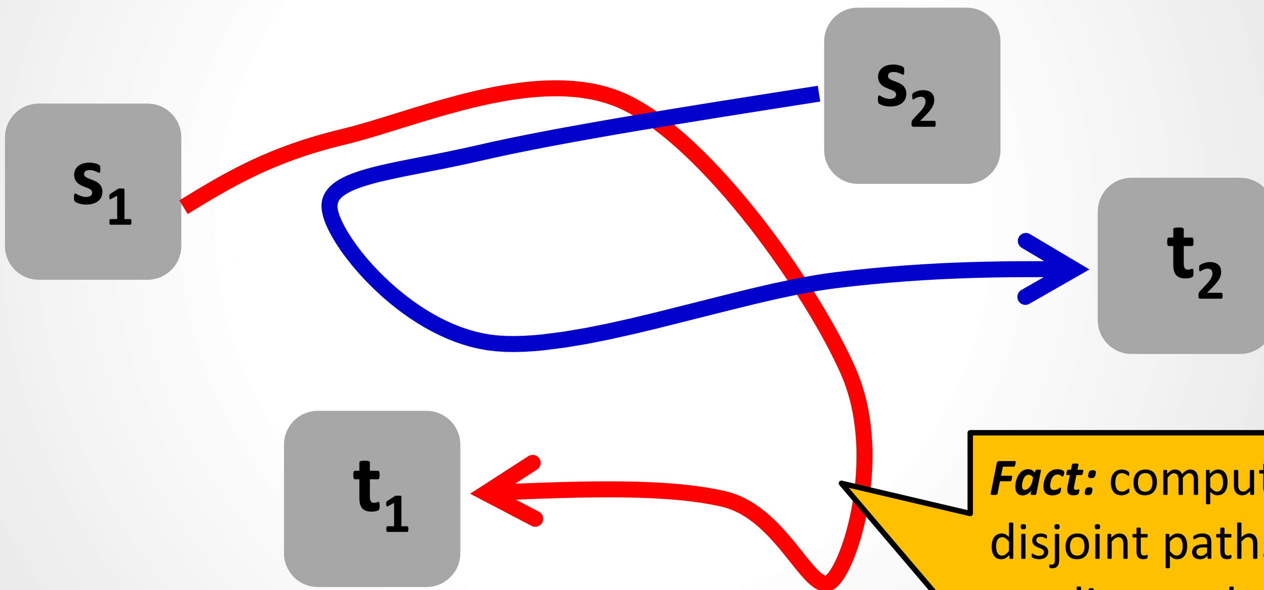
Similar to computing **shortest disjoint paths** (if capacities are 1, segments need to be disjoint): a well-known combinatorial problem!

NP-hard on directed networks (feasibility in P on undirected networks, optimality unknown).

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

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

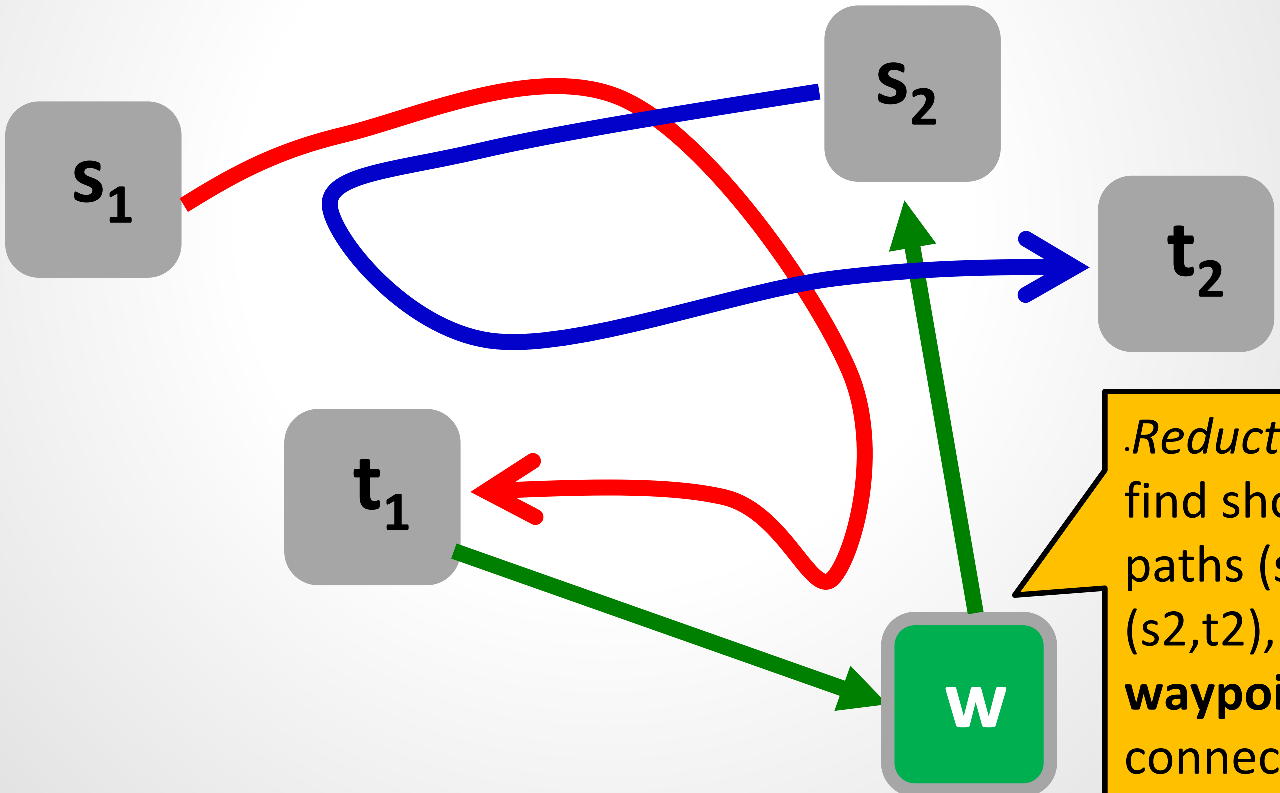


Fact: computing 2-disjoint paths is NP-hard on directed graphs.

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

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

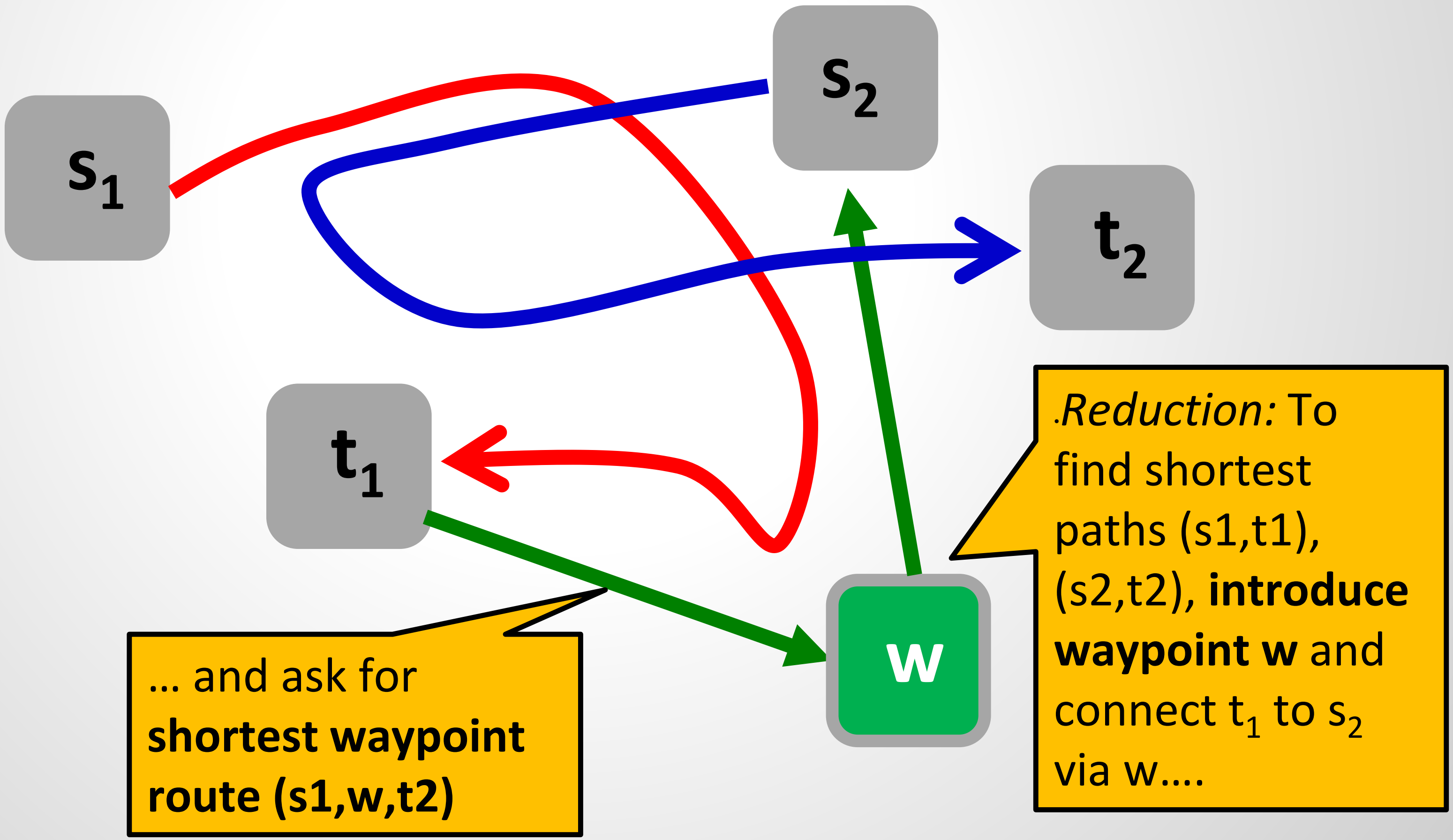
Reduction: *From* joint shortest paths $(s_1,t_1),(s_2,t_2)$ *to* shortest walk (s,w,t) problem



.Reduction: To find shortest paths (s1,t1), (s2,t2), **introduce waypoint w** and connect t_1 to s_2 via w….

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



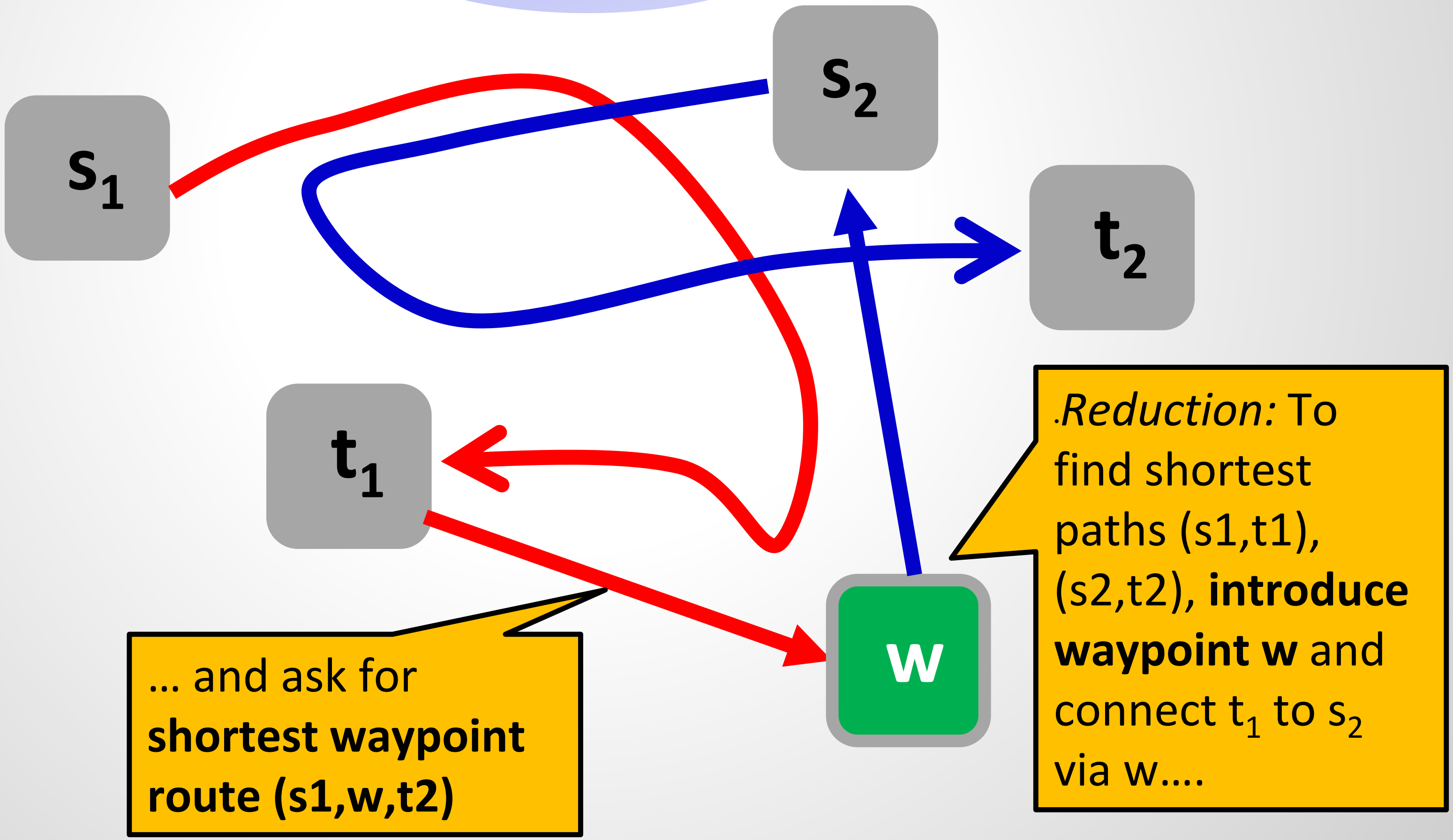
NP-hard on Direct... Netw...

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?

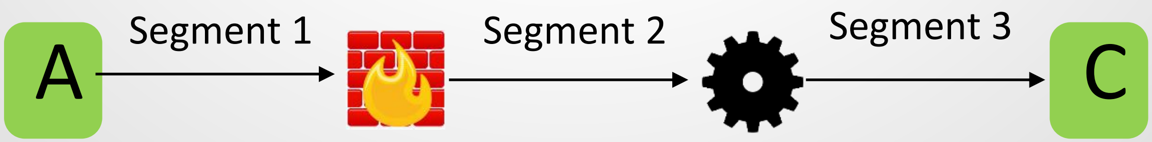
What about waypoint routes on *undirected* networks?

Option 1: If feasibility good enough: reduce it to disjoint paths problem!

- Replace capacitated links with undirected parallel links:

- Even works for multiple waypoints: Feasibility in P for constant number of flows

- So each path segment becomes a (disjoint) path



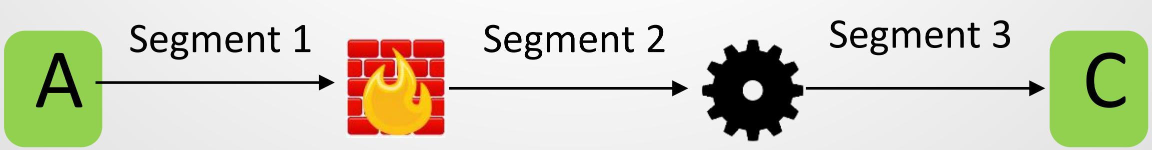
What about waypoint routes on *undirected* networks?

Option 1: If feasibility good enough: reduce it to disjoint paths problem!

- Replace capacitated links with undirected parallel links:

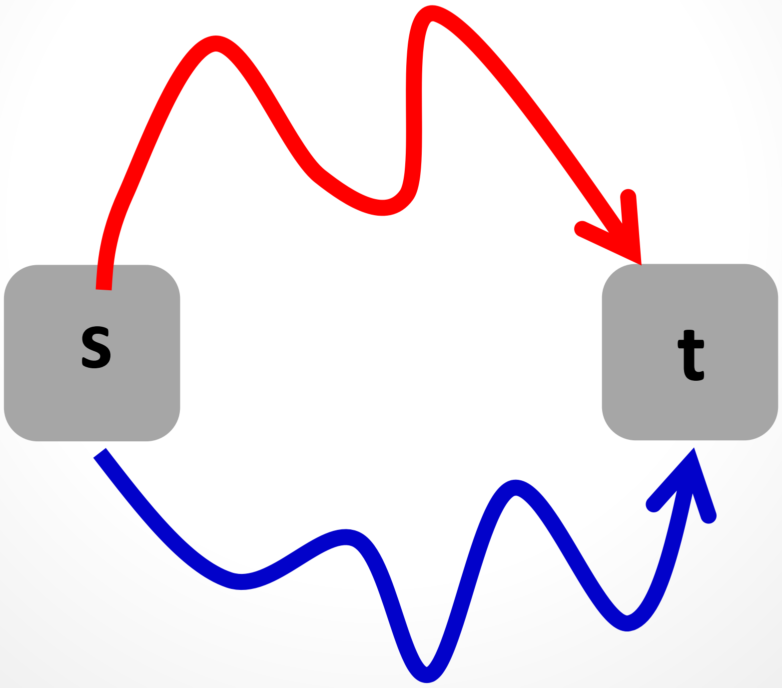
Good news: For a single waypoint, ***shortest*** paths can be computed even ***faster***!

- So each path segment becomes a (disjoint) path



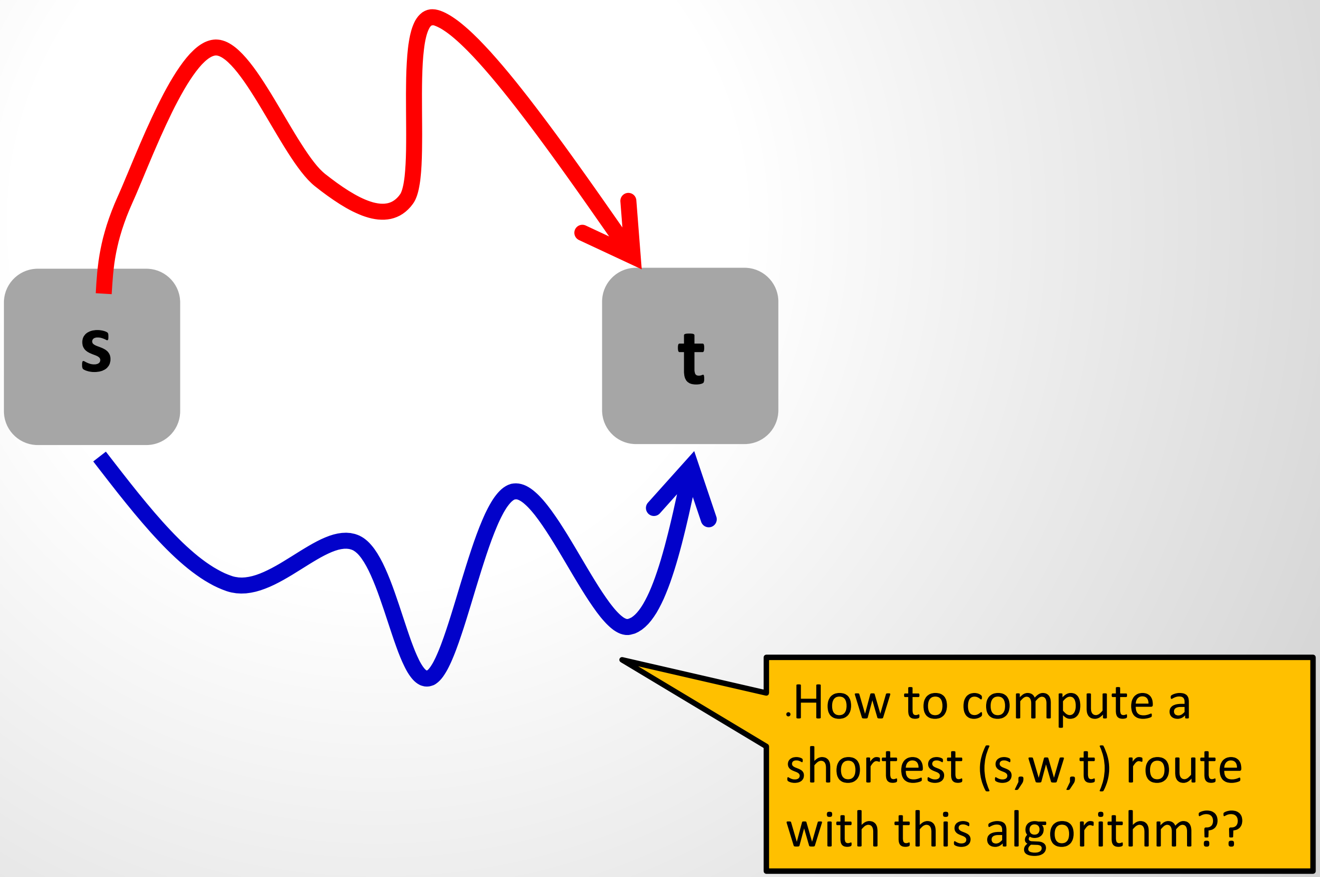
Good news: Not NP-hard on Undirected Networks: Suurballe's Algorithm

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



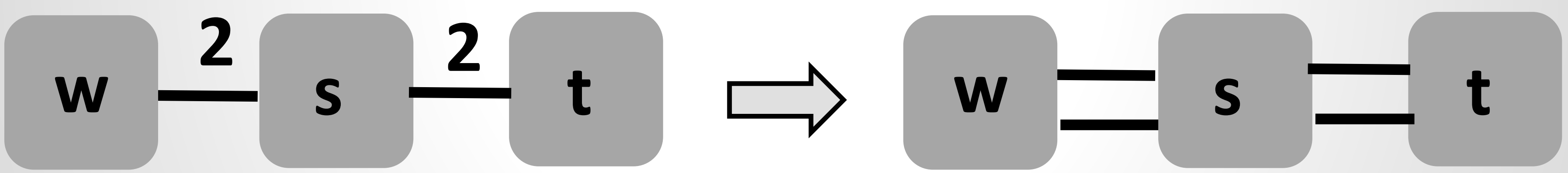
Good news: Not NP-hard on Undirected Networks: Suurballe's Algorithm

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



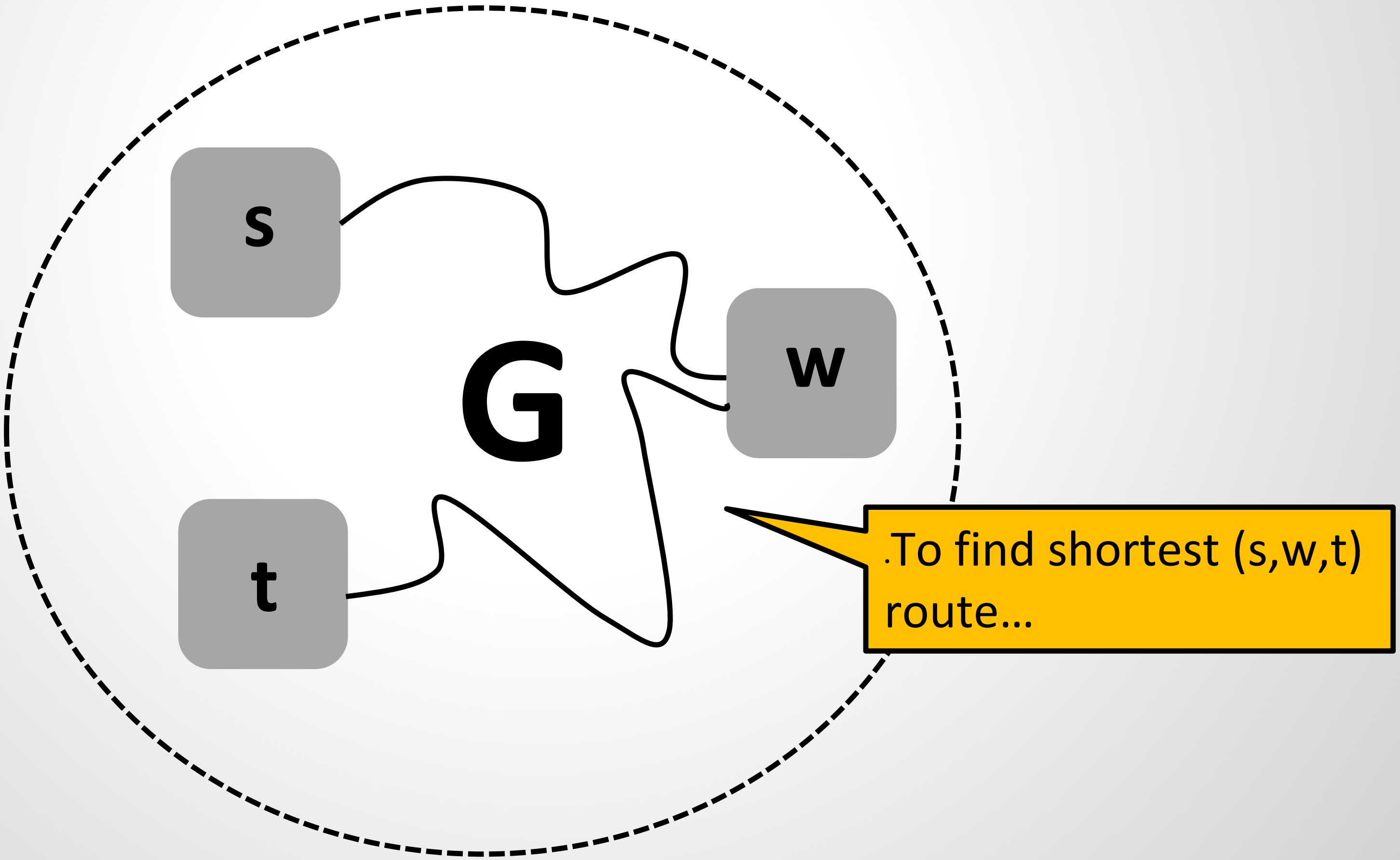
Good news: Not NP-hard on Undirected Networks: Suurballe's Algorithm

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



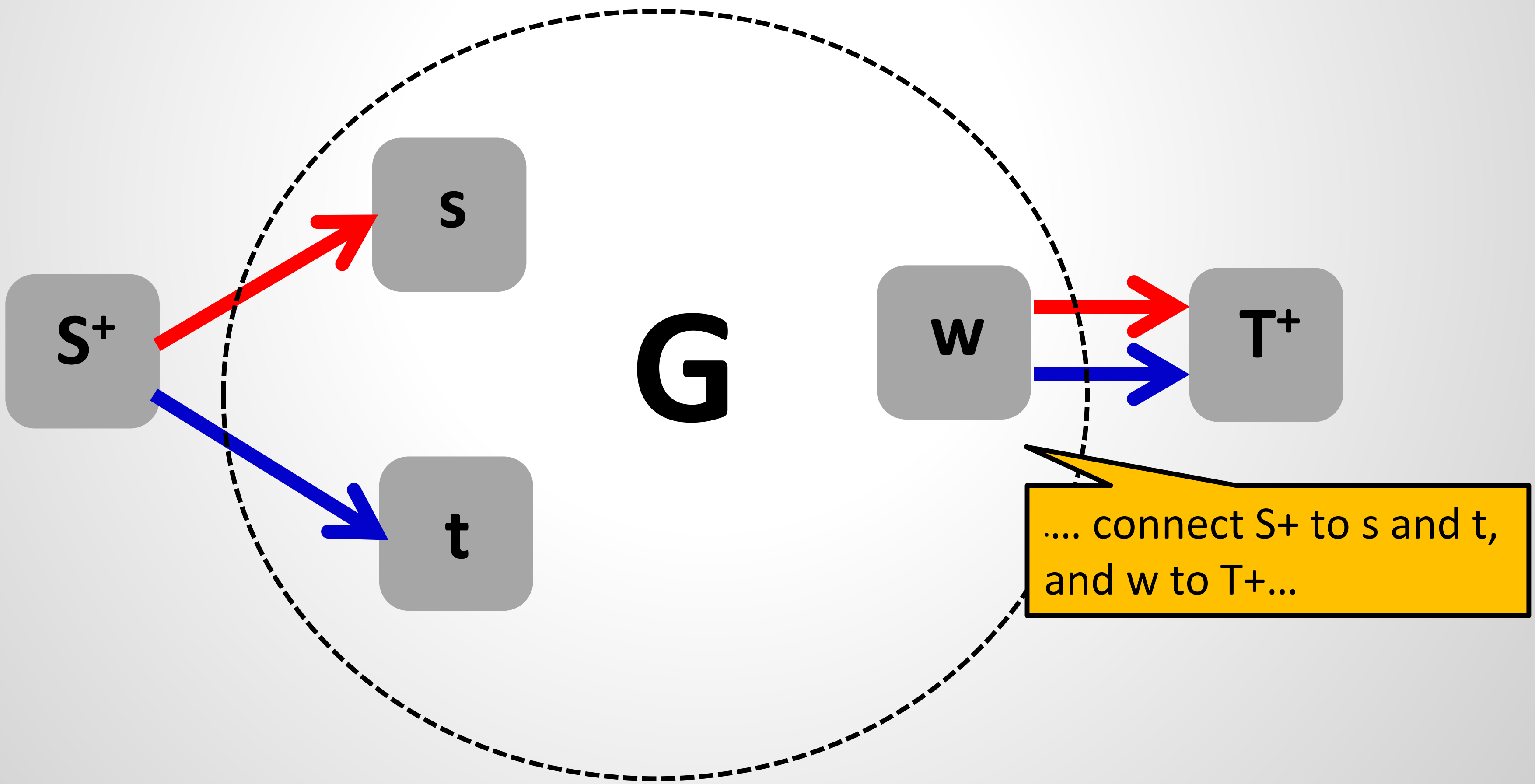
Waypoint Routing on Steroids

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



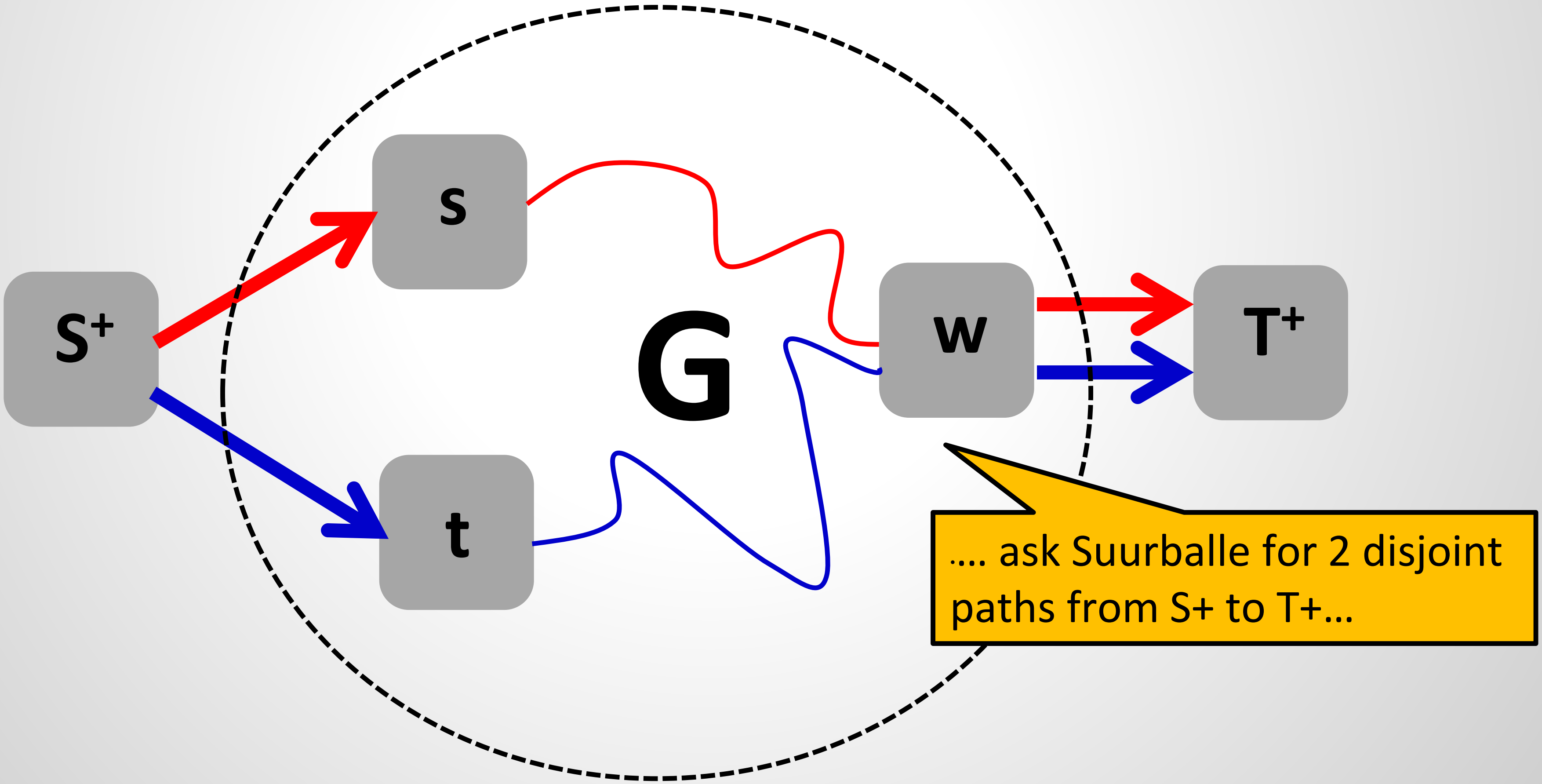
Waypoint Routing on Steroids

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



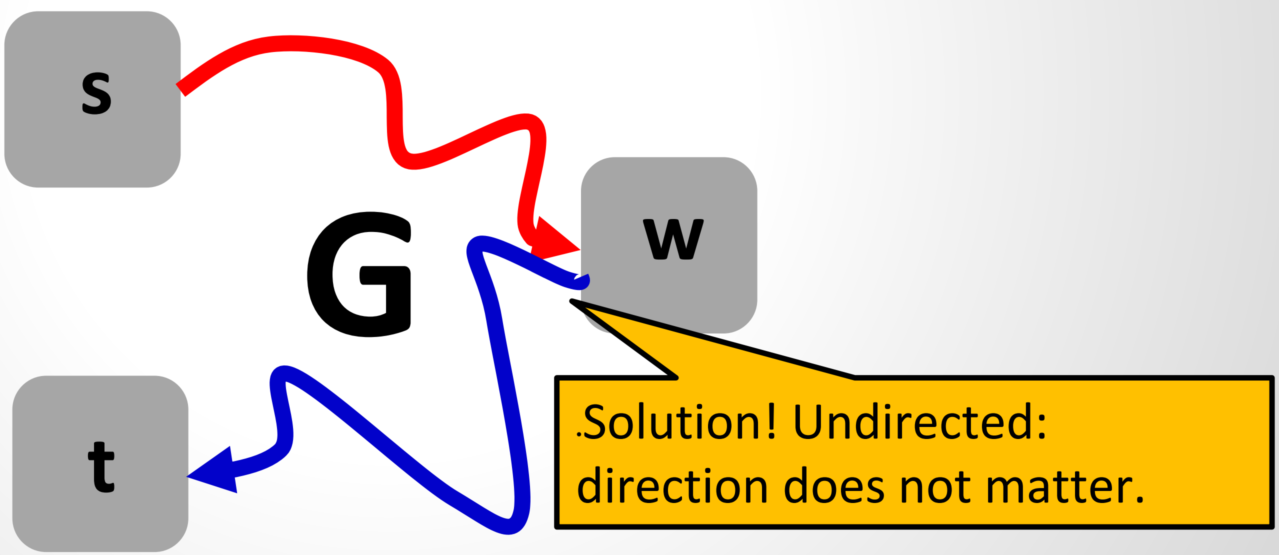
Waypoint Routing on Steroids

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



Waypoint Routing on Steroids

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



Open Question

For which other service chains can we compute optimal embeddings fast?

