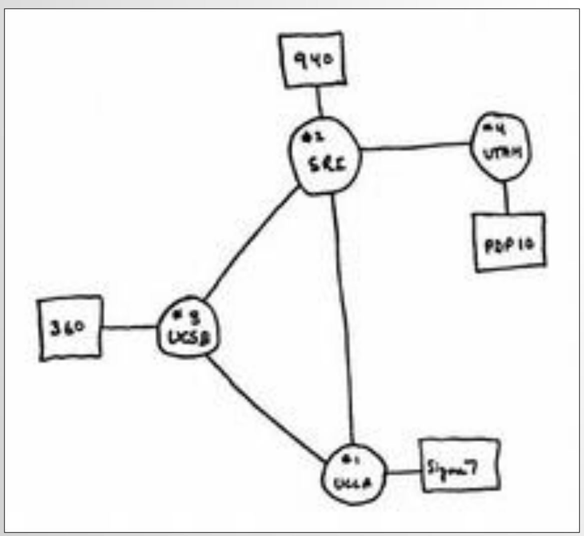
Algorithmic and Security Challenges in Programmable and Virtualized Networks

Stefan Schmid

Aalborg University, DK & TU Berlin, DE

Computer Networks

- Computer networks (datacenter networks, enterprise networks, Internet) have become a critical infrastructure of the information society
- The Internet is very successful so far: hardly any outages…
- … despite a huge shift in scale and applications



Goal: connectivity between researchers

Applications: file transfer, email

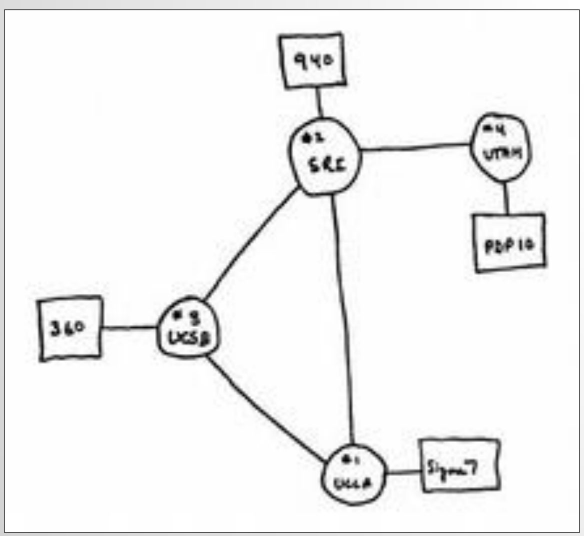
Goal: QoS, security, …

Applications: live streaming, IoT, etc.

Computer Networks

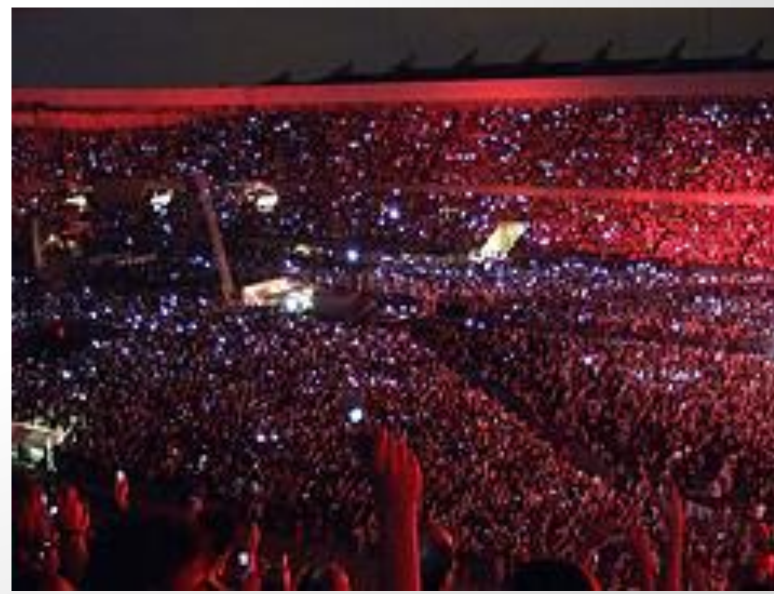
- ...tworks, enterprise networks, ...rastructure of the information
- ...r: hardly any outages…
- ...applications

> *Danny Hillis*, TED* talk, Feb. 2013, about trust in the Internet in the 80s: “There were two Dannys. I knew both. Not everyone knew everyone, but there was an atmosphere of trust.”



Goal: connectivity between researchers

Applications: file transfer, email



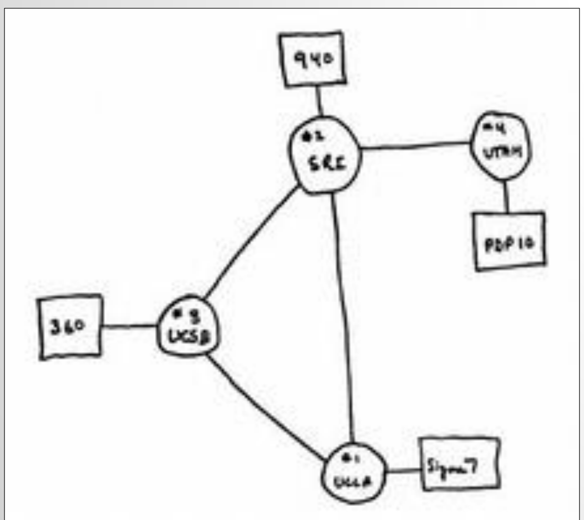
Goal: QoS, security, …

Applications: live streaming, IoT, etc.

Computer Networks

- Computer networks (datacenter networks, enterprise networks, Internet) have become a critical infrastructure of the information society
- The Internet i outages…
- … despite a hu

The underlying technologies have hardly changed over all these years!



Goal: connectivity between researchers

Applications: file transfer, email

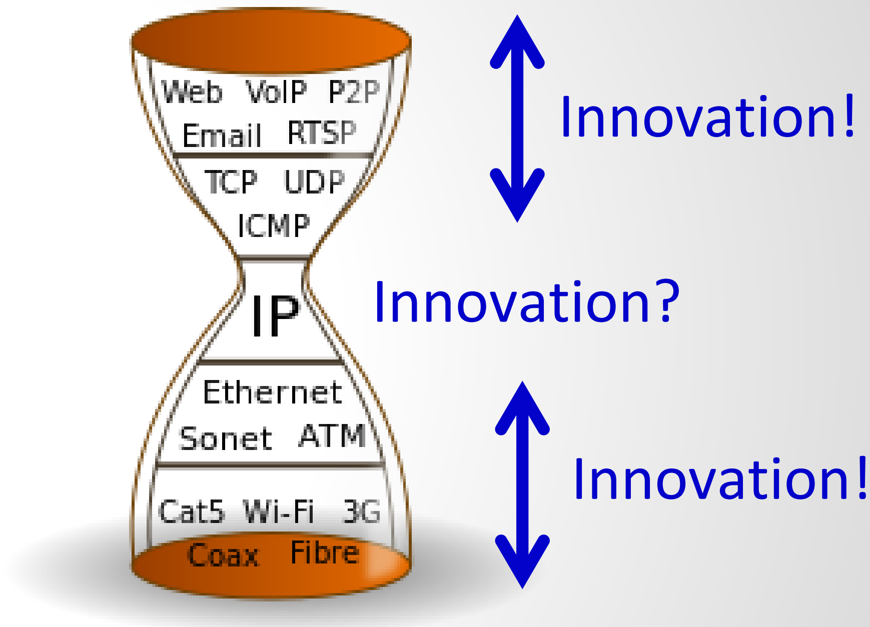
Goal: QoS, security, …

Applications: live streaming, IoT, etc.

Ready for the Future?

- However: do computer networks still meet the dependability requirements in the future?

- Example Internet-of-Things:
  - IPv4: ~4.3 billion addresses
  - Gartner study: 20+ billion "smart things" by 2020
  - Recent DDoS attack based on IoT (almost 1TB/s, coming from webcames, babyphones, etc.)




home › tech

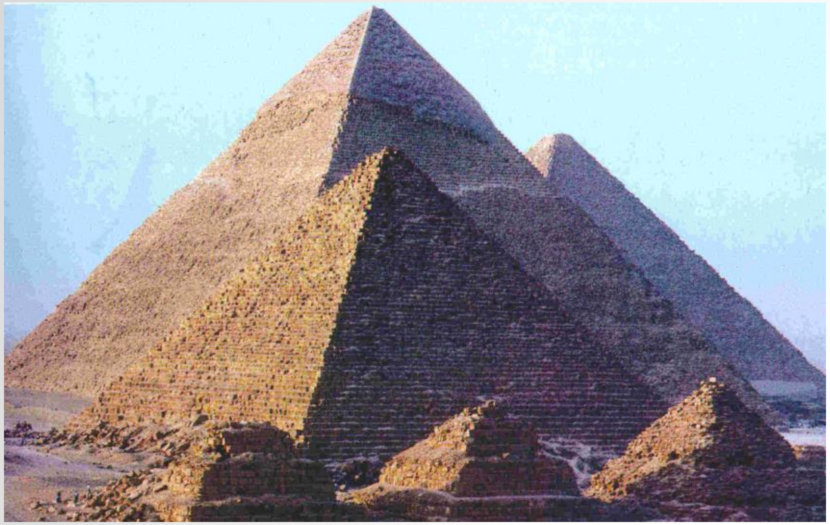
Internet of things

Can we secure the internet of things in time to prevent another cyber-attack?

Easy-to-hijack 'smart' devices just crashed some of the world's biggest online platforms. Experts say it's a wake-up call to improve security - and quickly

Support our fearless, independent journalism with a contribution or by

Problem 1: Security in the Internet



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

Source: Slide by Adrian Perrig

Problem 2: Reliability

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

Source: Talk by Nate Foster at DSDN Workshop

Problem 3: Troubleshooting

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, an emergency team was assembled with experts in compute, storage and networking:
  - **The compute team:** 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.

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

Problem 3: Troubleshooting
The Wall Street Bank Anecdote

- And the **network 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. Whether or not the problem was in the network, the network team would be blamed since they were unable to demonstrate otherwise."

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

Problem 3: Troubleshooting

The Wall Street Bank Anecdote

- And the **network 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. Whether or not the problem was in the network, the network team would be blamed since they were unable to demonstrate otherwise."

The case for two new paradigms:
Software-defined networks and network virtualization.

Problem 3: Troubleshooting

The Wall Street Bank Anecdote

- And the **network 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
> ...eriments to identify, quarantine
> ...er or not the problem was in the
> ...d be
> ...e."

Decoupling and consolidating the control plane and making the network **programmable**: enables innovations (design your own routing algorithm!) as well as automatic, formal verification.

Provide **isolation**: logical isolation (e.g., between different tenants) and in terms of performance. Allow different network stacks to co-exist.

The case for two new pa...gms:
Software-defined networks and network virtualization.

Agenda Today:

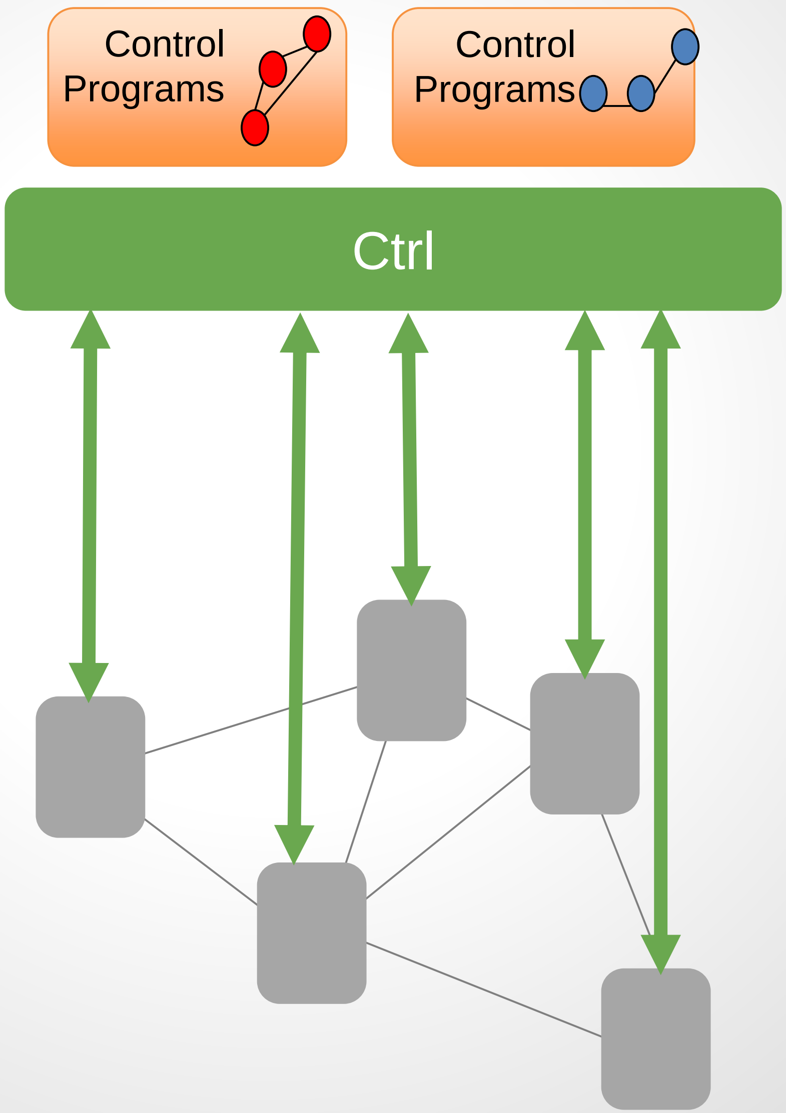
Challenges in software-defined and virtualized networks

However, these new paradigms also come with new challenges:

- ❑ **Challenge 1:** Correctly operating software-defined networks is non-trivial and poses interesting algorithmic problems

- ❑ **Challenge 2:** Software-defined and virtualized networks do not only offer interesting new security solutions, but also introduce new security issues. In particular, we discuss a new threat vector: the insecure data plane.

A Mental Model for SDNs



In a nutshell:

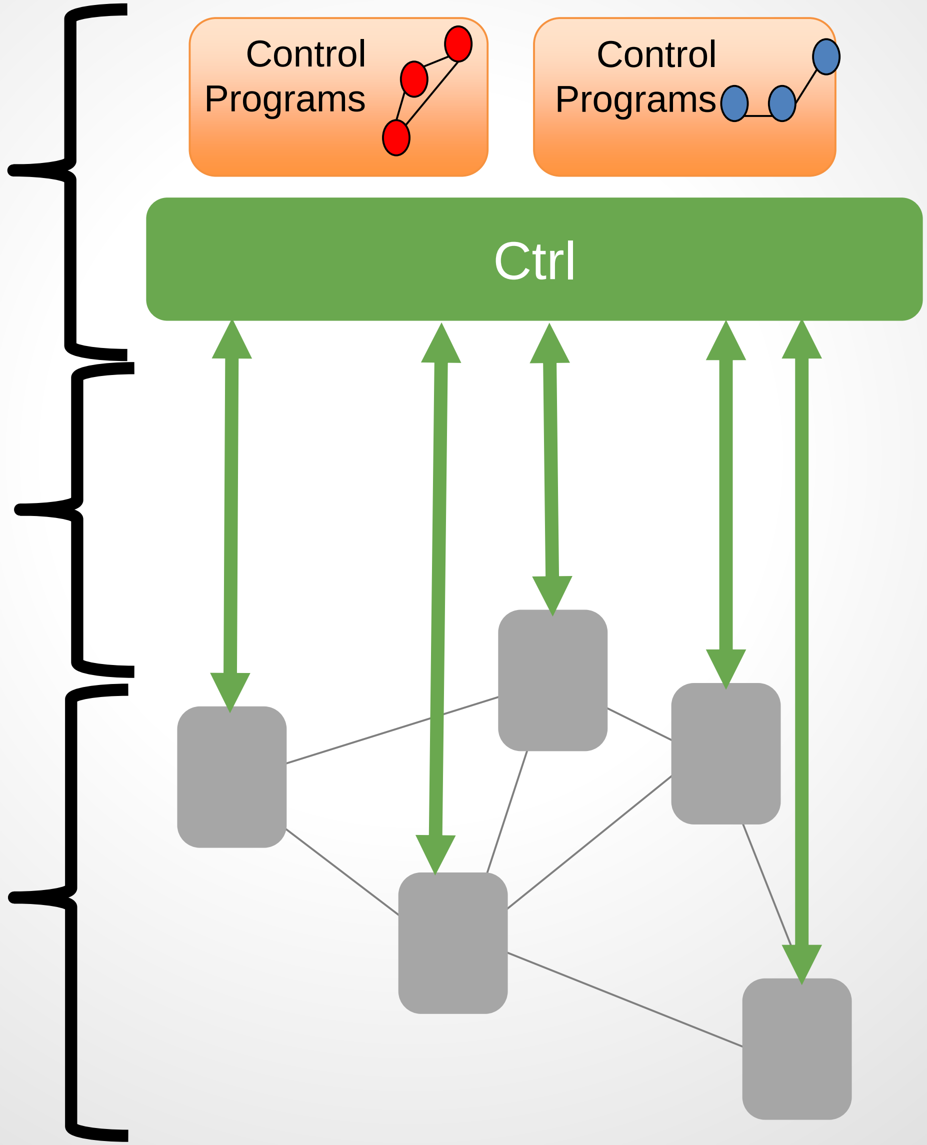
SDN outsources and consolidates control over multiple devices to (logically) centralized software controller

Algorithmic Problems in SDNs

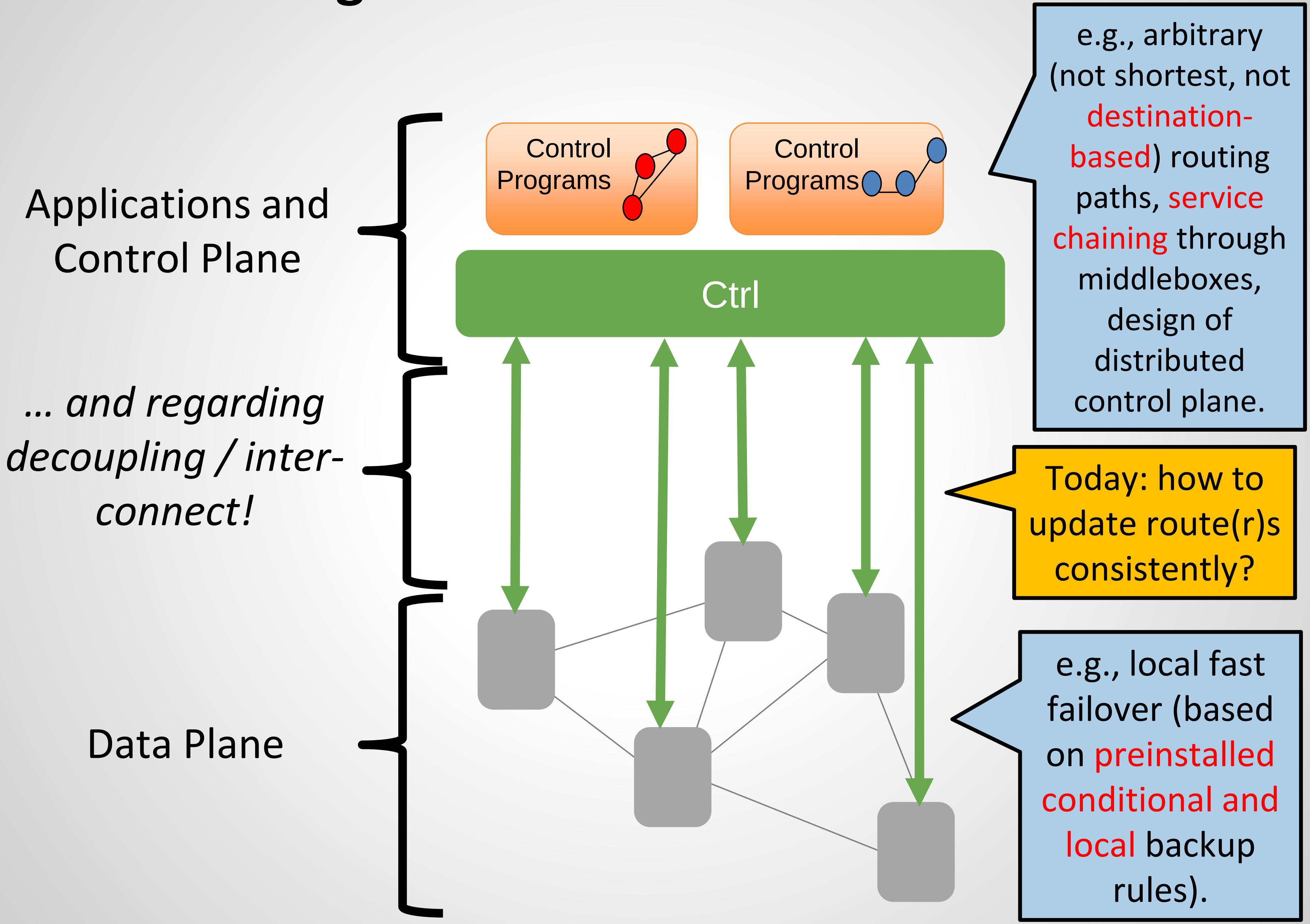
Applications and Control Plane

… and regarding decoupling / inter-connect!

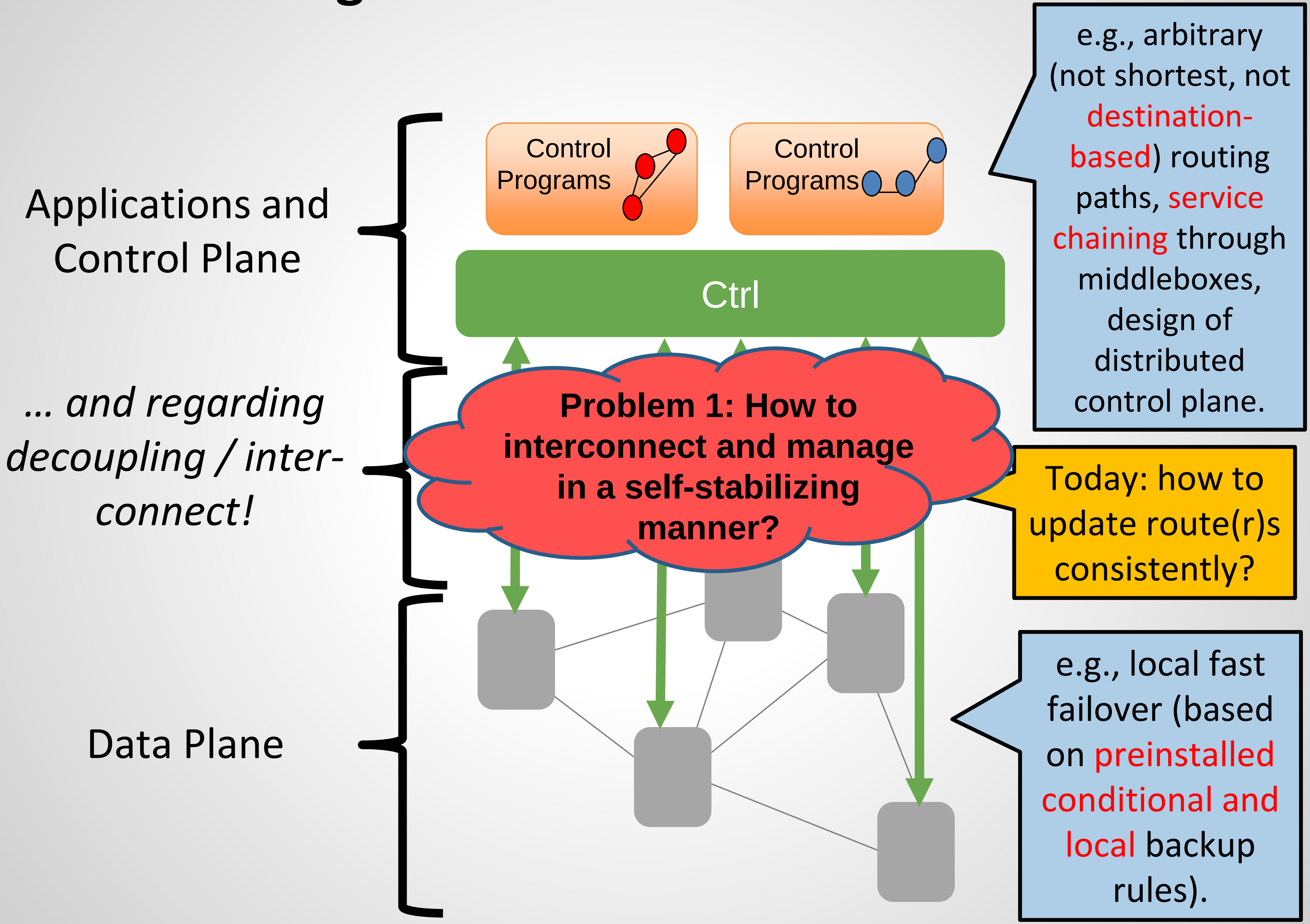
Data Plane



Algorithmic Problems in SDNs



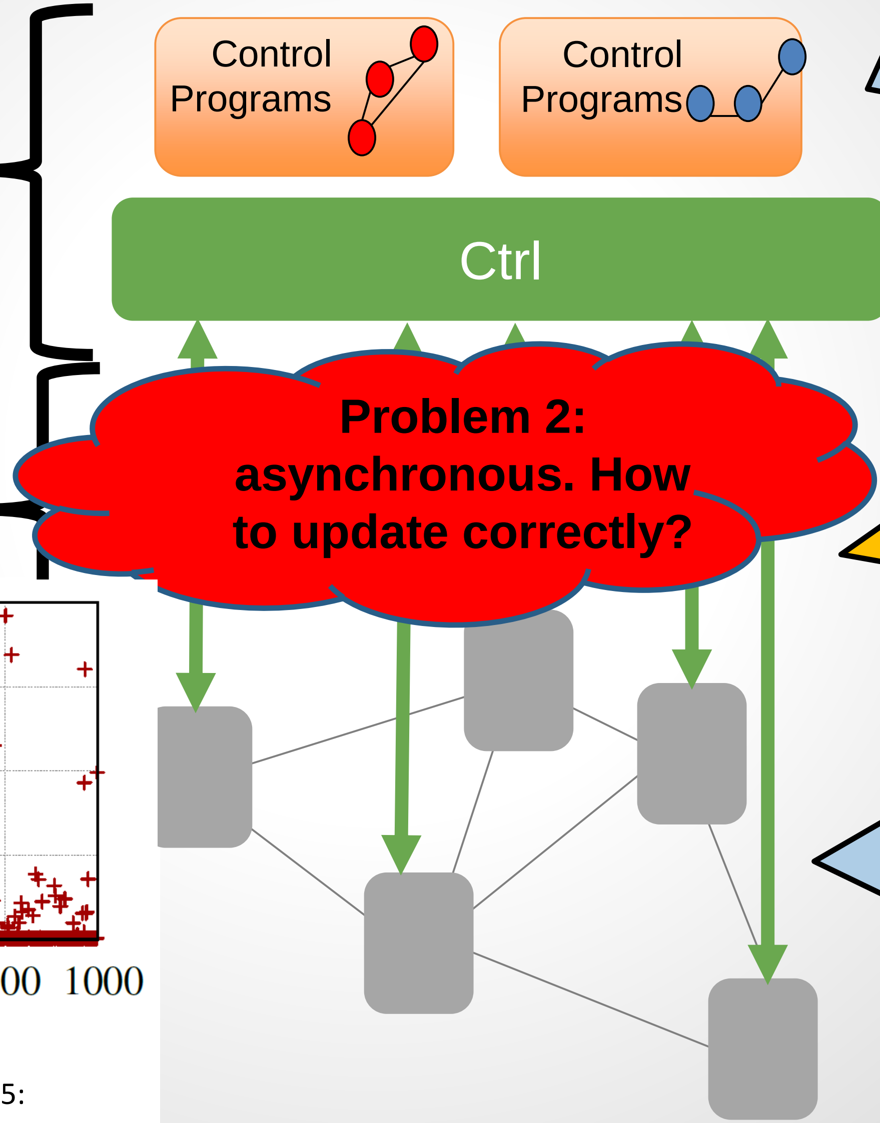
Algorithmic Problems in SDNs

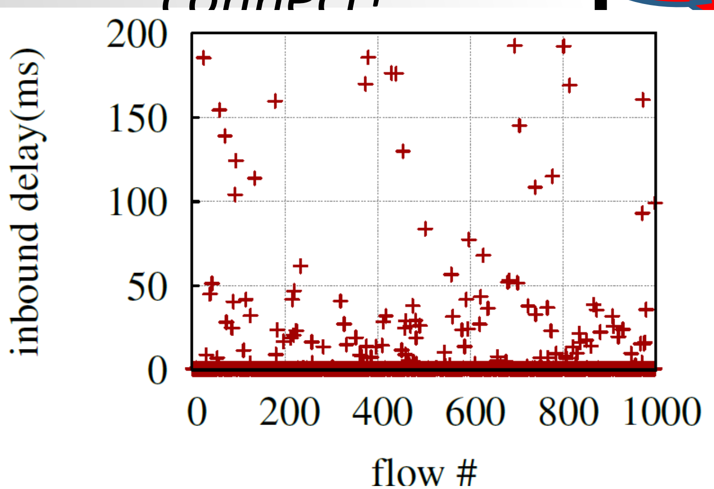


Algorithmic Problems in SDNs

Applications and Control Plane

... and regarding decoupling / inter-connect!





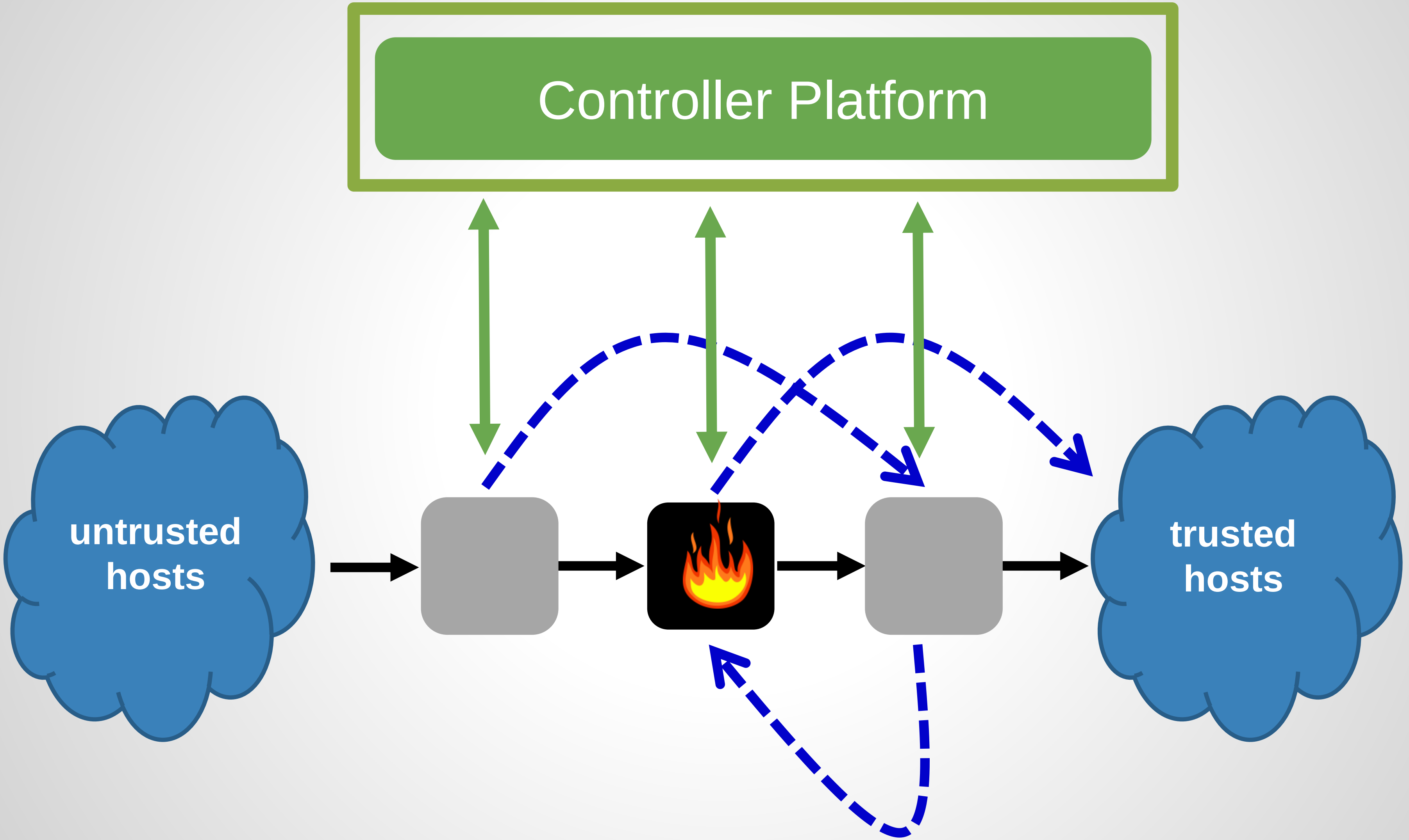
He et al., ACM SOSR 2015: without network latency

e.g., arbitrary (not shortest, not destination-based) routing paths, service chaining through middleboxes, design of distributed control plane.

Today: how to update route(r)s consistently?

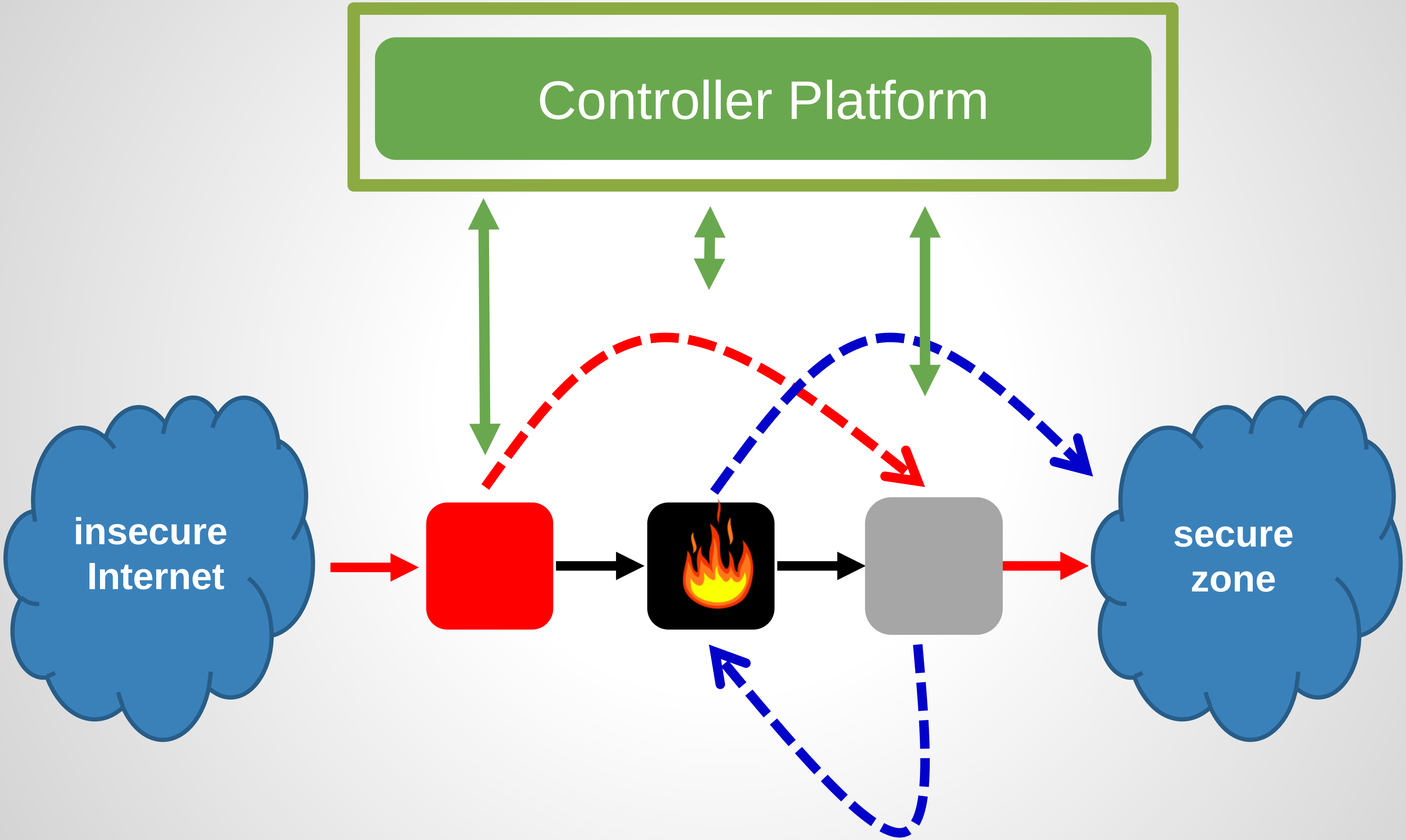
e.g., local fast failover (based on preinstalled conditional and local backup rules).

What can possibly go wrong?



