Exploiting Flexibilities in SDNs: Opportunities, Challenges, and Security Implications

Stefan Schmid

Aalborg University, DK & TU Berlin, DE

Nice to meet you!

A rehash: It's a great time to be a scientist!

"We are at an interesting inflection point!"
Keynote by George Varghese
at SIGCOMM 2014

Big Networking Challenges: Dependability

Even techsavvy companies struggle to provide reliable operations

Mostly manual and ad-hoc!

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

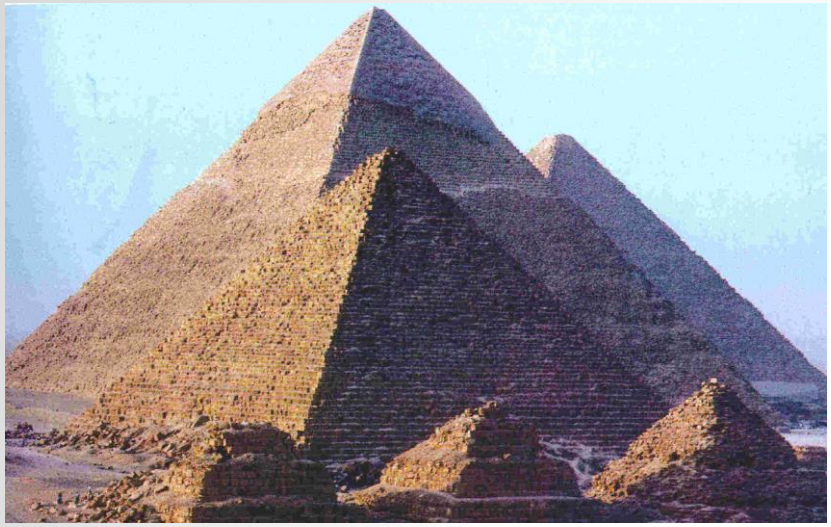
A network change was […] executed incorrectly […] more "stuck" volumes and added more requests to the re-mirroring storm

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

Source: Talk by Nate Foster at DSDN Workshop

Big Networking Challenges: Security



The Internet on first sight:

- Monumental
- Passed the “Test-of-Time”
- Should not and cannot be changed

The Internet on second sight:

- Antique
- Britle
- Successful attacks more and more frequent (e.g., based on IoT)

Source: Slide by Adrian Perrig

Big Networking Challenges: Lack of Good 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!

Guess who gets blamed?

- 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

Big Networking Challenges: Predictable Performance

- Cloud-based applications like batch processing, streaming, and scale-out databases generate a significant amount of network traffic
- Considerable fraction of their runtime is due to network activity
  - Facebook traces: network transfers account for 33% of the execution time
- But: available bandwidth (to tenant) varies significantly over time
- So: How to render networking more predictable?

A Case for SDN and NV?

Software-defined networking: Introduces many new flexibilities by decoupling and consolidating the control plane. Enables fast control plane innovations, automatic verification, new debugging tools, …

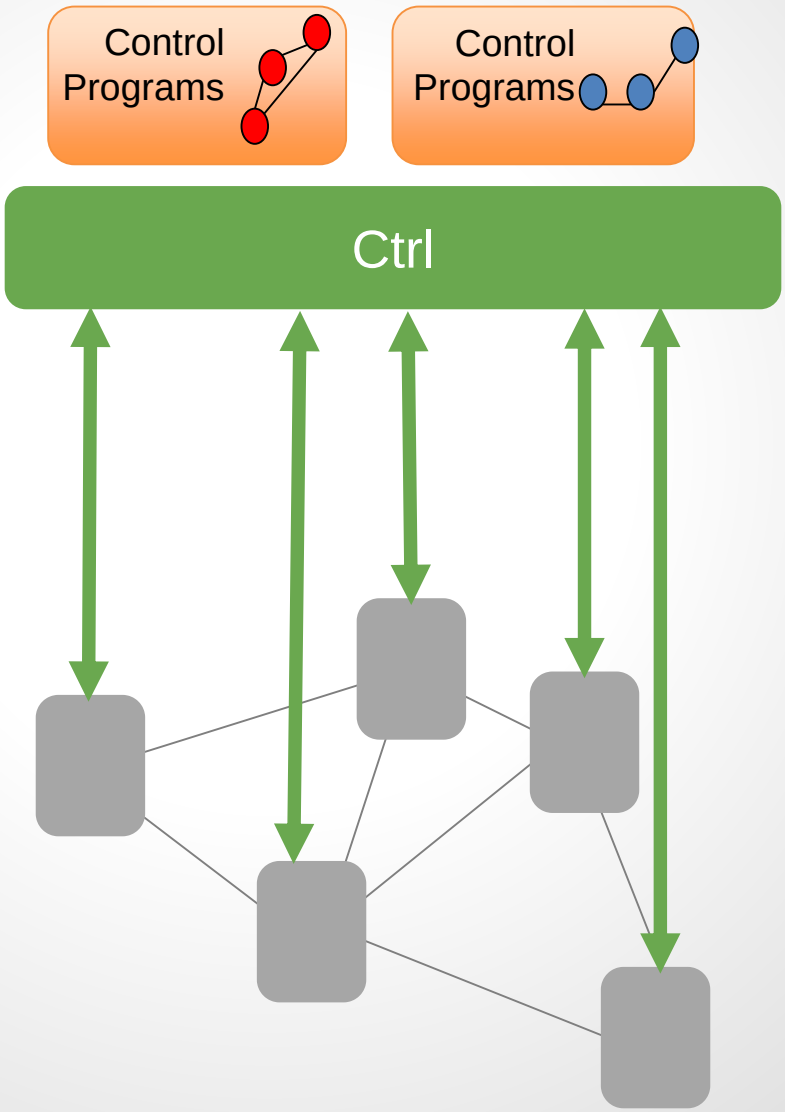
A killer application for SDN

Network virtualization: Allows to ensure performance isolation, support networks with different stacks to co-exist on same infrastructure. Etc.

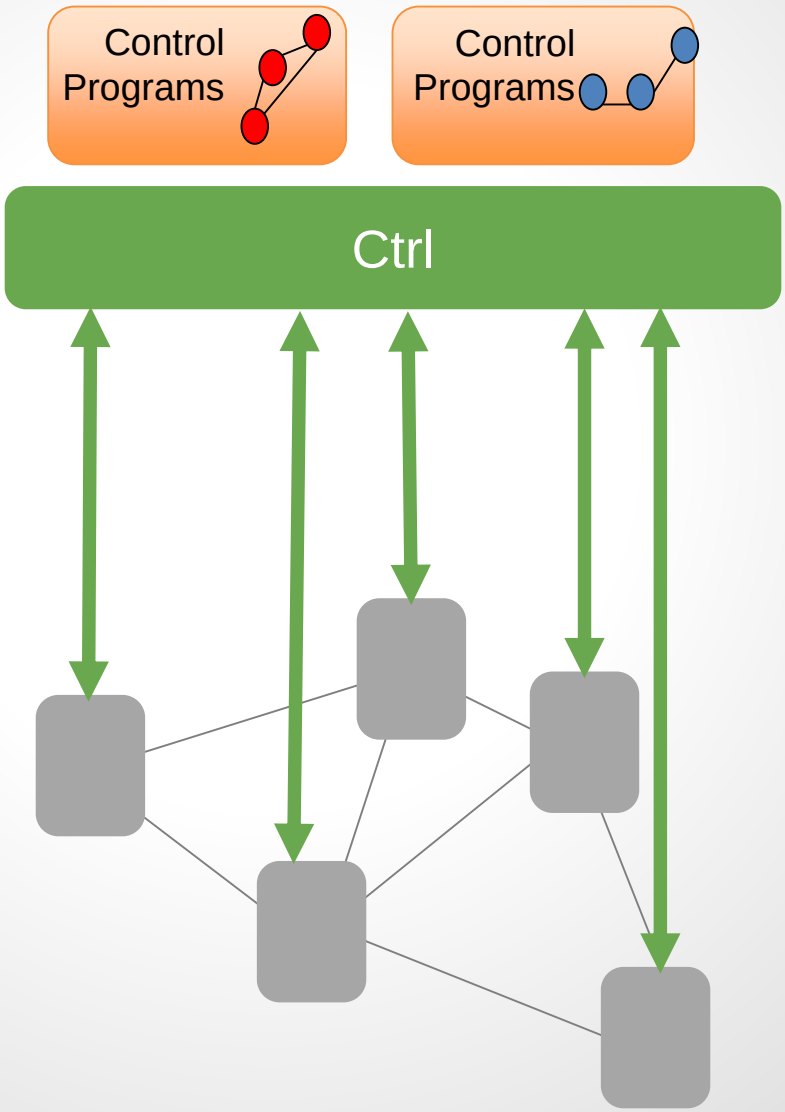
New opportunities - new challenges!

Setting the Stage: A Mental Model for SDNs

SDN outsources and consolidates control over multiple devices to (logically) centralized software controller

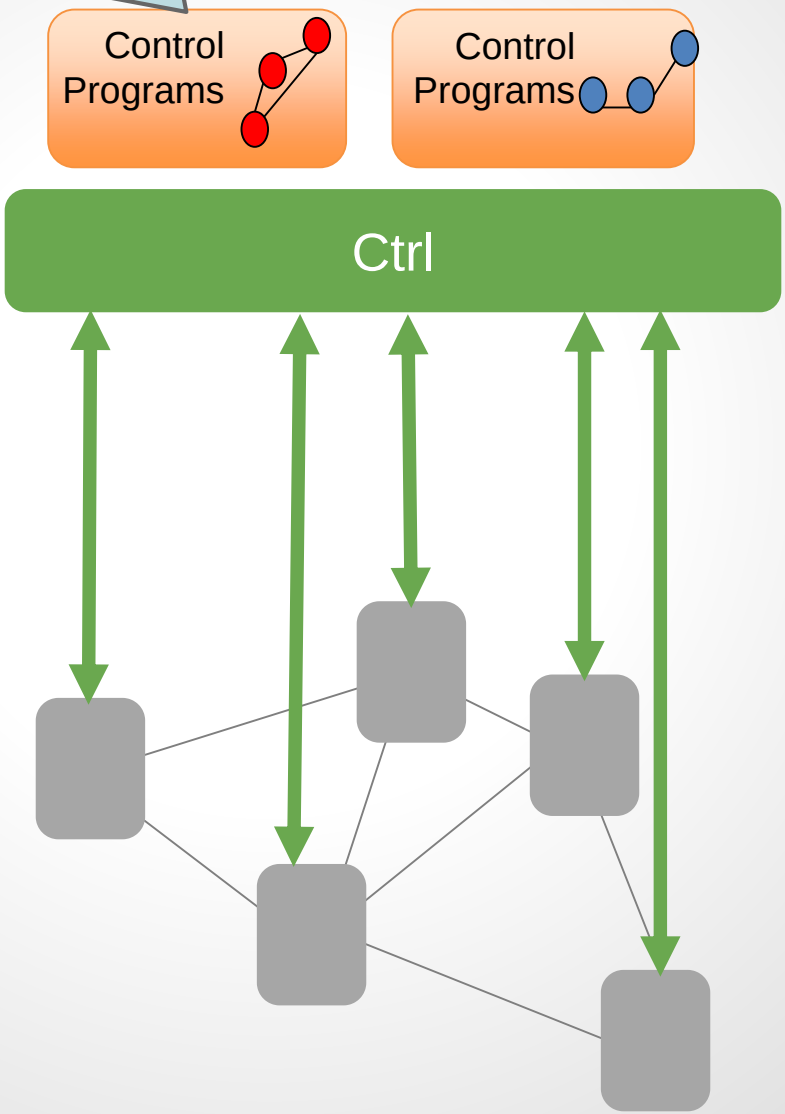


SDN = More Flexibilities…



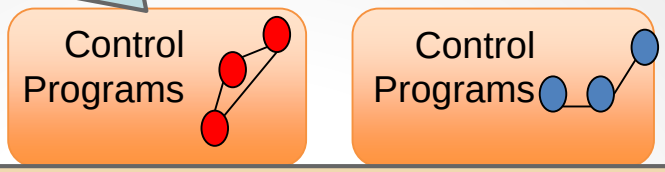
exibilities…

Flexibility 1: New flexibilities to devise algorithms for traffic engineering, load-balancing, logically-centralized failover, etc.



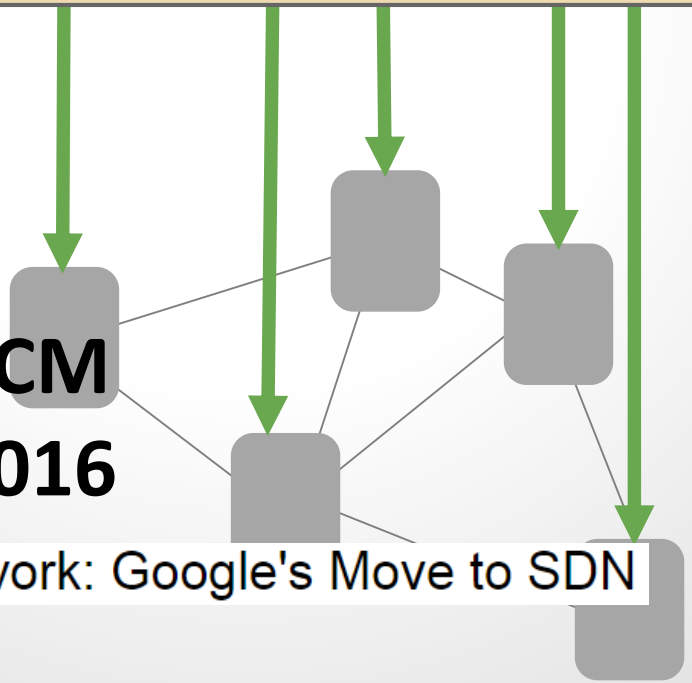
exibilities…

Flexibility 1: New flexibilities to devise algorithms for traffic engineering, load-balancing, logically-centralized failover, etc.



Key reasons for Google's move to SDN: more fine-grained traffic engineering, and more predictable and faster failure recovery.

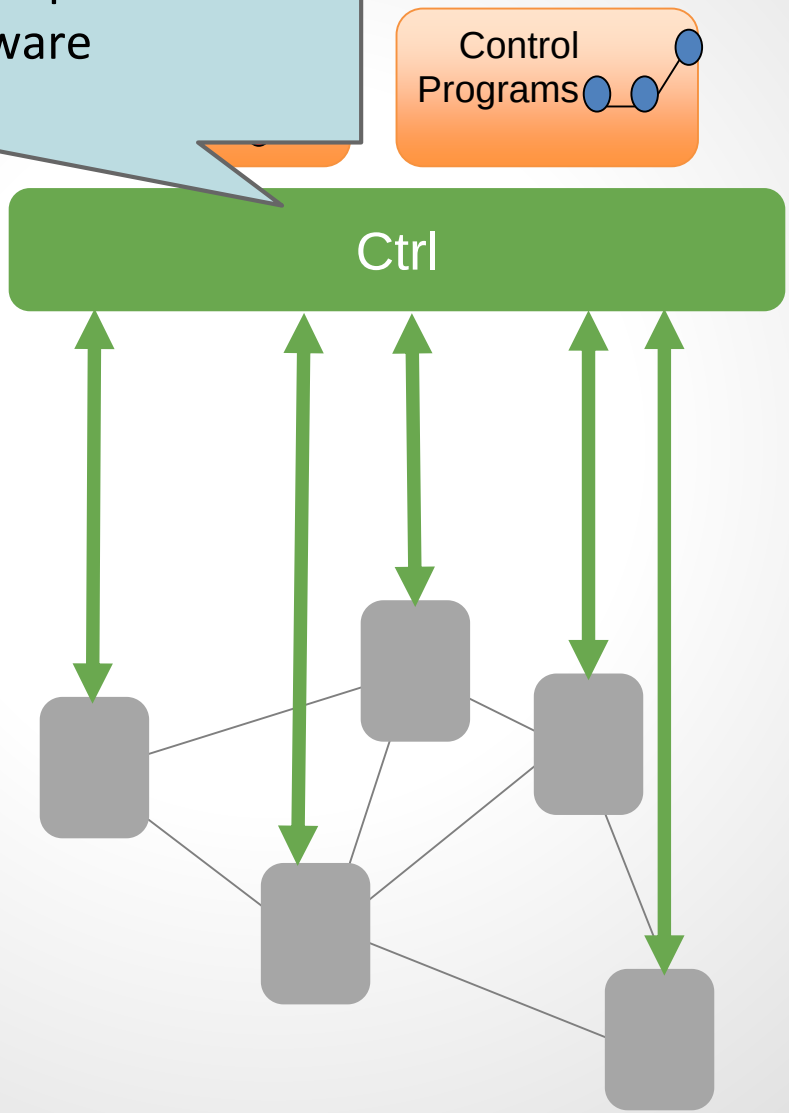
C.ACM 3/2016



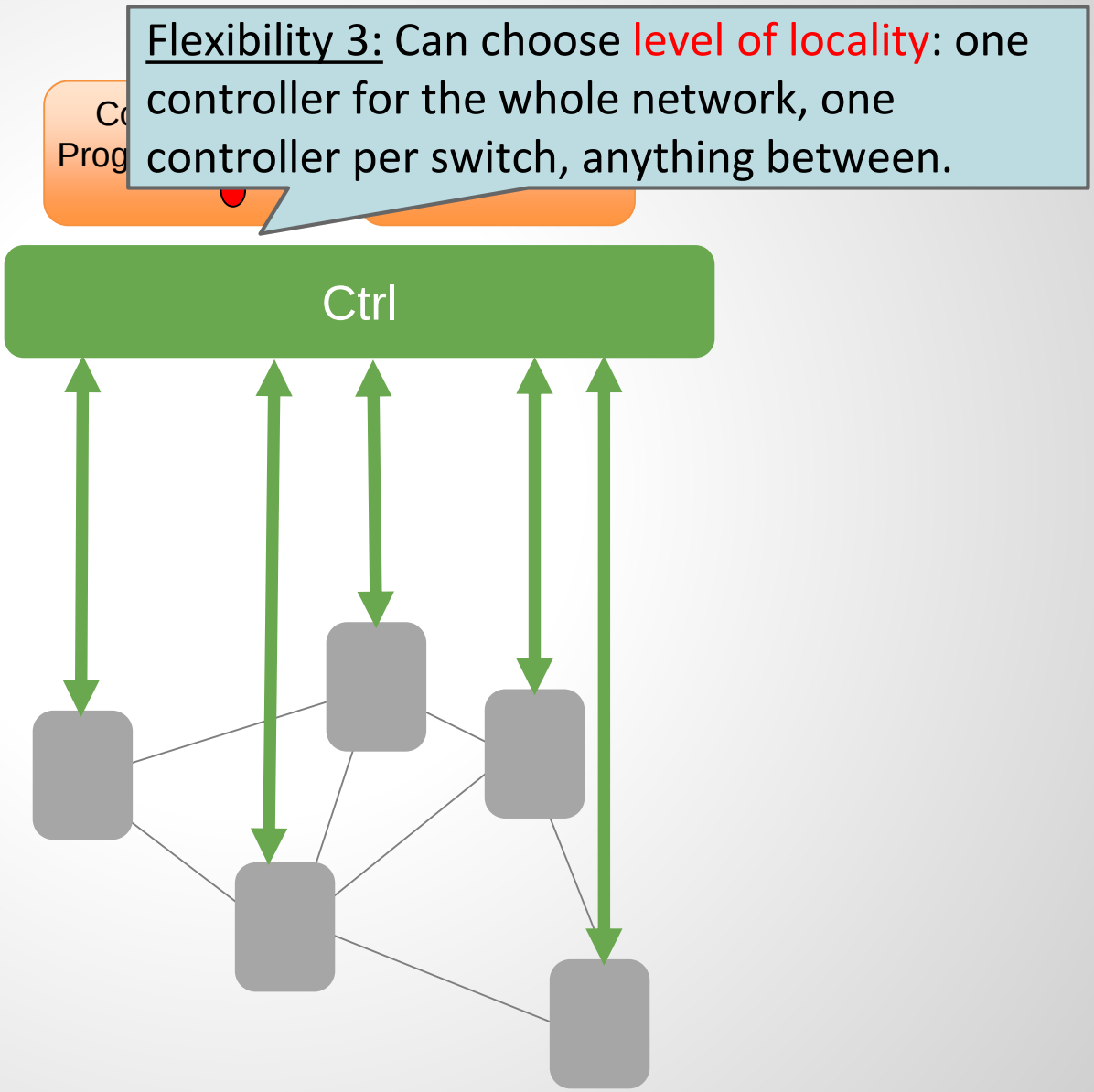
A Purpose-Built Global Network: Google's Move to SDN

SDN = More Flexibilities…

Flexibility 2: Decoupling! Control plane can evolve independently of data plane: innovation at speed of software development.



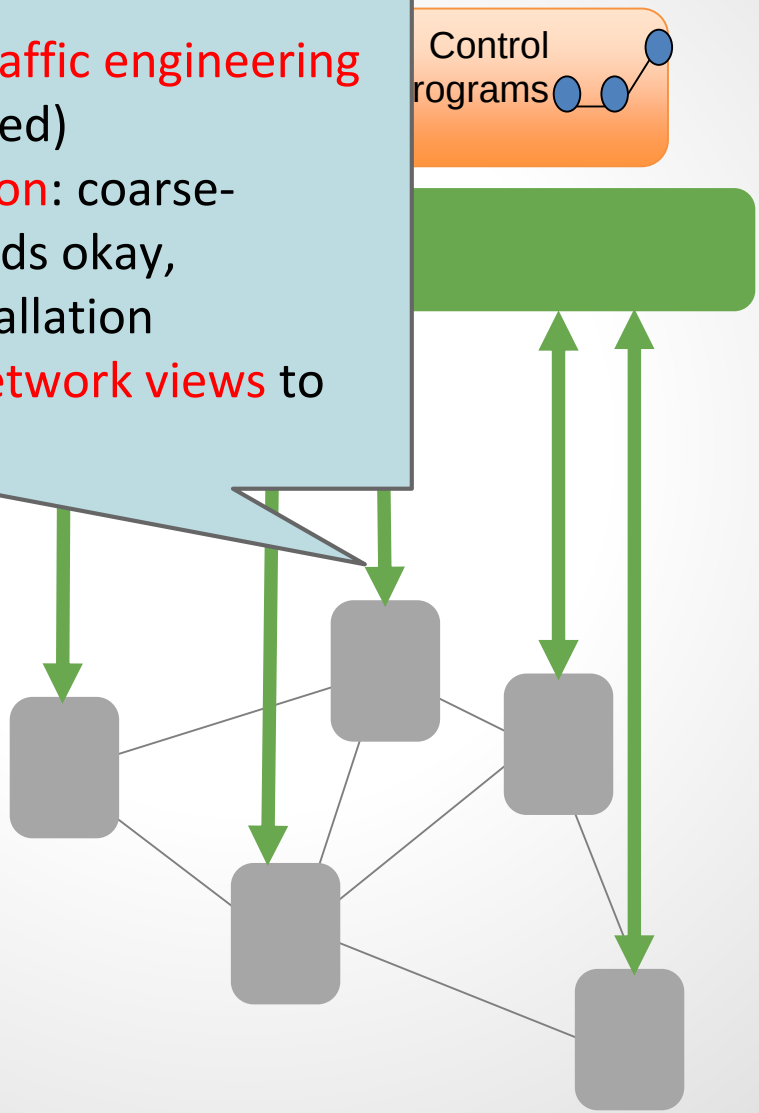
SDN = More Flexibilities…



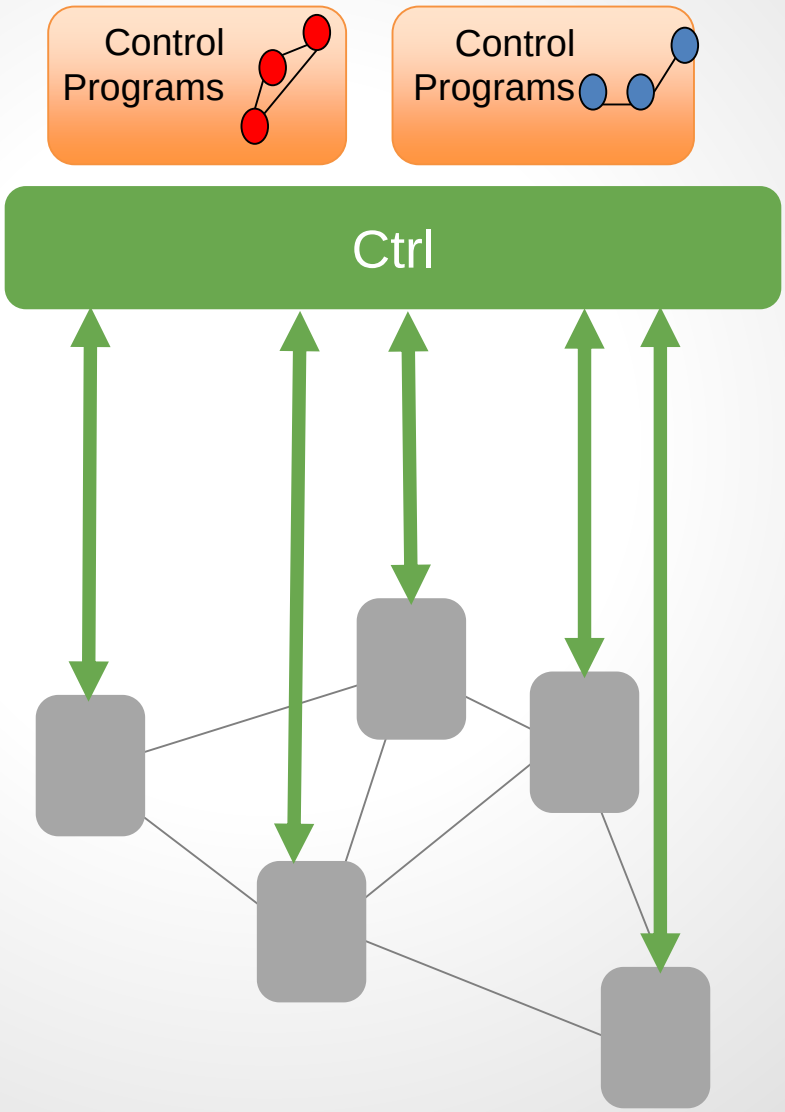
SDN = More Flexibilities…

Flexibility 4: OpenFlow is about generalization!

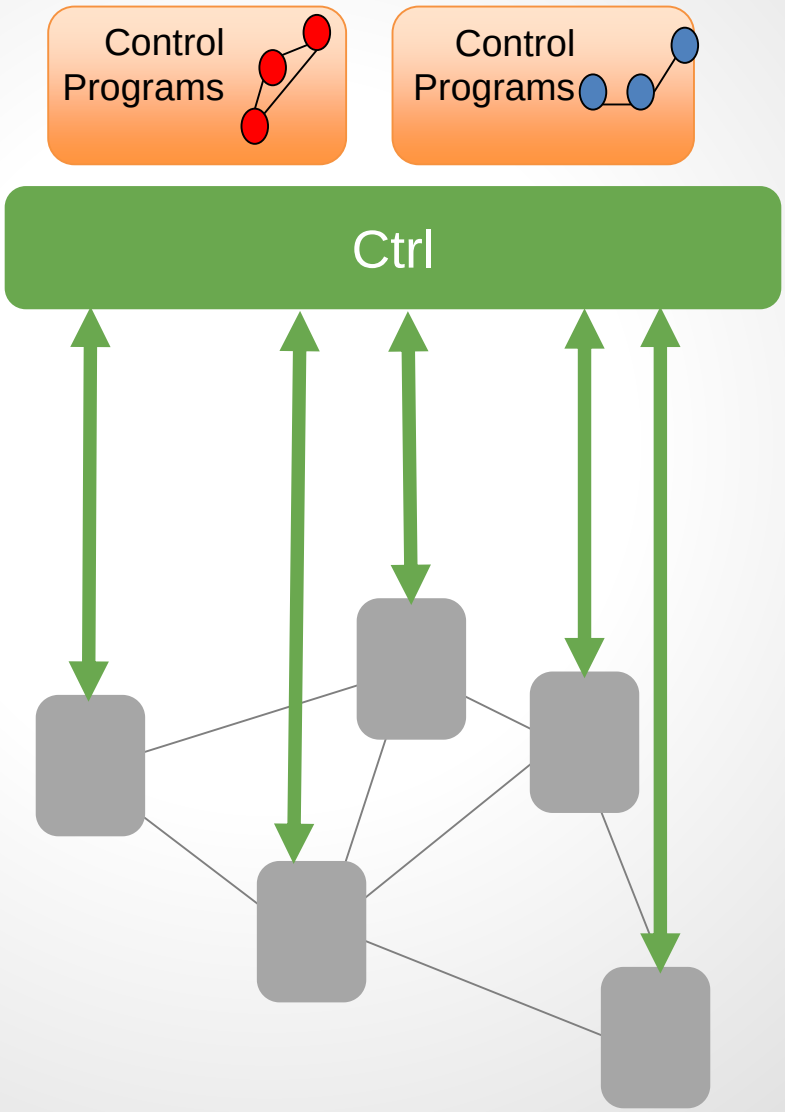
- Generalize devices (L2-L4: switches, routers, middleboxes)
- Generalize routing and traffic engineering (not only destination-based)
- Generalize flow-installation: coarse-grained rules and wildcards okay, proactive vs reactive installation
- Provide flexible logical network views to the application / tenant



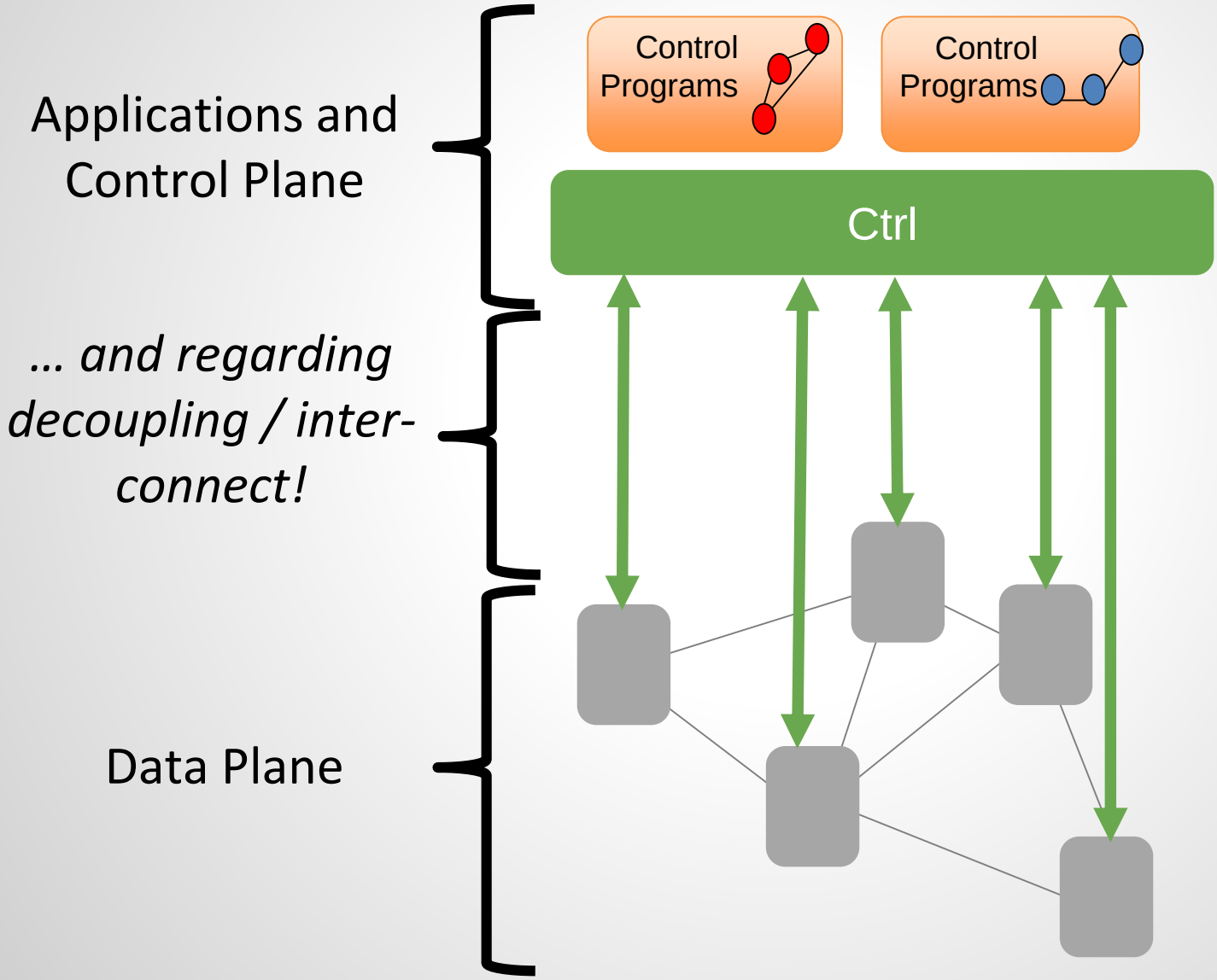
SDN = More Flexibilities…



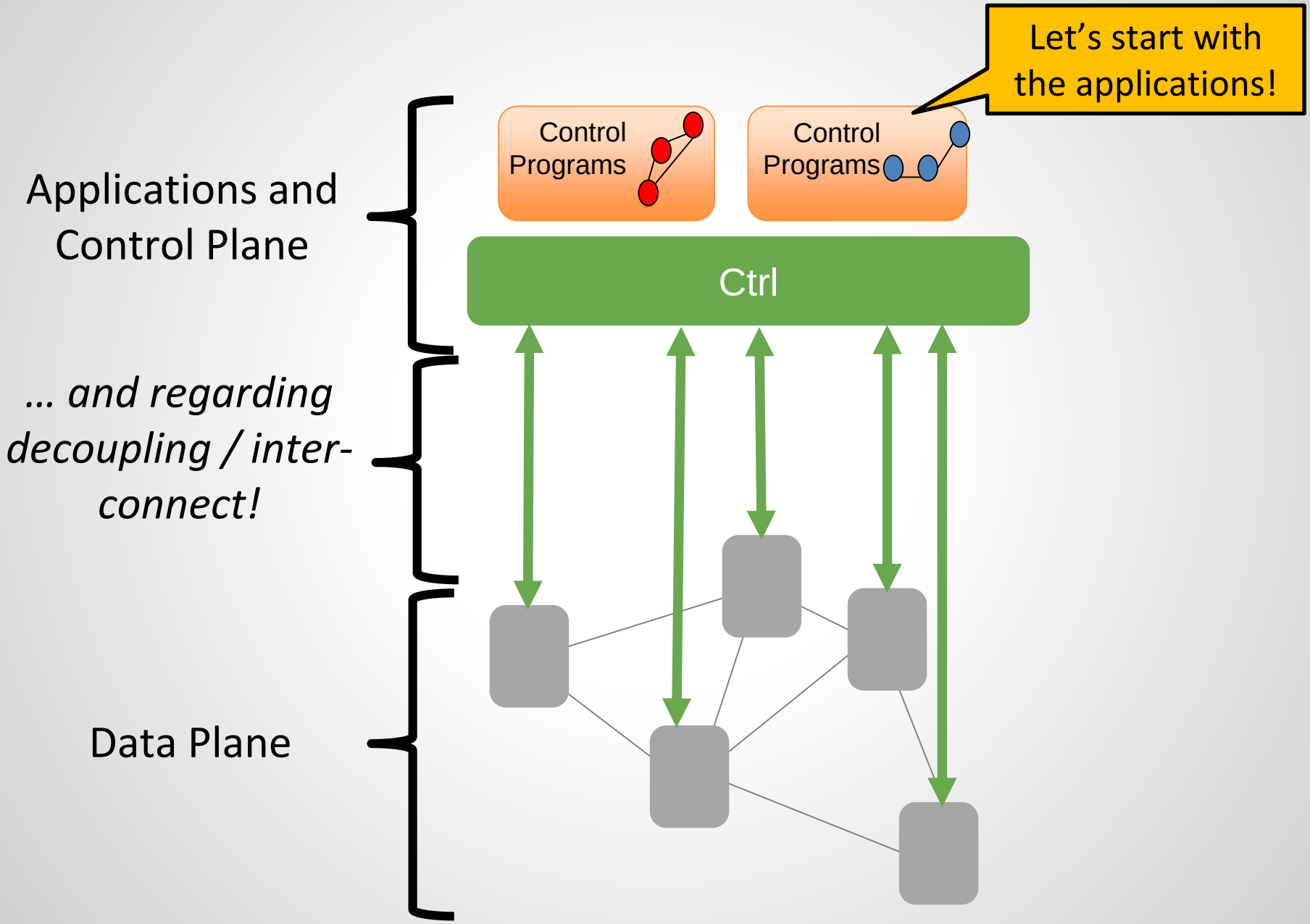
… But Also New Challenges!



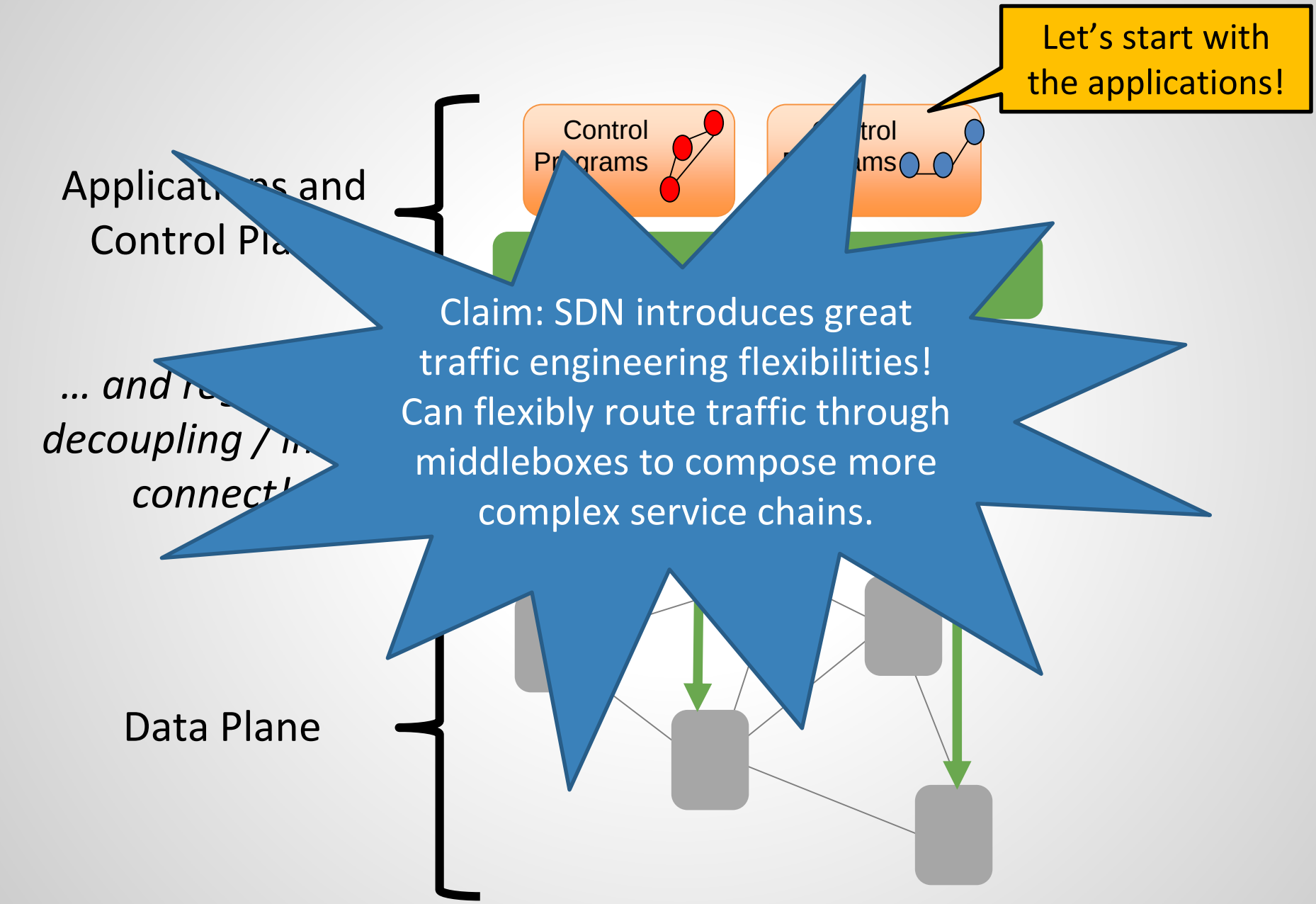
… But Also New Challenges!



… But Also New Challenges!

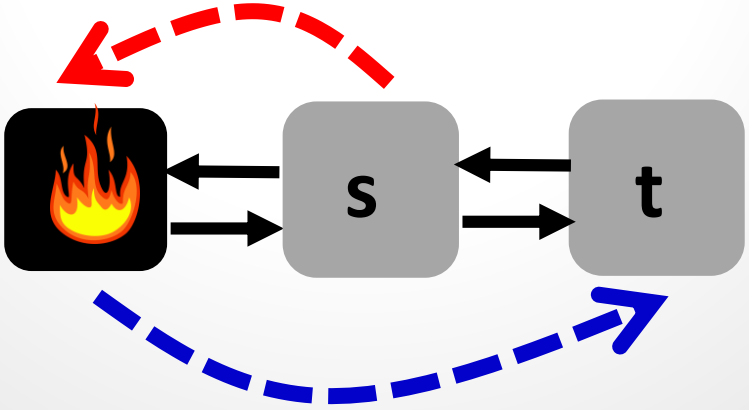


… But Also New Challenges!



Routing Through Waypoints

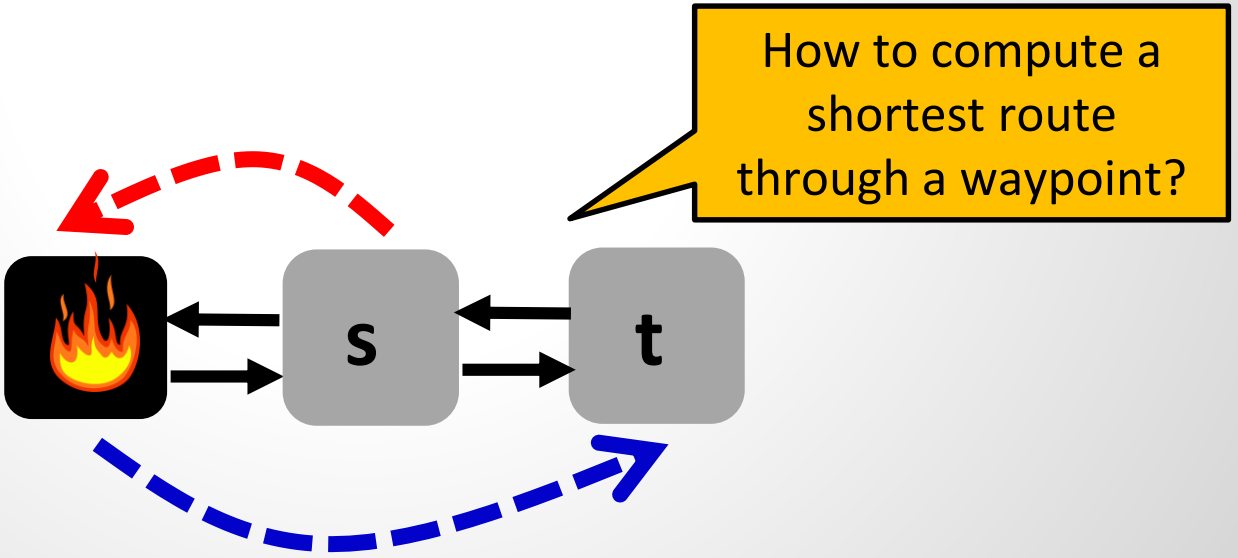
- Traditionally: routes form simple paths (e.g., shortest paths)
  - Traffic engineering is a classic and hard topic: not today ☺
- Novel aspect: routing through middleboxes may require more general paths, with loops: a walk



Routing Through Waypoints

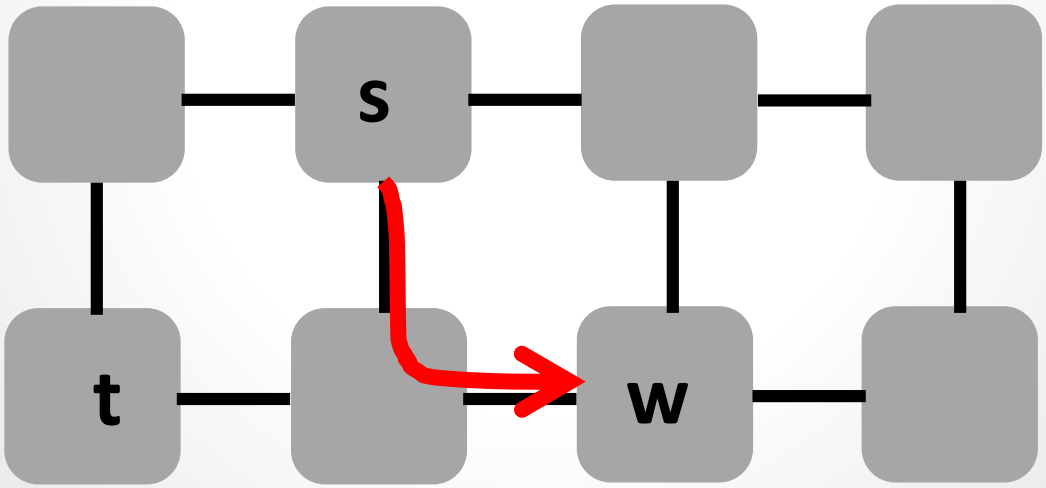
- Traditionally: routes form simple paths (e.g., shortest paths)
  - Traffic engineering is a classic and hard topic: not today ☺

- Novel aspect: routing through middleboxes may require more general paths, with loops: a walk



Routing or "Walking" Through Waypoints: A Deep Combinatorial Problem

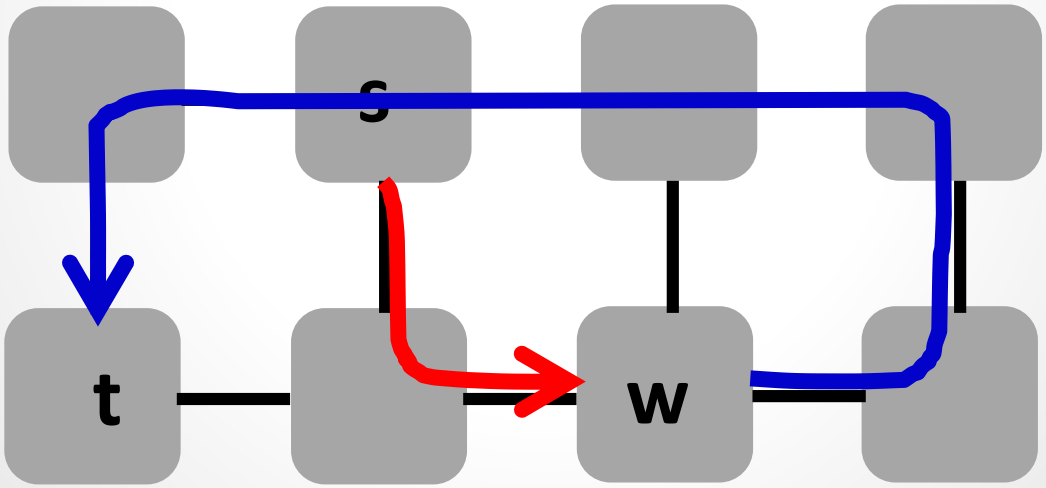
- Computing shortest routes through waypoints is non-trivial!



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

Routing or "Walking" Through Waypoints: A Deep Combinatorial Problem

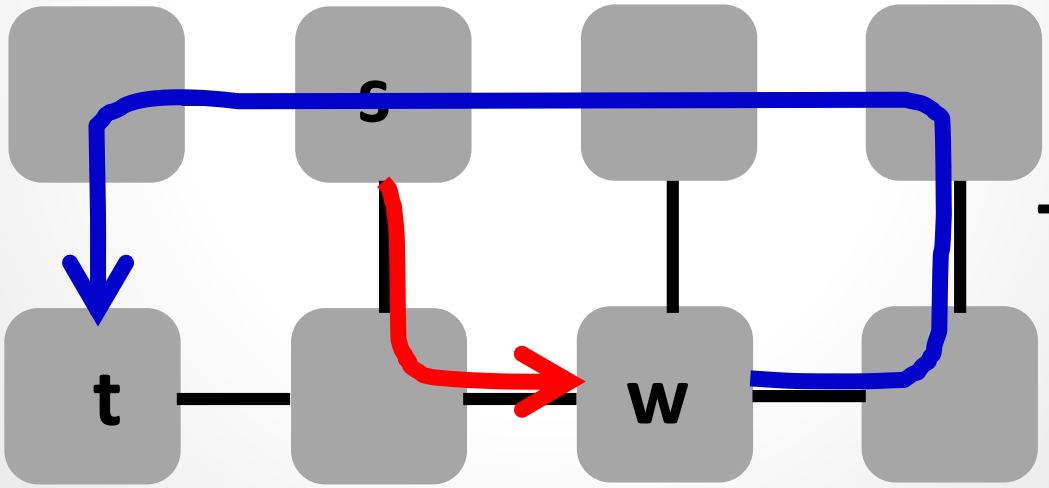
- Computing shortest routes through waypoints is non-trivial!



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

Routing or "Walking" Through Waypoints: A Deep Combinatorial Problem

- Computing shortest routes through waypoints is non-trivial!

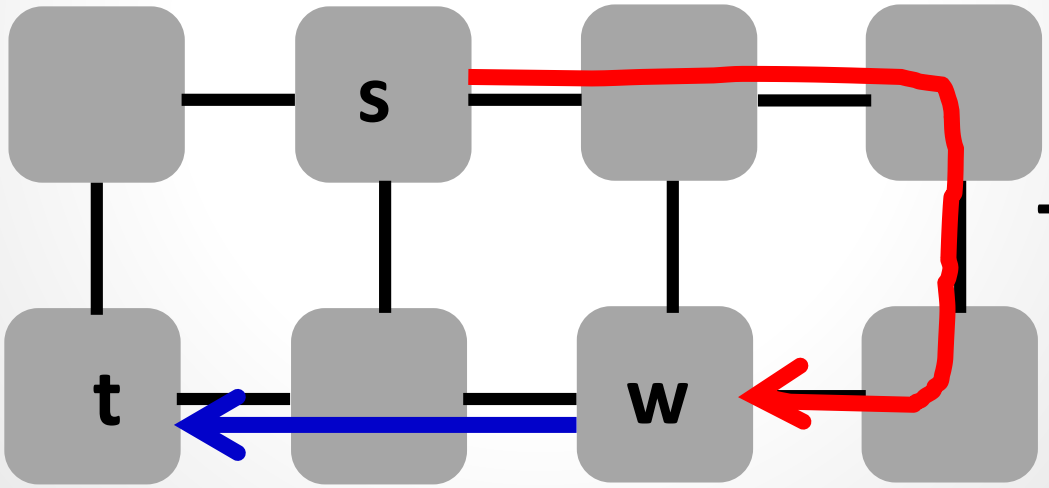


Total length: 2+6=8

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

Routing or "Walking" Through Waypoints: A Deep Combinatorial Problem

- Computing shortest routes through waypoints is non-trivial!

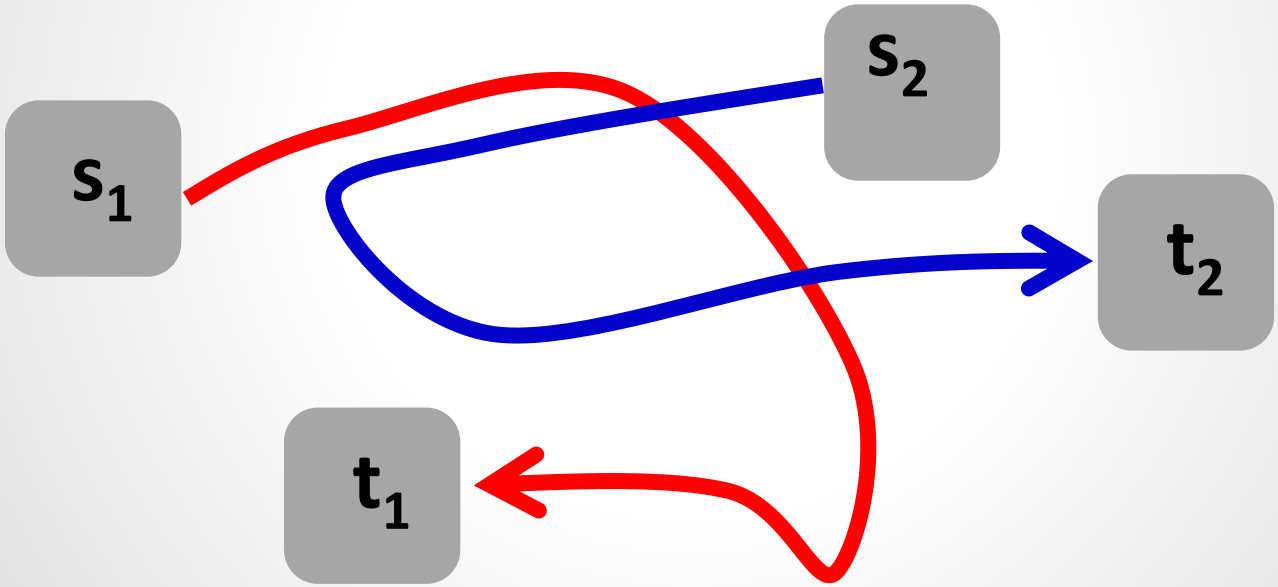


Total length: 4+2=6

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

Routing or "Walking" Through Waypoints: A Deep Combinatorial Problem

- Related to the 2-disjoint path problem: NP-hard on directed networks…



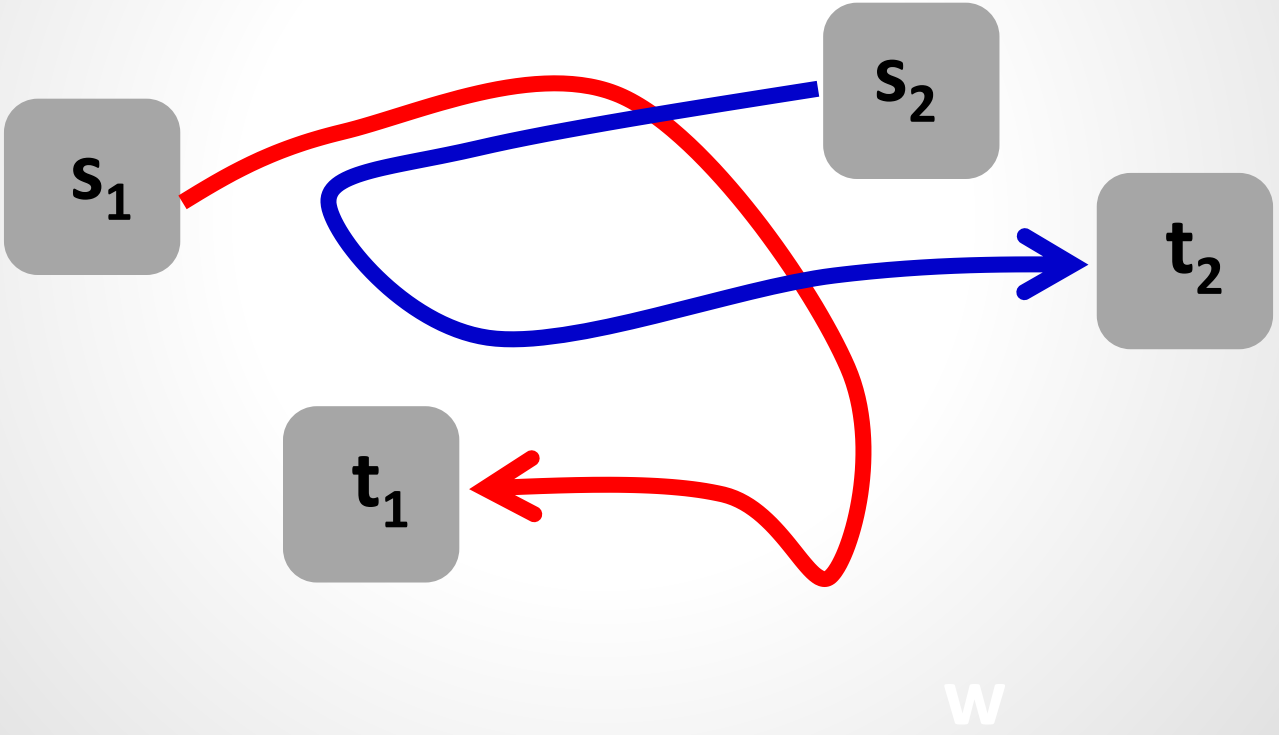
- … so is routing via a single waypoint!

Why?

Routing or "Walking" Through Waypoints: A Deep Combinatorial Problem

Reduction: *From* joint shortest paths $(s_1,t_1),(s_2,t_2)$

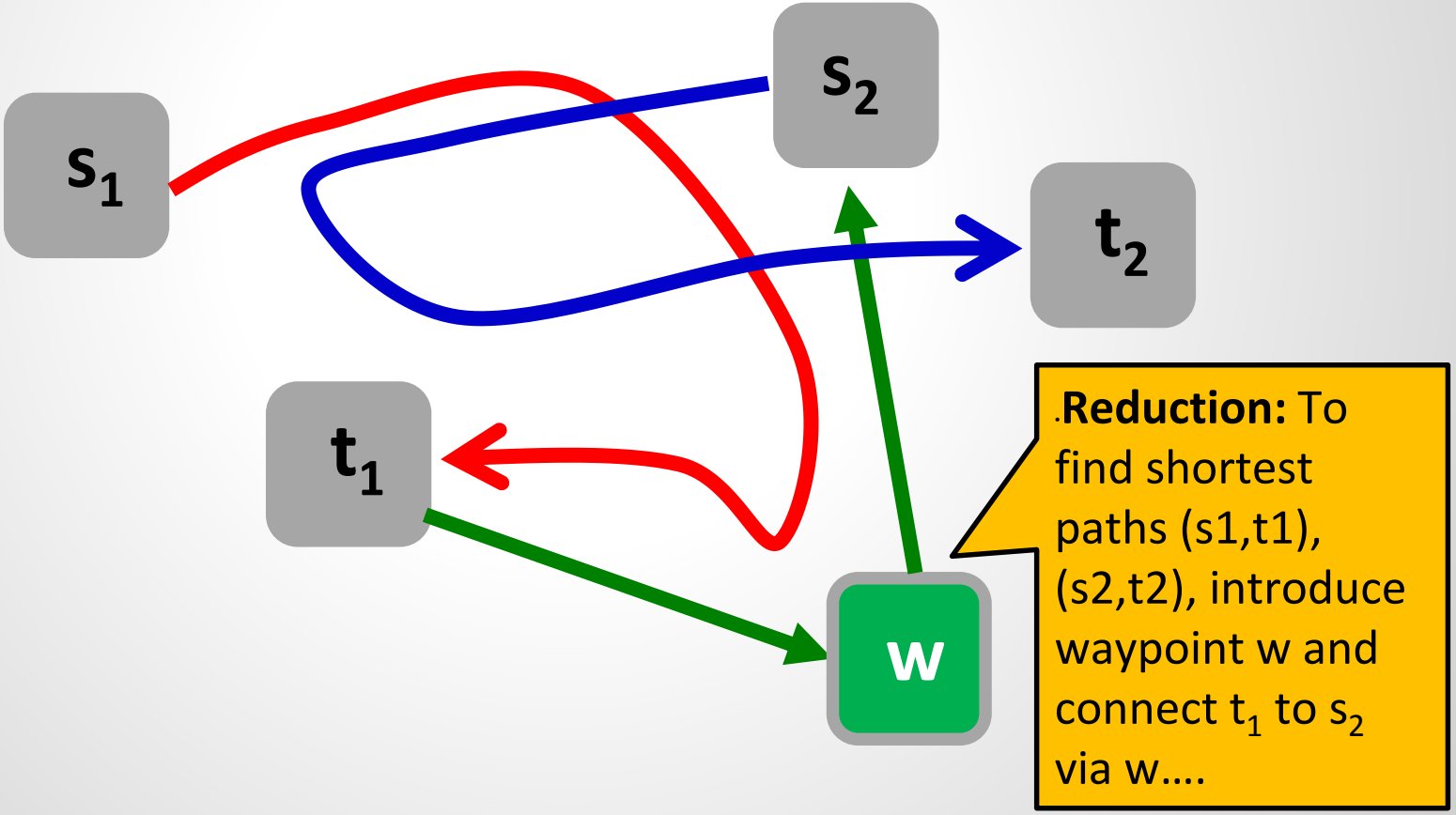
to shortest walk (s,w,t) problem



Routing or "Walking" Through Waypoints: A Deep Combinatorial Problem

Reduction: *From* joint shortest paths $(s_1,t_1),(s_2,t_2)$

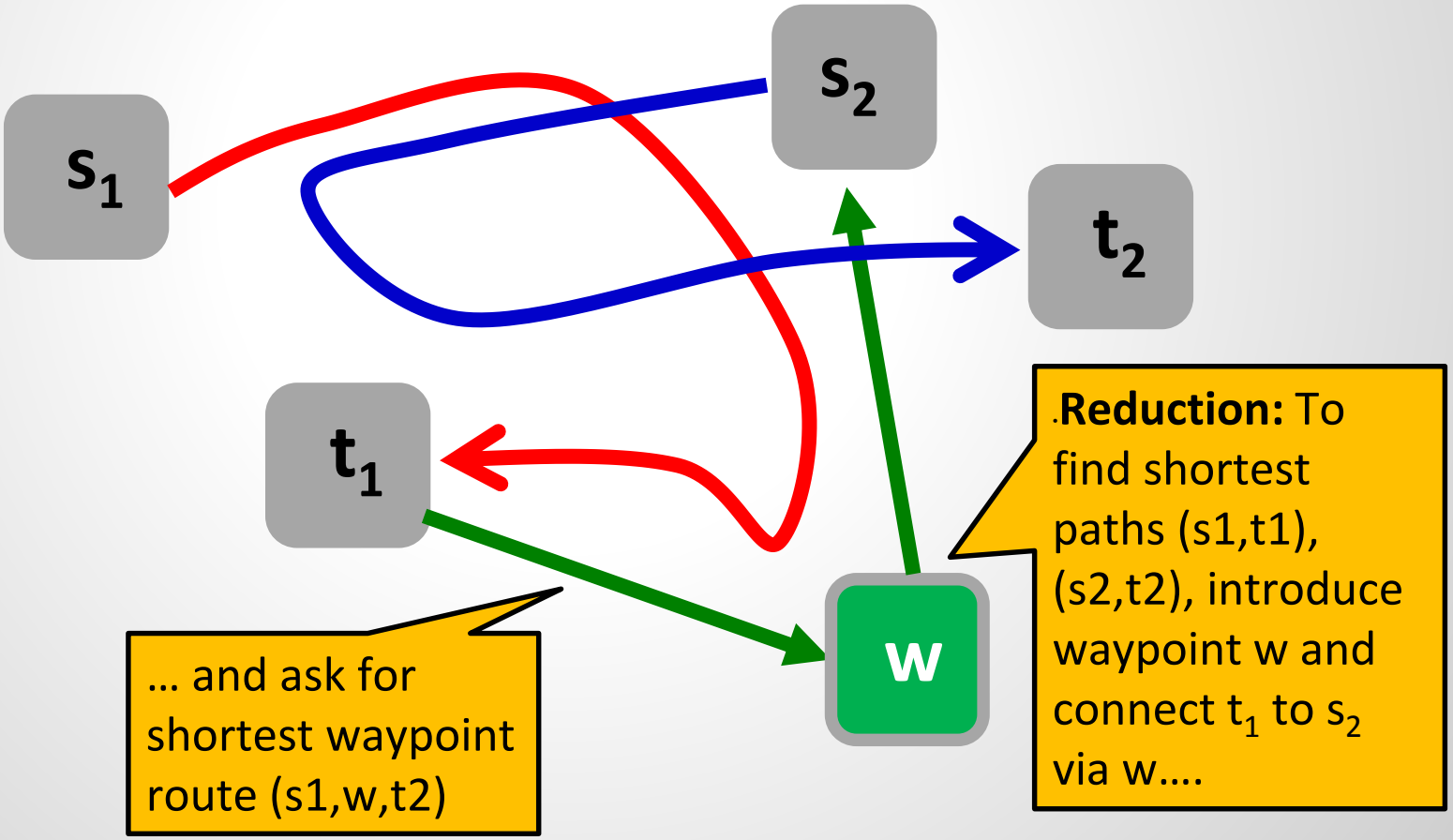
to shortest walk (s,w,t) problem



Routing or "Walking" Through Waypoints: A Deep Combinatorial Problem

Reduction: *From* joint shortest paths $(s_1,t_1),(s_2,t_2)$

to shortest walk (s,w,t) problem

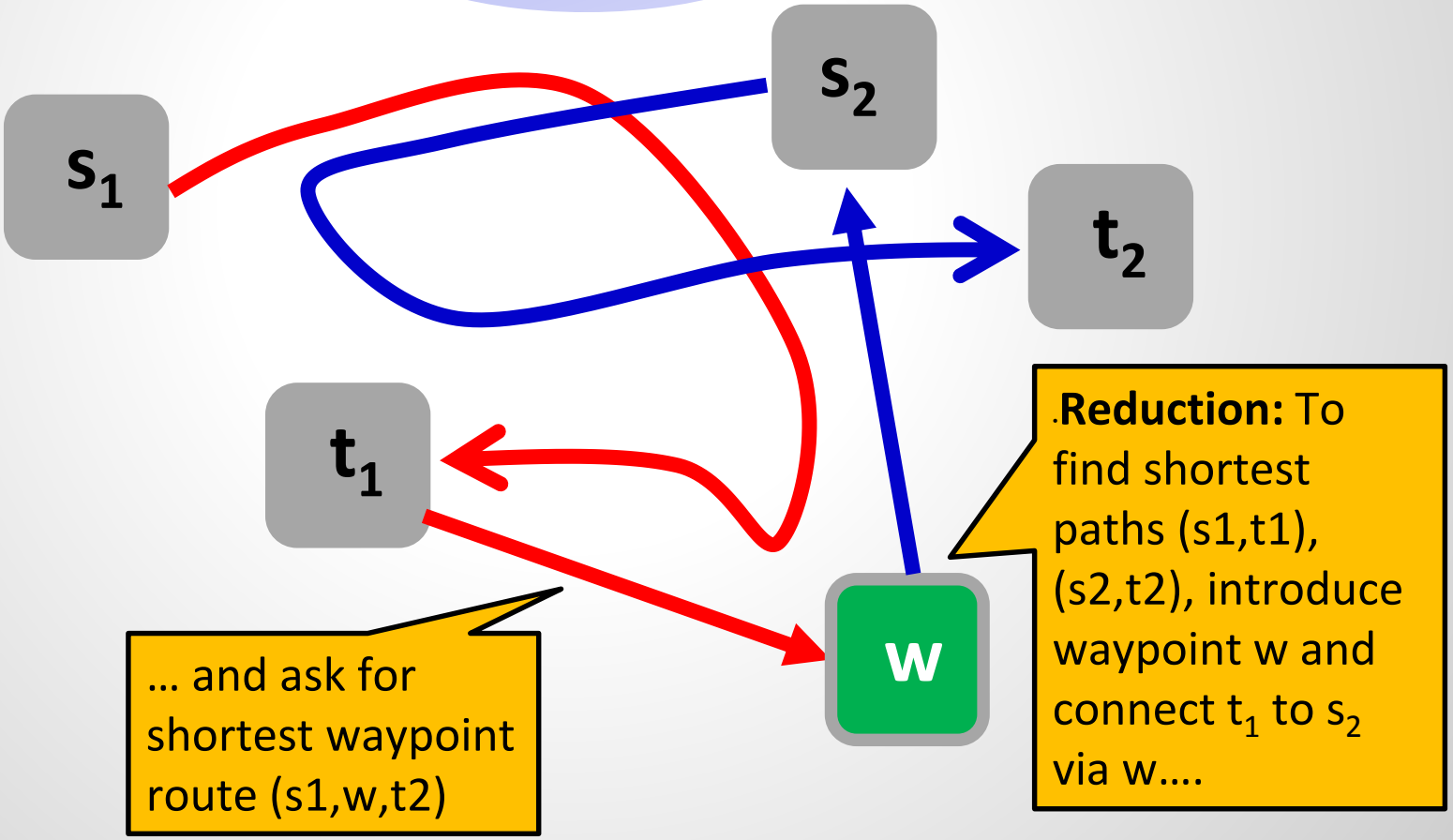


Routing or "Walking" Th[illegible]oints:

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!