Further reading:

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

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

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!

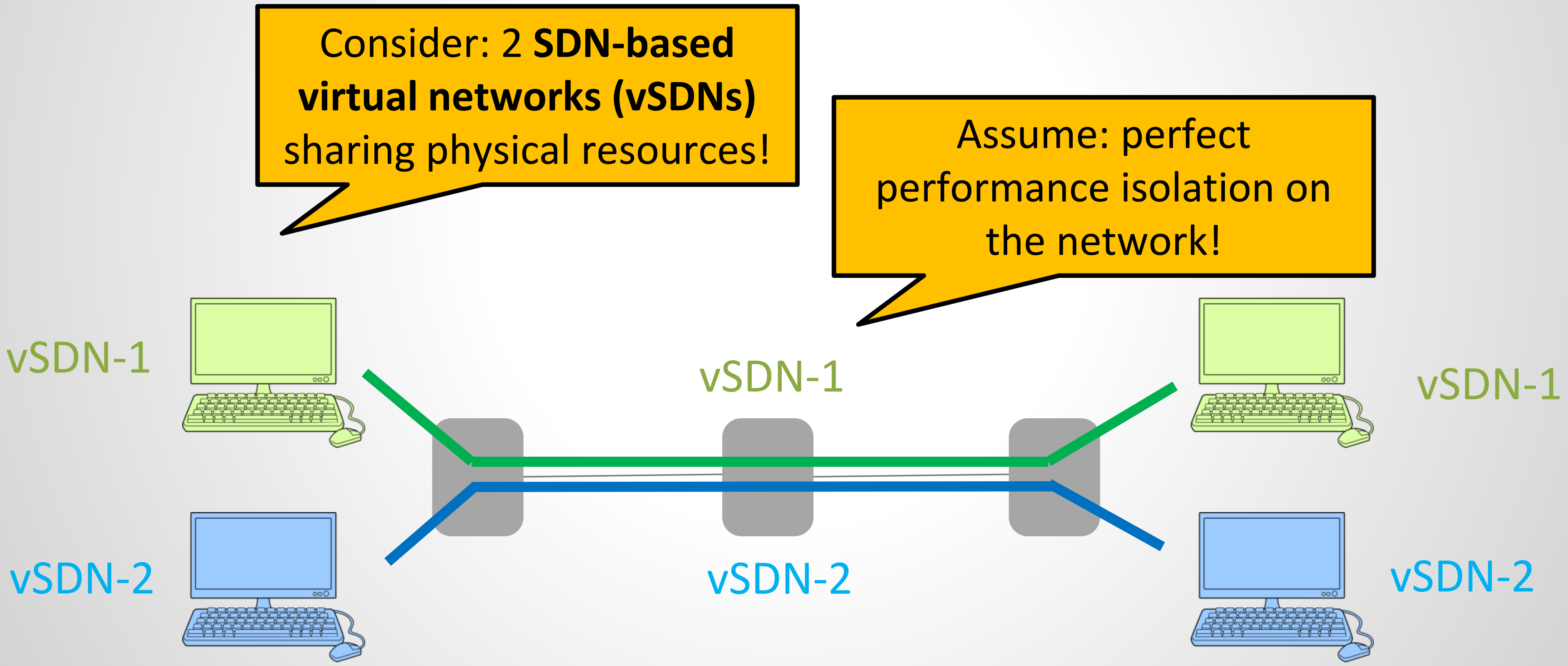
Boss: So can I promise our customers a predictable performance?

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?

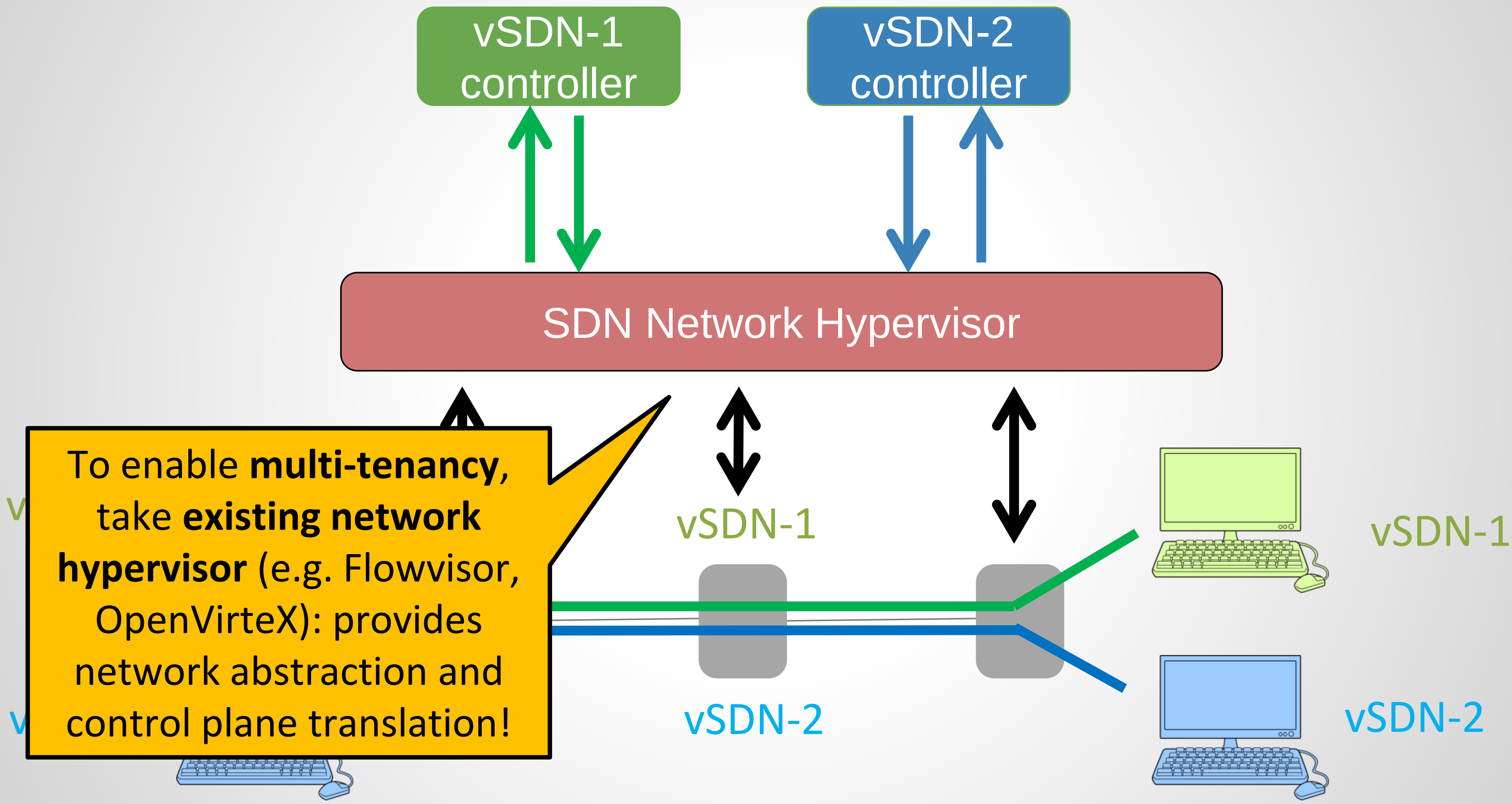
You: hmmm….

The Many Faces of Performance Interference



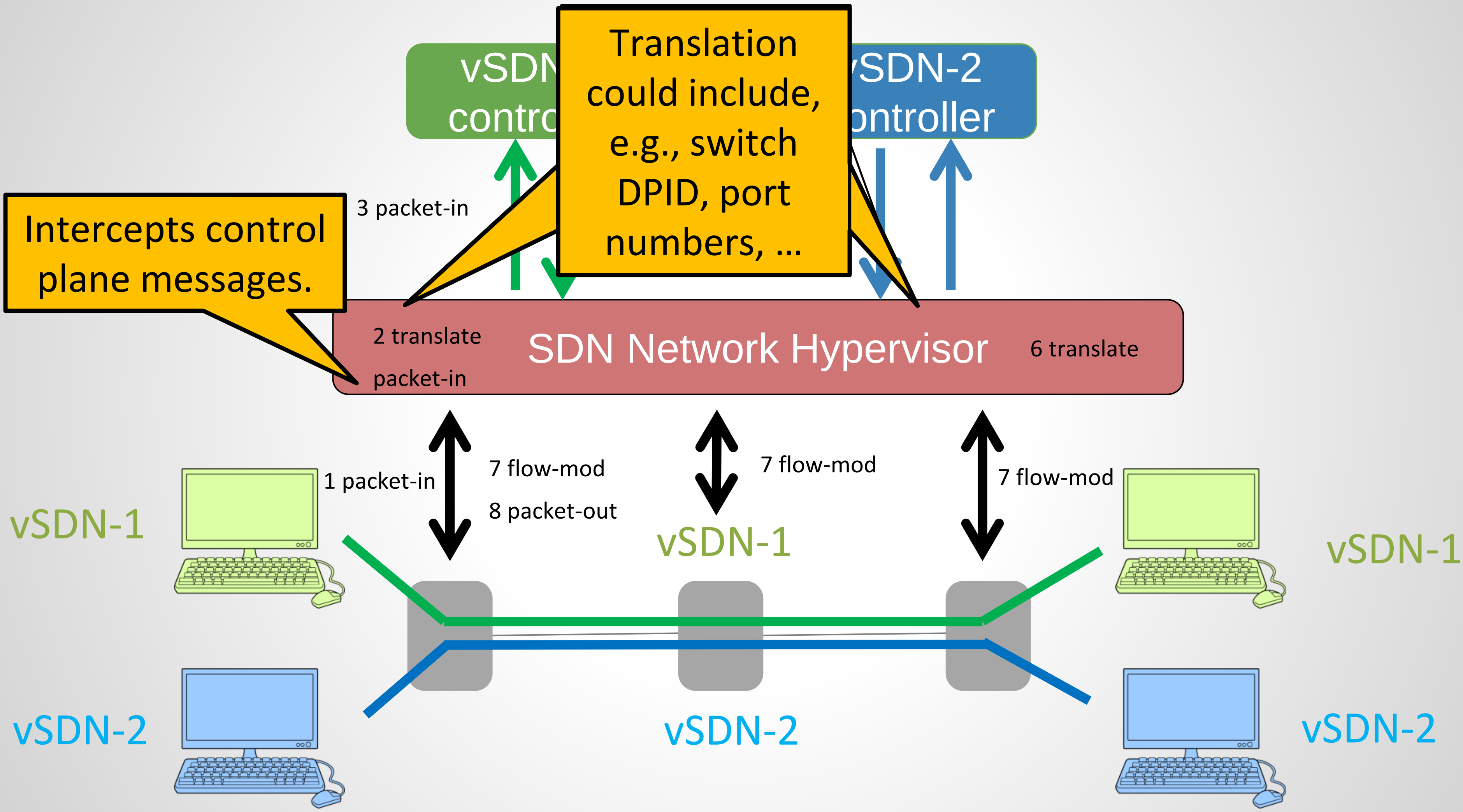
An Experiment: 2 vSDNs with bw guarantee!

The Many Faces of Performance Interference



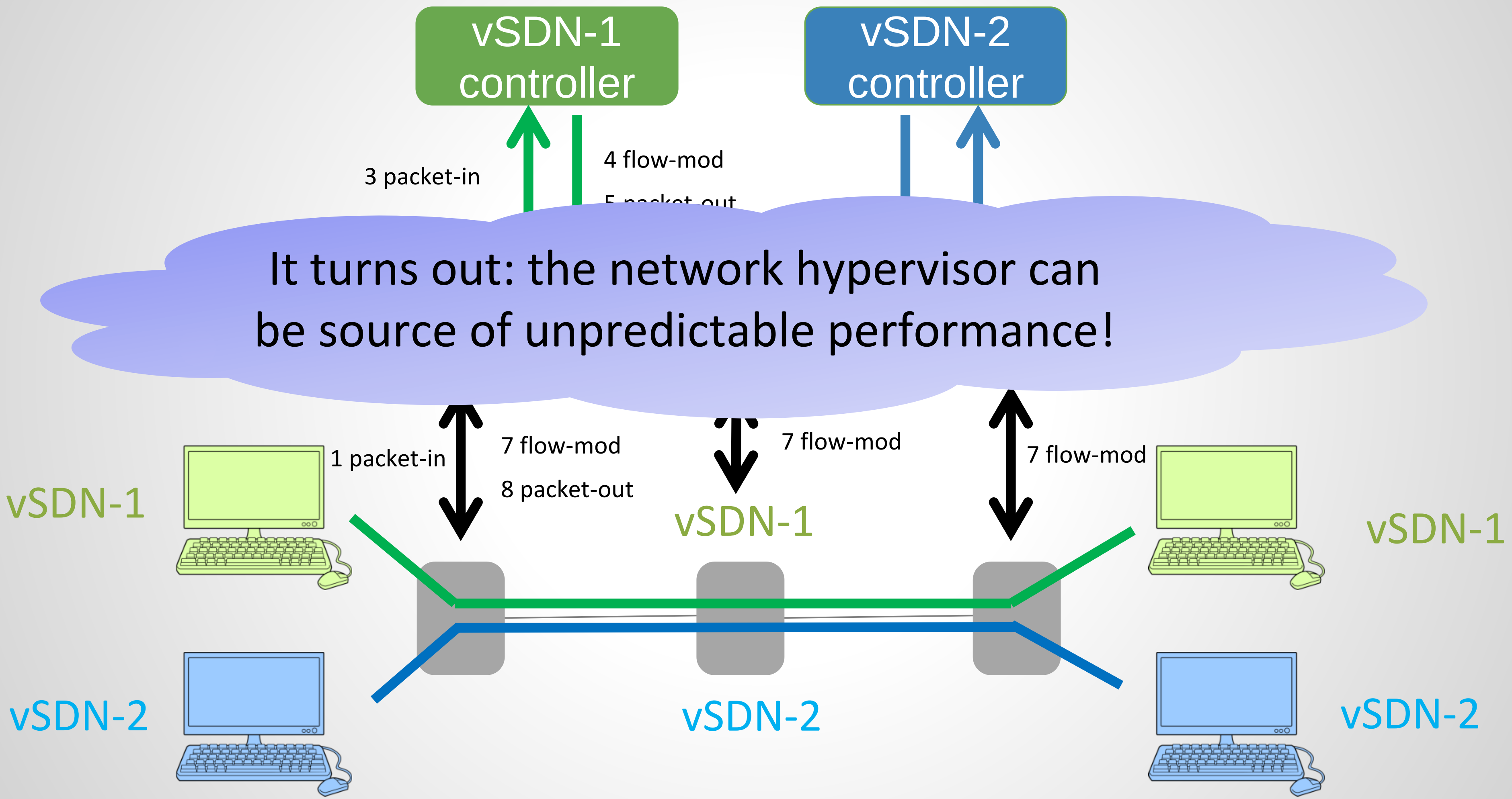
An Experiment: 2 vSDNs with bw guarantee!

The Many Faces of Performance Interference



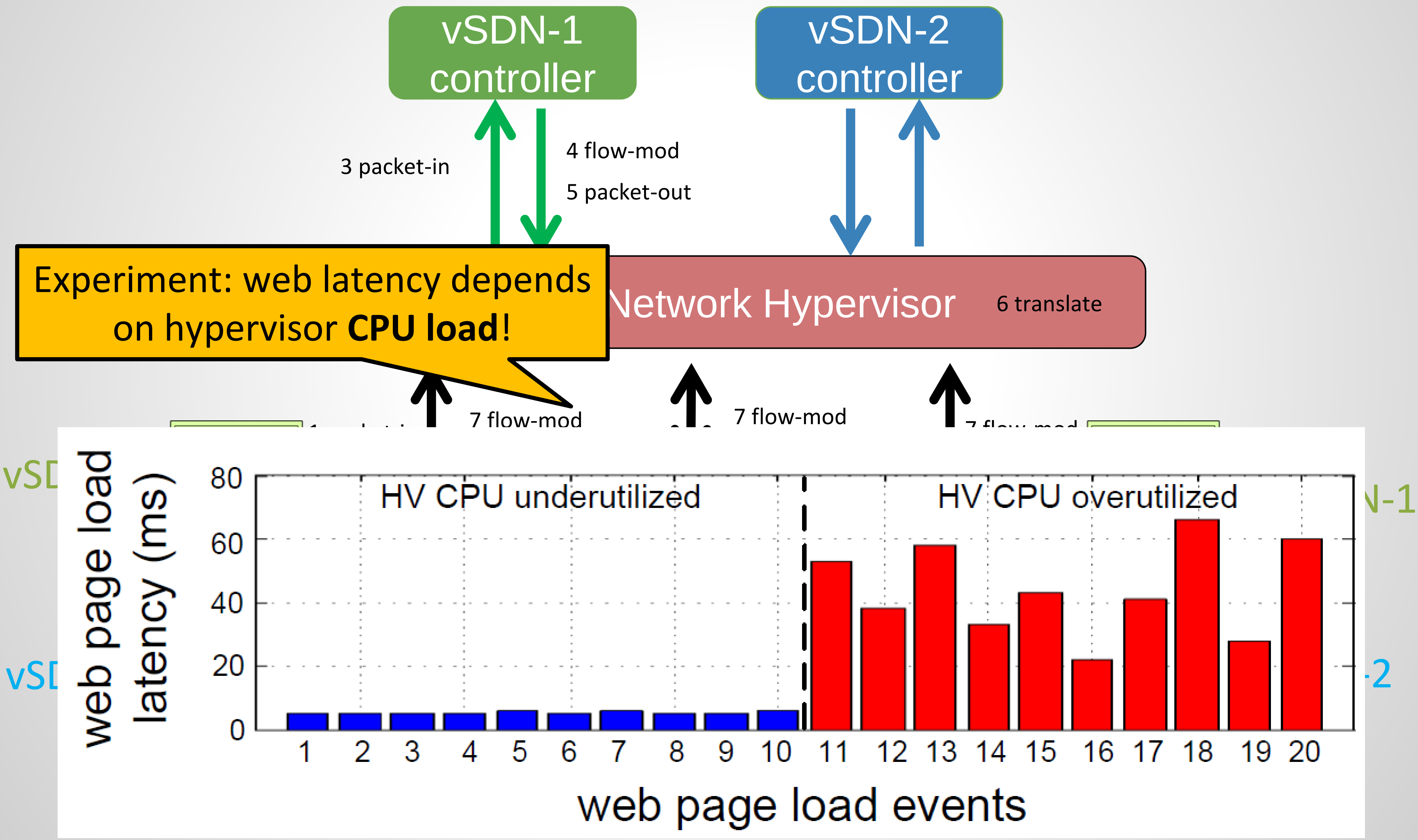
An Experiment: 2 vSDNs with bw guarantee!

The Many Faces of Performance Interference

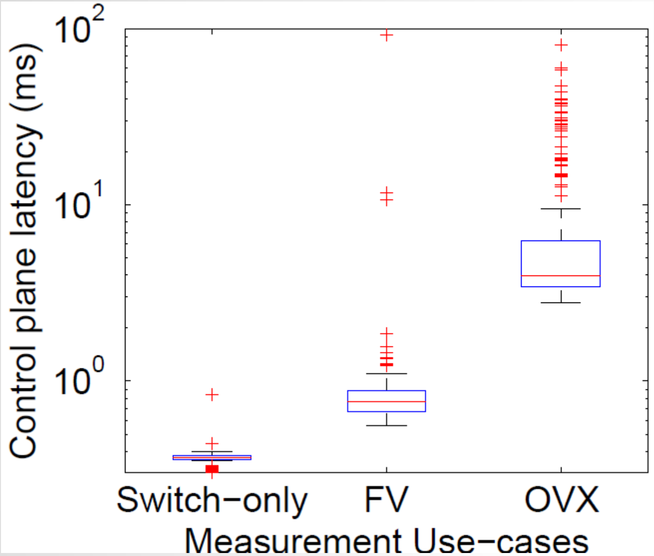


An Experiment: 2 vSDNs with bw guarantee!

The Many Faces of Performance Interference



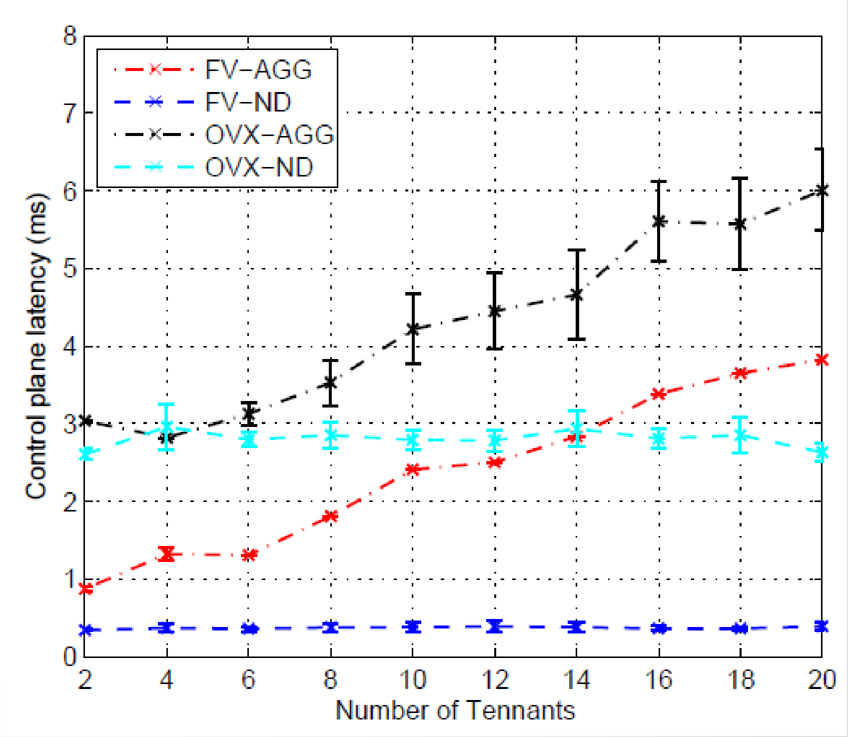
The Many Faces of Performance Interference



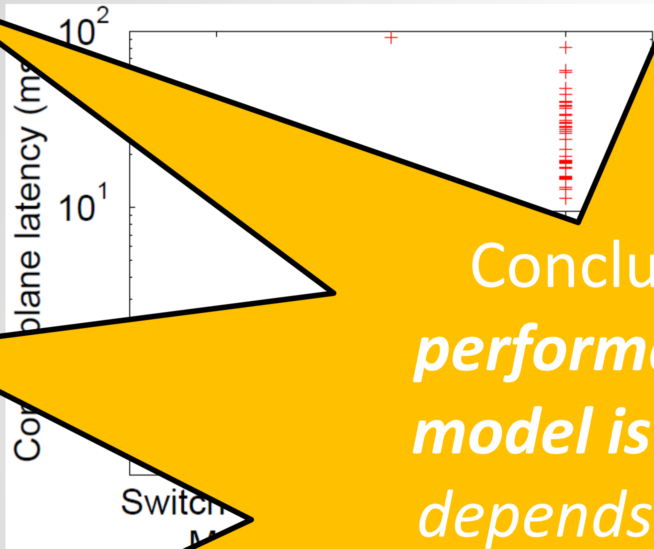
Performance also depends on hypervisor type…

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

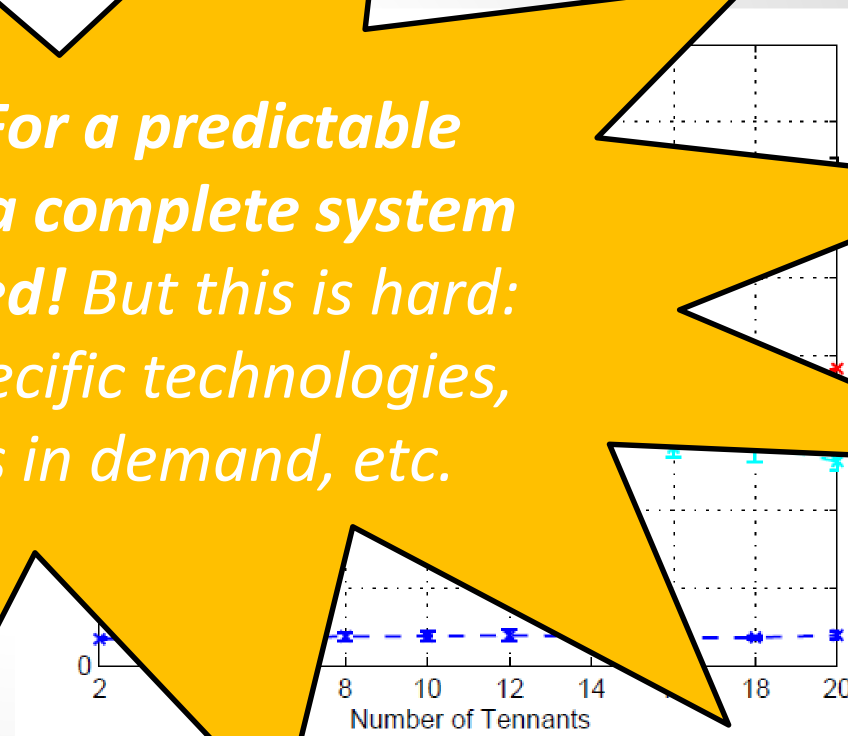
… number of tenants…



The Many Faces of Performance Interference



… er of tenants…

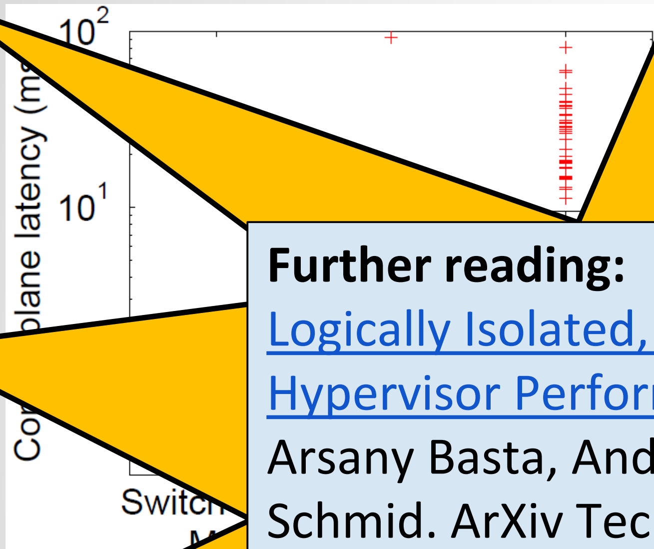


Performance a… on hypervi… type

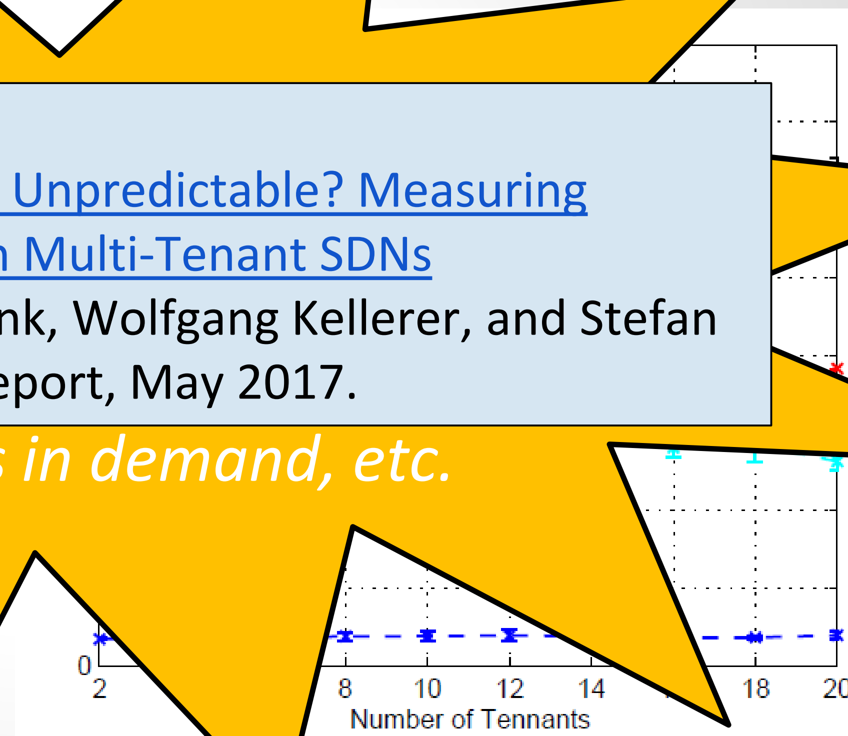
(multithreaded or not, which … of Nagle's algorithm, etc.

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.