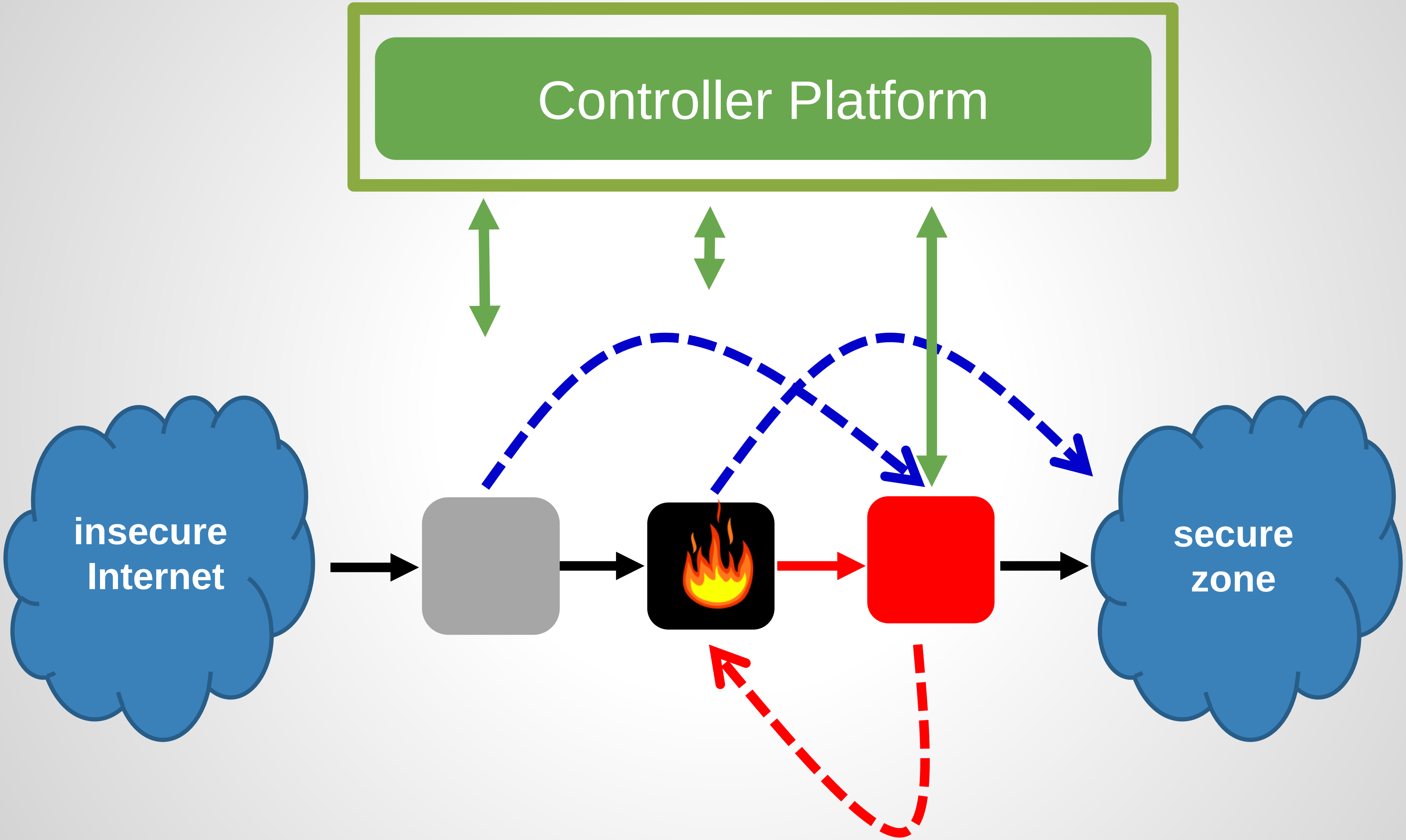
Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

Example 1: Bypassed Waypoint



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

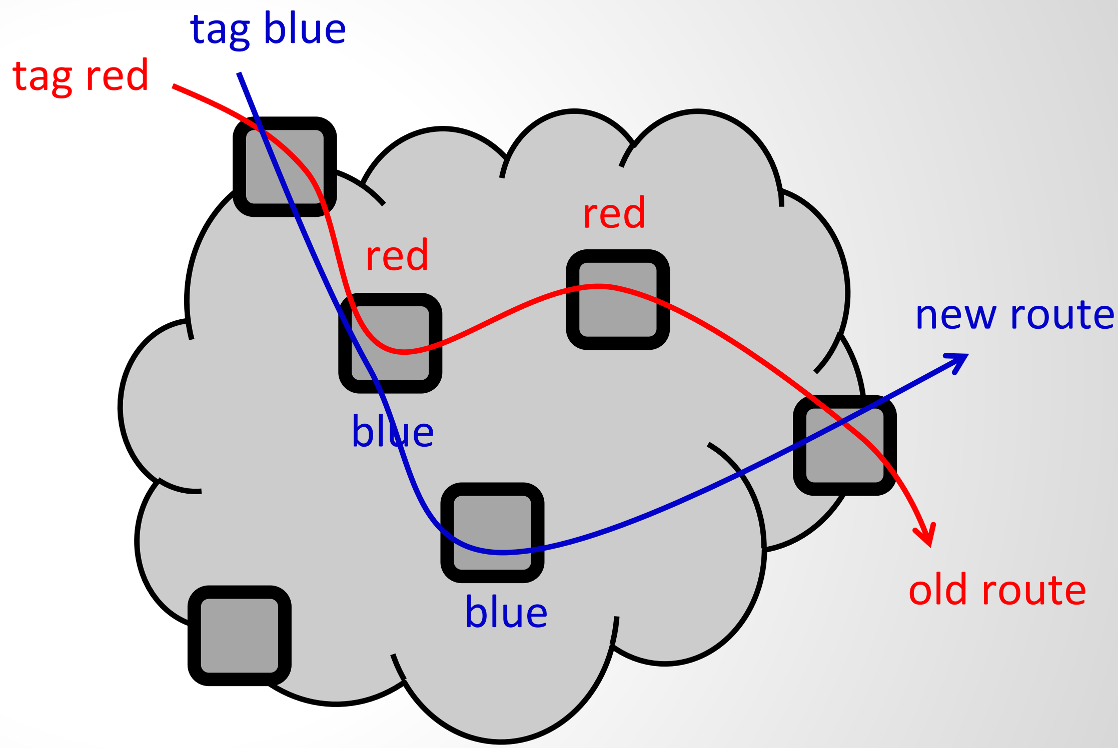
Example 2: *Transient* Loop



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



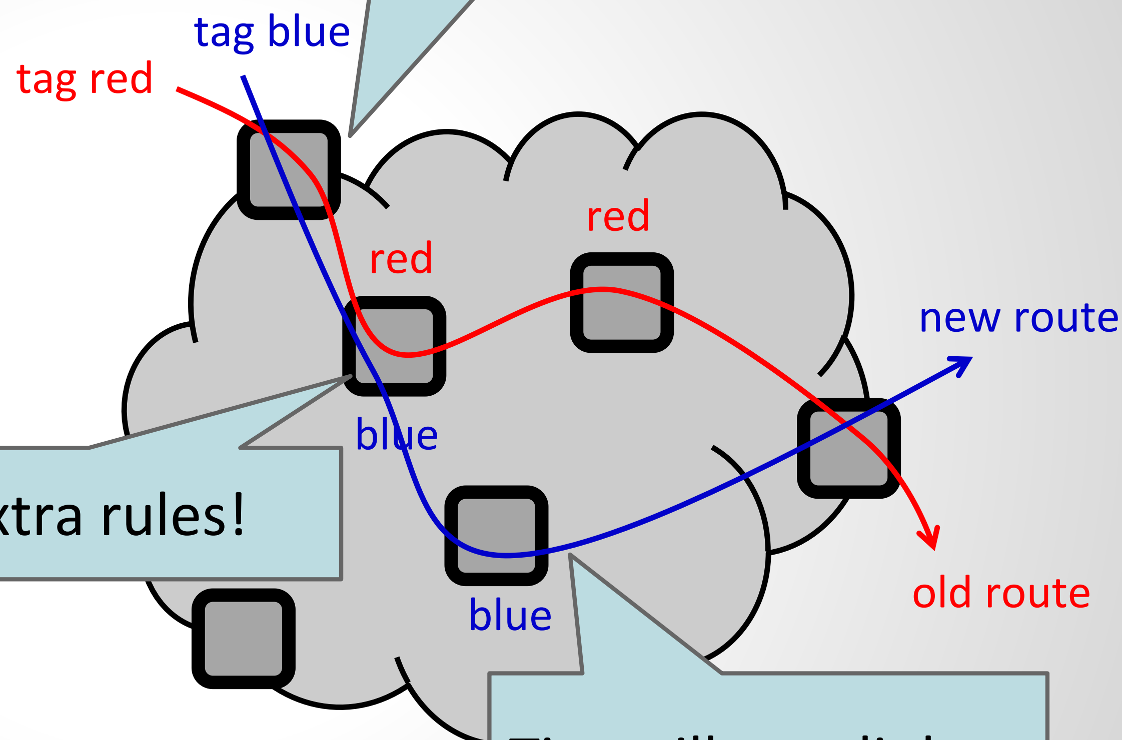
Reitblatt et al. Abstractions for Network Update, ACM SIGCOMM 2012.

Tagging: A Univers

Where to tag?
Header space?
Overhead!

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

Cost of extra rules!

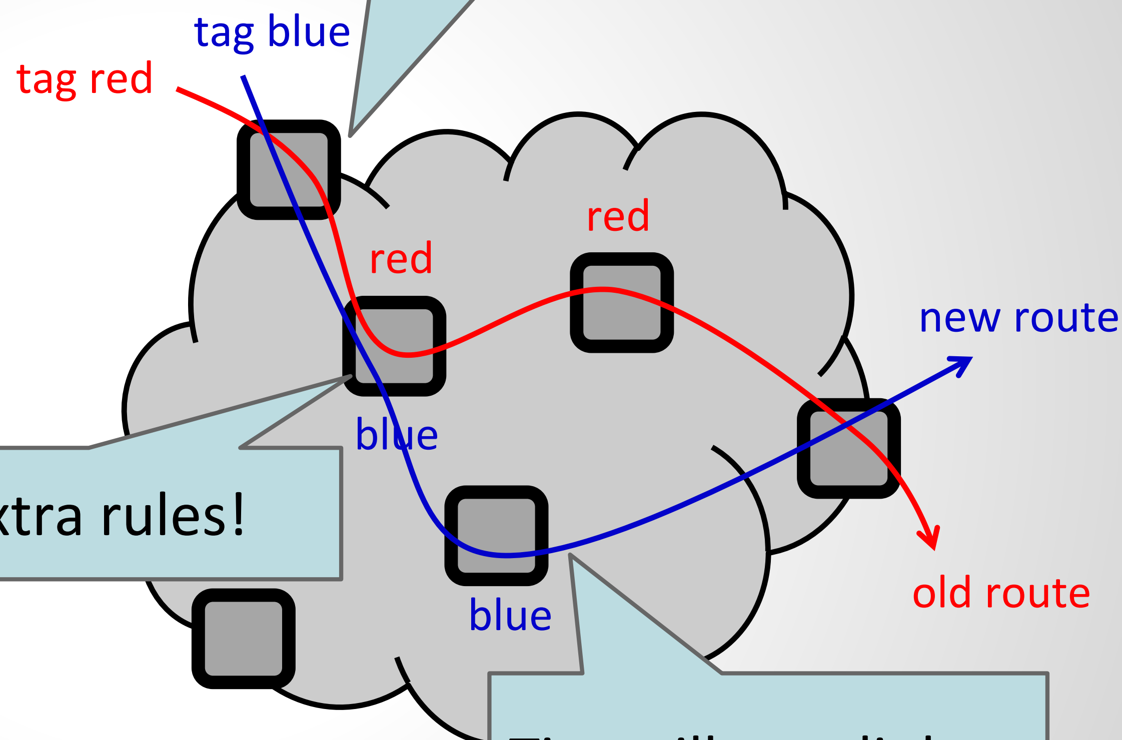


Time till new link becomes available!

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



Where to tag?
Header space?
Overhead!

Cost of extra rules!

Time till new link becomes available!

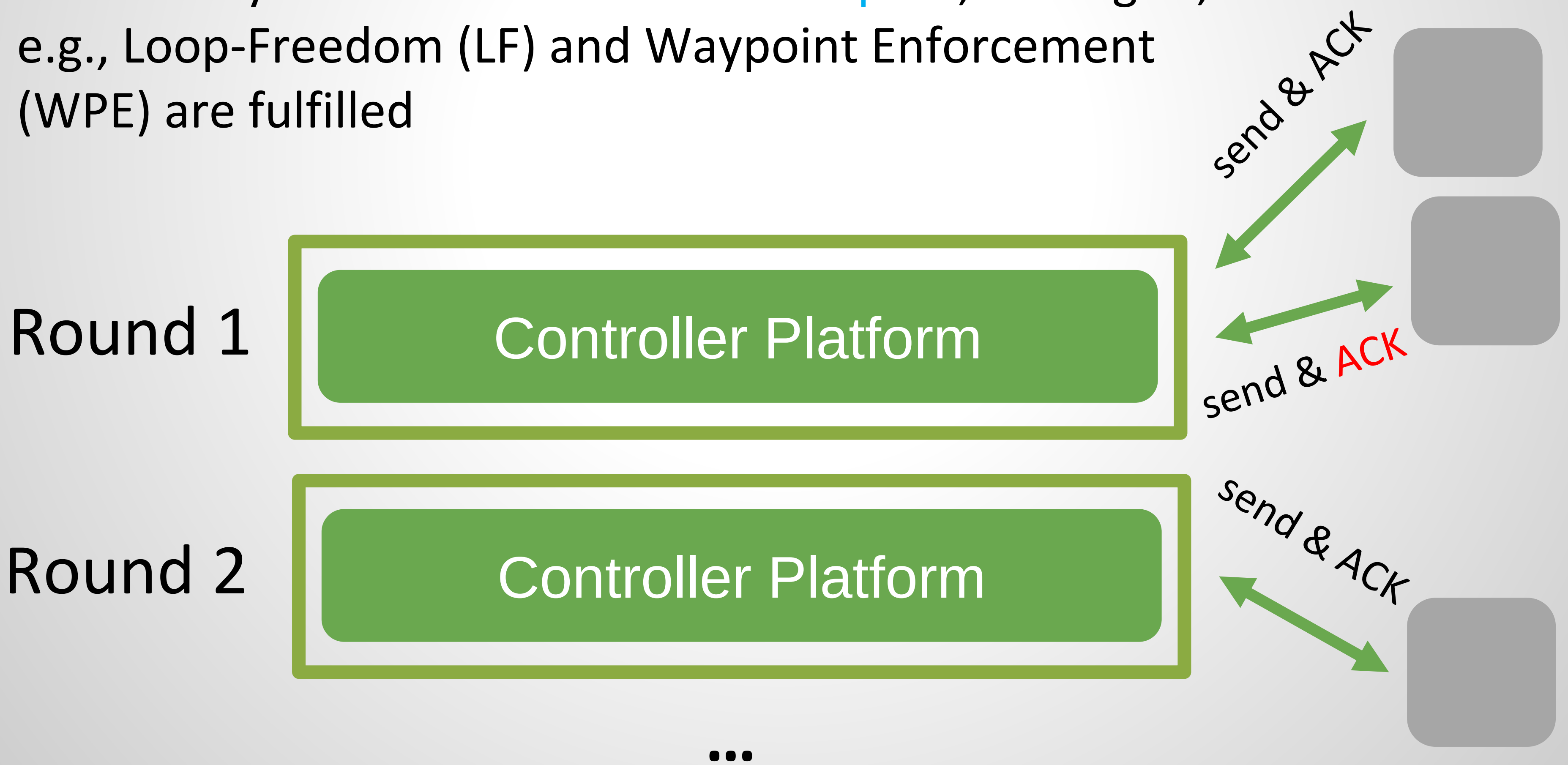
Can we do without tagging, and at least preserve weaker consistency properties?

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

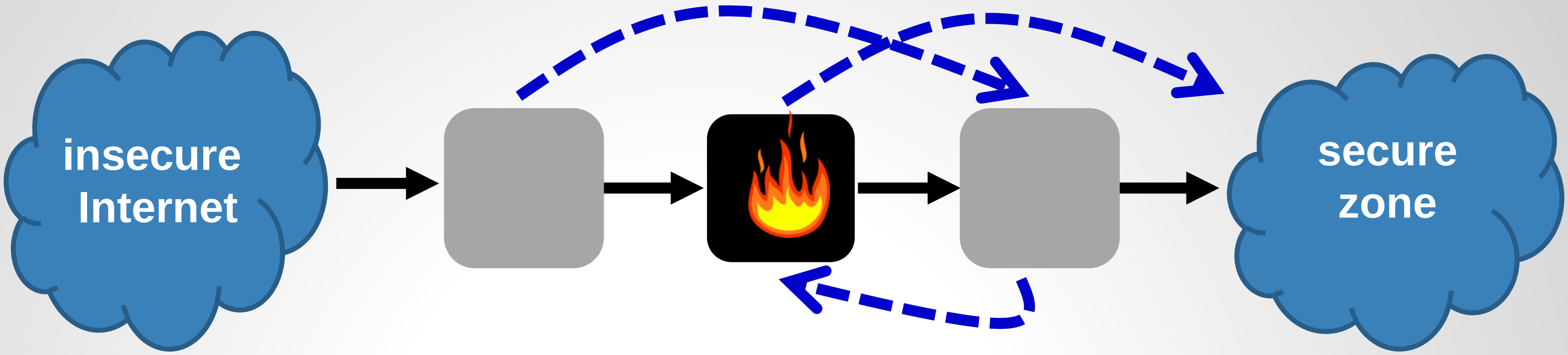
Idea: Schedule Subsets of Nodes!

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

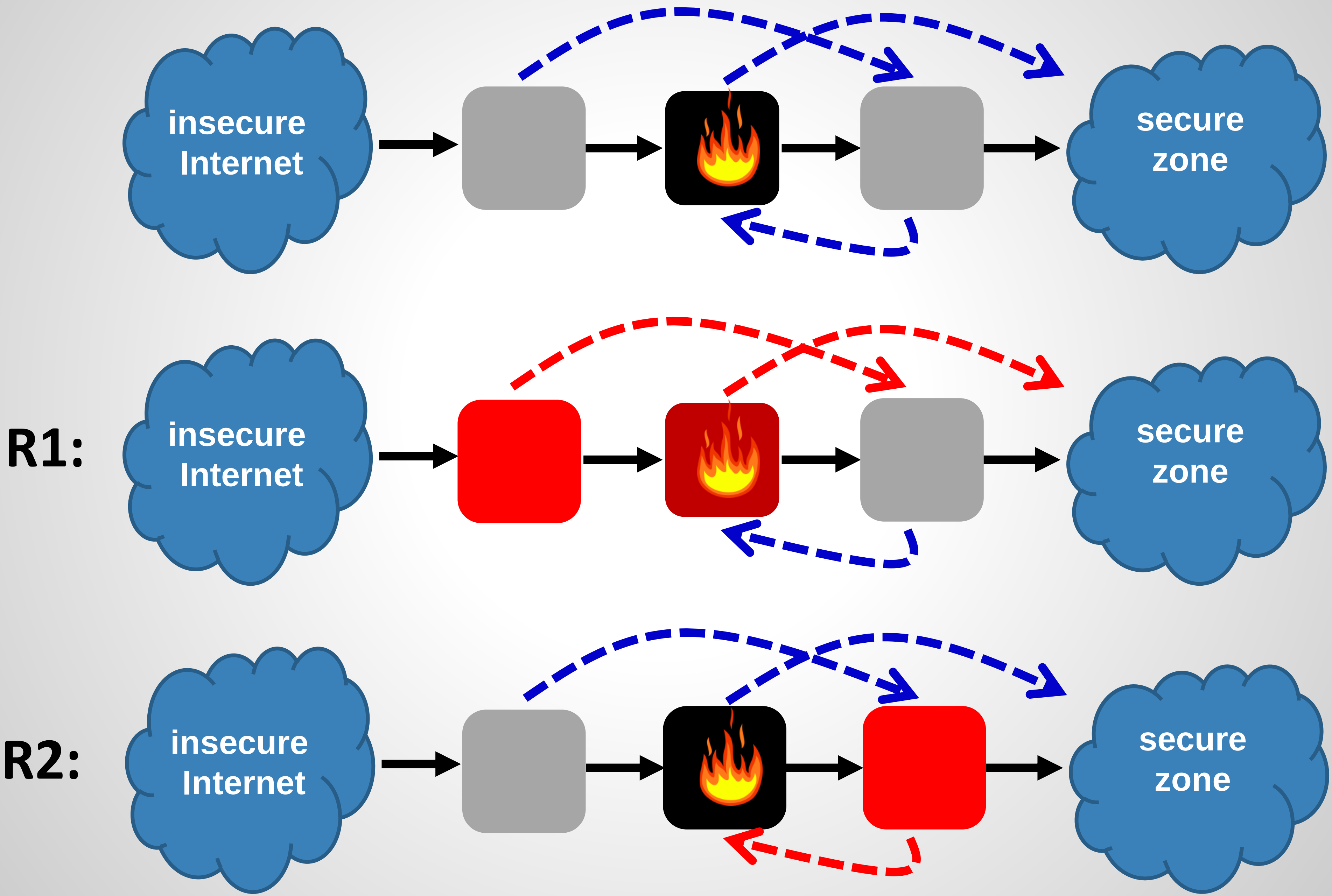
Packet may take a mix of old and new path, as long as, e.g., Loop-Freedom (LF) and Waypoint Enforcement (WPE) are fulfilled



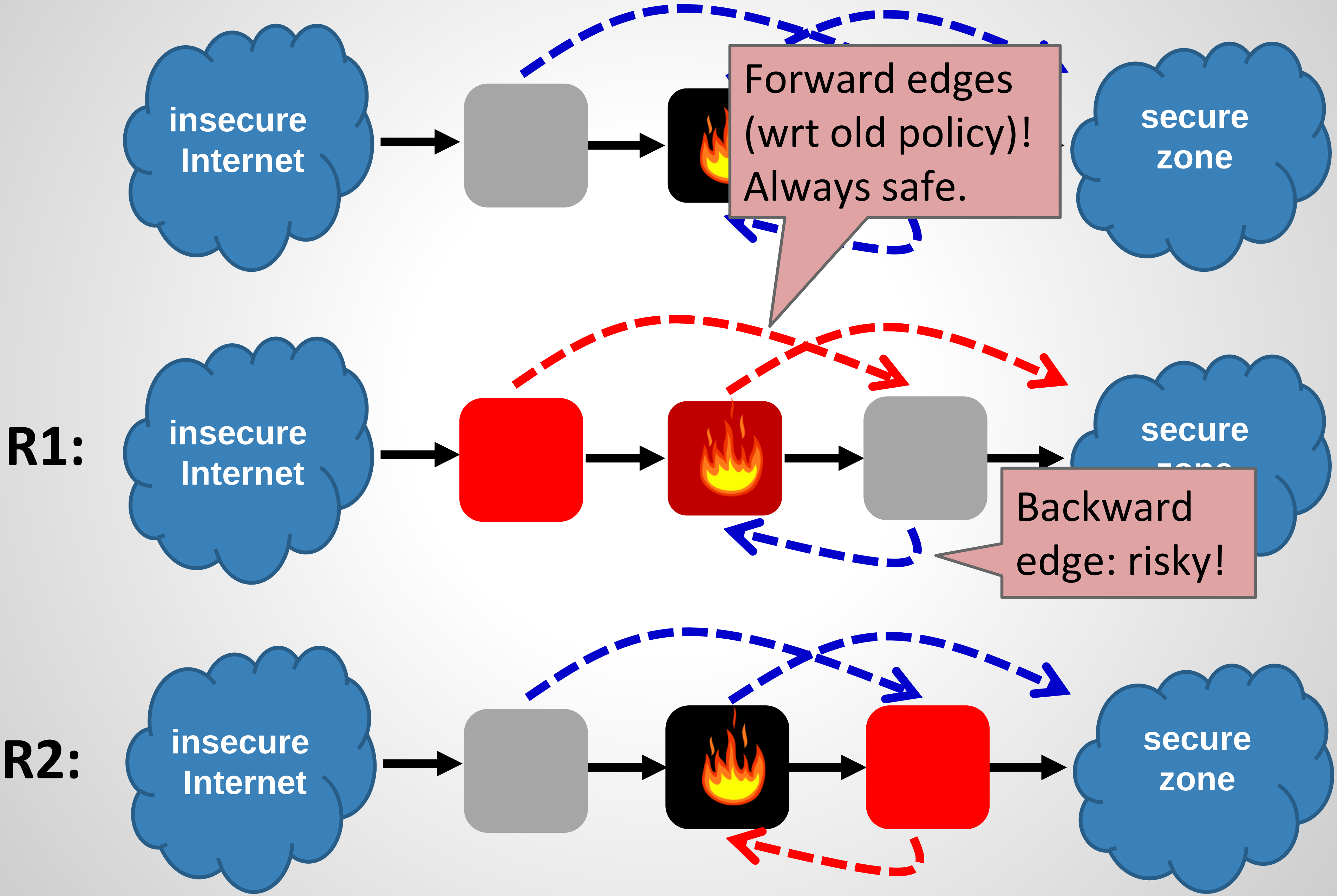
Going Back to Our Examples: LF Update



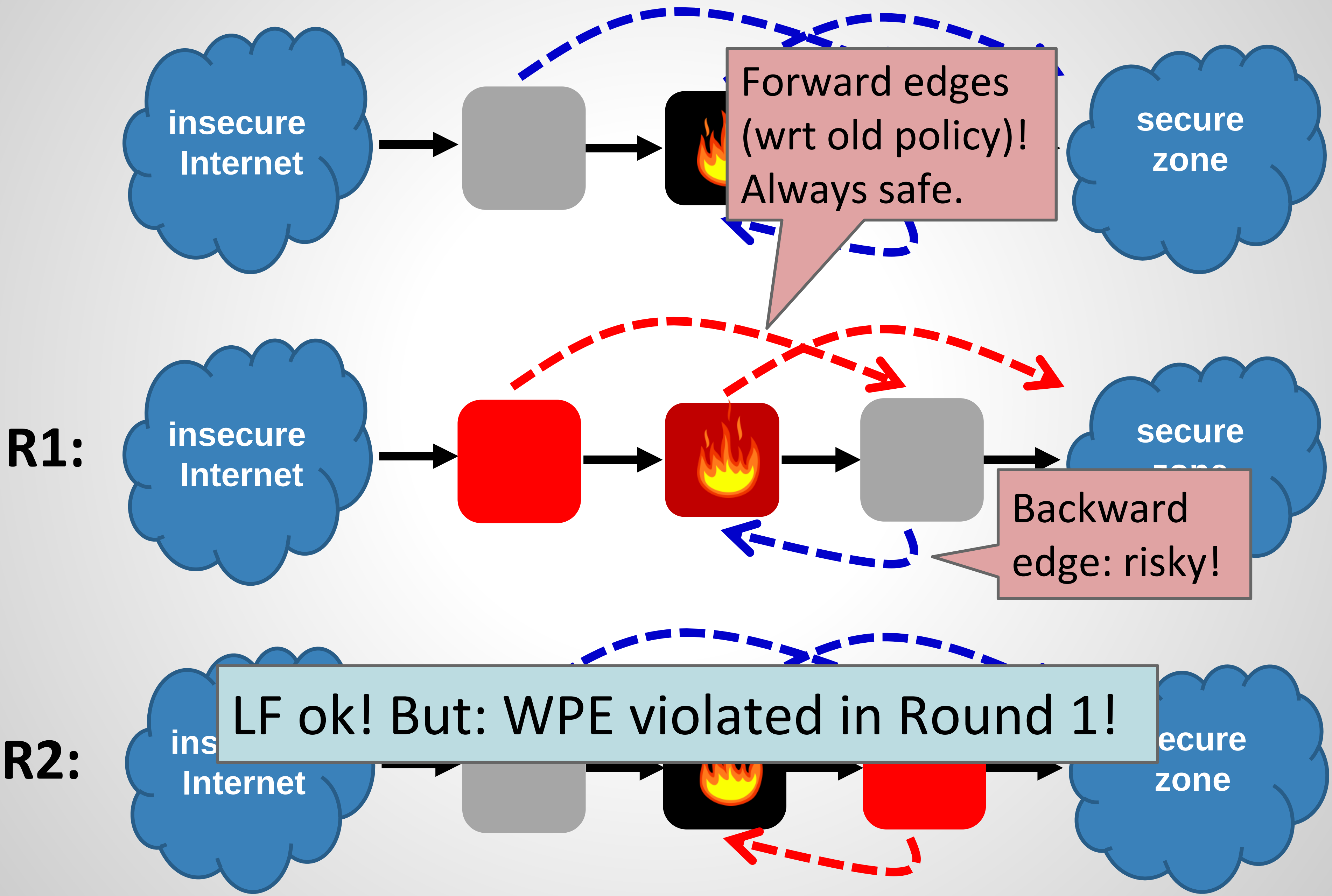
Going Back to Our Examples: LF Update



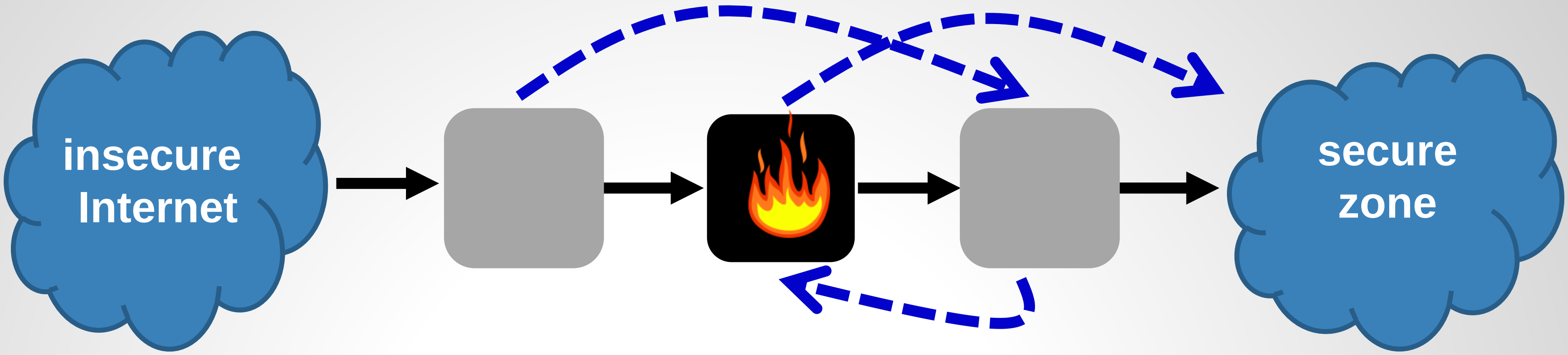
Going Back to Our Examples: LF Update



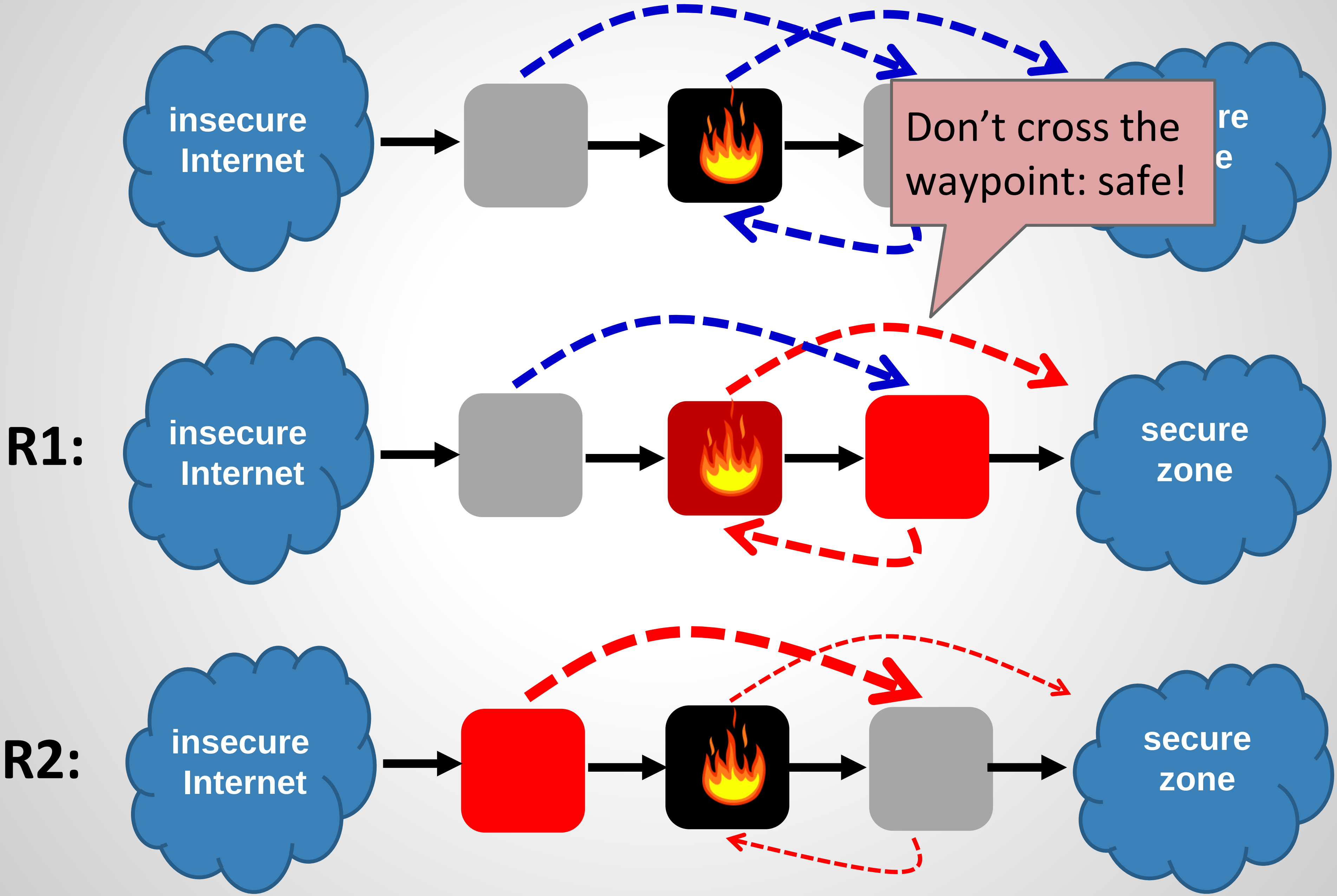
Going Back to Our Examples: LF Update



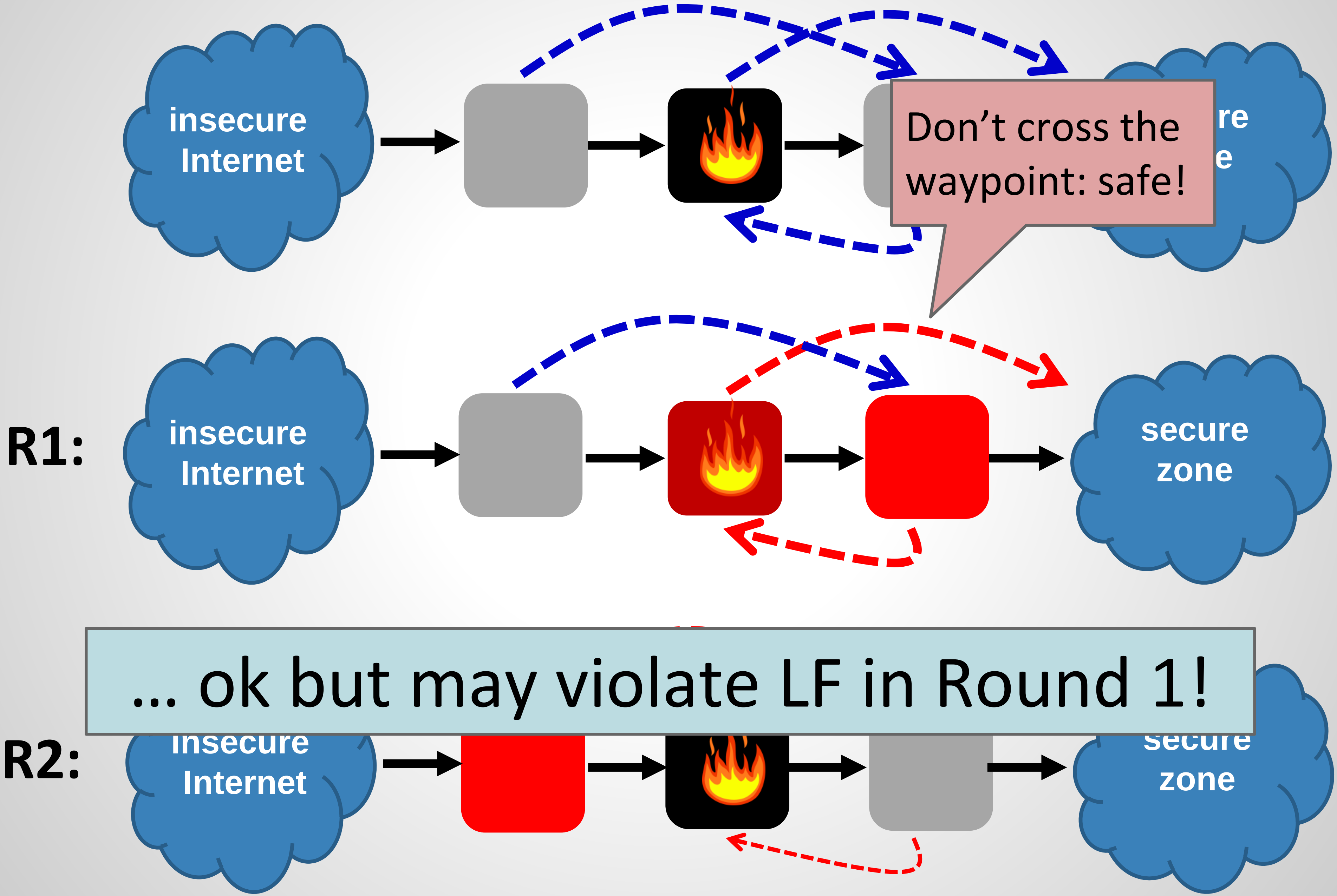
Going Back to Our Examples: WPE Update



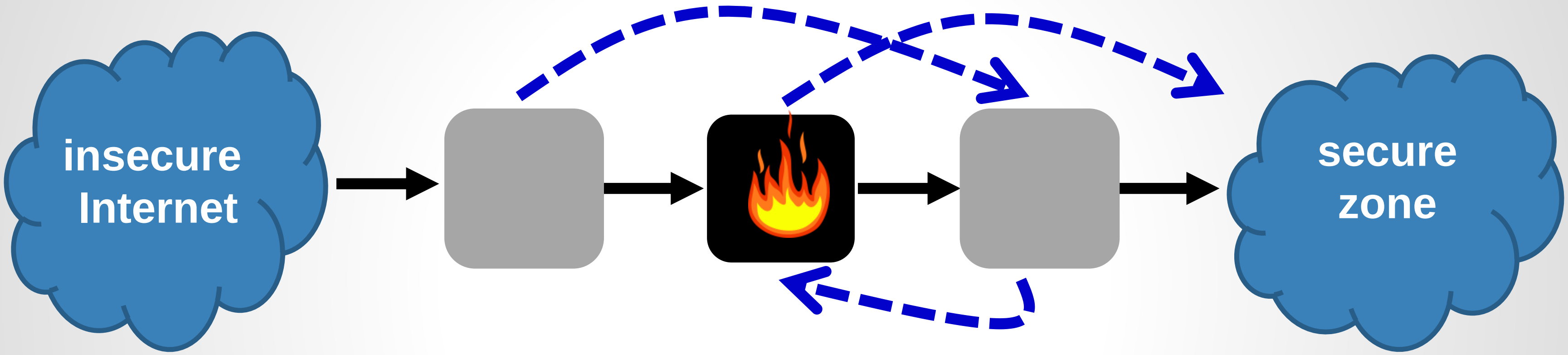
Going Back to Our Examples: WPE Update



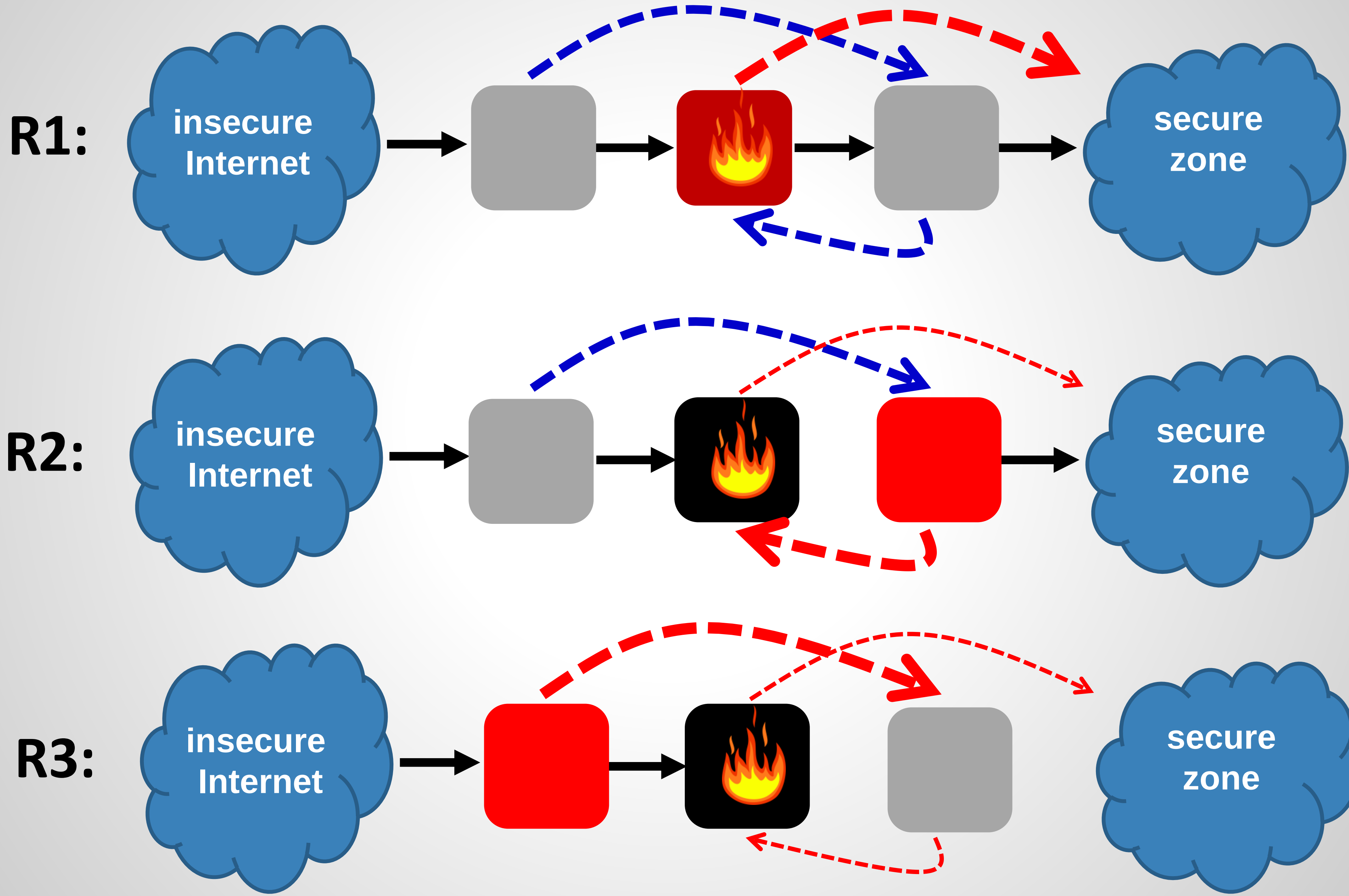
Going Back to Our Examples: WPE Update



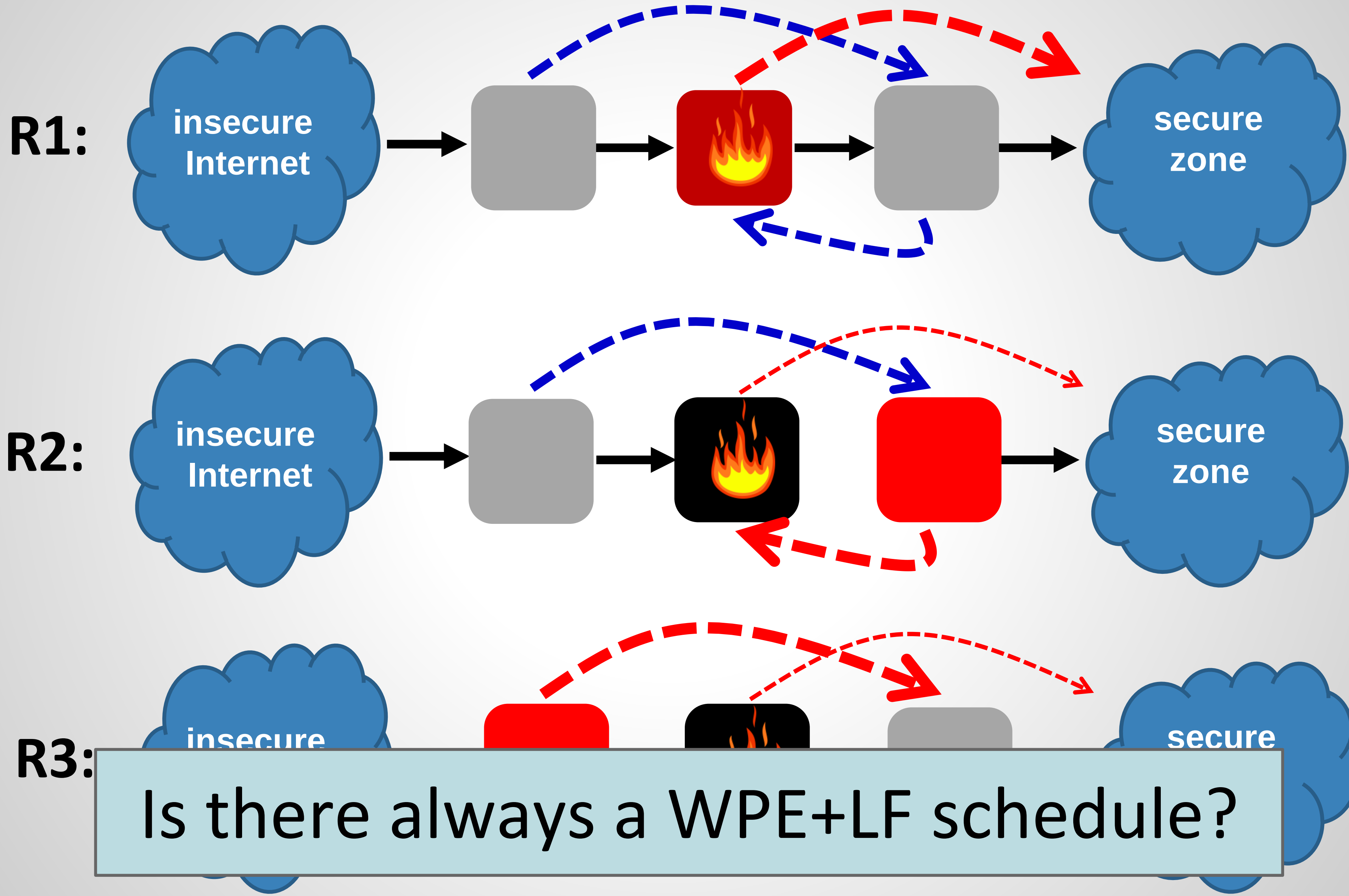
Going Back to Our Examples: Both WPE+LF?