Reduc

to shortest w



What about waypoint routes on undirected networks?

What about waypoint routes on undirected networks?

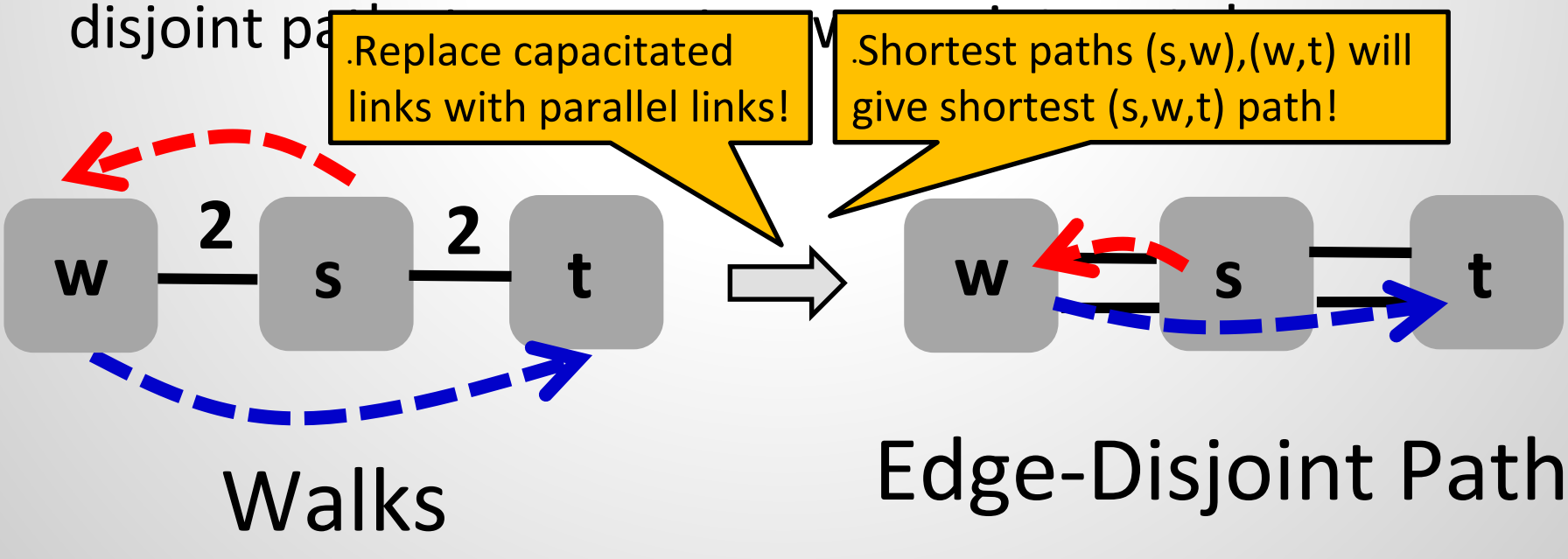
- Reduction from disjoint paths no longer works: disjoint paths problem not NP-hard on undirected networks
  - Feasible paths can be computed: still a very deep problem
  - Shortest paths: recent breakthrough (polytime randomized algorithm)

- Indeed, algorithm exists: We can reduce to edge-disjoint paths to compute a waypoint route!

What about waypoint routes on undirected networks?

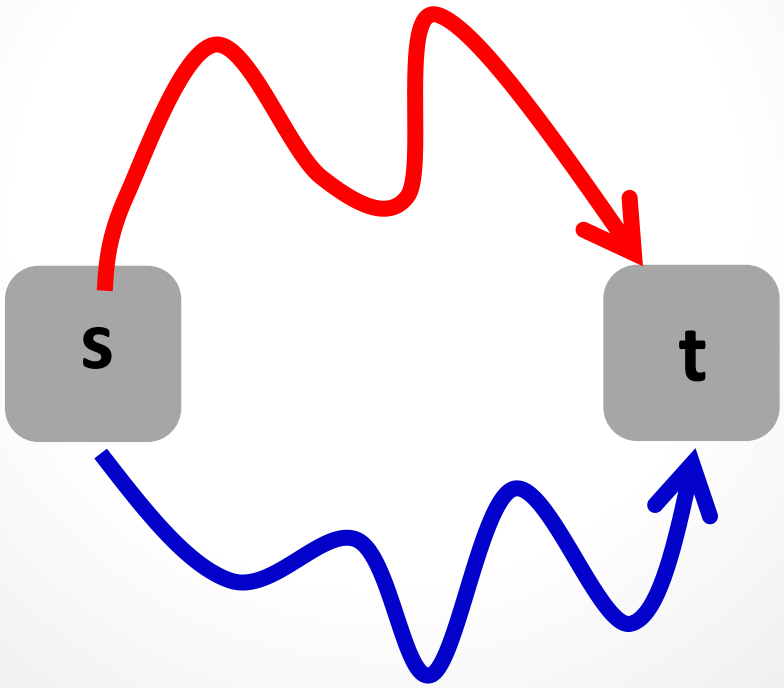
- Reduction from disjoint paths no longer works: disjoint paths problem not NP-hard on undirected networks
  - Feasible paths can be computed: still a very deep problem
  - Shortest paths: recent breakthrough (polytime randomized algorithm)

- Indeed, algorithm exists: We can reduce to edge-disjoint pa



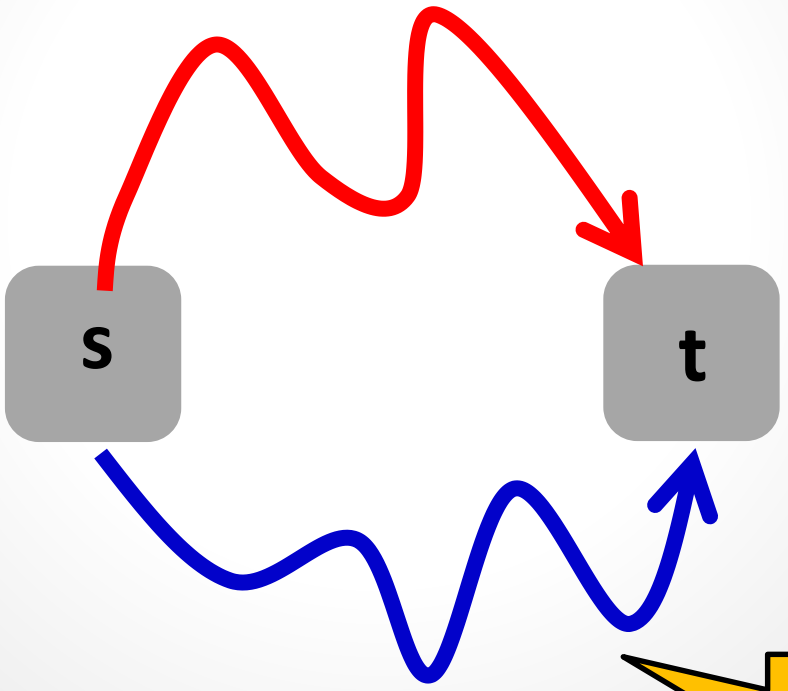
We Can Do Even Faster Waypoint Routing on Undirected Networks: Suurballe's Algorithm

- Suurballe's algorithm: finds two shortest paths between same endpoints:



We Can Do Even Faster Waypoint Routing on Undirected Networks: Suurballe's Algorithm

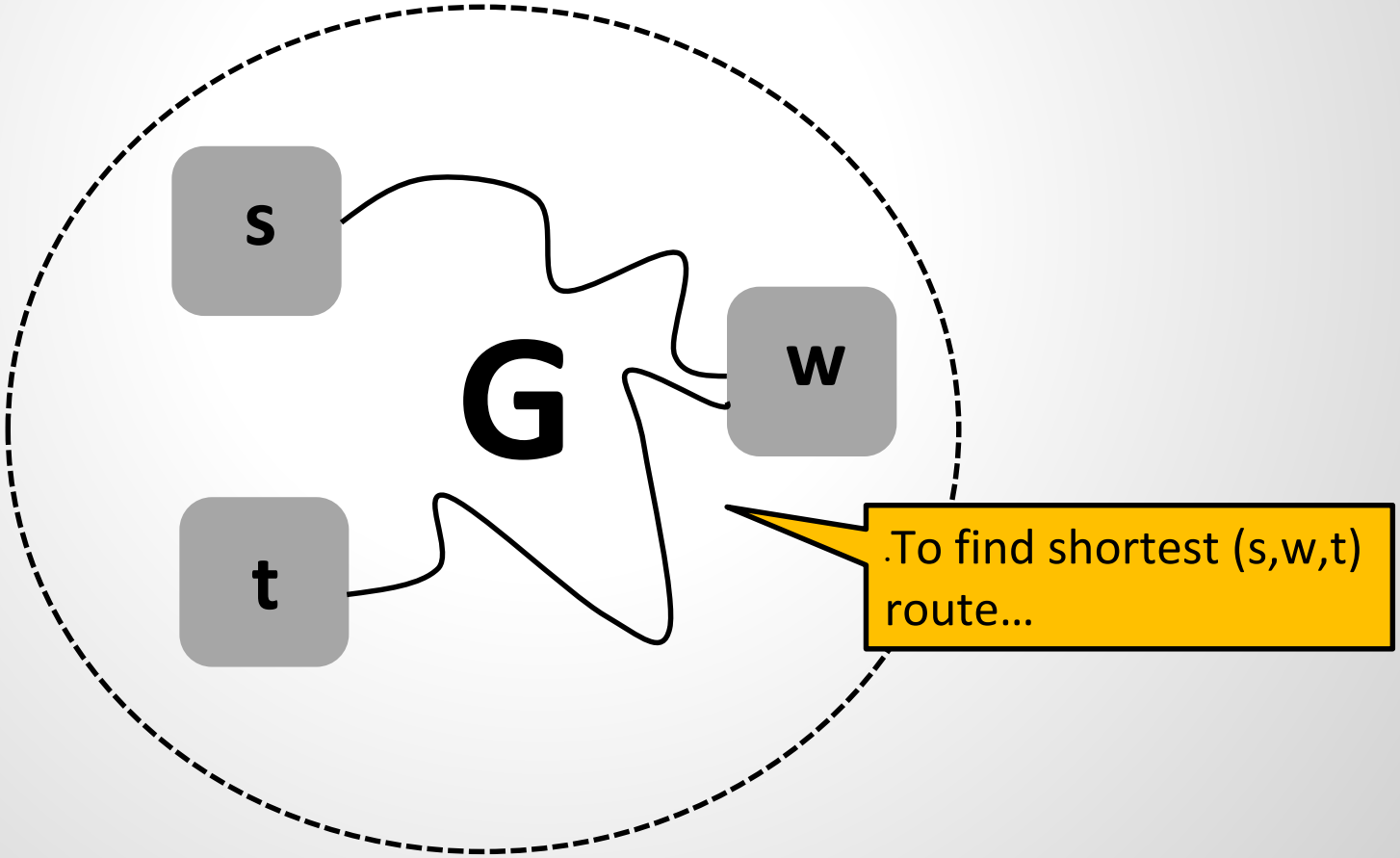
- Suurballe's algorithm: finds two shortest paths between same endpoints:



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

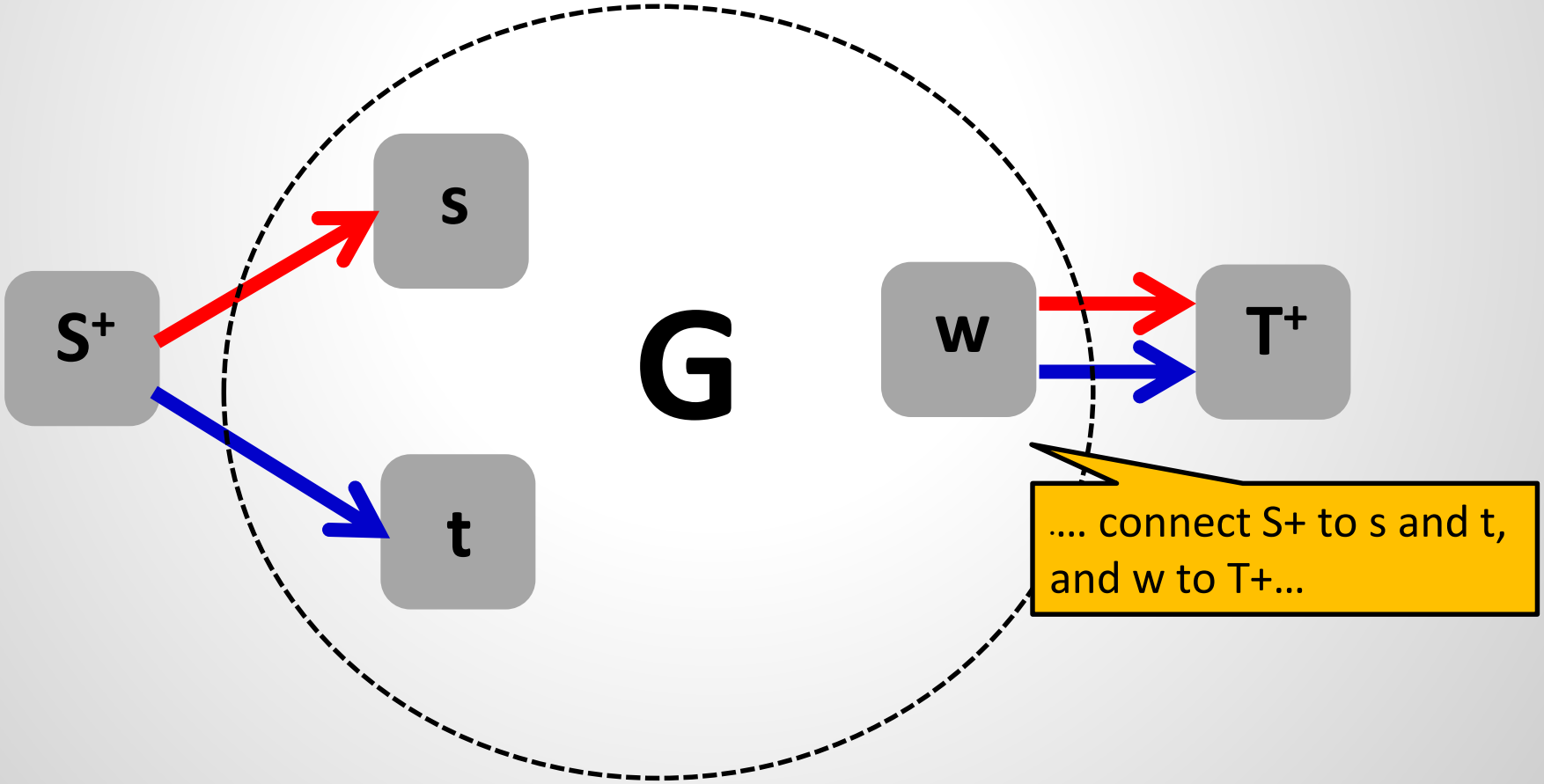
Waypoint Routing on Steroids

- Reduction to Suurballe's algorithm:



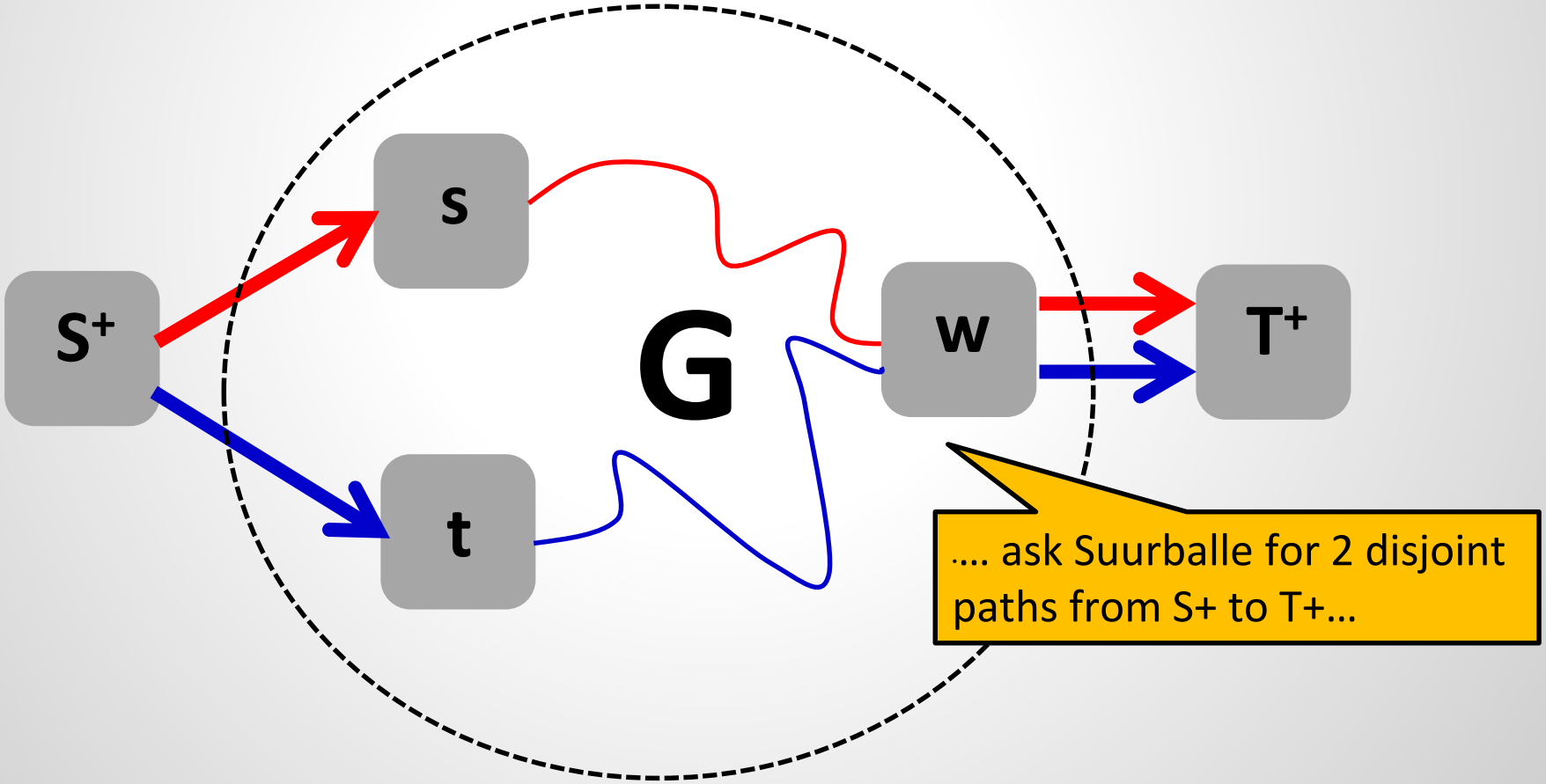
Waypoint Routing on Steroids

- Reduction to Suurballe's algorithm:



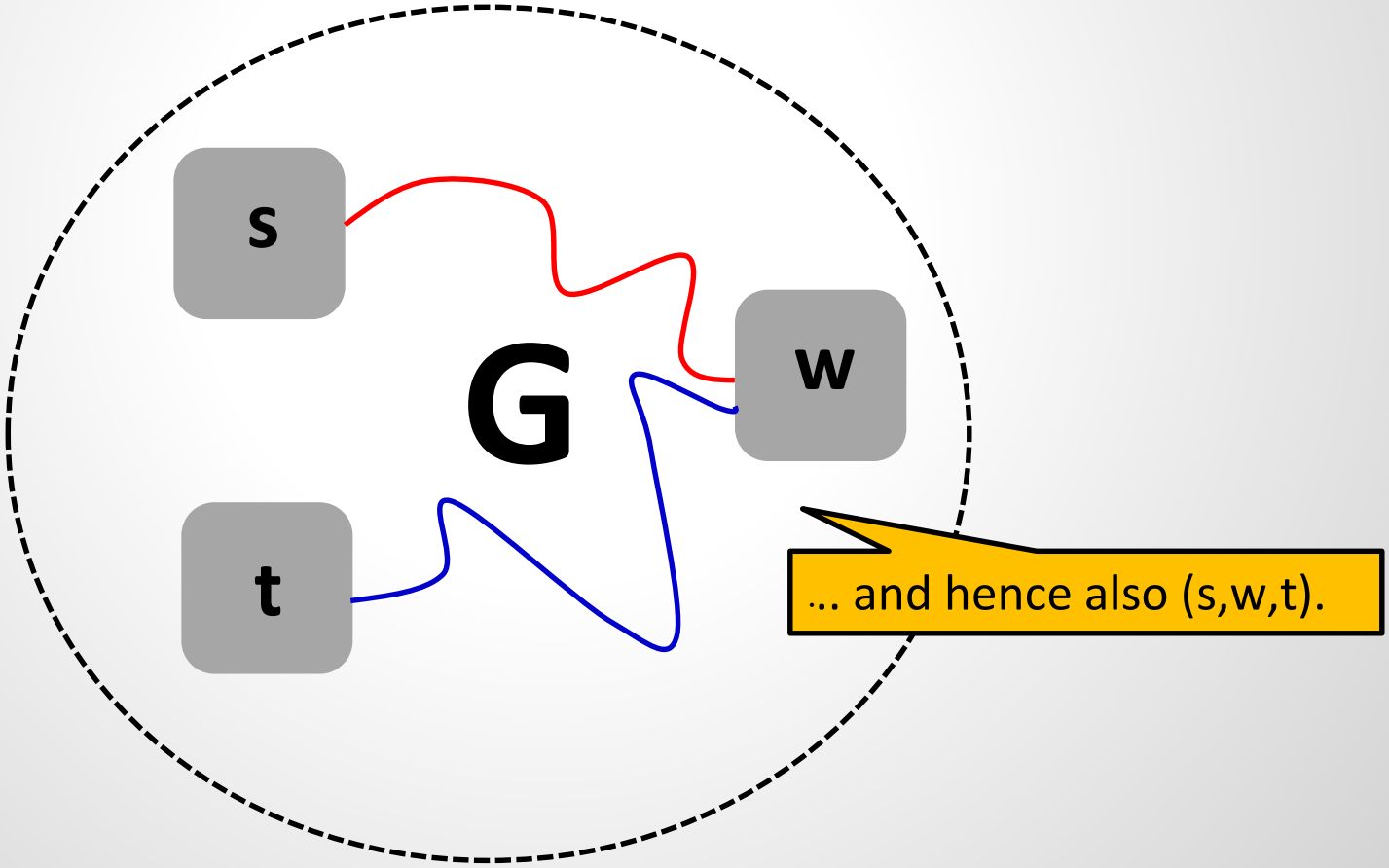
Waypoint Routing on Steroids

- Reduction to Suurballe's algorithm:



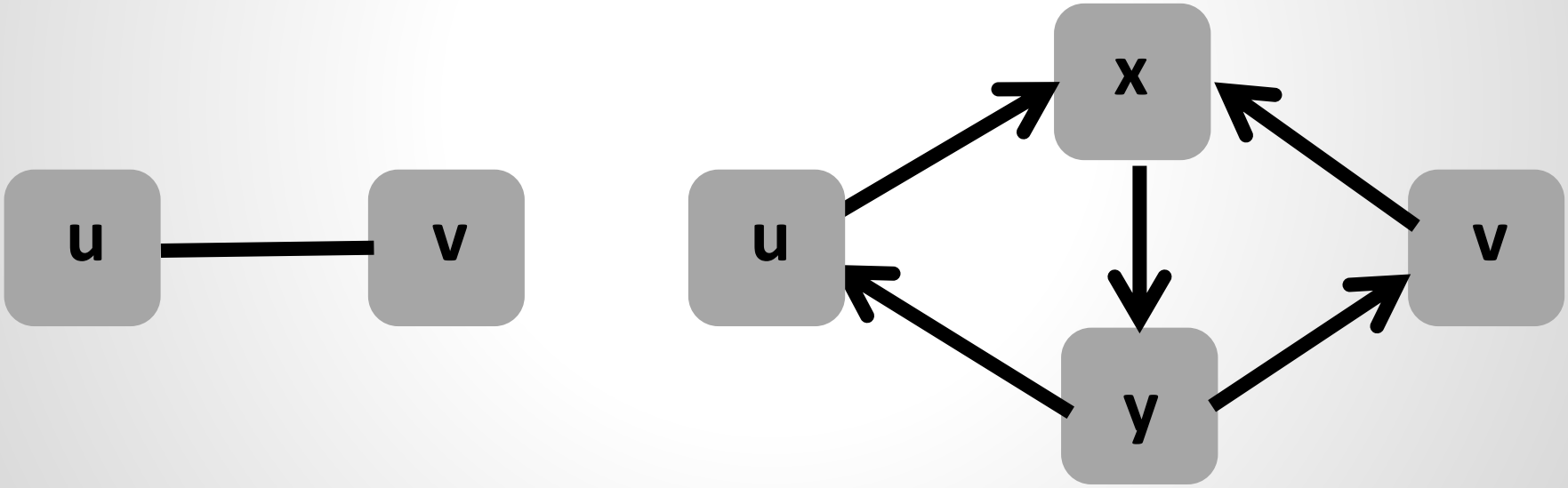
Waypoint Routing on Steroids

- Reduction to Suurballe's algorithm:



Remarks: Under the rug…

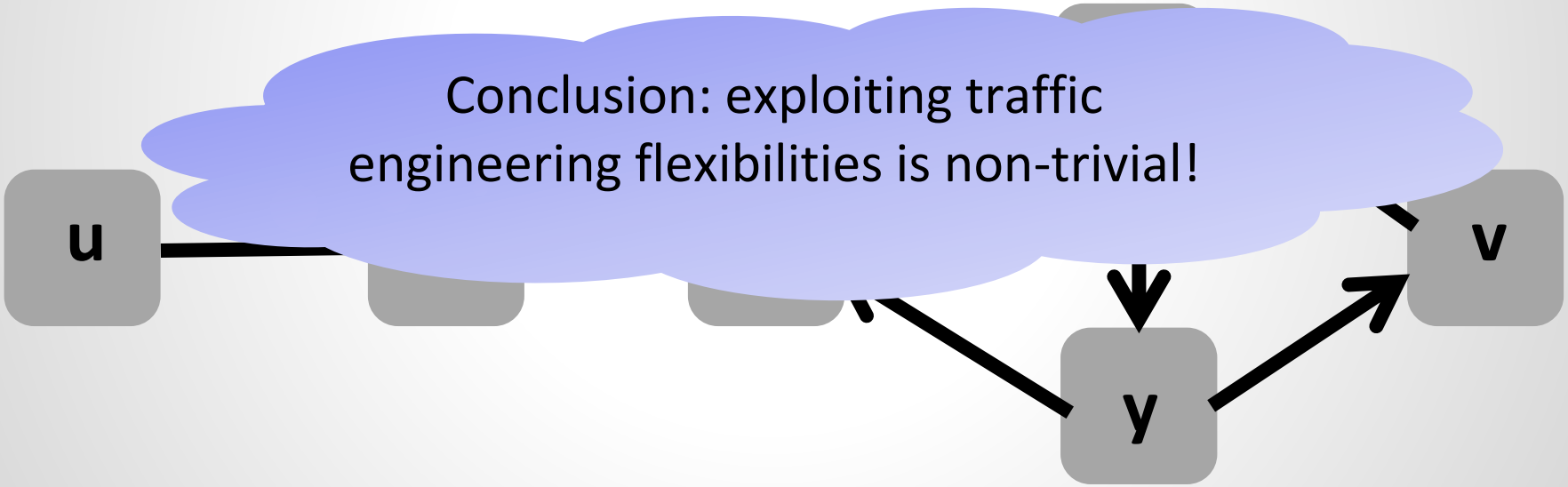
- **Remark 1:** We have not talked about directions but does not matter on undirected graph
- **Remark 2:** Suurballe is for directed graphs, so need gadget to transform problem in right form:



- **Remark 3:** Suurballe: vertex disjoint
  - Suurballe & Tarjan: edge disjoint

Remarks: Under the rug…

- **Remark 1:** We have not talked about directions but does not matter on undirected graph
- **Remark 2:** Suurballe is for directed graphs, so need gadget to transform problem in right form:



- **Remark 3:** Suurballe: vertex disjoint
  - Suurballe & Tarjan: edge disjoint

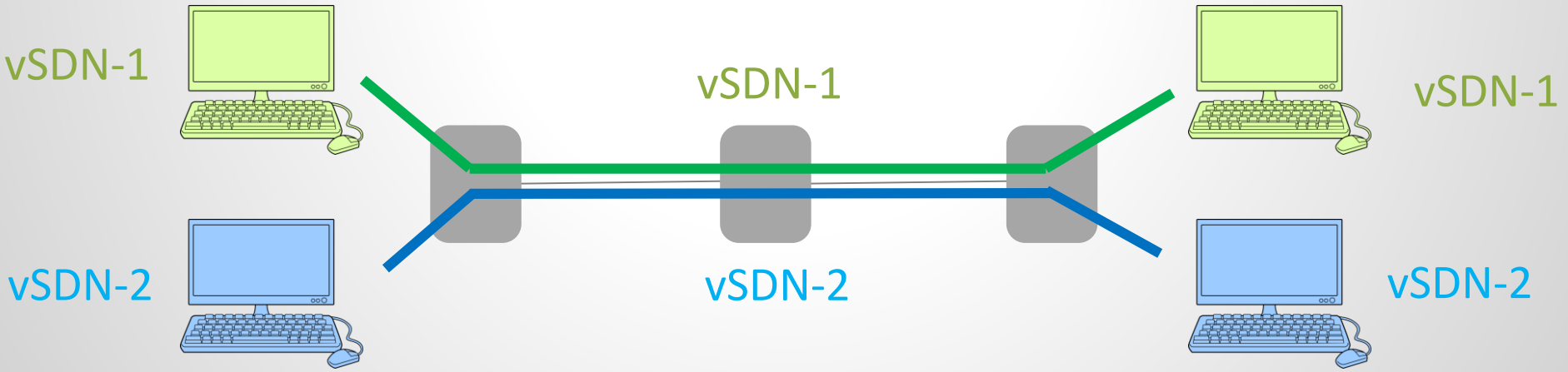
Further Reading

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


Next claim: Can use SDN+NV to make bandwidth reservations and get predictable network performance!

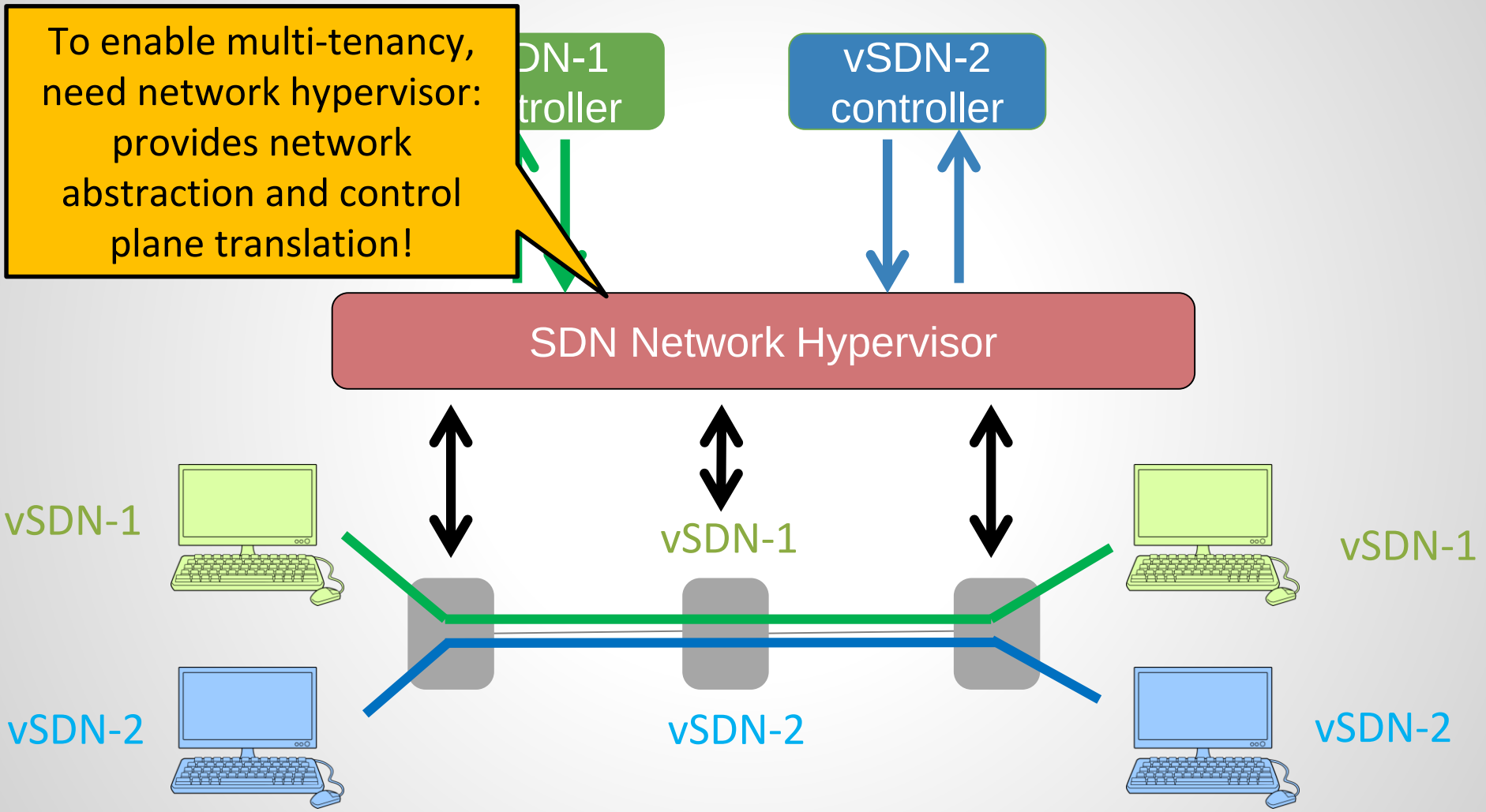
Predictable Performance:
About More Than Bandwidth Reservations!

Predictable Performance: About More Than Bandwidth Reservations!



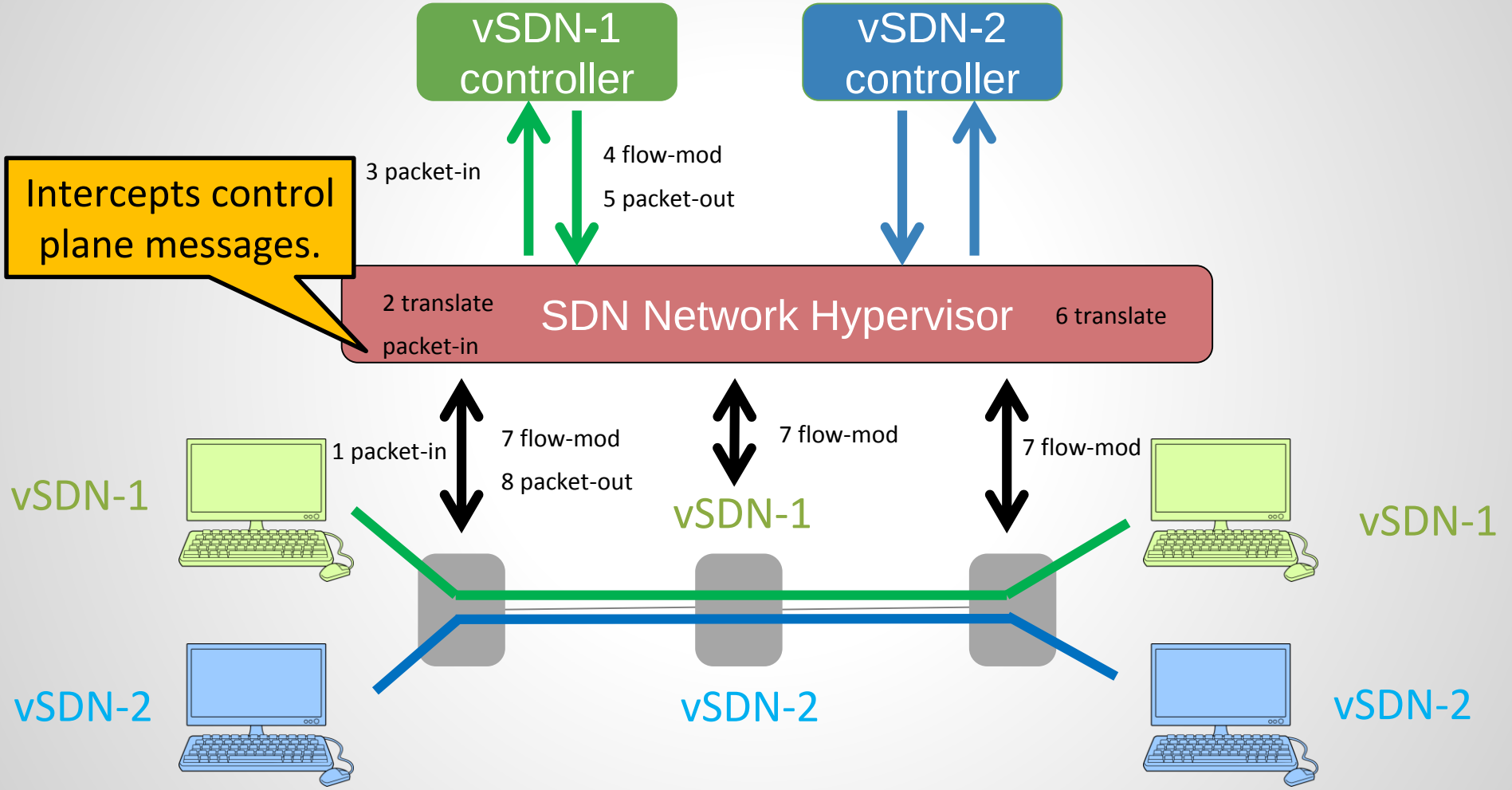
An Experiment: 2 vSDNs with bw guarantee!

Predictable Performance: About More Than Bandwidth Reservations!



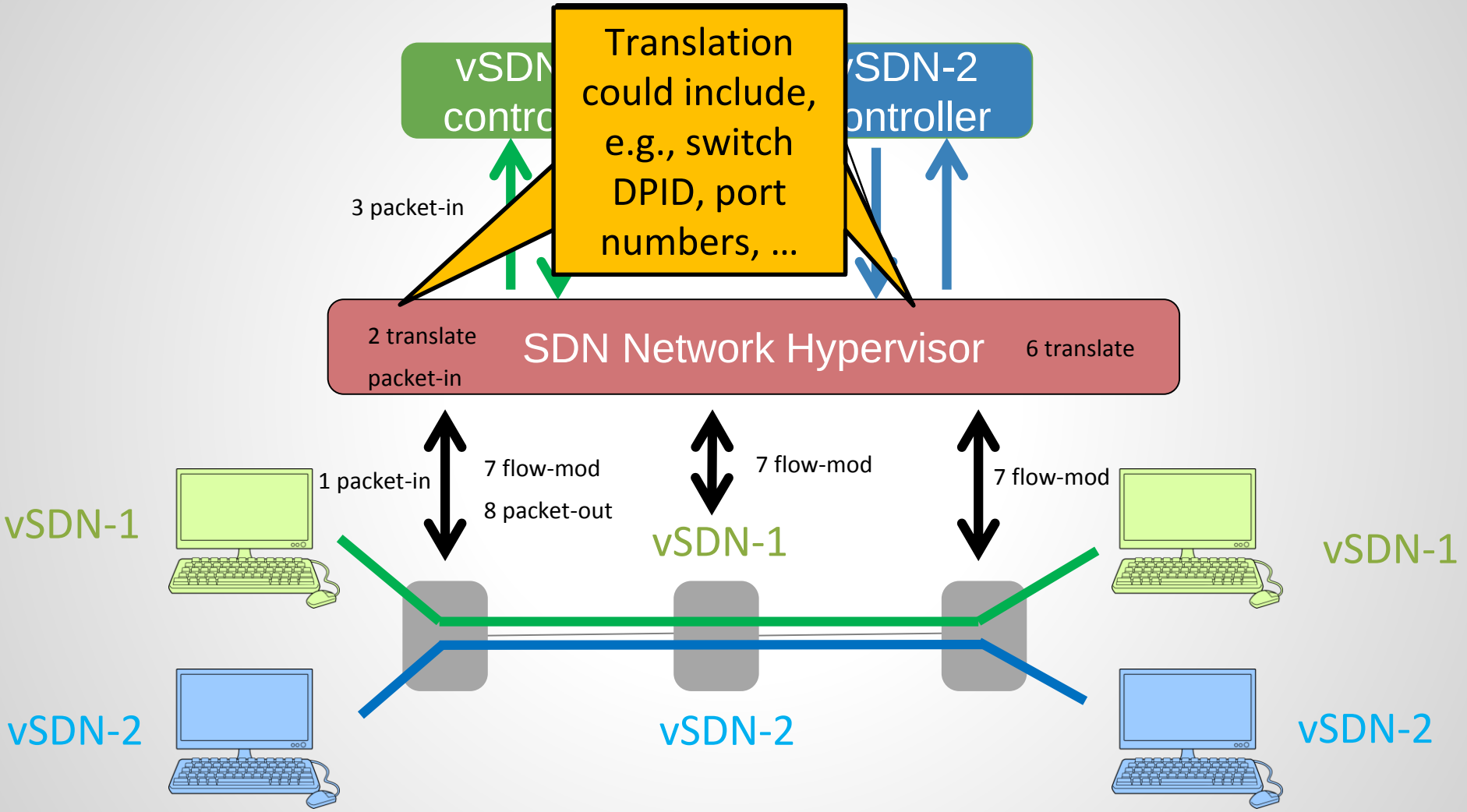
An Experiment: 2 vSDNs with bw guarantee!

Predictable Performance: About More Than Bandwidth Reservations!



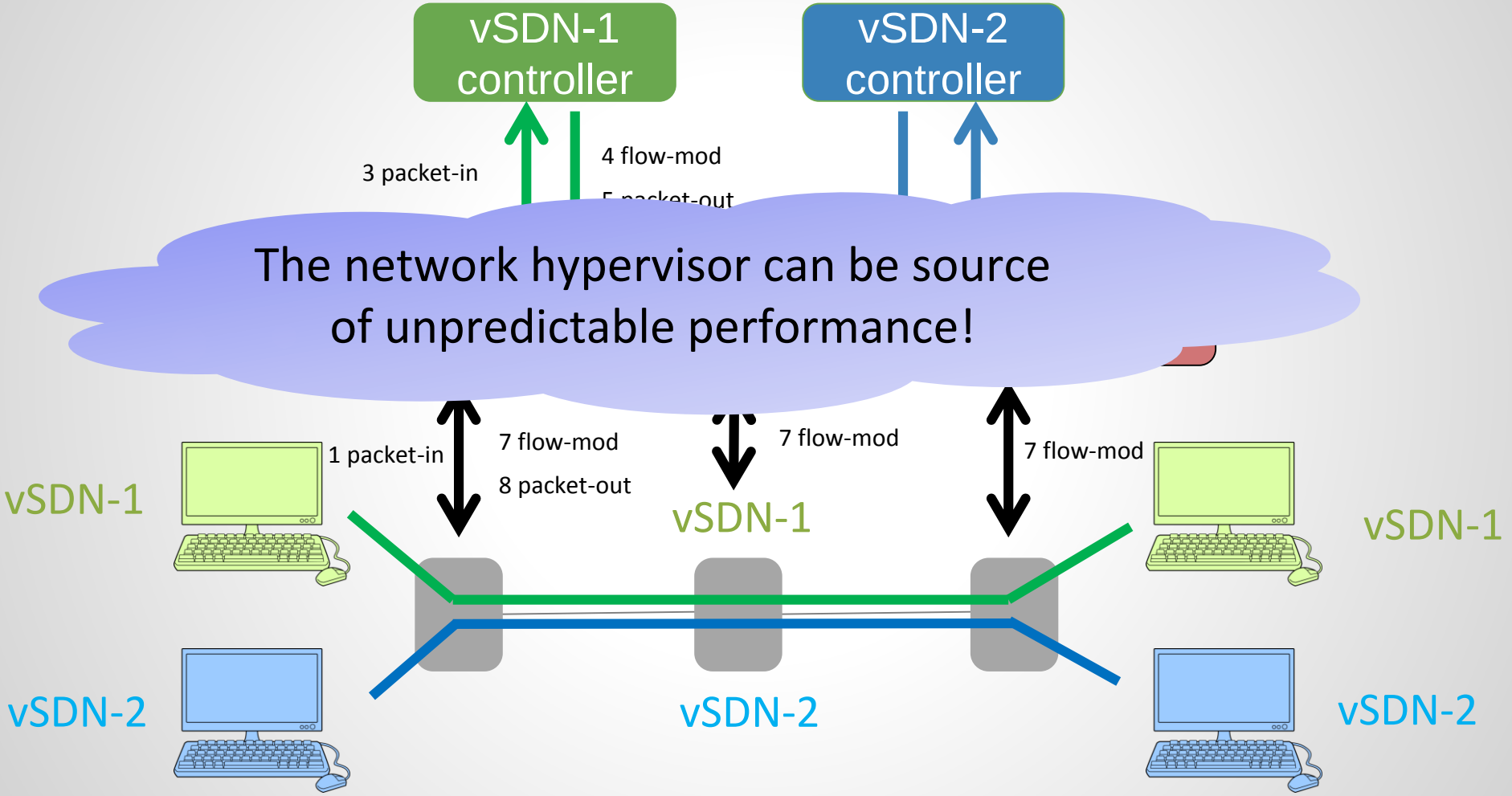
An Experiment: 2 vSDNs with bw guarantee!

Predictable Performance: About More Than Bandwidth Reservations!



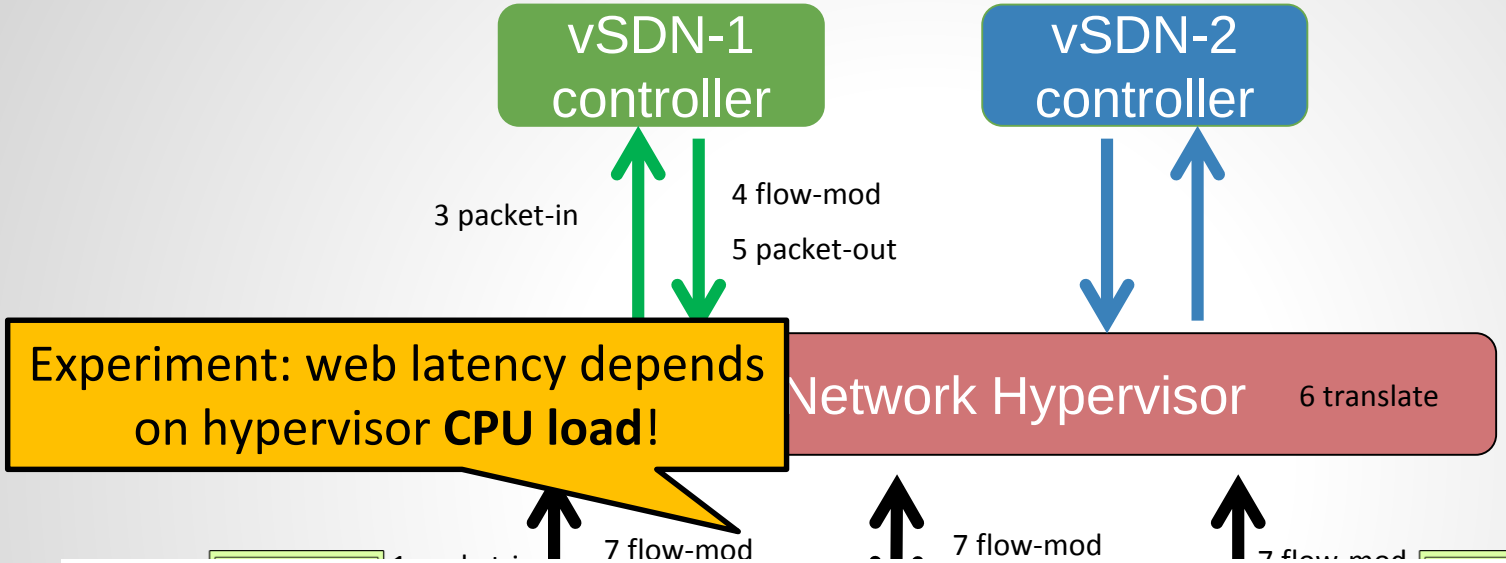
An Experiment: 2 vSDNs with bw guarantee!

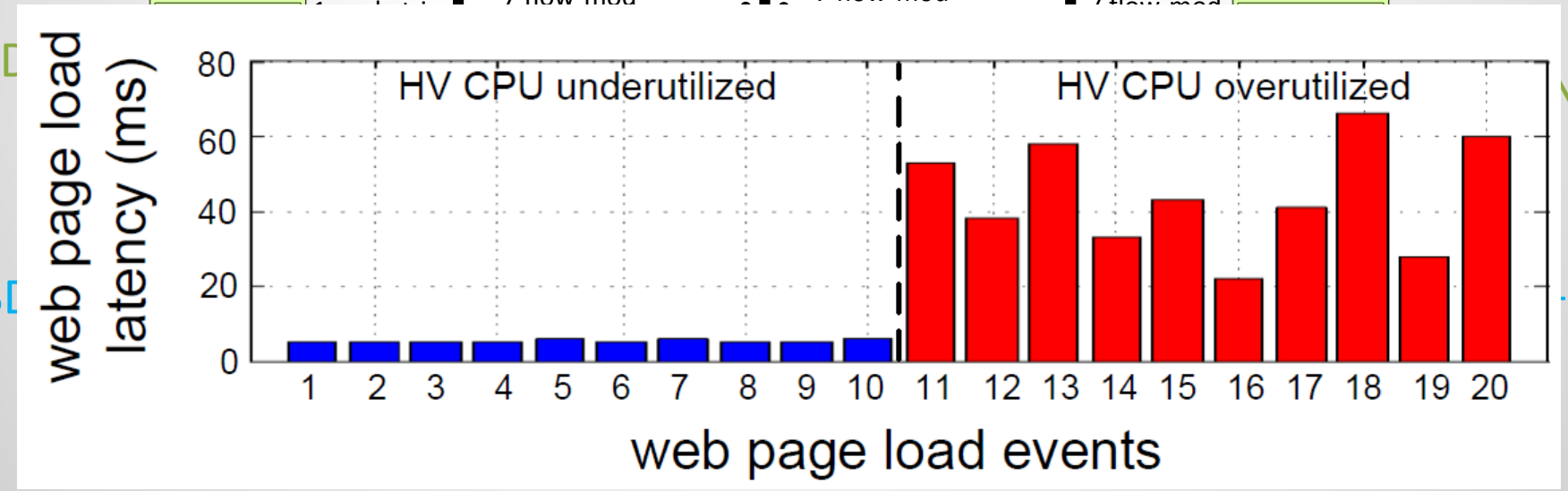
Predictable Performance: About More Than Bandwidth Reservations!



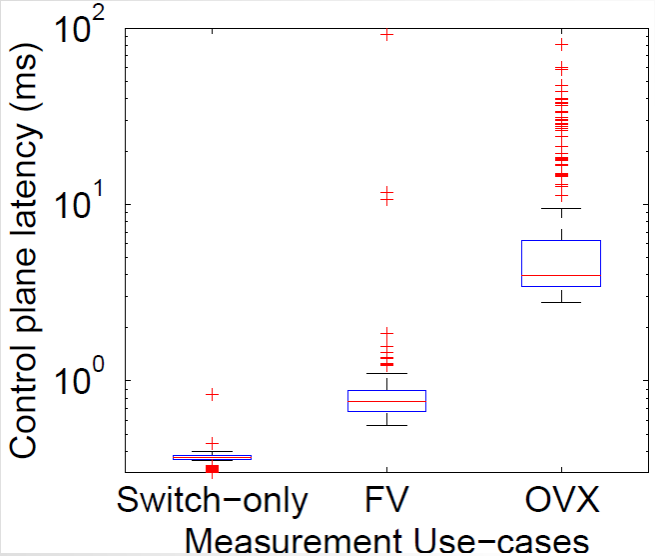
An Experiment: 2 vSDNs with bw guarantee!

Predictable Performance: About More Than Bandwidth Reservations!

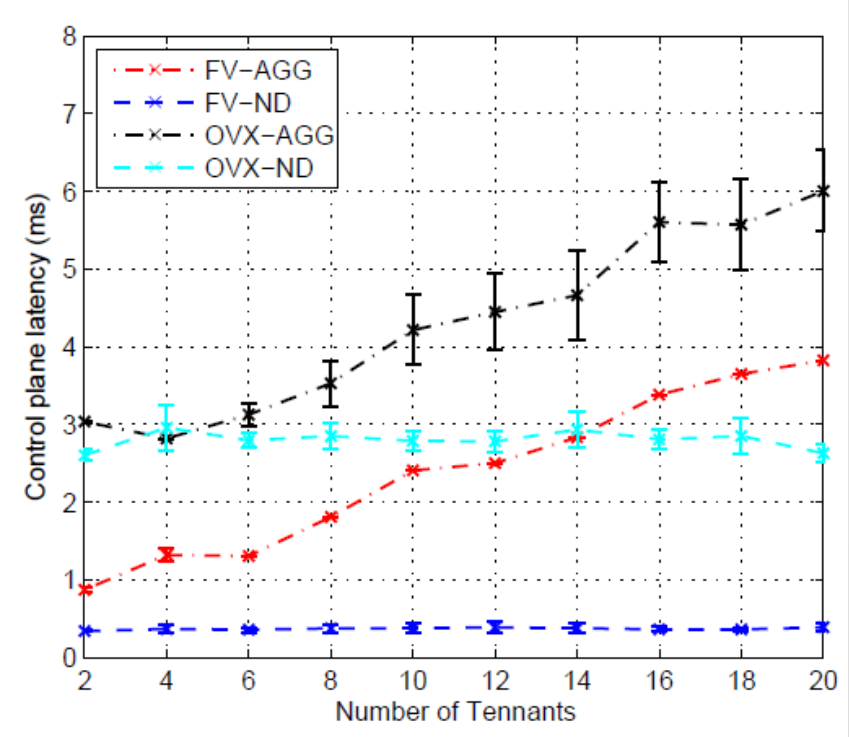




What you need to know about your network hypervisor!



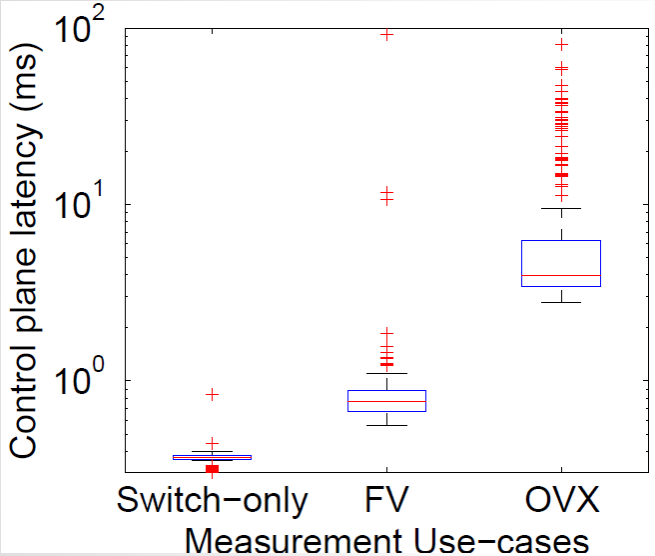
… number of tenants…



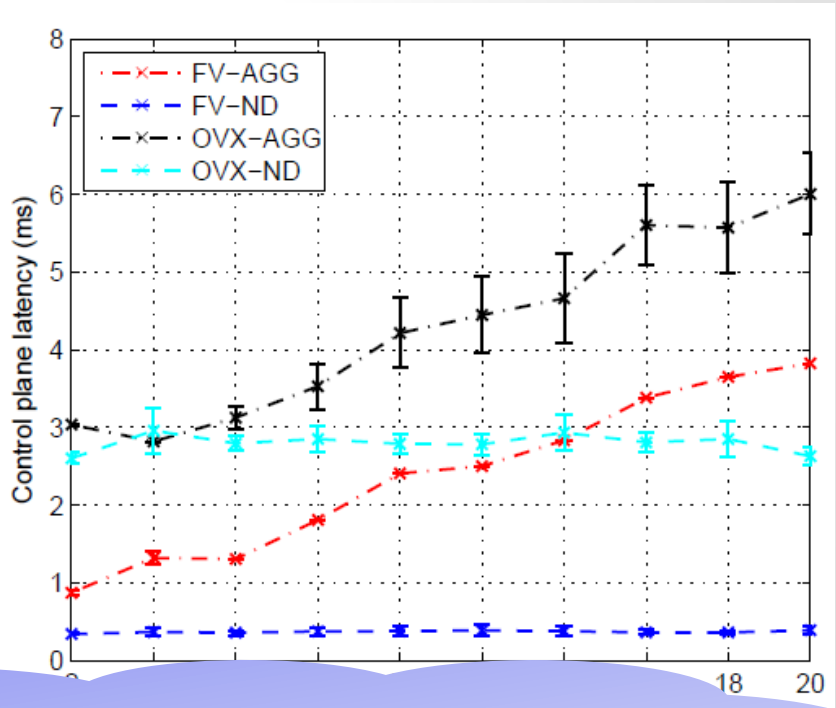
Performance also depends on hypervisor type…

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

What you need to know about your network hypervisor!



… number of tenants…



Performance also depends on hypervisor type…

(multithreaded or not, which ver…

of Nc…

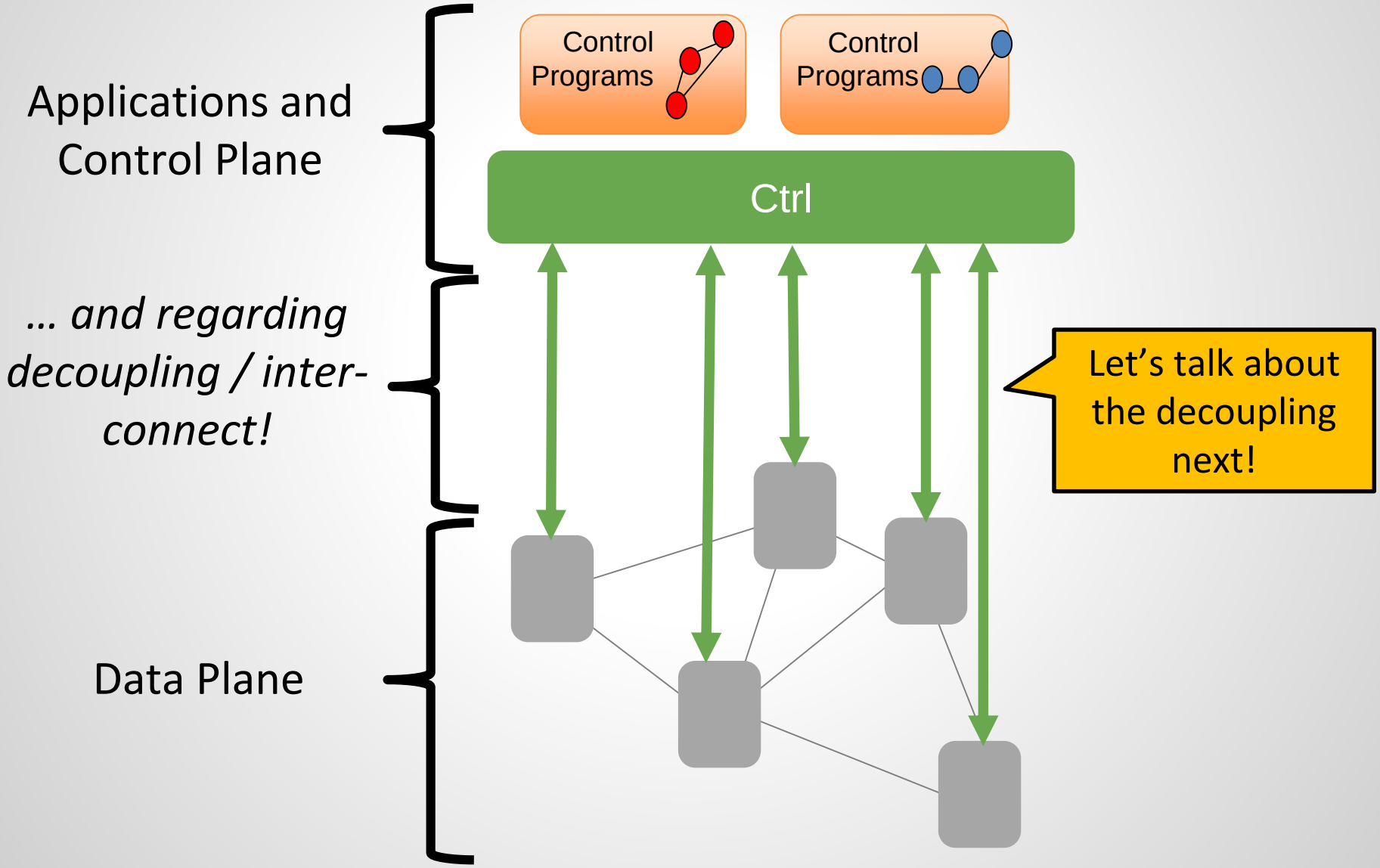
Conclusion: for predictable performance, need to account for all resources!