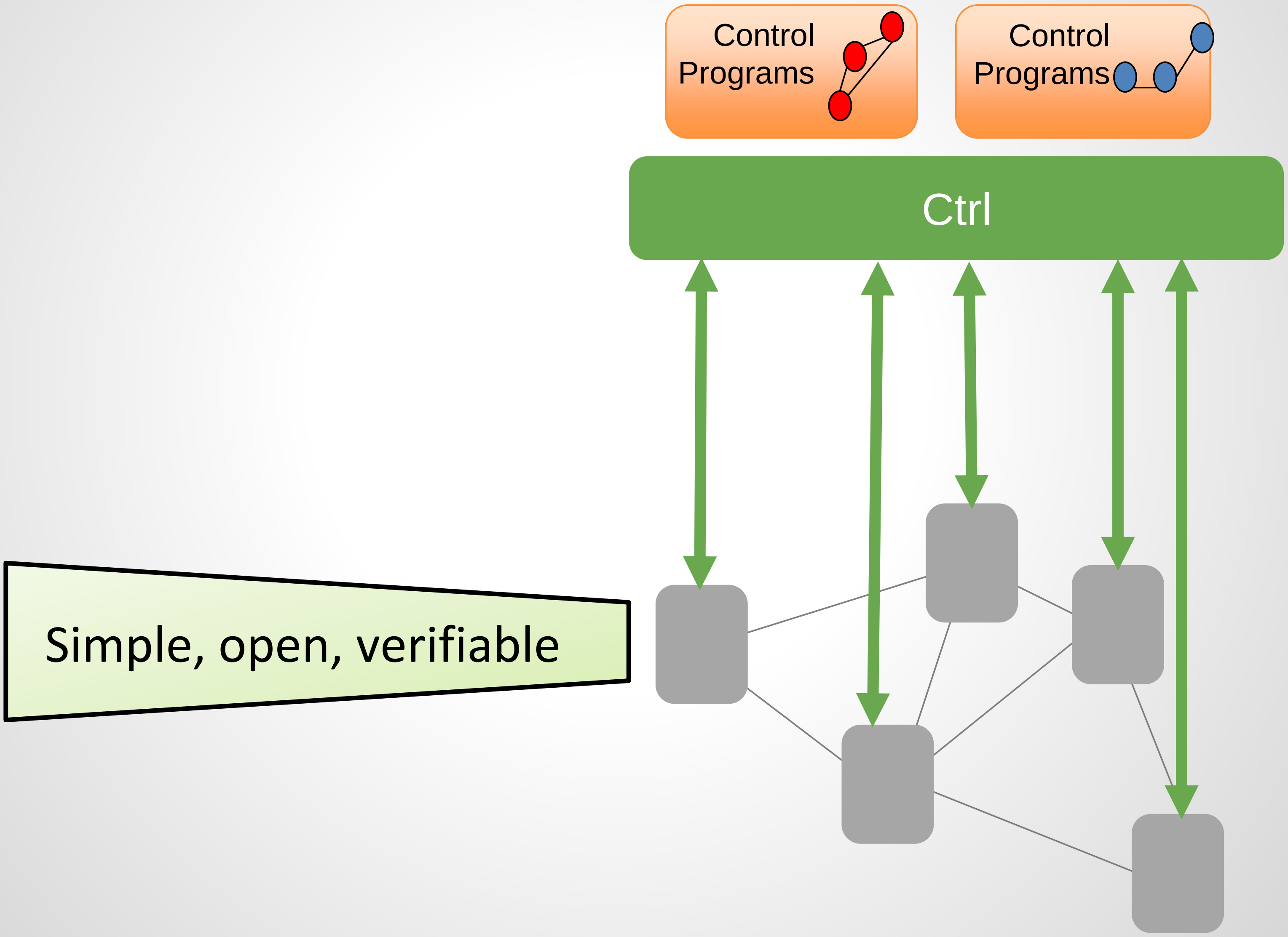
uncertainties in demand, etc.

Performance a

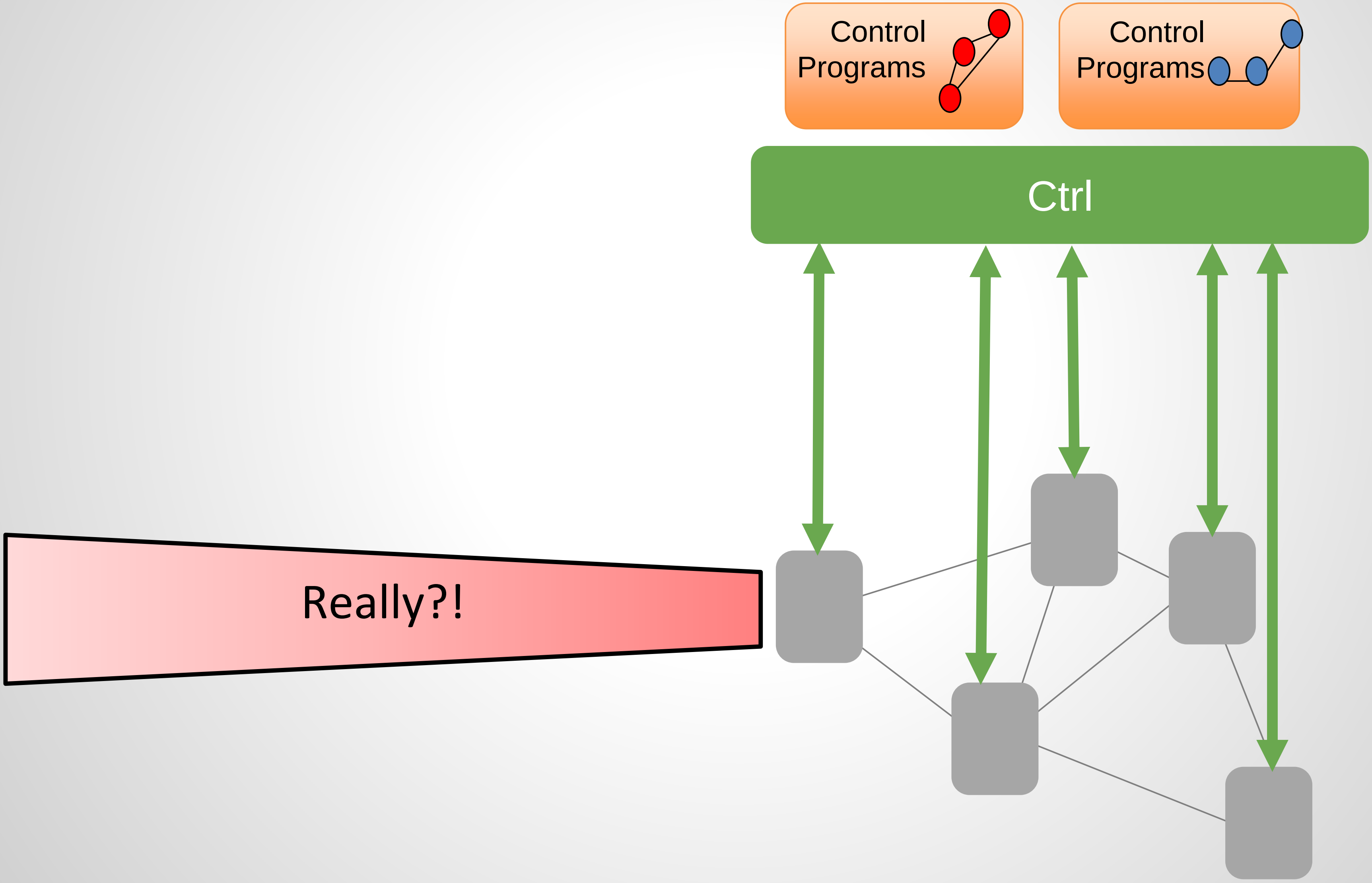
on hypervi typ

(multithreaded o not, which
of Nagle's algorithm, etc.

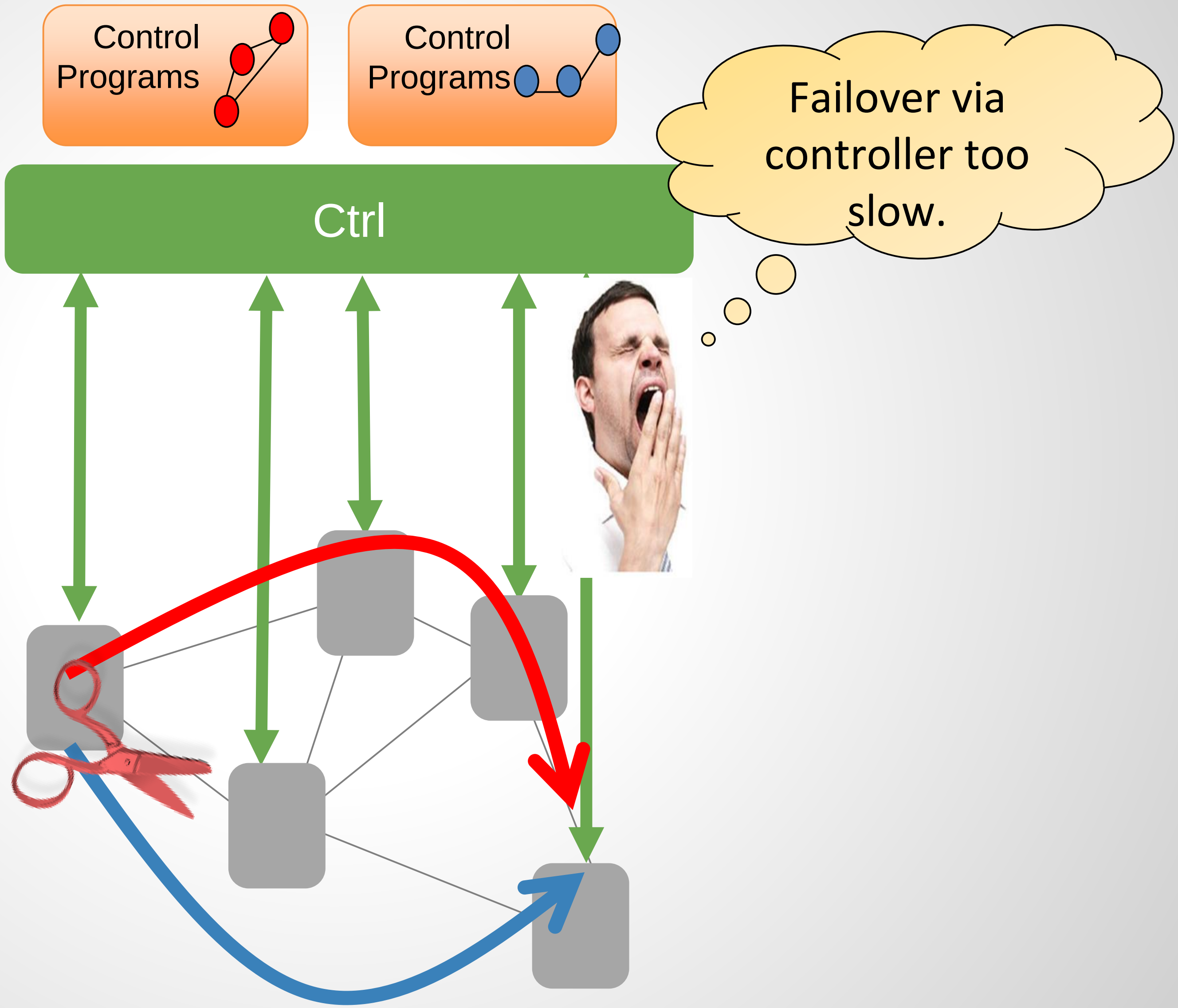
A Mental Model for This Talk



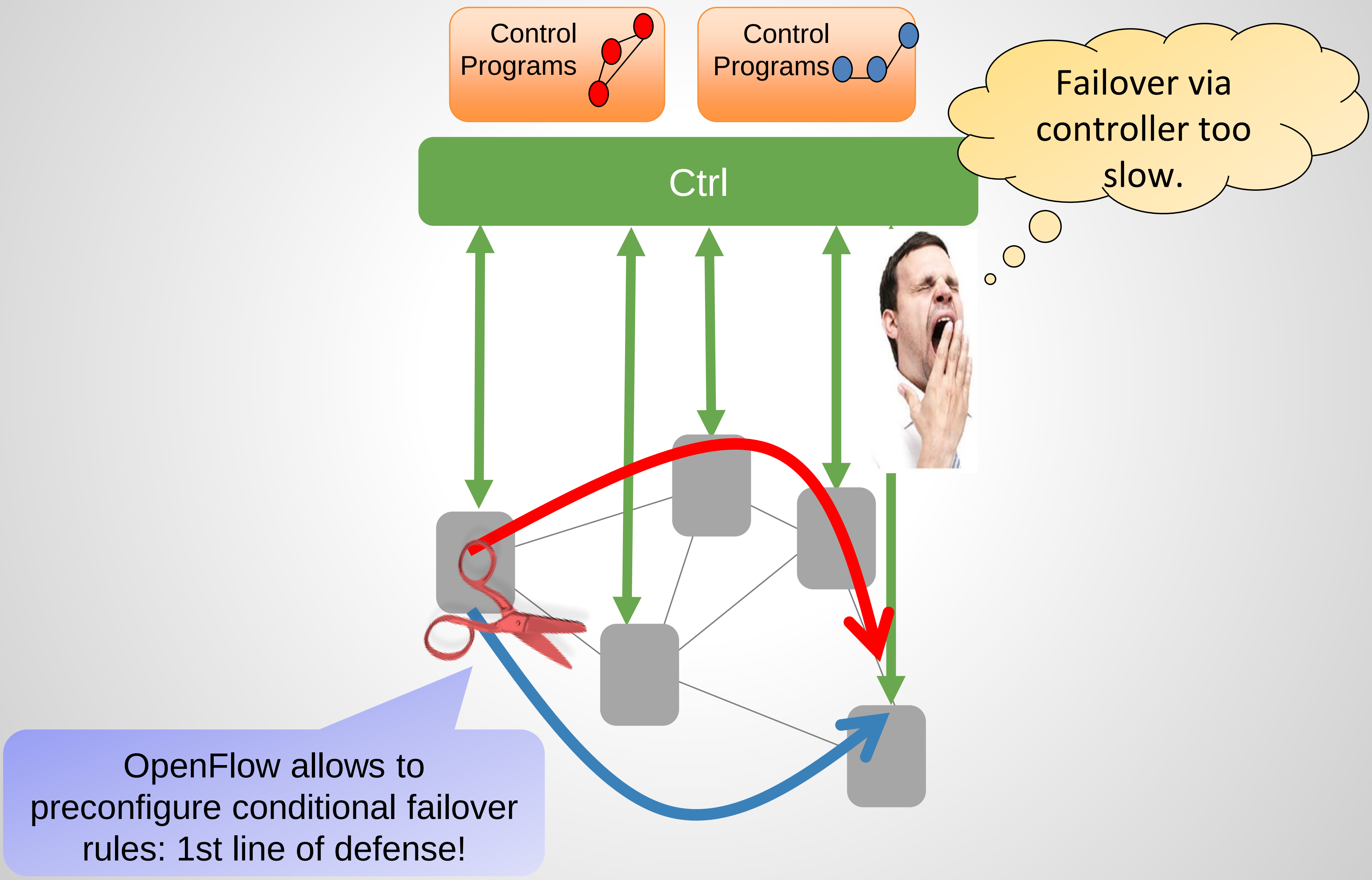
A Mental Model for This Talk



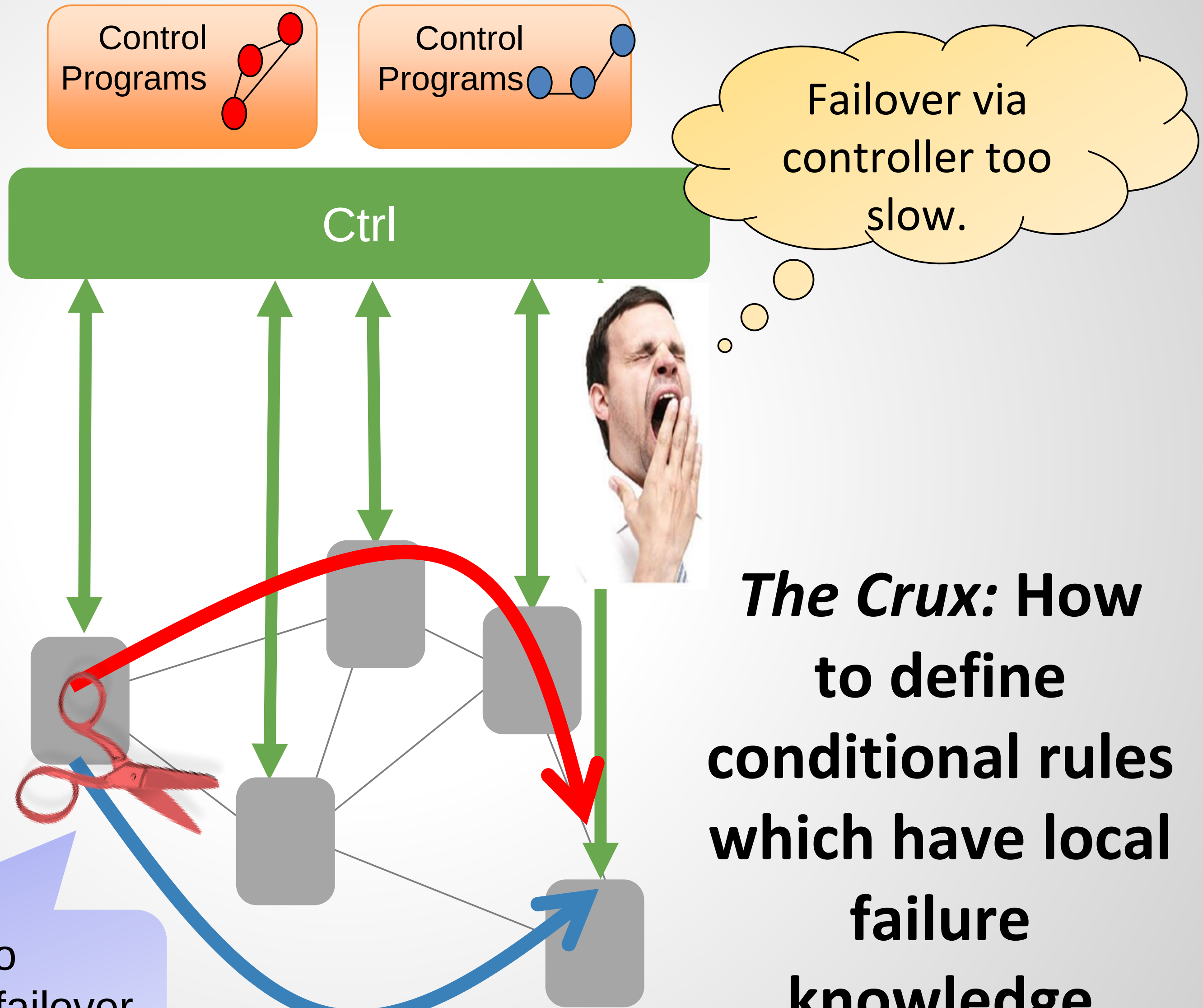
A Mental Model for This Talk



A Mental Model for This Talk



A Mental Model for This Talk

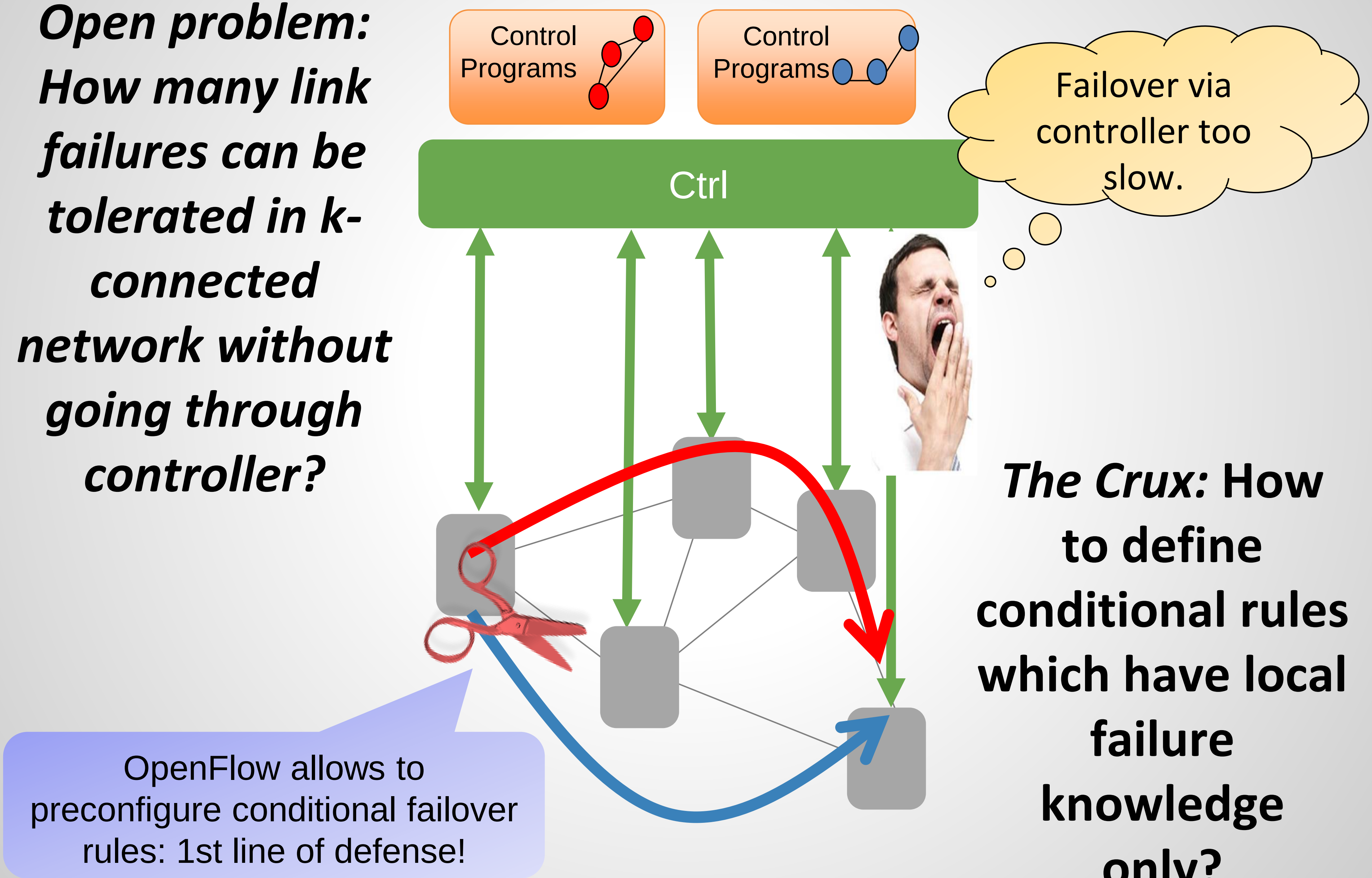


The Crux: How to define conditional rules which have local failure knowledge only?

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?**



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

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:

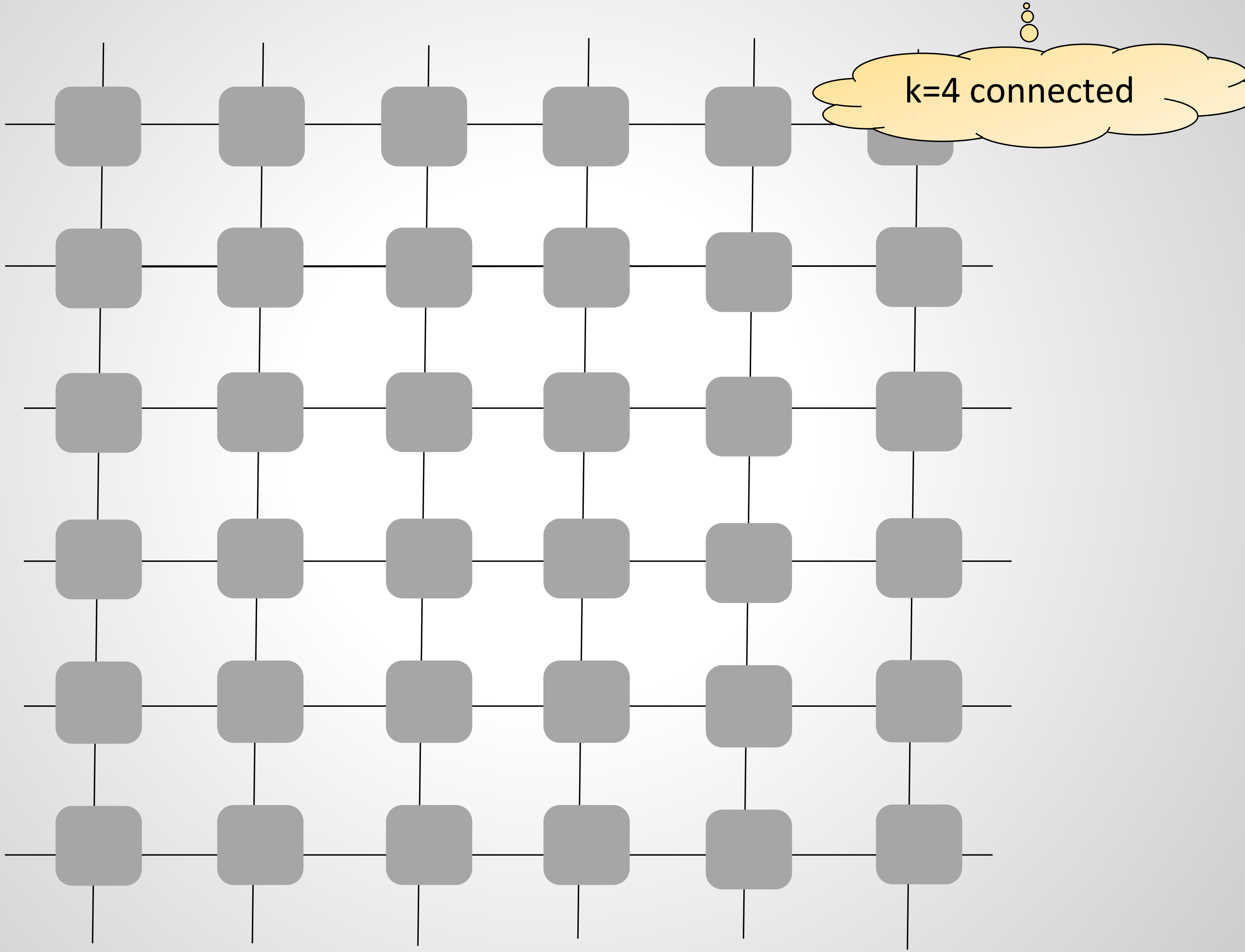
- Route along fixed arborescence ("directed spanning tree") towards the destination *d*
- If packet hits a failed edge at vertex *v*, reroute along a different arborescence

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

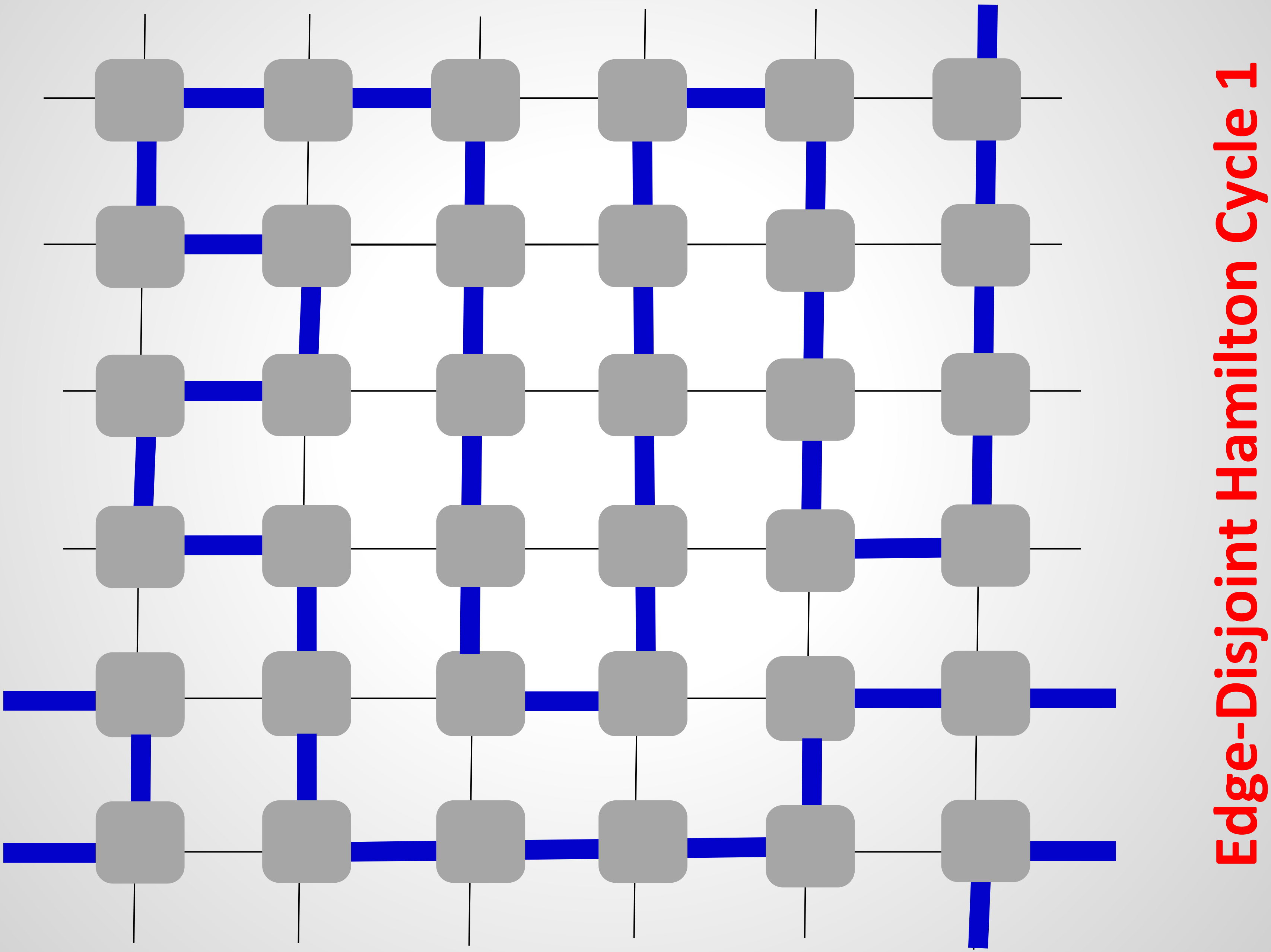
Simple Example: Hamilton Cycle

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

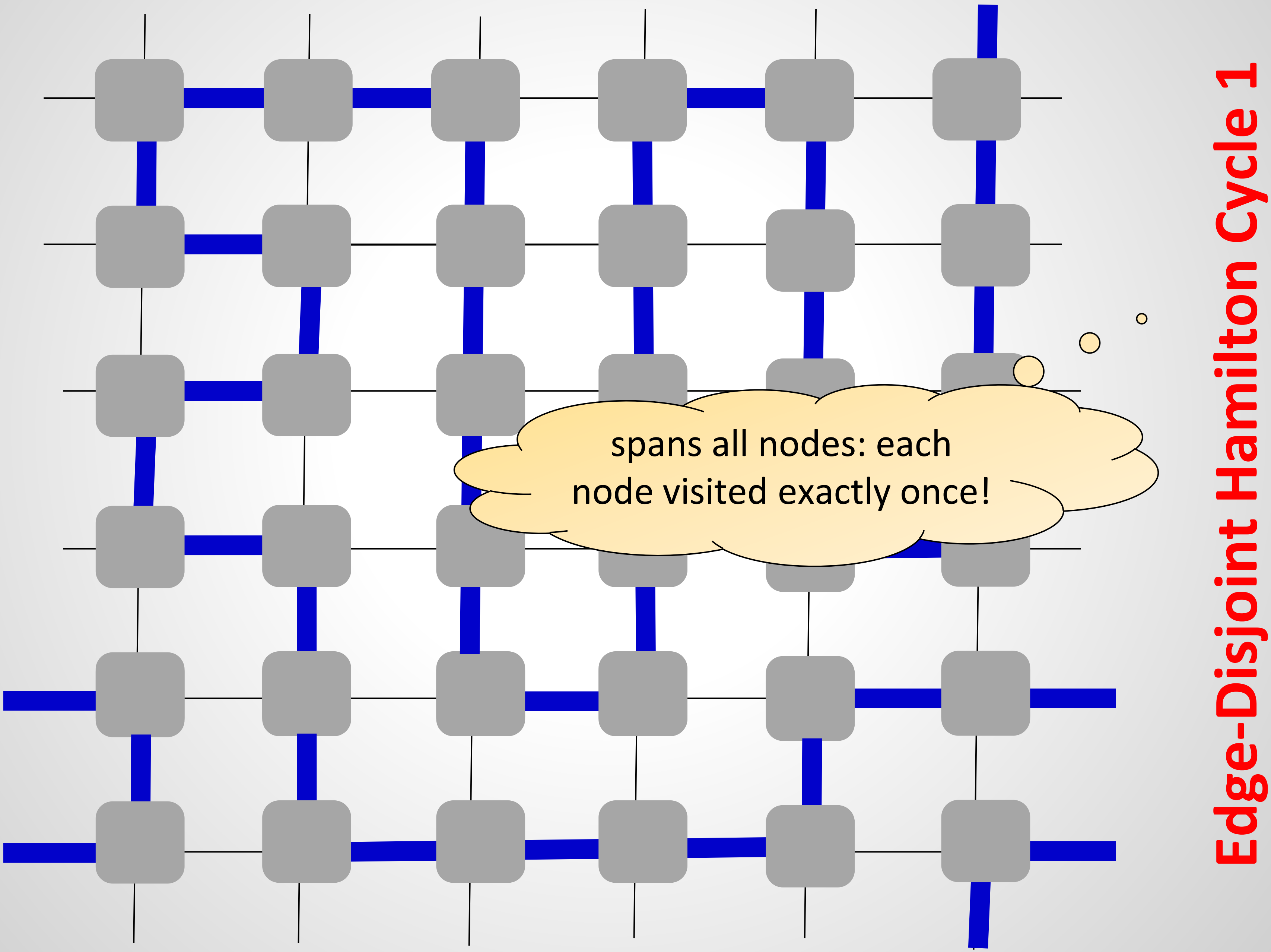
Example: 3-Resilient Routing Function for 2-dim Torus