Going Back to Our Examples: WPE+LF!

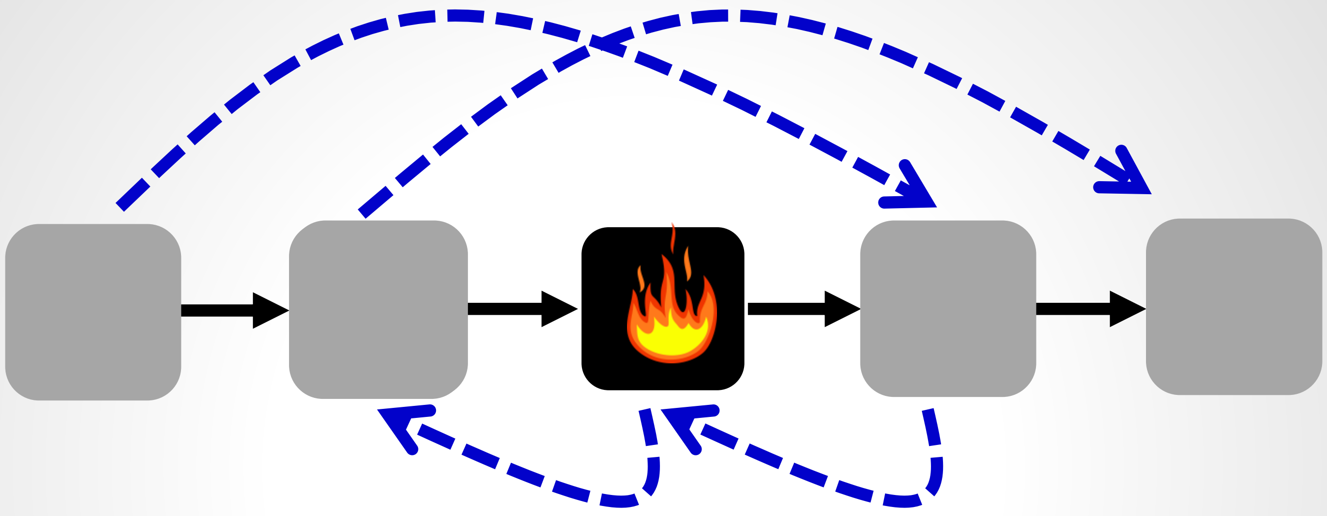


Going Back to Our Examples: WPE+LF!

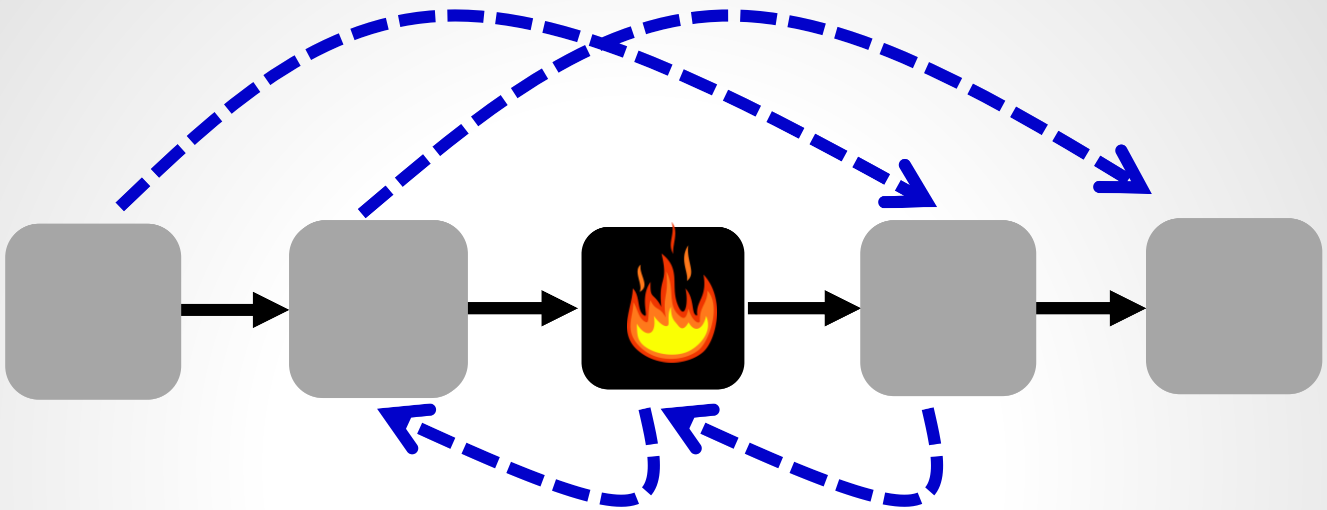


Is there always a WPE+LF schedule?

What about this one?



LF and WPE may conflict!

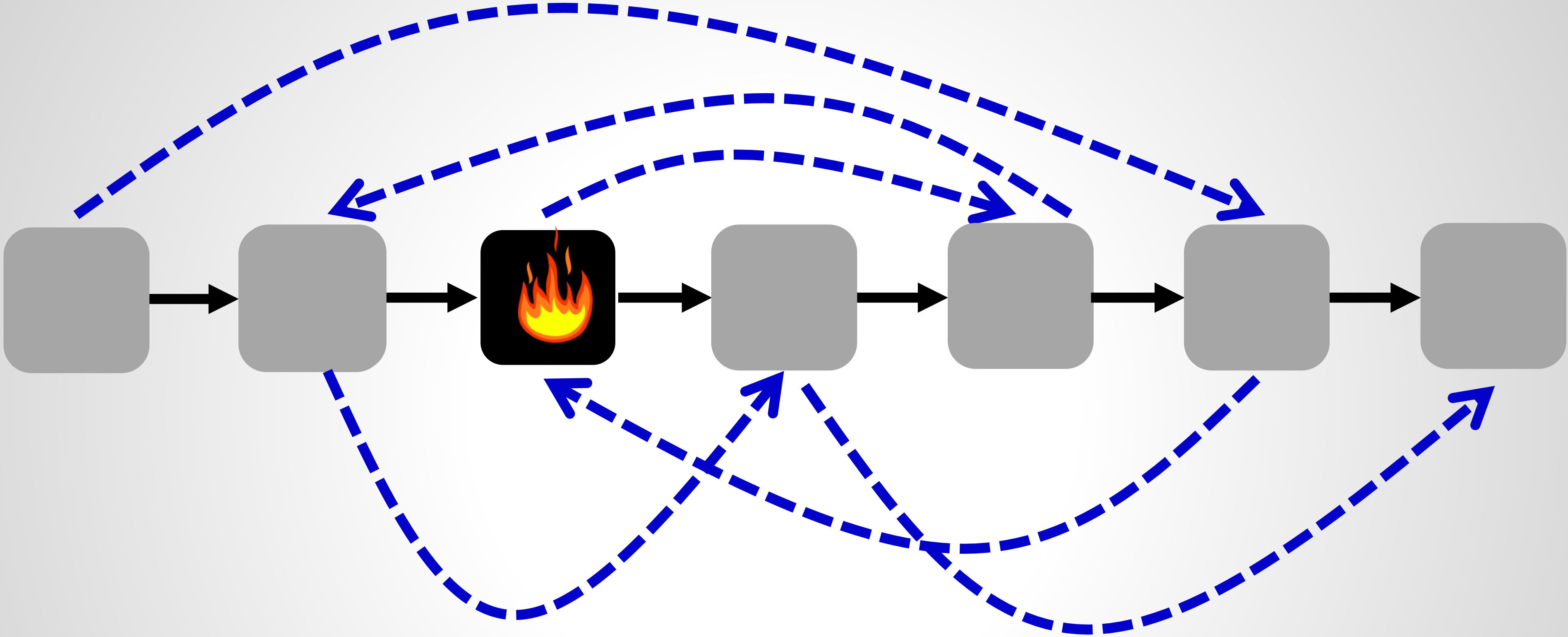


- ❑ Cannot update any **forward edge** in R1: WP
- ❑ Cannot update any **backward edge** in R1: LF

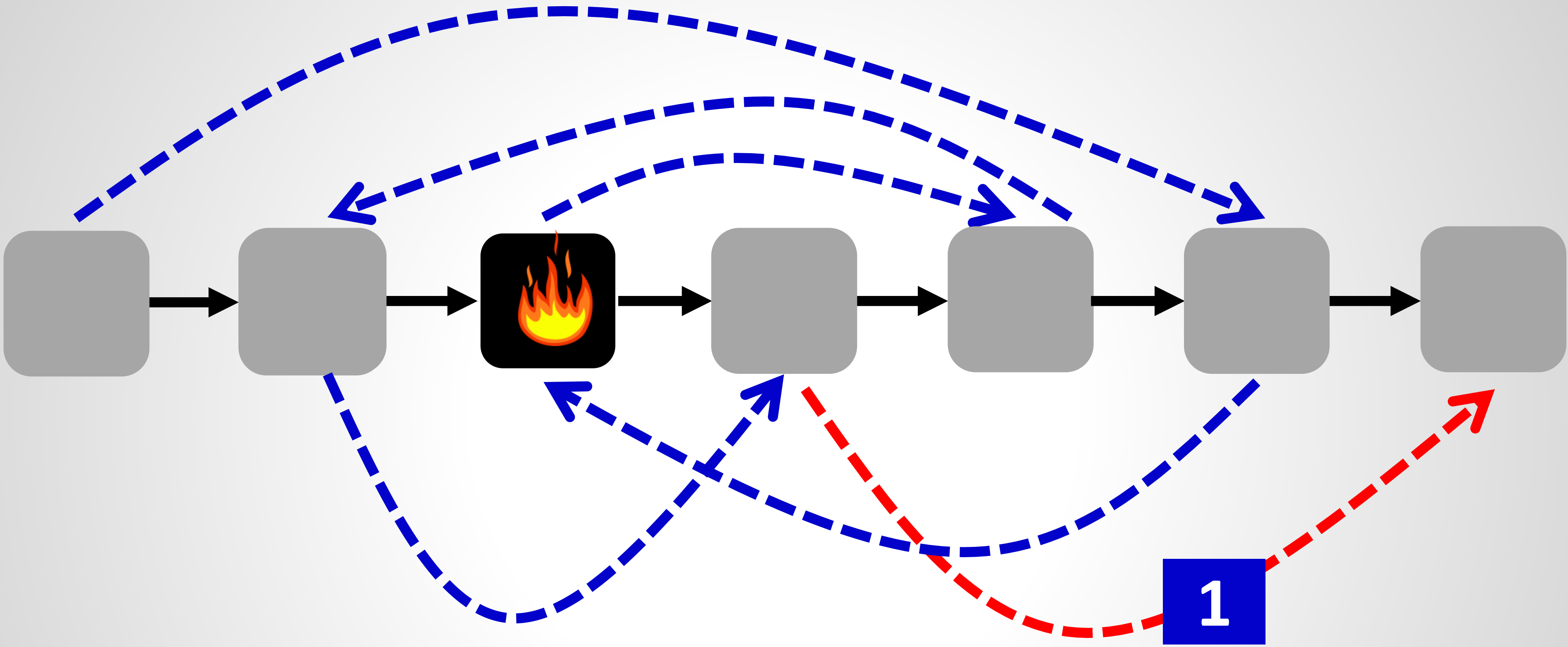
No schedule exists!

Resort to tagging…

What about this one?

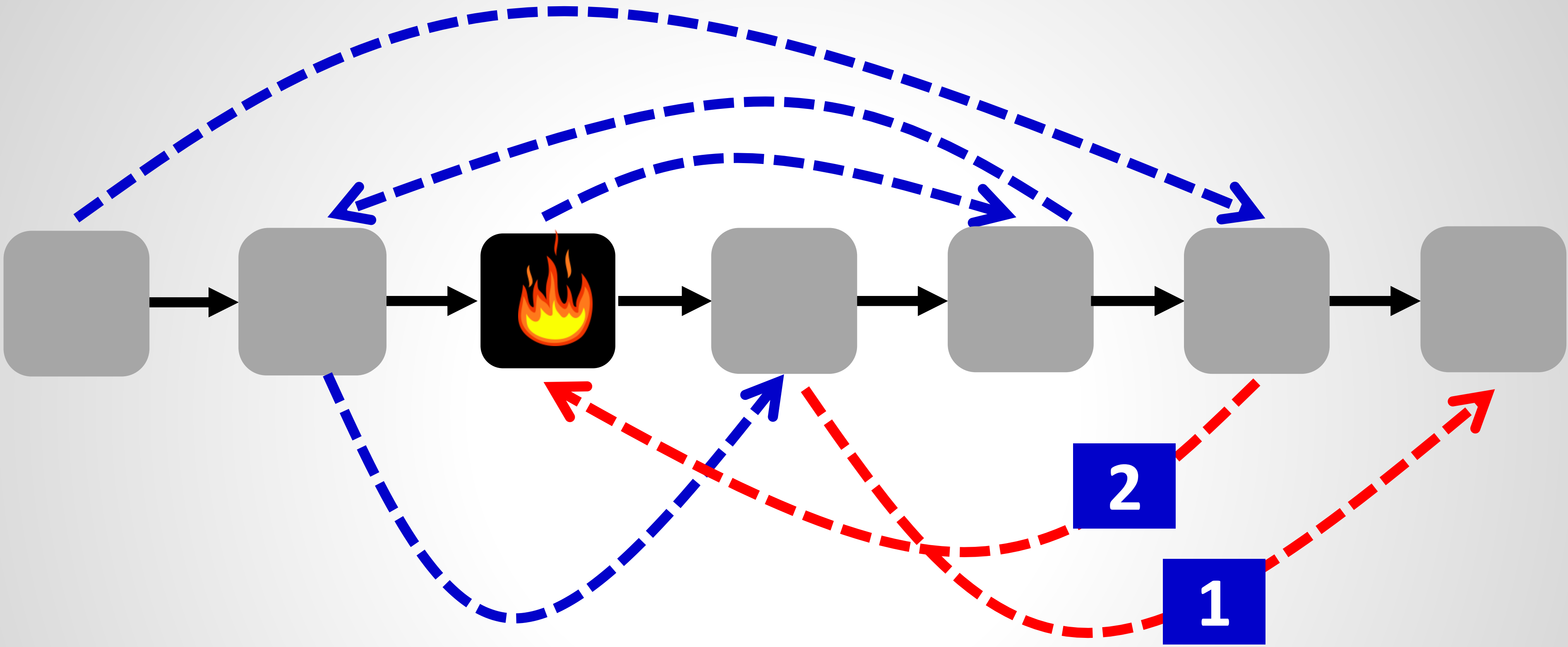


What about this one?



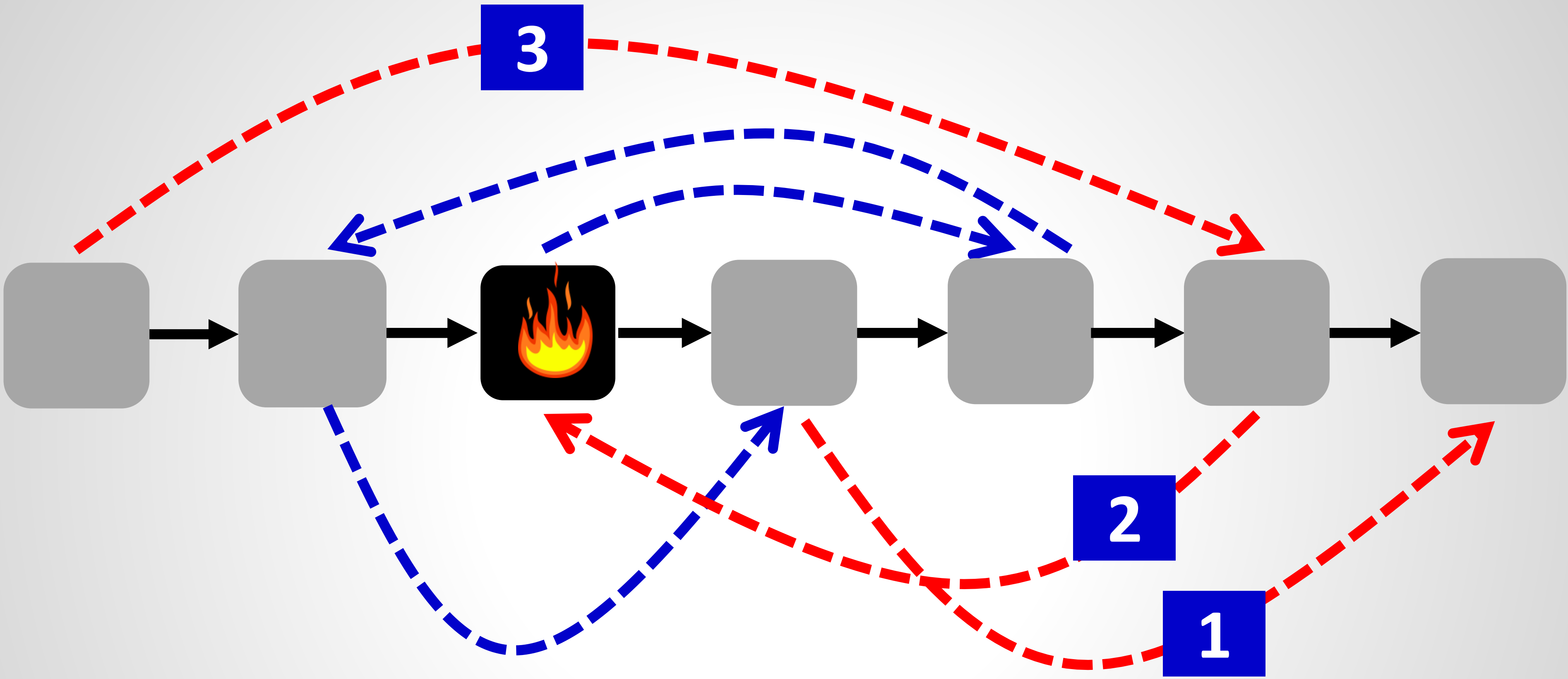
- Forward edge after the waypoint: safe!
  - No loop, no WPE violation

What about this one?



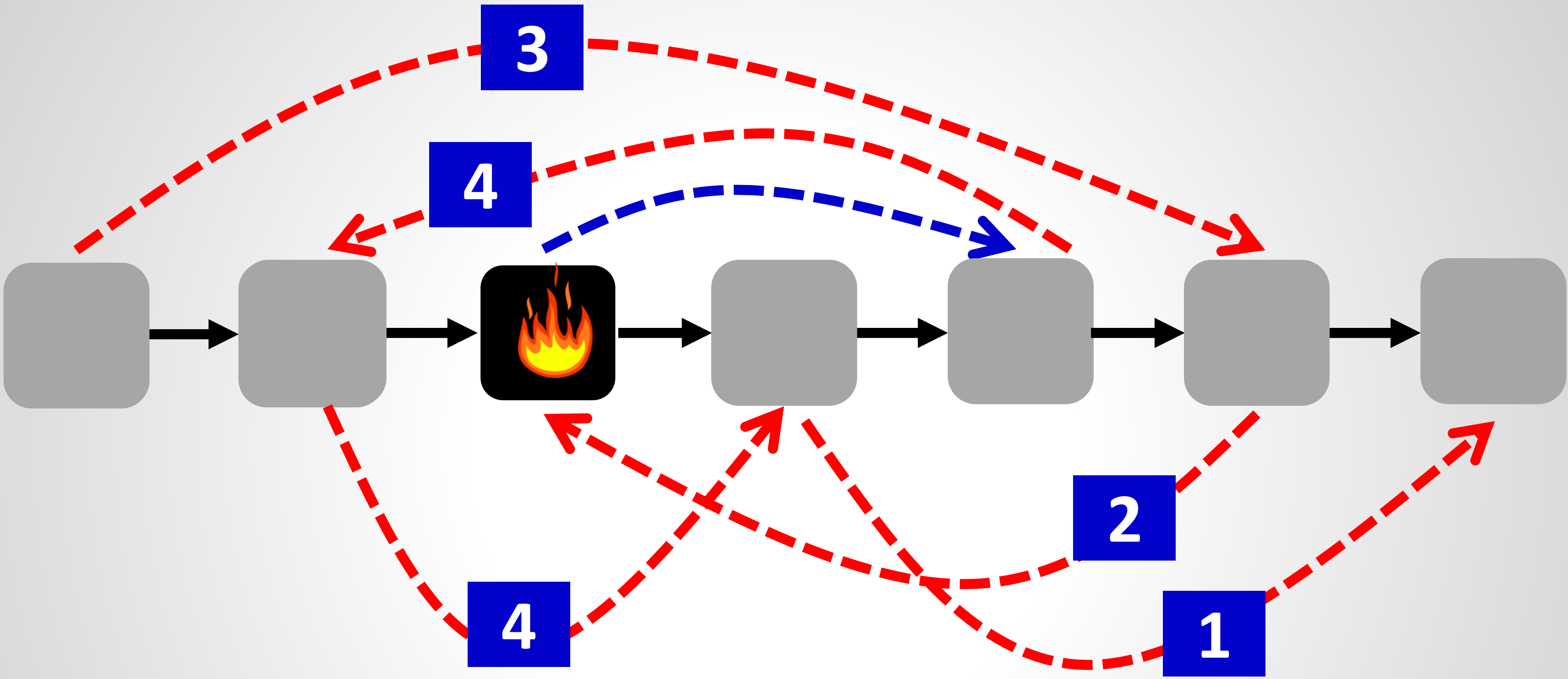
- Now this backward is safe too!
  - No loop because exit through 1

What about this one?



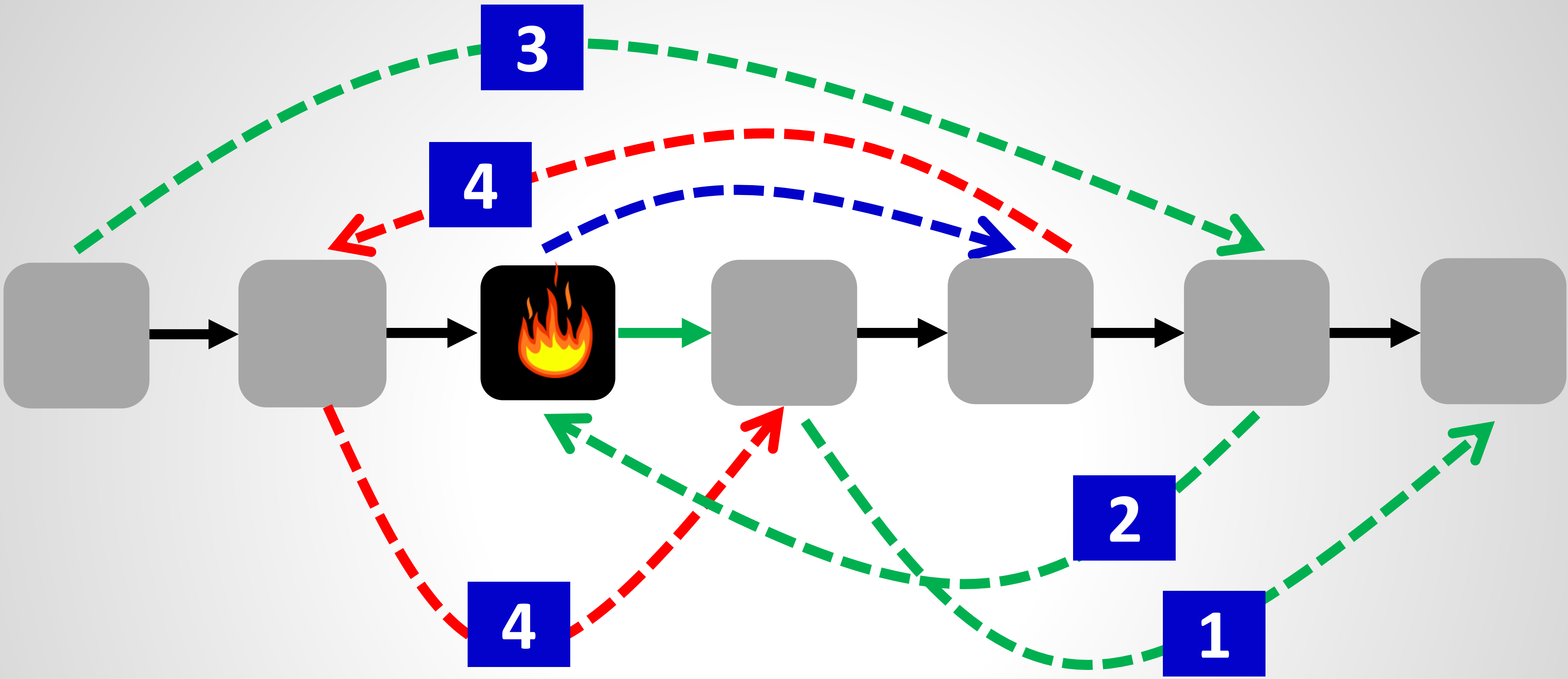
- Now this is safe: 2 ready back to WP!
  - No waypoint violation

What about this one?



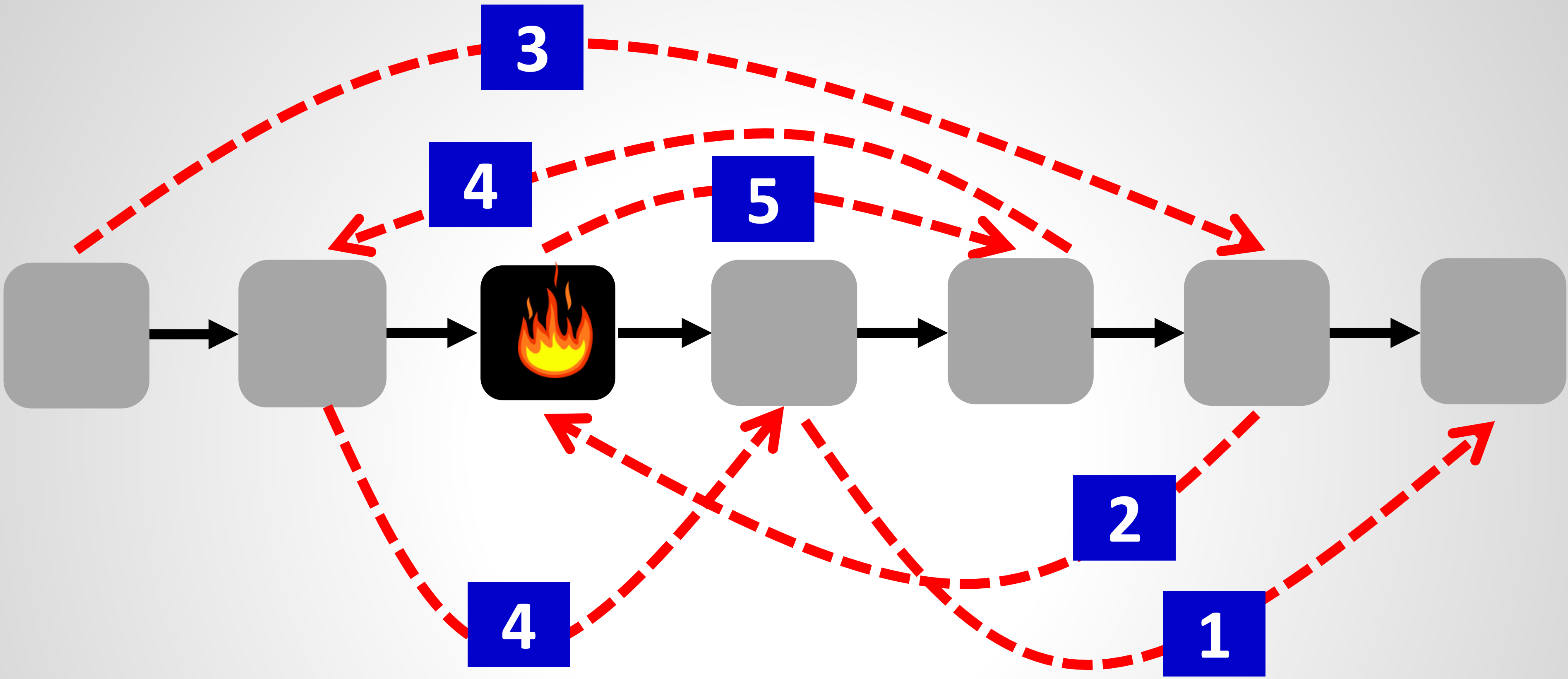
- Ok: loop-free and also not on the path (exit via 1)

What about this one?

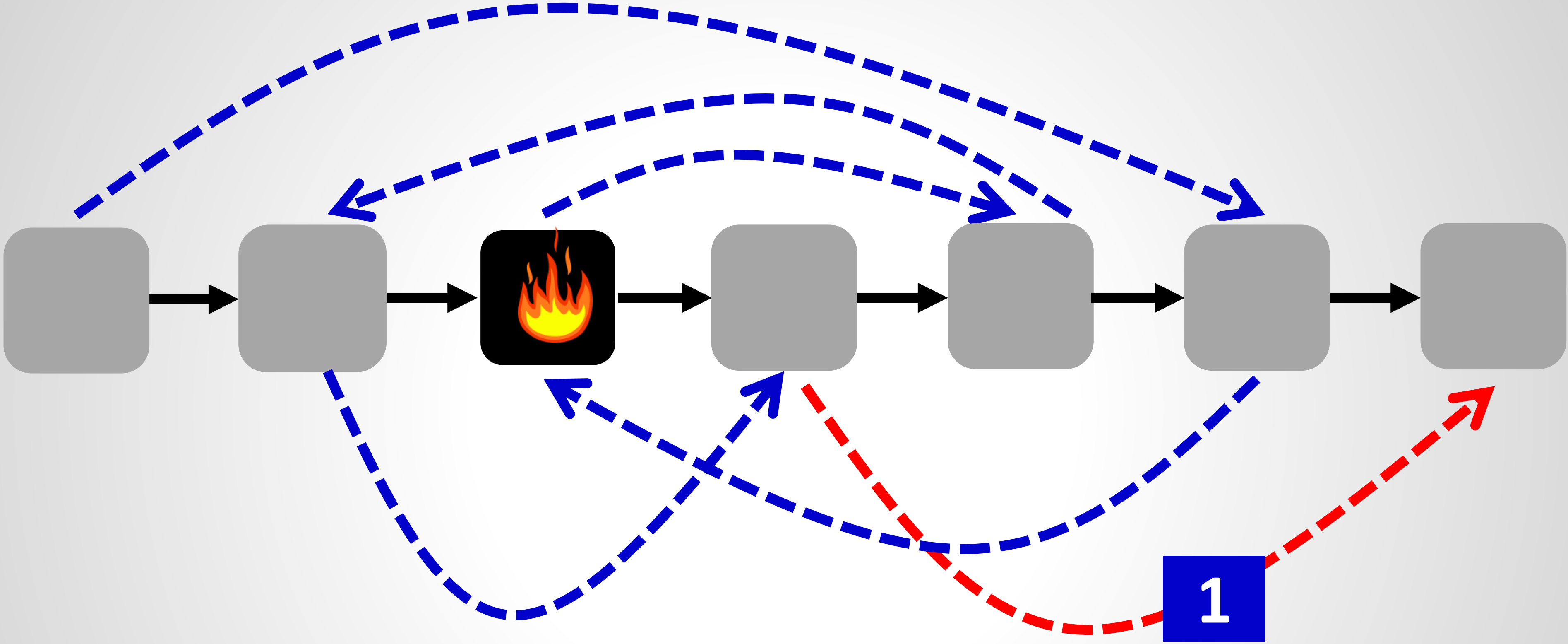


- ❑ Ok: loop-free and also not on the path (exit via 1)

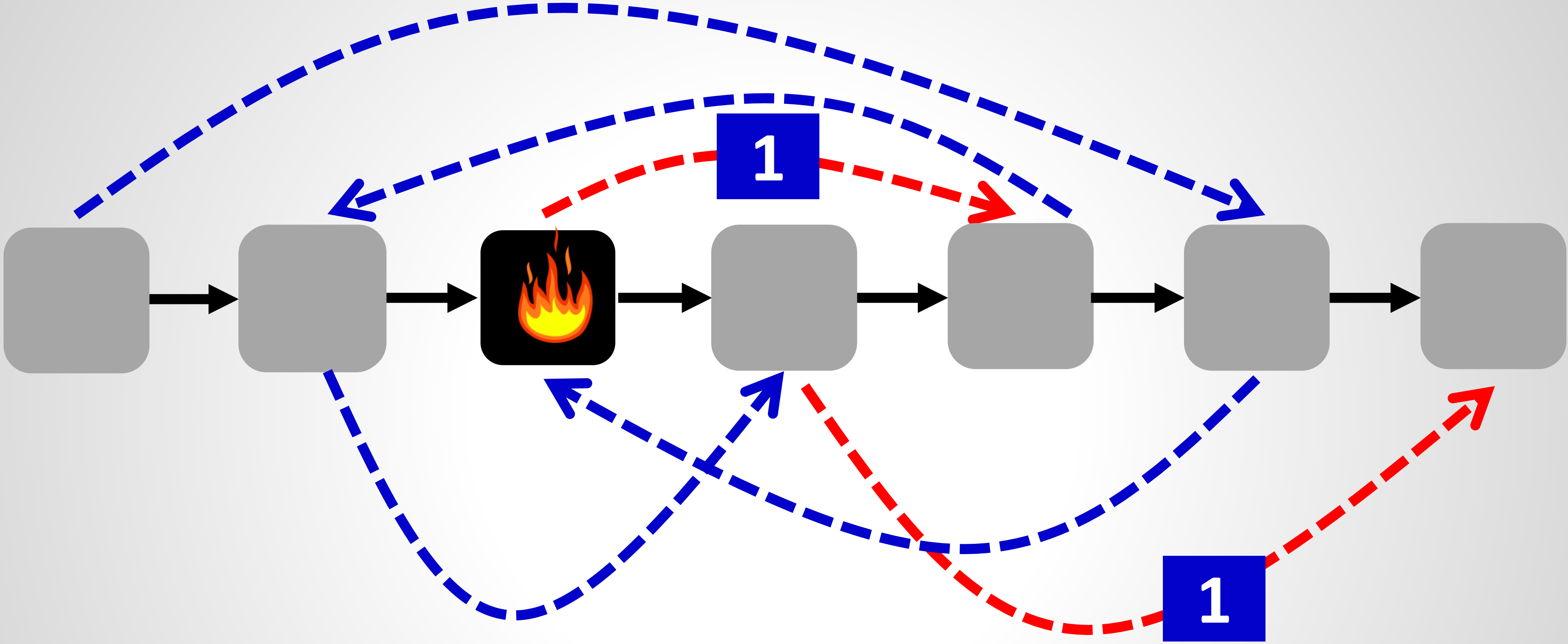
What about this one?



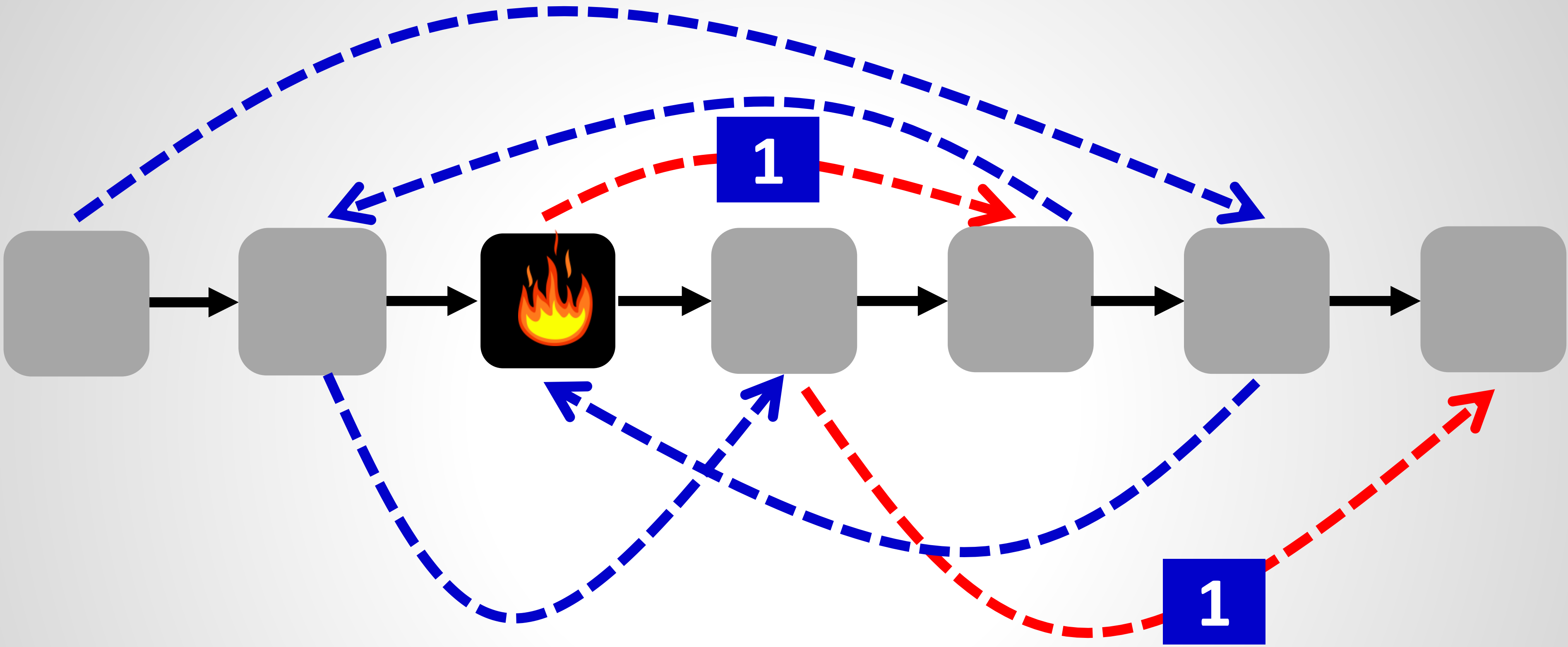
Back to the start: What if....