Further Reading

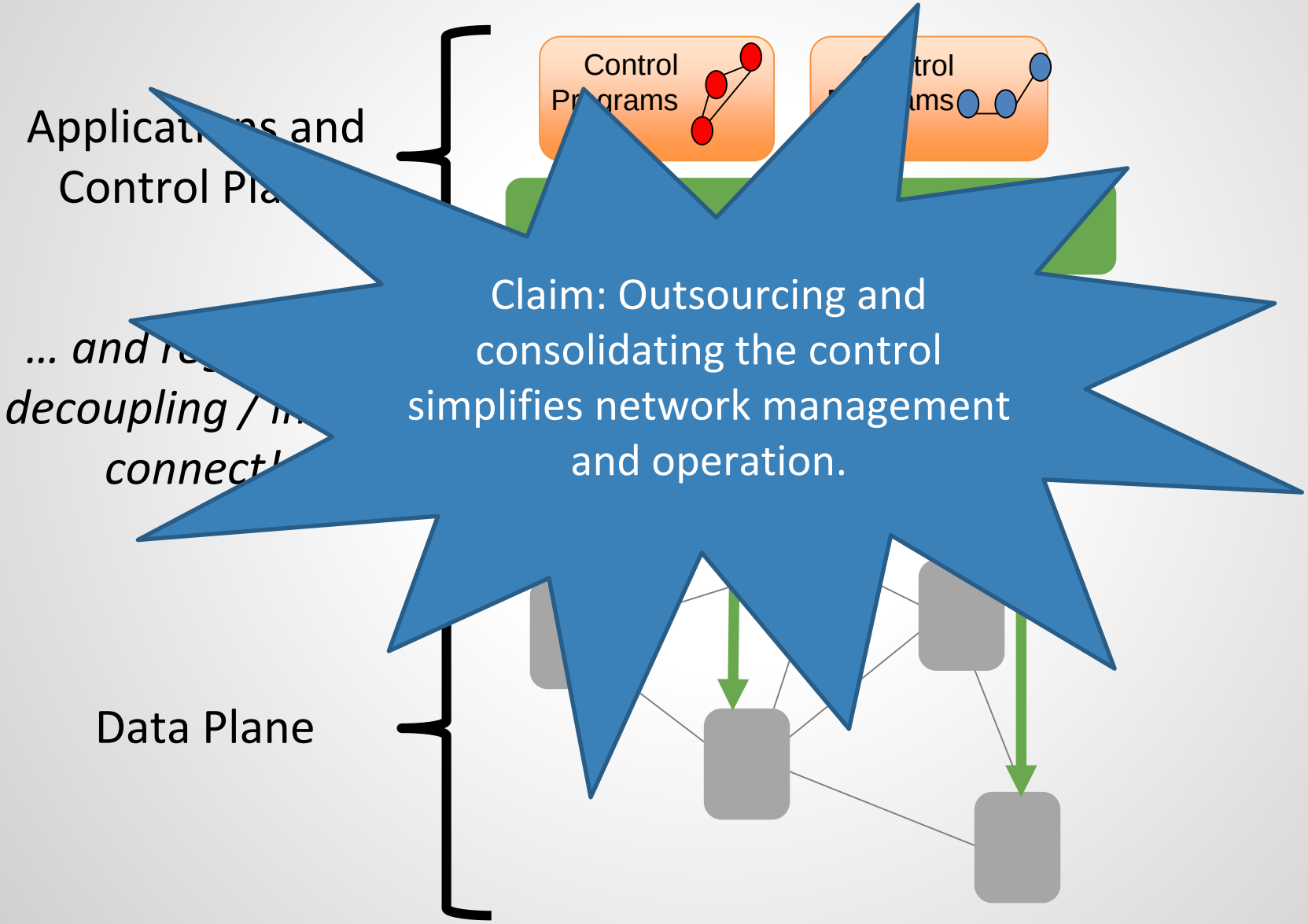
Logically Isolated, Actually Unpredictable? Measuring Hypervisor Performance in Multi-Tenant SDNs

Arsany Basta, Andreas Blenk, Wolfgang Kellerer, and Stefan Schmid. ArXiv Technical Report, May 2017.

Algorithmic Problems in SDNs



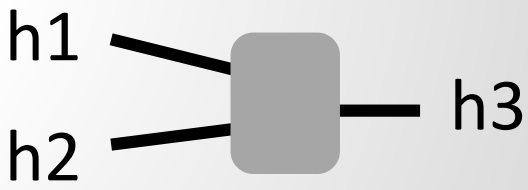
Algorithmic Problems in SDNs



Challenge 1: Controller may miss events

- Fundamental networking task: MAC learning
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets

- Example
  - h1 sends to h2:



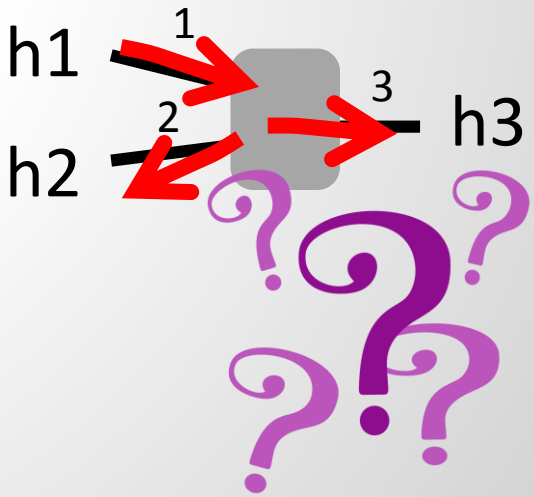
Thanks to Jennifer Rexford for example!

Challenge 1: Controller may miss events

- Fundamental networking task: MAC learning
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets

- Example
  - h1 sends to h2:

flood



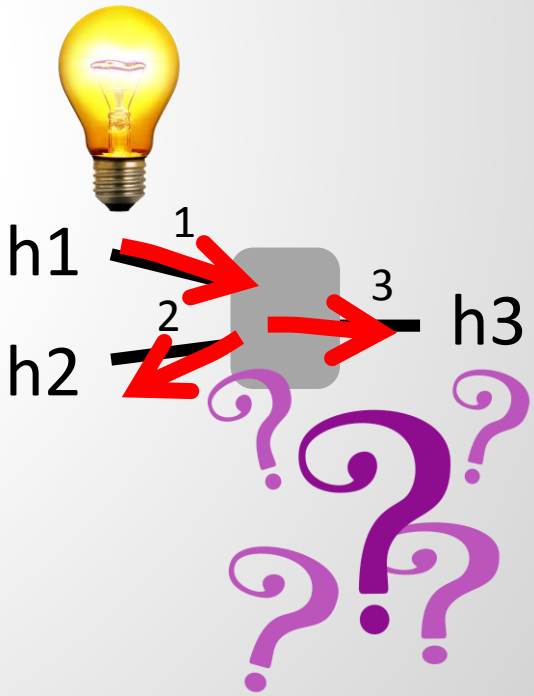
Thanks to Jennifer Rexford for example!

Challenge 1: Controller may miss events

- Fundamental networking task: MAC learning
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets

- Example
  - h1 sends to h2:

    flood, learn (h1,p1)



Thanks to Jennifer Rexford for example!

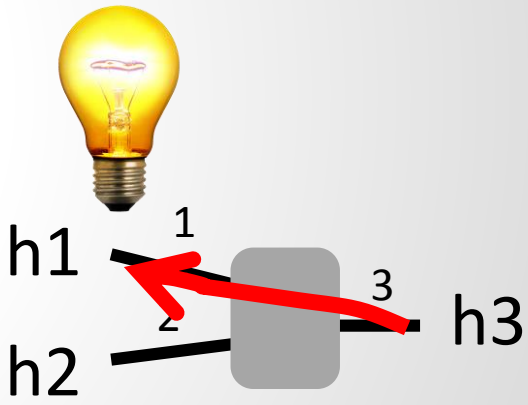
Challenge 1: Controller may miss events

- Fundamental networking task: MAC learning
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets

- Example
  - h1 sends to h2:

    flood, learn (h1,p1)
  - h3 sends to h1:

    forward to p1



Thanks to Jennifer Rexford for example!

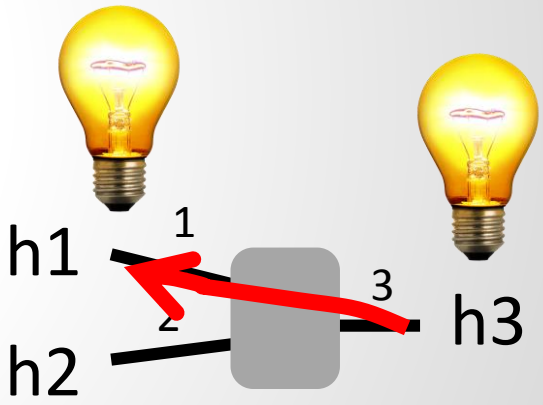
Challenge 1: Controller may miss events

- Fundamental networking task: MAC learning
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets

- Example
  - h1 sends to h2:

    flood, learn (h1,p1)
  - h3 sends to h1:

    forward to p1, learn (h3,p3)



Thanks to Jennifer Rexford for example!

Challenge 1: Controller may miss events

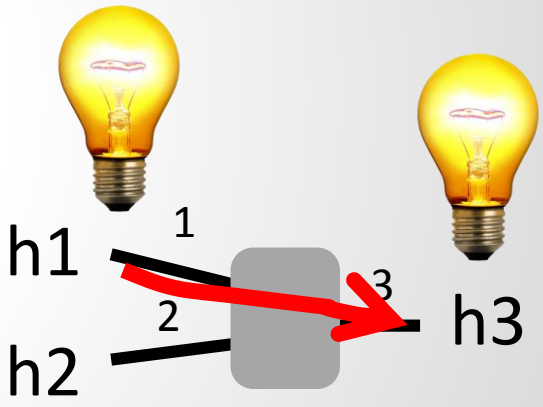
- Fundamental networking task: MAC learning
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets

- 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

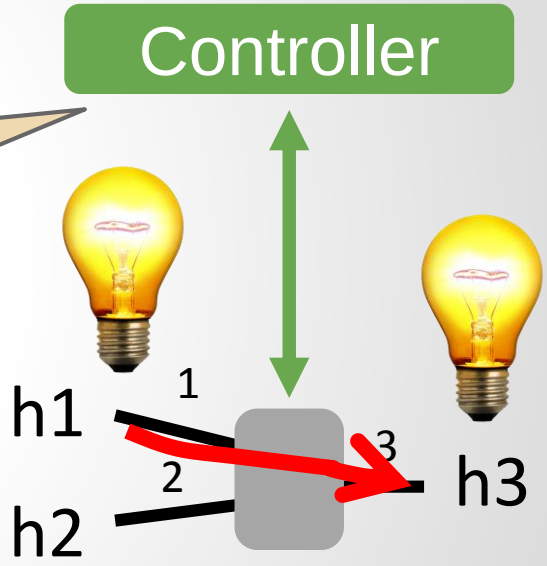


Thanks to Jennifer Rexford for example!

Challenge 1: Controller may miss events

- Fundamental networking task: MAC learning
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets

Now: how to do via controller?



- Example
  - h1 sends to h2:

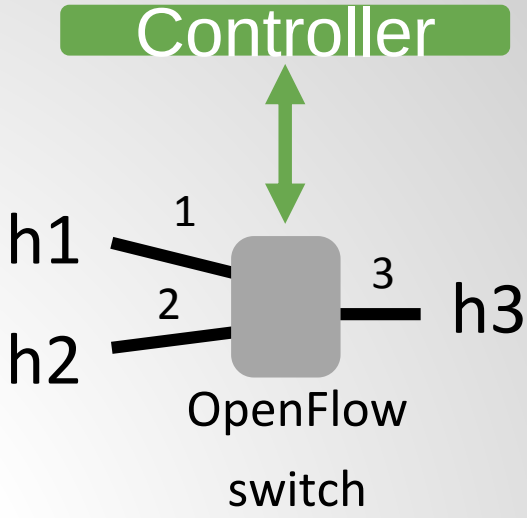
    flood, learn (h1,p1)
  - h3 sends to h1:

    forward to p1, learn (h3,p3)
  - h1 sends to h3:

    forward to p3

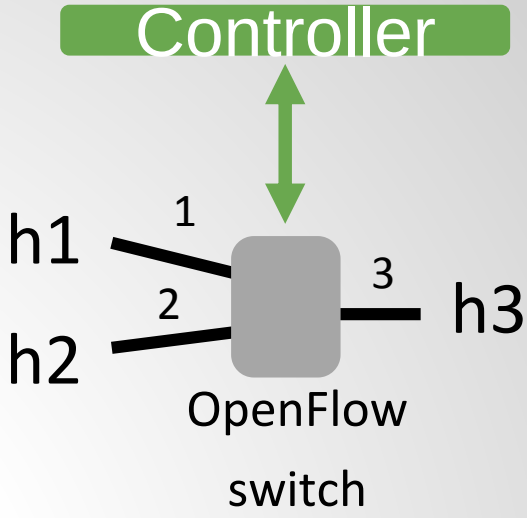
Thanks to Jennifer Rexford for example!

Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

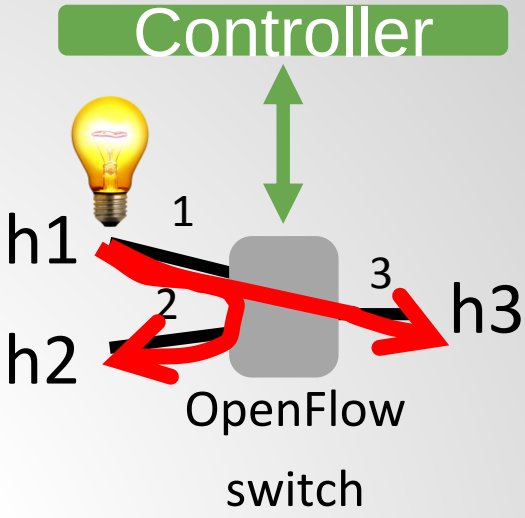
Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

- What happens when h1 sends to h2?

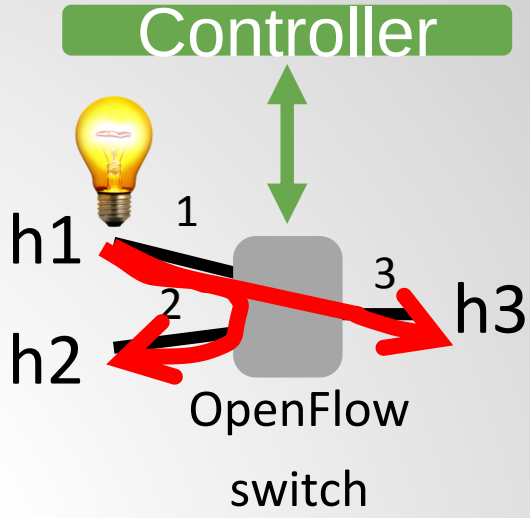
Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

- What happens when h1 sends to h2?
  - Controller learns that h1@p1 and floods

Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

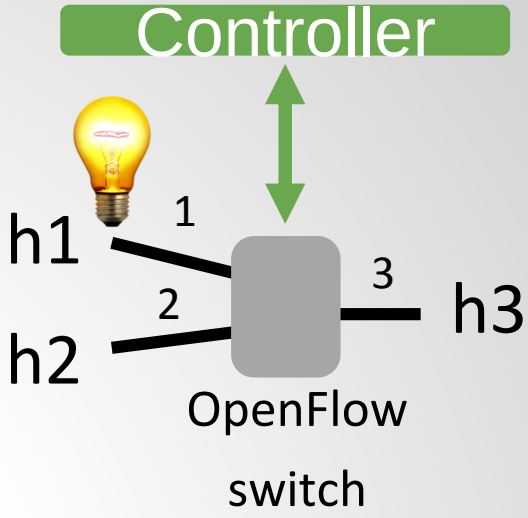
Pattern	Action
*	Send to controller

h1 sends to h2

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

- What happens when h1 sends to h2?
  - Controller learns that h1@p1 and floods

Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

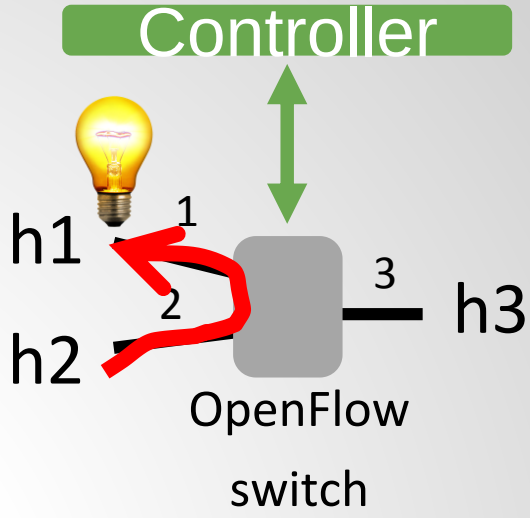
Pattern	Action
*	Send to controller

h1 sends to h2

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

- What happens when h2 sends to h1?

Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

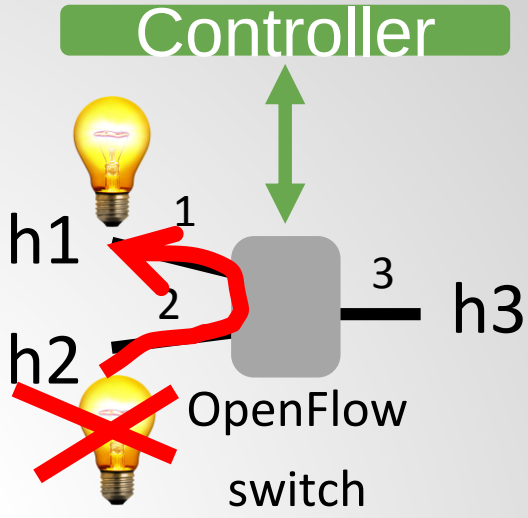
Pattern	Action
*	Send to controller

h1 sends to h2

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

- What happens when h2 sends to h1?
  - Switch knows destination: message forwarded to h1

Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

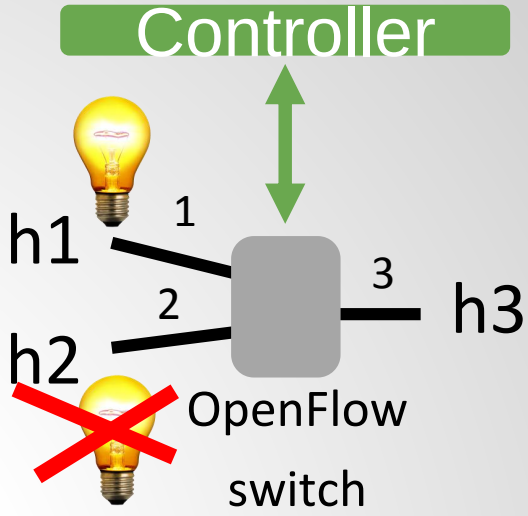
Pattern	Action
*	Send to controller

h1 sends to h2

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

- What happens when h2 sends to h1?
  - Switch knows destination: message forwarded to h1
  - BUT: No controller interaction, no new rule for h2

Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

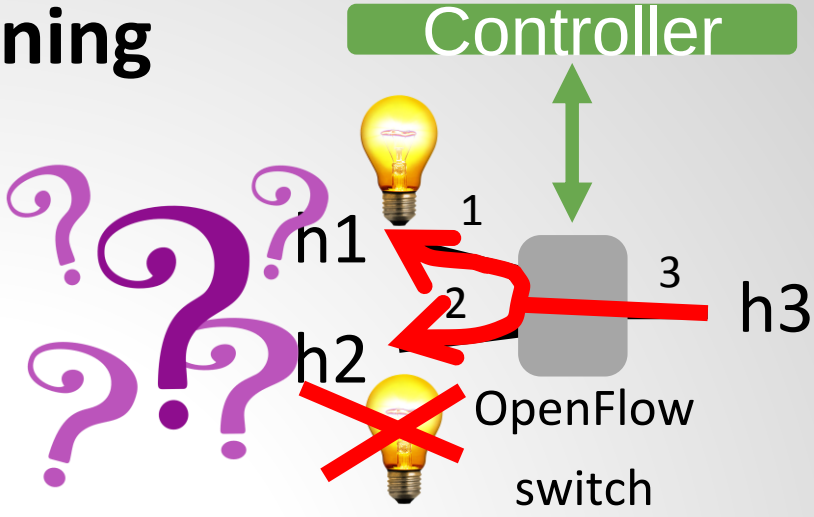
Pattern	Action
*	Send to controller

h1 sends to h2

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

- What happens when h2 sends to h1?
  - Switch knows destination: message forwarded to h1
  - BUT: No controller interaction, no new rule for h2
- What happens when h3 sends to h2?

Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

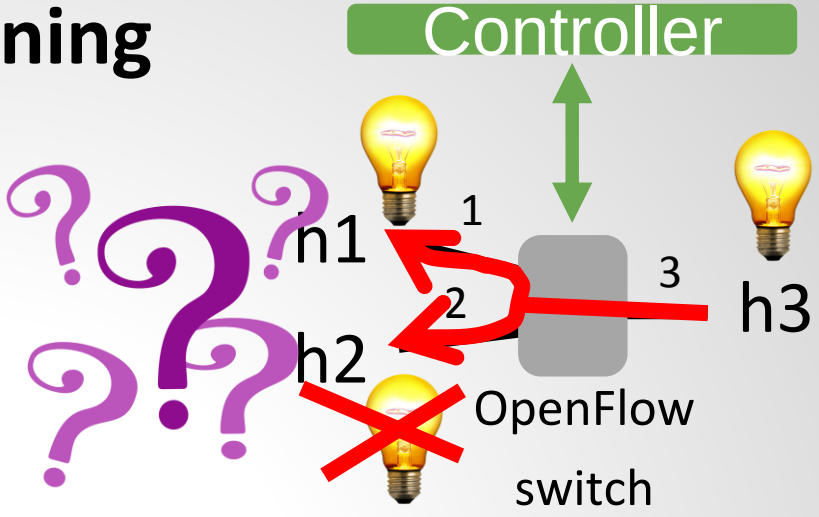
Pattern	Action
*	Send to controller

h1 sends to h2

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

- What happens when h2 sends to h1?
  - Switch knows destination: message forwarded to h1
  - BUT: No controller interaction, no new rule for h2
- What happens when h3 sends to h2?
  - Flooded! Controller did not put the rule to h2.

Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

Pattern	Action
*	Send to controller

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

Controller however does learn about h3.
BUT NOW: Future communications from h2 to h3 missed by controller too: no opportunity to learn about h2, so future requests to h2 flooded as well?!?

h1?

forwarded to h1

new rule for h2

- What happens when h3 sends to h2?
  - Flooded! Controller did not put the rule to h2!

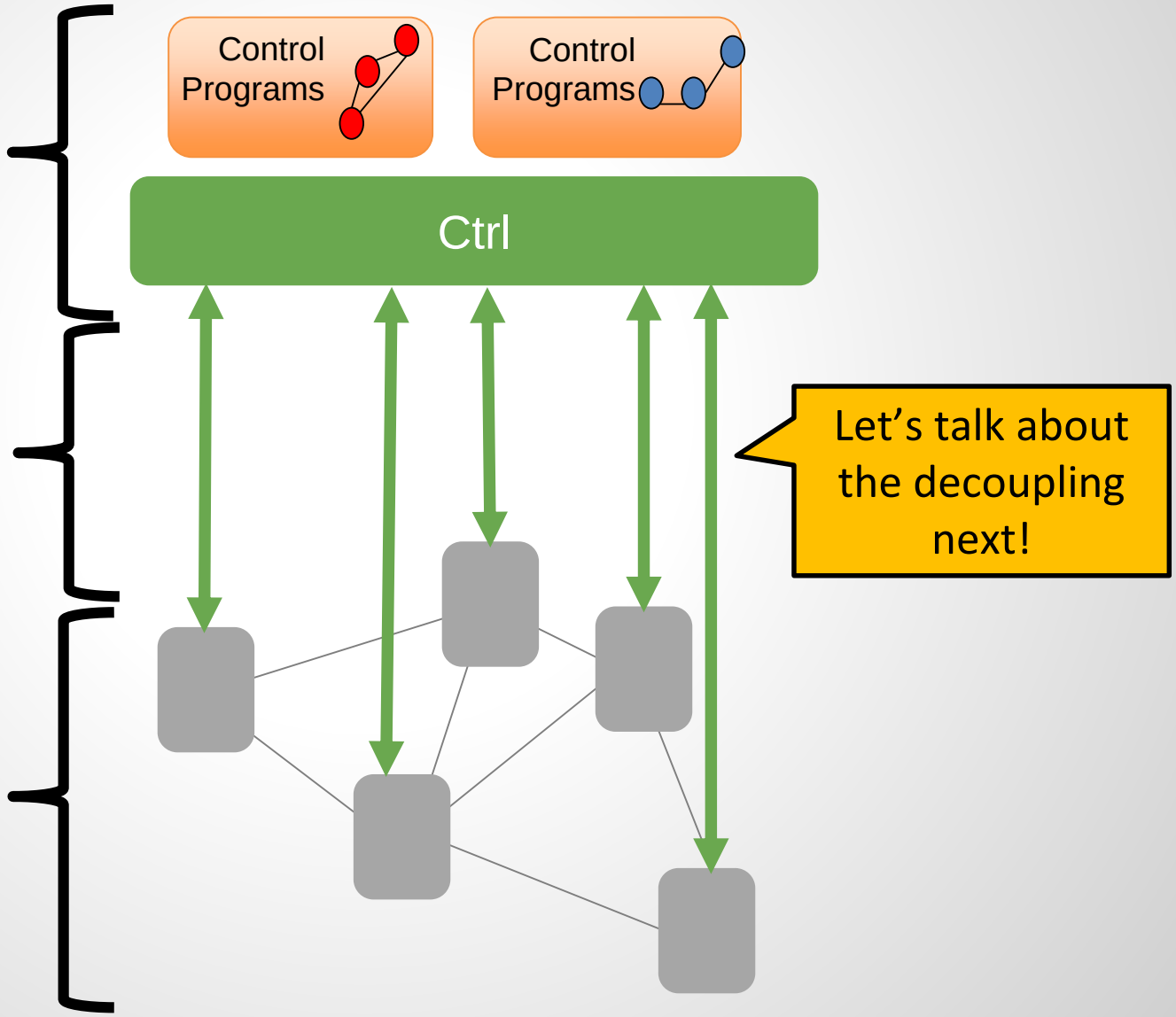
Algorithmic Problems in SDNs

Repetition

Applications and Control Plane

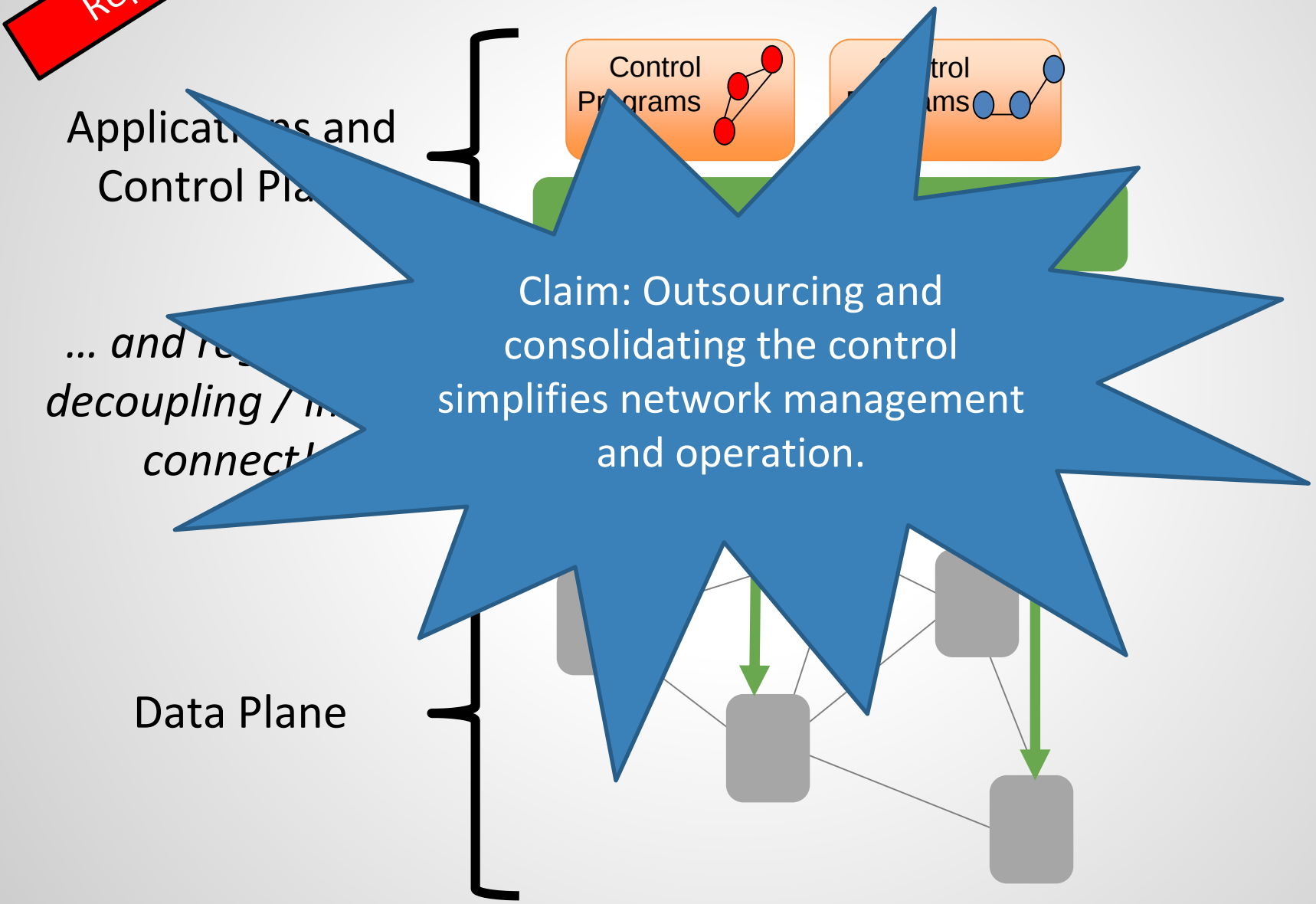
… and regarding decoupling / inter-connect!

Data Plane



Algorithmic Problems in SDNs

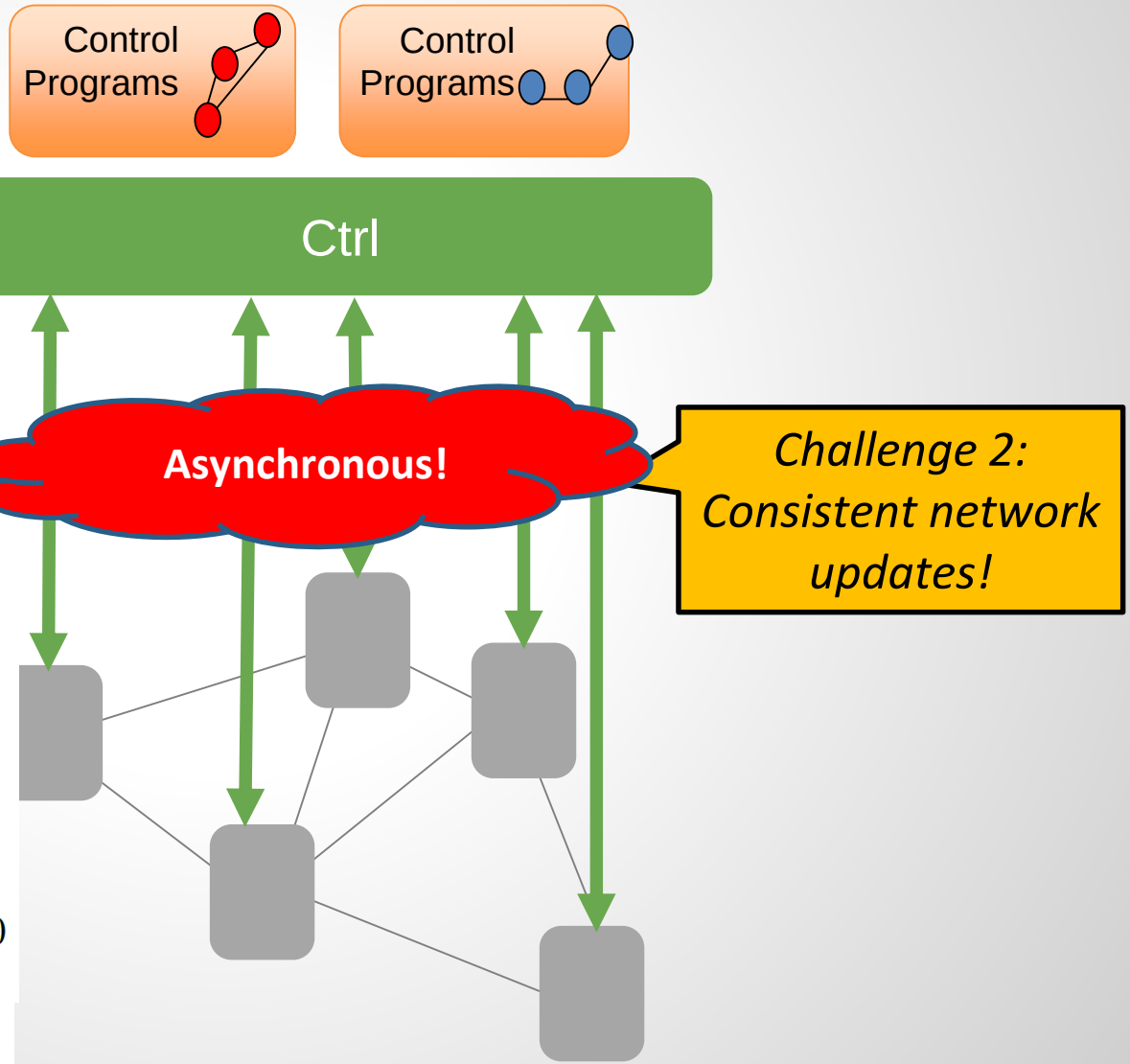
Repetition

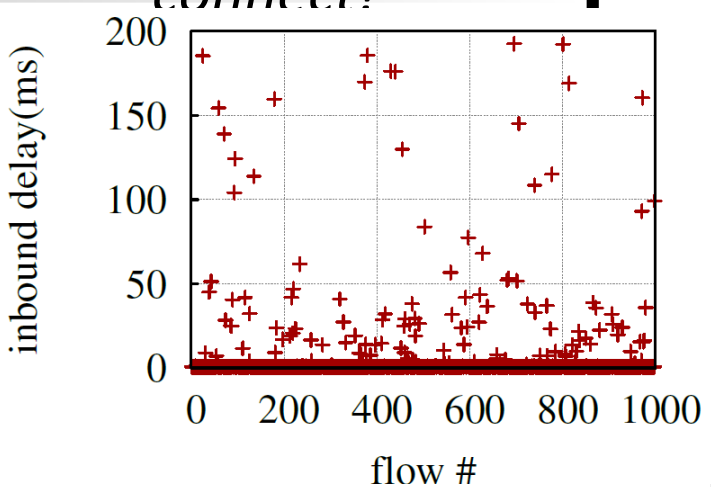


Algorithmic Problems in SDNs

Applications and Control Plane

… and regarding decoupling / inter-connect!



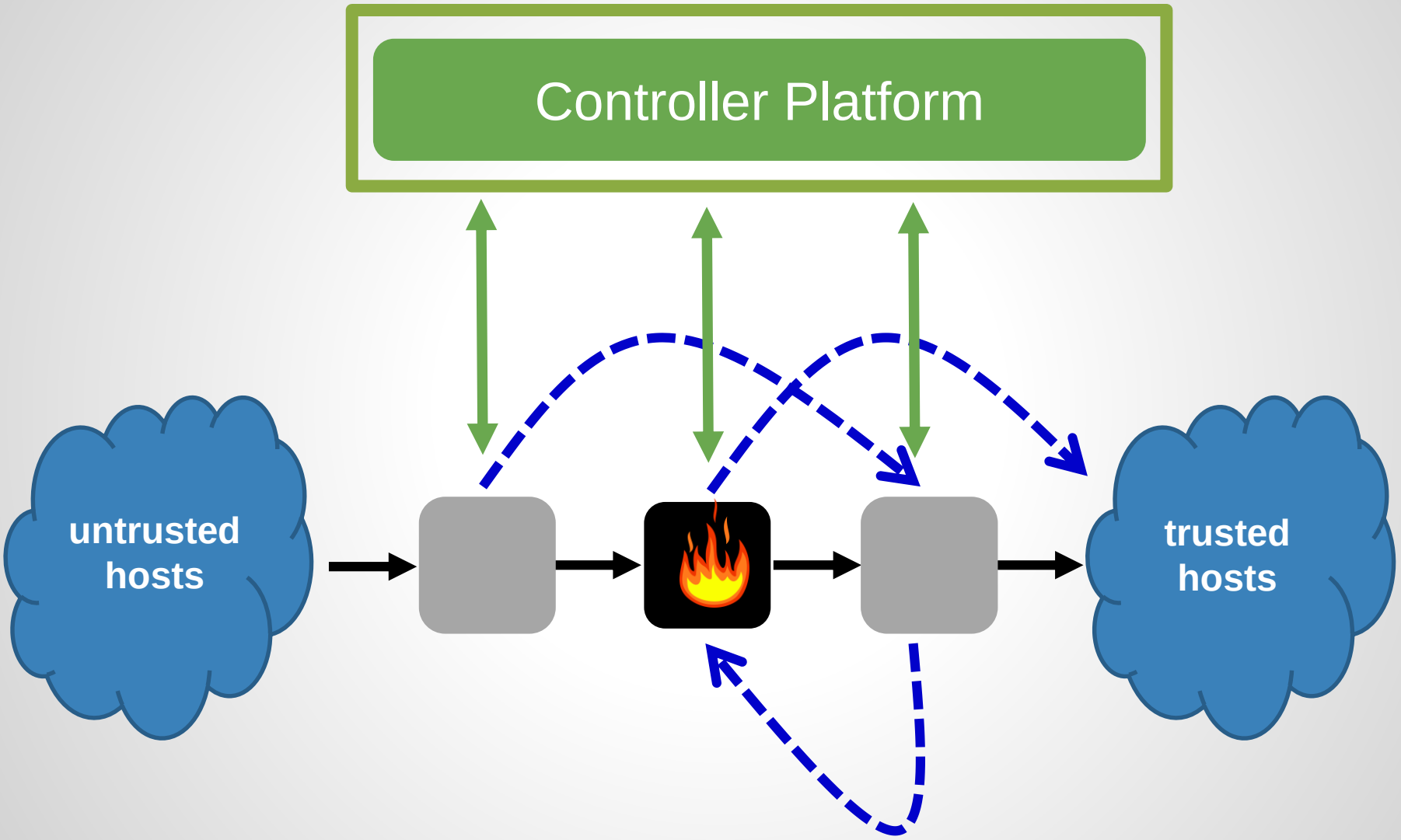


He et al., ACM SOSR 2015:

without network latency

Challenge 2: Route Updates

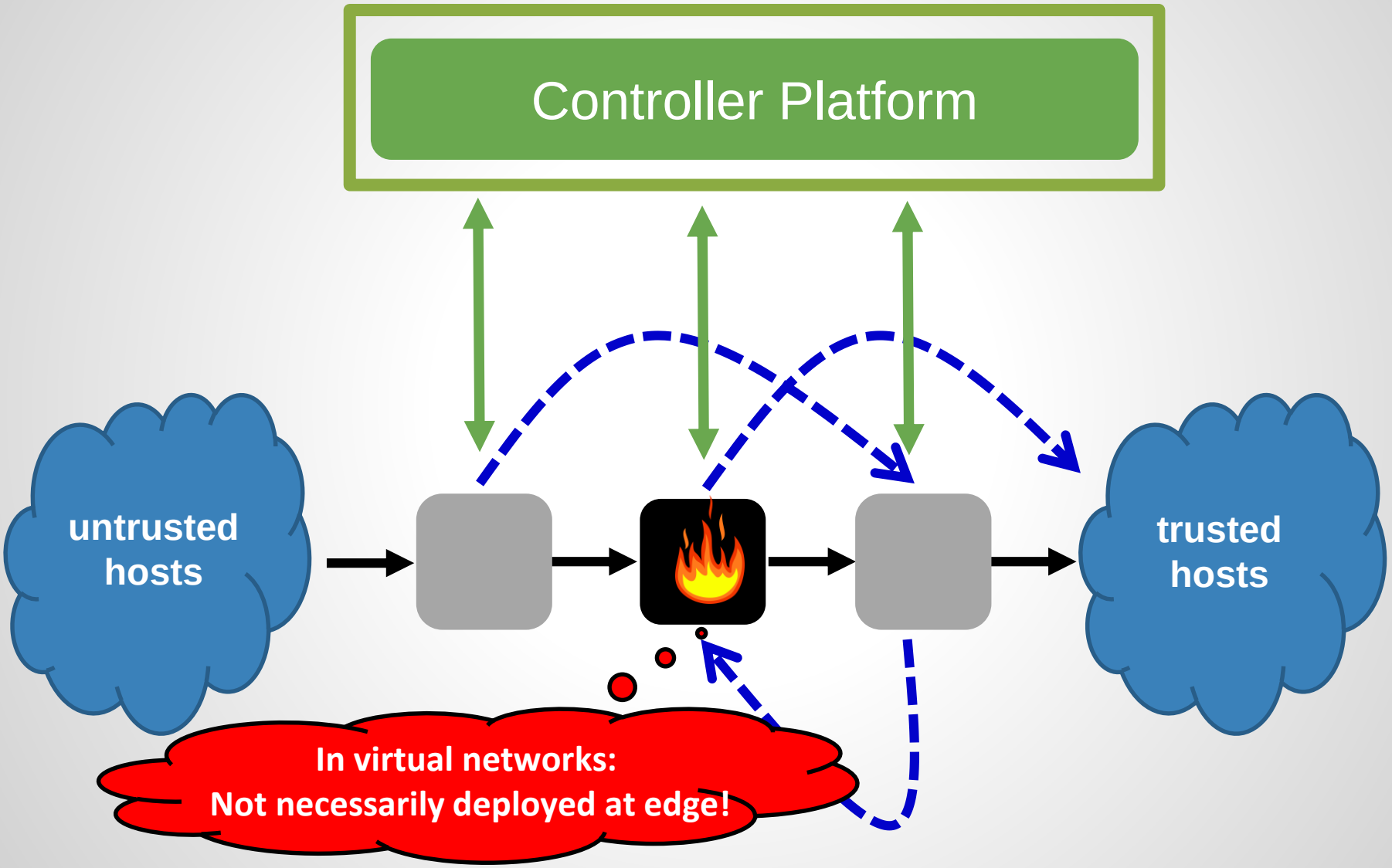
What can possibly go wrong?



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

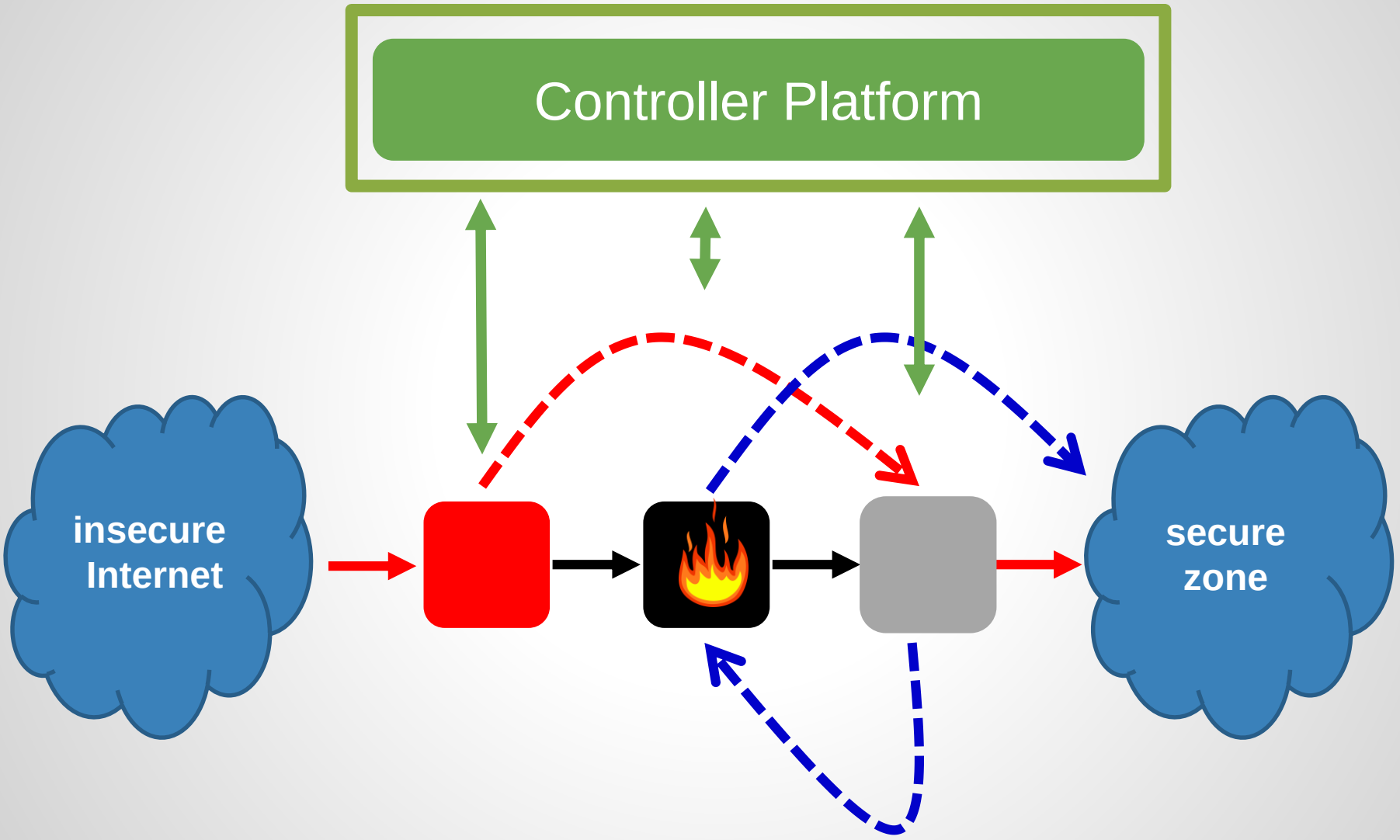
Challenge 2: Route Updates

What can possibly go wrong?



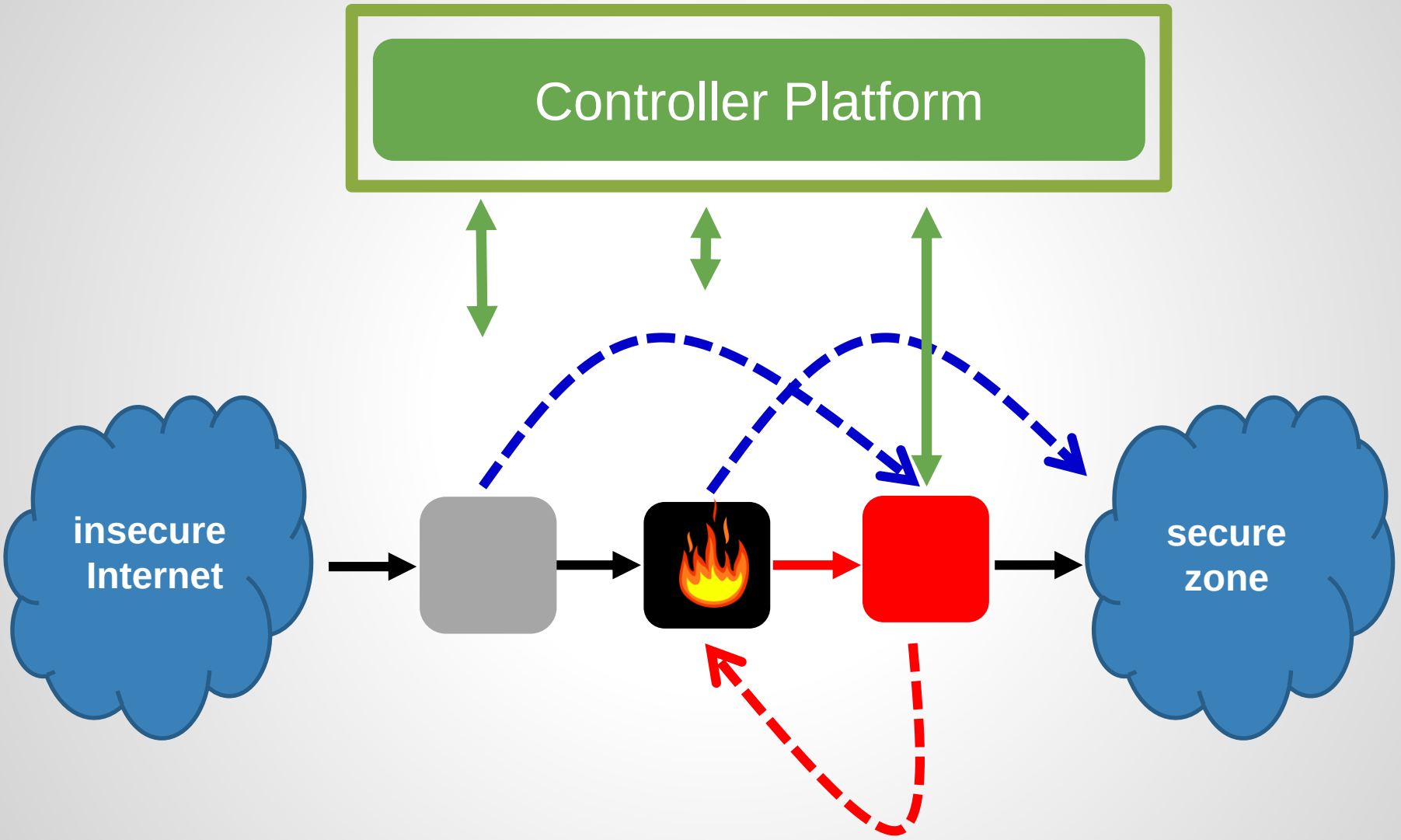
Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

Problem 1: Bypassed Waypoint



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

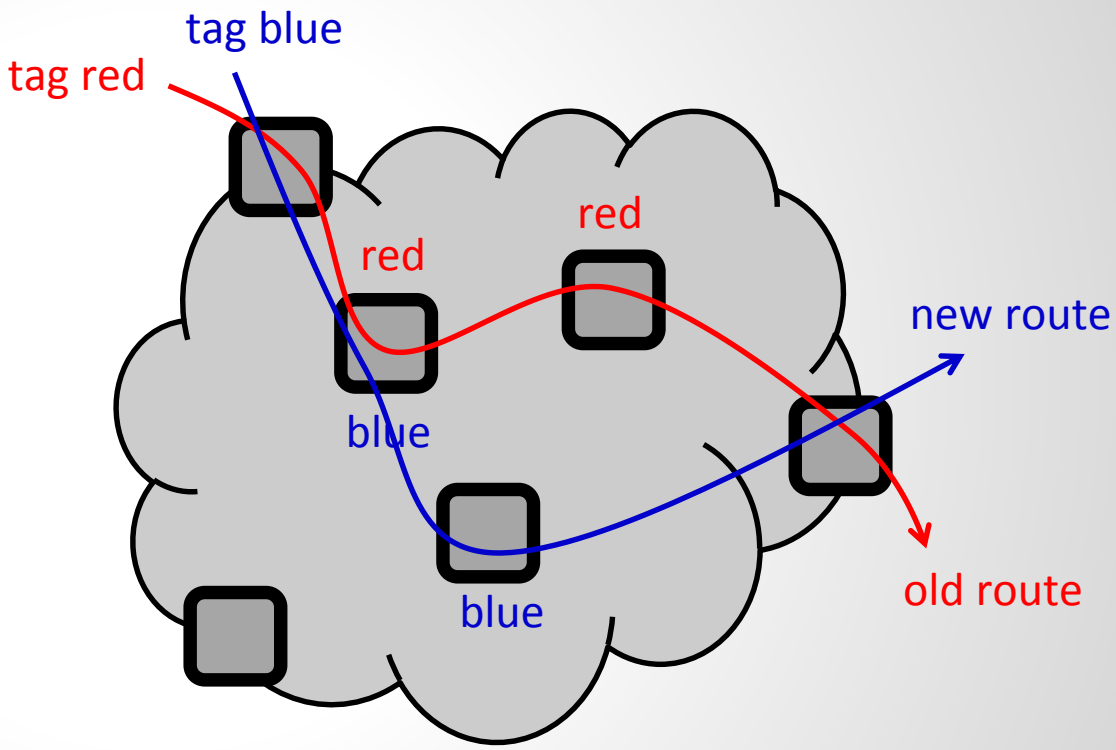
Problem 2: *Transient* Loop



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

Tagging: A Universal Solution?

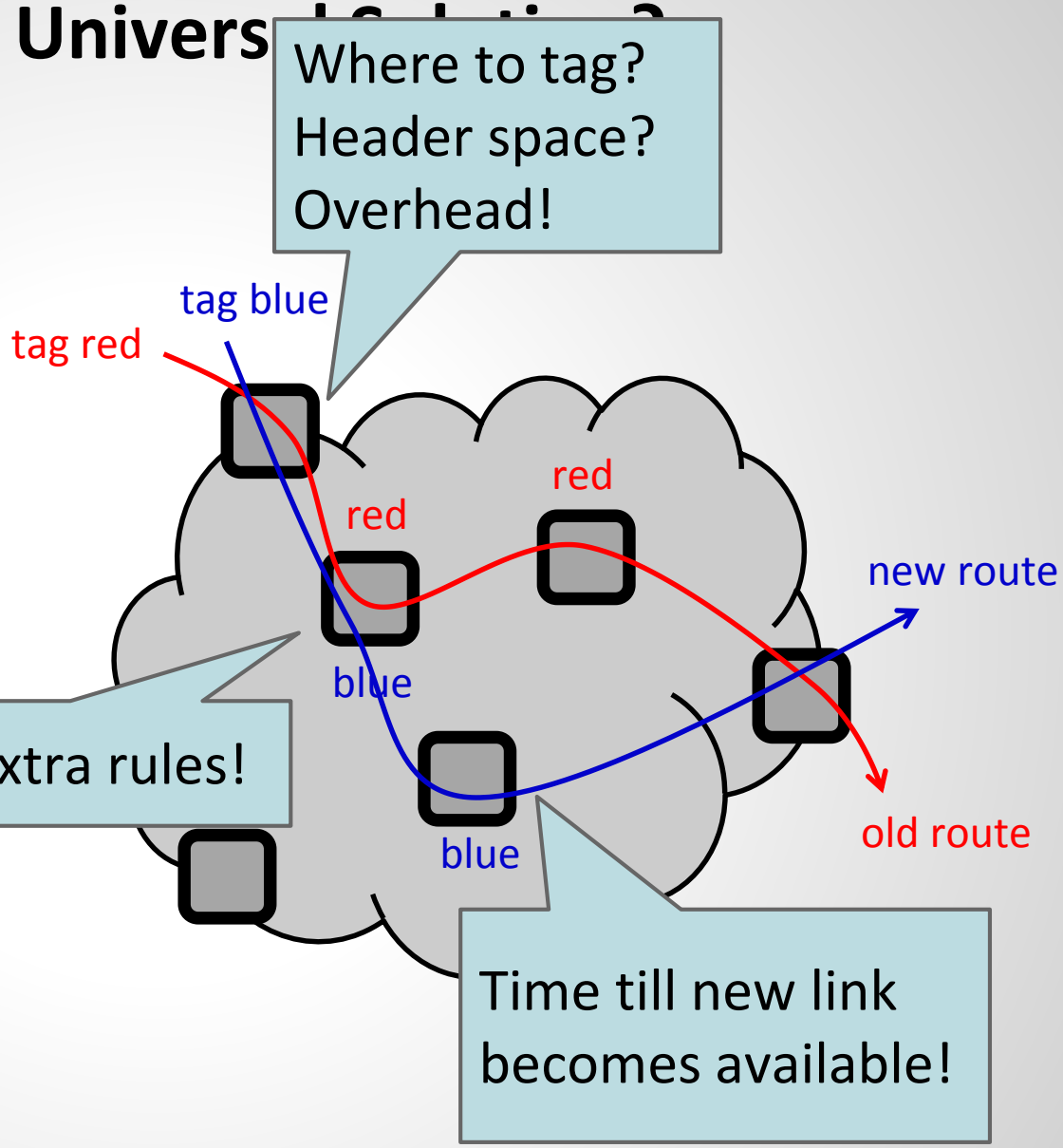
- Old route: red
- New route: blue
- 2-Phase Update:
  - Install blue flow rules internally
  - Flip tag at ingress ports



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

Tagging: A Universal Solution?

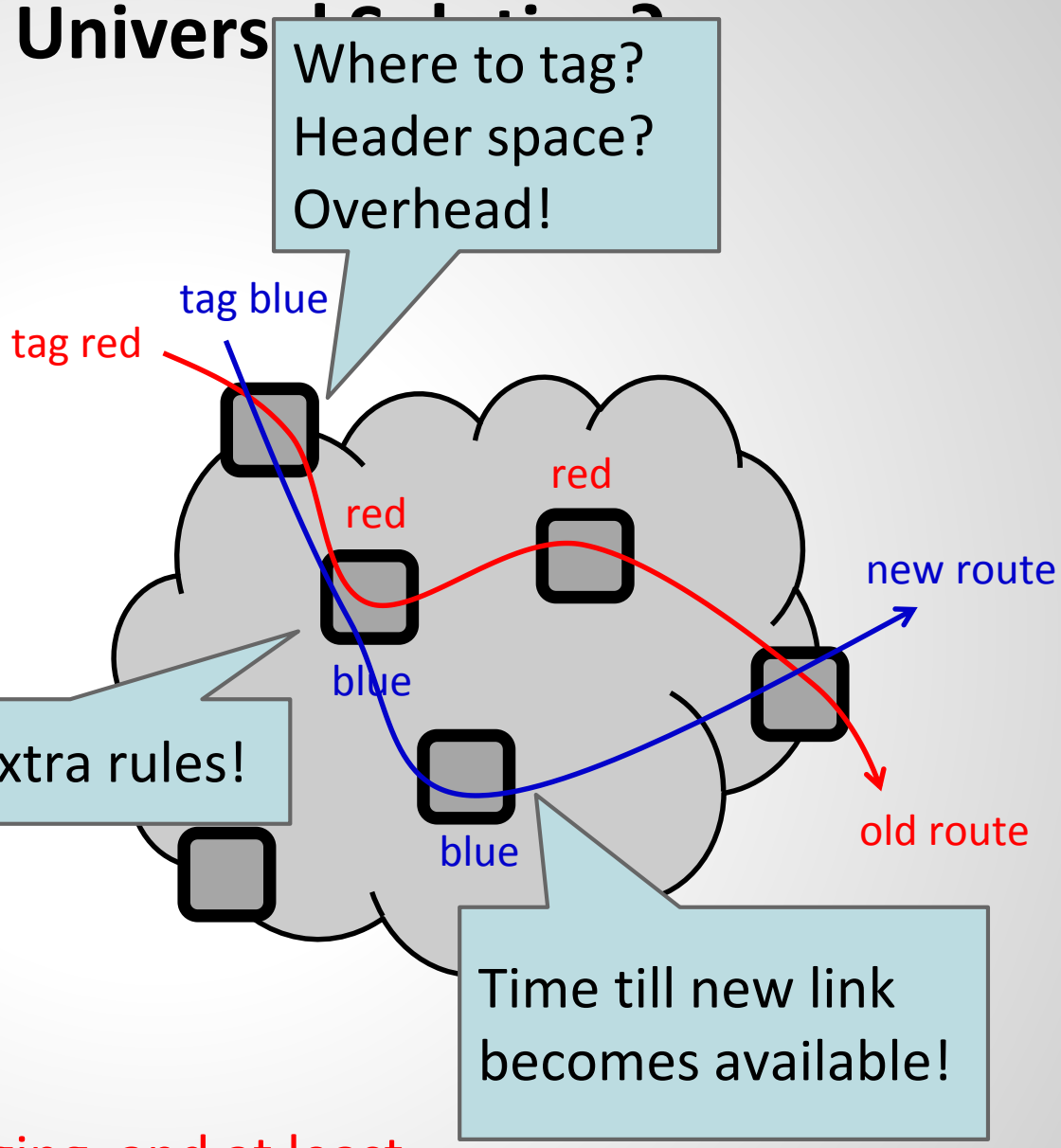
- Old route: red
- New route: blue
- 2-Phase Update:
  - Install blue rules internally
  - Flip tag at ingress ports



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

Tagging: A Universal Solution?

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



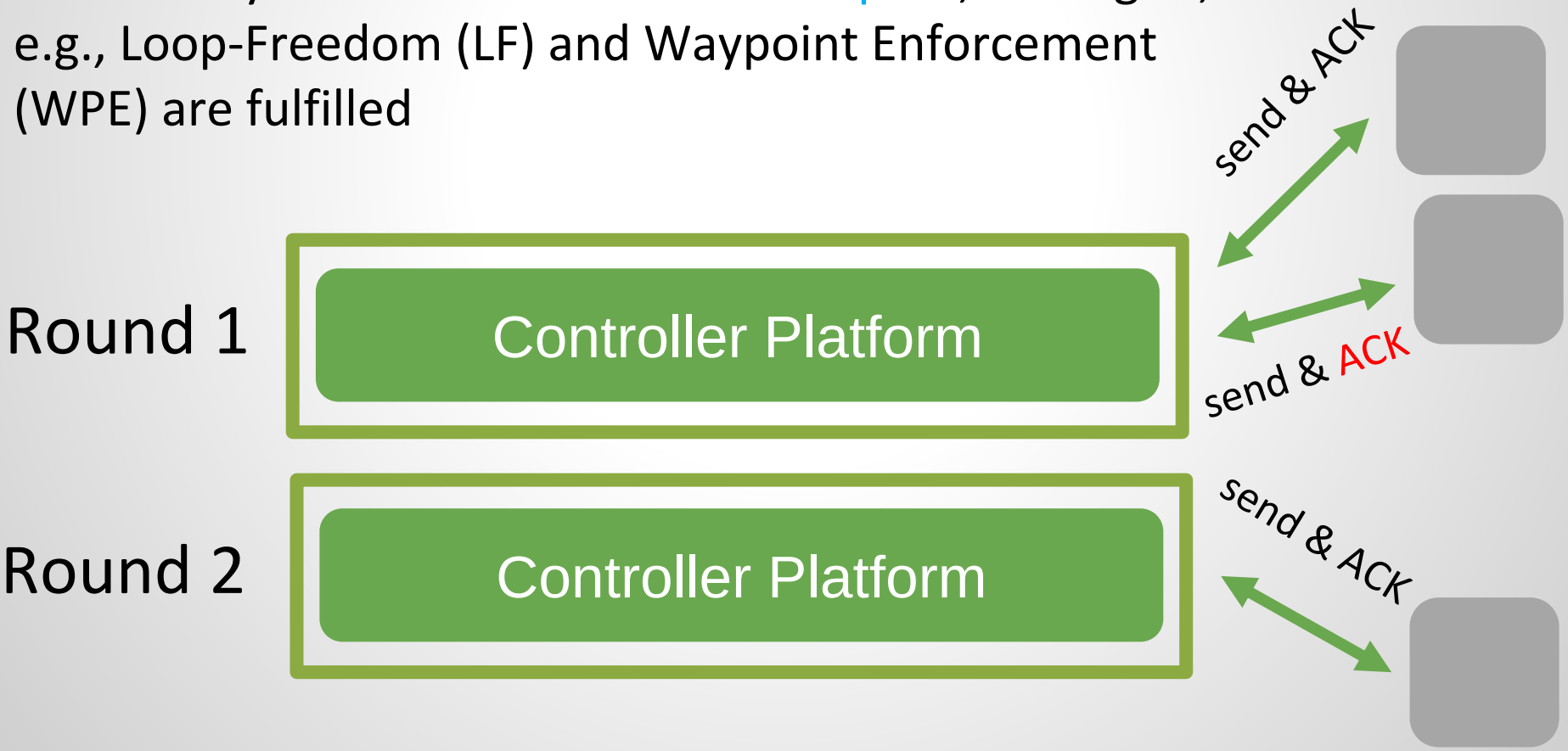
Possible solution without tagging, and at least preserve weaker consistency properties?