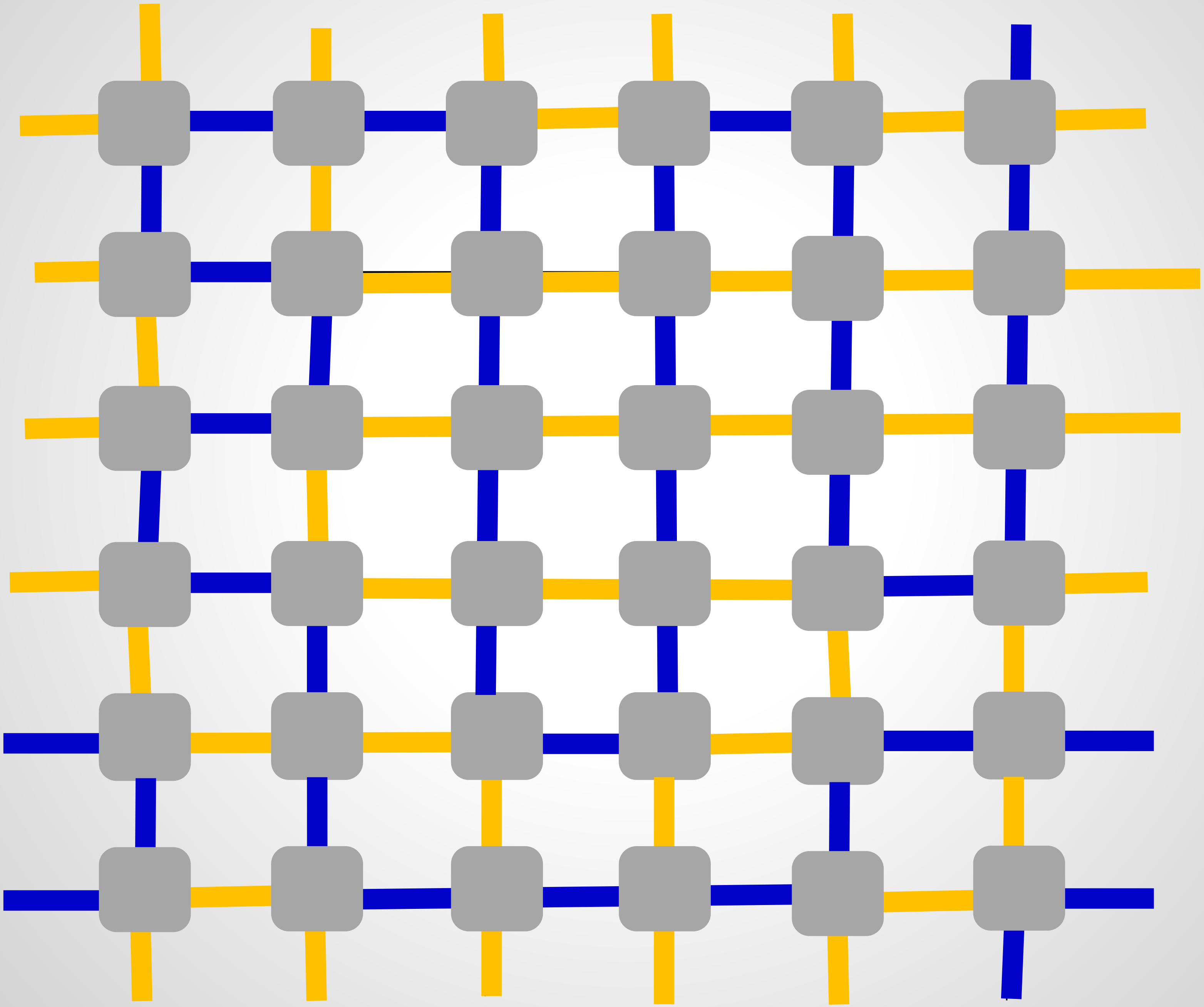
Example: 3-Resilient Routing Function for 2-dim Torus



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

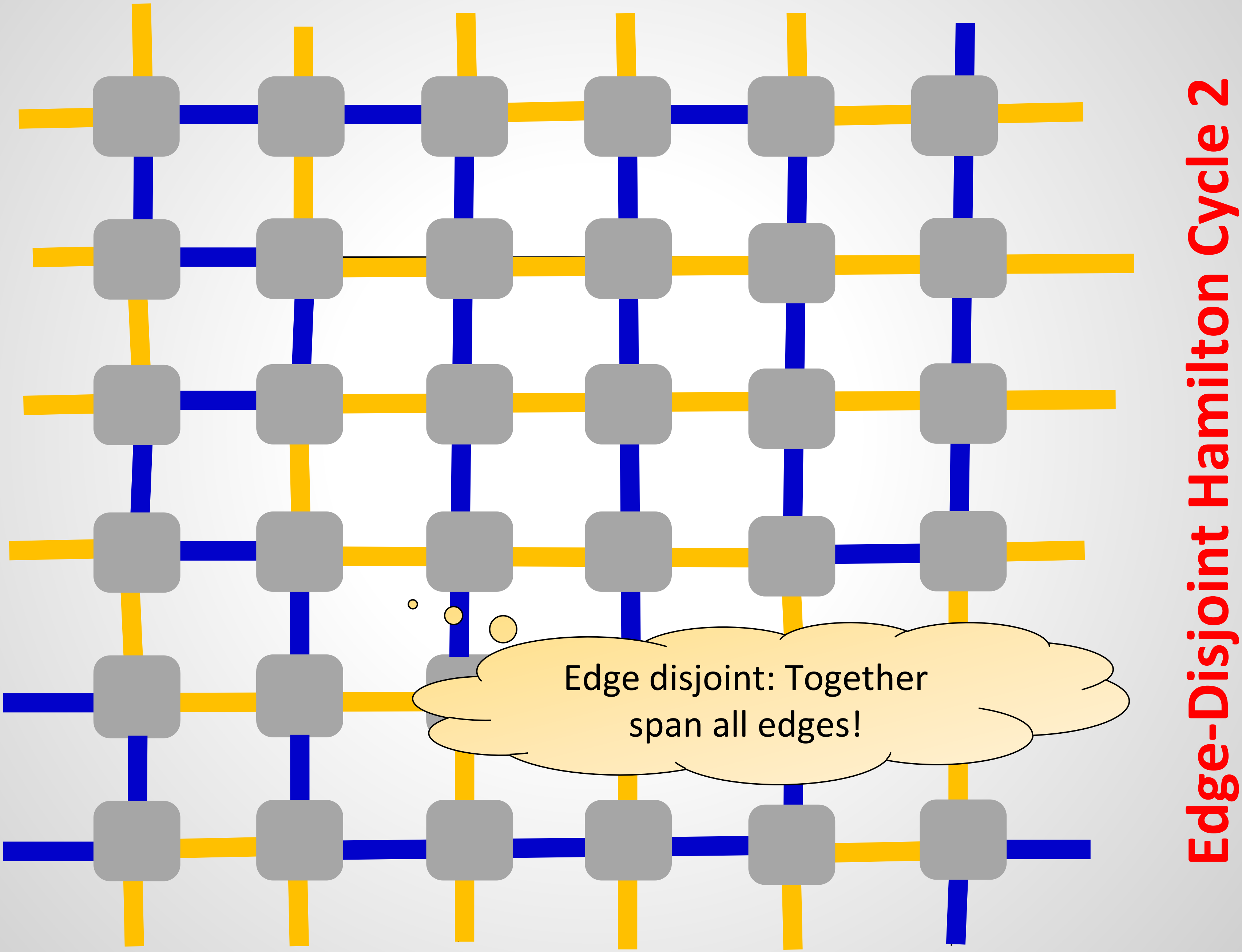


Example: 3-Resilient Routing Function for 2-dim Torus

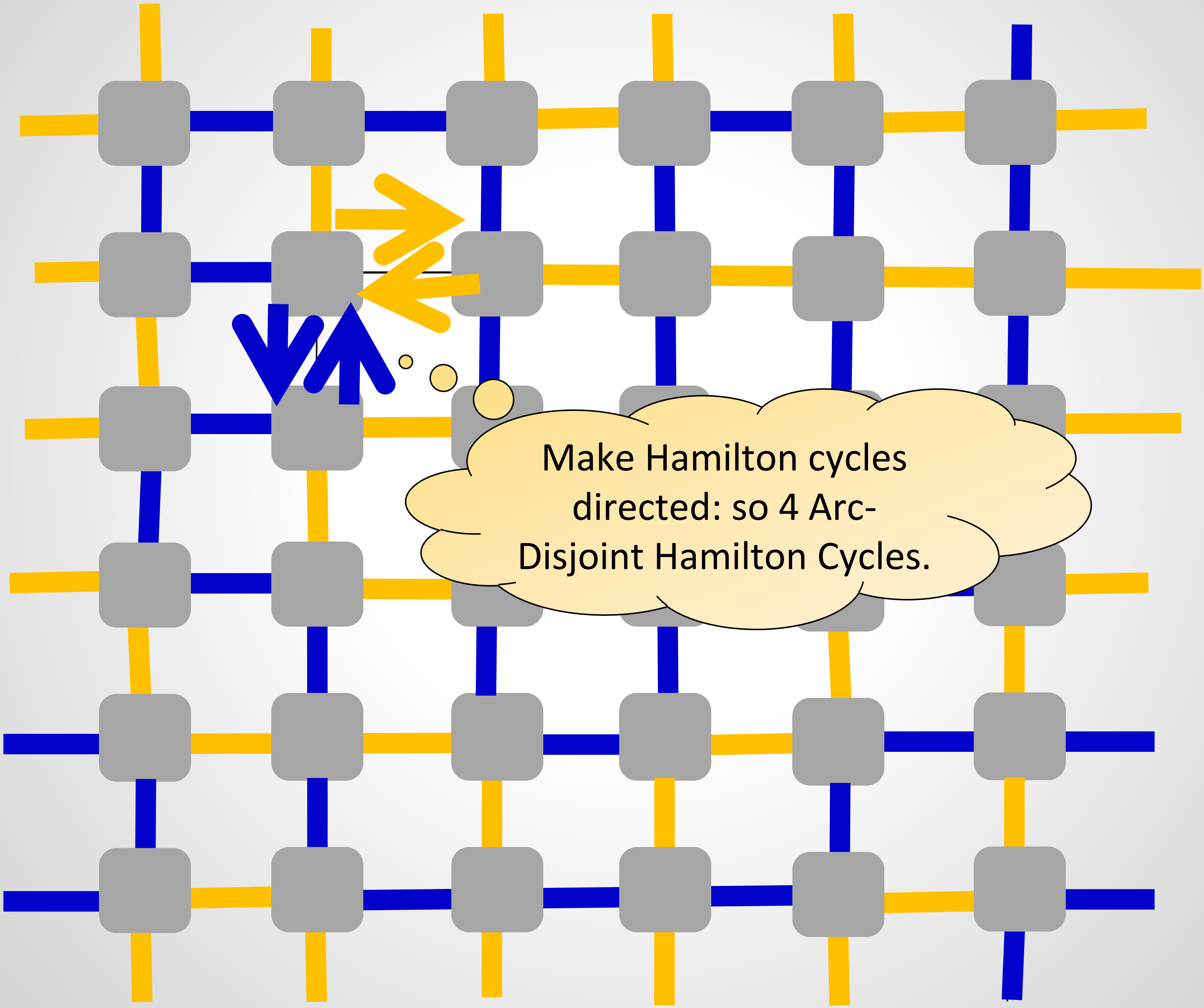


Edge-Disjoint Hamilton Cycle 2

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

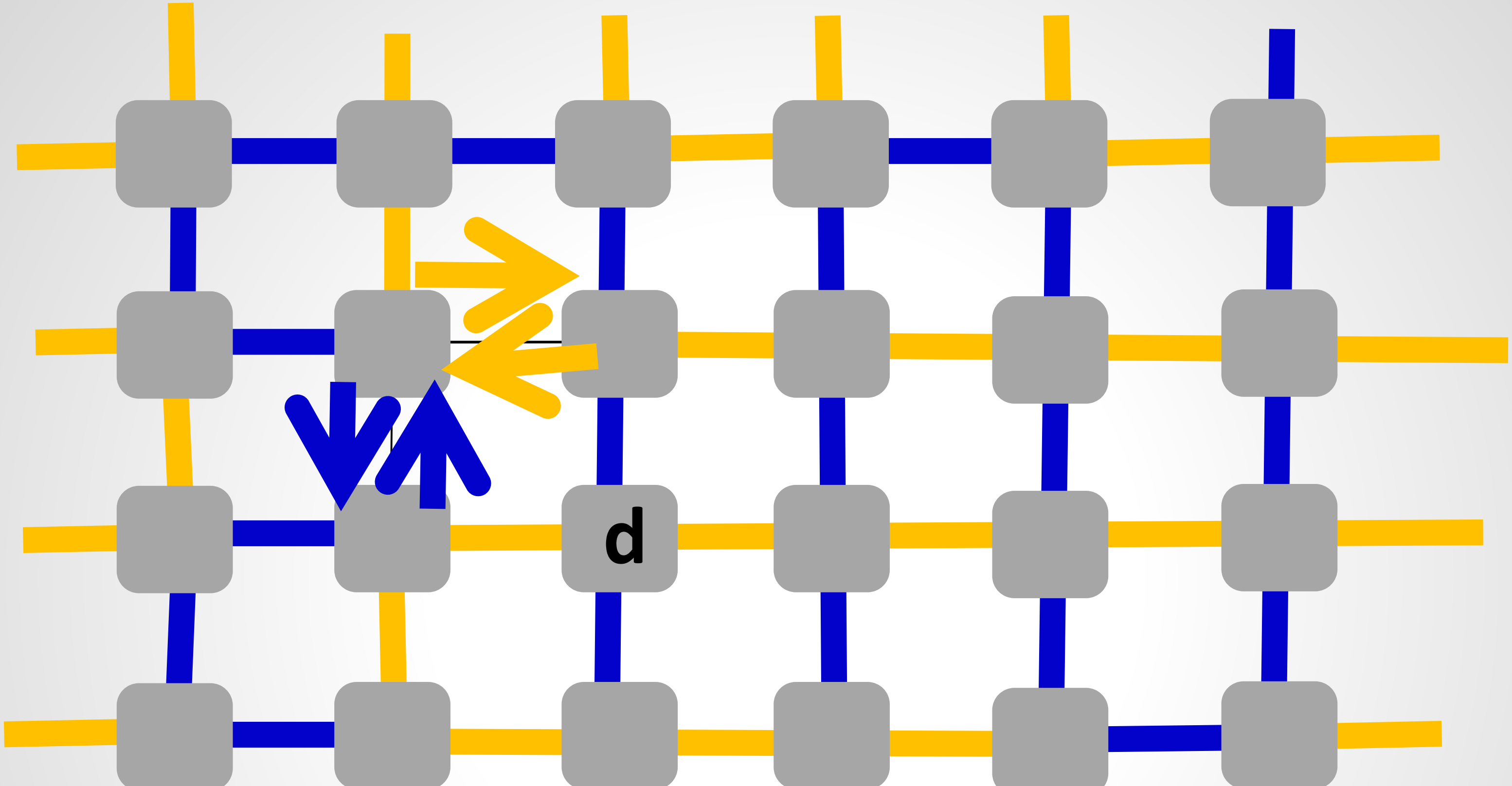


Example: 3-Resilient Routing Function for 2-dim Torus



4 Arc-Disjoint Arborescences

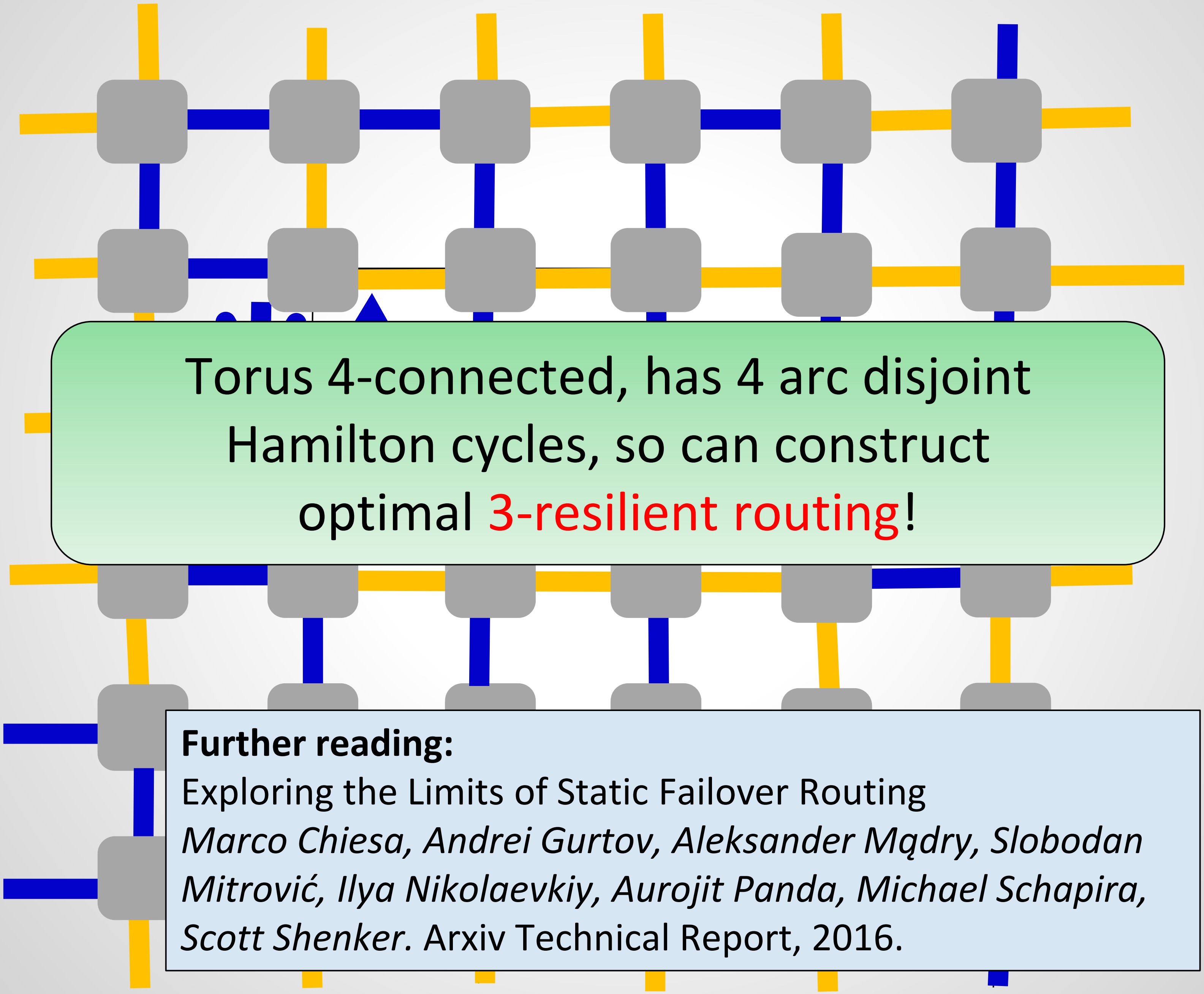
Example: 3-Resilient Routing Function for 2-dim Torus



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!

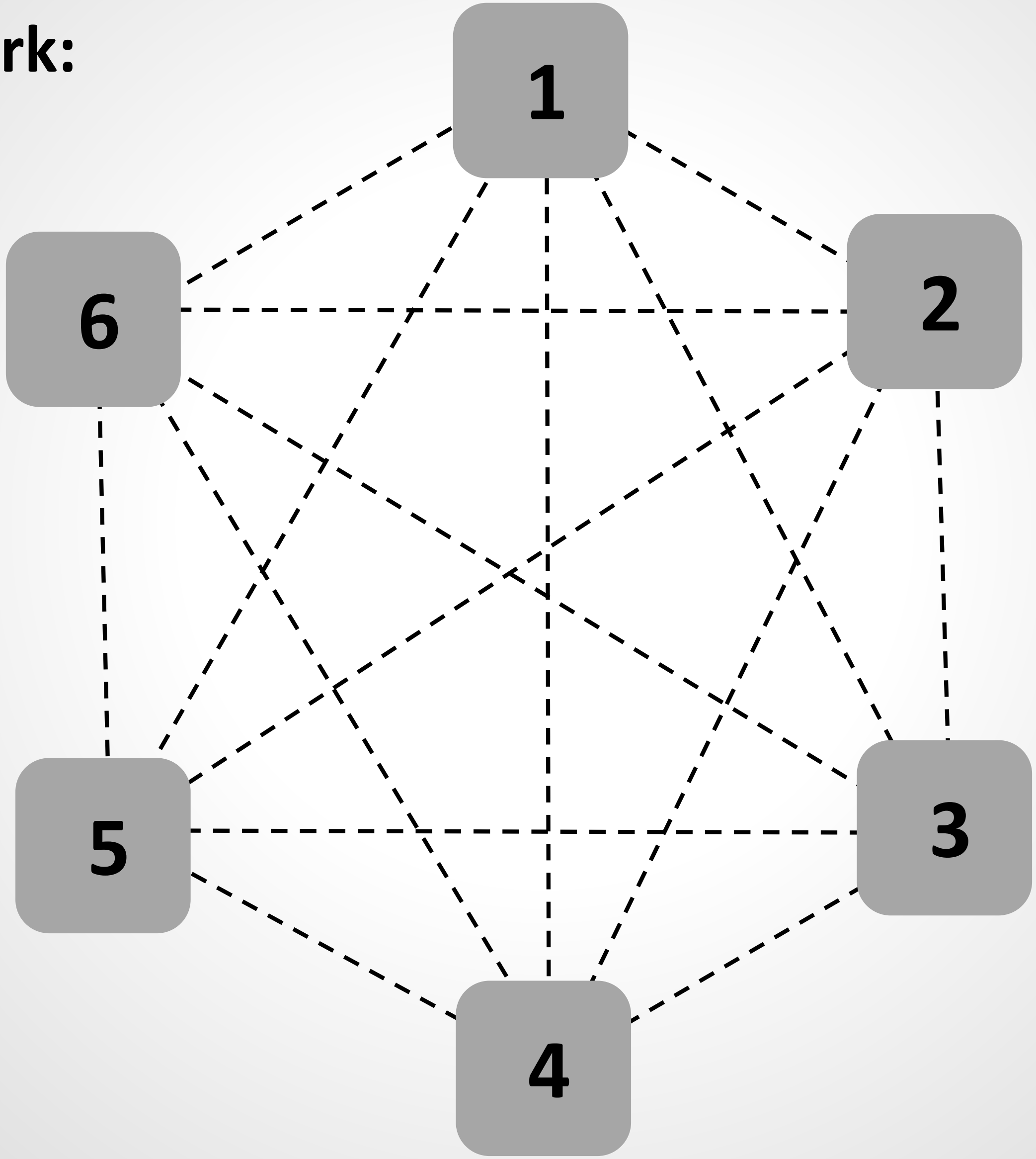
Further reading:
Exploring the Limits of Static Failover Routing
Marco Chiesa, Andrei Gurtov, Aleksander Mądry, Slobodan Mitrović, Ilya Nikolaevkiy, Aurojit Panda, Michael Schapira, Scott Shenker. Arxiv Technical Report, 2016.

4 Arc-Disjoint Arborescences

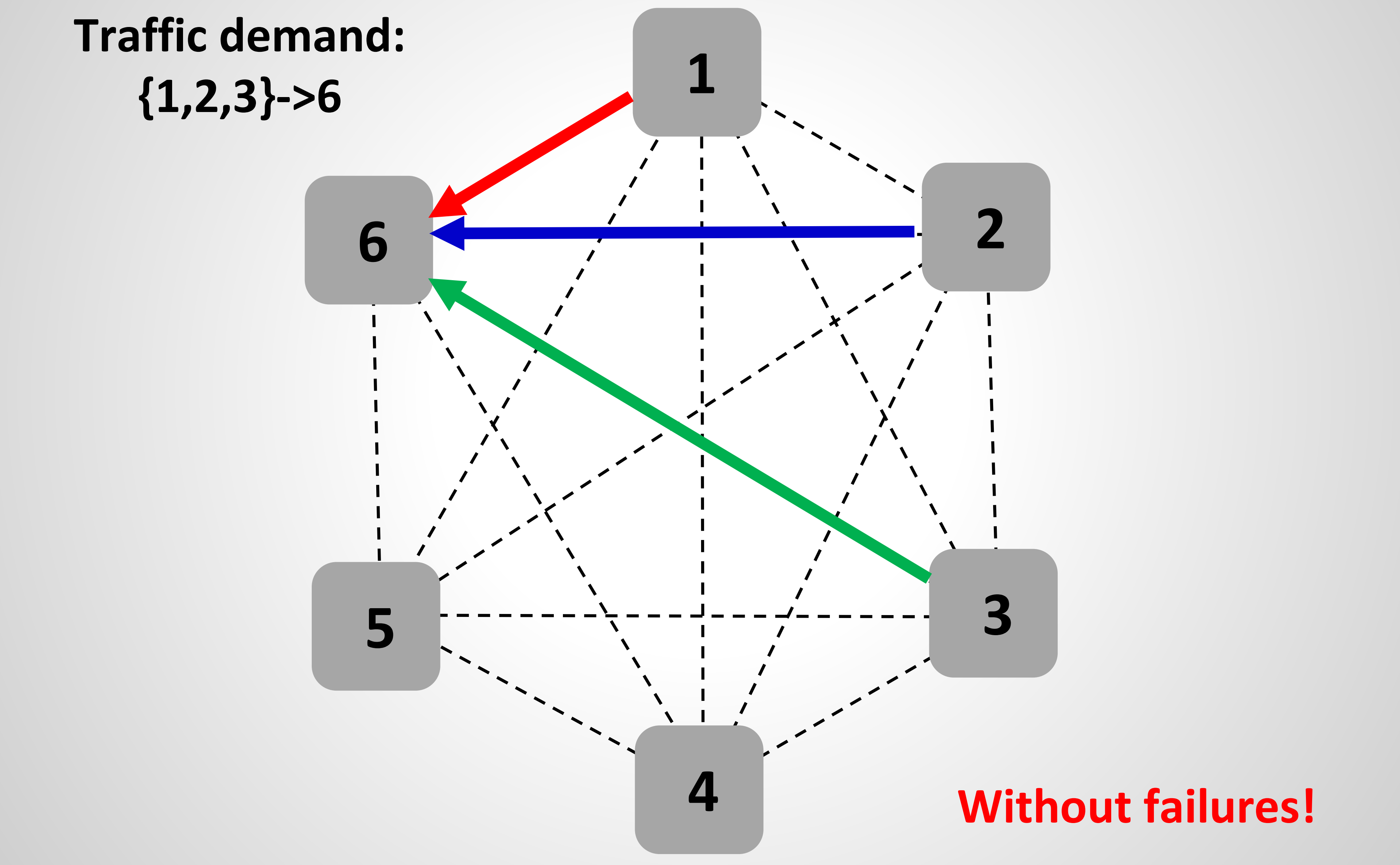
Load-Aware Local Fast Failover: Non-Trivial Already in the Clique!

Local Fast Failover with Load

The network:

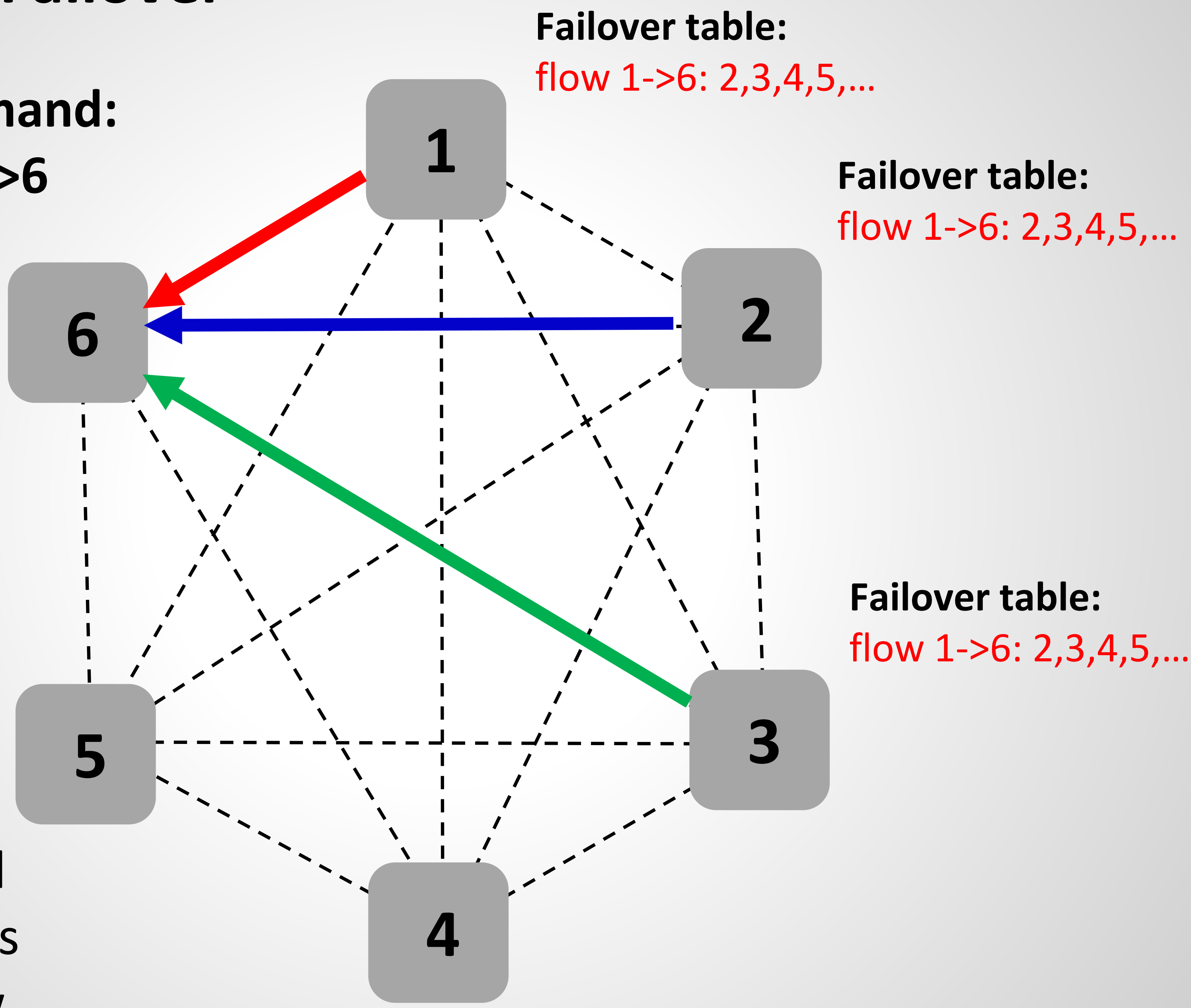


Local Fast Failover with Load



Local Fast Failover

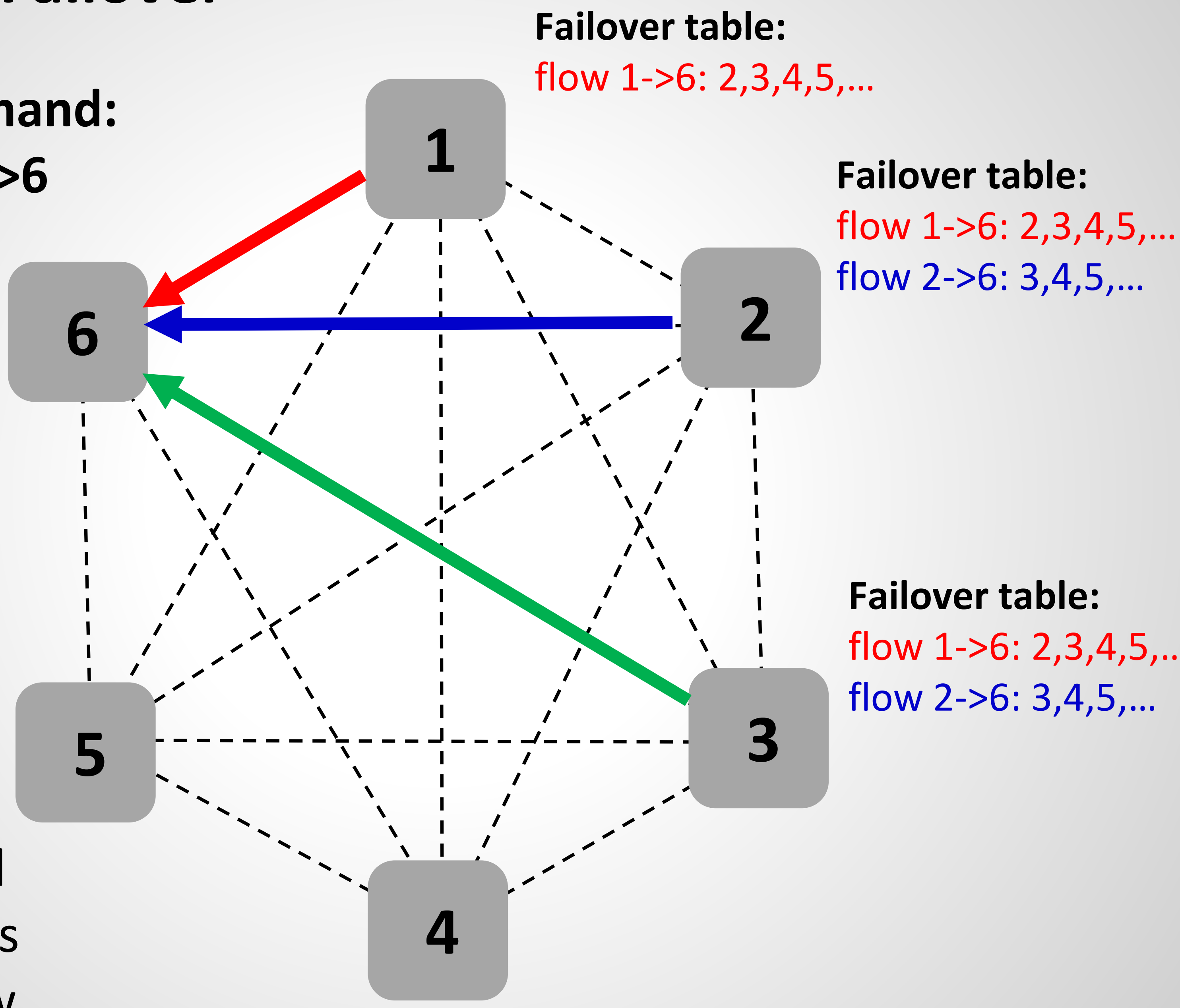
Traffic demand:
{1,2,3}->6



Preinstalled failover rules for red flow

Local Fast Failover

Traffic demand:
{1,2,3}->6



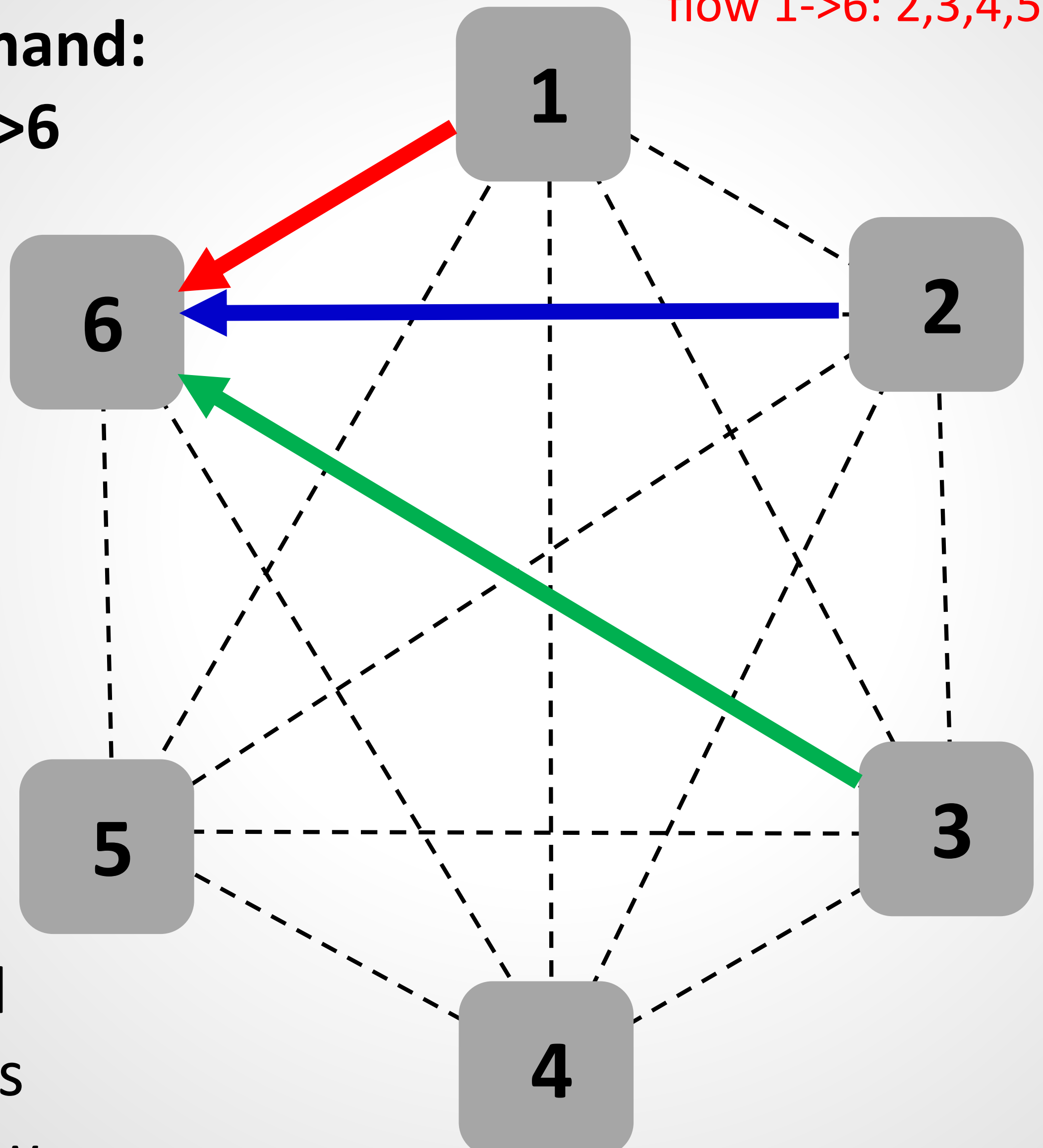
Preinstalled failover rules for blue flow

Local Fast Failover

Traffic demand:
{1,2,3}->6

Failover table:
flow 1->6: 2,3,4,5,...

Failover table:
flow 1->6: 2,3,4,5,...
flow 2->6: 3,4,5,...

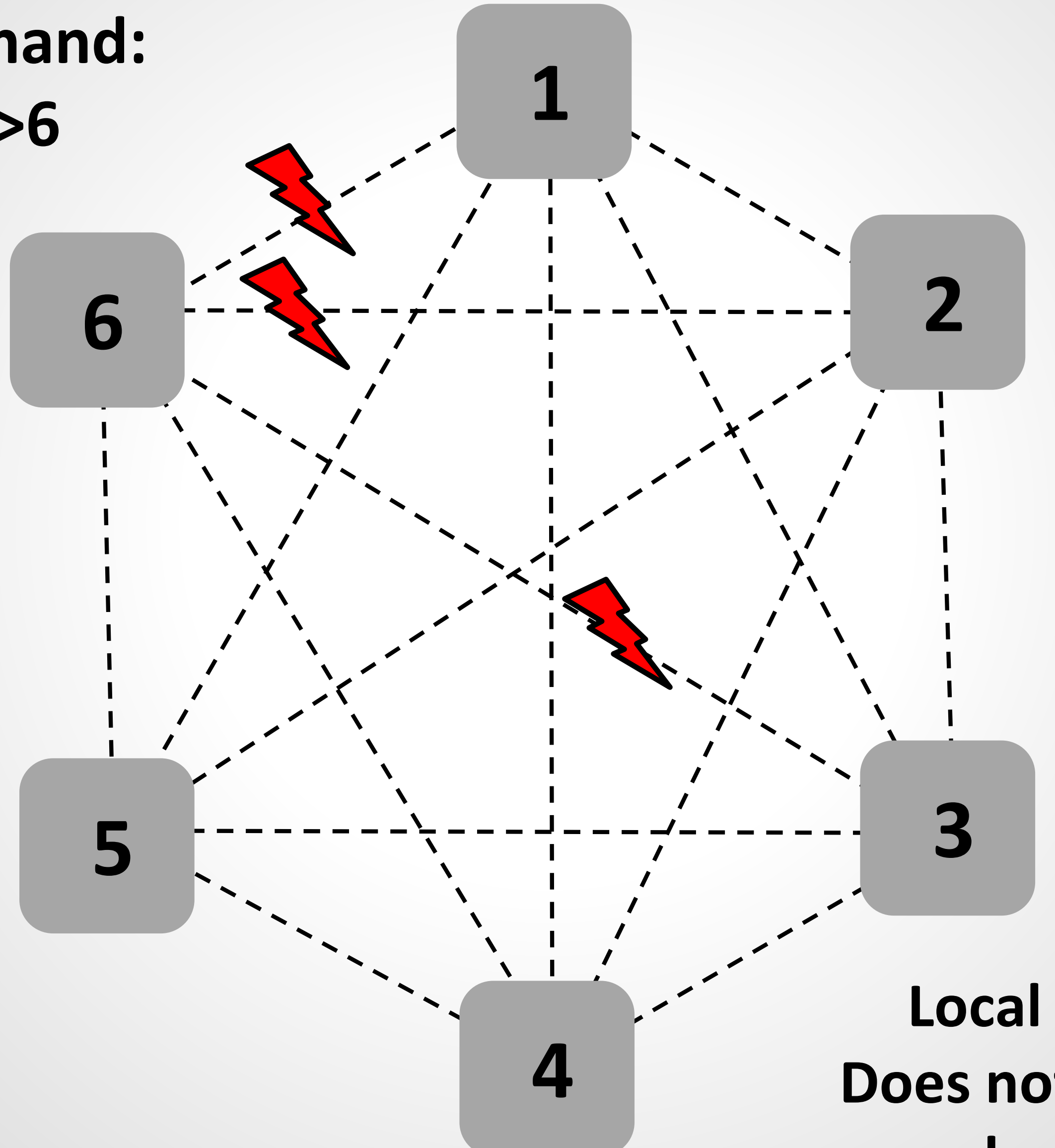


Failover table:
flow 1->6: 2,3,4,5,...
flow 2->6: 3,4,5,...
flow 3->6: 4,5,...

Preinstalled failover rules for green flow

Local Fast Failover

Traffic demand:
{1,2,3}->6

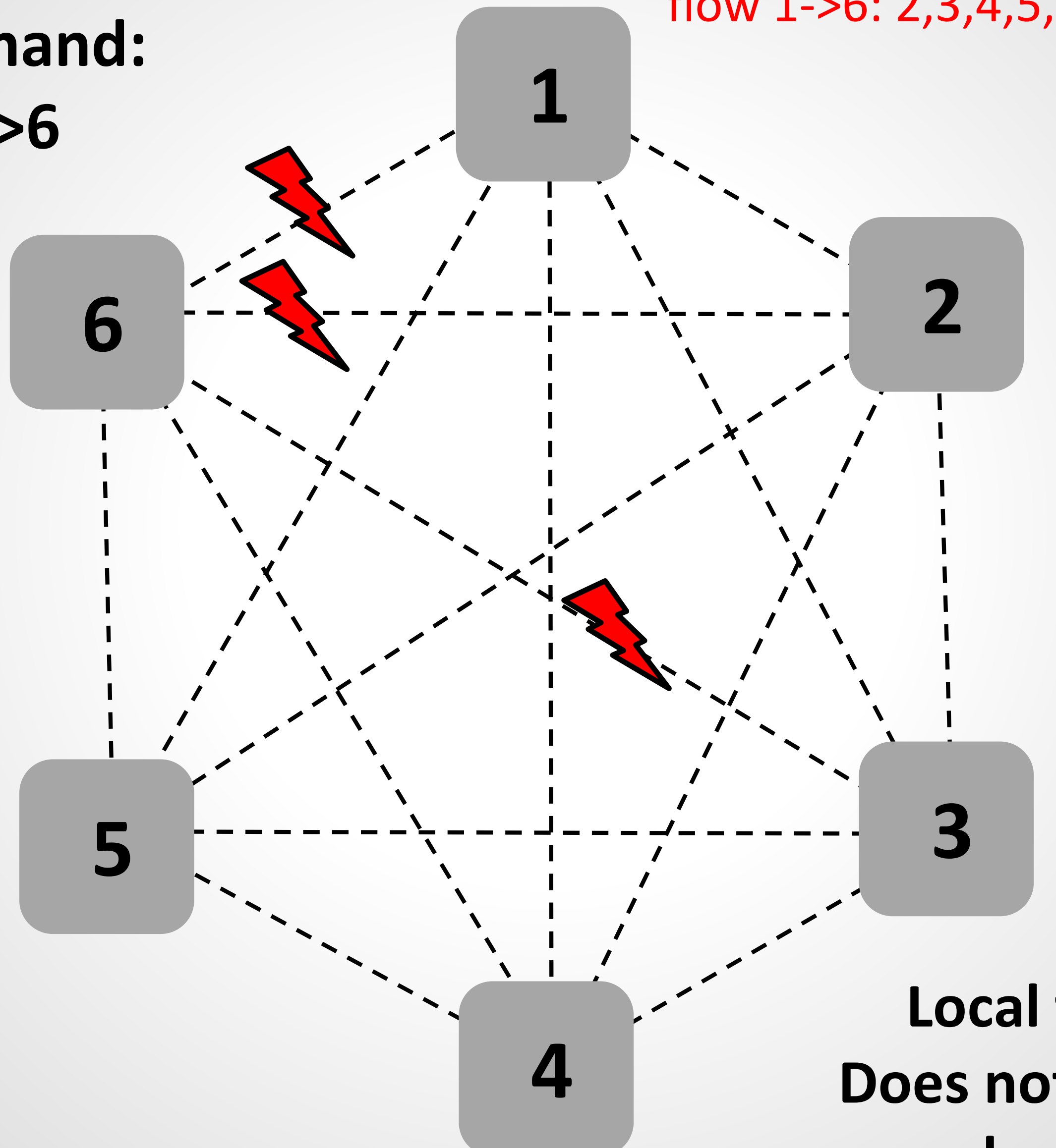


Local failover @1:
Does not know failures
downstream!

Local Fast Failover

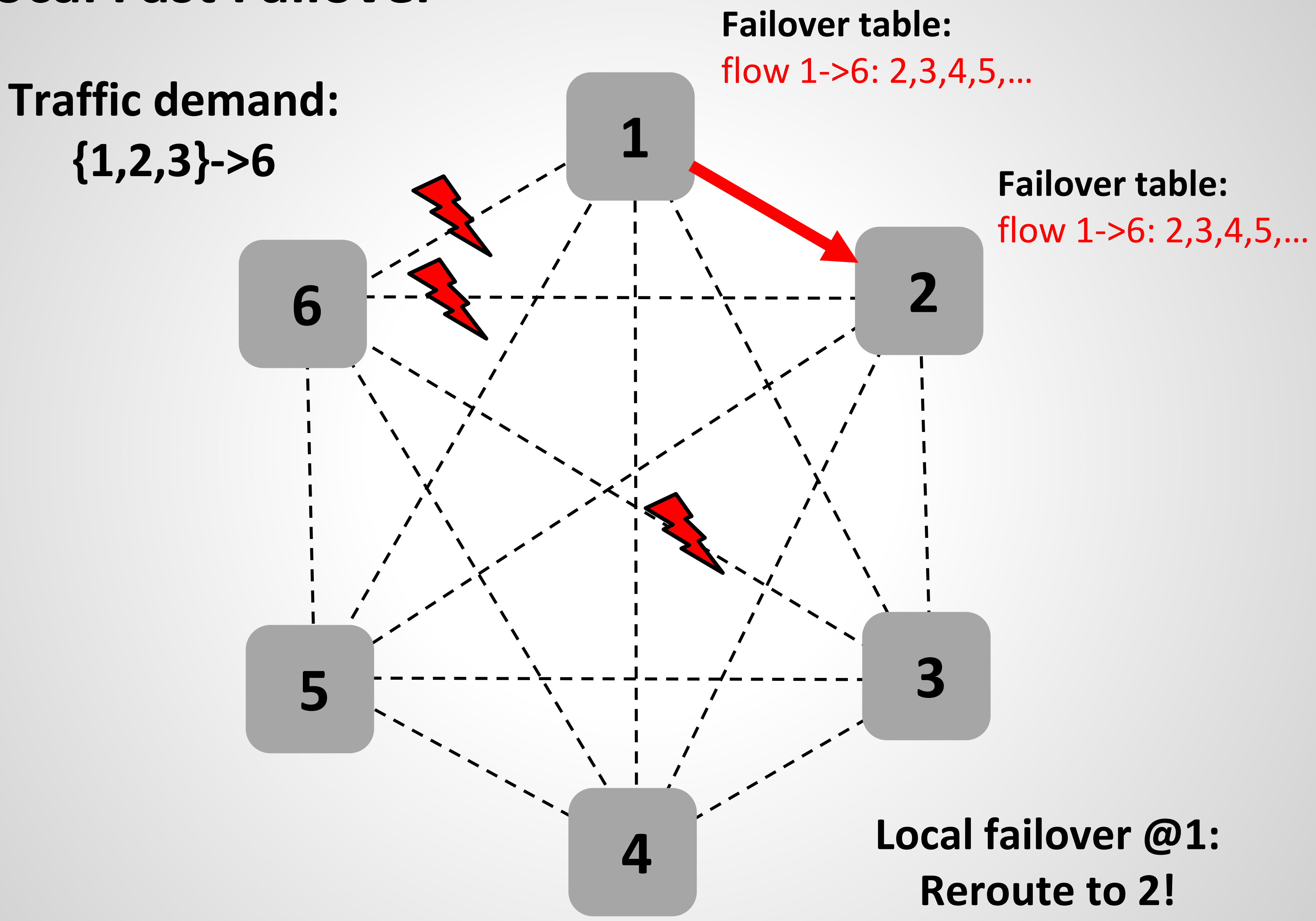
Traffic demand:
{1,2,3}->6

Failover table:
flow 1->6: 2,3,4,5,...

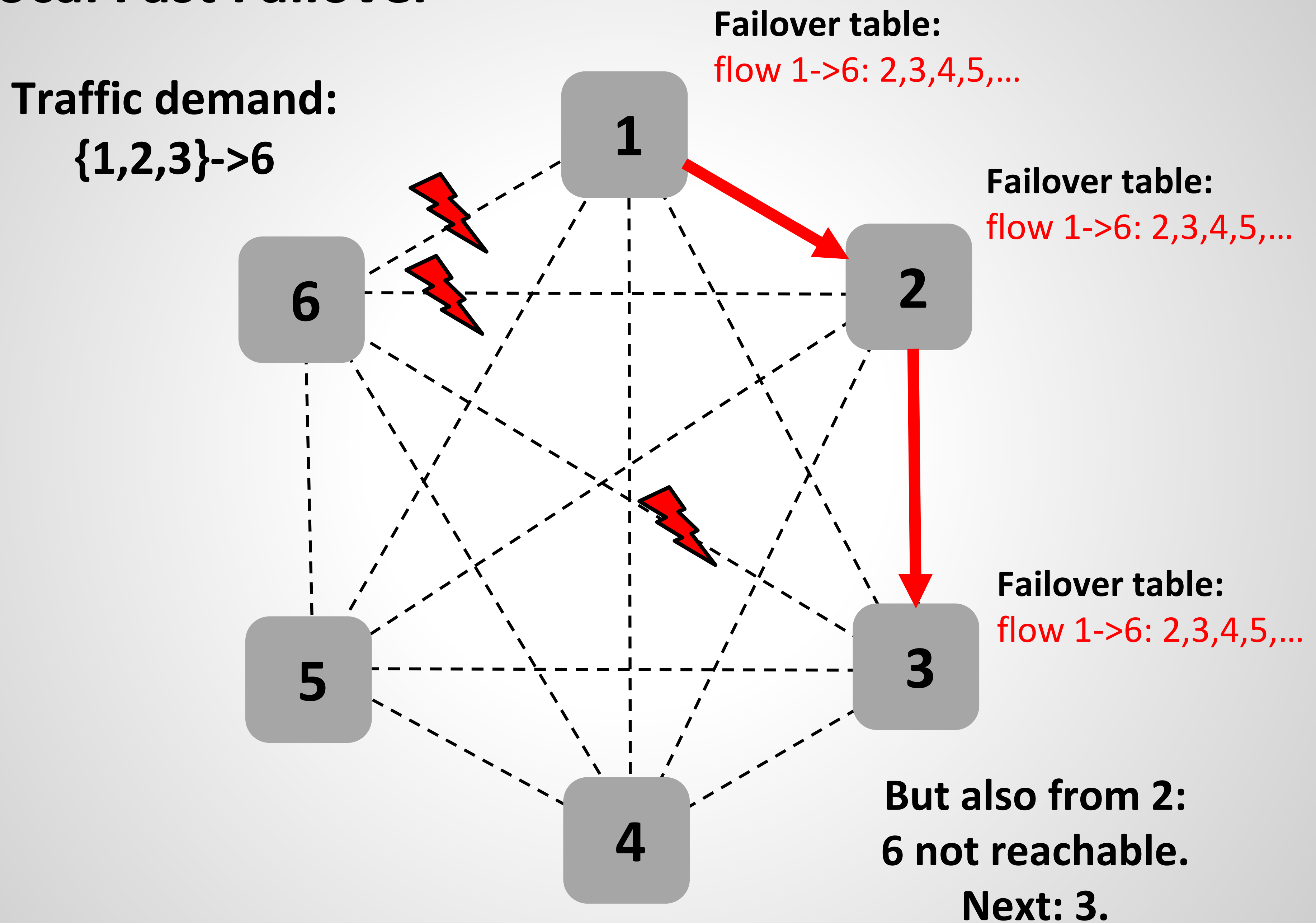


Local failover @1:
Does not know failures downstream!