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

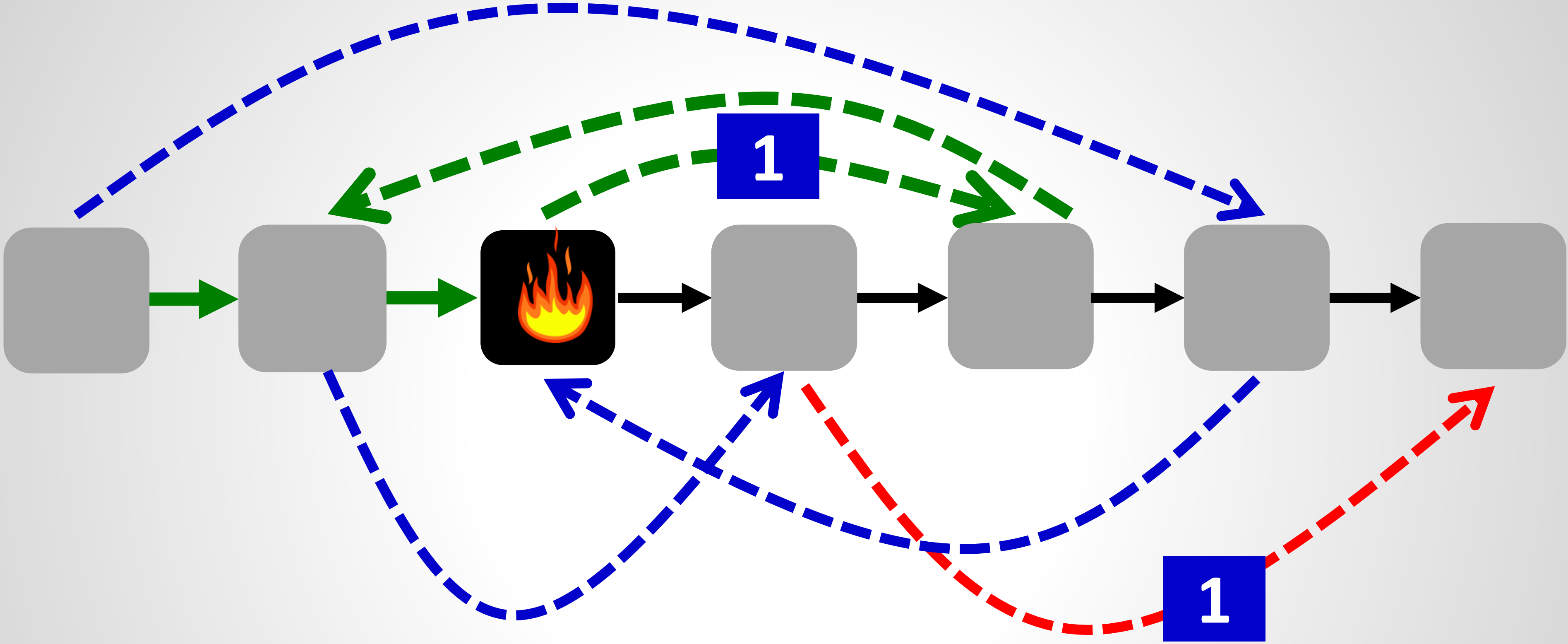


Back to the start: What if…. also this one?!



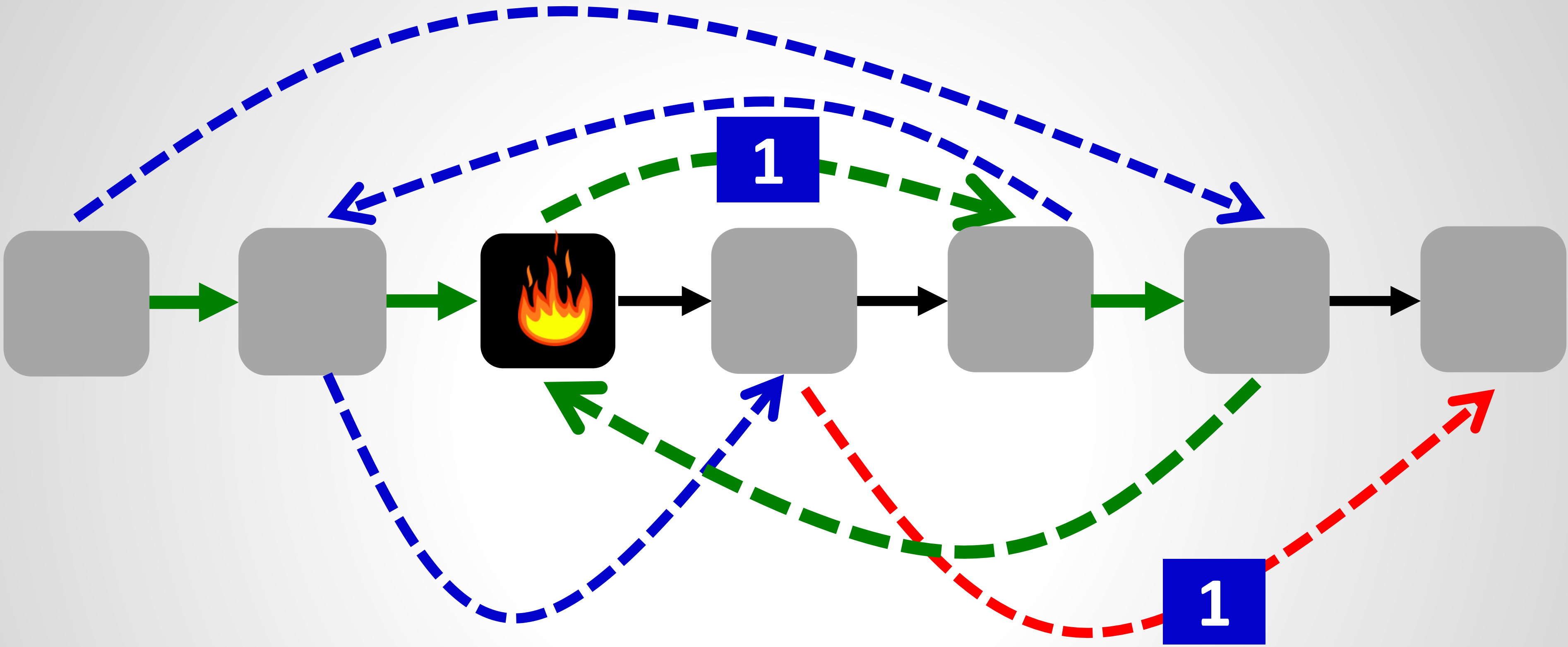
❑ Update any of the 2 backward edges? LF ☹

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



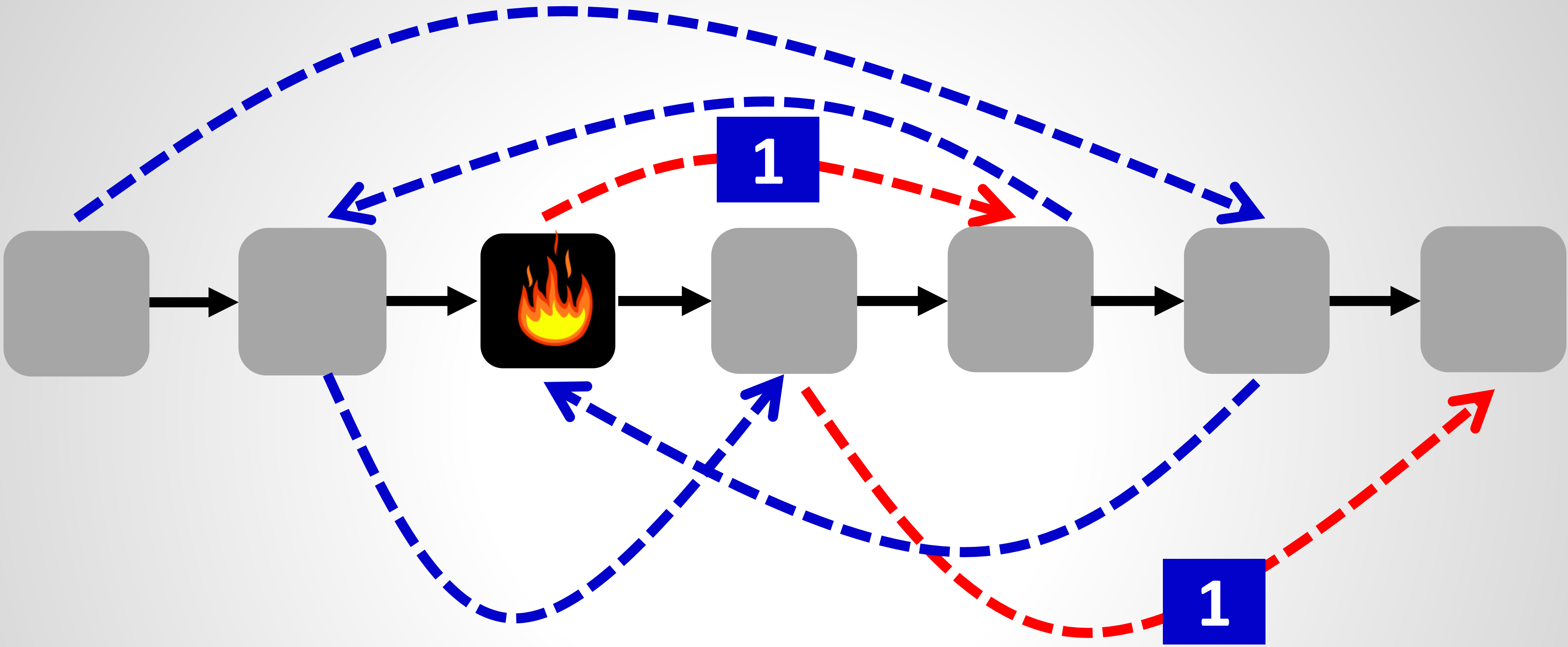
❑ Update any of the 2 backward edges? LF ☹

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



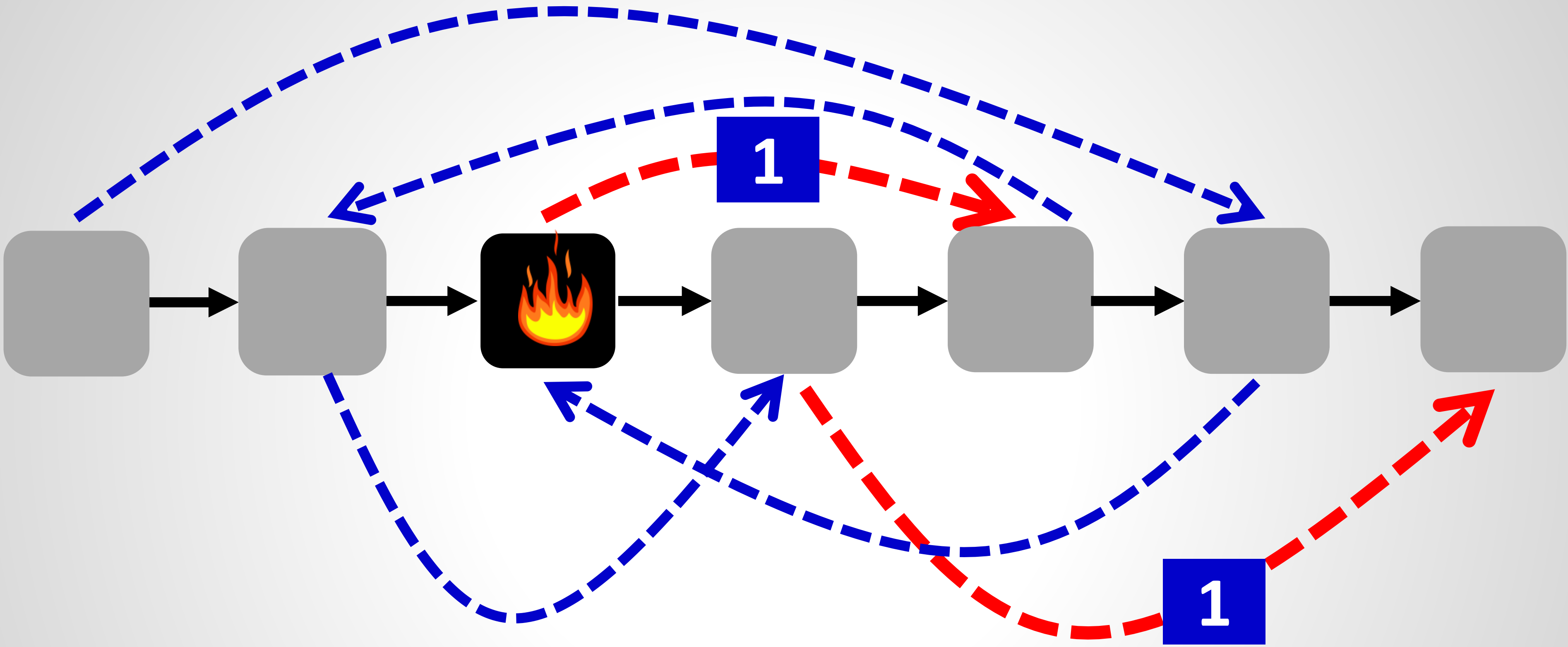
- ❑ Update any of the 2 backward edges? LF ☹

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

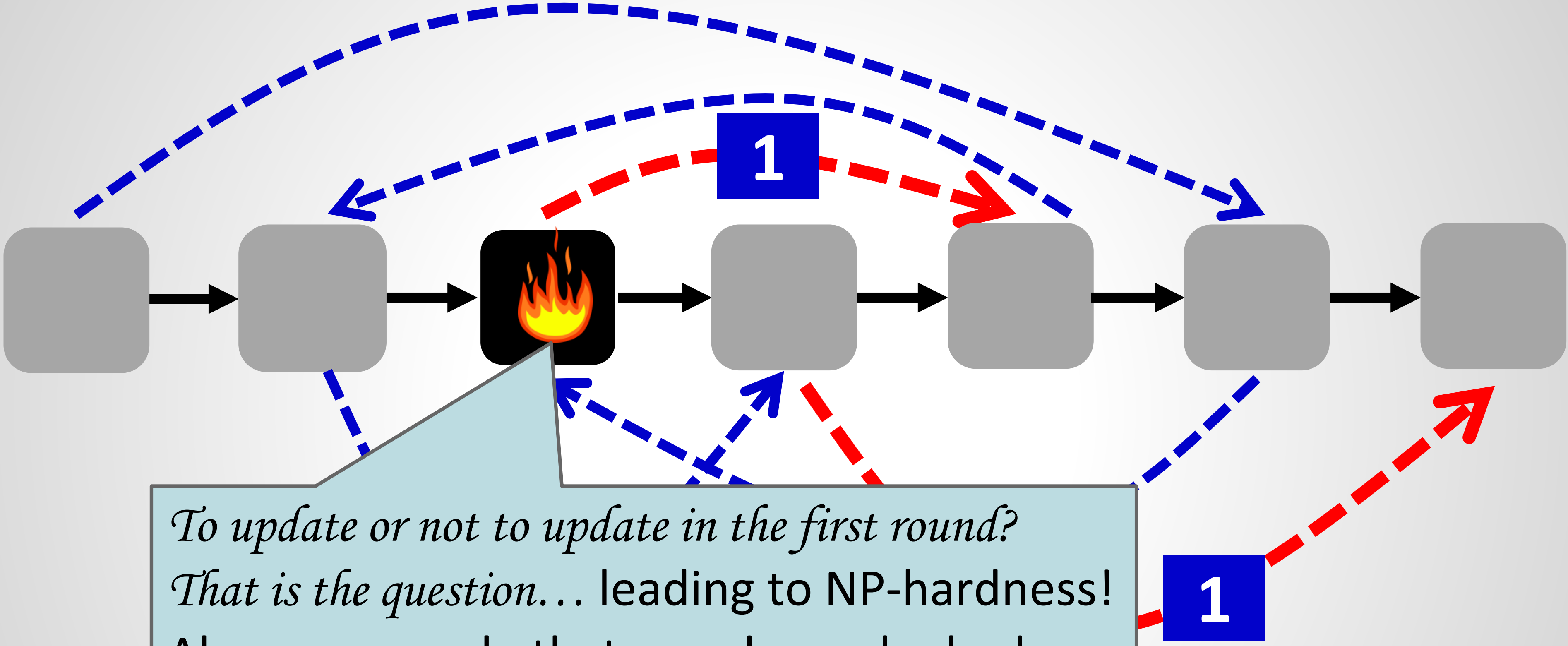


- ❑ Update any of the 2 backward edges? LF ☹
- ❑ Update any of the 2 other forward edges? WPE ☹
- ❑ What about a combination? No…

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

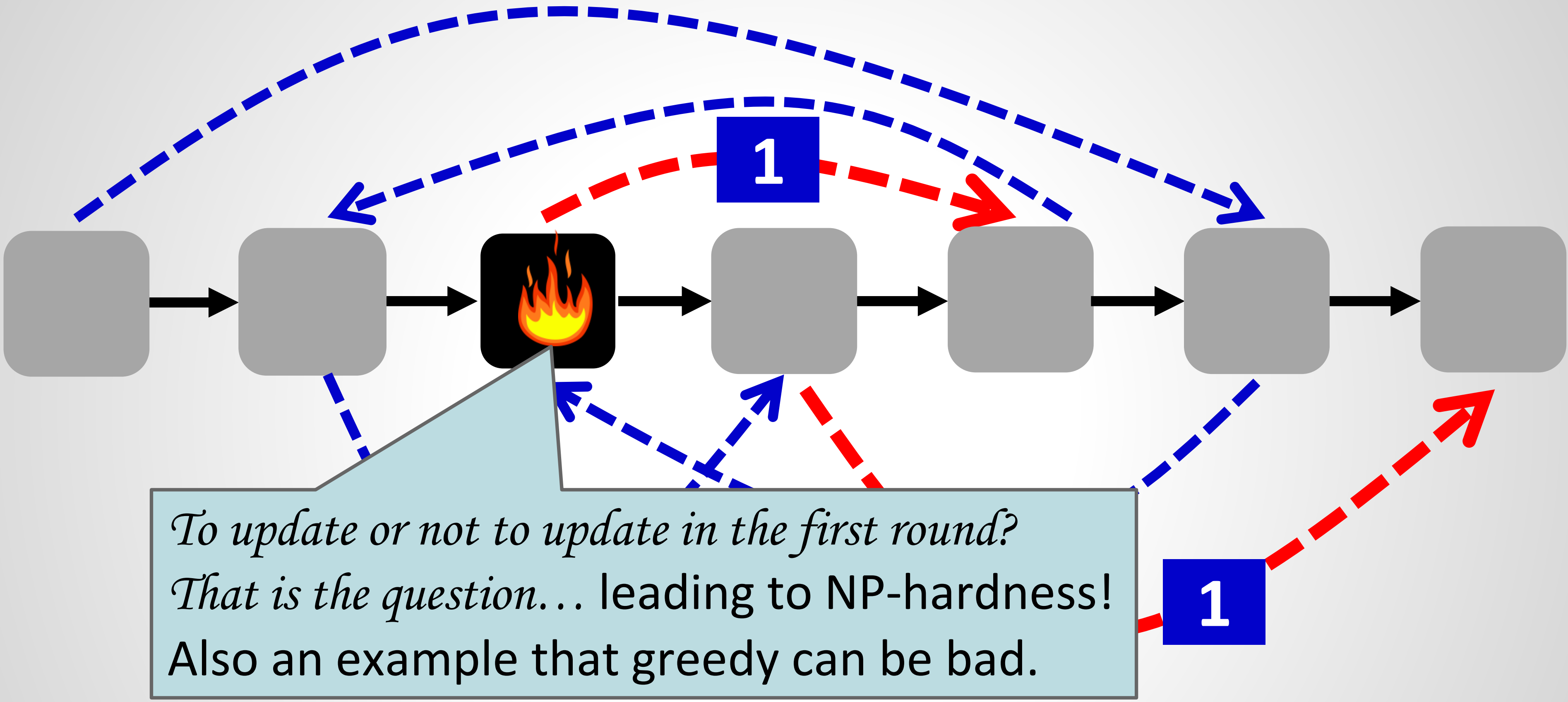


Back to the start: What if…. also this one?!



To update or not to update in the first round?
That is the question… leading to NP-hardness!
Also an example that greedy can be bad.

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

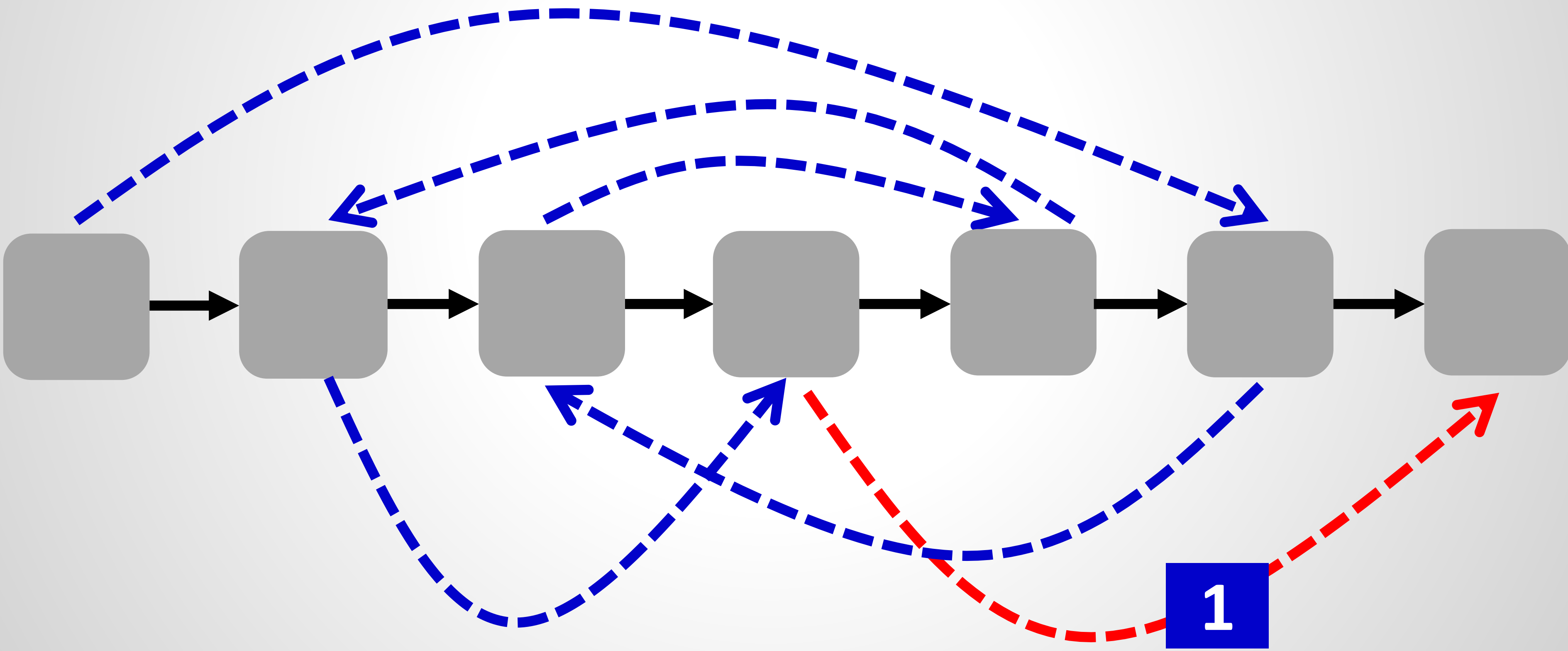


Bad news: Even decidability hard: cannot quickly test feasibility and if infeasible resort to say, tagging solution!

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

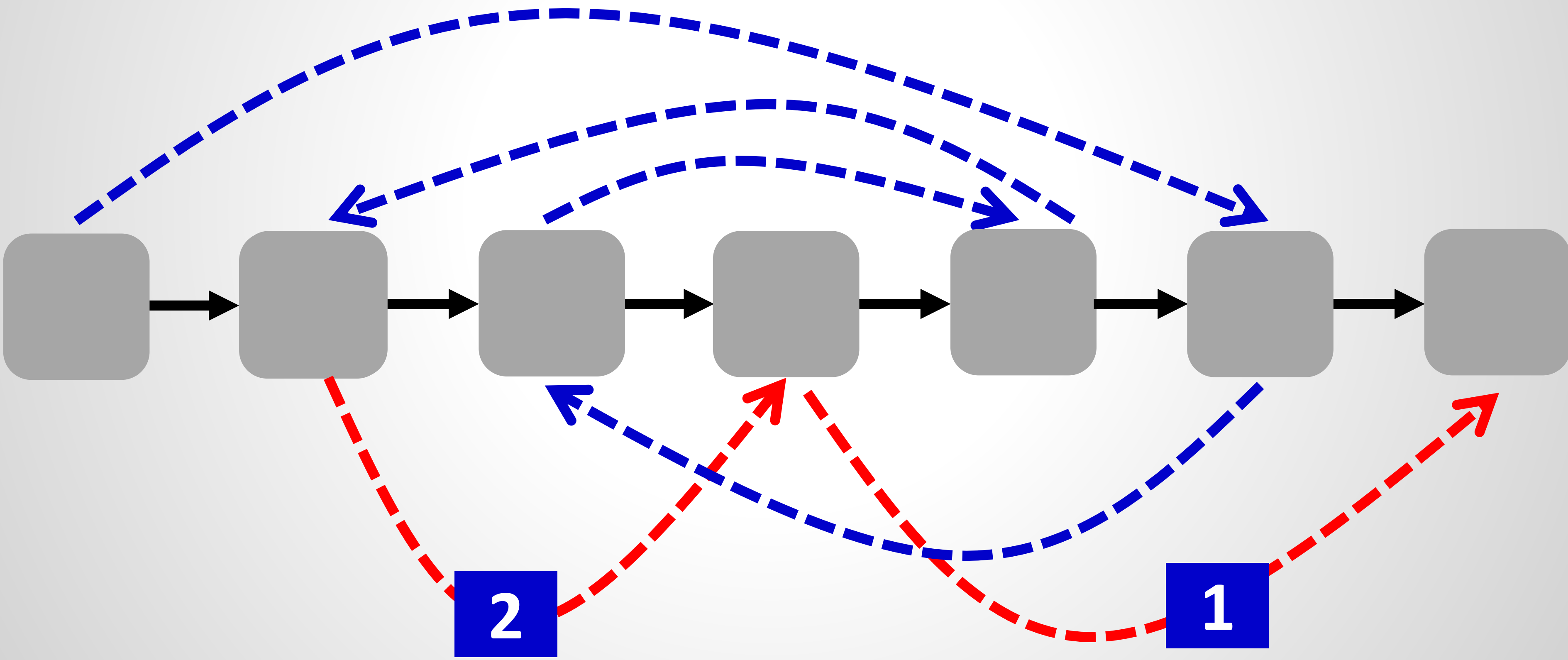
Let us focus on loop-freedom only: always possible in *n* rounds! How?

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



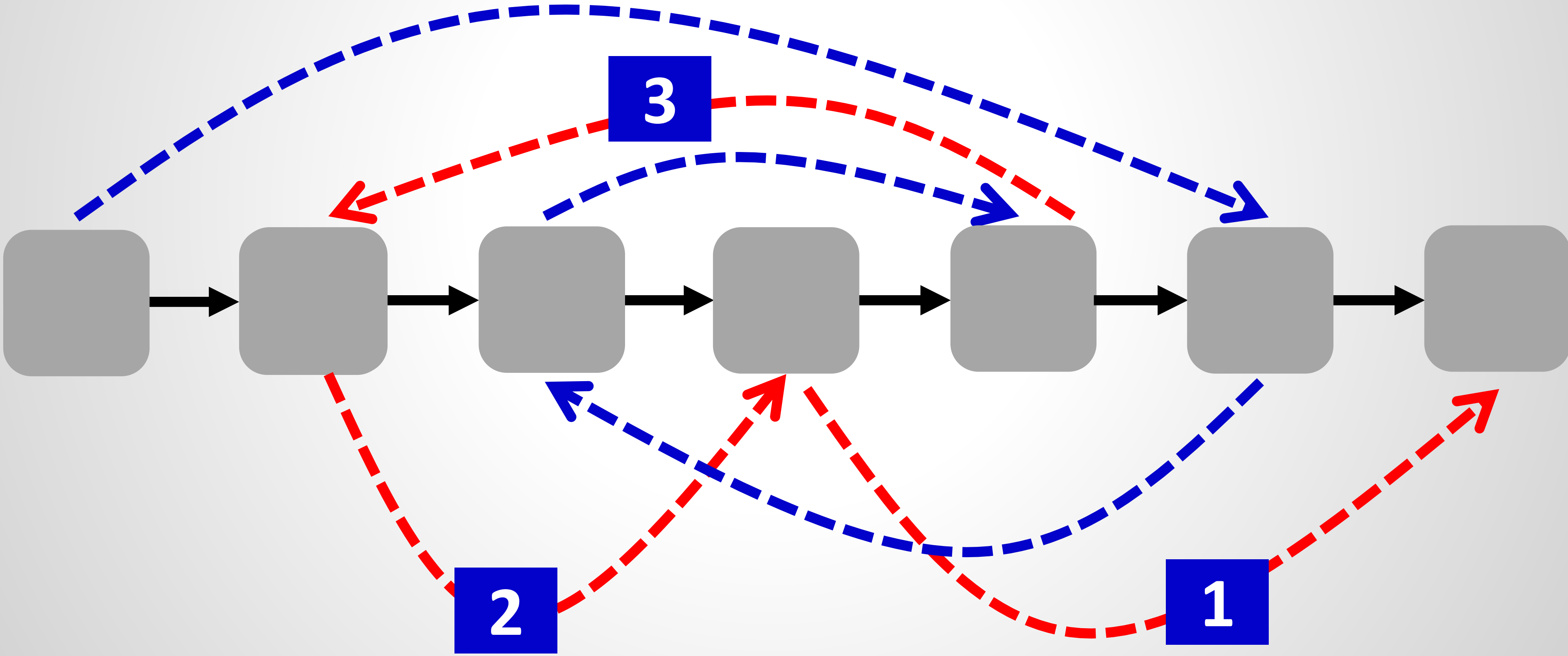
From the destination! Invariant: path suffix updated!

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



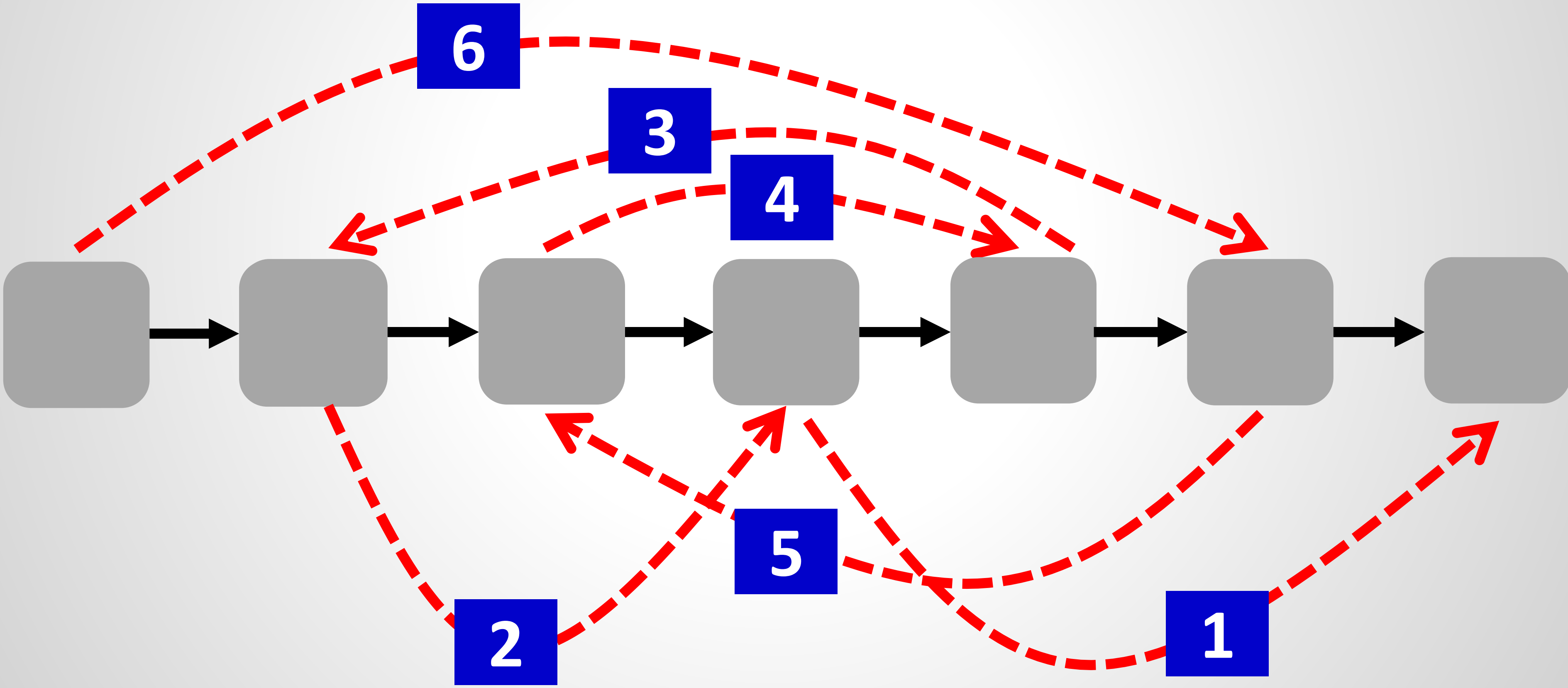
From the destination! Invariant: path suffix updated!

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



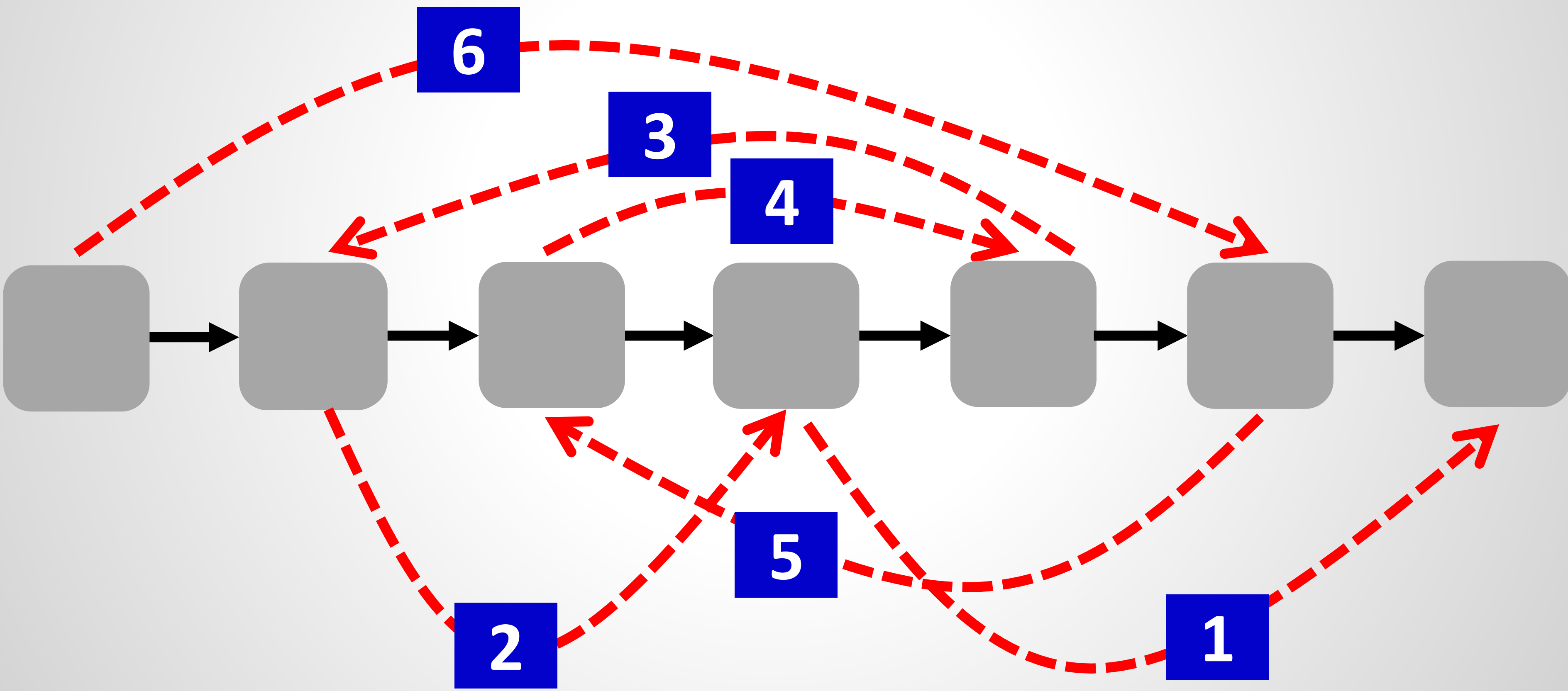
From the destination! Invariant: path suffix updated!

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?

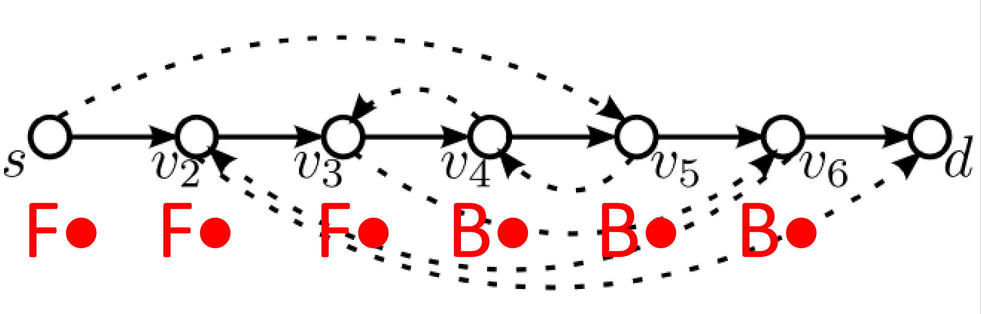
Example: Optimal 2-Round Update Schedules

Round 1 (R1): Clearly, I can only update „forward“ links (wrt to old route)!

Round 2 (R2) (or last round in general): By a **symmetry** argument, I can only update the „forward“ links with respect to the new route: an update schedule read backward (i.e., updating **from new to old policy**), must also be legal!

Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

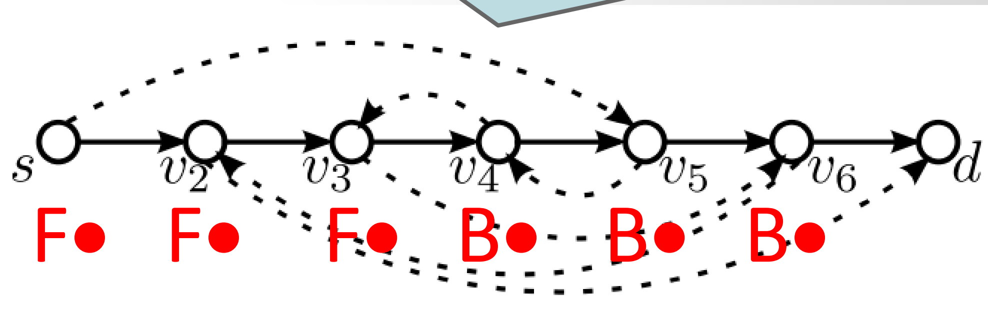
- Classify nodes/edges with **2-letter code**:
  - F•, B•: Does (dashed) new edge point forward or backward wrt (solid) old path?



Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

- Classify nodes/edg
  - F•, B•: Does (dashed) new edge point forward or backward wrt (solid) old path?

Old policy from left to right!

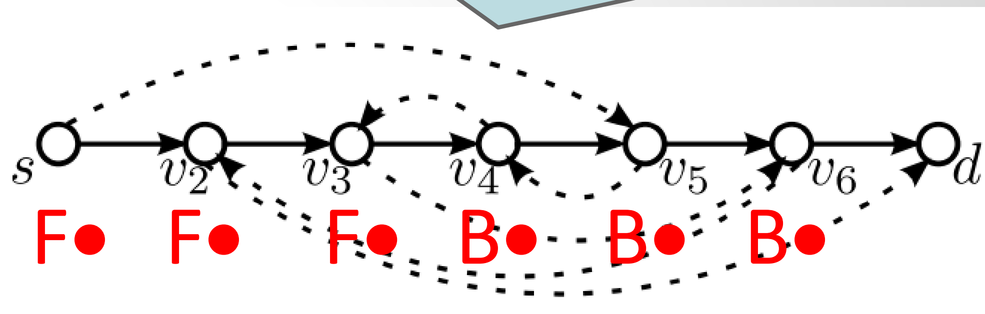


Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

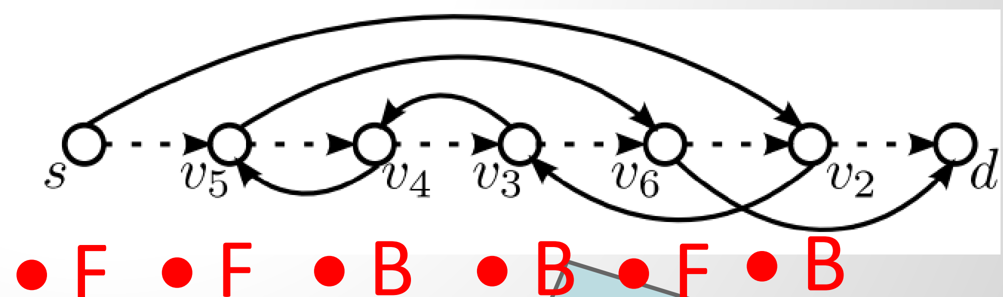
- Classify nodes/edg

Old policy from left to right!