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

Idea: Schedule Subsets of Nodes!

Idea: Schedule safe update subsets in **multiple rounds!**

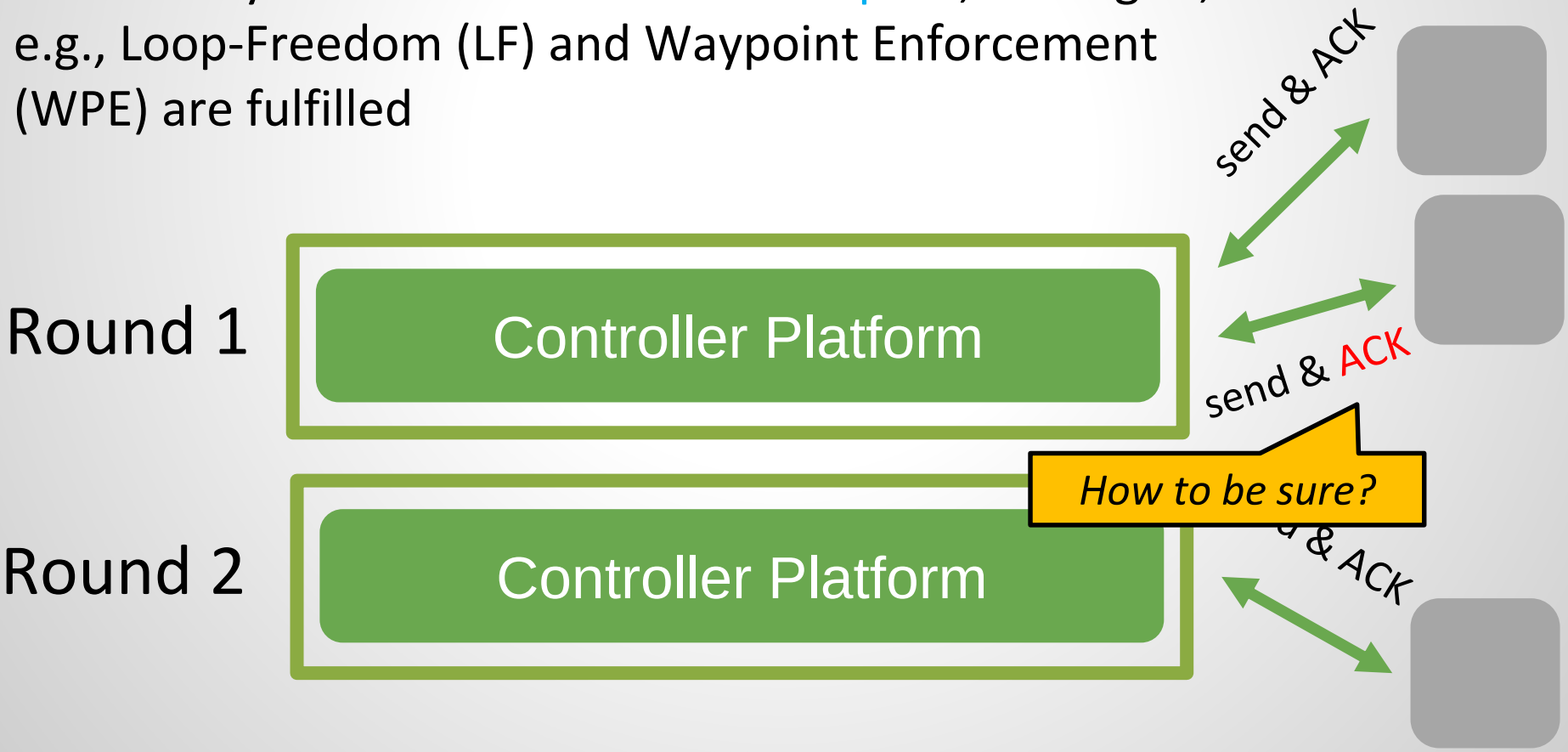
Packet may take a mix of old and new path, as long as, e.g., Loop-Freedom (LF) and Waypoint Enforcement (WPE) are fulfilled



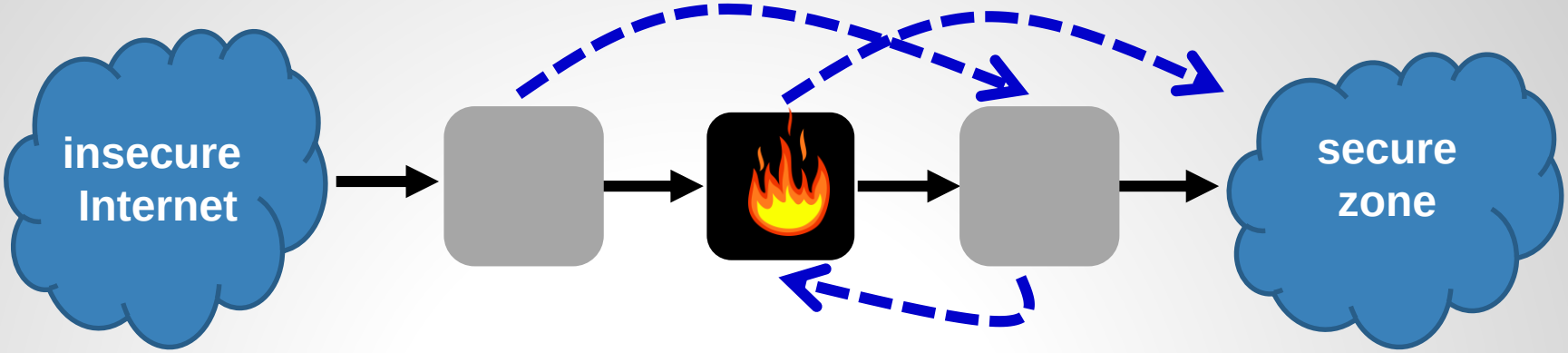
Idea: Schedule Subsets of Nodes!

Idea: Schedule safe update subsets in **multiple rounds!**

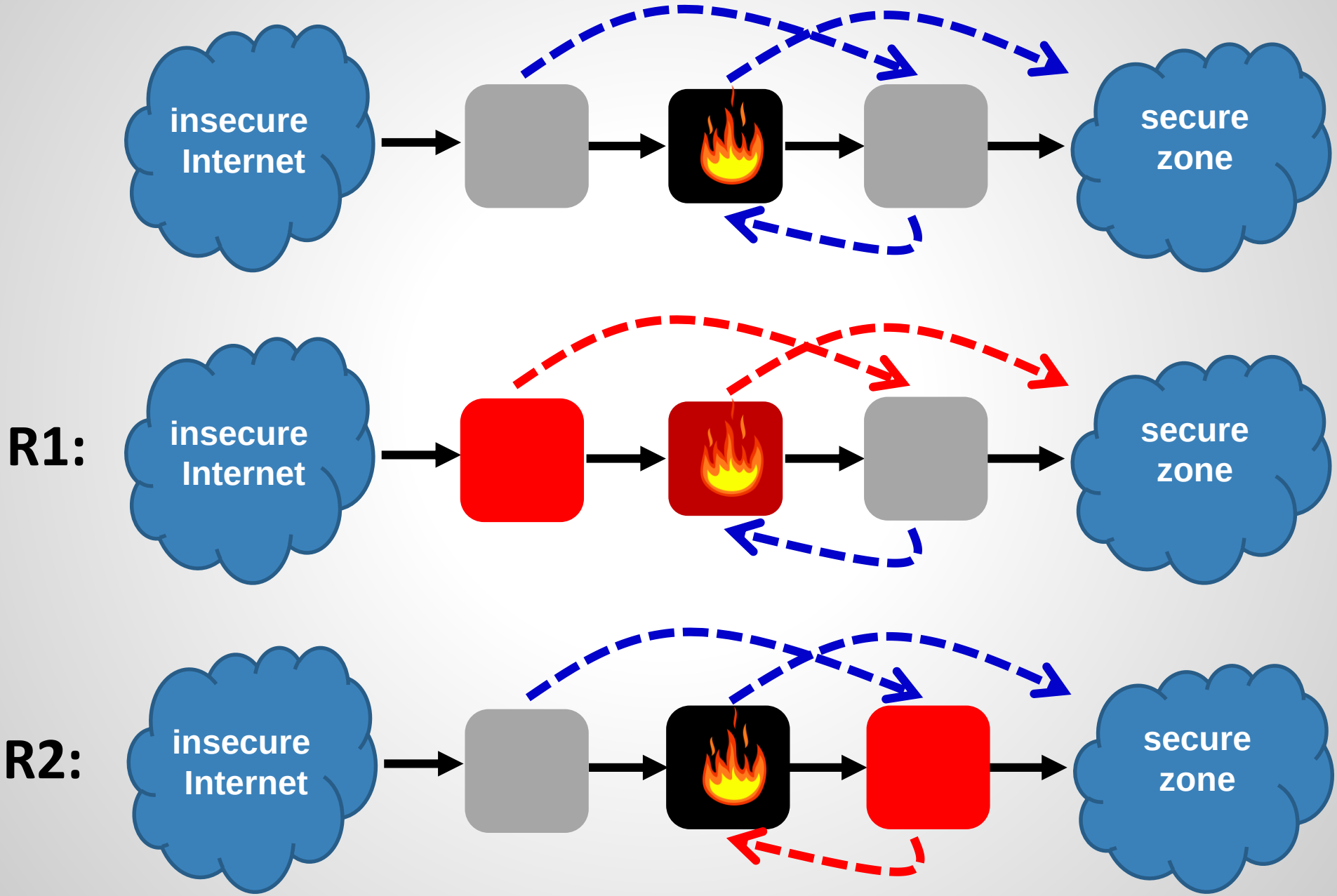
Packet may take a mix of old and new path, as long as, e.g., Loop-Freedom (LF) and Waypoint Enforcement (WPE) are fulfilled



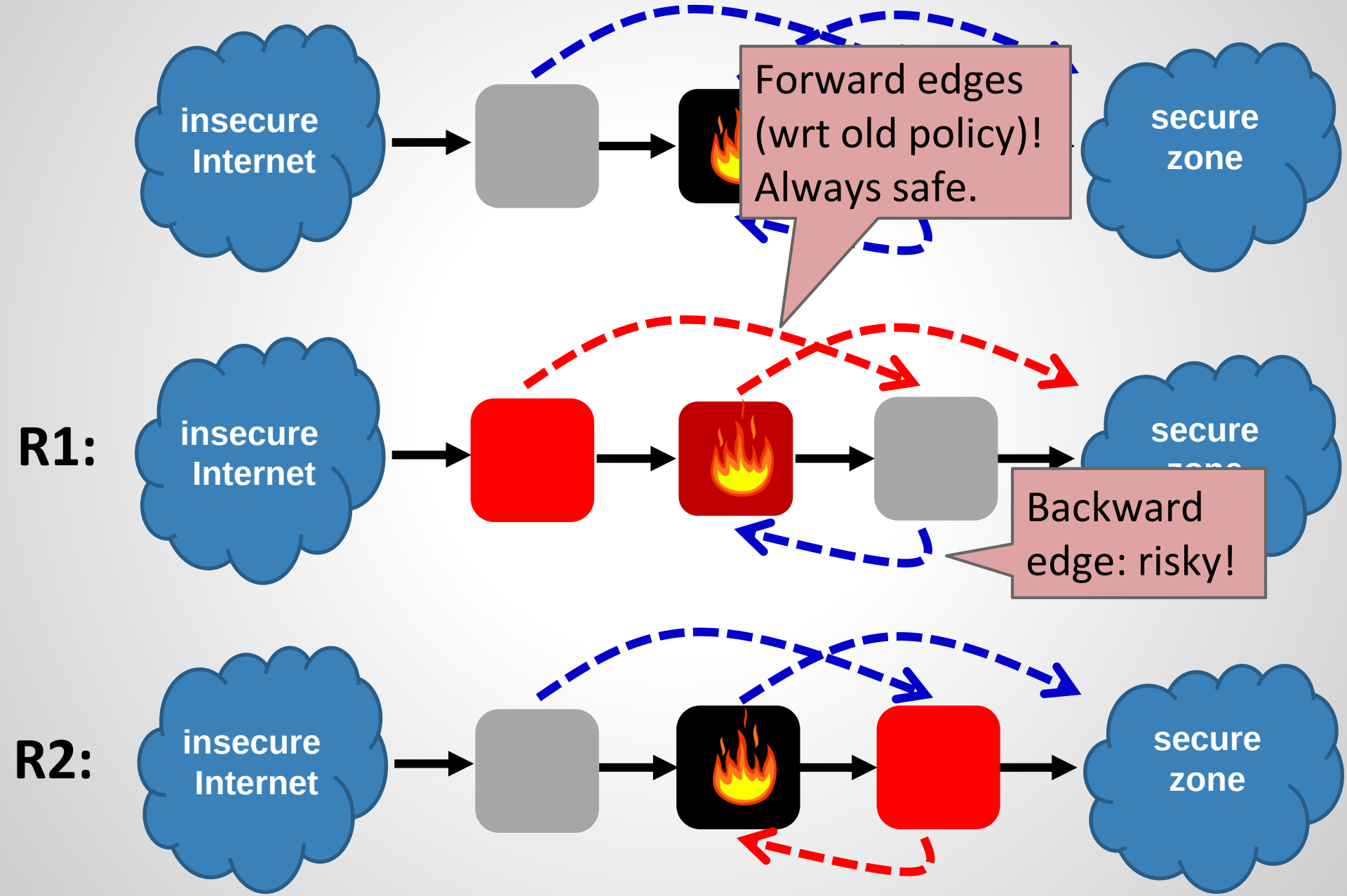
Going Back to Our Examples: LF Update



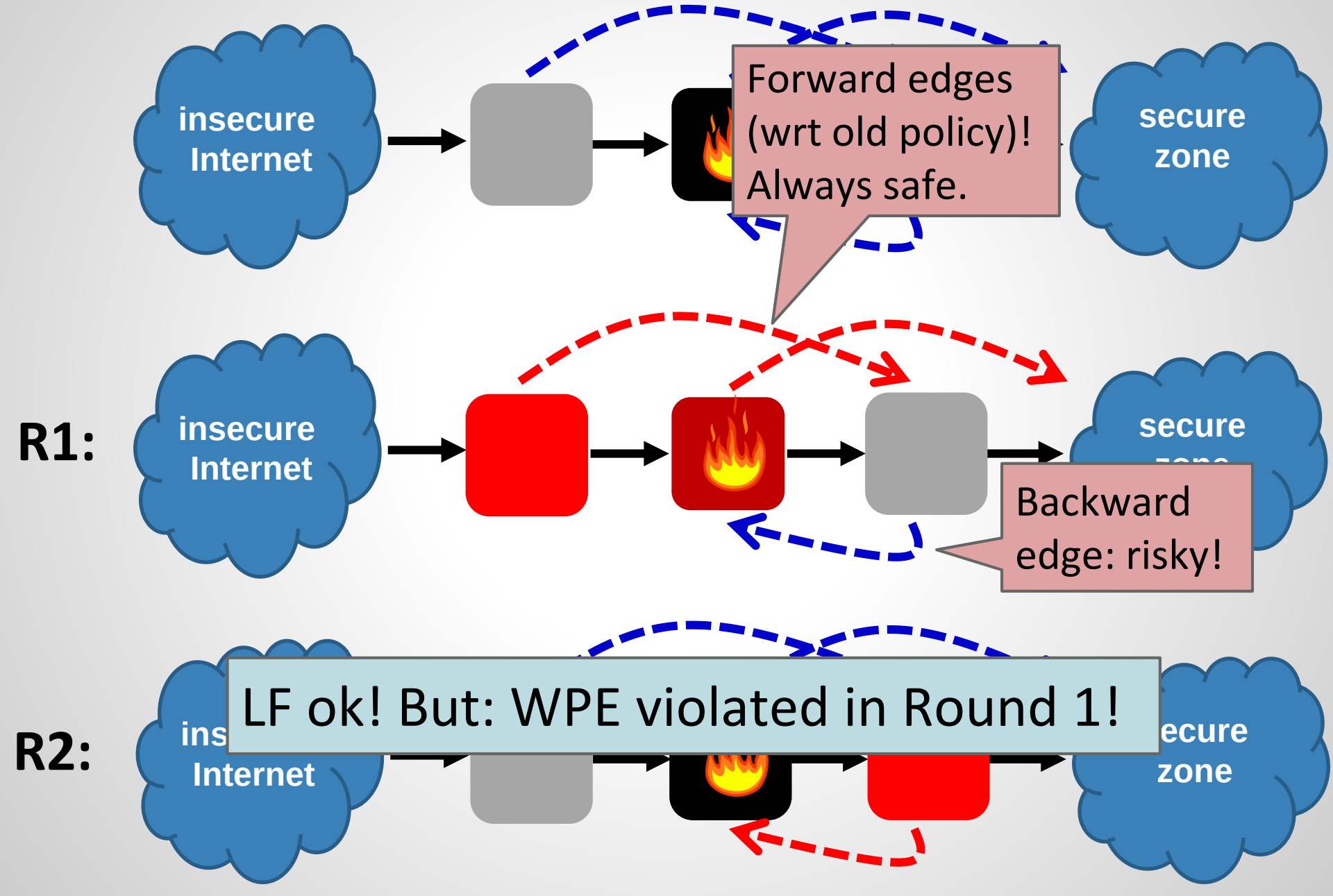
Going Back to Our Examples: LF Update



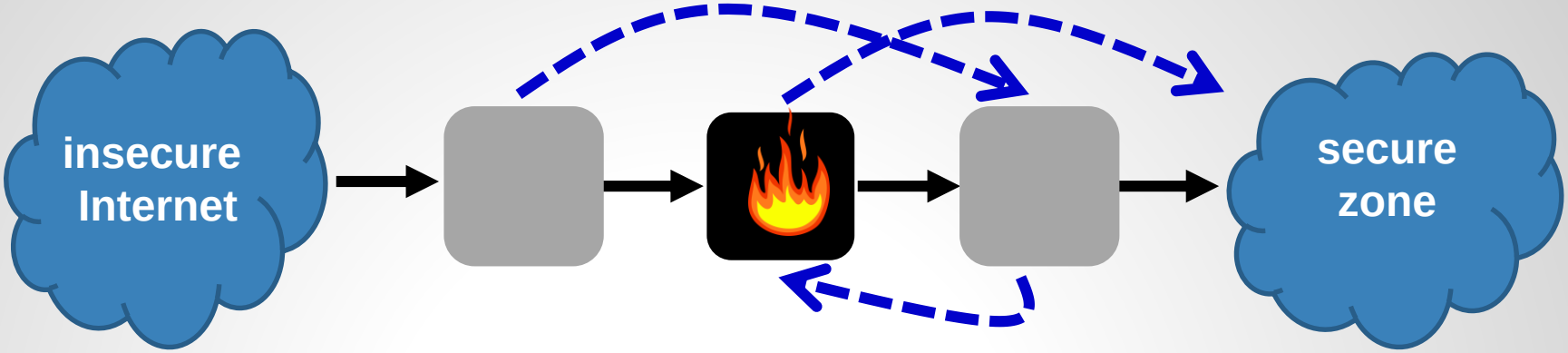
Going Back to Our Examples: LF Update



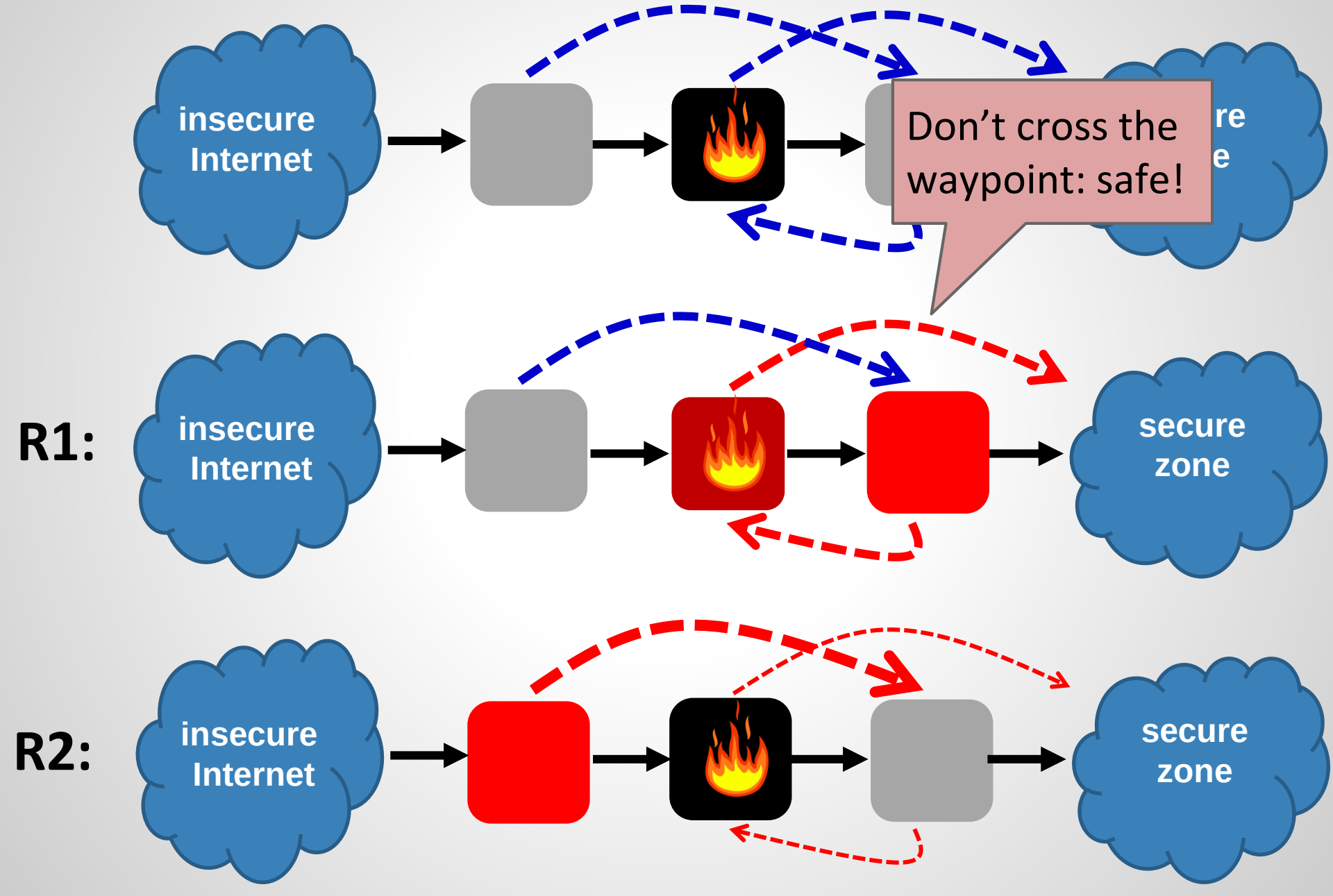
Going Back to Our Examples: LF Update



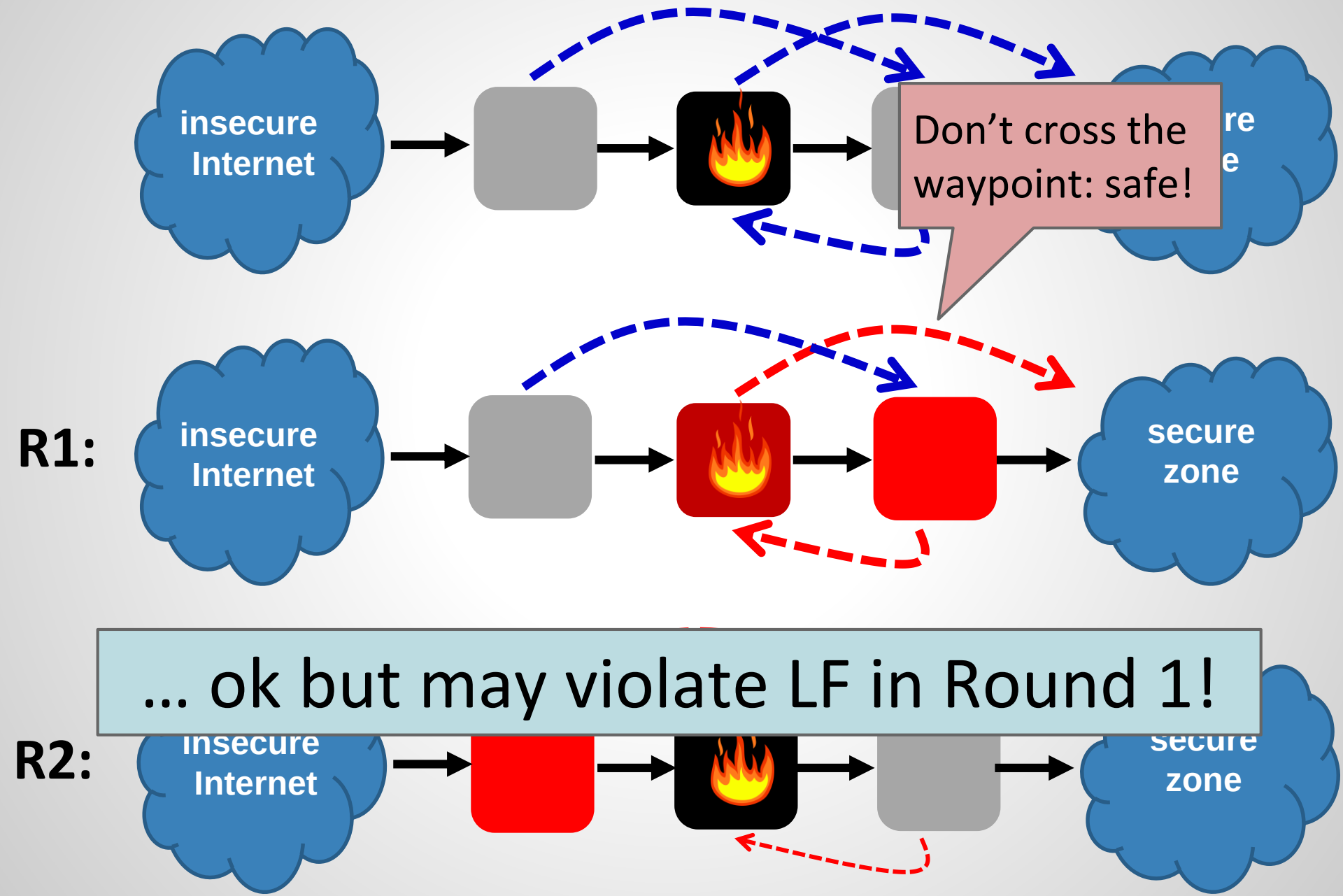
Going Back to Our Examples: WPE Update



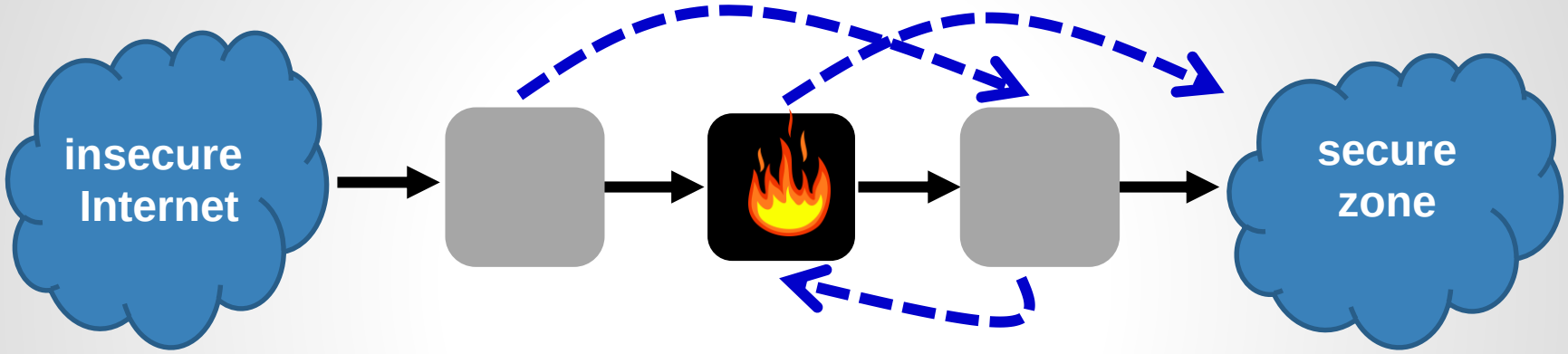
Going Back to Our Examples: WPE Update



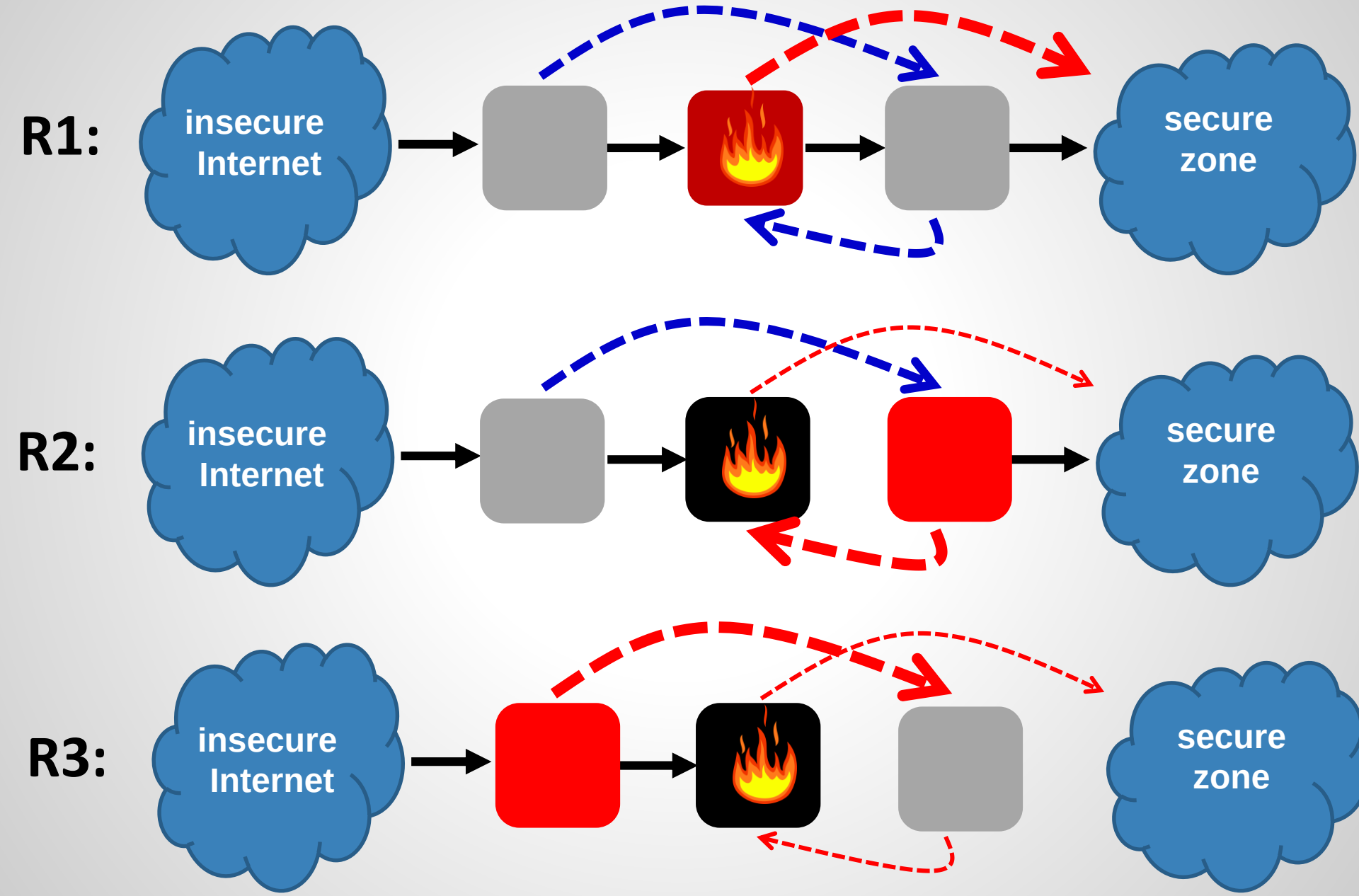
Going Back to Our Examples: WPE Update



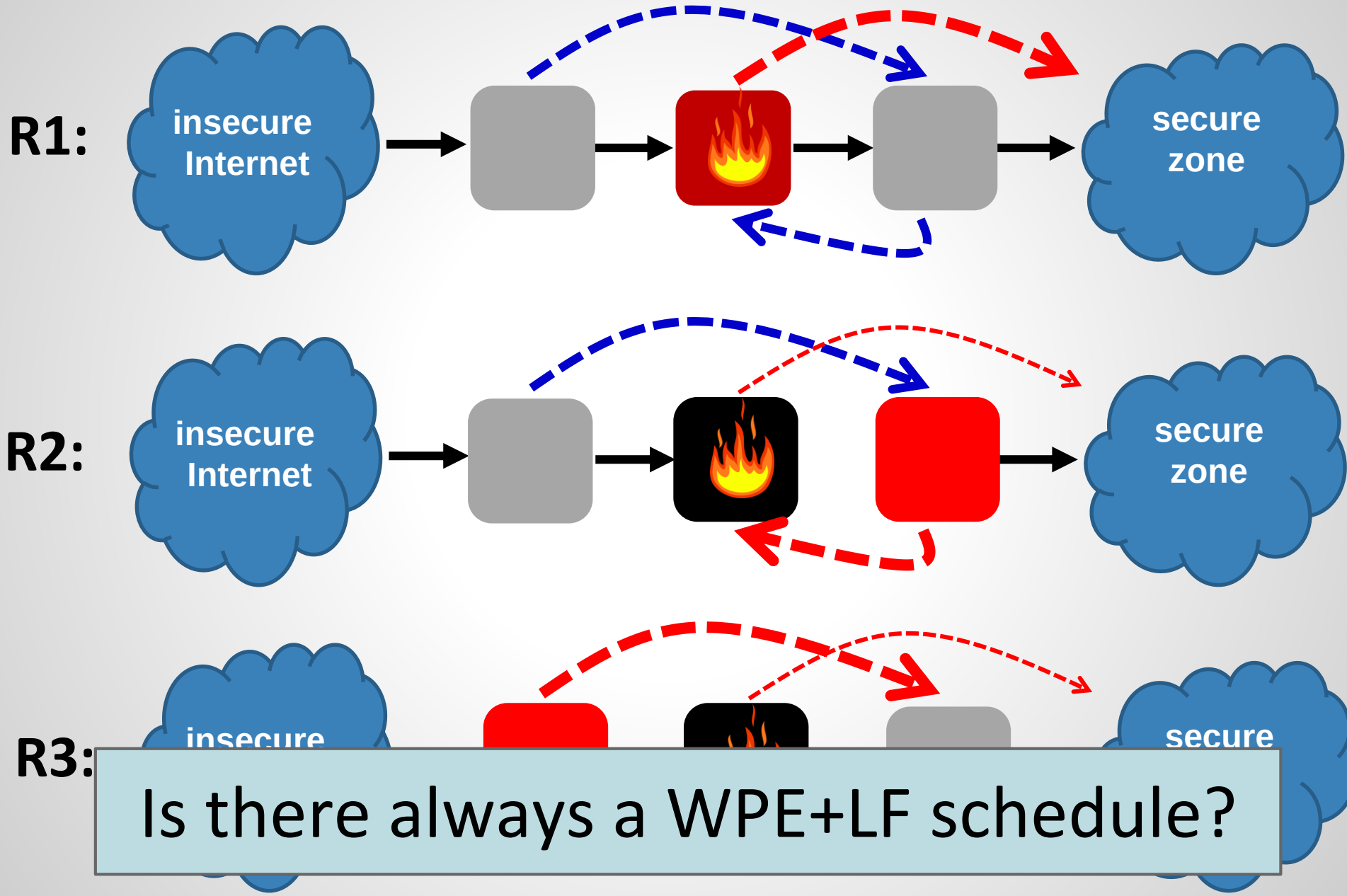
Going Back to Our Examples: Both WPE+LF?



Going Back to Our Examples: WPE+LF!

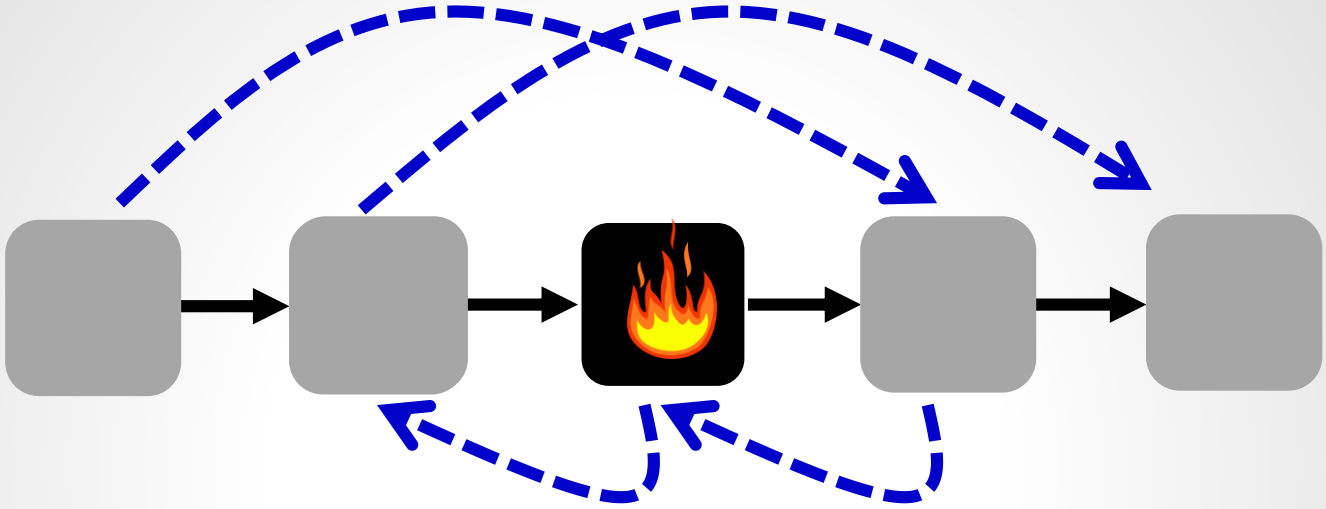


Going Back to Our Examples: WPE+LF!

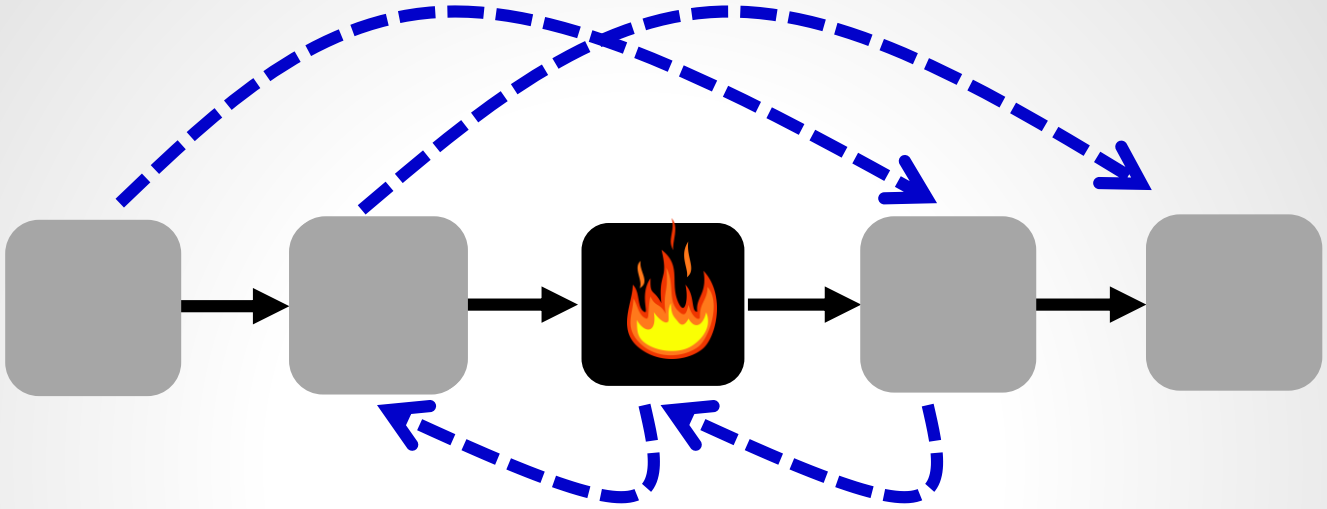


Is there always a WPE+LF schedule?

What about this one?



LF and WPE may conflict!

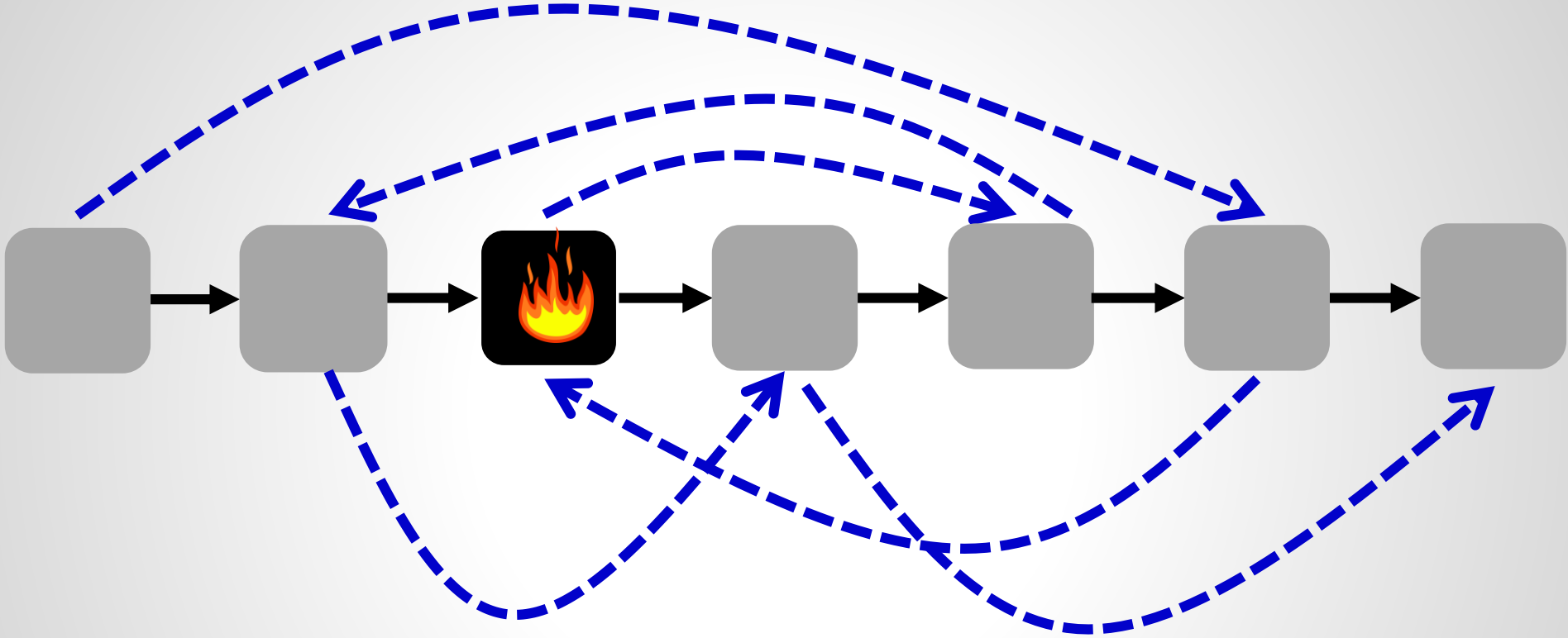


- ❑ Cannot update any **forward edge** in R1: WP
- ❑ Cannot update any **backward edge** in R1: LF

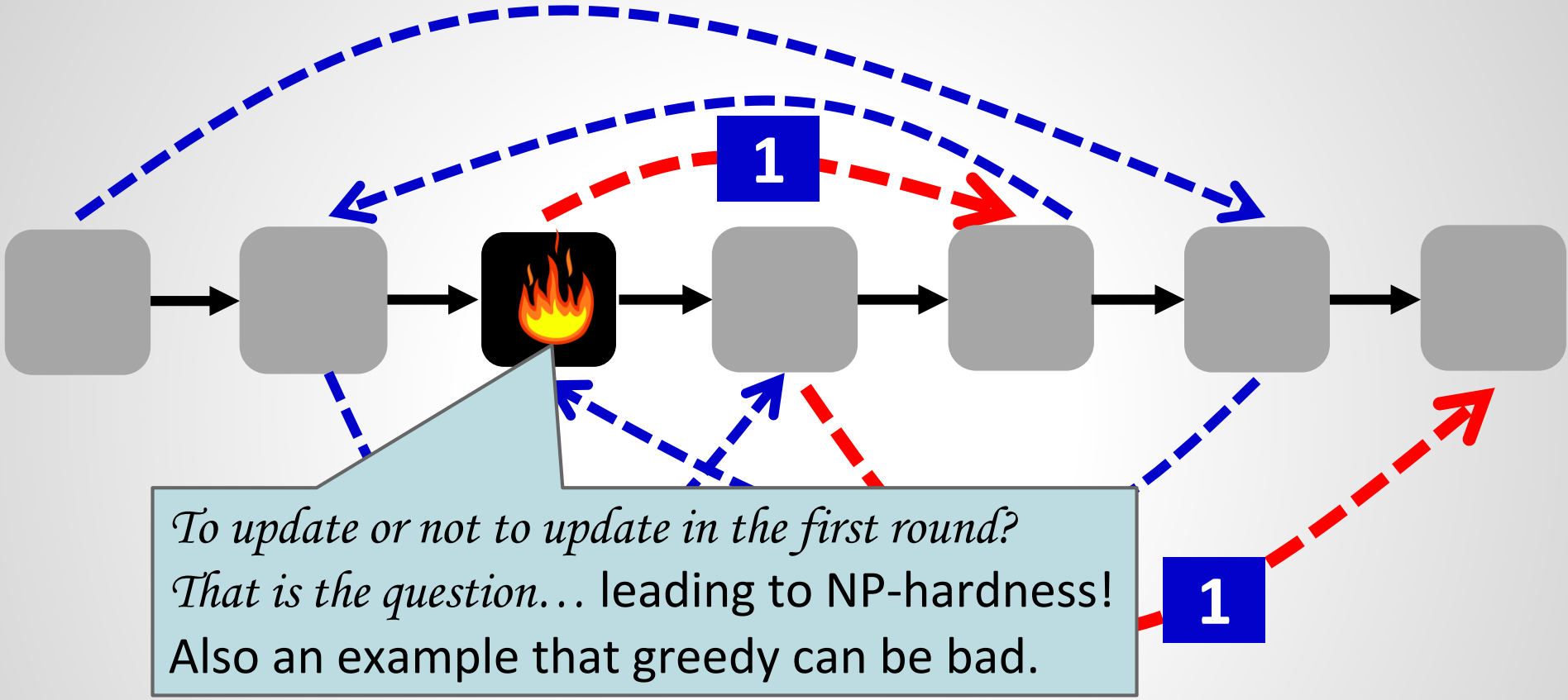
No schedule exists!

Resort to tagging…

What about this one?



NP-Hard!



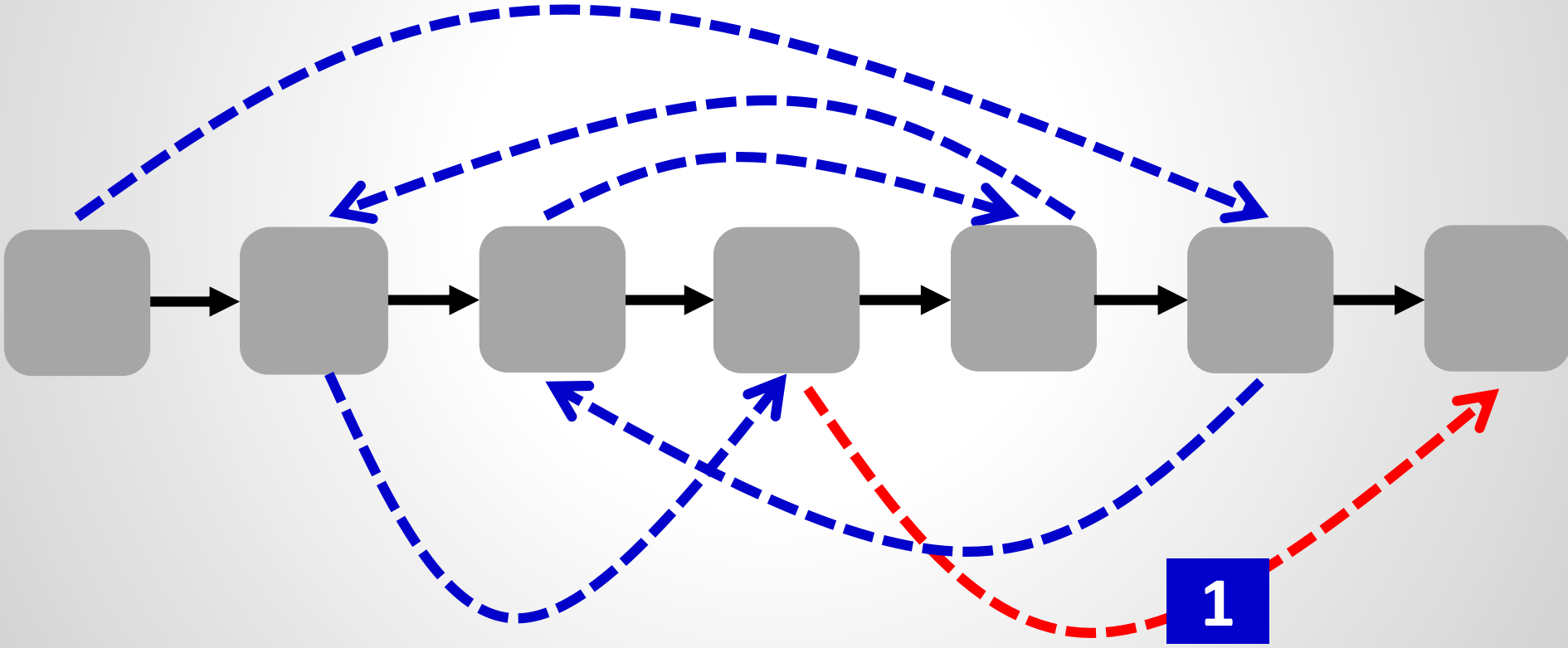
Bad news: Even decidability hard: cannot quickly test feasibility and if infeasible resort to say, tagging solution!

We don't know!

Open question: very artificial? Under which circumstances poly-time?

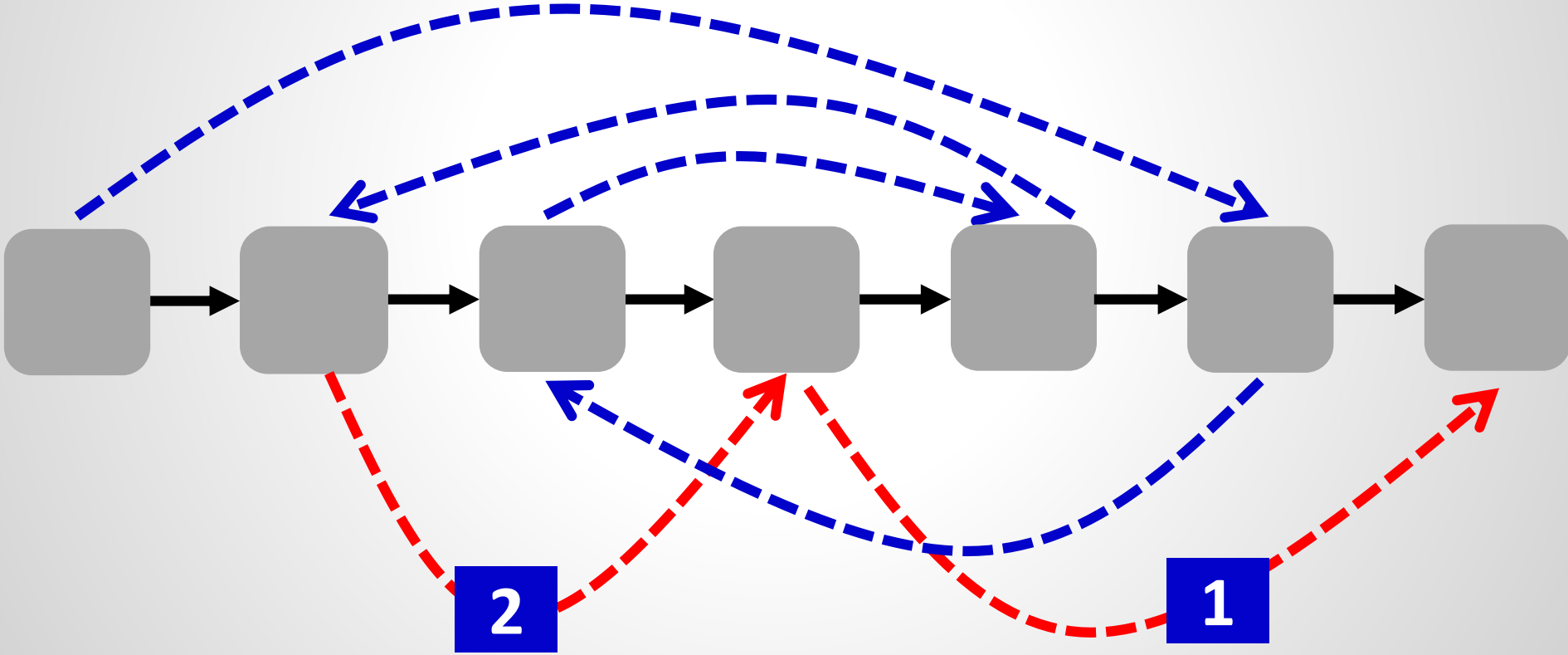
Let us focus on loop-freedom only: always possible in *n* rounds! How?

Let us focus on loop-freedom only:
always possible in *n* rounds! How?



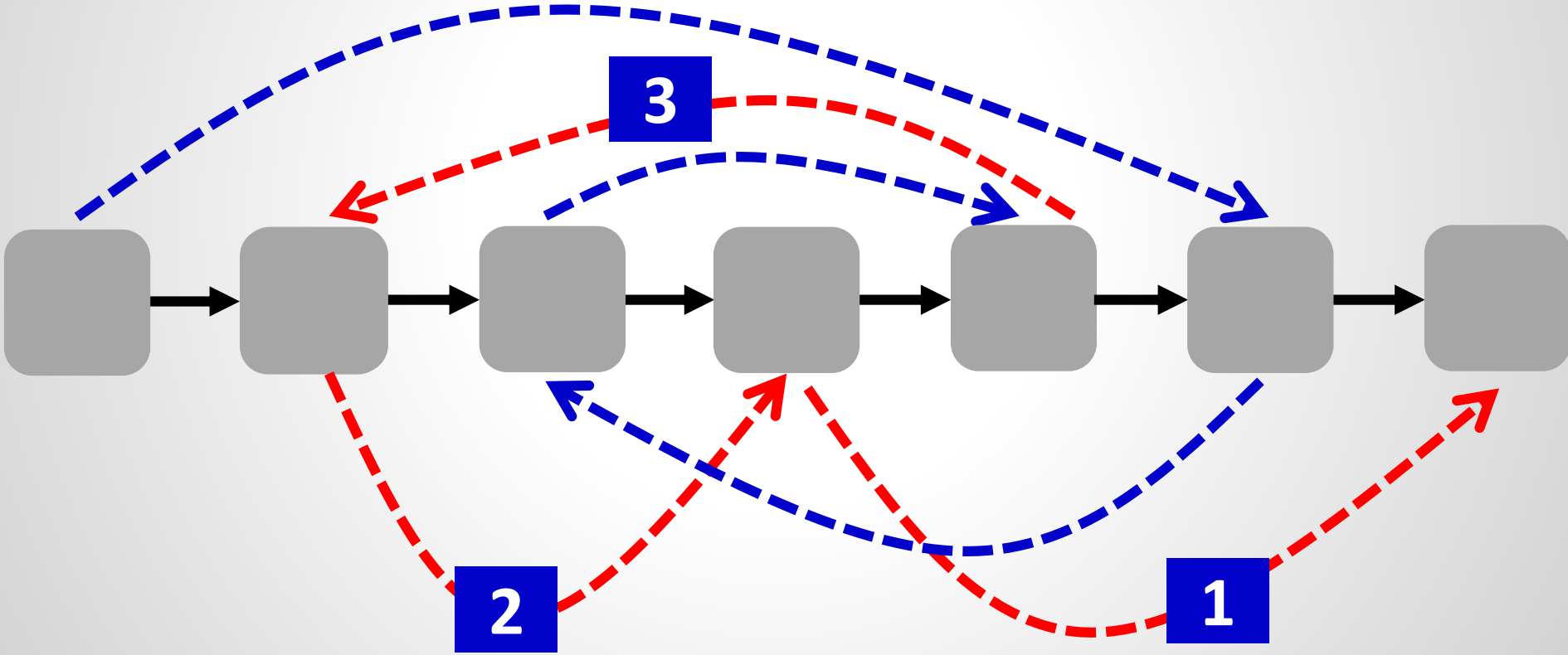
From the destination! Invariant: path suffix updated!

Let us focus on loop-freedom only:
always possible in *n* rounds! How?



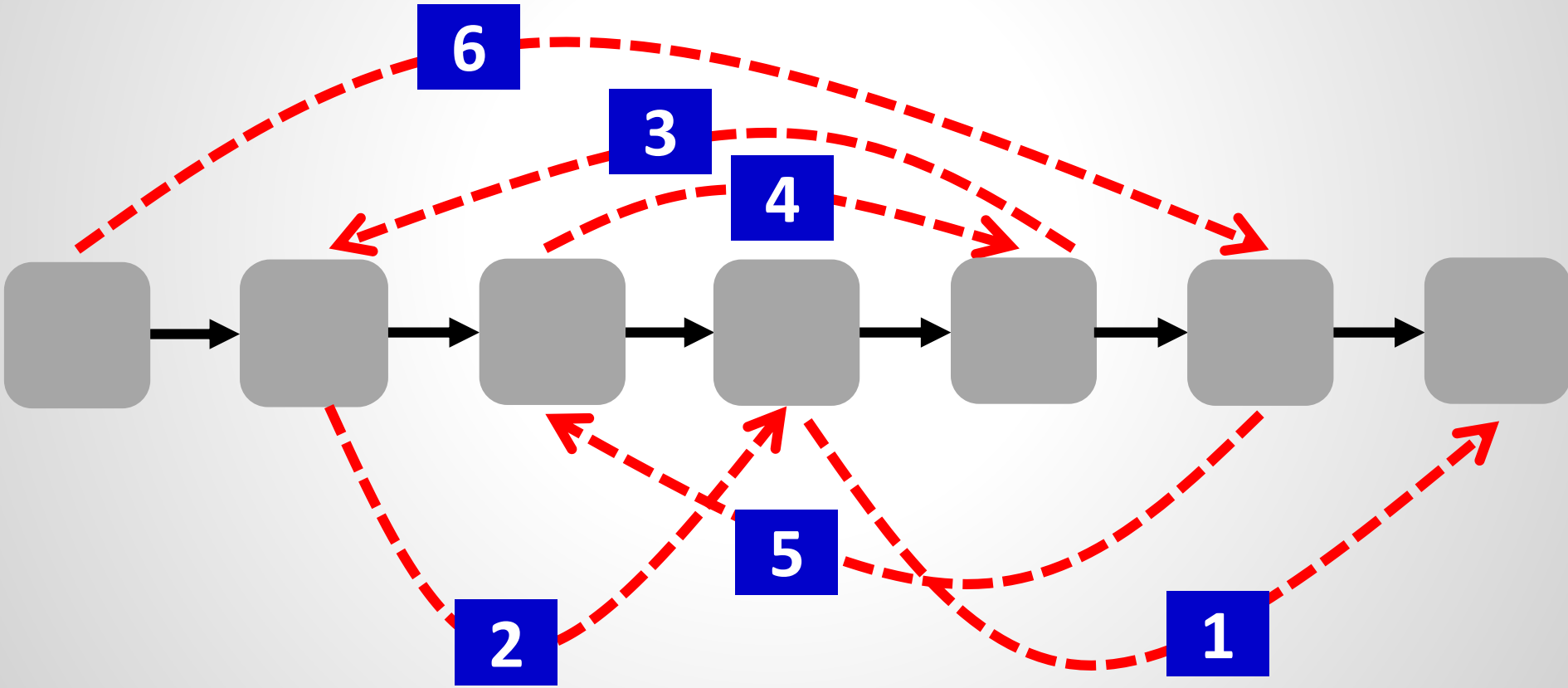
From the destination! Invariant: path suffix updated!

Let us focus on loop-freedom only:
always possible in *n* rounds! How?



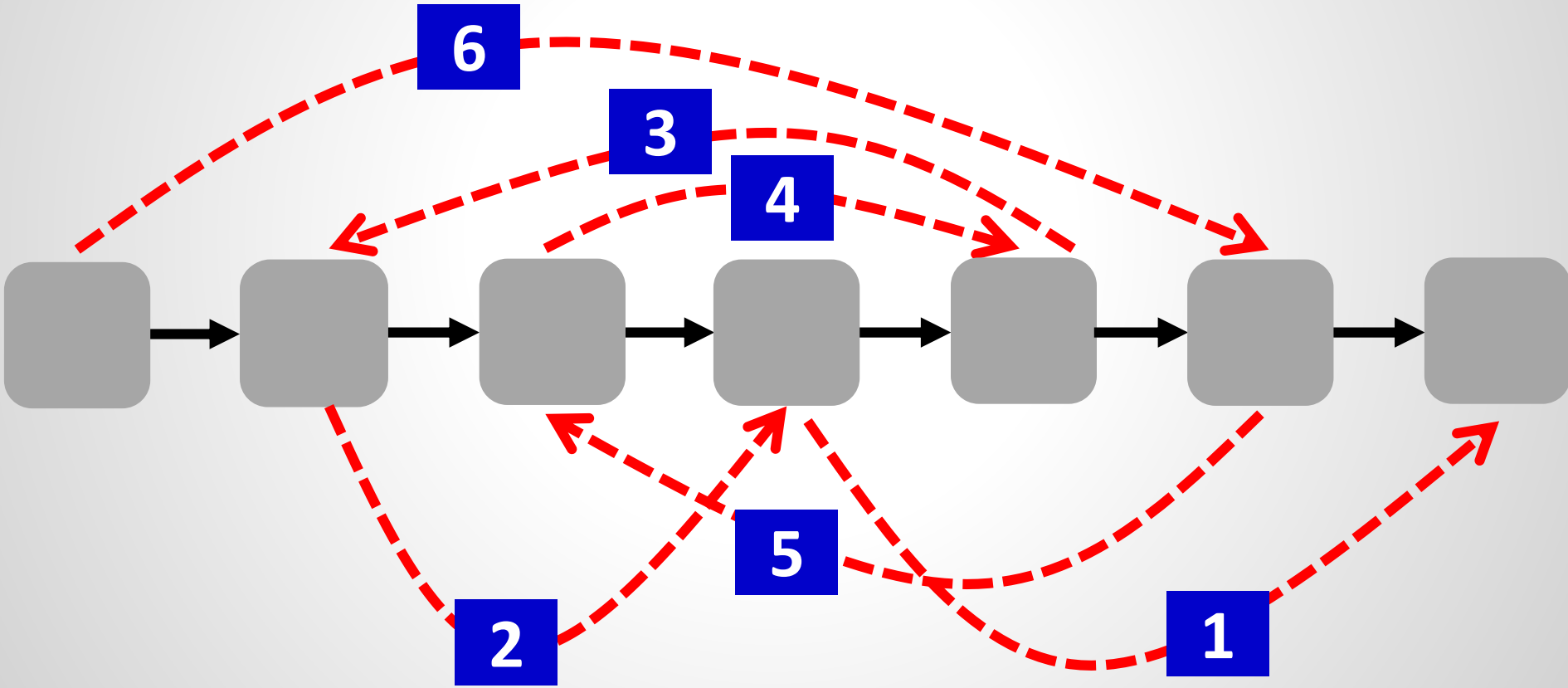
From the destination! Invariant: path suffix updated!