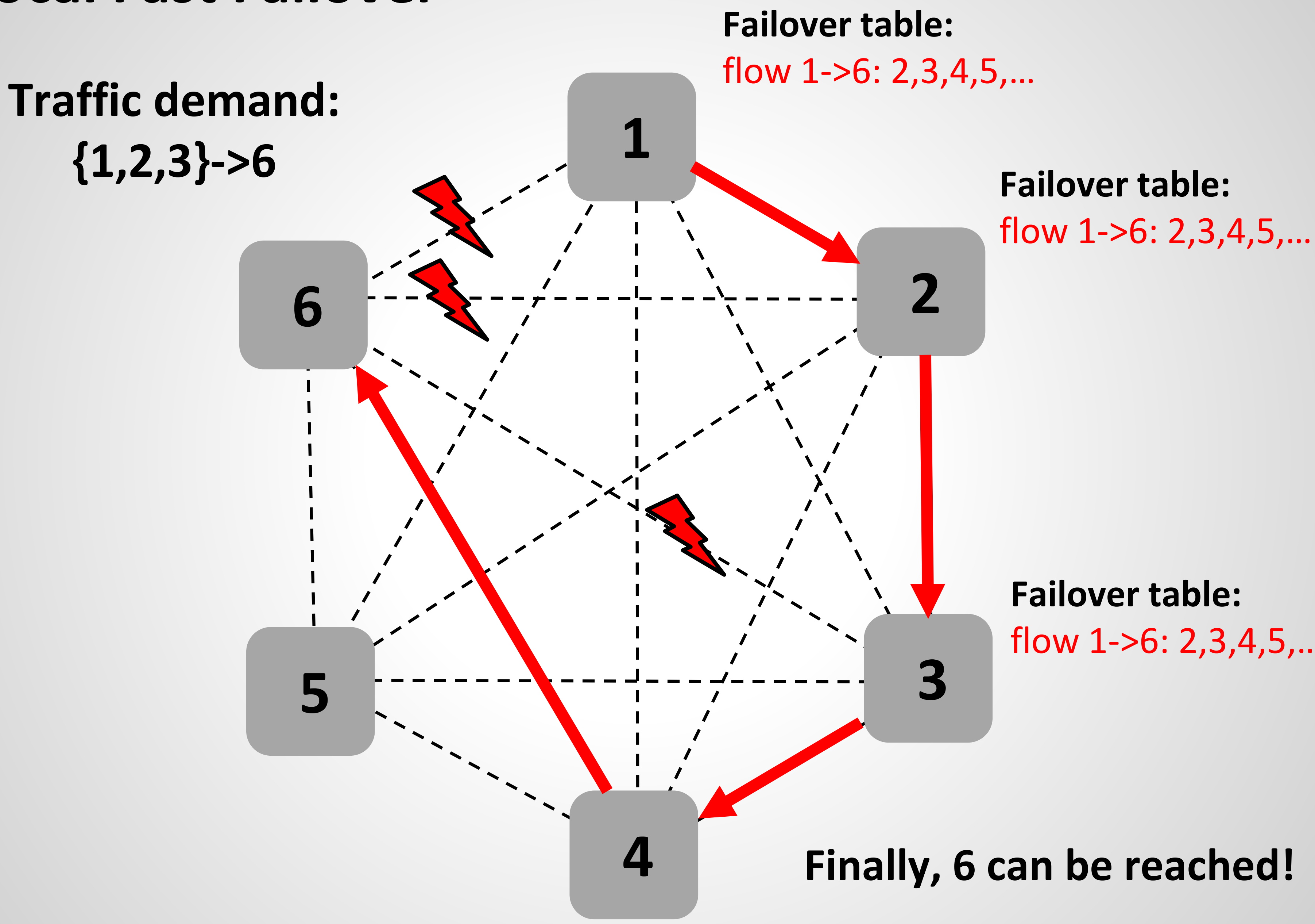
Local Fast Failover



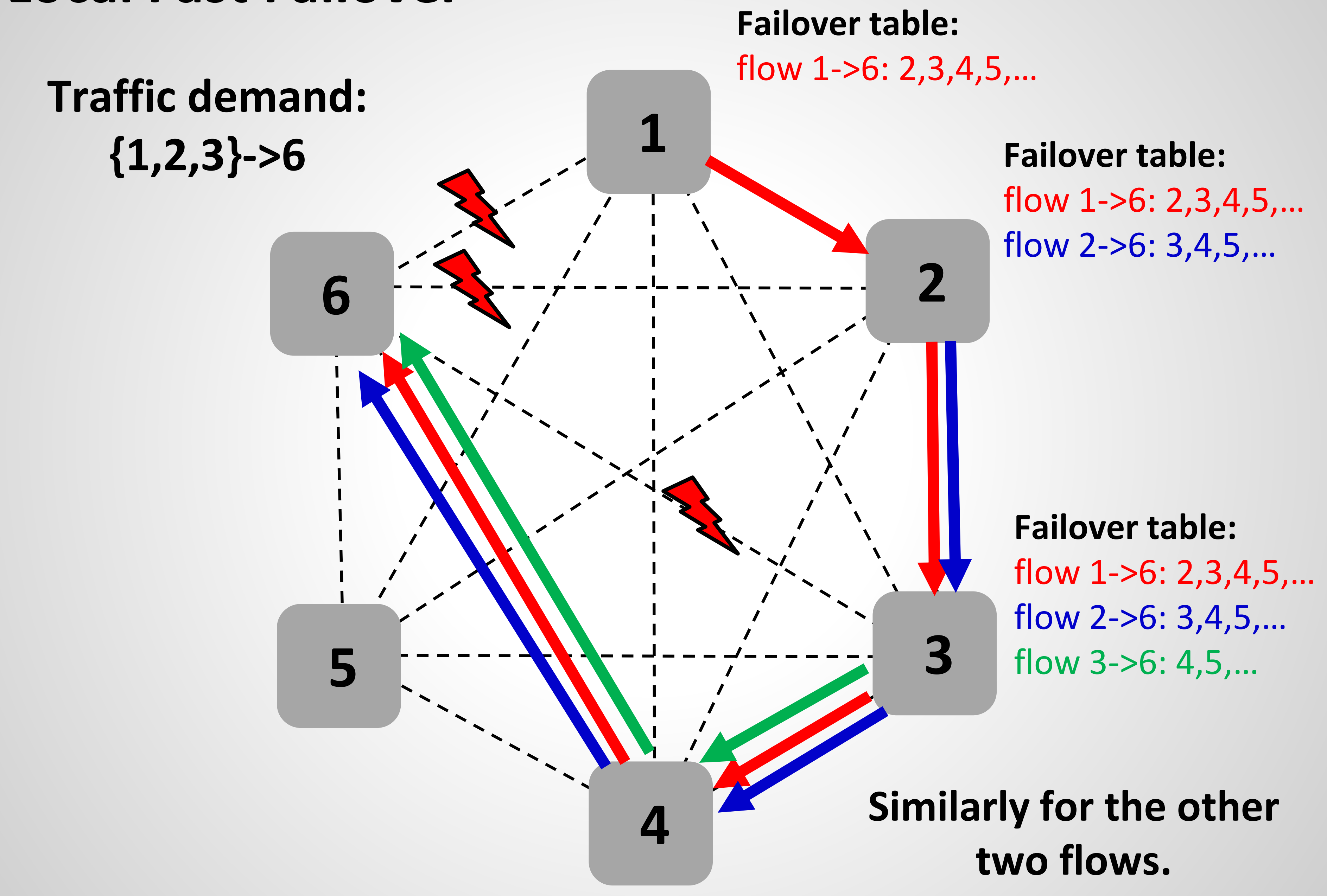
Local Fast Failover



Local Fast Failover

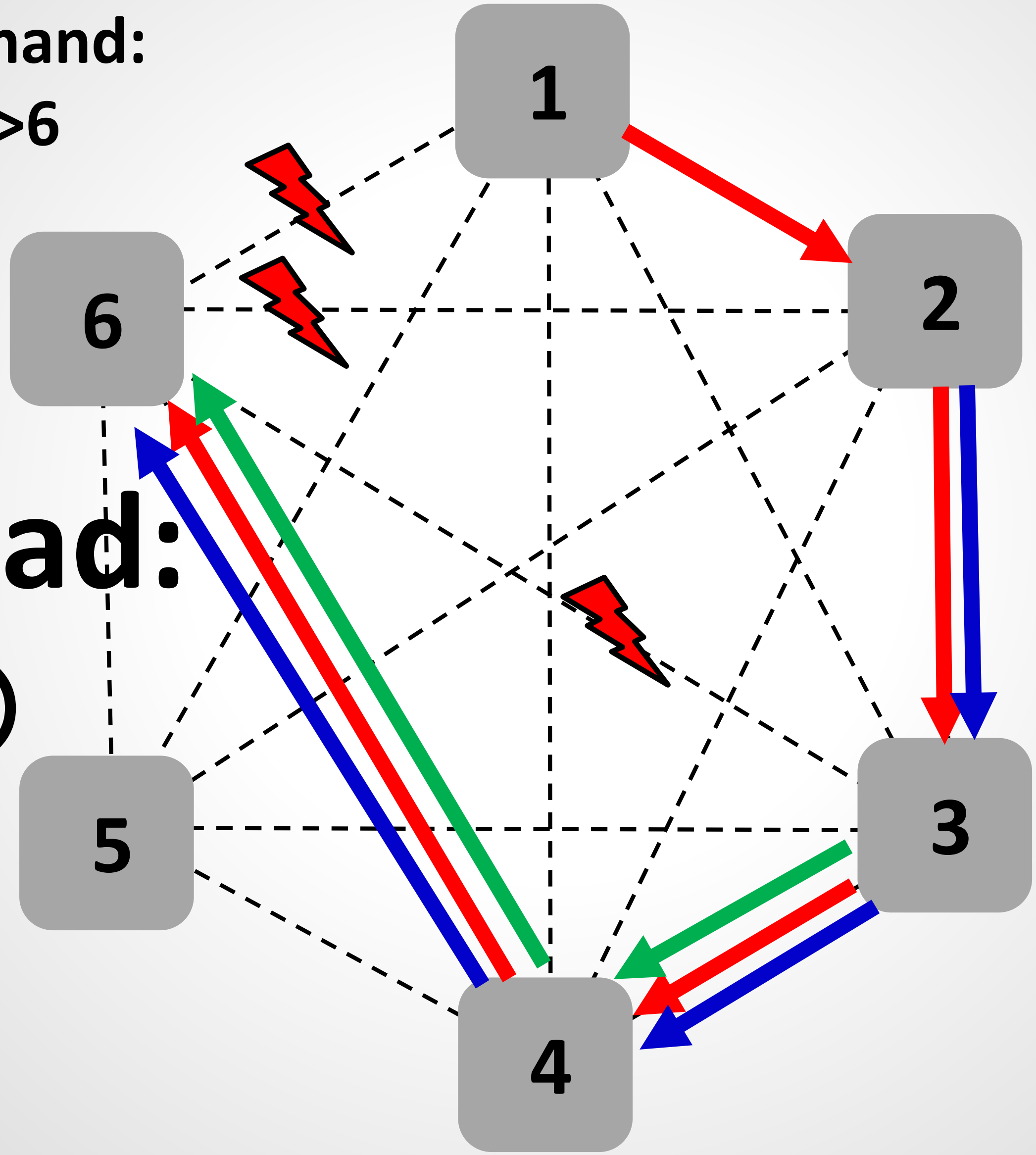


Local Fast Failover



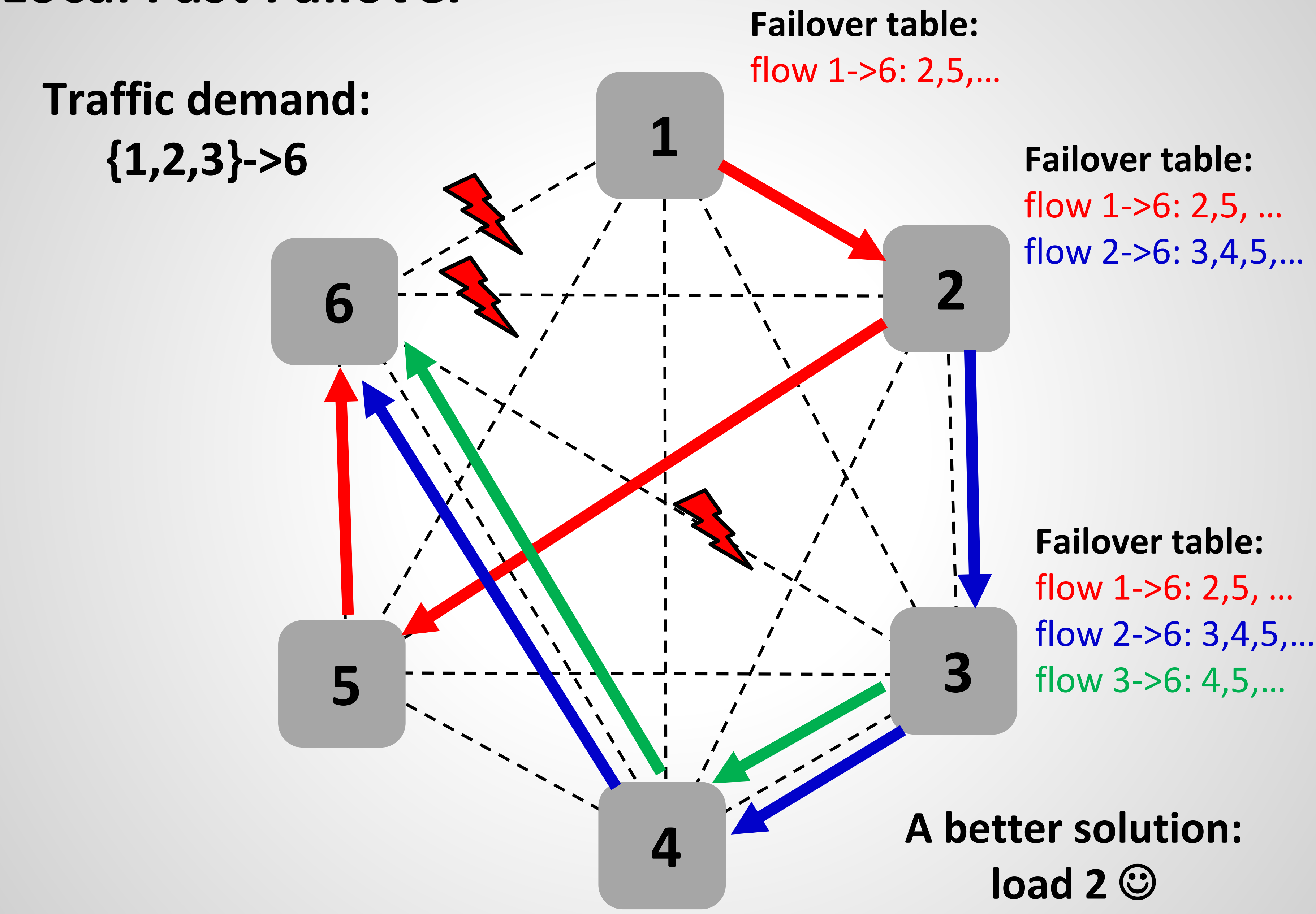
Local Fast Failover

Traffic demand:
{1,2,3}->6



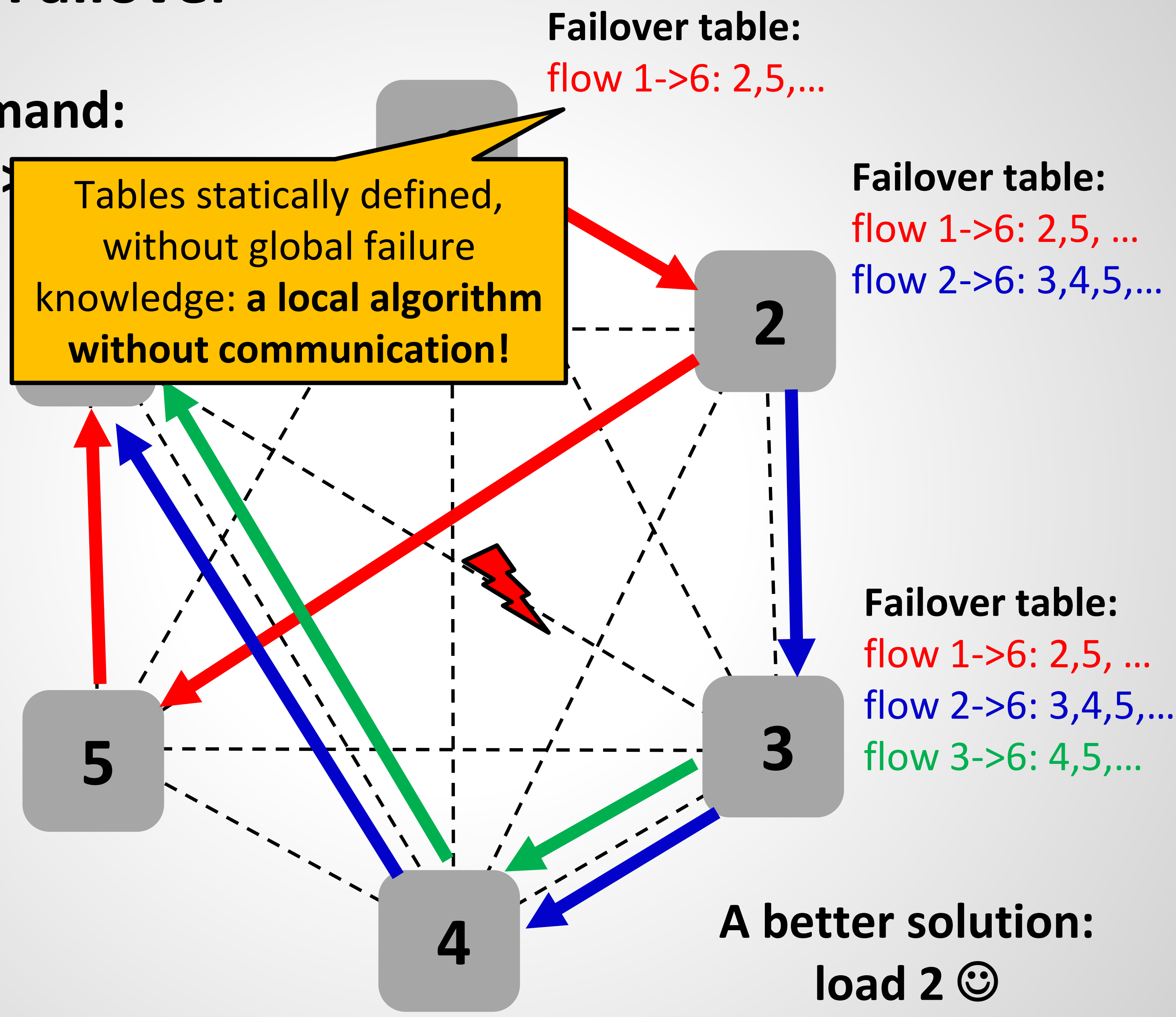
Max load:
3 ☹

Local Fast Failover



Local Fast Failover

Traffic demand:
{1,2,3}-›



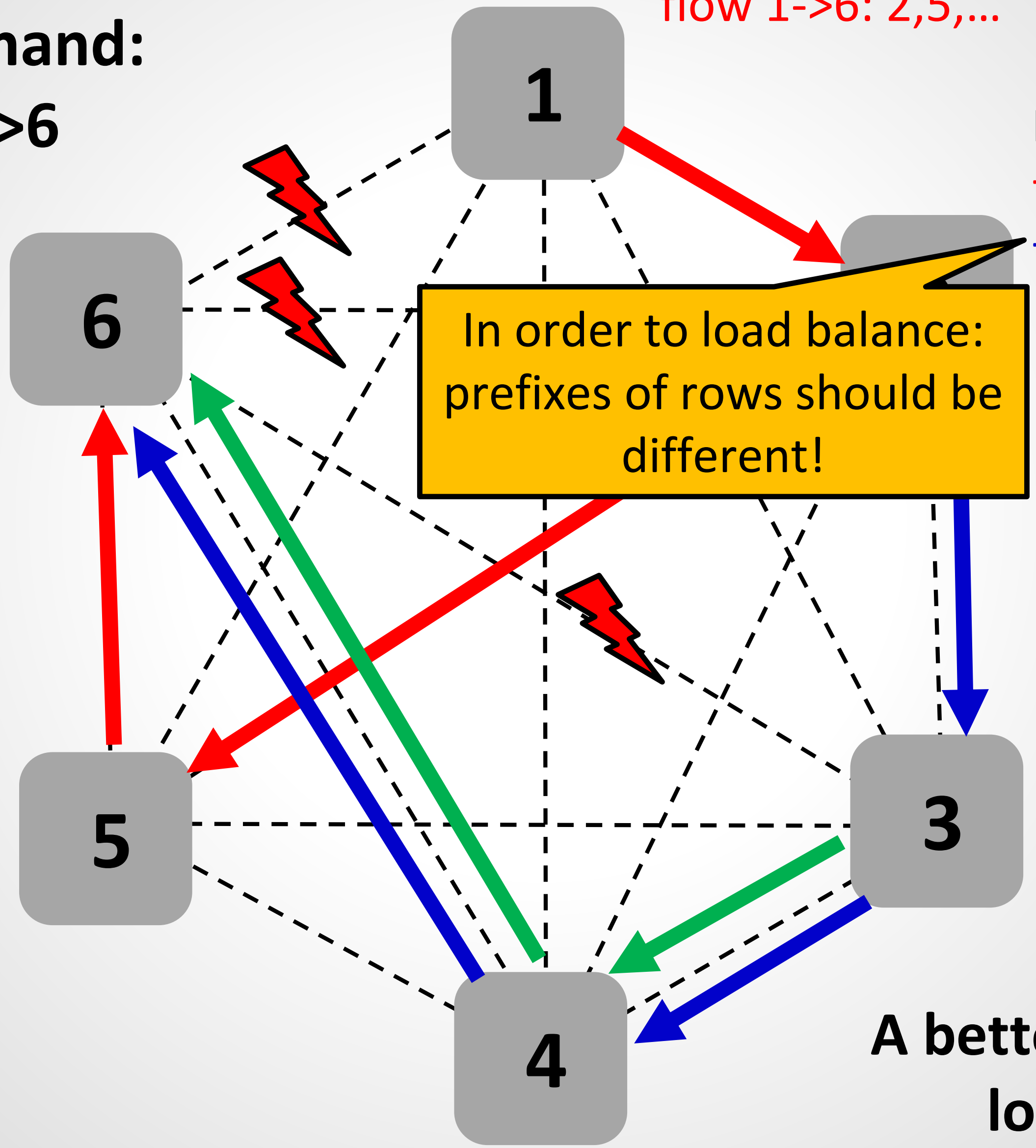
A better solution:
load 2 ☺

Local Fast Failover

Traffic demand:
{1,2,3}->6

Failover table:
flow 1->6: 2,5,...

Failover table:
flow 1->6: 2,5, ...
flow 2->6: 3,4,5,...



Failover table:
flow 1->6: 2,5, ...
flow 2->6: 3,4,5,...
flow 3->6: 4,5,...

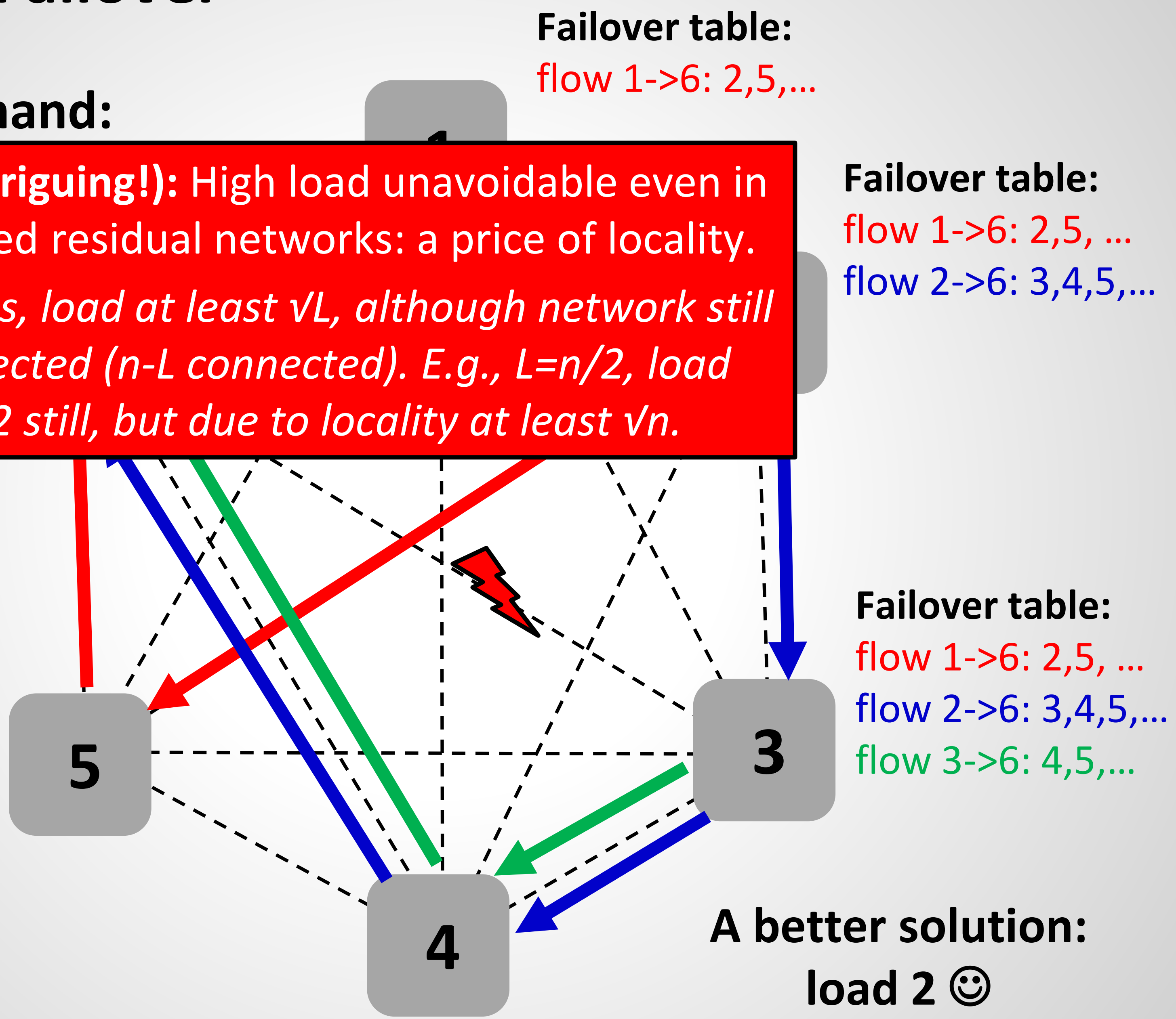
A better solution:
load 2 ☺

Local Fast Failover

Traffic demand:

Bad news (intriguing!): High load unavoidable even in well-connected residual networks: a price of locality.

Given L failures, load at least √L, although network still highly connected (n-L connected). E.g., L=n/2, load could be 2 still, but due to locality at least √n.



A better solution: load 2 ☺

Local Fast Failover

Traffic demand:

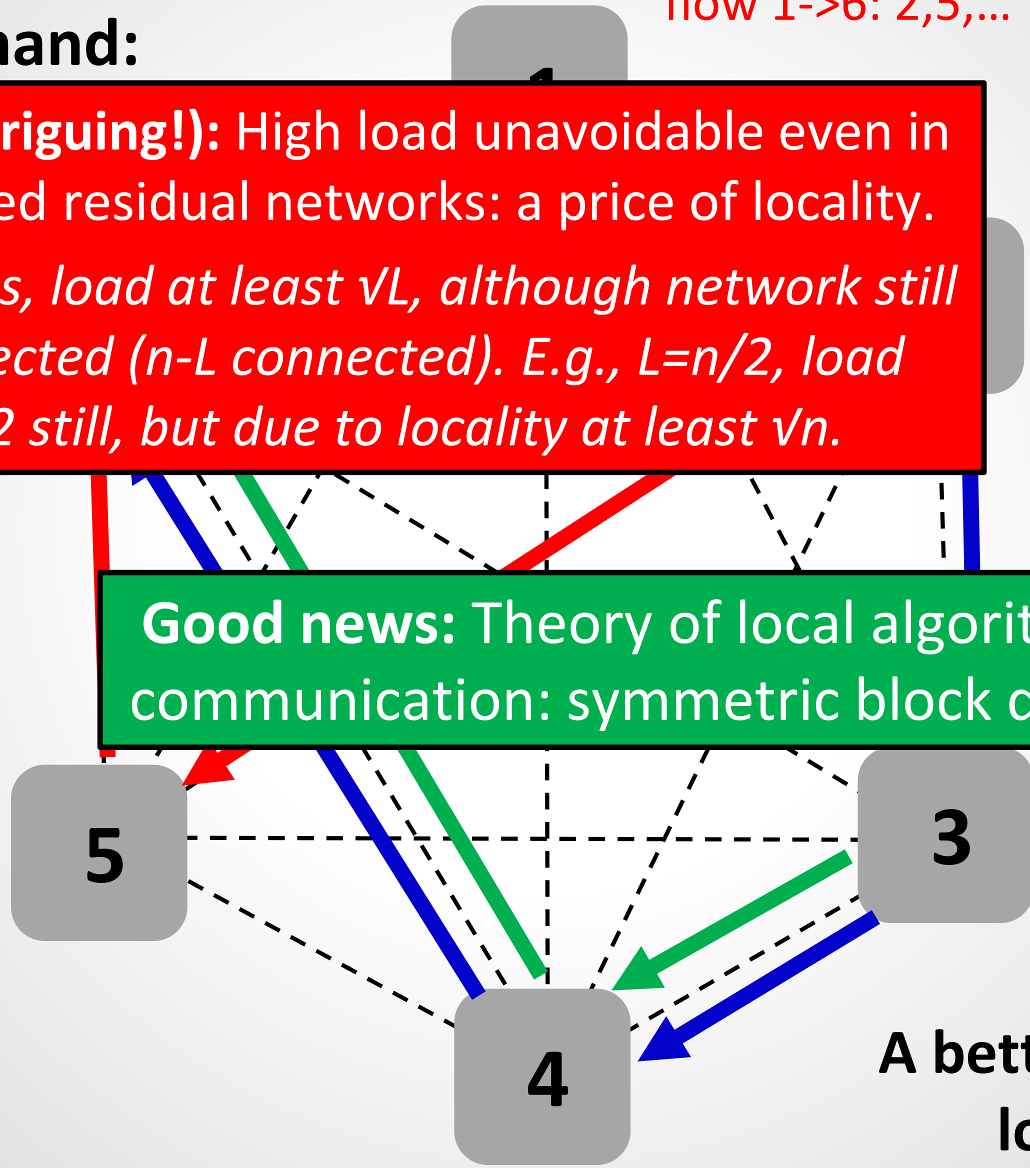
Failover table:
flow 1->6: 2,5,...

Failover table:
flow 1->6: 2,5, ...
flow 2->6: 3,4,5,...

Bad news (intriguing!): High load unavoidable even in well-connected residual networks: a price of locality.

Given L failures, load at least √L, although network still highly connected (n-L connected). E.g., L=n/2, load could be 2 still, but due to locality at least √n.

Good news: Theory of local algorithms without communication: symmetric block design theory.

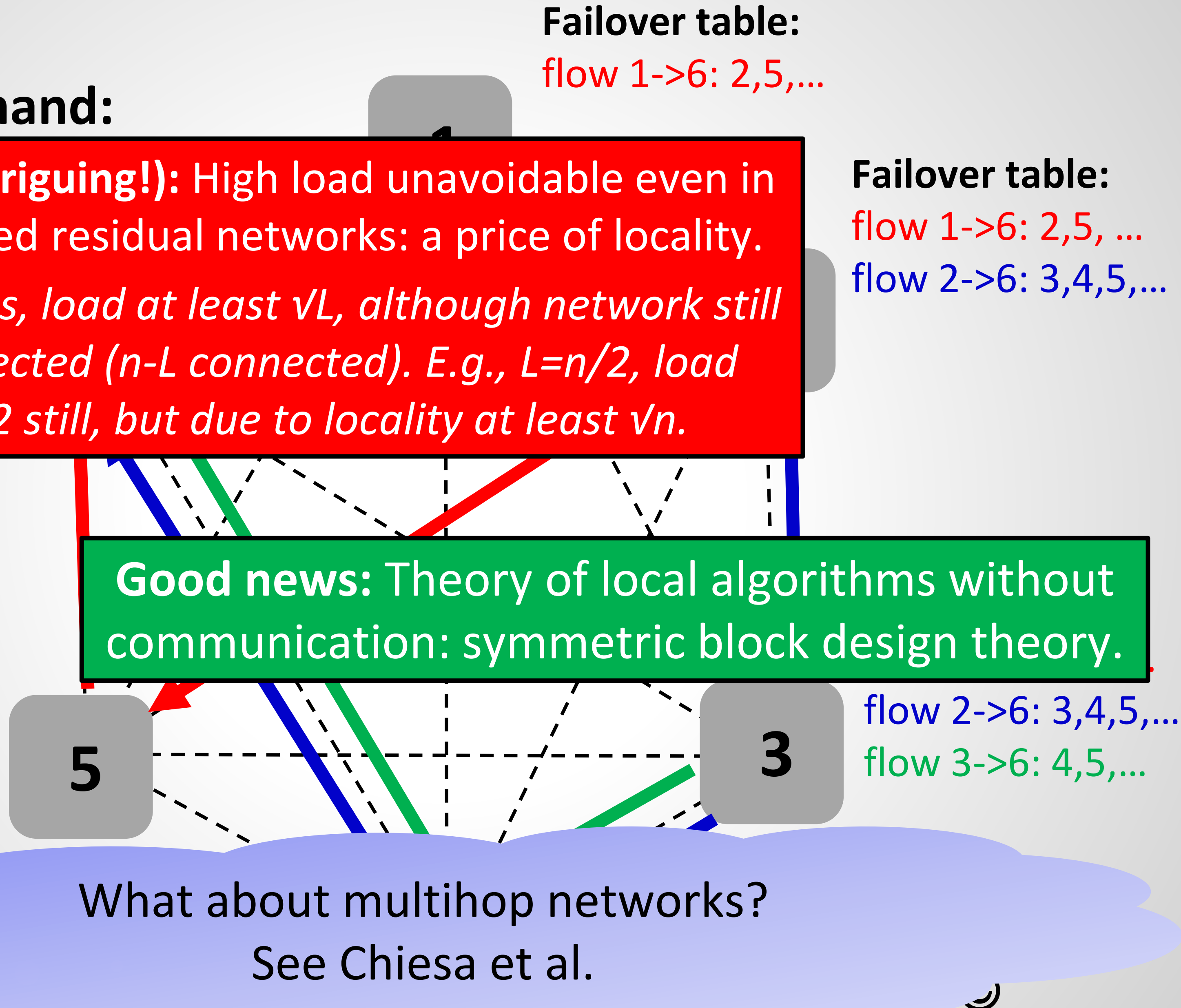


flow 2->6: 3,4,5,...
flow 3->6: 4,5,...

A better solution: load 2 ☺

Local Fast Failover

Traffic demand:



Bad news (intriguing!): High load unavoidable even in well-connected residual networks: a price of locality.

Given L failures, load at least √L, although network still highly connected (n-L connected). E.g., L=n/2, load could be 2 still, but due to locality at least √n.

Good news: Theory of local algorithms without communication: symmetric block design theory.

What about multihop networks?
See Chiesa et al.

Local Fast Failover

Failover table:
flow 1->6: 2,5,...

Traffic demand:

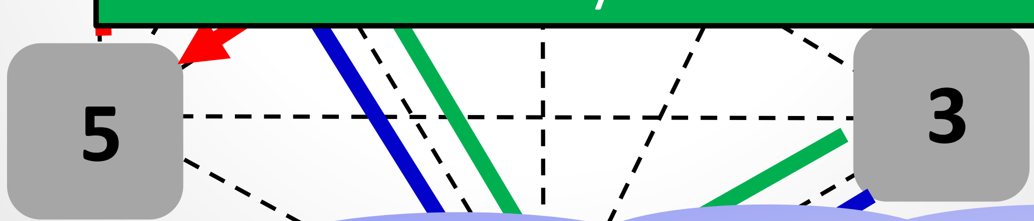
Bad news (intriguing!): High load unavoidable even in well-connected residual networks: a price of locality.

Given L failures, load at least $\sqrt{L}$, although network still
hi

Failover table:
flow 1->6: 2,5, ...
flow 2->6: 3,4,5,...

Further reading:
[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.

out
communication: symmetric block design theory.



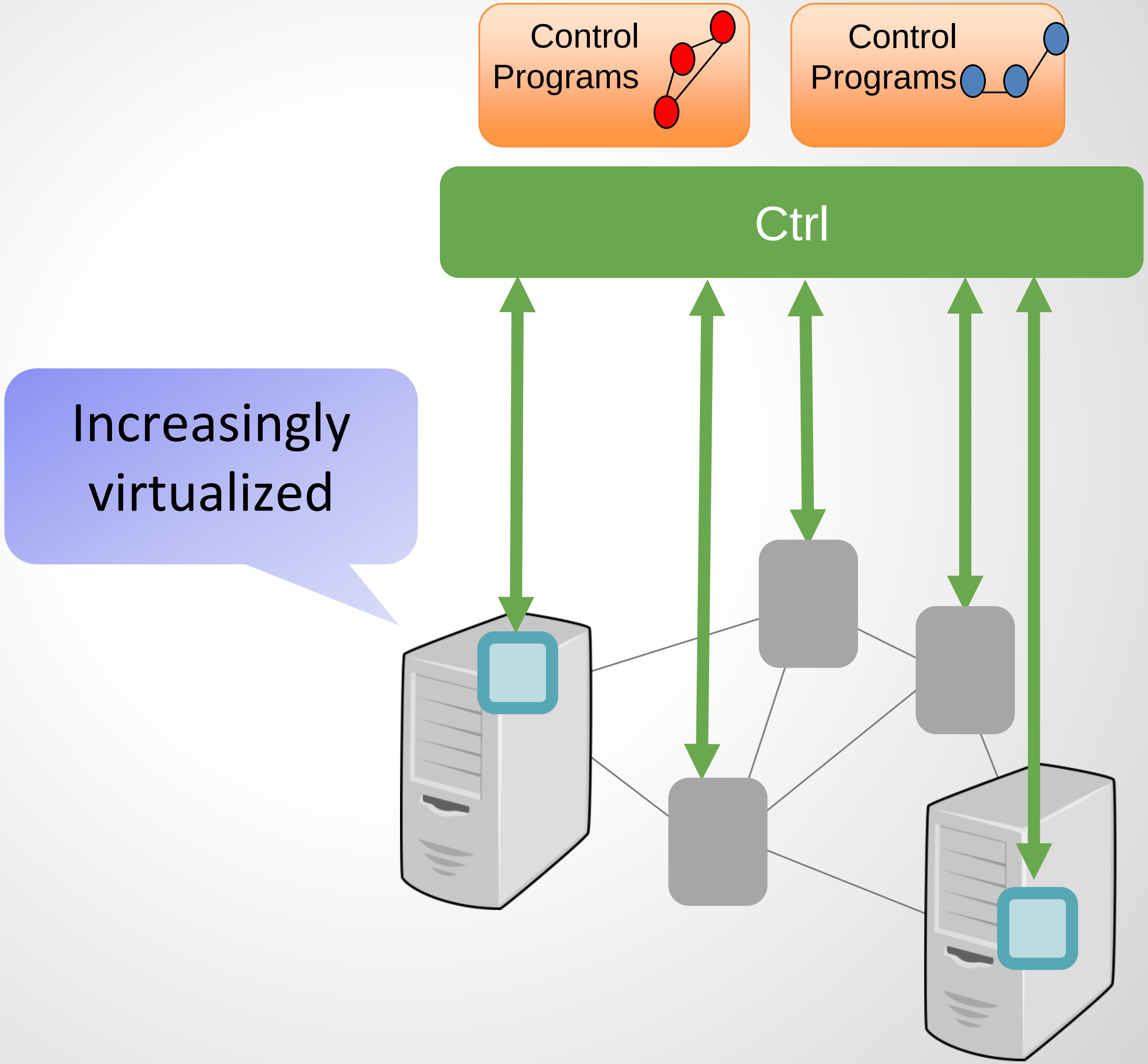
flow 2->6: 3,4,5,...
flow 3->6: 4,5,...

What about multihop networks?
See Chiesa et al.