  - F•, B•: Does (dashed) new edge point forward or backward wrt (solid) old path?



  - •F, •B: Does the (solid) old edge point forward or backward wrt (dashed) new path?



New policy from left to right!

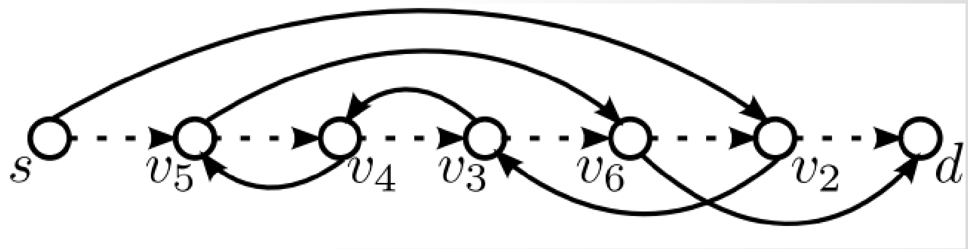
Optimal Algorithm for 2-Round Instances: ...ging Symmetry!

...es with 2-letter code:

Insight 1: In the 1st round, I can safely update all forwarding (F•) edges! For sure loopfree.

Insight 2: Valid schedules are reversible! A valid schedule from old to new *read backward* is a valid schedule for new to old!

Insight 3: Hence in the last round, I can safely update all forwarding (•F) edges! For sure loopfree.



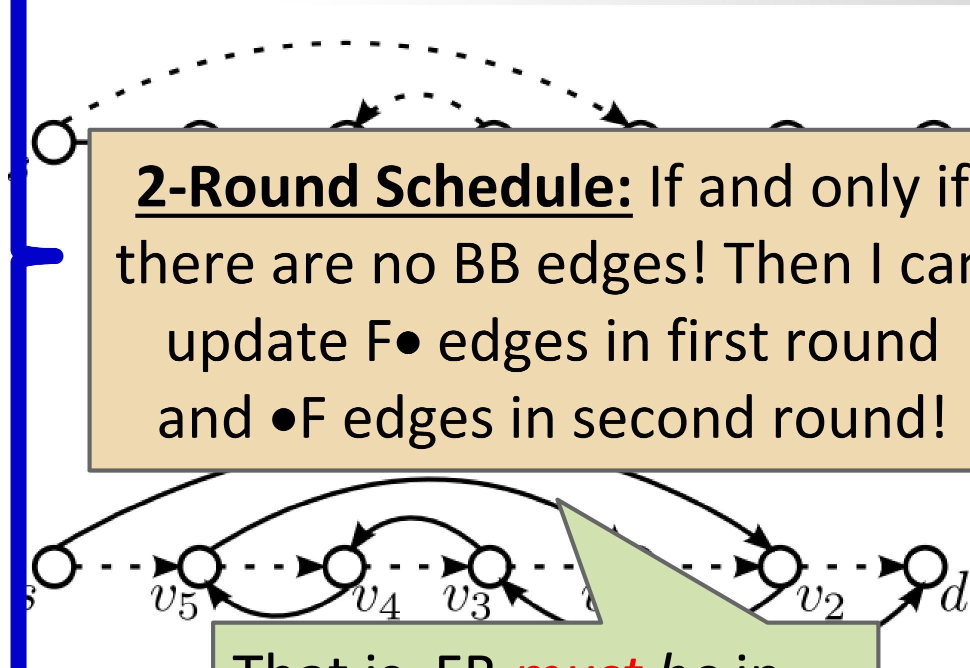
Optimal Algorithm for 2-Round Instances: ...ging Symmetry!

...es with 2-letter code:

<u>Insight 1:</u> In the 1st round, I can safely update all forwarding (F•) edges! For sure loopfree.

<u>Insight 2:</u> Valid schedules are reversible! A valid schedule from old to new *read backward* is a valid schedule for new to old!

<u>Insight 3:</u> Hence in the last round, I can safely update all forwarding (•F) edges! For sure loopfree.

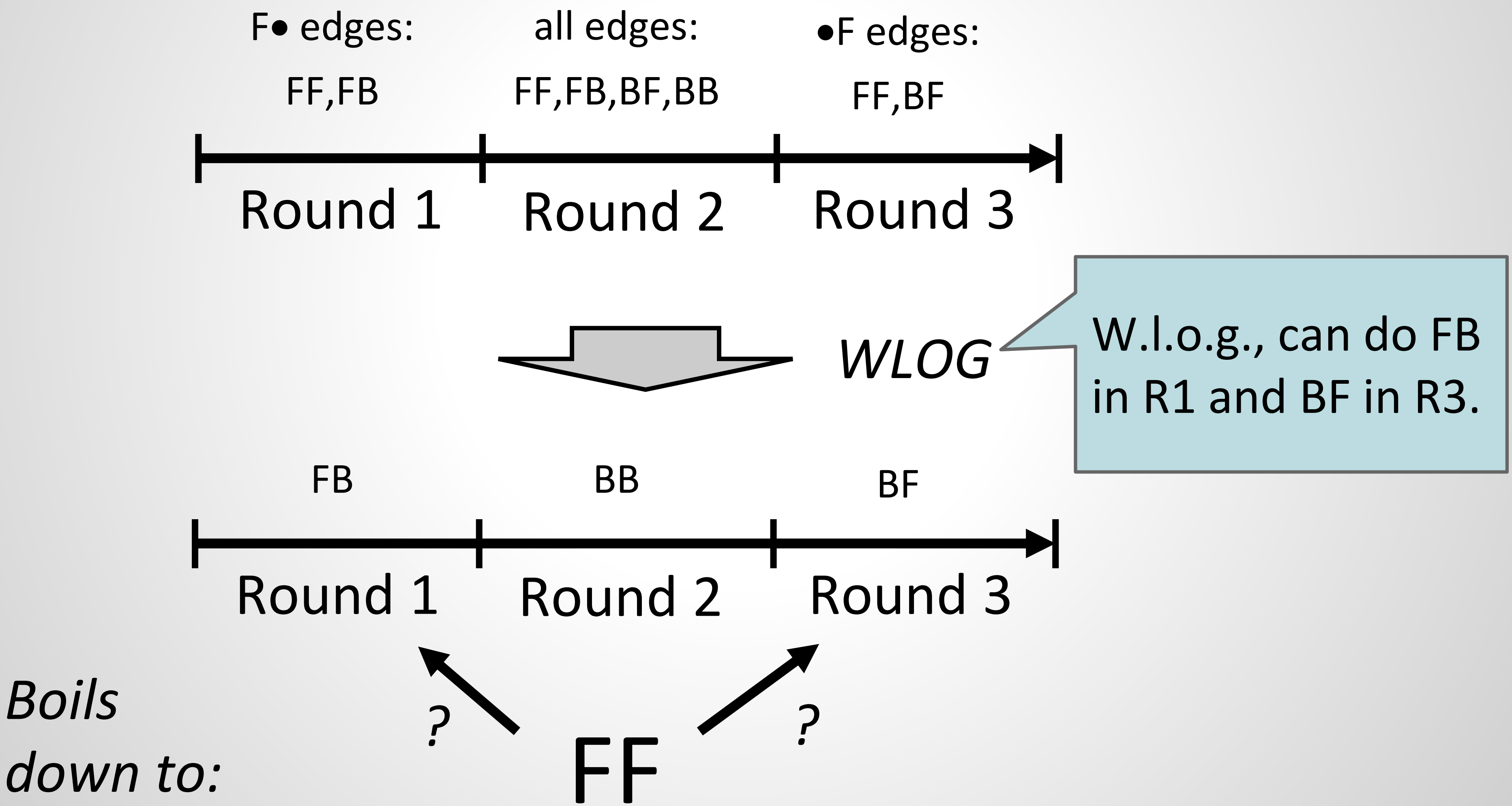


<u>2-Round Schedule:</u> If and only if there are no BB edges! Then I can update F• edges in first round and •F edges in second round!

That is, FB *must be* in first round, BF *must be* in second round, and FF *are flexible*!

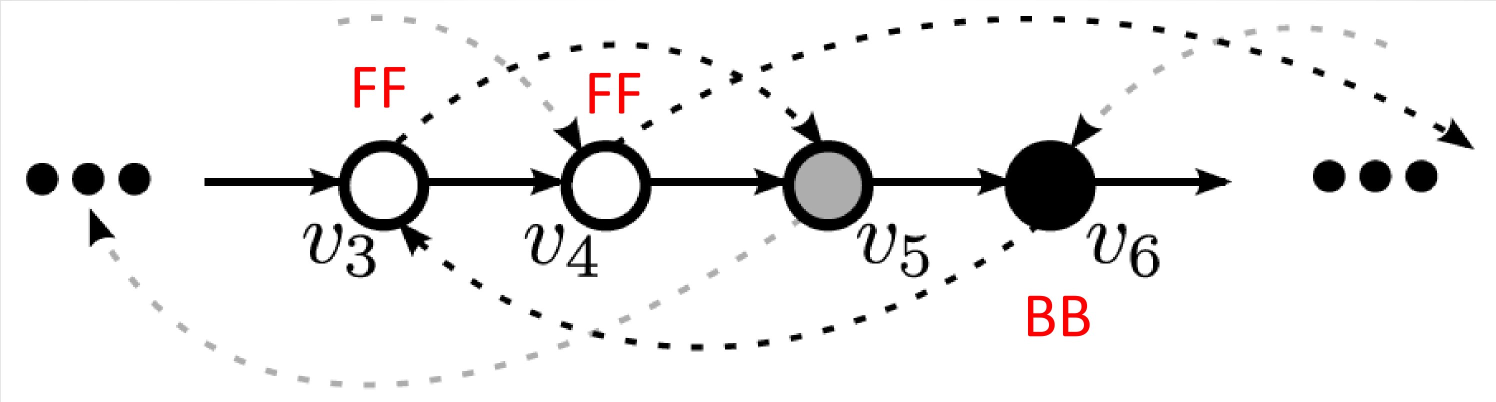
3 Rounds Are Hard: Intuition Why

- Structure of a 3-round schedule:



3 Rounds Are Hard: Intuition Why

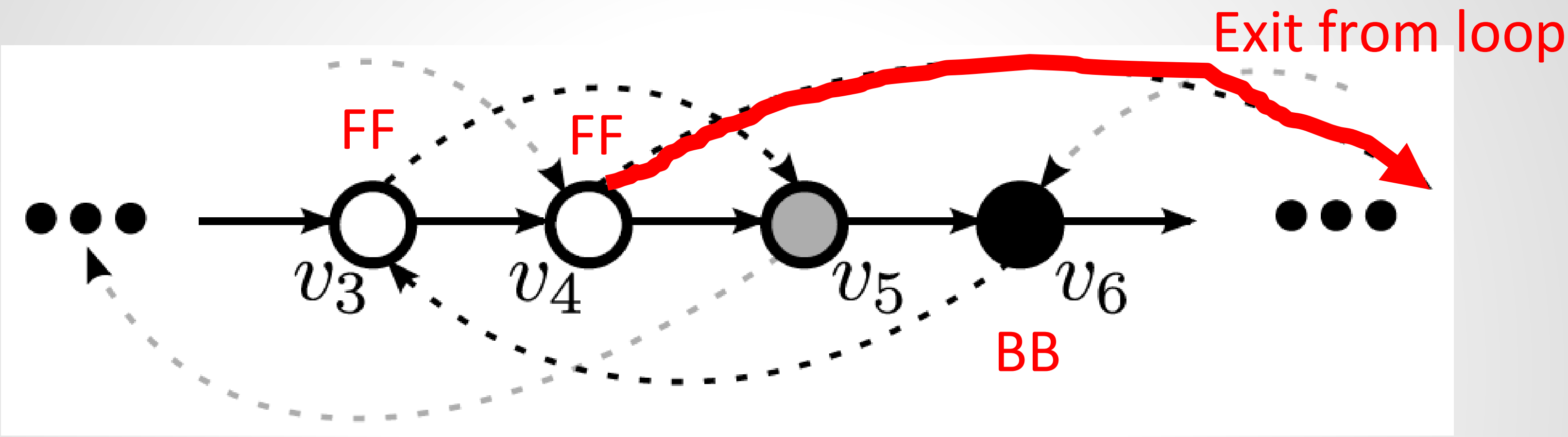
A hard decision problem: when to update FF?



- We know: BB node v_6 can only be updated in R2
- When to update FF nodes to enable update BB in R2?

3 Rounds Are Hard: Intuition Why

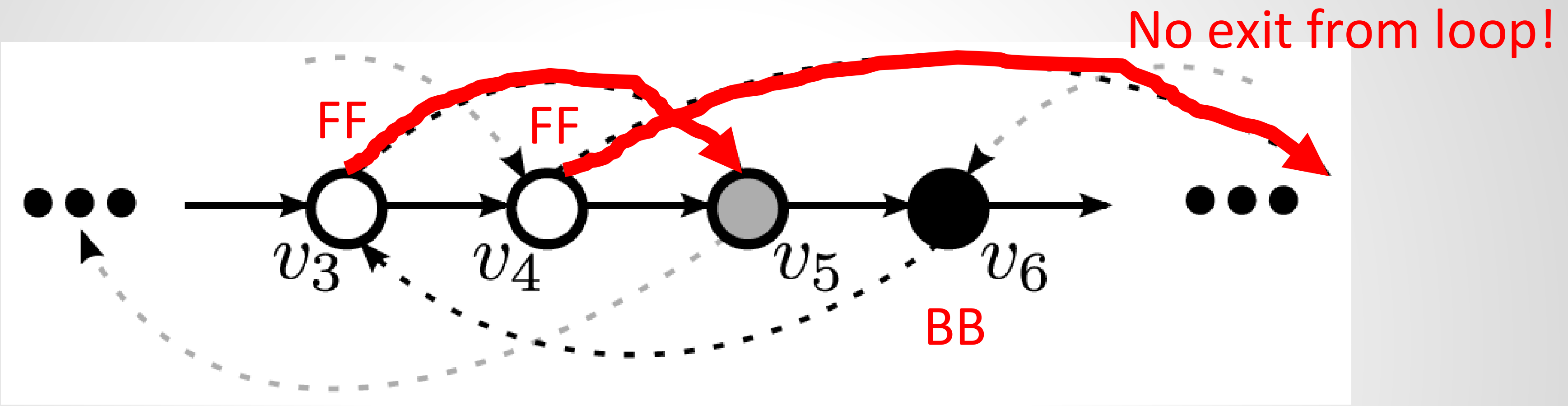
A hard decision problem: when to update FF?



- We know: BB node v_6 can only be updated in R2
- When to update FF nodes to enable update BB in R2
- E.g, updating FF-node v_4 in R1 allows to update BB v_6 in R2

3 Rounds Are Hard: Intuition Why

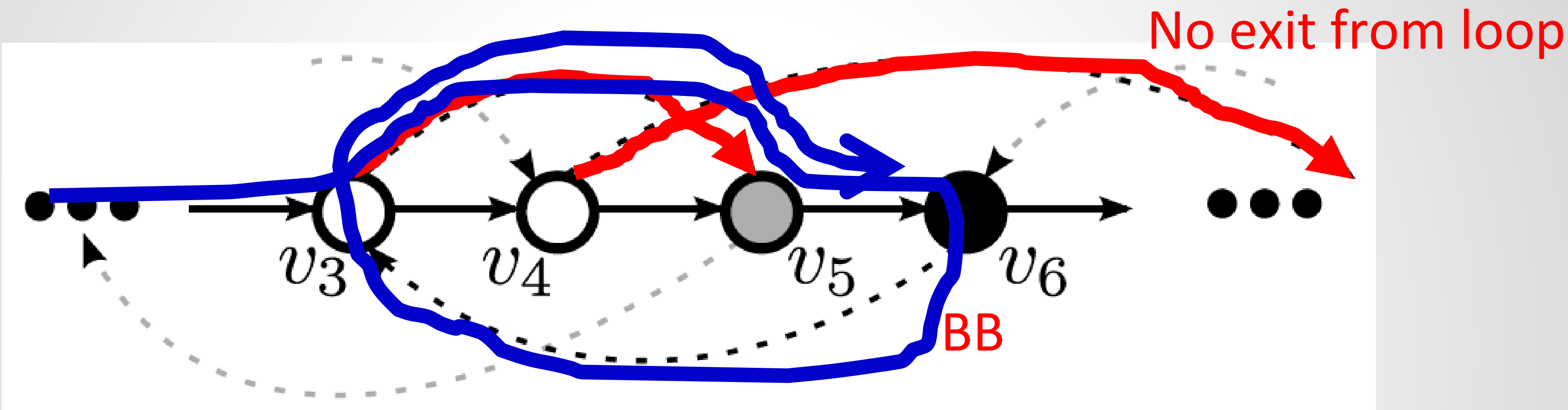
A hard decision problem: when to update FF?



- We know: BB node v_6 can only be updated in R2
- When to update FF nodes to enable update BB in R2
- E.g, updating FF-node v_4 in R1 allows to update BB v_6 in R2
- But only if FF-node v_3 is not updated as well in R1: potential loop

3 Rounds Are Hard: Intuition Why

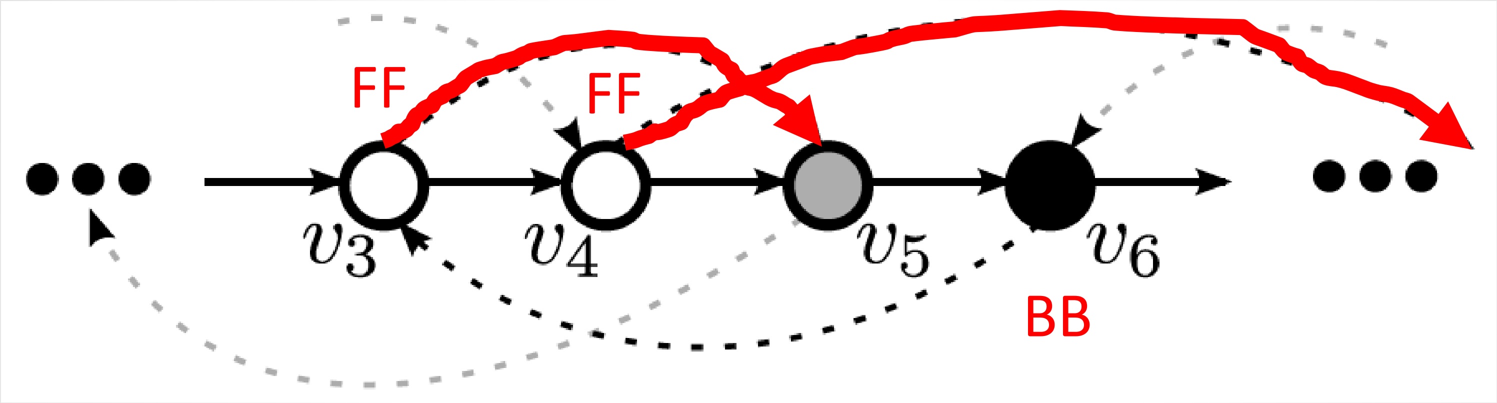
A hard decision problem: when to update FF?



- We know: BB node v_6 can only be updated in R2
- When to update FF nodes to enable update BB in R2
- E.g, updating FF-node v_4 in R1 allows to update BB v_6 in R2
- But only if FF-node v_3 is not updated as well in R1: potential loop

3 Rounds Are Hard: Intuition Why

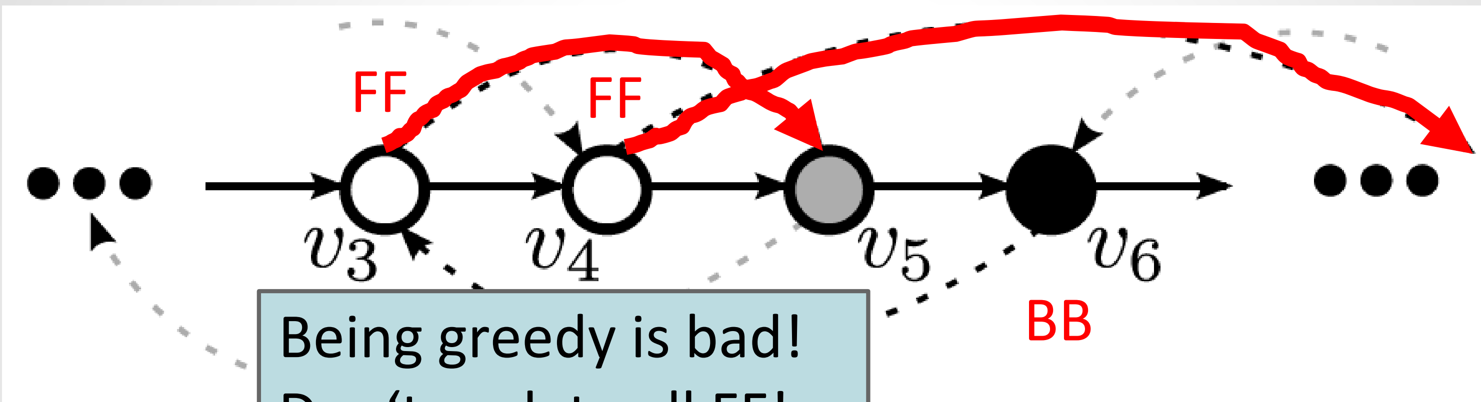
A hard decision problem: when to update FF?



- We know: BB node v_6 can only be updated in R2
- When to update FF nodes to enable update BB in R2
- E.g, updating FF-node v_4 in R1 allows to update BB v_6 in R2
- But only if FF-node v_3 is not updated as well in R1: potential loop
- Looks like a gadget: which FF nodes to update when is hard!

3 Rounds Are Hard: Intuition Why

A hard decision problem: when to update FF?



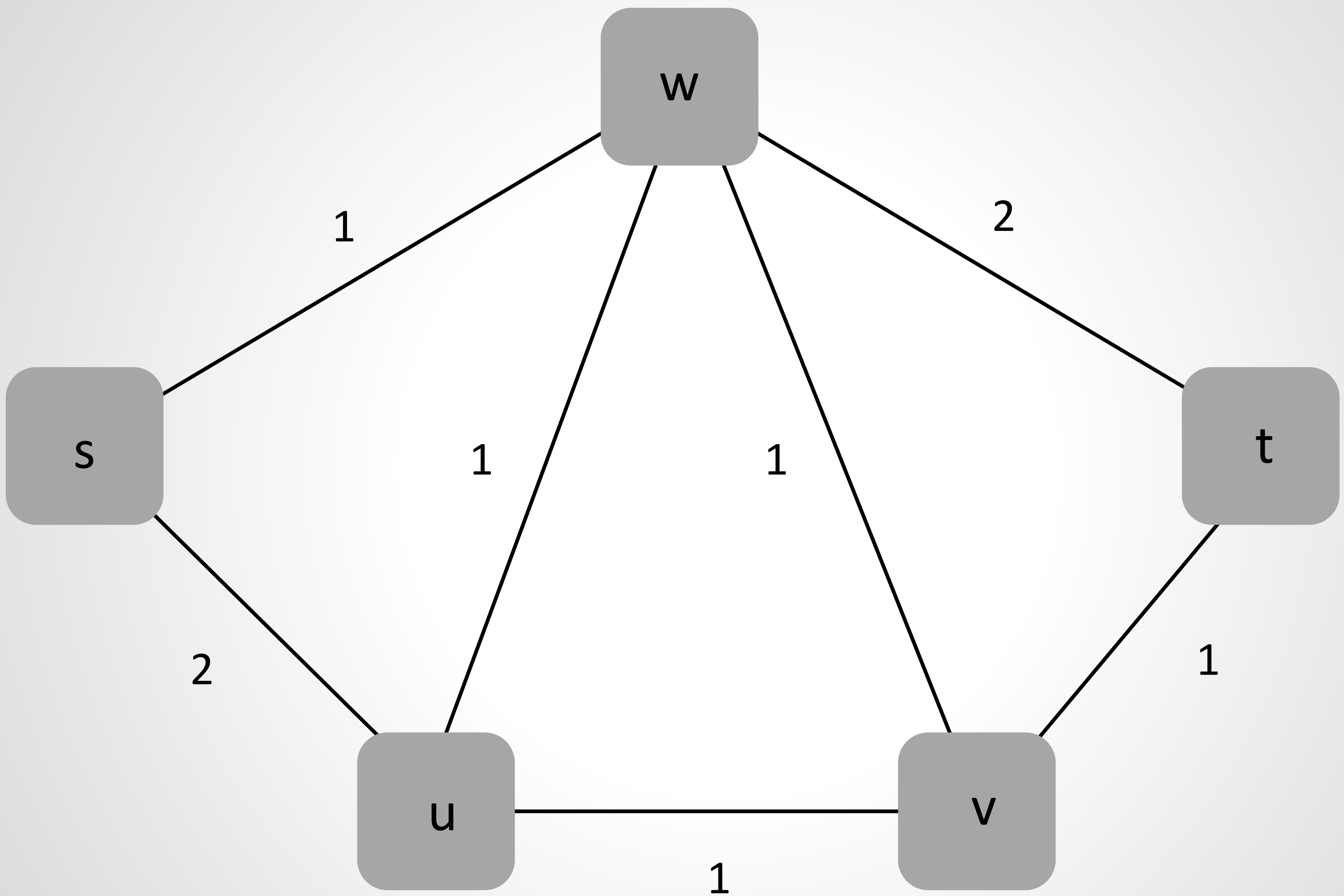
Being greedy is bad!
Don't update all FF!

- We know: [illegible]ode v_6 can only be updated in R2
- When to update FF nodes to enable update BB in R2
- E.g, updating FF-node v_4 in R1 allows to update BB v_6 in R2
- But only if FF-node v_3 is not updated as well in R1: potential loop
- Looks like a gadget: which FF nodes to update when is hard!

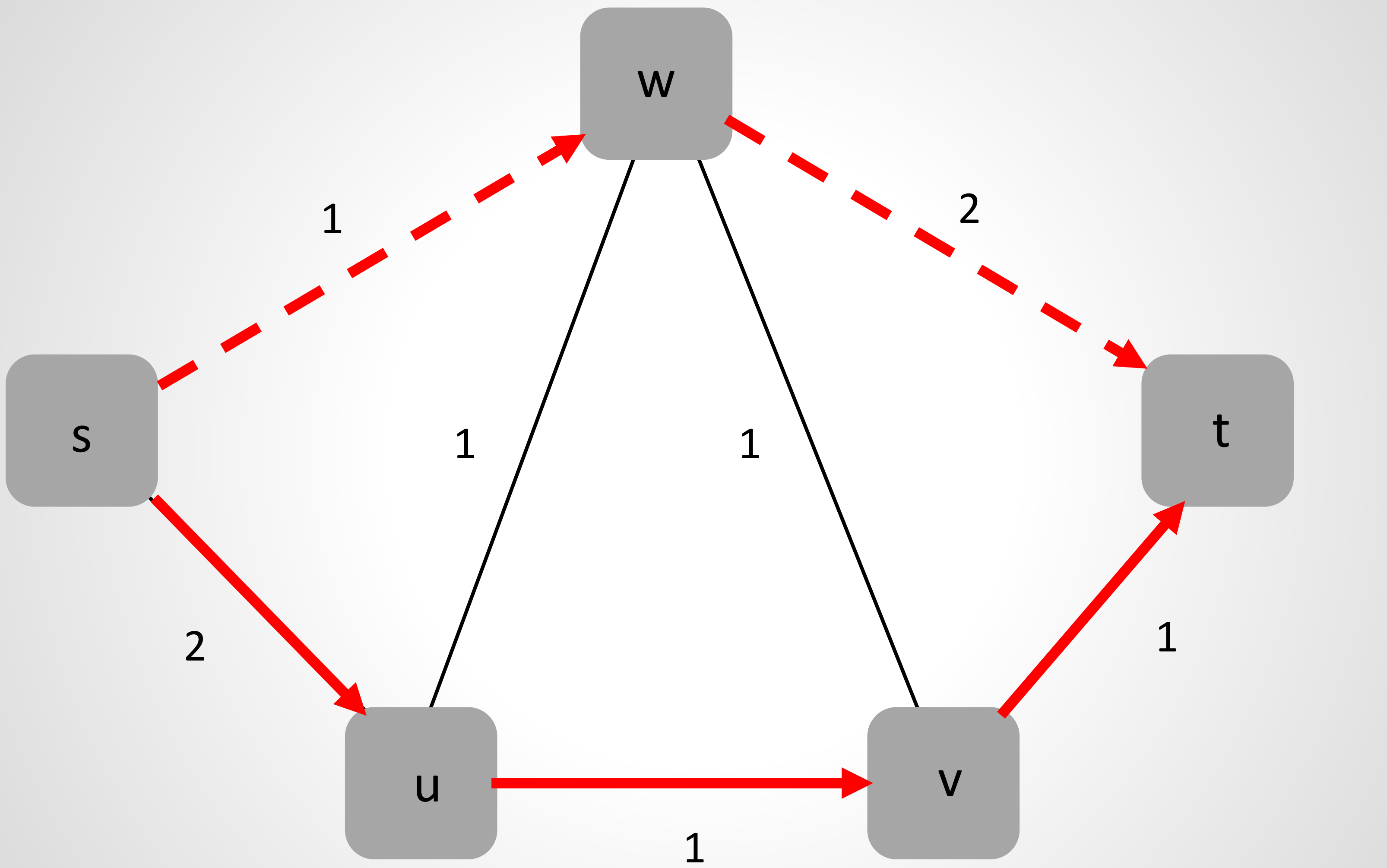
Loop-Freedom: Summary of Results or the Lack Thereof ☺

- Minimizing the **number of rounds**
  - For 2-round instances: polynomial time
  - For 3-round instances: NP-hard, no approximation known
- Relaxed notion of loop-freedom: O(log n) rounds
  - No approximation known
- Maximizing the **number of updated edges** per round: NP-hard (dual feedback arc set) and bad (large number of rounds)
  - dFASP on simple graphs (out-degree 2 and originates from paths!)
  - Even hard on bounded treewidth?
  - Resulting number of rounds up to $\Omega(n)$ although $O(1)$ possible
- Multiple policies: aggregate updates to given switch!
  - Related to Shortest Common Supersequence Problem

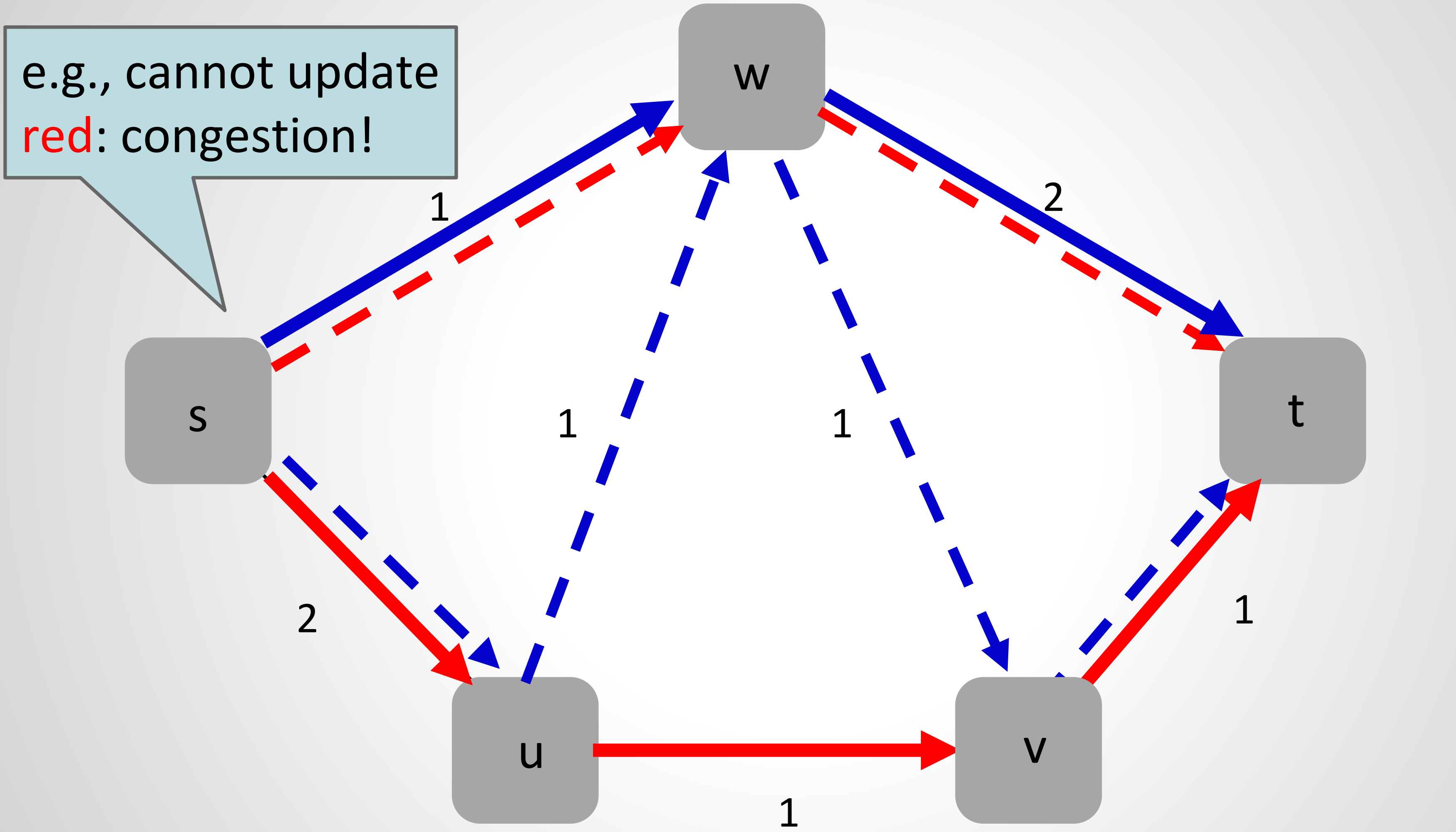
What about capacity constraints?



What about capacity constraints?

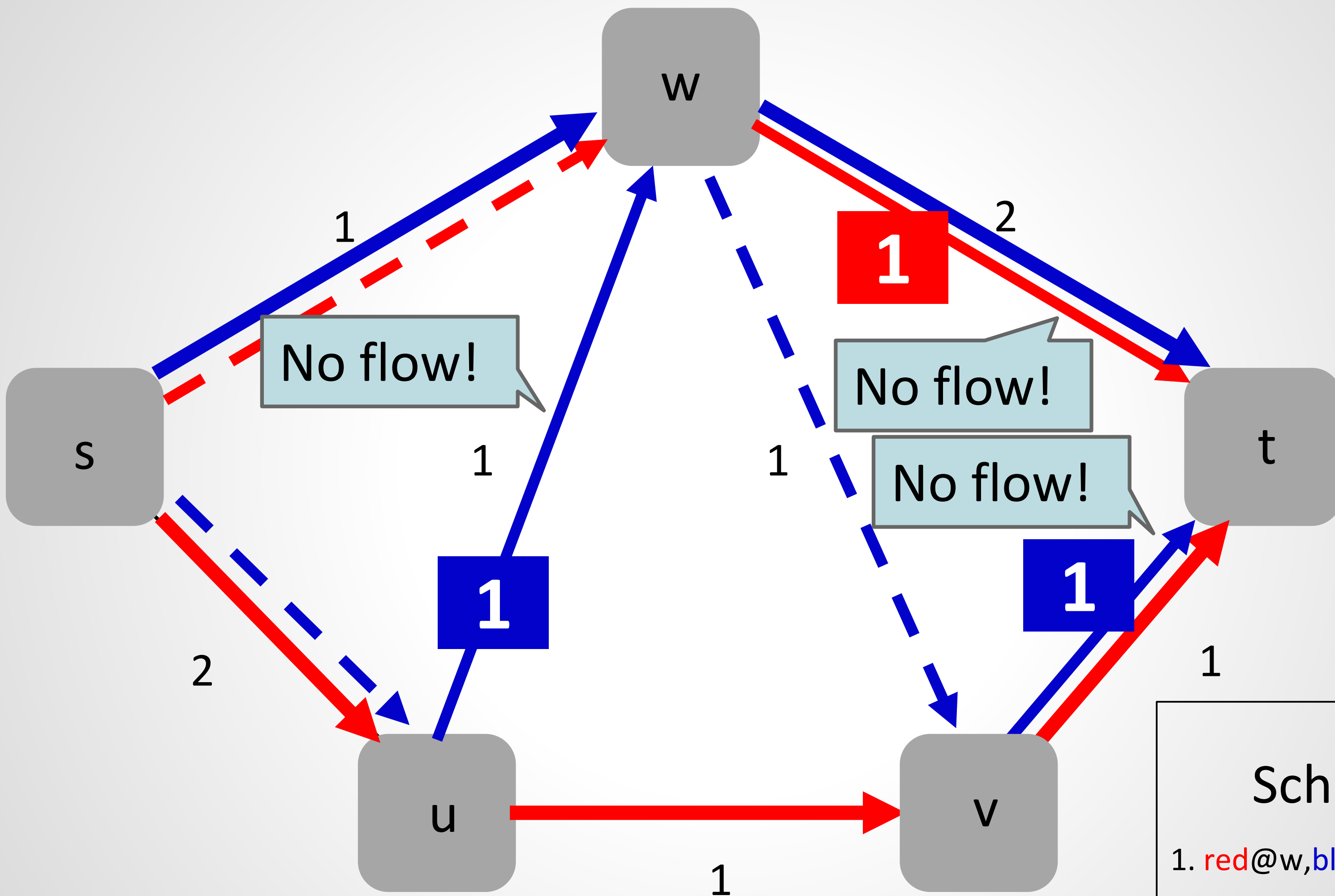


What about capacity constraints?



Can you find an update schedule?

What about capacity constraints?

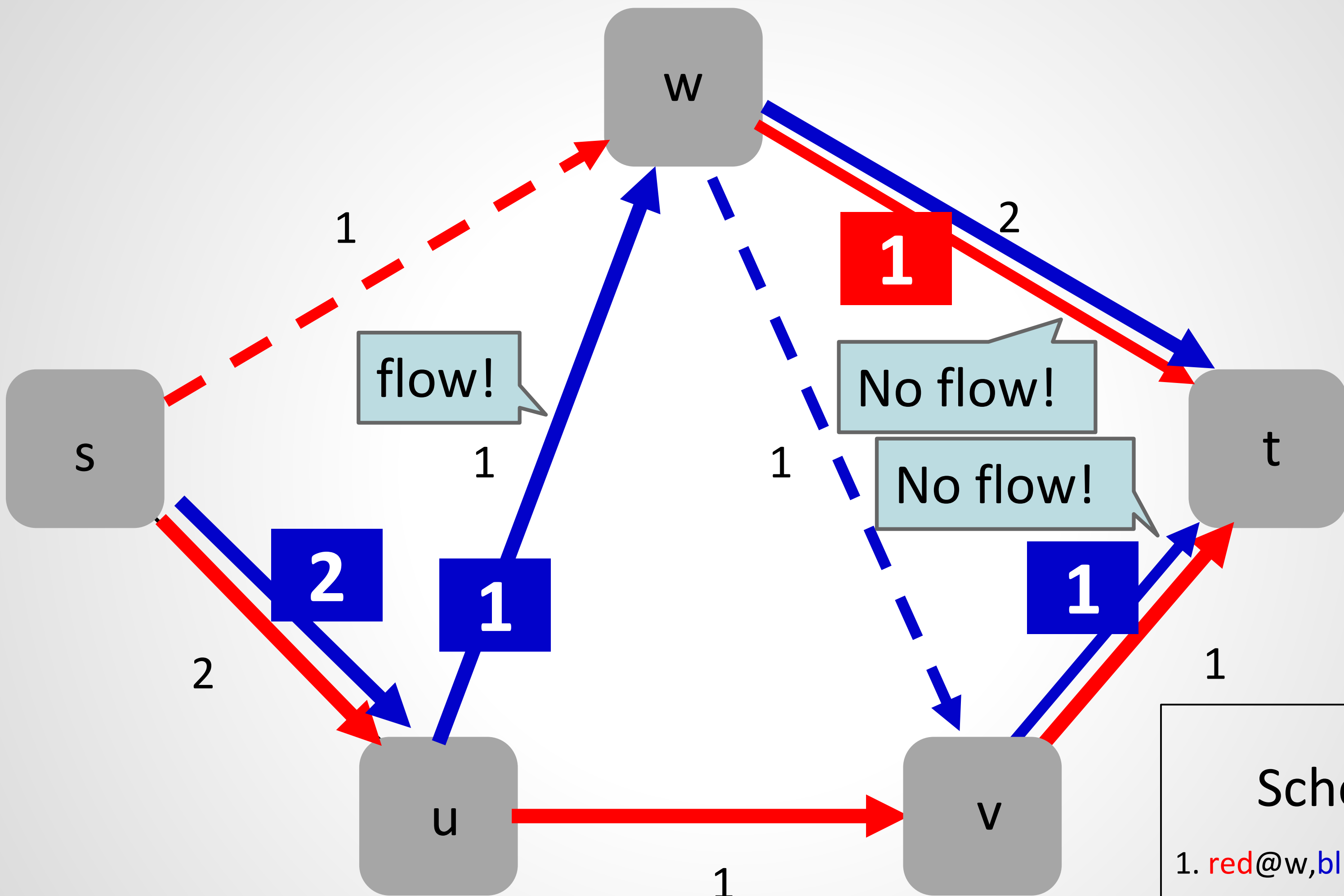


Round 1: prepare

Schedule:

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

What about capacity constraints?