Let us focus on loop-freedom only:
always possible in *n* rounds! How?



From the destination! Invariant: path suffix updated!

Let us focus on loop-freedom only:
always possible in *n* rounds! How?



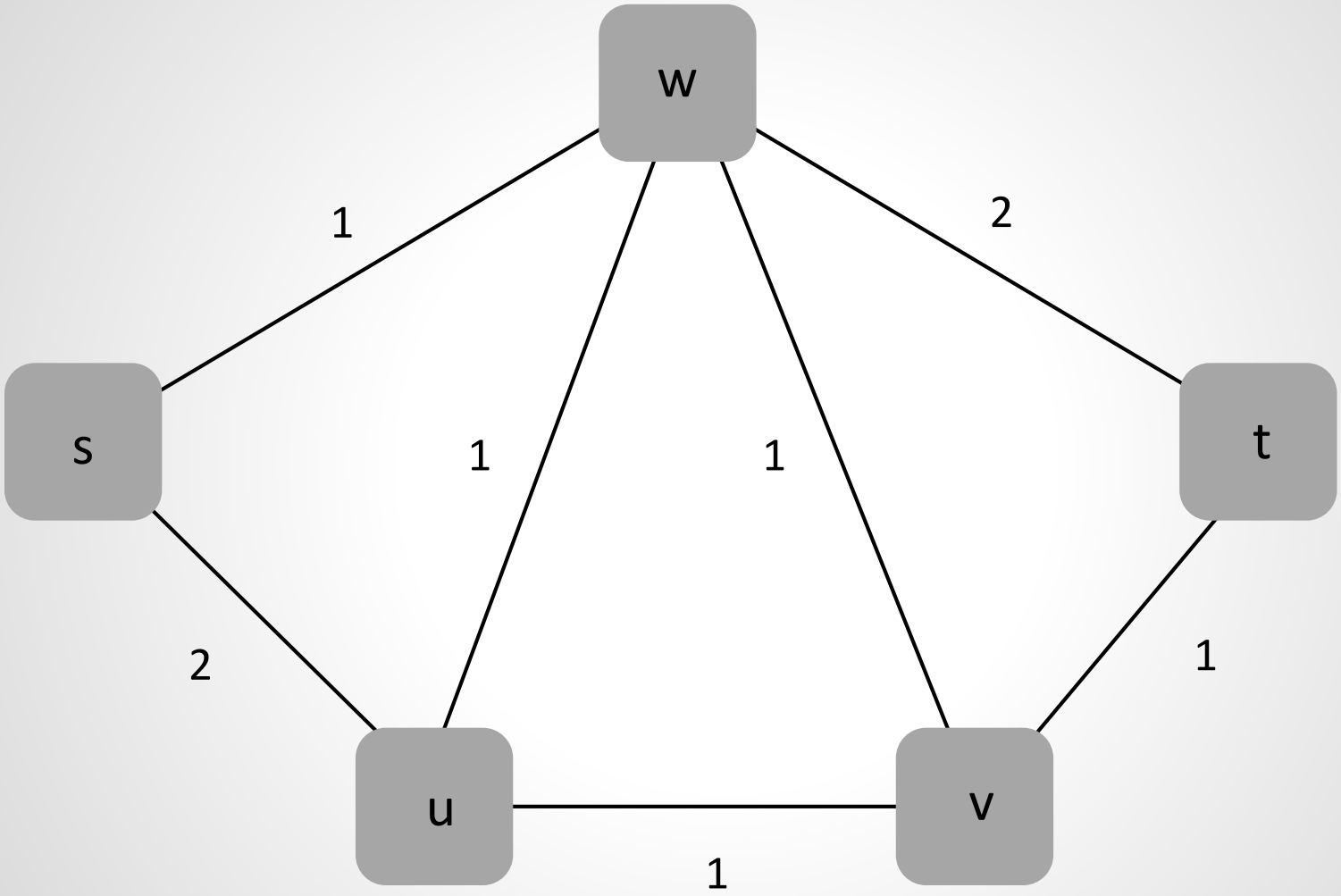
From the destination! Invariant: path suffix updated!

But how to minimize # rounds?

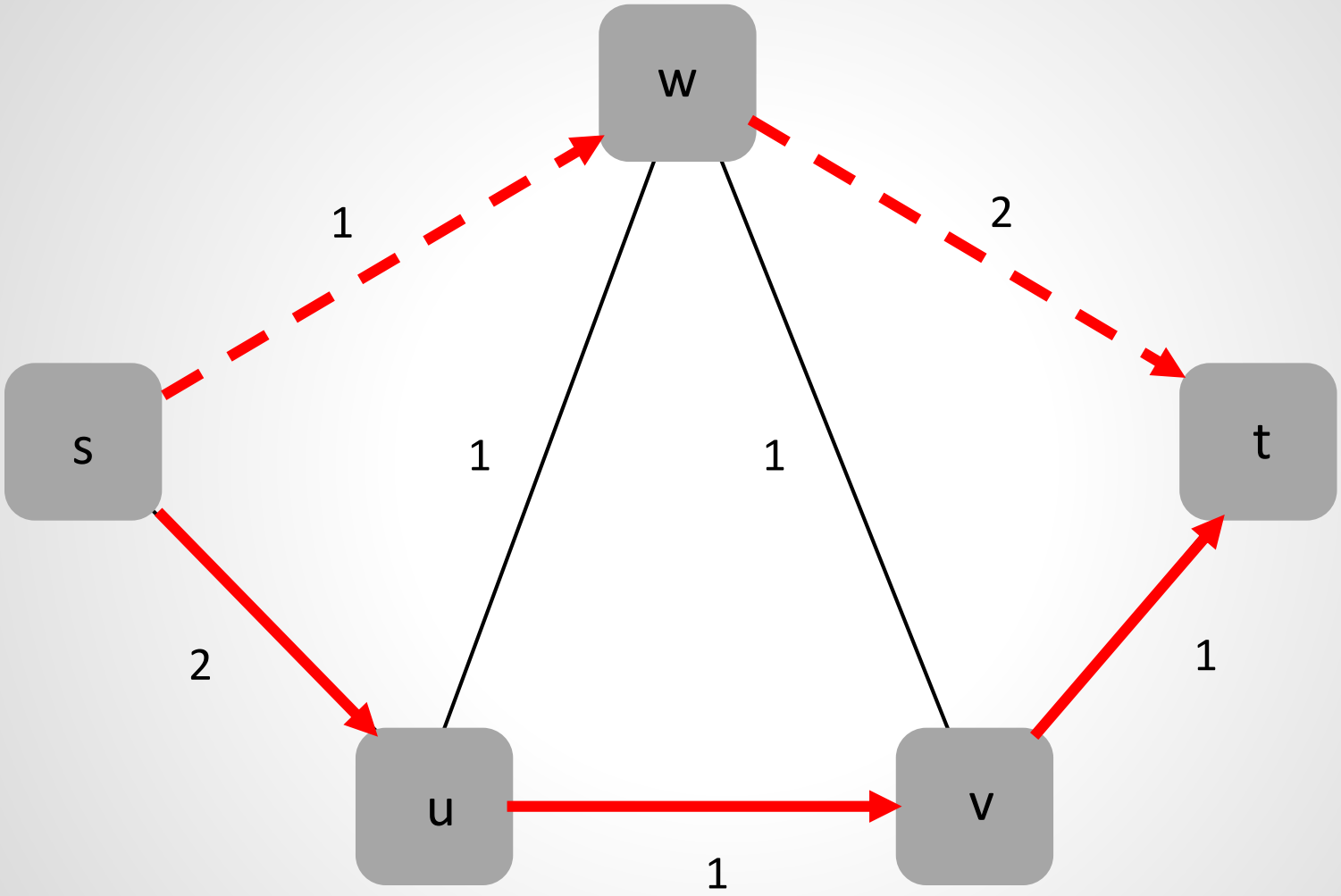
But how to minimize # rounds?

2 rounds easy, 3 rounds NP-hard. Let's take it offline!

What about capacity constraints?

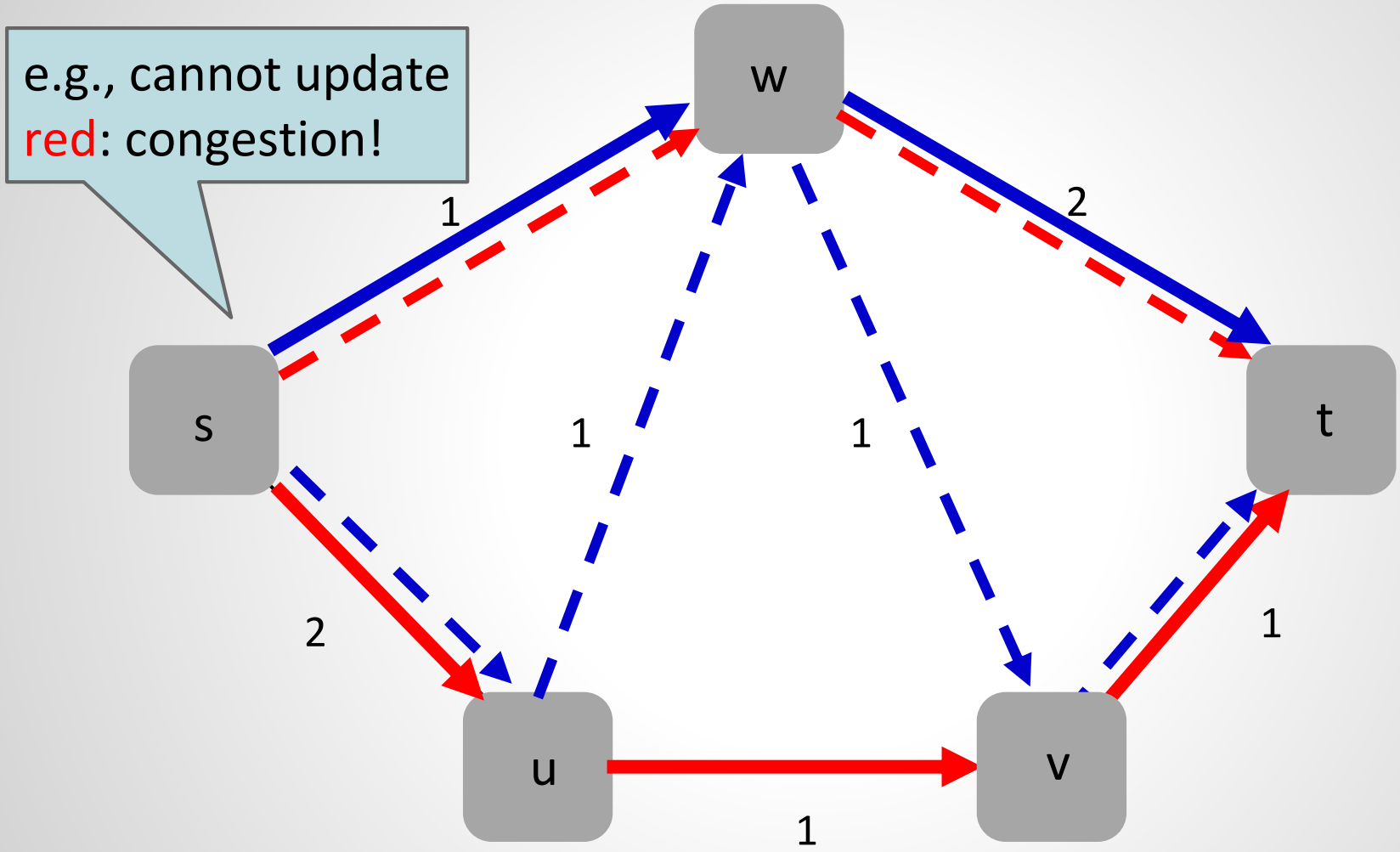


What about capacity constraints?



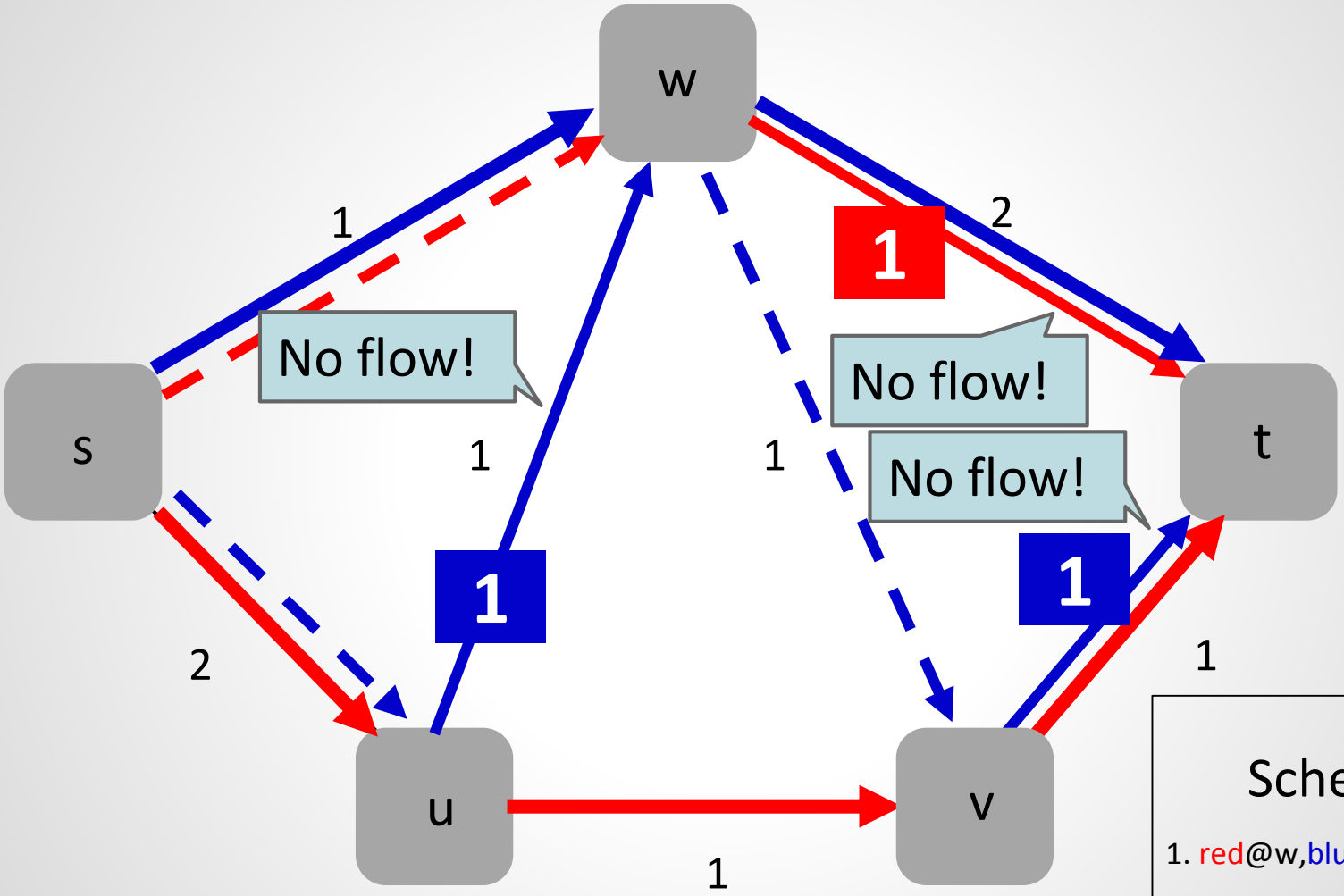
Flow 1

What about capacity constraints?



Can you find an update schedule?

What about capacity constraints?

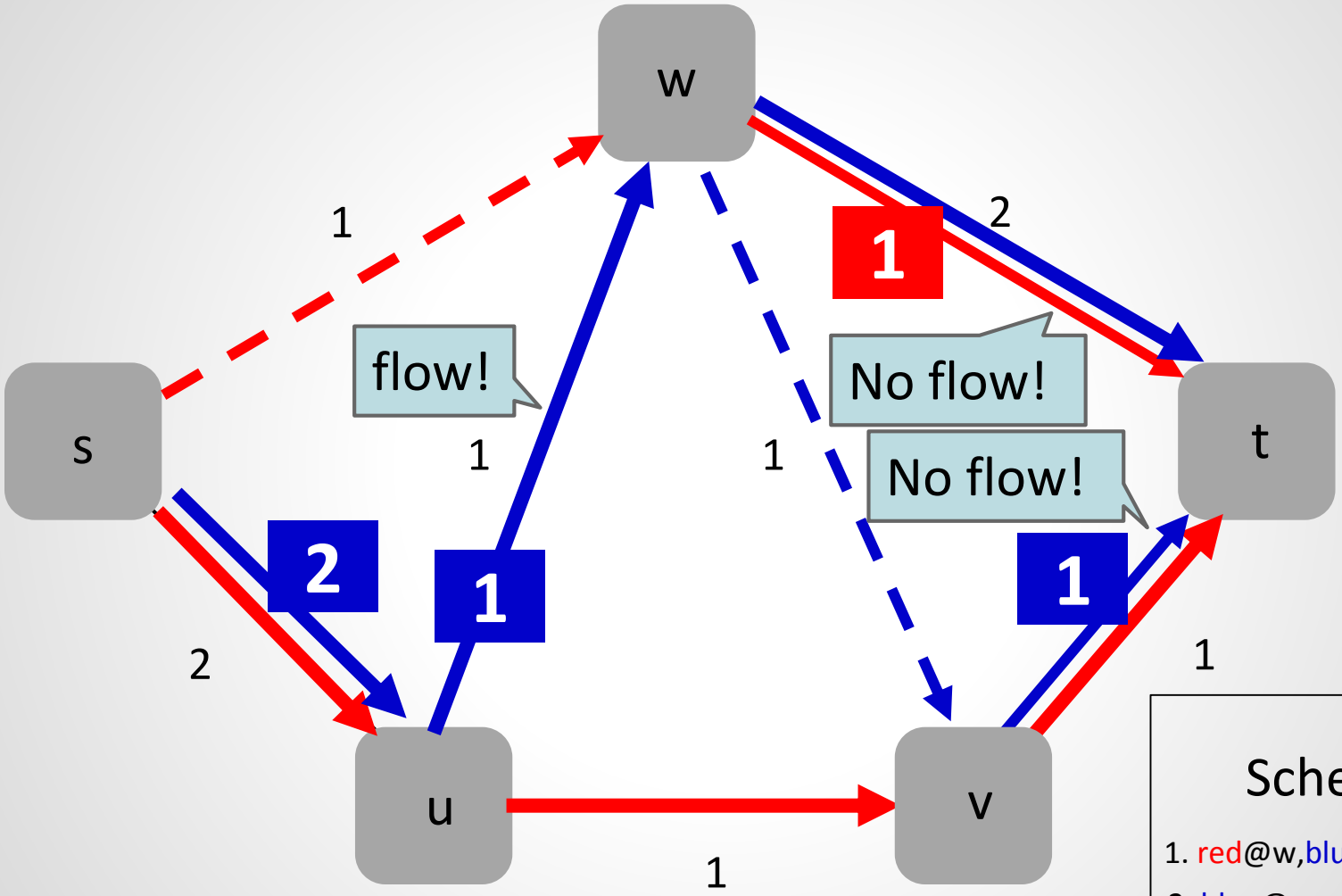


Schedule:

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

Round 1: prepare

What about capacity constraints?

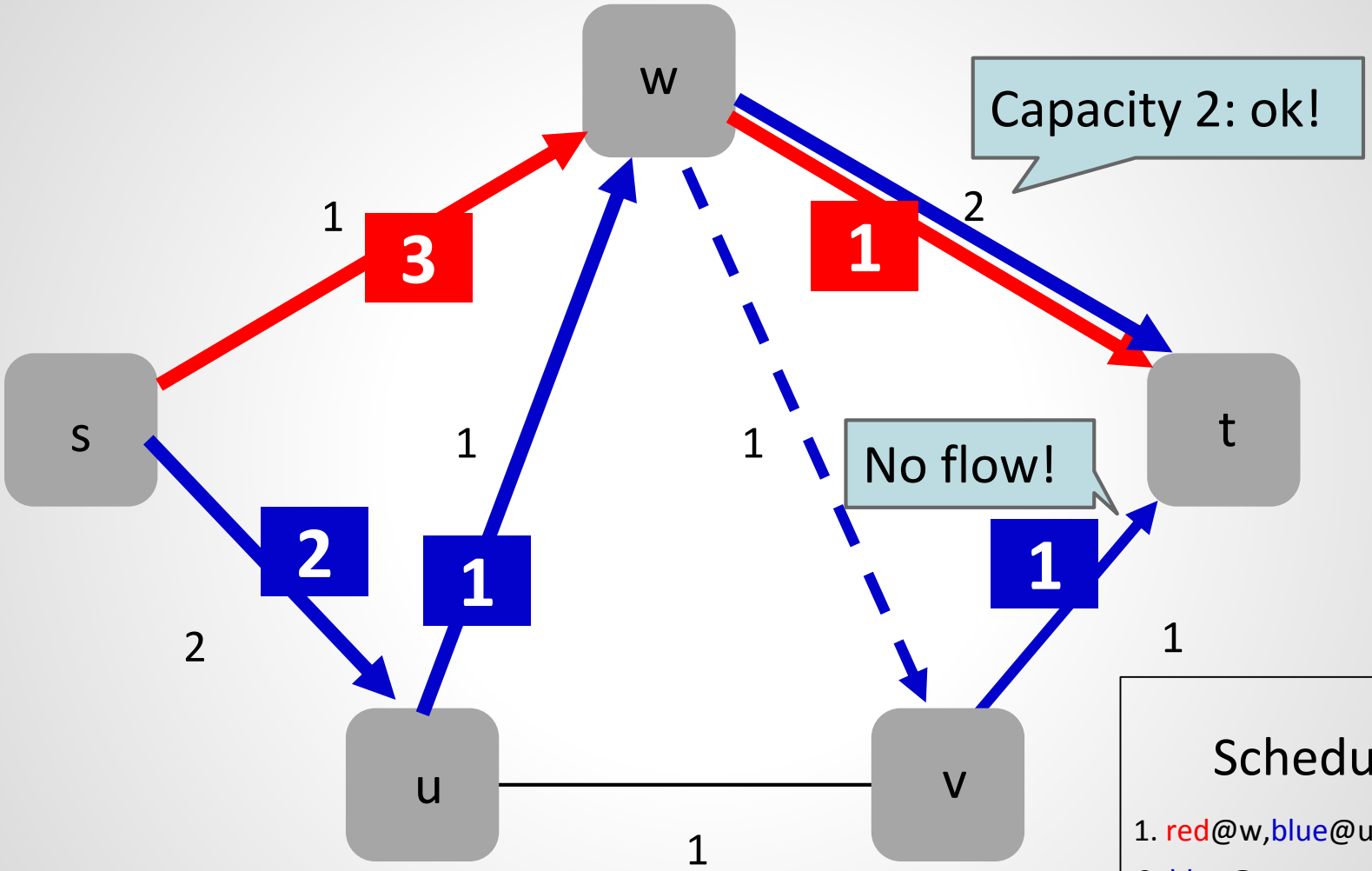


Schedule:

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

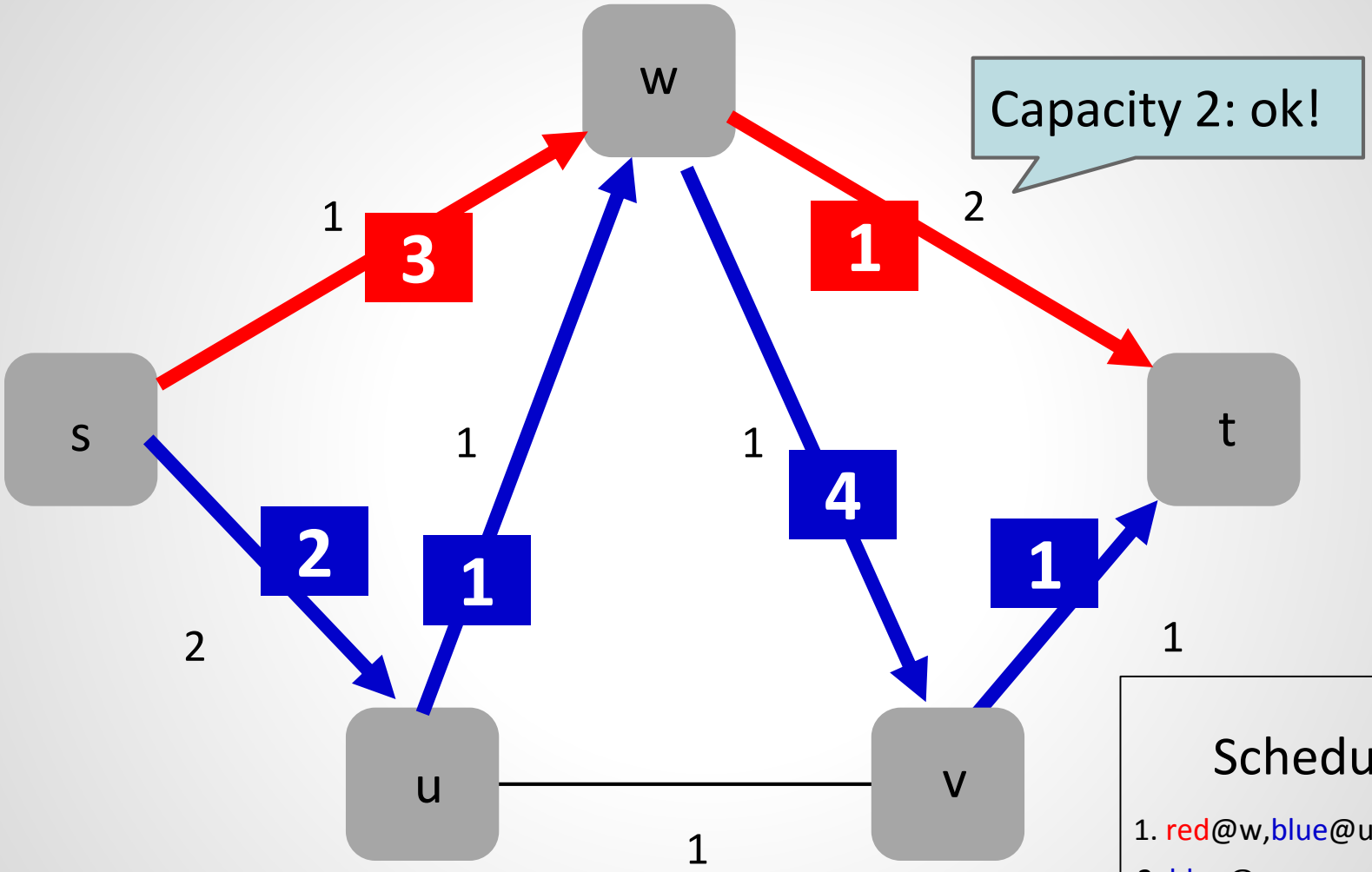
Round 2

What about capacity constraints?



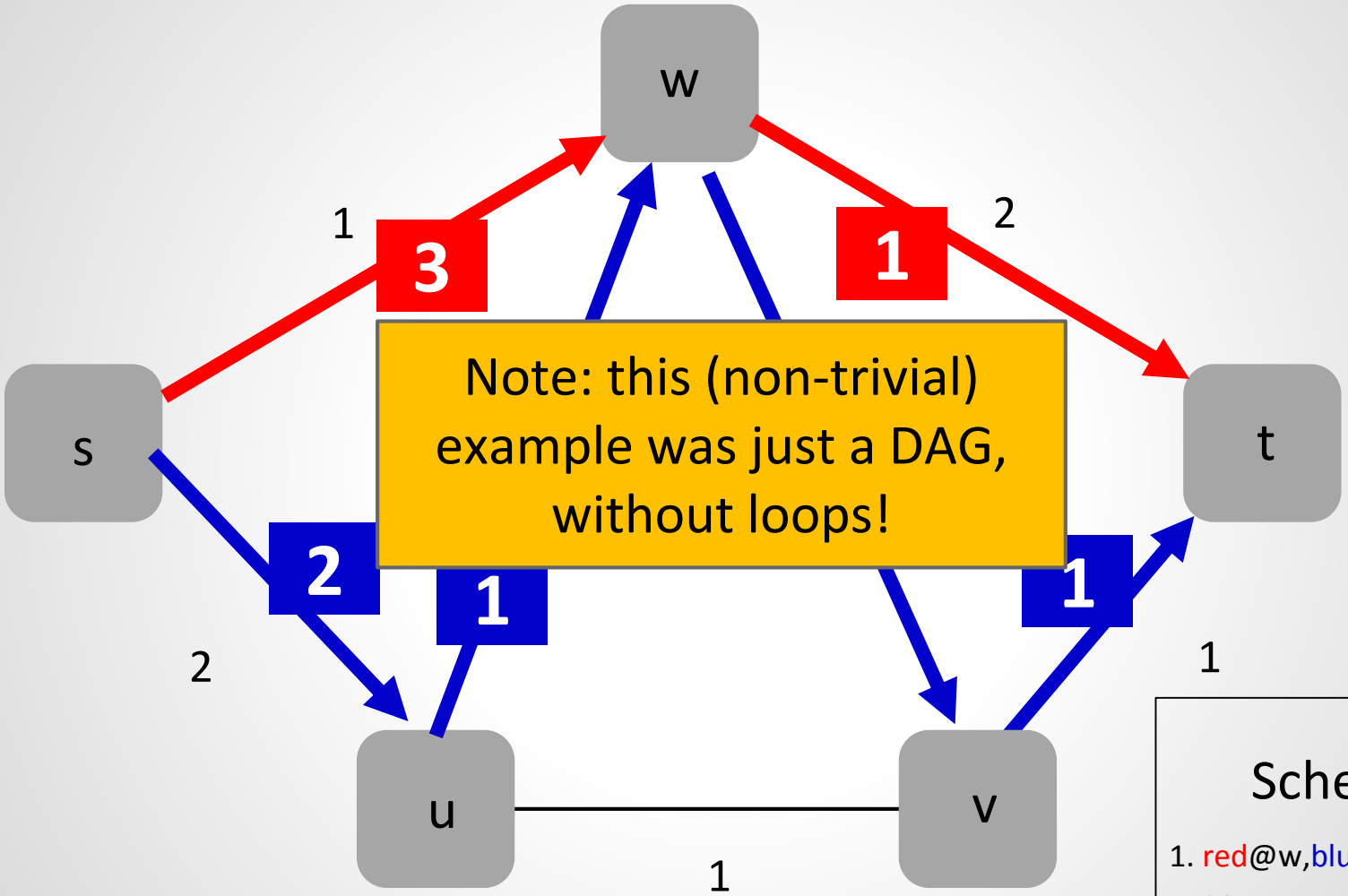
Round 3

What about capacity constraints?



Round 4

What about capacity constraints?



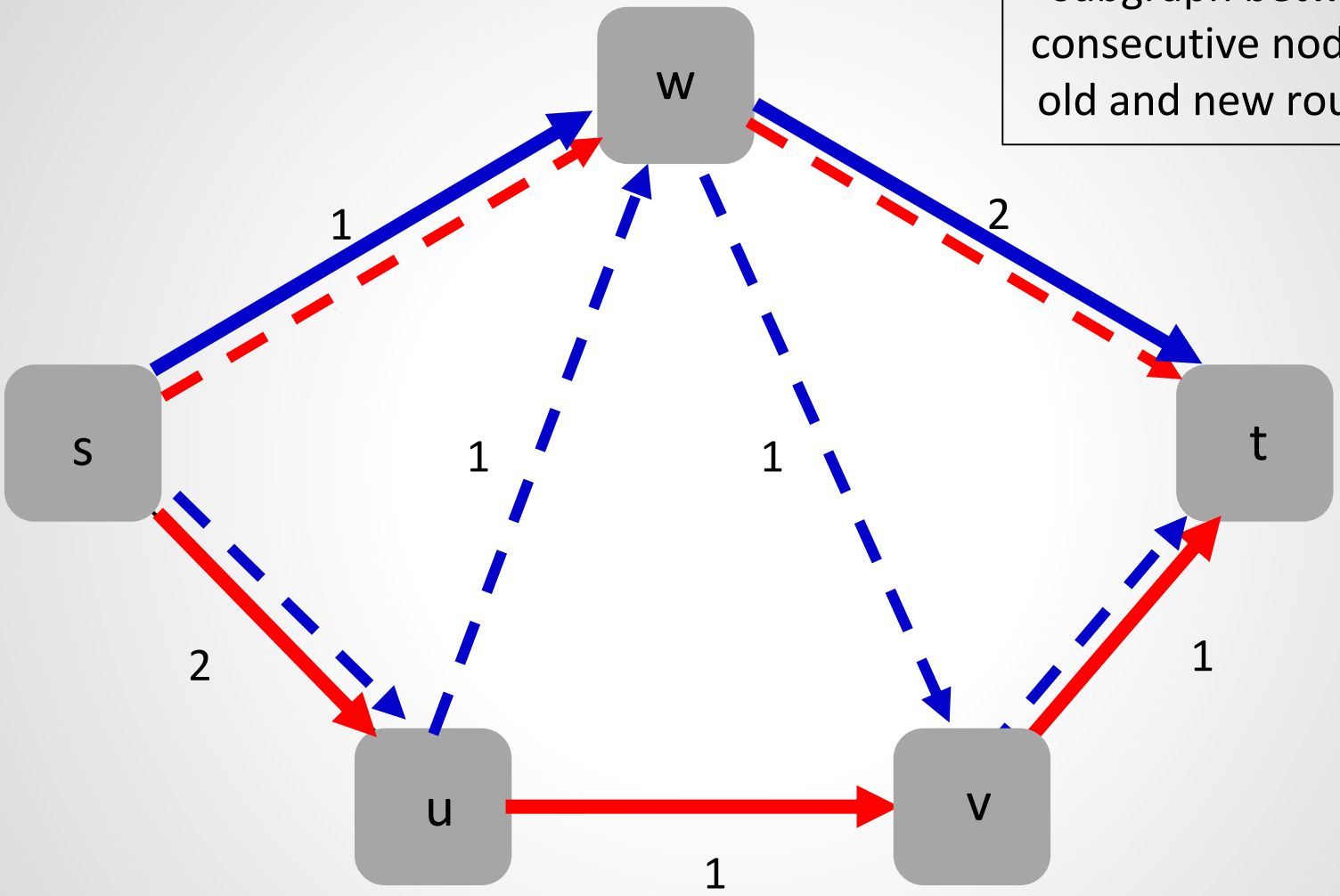
Schedule:

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

Round 4

Block Decomposition of DAGs

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

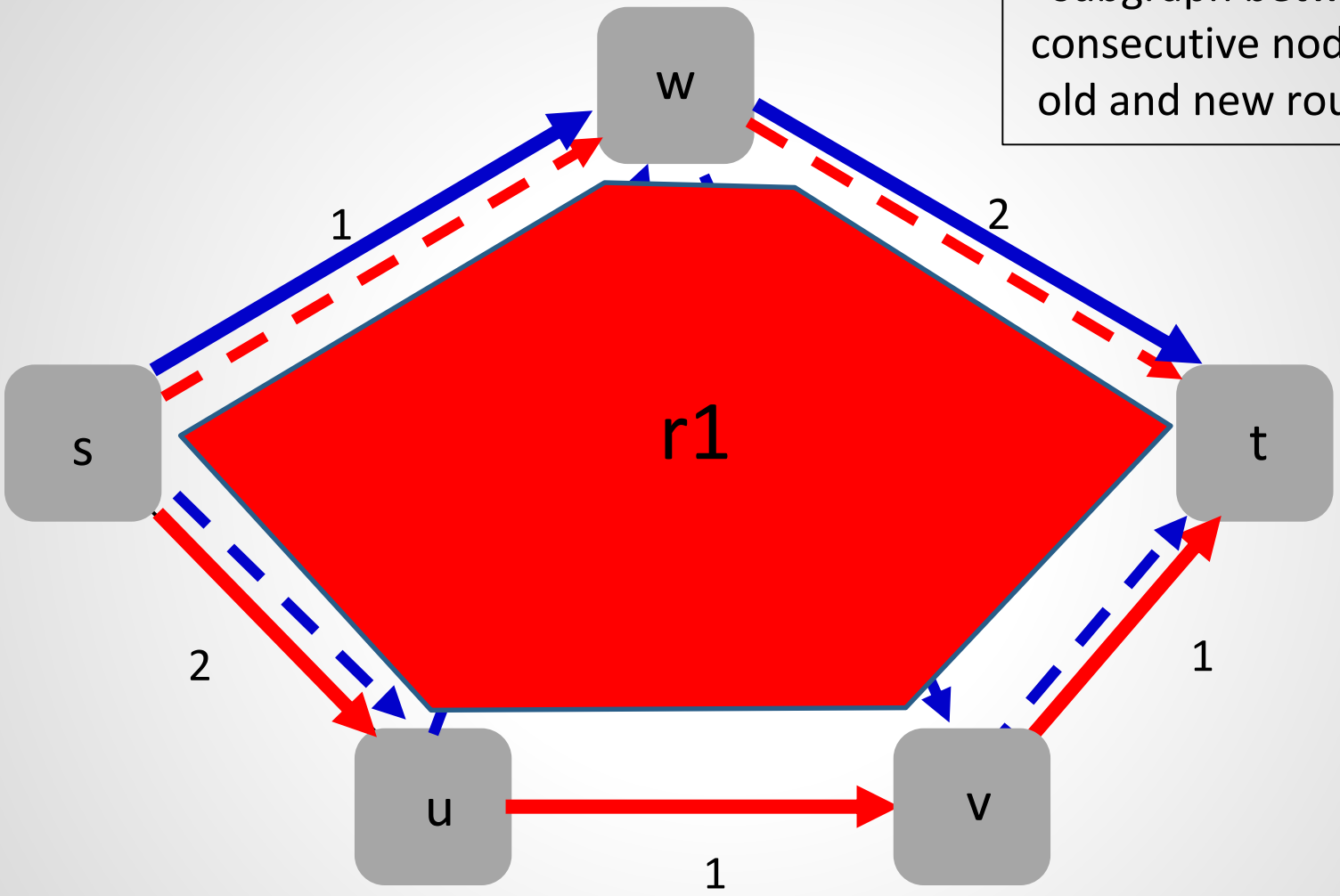


Flow 1

Flow 2

Block Decomposition of DAGs

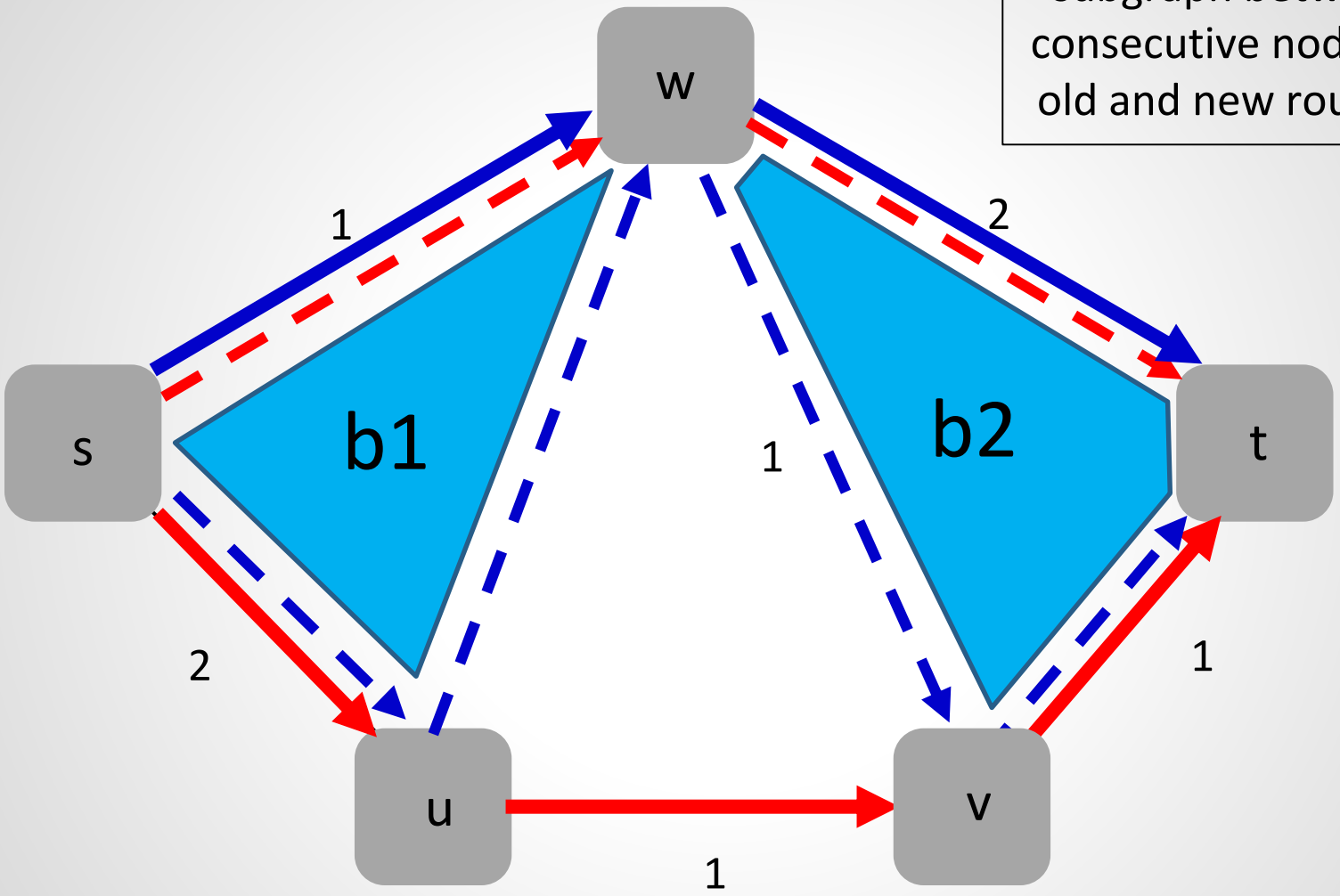
Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Just one red block: r1

Block Decomposition of DAGs

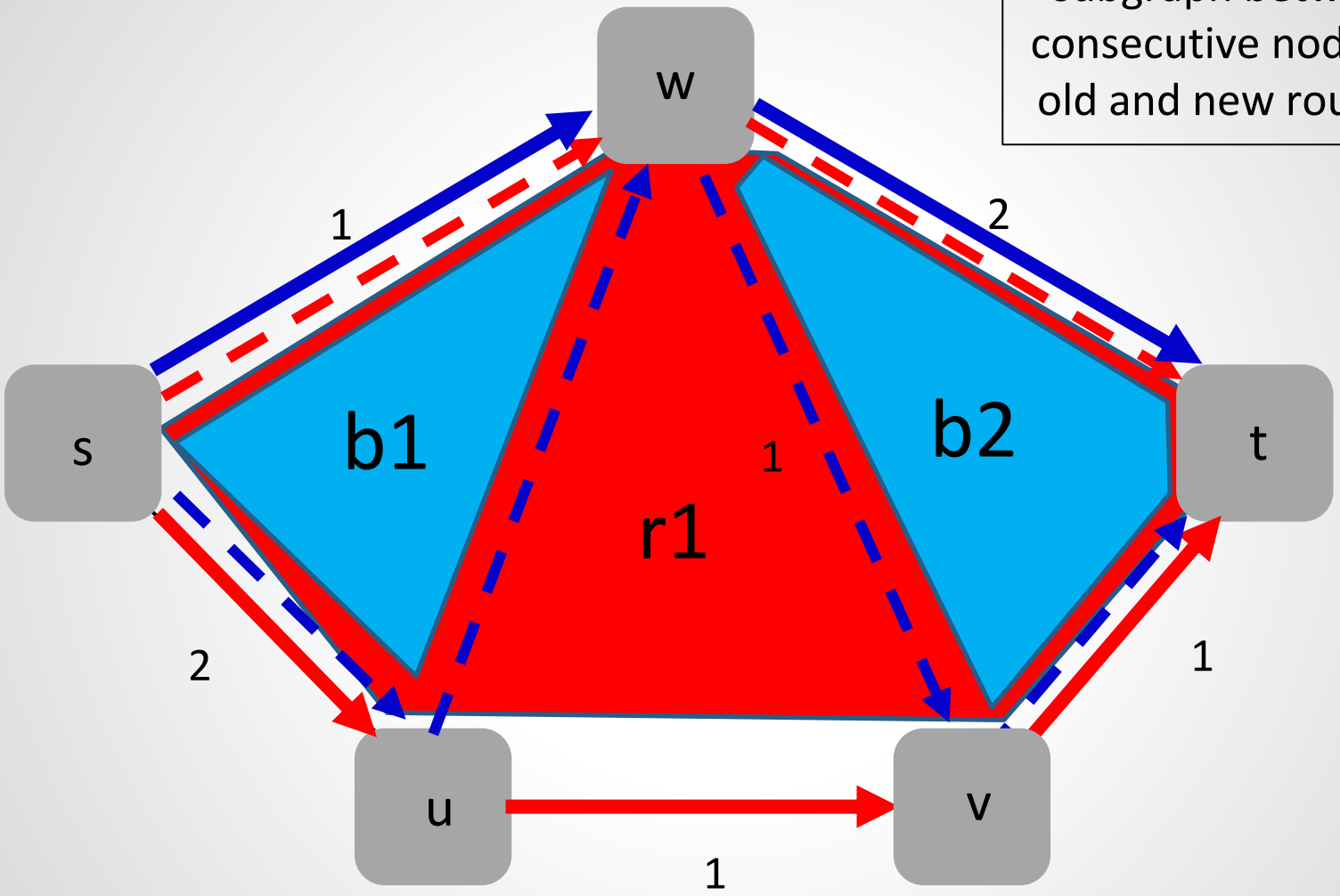
Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Two blue blocks: b1 and b2

Block Decomposition of DAGs

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



Dependencies: update b2 after r1 after b1.

Algorithms and Properties

- For k=2 flows
  - Using dependency graph of DAG block decomposition: feasible update exists if and only if cycle-free dependency
  - Also directly yields optimal number of rounds!
- For general k flows
  - Harder: We need a weaker notion of dependency graph
  - Only feasibility for constant k in polynomial-time
  - For general k, NP-hard
- Not much more is known so far
  - NP-hard on general networks already for 2 flows

Further Reading

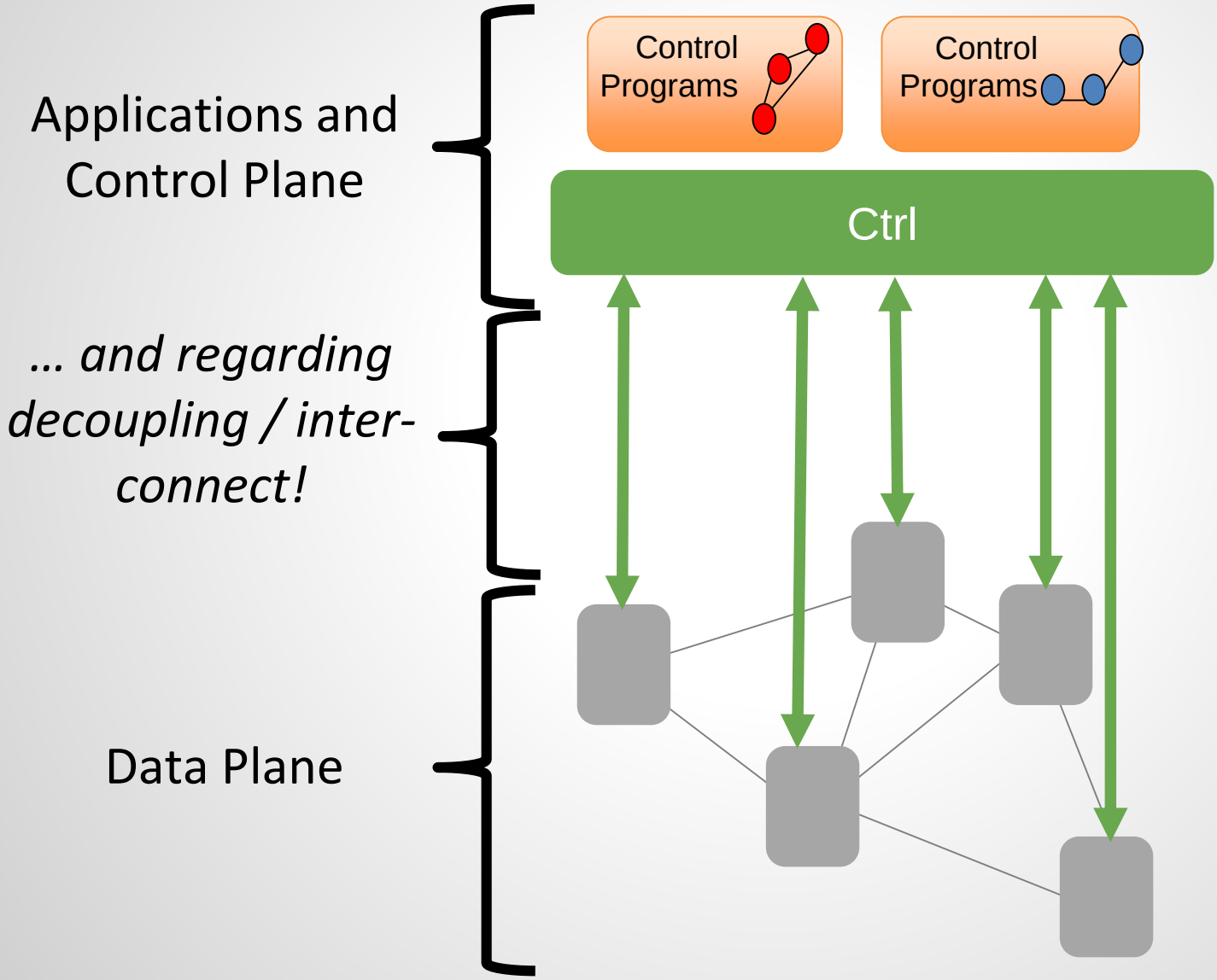
Can't Touch This: Consistent Network Updates for Multiple Policies
Szymon Dudycz, Arne Ludwig, and Stefan Schmid.
46th IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**), Toulouse, France, June 2016.

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

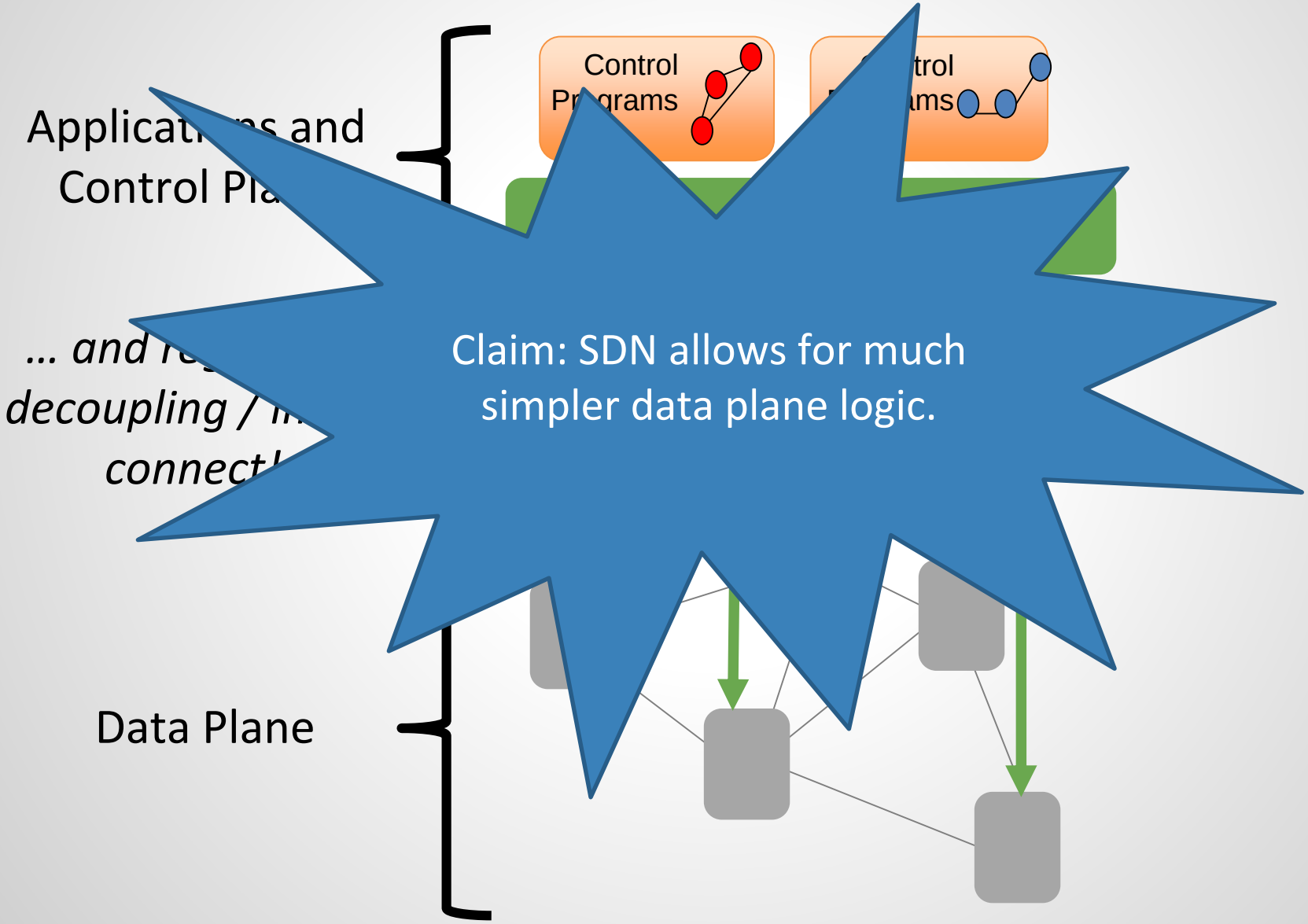
Scheduling Loop-free Network Updates: It's Good to Relax!
Arne Ludwig, Jan Marcinkowski, and Stefan Schmid.
ACM Symposium on Principles of Distributed Computing (**PODC**), Donostia-San Sebastian, Spain, July 2015.

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

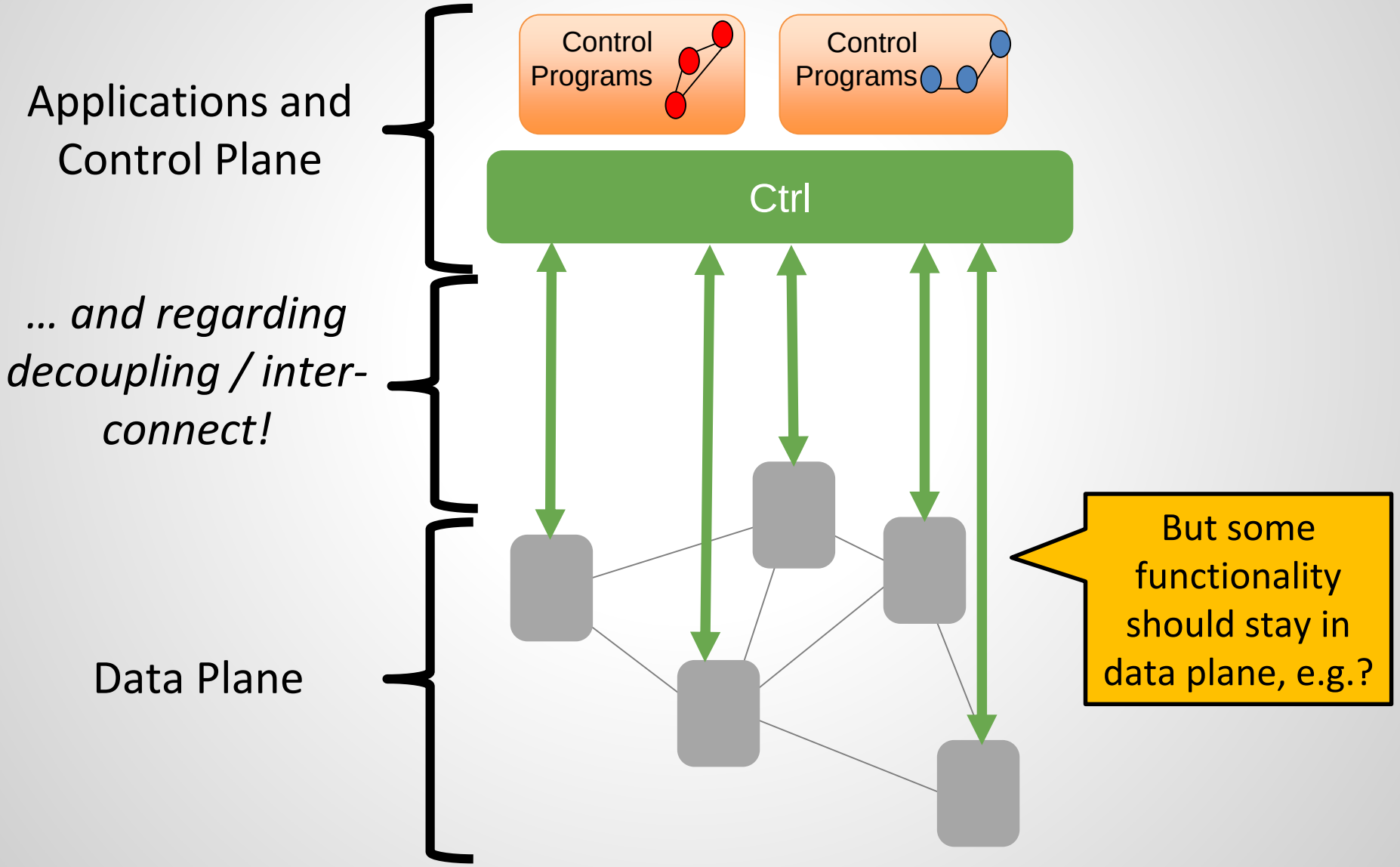
Algorithmic Problems in SDNs



Algorithmic Problems in SDNs

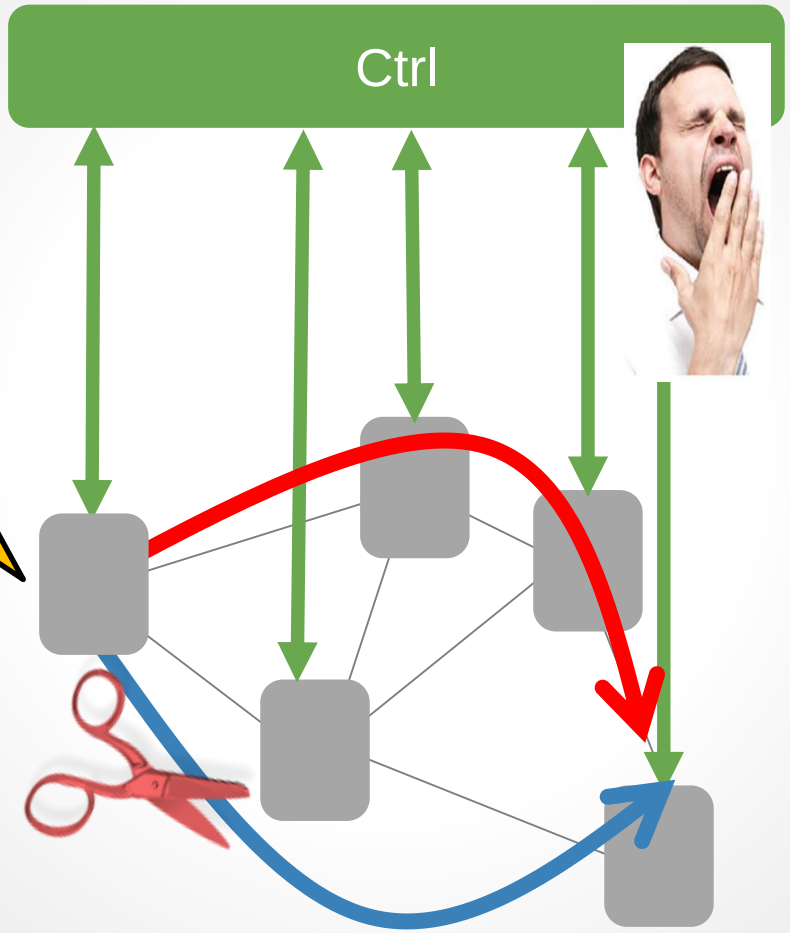


Algorithmic Problems in SDNs



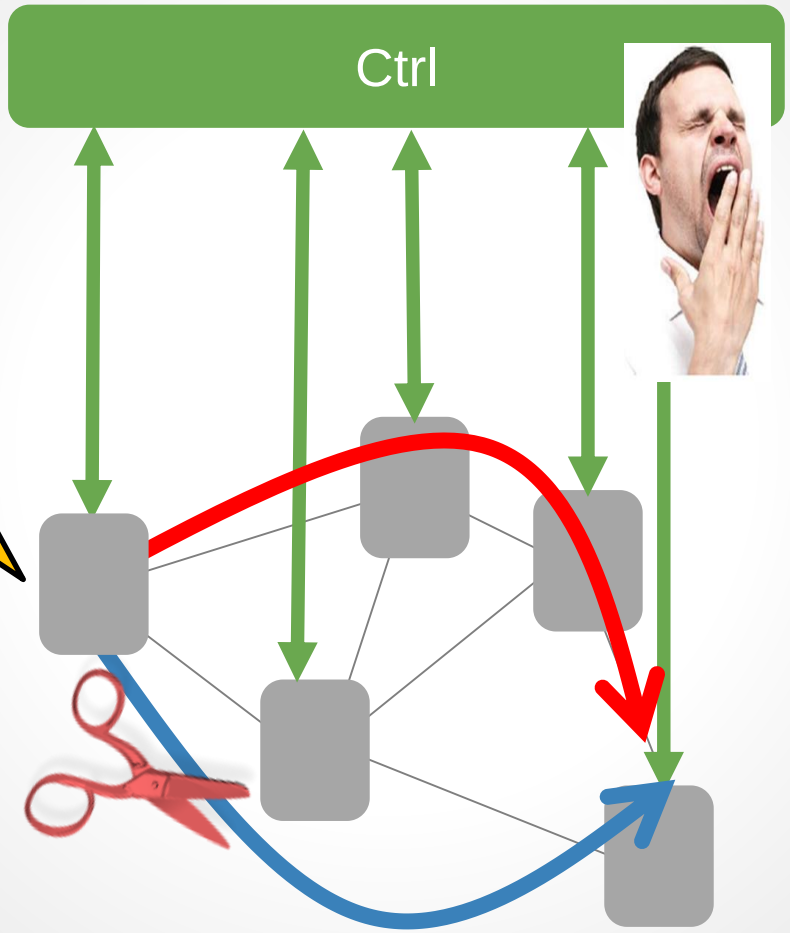
Should Stay in Data Plane: Local Fast Failover

OpenFlow supports preconfigured failover rules: First line of defense. Via controller too slow!