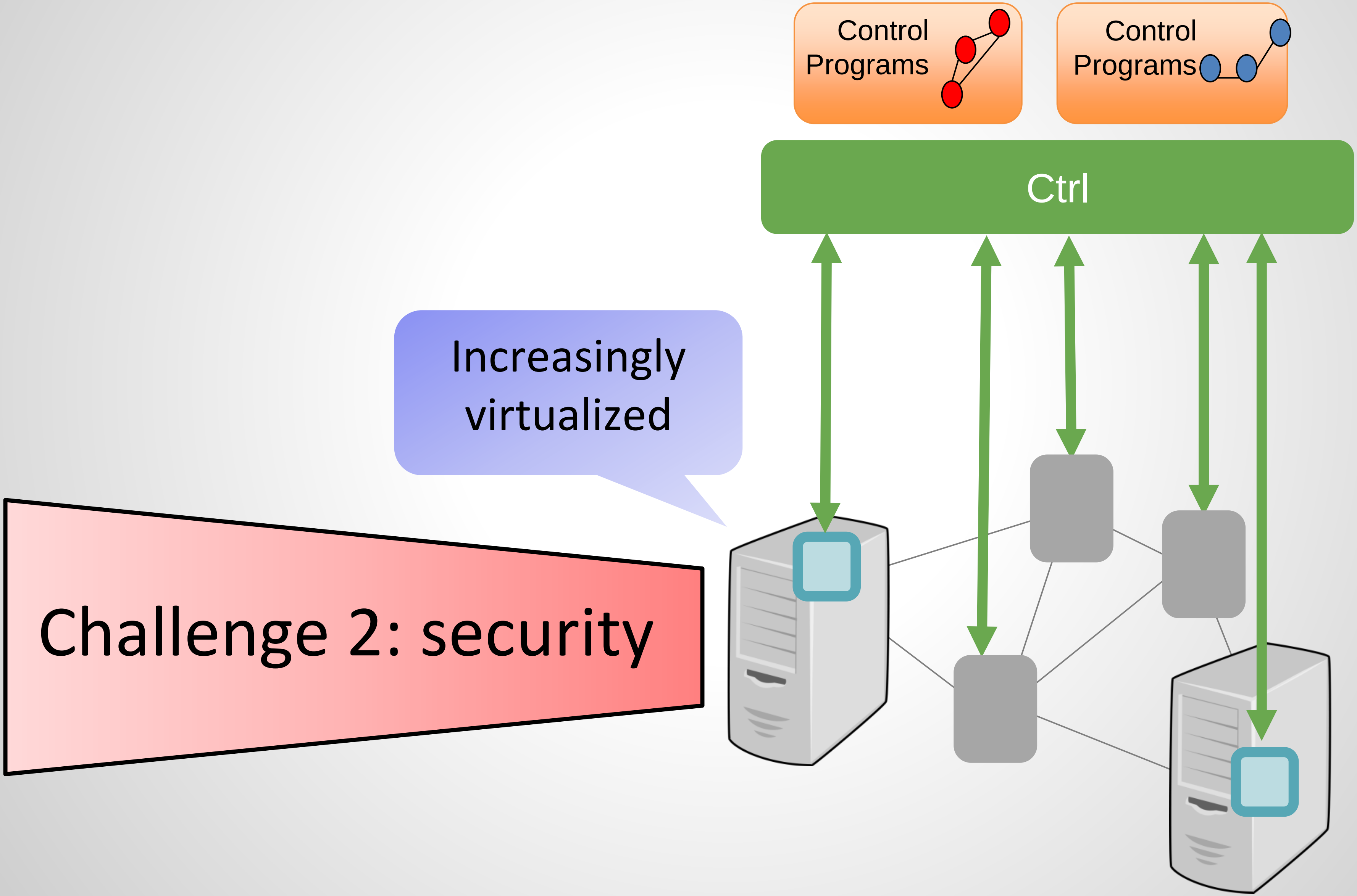
Open Problems

- Optimal resiliency on general networks
  - An open conjecture!
- Beyond resilience:
  - Stretch («space-filling curves»?)
  - Load
  - Combination
- Optimized to specific networks again

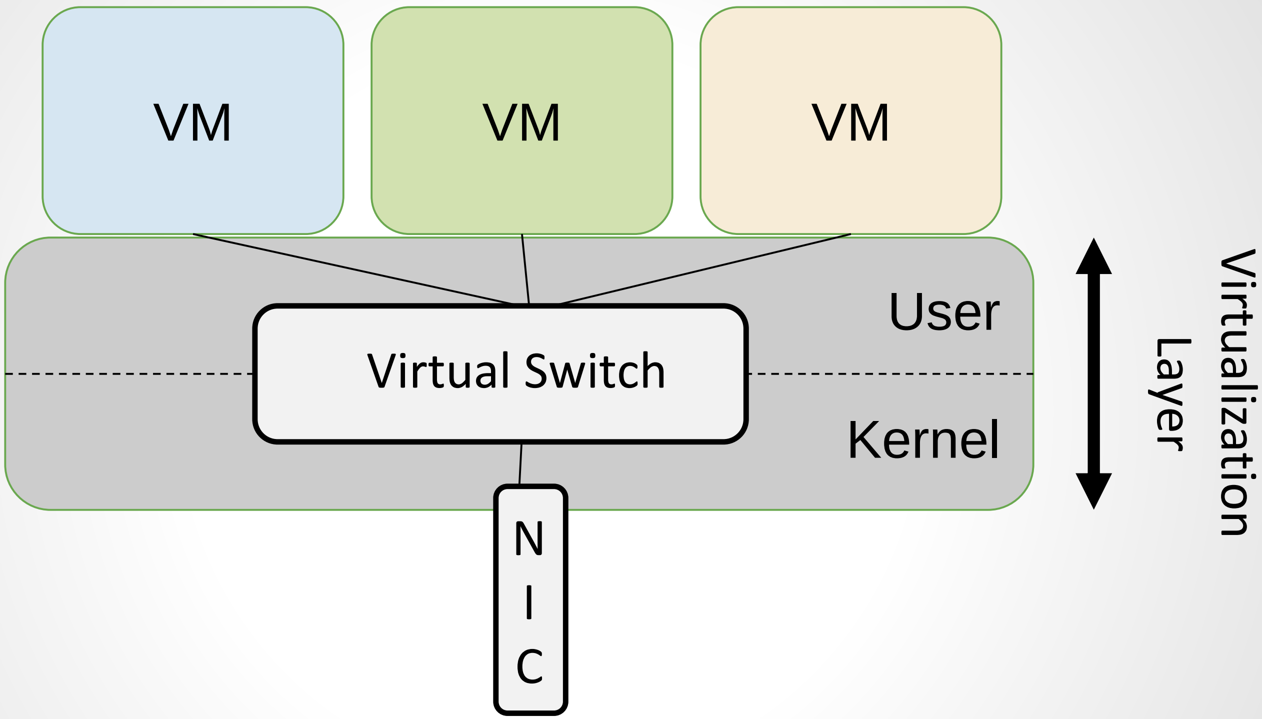
A Mental Model for This Talk



A Mental Model for This Talk



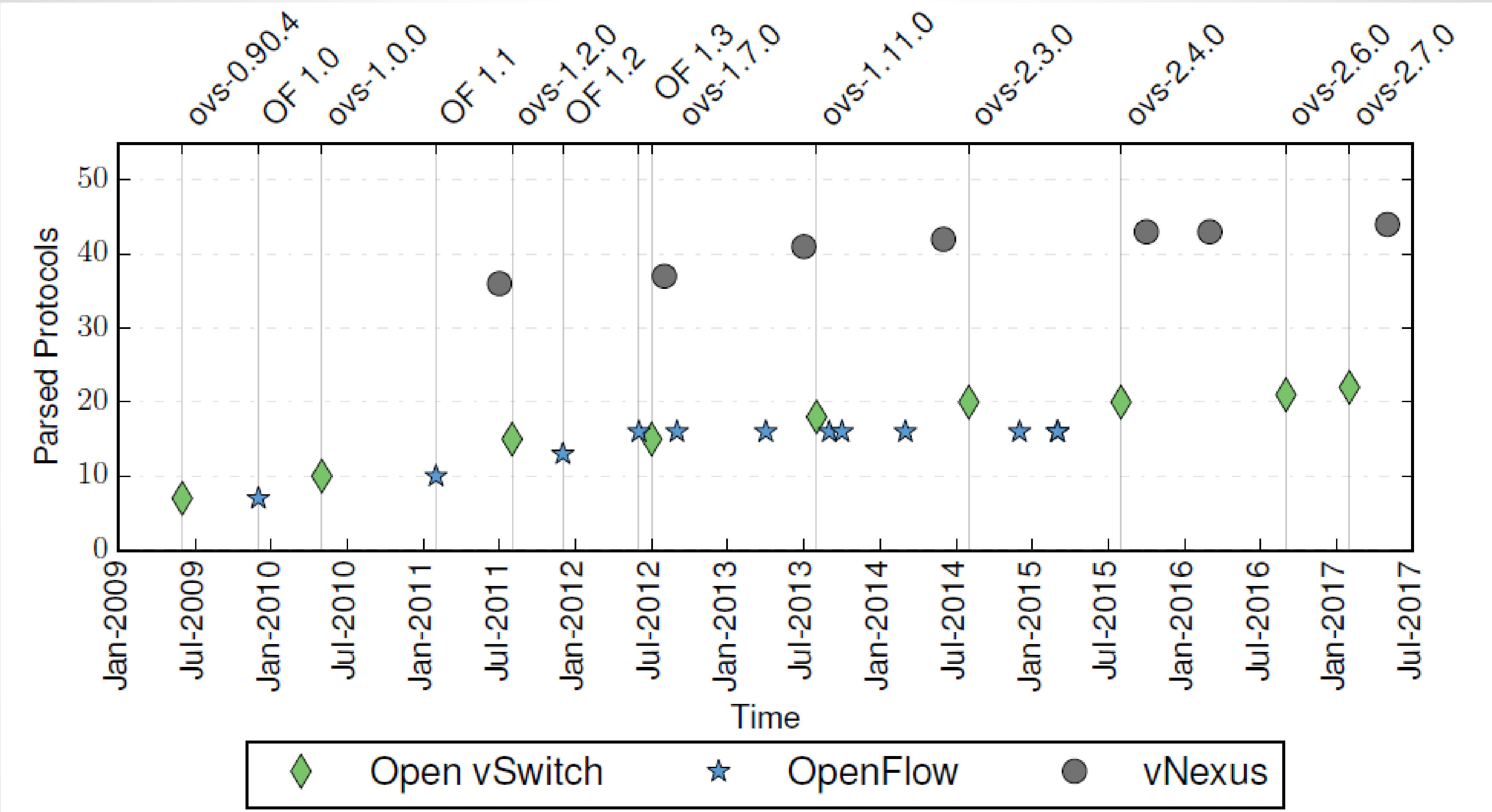
Virtual Switches



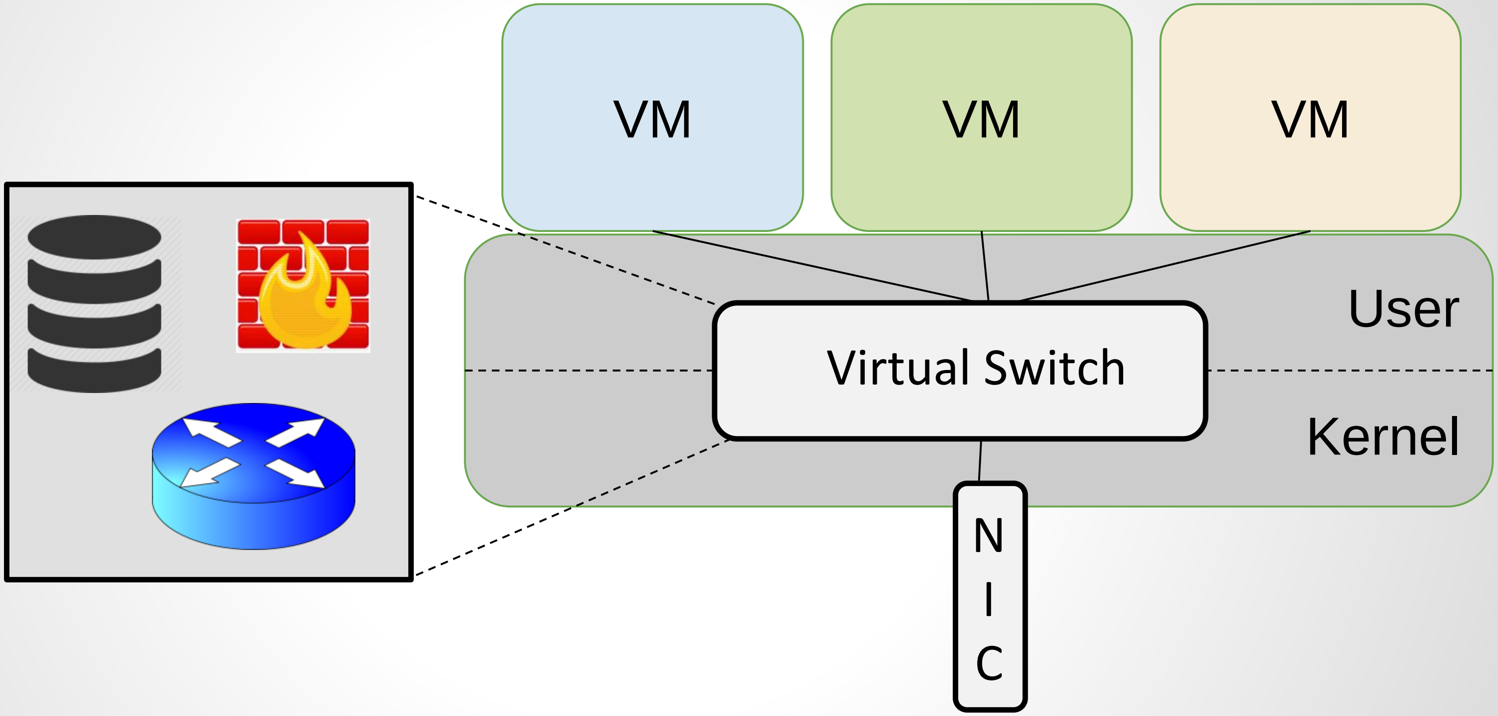
Virtual switches reside in the **server's virtualization layer** (e.g., Xen's Dom0). Goal: provide connectivity and isolation.

Increasing Complexity:
 # Parsed Protocols

Number of parsed high-level protocols constantly increases:



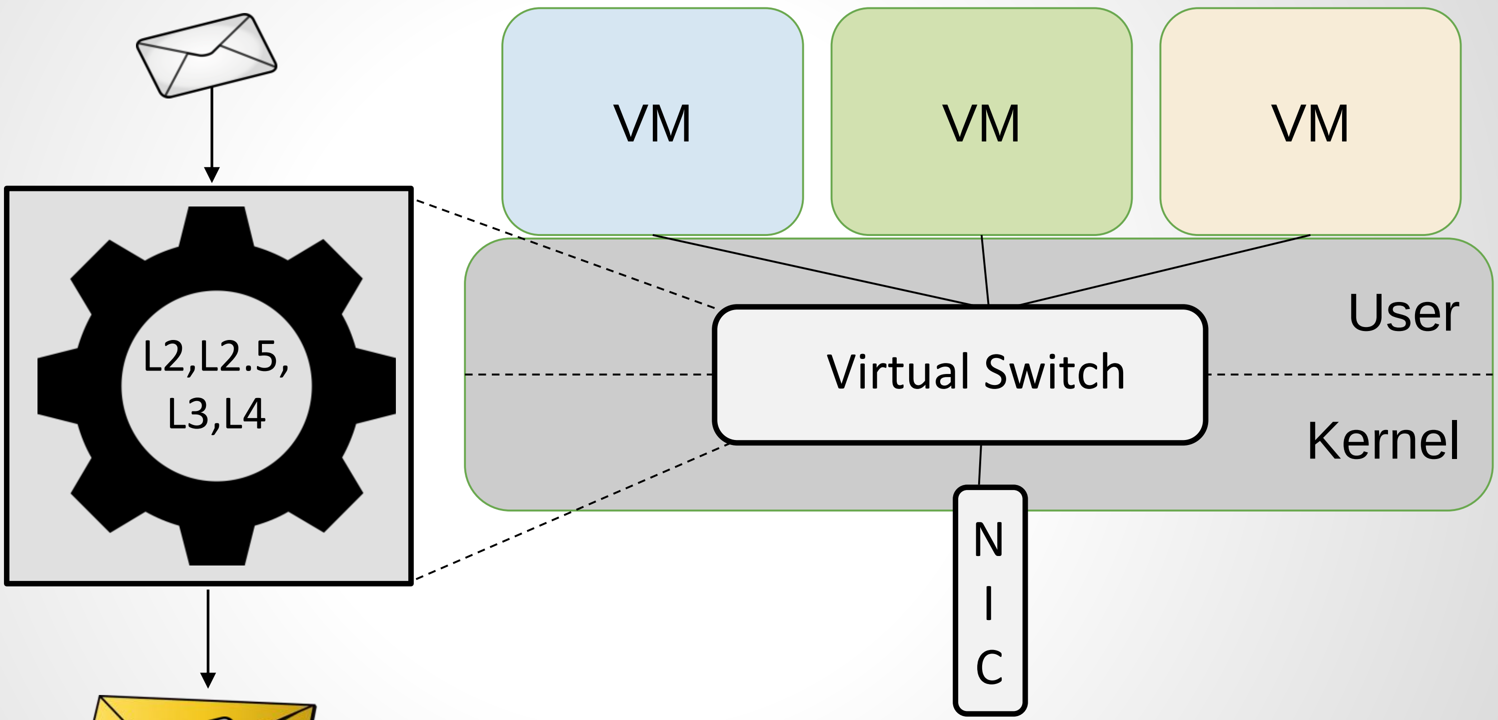
Increasing Complexity:
Introduction of middlebox functionality



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

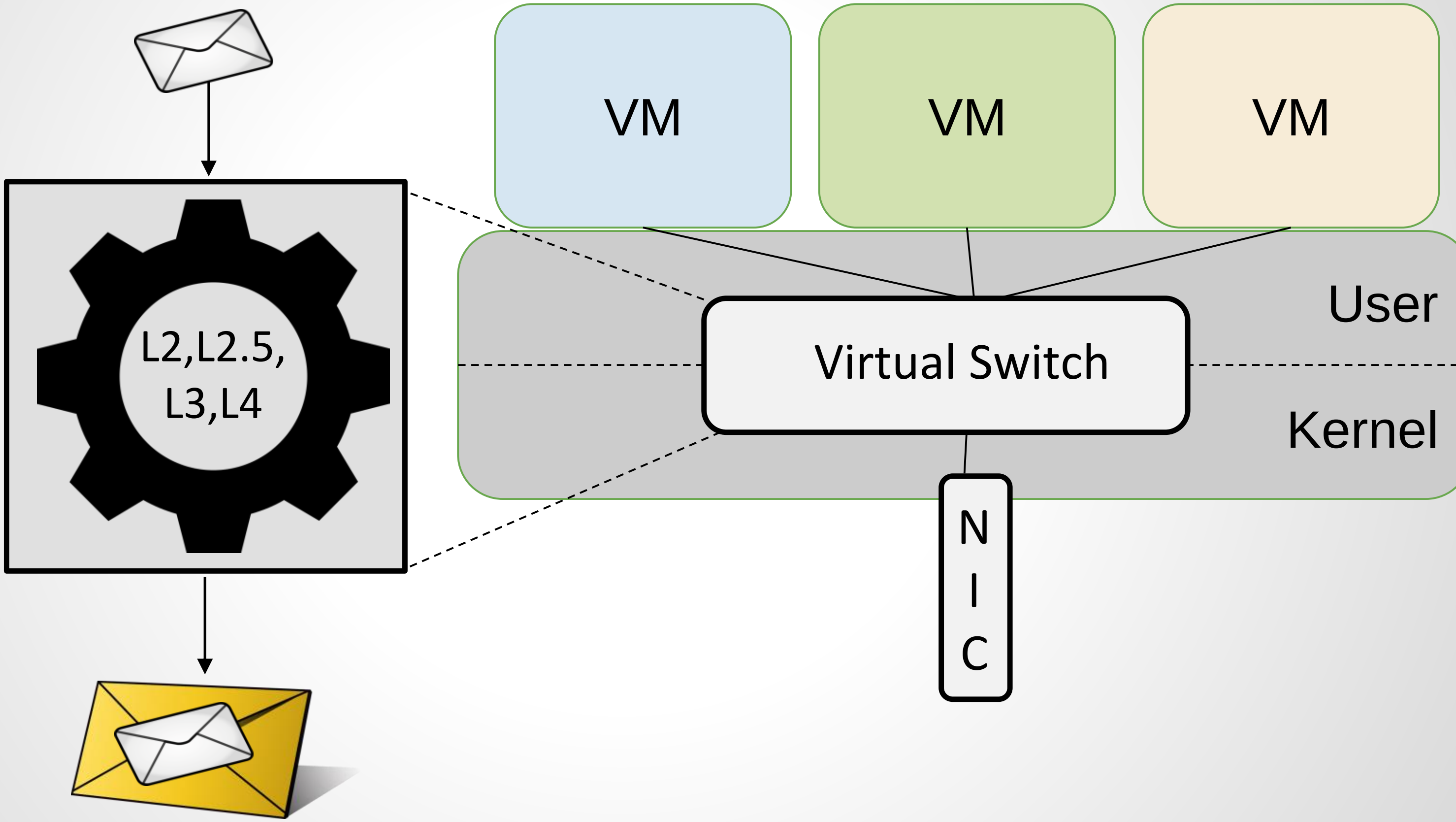
Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



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

Increasing Complexity:
Unified Packet Parsing

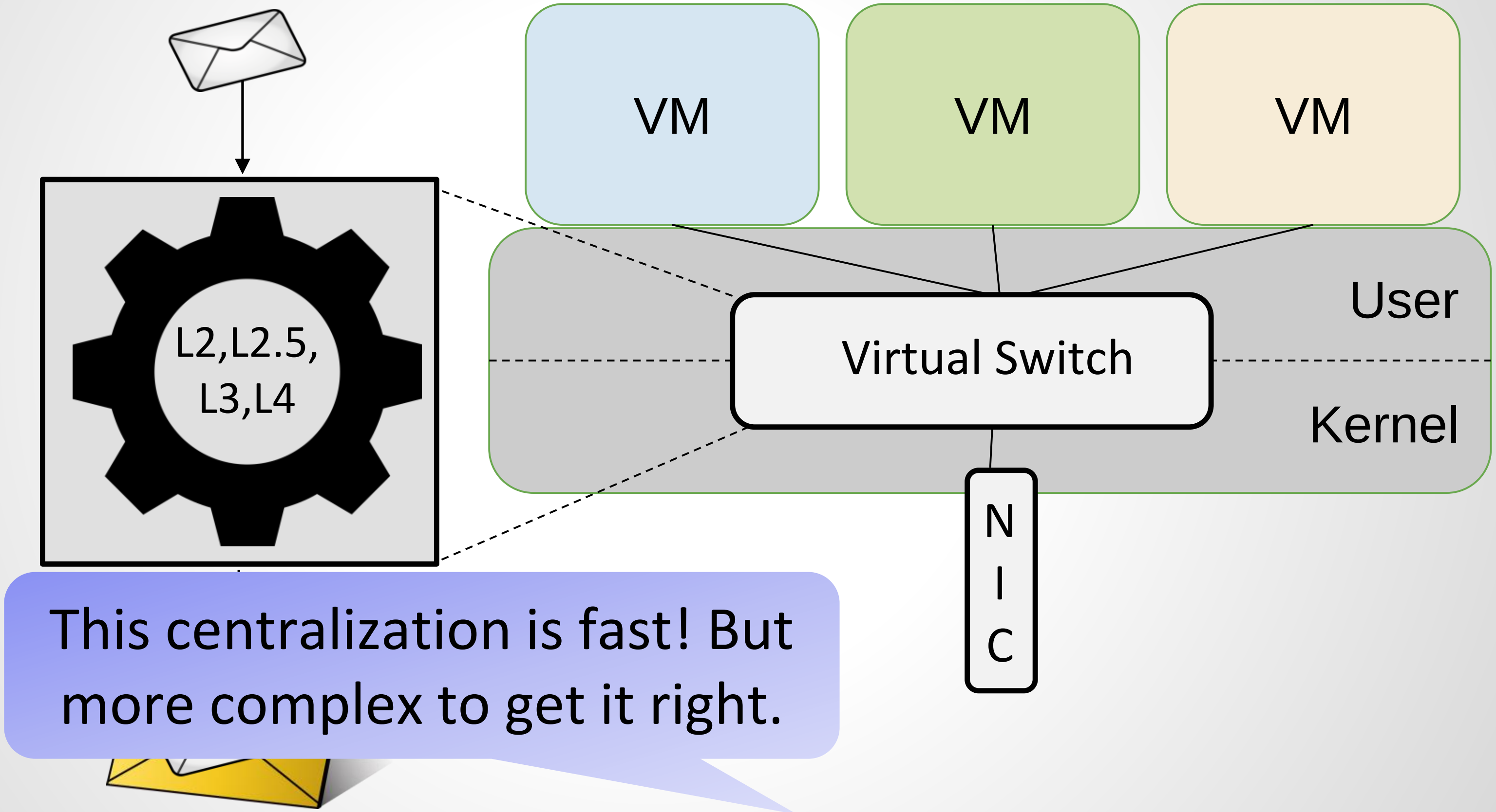
Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



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



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

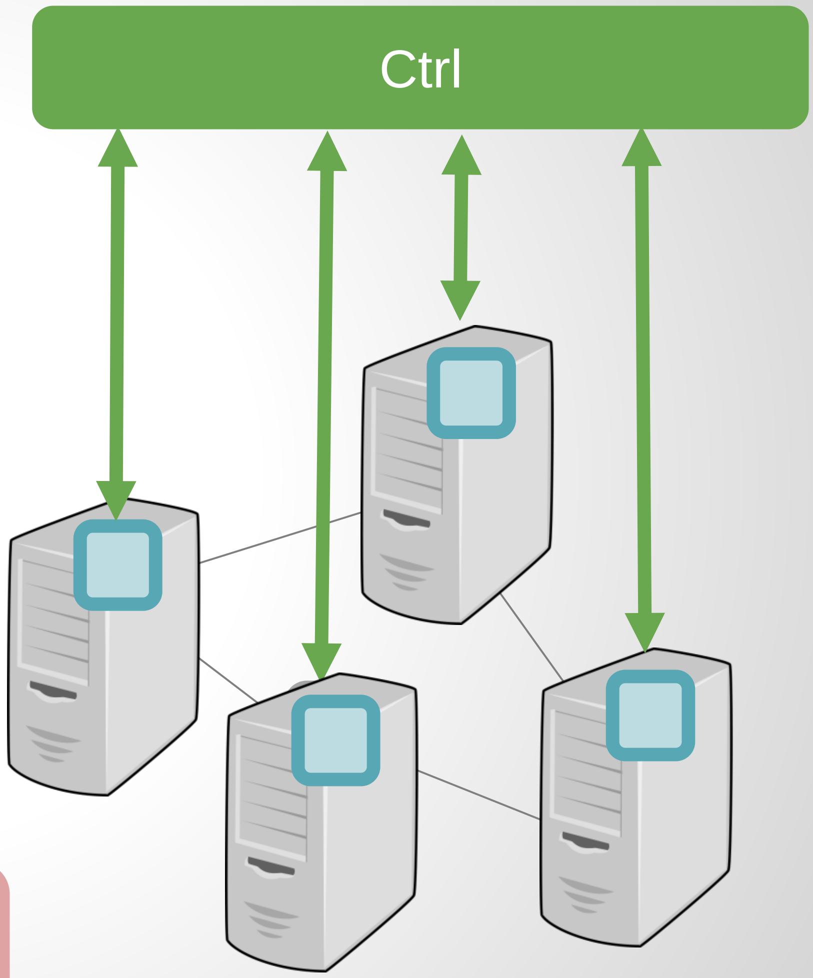
Complexity: The Enemy of Security!

- 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.

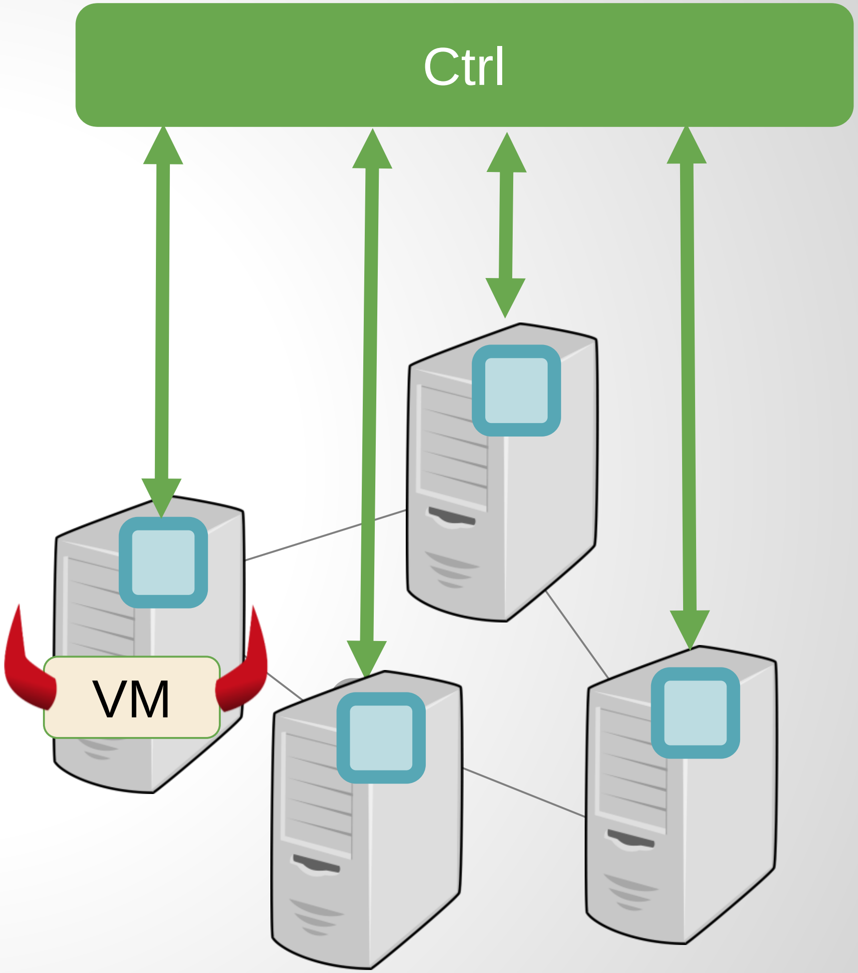
2. Impact of attack larger than with traditional data planes.



The Attack Surface: Closer...

Attack surface becomes closer:

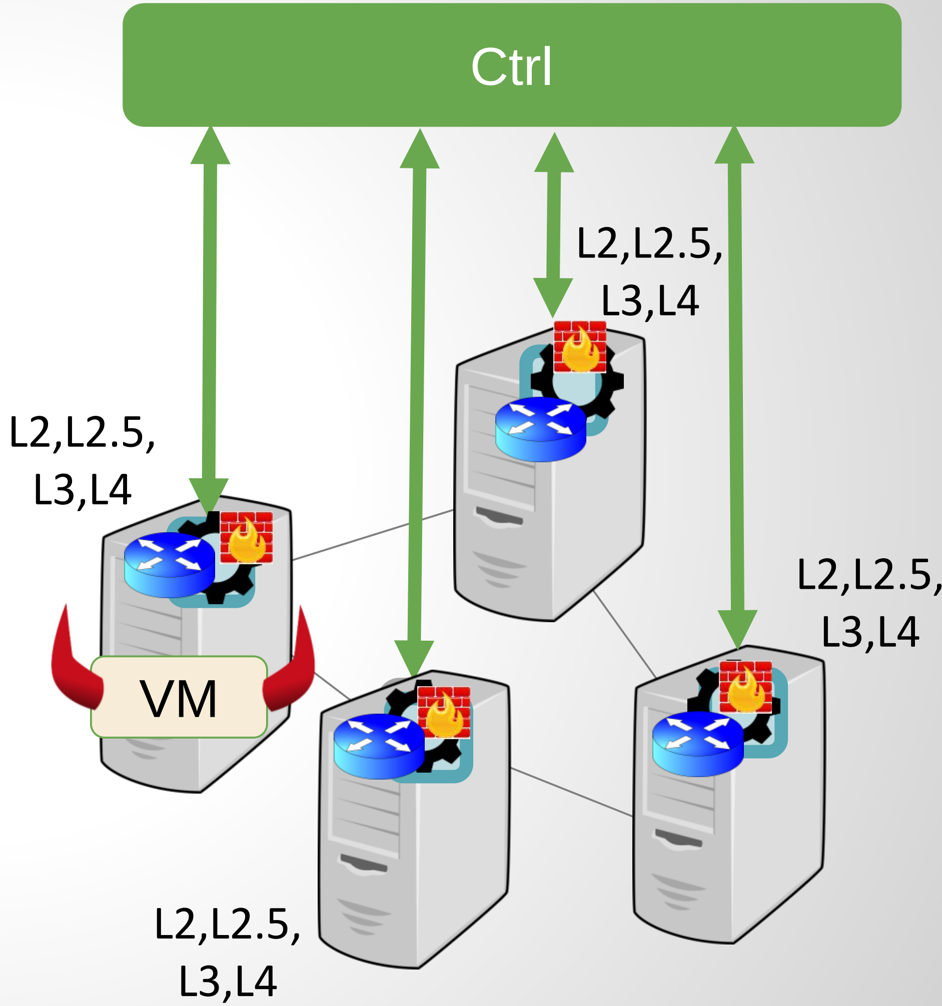
- ❑ Packet parser typically integrated into the code base of virtual switch
- ❑ First component of the virtual switch to process network packets it receives from the network interface
- ❑ May process attacker-controlled packets!