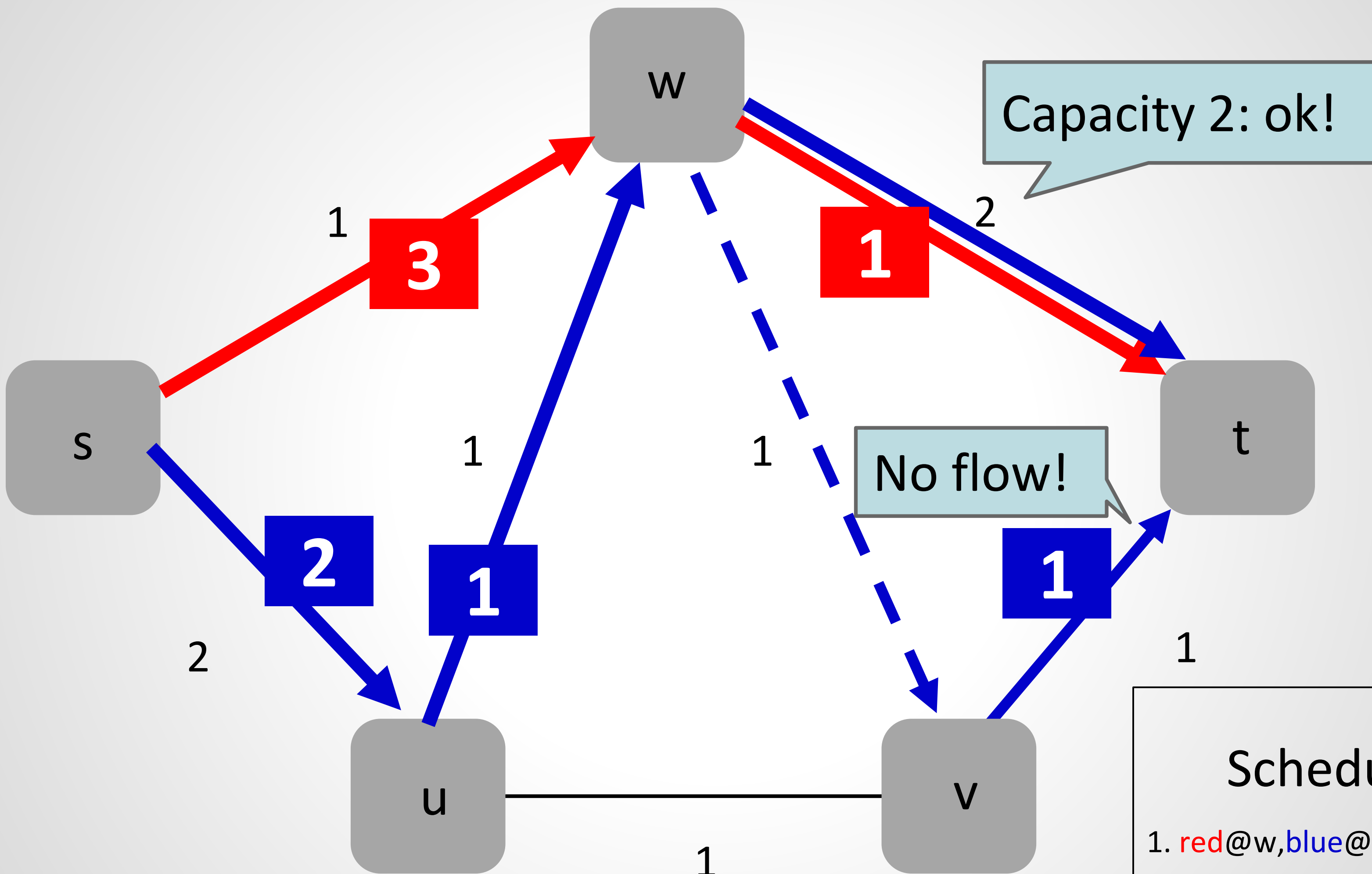


Round 2

Schedule:

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

What about capacity constraints?

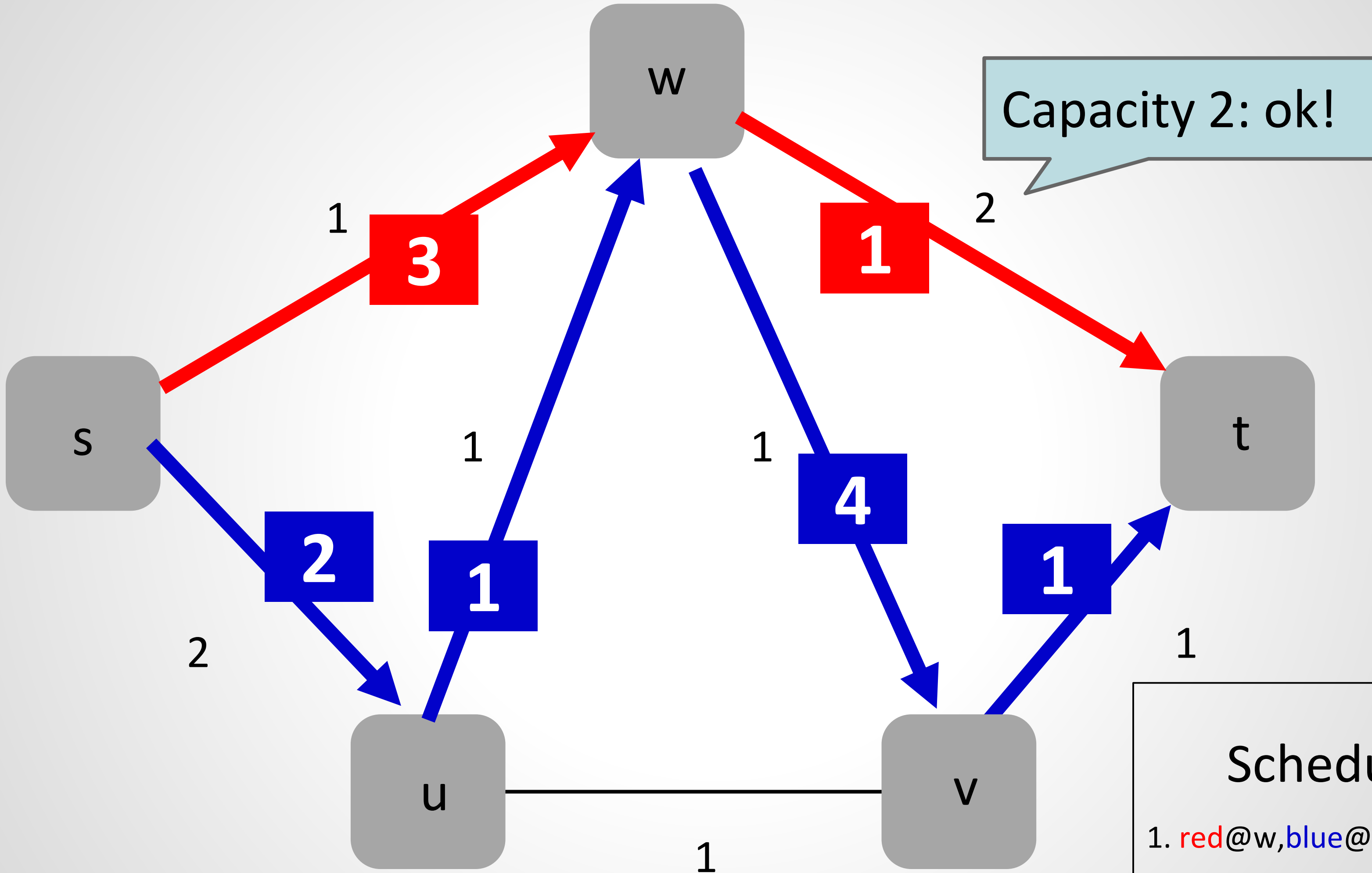


Schedule:

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

Round 3

What about capacity constraints?

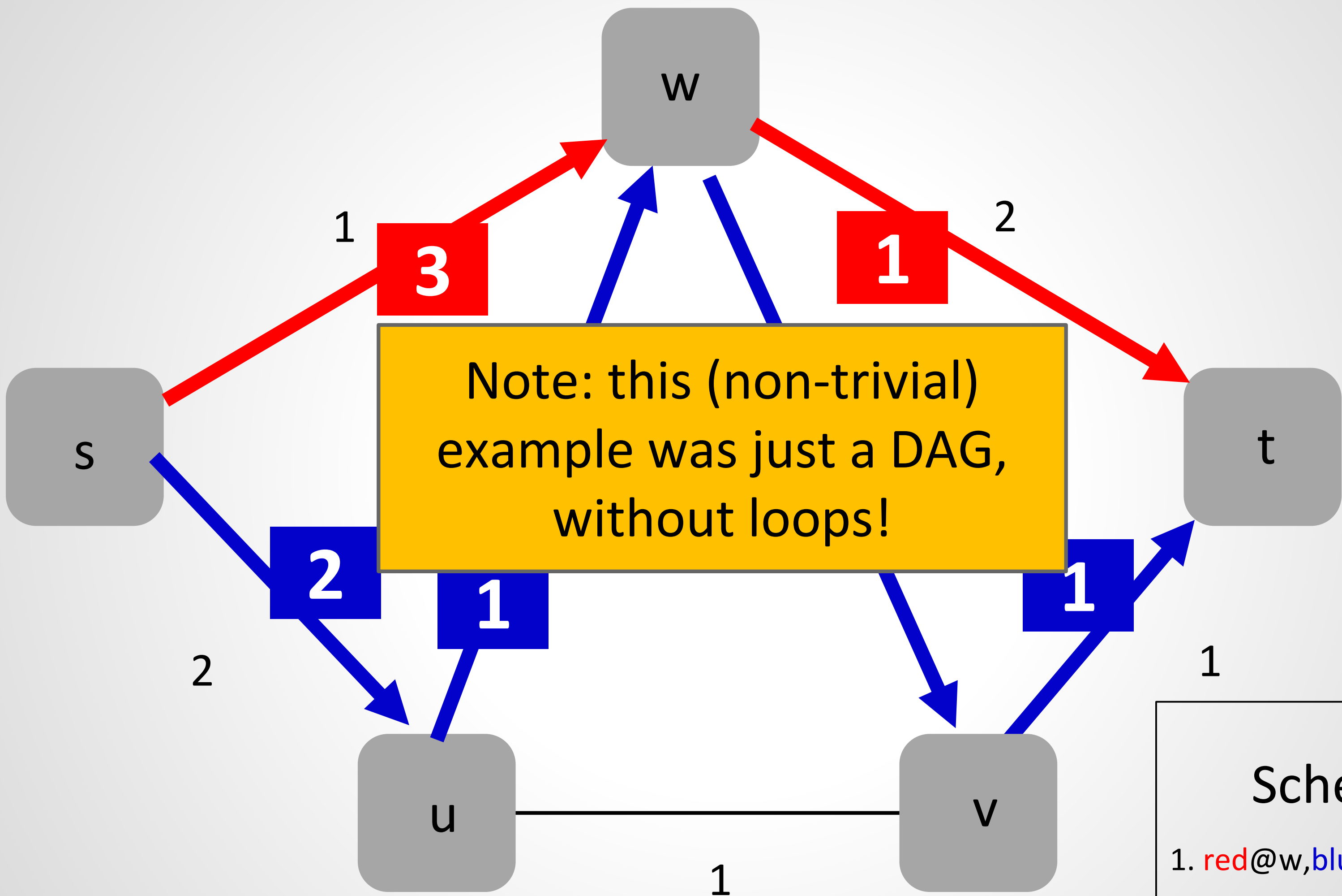


Schedule:

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

Round 4

What about capacity constraints?



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

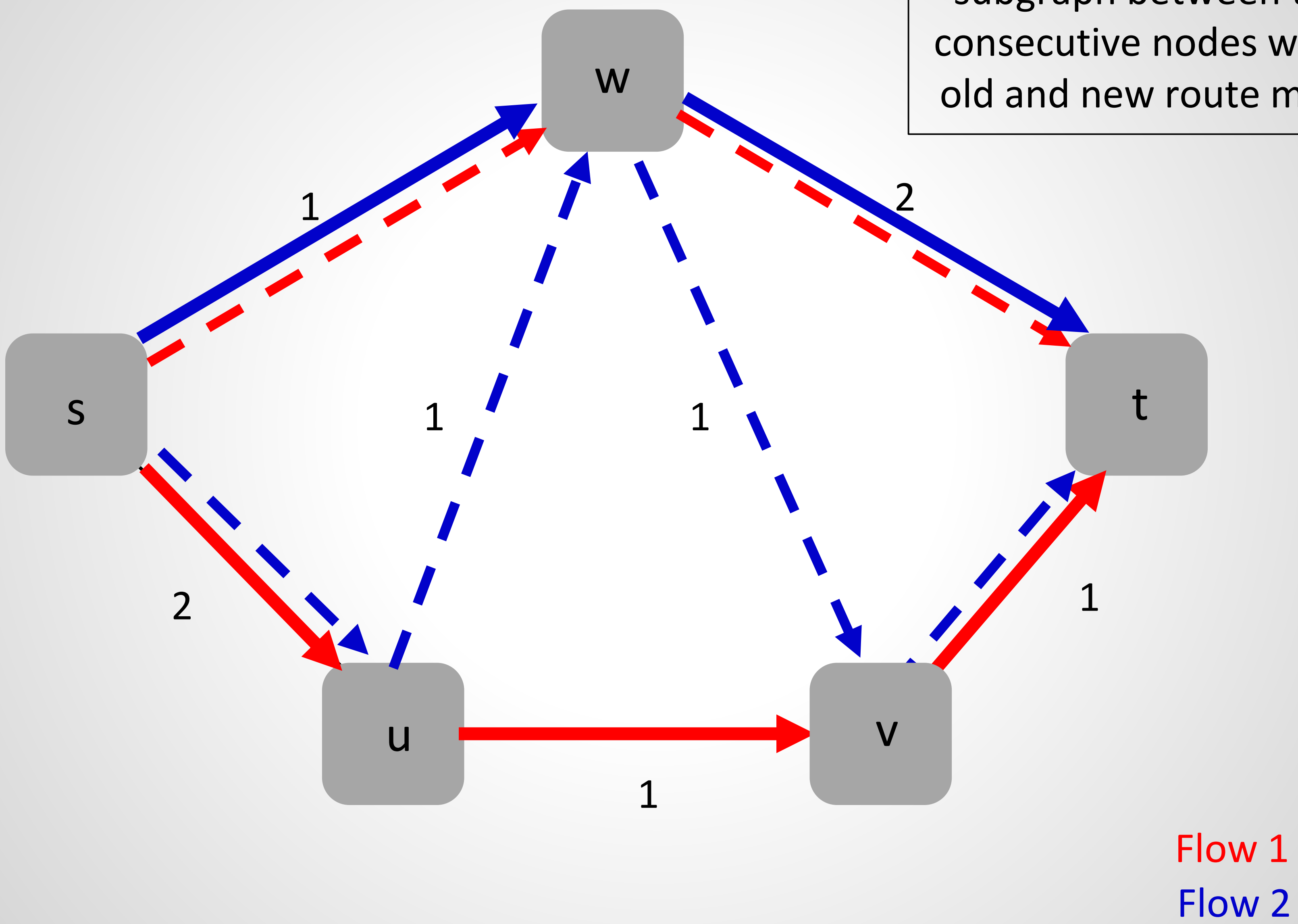
Schedule:

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

Round 4

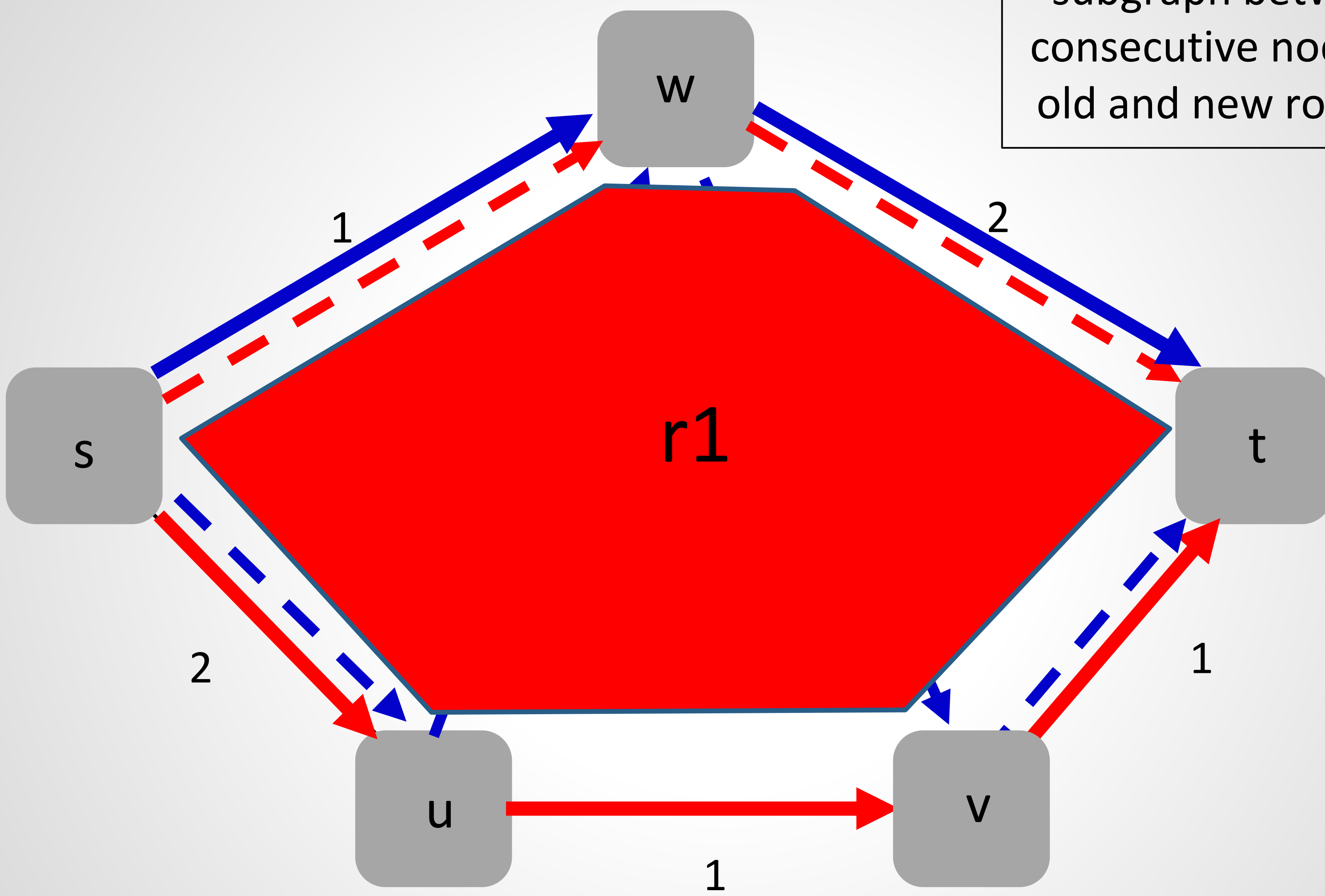
Block Decomposition of DAGs

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



Block Decomposition of DAGs

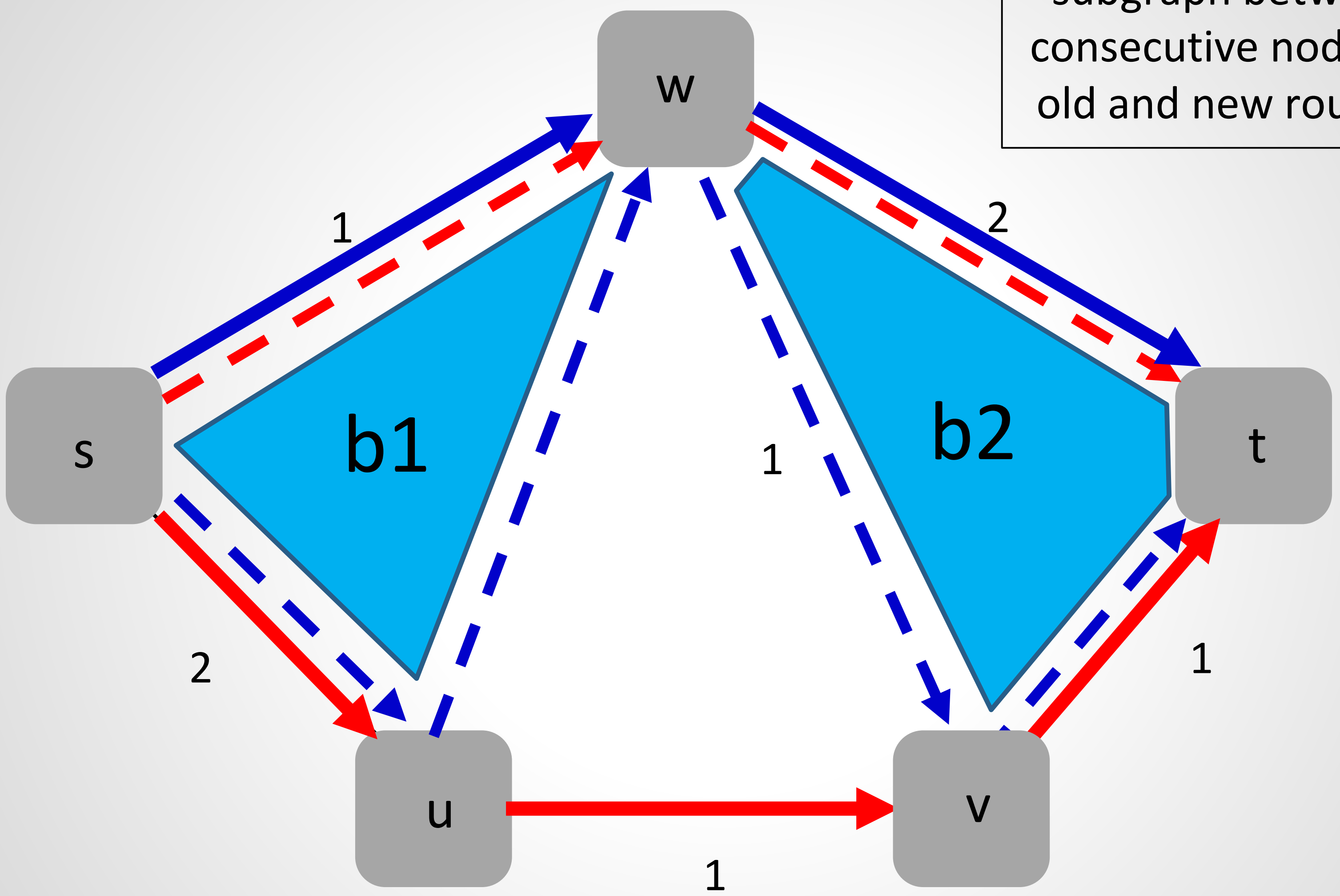
Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Just one red block: r1

Block Decomposition of DAGs

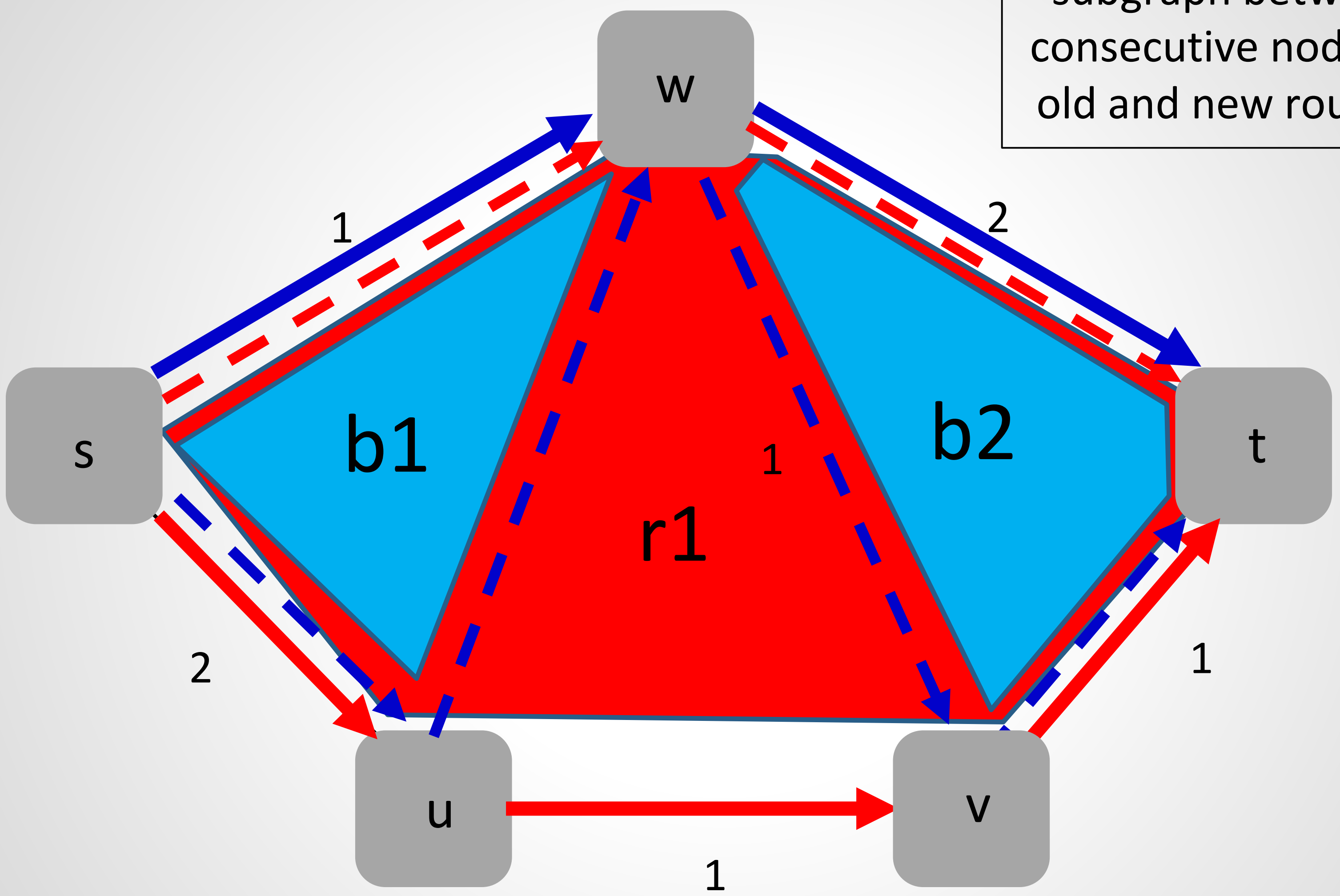
Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



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.

Congestion-Free Rerouting: Summary of Results

- Congestion-free rerouting: a fundamental problem, but not much known!

Often hard:

- NP-hard already for 2 flows in general flow networks
- NP-hard already on DAGs for general k flows

But not always:

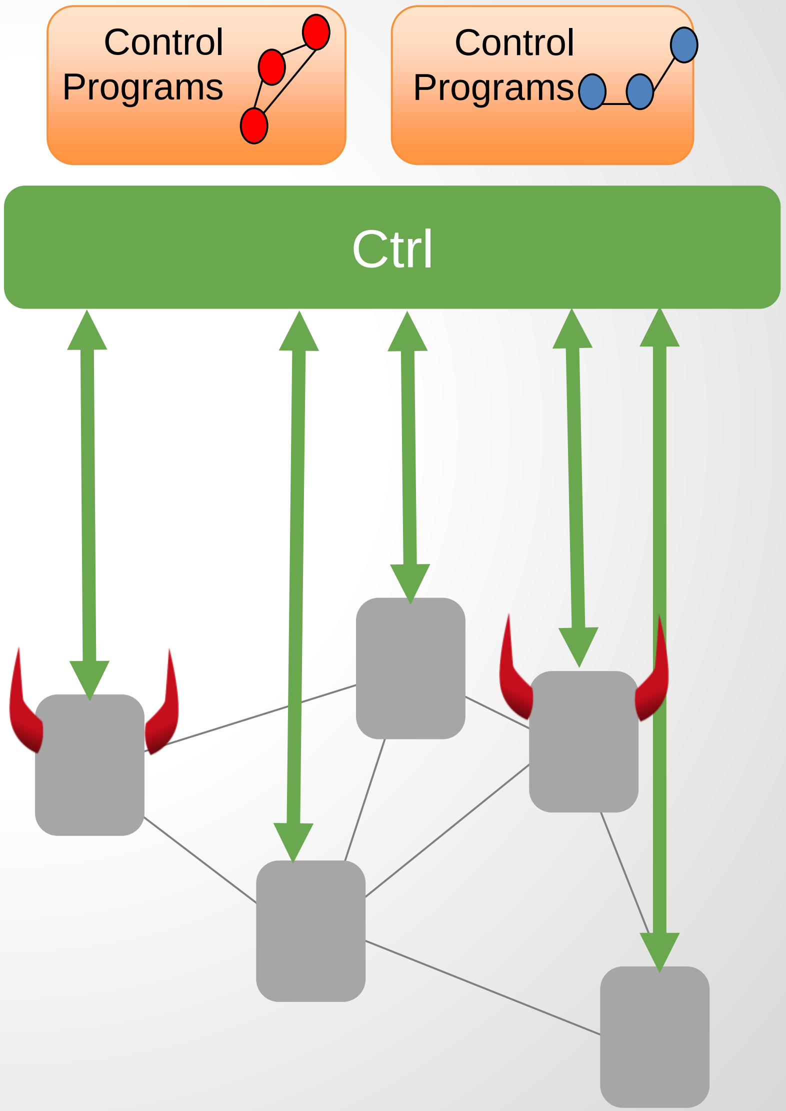
- For k=2 flows, poly-time algorithm on DAGs exists
  - Algorithm based on block decomposition of flow graph = dependency graph
  - Optimal number of rounds
- For k=const flows, poly-time algorithm on DAGs exists
  - Weaker notion of dependency graph
  - Feasibility (but not optimality?) in time $2^{O(k \log k)} O(n)$, k = # flows

From Consistency to Security

- Software-defined networks and network virtualization also introduce new security challenges

- In general, much research on control plane security (e.g., secure BGP)

Our recent research: the insecure data plane, a new threat vector!



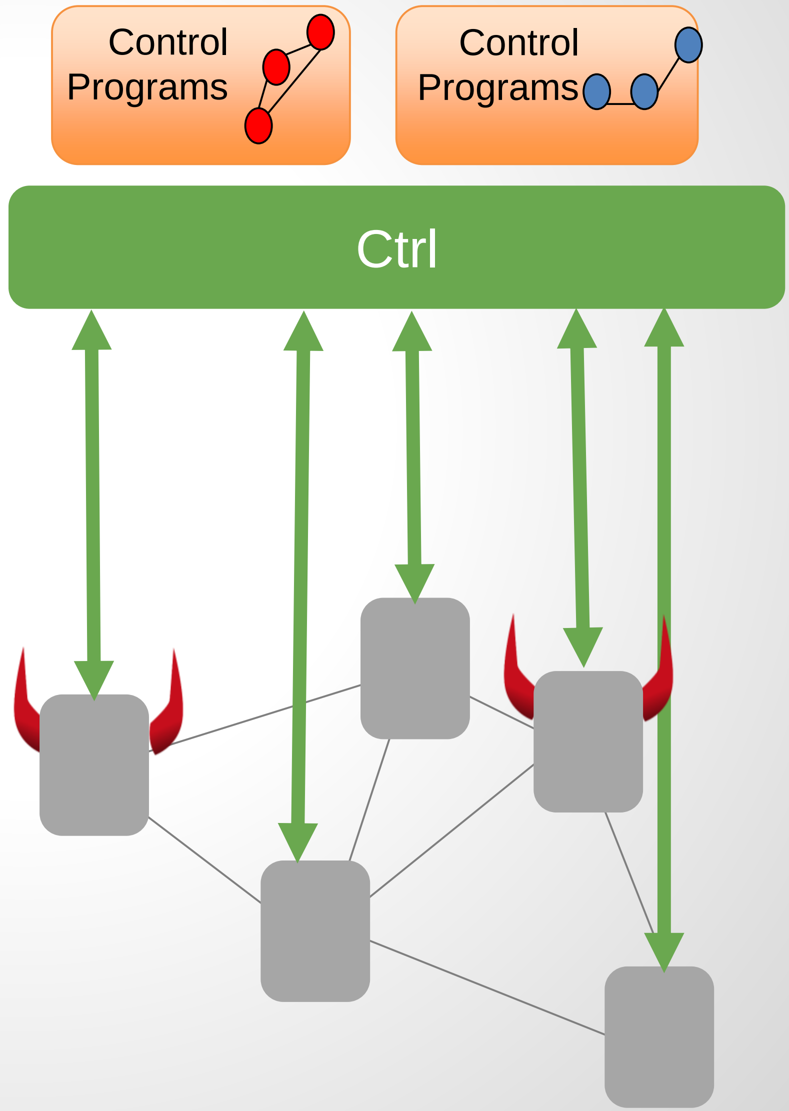
The Insecure Data Plane

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

A tough problem: how to build a secure computer network if you don't trust the hardware??

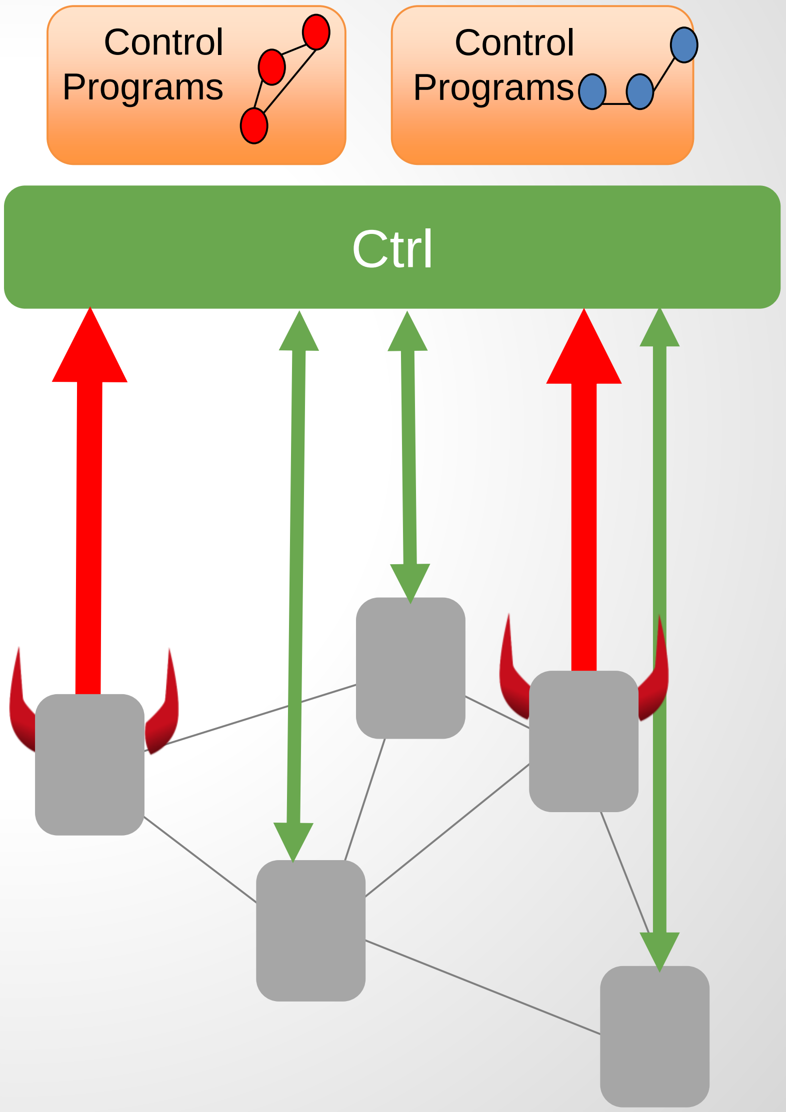
(Building your own is expensive!)