Should Stay in Data Plane: Local Fast Failover

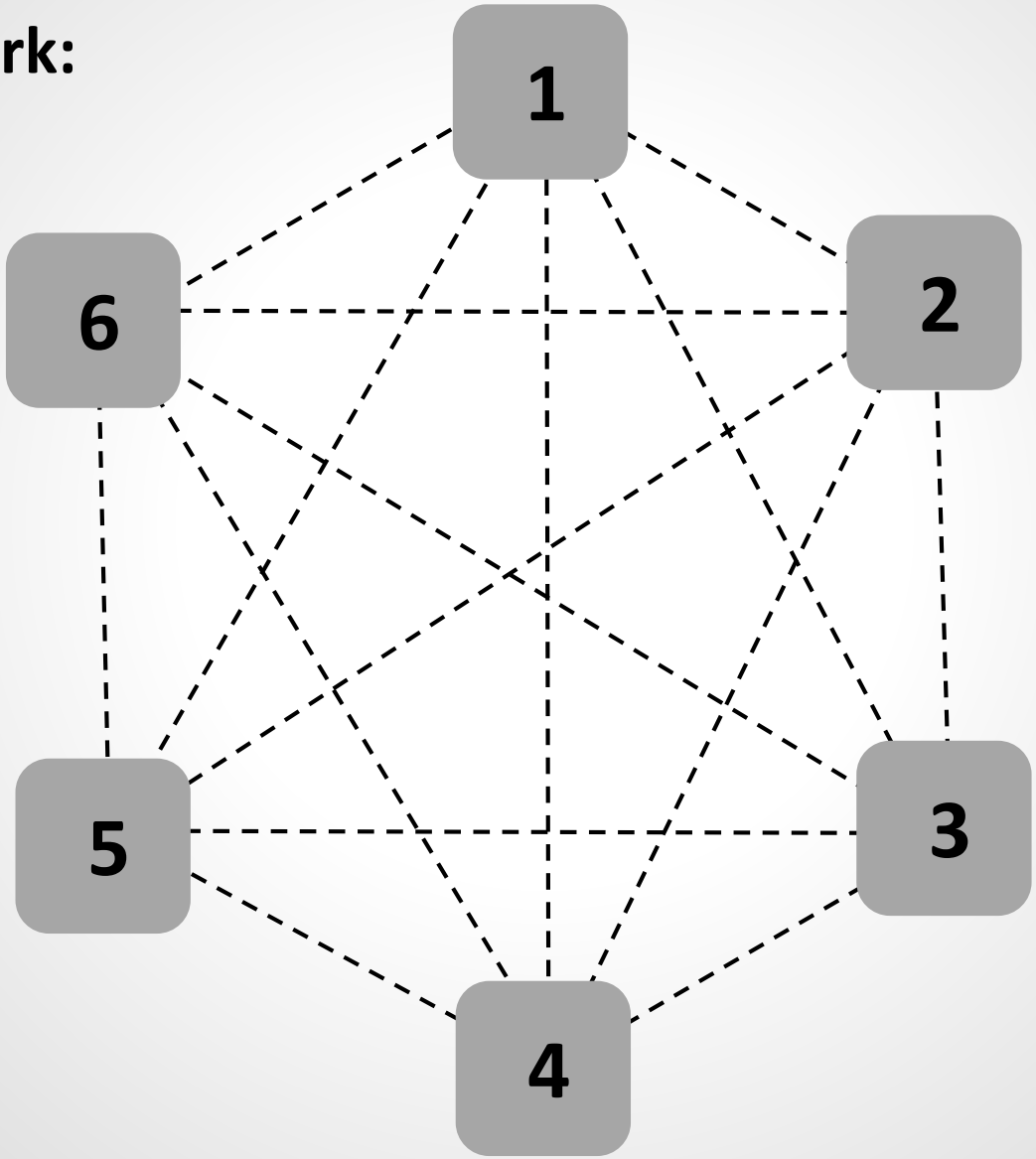
OpenFlow supports preconfigured failover rules: First line of defense. Via controller too slow!



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

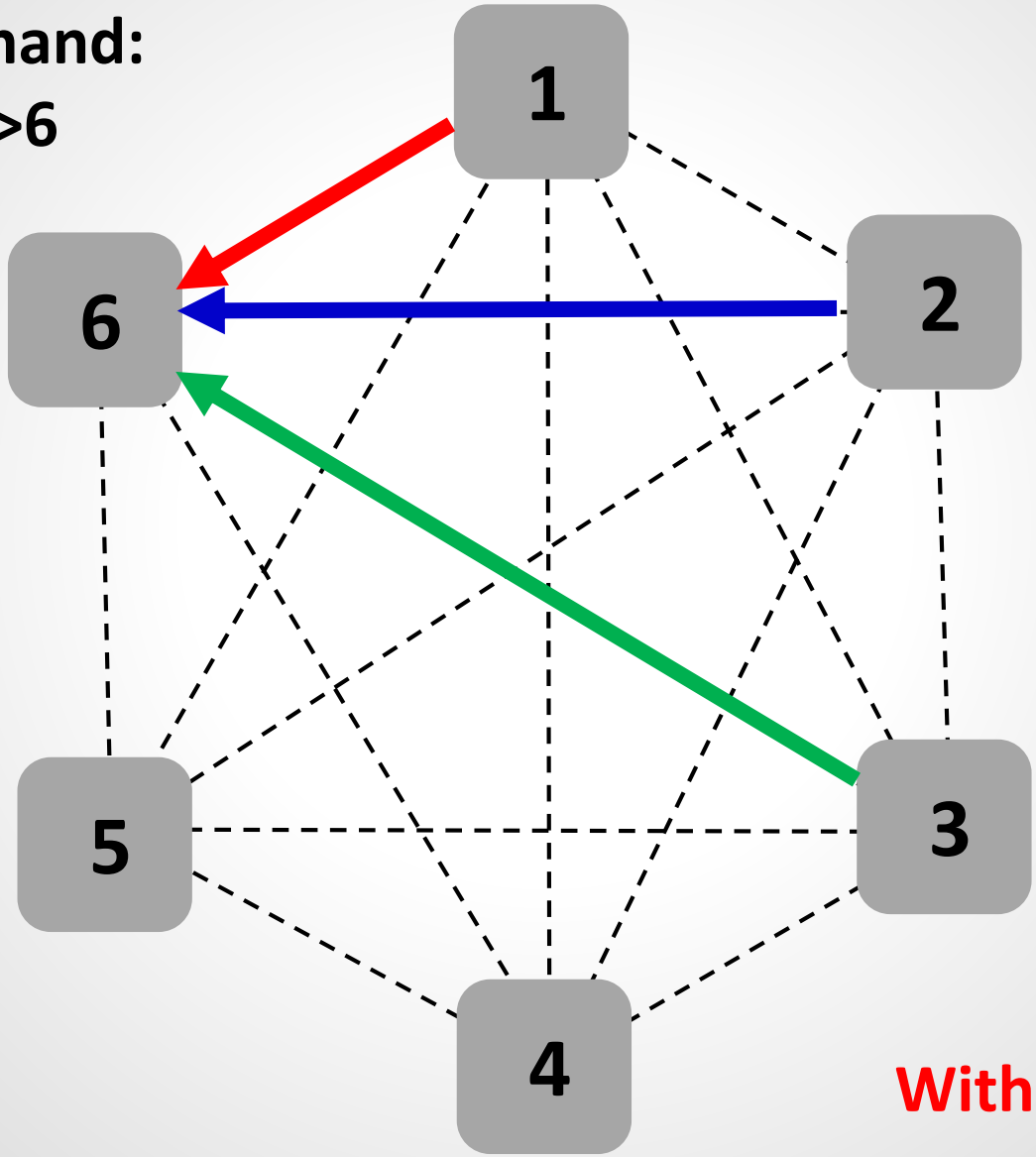
Local Fast Failover

The network:



Local Fast Failover

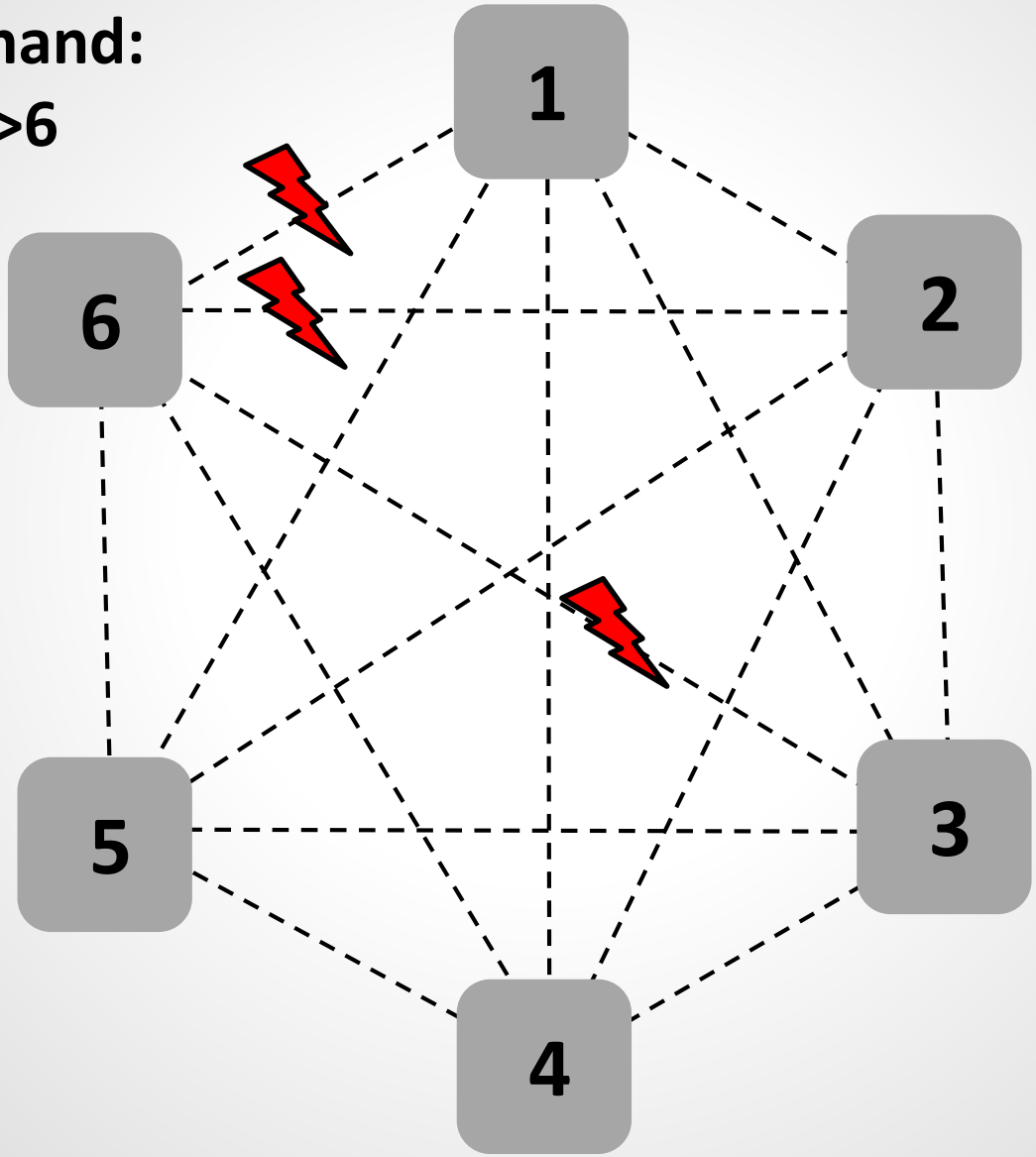
Traffic demand:
{1,2,3}->6



Without failures!

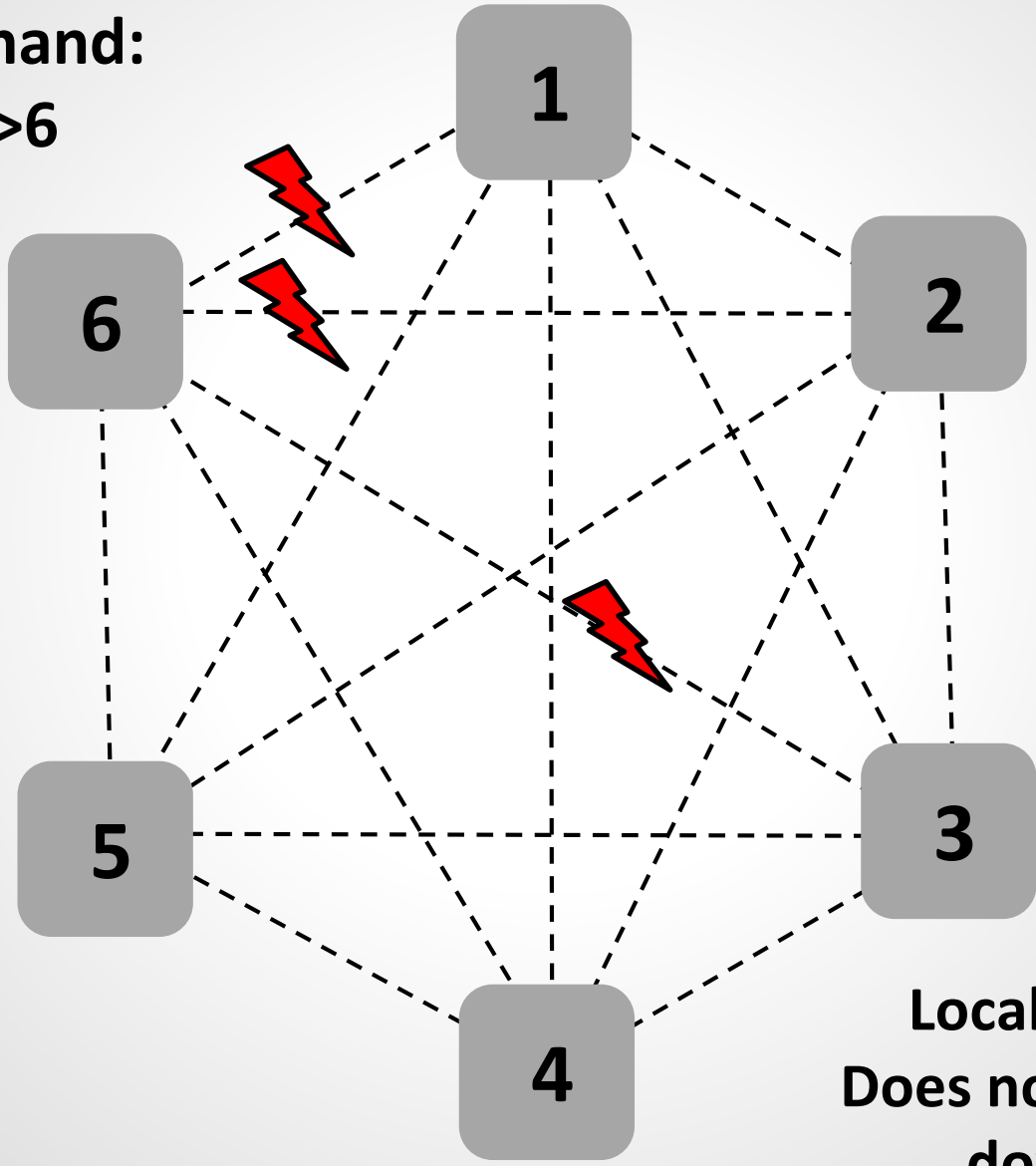
Local Fast Failover

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



Local Fast Failover

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

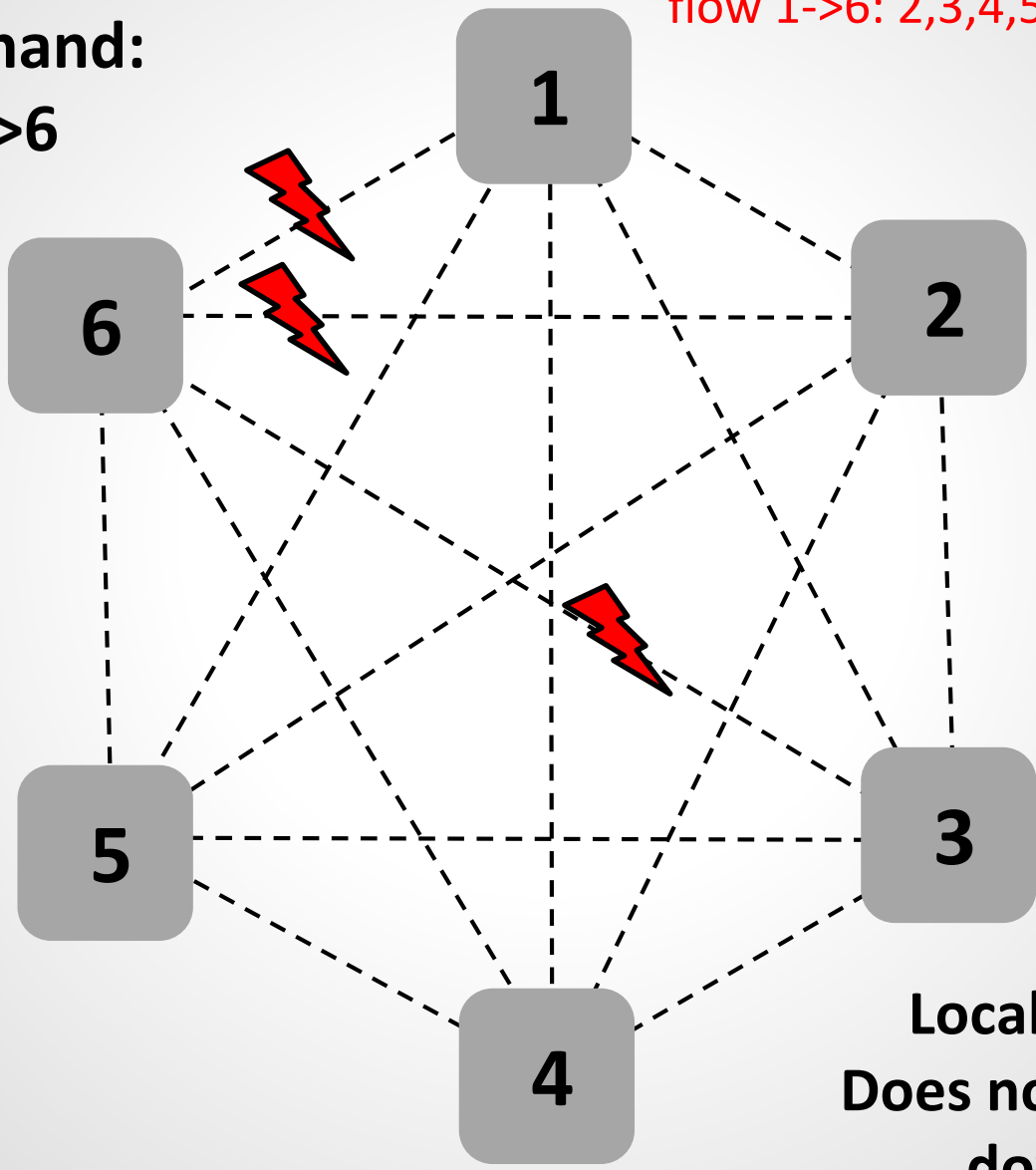


Local failover @1:
Does not know failures downstream!

Local Fast Failover

Failover table:
flow 1->6: 2,3,4,5,…

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



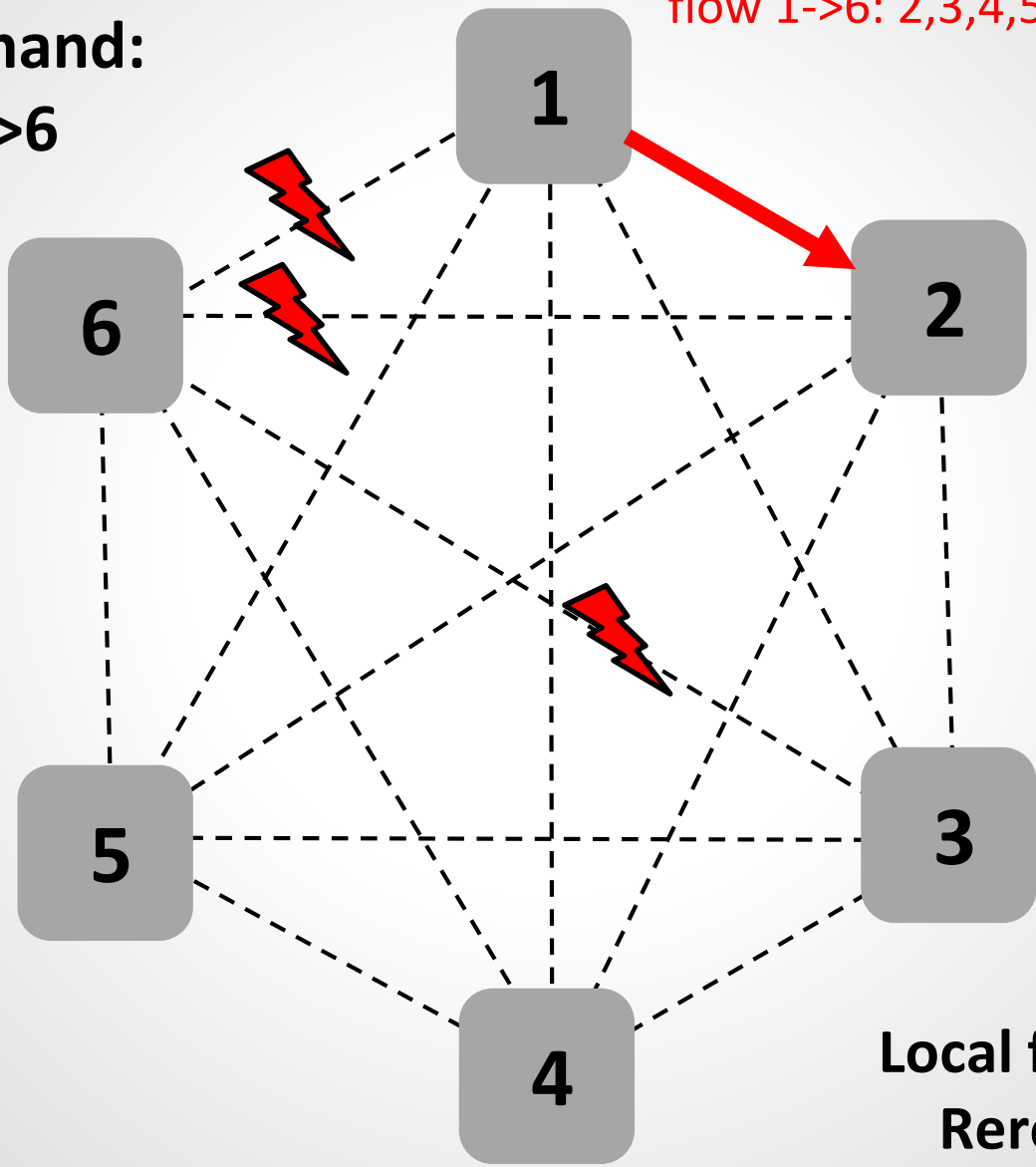
Local failover @1:
Does not know failures downstream!

Local Fast Failover

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

Failover table:
flow 1->6: 2,3,4,5,…

Failover table:
flow 1->6: 2,3,4,5,…

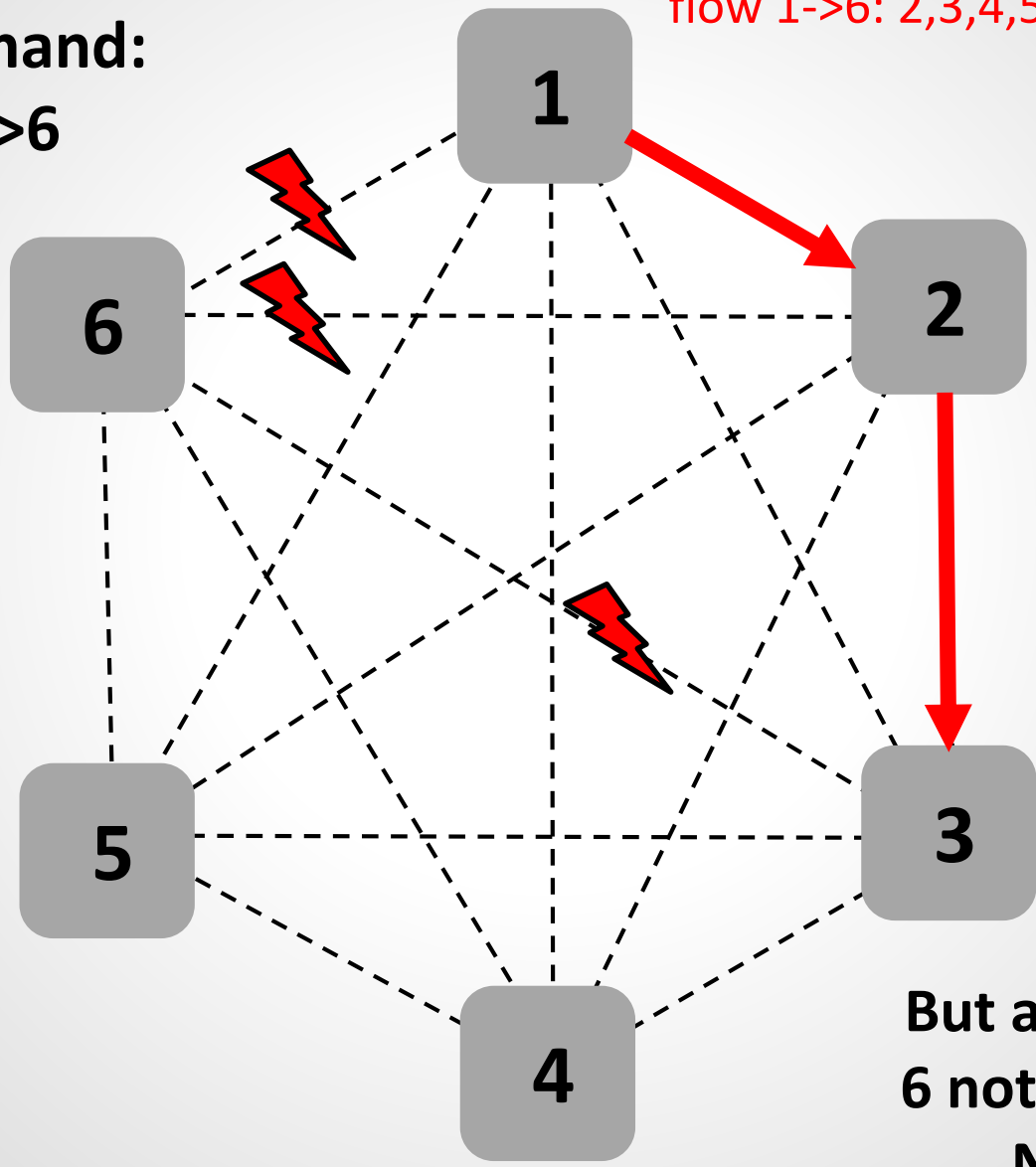


Local failover @1:
Reroute to 2!

Local Fast Failover

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

Failover table:
flow 1->6: 2,3,4,5,…



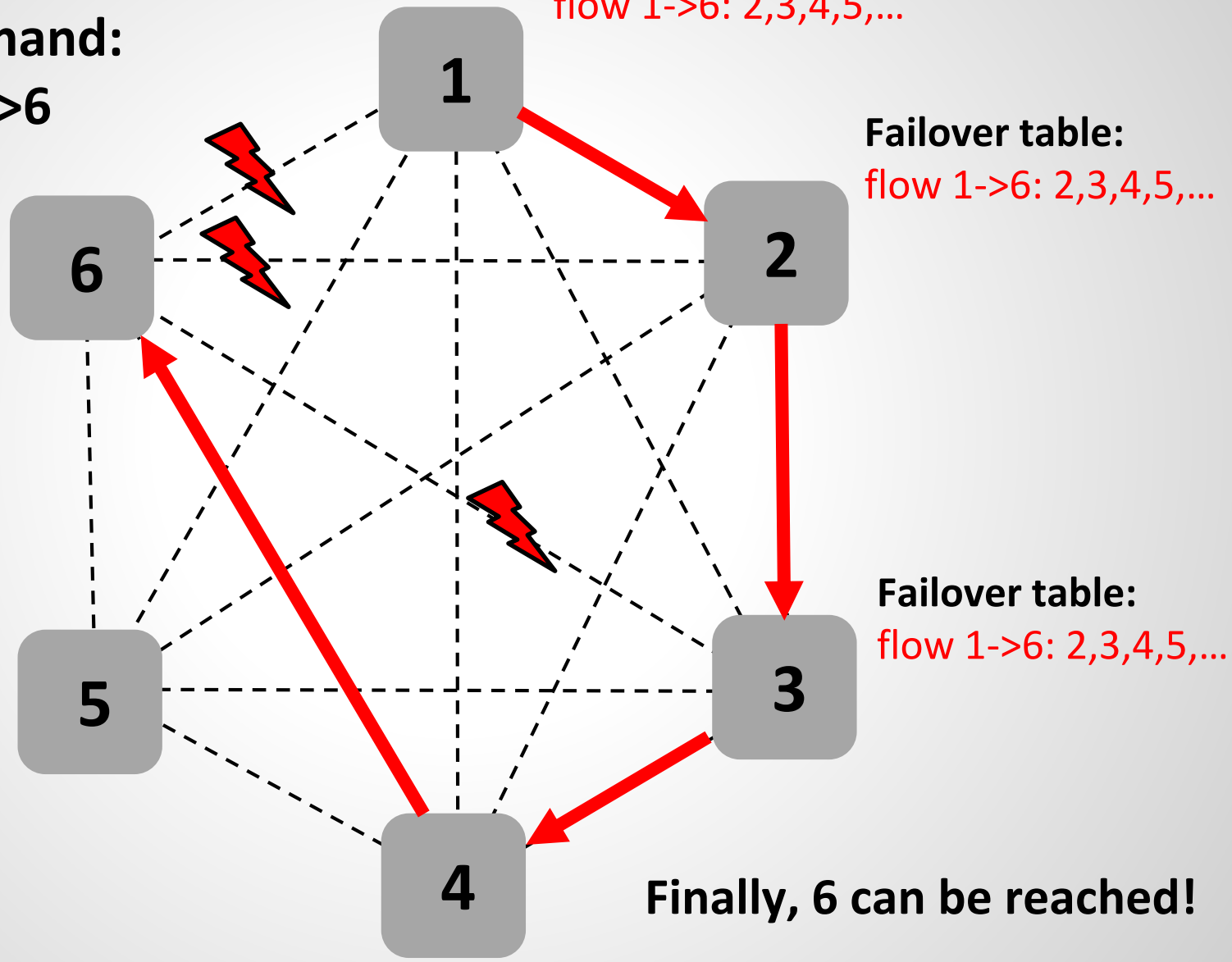
Failover table:
flow 1->6: 2,3,4,5,…

Failover table:
flow 1->6: 2,3,4,5,…

But also from 2:
6 not reachable.
Next: 3.

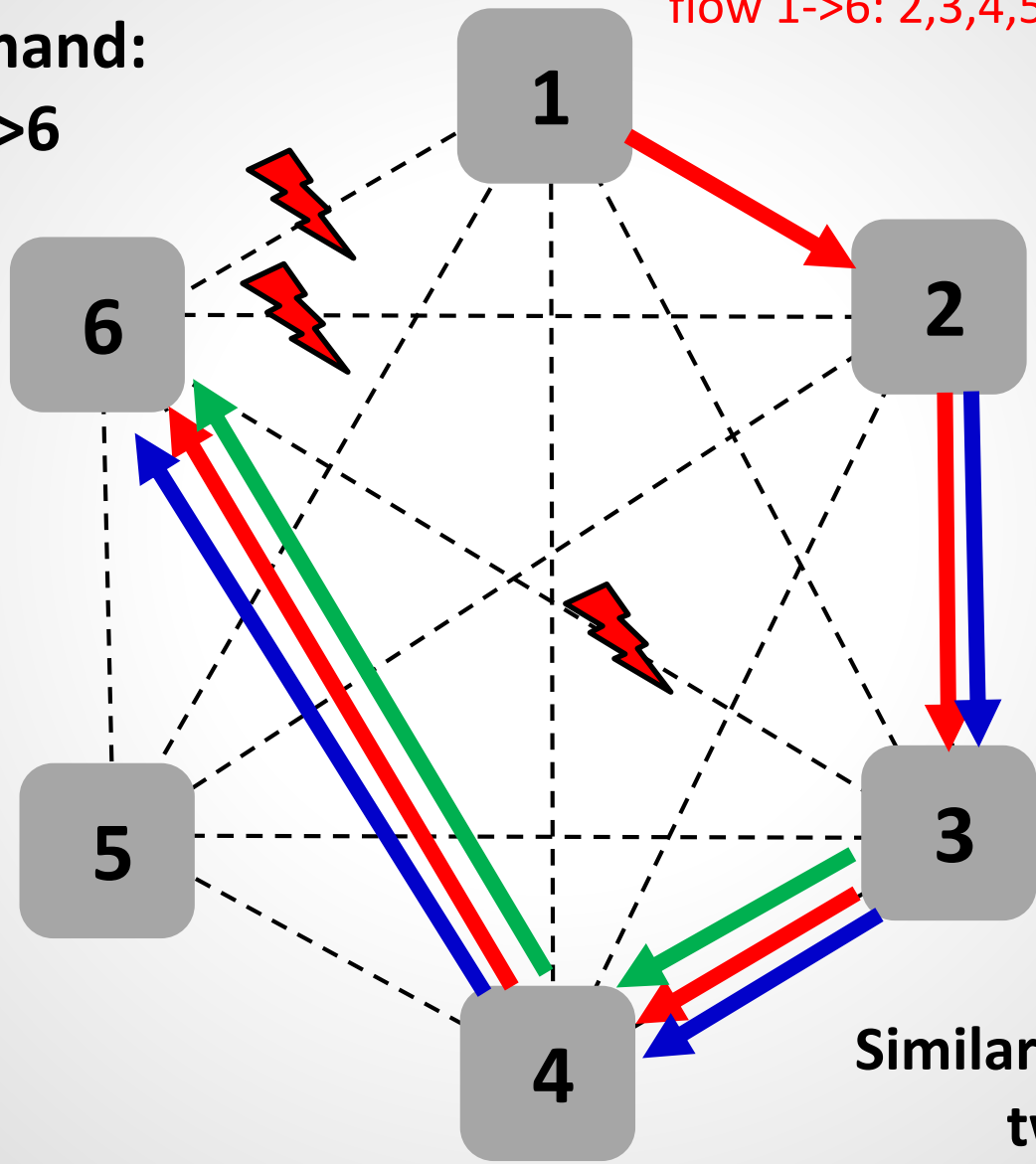
Local Fast Failover

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



Local Fast Failover

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



Failover table:
flow 1->6: 2,3,4,5,…

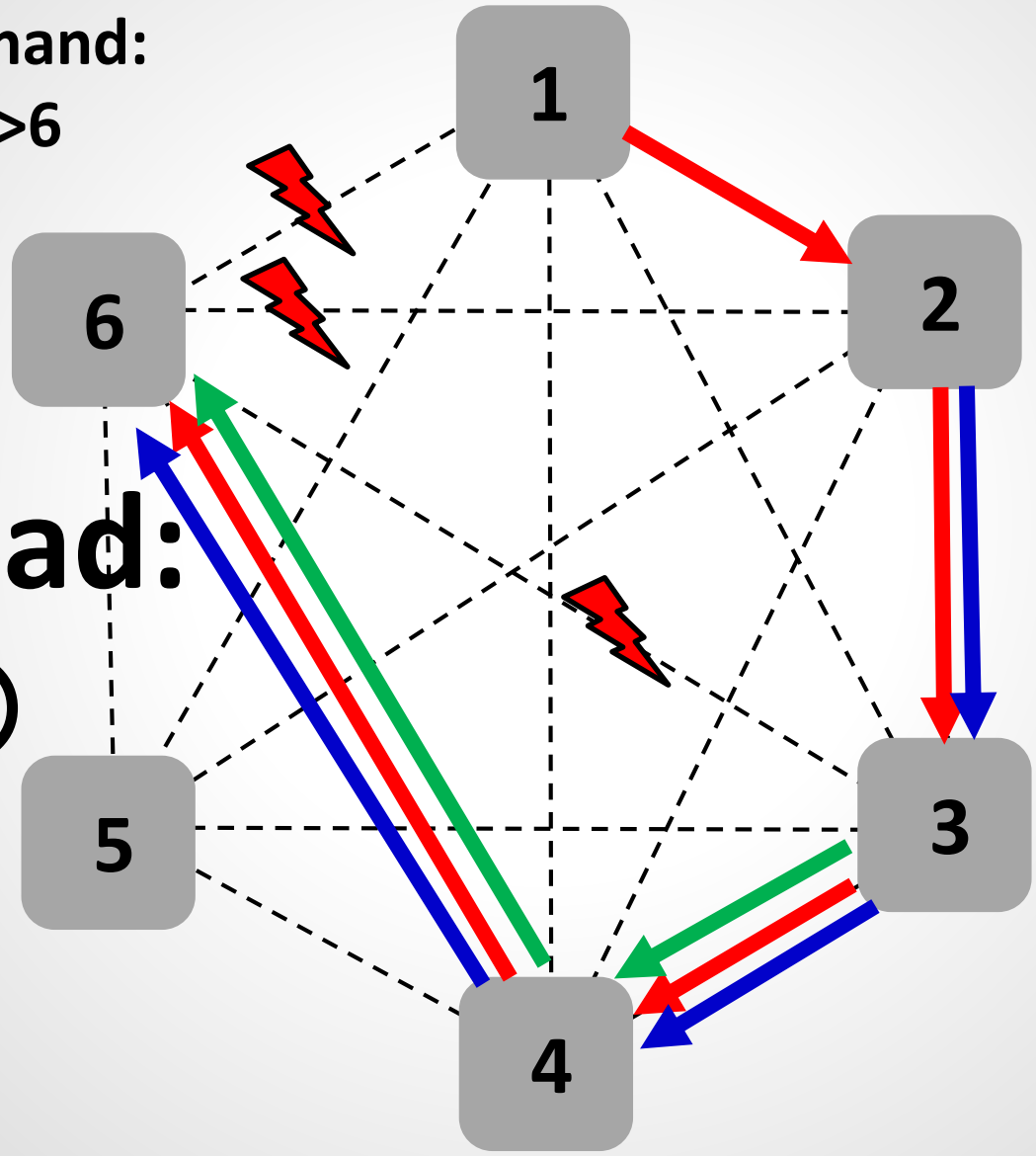
Failover table:
flow 1->6: 2,3,4,5,…
flow 2->6: 3,4,5,…

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

Similarly for the other two flows.

Local Fast Failover

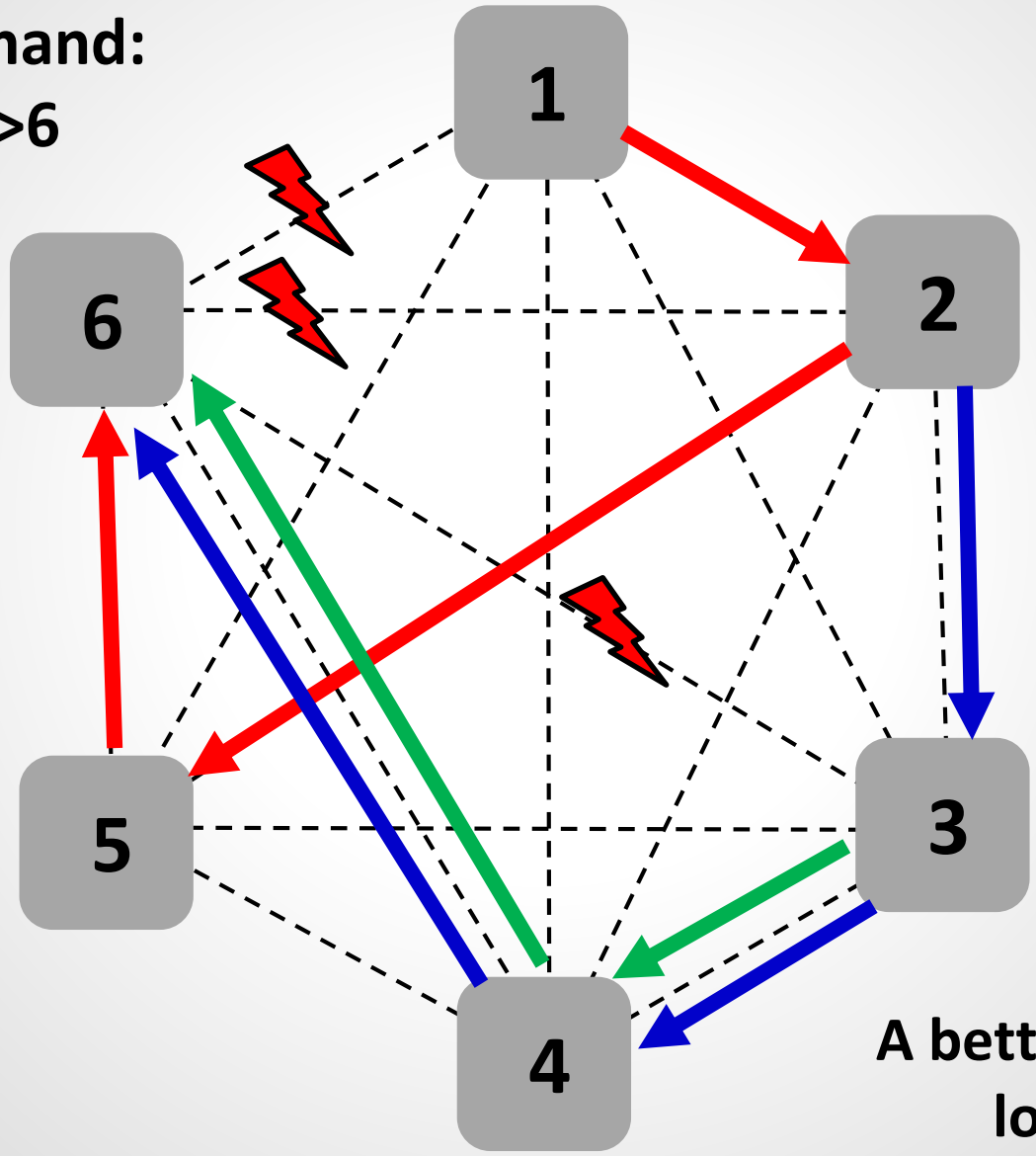
Traffic demand:
{1,2,3}->6



Max load:
3 ☹

Local Fast Failover

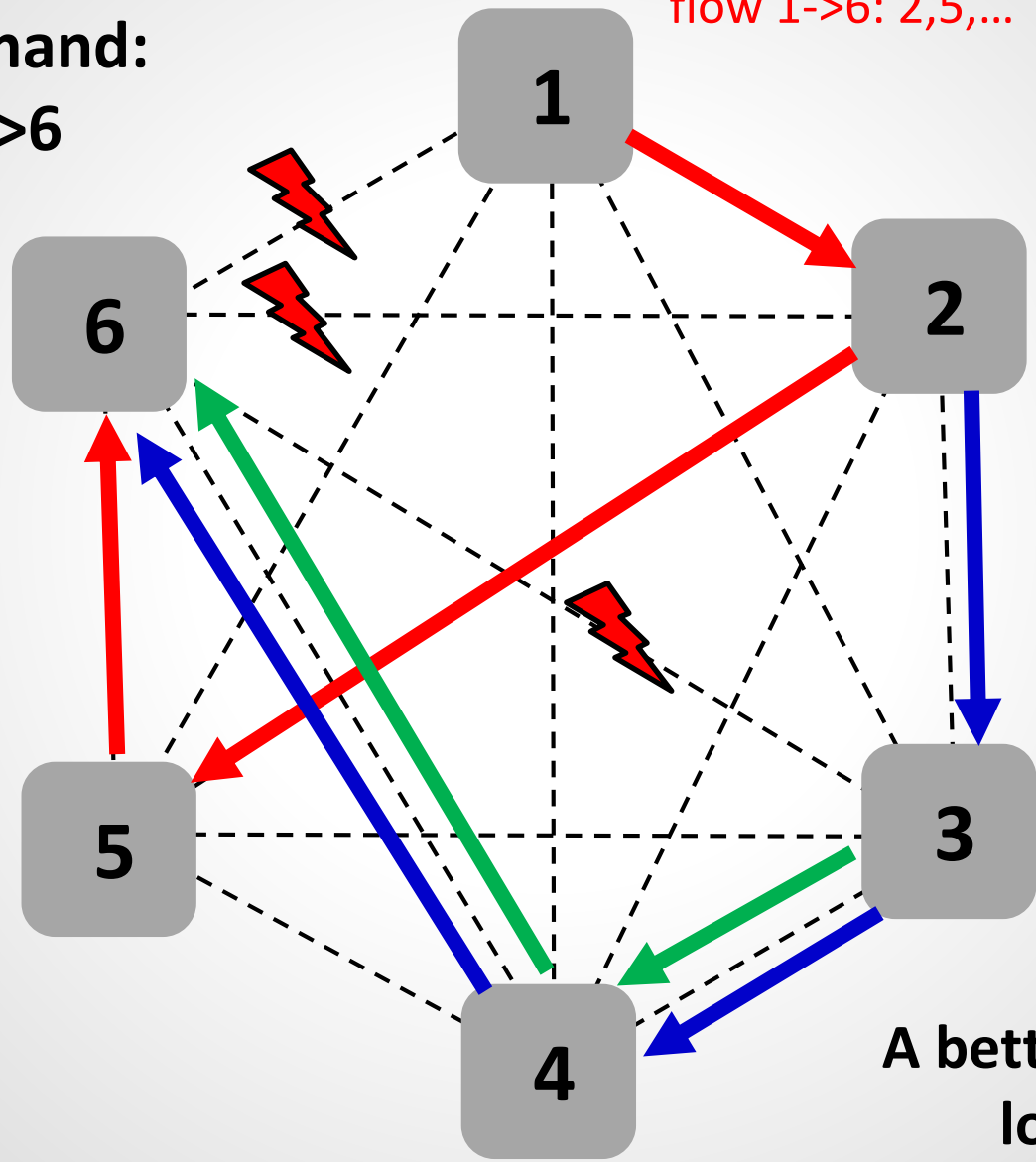
Traffic demand:
{1,2,3}->6



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 ☺

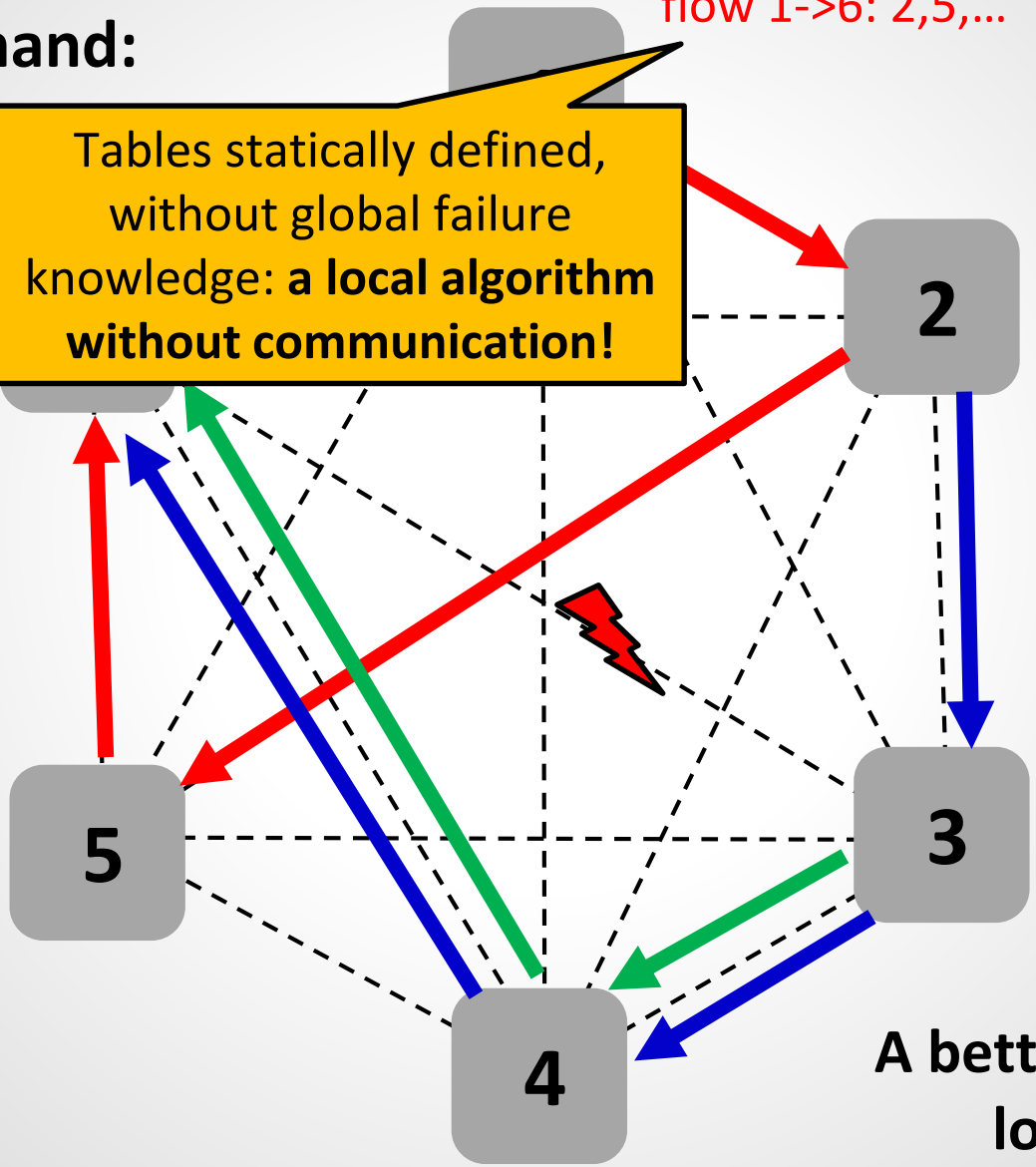
Local Fast Failover

Traffic demand:

{1,2,3}->

Failover table:
flow 1->6: 2,5,…

Tables statically defined, without global failure knowledge: **a local algorithm without communication!**



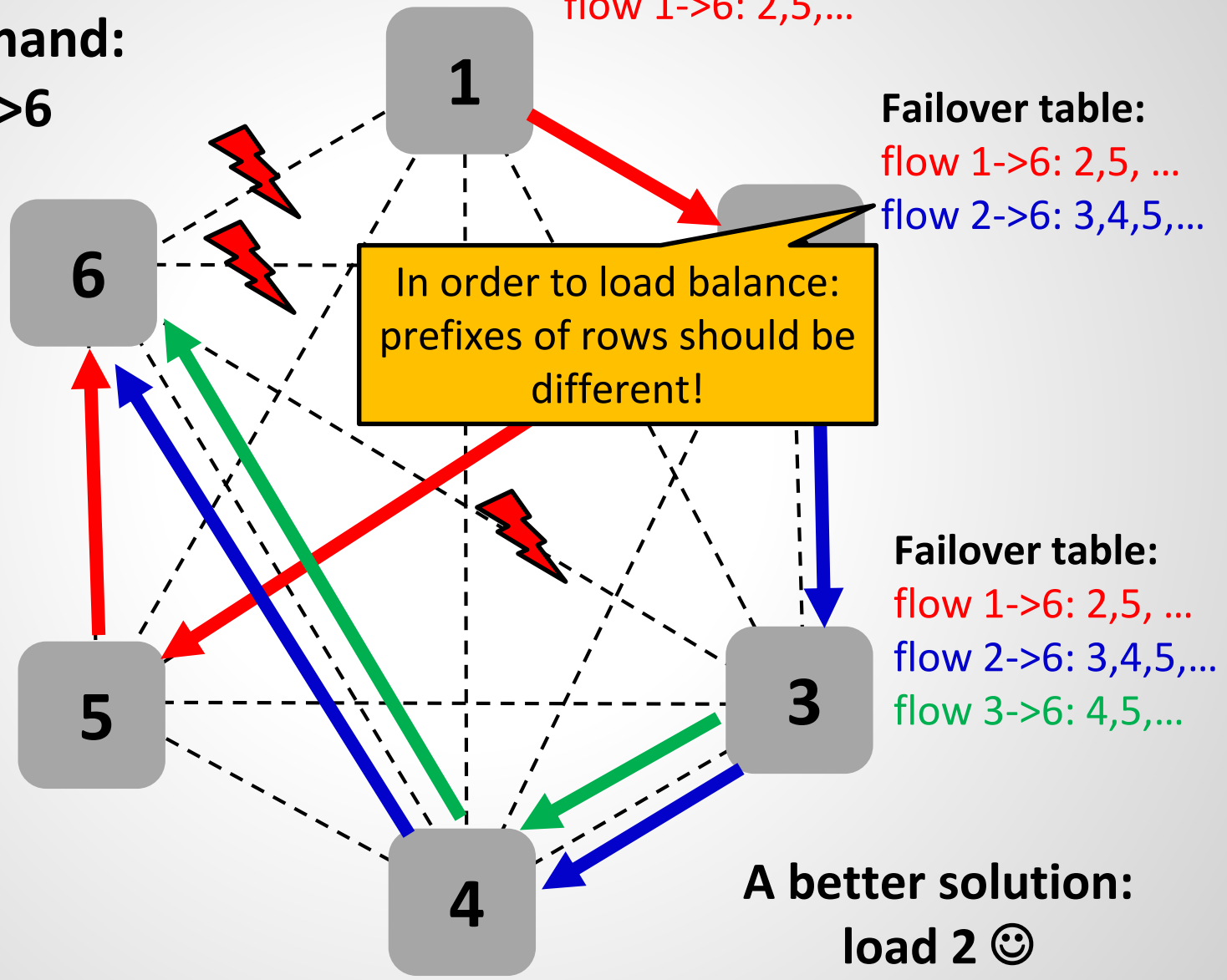
Failover table:
flow 1->6: 2,5, …
flow 2->6: 3,4,5,…

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:
{1,2,3}->6



Local Fast Failover

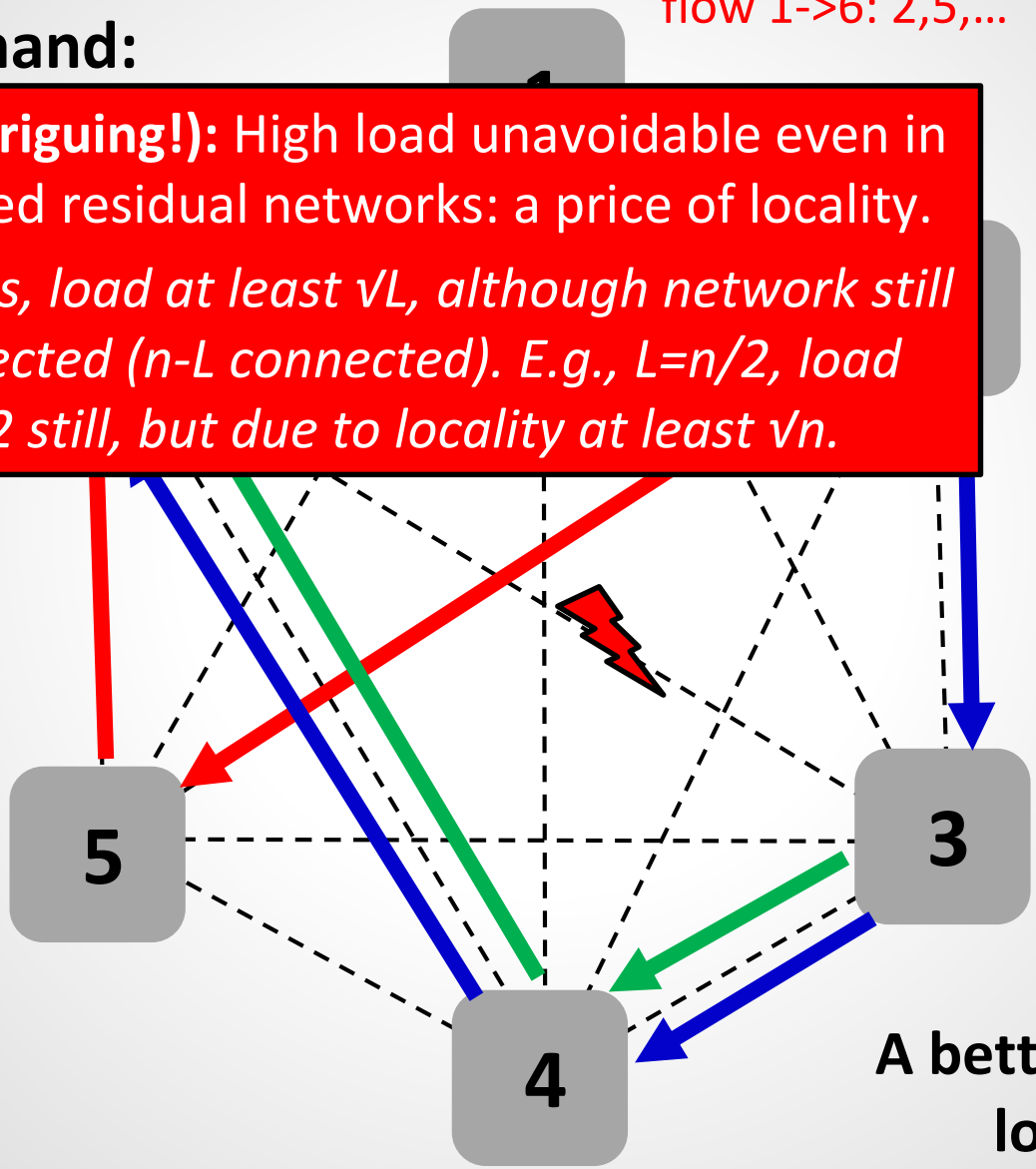
Failover table:
flow 1->6: 2,5,…

Traffic demand:

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

Given L failures, load at least √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.

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

Failover table:
flow 1->6: 2,5,…

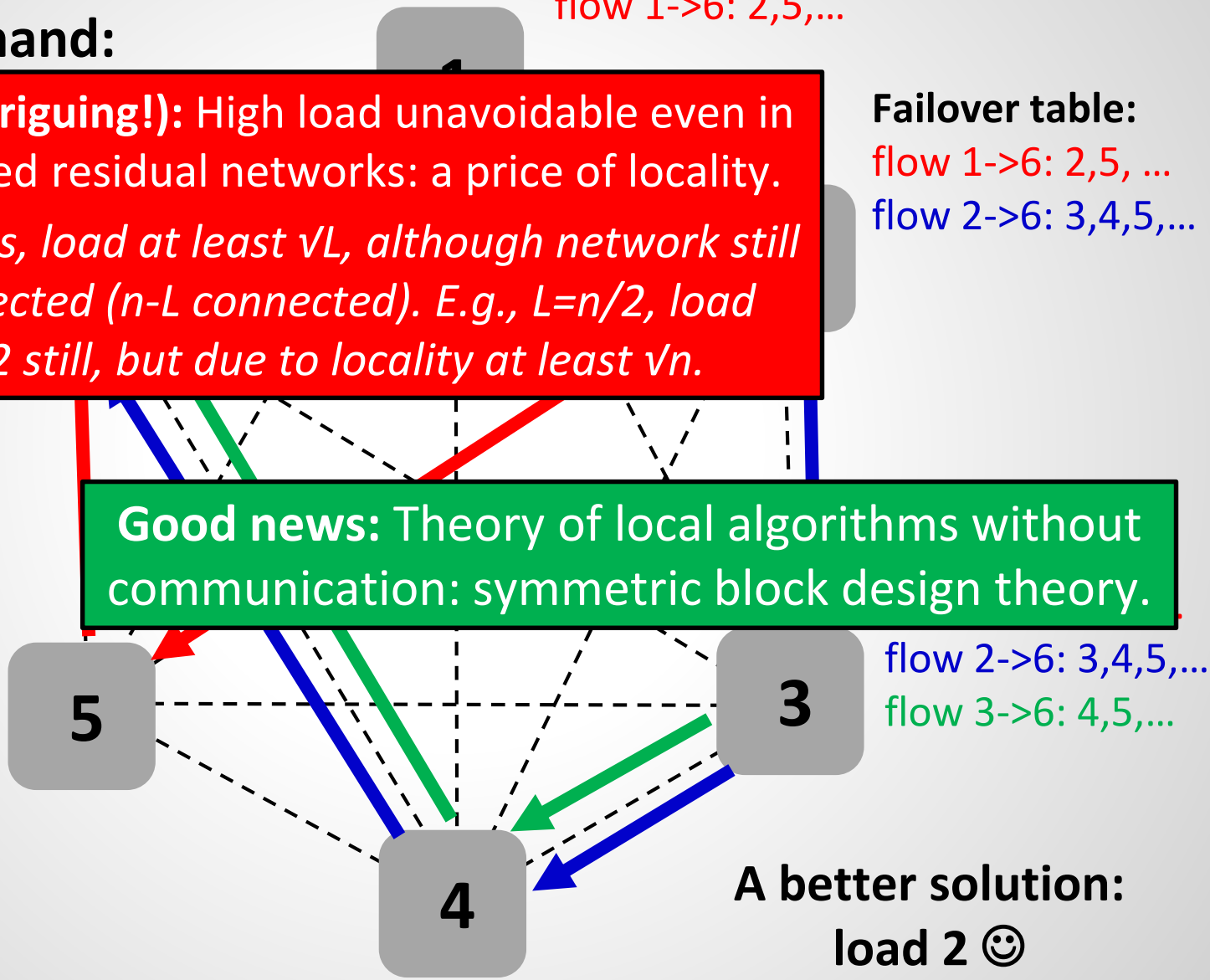
Traffic demand:

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

Given L failures, load at least √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.

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

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

Failover table:
flow 1->6: 2,5,…

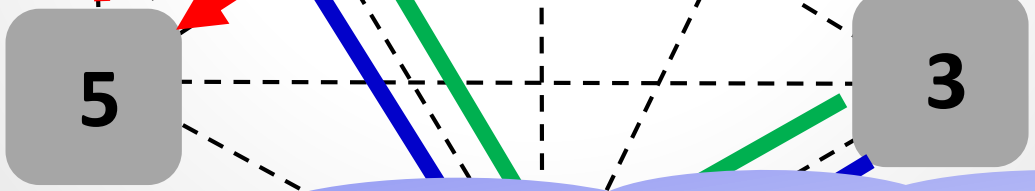
Traffic demand:

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

Given L failures, load at least √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.

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

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



flow 2->6: 3,4,5,…
flow 3->6: 4,5,…

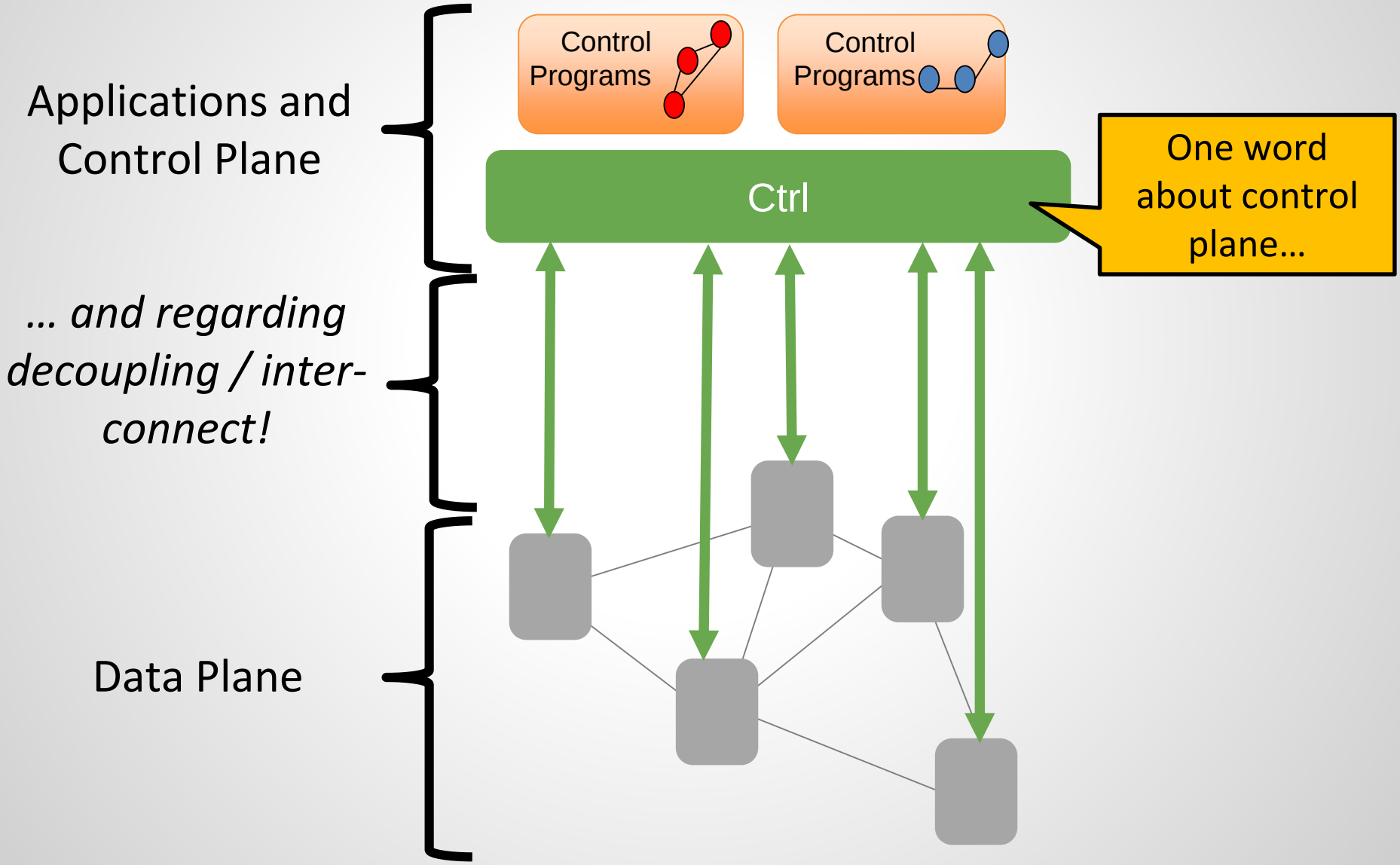
What about multihop networks?
See Chiesa et al.