The Attack Surface: … More Complex …

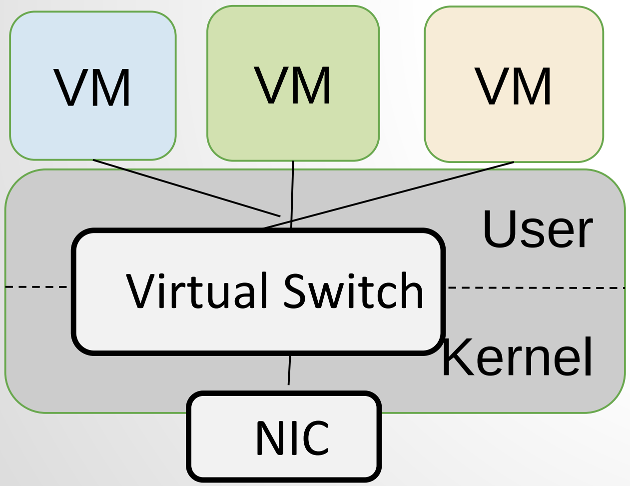
Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN

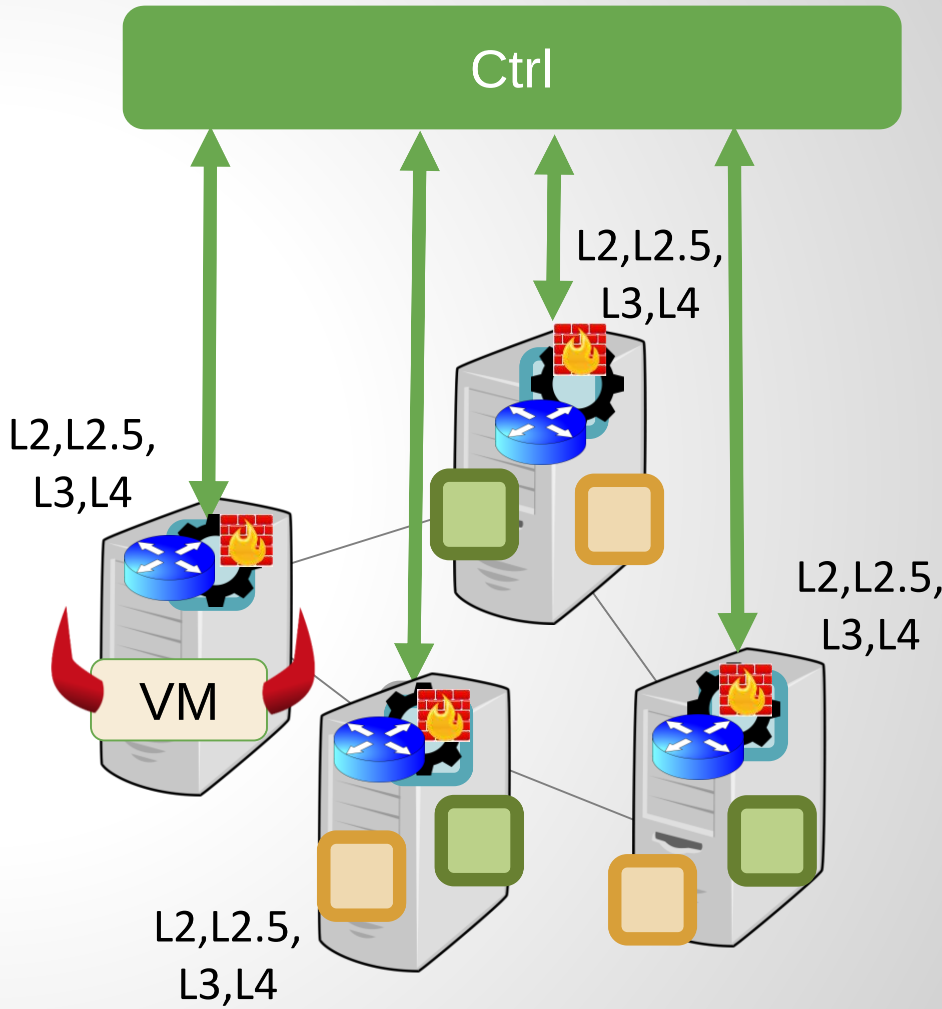
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



… Elevated Priviledges and Collocation …

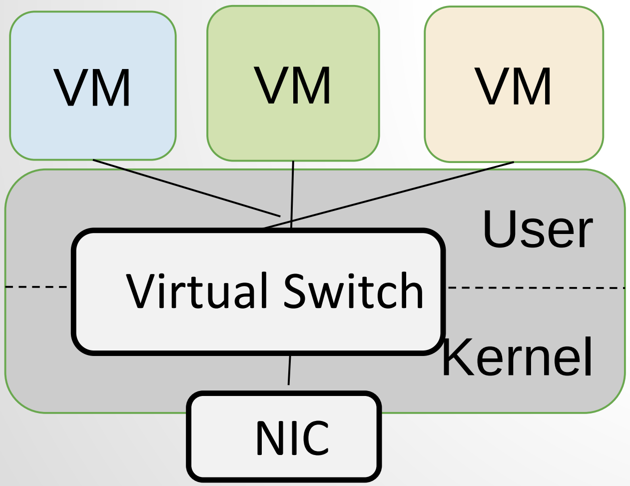
- Collocated (at least partially) with hypervisor’s Dom0 kernel space, guest VMs, image management, block storage, identity management, …

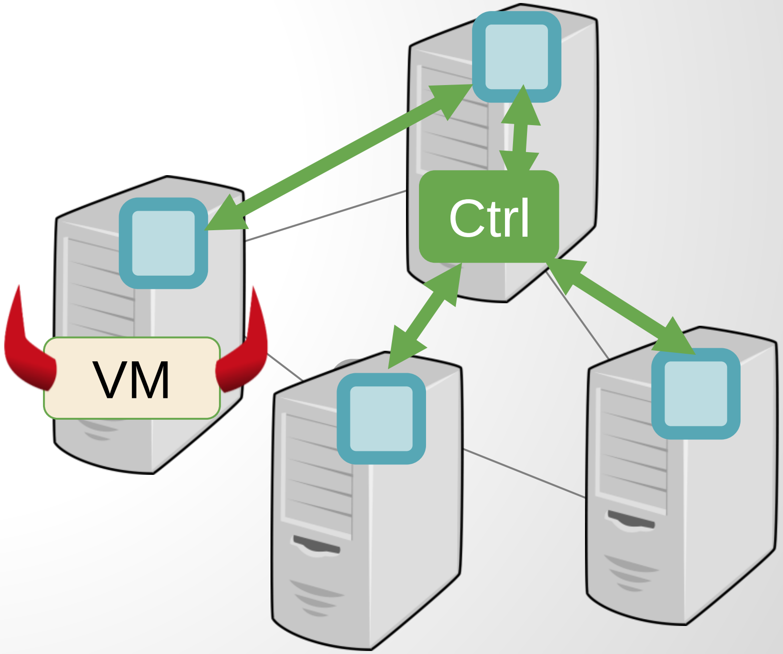




… Elevated Priviledges and Collocation …

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



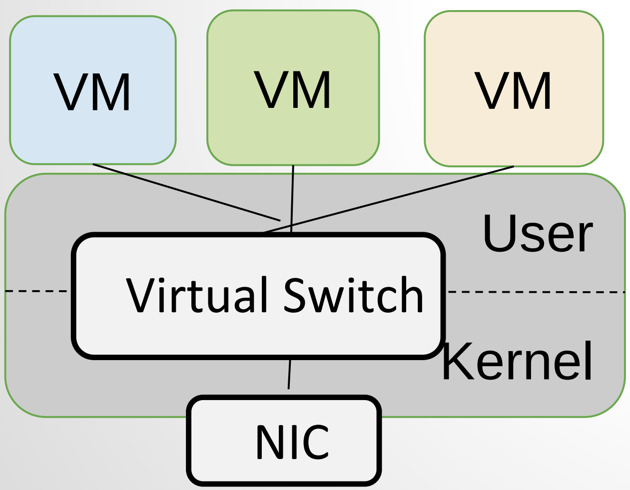


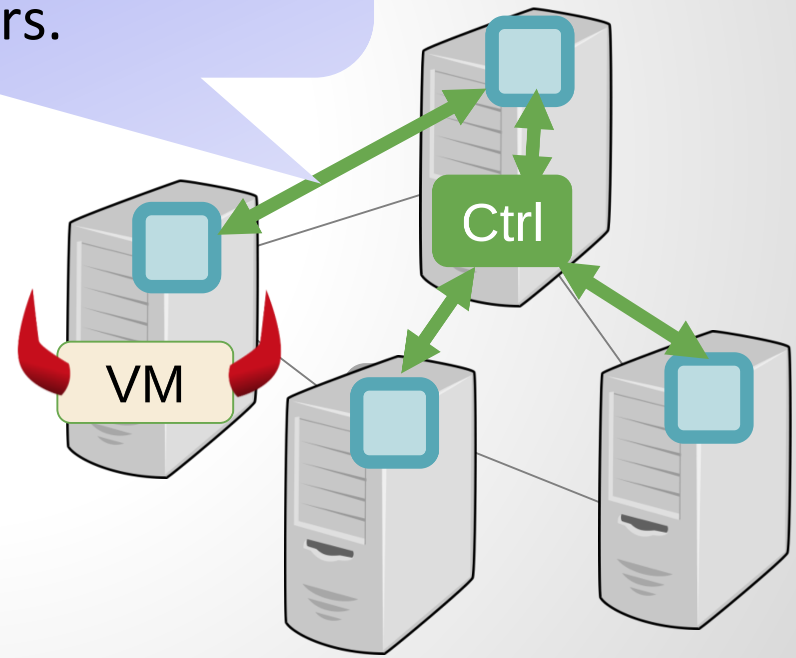
- … the controller itself.

… Centralization …

- Collocate
  with hype
  space), g
  managem
  identity m

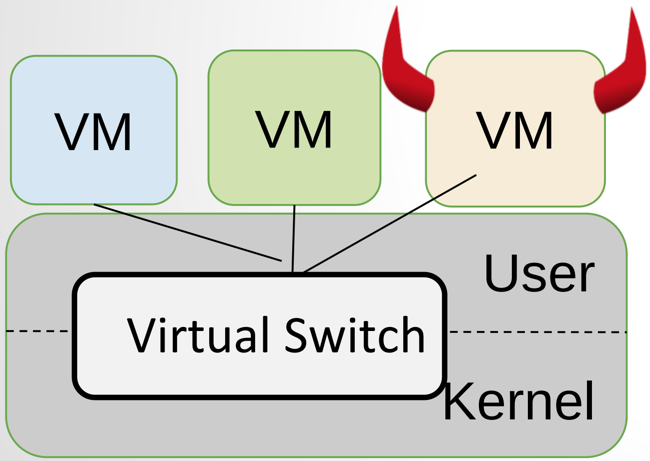
Available communication channels to (SDN/Openstack) controller! Controller needs to be reachable from all servers.





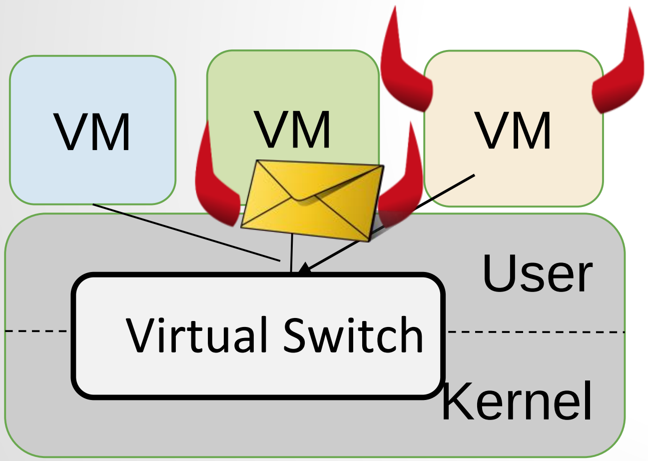
- … the controller itself.

Larger Impact: Case Study OVS



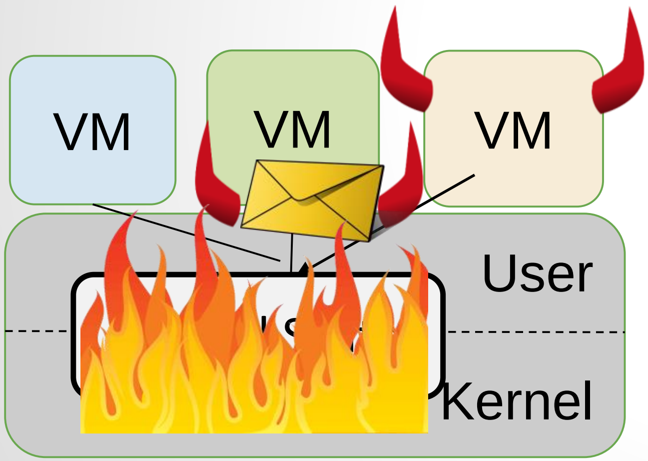
1. Rent a VM in the cloud (**cheap**)

Larger Impact: Case Study OVS



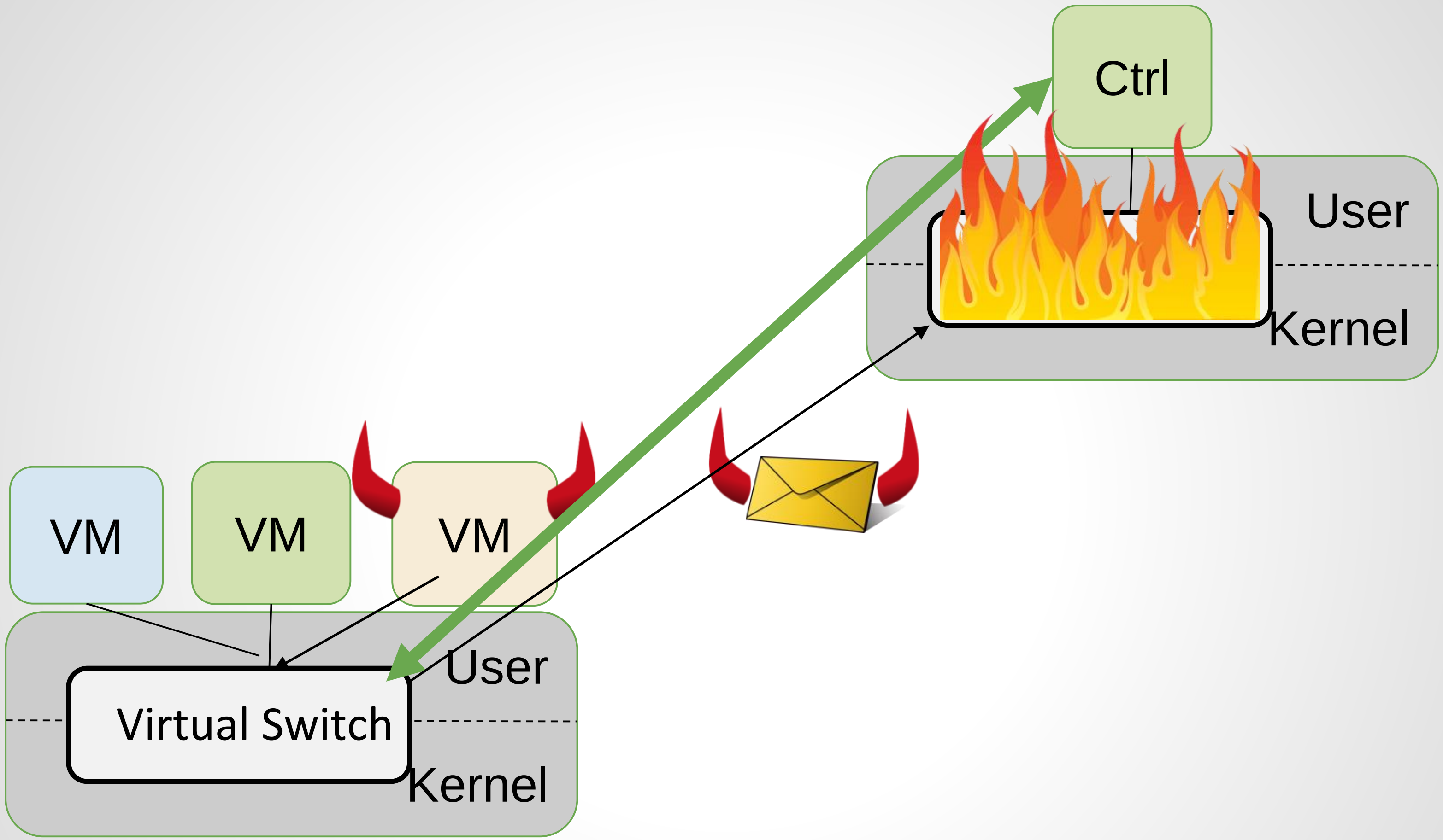
2. Send **malformed MPLS packet** to virtual switch (**unified parser** parses label stack packet beyond the threshold)

Larger Impact: Case Study OVS



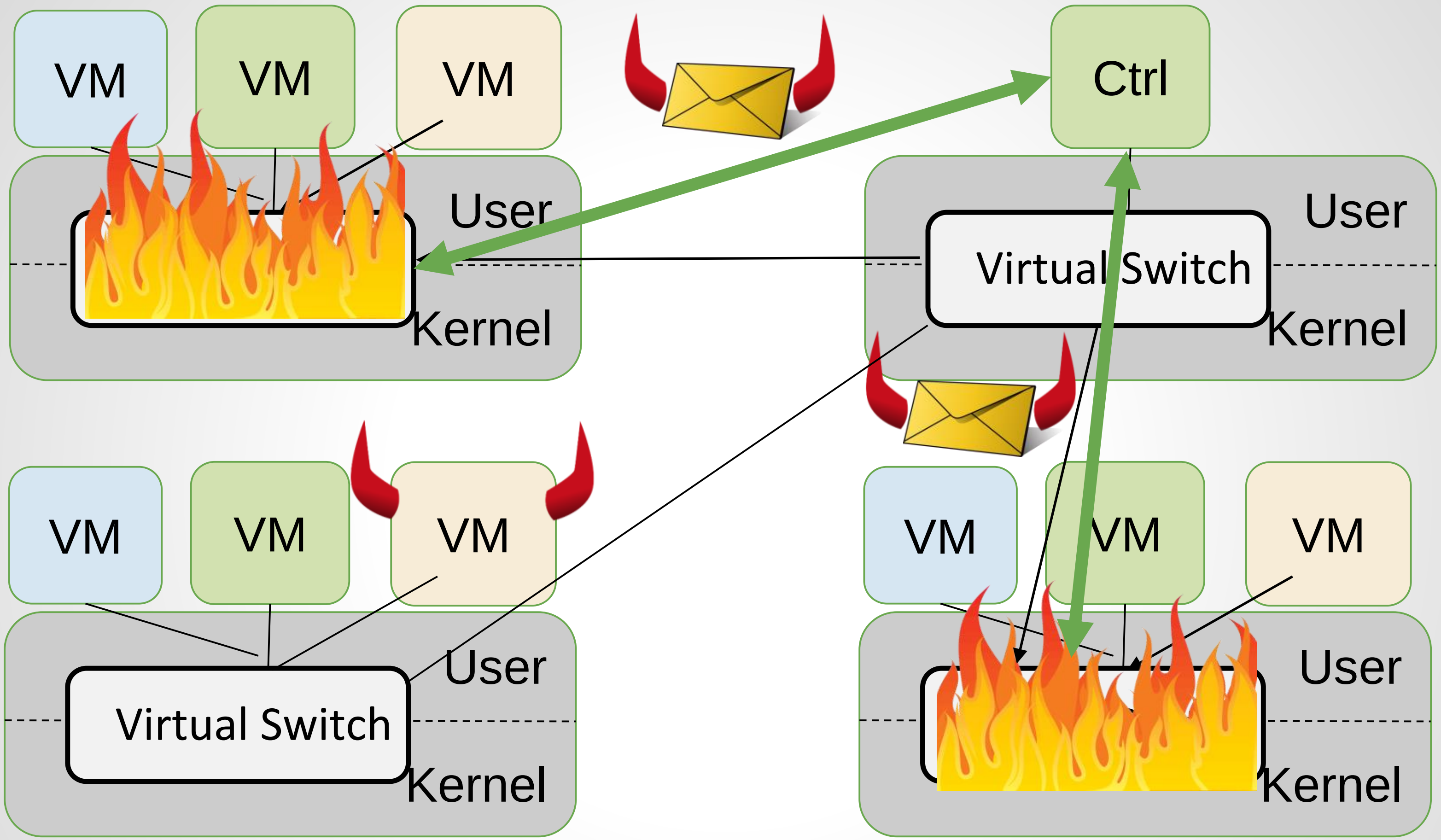
3. **Stack buffer overflow** in (unified) MPLS parsing code: enables remote code execution

Larger Impact: Case Study OVS



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

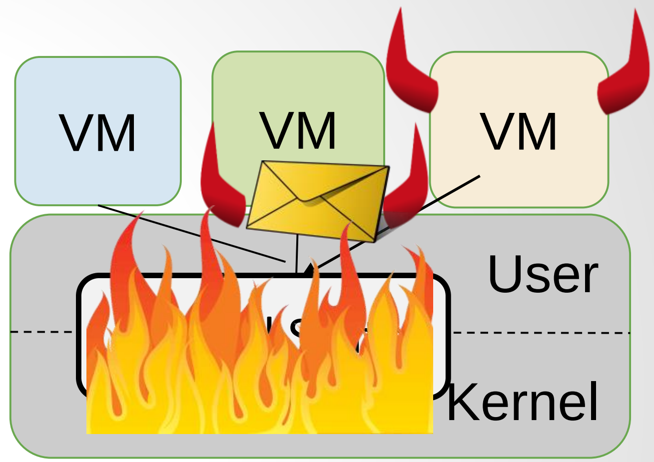
Larger Impact: Case Study OVS



5. Spread

A Novel 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.

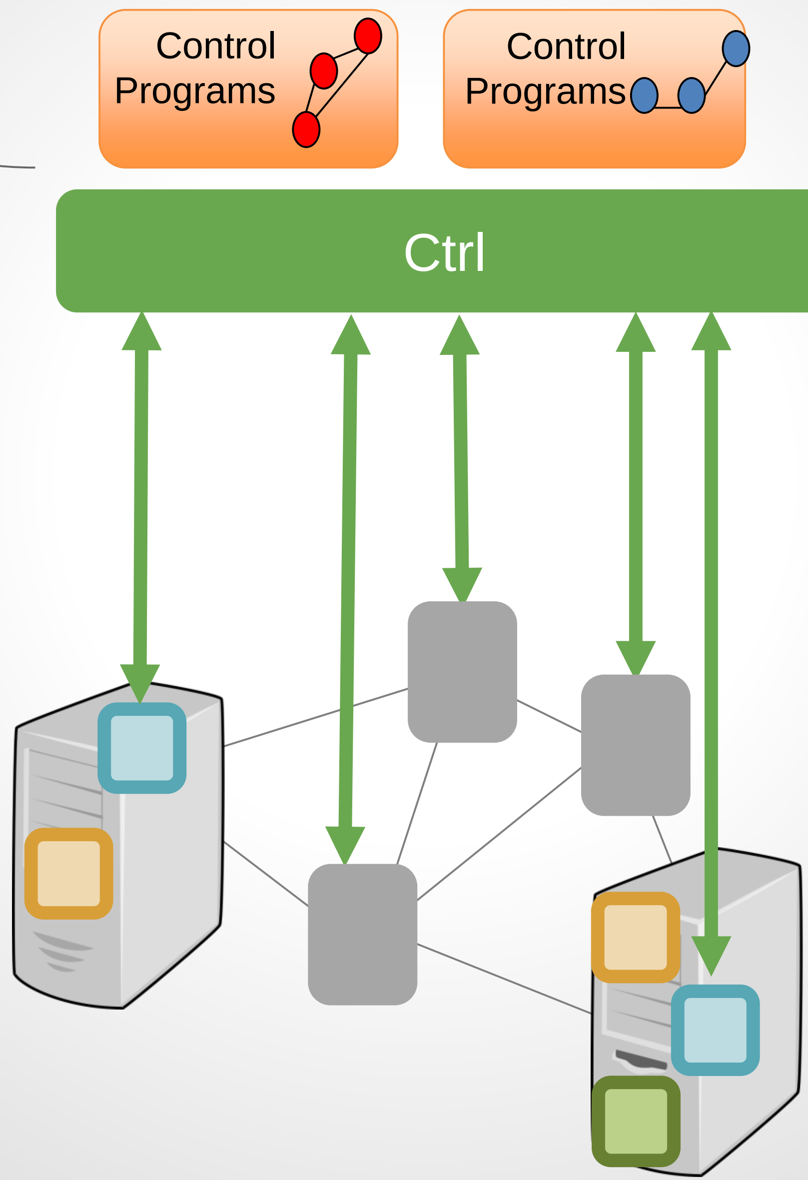
Conclusions

Opportunities

E.g., innovative services

Challenges

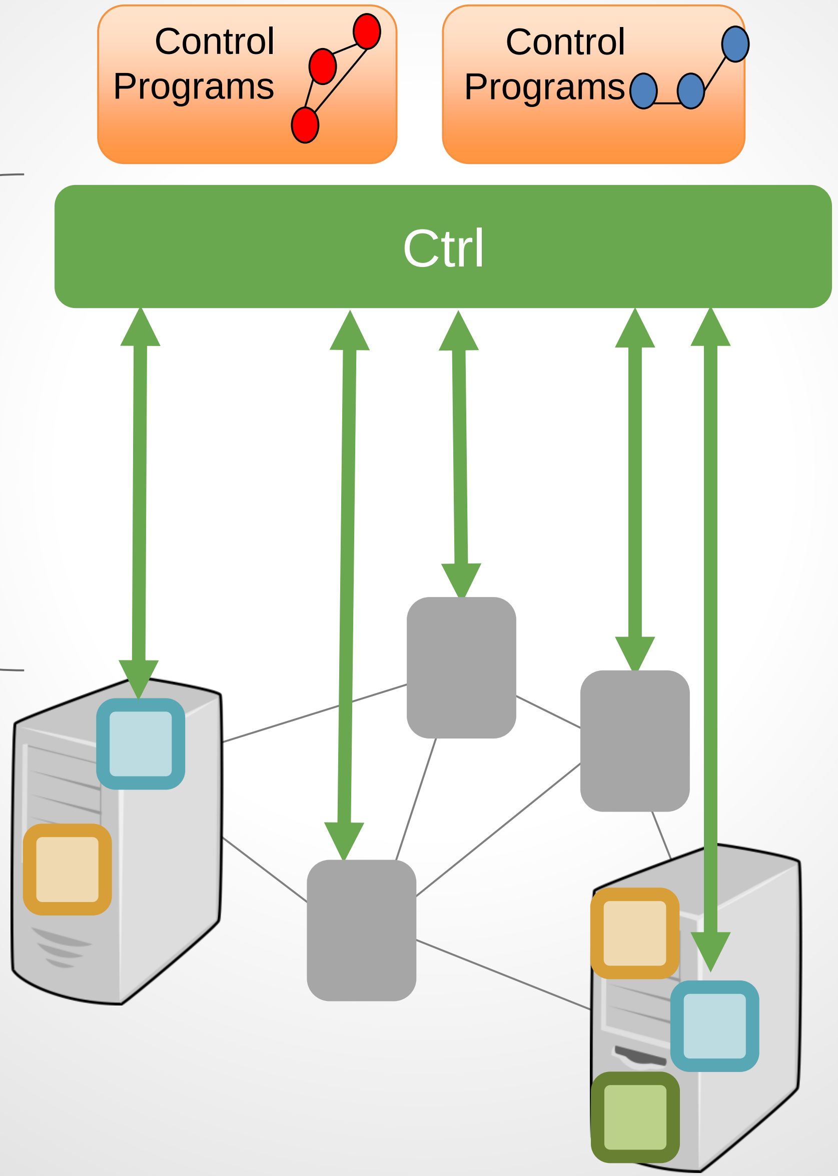
E.g., waypoint routing, traffic engineering



Conclusions

Opportunities

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



Challenges

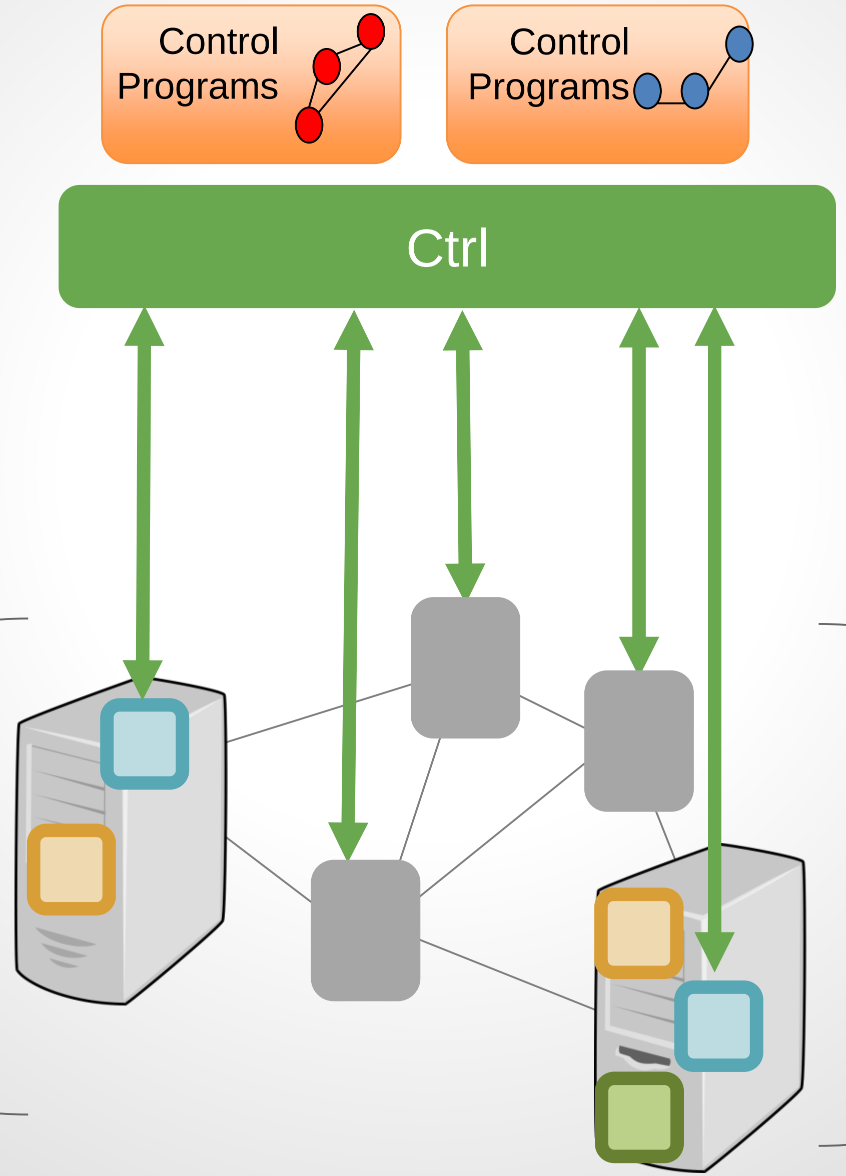
E.g., keeping controller up-to-date

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, ….?

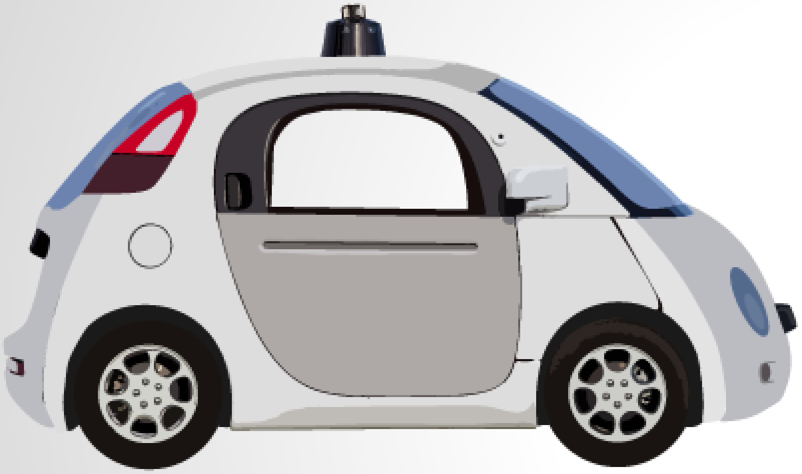
Stepping Back Even A Little Bit More…

- SDN + virtualization offer great flexibilities: are enablers

- Exploiting and analyzing them is still complex:
  - Algorithms are non-trivial (e.g., waypoint routing)
  - Interfaces / abstractions / languages still quite low-level (e.g., configuration of conditional failover rules)
  - Networked systems are still complex and hard to model (e.g., hypervisor interference)
  - Many uncertainties: hardware, demand, interference

Maybe we need a different approach to networking? Self-adjusting, data-driven, machine-learning, … networks!

A Better Vision of Future Networked Systems?

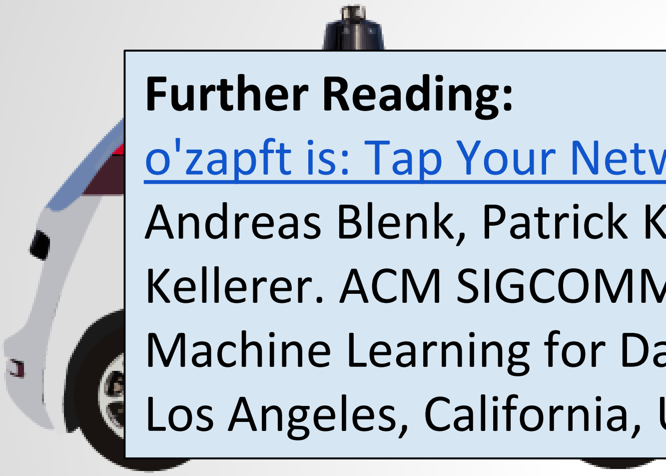


Analogy to self-driving cars: more high-level task-, measurement-, data- and learning-driven rather than model-driven?

Also: self-stabilizing, self-adjusting, self-optimizing….

Credits: Why (and How) Networks Should Run Themselves. Nick Feamster and Jennifer Rexford

A Better Vision of Future Networked Systems?



Further Reading:
o'zapft is: Tap Your Network Algorithm's Big Data!
Andreas Blenk, Patrick Kalmbach, Stefan Schmid, and Wolfgang Kellerer. ACM SIGCOMM 2017 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks (**Big-DAMA**), Los Angeles, California, USA, August 2017.

Also: self-stabilizing, self-adjusting, self-optimizing….

Credits: Why (and How) Networks Should Run Themselves. Nick Feamster and Jennifer Rexford

Thank you! Questions?