The Insecure Data Plane

Attack vector 1:

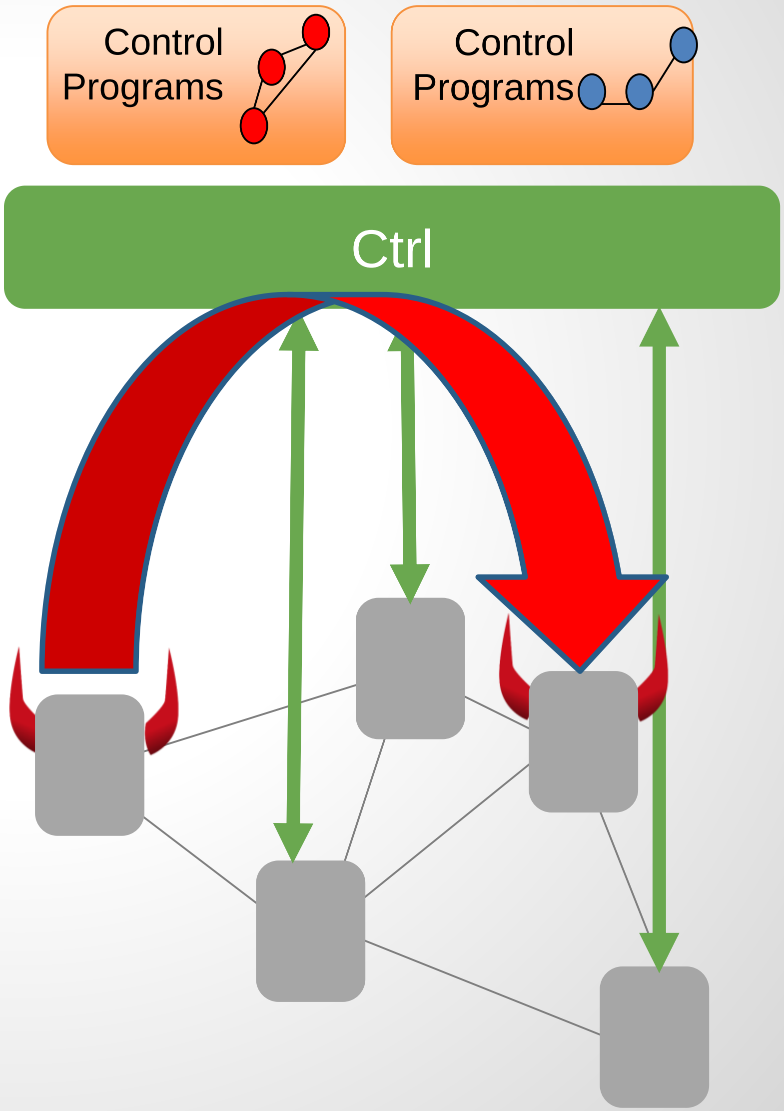
- ❑ DoS on controller
- ❑ Harms availability
- ❑ E.g., force other switches into default behavior



The Insecure Data Plane

Attack vector 2:

- «Teleportation» or covert communication
- Controller reacts to switch events (packet-ins) by sending flowmods/packet-outs/… etc.: can be exploited to transmit information (e.g., src MAC 0xBADDAD)
- Can also modulate information implicitly (e.g., frequency of packetins)



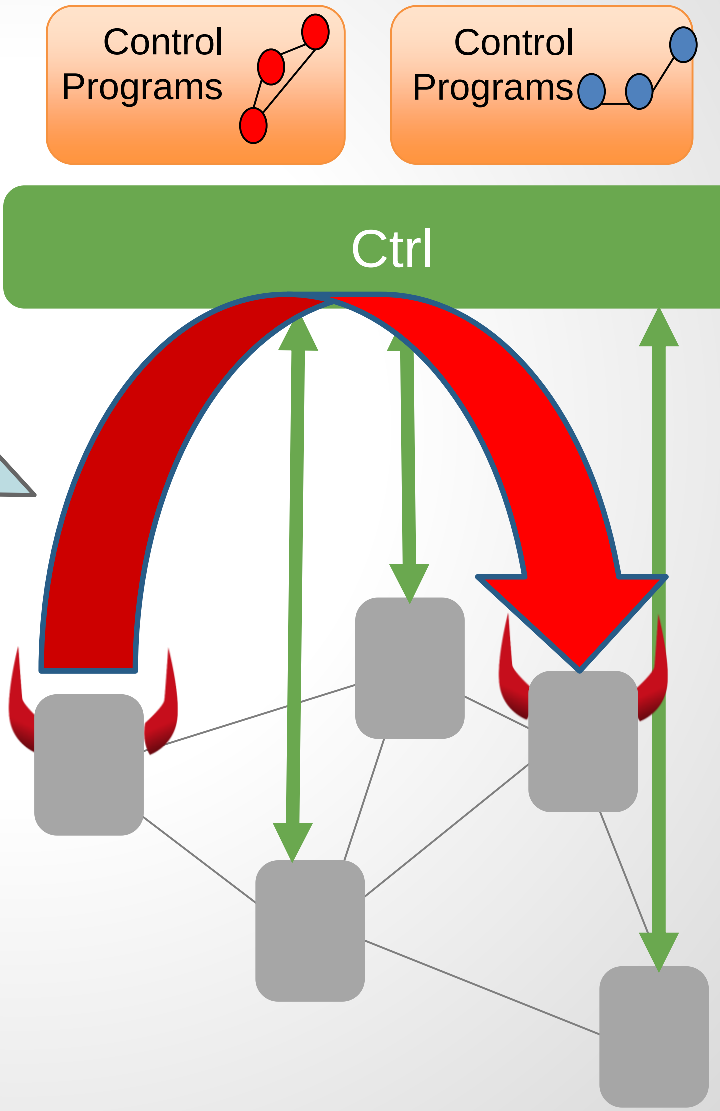
The Insecure Data Plane

Attack vector 2:

Hard to detect by security middleboxes in the data plane! Also hard to detect as OpenFlow channel is encrypted.

exploited to transmit information (e.g., src MAC 0xBADDAD)

- Can also modulate information implicitly (e.g., frequency of packetins)



The Insecure Data Plane

Attack vector 3:

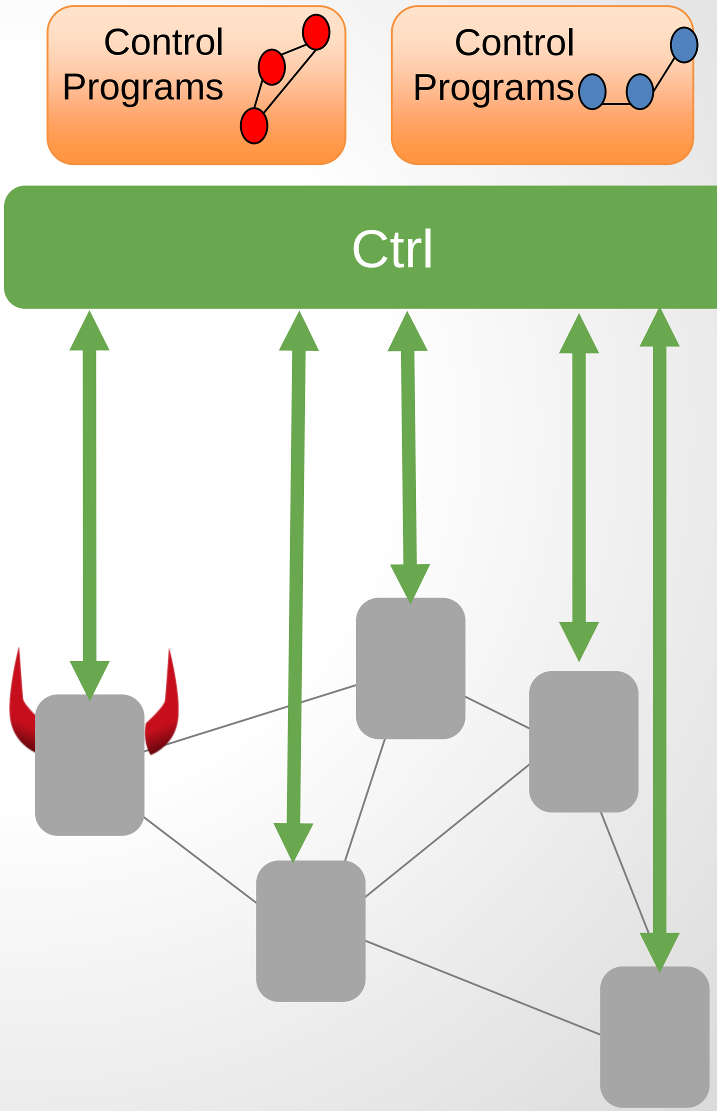
- ❑ The virtualized data plane

Background:

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

Advantage:

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



The Insecure Data Plane

Attack vector 3:

- The virtualized data plane

Background:

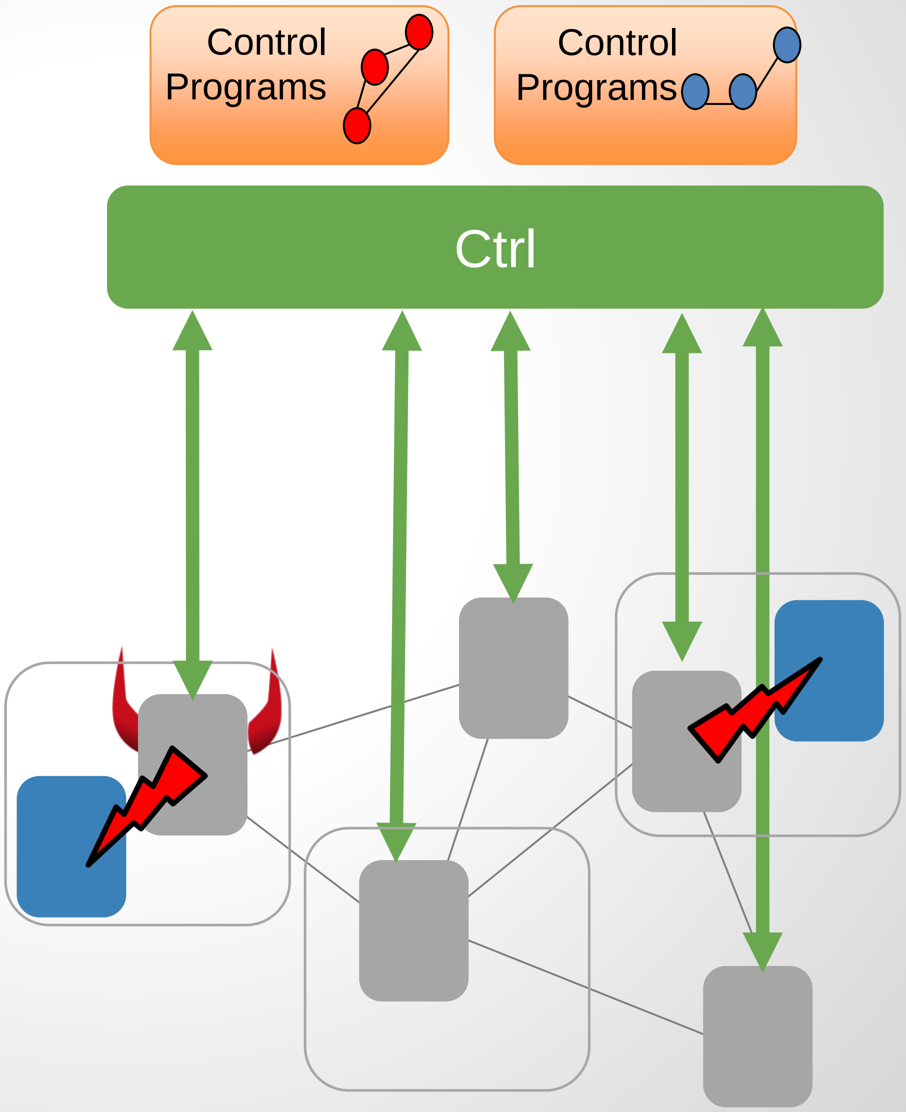
- Packet processing and other network functions are more and more virtualized
- E.g., they run on servers at the edge of the datacenter
- Example: OVS

Advantage:

  - Cheap and performance ok!
  - Fast and easy deployment

Disadvantage:

- New vulnerabilities, e.g., collocation!



The Insecure Data Plane

Attack vector 3:

- The virtualized data plane

Background:

- Packet processing and other network functions are more and more virtualized

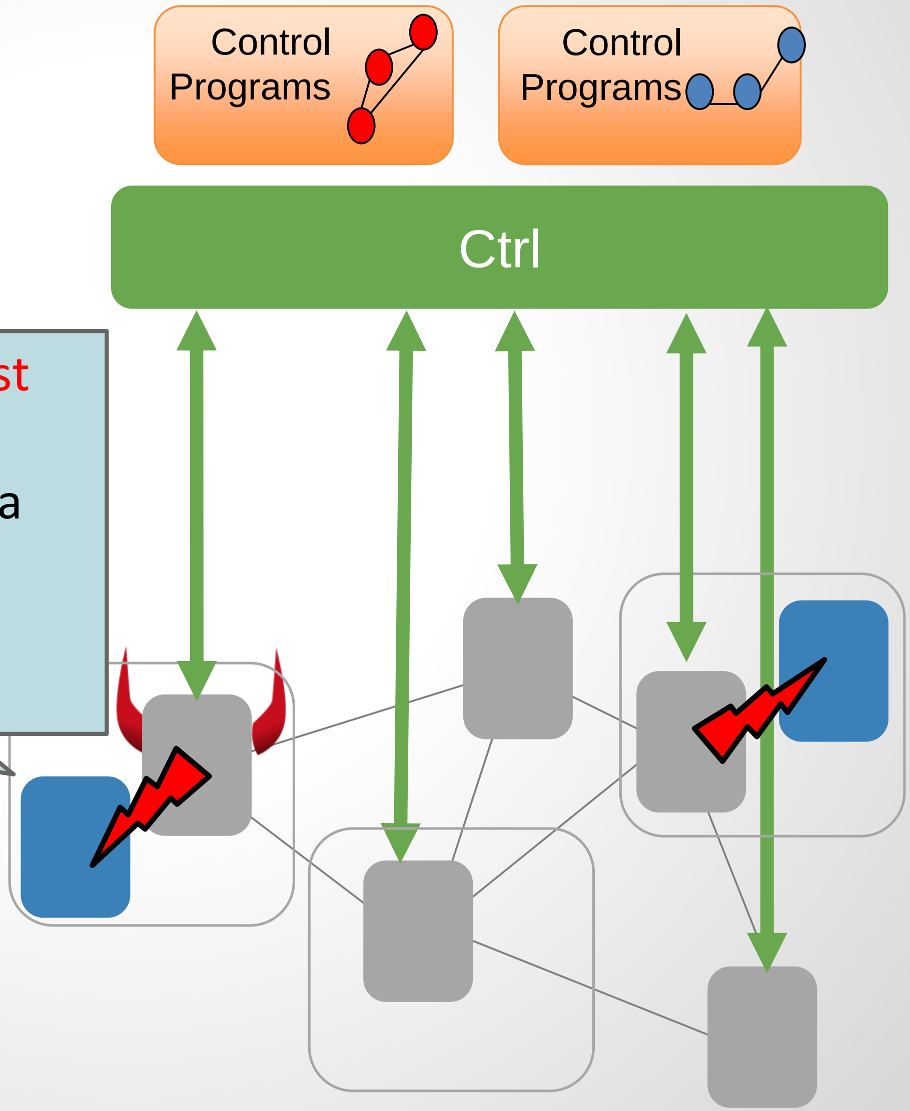
e.g., controllers, hypervisors, guest VMs, image management (the images VMs use for boot-up), data storage, network management, identity management (of the adminstrators and tenants), etc.

A

- Cheap and performa…
- Fast and easy deployment

Disadvantage:

- New vulnerabilities, e.g., collocation!



A Case Study: OVS

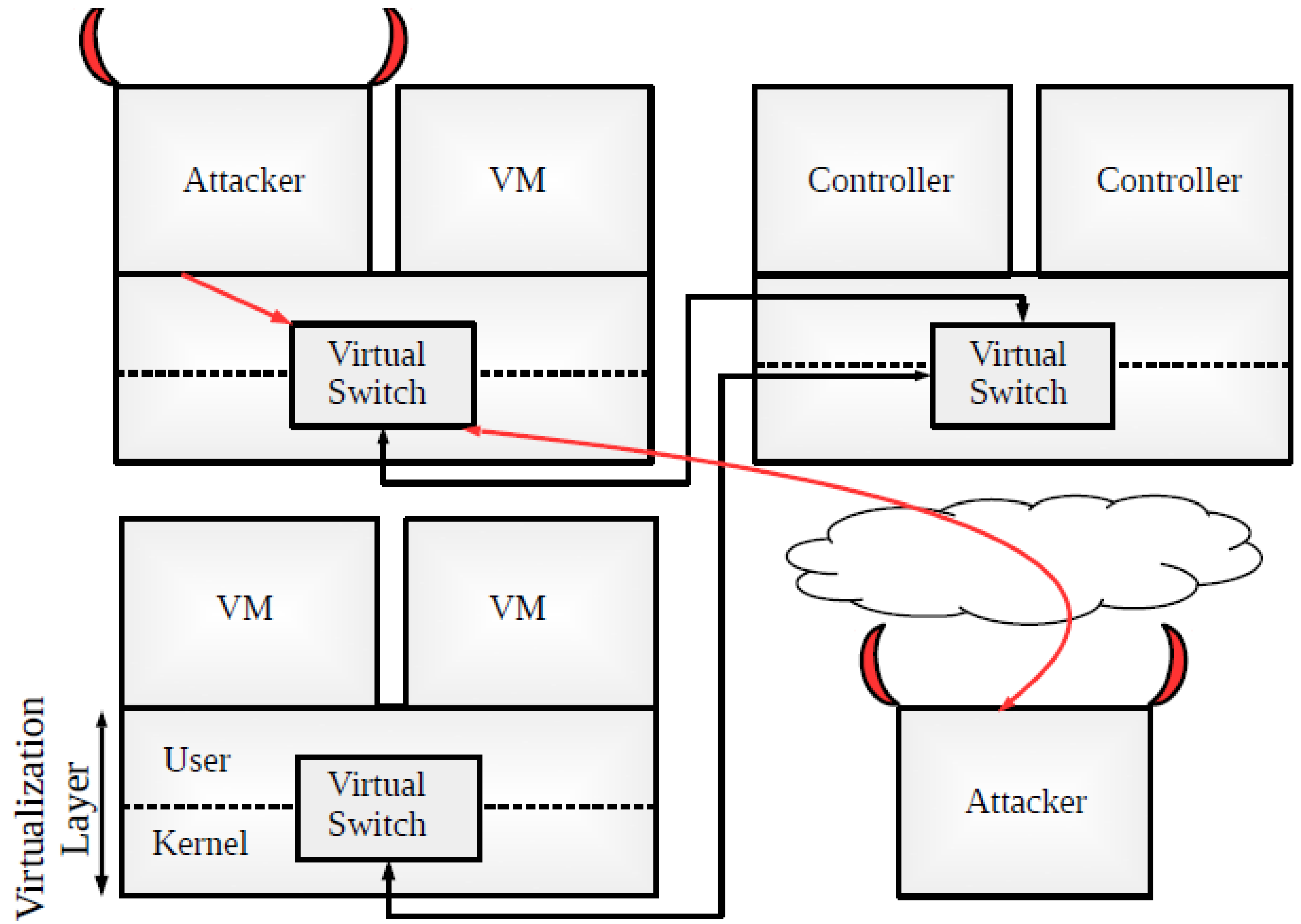
- OVS: a production quality switch, widely deployed in the Cloud
- After fuzzing just 2% of the code, we found major vulnerabilities:
  - E.g., two stack overflows when malformed MPLS packets are parsed
- These vulnerabilities can easily be weaponized:
  - Can be exploited for arbitrary remote code execution
  - 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)

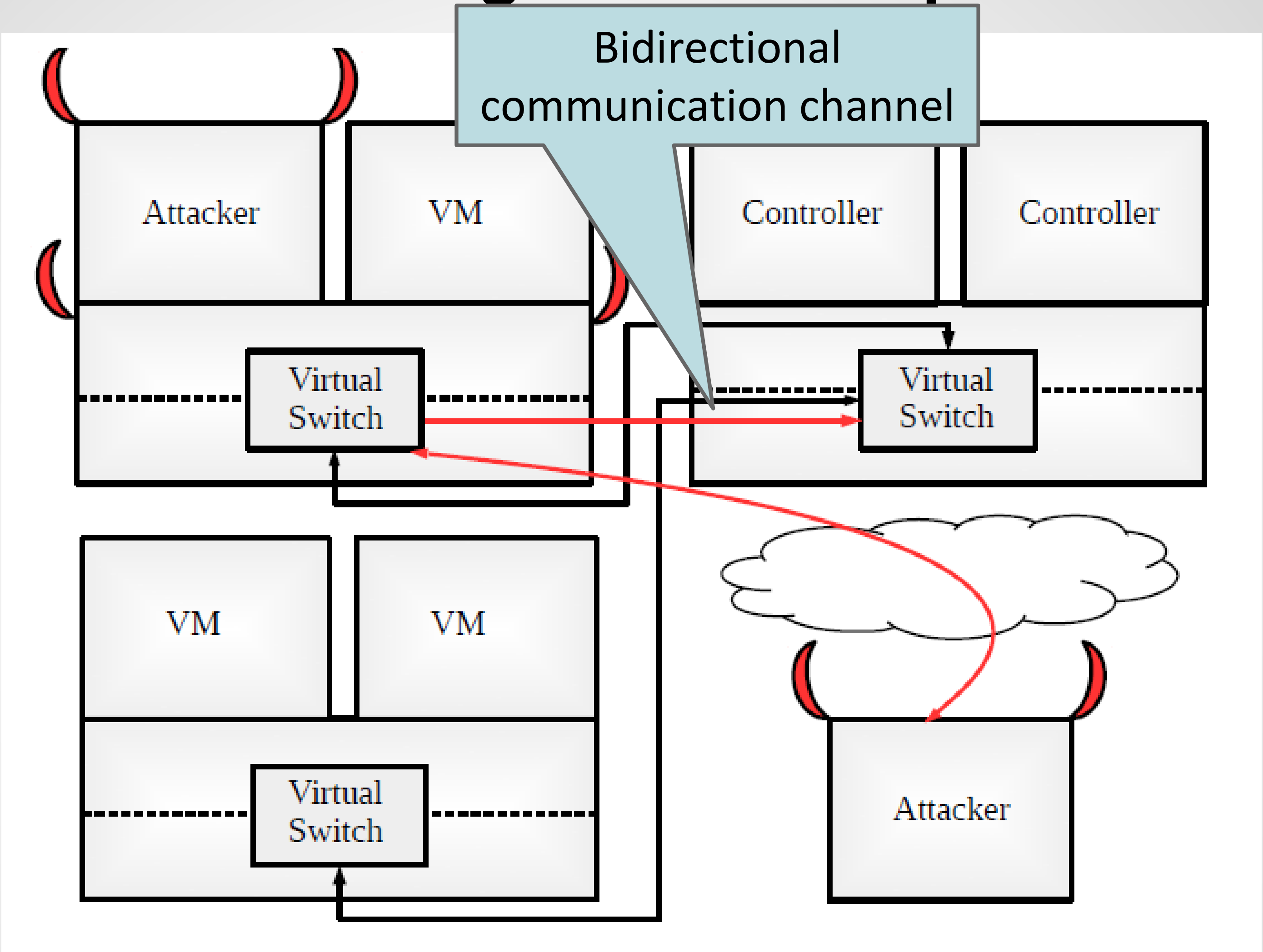
The Reign Worm: Step 1



(a) Step 1.

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

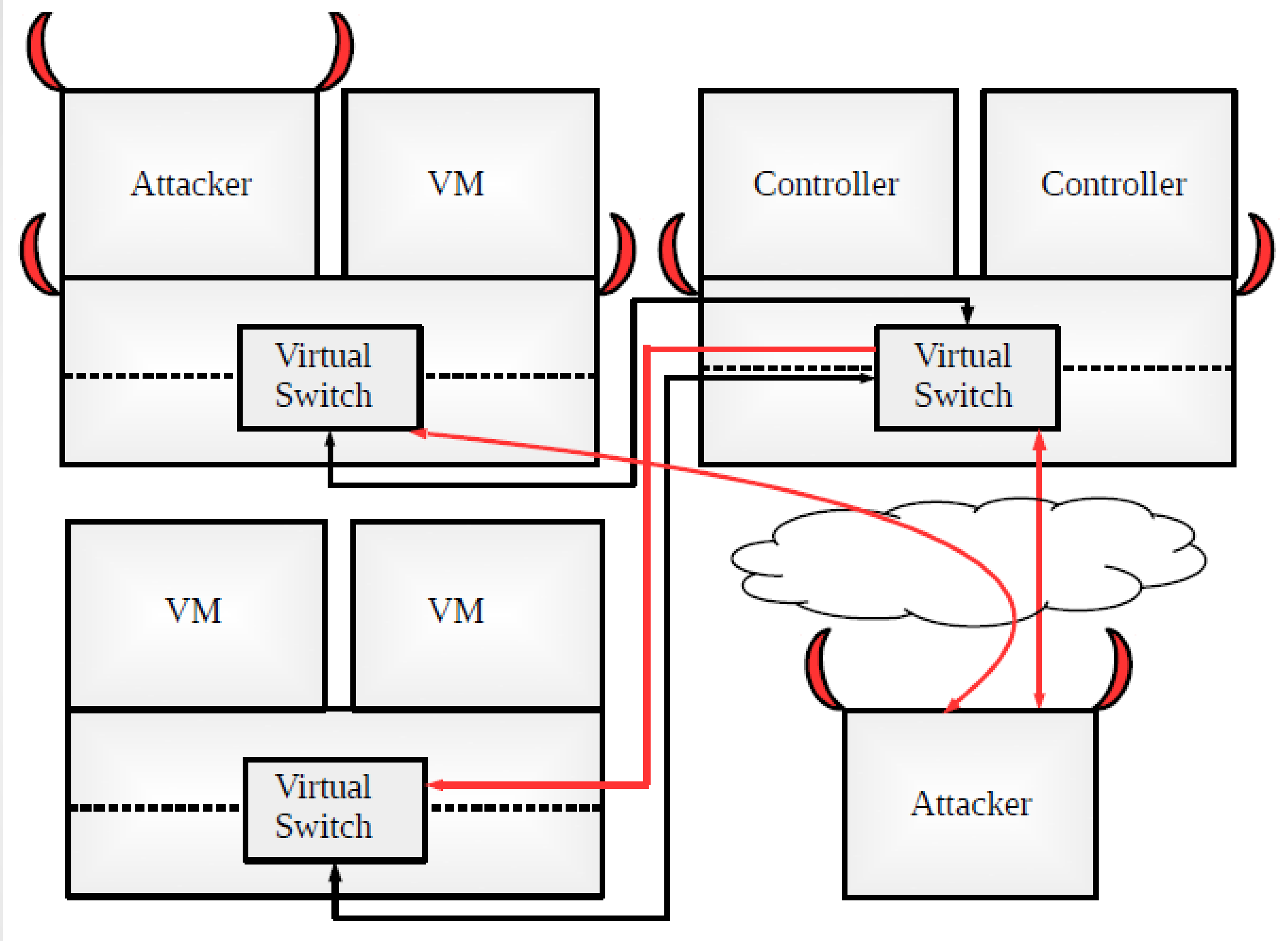
The Reign Worm: Step 2



(b) Step 2.

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

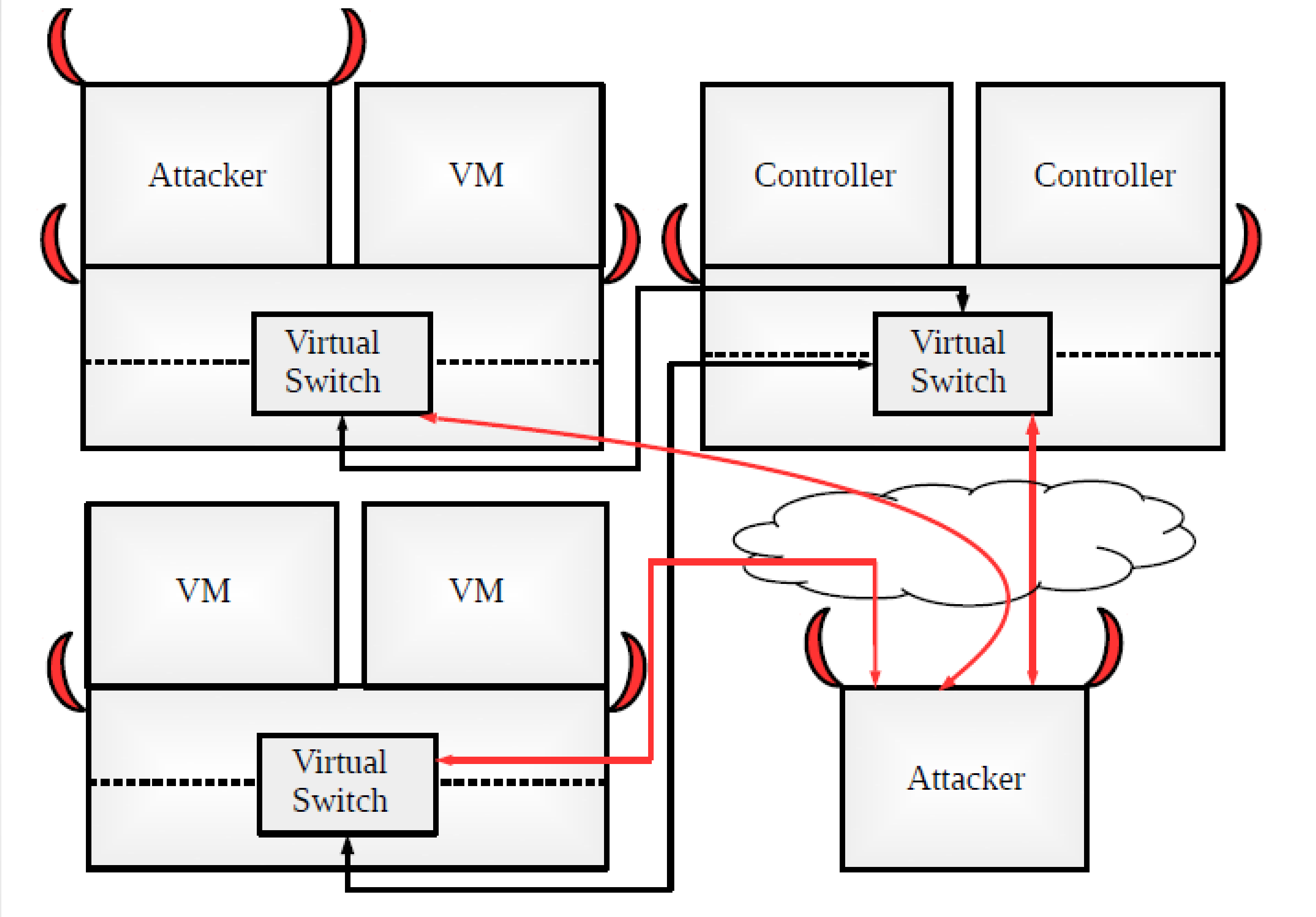
The Reign Worm: Step 3



(c) Step 3.

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

The Reign Worm: Step 4

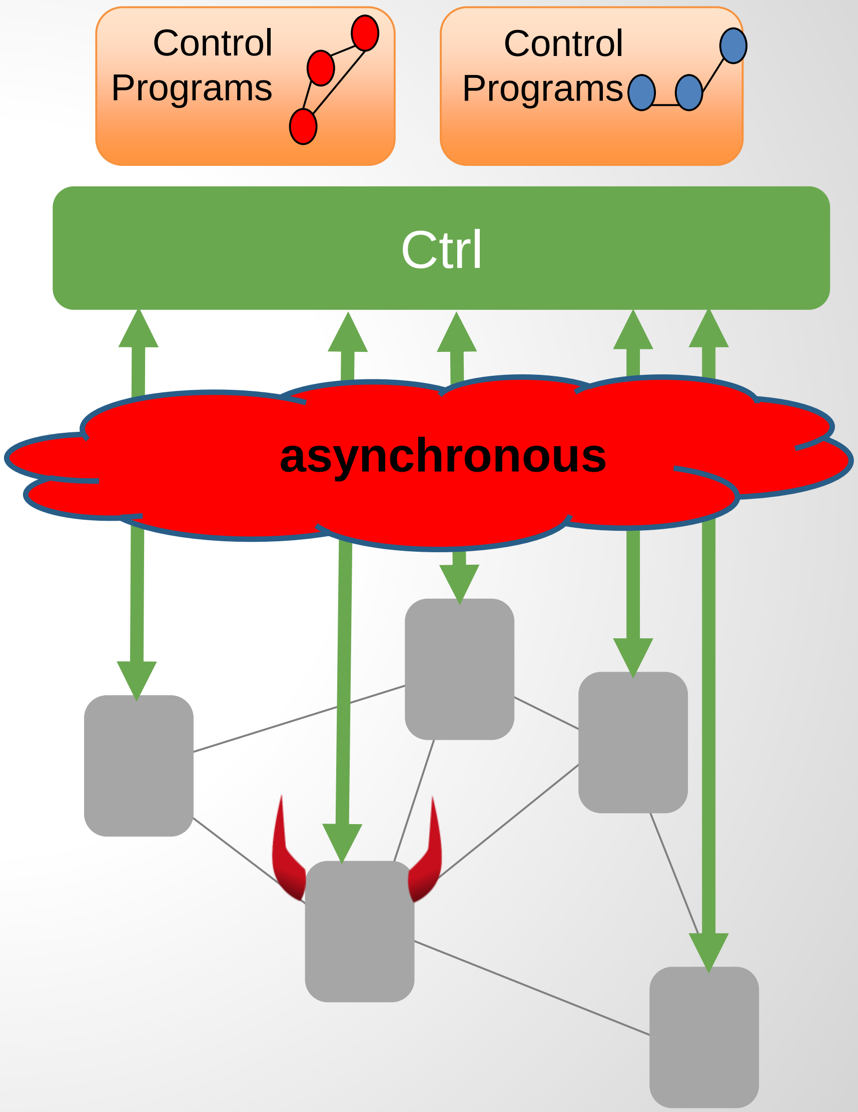


(d) Step 4.

All the servers are controlled by the remote attacker.

Conclusion

- SDN promises innovation and correct and verifiable networking, but also introduces new algorithmic challenges…
  - Example: route updates
- … as well as security challenges
  - Example: covert channels and vulnerable data plane



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.

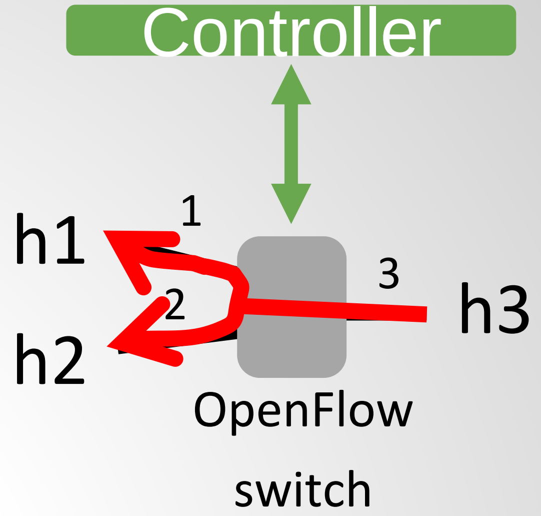
teleportation

attacking the cloud

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.

Backup Slides

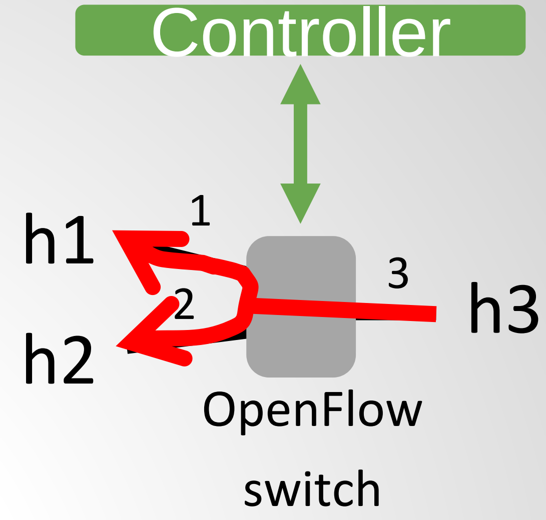
Jennifer Rexford's Example: SDN MAC Learning Done Wrong



- MAC learning: The «Hello World»
  - a bug in early controller versions
- In legacy networks simple
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets (source address!)
- Pitfalls in SDN: learn sender => miss response
  - Assume: low priority rule * (no match): send to controller
  - h1->h2: Add rule h1@port1 (location learned)
  - Controller misses h2->h1 (as h1 known, h2 stay unknown!)
  - When h3->h2: flooding forever (learns h3, never learns h2)

Thanks to Jen Rexford for example!

Jennifer Rexford's Example: SDN MAC Learning Done Wrong

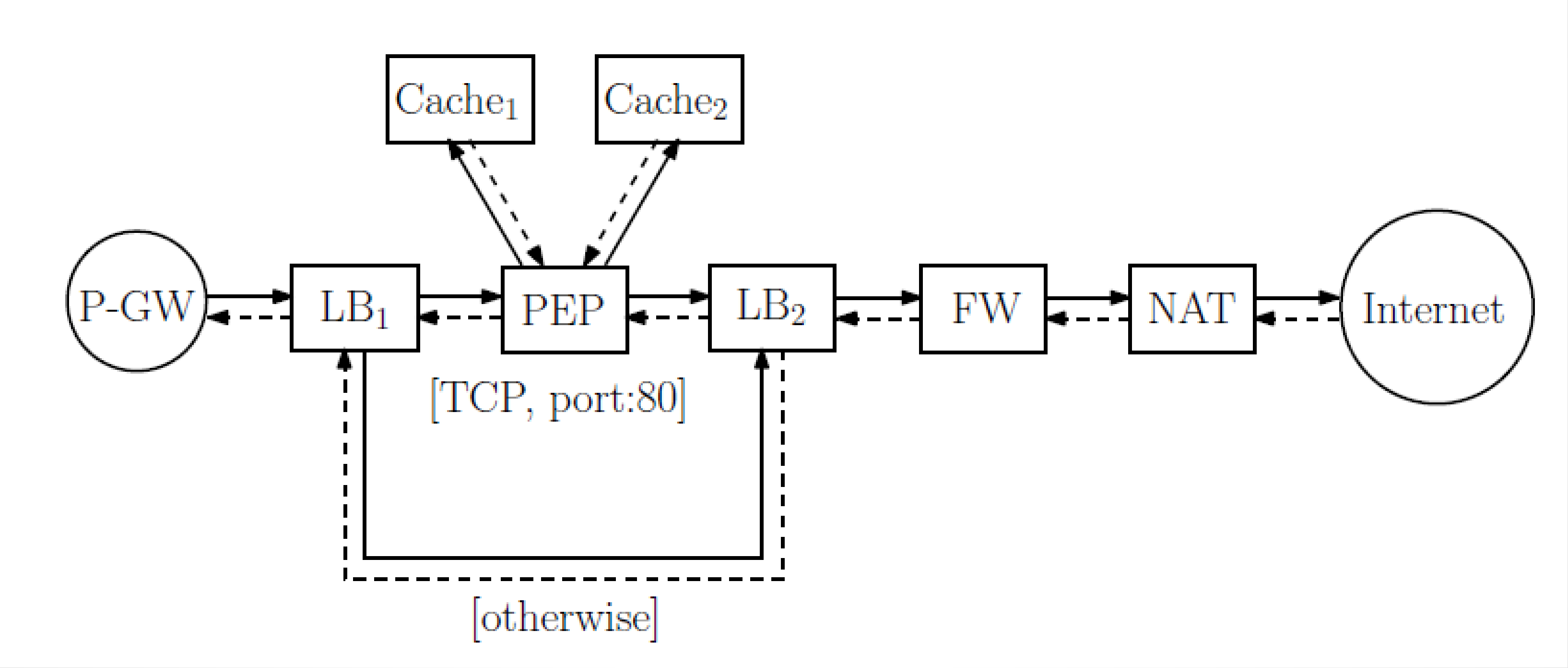


- MAC learning: The «Hello World»
  - a bug in early controller versions
- In legacy networks simple
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets (source address!)
- Pitfalls in SDN: learn sender => miss response
  - Assume: low priority rule *
  - h1->h2: Add rule h1@port1
  - Controller misses h2->
  - When h3->h2: flooding forever (learns h3, never learns h2)

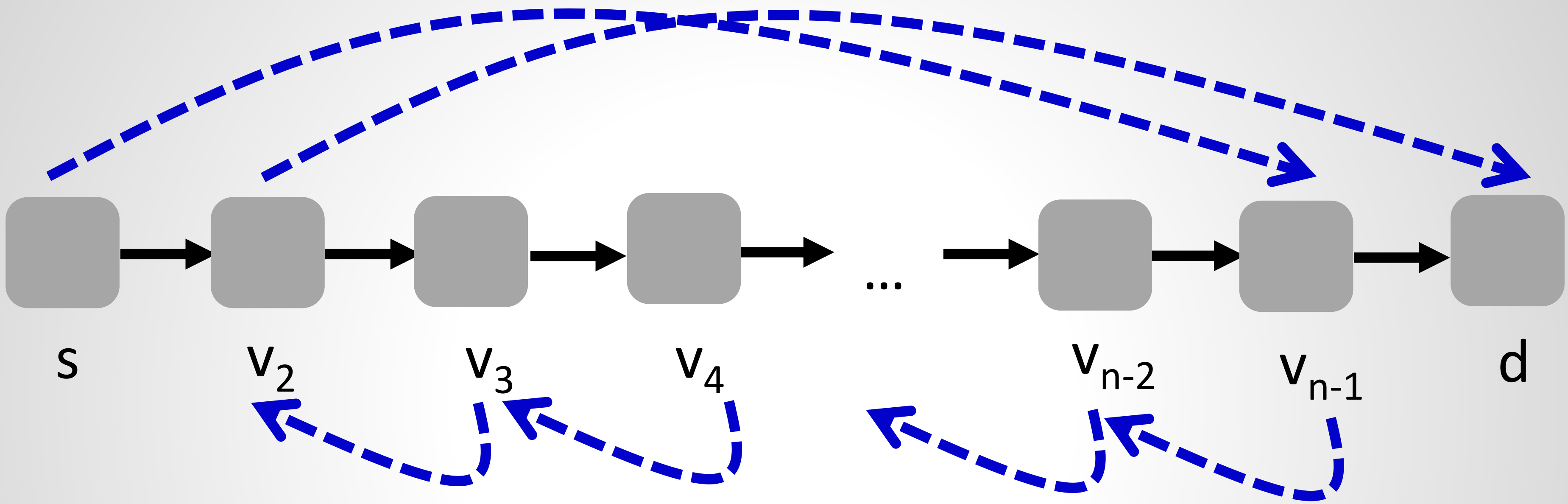
> Controller never sees source h2: switch already knows all destinations h1 and h3, so for h2 it keeps flooding.

Thanks to Jen Rexford for example!