Further Reading

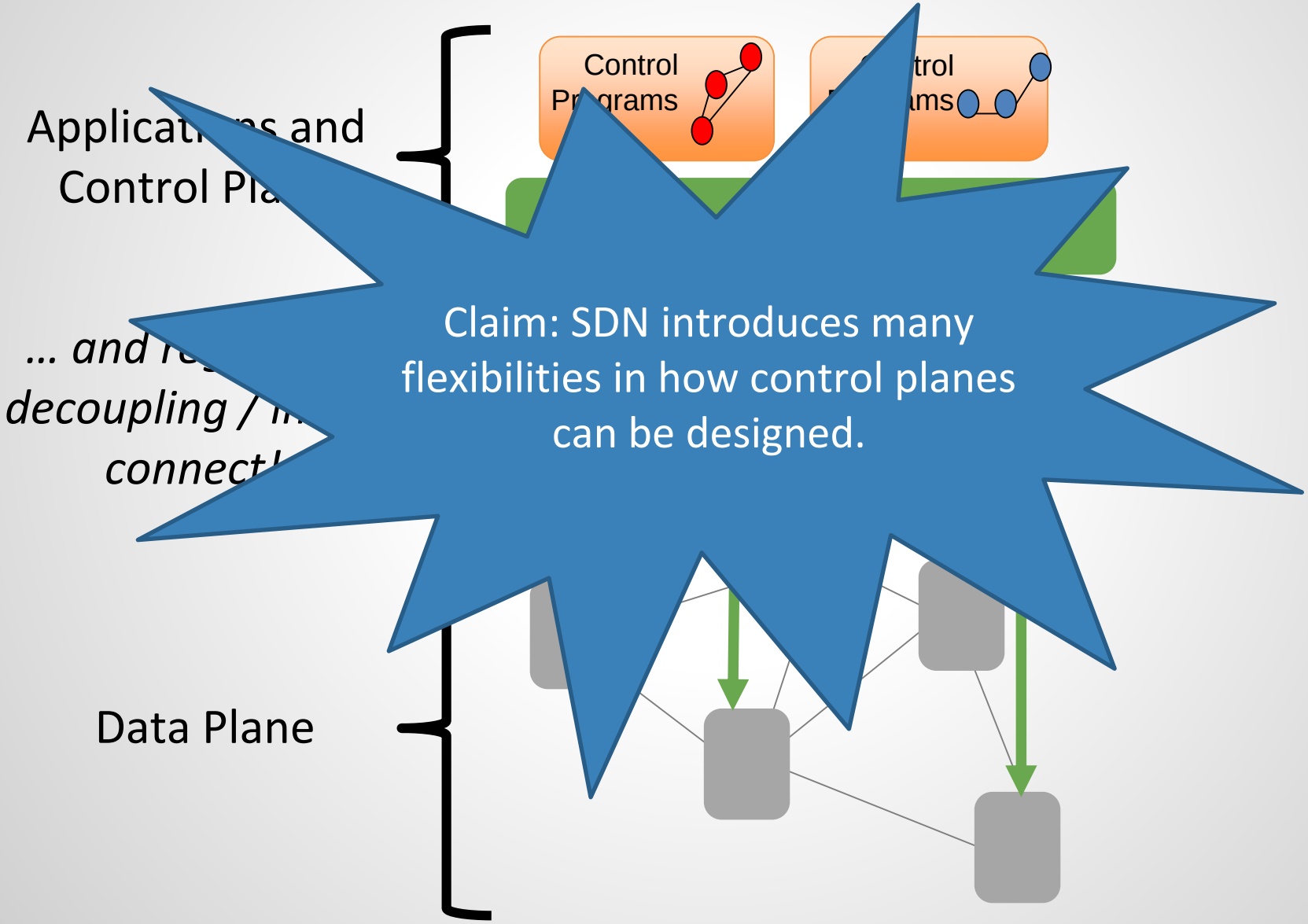
Load-Optimal Local Fast Rerouting for Dependable Networks
Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
47th IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**), Denver, Colorado, USA, June 2017.

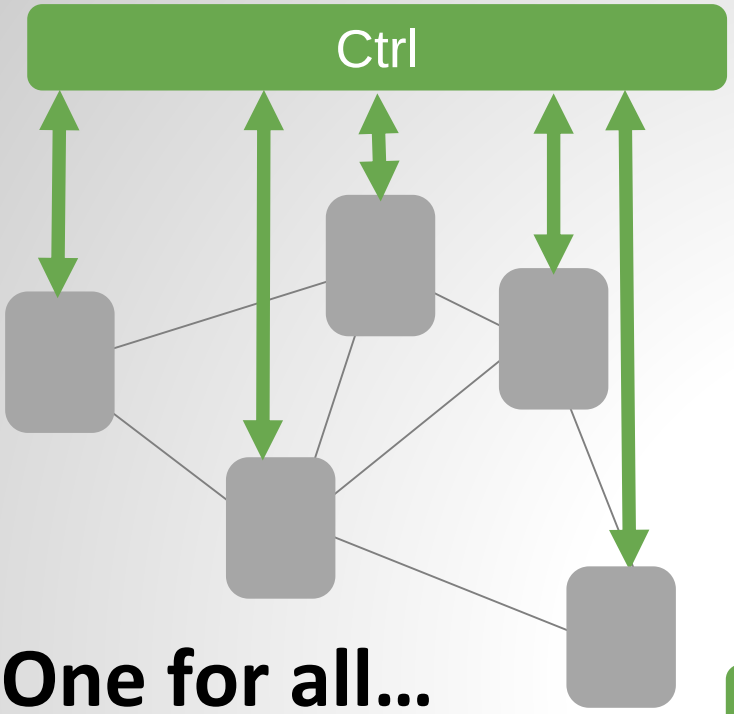
How (Not) to Shoot in Your Foot with SDN Local Fast Failover: A Load-Connectivity Tradeoff
Michael Borokhovich and Stefan Schmid.
17th International Conference on Principles of Distributed Systems (**OPODIS**), Nice, France, Springer LNCS, December 2013.

Algorithmic Problems in SDNs



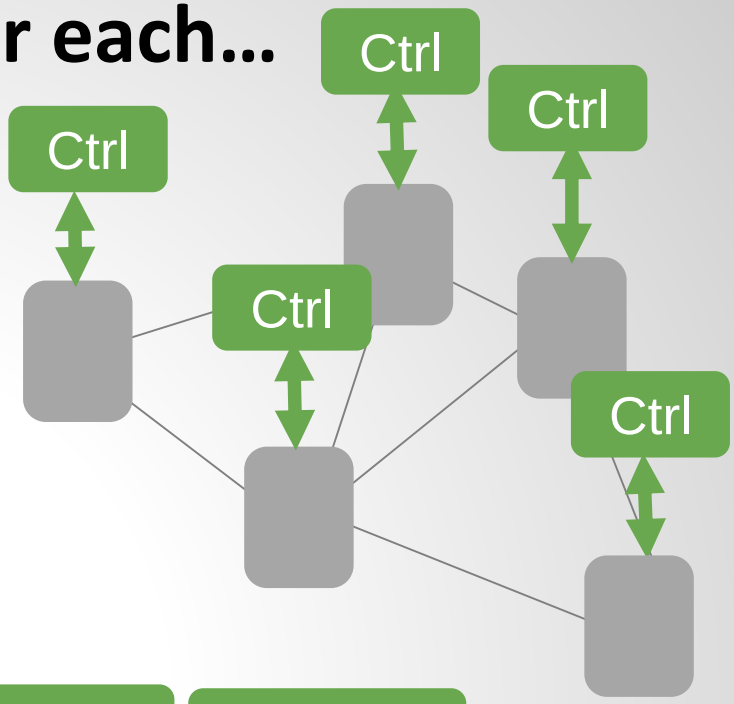
Algorithmic Problems in SDNs



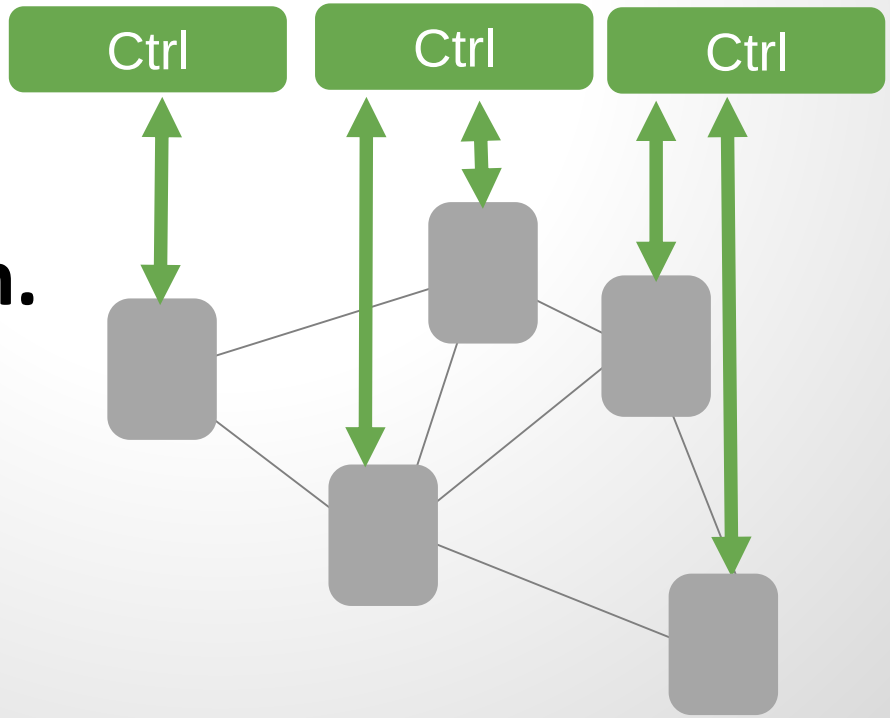


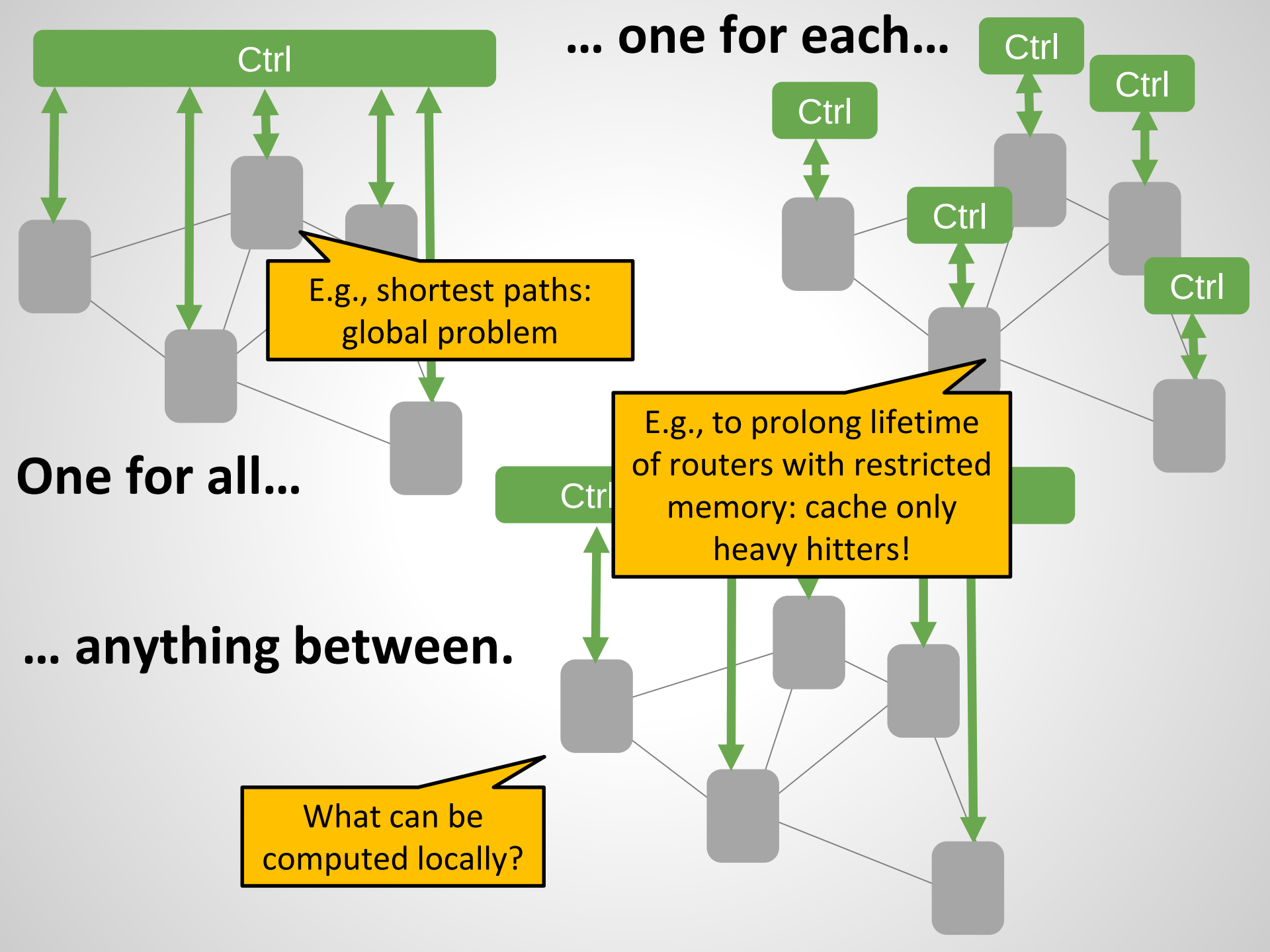
One for all…

… one for each…



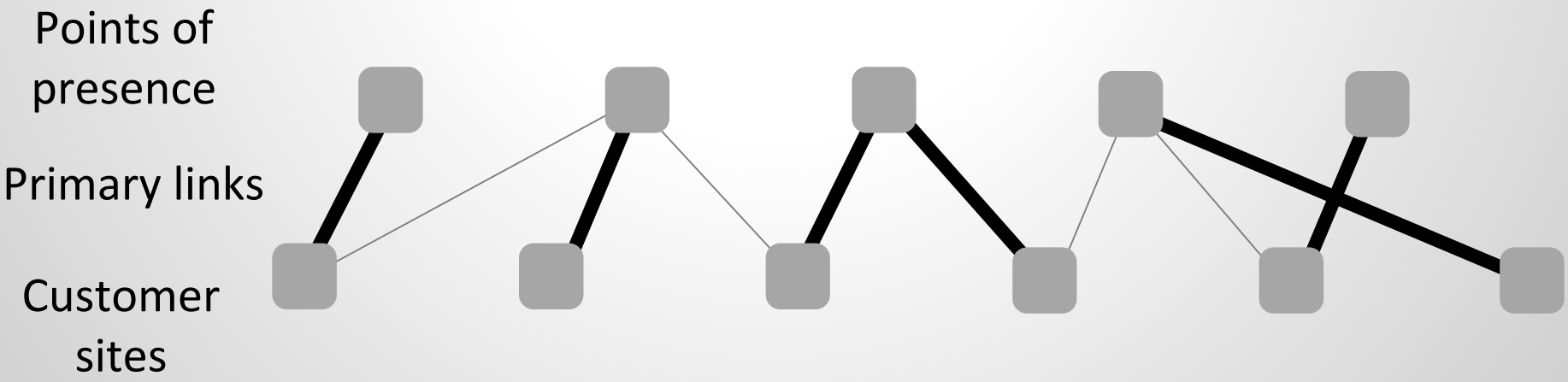
… anything between.




Ctrl
… one for each…
Ctrl
Ctrl
Ctrl
Ctrl
Ctrl
Ctrl
E.g., shortest paths: global problem
One for all…
E.g., to prolong lifetime of routers with restricted memory: cache only heavy hitters!
… anything between.
What can be computed locally?

Challenge: Right Level of Locality?

- Some insights from distributed computing are handy: but come in a new light!
- But finding right level of locality is non-trivial: tradeoff between inter-controller communication and quality of solution
- E.g., in load balancing



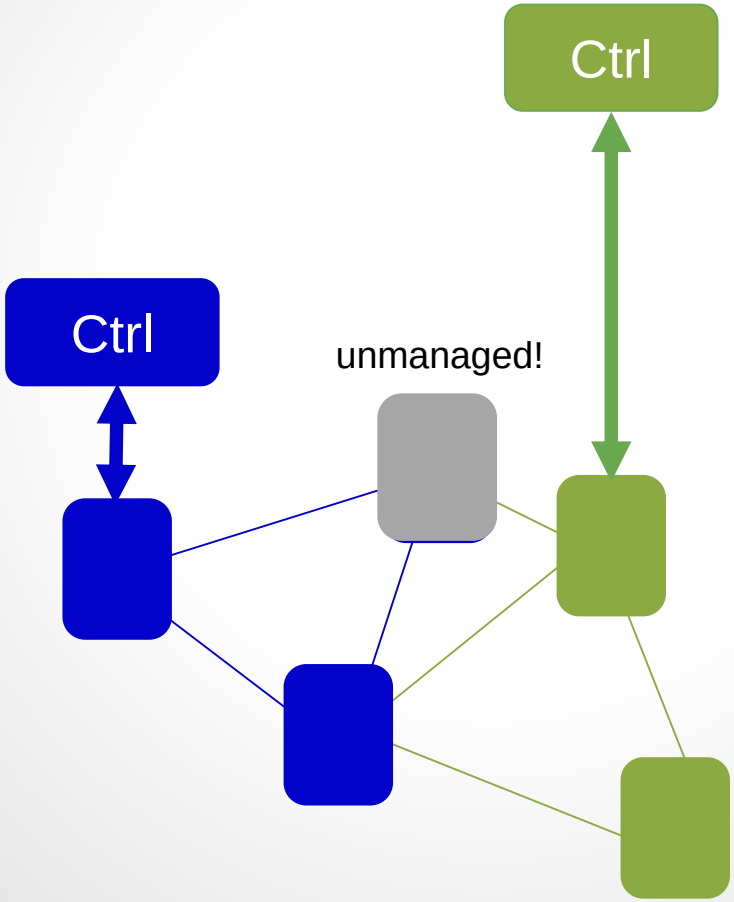
Further Reading

Exploiting Locality in Distributed SDN Control
Stefan Schmid and Jukka Suomela.
ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (**HotSDN**), Hong Kong, China, August 2013.

A Distributed and Robust SDN Control Plane for Transactional Network Updates
Marco Canini, Petr Kuznetsov, Dan Levin, and Stefan Schmid.
34th IEEE Conference on Computer Communications (**INFOCOM**), Hong Kong, April 2015.

Challenge: Right Level of Locality?

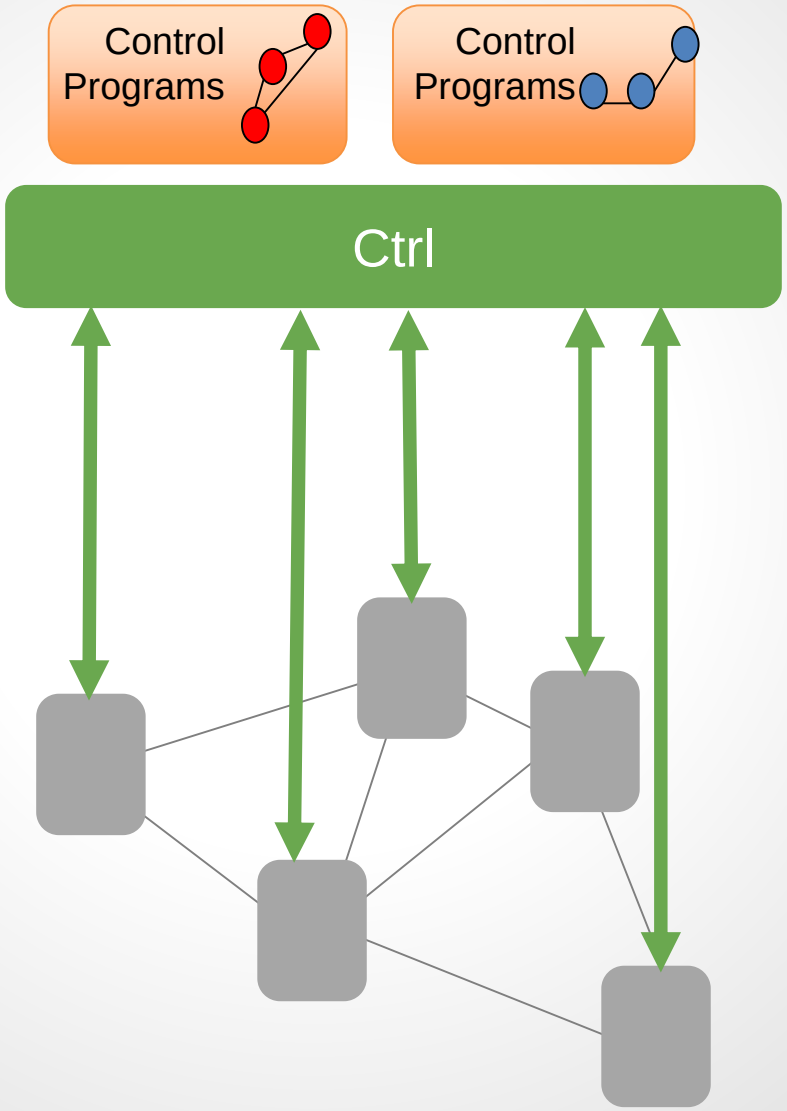
- Also: how to manage switches inband?



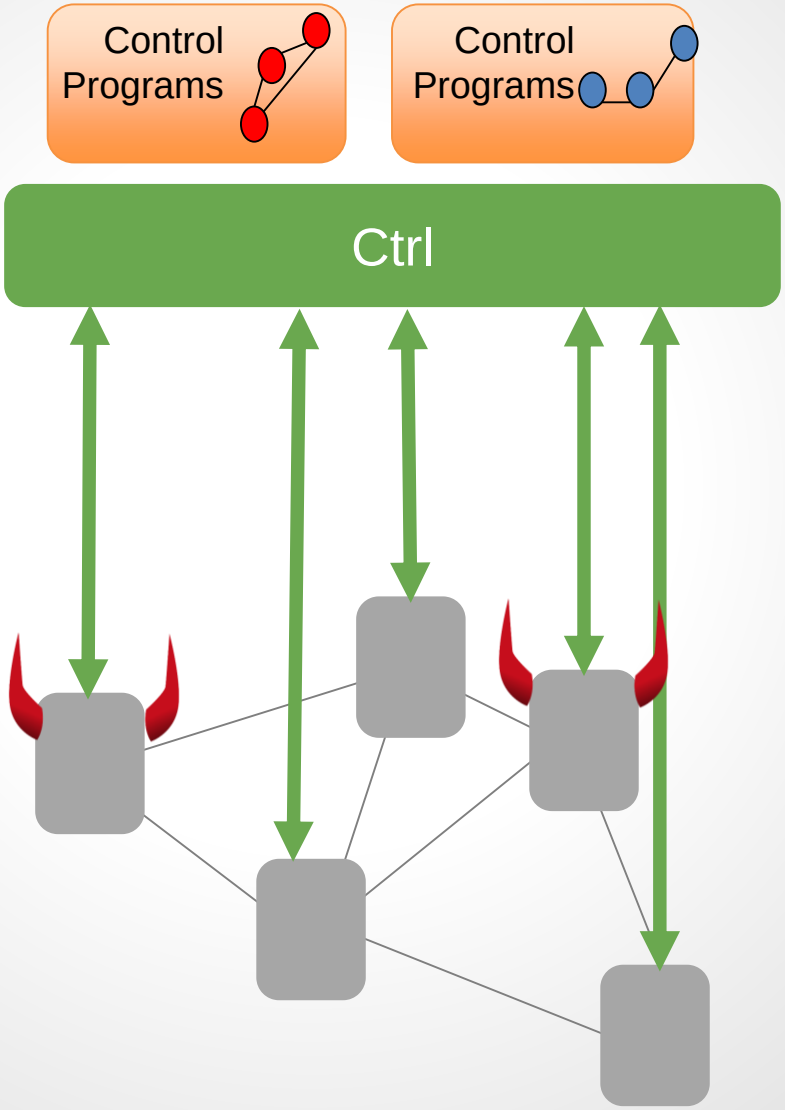
Further Reading

Medieval: Towards A Self-Stabilizing, Plug & Play, In-Band SDN Control Network (Demo Paper)
Liron Schiff, Stefan Schmid, and Marco Canini.
ACM Sigcomm Symposium on SDN Research (**SOSR**), Santa Clara, California, USA, June 2015.

Let’s talk about security!

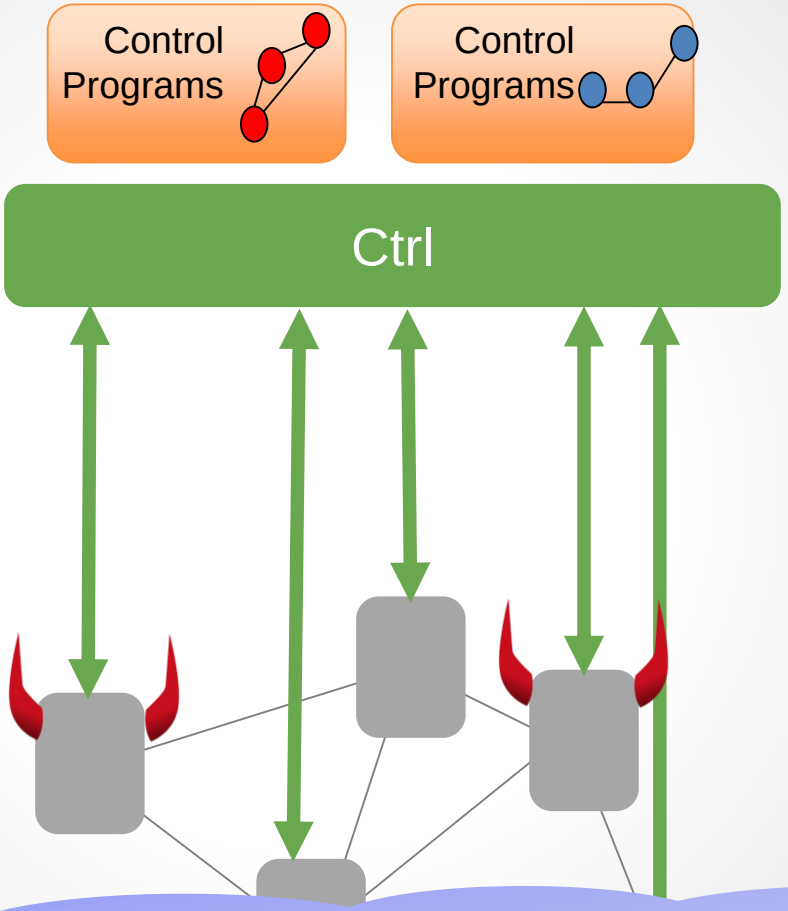


Let's talk about security!



In particular: malicious switches (less studied).

Let's talk about security!



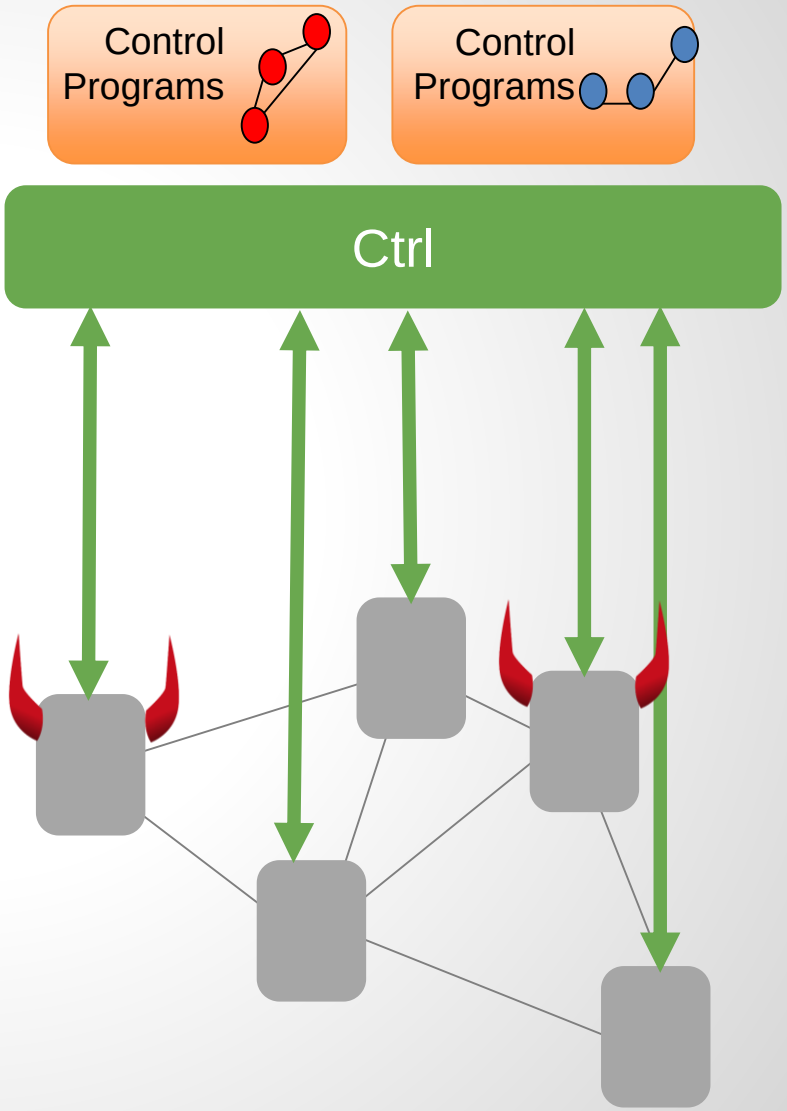
In particular: malicious switches (less studied).

Note: Governments etc. don't have resources to build their own trusted hardware.

Let's talk about security!

The case for insecure data planes: many incidents

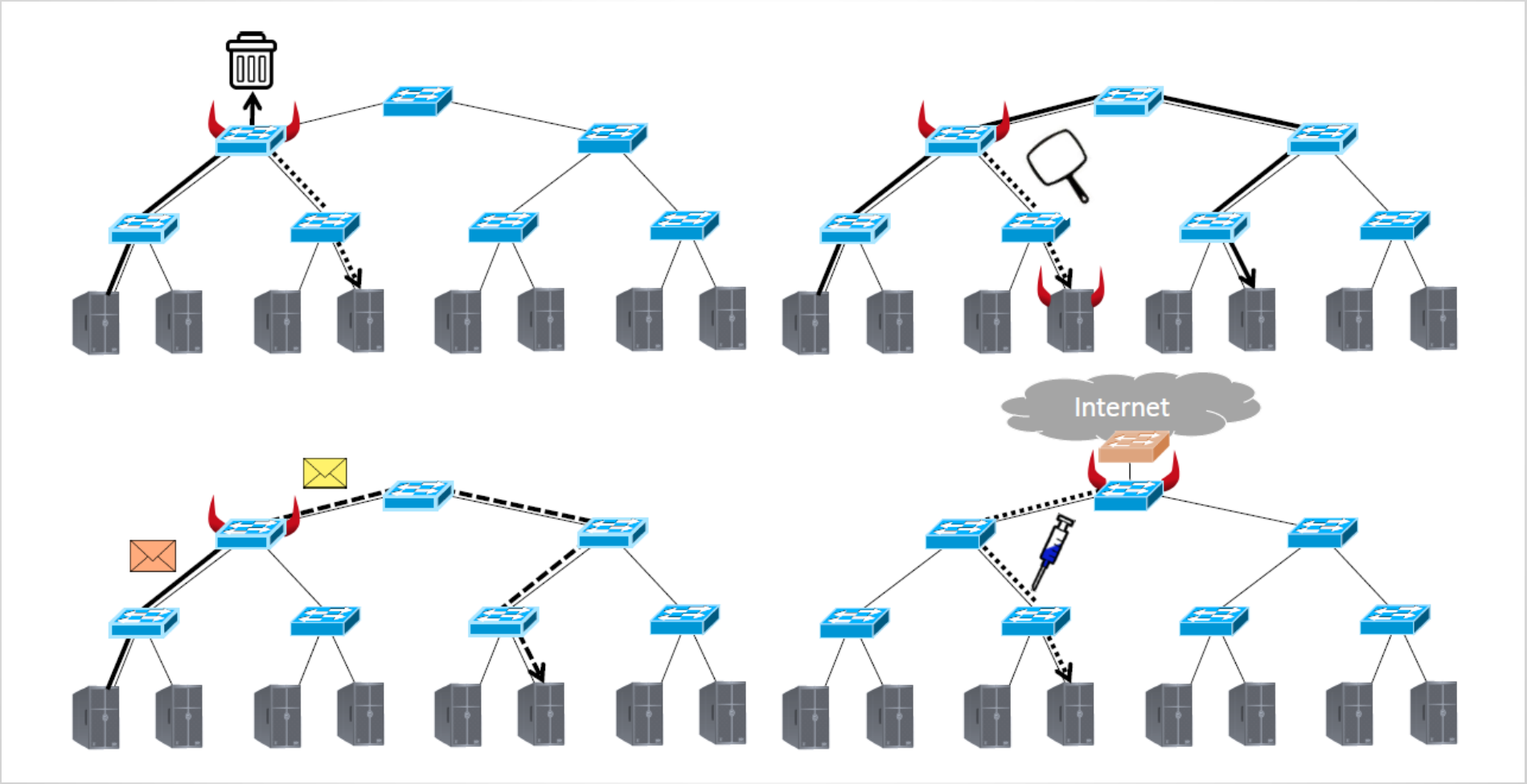
- ❑ Attackers have compromised routers
- ❑ Compromised routers are traded underground
- ❑ Vendors have left backdoors open
- ❑ National security agencies can bug network equipment
- ❑ …



What a malicious switch could do:

1 drop/reroute/exfiltrate

2 mirror



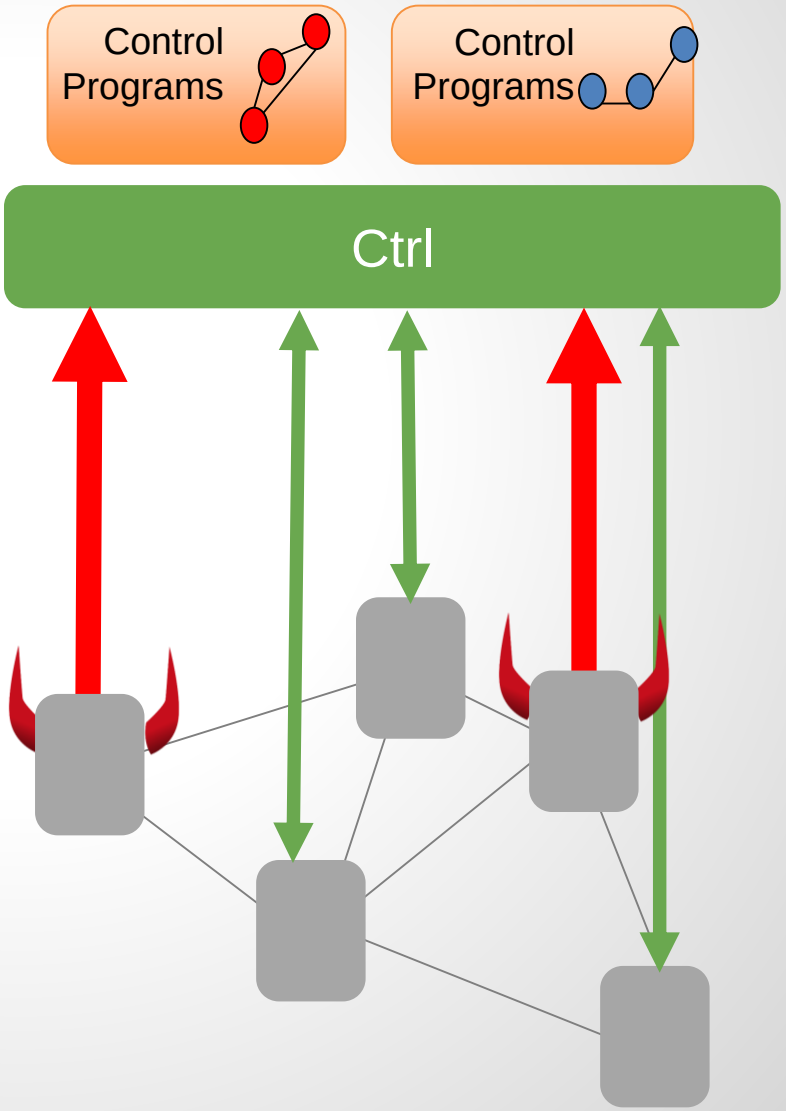
3 modify

4 inject

More and New Attacks in SDN

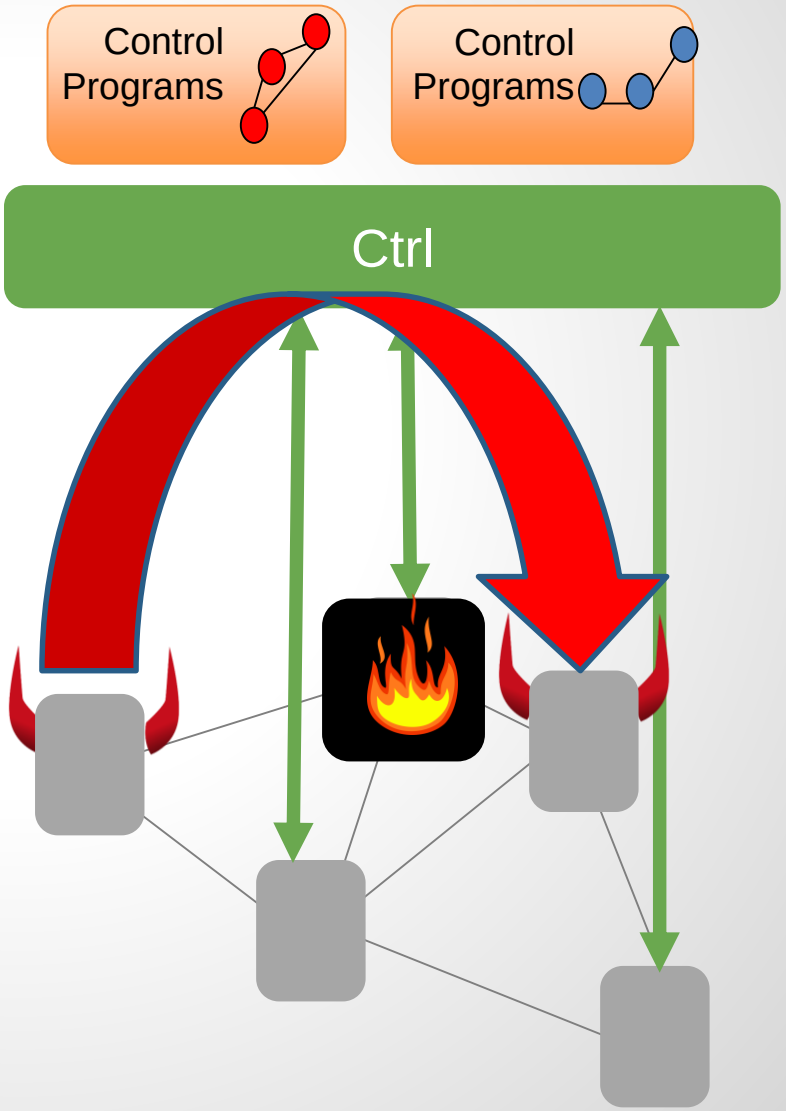
New attack vector:

- ❑ DoS on controller
- ❑ Harms availability
- ❑ E.g., force other switches into default behavior



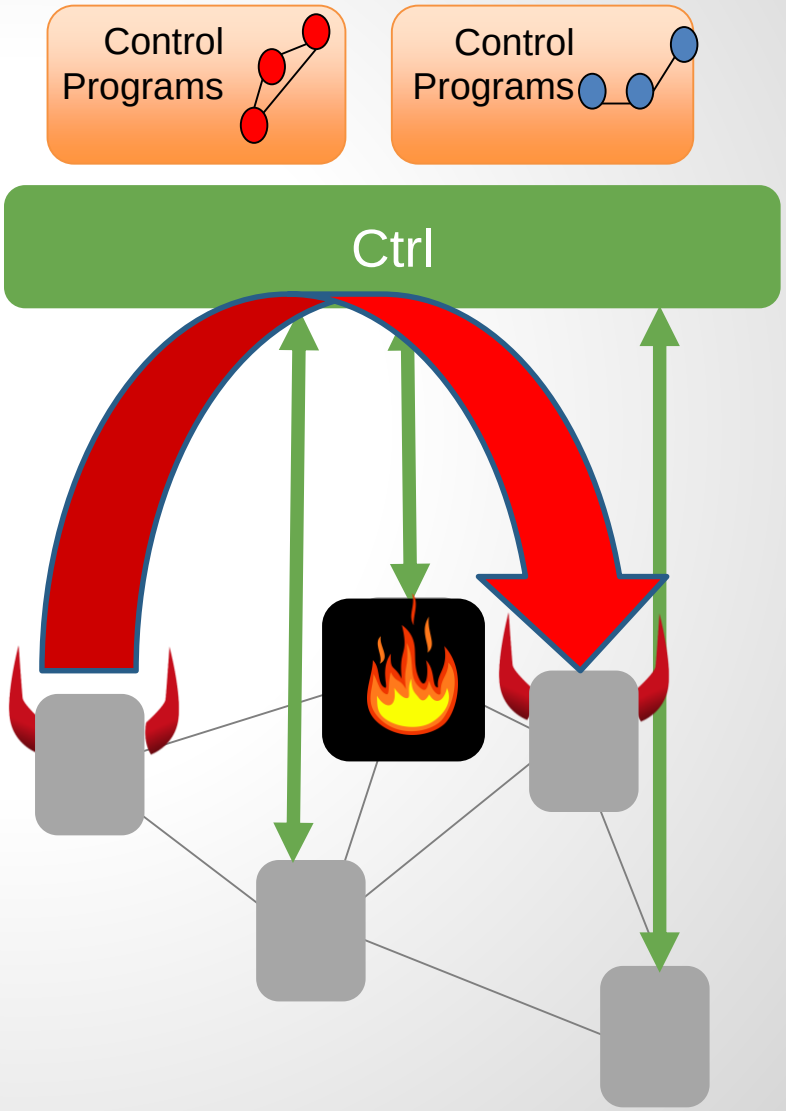
Another New SDN Attack: Teleportation

- Idea: exploit controller to communicate information: «Teleportation»



Another New SDN Attack: Teleportation

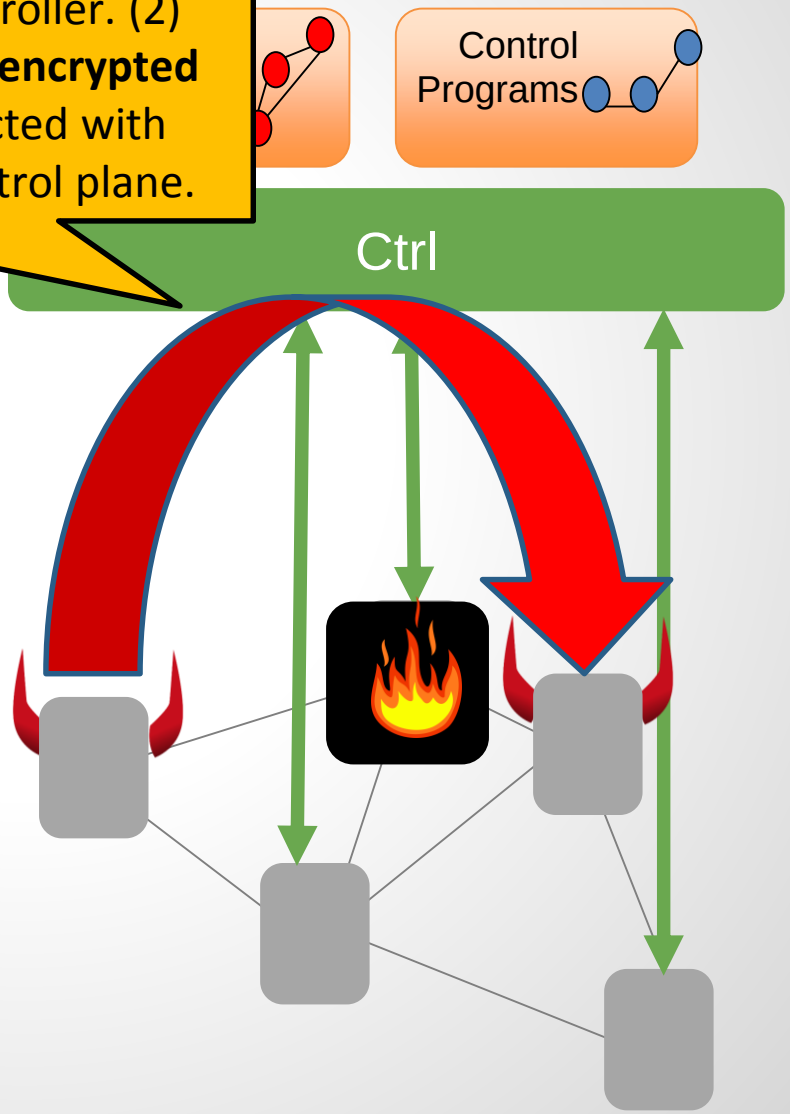
- Idea: exploit controller to communicate information: «Teleportation»
- Controller reacts to switch events (packet-ins) by sending flowmods/packet-outs/… etc.: can be exploited to transmit information
- E.g., in MAC learning: src MAC 0xBADDAD
- Can also modulate information implicitly (e.g., frequency of packetins)
- E.g.: covert communication, bypass firewall, coordinate attack



Another New SDN Attack: Teleportation

Difficult to detect: (1) The teleported information follows the **normal traffic pattern** of control communication, indirectly between any switch and the controller. (2) Teleportation channel is inside the typically **encrypted OpenFlow** channel. Cannot easily be detected with modern IDS, even if they operate in the control plane.

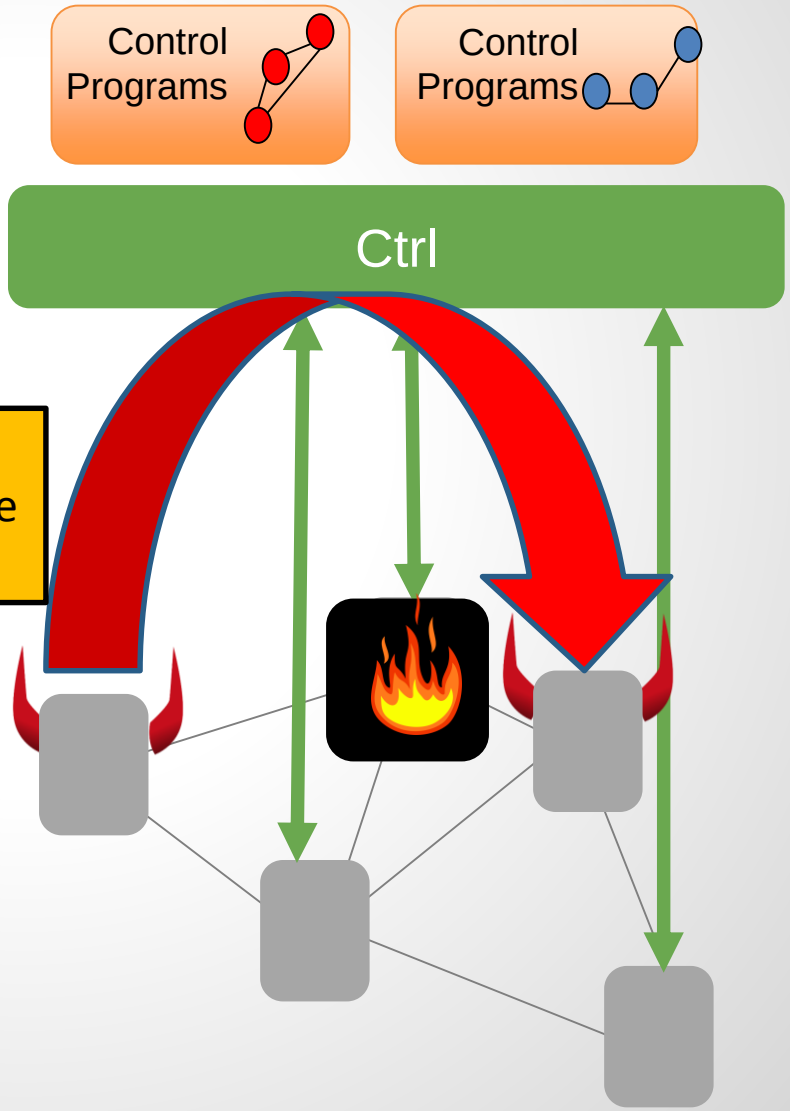
- events (packet-ins) by sending flowmods/packet-outs/… etc.: can be exploited to transmit information
- ❑ E.g., in MAC learning: src MAC 0xBADDAD
- ❑ Can also modulate information implicitly (e.g., frequency of packetins)
- ❑ E.g.: covert communication, bypass firewall, coordinate attack



Another New SDN Attack: Teleportation

- Idea: exploit controller to communicate information: «Teleportation»
- Controller reacts to switch events (packet-ins) by sending flowmods/packet-outs/… etc.: can be exploited to transmit informatic
- E.g., in MA 0xBADDAD
- Can also modulate information implicitly (e.g., frequency of packetins)
- E.g.: covert communication, bypass firewall, coordinate attack

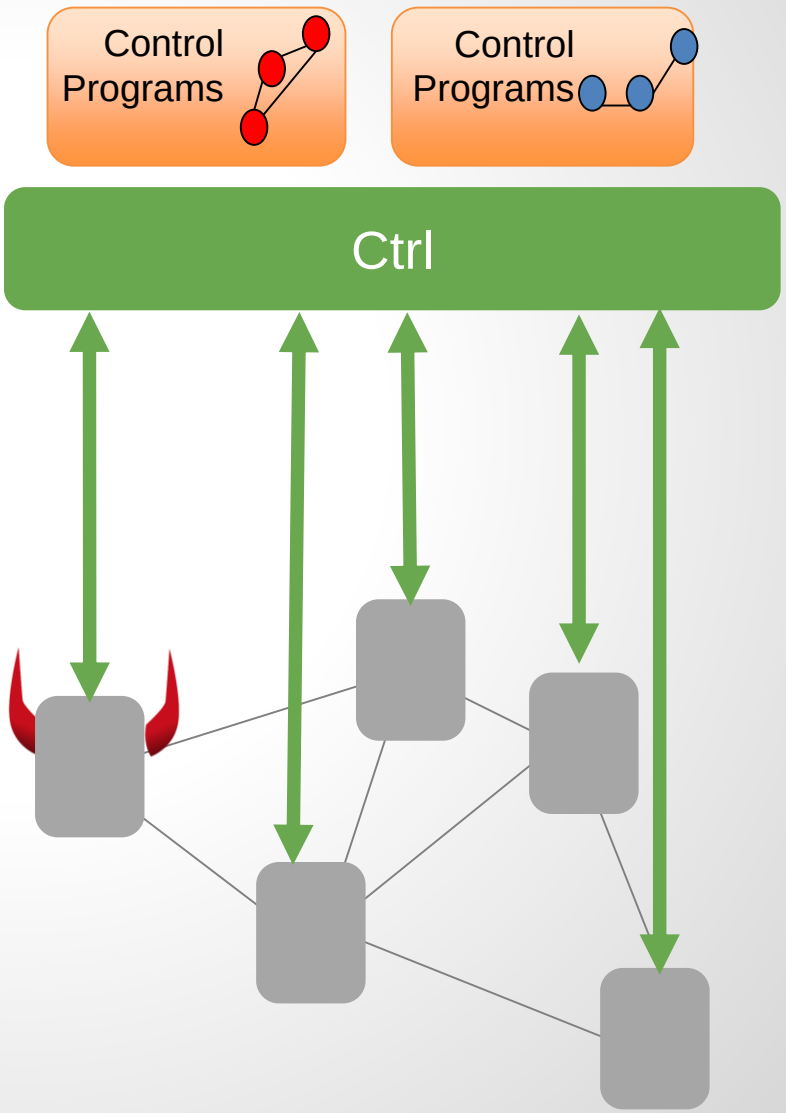
E.g., 2 switches try to use the same DPID, exploit pave path technique, etc.



Another Front: Virtualized Switches

Attack vector:

- The virtualized data plane



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.

Another Front: Virtualized Switches

Attack vector:

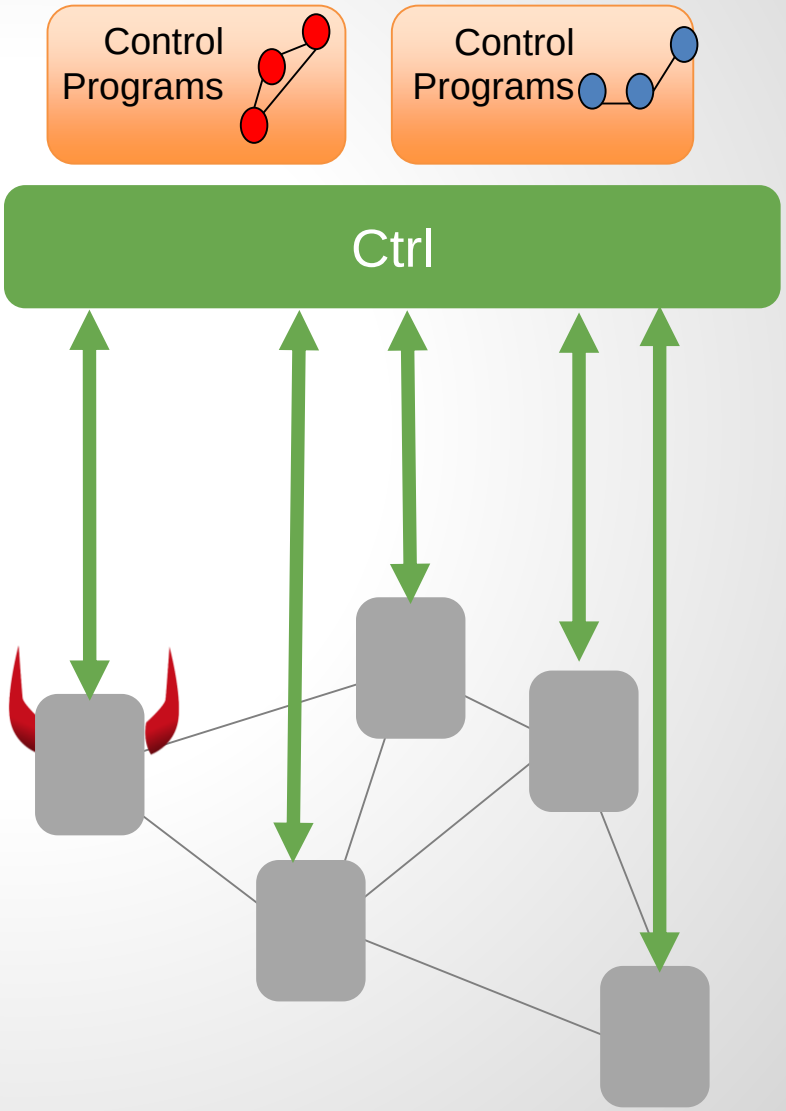
- The virtualized data plane

Background:

- Packet processing and other network functions are more and more virtualized
- E.g., runing on servers at the edge of the datacenter
- Example: OVS

Advantage:

- Cheap and performance ok!
- Fast and easy deployment



Security Challenges: Insecure Dataplane

Attack vector:

- ❑ The virtualized data plane

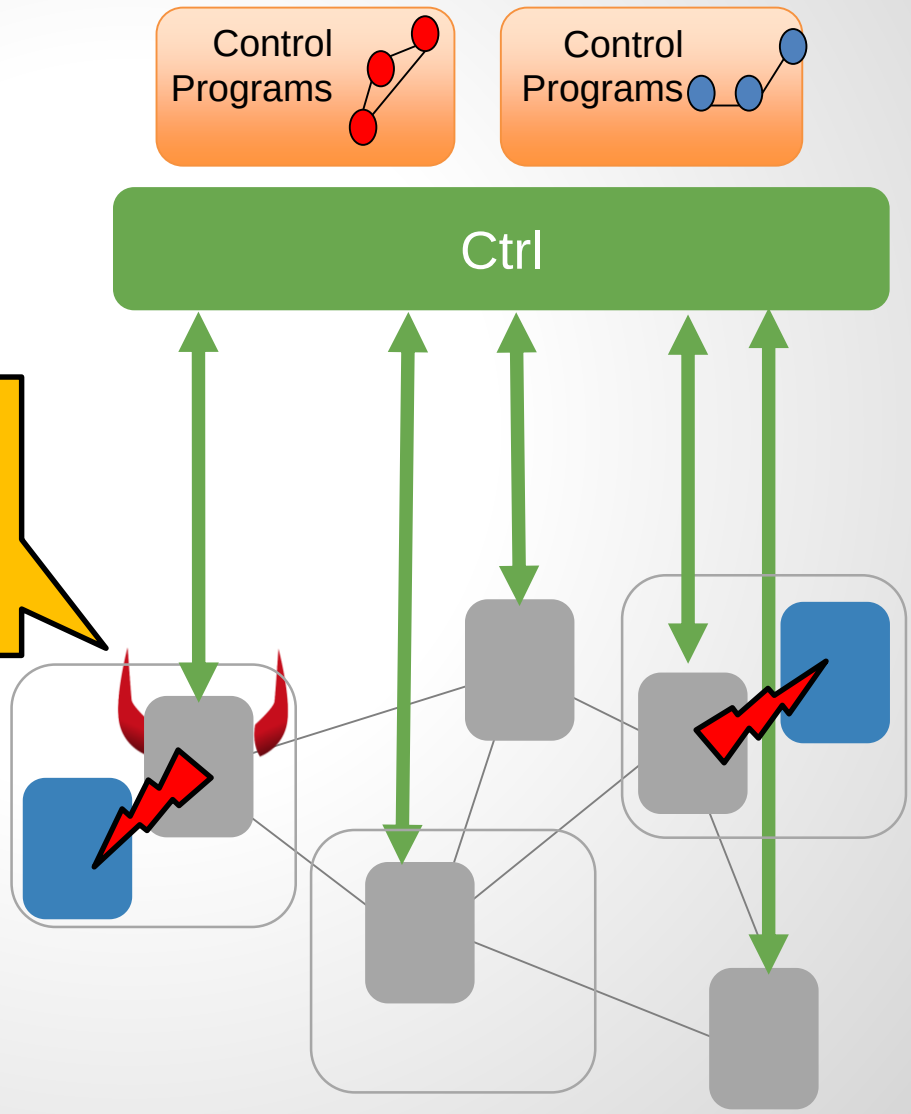
Background:

- ❑ Packet processing and other network functions are more and more virtua
- ❑ E.g., the e
- ❑ Example: OVS

New vulnerability: collocation. Switches run with evelated (root) priviledges.

Advantage:

- ❑ Cheap and performance ok!
- ❑ Fast and easy deployment



Security Challenges: Insecure Dataplane

Attack vector:

- ❑ The virtualized data plane

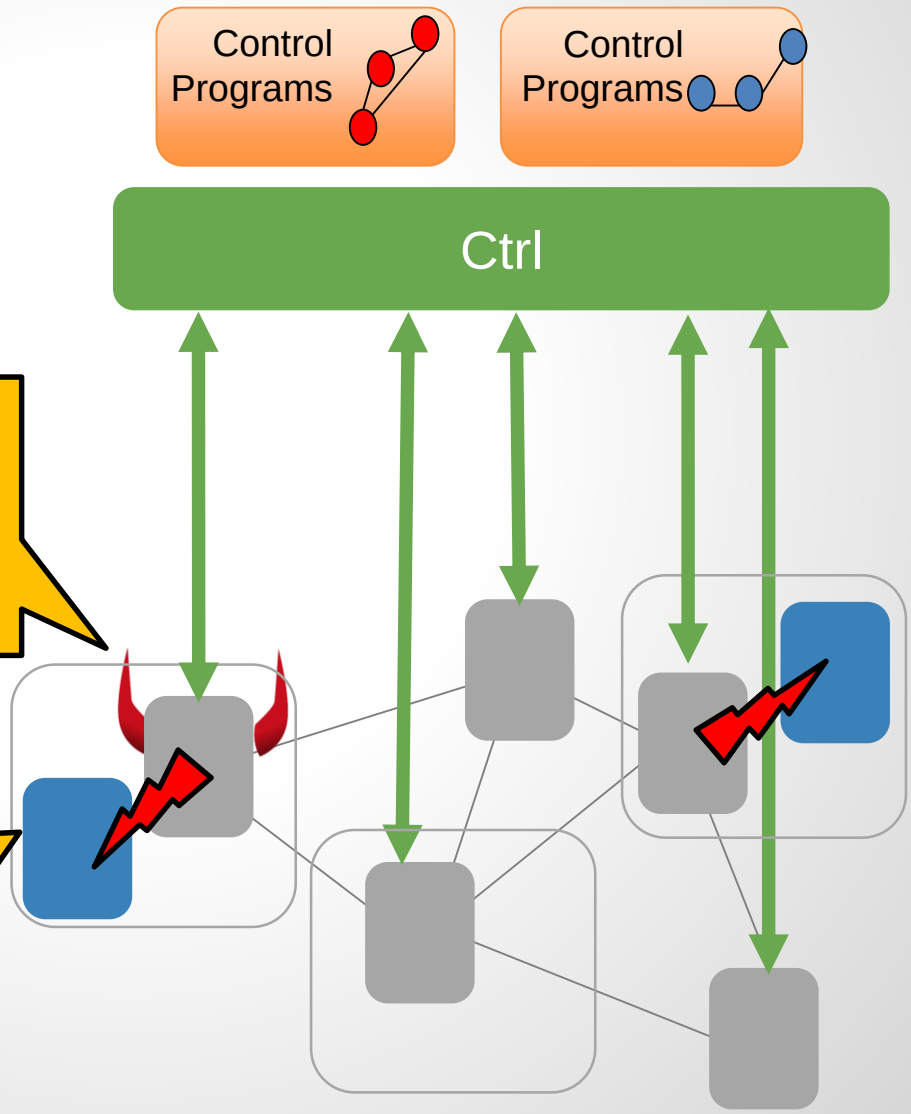
Background:

- ❑ Packet processing and other network functions are more and more virtua
- ❑ E.g., the e
- ❑ Example: OVS

New vulnerability: collocation. Switches run with evelated (root) priviledges.

Advantage:

Collocated with e.g., controllers, hypervisors, guest VMs, VM image and network management, identity management (of admins and tenants), etc.