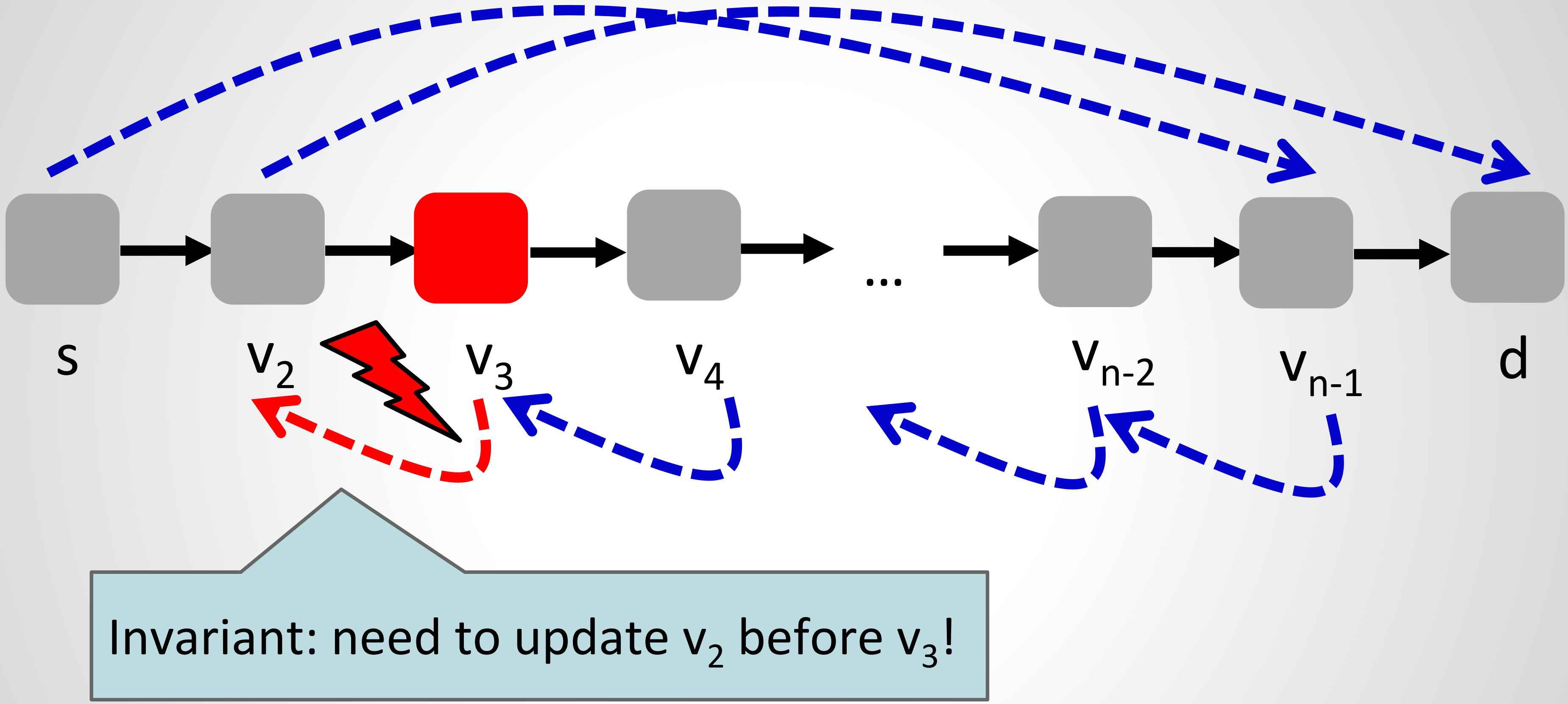
Complex Service Chains



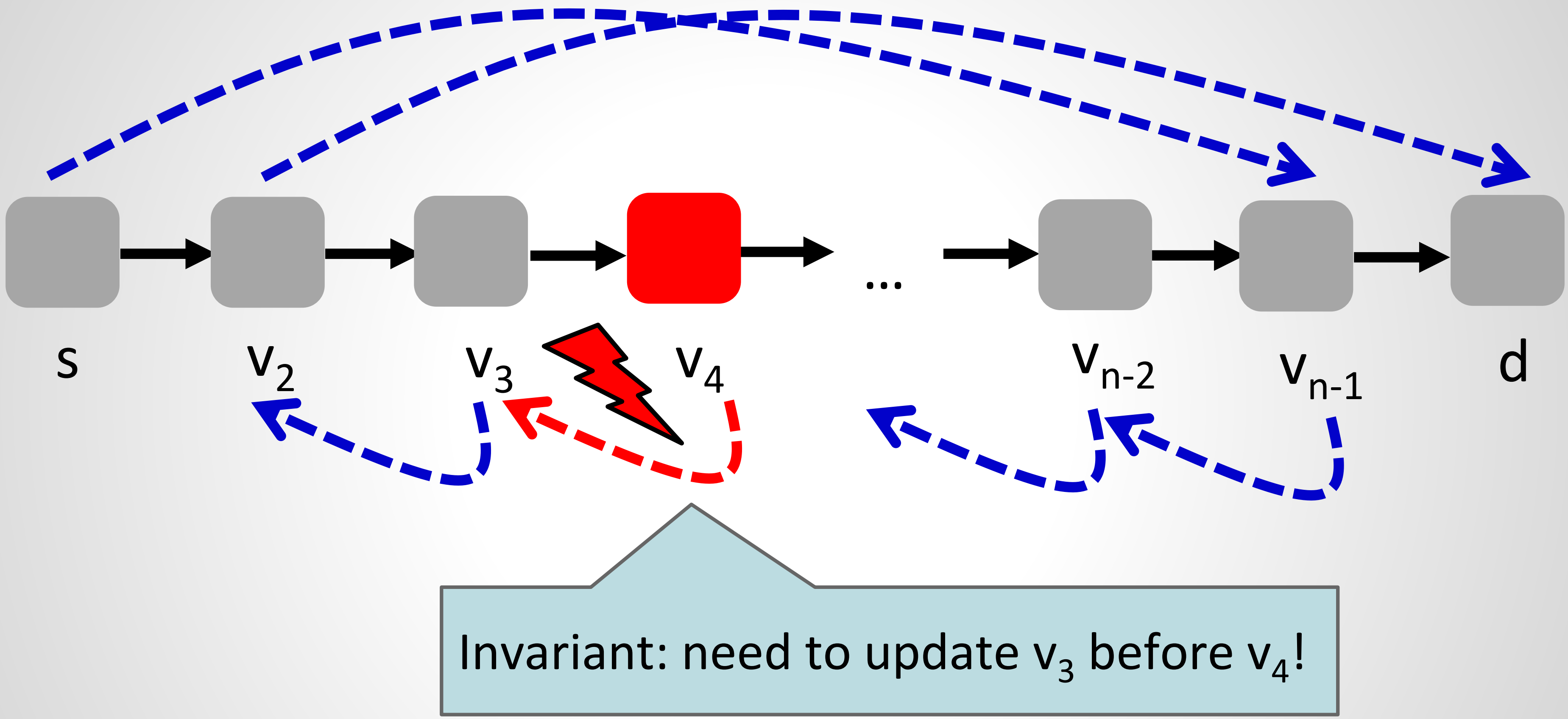
It‘s Good to Relax: How to update LF?



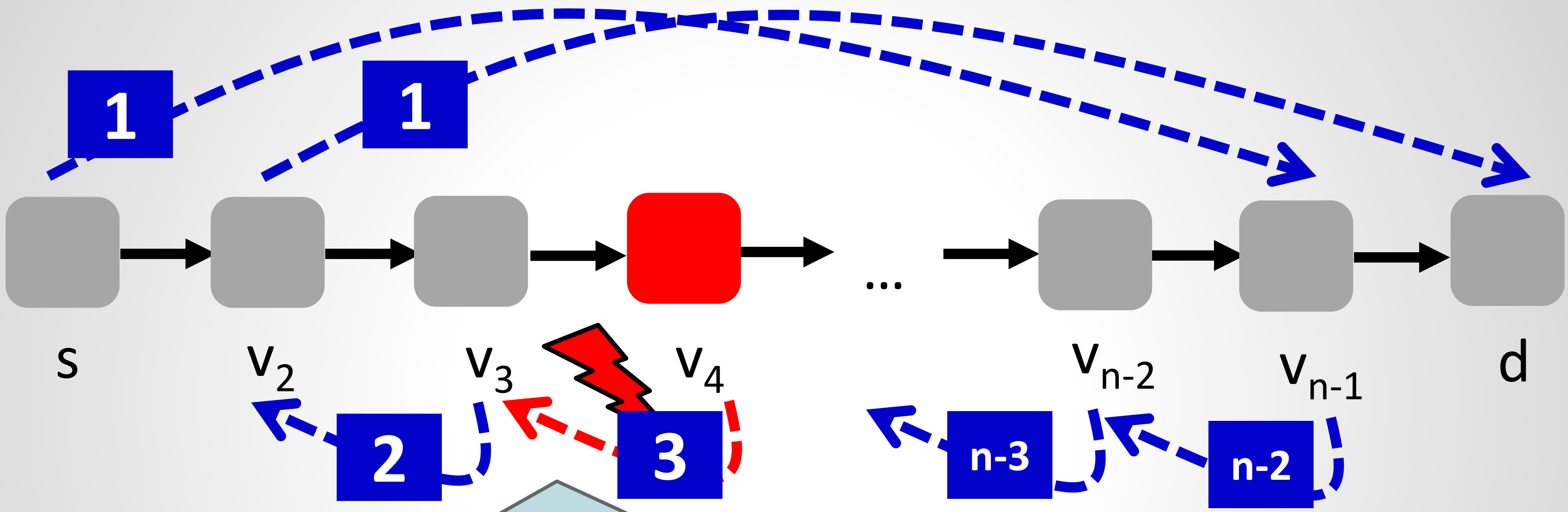
LF Updates Can Take Many Rounds!



LF Updates Can Take Many Rounds!



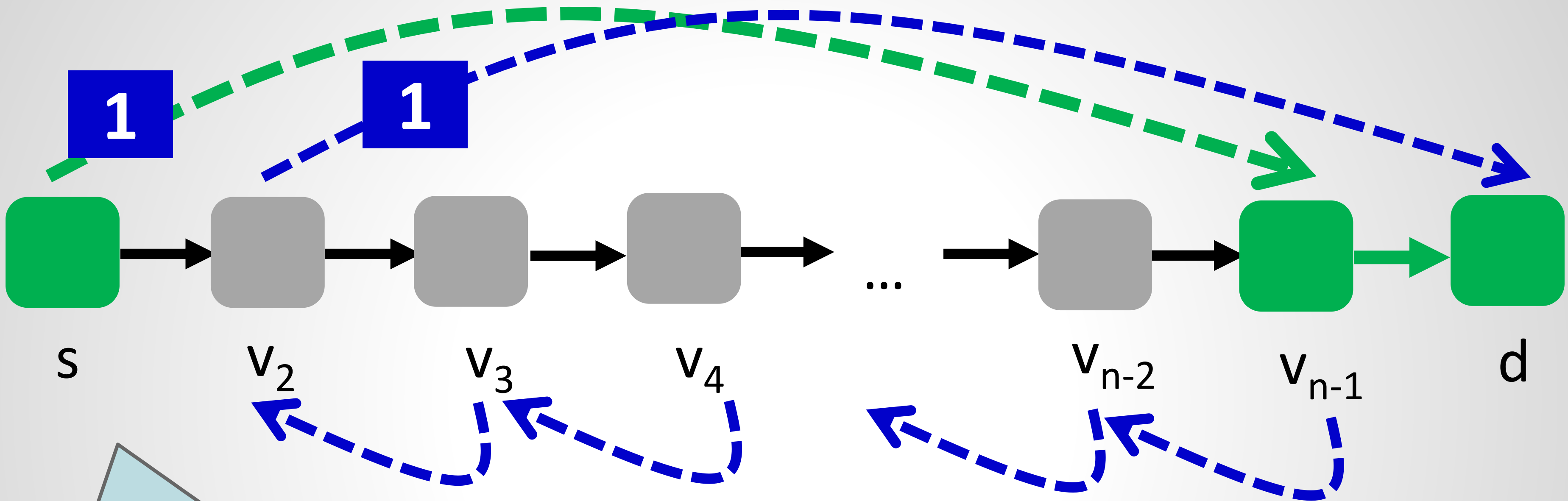
LF Updates Can Take Many Rounds!



Induction: need to update v_{i-1} before v_i (before v_{i+1} etc.)!

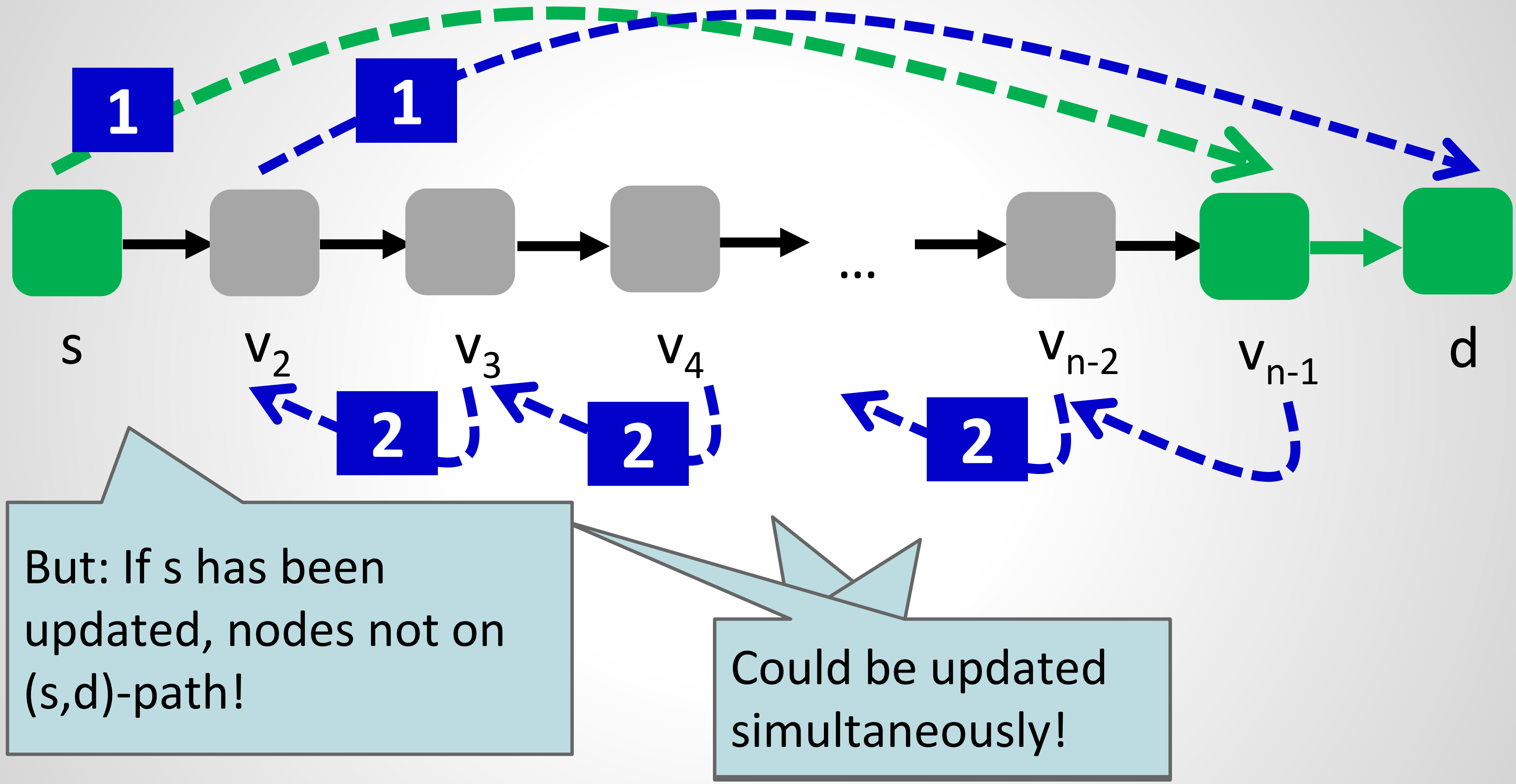
Ω(n) rounds?! In principle, yes…: Need a path back out before updating backward edge!

It is good to relax!

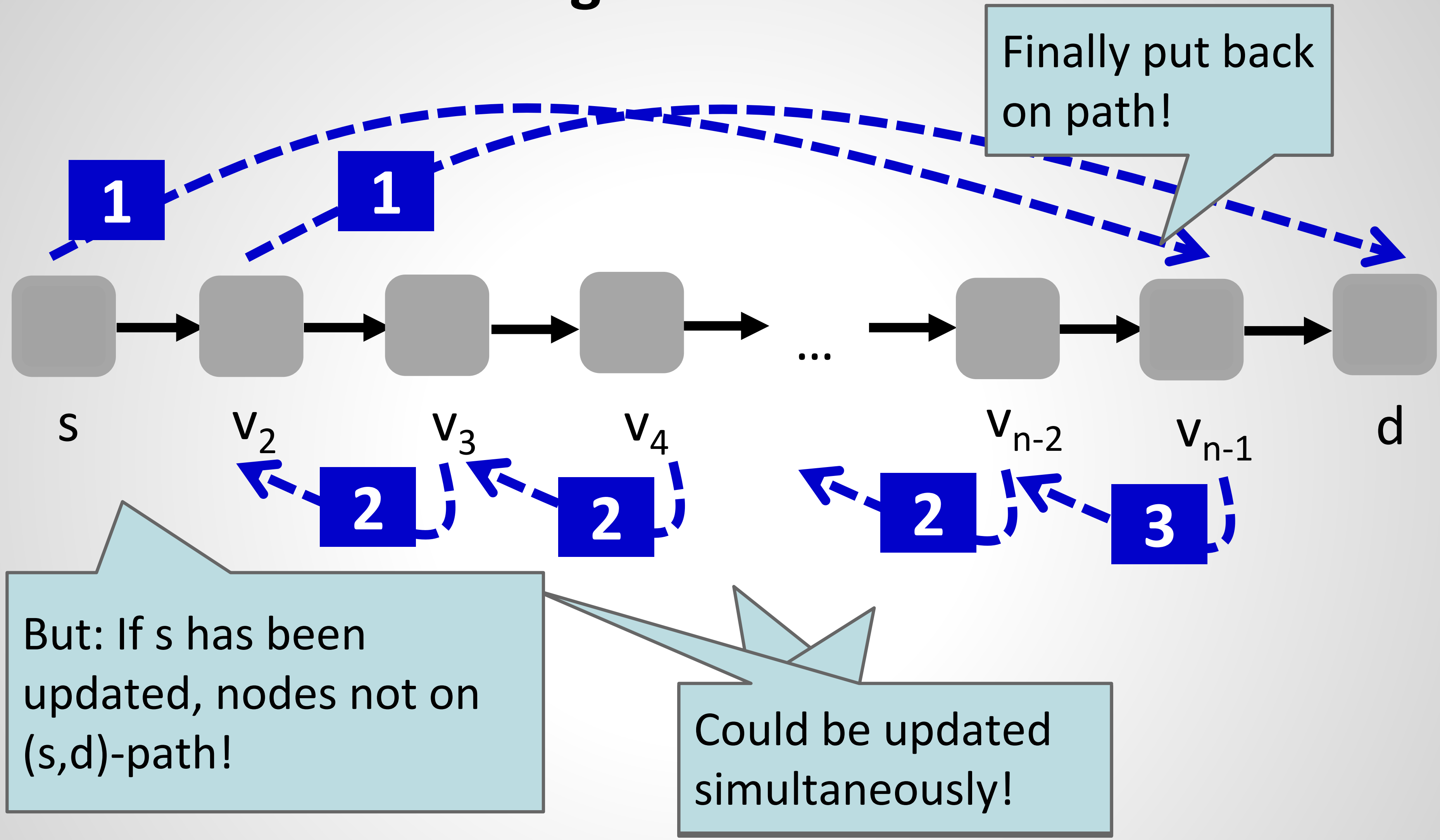


But: If s has been updated, nodes not on (s,d)-path!

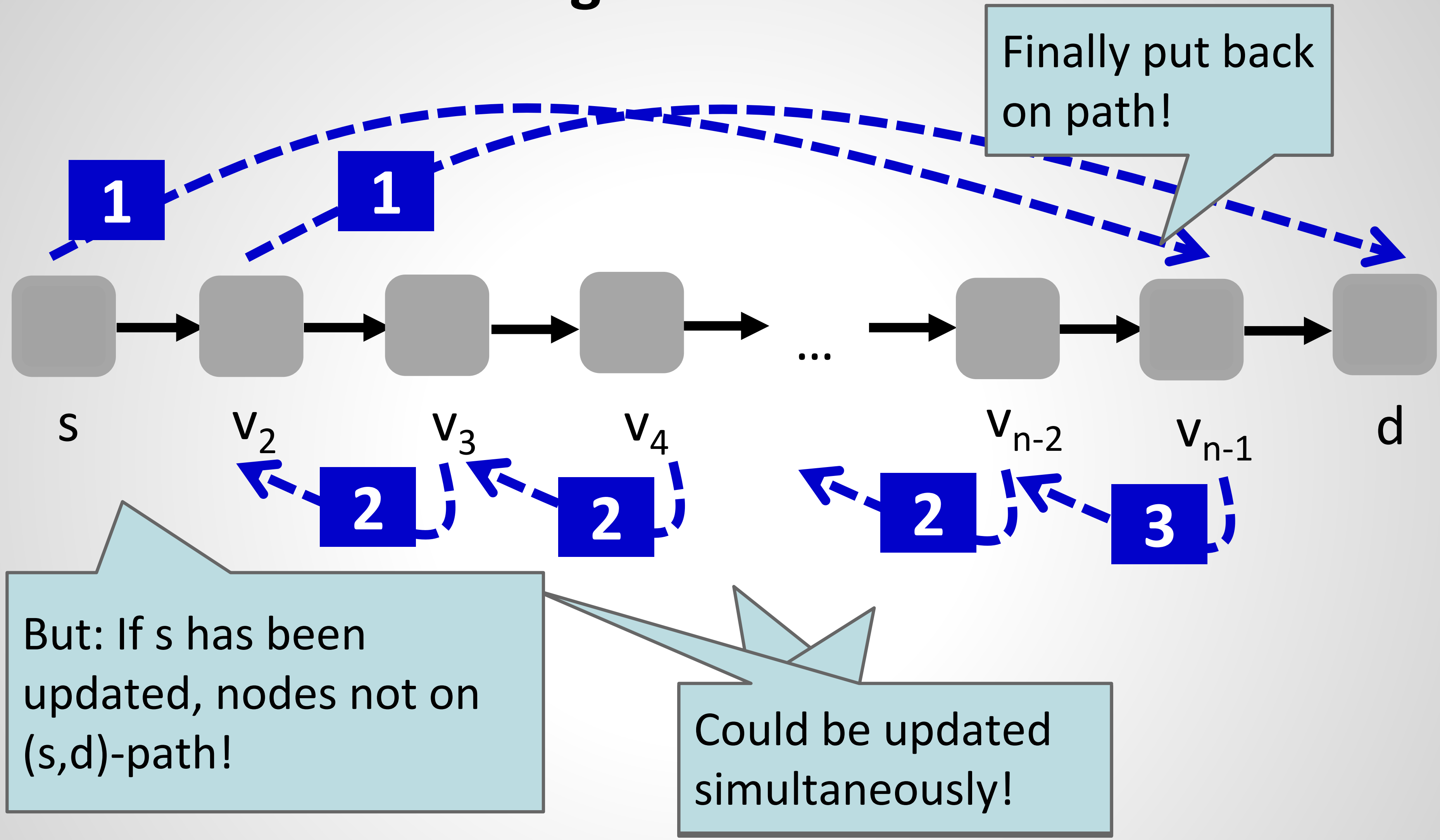
It is good to relax!



It is good to relax!

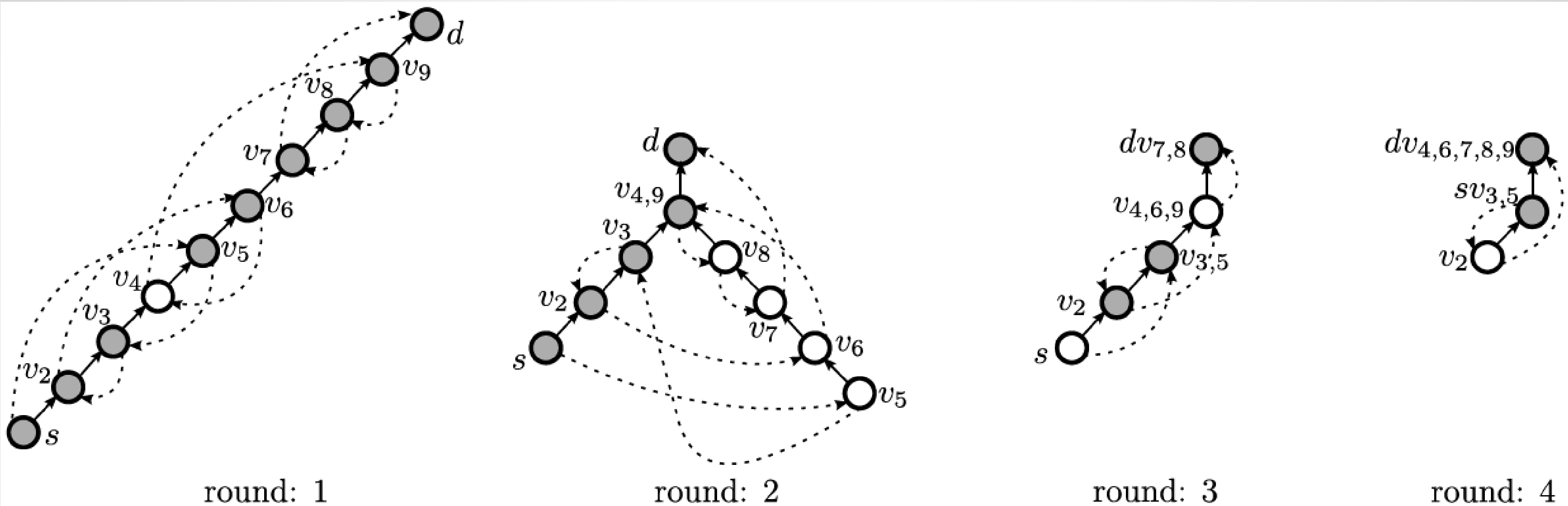


It is good to relax!



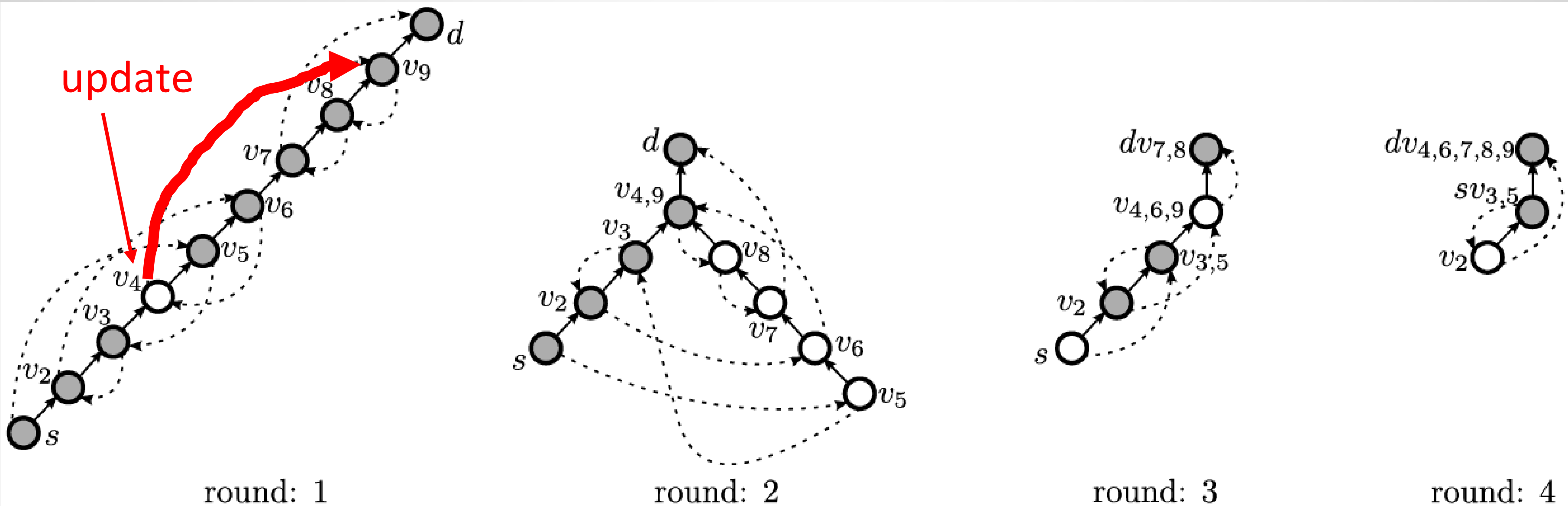
3 rounds only!

A log(n)-time Algorithm: *Peacock* in Action



Shortcut Prune Shortcut Prune

A log(n)-time Algorithm: *Peacock* in Action



Shortcut

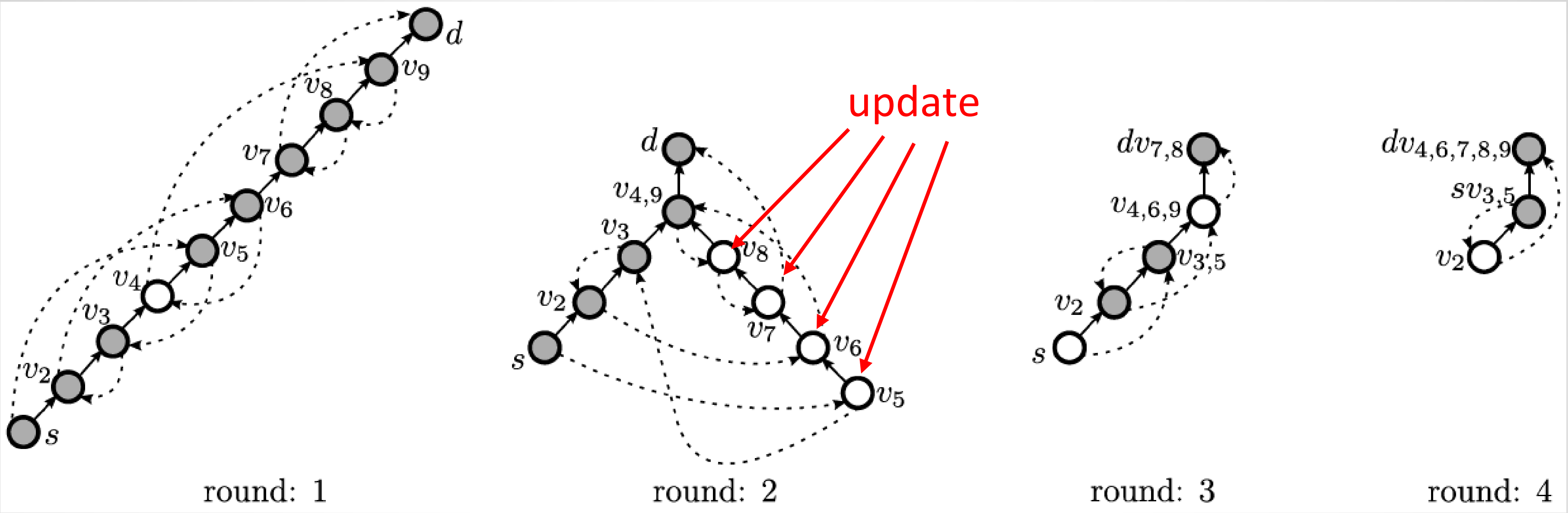
Prune

Shortcut

Prune

Greedily choose far-reaching (independent) forward edges.

A log(n)-time Algorithm: *Peacock* in Action



Shortcut

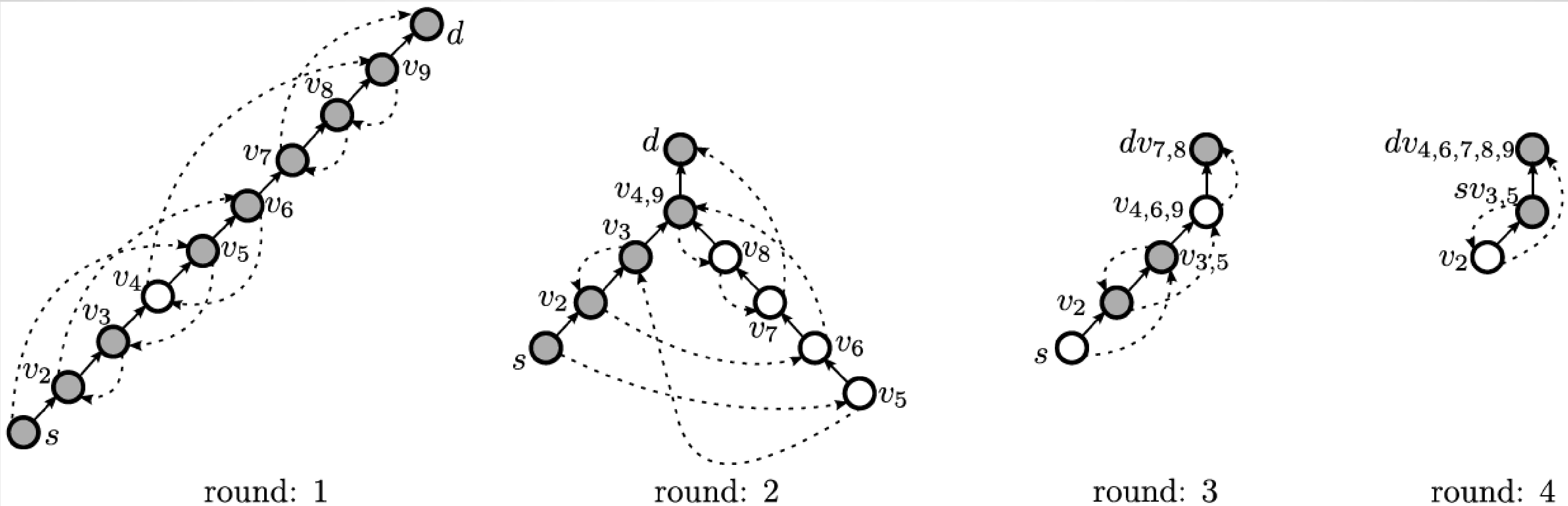
Prune

Shortcut

Prune

R1 generated many nodes in branches which can be updated simultaneously!

A log(n)-time Algorithm: *Peacock* in Action



Shortcut

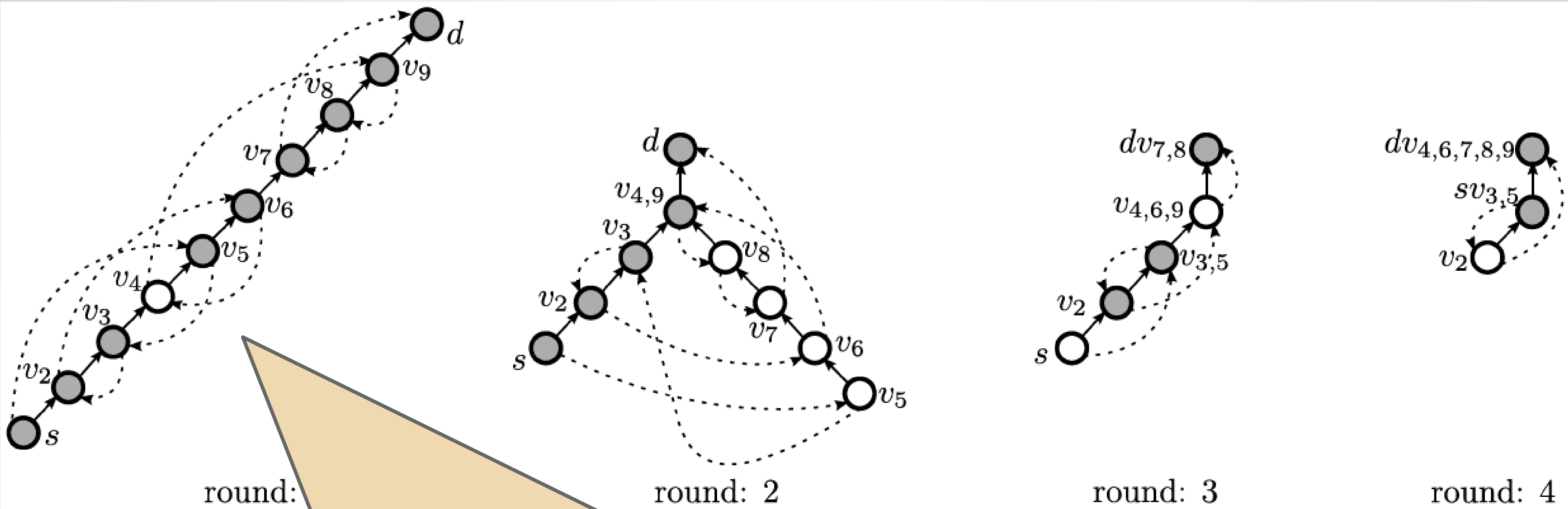
Prune

Shortcut

Prune

Line re-established!
(all merged with a node on the s-d-path)

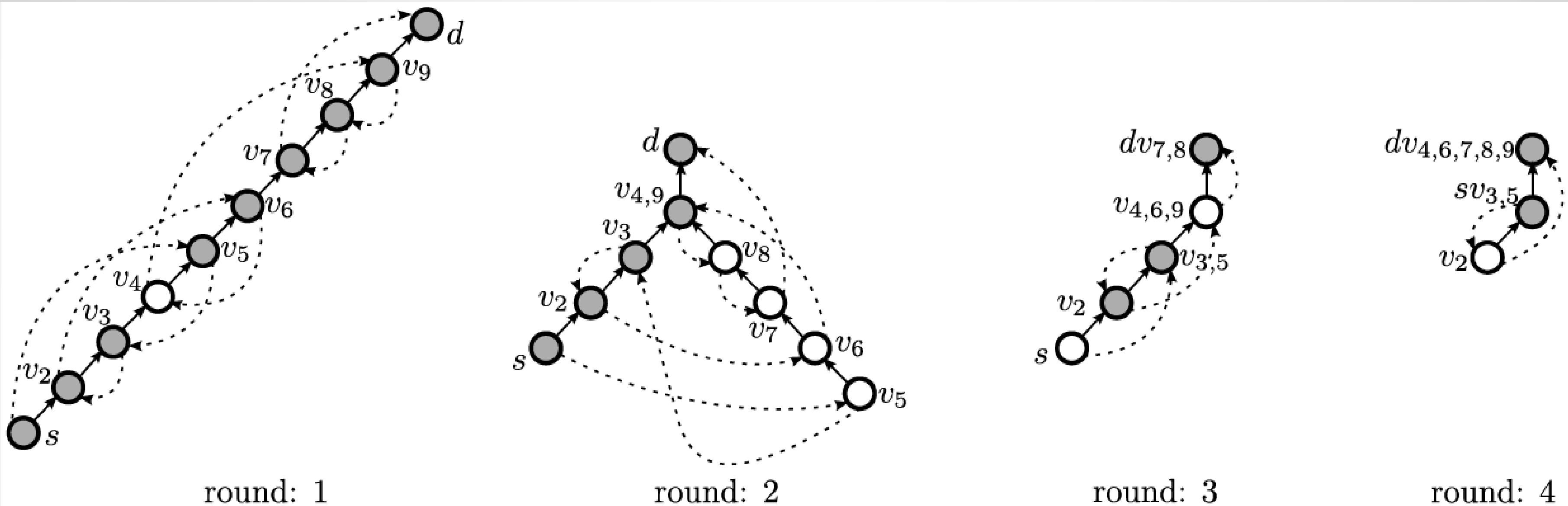
A log(n)-time Algorithm: *Peacock* in Action



Peacock orders nodes wrt to distance: edge of length x **can block** at most 2 edges of length x, so distance 2x.

Prune

A log(n)-time Algorithm: *Peacock* in Action



Shortcut

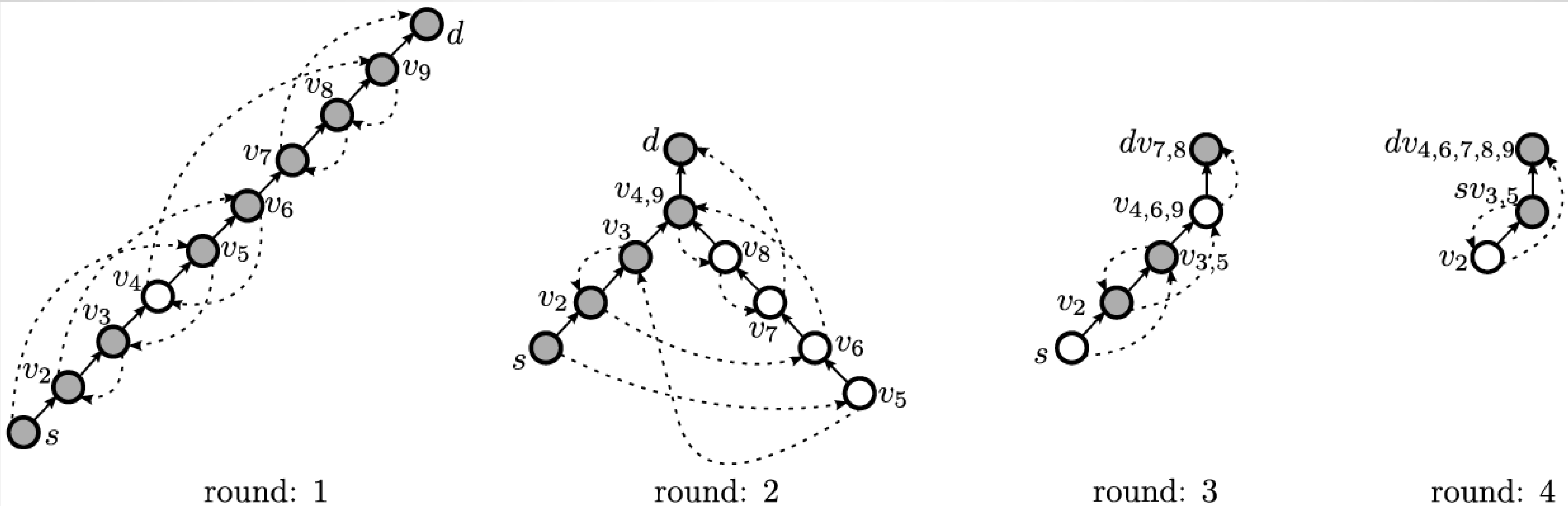
Prune

Shortcut

Prune

At least 1/3 of nodes merged in each round pair (shorter s-d path): logarithmic runtime!

A log(n)-time Algorithm: *Peacock* in Action



Shortcut

Prune

Shortcut

Prune