A Case Study: OVS

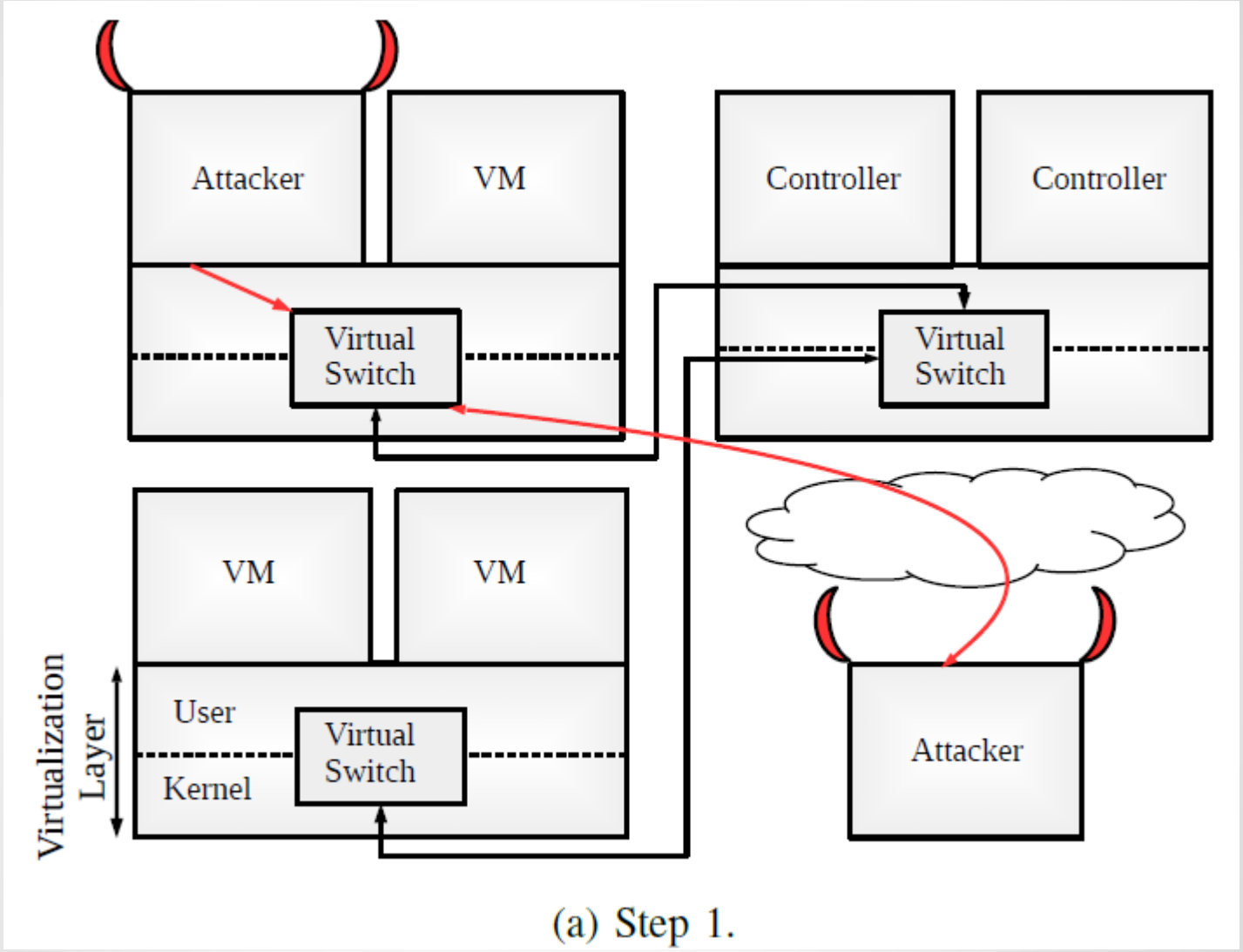
- OVS: a production quality switch, widely deployed in the Cloud
- After fuzzing just 2% of the code, found major vulnerabilities:
  - E.g., two stack overflows when malformed MPLS packets are parsed
- These vulnerabilities can easily be weaponized:
  - Can be exploited for arbitrary remote code execution
  - E.g., our «reign worm» compromised cloud setups within 100s
- Significance
  - It is often believed that only state-level attackers (with, e.g., control over the vendor's supply chain) can compromise the data plane
  - Virtualized data planes can be exploited by very simple, low-budget attackers: e.g., by renting a VM in the cloud and sending a single malformed MPLS packet

The Reign Worm

Exploits 4 problems:

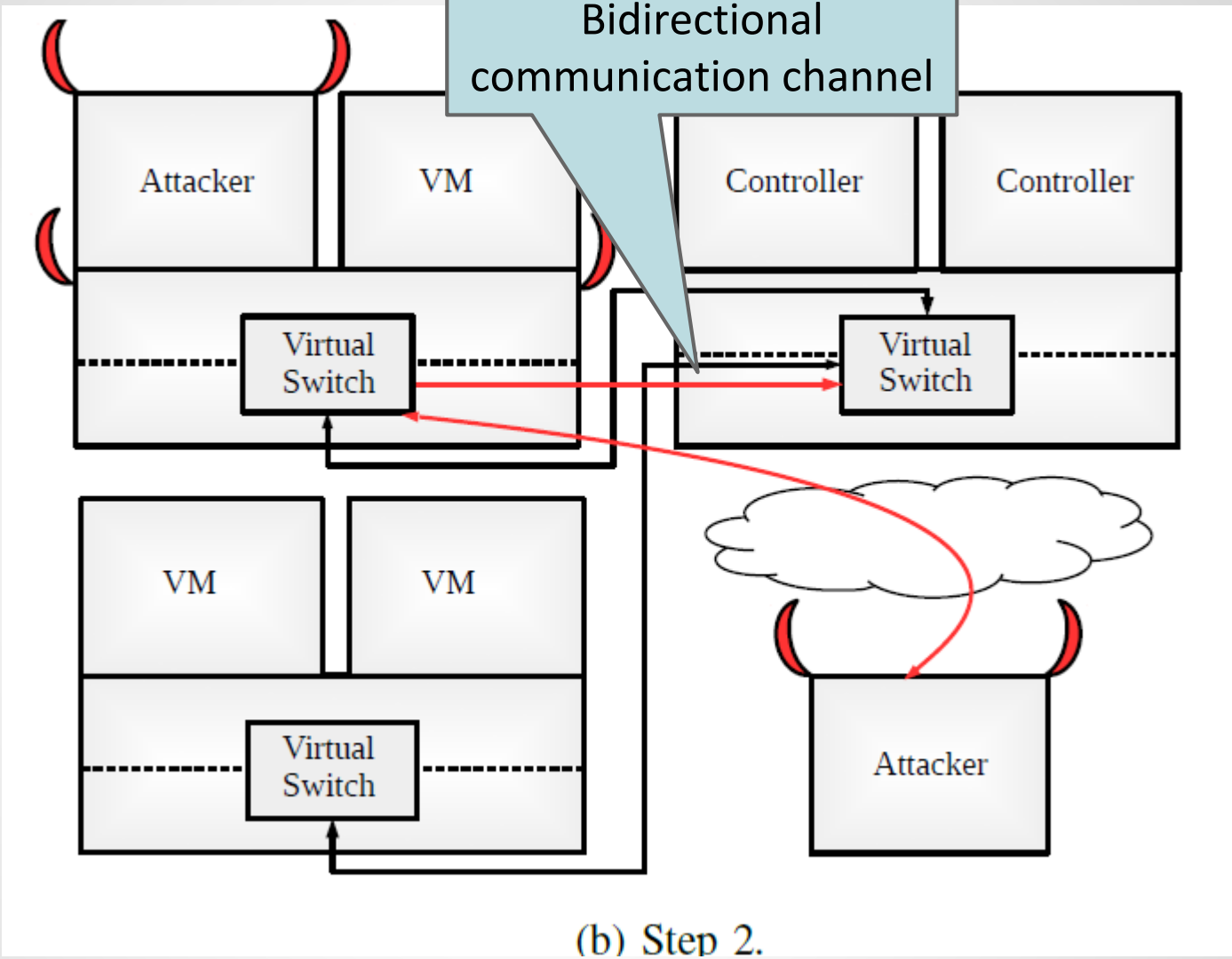
1. **Security assumptions:** Virtual switches often run with elevated (root) priviledges by design.
2. **Collocation:** virtual switchs reside in virtualized servers (Dom0), and are hence collocated with other and possibly critical cloud software, including controller software
3. **Logical centralization:** the control of data plane elements is often outsourced to a centralized software. The corresponding bidirectional communication channels can be exploited to spread the worm further.
4. **Support for extended protocol parsers:** Virtual switches provide functionality which goes beyond basic protocol locations of normal switches (e.g., handling MPLS in non-standard manner)

The Reign Worm: Step 1



Attacker VM sends a malicious packet that compromises its server, giving the remote attacker control of the server.

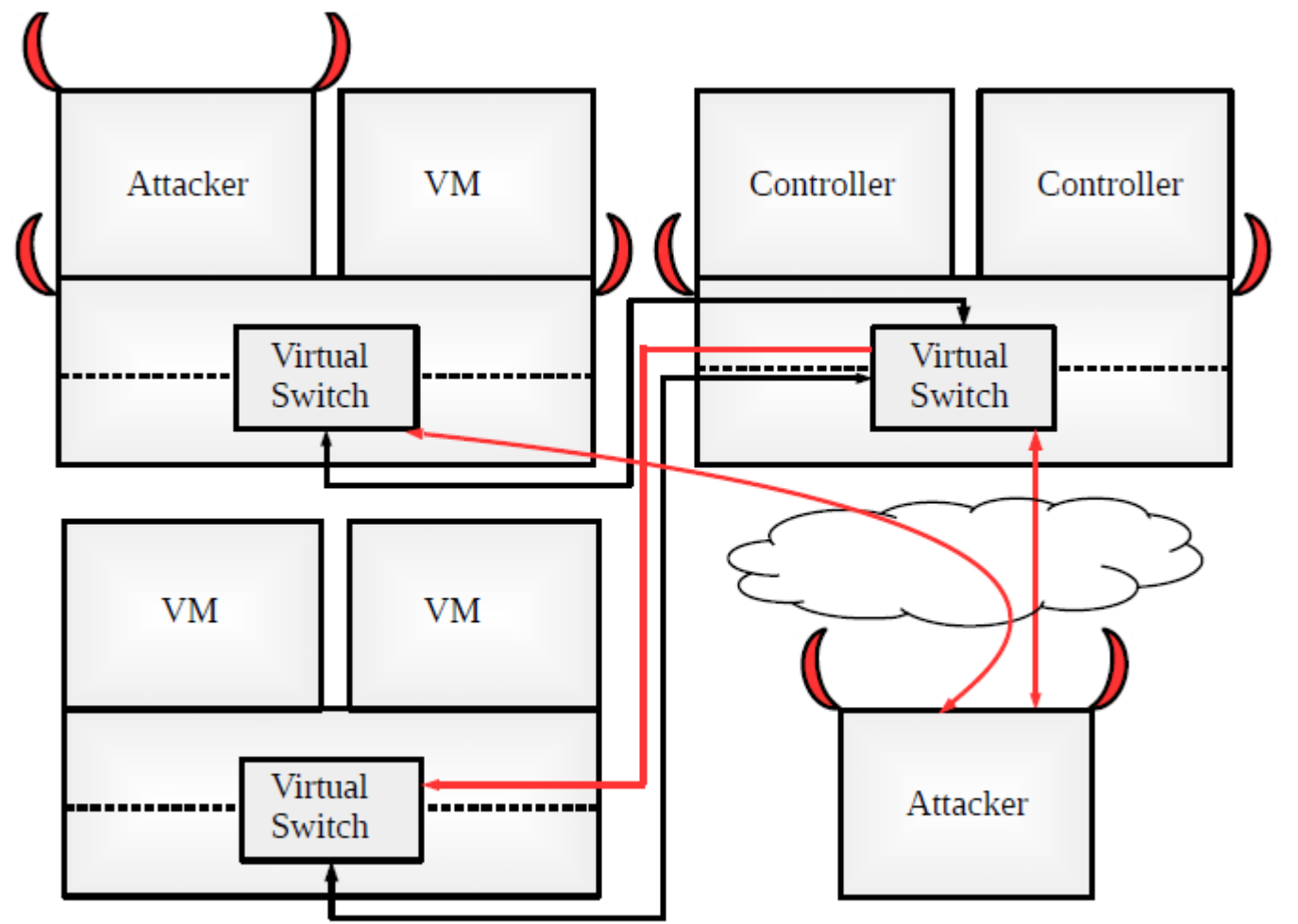
The Reign Worm: Step 2



(b) Step 2.

Attacker controlled server compromises the controllers' server, giving the remote attacker control of the controllers' server.

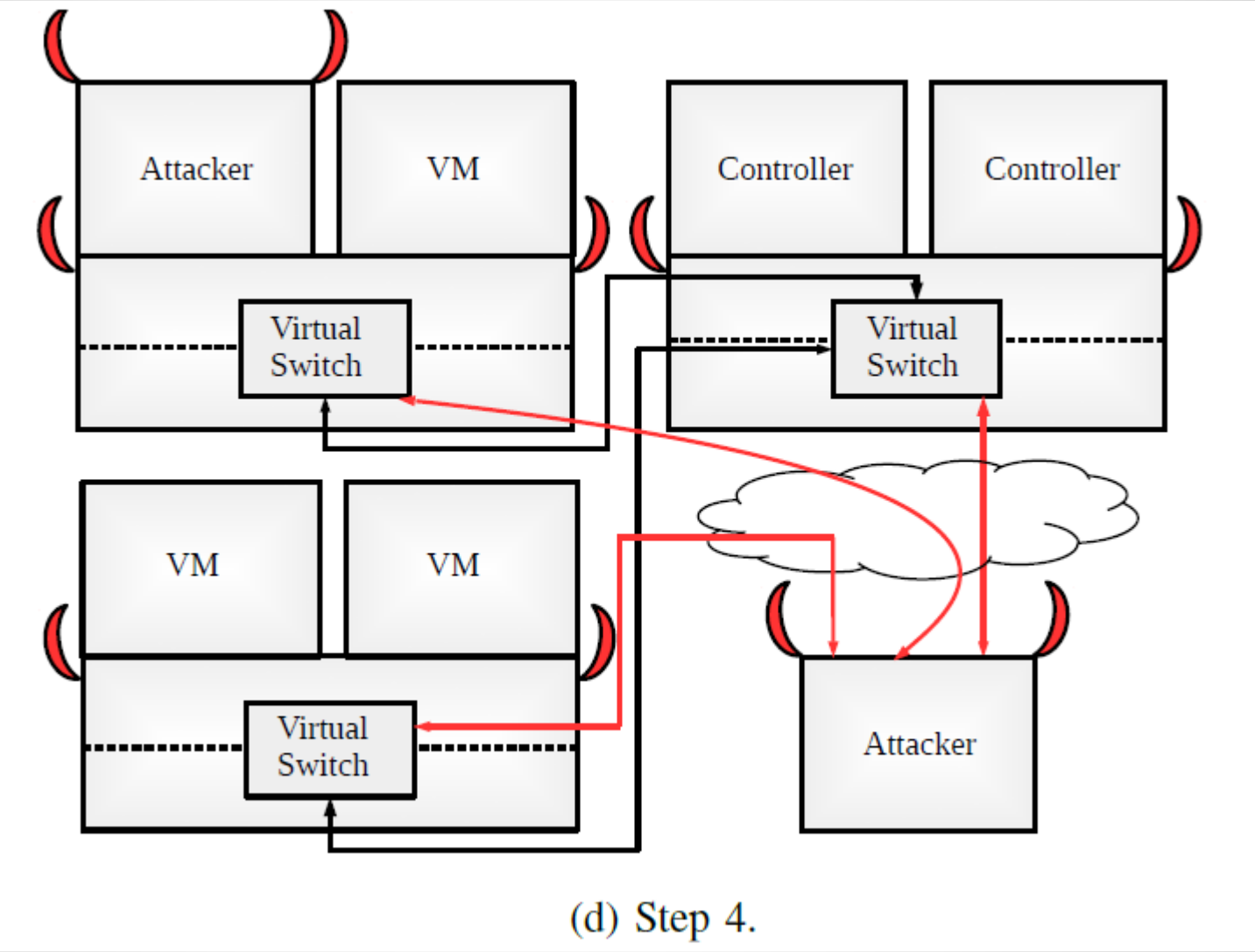
The Reign Worm: Step 3



(c) Step 3.

The compromised controllers' server propagates the worm to the remaining uncompromised server.

The Reign Worm: Step 4



(d) Step 4.

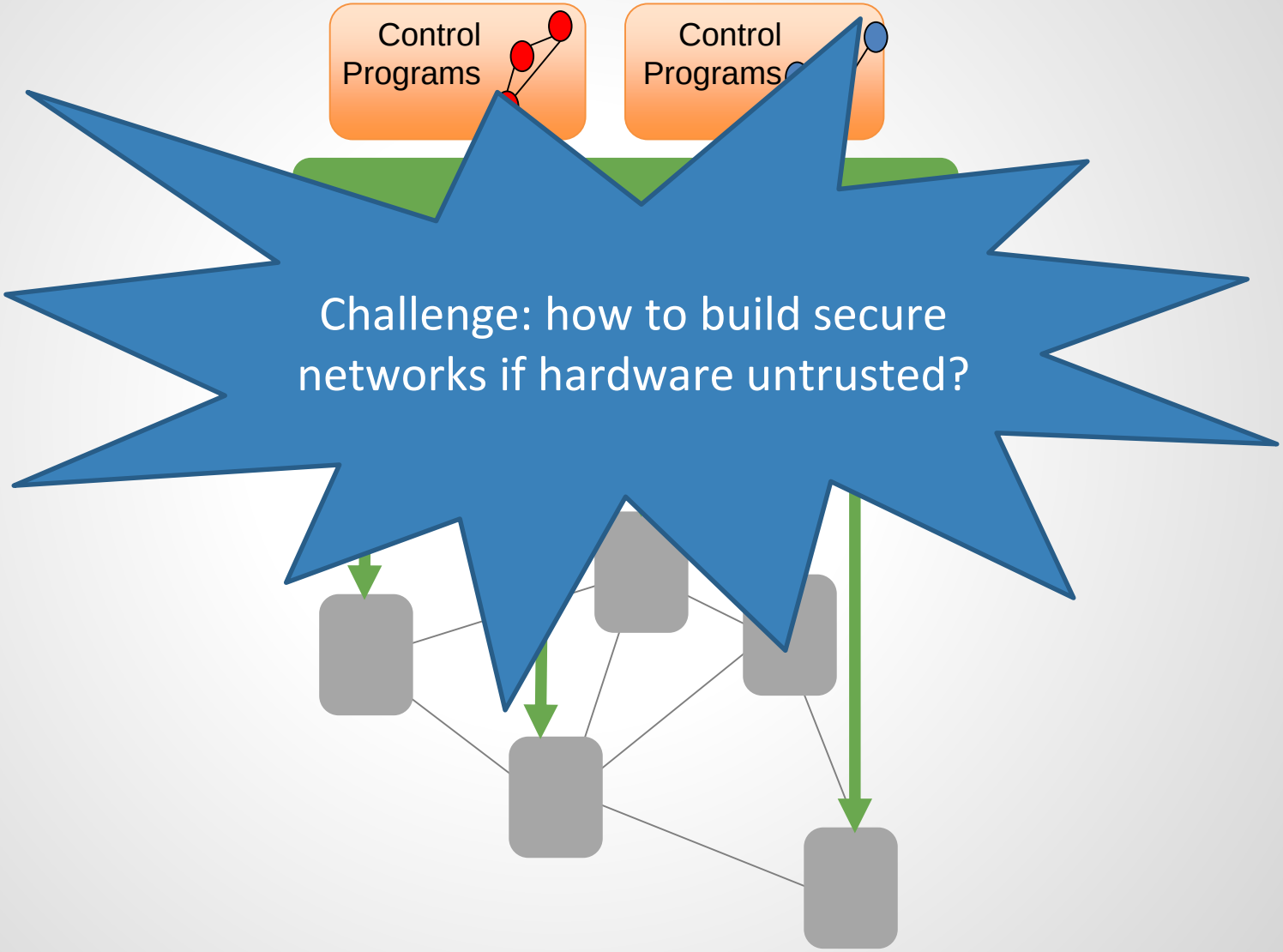
All the servers are controlled by the remote attacker.

Further Reading

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.

Let’s talk about security!

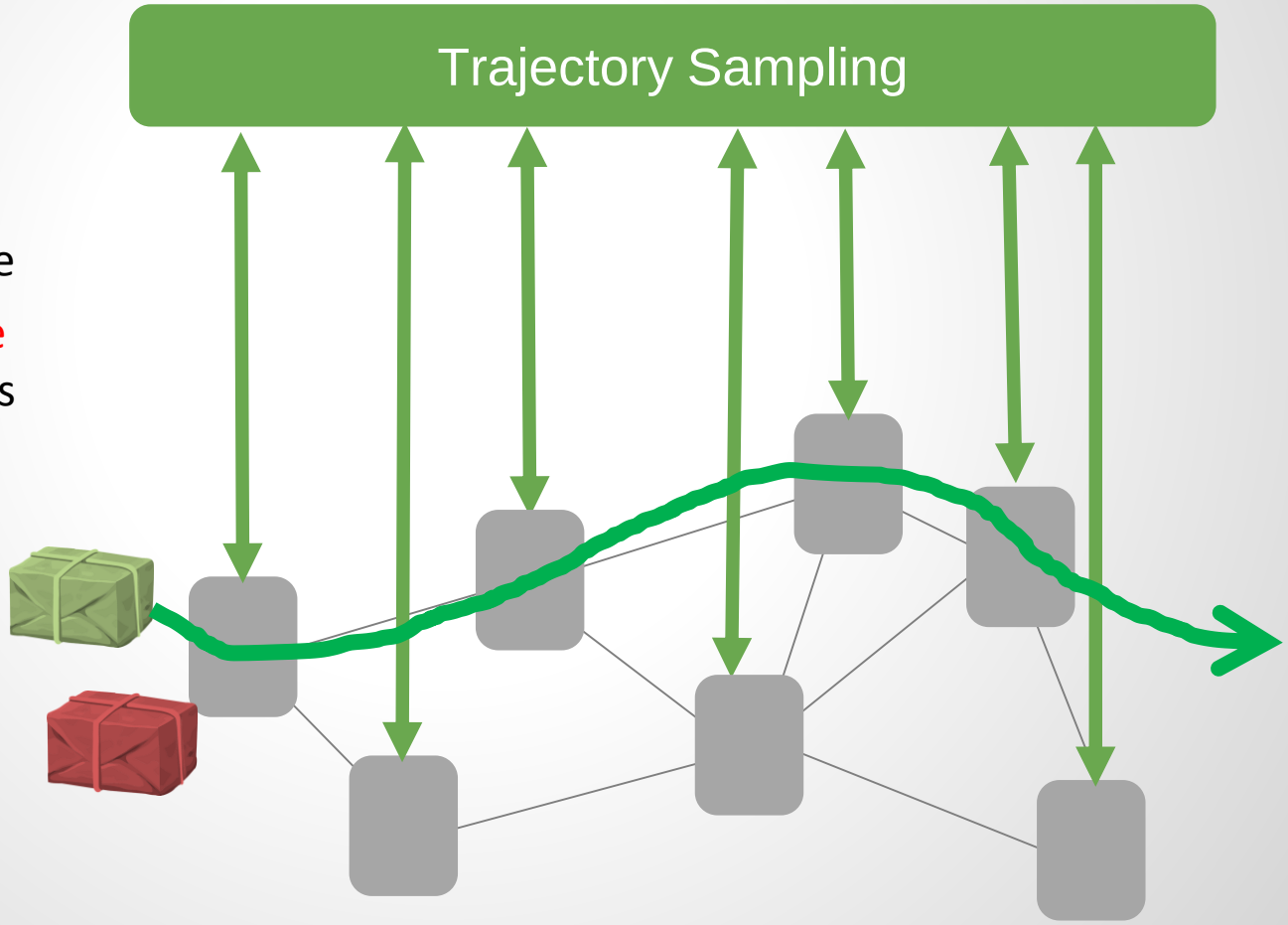


Opportunity: Adversarial Trajectory Sampling

- Classic tool to monitor packet routes: trajectory sampling

Principle:

- Sample subset of packets (based on hash value) anytime
- Get complete route for sampled packets
- Efficient: sampling

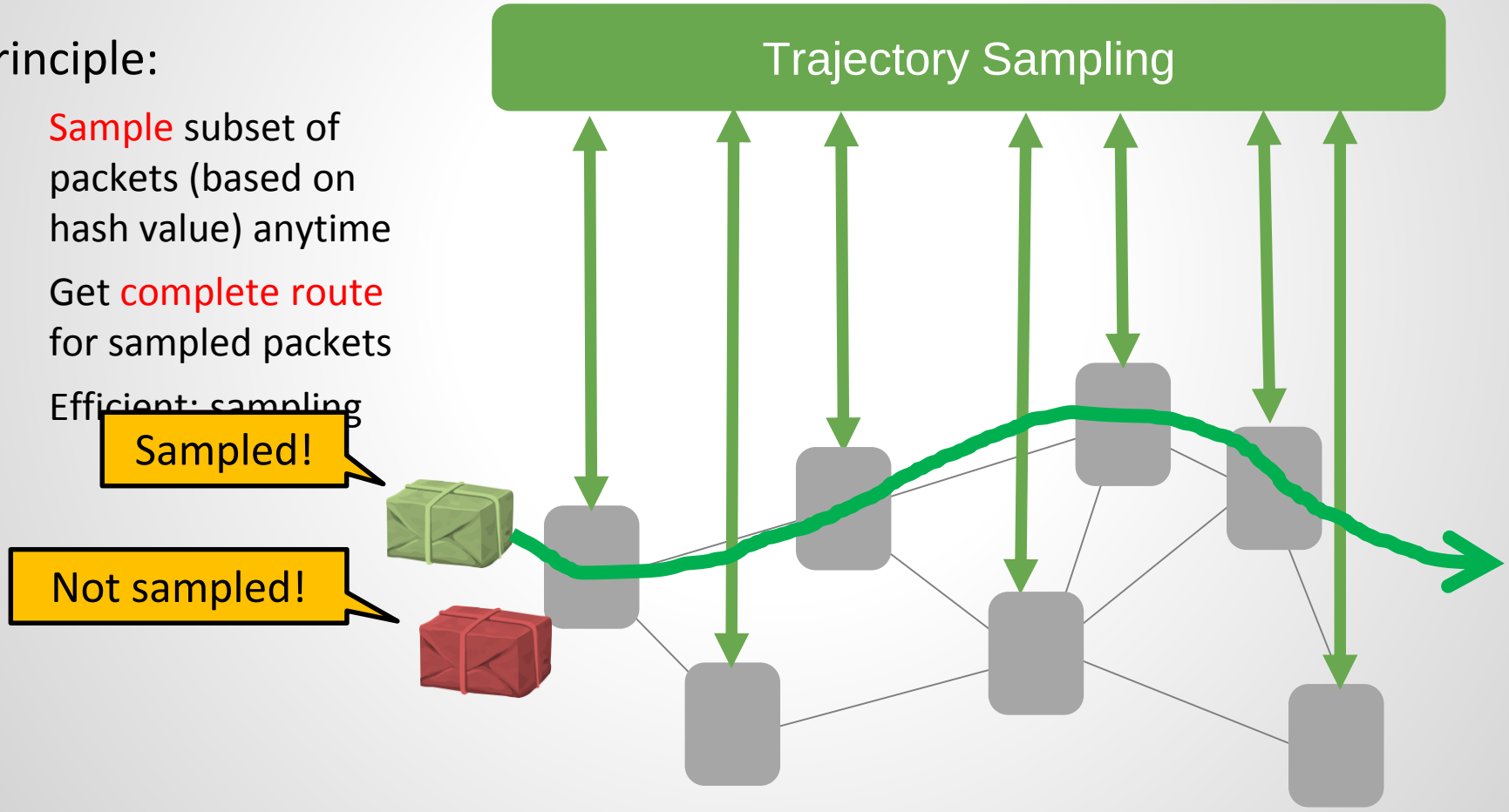


Opportunity: Adversarial Trajectory Sampling

- Classic tool to monitor packet routes: trajectory sampling

Principle:

- Sample subset of packets (based on hash value) anytime
- Get complete route for sampled packets
- Efficient: sampling

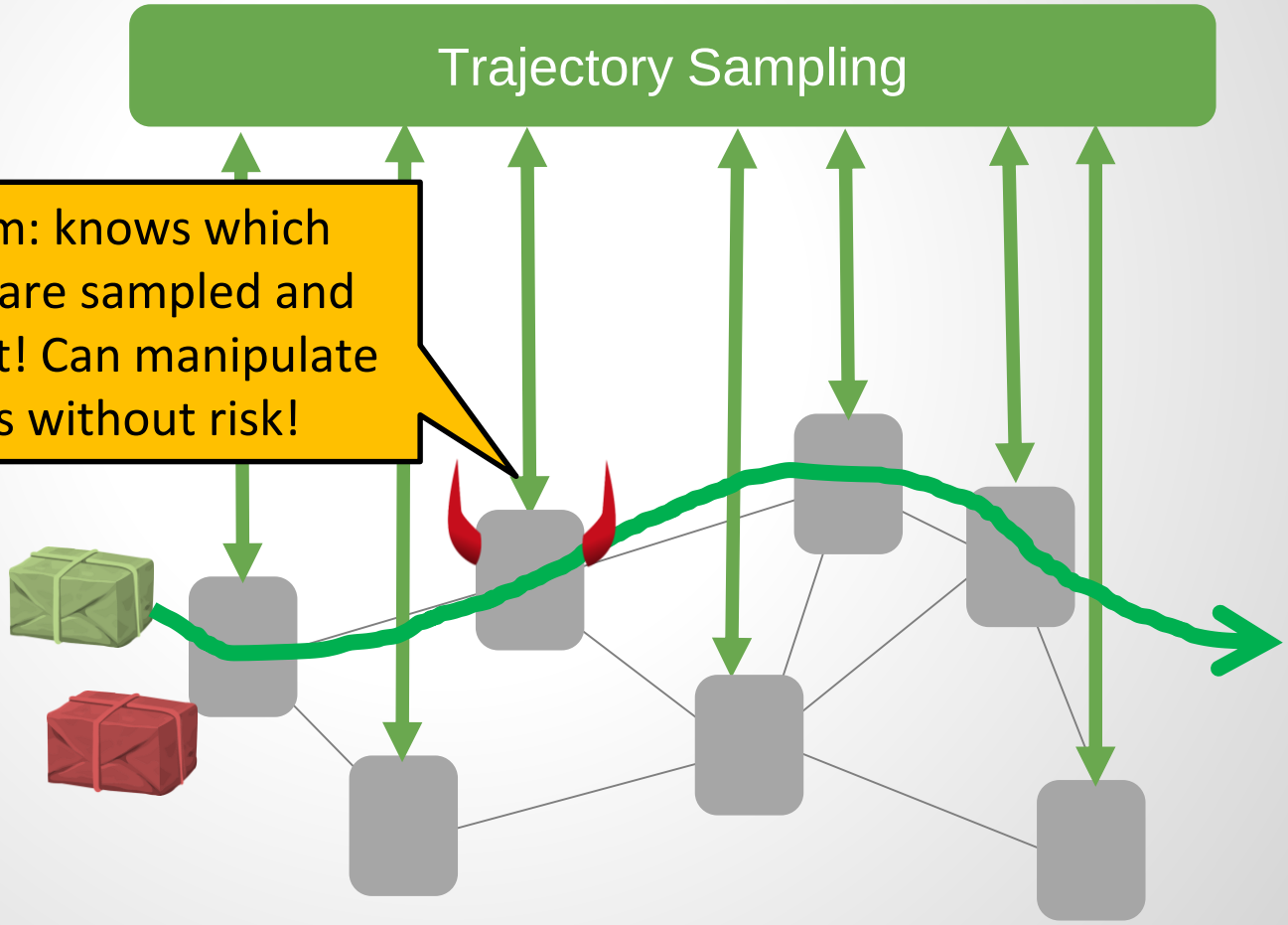


Opportunity: Adversarial Trajectory Sampling

- Classic tool to monitor packet routes: trajectory sampling

Principle:

- Sample subset of packet
  hash v
- Get co
  for san
- Efficient: sampling

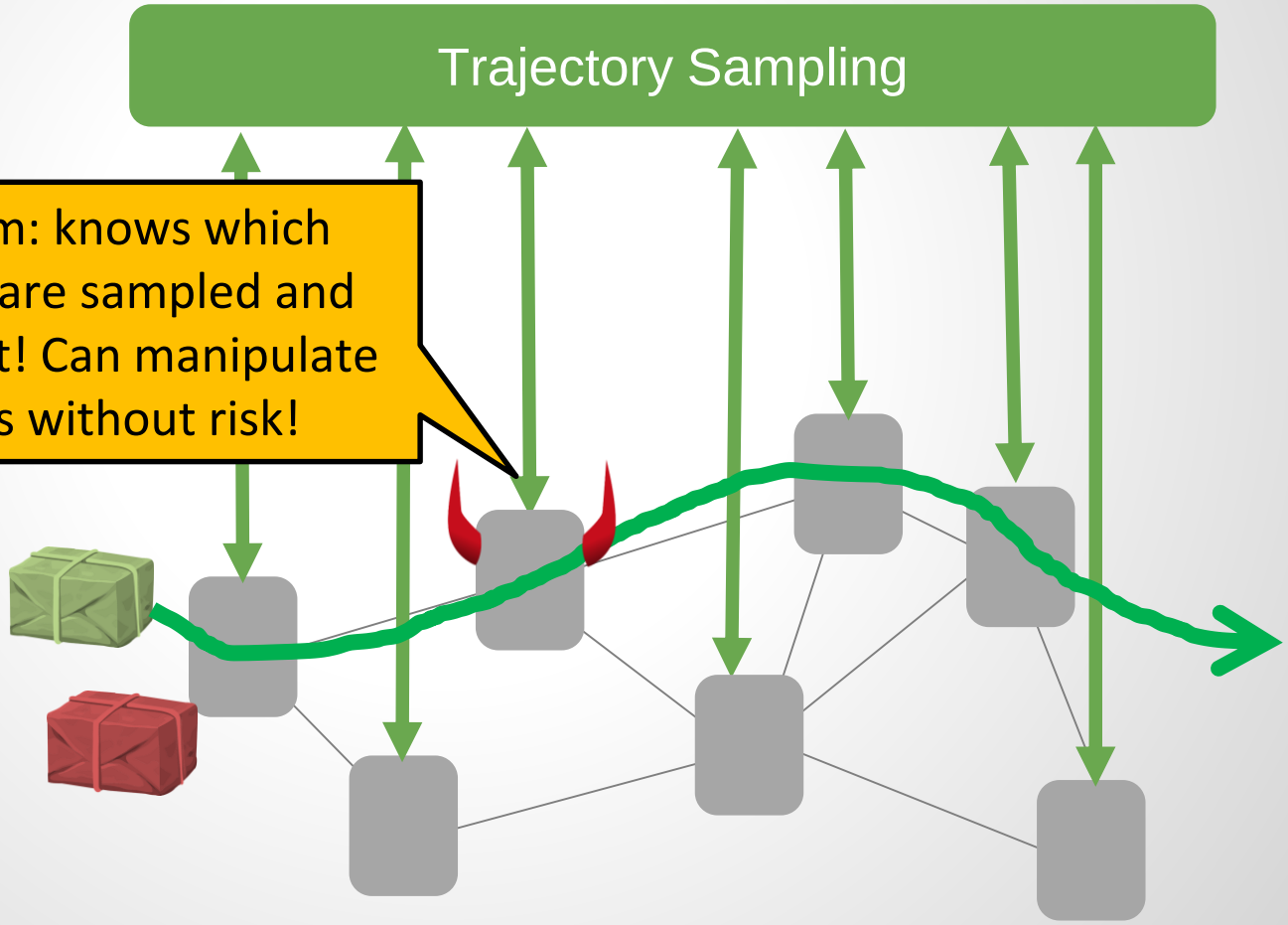


Opportunity: Adversarial Trajectory Sampling

- Classic tool to monitor packet routes: trajectory sampling

Principle:

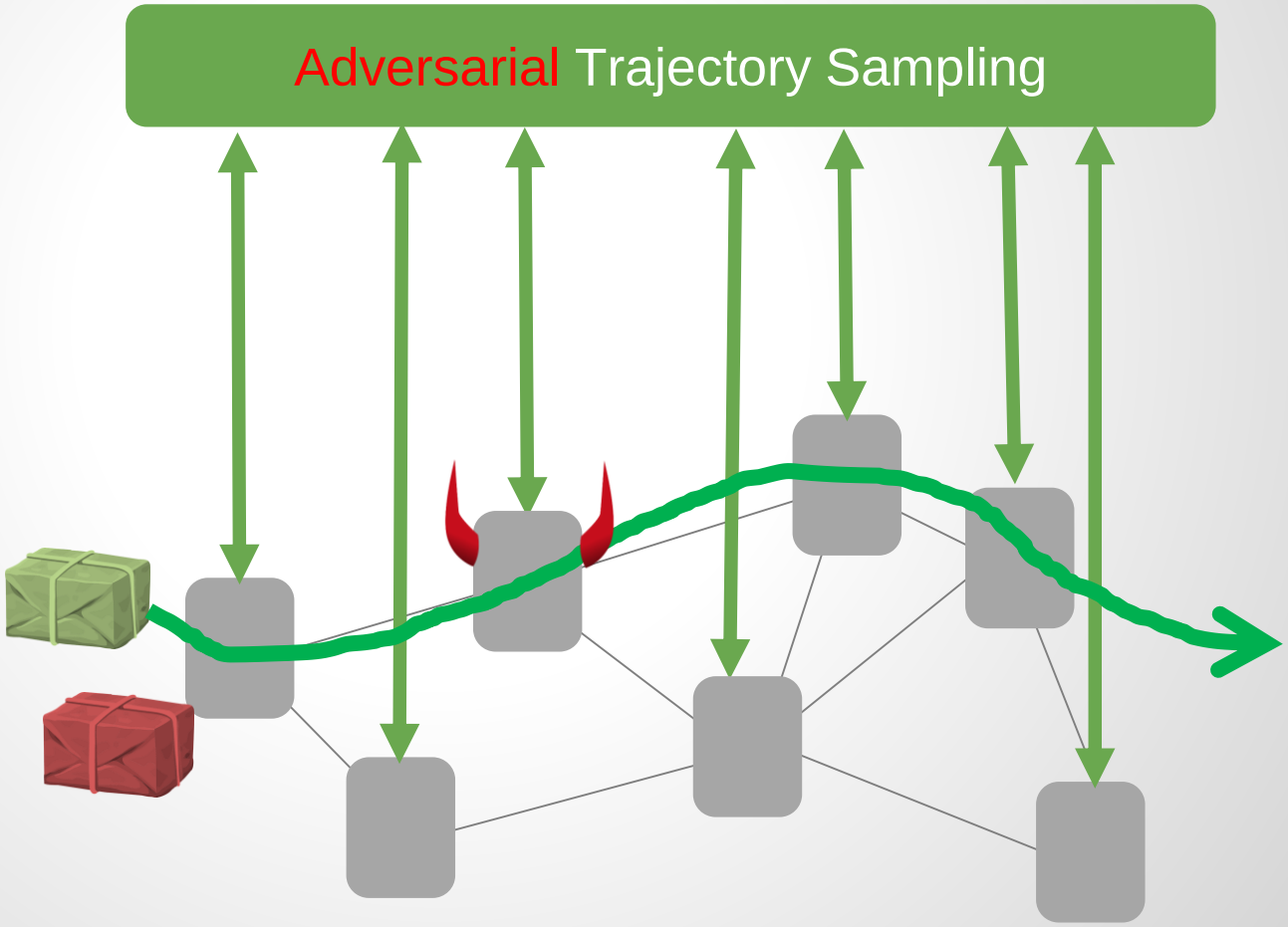
- Sample subset of packet
  hash v
- Get co
  for san
- Efficient: sampling



How to make trajectory sampling secure to malicious switches?

Opportunity: Adversarial Trajectory Sampling

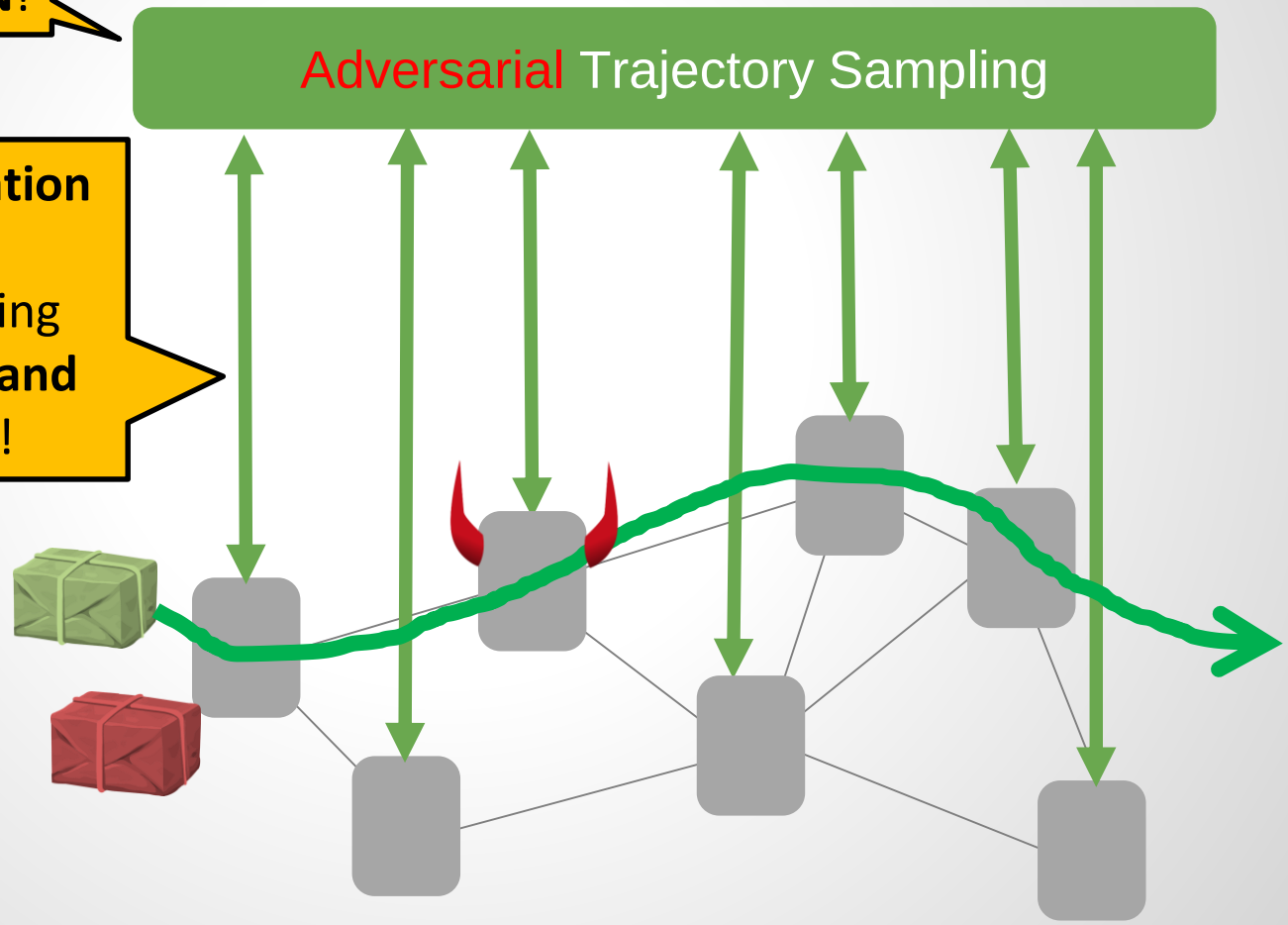
- Classic tool to monitor packet routes: trajectory sampling



Opportunity: Adversarial Trajectory Sampling

- Classic tool to monitor packet routes: trajectory sampling

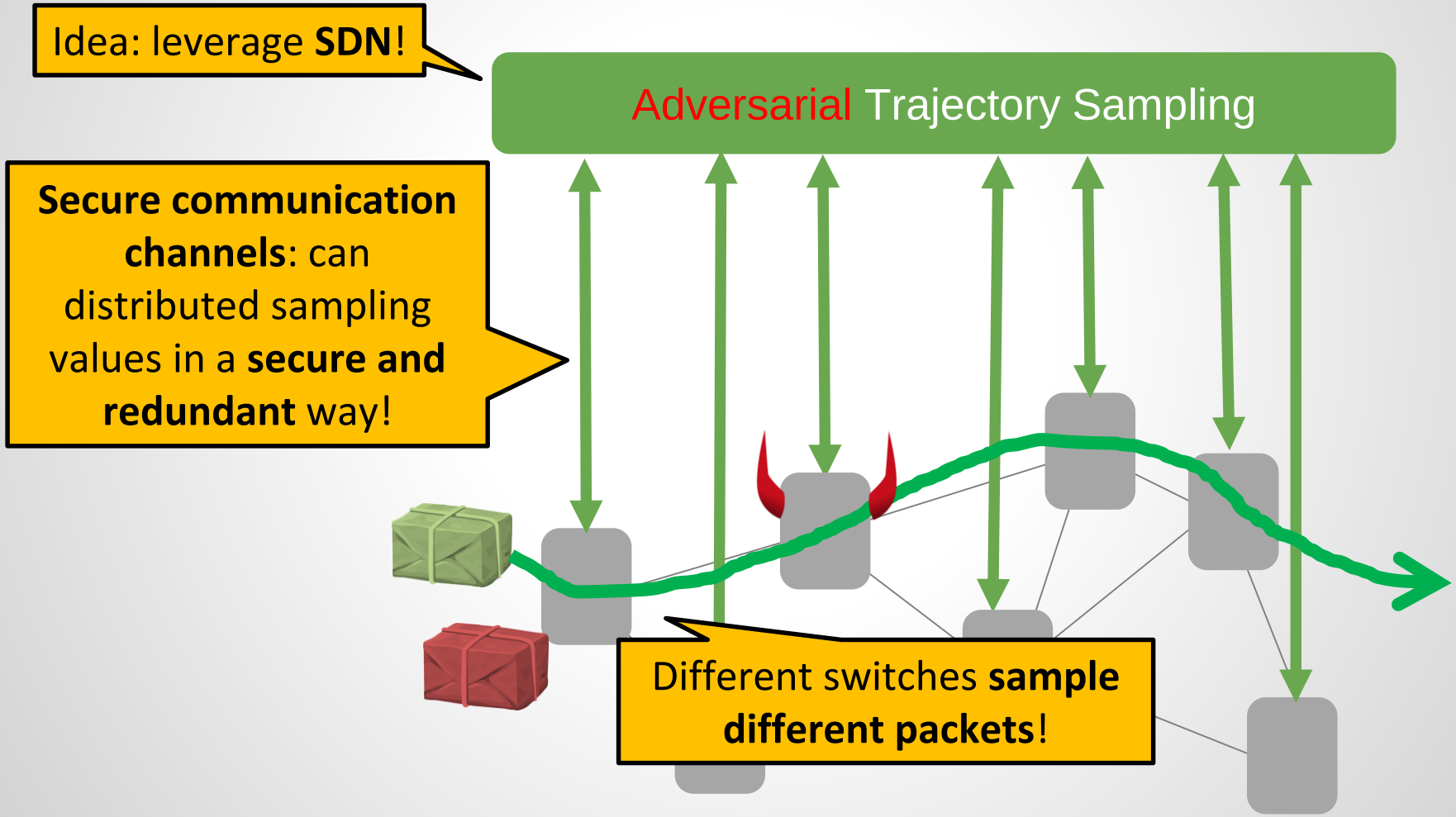
Idea: leverage **SDN**!



Secure communication channels: can distributed sampling values in a **secure and redundant** way!

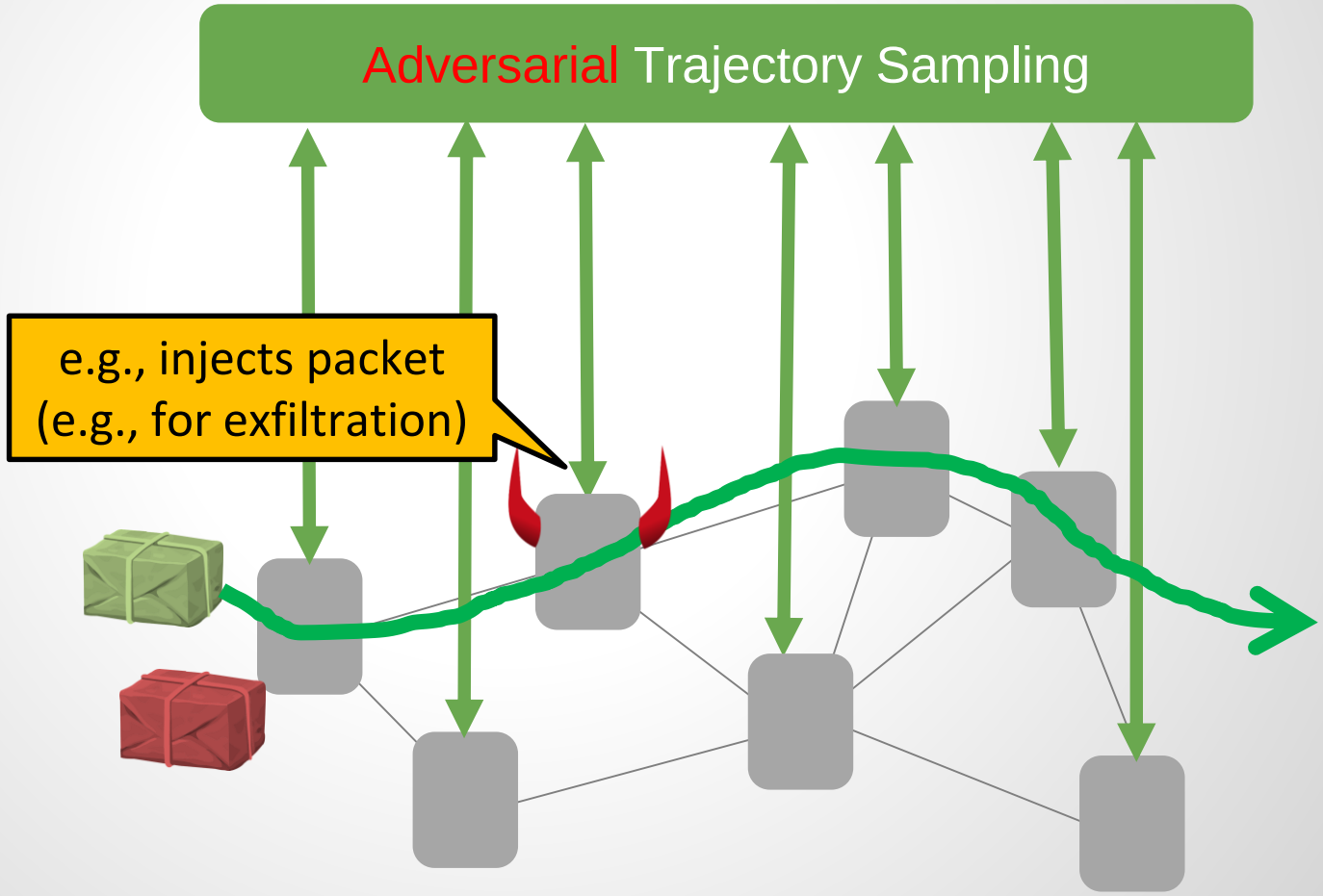
Opportunity: Adversarial Trajectory Sampling

- Classic tool to monitor packet routes: trajectory sampling



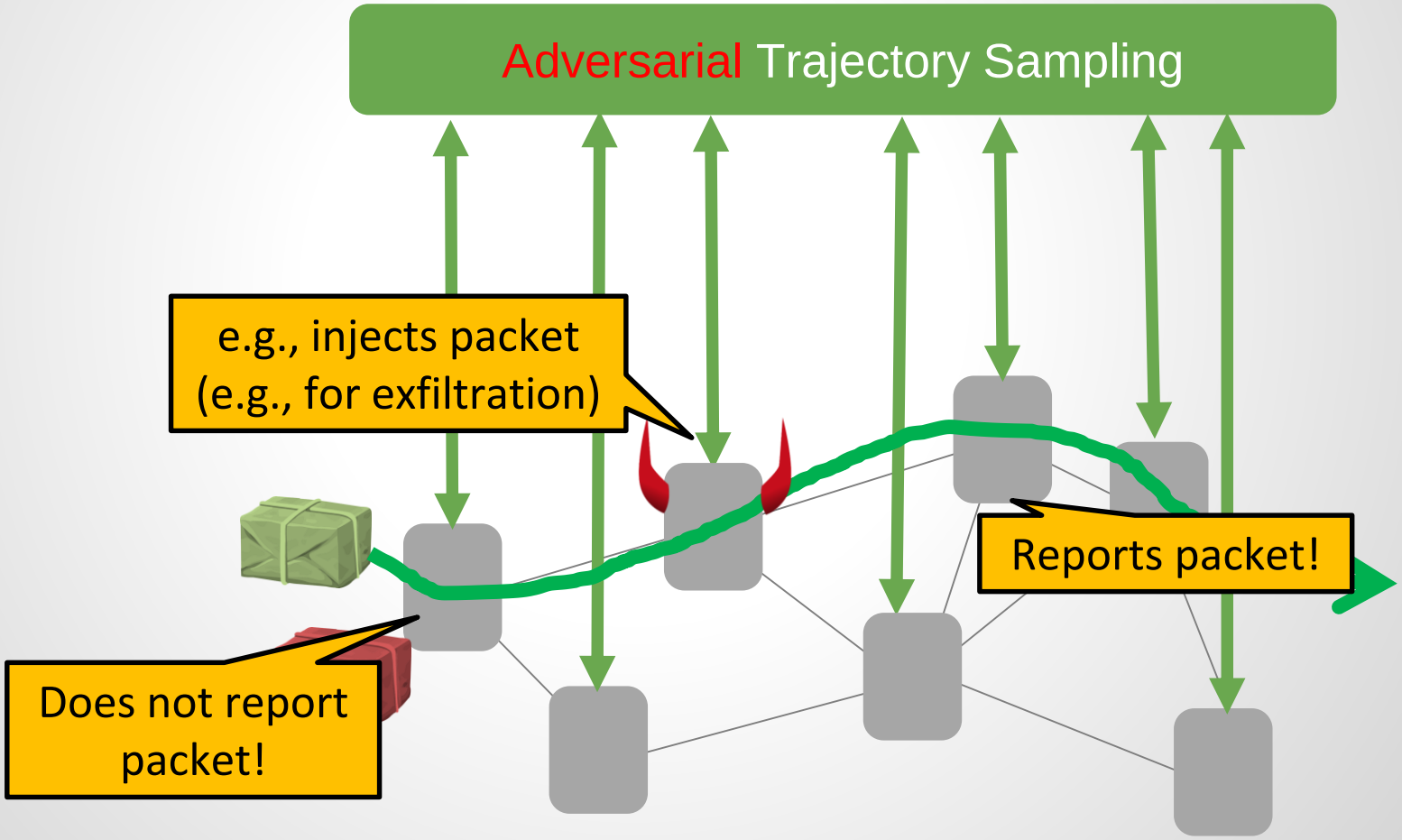
Opportunity: Adversarial Trajectory Sampling

- Classic tool to monitor packet routes: trajectory sampling



Opportunity: Adversarial Trajectory Sampling

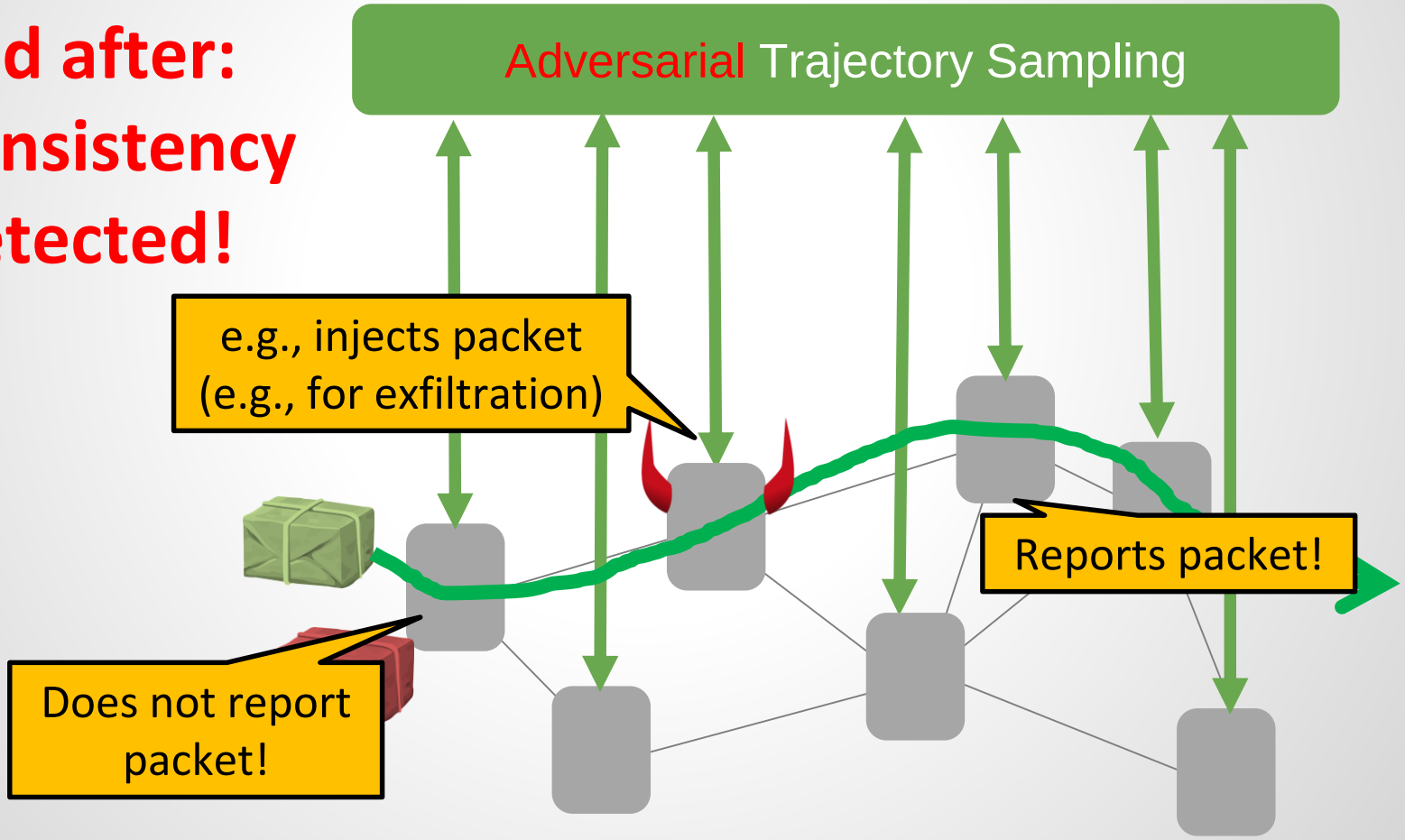
- Classic tool to monitor packet routes: trajectory sampling



Opportunity: Adversarial Trajectory Sampling

- Classic tool to monitor packet routes: trajectory sampling

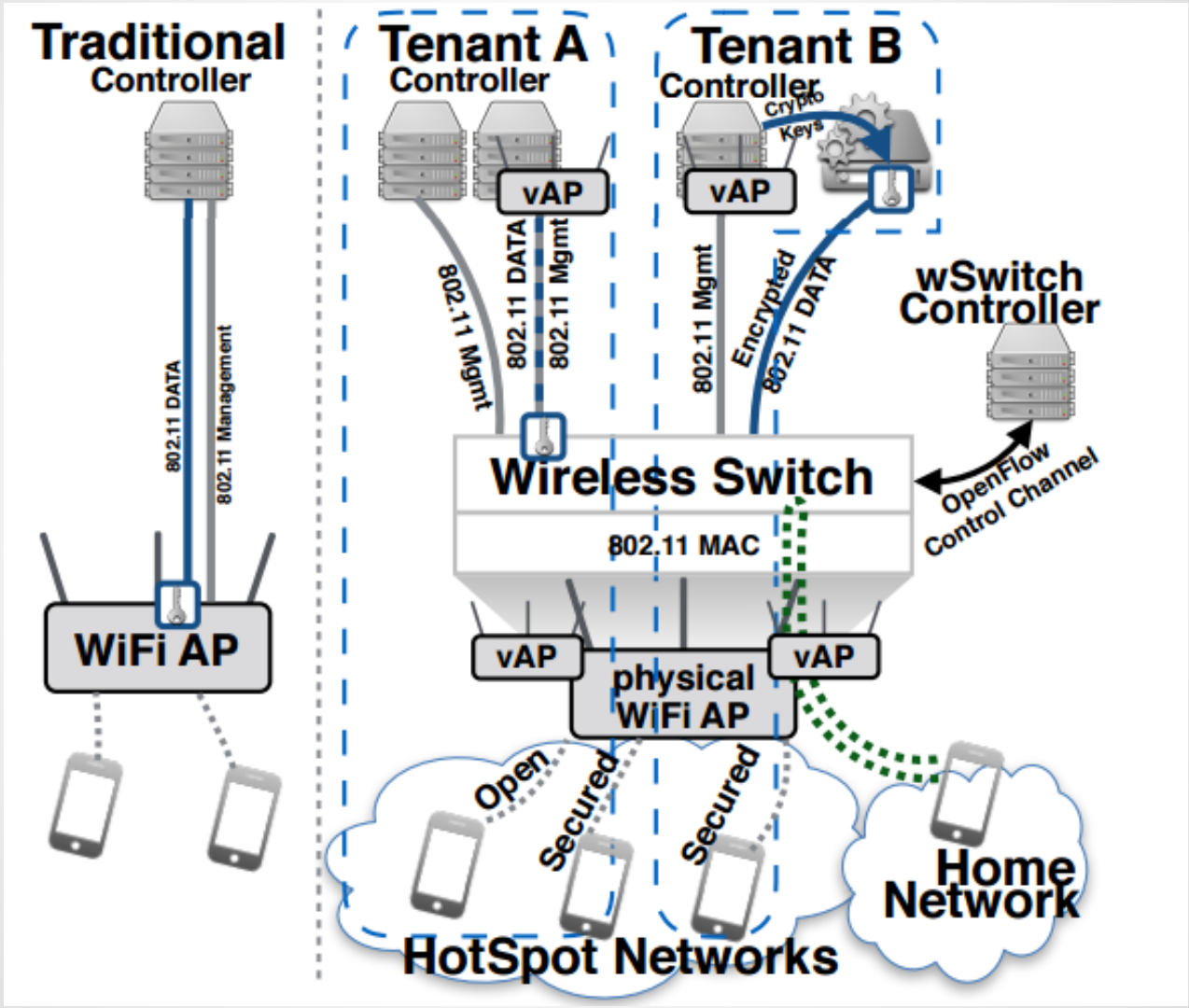
If sampled before and after: Inconsistency detected!



Further Reading

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

Software-Defined Wifi



Further Reading

OpenSDWN: Programmatic Control over Home and Enterprise WiFi

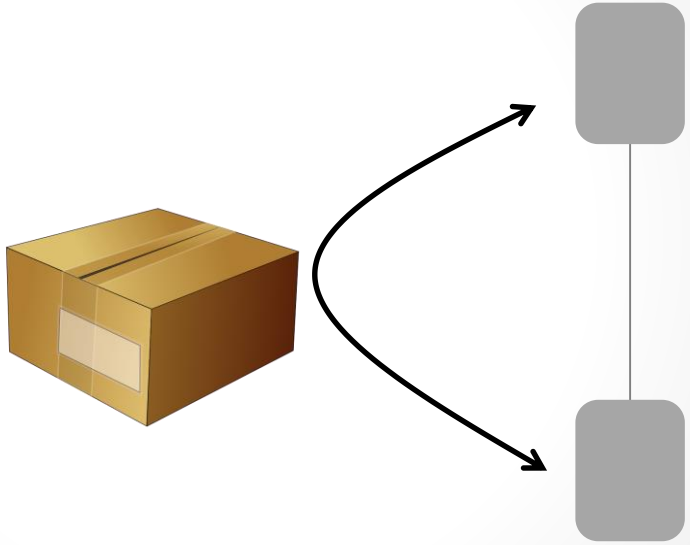
Julius Schulz-Zander, Carlos Mayer, Bogdan Ciobotaru, Stefan Schmid, and Anja Feldmann.

ACM Sigcomm Symposium on SDN Research (**SOSR**), Santa Clara, California, USA, June 2015.

Automatically Verifiable!

Computational Tractability?

Emulating Turing machine with two switches? (Ribbon in packet?)



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.

Conclusion

- SDN introduces many flexibilities
- But also new challenges
  - How to exploit flexibilities algorithmically?
  - How to deal with remote controller(s)?
- New (and old) security challenges
- Another grand challenge: predictable performance in virtualized SDNs

Algorithms for flow rerouting:

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

Security of the data plane:

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.
See also CVE-2015-7516.

teleportation

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.

attacking the cloud